عنوان: **Building WCF REST Services**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۱/۰۵ ۱۶:۵۰:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: WCF

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: ۱۳۸۸/۰۲/۲۹ ۱۷:۲۷:۰۰
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 کاملا به موقع. دستت درد نکنه.

آقای نصیری به یک سری مطالب در مورد بالا بردن Performance سایت مانند استفاده از Cashing 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

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: ۱۳۸۸/۰۳/۱۷ ۱۴:۲۶:۰۵

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 پیشنیاز	دات نت سه	دات نت یک
هدف	جایگزینی یکپارچه‌ی فناورهای قبلی شامل WSE ، ASMX ، COM+ Eenterprise services و MSMQ ، .NET Remoting	ارائه وب سرویس
پروتکل‌های پشتیبانی شده	HTTP TCP Named pipes MSMQ Custom UDP	HTTP only
پشتیبانی از WS-* standards	بلی	خیر
پشتیبانی از اطلاعات باینری	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 می خواهم 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 لینکشوبدین؟
راستی اگر نداشتین لطفا چند کتاب یا منبع خوب انگلییسی 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.

عنوان: ASP.NET Web API - قسمت اول
نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۴/۱۱ ۸:۴
آدرس: www.dotnettips.info
گروه‌ها: WCF, ASP.NET Web API, ASMX, Web Service

بخش هایی از کتاب "مرجع کامل ASP.NET MVC (با پوشش کامل ASP.NET MVC 4)"
ترجمه و تالیف: بهروز راد
وضعیت: در دست چاپ

Web API چیست؟

Web API، نوع قالب جدیدی برای پروژه‌های مبتنی بر وب در .NET. است که بر مبنای اصول و الگوهای موجود در ASP.NET MVC ساخته شده است و همراه با ASP.NET MVC 4 وجود دارد. Web API توسعه گران را قادر می‌سازد تا با استفاده از یک الگوی ساده که در Controllerها پیاده سازی می‌شود، وب سرویس‌های مبتنی بر پروتوکل HTTP را با کدها و تنظیمات کم ایجاد کنند. این سبک جدید برای ایجاد وب سرویس ها، می‌تواند در انواع پروژه‌های .NET. مانند ASP.NET MVC، ASP.NET Web Forms، Windows Application و ... استفاده شود.

یک سوال کاملاً منطقی در اینجا به وجود می‌آید. چرا نیاز به بستری جدید برای ایجاد وب سرویس داریم؟ آیا در حال حاضر مایکروسافت بستری محبوب و فراگیر برای توسعه‌ی وب سرویس هایی که بتوانند با پروتوکل SOAP تعامل داشته باشند در اختیار ندارد؟ مگر وب سرویس‌های ASMX از زمان معرفی ASP.NET وجود نداشته اند؟ آیا تکنولوژی WCF مایکروسافت، بیشترین انعطاف پذیری و قدرت را برای تولید وب سرویس‌ها در اختیار قرار نمی‌دهد؟ وب سرویس‌ها جایگاه خود را یافته اند و توسعه گران با تکنولوژی‌های موجود به خوبی آنها را پیاده سازی و درک می‌کنند. چرا Web API؟

چرا Web Api؟

برای پاسخ به این سوال، باید برخی مشکلات را بررسی کنیم و ببینیم ابزارهای موجود چه راه حلی برای آنها در نظر گرفته اند. اگر با گزینه هایی که در ادامه می‌آیند موافق هستید، خواندن این مطلب را ادامه دهید، و اگر اعتقادی به آنها ندارید، پس نیازهای شما به خوبی با بسترهای موجود پاسخ داده می‌شوند.

من معتقد هستم که راه بهتری برای ایجاد وب سرویس‌ها وجود دارد.

من معتقد هستم که روش‌های ساده‌تری برای ایجاد وب سرویس‌ها وجود دارد و WCF بیش از حد پیچیده است.

من معتقد هستم که تکنولوژی‌های پایه‌ی وب مانند افعال GET، POST، PUT و DELETE برای انجام اعمال مختلف توسط وب سرویس‌ها کافی هستند.

اگر همچنان در حال خواندن این مطلب هستید، توضیحات خود را با شرح تفاوت میان Web API و تکنولوژی‌های دیگر هم حوزه‌ی آن ادامه می‌دهیم و خواهید دید که استفاده از Web API چقدر آسان است.

تفاوت Web API و WCF

وب سرویس‌های ASMX تا چندین سال، انتخاب اول برای ایجاد وب سرویس‌های مبتنی بر پروتوکل SOAP با استفاده از پروتوکل HTTP بودند. وب سرویس‌های ASMX، از وب سرویس‌های ساده که نیاز به قابلیت تعامل پایین داشتند و در نتیجه به پروتوکل SOAP نیز وابسته نبودند پشتیبانی نمی‌کردند. WCF جای وب سرویس‌های ASMX را گرفت و خود را به عنوان آخرین و بهترین روش برای ایجاد وب سرویس‌ها در بستر .NET. معرفی کرد. نمونه ای از یک سرویس WCF بر مبنای پروتوکل HTTP در .NET. به صورت ذیل است.

```
[ServiceContract]
public interface IService1
{
    [OperationContract]
    string GetData(int value);
    [OperationContract]
    CompositeType GetDataUsingDataContract(CompositeType composite);
}
...
public class Service1 : IService1
{
    public string GetData(int value)
    {
        return string.Format("You entered: {0}", value);
```

```
    }

    public CompositeType GetDataUsingDataContract(CompositeType composite)
    {
        if (composite == null)
        {
            throw new ArgumentNullException("composite");
        }
        if (composite.BoolValue)
        {
            composite.StringValue += "Suffix";
        }
        return composite;
    }
}
```

در WCF، پایه و اساس وب سرویس را یک interface تشکیل می‌دهد. در حقیقت اجزای وب سرویس را باید در یک interface تعریف کرد. هر یک از متدهای وب سرویس در interface تعریف شده که صفت OperationContract برای آنها در نظر گرفته شده باشد، به عنوان یکی از اعمال و متدهای قابل فراخوانی توسط استفاده کننده از وب سرویس در دسترس هستند. سپس کلاسی باید ایجاد کرد که interface ایجاد شده را پیاده سازی می‌کند. در قسمت بعد، با مفاهیم پایه‌ی Web API و برخی کاربردهای آن در محیط ASP.NET MVC آشنا می‌شوید.

نتیجه گیری

Web API، یک روش جدید و آسان برای ایجاد وب سرویس ها، بر مبنای مفاهیم آشنای ASP.NET MVC و پایه‌ی وب است. از این روش می‌توان در انواع پروژه‌های .NET استفاده کرد.

نظرات خوانندگان

نویسنده: یوسف نژاد
تاریخ: ۱۳۹۱/۰۴/۱۱ ۱۱:۵۹

این مقوله خیلی مفیده و کاربردی هست. خیلی وقت بود میخواستم در موردش بیشتر تحقیق کنم. با تشکر بابت زحماتتون و آغاز این سری مطلب جدید.

نویسنده: شهروز جعفری
تاریخ: ۱۳۹۱/۰۴/۱۱ ۱۶:۷

زمان انتشار کی هس؟

نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۴/۱۱ ۱۹:۵

بستگی به ناشر داره. اما نباید بیشتر از دو هفته طول بکشه.

نویسنده: شهروز جعفری
تاریخ: ۱۳۹۱/۰۴/۱۱ ۱۹:۸

نظر کلی من اینه که نمیشه به wcf گفت پیچیده آخه هدفش فرق میکنه.و در ضمن مگه APIحدوداً همون Rest با معماری خیلی ساده‌تر نیست؟

نویسنده: شهروز جعفری
تاریخ: ۱۳۹۱/۰۴/۱۱ ۱۹:۸

بیصبرانه منتظرش هستم

نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۴/۱۱ ۲۲:۴۷

REST بیشتر برای مواقعی هست که شما عملیات CRUD انجام میدید. در حالی که با Web API می‌تونید علاوه بر CRUD، کارهای بسیار بیشتری انجام بدید. مفاهیم و قابلیت‌های موجود در ASP.NET MVC مانند فیلترها به خوبی در Web API پشتیبانی و به راحتی قابل استفاده هستند. ضمن اینکه با Web API می‌تونید معماری REST رو با تغییر کوچکی در route پیش فرض به دست بیارید و بدین شکل، مهاجرت از REST به Web API بسیار راحت هست. در اوایل معرفی Web API، از پروتوکل OData نیز پشتیبانی اولیه میشد که متاسفانه مایکروسافت در نسخه‌ی RC این پشتیبانی رو حذف کرد. شاید در نسخه‌های بعدی این قابلیت نیز اضافه بشه که به قدرتمندتر شدن Web API کمک می‌کنه.
ضمناً، پشتیبانی مایکروسافت از WCF REST API نیز به اتمام رسیده و پیشنهاد شده که از Web API استفاده کنید.
http://aspnet.codeplex.com/wikipage?title=WCF%20REST

نویسنده: رضا.ب
تاریخ: ۱۳۹۱/۰۴/۱۲ ۳:۳

در بند سوم اشاره کردین : من معتقد هستم که تکنولوژی‌های پایه‌ی وب مانند افعال GET، POST، PUT و DELETE برای انجام اعمال مختلف توسط وب سرویس‌ها کافی هستند. ا گر ضروری نیستند بیشتر از CRUD باشند پس خاصیت ویژه‌ای که شما میگین "کارهای بسیار بیشتری" میتونه انجام بده چی هست که WCF پاسخگو نیست؟

در ضمن فکر میکنم REST فقط با منابع و ورب‌های HTTP کار داره. و برای همین سهولت و سادگیش پروتکول SOAP نسخه منسوخ

شده‌ی وب‌سرورها به‌حساب میاد. اینطور نیست؟
سوال دیگه‌ام در مورد میزان نقش Web API هست. آیا رسالت واقعی یک وب‌سرویس رو هدف گرفته؟ یعنی پیاده سازی یک Endpoint که شامل یه‌سری interface هستند که امکاناتی رو در اختیار کلاینت قرار میده؟
ممنون از توجه‌تون.

نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۴/۱۲ ۸:۳۷

دقت داشته باشید که Web API عرضه نشده تا WCF رو منسوخ کنه. برنامه هایی که صرفاً از بستر پروتوکل HTTP به عنوان یک سرویس برای رد و بدل کردن داده‌ها استفاده می‌کنند، بهتره که از این به بعد از Web API استفاده کنند. ضمن سادگی و مفاهیم آشنای ASP.NET MVC، روش یکپارچه ای برای ایجاد وب سرویس‌های HTTP نیز به وجود اومده که مشکلات استفاده از WCF رو از بین می‌بره. WCF ذاتاً برای پیغام‌های SOAP محور طراحی شده و به کار گرفتن اون برای وب سرویس‌های HTTP یا به زور خوراندن HTTP به اون بی معنیه. در WCF راه‌های مختلفی برای ایجاد وب سرویس‌های HTTP وجود داره که باعث گمراهی و سردرگمی توسعه گر میشه و حتی فریمورک‌های مختلفی مانند OpenRasta و ServiceStack نیز بدین منظور وجود دارند. بنابراین پشتیبانی WCF از HTTP به یک پروژه‌ی دیگه تحت نام ASP.NET Web API منتقل شده و WCF Web API دیگه پشتیبانی نمیشه. کمی تغییر نام و کمی جابجایی مفاهیم دراینجا صورت گرفته. WCF همچنان قدرتمنده و نباید Web API به هیچ وجه به عنوان جایگزینی برای اون تصور بشه. ایجاد بسترهایی برای ارتباطات دو طرفه یا صفی از پیغام‌ها یا سویچ بین کانال‌ها در هنگام فعال نبودن یک کانال، اینها همه از قابلیت‌هایی هست که Web API هرگز جایگزینی برای اونها نخواهد بود و مختص WCF هستند.

نویسنده: ramin
تاریخ: ۱۳۹۱/۰۴/۱۲ ۹:۲۶

سلام آقای راد
ببخشید که سوال بی ربط رو اینجا میپرسم
آیا برنامه ای برای انتشار ویرایش جدید کتاب Entity Framework دارین؟

نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۴/۱۲ ۱۰:۱۴

لطفاً سوالات اینچنینی رو از طریق ایمیل behrouz.rad[atsign]gmail بپرسید.
بله، بعد از کتاب ASP.NET MVC، کتاب Entity Framework رو آپدیت می‌کنم.

نویسنده: torisoft
تاریخ: ۱۳۹۱/۰۴/۱۶ ۲۳:۵

سلام جناب راد
از Web API تو سیلورلایت هم میشه استفاده کرد ؟
اگه استفاده میشه آیا مثبت میدونید استفاده از اونو تو سیلور ؟
با تشکر

نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۴/۱۷ ۸:۱۱

بله مشکلی نداره. پروژه‌ی Silverlight رو در یک پروژه‌ی وب Host کنید.
Silverlight هم یک نوع پروژه است، مثل Web و Desktop. اگر پروژه‌ی شما بر مبنای Silverlight هست و نیاز دارید تا امکانات اون رو به صورت سرویس ارائه بدید، می‌تونید از Web API برای عرضه‌ی این امکانات استفاده کنید.

نویسنده: حمید
تاریخ: ۱۳۹۱/۰۴/۱۷ ۱۱:۲۲

سلام. وقت بخیر.
مطالب خیلی خوب و به روزی دارین و خدا قوت..
با عرض معذرت می‌خواستم بگم من MVC4 رو نصب کردم اما بازم بعد انتخاب MVC4 از لیست Templateهای ویژوال استودیو گزینه Web API رو مشاهده نمی‌کنم.آیا افزونه یا برنامه خاصی باید نصب کنم.از قبل از زحمتتون تشکر می‌کنم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۱۷ ۱۱:۵۰

مراجعه کنید به قسمت دوم ، تصویر سوم

نویسنده: حمید
تاریخ: ۱۳۹۱/۰۴/۲۰ ۲۲:۵۷

سلام.مشکل من همینه که همین تصویر سوم رو که میگین تو این بخش من گزینه Web API رو ندارم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۲۰ ۲۳:۱۱

از طریق NuGet هم می‌تونید برای نصب آن اقدام کنید. این رو هم تست کنید:
http://nuget.org/packages/aspnetwebapi

نویسنده: Saeed M. Farid
تاریخ: ۱۳۹۱/۰۴/۲۱ ۱۱:۳

سلام و ممنون از مطلب مفید:

امکانش هست در مورد *"سویچ بین کانال‌ها در هنگام فعال نبودن یک کانال"* کمی بیشتر راهنمایی کنید یا مرجع (لینک) معرفی کنید؟ من از صحبت شما اینطور برداشت کردم که میشه در channel shape (های)ی که مثلاً برای duplex communications (یعنی IDuplexChannel) پیاده سازی کردم، اگه چنین کانالی در دسترس نبود، سوئیچ کنه روی طراحی مبتنی بر one-way messaging من؟ اصلاً چنین امکانی در سطح IChannelListener هست یا ChannelFactory ؟ کلاً اگه ممکنه یه توضیح کلی در مورد چنین امکانی که در موردش صحبت کردین بدین یا اگه جایی در موردش قبلاً بحث شده (که حتماً شده!) من رو هدایت کنید به اون، چون گلوگاه سیستم‌هام همین مورد هست.
پیشاپیش ازتون ممنونم...

نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۴/۲۱ ۱۹:۲۹

اصولاً در Web API چیزی با عنوان Channel با اون مفهوم که در WCF هست نداریم. در Web API فقط یک Transport Channel برای HTTP وجود داره، چون هدف ایجاد Web API، فقط برقراری ارتباط در سطح HTTP هست، نه مثلاً MSMQ.
Protocol Channel هم همان مفاهیمی هستند که در ASP.NET MVC وجود دارند و مثلاً قسمتی از اون، تصدیق هویت و تعیین مجوز کاربر برای دسترسی به منابع با استفاده از فیلتر Authorize هست.
لطفاً دنبال تطبیق و تناظر بین مفاهیم پیچیده‌ی WCF و یافتن معادل در Web API نباشید. Web API به وجود آمده تا ایجاد وب سرویس‌ها در بستر HTTP رو ساده کنه، همین!

نویسنده: سیروس
تاریخ: ۱۳۹۱/۱۲/۰۲ ۱۸:۳۳

سلام
یک سوال مهم داشتم، آیا استفاده از web api در Windows Form مانند WCF ممکن است، یعنی پروژه ما هم هاست و هم کلاینت رو MVC یا ASP.Net نیست، اگه میشه یه منبع معرفی کنید.

نویسنده: محسن
تاریخ: ۱۳۹۱/۱۲/۰۲ ۲۲:۳۶

اولین نتیجه جستجوی گوگل در مورد winforms web api :

Using Microsoft Web API from a Windows and WinRT Client Application

نویسنده: محمد آزاد
تاریخ: ۱۳۹۱/۱۲/۰۲ ۲۲:۳۸

تو مقدمه به این مطلب اشاره شده دوست عزیز
این سبک جدید برای ایجاد وب سرویس ها، می‌تواند در انواع پروژه‌های .NET مانند ASP.NET MVC، ASP.NET Web Forms، Windows Application و ... استفاده شود.

نویسنده: سیروس
تاریخ: ۱۳۹۱/۱۲/۰۵ ۱۰:۱۰

محسن => دوست عزیز من اون مطلب رو قبلا هم مطالعه کردم، قسمت هاست رو MVCست.اینقدر بی سواد نیستم که نتونم سرچ کنم.
آزاد => میدونم که تو Win APP قابل استفاده هست، اما می‌خوام بدونم پروژه هاست مثل WCF می‌تونه رو مستقل از Asp.Net باشه یا نه چون ظاهرا پیاده سازی WebAPI فقط روی ASP.Net امکان پذیر است.

نویسنده: محسن
تاریخ: ۱۳۹۱/۱۲/۰۵ ۱۰:۱۶

نتیجه جستجوی گوگل در مورد wep api self host :

Self-Host a Web API

نویسنده: یاسر مرادی
تاریخ: ۱۳۹۱/۱۲/۰۵ ۱۲:۳۶

من فکر کنم مطلب این دوستمون رو این جوری مطرح کنم بهتره
وقتی شما از WCF Data Services استفاده می‌کنید، WCF Data Services Client دارید، که به شما امکان نوشتن کوئری‌های Linq در سمت کلاینت، Change Tracking و Merge و ... رو می‌ده
اما من همچین آیتمی رو برای Web API پیدا نکردم، بهترین چیزی که دیدم Http Client بوده که در حد مثال زدن خوبه، ولی به درد پروژه نویسی نمی‌خوره، این که شما یک کلاینت قوی داشته باشید، خیلی مهمه، Http Client تفاوت مفهومی زیادی با $.ajax نداره
حتی در JayData هم همین طور هستش، و شما پشتیبانی خیلی بهتری از WCF Data Services می‌بینید تا از Web API، همین طور در Breeze.js
در اندروید و iOS هم شما پشتیبانی WCF Dat Services Client رو دارید، ولی Web API خیر
موفق باشید

نویسنده: محسن
تاریخ: ۱۳۹۱/۱۲/۰۵ ۱۲:۵۸

سؤال مطرح شده در مورد هاست کردن یک سرویس در برنامه ویندوزی بود که اصطلاحا Self hosting نام دارد.

Web API امکان استفاده از OData را هم دارد:

Getting started with ASP.NET Web API OData in 3 simple steps

نویسنده: یاسر مرادی
تاریخ: ۱۳۹۱/۱۲/۰۵ ۱۳:۳۵

قبول، ولی در هر حال آیا راهی جز Http Client برای دسترسی به Web API وجود دارد ؟

مثلا مبدل Linq به OData ؟
به همراه Change Tracking و ...
در ضمن موارد مهمی از OData مانند $batch در Web API پشتیبانی نشده اند، و باید برایشان Message Formatter نوشت، این نیز کار را سخت می‌کند
بر خلاف نظر دوستان به نظر من به هیچ وجه هیچ فریم ورکی راحت‌تر از WCF Data Services وجود ندارد، که جمعا با 3 خط کد راه اندازی می‌شود.

نویسنده: محسن
تاریخ: ۱۳۹۱/۱۲/۰۵ ۱۷:۲۴

NuGet مربوط به Web API OData مرتبا به روز میشه. آخرین به روز رسانی آن مربوط به 5 روز قبل بوده.

ضمن اینکه خروجی OData استاندارد است. بنابراین با کلاینت‌های موجود کار می‌کنه. فرقی نمی‌کنه تولید کننده چی هست تا زمانیکه استاندارد رعایت بشه.

