عنوان: **سازگار سازی EFTracingProvider با EF Code first**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۱ ۱۴:۲۵
آدرس: www.dotnettips.info
برچسب‌ها: Entity framework

برای ثبت SQL تولیدی توسط EF، ابزارهای پروفایلر زیادی وجود دارند (+). علاوه بر این‌ها یک پروایدر سورس باز نیز برای این منظور به نام EFTracingProvider موجود می‌باشد که برای EF Database first نوشته شده است. در ادامه نحوه‌ی استفاده از این پروایدر را در برنامه‌های EF Code first مرور خواهیم کرد.

الف) دریافت کدهای EFTracingProvider اصلی: (+)
از کدهای دریافتی این مجموعه، فقط به دو پوشه EFTracingProvider و EFProviderWrapperToolkit آن نیاز است.

ب) اصلاح کوچکی در کدهای این پروایدر جهت بررسی نال بودن شیءایی که باید dispose شود
در فایل DbConnectionWrapper.cs، متد Dispose را یافته و به نحو زیر اصلاح کنید (بررسی نال نبودن wrappedConnection اضافه شده است):

```
protected override void Dispose(bool disposing)
        {
            if (disposing)
            {
                if (this.wrappedConnection != null)
                    this.wrappedConnection.Dispose();
            }

            base.Dispose(disposing);
        }
```

ج) ساخت یک کلاس پایه Context با قابلیت لاگ فرامین SQL صادره، جهت میسر سازی استفاده مجدد از کدهای آن
د) رفع خطای The given key was not present in the dictionary در حین استفاده از EFTracingProvider

در ادامه کدهای کامل این دو قسمت به همراه یک مثال کاربردی را ملاحظه می‌کنید:

```
using System;
using System.Configuration;
using System.Data;
using System.Data.Common;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Data.Entity.Migrations;
using System.Diagnostics;
using System.Linq;
using EFTracingProvider;

namespace Sample
{
    public class Person
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }

    public class Configuration : DbMigrationsConfiguration<MyContext>
    {
        public Configuration()
        {
            var className = this.ContextType.FullName;
            var connectionStringData = ConfigurationManager.ConnectionStrings[className];
            if (connectionStringData == null)
                throw new InvalidOperationException(string.Format("ConnectionStrings[{0}] not found.",
className));

            TargetDatabase = new DbConnectionInfo(connectionStringData.ConnectionString,
```

```
connectionStringData.ProviderName);
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }

        protected override void Seed(MyContext context)
        {
            for (int i = 0; i < 7; i++)
                context.Users.Add(new Person { Name = "name " + i });

            base.Seed(context);
        }
    }

    public class MyContext : MyLoggedContext
    {
        public DbSet<Person> Users { get; set; }
    }

    public abstract class MyLoggedContext : DbContext
    {
        protected MyLoggedContext()
            : base(existingConnection: createConnection(), contextOwnsConnection: true)
        {
            var ctx = ((IObjectContextAdapter)this).ObjectContext;
            ctx.GetTracingConnection().CommandExecuting += (s, e) =>
            {
                Console.WriteLine("{0}\n", e.ToTraceString());
            };
        }

        private static DbConnection createConnection()
        {
            var st = new StackTrace();
            var sf = st.GetFrame(2); // Get the derived class Type in a base class static method
            var className = sf.GetMethod().DeclaringType.FullName;

            var connectionStringData = ConfigurationManager.ConnectionStrings[className];
            if (connectionStringData == null)
                throw new InvalidOperationException(string.Format("ConnectionStrings[{0}] not found.",
className));

            if (!isEFTracingProviderRegistered())
                EFTracingProviderConfiguration.RegisterProvider();

            EFTracingProviderConfiguration.LogToFile = "log.sql";
            var wrapperConnectionString =
                string.Format(@"wrappedProvider={0};{1}", connectionStringData.ProviderName,
connectionStringData.ConnectionString);
            return new EFTracingConnection { ConnectionString = wrapperConnectionString };
        }

        private static bool isEFTracingProviderRegistered()
        {
            var data = (DataSet)ConfigurationManager.GetSection("system.data");
            var providerFactories = data.Tables["DbProviderFactories"];
            return providerFactories.Rows.Cast<DataRow>()
                                         .Select(row => (string)row.ItemArray[1])
                                         .Any(invariantName => invariantName == "EF Tracing Data
Provider");
        }
    }

    public static class Test
    {
        public static void RunTests()
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<MyContext, Configuration>());
            using (var ctx = new MyContext())
            {
                var users = ctx.Users.AsEnumerable();
                if (users.Any())
                {
                    foreach (var user in users)
                    {
                        Console.WriteLine(user.Name);
                    }
                }

                var rnd = new Random();
                var user1 = ctx.Users.Find(1);
```

```
                user1.Name = "test user " + rnd.Next();
                ctx.SaveChanges();
            }

        }
    }
}
```

