عنوان: آموزش MEF#2(استفاده از MEF در Asp.Net MVC)
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۱/۱۱/۲۹ ۲۳:۳۰
آدرس: www.dotnettips.info
گروه‌ها: MVC, MEF, ASP.Net MVC 4

در پست قبلی با تکنولوژی MEF آشنا شدید.در این پست قصد دارم روش استفاده از MEF رو در Asp.Net MVC نمایش بدم. برای شروع یک پروژه پروژه MVC ایجاد کنید.
در قسمت Model کلاس Book رو ایجاد کنید و کدهای زیر رو در اون قرار بدید.

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

        public string Title { get; set; }

        public string ISBN { get; set; }
    }
```

یک فولدر به نام Repositories ایجاد کنید و یک اینترفیس به نام IBookRepository رو به صورت زیر ایجاد کنید.

```
public interface IBookRepository
    {
        IList<Book> GetBooks();
    }
```

حالا نوبت به کلاس BookRepository می‌رسه که باید به صورت زیر ایجاد بشه.

```
[Export( typeof( IBookRepository ) )]
    public class BookRepository
    {
        public IList<Book> GetBooks()
        {
            List<Book> listOfBooks = new List<Book>( 3 );
            listOfBooks.AddRange( new Book[]
            {
                new Book(){Id=1 , Title="Book1"},
                new Book(){Id=2 , Title="Book2"},
                new Book(){Id=3 , Title="Book3"},
            } );
            return listOfBooks;
        }
    }
```

بر روی پوشه کنترلر کلیک راست کرده و یک کنترلر به نام BookController ایجاد کنید و کدهای زیر رو در اون کپی کنید.

```
[Export]
    [PartCreationPolicy( CreationPolicy.NonShared )]
    public class BookController : Controller
    {
        [Import( typeof( IBookRepository ) )]
        BookRepository bookRepository;

        public BookController()
        {
        }

        public ActionResult Index()
        {
            return View( this.bookRepository.GetBooks() );
        }
    }
```

PartCreationPolicyکه شامل 3 نوع می‌باشد.
Shared: بعنی در نهایت فقط یک نمونه از این کلاس در هز Container وجود دارد.
NonShared : یعنی به ازای هر درخواستی که از نمونه‌ی Export شده می‌شود یک نمونه جدید ساخته می‌شود.
Any : هر 2 حالت فوق Support می‌شود.

حالا قصد داریم یک ControllerFactory با استفاده از MEF ایجاد کنیم.(Controller Factory برای ایجاد نمونه ای از کلاس Controller مورد نظر استفاده می‌شود) برای بیشتر پروژه‌ها استفاده از DefaultControllerFactory کاملا مناسبه.

```
public class MEFControllerFactory : DefaultControllerFactory
    {
        private readonly CompositionContainer _compositionContainer;

        public MEFControllerFactory( CompositionContainer compositionContainer )
        {
            _compositionContainer = compositionContainer;
        }

        protected override IController GetControllerInstance( RequestContext requestContext, Type
controllerType )
        {
            var export = _compositionContainer.GetExports( controllerType, null, null
).SingleOrDefault();

            IController result;

            if ( export != null )
            {
                result = export.Value as IController;
            }
            else
            {
                result = base.GetControllerInstance( requestContext, controllerType );
                _compositionContainer.ComposeParts( result );
            }

}
    }
```

اگر با مفاهیمی نظیر CompositionContainer آشنایی ندارید می‌تونید پست قبلی رو مطالعه کنید.
حالا قصد داریم یک DependencyResolver رو با استفاده از MEF به صورت زیر ایجاد کنیم.(DependencyResolver برای ایجاد نمونه ای از کلاس مورد نظر برای کلاس هایی است که به یکدیگر نیاز دارند و برای ارتباط بین آن از Depedency Injection استفاده شده است.

