عنوان: **بازسازی جدول MigrationHistory با کد نویسی در EF Code first**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۱۳ ۸:۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

فرض کنید با استفاده از ابزار EF Power tools معادل Code first یک بانک اطلاعاتی موجود را تهیه کرده‌اید. اکنون برای استفاده از آن با گردش کاری متداول EF Code first نیاز است تا جدولی را به نام MigrationHistory نیز به این بانک اطلاعاتی اضافه کنیم. از این جدول برای نگهداری سوابق به روز رسانی ساختار بانک اطلاعاتی بر اساس مدل‌های برنامه و سپس مقایسه آن‌ها استفاده می‌شود. یا حتی ممکن است به اشتباه در حین کار با بانک اطلاعاتی این جدول حذف شده باشد. روش باز تولید آن توسط دستورهای پاور شل به سادگی اجرای سه دستور ذیل است:

```
enable-migrations
add-migration Initial -IgnoreChanges
update-database
```

IgnoreChanges سبب می‌شود تا EF فرض کند، تطابق یک به یکی بین مدل‌های برنامه و ساختار جداول بانک اطلاعاتی وجود دارد. سپس بر این اساس، جدول MigrationHistory جدیدی را آغاز می‌کند.

سؤال: چگونه می‌توان همین عملیات را با کدنویسی انجام داد؟

متد UpdateDatabase کلاس ذیل، دقیقا معادل است با اجرای سه دستور فوق :

```
using System.Data.Entity.Migrations;
using System.Data.Entity.Migrations.Design;

namespace EFTests
{
    /// <summary>
    /// Using Entity Framework Code First with an existing database.
    /// </summary>
    public static class CreateMigrationHistory
    {
        /// <summary>
        /// Creates a new '__MigrationHistory' table.
        /// Enables migrations using Entity Framework Code First on an existing database.
        /// </summary>
        public static void UpdateDatabase(DbMigrationsConfiguration configuration)
        {
            var scaffolder = new MigrationScaffolder(configuration);
            // Creates an empty migration, so that the future migrations will start from the current
state of your database.
            var scaffoldedMigration = scaffolder.Scaffold("IgnoreChanges", ignoreChanges: true);

            // enable-migrations
            // add-migration Initial -IgnoreChanges
            configuration.MigrationsAssembly = new
MigrationCompiler(ProgrammingLanguage.CSharp.ToString())
                                                .Compile(configuration.MigrationsNamespace,
scaffoldedMigration);

            // update-database
            var dbMigrator = new DbMigrator(configuration);
            dbMigrator.Update();
        }
    }
}
```

توضیحات

MigrationScaffolder کار تولید خودکار کلاس‌های cs مهاجرت‌های EF را انجام می‌دهد. زمانیکه به متد Scaffold آن پارامتر ignoreChanges: true ارسال شود، کلاس مهاجرتی را ایجاد می‌کند که خالی است (متدهای up و down آن خالی تشکیل می‌شوند). سپس این کلاس‌ها کامپایل شده و در حین اجرای برنامه مورد استفاده قرار می‌گیرند.

برای استفاده از آن، نیاز به کلاس MigrationCompiler خواهید داشت. این کلاس در مجموعه آزمون‌های واحد EF به عنوان یک

کلاس کمکی وجود دارد: MigrationCompiler.cs
صرفا جهت تکمیل بحث و همچنین سهولت ارجاعات آتی، کدهای آن در ذیل نیز ذکر خواهد شد:

```
using System;
using System.CodeDom.Compiler;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data.Common;
using System.Data.Entity;
using System.Data.Entity.Migrations;
using System.Data.Entity.Migrations.Design;
using System.Data.Entity.Spatial;
using System.IO;
using System.Linq;
using System.Linq.Expressions;
using System.Reflection;
using System.Resources;
using System.Text;

namespace EF_General.Models.Ex22
{
    public enum ProgrammingLanguage
    {
        CSharp,
        VB
    }

    public class MigrationCompiler
    {
        private readonly CodeDomProvider _codeProvider;

        public MigrationCompiler(string language)
        {
            _codeProvider = CodeDomProvider.CreateProvider(language);
        }

        public Assembly Compile(string @namespace, params ScaffoldedMigration[] scaffoldedMigrations)
        {
            var options = new CompilerParameters
            {
                GenerateExecutable = false,
                GenerateInMemory = true
            };

            options.ReferencedAssemblies.Add(typeof(string).Assembly.Location);
            options.ReferencedAssemblies.Add(typeof(Expression).Assembly.Location);
            options.ReferencedAssemblies.Add(typeof(DbMigrator).Assembly.Location);
            options.ReferencedAssemblies.Add(typeof(DbContext).Assembly.Location);
            options.ReferencedAssemblies.Add(typeof(DbConnection).Assembly.Location);
            options.ReferencedAssemblies.Add(typeof(Component).Assembly.Location);
            options.ReferencedAssemblies.Add(typeof(MigrationCompiler).Assembly.Location);
            options.ReferencedAssemblies.Add(typeof(DbGeography).Assembly.Location);

            var embededResources = GenerateEmbeddedResources(scaffoldedMigrations, @namespace);
            foreach (var resource in embededResources)
                options.EmbeddedResources.Add(resource);

            var sources = scaffoldedMigrations.SelectMany(g => new[] { g.UserCode, g.DesignerCode });

            var compilerResults = _codeProvider.CompileAssemblyFromSource(options, sources.ToArray());
            foreach (var resource in embededResources)
                File.Delete(resource);

            if (compilerResults.Errors.Count > 0)
            {
                throw new InvalidOperationException(BuildCompileErrorMessage(compilerResults.Errors));
            }

            return compilerResults.CompiledAssembly;
        }

        private static string BuildCompileErrorMessage(CompilerErrorCollection errors)
        {
            var stringBuilder = new StringBuilder();

            foreach (CompilerError error in errors)
            {
                stringBuilder.AppendLine(error.ToString());
            }
```

```
            return stringBuilder.ToString();
        }

        private static IEnumerable<string> GenerateEmbeddedResources(IEnumerable<ScaffoldedMigration>
scaffoldedMigrations, string @namespace)
        {
            foreach (var scaffoldedMigration in scaffoldedMigrations)
            {
                var className = GetClassName(scaffoldedMigration.MigrationId);
                var embededResource = Path.Combine(
                    Path.GetTempPath(),
                    @namespace + "." + className + ".resources");

                using (var writer = new ResourceWriter(embededResource))
                {
                    foreach (var resource in scaffoldedMigration.Resources)
                        writer.AddResource(resource.Key, resource.Value);
                }

                yield return embededResource;
            }
        }

        private static string GetClassName(string migrationId)
        {
            return migrationId
                .Split(new[] { '_' }, 2)
                .Last()
                .Replace(" ", string.Empty);
        }
    }
}
```

