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 GetAll , Add 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: **پیاده سازی UnitOfWork برای BrightStarDb**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۳/۰۲/۲۶ ۲۰:۰
آدرس: www.dotnettips.info
گروه‌ها: Design patterns, NoSQL, UnitOfWork, BrightstarDB

در این پست با BrightStarDb و مفاهیم اولیه آن آشنا شدید. همان طور که پیش‌تر ذکر شد BrightStarDb از تراکنش‌ها جهت ذخیره اطلاعات پشتیبانی می‌کند. قصد داریم روش شرح داده شده در اینجا را بر روی BrightStarDb فعال کنیم. ابتدا بهتر است با روش ساخت مدل در B*Db آشنا شویم.
*یکی از پیش نیازهای این پست مطالعه این دو مطلب (^) و (^) می‌باشد.
فرض می‌کنیم در دیتابیس مورد نظر یک Store به همراه یک جدول به صورت زیر داریم:

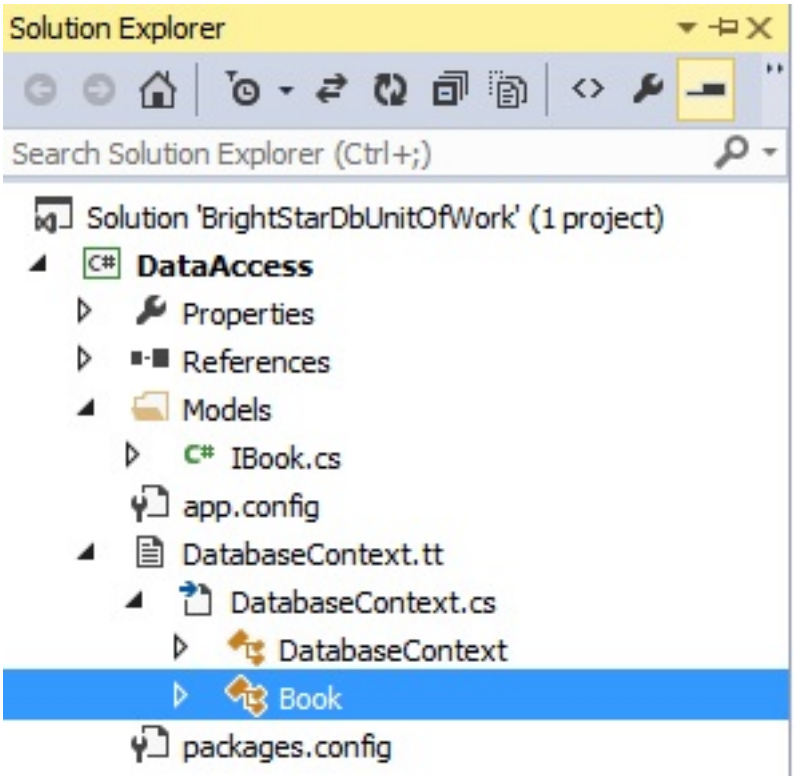
```
[Entity]
    public interface IBook
    {
        [Identifier]
        string Id { get; }

        string Title { get; set; }

        string Isbn { get; set; }
    }
```

بر روی پروژه مورد نظر کلیک راست کرده و گزینه Add new Item را انتخاب نمایید. از برگه Data گزینه BrightStar Entity Context را انتخاب کنید

بعد از انتخاب گزینه بالا یک فایل با پسوند tt به پروژه اضافه خواهد شد که وظیفه آن جستجو در اسمبلی مورد نظر و پیدا کردن تمام اینترفیس هایی که دارای EntityAttribute هستند و همچنین ایجاد کلاس‌های متناظر جهت پیاده سازی اینترفیس‌های بالا است. در نتیجه ساختار پروژه تا این جا به صورت زیر خواهد شد.



واضح است که فایلی به نام Book به عنوان پیاده سازی مدل IBook به عنوان زیر مجموعه فایل DatabaseContext.tt به پروژه اضافه شده است.

تا اینجا برای استفاده از Context مورد نظر باید به صورت زیر عمل نمود:

```
DatabaseContext context = new DatabaseContext();
  context.Books.Add(new Book());
```

Context پیش فرض ساخته شده توسط B*Db از Generic DbSetهای معادل EF پشتیبانی نمی‌کند و از طرفی IUnitOfWork مورد نظر به صورت زیر است

```
public interface IUnitOfWork
    {
        BrightstarEntitySet<T> Set<T>() where TEntity : class;
        void DeleteObject(object obj);
         void SaveChanges();
    }
```

در اینجا فقط به جای IDbSet از BrightStarDbSet استفاده شده است. همان طور که در این مقاله توضیح داده شده است، برای پیاده سازی مفهوم UnitOfWork؛ نیاز است تا کلاس DatabaseContext که نماینده BrightStarDbContext پروژه است، از اینترفیس IUnitOfWork طراحی شده ارث بری کند. جهت انجام این مهم و همچنین جهت اضافه کردن قابلیت ایجاد Generic DbSetها نیز باید کمی در فایل Template Generator تغییر ایجاد نماییم. این تغییرات را قبلا در طی یک پروژه ایجاد کرده‌ام و شما می‌توانید آن را از اینجا دریافت کنید. بعد از دانلود کافیست فایل DatabaseContext.tt مورد نظر را در پروژه خود کپی کرده و گزینه Run Custom Tools را فراخوانی نمایید.

نکته: برای حذف یک آبجکت از Store، باید از متد DeleteObject تعبیه شده در Context استفاده نماییم. در نتیجه متد مورد نظر نیز در اینترفیس بالا در نظر گرفته شده است.

استفاده از IOC Container جهت رجیستر کردن IUnitOfWrok

در این قدم باید IUnitOfWork را در یک IOC container رجیستر کرده تا در جای مناسب عملیات وهله سازی از آن میسر باشد.
من در اینجا از Castle Windsor Container استفاده کردم. کلاس زیر این کار را برای ما انجام خواهد داد:

```
public class DependencyResolver
    {
        public static void Resolve(IWindsorContainer container)
        {
            var context = new
DatabaseContext("type=embedded;storesdirectory=c:\brightstar;storename=test ");
            container.Register(Component.For<IUnitOfWork>().Instance(context).LifestyleTransient());
        }
    }
```

حال کافیست در کلاس‌های سرویس برنامه UnitOfWork رجیستر شده را به سازنده آن‌ها تزریق نماییم.

```
public class BookService
    {
        public BookService(IUnitOfWork unitOfWork)
        {
            UnitOfWork = unitOfWork;
        }

        public IUnitOfWork UnitOfWork
        {
            get;
            private set;
        }

        public IList<IBook> GetAll()
        {
            return UnitOfWork.Set<IBook>().ToList();
        }

        public void Add()
        {
            UnitOfWork.Set<IBook>().Add(new Book());
        }

        public void Remove(IBook entity)
        {
            UnitOfWork.DeleteObject(entity);
        }
    }
```

سایر موارد دقیقا معادل مدل EF آن است.
نکته: در حال حاضر امکان جداسازی مدل‌های برنامه (تعاریف اینترفیس) در قالب یک پروژه دیگر(نظیر مدل CodeFirst در EF) در B*Db امکان پذیر نیست.
نکته : برای اضافه کردن آیتم جدید به Store نیاز به وهله سازی از اینترفیس IBook داریم. کلاس Book ساخته شده توسط DatabaseContext.tt در عملیات Insert و update کاربرد خواهد داشت.