

همانطور که پیشتر در [این مقاله](#) بحث شده است، بوسیله AOP می‌توان قابلیت‌هایی که قسمت عمده‌ای از برنامه را تحت پوشش قرار می‌دهند، کپسوله کرد. یکی از قابلیت‌هایی که در بخشهای مختلف یک سیستم نرم‌افزاری مورد نیاز است، Authorization یا اعتبارسنجی‌ست. در ادامه به بررسی یک پایاده‌سازی به این روش می‌پردازیم.

کتابخانه SNAP

[کتابخانه SNAP](#) به گفته سازنده آن، با یکپارچه‌سازی AOP با IoC Containerهای محبوب، برنامه‌نویسی به این سبک را ساده می‌کند. این کتابخانه هم اکنون علاوه بر structureMap از IoC Providerهای LinFu، Ninject، Autofac و Castle Windsor نیز پشتیبانی میکند.

دریافت SNAP.StructureMap

برای دریافت آن نیاز است دستور پاورشل ذیل را در کنسول [نیوگت](#) ویژوال استودیو اجرا کنید:

```
PM> Install-Package snap.structuremap
```

پس از اجرای دستور فوق، کتابخانه SNAP.StructureMap که در زمان نگارش این مطلب نسخه 1.8.0 آن موجود است به همراه کليه نیازمندی‌های آن که شامل موارد زیر می‌باشد نصب خواهد شد.

```
StructureMap (≥ 2.6.4.1)
CommonServiceLocator.StructureMapAdapter (≥ 1.1.0.3)
SNAP (≥ 1.8)
fasterflect (≥ 2.1.2)
Castle.Core (≥ 3.1.0)
CommonServiceLocator (≥ 1.0)
```

تنظیمات SNAP

از آنجا که تنظیمات SNAP همانند تنظیمات StructureMap تنها باید یک بار اجرا شود، بهترین جا برای آن در یک برنامه وب، Application_Start فایل Global.asax است.

```
namespace Framework.UI.Asp
{
    public class Global : HttpApplication
    {
        void Application_Start(object sender, EventArgs e)
        {
            initSnap();
            initStructureMap();
        }

        private static void initSnap()
        {
            SnapConfiguration.For<StructureMapAspectContainer>(c =>
            {
                // Tell Snap to intercept types under the "Framework.ServiceLayer..." namespace.
                c.IncludeNamespace("Framework.ServiceLayer.*");
                // Register a custom interceptor (a.k.a. an aspect).
                c.Bind<Framework.ServiceLayer.Aspects.AuthorizationInterceptor>()
                    .To<Framework.ServiceLayer.Aspects.AuthorizationAttribute>();
            });
        }

        void Application_EndRequest(object sender, EventArgs e)
        {
            ObjectFactory.ReleaseAndDisposeAllHttpScopedObjects();
        }
    }
}
```

```

    }
    private static void initStructureMap()
    {
        var thread = StructureMap.Pipeline.Lifecycles.GetLifecycle(InstanceScope.HttpSession);
        ObjectFactory.Configure(x =>
        {
            x.For<IUserManager>().Use<EFUserManager>();
            x.For<IAuthorizationManager>().LifecycleIs(thread)
              .Use<EFAuthorizationManager>().Named("AuthorizationManager");
            x.For<Framework.DataLayer.IUnitOfWork>()
              .Use<Framework.DataLayer.Context>();

            x.SetAllProperties(y =>
            {
                y.OfType<IUserManager>();
                y.OfType<Framework.DataLayer.IUnitOfWork>();
                y.OfType<Framework.Common.Web.IPageHelpers>();
            });
        });
    }
}

```

بخش اعظم کدهای فوق در مقاله‌های « [استفاده از StructureMap به عنوان یک IoC Container](#) » و « [تزریق خودکار وابستگی‌ها در برنامه‌های ASP.NET Web forms](#) » شرح داده شده‌اند، تنها بخش جدید متد `initSnap()` است، که خط اول آن به `snap` می‌گوید همه کلاس‌هایی که در فضای نام `Framework.ServiceLayer` و زیرمجموعه‌های آن هستند را پوشش دهد. خط دوم نیز کلاس `AuthorizationInterceptor` را به عنوان مرجعی برای `handle` کردن `AuthorizationAttribute` معرفی می‌کند.

در ادامه به بررسی کلاس `AuthorizationInterceptor` می‌پردازیم.

```

namespace Framework.ServiceLayer.Aspects
{
    public class AuthorizationInterceptor : MethodInterceptor
    {
        public override void InterceptMethod(IInvocation invocation, MethodBase method, Attribute attribute)
        {
            var AuthManager = StructureMap.ObjectFactory
                .GetInstance<Framework.ServiceLayer.UserManager.IAuthorizationManager>();
            var FullName = GetMethodFullName(method);
            if (!AuthManager.IsActionAuthorized(FullName))
                throw new Common.Exceptions.UnauthorizedAccessException("");

            invocation.Proceed(); // the underlying method call
        }

        private static string GetMethodFullName(MethodBase method)
        {
            var TypeName = (((System.Reflection.MemberInfo)(method)).DeclaringType).FullName;
            return TypeName + "." + method.Name;
        }
    }

    public class AuthorizationAttribute : MethodInterceptAttribute
    {
    }
}

```

کلاس مذکور از کلاس `MethodInterceptor` کتابخانه `snap` ارث بری کرده و متد `InterceptMethod` را تحریف می‌کند. این متد، کار اجرای متد اصلی ای که با این `Aspect` تزئین شده را بر عهده دارد. بنابراین می‌توان پیش از اجرای متد اصلی، اعتبارسنجی را انجام داد. **کلاس `MethodInterceptor`**

کلاس `MethodInterceptor` شامل چندین متد دیگر نیز هست که میتوان برای سایر مقاصد از جمله مدیریت خطا و `Event` `logging` از آنها استفاده کرد.

```

namespace Snap
{