نویسنده: یاسر مرادی
تاریخ: ۱۳۹۱/۱۲/۰۵ ۱۸:۵

دوست عزیز، فکر کنم سوال من خیلی واضح باشه
مسئله اول این هستش که مواردی از OData هست که در WCF Data Services وجود داره، ولی در Web API خیر، OData یک سری استاندارد هستش، بالاخره باید یک جایی پیاده سازی بشه، مثل HTML 5، که قسمت‌های مختلفش در درصدهای متفاوت در مرورگرهای متفاوت پیاده سازی شده، در این میان Chrome بهتر از IE هستش، چرا ؟ چون استانداردهای بیشتری رو پیاده سازی کرده
دوم این که آیا شما به صورت عملی از Breeze js و Jay Data و WCF Data Services Client استفاده کرده اید ؟ درسته که اینها به OData وصل می‌شوند، ولی میزان امکانات اینها برای WCF Data Services قابل قیاس با Web API نیست.
سوال اصلی من با این تفاسیر این است :
اگر قبول کنیم که راهی برای دسترسی به Web API وجود ندارد، الا استفاده از jQuery Ajax و Http Client، شما به چه صورت یک پروژه بزرگ رو با Web API می‌نویسید ؟
Change Tracking رو چه جوری پیاده سازی می‌کنید ؟
به چه صورت در کلاینت هایی مانند اندروید، و یا Win RT و ... از Linq برای دسترسی به سرویس هاتون استفاده می‌کنید ؟
اگر فرض کنیم که می‌خواهیم یک سرویس عمومی بنویسیم که همه جا به سادگی قابل استفاده باشه، آیا از Web API استفاده می‌کنید ؟
خلاصه : مزیت واقعی Web API چیست و چه زمانی پروژه ای رو با Web API شروع می‌کنید ؟
موفق و پایدار باشید

نویسنده: محسن
تاریخ: ۱۳۹۱/۱۲/۰۵ ۱۸:۳۱

«مزیت واقعی Web API چیست و چه زمانی پروژه ای رو با Web API شروع می‌کنید ؟»

WCF or ASP.NET Web APIs

به علاوه هدف اصلی Web API و یکپارچگی آن با خصوصا MVC (و بعد وب فرم‌ها) در درجه اول توسعه ActionResultهای پیش فرض MVC است (به همین جهت اول اسم آن ASP.NET است و نه مثلا اندروید):

ASP.NET Web API vs. ASP.NET MVC APIs

نویسنده: یاسر مرادی
تاریخ: ۱۳۹۱/۱۲/۰۵ ۱۹:۳

مقاله اول Web API رو با **WCF خام** مقایسه کرده، نه با WCF Data Services

مقاله دوم هم Actionهای Web API رو با MVC قیاس کرده
اگر شما یک مقاله بنویسید که مثلا Web API رو با ASP.NET Web Service قیاس بکنه، و نشون بده مزیت‌های Web API بیشتره، این می‌شه مزیت Web API بر ASP.NET Web Service، نه بر WCF Data Services
ممکنه این موارد هم مهم باشند، ولی اون چیزی که برای من سوال شده این هستش که چه زمانی در یک پروژه WCF Data Services رو می‌گذاریم کنار و از Web API استفاده می‌کنیم ؟
در واقع با توجه به امکانات واقعا زیاد WCF Data Services چرا باید اساسا از Web API استفاده بشه، اگر شما می‌فرمایید که 5 روز پیش برای Web API نسخه آمده، این عدد برای Data Services چهار روز پیش بوده
اگر بحث امکانات هست، لیست زیادی از امکانات رو من شمردم و می‌شه شمرد، از امکاناتی که تو Data Services هست، ولی تو Web API نیست.
اگر من اندروید رو مثال زدم، برای سمت کلاینت بود، شما در اندروید با چی به Web API وصل می‌شید ؟
با jQuery Ajax ؟
یا می‌خواهید به App Serverهای .NET. ای برنامه‌های دیگر، بگویید با Http Client از سرویس‌های شما استفاده کنند ؟
با سپاس

نویسنده: محسن
تاریخ: ۱۳۹۱/۱۲/۰۵ ۱۹:۱۶

هدف مایکروسافت از یکپارچه کردن WEB API با ASP.NET و خصوصا MVC ارائه یک سری *Super* ActionResult است بجای ActionResultهای معمولی MVC3. برای نمونه:

Using Kendo UI grid with Web API and OData

نویسنده: میثم99
تاریخ: ۱۳۹۳/۰۱/۲۱ ۲۳:۰

سلام
می‌خواهم بدانم برای امنیت web api در پروژه های web form چه کارهایی باید انجام دهیم بیشتر مطالب در مورد mvc هست مثلا Anti-Forgery Tokens برای mvc به راحتی می‌توان استفاده کرد ولی برای web form چکار بهتر است انجام دهیم؟
در اینجا ما مستقیما با دستورات post put و delete کار داریم که اطلاعات بانک اطلاعاتی رو تغییر می‌دهند. حالا چطور می‌توان امنیت رو کاملا تامین کرد ؟

مثلا کاربران شناسایی شده اطلاعات را وارد کنند و اینکه شخصی نتواند با یک دستور ای جکس توسط مرورگر اطلاعات اشتباه در سایت ثبت کند؟ و یا هر مشکل امنیتی دیگری که پیش بیاید؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۲۱ ۲۳:۲۱

روش‌های زیادی برای تامین امنیت در وب API و کار با «کاربران شناسایی شده » وجود دارند. لیست رسمی
از این لیست رسمی، دو مورد معروف آن در سایت جاری بررسی شده:

ASP.NET Identity

Forms authentication

مباحث پایه‌ای این‌ها مشترک است بین MVC و وب فرم‌ها و سایر فناوری‌های مشابه.

عنوان: **ASP.NET Web API - قسمت دوم**
نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۴/۱۳ ۷:۵۸
آدرس: www.dotnettips.info
برچسب‌ها: WCF, ASP.NET Web API, ASMX, Web Service

در قسمت اول به دلایل ایجاد ASP.NET Web API پرداخته شد. در این قسمت، یک مثال ساده از Web API را بررسی می‌کنیم.
تلاش‌های بسیاری توسط توسعه گران صورت پذیرفته است تا فرایند ایجاد وب سرویس WCF در بستر HTTP آسان شود. امروزه وب سرویس هایی که از قالب REST استفاده می‌کنند مطرح هستند.
ASP.NET Web API از مفاهیم موجود در ASP.NET MVC مانند Controllerها استفاده می‌کند و بر مبنای آنها ساخته شده است. بدین شکل، توسعه گر می‌تواند با دانش موجود خود به سادگی وب سرویس‌های مورد نظر را ایجاد کند. Web API، پروتوکل SOAP را به کتاب‌های تاریخی! سپرده است تا از آن به عنوان روشی برای تعامل بین سیستم‌ها یاد شود. امروزه به دلیل فراگیری پروتوکل HTTP، بیشتر محیط‌های برنامه نویسی و سیستم ها، از مبانی اولیه‌ی پروتوکل HTTP مانند اَفعال آن پشتیبانی می‌کنند.
حال قصد داریم تا وب سرویسی را که در قسمت اول با WCF ایجاد کردیم، این بار با استفاده از Web API ایجاد کنیم. به تفاوت این دو دقت کنید.

```
using System.Web.Http;

namespace MvcApplication1.Controllers
{
    public class ValuesController : ApiController
    {
        // GET api/values/5
        public string Get(int id)
        {
            return string.Format("You entered: {0}", id);
        }
    }
}
```

اولین تفاوتی که مشهود است، تعداد خطوط کمتر مورد نیاز برای ایجاد وب سرویس با استفاده از Web API است، چون نیاز به interface و کلاس پیاده ساز آن وجود ندارد. در Web API، Controllerهایی که در نقش وب سرویس هستند از کلاس ApiController ارث می‌برند. اَعمال مورد نظر در قالب متدها در Controller تعریف می‌شوند. در مثال قبل، متد Get، یکی از اَعمال است.
نحوه‌ی برگشت یک مقدار از متدها در Web API، مانند WCF است. می‌توانید خروجی متد Get را با اجرای پروژه‌ی قبل در Visual Studio و تست آن با یک مرورگر ملاحظه کنید. دقت داشته باشید که یکی از اصولی که Web API به آن معتقد است این است که وب سرویس‌ها می‌توانند ساده باشند. در Web API، تست و دیباگ وب سرویس‌ها بسیار راحت است. با مرورگر Internet Explorer به آدرس http://localhost:{port}/api/values/3 بروید. پیش از آن، برنامه‌ی Fiddler را اجرا کنید. شکل ذیل، نتیجه را نشان می‌دهد.

در اینجا نتیجه، عبارت "You entered: 3" است که به صورت یک متن ساده برگشت داده شده است.

ایجاد یک پروژه‌ی Web API

در Visual Studio، مسیر ذیل را طی کنید.

File> New> Project> Installed Templates> Visual C#> Web> ASP.NET MVC 4 Web Application

نام پروژه را HelloWebAPI بگذارید و بر روی دکمه‌ی OK کلیک کنید (شکل ذیل)

در فرمی که باز می‌شود، گزینه‌ی Web API را انتخاب و بر روی دکمه‌ی OK کلیک کنید (شکل ذیل). البته دقت داشته باشید که ما همیشه مجبور به استفاده از قالب Web API برای ایجاد پروژه‌های خود نیستیم. می‌توان در هر نوع پروژه ای از Web API استفاده کرد.

اضافه کردن مدل

مدل، شی ای است که نمایانگر داده‌ها در برنامه است. Web API می‌تواند به طور خودکار، مدل را به فرمت XML، JSON یا فرمت دلخواهی که خود می‌توانید برای آن ایجاد کنید تبدیل و سپس داده‌های تبدیل شده را در بدنه‌ی پاسخ HTTP به Client ارسال کند. تا زمانی که Client بتواند فرمت دریافتی را بخواند، می‌تواند از آن استفاده کند. بیشتر Clientها می‌توانند فرمت JSON یا XML را پردازش کنند. به علاوه، Client می‌تواند نوع فرمت درخواستی از Server را با تنظیم مقدار هدر Accept در درخواست ارسالی تعیین کند. اجازه بدهید کار خود را با ایجاد یک مدل ساده که نمایانگر یک محصول است آغاز کنیم.

بر روی پوشه‌ی Models کلیک راست کرده و از منوی Add، گزینه‌ی Class را انتخاب کنید.

نام کلاس را Product گذاشته و کدهای ذیل را در آن بنویسید.

```
namespace HelloWebAPI.Models
{
    public class Product
    {
```

```
        public int Id { get; set; }
        public string Name { get; set; }
        public string Category { get; set; }
        public decimal Price { get; set; }
    }
}
```

مدل ما، چهار Property دارد که در کدهای قبل ملاحظه می‌کنید.

اضافه کردن Controller

در پروژه ای که با استفاده از قالب پیش فرض Web API ایجاد می‌شود، دو Controller نیز به طور خودکار در پروژه‌ی Controller قرار می‌گیرند:

HomeController: یک Controller معمول ASP.NET MVC است که ارتباطی با Web API ندارد.
ValuesController: یک Controller مختص Web API است که به عنوان یک مثال در پروژه قرار داده می‌شود.

توجه: Controllerها در Web API بسیار شبیه به Controllerها در ASP.NET MVC هستند، با این تفاوت که به جای کلاس Controller، از کلاس ApiController ارث می‌برند و بزرگترین تفاوتی که در نگاه اول در متدهای این نوع کلاس‌ها به چشم می‌خورد این است که به جای برگشت Viewها، داده برگشت می‌دهند.

کلاس ValuesController را حذف و یک Controller به پروژه اضافه کنید. بدین منظور، بر روی پوشه‌ی Controllers، کلیک راست کرده و از منوی Add، گزینه‌ی Controller را انتخاب کنید.

توجه: در ASP.NET MVC 4 می‌توانید بر روی هر پوشه‌ی دلخواه در پروژه کلیک راست کرده و از منوی Add، گزینه‌ی Controller را انتخاب کنید. پیشتر فقط با کلیک راست بر روی پوشه‌ی Controller، این گزینه در دسترس بود. حال می‌توان کلاس‌های مرتبط با Controllerهای معمول را در یک پوشه و Controllerهای مربوط به قابلیت Web API را در پوشه‌ی دیگری قرار داد.

نام Controller را ProductsController بگذارید، از قسمت Template، گزینه‌ی Empty API Controller را انتخاب و بر روی دکمه‌ی OK کلیک کنید (شکل ذیل).

فایلی با نام ProductsController.cs در پوشه‌ی Controllers قرار می‌گیرد. آن را باز کنید و کدهای ذیل را در آن قرار دهید.

```
namespace HelloWebAPI.Controllers
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Net;
    using System.Net.Http;
    using System.Web.Http;
    using HelloWebAPI.Models;

    public class ProductsController : ApiController
    {

        Product[] products = new Product[]
        {
            new Product { Id = 1, Name = "Tomato Soup", Category = "Groceries", Price = 1.39M },
            new Product { Id = 2, Name = "Yo-yo", Category = "Toys", Price = 3.75M },
            new Product { Id = 3, Name = "Hammer", Category = "Hardware", Price = 16.99M }
        };

        public IEnumerable<Product> GetAllProducts()
        {
            return products;
        }

        public Product GetProductById(int id)
        {
            var product = products.FirstOrDefault((p) => p.Id == id);
            if (product == null)
            {
                var resp = new HttpResponseMessage(HttpStatusCode.NotFound);
                throw new HttpResponseException(resp);
            }
            return product;
```

```
        }

        public IEnumerable<Product> GetProductsByCategory(string category)
        {
            return products.Where(
                (p) => string.Equals(p.Category, category,
                    StringComparison.OrdinalIgnoreCase));
        }
    }
}
```

برای ساده نگهداشتن مثال، لیستی از محصولات را در یک آرایه قرار داده ایم اما واضح است که در یک پروژه‌ی واقعی، این لیست از پایگاه داده بازیابی می‌شود. در مورد کلاس‌های HttpResponseMessage و HttpResponseException بعداً توضیح می‌دهیم.
در کدهای Controller قبل، سه متد تعریف شده اند:
متد GetAllProducts که کل محصولات را در قالب نوع IEnumerable<Product> برگشت می‌دهد.
متد GetProductById که یک محصول را با استفاده از مشخصه‌ی آن (خصیصه‌ی Id) برگشت می‌دهد.
متد GetProductsByCategory که تمامی محصولات موجود در یک دسته‌ی خاص را برگشت می‌دهد.

تمام شد! حال شما یک وب سرویس با استفاده از Web API ایجاد کرده اید. هر یک از متدهای قبل در Controller، به یک آدرس به شرح ذیل تناظر دارند.

GetAllProducts به /api/products

GetProductById به /api/products/ *id*

GetProductsByCategory به /api/products/?category= *category*

در آدرس‌های قبل، id و category، مقادیری هستند که همراه با آدرس وارد می‌شوند و در پارامترهای متناظر خود در متدهای مربوطه قرار می‌گیرند. یک Client می‌تواند هر یک از متدها را با ارسال یک درخواست از نوع GET اجرا کند.

در قسمت بعد، کار خود را با تست پروژه و نحوه‌ی تعامل jQuery با آن ادامه می‌دهیم.

نظرات خوانندگان

نویسنده: mze666
تاریخ: ۱۳۹۱/۰۴/۱۳ ۸:۷

سلام آقای راد من MVC رو بلدم ولی کاربرد این WebApi , Web Service رو نمیدونم. یعنی اگر براتون ممکنه چند تا مثال واقعی از این که کجاها استفاده میشه بزنید. ممنون.

نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۴/۱۳ ۸:۱۳

وب سرویس‌ها کاربردهای متفاوتی دارند. برای ارتباط بین سیستم ها، استفاده از داده هایی که توسط یک شرکت عرضه میشه مثل اطلاعات آب و هوا یا بورس، عملیات‌های مختلفی که بر روی پایگاه داده انجام میشه، ارسال SMS، تراکنش‌های بانکی و ...

نویسنده: ایمان اسلامی
تاریخ: ۱۳۹۱/۰۴/۱۳ ۸:۱۵

ممنون از مطالب خوبتون
امیدوارم به همین شکل مطلوب ادامه داشته باشه و بهتر از اون ، به زودی شاهد چاپ کتابتون باشیم.

نویسنده: زهرا
تاریخ: ۱۳۹۱/۰۴/۱۳ ۸:۲۳

سلام آقای راد

میخواستم بپرسم که mvc 4 رو چطور به لیست پروژه هام اضافه کنم؟ و اینکه آیا این کاری که شما انجام دادید در asp.net webform هم جواب میده؟ یا اینکه باید در solution یک پروژه mvc 4 ایجاد کرد و از اون استفاده کرد؟

با تشکر

نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۴/۱۳ ۹:۲۷

سلام.
اگر نسخه‌ی آفلاین RC اون رو میخواید، از این لینک دریافت کنید.
بله، Web API در ASP.NET Web Forms هم قابل استفاده است.
در پروژه‌های Web Forms، از دیالوگ Add New Item، گزینه‌ی Web API Controller Class رو باید انتخاب کنید. route رو هم باید در متد Application_Start فایل Global.asax به صورت ذیل تعریف کنید.

```
void Application_Start(object sender, EventArgs e)
{
    RouteTable.Routes.MapHttpRoute(
        name: "DefaultApi",
        routeTemplate: "api/{controller}/{id}",
        defaults: new { id = System.Web.Http.RouteParameter.Optional }
        );
}
```

نویسنده: علی

تاریخ: ۱۳۹۱/۰۸/۲۱ ۷:۱۲

سلام آقای راد
نمی دونم چطور می‌شه از آدمایی مثل شما تشکر کرد،مطالب واقعا مفید و آموزندس
خیلی خیلی متشکرم
آقای راد یک سوال از خدمتتون داشتم،مدتیه که من و خانمم در حال ترجمه یک کتاب wcf هستیم ، این اولین کار ترجممونه ،می خواستم ازتون بپرسم که میزان محبوبیت wcf الان تو ایران چقدره ،به نظر شما آینده ای داره ؟ کلا چقدر ارزش وقت گذاشتن داره ؟

نویسنده: محمد صاحب
تاریخ: ۱۳۹۱/۰۸/۲۱ ۸:۴۶

دوست عزیز امیدوارم موفق باشید.
تا آقای راد جواب شما رو بدن این کامنت و قسمت حاشیه این پست رو ببیند بی ارتباط نیست...

نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۸/۲۱ ۱۸:۳۴

در مورد میزان محبوبیت WCF در ایران اطلاعی ندارم و مطلب خاصی هم در مورد اون در وبلاگ‌های فارسی زبان منتشر نمیشه. اما در حوزه ای که مربوط به خودم هست، حداقل قسمتی از پروژه‌های شرکت فولاد خوزستان که با تیم سازنده‌ی اونها ارتباط دارم از WCF در پروژه‌های اتوماسیون استفاده می‌کنند.
در مورد قسمت دوم سوالتون هم که دوستمون لینک‌های خوبی قرار دادند.

عنوان: **نحوه استفاده از TransactionFlow در WCF**
نویسنده: فرهاد فرهمندخواه
تاریخ: ۱۳۹۱/۰۴/۱۵ ۱۰:۴۱
آدرس: www.dotnettips.info
برچسب‌ها: WCF

شش مرحله برای ایجاد WCFTransactions در WCF مقدمه و هدف:

هدف از مطلب فوق اجرا نمودن عملیات Update ، Insert و غیرو... 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 غیرو... 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 درون 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 پروژه 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 کافیست 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 دورن دیتابیس درج نشده است.

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

نحوه تنظیم:

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

البته در شرایطی که Serviceهای 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: ASP.NET Web API - قسمت سوم
نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۴/۱۶ ۱۲:۱
آدرس: www.dotnettips.info
گروه‌ها: WCF, ASP.NET Web API, ASMX, Web Service

در قسمت اول به دلایل ایجاد Web API پرداخته شد و در قسمت دوم مثالی ساده از Web API را بررسی کردیم. در این قسمت، مثال قبل را تست کرده و نحوه‌ی تعامل jQuery با آن را بررسی می‌کنیم.

فراخوانی Web API از طریق مرورگر

با فشردن کلید F5، پروژه را اجرا کنید. شکل ذیل ظاهر می‌شود.

صفحه ای که ظاهر می‌شود، یک View است که توسط HomeController و متد Index آن برگشت داده شده است. برای فراخوانی متدهای موجود در کلاس Controller مثال قسمت قبل که مربوط به Web API است، باید به یکی از آدرس‌های اشاره شده در قسمت قبل برویم. به عنوان مثال، برای به دست آوردن لیست تمامی محصولات، به آدرس http://localhost: *xxxx* /api/products بروید. xxxx، شماره‌ی پورتی است که Web Server داخلی Visual Studio در هنگام اجرای پروژه به آن اختصاص می‌دهد. آن را نسبت به پروژه‌ی خود تغییر دهید.

نتیجه‌ی دریافتی بستگی به نوع مرورگری دارد که استفاده می‌کنید. Internet Explorer از شما در مورد باز کردن یا ذخیره‌ی

فایلی با نام products پرسش می‌کند (شکل ذیل).

محتوای فایل، بدنه‌ی پاسخ دریافتی است. اگر این فایل را باز کنید، خواهید دید که که محتوای آن، لیستی از محصولات با فرمت JSON مانند ذیل است.

```
[{"Id":1,"Name":"Tomato soup","Category":"Groceries","Price":1.39},{"Id":2,"Name":
"Yo-yo","Category":"Toys","Price":3.75},{"Id":3,"Name":"Hammer","Category":
"Hardware","Price":16.99}]
```

اما مرورگر Firefox، محصولات را در قالب XML نشان می‌دهد (شکل ذیل).

```
- <ArrayOfProduct>
  - <Product>
      <Category>Groceries</Category>
      <Id>1</Id>
      <Name>Tomato Soup</Name>
      <Price>1.39</Price>
    </Product>
  - <Product>
      <Category>Toys</Category>
      <Id>2</Id>
      <Name>Yo-yo</Name>
      <Price>3.75</Price>
    </Product>
  - <Product>
      <Category>Hardware</Category>
      <Id>3</Id>
      <Name>Hammer</Name>
      <Price>16.99</Price>
    </Product>
  </ArrayOfProduct>
```

دلیل تفاوت در نتیجه‌ی دریافتی این است که مرورگر Internet Explorer و Firefox، هر یک مقدار متفاوتی را در هدر Accept درخواست، ارسال می‌کنند. بنابراین، Web API نیز مقدار متفاوتی را در پاسخ برگشت می‌دهد.

حال به آدرس‌های ذیل بروید:

http://localhost: *xxxx* /api/products/1
http://localhost: *xxxx* /api/products?category=hardware

اولین آدرس، باید محصولی با مشخصه‌ی 1 را برگشت دهد و دومین آدرس، لیستی از تمامی محصولاتی که در دسته‌ی hardware قرار دارند را برگشت می‌دهد (در مثال ما فقط یک آیتم این شرط را دارد).

نکته: در صورتی که در هنگام فراخوانی هر یک از متدهای Web API با خطای ذیل مواجه شدید، دستور [("AcceptVerbs("GET","POST") را به ابتدای متدها اضافه کنید.

```
The requested resource does not support http method 'GET'
```

فراخوانی Web API با استفاده از کتابخانه‌ی jQuery

در قسمت قبل، متدهای Web API را مستقیماً از طریق وارد کردن آدرس آنها در نوار آدرس مرورگر فراخوانی کردیم. اما در اکثر اوقات، این متدها با روش‌های برنامه نویسی توسط یک Client فراخوانی می‌شوند. اجازه بدهید Clientیی ایجاد کنیم که با استفاده از jQuery، متدهای ما را فراخوانی می‌کند.
در Solution Explorer، از پوشه‌ی Views و سپس Home، فایل Index.cshtml را باز کنید.

تمامی محتویات این View را حذف و کدهای ذیل را در آن قرار دهید.

```
<!DOCTYPE html>
<html>
<head>
    <title>ASP.NET Web API</title>
    <script src="../../Scripts/jquery-1.7.2.min.js"
        type="text/javascript"></script>
</head>
<body>
    <div>
        <h1>All Products</h1>
        <ul id='products' />
    </div>
    <div>
        <label for="prodId">ID:</label>
        <input type="text" id="prodId" size="5"/>
        <input type="button" value="Search" onclick="find();" />
        <p id="product" />
    </div>
</body>
</html>
```

بازیابی لیستی از محصولات

برای بازیابی لیستی از محصولات، فقط کافی است تا یک درخواست از نوع GET به آدرس "/api/products" بفرستید. این کار با jQuery به صورت ذیل انجام می‌شود.

```
<script type="text/javascript">
    $(document).ready(function () {
        // Send an AJAX request
        $.getJSON("api/products/",
        function (data) {
            // On success, 'data' contains a list of products.
            $.each(data, function (key, val) {

                // Format the text to display.
                var str = val.Name + ': $' + val.Price;

                // Add a list item for the product.
                $('<li/>', { html: str })
                .appendTo($('#products'));
            });
        });
    });
</script>
```

متد getJSON، یک درخواست AJAX از نوع GET را ارسال می‌کند و پاسخ دریافتی آن نیز با فرمت JSON خواهد بود. دومین پارامتر متد getJSON، یک callback است که پس از دریافت موفقیت آمیز پاسخ اجرا می‌شود.

بازیابی یک محصول با استفاده از مشخصه‌ی آن

برای بازیابی یک محصول با استفاده از مشخصه‌ی آن، یک درخواست از نوع GET به آدرس "/ api/products/ *id*" ارسال کنید. id، مشخصه‌ی محصول است. کد ذیل را در ادامه‌ی کد قبل و پیش از تگ <script/> قرار دهید.

```
function find() {
    var id = $('#prodId').val();
    $.getJSON("api/products/" + id,
        function (data) {
            var str = data.Name + ': $' + data.Price;
            $('#product').html(str);
        })
    .fail(
        function (jqXHR, textStatus, err) {
            $('#product').html('Error: ' + err);
        });
}
```

باز هم از متد getJSON استفاده کردیم، اما این بار مقدار id برای آدرس از یک Text Box خوانده و آدرس ایجاد می‌شود. پاسخ دریافتی، یک محصول در قالب JSON است.

