بسم الله الرحمن الرحیم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتابیس در ایران

serial کردن JSON و XML در ASP.NET Web API

مدرس : مهندس افشین رفوآ

دوره آموزشی Web API

در این مقاله قصد داریم قالب بندی JSON و XML را در ASP.NET Web API توضیح دهیم. در asp.net web API یک media formatter می تواند:

- object های CLR را از پیام HTTP بخواند.
- object های CLR را در پیام HTTP بنویسد.

Web API هر دو نوع JSON و XML را در بر می گیرد. فریم ورک، این قالب بندی ها را در خط لوله (pipeline)به صورت پیش فرض قرار می دهد. کاربران در درخواست HTTP می توانند هر دو نوع JSON و XML را درخواست کنند.

قالب بندی داده JSON (media-type formatter)

قالب بندی JSON با کلاس JsonMediaTypeFormatter تعریف شده است. به صورت پیش فرض، JsonMediaTypeFormatter از کتابخانه Json.NET برای serial کردن استفاده می کند. Json.NET یک پروژه شخص ثالث(third-party) متن باز می باشد.

شما می توانید کلاس JsonMediaTypeFormatter را برای استفاده از DataContractJsonSerializer به جای Json.NET پیکره بندی کنید. برای این کار، خاصیت UseDataContractJsonSerializer را True قرار دهید:

```
var json = GlobalConfiguration.Configuration.Formatters.JsonFormatter;
json.UseDataContractJsonSerializer = true;
```

serial کردن JSON

در این قسمت بعضی از رفتار های خاص قالب بندی JSON را با استفاده از قالب Json.NET توضیح می دهیم البته باید به این نکته نیز اشاره کرد که این مستندات توضیح کامل و جامع از کتابخانه Json.NET نمی باشد و تنها اشاره ای به آن می باشد.

serial کردن چطور انجام می گیرد؟

به صورت پیش فرض، تمام خاصیت های عمومی، شامل serial کردن JSON می باشند. برای حذف یک خاصیت، باید به آن JsonIgnore attribute را اضافه کنید:

```
public class Product
{
    public string Name { get; set; }
    public decimal Price { get; set; }
    [JsonIgnore]
    public int ProductCode { get; set; } // omitted
}
```

اگر بخواهید رویکرد opt-in را دنبال کنید،باید DataContract attribute را به کلاس اضافه کنید. اگر این attribute موجود باشد، اعضا (member)نادیده گرفته می شوند مگر آنکه مقدار DataMember را داشته باشند. شما همچنین می توانید از DataMember برای serial کردن اعضای خصوصی(private member) استفاده کنید.

```
[DataContract]
public class Product
{
    [DataMember]
    public string Name { get; set; }
    [DataMember]
    public decimal Price { get; set; }
    public int ProductCode { get; set; }  // omitted by default
}
```

خاصیت های فقط خواندنی(Read-only)

خاصیت های فقط خواندنی به صورت پیش فرض serial می باشند.

تاریخ

به صورت پیش فرض، Json.NET تاریخ را بر اساس فرمت ISO 8601 می نویسد. فرمت تاریخ در UTC(تاریخ هماهنگ جهانی) با پسوند Z نوشته می شود. تاریخ یک زمان محلی شامل یک آفست نیز می باشد. برای مثال:

```
2012-07-27T18:51:45.53403Z          // UTC
2012-07-27T11:51:45.53403-07:00     // Local
```

به طور پیش فرض، Json.NET از محدوده زمانی حفاظت می کند. با قرار دادن خاصیت DateTimeZoneHandling شما می توانید این ویژگی را override کنید.

```
// Convert all dates to UTC
var json = GlobalConfiguration.Configuration.Formatters.JsonFormatter;
json.SerializerSettings.DateTimeZoneHandling = Newtonsoft.Json.DateTimeZoneHandling.Utc;
```

اگر می خواهید از فرمت Microsoft JSON date format (`"\/Date(ticks)\/"`) به جای ISO 8001 استفاده کنید، خاصیت DateFormatHandling را در تنظیمات قالب تنظیم کنید:

```
var json = GlobalConfiguration.Configuration.Formatters.JsonFormatter;
json.SerializerSettings.DateFormatHandling
= Newtonsoft.Json.DateFormatHandling.MicrosoftDateFormat;
```

