عنوان: بازنویسی ساده‌تر پیش فرض‌های EF Code first در نگارش 6 آن
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۰۶ ۱۴:۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

فرض کنید مطابق اصول نامگذاری که تعیین کرده‌اید، تمام جداول بانک اطلاعاتی شما باید با پیشوند tbl شروع شوند. برای انجام اینکار در نگارش‌های قبلی EF Code first می‌بایستی از ویژگی Table جهت مزین کردن تمامی کلاس‌ها استفاده می‌شد و یا به ازای تک تک موجودیت‌ها، یک کلاس تنظیمات ویژه را افزود و سپس از متد ToTable برای تعیین نامی جدید، استفاده می‌شد. در EF 6 امکان بازنویسی ساده‌تر پیش فرض‌های تعیین نام جداول، طول فیلدها و غیره، پیش بینی شده‌اند که در ادامه تعدادی از آن‌ها را مرور خواهیم کرد.

تعیین پیشوندی برای نام کلیه‌ی جداول بانک اطلاعاتی

اگر نیاز باشد تا به تمامی جداول تهیه شده، بر اساس نام کلاس‌های مدل‌های برنامه، یک پیشوند tbl اضافه شود، می‌توان با بازنویسی متد OnModelCreating کلاس Context برنامه شروع کرد:

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            // TableNameConvention
            modelBuilder.Types()
                        .Configure(entity => entity.ToTable("tbl" + entity.ClrType.Name));

            base.OnModelCreating(modelBuilder);
        }
```

سپس متد modelBuilder.Types، کلیه موجودیت‌های برنامه را در اختیار قرار داده و در ادامه می‌توان برای مثال از متد ToTable، برای تعیین نامی جدید به ازای کلیه کلاس‌های مدل‌های برنامه استفاده کرد.

تعیین نام دیگری برای کلید اصلی کلیه‌ی جداول برنامه

فرض کنید نیاز است کلیه PKها، با پیشوند نام جدول جاری در بانک اطلاعاتی تشکیل شوند. یعنی اگر نام PK مساوی Id است و نام جدول Menu، نام کلید اصلی نهایی تشکیل شده در بانک اطلاعاتی باید MenuId باشد و نه Id.

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            // PrimaryKeyNameConvention
            modelBuilder.Properties()
                        .Where(p => p.Name == "Id")
                        .Configure(p => p.IsKey().HasColumnName(p.ClrPropertyInfo.ReflectedType.Name +
"Id"));

            base.OnModelCreating(modelBuilder);
        }
```

این مورد نیز با بازنویسی متد OnModelCreating کلاس Context و سپس استفاده از متد modelBuilder.Properties برای دسترسی به کلیه خواص در حال نگاشت، قابل انجام است. در اینجا کلیه خواصی که نام Id دارند، توسط متد IsKey تبدیل به PK شده و سپس به کمک متد HasColumnName، نام دلخواه جدیدی را خواهند یافت.

تعیین حداکثر طول کلیه فیلدهای رشته‌ای تمامی جداول بانک اطلاعاتی

اگر نیاز باشد تا پیش فرض MaxLength تمام خواص رشته‌ای را تغییر داد، می‌توان از پیاده سازی اینترفیس جدید IStoreModelConvention کمک گرفت:

```
public class StringConventions : IStoreModelConvention<EdmProperty>
    {
```

```
        public void Apply(EdmProperty property, DbModel model)
        {
            if (property.PrimitiveType.PrimitiveTypeKind == PrimitiveTypeKind.String)
            {
                property.MaxLength = 450;
            }
        }
    }
```

در اینجا MaxLength کلیه خواص رشته‌ای در حال نگاشت به بانک اطلاعاتی، به 450 تنظیم می‌شود. سپس برای معرفی آن به برنامه خواهیم داشت:

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Conventions.Add<StringConventions>();
            base.OnModelCreating(modelBuilder);
        }
```

توسط متد modelBuilder.Conventions.Add، می‌توان قراردادهای جدید سفارشی را به برنامه افزود.

نظم بخشیدن به تعاریف قراردادهای پیش فرض

اگر علاقمند نیستید که کلاس Context برنامه را شلوغ کنید، می‌توان با ارث بری از کلاس پایه Convention، قراردادهای جدید را تعریف و سپس توسط متد modelBuilder.Conventions.Add، کلاس نهایی تهیه شده را به برنامه معرفی کرد.

