

عنوان: ویدیوهای آموزشی NHibernate
نویسنده: وحید نصیری
تاریخ: ۱۳۸۷/۰۸/۰۸ ۱۰:۵۱:۱۴
آدرس: www.dotnettips.info
برچسب‌ها: NHibernate

دو سری ویدیوی رایگان آموزشی [NHibernate](http://www.summerofnhibernate.com) در سال جاری در مجامع مرتبط ارائه شده است که دیدن آنها خالی از لطف نیست. حتی اگر از NHibernate هم نخواهید استفاده کنید مفاهیم `unit testing` , `refactoring` و امثال آن در این مجموعه‌ها به شکل بسیار مبسوطی توضیح داده شده‌اند.

سری اول:

Summer of NHibernate Screencast Series

[/http://www.summerofnhibernate.com](http://www.summerofnhibernate.com)

سری دوم:

تا این لحظه 5 ویدیوی مقدماتی NHibernate در سایت <http://www.dimecasts.net/Casts/ByTag/NHibernate> منتشر شده‌اند. سایت جالبی است و محدود به این مورد خاص نیست.

نظرات خوانندگان

نویسنده: hajloo
تاریخ: ۱۳۸۷/۰۸/۰۸ ۱۲:۱۹:۰۰

بسیار استفاده کردم . واقعا ممنون . فکر کنم یک وبلاگ خوب کامپیوتری دیگه پیدا کردم . امیدوارم همینطور خوب & Slow Steady (آهسته و پیوسته) ادامه بدید .

نویسنده: hajloo
تاریخ: ۱۳۸۷/۰۸/۰۸ ۱۲:۲۴:۰۰

در مورد PDC هم بنویس واینکه به نظرت اصلا یک همچین مجامعی در دنیا برگزار شدنشون یا برگزار نشدنشون فرقی به حال ما می کنه ؟ اصلا خودت سعی می کنی که به اون سمت بری ؟ لطفا نظر شخصی بنویس و کاری به شرکت و ... نداشته باش . مطمئنا وطالب این وبلاگ رو دنبال می کنم

نویسنده: Hossein Moradinia
تاریخ: ۱۳۸۸/۰۵/۲۷ ۱۸:۴۴:۳۳

سلام وحید جان
لینک اول جواب نمیده

نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۵/۲۷ ۲۰:۲۶:۵۶

یا مشکل DNS هست یا از اون طرف ما رو فیلتر کردند.
از یک فیلتر شکن استفاده کنید کار می‌کنه.

نویسنده: Anonymous
تاریخ: ۱۳۸۸/۰۶/۰۴ ۲۳:۲۴:۱۸

آقای نصیری با استفاده از ... هم کار نمی‌کنه اگه لطف کنید خودتان جایی upload کنید ممنون می‌شوم

نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۶/۰۵ ۰۰:۱۹:۱۲

سلام
یک سری از این‌ها که زیر 200 مگ بود را از اینجا دریافت کنید:
<http://vahid.nasiri.googlepages.com/NH-links.txt>

مابقی را باید به سایت مربوطه مراجعه کنید (چون نمی‌شود به ریپدشیر منتقل کرد).

نویسنده: Anonymous
تاریخ: ۱۳۸۸/۰۶/۰۵ ۰۰:۵۵:۲۲

آقای نصیری تنها چیزی که می‌توانم بگویم خدا خیرتان دهد

نویسنده: Anonymous
تاریخ: ۱۳۸۸/۰۶/۰۵ ۱۵:۰۷:۰۳

یک سری از این لینک‌ها فقط کد هستند video این‌ها را ندارید؟ باز هم تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۶/۰۵ ۱۶:۱۴:۳۰

عرض کردم. رپیدشیر فقط تا 200 مگ اجازه remote upload یا بقولی ترنس‌لود را می‌دهد. برای مابقی به سایت اصلی مراجعه کنید (یعنی 6 قسمت دیگر که هر کدام بالای 200 مگ است).

عنوان: خلاصه‌ای از آغاز به کار با NHibernate
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۷/۲۴ ۲۱:۳۸:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: NHibernate

اگر شش یا هفت قسمت قبل را بخوایم به صورت سریع مرور کنیم می‌توان به ویدیوی زیر مراجعه کرد:

[Getting Started with NHibernate](#)

در طی یک ربع، خیلی سریع به دریافت فایل‌های لازم، ایجاد یک پروژه جدید، افزودن ارجاعات لازم، استفاده از fluent NHibernate برای ساخت نگاشت‌ها و سپس استفاده از LINQ to NHibernate برای کوئری گرفتن از اطلاعات دیتابیس اشاره کرده است (که از این لحاظ کاملاً به روز است).

نظرات خوانندگان

نویسنده: Ashkan

تاریخ: ۱۵:۵۶:۵۷ ۱۳۸۸/۰۷/۲۷

با سلام

من مدتی قبل در مورد ORMها تحقیق کردم و به این نتیجه رسیدم که بهترین آنها Entity Framework است که حتی در نهایت LINQ to SQL را هم آرام آرام حذف می کند و جایگزین آن می شود. می توانید مقاله ای در مورد مقایسه Entity و nHibernate Framework تحریر کنید؟ (مثل مقایسه ای که در مورد jQuery و ASP AJAX داشتید)

نویسنده: وحید نصیری

تاریخ: ۱۷:۲۶:۳۷ ۱۳۸۸/۰۷/۲۷

لطفا این سری را مطالعه بفرمائید (از ابتدا).
لابلای توضیحات تفاوت‌های بنیادین برشمرده شده است.

NHibernate کتابخانه‌ی تبدیل شده پروژه بسیار محبوب Hibernate جاوا به سی شارپ است و یکی از ORM های بسیار موفق، به شمار می‌رود. در طی تعدادی مقاله قصد آشنایی با این فریم ورک را داریم.

چرا نیاز است تا از یک ORM استفاده شود؟

تهیه قسمت و یا لایه دسترسی به داده‌ها در یک برنامه عموماً تا 30 درصد زمان کل تهیه یک محصول را تشکیل می‌دهد. اما باید در نظر داشت که این پروسه‌ی تکراری هیچ کار خارق العاده‌ای نبوده و ارزش افزوده‌ی خاصی را به یک برنامه اضافه نمی‌کند. تقریباً تمام برنامه‌های تجاری نیاز به لایه دسترسی به داده‌ها را دارند. پس چرا ما باید به ازای هر پروژه، این کار تکراری و کسل کننده را بارها و بارها تکرار کنیم؟

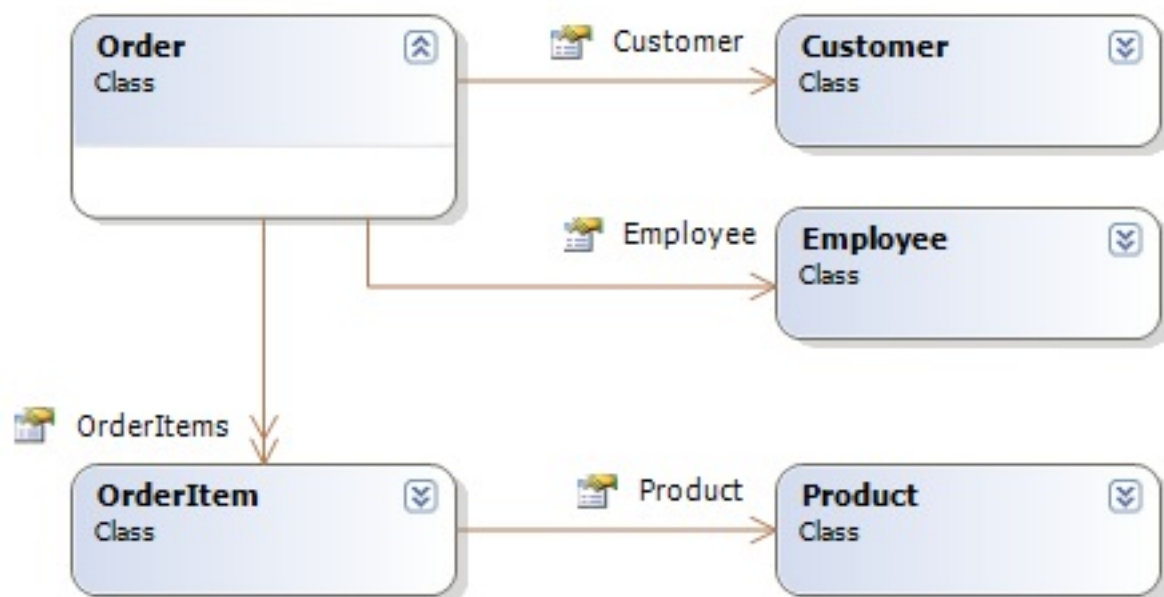
هدف NHibernate، کاستن این بار از روی شانه‌های یک برنامه نویس است. با کمک این کتابخانه، دیگر رویه ذخیره شده‌ای را نخواهید نوشت. دیگر هیچگاه با ADO.Net سر و کار نخواهید داشت. به این صورت می‌توان عمده وقت خود را صرف قسمت‌های اصلی و طراحی برنامه کرد تا کد نویسی یک لایه تکراری. همچنین عده‌ای از بزرگان اینگونه ابزارها اعتقاد دارند که برنامه نویسی‌هایی که لایه دسترسی به داده‌ها را خود طراحی می‌کنند، مشغول کلاهبرداری از مشتری‌های خود هستند! (صرف زمان بیشتر برای تهیه یک محصول و همچنین وجود باگ‌های احتمالی در لایه دسترسی به داده‌های طراحی شده توسط یک برنامه نویس نه چندان حرفه‌ای)

برای مشاهده سایر مزایای استفاده از یک ORM لطفاً به مقاله "[5 دلیل برای استفاده از یک ابزار ORM](#)" مراجعه نمایید.

در ادامه برای معرفی این کتابخانه یک سیستم ثبت سفارشات را با هم مرور خواهیم کرد.

بررسی مدل سیستم ثبت سفارشات

در این مدل ساده‌ی ما، مشتری‌ها (customers) امکان ثبت سفارشات (orders) را دارند. سفارشات توسط یک کارمند (employee) که مسئول ثبت آن‌ها است به سیستم وارد می‌شود. هر سفارش می‌تواند شامل یک یا چند آیتم (one-to-many) (order items) باشد و هر آیتم معرف یک محصول (product) است که قرار است توسط یک مشتری (customer) خریداری شود. کلاس دیاگرام این مدل به صورت زیر می‌تواند باشد.



نگاشت مدل

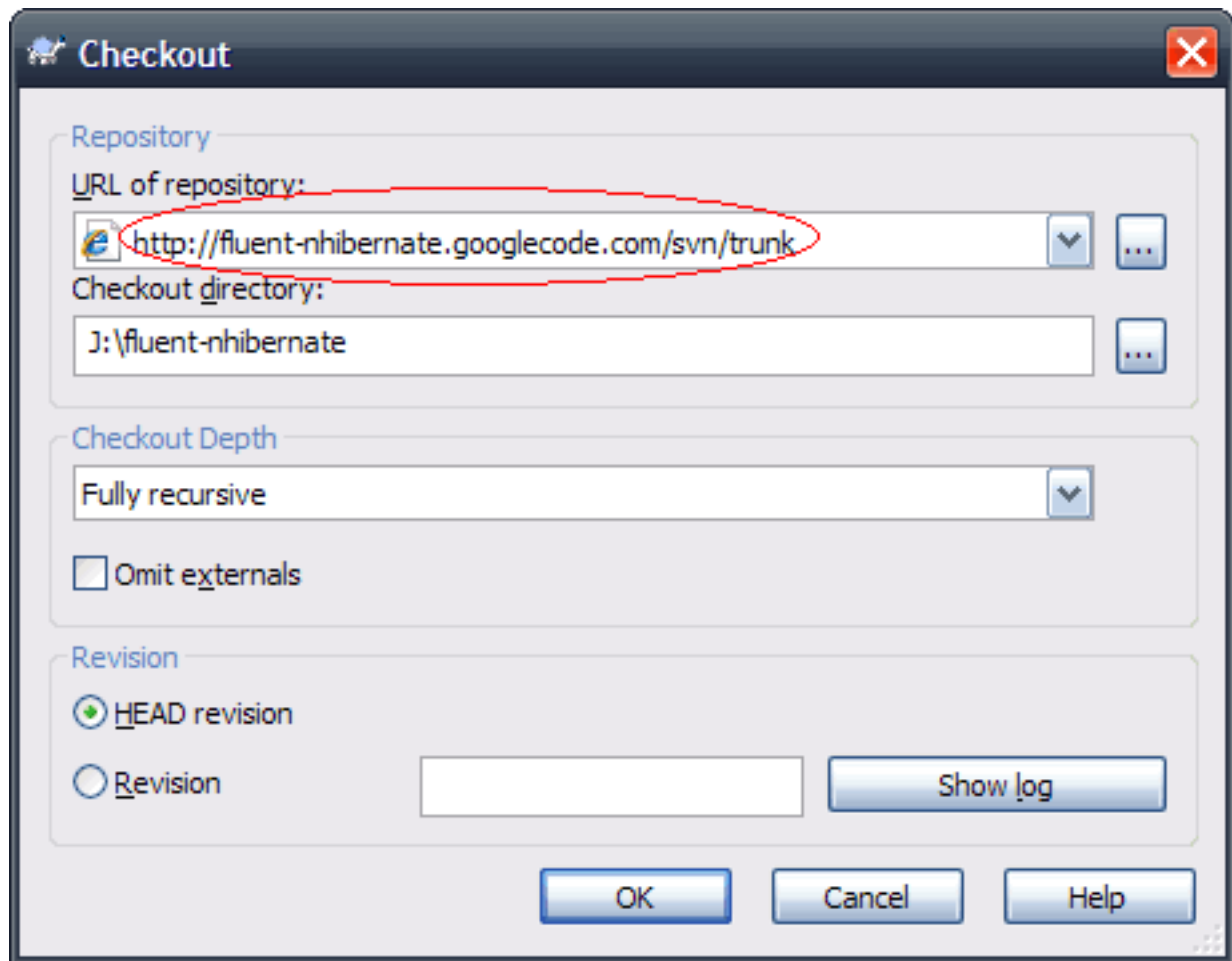
زمانیکه مدل سیستم مشخص شد، اکنون نیاز است تا حالات (داده‌ها) آن‌را در مکانی ذخیره کنیم. عموماً اینکار با کمک سیستم‌های مدیریت پایگاه‌های داده مانند MySQL، Oracle، IBM DB2، SQL Server و امثال آن‌ها صورت می‌گیرد. زمانیکه از NHibernate استفاده کنید اهمیتی ندارد که برنامه شما قرار است با چه نوع دیتابیزی کار کند؛ زیرا این کتابخانه اکثر دیتابیسی‌های شناخته شده موجود را پشتیبانی می‌کند و برنامه از این لحاظ مستقل از نوع دیتابیس عمل خواهد کرد و اگر نیاز بود روزی بجای اس کیوال سرور از مای اس کیوال استفاده شود، تنها کافی است تنظیمات ابتدایی NHibernate را تغییر دهید (بجای بازنویسی کل برنامه).

اگر برای ذخیره سازی داده‌ها و حالات سیستم از دیتابیس استفاده کنیم، نیاز است تا اشیاء مدل خود را به جداول دیتابیس نگاشت نمائیم. این نگاشت عموماً یک به یک نیست (لزومی ندارد که حتماً یک شیء به یک جدول نگاشت شود). در گذشته‌ی نچندان دور کتابخانه‌ی NHibernate، این نگاشت عموماً توسط فایل‌های XML ایی به نام hbm صورت می‌گرفت. این روش هنوز هم پشتیبانی شده و توسط بسیاری از برنامه نویسی‌ها بکار گرفته می‌شود. روش دیگری که برای تعریف این نگاشت مرسوم است، مزین سازی اشیاء و خواص آن‌ها با یک سری از ویژگی‌ها می‌باشد که فریم ورک برتر این عملیات Castle Active Record نام دارد.

