عنوان: **آشنایی با NHibernate - قسمت هفتم**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۷/۲۳ ۱۳:۴۴:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: NHibernate

مدیریت بهینه‌ی سشن فکتوری

ساخت یک شیء SessionFactory بسیار پر هزینه و زمانبر است. به همین جهت لازم است که این شیء یکبار حین آغاز برنامه ایجاد شده و سپس در پایان کار برنامه تخریب شود. انجام اینکار در برنامه‌های معمولی ویندوزی (WPF، WinForms و ...)، ساده است اما در محیط Stateless وب و برنامه‌های ASP.Net ، نیاز به راه حلی ویژه وجود خواهد داشت و تمرکز اصلی این مقاله حول مدیریت صحیح سشن فکتوری در برنامه‌های ASP.Net است.

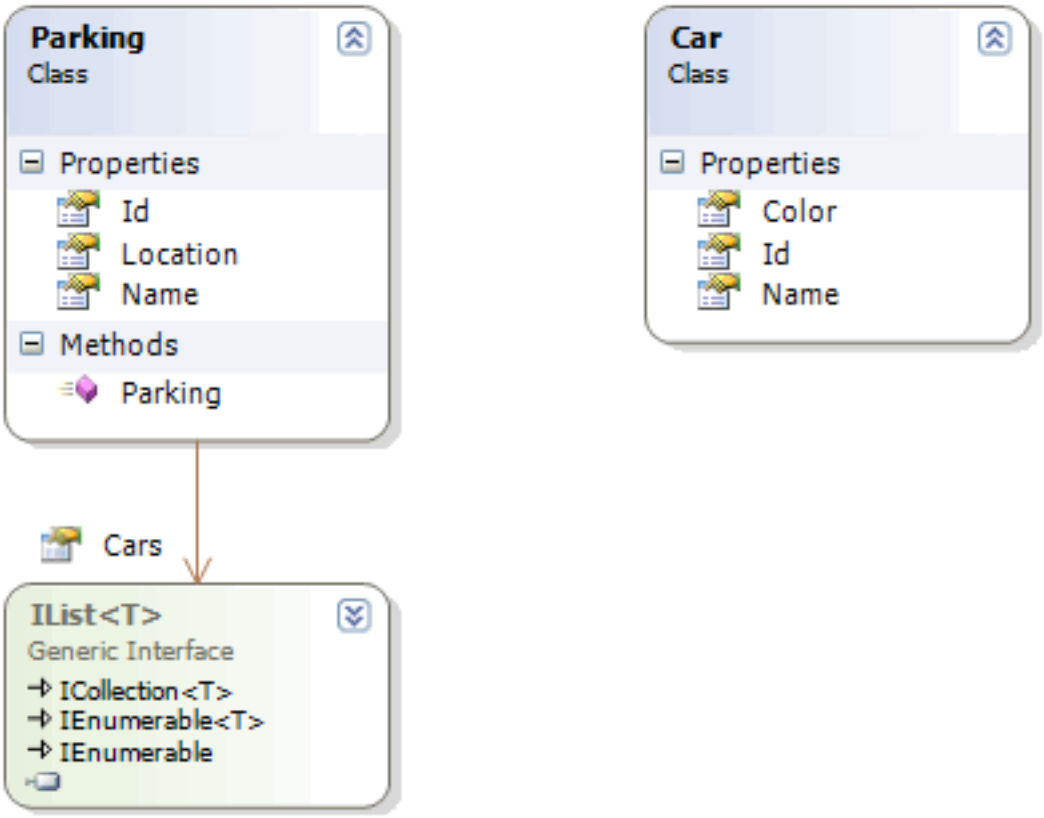
برای پیاده سازی شیء سشن فکتوری به صورتی که یکبار در طول برنامه ایجاد شود و بارها مورد استفاده قرار گیرد باید از یکی از الگوهای معروف طراحی برنامه نویسی شیء گرا به نام Singleton Pattern استفاده کرد. پیاده سازی نمونه‌ی thread safe آن که در برنامه‌های ذاتا چند ریسمانی وب و همچنین برنامه‌های معمولی ویندوزی می‌تواند مورد استفاده قرار گیرد، در آدرس ذیل قابل مشاهده است:

#Implementing the Singleton Pattern in C

از پنجمین روش ذکر شده در این مقاله جهت ایجاد یک lazy, lock-free, thread-safe singleton استفاده خواهیم کرد.

