عنوان: **چرخه‌ی حیات یک درخواست در ASP.NET MVC**
نویسنده: کیانی مقدم علیرضا
تاریخ: ۱۳۹۴/۰۷/۳۰ ۱۶:۵۰
آدرس: www.dotnettips.info
گروه‌ها: ASP.Net, MVC

در Asp.net دو چرخه‌ی حیات مهم وجود دارند که اساس چارچوب MVC را تشکیل می‌دهند :

چرخه‌ی حیات برنامه (Application) ؛ از لحظه‌ای که برنامه برای اولین بار اجرا می‌شود و تا لحظه‌ی خاتمه‌ی آن را شامل می‌شود.
چرخه‌ی حیات یک درخواست (Request)؛ مسیری که یک درخواست طی می‌کند، اصطلاحا PipeLine نامیده می‌شود که همان چرخه‌ی حیات یک درخواست نیز هست و از لحظه‌ای که درخواست تحویل asp.net شده، تا زمانیکه درخواست ارسال می‌شود را شامل می‌شود.

تمرکز بنده بیشتر بر روی روند و مسیری است که یک درخواست طی می‌کند و قصد دارم با بهره گیری از کتاب Pro Asp.net Mvc 5 و دیگر منابع، چرخه‌ی حیات درخواست را در برنامه‌های Mvc بررسی کرده و در مقالات آتی ماژولها و هندلرها را بررسی کنم.

در asp.net ، برنامه global فایلهای شامل دو فایل Global.asax.cs , Global.asax است.

فایل Global.asax که هیچ گاه نیاز به ویرایش آن نداریم محتویاتی مانند زیر دارد:

```
<%@ Application Codebehind="Global.asax.cs" Inherits="YourAppName.MvcApplication" Language="C#" %>
```

و صرفا فایل code behind مرتبط را برای asp.net مشخص می‌کند. این نوع مشخص سازی را از وب فرمها به یاد داریم.
در این مقاله منظور از فایل global فایل Global.asax.cs است که مشتق شده از کلاس System.Web.HttpApplication است :

```
public class MvcApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            ...//
        }
    }
```

به صورت پیش فرض کدهای بالا ایجاد شده و کلاس MvcApplication شامل متد Application_Start است که نقطه‌ی شروع چرخه‌ی حیات برنامه را مشخص می‌کند. اما متد دیگری به نام Application_End() نیز وج ود دارد که در زمان خاتمه‌ی برنامه فراخوانی خواهد شد و فرصتی را برای آزاد سازی منابع اشغال شده توسط برنامه فراهم می‌آورد .
Asp.net برای پاسخگویی به درخواست‌های واصله، وهله‌هایی از کلاس MvcApplication را می‌سازد ولی این دو متد صرفا در نقاط شروع و پایان برنامه فراخوانی شده و عملا در وهله‌های یاد شده صدا زده نخواهند شد و به جای آنها رویدادهایی را که در ذیل آنها را معرفی می‌کنیم، فراخوانی شده و چرخه‌ی حیات درخواست را برای ما مشخص می‌سازند .
BeginRequest : به عنوان اولین رویداد، به محض وصول یک درخواست جدید رخ خواهد داد.

AuthenticateRequest ,PostAuthenticateRequest : رویداد AuthenticateRequest برای شناسایی کاربر ارسال کننده درخواست، کاربرد دارد و پس از پردازش کلیه‌ی توابع، رویداد PostAuthenticateRequest صدا زده می‌شود.
AuthorizeRequest :به‌هنگام صدور مجوزهای یک درخواست رخ می‌دهد و مشابه رویداد بالا پس از پردازش کلیه‌ی توابع، رویداد PostAuthorizeRequest صدا زده خواهد شد.
ResolveRequestCache : پس از صدور مجوزهای یک درخواست در رویداد authorization زمانیکه ماژولهای کش می‌خواهند اطلاعاتی را از کش سرور مطالبه کنند، رخ می‌دهد و به مانند دو رخداد قبلی، PostResolveRequestCache نیز پس از اتمام پردازش توابع رویداد رخ می‌دهد.
MapRequestHandler : زمانی که Asp.net می‌خواهد هندلری را برای پاسخگویی به درخواست واصله انتخاب کند رخ می‌دهد و PostMapRequestHandler نیز پس از این انتخاب، تریگر می‌شود.

AcquireRequestState : جهت بدست آوردن داده‌هایی نظیر سشن و ... مرتبط با درخواست جاری کاربرد داشته و PostAcquireRequestState نیز پس از پردازش توابع رویداد رخ خواهد داد.
PreRequestHandlerExecute : بلافاصله قبل و همچنین بلافاصله بعد از این که یک هندلر بخواهد درخواستی را پردازش کند، رخ می‌دهد. PostRequestHandlerExecute نیز همانند دیگر رویدادهای گذشته، پس از اتمام پردازش توابع، این رویداد رخ خواهد داد.

