عنوان: طراحی افزونه پذیر با ASP.NET MVC 4.x/5.x - قسمت سوم
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۱/۲۸ ۱۲:۱۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, MVC, Plugin

پس از ب ررسی ساختار یک پروژه‌ی افزونه پذیر و همچنین بهبود توزیع فایل‌های استاتیک آن ، اکنون نوبت به کار با داده‌ها است. هدف اصلی آن نیز داشتن مدل‌های اختصاصی و مستقل Entity framework code-first به ازای هر افزونه است و سپس بارگذاری و تشخیص خودکار آن‌ها در Context مرکزی برنامه.

پیشنیازها

- آشنایی با مباحث Migrations در EF Code first
- آشنایی با مباحث الگوی واحد کار
- چگونه مدل‌های EF را به صورت خودکار به Context اضافه کنیم؟
- چگونه تنظیمات مدل‌های EF را به صورت خودکار به Context اضافه کنیم؟

کدهایی را که در این قسمت مشاهده خواهید کرد، در حقیقت همان برنامه‌ی توسعه یافته « آشنایی با مباحث الگوی واحد کار » است و از ذکر قسمت‌های تکراری آن جهت طولانی نشدن مبحث، صرفنظر خواهد شد. برای مثال Context و مدل‌های محصولات و گروه‌های آن‌ها به همراه کلاس‌های لایه سرویس برنامه‌ی اصلی، دقیقا همان کدهای مطلب « آشنایی با مباحث الگوی واحد کار » است.

تعریف domain classes مخصوص افزونه‌ها

در ادامه‌ی پروژه‌ی افزونه پذیر فعلی، پروژه‌ی class library جدیدی را به نام MvcPluginMasterApp.Plugin1.DomainClasses اضافه خواهیم کرد. از آن جهت تعریف کلاس‌های مدل افزونه‌ی یک استفاده می‌کنیم. برای مثال کلاس News را به همراه تنظیمات Fluent آن به این پروژه‌ی جدید اضافه کنید:

```
using System.Data.Entity.ModelConfiguration;

namespace MvcPluginMasterApp.Plugin1.DomainClasses
{
    public class News
    {
        public int Id { set; get; }

        public string Title { set; get; }

        public string Body { set; get; }
    }

    public class NewsConfig : EntityTypeConfiguration<News>
    {
        public NewsConfig()
        {
            this.ToTable("Plugin1_News");
            this.HasKey(news => news.Id);
            this.Property(news => news.Title).IsRequired().HasMaxLength(500);
            this.Property(news => news.Body).IsOptional().IsMaxLength();
        }
    }
}
```

این پروژه برای کامپایل شدن نیاز به بسته‌ی نیوگت ذیل دارد:

```
PM> install-package EntityFramework
```

مشکل! برنامه‌ی اصلی، همانند مطلب « آشنایی با مباحث الگوی واحد کار » دارای domain classes خاص خودش است به همراه تنظیمات Context ایی که صریحا در آن مدل‌های متناظر با این پروژه در معرض دید EF قرار گرفته‌اند:

```
public class MvcPluginMasterAppContext : DbContext, IUnitOfWork
{
    public DbSet<Category> Categories { set; get; }
    public DbSet<Product> Products { set; get; }
```

اکنون برنامه‌ی اصلی چگونه باید مدل‌ها و تنظیمات سایر افزونه‌ها را یافته و به صورت خودکار به این Context اضافه کند؟ با توجه به اینکه این برنامه هیچ ارجاع مستقیمی را به افزونه‌ها ندارد.

