عنوان: آشنایی با NHibernate - قسمت چهارم
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۷/۲۰ ۱۰:۱۵:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: NHibernate

در این قسمت یک مثال ساده از load ، insert و delete را بر اساس اطلاعات قسمت‌های قبل با هم مرور خواهیم کرد. برای سادگی کار از یک برنامه Console استفاده خواهد شد (هر چند مرسوم شده است که برای نوشتن آزمایشات از آزمون‌های واحد بجای این نوع پروژه‌ها استفاده شود). همچنین فرض هم بر این است که database schema برنامه را مطابق قسمت قبل در اس کیوال سرور ایجاد کرده اید (نکته آخر بحث قسمت سوم).

یک پروژه جدید از نوع کنسول را به solution برنامه (همان NHSample1 که در قسمت‌های قبل ایجاد شد)، اضافه نمائید.
سپس ارجاعاتی را به اسمبلی‌های زیر به آن اضافه کنید:

FluentNHibernate.dll

NHibernate.dll

NHibernate.ByteCode.Castle.dll

NHSample1.dll : در قسمت‌های قبل تعاریف موجودیت‌ها و نگاشت آن‌ها را در این پروژه class library ایجاد کرده بودیم و اکنون قصد استفاده از آن را داریم.

اگر دیتابیس قسمت قبل را هنوز ایجاد نکرده‌اید، کلاس CDb را به برنامه افزوده و سپس متد CreateDb آن‌را به برنامه اضافه نمائید.

```
using FluentNHibernate;
using FluentNHibernate.Cfg;
using FluentNHibernate.Cfg.Db;
using NHSample1.Mappings;

namespace ConsoleTestApplication
{
  class CDb
  {
      public static void CreateDb(IPersistenceConfigurer dbType)
      {
          var cfg = Fluently.Configure().Database(dbType);

          PersistenceModel pm = new PersistenceModel();
          pm.AddMappingsFromAssembly(typeof(CustomerMapping).Assembly);
          var sessionSource = new SessionSource(
                              cfg.BuildConfiguration().Properties,
                              pm);

          var session = sessionSource.CreateSession();
          sessionSource.BuildSchema(session, true);
      }
  }
}
```

اکنون برای ایجاد دیتابیس اس کیوال سرور بر اساس نگاشت‌های قسمت قبل، تنها کافی است دستور ذیل را صادر کنیم:

```
CDb.CreateDb(
               MsSqlConfiguration
                  .MsSql2008
                  .ConnectionString("Data Source=(local);Initial Catalog=HelloNHibernate;Integrated
Security = true")
                  .ShowSql());
```

تمامی جداول و ارتباطات مرتبط در دیتابیسی که در کانکشن استرینگ فوق ذکر شده است، ایجاد خواهد شد.

در ادامه یک کلاس جدید به نام Config را به برنامه کنسول ایجاد شده اضافه کنید:

```
using FluentNHibernate.Cfg;
using FluentNHibernate.Cfg.Db;
using NHibernate;
using NHSample1.Mappings;

namespace ConsoleTestApplication
{
  class Config
  {
      public static ISessionFactory CreateSessionFactory(IPersistenceConfigurer dbType)
      {
          return
             Fluently.Configure().Database(dbType
              ).Mappings(m => m.FluentMappings.AddFromAssembly(typeof(CustomerMapping).Assembly))
               .BuildSessionFactory();
      }
  }
}
```

اگر بحث را دنبال کرده باشید، این کلاس را پیشتر در کلاس FixtureBase آزمون واحد خود، به نحوی دیگر دیده بودیم. برای کار با NHibernate نیاز به یک سشن مپ شده به موجودیت‌های برنامه می‌باشد که توسط متد CreateSessionFactory کلاس فوق ایجاد خواهد شد. این متد را به این جهت استاتیک تعریف کرده‌ایم که هیچ نوع وابستگی به کلاس جاری خود ندارد. در آن نوع دیتابیس مورد استفاده (برای مثال اس کیوال سرور 2008 یا هر مورد دیگری که مایل بودید)، به همراه اسمبلی حاوی اطلاعات نگاشت‌های برنامه معرفی شده‌اند.