```

```
public abstract class MethodInterceptor : IAttributeInterceptor, IInterceptor, IHideBaseTypes
{
    protected MethodInterceptor();

    public int Order { get; set; }
    public Type TargetAttribute { get; set; }

    public virtual void AfterInvocation();
    public virtual void BeforeInvocation();
    public void Intercept(IInvocation invocation);
    public abstract void InterceptMethod(IInvocation invocation, MethodBase method, Attribute
attribute);
    public bool ShouldIntercept(IInvocation invocation);
}
}
```

یک نکته

نکته مهمی که در اینجا پیش می آید این است که برای اعتبارسنجی، کد کاربری شخصی که لاگین کرده، باید به طریقی در اختیار متد `IsActionAuthorized()` قرار بگیرد. برای این کار می توان در یک `HttpMudole` به عنوان مثال همان مازولی که برای تسهیل در کار تزریق خودکار وابستگی ها در سطح فرم ها استفاده می شود، با استفاده از امکانات `structureMap` به وهله ی ایجاد شده از `AuthorizationManager` (که با کمک `structureMap` با طول عمر `InstanceScope.HttpSession` ساخته شده است) دسترسی پیدا کرده و خاصیت مربوطه را مقداردهی کرد.

```
private void Application_PreRequestHandlerExecute(object source, EventArgs e)
{
    var page = HttpContext.Current.Handler as BasePage; // The Page handler
    if (page == null)
        return;

    WireUpThePage(page);
    WireUpAllUserControls(page);

    var Usrcod = HttpContext.Current.Session["Usrcod"];
    if (Usrcod != null)
    {
        var _AuthorizationManager = ObjectFactory
.GetNamedInstance<Framework.ServiceLayer.UserManager.IAuthorizationManager>("_AuthorizationManager");

        ((Framework.ServiceLayer.UserManager.EFAuthorizationManager)_AuthorizationManager)
.AuditUserId = Usrcod.ToString();
    }
}
```

روش استفاده

نحوه استفاده از Aspect تعریف شده در کد زیر قابل مشاهده است:

```
namespace Framework.ServiceLayer.UserManager
{
    public class EFUserManager : IUserManager
    {
        IUnitOfWork _uow;
        IDbSet<User> _users;

        public EFUserManager(IUnitOfWork uow)
        {
            _uow = uow;
            _users = _uow.Set<User>();
        }

        [Framework.ServiceLayer.Aspects.Authorization]
        public List<User> GetAll()
        {
            return _users.ToList<User>();
        }
    }
}
```

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۸/۰۱ ۱۹:۷

با تشکر از شما. چند سؤال: متد AuthManager.IsActionAuthorized چگونه تعریف شده؟ و همچنین EFAuthorizationManager حدوداً چه تعریفی دارد؟

نویسنده: کاوه شهبازی
تاریخ: ۱۳۹۲/۰۸/۰۱ ۱۹:۵۱

۱- متد IsActionAuthorized نام کامل متدی که قرار است اجرا شود را به عنوان پارامتر گرفته و در دیتابیس (در این پیاده سازی به وسیله EntityFramework) چک میکند که کاربری که Id اش در AuthManager.AuditUserId است (یعنی کاربری که درخواست اجرای متد را داده است) اجازه اجرای این متد را دارد یا نه. بسته به نیازمندی برنامه شما این دسترسی میتواند به طور ساده فقط مستقیماً برای کاربر ثبت شود و یا ترکیبی از دسترسی خود کاربر و دسترسی گروه هایی که این کاربر در آن عضویت دارد باشد.

۲- EFAuthorizationManager کلاس ساده ایست

```
namespace Framework.ServiceLayer.UserManager
{
    public class EFAuthorizationManager : IAuthorizationManager
    {
        public String AuditUserId { get; set; }
        IUnitOfWork _uow;

        public EFAuthorizationManager(IUnitOfWork uow)
        {
            _uow = uow;
        }

        public bool IsActionAuthorized(string actionName)
        {
            var res = _uow.Set<User>()
                .Any(u => u.Id == AuditUserId &&
                    u.AllowedActions.Any(a => a.Name == actionName));
            return res;
        }

        public bool IsPageAuthorized(string pageURL)
        {
            //TODO: بررسی وجود دسترسی باید پیاده سازی شود
            //فقط برای تست
            return true;
        }
    }
}
```

خلاصه ای از کلاسهای مدل مرتبط را هم در زیر مشاهده می کنید

```
namespace Framework.DataModel
{
    public class User : BaseEntity
    {
        public string UserName { get; set; }
        public string Password { get; set; }

        //...

        [Display(Name = "عملیات مجاز")]
        public virtual ICollection<Action> AllowedActions { get; set; }
    }

    public class Action:BaseEntity
    {
        public string Name { get; set; }
        public Entity RelatedEntity { get; set; }
    }
}
```

```
//...  
    public virtual ICollection<User> AllowedUsers { get; set; }  
}  
public abstract class BaseEntity  
{  
    [Key]  
    public int Id { get; set; }  
    //...  
}  
}
```

هنگامی که از روش [AOP](#) استفاده می‌کنیم گاهی نیاز است متد تزئین‌شده را از متدی درون خود کلاس فراخوانی کنیم و می‌خواهیم aspectهای آن متد نیز فراخوانی شوند.

پیش‌نیاز: دوره‌ی AOP

(برای سادگی کار از تعریف attribute خودداری کردم. شما می‌توانید با توجه به آموزش، attributeهای دلخواه را به متدها بیافزایید).

Interface و کلاس پیاده‌سازی‌شده‌ی آن در لایه سرویس:

```
public interface IMyService
{
    void foo();
    void bar();
}

public class MyService : IMyService
{
    public void foo()
    {
        Console.WriteLine("foo");
        bar();
    }

    public void bar()
    {
        Console.WriteLine("bar");
    }
}
```

نام متد در خروجی نوشته می‌شود. همچنین می‌خواهیم پیش از فراخوانی این متدها، متنی در خروجی نوشته شود.

