عنوان: **Serialization #1**
نویسنده: سیدمجتبی حسینی
تاریخ: ۱۳۹۱/۰۷/۱۳ ۲:۴۲
آدرس: www.dotnettips.info
برچسب‌ها: C#, Serialization

در این نوشتار به Serialization و Deserialization یعنی مکانیزمی که توسط آن اشیاء می‌توانند به صورت متنی مسطح و یا به شکل باینری درآیند، پرداخته می‌شود.

مفهوم Serialization

سریالی کردن، عملیاتی است که یک شیء و یا مجموعه اشیائی که به یکدیگر ارجاع می‌دهند را به شکل گروهی از بایت‌ها یا فرمت XML درآورده که می‌توان آن‌ها را ذخیره کرد و یا انتقال داد.
Deserialization معکوس عملیات بالاست که گروهی از داده‌ها را دریافت کرده و بصورت یک شیء و یا مجموعه‌ای از اشیائی که به یکدیگر ارجاع می‌دهند، تبدیل می‌کند.
Serialization و Deserialization نوعاً برای موارد زیر بکار می‌روند:
انتقال اشیاء از طریق یک شبکه یا مرز یک نرم افزار .

ذخیره اشیاء در یک فایل یا بانک اطلاعاتی.

موتورهای سریالی‌کننده

چهار شیوه برای سریالی کردن در دات نت فریم ورک وجود دارد:
سریالی‌کننده قرارداد داده (Data Contract Serializer)

سریالی‌کننده باینری

سریالی‌کننده XML مبتنی بر صفت

سریالی‌کننده اینترفیس IXmlSerializer

سه مورد اول، موتورهای سریالی‌کننده‌ای هستند که بیشترین یا تقریباً همه کارهای سریالی‌کردن را انجام می‌دهند. مورد آخر برای انجام مواردی است که خودتان قصد سریالی‌سازی دارید که این موتور با استفاده از XmlReader و XmlWriter این کار را انجام می‌دهد. IXmlSerializer می‌تواند به همراه سریالی‌کننده قرارداد داده و یا سریالی‌کننده XML در موارد پیچیده سریالی‌کردن استفاده شود.

سریالی‌کننده Data Contract

برای انجام این کار دو نوع سریالی‌کننده وجود دارد :
DataContractSerializer، که اصطلاحاً loosely Coupled شده است به نوع سریالی‌کننده Data Contract.

NetDataContractSerializer که اصطلاحاً tightly Coupled شده است به نوع سریالی‌کننده Data Contract.

مدل زیر را در نظر بگیرید:

```
public class News
    {
        public int Id;
```

```
        public string Body;
        public DateTime NewsDate;
    }
```

برای سریالی‌کردن نوع News به شیوه Data Contract، باید:

فضای نام System.Runtime.Serialization را به کد برنامه اضافه کنیم.

صفت [DataContract] را به نوعی که تعریف کرده‌ایم، اضافه کنیم.

صفت [DataMember] را به اعضایی که می‌خواهیم در سریالی شدن شرکت‌کنند، اضافه کنیم.

و به این ترتیب کلاس News به شکل زیر درمی‌آید:

```
using System.Runtime.Serialization;
using System.Xml;
using System;
using System.IO;
namespace ConsoleApplication1
{
    [DataContract]
    public class News
    {
        [DataMember] public int Id;
        [DataMember] public string Body;
        [DataMember] public DateTime NewsDate;
    }
}
```

سپس به شکل زیر از مدل خود نمونه‌ای ساخته و با ایجاد یک فایل، نتیجه سریالی شده مدل را در آن ذخیره میکنیم .

```
var news = new News
                           {
                               Id = 1,
                               Body = "NewsBody",
                               NewsDate = new DateTime(2012, 10, 4)
                           };
 var ds = new DataContractSerializer(typeof (News));
 using (Stream s=File.Create("News.Xml"))
 {
        ds.WriteObject(s, news);//سریالی‌کردن
 }
```

که محتویات فایل News.Xml به صورت زیر است:

```
<News xmlns="http://schemas.datacontract.org/2004/07/ConsoleApplication7"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance"><Body>NewsBody</Body><Id>1</Id><NewsDate>2012-10-
04T00:00:00</NewsDate></News>
```