اکنون سورس کامل مثال برنامه را در نظر بگیرید:

کلاس CDbOperations جهت اعمال ثبت و حذف اطلاعات:

```
using System;
using NHibernate;
using NHSample1.Domain;

namespace ConsoleTestApplication
{
  class CDbOperations
  {
      ISessionFactory _factory;

      public CDbOperations(ISessionFactory factory)
      {
          _factory = factory;
      }

      public int AddNewCustomer()
      {
          using (ISession session = _factory.OpenSession())
          {
              using (ITransaction transaction = session.BeginTransaction())
              {
                  Customer vahid = new Customer()
                  {
                      FirstName = "Vahid",
                      LastName = "Nasiri",
                      AddressLine1 = "Addr1",
                      AddressLine2 = "Addr2",
                      PostalCode = "1234",
                      City = "Tehran",
                      CountryCode = "IR"
                  };

                  Console.WriteLine("Saving a customer...");

                  session.Save(vahid);
                  session.Flush();//چندین عملیات با هم و بعد

                  transaction.Commit();

                  return vahid.Id;
```

```
            }
        }
    }

    public void DeleteCustomer(int id)
    {
        using (ISession session = _factory.OpenSession())
        {
            using (ITransaction transaction = session.BeginTransaction())
            {
                Customer customer = session.Load<Customer>(id);
                Console.WriteLine("Id:{0}, Name: {1}", customer.Id, customer.FirstName);

                Console.WriteLine("Deleting a customer...");
                session.Delete(customer);

                session.Flush();//چندین عملیات با هم و بعد

                transaction.Commit();
            }
        }
    }
  }
}
```

و سپس استفاده از آن در برنامه

```
using System;
using FluentNHibernate.Cfg.Db;
using NHibernate;
using NHSample1.Domain;

namespace ConsoleTestApplication
{
  class Program
  {
      static void Main(string[] args)
      {
          //CDb.CreateDb(SQLiteConfiguration.Standard.ConnectionString("data
source=sample.sqlite").ShowSql());
          //return;

          //todo: Read ConnectionString from app.config or web.config
          using (ISessionFactory session = Config.CreateSessionFactory(
                MsSqlConfiguration
                   .MsSql2008
                   .ConnectionString("Data Source=(local);Initial Catalog=HelloNHibernate;Integrated
Security = true")
                   .ShowSql()
              ))
          {
              CDbOperations db = new CDbOperations(session);
              int id = db.AddNewCustomer();
              Console.WriteLine("Loading a customer and delete it...");
              db.DeleteCustomer(id);
          }

          Console.WriteLine("Press a key...");
          Console.ReadKey();
      }
  }
}
```

توضیحات:

نیاز است تا ISessionFactory را برای ساخت سشن‌های دسترسی به دیتابیس ذکر شده در تنظیمات آن جهت استفاده در تمام تردهای برنامه، ایجاد نمائیم. لازم به ذکر است که تا قبل از فراخوانی BuildSessionFactory این تنظیمات باید معرفی شده باشند و پس از آن دیگر اثری نخواهند داشت.

ایجاد شیء ISessionFactory هزینه بر است و گاهی بر اساس تعداد کلاس‌هایی که باید مپ شوند، ممکن است تا چند ثانیه به طول انجامد. به همین جهت نیاز است تا یکبار ایجاد شده و بارها مورد استفاده قرار گیرد. در برنامه به کرات از using استفاده شده تا اشیاء IDisposable را به صورت خودکار و حتمی، معدوم نماید.