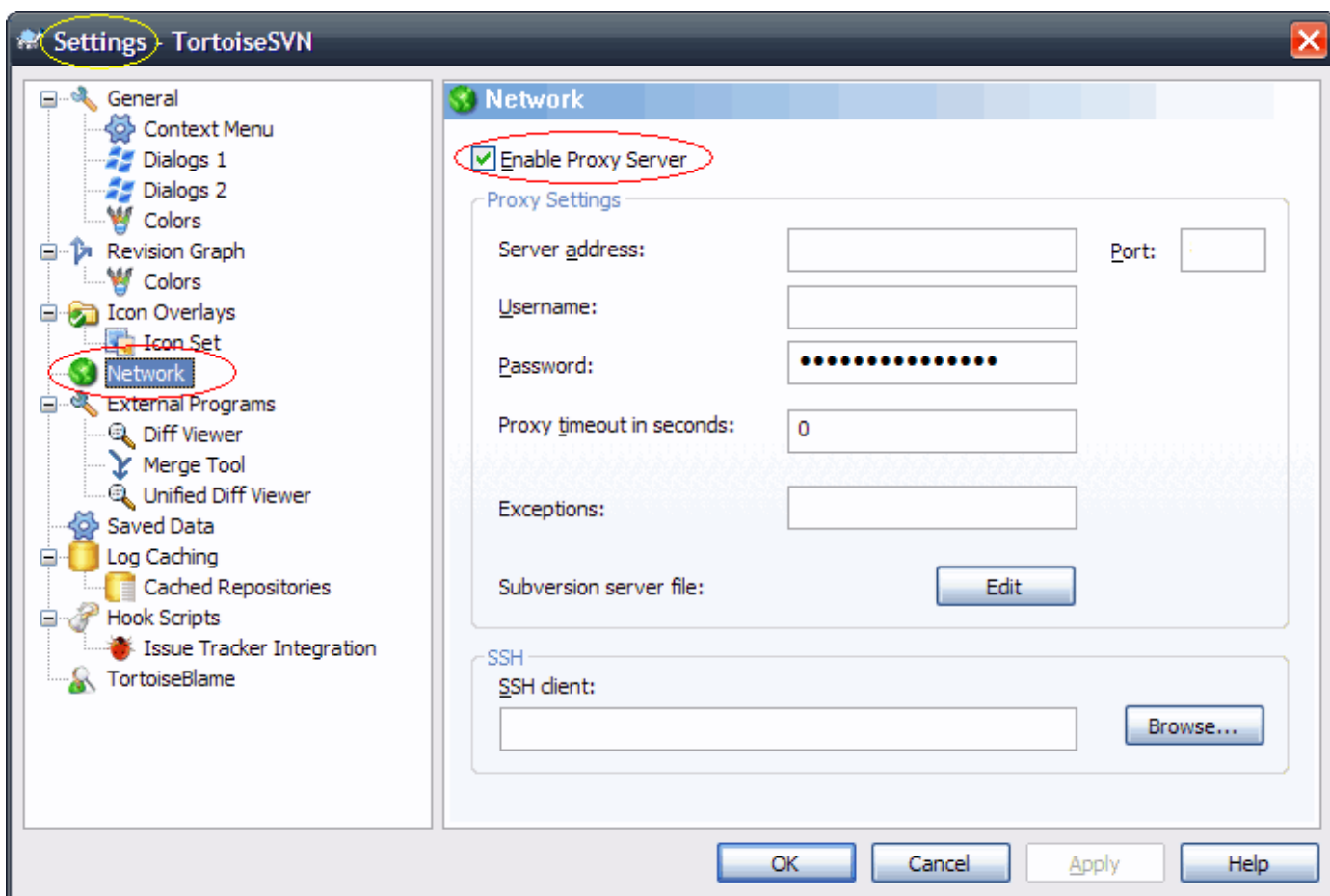
اخیراً کتابخانه‌ی دیگری برای انجام این نگاشت تهیه شده به نام Fluent NHibernate که بسیار مورد توجه علاقمندان به این فریم ورک واقع گردیده است. با کمک کتابخانه‌ی Fluent NHibernate عملیات نگاشت اشیاء به جداول، بجای استفاده از فایل‌های XML، توسط کدهای برنامه صورت خواهند گرفت. این مورد مزایای بسیاری را همانند استفاده از یک زبان برنامه نویسی کامل برای تعریف نگاشت‌ها، بررسی خودکار نوع‌های داده‌ای و حتی امکان تعریف منطقی خاص برای قسمت نگاشت برنامه، به همراه خواهد داشت.

آماده سازی سیستم برای استفاده از NHibernate

در ادامه بجای دریافت پروژه سورس باز [NHibernate](#) از سایت سورس فورج، پروژه سورس باز Fluent NHibernate را از سایت گوگل کد دریافت خواهیم کرد که بر فراز کتابخانه‌ی NHibernate بنا شده است و آن‌را کاملاً پوشش می‌دهد. سورس این کتابخانه را با checkout مسیر زیر توسط [TortoiseSVN](#) می‌توان دریافت کرد.



البته احتمالا برای دریافت آن از گوگل کد با توجه به تحریم موجود نیاز به پروکسی خواهد بود. برای تنظیم پروکسی در TortoiseSVN به قسمت تنظیمات آن مطابق تصویر ذیل مراجعه کنید:



همچنین جهت سهولت کار، آخرین نگارش موجود در زمان نگارش این مقاله را از [این آدرس](#) نیز می‌توانید دریافت نمایید.

پس از دریافت پروژه، باز کردن فایل solution آن در VS و سپس build کل مجموعه، اگر به پوشه‌های آن مراجعه نمایید، فایل‌های زیر قابل مشاهده هستند:

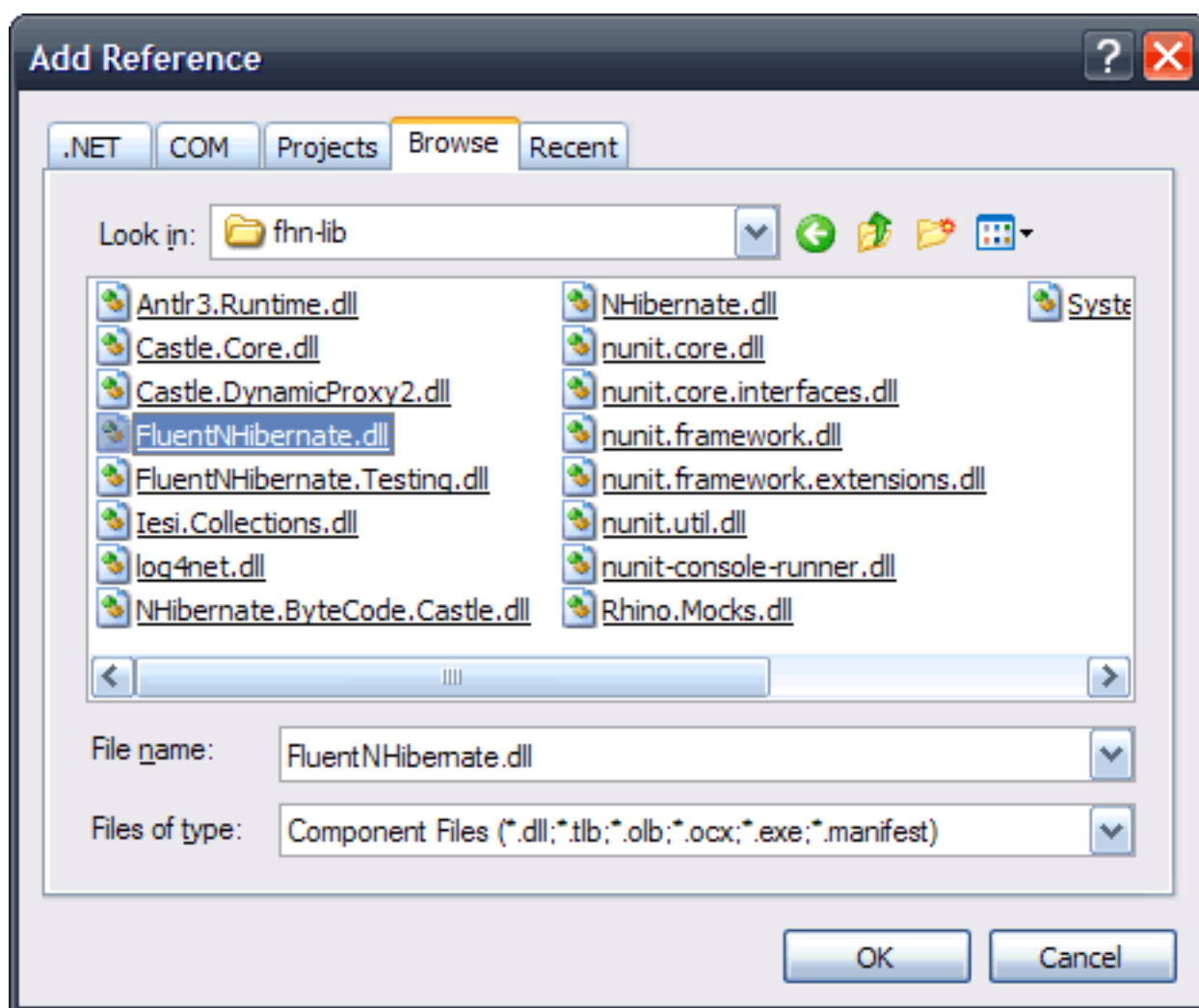
- NHibernate.dll : اسمبلی فریم ورک NHibernate است.
- NHibernate.Linq.dll : اسمبلی پروایدر LINQ to NHibernate می‌باشد.
- FluentNHibernate.dll : اسمبلی فریم ورک Fluent NHibernate است.
- Iesi.Collections.dll : یک سری مجموعه‌های ویژه مورد استفاده NHibernate را ارائه می‌دهد.
- Log4net.dll : فریم ورک لاگ کردن اطلاعات NHibernate می‌باشد. (این فریم ورک نیز جهت عملیات logging بسیار معروف و محبوب است)
- Castle.Core.dll : کتابخانه پایه Castle.DynamicProxy2.dll است.
- Castle.DynamicProxy2.dll : جهت اعمال lazy loading در فریم ورک NHibernate بکار می‌رود.
- System.Data.SQLite.dll : پروایدر دیتابیس SQLite است.
- Nunit.framework.dll : نیز یکی از فریم ورک‌های بسیار محبوب آزمون واحد در دات نت فریم ورک است.

برای سادگی مراجعات بعدی، این فایل‌ها را یافته و در پوشه‌ای به نام lib کپی نمایید.

برپایی یک پروژه جدید

پس از دریافت Fluent NHibernate ، یک پروژه Class Library جدید را در VS.Net آغاز کنید (برای مثال به نام NHSample1). سپس یک پروژه دیگر را نیز از نوع Class Library به نام UnitTests به این solution ایجاد شده جهت انجام آزمون‌های واحد برنامه اضافه نمائید.

اکنون به پروژه NHSample1 ، ارجاع‌هایی را به فایل‌های FluentNHibernate.dll و سپس NHibernate.dll در که پوشه lib ایی که در قسمت قبل ساختیم، قرار دارند، اضافه نمائید.



در ادامه یک پوشه جدید به پروژه NHSample1 به نام Domain اضافه کنید. سپس به این پوشه، کلاس Customer را اضافه نمائید:

```
namespace NHSample1.Domain
{
    public class Customer
    {
        public int Id { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string AddressLine1 { get; set; }
        public string AddressLine2 { get; set; }
        public string PostalCode { get; set; }
        public string City { get; set; }
        public string CountryCode { get; set; }
    }
}
```

اکنون نوبت تعریف نگاشت این شیء است. این کلاس باید از کلاس پایه ClassMap مشتق شود. سپس نگاشت‌ها در سازنده‌ی این کلاس باید تعریف گردند.

```
using FluentNHibernate.Mapping;

namespace NHSample1.Domain
{
    class CustomerMapping : ClassMap<Customer>
    {
    }
}
```

همانطور که ملاحظه می‌کنید، نوع این کلاس Generic، همان کلاسی است که قصد داریم نگاشت مرتبط با آن را تهیه نماییم. در ادامه تعریف کامل این کلاس نگاشت را در نظر بگیرید:

```
using FluentNHibernate.Mapping;

namespace NHSample1.Domain
{
    class CustomerMapping : ClassMap<Customer>
    {
        public CustomerMapping()
        {
            Not.LazyLoad();
            Id(c => c.Id).GeneratedBy.HiLo("1000");
            Map(c => c.FirstName).Not.Nullable().Length(50);
            Map(c => c.LastName).Not.Nullable().Length(50);
            Map(c => c.AddressLine1).Not.Nullable().Length(50);
            Map(c => c.AddressLine2).Length(50);
            Map(c => c.PostalCode).Not.Nullable().Length(10);
            Map(c => c.City).Not.Nullable().Length(50);
            Map(c => c.CountryCode).Not.Nullable().Length(2);
        }
    }
}
```

به صورت پیش فرض نگاشت‌های Fluent NHibernate از نوع lazy load هستند که در اینجا عکس آن در نظر گرفته شده است. سپس وضعیت نگاشت تک تک خواص کلاس Customer را مشخص می‌کنیم. توسط `Id(c => c.Id).GeneratedBy.HiLo` به سیستم اعلام خواهیم کرد که فیلد Id از نوع identity است که از 1000 شروع خواهد شد. مابقی موارد هم بسیار واضح هستند. تمامی خواص کلاس Customer ذکر شده، نال را نمی‌پذیرند (منهای AddressLine2) و طول آن‌ها نیز مشخص گردیده است. با کمک Fluent NHibernate، بحث بررسی نوع‌های داده‌ای و همچنین یکی بودن موارد مطرح شده در نگاشت با کلاس اصلی Customer به سادگی توسط کامپایلر بررسی شده و خطاهای آتی کاهش خواهند یافت.

برای آشنایی بیشتر با lambda expressions می‌توان به مقاله زیر مراجعه کرد:

[Step-by-step Introduction to Delegates and Lambda Expressions](#)

ادامه دارد...

نظرات خوانندگان

نویسنده: dadoo

تاریخ: ۱۳۸۸/۰۷/۱۸ ۰۸:۴۳:۳۰

آقای نصیری عزیز

باسلام

آیا استفاده از این ORM مناسبتر است یا LINQ؟ آیا می توانید مقایسه ای هر چند مختصر بین این دو ابزار داشته باشید. ممنون

نویسنده: LoveAjax

تاریخ: ۱۳۸۸/۰۷/۱۸ ۱۰:۱۲:۰۳

ایا nhibernate و fluent تحت هاست های medium trust اجرا می شوند

نویسنده: DotNetCoders

تاریخ: ۱۳۸۸/۰۷/۱۸ ۱۲:۲۰:۱۴

سلام!

جناب نصیری NHibernate رو میشه با VB.Net هم پیاده سازی کرد؟ یا فقط C#؟

نویسنده: وحید نصیری

تاریخ: ۱۳۸۸/۰۷/۱۸ ۱۳:۱۳:۱۱

@DotNetCoders

سورس اصلی کتابخانه، به زبان سی شارپ است اما نهایتا شما از اسمبلی های کامپایل شده مربوطه استفاده خواهید کرد و از اینجا به بعد دیگر تفاوتی نمی کند که زبان دات نتی مورد استفاده چی باشد.

@dadoo

باید دقت داشته باشید که LINQ به تنهایی فقط یک language feature است و نه یک data access technology . بنابراین باید دقیقا sql to entities یا entities to linq را مشخص کرد.

سابقه نزدیک به یک دهه پروژه اصلی Hibernate که توسط جاوا کارها توسعه داده شده، در این فریم ورک لحاظ شده که از هر لحاظ نسبت به entities to LINQ اون رو پخته تر کرده. ضمنا پروایدر LINQ هم برای NH اخیرا توسعه داده شده و از این لحاظ کم و کسری ندارد.

linq to sql برای اس کیوال سرور توسعه داده شد. بعد مایکروسافت اومد اون رو با entities to linq تکمیل کرد (البته linq to sql مطابق وبلاگ رسمی برنامه نویس های MS هنوز هم توسعه پیدا می کنه و در دات 4 شاهد اون خواهیم بود) و توسط linq to entities امکان استفاده از سایر دیتابیس ها هم فراهم شده البته اگر پروایدر آن موجود باشد که تعدادی از آن ها هم تجاری هستند. اما با NH این مشکل رو ندارید چون تقریبا همه نوع دیتابیس معروفی را ساپورت می کند و رایگان هم هست.

learning curve مربوط به NH بیشتر است از سایر orm ها. NH از دات نت فریم 2 به بعد را پشتیبانی می کند اما entities to linq فقط از دات نت فریم ورک سه و نیم سرویس پک یک به بعد به صورت کامل در دسترس است.

در کل در گوگل nhibernate vs linq را جستجو کنید.