تغییرات اینترفیس Unit of work جهت افزونه پذیری

در ادامه، اینترفیس بهبود یافته‌ی IUnitOfWork را جهت پذیرش DbSetهای پویا و همچنین EntityTypeConfigurationهای پویا، ملاحظه می‌کنید:

```
namespace MvcPluginMasterApp.PluginsBase
{
    public interface IUnitOfWork : IDisposable
    {
        IDbSet<TEntity> Set<TEntity>() where TEntity : class;
        int SaveAllChanges();
        void MarkAsChanged<TEntity>(TEntity entity) where TEntity : class;
        IList<T> GetRows<T>(string sql, params object[] parameters) where T : class;
        IEnumerable<TEntity> AddThisRange<TEntity>(IEnumerable<TEntity> entities) where TEntity :
class;
        void SetDynamicEntities(Type[] dynamicTypes);
        void ForceDatabaseInitialize();
        void SetConfigurationsAssemblies(Assembly[] assembly);
    }
}
```

متدهای جدید آن:

SetDynamicEntities : توسط این متد در ابتدای برنامه، نوع‌های مدل‌های جدید افزونه‌ها به صورت خودکار به Context اضافه خواهند شد.

SetConfigurationsAssemblies : کار افزودن اسمبلی‌های حاوی تعاریف EntityTypeConfigurationهای جدید و پویا را به عهده دارد.

ForceDatabaseInitialize: سبب خواهد شد تا مباحث migrations، پیش از شروع به کار برنامه، اعمال شوند.

در کلاس Context ذیل، نحوه‌ی پیاده سازی این متدهای جدید را ملاحظه می‌کنید:

```
namespace MvcPluginMasterApp.DataLayer.Context
{
    public class MvcPluginMasterAppContext : DbContext, IUnitOfWork
    {
        private readonly IList<Assembly> _configurationsAssemblies = new List<Assembly>();
        private readonly IList<Type[]> _dynamicTypes = new List<Type[]>();

        public void ForceDatabaseInitialize()
        {
            Database.Initialize(force: true);
        }

        public void SetConfigurationsAssemblies(Assembly[] assemblies)
        {
            if (assemblies == null) return;

            foreach (var assembly in assemblies)
            {
                _configurationsAssemblies.Add(assembly);
            }
        }

        public void SetDynamicEntities(Type[] dynamicTypes)
        {
            if (dynamicTypes == null) return;
            _dynamicTypes.Add(dynamicTypes);
        }
```

```
        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            addConfigurationsFromAssemblies(modelBuilder);
            addPluginsEntitiesDynamically(modelBuilder);
            base.OnModelCreating(modelBuilder);
        }

        private void addConfigurationsFromAssemblies(DbModelBuilder modelBuilder)
        {
            foreach (var assembly in _configurationsAssemblies)
            {
                modelBuilder.Configurations.AddFromAssembly(assembly);
            }
        }

        private void addPluginsEntitiesDynamically(DbModelBuilder modelBuilder)
        {
            foreach (var types in _dynamicTypes)
            {
                foreach (var type in types)
                {
                    modelBuilder.RegisterEntityType(type);
                }
            }
        }
    }
}
```

در متد استاندارد OnModelCreating، فرصت افزودن نوع‌های پویا و همچنین تنظیمات پویای آن‌ها وجود دارد. برای این منظور می‌توان از متدهای modelBuilder.RegisterEntityType و modelBuilder.Configurations.AddFromAssembly کمک گرفت.

بهبود اینترفیس IPlugin جهت پذیرش نوع‌های پویای EF

در قسمت اول، با اینترفیس IPlugin آشنا شدیم. هر افزونه باید دارای کلاسی باشد که این اینترفیس را پیاده سازی می‌کند. از آن جهت دریافت تنظیمات و یا ثبت تنظیمات مسیریابی و امثال آن استفاده می‌شود.
در اینجا متد GetEfBootstrapper آن کار دریافت تنظیمات EF هر افزونه را به عهد دارد.

```
namespace MvcPluginMasterApp.PluginsBase
{
    public interface IPlugin
    {
        EfBootstrapper GetEfBootstrapper();
        //...به همراه سایر متدهای مورد نیاز
    }

    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; }
    }
}
```

ConfigurationsAssemblies مشخص کننده‌ی اسمبلی‌هایی است که حاوی تعاریف EntityTypeConfigurationهای افزونه‌ی جاری هستند.
DomainEntities بیانگر لیست مدل‌ها و موجودیت‌های هر افزونه است.