و برای Deserialize کردن این فایل داریم:

```
News deserializednews;
using (Stream s = File.OpenRead("News.Xml"))
{
       deserializednews = (News)ds.ReadObject(s);//Deserializing
}
Console.WriteLine(deserializednews.Body);
```

همان‌طور که ملاحظه می‌کنید فایل ایجاد شده از خوانایی خوبی برخوردار نیست که برای دستیابی به فایلی با خوانایی بالاتر از

XmlWriter استفاده میکنیم:

```
XmlWriterSettings settings = new XmlWriterSettings {Indent = true};
using (XmlWriter writer = XmlWriter.Create("News.Xml", settings))
{
          ds.WriteObject(writer, news);
}
System.Diagnostics.Process.Start("News.Xml");
```

به این ترتیب موفق به ایجاد فایلی با خوانایی بالاتر می‌شویم:

```
<?xml version="1.0" encoding="utf-8"?>
<News xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.datacontract.org/2004/07/ConsoleApplication7">
  <Body>NewsBody</Body>
  <Id>1</Id>
  <NewsDate>2012-10-04T00:00:00</NewsDate>
</News>
```

نظرات خوانندگان

نویسنده: محمد
تاریخ: ۱۳۹۱/۰۷/۱۳ ۱۱:۴۲

دست شما درد نکند خیلی اینا رو میدیدم ولی نمیفهمیدم چین.
ممنون که روشنمون کردی

عنوان: **Serialization #2**
نویسنده: سیدمجتبی حسینی
تاریخ: ۱۳۹۱/۰۷/۱۵ ۲۳:۱۷
آدرس: www.dotnettips.info
برچسب‌ها: C#, Serialization

مطابق آنچه در قسمت قبل گفته شد برای آن‌که بتوان از مدل News برای سریالی‌کردن استفاده کرد، باید آن را به شکل ذیل پیاده‌سازی کرد:

```
[DataContract]
public class News
{
     [DataMember] public int Id;
     [DataMember] public string Body;
     [DataMember] public DateTime NewsDate;
}
```

با Override کردن [DataContract] به صورت [DataContract(Name=”MyCustomNews")] می‌توان نام ریشه XML فایل را به MyCustomNews تغییر داد. همچنین با Override کردن [DataMember] بصورت [DataMember(Name=”MyCustomFieldName")] می‌شود به هر فیلدی عنوان دلخواهی داد و همچنین با تعیین عبارت NameSpace به صورت [DataContract(Name =") "MyCustomNews", Namespace = "http://www.my.com"] می‌شود فضای نام را تغییر داد که با این تغییرات، خروجی زیر حاصل می‌شود:

```
<?xml version="1.0" encoding="utf-8"?>
<MyCustomNews xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.my.com">
  <Body>NewsBody</Body>
  <MyCustomFieldName>111</MyCustomFieldName>
  <NewsDate>2012-10-04T00:00:00</NewsDate>
</MyCustomNews>
```

ویژگی [DataMember] هم ازفیلد ها و هم از property ها، پشتیبانی می‌کند، خواه عمومی باشند یا خصوصی و نوع فیلد یا Property می‌تواند به یکی از اشکال زیر باشد:

انواع اولیه .

انواع Uri ، Guid ، TimeSpan ، DateTime و انواع Enum

انواع پوچ پذیر هر کدام از موارد بالا

نوع []byte

انواع تعریف شده توسط کاربر که توسط صفت [DataContract] محصور شده‌اند.

هر نوع IEnumerable

هر نوعی که با صفت [Serializable] محصور شود و یا اینترفیس ISerializable را پیاده سازی کند.

هر نوعی که اینترفیس IXmlSerializble را پیاده سازی نماید.