بررسی متد AddNewCustomer :

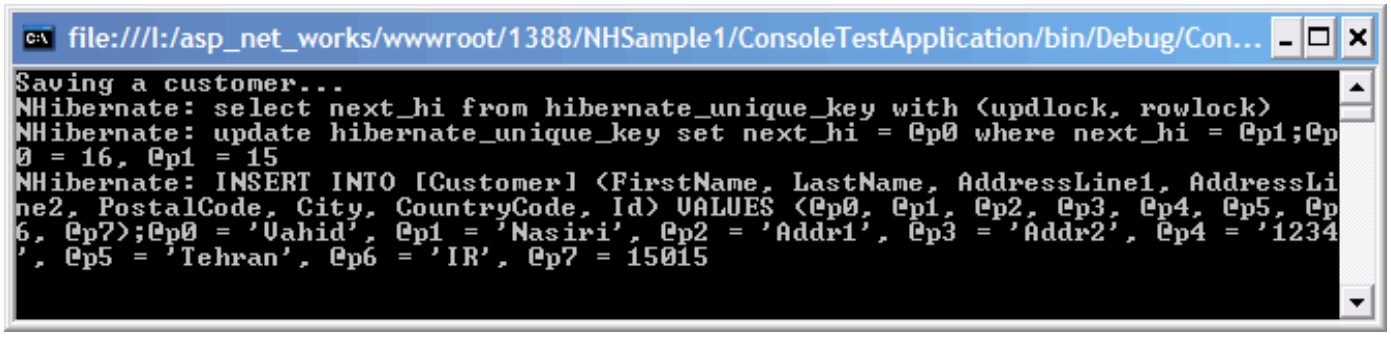
در ابتدا یک سشن را از ISessionFactory موجود درخواست می‌کنیم. سپس یکی از بهترین تمرین‌های کاری جهت کار با دیتابیس‌ها ایجاد یک تراکنش جدید است تا اگر در حین اجرای کوئری‌ها مشکلی در سیستم، سخت افزار و غیره پدید آمد، دیتابیسی ناهماهنگ حاصل نشود. زمانیکه از تراکنش استفاده شود، تا هنگامیکه دستور transaction.Commit آن با موفقیت به پایان نرسیده باشد، اطلاعاتی در دیتابیس تغییر نخواهد کرد و از این لحاظ استفاده از تراکنش‌ها جزو الزامات یک برنامه اصولی است.

در ادامه یک وهله از شیء Customer را ایجاد کرده و آن‌را مقدار دهی می‌کنیم (این شیء در قسمت‌های قبل ایجاد گردید). سپس با استفاده از session.Save دستور ثبت را صادر کرده، اما تا زمانیکه transaction.Commit فراخوانی و به پایان نرسیده باشد، اطلاعاتی در دیتابیس ثبت نخواهد شد.

نیازی به ذکر سطر فلاش در این مثال نبود و NHibernate اینکار را به صورت خودکار انجام می‌دهد و فقط از این جهت عنوان گردید که اگر چندین عملیات را با هم معرفی کردید، استفاده از session.Flush سبب خواهد شد که رفت و برگشت‌ها به دیتابیس حداقل شود و فقط یکبار صورت گیرد.

در پایان این متد، Id ثبت شده در دیتابیس بازگشت داده می‌شود.

چون در متد CreateSessionFactory ، متد ShowSql را نیز ذکر کرده بودیم، هنگام اجرای برنامه، عبارات SQL ایی که در پشت صحنه توسط NHibernate تولید می‌شوند را نیز می‌توان مشاهده نمود:



بررسی متد DeleteCustomer :

ایجاد سشن و آغاز تراکنش آن همانند متد AddNewCustomer است. سپس در این سشن، یک شیء از نوع Customer با Id ایی مشخص load خواهد گردید. برای نمونه، نام این مشتری نیز در کنسول نمایش داده می‌شود. سپس این شیء مشخص و بارگذاری شده را به متد session.Delete ارسال کرده و پس از فراخوانی transaction.Commit ، این مشتری از دیتابیس حذف می‌شود.

