

ویدیوی رایگانی در مورد پیاده سازی WCF Services با استفاده از اصول REST ، ویژگی‌های WebGet و WebInvoke, کار با کلاس‌های SyndicationFeed و Rss20FeedFormatter و هاست آن در IIS .

[دریافت ویدیو](#)

[دریافت سورس کد مربوطه](#)

[ماخذ](#)

عنوان: ویدیوهای رایگان آموزشی WCF از مایکروسافت
نویسنده: وحید نصیری
تاریخ: ۱۷:۲۷:۰۰ ۱۳۸۸/۰۲/۲۹
آدرس: www.dotnettips.info
برچسب‌ها: WCF

[Hello World](#)

[Type Serialization](#)

[DataContract Serialization](#)

[Typed and Untyped Messages](#)

[Bindings](#)

[Message Encoding](#)

[Message Patterns](#)

[Sessions](#)

[Instancing](#)

[Concurrency](#)

[Exceptions](#)

[Transactions](#)

[HTTPS Transport Security](#)

[Message Security](#)

[Authorization](#)

[Auditing](#)

نظرات خوانندگان

نویسنده: Alex's Blog
تاریخ: ۱۳۸۸/۰۲/۳۰ ۰۶:۵۰:۰۶

ممنون

نویسنده: SirAsad
تاریخ: ۱۳۸۸/۰۲/۳۱ ۱۰:۰۲:۰۹

بسیار جالب و کاملاً به موقع. دستت درد نکنه.

آقای نصیری به یک سری مطالب در مورد بالا بردن Performance سایت مانند استفاده از Caching نیاز دارم , شما چی پیشنهاد میدید ؟

با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۲/۳۱ ۱۱:۱۳:۰۱

سلام

<http://www.codeproject.com/KB/aspnet/10ASPNetPerformance.aspx>

نویسنده: Meysam
تاریخ: ۱۳۸۹/۰۵/۱۶ ۱۰:۴۹:۵۰

بعضی از ویدئو ها داتلود نمی شوند.
ممکنه بررسی بفرمائید؟
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۵/۱۶ ۱۳:۰۳:۵۷

مطلب مربوط به یک سال قبل است ...
اصل مطلب مربوط به [اینجا](#) است.

عنوان: ویدیوهای رایگان WCF مخصوص توسعه دهندگان WPF
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۳/۱۶ ۲۲:۰۸:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: WCF

اخیرا یک سری ویدیوی رایگان در سایت codePlex در زمینه WCF منتشر شده‌اند که از آدرس زیر قابل دریافت هستند:

[WCF Guidance for WPF Developers](http://www.codeplex.com/wcf/guidanceforwpf)

این ویدیوها هر از چندگاهی نیز به روز شده و اضافه می‌شوند. بنابراین اگر به این مبحث علاقمندید، می‌توانید مشترک فید RSS آن پروژه در CodePlex شوید.

نظرات خوانندگان

نویسنده: پیمان
تاریخ: ۱۴:۲۶:۰۵ ۱۳۸۸/۰۳/۱۷

با سلام خدمت مهندس نصیری . مطالبتون بسیار عالی بود . لطفا بر روی آموزش تصویری بیشتر مانور بدید . باتشکر...

عنوان: مقایسه‌ای کوتاه بین WCF و ASMX

نویسنده: وحید نصیری

تاریخ: ۱۳۸۹/۰۳/۱۴ ۱۹:۲۰:۰۰

آدرس: www.dotnettips.info

برچسب‌ها: WCF

ویژگی	WCF	ASMX
حداقل پیشنهاد	دات نت سه	دات نت یک
هدف	جایگزینی یکپارچه‌ی فناوری‌های قبلی شامل ASMX , WSE MSMQ , COM+ Enterprise NET Remoting. و services	ارائه وب سرویس
پروتکل‌های پشتیبانی شده	HTTP TCP Named pipes MSMQ Custom UDP	HTTP only
پشتیبانی از WS-* standards	بلی	خیر
پشتیبانی از اطلاعات بایناری	بلی	خیر
پشتیبانی از REST	بلی	خیر
میزبان‌های مهیا	در هر نوع برنامه‌ی تهیه شده با دات 3 به بعد قابل میزبانی است، مانند یک برنامه کنسول، یک سرویس ویندوز ان تی و غیره. به این لیست IIS را هم می‌توان اضافه کرد.	فقط IIS
سرعت	WCF Services نسبت به ASMX Web Services از 25 تا 50 درصد سریعتر هستند + و +	
نحوه‌ی پاسخ دهی به درخواست‌ها (یا ایجاد یک وهله جدید)	Singleton / private session / per call	per-call
پشتیبانی از تراکنش‌ها (transaction)	پشتیبانی تو کار +	خیر
امنیت	پشتیبانی تو کار +	خودتان باید فکری برای این موضوع نمائید.
بسط پذیری	بلی +	خیر
مدت زمان یادگیری	حداقل یک ماه	یک روز!

نظرات خوانندگان

نویسنده: Parham, پرهام
تاریخ: ۱۳۸۹/۰۳/۱۵ ۰۸:۵۰:۴۴

به نظر من این دو اصلا با هم قابل مقایسه نیستند!!!

نویسنده: peyman naji
تاریخ: ۱۳۸۹/۰۳/۱۵ ۱۴:۰۷:۴۷

ممنون آقای نصیری .

نویسنده: Meysam
تاریخ: ۱۳۸۹/۰۳/۱۵ ۱۸:۲۶:۳۱

واسه نیازمندیهای من OData اساسی حال میده!

نویسنده: ...:A-3BT:...
تاریخ: ۱۳۸۹/۰۳/۱۹ ۰۸:۰۵:۲۴

حب من یک چیزی رو نفهمیدم که OData دقیقا چه فرقی با ADO.NET Data Services داره؟
ولی در کل WCF خیلی خوبه، حالا کسی توی ایران توی پروژه‌های تجاری ازش استفاده کرده؟

نویسنده: سامان نام نیک
تاریخ: ۱۳۸۹/۰۹/۱۶ ۱۳:۵۳:۲۶

با سلام
آقای نصیری بنده به تازگی می‌خواهم کار با wcf را شروع کنم. شما خودتان مطلب فارسی برای شروع کار آن نوشته اید که از آن شروع کنیم و بعد منابع انگلیسی را مطالعه کنیم؟ من کتاب سیلورلایت شمارامطالعه کردم و بهترین منبع برای شروع به کار بود
ممنون میشم اگه دارین لینکشو بدین؟
راستی اگر نداشتین لطفا چند کتاب یا منبع خوب انگلیسی زبان برای wcf معرفی کنید
با تشکر فراوان

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۹/۱۶ ۱۳:۵۹:۱۶

یک CBT از شرکت APP-DEV در این زمینه هست. باید بگردید پیداش کنید. خیلی خوب است.
این را در گوگل جستجو کنید: CBT Appdev WCF training

نویسنده: shahin
تاریخ: ۱۳۸۹/۱۰/۱۵ ۲۳:۵۸:۱۳

بسط پذیری یعنی چی؟

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۱۰/۱۶ ۰۰:۵۸:۴۰

لینک مطلبش کنارش هست. مفصل توضیح داده.

شش مرحله برای ایجاد WCFTransactions در WCF مقدمه و هدف:

هدف از مطلب فوق اجرا نمودن عملیات Insert ، Update و غیره... بوسیله چندین Connection در یک Transaction در زمان اجرای سرویسهای WCF میباشد. برای پیاده سازی و شرح Transaction ، سه پروژه ایجاد می‌نماییم. دو پروژه WCF سرویس و یک پروژه Client ، هر سه پروژه را در یک Solution به نام WCFTransaction اضافه می‌نماییم. در هر دو پروژه WCF بطور جداگانه Connection روی Database ایجاد می‌نماییم. سپس سعی می‌کنیم بوسیله Transaction عملیات Insert هر دو Service را کنترل نماییم. بطوریکه اگر یکی از Service ها در زمان عملیات Insert دچار مشکل شود. دیگری نیز Commit نگردد. به عبارتی در قدیم نمی‌توانستیم بیش از یک Connection در یک Transaction ایجاد نماییم. اما بوسیله Transactionscope ، انجام عملیات ، Insert ، Update و غیره... بوسیله چندین Connection به یک Database بطور همزمان در یک Transaction فراهم شده است. برای نمایش دادن عملیات Rollback نیز، به عمد خطایی ایجاد می‌کنیم، تا نحوه Rollback شدن در Transaction را مشاهده نماییم.

سعی شده است پیاده سازی و استفاده از Transaction در شش مرحله انجام شود.

مرحله اول: ایجاد دو پروژه WCFService و یک پروژه Client جهت فراخوانی (Call) کردن سرویسها

در این مرحله همانطور که از قبل نیز توضیح داده شده است، دو پروژه WCF به نامهای WCFService1 و WCFService2 ایجاد شده است و یک پروژه Client به نام WCFTransactions نیز ایجاد می‌کنیم.



مرحله دوم: افزودن Attribute ی به نام TransactionFlow به Interface سرویسها.

در این مرحله در Interface هریک از سرویس‌ها متد جدیدی به نام UpdateData اضافه می‌نماییم. که عملیات Insert into Database را انجام می‌دهد. حال بالای متد UpdateData از صفت TransactionFlow استفاده می‌نماییم. تا قابلیت Transaction برای متد فوق فعال گردد و متد فوق اجازه می‌یابد از Transaction استفاده نماید.

```
<ServiceContract(> _
Public Interface IService1

    <OperationContract(> _
    Function GetData(ByVal value As Integer) As String
```



```

<OperationContract(>> _
Function GetDataUsingDataContract(ByVal composite As CompositeType) As CompositeType

<OperationContract(>> _
<TransactionFlow(TransactionFlowOption.Allowed)> _
Sub UpdateData()

End Interface

```

مرحله سوم:

در این مرحله متد **UpdateData** را پیاده سازی می‌نماییم. بطوریکه یک **Insert Into** ساده در **Database** انجام می‌دهیم. و بالای متد فوق نیز کد زیر را می‌افزاییم.

```

<OperationBehavior(TransactionScopeRequired:=True)>

```

کد متد **UpdateData**

```

<OperationBehavior(TransactionScopeRequired:=True)>
Public Sub UpdateData() Implements IService1.UpdateData
Dim objConnection As SqlConnection = New SqlConnection(strConnection)
objConnection.Open()
Dim objCommand As SqlCommand = New SqlCommand("insert into T(ID, Age) values(10,10)",
objConnection)
objCommand.ExecuteNonQuery()
objConnection.Close()
End Sub

```

مرحله دوم و سوم را برای **Service** دوم نیز تکرار می‌نماییم.

مرحله چهارم:

در این مرحله **TransactionFlow** را در **Web.Config** دو سرویس فعال می‌نماییم. تا قابلیت استفاده از **TransactionFlow** برای سرویسها نیز فعال گردد. نحوه فعال نمودن بصورت زیر میباشد:

برای **WcfService1** خواهیم داشت:

```

<bindings>
    <wsHttpBinding>
        <binding name="TransactionalBind" transactionFlow="true"/>
    </wsHttpBinding>
</bindings>

```

و در ادامه داریم:

```

<endpoint address="" binding="wsHttpBinding"
bindingConfiguration="TransactionalBind"
contract="WcfService1.IService1">

```

برای **WcfService2** نیز خواهیم داشت:

```

<bindings>
    <wsHttpBinding>
        <binding name="TransactionalBind" transactionFlow="true"/>
    </wsHttpBinding>
</bindings>

```

و در ادامه داریم:

```
<endpoint address="" binding="wsHttpBinding"
bindingConfiguration="TransactionalBind"
contract="WcfService2.IService1">
```

مرحله پنجم:

در این مرحله دو سرویس فوق را به پروژۀ **WCFTransactions** اضافه نموده و قطعه کد زیر را درون فرم **Load** می‌نویسیم.

```
Private Sub frmmain_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load

    Using ts As New TransactionScope(TransactionScopeOption.Required)
        Try
            Dim obj As ServiceReference1.Service1Client = New ServiceReference1.Service1Client()
            obj.UpdateData()
            Dim obj1 As ServiceReference2.Service1Client = New ServiceReference2.Service1Client()
            obj1.UpdateData()
            ts.Complete()

            Catch ex As Exception
                ts.Dispose()
            End Try
        End Using
    End Sub
```

پس از اجرای برنامه دو رکورد در جدول درج خواهد شد.

مرحله ششم:

حال برای **RollBack** کردن کل عملیات و مشاهده آنها کافیه در یکی از متدهای **UpdateData** یک **Throw Exception** ایجاد نماییم.

سعی می‌کنیم با کمی تغییر در متد **UpdateData** در **WCFService2**، خطایی ایجاد شود، تا نحوه **RollBack** را مشاهده نماییم.

```
Public Sub UpdateData() Implements IService1.UpdateData
```

()Throw New Exception

```
Dim objConnection As SqlConnection = New SqlConnection(strConnection)
objConnection.Open()
Dim objCommand As SqlCommand = New SqlCommand("insert into T(ID,Age) values(101,101)",
objConnection)
objCommand.ExecuteNonQuery()
objConnection.Close()
End Sub
```

فقط کد زیر به متد **UpdateData** اضافه شده است:

```
Throw New Exception()
```

و در رویداد **Load** فرم نیز پیاده سازی آن بشکل زیر خواهد بود:

```
Using ts As New TransactionScope(TransactionScopeOption.Required)
    Try
        Dim obj As ServiceReference1.Service1Client = New ServiceReference1.Service1Client()
        obj.UpdateData()
        Throw New Exception("There was Error")
        Dim obj1 As ServiceReference2.Service1Client = New ServiceReference2.Service1Client()
        obj1.UpdateData()
        ts.Complete()

    Catch ex As Exception
        ts.Dispose()
    End Try
End Using
```

وقتی برنامه را اجرا نمایید، مشاهده می‌کنید که هیچ رکوردی درون دیتابیس درج نشده است.

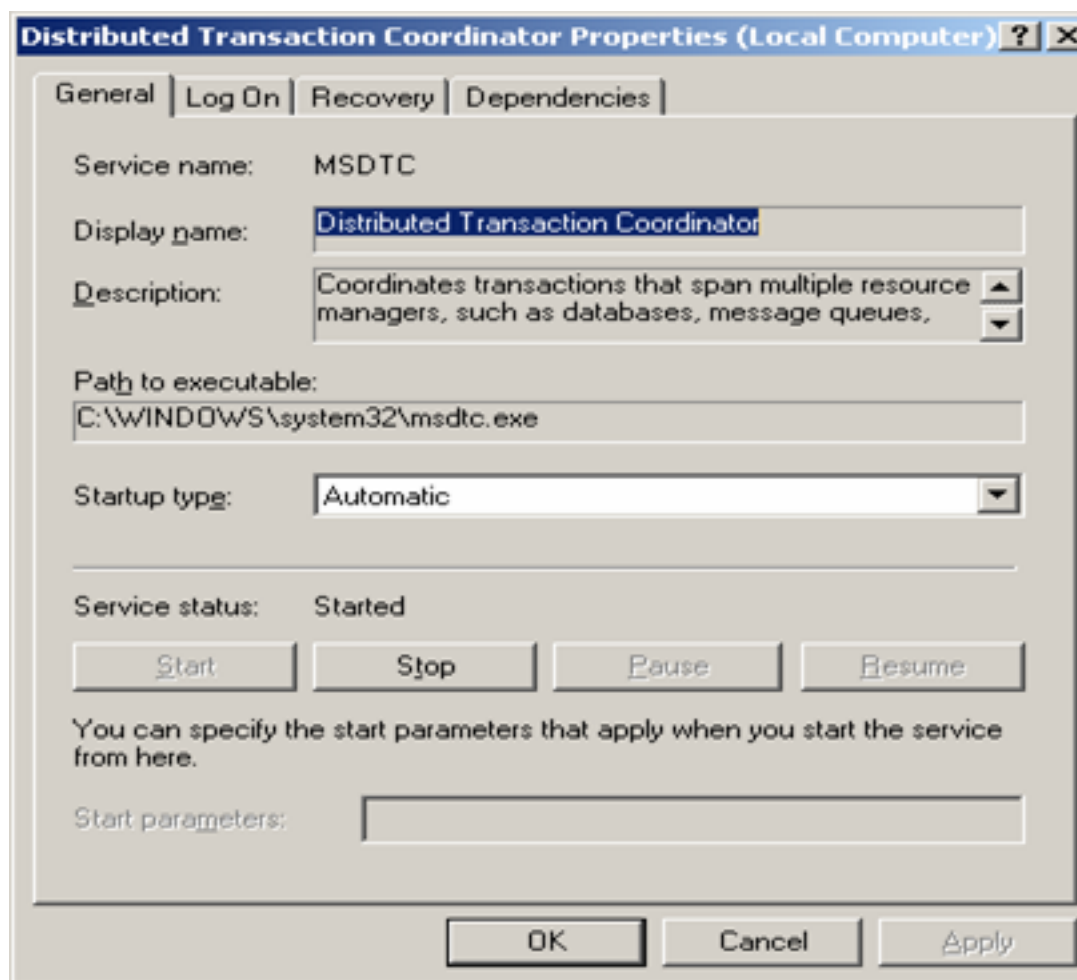
بسیار مهم: برای اینکه بتوانید بصورت **Distributed** عملیات **Transaction** را انجام دهید می‌بایست تنظیماتی را روی سرور که دیتابیس و سرویسها و کامپیوتر کلاینت انجام دهید که بصورت زیر می‌باشد:

نحوه تنظیم:

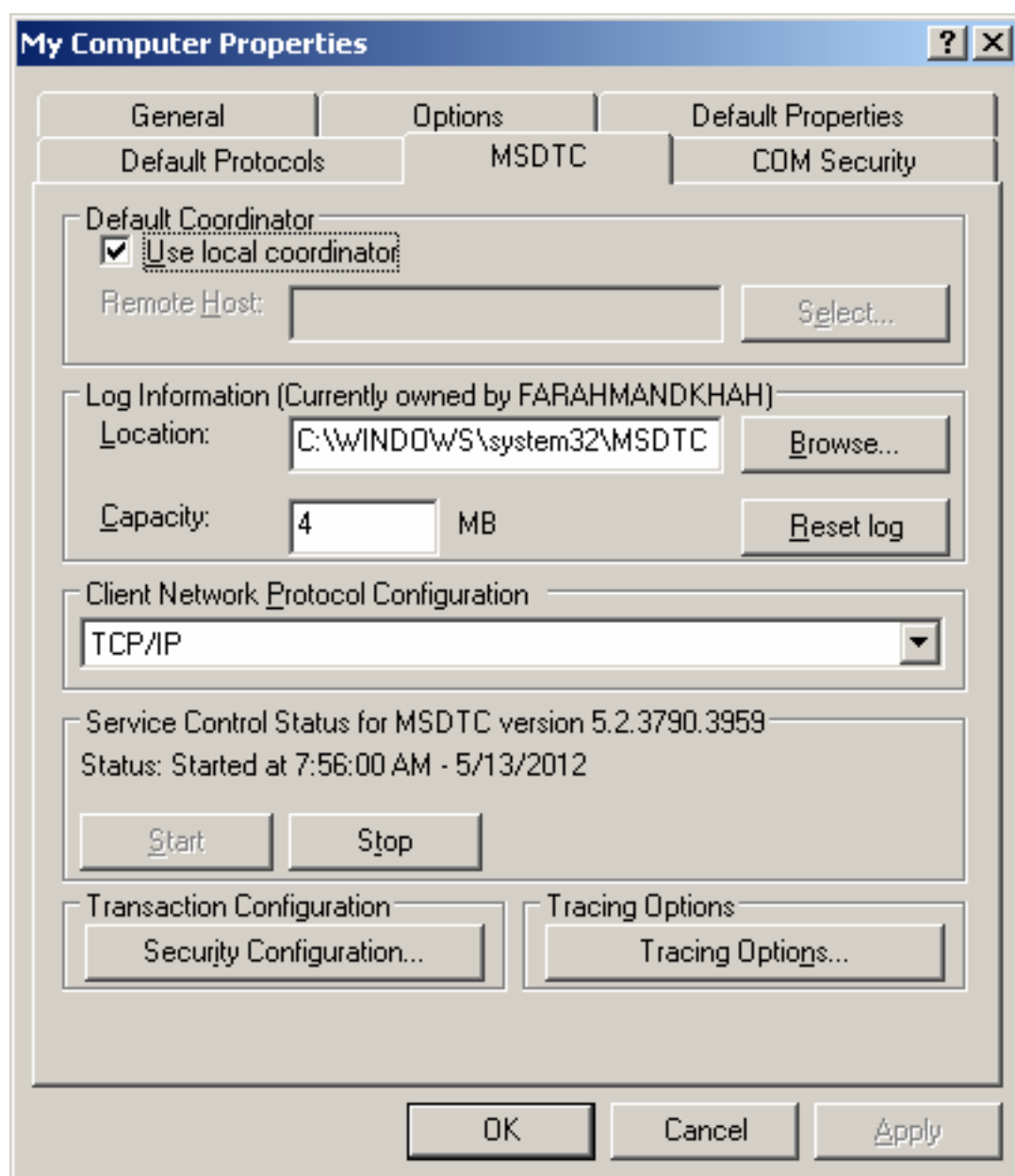
1- سرویس **Distribute Transaction Coordinator** را روی هر دو **Server**های **Database** ، **WCFService** و کامپیوتر کلاینت، **Start** می‌نماییم.

البته در شرایطی که **Service**های **WCF** و برنامه **Client** و **Database** روی یک سیستم باشد، تنظیمات فوق فقط روی همان سیستم انجام می‌شود.

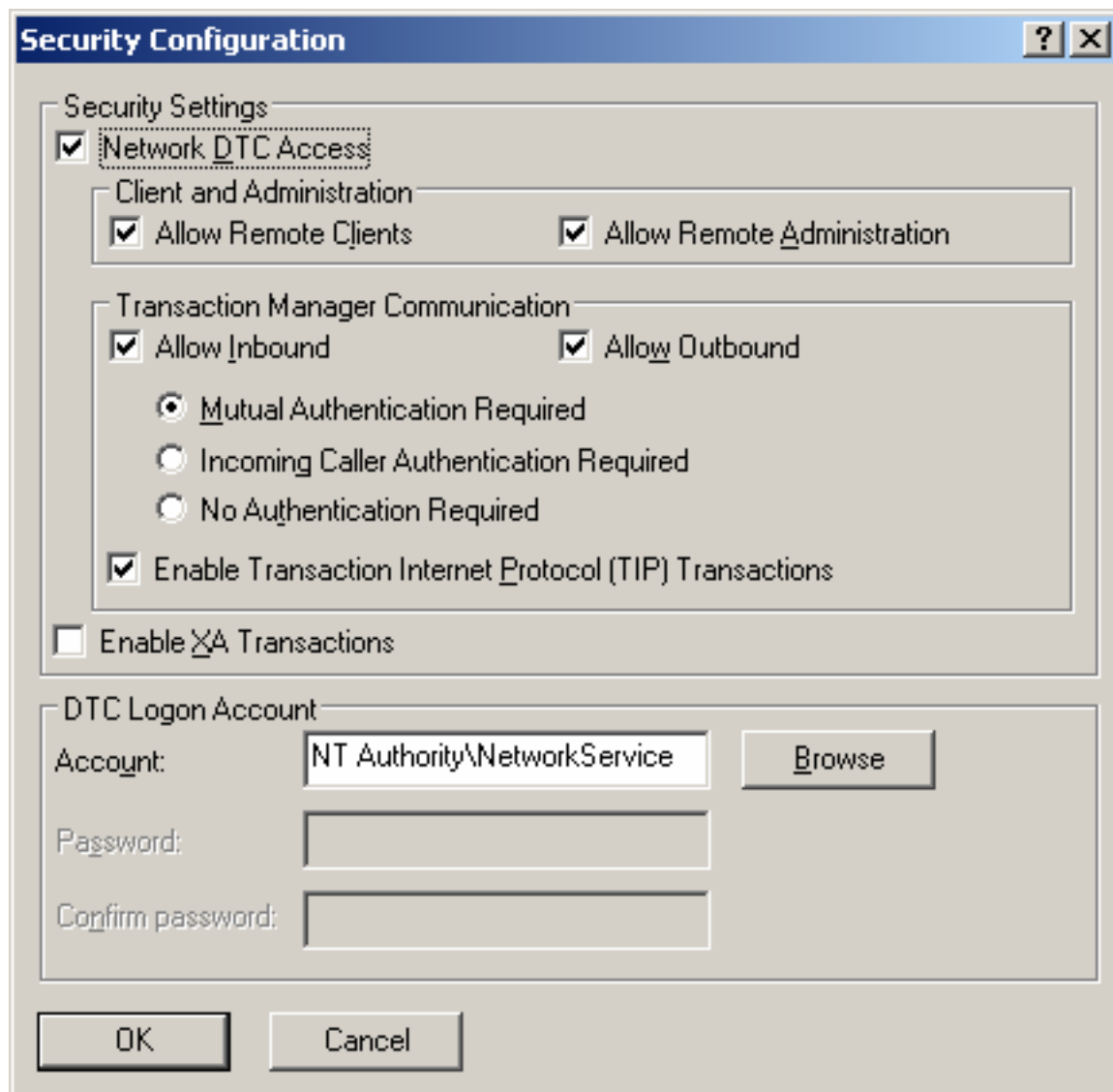
برای دسترسی به قسمت **Service** های **Windows** ابتدا **Administrative Tools** و سپس **Service** را باز نمایید و روی **Start** کلیک کنید.



2- در ادامه روی MY Computer کلیک راست نموده و تب MSDTC را انتخاب نمایید:



در ادامه روی [Security Configuration](#) کلیک نمایید. تا فرم زیر نمایش داده شود.



مطمئن شوید که آیتمهای زیر انتخاب شده باشند:

• [Network DTC Access](#)

• [Allow Remote Clients](#)

• [Allow Inbound](#)

• [Allow Outbound](#)

• [Enable Transaction Internet Protocol\(TIP\) Transactions](#)

سپس با OK کردن Service، سرویس بطور خودکار Restart می‌شود.

در ضمن اگر از SQL Server 2000 استفاده می‌نمایید، لازم است تنظیم زیر را انجام دهید.

روی SQL Server Service Manager کلیک نموده و کامبوی Service را Dropdown نمایید و [Distribute Transaction Coordinator](#) را انتخاب کنید. اما برای ورژن‌های بالاتر از SQL Server 2000 نیاز به انتخاب [Distribute Transaction](#)

Coordinator نمی‌باشد.

امیدوارم مطلب فوق مفید واقع شود، چنانچه کم و کاستی مشاهده نمودید، اینجانب را از نظرات خود بهره مند سازید.

منبع:

<http://www.codeproject.com/Articles/38793/6-Steps-to-Enable-Transactions-in-WCF>

عنوان: مدیریت Instance در WCF

نویسنده: مسعود پاکدل

تاریخ: ۱۷:۴۰ ۱۳۹۲/۰۲/۲۷

آدرس: www.dotnettips.info

گروه‌ها: WCF, InstanceContextMode

نحوه پیاده سازی و مدیریت Instance در پروژه‌های مبتنی بر WCF

نکته : آشنایی اولیه با مفاهیم WCF جهت درک صحیح مطالب الزامی است.

تشریح مسئله : در صورتی که نیاز باشد که نمونه ساخته شده از سرویس (سمت سرور) به صورت Singleton باشد بهترین روش برای پیاده سازی به چه صورت است.

برای شروع ابتدا مثال زیر را پیاده سازی می‌کنیم.

یک Contract به صورت زیر تعریف می‌کنیم:

```
[ServiceContract(SessionMode=SessionMode.Allowed)]
public interface IMyService
{
    [OperationContract]
    int GetData();
}
```

حالا یک سرویس برای پیاده سازی Interface بالا می‌نویسیم.

```
[ServiceBehavior( InstanceContextMode = InstanceContextMode.PerCall )]
public class PerCallService : IMyService
{
    int count;
    public int GetData()
    {
        return ++count;
    }
}
```

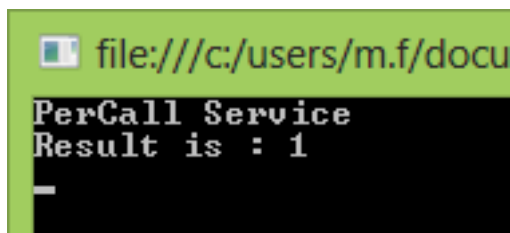
همانطور که از نام سرویس مشخص است از این سرویس به ازای هر فراخوانی یک نمونه سمت سرور ساخته می‌شود.

حالا برای مشاهده نتیجه یک پروژه ConsoleApplication ایجاد کنید و سرویس مورد نظر را از روش AddServiceReference به پروژه اضافه کرده در فایل Program کدهای زیر را کپی کنید.

```
static void Main( string[] args )
{
    Console.WriteLine( "PerCall Service" );

    MyPerCallService.MyServiceClient client = new MyPerCallService.MyServiceClient();
    int count = 0;
    for ( int i = 0 ; i < 5 ; i++ )
    {
        count = client.GetData();
    }
    Console.WriteLine( count );
    Console.ReadLine();
}
```

بعد از اجرا خروجی به صورت زیر است:



بعد از 5 بار فراخوانی متد GetData باز خروجی دارای مقدار 1 است. یعنی به ازای هر بار فراخوانی متد GetData یک نمونه از سرویس مورد نظر ساخته می‌شود. این عمل توسط خصوصیت InstanceContextMode که از نوع PerCall است به سرویس اعمال می‌شود.

حالا یک سرویس دیگر به صورت زیر ایجاد کنید.

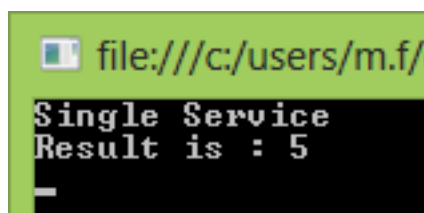
```
[ServiceBehavior( InstanceContextMode = InstanceContextMode.Single )]
public class SingleService : IMyService
{
    int count;
    public int GetData()
    {
        return ++count;
    }
}
```

تنها تفاوت این سرویس با سرویس قبلی در این است که InstanceContextMode این سرویس به صورت Single معرفی شده است. یعنی به ازای n فراخوانی فقط یک نمونه از کلاس ساخته می‌شود. این سرویس رو هم مثل روش قبلی به Client Application اضافه کنید. کد کلاس Program رو به صورت زیر تغییر دهید.

```
static void Main( string[] args )
{
    Console.WriteLine( "Single Service" );

    MySingleService.MyServiceClient client = new MySingleService.MyServiceClient();
    int count = 0;
    for ( int i = 0 ; i < 5 ; i++ )
    {
        count = client.GetData();
    }
    Console.WriteLine("Result is : {0}", count );
    Console.ReadLine();
}
```

که بعد از اجرا خروجی به صورت زیر است.



به ازای 5 بار فراخوانی سرویس متغیر Count سمت سرور مقدار قبلی خود را حفظ کرده است.

نظرات خوانندگان

نویسنده: ahmadb7

تاریخ: ۸:۴۲ ۱۳۹۲/۰۲/۳۱

با سلام؛ امکان داره منبع خوبی برای یادگیری WCF معرفی کنید

نویسنده: مسعود م. پاکدل

تاریخ: ۹:۱۱ ۱۳۹۲/۰۲/۳۱

می تونید از کتاب 348 صفحه ای WCF 4.0 Multi-tier Services Development with LINQ to Entities نوشته Mike Liu استفاده کنید.

خیلی روان و سلیس برای سطوح مبتدی و متوسط نوشته شده.

کتاب Pro WCF 4 Practical Microsoft SOA Implementation هم گزینه‌ی خیلی مناسبه. البته MSDN رو هم فراموش نکنید.

نویسنده: احسان شیروان

تاریخ: ۱۰:۵۵ ۱۳۹۳/۰۶/۱۹

سلام

یه سوالی برام پیش اومده ممنون میشم راهنمایی فرمایید:

من یک سرویس WCF ایجاد کردم و اونو به شکل زیر تنظیم کردم :

```
[ServiceContract(SessionMode=SessionMode.Required)]
```

و همچنین برای کلاس پیاده سازی کننده اینترفیس :

```
[ServiceBehavior( InstanceContextMode = InstanceContextMode.PerSession)]
```

من داخل این کلاس یه متغیر از یک کلاس به صورت سراسری تعریف کردم که میخوام ازش توی متدهای متفاوت استفاده کنم اما ظاهرا با هر بار فراخوانی باز هم این متغیر داده‌های خودشو از دست میده البته static نیست و به دلیل ساختار اون نمیتونم استاتیکش کنم

ممنون میشم راهنمایی نمایید

تشریح مسئله : چگونه متدهای سرویس WCF را Overload کنیم.

نکته : آشنایی با مفاهیم اولیه WCF برای فهم بهتر مفاهیم الزامی است.

همانطور که می‌دانیم امکان Overload کردن متدها در سرویس‌های WCF وجود ندارد. یعنی نمی‌توان 2 متد با نام و پارامترهای متفاوت داشت. به مثال زیر دقت کنید.
ابتدا یک Contract به صورت زیر تعریف کنید

```
[ServiceContract]
public interface ISampleService
{
    [OperationContract]
    int Sum( int number1, int number2 );

    [OperationContract]
    float Sum( float number1, float number2 );
}
```

در Contract بالا دو متد با یک نام ولی آرگومان‌های متفاوت داریم. حالا یک سرویس برای این Contract می‌نویسیم.

```
public class SampleService : ISampleService
{
    public int Sum( int number1, int number2 )
    {
        return number1 + number1;
    }

    public float Sum( float number1, float number2 )
    {
        return number1 + number1;
    }
}
```

اگر پروژه را کامپایل کنید پروژه بدون هیچ گونه مشکلی کامپایل خواهد شد. ولی اگر قصد استفاده از این سرویس را داشته باشیم با خطا روبرو خواهیم شد. از روش AddServiceReference استفاده کنید و سرویس مورد نظر را سمت کلاینت اضافه کنید. با خطای زیر روبرو خواهید شد.

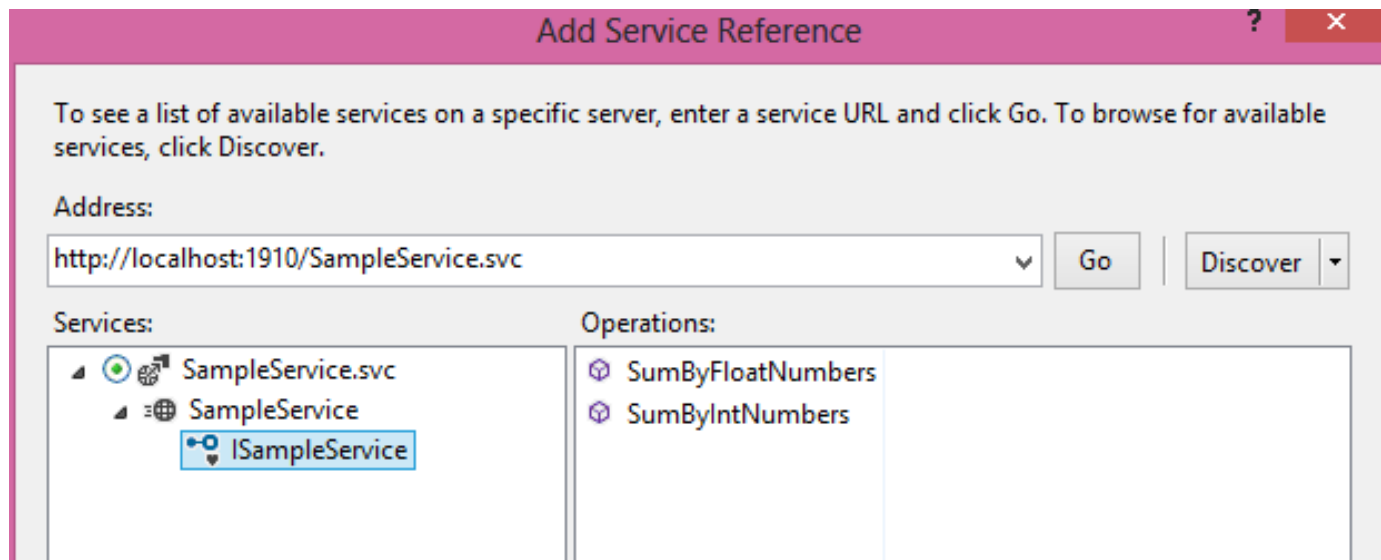
```
Cannot have two operations in the same contract with the same name,
methods Sum and Sum in type Service.ISampleService violate this rule.
You can change the name of one of the operations by changing the method name
or by using the Name property of OperationContractAttribute.
```

در این خطا به صورت کامل روش حل این مسئله گفته شده است. برای حل این مسئله باید از خاصین Name در OperationContractAttribute استفاده کرد. Contract بالا را به صورت زیر تغییر دهید.

```
[ServiceContract]
public interface ISampleService
{
    [OperationContract( Name = "SumByIntNumbers" )]
    int Sum( int number1, int number2 );

    [OperationContract( Name = "SumByFloatNumbers" )]
    float Sum( float number1, float number2 );
}
```

حال اگر سرویس مورد نظر را به پروژه سمت کلاینت اضافه کنیم دو متد با نامهای SumByFloatNumbers و SumByIntNumbers خواهیم داشت. البته اگر از روش Self Hosted استفاده کنیم دقیقاً دو متد با نام Sum خواهیم داشت و Overloading را سمت سرور و کلاینت خواهیم داشت ولی در روش IIS Hosting و استفاده از AddServiceReference از خاصیت Name برای این کار استفاده میشود.



موفق باشید.

تشریح مسئله : در صورتی که بعد از انتشار برنامه؛ در نسخه بعدی مدل سمت سرور تغییر کرده باشد و امکان بروز رسانی مدل های سمت کلاینت وجود نداشته باشد برای حل این مسئله بهترین روش کدام است.
نکته : برای فهم بهتر مطالب آشنایی اولیه با مفاهیم WCF الزامی است.
ابتدا مدل زیر را در نظر بگیرید:

```
[DataContract]
public class Book
{
    [DataMember]
    public int Code { get; set; }

    [DataMember]
    public string Name { get; set; }
}
```

حالا یک سرویس برای دریافت و ارسال اطلاعات این مدل به کلاینت می نویسیم.

```
[ServiceContract]
public interface ISampleService
{
    [OperationContract]
    IEnumerable<Book> GetAll();

    [OperationContract]
    void Save( Book book );
}
```

و سرویسی که Contract بالا رو پیاده سازی کند.

```
public class SampleService : ISampleService
{
    public List<Book> ListOfBook
    {
        get;
        private set;
    }

    public SampleService()
    {
        ListOfBook = new List<Book>();
    }

    public IEnumerable<Book> GetAll()
    {
        ListOfBook.AddRange( new Book[]
        {
            new Book(){Code=1 , Name="Book1"},
            new Book(){Code=2 , Name="Book2"},
        } );
        return ListOfBook;
    }

    public void Save( Book book )
    {
        ListOfBook.Add( book );
    }
}
```

متد GetAll برای ارسال اطلاعات به کلاینت و متد Save نیز برای دریافت اطلاعات از کلاینت.
حالا یک پروژه Console Application بسازید و از روش AddServiceReference سرویس مورد نظر را به Client اضافه کنید.
برنامه را تست کنید. بدون هیچ مشکلی کار می کند.

حالا اگر در نسخه بعدی سیستم مجبور شویم به مدل Book یک خاصیت دیگر به نام Author را نیز اضافه کنیم و امکان Update کردن سرویس در سمت کلاینت وجود نداشته باشد چه اتفاقی خواهد افتاد. به صورت زیر:

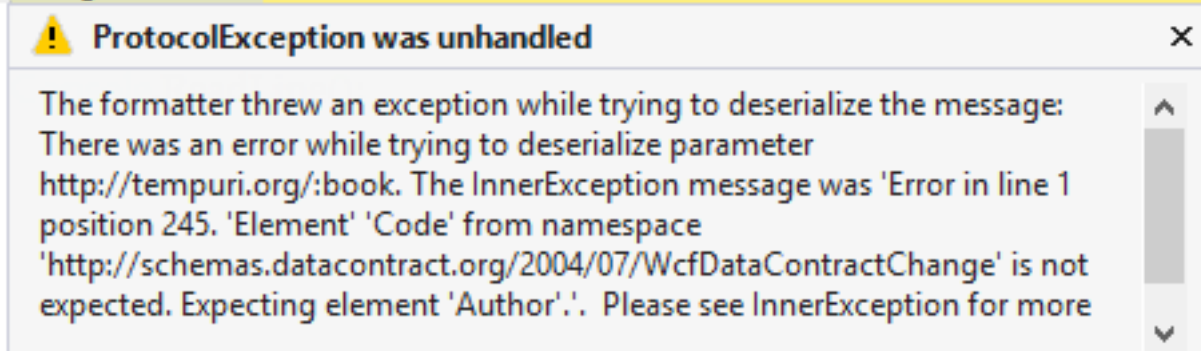
```
[DataContract]
public class Book
{
    [DataMember]
    public int Code { get; set; }
    [DataMember]
    public string Name { get; set; }

    [DataMember]
    public string Author { get; set; }
}
```

به طور پیش فرض اگر در DataContract های سمت سرور و کلاینت اختلاف وجود داشته باشد این موارد نادیده گرفته می شوند. یعنی همیشه مقدار خاصیت Author برابر null خواهد بود. نکته : برای Value Type ها مقادیر پیش فرض و برای Reference Type ها مقدار Null. اگر برای DataMemberAttribute خاصیت IsRequired را برابر true کنیم از این پس برای هر درخواستی که مقدار Author آن مقدار نداشته باشد یک Protocol Exception پرتاب می شود. به صورت زیر:

```
[DataMember(IsRequired = true)]
public string Author { get; set; }
```

sampleService.Save(new BookService.Book() { Code = 3, Name = "E



اما این همیشه راه حل مناسبی نیست.