آماده‌سازی `Interceptor`

یک `interceptor` ساده که نام متد را در خروجی می‌نویسد.

```
//using Castle.DynamicProxy;

public class Interceptor : IInterceptor
{
    public void Intercept(IInvocation invocation)
    {
        Console.WriteLine("Intercepted: " + invocation.Method.Name);
        invocation.Proceed();
    }
}
```

معرفی `Interceptor` به سیستم

همانند قبل:

```
//using System;
//using Castle.DynamicProxy;
//using StructureMap;

class Program
{
    static void Main(string[] args)
    {
        ObjectFactory.Initialize(x =>
        {
            var dynamicProxy = new ProxyGenerator();
            x.For<IMyService>()
                .EnrichAllWith(myTypeInterface =>
                    dynamicProxy.CreateInterfaceProxyWithTarget(myTypeInterface, new
Intercept()))
                .Use<MyService>();
        });
    }
}
```

```

    });
    var myService = ObjectFactory.GetInstance<IMyService>();
    myService.foo();
}
}

```

انتظار ما این است که خروجی زیر تولید شود:

```

Intercepted foo
foo
Intercepted bar
bar

```

اما نتیجه این می‌شود که دلخواه ما نیست:

```

Intercepted foo
foo
bar

```

راه حل

برای حل این مشکل دو کار باید انجام داد:
1- متد تزئین‌شده باید **virtual** باشد.

```

public class MyService : IMyService
{
    public virtual void foo()
    {
        Console.WriteLine("foo");
        bar();
    }

    public virtual void bar()
    {
        Console.WriteLine("foo");
        bar();
    }
}

```

2- شیوه معرفی متد به سیستم باید به روش زیر باشد:

```

// جایگزین روش پیشین در متد
x.For<IMyService>()
    .EnrichAllWith(myTypeInterface => dynamicProxy.CreateClassProxy<MyService>(new
Intercept()))
    .Use<MyService>();

```

دلیل این مسئله به دو روش proxy برمی‌گردد که برای اطلاع بیشتر به [مستندات پروژه Castle](#) مراجعه کنید.
در این‌جا روش Inheritance-based به کار رفته است. در این روش، تنها متدهای **virtual** را می‌توان intercept کرد. در روش پیشین (Composition-based) برای تمامی متدها عملیات intercept انجام می‌شد (کلاس proxy پیاده‌سازی‌شده‌ی interface ما بود) که در این‌جا این‌گونه نیست و می‌تواند به سرعت برنامه کمک کند.

در طی چند قسمت، نحوه‌ی طراحی یک سیستم افزونه پذیر را با ASP.NET MVC بررسی خواهیم کرد. عناوین مواردی که در این سری پیاده سازی خواهند شد به ترتیب ذیل هستند:

- 1- چگونه Areaهای استاندارد را تبدیل به یک افزونه‌ی مجزا و منتقل شده‌ی به یک اسمبلی دیگر کنیم.
- 2- چگونه ساختار پایه‌ای را جهت تامین نیازهای هر افزونه جهت تزریق وابستگی‌ها تا ثبت مسیریابی‌ها و امثال آن تدارک ببینیم.
- 3- چگونه فایل‌های JS ، CSS و همچنین تصاویر ثابت هر افزونه را داخل اسمبلی آن قرار دهیم تا دیگر نیازی به ارائه‌ی مجزای آن‌ها نباشد.
- 4- چگونه Entity Framework Code-First را با این طراحی یکپارچه کرده و از آن جهت یافتن خودکار مدل‌ها و موجودیت‌های خاص هر افزونه استفاده کنیم؛ به همراه مباحث Migrations خودکار و همچنین پیاده سازی الگوی واحد کار.

در مطلب جاری، موارد اول و دوم بررسی خواهند شد. پیشنیازهای آن مطالب ذیل هستند:

(الف) [منظور از یک Area چیست؟](#)

(ب) [توزیع پروژه‌های ASP.NET MVC بدون ارائه فایل‌های View آن](#)

(ج) [آشنایی با تزریق وابستگی‌ها در ASP.NET MVC](#) و همچنین [اصول طراحی یک سیستم افزونه پذیر به کمک StructureMap](#)

(د) [آشنایی با رخدادهای Build](#)

تبدیل یک Area به یک افزونه‌ی مستقل

روش‌های زیادی برای خارج کردن Areaهای استاندارد ASP.NET MVC از یک پروژه و قرار دادن آن‌ها در اسمبلی‌های دیگر وجود دارند؛ اما در حال حاضر تنها روشی که نگهداری می‌شود و همچنین اعضای آن همان اعضای تیم نیوگت و ASP.NET MVC هستند، همان روش استفاده از [Razor Generator](#) است.

بنابراین ساختار ابتدایی پروژه‌ی افزونه پذیر ما به صورت ذیل خواهد بود:

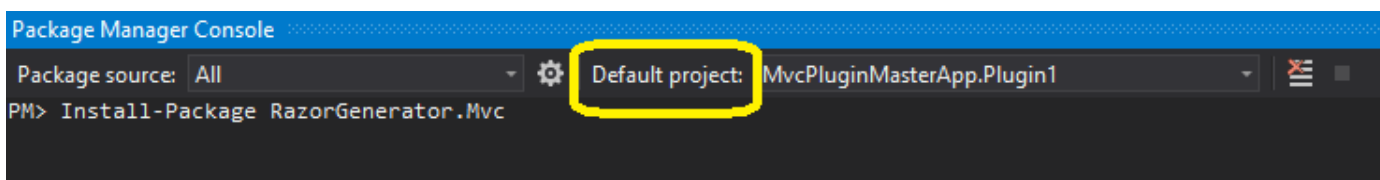
- 1) ابتدا افزونه‌ی [Razor Generator](#) را نصب کنید.
- 2) سپس یک پروژه‌ی معمولی ASP.NET MVC را آغاز کنید. در این سری نام MvcPluginMasterApp برای آن در نظر گرفته شده‌است.

