عنوان: **Http Batch Processing در Asp.Net Web Api**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۳/۱۱/۱۶ ۲۳:۴۵
آدرس: www.dotnettips.info
گروه‌ها: ASP.NET Web API, OData, Performance, ASP.NET Web API 2, Batch Request

بعد از معرفی نسخه‌ی 2 از Asp.Net Web Api و پشتیبانی رسمی آن از OData بسیاری از توسعه دهندگان سیستم نفس راحتی کشیدند؛ زیرا از آن پس می‌توانستند علاوه بر امکانات جالب و مهمی که تحت پروتکل OData میسر بود، از سایر امکانات تعبیه شده در نسخه‌ی دوم web Api نیز استفاده نمایند. یکی از این قابلیت‌ها، مبحث مهم Batching Processing است که در طی این پست با آن آشنا خواهیم شد.
منظور از Batch Request این است که درخواست دهنده بتواند چندین درخواست (Multiple Http Request) را به صورت یک Pack جامع، در قالب فقط یک درخواست (Single Http Request) ارسال نماید و به همین روال تمام پاسخ‌های معادل درخواست ارسال شده را به صورت یک Pack دیگر دریافت کرده و آن را پردازش نماید. نوع درخواست نیز مهم نیست یعنی می‌توان در قالب یک Pack چندین درخواست از نوع Post و Get یا حتی Put و ... نیز داشته باشید. بدیهی است که پیاده سازی این قابلیت در جای مناسب و در پروژه‌هایی با تعداد کاربران زیاد می‌تواند باعث بهبود چشمگیر کارآیی پروژه شود.

برای شروع همانند سایر مطالب می‌توانید از این پست جهت راه اندازی هاست سرویس‌های Web Api استفاده نمایید. برای فعال سازی قابلیت batching Request نیاز به یک MessageHandler داریم تا بتوانند درخواست‌هایی از این نوع را پردازش نمایند. خوشبختانه به صورت پیش فرض این Handler پیاده سازی شده‌است و ما فقط باید آن را با استفاده از متد MapHttpBatchRoute به بخش مسیر یابی (Route Handler) پروژه معرفی نماییم.

```
public class Startup
    {
        public void Configuration(IAppBuilder appBuilder)
        {
            var config = new HttpConfiguration();

            config.Routes.MapHttpBatchRoute(
                routeName: "Batch",
                routeTemplate: "api/$batch",
                batchHandler: new DefaultHttpBatchHandler(GlobalConfiguration.DefaultServer));

            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "Default",
                routeTemplate: "{controller}/{action}/{name}",
                defaults: new { name = RouteParameter.Optional }
            );

            config.Formatters.Clear();
            config.Formatters.Add(new JsonMediaTypeFormatter());
            config.Formatters.JsonFormatter.SerializerSettings.Formatting =
Newtonsoft.Json.Formatting.Indented;
            config.Formatters.JsonFormatter.SerializerSettings.ContractResolver = new
CamelCasePropertyNamesContractResolver();

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

مهم‌ترین نکته‌ی آن استفاده از DefaultHttpBatchHandler و معرفی آن به بخش batchHandler مسیریابی است. کلاس DefaultHttpBatchHandler برای وهله سازی نیاز به آبجکت سروری که سرویس‌های WebApi در آن هاست شده‌اند دارد که با دستور GlobalConfiguration.DefaultServer به آن دسترسی خواهید داشت. در صورتی که HttpServer خاص خود را دارید به صورت زیر عمل نمایید:

```
var config = new HttpConfiguration();
 HttpServer server = new HttpServer(config);
```

تنظیمات بخش سرور به اتمام رسید. حال نیاز داریم بخش کلاینت را طوری طراحی نماییم که بتواند درخواست را به صورت دسته‌ای ارسال نماید. در زیر یک مثال قرار داده شده است:

```
using System.Net.Http;
using System.Net.Http.Formatting;

public class Program
    {
        private static void Main(string[] args)
        {
            string baseAddress = "http://localhost:8080";
            var client = new HttpClient();
            var batchRequest = new HttpRequestMessage(HttpMethod.Post, baseAddress + "/api/$batch")
            {
                Content = new MultipartContent("mixed")
                {
                    new HttpMessageContent(new HttpRequestMessage(HttpMethod.Post, baseAddress +
"/api/Book/Add")
                    {
                        Content = new ObjectContent<string>("myBook", new JsonMediaTypeFormatter())
                    }),
                    new HttpMessageContent(new HttpRequestMessage(HttpMethod.Get, baseAddress +
"/api/Book/GetAll"))
                }
            };

            var batchResponse = client.SendAsync(batchRequest).Result;

            MultipartStreamProvider streamProvider =
batchResponse.Content.ReadAsMultipartAsync().Result;
            foreach (var content in streamProvider.Contents)
            {
                var response = content.ReadAsHttpResponseMessageAsync().Result;
            }
        }
    }
```

همان طور که می‌دانیم برای ارسال درخواست به سرویس Web Api باید یک نمونه از کلاس HttpRequestMessage وهله سازی شود سازنده‌ی آن به نوع HttpMethod اکشن نظیر (POST یا GET) و آدرس سرویس مورد نظر نیاز دارد. نکته‌ی مهم آن این است که خاصیت Content این درخواست باید از نوع MultipartContent و subType آن نیز باید mixed باشد. در بدنه‌ی آن نیز می‌توان تمام درخواست‌ها را به ترتیب و با استفاده از وهله سازی از کلاس HttpMessageContent تعریف کرد.
برای دریافت پاسخ این گونه درخواست‌ها نیز از متد الحاقی ReadAsMultipartAsync استفاده می‌شود که امکان پیمایش بر بدنه‌ی پیام دریافتی را می‌دهد.

مدیریت ترتیب درخواست ها

شاید این سوال به ذهن شما نیز خطور کرده باشد که ترتیب پردازش این گونه پیام‌ها چگونه خواهد بود؟ به صورت پیش فرض ترتیب اجرای درخواست‌ها حائز اهمیت است. بعنی تا زمانیکه پردازش درخواست اول به اتمام نرسد، کنترل اجرای برنامه، به درخواست بعدی نخواهد رسید که این مورد بیشتر زمانی رخ می‌دهد که قصد دریافت اطلاعاتی را داشته باشید که قبل از آن باید عمل Persist در پایگاه داده اتفاق بیافتد. اما در حالاتی غیر از این می‌توانید این گزینه را غیر فعال کرده تا تمام درخواست‌ها به صورت موازی پردازش شوند که به طور قطع کارایی آن نسبت به حالت قبلی بهینه‌تر است.
برای غیر فعال کردن گزینه‌ی ترتیب اجرای درخواست‌ها، به صورت زیر عمل نمایید:

```
config.Routes.MapHttpBatchRoute(
                routeName: "WebApiBatch",
                routeTemplate: "api/$batch",
                batchHandler: new DefaultHttpBatchHandler(GlobalConfiguration.DefaultServer)
                {
                    ExecutionOrder = BatchExecutionOrder.NonSequential
                });
```

تفاوت آن فقط در مقدار دهی خاصیت ExecutionOrder به صورت NonSequential است.