روش دیگر این است که Deserialize کردن مدل را تغییر دهیم. بدین معنی که هر گاه مقدار Author برابر Null بود یک مقدار پیش فرض برای آن در نظر بگیریم. این کار با نوشتن یک متد و قراردادن OnDeserializingAttribute به راحتی امکان پذیر است. کلاس Book به صورت زیر تغییر می کند.

```
[DataContract]
public class Book
{
    [DataMember]
    public int Code { get; set; }
    [DataMember]
    public string Name { get; set; }

    [DataMember(IsRequired = true)]
    public string Author { get; set; }

    [OnDeserializing]
    private void OnDeserializing( StreamingContext context )
    {
        if ( string.IsNullOrEmpty( Author ) )
        {

```

```

        Author = "Masoud Pakdel";
    }
}

```

حال اگر از سمت کلاینت کلاس Book دریافت شود که مقدار خاصیت Author آن برابر Null باشد توسط متد OnDeserializing مقدار پیش فرض به آن اعمال می شود. مثل تصویر زیر:

```

public void Save( Book book )
{
    ListOfBook.Add( book );
}

```

book {WcfDataContractChange.Book}

- Author: "Masoud Pakdel"
- Code: 3
- Name: "Book3"

روش بعدی استفاده از اینترفیس IExtensibleDataObject است. بعد از اینکه کلاس Book این اینترفیس را پیاده سازی کرد مشکل Versioning Round Trip حل می شود. به این صورت که سرویس یا کلاینتی که نسخه قدیمی را می شناسد اگر نسخه جدید را دریافت کند خصوصیتی را که نمی شناسد مثل Author در خاصیت ExtensionData ذخیره می شود و هنگامی که کلاس Book برای سرویس یا کلاینتی که نسخه جدید را می شناسد DataContractSerializer اطلاعات مورد نظر را از خصوصیت ExtensionData بیرون می کشد و کلاس Book جدید را باز سازی می کند. بررسی کلاس ExtensionData توسط خود DataContractSreializer انجام می شود و نیاز به هیچ گونه ای کد نویسی ندارد.

```

[DataContract]
public class Book : IExtensibleDataObject
{
    [DataMember]
    public int Code { get; set; }
    [DataMember]
    public string Name { get; set; }

    [DataMember]
    public string Author { get; set; }

    public virtual ExtensionDataObject ExtensionData
    {
        get { return _extensionData; }
        set
        {
            _extensionData = value;
        }
    }
    private ExtensionDataObject _extensionData;
}

```

اگر کد متد GetAll سمت سرور را به صورت زیر تغییر دهیم که خاصیت Author هم مقدار داشته باشد با استفاده از خاصیت ExtensionData کلاینت هم از این مقدار مطلع خواهد شد.

```

public IEnumerable<Book> GetAll()
{
    ListOfBook.AddRange( new Book[]
    {
        new Book(){Code=1 , Name="Book1", Author="Masoud Pakdel"},
        new Book(){Code=2 , Name="Book2" },
    }

```



```

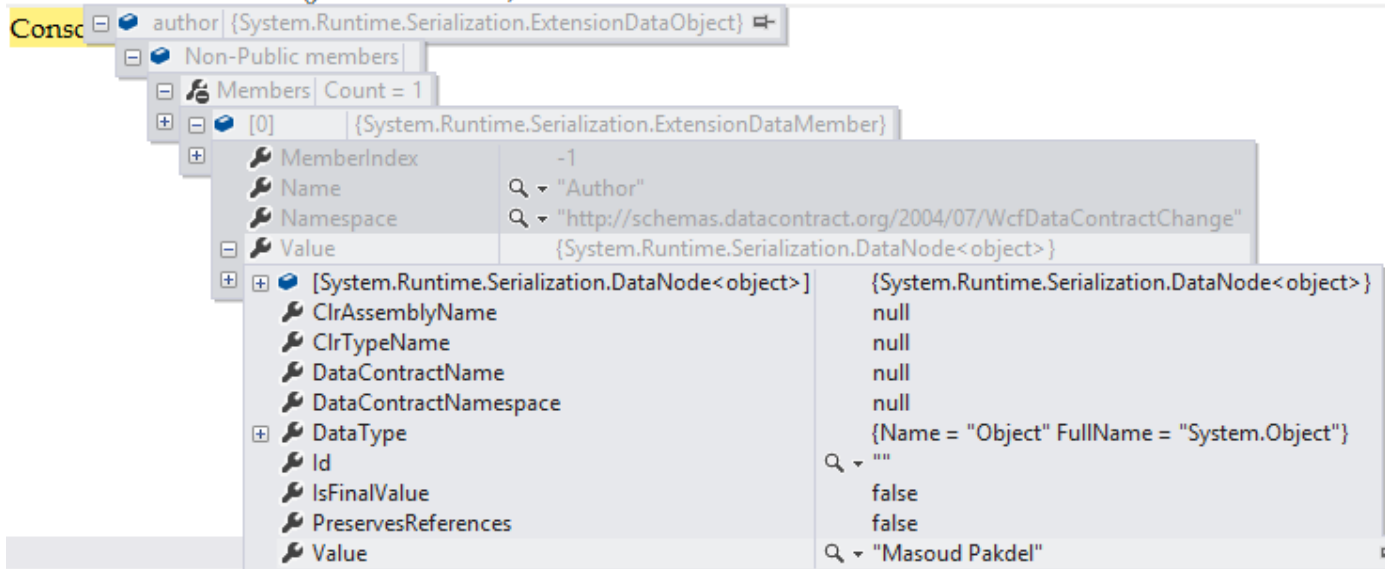
    } );
    return ListOfBook;
}

```

کلاینت هم به صورت زیر :

```
var result = sampleService.GetAll();
```

```
var author = result.First().ExtensionData;
```



همان طور که می بینید این نسخه از کلاینت هیچ گونه اطلاعی از وجود یک خاصیت به نام Author ندارد ولی از طریق ExtensionData متوجه می شود یک خاصیت به نام Author به مدل سمت سرور اضافه شده است.

اما در صورتی که قصد داشته باشیم که یک سرویس خاص از همان نسخه قدیمی کلاس Book استفاده کند و نیاز به نسخه جدید آن نداشته باشد می توانیم این کار را از طریق مقدار دهی True به خاصیت IgnoreExtensionDataObject در ServiceBehaviorAttribute انجام داد. بدین شکل

```

[ServiceBehavior( IgnoreExtensionDataObject = true )]
public class SampleService : ISampleService

```

از این پس سرویس بالا از همان مدل Book بدون خاصیت Author استفاده می کند.

منابع :

<http://msdn.microsoft.com/en-us/library/system.runtime.serialization.iextendibledataobject.aspx>

<http://msdn.microsoft.com/en-us/library/ms731083.aspx>

<http://msdn.microsoft.com/en-us/library/ms733832.aspx>

تشریح مسئله : KnownTypeAttribute چیست و چگونه از آن استفاده کنیم؟

پیش نیاز : آشنایی اولیه با مفاهیم WCF برای فهم بهتر مطالب

در ابتدا یک Wcf Service Application ایجاد کنید و مدل زیر را بسازید:

```
[DataContract]
public abstract class Person
{
    [DataMember]
    public int Code { get; set; }

    [DataMember]
    public string Name { get; set; }
}
```

{ یک کلاس پایه برای Person ایجاد کردیم به صورت abstract که وهله سازی از آن میسر نباشد و 2 کلاس دیگر می‌سازیم که از کلاس بالا ارث ببرند:

کلاس #1

```
[DataContract]
public class Student : Person
{
    [DataMember]
    public int StudentId { get; set; }
}
```

کلاس #2

```
[DataContract]
public class Teacher : Person
{
    public int TeacherId { get; set; }
}
```

فرض کنید قصد داریم سرویسی ایجاد کنیم که لیست تمام اشخاص موجود در سیستم را در اختیار ما قرار دهد. (هم Student و هم Teacher). ابتدا Contract مربوطه را به صورت زیر تعریف می‌کنیم:

```
[ServiceContract]
public interface IStudentService
{
    [OperationContract]
    IEnumerable<Person> GetAll();
}
```

همان طور که می‌بینید خروجی متد GetAll از نوع Person است (نوع پایه کلاس Student , Teacher). سرویس مربوطه بدین شکل خواهد شد.

```
public class StudentService : IStudentService
{
    public IEnumerable<Person> GetAll()
    {
        List<Person> listOfPerson = new List<Person>();

        listOfPerson.Add( new Student() { Code = 1, StudentId = 123, Name = "Masoud Pakdel" } );
        listOfPerson.Add( new Student() { Code = 1, StudentId = 123, Name = "Mostafa Asgari" } );
        listOfPerson.Add( new Student() { Code = 1, StudentId = 123, Name = "Saeed Alizadeh" } );

        listOfPerson.Add( new Teacher() { Code = 1, TeacherId = 321, Name = "Mahdi Rad" } );
    }
}
```

```

        listOfPerson.Add( new Teacher() { Code = 1, TeacherId = 321, Name = "Mohammad Heydari" } );
        listOfPerson.Add( new Teacher() { Code = 1, TeacherId = 321, Name = "Saeed Khatami" } );

        return listOfPerson;
    }

```

در این سرویس در متد GetAll لیستی از تمام اشخاص رو ایجاد می‌کنیم. 3 تا Student و 3 تا Teacher رو به این لیست اضافه میکنیم. برای نمایش اطلاعات در خروجی یک پروژه Console Application ایجاد کنید و سرویس بالا رو از روش AddServiceReference به پروژه اضافه کنید سپس در کلاس Program کدهای زیر رو کپی کنید.

```

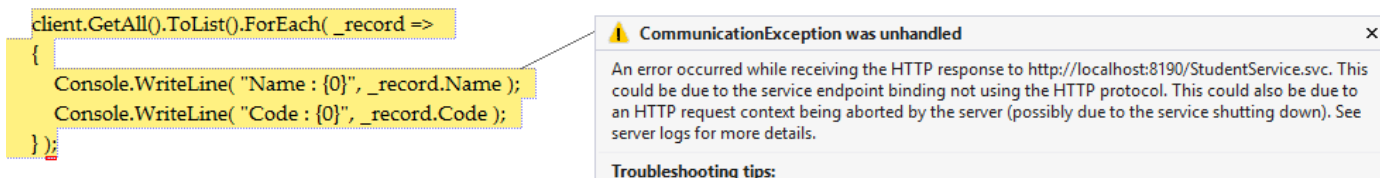
class Program
{
    static void Main( string[] args )
    {
        StudentService.StudentServiceClient client = new StudentService.StudentServiceClient();

        client.GetAll().ToList().ForEach( _record =>
        {
            Console.Write( "Name : {0}", _record.Name );
            Console.WriteLine( "Code : {0}", _record.Code );
        } );

        Console.ReadLine();
    }
}

```

پروژه رو کامپایل کنید. تا اینجا هیچ گونه مشکلی مشاهده نشد و انتظار داریم که خروجی مورد نظر رو مشاهده کنیم. بعد از اجرای پروژه با خطای زیر متوقف می‌شویم:



مشکل از اینجا ناشی می‌شود که هنگام عمل سریالایز، WCF Runtime با توجه به وهله سازی از کلاس Person می‌دونه که باید کلاس Student یا Teacher رو سریالایز کنه ولی در هنگام عمل دی سریالایز، WCF Runtime این موضوع رو درک نمی‌کنه به همین دلیل یک Communication Exception پرتاب می‌کنه. برای حل این مشکل و برای اینکه WCF Deserialize Engine رو متوجه نوع وهله سازی کلاس‌های مشتق شده از کلاس پایه کنیم باید از KnownTypeAttribute استفاده کنیم. فقط کافیست که این Attribute رو بالای کلاس Person به ازای تمام کلاس‌های مشتق شده از اون قرار بدید. بدین صورت:

```

[DataContract]
[KnownType( typeof( Student ) )]
[KnownType( typeof( Teacher ) )]
public abstract class Person
{
    [DataMember]
    public int Code { get; set; }

    [DataMember]
    public string Name { get; set; }
}

```

حالا پروژه سمت سرور رو دوباره کامپایل کنید و سرویس سمت کلاینت رو Update کنید. بعد پروژه رو دوباره اجرا کرده تا خروجی زیر رو مشاهده کنید.

```
Name : Masoud Pakdel Code : 1
Name : Mostafa Asgari Code : 1
Name : Saeed Alizadeh Code : 1
Name : Mahdi Rad Code : 1
Name : Mohammad Heydari Code : 1
Name : Saeed Khatami Code : 1
```

با وجود KnownType دیگه WCF Deserialize Engine میدونه که باید از کدام DataContract برای عمل دی سریالاز نمونه ساخته شده از کلاس Person استفاده کنه. دانستن این مطلب هنگام پیاده سازی [مفاهیم ارث بری در ORM ها](#) زمانی که از WCF استفاده می‌کنیم ضروری است.

نظرات خوانندگان

نویسنده: مصطفی عسگری
تاریخ: ۱۳۹۲/۰۳/۱۱ ۹:۴۹

ممنون ... استفاده بردیم.

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۳/۱۵ ۱۸:۹

ترجمه این مطلب در code project به تاریخ Jun 4 که میشه دیروز البته

[What is KnownType Attribute and How to Use It in WCF Technology](#)

نویسنده: مسعود م. پاکدل
تاریخ: ۱۳۹۲/۰۳/۱۵ ۲۲:۵۷

ممنون از دقت و اطلاع شما

تشریح مسئله : در DataContractSerializer قابلیت به عنوان سریالایز کردن objectها به صورت درختی وجود دارد که اصطلاحاً به اون Circular References گفته می‌شود در این پست قصد دارم روش پیاده سازی، به همراه مزایای استفاده از این روش رو توضیح بدم.

نکته : آشنایی با مفاهیم اولیه WCF برای درک بهتر مطالب الزامی است.

در ابتدا لازم است تا مدل برنامه را تعریف کنیم. ابتدا یک پروژه از نوع WCF Service Application ایجاد کنید و مدل زیر را بسازید.

Employee#

```
[DataContract]
public class Employee
{
    [DataMember]
    public string Name { get; set; }

    [DataMember]
    public Employee Manager { get; set; }
}
```

Department#

```
[DataContract]
public class Department
{
    [DataMember]
    public string DeptName { get; set; }

    [DataMember]
    public List<Employee> Staff { get; set; }
}
```

در مدل Employee یک خاصیت از نوع خود کلاس Employee وجود دارد که برای پیاده سازی مدل به صورت درختی است. در مدل Department هم لیستی از کارمندان دپارتمان را ذخیره می‌کنیم و قصد داریم این مدل رو از سمت سرور به کلاینت انتقال دهیم و نوع سریالایز کردن WCF رو در این مورد مشاهده کنیم. ابتدا سرویس و Contract مربوطه را می‌نویسیم.

Contract#

```
[ServiceContract]
public interface IDepartmentService
{
    [OperationContract]
    Department GetOneDepartment();
}
```

Service#

```
public class DepartmentService : IDepartmentService
{
    public Department GetOneDepartment()
    {
        List<Employee> listOfEmployees = new List<Employee>();

        var masoud = new Employee() { Name = "Masoud" };
        var saeed = new Employee() { Name = "Saeed", Manager = masoud };
        var peyman = new Employee() { Name = "Peyman", Manager = saeed };
    }
}
```

```

        var mostafa = new Employee() { Name = "Mostafa", Manager = saeed };
        return new Department() { DeptName = "IT", Staff = new List<Employee>() { masoud, saeed,
        peyman, mostafa } };
    }
}

```

همانطور که در سرویس بالا مشخص است لیستی از کارمندان ساخته شده که خود این لیست به صورت درختی است و بعضی از کارمندان به عنوان مدیر کارمند دیگر تعیین شد است. حال برای دریافت اطلاعات سمت کلاینت یک پروژه از نوع Console ایجاد کنید و از روش AddServiceReference سرویس مورد نظر را اضافه کنید و کدهای زیر را در کلاس Program کپی کنید.

```

class Program
{
    static void Main( string[] args )
    {
        DepartmentServiceClient client = new DepartmentServiceClient();

        var result = client.GetOneDepartment();
        WriteDataToFile( result );

        Console.ReadKey();
    }

    private static void WriteDataToFile( Department data )
    {
        DataContractSerializer dcs = new DataContractSerializer( typeof( Department ) );
        var ms = new MemoryStream();
        dcs.WriteObject( ms, data );
        ms.Seek( 0, SeekOrigin.Begin );
        var sr = new StreamReader( ms );
        var xml = sr.ReadToEnd();
        string filePath = @"d:\data.xml";
        if ( !File.Exists( filePath ) )
        {
            File.Create( filePath );
        }
        using ( TextWriter writer = new StreamWriter( filePath ) )
        {
            writer.Write( xml );
        }
    }
}

```

یک متد به نام WriteDataToFile نوشتم که اطلاعات Department رو به فرمت Xml در فایل ذخیره می‌کند. بعد از اجرای برنامه خروجی مورد نظر در فایل Xml به صورت زیر است.

```

<Department xmlns="http://schemas.datacontract.org/2004/07/Service"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <Name>IT</Name>
  <Staff>
    <Employee>
      <Manager i:nil="true"/>
      <Name>Masoud</Name>
    </Employee>
    <Employee>
      <Manager>
        <Manager i:nil="true"/>
        <Name>Masoud</Name>
      </Manager>
      <Name>Saeed</Name>
    </Employee>
    <Employee>
      <Manager>
        <Manager>
          <Manager i:nil="true"/>
          <Name>Masoud</Name>
        </Manager>
        <Name>Saeed</Name>
      </Manager>
      <Name>Peyman</Name>
    </Employee>
    <Employee>
      <Manager>
        <Manager>

```

```

    <Manager i:nil="true"/>
    <Name>Masoud</Name>
  </Manager>
  <Name>Saeed</Name>
</Manager>
  <Name>Mostafa</Name>
</Employee>
</Staff>
</Department>

```

در فایل بالا مشاهده می‌کنید که تعداد تکرار Masoud به اندازه تعداد استفاده اون در Department است. در این قسمت قصد داریم که از Circular Referencing موجود در DataContractSerializer استفاده کنیم. برای این کار کفایت از خاصیت IsReference موجود در DataContract استفاده کنیم. پس مدل Employee به صورت زیر تغییر میباید:

```

[DataContract( IsReference = true )]
public class Employee
{
    [DataMember]
    public string Name { get; set; }

    [DataMember]
    public Employee Manager { get; set; }
}

```

پروژه رو دوباره Run کنید و فایل xml ساخته شده به صورت زیر تغییر می‌کند.

```

<Department xmlns="http://schemas.datacontract.org/2004/07/Service"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <Name>IT</Name>
  <Staff>
    <Employee z:Id="i1" xmlns:z="http://schemas.microsoft.com/2003/10/Serialization/">
      <Manager i:nil="true"/>
      <Name>Masoud</Name>
    </Employee>
    <Employee z:Id="i2" xmlns:z="http://schemas.microsoft.com/2003/10/Serialization/">
      <Manager z:Ref="i1"/>
      <Name>Saeed</Name>
    </Employee>
    <Employee z:Id="i3" xmlns:z="http://schemas.microsoft.com/2003/10/Serialization/">
      <Manager z:Ref="i2"/>
      <Name>Peyman</Name>
    </Employee>
    <Employee z:Id="i4" xmlns:z="http://schemas.microsoft.com/2003/10/Serialization/">
      <Manager z:Ref="i2"/>
      <Name>Mostafa</Name>
    </Employee>
  </Staff>
</Department>

```

کاملاً واضح است که تعداد Masoud به عنوان Employee فقط یک بار است و از z:ref برای ارتباط بین Objectها استفاده می‌شود. در این روش فقط یک بار هر object سریالایز می‌شود و هر جا که نیاز به استفاده از object مربوطه باشد فقط یک ارجاع به آن خواهد شد.

مزایا : استفاده از این روش در هنگام عمل سریالایز داده‌های زیاد و زمانی که تعداد Objectهای موجود در ObjectGraph زیاد باشد باعث افزایش کارایی و سرعت انجام عملیات سریالایز می‌شود.

نکته : آشنایی با مفاهیم پایه WCF برای فهم بهتر مفاهیم توصیه می شود.

امروزه استفاده از WCF در پروژه های SOA بسیار فراگیر شده است. کمتر کسی است که در مورد قدرت تکنولوژی WCF نشنیده باشد یا از این تکنولوژی در پروژه های خود استفاده نکرده باشد. WCF مدل برنامه نویسی یکپارچه میکروسافت برای ساخت نرم افزارهای سرویس گرا است و برای توسعه دهندگان امکانی را فراهم می کند که راهکارهایی امن، و مبتنی بر تراکنش را تولید نمایند که قابلیت استفاده در بین پلتفرم های مختلف را دارند. قبل از WCF توسعه دهندگان پروژه های نرم افزاری برای تولید پروژه های توزیع شده باید شرایط موجود برای تولید و توسعه را در نظر می گرفتند. برای مثال اگر استفاده کننده از سرویس در داخل سازمان و بر پایه دات نت تهیه شده بود از net remoting استفاده می کردند و اگر استفاده کننده سرویس از خارج سازمان یا مثلا بر پایه تکنولوژی J2EE بود از Web Service استفاده می شد. با ظهور WCF این تکنولوژی با هم تجمیع شدند (بهتر بگم تبدیل به یک تکنولوژی واحد شدند) و دیگر خبری از net remoting یا web service ها نیست.

WCF با تمام قدرت و امکاناتی که داراست دارای نقاط ضعفی هم می باشد که البته این معایب (یا محدودیت) بیشتر جهت سازگار سازی سرویس های نوشته شده با سیستم ها و پروتکل های مختلف است. برای انتقال داده ها از طریق WCF بین سیستم های مختلف باید داده های مورد نظر حتما سریالایز شوند که مثال هایی از این دست رو در همین سایت می تونید مطالعه کنید:

(^) و (^) و (^)

با توجه به این که داده ها سریالایز می شوند، در نتیجه امکان انتقال داده هایی که از نوع object هستند در WCF وجود ندارد. بلکه نوع داده باید صراحتا ذکر شود و این نوع باید قابلیت سریالایز شدن را دارا باشد. برای مثال شما نمی تونید متدی داشته باشید که پارامتر ورودی آن از نوع delegate باشد یا کلاسی باشد که صفت [Serializable] در بالای اون قرار نداشته باشد یا کلاسی باشد که صفت DataContract برای خود کلاس و صفت DataMember برای خاصیت های اون تعریف نشده باشد. حالا سوال مهم این است اگر متدی داشته باشیم که پارامتر ورودی آن حتما باید از نوع delegate باشد چه باید کرد؟

برای تشریح بهتر مسئله یک مثال می زنم؟

سرویس داریم برای اطلاعات کتاب ها. قصد داریم متدی بنویسیم که پارامتر ورودی آن از نوع Lambda Expression است تا Query مورد نظر کاربر از سمت کلاینت به سمت سرور دریافت کند و خروجی مورد نظر را با توجه به Query ورودی به کلاینت برگشت دهد. (متدی متداول در اکثر پروژه ها). به صورت زیر عمل می کنیم.

*ابتدا یک Blank Solution ایجاد کنید.

*یک ClassLibrary به نام Model ایجاد کنید و کلاسی به نام Book در آن بسازید. (همانطور که می بینید کلاس مورد نظر سریالایز شده است):

```
[DataContract]
public class Book
{
    [DataMember]
    public int Code { get; set; }

    [DataMember]
    public string Title { get; set; }
}
```

* یک WCF Service Application ایجاد کنید

یک Contract برای ارتباط بین سرور و کلاینت می‌سازیم:

```
using System;
using System.Collections.Generic;
using System.Linq.Expressions;
using System.ServiceModel;

namespace WcfLambdaExpression
{
    [ServiceContract]
    public interface IBookService
    {
        [OperationContract]
        IEnumerable<Book> GetByExpression( Expression<Func<Book, bool>> expression );
    }
}
```

متد GetByExpression دارای پارامتر ورودی expression است که نوع آن نیز Lambda Expression می‌باشد. حال یک سرویس ایجاد می‌کنیم:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;

namespace WcfLambdaExpression
{
    public class BookService : IBookService
    {
        public BookService()
        {
            ListOfBook = new List<Book>();
        }

        public List<Book> ListOfBook
        {
            get;
            private set;
        }

        public IEnumerable<Book> GetByExpression( Expression<Func<Book, bool>> expression )
        {
            ListOfBook.AddRange( new Book[]
            {
                new Book(){Code = 1 , Title = "Book1"},
                new Book(){Code = 2 , Title = "Book2"},
                new Book(){Code = 3 , Title = "Book3"},
                new Book(){Code = 4 , Title = "Book4"},
                new Book(){Code = 5 , Title = "Book5"},
            } );

            return ListOfBook.AsQueryable().Where( expression );
        }
    }
}
```

بعد از Build پروژه همه چیز سمت سرور آماده است. یک پروژه دیگر از نوع Console ایجاد کنید و از روش AddServiceReference سعی کنید که سرویس مورد نظر را به پروژه اضافه کنید. در هنگام Add Service Reference برای اینکه سرویس سمت سرور و کلاینت هر دو با یک مدل کار کنند باید از یک Reference assembly استفاده کنند و کافی است از قسمت Advanced گزینه Reuse types in referenced assemblies را تیک بزنید و assembly های مورد نظر را انتخاب کنید. (در این پروژه باید Model و System.Xml.Linq را انتخاب کنید)

Data Type

☐ Always generate message contracts

Collection type: System.Array

Dictionary collection type: System.Collections.Generic.Dictionary

☒ Reuse types in referenced assemblies

☐ Reuse types in all referenced assemblies

☒ Reuse types in specified referenced assemblies:

<input type="checkbox"/> Common	
<input type="checkbox"/> Microsoft.CSharp	
<input checked="" type="checkbox"/> Model	
<input type="checkbox"/> mscorlib	
<input type="checkbox"/> System	
<input type="checkbox"/> System.Core	
<input type="checkbox"/> System.Data	

به طور حتم با خطا روبرو خواهید شد. دلیل آن هم این است که امکان سریالایز کردن برای پارامتر ورودی expression میسر نیست.
خطای مربوطه به شکل زیر خواهد بود:

Type 'System.Linq.Expressions.Expression`1[System.Func`2[WcfLambdaExpression.Book,System.Boolean]]' cannot be serialized.
Consider marking it with the DataContractAttribute attribute, and marking all of its members you want serialized with the DataMemberAttribute attribute.
If the type is a collection, consider marking it with the CollectionDataContractAttribute.
See the Microsoft .NET Framework documentation for other supported types

حال چه باید کرد؟

روش های زیادی برای برطرف کردن این محدودیت وجود دارد. اما در این پست روشی رو که خودم از اون استفاده می کنم رو براتون شرح می دهم.

در این روش باید از XElement استفاده شود که در فضای نام System.Linq.Xml قرار دارد. یعنی آرگومان ورودی سمت کلاینت باید به فرمت Xml سریالایز شود و سمت سرور دوباره دی سریالایز شده و تبدیل به یک Lambda Expression شود. اما سریالایز کردن Lambda Expression واقعا کاری سخت و طاقت فرسا است. با توجه به این که در اکثر پروژه ها این متدها به صورت Generic نوشته می شوند. برای حل این مسئله بعد از مدتی جستجو، کلاسی رو پیدا کردم که این کار رو برام انجام می داد. بعد از مطالعه دقیق و مشاهده روش کار کلاس، تغییرات مورد نظرم رو اعمال کردم و الان در اکثر پروژه ها هم دارم از این کلاس استفاده می کنم. یک مثال از روش استفاده :

برای اینکه از این کلاس در هر دو پروژه (سرور و کلاینت) استفاده می کنیم باید یک Class Library جدید به نام Common بسازید و یک ارجاع از اون رو به هر دو پروژه سمت سرور و کلاینت بدید.
سرویس و Contract بالا رو به صورت زیر باز نویسی کنید.

```
[ServiceContract]
public interface IBookService
{
    [OperationContract]
    IEnumerable<Book> GetByExpression( XElement expression );
}
```

و سرویس :

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Xml.Linq;

namespace WcfLambdaExpression
{
    public class BookService : IBookService
    {
        public BookService()
        {
            ListOfBook = new List<Book>();
        }

        public List<Book> ListOfBook
        {
            get;
            private set;
        }

        public IEnumerable<Book> GetByExpression( XElement expression )
        {
            ListOfBook.AddRange( new Book[]
            {
                new Book(){Code = 1 , Title = "Book1"},
                new Book(){Code = 2 , Title = "Book2"},
                new Book(){Code = 3 , Title = "Book3"},
                new Book(){Code = 4 , Title = "Book4"},
                new Book(){Code = 5 , Title = "Book5"},
            } );

            Common.ExpressionSerializer serializer = new Common.ExpressionSerializer();

            return ListOfBook.AsQueryable().Where( serializer.Deserialize( expression ) as
            Expression<Func<Book, bool>> );
        }
    }
}

```

بعد از Build پروژه از روش Add Service Reference استفاده کنید و می بینید که بدون هیچ گونه مشکلی سرویس مورد نظر به پروژه Console اضافه شد. برای استفاده سمت کلاینت به صورت زیر عمل کنید.

```

using System;
using System.Linq.Expressions;
using TestExpression.MyBookService;

namespace TestExpression
{
    class Program
    {
        static void Main( string[] args )
        {
            BookServiceClient bookService = new BookServiceClient();

            Expression<Func<Book, bool>> expression = x => x.Code > 2 && x.Code < 5;

            Common.ExpressionSerializer serializer = new Common.ExpressionSerializer();

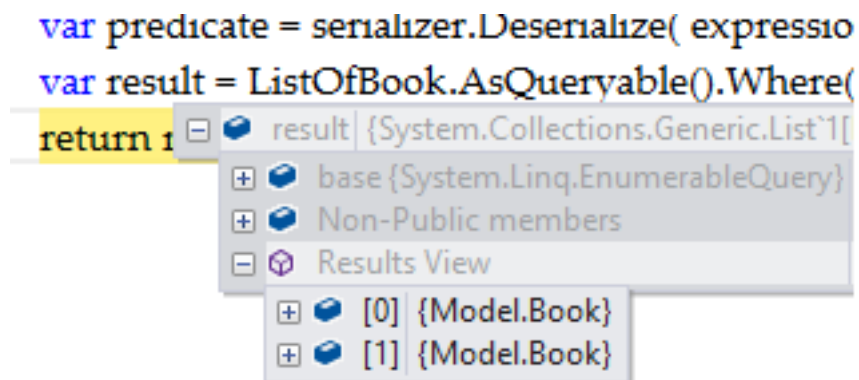
            bookService.GetByExpression( serializer.Serialize( expression ) );
        }
    }
}

```

بعد از اجرای پروژه، در سمت سرور خروجی های زیر رو مشاهده می کنیم.



خروجی هم به صورت زیر خواهد بود:



دریافت سورس کامل [Expression-Serialization](#)

نظرات خوانندگان

نویسنده: سابلنت
تاریخ: ۱۵:۱۱ ۱۳۹۲/۰۸/۰۲

بسیار عالیهِ . تازه شروع کردم به یادگیری WCF از مقالات شما نهایت استفاده رو بردم .

نویسنده: محمد
تاریخ: ۱۷:۱۶ ۱۳۹۲/۰۹/۱۹

سلام و ممنون از مقاله خوبتون، اما متأسفانه کلاس شما رو همیشه برای JSON استفاده نمود.

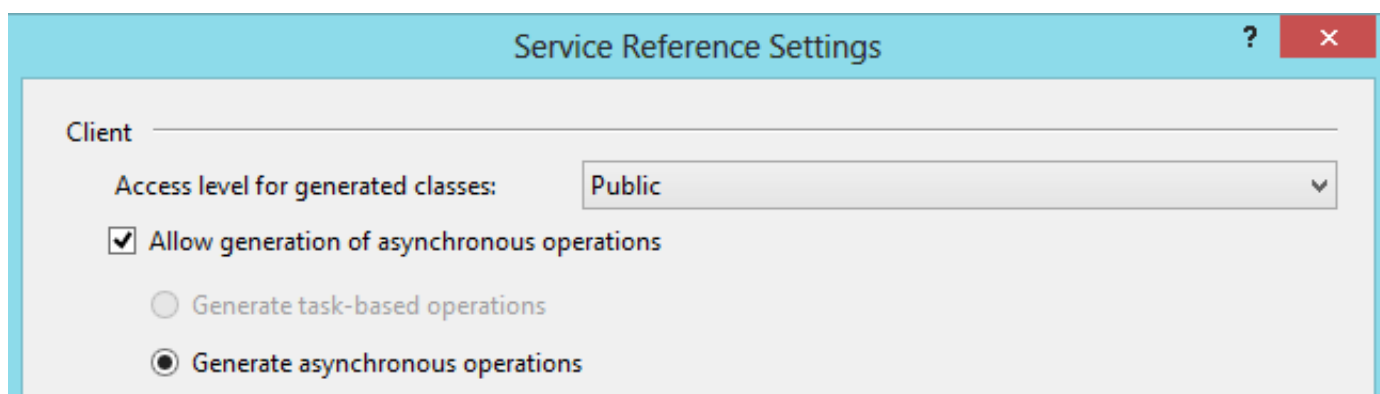
```
string json = JsonConvert.SerializeObject(serializer.Serialize(predicate3));  
predicate3 = JsonConvert.DeserializeObject<Expression<Func<Entity, bool>>>(json);
```

نویسنده: وحید نصیری
تاریخ: ۲۲:۵۸ ۱۳۹۲/۰۹/۱۹

- اینکار اضافی است. چون xml را تبدیل به json می‌کنید؛ بعد json را تبدیل به xml.
+ خروجی serializer.Serialize از نوع XElement است. بنابراین در قسمت آرگومان جنریک
JsonConvert.DeserializeObject باید XElement ذکر شود. مرحله بعدی آن فراخوانی serializer.Deserialize روی این
خروجی است.

```
Expression<Func<Book, bool>> expression = x => x.Code > 2 && x.Code < 5;  
var expressionSerializer = new Common.ExpressionSerializer();  
var xml = expressionSerializer.Serialize(expression);  
var xmlToJson = JsonConvert.SerializeObject(xml);  
var xmlObject = JsonConvert.DeserializeObject<XElement>(xmlToJson);  
var exp2 = expressionSerializer.Deserialize(xmlObject) as Expression<Func<Book, bool>>;
```

هنگام تولید و توسعه سیستم‌های مبتنی بر WCF حتما نیاز به سرویس هایی داریم که متدها را به صورت Async اجرا کنند. در دات نت 4.5 از Async&Await استفاده می‌کنیم (^). ولی در پروژه هایی که تحت دات نت 4 هستند این امکان وجود ندارد (البته می‌تونید Async&Await CTP رو برای دات نت 4 هم نصب کنید) (^). فرض کنید پروژه ای داریم تحت دات نت 3.5 یا 4 و قصد داریم یکی از متدهای سرویس WCF آن را به صورت Async پیاده سازی کنیم. ساده‌ترین روش این است که هنگام Add Service Reference از پنجره Advanced به صورت زیر عمل کنیم:



مهم‌ترین عیب این روش این است که در این حالت تمام متدهای این سرویس رو هم به صورت Sync و هم به صورت Async تولید می‌کنه در حالی که ما فقط نیاز به یک متد Async داریم .

در این پست قصد دارم پیاده سازی این متد رو بدون استفاده از Async&Await و Code Generation توکار دات نت شرح بدم که با دات نت 3.5 هم سازگار است.

ابتدا یک پروژه از نوع WCF Service Application ایجاد کنید.

یک ClassLibrary جدید به نام Model بسازید و کلاس زیر را به عنوان مدل در آن قرار دهید. (این اسمبلی باید هم به پروژه‌های کلاینت و هم به پروژه‌های سرور رفرنس داده شود)

```
[DataContract]
public class Book
{
    [DataMember]
    public int Code { get; set; }

    [DataMember]
    public string Title { get; set; }

    [DataMember]
    public string Author { get; set; }
}
```

حال پیاده سازی سرویس و Contract مربوطه را شروع می‌کنیم.

Class Library به نام Contract بسازید. قصد داریم از این لایه به عنوان قراردادهای سمت کلاینت و سرور استفاده کنیم. اینترفیس زیر را به عنوان BookContract در آن بسازید.

```
[ServiceContract]
public interface IBookService
{
    [OperationContract( AsyncPattern = true )]
    IAsyncResult BeginGetAllBook( AsyncCallback callback, object state );

    IEnumerable<Book> EndGetAllBook( IAsyncResult asyncResult );
}
```

برای پیاده سازی متدهای Async به این روش باید دو متد داشته باشیم. یکی به عنوان شروع عملیات و دیگری اتمام. دقت کنید نام گذاری به صورت Begin و End کاملاً اختیاری است و برای خوانایی بهتر از این روش نام گذاری استفاده می‌کنم. متدی که به عنوان شروع عملیات استفاده می‌شود باید حتماً OperationContractAttribute رو داشته باشد و مقدار خاصیت AsyncPattern اون هم true باشد. همان طور که می‌بیند این متد دارای 2 آرگومان ورودی است. یکی از نوع AsyncCallback و دیگری از نوع object. تمام متدهای Async به این روش باید این دو آرگومان ورودی را حتماً داشته باشند. خروجی این متد حتماً باید از نوع IAsyncResult باشد. متد دوم که به عنوان اتمام عملیات استفاده می‌شود نباید OperationContractAttribute را داشته باشد. ورودی اون هم فقط یک آرگومان از نوع IAsyncResult است. خروجی اون هم هر نوعی که سمت کلاینت احتیاج دارید می‌تونه باشه. در صورت عدم رعایت نکات فوق، هنگام ساخت ChannelFactory یا خطا روبرو خواهید شد. اگر نیاز به پارامتر دیگری هم داشتید باید آن‌ها را قبل از این دو پارامتر قرار دهید. برای مثال:

```
[OperationContract]
IEnumerable<Book> GetAllBook(int code , AsyncCallback callback, object state );
```

قبل از پیاده سازی سرویس باید ابتدا یک AsyncResult سفارشی بسازیم. ساخت AsyncResult سفارشی بسیار ساده است. کافی است کلاسی بسازیم که اینترفیس IAsyncResult را به ارث ببرد.

```
public class CompletedAsyncResult<TEntity> : IAsyncResult where TEntity : class , new()
{
    public IList<TEntity> Result
    {
        get
        {
            return _result;
        }
        set
        {
            _result = value;
        }
    }
    private IList<TEntity> _result;

    public CompletedAsyncResult( IList<TEntity> data )
    {
        this.Result = data;
    }

    public object AsyncState
    {
        get
        {
            return ( IList<TEntity> )Result;
        }
    }

    public WaitHandle AsyncWaitHandle
    {
        get
        {
            throw new NotImplementedException();
        }
    }

    public bool CompletedSynchronously
    {
        get
        {
            return true;
        }
    }
}
```



```

    public bool IsCompleted
    {
        get
        {
            return true;
        }
    }
}

```

در کلاس بالا یک خاصیت به نام Result در نظر گرفتیم که لیستی از نوع TEntity است. TEntity به صورت generic تعریف شده و نوع ورودی آن هر نوع کلاس غیر abstract می تواند باشد. این کلاس برای تمام سرویس های Async یک پروژه مورد استفاده قرار خواهد گرفت برای همین ورودی آن به صورت generic در نظر گرفته شده است.

#پیاده سازی سرویس

```

public class BookService : IBookService
{
    public BookService()
    {
        ListOfBook = new List<Book>();
    }

    public List<Book> ListOfBook
    {
        get;
        private set;
    }

    private List<Book> CreateListOfBook()
    {
        Parallel.For( 0, 10000, ( int counter ) =>
        {
            ListOfBook.Add( new Book()
            {
                Code = counter,
                Title = String.Format( "Book {0}", counter ),
                Author = "Masoud Pakdel"
            } );
        } );

        return ListOfBook;
    }

    public IAsyncResult BeginGetAllBook( AsyncCallback callback, object state )
    {
        var result = CreateListOfBook();
        return new CompletedAsyncResult<Book>( result );
    }

    public IEnumerable<Book> EndGetAllBook( IAsyncResult asyncResult )
    {
        return ( ( CompletedAsyncResult<Book> )asyncResult ).Result;
    }
}

```

*در متد BeginGetAllBook ابتدا به تعداد 10,000 کتاب در یک لیست ساخته می شوند و بعد این لیست در کلاس CompletedAsyncResult که ساختیم به عنوان ورودی سازنده پاس داده می شوند. چون CompletedAsyncResult ارث برده است پس return آن به عنوان خروجی مانعی ندارد. با فراخوانی متد EndGetAllBook سمت کلاینت مقدار asyncResult به عنوان خروجی برگشت داده می شود. به عملیات casting برای دستیابی به مقدار Result در CompletedAsyncResult دقت کنید.

#کدهای سمت کلاینت:

اکثر برنامه نویسان با استفاده از روش AddServiceReference یک سرویس کلاینت در اختیار خواهند داشت که با وهله سازی از این کلاس یک ChannelFactory ایجاد می شود. در این پست به جای استفاده از Code Generation توکار دات نت برای ساخت ChannelFactory از روش دیگری استفاده خواهیم کرد. به عنوان برنامه نویس باید بدانیم که در پشت پرده عملیات ساخت ChannelFactory چگونه است.