- 3) در ادامه یک پروژه‌ی معمولی دیگر ASP.NET MVC را نیز به پروژه‌ی جاری اضافه کنید. برای مثال نام آن در اینجا MvcPluginMasterApp.Plugin1 تنظیم شده‌است.

- 4) به پروژه‌ی MvcPluginMasterApp.Plugin1 یک Area جدید و معمولی را به نام NewsArea اضافه کنید.
- 5) از پروژه‌ی افزونه، تمام پوشه‌های غیر Area را حذف کنید. پوشه‌های Controllers و Models و Views حذف خواهند شد. همچنین فایل global.asax آنرا نیز حذف کنید. هر افزونه، کنترلرها و Viewهای خود را از طریق Area مرتبط دریافت می‌کند و در این حالت دیگر نیازی به پوشه‌های Controllers و Models و Views واقع شده در ریشه‌ی اصلی پروژه‌ی افزونه نیست.
- 6) در ادامه کنسول پاور شل نیوگت را باز کرده و دستور ذیل را صادر کنید:

```
PM> Install-Package RazorGenerator.Mvc
```

این دستور را باید یکبار بر روی پروژه‌ی اصلی و یکبار بر روی پروژه‌ی افزونه، اجرا کنید.

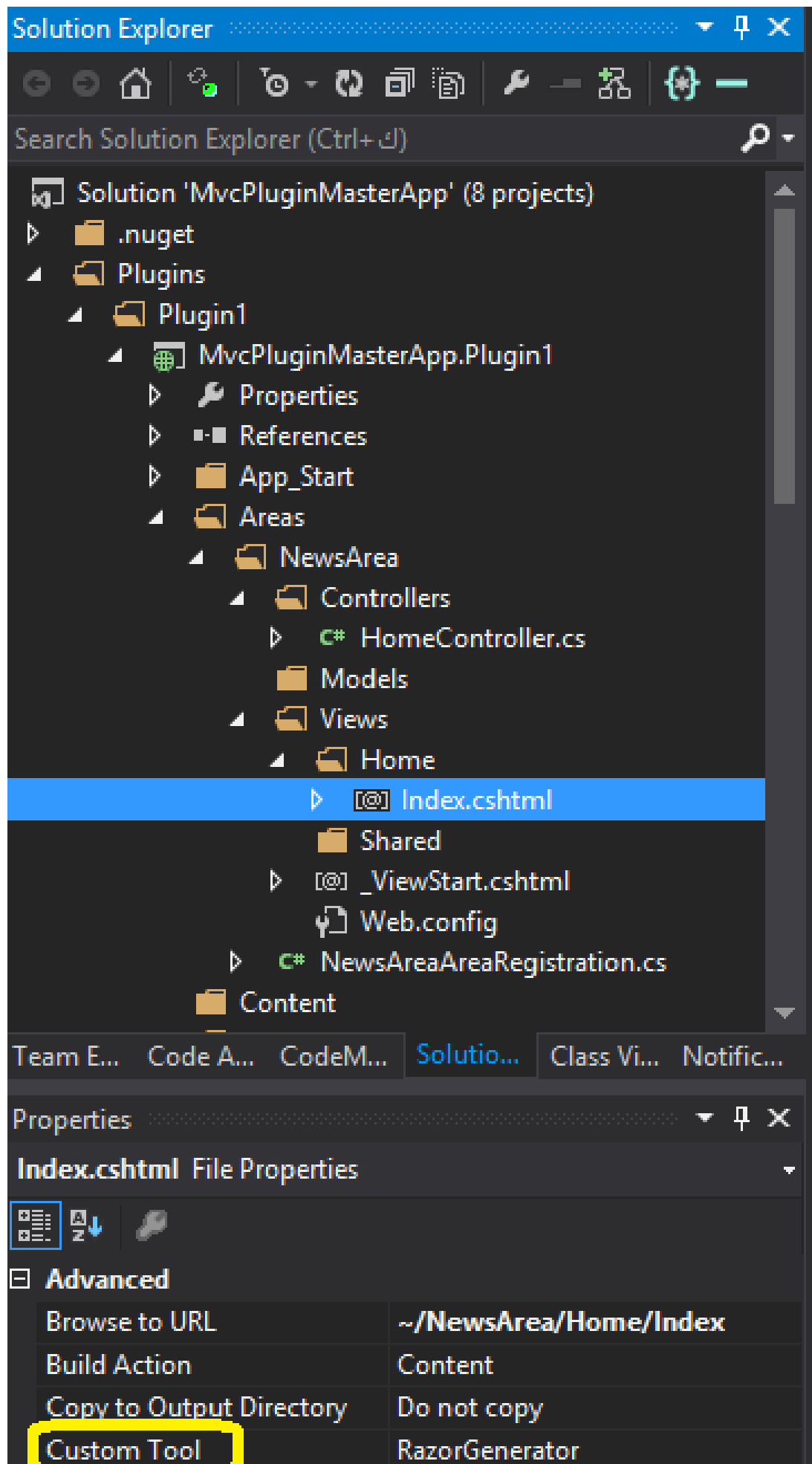


همانطور که در تصویر نیز مشخص شده است، برای اجرای دستور نصب RazorGenerator.Mvc نیاز است هربار پروژه‌ی پیش فرض را تغییر دهید.