DatabaseSeeder کار دریافت منطق متد Seed را بر عهده دارد. برای مثال اگر افزونه‌ای نیاز است در آغاز کار تشکیل جداول آن، دیتای پیش فرض و خاصی را در بانک اطلاعاتی ثبت کند، می‌توان از این متد استفاده کرد. اگر دقت کنید این Action یک وهله از IUnitOfWork را به افزونه ارسال می‌کند. بنابراین در این طراحی جدید، اینترفیس IUnitOfWork به پروژه‌ی MvcPluginMasterApp.PluginsBase منتقل می‌شود. به این ترتیب دیگر نیازی نیست تا تک تک افزونه‌ها ارجاع مستقیمی را به DataLayer پروژه‌ی اصلی پیدا کنند.

تکمیل متد GetEfBootstrapper در افزونه‌ها

اکنون جهت معرفی مدل‌ها و تنظیمات EF آن‌ها، تنها کافی است متد GetEfBootstrapper هر افزونه را تکمیل کنیم:

```
namespace MvcPluginMasterApp.Plugin1
{
    public class Plugin1 : IPlugin
    {
        public EfBootstrapper GetEfBootstrapper()
        {
            return new EfBootstrapper
            {
                DomainEntities = new[] { typeof(News) },
                ConfigurationsAssemblies = new[] { typeof(NewsConfig).Assembly },
                DatabaseSeeder = uow =>
                {
                    var news = uow.Set<News>();
                    if (news.Any())
                    {
                        return;
                    }

                    news.Add(new News
                    {
                        Title = "News 1",
                        Body = "news 1 news 1 news 1 ...."
                    });

                    news.Add(new News
                    {
                        Title = "News 2",
                        Body = "news 2 news 2 news 2 ...."
                    });
                }
            };
        }
```

در اینجا نحوه‌ی معرفی مدل‌های جدید را توسط خاصیت DomainEntities و تنظیمات متناظر را به کمک خاصیت ConfigurationsAssemblies مشاهده می‌کنید. باید دقت داشت که هر اسمبلی فقط باید یکبار معرفی شود و مهم نیست که چه تعداد تنظیمی در آن وجود دارند. کار یافتن کلیه‌ی تنظیمات از نوع EntityTypeConfigurationها به صورت خودکار توسط EF صورت می‌گیرد.
همچنین توسط delegate ایی به نام DatabaseSeeder، نحوه‌ی دسترسی به متد Set واحد کار و سپس استفاده‌ی از آن، برای تعریف متد Seed سفارشی نیز تکمیل شده‌است.

