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.