اجرای پروژه

پروژه را با فشردن کلید F5 اجرا کنید. پس از نمایش فرم، تمامی محصولات بر روی صفحه نمایش داده می‌شوند. عدد 1 را وارد و بر روی دکمه‌ی Search کلیک کنید، محصولی که مشخصه‌ی آن 1 است نمایش داده می‌شود (شکل ذیل).

اگر مشخصه ای را وارد کنید که وجود ندارد، خطای 404 با مضمون "Error: Not Found" بر روی صفحه نمایش داده می‌شود و در صورتی که به جای عدد، عبارتی غیر عددی وارد کنید، خطای 400 با مضمون: "Error: Bad Request" نمایش داده می‌شود. **در Web API، تمامی پاسخ‌ها باید در قالب کدهای وضعیت HTTP باشند** (شکل ذیل). این یکی از اصول اساسی کار با وب سرویس‌ها است. وفادار ماندن به مفاهیم پایه‌ی وب، دید بهتری در مورد اتفاقاتی که می‌افتد به شما می‌دهد.

در قسمت بعد با مفهوم مسیریابی در ASP.NET Web API آشنا می‌شوید.

نظرات خوانندگان

نویسنده: Nima
تاریخ: ۱۳۹۱/۰۴/۱۶ ۲۰:۶

سلام آقای راد

ممنون از مطلب مفیدتون..سوالی ار حضورتون داشتم ما در وب سرویسهای asmx میتونستیم از sessionها استفاده کنیم تا مثلا اگر میخواستیم از طریق jquery بخواهیم اون وب سرویس رو صدا کنیم این کار فقط برای کاربرانی که در سیستم وارد شده اند امکان پذیر باشد .از لحاظ ملاحظات امنیتی و استفاده از session آیا در قسمتهای بعدی بحث میکنید؟

با تشکر

نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۴/۱۷ ۷:۴۶

در Web API در حالت پیش فرض نمی‌تونید از Session استفاده کنید. اصولاً REST اصطلاحاً Stateless هست، اما اگر اصرار به استفاده از Session دارید، باید یک Route Handler سفارشی ایجاد و اینترفیس IRequiresSessionState رو پیاده سازی کنید. سپس پیاده سازی جدید رو به عنوان Route Handler برای route مختص Web API تعریف کنید.
در مورد تصدیق هویت، معمولاً به این شکل عمل میشه که یک فیلتر Authorize سفارشی ایجاد و نام کاربری و کلمه‌ی عبور از طریق یک Header سفارشی به Server ارسال میشه. Web API به خوبی با مفهوم فیلترها در ASP.NET MVC هماهنگ هست. سعی می‌کنم در مطلب جدایی به این موارد بپردازم.

نویسنده: Nima
تاریخ: ۱۳۹۱/۰۴/۱۷ ۹:۵۷

با تشکر از شما آقای راد اگر این زحمت رو بکشین ممنون میشم .دونستن مسائل امنیتی باعث استفاده بهتر از مواردی که شما فرمودین میشه. موفق باشید

نویسنده: مهران کلانتری
تاریخ: ۱۳۹۱/۰۴/۱۷ ۱۸:۵۴

سلام آقای راد خیلی خوب و سلیس توضیح میدید

در رابطه با مسائل امنیتی در این روش خیلی خوب می‌شه اگر توضیحی ارائه بدید.

متشکرم

نویسنده: شهروز جعفری
تاریخ: ۱۳۹۱/۰۴/۱۸ ۲۱:۲۵

در Rest قابلیتی بنام Syndication Feed Fromatter وجود دارد در Web API چطور؟

نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۴/۱۹ ۱۰:۳۲

در Web API هم این قابلیت وجود داره.

عنوان: ASP.NET Web API - قسمت چهارم
نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۴/۱۹ ۱۱:۱۵
آدرس: www.dotnettips.info
گروه‌ها: WCF, ASP.NET Web API, ASMX, Web Service

آشنایی با مفهوم مسیریابی در Web API

در این قسمت با نحوه‌ی تناظر آدرس‌ها توسط Web API به متدهای موجود در Controller آشنا می‌شوید.
در هر درخواستی که ارسال می‌شود، Web API، انتخاب Controller مناسب را با رجوع به جدولی با نام جدول مسیرها انجام می‌دهد. زمانی که یک پروژه‌ی جدید با استفاده از ASP.NET MVC 4 ایجاد می‌کنید، یک route پیش فرض به صورت ذیل در متد RegisterRoutes قرار می‌گیرد.

```
routes.MapHttpRoute(
    name: "DefaultApi",
    routeTemplate: "api/{controller}/{id}",
    defaults: new { id = RouteParameter.Optional }
);
```

عبارت api، ثابت است و قسمت‌های {controller} و {id} توسط آدرس مقداردهی می‌شوند. زمانی که آدرسی با این الگو تطبیق داشته باشد، کارهای ذیل انجام می‌گیرد:
{controller} به نام Controller تناظر پیدا می‌کند.
نوع درخواست ارسالی (GET، POST، PUT، DELETE) به نام متد تناظر پیدا می‌کند.
اگر قسمت {id} در آدرس وجود داشته باشد، به پارامتر id متد انتخاب شده پاس داده می‌شود.
اگر آدرس دارای Query String باشد، به پارامترهای همنام خود در متد، تناظر پیدا می‌کنند.

در ذیل، مثال هایی را از چند آدرس درخواستی و نتیجه‌ی حاصل از فراخوانی آنها مشاهده می‌کنید.
آدرس /api/products با نوع درخواست GET به متد GetAllProducts()
آدرس /api/products/1 با نوع درخواست GET به متد GetProductById(1)
آدرس /api/products?category=hardware با نوع درخواست GET به متد GetProductByCategory("hardware")

در آدرس اول، عبارت "products" به ProductsController تطبیق پیدا می‌کند. درخواست نیز از نوع GET است، بنابراین Web API به دنبال متدی در Controller می‌گردد که نام آن با عبارت GET "آغاز" شده باشد. همچنین، آدرس شامل قسمت {id} نیز نیست. بنابراین، Web API متدی را انتخاب می‌کند که پارامتر ورودی ندارد. متد GetAllProducts در ProductsController، تمامی این شروط را دارد، پس انتخاب می‌شود.
در دومین آدرس، همان حالت قبل وجود دارد، با این تفاوت که در آدرس درخواستی، قسمت {id} وجود دارد. از آنجا که نوع قسمت {id} در متد GetProductById، int تعریف شده است، باید یک عدد صحیح بعد از آدرس /api/products وجود داشته باشد تا متد GetProductById فراخوانی شود. این عدد به طور خودکار به نوع int تبدیل شده و در پارامتر اول متد GetProductById قرار می‌گیرد. در ذیل، برخی آدرس‌ها را ملاحظه می‌کنید که معتبر نیستند و باعث بروز خطا می‌شوند.

آدرس /api/products با نوع درخواست POST، باعث خطای 405Method Not Allowed می‌شود.
آدرس /api/users با نوع درخواست GET، باعث خطای 404Not Found می‌شود.
آدرس /api/products/abc با نوع درخواست GET، باعث خطای 400Bad Request می‌شود.

در آدرس اول، Client یک درخواست از نوع POST ارسال کرده است. Web API به دنبال متدی می‌گردد که نام آن با عبارت Post آغاز می‌شود. اما متدی با این شرط در ProductsController وجود ندارد. بنابراین، پاسخی که دریافت می‌شود، عبارت "405 Method Not Allowed" است. درخواست برای آدرس /api/users/ نیز معتبر نیست، چون Controllerی با نام UsersController وجود ندارد. و سومین آدرس نیز بدین دلیل نامعتبر است که قسمت abc نمی‌تواند به یک عدد صحیح تبدیل شود.

مشاهده‌ی درخواست ارسالی و پاسخ دریافتی

زمانی که با یک وب سرویس کار می‌کنید، مشاهده‌ی محتویات درخواست ارسالی و پاسخ دریافتی می‌تواند کاربرد زیادی در درک نحوه‌ی تعامل بین Client و وب سرویس و کشف خطاهای احتمالی داشته باشد. در Firefox با استفاده از افزونه‌ی Firebug و در Internet Explorer 9 به بالا با ابزار Developer Tools آن می‌توان درخواست‌ها و پاسخ‌ها را مشاهده کرد. در Internet Explorer، کلید F12 را برای اجرای Developer Tools فشار دهید. از قسمت Network بر روی دکمه‌ی Start Capturing کلیک کنید. حال کلید F5 را برای بارگذاری مجدد صفحه فشار دهید. Internet Explorer، درخواست و پاسخ رد و بدل شده بین مرورگر و Web Server را مانیتور کرده و گزارشی را نشان می‌دهد (شکل ذیل).

از ستون URL، آدرس /api/products/ را انتخاب و بر روی دکمه‌ی Go to detailed view کلیک کنید. در قسمتی که باز می‌شود، گزینه هایی برای مشاهده‌ی هدرهای درخواست، پاسخ و همچنین بدنه‌ی هر یک وجود دارد. به عنوان مثال، اگر قسمت Request headers را انتخاب کنید، خواهید دید که Internet Explorer از طریق هدر Accept، تقاضای پاسخ در قالب JSON را کرده است (شکل ذیل).

اگر قسمت Response body را انتخاب کنید، پاسخ دریافت شده در قالب JSON را خواهید دید.

در قسمت بعد، با مدیریت کدهای وضعیت HTTP برای اعمال چهارگانه‌ی CRUD آشنا می‌شوید.

نظرات خوانندگان

نویسنده: نیما
تاریخ: ۱۳۹۱/۰۴/۱۹ ۱۲:۱۲

سلام آقای راد

با تشکر از زحمتی که میکشید. فرمودید که :

"بنابراین web api به دنبال متدی در controller می‌گردد که نام آن با عبارت get "آغاز" شده باشد. "

آیا این کار باعث عدم دقت و ایجاد خطاهای ناخواسته نمیشه؟ این فقط متدی با get شروع بشه شاید برای من که خیلی کم mvc کار کردم یکم مشکل دار به نظر برسه.اگر ما دو متد داشته باشیم که در ابتدای آنها get باشد آیا برنامه خطا میگیرد؟ ممنون میشم یکم در این باره توضیح بدین

نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۴/۱۹ ۱۲:۲۶

شما محدود به رفتار پیش فرض Web API نیستید. می‌تونید route رو تغییر بدید و نام Action رو هم در اون ذکر کنید.

```
routes.MapHttpRoute(
    name: "DefaultApi",
    routeTemplate: "api/{controller}/{action}/{id}",
    defaults: new { id = RouteParameter.Optional }
);
```

و در ProductsController داشته باشید:

```
[HttpGet]
public string Details(int id)
{
   // do something
}
```

حال درخواستی برای /api/products/details/1 باعث اجرای متد Details میشه.
یا حتی می‌تونید route رو تغییر ندید و فقط از [HttpGet] و [HttpPost] و امثال اونها برای تعیین فعل استفاده کنید. به عنوان مثال، اگر route پیش فرض رو تغییر ندید و متد Details رو به شکل قبل داشته باشید، آدرسی مانند /api/products/1 با نوع GET باعث میشه تا متد Details اجرا بشه.

نویسنده: نیما
تاریخ: ۱۳۹۱/۰۴/۱۹ ۱۲:۲۷

بسیار ممنونم خیلی مفید بود

نویسنده: آریا
تاریخ: ۱۳۹۱/۰۴/۱۹ ۱۹:۱

خیلی عالی بود. متشکر

نویسنده: رضا.ب
تاریخ: ۱۳۹۱/۰۴/۲۱ ۱۰:۳۰

یه سوال که ربط چندانی به این پست نداره؛
asp.net wep api رو میتونیم یه لایه abstraction حساب کنیم که در اون منطق سیستم(BL) وجود داره و بنابراین از آن در سطح انتزاعی بالاتری در سیستم یا سیستم‌های مشابه استفاده میشن؟ (تاکید سوالم آنجاست که میزان عملکرد موثر asp.net web api تا کجاست؟)
ممنون.

نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۴/۲۱ ۱۹:۳۸

شما در سوالتون می‌تونید عبارت "ASP.NET Web API" رو با "Web Service تحت HTTP" جایگزین کنید. در Web Service هم منطق سیستم وجود داره، مثلاً محاسبه‌ی نرخ تورم در یک بازه‌ی زمانی با توجه به 30 قلم کالای اساسی. عملکرد Web API، همان عملکردی است که از یک Web Service تحت HTTP مانند ASMX انتظار دارید.

نویسنده: حمزه ء
تاریخ: ۱۳۹۳/۰۲/۲۰ ۱۲:۲۶

در قسمت سوم آموزش این مثال رو داشتیم :

```
$.getJSON("api/products/"+id,
                function (data) {
                    var str = data.Name + '    ' + data.Price;
                    $('#products').empty();
                    $('#products').html(str);
                }
                );
```

خب تا اینجا api/products/id اجرا میشه .
فرض کنید چند جستجو داریم و نیاز داریم برای هر کدوم اکشن متناظر با اون اجرا بشه برای مثال:
api/products/id
api/products/details/id
حالا چطور میتونم برای دو دکمه تعیین کنم ، با زدن هر کدوم چه تابعی اجرا بشه ؟

بهتر بگم چطور details رو برای یک دکمه به آدرس اضافه کنم ؟

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۰۲/۲۰ ۱۳:۲

از متد click استفاده کنید. داخل callback آن درخواست Ajax ایی را ارسال کنید به سرور.

نویسنده: حمزه ء
تاریخ: ۱۳۹۳/۰۲/۲۰ ۱۸:۲

ممنونم .
1-کدها رو رویداد کلیک نوشتم و اجرا شد . ولی توی آدرس بار مرورگر هیچ تغییری بوجود نیومد ؟ چطور میتونم زمانی که یک متد رو از web api فراخوانی کردم ، همزمان آدرس بار مرورگر هم تغییر کنه ؟
2- برای اینکه فقط یوزرهای سایت و آنلاین شده یا role های خاص بتونن از اون متد استفاده کنن ، attribute رو بالای اون اضافه کردم ، آیا درسته ؟

```
[Authorize(Roles="Admin")]
        //[Authorize(Users="")]
```

```
        public Product GetProductById(int Id)
        {
            var product = Products.FirstOrDefault(p => p.Id == Id);
            if(product==null)
            {
                throw new HttpResponseException(HttpStatusCode.NotFound);
            }
            return product;
        }
```

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۰۲/۲۰ ۱۸:۵۸

این ASP.NET MVC نیست. ASP.NET Web API است. می‌تونی دستی آدرس خاصی رو در مرورگر وارد کنی و نهایتا مثلا خروجی JSON یا XML بگیری (شاید بهتر باشه یکبار اینکار رو انجام بدی تا حس بهتری نسبت به این فناوری پیدا کنی که کارش چی هست. خروجی‌اش چی هست). در کل هدفش این نیست که خروجی HTML به شما بده. هدفش تامین داده برای کلاینت‌ها هست. سمت کلاینت رو آزاد هستی هر طور که دوست داشتی کار کنی. مثلا یک صفحه‌ی HTML درست کنی و اطلاعات Web API رو بگیری و نمایش بدی.

عنوان: ASP.NET Web API - قسمت پنجم
نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۴/۲۴ ۱۳:۱۸
آدرس: www.dotnettips.info
گروه‌ها: WCF, ASP.NET Web API, ASMX, Web Service

مدیریت کدهای وضعیت در Web API

تمامی پاسخ‌های دریافتی از Web API توسط Client، باید در قالب کدهای وضعیت HTTP باشند. دو کلاس جدید با نام‌های HttpResponseMessage و HttpResponseException همراه با ASP.NET MVC 4 معرفی شده اند که ارسال کدهای وضعیت پردازش درخواست به Client را آسان می‌سازند. به عنوان مثال، ارسال وضعیت برای چهار عمل اصلی بازیابی، ایجاد، آپدیت و حذف رکورد را بررسی می‌کنیم.

بازیابی رکورد

بر اساس مستندات پروتوکل HTTP، در صورتی که منبع درخواستی Client پیدا نشد، باید کد وضعیت 404 برگشت داده شود. این حالت را در متد ذیل پیاده سازی کرده ایم.

```
public Product GetProduct(int id)
{
    Product item = repository.Get(id);
    if (item == null)
    {
        throw new HttpResponseException(new HttpResponseMessage(HttpStatusCode.NotFound));
    }
    return item;
}
```

در صورتی که رکوردی با مشخصه‌ی درخواستی پیدا نشد، با استفاده از کلاس HttpResponseException، خطایی به Client ارسال خواهد شد. پارامتر سازنده‌ی این کلاس، شی ای از نوع کلاس HttpResponseMessage است. سازنده‌ی کلاس HttpResponseMessage، مقداری از یک enum با نام HttpStatusCode را می‌پذیرد. مقدار NotFound، نشان از خطای 404 است و زمانی به کار می‌رود که منبع درخواستی وجود نداشته باشد. اگر محصول درخواست شده یافت شد، در قالب JSON برگشت داده می‌شود. در شکل ذیل، پاسخ دریافتی در زمان درخواست محصولی که وجود ندارد را ملاحظه می‌کنید.

ایجاد رکورد

برای ایجاد رکورد، Client درخواستی از نوع POST را همراه با داده‌های رکورد در بدنه‌ی درخواست به Server ارسال می‌کند. در ذیل، پیاده سازی ساده ای از این حالت را مشاهده می‌کنید.

```
public Product PostProduct(Product item)
{
    item = repository.Add(item);
    return item;
}
```

این پیاده سازی کار می‌کند اما کمبودهایی دارد:

کد وضعیت پردازش درخواست : به طور پیش فرض، Web API، کد 200 را در پاسخ ارسال می‌کند، اما بر اساس مستندات پروتوکل HTTP، زمانی که یک درخواست از نوع POST منجر به تولید منبعی می-شود، Server باید کد وضعیت 201 را به Client برگشت بدهد.

آدرس منبع جدید ایجاد شده : بر اساس مستندات پروتوکل HTTP، زمانی که منبعی بر روی Server ایجاد می‌شود، باید آدرس منبع جدید ایجاد شده از طریق هدر Location به Client ارسال شود.

با توجه به این توضیحات، متد قبل به صورت ذیل در خواهد آمد.

```
public HttpResponseMessage PostProduct(Product item)
{
    item = repository.Add(item);
    var response = Request.CreateResponse(HttpStatusCode.Created, item);

    string uri = Url.Link("DefaultApi", new { id = item.Id });
    response.Headers.Location = new Uri(uri);
    return response;
}
```

همان طور که ملاحظه می‌کنید، خروجی متد از نوع کلاس HttpResponseMessage است، چون با استفاده از این نوع می‌توانیم جزئیات مورد نیاز را در مورد نتیجه‌ی پردازش درخواست به مرورگر ارسال کنیم. همچنین، داده‌های رکورد جدید نیز در بدنه‌ی پاسخ، با یک فرمت مناسب مانند XML یا JSON برگشت داده می‌شوند. با استفاده از متد CreateResponse کلاس Request و پاس دادن کد وضعیت و شی ای که قصد داریم به Client ارسال شود به این متد، شی ای از نوع کلاس HttpResponseMessage ایجاد می‌کنیم. آدرس منبع جدید نیز با استفاده از response.Headers.Location مشخص شده است. نمونه ای از پاسخ دریافت شده در سمت Client به صورت ذیل است.

آپدیت رکورد

آپدیت با استفاده از درخواست‌های از نوع PUT انجام می‌گیرد. یک مثال ساده در این مورد.

```
public void PutProduct(int id, Product product)
{
    product.Id = id;
    if (!repository.Update(product))
    {
        throw new HttpResponseException(new HttpResponseMessage(HttpStatusCode.NotFound));
    }
}
```

نام متد با عبارت Put آغاز شده است. بنابراین توسط Web API برای پردازش درخواست‌های از نوع PUT در نظر گرفته می‌شود. متد قبل، دو پارامتر ورودی دارد. id برای مشخصه‌ی محصول، و محصول آپدیت شده که در پارامتر دوم قرار می‌گیرد. مقدار پارامتر id از آدرس دریافت می‌شود و مقدار پارامتر product از بدنه‌ی درخواست. به طور پیش فرض، Web API، مقدار داده هایی با نوع ساده مانند int، string و bool را از طریق route، و مقدار نوع‌های پیچیده‌تر مانند داده‌های یک کلاس را از بدنه‌ی درخواست می‌خواند.

حذف یک رکورد

حذف یک رکورد، با استفاده از درخواست‌های از نوع DELETE انجام می‌گیرد. یک مثال ساده در این مورد.

```
public HttpResponseMessage DeleteProduct(int id)
{
    repository.Remove(id);
    return new HttpResponseMessage(HttpStatusCode.NoContent);
}
```

بر اساس مستندات پروتکل HTTP، اگر منبعی که Client قصد حذف آن را دارد از پیش حذف شده است، نباید خطایی به وی گزارش شود. معمولاً در متدهایی که وظیفه‌ی حذف منبع را بر عهده دارند، کد 204 مبنی بر پردازش کامل درخواست و پاسخ خالی برگشت داده می‌شود. این کد با استفاد از مقدار NoContent برای HttpStatusCode مشخص می‌شود.

فراخوانی متدها و مدیریت کدهای وضعیت HTTP در سمت Client

حال ببینیم چگونه می‌توان از متدهای قبل در سمت Client استفاده و خطاهای احتمالی آنها را مدیریت کرد.
بهتر است مثال را برای حالتی که در آن رکوردی آپدیت می‌شود بررسی کنیم. کدهای مورد نیاز برای فراخوانی متد PutProduct در سمت Client به صورت ذیل است.

```
var id = $("#myTextBox").val();

$.ajax({
  url: "/api/Test/" + id,
   type: 'PUT',
   data: { Id: "1", Name: "Tomato Soup", Category: "Groceries", Price: "1.39M" },
   cache: false,
   statusCode: {
       200: function (data) {
           alert("آپدیت انجام شد");
        },
       404:
            function () {
                alert("خطا در آپدیت");
            }
      }
});
```

از متدهای get، getJson یا post در jQuery نمی‌توان برای عمل آپدیت استفاده نمود، چون Web API انتظار دارد تا نام فعل درخواستی، PUT باشد. اما با استفاده از متد ajax و ذکر نام فعل در پارامتر type آن می‌توان نوع درخواست را PUT تعریف کرد.
خط 5 بدین منظور است. از طریق خصیصه‌ی statusCode نیز می‌توان کدهای وضعیت مختلف HTTP را بررسی کرد. دو کد 200 و 404 که به ترتیب نشان از موفقیت و عدم موفقیت در آپدیت رکورد هستند تعریف شده و پیغام مناسب به کاربر نمایش داده می‌شود.
در حالتی که آپدیت با موفقیت همراه باشد، بدنه‌ی پاسخ به شکل ذیل است.

