عنوان: **استفاده از چندین Context در EF 6 Code first**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۲۳ ۱۴:۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework

در نگارش قبلی EF Code first به ازای یک پروژه تنها یک سیستم Migration قابل تعریف بود و این سیستم مهاجرت، تنها با یک DbContext کار می‌کرد. در نگارش ششم این کتابخانه، سیستم مهاجرت Code first آن از چندین DbContext، به ازای یک پروژه که به یک بانک اطلاعاتی اشاره می‌کنند، پشتیبانی می‌کند. مزیت اینکار اندکی بهبود در نگهداری تنها کلاس DbContext تعریف شده است. برای مثال می‌توان یک کلاس DbContext مخصوص قسمت ثبت نام را ایجاد کرد. یک کلاس DbContext مخصوص کلیه جداول مرتبط با مقالات را و همینطور الی آخر. نهایتا تمام این Contextها سبب ایجاد یک بانک اطلاعاتی واحد خواهند شد.
اگر در یک پروژه EF Code first چندین Context وجود داشته باشد و دستور enable-migrations را بدون پارامتری فراخوانی کنیم، پیغام خطای More than one context type was found in the assmbly xyz را دریافت خواهیم کرد.
الف) اما در EF 6 می‌توان با بکار بردن سوئیچ جدید ContextTypeName، به ازای هر Context، مهاجرت مرتبط با آن‌را تنظیم نمود:

```
enable-migrations -ContextTypeName dbContextName1 -MigrationDirectory DataContexts\Name1Migrations
```

همچنین در اینجا نیز می‌توان با استفاده از سوئیچ MigrationDirectory، فایل‌های تولید شده را در پوشه‌های مجزایی ذخیره کرد.

ب) در مرحله بعد، نیاز به فراخوانی دستور add-migration است:

```
add-migration -ConfigurationTypeName FullNameSpaceCtx1.Configuration "InitialCreate"
```

با اجرای دستور enable-migrations یک کلاس Configuration جهت DbContext مشخص شده، ایجاد می‌شود. سپس آدرس کامل این کلاس را به همراه ذکر دقیق فضای نام آن در اختیار دستور add-migration قرار می‌دهیم.
ذکر کامل فضای نام، از این جهت مهم است که کلاس Configuration به ازای Contextهای مختلف ایجاد شده، یک نام را خواهد داشت؛ اما در فضاهای نام متفاوتی قرار می‌گیرد.
با اجرای دستور add-migration، کدهای سی شارپ مورد نیاز جهت اعمال تغییرات بر روی ساختار بانک اطلاعاتی تولید می‌شوند. در مرحله بعد، این کدها تبدیل به دستورات SQL متناظری شده و بر روی بانک اطلاعاتی اجرا خواهند شد.
بدیهی است اگر دو Context در برنامه تعریف کرده باشید، دوبار باید دستور enable-migrations و دوبار دستور add-migration را با پارامترهای اشاره کننده به Conetxtهای مدنظر اجرا کرد.

ج) سپس برای اعمال این تغییرات، باید دستور update-database را اجرا کرد.

```
update-database  -ConfigurationTypeName FullNameSpaceCtx1.Configuration
```

اینبار دستور update-database نیز بر اساس نام کامل کلاس Configuration مدنظر باید اجرا گردد و به ازای هر Context موجود، یکبار نیاز است اجرا گردد.
نهایتا اگر به بانک اطلاعاتی مراجعه کنید، تمام جداول و تعاریف را یکجا در همان بانک اطلاعاتی می‌توانید مشاهده نمائید.

داشتن چندین Context در برنامه و مدیریت تراکنش‌ها

در EF، هر DbContext معرف یک واحد کار است. یعنی تراکنش‌ها و چندین عمل متوالی مرتبط انجام شده، درون یک DbContext معنا پیدا می‌کنند. متد SaveChanges نیز بر همین اساس است که کلیه اعمال ردیابی شده در طی یک واحد کار را در طی یک تراکنش به بانک اطلاعاتی اعمال می‌کند. همچنین مباحثی مانند lazy loading نیز در طی یک Context مفهوم دارند. به علاوه دیگر امکان join نویسی بین دو Context وجود نخواهد داشت. باید اطلاعات را از یکی واکشی و سپس این اطلاعات درون حافظه‌ای را به دیگری ارسال کنید.

یک نکته

می‌توان یک DbSet را در چندین Context تعریف کرد. یعنی اگر بحث join نویسی مطرح است، با تکرار تعریف DbSetها اینکار قابل انجام است اما این مساله اساس جداسازی Contextها را نیز زیر سؤال می‌برد.