@LoveAjax

محیط مدیوم تراست، امکان ریفلکشن رو حذف می کنه و این مورد برای NH و تمام ORM های دیگر نیز مساله ساز خواهد بود. اما برای NH راه حل دارد مطابق مستندات آن:

<http://nhforge.org/wikis/howtonh/run-in-medium-trust.aspx>

نویسنده: محمد امین شریفی
تاریخ: ۱۳۸۸/۰۷/۱۸ ۱۹:۲۶:۵۱

درباره entity های ماکروسافت هم اگر امکانش هست بنویسید.
فناوری های LINQ to entity و ADO.net entity
برای کسی که فقط با MSSQL کار میکند، آیا فناوری های بالا کمبودی نسبت به NHibernate دارند؟
منظور از هاست های medium trust چیست، یعنی ORM ها را نمی توان روی آنها اجرا کرد؟

@}:-

نویسنده: Alex
تاریخ: ۱۳۸۹/۰۱/۱۶ ۱۶:۴۸:۰۰

آقای نصیری میتونید مزایای Fluent رو نسبت به خود NHibernate رو بگید یا مرجعی معرفی کنید؟

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۱/۱۶ ۲۰:۲۱:۰۰

در NHibernate سنتی کار ساخت نگاشت ها توسط یک سری فایل xml صورت می گیرد که ممکن است حین تهیه اولیه پر از اشتباهات تایپی و غیره باشند. این نوع فایل ها تحت کنترل کامپایلر نبوده و در حین کار مشکلات آن ها مشخص می شود.
در Fluent NHibernate کار تعریف نگاشت ها با استفاده از کدهای strongly typed دات نت صورت می گیرد که بلافاصله تحت کنترل کامپایلر هستند. همچنین مبحث Auto Mapping آن را می توانید در قسمت های بعد مطالعه کنید. امکان unit test نوشتن برای نگاشت های این روش بدون حتی درج یک رکورد در دیتابیس میسر است که باز هم در طی چند قسمت به آن پرداخته شده. با توجه به اینکه در روش دوم تعریف نگاشت ها، بلافاصله تحت نظر کامپایلر است امکان refactoring ساده تر آن نیز مهیا است. در روش Fluent اگر علاقمند بودید که این فایل های XML را هم مشاهده کنید به قسمت Mappings در Fluently.Configure خود، متد ExportTo اضافه کنید.

نویسنده: Alex
تاریخ: ۱۳۸۹/۰۱/۱۷ ۰۷:۲۵:۱۷

بینهایت ممنون از توضیحاتی که دادید.

نویسنده: peyman naji
تاریخ: ۱۳۸۹/۰۷/۰۶ ۱۱:۵۰:۳۲

با سلام

در ورژن آلفا 3.0 دیگه خبری از FluentNHibernate.dll نیست چکار باید کرد مهندس ؟ کلا قسمت اول رو نتونستم

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۷/۰۶ ۱۲:۳۵:۲۵

سلام،

سورس هر دو را دریافت کنید. سپس FluentNHibernate را بر اساس نگارش 3 مجددا کامپایل کرده و استفاده کنید :

[+](#)

نویسنده: fateme
تاریخ: ۱۳۸۹/۰۹/۲۱ ۱۱:۵۱:۴۱

جناب آقای نصیری

با سلام

من نمیتونم پروژه رو بگیرم وقتی آدرسو در checkout وارد میکنم در قسمت setting هم تنظیماتو انجام میدم error عدم دسترسی به آدرس رو میده چکار کنم؟

نویسنده: وحید نصیری
تاریخ: ۱۳:۰۳:۲۷ ۱۳۸۹/۰۹/۲۱

آدرس جدید: (+)

نویسنده: fateme
تاریخ: ۱۱:۴۵:۲۶ ۱۳۸۹/۰۹/۲۲

با سلام

من آدرس جدیدی که دادید رو در checkout وارد که می کنم error زیر رو بهم میده
:error validating server certificate for http://github.com:443

unknown certificate issuer

واقعا ممنونم که جواب سوالات رو میدید

نویسنده: وحید نصیری
تاریخ: ۱۲:۳۳:۰۳ ۱۳۸۹/۰۹/۲۲

سلام، علت اینکه از گوگل کد نمی‌تونید فایلی دریافت کنید این است که گوگل ما رو خیلی وقت است تحریم کرده و درب گوگل کد به روی ایرانی‌ها بسته است. به همین جهت عرض کردم که نیاز به پروکسی دارید و نحوه‌ی ورود اطلاعات پروکسی به TortoiseSVN را نیز ذکر کردم.
در مورد GitHub (آدرس جدید) با استفاده از مرورگر وب به آن وارد شوید. بالای صفحه یک دکمه‌ی دریافت هست. به این صورت به سادگی کل مجموعه رو به شکل یک فایل zip می‌تونید دریافت کنید.

نویسنده: وحید نصیری
تاریخ: ۱۲:۱۴:۵۶ ۱۳۸۹/۰۹/۲۴

[pre-release binaries \(v1.2\) with NH3 support](#)

آزمون واحد کلاس نگاشت تهیه شده

در مورد آشنایی با آزمون‌های واحد لطفاً به [برچسب مربوطه](#) در سمت راست سایت مراجعه بفرمائید. همچنین در مورد اینکه چرا به این نوع API کلمه Fluent اطلاق می‌شود، می‌توان به [تعریف آن](#) جهت مطالعه بیشتر مراجعه نمود.

در این قسمت قصد داریم برای بررسی وضعیت کلاس نگاشت تهیه شده یک آزمون واحد تهیه کنیم. برای این منظور ارجاعی را به اسمبلی `nunit.framework.dll` به پروژه `UnitTests` که در ابتدای کار به solution جاری در VS.Net افزوده بودیم، اضافه نمائید (همچنین ارجاع‌هایی به اسمبلی‌های پروژه `NHSample1`، `FluentNHibernate`، `System.Data.SQLite`، `NHibernate` و `NHibernate.ByteCode.Castle` نیز نیاز هستند). تمام اسمبلی‌های این فریم ورک‌ها از پروژه `FluentNHibernate` قابل استخراج هستند.

سپس سه کلاس زیر را به پروژه آزمون واحد اضافه خواهیم کرد.
کلاس `TestModel`: (جهت مشخص سازی محل دریافت اطلاعات نگاشت)

```
using FluentNHibernate;
using NHSample1.Domain;

namespace UnitTests
{
    public class TestModel : PersistenceModel
    {
        public TestModel()
        {
            AddMappingsFromAssembly(typeof(CustomerMapping).Assembly);
        }
    }
}
```

کلاس `FixtureBase`: (جهت ایجاد سشن NHibernate در ابتدای آزمون واحد و سپس پاکسازی اشیاء در پایان کار)

```
using NUnit.Framework;
using NHibernate;
using FluentNHibernate;
using FluentNHibernate.Cfg;
using FluentNHibernate.Cfg.Db;

namespace UnitTests
{
    public class FixtureBase
    {
        protected SessionSource SessionSource { get; set; }
        protected ISession Session { get; private set; }

        [SetUp]
        public void SetupContext()
        {
            var cfg = Fluently.Configure().Database(SQLiteConfiguration.Standard.InMemory);

            SessionSource = new SessionSource(
                cfg.BuildConfiguration().Properties,
                new TestModel());

            Session = SessionSource.CreateSession();
            SessionSource.BuildSchema(Session);
        }
    }
}
```

```

    [TearDown]
    public void TearDownContext()
    {
        Session.Close();
        Session.Dispose();
    }
}

```

و کلاس CustomerMapping_Fixture.cs : (جهت بررسی صحت نگاشت تهیه شده با کمک دو کلاس قبل)

```

using NUnit.Framework;
using FluentNHibernate.Testing;
using NHSample1.Domain;

namespace UnitTests
{
    [TestFixture]
    public class CustomerMapping_Fixture : FixtureBase
    {
        [Test]
        public void can_correctly_map_customer()
        {
            new PersistenceSpecification<Customer>(Session)
                .CheckProperty(c => c.Id, 1001)
                .CheckProperty(c => c.FirstName, "Vahid")
                .CheckProperty(c => c.LastName, "Nasiri")
                .CheckProperty(c => c.AddressLine1, "Addr1")
                .CheckProperty(c => c.AddressLine2, "Addr2")
                .CheckProperty(c => c.PostalCode, "1234")
                .CheckProperty(c => c.City, "Tehran")
                .CheckProperty(c => c.CountryCode, "IR")
                .VerifyTheMappings();
        }
    }
}

```

توضیحات:

اکنون به عنوان یک برنامه نویس متعهد نیاز است تا کار صورت گرفته در قسمت قبل را آزمایش کنیم.

کار بررسی صحت نگاشت تعریف شده در قسمت قبل توسط کلاس استاندارد PersistenceSpecification فریم ورک FluentNHibernate انجام خواهد شد (در کلاس CustomerMapping_Fixture). این کلاس برای انجام عملیات آزمون واحد نیاز به کلاس پایه دیگری به نام FixtureBase دارد که در آن کار ایجاد سشن NHibernate (در قسمت استاندارد Setup آزمون واحد) و سپس آزاد سازی آن را در هنگام خاتمه کار ، انجام می‌دهد (در قسمت TearDown آزمون واحد). این ویژگی‌ها که در مباحث آزمون واحد نیز به آن‌ها اشاره شده است، سبب اجرای متدهایی پیش از اجرا و بررسی هر آزمون واحد و سپس آزاد سازی خودکار منابع خواهند شد.

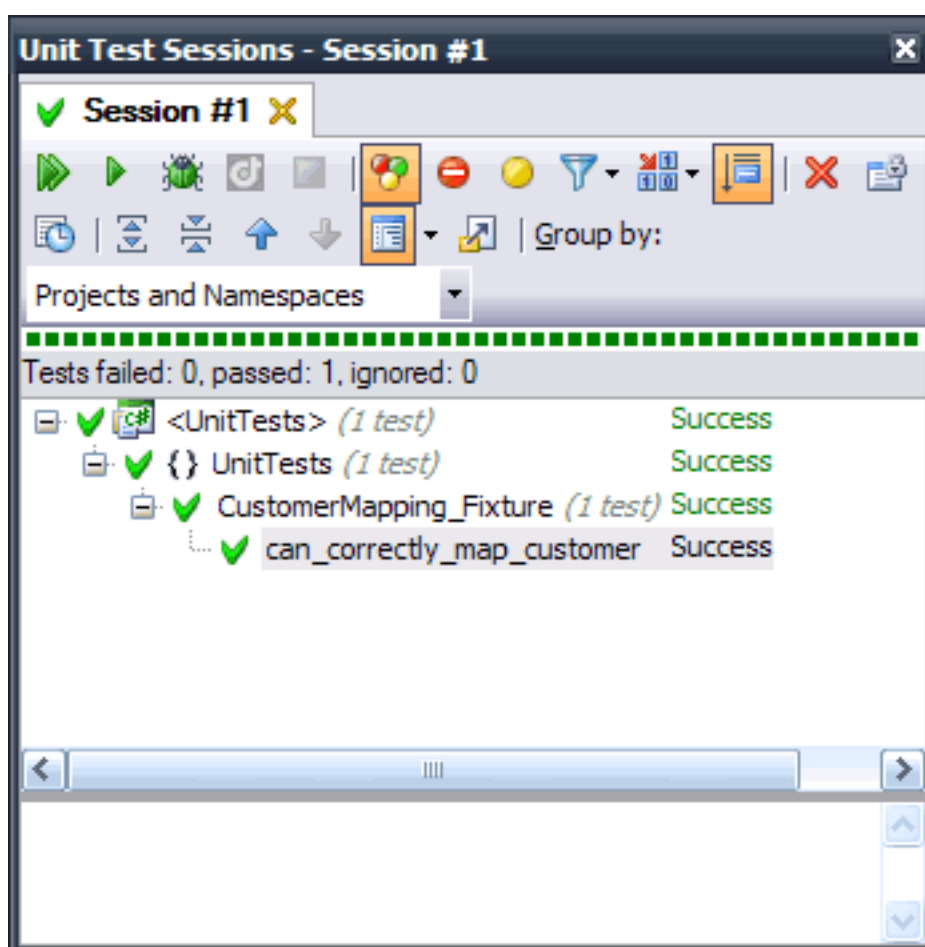
برای ایجاد یک سشن NHibernate نیاز است تا نوع دیتابیس و همچنین رشته اتصالی به آن (کانکشن استرینگ) مشخص شوند. فریم ورک Fluent NHibernate با ایجاد کلاس‌های کمکی برای این امر، به شدت سبب ساده سازی انجام آن شده است. در این مثال، نوع دیتابیس به SQLite و در حالت دیتابیس در حافظه (in memory)، تنظیم شده است (برای انجام امور آزمون واحد با سرعت بالا).

جهت اجرای هر دستوری در NHibernate نیاز به یک سشن می‌باشد. برای تعریف شیء سشن، نه تنها نیاز به مشخص سازی نوع و حالت دیتابیس مورد استفاده داریم، بلکه نیاز است تا وهله‌ای از کلاس استاندارد PersistenceModel را نیز جهت مشخص سازی کلاس نگاشت مورد استفاده مشخص نمائیم. برای این منظور کلاس TestModel فوق تعریف شده است تا این نگاشت را از اسمبلی مربوطه بخواند و مورد استفاده قرار دهد (بر پایی اولیه این مراحل شاید در ابتدای امر کمی زمانبر باشد اما در نهایت یک پروسه استاندارد است). توسط این کلاس به سیستم اعلام خواهیم کرد که اطلاعات نگاشت را باید از کدام کلاس دریافت کند. تا اینجای کار شیء SessionSource را با معرفی نوع دیتابیس و همچنین محل دریافت اطلاعات نگاشت اشیاء معرفی کردیم. در دو سطر بعدی متد SetupContext کلاس FixtureBase ، ابتدا یک سشن را از این منبع سشن تهیه می‌کنیم. شیء منبع سشن در این فریم ورک در حقیقت یک factory object است (الگوهای طراحی برنامه نویسی شیء‌گرا) که امکان دسترسی به انواع و اقسام

دیتابیس‌ها را فراهم می‌سازد. برای مثال اگر روزی نیاز بود از دیتابیس اس کیوال سرور استفاده شود، می‌توان از کلاس MsSqlConfiguration بجای SQLiteConfiguration استفاده کرد و همینطور الی آخر. در ادامه توسط شیء SessionSource کار ساخت database schema را نیز به صورت پویا انجام خواهیم داد. بله، همانطور که متوجه شده‌اید، کار ساخت database schema نیز به صورت پویا توسط فریم ورک NHibernate با توجه به اطلاعات کلاس‌های نگاشت، صورت خواهد گرفت.

این مراحل، نحوه ایجاد و برپایی یک آزمایشگاه آزمون واحد فریم ورک Fluent NHibernate را مشخص ساخته و در پروژه‌های شما می‌توانند به کرات مورد استفاده قرار گیرند.

در ادامه اگر آزمون واحد را اجرا نمائیم (متد can_correctly_map_customer در کلاس CustomerMapping_Fixture)، نتیجه باید شبیه به شکل زیر باشد:



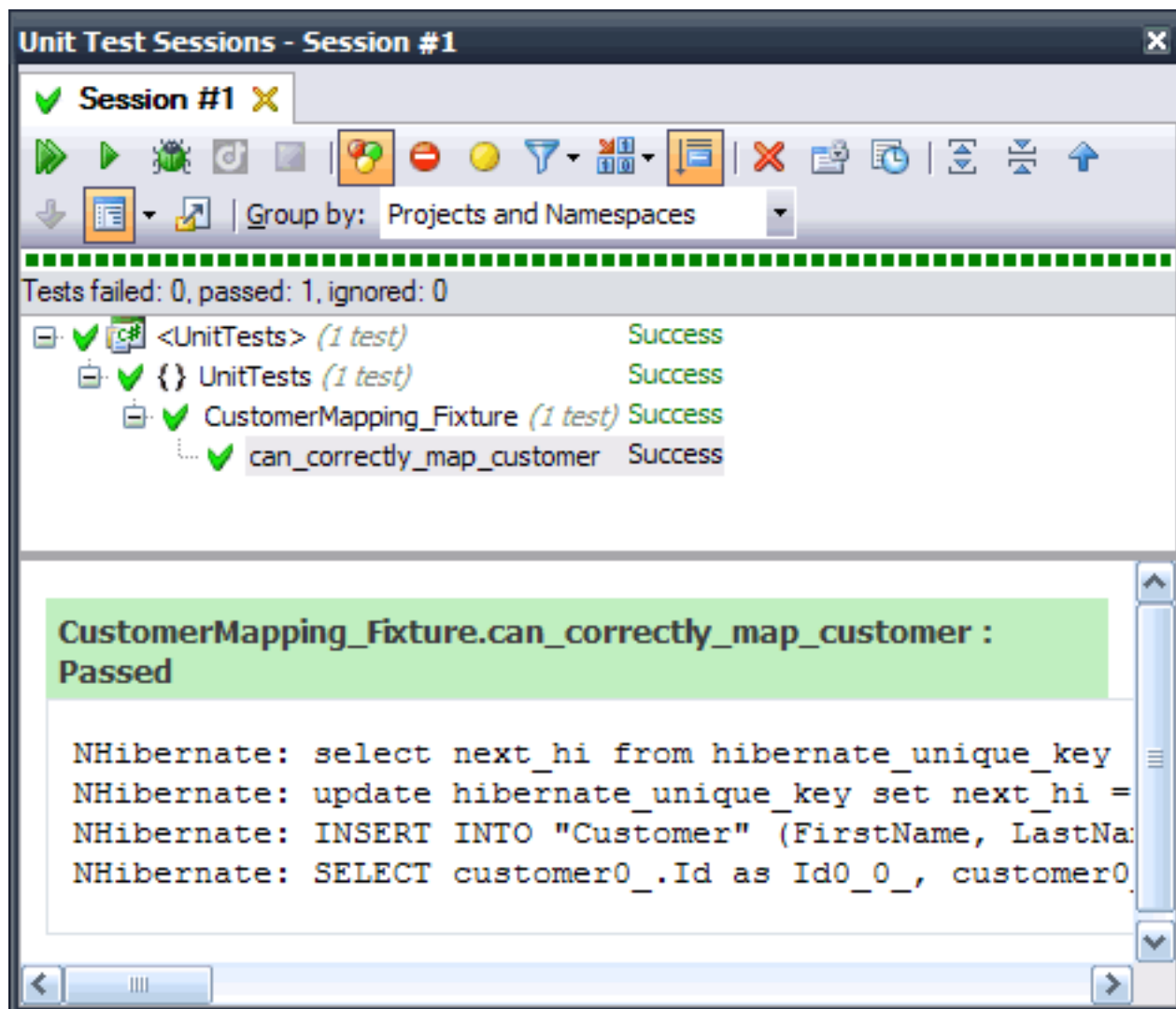
توسط متد CheckProperty کلاس PersistenceSpecification، امکان بررسی نگاشت تهیه شده میسر است. اولین پارامتر آن، یک lambda expression خاصیت مورد نظر جهت بررسی است و دومین آرگومان آن، مقداری است که در حین آزمون به خاصیت تعریف شده انتساب داده می‌شود.