بررسی مدل برنامه

در این مدل ساده ما یک یا چند پارکینگ داریم که در هر پارکینگ یک یا چند خودرو می‌توانند پارک شوند.



یک برنامه ASP.Net را آغاز کرده و ارجاعاتی را به اسمبلی‌های زیر به آن اضافه نمائید:

FluentNHibernate.dll

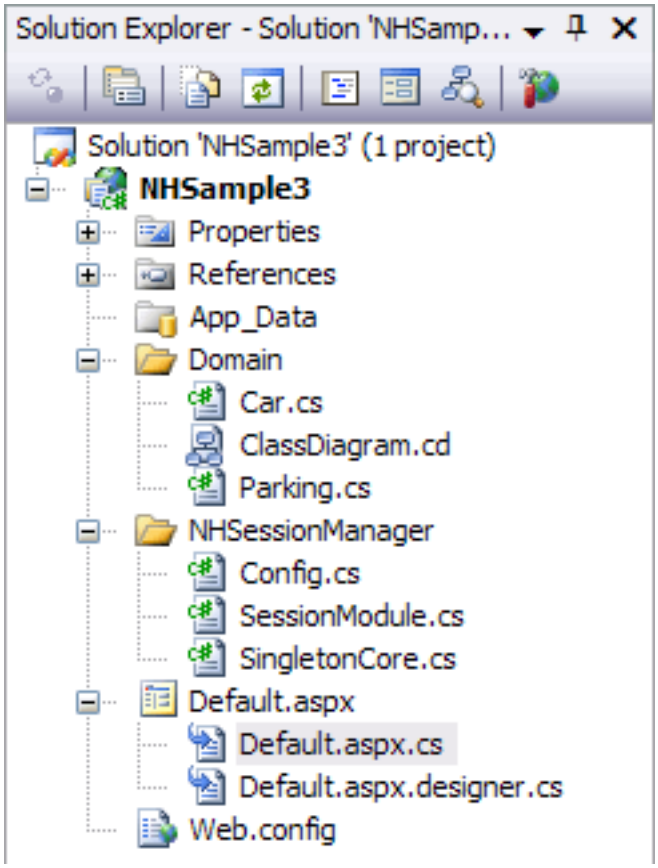
NHibernate.dll

NHibernate.ByteCode.Castle.dll

NHibernate.Linq.dll

و همچنین ارجاعی به اسمبلی استاندارد System.Data.Services.dll دات نت فریم ورک سه و نیم

تصویر نهایی پروژه ما به شکل زیر خواهد بود:



پروژه ما دارای یک پوشه domain ، تعریف کننده موجودیت‌های برنامه جهت تهیه نگاشت‌های لازم از روی آن‌ها است. سپس یک پوشه جدید را به نام NHSessionManager به آن جهت ایجاد یک Http module مدیریت کننده سشن‌های NHibernate در برنامه اضافه خواهیم کرد.

ساختار دومین برنامه (مطابق کلاس دیاگرام فوق):

```
namespace NHSample3.Domain
{
  public class Car
  {
      public virtual int Id { get; set; }
      public virtual string Name { get; set; }
      public virtual string Color { get; set; }
  }
}
```

```
using System.Collections.Generic;

namespace NHSample3.Domain
{
  public class Parking
  {
      public virtual int Id { get; set; }
      public virtual string Name { get; set; }
      public virtual string Location { get; set; }
      public virtual IList<Car> Cars { get; set; }

      public Parking()
      {
          Cars = new List<Car>();
      }
  }
}
```

مدیریت سشن فکتوری در برنامه‌های وب

در این قسمت قصد داریم Http Module ایی را جهت مدیریت سشن‌های NHibernate ایجاد نمائیم.