روش AddServiceReference

بعد از اضافه شدن سرویس سمت کلاینت کدهای زیر برای سرویس Book به صورت زیر تولید می شود.

```
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "4.0.0.0")]
public partial class BookServiceClient :
System.ServiceModel.ClientBase<UI.BookService.IBookService>, UI.BookService.IBookService {

    public BookServiceClient() {
    }

    public BookServiceClient(string endpointConfigurationName) :
        base(endpointConfigurationName) {
    }

    public BookServiceClient(string endpointConfigurationName, string remoteAddress) :
        base(endpointConfigurationName, remoteAddress) {
    }

    public BookServiceClient(string endpointConfigurationName, System.ServiceModel.EndpointAddress
remoteAddress) :
        base(endpointConfigurationName, remoteAddress) {
    }

    public BookServiceClient(System.ServiceModel.Channels.Binding binding,
System.ServiceModel.EndpointAddress remoteAddress) :
        base(binding, remoteAddress) {
    }

    public UI.BookService.Book[] BeginGetAllBook() {
        return base.Channel.BeginGetAllBook();
    }
}
```

همانطور که می بینید سرویس بالا از کلاس ClientBase ارث برده است. ClientBase دارای خاصیتی به نام ChannelFactory است که فقط خواندنی می باشد. با استفاده از مقادیر EndPointConfiguration یک وهله از کلاس ChannelFactory با توجه به مقدار generic کلاس ClientBase ایجاد خواهد شد. در کد زیر مقدار TChannel برابر IBookService است:

```
System.ServiceModel.ClientBase<UI.BookService.IBookService>
```

وهله سازی از ChannelFactory به صورت دستی

یک پروژه ConsoleApplication سمت کلاینت ایجاد کنید. برای فراخوانی متدهای سرویس سمت سرور باید ابتدا تنظیمات EndPoint رو به درستی انجام دهید. سپس با استفاده از EndPoint به راحتی می توانیم Channel مربوطه را بسازیم. کلاسی به نام ServiceMapper ایجاد می کنیم که وظیفه آن ساخت ChannelFactory به ازای درخواست ها است.

```
public class ServiceMapper<TChannel>
{
    public static TChannel CreateChannel()
    {
        TChannel proxy;

        var endPointAddress = new EndpointAddress( "http://localhost:7000/" + typeof( TChannel
).Name.Remove( 0, 1 ) + ".svc" );

        var httpBinding = new BasicHttpBinding();

        ChannelFactory<TChannel> factory = new ChannelFactory<TChannel>( httpBinding,
endPointAddress );

        proxy = factory.CreateChannel();

        return proxy;
    }
}
```

در متد CreateChannel یک وهله از کلاس EndpointAddress ایجاد شده است. پارامتر ورودی آن آدرس سرویس هاست شده می باشد برای مثال:

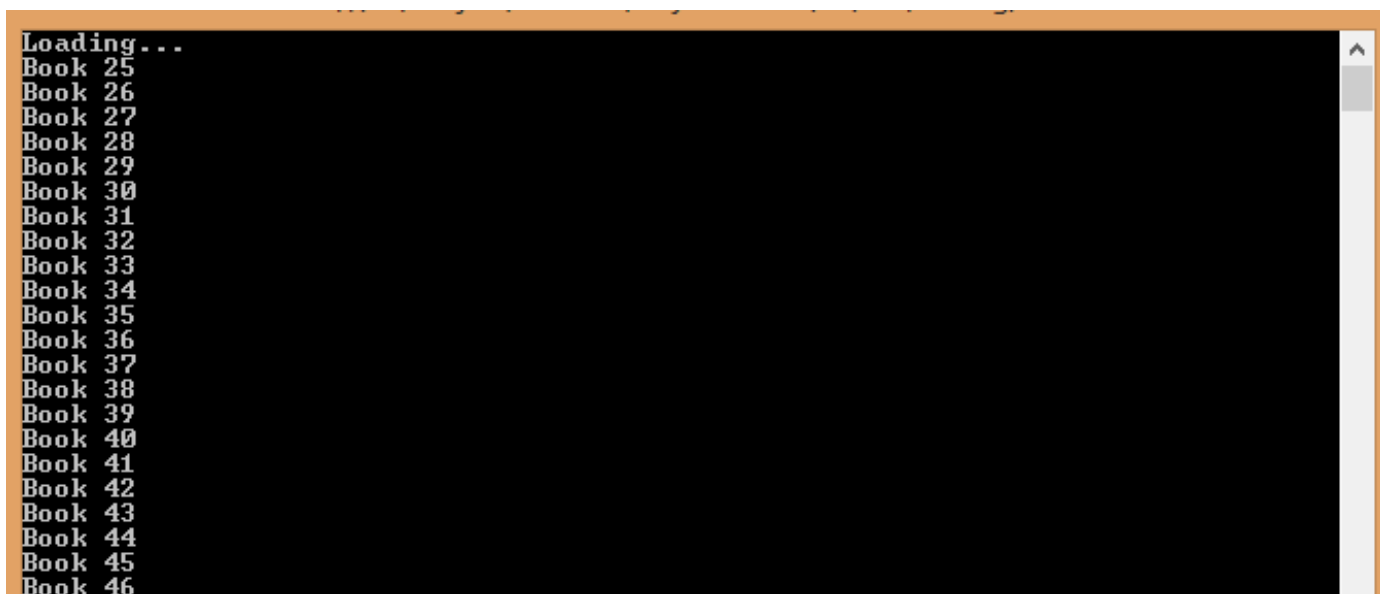
```
"http://localhost:7000/" + "BookService.svc"
```

دستور Remove برای حذف I از ابتدای نام سرویس است. پارامترهای ورودی برای سازنده کلاس ChannelFactory ابتدا یک نمونه از کلاس BasicHttpBinding می باشد. می توان از WSHttpBinding یا NetTcpBinding یا WSDLHttpBinding هم استفاده کرد. البته هر کدام از انواع Binding ها تنظیمات خاص خود را می طلبد که در مقاله ای جداگانه بررسی خواهیم کرد. پارامتر دوم هم EndPoint ساخته شده می باشد. در نهایت با دستور CreateChannel عملیات ساخت Channel به پایان می رسد.

بعد از اعمال تغییرات زیر در فایل Program پروژه Console و اجرای آن، خروجی به صورت زیر می باشد.

```
var channel = ServiceMapper<Contract.IBookService>.CreateChannel();
channel.BeginGetAllBook( new AsyncCallback( ( asyncResult ) =>
{
    channel.EndGetAllBook( asyncResult ).ToList().ForEach( _record =>
    {
        Console.WriteLine( _record.Title );
    } );
} ) , null );
Console.WriteLine( "Loading..." );
Console.ReadLine();
```

همان طور که می بینید ورودی متد BeginGtAllBook یک AsyncCallback است که در داخل آن متد EndGetAllBook صدا زده شده است. مقدار برگشتی متد EndGetAllBook خروجی مورد نظر ماست.
خروجی :



```
Loading...
Book 25
Book 26
Book 27
Book 28
Book 29
Book 30
Book 31
Book 32
Book 33
Book 34
Book 35
Book 36
Book 37
Book 38
Book 39
Book 40
Book 41
Book 42
Book 43
Book 44
Book 45
Book 46
```

نکته: برای اینکه مطمئن شوید که سرویس مورد نظر در آدرس "http://localhost:7000" هاست شده است (یعنی همان آدرسی که در EndPointAddress از آن استفاده کردیم) کافست از پنجره Project Properties برای پروژه سرویس وارد برگه Web شده و از بخش Servers گزینه Use Visual Studio Development Server و Specific Port 7000 رو انتخاب کنید.

نظرات خوانندگان

نویسنده:

هیمن روحانی

تاریخ:

۱۳۹۲/۱۰/۲۴ ۱۲:۳۱

سلام؛ من این مثال رو با دات نت 3.5 تست کردم. سمت کلاینت، channel فقط یک تابع به نام GetAllBook دارد و دو تابعی که در سرویس تعریف شده اند نمایش داده نمی شود؟

نویسنده:

مسعود پاکدل

تاریخ:

۱۳۹۲/۱۰/۲۴ ۱۳:۳۷

اگر از Add Service Reference برای اضافه کردن سرویس به کلاینت استفاده کردید باید از قسمت Advanced حتما تیک مربوط به Allow generation of asynchronous operation رو بزنید. ساخت متدهای Async در این روش به عهده code generator توکار دات است.

نویسنده:

هیمن روحانی

تاریخ:

۱۳۹۲/۱۰/۲۴ ۱۳:۴۸

جدا از روش Add Service Reference چه روش دیگه ای هست؟ پس تابع CreateFactory فقط از روی همین Service Reference یک نمونه می سازه؟

نویسنده:

مسعود پاکدل

تاریخ:

۱۳۹۲/۱۰/۲۴ ۱۴:۱۶

اگر از روش [ChannelFactory](#) استفاده کنید به دلیل دسترسی مستقیم به اسمبلی Service Contract تمام Operation Contract های تعریف شده در هر سرویس در دسترس خواهد بود. تابع CreateChannel با استفاده از تنظیمات Binding و EndpointAddress یک کانال به سرویس مربوطه خواهد ساخت و هیچ گونه نیازی به Add service reference در این روش نیست.

نویسنده:

هیمن روحانی

تاریخ:

۱۳۹۲/۱۰/۲۴ ۱۵:۰۲

یعنی برای استفاده از [ChannelFactory](#) باید به پروژه کلاینت اسمبلی Service Contract رو به عنوان reference اضافه کنم؟

نویسنده:

مسعود پاکدل

تاریخ:

۱۳۹۲/۱۰/۲۴ ۱۶:۱۸

بله. فقط به این نکته دقت داشته باشید که منظور از ServiceContract یعنی پروژه ای که فقط شامل اینترفیس هایی است که ServiceContractAttribute رو دارند. پیاده سازی این اینترفیس ها باید در یک پروژه دیگر برای مثال Service باشد که هیچ گونه رفرنسی به آن نیز نباید داده شود.

برای ساخت یک WCF Client یا دسترسی به یک سرویس WCF دو راه وجود دارد.

استفاده از WCF Proxy

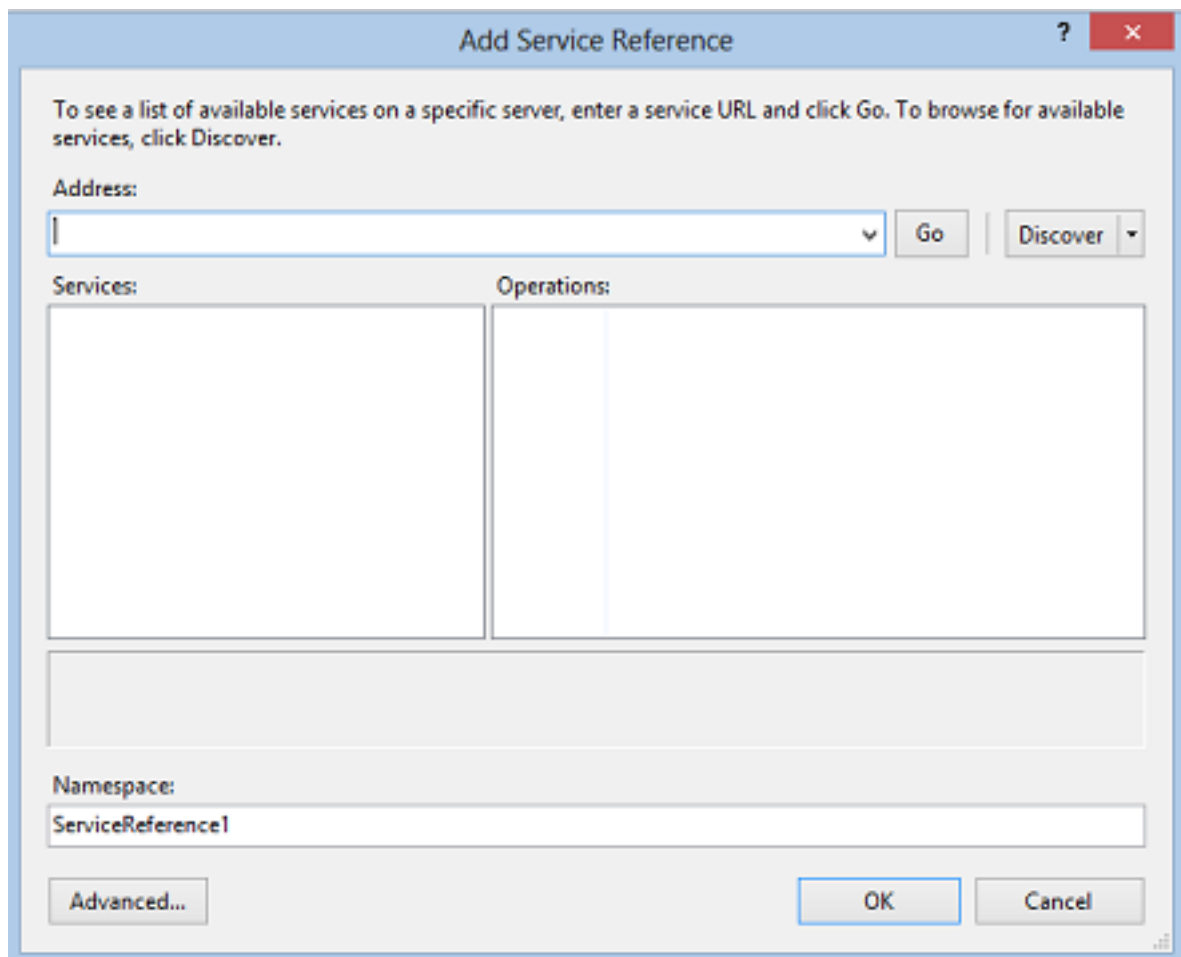
استفاده از ChannelFactory

قصد داریم طی یک مقایسه کوتاه این دو روش را بررسی کنیم:

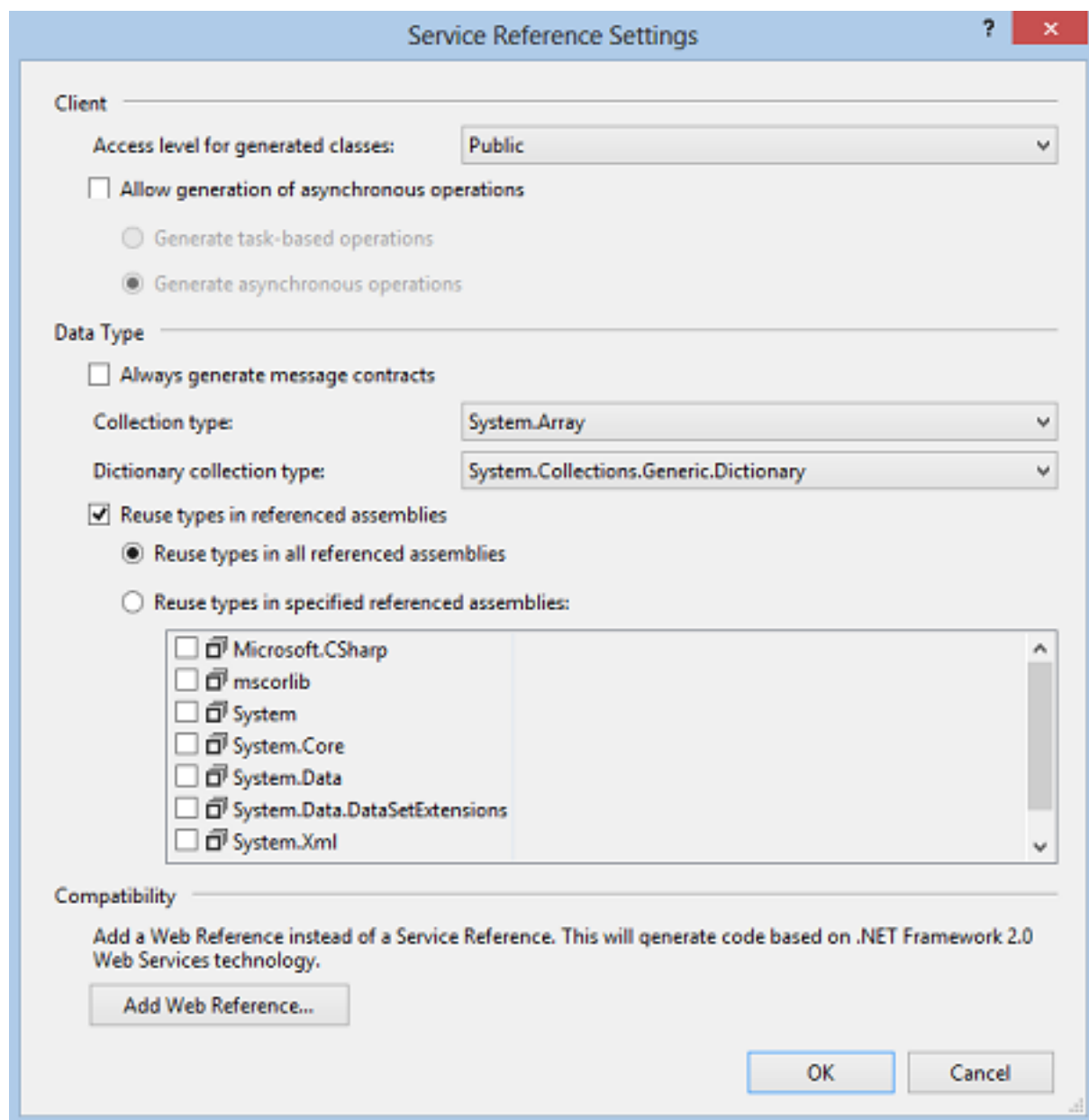
WCF Proxy:

Proxy در واقع یک کلاس CLR است که به عنوان نماینده یک اینترفیس که از نوع Service Contract است مورد استفاده قرار می‌گیرد. یا به زبان ساده تر، یک Proxy در واقع نماینده Service Contract ای که سمت سرور پیاده سازی شده است در سمت کلاینت خواهد بود. Proxy تمام متد یا Operation Contract های سمت سرور را داراست به همراه یک سری متدها و خواص دیگر برای مدیریت چرخه طول عمر سرویس، هم چنین اطلاعات مربوط به وضعیت سرویس و نحوه اتصال آن به سرور. ساخت Proxy به دو روش امکان پذیر است:

با استفاده از امکانات AddServiceReference موجود در Visual Studio. کافیسست از پنجره معروف زیر با استفاده از یک URL سرویس مورد نظر را به پروژه سمت کلاینت خود اضافه نمایید



همچنین می‌توانید از قسمت Advanced نیز برای تنظیمات خاص مورد نظر خود (مثل تولید کد برای متدهای Async یا تعیین نوع Collection ها در هنگام انتقال داده و ...) استفاده نمایید.



با استفاده از [SvcUtil.exe](#) . کاربرد در موارد Metadata Export, Service Validation, XmlSerialization Type و Generator و Metadata Download و ... خلاصه می‌شود. با استفاده از Vs.Net Command Prompt و svcutil می‌توان به سرویس مورد نظر دسترسی داشت. مثال

```
svcutil.exe /language:vb /out:generatedProxy.vb /config:app.config
http://localhost:8000/ServiceModelSamples/service
```

:ChannelFactory

ChannelFactory یک کلاس تعبیه شده در دات نت می‌باشد که به وسیله یک اینترفیس که به عنوان تعاریف سرویس سمت سرور است یک نمونه از سرویس مورد نظر را برای ما خواهد ساخت. اما به خاطر داشته باشید از این روش زمانی می‌توان استفاده کرد

که دسترسی کامل به سمت سرور و کلاینت داشته باشید.

برای آشنایی با نحوه پیاده سازی به این روش نیز می‌توانید از این [مقاله](#) کمک بگیرید.

مثال:

```
public static TChannel CreateChannel()
{
    IBookService service;

    var endPointAddress = new EndpointAddress( "http://localhost:7000/service.svc" );
    var httpBinding = new BasicHttpBinding();

    ChannelFactory<TChannel> factory = new ChannelFactory<TChannel>( httpBinding,
endPointAddress );

    instance= factory.CreateChannel();

    return instance;
}
```

همان طور که مشاهده می‌کنید در این روش نیاز به یک EndpointAddress به همراه یک نمونه از نوع Binding مورد نظر دارید. نوع این Binding حتما باید با نوع نمونه ساخته شده در سمت سرور که برای هاست کردن سرویس‌ها مورد استفاده قرار گرفته است یکی باشد. این نوع‌ها می‌تواند شامل MSMQ, WSDualHttpBinding, WSHttpBinding, BasicHttpBinding, NetTcpBinding و البته چند نوع دیگر نیز باشد.

در نتیجه برای ساخت یک سرویس به روش ChannelFactory باید مراحل زیر را طی نمایید:
یک نمونه از WCF Binding بسازید

یک نمونه از کلاس EndpointAddress به همراه آدرس سرویس مورد نظر در سمت سرور بسازید(البته می‌توان این مرحله را نادیده گرفت و آدرس سرویس را مستقیماً به نمونه ChannelFactory به عنوان پارامتر پاس داد)

یک نمونه از کلاس ChannelFactory یا استفاده از EndpointAddress بسازید
با استفاده از ChannelFactory یک نمونه از Channel مورد نظر را فراخوانی نمایید(فراخوانی متد CreateChannel)

تفاوت‌های دو روش

ChannelFactory	Proxy
شما نیاز به دسترسی مستقیم به اسمبلی حاوی Service Contract پروژه خود دارید.	فقط نیاز به یک URL برای ساخت سرویس مورد نظر دارد. بقیه مراحل توسط ابزارهای مرتبط انجام خواهد شد
پیاده سازی آن پیچیدگی بیشتر دارد	استفاده از این روش بسیار آسان و ساده است
نیاز به دانش اولیه از مفاهیم WCF برای پیاده سازی دارد	فهم مفاهیم این روش بسیار راحت است
زمانی که اطمینان دارید که مدل و entityها پروژه زیاد تغییر نخواهند کرد و از طرفی نیاز به کد نویسی کمتر در سمت کلاینت دارید، این روش موثرتر خواهد بود	زمانی که میزان تغییرات در کلاس‌های مدل و Entityها زیاد باشد این روش بسیار موثر است. (مدیریت تغییرات در WCF)
به تمام اینترفیس‌های تعریف شده در بخش Contracts دسترسی داریم.	فقط به اینترفیس‌هایی که دارای ServiceContractAttribute هستند دسترسی خواهیم داشت.
به تمام متدهای عمومی سرویس دسترسی داریم.	فقط به متدهای که دارای OperationContractAttribute هستند دسترسی خواهیم داشت.

آیا می‌توان از روش AddServiceReference تعبیه شده در Vs.Net، برای ساخت ChannelFactory استفاده کرد؟

بله! کافیت هنگام ساخت سرویس، در پنجره AddServiceReference از قسمت Advanced وارد برگه تنظیمات شوید. سپس تیک مربوط به قسمت های Reused Type in referenced assemblies و Reused Types in specified referenced assemblies را بزنید. بعد از لیست پایین، اسمبلی های مربوط به Domain Model و هم چنین Contract های سمت سرور را انتخاب نمایید. در این حالت شما از روش Channel Factory برای ساخت سرویس WCF استفاده کرده اید.

☒ Reuse types in referenced assemblies

☐ Reuse types in all referenced assemblies

☒ Reuse types in specified referenced assemblies:

<input type="checkbox"/>	Microsoft.CSharp
<input type="checkbox"/>	mscorlib
<input checked="" type="checkbox"/>	Service
<input type="checkbox"/>	System
<input type="checkbox"/>	System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
<input type="checkbox"/>	System.Data.DataSetExtensions

نظرات خوانندگان

نویسنده: افشار محبی
تاریخ: ۹:۴۳ ۱۳۹۲/۰۸/۰۶

استفاده از Channel Factory باعث می‌شود سورس کد پروژه بسیار تمیز و خلوت باشد. این موضوع اگر با سورس کنترل و محیط کار دسته جمعی همراه باشد خیلی تاثیر گذار خواهد بود چون Service Reference مقدار زیادی فایل تولید می‌کند. به علاوه با کمک Channel Factory امکان بعضی خودکار سازی‌ها هم بهتر فراهم خواهد بود.

در WCF به صورت پیش فرض متدها به صورت Request-Response هستند. این بدین معنی است که هر زمان درخواستی از سمت کلاینت به سرور ارسال شود تا زمانی که پاسخی از سمت سرور به کلاینت برگشت داده نشود، کلاینت منتظر خواهد ماند. برای مثال:

پروژه ای از نوع Wcf Service App می‌سازیم و یک سرویس با یک متد که خروجی آن نیز void است خواهیم داشت. به صورت زیر:

```
[ServiceContract]
public interface ISampleService
{
    [OperationContract]
    void Wait();
}
```

پیاده سازی Contract بالا:

```
public class SampleService : ISampleService
{
    public void Wait()
    {
        Thread.Sleep( new TimeSpan( 0, 1, 0 ) );
    }
}
```

در متد Wait، به مدت یک دقیقه اجرای برنامه سمت سرور را متوقف می‌کنیم. حال در یک پروژه از نوع Console App، سرویس مورد نظر را اضافه کرده و متد Wait آن را فراخوانی می‌کنیم. به صورت زیر:

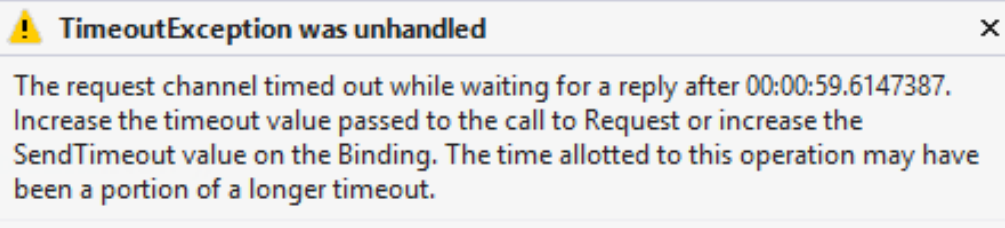
```
class Program
{
    static void Main( string[] args )
    {
        SampleService.SampleServiceClient client = new SampleService.SampleServiceClient();
        Console.WriteLine( DateTime.Now );
        client.Wait();
        Console.WriteLine( DateTime.Now );
        Console.ReadKey();
    }
}
```

همان طور که می‌بینید قبل از فراخوانی متد Wait زمان جاری سیستم را نمایش داده و سپس بعد از فراخوانی دوباره زمان مورد را نمایش می‌دهیم. در مرحله اول با خطای زیر مواجه خواهیم شد:

```
Console.WriteLine( DateTime.Now );
```

```
client.Wait();
```

```
Console.WriteLine( D
```



دلیل اینکه Timeout Exception پرتاب شد این است که به صورت پیش فرض مقدار خاصیت sendTimeout برابر 59 ثانیه است، در نتیجه قبل از اینکه پاسخی از سمت سرور به کلاینت برگشت داده شود این Exception رخ می‌دهد. برای حل این مشکل کفایت در فایل app.config در قسمت تنظیمات Binding، تغییر زیر را اعمال کنیم:

```
<basicHttpBinding>
  <binding name="BasicHttpBinding_ISampleService" sendTimeout="0:2:0"/>
</basicHttpBinding>
```

حال خروجی به صورت زیر است:

```
Before calling wait : 11/6/2013 9:58:29 PM
After calling wait : 11/6/2013 9:59:33 PM
```

مشخص است که تا زمانی که عملیات سمت سرور به پایان نرسد، (یا توجه به اینکه خروجی متد سمت سرور void است) اجرای برنامه در کلاینت نیز متوقف خواهد بود (اختلاف زمان‌های بالا کمی بیش از یک دقیقه است).

در این مواقع زمانی که باید متدی سمت سرور فراخوانی شود و قرار نیست که خروجی نیز در اختیار کلاینت قرار دهد بهتر است که از متدهای یک طرفه استفاده نماییم. متدهای یک طرفه یا به اصطلاح OneWay، هیچ پاسخی را به کلاینت برگشت نمی‌دهند و بلافاصله بعد از فراخوانی، کنترل اجرای برنامه را در اختیار کلاینت قرار خواهند داد. برای تعریف یک متد به صورت یک طرفه کفایت به صورت زیر عمل نماییم (مقدار خاصیت IsOneWay را در OperationContractAttribute برابر true خواهیم کرد):

```
[ServiceContract]
public interface ISampleService
{
    [OperationContract( IsOneWay = true )]
    void Wait();
}
```

حال اگر سرویس سمت کلاینت را به روز کرده و برنامه را اجرا کنیم خروجی به صورت زیر تغییر می‌کند:

```
Before calling wait : 11/6/2013 10:08:36 PM
After calling wait : 11/6/2013 10:08:40 PM
```

می‌بینید که اختلاف زمان‌های بالا در حد چند ثانیه است که آن هم صرفاً جهت فراخوانی متد سمت سرور بوده است. نکته مهم قابل ذکر این است که سرویس دهنده زمانی که با درخواستی جهت اجرای متد یک طرفه روبرو می‌شود، از آن جا که اجباری برای اجرای متد در همان زمان نیست در نتیجه این درخواست‌ها در بافر ذخیره می‌شوند و سپس در زمان مناسب اجرا خواهند شد. اگر بافر برای ذخیره اجرای متدهای یک طرفه پر باشد در این حالت کلاینت برای فراخوانی متدهای یک طرفه بعدی باید منتظر خالی شدن بافر سمت سرور بماند.

نظرات خوانندگان

نویسنده: reza110

تاریخ: ۱۴:۵۶ ۱۳۹۲/۰۸/۱۹

می‌خواستم بدانم اگر مثلاً به جای دستور Thread.Sleep در خواست در سمت سرور اجرای یک دستور روی دیتابیس باشد و به هر دلیلی ارتباط کلاینت قطع شود چگونه می‌توان ادامه کار سمت سرور را متوقف کرد. طبق بررسی روی task manager کرده ام حافظه مصرف شده همچنان افزایش می‌یابد.

نویسنده: مسعود پاکدل

تاریخ: ۱۵:۵۶ ۱۳۹۲/۰۸/۲۰

زمانی که ارتباط بین سرور و کلاینت به هر دلیلی قطع شود یا به بنا به دلایلی طی انجام عملیات سمت سرور Exception رخ دهد، وضعیت ChannelFactory برای آن سرویس به حالت Faulted تغییر پیدا می‌کند. در نتیجه دیگر امکان استفاده از این کانال ارتباطی میسر نیست و باید یک کانال ارتباطی جدید تهیه نمایید. اما برای اینکه بعد از متوقف سازی عملیات سمت سرور حافظه مصرفی دوباره بازگردانده شود باید از مفهوم UnitOfWork بهره جست البته با مقداری تغییرات برای همگام سازی با درخواست‌های WCF. روش مورد استفاده من به صورت زیر است (با فرض اینکه از EntityFramework به عنوان ORM استفاده میکنید):

« ابتدا یک Extension برای OperationContext تعریف می‌کنیم (با فرض اینکه IDbatabaseContext نماینده کلاس DbContext پروژه است):

```
private class OperationContainerExtension : IExtension<OperationContext>
{
    public OperationContainerExtension( IDbatabaseContext dbContext, string contextKey )
    {
        this.CurrentDbContext = dbContext;
        this.ContextKey = contextKey;
    }

    public IDbatabaseContext CurrentDbContext
    {
        get;
        private set;
    }

    public string ContextKey
    {
        get;
        private set;
    }

    public void Attach( OperationContext owner )
    {
    }

    public void Detach( OperationContext owner )
    {
    }
}
```

بعد در هنگام نمونه سازی از UnitOfWork در لایه سرویس Extension بالا به OperationContext جاری اضافه خواهد شد (اگر از IOC Container خاصی استفاده می‌کنید باید کد عملیات و هله سازی کلاس UnitOfWork را با کدهای زیر مرزین کنید):

```
if ( OperationContext.Current != null )
{
    OperationContext.Current.Extensions.Add( new OperationContainerExtension( dbContext ,
CONTEXTKEY ) );
    OperationContext.Current.OperationCompleted +=
CurrentOperationContext_OperationCompleted;
    OperationContext.Current.Channel.Faulted += Channel_Faulted;
}
```

می بینید که برای رویداد OperationCompleted و رویداد Fault در Channel نیز کدهای Dispose کردن UnitOfWork و همچنین DbContext را قرار دادم. به صورت زیر:

```
void Channel_Faulted( object sender, EventArgs e )
{
    IDatabaseContext dbContext = GetDbContext();
    if ( dbContext != null )
    {
        dbContext.Dispose();
        GC.Collect();
    }
}

private void CurrentOperationContext_OperationCompleted( object sender, EventArgs e )
{
    IDatabaseContext dbContext = GetDbContext();
    if ( dbContext != null )
    {
        dbContext.Dispose();
        GC.Collect();
    }
}
```

اگر به کدهای بالا دقت کنید متد GetDbContext نوشته شده برای به دست آوردن DbContext جاری در Session مربوطه است. کد آن نیز به صورت زیر می باشد

```
protected override IDatabaseContext GetDbContext()
{
    if ( OperationContext.Current != null )
    {
        var operationContainerExtension =
            OperationContext.Current.Extensions.OfType<OperationContainerExtension>().FirstOrDefault( e =>
                e.ContextKey == CONTEXTKEY );
        if ( operationContainerExtension != null )
        {
            return operationContainerExtension.CurrentDbContext;
        }
        return staticDbContext;
    }
    else
        return staticDbContext;
}
```

نکته آخر هم این است که CONTEXTKEY صرفاً یک فیلد از نوع string ولی با مقدار Guid برای به دست آوردن Extension مربوطه می باشد و تعریف آن در سازنده کلاس خواهد بود.

```
private string CONTEXTKEY = Guid.NewGuid().ToString();
```

در این صورت به طور قطع تمام منابع مورد استفاده سرویس های سمت سرور بعد از اتمام عملیات یا حتی وقوع هر خطا آزاد خواهند شد. اما اگر NHibernate را به عنوان ORM ترجیح می دهید به جای IDatabaseContext باید از ISession استفاده نمایید.

فرض کنید در حال توسعه یک سیستم مبتنی بر WCF هستید. بنابر نیاز باید یک سری اطلاعات مشخص در اکثر درخواست‌های بین سرور و کلاینت ارسال شوند یا ممکن است بعد از انجام بیش از 50 درصد پروژه این نیاز به وجود آید که یک یا بیش از یک پارامتر (که البته از سمت کلاینت تامین خواهند شد) در اکثر کوئری‌های گرفته شده سمت سرور شرکت داده شوند. خوب! در این وضعیت علاوه بر حس همدردی با اعضای تیم توسعه دهنده این پروژه چه می‌توان کرد؟

«اولین راه حلی که به ذهن می‌رسد این است که پارامترهای مشخص شده را در متدهای سرویس‌های مورد نظر قرار داد و به نوعی تمام سرویس‌ها را به روز رسانی کرد. این روش به طور قطع در خیلی از قسمت‌های پروژه به صورت مستقیم اثرگذار خواهد بود و در صورت نبود ابزارهای تست ممکن است با مشکلات جدی روبرو شوید.

«راه حل دوم این است که یک Message Header سفارشی بسازیم و در هر درخواست اطلاعات مورد نظر را در هدر قرار داده و سمت سرور این اطلاعات را به دست آوریم. این روش کمترین تغییر مورد نظر را برای پروژه دربر خواهد داشت و از طرفی نیاز متدهای سرویس به پارامتر را از بین می‌برد و دیگر نیازی نیست تا تمام متدهای سرویس‌ها دارای پارامترهای یکسان باشند.

پیاده سازی

برای شروع کلاس مورد نظر برای ارسال اطلاعات را به صورت زیر خواهیم ساخت:

```
[DataContract]
public class ApplicationContext
{
    [DataMember(IsRequired = true)]
    public string UserId
    {
        get { return _userId; }
        set
        {
            _userId = value;
        }
    }
    private string _userId;

    [DataMember(IsRequired = true)]
    public static ApplicationContext Current
    {
        get
        {
            return _current;
        }
        private set { _current = value; }
    }
    private static ApplicationContext _current;

    public static void Register( ApplicationContext appContext )
    {
        Current = appContext;
        IsRegistered = true;
    }
}
```

در این کلاس به عنوان نمونه مقدار Id کاربر جاری باید در هر درخواست به سمت سرور ارسال شود. حال نیاز به یک MessageInspector داریم ، کافیه که اینترفیس [IClientMessageInspector](#) را توسط یک کلاس به صورت زیر پیاده سازی نماییم:

```
public class ClientMessageHeaderInspector<T> : IClientMessageInspector
{
    private readonly T _vaccine;

    public ClientMessageHeaderInspector( T vaccine )
```

```

    {
        this._vaccine = vaccine;
    }

    public void AfterReceiveReply( ref Message reply, object correlationState )
    {
    }

    public object BeforeSendRequest( ref Message request, IClientChannel channel )
    {
        MessageHeader messageHeader = MessageHeader.CreateHeader( typeof( T ).Name, typeof( T
).Namespace, this._vaccine );
        request.Headers.Add( messageHeader );
        return null;
    }
}

```

نوع T مورد استفاده برای تعیین نوع داده ارسالی سمت سرور است که در این مثال کلاس ApplicationContext خواهد بود. در متد BeforeSendRequest باید Header سفارشی را ساخته و آن را به هدر درخواست اضافه نماییم. حال باید MessageInspector ساخته شده بالا را با استفاده از IEndPointBehavior به MessageInspectorهای نمونه ساخته شده از ClientRuntime اضافه نماییم. برای این کار به صورت زیر عمل می‌نماییم:

```

public class ApplicationContextMessageBehavior : IEndpointBehavior
{
    ClientMessageHeaderInspector<ApplicationContext> inspector = null;

    public ApplicationContextMessageBehavior()
    {
        inspector = new ClientMessageHeaderInspector<ApplicationContext>(
ApplicationContext.Current );
    }

    public void AddBindingParameters( ServiceEndpoint endpoint, BindingParameterCollection
bindingParameters )
    {
    }

    public void ApplyClientBehavior( ServiceEndpoint endpoint, ClientRuntime clientRuntime )
    {
        clientRuntime.MessageInspectors.Add( inspector );
    }

    public void ApplyDispatchBehavior( ServiceEndpoint endpoint, EndpointDispatcher
endpointDispatcher )
    {
    }

    public void Validate( ServiceEndpoint endpoint )
    {
    }
}

```

همان طور که می‌بینید در کلاس بالا یک نمونه از کلاس ClientMessageInspector را بر اساس ApplicationContext می‌سازیم و در متد ApplyClientBehavior به نمونه ClientRuntime اضافه می‌نماییم. اگر دقت کرده باشید می‌توان هر تعداد MessageInspector را به ClientRuntime اضافه کرد. در مرحله آخر باید تنظیمات مربوط به ChannelFactory را انجام دهیم.

```

public class ServiceMapper<TChannel>
{
    internal static EndpointAddress EPAddress
    {
        get
        {
            return _epAddress;
        }
    }
    private static EndpointAddress _epAddress;

    public static TChannel CreateChannel( Binding binding, string uriBase, string serviceName, bool
setCredential )
    {
    }
}

```



```

        _epAddress = new EndpointAddress( String.Format( "{0}{1}", uriBase, serviceName ) );
        var factory = new ChannelFactory<TChannel>( binding, _epAddress );
        ApplicationContext.Register( new ApplicationContext
        {
            UserId = Guid.NewGuid()
        } );

        factory.Endpoint.Behaviors.Add( new ApplicationContextMessageBehavior() );
        TChannel proxy = factory.CreateChannel();

        if ( factory.Endpoint.Behaviors.OfType<ApplicationContextMessageBehavior>().Any() )
        {
            using ( var scope = new OperationContextScope( ( IClientChannel )proxy ) )
            {
                OperationContext.Current.OutgoingMessageHeaders.Add( MessageHeader.CreateHeader(
                    typeof( ApplicationContext ).Name, typeof( ApplicationContext ).Namespace, ApplicationContext.Current )
                );
            }
        }
        return proxy;
    }

```

چند نکته:

«در متد CreateChannel، ابتدا تنظیمات مربوط به EndPointAddress و ChannelFactory انجام می‌شود. سپس یک نمونه از کلاس ApplicationContext را توسط متد Register به کلاس مورد نظر رجیستر می‌کنیم. به این ترتیب مقدار خاصیت Current در کلاس ApplicationContext برابر با نمونه ساخته شده می‌شود. سپس کلاس ApplicationContextMessageBehavior به خاصیت Behavior در ChannelFactory اضافه می‌شود. در انتها نیز هدر سفارشی ساخته شده به MessageHeaderهای نمونه جاری OperationContext اضافه می‌شود. این عمل توسط کد زیر انجام می‌گیرد:

```

OperationContext.Current.OutgoingMessageHeaders.Add( MessageHeader.CreateHeader( typeof(
ApplicationContext ).Name, typeof( ApplicationContext ).Namespace, AppConfig.Application ) );

```

از این پس هر درخواستی که از سمت کلاینت به سمت سرور ارسال شود به همراه خود یک نمونه از کلاس ApplicationContext را خواهد داشت. فقط دقت داشته باشید که برای ساخت ChannelFactory باید همیشه از متد CreateChannel استفاده نمایید.

استفاده از هدر سفارشی سمت سرور

حال قصد داریم که اطلاعات مورد نظر را از هدر درخواست در سمت سرور به دست آورده و از آن در کوئری‌های خود استفاده نماییم. کد زیر این کار را برای ما انجام می‌دهد:

```

if ( OperationContext.Current != null && OperationContext.Current.IncomingMessageHeaders.FindHeader(
    typeof( ApplicationContext ).Name , typeof( ApplicationContext ).Namespace ) > 0 )
{
    _application =
        OperationContext.Current.IncomingMessageHeaders.GetHeader<ApplicationContext>( typeof(
        ApplicationContext ).Name , typeof( ApplicationContext ).Namespace );
}

```

متد FindHeader در خاصیت IncomingMessageHeader با استفاده از نام و فضای نام به دنبال هدر سفارشی می‌گردد. اگر خروجی متد از 0 بیشتر بود یعنی هدر مورد نظر موجود است. در پایان نیز با استفاده از متد GetHeader، نمونه ساخته شده کلاس ApplicationContext را به دست می‌آوریم.