برای نمونه خروجی SQL پشت صحنه این عملیات که توسط NHibernate مدیریت می‌شود، به صورت زیر است:

```
Saving a customer...
NHibernate: select next_hi from hibernate_unique_key with (updlock, rowlock)
NHibernate: update hibernate_unique_key set next_hi = @p0 where next_hi = @p1;@p0 = 17, @p1 = 16
NHibernate: INSERT INTO [Customer] (FirstName, LastName, AddressLine1, AddressLine2, PostalCode, City,
CountryCode, Id) VALUES (@p0, @p1, @p2, @p3, @p4, @p5, @p6, @p7);@p0 = 'Vahid', @p1 = 'Nasiri', @p2 =
'Addr1', @p3 = 'Addr2', @p4 = '1234', @p5 = 'Tehran', @p6 = 'IR', @p7 = 16016
Loading a customer and delete it...
NHibernate: SELECT customer0_.Id as Id2_0_, customer0_.FirstName as FirstName2_0_, customer0_.LastName
as LastName2_0_, customer0_.AddressLine1 as AddressL4_2_0_, customer0_.AddressLine2 as AddressL5_2_0_,
customer0_.PostalCode as PostalCode2_0_, customer0_.City as City2_0_, customer0_.CountryCode as
CountryC8_2_0_ FROM [Customer] customer0_ WHERE customer0_.Id=@p0;@p0 = 16016
Id:16016, Name: Vahid
Deleting a customer...
NHibernate: DELETE FROM [Customer] WHERE Id = @p0;@p0 = 16016
Press a key...
```

استفاده از دیتابیس SQLite بجای SQL Server در مثال فوق:

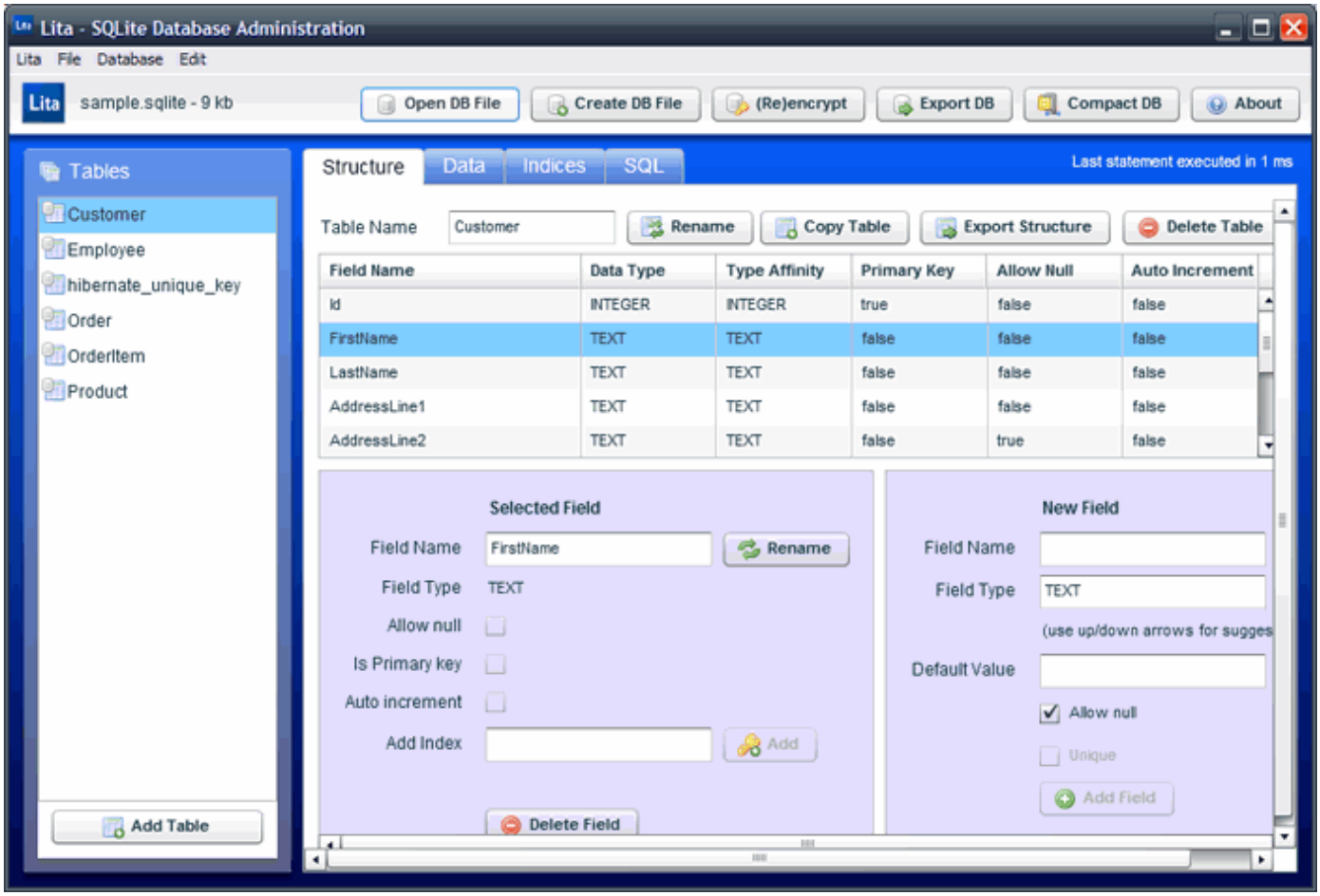
فرض کنید از هفته آینده قرار شده است که نسخه سبک و تک کاربره‌ای از برنامه ما تهیه شود. بدیهی است SQL server برای این منظور انتخاب مناسبی نیست (هزینه بالا برای یک مشتری، مشکلات نصب، مشکلات نگهداری و امثال آن برای یک کاربر نهایی و نه یک سازمان بزرگ که حتما ادمینی برای این مسایل در نظر گرفته می‌شود).
اکنون چه باید کرد؟ باید برنامه را از صفر بازنویسی کرد یا قسمت دسترسی به داده‌های آن‌را کلا مورد باز بینی قرار داد؟ اگر برنامه اسپاگتی ما اصلا لایه دسترسی به داده‌ها را نداشت چه؟! همه جای برنامه پر است از SqlCommand و Open و Close ! و عملا استفاده از یک دیتابیس دیگر یعنی باز نویسی کل برنامه.
همانطور که ملاحظه می‌کنید، زمانیکه با NHibernate کار شود، مدیریت لایه دسترسی به داده‌ها به این فریم ورک محول می‌شود و اکنون برای استفاده از دیتابیس SQLite تنها باید تغییرات زیر صورت گیرد:
ابتدا ارجاعی را به اسمبلی System.Data.SQLite.dll اضافه نمائید (تمام این اسمبلی‌های ذکر شده به همراه مجموعه FluentNHibernate ارائه می‌شوند). سپس:
الف) ایجاد یک دیتابیس خام بر اساس کلاس‌های domain و mapping تعریف شده در قسمت‌های قبل به صورت خودکار

```
CDb.CreateDb(SQLiteConfiguration.Standard.ConnectionString("data source=sample.sqlite").ShowSql());
```

ب) تغییر آرگومان متد CreateSessionFactory

```
//todo: Read ConnectionString from app.config or web.config
          using (ISessionFactory session = Config.CreateSessionFactory(
                    SQLiteConfiguration.Standard.ConnectionString("data
source=sample.sqlite").ShowSql()
              ))
          {
 ...
```

نمایی از دیتابیس SQLite تشکیل شده پس از اجرای متد قسمت الف ، در برنامه Lita :



