عنوان: **آموزش MEF#1**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۱/۱۱/۲۴ ۲۳:۱۵
آدرس: www.dotnettips.info
برچسب‌ها: MEF, Dependency Injection

Managed Extensibility Framework یا **MEF** کامپوننتی از Framework 4 است که برای ایجاد برنامه‌های توسعه پذیر (Extensible) با حجم کم کد استفاده میشه.این تکنولوژی به برنامه نویسان این امکان رو میده که توسعه‌های (Extension) برنامه رو بدون پیکربندی استفاده کنند. همچنین به توسعه دهندگان این اجازه رو می‌ده که به آسانی کدها رو کپسوله کنند .
MEF به عنوان بخشی از .NET 4 و Silverlight 4 معرفی شد. MEF یک راه حل ساده برای مشکل توسعه در حال اجرای برنامه‌ها ارائه می‌کند.تا قبل از این تکنولوژی ، هر برنامه‌ای که می‌خواست یک مدل Plugin را پشتیبانی کنه لازم بود که خودش زیر ساخت‌ها را از ابتدا ایجاد کنه . این Pluginها اغلب برای برنامه‌های خاصی بودند و نمی‌توانستند در پیاده سازی‌های چندگانه دوباره استفاده شوند. ولی MEF در راستای حل این مشکلات ، روش استانداردی رو برای میزبانی برنامه‌های کاربردی پیاده کرده است.
برای فهم بهتر مفاهیم یک مثال ساده رو با MEF پیاده سازی می‌کنم.
ابتدا یک پروژه از نوع Console Application ایجاد کنید . بعد با استفاده از Add Reference یک ارجاع به System.ComponentModel.Composition بدید. سپس یک Interface به نام IViewModel را به صورت زیر ایجاد کنید:

```
public interface IViewModel
    {
        string Name { get; set; }
    }
```

یک خاصیت به نام Name برای دسترسی به نام ViewModel ایجاد می‌کنیم.
سپس 2 تا ViewModel دیگه ایجاد می‌کنیم که IViewModel را پیاده سازی کنند. به صورت زیر:
:ViewModelFirst

```
[Export( typeof( IViewModel ) )]
    public class ViewModelFirst : IViewModel
    {
        public ViewModelFirst()
        {
            this.Name = "ViewModelFirst";
        }

        public string Name
        {
            get
            {
                return _name;
            }
            set
            {
                _name = value;
            }
        }
        private string _name;
    }
```

:ViewModelSecond

```
[Export( typeof( IViewModel ) )]
    public class ViewModelSecond : IViewModel
    {
        public ViewModelSecond()
        {
```

```
            this.Name = "ViewModelSecond";
        }

        public string Name
        {
            get
            {
                return _name;
            }
            set
            {
                _name = value;
            }
        }
        private string _name;
    }
```

Export Attribute استفاده شده در بالای کلاس‌های ViewModel به این معنی است که این کلاس‌ها اینترفیس IViewModel رو Export کردند تا در جای مناسب بتونیم این ViewModel ها Import کنیم.(Import , Export از مفاهیم اصلی در MEF هستند)
حالا نوبت به پیاده سازی کلاس Plugin می‌رسه.

```
public class PluginManager
    {
        public PluginManager()
        {

        }

        public IList<IViewModel> ViewModels
        {
            get
            {
                return _viewModels;
            }
            private set
            {
                _viewModels = value;
            }
        }

        [ImportMany( typeof( IViewModel ) )]
        private IList<IViewModel> _viewModels = new List<IViewModel>();

        public void SetupManager()
        {
            AggregateCatalog aggregateCatalog = new AggregateCatalog();

            CompositionContainer container = new CompositionContainer( aggregateCatalog );

            CompositionBatch batch = new CompositionBatch();

            batch.AddPart( this );

            aggregateCatalog.Catalogs.Add( new AssemblyCatalog( Assembly.GetExecutingAssembly() ) );

            container.Compose( batch );
        }
```