و در صورتی که خطایی رخ دهد، بدنه‌ی پاسخ دریافتی به صورت ذیل خواهد بود.

Console HTML CSS Script DOM Net Cookies
Clear Persist All HTML CSS JS XHR Images Flash Media
PUT 2
404 Not Found
localhost:2239
0
127.0.0.1:2239
Headers Put XML
Parameters
application/x-www-form-urlencoded
Category Groceries
Id 1
Name Tomato Soup
Price 1.39M
Source
Id=1&Name=Tomato+Soup&Category=Groceries&Price=1.39M

نظرات خوانندگان

نویسنده: princedotnet
تاریخ: ۱۳۹۱/۰۴/۳۱ ۰:۳۰

سلام جناب راد
2 تا سوال داشتم :
1.چطور می‌تونم اطلاعات گرفته شده از WebAPI رو توسط JSON.NET در یک پروژه سیلورلایت Deserialize کنم؟
2.چطور مدل هایی که در اون از روابط many to many - many to one یا... در Entity استفاده شده رو از یک WebAPI بگیرم؟
ممنون

نویسنده: آزاده
تاریخ: ۱۳۹۲/۰۶/۱۹ ۱۱:۴۶

سلام، با تشکر؛ من در صورتی که بخواهم کاری کنم که کاربر فقط از توی فرم و از طریق jqueyهای نوشته شده بتونه به اطلاعات دسترسی داشته باشد، یعنی در صورتی که از آدرس بار بروزر استفاده کرد، خروجی رو نگیرد چیکار باید بکنم؟

ممنون

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۶/۱۹ ۱۲:۸

از محدودیت POST استفاده کنید بجای GET.

نویسنده: آزاده
تاریخ: ۱۳۹۲/۰۶/۱۹ ۱۳:۱۲

سلام . ممنون از راهنماییتون.
یعنی همون متدی که دارم رو فقط به نوع Post تغییر بدم کافیه. و از اون به بعد از آدرس بار نمی‌شه بهش دسترسی داشت.
احتیاجی به تنظیمات خاصی نداره دیگه؟

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۶/۱۹ ۱۳:۳۴

کار معمولی با یک آدرس در مرورگر یعنی حالت Get. میشه این رو تغییر داد به Post که با بازکردن ساده آدرس در مرورگر کار نکنه.

عنوان: **مدیریت Instance در WCF**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۲/۲۷ ۱۷:۴۰
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: ۱۳۹۲/۰۲/۳۱ ۸:۴۲

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

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۲/۰۲/۳۱ ۹:۱۱

می تونید از کتاب 348 صفحه ای Multi-tier Services Development with LINQ to Entities WCF 4.0 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 مناسبیه. البته MSDN رو هم فراموش نکنید.

نویسنده: احسان شیروان
تاریخ: ۱۳۹۳/۰۶/۱۹ ۱۰:۵۵

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.Reqquired)]
```

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 ازدست میده البته static نیست و به دلیل ساختار اون نمیتونم استاتیکش کنم
ممنون میشم راهنمایی نمایید

عنوان: **WCF Method Overloading**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۲/۳۰ ۲۲:۴۰
آدرس: www.dotnettips.info
گروه‌ها: WCF, method overloading

تشریح مسئله : چگونه متدهای سرویس WCF را Overload کنیم.

نکته : آشنایی با مفاهیم اولیه WFC 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 SumByIntNumbers و SumByFloatNumbers خواهیم داشت. البته اگر از روش Self Hosted استفاده کنیم دقیقا 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.

عنوان: **مدیریت تغییرات در سیستم های مبتنی بر WCF**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۳/۰۲ ۱۵:۲۰
آدرس: www.dotnettips.info
برچسب‌ها: WCF, versioning

تشریح مسئله : در صورتی که بعد از انتشار برنامه؛ در نسخه بعدی مدل سمت سرور تغییر کرده باشد و امکان بروز رسانی مدل‌های 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 می‌نویسیم.

```
[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 می‌کند.

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 DataContractهای سمت سرور و کلاینت اختلاف وجود داشته باشد این موارد نادیده گرفته می‌شوند.
یعنی همیشه مقدار خاصیت Author برابر null خواهد بود.
نکته : برای Value Typeها مقادیر پیش فرض و برای Reference Typeها 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 می‌شود. به صورت زیر:

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

اما این همیشه راه حل مناسبی نیست.
روش دیگر این است که Desrialize 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 می‌کند.

```
[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 می‌شود.مثل تصویر زیر:

روش بعدی استفاده از اینترفیس IExtensibleDataObject است. بعد از اینکه کلاس Book این اینترفیس را پیاده سازی کرد مشکل Versioning Round Trip حل می‌شود. به این صورت که سرویس یا کلاینتی که نسخه قدیمی را می‌شناسد اگر نسخه جدید را دریافت کند خصوصیاتی را که نمی‌شناسد مثل Author در خاصیت ExtensionData ذخیره می‌شود و هنگامی که کلاس Book برای سرویس یا کلاینتی که نسخه جدید را می‌شناسد DataContractSerializer اطلاعات مورد نظر را از خصوصیت ExtensionData بیرون می‌کشد و کلاس Book جدید را باز سازی می‌کند. بررسی کلاس ExtensionData توسط خود DataContractSreializer انجام می‌شود 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;
Consc
```

همان طور که می‌بینید این نسخه از کلاینت هیچ گونه اطلاعی از وجود یک خاصیت به نام Author ندارد ولی از طریق ExtensionData متوجه می‌شود 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 می‌توانیم 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 می‌کند.

منابع :

http://msdn.microsoft.com/en-us/library/system.runtime.serialization.iextensibledataobject.aspx

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

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

و...

عنوان: آشنایی با KnownTypeAttribute در WCF
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۳/۱۱ ۰:۴۰
آدرس: www.dotnettips.info
برچسب‌ها: WCF, Serialization, knownType

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 DataContact 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

عنوان: **Data Contracts and Circular References**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۳/۲۰ ۸:۲۰
آدرس: www.dotnettips.info
برچسب‌ها: WCF, Serialization, Circular Reference, DataContract

تشریح مسئله : در DataContractSerializer قابلیتی به عنوان سریالایز کردن objectها به صورت درختی وجود داردکه اصطلاحا 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 کافیست 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>
```

کاملا 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 سریالاز 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.

عنوان: **استفاده از Lambda Expression در پروژه های مبتنی بر WCF**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۳/۳۰ ۱۹:۱۰
آدرس: www.dotnettips.info
برچسب‌ها: WCF, Serialization, Expression, Delegate

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

امروزه استفاده از WCF در پروژه‌های SOA بسیار فراگیر شده است. کمتر کسی است که در مورد قدرت تکنولوژی WCF نشنیده باشد یا از این تکنولوژی در پروژه‌های خود استفاده نکرده باشد. WCF مدل برنامه نویسی یکپارچه مایکروسافت برای ساخت نرم‌افزارهای سرویس گرا است و برای توسعه دهندگان امکانی را فراهم می‌کند که راهکارهایی امن، و مبتنی بر تراکنش را تولید نمایند که قابلیت استفاده در بین پلتفرم‌های مختلف را دارند. قبل از WCF توسعه دهندگان پروژه‌های نرم افزاری برای تولید پروژه‌های توزیع شده باید شرایط موجود برای تولید و توسعه را در نظر می‌گرفتند. برای مثال اگر استفاده کننده از سرویس در داخل سازمان و بر پایه دات نت تهیه شده بود از .net remoting استفاده می‌کردند و اگر استفاده کننده سرویس از خارج سازمان یا مثلا بر پایه تکنولوژی J2EE بود از Web Serviceها استفاده می‌شد. 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 serviceها نیست.

WCF با تمام قدرت و امکاناتی که داراست دارای نقاط ضعفی هم می‌باشد که البته این معایب (یا محدودیت) بیشتر جهت سازگار سازی سرویس‌های نوشته شده با سیستم‌ها و پروتکل‌های مختلف است.
برای انتقال داده‌ها از طریق WCF بین سیستم‌های مختلف باید داده‌های مورد نظر حتما سریالایز شوند که مثال هایی از این دست رو در همین سایت می‌تونید مطالعه کنید:
(^) و (^) و (^)

با توجه به این که داده‌ها سریالایز می‌شوند، در نتیجه امکان انقال 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 نمی‌تونید 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 خاصیت‌های 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 می‌زنم؟

سرویسی داریم برای اطلاعات کتاب ها. قصد داریم متدی بنوسیم 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 می‌کنیم.

*ابتدا یک Blank Solution ایجاد کنید.

*یک ClassLibrary به نام Model ایجاد کنید و کلاسی به نام Book در آن بسازید .(همانطور که میبینید 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 assemblyهای مورد نظر را انتخاب کنید.(در این پروژه باید Model و System.Xml.Linq را انتخاب کنید)

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؟

روش‌های زیادی برای بر طرف کردن این محدودیت وجود دارد. اما در این پست روشی رو که خودم از اون استفاده می‌کنم رو براتون شرح می‌دهم.

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 فرساست . با توجه به این که در اکثر پروژه‌ها این متدها به صورت Generic نوشته می‌شوند. برای حل این مسئله بعد از مدتی جستجو، کلاسی رو پیدا کردم که این کار رو برام انجام می‌داد. بعد از مطالعه دقیق و مشاهده روش کار کلاس، تغییرات مورد نظرم رو اعمال کردم و الان در اکثر پروژه هام دارم از این کلاس استفاده می‌کنم.

یک مثال از روش استفاده :

برای اینکه از این کلاس در هر دو پروژه (سرور و کلاینت) استفاده می‌کنیم 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 می‌بینید 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 خروجی‌های زیر رو مشاهده می‌کنیم.

```
Common.ExpressionSerializer serializer = new Common.ExpressionSerializer();
var predicate = serializer.Deserialize( expression ) as Expression<Func<Book, bool>>;
var result = predicate {x => ((x.Code > 2) AndAlso (x.Code < 5))}
return
}
```

Body	{((x.Code > 2) AndAlso (x.Code < 5))}
CanReduce	false
DebugView	".Lambda #Lambda1<System.Func`2[Model.Book,System.Boolean]>(Model.Book $x) {\r\n $x.Code > 2 && $x.Code < 5\r\n}"
Name	null
NodeType	Lambda
Parameters	Count = 1
ReturnType	{Name = "Boolean" FullName = "System.Boolean"}
TailCall	false
Type	{Name = "Func`2" FullName = "System.Func`2[[Model.Book, Model, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null],[System.Boolean,
Raw View	

خروجی هم به صورت زیر خواهد بود:

```
var predicate = serializer.Deserialize( expressio
var result = ListOfBook.AsQueryable().Where(
return r  result {System.Collections.Generic.List`1[
          base {System.Linq.EnumerableQuery}
          Non-Public members
          Results View
          [0] {Model.Book}
          [1] {Model.Book}
```

دریافت سورس کامل Expression-Serialization

نظرات خوانندگان

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

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

نویسنده: محمد
تاریخ: ۱۳۹۲/۰۹/۱۹ ۱۷:۱۶

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

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

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

- 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>>;
```

عنوان: **فراخوانی سرویس های WCF به صورت Async**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۴/۰۵ ۸:۳۰
آدرس: www.dotnettips.info
گروه‌ها: WCF, Asynchronous Programming, .NET 4.0, .Net 3.5

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 کاملا اختیاری است و برای خوانایی بهتر از این روش نام گذاری استفاده می‌کنم. متدی که به عنوان شروع عملیات استفاده می‌شود باید حتما OperationContractAttribute رو داشته باشد و مقدار خاصیت AsyncPattern اون هم true باشد. همان طور که می‌بینید 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 حتما داشته باشند. خروجی این متد حتما 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 درنظر گرفتم که لیستی از نوع TEntity است.(TEntityبه صورت generic تعریف شده و نوع ورودی آن هر نوع کلاس غیر abstract می‌تواند باشد). این کلاس برای تمام سرویس‌های 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 می‌شوند و بعد این لیست در کلاس CompletedAsyncResult که ساختیم به عنوان ورودی سازنده پاس داده می‌شوند. چون CompletedAsyncResult از IAsyncResult ارث برده است پس return آن به عنوان خروجی مانعی ندارد. با فراخوانی متد EndGetAllBook سمت کلاینت مقدار asyncResult به عنوان خروجی برگشت داده می‌شود. 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 می‌شود. 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 می‌شود.

```
[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 می‌بینید سرویس بالا از کلاس ClientBase ارث برده است. ClientBase دارای خاصیتی به نام ChannelFactory است که فقط خواندنی می‌باشد. 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 می‌توانیم Channel مربوطه را بسازیم.

کلاسی به نام ServiceMapper ایجاد می‌کنیم که وظیفه آن ساخت ChannelFactory به ازای درخواست‌ها 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 می‌باشد 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 می‌باشد. می‌توان از WSHttpBinding یا NetTCPBinding یا WSDLHttpBinding هم استفاده کرد. البته هر کدام از انواع Bindingها تنظیمات خاص خود را می‌طلبد که در مقاله ای جداگانه بررسی خواهم کرد. پارامتر دوم هم EndPoint ساخته شده می‌باشد. در نهایت با دستور CreateChannel عملیات ساخت Channel به پایان می‌رسد.

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

```
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 می‌بینید 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 :

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 استفاه کردیم) کافیست 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 نمی‌شود؟

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 دیگه‌ای هست؟ پس تابع CreateFactory فقط از روی همین Service Reference یک نمونه می‌سازه؟

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 ۱۵:۲

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 اینترفیس‌ها باید در یک پروژه دیگر برای مثال Service باشد که هیچ گونه رفرنسی به آن نیز نباید داده شود.

عنوان: **مقایسه بین Proxy و ChannelFactory در WCF**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۷/۱۵ ۸:۴۵
آدرس: www.dotnettips.info
برچسب‌ها: WCF, Proxy, ChannelFactory

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

استفاده از ChannelFactory

قصد دارم 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 Contractهای 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. کافیست 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 Collectionها در هنگام انتقال داده و ...) استفاده نمایید.

با استفاده از SvcUtil.exe . کاربرد svcutil.exe در موارد Metadata Export، Service Validtation، XmlSerialization Type و Generator و Metadata Download و ... خلاصه می‌شود. با استفاده از Vs.Net Command Prompt وsvcutil 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، BasicHttpBinding WShttpBinding، NetTcpBidning 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 مستقیما 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

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

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

بله! کافیست 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، اسمبلی‌های مربوط به Domain Model و هم چنین Contractهای سمت سرور را انتخاب نمایید. در این حالت شما از روش Channel Factory برای ساخت سرویس WCF استفاده کرده اید.

نظرات خوانندگان

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

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.

عنوان: **بررسی متد های یک طرفه در WCF**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۸/۱۶ ۹:۵
آدرس: www.dotnettips.info
برچسب‌ها: WCF, OneWay Operation

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( Da
```

TimeoutException was unhandled

The request channel timed out while waiting for a reply after 00:00:59.6147387. Increase the timeout value passed to the call to Request or increase the SendTimeout value on the Binding. The time allotted to this operation may have been a portion of a longer timeout.

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 داه شود این Exception رخ می‌دهد. برای حل این مشکل کافیست در فایل app.config کلاینت 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 کافیست 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 صرفا 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: ۱۳۹۲/۰۸/۱۹ ۱۴:۵۶

می‌خواستم بدانم اگر مثلا 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: ۱۳۹۲/۰۸/۲۰ ۱۵:۵۶

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 IDatabaseContext نماینده کلاس DbContext پروژه است):

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

            public IDatabaseContext 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 وهله سازی کلاس UnitOfWork را با کدهای زیر مزین 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 می‌باشد

```
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 صرفا یک فیلد از نوع string ولی با مقدار Guid برای به دست آوردن Extension مربوطه می‌باشد و تعریف آن در سازنده کلاس خواهد بود.

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

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

عنوان: **Message Header سفارشی در WCF**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۸/۲۰ ۲۱:۵
آدرس: www.dotnettips.info
برچسب‌ها: WCF, Custom Message Header

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 ، کافیست 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 MessageInspcetorهای 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, AppConfiguration.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 ChanelFactory 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 بعنی هدر مورد نظر موجود است. در پایان نیز با استفاده از متد GetHeader، نمونه ساخته شده کلاس ApplicationContext را به دست می‌آوریم.

عنوان: **MTOM در WCF**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۸/۲۴ ۸:۱۵
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 Bindingها از این Encoder استفاده می‌شود)

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

MTOM Message Encoder (در حالت استفاده از Http-Base Bindingها 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 استانداردی وجود دارد به نام **MTOM** یا همان **M**echanism **O**ptimization **T**ransmission **M**essage، 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 خاصیت هایی 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 فیلد هایی 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 پیاده سازی به این روش مد نظر داشته باشید.

عنوان: **اعتبارسنجی سرویس های WCF**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۱۰/۱۹ ۸:۴۵
آدرس: www.dotnettips.info
گروه‌ها: Security, WCF, UserNamePasswordValidator

حالتی را در نظر بگیرید که سرویس‌های یک برنامه در آدرسی مشخص هاست شده اند. اگر اعتبار سنجی برای این سرویس‌ها در نظر گرفته نشود به راحتی می‌توان با در اختیار داشتن آدرس مورد نظر تمام سرویس های برنامه را فراخوانی کرد و اگر رمزگذاری اطلاعات بر روی سرویس‌ها فعال نشده باشد می‌توان تمام اطلاعات این سرویس‌ها 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.Write("BookService host");

            Console.ReadKey();
        }
    }
```

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

حال نوبت به پیاده سازی سمت کلاینت می‌رسد. 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 اعتبار سنجی انجام نشد. برای پیاده سازی اعتبار سنجی باید یک سری تنظیمات بر روی Binding و Hosting سمت سرور و البته کلاینت بر قرار 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.Write("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 اعتبار سنجی 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 اعتبار سنجی به صورت پیش فرض در حالت Windows است (یعنی UserNamePasswordValidationMode برابر مقدار Windows است و اعتبار سنجی بر اساس کاربر انجام می‌شود) . این مورد باید به مقدار Custom تغییر یابد. در انتها نیز باید مدل اعتبار سنجی 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 می‌شود:

```
1 reference
public class CustomUserNamePasswordValidator : UserNamePasswordValidator
{
    0 references
    public override void Validate(string userName, string password)
    {
        if (userName != "Masoud" || password != "Pakdel")
            th  userName  "WrongUserName"  ect userName or password");
    }
}
```

SecurityException was unhandled by user code

An exception of type 'System.Security.SecurityException' occurred in Service.exe but was not handled in user code

دریافت سورس مثال بالا

نظرات خوانندگان

نویسنده: بهمن
تاریخ: ۱۳۹۲/۱۰/۲۱ ۱۱:۵۲

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 پروژه‌های 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: ۱۳۹۲/۱۰/۲۱ ۱۲:۴۵

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 سرویس‌ها در نظر نمی‌گیرد. 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 Certificateها بهره ببریم. در نتیجه برای پیاده سازی مثال بالا به روش WsHttpBinding از makecert.exe برای تولید certificateها استفاده می‌شد(عموما در مثال‌ها و نمونه‌ها از همین روش استفاده میشود) که در اجرای واقعی سرویس‌ها مناسب نیست. در Soap این Certificateها شامل اطلاعات رمزگذاری و مجوزها و کلیدهای عمومی و خصوصی خواهند بود در نتیجه از اهمیت به سزایی بر خوردارند. برای حفظ امنیت سرویس‌ها توصیه می‌شود certificateها را از یک CA (برای مثال VeriSign) خریداری شود یا حداقل می‌توانید از Microsoft Certificate Services که در ویندوزهای سرور نصب می‌شود استفاده نمایید. در واقع اگر یک Certificate Authority وجود نداشته باشد بهتر است از این روش استفاده نشود.

نویسنده: مهرسا
تاریخ: ۱۳۹۲/۱۲/۰۵ ۱۳:۱۵

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: ۱۳۹۲/۱۲/۰۵ ۱۴:۲

Credential خود یک property از نوع ClientCredential در نمونه‌های 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 ChannelFactory و Proxy

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

مرسی از جوابتون
امکان ست کردن تنظیمات سرور در وب کانفیگ هم هست؟ چون من سرویسمو در یک وب سایت گذاشتم.

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

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 certificateها 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: ۱۳۹۲/۱۰/۲۴ ۱۱:۳۵
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 فایده ای نداشت. برنامه‌ی مشتری به راحتی به سرویس وصل می‌شد و با سایر متدهایی که خروجی‌های تایپ‌های اصلی مثل bool و string را برمی گرداند 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 یه 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 وقت کشی‌ها 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: ۱۳۹۲/۱۰/۲۴ ۱۱:۴۱

استفاده از روش دیگر اتصال به سرور هم می‌تونه کمک کنه: مقایسه بین Proxy و ChannelFactory در WCF

نویسنده: افشار محبی
تاریخ: ۱۳۹۲/۱۰/۲۴ ۱۲:۵

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

نویسنده: ابوالفضل رجب پور
تاریخ: ۱۳۹۲/۱۰/۲۴ ۱۲:۲۸

با تشکر از ارجاع خوبتون.
مطلب اینجا راجع به یک نکته و روش رفعش در روش اول بود. مزایا و معایب هر روش سرجای خود محفوظ

عنوان: **ایجاد سرویس چندلایه‌ی WCF با Entity Framework در قالب پروژه - 1**
نویسنده: حامد قنادی
تاریخ: ۱۳۹۲/۱۰/۲۵ ۸:۲۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, LINQ, WCF

در این نوشتار که به صورت آموزش تصویری ارائه می‌شود؛ یک سرویس WCF در Visual Studio 2013 ایجاد می‌کنم، سپس روش استفاده از آن‌را در یک برنامه ویندوزی آموزش خواهم داد. در اینجا در نظرگرفته 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 بازکنید 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 پیش‌گزیده 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 بازشده 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:

Add New Item - prgDataLogger
Installed
Sort by: Default
Search Installed Templates (Ctrl+E)
Visual C#
Code
Data
General
Web
General
Markup
MVC
Razor
Scripts
SignalR
Web API
Web Forms
Windows Forms
WPF
Reporting
Silverlight
SQL Server
Online
ADO.NET Entity Data Model Visual C#
DataSet Visual C#
EF 5.x DbContext Generator Visual C#
EF 6.x DbContext Generator Visual C#
LINQ to SQL Classes Visual C#
SQL Server Database Visual C#
XML File Visual C#
XML Schema Visual C#
Click here to go online and find templates.
Type: Visual C#
A project item for creating an ADO.NET Entity Data Model.
Name:
MyNewsModel.edmx
Add
Cancel

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:

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:

در پایان مدل ما همانند نگاره‌ی زیر خواهد بود.

در بخش پسین درباره‌ی شیوه‌ی دست‌کاری کلاس‌های Entity خواهم نوشت.

عنوان: **ایجاد سرویس چندلایه‌ی WCF با Entity Framework در قالب پروژه - 2**
نویسنده: حامد قنادی
تاریخ: ۱۳۹۲/۱۰/۲۵ ۲۰:۳۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, LINQ, WCF, Entity Framework 6

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، ویژگی ها و سازنده‌ها را در یک فایل و متدها را در فایل دیگر قرار دهید.

به صورت خودکار کلیه‌ی ویژگی‌ها به توجه به پایگاه داده ساخته شده اند. برای نمونه ما برای فیلد IsDeleted در SQL Server 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 بازکنید. 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؛ خواهم نوشت.

عنوان: ایجاد سرویس چندلایه‌ی WCF با Entity Framework در قالب پروژه - 3
نویسنده: حامد قنادی
تاریخ: ۱۳۹۲/۱۰/۲۶ ۱۴:۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, LINQ, WCF, Entity Framework 6

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 برنامه نویس 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 دلواپسی 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: ۱۳۹۲/۱۰/۲۶ ۱۴:۸

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: ۱۳۹۲/۱۰/۲۷ ۴:۱

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 بادید 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: ۱۳۹۲/۱۰/۲۷ ۹:۲۸

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

عنوان: ایجاد سرویس چندلایه‌ی WCF با Entity Framework در قالب پروژه - 4
نویسنده: حامد قنادی
تاریخ: ۱۳۹۲/۱۰/۲۷ ۱۱:۴۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, LINQ, WCF

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 خواهم برد: روی نام interface همانند شکل کلیک کنید و سپس برابر با شکل عمل کنید:

```
namespace MyNewsWCFLibrary
{
    [ServiceContract]
    0 references
    interface IMyNewsService
```

Create derived type
To public

```
ionContract]
List<tblNews> GetAllNews();
```

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:

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

    0 references
    class MyNewsService : IMyNewsService
    {
    }
}
```

Implement interface 'IMyNewsService'
Explicitly implement interface 'IMyNewsService'

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 name 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 خواهم نوشت.