تعیین فرمت باینری برای سریالی‌کردن:

برای سریالی کننده‌های DataContractSerializer و NetDataContractSerializer می‌توان به روش زیر فرمت خروجی را به شکل فرمت باینری درآورد که خروجی آن تاحد زیادی کوچک‌تر و کم حجم‌تر می‌شود:

```
var s = new MemoryStream();
using (XmlDictionaryWriter w=XmlDictionaryWriter.CreateBinaryWriter(s))
{
        ds.WriteObject(w,news);
}
```

و برای Deserialize کردن آن به شیوه زیر عمل می‌کنیم:

```
var s2 = new MemoryStream(s.ToArray());
News deserializednews;
using (XmlDictionaryReader r=XmlDictionaryReader.CreateBinaryReader(s2,XmlDictionaryReaderQuotas.Max))
{
     deserializednews = (News)ds.ReadObject(r);
}
```

که در آن از ویژگی Max کلاس XmlDictionaryReaderQuotas برای به دست آوردن حداکثر سهمیه فضای دیسک مربوط به XmlDictionaryReaders استفاده می‌شود.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۱۶ ۱:۸

با تشکر. یک کتابخانه XML Serialization هم توسط آقای سینا ایروانیان تهیه شده که یک سری از محدودیت‌های کتابخانه‌های توکار دات نت را برطرف کرده.

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.

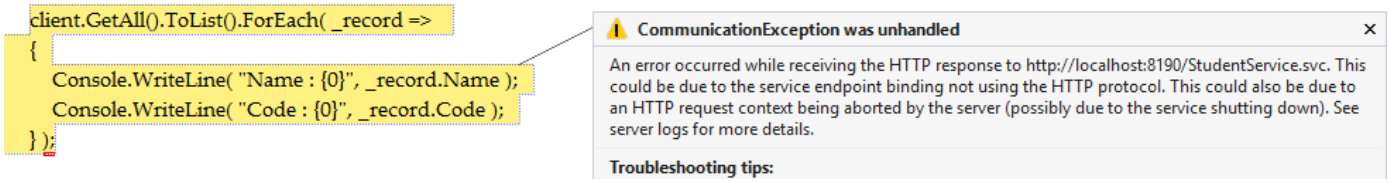
```
public class StudentService : IStudentService
    {
        public IEnumerable<Person> GetAll()
        {
            List<Person> listOfPerson = new List<Person>();

            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();
        }
    }
```

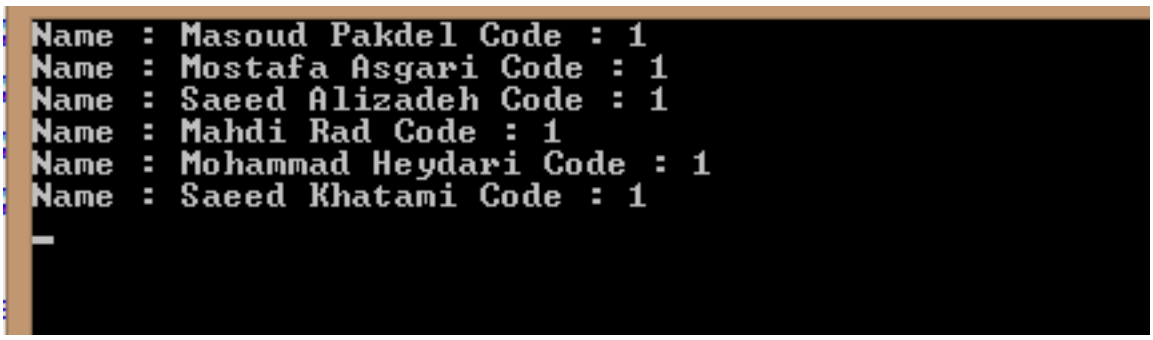
پروژه رو کامپایل کنید. تا اینجا هیچ گونه مشکلی مشاهده نشد و انتظار داریم که خروجی مورد نظر رو مشاهده کنیم. بعد از اجرای پروژه با خطای زیر متوقف می‌شویم:



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.