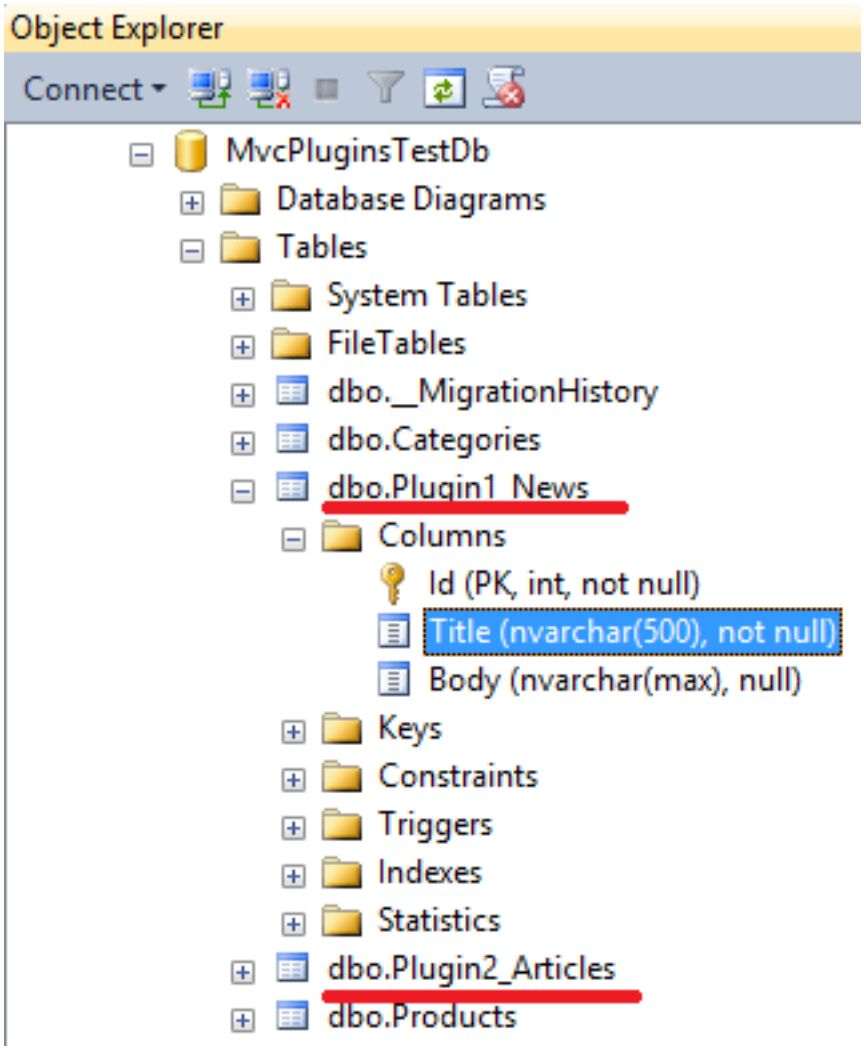
تدارک یک راه انداز EF، پیش از شروع به کار برنامه

در پوشه‌ی App_Start پروژه‌ی اصلی یا همان MvcPluginMasterApp، کلاس جدید EFBootstrapperStart را با کدهای ذیل اضافه کنید:

```
[assembly: PreApplicationStartMethod(typeof(EFBootstrapperStart), "Start")]
namespace MvcPluginMasterApp
{
    public static class EFBootstrapperStart
    {
        public static void Start()
        {
            var plugins = SmObjectFactory.Container.GetAllInstances<IPlugin>().ToList();
            using (var uow = SmObjectFactory.Container.GetInstance<IUnitOfWork>())
```

```
            {
                initDatabase(uow, plugins);
                runDatabaseSeeders(uow, plugins);
            }
        }

        private static void initDatabase(IUnitOfWork uow, IEnumerable<IPlugin> plugins)
        {
            foreach (var plugin in plugins)
            {
                var efBootstrapper = plugin.GetEfBootstrapper();
                if (efBootstrapper == null) continue;

                uow.SetDynamicEntities(efBootstrapper.DomainEntities);
                uow.SetConfigurationsAssemblies(efBootstrapper.ConfigurationsAssemblies);
            }

            Database.SetInitializer(new MigrateDatabaseToLatestVersion<MvcPluginMasterAppContext,
Configuration>());
            uow.ForceDatabaseInitialize();
        }

        private static void runDatabaseSeeders(IUnitOfWork uow, IEnumerable<IPlugin> plugins)
        {
            foreach (var plugin in plugins)
            {
                var efBootstrapper = plugin.GetEfBootstrapper();
                if (efBootstrapper == null || efBootstrapper.DatabaseSeeder == null) continue;

                efBootstrapper.DatabaseSeeder(uow);
                uow.SaveAllChanges();
            }
        }
    }
}
```

در اینجا یک راه انداز سفارشی از نوع PreApplicationStartMethod تهیه شده‌است. Pre بودن آن به معنای اجرای کدهای متد Start این کلاس، پیش از آغاز به کار برنامه و پیش از فراخوانی متد Application_Start فایل Global.asax.cs است.
همانطور که ملاحظه می‌کنید، ابتدا لیست تمام افزونه‌های موجود، به کمک StructureMap دریافت می‌شوند. سپس می‌توان در متد initDatabase به متد GetEfBootstrapper هر افزونه دسترسی یافت و توسط آن تنظیمات مدل‌ها را یافته و به Context اصلی برنامه اضافه کرد. سپس با فراخوانی ForceDatabaseInitialize تمام این موارد به صورت خودکار به بانک اطلاعاتی اعمال خواهند شد.
کار متد runDatabaseSeeders، یافتن DatabaseSeeder هر افزونه، اجرای آن‌ها و سپس فراخوانی متد SaveAllChanges در آخر کار است.