عنوان: WCF در MTOM
نویسنده: مسعود پاکدل
تاریخ: ۸:۱۵ ۱۳۹۲/۰۸/۲۴
آدرس: www.dotnettips.info
برچسب‌ها: WCF, MTOM, MessageBinding

در WCF سه نوع Message Encoder وجود دارد:

Text(Xml) Message Encoder (به صورت پیش فرض در تمام Http-Base Binding ها از این Encoder استفاده می‌شود)

Binary Message Encoder (به صورت پیش فرض در تمام Net* Binding ها از این encoder استفاده می‌شود که برای سرویس‌های وب مناسب نیست)

MTOM Message Encoder (در حالت استفاده از Http-Base Binding ها و انتقال اطلاعات به صورت باینری از این گزینه استفاده می‌شود که به صورت پیش فرض غیر فعال است)
Encoding یا رمزگذاری در WCF به این معنی است که داده‌های مورد نظر برای انتقال، به یکی از فرمت‌های Text-Xml ، MTOM یا Binary سریالایز شوند.

وضعیتی را در نظر بگیرید که در یک پروژه مبتنی بر WCF قصد دارید حجمی زیاد از داده به فرمت باینری (نظیر فایل ها) را بین سرور و کلاینت رد و بدل کنید. به صورت معمول بسیاری از برنامه نویسان، یک کلاس به همراه DataContractAttribute ایجاد می‌کنند که در آن خاصیتی به صورت آرایه ای از نوع بایت که DataMemberAttribute را نیز دارد برای انتقال محتویات فایل استفاده می‌شود. اما باید یک نکته را مد نظر داشت و آن این است که به صورت پیش فرض فرمت انتقال داده‌ها در WCF به صورت Text/Xml است و برای انتقال داده‌ها نیز از فرمت [Base 64](#) استفاده خواهد شد. مشکل اصلی این جاست که در حالت Text/Xml Encoding برای انتقال داده‌های باینری، برای هر سه بایت، چهار کاراکتر استفاده می‌شود در نتیجه، این باعث افزایش حجم داده تا 33 درصد خواهد شد که کارایی سیستم را تحت تاثیر قرار می‌دهد.

اما خبر خوب این است که استاندارد وجود دارد به نام **MTOM** یا همان **M essage T ransmission O ptimization M echanism**، برای این که بتوان محتوای باینری را بدون افزایش حجم داده انتقال داد. برای پیاده سازی این روش باید موارد زیر را در نظر داشته باشید:

«متد یا همان OperationContract که وظیفه آن ارسال یا دریافت داده‌ها با فرمت MTOM است فقط کلاس هایی را انتقال دهد که دارای MessageContractAttribute هستند. نباید از DataContractAttribute استفاده نمایید.
«خاصیتی که نوع آن آرایه ای از بایت‌ها است نباید دارای DataMemberAttribute باشد؛ بلکه به جای آن باید از MessageBodyMember استفاده نمایید.

«به جای `Byte[]` می‌توان از نوع `Stream` نیز استفاده کرد(الزامی نیست).
«مقدار خاصیت `MessageEncoding` در `Binding` استفاده شده باید MTOM تعیین شود.

پیاده سازی یک مثال

ابتدا کلاس مورد نظر را به صورت زیر تهیه می‌کنیم:

```
[MessageContract]
public class MyFile
{
    [MessageHeader]
    public String Filename { get; set; }

    [MessageBodyMember]
    public Byte[] Contents { get; set; }
}
```

چند تذکر

به جای DataContract از MessageContract استفاده می‌شود؛
 تمام خاصیت‌هایی که نوع آن‌ها غیر از Byte[] است باید دارای MessageHeader باشند؛
 خاصیتی که برای انتقال محتوای باینری تهیه شده است، باید از MessageBodyMember استفاده نماید؛
 مجاز به تعریف فیلد یا فیلدهایی که نوع آن‌ها Primitive Type است نمی‌باشید.

تنظیمات مربوط به Binding نیز به صورت خواهد بود:

```
<bindings>
  <wsHttpBinding>
    <binding name="WsHttpMtomBinding" messageEncoding="Mtom" />
  </wsHttpBinding>
</bindings>
```

اما یک نکته...

هدف از استفاده از MTOM برای افزایش کارایی انتقال داده‌های باینری در حجم زیاد است. در زیر نتایج مقایسه بررسی انتقال اطلاعات به دو صورت MTOM و Text برای حجم داده‌های متفاوت را مشاهده می‌کنید:

```
Text encoding with a 100 byte payload: 433
MTOM encoding with a 100 byte payload: 912

Text encoding with a 1000 byte payload: 1633
MTOM encoding with a 1000 byte payload: 2080

Text encoding with a 10000 byte payload: 13633
MTOM encoding with a 10000 byte payload: 11080

Text encoding with a 100000 byte payload: 133633
MTOM encoding with a 100000 byte payload: 101080

Text encoding with a 1000000 byte payload: 1333633
MTOM encoding with a 1000000 byte payload: 1001080
```

با دقت در نتایج بالا مشخص می‌شود که این روش در حجم داده‌های پایین (مثل 100 بایت یا 1000 بایت) عملکرد مورد انتظار را نخواهد داشت. پس این نکته را نیز در هنگام پیاده‌سازی به این روش مد نظر داشته باشید.

حالتی را در نظر بگیرید که سرویس های یک برنامه در آدرسی مشخص هاست شده اند. اگر اعتبار سنجی برای این سرویس ها در نظر گرفته نشود به راحتی می توان با در اختیار داشتن آدرس مورد نظر تمام سرویس های برنامه را فراخوانی کرد و اگر رمزگذاری اطلاعات بر روی سرویس ها فعال نشده باشد می توان تمام اطلاعات این سرویس ها را به راحتی به دست آورد. کمترین تلاش در این مرحله برای پیاده سازی امنیت این است که برای فراخوانی هر سرویس حداقل یک شناسه و رمز عبور چک شود و فقط در صورتی که فراخوانی سرویس همراه با شناسه و رمز عبور درست بود اطلاعات در اختیار کلاینت قرار گیرد. قصد داریم طی یک مثال این مورد را بررسی کنیم:

ابتدا یک پروژه با دو Console Application با نام های Service و Client ایجاد کنید. سپس در پروژه Service یک سرویس به نام BookService ایجاد کنید و کدهای زیر را در آن کپی نمایید:

Contract مربوطه به صورت زیر است:

```
[ServiceContract]
public interface IBookService
{
    [OperationContract]
    int GetCountOfBook();
}
```

کدهای مربوط به سرویس:

```
[ServiceBehavior(IncludeExceptionDetailInFaults = true)]
public class BookService : IBookService
{
    public int GetCountOfBook()
    {
        return 10;
    }
}
```

فایل Program در پروژه Service را باز نمایید و کدهای زیر را که مربوط به hosting سرویس مورد نظر است در آن کپی کنید:

```
class Program
{
    static void Main(string[] args)
    {
        ServiceHost host = new ServiceHost(typeof(BookService));

        var binding = new BasicHttpBinding();

        host.AddServiceEndpoint(typeof(IBookService), binding, "http://localhost/BookService");
        host.Open();

        Console.WriteLine("BookService host");

        Console.ReadKey();
    }
}
```

بر اساس کدهای بالا، سرویس BookService در آدرس http://localhost/BookService هاست می شود. نوع Binding نیز BasicHttpBinding انتخاب شده است.

حال نوبت به پیاده سازی سمت کلاینت می رسد. فایل Program سمت کلاینت را باز کرده و کدهای زیر را نیز در آن کپی نمایید:

```
static void Main(string[] args)
{
    Thread.Sleep(2000);
```

```

        BasicHttpBinding binding = new BasicHttpBinding();
        ChannelFactory<IBookService> channel = new ChannelFactory<IBookService>(binding, new
EndpointAddress("http://localhost/BookService"));
        Console.WriteLine("Count of book: {0}", channel.CreateChannel().GetCountOfBook());
        Console.ReadKey();
    }

```

در کدهای عملیات ساخت ChannelFactory برای برقراری اطلاعات با سرویس مورد نظر انجام شده است. پروژه را Build نمایید و سپس آن را اجرا کنید. خروجی زیر مشاهده می‌شود:



تا اینجا هیچ گونه اعتبارسنجی انجام نشد. برای پیاده سازی اعتبارسنجی باید یک سری تنظیمات بر روی Binding و Hosting سمت سرور و البته کلاینت برقرار شود. فایل Program پروژه Service را باز نمایید و محتویات آن را به صورت زیر تغییر دهید:

```

static void Main(string[] args)
{
    ServiceHost host = new ServiceHost(typeof(BookService));

    var binding = new BasicHttpBinding();
    binding.Security = new BasicHttpSecurity();
    binding.Security.Mode = BasicHttpSecurityMode.TransportCredentialOnly;
    binding.Security.Transport.ClientCredentialType = HttpClientCredentialType.Basic;

    host.Credentials.UserNameAuthentication.UserNamePasswordValidationMode =
System.ServiceModel.Security.UserNamePasswordValidationMode.Custom;

    host.Credentials.UserNameAuthentication.CustomUserNamePasswordValidator = new
CustomUserNamePasswordValidator();

    host.AddServiceEndpoint(typeof(IBookService), binding, "http://localhost/BookService");
    host.Open();

    Console.WriteLine("BookService host");
    Console.ReadKey();
}

```

تغییرات اعمال شده:

ابتدا نوع Security در Binding را به حالت TransportCredentialOnly تنظیم کردیم. در یک جمله هیچ گونه تضمینی برای صحت اطلاعات انتقالی در این حالت وجود ندارد و فقط یک اعتبارسنجی اولیه انجام خواهد شد. در نتیجه هنگام استفاده از این حالت باید با دقت عمل نمود و نباید فقط به پیاده سازی این حالت اکتفا کرد. (Encryption اطلاعات سرویس‌ها مورد بحث این پست نیست)

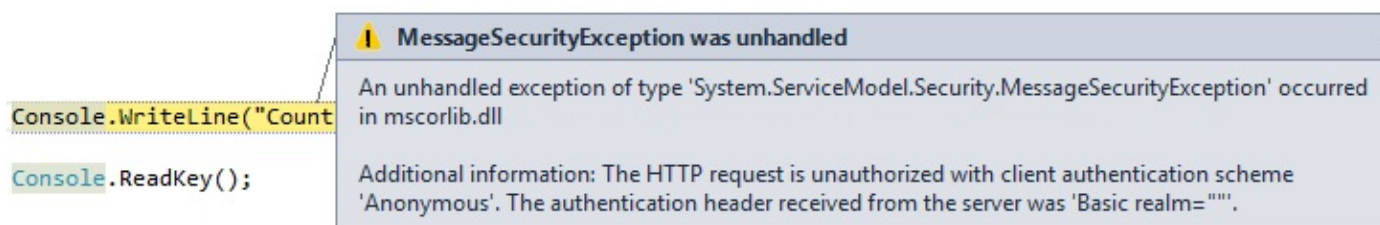
ClientCredentialType نیز باید به حالت Basic تنظیم شود. در WCF اعتبارسنجی به صورت پیش فرض در حالت Windows است (یعنی UserNamePasswordValidationMode برابر مقدار Windows است و اعتبارسنجی بر اساس کاربر انجام می‌شود). این مورد باید به مقدار Custom تغییر یابد. در انتها نیز باید مدل اعتبارسنجی دلخواه خود را به صورت زیر پیاده سازی کنیم: در پروژه سرویس یک کلاس به نام CustomUserNamePasswordValidator بسازید و کدهای زیر را در آن کپی کنید:

```
public class CustomUserNamePasswordValidator : UserNamePasswordValidator
{
    public override void Validate(string userName, string password)
    {
        if (userName != "Masoud" || password != "Pakdel")
            throw new SecurityException("Incorrect userName or password");
    }
}
```

Validator مورد نظر از کلاسی abstract به نام UserNamePasswordValidator ارث می‌برد، در نتیجه باید متد abstract به نام Validate را override نماید. در بدنه این متد شناسه و رمز عبور با یک مقدار پیش فرض چک می‌شوند و در صورت عدم درستی این پارامترها یک استثنا پرتاب خواهد شد.

تغییرات مورد نیاز سمت کلاینت:

اگر در این حالت پروژه را اجرا نمایید از آن جا که از این به بعد، درخواست‌ها سمت سرور اعتبار سنجی می‌شوند در نتیجه با خطای زیر روبرو خواهید شد:



این خطا از آن جا ناشی می‌شود که تنظیمات کلاینت و سرور از نظر امنیتی با هم تناسب ندارد. در نتیجه باید تنظیمات Binding کلاینت و سرور یکی شود. برای این کار کد زیر را به فایل Program سمت کلاینت اضافه می‌کنیم:

```
static void Main(string[] args)
{
    Thread.Sleep(2000);
    BasicHttpBinding binding = new BasicHttpBinding();

    binding.Security = new BasicHttpSecurity();
    binding.Security.Mode = BasicHttpSecurityMode.TransportCredentialOnly;
    binding.Security.Transport.ClientCredentialType = HttpClientCredentialType.Basic;

    ChannelFactory<IBookService> channel = new ChannelFactory<IBookService>(binding, new
    EndpointAddress("http://localhost/BookService"));

    channel.Credentials.UserName.UserName = "WrongUserName";
    channel.Credentials.UserName.Password = "WrongPassword";

    Console.WriteLine("Count of book: {0}", channel.CreateChannel().GetCountOfBook());

    Console.ReadKey();
}
```

توسط دستور زیر، مقدار شناسه و رمز عبور به درخواست اضافه می‌شود.

```
channel.Credentials.UserName.UserName = "WrongUserName";
channel.Credentials.UserName.Password = "WrongPassword";
```

در اینجا Username و Password اشتباه مقدار دهی شده اند تا روش کار Validator مورد بررسی قرار گیرد. حال اگر پروژه را اجرا نمایید خواهید دید که در Validator مورد نظر، عملیات اعتبار سنجی به درستی انجام می شود:



[دریافت سورس مثال بالا](#)

نظرات خوانندگان

نویسنده:

بهمن

تاریخ:

۱۱:۵۲ ۱۳۹۲/۱۰/۲۱

سلام. ممنون به خاطر زحماتتون.

بر طبق آموزشهای گوناگون برای اعمال امنیت روی سرویس میتوان از Certificate هایی استفاده کرد که خودمان آنها را تولید کرده ایم. البته سفارش شده که در زمان برنامه نویسی و پیاده سازی پروژه از آن استفاده شود نه برای زمان واقعی استفاده از سرویس.

آیا این امکان وجود دارد که از Certificate هایی که خودمان ایجاد کرده ایم در پروژه های واقعی استفاده کنیم؟ اگر این امکان وجود دارد آیا این Certificate ها کار رمز گزاری و رمز گشایی را برای ما انجام میدهند؟ و چه محدودیتهایی دارند؟ با تشکر؟

نویسنده:

مسعود پاکدل

تاریخ:

۱۲:۴۵ ۱۳۹۲/۱۰/۲۱

اگر به مثال بالا دقت کرده باشید حتما متوجه شدید که از BasicHttpBinding استفاده کردم. دلیل این موضوع این است که BasicHttpBinding به صورت پیش فرض هیچ گونه تمهیدات امنیتی را بر روی سرویس ها در نظر نمی گیرد. اگر قصد پیاده سازی مثال بالا را به وسیله WSHttpBinding (این binding به صورت توکار مباحث رمزگذاری و امضای دیجیتال را در خود دارد) داشته باشیم حتما باید از Certificate ها بهره ببریم. در نتیجه برای پیاده سازی مثال بالا به روش WsHttpBinding از [makecert.exe](#) برای تولید certificate ها استفاده می شد (عموما در مثال ها و نمونه ها از همین روش استفاده میشود) که در اجرای واقعی سرویس ها مناسب نیست. در [Soap](#) این Certificate ها شامل اطلاعات رمزگذاری و مجوزها و کلیدهای عمومی و خصوصی خواهند بود در نتیجه از اهمیت به سزایی برخوردارند. برای حفظ امنیت سرویس ها توصیه می شود certificate ها را از یک CA (برای مثال [VeriSign](#)) خریداری شود یا حداقل می توانید از Microsoft Certificate Services که در ویندوزهای سرور نصب می شود استفاده نمایید. در واقع اگر یک Certificate Authority وجود نداشته باشد بهتر است از این روش استفاده نشود.

نویسنده:

مهرسا

تاریخ:

۱۳:۱۵ ۱۳۹۲/۱۲/۰۵

سلام؛ من کدهای شمارو امتحان کردم ولی در کلاینت من نمیتونم اینو پیدا کنم channel.Credentials برای من اینو داره channel.ClientCredentials هر چی هم گشتم نتونستم پیداش کنم میگه کلاس Credentials وجود نداره

نویسنده:

مسعود پاکدل

تاریخ:

۱۴:۲ ۱۳۹۲/۱۲/۰۵

Credential خود یک property از نوع ClientCredential در نمونه های وهله سازی شده از ChannelFactory است. شما از روش Add Service Reference و proxy استفاده کرده اید در نتیجه ChannelFactory به صورت یک خاصیت در نمونه وهله سازی شده از client proxy در دسترس است. به صورت زیر عمل نمایید:

```
proxy.ChannelFactory.Credentials.UserName.UserName = "WrongUserName";
proxy.ChannelFactory.Credentials.UserName.Password = "WrongPassword";
```

در همین رابطه : [مقایسه بین روش Proxy و ChannelFactory](#)

نویسنده: مهرسا
تاریخ: ۱۱:۱۹ ۱۳۹۲/۱۲/۰۶

مرسی از جوابتون

امکان ست کردن تنظیمات سرور در وب کانفیگ هم هست؟ چون من سرویسو در یک وب سایت گذاشتم.

نویسنده: مسعود پاکدل
تاریخ: ۱۴:۱۵ ۱۳۹۲/۱۲/۰۶

بله. می‌توانید تمام تنظیمات را در فایل config قرار دهید. برای نمونه:

```
<behaviors>
  <serviceBehaviors>
    <behavior name="yourServiceNameBehavior">
      <serviceDebug includeExceptionDetailInFaults ="true"/>
      <serviceCredentials>
        <userNameAuthentication userNamePasswordValidationMode="Custom"
customUserNamePasswordValidatorType="MyCustomUserNameValidator, service" />
      </serviceCredentials>
    </behavior>
  </serviceBehaviors>
</behaviors>

</system.serviceModel>
```

در صورتی که از certificate ها استفاده کرده اید آن را هم باید به صورت زیر در این بخش قرار دهید:

```
<serviceCertificate findValue="localhost" storeLocation="LocalMachine" storeName="My"
x509FindType="FindBySubjectName" />
```

عنوان: تغییر فضای نام کلاس poco استفاده شده در wcf و از کار افتادن برنامه‌ی مشتری بدون دریافت پیام خطا

نویسنده: ابوالفضل رجب پور

تاریخ: ۱۱:۳۵ ۱۳۹۲/۱۰/۲۴

آدرس: www.dotnettips.info

گروه‌ها: ASP.Net, WCF, namespace

چند وقت پیش در پروژه‌ای یک سرویس WCF داشتم که اطلاعاتی را در قالب یک کلاس poco برگشت می‌داد. اخیرا بعد از اصلاحاتی در پروژه متوجه شدم که سرویس کار نمی‌کند. هیچ خطایی هم وجود نداشت. شروع به دیباگ کردم و متوجه شدم که سرویس برنامه اطلاعات را برگشت می‌دهد، اما برنامه‌ی مشتری تعداد اطلاعات دریافتی را صفر اعلام میکند و هیچ خطایی هم گزارش نمی‌شود.

چون اطلاعات در قالب باینری در قسمتی از کلاس poco برگشت می‌شد، ابتدا حدسم حجم فایل بود. اطلاعات کلاس poco:

```
public class OutgoingJob
{
    public int Id;
    public string JobId;
    public string Subject;
    public string Reciver;
    public byte[][] Attachments;
}
```

تنظیمات سایز ارسال و دریافت رو به حداکثر رسوندم. هیچ فایده‌ای نداشت. برنامه‌ی مشتری به راحتی به سرویس وصل می‌شد و با سایر متدهایی که خروجی‌های تایپ‌های اصلی مثل bool و string را برمی‌گرداند کار می‌کرد. فقط با متدی که لیست poco داشت، تعداد لیست اطلاعات دریافتی 0 اعلام می‌شد. متد WCF برای برگشت اطلاعات و لاگ کردن وقایع:

```
public List<OutgoingJob> GetJobsList(int Count)
{
    LogEvent("GetFaxsList Start...");

    List<OutgoingJob> OutgoingJobs = new List<OutgoingJob>();

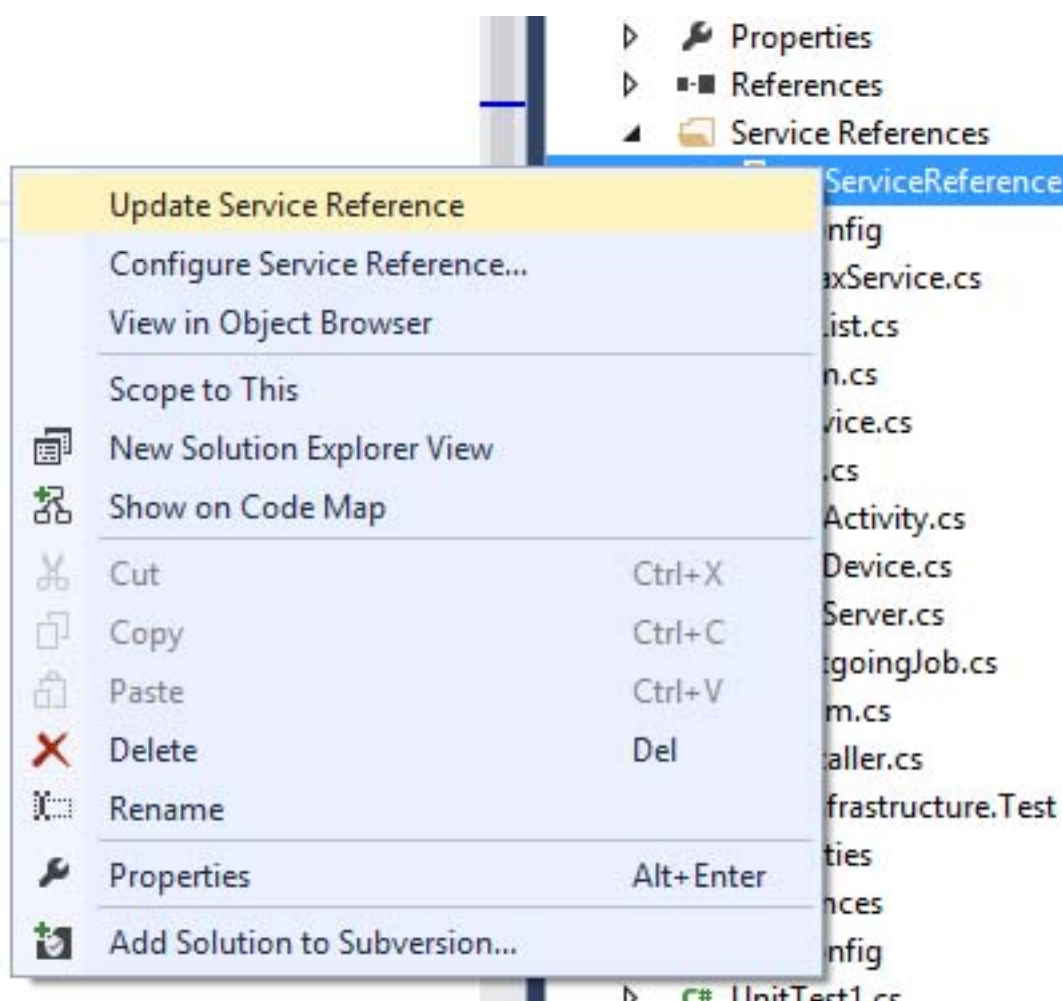
    // business for fill list

    LogEvent("return job Count = " + OutgoingJobs.Count);
    return OutgoingJobs;
}
```

```
[TestMethod]
public void TestMethod1()
{
    jobService ser = new jobService();
    var listjob = ser.GetJobsList(5);
    Assert.AreNotEqual(0, listjob.Count);
}
```

لاگ‌های متد WCF، تعداد را 1 اعلام می‌کند، اما تست، نتیجه را صفر برمی‌گرداند. بعد از کلی کلنجار با تنظیمات binding و serviceBehaviors متوجه شدم که اشکال کار به نکته‌ی کوچک خیلی خیلی ساده است. من هیچ تغییری در کلاس‌ها نداده بودم، اما برای مدیریت بهتر پروژه، فضای نام کلاس‌ها را تغییر داده بودم و مسبب همه‌ی مشکلات و وقت‌کشی‌ها همین بود.

راه حل هم که ساده ست. هنگامی که فضای نام کلاس‌های برگشتی را تغییر می‌دهید، حتما باید **update service reference** را در برنامه‌ی مشتری اجرا کنید تا اطلاعات سرویس بروز شود.



مشکل و راه حل خیلی ساده بود، ولی از من که خیلی وقت گرفت. امیدوارم وقت دوستان مثل من هدر نره
موفق باشید

نظرات خوانندگان

نویسنده:

محسن خان

تاریخ:

۱۱:۴۱ ۱۳۹۲/۱۰/۲۴

استفاده از روش دیگر اتصال به سرور هم می‌تونه کمک کنه: [مقایسه بین Proxy و ChannelFactory در WCF](#)

نویسنده:

افشار محبی

تاریخ:

۱۲:۵ ۱۳۹۲/۱۰/۲۴

بله کاملاً درسته. روش Channel Factory می‌تونه کمک خیلی خوبی به کاهش این گونه مشکلات بکند.

نویسنده:

ابوالفضل رجب پور

تاریخ:

۱۲:۲۸ ۱۳۹۲/۱۰/۲۴

با تشکر از ارجاع خوبتون.

مطلب اینجا راجع به یک نکته و روش رفعش در روش اول بود. مزایا و معایب هر روش سر جای خود محفوظ

در این مثال برای اینکه Instance Provider سفارشی خود را بتوانیم به عنوان یک Behavior به سرویس اضافه نماییم باید به خاصیت Description.Behaviors شی ServiceHost دسترسی داشته باشیم. زمانی که در پروژه‌های WCF از روش Self Hosting برای هاست سرویس‌ها استفاده کنیم به دلیل دسترسی مستقیم به شی ServiceHost هر گونه تنظیمات و عملیات Customization به راحتی امکان پذیر است؛ اما در IIS Hosting، از آن جا که به صورت پیش فرض از ServiceHostFactory موجود در WCF استفاده می‌شود ما دسترسی به شی ServiceHost نداریم. برای حل این مسئله باید یک CustomServiceHostFactory ایجاد نماییم که به راحتی در WCF این امکان تدارک دیده شده است.

بررسی یک مثال:

ابتدا کلاسی به صورت زیر ایجاد نمایید. در این کلاس می‌توانید کدهای لازم برای سفارشی کردن شی ServiceHost را قرار دهید:

```
public class CustomServiceHost : ServiceHost
{
    public CustomServiceHost( Type t, params Uri baseAddresses ) :
        base( t, baseAddresses ) {}

    public override void OnOpening()
    {
        this.Description.Add( new MyServiceBehavior() );
    }
}
```

اگر از این به بعد به جای استفاده از ServiceHost مستقیماً از CustomServiceHost استفاده نماییم، MyServiceBehavior به صورت خودکار به عنوان یک ServiceBehavior برای سرویس مورد نظر در نظر گرفته می‌شود. برای این که هنگام هاست سرویس مورد نظر به صورت خودکار از این شی کلاس استفاده شود می‌توان کلاس Factory ساخت سرویس را تغییر داد به صورت زیر:

```
public class CustomServiceHostFactory : ServiceHostFactory
{
    public override ServiceHost CreateServiceHost( Type t, Uri[] baseAddresses )
    {
        return new CustomServiceHost( t, baseAddresses )
    }
}
```

حال بر روی سرویس مورد نظر کلیک راست کرده و گزینه View Markup را انتخاب نمایید، چیزی شبیه به گزینه زیر را مشاهده خواهید کرد:

```
<%@ ServiceHost Language="C#" Debug="true" Service="WcfService1.Service1" CodeBehind="Service1.svc.cs" %>
```

کافیست کلاس CustomServiceHostFactory را به عنوان Factory این سرویس مشخص نماییم. به صورت زیر:

```
<%@ ServiceHost Language="C#" Debug="true" Factory="CustomServiceHostFactory"
Service="WcfService1.Service1" CodeBehind="Service1.svc.cs" %>
```

از این به بعد عملیات وهله سازی از سرویس بر اساس تنظیمات پیش فرض صورت گرفته در این کلاس‌ها انجام می‌گیرد.

اگر قصد داشته باشیم که تزریق وابستگی (Dependency Injection) را برای سرویس های WCF پایاده سازی کنیم نیاز به یک Instance Provider سفارشی داریم. در ابتدا باید سرویس های مورد نظر را در یک Ioc Container رجیستر نماییم سپس با استفاده از InstanceProvider عملیات و هله سازی از سرویس ها همراه با تزریق وابستگی انجام خواهد گرفت. فرض کنید سرویسی به صورت زیر داریم:

```
[ServiceBehavior( IncludeExceptionDetailInFaults = true)]
public class BookService : IBookService
{
    public BookService(IBookRepository bookRepository)
    {
        Repository = bookRepository;
    }

    public IBookRepository Repository
    {
        get;
        private set;
    }

    public IList<Entity.Book> GetAll()
    {
        return Repository.GetAll();
    }
}
```

همانطور که می بینید برای عملیات و هله سازی از این سرویس نیاز به تزریق کلاس BookRepository است که این کلاس باید ابنتر فیس IBookRepository را پایاده سازی کرده باشد. برای این که Instance Provider ما بتواند عملیات تزریق وابستگی را به درستی انجام دهد، ابتدا باید BookRepository و BookService را به یک IocContainer (در این جا از الگوی [ServiceLocator](#) و [UnityContainer](#) استفاده کردم) رجیستر نماییم. به صورت زیر:

```
var container = new UnityContainer();

container.RegisterType<IBookRepository, BookRepository>();
container.RegisterType<BookService, BookService>();

ServiceLocator.SetLocatorProvider(new ServiceLocatorProvider(() => { return container; }));
```

حال باید InstanceProvider را به صورت زیر ایجاد نماییم:

```
public class UnityInstanceProvinder : IInstanceProvider
{
    private Type serviceType;

    public UnityInstanceProvinder( Type serviceType )
    {
        this.serviceType = serviceType;
    }

    public object GetInstance( InstanceContext instanceContext, Message message )
    {
        return ServiceLocator.Current.GetInstance( serviceType );
    }

    public object GetInstance( InstanceContext instanceContext )
    {
        return GetInstance( instanceContext, null );
    }
}
```

```

public void ReleaseInstance( InstanceContext instanceContext, object instance )
{
    if ( instance is IDisposable )
    {
        ( ( IDisposable )instance ).Dispose();
    }
}

```

با پیاده سازی متدهای اینترفیس `IInstanceProvider` می توان عملیات وهله سازی سرویس های WCF را تغییر داد. متد `GetInstance` همین کار را برای ما انجام خواهد داد. در این متد ما با توجه به نوع `ServiceType` سرویس مورد نظر را از `ServiceLocator` تامین خواهیم کرد. چون وابستگی های سرویس هم در `IOC Cotnainer` موجود است در نتیجه سرویس به درستی وهله سازی خواهد شد. از آن جا که در WCF عملیات وهله سازی از سرویس ها به طور مستقیم به نوع سرویس بستگی دارد، هیچ نیازی به نوع `Contract` مربوطه نیست. در نتیجه `Service Type` به صورت مستقیم در اختیار این کلاس قرار خواهد گرفت. مرحله آخر معرفی `IInstanceProvider` به عنوان یک `Service Behavior` است. برای این کار کدهای زیر را در کلاسی به نام `UnityInstanceProviderContext` کپی نمایید:

```

public class UnityInstanceProviderContext : IServiceBehavior
{
    public void AddBindingParameters( ServiceDescription serviceDescription , ServiceHostBase
serviceHostBase , Collection<ServiceEndpoint> endpoints , BindingParameterCollection bindingParameters
)
    {
    }

    public void ApplyDispatchBehavior( ServiceDescription serviceDescription , ServiceHostBase
serviceHostBase )
    {
        serviceHostBase.ChannelDispatchers.ToList().ForEach( channelDispatcherBase =>
        {
            var channelDispatcher = ( channelDispatcherBase as ChannelDispatcher );
            if ( channelDispatcher != null )
            {
                channelDispatcher.Endpoints.ToList().ForEach( endpoint =>
                {
                    endpoint.DispatchRuntime.InstanceProvider = new UnityInstanceProvider(
serviceDescription.ServiceType );
                } );
            }
        } );
    }

    public void Validate( ServiceDescription serviceDescription , ServiceHostBase serviceHostBase )
    {
    }
}

```

در متد `ApplyDispatchBehavior` همان طور دیده می شود به ازای تمام `EndPoint` های هر `ChannelDispatcher` یک نمونه از کلاس `UnityInstanceProvider` به همراه پارامتر سازنده آن که نوع سرویس مورد نظر است به خاصیت `InstanceProvider` در `DispatchRuntime` معرفی می گردد. در هنگام هاست سرویس مورد نظر هم باید تمام این عملیات به عنوان یک `Behavior` در اختیار `Service Host` قرار گیرد. همانند نمونه کد ذیل:

```

using (ServiceHost host = new ServiceHost(typeof(BookService)))
{
    host.Description.Behaviors.Add( new UnityInstanceProviderContext() );
}

```

نظرات خوانندگان

نویسنده: وحید م
تاریخ: ۱۳۹۲/۱۱/۱۱ ۸:۴۸

با سلام؛ اگر یک برنامه چند لایه داشته باشیم (UI- DomainLayer-DataAccess- Service) و مثلاً یک پروژه‌ای هم از نوع webapi یا wcf در آن معماری قرار داشت، می‌خواهم در web api یا wcf هم برای اعمال dependency Injection با آن mapping ها که در لایه ui قرار دادم هم استفاده کنم. با تشکر.

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۱/۱۱ ۹:۳۶

در مطلب فوق محل قرارگیری container.RegisterType در نقطه آغاز برنامه است؛ جایی که نگاشت‌های مورد نیاز در سایر لایه‌ها هم انجام می‌شود. بنابراین فرقی نمی‌کند.

نویسنده: وحید م
تاریخ: ۱۳۹۲/۱۱/۱۱ ۱۰:۴۳

ممنون ولی سوال بنده کلی بود؟ وقتی یک معماری دارم بگونه ای گفته شد یا مثل cms IRIS آقای سعیدی فر و خواستم به پروژه پروژه دیگری اضافه کنم از نوع webapi یا wcf که به نوعی از لایه service هم برای اتصال به بانک استفاده میکنه DI را باید چگونه برای آن اعمال کرد ؟ آیا می‌بایست تنظیمات و mapping های داخل global مربوط به structuremap درون ui را در داخل پروژه webapi یا wcf هم قرار داد یا خیر؟ اگر webapi را جدا هاست کنیم چه تضمینی وجود دارد دیگر به پروژه webapi دسترسی نداشته باشد

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۱/۱۱ ۱۰:۵۰

صرف نظر از اینکه برنامه شما از چند DLL نهایتاً تشکیل میشه، تمام این‌ها داخل یک Application Domain اجرا می‌شن. یعنی عملاً یک برنامه‌ی واحد شما دارید که از اتصال قسمت‌های مختلف با هم کار می‌کنه. IoC Container هم تنظیماتش اول کار برنامه کش می‌شه. یعنی یکبار که تنظیم شد، در سراسر آن برنامه قابل دسترسی هست. بنابراین نیازی نیست همه جا تکرار بشه. یکبار آغاز کار برنامه اون رو تنظیم کنید کافی هست.

به صورت پیش فرض سرویس‌های WCF به صورت Sync اجرا خواهند شد، یعنی هر گاه درخواستی از سمت کلاینت به سرور ارسال شود سرور بعد از پردازش درخواست پاسخ مورد نظر را به کلاینت باز می‌گرداند. اما حالتی را در نظر بگیرید که بعد از دریافت Request از کلاینت بنا به دلایلی امکان پاسخ گویی سمت سرور در آن لحظه وجود ندارد. خوب چه اتفاقی خواهد افتاد؟ در این حالت thread جاری سمت کلاینت نیز در حالت wait است و برنامه سمت کلاینت از کار می‌افتد تا زمانی که پاسخ از سرور دریافت نماید. اما در WCF به صورت پیش فرض هر درخواست ارسالی باید در طی یک دقیقه در اختیار سرور قرار گیرد و سرور نیز باید در طی یک دقیقه پاسخ مورد نظر را برگرداند (مقادیر خواص SendTimeout و ReceiveTimeout برای مدیریت این موارد به کار می‌روند). افزایش مقادیر این خواص کمک خاصی به این حالت نمی‌کند زیرا هم چنان کلاینت در حالت wait است و سرور نیز پاسخ خاصی ارسال نمی‌کند. حتی اگر کل عملیات را به صورت Async پیاده سازی نماییم باز ممکن است بعد از منقضی شدن زمان پردازش با یک TimeoutException برنامه از کار بیفتد. برای حل اینگونه موارد پیاده سازی سرویس‌ها به صورت Long Polling به ما کمک خوبی خواهد کرد.

حال سناریو زیر را در نظر بگیرید:

سمت سرور:

«یک درخواست دریافت می‌شود؛

«سرور در حالت wait (البته توسط یک thread دیگر) منتظر تامین منابع برای پاسخ به کلاینت است؛

«در نهایت پاسخ مورد نظر ارسال خواهد شد.

سمت کلاینت:

«درخواست مورد نظر به سرور ارسال می‌شود؛

«کلاینت منتظر پاسخ از سمت سرور است (البته توسط یک Thread دیگر)؛

«اگر در حین انتظار برای پاسخ از سمت سرور، با یک TimeoutException روبرو شدیم به جای توقف برنامه و نمایش پیغام خطای Server is not available، باید عملیات به صورت خودکار restart شود.

«در نهایت پاسخ مورد نظر دریافت خواهد شد.

پیاده سازی این سناریو در WCF کار پیچیده ای نیست. بدین منظور می‌توانید از کلاس زیر استفاده کنید (لینک دانلود). سورس آن به صورت زیر است:

```
public abstract class LongPollingAsyncResult<TResult> : IAsyncResult where TResult : class
{
    #region - Fields -
    private AsyncCallback _callback;
    private TimeSpan _timeoutSpan;
    private TimeSpan _intervalWaitSpan;

    #endregion

    #region - Properties -
    public Exception Exception { get; private set; }

    public TResult Result { get; private set; }

    public object SyncRoot { get; private set; }

    #endregion

    #region - Ctor -
    public LongPollingAsyncResult(AsyncCallback callback, object asyncState, int timeoutSeconds = 300, int intervalWaitMilliseconds = 500)
    {
        SyncRoot = new object();
        _callback = callback;
        AsyncState = asyncState;
        AsyncWaitHandle = new ManualResetEvent(IsCompleted);
        _timeoutSpan = TimeSpan.FromSeconds(timeoutSeconds);
        _intervalWaitSpan = TimeSpan.FromMilliseconds(intervalWaitMilliseconds);
    }
}
```

```

        ThreadPool.QueueUserWorkItem(new WaitCallback(LoopWithIntervalAndTimeout));
    }
#endregion

#region - Private Helper Methods -
private void LoopWithIntervalAndTimeout(object input)
{
    try
    {
        Stopwatch stopwatch = new Stopwatch();
        stopwatch.Start();
        while (!IsCompleted)
        {
            if (stopwatch.Elapsed > _timeoutSpan)
                throw new TimeoutException();

            DoWork();

            if (!IsCompleted)
                Thread.Sleep(_intervalWaitSpan);
        }
    }
    catch (Exception e)
    {
        Complete(null, e);
    }
}

#endregion

#region - Protected/Abstract Methods -
protected void Complete(TResult result, Exception e = null, bool completedSynchronously =
false)
{
    lock (SyncRoot)
    {
        CompletedSynchronously = completedSynchronously;
        Result = result;
        Exception = e;
        IsCompleted = true;

        if (_callback != null)
            _callback(this);
    }
}

protected abstract void DoWork();
#endregion

#region - Public Methods -
public TResult WaitForResult()
{
    if (!IsCompleted)
        AsyncWaitHandle.WaitOne();

    if (Exception != null)
    {
        if (Exception is TimeoutException && WebOperationContext.Current != null)
            WebOperationContext.Current.OutgoingResponse.StatusCode =
HttpStatusCode.RequestTimeout;

        throw Exception;
    }

    return Result;
}

#endregion

#region - IAsyncResult Implementation -
public object AsyncState { get; private set; }
public WaitHandle AsyncWaitHandle { get; private set; }
public bool CompletedSynchronously { get; private set; }

```

```

        public bool IsCompleted { get; private set; }
    #endregion
}

```

در این حالت شما می‌توانید حداکثر زمان مورد نیاز برای درخواست را به عنوان پارامتر از طریق سازنده کلاس بالا تعیین نمایید. اگر این زمان بیش از زمان تعیین شده در خواص SendTimeout و ReceiveTimeout بود بعد از منقضی شدن زمان پردازش درخواست، به جای دریافت TimeoutException عملیات پردازش به کار خود ادامه خواهد داد. برای استفاده از کلاس تهیه شده ابتدا باید عملیات خود را به صورت Async پیاده سازی نمایید که در این [مقاله](#) به صورت کامل شرح داده شده است.