در ابتدا کلاس Config را در پوشه مدیریت سشن NHibernate با محتویات زیر ایجاد کنید:

```
using FluentNHibernate.Automapping;
using FluentNHibernate.Cfg;
using FluentNHibernate.Cfg.Db;
using NHibernate.Tool.hbm2ddl;

namespace NHSessionManager
{
  public class Config
  {
      public static FluentConfiguration GetConfig()
      {
          return
            Fluently.Configure()
                    .Database(
                       MsSqlConfiguration
                         .MsSql2008
                         .ConnectionString(x => x.FromConnectionStringWithKey("DbConnectionString"))
                        )
                      .ExposeConfiguration(
                          x => x.SetProperty("current_session_context_class", "managed_web")
                       )
                      .Mappings(
                        m => m.AutoMappings.Add(
                           new AutoPersistenceModel()
                               .Where(x => x.Namespace.EndsWith("Domain"))
                               .AddEntityAssembly(typeof(NHSample3.Domain.Car).Assembly))
                               );
      }

      public static void CreateDb()
      {
          bool script = false;//آیا خروجی در کنسول هم نمایش داده شود
          bool export = true;//آیا بر روی دیتابیس هم اجرا شود
          bool dropTables = false;//آیا جداول موجود دراپ شوند
          new SchemaExport(GetConfig().BuildConfiguration()).Execute(script, export, dropTables);
      }
  }
}
```

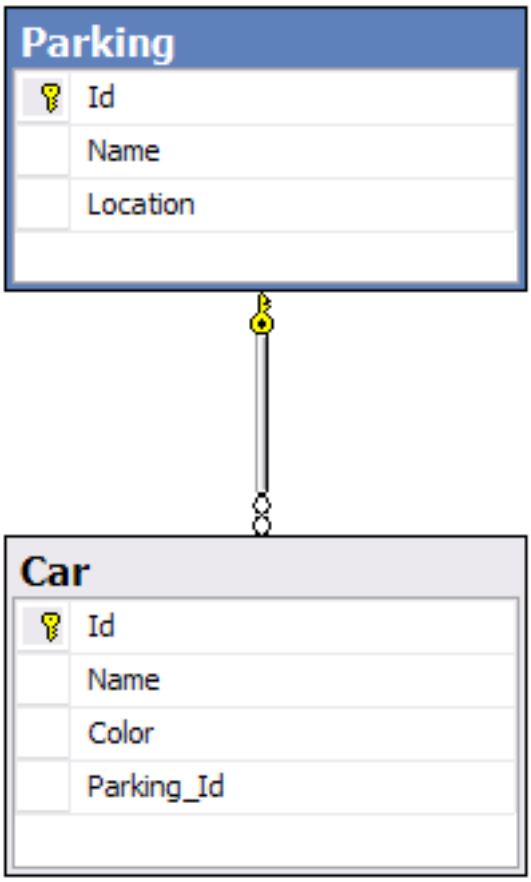
با این کلاس در قسمت‌های قبل آشنا شده‌اید. در این کلاس با کمک امکانات Auto mapping موجود در Fluent Nhibernate (مطلب قسمت قبلی این سری آموزشی) اقدام به تهیه نگاشت‌های خودکار از کلاس‌های قرار گرفته در پوشه دومین خود خواهیم کرد (فضای نام این پوشه به دومین ختم می‌شود که در متد GetConfig مشخص است).
دو نکته جدید در متد GetConfig وجود دارد:
الف) استفاده از متد FromConnectionStringWithKey ، بجای تعریف مستقیم کانکشن استرینگ در متد مذکور که روشی است توصیه شده. به این صورت فایل وب کانفیگ ما باید دارای تعریف کلید مشخص شده در متد GetConfig به نام DbConnectionString باشد:

```
<connectionStrings>
  <!--NHSessionManager-->
  <add name="DbConnectionString"
       connectionString="Data Source=(local);Initial Catalog=HelloNHibernate;Integrated Security =
true" />
</connectionStrings>
```