```
public class MyConventions : Convention
    {
        public MyConventions()
        {
            // PrimaryKeyNameConvention
            this.Properties()
                .Where(p => p.Name == "Id")
                .Configure(p => p.IsKey().HasColumnName(p.ClrPropertyInfo.ReflectedType.Name + "Id"));

            // TableNameConvention
            this.Types()
                .Configure(entity => entity.ToTable("tbl" + entity.ClrType.Name));
        }
    }
```

مثال‌های بیشتر

اگر به مستندات EF 6 مراجعه کنید ، مثال‌های بیشتری را در مورد بکارگیری اینترفیس IStoreModelConvention و یا بازنویسی قراردادهای موجود، خواهید یافت .

نظرات خوانندگان

نویسنده: رضا
تاریخ: ۱۳۹۲/۱۱/۲۲ ۲۲:۵

سلام ،
من StringConventions رو مطابق گفته شما انجام دادم ولی کار نکرد . هیچ خطایی هم نداد . فکر نمی‌کنید که از <> modelBuilder.Conventions.AddBefore باید استفاده کنیم ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۳ ۰:۵۰

بعد از MaxLengthAttributeConvention باید اضافه شود تا تنظیمات پیش فرض آن‌را بازنویسی کند و چون کلاس پایه MaxLengthAttributeConvention ، کلاس Convention است باید از روش زیر استفاده کرد:

```
public class CustomMaxLengthConvention : Convention
    {
        public CustomMaxLengthConvention()
        {
            this.Properties<string>().Configure(p => p.HasMaxLength(450));
        }
    }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
       modelBuilder.Conventions.AddAfter<MaxLengthAttributeConvention>(new
CustomMaxLengthConvention());
```

نویسنده: مسعود سنائی
تاریخ: ۱۳۹۳/۰۴/۲۷ ۱۴:۳۴

از این امکان چطور برای Ignore کردن یک Property میشه استفاده کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۴/۲۷ ۱۵:۰

```
//برای یک خاصیت مشخص در یک کلاس مشخص
modelBuilder.Entity<Department>().Ignore(t => t.Budget);

// برای یک کلاس مشخص
modelBuilder.Ignore<OnlineCourse>();

// برای نام خاصیتی مشخص در تمام کلاس‌های نگاشت شده
modelBuilder.Types().Configure(c => c.Ignore("IsDeleted"));

// صرفنظر کردن از تمام اینام‌ها در تمام کلاس‌های نگاشت شده
modelBuilder.Types().Configure(typeConfiguration =>
{
    foreach (var property in typeConfiguration.ClrType
        .GetProperties().Where(p => p.PropertyType.IsEnum))
    {
        typeConfiguration.Ignore(property);
    }
});


// صرفنظر کردن از خواصی که با یک نام مشخص شروع می‌شوند در تمام کلاس‌ها
modelBuilder.Types().Configure(typeConfiguration =>
{
    foreach (var property in typeConfiguration.ClrType
        .GetProperties().Where(p => p.Name.StartsWith("someName")))
    {
        typeConfiguration.Ignore(property);
    }
});
```

نویسنده: مسعود سنائی
تاریخ: ۱۳۹۳/۰۵/۰۲ ۱۷:۶

من با EF6 از

```
modelBuilder.Types().Configure(c=>c.Ignore("IsDeleted"));
```

استفاده کردم ولی خطای زیر رو میگیرم:

```
You cannot use Ignore method on the property 'IsDeleted' on type 'MyEntity'
because this type inherits from the type 'BaseEntity' where this property is mapped.
To exclude this property from your model, use NotMappedAttribute or Ignore method on the base type.
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۵/۰۲ ۲۲:۴۴

```
modelBuilder.Entity<BaseEntity>().Ignore(p => p.IsDeleted);
```