توضیحات:

تعریف TargetDatabase در Configuration سبب می‌شود تا خطای The given key was not present in the dictionary در حین استفاده از این پروایدر جدید برطرف شود. به علاوه همانطور که ملاحظه می‌کنید اطلاعات رشته اتصالی بر اساس قراردادهای توکار EF Code first به نام کلاس Context تنظیم شده است.

کلاس MyLoggedContext، کلاس پایه‌ای است که تنظیمات اصلی «EF Tracing Data Provider» در آن قرار گرفته‌اند. برای استفاده از آن باید رشته اتصالی مخصوصی تولید و در اختیار کلاس پایه DbContext قرار گیرد (توسط متد createConnection ذکر شده).

به علاوه در اینجا توسط خاصیت EFTracingProviderConfiguration.LogToFile می‌توان نام فایلی را که قرار است عبارات SQL تولیدی در آن درج شوند، ذکر نمود. همچنین یک روش دیگر دستیابی به کلیه عبارات SQL تولیدی را با مقدار دهی CommandExecuting در سازنده کلاس مشاهده می‌کنید.

اکنون که این کلاس پایه تهیه شده است، تنها کافی است Context معمولی برنامه به نحو زیر تعریف شود:

```
public class MyContext : MyLoggedContext
```

در ادامه اگر متد RunTests را اجرا کنیم، خروجی ذیل را می‌توان در کنسول مشاهده کرد:

```
insert [dbo].[People]([Name])
values (@0)
select [Id]
from [dbo].[People]
where @@ROWCOUNT > 0 and [Id] = scope_identity()
-- @0 (dbtype=String, size=-1, direction=Input) = "name 0"

insert [dbo].[People]([Name])
values (@0)
select [Id]
from [dbo].[People]
where @@ROWCOUNT > 0 and [Id] = scope_identity()
-- @0 (dbtype=String, size=-1, direction=Input) = "name 1"

insert [dbo].[People]([Name])
values (@0)
select [Id]
from [dbo].[People]
where @@ROWCOUNT > 0 and [Id] = scope_identity()
-- @0 (dbtype=String, size=-1, direction=Input) = "name 2"

insert [dbo].[People]([Name])
values (@0)
select [Id]
from [dbo].[People]
where @@ROWCOUNT > 0 and [Id] = scope_identity()
-- @0 (dbtype=String, size=-1, direction=Input) = "name 3"

insert [dbo].[People]([Name])
values (@0)
select [Id]
from [dbo].[People]
where @@ROWCOUNT > 0 and [Id] = scope_identity()
-- @0 (dbtype=String, size=-1, direction=Input) = "name 4"

insert [dbo].[People]([Name])
values (@0)
select [Id]
from [dbo].[People]
where @@ROWCOUNT > 0 and [Id] = scope_identity()
-- @0 (dbtype=String, size=-1, direction=Input) = "name 5"

insert [dbo].[People]([Name])
values (@0)
```

```
select [Id]
from [dbo].[People]
where @@ROWCOUNT > 0 and [Id] = scope_identity()
-- @0 (dbtype=String, size=-1, direction=Input) = "name 6"

SELECT
[Extent1].[Id] AS [Id],
[Extent1].[Name] AS [Name]
FROM [dbo].[People] AS [Extent1]

SELECT
[Extent1].[Id] AS [Id],
[Extent1].[Name] AS [Name]
FROM [dbo].[People] AS [Extent1]

name 0
name 1
name 2
name 3
name 4
name 5
name 6

update [dbo].[People]
set [Name] = @0
where ([Id] = @1)
-- @0 (dbtype=String, size=-1, direction=Input) = "test user 1355460609"

-- @1 (dbtype=Int32, size=0, direction=Input) = 1
```