نکته:

شاید سؤال بپرسید که در تابع can_correctly_map_customer عملاً چه اتفاقاتی رخ داده است؟ برای بررسی آن در متد SetupContext کلاس FixtureBase، اولین سطر آن را به صورت زیر تغییر دهید تا عبارات SQL نهایی تولید شده را نیز بتوانیم در

حين عمليات تست مشاهده نمائيم:

```
var cfg = Fluently.Configure().Database(SQLiteConfiguration.Standard.ShowSql().InMemory);
```



مطابق متد تست فوق، عبارات تولید شده به شرح زیر هستند:

```
NHibernate: select next_hi from hibernate_unique_key
NHibernate: update hibernate_unique_key set next_hi = @p0 where next_hi = @p1;@p0 = 2, @p1 = 1
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 = 1001
NHibernate: SELECT customer0_.Id as Id0_0_, customer0_.FirstName as FirstName0_0_, customer0_.LastName as LastName0_0_, customer0_.AddressLine1 as AddressL4_0_0_, customer0_.AddressLine2 as AddressL5_0_0_, customer0_.PostalCode as PostalCode0_0_, customer0_.City as City0_0_, customer0_.CountryCode as CountryC8_0_0_ FROM "Customer" customer0_ WHERE customer0_.Id=@p0;@p0 = 1001
```

نکته جالب این عبارات، استفاده از کوئری‌های پارامتری است به صورت پیش فرض که در نهایت سبب بالا رفتن امنیت بیشتر برنامه (یکی از راه‌های جلوگیری از تزریق اس کیوال در ADO.Net که در نهایت توسط تمامی این فریم ورک‌ها در پشت صحنه مورد

استفاده قرار خواهند گرفت) و همچنین سبب بکار افتادن سیستم‌های کش دیتابیس‌های پیشرفته مانند اس کیوال سرور می‌شوند (execution plan کوثری‌های پارامتری در اس کیوال سرور جهت بالا رفتن کارایی سیستم کش می‌شوند و اهمیتی هم ندارد که حتما رویه ذخیره شده باشند یا خیر).

ادامه دارد ...

نظرات خوانندگان

نویسنده: نیما

تاریخ: ۱۳۸۸/۰۷/۱۸ ۱۹:۳۶:۰۱

سلام استاد نصیری
با تشکر از بزرگواری جنابعالی در شریک کردن ما در تجربیات و دانسته هاتون در مورد قسمت دوم سوال داشتم: اینکه این سه تا کلاس رو تو داکيومنتهای NHibernate گفته شده که برای آزمایش واحد باید ایجاد کرد؟ امکانش هست مرحله به مرحله به زبان ساده تر بفرمایید وقتی آزمایش واحد شروع میشه چه اتفاقی میفته در اینجا؟
ممنون از شما استاد گرامی

نویسنده: وحید نصیری

تاریخ: ۱۳۸۸/۰۷/۱۸ ۱۹:۵۳:۱۶

سلام
- ضمن تشکر از لطف شما، بنده استاد نیستم. یک سری مطلب رو از این طرف اون طرف پیدا می‌کنم و با هم تقسیم می‌کنیم. فقط همین و لطفا این لفظ رو دیگر بکار نبرید.
- خیر. می‌شد برای آزمایش یک برنامه کنسول هم نوشت. اما دیگر مرسوم نیست. بجای استفاده از یک برنامه کنسول، آزمایش واحد بنویسید. هم روشی است استاندارد، هم به عنوان مستندات نحوه استفاده از متدهای پروژه می‌تونه مورد استفاده قرار بگیره، هم سبب میشه کد بهتری بنویسید چون مجبور خواهید شد در هم تنیدگی کدهای خودتون رو برای متد تست نوشتن کمتر کنید و هم در مقالات مربوطه (تگ unit test سمت راست صفحه) مابقی مزایا، نحوه تولید استفاده و غیره را لطفا مطالعه کنید.

نویسنده: Ahmadreza

تاریخ: ۱۳۸۸/۰۸/۲۱ ۲۱:۰۱:۵۸

سلام

آقای نصیری عزیز

من هم مثل شما از Resharper به عنوان اجرا کننده های تست های NUnit استفاده می کنم ولی من در تست های Pass شده مثل شما Log مربوط به NHibernate رو نمی بینم فقط تست های fail رو می بینم. مخواستم ببینم تنظیم خواصی داره؟ اگه لطفا کنید راهنمایی کنید ممنون میشم.

نویسنده: وحید نصیری

تاریخ: ۱۳۸۸/۰۸/۲۱ ۲۱:۱۲:۰۷

سلام

نکته نمایش خروجی SQL فوق، افزودن متد ShowSql است (نمونه آن در کدهای مقاله هست).

در ادامه، تعاریف سایر موجودیت‌های سیستم ثبت سفارشات و نگاشت آن‌ها را بررسی خواهیم کرد.

کلاس Product تعریف شده در فایل جدید Product.cs در پوشه domain برنامه:

```
namespace NHSample1.Domain
{
    public class Product
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public decimal UnitPrice { get; set; }
        public bool Discontinued { get; set; }
    }
}
```

کلاس ProductMapping تعریف شده در فایل جدید ProductMapping.cs (توصیه شده است که به ازای هر کلاس یک فایل جداگانه در نظر گرفته شود)، در پوشه Mappings برنامه:

```
using FluentNHibernate.Mapping;
using NHSample1.Domain;

namespace NHSample1.Mappings
{
    public class ProductMapping : ClassMap<Product>
    {
        public ProductMapping()
        {
            Not.LazyLoad();
            Id(p => p.Id).GeneratedBy.HiLo("1000");
            Map(p => p.Name).Length(50).Not.Nullable();
            Map(p => p.UnitPrice).Not.Nullable();
            Map(p => p.Discontinued).Not.Nullable();
        }
    }
}
```

همانطور که ملاحظه می‌کنید، روش تعریف آن‌ها همانند شیء Customer است که در قسمت‌های قبل بررسی شد و نکته جدیدی ندارد.

آزمون واحد بررسی این نگاشت نیز همانند مثال قبلی است.

کلاس ProductMapping_Fixture را در فایل جدید ProductMapping_Fixture.cs به پروژه UnitTests خود (که ارجاعات آن را در قسمت قبل مشخص کردیم) خواهیم افزود:

```
using NUnit.Framework;
using FluentNHibernate.Testing;
using NHSample1.Domain;

namespace UnitTests
{
    [TestFixture]
    public class ProductMapping_Fixture : FixtureBase
    {
        [Test]
        public void can_correctly_map_product()
        {
            new PersistenceSpecification<Product>(Session)
                .CheckProperty(p => p.Id, 1001)
                .CheckProperty(p => p.Name, "Apples")
                .CheckProperty(p => p.UnitPrice, 10.45m)
        }
    }
}
```

```

        .CheckProperty(p => p.Discontinued, true)
        .VerifyTheMappings();
    }
}

```

و پس از اجرای این آزمون واحد، عبارات SQL ایی که به صورت خودکار توسط این ORM جهت بررسی عملیات نگاشت صورت خواهند گرفت به صورت زیر می‌باشند:

```

ProductMapping_Fixture.can_correctly_map_product : Passed
NHibernate: select next_hi from hibernate_unique_key
NHibernate: update hibernate_unique_key set next_hi = @p1;@p0 = 2, @p1 = 1
NHibernate: INSERT INTO "Product" (Name, UnitPrice, Discontinued, Id) VALUES (@p0, @p1, @p2, @p3);@p0 =
'Apples', @p1 = 10.45, @p2 = True, @p3 = 1001
NHibernate: SELECT product0_.Id as Id1_0_, product0_.Name as Name1_0_, product0_.UnitPrice as
UnitPrice1_0_, product0_.Discontinued as Disconti4_1_0_ FROM "Product" product0_ WHERE
product0_.Id=@p0;@p0 = 1001

```

در ادامه تعریف کلاس کارمند، نگاشت و آزمون واحد آن به صورت زیر خواهند بود:

```

using System;
namespace NHSample1.Domain
{
    public class Employee
    {
        public int Id { set; get; }
        public string LastName { get; set; }
        public string FirstName { get; set; }
    }
}

```

```

using NHSample1.Domain;
using FluentNHibernate.Mapping;

namespace NHSample1.Mappings
{
    public class EmployeeMapping : ClassMap<Employee>
    {
        public EmployeeMapping()
        {
            Not.LazyLoad();
            Id(e => e.Id).GeneratedBy.Assigned();
            Map(e => e.LastName).Length(50);
            Map(e => e.FirstName).Length(50);
        }
    }
}

```

```

using NUnit.Framework;
using NHSample1.Domain;
using FluentNHibernate.Testing;

namespace UnitTests
{
    [TestFixture]
    public class EmployeeMapping_Fixture : FixtureBase
    {
        [Test]
        public void can_correctly_map_employee()
        {
            new PersistenceSpecification<Employee>(Session)
                .CheckProperty(p => p.Id, 1001)
                .CheckProperty(p => p.FirstName, "name1")
                .CheckProperty(p => p.LastName, "lname1")
                .VerifyTheMappings();
        }
    }
}

```

```
}
```

خروجی SQL حاصل از موفقیت آزمون واحد آن:

```
NHibernate: select next_hi from hibernate_unique_key
NHibernate: update hibernate_unique_key set next_hi = @p0 where next_hi = @p1;@p0 = 2, @p1 = 1
NHibernate: INSERT INTO "Employee" (LastName, FirstName, Id) VALUES (@p0, @p1, @p2);@p0 = 'lname1', @p1 = 'name1', @p2 = 1001
NHibernate: SELECT employee0_.Id as Id4_0_, employee0_.LastName as LastName4_0_, employee0_.FirstName as FirstName4_0_ FROM "Employee" employee0_ WHERE employee0_.Id=@p0;@p0 = 1001
```

همانطور که ملاحظه می‌کنید، این آزمون‌های واحد 4 مرحله را در یک سطر انجام می‌دهند:

(الف) ایجاد یک وهله از کلاس Employee

(ب) ثبت اطلاعات کارمند در دیتابیس

(ج) دریافت اطلاعات کارمند در وهله‌ای جدید از شیء Employee

(د) و در پایان بررسی می‌کند که آیا شیء جدید ایجاد شده با شیء اولیه مطابقت دارد یا خیر

اکنون در ادامه پیاده سازی سیستم ثبت سفارشات، به قسمت جالب این مدل می‌رسیم. قسمتی که در آن ارتباطات اشیاء و روابط one-to-many تعریف خواهند شد. تعاریف کلاس‌های OrderItem و OrderItemMapping را به صورت زیر در نظر بگیرید:

کلاس OrderItem تعریف شده در فایل جدید OrderItem.cs واقع شده در پوشه domain پروژه:

که در آن هر سفارش (order) دقیقاً از یک محصول (product) تشکیل می‌شود و هر محصول می‌تواند در سفارشات متعدد و مختلفی درخواست شود.

```
namespace NHSample1.Domain
{
    public class OrderItem
    {
        public int Id { get; set; }
        public int Quantity { get; set; }
        public Product Product { get; set; }
    }
}
```

کلاس OrderItemMapping تعریف شده در فایل جدید OrderItemMapping.cs:

```
using FluentNHibernate.Mapping;
using NHSample1.Domain;

namespace NHSample1.Mappings
{
    public class OrderItemMapping : ClassMap<OrderItem>
    {
        public OrderItemMapping()
        {
            Not.LazyLoad();
            Id(oi => oi.Id).GeneratedBy.Assigned();
            Map(oi => oi.Quantity).Not.Nullable();
            References(oi => oi.Product).Not.Nullable();
        }
    }
}
```

نکته جدیدی که در این کلاس نگاشت مطرح شده است، واژه کلیدی References می‌باشد که جهت بیان این ارجاعات و وابستگی‌ها بکار می‌رود. این ارجاع بیانگر یک رابطه many-to-one بین سفارشات و محصولات است. همچنین در ادامه آن Not.Nullable ذکر شده است تا این ارجاع را اجباری نمائید (در غیر اینصورت سفارش غیر معتبر خواهد بود).

نکته‌ی دیگر مهم آن این مورد است که Id در اینجا به صورت یک کلید تعریف نشده است. یک آیتم سفارش داده شده، موجودیت

به حساب نیامده و فقط یک شیء مقداری ([value object](#)) است و به خودی خود امکان وجود ندارد. هر وهله از آن تنها توسط یک سفارش قابل تعریف است. بنابراین id در اینجا فقط به عنوان یک index می‌تواند مورد استفاده قرار گیرد و فقط توسط شیء Order زمانیکه یک OrderItem به آن اضافه می‌شود، مقدار دهی خواهد شد.

اگر برای این نگاشت نیز آزمون واحد تهیه کنیم، به صورت زیر خواهد بود:

```
using NUnit.Framework;
using NHSample1.Domain;
using FluentNHibernate.Testing;

namespace UnitTests
{
    [TestFixture]
    public class OrderItemMapping_Fixture : FixtureBase
    {
        [Test]
        public void can_correctly_map_order_item()
        {
            var product = new Product
            {
                Name = "Apples",
                UnitPrice = 4.5m,
                Discontinued = true
            };

            new PersistenceSpecification<OrderItem>(Session)
                .CheckProperty(p => p.Id, 1)
                .CheckProperty(p => p.Quantity, 5)
                .CheckReference(p => p.Product, product)
                .VerifyTheMappings();
        }
    }
}
```

مشکل! این آزمون واحد با شکست مواجه خواهد شد، زیرا هنوز مشخص نکرده‌ایم که دو شیء Product را که در قسمت CheckReference فوق برای این منظور معرفی کرده‌ایم، چگونه باید با هم مقایسه کرد. در مورد مقایسه نوع‌های اولیه و اصلی مانند int و string و امثال آن مشکلی نیست، اما باید منطق مقایسه سایر اشیاء سفارشی خود را با پیاده سازی اینترفیس IEqualityComparer دقیقاً مشخص سازیم:

```
using System.Collections;
using NHSample1.Domain;

namespace UnitTests
{
    public class CustomEqualityComparer : IEqualityComparer
    {
        public bool Equals(object x, object y)
        {
            if (ReferenceEquals(x, y)) return true;
            if (x == null || y == null) return false;

            if (x is Product && y is Product)
                return (x as Product).Id == (y as Product).Id;

            if (x is Customer && y is Customer)
                return (x as Customer).Id == (y as Customer).Id;

            if (x is Employee && y is Employee)
                return (x as Employee).Id == (y as Employee).Id;

            if (x is OrderItem && y is OrderItem)
                return (x as OrderItem).Id == (y as OrderItem).Id;

            return x.Equals(y);
        }

        public int GetHashCode(object obj)
        {
            // شاید وقتی دیگر//
        }
    }
}
```



```

        return obj.GetHashCode();
    }
}

```

در اینجا فقط Id این اشیاء با هم مقایسه شده است. در صورت نیاز تمامی خاصیت‌های این اشیاء را نیز می‌توان با هم مقایسه کرد (یک سری از اشیاء بکار گرفته شده در این کلاس در ادامه بحث معرفی خواهند شد).