عنوان: ایجاد سرویس چندلایه‌ی WCF با Entity Framework در قالب پروژه - 5
نویسنده: حامد قنادی
تاریخ: ۱۳۹۲/۱۰/۲۸ ۵:۱۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, WCF

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 این‌بار 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 درج‌شده 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 پیش‌گزیده 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 کدغیرفعال‌کردن 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 قرارگرفته 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 GetAllCategory و GetAllNews 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 وگرنه 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 خواهم نوشت.

عنوان: ایجاد سرویس چندلایه‌ی WCF با Entity Framework در قالب پروژه - 7
نویسنده: حامد قنادی
تاریخ: ۱۳۹۲/۱۰/۳۰ ۵:۱۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, WCF, Entity Framework 6

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 این‌کار از Solution Explorer روی راه حل MyNews راست‌کلیک 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 بازشده 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>
    <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.

نظرات خوانندگان

نویسنده: محمد سعید
تاریخ: ۱۳۹۳/۱۲/۲۱ ۱۲:۵۵

با سلام؛ قبل از هر چیز ممنون از آموزش خوبتون. یک سوال در رابطه با قسمت اول این آموزش داشتم. من دقیقا با آموزش شما پیش رفتم و در قسمت هفتم آموزشتون در اجرای پروژه دچار مشکل شدم و در مرورگرم با خطای زیر مواجه شدم:

```
HTTP Error 403.14 - Forbidden
The Web server is configured to not list the contents of this directory.
Most likely causes:

    A default document is not configured for the requested URL, and directory browsing is not enabled
on the server.

Things you can try:

    If you do not want to enable directory browsing, ensure that a default document is configured and
that the file exists.
    Enable directory browsing using IIS Manager.
        Open IIS Manager.
        In the Features view, double-click Directory Browsing.
        On the Directory Browsing page, in the Actions pane, click Enable.
    Verify that the configuration/system.webServer/directoryBrowse@enabled attribute is set to true in
the site or application configuration file.

Detailed Error Information:
Module DirectoryListingModule
Notification ExecuteRequestHandler
Handler StaticFile
Error Code 0x00000000
Requested URL http://localhost:80/3724/
Physical Path C:\inetpub\wwwroot\3724\
Logon Method Anonymous
Logon User Anonymous
```

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۱۲/۲۱ ۱۳:۳

آدرس http://localhost:80/ **3724** / با آدرس http://localhost: **3724** /name.svc یکی نیست. در اینجا 3724 شماره پورتی است که IIS Express به صورت خودکار و اتفاقی به برنامه انتساب داده (به خواص پروژه قسمت web آن مراجعه کنید. این شماره ممکن است برای شما متفاوت باشه).

عنوان: ایجاد سرویس چندلایه‌ی WCF با Entity Framework در قالب پروژه - 8
نویسنده: حامد قنادی
تاریخ: ۱۳۹۲/۱۰/۳۰ ۲۲:۳۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, WCF, C#.NET, Entity Framework 6

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 آن‌را 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 بازکنید 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 راست‌کلیک کنید و گزینه‌ی Properties را انتخاب کنید. در پنجره‌ی بازشده 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.

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

عنوان: ایجاد سرویس چندلایه‌ی WCF با Entity Framework در قالب پروژه - 9
نویسنده: حامد قنادی
تاریخ: ۱۳۹۲/۱۱/۰۱ ۲۲:۰
آدرس: www.dotnettips.info
گروه‌ها: C#, Entity framework, WCF

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 آن‌را 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 روی‌داد 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 آن‌را برابر با Public کنید.

روی Combobox کلیک کنید، سپس ویژگی DataBinding -> Text آن‌را خالی کنید. سپس روی فلش بالای Combobox دسته خبر کلیک کنید و تنظیمات آن‌را 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 مقداردهی 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 آن‌را 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 خواهم 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 واکشی 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 واکشی 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 واکشی 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 سایز 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>
```

```
        <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 یه 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 خواهم پرداخت.
پیروز باشید.

نویسنده: علیرضا طهوری
تاریخ: ۱۳۹۲/۱۲/۲۴ ۱۱:۱

سلام
با تشکر از این آموزش. فقط یه 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: « پیاده 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 میگردونه. این خطا رو میتوان 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: ۱۳۹۳/۰۱/۲۶ ۱۴:۴۲

- 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: ۱۳۹۳/۰۴/۱۵ ۹:۱

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: ۱۳۹۳/۰۴/۱۵ ۹:۸

- 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: ۱۳۹۳/۰۴/۱۵ ۱۲:۲۲

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 روا 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".
```

عنوان: **استفاده از EF در اپلیکیشن های N-Tier : قسمت دوم**
نویسنده: آرمین ضیاء
تاریخ: ۱۳۹۲/۱۱/۰۷ ۷:۴۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, Tips, WCF, N-Layer Architecture, Entity Framework 6

در قسمت قبل معماری اپلیکیشن‌های N-Tier و بروز رسانی موجودیت‌های منفصل توسط Web API را بررسی کردیم. در این قسمت بروز رسانی موجودیت‌های منفصل توسط WCF را بررسی می‌کنیم.

بروز رسانی موجودیت‌های منفصل توسط WCF

سناریویی را در نظر بگیرید که در آن عملیات CRUD توسط WCF پیاده سازی شده اند و دسترسی داده‌ها با مدل Code-First انجام می‌شود. فرض کنید مدل اپلیکیشن مانند تصویر زیر است.

همانطور که می‌بینید مدل ما متشکل از پست‌ها و نظرات کاربران است. برای ساده نگاه داشتن مثال جاری، اکثر فیلدها حذف شده اند. مثلا متن پست ها، نویسنده، تاریخ و زمان انتشار و غیره. می‌خواهیم تمام کد دسترسی داده‌ها را در یک سرویس WCF پیاده سازی کنیم تا کلاینت‌ها بتوانند عملیات CRUD را توسط آن انجام دهند. برای ساختن این سرویس مراحل زیر را دنبال کنید.
در ویژوال استودیو پروژه جدیدی از نوع Class Library بسازید و نام آن را به Recipe2 تغییر دهید.
با استفاده از NuGet Package Manager کتابخانه Entity Framework 6 را به پروژه اضافه کنید.
سه کلاس با نام‌های Post, Comment و Recipe2Context به پروژه اضافه کنید. کلاس‌های Post و Comment موجودیت‌های مدل ما هستند که به جداول متناظرشان نگاشت می‌شوند. کلاس Recipe2Context آبجکت DbContext ما خواهد بود که بعنوان درگاه عملیاتی EF عمل می‌کند. دقت کنید که خاصیت‌های لازم WCF یعنی DataContract و DataMember در کلاس‌های موجودیت‌ها بدرستی استفاده می‌شوند. لیست زیر کد این کلاس‌ها را نشان می‌دهد.

```
[DataContract(IsReference = true)]
public class Post
{
    public Post()
    {
        comments = new HashSet<Comments>();
    }

    [DataMember]
    public int PostId { get; set; }
```

```
    [DataMember]
    public string Title { get; set; }
    [DataMember]
    public virtual ICollection<Comment> Comments { get; set; }
}

[DataContract(IsReference=true)]
public class Comment
{
    [DataMember]
    public int CommentId { get; set; }
    [DataMember]
    public int PostId { get; set; }
    [DataMember]
    public string CommentText { get; set; }
    [DataMember]
    public virtual Post Post { get; set; }
}

public class EFRecipesEntities : DbContext
{
    public EFRecipesEntities() : base("name=EFRecipesEntities") {}

    public DbSet<Post> posts;
    public DbSet<Comment> comments;
}
```

یک فایل App.config به پروژه اضافه کنید و رشته اتصال زیر را به آن اضافه نمایید.

```
<connectionStrings>
  <add name="Recipe2ConnectionString"
    connectionString="Data Source=.;
    Initial Catalog=EFRecipes;
    Integrated Security=True;
    MultipleActiveResultSets=True"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

حال یک پروژه WCF به Solution جاری اضافه کنید. برای ساده نگاه داشتن مثال جاری، نام پیش فرض Service1 را بپذیرید. فایل IService1.cs را باز کنید و کد زیر را با محتوای آن جایگزین نمایید.

```
[ServiceContract]
public interface IService1
{
    [OperationContract]
    void Cleanup();
    [OperationContract]
    Post GetPostByTitle(string title);
    [OperationContract]
    Post SubmitPost(Post post);
    [OperationContract]
    Comment SubmitComment(Comment comment);
    [OperationContract]
    void DeleteComment(Comment comment);
}
```

فایل Service1.svc.cs را باز کنید و کد زیر را با محتوای آن جایگزین نمایید. بیاد داشته باشید که پروژه Recipe2 را ارجاع کنید و فضای نام آن را وارد نمایید. همچنین کتابخانه EF 6 را باید به پروژه اضافه کنید.

```
public class Service1 : IService
{
    public void Cleanup()
    {
        using (var context = new EFRecipesEntities())
        {
            context.Database.ExecuteSqlCommand("delete from [comments]");
            context. Database.ExecuteSqlCommand ("delete from [posts]");
        }
    }

    public Post GetPostByTitle(string title)
```

```
    {
        using (var context = new EFRecipesEntities())
        {
            context.Configuration.ProxyCreationEnabled = false;
            var post = context.Posts.Include(p => p.Comments).Single(p => p.Title == title);
            return post;
        }
    }

    public Post SubmitPost(Post post)
    {
        context.Entry(post).State =
            // if Id equal to 0, must be insert; otherwise, it's an update
            post.PostId == 0 ? EntityState.Added : EntityState.Modified;
        context.SaveChanges();
        return post;
    }

    public Comment SubmitComment(Comment comment)
    {
        using (var context = new EFRecipesEntities())
        {
            context.Comments.Attach(comment);
            if (comment.CommentId == 0)
            {
                // this is an insert
                context.Entry(comment).State = EntityState.Added);
            }
            else
            {
                // set single property to modified, which sets state of entity to modified, but
                // only updates the single property – not the entire entity
                context.entry(comment).Property(x => x.CommentText).IsModified = true;
            }
            context.SaveChanges();
            return comment;
        }
    }

    public void DeleteComment(Comment comment)
    {
        using (var context = new EFRecipesEntities())
        {
            context.Entry(comment).State = EntityState.Deleted;
            context.SaveChanges();
        }
    }
}
```

در آخر پروژه جدیدی از نوع Windows Console Application به Solution جاری اضافه کنید. از این اپلیکیشن بعنوان کلاینتی برای تست سرویس WCF استفاده خواهیم کرد. فایل program.cs را باز کنید و کد زیر را با محتوای آن جایگزین نمایید. روی نام پروژه کلیک راست کرده و گزینه Add Service Reference را انتخاب کنید، سپس ارجاعی به سرویس Service1 اضافه کنید. رفرنسی هم به کتابخانه کلاس‌ها که در ابتدای مراحل ساختید باید اضافه کنید.

```
class Program
{
    static void Main(string[] args)
    {
        using (var client = new ServiceReference2.Service1Client())
        {
            // cleanup previous data
            client.Cleanup();
            // insert a post
            var post = new Post { Title = "POCO Proxies" };
            post = client.SubmitPost(post);
            // update the post
            post.Title = "Change Tracking Proxies";
            client.SubmitPost(post);
            // add a comment
            var comment1 = new Comment { CommentText = "Virtual Properties are cool!", PostId =
post.PostId };
            var comment2 = new Comment { CommentText = "I use ICollection<T> all the time", PostId =
post.PostId };
            comment1 = client.SubmitComment(comment1);
            comment2 = client.SubmitComment(comment2);
            // update a comment
```

```
            comment1.CommentText = "How do I use ICollection<T>?";
            client.SubmitComment(comment1);
            // delete comment 1
            client.DeleteComment(comment1);
            // get posts with comments
            var p = client.GetPostByTitle("Change Tracking Proxies");
            Console.WriteLine("Comments for post: {0}", p.Title);
            foreach (var comment in p.Comments)
            {
                Console.WriteLine("\tComment: {0}", comment.CommentText);
            }
        }
    }
}
```

اگر اپلیکیشن کلاینت (برنامه کنسول) را اجرا کنید با خروجی زیر مواجه می‌شوید.

```
Comments for post: Change Tracking Proxies
Comment: I use ICollection<T> all the time
```

شرح مثال جاری

ابتدا با اپلیکیشن کنسول شروع می‌کنیم، که کلاینت سرویس ما است. نخست در یک بلاک {} using وهله ای از کلاینت سرویس مان ایجاد می‌کنیم. درست همانطور که وهله ای از یک EF Context می‌سازیم. استفاده از بلوک‌های using توصیه می‌شود چرا که متد Dispose بصورت خودکار فراخوانی خواهد شد، چه بصورت عادی چه هنگام بروز خطا. پس از آنکه وهله ای از کلاینت سرویس را در اختیار داشتیم، متد Cleanup را صدا می‌زنیم. با فراخوانی این متد تمام داده‌های تست پیشین را حذف می‌کنیم. در چند خط بعدی، متد SubmitPost را روی سرویس فراخوانی می‌کنیم. در پیاده سازی فعلی شناسه پست را بررسی می‌کنیم. اگر مقدار شناسه صفر باشد، خاصیت State موجودیت را به Added تغییر می‌دهید تا رکورد جدیدی ثبت کنیم. در غیر اینصورت فرض بر این است که چنین موجودیتی وجود دارد و قصد ویرایش آن را داریم، بنابراین خاصیت State را به Modified تغییر می‌دهیم. از آنجا که مقدار متغیرهای int بصورت پیش فرض صفر است، با این روش می‌توانیم وضعیت پست‌ها را مشخص کنیم. یعنی تعیین کنیم رکورد جدیدی باید ثبت شود یا رکوردی موجود بروز رسانی گردد. رویکردی بهتر آن است که پارامتری اضافی به متد پاس دهیم، یا متدی مجزا برای ثبت رکوردهای جدید تعریف کنیم. مثلا رکوردی با نام InsertPost. در هر حال، بهترین روش بستگی به ساختار اپلیکیشن شما دارد.

اگر پست جدیدی ثبت شود، خاصیت PostId با مقدار مناسب جدید بروز رسانی می‌شود و وهله پست را باز می‌گردانیم. ایجاد و بروز رسانی نظرات کاربران مشابه ایجاد و بروز رسانی پست‌ها است، اما با یک تفاوت اساسی: بعنوان یک قانون، هنگام بروز رسانی نظرات کاربران تنها فیلد متن نظر باید بروز رسانی شود. بنابراین با فیلدهای دیگری مانند تاریخ انتشار و غیره اصلا کاری نخواهیم داشت. بدین منظور تنها خاصیت CommentText را بعنوان Modified علامت گذاری می‌کنیم. این امر منجر می‌شود که Entity Framework عبارتی برای بروز رسانی تولید کند که تنها این فیلد را در بر می‌گیرد. توجه داشته باشید که این روش تنها در صورتی کار می‌کند که بخواهید یک فیلد واحد را بروز رسانی کنید. اگر می‌خواستیم فیلدهای بیشتری را در موجودیت Comment بروز رسانی کنیم، باید مکانیزمی برای ردیابی تغییرات در سمت کلاینت در نظر می‌گرفتیم. در مواقعی که خاصیت‌های متعددی می‌توانند تغییر کنند، معمولا بهتر است کل موجودیت بروز رسانی شود تا اینکه مکانیزمی پیچیده برای ردیابی تغییرات در سمت کلاینت پیاده گردد. بروز رسانی کل موجودیت بهینه‌تر خواهد بود.

برای حذف یک دیدگاه، متد Entry را روی آبجکت DbContext فراخوانی می‌کنیم و موجودیت مورد نظر را بعنوان آرگومان پاس می‌دهیم. این امر سبب می‌شود که موجودیت مورد نظر بعنوان Deleted علامت گذاری شود، که هنگام فراخوانی متد SaveChanges اسکریپت لازم برای حذف رکورد را تولید خواهد کرد.

در آخر متد GetPostByTitle یک پست را بر اساس عنوان پیدا کرده و تمام نظرات کاربران مربوط به آن را هم بارگذاری می‌کند. از آنجا که ما کلاس‌های POCO را پیاده سازی کرده ایم، Entity Framework آبجکتی را بر می‌گرداند که Dynamic Proxy نامیده می‌شود. این آبجکت پست و نظرات مربوط به آن را در بر خواهد گرفت. متاسفانه WCF نمی‌تواند آبجکت‌های پروکسی را مرتب سازی (serialize) کند. اما با غیرفعال کردن قابلیت ایجاد پروکسی‌ها (ProxyCreationEnabled=false) ما به Entity Framework

می‌گوییم که خود آبجکت‌های اصلی را بازگرداند. اگر سعی کنید آبجکت پروکسی را سریال کنید با پیغام خطای زیر مواجه خواهید شد:

```
The underlying connection was closed: The connection was closed unexpectedly
```

می توانیم غیرفعال کردن تولید پروکسی را به متد سازنده کلاس سرویس منتقل کنیم تا روی تمام متدهای سرویس اعمال شود.

در این قسمت دیدیم چگونه می‌توانیم از آبجکت‌های POCO برای مدیریت عملیات CRUD توسط WCF استفاده کنیم. از آنجا که هیچ اطلاعاتی درباره وضعیت موجودیت‌ها روی کلاینت ذخیره نمی‌شود، متدهایی مجزا برای عملیات CRUD ساختیم. در قسمت‌های بعدی خواهیم دید چگونه می‌توان تعداد متدهایی که سرویس مان باید پیاده سازی کند را کاهش داد و چگونه ارتباطات بین کلاینت و سرور را ساده‌تر کنیم.

نظرات خوانندگان

نویسنده: جلال
تاریخ: ۱۳۹۲/۱۲/۱۰ ۲۰:۲۰

در این سناریو، فرض را بر این گذاشته اید که موجودیتها یا جدید هستند و یا ویرایش شده اند و بنابراین حتی اگر یک پست کامنتهایی داشته باشد که ویرایش نشده اند، برای انها دستور update صادر میشود و این، در مواردیکه تعداد کامنتها(که البته همیشه این موجودیتهای اینگونه به سادگی کامنت نیستند) زیاد باشد، روی کارایی تأثیر منفی خواهد داشت، چه راهی برای تشخیص موجودیتهایی که سمت کلاینت تغییری نکرده اند، پیشنهاد میدهید؟

عنوان: **پیاده سازی InstanceProvider برای سرویس های WCF**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۱۱/۱۰ ۲۰:۵
آدرس: www.dotnettips.info
گروه‌ها: WCF, Dependency Injection, InstanceProvider

اگر قصد داشته باشیم که تزریق وابستگی (Dependency Injection) را برای سرویس‌های WCF پیاده سازی کنیم نیاز به یک Instance Provider سفارشی داریم. در ابتدا باید سرویس‌های مورد نظر را در یک Ioc Container رجیستر نماییم سپس با استفاده از InstanceProvider عملیات وهله سازی از سرویس‌ها 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 می‌بینید برای عملیات وهله سازی از این سرویس نیاز به تزریق کلاس BookRepository است که این کلاس باید اینترفیس IBookRepository را پیاده 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 نمایم:

```
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 می‌توان عملیات وهله سازی سرویس‌های 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 وابستگی‌های 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 سرویس‌ها 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 UnityInstanceProvinder(
serviceDescription.ServiceType );
                            } );
                    }
                } );
        }

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

در متد ApplyDispatchBehavior همان طور دیده می‌شود به ازای تمام EndPointهای هر ChannelDispatcher یک نمونه از کلاس UnityInstanceProvider به همراه پارامتر سازنده آن که نوع سرویس مورد نظر است به خاصیت InstanceProvider در DispatchRuntime معرفی می‌گردد.

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 مثلا 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 نهایتا تشکیل میشه، تمام این‌ها داخل یک Application Domain اجرا می‌شن. یعنی عملا 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.

عنوان: **استفاده از EF در اپلیکیشن های N-Tier : قسمت ششم**
نویسنده: آرمین ضیاء
تاریخ: ۱۳۹۲/۱۱/۱۱ ۰:۲۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, Tips, WCF, N-Layer Architecture, Web API, Entity Framework 6

در قسمت قبل رویکردهای مختلف برای حذف موجودیت‌های منفصل را بررسی کردیم. در این قسمت مدیریت همزمانی یا Concurrency را بررسی خواهیم کرد.

فرض کنید می‌خواهیم مطمئن شویم که موجودیتی که توسط یک کلاینت WCF تغییر کرده است، تنها در صورتی بروز رسانی شود که شناسه (token) همزمانی آن تغییر نکرده باشد. به بیان دیگر شناسه ای که هنگام دریافت موجودیت بدست می‌آید، هنگام بروز رسانی باید مقداری یکسان داشته باشد.

مدل زیر را در نظر بگیرید.

می‌خواهیم یک سفارش (order) را توسط یک سرویس WCF بروز رسانی کنیم در حالی که اطمینان حاصل می‌کنیم موجودیت سفارش از زمانی که دریافت شده تغییری نکرده است. برای مدیریت این وضعیت دو رویکرد تقریبا متفاوت را بررسی می‌کنیم. در هر دو رویکرد از یک ستون همزمانی استفاده می‌کنیم، در این مثال فیلد TimeStamp.

در ویژوال استودیو پروژه جدیدی از نوع WCF Service Library بسازید و نام آن را به Recipe6 تغییر دهید.
روی نام پروژه کلیک راست کنید و گزینه Add New Item را انتخاب کنید. سپس گزینه‌های Data -> Entity Data Model را برگزینید. از ویزارد ویژوال استودیو برای اضافه کردن مدل جاری و جدول Orders استفاده کنید. در EF Designer روی فیلد TimeStamp کلیک راست کنید و گزینه Properties را انتخاب کنید. سپس مقدار CuncurrencyMode آنرا به Fixed تغییر دهید.
فایل IService1.cs را باز کنید و تعریف سرویس را مطابق لیست زیر بروز رسانی کنید.