ReleaseRequestState : زمانی رخ می‌دهد که داده‌های مرتبط با درخواست جاری، در ادامه‌ی روند پردازش درخواست مورد نیاز نباشند و پس از پردازش توابع رویداد، PostReleaseRequestState رخ خواهد داد .

UpdateRequestCache : به جهت اینکه ماژولهای مسئول کش، توانایی به روز رسانی داده‌های خود، برای پاسخگویی به درخواستهای بعدی را داشته باشند، این رویداد رخ می‌دهد.
LogRequest : قبل از انجام عملیات لاگین برای درخواست جاری رخ می‌دهد و پس از پردازش توابع رویداد نیز PostLogRequest تریگر می‌شود.

EndRequest : پس از پایان کار پردازش درخواست جاری و مهیا شدن پاسخ مرتبط جهت ارسال به مرورگر تریگر خواهد شد.

PreSendRequestHeaders : قبل از ارسال HTTP headers به مرورگر این رویداد رخ خواهد داد.
PreSendRequestContent : بعد از ارسال شدن هدرها و قبل از ارسال محتوای صفحه به مرورگر رخ می‌دهد.

Error : هر زمان و در هر مرحله از پردازش درخواست، چنانچه خطایی صورت پذیرد این رویداد رخ خواهد داد.

فریم ورک Asp.net جهت مدیریت بهتر یک درخواست، در تمام مسیر پردازش، رویدادهای بالا را مهیا کرده است. در ادامه نحوه‌ی هندل کردن رویدادهای چرخه‌ی حیات درخواست را در فایل global توضیح می‌دهم. هر چند که استفاده از این فایل بدین منظور، صرفا برای مدیریت مسائل ابتدایی مناسب بوده و در یک پروژه‌ی بزرگ موجب به هم ریختگی فایل global با کدهای زیاد و خوانایی پایین بوده که قابلیت استفاده مجدد در دیگر پروژه‌ها را نیز ندارد.

Asp.net این مشکل را با معرفی ماژولها که در مقالات آتی توضیح خواهم داد، مرتفع کرده است.

در فایل global هر گاه متدی را با پیشوند Application_ و نام یکی از رویدادهای بالا بنویسید Asp.net آن را به عنوان هندلری برای رویداد مذکور می‌شناسد. به عنوان مثال متدی با نام Application_BeginRequest متد رویداد BeginRequest می‌باشد.

ابتدا یک پروژه‌ی MVC جدید را به نام SimpleApp ایجاد کرده و فایل global آن را مطابق ذیل تغییر می‌دهیم:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Routing;
namespace SimpleApp
{
    public class MvcApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            AreaRegistration.RegisterAllAreas();
            RouteConfig.RegisterRoutes(RouteTable.Routes);
        }
        protected void Application_BeginRequest()
        {
            RecordEvent("BeginRequest");
        }
        protected void Application_AuthenticateRequest()
        {
            RecordEvent("AuthenticateRequest");
        }
```

```
        protected void Application_PostAuthenticateRequest()
        {
            RecordEvent("PostAuthenticateRequest");
        }
        private void RecordEvent(string name)
        {
            List<string> eventList = Application["events"] as List<string>;
            if (eventList == null)
            {
                Application["events"] = eventList = new List<string>();
            }
            eventList.Add(name);
        }
    }
}
```

در اینجا متدی به نام RecordEvent را در کدهای ذکر شده مشاهده می‌کنید که نام یک رویداد را دریافت و جهت در دسترس قرار دادن در کل برنامه به خاصیت Application از کلاس HttpApplication نسبت داده و متد مذکور را از سه متد دیگر فراخوانی کرده‌ایم. این متدها در زمان رخ دادن رویدادهای BeginRequest, AuthenticateRequest, PostAuthenticateRequest صدا زده خواهند شد.

حال جهت نمایش اطلاعات رویداد نیاز است تغییراتی مشابه ذیل در کنترلر Home ایجاد نماییم.