ب) قسمت ExposeConfiguration آن نیز جدید است.
در اینجا به AutoMapper خواهیم گفت که قصد داریم از امکانات مدیریت سشن مخصوص وب فریم ورک NHibernate استفاده کنیم. فریم ورک NHibernate دارای کلاسی است به نام NHibernate.Context.ManagedWebSessionContext که جهت مدیریت سشن‌های خود در پروژه‌های وب ASP.Net پیش بینی کرده است و از این متد در Http module ایی که ایجاد خواهیم کرد جهت

ردگیری سشن جاری آن کمک خواهیم گرفت.

اگر متد CreateDb را فراخوانی کنیم، جداول نگاشت شده به کلاس‌های پوشه دومین برنامه، به صورت خودکار ایجاد خواهند شد که دیتابیس دیاگرام آن به صورت زیر می‌باشد:



سپس کلاس SingletonCore را جهت تهیه تنها و تنها یک وهله از شیء سشن فکتوری در کل برنامه ایجاد خواهیم کرد (همانطور که عنوان شده، ایده پیاده سازی این کلاس thread safe ، از مقاله معرفی شده در ابتدای بحث گرفته شده است):

```
using NHibernate;

namespace NHSessionManager
{
  /// <summary>
  /// lazy, lock-free, thread-safe singleton
  /// </summary>
  public class SingletonCore
  {
      private readonly ISessionFactory _sessionFactory;

      SingletonCore()
      {
          _sessionFactory = Config.GetConfig().BuildSessionFactory();
      }

      public static SingletonCore Instance
      {
          get
          {
              return Nested.instance;
```

```
            }
        }

        public static ISession GetCurrentSession()
        {
            return Instance._sessionFactory.GetCurrentSession();
        }

        public static ISessionFactory SessionFactory
        {
            get { return Instance._sessionFactory; }
        }

        class Nested
        {
            // Explicit static constructor to tell C# compiler
            // not to mark type as beforefieldinit
            static Nested()
            {
            }

            internal static readonly SingletonCore instance = new SingletonCore();
        }
    }
}
```

اکنون می‌توان از این Singleton object جهت تهیه یک Http Module کمک گرفت. برای این منظور کلاس SessionModule را به برنامه اضافه کنید:

```
using System;
using System.Web;
using NHibernate;
using NHibernate.Context;

namespace NHSessionManager
{
  public class SessionModule : IHttpModule
  {
      public void Dispose()
      { }

      public void Init(HttpApplication context)
      {
          if (context == null)
              throw new ArgumentNullException("context");

          context.BeginRequest += Application_BeginRequest;
          context.EndRequest += Application_EndRequest;
      }

      private void Application_BeginRequest(object sender, EventArgs e)
      {
          ISession session = SingletonCore.SessionFactory.OpenSession();
          ManagedWebSessionContext.Bind(HttpContext.Current, session);
          session.BeginTransaction();
      }

      private void Application_EndRequest(object sender, EventArgs e)
      {
          ISession session = ManagedWebSessionContext.Unbind(
              HttpContext.Current, SingletonCore.SessionFactory);
          if (session == null) return;

          try
          {
              if (session.Transaction != null &&
                  !session.Transaction.WasCommitted &&
                  !session.Transaction.WasRolledBack)
              {
                  session.Transaction.Commit();
              }
              else
              {
                  session.Flush();
              }
          }
          catch (Exception)
```

```
            {
                session.Transaction.Rollback();
            }
            finally
            {
                if (session != null && session.IsOpen)
                {
                    session.Close();
                    session.Dispose();
                }
            }
        }
    }
}
```