یک مثال

قصد داریم Operation زیر را به صورت Long Polling پیاده سازی نماییم:

```

[OperationContract]
public string GetNotification();

```

ابتدا متد زیر باید به صورت Async تبدیل شود:

```

[OperationContract(AsyncPattern = true)]
public IAsyncResult BeginWaitNotification(AsyncCallback callback, object state);

public string EndWaitNotification(IAsyncResult result);

```

حال نوع بازگشتی سرویس مورد نظر را با استفاده از کلاس LongPollingAsyncResult به صورت زیر ایجاد خواهیم کرد:

```

public class MyNotificationResult : LongPollingAsyncResult<string>
{
    protected override DoWork()
    {
        // کدهای مورد نظر را اینجا قرار دهید
        base.Complete(...);
    }
}

```

در نهایت پیاده سازی متدهای Begin و End همانند ذیل خواهد بود:

```

public IAsyncResult BeginWaitNotification(AsyncCallback callback, object state)
{
    return new MyNotificationResult(callback, state);
}

public string EndWaitNotification(IAsyncResult result)
{
    MyNotificationResult myResult = result as MyNotificationResult;
    if(myResult == null)
        throw new ArgumentException("result was of the wrong type!");

    myResult.WaitForResult();
    return myResult.Result;
}

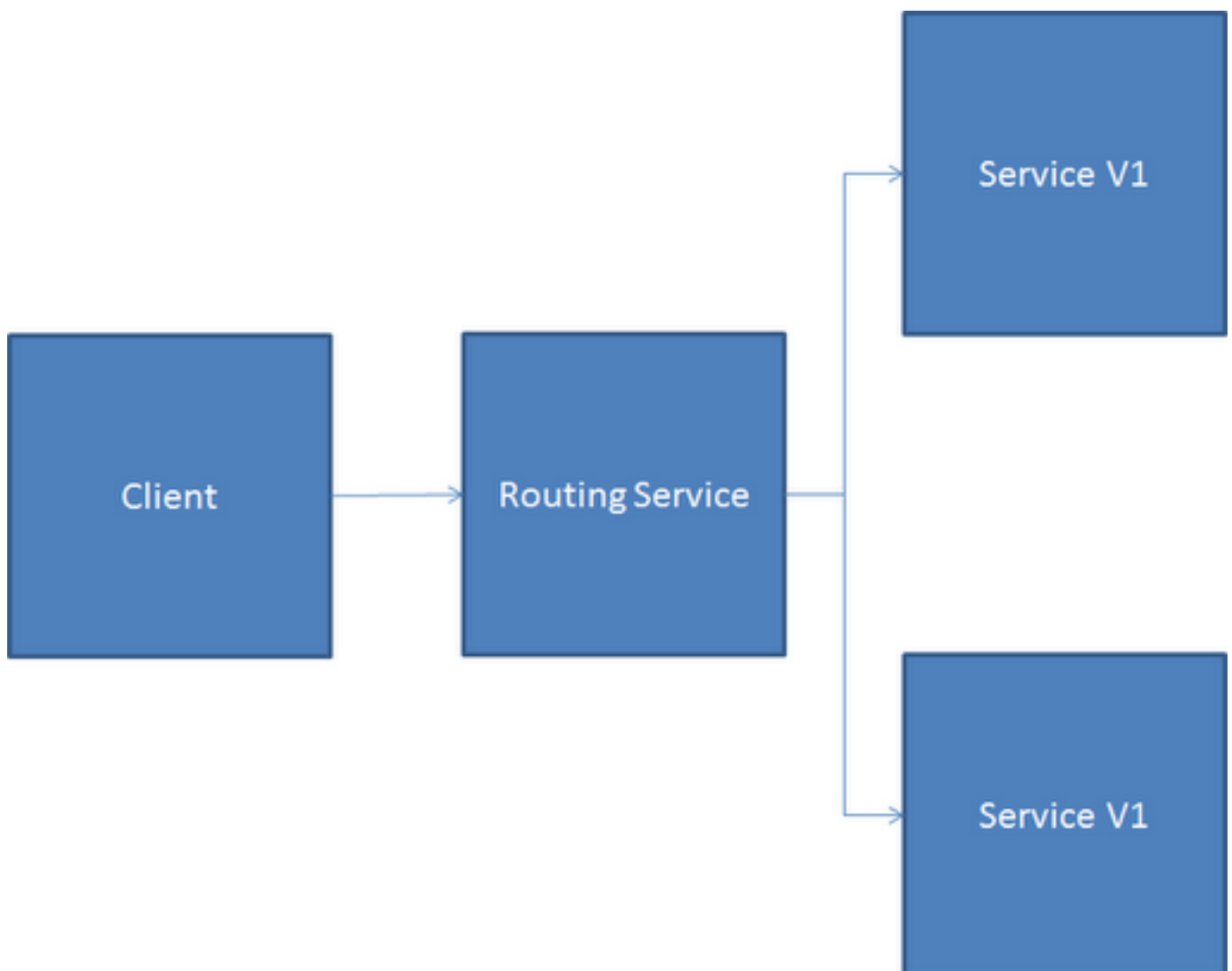
```

در این حالت کلاینت می‌تواند یک درخواست به صورت LongPolling به سرور ارسال نماید و البته مدیریت این درخواست در یک thread دیگر انجام می‌گیرد که نتیجه آن از عدم تداخل پردازش این درخواست با سایر قسمت‌های برنامه است.

به صورت معمول در سیستم‌های مبتنی بر WCF ارتباط بین سرور و کلاینت در قالب EndpointConfiguration تعریف می‌شوند. یعنی کلاینت برای برقراری ارتباط با سرور نیاز به آدرسی که سرور مورد نظر در آن هاست شده است دارد. این روش هنگامی که فقط یک مقصد وجود داشته باشد روش موثری است. اما اگر سرویس‌های مورد نظر در چند سرور هاست شده باشند نیاز به سیستم مسیریابی خواهیم داشت. خوشبختانه در WCF 4.0 این امکان به خوبی تدارک دیده شده است.

WCF Routing Service چیست؟

Routing Service به عنوان سرویس مسیریابی WCF در دات نت 4 معرفی شد. به وسیله Routing Service می‌توان Endpoint Configuration مقصدهای مختلف را با هم تجمیع کرد و در نتیجه تعداد تنظیمات برای Endpoint در سمت کلاینت کاهش پیدا می‌کند به طوری که کلاینت فقط با یک مقصد در ارتباط است. مقصد کلاینت همان Routing Service می‌باشد که در این سرور درخواست‌های رسیده از کلاینت‌ها با توجه به فیلتر انجام شده به مقصد اصلی ارسال خواهند شد. با استفاده از Routing Service معماری سیستم به صورت تغییر پیدا می‌کند:



اهداف:

موارد زیر اهداف و مزایای استفاده از Routing Service است:

Service versioning «

Content-based routing scenario «

Service partitioning «

Protocol bridging «

هر کدام از موارد بالا در طی پست‌های جداگانه شرح داده خواهند شد.

بررسی یک مثال:

دو Contract به صورت زیر تعریف می‌کنیم:

```
[ServiceContract]
public interface ICalculatorV1
{
    [OperationContract]
    int Add(int a, int b);
}

[ServiceContract]
public interface ICalculatorV2
{
    [OperationContract]
    int Sub(int a, int b);
}
```

پیاده سازی Contract‌های بالا در قالب سرویس به صورت زیر خواهد بود:

```
public class CalculatorV1 : ICalculatorV1
{
    public int Add(int a, int b)
    {
        return a + b;
    }
}

public class CalculatorV2 : ICalculatorV2
{
    public int Sub(int a, int b)
    {
        return a - b;
    }
}
```

تنظیمات Binding سرویس‌ها:

```
system.serviceModel>
<services>
  <service name="WCFRoutingSample.CalculatorV1">
    <host>
      <baseAddresses>
        <add baseAddress = "http://localhost:8732/CalculatorServiceV1/" />
      </baseAddresses>
    </host>

    <endpoint address = "" binding="basicHttpBinding" contract="WCFRoutingSample.ICalculatorV1">
```

```

        <identity>
            <dns value="localhost"/>
        </identity>
    </endpoint>

    <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange"/>
</service>
<service name="WCFRoutingSample.CalculatorV2">
    <host>
        <baseAddresses>
            <add baseAddress = "http://localhost:8733/CalculatorServiceV2/" />
        </baseAddresses>
    </host>

    <endpoint address="" binding="basicHttpBinding" contract="WCFRoutingSample.ICalculatorV2">

        <identity>
            <dns value="localhost"/>
        </identity>
    </endpoint>

    <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange"/>
</service>
</services>
<behaviors>
    <serviceBehaviors>
        <behavior>
            <serviceMetadata httpGetEnabled="True"/>
            <serviceDebug includeExceptionDetailInFaults="False" />
        </behavior>
    </serviceBehaviors>
</behaviors>
</system.serviceModel>

```

حال باید RoutingService را به صورت زیر هاست نماییم:

```

class Program
{
    static void Main(string[] args)
    {
        var host = new ServiceHost(typeof(RoutingService));
        host.Open();
        Console.WriteLine("Server is running.");
        Console.ReadLine();
        host.Close();
    }
}

```

مهم‌ترین بخش تنظیمات مربوط به Routing Service است:

```

<system.serviceModel>
    <behaviors>
        <serviceBehaviors>
            <behavior name="routingBehv">
                <routing routeOnHeadersOnly="false" filterTableName="filters"/>
                <serviceDebug includeExceptionDetailInFaults="true"/>
                <serviceMetadata httpGetEnabled="true"/>
            </behavior>
        </serviceBehaviors>
    </behaviors>
    <routing>
        <filters>
            <filter name="CalV1ServiceFilter" filterType="EndpointName" filterData="Calv1Service"/>
            <filter name="CalV2ServiceFilter" filterType="EndpointName" filterData="Calv2Service"/>
        </filters>
        <filterTables>
            <filterTable name="filters">
                <add filterName="CalV1ServiceFilter" endpointName="Calv1Service" />
                <add filterName="CalV2ServiceFilter" endpointName="Calv2Service"/>
            </filterTable>
        </filterTables>
    </routing>
</system.serviceModel>
<!-- Routing service with endpoint definition -->

```

```

<service name="System.ServiceModel.Routing.RoutingService"
  behaviorConfiguration="routingBehv">
  <endpoint
    address="/Calv1"
    binding="basicHttpBinding"
    contract="System.ServiceModel.Routing.IRequestReplyRouter"
    name="Calv1Service"/>
  <endpoint
    address="/Calv2"
    binding="basicHttpBinding"
    contract="System.ServiceModel.Routing.IRequestReplyRouter"
    name="Calv2Service"/>

  <host>
    <baseAddresses>
      <add baseAddress="http://localhost:9000/CalculatorService"/>
    </baseAddresses>
  </host>
</service>
</services>
<client>
  <endpoint address="http://localhost:8732/CalculatorServiceV1"
    binding="basicHttpBinding"
    contract="*"
    name="Calv1Service"/>
  <endpoint address="http://localhost:8733/CalculatorServiceV2"
    binding="basicHttpBinding"
    contract="*"
    name="Calv2Service"/>
</client>
</system.serviceModel>

```

همان طور که مشاهده می‌کنید ابتدا اطلاعات Binding دو سرویس نوشته در بالا را به عنوان Endpoint های مختلف تعریف کردیم و سپس با استفاده از FilterTable نوع درخواست را به Endpoint مورد نظر وصل کردیم (در این مثال فیلتر بر اساس نوع Endpoint است). به وسیله این تعاریف Routing Service می‌داند که هر درخواست را به کدام سرویس ارسال نماید و پاسخ به کجا بازگشت داده شود.

در [پست قبلی](#) توضیحات کلی درباره WCF Routing Service داده شد و یک مثال را نیز با هم بررسی کردیم. همان طور که در مثال مشاهده شد با استفاده از تعاریف فیلتر در جدول فیلترها توانستیم درخواست‌های مورد نظر را به مقاصد مربوطه اتصال دهیم. در این پست نگاه عمیق‌تری به FilterTable خواهیم داشت.

MessageFilter ها:

با استفاده از این نوع، می‌توان فیلتر مورد نظر را بر روی Message گسترش داد. برای مثال ارزیابی نام فرستنده Message یا حتی نوع عملیات Soap. حتی می‌توانیم فیلترها را با استفاده از And با هم ترکیب نماییم.

FilterType ها

این enum دارای مقادیر زیر است:

Action : با استفاده [ActionMessageFilter](#) فیلتر مورد نظر انجام می‌شود.

And : با استفاده از [StrictAndMessageFilter](#) دو فیلتر مورد نظر را با هم ترکیب می‌کند.

Custom : می‌توان فیلتر مورد نظر را تعریف کرده و این جا فراخوانی نمایید.

MatchAll : با استفاده از [MatchAllMessageFilter](#) تمام فیلترها بررسی خواهند شد.

EndpointAddress : برای فیلتر آدرس درخواست‌های با استفاده از [EndpointAddressMessageFilter](#) مورد استفاده قرار می‌گیرد.

EndpointName : فیلتر با استفاده [EndpointNameMessageFilter](#) بر روی نام Endpoint سرویس مورد نظر انجام می‌گیرد.

FilterData برای تعیین مقادیر مورد نیاز برای FilterType مورد استفاده قرار می‌گیرد.

برای مثال:

```
<filters>
  <filter name="EndpointNameFilter" filterType="EndpointName"
    filterData="calculatorEndpoint"/>
  <filter name="RoundRobinFilter1" filterType="Custom"
    customType="RoutingServiceFilters.RoundRobinMessageFilter,
    RoutingService" filterData="group1"/>
  <filter name="RoundRobinFilter2" filterType="Custom"
    customType="RoutingServiceFilters.RoundRobinMessageFilter,
    RoutingService" filterData="group1"/>
</filters>
```

Filter Table

در واقع مجموعه ای است از اشیای تعریف شده از نوع [FilterTableEntryElement](#) که ارتباط را بین یک فیلتر و مقصد (Endpoint) تعیین می‌نماید. هم چنین امکان تعریف اولویت برای هر کدام از مقصدها یا Endpoint ها وجود دارد. یک مثال:

```
<routing>
  <filters>
    <filter name="AddAction" filterType="Action" filterData="Add" />
    <filter name="SubtractAction" filterType="Action" filterData="Subtract" />
  </filters>
  <filterTables>
    <table name="routingTable1">
      <filters>
        <add filterName="AddAction" endpointName="Addition" />
        <add filterName="SubtractAction" endpointName="Subtraction" />
      </filters>
    </table>
  </filterTables>
</routing>
```


می‌توان برای فیلترها اولویت تعیین کرد. این کار از طریق تنظیم خاصیت Priority امکان پذیر است. در صورت عدم تعیین Priority مقدار پیش فرض صفر خواهد بود.

```
<filterTables>
  <filterTable name="filterTable1">
    <add filterName="EndpointNameFilter" endpointName="regularCalcEndpoint"
      priority="1"/>
    <add filterName="MatchAllMessageFilter" endpointName="defaultCalcEndpoint"
      priority="0"/>
  </filterTable>
</filterTables>
```

در مثال بالا برای یک endpointName مشترک دو فیلتر نوشته شده است با اولویت‌های متفاوت. دو صورتی که اولویت‌ها یکسان باشد با توجه به ترتیب تعریف در filterTable، فیلترها اعمال خواهند شد.

تهیه BackupList

BackupListها این امکانی را در اختیار ما قرار خواهند داد که بتوانیم در صورت عدم موفقیت در عملیات مسیر یابی (برای مثال وقوع CommunicationException) لیستی از مسیرهای جایگزین را تعیین نماییم. در صورت وقوع هر گونه خطا در هنگام فراخوانی سرویس، به جای مواجه شدن با یک استثنا، عملیات مسیر یابی به صورت خودکار به endpointهای تعیین شده در BackupList منتقل خواهد شد.

```
<filterTables>
  <filterTable name="filterTable1">
    <add filterName="MatchAllFilter1" endpointName="Destination"
      backupList="backupEndpointList"/>
  </filterTable>
</filterTables>
<backupLists>
  <backupList name="backupEndpointList">
    <add endpointName="backupServiceQueue" />
    <add endpointName="alternateServiceQueue" />
  </backupList>
</backupLists>
```

در مثال بالا دو endpoint در لیست backup قرار دارد. در صورت وقوع استثنا در Destination عملیات ابتدا به backupServiceQueue منتقل می‌شود و اگر باز هم خطایی وجود داشت نوبت به alternateServiceQueue خواهد رسید.

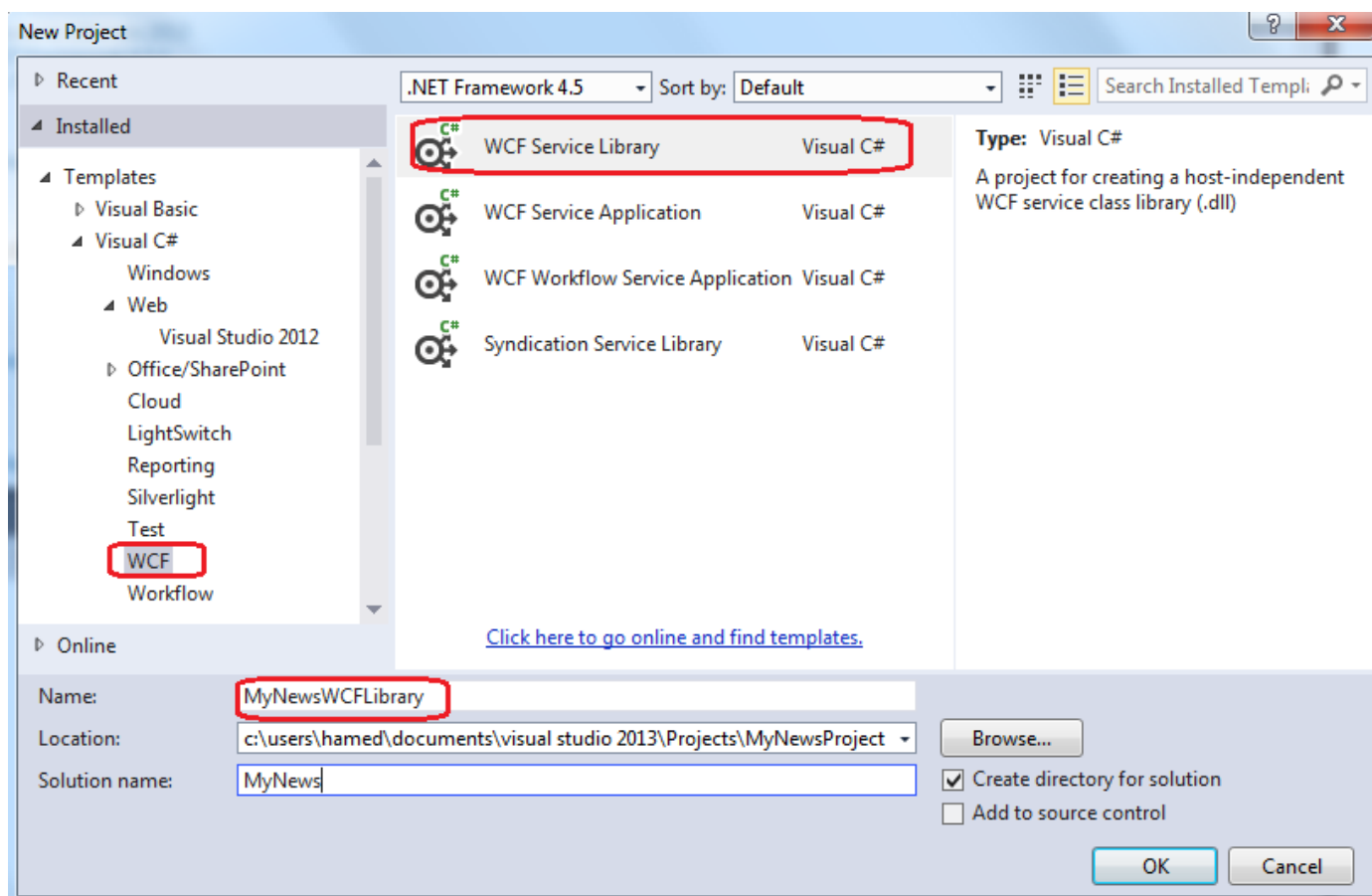
در این نوشتار که به صورت آموزش تصویری ارائه می‌شود؛ یک سرویس WCF در Visual Studio 2013 ایجاد می‌کنم. سپس روش استفاده از آن را در یک برنامه ویندوزی آموزش خواهم داد. در اینجا در نظر گرفته شده است که شما افزونه‌ی [Resharper](#) را روی ویژوال استودیوی خود نصب دارید. پس در صورتیکه هنوز به سراغ آن نرفته اید درنگ نکنید و واپسین نگارش آن را دانلود کنید.

در این پروژه‌ی ساده در نظر می‌گیریم که دو جدول یکی برای اخبار، شامل عنوان، متن خبر و تاریخ ثبت و دسته بندی و دیگری برای نگهداری دسته‌ها در پایگاه داده داریم و می‌خواهیم سرویس‌های مناسب با این دو جدول را بسازیم. با کد زیر، پایگاه داده‌ی dbTest و جدول‌های tblNews و tblCategory در SQL Server 2012 ساخته می‌شود:

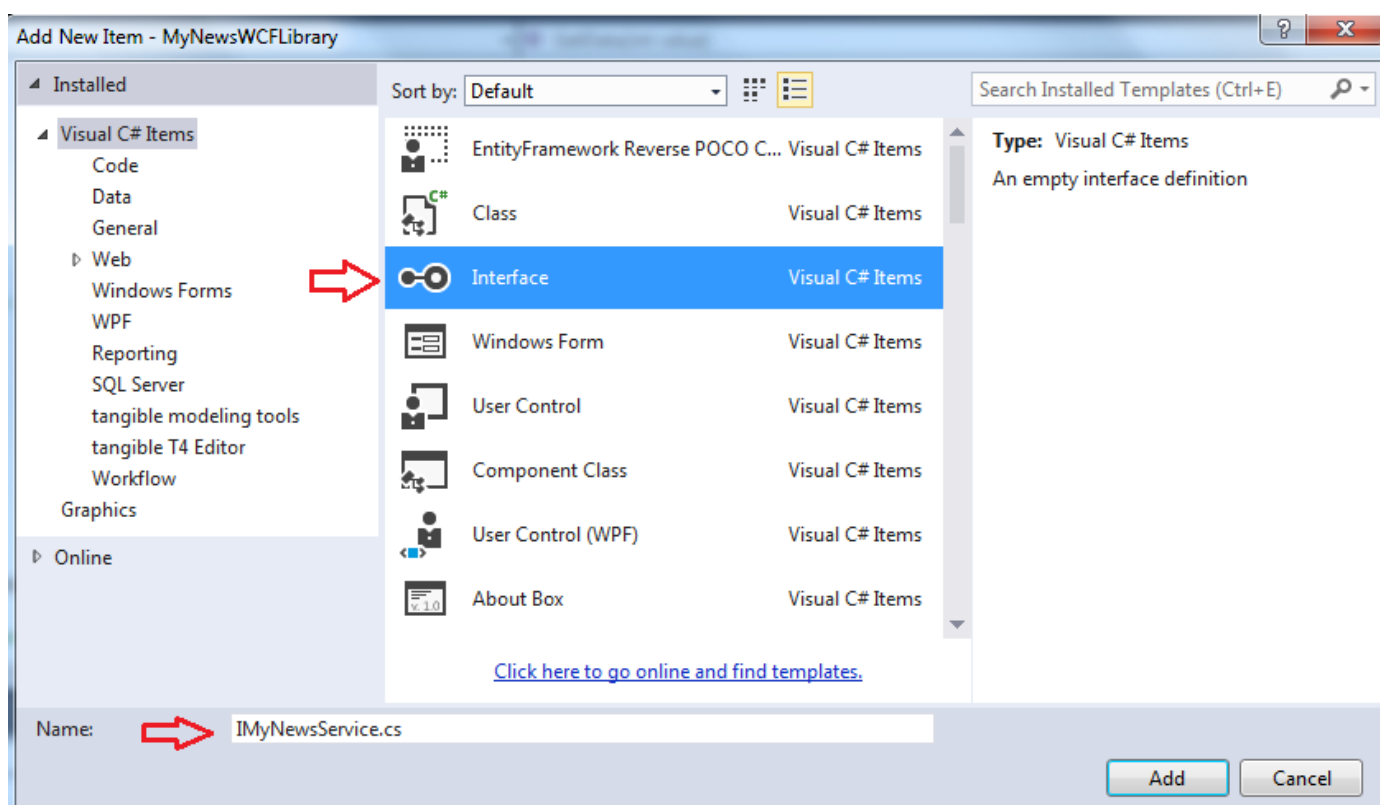
```
USE [master]
GO
/***** Object: Database [dbMyNews]      Script Date: 2014/01/14 09:46:04 ب.ظ *****/
CREATE DATABASE [dbMyNews]
    CONTAINMENT = NONE
    ON PRIMARY
    ( NAME = N'dbMyNews', FILENAME = N'D:\dbMyNews.mdf' , SIZE = 5120KB , MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB )
    LOG ON
    ( NAME = N'dbMyNews_log', FILENAME = N'D:\dbMyNews_log.ldf' , SIZE = 1024KB , MAXSIZE = 2048GB , FILEGROWTH = 10%)
GO
USE [dbMyNews]
GO
/***** Object: Table [dbo].[tblCategory]  Script Date: 2014/01/14 09:46:04 ب.ظ *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[tblCategory](
    [tblCategoryId] [int] IDENTITY(1,1) NOT NULL,
    [CatName] [nvarchar](50) NOT NULL,
    [IsDeleted] [bit] NOT NULL,
    CONSTRAINT [PK_tblCategory] PRIMARY KEY CLUSTERED
    (
        [tblCategoryId] ASC
    )WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/***** Object: Table [dbo].[tblNews]      Script Date: 2014/01/14 09:46:04 ب.ظ *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[tblNews](
    [tblNewsId] [int] IDENTITY(1,1) NOT NULL,
    [tblCategoryId] [int] NOT NULL,
    [Title] [nvarchar](50) NOT NULL,
    [Description] [nvarchar](max) NOT NULL,
    [RegDate] [datetime] NOT NULL,
    [IsDeleted] [bit] NULL,
    CONSTRAINT [PK_tblNews] PRIMARY KEY CLUSTERED
    (
        [tblNewsId] ASC
    )WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
ALTER TABLE [dbo].[tblNews] WITH CHECK ADD CONSTRAINT [FK_tblNews_tblCategory] FOREIGN
KEY([tblCategoryId])
REFERENCES [dbo].[tblCategory] ([tblCategoryId])
GO
ALTER TABLE [dbo].[tblNews] CHECK CONSTRAINT [FK_tblNews_tblCategory]
GO
USE [master]
```

```
GO
ALTER DATABASE [dbMyNews] SET READ_WRITE
GO
```

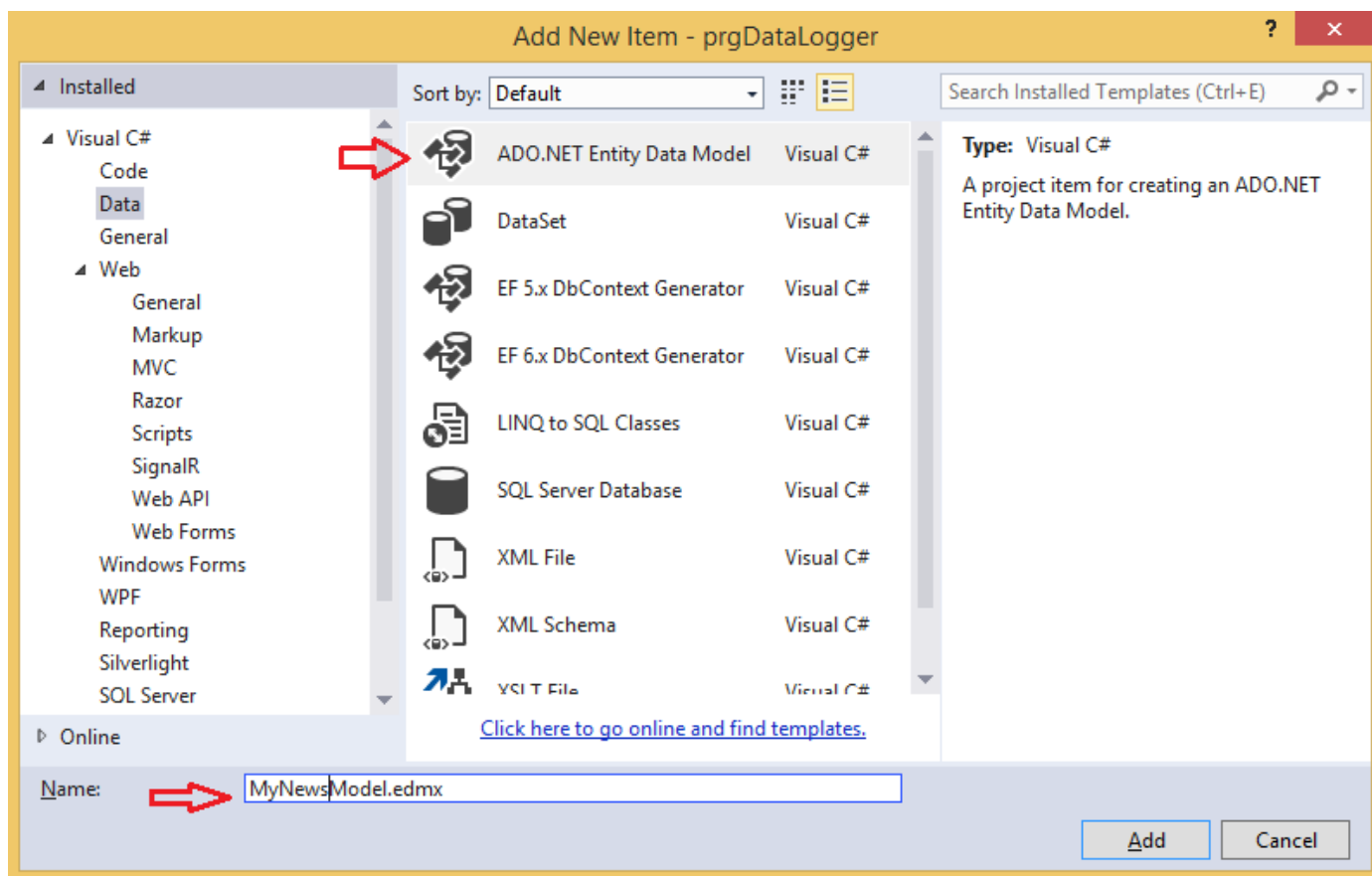
اکنون Visual Studio 2013 را باز کنید سپس روی گزینه New Project کلیک کنید و برابر با نگاره‌ی زیر عمل کنید:

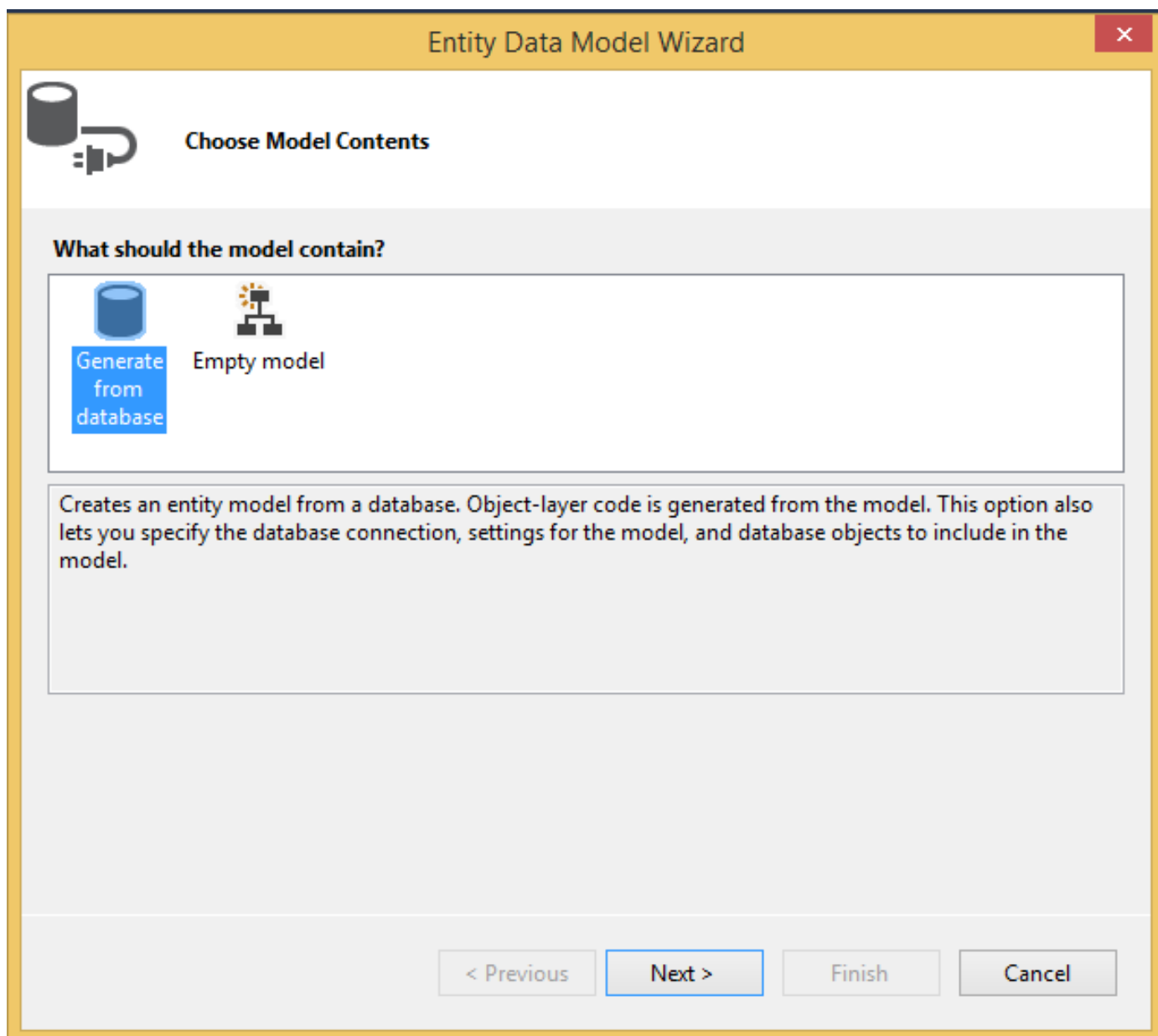


پروژه MyNewsWCFLibrary در راه حل MyNews ساخته می‌شود. این پروژه به صورت پیش‌گزینه دارای یک کلاس به نام Service و یک interface به نام IService است. هر دو را حذف کنید و سپس روی نام پروژه راست‌کلیک کرده، از منوی باز شده گزینه‌ی Add -> New Item را انتخاب کنید. سپس برابر با نگاره‌ی زیر عمل کنید:

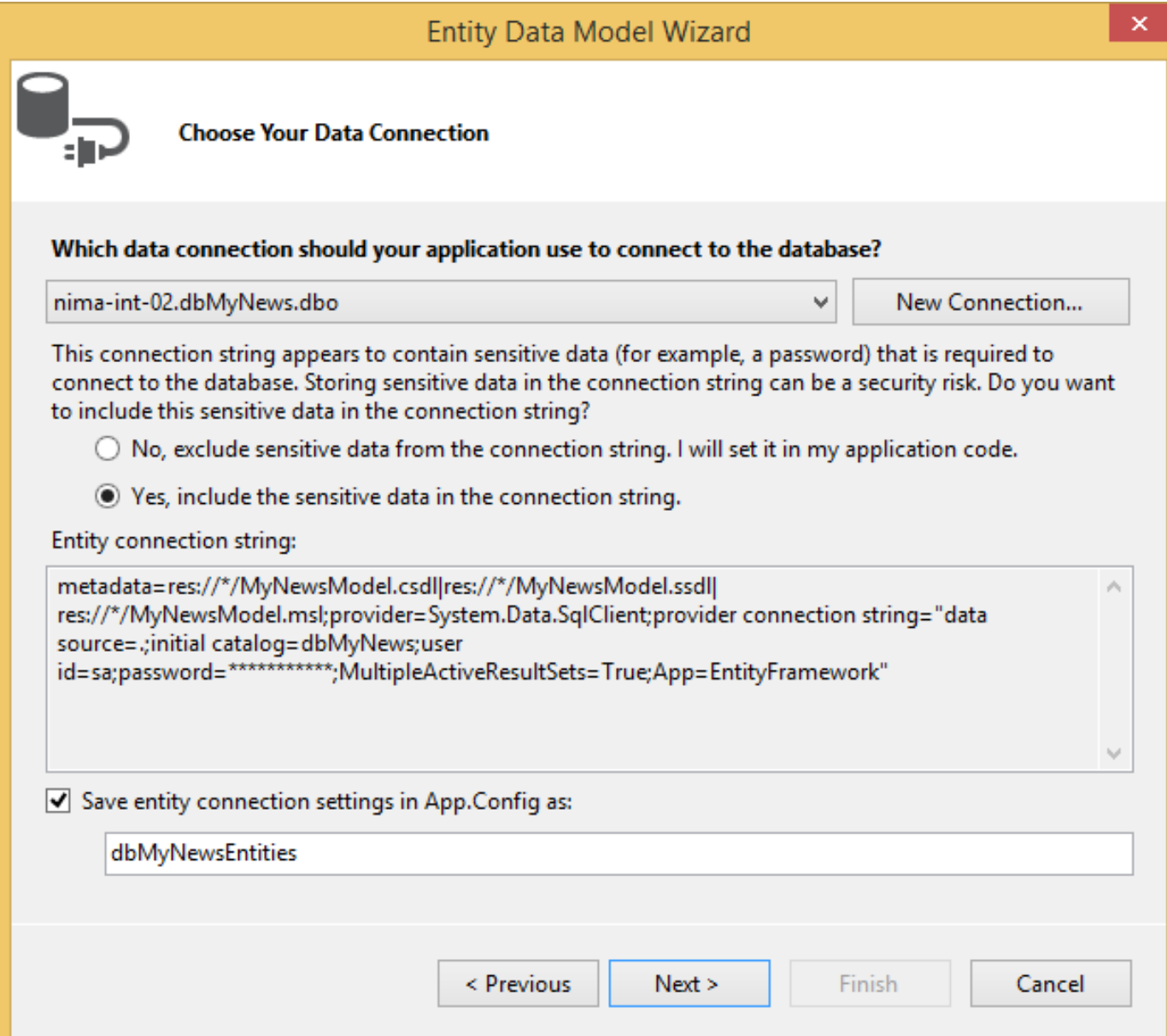


در لایه‌ی Service Interface کلیه‌ی روال‌های مورد نیاز برای ارتباط با پایگاه داده را می‌سازیم. پیش از آن باید یک Model برای ارتباط با پایگاه داده ساخته باشیم. برای این کار از پنجره Add New Item و از زیرمجموعه Data، گزینه ADO.NET Entity Data Model را انتخاب کنید و به‌سان زیر پیش روید:





در گام پسین روی دکمه New Connection کلیک کنید و رشته‌ی اتصال به پایگاه داده‌ی dbMyNews را بسازید. سپس همانند تنظیمات نگاره‌ی زیر ادامه دهید:



Entity Data Model Wizard

Choose Your Data Connection

Which data connection should your application use to connect to the database?

nima-int-02.dbMyNews.dbo New Connection...

This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

☐ No, exclude sensitive data from the connection string. I will set it in my application code.

☒ Yes, include the sensitive data in the connection string.

Entity connection string:

```
metadata=res://*/MyNewsModel.csdl|res://*/MyNewsModel.ssdl|
res://*/MyNewsModel.msl;provider=System.Data.SqlClient;provider connection string="data
source=.;initial catalog=dbMyNews;user
id=sa;password=*****;MultipleActiveResultSets=True;App=EntityFramework"
```

☒ Save entity connection settings in App.Config as:


dbMyNewsEntities

< Previous Next > Finish Cancel

در گام پسین گزینه‌ی Entity Framework 6.0 را برگزینید و روی دکمه‌ی Next کلیک کنید.

در پنجره نشان داده شده، جدول‌های مورد نیاز را همانند نگاره‌ی زیر انتخاب کرده و روی دکمه Finish کلیک کنید:

Entity Data Model Wizard

Choose Your Database Objects and Settings

Which database objects do you want to include in your model?

