

یکی از روش‌های تهیه‌ی برنامه‌های چند مستاجری ، ایجاد بانک‌های اطلاعاتی مستقلی به ازای هر مشتری است؛ یا نمونه‌ی دیگر آن، برنامه‌هایی هستند که اطلاعات هر سال را در یک بانک اطلاعاتی جداگانه نگهداری می‌کنند. در ادامه قصد داریم، نحوه‌ی کار با این بانک‌های اطلاعاتی را به صورت همزمان، توسط EF Code first و در حالت استفاده از الگوی واحد کار و تزریق وابستگی‌ها، به همراه فعال سازی خودکار مباحث migrations و به روز رسانی ساختار تمام بانک‌های اطلاعاتی مورد استفاده، بررسی کنیم.

مشخص سازی رشته‌های متفاوت اتصالی

فرض کنید برنامه‌ی جاری شما قرار است از دو بانک اطلاعاتی مشخص استفاده کند که تعاریف رشته‌های اتصالی آن‌ها در وب کانفیگ به صورت ذیل مشخص شده‌اند:

```
<connectionStrings>
  <clear />
  <add name="Sample07Context" connectionString="Data Source=(local);Initial
Catalog=TestDbIoC;Integrated Security = true" providerName="System.Data.SqlClient" />
  <add name="Database2012" connectionString="Data Source=(local);Initial
Catalog=testdb2012;Integrated Security = true" providerName="System.Data.SqlClient" />
</connectionStrings>
```

البته، ذکر این مورد کاملاً اختیاری است و می‌توان رشته‌های اتصالی را به صورت پویا نیز در زمان اجرا مشخص و مقدار دهی کرد.

تغییر Context برنامه جهت پذیرش رشته‌های اتصالی پویای قابل تغییر در زمان اجرا

اکنون که قرار است کاربران در حین ورود به برنامه، بانک اطلاعاتی مدنظر خود را انتخاب کنند و یا سیستم قرار است به ازای کاربری خاص، رشته‌ی اتصالی خاص او را به Context ارسال کند، نیاز است Context برنامه را به صورت ذیل تغییر دهیم:

```
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using EF_Sample07.DomainClasses;

namespace EF_Sample07.DataLayer.Context
{
    public class Sample07Context : DbContext, IUnitOfWork
    {
        public DbSet<Category> Categories { set; get; }
        public DbSet<Product> Products { set; get; }

        /// <summary>
        /// It looks for a connection string named Sample07Context in the web.config file.
        /// </summary>
        public Sample07Context()
            : base("Sample07Context")
        {
        }

        /// <summary>
        /// To change the connection string at runtime. See the SmObjectFactory class for more info.
        /// </summary>
        public Sample07Context(string connectionString)
            : base(connectionString)
        {
            //Note: defaultConnectionFactory in the web.config file should be set.
        }

        public void SetConnectionString(string connectionString)
        {
            this.Database.Connection.ConnectionString = connectionString;
        }
    }
}
```

}

در اینجا دو متد سازنده را مشاهده می‌کنید. سازنده‌ی پیش فرض، از رشته‌ای اتصال با نامی مساوی Sample07Context استفاده می‌کند و سازنده‌ی دوم، امکان پذیرش یک رشته‌ی اتصال پویا را دارد. مقدار پارامتر ورودی آن می‌تواند نام رشته‌ی اتصال و یا حتی مقدار کامل رشته‌ی اتصال باشد. حالت پذیرش نام رشته‌ی اتصال زمانی مفید است که همانند مثال ابتدای بحث، این نام‌ها را پیشتر در فایل کانفیگ برنامه ثبت کرده باشید و حالت پذیرش مقدار کامل رشته‌ی اتصال، جهت مقدار دهی پویای آن بدون نیاز به ثبت اطلاعاتی در فایل کانفیگ برنامه مفید است.