```
[ServiceContract]
public interface IService1
{
    [OperationContract]
```

```
    Order InsertOrder();
    [OperationContract]
    void UpdateOrderWithoutRetrieving(Order order);
    [OperationContract]
    void UpdateOrderByRetrieving(Order order);
}
```

فایل Service1.cs را باز کنید و پیاده سازی سرویس را مطابق لیست زیر تکمیل کنید.

```
public class Service1 : IService1
{
    public Order InsertOrder()
    {
        using (var context = new EFRecipesEntities())
        {
            // remove previous test data
            context.Database.ExecuteSqlCommand("delete from [orders]");
            var order = new Order
            {
                Product = "Camping Tent",
                Quantity = 3,
                Status = "Received"
            };
            context.Orders.Add(order);
            context.SaveChanges();
            return order;
        }
    }

    public void UpdateOrderWithoutRetrieving(Order order)
    {
        using (var context = new EFRecipesEntities())
        {
            try
            {
                context.Orders.Attach(order);
                if (order.Status == "Received")
                {
                    context.Entry(order).Property(x => x.Quantity).IsModified = true;
                    context.SaveChanges();
                }
            }
            catch (OptimisticConcurrencyException ex)
            {
                // Handle OptimisticConcurrencyException
            }
        }
    }

    public void UpdateOrderByRetrieving(Order order)
    {
        using (var context = new EFRecipesEntities())
        {
            // fetch current entity from database
            var dbOrder = context.Orders
            .Single(o => o.OrderId == order.OrderId);
            if (dbOrder != null &&
                // execute concurrency check
                StructuralComparisons.StructuralEqualityComparer.Equals(order.TimeStamp,
dbOrder.TimeStamp))
            {
                dbOrder.Quantity = order.Quantity;
                context.SaveChanges();
            }
            else
            {
                // Add code to handle concurrency issue
            }
        }
    }
}
```

برای تست این سرویس به یک کلاینت نیاز داریم. پروژه جدیدی از نوع Console Application به راه حل جاری اضافه کنید و کد آن را مطابق لیست زیر تکمیل کنید. با کلیک راست روی نام پروژه و انتخاب گزینه Add Service Reference سرویس پروژه را هم ارجاع کنید. دقت کنید که ممکن است پیش از آنکه بتوانید سرویس را ارجاع کنید نیاز باشد روی آن کلیک راست کرده و از منوی

Debug گزینه Start Instance را انتخاب کنید تا وهله از سرویس به اجرا در بیاید.

```
class Program
{
    static void Main(string[] args)
    {
        var service = new Service1Client();
        var order = service.InsertOrder();
        order.Quantity = 5;
        service.UpdateOrderWithoutRetrieving(order);
        order = service.InsertOrder();
        order.Quantity = 3;
        service.UpdateOrderByRetrieving(order);
    }
}
```

اگر به خط اول متد ()Main یک breakpoint اضافه کنید و اپلیکیشن را اجرا کنید می‌توانید افزودن و بروز رسانی یک Order با هر دو رویکرد را بررسی کنید.

شرح مثال جاری

متد ()InsertOrder داده‌های پیشین را حذف می‌کند، سفارش جدیدی می‌سازد و آن را در دیتابیس ثبت می‌کند. در آخر موجودیت جدید به کلاینت باز می‌گردد. موجودیت بازگشتی هر دو مقدار OrderId و TimeStamp را دارا است که توسط دیتابیس تولید شده اند. سپس در کلاینت از دو رویکرد نسبتا متفاوت برای بروز رسانی موجودیت استفاده می‌کنیم.

در رویکرد نخست، متد ()UpdateOrderWithoutRetrieving موجودیت دریافت شده از کلاینت را Attach می‌کند و چک می‌کند که مقدار فیلد Status چیست. اگر مقدار این فیلد "Received" باشد، فیلد Quantity را با EntityState.Modified علامت گذاری می‌کنیم و متد ()SaveChanges را فراخوانی می‌کنیم. EF دستورات لازم برای بروز رسانی را تولید می‌کند، که فیلد quantity را مقدار دهی کرده و یک عبارت where هم دارد که فیلدهای OrderId و TimeStamp را چک می‌کند. اگر مقدار TimeStamp توسط یک دستور بروز رسانی تغییر کرده باشد، بروز رسانی جاری با خطا مواجه خواهد شد. برای مدیریت این خطا ما بدنه کد را در یک بلاک try/catch قرار می‌دهیم، و استثنای OptimisticConcurrencyException را مهار می‌کنیم. این کار باعث می‌شود اطمینان داشته باشیم که موجودیت Order دریافت شده از متد ()InsertOrder تاکنون تغییری نکرده است. دقت کنید که در مثال جاری تمام خواص موجودیت بروز رسانی می‌شوند، صرفنظر از اینکه تغییر کرده باشند یا خیر.

رویکرد دوم نشان می‌دهد که چگونه می‌توان وضعیت همزمانی موجودیت را پیش از بروز رسانی مشخصا دریافت و بررسی کرد. در اینجا می‌توانید مقدار TimeStamp موجودیت را از دیتابیس بگیرید و آن را با مقدار موجودیت کلاینت مقایسه کنید تا وجود تغییرات مشخص شود. این رویکرد در متد ()UpdateOrderByRetrieving نمایش داده شده است. گرچه این رویکرد برای تشخیص تغییرات خواص موجودیت‌ها و یا روابط شان مفید و کارآمد است، اما بهترین روش هم نیست. مثلا ممکن است از زمانی که موجودیت را از دیتابیس دریافت می‌کنید، تا زمانی که مقدار TimeStamp آن را مقایسه می‌کنید و نهایتا متد ()SaveChanges را صدا میزنید، موجودیت شما توسط کلاینت دیگری بروز رسانی شده باشد.

مسلما رویکرد دوم هزینه برتر از رویکرد اولی است، چرا که برای مقایسه مقادیر همزمانی موجودیت ها، یکبار موجودیت را از دیتابیس دریافت می‌کنید. اما این رویکرد در مواقعی که Object graphهای بزرگ یا پیچیده (complex) دارید بهتر است، چون پیش از ارسال موجودیت‌ها به context در صورت برابر نبودن مقادیر همزمانی پروسس را لغو می‌کنید.

نظرات خوانندگان

نویسنده: Senator
تاریخ: ۱۳۹۲/۱۱/۱۶ ۱۸:۵۴

خیلی ممنون.
عالی بود ...

عنوان: **استفاده از EF در اپلیکیشن های N-Tier : قسمت هفتم**
نویسنده: آرمین ضیاء
تاریخ: ۱۳۹۲/۱۱/۱۱ ۱:۱۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, Tips, WCF, N-Layer Architecture, Entity Framework 6

در قسمت قبلی مدیریت همزمانی در بروز رسانی‌ها را بررسی کردیم. در این قسمت مرتب سازی (serialization) پراکسی‌ها در سرویس‌های WCF را بررسی خواهیم کرد.

مرتب سازی پراکسی‌ها در سرویس‌های WCF

فرض کنید یک پراکسی دینامیک (dynamic proxy) از یک کوئری دریافت کرده اید. حال می‌خواهید این پراکسی را در قالب یک آبجکت CLR سریال کنید. هنگامی که آبجکت‌های موجودیت را بصورت POCO-based پیاده سازی می‌کنید، EF بصورت خودکار یک آبجکت دینامیک مشتق شده را در زمان اجرا تولید می‌کند که *dynamix proxy* نام دارد. این آبجکت برای هر موجودیت POCO تولید می‌شود. این آبجکت پراکسی بسیاری از خواص مجازی (virtual) را بازنویسی می‌کند تا عملیاتی مانند ردیابی تغییرات و بارگذاری تنبل را انجام دهد.

فرض کنید مدلی مانند تصویر زیر دارید.

ما از کلاس *ProxyDataContractResolver* برای deserialize کردن یک آبجکت پراکسی در سمت سرور و تبدیل آن به یک آبجکت POCO روی کلاینت WCF استفاده می‌کنیم. مراحل زیر را دنبال کنید.

در ویژوال استودیو پروژه جدیدی از نوع WCF Service Application بسازید. یک Entity Data Model به پروژه اضافه کنید و جدول Clients را مطابق مدل مذکور ایجاد کنید.
کلاس POCO تولید شده توسط EF را باز کنید و کلمه کلیدی virtual را به تمام خواص اضافه کنید. این کار باعث می‌شود EF کلاس‌های پراکسی دینامیک تولید کند. کد کامل این کلاس در لیست زیر قابل مشاهده است.

```
public class Client
{
    public virtual int ClientId { get; set; }
    public virtual string Name { get; set; }
    public virtual string Email { get; set; }
}
```

نکته: بیاد داشته باشید که هرگاه مدل EDMX را تغییر می‌دهید، EF بصورت خودکار کلاس‌های موجودیت‌ها را مجددا ساخته و تغییرات مرحله قبلی را بازنویسی می‌کند. بنابراین یا باید این مراحل را هر بار تکرار کنید، یا قالب T4 مربوطه را ویرایش کنید. اگر هم از مدل Code-First استفاده می‌کنید که نیازی به این کارها نخواهد بود.

ما نیاز داریم که DataContractSerializer از یک کلاس ProxyDataContractResolver استفاده کند تا پراکسی کلاینت را به موجودیت کلاینت برای کلاینت سرویس WCF تبدیل کند. یعنی client proxy به client entity تبدیل شود، که نهایتا در اپلیکیشن کلاینت سرویس استفاده خواهد شد. بدین منظور یک ویژگی operation behavior می‌سازیم و آن را به متد ()GetClient در سرویس الحاق می‌کنیم. برای ساختن ویژگی جدید از کدی که در لیست زیر آمده استفاده کنید. بیاد داشته باشید که کلاس ProxyDataContractResolver در فضای نام Entity Framework قرار دارد.

```
using System.ServiceModel.Description;
using System.ServiceModel.Channels;
using System.ServiceModel.Dispatcher;
using System.Data.Objects;

namespace Recipe8
{
    public class ApplyProxyDataContractResolverAttribute :
        Attribute, IOperationBehavior
    {
        public void AddBindingParameters(OperationDescription description,
            BindingParameterCollection parameters)
        {
        }
        public void ApplyClientBehavior(OperationDescription description,
            ClientOperation proxy)
        {
            DataContractSerializerOperationBehavior
                dataContractSerializerOperationBehavior =
                    description.Behaviors
                    .Find<DataContractSerializerOperationBehavior>();
                dataContractSerializerOperationBehavior.DataContractResolver =
                    new ProxyDataContractResolver();
        }
        public void ApplyDispatchBehavior(OperationDescription description,
            DispatchOperation dispatch)
        {
            DataContractSerializerOperationBehavior
                dataContractSerializerOperationBehavior =
                    description.Behaviors
                    .Find<DataContractSerializerOperationBehavior>();
            dataContractSerializerOperationBehavior.DataContractResolver =
                new ProxyDataContractResolver();
        }
        public void Validate(OperationDescription description)
        {
        }
    }
}
```

فایل IService1.cs را باز کنید و کد آن را مطابق لیست زیر تکمیل نمایید.

```
[ServiceContract]
public interface IService1
{
    [OperationContract]
    void InsertTestRecord();
    [OperationContract]
    Client GetClient();
    [OperationContract]
    void Update(Client client);
}
```

فایل IService1.svc.cs را باز کنید و پیاده سازی سرویس را مطابق لیست زیر تکمیل کنید.

```
public class Client
{
    [ApplyProxyDataContractResolver]
    public Client GetClient()
```

```
    {
        using (var context = new EFRecipesEntities())
        {
            context.Cofiguration.LazyLoadingEnabled = false;
            return context.Clients.Single();
        }
    }
    public void Update(Client client)
    {
        using (var context = new EFRecipesEntities())
        {
            context.Entry(client).State = EntityState.Modified;
            context.SaveChanges();
        }
    }
    public void InsertTestRecord()
    {
        using (var context = new EFRecipesEntities())
        {
            // delete previous test data
            context.ExecuteSqlCommand("delete from [clients]");
            // insert new test data
            context.ExecuteStoreCommand(@"insert into
                [clients](Name, Email) values ('Armin Zia','armin.zia@gmail.com')");
        }
    }
}
```

حال پروژه جدیدی از نوع Console Application به راه حل جاری اضافه کنید. این اپلیکیشن، کلاینت تست ما خواهد بود. پروژه سرویس را ارجاع کنید و کد کلاینت را مطابق لیست زیر تکمیل نمایید.

```
using Recipe8Client.ServiceReference1;

namespace Recipe8Client
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var serviceClient = new Service1Client())
            {
                serviceClient.InsertTestRecord();

                var client = serviceClient.GetClient();
                Console.WriteLine("Client is: {0} at {1}", client.Name, client.Email);

                client.Name = "Armin Zia";
                client.Email = "arminzia@live.com";
                serviceClient.Update(client);

                client = serviceClient.GetClient();
                Console.WriteLine("Client changed to: {0} at {1}", client.Name, client.Email);
            }
        }
    }
}
```

اگر اپلیکیشن کلاینت را اجرا کنید با خروجی زیر مواجه خواهید شد.

```
Client is: Armin Zia at armin.zia@gmail.com
Client changed to: Armin Zia at arminzia@live.com
```

شرح مثال جاری

مایکروسافت استفاده از آبجکت‌های POCO با WCF را توصیه می‌کند چرا که مرتب سازی (serialization) آبجکت موجودیت را ساده‌تر می‌کند. اما در صورتی که از آبجکت‌های POCO ای استفاده می‌کنید که *changed-based notification* دارند (یعنی خواص موجودیت را با virtual علامت گذاری کرده اید و کلکسیون‌های خواص پیمایشی از نوع ICollection هستند)، آنگاه EF برای موجودیت‌های بازگشتی کوئری‌ها *پراکسی‌های دینامیک* تولید خواهد کرد.

استفاده از پراکسی‌های دینامیک با WCF دو مشکل دارد. مشکل اول مربوط به سریال کردن پراکسی است. کلاس DataContractSerializer تنها قادر به serialize/deserialize کردن انواع شناخته شده (known types) است. در مثال جاری این یعنی موجودیت Client. اما از آنجا که EF برای موجودیت‌ها پراکسی می‌سازد، حالا باید آبجکت پراکسی را سریال کنیم، نه خود کلاس Client را. اینجا است که از DataContractResolver استفاده می‌کنیم. این کلاس می‌تواند حین سریال کردن آبجکت ها، نوعی را به نوع دیگر تبدیل کند. برای استفاده از این کلاس ما یک ویژگی سفارشی ساختیم، که پراکسی‌ها را به کلاس‌های POCO تبدیل می‌کند. سپس این ویژگی را به متد GetClient() اضافه کردیم. این کار باعث می‌شود که پراکسی دینامیکی که توسط متد GetClient() برای موجودیت Client تولید می‌شود، به درستی سریال شود.

مشکل دوم استفاده از پراکسی‌ها با WCF مربوط به بارگذاری تبل یا Lazy Loading می‌شود. هنگامی که DataContractSerializer موجودیت‌ها را سریال می‌کند، تمام خواص موجودیت را دستیابی خواهد کرد که منجر به اجرای lazy-loading روی خواص پیمایشی می‌شود. مسلما این رفتار را نمی‌خواهیم. برای رفع این مشکل، مشخصا قابلیت بارگذاری تنبل را خاموش یا غیرفعال کرده ایم.

عنوان: **پیاده سازی ServiceHostFactory سفارشی در WCF**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۱۱/۱۲ ۲۱:۵
آدرس: www.dotnettips.info
گروه‌ها: WCF, Self-Hosting, ServiceFactory, IIS-Hosting

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 مستقیما 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.

عنوان: Long Polling در WCF
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۱۱/۱۷ ۱۴:۱۵
آدرس: www.dotnettips.info
گروه‌ها: WCF, Long Polling

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 _timoutSpan;
        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);
            _timoutSpan = 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 > _timoutSpan)
                        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.

عنوان: Routing Service در WCF
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۱۱/۲۳ ۸:۴۵
آدرس: www.dotnettips.info
گروه‌ها: WCF, Routing Service

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 مسیر یابی 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 فالب 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 سرویس ها:

```
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>
    <services>
      <!-- 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 Endpointهای 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.

عنوان: **فیلترها در WCF Routing Service**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۱۱/۲۳ ۱۲:۵
آدرس: www.dotnettips.info
گروه‌ها: WCF, routing

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 ادرس 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 هرکدام از مقصدها یا Endpointها 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 Prioirty 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.

عنوان: **Content Negotiation در WCF**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۳/۰۳/۰۶ ۹:۴۵
آدرس: www.dotnettips.info
گروه‌ها: WCF, Serialization, Content Negotiation

Content Negotiation ، مکانیزمی است که طی آن مصرف کننده یک سرویس http تعیین می‌کند که خروجی مورد نظر از سرویس به چه فرمتی در اختیار آن قرار گیرد. این قابلیت بسیار زیبا در Asp.Net Web Api فراهم می‌باشد. اما از آن جا که در WCF به صورت توکار مکانیزمی جهت پیاده سازی این قابلیت در نظر گرفته نشده است می‌توان از طریق یک کتابخانه ثالث به نام WCFRestContrib به این مهم دست یافت.

به صورت معمول برای پیاده سازی Content Negotiation، مصرف کننده باید در Accept هدر درخواست، برای سرویس مورد نظر، نوع Content-Type را نیز تعیین نمایید. از طرفی سرویس دهنده نیز باید معادل Mime Type درخواست شده، یک Formatter جهت سریالایز داده‌ها در اختیار داشته باشد. در WCF از طریق کتابخانه WcfRestContrib می‌توانیم به صورت زیر Content Negotiation را پیاده سازی نماییم:

ابتدا از طریق Nuget کتابخانه زیر را نصب کنید:

```
install-package WcfRestContrib
```

حال فرض کنید سرویسی به صورت زیر داریم:

```
[ServiceContract]
public interface IBooksService
{
    [OperationContract]
    void AddBook(string isbn, Book book);
}
```

کدهای بالا روشی مرسوم برای تعریف Service Contractهای WCF است. برای اینکه سرویس WCF بالا به صورت Rest طراحی شود و از طرفی قابلیت سریالاز داده‌ها به چندین فرمت را داشته باشد باید به صورت زیر عمل نماییم:

```
[ServiceContract]
public interface IBooksService
{
    [WebInvoke(UriTemplate = "/{isbn}", Method=Verbs.Put)]
    [WebDispatchFormatter]
    [OperationContract]
    void AddBook(string isbn, Book book);
    ....
}
```

وظیفه WebDispatchFormatterAttribute تعریف شده برای Operation بالا این است که نوع فرمت مورد نیاز را از Accept هدر درخواست واکشی کرده و با توجه به MimeTypeهای تعریف شده در سرویس، داده‌ها را به آن فرمت سریالاز نماید. در صورتی که MimeType درخواست شده از سوی مصرف کننده، سمت سرور تعریف نشده بود، MimeType پیش فرض انتخاب می‌شود.
گام بعدی مشخص کردن انواع MimeTypeها برای این سرویس است. در WcfRestContrib به صورت پیش فرض چهار Formatter تعبیه شده است:

» Xml : از DataContractSerializer موجود در WCF برای سریالاز و دی سریالایز داده‌ها استفاده می‌کند.
» Json : از طریق DataContactJsonSerializer برای سریالاز و دی سریالایز داده‌ها استفاده می‌کند.
POX : همانند مورد اول از DataContractSerializer استفاده می‌کند با این تفاوت که DataContractها بدون Namesapce و Attribute و DataMemberها نیز بدون Order می‌باشند.
» Form Url Encoded

در صورتی که نیاز به formatter دیگری دارید می‌توانید با استفاده از CustomFormatter موجود در این کتابخانه، Formatter

دلخواه خود را پیاده سازی نمایید.

همان طور که در بالا ذکر شد، در صورتی که MimeType درخواست شده از سوی مصرف کننده، سمت سرور تعریف نشده باشد، MimeType پیش فرض انتخاب می‌شود. برای تعریف MimeType پیش فرض می‌توان از خاصیت WebDispatchFormatterConfigurationAttribute که در فضای نام WcfRestContrib.ServiceModel.Description قرار دارد استفاده کرد. تعاریف سایر MimeTypeها نیز با استفاده از WebDispatchFormatterMimeTypeAttribute انجام می‌شود. به صورت زیر:

```
[WebDispatchFormatterConfiguration("application/xml")]
[WebDispatchFormatterMimeType(typeof(WcfRestContrib.ServiceModel.Dispatcher.Formatters.PoDataContract),
"application/xml",  "text/xml")]
[WebDispatchFormatterMimeType(
typeof(WcfRestContrib.ServiceModel.Dispatcher.Formatters.DataContractJson),  "application/json")]
[WebDispatchFormatterMimeType(
typeof(WcfRestContrib.ServiceModel.Dispatcher.Formatters.FormUrlEncoded), "application/x-www-form-
urlencoded")]
public class Books : IBooksService
{
   public void AddBook(string isbn, Book book)
   {
   }
}
```

همانند سایر تنظیمات WCF می‌توان تمامی این موارد را در فایل Config پروژه سرویس نیز تعریف کرد: برای مثال:

```
<system.serviceModel>
    <extensions>
        <behaviorExtensions>
            <add name="webFormatter"

type="WcfRestContrib.ServiceModel.Configuration.WebDispatchFormatter.ConfigurationBehaviorElement,
WcfRestContrib,
                        Version=x.x.x.x, Culture=neutral, PublicKeyToken=89183999a8dc93b5"/>
        </behaviorExtensions>
    </extensions>
    <serviceBehaviors>
        <behavior name="Rest">
          <webFormatter>
            <formatters defaultMimeType="application/xml">
              <formatter mimeTypes="application/xml,text/xml"
                         type="WcfRestContrib.ServiceModel.Dispatcher.Formatters.PoxDataContract,
WcfRestContrib"/>
              <formatter mimeTypes="application/json"
                         type="WcfRestContrib.ServiceModel.Dispatcher.Formatters.DataContractJson,
WcfRestContrib"/>
              <formatter mimeTypes="application/x-www-form-urlencoded"
                         type="WcfRestContrib.ServiceModel.Dispatcher.Formatters.FormUrlEncoded,
WcfRestContrib"/>
            </formatters>
          </webFormatter>
        </behavior>
    </serviceBehaviors>
</system.serviceModel>
```

نکته:

در صورتی که قصد داشته باشیم که باتوجه به direction مورد نظر (نظیر Outgoing یا Incoming) داده‌ها سریالایز/ دی سریالایز شوند، می‌توان این مورد را در هنگام تعریف OperationContract تعیین کرد:

```
[WebDispatchFormatter(WebDispatchFormatter.FormatterDirection.Outgoing)]
```

مطلب تکمیلی:

مشاهده پیاده سازی Content Negotiation در Asp.Net MVC

عنوان: ایجاد سرویس چندلایه‌ی WCF با Entity Framework در قالب پروژه - 10
نویسنده: حامد قنادی
تاریخ: ۱۳۹۳/۰۴/۲۷ ۲۲:۲۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, WCF

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 این‌کار هنگامی‌که 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 آن‌را 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 هنگامی‌که قصد به‌هنگام‌کردن 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 این‌که شناسه‌ی جدول آن برابر با 1 باشد، مقدار 2 خواهد گرفت. ولی بقیه‌ی ستون‌های آن به علت این‌که 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 این‌که ستون‌هایی که مقدار ندارند، در جدول اصلی هم مقدار null بگیرند برای ما مهم است یا نه. برای نمونه هنگامی‌که می‌خواهیم سطری به جدول بیفزاییم یا این‌که 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 خواهم نوشت.