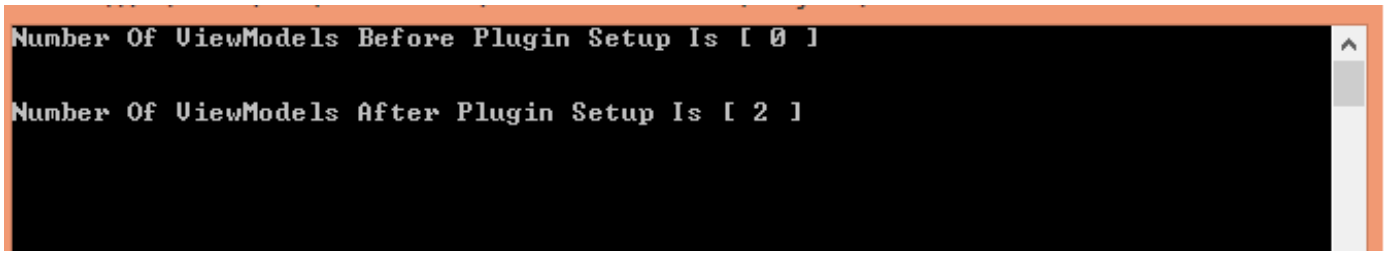
کلاس PluginManager برای شناسایی و استفاده از کلاس هایی که صفت‌های Export رو دارند نوشته شده(دقیقا شبیه یک UnityContainer در Microsoft Unity Application Block یا IKernel در Ninject) عمل می‌کنه با این تفاوت که نیازی به Register با Bind کردن ندارند)
ابتدا یک لیست از کلاس هایی که IViewModel رو Export کردند داریم.
بعد در متد SetupManager ابتدا یک AggregateCatalog نیاز داریم تا بتونیم Composition Partها رو بهش اضافه کنیم. به کد زیر توجه کنید:

```
aggregateCatalog.Catalogs.Add( new AssemblyCatalog( Assembly.GetExecutingAssembly() ) );
```

تو این قطعه کد من یک Assembly Catalog رو که به Assembly جاری برنامه اشاره می‌کنه به AggregateCatalog اضافه کردم.
متد batch.AddPart(this) در واقع به این معنی است که به MEF گفته می‌شود این کلاس ممکن است شامل Export هایی باشد که به یک یا چند Import وابستگی دارند.
متد AddExport(this) در CompositionBatch به این معنی است که این کلاس ممکن است شامل Exportهایی باشد که به Import وابستگی ندارند.
حالا برای مشاهده نتایج کد زیر را در کلاس Program اضافه می‌کنیم:

```
static void Main( string[] args )
        {
            PluginManager plugin = new PluginManager();

            Console.WriteLine( string.Format( "Number Of ViewModels Before Plugin Setup Is [ {0} ]",
plugin.ViewModels.Count ) );

            Console.WriteLine( Environment.NewLine );

            plugin.SetupManager();

            Console.WriteLine( string.Format( "Number Of ViewModels After Plugin Setup Is [ {0} ]",
plugin.ViewModels.Count ) );

            Console.ReadLine();
        }
```

در کلاس بالا ابتدا تعداد کلاس‌های موجود در لیست ViewModels رو قبل از Setup کردن Plugin نمایش داده سپس بعد از Setup کردن Plugin دوباره تعداد کلاس‌های موجود در لیست ViewModel رو مشاهده می‌کنیم.که خروجی به شکل زیر تولید خواهد شد.



متد SetupManager در کلاس Plugin (با توجه به AggregateCatalog) که در این برنامه فقط Assembly جاری رو بهش اضافه کردیم تمام کلاس هایی رو که نوع IViewModel رو Export کردند پیدا کرده و در لیست اضافه می‌کنه(این کار رو با توجه به ImportMany Attribute) انجام میده. در پست‌های بعدی روش استفاده از MEF رو در Prism یا WAF توضیح می‌دم.

نظرات خوانندگان

نویسنده: MehRad
تاریخ: ۱۳۹۱/۱۱/۲۵ ۱۱:۴۷

با تشکر از مطلب خوبتون
اگر امکان داره استفاده از MEF رو در ASP.NET MVC هم توضیح بدید

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۱/۱۱/۲۵ ۱۳:۲۳

ممنون.
بله حتما در پست‌های بعدی در مورد MEF و استفاده اون در (WAF(**W** PF **A** pplication **F** ramework و MVC و Prism توضیحاتی رو خواهد داد.

نویسنده: علیرضا پایدار
تاریخ: ۱۳۹۲/۰۶/۲۶ ۱۳:۸

ممنون مفید بود.
توی Ninject میتونستیم مشخص کنیم یک پلاگین وابسته به پلاگین دیگه باشه. این کار در MEF به چه شکلی انجام میگیرد؟

نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۶/۲۶ ۱۳:۴۷

MEF برای پیاده سازی مبحث Chaining Dependencies از مفهوم Contract در Export Attribute استفاده می‌کند. پارامتر اول در Export برای ContractName است. به صورت زیر:

```
[Export( "ModuleA" , typeof( IMyInterface) )]
public class ClassA : IMyInterface
{
}

[Export( "ModuleB" , typeof( IMyInterface))]
public class ClassB : IMyInterface
{
}
```

در نتیجه در هنگام Import کردن کلاس‌های بالا باید حتما ContractName آن‌ها را نیز مشخص کنیم:

```
public class ModuleA
    {
        [ImportingConstructor]
        public ModuleA([ImportMany( "ModuleA" , IMyInterface)] IEnumerable<IMyInterface> controllers  )
        {

        }
```

با استفاده از ImportMany Attribute و ContractName به راحتی می‌توانیم تمام آبجکت‌ها Export شده در هر ماژول را تفکیک کرد.