یک متد دیگر هم در اینجا در انتهای کلاس به نام SetConnectionString تعریف شده‌است. از این متد در حین ورود کاربر به سایت می‌توان استفاده کرد. برای مثال حداقل دو نوع طراحی را می‌توان در نظر گرفت:

الف) کاربر با برنامه‌ای کار می‌کند که به ازای سال‌های مختلف، بانک‌های اطلاعاتی مختلفی دارد و در ابتدای ورود، یک drop down انتخاب سال کاری برای او در نظر گرفته شده‌است (علاوه بر سایر ورودی‌های استاندارد مانند نام کاربری و کلمه‌ی عبور). در این حالت بهتر است متد SetConnectionString نام رشته‌ی اتصال را بر اساس سال انتخابی، در حین لاگین دریافت کند که اطلاعات آن در فایل کانفیگ سایت پیشتر مشخص شده‌است.

ب) کاربر یا مشتری پس از ورود به سایت، نیاز است صرفاً از بانک اطلاعاتی خاص خودش استفاده کند. بنابراین اطلاعات تعریف کاربران و مشتری‌ها در یک بانک اطلاعاتی مجزا قرار دارند و پس از لاگین، نیاز است رشته‌ی اتصال او به صورت پویا از بانک اطلاعاتی خوانده شده و سپس توسط متد SetConnectionString تنظیم گردد.

مدیریت سشن‌های رشته‌ی اتصال جاری

پس از اینکه کاربر، در حین ورود مشخص کرد که از چه بانک اطلاعاتی قرار است استفاده کند و یا اینکه برنامه بر اساس اطلاعات ثبت شده‌ی او تصمیم‌گیری کرد که باید از کدام رشته‌ی اتصال استفاده کند، نگهداری این رشته‌ی اتصال نیاز به سشن دارد تا به ازای هر کاربر متصل به سایت منحصر بفرد باشد. در مورد مدیریت سشن‌ها در برنامه‌های وب، از نکات مطرح شده‌ی در مطلب «[مدیریت سشن‌ها در برنامه‌های وب به کمک تزریق وابستگی‌ها](#)» استفاده خواهیم کرد:

```
using System;
using System.Threading;
using System.Web;
using EF_Sample07.DataLayer.Context;
using EF_Sample07.ServiceLayer;
using StructureMap;
using StructureMap.Web;
using StructureMap.Web.Pipeline;

namespace EF_Sample07.IocConfig
{
    public static class SmObjectFactory
    {
        private static readonly Lazy<Container> _containerBuilder =
            new Lazy<Container>(defaultContainer, LazyThreadSafetyMode.ExecutionAndPublication);

        public static IContainer Container
        {
            get { return _containerBuilder.Value; }
        }

        public static void HttpContextDisposeAndClearAll()
        {
            HttpContextLifecycle.DisposeAndClearAll();
        }

        private static Container defaultContainer()
        {
            return new Container(ioc =>
            {
                // session manager setup
                ioc.For<ISessionProvider>().Use<DefaultWebSessionProvider>();
                ioc.For<HttpSessionStateBase>()
                    .Use(ctx => new HttpSessionStateWrapper(HttpContext.Current.Session));

                ioc.For<IUnitOfWork>()
                    .HybridHttpOrThreadLocalScoped()
            });
        }
    }
}
```

```

        .Use<Sample07Context>()
        // Remove these 2 lines if you want to use a connection string named
        Sample07Context, defined in the web.config file.
        .Ctor<string>("connectionString")
        .Is(ctx => getCurrentConnectionString(ctx));

        ioc.For<ICategoryService>().Use<EfCategoryService>();
        ioc.For<IProductService>().Use<EfProductService>();

        ioc.For<ICategoryService>().Use<EfCategoryService>();
        ioc.For<IProductService>().Use<EfProductService>();

        ioc.Policies.SetAllProperties(properties =>
        {
            properties.OfType<IUnitOfWork>();
            properties.OfType<ICategoryService>();
            properties.OfType<IProductService>();
            properties.OfType<ISessionProvider>();
        });
    });
}

private static string getCurrentConnectionString(IContext ctx)
{
    if (HttpContext.Current != null)
    {
        // this is a web application
        var sessionProvider = ctx.GetInstance<ISessionProvider>();
        var connectionString = sessionProvider.Get<string>("CurrentConnectionString");
        if (string.IsNullOrEmpty(connectionString))
        {
            // It's a default connectionString.
            connectionString = "Database2012";
            // this session value should be set during the login phase
            sessionProvider.Store("CurrentConnectionStringName", connectionString);
        }

        return connectionString;
    }
    else
    {
        // this is a desktop application, so you can store this value in a global static
        variable.
        return "Database2012";
    }
}
}
}