داشتن چندین Context در برنامه و مدیریت رشته‌های اتصالی

در EF Code first روش‌های مختلفی برای تعریف رشته اتصالی به بانک اطلاعاتی وجود دارند. اگر تغییر خاصی در کلاس مشتق شده از DbContext ایجاد نکنیم، نام کلید رشته اتصالی تعریف شده در فایل کانفیگ باید به نام کامل کلاس Context برنامه اشاره کند. اما با داشتن چندین Context به ازای یک دیتابیس می‌توان از روش ذیل استفاده کرد:

```
public class Ctx1 : DbContext
    {
        public Ctx1()
            : base("DefaultConnection")
        {
            //Database.Log = sql => Debug.Write(sql);
        }
    }

  public class Ctx2 : DbContext
    {
        public Ctx2()
            : base("DefaultConnection")
        {
            //Database.Log = sql => Debug.Write(sql);
        }
    }
```

```
<connectionStrings>
    <add name="DefaultConnection" connectionString="…." providerName="System.Data.SqlClient" />
  </connectionStrings>
```

در اینجا در سازنده کلاس‌های Context تعریف شده، نام کلید رشته اتصالی موجود در فایل کانفیگ برنامه ذکر شده است. به این ترتیب این دو Context به یک بانک اطلاعاتی اشاره خواهند کرد.

چه زمانی بهتر است از چندین Context در برنامه استفاده کرد؟

عموما در طراحی‌های سازمانی SQL Server، تمام جداول از schema مدیریتی به نام dbo استفاده نمی‌کنند. جداول فروش از schema خاص خود و جداول کاربران از schema دیگری استفاده خواهند کرد. با استفاده از چندین Context می‌توان به ازای هر کدام از schemaهای متفاوت موجود، «یک ناحیه ایزوله» را ایجاد و مدیریت کرد.

```
public class Ctx2 : DbContext
    {
        public Ctx2()
            : base("DefaultConnection")
        {
            //Database.Log = sql => Debug.Write(sql);
        }

         protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.HasDefaultSchema("sales");
            base.OnModelCreating(modelBuilder);
        }
    }
```

در این حالت در EF 6 می‌توان DefaultSchema کلی یک Context را در متد بازنویسی شده OnModelCreating به نحو فوق تعریف و مدیریت کرد. در این مثال به صورت خودکار کلیه DbSetهای Ctx2 از schema ایی به نام sales استفاده می‌کنند.

نظرات خوانندگان

نویسنده: مهدی سعیدی فر
تاریخ: ۱۳۹۲/۰۹/۰۳ ۲۲:۴۹

می خواستم ببینم می‌توان از این امکان برای پیاده سازی دیتابیس‌های توزیع شده استفاده کرد؟
مثلا جدول کاربران در یک پایگاه داده‌ی مستقل و جدول دیدگاه‌های کاربران در یک پایگاه داده مستقل دیگر باشد و به ازای هر کدام یک Context جداگانه تعریف کرد و در برنامه با آن‌ها تعامل کرد به گونه ای که به نظر آید با یک پایگاه داده سر و کار داریم.
آیا اگر از این شیوه‌ی طراحی استفاده شود دیگر مسائل رابطه‌ی بین جداول منتفی است؟ و اگر بله شبیه سازی رابطه‌ها باید به این صورت پیاده سازی شود که اطلاعات جداول به صورت جداگانه از دیتابیس‌ها خوانده شود و سپس با استفاده از Linq To Object رابطه‌ی بین آن‌ها برقرار شود؟
با این شیوه‌ی طراحی تراکنش‌ها چگونه پیاده سازی می‌شود؟ آیا هر Context دارای یک تراکنش جداگانه است و یا امکان پیاده سازی آن به صورت یک تراکنش هم وجود دارد؟ الگوی Unit Of Work را باید به ازای هر Context جداگانه تعریف کرد؟
البته من اطلاعات خیلی ناقصی از پایگاه‌های داده توزیع شده دارم و ممکنه حرفام کاملا اشتباه باشد. متاسفانه من جایی را پیدا نکردم که در مورد پیاده سازی عملی آن بحث کرده باشد و بیشتر با یک سری مفاهیم تئوری برخورد کردم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۰۴ ۱:۲۳

