عنوان: **بالا بردن سرعت بارگذاری اولیه EF Code first با تعداد مدل‌های زیاد**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۰۹ ۱۹:۸
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

EF Code first هربار در حین آغاز اجرای برنامه و اولین کوئری که به بانک اطلاعاتی ارسال می‌کند، کار تشخیص روابط بین کلاس‌ها و همچنین نگاشت آن‌ها را به بانک اطلاعاتی، انجام می‌دهد. این مورد شاید با تعداد کم کلاس‌ها آنچنان به نظر نرسد، اما اگر تعداد کلاس‌های شما به بالای 200 عدد رسید، زمان آغاز برنامه آزار دهنده خواهد شد. راه حلی برای این مساله وجود دارد به نام ایجاد Viewهای متناظر با نگاشت‌ها و سپس کامپایل آن به عنوان جزئی از برنامه، که در ادامه نحوه انجام این‌کار را مرور خواهیم کرد.

بررسی ساختار pre-generated views

برای کامپایل نگاشت‌های EF در خود برنامه (بجای تولید پویای هربار آن‌ها)، ابتدا باید فایل edmx متناظر با مدل‌ها و روابط بین آن‌ها تشکیل شود:

```
var ms = new MemoryStream();
    using (var writer = XmlWriter.Create(ms))
    {
        EdmxWriter.WriteEdmx(new Context(), writer);
    }
```

پس از اینکه edmx تشکیل شد، باید از ساختار فشرده آن سه جزء زیر را استخراج کرد:

الف) storageModels : ssdl

ب) conceptualModels : csdl

ج) mappings : msl

اینکار را به صورت زیر می‌توان انجام داد:

```
var xDoc = XDocument.Load(ms);

    var ssdl = xDoc.Descendants("{http://schemas.microsoft.com/ado/2009/02/edm/ssdl}Schema").Single();
    var csdl = xDoc.Descendants("{http://schemas.microsoft.com/ado/2008/09/edm}Schema").Single();
    var msl =
xDoc.Descendants("{http://schemas.microsoft.com/ado/2008/09/mapping/cs}Mapping").Single();
```

پس از آن باید محتوای این سه جزء را توسط متد Save هر کدام، در فایل‌های xml ایی ذخیره کرد و توسط ابزاری به نام EdmGen.exe که جزئی از ویژوال استودیو است، فایل Context.Views.cs را تولید، به برنامه اضافه و سپس کامپایل کرد:

```
EdmGen.exe /mode:ViewGeneration /incsdl:Context.csdl  /inmsl:Context.msl /inssdl:Context.ssdl
/outviews:Context.Views.cs
```

بهتر است این پروسه هر بار که قرار است ارائه نهایی برنامه صورت گیرد، انجام شود.