(7) اکنون پس از نصب RazorGenerator.Mvc، نوبت به اجرای آن بر روی هر دو پروژه‌ی اصلی و افزونه است:

```
PM> Enable-RazorGenerator
```

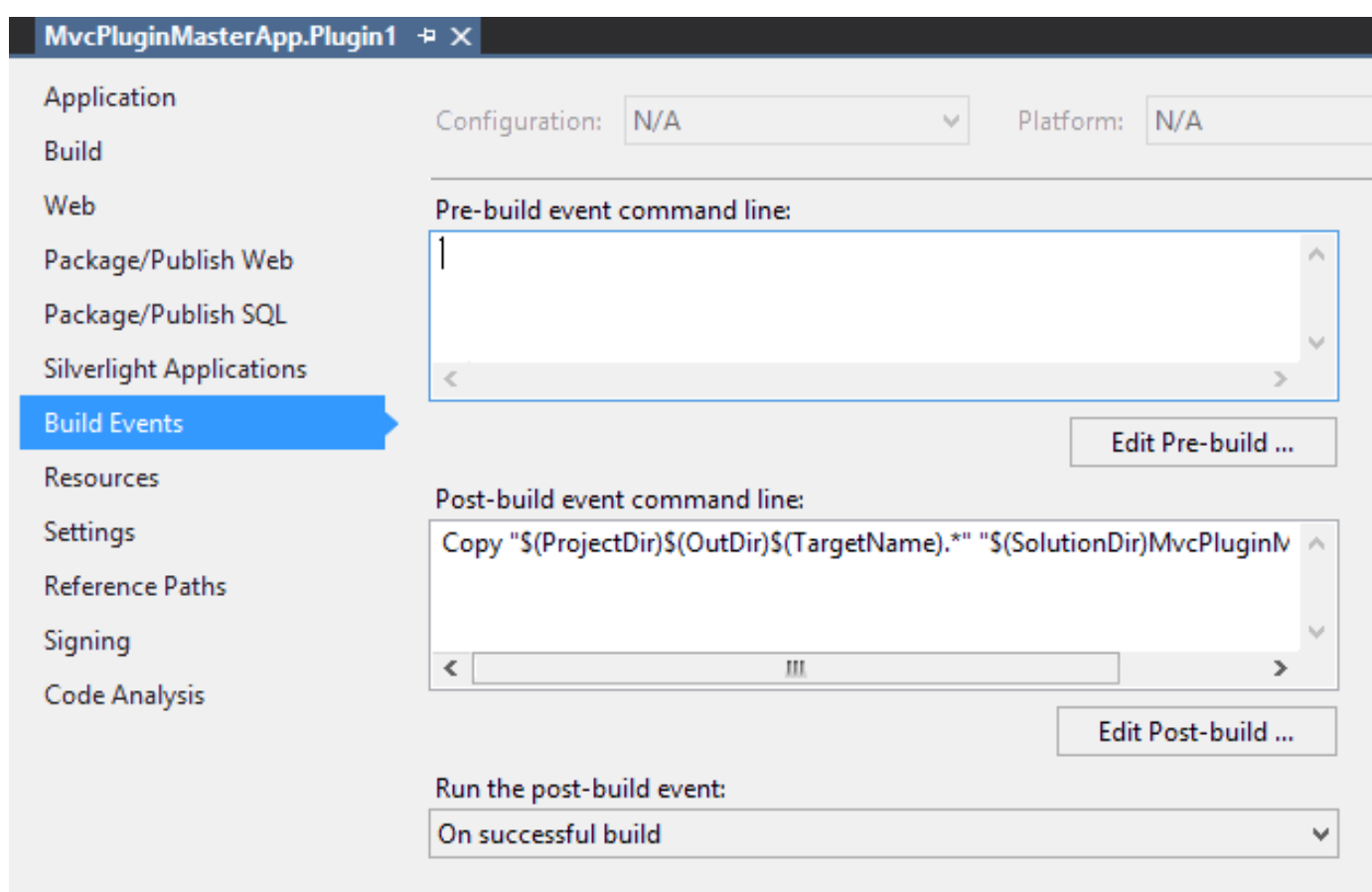
بدیهی است این دستور را نیز باید همانند تصویر فوق، یکبار بر روی پروژه‌ی اصلی و یکبار بر روی پروژه‌ی افزونه اجرا کنید. همچنین هربار که View جدیدی اضافه می‌شود نیز باید این کار را تکرار کنید یا اینکه مطابق شکل زیر، به خواص View جدید مراجعه کرده و Custom tool آن را به صورت دستی به RazorGenerator تنظیم نمائید. دستور Enable-RazorGenerator این کار را به صورت خودکار انجام می‌دهد.



تا اینجا موفق شدیم View های افزونه را داخل فایل dll آن مدفون کنیم. به این ترتیب با کپی کردن افزونه به پوشه‌ی bin پروژه‌ی اصلی، دیگر نیازی به ارائه‌ی فایل‌های View آن نیست و تمام اطلاعات کنترلرها، مدل‌ها و View ها به صورت یکجا از فایل dll افزونه‌ی ارائه شده خوانده می‌شوند.

کپی کردن خودکار افزونه به پوشه‌ی Bin پروژه‌ی اصلی

پس از اینکه ساختار اصلی کار شکل گرفت، هربار پس از کامپایل افزونه (یا افزونه‌ها)، نیاز است فایل‌های پوشه‌ی bin آن را به پوشه‌ی bin پروژه‌ی اصلی کپی کنیم (پروژه‌ی اصلی در این حالت هیچ ارجاع مستقیمی را به افزونه‌ی جدید نخواهد داشت). برای خودکار سازی این کار، به خواص پروژه‌ی افزونه مراجعه کرده و قسمت Build events آن را به نحو ذیل تنظیم کنید:



در اینجا دستور ذیل در قسمت Post-build event نوشته شده است:

```
Copy "$(ProjectDir)$(OutDir)$(TargetName).*" "$(SolutionDir)MvcPluginMasterApp\bin\"
```

و سبب خواهد شد تا پس از هر کامپایل موفق، فایل‌های اسمبلی افزونه به پوشه‌ی bin پروژه‌ی MvcPluginMasterApp به صورت خودکار کپی شوند.