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

عنوان: **پیاده سازی UnitOfWork به وسیله MEF**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۱/۱۲/۲۴ ۲۲:۵
آدرس: www.dotnettips.info
برچسب‌ها: Entity framework, MEF, UnitOfWork

در این پست قصد دارم یک UnitOfWork به روش MEF پیاده سازی کنم. ORM مورد نظر EntityFramework CodeFirst است. در صورتی که با MEF , UnitOfWork آشنایی ندارید از لینک‌های زیر استفاده کنید:

MEF

UnitOfWork

برای شروع ابتدا مدل برنامه رو به صورت زیر تعریف کنید.

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

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

سپس فایل Map رو برای مدل بالا به صورت زیر تعریف کنید.

```
public class CategoryMap : EntityTypeConfiguration<Entity.Category>
    {
        public CategoryMap()
        {
            ToTable( "Category" );

            HasKey( _field => _field.Id );

            Property( _field => _field.Title )
            .IsRequired();
        }
    }
```

برای پیاده سازی الگوی واحد کار ابتدا باید یک اینترفیس به صورت زیر تعریف کنید.

```
using System.Data.Entity;
using System.Data.Entity.Infrastructure;

namespace DataAccess
{
    public interface IUnitOfWork
    {
        DbSet<TEntity> Set<TEntity>() where TEntity : class;
        DbEntityEntry<TEntity> Entry<TEntity>() where TEntity : class;
        void SaveChanges();
        void Dispose();
    }
}
```

DbContext مورد نظر باید اینترفیس مورد نظر را پیاده سازی کند و برای اینکه بتونیم اونو در CompositionContainer اضافه کنیم باید از Export Attribute استفاده کنیم.

چون کلاس DatabaseContext از اینترفیس IUnitOfWork ارث برده است برای همین از InheritedExport استفاده می‌کنیم.

```
[InheritedExport( typeof( IUnitOfWork ) )]
    public class DatabaseContext : DbContext, IUnitOfWork
    {
        private DbTransaction transaction = null;

        public DatabaseContext()
        {
            this.Configuration.AutoDetectChangesEnabled = false;
            this.Configuration.LazyLoadingEnabled = true;
        }
```

```
        protected override void OnModelCreating( DbModelBuilder modelBuilder )
        {
            modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();

            modelBuilder.AddFormAssembly( Assembly.GetAssembly( typeof( Entity.Map.CategoryMap ) ) );
        }

        public DbEntityEntry<TEntity> Entry<TEntity>() where TEntity : class
        {
            return this.Entry<TEntity>();
        }
    }
```

نکته قابل ذکر در قسمت OnModelCreating این است که یک Extension Methodبه نام AddFromAssembly (همانند NHibernate) اضافه شده است که از Assembly مورد نظر تمام کلاس‌های Map رو پیدا می‌کنه و اونو به ModelBuilder اضافه می‌کنه. کد متد به صورت زیر است:

```
public static class ModelBuilderExtension
    {
        public static void AddFormAssembly( this DbModelBuilder modelBuilder, Assembly assembly )
        {
            Array.ForEach<Type>( assembly.GetTypes().Where( type => type.BaseType != null &&
type.BaseType.IsGenericType && type.BaseType.GetGenericTypeDefinition() == typeof(
EntityTypeConfiguration<> ) ).ToArray(), delegate( Type type )
            {
                dynamic instance = Activator.CreateInstance( type );
                modelBuilder.Configurations.Add( instance );
            } );
        }
    }
```

برای پیاده سازی قسمت BusinessLogic ابتدا کلاس BusiessBase را در آن قرار دهید:

```
public class BusinessBase<TEntity> where TEntity : class
    {
        public BusinessBase( IUnitOfWork unitOfWork )
        {
            this.UnitOfWork = unitOfWork;
        }

        [Import]
        public IUnitOfWork UnitOfWork
        {
            get;
            private set;
        }

        public virtual IEnumerable<TEntity> GetAll()
        {
            return UnitOfWork.Set<TEntity>().AsNoTracking();
        }

        public virtual void Add( TEntity entity )
        {
            try
            {
                UnitOfWork.Set<TEntity>().Add( entity );
                UnitOfWork.SaveChanges();
            }
            catch
            {
                throw;
            }
            finally
            {
                UnitOfWork.Dispose();
            }
        }
    }
```