جهت مطالعه توضیحات بیشتری در مورد CodeDom می‌توان به مطلب « کامپایل پویای کد در دات نت » مراجعه کرد.
استفاده از این کلاس‌ها نیز بسیار ساده است. یکبار دستور ذیل را در ابتدای کار برنامه فراخوانی کنید تا جدول MigrationHistory دوباره ساخته شود:

```
CreateMigrationHistory.UpdateDatabase(new Configuration());
```

با این فرض که کلاس Configuration شما چنین شکلی را دارد:

```
public class Configuration : DbMigrationsConfiguration<MyContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }
    }
```

نظرات خوانندگان

نویسنده: reza110
تاریخ: ۱۳۹۲/۰۸/۱۴ ۱۶:۵۸

با تشکر از اینکه به این مطلب مستقلا پرداختید
فقط قسمتهای زیر شناخته نمی‌شد آیا به اسمبلی خاصی نیاز دارد؟

```
using System.Data.Entity.Spatial;
 options.ReferencedAssemblies.Add(typeof(DbGeography).Assembly.Location);
scaffoldedMigration.Resources
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۱۴ ۱۷:۱۹

DbGeography از EF **5** به بعد اضافه شده. اگر پروژه 4 EF دارید، راحت قابل ارتقاء به نگارش 6 هست. البته EF نگارش 5 مخصوص دات نت 4 این قابلیت رو نداشت (فقط نگارش مخصوص دات نت 4.5 آن شامل این پیشرفت‌ها می‌شد). اما EF 6 مخصوص دات نت 4 هم این فضاهای نام رو داره و این محدودیت‌ها برطرف شده.

نویسنده: reza110
تاریخ: ۱۳۹۲/۰۸/۱۹ ۱۴:۴۹

در خط

```
var scaffolder = new MigrationScaffolder(configuration);
```

با خطای زیر مواجه شد:

The Entity Framework provider type 'System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089' for the 'System.Data.SqlClient' ADO.NET provider could not be loaded. Make sure the provider assembly is available to the running application. See http://go.microsoft.com/fwlink/?LinkId=260882 for more information.

ضمنا آیا اسمبلی مربوط به EF6 را نمی‌توان بجز نیوگت از روش دیگری بدست آورد.؟
همچنین EF Power Tools مربوط به EF6 را از کجا می‌توان تهیه کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۱۹ ۱۵:۱۴

- مطلب جاری فقط برای حالتی است که جدول migration حذف شده یا وجود ندارد.
+ مطلب « ارتقاء به Entity framework 6 و استفاده از بانک‌های اطلاعاتی غیر از SQL Server » را باید دقیق مطالعه کنید. یک سری اسمبلی باید حذف شوند. تعدادی اضافه شوند. فایل کانفیگ حتما باید ویرایش شود و تعریف پروایدر را داشته باشد. این کارها را نیوگت به صورت خودکار انجام می‌دهد. ضمنا اینکار باید برای تمام زیر پروژه‌های شما نیز تکرار شود و طوری نباشد که دو کتابخانه از 4 استفاده کنند، دوتای دیگر از 5 و اصلی هم از 6. همه باید یک دست شوند و اسمبلی منسوخ شده قدیمی نیز حذف.
- اسمبلی EF به تنهایی کافی نیست ولی از اینجا به صورت جداگانه قابل دریافت است. باید دقت داشت که ارتقاء به نگارش 6 سه مرحله‌ای است که عنوان شد.
- از اینجا

نویسنده: reza110

تاریخ: ۱۳۹۲/۰۸/۲۱ ۱۰:۳۸

با تشکر فراوان از زحمات شما
با دانلود پکیجی که اشاره کردید و افزودن اسمبلی‌های مربوطه (Entityframework .sqlserver, Entityframework6,) و اصلاح آدرس فضای نام در قسمتهای مختلف برنامه قابلیت ایجاد جدول مهاجرت ایجاد گردید.
سوالی که داشتم این بود که در سایت نیوگت پکیج هایی که وجود دارد را چگونه می‌توان مستقیما بدون استفاده از vs دریافت کرد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۲۱ ۱۰:۵۰

اینجا توضیح دادم