عنوان: **تزریق خودکار وابستگی‌ها در ASP.NET Web API به همراه رها سازی خودکار منابع IDisposable**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۵/۳۰ ۱۳:۴۶
آدرس: www.dotnettips.info
گروه‌ها: Design patterns, Dependency Injection, IoC

در انتهای مطلب « تزریق خودکار وابستگی‌ها در برنامه‌های ASP.NET MVC » اشاره‌ای کوتاه به روش DependencyResolver توکار Web API شد که این روش پس از بررسی‌های بیشتر ( ^ و ^ ) به دلیل ماهیت service locator بودن آن و همچنین از دست دادن Context جاری Web API، مردود اعلام شده و استفاده از IHttpControllerActivator توصیه می‌گردد. در ادامه این روش را توسط Structure map 3 پیاده سازی خواهیم کرد.

### پیش نیازها

- شروع یک پروژه‌ی جدید وب با پشتیبانی از Web API
- نصب دو بسته‌ی نیوگت مرتبط با Structure map 3

```
PM>install-package structuremap
PM>install-package structuremap.web
```

### پیاده سازی IHttpControllerActivator توسط Structure map

```
using System;
using System.Net.Http;
using System.Web.Http.Controllers;
using System.Web.Http.Dispatcher;
using StructureMap;

namespace WebApiDISample.Core
{
    public class StructureMapHttpControllerActivator : IHttpControllerActivator
    {
        private readonly IContainer _container;
        public StructureMapHttpControllerActivator(IContainer container)
        {
            _container = container;
        }

        public IHttpController Create(
                HttpRequestMessage request,
                HttpControllerDescriptor controllerDescriptor,
                Type controllerType)
        {
            var nestedContainer = _container.GetNestedContainer();
            request.RegisterForDispose(nestedContainer);
            return (IHttpController)nestedContainer.GetInstance(controllerType);
        }
    }
}
```

در اینجا نحوه‌ی پیاده سازی IHttpControllerActivator را توسط StructureMap ملاحظه می‌کنید.
نکته‌ی مهم آن استفاده از NestedContainer آن است. معرفی آن به متد request.RegisterForDispose سبب می‌شود تا کلیه کلاس‌های IDisposable نیز در پایان کار به صورت خودکار رها سازی شده و نشتی حافظه رخ ندهد.

### معرفی StructureMapHttpControllerActivator به برنامه

فایل WebApiConfig.cs را گشوده و تغییرات ذیل را در آن اعمال کنید:

```
using System.Web.Http;
using System.Web.Http.Dispatcher;
using StructureMap;
using WebApiDISample.Core;
using WebApiDISample.Services;
```

```
namespace WebApiDISample
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            // IoC Config
            ObjectFactory.Configure(c => c.For<IEmailsService>().Use<EmailsService>());

            var container = ObjectFactory.Container;
            GlobalConfiguration.Configuration.Services.Replace(
                typeof(IHttpControllerActivator), new StructureMapHttpControllerActivator(container));


            // Web API routes
            config.MapHttpAttributeRoutes();
            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}
```

در ابتدا تنظیمات متداول کلاس‌ها و اینترفیس‌ها صورت می‌گیرد. سپس نحوه‌ی معرفی StructureMapHttpControllerActivator را به GlobalConfiguration.Configuration.Services مخصوص Web API ملاحظه می‌کنید. این مورد سبب می‌شود تا به صورت خودکار کلیه وابستگی‌های مورد نیاز یک Web API Controller به آن تزریق شوند.

**تهیه سرویسی برای آزمایش برنامه**

```
namespace WebApiDISample.Services
{
    public interface IEmailsService
    {
        void SendEmail();
    }
}

using System;

namespace WebApiDISample.Services
{
    /// <summary>
    /// سرویسی که دارای قسمت دیسپوز نیز هست
    /// </summary>
    public class EmailsService : IEmailsService, IDisposable
    {
        private bool _disposed;

        ~EmailsService()
        {
            Dispose(false);
        }

        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }

        public void SendEmail()
        {
            //todo: send email!
        }

        protected virtual void Dispose(bool disposeManagedResources)
        {
            if (_disposed) return;
            if (!disposeManagedResources) return;

            //todo: clean up resources here ...
```

```
            _disposed = true;
        }
    }
}
```

در اینجا یک سرویس ساده ارسال ایمیل را بدون پیاده سازی خاصی مشاهده می‌کنید.
نکته‌ی مهم آن استفاده از IDisposable در این کلاس خاص است (ضروری نیست؛ صرفا جهت بررسی بیشتر اضافه شده‌است).
اگر در کدهای برنامه، یک چنین کلاسی وجود داشت، نیاز است متد Dispose آن نیز توسط IoC Container فراخوانی شود. برای آزمایش آن یک break point را در داخل متد Dispose قرار دهید.

**استفاده از سرویس تعریف شده در یک Web API Controller**

```
using System.Web.Http;
using WebApiDISample.Services;

namespace WebApiDISample.Controllers
{
    public class ValuesController : ApiController
    {
        private readonly IEmailsService _emailsService;
        public ValuesController(IEmailsService emailsService)
        {
            _emailsService = emailsService;
        }

        // GET api/values/5
        public string Get(int id)
        {
            _emailsService.SendEmail();
            return "_emailsService.SendEmail(); called!";
        }
    }
}
```

در اینجا مثال ساده‌ای را از نحوه‌ی تزریق سرویس ارسال ایمیل را در ValuesController مشاهده می‌کنید.
تزریق وهله‌ی مورد نیاز آن، به صورت خودکار توسط StructureMapHttpControllerActivator که در ابتدای بحث معرفی شد، صورت می‌گیرد.

فراخوانی متد Get آن‌را نیز توسط کدهای سمت کاربر ذیل انجام خواهیم داد:

```
<h2>Index</h2>

@section scripts
{
    <script type="text/javascript">
        $(function () {
            $.getJSON('/api/values/1?timestamp=' + new Date().getTime(), function (data) {
                alert(data);
            });
        });
    </script>
}
```

درون متد Get کنترلر، یک break point قرار دهید. همچنین داخل متد Dispose لایه سرویس نیز جهت بررسی بیشتر یک break point قرار دهید.
اکنون برنامه را اجرا کنید. هنگام فراخوانی متد Get، وهله‌ی سرویس مورد نظر، نال نیست. همچنین متد Dispose نیز به صورت خودکار فراخوانی می‌شود.

**کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید**

WebApiDISample.zip