تنظیم فضا‌های نام کلیه مسیرهای پروژه

در همین حالت اگر پروژه را اجرا کنید، موتور ASP.NET MVC به صورت خودکار اطلاعات افزونه‌ی کپی شده به پوشه‌ی bin را دریافت و به Application domain جاری اعمال می‌کند؛ برای اینکار نیازی به کد نویسی اضافه‌تری نیست و خودکار است. برای آزمایش آن فقط کافی است یک break point را داخل کلاس RazorGeneratorMvcStart افزونه قرار دهید. اما ... پس از اجرا، بلافاصله پیام تداخل فضاهای نام را دریافت می‌کنید. خطاهای حاصل عنوان می‌کند که در App domain جاری، دو کنترلر Home وجود دارند؛ یکی در پروژه‌ی اصلی و دیگری در پروژه‌ی افزونه و مشخص نیست که مسیریابی‌ها باید به کدامیک ختم شوند.

برای رفع این مشکل، به فایل NewsAreaAreaRegistration.cs پروژه‌ی افزونه مراجعه کرده و مسیریابی آن‌را به نحو ذیل تکمیل کنید تا فضای نام اختصاصی این Area صریحاً مشخص گردد.

```
using System.Web.Mvc;

namespace MvcPluginMasterApp.Plugin1.Areas.NewsArea
{
    public class NewsAreaAreaRegistration : AreaRegistration
    {
        public override string AreaName
        {
            get
            {
                return "NewsArea";
            }
        }

        public override void RegisterArea(AreaRegistrationContext context)
        {
            context.MapRoute(
                "NewsArea_default",
                "NewsArea/{controller}/{action}/{id}",
                // تکمیل نام کنترلر پیش فرض
                new { controller = "Home", action = "Index", id = UrlParameter.Optional },
                // مشخص کردن فضای نام مرتبط جهت جلوگیری از تداخل با سایر قسمت‌های برنامه
                namespaces: new[] { string.Format("{0}.Controllers", this.GetType().Namespace) }
            );
        }
    }
}
```

همینکار را باید در پروژه‌ی اصلی و هر پروژه‌ی افزونه‌ی جدیدی نیز تکرار کرد. برای مثال به فایل RouteConfig.cs پروژه‌ی اصلی مراجعه کرده و تنظیم ذیل را اعمال نمایید:

```
using System.Web.Mvc;
using System.Web.Routing;

namespace MvcPluginMasterApp
{
    public class RouteConfig
    {
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional },
                // مشخص کردن فضای نام مرتبط جهت جلوگیری از تداخل با سایر قسمت‌های برنامه
                namespaces: new[] { string.Format("{0}.Controllers", typeof(RouteConfig).Namespace) }
            );
        }
    }
}
```

بدون تنظیم فضاهای نام هر مسیریابی، امکان استفاده‌ی بهینه و بدون خطا از Areaها وجود نخواهد داشت.

طراحی قرارداد پایه افزونه‌ها

تا اینجا با نحوه‌ی تشکیل ساختار هر پروژه‌ی افزونه آشنا شدیم. اما هر افزونه در آینده نیاز به مواردی مانند منوی اختصاصی در منوی اصلی سایت، تنظیمات مسیریابی اختصاصی، تنظیمات EF و امثال آن نیز خواهد داشت. به همین منظور، یک پروژه‌ی class library جدید را به نام MvcPluginMasterApp.PluginsBase آغاز کنید. سپس قرار داد IPlugin را به نحو ذیل به آن اضافه نمایید:

```
using System;
using System.Reflection;
using System.Web.Optimization;
using System.Web.Routing;
using StructureMap;

namespace MvcPluginMasterApp.PluginsBase
{
    public interface IPlugin
    {
        EfBootstrapper GetEfBootstrapper();
        MenuItem GetMenuItem(RequestContext requestContext);
        void RegisterBundles(BundleCollection bundles);
        void RegisterRoutes(RouteCollection routes);
        void RegisterServices(IContainer container);
    }

    public class EfBootstrapper
    {
        /// <summary>
        /// Assemblies containing EntityTypeConfiguration classes.
        /// </summary>
        public Assembly[] ConfigurationsAssemblies { get; set; }

        /// <summary>
        /// Domain classes.
        /// </summary>
        public Type[] DomainEntities { get; set; }

        /// <summary>
        /// Custom Seed method.
        /// </summary>
        //public Action<IUnitOfWork> DatabaseSeeder { get; set; }
    }

    public class MenuItem
    {
        public string Name { get; set; }
        public string Url { get; set; }
    }
}
```

پروژه‌ی این قرارداد برای کامپایل شدن، نیاز به بسته‌های نیوگت ذیل دارد:

```
PM> install-package EntityFramework
PM> install-package Microsoft.AspNet.Web.Optimization
PM> install-package structuremap.web
```

همچنین باید به صورت دستی، در قسمت ارجاعات پروژه، ارجاعی را به اسمبلی استاندارد System.Web نیز به آن اضافه نمایید.

توضیحات قرار داد IPlugin

از این پس هر افزونه باید دارای کلاسی باشد که از اینترفیس IPlugin مشتق می‌شود. برای مثال فعلا کلاس ذیل را به افزونه‌ی پروژه اضافه نمایید:

```
using System.Web.Mvc;
using System.Web.Optimization;
using System.Web.Routing;
using MvcPluginMasterApp.PluginsBase;
using StructureMap;

namespace MvcPluginMasterApp.Plugin1
{
    public class Plugin1 : IPlugin
```

```

{
    public EfBootstrapper GetEfBootstrapper()
    {
        return null;
    }

    public MenuItem GetMenuItem(RequestContext requestContext)
    {
        return new MenuItem
        {
            Name = "Plugin 1",
            Url = new UrlHelper(requestContext).Action("Index", "Home", new { area = "NewsArea" })
        };
    }

    public void RegisterBundles(BundleCollection bundles)
    {
        //todo: ...
    }

    public void RegisterRoutes(RouteCollection routes)
    {
        //todo: add custom routes.
    }

    public void RegisterServices(IContainer container)
    {
        // todo: add custom services.

        container.Configure(cfg =>
        {
            //cfg.For<INewsService>().Use<EfNewsService>();
        });
    }
}
}

```

در قسمت جاری فقط از متد `GetMenuItem` آن استفاده خواهیم کرد. در قسمت‌های بعد، تنظیمات `EF`، تنظیمات مسیریابی‌ها و `Bundling` و همچنین ثبت سرویس‌های افزونه را نیز بررسی خواهیم کرد. برای اینکه هر افزونه در منوی اصلی ظاهر شود، نیاز به یک نام، به همراه آدرسی به صفحه‌ی اصلی آن خواهد داشت. به همین جهت در متد `GetMenuItem` نحوه‌ی ساخت آدرسی را به اکشن متد `Index` کنترلر `Home` واقع در `Area` ای به نام `NewsArea`، مشاهده می‌کنید.