سپس برای بکار گیری این کلاس جدید، سطر مربوط به استفاده از PersistenceSpecification به صورت زیر تغییر خواهد کرد:

```
new PersistenceSpecification<OrderItem>(Session, new CustomEqualityComparer())
```

پس از این تغییرات و مشخص سازی نحوه‌ی مقایسه دو شیء سفارشی، آزمون واحد ما پاس شده و خروجی SQL تولید شده آن به صورت زیر می‌باشد:

```

NHibernate: select next_hi from hibernate_unique_key
NHibernate: update hibernate_unique_key set next_hi = @p0 where next_hi = @p1;@p0 = 2, @p1 = 1
NHibernate: INSERT INTO "Product" (Name, UnitPrice, Discontinued, Id) VALUES (@p0, @p1, @p2, @p3);@p0 =
'Apples', @p1 = 4.5, @p2 = True, @p3 = 1001
NHibernate: INSERT INTO "OrderItem" (Quantity, Product_id, Id) VALUES (@p0, @p1, @p2);@p0 = 5, @p1 =
1001, @p2 = 1
NHibernate: SELECT orderitem0_.Id as Id0_1_, orderitem0_.Quantity as Quantity0_1_,
orderitem0_.Product_id as Product3_0_1_, product1_.Id as Id3_0_, product1_.Name as Name3_0_,
product1_.UnitPrice as UnitPrice3_0_, product1_.Discontinued as Disconti4_3_0_ FROM "OrderItem"
orderitem0_ inner join "Product" product1_ on orderitem0_.Product_id=product1_.Id WHERE
orderitem0_.Id=@p0;@p0 = 1

```

قسمت پایانی کار تعاریف کلاس‌های نگاشت، مربوط به کلاس Order است که در ادامه بررسی خواهد شد.

```

using System;
using System.Collections.Generic;

namespace NHSample1.Domain
{
    public class Order
    {
        public int Id { get; set; }
        public DateTime OrderDate { get; set; }
        public Employee Employee { get; set; }
        public Customer Customer { get; set; }
        public IList<OrderItem> OrderItems { get; set; }
    }
}

```

نکته‌ی مهمی که در این کلاس وجود دارد استفاده از IList جهت معرفی مجموعه‌ای از آیتم‌های سفارشی است (بجای List و یا IEnumerable که در صورت استفاده خطای type cast exception در حین نگاشت حاصل می‌شد).

```

using NHSample1.Domain;
using FluentNHibernate.Mapping;

namespace NHSample1.Mappings
{
    public class OrderMapping : ClassMap<Order>
    {
        public OrderMapping()
        {
            Not.LazyLoad();
            Id(o => o.Id).GeneratedBy.GuidComb();
            Map(o => o.OrderDate).Not.Nullable();
            References(o => o.Employee).Not.Nullable();
            References(o => o.Customer).Not.Nullable();
            HasMany(o => o.OrderItems)
                .AsList(index => index.Column("ListIndex").Type<int>());
        }
    }
}

```

```
}
}
```

در تعاریف نگاشت این کلاس نیز دو ارجاع به اشیاء کارمند و مشتری وجود دارد که با References مشخص شده‌اند. قسمت جدید آن HasMany است که جهت تعریف رابطه one-to-many بکار گرفته شده است. یک سفارش رابطه many-to-one با یک مشتری و همچنین کارمندی که این رکورد را ثبت می‌کند، دارد. در اینجا مجموعه آیتم‌های یک سفارش به صورت یک لیست بازگشت داده می‌شود و ایندکس آن به ستونی به نام ListIndex در یک جدول دیتابیس نگاشت خواهد شد. نوع این ستون، int می‌باشد.

```
using System;
using System.Collections.Generic;
using NUnit.Framework;
using NHibernate.Domain;
using FluentNHibernate.Testing;

namespace UnitTests
{
    [TestFixture]
    public class OrderMapping_Fixture : FixtureBase
    {
        [Test]
        public void can_correctly_map_an_order()
        {
            {
                var product1 =
                    new Product
                    {
                        Name = "Apples",
                        UnitPrice = 4.5m,
                        Discontinued = true
                    };
                var product2 =
                    new Product
                    {
                        Name = "Pears",
                        UnitPrice = 3.5m,
                        Discontinued = false
                    };

                Session.Save(product1);
                Session.Save(product2);

                var items = new List<OrderItem>
                {
                    new OrderItem
                    {
                        Id = 1,
                        Quantity = 100,
                        Product = product1
                    },
                    new OrderItem
                    {
                        Id = 2,
                        Quantity = 200,
                        Product = product2
                    }
                };

                var customer = new Customer
                {
                    FirstName = "Vahid",
                    LastName = "Nasiri",
                    AddressLine1 = "Addr1",
                    AddressLine2 = "Addr2",
                    PostalCode = "1234",
                    City = "Tehran",
                    CountryCode = "IR"
                };

                var employee =
                    new Employee
                    {
                        FirstName = "name1",
                        LastName = "lname1"
                    };
            }
        }
    }
}
```

```

var order = new Order
{
    Customer = customer,
    Employee = employee,
    OrderDate = DateTime.Today,
    OrderItems = items
};

new PersistenceSpecification<Order>(Session, new CustomEqualityComparer())
    .CheckProperty(o => o.OrderDate, order.OrderDate)
    .CheckReference(o => o.Customer, order.Customer)
    .CheckReference(o => o.Employee, order.Employee)
    .CheckList(o => o.OrderItems, order.OrderItems)
    .VerifyTheMappings();
}
}
}
}
}

```

همانطور که ملاحظه می‌کنید در این متد آزمون واحد، نیاز به مشخص سازی منطق مقایسه اشیاء سفارش، مشتری و آیتم‌های سفارش داده شده نیز وجود دارد که پیشتر در کلاس CustomEqualityComparer معرفی شدند؛ در غیر این صورت این آزمون واحد با شکست مواجه می‌شد.

متد آزمون واحد فوق کمی طولانی است؛ زیرا در آن باید تعاریف انواع و اقسام اشیاء مورد استفاده را مشخص نمود (و ارزش کار نیز دقیقاً در همینجا مشخص می‌شود که بجای SQL نوشتن، با اشیایی که توسط کامپایلر تحت نظر هستند سر و کار داریم). تنها نکته جدید آن استفاده از CheckList برای بررسی IList تعریف شده در قسمت قبل است.

خروجی SQL این آزمون واحد پس از اجرا و موفقیت آن به صورت زیر است:

```

OrderMapping_Fixture.can_correctly_map_an_order : Passed
NHibernate: select next_hi from hibernate_unique_key
NHibernate: update hibernate_unique_key set next_hi = @p0 where next_hi = @p1;@p0 = 2, @p1 = 1
NHibernate: select next_hi from hibernate_unique_key
NHibernate: update hibernate_unique_key set next_hi = @p0 where next_hi = @p1;@p0 = 3, @p1 = 2
NHibernate: INSERT INTO "Product" (Name, UnitPrice, Discontinued, Id) VALUES (@p0, @p1, @p2, @p3);@p0 =
'Apples', @p1 = 4.5, @p2 = True, @p3 = 1001
NHibernate: INSERT INTO "Product" (Name, UnitPrice, Discontinued, Id) VALUES (@p0, @p1, @p2, @p3);@p0 =
'Pears', @p1 = 3.5, @p2 = False, @p3 = 1002
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 = 2002
NHibernate: select next_hi from hibernate_unique_key
NHibernate: update hibernate_unique_key set next_hi = @p0 where next_hi = @p1;@p0 = 4, @p1 = 3
NHibernate: INSERT INTO "Employee" (LastName, FirstName, Id) VALUES (@p0, @p1, @p2);@p0 = 'lname1', @p1
= 'name1', @p2 = 3003
NHibernate: INSERT INTO "OrderItem" (Quantity, Product_id, Id) VALUES (@p0, @p1, @p2);@p0 = 100, @p1 =
1001, @p2 = 1
NHibernate: INSERT INTO "OrderItem" (Quantity, Product_id, Id) VALUES (@p0, @p1, @p2);@p0 = 200, @p1 =
1002, @p2 = 2
NHibernate: INSERT INTO "Order" (OrderDate, Employee_id, Customer_id, Id) VALUES (@p0, @p1, @p2,
@p3);@p0 = 2009/10/10 12:00:00 ق.ظ, @p1 = 3003, @p2 = 2002, @p3 = 0
NHibernate: UPDATE "OrderItem" SET Order_id = @p0, ListIndex = @p1 WHERE Id = @p2;@p0 = 0, @p1 = 0, @p2
= 1
NHibernate: UPDATE "OrderItem" SET Order_id = @p0, ListIndex = @p1 WHERE Id = @p2;@p0 = 0, @p1 = 1, @p2
= 2
NHibernate: SELECT order0_.Id as Id1_2_, order0_.OrderDate as OrderDate1_2_, order0_.Employee_id as
Employee3_1_2_, order0_.Customer_id as Customer4_1_2_, employee1_.Id as Id4_0_, employee1_.LastName as
LastName4_0_, employee1_.FirstName as FirstName4_0_, customer2_.Id as Id2_1_, customer2_.FirstName as
FirstName2_1_, customer2_.LastName as LastName2_1_, customer2_.AddressLine1 as AddressL4_2_1_,
customer2_.AddressLine2 as AddressL5_2_1_, customer2_.PostalCode as PostalCode2_1_, customer2_.City as
City2_1_, customer2_.CountryCode as CountryC8_2_1_ FROM "Order" order0_ inner join "Employee"
employee1_ on order0_.Employee_id=employee1_.Id inner join "Customer" customer2_ on
order0_.Customer_id=customer2_.Id WHERE order0_.Id=@p0;@p0 = 0
NHibernate: SELECT orderitems0_.Order_id as Order4_2_, orderitems0_.Id as Id2_, orderitems0_.ListIndex
as ListIndex2_, orderitems0_.Id as Id0_1_, orderitems0_.Quantity as Quantity0_1_,
orderitems0_.Product_id as Product3_0_1_, product1_.Id as Id3_0_, product1_.Name as Name3_0_,
product1_.UnitPrice as UnitPrice3_0_, product1_.Discontinued as Disconti4_3_0_ FROM "OrderItem"
orderitems0_ inner join "Product" product1_ on orderitems0_.Product_id=product1_.Id WHERE
orderitems0_.Order_id=@p0;@p0 = 0

```

تا اینجای کار تعاریف اشیاء ، نگاشت آن‌ها و همچنین بررسی صحت این نگاشت‌ها به پایان می‌رسد.

نکته:

دیتابیس برنامه را جهت آزمون‌های واحد برنامه، از نوع SQLite ساخته شده در حافظه مشخص کردیم. اگر علاقمند باشید که database schema تولید شده توسط NHibernate را مشاهده نمایید، در متد SetupContext کلاس FixtureBase که در قسمت قبل معرفی شد، سطر آخر را به صورت زیر تغییر دهید، تا اسکریپت دیتابیس نیز به صورت خودکار در خروجی اس کیوال آزمون واحد لحاظ شود (پارامتر دوم آن مشخص می‌کند که schema ساخته شده، نمایش داده شود یا خیر):

```
SessionSource.BuildSchema(Session, true);
```

پس از این تغییر و انجام مجدد آزمون واحد، اسکریپت دیتابیس ما به صورت زیر خواهد بود (که جهت ایجاد یک دیتابیس SQLite می‌تواند مورد استفاده قرار گیرد):

```
drop table if exists "OrderItem"
drop table if exists "Order"
drop table if exists "Customer"
drop table if exists "Product"
drop table if exists "Employee"
drop table if exists hibernate_unique_key

create table "OrderItem" (
    Id INTEGER not null,
    Quantity INTEGER not null,
    Product_id INTEGER not null,
    Order_id INTEGER,
    ListIndex INTEGER,
    primary key (Id)
)

create table "Order" (
    Id INTEGER not null,
    OrderDate DATETIME not null,
    Employee_id INTEGER not null,
    Customer_id INTEGER not null,
    primary key (Id)
)

create table "Customer" (
    Id INTEGER not null,
    FirstName TEXT not null,
    LastName TEXT not null,
    AddressLine1 TEXT not null,
    AddressLine2 TEXT,
    PostalCode TEXT not null,
    City TEXT not null,
    CountryCode TEXT not null,
    primary key (Id)
)

create table "Product" (
    Id INTEGER not null,
    Name TEXT not null,
    UnitPrice NUMERIC not null,
    Discontinued INTEGER not null,
    primary key (Id)
)

create table "Employee" (
    Id INTEGER not null,
    LastName TEXT,
    FirstName TEXT,
    primary key (Id)
)
```

```
create table hibernate_unique_key (
    next_hi INTEGER
)
```

البته اگر مستندات SQLite را مطالعه کرده باشید می‌دانید که مفهوم کلید خارجی در این دیتابیس وجود دارد اما اعمال نمی‌شود! (برای اعمال آن باید تریگر نوشت) به همین جهت در این اسکریپت تولیدی خبری از کلید خارجی نیست.

برای اینکه از دیتابیس اس کیوال سرور استفاده کنیم، در همان متد SetupContext کلاس مذکور، سطر اول را به صورت زیر تغییر دهید (نوع دیتابیس اس کیوال سرور 2008 مشخص شده و سپس رشته اتصالی به دیتابیس ذکر گردیده است):

```
var cfg = Fluently.Configure().Database(
    // SQLiteConfiguration.Standard.ShowSql().InMemory
    MsSqlConfiguration
    .MsSql2008
    .ShowSql()
    .ConnectionString("Data Source=(local);Initial Catalog=testdb2009;Integrated Security =
true")
);
```

اکنون اگر مجدداً آزمون واحد را اجرا نمائیم، اسکریپت تولیدی به صورت زیر خواهد بود (در اینجا مفهوم استقلال برنامه از نوع دیتابیس را به خوبی می‌توان درک کرد):

```
if exists (select 1 from sys.objects where object_id = OBJECT_ID(N'[FK3EF88858466CFBF7]') AND
parent_object_id = OBJECT_ID('[OrderItem]'))
alter table [OrderItem] drop constraint FK3EF88858466CFBF7

if exists (select 1 from sys.objects where object_id = OBJECT_ID(N'[FK3EF888589F32DE52]') AND
parent_object_id = OBJECT_ID('[OrderItem]'))
alter table [OrderItem] drop constraint FK3EF888589F32DE52

if exists (select 1 from sys.objects where object_id = OBJECT_ID(N'[FK3117099B1EBA72BC]') AND
parent_object_id = OBJECT_ID('[Order]'))
alter table [Order] drop constraint FK3117099B1EBA72BC

if exists (select 1 from sys.objects where object_id = OBJECT_ID(N'[FK3117099BB2F9593A]') AND
parent_object_id = OBJECT_ID('[Order]'))
alter table [Order] drop constraint FK3117099BB2F9593A

if exists (select * from dbo.sysobjects where id = object_id(N'[OrderItem]') and OBJECTPROPERTY(id,
N'IsUserTable') = 1) drop table [OrderItem]

if exists (select * from dbo.sysobjects where id = object_id(N'[Order]') and OBJECTPROPERTY(id,
N'IsUserTable') = 1) drop table [Order]

if exists (select * from dbo.sysobjects where id = object_id(N'[Customer]') and OBJECTPROPERTY(id,
N'IsUserTable') = 1) drop table [Customer]

if exists (select * from dbo.sysobjects where id = object_id(N'[Product]') and OBJECTPROPERTY(id,
N'IsUserTable') = 1) drop table [Product]

if exists (select * from dbo.sysobjects where id = object_id(N'[Employee]') and OBJECTPROPERTY(id,
N'IsUserTable') = 1) drop table [Employee]

if exists (select * from dbo.sysobjects where id = object_id(N'hibernate_unique_key') and
OBJECTPROPERTY(id, N'IsUserTable') = 1) drop table hibernate_unique_key

create table [OrderItem] (
    Id INT not null,
    Quantity INT not null,
    Product_id INT not null,
    Order_id INT not null,
    ListIndex INT not null,
    primary key (Id)
)

create table [Order] (
    Id INT not null,
    OrderDate DATETIME not null,
```

```

        Employee_id INT not null,
        Customer_id INT not null,
        primary key (Id)
    )

create table [Customer] (
    Id INT not null,
    FirstName NVARCHAR(50) not null,
    LastName NVARCHAR(50) not null,
    AddressLine1 NVARCHAR(50) not null,
    AddressLine2 NVARCHAR(50) null,
    PostalCode NVARCHAR(10) not null,
    City NVARCHAR(50) not null,
    CountryCode NVARCHAR(2) not null,
    primary key (Id)
)

create table [Product] (
    Id INT not null,
    Name NVARCHAR(50) not null,
    UnitPrice DECIMAL(19,5) not null,
    Discontinued BIT not null,
    primary key (Id)
)

create table [Employee] (
    Id INT not null,
    LastName NVARCHAR(50) null,
    FirstName NVARCHAR(50) null,
    primary key (Id)
)

alter table [OrderItem]
    add constraint FK3EF88858466CFBF7
    foreign key (Product_id)
    references [Product]

alter table [OrderItem]
    add constraint FK3EF888589F32DE52
    foreign key (Order_id)
    references [Order]

alter table [Order]
    add constraint FK3117099B1EBA72BC
    foreign key (Employee_id)
    references [Employee]

alter table [Order]
    add constraint FK3117099BB2F9593A
    foreign key (Customer_id)
    references [Customer]

create table hibernate_unique_key (
    next_hi INT
)