تمام متدهای پایه مورد نظر را باید در این کلاس قرار داد که برای مثال من متد Add , GetAll را براتون پیاده سازی کردم.
UnitOfWork توسط ImportAttribute مقدار دهی می‌شود و نیاز به وهله سازی از آن نیست
کلاس Category رو هم باید به صورت زیر اضافه کنید.

```
public class Category : BusinessBase<Entity.Category>
    {
        [ImportingConstructor]
        public Category( [Import( typeof( IUnitOfWork ) )] IUnitOfWork unitOfWork )
            : base( unitOfWork )
        {
        }
    }
```

.در انتها باید UI مورد نظر طراحی شود که من در اینجا از Console Application استفاده کردم. یک کلاس به نام Plugin ایجاد کنید و کدهای زیر را در آن قرار دهید.

```
public class Plugin
    {
        public void Run()
        {
            AggregateCatalog catalog = new AggregateCatalog();

            Container = new CompositionContainer( catalog );

            CompositionBatch batch = new CompositionBatch();

            catalog.Catalogs.Add( new AssemblyCatalog( Assembly.GetExecutingAssembly() ) );

            batch.AddPart( this );

            Container.Compose( batch );
        }

        public CompositionContainer Container
        {
            get;
            private set;
        }
    }
```

در کلاس Plugin توسط AssemblyCatalog تمام Export Attributeهای موجود جستجو می‌شود و بعد به عنوان کاتالوگ مورد نظر به Container اضافه می‌شود. انواع Catalog در MEF به شرح زیر است:
AssemblyCatalog : در اسمبلی مورد نظر به دنبال تمام Export Attributeها می‌گردد و آن‌ها را به عنوان ExportedValue در Container اضافه می‌کند.
TypeCatalog: فقط یک نوع مشخص را به عنوان ExportAttribute در نظر می‌گیرد.
DirectoryCatalog : در یک مسیر مشخص تمام Assembly مورد نظر را از نظر Export Attribute جستجو می‌کند و آن‌ها را به عنوان ExportedValue در Container اضافه می‌کند.
ApplicationCatalog : در اسمبلی و فایل‌های (EXE) مورد نظر به دنبال تمام Export Attributeها می‌گردد و آن‌ها را به عنوان ExportedValue در Container اضافه می‌کند.
AggregateCatalog : تمام موارد فوق را Support می‌کند.

کلاس Program رو به صورت زیر بازنویسی کنید.

```
class Program
    {
        static void Main( string[] args )
        {
            Plugin plugin = new Plugin();
            plugin.Run();

            Category category = new Category(plugin.Container.GetExportedValue<IUnitOfWork>());
            category.GetAll().ToList().ForEach( _record => Console.Write( _record.Title ) );
```

```
        }
    }
```

پروژه اجرا کرده و نتیجه رو مشاهده کنید.

نظرات خوانندگان

نویسنده: شایان مرادی
تاریخ: ۱۳۹۱/۱۲/۲۵ ۲:۵

سلام
سپاس از مطلبتون
در قسمت زیر شما نام کلاس CategoriMap رو ذکر کردید
در این صورت به ازای هر کلاس باید در این قسمت نام Map اون ذکر شود؟
خوب اگر قرار باشه به ازای هر کلاس نام Map آن ذکر شود دیگر نیازی به AddFormAssembly نبود مستقیماً نام .CategoryMap در modelBuilder اضافه میکردیم

```
protected override void OnModelCreating( DbModelBuilder modelBuilder )
        {
            modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();

            modelBuilder.AddFormAssembly( Assembly.GetAssembly( typeof( Entity.Map.CategoryMap ) ) );
        }
```

نویسنده: محسن
تاریخ: ۱۳۹۱/۱۲/۲۵ ۸:۲۵

اگر به پیاده سازی AddFromAsm دقت کنید یک ForEach داره. یعنی فقط شما یک اسمبلی رو بهش می‌دی، خودش مابقی رو پیدا می‌کنه . حتی اضافه کردن DbSetها هم قابلیت خودکار سازی داره .

نویسنده: محسن.د
تاریخ: ۱۳۹۱/۱۲/۲۵ ۱۱:۱۱

یک نکته که در مورد پیاده سازی بالا وجود داره اینه که متد save رو در خود توابع مربوط به repositoryها قرار داده اید و این با الگوی unitOfWork همخوانی نداره .