نظرات خوانندگان

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

به نظرتون این جوری اصل SRP رو نقض نکردیم؟

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

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.

عنوان: **کار با وب سرویس جاوایی تشخیص ایمیل‌های موقتی در دات نت**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۰/۲۷ ۱۲:۴۰
آدرس: www.dotnettips.info
گروه‌ها: ASP.Net, MVC, Security, WCF, xml, Validation, Java

یکی از وب سرویس‌های سایت name api ، امکان تشخیص موقتی بودن ایمیل مورد استفاده‌ی جهت ثبت نام در یک سایت را فراهم می‌کند. آدرس WSDL آن نیز در اینجا قرار دارد. اگر مطابق معمول استفاده از سرویس‌های وب در دات نت، بر روی ارجاعات پروژه کلیک راست کرده و گزینه‌ی Add service refrence را انتخاب کنیم و سپس آدرس WSDL یاد شده را به آن معرفی کنیم، بدون مشکل ساختار این وب سرویس دریافت و برای استفاده‌ی از آن به یک چنین کدی خواهیم رسید:

```
var client = new SoapDisposableEmailAddressDetectorClient();

var context = new soapContext
{
    //todo: get your API key here: http://www.nameapi.org/en/register/
    apiKey = "test"
};
var result = client.isDisposable(context, "DaDiDoo@mailinator.com");

if (result.disposable.ToString() == "YES")
{
    Console.WriteLine("YES! It's Disposable!");
}
```

متد isDisposable ارائه شده‌ی توسط این وب سرویس، دو پارامتر context که در آن باید API Key خود را مشخص کرد و همچنین آدرس ایمیل مورد بررسی را دریافت می‌کند. اگر به همین ترتیب این پروژه را اجرا کنید، با خطای Bad request از طرف سرور متوقف خواهید شد:

```
Additional information: The remote server returned an unexpected response: (400) Bad Request.
```

اگر به خروجی این وب سرویس در فیدلر مراجعه کنیم، چنین شکلی را خواهد داشت:

```
<html><head><title>Bad Request</title></head><body><h1>Bad Request</h1><p>No api-key
provided!</p></body></html>
```

عنوان کرده‌است که api-key را، در درخواست وب خود ذکر نکرده‌ایم.

اگر همین وب سرویس را توسط امکانات سایت http://wsdlbrowser.com بررسی کنید، بدون مشکل کار می‌کند. اما تفاوت در کجاست؟

خروجی ارسالی به سرور، توسط سایت http://wsdlbrowser.com به این شکل است:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns1="http://disposableemailaddressdetector.email.services.v4_0.soap.server.nameapi.org/">
  <SOAP-ENV:Body>
    <ns1:isDisposable>
      <context>
        <apiKey>test</apiKey>
      </context>
      <emailAddress>sdsdg@site.com</emailAddress>
    </ns1:isDisposable>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

و نمونه‌ی تولید شده‌ی توسط WCF (امکان Add service reference در حقیقت یک WCF Client را ایجاد می‌کند) به صورت زیر می‌باشد:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Body xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
     <isDisposable
```

```
xmlns="http://disposableemailaddressdetector.email.services.v4_0.soap.server.nameapi.org/">
          <context xmlns=""><apiKey>test</apiKey></context>
          <emailAddress xmlns="">DaDiDoo@mailinator.com</emailAddress>
      </isDisposable>
  </s:Body>
</s:Envelope>
```

از لحاظ اصول XML، خروجی تولیدی توسط WCF هیچ ایرادی ندارد. از این جهت که نام فضای نام مرتبط با http://schemas.xmlsoap.org/soap/envelope/ را به s تنظیم کرده‌است و سپس با استفاده از این نام، Envelope را تشکیل داده‌است. اما ... این وب سرور جاوایی دقیقا با نام SOAP-ENV کار می‌کند و فضای نام ns1 بعدی آن. کاری هم به اصول XML ندارد که باید بر اساس نام xmlns ذکر شده، کار Parse ورودی دریافتی صورت گیرد و نه بر اساس یک رشته‌ی ثابت از پیش تعیین شده. بنابراین باید راهی را پیدا کنیم تا بتوان این s را تبدیل به SOAP-ENV کرد.

برای این منظور به سه کلاس ذیل خواهیم رسید:

```
public class EndpointBehavior : IEndpointBehavior
{
    public void AddBindingParameters(ServiceEndpoint endpoint, BindingParameterCollection
bindingParameters)
    { }

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

    public void Validate(ServiceEndpoint endpoint)
    { }

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

public class ClientMessageInspector : IClientMessageInspector
{
    public void AfterReceiveReply(ref Message reply, object correlationState)
    { }

    public object BeforeSendRequest(ref Message request, System.ServiceModel.IClientChannel channel)
    {
        request = new MyCustomMessage(request);
        return request;
    }
}

/// <summary>
/// To customize WCF envelope and namespace prefix
/// </summary>
public class MyCustomMessage : Message
{
    private readonly Message _message;

    public MyCustomMessage(Message message)
    {
        _message = message;
    }

    public override MessageHeaders Headers
    {
        get { return _message.Headers; }
    }

    public override MessageProperties Properties
    {
        get { return _message.Properties; }
    }

    public override MessageVersion Version
    {
        get { return _message.Version; }
    }

    protected override void OnWriteStartBody(XmlDictionaryWriter writer)
    {
```

```
        writer.WriteStartElement("Body", "http://schemas.xmlsoap.org/soap/envelope/");
    }

    protected override void OnWriteBodyContents(XmlDictionaryWriter writer)
    {
        _message.WriteBodyContents(writer);
    }

    protected override void OnWriteStartEnvelope(XmlDictionaryWriter writer)
    {
        writer.WriteStartElement("SOAP-ENV", "Envelope", "http://schemas.xmlsoap.org/soap/envelope/");
        writer.WriteAttributeString("xmlns", "ns1", null, value:
"http://disposableemailaddressdetector.email.services.v4_0.soap.server.nameapi.org/");
    }
}
```

که پس از تعریف client به نحو ذیل معرفی می‌شوند:

```
var client = new SoapDisposableEmailAddressDetectorClient();
client.Endpoint.Behaviors.Add(new EndpointBehavior());
```

توسط EndpointBehavior سفارشی، می‌توان به متد **Envelope** OnWriteStart دسترسی یافت و سپس s آن‌را با SOAP-ENV درخواستی این وب سرویس جایگزین کرد. اکنون اگر برنامه را اجرا کنید، بدون مشکل کار خواهد کرد و دیگر پیام یافت نشدن API-Key را صادر نمی‌کند.

کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید.

عنوان: **اتصال به سرویس WCF در NETCF 3.5**
نویسنده: مهدی پایروند
تاریخ: ۱۳۹۳/۱۱/۰۴ ۱۸:۱۰
آدرس: www.dotnettips.info
گروه‌ها: WCF, Windows Compact, .NET Compact Framework

با توجه به پیشرفتی که در حوزه اپلیکشن‌های وابسته به فریمورک دات نت بوجود آمده، ولی شاید حرکت عملی بزرگی از سمت تولیدکنندگان در حوزه کامپکت صورت نگرفته و همچنان شاهد فرمانروایی سیستم عاملهایی چون Windows Compact 6.0 با استفاده از دات نت فریمورک‌هایی نهایت با نسخه 3.5 هستیم. البته میتوان ارزان‌تر بودن در خارج و مسئله تحریم در داخل را هم در نظر داشت و نمونه عینی این مورد را میتوان در دستگاههای وارد شده در حوزه Compact ، دید. البته شرکت‌های تولید کننده خارجی که عمدتا در کشورهای جنوب شرق و شرق آسیا هستند، جزو شرکت‌های مطرح در این زمینه هستند که بازارهای خوبی هم در کشورهای توسعه یافته‌ای چون آمریکا پیدا کرده‌اند.

در این بین برای عقب نماندن از تکنولوژی‌های جدید بوجود آمده در حوزه دات نت مانند WCF این مقاله کمکی هر چند کوچک برای استفاده از این قابلیت موثر در فریمورک کامپکت می‌تواند باشد.
پیشنیازهای لازم:

- Microsoft Visual Studio 2008 + Service Pack 1

- نصب Power Toys for .NET Compact Framework 3.5

پیاده سازی سرویس (بر روی سیستمی غیر از ویندوز کامپکت):

در ویژوال استودیو 2008 سرویس پک یک، پروژه ای از نوع class library را ایجاد کرده و سرویسی تستی را برای استفاده ایجاد میکنیم:

```
[ServiceContract(Namespace = "http://samples.wcf.cfnet.sample")]
    public interface ICalculator
    {
        [OperationContract]
        int Add(int a, int b);
    }
```

و پیاده سازی آن:

```
public class CalculatorService : ICalculator
    {
        public static int count;

        public int Add(int a, int b)
        {
            count++;
            Console.WriteLine(string.Format("{3}\tReceived 'Add({0}, {1})' returning {2}", a, b, a + b,
count));
            return a + b;
        }
```

سرور سرویس:

برای هاست این سرویس از یک برنامه‌ی کنسول که در سلوشن ایجاد میکنیم استفاده میکنیم. البته امکان‌های دیگر برای هاست سرویس در هر پروسس دات نتی را میتوان یاد آور شد. برای هاست کردن شروع یک سرویس WCF باید یک IP درون شبکه را که قابل دسترسی از سمت ویندوز کامپکت بوده و به سیستم انتساب داده شده، دریافت و استفاده کنیم:

```
var addressList = Dns.GetHostEntry(Dns.GetHostName());

string hostIP = addressList.AddressList.Single(x=>x.ToString().StartsWith("192.168.10.")).ToString();
```

```
Uri address = new Uri(string.Format("http://{0}:8000/Calculator", hostIP));
```

در قطعه بالا IP در رنج مناسب و قابل دسترسی انتخاب میشود چون ویندوز کامپکت (فارق از اینکه در شبیه ساز باشد یا واقعی) از طریق شبکه به سرور دسترسی پیدا میکند باید IP مناسب انتساب داده شده انتخاب شود.

```
ServiceHost serviceHost = new ServiceHost(typeof(CalculatorService),address);
serviceHost.AddServiceEndpoint(typeof(ICalculator), new BasicHttpBinding(), "Calculator");
```

در ادامه یک سرویس هاست را new کرده و سرویس و بایندینگ را به آن در سازنده پاس میدهیم.

```
var serviceMetadataBehavior =
new ServiceMetadataBehavior { HttpGetEnabled = true };
serviceHost.Description.Behaviors.Add(serviceMetadataBehavior);
```

این قسمت برای ادامه کارکرد سرویس لازم نیست ولی در ادامه‌ی مقاله برای تولید کدهای سمت کلاینت باید این قابلیت فعال باشد و پس از آن دیگر احتیاجی نیست و میتوان این چند خط کد را کامنت کرد.

```
serviceHost.Open();
Console.WriteLine("CalculatorService is running at " + address.ToString());
Console.WriteLine("Press <ENTER> to terminate");
Console.ReadLine();
serviceHost.Close();
```

و در نهایت، شروع سرویس با فرمان Open و خاتمه آن با فرمان Close .

کلاینت سرویس (در داخل ویندوز کامپکت):

همراه با ارائه دات نت فریمورک 3.5 برای کار با سرویس WCF که از آن یک نسخه‌ی ارائه شده برای کامپکت نیز تهیه شده‌است، ابزاری مانند netcfSvcUtil.exe که در SDK نسخه‌ی کامپکت موجود است و کاربرد هندل کردن بعضی از موارد مانند تولید کد پروکسی‌های سمت کلاینت را دارد که در ادامه طرز استفاده از آن را بررسی خواهیم کرد. بعد از اجرای سرویس WCF با رفتار HttpGetEnabled = true برای بررسی سریع کارکرد صحیح سرویس، آدرس آن را در مرورگر میبینیم. تصویر زیر نتیجه‌ی آن در مرورگر است:

در خط فرمان به آدرس مربوط به این ابزار رفته (بسته به نسخه‌ی سیستم عامل ممکن است در پوشه‌های زیر یافت شود) :

```
(Windows Drive)\Program Files (x86)\Microsoft.NET\SDK\CompactFramework\v3.5\bin
(Windows Drive)\Program Files\Microsoft.NET\SDK\CompactFramework\v3.5\bin
```

و فرمان زیر را اجرا میکنیم:

```
netcfSvcUtil.exe /language:C# /target:code /directory:D:\GeneratedCode\CF\CaculatorService
http://192.168.10.189:8000/BooksService.svc?wsdl
```

البته ذکر IP شبکه در اینجا الزامی نیست؛ زیرا در صورت استفاده از آدرسهای داخلی سیستم، این فرمان به مشکلی بر نخواهد خورد. در این فرمان تولید کد با زبان c# و تولید کد که بصورت پیش فرض نیز وجود دارد و محل ذخیره سازی کدهای تولیدی را مشخص میکنیم و بعد از اجرای این فرمان، باید دو فایل در مسیر اشاره شده در فرمان تولید شود که اساس کار ما در سمت کلاینت خواهد بود:

```
Developer Command Prompt for VS2013
C:\Program Files (x86)\Microsoft Visual Studio 12.0>netcfSvcUtil.exe /language:C
# /target:code /enableDataBinding /directory:D:\GeneratedCode\CF\CaculatorServic
e http://192.168.10.189:8000/Calculator?wsdl
Microsoft (R) .NET Compact Framework Service Model Metadata Tool
[Microsoft (R) Windows (R) Communication Foundation, Version 3.5.0.0]
Copyright (c) Microsoft Corporation.  All rights reserved.

Attempting to download metadata from 'http://192.168.10.189:8000/Calculator?wsdl
' using WS-Metadata Exchange or DISCO.
Generating files...
D:\GeneratedCode\CF\CaculatorService\CalculatorService.cs
D:\GeneratedCode\CF\CaculatorService\CFClientBase.cs

C:\Program Files (x86)\Microsoft Visual Studio 12.0>
```

کلاینت سرویس نیز با استفاده کدهای تولیدی بصورت زیر آماده سازی و اجرا میشود:

```
var addressList = Dns.GetHostEntry(Dns.GetHostName());
var localAddress = addressList.AddressList.Single(x =>
x.ToString().StartsWith("192.168.10.")).ToString();
```

دوباره IP مناسب در شبکه جاری استخراج میشود. بایندیگ مورد نیاز برای ارتباط با سرور ساخته میشود:

```
var binding = CalculatorClient.CreateDefaultBinding();
```

نکته‌ای که دراین قسمت باید مدنظر قرار گیرد این است که در زمان تولید کدها اگر از localhost یا 127.0.0.1 و یا آدرسهای دیگر انتساب داده شده به سرور استفاده کرده باشید در متد CreateDefaultBinding از همان آدرس استفاده میشود و برای اصلاح آن بصورت زیر عمل میکنیم:

```
string remoteAddress = CalculatorClient.EndpointAddress.Uri.ToString();
remoteAddress = remoteAddress.Replace("localhost", serviceAddress.Text);
```

یک EndpointAddress با استفاده از این آدرس ساخته و به‌همراه بایندینگ، یک آبجکت از جنس CalculatorClient که در کدهای تولیدی داریم میسازیم:

```
CalculatorClient _client = new CalculatorClient(binding, endpoint);
```

برای تست نیز تنها متد این سرویس را با یک جفت عدد، صدا میزنیم:

```
var result = _client.Add(82, 18).ToString(CultureInfo.InvariantCulture);
```

به این ترتیب خروجی مورد نظر زیر را در کنسول سرویس مشاهده خواهیم کرد:

```
file:///E:/WINCE/Projects/WCFSamples/WCFSamples.Server/WCFSamples.Serve...
CalculatorService is running at http://192.168.10.189:8000/Calculator
Press <ENTER> to terminate
1       Received 'Add(82, 18)' returning 100
```

عنوان: **کار با اسکنر در برنامه های تحت وب (قسمت اول)**
نویسنده: محمد رعیت پیشه
تاریخ: ۱۳۹۳/۱۱/۱۷ ۰:۱۵
آدرس: www.dotnettips.info
گروه‌ها: ASP.Net, MVC, WCF, TWAIN, Cross-Origin Request

در اکثر برنامه‌های سازمانی، مثل برنامه‌های مدیریت آرشیو اسناد، همواره این نیاز جزو خواسته‌های کاربران بوده که بتوانند به صورت مستقیم و از طریق تنها یک کلیک، تصویر مورد نظر را اسکن کرده و به صورت خودکار در برنامه وارد کنند؛ یعنی بدون اینکه نیاز باشد با استفاده از یک برنامه دیگر ابتدا تصویر را اسکن کرده و سپس در فرم وب، فایل اسکن شده را Browse کنند.
این نیاز اساسا به معنی دسترسی به سخت افزار کاربر از طریق مرورگر می‌باشد که به دلایل متعددی امکان پذیر نیست! مشکلات امنیتی ایجاد شده در این راه بسیار جدی است. اما خوشبختانه راههایی برای رسیدن به این هدف وجود دارند:

1- **ActiveX** : که به صورت native فقط در IE پشتیبانی می‌شود (در نسخه‌های جدیدتر IE نیاز به بروز رسانی پلاگین ActiveX controls می‌باشد) و برای استفاده‌ی از آن در مرورگرهای Firefox و Chrome هم باید از پلاگین‌های جانبی روی هر مرورگر استفاده کرد که مثلا برای اجرای بر روی Firefox، باید افزونه‌ی IE Tab را نصب کرد که تنها کارش این است که سایت را از طریق موتور IE در پنجره‌ی فایرفاکس اجرا کند! که البته این مورد مثل این می‌ماند که سایت در IE باز شده باشد که ممکن است زیاد خوشایند نباشد؛ در غیر اینصورت باید پروژه را از اول بر مبنای اجرای بر روی IE طراحی و پیاده سازی کرد. در ضمن از مشکلات اجرای پلاگین نوشته شده توسط برنامه نویس در IE و نسخه‌های مختلف آن هم چشم پوشی می‌کنیم. یکی از مزیت‌های این روش نسبت به بقیه این هست که دانلود و نصب پلاگین مورد نیاز به صورت خودکار و توسط مرورگر انجام می‌شود.

2- **استفاده از یک کامپوننت جانبی** : مثل کامپوننت‌های LEADTOOLS که ابزارهای متنوع و SDKهای بسیار قدرتمندی را برای اینکار و کارهای دیگر، مانند کار با اسکن تصاویر مغزی، پردازش تصویر، بارکد خوان‌ها، مدیریت اسناد و ... فراهم کرده است. قیمت این ابزار بسیار زیاد است و در برخی موارد امکانات فراهم آورده بسیار بیشتر است از خواسته‌ی ما. این ابزار، هم در HTML & Javascript و هم در DotNet قابل استفاده است و مستندات نسبتا خوبی هم در این زمینه ارائه کرده است. همچنین کامپوننت Dynamsoft که باز هم غیر رایگان هست و قیمت بالایی نیز دارد.

اگر روال کار کامپوننت‌های بالا را مورد بررسی قرار دهید (از طریق اجرای Demo ها، اینجا و اینجا) متوجه خواهید شد که هر دو نیاز به نصب یک سرویس یا App سمت کلاینت جهت اجرای دستورات خود دارند. پس می‌شود اینطور نتیجه گرفت که انجام اینکار بدون اینکه چیزی سمت کاربر نصب شود، ممکن نیست و در هر دو، لینک نصب فایل exe سرویس برای دانلود قرار داده شده است. بر این اساس به راه حل سومی خواهیم رسید که خودمان این سرویس را جهت تعامل با اسکنر سمت کاربر طراحی و پیاده سازی نماییم.

اما چطور ممکن است که با اجرای یک فایل exe سمت کاربر (با این فرض که کاربر در یک دامین مطمئن قرار دارد و می‌شود درخواست نصب سرویس را نمود) این امکان را برای کاربر فراهم نمود که با یک کلیک در مرورگر، اسکنر به صورت خودکار اسکن را آغاز کرده و سپس تصویر حاصل را به یکی از کنترلرهای ما در سمت سرور ارسال نماید؟

برای اینکار ما با دو صورت مساله مواجه هستیم؛ اول اینکه چطور تصویر را سمت کاربر اسکن کنیم و دوم اینکه چطور این تصویر را به سرور ارسال نماییم!
برای مساله‌ی اول از کتابخانه (Windows Image Acquisition (WIA استفاده خواهیم نمود که این کتابخانه به ما این امکان را میدهد تا با سخت افزارهایی که از TWAIN پشتیبانی می‌کنند، بتوانیم ارتباط برقرار نماییم.

برای مساله‌ی دوم هم نیاز به پیاده سازی یک WCF Service و اجرای آن (هاست کردن) در سمت کلاینت داریم. در واقع با صدا زدن متدهای این سرویس، از کتابخانه‌ی بالا استفاده کرده و اسکن را انجام می‌دهیم.

ادامه دارد...

عنوان: **کار با اسکنر در برنامه های تحت وب (قسمت دوم و آخر)**
نویسنده: محمد رعیت پیشه
تاریخ: ۱۳۹۳/۱۱/۲۰ ۲۳:۳۵
آدرس: www.dotnettips.info
گروه‌ها: ASP.Net, MVC, WCF, TWAIN, Cross-Origin Request, Scanning

در قسمت قبل « کار با اسکنر در برنامه‌های تحت وب (قسمت اول) » دیدی از کاری که قرار است انجام دهیم، رسیدیم. حالا سراغ یک پروژه‌ی عملی و پیاده سازی مطالب مطرح شده می‌رویم.

ابتدا پروژه‌ی WCF را شروع می‌کنیم. ویژوال استودیو را باز کرده و از قسمت New Project > Visual C# > WCF یک پروژه‌ی WCF Service Application جدید را مثلا با نام "WcfServiceScanner" ایجاد نمایید. پس از ایجاد، دو فایل IService1.cs و Service1.scv موجود را به IScannerService و ScannerService تغییر نام دهید. سپس ابتدا محتویات کلاس اینترفیس IScannerService را به صورت زیر تعریف نمایید :

```
[ServiceContract]
    public interface IScannerService
    {
        [OperationContract]
        [WebInvoke(Method = "GET",
           BodyStyle = WebMessageBodyStyle.Wrapped,
           RequestFormat = WebMessageFormat.Json,
           ResponseFormat = WebMessageFormat.Json,
           UriTemplate = "GetScan")]
        string GetScan();
    }
```

در اینجا ما فقط اعلان متدهای مورد نیاز خود را ایجاد کرده‌ایم. علت استفاده از Attribute ایی با نام WebInvoke ، مشخص نمودن نوع خروجی به صورت Json است و همچنین عنوان آدرس مناسبی برای صدا زدن متد. پس از آن کلاس ScannerService را مطابق کدهای زیر تغییر دهید:

```
public class ScannerService : IScannerService
    {
        public string GetScan()
        {
            // TODO Add code here
        }
    }
```

تا اینجا فقط یک WCF Service معمولی ساخته‌ایم .در ادامه به سراغ کلاس WIA برای ارتباط با اسکنر می‌رویم.
بر روی پروژه‌ی خود راست کلیک کرده و Add Reference را انتخاب نموده و سپس در قسمت COM، گزینه‌ی Microsoft Windows Image Acquisition Library v2.0 را به پروژه‌ی خود اضافه نمایید.
با اضافه شدن این ارجاع به پروژه، دسترسی به فضای نام WIA برای ما امکان پذیر می‌شود که ارجاعی از آن را در کلاس ScannerService قرار می‌دهیم.