☒ Tables

☒ dbo

☒ tblCategory

☒ tblNews

☐ Views

☐ Stored Procedures and Functions

☐ Pluralize or singularize generated object names

☒ Include foreign key columns in the model

☐ Import selected stored procedures and functions into the entity model

Model Namespace:

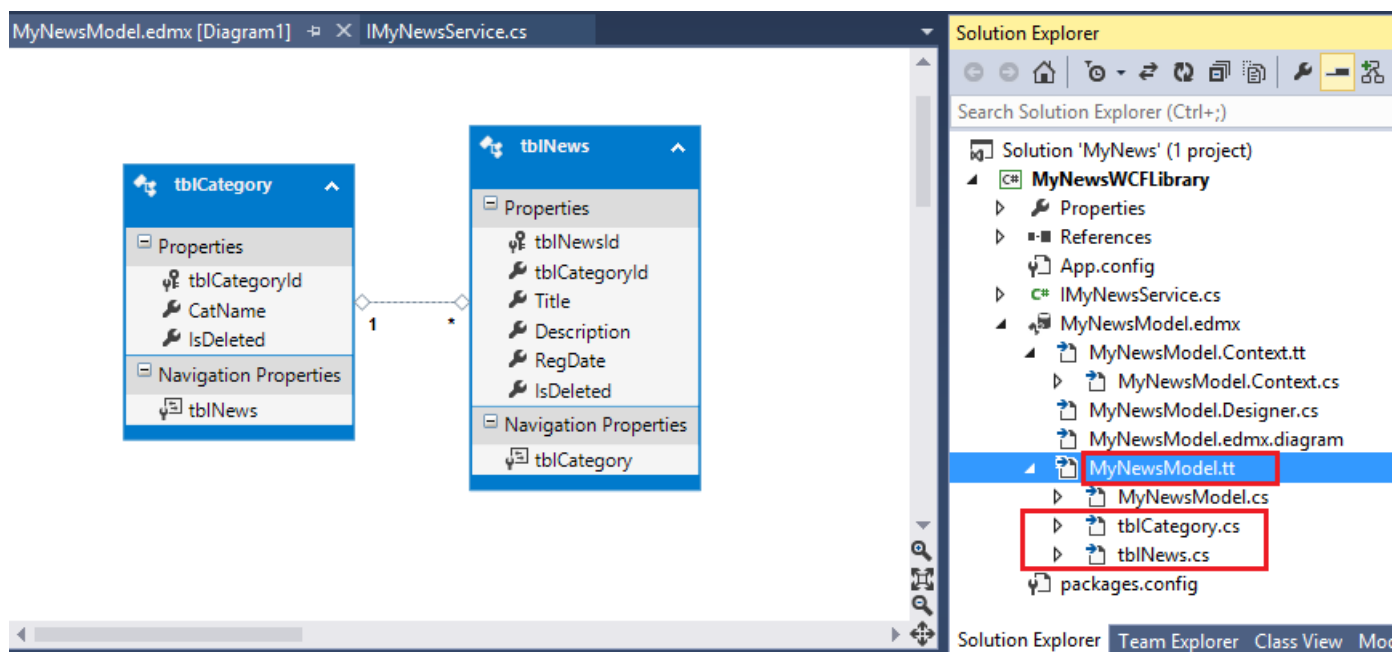
< Previous

Next >

Finish

Cancel

در پایان مدل ما همانند نگاره‌ی زیر خواهد بود.



در بخش پسین درباره‌ی شیوه‌ی دست‌کاری کلاس‌های Entity خواهیم نوشت.

برای استفاده از کلاس‌های Entity که در نوشتار پیشین ایجاد کردیم در WCF باید آن کلاس‌ها را دست‌کاری کنیم. متن کلاس tblNews را در نظر بگیرید:

```
namespace MyNewsWCFLibrary
{
    using System;
    using System.Collections.Generic;

    public partial class tblNews
    {
        public int tblNewsId { get; set; }
        public int tblCategoryId { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }
        public System.DateTime RegDate { get; set; }
        public Nullable<bool> IsDeleted { get; set; }

        public virtual tblCategory tblCategory { get; set; }
    }
}
```

مشاهده می‌کنید که برای تعریف کلاس‌ها از کلمه کلیدی partial استفاده شده است. استفاده از کلمه کلیدی partial به شما اجازه می‌دهد که یک کلاس را در چندین فایل جداگانه تعریف کنید. به عنوان مثال می‌توانید فیلدها، ویژگی‌ها و سازنده‌ها را در یک فایل و متدها را در فایل دیگر قرار دهید.

به صورت خودکار کلیه ویژگی‌ها به توجه به پایگاه داده ساخته شده اند. برای نمونه ما برای فیلد IsDeleted در SQL Server به ستون Allow Nulls را کلیک کرده بودیم که در نتیجه در اینجا عبارت Nullable پیش از نوع فیلد نشان داده شده است. برای استفاده از این کلاس در WCF باید صفت DataContract را به کلاس داد. این قرارداد به ما اجازه استفاده از ویژگی‌هایی که صفت DataMember را می‌گیرند را می‌دهد. کلاس بالا را به شکل زیر بازنویسی کنید:

```
using System.Runtime.Serialization;

namespace MyNewsWCFLibrary
{
    using System;
    using System.Collections.Generic;

    [DataContract]
    public partial class tblNews
    {
        [DataMember]
        public int tblNewsId { get; set; }
        [DataMember]
        public int tblCategoryId { get; set; }
        [DataMember]
        public string Title { get; set; }
        [DataMember]
        public string Description { get; set; }
        [DataMember]
        public System.DateTime RegDate { get; set; }
        [DataMember]
        public Nullable<bool> IsDeleted { get; set; }

        public virtual tblCategory tblCategory { get; set; }
    }
}
```

هم‌چنین کلاس tblCategory را به صورت زیر تغییر دهید:

```
namespace MyNewsWCFLibrary
{
    using System;
```

```

using System.Collections.Generic;
using System.Runtime.Serialization;

[DataContract]
public partial class tblCategory
{
    public tblCategory()
    {
        this.tblNews = new HashSet<tblNews>();
    }

    [DataMember]
    public int tblCategoryId { get; set; }
    [DataMember]
    public string CatName { get; set; }
    [DataMember]
    public bool IsDeleted { get; set; }

    public virtual ICollection<tblNews> tblNews { get; set; }
}

```

با انجام کد بالا از بابت مدل کارمان تمام شده است. ولی فرض کنید در اینجا تصمیم به تغییری در پایگاه داده می‌گیرید. برای نمونه می‌خواهید ویژگی Allow Nulls فیلد IsDeleted را نیز False کنیم و مقدار پیش‌گزیده به این فیلد بدهید. برای این کار باید دستور زیر را در SQL Server اجرا کنیم:

```

BEGIN TRANSACTION
GO
ALTER TABLE dbo.tblNews
DROP CONSTRAINT FK_tblNews_tblCategory
GO
ALTER TABLE dbo.tblCategory SET (LOCK_ESCALATION = TABLE)
GO
COMMIT
BEGIN TRANSACTION
GO
CREATE TABLE dbo.Tmp_tblNews
(
    tblNewsId int NOT NULL IDENTITY (1, 1),
    tblCategoryId int NOT NULL,
    Title nvarchar(50) NOT NULL,
    Description nvarchar(MAX) NOT NULL,
    RegDate datetime NOT NULL,
    IsDeleted bit NOT NULL
) ON [PRIMARY]
TEXTIMAGE_ON [PRIMARY]
GO
ALTER TABLE dbo.Tmp_tblNews SET (LOCK_ESCALATION = TABLE)
GO
ALTER TABLE dbo.Tmp_tblNews ADD CONSTRAINT
DF_tblNews_IsDeleted DEFAULT 0 FOR IsDeleted
GO
SET IDENTITY_INSERT dbo.Tmp_tblNews ON
GO
IF EXISTS(SELECT * FROM dbo.tblNews)
EXEC('INSERT INTO dbo.Tmp_tblNews (tblNewsId, tblCategoryId, Title, Description, RegDate, IsDeleted)
SELECT tblNewsId, tblCategoryId, Title, Description, RegDate, IsDeleted FROM dbo.tblNews WITH (HOLDLOCK
TABLOCKX)')
GO
SET IDENTITY_INSERT dbo.Tmp_tblNews OFF
GO
DROP TABLE dbo.tblNews
GO
EXECUTE sp_rename N'dbo.Tmp_tblNews', N'tblNews', 'OBJECT'
GO
ALTER TABLE dbo.tblNews ADD CONSTRAINT
PK_tblNews PRIMARY KEY CLUSTERED
(
    tblNewsId
) WITH( STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS =
ON) ON [PRIMARY]
GO
ALTER TABLE dbo.tblNews ADD CONSTRAINT
FK_tblNews_tblCategory FOREIGN KEY
(
    tblCategoryId
) REFERENCES dbo.tblCategory

```

```
(
tblCategoryId
) ON UPDATE NO ACTION
ON DELETE NO ACTION

GO
COMMIT
```

پس از آن مدل Entity Framework را باز کنید و در جایی از صفحه راست‌کلیک کرده و از منوی بازشده گزینه Update Model from Database را انتخاب کنید. سپس در پنجره بازشده، چون هیچ جدول، نما یا روالی به پایگاه داده‌ها نیفزوده ایم؛ دگمه‌ی Finish را کلیک کنید. دوباره کلاس tblNews را باز کنید. متوجه خواهید شد که همه‌ی DataContract‌ها و DataMember‌ها را حذف شده است. ممکن است بگویید می‌توانستیم کلاس یا مدل را تغییر دهیم و به وسیله‌ی Generate Database from Model به‌هنگام کنیم. ولی در نظر بگیرید که نیاز به ایجاد چندین جدول دیگر داریم و مدلی با ده‌ها Entity دارید. در این صورت همه‌ی تغییراتی که در کلاس داده ایم زدوده خواهد شد. در بخش پسین، درباره‌ی این‌که چه کنیم که عبارت‌هایی که به کلاس‌ها می‌افزاییم حذف نشود؛ خواهیم نوشت.

پیش از ادامه‌ی نوشتار بهتر است توضیحاتی درباره‌ی قالب‌های T4 داده شود. این قالب‌های مصنوعی حاوی کدهایی که است که هدف آن صرفه‌جویی در نوشتن کد توسط برنامه‌نویس است. مثلاً در MVC شما یکبار قالبی برای صفحه Index خود تهیه می‌کنید که برای نمونه بجای ساخت جدول ساده، از گرید Kendo استفاده کند و همچنین دارای دکمه ویرایش و جزئیات باشد. از این پس هر بار که نیاز به ساخت یک نمای نوع لیست برای یک ActionResult داشته باشید فرم ساز MVC از قالب شما استفاده خواهد کرد. روشن است که خود Visual Studio نیز از T4 در ساخت بسیاری از فرم‌ها و کلاس‌ها بهره می‌برد.

خبر خوب این‌که برای ساخت کلاس‌های هر موجودیت در Entity Framework نیز از قالب‌های T4 استفاده می‌شود و این‌که این قالب‌ها در دسترس توسعه‌دهندگان برای ویرایش یا افزودن است.

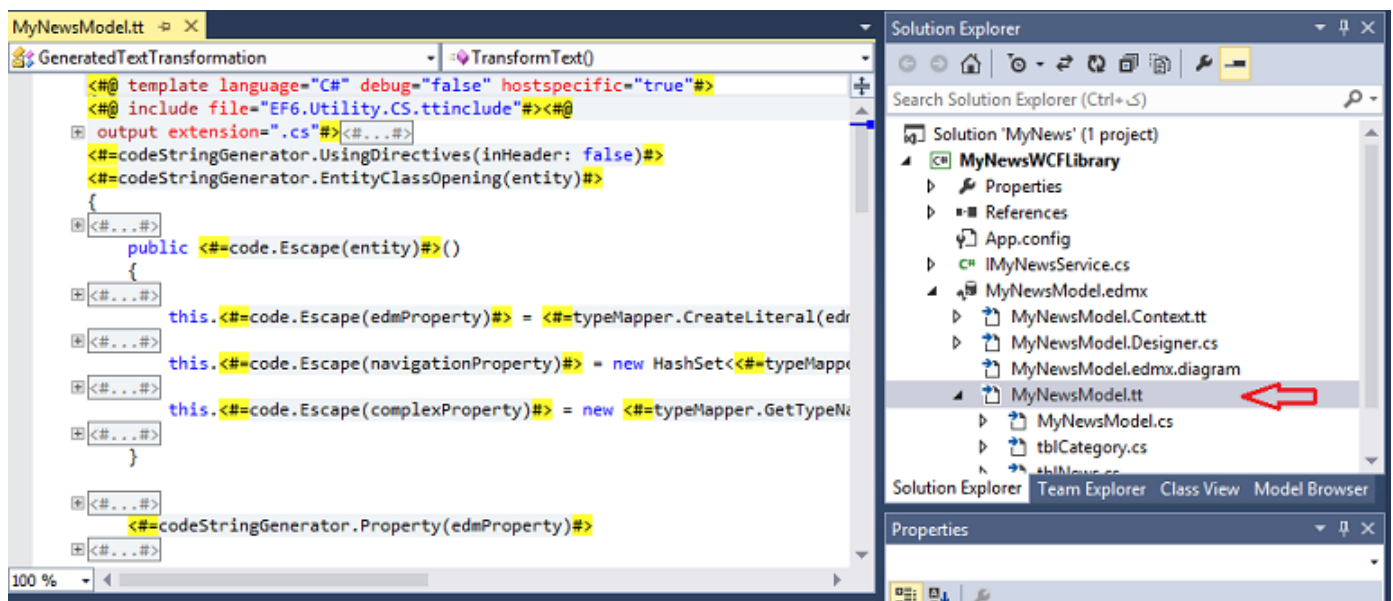
افزونه‌ی [Tangible](#) را دریافت کنید و سپس نصب کنید. این افزونه ظاهر نامفهوم قالب‌های T4 را ساده و روشن می‌کند. ما نیاز داریم که خود Visual Studio زحمت این سه کار را بکشد:

1- بالای هر کلاس موجودیت عبارت `using System.Runtime.Serialization` را بنویسید.

2- صفت `[DataContract]` را پیش از تعریف کلاس بیفزاید.

3- صفت `[DataMember]` را پیش از تعریف هر ویژگی بیفزاید.

همانند شکل زیر روی فایل `MyNewsModel.tt` دوکلیک کنید تا محتوای آن در سمت چپ نشان داده شود. این محتوا باید ظاهری همانند شکل پیدا کرده باشد:



کد زیر را در محتوای فایل جست‌وجو کنید:

```
public string Property(EdmProperty edmProperty)
{
    return string.Format(
        CultureInfo.InvariantCulture,
        "{0} {1} {2} {{ {3}get; {4}set; }}",
        Accessibility.ForProperty(edmProperty),
        _typeMapper.GetTypeName(edmProperty.TypeUsage),
        _code.Escape(edmProperty),
        _code.SpaceAfter(Accessibility.ForGetter(edmProperty)),
        _code.SpaceAfter(Accessibility.ForSetter(edmProperty)));
}
```

}

متن آن‌را به این صورت تغییر دهید:

```
public string Property(EdmProperty edmProperty)
{
    return string.Format(
        CultureInfo.InvariantCulture,
        "[DataMember]" + Environment.NewLine +
        "{0} {1} {2} {{ {3}get; {4}set; }}",
        Accessibility.ForProperty(edmProperty),
        _typeMapper.GetTypeName(edmProperty.TypeUsage),
        _code.Escape(edmProperty),
        _code.SpaceAfter(Accessibility.ForGetter(edmProperty)),
        _code.SpaceAfter(Accessibility.ForSetter(edmProperty)));
}
```

بار دیگر به دنبال این کد بگردید:

```
public string EntityClassOpening(EntityType entity)
{
    return string.Format(
        CultureInfo.InvariantCulture,
        "{0} {1}partial class {2}{3}",
        Accessibility.ForType(entity),
        _code.SpaceAfter(_code.AbstractOption(entity)),
        _code.Escape(entity),
        _code.StringBefore(" : ", _typeMapper.GetTypeName(entity.BaseType)));
}
```

این کد را نیز به این صورت تغییر دهید:

```
public string EntityClassOpening(EntityType entity)
{
    return string.Format(
        CultureInfo.InvariantCulture,
        "[DataContract]" + Environment.NewLine +
        "{0} {1}partial class {2}{3}",
        Accessibility.ForType(entity),
        _code.SpaceAfter(_code.AbstractOption(entity)),
        _code.Escape(entity),
        _code.StringBefore(" : ", _typeMapper.GetTypeName(entity.BaseType)));
}
```

برای واپسین تغییر به دنبال کد زیر بگردید:

```
public string UsingDirectives(bool inHeader, bool includeCollections = true)
{
    return inHeader == string.IsNullOrEmpty(_code.VsNamespaceSuggestion())
        ? string.Format(
            CultureInfo.InvariantCulture,
            "{0}using System;{1}" +
            "{2}",
            inHeader ? Environment.NewLine : "",
            includeCollections ? (Environment.NewLine + "using System.Collections.Generic;") : "",
            inHeader ? "" : Environment.NewLine)
        : "";
}
```

سپس کد زیر را جاگزین آن کنید:

```
public string UsingDirectives(bool inHeader, bool includeCollections = true)
{
    return inHeader == string.IsNullOrEmpty(_code.VsNamespaceSuggestion())
        ? string.Format(
            CultureInfo.InvariantCulture,
            "using System.Runtime.Serialization;" + Environment.NewLine +
```

```
{0}using System;{1}" +  
"{2}",  
inHeader ? Environment.NewLine : "",  
includeCollections ? (Environment.NewLine + "using System.Collections.Generic;") : "",  
inHeader ? "" : Environment.NewLine)  
: "";  
}
```

فایل MyNewsModel.tt را ذخیره کنید و از آن خارج شوید. بار دیگر هر کدام از کلاس‌های tblNews و tblCategory را باز کنید. خواهید دید که به صورت خودکار تغییرات مد نظر ما به آن افزوده شده است. از این پس بدون هیچ دلوپسی بابت حذف صفت‌ها، می‌توانید هرچند بار که خواستید مدل خود را به‌هنگام کنید. در بخش پسین دوباره به WCF بازخواهیم گشت و به تعریف روال‌های مورد نیاز خواهیم پرداخت.

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۴:۸ ۱۳۹۲/۱۰/۲۶

با تشکر از شما. روش دیگری برای حل مساله استفاده از AOP است:

[استفاده از IL Code Weaving برای تولید ویژگی‌های تکراری مورد نیاز در WCF](#)

نویسنده: حمید
تاریخ: ۴:۱ ۱۳۹۲/۱۰/۲۷

هرچند که به نکته خوبی، اشاره کردین اما این کار از اساس غلط است چون شما دارید کلاسهای لایه داده خود را expose می‌کنید. سرویس‌ها بادی DTOها را به بیرون EXPOSE کنند و تبدیل کلاسهای لایه BUSINESS به dtoها از طریق ابزاری مثل AUTOMAPPER انجام می‌شود. متشکرم

نویسنده: محسن خان
تاریخ: ۹:۲۸ ۱۳۹۲/۱۰/۲۷

بایدی وجود ندارد در این حالت و بهتر است که اینگونه باشد یا حتی مخلوطی از این دو در عمل:

[Pros and Cons of Data Transfer Objects](#)

In large projects with so many entities, DTOs add a remarkable level of (extra) complexity and work to do. In short, a pure, 100% DTO solution is often just a 100 percent painful solution

برای ادامه‌ی کار به لایه‌ی Interface بازمی‌گردیم. کلیه‌ی متدهایی که به آن نیاز داریم، نخست در این لایه تعریف می‌شود. در این‌جا نیز از قراردادهایی برای تعریف کلاس و روال‌های آن بهره می‌بریم که در ادامه به آن می‌پردازیم. پیش از آن باید بررسی کنیم، برای استفاده از این دو موجودیت، به چه متدهایی نیاز داریم. من گمان می‌کنم موارد زیر برای کار ما کافی باشد:

1- نمایش کلیه‌ی رکوردهای جدول خبر

2- انتخاب رکوردی از جدول خبر با پارامتر ورودی شناسه‌ی جدول خبر

3- درج یک رکورد جدید در جدول خبر

4- ویرایش یک رکورد از جدول خبر

5- حذف یک رکورد از جدول خبر

6- افزودن یک دسته

7- حذف یک دسته

8- نمایش دسته‌ها

هم‌اکنون به صورت زیر آن‌ها را تعریف کنید:

```
using System.Collections.Generic;
using System.Linq;
using System.ServiceModel;
using System.Text;
using System.Threading.Tasks;

namespace MyNewsWCFLibrary
{
    [ServiceContract]
    interface IMyNewsService
    {
        [OperationContract]
        List<tblNews> GetAllNews();

        [OperationContract]
        tblNews GetNews(int tblNewsId);

        [OperationContract]
        int AddNews(tblNews News);

        [OperationContract]
        bool EditNews(tblNews News);

        [OperationContract]
        bool DeleteNews(int tblNewsId);

        [OperationContract]
        int AddCategory(tblCategory News);

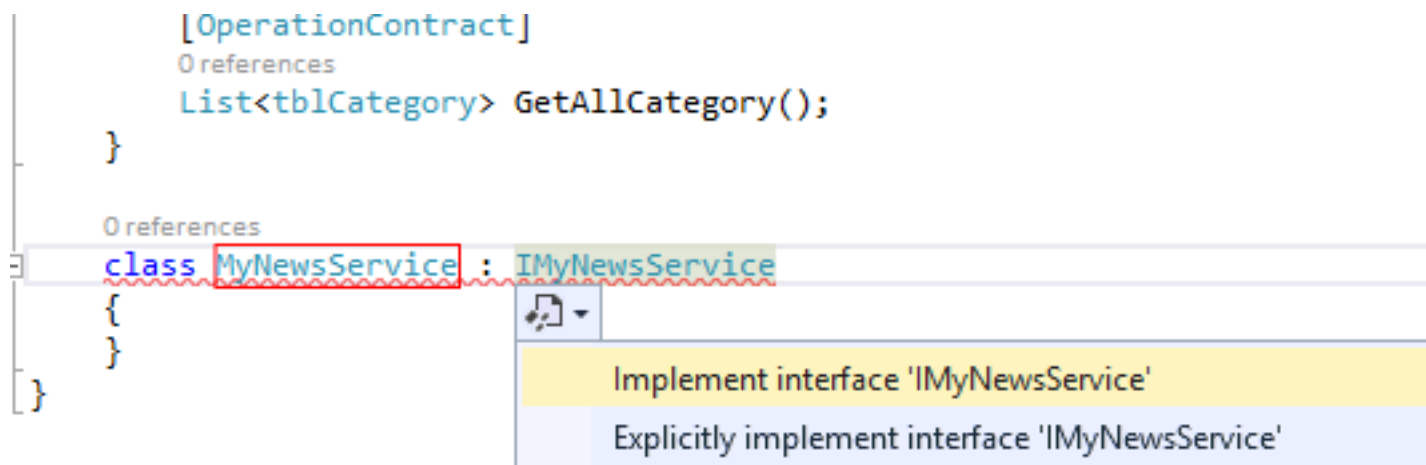
        [OperationContract]
        bool DeleteCategory(int tblCategoryId);

        [OperationContract]
        List<tblCategory> GetAllCategory();
    }
}
```

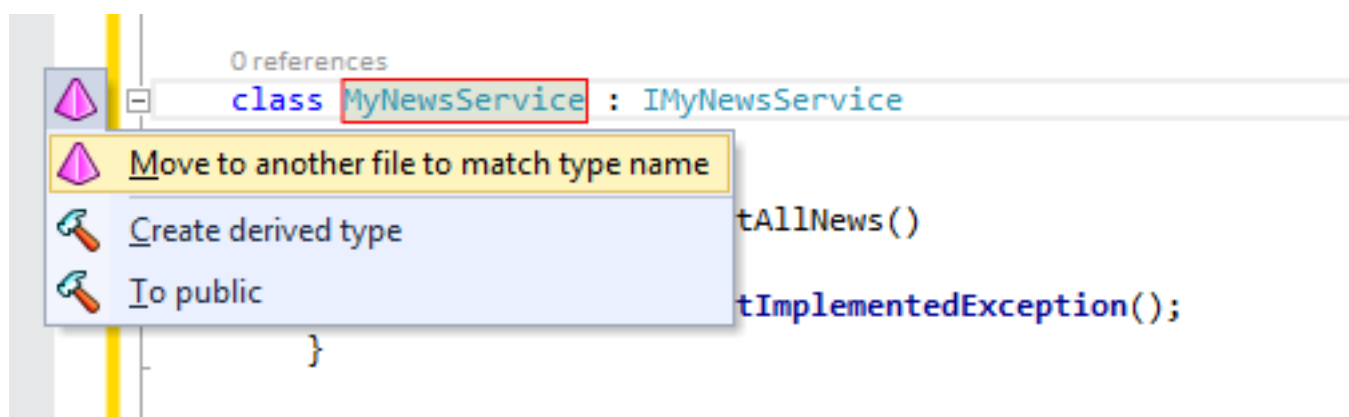
همان‌گونه که مشاهده می‌کنید از دو قرارداد جدید ServiceContract و OperationContract در فضای نام System.ServiceModel بهره برده ایم. ServiceContract صفتی است که بر روی Interface اعمال می‌شود و تعیین می‌کند که مشتری چه فعالیت‌هایی را روی سرویس می‌تواند انجام دهد و OperationContract تعیین می‌کند، چه متدهایی در اختیار قرار خواهند گرفت. برای ادامه‌ی کار نیاز است تا کلاس اجرا را ایجاد کنیم. برای این‌کار از ابزار Resharper بهره خواهیم برد: روی نام interface همانند شکل کلیک کنید و سپس برابر با شکل عمل کنید:



کلاسی به نام `MyNewsService` با ارث‌بری از `IMyNewsService` ایجاد می‌شود. زیر حرف `I` از `IMyNewsService` یک خط دیده می‌شود که با کلیک روی آن برابر با شکل زیر عمل کنید:



ملاحظه خواهید کرد که کلیه‌ی متدها برابر با `Interface` ساخته خواهد شد. اکنون همانند شکل روی نشان هرم شکلی که هنگامی که روی نام کلاس کلیک می‌کنید، در سمت چپ نشان داده می‌شود کلیک کنید و گزینه `Move to another file to match type` را انتخاب کنید:



به صورت خودکار محتوای این کلاس به یک فایل دیگر انتقال می‌یابد. اکنون هر کدام از متدها را به شکل دلخواه ویرایش می‌کنیم.
من کد کلاس را این‌گونه تغییر دادم:

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;

namespace MyNewsWCFLibrary
{
    class MyNewsService : IMyNewsService
    {
        private dbMyNewsEntities dbMyNews = new dbMyNewsEntities();
        public List<tblNews> GetAllNews()
        {
            return dbMyNews.tblNews.Where(p => p.IsDeleted == false).ToList();
        }

        public tblNews GetNews(int tblNewsId)
        {
            return dbMyNews.tblNews.FirstOrDefault(p => p.tblNewsId == tblNewsId);
        }

        public int AddNews(tblNews News)
        {
            dbMyNews.tblNews.Add(News);
            dbMyNews.SaveChanges();
            return News.tblNewsId;
        }

        public bool EditNews(tblNews News)
        {
            try
            {
                dbMyNews.Entry(News).State = EntityState.Modified;
                dbMyNews.SaveChanges();
                return true;
            }
            catch (Exception exp)
            {
                return false;
            }
        }

        public bool DeleteNews(int tblNewsId)
        {
            try
            {
                tblNews News = dbMyNews.tblNews.FirstOrDefault(p => p.tblNewsId == tblNewsId);
                News.IsDeleted = true;
                dbMyNews.SaveChanges();
                return true;
            }
            catch (Exception exp)
            {
                return false;
            }
        }
    }
}
```

```

    public int AddCategory(tblCategory Category)
    {
        dbMyNews.tblCategory.Add(Category);
        dbMyNews.SaveChanges();
        return Category.tblCategoryId;
    }

    public bool DeleteCategory(int tblCategoryId)
    {
        try
        {
            tblCategory Category = dbMyNews.tblCategory.FirstOrDefault(p => p.tblCategoryId ==
tblCategoryId);
            Category.IsDeleted = true;
            dbMyNews.SaveChanges();
            return true;
        }
        catch (Exception exp)
        {
            return false;
        }
    }

    public List<tblCategory> GetAllCategory()
    {
        return dbMyNews.tblCategory.Where(p => p.IsDeleted == false).ToList();
    }
}

```

ولی شما ممکن است دربارهی حذف، دوست داشته باشید رکوردها از پایگاه داده حذف شوند و نه این‌که با یک فیلد بولی آن‌ها را مدیریت کنید. در این صورت کد شما می‌تواند این‌گونه نوشته شود:

```

using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;

namespace MyNewsWCFLibrary
{
    class MyNewsService : IMyNewsService
    {
        private dbMyNewsEntities dbMyNews = new dbMyNewsEntities();
        public List<tblNews> GetAllNews()
        {
            return dbMyNews.tblNews.ToList();
        }

        public tblNews GetNews(int tblNewsId)
        {
            return dbMyNews.tblNews.FirstOrDefault(p => p.tblNewsId == tblNewsId);
        }

        public int AddNews(tblNews News)
        {
            dbMyNews.tblNews.Add(News);
            dbMyNews.SaveChanges();
            return News.tblNewsId;
        }

        public bool EditNews(tblNews News)
        {
            try
            {
                dbMyNews.Entry(News).State = EntityState.Modified;
                dbMyNews.SaveChanges();
                return true;
            }
            catch (Exception exp)
            {
                return false;
            }
        }

        public bool DeleteNews(tblNews News)
        {

```

```

        try
        {
            dbMyNews.tblNews.Remove(News);
            dbMyNews.SaveChanges();
            return true;
        }
        catch (Exception exp)
        {
            return false;
        }
    }

    public int AddCategory(tblCategory Category)
    {
        dbMyNews.tblCategory.Add(Category);
        dbMyNews.SaveChanges();
        return Category.tblCategoryId;
    }

    public bool DeleteCategory(tblCategory Category)
    {
        try
        {
            dbMyNews.tblCategory.Remove(Category);
            dbMyNews.SaveChanges();
            return true;
        }
        catch (Exception exp)
        {
            return false;
        }
    }

    public List<tblCategory> GetAllCategory()
    {
        return dbMyNews.tblCategory.ToList();
    }
}

```

البته باید در نظر داشته باشید که در صورت هر گونه تغییر در پارامترهای ورودی، لایه‌ی Interface نیز باید تغییر کند. گونه‌ی دیگر نوشتن متد حذف خبر می‌تواند به صورت زیر باشد:

```

public bool DeleteNews(int tblNewsId)
{
    try
    {
        tblNews News = dbMyNews.tblNews.FirstOrDefault(p => p.tblNewsId == tblNewsId);
        dbMyNews.tblNews.Remove(News);
        dbMyNews.SaveChanges();
        return true;
    }
    catch (Exception exp)
    {
        return false;
    }
}

```

در بخش 5 درباره‌ی تغییرات App.Config خواهیم نوشت.

پس از ایجاد متدها، نوبت به تغییرات App.Config می‌رسد. هرچند خود Visual Studio برای کلاس پیش‌گزیده‌ی خود تنظیماتی را در App.Config افزوده است ولی چنانچه در در خاطر دارید ما آن فایل‌ها را حذف کردیم و فایل‌های جدیدی به جای آن افزودیم. از این رو مراحل زیر را انجام دهید:

1- فایل App.Config را از Solution Explorer باز کنید.

2- به جای عبارت MyNewsWCFLibrary.Service1 در قسمت Service Name این عبارت را بنویسید:

MyNewsWCFLibrary.MyNewsService

3- در قسمت BaseAddress عبارت Design_Time_Addresses را حذف کنید.

4- در قسمت BaseAddress شماره پورت را به 8080 تغییر دهید.

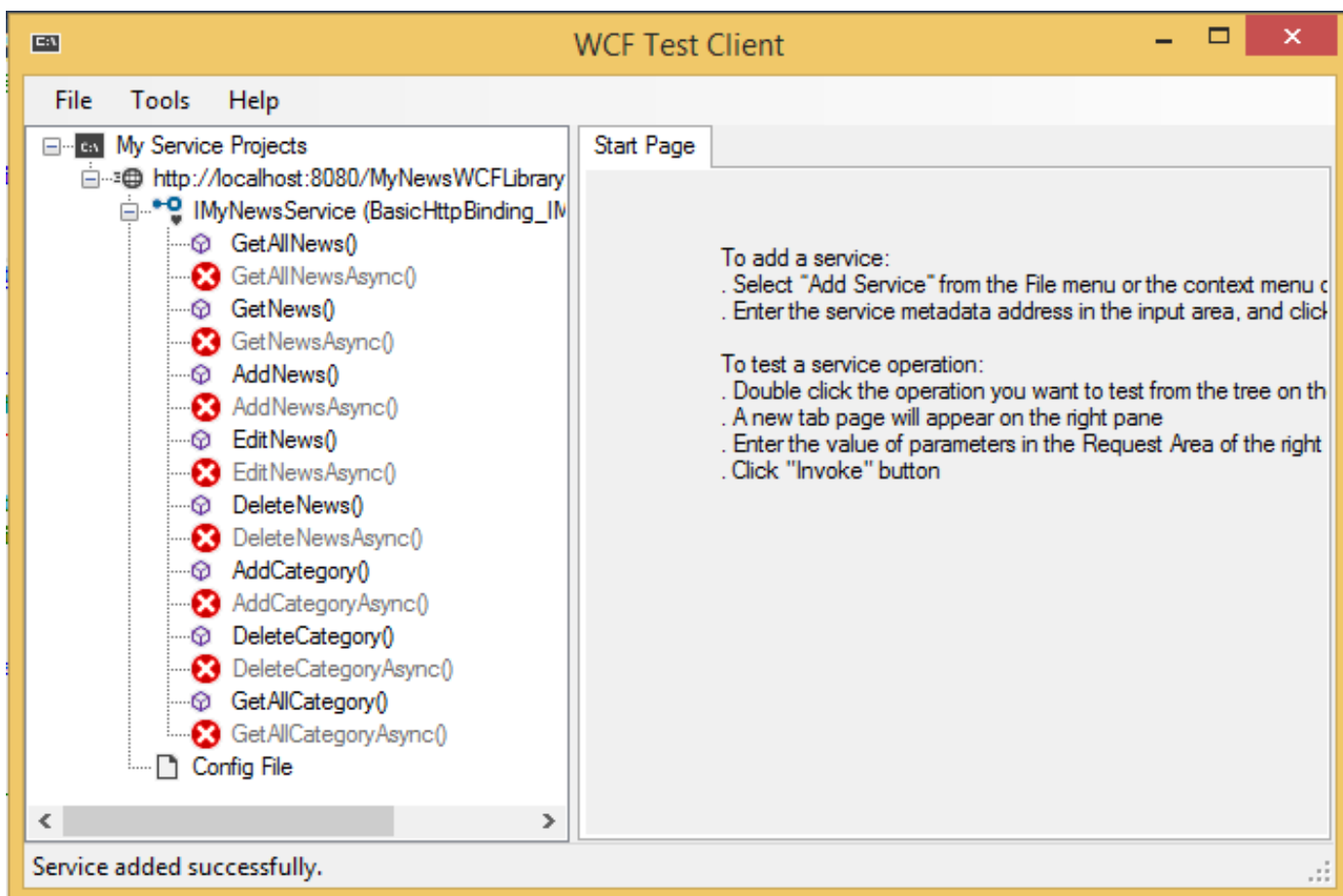
5- در قسمت BaseAddress به جای Service1 بنویسید: MyNewsService

6- در قسمت endpoint به جای عبارت MyNewsWCFLibrary.IService1 بنویسید: MyNewsWCFLibrary.IMyNewsService

در پایان تگ Service در App.Config باید همانند کد زیر باشد:

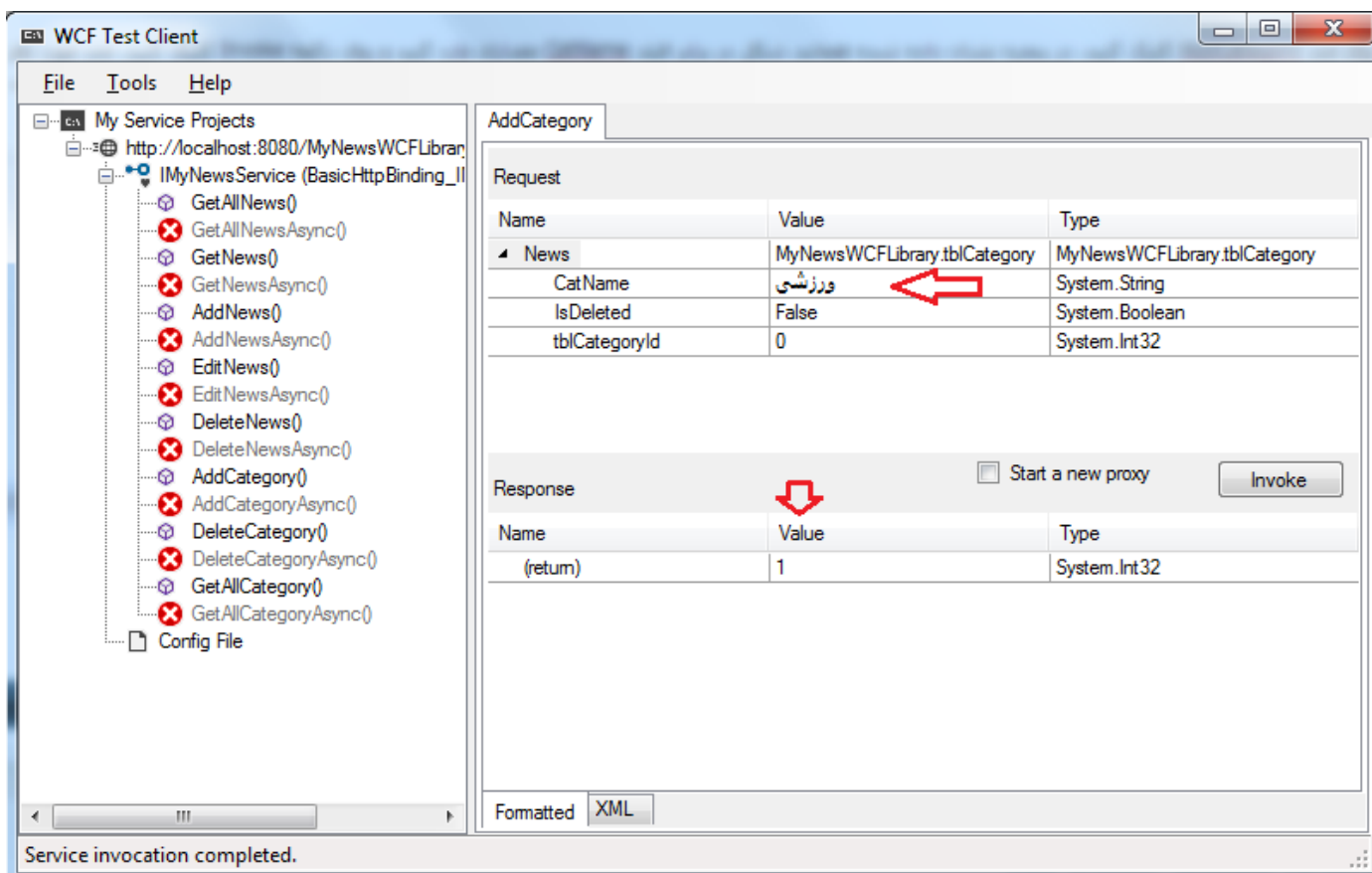
```
<services>
  <service name="MyNewsWCFLibrary.MyNewsService">
    <host>
      <baseAddresses>
        <add baseAddress="http://localhost:8080/MyNewsWCFLibrary/MyNewsService/" />
      </baseAddresses>
    </host>
    <!-- Service Endpoints -->
    <!-- Unless fully qualified, address is relative to base address supplied above -->
    <endpoint address="" binding="basicHttpBinding" contract="MyNewsWCFLibrary.IMyNewsService">
      <!--
        Upon deployment, the following identity element should be removed or replaced to reflect
        the identity under which the deployed service runs. If removed, WCF will infer an
        appropriate identity automatically.
      -->
      <identity>
        <dns value="localhost" />
      </identity>
    </endpoint>
    <!-- Metadata Endpoints -->
    <!-- The Metadata Exchange endpoint is used by the service to describe itself to clients. -->
    <!-- This endpoint does not use a secure binding and should be secured or removed before
    deployment -->
    <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
  </service>
</services>
```

تغییرات را ذخیره کنید و پروژه را اجرا کنید. باید پنجره‌ای شبیه به پنجره‌ی زیر نشان داده شود:

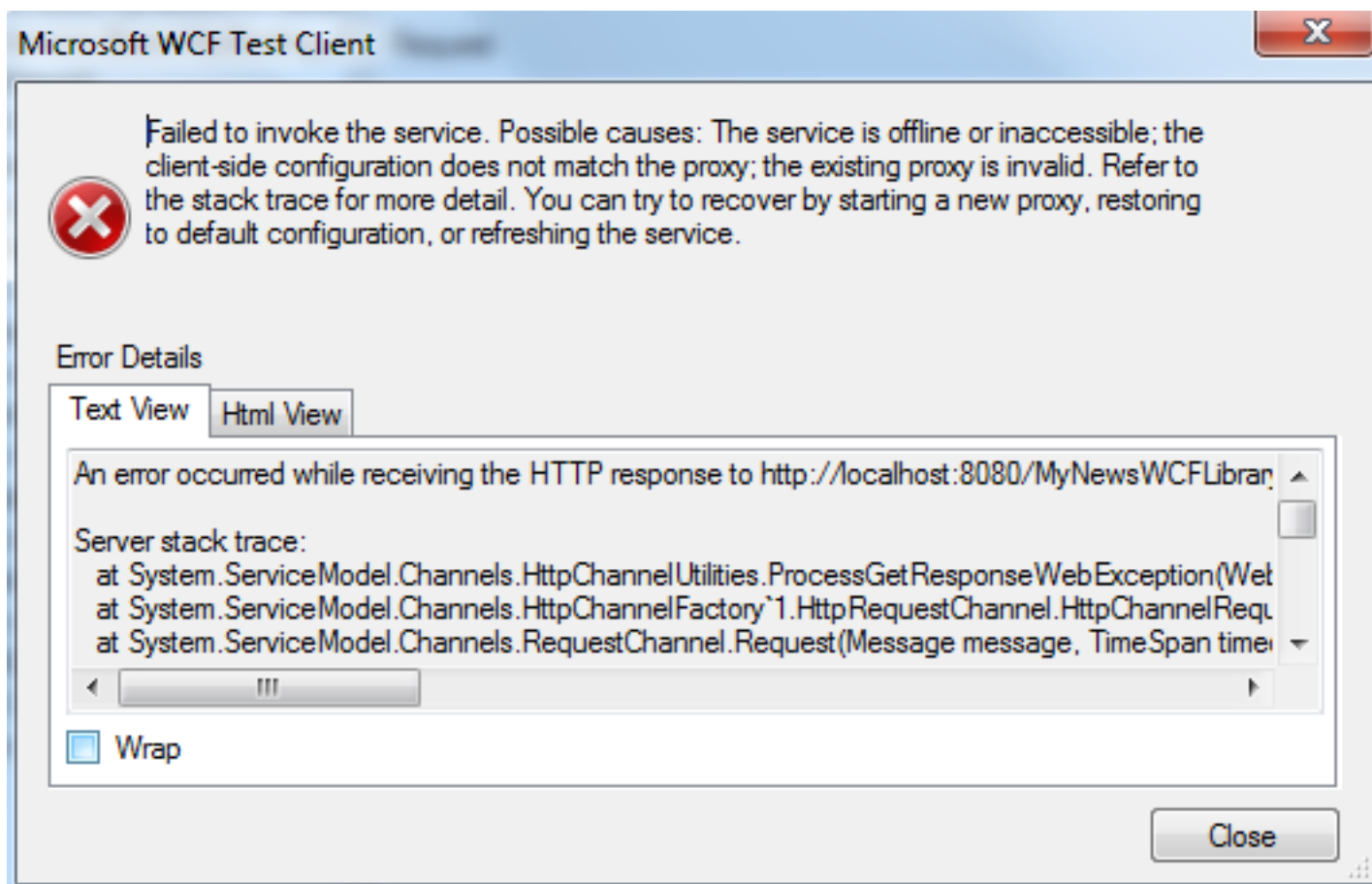


در صورت مشاهده پیام خطا، ویژوال استودیو را ببندید و این بار به صورت Run as administrator باز کنید.

برای نمونه روی متد AddCategory کلیک کنید. در پنجره نشان داده شده همانند شکل در برابر فیلد CatName مقداری وارد کنید و روی دکمه Invoke کلیک کنید. متد مورد نظر اجرا شده و مقداری که وارد کرده ایم در پایگاه داده‌ها ذخیره می‌شود. مقداری که در قسمت پایین دیده می‌شود خروجی متد است که در اینجا شناسه رکورد درج شده است.



بار دیگر برای مشاهده رکورد درج‌شده روی متد `GetAllCategory` کلیک کنید. به علت این‌که این متد ورودی ندارد در قسمت بالا چیزی نشان داده نمی‌شود. روی دکمه `Invoke` کلیک کنید. با پیغام خطای زیر روبه‌رو خواهید شد:



افزودن ویژگی Virtual به tblNews و tblCategory در [بخش دوم](#) خواندید؛ باعث می‌شود که Entity Framework در هنگام اجرا کلاس‌هایی با عنوان "پروکسی‌های پویا" به کلاس‌های Address و Customer بیفزاید و بنابراین قابلیت Lazy Loading برای این کلاس‌ها در زمان اجرای برنامه فراهم می‌گردد.

ولی با افزودن پروکسی‌های پویا به کلاس‌های ما، این کلاس‌ها قابلیت انتقال خود از طریق سرویس‌های WCF را از دست می‌دهند زیرا پروکسی‌های پویا به طور پیش‌گزینه قابلیت سریالایز و دیسریالایز شدن را ندارند!

خوشبختانه می‌توانیم این ویژگی را در کلاس DbContext غیرفعال کنیم. برای این منظور قالب سازنده‌ی آن یا MyNewsModel.Context.tt را از Solution Explorer باز کنید و کد زیر را در آن پیدا کنید:

```
<#Accessibility.ForType(container)#> partial class <#code.Escape(container)#> : DbContext
{
    public <#code.Escape(container)#>()
        : base("name=<#container.Name#>")
    {
```

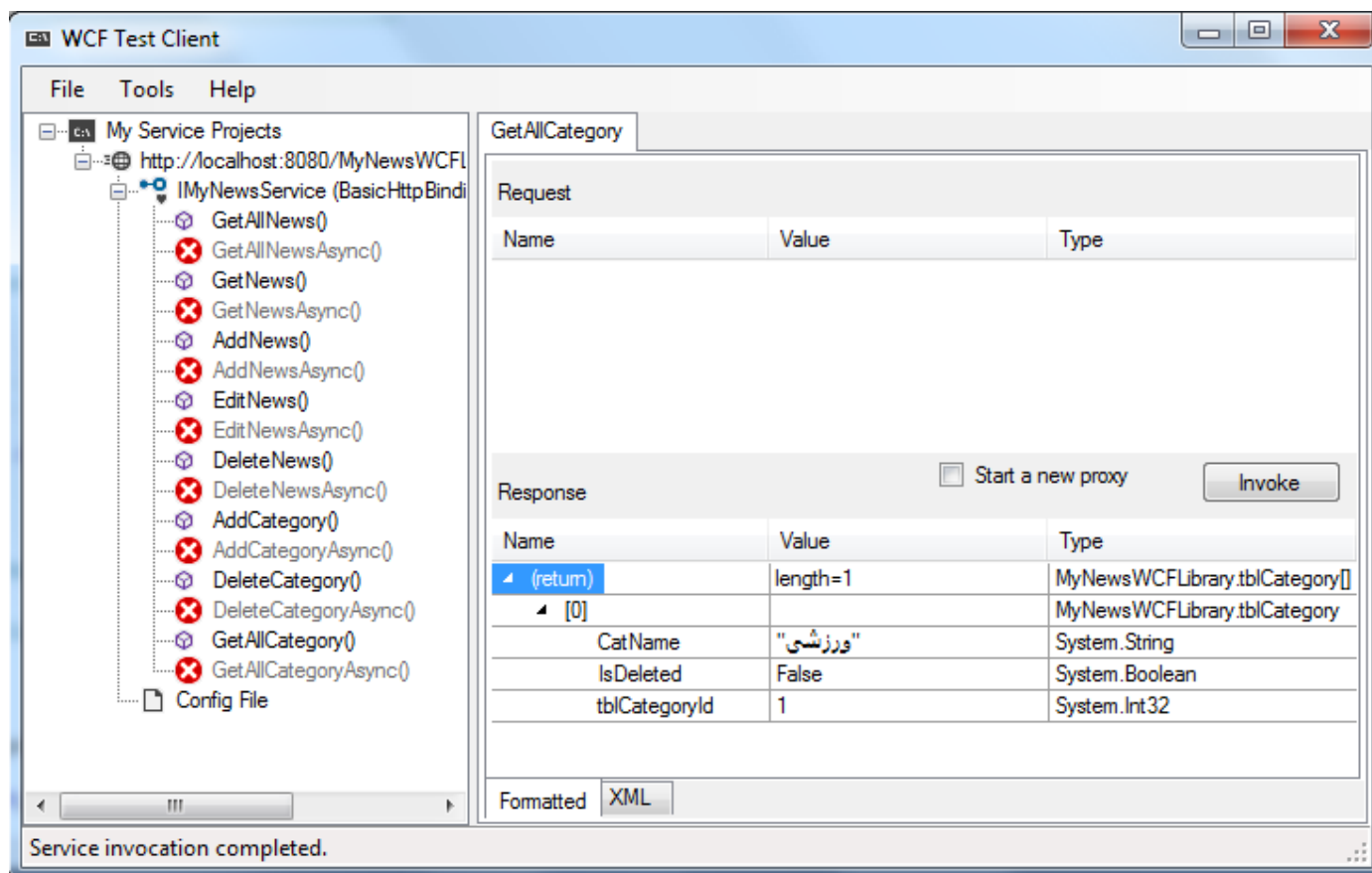
سپس در ادامه‌ی آن کد غیرفعال کردن پروکسی پویا را به این شکل بنویسید:

```
<#Accessibility.ForType(container)#> partial class <#code.Escape(container)#> : DbContext
{
    public <#code.Escape(container)#>()
        : base("name=<#container.Name#>")
    {
        Configuration.ProxyCreationEnabled = false;
```

اکنون اگر فایل را ذخیره کنیم سپس فایل MyNewsModel.Context.cs را از Solution Explorer باز کنید؛ خواهید دید که این خط

کد در جای خود قرار گرفته است.

بار دیگر پروژه را اجرا کنید روی متد GetAllCategory کلیک کنید. این بار اگر دکمه Invoke را بفشارید با همانند شکل زیر را خواهید دید:

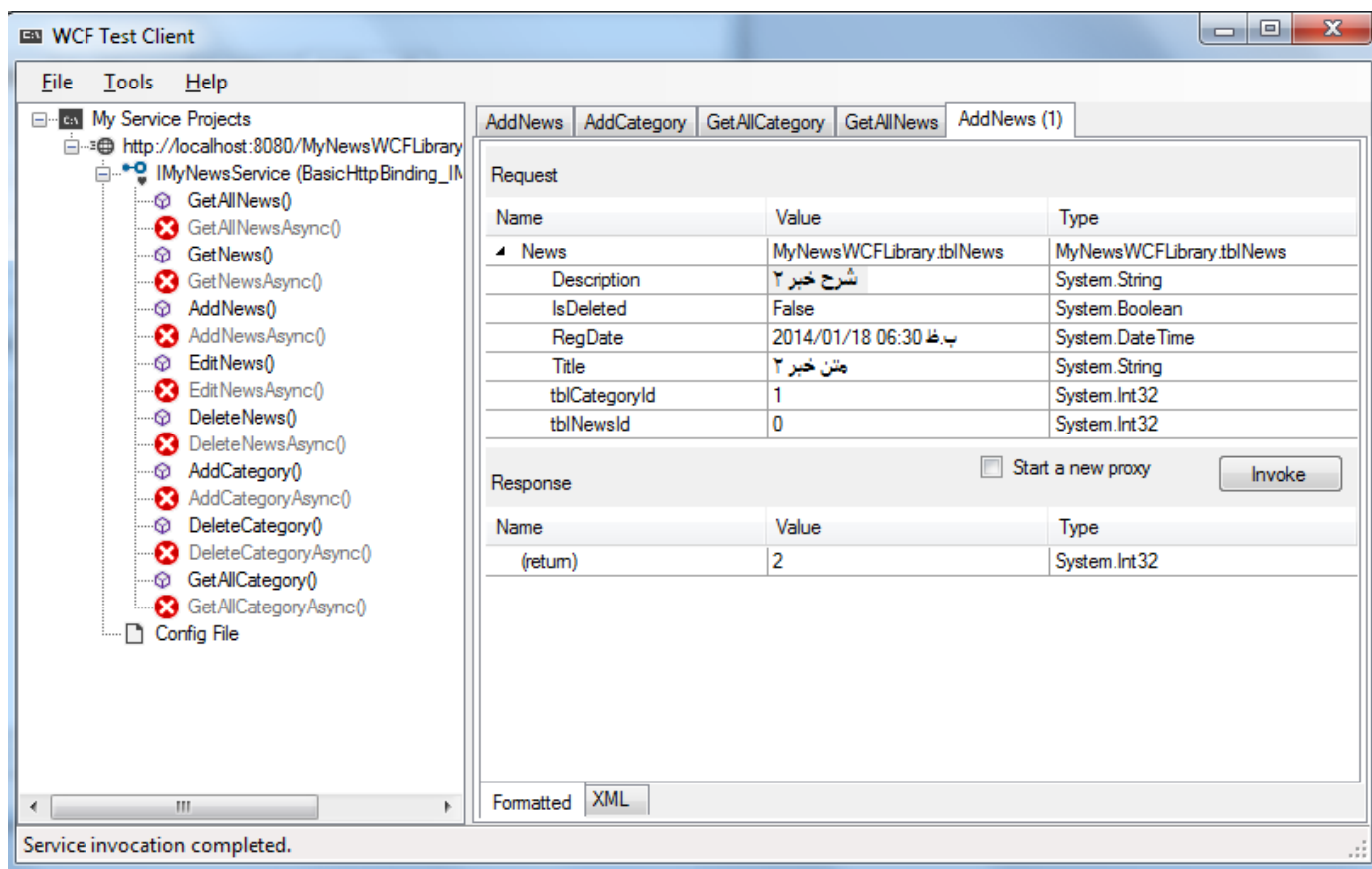


در بخش ششم پیرامون ارتباط جدول‌های tblNews و tblCategory و نمایش محتویات وابسته جدول خبر به دسته و تنظیمات آن در t4 و کلاس Service

در بخش هفتم پیرامون میزبانی WCFLibrary در یک Web Application

و در بخش هشتم پیرامون ایجاد یک برنامه‌ی ویندوزی جهت استفاده از سرویس‌های WCF خواهم نوشت.

پروژه را اجرا کنید و در WCF Test Client به وسیله‌ی متد AddNews دو خبر جدید درج کنید.



روی متدهای GetAllNews و GetAllCategory به صورت جداگانه کلیک کنید. متوجه خواهید شد که هرچند در کلاس tblNews شی‌ای از نوع tblCategory و در کلاس tblCategory شی‌ای از نوع مجموعه‌ی tblNews به صورت Virtual تعریف شده است ولی در بر خلاف انتظارمان اثری از آن در این‌جا دیده نمی‌شود. نتیجه‌ی مشاهده‌شده به خاطر است که در هر دو تعریف صفت DataMember را به ویژگی‌های ناوبری اختصاص نداده ایم و این می‌تواند راهبرد ما در طراحی WCF باشد. ولی اگر می‌خواهید ویژگی ناوبری میان موجودیت‌ها در متدهای ما هم دیده شود ادامه‌ی این درس را بخوانید و گرنه ممکن است تصمیم داشته باشید در صورت نیاز به پیوند میان موجودیت‌ها، متد جدیدی بنویسید و از دستورهای Linq استفاده کنید و یا برای این‌کار از Stored Procedured بهره ببرید.

در اینجا من این سناریو را دنبال می‌کنم که در صورتی که متد GetAllNews اجرا شود؛ بدون این‌که نیاز باشد برای دانستن نام دسته‌ی خبر از متد دیگری مانند GetAllCategory استفاده کنیم؛ رکورد وابسته موجودیت دسته در هر خبر نشان داده شود.

از Solution Explorer فایل MyNewsModel.tt را باز کنید و دنبال کد زیر بگردید:

```
public string NavigationProperty(NavigationProperty navigationProperty)
{
    var endType = _typeMapper.GetTypeName(navigationProperty.ToEndMember.GetEntityType());
    return string.Format(
```

```

        CultureInfo.InvariantCulture,
        "{0} {1} {2} {{ {3}get; {4}set; }}",
        AccessibilityAndVirtual(Accessibility.ForProperty(navigationProperty)),
        navigationProperty.ToEndMember.RelationshipMultiplicity == RelationshipMultiplicity.Many ?
        ("ICollection<" + endType + ">") : endType,
        _code.Escape(navigationProperty),
        _code.SpaceAfter(Accessibility.ForGetter(navigationProperty)),
        _code.SpaceAfter(Accessibility.ForSetter(navigationProperty)));
    }

```

سپس آن‌را به صورت زیر ویرایش کنید:

```

public string NavigationProperty(NavigationProperty navigationProperty)
{
    var endType = _typeMapper.GetTypeName(navigationProperty.ToEndMember.GetEntityType());
    return string.Format(
        CultureInfo.InvariantCulture,
        "{0}{1} {2} {3} {{ {4}get; {5}set; }}",
        navigationProperty.ToEndMember.RelationshipMultiplicity != RelationshipMultiplicity.Many ?
        "[DataMember]" + Environment.NewLine : "",
        AccessibilityAndVirtual(Accessibility.ForProperty(navigationProperty)),
        navigationProperty.ToEndMember.RelationshipMultiplicity == RelationshipMultiplicity.Many ?
        ("ICollection<" + endType + ">") : endType,
        _code.Escape(navigationProperty),
        _code.SpaceAfter(Accessibility.ForGetter(navigationProperty)),
        _code.SpaceAfter(Accessibility.ForSetter(navigationProperty)));
}

```

پس از ذخیره‌ی فایل، خواهید دید که صفت DataMember در کلاس tblNews پیش از ویژگی tblCategory افزوده شده است. بار دیگر پروژه را اجرا کنید. روی متد GetAllNews کلیک کنید و روی دکمه Invoke بفشارید. خواهید دید که هرچند tblCategory در ویژگی‌های آن قرار گرفته است ولی مقدار آن Null است. برای حل این مشکل باید از Solution Explorer فایل MyNewsService.cs را باز کنید و به به جای کد مربوط به متدهای GetAllNews و GetNews کدهای زیر را قرار دهید:

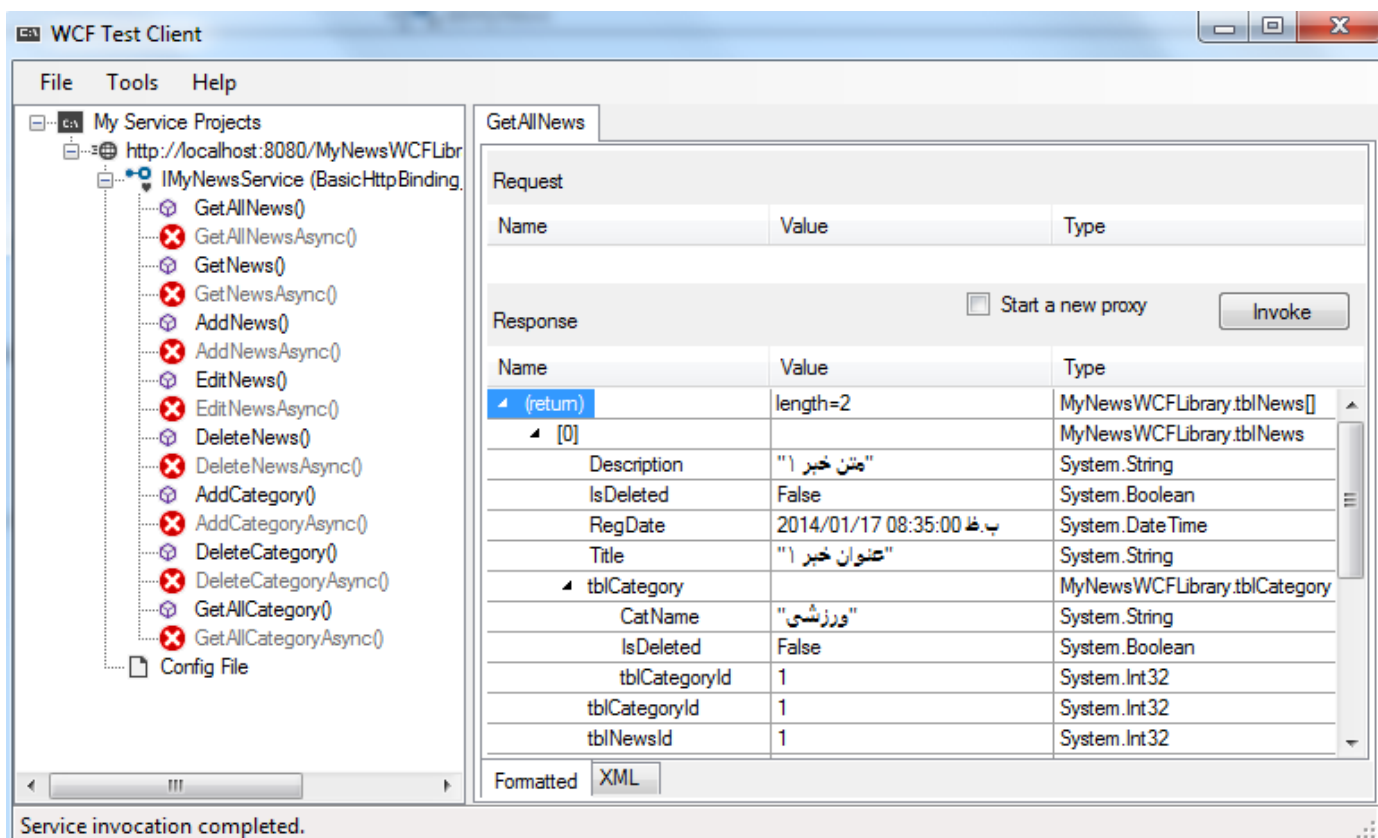
```

public List<tblNews> GetAllNews()
{
    return dbMyNews.tblNews.Include(p=>p.tblCategory).Where(c=>c.IsDeleted == false).ToList();
}

public tblNews GetNews(int tblNewsId)
{
    return dbMyNews.tblNews.Include(p => p.tblCategory).FirstOrDefault(p => p.tblNewsId ==
tblNewsId);
}

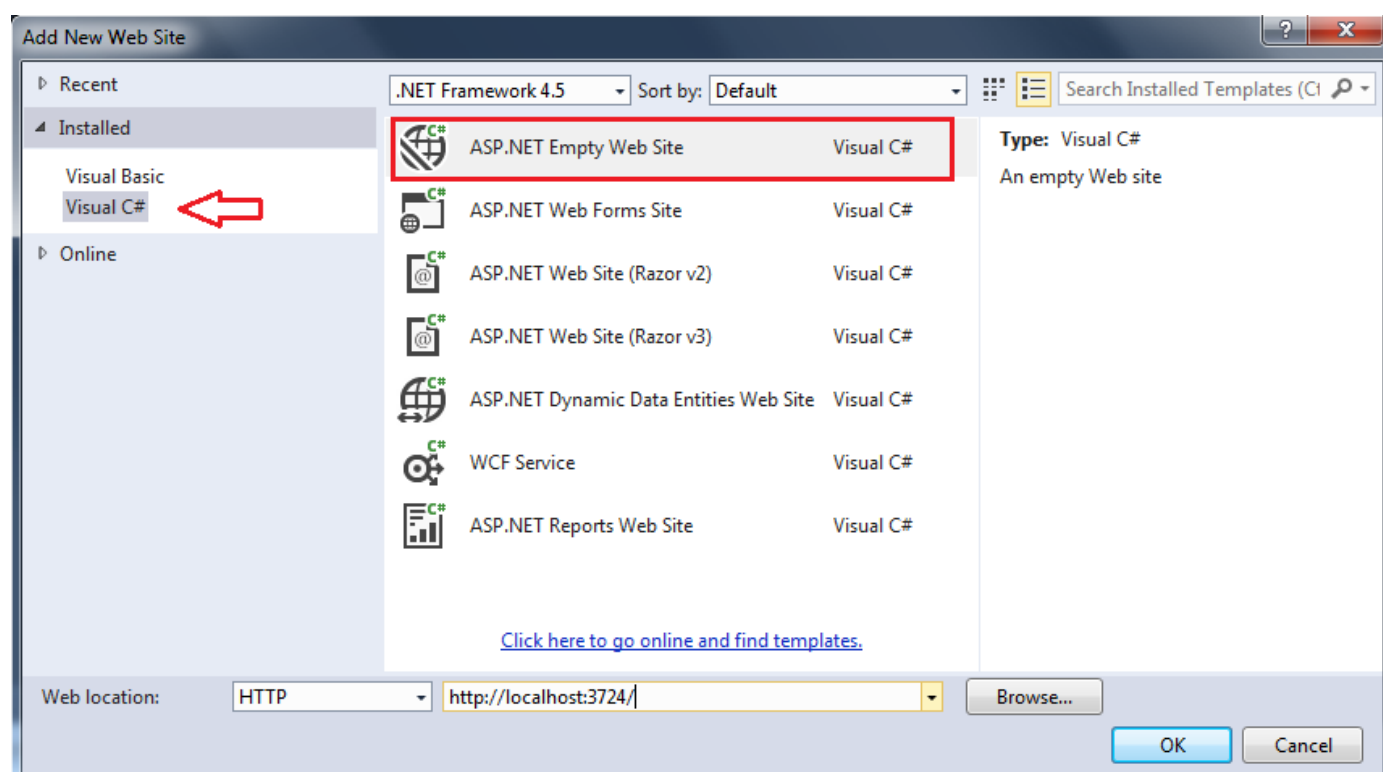
```

این بار اگر پروژه را اجرا کنید با نتیجه‌ای مانند شکل زیر روبه‌رو خواهید شد:

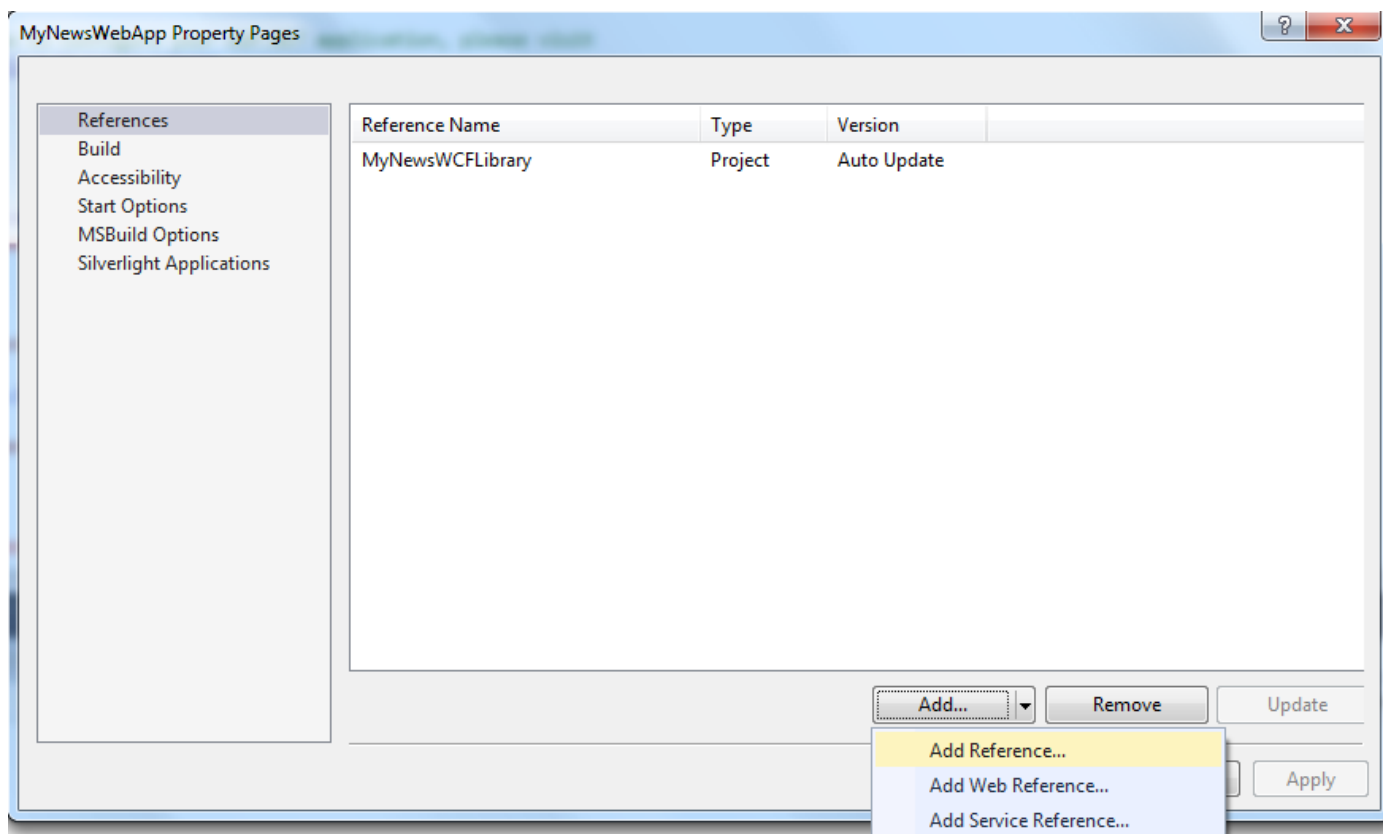


در بخش هفتم پیرامون میزبانی WCF Library خواهیم نوشت.

خروجی پروژه‌ی WCF Service Library یک فایل DLL است که هنگامی که با کنسول WCF Test Client اجرا می‌شود در آدرسی که در Web.Config تنظیم کرده بودیم اجرا می‌شود. اگر یک پروژه‌ی ویندوزی در همین راه حل بسازیم؛ خواهیم توانست از این آدرس برای دسترسی به WCF بهره ببریم. ولی اگر بخواهیم در IIS سرور قرار دهیم؛ باید در وبسایت آنرا میزبانی کنیم. برای این کار از Solution Explorer روی راه حل MyNews راست کلیک کنید و از منوی باز شده روی Add -> New Web Site کلیک کنید. سپس مراحل زیر را برابر با شکل‌های زیر انجام دهید:



سپس روی Web Site ایجادشده راست کلیک کنید و از منوی باز شده Property Pages را انتخاب کنید. روی گزینه‌ی Add Reference کلیک کنید، سپس پروژه‌ی MyNewsWCFLibrary را از قسمت Solution انتخاب کرده و دکمه‌ی OK را بفشارید.



دکمه‌ی OK را بفشارید و از Solution Explorer فایل Web.Config را باز کنید. پیش از تغییرات مد نظر باید چنین محتوایی داشته باشد:

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  For more information on how to configure your ASP.NET application, please visit
  http://go.microsoft.com/fwlink/?LinkId=169433
-->
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.5" />
    <httpRuntime targetFramework="4.5" />
  </system.web>
</configuration>
```

متن آنرا به این صورت تغییر دهید:

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  For more information on how to configure your ASP.NET application, please visit
  http://go.microsoft.com/fwlink/?LinkId=169433
-->
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.5" />
    <httpRuntime targetFramework="4.5" />
  </system.web>
  <system.serviceModel>
    <serviceHostingEnvironment>
      <serviceActivations>
        <add factory="System.ServiceModel.Activation.ServiceHostFactory"
relativeAddress="./HamedService.svc" service="MyNewsWCFLibrary.MyNewsService"/>
      </serviceActivations>
    </serviceHostingEnvironment>
  </system.serviceModel>
  <behaviors>
    <serviceBehaviors>
      <behavior>
```

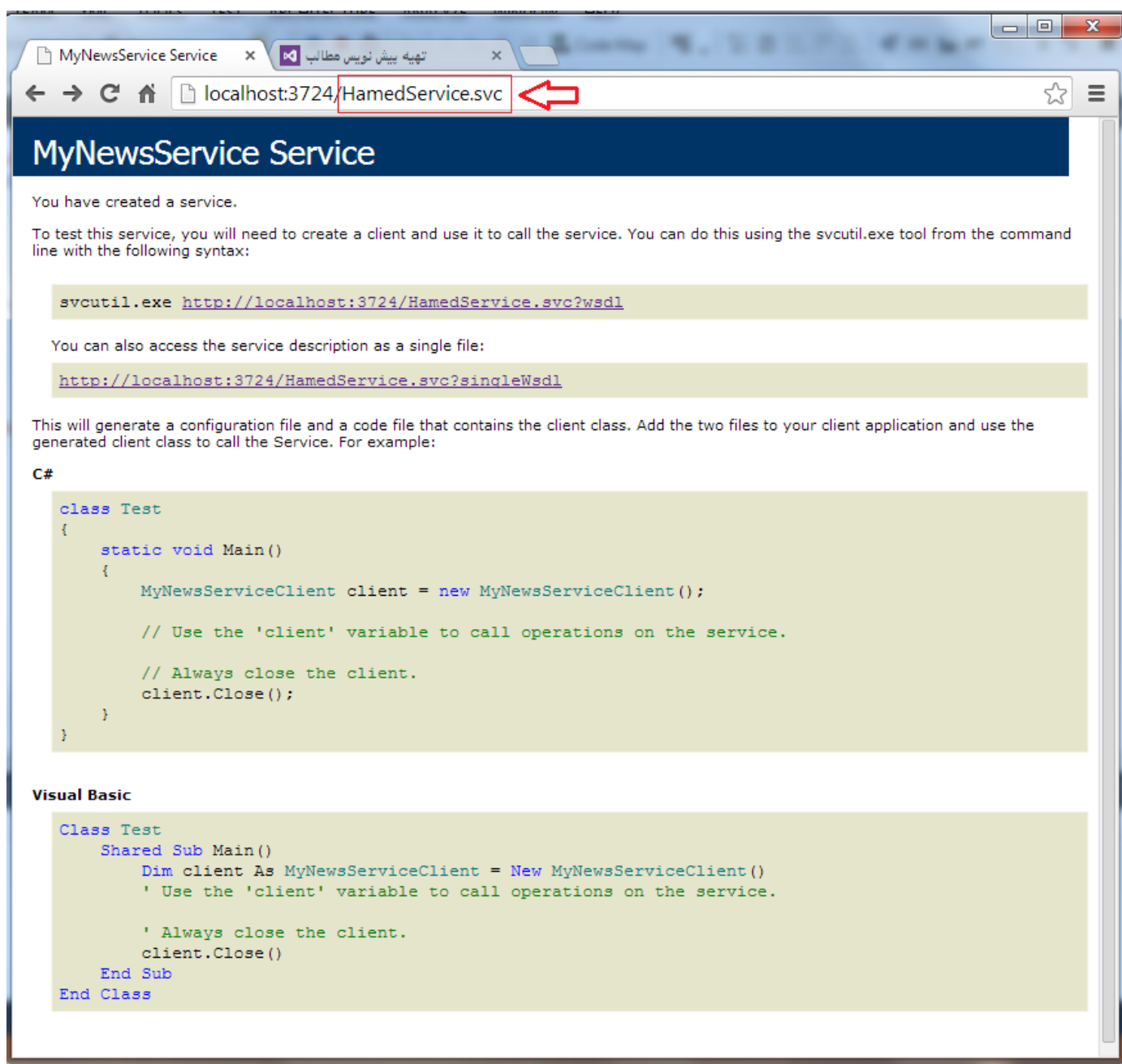
```