نویسنده: شایان مرادی
تاریخ: ۱۳۹۱/۱۲/۲۵ ۱۱:۲۵

بله در جریانم
اما در اینجا به صورت مستقیم نام MaP مستقیم ذکر شده
برای این این سوال برام پیش امد . چون اگر قرار بود خود کار ذکر بشن پس دیگه به ذکر Entity.Map.CategoryMap نبود

نویسنده: محسن
تاریخ: ۱۳۹۱/۱۲/۲۵ ۱۲:۳

مهم نیست. همینقدر که ایده اینکار مطرح شده مابقی‌اش هنر Reflection مصرف کننده است. مثلا از یک رشته (ذخیره شده در تنظیمات برنامه) هم می‌شود این نام‌ها را دریافت کرد: Assembly.Load

نویسنده: شایان مرادی
تاریخ: ۱۳۹۱/۱۲/۲۵ ۲۳:۴۶

متد Savechange در interface IUnitOfWork قرار دارد .اما در DatabaseContext پیاده سازی نشده که اون هم احتمالا یادشون رفته باشه.

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۱/۱۲/۲۵ ۲۳:۵۳

در مورد سوال اول که چرا CategoryMap رو به متد AddFromAssembly پاس دادم.
متد AddFromAssembly نیاز به یک Assembly دارد تا بتونه تمام کلاس هایی رو که از کلاس EntityTypeConfiguration ارث برده اند رو پیدا کنه و اونها رو به صورت خودکار به ModelBuilder اضافه کنه. به همین دلیل من Assembly کلاس CategoryMap رو به اون پاس دادم. دقت کنید که اگر من n تا کلاس Map دیگه هم توی این ClassLibrary داشتم باز توسط همین دستور این کار به صورت خودکار انجام می‌شد. (پیشنهاد می‌کنم تست کنید)
نکته: این متد برگرفته شده از متد AddFromAssembly در NHibernate Session Configuration است.
البته بهتر است که یک کلاس پایه برای این کار بسازید و اون کلاس رو به AddFromAssembly پاس بدید.

در مورد سوال دوست عزیز مبنی بر اینکه متد Save رو در خود توابع Repository قرار دادم.
اگر به کدهای نوشته شده دقت کنید من اصلا مفهومی به نام Repository رو پیاده سازی نکردم. به این دلیل که خود DbContext ترکیبی از UnitOfWork , Repository Pattern است. متد SaveChange صدا زده شده همان متد SaveChange در DbContext است.
فرض کنید من یک کلاس Business دیگر به صورت زیر داشتم.

```
public class MyBusiness : BusinessBase<Entity.MyEntity>
    {
        [ImportingConstructor]
        public MyBusiness  ( [Import( typeof( IUnitOfWork ) )] IUnitOfWork unitOfWork )
            : base( unitOfWork )
        {
        }
        public void Add(Entity.MyEntity entity)
        {
            UnitOfWork.Set<MyEntity>().Add(entity);
        }
    }
```

حالا کد کلاس Business Category به صورت زیر تغییر می‌کنه.

```
public class Category : BusinessBase<Entity.Category>
    {
        [ImportingConstructor]
        public Category( [Import( typeof( IUnitOfWork ) )] IUnitOfWork unitOfWork )
            : base( unitOfWork )
        {
        }

        public override void Add( Entity.Category entity )
        {
            new MyBusiness().Add( new Entity.MyEntity() );
            UnitOfWork.Set<Entity.Category>().Add( entity );
            UnitOfWork.SaveChanges();
        }
    }
```

همان طور که می‌بینید فقط یک بار متد SaveChange فراخوانی شده است. Virtual کردن متد BusinessBase دقیقا به همین دلیل است.

نویسنده: علي
تاریخ: ۱۳۹۱/۱۲/۲۶ ۰:۲۶

کلاس پایه DbContext پیاده سازی SaveChanges رو داره.

نویسنده: MehRad
تاریخ: ۱۳۹۱/۱۲/۲۷ ۳:۴۹

تشکر میکنم بابت مقاله خوب و کاملتون.

نویسنده: Masoud
تاریخ: ۱۳۹۱/۱۲/۲۷ ۱۸:۴۶

اگه به جای category که شما تعریف کردین . موجودیتهای مثل خبرها یا آخرین نظرات و ... داشتیم چطور میتونیم اینا رو گروه بندی کنیم جوری که بشه هر کدوم رو در صفحه اصلی نشون داد؟مثلا همه موجودیت‌ها رو برسی کنه و بر اساس گروهشون اونایی که گروه خاصی دارن در قسمت منوی صفحه اصلی نمایش بده.ایا این امکان در MEF هست؟

نویسنده: ج.زوسر
تاریخ: ۱۳۹۲/۰۱/۱۷ ۲۱:۵۹

مرسی از مقاله خیلی خوبتون .

چه جور می‌تونم این روش روی local تست کنم .که اثرش رو مشاهده کنم .