کدهای کامل این سری را از اینجا می‌توانید دریافت کنید:

MvcPlugin

نظرات خوانندگان

نویسنده: غلامرضا ربال
تاریخ: ۱۳۹۴/۰۱/۲۸ ۱۵:۴۲

با تشکر.
اگر لازم باشد پلاگین ما به مدل موجود در پروژه اصلی دسترسی داشته باشد باید به چه شکلی عمل کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۱/۲۸ ۱۶:۲۳

- در عمل کل برنامه و تمام افزونه‌های آن از یک IUnitOfWork استفاده می‌کنند؛ یعنی تمام آن‌ها به تمام مدل‌های اضافه شده‌ی به Context اصلی برنامه دسترسی دارند. بنابراین هر پلاگین در صورت نیاز امکان دسترسی به مدل‌های برنامه‌ی اصلی یا سایر افزونه‌ها را دارا است. تمام این افزونه‌ها در کنار هم یک سیستم را تشکیل می‌دهند و مانند شکل انتهای بحث، از یک بانک اطلاعاتی استفاده می‌کنند.
- به همین جهت تنها کاری که باید انجام داد، افزودن ارجاعی به کلاس‌های مدل مورد نظر هست. پس از آن شبیه به کاری که در DatabaseSeeder انجام شده، می‌توان با استفاده از متد Set، به کلیه امکانات مدلی خاص دسترسی یافت:

```
DatabaseSeeder = uow =>
{
     var news = uow.Set<News>();
```

اگر نمی‌خواهید ارجاعی را به کلاس‌های مدل مورد نظر اضافه کنید، با توجه به اینکه این کلاس‌ها هم اکنون جزئی از وهله‌ی Context ارائه شده‌ی توسط IUnitOfWork هستند، باید متوسل به Reflection و تدارک متد Set ویژه‌ای شوید که بجای News، معادل رشته‌ای آن‌را دریافت کند.
ولی در کل افزودن ارجاعی به کلاس‌های مدل دیگر، مشکل ساز نیست؛ چون این کلاس‌ها عملا منطق خاصی را پیاده سازی نمی‌کنند و همچنین وابستگی خاصی هم به پروژه‌ی خاصی ندارند. یک سری کلاس دارای خاصیت‌های get/set دار معمولی هستند به همراه تنظیمات آن‌ها.

نویسنده: پوریا انوشیروانی
تاریخ: ۱۳۹۴/۰۲/۰۵ ۱۷:۳۷

من در قسمت کلاس اهراز هویت به کمی مشکل برخوردم .
کلاس زیر رو دارم که البته در پلاگینی جدا نوشته شده :

```
public class CmsUser : IdentityUser
    {
        public string DisplayName { get; set; }

    }
```

هر پست باید دارای یک نویسنده باشد که می‌خوام از CmsUser استفاده کنم .
این وابستگی و ارتباط باید کجا نوشته شود ؟

```
public class Post
    {

        private IList<string> _tags = new List<string>();
        public int id { get; set; }
        public string name { get; set; }
        public string slug { get; set; }
        public string description { get; set; }
        public DateTime? publishTime { get; set; }
        public string content { get; set; }
        public IList<string> tags
        {
```