```
public class MefDependencyResolver : IDependencyResolver
    {
        private readonly CompositionContainer _container;

        public MefDependencyResolver( CompositionContainer container )
        {
            _container = container;
        }

        public IDependencyScope BeginScope()
        {
            return this;
        }

        public object GetService( Type serviceType )
        {
            var export = _container.GetExports( serviceType, null, null ).SingleOrDefault();

            return null != export ? export.Value : null;
        }

        public IEnumerable<object> GetServices( Type serviceType )
        {
            var exports = _container.GetExports( serviceType, null, null );
            var createdObjects = new List<object>();

            if ( exports.Any() )
```

```
            {
                foreach ( var export in exports )
                {
                    createdObjects.Add( export.Value );
                }
            }

            return createdObjects;
        }

        public void Dispose()
        {

        }
    }
```

حال یک کلاس Plugin ایجاد می‌کنیم.

```
public class Plugin
    {
        public void Setup()
        {
            var container = new CompositionContainer( new DirectoryCatalog( HostingEnvironment.MapPath(
"~/bin" ) ) );

            CompositionBatch batch = new CompositionBatch();

            batch.AddPart( this );

            ControllerBuilder.Current.SetControllerFactory( new MEFControllerFactory( container ) );

            System.Web.Http.GlobalConfiguration.Configuration.DependencyResolver = new
MefDependencyResolver( container );

            container.Compose( batch );
        }
    }
```

همانطور که در این کلاس می‌بینید ابتدا یک CompositionContainer ایجاد کردیم که یک ComposablePartCatalog از نوع DirectoryCatalog به اون پاس دادم.
DirectoryCatalog یک مسیر رو دریافت کرده و Assemblyهای موجود در مسیر مورد نظر رو به عنوان Catalog در Container اضافه میکنه. می‌تونستید از یک AssemblyCatalog هم به صورت زیر استفاده کنید.

```
var container = new CompositionContainer( new AssemblyCatalog( Assembly.GetExecutingAssembly() ) );
```

در تکه کد زیر ControllerFactory پروژه رو از نوع MEFControllerFactory قرار دادیم.

```
ControllerBuilder.Current.SetControllerFactory( new MEFControllerFactory( container ) );
```

و در تکه کد زیر هم DependencyResolver پروژه از نوع MefDependencyResolver قرار دادیم.

```
System.Web.Http.GlobalConfiguration.Configuration.DependencyResolver = new MefDependencyResolver(
container );
```

کافیست در فایل Global نیز تغییرات زیر را اعمال کنیم.

```
protected void Application_Start()
        {
            Plugin myPlugin = new Plugin();
            myPlugin.Setup();

            AreaRegistration.RegisterAllAreas();
```

```
            RegisterRoutes( RouteTable.Routes );
        }

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

            routes.MapRoute(
                "Default", // Route name
                "{controller}/{action}/{id}", // URL with parameters
                new { controller = "Book", action = "Index", id = UrlParameter.Optional } // Parameter
defaults
            );
        }
```

در انتها View متناظر با BookController رو با سلیقه خودتون ایجاد کنید و بعد پروژه رو اجرا و نتیجه رو مشاهده کنید.

نظرات خوانندگان

نویسنده: حسینی
تاریخ: ۱۳۹۱/۱۲/۱۷ ۱:۹

سلام
تشکر از مطلب خوبتون
اگر بخوایم الگوی Unitof Work با MEF پیاده سازی کنیم دراون صورت به چه صورته؟

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۱/۱۲/۱۹ ۸:۲۶

با توجه به این که پیاده سازی الگوی UnitOfWork در ORMهای مختلف متفاوت است یک مثال کلی در این زمینه پیاه سازی می‌کنم.
فرض کنیم بک اینترفیس به صورت زیر داریم:

```
public interface IUnitOfWork
    {
        ISession CurrentSession { get; }
        void BeginTransaction();
        void Commit();
        void RollBack();
    }
```

می‌تونید به جای استفاده از ISession از DbSet در EF CodeFirst هم استفاده کنید.
حالا نیاز به کلاس UnitOfWork برای پیاده سازی Interface بالا داریم. به صورت زیر:

```
[Export(typeof(IUnitOfWork)]
    public class UnitOfWork : IUnitOfWork
    {
        public ISession CurrentSession
        {
            get { throw new NotImplementedException(); }
        }

        public void BeginTransaction()
        {
            throw new NotImplementedException();
        }

        public void Commit()
        {
            throw new NotImplementedException();
        }

        public void RollBack()
        {
            throw new NotImplementedException();
        }
    }
```

پیاده سازی متدها رو به عهده خودتون. فقط از Export Attribute برای تعیین نوع وابستگی کلاس UnitOfWork استفاده کردم.
و در آخر کلاس Repository مربوطه هم به شکل زیر است.

```
public class Respository
    {

       [Import]

       private IUnitOfWork uow;

        public Respository()
        {
        }
    }
```

در کلاس Repository فیلد uow به دلیل داشتن Import Attribute همیشه توسط Composition Container

مقدار دهی می‌شه.

نویسنده: سینا نادی حق
تاریخ: ۱۳۹۲/۰۹/۲۰ ۱۵:۳

سلام؛ من MEF رو تو MVC اعمال کردم الان به یک مشکلی بر خوردم. ممنون میشم اگه کمکم کنید.
من Pluginها رو توی پروژه جدا با Class Library می‌نویسم، پروژه MVC هم razor engine هست. ولی وقتی می‌خوام viewها رو به Modelها نسبت بدم نمیشه(Model توی همون پروژه Class Library پلاگین هست)
کد view:

```
@model Plugin1.Models.Post
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

@Model.Title
```

ارورشم اینه:

The type or namespace name 'Plugin1' could not be found (are you missing a using directive or an assembly reference?)

در کل چطوری می‌تونم Modelها رو به Viewها در این حالت نسبت بدم.
ممنون

نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۹/۲۰ ۱۷:۱۳

در ابتدا باید عنوان کنم که از آن جا مدل باید بین پلاگین‌ها و سایر ماژول‌ها به اشتراک گذاشته شود در نتیجه مدل برنامه **نباید** در Class Library پلاگین‌ها باشد. از طرفی شما نیازی به Export کردن مدل‌های برنامه توسط MEF ندارید. دلیل خطای برنامه شما این است که Class Library پلاگین شما به پروژه MVC رفرنس داده نشده است(که البته درست هم عمل نموده اید، فقط مدل برنامه را در یک پروژه دیگر برای مثال DomainModel قرار داده و سپس پروژه DomainModel را به صورت مستقیم به پروژه MVC رفرنس دهید).
فقط به این نکته نیز توجه داشته باشید که بعد از تعریف پلاگین‌ها در class library دیگر حتما اسمبلی‌های مربوطه را از طریق کاتالوگ‌های موجود (نظیر AssembleyCatalog و DirectoryCatalog و...) به CompositionContainer پروژه MVC خود اضافه کنید.

نویسنده: سینوس
تاریخ: ۱۳۹۲/۱۱/۰۵ ۲۳:۴۰

با سلام و خسته نباشید. من پروژم رو با MEF راه اندازی کردم مشکل خاصی هم ندارم

با توجه به فرمایش شما باید Modelها توی یک پروژه جدا و عمومی استفاده بشه. مشکلی که من دارم اینه که وقتی پروژه (core) آماده شد و آپلود کردم دیگه نمی‌خوام وقتی کاربری از یک ماژول استفاده می‌کند خود پروژه یا پروژه مدل‌ها update بشه. دلیلشم این هست که هر کاربری ماژول‌های جداگانه دارد و ماژول‌ها رو runtime اضافه، حذف و یا بروز می‌کند.
ممنون از لطفتون