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

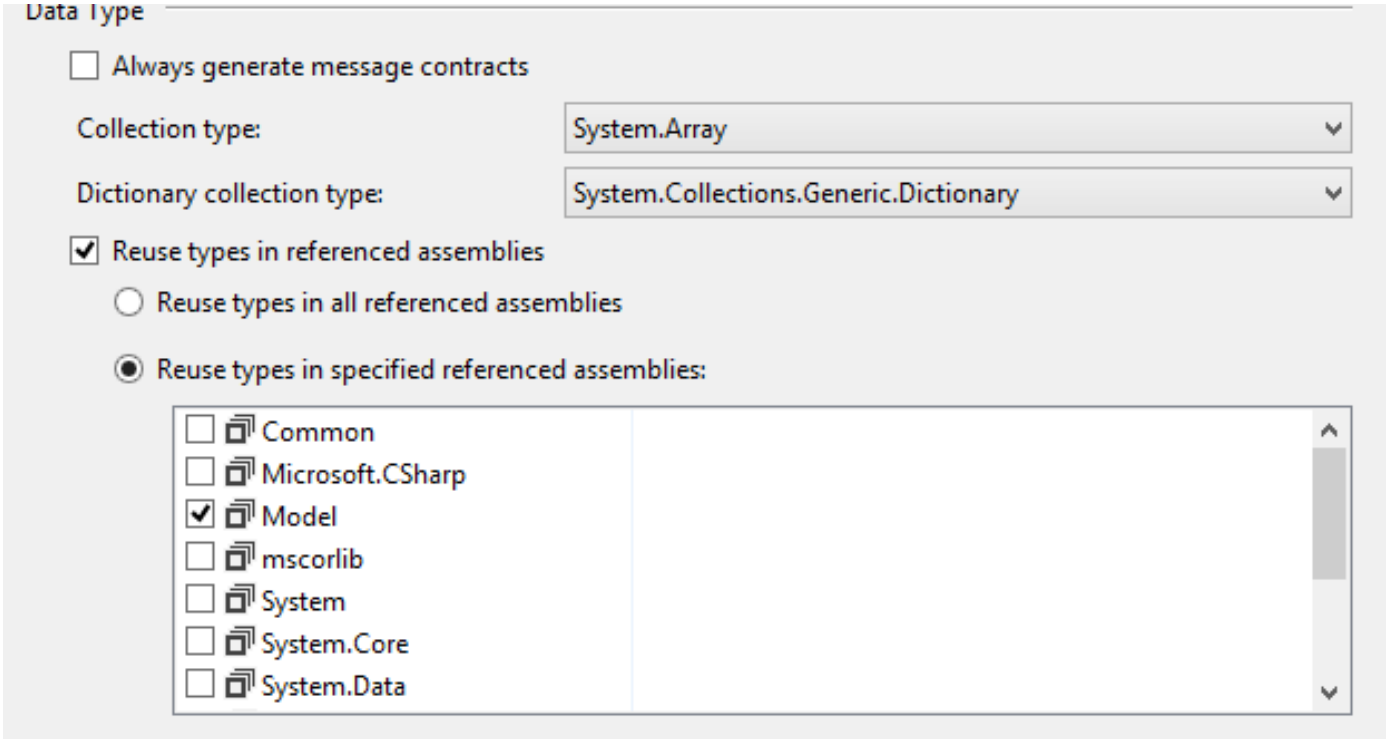
یک Contract برای ارتباط بین سرور و کلاینت می‌سازیم:

```
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 );
    }
```

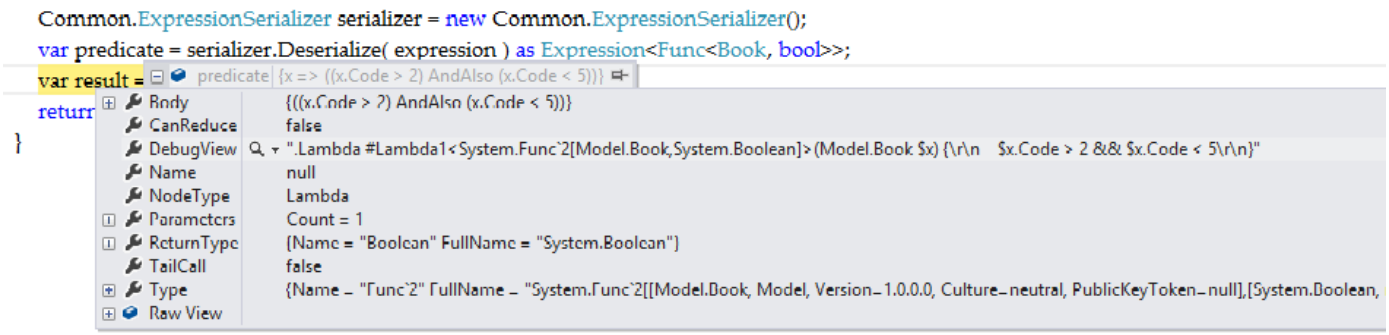
و سرویس :

```
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>> );
        }
    }
```

