

(JavaScript Object Notation) [JSON](#) یک راه مناسب برای نگهداری اطلاعات است و از لحاظ ساختاری شباهت زیادی به XML ، رقیب قدیمی خود دارد.

وب سرویس و آجاکس برای انتقال اطلاعات از این روش استفاده می‌کنند و بعضی از پایگاه‌های داده مانند [RavenDB](#) بر مبنای این تکنولوژی پایه گذاری شده اند.

هیچ چیزی نمی‌تواند مثل یک مثال؛ خوانایی ، سادگی و کم حجم بودن این روش را نشان دهد :

اگر یک شی با ساختار زیر در سی شارپ داشته باشید :

```
class Customer
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
}
```

ساختار JSON متناظر با آن (در صورت این که مقدار دهی شده باشد) به صورت زیر است:

```
{
  "Id":1,
  "FirstName":"John",
  "LastName":"Doe"
}
```

و در یک مثال پیچیده‌تر :

```
class Customer
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public Car Car { get; set; }
    public IEnumerable<Location> Locations { get; set; }
}

class Location
{
    public int Id { get; set; }
    public string Address { get; set; }
    public int Zip { get; set; }
}

class Car
{
    public int Id { get; set; }
    public string Model { get; set; }
}
```

```
{
  "Id":1,
  "FirstName":"John",
  "LastName":"Doe",
  "Car": {
    "Id":1,
    "Model":"Nissan GT-R"
  },
  "Locations":[
```

```
{
  {
    "Id":1,
    "Address":"30 Mortensen Avenue, Salinas",
    "Zip":93905
  },
  {
    "Id":2,
    "Address":"65 West Alisal Street, #210, Salinas",
    "Zip":95812
  }
}
```

ساختار JSON را مجموعه ای از (نام - مقدار) تشکیل می‌دهد. ساختار مشابه آن در زبان سی شارپ [KeyValuePair](#) است.

مشاهده [این تصاویر](#) ، بهترین درک را از ساختار JSON به شما می‌دهد.

[Json.net](#) یکی از بهترین کتابخانه هایی است که برای کار با این تکنولوژی در .net ارائه شده است. بهترین روش اضافه نمودن آن به پروژه [NuGet](#) است. برای این کار دستور زیر را در Package Manager Console وارد کنید.

```
PM> Install-Package Newtonsoft.Json
```

با استفاده از کد زیر می‌توانید یک Object را به فرمت JSON تبدیل کنید.

```
var customer = new Customer
{
    Id = 1,
    FirstName = "John",
    LastName = "Doe",
    Car = new Car
    {
        Id = 1,
        Model = "Nissan GT-R"
    },
    Locations = new[]
    {
        new Location
        {
            Id = 1,
            Address = "30 Mortensen Avenue,
Salinas",
            Zip = 93905
        },
        new Location
        {
            Id = 2,
            Address = "65 West Alisal Street, #210,
Salinas",
            Zip = 95812
        }
    }
};

var data = Newtonsoft.Json.JsonConvert.SerializeObject(customer);
```

خروجی تابع `SerializeObject` رشته ای است که محتوی آن را در چهارمین بلاک کد که در بالاتر آمده است، می‌توانید مشاهده کنید.

برای `Deserialize` کردن (Cast اطلاعات با فرمت JSON به کلاس موردنظر) از روش زیر بهره می‌گیریم :

```
var customer = Newtonsoft.Json.JsonConvert.DeserializeObject<Customer>(data);
```

آشنایی با این تکنولوژی، پیش درآمدی برای چشیدن طعم NoSQL و معرفی کارآمدترین روش‌های آن است که در آینده خواهیم آموخت...
خوشحال می‌شوم اگر نظرات شما را در باره این موضوع بدانم.

نظرات خوانندگان

نویسنده: احمدعلی شفیعی
تاریخ: ۱۵:۳۳ ۱۳۹۱/۰۴/۱۰

خیلی ممنون. واقعا کاربردی بود.

نویسنده: مهدی پایروند
تاریخ: ۱۶:۰۰ ۱۳۹۱/۰۴/۱۰

بهترین کتابخانه برای سریالایز کردن دیتا به سمت کلاینت است. و هم سرعتی تقریبا 8 برابر سرعت
System.Web.Script.Serialization.JavaScriptSerializer دارد

نویسنده: سروش ترک زاده
تاریخ: ۱۷:۴۸ ۱۳۹۱/۰۴/۱۰

در تصدیق حرف شما، دیدن [این مقایسه](#) خالی از لطف نیست. (پایین صفحه : Performance Comparison)

نویسنده: ssm
تاریخ: ۲۰:۱۷ ۱۳۹۱/۰۴/۱۰

با سلام
بسیار عالی بود مشتاقانه منتظر مطالب بعدی شما هستم
موفق باشید

نویسنده: مهدی ترابی
تاریخ: ۲۳:۰۴ ۱۳۹۱/۰۴/۱۰

بسیار خوب.
لطفن JavaScript Object Notation تصحیح شود به JavaScript Object Notation

نویسنده: سروش ترک زاده
تاریخ: ۱۱:۴۶ ۱۳۹۱/۰۴/۱۱

ممنون از دقت نظر شما. تصحیح شد.

نویسنده: رضا.ب
تاریخ: ۳:۰۷ ۱۳۹۱/۰۴/۱۲

با درود،

علت کمرنگ شدن نقش XML (اگر کمرنگ شده؟) بخاطر همین سادگی و سبکی JSON هست؟

نویسنده: سروش ترک زاده
تاریخ: ۱۰:۳۳ ۱۳۹۱/۰۴/۱۲

سلام

JSON رقیبی قوی برای XML محسوب می‌شه، اما به نظر هنوز زود هست که بگیم XML در حال حذف شدن هست. چون هنوز سیستم‌های قدرتمندی مثل Microsoft Workflow Generator یا همین RSS که زیاد در طول روز با اون سروکار داریم، بر مبنای این تکنولوژی کار می‌کنند.

نویسنده: رضا

تاریخ: ۱۳۹۳/۰۳/۲۶ ۹:۳۵

با سلام و تبریک به خاطر مقاله مفیدتون
یه سوال برام پیش اومده ، سرعت عمل و کارایی JSON بهتره یا XML ?
برای انجام یه برنامه لازم هست که بدونم از کدوم بهتره استفاده کنم.
ممنون و متشکر

نویسنده: سروش ترک زاده

تاریخ: ۱۳۹۳/۰۳/۲۶ ۱۱:۵۲

سلام
سرعت عمل، به ابزاری بستگی دارد که به وسیله آن اطلاعات serialize و deserialize می‌شود.
بهترین ابزاری که برای کار با XML معرفی شده است،(البته تا جایی که من خبر دارم) LinqToXML است. کار کردن با آن ساده است
اما دردسرهای خاص خودش رو داره.
از طرفی فرمت JSON نسبت به XML، حجم کمتری دارد (حداقل به این دلیل که نیاز به باز و بسته کردن tag نیست)
در مجموع من JSON رو پیشنهاد می‌کنم.

به شما خواننده گرامی پیشنهاد می‌کنم مطلب قبلی "[آشنایی با JSON؛ ساده - خوانا - کم حجم](#)" که پیش درآمدی بر این موضوع است را مطالعه کنید.

[NoSQL](#) یک مفهوم عام است و تعریف ساده آن "پایگاه داده بدون SQL است". به این معنی که در آن خبری از جدول ها، روابط بین آن‌ها و ... نیست!

اما چرا باید با وجود اینکه SQL به اغلب نیازهای ما پاسخ داده است، باید سراغ تکنولوژی‌های دیگر رفت؟

وقتی نگاهی به لیست شرکت‌های بزرگی می‌اندازیم که جز مشتریان پر و پا قرص NoSQL هستند ([+](#) و [+](#))، تعجب می‌کنیم! آیا آن‌ها از قدرت و قابلیت‌های SQL بی‌خبراند؟

پاسخ این گونه از سوال‌ها به تحلیل سیستم مربوط می‌شود. به عهده تحلیل گر است تا با توجه به اجزاء سیستم و ارتباط آن‌ها بهترین روش را برای ذخیره سازی اطلاعات انتخاب کند.

NoSQL بر اساس نحوه پیاده سازی اش دسته بندی شده است؛ که مهم‌ترین آن‌ها در زیر آمده است :
Wide Column Store

Document Store

Key Value / Tuple Store

Graph Databases

Multimodel Databases

Object Databases

برای آشنایی بهتر با هر کدام به nosql-database.org مراجعه کنید.

انتخاب روش؛ یک مثال ساده :

فرض کنید روال استخدام نیروی کار جدید در یک سازمان، از قرار زیر باشد:

ثبت مشخصات فردی

ارائه مدارک تحصیلی

شرکت در آزمون استخدامی

شرکت در مصاحبه (در صورت قبول شدن در آزمون)

شرکت در دوره آموزشی (در صورت قبول شدن در مصاحبه)

روش‌های ممکن برای نگهداری اطلاعات :

روش اول، تهیه پوشه‌هایی برای نگهداری اطلاعات مربوط به هر مرحله به صورت مجزا است.



روش دوم، تهیه یک پرونده برای هر شخص و نگهداری اسناد مربوط به شخص (در هر مرحله) است.



انتخاب روش اول امکان پذیر است، اما باعث پیچیده تر شدن سیستم و اتلاف زمان می شود که مطلوب نیست. برای پیاده سازی روش دوم، SQL پاسخ گوی نیاز پروژه نیست و با توجه به نیاز پروژه بهترین روش نگهداری اطلاعات، Document Store (نگهداری اطلاعات بر اساس ساختار اسناد) است. خوش بختانه تعداد پایگاه های داده ای که بر اساس تکنولوژی Document Store پیاده سازی شده اند، زیاد است و از قدرتمندترین آن ها می توان به [CouchDB](#)، [MongoDB](#) و [RavenDB](#) اشاره کرد. هرکدام از این انتخاب ها مزایا و معایبی دارند که باید با توجه به نیاز خود، [مقایسه ای](#) انجام داده و بهترین را انتخاب کنید. انتخاب من RavenDB بوده است و دلایل آن :

بر اساس زبان سی شارپ نوشته شده است و همچنین با LINQ خیلی خوب کار می کند.

Transaction را پشتیبانی می کند.

اساس ذخیره سازی آن JSON است.

محیط Management Studio کاربر پسندی دارد.

نقطه آغازین بحث بعد RavenDB خواهد بود که *Bryan Wheeler* (مدیر توسعه بسترهای نرم افزاری در [msn](#)) در باره آن گفته :

RavenDB just rocked my world. It's extremely approachable, even for non-database guys – it took me less than 30 minutes to get up and running

خوشحال می شوم، نظرات و تجربیات شما را در رابطه با NoSQL بدانم.

نظرات خوانندگان

نویسنده: RaminMjjz
تاریخ: ۱۸:۵۴ ۱۳۹۱/۰۴/۱۱

سلام.
ابتدا تشکر میکنم از مطلبی که دارید ارائه میدهید.
شیوه نگارشتون هم بسیار خوبه.
فقط یک پیشنهاد دارم. اونهم اینه که یک مطلب اختصاص بدهید به؛ در چه پروژه‌هایی باید از NoSQL استفاده کرد و چه پروژه‌هایی نباید.

نویسنده: mze666
تاریخ: ۱۹:۰۰ ۱۳۹۱/۰۴/۱۱

سلام - خیلی ممنون بابت مطلب خوبتون. فقط اگر براتون ممکنه یه آموزش گام به گام یا یه نمونه پروژه از RavenDB که به نظرم بهترین هستش رو بذارید. ممنون

نویسنده: مجتبی چنانی
تاریخ: ۱۹:۲۰ ۱۳۹۱/۰۴/۱۱

با سلام
من به عنوان کسی که در پروژه‌های خود از انواع ذخیره سازی‌ها بر اساس نیاز استفاده کردم(سرعت! راحتی! پلتفرم‌ها! ...) هم نظر میدم و هم پاسخ شما دوست عزیز را می‌دم.
قطعا انتخاب اینکه از چه روشی برای ذخیره سازی داده‌ها استفاده شود بسته به تیم پیاده سازکننده پروژه و نیز طراحان و ... دارد.
من با یک مثال توضیحی را خدمت شما می‌دهم.

در یک پروژه که اخیرا در حال اجرا هست(در دست من و هم تیمی‌های من) این پروژه یک پروژه بزرگ و با دیدها و اهداف وسیعی هست. ما در این برنامه هم از ادرس دهی بر اساس پوشه‌ها و دایرکتوری‌ها داده‌ها را ذخیره کردیم(اطلاعاتی مانند لینک فایل‌ها و یا تصاویر و...) و حتی در بعضی محل‌ها نیاز بود که اطلاعات یک فرد را در یک فایل xml قرار می‌دادیم و بعضی وقتها هم در پایگاه داده و هم فایل xml به این دلیل که در مورد اول تنها برنامه سمت کلاینت نیاز به این اطلاعات داشت و در آنجا پارسر قوی xml وجود داشت اما در مورد دوم ما به یک سری دیتا نیاز داشتیم که هم در سرور به آنها نیاز داریم و هم کلاینت! خب در بحث وب ما به مدیران اگر می‌خواستیم xml ارائه کنیم قطعا راه حل خوبی نبود و از سرعت و کارایی ما کم می‌کرد لذا از پایگاه داده استفاده کردم ولی برای زمانی که کاربر کلاینتی ما نیاز به اطلاعات داشت به این دلیل که بار سرور زیاد نشود از xml‌ها استفاده می‌شد که با یک لینک مستقیم می‌توانست به دست آورد(البته خود لینک همین فایل xml هم ساخته می‌شد! هیچ جا ذخیره نمی‌شد!)

عذر می‌خوام اگر بجای نویسنده پاسخ دادم البته این پاسخ من خیلی سربسته بود و انشا.. مفید بوده.

از نویسنده مطلب بابت مطلب خوبشون که کم دیدم در تارنماهای فارسی به اون بپردازن(متأسفانه بسیاری از اساتید دانشگاهی با این مفهوم حتی آشنایی ندارند با اینکه دانستن کلیت ان یک تعریف ساده است!) موفق باشید.

نویسنده: سروش ترک زاده
تاریخ: ۱۹:۲۷ ۱۳۹۱/۰۴/۱۱

از شما ممنونم به خاطر پیشنهاد خوبتون.

به نظر خودم موضوعی که شما مطرح کردید جای بحث بیشتری دارد و حتما این مورد رو برای نوشته‌های آینده مد نظر قرار خواهیم داد.

نویسنده: سروش ترک زاده
تاریخ: ۱۳۹۱/۰۴/۱۱ ۱۹:۳۰

ممنون از شما که این مثال رو مطرح کردید.
در نوشته‌های من جای یک مثال برای واضح شدن بیشتر موضوع خالی بود!

نویسنده: سروش ترک زاده
تاریخ: ۱۳۹۱/۰۴/۱۱ ۱۹:۳۶

موافقم، هیچ چیز مثل یک مثال کاربردی یادگرفتنی نیست...

نویسنده: امیرحسین جلوداری
تاریخ: ۱۳۹۱/۰۴/۱۱ ۱۹:۴۰

یه مشکلی داره RavenDb ! ... اونم اینکه مجوزش از هموناس که اگه پروژه تجاری باشه باس پولشو بدی! (اگه متن باز که باشه هیچ!)

نویسنده: mze666
تاریخ: ۱۳۹۱/۰۴/۱۱ ۱۹:۴۷

بله درسته ولی اگه بخوایم حساب کنیم ما از خیلی چیزا توی برنامه‌های تجاریمون استفاده میکنیم (Telerik, Stimulsoft, ...) ولی پولشو نمیدیم. اینم روش. (البته نمیگم کار خوبی میکنیم!)

نویسنده: RaminMjj
تاریخ: ۱۳۹۱/۰۴/۱۱ ۲۲:۱۱

با تشکر از جوابی که دادید.
ولی من میخوام مطلبی مشابه این مقاله ارائه بشه تا بیشتر با NoSQL آشنا بشیم
<http://www.dbta.com/Articles/Editorial/Trends-and-Applications/SQL-or-NoSQL-How-to-Choose-the-Right-Database-for-Your-Application-71240.aspx>

نویسنده: peyman
تاریخ: ۱۳۹۱/۰۴/۱۱ ۲۲:۲۶

فکر میکنم Neo4j هم عالی باشه ! گر چه مایکروسافت هم داره روی Trinity کار میکنه که هر دو از نوع گراف دیتابیس‌ها هستن و به نظر من کار با گراف دیتابیس‌ها خیلی زیاتر و لذتبخش‌تر هست

نویسنده: محمد
تاریخ: ۱۳۹۱/۰۴/۱۱ ۲۲:۵۷

اتلاف زمان صحیح است و نه «اتلاف زمان». اتلاف از تلف کردن میاد. ممنون به هر حال.

