عنوان: **استفاده از JSON.NET در ASP.NET MVC**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۶/۱۵ ۲۳:۲۵
آدرس: www.dotnettips.info
گروه‌ها: MVC, JSON, json.net

تا نگارش فعلی ASP.NET MVC، یعنی نگارش 5 آن، به صورت توکار از JavaScriptSerializer برای پردازش JSON کمک گرفته می‌شود. این کلاس نسبت به JSON.NET هم کندتر است و هم قابلیت سفارشی سازی آنچنانی ندارد. برای مثال مشکل Self referencing loop را نمی‌تواند مدیریت کند.
برای استفاده از JSON.NET در یک اکشن متد، به صورت معمولی می‌توان به نحو ذیل عمل کرد:

```
[HttpGet]
        public ActionResult GetSimpleJsonData()
        {
            return new ContentResult
            {
                Content = JsonConvert.SerializeObject(new { id = 1 }),
                ContentType = "application/json",
                ContentEncoding = Encoding.UTF8
            };
        }
```

در اینجا با استفاده از متد JsonConvert.SerializeObject، اطلاعات شیء مدنظر تبدیل به یک رشته شده و سپس با content type مناسبی در اختیار مصرف کننده قرار می‌گیرد.
اگر بخواهیم این عملیات را کمی بهینه‌تر کنیم، نیاز است بتوانیم از استریم‌ها استفاده کرده و خروجی JSON را بدون تبدیل به رشته، مستقیما در استریم response.Output بنویسیم. با اینکار به سرعت بیشتر و همچنین مصرف منابع کمتری خواهیم رسید.
نمونه‌ای از این پیاده سازی را در ذیل مشاهده می‌کنید:

```
using System;
using System.Web.Mvc;
using Newtonsoft.Json;

namespace MvcJsonNetTests.Utils
{
    public class JsonNetResult : JsonResult
    {
        public JsonNetResult()
        {
            Settings = new JsonSerializerSettings { ReferenceLoopHandling = ReferenceLoopHandling.Error
};
        }

        public JsonSerializerSettings Settings { get; set; }

        public override void ExecuteResult(ControllerContext context)
        {
            if (context == null)
                throw new ArgumentNullException("context");

            if (this.JsonRequestBehavior == JsonRequestBehavior.DenyGet &&
                string.Equals(context.HttpContext.Request.HttpMethod, "GET",
StringComparison.OrdinalIgnoreCase))
            {
                throw new InvalidOperationException("To allow GET requests, set JsonRequestBehavior to
AllowGet.");
            }

            if (this.Data == null)
                return;

            var response = context.HttpContext.Response;
            response.ContentType = string.IsNullOrEmpty(this.ContentType) ? "application/json" :
this.ContentType;

            if (this.ContentEncoding != null)
                response.ContentEncoding = this.ContentEncoding;

            var serializer = JsonSerializer.Create(this.Settings);
            using (var writer = new JsonTextWriter(response.Output))
            {
```

```
                serializer.Serialize(writer, Data);
                writer.Flush();
            }
        }
    }
}
```

اگر دقت کنید، کار با ارث بری از JsonResult توکار ASP.NET MVC شروع شده‌است. کدهای ابتدای متد ExecuteResult با کدهای اصلی JsonResult یکی هستند. فقط انتهای کار بجای استفاده از JavaScriptSerializer، از JSON.NET استفاده شده‌است.
در این حالت برای استفاده از این Action Result جدید می‌توان نوشت:

```
[HttpGet]
        public ActionResult GetJsonData()
        {
            return new JsonNetResult
            {
                Data = new
                {
                    Id = 1,
                    Name = "Test 1"
                },
                JsonRequestBehavior = JsonRequestBehavior.AllowGet,
                Settings = { ReferenceLoopHandling = ReferenceLoopHandling.Ignore }
            };
        }
```

طراحی آن با توجه به ارث بری از JsonResult اصلی، مشابه نمونه‌ای است که هم اکنون از آن استفاده می‌کنید. فقط اینبار قابلیت تنظیم Settings پیشرفته‌ای نیز به آن اضافه شده‌است.

تا اینجا قسمت ارسال اطلاعات از سمت سرور به سمت کاربر بازنویسی شد. امکان بازنویسی و تعویض موتور پردازش JSON دریافتی از سمت کاربر، در سمت سرور نیز وجود دارد. خود ASP.NET MVC به صورت استاندارد توسط کلاسی به نام JsonValueProviderFactory ، اطلاعات اشیاء JSON دریافتی از سمت کاربر را پردازش می‌کند. در اینجا نیز اگر دقت کنید از کلاس JavaScriptSerializer استفاده شده‌است.
برای جایگزینی آن باید یک ValueProvider جدید را تهیه کنیم:

```
using System;
using System.Dynamic;
using System.Globalization;
using System.IO;
using System.Web.Mvc;
using Newtonsoft.Json;
using Newtonsoft.Json.Converters;

namespace MvcJsonNetTests.Utils
{
    public class JsonNetValueProviderFactory : ValueProviderFactory
    {
        public override IValueProvider GetValueProvider(ControllerContext controllerContext)
        {
            if (controllerContext == null)
                throw new ArgumentNullException("controllerContext");

            if (controllerContext.HttpContext == null ||
                controllerContext.HttpContext.Request == null ||
                controllerContext.HttpContext.Request.ContentType == null)
            {
                return null;
            }

            if (!controllerContext.HttpContext.Request.ContentType.StartsWith(
                    "application/json", StringComparison.OrdinalIgnoreCase))
            {
                return null;
            }

            using (var reader = new StreamReader(controllerContext.HttpContext.Request.InputStream))
            {
                var bodyText = reader.ReadToEnd();
```

```
                return string.IsNullOrEmpty(bodyText)
                    ? null
                    : new DictionaryValueProvider<object>(
                        JsonConvert.DeserializeObject<ExpandoObject>(bodyText, new
JsonSerializerSettings
                        {
                            Converters = { new ExpandoObjectConverter() }
                        }),
                        CultureInfo.CurrentCulture);
            }
        }
    }
}
```