Intent کردن

برای نوشتن intent JSON، تنظیمات Formatting را Formatting.**Indented** قرار دهید.

```
var json = GlobalConfiguration.Configuration.Formatters.JsonFormatter;
json.SerializerSettings.Formatting = Newtonsoft.Json.Formatting.Indented;
```

Camel Casing

برای نوشتن خاصیت JSON با camel casing بدون آنکه data model شما تغییر کند،

CamelCasePropertyNamesContractResolver را روی قالب قرار دهید:

```
var json = GlobalConfiguration.Configuration.Formatters.JsonFormatter;
json.SerializerSettings.ContractResolver = new CamelCasePropertyNamesContractResolver();
```

object های ناشناس و Weakly-Typed

یک متد action می تواند یک object ناشناس برگرداند و آن را در serial JSON کند. برای مثال:

```
public object Get()
{
    return new {
        Name = "Alice",
        Age = 23,
        Pets = new List<string> { "Fido", "Polly", "Spot" }
    };
}
```

پاسخ شامل JSON زیر می باشد:

```
{"Name":"Alice","Age":23,"Pets":["Fido","Polly","Spot"]}
```

اگر Web API از سمت کاربر object های JSON ناقص دریافت کند شما می توانید درخواست را در نوع Newtonsoft.Json.Linq.JObject ، deserial کنید:

```
public void Post(JObject person)
{
    string name = person["Name"].ToString();
    int age = person["Age"].ToObject<int>();
}
```

با این حال، معمولا بهتر است از داده های Strongly typed استفاده کنید. قالب XML از داده های ناشناس یا موارد JObject پشتیبانی نمی کند. اگر شما این ویژگی ها را برای داده JSON خود استفاده کنید باید قالب XML را به صورتی که در ادامه خواهیم گفت از pipeline خود حذف کنید.

قالب XML

قالب XML از کلاس XmlMediaTypeFormatter به دست می آید. به صورت پیش فرض XmlMediaTypeFormatter از کلاس DataContractSerializer برای serial کردن استفاده می کند.

شما می توانید XmlMediaTypeFormatter را برای استفاده از XmlSerializer به جای DataContractSerializer تنظیم کنید. برای این کار مقدار UseXmlSerializer را true قرار دهید.

```
var xml = GlobalConfiguration.Configuration.Formatters.XmlFormatter;
```

```
xml.UseXmlSerializer = true;
```

کلاس XmlSerializer از media typeهای محدودتری نسبت به DataContractSerializer پشتیبانی می کند اما کنترل بیشتری روی نتایج XML دارد. اگر شما نیاز داشته باشید تا یک طرح XML موجود را بسط دهید می توانید از XmlSerializer استفاده کنید.

Serial کردن XML

در این قسمت بعضی از رفتار های خاص قالب XML را با استفاده از DataContractSerializer پیش فرض توضیح خواهیم داد:

به صورت پیش فرض، DataContractSerializer مانند زیر رفتار می کند:

- برای حذف تمام خاصیت های عمومی خواندن و نوشتن که serial شده اند از attribute IgnoreDataMember استفاده می کند.
- اعضای خصوصی و حفاظت شده را serial نمی کند.
- خواص read-only، serial نمی شوند(با این حال، محتوای یک مجموعه read-only، serial شده است)
- نام کلاس و اعضای نوشته شده در XML دقیقا همان هایی است که در تعریف کلاس مشخص شده است.
- از namespace پیش فرض XML استفاده می کند.

اگر نیاز به دسترسی و مدیریت بیشتری روی serial کردن نیاز دارید می توانید DataContract attribute را به کلاس اضافه کنید. زمانی که این attribute وجود داشته باشد، کلاس به صورت زیر serial می شود:

رویکرد Opt in: خاصیت ها به صورت پیش فرض serial نمی شوند و برای serial کردن آنها باید از attribute DataMember استفاده کرد.

برای serial کردن یک عضو خصوصی یا محافظت شده، از DataMember attribute استفاده می کنیم.

خاصیت های فقط خواندنی(read-only) serial نمی شوند.

برای نمایش نام کلاس در XML، پارامتر نام را در DataContract attribute قرار دهید.