علاوه بر این‌ها اگر علاقمند باشید که کار فایل EdmGen را شبیه سازی کنید، کلاس زیر این‌کار را انجام داده و قادر است خروجی vb یا cs متناظری را نیز تولید کند:

```
using System;
using System.Data.Entity;
using System.Data.Entity.Design;
using System.Data.Entity.Infrastructure;
using System.Data.Mapping;
using System.Data.Metadata.Edm;
using System.IO;
using System.Linq;
```

```
using System.Xml;
using System.Xml.Linq;

namespace EfUtils
{
    public static class PreGeneratedViewsWriter
    {
        public static void CreatePreGeneratedViewsFile(
                this DbContext contextInstance,
                LanguageOption language = LanguageOption.GenerateCSharpCode,
                string viewsFile = "Context.Views.cs",
                string edmxFile = "context.edmx",
                string ssdlFile = "context.ssdl.xml",
                string csdlFile = "context.csdl.xml",
                string mslFile = "context.msl.xml")
        {
            using (var contextViewsMemoryStream = new MemoryStream())
            {
                using (var edmxMemoryStream = new MemoryStream())
                {
                    var edmx = createEdmx(contextInstance, edmxFile, edmxMemoryStream);
                    var mappingItemCollection = createMappingItemCollection(ssdlFile, csdlFile,
mslFile, edmx);
                    generateViews(language, viewsFile, contextViewsMemoryStream,
mappingItemCollection);
                }
            }
        }

        private static void generateViews(LanguageOption language, string viewsFile, MemoryStream
contextViewsMemoryStream, StorageMappingItemCollection mappingItemCollection)
        {
            var viewGenerator = new EntityViewGenerator // It's defined in
System.Data.Entity.Design.dll
            {
                LanguageOption = language
            };
            using (var streamWriter = new StreamWriter(contextViewsMemoryStream))
            {
                var errors = viewGenerator.GenerateViews(mappingItemCollection, streamWriter).ToList();

                if (errors.Any())
                    throw new InvalidOperationException(errors.First().Message);

                contextViewsMemoryStream.Position = 0;
                using (var reader = new StreamReader(contextViewsMemoryStream))
                {
                    var codeData = reader.ReadToEnd();
                    File.WriteAllText(viewsFile, codeData);
                }
            }
        }

        private static StorageMappingItemCollection createMappingItemCollection(string ssdlFile, string
csdlFile, string mslFile, XDocument edmx)
        {
            var ssdl =
edmx.Descendants("{http://schemas.microsoft.com/ado/2009/02/edm/ssdl}Schema").Single();
            ssdl.Save(ssdlFile);
            var storeItemCollection = new StoreItemCollection(new[] { ssdl.CreateReader() });

            var csdl =
edmx.Descendants("{http://schemas.microsoft.com/ado/2008/09/edm}Schema").Single();
            csdl.Save(csdlFile);
            var edmItemCollection = new EdmItemCollection(new[] { csdl.CreateReader() });

            var msl =
edmx.Descendants("{http://schemas.microsoft.com/ado/2008/09/mapping/cs}Mapping").Single();
            msl.Save(mslFile);

            var mappingItemCollection = new StorageMappingItemCollection(edmItemCollection,
storeItemCollection, new[] { msl.CreateReader() });
            return mappingItemCollection;
        }

        private static XDocument createEdmx(DbContext contextInstance, string edmxFile, MemoryStream
edmxMemoryStream)
        {
            var settings = new XmlWriterSettings { Indent = true };
            using (var writer = XmlWriter.Create(edmxMemoryStream, settings))
            {
```

```
                EdmxWriter.WriteEdmx(contextInstance, writer);
            }
            File.WriteAllBytes(edmxFile, edmxMemoryStream.ToArray());

            edmxMemoryStream.Position = 0;
            var edmx = XDocument.Load(edmxMemoryStream);
            return edmx;
        }
    }
}
```

در اینجا همان مراحلی که عنوان شد، تکرار می‌شود. فایل edmx متناظر با وهله‌ای از DbContext برنامه، تولید شده و سه جزء آن استخراج می‌شوند. سپس این موارد به EntityViewGenerator موجود در اسمبلی System.Data.Entity.Design.dll ارسال شده و کد نهایی متناظر قابل کامپایل در برنامه تولید می‌گردد.
پس از تولید فایل Context.Views.cs یا Context.Views.vb، آن‌را به پروژه اضافه کنید.
اینبار نحوه استفاده از آن باید به صورت زیر باشد:

```
Database.SetInitializer<MyContext>(null);
```

از این جهت که تمام اطلاعات لازم جهت آغاز کار، در فایل تولیدی Context.Views وجود دارد و اکنون جزئی از فایل اجرایی برنامه است و نیازی به تکرار ساخت مجدد پویای آن نیست.

مرجع:
Entity Framework Code First View Generation Templates On Visual Studio Code Gallery

نظرات خوانندگان

نویسنده: مهدی پایروند
تاریخ: ۱۳۹۱/۰۷/۰۹ ۲۰:۳۸

در یکی از پروژه هایی که بجای تعداد 200 جدول رابطه زیادی هم داشت مجبور به استفاده از این روش شدم، البته سبک Model First

نویسنده: Petek
تاریخ: ۱۳۹۱/۰۷/۱۱ ۱۱:۴۶

با سلام
آیا امکان این هست که بشه از این در روش DataBase First استفاده کرد . با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۱۱ ۱۲:۰

بله. همین روال رو می‌تونید توسط افزونه « Entity Framework Power Tools » هم انجام بدید (گزینه Generate Views را به منوی کلیک راست بر روی Entity Data Model/*.EDMX اضافه می‌کند).