```

که نکات ذیل در مورد آن جالب توجه است:

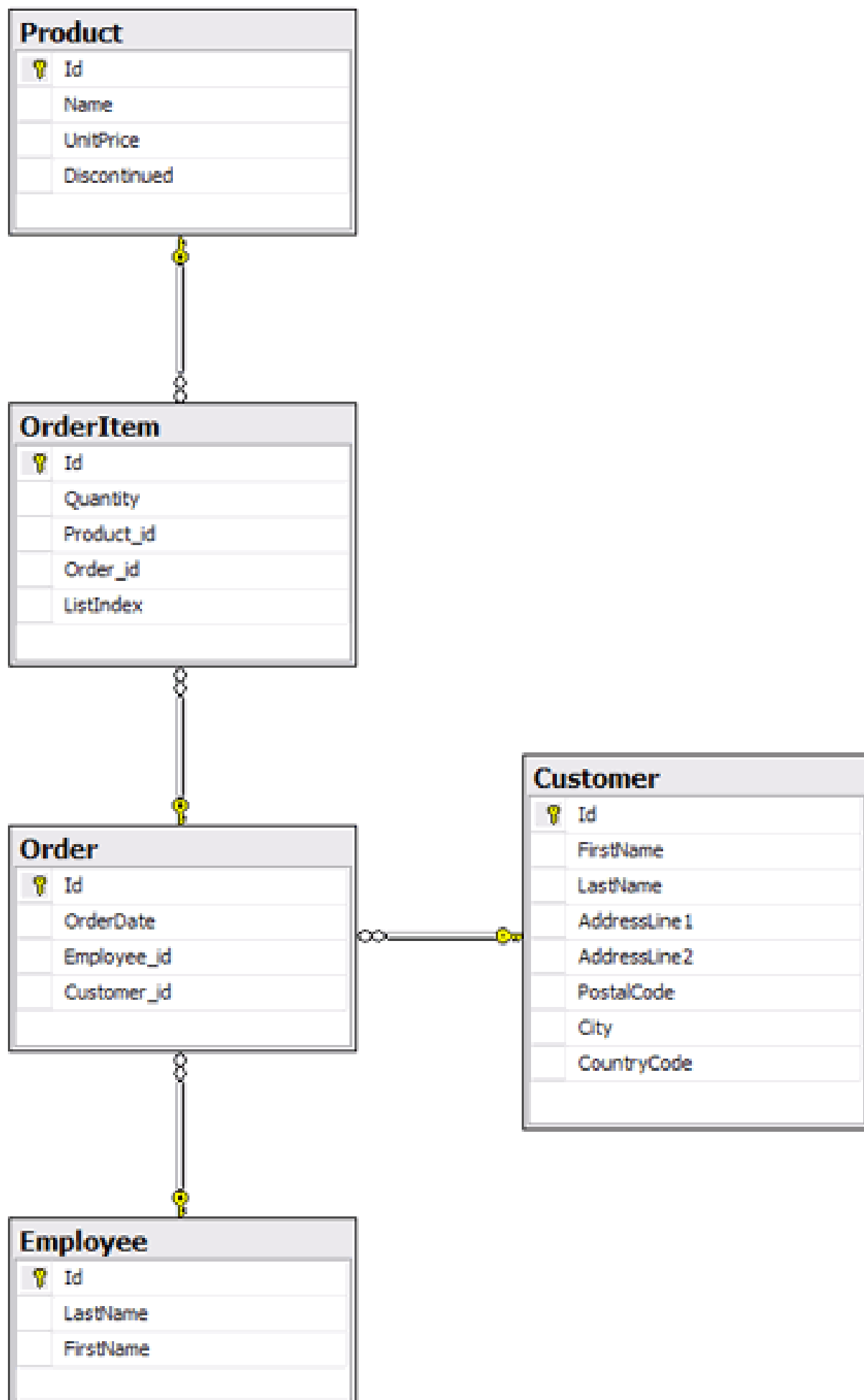
الف) جداول مطابق نام کلاس‌های ما تولید شده‌اند.

ب) نام فیلدها دقیقاً مطابق نام خواص کلاس‌های ما تشکیل شده‌اند.

ج) Id ها به صورت primary key تعریف شده‌اند (از آنجائیکه ما در هنگام تعریف نگاشت‌ها، آن‌ها را از نوع identity مشخص کرده بودیم).

د) رشته‌ها به نوع nvarchar با اندازه 50 نگاشت شده‌اند.

ه) کلیدهای خارجی بر اساس نام جدول با پسوند id_ تشکیل شده‌اند.



ادامه دارد ...

نظرات خوانندگان

نویسنده: DotNetCoders
تاریخ: ۱۳۸۸/۰۷/۱۹ ۱۲:۳۵:۳۱

عالیه جناب نصیری !

یه سوال بی ربط : توی جدول OrderItem فیلد ListIndex رو برای چی گذاشتید ؟

خسته نباشید...

نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۷/۱۹ ۱۲:۴۷:۱۸

سلام،

در یک order_id مشخص، میشه اسمش رو گذاشت شماره ردیف سند.

نویسنده: Ali
تاریخ: ۱۳۸۹/۰۹/۱۷ ۲۱:۳۳:۵۰

با سلام

من در NHibernate 2 از این کلاس استفاده می کردم:(نگاشت، مربوط به جدول tblAhkam است)

```
public class Ahkam
{
    { ;public virtual int Id { get; set
    { ;public virtual int HDate { get; set

    public virtual string SepratedDate
    }
    get
    }

    return Functions.SepratePersianDate(HDate);//Convert 890221 to 89/02/21
    {
    {
    {
```

و در لایه BLL تبدیل به Dataset می کردم و استفاده می شد و مشکلی هم وجود نداشت. اما وقتی با NHibernate 3 برنامه رو اجرا کردم به مشکل برخورد و فهمیدم چون فیلد SepratedDate در جدول بانک وجود ندارد باعث خطا شده است. راه حلی وجود دارد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۹/۱۷ ۲۲:۴۰:۳۹

سؤال شما مرتبط است به موضوع "nhibernate derived properties" (+) و برای بررسی مشکل شما نیاز به این موارد است:

- چگونه نگاشت‌ها را تعریف کرده‌اید. (نیاز به سورس است)
- دقیقا چه خطایی می‌گیرید. متن آن خیلی مهم است.

لطفاً از امکانات انجمن‌ها برای ادامه‌ی بحث استفاده کنید.

+ اگر از fluent NHibernate استفاده می‌کنید، نگارش سازگار با NHibernate 3 آن هنوز ارائه نشده (به زودی): [\(+\)](#)

نویسنده: fateme

تاریخ: ۱۳۸۹/۱۰/۲۰ ۱۲:۱۲:۴۱

سلام آقای نصیری

خیلی ممنون از آموزش بسیار عالی تون

من با اینکه در قسمت unit tests کلاس CustomEqualityComparer ساختم ولی و این کلاسو در orderItemMapping_Fixture

هم آوردم ولی باز error not-null property references a null or transient valueNHSample1.Domain.OrderItem.Product

رو میدید وقتی unit test رو اجرا می‌کنم

لطف می‌کنید راهنماییم کنید؟

نویسنده: وحید نصیری

تاریخ: ۱۳۸۹/۱۰/۲۰ ۱۲:۲۶:۴۵

این خطا زمانی حاصل میشه که شیءایی که خودش یک یا چند خواصش شیء دیگر هستند (ارجاعات به جداول دیگر)، به درستی مقدار دهی نشده و حداقل یکی از این موارد نال است.

نویسنده: fateme

تاریخ: ۱۳۸۹/۱۰/۲۰ ۱۵:۲۴:۵۷

ببخشید من تنونستم مشکلو حل کنم

دقیقاً کدی که شما واسه آموزش گذاشتیدو وارد کردم

ولی وارد کلاس CustomEqualityComparer نمی‌شه

نویسنده: وحید نصیری

تاریخ: ۱۳۸۹/۱۰/۲۰ ۱۸:۱۰:۱۶

شاید جایی رو از قلم انداخته‌اید (solution کامل شما باید برای دیباگ موجود باشد).
سورس کامل قابل دریافت این موارد در پایان قسمت چهارم ارائه شده. به آن مراجعه کنید.

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 آن را 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 دیتابیس 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();//بعد و هم
                    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();//بعد و هم چندین عملیات با هم
            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، دیتابیس 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 آن را 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، 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:

```
file:///I:/asp_net_works/wwwroot/1388/NHSample1/ConsoleTestApplication/bin/Debug/Con...
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;@p
0 = 16, @p1 = 15
NHibernate: INSERT INTO [Customer] (FirstName, LastName, AddressLine1, AddressLi
ne2, PostalCode, City, CountryCode, Id) VALUES (@p0, @p1, @p2, @p3, @p4, @p5, @p
6, @p7);@p0 = 'Vahid', @p1 = 'Nasiri', @p2 = 'Addr1', @p3 = 'Addr2', @p4 = '1234
', @p5 = 'Tehran', @p6 = 'IR', @p7 = 15015
```

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 حتماً ادמיینی برای این مسایل در نظر گرفته می‌شود). اکنون چه باید کرد؟ باید برنامه را از صفر بازنویسی کرد یا قسمت دسترسی به داده‌های آنرا کلاً مورد بازبینی قرار داد؟ اگر برنامه اسپاگتی ما اصلاً لایه دسترسی به داده‌ها را نداشت چه؟! همه جای برنامه پر است از SqlCommand و Open و Close ! و عملاً 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:

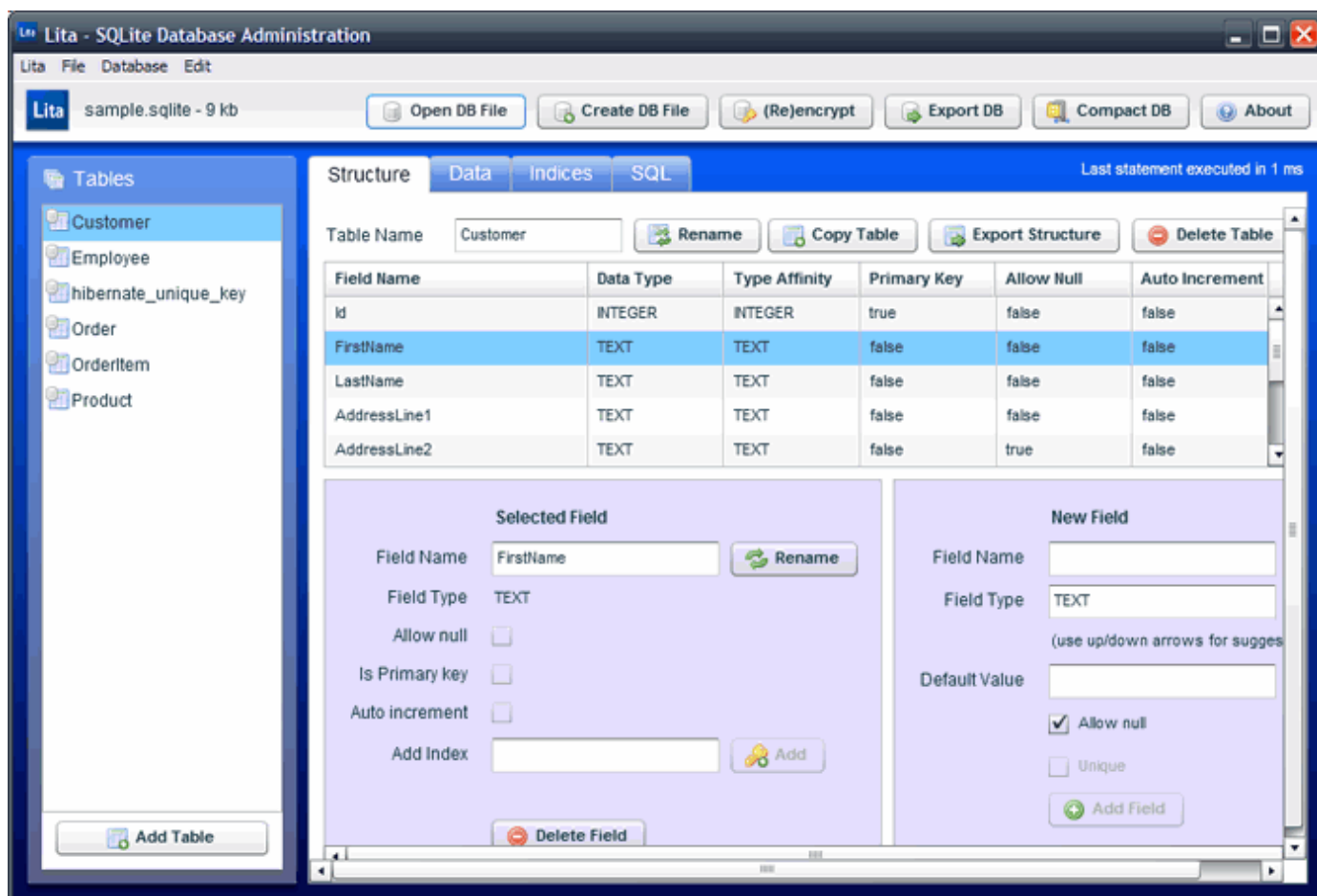
الف) ایجاد یک دیتابیس خام بر اساس کلاس‌های domain و mapping تعریف شده در قسمت‌های قبل به صورت خودکار

```
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 browser 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 می کنید.

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

سلام سری مقاله های جالب و مفیدی هستند و برای اینکه بنظم 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 می تونید 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 فایل های کمکی LINQ قدیمی ندارد و از یک کتابخانه ی دیگر و پیشرفته تر به صورت یکپارچه استفاده می کند. بنابراین نیازی نیست که از فایل LINQ موجود در (+) استفاده کرد. سایر موارد آن برای NH 3.0 به روز شده اند و قابل استفاده است.

عنوان: آشنایی با NHibernate - قسمت پنجم

نویسنده: وحید نصیری

تاریخ: ۱۳۸۸/۰۷/۲۱ ۱۰:۴۱:۰۰

آدرس: www.dotnettips.info

برچسب‌ها: NHibernate

استفاده از LINQ جهت انجام کوئری‌ها توسط NHibernate

نگارش نهایی 1.0 کتابخانه‌ی LINQ to NHibernate اخیراً (حدود سه ماه قبل) منتشر شده است. در این قسمت قصد داریم با کمک این کتابخانه، اعمال متداول انجام کوئری‌ها را بر روی دیتابیس قسمت قبل انجام دهیم. توسط این نگارش ارائه شده، کلیه اعمال قابل انجام با criteria API این فریم ورک را می‌توان از طریق LINQ نیز انجام داد (NHibernate برای کار با داده‌ها و جستجوهای پیشرفته بر روی آن‌ها، HQL : Hibernate Query Language و Criteria API را سال‌ها قبل توسعه داده است).

جهت دریافت پروایدر LINQ مخصوص NHibernate به آدرس زیر مراجعه نمایید:

<http://sourceforge.net/projects/nhibernate/files>

پس از دریافت آن، به همان برنامه کنسول قسمت قبل، دو ارجاع را باید افزود:

الف) ارجاعی به اسمبلی NHibernate.Linq.dll

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

در ابتدای متد Main برنامه قصد داریم تعدادی مشتری را به دیتابیس اضافه نمائیم. به همین منظور متد AddNewCustomers را به کلاس CDbOperations برنامه کنسول قسمت قبل اضافه نمائید. این متد لیستی از مشتری‌ها را دریافت کرده و آن‌ها را در طی یک تراکنش به دیتابیس اضافه می‌کند:

```
public void AddNewCustomers(params Customer[] customers)
{
    using (ISession session = _factory.OpenSession())
    {
        using (ITransaction transaction = session.BeginTransaction())
        {
            foreach (var data in customers)
                session.Save(data);

            session.Flush();

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

در اینجا استفاده از واژه کلیدی params سبب می‌شود که بجای تعریف الزامی یک آرایه از نوع مشتری‌ها، بتوانیم تعداد دلخواهی پارامتر از نوع مشتری را به این متد ارسال کنیم.