برای نمایش نام اعضا در XML، پارامتر نام را در DataMember attribute قرار دهید.

برای نمایش نام namespace در XML، پارامتر نام را در DataContract attribute قرار دهید.

خاصیت های فقط خواندنی(read-only)

خاصیت های read-only، serial نمی شوند. اگر یک خاصیت read-only یک فیلد مخفی حمایت کننده داشته باشد می توانید فیلد مخفی را با DataMember attribute مشخص کنید. این رویکرد به attribute DataContract درون کلاس نیاز دارد.

```
[DataContract]
public class Product
{
    [DataMember]
    private int pcode;  // serialized
    // Not serialized (read-only)
    public int ProductCode { get { return pcode; } }
}
```

تاریخ

تاریخ در قالب ISO 8601 نوشته شده است. برای مثال 2012-05-23T20:21:37.9116538Z.

Intent کردن

برای نوشتن XML های Intent شده، خاصیت Intent را true قرار دهید:

```
var xml = GlobalConfiguration.Configuration.Formatters.XmlFormatter;
xml.Indent = true;
```

تنظیم کردن serial Per-Type کننده های XML

شما می توانید برای انواع مختلف CLR، روش های مختلف serial کردن را XML تنظیم کنید. برای مثال، فرض کنید شما داده خاصی دارید که برای سازگاری به XmlSerializer احتیاج دارد. شما می توانید از XmlSerializer برای این object استفاده کنید و برای انواع دیگر از DataContractSerializer استفاده کنید. برای تنظیم قالب XML برای یک مقدار خاص، SetSerializer را فراخوانی می کنیم.

```
var xml = GlobalConfiguration.Configuration.Formatters.XmlFormatter;
// Use XmlSerializer for instances of type "Product":
xml.SetSerializer<Product>(new XmlSerializer(typeof(Product)));
```

می توانید یک XmlSerializer و یا هر object را که از XmlObjectSerializer مشتق می شود مشخص سازید.

حذف کردن قالب JSON یا XML

اگر شما نیازی به قالب های JSON و XML نداشته باشید می توانید آن ها را حذف کنید. دلایل اصلی این کار در زیر نشان داده شده است:

- برای آنکه بخواهید پاسخ Web API را در قالب یک داده خاص محدود سازید. برای مثال، شما شاید بخواهید تنها از پاسخ های JSON پشتیبانی کنید و قالب XML را حذف کنید.
- برای آنکه بخواهید قالب پیش فرض را با قالب شخصی عوض کنید. برای مثال شاید شما بخواهید قالب JSON پیش فرض را با قالب JSON پیاده سازی شده خودتان جا به جا کنید.

کد زیر نحوه حذف قالب های پیش فرض را نمایش می دهد. این کد را از متد Application_Start خودتان که در Global.asax تعریف شده فراخوانی کنید.

```
void ConfigureApi(HttpConfiguration config)
{
    // Remove the JSON formatter
    config.Formatters.Remove(config.Formatters.JsonFormatter);
    // or
    // Remove the XML formatter
    config.Formatters.Remove(config.Formatters.XmlFormatter);
}
```

مدیریت مراجع دایره ای (Circular references)

به صورت پیش فرض، قالب های JSON و XML، تمامی object ها را به عنوان مقادیر می نویسند. اگر دو خاصیت به object مشابه اشاره کنند و یا اگر object مشابه دو بار در یک مجموعه نمایش داده شود، قالب object را دو بار serial می کند. زمانی که گراف شما دور داشته باشد این به یک موضوع خاص تبدیل می شود زیرا قالب باعث ایجاد یک مقدار خطا در زمانی که حلقه را در گراف شناسایی می کند می شود.

به model و Controller زیر دقت کنید.

```
public class Employee
{
    public string Name { get; set; }
    public Department Department { get; set; }
}
public class Department
{
    public string Name { get; set; }
    public Employee Manager { get; set; }
}
public class DepartmentsController : ApiController
{
    public Department Get(int id)
    {
        Department sales = new Department() { Name = "Sales" };
        Employee alice = new Employee() { Name = "Alice", Department = sales };
        sales.Manager = alice;
        return sales;
    }
}
```

فراخوانی این action باعث می شود باعث خطای 500(Internal Server Error) برای پاسخگویی به کاربر شود.