```
using WIA;
```

اکنون متد GetScan را مطابق زیر اصلاح می‌نماییم:

```
public string GetScan()
        {
            var imgResult = String.Empty;
            var dialog = new CommonDialogClass();
            try
            {
                // نمایش فرم پیشفرض اسکنر
                var image = dialog.ShowAcquireImage(WiaDeviceType.ScannerDeviceType);

                // ذخیره تصویر در یک فایل موقت
                var filename = Path.GetTempFileName();
```

```
            image.SaveFile(filename);
            var img = Image.FromFile(filename);

            // img تبدیل تصویر به Base64 جهت ارسال سمت کاربر و نمایش در تگ
            imgResult = ImageHelper.ImageToBase64(img, ImageFormat.Jpeg);
        }
        catch
        {
            // از آنجاییه که امکان نمایش خطا وجود ندارد در صورت بروز خطا رشته خالی
            // بازگردانده می‌شود که به معنای نبود تصویر می‌باشد
        }

        return imgResult;
    }
```

دقت داشته باشید که کدها را در زمان توسعه بین Try..Catch قرار ندهید چون ممکن‌است در این زمان به خطاهایی برخورد کنید که نیاز باشد در مرورگر آنها را دیده و رفع خطا نمایید.

CommonDialogClass کلاس اصلی در اینجا جهت نمایش فرم کار با اسکنر می‌باشد و متدهای مختلفی را جهت ارتباط با اسکنر در اختیار ما قرار می‌دهد که بسته به نیاز خود می‌توانید از آنها استفاده کنید. برای نمونه در مثال ما نیز متد اصلی که مورد استفاده قرار گرفته، ShowAcquireImage می‌باشد که این متد، فرم پیش فرض دریافت اسکنر را به کاربر نمایش می‌دهد و کاربر از طریق آن می‌تواند قبل از شروع اسکن، یکسری تنظیمات را انجام دهد.

این متد ابتدا به صورت خودکار فرم تعیین دستگاه اسکنر ورودی را نمایش داده :

و سپس فرم پیش فرض اسکنرهای TWAIN را جهت تعیین تنظیمات اسکن نمایش می‌دهد که این امکان نیز در این فرم فراهم است تا دستگاه‌های Feeder یا Flated انتخاب گردند.

خروجی این متد همان عکس اسکن شده است که از نوع WIA.ImageFile می‌باشد و ما پس از دریافتش، ابتدا آن را در یک فایل موقت ذخیره نموده و سپس با استفاده از یک متد کمکی آن را به فرمت Base64 برای درخواست کننده اسکن ارسال می‌نماییم.

کدهای کلاس کمکی ImageHelper:

```
public static string ImageToBase64(Image image, System.Drawing.Imaging.ImageFormat format)
        {
            if (image != null)
            {
                using (MemoryStream ms = new MemoryStream())
                {
                    // Convert Image to byte[]
                    image.Save(ms, format);
                    byte[] imageBytes = ms.ToArray();

                    // Convert byte[] to Base64 String
                    string base64String = Convert.ToBase64String(imageBytes);
                    return base64String;
                }
            }
            return String.Empty;
        }
```

توجه داشته باشید که خروجی این متد قرار است توسط callBack یک متد جاوا اسکریپتی مورد استفاده قرار گرفته و احیانا عکس مورد نظر در صفحه نمایش داده شود. پس بهتر است که از قالب تصویر به شکل Base64 استفاده گردد. ضمن اینکه پلاگین‌های Jquery مرتبط با ویرایش تصویر هم از این قالب پشتیبانی می‌کنند. (اینجا)
این مثال به ساده‌ترین شکل نوشته شد. کلاس دیگری هم در اینجا وجود دارد و در صورتیکه از اسکنر نوع Feeder استفاده می‌کنید، می‌توانید از کدهای آن استفاده کنید.

کار ما تا اینجا در پروژه‌ی WCF Service تقریبا تمام است. اگر پروژه را یکبار Build نمایید برای اولین بار احتمالا پیغام خطاهای زیر ظاهر خواهند شد:

جهت رفع این خطا، در قسمت Referenceهای پروژه خود، WIA را انتخاب نموده و از Propertiesهای آن خصوصیت Embed Interop Types را به False تغییر دهید؛ مشکل حل می‌شود.

به سراغ پروژه‌ی ویندوز فرم جهت هاست کردن این WCF سرویس می‌رویم. می‌توانید این سرویس را بر روی یک Console App یا Windows Service هم هاست کنید که در اینجا برای سادگی مثال، از WinForm استفاده می‌کنیم.
یک پروژه‌ی WinForm جدید را ایجاد کنید و سپس از قسمت Solution > Add Reference به مسیر پروژه‌ی قبلی رفته و dllهای آن را به پروژه خود اضافه نمایید.
Form1.cs را باز کرده و ابتدا دو متغیر زیر را در آن به صورت عمومی تعریف نمایید:

```
private readonly Uri _baseAddress = new Uri("http://localhost:6019");
        private ServiceHost _host;
```

برای استفاده از کلاس ServiceHost لازم است تا ارجاعی به فضای نام System.ServiceModel داده شود. متغیر _baseAddress نگه دارنده‌ی آدرس ثابت سرویس اسکنر در سمت کلاینت می‌باشد و به این ترتیب ما دقیقا می‌دانیم باید سرویس را با کدام آدرس در کدهای جاوا اسکریپتی خود فراخوانی نماییم.
حال در رویداد Form_Load برنامه، کدهای زیر را جهت هاست کردن سرویس اضافه می‌نماییم:

```
private void Form1_Load(object sender, EventArgs e)
        {
            _host = new ServiceHost(typeof(WcfServiceScanner.ScannerService), _baseAddress);
            _host.Open();
        }

        private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
            _host.Close();
        }
```

همین چند خط برای هاست کردن سرویس روی آدرس localhost و پورت 8010 کامپیوتر کلاینت کافی است. اما یکسری تنظیمات مربوط به خود سرویس هم وجود دارد که باید در زمان پیاده سازی سرویس، در خود پروژه‌ی سرویس، ایجاد می‌گردید. اما از آنجا که ما قرار است سرویس را در یک پروژه‌ی دیگر هاست کنیم، بنابراین این تنظیمات را باید در همین پروژه‌ی WinForm قرار دهیم.
فایل App.Config پروژه‌ی WinForm را باز کرده و کدهای آنرا مطابق زیر تغییر دهید:

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

```
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
    </startup>

  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="BehaviourMetaData">
          <serviceMetadata httpGetEnabled="true" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <services>
      <service name="WcfServiceScanner.ScannerService"
               behaviorConfiguration="BehaviourMetaData">
        <endpoint address=""
                  binding="basicHttpBinding"
                  contract="WcfServiceScanner.IScannerService" />
      </service>
    </services>
  </system.serviceModel>

</configuration>
```

اکنون پروژه‌ی هاست آماده اجرا می‌باشد. اگر آنرا اجرا کنید و در مرورگر خود آدرس ذکر شده را وارد کنید، صفحه‌ی زیر را مشاهده خواهد کرد که به معنی صحت اجرای سرویس اسکنر می‌باشد.

ScannerService Service

You have created a service.

To test this service, you will need to create a client and use it to call the service. You can do this using the svcutil.exe tool from the command line with the following syntax:

```
svcutil.exe http://localhost:6019/?wsdl
```

You can also access the service description as a single file:

```
http://localhost:6019/?singleWsdl
```

This will generate a configuration file and a code file that contains the client class. Add the two files to your client application and use the generated client class to call the Service. For example:

C#

```
class Test
{
    static void Main()
    {
        ScannerServiceClient client = new ScannerServiceClient();

        // Use the 'client' variable to call operations on the service.

        // Always close the client.
        client.Close();
    }
}
```

اگر موفق به اجرا نشدید و احیانا با خطای زیر مواجه شدید، اطمینان حاصل کنید که ویژوال استودیو Run as Administrator باشد. مشکل حل خواهد شد.

به سراغ پروژه‌ی بعدی، یعنی وب سایت خود می‌رویم. یک پروژه‌ی MVC جدید ایجاد نمایید و در View مورد نظر خود، کدهای زیر را جهت صدا زدن متد GetScan اضافه می‌کنیم.

(از آنجا که کدها به صورت جاوا اسکریپت می‌باشد، پس مهم نیست که حتما پروژه MVC باشد؛ یک صفحه‌ی HTML ساده هم کافی است).

```
<a href="#" id="get-scan">Get Scan</a>
<img src="" id="img-scanned" />
<script>
    $("#get-scan").click(function () {
        var url = 'http://localhost:6019/';
        $.get(url, function (data) {
            $("#img-scanned").attr("src","  "+ data.GetScanResult);
        });
    });
</script>
```

دقت کنید در هنگام دریافت اطلاعات از سرویس، نتیجه به شکل GetScanResult خواهد بود. الان اگر پروژه را اجرا نمایید و روی لینک کلیک کنید، اسکنر شروع به دریافت اسکن خواهد کرد اما نتیجه‌ای بازگشت داده نخواهد شد و علت هم مشکل امنیتی CORS می‌باشد که به دلیل دریافت اطلاعات از یک دامین دیگر رخ می‌دهد و اگر با Firebug درخواست را بررسی کنید متوجه خطا به شکل زیر خواهید شد.

راه حل‌های زیادی برای این مشکل ارائه شده است، و متاسفانه بسیاری از آنها در شرایط پروژه‌ی ما جوابگو نمی‌باشد (به دلیل هاست روی یک پروژه ویندوزی). تنها راه حل مطمئن (تست شده) استفاده از یک کلاس سفارشی در پروژه‌ی WCF Service می‌باشد که مثال آن در اینجا آورده شده است.

برای رفع مشکل به پروژه WcfServiceScanner بازگشته و کلاس جدیدی را به نام CORSSupport ایجاد کرده و کدهای زیر را به آن اضافه کنید:

```
public class CORSSupport : IDispatchMessageInspector
    {
        Dictionary<string, string> requiredHeaders;
        public CORSSupport(Dictionary<string, string> headers)
        {
            requiredHeaders = headers ?? new Dictionary<string, string>();
        }

        public object AfterReceiveRequest(ref System.ServiceModel.Channels.Message request,
System.ServiceModel.IClientChannel channel, System.ServiceModel.InstanceContext instanceContext)
        {
            var httpRequest = request.Properties["httpRequest"] as HttpRequestMessageProperty;
            if (httpRequest.Method.ToLower() == "options")
                instanceContext.Abort();
            return httpRequest;
        }

        public void BeforeSendReply(ref System.ServiceModel.Channels.Message reply, object
correlationState)
        {
            var httpResponse = reply.Properties["httpResponse"] as HttpResponseMessageProperty;
            var httpRequest = correlationState as HttpRequestMessageProperty;

            foreach (var item in requiredHeaders)
            {
                httpResponse.Headers.Add(item.Key, item.Value);
            }
            var origin = httpRequest.Headers["origin"];
            if (origin != null)
                httpResponse.Headers.Add("Access-Control-Allow-Origin", origin);

            var method = httpRequest.Method;
            if (method.ToLower() == "options")
                httpResponse.StatusCode = System.Net.HttpStatusCode.NoContent;

        }
    }

    // Simply apply this attribute to a DataService-derived class to get
    // CORS support in that service
    [AttributeUsage(AttributeTargets.Class)]
    public class CORSSupportBehaviorAttribute : Attribute, IServiceBehavior
    {
        #region IServiceBehavior Members

        void IServiceBehavior.AddBindingParameters(ServiceDescription serviceDescription,
ServiceHostBase serviceHostBase, System.Collections.ObjectModel.Collection<ServiceEndpoint> endpoints,
BindingParameterCollection bindingParameters)
        {
        }

        void IServiceBehavior.ApplyDispatchBehavior(ServiceDescription serviceDescription,
ServiceHostBase serviceHostBase)
        {
            var requiredHeaders = new Dictionary<string, string>();

            //Chrome doesn't accept wildcards when authorization flag is true
```

```
            //requiredHeaders.Add("Access-Control-Allow-Origin", "*");
            requiredHeaders.Add("Access-Control-Request-Method", "POST,GET,PUT,DELETE,OPTIONS");
            requiredHeaders.Add("Access-Control-Allow-Headers", "Accept, Origin, Authorization, X-
Requested-With,Content-Type");
            requiredHeaders.Add("Access-Control-Allow-Credentials", "true");
            foreach (ChannelDispatcher cd in serviceHostBase.ChannelDispatchers)
            {
                foreach (EndpointDispatcher ed in cd.Endpoints)
                {
                    ed.DispatchRuntime.MessageInspectors.Add(new CORSSupport(requiredHeaders));
                }
            }
        }

        void IServiceBehavior.Validate(ServiceDescription serviceDescription, ServiceHostBase
serviceHostBase)
        {
        }

        #endregion
    }
```

فضاهای نام لازم برای این کلاس به شرح زیر می‌باشد:

```
using System.ServiceModel;
using System.ServiceModel.Channels;
using System.ServiceModel.Description;
using System.ServiceModel.Dispatcher;
```

کلاس ScannerService را باز کرده و آنرا به ویژگی

```
[CORSSupportBehavior]
    public class ScannerService : IScannerService
    {
```

مزین نمایید.

کار تمام است، یکبار دیگر ابتدا پروژه‌ی WcfServiecScanner و سپس پروژه هاست را Build کرده و برنامه‌ی هاست را اجرا کنید. اکنون مشاهده می‌کنید که با زدن دکمه‌ی اسکن، اسکنر فرم تنظیمات اسکن را نمایش می‌دهد که پس از زدن دکمه‌ی Scan، پروسه آغاز شده و پس از اتمام، تصویر اسکن شده در صفحه‌ی وب سایت نمایش داده می‌شود.

نظرات خوانندگان

نویسنده: امیر بختیاری
تاریخ: ۱۳۹۳/۱۱/۲۱ ۱:۴۸

با سلام و تشکر خدمت شما دوست گرامی
من تمام مراحل را به دقت ایجاد کردم ولی بازم همون خطایی که خودتون هم عنوان کردید را میده

```
Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at
http://localhost:13748/. This can be fixed by moving the resource to the same domain or enabling CORS.
```

فایل پروژه را به طور کامل میزارم ببینید علت چیه WcfServiceScanner.zip

با تشکر مجدد

نویسنده: شروین ایرانی
تاریخ: ۱۳۹۳/۱۱/۲۱ ۹:۵

با سلام؛ آیا امکانش هست که سورس پروژه‌ی مثال رو هم قرار بدید؟ تشکر

نویسنده: امیر بختیاری
تاریخ: ۱۳۹۳/۱۱/۲۱ ۱۰:۳۱

سورس پروژه را از نظر قبلی می‌تونید دریافت کنید

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۱/۲۲ ۱۶:۱۵

این مثال را بررسی کنید: SelfHostWcfScannerService.zip
- ابتدا سرویس self host آن‌را با دسترسی ادمین اجرا کنید.
- بعد برنامه‌ی وب را اجرا کنید.

نویسنده: امیر بختیاری
تاریخ: ۱۳۹۳/۱۱/۲۲ ۱۷:۳۰

با تشکر
این مثال به قسمت Catch رفته و عکس پیشفرض رو نشان می‌دهد
درحالی که بر روی کامپیوتر من اسکنر نصب است و با دموی DynamSoft هم به خوبی جواب گرفتم و مشکلی نداشتم
به نظرتون چرا پنجره اسکنر رو باز نمیکنه. اسکنر مجازی میشه نصب کرد و تست گرفت

نویسنده: محمد رعیت پیشه
تاریخ: ۱۳۹۳/۱۱/۲۲ ۱۷:۵۲

اینجا رو بررسی کنید و ببینید خطایی که دریافت می‌کنید توضیحش چیست. موردی که خودم تست کردم این بود که وقتی روی سیستمی که اسکنر متصل نبود امتحان می‌کردم، به خطا می‌خوردم و به محض اتصال اسکنر خطا برطرف می‌شد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۱/۲۲ ۱۸:۱۷

در همان فایل ScannerService اگر catch را به این صورت تغییر دهید، استثناءها در کنسول سرویس نمایش داده می‌شوند.

```
catch(Exception ex)
{
  Console.WriteLine(ex);
```

برای مثال این مورد را من دریافت می‌کنم:

```
System.Runtime.InteropServices.COMException (0x80210015):
Exception from HRESULT: 0x80210015 at WIA.CommonDialogClass.ShowAcquireImage
```

کد 0x80210015 به معنای عدم یافت شدن اسکنر است (که صحیح است).

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۱/۲۲ ۱۸:۵۱

یک نکته‌ی تکمیلی

در متد GetScan در پایان کار، نیاز است منابع COM مورد استفاده رها شوند تا نشتی حافظه رخ ندهد:

```
finally
{
  if(image!=null) Marshal.ReleaseComObject(image);
  if(dialog!=null) Marshal.ReleaseComObject(dialog);
}
```

نویسنده: شروین ایرانی
تاریخ: ۱۳۹۳/۱۱/۲۸ ۱۰:۹

با سلام؛ من یه عکسی تهیه کردم از مراحل اجرای برنامه. ابتدا پروژه SelfHostService را Build کردم و سپس فایل exe را Run Administrator کردم و کنسول باز شد.سپس پروژه WebApplicationTest را بوسیله مرورگر فایرفاکس اجرا کردم. ولی تصویر پیش فرض را لود کرد. ضمنا در FireBug هم خطایی نداد. مشکل کجاست. اسکنر هم نصب هست. مرسی.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۱/۲۸ ۱۰:۱۳

در نظرات قبلی عنوان شد : «... در همان فایل ScannerService اگر catch را به این صورت تغییر دهید ...»

نویسنده: شروین ایرانی
تاریخ: ۱۳۹۳/۱۱/۲۸ ۱۰:۲۴

بله انجام شد. خطای 0x80210015 میده. ولی اسکنر کاملا نصب و کار میکنه در محیط‌های دیگه.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۱/۲۸ ۱۱:۸

- معنای دوم خطای 0x80210015 عدم سازگاری درایورهای اسکنر شما با WIA است (Windows Image Acquisition). اسکنر شما احتمالا فقط TWAIN compatible است و WIA or TWAIN compatible نیست.

- لیستی از این نوع اسکنرها

نویسنده: محمدرضابرخورداری
تاریخ: ۱۳۹۳/۱۲/۱۴ ۱۲:۱۱

با سلام؛ من تمامی مراحل رو طی کردم اما با خطای زیر مواجه شدم :

```
Host is running.
System.Runtime.InteropServices.COMException (0x80070050): The file exists. (Exception from HRESULT:
0x80070050)
at WIA.ImageFileClass.SaveFile(String Filename)
at SelfHost.ScannerService.GetScan() in
g:\project\asnad\scan\SelfHostWcfScannerService\SelfHostWcfService\ScannerService.cs:line 25
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۲/۱۴ ۱۲:۳۴

سطر Path.GetTempFileName را تبدیل کنید به مسیری که به آن دسترسی دارید. همچنین بررسی کنید نام این فایل منحصربفرد باشد.

نویسنده: محمدرضابرخوردار
تاریخ: ۱۳۹۳/۱۲/۱۶ ۱۹:۴۱

بسیار ممنون، مشکل حل شد. یه سوال دیگه : آیا در WIA امکان اسکن دو رو هم وجود داره؟ به چه شکلی

نویسنده: محمد رعیت پیشه
تاریخ: ۱۳۹۴/۰۱/۱۴ ۱۶:۴۲

پاسخ شما اسکن کردن در قالب تصاویر tiff هست، که می‌توانند چندین عکس در یک عکس باشند، مثل اسکن پشت و رو کارت ملی.
فقط برای نمایش در مرورگر به مشکل برخورد می‌کنید که یک راه استفاده از افزونه AlternaTIFF می باشد.

نویسنده: محمدرضابرخوردار
تاریخ: ۱۳۹۴/۰۱/۱۵ ۱۳:۱۲

Path.GetTempFileName باعث خطای The file exists می‌شود. زمانیکه با آدرس قابل دسترس و نام ثابت برای فایل جایگزین میکنم برای بار دوم همین خطا را میدهد.

نویسنده: محمد رعیت پیشه
تاریخ: ۱۳۹۴/۰۱/۱۶ ۲۳:۳۴

به نظرم یکتا بودن آدرس فایل در سمت کلاینت ضروری نمی‌باشد، در اینصورت می‌توانید یک مسیر ثابت و مشخص رو در نظر بگیرید و قبل از ذخیره، وجود اون رو بررسی کنید و چنانچه از قبل وجود داشت ابتدا آن را پاک نموده سپس ذخیره کنید.

نویسنده: فطرسی
تاریخ: ۱۳۹۴/۰۳/۲۳ ۲۰:۴۶

با سلام و عرض خسته نباشید.
خیلی ممنون بابت زحمتی که کشیدید و بسیار عالی است. من می‌خواستم که پروژه SelfHostService رو به صورت Windows Service استفاده کنم البته می‌دونم که برای اینکار نمی‌شه چون پنجره اسکنر رو نمیشه نشون بده ولی شما فرض کنید همون رشته که در صورت خطا برمیگردونه رو بخواهم برگردونم به صفحه. حالا دوباره به مشکل Cors برخورد کردم در حالت قبل شما خیلی خوب اون رو حل کردید ولی برای ویندوز سرویس هم میشه قرار بدهید. با تشکر این خیلی به دردم می‌خوره.

نویسنده: محسن خان
تاریخ: ۱۳۹۴/۰۳/۲۳ ۲۳:۴۶

پروژه دومین نظر این بحث از بالا، سرویس را درون یک برنامه‌ی ویندوزی اجرا کرده.

نویسنده: محمد رعیت پیشه
تاریخ: ۱۳۹۴/۰۳/۲۴ ۱:۲

اتفاقا من این پروژه رو روی ویندوز سرویس هاست کرده بودم و مشکلی هم نداشت!
در واقع این شما نیستید که بخواهید فرمی را نشان بدهید، این مورد بر عهده کتابخانه WIA می‌باشد.
برای اون مشکل یعنی با کلاس CORSSupportBehavior هم حل نشده؟
بحث‌های نصب یک سرویس که خارج از مطلب هست اما به سادگی می‌تونید پیدا کنید، پس از نصب سرویس شما در متد OnStart این کدها را خواهید داشت

```
_host = new ServiceHost(typeof(WcfServiceScanner.ScannerService), _baseAddress);
    _host.Open();
```

و در متد OnStop هم

```
_host.Close();
```

بقیه موارد فرقی با هاست روی یک پروژه کنسول نمی‌کند.