    <serviceMetadata httpGetEnabled="true"/>
  </behavior>
</serviceBehaviors>
</behaviors>
</system.serviceModel>
</configuration>

```

همان‌گونه که مشاهده می‌کنید به وسیله‌ی تگ add factory سرویس‌ها را به وب‌سایت معرفی می‌کنیم. با relativeAddress می‌توانیم هر نامی را به عنوان نام سرویس که در URL قرار می‌گیرد معرفی کنیم. چنان‌که من به جای MyNewsService از نام HamedService استفاده کردم. و در صفت service فضای نام و نام کلاس سرویس را معرفی می‌کنیم.

اکنون پروژه را اجرا کنید. در مرورگر باید صفحه را به این‌صورت مشاهده کنید:

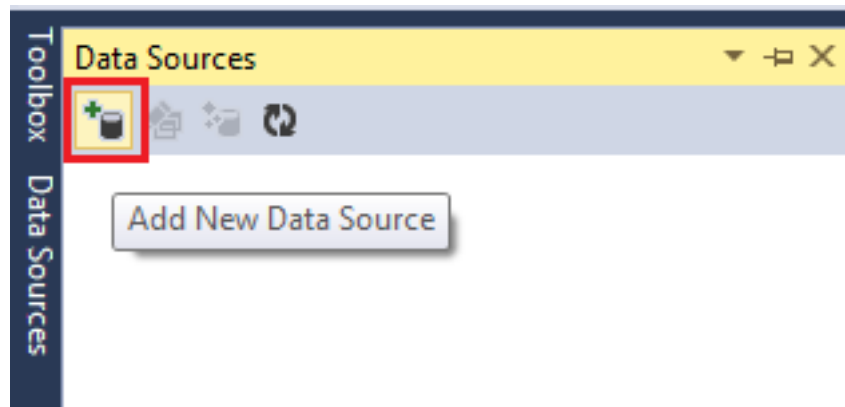


نیازی به یادآوری نیست که شما می‌توانید این پروژه را در IIS سرور راه‌اندازی کنید تا کلیه‌ی مشتری‌ها به آن دسترسی داشته باشند. هرچند پیش از آن باید امنیت را نیز در WCF برقرار کنید.

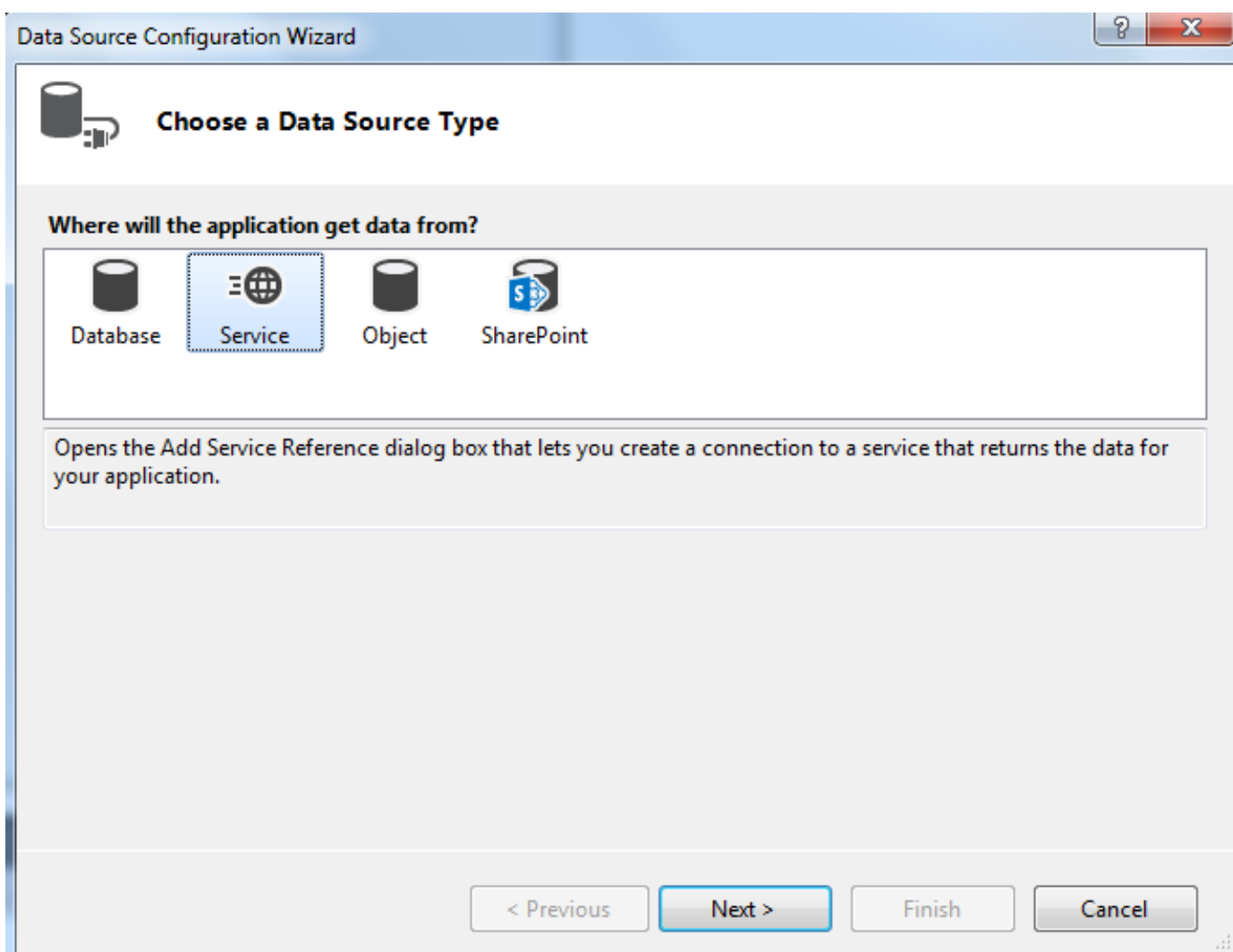
توجه داشته باشید که روشی که در این بخش به عنوان میزبانی WCF مطرح کردم یکی از روش‌های میزبانی WCF است. مثلاً شما می‌توانستید به جای ایجاد یک WCFLibrary و یک Web Site به صورت جداگانه یک پروژه از نوع WCF Service و یا Web Site ایجاد می‌کردید و سرویس‌ها و مدل Entity Framework را به طور مستقیم در آن می‌افزودید. روشی که در این درس از آن بهره برده ایم البته مزایایی دارد از جمله این‌که خروجی پروژه فقط یک فایل DLL است و با هر بار تغییر فقط کافی است همان فایل را در پوشه Bin از وب‌سایتی که روی سرور می‌گذارید کپی کنید.

در بخش هشتم با هم یک پروژه‌ی تحت ویندوز خواهیم ساخت و از سرویس WCF ای که ساخته ایم در آن استفاده خواهیم کرد.

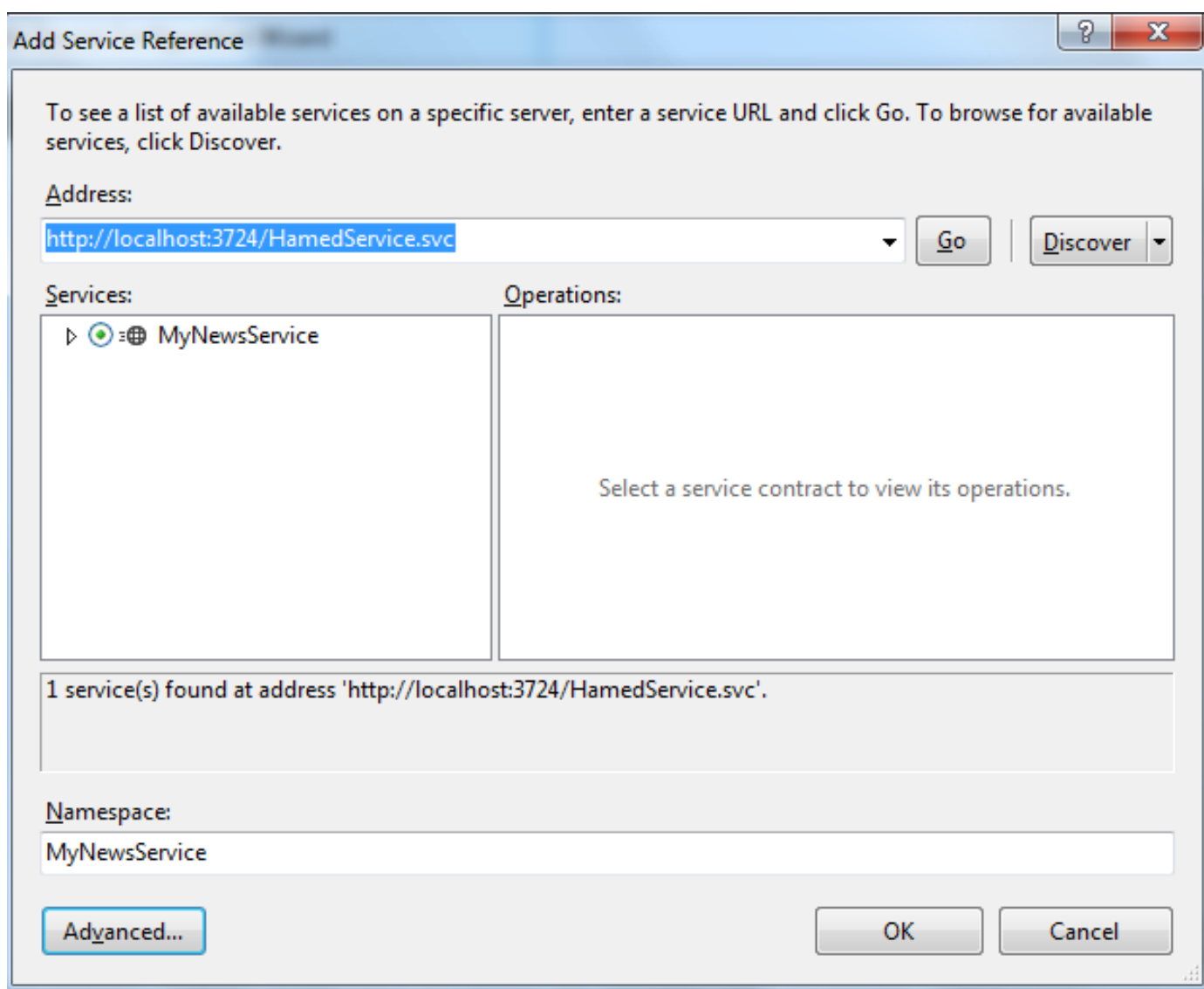
در Solution Explorer روی نام راه حل - MyNews - راست کلیک کنید و Add-> New Project را انتخاب کنید. سپس یک پروژه از نوع Windows Forms Application انتخاب کنید و نام آن را MyNewsWinApp بگذارید. یا کلیدهای ترکیبی Shift + Alt + D پنجره‌ی Data Sources را نمایان کنید. برابر با شکل روی ابزار Add New Data Source کلیک کنید:



از پنجره‌ی باز شده روی گزینه‌ی Service کلیک کنید:



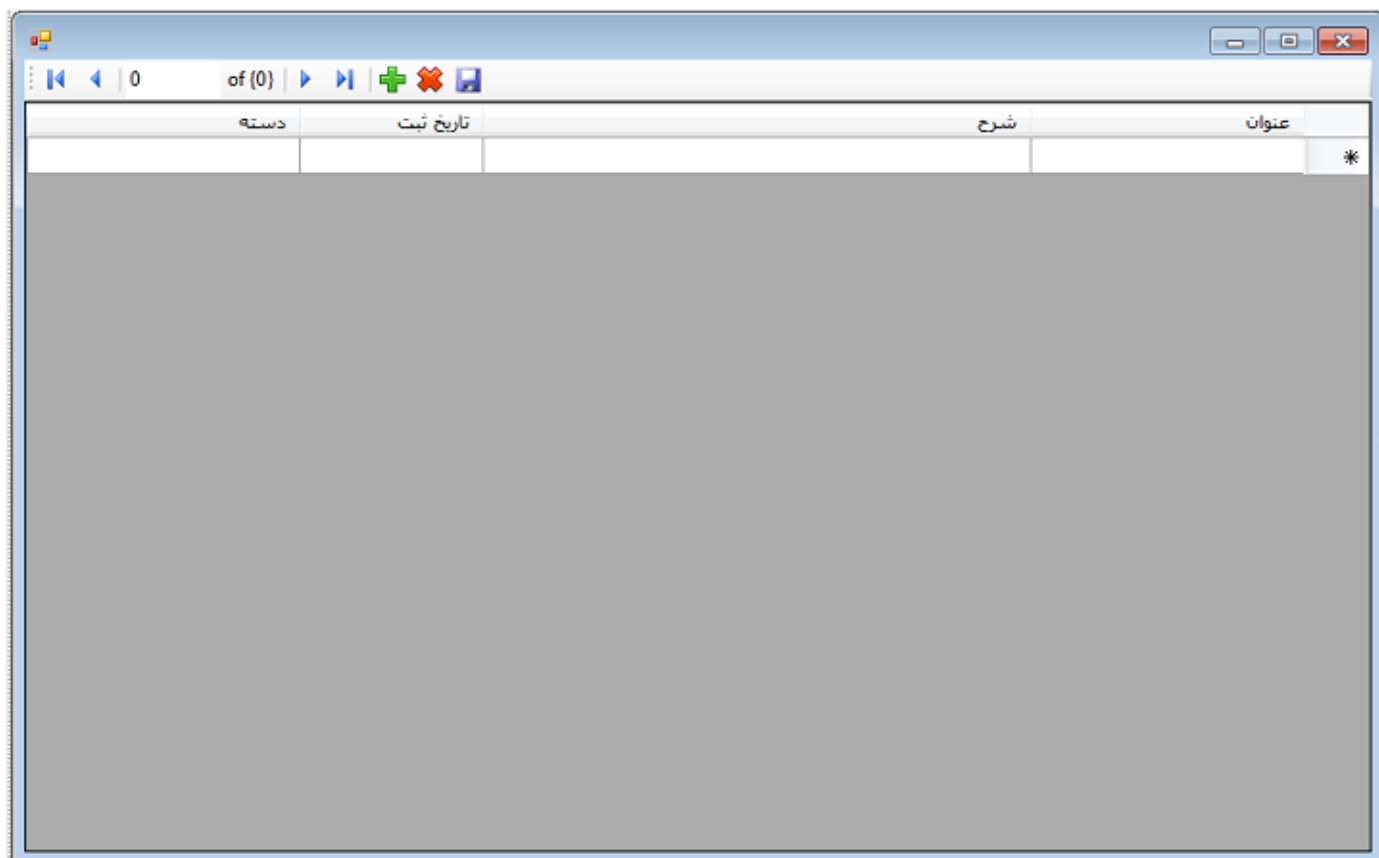
روی گزینه‌ی Next کلیک کنید و در پنجره‌ای که باز می‌شود در قسمت Address نشانی وب‌سایتی که در بخش پیشین تولید کردیم و ممکن است شما در IIS افزوده باشید؛ قرار دهید و روی دکمه‌ی GO بفشارید تا سرویس در کادر پایین افزوده شود. سپس در قسمت Namespace نامی مناسب برای فراخوانی سرویس وارد کنید آن‌گاه دکمه‌ی OK را بفشارید.



از پنجره‌ی بازشده روی دکمه‌ی Finish کلیک کنید. پس از مکثی کوتاه سرویس به همراه دو موجودیت آن درون Data Sources دیده خواهد شد. از آن‌طرف در Solution Explorer نیز در پوشه‌ی Service References سرویس تعریف‌شده ارجاع داده خواهد گرفت.

از Data Sources روی tblNews کلیک کنید سپس آن‌را کشیده و به روی فرم رها کنید. خواهید دید که یک DataGridView شامل همه‌ی ویژگی‌های موجودیت tblNews و یک Binding Navigator که با موجودیت tblNews در پیوند است و یک منبع داده به نام tblNewsBindingSource به صورت خودکار در فرم افزوده خواهد شد.

چیدمان فرم، رنگ‌ها، اندازه‌ها و فونت را آن‌گونه که می‌پسندید تنظیم کنید. سپس ستون‌هایی که به آن‌ها نیازی ندارید حذف یا پنهان کرده و عنوان ستون‌های مانده را ویرایش کنید. کلیدهای افزودن، حذف و ذخیره را روی Navigator ایجاد کنید و بقیه‌ی کلیدها را اگر به آن نیازی ندارید حذف کنید. البته می‌توانید بنا به سلیقه‌ی کاری‌تان یک Panel برای این‌کار اختصاص دهید. در این‌جا یک فرم ساده در نظر گرفته شده است:



اکنون نوبت به کدنویسی است. سورس فرم را باز کنید و نخست سرویس را به این صورت در جای مناسب تعریف کنید:

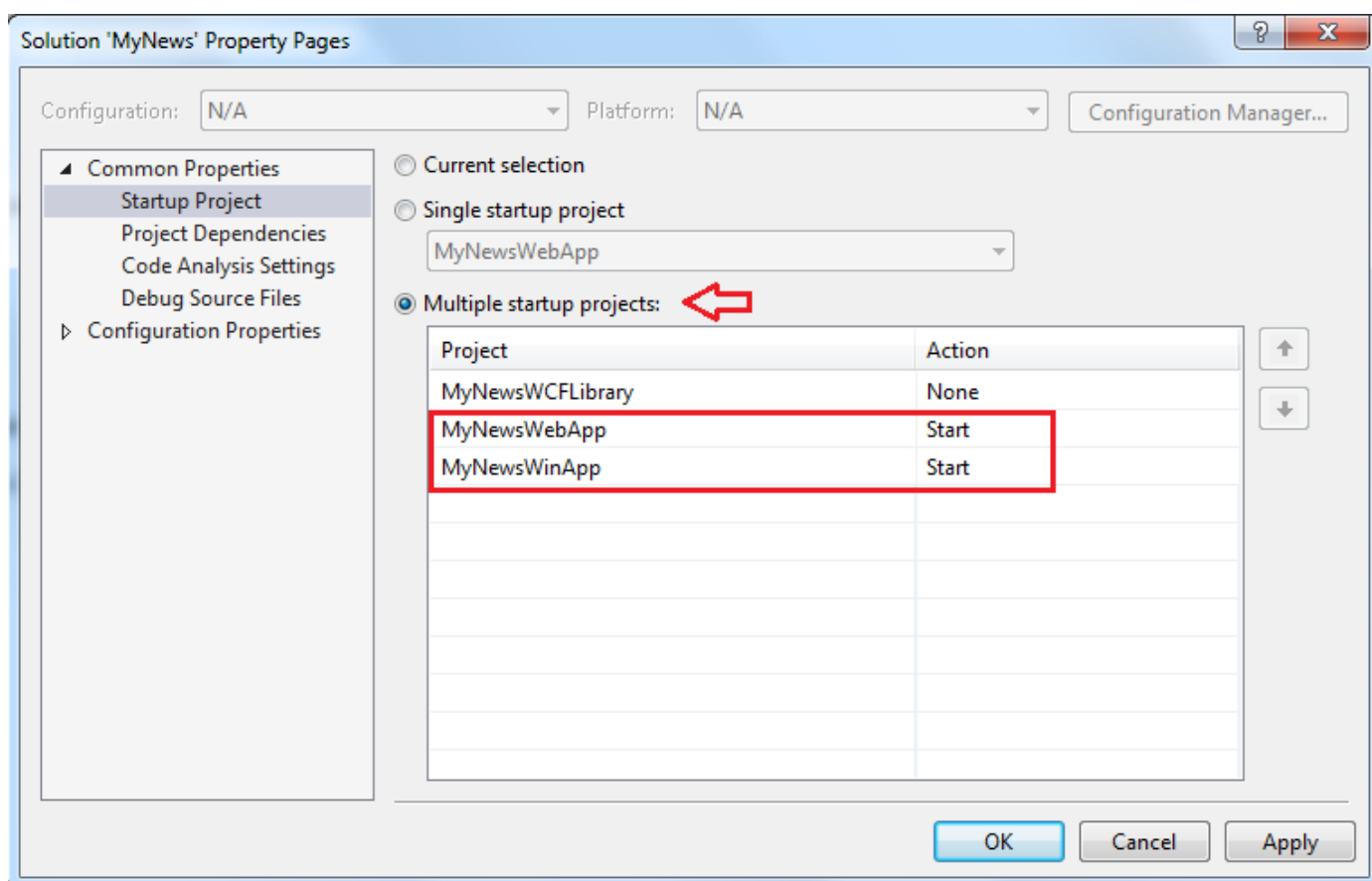
```
MyNewsService.MyNewsServiceClient MyNews = new MyNewsService.MyNewsServiceClient();
```

یک تابع کوچک برای تبدیل تاریخ میلادی به شمسی بنویسید سپس رویداد Load فرم را به این صورت بنویسید:

```
string MiladiToShamsi(DateTime MyDate)
{
    System.Globalization.PersianCalendar pers = new System.Globalization.PersianCalendar();
    return string.Format("{0}/{1}/{2}", pers.GetYear(MyDate),
        pers.GetMonth(MyDate).ToString("D2"), pers.GetDayOfMonth(MyDate).ToString("D2"));
}

private void Form1_Load(object sender, EventArgs e)
{
    tblNewsBindingSource.DataSource = MyNews.GetAllNews().Select(p => new {p.tblNewsId,
        p.tblCategory.CatName, p.Title, p.Description, RegDate= MiladiToShamsi( p.RegDate) });
}
```

پیش از اجرای پروژه از Solution Explorer روی نام راه حل راست کلیک کنید و گزینه‌ی Properties را انتخاب کنید. در پنجره‌ی باز شده تنظیمات زیر را انجام دهید:



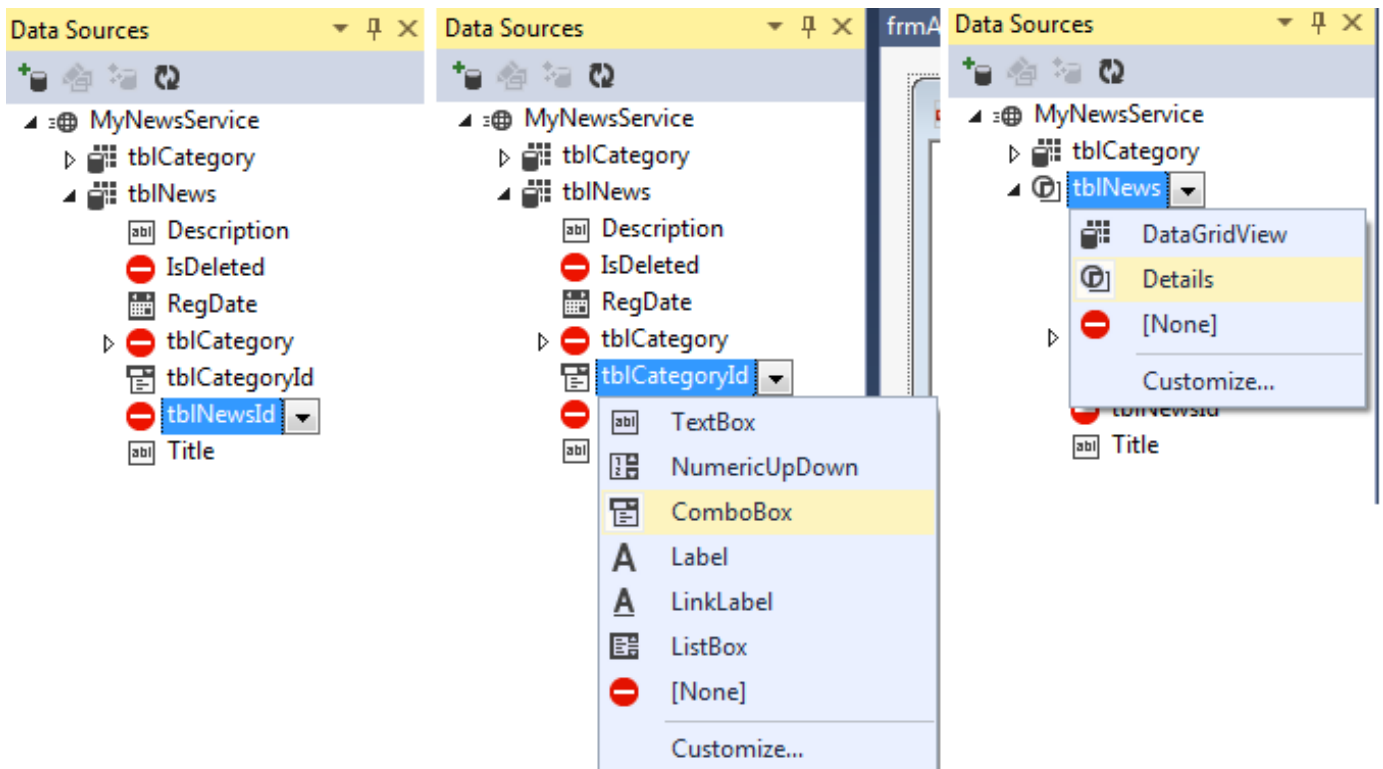
این کار باعث می‌شود که به طور هم‌زمان پروژه‌ی وب‌سایت و ویندوز اجرا شود. اکنون پروژه را اجرا کنید. اگر با پیغام خطا روبه‌رو شدید؛ تگ Connection String را از App.Config پروژه WCF Library به Web.Config پروژه وب‌سایت کپی کنید. در این صورت پروژه به راحتی اجرا خواهد شد.

عنوان	شرح	تاریخ ثبت	نام دسته
رئیس کمیته ملی المپیک انتخاب شد	کیومرث هاشمی رئیس کمیته ملی المپیک شد. به گزارش ایسنا، چهل‌ویکمین دوره انتخابات کمیته ملی المپیک پس از یک سال و نیم تاخیر از ساعت ۱۵ امروز (دوشنبه) آغاز شد که پس از رای‌گیری برای پست ریاست، کیومرث هاشمی به عنوان رئیس کمیته ملی المپیک انتخاب شد.	۱۳۹۲/۱۰/۲۷	ورزشی
صعود آسان استقلال و تراکتورسازی	تیم های فوتبال استقلال و تراکتورسازی، با غلبه بر حریفان خود به مرحله یک چهارم نهایی جام حذفی ایران صعود کردند. مس کرمان نیز نفت امیدیه را ۲ بر ۱ مغلوب کرد تا جمع تیم های صعود کننده به مرحله یک چهارم نهایی جام حذفی تکمیل شود.	۱۳۹۲/۱۰/۲۸	ورزشی

در بخش پسین پیرامون افزودن، ویرایش و حذف و برخی توضیحات برای توسعه‌ی کار خواهیم نوشت.

یک Windows Form جدید ایجاد کنید و نام آن را frmAddEditNews بگذارید.

برابر با شکل ویژگی‌های tblCategory، IsDeleted و tblNewsId را برابر با None کنید و tblCategoryId را از نوع Combobox انتخاب کنید. سپس با فشار فلش کنار tblNews گزینه‌ی Details را انتخاب کنید.



روی tblNews کلیک کرده آن را بکشید و روی فرم رها کنید. آنگاه ظاهر فرم و چیدمان کنترل‌ها را تنظیم کنید و دو دکمه ذخیره و لغو برابر با شکل در فرم ایجاد کنید:

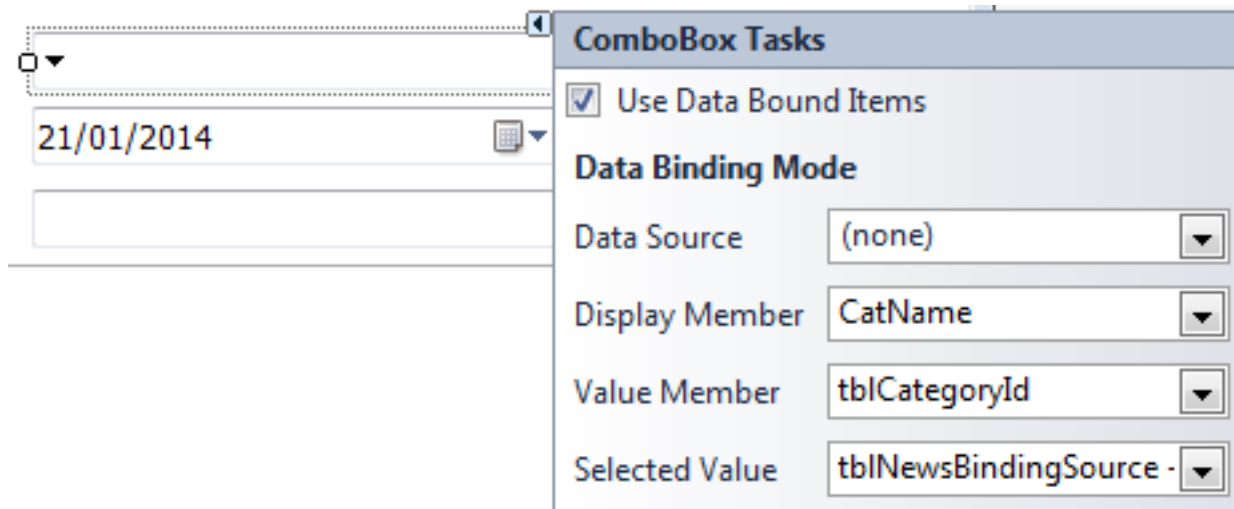
کد روی داد دو دکمه را این‌گونه بنویسید:

```
private void btnCancel_Click(object sender, EventArgs e)
{
    this.Close();
}

private void btnSave_Click(object sender, EventArgs e)
{
    this.DialogResult = System.Windows.Forms.DialogResult.OK;
}
```

در پایین فرم روی `tblNewsBindingSource` کلیک کنید و از قسمت `Properties` ویژگی `Modifiers` آن را برابر با `Public` کنید.

روی `Combobox` کلیک کنید، سپس ویژگی `Text -> DataBinding` آن را خالی کنید. سپس روی فلش بالای `Combobox` دسته خبر کلیک کنید و تنظیمات آن را مانند شکل زیر انجام دهید.



برای پرشدن آن کد زیر را در روی‌داد Load فرم این‌گونه بنویسید:

```
private void frmAddEditNews_Load(object sender, EventArgs e)
{
    MyNewsService.MyNewsServiceClient MyNews = new MyNewsService.MyNewsServiceClient();
    tblCategoryIdComboBox.DataSource = MyNews.GetAllCategory();
}
```

به فرم اصلی بازگردید و برای روی‌داد دکمه‌ی ویرایش چنین بنویسید:

```
private void btnEdit_Click(object sender, EventArgs e)
{
    if (tblNewsDataGridView.CurrentRow == null)
    {
        MessageBox.Show("سطری برای ویرایش انتخاب کنید");
    }
    else
    {
        //tblNews news = tblNewsDataGridView.CurrentRow.DataBoundItem as tblNews;
        tblNews news =
        MyNews.GetNews(Convert.ToInt32(tblNewsDataGridView.CurrentRow.Cells["tblNewsId"].Value));
        frmAddEditNews frmAdd = new frmAddEditNews();
        frmAdd.tblNewsBindingSource.DataSource = news;
        if (frmAdd.ShowDialog() == DialogResult.OK)
        {
            MyNews.EditNews(news);
            tblNewsBindingSource.DataSource = MyNews.GetAllNews().Select(p => new {
p.tblNewsId, p.tblCategory.CatName, p.Title, p.Description, RegDate = MiladiToShamsi(p.RegDate) });
        }
    }
}
```

در صورتی که متد GetAllNews را به صورت ساده به ویژگی DataSource دیتاگرید نسبت داده بودیم می‌توانستید از کد زیر برای مقاردهی به متغیر news بهره ببریم. ولی در حال حاضر این خط کد پیغام خطا می‌دهد. البته راه‌های دیگری برای حل این مشکل وجود دارد که در این درس قصد پرداختن به آن را ندارم.

```
tblNews news = tblNewsDataGridView.CurrentRow.DataBoundItem as tblNews;
```

کد مربوط به روی‌داد دکمه‌ی افزودن و حذف را نیز به صورت زیر بنویسید:

```
private void btnAdd_Click(object sender, EventArgs e)
{
    tblNews news = new tblNews();
    frmAddEditNews frmAdd = new frmAddEditNews();
}
```

```

        frmAdd.tblNewsBindingSource.DataSource = news;
        if (frmAdd.ShowDialog() == DialogResult.OK)
        {
            MyNews.AddNews(news);
            tblNewsBindingSource.DataSource = MyNews.GetAllNews().Select(p => new { p.tblNewsId,
p.tblCategory.CatName, p.Title, p.Description, RegDate = MiladiToShamsi(p.RegDate) });
        }

        private void btnRemove_Click(object sender, EventArgs e)
        {
            if (MessageBox.Show("هشدار", "آیا با حذف این سطر اطمینان دارید؟", MessageBoxButtons.YesNo) ==
System.Windows.Forms.DialogResult.Yes)
            {
                MyNews.DeleteNews(Convert.ToInt32(tblNewsDataGridView.CurrentRow.Cells["tblNewsId"].Value));
                tblNewsBindingSource.DataSource = MyNews.GetAllNews().Select(p => new { p.tblNewsId,
p.tblCategory.CatName, p.Title, p.Description, RegDate = MiladiToShamsi(p.RegDate) });
            }
        }
    }

```

برنامه را اجرا کنید. کار ما کم و بیش به پایان رسیده است. شما یک پروژه‌ی ویندوز ساده با استفاده از WCF ای که از Entity Framework برای اتصال به پایگاه داده بهره می‌برد؛ ایجاد کردید. WCF بسیار گسترده‌تر از این است و در این‌جا تنها به بخشی از آن پرداختیم. احتمالاً در صورت استقبال خوانندگان در آینده درباره‌ی تنظیمات ریز WCF برای امنیت، سرعت، محدودیت و استفاده در محیط‌های مختلف خواهیم نوشت.

شاد و پیروز باشید.

نظرات خوانندگان

نویسنده: وحید

تاریخ: ۱۳۹۲/۱۱/۰۳ ۶:۴۲

بسیار عالی بود

نویسنده: پژمان

تاریخ: ۱۳۹۲/۱۱/۰۳ ۱۸:۵۱

مرسی ، خیلی عالی بود. اگه میشه در مورد security in WCF مقاله بگذارید ممنون میشم. باز هم ممنون

نویسنده: پوریا منفرد

تاریخ: ۱۳۹۲/۱۱/۰۶ ۰:۴۱

سلام مطالب فوق العاده کاربردی هستند مشتاقانه منتظر ادامه این بحث هستیم
سوال:

در صورتی که بخوام از سرویس WCF روی یک سرور جدا استفاده کنم چطور با WinApp خودم به سرویس‌های WCF Server وصل شم؟ بدون واسطه Web App ؟
و اینکه سرعت واکنشی اطلاعات (رکوردهای زیاد 2 ، 3 هزارتا یا بیشتر) چگونه هست؟ با WCF و WinApp واسه نرم افزارهای سازمانی که تحت شبکه محلی و وایرلس داخل شهری هستن بخوام ازین روش استفاده شه آیا در بلند مدت با افزایش رکوردها به مشکل برخورد نمی‌کنم از نظر کار با دیتابیس و داده ها؟

نویسنده: پوریا منفرد

تاریخ: ۱۳۹۲/۱۱/۰۶ ۱:۱۸

تستی که من با تعداد رکوردها برای واکنشی از دیتابیس انجام دادم به یه مشکل برخورد کردم:
زمانی که تعداد رکوردها زیر 100 تا باشه خب win app به راحتی اطلاعات رو بارگزاری می‌کنه ولی وقتی بیش از این مقدار مثلا 288 رکورد در زمان اجرای پروژه به مشکل برخورد می‌کنم که فرم بارگزاری نمیشه و از حالت Start می‌پره بیرون
دلیلش چی می‌تونه باشه؟ محدودیت‌های وب سرویس؟ چطور و چگونه این مشکل رو برطرف کنیم؟
پیام Catch :

The maximum message size quota for incoming messages (65536) has been exceeded.
To increase the quota, use the MaxReceivedMessageSize property on the appropriate binding element.

از پیام معلومه که از حداکثر مقدار دریافتی یک واکنشی بیش از حد درخواست کردیم ...
یک راه حل جامع چی می‌تونه باشه؟

نویسنده: پوریا منفرد

تاریخ: ۱۳۹۲/۱۱/۰۶ ۱:۴۳

راه حلی که بنده پیدا کردم؛ تغییراتی در مقدار سائز پیام دریافتی به شکل زیر در appConfig مربوط به پروژه WinApp از این شکل :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <binding name="BasicHttpBinding_IMyNewsService" />
      </basicHttpBinding>
    </bindings>
  </system.serviceModel>
</configuration>
```

```
<client>
  <endpoint address="http://localhost:4636/SedaService.svc" binding="basicHttpBinding"
    bindingConfiguration="BasicHttpBinding_IMyNewsService"
contract="MyNewsService.IMyNewsService"
    name="BasicHttpBinding_IMyNewsService" />
</client>
</system.serviceModel>
</configuration>
```

به این شکل تغییر دهید :

```
<bindings>
  <basicHttpBinding>
    <binding maxReceivedMessageSize="2147483647" name="BasicHttpBinding_IMyNewsService" />
  </basicHttpBinding>
</bindings>
```

حالا امنیت رو نمیدونم اینجا نقض کردم یا نه؟ لطفا اگر اطلاعاتی دارید راهنمایی بفرمایید که امنیت نقض شده یا نه؟ و کلا با به عدد این شکلی که Max رو مشخص می‌کنه بنظرم نسبت به آینده نگری یک نرم افزار تجاری منطقی نیست...

نویسنده: حامد قنادی
تاریخ: ۱۳۹۲/۱۱/۰۶ ۶:۵۹

با درود و سپاس از همه‌ی همراهان.
همان‌سان که پیش‌تر هم نوشته ام می‌توانید سرویس WCF را در IIS یک سرور دیگر راه‌اندازی کنید و آدرس آی‌پی و یا DNS مربوط به آن‌را در WinApp خود استفاده کنید.
هنوز به تنظیمات خاص Web.Config نرسیده ایم در آن‌جا به امنیت و محدودیت‌ها خواهیم پرداخت.
پیروز باشید.

نویسنده: علیرضا طهوری
تاریخ: ۱۳۹۲/۱۲/۲۴ ۱۱:۰۱

سلام
با تشکر از این آموزش. فقط به خواهش دارم. اگر براتون مقدوره در مورد امنیت برای تبادل داده‌ها در wcf این مبحث رو ادامه بدید.

ممنون

نویسنده: حسین پاکدل
تاریخ: ۱۳۹۳/۰۱/۱۷ ۱۵:۱۴

با عرض سلام؛ آیا برای استفاده از یک وب سرویس هم باید مبحث "Dependency Injection" در نظر گرفته بشه؟ اگر پاسخ مثبت است لطفا با مثالی ساده توضیح دهید روش کار به چه صورت است؟ ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۱۷ ۱۶:۵۴

- برای تولید سرویس: « [پایاده سازی InstanceProvider برای سرویس‌های WCF](#) »
- برای استفاده از سرویس: در همان لایه سرویس برنامه از آن استفاده کنید. مباحث و مفاهیم تزریق وابستگی‌های آن [تفاوتی با حالت استفاده از یک دیتابیس یا یک WebClient ندارد و یکی است](#).

نویسنده: حسین پاکدل
تاریخ: ۱۳۹۳/۰۱/۲۶ ۱۴:۳۴

مشکلی که در استفاده از وب سرویس دارم اینه که وب سرویس در ازای بعضی از درخواست‌ها خطایی از نوع `System.ServiceModel.FaultException` بر میگردد. این خطا رو میتون در `Controller` با `HandleError` به `View` ی خاصی هدایت کرد. اما من قصد دارم پیام بازگردانده شده از نوع `FaultException` رو به کاربر نمایش بدم. برای این کار چه باید کرد؟ ممنون

نویسنده: وحید نصیری
تاریخ: ۱۴:۴۲ ۱۳۹۳/۰۱/۲۶

- به دلایل امنیتی نباید جزئیات خطاها را به کاربران نمایش داد. صرفا به نمایش صفحات و پیام‌های عمومی بسنده کنید.
+ در مورد MVC و مدیریت خطاها در آن بحث مجزایی در سایت وجود دارد ([^](#))؛ قسمت «دسترسی به اطلاعات استثناء در صفحه نمایش خطاها»

نویسنده: خلوت گزیده
تاریخ: ۹:۱ ۱۳۹۳/۰۴/۱۵

سلام ممنون از مطالب خوب و ارزشمندی که گذاشتید
فقط یه سوال دارم که هر چی گشتم نتونستم حل کنم
اونم نحوه پابلیش و خروجی گرفتن از برنامه برای IIS هست
ممنون میشم راهنمایی کنید که پروژه ای رو که ساختید چطوری میشه پابلیش کرد
بازهم ممنون

نویسنده: محسن خان
تاریخ: ۹:۸ ۱۳۹۳/۰۴/۱۵

- در قسمت هفتم، تنظیمات برنامه‌های وب آن بحث شده. پابلیش آن کپی و پیست پروژه در یک دایرکتوری مجازی در IIS است (یعنی فرقی با راه اندازی یک وب سایت معمولی ASP.NET نداره در اساس).
- اگر به خطایی برخوردید در این بین، عین خطا را ارسال کنید تا بیشتر بشود بحث کرد.

نویسنده: خلوت گزیده
تاریخ: ۱۲:۲۲ ۱۳۹۳/۰۴/۱۵

سلام
فکر می‌کنم ایراد از تنظیمات IIS ویندوز باشه و ربطی به برنامه نویسی نداره
اول که IIS تنظیم می‌کردم این Error میداد

HTTP Error 404.3 - Not Found The page you are requesting cannot be served because of the extension configuration. If the page is a script, add a handler. If the file should be downloaded, add a MIME map

که کارهایی که در وبلاگ زیر گفته شده انجام دادم

<http://blogs.msdn.com/b/ericwhite/archive/2010/05/11/getting-started-building-a-wcf-web-service.aspx>

الان پیغام زیر رو می‌ده

Server Error in '/MyNewService' Application.
Could not load type 'System.ServiceModel.Activation.HttpModule' from assembly 'System.ServiceModel, Version=3.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089'.

ممنون میشم اگه بتونی مشکل منو حل کنی
با تشکر

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۰۴/۱۵ ۱۲:۴۱

خطای آخری رو که ارسال کردید اینجا توضیح داده شده: <http://support.microsoft.com/kb/2015129>

خلاصه‌اش اینکه باید دستور `aspnet_regiis.exe /iru` رو در خط فرمان اجرا کنید. محل قرارگیری برنامه `aspnet_regiis.exe` در پوشه ویندوز هست (فایل‌ها رو جستجو کنید تا یافت بشه).

نویسنده: خلوت گزیده
تاریخ: ۱۳۹۳/۰۴/۱۵ ۱۳:۲۲

ممنون دوست عزیز
درضمن باید تنظیمات زیر رو هم اعمال کنید

Everywhere the problem to this solution was mentioned as re-registering aspNet by using `aspnet_regiis.exe`. But this did not work for me.
Though this is a valid solution (as explained beautifully here) but it did not work with Windows 8.
For Windows 8 you need to Windows features and enable everything under ".Net Framework 3.5" and ".Net Framework 4.5 Advanced Services".

بهره‌گیری از یک تابع پویا برای افزودن، ویرایش

در مثال‌های [گذشته](#) دیدید که برای هر کدام از عمل‌های درج، ویرایش و حذف، تابع‌های مختلفی نوشته بودیم که این کار هنگامی که یک پروژه‌ی بزرگ در دست داریم زمان‌بر خواهد بود. چه بسا یک جدول بزرگ داشته باشیم و بخواهیم در هر فرمی، ستون یا ستون‌های خاص به‌روزرسانی شوند. برای رفع این نگرانی افزودن تابع زیر به سرویس‌مان گره‌گشا خواهد بود.

```
public bool AddOrUpdateOrDelete<TEntity>(TEntity newItem, bool updateIsNull) where TEntity : class
{
    try
    {
        var dbMyNews = new dbMyNewsEntities();
        if (updateIsNull)
            dbMyNews.Set<TEntity>().AddOrUpdate(newItem);
        else
        {
            dbMyNews.Set<TEntity>().Attach(newItem);
            var entry = dbMyNews.Entry(newItem);
            foreach (
                var pri in newItem.GetType().GetProperties()
                    .Where(pri =>
                        (pri.GetGetMethod(false).ReturnParameter.ParameterType.IsSerializable &&
                         pri.GetValue(newItem, null) != null)))
            {
                entry.Property(pri.Name).IsModified = true;
            }
            dbMyNews.SaveChanges();
            return true;
        }
    }
    catch (Exception)
    {
        return false;
    }
}
```

این تابع دو پارامتر ورودی newItem و updateIsNull دارد که نخستین، همان نمونه‌ای از Entity است که قصد افزودن، ویرایش یا حذف آن را داریم و با دومی مشخص می‌کنیم که آیا ستون‌هایی که دارای مقدار null هستند نیز در موجودیت اصلی به‌هنگام شوند یا خیر. این پارامتر جهت رفع این مشکل گذاشته شده است که هنگامی که قصد به‌هنگام کردن یک یا چند ستون خاص را داشتیم و تابع update را به گونه‌ی زیر صدا می‌زدیم، بقیه‌ی ستون‌ها مقدار null می‌گرفت.

```
var news = new tblNews();
news.tblCategoryId = 2;
news.tblNewsId = 1;
MyNews.EditNews(news);
```

توسط تکه کد بالا، ستون tblCategoryId از جدول tblNews با شرط این که شناسه‌ی جدول آن برابر با 1 باشد، مقدار 2 خواهد گرفت. ولی بقیه‌ی ستون‌های آن به علت این که مقداری برای آن مشخص نکرده ایم، مقدار null خواهد گرفت. راهی که برای حل آن استفاده می‌کردیم، به این صورت بود:

```
var news = MyNews.GetNews(1);
news.tblCategoryId = 2;
MyNews.EditNews(news)
```

در این روش یک رفت و برگشت بی‌هوده به WCF انجام خواهد شد در حالتی که ما اصلاً نیازی به مقدار ستون‌های دیگر نداریم و اساساً کاری روی آن نمی‌خواهیم انجام دهیم.

در تابع AddOrUpdateOrDelete نخست بررسی می‌کنیم که آیا این که ستون‌هایی که مقدار ندارند، در جدول اصلی هم مقدار null بگیرند برای ما مهم است یا نه. برای نمونه هنگامی که می‌خواهیم سطر بی‌جدول بیفزاییم یا این که واقعاً بخواهیم مقدار دیگر

ستون‌ها برابر با null شود. در این صورت همان متد AddOrUpdate از Entity Framework اجرا خواهد شد. حالت دیگر که در حذف و ویرایش از آن بهره می‌بریم با یک دستور foreach همه‌ی پروپرتی‌هایی که Serializable باشد (که در این صورت پروپرتی‌های virtual حذف خواهد شد) و مقدار آن نامساوی با null باشد، در حالت ویرایش خواهند گرفت و در نتیجه دیگر ستون‌ها ویرایش نخواهد شد. این دستور دیدگاه جزءنگر دستور زیر است که کل موجودیت را در وضعیت ویرایش قرار می‌داد:

```
dbMyNews.Entry(news).State = EntityState.Modified;
```

با آن‌چه گفته شد، می‌توانید به جای سه تابع زیر:

```
public int AddNews(tblNews News)
{
    dbMyNews.tblNews.Add(News);
    dbMyNews.SaveChanges();
    return News.tblNewsId;
}

public bool EditNews(tblNews News)
{
    try
    {
        dbMyNews.Entry(News).State = EntityState.Modified;
        dbMyNews.SaveChanges();
        return true;
    }
    catch (Exception exp)
    {
        return false;
    }
}

public bool DeleteNews(int tblNewsId)
{
    try
    {
        tblNews News = dbMyNews.tblNews.FirstOrDefault(p => p.tblNewsId == tblNewsId);
        News.IsDeleted = true;
        dbMyNews.SaveChanges();
        return true;
    }
    catch (Exception exp)
    {
        return false;
    }
}
```

تابع زیر را بنویسید:

```
public bool AddOrEditNews(tblNews News)
{
    return AddOrUpdateOrDelete(News, News.tblNewsId == 0);
}
```

به همین سادگی. من در این‌جا شرط کردم فقط در حالت درج، از قسمت نخست تابع بهره گرفته شود. در سمت برنامه از این تابع برای عمل درج، ویرایش و حذف به سادگی و بدون نگرانی استفاده می‌کنید. برای نمونه جهت حذف در یک خط به این صورت می‌نویسید:

```
MyNews.AddOrEditNews (new tblNews { tblNewsId = 1, IsDeleted =true });
```

در بخش پسین آموزش، پیرامون ایجاد امنیت در WCF خواهیم نوشت.

نظرات خوانندگان

نویسنده: محمد آزاد
تاریخ: ۳:۲۵ ۱۳۹۳/۰۴/۲۸

به نظرتون این جواری اصل SRP رو نقض نکردیم؟

نویسنده: محسن خان
تاریخ: ۱۱:۴۱ ۱۳۹۳/۰۴/۲۸

خود EF متدی به نام AddOrUpdate داره: <http://msdn.microsoft.com/en-us/library/hh846520%28v=vs.103%29.aspx>

در اصل تک مسئولیتی، مسئولیت به دلیل تغییر یک کلاس ترجمه میشه. بنابراین در این اصل می‌گن که یک کلاس باید فقط یک دلیل برای تغییر داشته باشه. برای مثال کلاسی که هم اطلاعات گزارشی رو تهیه می‌کنه و هم اون رو پرینت می‌کنه، دو مسئولیت رو به عهده گرفته که میشه از هم جداشون کرد. اما در اینجا یک مسئولیت به روز رسانی اطلاعات یک موجودیت خاص بیشتر در کار نیست. دلیل دومی برای تغییر کلاس نداریم. وابستگی خارجی دومی نداره.