بارگذاری و تشخیص خودکار افزونه‌ها

پس از اینکه هر افزونه دارای کلاسی مشتق شده از قرارداد `IPlugin` شد، نیاز است آن‌ها را به صورت خودکار یافته و سپس پردازش کنیم. این کار را به کتابخانه‌ی `StructureMap` واگذار خواهیم کرد. برای این منظور پروژه‌ی جدیدی را به نام `MvcPluginMasterApp.IocConfig` آغاز کرده و سپس تنظیمات آن را به نحو ذیل تغییر دهید:

```

using System;
using System.IO;
using System.Threading;
using System.Web;
using MvcPluginMasterApp.PluginsBase;
using StructureMap;
using StructureMap.Graph;

namespace MvcPluginMasterApp.IocConfig
{
    public static class SmObjectFactory
    {
        {
            private static readonly Lazy<Container> _containerBuilder =
                new Lazy<Container>(defaultContainer, LazyThreadSafetyMode.ExecutionAndPublication);

            public static IContainer Container
            {
                get { return _containerBuilder.Value; }
            }

            private static Container defaultContainer()
        }
    }
}

```

```

    {
        return new Container(cfg =>
        {
            cfg.Scan(scanner =>
            {
                scanner.AssembliesFromPath(
                    path: Path.Combine(HttpRuntime.AppDomainAppPath, "bin"),
                    // یک اسمبلی نباید دوبار بارگذاری شود
                    assemblyFilter: assembly =>
                    {
                        return !assembly.FullName.Equals(typeof(IPlugin).Assembly.FullName);
                    });
                scanner.WithDefaultConventions(); //Connects 'IName' interface to 'Name' class
                scanner.AddAllTypesOf<IPlugin>().NameBy(item => item.FullName);
            });
        });
    }
}

```

این پروژه‌ی class library جدید برای کامپایل شدن نیاز به بسته‌های نیوگت ذیل دارد:

```

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

```

همچنین باید به صورت دستی، در قسمت ارجاعات پروژه، ارجاعی را به اسمبلی استاندارد System.Web نیز به آن اضافه نمایید.

کاری که در کلاس SmObjectFactory انجام شده، بسیار ساده است. مسیر پوشه‌ی Bin پروژه‌ی اصلی به structuremap معرفی شده‌است. سپس به آن گفته‌ایم که تنها اسمبلی‌هایی را که دارای اینترفیس IPlugin هستند، به صورت خودکار بارگذاری کن. در ادامه تمام نوع‌های IPlugin را نیز به صورت خودکار یافته و در مخزن تنظیمات خود، اضافه کن.

تامین نیازهای مسیریابی و Bundling هر افزونه به صورت خودکار

در ادامه به پروژه‌ی اصلی مراجعه کرده و در پوشه‌ی App_Start آن کلاس ذیل را اضافه کنید:

```

using System.Linq;
using System.Web.Optimization;
using System.Web.Routing;
using MvcPluginMasterApp;
using MvcPluginMasterApp.IocConfig;
using MvcPluginMasterApp.PluginsBase;

[assembly: WebActivatorEx.PostApplicationStartMethod(typeof(PluginsStart), "Start")]

namespace MvcPluginMasterApp
{
    public static class PluginsStart
    {
        public static void Start()
        {
            var plugins = SmObjectFactory.Container.GetAllInstances<IPlugin>().ToList();
            foreach (var plugin in plugins)
            {
                plugin.RegisterServices(SmObjectFactory.Container);
                plugin.RegisterRoutes(RouteTable.Routes);
                plugin.RegisterBundles(BundleTable.Bundles);
            }
        }
    }
}

```

بدیهی است در این حالت نیاز است ارجاعی را به پروژه‌ی MvcPluginMasterApp.PluginsBase به پروژه‌ی اصلی اضافه کنیم. در اینجا با استفاده از کتابخانه‌ای به نام WebActivatorEx (که باز هم توسط نویسندگان اصلی Razor Generator تهیه شده‌است)، یک متد PostApplicationStartMethod سفارشی را تعریف کرده‌ایم.

مزیت استفاده از اینکار این است که فایل Global.asax.cs برنامه شلوغ نخواهد شد. در غیر اینصورت باید تمام این کدها را در انتهای متد Application_Start قرار می‌دادیم. در اینجا با استفاده از structuremap، تمام افزونه‌های موجود به صورت خودکار بررسی شده و سپس پیشنیازهای مسیریابی و Bundling و همچنین تنظیمات IoC Container مورد نیاز آن‌ها به هر افزونه به صورت مستقل، تزریق خواهد شد.

اضافه کردن منوهای خودکار افزونه‌ها به پروژه‌ی اصلی

پس از اینکه کار پردازش اولیه‌ی IPlugin‌ها به پایان رسید، اکنون نوبت به نمایش آدرس اختصاصی هر افزونه در منوی اصلی سایت است. برای این منظور فایل جدیدی را به نام _PluginsMenu.cshtml، در پوشه‌ی shared پروژه‌ی اصلی اضافه کنید؛ با این محتوا:

```
@using MvcPluginMasterApp.IocConfig
@using MvcPluginMasterApp.PluginsBase
@{
    var plugins = SmObjectFactory.Container.GetAllInstances<IPlugin>().ToList();
}
@foreach (var plugin in plugins)
{
    var menuItem = plugin.GetMenuItem(this.Request.RequestContext);
    <li>
        <a href="@menuItem.Url">@menuItem.Name</a>
    </li>
}
```

در اینجا تمام افزونه‌ها به کمک structuremap یافت شده و سپس آیتم‌های منوی آن‌ها به صورت خودکار دریافت و اضافه می‌شوند.