قسمتی از این خروجی مرتبط است به متد Seed تعریف شده که تعدادی رکورد را در بانک اطلاعاتی ثبت می‌کند.
دو select نیز در انتهای کار قابل مشاهده است. اولین مورد به علت فراخوانی متد Any صادر شده است و دیگری به حلقه foreach مرتبط می‌باشد (چون از AsEnumerable استفاده شده، هربار ارجاع به شیء users، یک رفت و برگشت به بانک اطلاعاتی را سبب خواهد شد. برای رفع این حالت می‌توان از متد ToList استفاده کرد.)
در پایان کار، متد update مربوط است به فراخوانی متدهای find و save changes ذکر شده. این خروجی در فایل sql.log نیز در کنار فایل اجرایی برنامه ثبت شده و قابل مشاهده می‌باشد.

کاربردها

اطلاعات این مثال می‌تواند پایه نوشتن یک برنامه entity framework profiler باشد.

نظرات خوانندگان

نویسنده: مهدی پایروند
تاریخ: ۱۳۹۱/۰۵/۲۱ ۲۲:۲۷

یه سوال برای راه انداختن یه همچین برنامه ای بنظرتون باید یک listener رو روی پورتی ست کرد یا از طریق دیگه ای مثل reflection باید انجام بشه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۱ ۲۲:۴۹

WCF می‌تونه انتخاب خوبی باشه یا استفاده از named pipes

نویسنده: صالح باقری
تاریخ: ۱۳۹۱/۰۵/۲۲ ۴:۲

مدتهاست که EF رو پیگیری میکنم ولی هنوز در مورد پروژه خودم به نتیجه ای نرسیدم.

پروژه ای که من دارم دیتابیس آن کاملاً ساخته شده و بر اساس دیتابیس، کلاسهای مرتبط با آن (BLL) نیز نوشته شده است. ولی کلاس‌های DAL رو ننوشتم تا اینکه با EF آشنا شدم.

میخواستم ببینم که اگه از DB First استفاده کنم چطور میتونم از کلاسهای نوشته شده قبل استفاده کنم؟ یا چطور میتونم جداول رو با کلاسهایی که خودم نوشتم مرتبط کنم؟ کلا اینکار پیشنهاد میشه یا خیر؟

اگه از CodeFirst استفاده کنم، تکلیف Db طراحی شده خودم چی میشه؟ چون دیتابیسی که EF ایجاد میکنه رو اصلاً نمیپسندم.

در کل آیا EF روشی برای ارتباط Code First و Db First داره؟

آیا با اینهمه کدهای پیچیده ای که EF ایجاد میکنه میشه ازش در پروژه‌های وب بزرگ که ترافیک سنگینی دارند استفاده کرد؟

در کل نظر خود شما چی هست؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۲ ۸:۲۶

- موضوع بحث جاری مطلب دیگری است.
- ابزار برای تبدیل جداول دیتابیسی موجود به کلاس‌های code first وجود دارد .
- پیشنیاز درک کدهای تولید شده آن مطالعه 15 قسمتی است که در سایت وجود دارد.