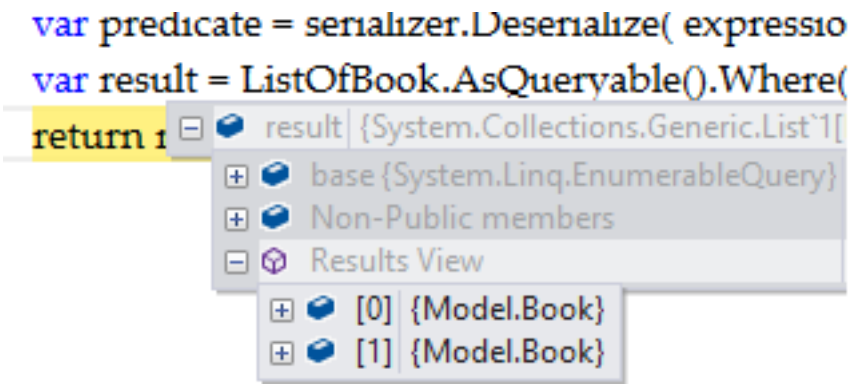
بعد از Build پروژه از روش Add Service Reference استفاده کنید و می‌بینید که بدون هیچ گونه مشکلی سرویس مورد نظر به پروژه Console اضافه شد. برای استفاده سمت کلاینت به صورت زیر عمل کنید.

```
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.



خروجی هم به صورت زیر خواهد بود:



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: **بررسی خطای Circular References در ASP.NET MVC Json Serialization**
نویسنده: مجتبی کاویانی
تاریخ: ۱۳۹۲/۰۹/۱۸ ۰:۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, JSON, Serialization, MVC

خیلی وقت‌ها لازم است تا نتیجه کوئری حاصله را بصورت Json به ویوی مورد نظر ارسال نمایید. برای اینکار کافیست مانند زیر عمل کنیم

```
[HttpGet]
public JsonResult Get(int id)
{
    return Json(repository.Find(id), JsonRequestBehavior.AllowGet);
}
```

اما اگر کوئری پیچیده و یا یک مدل سلسله مراتبی داشته باشید که با خودش کلید خارجی داشته باشد، هنگام تبدیل نتایج به خروجی Json، با خطای Circular References مواجه می‌شوید.

```
A circular reference was detected while serializing an object of type
‘System.Data.Entity.DynamicProxies.ItemCategory_A79…’
```

علت این مشکل این است که Json Serialization پش فرض ASP.NET MVC فقط یک سطح پایین‌تر را لود می‌کند و در مدل‌های که خاصیتی از نوع خودشان داشته باشند خطای Circular References را فرا می‌خواند. کلاس نمونه در زیر آوره شده است.

```
public class Item
    {
        public int Id { get; set; }
        [ForeignKey]
        public int ItemId { get; set; }
        public string Name { get; set; }
        public ICollection<Item> Items { get; set; }
    }
```

راه حل:

چندین راه حل برای رفع این خطا وجود دارد؛ یکی استفاده از Automapper و راه حل دیگر استفاده از کتابخانه‌های‌های قوی‌تر کار بار Json مثل Json.net است. اما راه حل ساده‌تر تبدیل خروجی کوئری به یک شی بی نام و سپس تبدیل به Json می‌باشد