کلاس فوق کار پیاده سازی اینترفیس IHttpModule را جهت دخالت صریح در request handling pipeline برنامه ASP.Net جاری انجام می‌دهد. در این کلاس مدیریت متدهای استاندارد Application_BeginRequest و Application_EndRequest به صورت خودکار صورت می‌گیرد.
در متد Application_BeginRequest ، در ابتدای هر درخواست یک سشن جدید ایجاد و به مدیریت سشن وب NHibernate بایند می‌شود، همچنین یک تراکنش نیز آغاز می‌گردد. سپس در پایان درخواست، این انقیاد فسخ شده و تراکنش کامل می‌شود، همچنین کار پاکسازی اشیاء نیز صورت خواهد گرفت.

با توجه به این موارد، دیگر نیازی به ذکر using جهت dispose کردن سشن جاری در کدهای ما نخواهد بود، زیرا در پایان هر درخواست اینکار به صورت خودکار صورت می‌گیرد. همچنین نیازی به ذکر تراکنش نیز نمی‌باشد، چون مدیریت آن‌را خودکار کرده‌ایم.

جهت استفاده از این Http module تهیه شده باید چند سطر زیر را به وب کانفیگ برنامه اضافه کرد:

```
<httpModules>
    <!--NHSessionManager-->
    <add name="SessionModule" type="NHSessionManager.SessionModule"/>
  </httpModules>
```

بدیهی است اگر نخواهید از Http module استفاده کنید باید این کدها را در فایل Global.asax برنامه قرار دهید.

اکنون مثالی از نحوه‌ی استفاده از امکانات فراهم شده فوق به صورت زیر می‌تواند باشد:
ابتدا کلاس ParkingContext را جهت مدیریت مطلوب‌تر LINQ to NHibernate تشکیل می‌دهیم.

```
using System.Linq;
using NHibernate;
using NHibernate.Linq;
using NHSample3.Domain;

namespace NHSample3
{
  public class ParkingContext : NHibernateContext
  {
      public ParkingContext(ISession session)
          : base(session)
      { }

      public IOrderedQueryable<Car> Cars
      {
          get { return Session.Linq<Car>(); }
      }

      public IOrderedQueryable<Parking> Parkings
      {
          get { return Session.Linq<Parking>(); }
      }
  }
}
```

سپس در فایل Default.aspx.cs برنامه ، برای نمونه تعدادی رکورد را افزوده و نتیجه را در یک گرید ویوو نمایش خواهیم داد:

```
using System;
using System.Collections.Generic;
using System.Linq;
using NHibernate;
using NHSample3.Domain;
using NHSessionManager;

namespace NHSample3
{
  public partial class _Default : System.Web.UI.Page
  {
      protected void Page_Load(object sender, EventArgs e)
      {
          //ایجاد دیتابیس در صورت نیاز
          //Config.CreateDb();

          //ثبت یک سری رکورد در دیتابیس
          ISession session = SingletonCore.GetCurrentSession();

          Car car1 = new Car() { Name = "رنو", Color = "مشکلی" };
          session.Save(car1);
          Car car2 = new Car() { Name = "پژو", Color = "سفید" };
          session.Save(car2);

          Parking parking1 = new Parking()
          {
              Location = "آدرس پارکینگ مورد نظر",
              Name = "پارکینگ یک",
              Cars = new List<Car> { car1, car2 }
          };

          session.Save(parking1);

          //نمایش حاصل در یک گرید ویوو
          ParkingContext db = new ParkingContext(session);
          var query = from x in db.Cars select new { CarName = x.Name, CarColor = x.Color };
          GridView1.DataSource = query.ToList();
          GridView1.DataBind();
      }
  }
}
```