نویسنده: Petek
تاریخ: ۱۳۹۱/۰۷/۱۱ ۱۷:۵۰

با سلام
ممنوم از راهنماییتون خیلی عالیه .
با استفاده از این کار یه view در کنار Model ام ایجاد شد و سرعت بارگذاری برای این اولین بار رو از 10 ثانیه به کمتر از 2 ثانیه تقلیل داد آیا چیز دیگه ای نیاز نیست و نیز من برای ایجاد دیتابیسم با استفاده از Model در اولین واکشی به این صورت عمل می‌کنم آیا مشکلی پیش نمیاد در استفاده از این روش . با تشکر

```
if (!db_Context.DatabaseExists())
      db_Context.CreateDatabase();
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۱۱ ۱۸:۵۵

- خیر. به تنظیم دیگری نیاز ندارد. این کلاس‌ها به صورت خودکار تشخیص داده شده و استفاده می‌شوند. البته به ازای هربار تغییر مدل‌ها نیاز است مجددا تولید شوند.
- اگر روش شما db first است که عنوان کردید، بررسی فوق (ایجاد بانک اطلاعاتی) کار اضافی است. اگر روش code first است، باز هم نیازی نیست چون در حالت خودکار migrations اینکار را انجام می‌دهد.
در کل بهتر است تمام جوانب را بررسی و آزمایش کنید.
کاری که در اینجا انجام می‌شود ایجاد یک cached metadata کامپایل شده است بجای تولید پویای هربار آن (تفاوت مهم و اصلی با روش‌های متداول).

نویسنده: امیر
تاریخ: ۱۳۹۱/۱۱/۲۸ ۰:۵۲

سلام و خسته نباشید
سوالی که من دارم اینکه بعد از ساختن فایل Context.Views.cs چطور باید ازش استفاده کرد . من این فایلو ساختم و کنار Context.cs تو پروژهام گذاشتم . ولی فرق خاصی احساس نمی‌کنم . میشه بیشتر راهنمایی کنید

نویسنده: محسن
تاریخ: ۱۳۹۱/۱۱/۲۸ ۰:۵۸

اگر فرقی احساس نکردید منتظر نگارش بعدی EF باشید. این مورد رو برطرف کردن:

«significantly improved warm up time (view generation), especially for large models»

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۷/۲۷ ۱:۶

به روز شده این اطلاعات برای EF 6
اطلاعات بیشتر

نویسنده: سوین
تاریخ: ۱۳۹۲/۰۹/۰۴ ۱۱:۵۰

آیا بعد از ایجاد View برای بالا بردن لود اولیه ، باز هم migration کار میکنه یا نه ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۰۴ ۱۲:۱۱

بله. هم روش دستی migrations و هم روش خودکار در اینجا کار می‌کنند. اینکه در انتهای بحث عنوان شده مقدار را null قرار بدید، برای رسیدن به حداکثر سرعت بارگذاری برنامه است. در این حالت می‌توانید از روش دستی برای انجام migrations استفاده کنید. این نکته در انتهای مطلب پنجم سری EF سایت بحث شده.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۲۴ ۱۰:۵۵

اگر علاقمند باشید که مدیریت تولید این Viewها را خودکار کنید می‌توانید از پروژه Interactive Pre Generated Views استفاده نمائید:

پروژه: Interactive Pre Generated Views for Entity Framework 6
بسته نیوگت
نحوه استفاده: Using Pre-Generated Views Without Having To Pre-Generate Views

نویسنده: امین
تاریخ: ۱۳۹۲/۱۱/۲۲ ۱۸:۱۵

باسلام.این کد را در کجا و چطوری در مدل code first استفاده بکنیم؟

```
using (var ctx = new MyContext())
{
    InteractiveViews
        .SetViewCacheFactory(
            ctx,
            new FileViewCacheFactory(@"C:\MyViews.xml"));
}
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۲ ۱۸:۲۶

در آغاز برنامه (مثلا در application_start برنامه‌های وب و یا ابتدای متد Main برنامه‌های دسکتاپ): « وادار کردن EF Code

first به ساخت بانک اطلاعاتی پیش از شروع به کار برنامه »