نویسنده: ghafoori
تاریخ: ۱۳۹۱/۰۴/۱۱ ۲۳:۱۵

اگر برنامه داخل ایران باشه اینکارو می‌کنیم اما اگر بخواهیم اون را داخل سرور امریکا یا هر دیتاسنتری که به مجوزها گیر میده برنامه را داشته باشیم باید چکار کنیم اینجا مانگو خودش را بهتر نشون میده

نویسنده: نیما
تاریخ: ۱۳۹۱/۰۴/۱۲ ۱:۴۰

سلام دوست عزیز

ممنون از مطلبتون. در نگاه اول مطلب شما اینجوری به من القا کرد که اطلاعات مثلا سریالایز بشن حالا چه بصورت json یا xml. من رو پروژه ای کار کردم که اطلاعات بصورت xml بود و فرض کنید حجم این فایل های xml به حدود 10 کیلو بطور میانگین میرسید. گزارشگیری از این xml ها بسیار وقت گیر بود مخصوصا که اگر قرار بود group by یا اعمال دیگری رو انجام بدیم و خیلی اوقات به timeout میخورد که با عوض کردن این شیوه و قرار دادن اطلاعات در جداول مختلف مشکلات بکلی حل شد. ممنون میشم بیشتر توضیح بدین. موفق باشید

نویسنده: رضا.ب
تاریخ: ۱۳۹۱/۰۴/۱۲ ۲:۱۴

دوست عزیز، لطفا بیشتر توضیح دهید. من چند بار کامنتتون رو خوندم متوجه نشدم چی میگین. ممنون.

نویسنده: رضا.ب
تاریخ: ۱۳۹۱/۰۴/۱۲ ۳:۱۱

دو سوال داشتم:
- امکان انتقال (Migrate) بین یه دیتابیس relational و nosql در عمل ممکن هست؟ (منظورم تبدیل رابطه ای ها به nosql هست. چون برعکسش محاله ظاهرا؟)
- نقش ORM ها در برقراری ارتباط Object ی و منطق برنامه های شی گراء با این نوع دیتابیسی های براساس سند (بدون ساختار) کجاست؟ اصلا ORM معنی میده هنگام کار با NoSQL؟
با تشکر. ممنونم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۱۲ ۸:۴۰

- در ravendb امکان replication به sql server وجود دارد.
- یکی از اهداف مهم ORM ها در دات نت، نوشتن کوئری های strongly typed است. در ravendb شما از روز اول با کوئری های strongly typed سروکار دارید. همچنین از همان ابتدای کار هم با کلاس های دات نت و نگاشت خودکار آن ها کار می کنید. کلا ravendb بر مبنای معماری و همچنین توانمندی و پیشرفت های زبان های دات نت تهیه شده.

نویسنده: mze666
تاریخ: ۱۳۹۱/۰۴/۱۲ ۹:۴۴

سلام آقای نصیری - میخوامم از شما یا آقای ترک زاده خواهش کنم که یه مورد از این ها (RavenDB, CouchDB, ...) که به نظر خودتون خوبه رو آموزش بدید.
یه آموزش اساسی مثل MVC یا Code First که تو چند جلسه تمام مباحثش رو گفتید.
ممنون.

نویسنده: سروش ترک زاده
تاریخ: ۱۳۹۱/۰۴/۱۲ ۱۰:۳۹

ما توی مکتب این جواری گفته بودن بهمون...
ممنون که تذکر دادین. اصلاح شد :-)

نویسنده: سروش ترک زاده
تاریخ: ۱۰:۵۶ ۱۳۹۱/۰۴/۱۲

ممنون از جناب آقای نصیری که پاسخشان در رابطه با ORM کامل و کافی بود.
اما در مورد سوال اول شما :
در بعضی موارد تبدیل پایگاه داده Table-Relational به بعضی موارد مثل Document Store کاملاً امکان پذیر است؛ اما تبدیل آن به نوع KeyValue اساساً معنی ندارد، زیرا کاربرد این دو روش کاملاً متفاوت است.
اما این نکته قابل توجه است که اگر تحلیل سیستم شما بر اساس Table-Relational انجام گرفته باشد؛ بعد از تبدیل به Document-Store، با کاهش سرعت مواجه می‌شوید.
و به نظر من زمانی باید سراغ روش‌های NoSQL رفت که ساختار Table-Relational پاسخ مناسبی برای نیاز ما نباشد.

نویسنده: سروش ترک زاده
تاریخ: ۱۱:۱۳ ۱۳۹۱/۰۴/۱۲

سلام
نظر شما تا حدودی صحیح است اما کلاس‌های دات نت مثل [XMLWriter](#) , [XDocument](#) و ... قابل مقایسه با Engine قدرتمندی که برای یک پایگاه داده نوشته می‌شود، نیستند.
همچنین یکی از نیازها که باعث می‌شود سراغ NoSQL برویم، حجم عظیم اطلاعات است.
پس هیچ نگرانی در مورد حجم اطلاعات نباید وجود داشته باشد...

نویسنده: رضا ب.
تاریخ: ۱۵:۳۷ ۱۳۹۱/۰۴/۱۲

خب یعنی برای رفتن سمت هر NoSQL باید دلیل مرجعی داشته باشیم. و بخاطر جدید بودن و استفاده سازمان‌های عظیم از آنها و یا حتی آسان‌تر بودن، دلیل نمیشود که پایگاه‌داده‌ای رابطه‌ای رو رها کنیم.
و این زمانی اتفاق می‌وفته که این 6 نوعی که ذکر کردید، رو کاملاً بشناسیم. مزایا معایب و موارد کاربرد اون رو بدونیم و با اثبات ردِ کارایی مطلوب دیتابیس‌های رابطه‌ای به انتخاب NoSQLی دست بگذاریم.
در مورد کامنتتون متوجه نشدم علت اینکه یه پایگاه داده‌ی رابطه‌ای چرا نمیتونه به جفت مقدار/کلید تبدیل بشه؟ شاید بلعکس‌اش محال باشه. مثلاً وراثت یا جدول‌های با ستون‌های پویا و ... اصلاً در پایگاه‌های رابطه‌ای بی‌معنی هستند.

نویسنده: سروش ترک زاده
تاریخ: ۱۵:۵ ۱۳۹۱/۰۴/۱۳

شاید من نتوانستم منظور خودم رو واضح بگم؛
Table-Relational و NoSQL نقطه مقابل هم نیستند و انتخاب شما بین یکی از روش‌های ذخیره کردن اطلاعات (Graph Databases , Table Relational , Object Databases ، و ...) مشابه مثال انتخاب یکی از Type هایی مثل bit ، TimeSpam ، long و ... برای ذخیره کردن یک مقدار کوچک است. درست است که همه این کارها را با string می‌توان انجام داد و لی می‌توان با انتخاب درست در سرعت و فضایی که قرار است مصرف شود، صرفه جویی کرد.

و در باره مورد بعد که مطرح کردید، شاید یک مثال ساده قضیه رو روشن‌تر کند؛ می‌شود یک عدد کوچک رو در متغیری از جنس TimeSpam ریخت، اما اگر این عدد به معنی زمان نباشد، روش ما بهینه و حتی درست نیست، اما کار انجام شده است...
در صورتی که می‌شود این مقدار را در یک متغیر از جنس int ذخیره کرد.

امیدوارم شبهه ای که برای شما ایجاد شده است، با ارائه یک مثال کاربردی از RavenDB که در پست بعدی خواهم گفت، برطرف شود...

نویسنده: محمد صاحب
تاریخ: ۱۲:۴۰ ۱۳۹۱/۰۴/۱۴

تا آماده شدن مثال کاربردی؛ دیدن این [پست](#) خالی از لطف نیست.

نویسنده: محسن
تاریخ: ۱۳۹۱/۱۰/۲۳ ۱۲:۱۰

سلام

از RavenDB راضی بودین؟ آیا واقعا از جستجوی Full-Text بهره مند است و تونسته Lucene رو خوب تعبیه کنه؟

نویسنده: سروش ترک زاده
تاریخ: ۱۳۹۱/۱۰/۲۵ ۱۳:۳۰

سلام

تا اندازه ای که کارکردم خوب بود، البته پیش نیومد که توی پروژه Enterprise از آن استفاده کنم و در مورد Full-Text , Lucene راستش تا حالا امتحان نکردم... شاید دوستان دیگر بتوانند راهنمایی کنند.

نویسنده: بازرگان
تاریخ: ۱۳۹۱/۱۱/۲۰ ۱۴:۴۴

گوگل پلاس و فیسبوک برای بانک اطلاعاتشون از چه شیوه هایی استفاده میکنند ؟

نویسنده: سعید
تاریخ: ۱۳۹۱/۱۱/۲۰ ۱۷:۲۱

گوگل از بانک اطلاعاتی ساخت خودش استفاده می‌کنه: [اطلاعات بیشتر](#) ، فیس بوک هم [در اینجا](#)

عنوان: RavenDB: تجربه متفاوت از پایگاه داده

نویسنده: سروش ترک زاده

تاریخ: ۱۹:۳۱ ۱۳۹۱/۰۴/۱۵


















آدرس: www.dotnettips.info

برچسب‌ها: C#, JSON, NoSQL, RavenDB

" به شما خواننده گرامی پیشنهاد می‌کنم [مطلب قبلی](#) را مطالعه کنید تا پیش زمینه مناسبی در باره این مطلب کسب کنید. "

ماهیت این پایگاه داده وب سرویسی مبتنی بر REST است و فرمت اطلاعاتی که از سرور دریافت می‌شود، [JSON](#) است.

گام اول: باید آخرین نسخه RavenDB را دریافت کنید. همان طور که مشاهده می‌کنید، ویرایش‌های مختلف کتابخانه‌هایی که برای نسخه Client و همچنین Server طراحی شده است، در این فایل قرار گرفته است.

Name	Date modified	Type	Size
 Backup	۲۰۱۲/۰۲/۰۶ ۰۴:۲۰ ...	File folder	
 Bundles	۲۰۱۲/۰۲/۰۶ ۰۴:۲۰ ...	File folder	
 Client	۲۰۱۲/۰۲/۰۶ ۰۴:۲۰ ...	File folder	
 Client-3.5	۲۰۱۲/۰۲/۰۶ ۰۴:۲۰ ...	File folder	
 EmbeddedClient	۲۰۱۲/۰۲/۰۶ ۰۴:۲۰ ...	File folder	
 Samples	۲۰۱۲/۰۲/۰۶ ۰۴:۲۰ ...	File folder	
 Server	۲۰۱۲/۰۲/۰۶ ۰۴:۲۰ ...	File folder	
 Silverlight	۲۰۱۲/۰۲/۰۶ ۰۴:۲۰ ...	File folder	
 Silverlight-4	۲۰۱۲/۰۲/۰۶ ۰۴:۲۰ ...	File folder	
 Smuggler	۲۰۱۲/۰۲/۰۶ ۰۴:۲۰ ...	File folder	
 Web	۲۰۱۲/۰۲/۰۶ ۰۴:۲۰ ...	File folder	
 acknowledgments	۲۰۱۲/۰۲/۰۶ ۰۴:۱۸ ...	Text Document	
 license	۲۰۱۲/۰۲/۰۶ ۰۴:۱۸ ...	Text Document	
 Raven-GetBundles	۲۰۱۲/۰۲/۰۶ ۰۴:۱۸ ...	PS1 File	
 Raven-UpdateBundles	۲۰۱۲/۰۲/۰۶ ۰۴:۱۸ ...	PS1 File	
 readme	۲۰۱۲/۰۲/۰۶ ۰۴:۱۸ ...	Text Document	
 Start	۲۰۱۲/۰۲/۰۶ ۰۴:۱۸ ...	Windows Comma...	

برای راه اندازی Server باید فایل Start را اجرا کنید، چند ثانیه بعد محیط مدیریتی آن را در مرورگر خود مشاهده می‌کنید. در بالای صفحه روی لینک Databases کلیک کنید و در صفحه باز شده گزینه New Database را انتخاب کنید. با دادن یک نام دلخواه حالا شما یک پایگاه داده ایجاد کرده اید. تا همین جا دست نگه دارید و اجازه دهید با این محیط دوست داشتنی و قابلیت‌های آن بعدا آشنا شویم.

در گام دوم به Visual Studio می‌رویم و نحوه ارتباط با پایگاه داده و استفاده از دستورات آن را فرا می‌گیریم.

گام دوم:

با یک پروژه Test شروع می‌کنیم که در هر گام تکمیل می‌شود و می‌توانید پروژه کامل را در پایان این پست دانلود کنید.

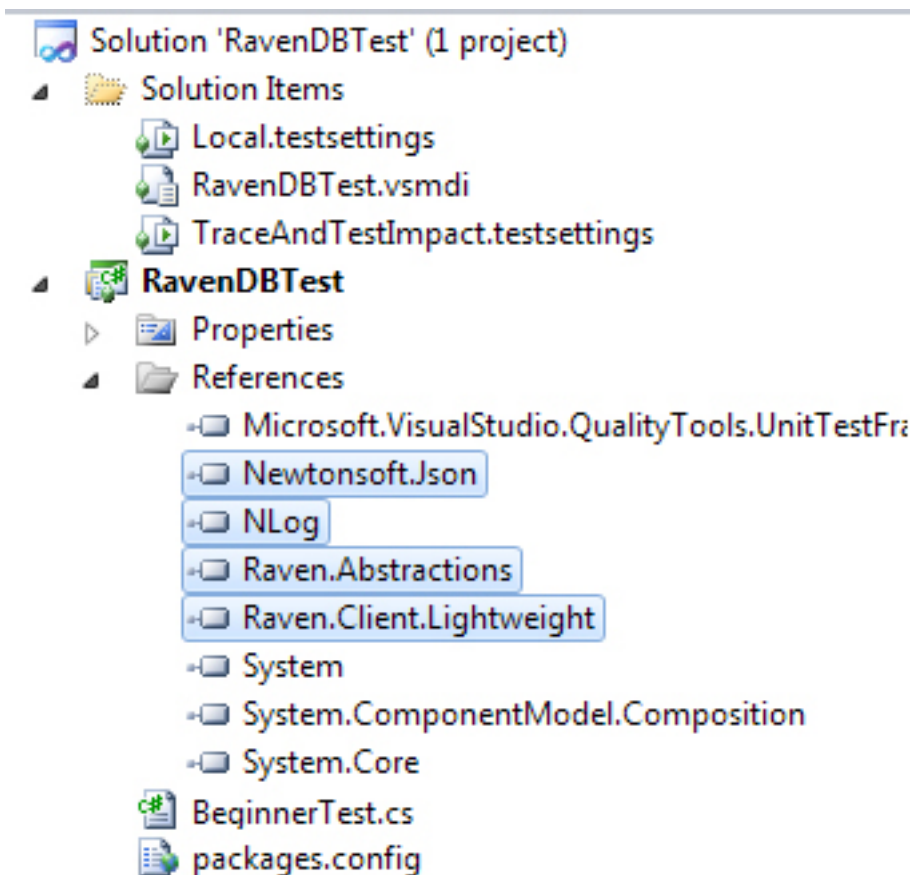
برای استفاده از کتابخانه‌های مورد نیاز دو راه وجود دارد:

استفاده از NuGet : با استفاده از دستور زیر Package مورد نیاز به پروژه شما افزوده می‌شود.

```
PM> Install-Package RavenDB -Version 1.0.919
```

اضافه کردن کتابخانه‌ها به صورت دستی : کتابخانه‌های مورد نیاز شما در همان فایل‌ها که دانلود شده بود و در پوشه Client قرار دارند.

کتابخانه‌هایی را که NuGet به پروژه من اضافه کرد، در تصویر زیر مشاهده می‌کنید :



با Newtonsoft.Json در اولین بخش بحث آشنا شدید. NLog هم یک کتابخانه قوی و مستقل برای مدیریت Log است که این پایگاه داده از آن بهره برده است.