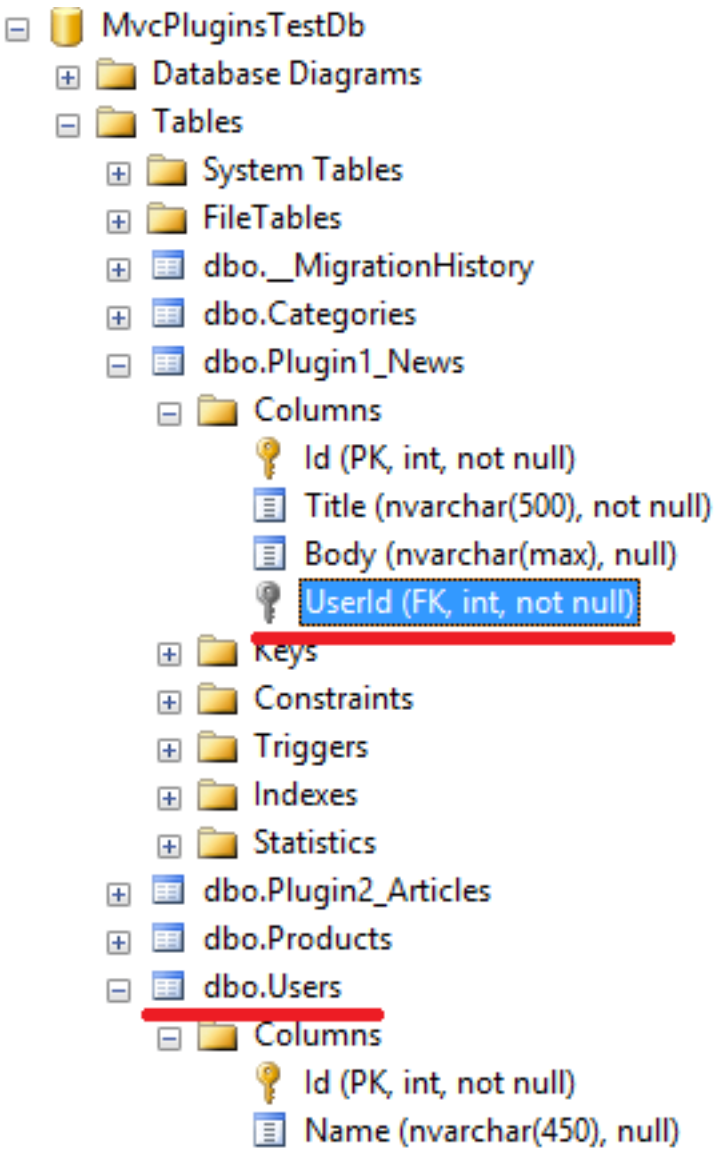
```
            get { return _tags; }
            set { _tags = value; }
        }
        public string CombineTags
        {
            get { return string.Join(",", _tags); }
            set { _tags = value.Split(',').Select(x => x.Trim()).ToList(); }
        }

        public string AuthorID { get; set; }

      //  [ForeignKey("AuthorID")]
      //  public CmsUser Author { get; set; }
    }
```

هرجا می‌نویسم برای کلیدهای خارجیش اخطار میده و این که زیاد نمی‌خوام پلاگین پست از وجود CmsUser با خبر باشه

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۲/۰۵ ۱۹:۴۹

- موجودیت‌های مشترک بین افزونه‌ها را در یک پروژه‌ی مجزا قرار دهید؛ مانند: CommonEntities
- از این پروژه‌ی مشترک، ارجاعی را به افزونه‌های مورد نظر اضافه کنید.

پروژه‌ی جاری جهت افزودن کلید خارجی به کاربران مشترک بین تمام افزونه‌ها به روز شد، با این تغییرات و با این خروجی (که در آن در هر دو افزونه‌ی تعریف شده، ارجاعی به کلاس User مشترک هست):



نویسنده: پوریا انوشیروانی
تاریخ: ۱۳۹۴/۰۲/۰۶ ۱۰:۵۴

برای استفاده از Identity چون دوتا Context ایجاد میشه ، کلی دردسر میشه
چطوری این رو بسازیم که تداخل هم ایجاد نشه :

```
public class Cmscontex : IdentityDbContext<CmsUser>
```

identity به کلاسی که از IdentityDbContext ارث برده شده نیاز داره . در صورتی که ما قبلا

```
public class UIAppContext : DbContext, IUnitOfWork
```