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۷/۱۴ ۱۲:۱۵

مهندس سلام
من از MEF برای تزریق کامپوننت هام به یک الگوی کار در MVC استفاده کردم، کار به درستی انجام میشه اما Attributeهای کلاس‌ها مثل پیام‌های خطا، طول فیلد و ... روی خروجی نهایی اعمال نمیشه و دیتابیس طبق پیش فرض‌ها ساخته میشه.
البته اگر از یک فایل کانفیگ (fluent API) جدا استفاده کنم مشکل حل میشه اما در این فایلها نمی‌توان پیام خطا برای حالت‌های مختلف تعریف کرد (البته به نظرم)

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۷/۱۴ ۱۷:۲۵

در کدهای این مطلب نکات خودکار کردن تعاریف DbSetها در EF Code first ذکر نشدند. فقط نکات افزودن خودکار کلاس‌های تنظیمات نگاشت‌ها در EF Code first پیاده سازی شدند.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۷/۱۴ ۲۱:۱۷

اما این پست‌ها ربطی به سوال من نداره قبلا همش بررسی کردم مهندس. مشکل توی عدم تزریق Metadataهای کلاس مانند DisplayName، ErrorMessageها و ... است که در FluentApi ظاهرا قابل پیاده سازی نیست

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۷/۱۴ ۲۲:۴۴

من الان یک ویژگی StringLength به طول 30 رو روی خاصیت Title کلاس Category مقاله جاری اضافه کردم. با همین کدهای فوق، این فیلد با طول 30 الان در دیتابیس قابل مشاهده است. آغاز دیتابیس اصلا کاری به MEF نداره.

این هم فایل مثالی که در آن ویژگی طول رشته اعمال شده برای آزمایش:

MefSample.cs

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۷/۱۵ ۹:۹

من کلاسهام به این شکله:
کلاس کانتکس‌های من

```
public class VegaContext : DbContext, IUnitOfWork, IDbContext
    {
#region Constructors (2)

        /// <summary>
        /// Initializes the <see cref="VegaContext" /> class.
        /// </summary>
        static VegaContext()
        {
            Database.SetInitializer<VegaContext>(null);
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="VegaContext" /> class.
        /// </summary>
        public VegaContext() : base("LocalSqlServer") { }

#endregion Constructors

#region Properties (2)

        /// <summary>
        /// Gets or sets the languages.
        /// </summary>
        /// <value>
        /// The languages.
        /// </value>
        public DbSet<Language> Languages { get; set; }

        /// <summary>
        /// Gets or sets the resources.
        /// </summary>
        /// <value>
        /// The resources.
        /// </value>
        public DbSet<Resource> Resources { get; set; }

#endregion Properties

#region Methods (2)

// Public Methods (1)

        /// <summary>
        /// Setups the specified model builder.
        /// </summary>
        /// <param name="modelBuilder">The model builder.</param>
        public void Setup(DbModelBuilder modelBuilder)
        {
            //todo
            modelBuilder.Configurations.Add(new ResourceMap());
            modelBuilder.Configurations.Add(new LanguageMap());
            modelBuilder.Entity<Resource>().ToTable("Vega_Languages_Resources");
            modelBuilder.Entity<Language>().ToTable("Vega_Languages_Languages");
            //base.OnModelCreating(modelBuilder);
        }
// Protected Methods (1)

        /// <summary>
        /// This method is called when the model for a derived context has been initialized, but
        /// before the model has been locked down and used to initialize the context.  The default
        /// implementation of this method does nothing, but it can be overridden in a derived class
        /// such that the model can be further configured before it is locked down.
        /// </summary>
        /// <param name="modelBuilder">The builder that defines the model for the context being
created.</param>
        /// <remarks>
        /// Typically, this method is called only once when the first instance of a derived context
        /// is created.  The model for that context is then cached and is for all further instances of
        /// the context in the app domain.  This caching can be disabled by setting the ModelCaching
        /// property on the given ModelBuidler, but note that this can seriously degrade performance.
        /// More control over caching is provided through use of the DbModelBuilder and
DbContextFactory
```

```
        /// classes directly.
        /// </remarks>
        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Configurations.Add(new ResourceMap());
            modelBuilder.Configurations.Add(new LanguageMap());
            modelBuilder.Entity<Resource>().ToTable("Vega_Languages_Resources");
            modelBuilder.Entity<Language>().ToTable("Vega_Languages_Languages");
            base.OnModelCreating(modelBuilder);
        }

#endregion Methods

        #region IUnitOfWork Members
        /// <summary>
        /// Sets this instance.
        /// </summary>
        /// <typeparam name="TEntity">The type of the entity.</typeparam>
        /// <returns></returns>
        public new IDbSet<TEntity> Set<TEntity>() where TEntity : class
        {
            return base.Set<TEntity>();
        }
        #endregion
    }
```