```
[HttpGet]
public JsonResult List()
{
    var data = repository.AllIncluding(itemcategory => itemcategory.Items);
    var collection = data.Select(x => new
    {
        id = x.Id,
        name = x.Name,
        items = x.Items.Select(item => new
        {
            id=item.Id,
            name = item.Name
        })
    });
    return Json(collection, JsonRequestBehavior.AllowGet);
}
```

همین طور که در مثال بالا مشاهده می‌نمایید ابتدا همه رکوردها در متغییر data ریخته شده و سپس با یک کوئری دیگر که در آن دوباره از پروپرتی items که از نوع کلاس item می‌باشد شی بی نامی ایجاد نموده ایم. با این کار براحتی این خطا رفع می‌گردد.

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۹/۱۸ ۰:۴۹

با تشکر. یک سؤال: آیا تنظیم context.Configuration.ProxyCreationEnabled = false قبل از نوشتن کوئری Find (بلافاصله پس از ایجاد context) مشکل را حل می‌کند؟

نویسنده: مجتبی کاویانی
تاریخ: ۱۳۹۲/۰۹/۱۸ ۱۶:۲

خیر؛ این خطا مربوط به Json Serialization می‌باشد. ProxyCreation برای مباحث Lazy Loading و Change Tracking کاربرد دارد.

نویسنده: Ara
تاریخ: ۱۳۹۲/۰۹/۲۷ ۲۳:۹

سلام؛ راست می‌گند. اگه شما یک ابجکت رو مستقیم از dbcontext بگیرید و بدون اون که lazyloading غیر فعال باشه بدین به serializer تمام روابط اون آبجکت هم سریالایز می‌شوند که خیلی مشکل زاست حتی با json دات نت و اگر اون شی با شی دیگه که اون هم با این شی رابطه داشته باشه تو Cycle می‌افته و بهترین روش همونی بود که دوستمون گفتند یا استفاده از viewModel یا DTO هاست.

نویسنده: محمد
تاریخ: ۱۳۹۳/۰۶/۱۲ ۱۸:۵

سلام وخسته نباشید . من تو اینترنت سرچ کردم توی stack گفته بودند که اگه به صورت عمومی غیر فعالش کنی هم میشه. این کد رو هم گفته بودند تو قسمت Application_Start بزارید درست میشه

```
GlobalConfiguration.Configuration.Formatters.JsonFormatter.SerializerSettings.ReferenceLoopHandling =
  Newtonsoft.Json.ReferenceLoopHandling.Serialize;
GlobalConfiguration.Configuration.Formatters.JsonFormatter.SerializerSettings.PreserveReferencesHandlin
g =
  Newtonsoft.Json.PreserveReferencesHandling.Objects;
```

ولی برای من نشد. من میخوام به طور عمومی طوری تنظیمش کنم که اگه جایی به circular برخورد کرد بیخیالش بشه و ارور نده. آیا راهی وجود داره؟

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۰۶/۱۲ ۱۸:۱۸

GlobalConfiguration.Configuration.Formatters مربوط به Web API هست. برای MVC باید return Json توکار رو با نمونه Newtonsoft.Json در همه جا تعویض کنید.

نویسنده: محمد
تاریخ: ۱۳۹۳/۰۶/۱۲ ۱۸:۲۸

خیلی ممنون . میشه یه نمونه کد یا سایت یا چیزی برام بزارید که من دقیقا بدونم چیرو کجا و چجوری تغییر بدم؟
کاری که من خودم کرده بودم این بود که از کلاس JsonResult یک کلاس دیگه ساخته بودم که ازش ارث می‌برد و بعد با تنظیمات ReferenceLoopHandling.Ignore متد Execute.... اون رو override کردم . جواب هم داد . فقط یه گیری داشت .اونم اینکه من تو مدلم یک فیلدی دارم که از نوع Byte[] هستش . و توش فایل هامو نگه میدارم . تو حالتی که اولیه خودش من بالای این فیلد [ScriptIgnore] گذاشته بودم و خوب کار میکرد . اما وقتی با این کلاس جدیدم اونو serelize میکنم همه چیزو serelize میکنه و

این باعث شده خیلی کند بشه . یه راهنمایی بکنید که یا حالت اول باشه ولی ارور circular نده یا حالت دوم باشه ولی فیلدهای باینری رو serelize نکنه . ممنون میشم کمکم کنید .

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۰۶/۱۲ ۱۸:۳۵

در Newtonsoft.Json برای صرفنظر کردن از یک خاصیت، یا از ویژگی IgnoreDataMember استفاده کنید یا از ویژگی JsonIgnore آن.

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