" دلیل اینکه از پروژه تست استفاده کردم ؛ تمرکز روی کدها و مشاهده تاثیر آنها ، مستقل از UI و لایه‌های دیگر نرم افزار است. بدیهی است که استفاده از آنها در هر پروژه امکان پذیر است. "

برای شروع نیاز به آدرس Server و نام پایگاه داده داریم که می‌توانید در App.config به عنوان تنظیمات نرم افزار شما ذخیره شود و هنگام اجرای نرم افزار مقدار آن‌ها را خوانده و در متغیرهای readonly ذخیره شوند.

```
<appSettings>
  <add key="ServerName" value="http://SorousH-HP:8080/" />
  <add key="DatabaseName" value="TestDatabase" />
</appSettings>
```

هنگامی که صفحه Management Studio در مرورگر باز است، می‌توانید از نوار آدرس مرورگر خود آدرس سرور را به دست آورید.

```
[TestClass]
public class BeginnerTest
{
    private readonly string serverName;
    private readonly string databaseName;

    public BeginnerTest()
    {
        serverName = ConfigurationManager.AppSettings["ServerName"];
        databaseName = ConfigurationManager.AppSettings["DatabaseName"];
    }
}
```

برای برقراری ارتباط با پایگاه داده نیاز به یک شیء از جنس DocumentStore و جهت انجام عملیات مختلف (ذخیره، حذف و ...) نیاز به یک شیء از جنس IDocumentSession است. کد زیر، نحوه کار با آن‌ها را به شما نشان می‌دهد:

```
[TestClass]
public class BeginnerTest
{
    private readonly string serverName;
    private readonly string databaseName;

    private DocumentStore documentStore;
    private IDocumentSession session;

    public BeginnerTest()
    {
        serverName = ConfigurationManager.AppSettings["ServerName"];
        databaseName = ConfigurationManager.AppSettings["DatabaseName"];
    }

    [TestInitialize]
    public void TestStart()
    {
        documentStore = new DocumentStore { Url = serverName };
        documentStore.Initialize();
        session = documentStore.OpenSession(databaseName);
    }

    [TestCleanup]
    public void TestEnd()
    {
        session.SaveChanges();
        documentStore.Dispose();
        session.Dispose();
    }
}
```

در طراحی این پایگاه داده از الگوی Unit Of Work استفاده شده است. به این معنی که تمام تغییرات در حافظه ذخیره می‌شوند و به محض اجرای دستور session.SaveChanges() ارتباط برقرار شده و تمام تغییرات ذخیره خواهند شد.

هنگام شروع (تابع TestStart) متغیر session مقدار دهی می‌شود و در پایان کار (تابع TestEnd) تغییرات ذخیره شده و منابعی که توسط این دو شیء در حافظه استفاده شده است، رها می‌شود.

البته بر مبنای طراحی شما، دستور `session.SaveChanges();` می‌تواند پس از انجام هر عملیات اجرا شود.

برای آشنا شدن با نحوه ذخیره کردن اطلاعات، به کد زیر دقت کنید:

```
class User
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Address { get; set; }
    public int Zip { get; set; }
}

[TestMethod]
public void Insert()
{
    var user = new User
    {
        Id = 1,
        Name = "John Doe",
        Address = "no-address",
        Zip = 65826
    };
    session.Store(user);
}
```

اگر همه چیز درست پیش رفته باشد، وقتی به محیط RavenDB Studio که هنوز در مرورگر شما باز است، نگاهی می‌اندازید، یک سند جدید ایجاد شده است که با کلیک روی آن، اطلاعات آن قابل مشاهده است. لحظه‌ی لذت بخشی است... یکی از روش‌های خواندن اطلاعات هم به صورت زیر است:

```
[TestMethod]
public void Select()
{
    var user = session.Load<User>(1);
}
```

نتیجه خروجی این دستور هم یک شیء از جنس کلاس `User` است.

تا این جا، ساده‌ترین مثال‌های ممکن را مشاهده کردید و حتما در بحث بعد مثال‌های جالب‌تر و دقیق‌تری را بررسی می‌کنیم و همچنین نگاهی به جزئیات طراحی و قراردادهای از پیش تعیین شده می‌اندازیم.

" به شما پیشنهاد می‌کنم که منتظر بحث بعدی نباشید! همین حالا دست به کار شوید... "

نسخه بدون کتابخانه‌های موردنیاز (2 مگابایت) : [RavenDBTest_Small.zip](#)

نسخه کامل (15 مگابایت) : [RavenDBTest.zip](#)

نظرات خوانندگان

نویسنده: ناصر طاهری
تاریخ: ۱۷:۱۲ ۱۳۹۱/۰۴/۱۶

سلام.
مقوله‌ی جالبیه برای من.
منتظر ادامه هستم. موفق باشید.

نویسنده: حسین
تاریخ: ۱۹:۱۳ ۱۳۹۱/۰۴/۱۶

بسیار ممنون که کاربردی پیش میرین. من پروژمو با همین روش پیاده سازی می‌کنم و از اطلاعات خوبتون استفاده کردم.

نویسنده: ramín_rp
تاریخ: ۱۰:۴۳ ۱۳۹۱/۰۴/۱۷

سلام
وقتی raven db رو استارت میکنم و محیط مدیریت اون تو browser اجرا میشه برای انجام عملیاتی مثل ایجاد دیتابیس user,pass میخواد که هرچی میدم قبول نمیکنه
حتی raven رو به عنوان service هم نصب کردم ولی مشکل حل نشد
جستجو تو نت هم نتیجه نداد
مشکل از چیه؟

(مستندات رسمی raven db خوب نیست، مستندات mongodb واقعا کامل و جامع هست)

نویسنده: وحید نصیری
تاریخ: ۱۱:۳۷ ۱۳۹۱/۰۴/۱۷

توضیحات بیشتر [در اینجا](#)

By default RavenDB allow anonymous access only for read requests (HTTP GET), and since we creating data, we need to specify a username and password. You can control this by changing the AnonymousAccess setting in the server configuration file. Enter your username and password of your Windows account and a sample data will be generated for you.

نویسنده: سروش ترک زاده
تاریخ: ۲۱:۱۸ ۱۳۹۱/۰۴/۱۷

سلام
ببخشید که دیر به سوال شما پاسخ دادم...
یه راه دیگه، علاوه بر راهی که توسط جناب آقای نصیری ارائه شده است، وجود دارد.
در پوشه Server فایل Raven.Server.exe را با Notepad باز کنید، سپس مقدار تنظیمات با کلید "Raven/AnonymousAccess" را به "All" تغییر دهید. توجه کنید که به بزرگ و کوچک بودن حروف حساس است.

در ضمن RavenDB از نظر سابقه و تعداد کاربران، قابل مقایسه با پایگاه داده هایی مثل SQL نیست و حق با شماست...

نویسنده: صابر

تاریخ: ۱۱:۱۹ ۱۳۹۱/۰۴/۲۱

سلام

نمی‌دانم مشکل از چیه ؟ ولی وقتی من سعی می‌کنم که بسته‌ی RavenDB رو از طریق Nuget دریافت کنم Error زیر رو می‌ده .

```
Install-Package : The element 'metadata' in namespace
'http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd' has invalid child element
'frameworkAssemblies' in namespace
'http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd'. List of possible elements expected:
'summary' in namespace
'http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd'.
At line:1 char:1
+ Install-Package RavenDB -Version 1.0.919
+ ~~~~~
+ CategoryInfo          : NotSpecified: (:) [Install-Package], InvalidOperationException
+ FullyQualifiedErrorId : NuGet.VisualStudio.Cmdlets.InstallPackageCmdlet
```

نویسنده: وحید نصیری

تاریخ: ۱۱:۵۸ ۱۳۹۱/۰۴/۲۱

به احتمال زیاد VS.NET شما دسترسی به اینترنت ندارد ([^](#) و [^](#)).

نویسنده: peyman

تاریخ: ۹:۱۸ ۱۳۹۱/۰۵/۰۸

آقا سروش کی شروع میکنی سری جدید رو منتظریم قربان !

نویسنده: سروش ترک زاده

تاریخ: ۱۸:۵۹ ۱۳۹۱/۰۵/۰۸

سلام

ببخشید که دیر شد، به احتمال زیاد پنجشنبه ادامه آموزش را روی سایت قرار خواهیم داد...

نویسنده: پژمان

تاریخ: ۲۳:۲۵ ۱۳۹۱/۰۶/۰۱

ravendb هم مثل اینکه برای جستجو از لوسین استفاده می‌کنه.

نویسنده: فرزاد

تاریخ: ۲۳:۵۰ ۱۳۹۱/۰۷/۱۰

سلام

می‌خواهم یه نرم افزار تحت ویندوز بنویسم که نیاز دارم از بانک اطلاعاتی استفاده کنم از طرفی هم نمی‌خواهم با Sql server و یا access کار کنم چون نیاز به نرم افزار هایی با حجم زیاد هست که برای کاربر دردسر میشه. می‌خواستم اگه میشه بفرمایید که به نظرتون از چی استفاده کنم بهتره؟

ممنون

نویسنده: حسین مرادی نیا

تاریخ: ۵:۴۸ ۱۳۹۱/۰۷/۱۱

نکته اینکه وقتی بانک اطلاعات Access رو استفاده کنین ، حتما نیازی نیست که Access روی کامپیوتر کاربر نصب باشه تا بتونه از برنامه شما استفاده کنه.

به هر حال میتونید از Sql Server CE استفاده کنید: <http://www.dotnettips.info/search/label/SQL%20Server%20CE>

عنوان: مدیریت کوکی ها با jQuery

نویسنده: امیرحسین مرجانی

تاریخ: ۳:۵ ۱۳۹۱/۰۶/۱۲

آدرس: www.dotnettips.info

برچسب‌ها: JSON, jQuery, jQuery-Tips

در گذشته نه چندان دور، کوکی‌ها نقش اصلی را در مدیریت کاربران، و ذخیره اطلاعات کاربران ایفا می‌کردند. ولی بعد از کشف شدن باگ امنیتی (که ناشی از اشتباه برنامه نویسی بود) در کوکی‌ها، برای مدتی کنار گذاشته شدند و اکثر اطلاعات کاربران در session های سمت سرور ذخیره می‌شد. ذخیره اطلاعات زیاد و نه چندان مهم کاربران در session های سمت سرور، بار زیادی را به سخت افزار تحمیل می‌کرد. بعد از این، برنامه نویسان به سمتی استفاده متعادل از هرکدام این‌ها (کوکی و سشن) رفتند. اکثر دوستان با مدیریت سمت سرور کوکی‌ها آشنایی دارند، بنده قصد دارم در اینجا با استفاده از یک پلاگین جی کوئری مدیریت کوکی‌ها را نمایش دهم. در این برنامه ما از پلاگی jquery.cookie استفاده می‌کنیم که شما می‌توانید با مراجعه به [صفحه این پلاگین](#) اطلاعات کاملی از این پلاگین به دست بیاورید. کار با این پلاگین بسیار ساده است. ابتدا فایل پلاگین را به صفحه خودتون اضافه می‌کنید.

```
<script src="/path/to/jquery.cookie.js"></script>
```

حالا خیلی راحت می‌توانید با این دستور یک مقدار را در کوکی قرار دهید.

```
$.cookie('the_cookie', 'the_value');
```

و برای گرفتن کوکی نوشته شده هم به این صورت عمل می‌کنید.

```
$.cookie('the_cookie'); // => "the_value"
```

همان طور که دیدید کار بسیار ساده ای است. ولی قدرت این پلاگین در option هایی است که در اختیار ما قرار می‌دهد. مثلا شما می‌توانید انتخاب کنید این کوکی برای چند روز معتبر باشد، و یا اطلاعات را به صورت json ذخیره و بازیابی کنید، و حتی option های دیگری برای بحث امنیت کوکی شما. برای درک بهتر از قطعه کدی که کمی پیچیده‌تر از مثال منبع است، استفاده می‌کنیم.

به کد زیر توجه کنید :

: JavaScript

```
<script type="text/javascript">
$(function () {
    $('#write').click(function () {
        $.cookie('data', '{"iri":"Iran","usa":"United States"}', { expires: 365, json: true });
        alert('Writed');
    });

    $('#show').click(function () {
        var obj = jQuery.parseJSON($.cookie('data'));
        alert(obj.iri);
    });

    $('#remove').click(function () {
        $.removeCookie('data');
    });
})
</script>
```

: HTML

```
<body>
  <a href="#" id="write">Write</a>
  <br />
  <a href="#" id="show">Show</a>
  <br />
  <a href="#" id="remove">Remove</a>
</body>
```

در اینجا ما سه لینک داریم که هر کدام برای ما عملی را نمایش میدهند.

توضیحات کد :

```
$('#write').click(function () {
    $.cookie('data', '{"iri":"Iran","usa":"United States"}', { expires: 365, json: true });
    alert('Writed');
});
```

با کلیک بر روی لینک Write کوکی data با مقدار مشخص پر می‌شود.

دقت داشته باشید که این مقدار از نوع json انتخاب شده است و در انتها نیز این را مشخص کرده ایم ، همچنین اعلام کرده ایم که این کوکی برای 365 روز معتبر است.

[حالا مرورگر خودتان را ببندید و دوباره باز کنید.](#)

این بار بر روی Show کلیک می‌کنیم :

```
$('#show').click(function () {
    var obj = jQuery.parseJSON($.cookie('data'));
    alert(obj.iri);
});
```

با کلیک بر روی لینک Show مقدار از کوکی خوانده می‌شود و نمایش داده می‌شود. دقت کنید ، به دلیل اینکه مقدار ذخیره شده ما از نوع json است باید دوباره این مقدار را pars کنیم تا به مقادیر property آن دسترسی داشته باشیم.

همچنین شما می‌توانید خیلی راحت کوکی ساخته شده را از بین ببرید :

```
$('#remove').click(function () {
    $.removeCookie('data');
});
```

و یا این که کوکی را برابر null قرار دهید.

نکته ای که باید رعایت کنید و در این مثال هم نیامده است ، این است که ، هنگامی که شما می‌خواهید object ی که با کد تولید کرده اید در کوکی قرار بدهید ، باید از متد JSON.stringify استفاده کنید و مقدار را به این صورت در کوکی قرار دهید.

```
$.cookie('data', JSON.stringify(jsonobject), { expires: 365, json: true });
```

که در اینجا jsonobject ، ابجکتی است که شما تولید کرده اید و قصد ذخیره آن را دارید.

من از این امکان در نسخه بعدی [این پروژه](#) استفاده کرده ام ، و به کمک این پلاگین ساده اما مفید ، وب سایت هایی که کاربر نتایج آن را مشاهده کرده است در کوکی کاربر ذخیره می‌کنم تا در مراجعه بعدی میزان تغییرات رنکینگ‌های وب سایت ای در خواست شده را ، به کاربر نمایش دهم. نسخه بعد [all-ranks.com](#) تا آخر هفته آینده در سرور اختصاصی (و نه این هاست رایگان (!)) قرار می‌گیرد و به مرور قسمت هایی که در این پروژه پیاده سازی شده (پلاگین‌های جی کوئری و کدهای سرور) در اینجا شرح می‌دهم. امیدوارم تونسته باشم مطلب مفید و مناسبی به شما دوستان عزیزم انتقال بدم.

نظرات خوانندگان

نویسنده: امیرحسین جلوداری
تاریخ: ۱۱:۵۵ ۱۳۹۱/۰۶/۱۲

خیلی ممنون ... این سبک مطلبارو دوست دارم (:

نویسنده: امیرحسین مرجانی
تاریخ: ۱۲:۲ ۱۳۹۱/۰۶/۱۲

پس در این زمینه هم عقیده ایم (:
موفق باشید.

در مطلب قبلی با استفاده از دستور For XML خروجی xml تولید کردیم اما با همین دستور می‌توان تا حدودی خروجی Json نیز تولید نمود. البته به صورت native هنوز در sql server این امکان وجود ندارد که با رای دادن به این [لینک](#) از تیم ماکروسافت بخواهید که این امکان را در نسخه بعدی اضافه کند.

برای این کار یک جدول موقت ایجاد کرده و چند رکورد در آن درج می‌کنیم:

```
declare @t table(id int, name nvarchar(max), active bit)
insert @t values (1, 'Group 1', 1), (2, 'Group 2', 0)
```

حال با استفاده از همان for xml و پارامتر type که نوع خروجی xml را خودمان می‌توانیم تعیین نماییم و پارامتر Path این کار را بصورت زیر انجام می‌دهیم:

```
select '[' + STUFF((
    select
        '{ "id":' + cast(id as varchar(max))
        + ', "name":' + name + ', "active":' + cast(active as varchar(max))
        + '}'
    from @t t1
    for xml path(''), type
).value('.', 'varchar(max)'), 1, 1, '') + ']'
```

توجه کنید در این جا از پارامتر path بدون نام استفاده شده است و از تابع STUFF برای در یک رشته در رشته دیگر استفاده شده است. خروجی در زیر آورده شده است:

```
[{"id":1,"name":"Group 1","active":1},{"id":2,"name":"Group 2","active":0}]
```

حالت پیشرفته‌تر آن است که بتوانیم یک join را بصورت فرزندان آن در json نمایش دهیم قطعه کد زیر را مشاهده فرمایید:

```
declare @group table(id int, name nvarchar(max), active bit)
insert @group values (1, 'Group 1', 1), (2, 'Group 2', 0)

declare @member table(id int, groupid int, name nvarchar(max))
insert @member values (1, 1, 'Ali'), (2, 1, 'Mojtaba'), (3, 2, 'Hamid')

select '[' + STUFF((
    select
        '{ "id":' + cast(g.id as varchar(max))
        + ', "name":' + g.name + ', "members": { "children": [' +
        (select + STUFF((
            select
                '{ "id":' + cast(m.id as varchar(max))
                + ', "name":' + m.name + '}'
            from @member m
            where m.groupid = g.id
            for xml path(''), type
        ).value('.', 'varchar(max)'), 1, 1, '')
        + ']}]'
        + ', "active":' + cast(g.active as varchar(max))
        + '}'
    from @group g
    for xml path(''), type
).value('.', 'varchar(max)'), 1, 1, '') + ']'
```

خروجی json بصورت زیر است:

```
[{"id":1,"name":"Group 1","members":
  { "children": [{"id":1,"name":"Ali"}, {"id":2,"name":"Mojtaba"}]}
,"active":1},
{"id":2,"name":"Group 2","members":
  { "children": [{"id":3,"name":"Hamid"}]}
,"active":0}]
```

حالت‌های خاص و پیشرفته‌تر را با امکانات t-sql خودتان می‌توانید به همین شکل تولید نمایید.

در مطالب قبلی با پروتکل OData و WCF Data Service و قراردادهای کوثری نویسی آن آشنا شدید. حال می‌خواهیم با استفاده از JQuery به داده‌های وب سرویس WCF Data Service دسترسی یابیم. اما پیش نیازهای لازم است

پیش نیاز اول : دسترسی به خروجی Json وب سرویس WCF Data

خروجی پیش فرس وب سرویس WCF Data Services ساختار Xml دارد پس می‌بایست وب سرویس را متوجه سازیم که ما با خروجی Json نیاز داریم. از نسخه 5 به بعد اگر MaxProtocolVersion را بر روی V3 قرار دهیم دیگر با Accept Header برابر application/json کار نخواهد کرد و می‌بایست از application/json;odata=verbose استفاده نمود یا نسخه پروتکل را بر روی V2 یا پایین‌تر تنظیم کنید. علاوه بر آن کتابخانه‌های و قطعه کدهای تهیه شده است که با پارامتر \$format این کار را برای ما انجام می‌دهد در زیر آدرس دو نمونه آورده شده است.

[DataServicesJSONP](#)

[WCF Data Services Toolkit](#)

قطعه کد اول یک Attribute است که با اضافه کردن آن به بالای کلاس WebService و استفاده از پارامتر \$format=json در آدرس وب سرویس این کار را برای ما انجام می‌دهد.

```
[JSONPSupportBehavior]
public class Northwind : DataService<NorthwindEntities>
```

و نمونه آدرس

```
http://localhost:8358/Northwind.svc/Products?$format=json
```

دومی کتابخانه ای است که مانند روش اول عمل می‌کند اما به جای ارث برای از کلاس DataService می‌بایست از کلاس ODataService استفاده نماییم.

نکته: در صورتی که بخواهیم از نسخه V3 استفاده نماییم Accept Header را باید به application/json;odata=verbose تغییر دهیم

```
public class Northwind : ODataService<NorthwindEntities>
```

استفاده از WCF Data Services به کمک JQuery

تابع getJSON مخصوص درخواست‌های است خروجی بصورت json برگردانده می‌شود اما با نسخه V3 سازگار نمی‌باشد و از روش پارامتر \$format می‌توان استفاده نمود

```
$.getJSON("Northwind.svc/Products?$format=json", function (data) {
    $.each(data.d, function (i, item) {
        $("

```


همچنین از تابع Ajax که امکانات بیشتری را در اختیارمان قرار می دهد به راحتی می توان استفاده نمود به مثال زیر دقت کنید:

```
$.ajax('Northwind.svc/Products', {
    dataType: "json",
    beforeSend: function (xhr) {
        xhr.setRequestHeader("Accept", "application/json;odata=verbose");
        xhr.setRequestHeader("MaxDataServiceVersion", "3.0");
    },
    success: function (data) {
        $.each(data.d, function (i, item) {
            $("<p/>").html(item.Product_Name + " " +
item.Unit_Price).appendTo("#products");
        });
    }
});
```

با استفاده از beforeSend مقدار Accept Header و MaxDataServiceVersion را تعیین نموده ایم.
بنابراین به کمک قراردادهای کوئری نویسی که در مطالب قبلی گفته شد می توان با استفاده از Url تابع Ajax به داده مورد نظر خود رسید.

نظرات خوانندگان

نویسنده: مرادی

تاریخ: ۱۷:۴۲ ۱۳۹۱/۱۱/۰۵

با سلام، بهتر نیست از jay data یا از breeze.js استفاده کنید ؟

نویسنده: مجتبی کاویانی

تاریخ: ۱۸:۳۴ ۱۳۹۱/۱۱/۰۵

در مطلب بعدی به این دو اشاره خواهم کرد قطعا امکانات بیشتری در اختیارمان قرار می‌دهد

نویسنده: abdali

تاریخ: ۱۵:۱۹ ۱۳۹۲/۰۴/۱۱

لطف میکنید کد رو قرار بدین ، چند بار امتحان کردم با خطا مواجه شدم . مرسی

در برنامه‌های وب امروز نیازی به فراخوانی ثوابت که در طول حیات برنامه انگشت شمار تغییر میکنند نیست و با توجه به استفاده از فرامین و متدهای سمت کلاینت احتیاج هست تا این ثوابت بار اول لود صفحه به کلاینت پاس داده شوند.

میتوان در این گونه موارد از قابلیت‌های گوناگونی استفاده کرد که در اینجا ما با استفاده از یک فیلد مخفی و json مقدار را به کلاینت پاس میدهیم و در این مثال در سمت کلاینت نیز دراپ دان را با این مقادیر پر میکنیم:

```
public enum PersistType
{
    Persistable = 1,
    NotPersist = 2,
    AlwaysPersist = 3
}
```

لیست را باید قبل از پر کردن در فیلد مخفی به json بصورت serialize شده تبدیل کرد، برای این منظور از JavaScriptSerializer موجود در اسمبلی‌های دات نت در متد زیر استفاده شده:

```
public static string ConvertEnumToJavascript(Type t)
{
    if (!t.IsEnum) throw new Exception("Type must be an enumeration");
    var values = System.Enum.GetValues(t);
    var dict = new Dictionary<int, string>();
    foreach (object obj in values)
    {
        string name = System.Enum.GetName(t, obj);
        dict.Add(Convert.ToInt32(System.Enum.Format(t, obj, "D")), name);
    }
    return new JavaScriptSerializer().Serialize(dict);
}
```

با توجه به اینکه در سمت کلاینت مقدار json ذخیره شده در فیلد مخفی را میتوان به صورت آبجکت برخورد کرد پس یک متد در سمت کلاینت این آبجکت را در loop قرار داده و درمتغیری در فایل جاوا اسکریپت نگهداری میکنیم:

```
var Enum_PersistType = null;
function SetEnumTypes() {
    Enum_PersistType = JSON.parse($('#hfJsonEnum_PersistType').val());
}
```

و در هر قسمت که نیاز به مقدار enum بود با توجه به ایندکس مقدار را برای نمایش ازاین متغیر بیرون میکشیم:

```
function GetPersistTypeTitle_Concept(enumId) {
    return Enum_PersistType[enumId];
}
```

برای مثال در dropdown در سمت کلاینت این نوع استفاده شده و در حالتی از صفحه فقط برای نمایش عنوان آن احتیاج به دریافت آن از سمت سرور باشد میتوان از این روش کمک گرفت

و یا در سمت javascript میتوان با استفاده از jQuery مقادیر متغییر را در dropdown پر کرد.

```
function FillDropdown() {
    $("#ddlPersistType").html("");
    $.each(Enum_PersistType, function (key, value) {
        $("#ddlPersistType").append("<option></option>".val(key).html(value));
    });
}
```

[FillDropdownListOnClient.zip](#)

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۱:۵۴ ۱۳۹۲/۰۲/۱۲

با تشکر از شما.

فایل Newtonsoft.Json.dll در پروژه شما هست. JavaScriptSerializer توکار دات نت ازش استفاده نمی‌کنه. فقط از اسمبلی System.Web.Extensions.dll هست که استفاده می‌کنه.

نویسنده: مهدی پایروند
تاریخ: ۱۱:۵۷ ۱۳۹۲/۰۲/۱۲

ممنون از شما، برای ادامه این سری لازم میشده که در آینده اضافه میشه.
شما میتونید در صورت دلخواه کد قسمت serialize رو با این کتابخانه بنویسید:

```
//return new JavaScriptSerializer().Serialize(dict);  
return Newtonsoft.Json.JsonConvert.SerializeObject(dict);
```

نویسنده: سام ناصری
تاریخ: ۱۳:۴۴ ۱۳۹۲/۰۲/۱۲

به نظر من موضوع رو خیلی پیچیده کردی. برای تولید json لازم نیست که از کتابخانه خاصی استفاده کنی با همون استرینگ مینیوپولیشن ساده هم میشه این کار را کرد:

```
public static class EnumHelper  
{  
    public static Dictionary<string, EnumValueType> ToDictionary<EnumType, EnumValueType>()  
    {  
        return  
        Enum.GetValues(typeof(EnumType)).Cast<EnumValueType>().ToDictionary(i=>Enum.GetName(typeof(EnumType), i),  
i=>i);  
    }  
    public static string ToJson<EnumType>()  
    {  
        return "{" + string.Join(",",  
ToJson<EnumType, int>().Select(i=>string.Format(@"{{""{0}"" : {1}}", i.Key, i.Value)).ToArray())+ "}"  
;    }  
}
```

کلاس ساده بالا به سادگی json مورد نیاز را تولید میکند.

در ضمن برای اینجکت کردنش به صفحه هم لازم نیست که از فیلد مخفی استفاده کنی. به جاش json را مستقیم در محل مورد نظر رندر کن:

در Asp.Net MVC Razor

```
var x = @EnumHelper.ToJson<MyEnum>()
```

در Asp.Net Web Forms

```
var x = <%=EnumHelper.ToJson<MyEnum>()%>
```

همچنین همانطور که در مثالهای فوق نشان داده ام حتی لازم نیست از JSON.Parse استفاده کنی. البته من اینها را بر اساس ذهنیاتم خیلی سریع نوشتم و کدهای فوق را تست نکرده ام که ببینم درست کار میکنند یا نه. اما منظورم این بود که بپرسم چرا از فیلد مخفی استفاده کردی و چرا از JSON.Parse استفاده کردی و اینکه چرا از JavaScriptSerializer استفاده کردی؟

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۱۲ ۱۴:۲۶

در حالت کلی بهتره که از JavaScriptSerializer استفاده بشه چون [می‌تونه یک سری escape حروف خاص رو](#) لحاظ کنه.

نویسنده: سام ناصری
تاریخ: ۱۳۹۲/۰۲/۱۲ ۱۵:۰۶

موضوع این مقاله درباره Enum است و نه ارسال داده‌های کلی به کلاینت. پس آیا فکر میکنید در اینجا چیزی برای اسکیپ شدن وجود داره؟

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۱۲ ۱۶:۳۲

در مورد پیچیدگی صحبت کردید. راه شما به مراتب پیچیده‌تر است از روش مطرح شده و خوانایی کمتری داره. به علاوه هدف از ارائه مقالات بهتره ارائه راه حل‌هایی باشه تا حد امکان عمومی تا این که یک سری هک خاص مطرح بشه فقط مختص به یک روش خاص که فقط در یک مساله مشخص قابل استفاده باشه. بعد هم اگر کسی این هک رو جای دیگری استفاده کرد، چون نمی‌دونه یک سری از کاراکترها باید escape بشن، در ضمن کار گیر میفته. دید دادن برای حل مساله اینجا شاید بیشتر مطرح باشه تا حل مساله با یک هک ساده که فقط همینجا قابل استفاده است. همچنین زمانیکه یک سری متد تست شده داخل فریم ورک هست چرا باید رفت سراغ هک؟

ضمناً در ASP.NET MVC نیاز دارید که یک Html.Raw رو هم اضافه کنید و گر نه اطلاعات درج شده در صفحه encode می‌شن و در متغیر جاوا اسکریپتی قابل استفاده نخواهند بود.

نویسنده: مهدی پایروند
تاریخ: ۱۳۹۲/۰۲/۱۲ ۲۳:۰۸

در موردی مطلبی که آقای ناصری فرمودند باید بگم زمانیکه برنامه به سمت چند زبانه میره اهمیت پیدا میکنه که میتونید برای مثال مقدار یا لیستی از مقادیر متنی برای زبان خاصی رو با resource خودش به فیلد مخفی پاس بدید و در نمایش پیغام‌های مختلف سمت کلاینت استفاده کنید. مثل متن پیغام‌هایی که خاص ارتباط ajax میباشد که به زبان‌های مختلف ارائه کرد.

نویسنده: سام ناصری
تاریخ: ۱۳۹۲/۰۲/۱۳ ۴:۳۳

من کلاً نمیفهمم. در ضمن من سه تا سوال مطرح کردم (پاراگراف آخر کامنتم) که من باز هم نمیفهمم این جواب کدومشونه.

خیلی وقت‌ها لازم است تا نتیجه کوئری حاصله را بصورت Json به ویوی مورد نظر ارسال نمایید. برای اینکار کافیت 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 پیش 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 آورده 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 کتابخانه‌های 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 متغیر 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 ابجکت 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.PreserveReferencesHandling =
    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 متدExecute اون رو override کردم . جواب هم داد . فقط یه گیری داشت .اونم اینکه من تو مدل 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 کمک کنید .

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

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

چرا JSON.NET؟

[JSON.NET](#) یک کتابخانه‌ی سورس باز کار با اشیاء JSON در دات نت است. تاریخچه‌ی آن به 8 سال قبل بر می‌گردد و توسط یک برنامه نویسی نیوزیلندی به نام James Newton King تهیه شده‌است. اولین نگارش آن در سال 2006 ارائه شد؛ مقارن با زمانی که اولین استاندارد JSON نیز ارائه گردید.

این کتابخانه از آن زمان تا کنون، 6 میلیون بار دانلود شده‌است و به علت کیفیت بالای آن، این روزها پایه اصلی بسیاری از کتابخانه‌ها و فریم ورک‌های دات نت می‌باشد؛ مانند RavenDB تا ASP.NET Web API و SignalR مایکروسافت و همچنین گوگل نیز از آن جهت تدارک کلاینت‌های کار با API خود استفاده می‌کنند.

هرچند دات نت برای نمونه در نگارش سوم آن جهت مصارف WCF کلاسی را به نام [DataContractJsonSerializer](#) ارائه کرد، اما کار کردن با آن محدود است به فرمت خاص WCF به همراه عدم انعطاف پذیری و سادگی کار با آن. به علاوه باید در نظر داشت که JSON.NET از دات نت 2 به بعد تا مونو، Win8 و ویندوز فون را نیز پشتیبانی می‌کند.

برای نصب آن نیز کافی است دستور ذیل را در کنسول پاورشل نیوگت اجرا کنید:

```
PM> install-package Newtonsoft.Json
```

معماری JSON.NET

کتابخانه‌ی JSON.NET از سه قسمت عمده تشکیل شده‌است:

الف) JsonSerializer

ب) LINQ to JSON

ج) JSON Schema

الف) JsonSerializer

کار JsonSerializer تبدیل اشیاء دات نت به JSON و برعکس است. مزیت مهم آن امکانات قابل توجه تنظیم عملکرد و خروجی آن می‌باشد که این تنظیمات را به شکل ویژگی‌های خواص نیز می‌توان اعمال نمود. به علاوه امکان سفارشی سازی هر کدام نیز توسط کلاسی به نام JsonConvert، پیش بینی شده‌است.

یک مثال:

```
var roles = new List<string>
{
    "Admin",
    "User"
};
string json = JsonConvert.SerializeObject(roles, Formatting.Indented);
```

در اینجا نحوه‌ی استفاده از JSON.NET را جهت تبدیل یک شیء دات نت، به معادل JSON آن مشاهده می‌کنید. اعمال تنظیم Formatting.Indented سبب خواهد شد تا خروجی آن دارای Indentation باشد. برای نمونه اگر در برنامه‌ی خود قصد دارید فرمت JSON تو در تویی را به نحو زیبا و خوانایی نمایش دهید یا چاپ کنید، همین تنظیم ساده کافی خواهد بود.

و یا در مثال ذیل استفاده از یک anonymous object را مشاهده می‌کنید:

```
var jsonString = JsonConvert.SerializeObject(new
{
    Id = 1,
    Name = "Test"
}, Formatting.Indented);
```

به صورت پیش فرض تنها خواص عمومی کلاس‌ها توسط JSON.NET تبدیل خواهند شد.

تنظیمات پیشرفته‌تر JSON.NET

مزیت مهم JSON.NET بر سایر کتابخانه‌های موجود مشابه، قابلیت‌های سفارشی سازی قابل توجه آن است. در مثال ذیل نحوه‌ی معرفی JsonSerializerSettings را مشاهده می‌نمائید:

```
var jsonData = JsonConvert.SerializeObject(new
{
    Id = 1,
    Name = "Test",
    DateTime = DateTime.Now
}, new JsonSerializerSettings
{
    Formatting = Formatting.Indented,
    Converters =
    {
        new JavaScriptDateTimeConverter()
    }
});
```

در اینجا با استفاده از تنظیم JavaScriptDateTimeConverter، می‌توان خروجی DateTime استاندارد را به مصرف کنندگان جاوا اسکریپتی سمت کاربر ارائه داد؛ با خروجی ذیل:

```
{
  "Id": 1,
  "Name": "Test",
  "DateTime": new Date(1409821985245)
}
```

نوشتن خروجی JSON در یک استریم

خروجی متد JsonConvert.SerializeObject یک رشته‌است که در صورت نیاز به سادگی توسط متد File.WriteAllText در یک فایل قابل ذخیره می‌باشد. اما برای رسیدن به حداکثر کارایی و سرعت می‌توان از استریم‌ها نیز استفاده کرد:

```
using (var stream = File.CreateText(@"c:\output.json"))
{
    var jsonSerializer = new JsonSerializer
    {
        Formatting = Formatting.Indented
    };
    jsonSerializer.Serialize(stream, new
    {
        Id = 1,
        Name = "Test",
        DateTime = DateTime.Now
    });
}
```

کلاس JsonSerializer و متد Serialize آن یک استریم را نیز جهت نوشتن خروجی می‌پذیرند. برای مثال response.Output برنامه‌های وب نیز یک استریم است و در اینجا نوشتن مستقیم در استریم بسیار سریعتر است از تبدیل شیء به رشته و سپس ارائه خروجی آن؛ زیرا سربار تهیه رشته JSON از آن حذف می‌گردد و نهایتاً GC کار کمتری را باید انجام دهد.

تبدیل رشته‌ای به اشیاء دات نت

اگر رشته‌ی jsonData ای را که پیشتر تولید کردیم، بخواهیم تبدیل به نمونه‌ای از شیء User ذیل کنیم:

```
public class User
{
```

```
public int Id { set; get; }
public string Name { set; get; }
public DateTime DateTime { set; get; }
}
```

خواهیم داشت:

```
var user = JsonConvert.DeserializeObject<User>(jsonData);
```

در اینجا از متد `DeserializeObject` به همراه مشخص سازی صریح نوع شیء نهایی استفاده شده است. البته در اینجا با توجه به استفاده از `JavaScriptDateTimeConverter` برای تولید `jsonData`، نیاز است چنین تنظیمی را نیز در حالت `DeserializeObject` مشخص کنیم:

```
var user = JsonConvert.DeserializeObject<User>(jsonData, new JsonSerializerSettings
{
    Converters = { new JavaScriptDateTimeConverter() }
});
```

مقدار دهی یک نمونه یا وهله‌ی از پیش موجود

متد `JsonConvert.DeserializeObject` یک شیء جدید را ایجاد می‌کند. اگر قصد دارید صرفاً تعدادی از خواص یک وهله‌ی موجود، توسط JSON.NET مقدار دهی شوند از متد `PopulateObject` استفاده کنید:

```
JsonConvert.PopulateObject(jsonData, user);
```

کاهش حجم JSON تولیدی

زمانیکه از متد `JsonConvert.SerializeObject` استفاده می‌کنیم، تمام خواص عمومی تبدیل به معادل JSON آن‌ها خواهند شد؛ حتی خواصی که مقدار ندارند. این خواص در خروجی JSON، با مقدار `null` مشخص می‌شوند. برای حذف این خواص از خروجی JSON نهایی تنها کافی است در تنظیمات `JsonSerializerSettings`، مقدار `NullValueHandling = NullValueHandling.Ignore` مشخص گردد.

```
var jsonData = JsonConvert.SerializeObject(object, new JsonSerializerSettings
{
    NullValueHandling = NullValueHandling.Ignore,
    Formatting = Formatting.Indented
});
```

مورد دیگری که سبب کاهش حجم خروجی نهایی خواهد شد، تنظیم `DefaultValueHandling = DefaultValueHandling.Ignore` است. در این حالت کلیه خواصی که دارای مقدار پیش فرض خودشان هستند، در خروجی JSON ظاهر نخواهند شد. مثلاً مقدار پیش فرض خاصیت `int` مساوی صفر است. در این حالت کلیه خواص از نوع `int` که دارای مقدار صفر می‌باشند، در خروجی قرار نمی‌گیرند.

به علاوه حذف `Formatting = Formatting.Indented` نیز توصیه می‌گردد. در این حالت فشرده‌ترین خروجی ممکن حاصل خواهد شد.

مدیریت ارث بری توسط JSON.NET

در مثال ذیل کلاس کارمند و کلاس مدیر را که خود نیز در اصل یک کارمند می‌باشد، ملاحظه می‌کنید:

```
public class Employee
{
    public string Name { set; get; }
}

public class Manager : Employee
{
    public IList<Employee> Reports { set; get; }
}
```

در اینجا هر مدیر لیست کارمندانی را که به او گزارش می‌دهند نیز به همراه دارد. در ادامه نمونه‌ای از مقدار دهی این اشیاء ذکر شده‌اند:

```
var employee = new Employee { Name = "User1" };
var manager1 = new Manager { Name = "User2" };
var manager2 = new Manager { Name = "User3" };
manager1.Reports = new[] { employee, manager2 };
manager2.Reports = new[] { employee };
```

با فراخوانی

```
var list = JsonConvert.SerializeObject(manager1, Formatting.Indented);
```

یک چنین خروجی JSON ایی حاصل می‌شود:

```
{
  "Reports": [
    {
      "Name": "User1"
    },
    {
      "Reports": [
        {
          "Name": "User1"
        }
      ],
      "Name": "User3"
    }
  ],
  "Name": "User2"
}
```

این خروجی JSON جهت تبدیل به نمونه‌ی معادل دات نتی خود، برای مثال جهت رسیدن به manager1 در کدهای فوق، چندین مشکل را به همراه دارد:

- در اینجا مشخص نیست که این اشیاء، کارمند هستند یا مدیر. برای مثال مشخص نیست User2 چه نوعی دارد و باید به کدام شیء نگاشت شود.

- مشکل دوم در مورد کاربر User1 است که در دو قسمت تکرار شده‌است. این شیء JSON اگر به نمونه‌ی معادل دات نتی خود نگاشت شود، به دو وهله از User1 خواهیم رسید و نه یک وهله‌ی اصلی که سبب تولید این خروجی JSON شده‌است.

برای حل این دو مشکل، تغییرات ذیل را می‌توان به JSON.NET اعمال کرد:

```
var list = JsonConvert.SerializeObject(manager1, new JsonSerializerSettings
{
    Formatting = Formatting.Indented,
    TypeNameHandling = TypeNameHandling.Objects,
    PreserveReferencesHandling = PreserveReferencesHandling.Objects
});
```

با این خروجی:

```
{
  "$id": "1",
  "$type": "JsonNetTests.Manager, JsonNetTests",
  "Reports": [
    {
      "$id": "2",
      "$type": "JsonNetTests.Employee, JsonNetTests",
      "Name": "User1"
    },
    {
      "$id": "3",
      "$type": "JsonNetTests.Manager, JsonNetTests",
      "Reports": [
        {
          "$ref": "2"
        }
      ],
      "Name": "User3"
    }
  ],
  "Name": "User2"
}
```

- با تنظیم `TypeNameHandling = TypeNameHandling.Objects` سبب خواهیم شد تا خاصیت اضافه‌ای به نام `type$` به خروجی JSON اضافه شود. این نوع، در حین فراخوانی متد `JsonConvert.DeserializeObject` جهت تشخیص صحیح نگاشت اشیاء بکار گرفته خواهد شد و اینبار مشخص است که کدام شیء، کارمند است و کدامیک مدیر.

- با تنظیم `PreserveReferencesHandling = PreserveReferencesHandling.Objects` شماره Id خودکاری نیز به خروجی JSON اضافه می‌گردد. اینبار اگر به گزارش دهنده‌ها با دقت نگاه کنیم، مقدار `ref=2$` را خواهیم دید. این مورد سبب می‌شود تا در حین نگاشت نهایی، دو وهله متفاوت از شیء با `Id=2` تولید نشود.

باید دقت داشت که در حین استفاده از `JsonConvert.DeserializeObject` نیز باید `JsonSerializerSettings` یاد شده، تنظیم شوند.

ویژگی‌های قابل تنظیم در JSON.NET

علاوه بر `JsonSerializerSettings` که از آن صحبت شد، در JSON.NET امکان تنظیم یک سری از ویژگی‌ها به ازای خواص مختلف نیز وجود دارند.

- برای نمونه ویژگی `JsonIgnore` معروفترین آن‌ها است:

```
public class User
{
    public int Id { set; get; }

    [JsonIgnore]
    public string Name { set; get; }

    public DateTime DateTime { set; get; }
}
```

`JsonIgnore` سبب می‌شود تا خاصیتی در خروجی نهایی JSON تولیدی حضور نداشته باشد و از آن صرف‌نظر شود.

- با استفاده از ویژگی `JsonProperty` اغلب مواردی را که پیشتر بحث کردیم مانند `TypeNameHandling`، `NullValueHandling` و غیره، می‌توان تنظیم نمود. همچنین گاهی از اوقات کتابخانه‌های جاوا اسکریپتی سمت کاربر، از اسامی خاصی که از روش‌های نامگذاری دات نت پیروی نمی‌کنند، در طراحی خود استفاده می‌کنند. در اینجا می‌توان نام خاصیت نهایی را که قرار است رندر شود نیز صریحاً مشخص کرد. برای مثال:

```
[JsonProperty(PropertyName = "m_name", NullValueHandling = NullValueHandling.Ignore)]
public string Name { set; get; }
```

همچنین در اینجا امکان تنظیم Order نیز وجود دارد. برای مثال مشخص کنیم که خاصیت X در ابتدا قرار گیرد و پس از آن خاصیت Y رندر شود.

- استفاده از ویژگی JsonObject به همراه مقدار OptIn آن به این معنا است که از کلیه خواصی که دارای ویژگی JsonProperty نیستند، صرفنظر شود. حالت پیش فرض آن OptOut است؛ یعنی تمام خواص عمومی در خروجی JSON حضور خواهند داشت منهای مواردی که با JsonIgnore مزین شوند.

```
[JsonObject(MemberSerialization.OptIn)]
public class User
{
    public int Id { set; get; }

    [JsonProperty]
    public string Name { set; get; }

    public DateTime DateTime { set; get; }
}
```

- با استفاده از ویژگی JsonConverter می‌توان نحوه‌ی رندر شدن مقدار خاصیت را سفارشی‌سازی کرد. برای مثال:

```
[JsonConverter(typeof(JavaScriptDateTimeConverter))]
public DateTime DateTime { set; get; }
```

تهیه یک JsonConverter سفارشی

با استفاده از JsonConverter می‌توان کنترل کاملی را بر روی اعمال serialization و deserialization مقادیر خواص اعمال کرد. مثال زیر را در نظر بگیرید:

```
public class HtmlColor
{
    public int Red { set; get; }
    public int Green { set; get; }
    public int Blue { set; get; }
}

var colorJson = JsonConvert.SerializeObject(new HtmlColor
{
    Red = 255,
    Green = 0,
    Blue = 0
}, Formatting.Indented);
```

در اینجا علاقمندیم، در حین عملیات serialization، بجای اینکه مقادیر اجزای رنگ تهیه شده به صورت int نمایش داده شوند، کل رنگ با فرمت hex رندر شوند. برای اینکار نیاز است یک JsonConverter سفارشی را تدارک دید:

```
public class HtmlColorConverter : JsonConverter
{
    public override bool CanConvert(Type objectType)
    {
        return objectType == typeof(HtmlColor);
    }

    public override object ReadJson(JsonReader reader, Type objectType,
        object existingValue, JsonSerializer serializer)
    {
        throw new NotSupportedException();
    }

    public override void WriteJson(JsonWriter writer, object value, JsonSerializer serializer)
    {
        var color = value as HtmlColor;
        if (color == null)
        {
            // ...
        }
    }
}
```

```

        return;

        writer.WriteValue("#" + color.Red.ToString("X2")
            + color.Green.ToString("X2") + color.Blue.ToString("X2"));
    }
}

```

کار با ارث بری از کلاس پایه `JsonConverter` شروع می‌شود. سپس باید تعدادی از متدهای این کلاس پایه را بازنویسی کرد. در متد `CanConvert` اعلام می‌کنیم که تنها اشیایی از نوع کلاس `HtmlColor` را قرار است پردازش کنیم. سپس در متد `WriteJson` منطق سفارشی خود را می‌توان پیاده سازی کرد. از آنجائیکه این تبدیلگر صرفاً قرار است برای حالت `serialization` استفاده شود، قسمت `ReadJson` آن پیاده سازی نشده‌است.

در آخر برای استفاده از آن خواهیم داشت:

```

var colorJson = JsonConvert.SerializeObject(new HtmlColor
{
    Red = 255,
    Green = 0,
    Blue = 0
}, new JsonSerializerSettings
{
    Formatting = Formatting.Indented,
    Converters = { new HtmlColorConverter() }
});

```


نظرات خوانندگان

نویسنده: افتابی
تاریخ: ۲۱:۵۴ ۱۳۹۳/۰۶/۱۳

سلام؛ من مطالب مربوطه رو خوندم فقط اینکه توی یه صفحه rozar در mvc من به چه نحو میتونم از آن استفاده کنم ، حتی توی سایت خودش هم رفتم و sample ها رو دیدم فقط میخوام در یک پروژه به چه نحو ازش استفاده کنم و کجا کارش ببرم؟

نویسنده: وحید نصیری
تاریخ: ۲۱:۵۹ ۱۳۹۳/۰۶/۱۳

در یک اکشن متد، بجای return Json پیش فرض و توکار، می‌شود نوشت:

```
return Content(JsonConvert.SerializeObject(obj));
```

البته این ساده‌ترین روش استفاده از آن است؛ برای مقاصد Ajax ایی. و یا برای ذکر Content type می‌توان به صورت زیر عمل کرد:

```
return new ContentResult
{
    Content = JsonConvert.SerializeObject(obj),
    ContentType = "application/json"
};
```

نویسنده: رحمت اله رضایی
تاریخ: ۱۴:۳ ۱۳۹۳/۰۶/۱۴

"ASP.NET Web API و SignalR از این کتابخانه استفاده می‌کنند". دلیلی دارد هنوز ASP.NET MVC از این کتابخانه استفاده نکرده است؟

نویسنده: وحید نصیری
تاریخ: ۱۴:۱۹ ۱۳۹۳/۰۶/۱۴

- تا ASP.NET MVC 5 از JavaScriptSerializer در [JsonResult](#) استفاده می‌شود.
- در نگارش بعدی ASP.NET MVC که با Web API یکی شده (یعنی در یک کنترلر هم می‌توانید ActionResult داشته باشید و هم خروجی‌های متداول Web API را با هم) اینبار تامین کننده‌ی [JsonResult](#) از طریق تزریق وابستگی‌ها تامین می‌شود و می‌تواند هر کتابخانه‌ای که صلاح می‌دانید باشد. البته [یک مقدار پیش فرض](#) هم دارد که دقیقاً از JSON.NET استفاده می‌کند.

نویسنده: وحید نصیری
تاریخ: ۱۲:۳۵ ۱۳۹۳/۰۷/۱۸

یک نکته‌ی تکمیلی

استفاده از استریم‌ها برای کار با فایل‌ها در JSON.NET