در تعاریف کلاسهایی که از IDBContext ارث می‌برن اکسپورت شدن (این یک نمونه از کلاس‌های منه)
در طرف دیگر برای لود کردن کلاس زیر نوشتم

```
public class LoadContexts
    {
        public LoadContexts()
        {
            var directoryPath = HttpRuntime.BinDirectory;//AppDomain.CurrentDomain.BaseDirectory;
//"Dll folder path";

            var directoryCatalog = new DirectoryCatalog(directoryPath, "*.dll");

            var aggregateCatalog = new AggregateCatalog();
            aggregateCatalog.Catalogs.Add(directoryCatalog);

            var container = new CompositionContainer(aggregateCatalog);
            container.ComposeParts(this);
        }

        //[Import]
        //public IPlugin Plugin { get; set; }

        [ImportMany]
        public IEnumerable<IDbContext> Contexts { get; set; }
    }
```

و در کانتکس اصلی برنامه این پلاگین هارو لود می‌کنم

```
public class MainContext : DbContext, IUnitOfWork
    {
        public MainContext() : base("LocalSqlServer") { }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
            var contextList = new LoadContexts(); //ObjectFactory.GetAllInstances<IDbContext>();
            foreach (var context in contextList.Contexts)
                context.Setup(modelBuilder);

            Database.SetInitializer(new MigrateDatabaseToLatestVersion<MainContext, Configuration>());
            //Database.SetInitializer(new DropCreateDatabaseAlways<MainContext>());
        }

        /// <summary>
        /// Sets this instance.
        /// </summary>
        /// <typeparam name="TEntity">The type of the entity.</typeparam>
        /// <returns></returns>
        public IDbSet<TEntity> Set<TEntity>() where TEntity : class
```

```
        {
            return base.Set<TEntity>();
        }
    }
```

با موفقیت همه پلاگین‌ها لود میشه و مشکلی در عملیات نیست. اما Attributeهای کلاس هارو نمیشناسه. مثلا پیام خطا تعریف شده در MVC نمایش داده نمیشه چون وجود نداره ولی وقتی کلاس مورد نظر از IValidatableObject ارث میبره خطای‌های من نمایش داده میشه. می‌خوام از خود متادیتاهای استاندارد استفاده کنم.

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۷/۱۵ ۹:۵۱

- بنابراین روی ساختار دیتابیس تاثیر داره. مثالش هم پیوست شد برای آزمایش.

- عمل نکردن خطاهای اعتبارسنجی به بود و نبود یک سری از تعاریف لازم در View هم بر می‌گرده. توضیح شما یعنی عمل نکردن اعتبارسنجی سمت کلاینت ولی عمل کردن اعتبارسنجی سمت سرور. به ترتیب باید jquery.validate.min.js ، jquery.min.js و jquery.validate.unobtrusive.min.js به View الحاق شده باشند. تنظیمات ClientValidationEnabled و UnobtrusiveJavaScriptEnabled در وب کانفیگ فعال باشند. از متدهایی مانند ValidationMessageFor استفاده شده باشد. این متدها یک سری ویژگی‌های خاص unobtrusive رو به عناصر HTML برای شناسایی توسط jquery.validate اضافه می‌کنند و بدون این‌ها عملا اعتبارسنجی سمت کاربر رخ نمی‌ده.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۷/۱۵ ۲۰:۳۵

ممنون از پاسخ شما.
اما مهندس توی کامنت قبلی گفتم "با موفقیت همه پلاگین‌ها لود میشه و مشکلی در عملیات نیست. اما Attributeهای کلاس هارو نمیشناسه. مثلا پیام خطا تعریف شده در MVC نمایش داده نمیشه چون وجود نداره ولی وقتی کلاس مورد نظر از IValidatableObject ارث میبره خطای‌های من نمایش داده میشه. می‌خوام از خود متادیتاهای استاندارد استفاده کنم. "

پس خطا نمایش داده میشه و مشکلی توی طرف کلاینت ندارم.
در هر صورت ممنون از اینکه وقت گذاشتید و پاسخ دادید.

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۷/۱۵ ۲۱:۵۳

پردازش IValidatableObject سمت سرور هست. فقط نمایش نتیجه این نوع اعتبار سنجی سمت سرور، در سمت کلاینت بعد از post back کامل نمایش داده میشه.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۷/۱۶ ۱۰:۵۸

بله درسته
بعد از تست متوجه شدم وقتی خودم متا دیتا تعریف می‌کنم(ارث بری از متادیتای استاندارد) خطای طرف کلاینت عمل نمی‌کنه اما وقتی از متادیتای استاندارد خود دات نت استفاده میکنم خطای طرف کلاینت فعال نمیشه

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۷/۱۶ ۱۱:۴

مطلب چطور باید سؤال پرسید رو اگر از ابتدا رعایت کرده بودید بحث به درازا نمی‌کشید. (سؤالی که در هر مرحله داره صورت مساله توضیح داده نشده‌اش عوض میشه؛ مثالی که نمی‌تونی از راه دور سریع تستش کنی و جزئیات متغیرش مشخص نیست)

نویسنده: reza110
تاریخ: ۱۳۹۲/۰۹/۱۸ ۱۷:۱۴

اگر امکان دارد سورس مثال را در سایت قرار دهید.
با تشکر

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۹/۱۸ ۱۸:۳۴

من کمی بالاتر ارسالش کردم.