اگه دوتا Context ایجاد کنیم ارتباط کلاس‌ها خیلی بد میشه .
لطفا اگر میشه در رابط با identity هم مثالی بزنید

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۲/۰۶ ۱۱:۵۶

مطلب « اعمال تزریق وابستگی‌ها به مثال رسمی ASP.NET Identity » را مطالعه کنید. نیازی نیست چندین Context ایجاد کنید. اینبار Context اصلی برنامه همان ApplicationDbContext خواهد بود. یعنی در مثال جاری، کلاس MvcPluginMasterAppContext با ApplicationDbContext **جایگزین** می‌شود؛ به همراه افزودن کدهای سفارشی کلاس MvcPluginMasterAppContext به ApplicationDbContext.

نویسنده: غلامرضا ربال
تاریخ: ۱۳۹۴/۰۲/۰۷ ۱۶:۲۲

برای ارتباطهای n به n به چه شکلی باید عمل کرد؟ منظورم موقعی است که باید لیستی از کلاس دیگر در کلاس یوزر ما موجود باشد.
با تشکر

نویسنده: میثم خادمی
تاریخ: ۱۳۹۴/۰۲/۲۲ ۹:۸

سلام

پس از لود شدن dllهای پلاگین امکان استفاده از روش ^ و ^ برای اضافه کردن و بارگذاری اطلاعات در Context وجود ندارد ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۲/۲۲ ۱۰:۳

قسمت «تغییرات اینترفیس Unit of work جهت افزونه پذیری » در متن فوق از همین مطالب استفاده می‌کند (متدهای addConfigurationsFromAssemblies و addPluginsEntitiesDynamically).

نویسنده: میثم خادمی
تاریخ: ۱۳۹۴/۰۲/۲۲ ۱۱:۲۴

ولی در عمل در EfBootstrapper باید برای کلیه کلاس‌های Ef مقدار DomainEntities و برای کلیه کلاس‌های Fluent Api مقدار ConfigurationsAssemblies مشخص شود . در حالی که در لینک هایی که ^ و ^ کلیه کلاس هایی که از BaseEntity مشتق شده اند به Context اضافه شده و نیازی به معرفی ندارند و کلاس‌ها تنظیمات FluentApiها نیز از روی nameSpace مشخص شده اضافه می‌شوند.

بهتر نیست از BaseEntity استفاده بشود؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۲/۲۲ ۱۱:۳۱

- این روش دقیق‌تر هست، با احتمال اشتباه و سعی و خطای کمتر و سریعتر.
- modelBuilder.Configurations.AddFromAssembly متد توکار خود EF هست(در نگارش‌های اخیر آن) و به صورت خودکار کلاس‌های مشتق شده‌ی از EntityTypeConfiguration را اسکن می‌کند.
- در کل هر طور که صلاح می‌دانید روش اسکن آن‌را تغییر دهید. اصل ماجرا، یعنی متدهای addConfigurationsFromAssemblies و addPluginsEntitiesDynamically تفاوتی نخواهند کرد.

نویسنده: فرزین بیاتی
تاریخ: ۱۳۹۴/۰۳/۱۱ ۱۶:۳۹

سلام

آیا میشه با این روش، PartialView رو هم لود کرد ؟ یعنی PartialView داخل Plugin به عنوان EmbedeResource باشه و از پروژه اصلی لودش کرد ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۳/۱۱ ۱۶:۴۵

- به چه مشکلی برخوردید دقیقا؟ اجرا نشد؟ خطا گرفتید؟
- partial view هم مانند یک view معمولی باید custom tools اش به razor generator تنظیم شود. بعد از آن کار کردن با آن معمولی و مانند قبل خواهد بود.

نویسنده: فرزین بیاتی
تاریخ: ۱۳۹۴/۰۳/۱۲ ۱۰:۴۵

سلام
مشکل نبود فایل RazorGeneratorMvcStart.cs بود که پس از افزودن رفع شد