دریافت سورس برنامه تا این قسمت

نکته:
در سه قسمت قبل، تمام خواص پابلیک کلاس‌های پوشه domain را به صورت معمولی و متداول معرفی کردیم. اگر نیاز به lazy loading در برنامه وجود داشت، باید تمامی کلاس‌ها را ویرایش کرده و واژه کلیدی virtual را به کلیه خواص پابلیک آن‌ها اضافه کرد. علت هم این است که برای عملیات lazy loading ، فریم ورک NHibernate باید یک سری پروکسی را به صورت خودکار جهت کلاس‌های برنامه ایجاد نماید و برای این امر نیاز است تا بتواند این خواص را تحریف (override) کند. به همین جهت باید آن‌ها را به صورت virtual تعریف کرد. همچنین تمام سطرهای Not.LazyLoad نیز باید حذف شوند.

ادامه دارد ...

نظرات خوانندگان

نویسنده: peyman naji
تاریخ: ۱۳۸۹/۰۸/۲۴ ۰۸:۲۹:۵۵

با سلام
مشکلی که با اون برخورد کردم اینه که جداول generate میشه دیتا هم میشه وارد کرد . اما وقتی با برنامه brwoser که معرفی کردید میخوام دیتا بیس رو باز کنم هیچ چیزی نمایش داده نمیشه . راهنمایی بفرمائید . با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۸/۲۴ ۰۸:۵۵:۴۲

سلام،
فقط در یک حالت این مورد ممکن است:
- SQLite مد کار کردن در حالت تشکیل دیتابیس در حافظه هم دارد. این مورد برای انجام آزمایشات واحد بسیار مرسوم و مفید است چون هم سریع است و هم پس از پایان کار اثری از رکوردها باقی نخواهد ماند.
بنابراین بررسی کنید که بانک اطلاعاتی SQLite را در چه حالتی آغاز می‌کنید.

نویسنده: مهدی پایروند
تاریخ: ۱۳۸۹/۱۰/۰۴ ۱۴:۱۱:۲۸

سلام سری مقاله های جالب و مفیدی هستند و برای اینکه بنظرم سوال من به این بخش از سری ارتباط داره این پست رو انتخاب کردم
اگه که دیتا بیس از قبل تهیه شده باشه و دیگه اینکه یک موجودیت خواص خودشو از چند تا جدول دریافت کنه چطور باید عملیات مپ رو با استفاده از Fluent انجام داد. متشکرم

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۱۰/۰۴ ۱۴:۲۵:۱۶

روابط یک به چند و چند به چند رو در طی مقالات این سری توضیح دادم. باید وقت بگذارید این‌ها را مطالعه کنید (مواردی مانند one-to-many و many-to-many و غیره که ذکر شده به همین دلیل است).
یا اینکه می‌تونید از ابزار استفاده کنید: NHibernate Mapping Generator

نویسنده: مهدی پایروند
تاریخ: ۱۳۸۹/۱۰/۰۴ ۱۴:۳۵:۴۳

خیلی ممنون از راهنماییتون، یه سوال دیگه اینکه محصور کننده هایی مانند Linq To NHibernate و غیره از نسخه های قبلی NH استفاده کرده اند آیا راهی برای رفع این مشکل وجود دارد مثال
NHibernate.Linq نسخه 1.0.0.4000 نیاز به NHibernate نسخه 2.1.0.4000 دارد و غیره

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۱۰/۰۴ ۱۶:۴۵:۱۳

NH 3.0 نیازی به فایل‌های کمکی LINQ قدیمی ندارد و از یک کتابخانه‌ی دیگر و پیشرفته‌تر به صورت یکپارچه استفاده می‌کند.
بنابراین نیازی نیست که از فایل LINQ موجود در (+) استفاده کرد. سایر موارد آن برای NH 3.0 به روز شده‌اند و قابل استفاده است.