در اینجا ابتدا بررسی می‌شود که آیا اطلاعات دریافتی دارای هدر application/json است یا خیر. اگر خیر، توسط این کلاس پردازش نخواهند شد.
در ادامه، اطلاعات JSON دریافتی به شکل یک رشته‌ی خام دریافت شده و سپس به متد JsonConvert.DeserializeObject ارسال می‌شود. با استفاده از تنظیم ExpandoObjectConverter، می‌توان محدودیت کلاس JavaScriptSerializer را در مورد خواص و یا پارامترهای dynamic، برطرف کرد.

```
[HttpPost]
  public ActionResult TestValueProvider(string data1, dynamic data2)
```

برای مثال اینبار می‌توان اطلاعات دریافتی را همانند امضای متد فوق، به یک پارامتر از نوع dynamic، بدون مشکل نگاشت کرد.

و در آخر برای معرفی این ValueProvider جدید می‌توان در فایل Global.asax.cs به نحو ذیل عمل نمود:

```
using System.Linq;
using System.Web.Mvc;
using System.Web.Routing;
using MvcJsonNetTests.Utils;

namespace MvcJsonNetTests
{
    public class MvcApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            AreaRegistration.RegisterAllAreas();
            RouteConfig.RegisterRoutes(RouteTable.Routes);

            ValueProviderFactories.Factories.Remove(
                ValueProviderFactories.Factories.OfType<JsonValueProviderFactory>().FirstOrDefault());
            ValueProviderFactories.Factories.Add(new JsonNetValueProviderFactory());
        }
    }
}
```

ابتدا نمونه‌ی قدیمی آن یعنی JsonValueProviderFactory حذف می‌شود و سپس نمونه‌ی جدیدی که از JSON.NET استفاده می‌کند، معرفی خواهد شد.

البته نگارش بعدی ASP.NET MVC موتور پردازشی JSON خود را از طریق تزریق وابستگی‌ها دریافت می‌کند و از همان ابتدای کار قابل تنظیم و تعویض است. مقدار پیش فرض آن نیز به JSON.NET تنظیم شده‌است.

دریافت یک مثال کامل

MvcJsonNetTests.zip

نظرات خوانندگان

نویسنده: مهدی پایروند
تاریخ: ۱۳۹۳/۰۶/۱۶ ۱۱:۱۸

تا جایی که من میدونم JavaScriptSerializer با Dictionaryها هم مشکل داشت ولی JSON.NET این مشکل رو نداره.

نویسنده: احمد
تاریخ: ۱۳۹۳/۰۶/۱۶ ۱۳:۲۲

سلام. اگر در مثال پیوست شده کلاس زیر را استفاده کنیم خطا می‌دهد:

```
public class MyClass
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }


 [HttpPost]
        public ActionResult TestValueProvider(string data1, MyClass data2)
        {

            var id = data2.Id;
            var name = data2.Name;

            return new JsonNetResult
            {
                Data = new { result = data1 },
                JsonRequestBehavior = JsonRequestBehavior.AllowGet,
                Settings = { ReferenceLoopHandling = ReferenceLoopHandling.Ignore }
            };
        }
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۶/۱۶ ۱۵:۱۷

نسخه‌ی بهبود یافته JsonNetValueProviderFactory را در اینجا می‌توانید مطالعه کنید. نسخه‌ی JsonNetResult آن جالب نیست چون از string استفاده کرده بجای stream .

JsonNetValueProviderFactory.cs

+ نحوه‌ی ثبت بهتر این کلاس دقیقا در همان ایندکس اصلی آن:

```
public static void RegisterFactory()
        {
            var defaultJsonFactory = ValueProviderFactories.Factories
                .OfType<JsonValueProviderFactory>().FirstOrDefault();
            var index = ValueProviderFactories.Factories.IndexOf(defaultJsonFactory);
            ValueProviderFactories.Factories.Remove(defaultJsonFactory);
            ValueProviderFactories.Factories.Insert(index, new JsonNetValueProviderFactory());
        }
```

نویسنده: احمد
تاریخ: ۱۳۹۳/۰۶/۱۶ ۱۵:۵۱

خیلی ممنون.
یک مشکل دیگر:

```
public enum MyEnum
    {
        One=1,
        Two=2
    }

 [HttpPost]
```

```
 public ActionResult TestValueProvider(string data1,MyEnum id,string name)
{
```

Enum قابل تبدیل نیست و خطای مشابه سوال قبل را می‌دهد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۶/۱۶ ۱۸:۵۱

مشکلی نیست. enum در سمت کلاینت باید به صورت رشته‌ای مقدار دهی شود:

```
$.ajax({
                //....
                data: JSON.stringify(
                {
                    data1: "Test1",
                    data2: { Id: 1, Name: "dynamic test" },
                    data3: 'Two' // مقدار دهی عضو ایی‌نام
                }),
```