برای حفظ مراجع(references) object در JSON، کد زیر را به متد Application_Start که در Global.asax تعریف شده است اضافه کنید.

```
var json = GlobalConfiguration.Configuration.Formatters.JsonFormatter;
json.SerializerSettings.PreserveReferencesHandling =
 Newtonsoft.Json.PreserveReferencesHandling.All;
```

حالا داده ای از نوع JSON شبیه به کد زیر بر می گرداند:

```
{"$id":"1","Name":"Sales","Manager":{"$id":"2","Name":"Alice","Department":{"$ref":"1"}}}
```

دقت داشته باشید که قالب، خاصیت $id را به هر دو object اضافه کرده باشد. همچنین، توجه داشته باشید خاصیت Employee.Department یک حلقه می سازد و آنگاه مقدار آن را با مرجع object {"$ref":"1"} جا به جا می کند.

مراجع(references) object در JSON استاندارد نمی باشند. قبل از استفاده از این ویژگی، به این نکته توجه داشته باشید که کاربران شما بتوانند نتایج را تفسیر کنند. این ممکن است به سادگی باعث حذف دور ها از گراف شود.

برای حفظ کردنobject references در XML شما دو انتخاب دارید. انتخاب ساده تر این است که به کلاس model خود [DataContract(IsReference=true)] اضافه کنید. پارامتر ISRefrence، object references را فعال می کند. به خاطر داشته باشید DataContract ، serial کردن را Opt-in می کند. پس شما نیاز دارید DataMember attribute را به خاصیت ها اضافه کنید.

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

حالا قالب ، یک XML مشابه کد زیر تولید می کند:

```
<Department xmlns:i="http://www.w3.org/2001/XMLSchema-instance" z:Id="i1"
            xmlns:z="http://schemas.microsoft.com/2003/10/Serialization/"
            xmlns="http://schemas.datacontract.org/2004/07/Models">
  <Manager>
    <Department z:Ref="i1" />
    <Name>Alice</Name>
  </Manager>
  <Name>Sales</Name>
</Department>
```

شما یک راه دیگر دارید تا بتوانید attribute ها را از کلاس های model خود حذف کنید: یک DataContractSerializer مشخص بسازید و مقدار *preserveObjectReferences* را در constructor در حالت true قرار دهید. سپس این مورد را به عنوان قالب per-type در قالب XML تنظیم کنید. کد زیر چگونگی این فرآیند را نشان می دهد:

```
var xml = GlobalConfiguration.Configuration.Formatters.XmlFormatter;
var dcs = new DataContractSerializer(typeof(Department), null, int.MaxValue,
false, /* preserveObjectReferences: */ true, null);
xml.SetSerializer<Department>(dcs);
```

آزمایش serial کردن object

با توجه به طراحی Web API شما، بهتر است که چگونگی serial شدن داده های خود را آزمایش کنید. شما می توانید این کار را بدون ساختن یک Controller یا فراخوانی یک action کنترولر انجام دهید:

```
string Serialize<T>(MediaTypeFormatter formatter, T value)
{
    // Create a dummy HTTP Content.
    Stream stream = new MemoryStream();
    var content = new StreamContent(stream);
    /// Serialize the object.
    formatter.WriteToStreamAsync(typeof(T), value, stream, content, null).Wait();
    // Read the serialized string.
    stream.Position = 0;
    return content.ReadAsStringAsync().Result;
}
T Deserialize<T>(MediaTypeFormatter formatter, string str) where T : class
{
    // Write the serialized string to a memory stream.
    Stream stream = new MemoryStream();
    StreamWriter writer = new StreamWriter(stream);
    writer.Write(str);
    writer.Flush();
    stream.Position = 0;
    // Deserialize to an object of type T
    return formatter.ReadFromStreamAsync(typeof(T), stream, null, null).Result as T;
}
// Example of use
void TestSerialization()
{
    var value = new Person() { Name = "Alice", Age = 23 };
    var xml = new XmlMediaTypeFormatter();
    string str = Serialize(xml, value);
    var json = new JsonMediaTypeFormatter();
    str = Serialize(json, value);
    // Round trip
    Person person2 = Deserialize<Person>(json, str);
}
```