مدیریت سشن فکتوری در برنامه‌های غیر وب

در برنامه‌های ویندوزی مانند WPF ، WinForms و غیره، تا زمانیکه یک فرم باز باشد، کل فرم و اشیاء مرتبط با آن به یکباره تخریب نخواهند شد، اما در یک برنامه ASP.Net جهت حفظ منابع سرور در یک محیط چند کاربره، پس از پایان نمایش یک صفحه وب، اثری از آثار اشیاء تعریف شده در کدهای آن صفحه در سرور وجود نداشته و همگی بلافاصله تخریب می‌شوند. به همین جهت بحث‌های ویژه state management در ASP.Net در اینباره مطرح است و مدیریت ویژه‌ای باید روی آن صورت گیرد که در قسمت قبل مطرح شد.
از بحث فوق، تنها استفاده از کلاس‌های Config و SingletonCore ، جهت استفاده و مدیریت بهینه‌ی سشن فکتوری در برنامه‌های ویندوزی کفایت می‌کنند.

دریافت سورس برنامه قسمت هفتم

ادامه دارد ....

نظرات خوانندگان

نویسنده: mohammad
تاریخ: ۱۳۸۹/۰۹/۱۴ ۱۴:۰۱:۳۸

با سلام خدمت استاد
ممنون از مقاله خوبتون
ایا مفهومی شبیه به مدیریت بهینه‌ی سشن فکتوری در Entity Framwork هم وجود دارد. یعنی می توان در برنامه های وب جهت استفاده از شی DataContext از الگوی سینگلتون استفاده نمود؟

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۹/۱۴ ۱۸:۲۱:۵۷

بحث entity framework با NHibernate متفاوت است.
در NHibernate این متد BuildSessionFactory فوق کار بارگذاری متادیتا و نگاشت‌ها و غیره رو انجام میده؛ یعنی خودکار نیست و اگر قرار باشه به ازای هر کوئری یکبار فراخوانی شود اصلا نمی‌شود با برنامه کار کرد چون به شدت کند خواهد بود. به همین جهت کش کردن آن‌را با استفاده از الگوی singleton به صورت فوق تنها یکبار باید انجام داد. یکبار در طول عمر برنامه باید نگاشت‌ها صورت گیرد و پس از آن بارها و بارها از آن استفاده شود چون قرار نیست در طول عمر یک برنامه در حال اجرا تغییری کند.
در حالیکه در Entity framework اینکار (بارگذاری متادیتا و نگاشت‌های تعریف شده) به صورت خودکار زمانیکه برنامه برای بار اول اجرا می‌شود رخ داده و به صورت خودکار هم کش می‌شود. ماخذ: (+) ؛ بنابراین برای مدیریت آن اصلا لازم نیست شما کاری انجام دهید.
فقط در Entity framework یک لایه بسیار کم هزینه به نام ObjectContext وجود دارد که توصیه شده در برنامه‌های ASP.NET به ازای هر درخواست ایجاد و تخریب شود که می‌توانید از مقاله فوق ایده بگیرید و اصلا نباید کش شود یا هر بحث دیگری. ماخذ: (+) ؛ حتی اگر اینکار را هم انجام ندادید مهم نیست چون سربار بسیار کمی دارد.

نویسنده: رضا
تاریخ: ۱۳۹۱/۰۵/۲۷ ۱۵:۳۰

با سلام.
برای استفاده از NHibernate باید تمام نگاشت‌ها رو به صورت دستی انجام داد؟ یعنی مثل EF محیط Wizard وجود نداره که پس از طراحی دیتابیس این نگاشت جدول‌ها اوتوماتیک صورت بگیره ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۷ ۱۶:۲۶

سیستم NHibernate از روز اول آن Code first بوده. EF هم در نگارش آخر آن به این نتیجه رسیده که روش Code first انعطاف پذیری بیشتری داره و دقیقا چیزی هست که برنامه نویس‌ها با آن راحت‌تر هستند.
البته برای NH یک سری ابزار تجاری توسط شرکت‌های ثالث درست شده که طراح دارد ولی ... فکر نمی‌کنم با استقبال مواجه شده باشد چون روش Code first یعنی رها شدن از انبوهی کد که توسط ابزارها نوشته می‌شن و عموما هم بهینه نیستند.