عنوان: **تغییر PartCreationPolicy پیش فرض در MEF**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۳/۲۲ ۸:۳۰
آدرس: www.dotnettips.info
برچسب‌ها: Default, MEF, Dependency Injection, PartCreationPolicy

تشریح مسئله : در MEF به صورت پیش فرض نوع نمونه ساخته شده از اشیا به صورت Singleton است. در صورتی که بخواهیم یک نمونه جدید از اشیا به ازای هر درخواست ساخته شود باید PartCreationPolicyAttribute رو به ازای هر کلاس مجددا تعریف کنیم و نوع اون رو به NonShared تغییر دهیم. در پروژه‌های بزرگ این مسئله کمی آزار دهنده است. برای تغییر رفتار Container در MEF هنگام نمونه سازی Objectها باید چه کار کرد؟

نکته: آشنایی با مفاهیم MEF برای درک بهتر مطالب الزامی است.

*در صورتی که با مفاهیم MEF آشنایی ندارید می‌توانید از اینجا شروع کنید.

در MEF سه نوع PartCreationPolicy وجود دارد:
#1 Shared : آبجکت مورد نظر فقط یک بار در کل طول عمر Composition Container ساخته می‌شود.(Singleton)
#2 NonShared : آبجکت مورد نظر به ازای هر درخواست دوباره نمونه سازی می‌شود.
#3 Any : از حالت پیش فرض CompositionContainar برای نمونه سازی استفاده میشود که همان مورد اول است(Shared)

در اکثر پروژه‌ها ساخت نمونه اشیا به صورت Singleton میسر نیست و باعث اشکال در پروژه می‌شود. برای حل این مشکل باید PartCreationPolicy رو برای هر شی مجزا تعریف کنیم. برای مثال

```
[Export]
    [PartCreationPolicy( CreationPolicy.NonShared )]
    internal class ShellViewModel : ViewModel<IShellView>
    {
        private readonly DelegateCommand exitCommand;


        [ImportingConstructor]
        public ShellViewModel( IShellView view )
            : base( view )
        {
            exitCommand = new DelegateCommand( Close );
        }
    }
```

حال فرض کنید تعداد آبجکت شما در یک پروژه بیش از چند صد تا باشد. در صورتی که یک مورد را فراموش کرده باشید و UnitTest قوی و مناسب در پروژه تعبیه نشده باشد قطعا در طی پروژه مشکلاتی به وجود خواهد آمد و امکان Debug سخت خواهد شد.
برای حل این مسئله بهتر است که رفتار Composition Container رو در هنگام نمونه سازی تغییر دهیم. یعنی آبجکت‌ها به صورت پیش فرض به صورت NonShared تولید شوند و در صورت نیاز به نمونه Shared این Attribute رو در کلاس مورد نظر استفاده کنیم. کافیست از کلاس Composition Container که قلب MEF محسوب می‌شود ارث برده و رفتار مورد نظر را Override کنیم.
برای نمونه :

```
public class CustomCompositionContainer : CompositionContainer
{
    public CustomCompositionContainer(ComposablePartCatalog catalog)
        : base(catalog)
    {
    }

    protected override IEnumerable<Export> GetExportsCore(ImportDefinition definition)
    {
        definition = AdaptDefinition(definition);

        return base.GetExportsCore(definition);
    }
```

```
    private ImportDefinition AdaptDefinition(ImportDefinition definition)
    {
        ContractBasedImportDefinition namedDefinition = definition as ContractBasedImportDefinition;
        if (namedDefinition != null && namedDefinition.RequiredCreationPolicy == CreationPolicy.Any)
        {
            definition = new ContractBasedImportDefinition(namedDefinition.ContractName,
                                                           namedDefinition.RequiredMetadata,
                                                           namedDefinition.Cardinality,
                                                           namedDefinition.IsRecomposable,
                                                           namedDefinition.IsPrerequisite,
                                                         CreationPolicy.NonShared);
        }

        return definition;
    }
}
```

مشاهده می‌کنید که متد GetExportCore در کلاس بالا Override شده است و توسط متد AdaptDefinition اگر PartCreationPolicy به صورت Any بود نمونه ساخته شده به صورت NonShared ایجاد می‌شود.

حال فقط کافیست در پروژه به جای استفاده از CompositionContainer از CustomCompositionContainer استفاده کنیم.