```

در اینجا نحوه‌ی پویا سازی تامین رشته‌ی اتصالی را مشاهده می‌کنید. در مورد اینترفیس `ISessionProvider` و کلاس پایه `HttpSessionStateBase` پیشتر در مطلب «[مدیریت سشن‌ها در برنامه‌های وب به کمک تزریق وابستگی‌ها](#)» بحث شد. نکته‌ی مهم این تنظیمات، قسمت مقدار دهی سازنده‌ی کلاس `Context` برنامه به صورت پویا توسط `IoC Container` جاری است. در اینجا هر زمانیکه قرار است وهله‌ای از `Sample07Context` ساخته شود، از سازنده‌ی دوم آن که دارای پارامتری به نام `connectionString` است، استفاده خواهد شد. همچنین مقدار آن به صورت پویا از متد `getCurrentConnectionString` که در انتهای کلاس تعریف شده‌است، دریافت می‌گردد.

در این متد ابتدا مقدار `HttpContext.Current` بررسی شده‌است. این مقدار اگر نال باشد، یعنی برنامه‌ی جاری یک برنامه‌ی دسکتاپ است و مدیریت رشته‌ی اتصالی جاری آن را توسط یک خاصیت `Static` یا `Singleton` تعریف شده‌ی در برنامه نیز می‌توان تامین کرد. از این جهت که در هر زمان، تنها یک کاربر در `App Domain` جاری برنامه‌ی دسکتاپ می‌تواند وجود داشته باشد و `Singleton` یا `Static` تعریف شدن اطلاعات رشته‌ی اتصالی، مشکلی را ایجاد نمی‌کند. اما در برنامه‌های وب، چندین کاربر در یک `App Domain` به سیستم وارد می‌شوند. به همین جهت است که مشاهده می‌کنید در اینجا از تامین کننده‌ی سشن، برای نگهداری اطلاعات رشته‌ی اتصالی جاری کمک گرفته شده‌است.

کلید این سشن نیز در این مثال مساوی `CurrentConnectionStringName` تعریف شده‌است. بنابراین در حین لاگین موفقیت آمیز کاربر، دو مرحله‌ی زیر باید طی شوند:

```

sessionProvider.Store("CurrentConnectionString", "Sample07Context");
uow.SetConnectionString(WebConfigurationManager.ConnectionStrings[_sessionProvider.Get<string>("Current

```

```
ConnectionString").ConnectionString);
```

ابتدا باید سشن `CurrentConnectionStringName` به بانک اطلاعاتی انتخابی کاربر تنظیم شود. برای نمونه در این مثال خاص، از نام رشته‌ی اتصال مشخص شده‌ی در وب کانفیگ برنامه (مثال ابتدای بحث) به نام `Sample07Context` استفاده شده‌است. سپس از متد `SetConnectionString` برای خواندن مقدار نام مشخص شده در سشن `CurrentConnectionStringName` کمک گرفته‌ایم. هرچند سازنده‌ی کلاس `Context` برنامه، هر دو حالت استفاده از نام رشته‌ی اتصال و یا مقدار کامل رشته‌ی اتصال را پشتیبانی می‌کند، اما خاصیت `this.Database.Connection.ConnectionString` تنها رشته‌ی کامل اتصال را می‌پذیرد (بکار رفته در متد `SetConnectionString`).