سپس به فایل _Layout.cshtml پروژه‌ی اصلی مراجعه و توسط فراخوانی Html.RenderPartial، آن‌را در بین سایر آیتم‌های منوی اصلی اضافه می‌کنیم:

```
<div class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            @Html.ActionLink("MvcPlugin Master App", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })
        </div>
        <div class="navbar-collapse collapse">
            <ul class="nav navbar-nav">
                <li>@Html.ActionLink("Master App/Home", "Index", "Home", new { area = "" }, null)</li>
                @{ Html.RenderPartial("_PluginsMenu"); }
            </ul>
        </div>
    </div>
</div>
```

اکنون اگر پروژه را اجرا کنیم، یک چنین شکلی را خواهد داشت:



بنابراین به صورت خلاصه

- (1) هر افزونه، یک پروژه‌ی کامل ASP.NET MVC است که پوشه‌های ریشه‌ی اصلی آن حذف شده‌اند و اطلاعات آن توسط یک Area جدید تامین می‌شوند.
- (2) تنظیم فضای نام مسیریابی‌های تمام پروژه‌ها را فراموش نکنید. در غیر اینصورت شاهد تداخل پردازش کنترلرهای هم نام خواهید بود.
- (3) جهت سهولت کار، می‌توان فایل‌های bin هر افزونه را توسط رخداد post-build، به پوشه‌ی bin پروژه‌ی اصلی کپی کرد.
- (4) Viewهای هر افزونه توسط Razor Generator در فایل dll آن مدفون خواهند شد.
- (5) هر افزونه باید دارای کلاسی باشد که اینترفیس IPlugin را پیاده سازی می‌کند. از این اینترفیس برای ثبت اطلاعات هر افزونه یا دریافت اطلاعات سفارشی از آن کمک می‌گیریم.
- (6) با استفاده از استراکچر مپ و قرارداد IPlugin، منوهای هر افزونه را به صورت خودکار یافته و سپس به فایل layout اصلی اضافه می‌کنیم.

کدهای کامل این قسمت را از اینجا می‌توانید دریافت کنید:

[MvcPluginMasterApp-Part1.zip](#)

نظرات خوانندگان

نویسنده: محمد رعیت پیشه
تاریخ: ۱۶:۲۱ ۱۳۹۴/۰۱/۲۷

یک سوال، هنگام حذف افزونه با توجه به اینکه ممکنه کاربری در حال کار با بخش‌های مختلف اون باشه چه اتفاقی برای حذف ارجاع‌های اون به برنامه می‌افتد؟ آیا اجازه حذف لازم است؟

نویسنده: وحید نصیری
تاریخ: ۱۶:۲۵ ۱۳۹۴/۰۱/۲۷

- برنامه‌ی اصلی ارجاع مستقیمی را به هیچ افزونه‌ای ندارد.
+ هر نوع تغییری در پوشه‌ی bin برنامه [سبب ری استارت](#) آن خواهد شد. بنابراین اگر افزونه‌ای اضافه شود، برنامه به صورت خودکار ری استارت شده و بلافاصله افزونه‌ی جدید، قابل استفاده خواهد بود. اگر فایل افزونه‌ای از پوشه‌ی bin حذف شود، باز هم سبب ری استارت برنامه و بارگذاری خودکار منوها و محاسبه‌ی مجدد آن‌ها می‌گردد که اینبار دیگر شامل اطلاعات افزونه‌ی حذف شده نیست.

نویسنده: حامد 67
تاریخ: ۲۱:۴۱ ۱۳۹۴/۰۱/۲۷

سلام؛ یه سوال امنیتی، آیا راهکاری دارید که کسی به طور غیر مجاز برای برنامه پلاگین ننویسه منظور این هستش که فردی که پلاگین رو نوشته فقط با تایید بتونه فعالش کنه و از لحاظ امنیتی قابل چک باشه و بدون تایید اجرایی نشه چون من نگران هستم فردی پلاگین بنویسد و عمدا یا غیر عمد پلاگینی توسعه دهد که اطلاعات و روند فعالیت برنامه را جاسوسی کند خودم این ذهنیت رو دارم که هش کد هر پلاگین باید توسط مدیر تایید بشه و سپس قابل اجرا باشه تا کسی نتونه بعدا پلاگین را تغییر بده و امنیت سیستم را به خطر بنداره
در کل ملاحظات امنیتی پلاگین‌ها را چگونه در نظر بگیریم ؟

نویسنده: وحید نصیری
تاریخ: ۲۲:۰۵ ۱۳۹۴/۰۱/۲۷

از مطلب « [تهیه XML امضاء شده جهت تولید مجوز استفاده از برنامه](#) » ایده بگیرید. یک متد GetLicense به اینترفیس IPlugin اضافه کنید و در آن مجوز ارائه شده توسط افزونه را در برنامه‌ی اصلی بررسی کنید (در کلاس PluginsStart و همچنین فایل _PluginsMenu.cshtml). فقط کسانی می‌توانند «XML امضاء شده» تولید کنند که دسترسی به کلیدهای خصوصی و امن شما را داشته باشند.

نویسنده: میثم 99
تاریخ: ۱۵:۵۳ ۱۳۹۴/۰۱/۳۰

سلام؛ اگر بخواهیم مسیر یابی پروژه را به attribute routing تغییر بدهیم چه کارهایی باید انجام دهیم.

نویسنده: وحید نصیری
تاریخ: ۱۸:۲۶ ۱۳۹۴/۰۱/۳۰

از این مطالب تکمیلی استفاده کنید:
- « [قابلیت Attribute Routing در ASP.NET MVC 5](#) »
- « [Attribute Routing در ASP.NET MVC 5](#) »