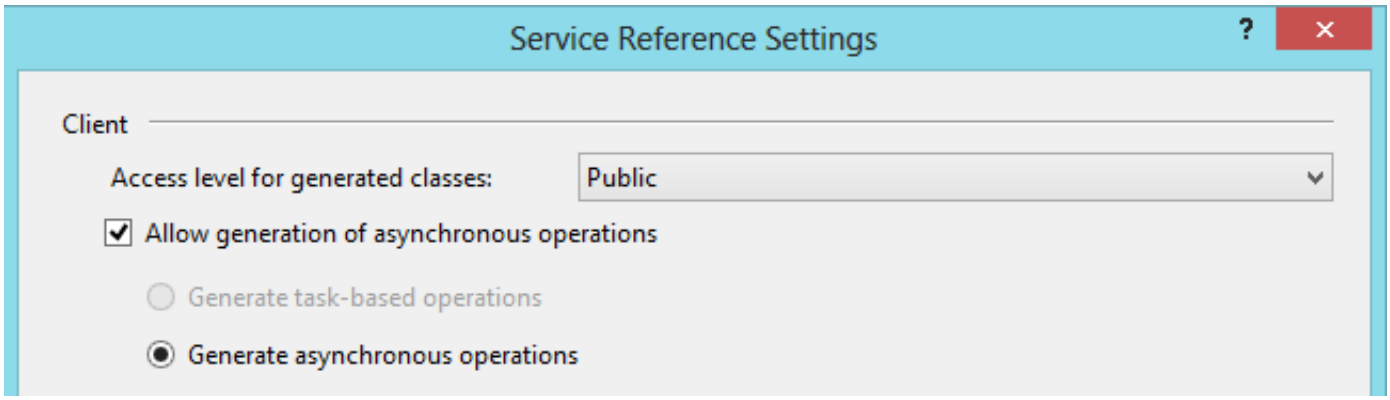
عنوان: **فراخوانی سرویس های WCF به صورت Async**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۴/۰۵ ۸:۳۰
آدرس: www.dotnettips.info
گروه‌ها: WCF, Asynchronous Programming, .NET 4.0, .Net 3.5

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:

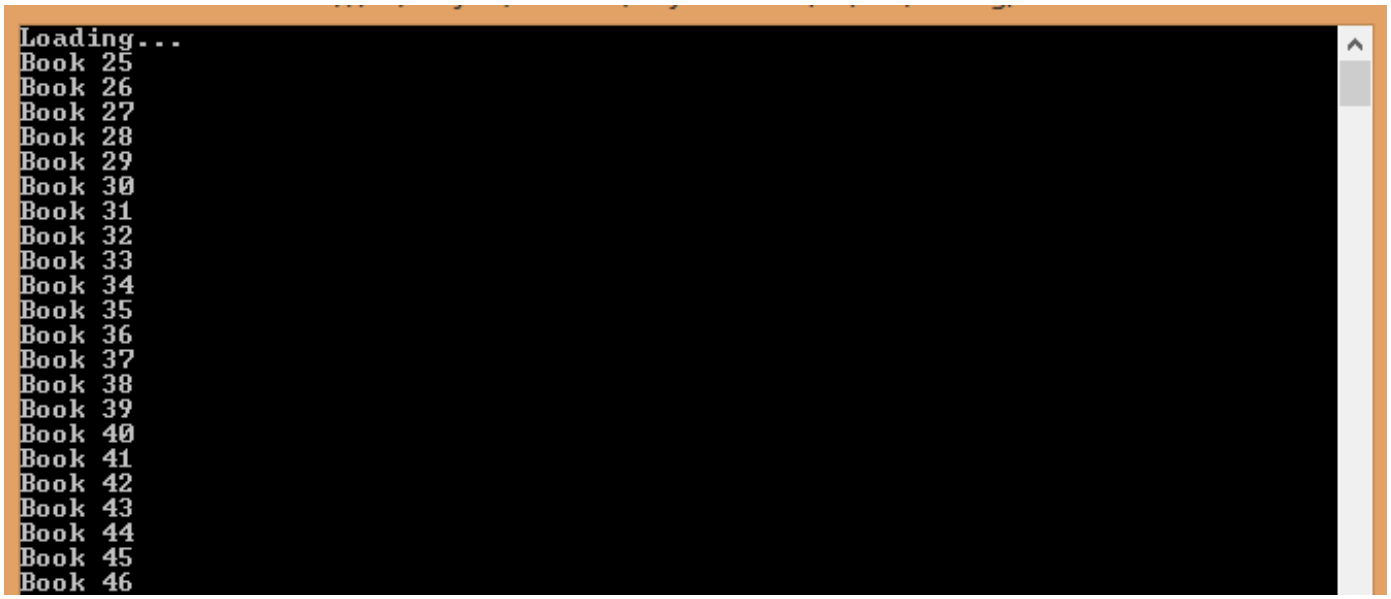
```
"http://localhost:7000/" +  "BookService.svc"
```

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.