تا اینجا کار پویا سازی انتخاب و استفاده از رشته‌ی اتصال برنامه به پایان می‌رسد. هر زمانیکه قرار است `Context` برنامه توسط `IoC Container` نمونه سازی شود، به متد `getCurrentConnectionString` رجوع کرده و مقدار رشته‌ی اتصال را از سشن تنظیم شده‌ای به نام `CurrentConnectionStringName` دریافت می‌کند. سپس از مقدار آن جهت مقدار دهی سازنده‌ی دوم کلاس `Context` استفاده خواهد کرد.

مدیریت migrations خودکار برنامه در حالت استفاده از چندین بانک اطلاعاتی

یکی از مشکلات کار با برنامه‌های چند دیتابرسی، به روز رسانی ساختار تمام بانک‌های اطلاعاتی مورد استفاده، پس از تغییری در ساختار مدل‌های برنامه است. از این جهت که اگر تمام بانک‌های اطلاعاتی به روز نشوند، کوئری‌های جدید برنامه که از خواص و فیلدهای جدید استفاده می‌کنند، دیگر کار نخواهند کرد. پویا سازی اعمال این تغییرات را می‌توان به صورت ذیل انجام داد:

```
using System;
using System.Data.Entity;
using System.Web;
using EF_Sample07.DataLayer.Context;
using EF_Sample07.IoCConfig;

namespace EF_Sample07.WebFormsAppSample
{
    public class Global : HttpApplication
    {
        void Application_Start(object sender, EventArgs e)
        {
            initDatabases();
        }

        private static void initDatabases()
        {
            // defined in web.config
            string[] connectionStringNames =
            {
                "Sample07Context",
                "Database2012"
            };

            foreach (var connectionStringName in connectionStringNames)
            {
                Database.SetInitializer(
                    new MigrateDatabaseToLatestVersion<Sample07Context,
                    Configuration>(connectionStringName));

                using (var ctx = new Sample07Context(connectionStringName))
                {
                    ctx.Database.Initialize(force: true);
                }
            }
        }

        void Application_EndRequest(object sender, EventArgs e)
        {
            SmObjectFactory.HttpContextDisposeAndClearAll();
        }
    }
}
```

نکته‌ی مهمی که در اینجا بکار گرفته شده است، مشخص سازی صریح سازنده‌ی شیء [MigrateDatabaseToLatestVersion](#) است. به صورت معمول در اکثر برنامه‌های تک دیتابرسی، نیازی به مشخص سازی پارامتر سازنده‌ی این کلاس نیست و در این حالت از سازنده‌ی بدون پارامتر کلاس Context برنامه استفاده خواهد شد. اما اگر سازنده‌ی آن را مشخص کنیم، به صورت خودکار از متد سازنده‌ای در کلاس Context استفاده می‌کند که پارامتر رشته‌ی اتصال را به صورت پویا می‌پذیرد. در این مثال خاص، متد `initDatabases` در حین آغاز برنامه فراخوانی شده است. منظور این است که اینکار در طول عمر برنامه تنها کافی است یکبار انجام شود و پس از آن است که EF Code first می‌تواند از رشته‌های اتصال متفاوتی که به آن ارسال می‌شود، بدون مشکل استفاده کند. زیرا اطلاعات نگاشت کلاس‌های مدل برنامه به جداول بانک اطلاعاتی به این ترتیب است که کش می‌شوند و یا بر اساس کلاس Configuration به صورت خودکار به بانک اطلاعاتی اعمال می‌گردند.

کدهای کامل این مثال را که در حقیقت نمونه‌ی بهبود یافته‌ی مطلب « [EF Code First #12](#) » است، از اینجا می‌توانید دریافت کنید:

[Uow-Sample](#)

نظرات خوانندگان

نویسنده: اس حیدری

تاریخ: ۱۳۹۳/۱۲/۱۱ ۱۲:۳۰

سلام

متد SetConnectionString که در واقع کانکشن استرینگ را ست می‌کند و تنها زمانی مفید خواهد بود که کانکشن استرینگ‌ها به یک نوع Rdbms متصل شوند. مثلاً همه کانکشن استرینگ‌ها به کانکشن‌های مختلفی اشاره کند که همگی به دیتابیس‌های مختلف sql server متصل شوند.