```
using System.Web.Mvc;
namespace SimpleApp.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View(HttpContext.Application["events"]);
        }

    }
}
```

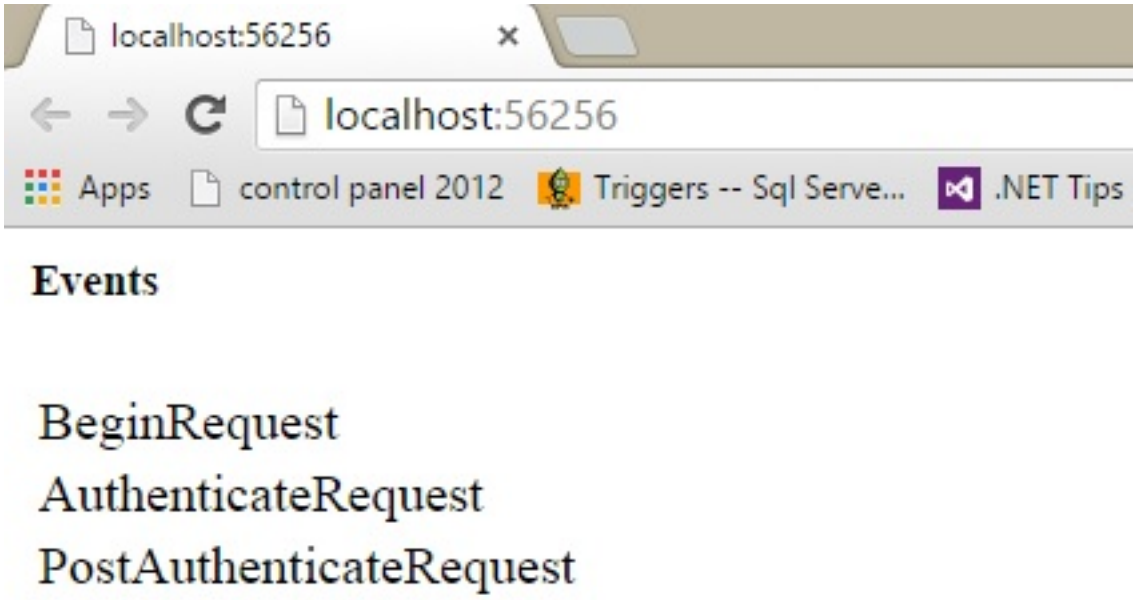
ویوی مرتبط با اکشن متد index را مطابق کدهای ذیل بازنویسی می‌کنیم:

```
@model List<string>
@{
    ViewBag.Title = "Events List";

}
<h5>Events</h5>
<table>
    @foreach (string eventName in Model)
    {
        <tr>
            <td>@eventName</td>
        </tr>
    }