```
public static T DeserializeFromFile<T>(string filePath, JsonSerializerSettings settings = null)
{
    if (!File.Exists(filePath))
        return default(T);

    using (var fileStream = File.OpenRead(filePath))
    {
        using (var streamReader = new StreamReader(fileStream))
        {
            using (var reader = new JsonTextReader(streamReader))
            {
                var serializer = settings == null ? JsonSerializer.Create() :
                JsonSerializer.Create(settings);
            }
        }
    }
}
```

```

        return serializer.Deserialize<T>(reader);
    }
}

public static void SerializeToFile(string filePath, object data, JsonSerializerSettings
settings = null)
{
    using (var fileStream = new FileStream(filePath, FileMode.Create))
    {
        using (var streamReader = new StreamWriter(fileStream))
        {
            using (var reader = new JsonTextWriter(streamReader))
            {
                var serializer = settings == null ? JsonSerializer.Create() :
                JsonSerializer.Create(settings);
                serializer.Serialize(reader, data);
            }
        }
    }
}

```

[پس از بررسی مقدماتی](#) امکانات کتابخانه‌ی JSON.NET، در ادامه به تعدادی از تنظیمات کاربردی آن با ذکر مثال‌هایی خواهیم پرداخت.

گرفتن خروجی از CamelCase در JSON.NET

یک سری از کتابخانه‌های جاوا اسکریپتی سمت کلاینت، به نام‌های خواص [CamelCase](#) نیاز دارند و حالت پیش فرض اصول نامگذاری خواص در دات نت عکس آن است. برای مثال بجای UserName به userName نیاز دارند تا بتوانند صحیح کار کنند. روش اول حل این مشکل، استفاده از ویژگی JsonProperty بر روی تک تک خواص و مشخص کردن نام‌های مورد نیاز کتابخانه‌ی جاوا اسکریپتی به صورت صریح است. روش دوم، استفاده از تنظیمات ContractResolver می‌باشد که با تنظیم آن به CamelCasePropertyNameContractResolver به صورت خودکار به تمامی خواص به صورت یکسانی اعمال می‌گردد:

```
var json = JsonConvert.SerializeObject(obj, new JsonSerializerSettings
{
    ContractResolver = new CamelCasePropertyNamesContractResolver()
});
```

درج نام‌های المان‌های یک Enum در خروجی JSON

اگر یکی از عناصر در حال تبدیل به JSON، از نوع enum باشد، به صورت پیش فرض مقدار عددی آن در JSON نهایی درج می‌گردد:

```
using Newtonsoft.Json;

namespace JsonNetTests
{
    public enum Color
    {
        Red,
        Green,
        Blue,
        White
    }

    public class Item
    {
        public string Name { set; get; }
        public Color Color { set; get; }
    }

    public class EnumTests
    {
        public string GetJson()
        {
            var item = new Item
            {
                Name = "Item 1",
                Color = Color.Blue
            };

            return JsonConvert.SerializeObject(item, Formatting.Indented);
        }
    }
}
```

با این خروجی:

```
{
  "Name": "Item 1",
  "Color": 2
}
```

اگر علاقمند هستید که بجای عدد 2، دقیقاً مقدار Blue در خروجی JSON درج گردد، می‌توان به یکی از دو روش ذیل عمل کرد:
الف) مزین کردن خاصیت از نوع enum به ویژگی JsonSerializer.Converters از نوع StringEnumConverter:

```
[JsonConverter(typeof(StringEnumConverter))]
public Color Color { set; get; }
```

ب) و یا اگر می‌خواهید این تنظیم به تمام خواص از نوع enum به صورت یکسانی اعمال شود، می‌توان نوشت:

```
return JsonConvert.SerializeObject(item, new JsonSerializerSettings
{
    Formatting = Formatting.Indented,
    Converters = { new StringEnumConverter() }
});
```

تهیه خروجی JSON از مدل‌های مرتبط، بدون Stack overflow

دو کلاس گروه‌های محصولات و محصولات ذیل را در نظر بگیرید:

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

    public virtual ICollection<Product> Products { get; set; }

    public Category()
    {
        Products = new List<Product>();
    }
}

public class Product
{
    public int Id { get; set; }
    public string Name { get; set; }

    public virtual Category Category { get; set; }
}
```

این نوع طراحی در Entity framework بسیار مرسوم است. در اینجا طرف‌های دیگر یک رابطه، توسط خاصیتی virtual معرفی می‌شوند که به آن‌ها خواص راهبری یا navigation properties هم می‌گویند.
با توجه به این دو کلاس، سعی کنید مثال ذیل را اجرا کرده و از آن، خروجی JSON تهیه کنید:

```
using System.Collections.Generic;
using Newtonsoft.Json;
using Newtonsoft.Json.Converters;

namespace JsonNetTests
{
    public class SelfReferencingLoops
    {
        public string GetJson()
        {
            var category = new Category
            {
                Id = 1,
                Name = "Category 1"
            };
            var product = new Product
```

```

        {
            Id = 1,
            Name = "Product 1"
        };

        category.Products.Add(product);
        product.Category = category;

        return JsonConvert.SerializeObject(category, new JsonSerializerSettings
        {
            Formatting = Formatting.Indented,
            Converters = { new StringEnumConverter() }
        });
    }
}

```

برنامه با این استثناء متوقف می‌شود:

```

An unhandled exception of type 'Newtonsoft.Json.JsonSerializationException' occurred in
Newtonsoft.Json.dll
Additional information: Self referencing loop detected for property 'Category' with type
'JsonNetTests.Category'. Path 'Products[0]'.

```

اصل خطای معروف فوق «Self referencing loop detected» است. در اینجا کلاس‌هایی که به یکدیگر ارجاع می‌دهند، در حین عملیات Serialization سبب بروز یک حلقه‌ی بازگشتی بی‌نهایت شده و در آخر، برنامه با خطای stack overflow خاتمه می‌یابد.

راه حل اول:

به تنظیمات JSON.NET، مقدار `ReferenceLoopHandling = ReferenceLoopHandling.Ignore` را اضافه کنید تا از حلقه‌ی بازگشتی بی‌پایان جلوگیری شود:

```

return JsonConvert.SerializeObject(category, new JsonSerializerSettings
{
    Formatting = Formatting.Indented,
    ReferenceLoopHandling = ReferenceLoopHandling.Ignore,
    Converters = { new StringEnumConverter() }
});

```

راه حل دوم:

به تنظیمات JSON.NET، مقدار `PreserveReferencesHandling = PreserveReferencesHandling.Objects` را اضافه کنید تا مدیریت ارجاعات اشیاء توسط خود JSON.NET انجام شود:

```

return JsonConvert.SerializeObject(category, new JsonSerializerSettings
{
    Formatting = Formatting.Indented,
    PreserveReferencesHandling = PreserveReferencesHandling.Objects,
    Converters = { new StringEnumConverter() }
});

```

خروجی حالت دوم به این شکل است:

```

{
  "$id": "1",
  "Id": 1,
  "Name": "Category 1",
  "Products": [
    {
      "$id": "2",
      "Id": 1,
      "Name": "Product 1",
      "Category": {
        "$ref": "1"
      }
    }
  ]
}

```

```
]
}
```

همانطور که ملاحظه می‌کنید، دو خاصیت `id$` و `ref$` توسط JSON.NET به خروجی JSON اضافه شده‌است تا توسط آن بتواند ارجاعات و نمونه‌های اشیاء را تشخیص دهد.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۵:۱۷ ۱۳۹۳/۰۶/۲۳

گرفتن خروجی مرتب شده بر اساس نام خواص (جهت مقاصد نمایشی):

تعریف DefaultContractResolver

```
public class OrderedContractResolver : DefaultContractResolver
{
    protected override IList<JsonProperty> CreateProperties(
        System.Type type, MemberSerialization memberSerialization)
    {
        return base.CreateProperties(type, memberSerialization).OrderBy(p =>
p.PropertyName).ToList();
    }
}
```

و بعد معرفی آن به نحو ذیل:

```
return JsonConvert.SerializeObject(data, new JsonSerializerSettings
{
    ContractResolver = new OrderedContractResolver()
});
```

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، مستقیماً در `response.Output` استریم 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 شده است. کدهای ابتدای متد `ExecuteResult` با [کدهای اصلی](#) `JsonResult` یکی هستند. فقط انتهای کار بجای استفاده از `JavaScriptSerializer`، از `JSON.NET` استفاده شده است. در این حالت برای استفاده از این `Action Result` جدید می توان 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 نمونه ای است که هم اکنون از آن استفاده می کنید. فقط اینبار قابلیت تنظیم `Settings` پیشرفته ای نیز به آن اضافه شده است.

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 می کند. در اینجا نیز اگر دقت کنید از کلاس `JavaScriptSerializer` استفاده شده است. 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: ۱۱:۱۸ ۱۳۹۳/۰۶/۱۶

تا جایی که من میدونم [JavaScriptSerializer](#) با Dictionary ها هم مشکل داشت ولی JSON.NET این مشکل رو نداره.

نویسنده: احمد
تاریخ: ۱۳:۲۲ ۱۳۹۳/۰۶/۱۶

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

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

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 دقیقاً 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: ۱۵:۵۱ ۱۳۹۳/۰۶/۱۶

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

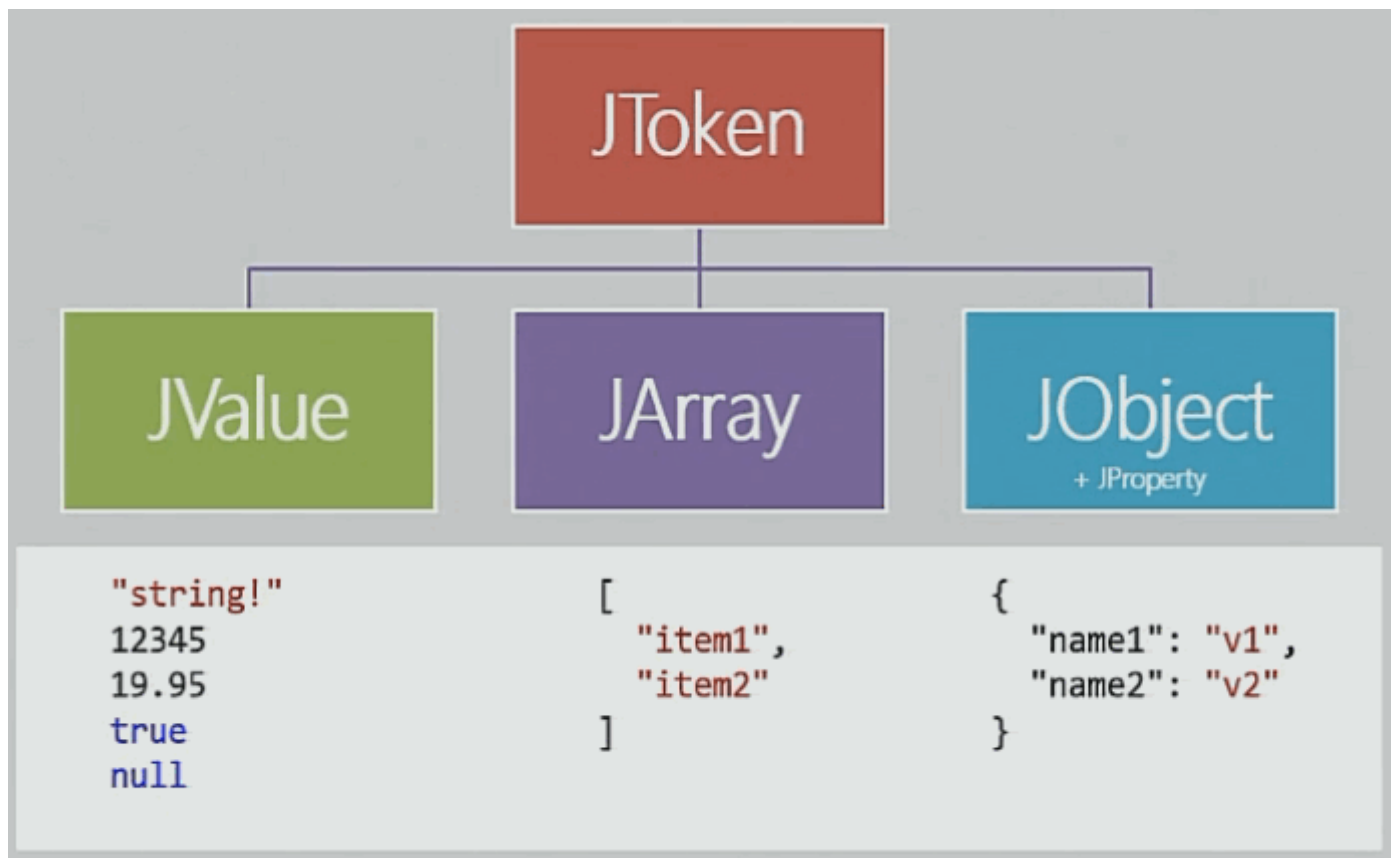
تاریخ:

۱۸:۵۱ ۱۳۹۳/۰۶/۱۶

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 ای‌نام
    }
    ),
```

عموما از امکانات LINQ to JSON کتابخانه‌ی JSON.NET زمانی استفاده می‌شود که ورودی JSON تو در توی حجیمی را دریافت کرده‌اید اما قصد ندارید به ازای تمام موجودیت‌های آن یک کلاس معادل را جهت نگاشت به آن‌ها تهیه کنید و صرفاً یک یا چند مقدار تو در توی آن جهت عملیات استخراج نهایی مدنظر است. به علاوه در اینجا LINQ to JSON واژه‌ی کلیدی dynamic را نیز پشتیبانی می‌کند.



همانطور که در تصویر مشخص است، خروجی‌های JSON عموماً ترکیبی هستند از مقادیر، آرایه‌ها و اشیاء. هر کدام از این‌ها در LINQ to JSON به اشیاء JValue، JArray و JObject نگاشت می‌شوند. البته در حالت JObject هر عضو به یک JProperty و JValue تجزیه خواهد شد.

برای مثال آرایه [1,2] تشکیل شده‌است از یک JArray به همراه دو JValue که مقادیر آن‌را تشکیل می‌دهند. اگر مستقیماً بخواهیم یک JArray را تشکیل دهیم می‌توان از شیء JArray استفاده کرد:

```
var array = new JArray(1, 2, 3);
var arrayToJson = array.ToString();
```

و اگر یک JSON رشته‌ای دریافتی را داریم می‌توان از متد Parse مربوط به JArray کمک گرفت:

```
var json = "[1,2,3]";
var jArray = JArray.Parse(json);
var val = (int)jArray[0];
```

خروجی JArray یک لیست از JToken ها است و با آن می‌توان مانند لیست‌های معمولی کار کرد.

در حالت کار با اشیاء، شیء JObject امکان تهیه اشیاء JSON ایی را دارا است که می‌تواند مجموعه‌ای از JProperty ها باشد:

```
var jobject = new JObject(
    new JProperty("prop1", "value1"),
    new JProperty("prop2", "value2")
);
var jobjectToJson = jobject.ToString();
```

با JObject به صورت dynamic نیز می‌توان کار کرد:

```
dynamic jsonObj = new JObject();
jsonObj.Prop1 = "value1";
jsonObj.Prop2 = "value2";
jsonObj.Roles = new[] { "Admin", "User"};
```

این روش بسیار شبیه است به حالتی که با اشیاء جاوا اسکریپتی در سمت کلاینت می‌توان کار کرد. و حالت عکس آن توسط متد JObject.Parse قابل انجام است:

```
var json = "{ 'prop1': 'value1', 'prop2': 'value2' }";
var jsonObj = JObject.Parse(json);
var val1 = (string)jsonObj["prop1"];
```

اکنون که با اجزای تشکیل دهنده‌ی LINQ to JSON آشنا شدیم، مثال ذیل را در نظر بگیرید:

```
var array = @"[
{
  'prop1': 'value1',
  'prop2': 'value2'
},
{
  'prop1': 'test1',
  'prop2': 'test2'
}
]";
var objects = JArray.Parse(array);
var obj1 = objects.FirstOrDefault(token => (string) token["prop1"] == "value1");
```

خروجی JArray یا JObject از نوع IEnumerable است و بر روی آن‌ها می‌توان کلیه متدهای LINQ را فراخوانی کرد. برای مثال در اینجا اولین شیء‌ایی که مقدار خاصیت prop1 آن مساوی value1 است، یافت می‌شود و یا می‌توان اشیاء را بر اساس مقدار خاصیتی مرتب کرده و سپس آن‌ها را بازگشت داد:

```
var values = objects.OrderBy(token => (string) token["prop1"])
    .Select(token => new { Value = (string) token["prop2"] })
    .ToList();
```

امکان انجام sub queries نیز در اینجا پیش بینی شده‌است:

```
var array = @"[
{
  'prop1': 'value1',
  'prop2': [1,2]
},
{
  'prop1': 'test1',
  'prop2': [1,2,3]
}
]";
var objects = JArray.Parse(array);
var objectContaining3 = objects.Where(token => token["prop2"].Any(v => (int)v == 3)).ToList();
```

در این مثال، خواص prop2 از نوع آرایه‌ای از اعداد صحیح هستند. با کوئری نوشته شده، اشیایی که خاصیت prop2 آن‌ها دارای عضو 3 است، یافت می‌شوند.

ASP.NET Web API در سمت سرور، برای مدیریت ApiController ها و در سمت کلاینت‌های دات نتی آن، برای مدیریت HttpClient، به صورت پیش فرض از JSON.NET استفاده می‌کند. در ادامه نگاهی خواهیم داشت به تنظیمات JSON در سرور و کلاینت‌های ASP.NET Web API.

آماده سازی یک مثال Self host

برای اینکه خروجی‌های JSON را بهتر و بدون نیاز به ابزار خاصی مشاهده کنیم، می‌توان یک پروژه‌ی کنسول جدید را آغاز کرده و سپس آن را تبدیل به Host مخصوص Web API کرد. برای اینکار تنها کافی است در کنسول پاور شل نیوگت دستور ذیل را صادر کنید:

```
PM> Install-Package Microsoft.AspNet.WebApi.OwinSelfHost
```

سپس کنترلر Web API ما از کدهای ذیل تشکیل خواهد شد که در آن در متد Post، قصد داریم اصل محتوای دریافتی از کاربر را نمایش دهیم. توسط متد GetAll آن، خروجی نهایی JSON آن در سمت کاربر بررسی خواهد شد.