اما اگر یک کانکشن استرینگ به sql server و یک کانکشن استرینگ به دیتابیس‌سی از نوع sql ce یا نوع‌های دیگه متصل شود با خطای provider مواجه خواهیم شد. چرا که کانکشن استرینگ، کانکشن ست شده و Provider آن از کانکشن پیش فرض مقدار گرفته است. و عملاً هم نمی‌توان provider را در این تابع مقدار داد چرا که Readonly است!

نویسنده: وحید نصیری

تاریخ: ۱۳۹۳/۱۲/۱۱ ۱۲:۴۸

امکان ساخت رشته‌ی اتصالی، به همراه ذکر صریح Provider مورد استفاده [هم وجود دارد](#). چند مثال:

```
public static string CreateConnectionStringForSQLCe(string dbPath = @"|DataDirectory|\NerdDinners.sdf")
{
    SqlCeConnectionStringBuilder sqlConnection = new SqlCeConnectionStringBuilder();
    sqlConnection.Password = "9023fase93";
    sqlConnection.DataSource = dbPath;

    EntityConnectionStringBuilder connection = new EntityConnectionStringBuilder();
    connection.Metadata =
@"res://*/NerdDinnersModel.csdl|res://*/NerdDinnersModel.ssdl|res://*/NerdDinnersModel.msl";
    connection.Provider = "System.Data.SqlServerCe.3.5";
    connection.ProviderConnectionString = sqlConnection.ToString();

    return connection.ToString();
}

public static string CreateConnectionStringForSQLServer()
{
    //Build an SQL connection string
    SqlConnectionStringBuilder sqlString = new SqlConnectionStringBuilder()
    {
        DataSource = "MyPC", // Server name
        InitialCatalog = "db1", //Database
        UserID = "user1", //Username
        Password = "mypassword", //Password
    };

    //Build an Entity Framework connection string
    EntityConnectionStringBuilder entityString = new EntityConnectionStringBuilder
    {
        Provider = "System.Data.SqlClient",
        Metadata = "res://*/testModel.csdl|res://*/testModel.ssdl|res://*/testModel.msl",
        ProviderConnectionString = sqlString.ToString()
    };
    return entityString.ConnectionString;
}
```

این روش با EF code first هم کار می‌کند و در سازنده‌ی دوم کلاس Context که connectionString را می‌پذیرد، قابل استفاده‌است.

نویسنده: غلامرضا ربال

تاریخ: ۲۳:۴۴ ۱۳۹۴/۰۲/۱۰

سلام.

در این مقاله با استفاده از سشن ، از پردازش‌های موازی چشم پوشی شده است ؟ با توجه به [مطلبی](#) که در سایت ارائه داده بودید .آیا بهتر نبود این رشته در کنار اطلاعات کاربر در دیتابیس به عنوانی فیلدی در نظر گرفته شود؟

با تشکر

نویسنده: وحید نصیری

تاریخ: ۲۳:۵۶ ۱۳۹۴/۰۲/۱۰

- در اینجا بدون سشن رشته‌ی اتصال، مشخص نیست کاربر جاری در صفحه‌ی X قرار است از چه بانک اطلاعاتی استفاده کند تا بخواهد از آن کوئری بگیرد.

- در این مطلب با توجه به اینکه سشن، توسط اینترفیس [ISessionProvider](#) تامین می‌شود، تعویض پذیر است. یک SessionProvider سفارشی را برای مثال با کوکی‌های رمزنگاری شده یا روش‌های مشابه تهیه کنید. تزریق و استفاده‌ی از آن خودکار خواهد بود؛ بدون نیازی به تغییری در کدهای سایر قسمت‌های برنامه.