- در SQL Server قابلیتی به نام Linked servers وجود دارد که توسط آن در یک شبکه داخلی می‌شود چندین بانک اطلاعاتی SQL Server را در نودهای مختلف شبکه به هم متصل کرد و با آن‌ها یکپارچه و مانند یک دیتابیس واحد کار کرد. این مورد در برنامه‌های سازمانی خیلی مرسوم است. قرار نیست به ازای هر برنامه‌ای که برای یک سازمان نوشته می‌شود، هر کدام جداگانه بانک اطلاعاتی کاربران و لاگین خاص خودشان را داشته باشند. عموما یک دیتابیس مرکزی مثلا مدیریت منابع انسانی وجود دارد که مابقی برنامه‌ها را تغذیه می‌کند. حتی می‌شود به یک بانک اطلاعاتی MySQL هم متصل شد (^).
- ORMها صرفا کارهایی را قادر به انجام هستند که بانک اطلاعاتی مورد استفاده پشتیبانی می‌کند. برای نمونه در SQL Server امکان تهیه رابطه بین دو جدول از دو بانک اطلاعاتی مختلف وجود ندارد . منظور مثلا ایجاد کلید خارجی بین جداول دو بانک اطلاعاتی مختلف است و نه نوشتن کوئری بین آن‌ها که از این لحاظ مشکلی نیست.
A FOREIGN KEY constraint can reference columns in tables in the same database or within the same table
- هدف از پشتیبانی چند Context در EF 6، تلفیق این‌ها با هم در قالب یک بانک اطلاعاتی است (مطلب فوق).
- همچنین در EF 6 می‌توان چندین Context را به چندین بانک اطلاعاتی مختلف با رشته‌های اتصالی متفاوت، انتساب داد. مباحث آغاز بانک‌های اطلاعاتی هر کدام جداگانه عمل کرده و مستقل از هم عمل می‌کنند.
یک مثال جهت اجرا و آزمایش:
Sample25.cs
- اگر می‌خواهید به انعطاف پذیری Linked servers برسید (مثلا در طی یک کوئری و نه چند کوئری از جداول دو بانک اطلاعاتی مختلف در دو سرور متفاوت کوئری بگیرید)، نیاز خواهید داشت مثلا View یا SP تهیه کنید و سپس این‌ها را در برنامه مورد استفاده قرار دهید. Viewها با استفاده از T-SQL می‌توانند کوئری واحدی از چند بانک اطلاعاتی تهیه کنند. اینبار برنامه هم نهایتا به یک بانک اطلاعاتی متصل می‌شود و برای آن اهمیتی ندارد که View یا SP به چه نحوی تهیه شده و با چند بانک اطلاعاتی کار می‌کند.
- تراکنش‌های توزیع شده هم نیاز به تنظیمات خاصی در SQL Server دارند و باید MSDTC راه اندازی و تنظیم شود: (^). در غیراینصورت پیام خطای The underlying provider failed on Open. MSDTC on server is unavailable را دریافت خواهید کرد.
بعد از آن باید از TransactionScope برای کار همزمان با چند Context استفاده کنید.

نویسنده: محمد دبیرسیاقی
تاریخ: ۱۳۹۲/۰۹/۱۳ ۲۱:۱۷

من از دو context در برنامم برای schemaهای مختلف استفاده میکنم مشکلی که دارم اینه که entity یک schema وابستگی به entity شمای دیگر دارد و من entityهای هر schema را در یک assembly قرار دادم .الان نمیدانم چطور روابط را برقرار کنم مثلا در یک schema جدول کاربران را دارم که در یک Assembly است و در شمای دیگر جدول مقالات را در assembly دیگری دارم

.ارتباط این دو به این صورت است که وقتی مقاله ای درج می‌شود باید کاربری که مقاله را درج کرده نیز در دیتابیس درج شود.
الان باید یک پروپرتی از جنس مقاله در کلاس کاربر و یک پروپرتی از جنس کاربر در کلاس مقاله در نظر بگیرم.با وجود اینکه هر کدام در یک Assembly هستند باید Refrence از یک اسمبلی در دیگری داشته باشم که این موضوع امکان پذیر نیست لطفا راهنمایی نمائید ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۱۳ ۲۳:۵۴

- عموما circular reference بین اسمبلی‌ها نشانه‌ی طراحی بد است.
- استفاده از چند Context برای اینکه هر کدام قرار است در یک دیتابیس جدا ذخیره شوند؟ نمی‌شود FK بین این‌ها (جداول دو دیتابیس مختلف) تعریف کرد (SQL Server چنین کاری را پشتیبانی نمی‌کند).
- اگر برنامه ماژولار است، در EF می‌توان به صورت خودکار تمام ماژول‌ها را در طی یک Context یکپارچه بارگذاری کرد (^ و ^).
- هدف از ایجاد Schema در SQL Server، ایجاد ظروفی برای گروه بندی منطقی اشیاء است. مثلا عده‌ای به سه SP خاص دسترسی داشته باشند. عده‌ای فقط بتوانند با Viewها کار کنند. یا حتی عده‌ای به تمام موارد دسترسی داشته باشند. بنابراین یک نوع ایزوله سازی قسمت‌های مختلف بانک اطلاعاتی مدنظر هست، در اصل. حالا اگر عده‌ای فقط به سه جدول خاص دسترسی دارند، آیا می‌توانند ارجاعی را به جدول چهارمی که در یک schema دیگر تعریف شده داشته باشند؟ بله. البته فقط به این شرط که کاربران schema سه جدول فعلی به schema جدول چهارم، دسترسی و مجوز لازم را داشته باشد و برای این دسترسی دادن‌ها هم باید مستقلا T-SQL بنویسید.
و ... ضمنا گاهی از اوقات از Schema برای مدیریت نام‌های هم نام استفاده می‌شود. چیزی شبیه به namespace در سی‌شارپ مثلا. نمونه‌اش طراحی چند مستاجری است.
نتیجه گیری؟ برای سرگرمی Schema ایجاد نکنید؛ مگر اینکه واقعا قصد ایزوله سازی قسمت‌های مختلف یک بانک اطلاعاتی را از کاربرانی خاص داشته باشید. به Schema به شکل یک Sandbox امنیتی (یک قرنطینه) نگاه کنید.