</table>
```

خروجی آن مطابق ذیل خواهد بود :



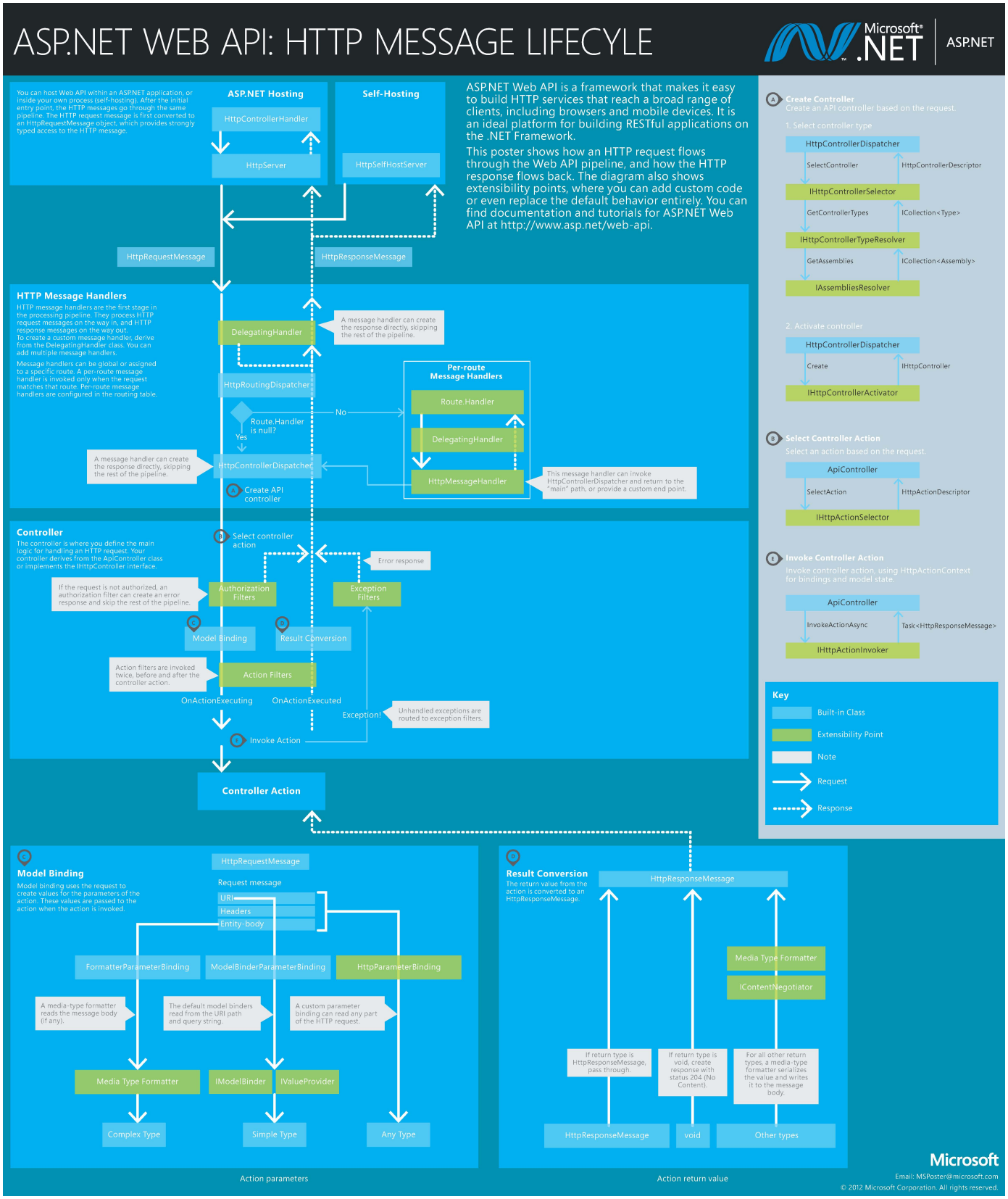
در این مقاله سعی کردیم ابتدا چرخه‌ی حیات یک Request را فرا گرفته و سپس از طریق فایل global و توسط متدهایی با پیشوند Application_ +نام رویداد (اصطلاحا متدهای ویژه نامیده می‌شوند) چرخه حیات یک درخواست را مدیریت کنیم.

نظرات خوانندگان

نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۷/۳۰ ۱۷:۶

ممنون از شما
البته قبلا من هم مقالاتی (+ و +) در همین راستا منتشر کردم و خیلی خوب میشه مقالات شما با در نظر گرفتن آن‌ها حالت تکمیلی‌تر و کاملتری به خود بگیرند.

نویسنده: محسن کریمی
تاریخ: ۱۳۹۴/۰۸/۰۲ ۲۰:۳۸


ASP.NET WEB API: HTTP MESSAGE LIFECYLE
Microsoft .NET
ASP.NET
ASP.NET Hosting
Self-Hosting
HttpControllerHandler
HttpServer
HttpSelfHostServer
HttpRequestMessage
HttpResponseMessage
ASP.NET Web API is a framework that makes it easy to build HTTP services that reach a broad range of clients, including browsers and mobile devices. It is an ideal platform for building RESTful applications on the .NET Framework.
This poster shows how an HTTP request flows through the Web API pipeline, and how the HTTP response flows back. The diagram also shows extensibility points, where you can add custom code or even replace the default behavior entirely. You can find documentation and tutorials for ASP.NET Web API at http://www.asp.net/web-api.
HTTP Message Handlers
DelegatingHandler
HttpRoutingDispatcher
Route.Handler is null?
No
Yes
Per-route Message Handlers
Route.Handler
DelegatingHandler
HttpMessageHandler
HttpControllerDispatcher
Create API controller
Controller
Select controller action
Error response
Authorization Filters
Exception Filters
Model Binding
Result Conversion
Action Filters
OnActionExecuting
OnActionExecuted
Exception!
Invoke Action
Controller Action
Create Controller
HttpControllerDispatcher
SelectController
HttpControllerDescriptor
IHttpControllerSelector
GetControllerTypes
ICollection<Type>
IHttpControllerTypeResolver
GetAssemblies
ICollection<Assembly>
IAssembliesResolver
Create
IHttpController
IHttpControllerActivator
Select Controller Action
ApiController
SelectAction
HttpActionDescriptor
IHttpActionSelector
Invoke Controller Action
InvokeActionAsync
Task<HttpResponseMessage>
IHttpActionInvoker
Key
Built-in Class
Extensibility Point
Note
Request
Response
Model Binding
HttpRequestMessage
Request message
URI
Headers
Entity body
FormatterParameterBinding
ModelBinderParameterBinding
HttpParameterBinding
Media Type Formatter
IModelBinder
IValueProvider
Complex Type
Simple Type
Any Type
Action parameters
Result Conversion
HttpResponseMessage
Media Type Formatter
IContentNegotiator
HttpResponseMessage
void
Other types
Action return value
Microsoft
Email: MSPoster@microsoft.com
© 2012 Microsoft Corporation. All rights reserved.