```
using System;
using System.Collections.Generic;
using System.Net;
using System.Net.Http;
using System.Threading.Tasks;
using System.Web.Http;

namespace WebApiSelfHostTests
{
    public class UsersController : ApiController
    {
        public IEnumerable<User> GetAllUsers()
        {
            return new[]
            {
                new User{ Id = 1, Name = "User 1", Type = UserType.Admin },
                new User{ Id = 2, Name = "User 2", Type = UserType.User }
            };
        }

        public async Task<HttpResponseMessage> Post(HttpRequestMessage request)
        {
            var jsonContent = await request.Content.ReadAsStringAsync();
            Console.WriteLine("JsonContent (Server Side): {0}", jsonContent);
            return new HttpResponseMessage(HttpStatusCode.Created);
        }
    }
}
```

که در آن شیء کاربر چنین ساختاری را دارد:

```
namespace WebApiSelfHostTests
{
    public enum UserType
    {
        User,
        Admin,
        Writer
    }

    public class User
    {
        public int Id { set; get; }
        public string Name { set; get; }
        public UserType Type { set; get; }
    }
}
```



```
}
}
```

برای اعمال تنظیمات self host ابتدا نیاز است یک کلاس Startup مخصوص Owin را تهیه کرد:

```
using System.Web.Http;
using Newtonsoft.Json;
using Newtonsoft.Json.Converters;
using Owin;

namespace WebApiSelfHostTests
{
    /// <summary>
    /// PM> Install-Package Microsoft.AspNet.WebApi.OwinSelfHost
    /// </summary>
    public class Startup
    {
        public void Configuration(IAppBuilder appBuilder)
        {
            var config = new HttpConfiguration();
            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );

            appBuilder.UseWebApi(config);
        }
    }
}
```

که سپس با فراخوانی چند سطر ذیل، سبب راه اندازی سرور Web API، بدون نیاز به IIS خواهد شد:

```
var server = WebApp.Start<Startup>(url: BaseAddress);

Console.WriteLine("Press Enter to quit.");
Console.ReadLine();
server.Dispose();
```

در ادامه اگر در سمت کلاینت، دستورات ذیل را برای دریافت لیست کاربران صادر کنیم:

```
using (var client = new HttpClient())
{
    var response = client.GetAsync(BaseAddress + "api/users").Result;
    Console.WriteLine("Response: {0}", response);
    Console.WriteLine("JsonContent (Client Side): {0}", response.Content.ReadAsStringAsync().Result);
}
```

به این خروجی خواهیم رسید:

```
JsonContent (Client Side): [{"Id":1,"Name":"User 1","Type":1},{ "Id":2,"Name":"User 2","Type":0}]
```

همانطور که ملاحظه می‌کنید، مقدار Type مساوی صفر است. در اینجا چون Type را به صورت enum تعریف کرده‌ایم، به صورت پیش فرض مقدار عددی عضو انتخابی در JSON نهایی درج می‌گردد.

تنظیمات JSON سمت سرور Web API

برای تغییر این خروجی، در سمت سرور تنها کافی است به کلاس Startup مراجعه و HttpConfiguration را به صورت ذیل تنظیم کنیم:

```
public class Startup
```

```
{
    public void Configuration(IApplicationBuilder appBuilder)
    {
        var config = new HttpConfiguration();
        config.Formatters.JsonFormatter.SerializerSettings = new JsonSerializerSettings
        {
            Converters = { new StringEnumConverter() }
        };
    }
}
```

در اینجا با انتخاب `StringEnumConverter`، سبب خواهیم شد تا کلیه مقادیر `enum`، دقیقاً مساوی همان مقدار اصلی رشته‌ای آن‌ها در JSON نهایی درج شوند. اینبار اگر برنامه را اجرا کنیم، چنین خروجی حاصل می‌گردد و در آن دیگر `Type` مساوی صفر نیست:

```
JsonContent (Client Side): [{"Id":1,"Name":"User 1","Type":"Admin"}, {"Id":2,"Name":"User 2","Type":"User"}]
```

تنظیمات JSON سمت کلاینت Web API

اکنون در سمت کلاینت قصد داریم اطلاعات یک کاربر را با فرمت JSON به سمت سرور ارسال کنیم. روش متداول آن توسط کتابخانه‌ی `HttpClient`، استفاده از متد `PostAsJsonAsync` است:

```
var user = new User
{
    Id = 1,
    Name = "User 1",
    Type = UserType.Writer
};

var client = new HttpClient();
client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

var response = client.PostAsJsonAsync(BaseAddress + "api/users", user).Result;
Console.WriteLine("Response: {0}", response);
```

با این خروجی سمت سرور

```
JsonContent (Server Side): {"Id":1,"Name":"User 1","Type":2}
```

در اینجا نیز `Type` به صورت عددی ارسال شده‌است. برای تغییر آن نیاز است به متدی با سطح پایین‌تر از `PostAsJsonAsync` مراجعه کنیم تا در آن بتوان `JsonMediaTypeFormatter` را مقدار دهی کرد:

```
var jsonMediaTypeFormatter = new JsonMediaTypeFormatter
{
    SerializerSettings = new JsonSerializerSettings
    {
        Converters = { new StringEnumConverter() }
    }
};
var response = client.PostAsync(BaseAddress + "api/users", user,
jsonMediaTypeFormatter).Result;
Console.WriteLine("Response: {0}", response);
```

خاصیت `SerializerSettings` کلاس `JsonMediaTypeFormatter` برای اعمال تنظیمات JSON.NET پیش‌بینی شده‌است. اینبار مقدار دریافتی در سمت سرور به صورت ذیل است و در آن، `Type` دیگر عددی نیست:

```
JsonContent (Server Side): {"Id":1,"Name":"User 1","Type":"Writer"}
```

مثال کامل این بحث را از اینجا می‌توانید دریافت کنید:

نظرات خوانندگان

نویسنده:

وحید نصیری

تاریخ:

۱۰:۴۵ ۱۳۹۳/۰۶/۲۴

یک نکته‌ی تکمیلی

اگر نمی‌خواهید یک وابستگی جدید را (Microsoft.AspNet.WebApi.Client) به پروژه اضافه کنید، کدهای ذیل همان کار HttpClient را برای ارسال اطلاعات، انجام می‌دهند. کلاس WebRequest آن در فضای نام System.Net موجود است:

```
using System;
using System.IO;
using System.Net;
using Newtonsoft.Json;

namespace WebToolkit
{
    public class SimpleHttp
    {
        public HttpStatusCode PostAsJson(string url, object data, JsonSerializerSettings settings)
        {
            if (string.IsNullOrEmpty(url))
                throw new ArgumentNullException("url");

            return PostAsJson(new Uri(url), data, settings);
        }

        public HttpStatusCode PostAsJson(Uri url, object data, JsonSerializerSettings settings)
        {
            if (url == null)
                throw new ArgumentNullException("url");

            var postRequest = (HttpWebRequest)WebRequest.Create(url);
            postRequest.Method = "POST";
            postRequest.UserAgent = "SimpleHttp/1.0";
            postRequest.ContentType = "application/json; charset=utf-8";

            using (var stream = new StreamWriter(postRequest.GetRequestStream()))
            {
                var serializer = JsonSerializer.Create(settings);
                using (var writer = new JsonTextWriter(stream))
                {
                    serializer.Serialize(writer, data);
                    writer.Flush();
                }
            }

            using (var response = (HttpWebResponse)postRequest.GetResponse())
            {
                return response.StatusCode;
            }
        }
    }
}
```

نویسنده:

رشیدیان

تاریخ:

۱۸:۱ ۱۳۹۳/۰۶/۲۶

سلام و وقت بخیر

من وقتی می‌خوام اطلاعات یک فایل جیسون رو به آبجکت تبدیل کنم، با این خطا مواجه میشم:
Additional text encountered after finished reading JSON content: „ Path ", line 1, position 6982

بعد از جستجو متوجه شدم که خطا به دلیل وجود کرکترهای کنترلی هست، پس فایل مذکور رو با روشهای زیر (هر کدام رو جداگانه تست کردم) تمیز کردم:

```
public string RemoveControlCharacters1(string input)
{
    string output = Regex.Replace(input, @"[\u0000-\u001F]", string.Empty);
    return output;
}
```

```

    }

    public string RemoveControlCharacters2(string input)
    {
        string output = new string(input.Where(c => !char.IsControl(c)).ToArray());
        return output;
    }

    public string RemoveControlCharacters3(string inString)
    {
        if (inString == null) return null;
        StringBuilder newString = new StringBuilder();
        char ch;
        for (int i = 0; i < inString.Length; i++)
        {
            ch = inString[i];
            if (!char.IsControl(ch))
            {
                newString.Append(ch);
            }
        }
        return newString.ToString();
    }
}

```

اما کماکان همان خطا را در زمان اجر میبینم.

آیا مشکل چیز دیگری است؟

پرسش: چطور میشود به جیسون دات نت گفت که اصلا کرکترهای کنترلی و یا چیزهایی را که ممکن است خطا ایجاد کنند، ندید بگیرد؟

نویسنده: وحید نصیری
تاریخ: ۱۸:۱۲ ۱۳۹۳/۰۶/۲۶

با تنظیم eventArgs.ErrorContext.Handled = true از خطاهای موجود صرفنظر می‌شود:

```

new JsonSerializerSettings
{
    Error = (sender, eventArgs) =>
    {
        Debug.WriteLine(eventArgs.ErrorContext.Error.Message);
        //if an error happens we can mark it as handled, and it will continue
        eventArgs.ErrorContext.Handled = true;
    }
}

```

نویسنده: رشیدیان
تاریخ: ۱۸:۲۳ ۱۳۹۳/۰۶/۲۶

سپاسگزارم از پاسخ سریع شما.

بخشید کد من به این شکل هست و نمیدونم کجا باید تغییرات رو اعمال کنم:

```
var items = JsonConvert.DeserializeObject<List<Classified>>(json);
```

نویسنده: وحید نصیری
تاریخ: ۱۸:۲۶ ۱۳۹۳/۰۶/۲۶

در پارامتر دوم متد « [تبدیل JSON رشته‌ای به اشیاء دات نت](#) ».

نویسنده: رضایی
تاریخ: ۱۸:۴۶ ۱۳۹۳/۰۶/۲۶

سلام؛ من از کد زیر استفاده کردم

```
myUserApi.Id = UserId;
```

```
return new JsonResult
{
    Data = myUserApi,
    ContentType = "application/json"
};
```

اما این خروجی تولید میشه

```
{
  "$id": "1",
  "Settings": {
    "$id": "2",
    "ReferenceLoopHandling": 0,
    "MissingMemberHandling": 0,
    "ObjectCreationHandling": 0,
    "NullValueHandling": 0,
    "DefaultValueHandling": 0,
    "Converters": [],
    "PreserveReferencesHandling": 0,
    "TypeNameHandling": 0,
    "MetadataPropertyHandling": 0,
    "TypeNameAssemblyFormat": 0,
    "ConstructorHandling": 0,
    "ContractResolver": null,
    "ReferenceResolver": null,
    "TraceWriter": null,
    "Binder": null,
    "Error": null,
    "Context": {
      "$id": "3",
      "m_additionalContext": null,
      "m_state": 0
    },
    "DateFormatString": "yyyy'-'MM'-'dd'T'HH':'mm':'ss.FFFFFFFF",
    "MaxDepth": null,
    "Formatting": 0,
    "DateFormatHandling": 0,
    "DateTimeZoneHandling": 3,
    "DateParseHandling": 1,
    "FloatFormatHandling": 0,
    "FloatParseHandling": 0,
    "StringEscapeHandling": 0,
    "Culture": "(Default)",
    "CheckAdditionalContent": false
  },
  "ContentEncoding": null,
  "ContentType": "application/json",
  "Data": "{\r\n  \"Id\":2\r\n}",
  "JsonRequestBehavior": 1,
  "MaxJsonLength": null,
  "RecursionLimit": null
}
```

نویسنده: وحید نصیری
تاریخ: ۱۸:۵۵ ۱۳۹۳/۰۶/۲۶

مرتبط است به نکته‌ی « [تهیه خروجی JSON از مدل‌های مرتبط، بدون Stack overflow](#) »

ممکن است بخواهیم در پاسخ یک تقاضای Ajax ایی، اگر عملیات در سمت سرور با موفقیت انجام شد، خروجی یک Controller action را به کاربر نهایی نشان دهیم. در چنین سناریویی لازم است که بتوانیم خروجی یک action را بصورت رشته برگردانیم. در این مقاله به این مسئله خواهیم پرداخت.

فرض کنید در یک سیستم وبلاگ ساده قصد داریم امکان کامنت گذاشتن بصورت Ajax را پیاده سازی کنیم. یک ایده عملی و کارآ این است: بعد از اینکه کاربر متن کامنت را وارد کرد و دکمه‌ی ارسال کامنت را زد، تقاضا به سمت سرور ارسال شود و اگر سرور پیام موفقیت را صادر کرد، متن نوشته شده توسط کاربر را به کمک کدهای JavaScript و در همان سمت کلاینت بصورت یک کادر کامنت جدید به محتوای صفحه اضافه کنیم. بنده در اینجا برای اینکه بتوانم اصل موضوع مورد بحث را توضیح دهم، از یک سناریوی جایگزین استفاده می‌کنم؛ کاربر موقعی که دکمه ارسال را زد، تقاضا به سرور ارسال میشود. سرور بعد از انجام عملیات، تحت یک شی JSON هم نتیجه‌ی انجام عملیات و هم محتوای HTML نمایش کامنت جدید در صفحه را به سمت کلاینت ارسال خواهد کرد و کلاینت در صورت موفقیت آمیز بودن عملیات، آن محتوا را به صفحه اضافه می‌کند.

با توجه به توضیحات داده شده، ابتدا یک شیء نیاز داریم تا بتوانیم توسط آن نتیجه‌ی عملیات Ajax ایی را بصورت JSON به سمت کلاینت ارسال کنیم:

```
public class JsonResult
{
    public bool success { set; get; }
    public bool HasWarning { set; get; }
    public string WarningMessage { set; get; }
    public int errorcode { set; get; }

    public string message {set; get; }
    public object data { set; get; }
}
```

سپس به متدی نیاز داریم که کار تبدیل نتیجه‌ی action را به رشته، انجام دهد:

```
public static string RenderViewToString(ControllerContext context,
    string viewPath,
    object model = null,
    bool partial = false)
{
    ViewEngineResult viewEngineResult = null;
    if (partial) viewEngineResult = ViewEngines.Engines.FindPartialView(context, viewPath);
    else viewEngineResult = ViewEngines.Engines.FindView(context, viewPath, null);
    if (viewEngineResult == null) throw new FileNotFoundException("View cannot be found.");
    var view = viewEngineResult.View;
    context.Controller.ViewData.Model = model;
    string result = null;
    using(var sw = new StringWriter()) {
        var ctx = new ViewContext(context, view, context.Controller.ViewData,
            context.Controller.TempData, sw);
        view.Render(ctx, sw);
        result = sw.ToString();
    }
    return result;
}
```

در اینجا موتور View را بر اساس اطلاعات یک View، مدل و سایر اطلاعات Context جاری کنترلر، وادار به تولید معادل رشته‌ای آن می‌کنیم.

فرض کنیم در سمت Controller هم از کدی شبیه به این استفاده میکنیم:

```

public JsonResult AddComment(CommentViewModel model) {
    MyJsonResult result = new MyJsonResult() {
        success = false;
    };
    if (!ModelState.IsValid) {
        result.success = false;
        result.message = "لطفاً اطلاعات فرم را کامل وارد کنید";
        return Json(result);
    }
    try {
        Comment theComment = model.toCommentModel();
        //EF service factory
        Factory.CommentService.Create(theComment);
        Factory.SaveChanges();
        result.data = Tools.RenderViewToString(this.ControllerContext, "/views/posts/_AComment", model,
true);
        result.success = true;
    } catch (Exception ex) {
        result.success = false;
        result.message = "اشکال زمان اجرا";
    }
    return Json(result);
}

```

و در سمت کلاینت برای ارسال Form به صورت Ajax ایی خواهیم داشت:

```

@using (Ajax.BeginForm("AddComment", "posts",
new AjaxOptions()
{
    HttpMethod = "Post",
    OnSuccess = "AddCommentSuccess",
    LoadingElementId = "AddCommentLoading"
}, new { id = "frmAddComment", @class = "form-horizontal" }))
{
    @Html.HiddenFor(m => m.PostId)
    <label for="fname">@Texts.ContactName</label>
    <input type="text" id="fname" name="FullName" class="form-control" placeholder="@Texts.ContactName"
">
    <label for="email">@Texts.Email</label>
    <input type="email" id="inputEmail" name="email" class="form-control" placeholder="@Texts.Email">
    <br><textarea name="C_Content" cols="60" rows="10" class="form-control"></textarea><br>
    <input type="submit" value="@Texts.SubmitComments" name="" class="btn btn-primary">
    <div class="loading-mask" style="display:none">@Texts.LoadingMessage</div>
}

```

در اینجا در صورت موفقیت آمیز بودن عملیات، متد جاوا اسکریپتی AddCommentSuccess فراخوانی خواهد شد. باید توجه شود Texts در اینجا یک Resource هست که به منظور نگهداری کلمات استفاده شده در سایت، برای زبانهای مختلف استفاده می‌شود (رجوع شود به مفهوم بومی سازی در Asp.net).

و در قسمت script ها داریم:

```

<script type="text/javascript">
function AddCommentSuccess(jsData) {
    if (jsData && jsData.message)
        alert(jsData.message);
    if (jsData && jsData.success) {
        document.getElementById("frmAddComment").reset();
        //افزودن کامنت جدید ساخته شده توسط کاربر به لیست کامنتهای صفحه
        $("#divAllComments").html(jsData.data + $("#divAllComments").html());
    }
}
</script>

```

متد AddCommentSuccess اطلاعات شیء JSON بازگشتی از کنترلر را دریافت و سپس پیام آنرا در صورت موفقیت آمیز بودن عملیات، به DIV ایی با id مساوی divAllComments اضافه می‌کند.

روش‌های زیادی برای ارسال داده‌های سمت سرور تهیه شده در یک برنامه‌ی ASP.NET به کدهای سمت کاربر JavaScript ای وجود دارند که تعدادی از مهم‌ترین‌های آن‌ها را در این مطلب بررسی خواهیم کرد.

روش اول: دریافت اطلاعات سمت سرور به کمک درخواست‌های Ajax

استفاده از Ajax یکی از روش‌های کلاسیک دریافت اطلاعات سمت سرور در کدهای جاوا اسکریپتی است.

```
<script type="text/javascript">
var products = [];
$(function() {
    $.getJSON("/home/products", function(response) {
        products = response.products;
    });
});
</script>
```

برای نمونه در اینجا با استفاده از امکانات jQuery، درخواست Ajax ای به سرور ارسال شده و سپس نتیجه‌ی دریافتی، به آرایه‌ی جاوا اسکریپتی products نسبت داده شده‌است.

- مزایا: استفاده از Ajax، روشی بسیار متداول و شناخته شده‌است و به کمک انواع و اقسام روش‌های بازگشت JSON از سرور، می‌توان با آن کار کرد.

- معایب: درخواست Ajax، صرفاً پس از بارگذاری اولیه‌ی صفحه به سمت سرور ارسال خواهد شد و در این بین، کاربر وقفه‌ای را مشاهده خواهد کرد. همچنین در اینجا بجای یک درخواست از سرور، حداقل دو درخواست باید ارسال شوند؛ یکی برای بارگذاری صفحه‌ی اصلی و دیگری برای دریافت اطلاعات Ajax ای از سرور به صورت غیرهمزمان.

روش دوم: دریافت اطلاعات از یک فایل جاوا اسکریپتی خارجی

اطلاعات سمت کاربر را از یک فایل جاوا اسکریپتی خارجی الحاق شده‌ی به صفحه‌ی جاری نیز می‌توان تهیه کرد:

```
<script src="/file.js"></script>
```

یک چنین فایلی را می‌توان توسط [کدهای سمت سرور نیز بازگشت داد](#) و اطلاعات آن‌را تهیه و پر کرد. در این حالت فقط کافی است که ContentType شیء Response را از نوع application/javascript مشخص کنیم و سپس یک خروجی متنی جاوا اسکریپتی را از سمت سرور بازگشت دهیم.

این روش نیز تقریباً مانند حالت یک درخواست Ajax ای کار می‌کند و اطلاعات مورد نیاز را در طی یک درخواست جداگانه، پس از بارگذاری صفحه‌ی اصلی، از سرور دریافت خواهد کرد. البته در حالت کار با Ajax، می‌توان در طی یک callback، نتیجه را دریافت کرد و سپس عکس العمل نشان داد؛ اما در اینجا callback ای وجود ندارد.

روش سوم: استفاده از SignalR

در SignalR ابتدا سعی می‌شود تا با استفاده از Web Sockets ارتباطی ماندگار بین کلاینت و سرور برقرار شود و سپس در این حالت، سرور می‌تواند مدام اطلاعاتی، مانند تغییرات داده‌های خود را به سمت کاربر، جهت نمایش و یا محاسبات خاص خود ارسال کند. اگر حالت Web Socket میسر نباشد (توسط سرور یا کلاینت پشتیبانی نشود)، به حالت‌های دیگری مانند server events, forever frames, long polling سوئیچ خواهد کرد. [اطلاعات بیشتر](#)

روش چهارم: قرار دادن اطلاعات سمت سرور در کدهای HTML صفحه

روش متداول دیگری جهت تامین اطلاعات جاوا اسکریپتی سمت کاربر، قرار دادن آن‌ها در ویژگی‌های data-* ارائه شده در HTML5 است.

```
<ul>
@foreach (var product in products)
{
    <li id="product@product.Id" data-rank="@product.Rank">@product.Name</li>
}
</ul>
```

در اینجا لیستی از محصولات، به صورت متداولی از کنترلر دریافت گردیده و سپس ویژگی data-rank هر ردیف نمایش داده شده نیز توسط این اطلاعات سمت سرور مقدار دهی شده‌اند.

اکنون برای دسترسی به مقدار data-rank سطری مانند product1، در کدهای جاوا اسکریپتی صفحه می‌توان نوشت:

```
<script type="text/javascript">
var product1Rank = $("#product1").data("rank");
</script>
```

این روش برای قرار دادن اطلاعات ثابت اشیاء، روش مرسوم است.

روش پنجم: قرار دادن اطلاعات سمت سرور در کدهای جاوا اسکریپتی صفحه

این روش همانند روش چهارم است، با این تفاوت که اینبار اطلاعات مورد نیاز، مستقیماً به یک متغیر جاوا اسکریپتی انتساب داده شده‌است:

```
<script type="text/javascript">
var product1Name = "@product1.Name";
</script>
```

مزیت این روش، عدم ارسال درخواست اضافه‌تری به سرور برای دریافت اطلاعات مورد نیاز است. مقدار product1Name در همان بار اول رندر صفحه از سمت سرور، مشخص می‌گردد.

روش ششم: انتساب یک شیء دات نیتی به یک متغیر جاوا اسکریپتی

این روش همانند روش پنجم است، با این تفاوت که اینبار قصد داریم بجای یک مقدار ثابت رشته‌ای یا عددی، برای مثال، آرایه‌ای از اشیاء را به یک متغیر جاوا اسکریپتی انتساب دهیم. در اینجا ابتدا اطلاعات مورد نظر را به فرمت JSON تبدیل می‌کنیم:

```
// سمت سرور
[HttpGet]
public ActionResult Index()
{
    var array = new[]
    {
        "Afghanistan",
        "Albania",
        "Algeria",
        "American Samoa",
        "Andorra",
        "Angola",
        "Anguilla",
        "Antarctica",
        "Antigua and/or Barbuda"
    };
    ViewBag.JsonString = new JavaScriptSerializer().Serialize(array);
    return View();
}
```

سپس توسط متد `Html.Raw` می‌توان این رشته‌ی JSON را که اکنون حاوی آرایه جاوا اسکریپتی سمت سرور است، به یک متغیر جاوا اسکریپتی نسبت داد:

```
// سمت کلاینت  
<script type="text/javascript">  
var jsonArray = @Html.Raw(@ViewBag.JsonString);  
</script>
```

استفاده از `Html.Raw` در این حالت از این جهت ضروری است که اطلاعاتی مانند [] به صورت encode شده در سمت کاربر نمایش داده نشوند؛ چون Razor به صورت پیش فرض اطلاعات را Encode می‌کند. و یا اینکار را به صورت خلاصه به شکل زیر نیز می‌توان در سمت کاربر انجام داد:

```
<script type="text/javascript">  
var model = @Html.Raw(Json.Encode(Model));  
// your js code here  
</script>
```

در اینجا کار تبدیل اطلاعات مدل به JSON، در همان سمت `RazorView` انجام شده‌است.

چندی قبل مطلب «[اطلاع از بروز رسانی نرم افزار ساخته شده](#)» را در سایت جاری مطالعه کردید. در این روش بسیار متداول، شماره نگارش‌های جدید برنامه در یک فایل XML و مانند آن قرار می‌گیرند و برنامه هر بار این فایل را جهت یافتن شماره‌های مندرج در آن اسکن می‌کند. اگر پروژه‌ی شما سورس باز است و در GitHub هاست شده، روش دیگری نیز برای یافتن این اطلاعات وجود دارد. در GitHub می‌توان از طریق آدرسی به شکل https://api.github.com/repos/user_name/project_name/releases به اطلاعات آخرین ارائه‌های یک پروژه (قرار گرفته در برگه‌ی releases آن) با فرمت JSON دسترسی یافت (یک مثال). در ادامه قصد داریم روش استفاده‌ی از آن را بررسی کنیم.

ساختار JSON ارائه‌های یک پروژه در GitHub

ساختار کلی اطلاعات ارائه‌های یک پروژه در GitHub چنین شکلی را دارد:

```
[
  {
    release_info,
    "assets": [
      {
      }
    ]
  }
]
```

در اینجا آرایه‌ای از اطلاعات ارائه‌ی یک پروژه ارسال می‌شود. هر ارائه نیز دارای دو قسمت است: لینکی به صفحه‌ی اصلی release در GitHub و سپس آرایه‌ای به نام assets که در آن اطلاعات فایل‌های پیوستی مانند نام فایل، آدرس، اندازه و امثال آن قرار گرفته‌اند.

تهیه‌ی کلاس‌های معادل فرمت JSON ارائه‌های برنامه در GitHub

اگر بخواهیم قسمت‌های مهم خروجی JSON فوق را تبدیل به کلاس‌های معادل دات نت کنیم، به دو کلاس ذیل خواهیم رسید:

```
using Newtonsoft.Json;
using System;

namespace ApplicationAnnouncements
{
    public class GitHubProjectRelease
    {
        [JsonProperty(PropertyName = "url")]
        public string Url { get; set; }

        [JsonProperty(PropertyName = "assets_url")]
        public string AssetsUrl { get; set; }

        [JsonProperty(PropertyName = "upload_url")]
        public string UploadUrl { get; set; }

        [JsonProperty(PropertyName = "html_url")]
        public string HtmlUrl { get; set; }

        [JsonProperty(PropertyName = "id")]
        public int Id { get; set; }

        [JsonProperty(PropertyName = "tag_name")]
        public string TagName { get; set; }

        [JsonProperty(PropertyName = "target_commitish")]
        public string TargetCommitish { get; set; }

        [JsonProperty(PropertyName = "name")]
        public string Name { get; set; }
    }
}
```

```

    public string Name { get; set; }

    [JsonProperty(PropertyName = "body")]
    public string Body { get; set; }

    [JsonProperty(PropertyName = "draft")]
    public bool Draft { get; set; }

    [JsonProperty(PropertyName = "prerelease")]
    public bool PreRelease { get; set; }

    [JsonProperty(PropertyName = "created_at")]
    public DateTime CreatedAt { get; set; }

    [JsonProperty(PropertyName = "published_at")]
    public DateTime PublishedAt { get; set; }

    [JsonProperty(PropertyName = "assets")]
    public Asset[] Assets { get; set; }
}

public class Asset
{
    [JsonProperty(PropertyName = "url")]
    public string Url { get; set; }

    [JsonProperty(PropertyName = "id")]
    public int Id { get; set; }

    [JsonProperty(PropertyName = "name")]
    public string Name { get; set; }

    [JsonProperty(PropertyName = "label")]
    public string Label { get; set; }

    [JsonProperty(PropertyName = "content_type")]
    public string ContentType { get; set; }

    [JsonProperty(PropertyName = "state")]
    public string State { get; set; }

    [JsonProperty(PropertyName = "size")]
    public int Size { get; set; }

    [JsonProperty(PropertyName = "download_count")]
    public int DownloadCount { get; set; }

    [JsonProperty(PropertyName = "created_at")]
    public DateTime CreatedAt { get; set; }

    [JsonProperty(PropertyName = "updated_at")]
    public DateTime UpdatedAt { get; set; }
}

```

در اینجا از ویژگی [JsonProperty](#) جهت معرفی نام‌های واقعی خواص ارائه شده‌ی توسط GitHub استفاده کرده‌ایم. پس از تشکیل این کلاس‌ها، مرحله‌ی بعد، دریافت اطلاعات JSON از آدرس API ارائه‌های پروژه در GitHub و سپس نگاشت آن‌ها می‌باشد:

```

using (var webClient = new WebClient())
{
    webClient.Headers.Add("user-agent", "DNTProfiler");
    var jsonData = webClient.DownloadString(url);
    var githubProjectReleases = JsonConvert.DeserializeObject<GithubProjectRelease[]>(jsonData);

    foreach (var release in githubProjectReleases)
    {
        foreach (var asset in release.Assets)
        {
            // ...
        }
    }
}

```

حین کار با WebClient یا هر روش دیگری که برای دریافت اطلاعات از وب استفاده می‌کنید، حتما نیاز است user-agent را ذکر

کرد. در غیر اینصورت GitHub درخواست شما را برگشت خواهد زد. پس از دریافت اطلاعات JSON، با استفاده از متد `JsonConvert.DeserializeObject` کتابخانه‌ی [JSON.NET](https://www.json.net/)، می‌توان آن‌ها را تبدیل به آرایه‌ای از `GitHubProjectRelease` کرد.

یک نکته: اگر به صفحه‌ی اصلی ارائه‌های یک پروژه در GitHub دقت کنید، شمارهی تعداد بار دریافت یک ارائه مشخص نشده‌است. در این API، عدد `DownloadCount`، بیانگر تعداد بار دریافت پروژه‌ی شما است.