برای مطالعه بیشتر
Understanding the Difference between Owners and Schemas in SQL Server
Implementation of Database Object Schemas

نویسنده: میثم فغفوری
تاریخ: ۱۳۹۲/۱۱/۰۷ ۰:۱۹

خسته نباشید. سناریوی بنده این است که میخوام سایتی طراحی کنم که کاملا ماژولار باشد یعنی هر بخش رو به صورت user controller طراحی و در هسته اصلی لود کنم و هر کدوم از این کنترلرها جداول مخصوص به خودشون رو دارن توی بانک اطلاعاتی که خوب طبیعتا کلاس POCO و DBContext مخصوص هر ماژول باید توی سورس کد خود ماژول نوشته بشه. با فرض اینکه بعد از کامپایل پروژه دیگه دسترسی به migration نداریم میتونین بنده رو راهنمایی کنین که چطور میتونم با این روشی که فرمودید جداول جداگانه هر کنترلر یا ماژول رو به بانک اضافه کنم بدون دسترسی به migration ؟ خودم هر راهی به ذهنم رسید انجام دادم ولی همچنان ارور تغییر در کلاس‌های POCO را میدهد سایت.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۰۷ ۰:۴۷

از یک Context مرکزی استفاده کنید که موجودیت‌ها را به صورت خودکار خوانده و اضافه می‌کند . همچنین فعال سازی گزینه‌های مهاجرت خودکار را هم مطالعه کنید.

نویسنده: میثم فغفوری
تاریخ: ۱۳۹۲/۱۱/۰۷ ۱۵:۳

ممنون مشکلم حل شد فقط برای عملیات Seed برای جداول هر ماژول هم راهی هست؟ با استفاده از همون Reflection.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۰۷ ۱۶:۱۵

ایده کار، این بود که قسمتی را مثلا توسط یک کلاس پایه، یا یک اینترفیس خالی علامتگذاری کنید و سپس اطلاعات آن‌را تک تک به متد Entity مربوط به DbModelBuilder ارسال کنید. همین ایده را در متد Seed هم می‌شود پیاده سازی کرد. یک اینترفیس خالی را مثلا به نام IMySeed تعریف کنید و به کلاس دلخواهی انتساب دهید (یا از MEF استفاده کنید). سپس اینطرف با Reflection این نوع کلاس‌ها را بارگذاری کرده و Context متد Seed را به طراحی انجام شده، برای عملیات نهایی و دلخواه ارسال کنید.

نویسنده: محمد شهریاری
تاریخ: ۱۳۹۳/۰۳/۰۳ ۱۹:۲۵

با سلام
من از dbfirst 5 EF به صورت Context های جداگانه در پروژه‌های وب جدا استفاده کردم و در نهایت تمامی این assembly ها را در یک وب سایت publish می‌کنم . در صورتی که از یک Entity به صورت مشترک در 2 context استفاده کرده باشم با خطای زیر

```
System.Data.MetadataException: Schema specified is not valid. Errors:
 Multiple types with the name &amp;#39;Customer&amp;#39; exist in the EdmItemCollection in different
namespaces .  Convention based mapping requires unique names without regard to namespace in the
EdmItemCollection
```

مواجه میشم . با اینکه Assembly های مربوط به Contextها متفاوت هست اما با این خطا روبرو میشم . آیا قابلیت گفته شده در EF 6 این مشکل برطرف شده است ؟ و یا در ef 5 راهکاری برای این مشکل وجود ندارد ؟
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۳/۰۳ ۲۰:۱۴

جستجوی عین خطا در گوگل، پاسخ اول .

نویسنده: محمد رعیت پیشه
تاریخ: ۱۳۹۳/۰۳/۲۶ ۱۲:۷

در خط

```
enable-migrations -ContextTypeName dbContextName1 -MigrationDirectory DataContexts\Name1Migrations
```

فکر میکنم MigrationDirectory باید به MigrationsDirectory تغییر کند.