پس از افزودن این ارجاعات، کلاس جدیدی را به نام CLinqTest به برنامه کنسول اضافه نمائید. ساختار کلی این کلاس که قصد استفاده از پروایدر LINQ مخصوص NHibernate را دارد باید به شکل زیر باشد (به کلاس پایه NHibernateContext دقت نمائید):

```
using System.Collections.Generic;
using System.Linq;
using NHibernate;
using NHibernate.Linq;
using NHSample1.Domain;

namespace ConsoleTestApplication
```

```
{
    class CLinqTest : NHibernateContext
    { }
}
```

اکنون پس از مشخص شدن context یا زمینه، نحوه ایجاد یک کوئری ساده LINQ to NHibernate به صورت زیر می‌تواند باشد:

```
using System.Collections.Generic;
using System.Linq;
using NHibernate;
using NHibernate.Linq;
using NHSample1.Domain;

namespace ConsoleTestApplication
{
    class CLinqTest : NHibernateContext
    {
        ISessionFactory _factory;

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

        public List<Customer> GetAllCustomers()
        {
            using (ISession session = _factory.OpenSession())
            {
                var query = from x in session.Linq<Customer>() select x;
                return query.ToList();
            }
        }
    }
}
```

ابتدا علاوه بر سایر فضا‌های نام مورد نیاز، فضای نام NHibernate.Linq به پروژه افزوده می‌شود. سپس از extension متدی به نام Linq بر روی اشیاء ISession از نوع یکی از موجودیت‌های تعریف شده در برنامه در قسمت‌های قبل، می‌توان جهت تهیه کوئری‌های Linq مورد نظر بهره برد. در این کوئری، لیست تمامی مشتری‌ها بازگشت داده می‌شود.

سپس جهت استفاده و بررسی آن در متد Main برنامه خواهیم داشت:

```
static void Main(string[] args)
{
    using (ISessionFactory session = Config.CreateSessionFactory(
        MsSqlConfiguration
            .MsSql2008
            .ConnectionString("Data Source=(local);Initial Catalog=HelloNHibernate;Integrated
Security = true")
            .ShowSql()
    ))
    {
        var customer1 = new Customer()
        {
            FirstName = "Vahid",
            LastName = "Nasiri",
            AddressLine1 = "Addr1",
            AddressLine2 = "Addr2",
            PostalCode = "1234",
            City = "Tehran",
            CountryCode = "IR"
        };

        var customer2 = new Customer()
        {
            FirstName = "Ali",
            LastName = "Hasani",
            AddressLine1 = "Addr..1",
            AddressLine2 = "Addr..2",
            PostalCode = "4321",
            City = "Shiraz",
        };
    }
}
```

```

        CountryCode = "IR"
    };

    var customer3 = new Customer()
    {
        FirstName = "Mohsen",
        LastName = "Shams",
        AddressLine1 = "Addr...1",
        AddressLine2 = "Addr...2",
        PostalCode = "5678",
        City = "Ahwaz",
        CountryCode = "IR"
    };

    CDbOperations db = new CDbOperations(session);
    db.AddNewCustomers(customer1, customer2, customer3);

    CLinqTest lt = new CLinqTest(session);
    foreach (Customer customer in lt.GetAllCustomers())
    {
        Console.WriteLine("Customer: LastName = {0}", customer.LastName);
    }

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

```

در این متد ابتدا تعدادی رکورد تعریف و سپس به دیتابیس اضافه شدند. در ادامه لیست تمامی آن‌ها از دیتابیس دریافت و نمایش داده می‌شود.

مهمترین مزیت استفاده از LINQ در این نوع کوئری‌ها نسبت به روش‌های دیگر، استفاده از کدهای strongly typed دات نت تحت نظر کامپایلر است، نسبت به رشته‌های معمولی SQL که کامپایلر کنترلی را بر روی آن‌ها نمی‌تواند داشته باشد (برای مثال اگر نوع یک ستون تغییر کند یا نام آن، در حالت استفاده از LINQ بلافاصله یک خطا را از کامپایلر جهت تصحیح مشکلات دریافت خواهیم کرد که این مورد در زمان استفاده از یک رشته معمولی صادق نیست). همچنین مزیت فراهم بودن Intellisense را حین نوشتن کوئری‌هایی از این دست نیز نمی‌توان ندید گرفت.

مثالی دیگر:

لیست تمام مشتری‌های شیرازی را نمایش دهید:

ابتدا متد GetCustomersByCity را به کلاس CLinqTest فوق اضافه می‌کنیم:

```

public List<Customer> GetCustomersByCity(string city)
{
    using (ISession session = _factory.OpenSession())
    {
        var query = from x in session.Linq<Customer>()
                     where x.City == city
                     select x;
        return query.ToList();
    }
}

```

سپس برای استفاده از آن، چند سطر ساده زیر به ادامه متد Main اضافه می‌شوند:

```

foreach (Customer customer in lt.GetCustomersByCity("Shiraz"))
{
    Console.WriteLine("Customer: LastName = {0}", customer.LastName);
}

```

یکی دیگر از مزایای استفاده از LINQ to NHibernate، امکان بکارگیری LINQ بر روی تمامی دیتابیس‌های پشتیبانی شده توسط NHibernate است؛ برای مثال مای اس کیوال، اوراکل و لیست کامل دیتابیس‌های پشتیبانی شده توسط NHibernate را در [این آدرس](#) می‌توانید مشاهده نمایید. (البته به نظر لیست آن،

آنچنان هم به روز نیست؛ چون در نگارش آخر NHibernate، پشتیبانی از اس کیوال سرور 2008 هم اضافه شده است)

نکته:

در کوئری‌های مثال‌های فوق همواره باید `<session.Linq>T` را ذکر کرد. اگر علاقمند بودید شبیه به روشی که در LINQ to SQL موجود است مثلاً `db.TableName` بجای `<session.Linq>T` در کوئری‌ها ذکر گردد، می‌توان اصلاحاتی را به صورت زیر اعمال کرد: یک کلاس جدید را به نام `SampleContext` به برنامه کنسول جاری با محتویات زیر اضافه نمایید:

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

namespace ConsoleTestApplication
{
    class SampleContext : NHibernateContext
    {
        public SampleContext(ISession session)
            : base(session)
        { }

        public IObservable<Customer> Customers
        {
            get { return Session.Linq<Customer>(); }
        }

        public IObservable<Employee> Employees
        {
            get { return Session.Linq<Employee>(); }
        }

        public IObservable<Order> Orders
        {
            get { return Session.Linq<Order>(); }
        }

        public IObservable<OrderItem> OrderItems
        {
            get { return Session.Linq<OrderItem>(); }
        }

        public IObservable<Product> Products
        {
            get { return Session.Linq<Product>(); }
        }
    }
}
```

در این کلاس به ازای تمام موجودیت‌های تعریف شده در پوشه domain برنامه اصلی خود (همان `NHSample1` قسمت‌های اول و دوم)، یک متد از نوع `IObservable` را باید تشکیل دهیم که پیاده سازی آن را ملاحظه می‌نمائید. سپس بازنویسی متد `GetCustomersByCity` بر اساس `SampleContext` فوق به صورت زیر خواهد بود که به کوئری‌های LINQ to SQL بسیار شبیه است:

```
using System.Collections.Generic;
using System.Linq;
using NHibernate;
using NHSample1.Domain;

namespace ConsoleTestApplication
{
    class CSampleContextTest
    {
        ISessionFactory _factory;

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

        public List<Customer> GetCustomersByCity(string city)
        {
            using (ISession session = _factory.OpenSession())
```

```
        {
            using (SampleContext db = new SampleContext(session))
            {
                var query = from x in db.Customers
                            where x.City == city
                            select x;
                return query.ToList();
            }
        }
    }
}
```

[دریافت سورس برنامه تا این قسمت](#)

و در تکمیل این بحث، می‌توان به لیستی از [101 مثال LINQ](#) ارائه شده در MSDN اشاره کرد که یکی از بهترین و سریع ترین مراجع یادگیری مبحث LINQ است.

ادامه دارد ...

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۴:۳۴:۲۴ ۱۳۸۸/۱۲/۱۳

در مورد LINQ to NHibernate در نگارش‌های اخیر آن کمی تغییر وجود داشته که نیاز است مطلب زیر را مطالعه بفرمائید:
<http://blogs.imeta.co.uk/sstrong/archive/2009/12/16/824.aspx>

نویسنده: وحید نصیری
تاریخ: ۲۳:۱۵:۴۸ ۱۳۸۹/۱۰/۰۴

NH 3.0 پشتیبانی یکپارچه‌ای را از LINQ ارائه می‌دهد و دیگر نیازی نیست مانند نگارش 2 آن پروایدر مخصوصی را جداگانه دریافت کرد. آن پروایدر قدیمی هم به نظر کنار گذاشته شده و از یک کتابخانه‌ی پخته‌تر به نام Remotion Linq Library استفاده گردیده است (+).
در این نگارش برای دسترسی به IQueryable interface می‌توان از متد session.Query استفاده کرد (بجای session.Linq نگارش قبلی).

نویسنده: fateme
تاریخ: ۱۱:۵۹:۲۸ ۱۳۸۹/۱۱/۱۲

سلام جناب آقای نصیری
ممنون از آموزشهای بسیار عالیتون
لینک دریافت پروایدر LINQ که گذاشتید موقع دانلود error میده می تونید لینک دیگه ای معرفی کنید

نویسنده: وحید نصیری
تاریخ: ۱۲:۲۸:۲۶ ۱۳۸۹/۱۱/۱۲

در کامنت‌های فوق توضیح دادم. این توضیحات مربوط به نگارش 2 بود. الان نگارش 3 را که دریافت کنید LINQ با آن یکپارچه است و نیازی به دریافت پروایدر کمکی نیست و پروایدر قدیمی هم منسوخ شده و دیگر ادامه نخواهد یافت.

نویسنده: fateme
تاریخ: ۱۴:۳۴:۲۱ ۱۳۸۹/۱۱/۱۲

جناب آقای نصیری ممنون از راهنماییتون
یه سوال دیگه ای که دارم اینکه برنامه من کلاس پایه NHibernateContext نمیشناسه ممنون میشم اگه مجددا راهنماییم کنید

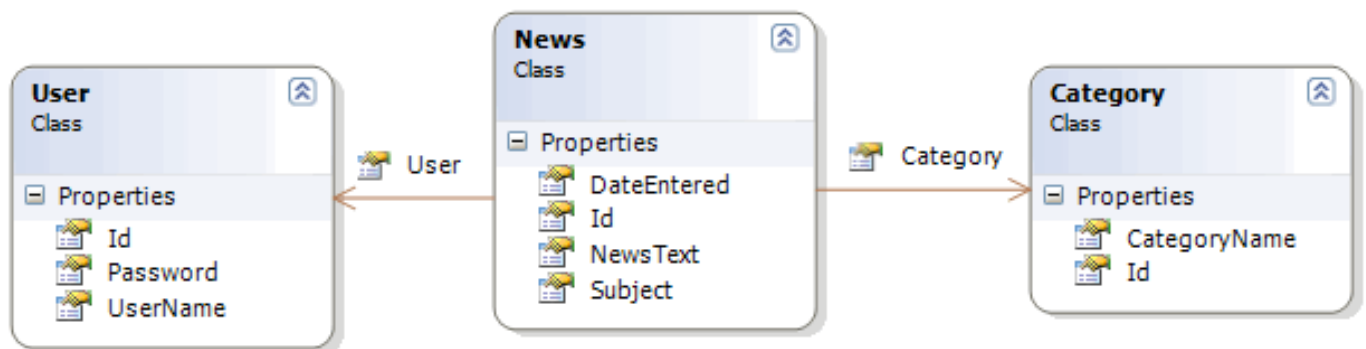
نویسنده: وحید نصیری
تاریخ: ۱۵:۰۰:۳۲ ۱۳۸۹/۱۱/۱۲

به NHibernateContext در NH 3.0 نیازی نیست. آنرا از مثال‌ها حذف کنید. فقط بجای Linq شما Query خواهید داشت (یک تغییر نام مختصر به همراه ساده سازی نحوه استفاده).

آشنایی با Automapping در فریم ورک Fluent NHibernate

اگر قسمت‌های قبل را دنبال کرده باشید، احتمالا به پروسه طولانی ساخت نگاشت‌ها توجه کرده‌اید. با کمک فریم ورک Fluent NHibernate می‌توان پروسه نگاشت domain model خود را به data model متناظر آن به صورت خودکار نیز انجام داد و قسمت عمده‌ای از کار به این صورت حذف خواهد شد. (این مورد یکی از تفاوت‌های مهم NHibernate با نمونه‌های مشابهی است که مایکروسافت تا تاریخ نگارش این مقاله ارائه داده است. برای مثال در نگارش‌های فعلی LINQ to SQL یا Entity framework، اول دیتابیس مطرح است و بعد ساخت کد از روی آن، در حالیکه در اینجا ابتدا کد و طراحی سیستم مطرح است و بعد نگاشت آن به سیستم داده‌ای و دیتابیس)

امروز قصد داریم یک سیستم ساده ثبت خبر را از صفر با NHibernate پیاده سازی کنیم و همچنین مروری داشته باشیم بر قسمت‌های قبلی.



مطابق کلاس دیاگرام فوق، این سیستم از سه کلاس خبر، کاربر ثبت کننده‌ی خبر و گروه خبری مربوطه تشکیل شده است.

ابتدا یک پروژه کنسول جدید را به نام NHSample2 آغاز کنید. سپس ارجاعاتی را به اسمبلی‌های زیر به آن اضافه نمایید:

FluentNHibernate.dll

NHibernate.dll

NHibernate.ByteCode.Castle.dll

NHibernate.Linq.dll

و ارجاعی به اسمبلی استاندارد System.Data.Services.dll در فریم ورک سه و نیم

سپس پوشه‌ای را به نام Domain به این پروژه اضافه نمایید (کلیک راست روی نام پروژه در VS.Net و سپس مراجعه به منوی Add->New folder). در این پوشه تعاریف موجودیت‌های برنامه را قرار خواهیم داد. سه کلاس جدید User، Category و News را در این پوشه ایجاد نمایید. محتویات این سه کلاس به شرح زیر هستند:

```
namespace NHSample2.Domain
{
    public class User
    {
        public virtual int Id { get; set; }
    }
}
```



```

        public virtual string UserName { get; set; }
        public virtual string Password { get; set; }
    }
}

```

```

namespace NHSample2.Domain
{
    public class Category
    {
        public virtual int Id { get; set; }
        public virtual string CategoryName { get; set; }
    }
}

```

```

using System;

namespace NHSample2.Domain
{
    public class News
    {
        public virtual Guid Id { get; set; }
        public virtual string Subject { get; set; }
        public virtual string NewsText { get; set; }
        public virtual DateTime DateEntered { get; set; }
        public virtual Category Category { get; set; }
        public virtual User User { get; set; }
    }
}

```

همانطور که در قسمت‌های قبل نیز ذکر شد، تمام خواص پابلیک کلاس‌های Domain ما به صورت virtual تعریف شده‌اند تا lazy loading را در NHibernate فعال سازیم. در حالت [lazy loading](#)، اطلاعات تنها زمانیکه به آن‌ها نیاز باشد بارگذاری خواهند شد. این مورد در حالتیکه نیاز به نمایش اطلاعات تنها یک شیء وجود داشته باشد بسیار مطلوب می‌باشد، یا هنگام ثبت و به روز رسانی اطلاعات نیز یکی از بهترین روش‌ها است. اما زمانیکه با لیستی از اطلاعات سروکار داشته باشیم باعث کاهش افت کارایی خواهد شد زیرا برای مثال نمایش آن‌ها سبب خواهد شد که 100 ها کوئری دیگر جهت دریافت اطلاعات هر رکورد در حال نمایش اجرا شود (مفهوم دسترسی به اطلاعات تنها در صورت نیاز به آن‌ها). Lazy loading و eager loading (همانند مثال‌های قبلی) هر دو در NHibernate به سادگی قابل تنظیم هستند (برای مثال LINQ to SQL به صورت پیش فرض همواره lazy load است و تا این تاریخ راه استاندارد برای امکان تغییر و تنظیم این مورد پیش بینی نشده است).

اکنون کلاس جدید Config را به برنامه اضافه نمائید:

```

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

namespace NHSample2
{
    class Config
    {
        public static Configuration GenerateMapping(IPersistenceConfigurer dbType)
        {
            var cfg = dbType.ConfigureProperties(new Configuration());

            new AutoPersistenceModel()
                .Where(x => x.Namespace.EndsWith("Domain"))
                .AddEntityAssembly(typeof(NHSample2.Domain.News).Assembly).Configure(cfg);

            return cfg;
        }

        public static void GenerateDbScript(Configuration config, string filePath)
        {
            bool script = true; // فقط اسکریپت دیتابیس تولید گردد
        }
    }
}

```

```

        bool export = false; //نیازی نیست بر روی دیتابیس هم اجرا شود
        new SchemaExport(config).SetOutputFile(filePath).Create(script, export);
    }

    public static void BuildDbSchema(Configuration config)
    {
        bool script = false; //آیا خروجی در کنسول هم نمایش داده شود
        bool export = true; //آیا بر روی دیتابیس هم اجرا شود
        bool drop = false; //آیا اطلاعات موجود درآپ شوند
        new SchemaExport(config).Execute(script, export, drop);
    }

    public static void CreateSQL2008DbPlusScript(string connectionString, string filePath)
    {
        Configuration cfg =
            GenerateMapping(
                MsSqlConfiguration
                    .MsSql2008
                    .ConnectionString(connectionString)
                    .ShowSql()
                );
        GenerateDbScript(cfg, filePath);
        BuildDbSchema(cfg);
    }

    public static ISessionFactory CreateSessionFactory(IPersistenceConfigurer dbType)
    {
        return
            Fluently.Configure().Database(dbType)
                .Mappings(m => m.AutoMappings
                    .Add(
                        new AutoPersistenceModel()
                            .Where(x => x.Namespace.EndsWith("Domain"))
                            .AddEntityAssembly(typeof(NHSample2.Domain.News).Assembly))
                    )
                .BuildSessionFactory();
    }
}

```

در متد `GenerateMapping` از قابلیت `Automapping` موجود در فریم ورک `Fluent NHibernate` استفاده شده است (بدون نوشتن حتی یک سطر جهت تعریف این نگاشت‌ها). این متد نوع دیتابیس مورد نظر را جهت ساخت تنظیمات خود دریافت می‌کند. سپس با کمک کلاس `AutoPersistenceModel` این فریم ورک، به صورت خودکار از اسمبلی برنامه نگاشت‌های لازم را به کلاس‌های موجود در پوشه `Domain` ما اضافه می‌کند (مرسوم است که این پوشه در یک پروژه `Class library` مجزا تعریف شود که در این برنامه جهت سهولت کار در خود برنامه قرار گرفته است). قسمت `Where` ذکر شده به این جهت معرفی گردیده است تا `Fluent NHibernate` برای تمامی کلاس‌های موجود در اسمبلی جاری، سعی در تعریف نگاشت‌های لازم نکند. این نگاشت‌ها تنها به کلاس‌های موجود در پوشه دومین ما محدود شده‌اند.

سه متد بعدی آن، جهت ایجاد اسکریپت دیتابیس از روی این نگاشت‌های تعریف شده و سپس اجرای این اسکریپت بر روی دیتابیس جاری معرفی شده، تهیه شده‌اند. برای مثال `CreateSQL2008DbPlusScript` یک مثال ساده از استفاده دو متد قبلی جهت ایجاد اسکریپت و دیتابیس متناظر اس کیوال سرور 2008 بر اساس نگاشت‌های برنامه است.

با متد `CreateSessionFactory` در قسمت‌های قبل آشنا شده‌اید. تنها تفاوت آن در این قسمت، استفاده از کلاس `AutoPersistenceModel` جهت تولید خودکار نگاشت‌ها است.

در ادامه دیتابیس متناظر با موجودیت‌های برنامه را ایجاد خواهیم کرد:

```

using System;

namespace NHSample2
{
    class Program
    {
        static void Main(string[] args)
        {
            Config.CreateSQL2008DbPlusScript(
                "Data Source=(local);Initial Catalog=HelloNHibernate;Integrated Security = true",
                "db.sql");
        }
    }
}

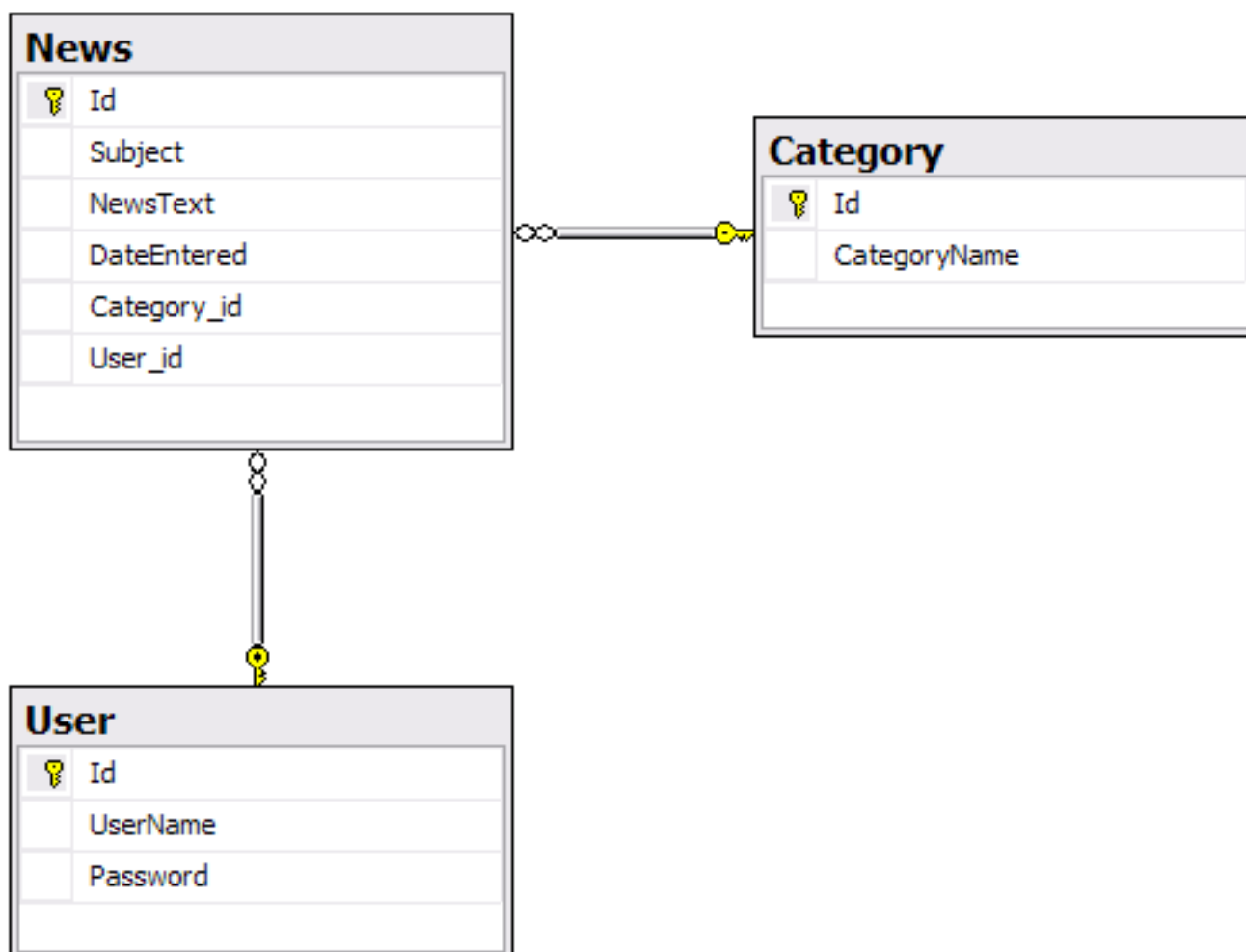
```

```

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

```

پس از اجرای برنامه، ابتدا فایل اسکریپت دیتابیس به نام db.sql در پوشه اجرایی برنامه تشکیل خواهد شد و سپس این اسکریپت به صورت خودکار بر روی دیتابیس معرفی شده اجرا می‌گردد. دیتابیس دیاگرام حاصل را در شکل زیر می‌توانید مشاهده نمایید:



همچنین اسکریپت تولید شده آن، صرفنظر از عبارات drop اولیه، به صورت زیر است:

```

create table [Category] (
    Id INT IDENTITY NOT NULL,
    CategoryName NVARCHAR(255) null,
    primary key (Id)
)

create table [User] (
    Id INT IDENTITY NOT NULL,

```

```

        UserName NVARCHAR(255) null,
        Password NVARCHAR(255) null,
        primary key (Id)
    )

create table [News] (
    Id UNIQUEIDENTIFIER not null,
    Subject NVARCHAR(255) null,
    NewsText NVARCHAR(255) null,
    DateEntered DATETIME null,
    Category_id INT null,
    User_id INT null,
    primary key (Id)
)

alter table [News]
add constraint FKE660F9E1C9CF79
foreign key (Category_id)
references [Category]

alter table [News]
add constraint FKE660F95C1A3C92
foreign key (User_id)
references [User]

```

اکنون یک سری گروه خبری، کاربر و خبر را به دیتابیس خواهیم افزود:

```

using System;
using FluentNHibernate.Cfg.Db;
using NHibernate;
using NHSample2.Domain;

namespace NHSample2
{
    class Program
    {
        static void Main(string[] args)
        {
            using (ISessionFactory sessionFactory = Config.CreateSessionFactory(
                MsSqlConfiguration
                    .MsSql2008
                    .ConnectionString("Data Source=(local);Initial Catalog=HelloNHibernate;Integrated
Security = true")
                    .ShowSql())
            {
                using (ISession session = sessionFactory.OpenSession())
                {
                    using (ITransaction transaction = session.BeginTransaction())
                    {
                        //توجه به کلیدهای خارجی تعریف شده ابتدا باید گروه‌ها را اضافه کرد
                        Category ca = new Category() { CategoryName = "Sport" };
                        session.Save(ca);
                        Category ca2 = new Category() { CategoryName = "IT" };
                        session.Save(ca2);
                        Category ca3 = new Category() { CategoryName = "Business" };
                        session.Save(ca3);

                        //سپس یک کاربر را به دیتابیس اضافه می‌کنیم
                        User u = new User() { Password = "123$5@1", UserName = "VahidNasiri" };
                        session.Save(u);

                        //اکنون می‌توان یک خبر جدید را ثبت کرد

                        News news = new News()
                        {
                            Category = ca,
                            User = u,
                            DateEntered = DateTime.Now,
                            Id = Guid.NewGuid(),
                            NewsText = "متن خبر جدید",
                            Subject = "عنوانی دلخواه"
                        };
                        session.Save(news);
                    }
                }
            }
        }
    }
}

```

```

        transaction.Commit(); //پایان تراکنش
    }
}

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

```

جهت بررسی انجام عملیات ثبت هم می‌توان به دیتابیس مراجعه کرد، برای مثال:

```

2 SELECT TOP 1000 [Id]
3     , [Subject]
4     , [NewsText]
5     , [DateEntered]
6     , [Category_id]
7     , [User_id]
8 FROM [HelloNHibernate].[dbo].[News]

```

Results		Messages				
	Id	Subject	NewsText	DateEntered	Category_id	User_id
1	0552730B-83A...	عنوانی دلخواه	متن خبر جدید	2009-10-13 21:55:08.000	1	1

و یا می‌توان از LINQ استفاده کرد:

برای مثال کاربر VahidNasiri تعریف شده را یافته، اطلاعات آن را نمایش دهید؛ سپس نام او را به Vahid ویرایش کرده و دیتابیس را به روز کنید.

برای اینکه کوئری‌های LINQ ما شبیه به LINQ to SQL شوند، کلاس NewsContext را به صورت ذیل تشکیل می‌دهیم. این کلاس از کلاس پایه NHibernateContext مشتق شده و سپس به ازای تمام موجودیت‌های برنامه، یک متد از نوع IQueryable را تشکیل خواهیم داد.

```

using System.Linq;
using NHibernate;
using NHibernate.Linq;
using NHSample2.Domain;

namespace NHSample2
{
    class NewsContext : NHibernateContext
    {
        public NewsContext(ISession session)
            : base(session)
        { }

        public IQueryable<News> News
        {
            get { return Session.Linq<News>(); }
        }
    }
}

```

```

public IOrderedQueryable<Category> Categories
{
    get { return Session.Linq<Category>(); }
}

public IOrderedQueryable<User> Users
{
    get { return Session.Linq<User>(); }
}
}

```

اکنون جهت یافتن کاربر و به روز رسانی اطلاعات او در دیتابیس خواهیم داشت:

```

using System;
using FluentNHibernate.Cfg.Db;
using NHibernate;
using System.Linq;
using NHSample2.Domain;

namespace NHSample2
{
    class Program
    {
        static void Main(string[] args)
        {
            using (ISessionFactory sessionFactory = Config.CreateSessionFactory(
                MsSqlConfiguration
                    .MsSql2008
                    .ConnectionString("Data Source=(local);Initial Catalog=HelloNHibernate;Integrated
Security = true")
                    .ShowSql())
            {
                using (ISession session = sessionFactory.OpenSession())
                {
                    using (ITransaction transaction = session.BeginTransaction())
                    {
                        using (NewsContext db = new NewsContext(session))
                        {
                            var query = from x in db.Users
                                where x.UserName == "VahidNasiri"
                                select x;

                            //اگر چیزی یافت شد
                            if (query.Any())
                            {
                                User vahid = query.First();
                                //نمایش اطلاعات کاربر
                                Console.WriteLine("Id: {0}, UserName: {0}", vahid.Id, vahid.UserName);
                                //به روز رسانی نام کاربر
                                vahid.UserName = "Vahid";
                                session.Update(vahid);

                                transaction.Commit(); //پایان تراکنش
                            }
                        }
                    }
                }

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

```

مباحث تکمیلی AutoMapper

اگر به اسکریپت دیتابیس تولید شده دقت کرده باشید، عملیات AutoMapper یک سری پیش فرض‌هایی را اعمال کرده است. برای مثال فیلد Id را از نوع identity و به صورت کلید تعریف کرده، یا رشته‌ها را به صورت nvarchar با طول 255 ایجاد نموده است. امکان سفارشی سازی این موارد نیز وجود دارد.

مثال:

```
using FluentNHibernate.Conventions.Helpers;

public static Configuration GenerateMapping(IPersistenceConfigurer dbType)
{
    var cfg = dbType.ConfigureProperties(new Configuration());

    new AutoPersistenceModel()
        .Conventions.Add()
        .Where(x => x.Namespace.EndsWith("Domain"))
        .Conventions.Add(
            PrimaryKey.Name.Is(x => "ID"),
            DefaultLazy.Always(),
            ForeignKey.EndsWith("ID"),
            Table.Is(t => "tbl" + t.EntityType.Name)
        )
        .AddEntityAssembly(typeof(NHSample2.Domain.News).Assembly)
        .Configure(cfg);

    return cfg;
}
```

تابع `GenerateMapping` معرفی شده را اینجا با قسمت `Conventions.Add` تکمیل کرده ایم. به این صورت دقیقاً مشخص شده است که فیلدهایی با نام `ID` باید `primary key` در نظر گرفته شوند، همواره `lazy loading` صورت گیرد و نام کلید خارجی به `ID` ختم شود. همچنین نام جداول با `tbl` شروع گردد.

[روش دیگری](#) نیز برای معرفی این قرار داده‌ها و پیش فرض‌ها وجود دارد. فرض کنید می‌خواهیم طول رشته پیش فرض را از 255 به 500 تغییر دهیم. برای اینکار باید اینترفیس `IPropertyConvention` را پیاده سازی کرد:

```
using FluentNHibernate.Conventions;
using FluentNHibernate.Conventions.Instances;

namespace NHSample2.Conventions
{
    class MyStringLengthConvention : IPropertyConvention
    {
        public void Apply(IPropertyInstance instance)
        {
            instance.Length(500);
        }
    }
}
```

سپس نحوه‌ی معرفی آن به صورت زیر خواهد بود:

```
public static Configuration GenerateMapping(IPersistenceConfigurer dbType)
{
    var cfg = dbType.ConfigureProperties(new Configuration());

    new AutoPersistenceModel()
        .Conventions.Add()
        .Where(x => x.Namespace.EndsWith("Domain"))
        .Conventions.Add<MyStringLengthConvention>()
        .AddEntityAssembly(typeof(NHSample2.Domain.News).Assembly)
        .Configure(cfg);

    return cfg;
}
```

نکته:

اگر برای یافتن اطلاعات بیشتر در این مورد در وب جستجو کنید، اکثر مثال‌هایی را که مشاهده خواهید کرد بر اساس نگارش بتای `fluent NHibernate` هستند و هیچکدام با نگارش نهایی این فریم ورک کار نمی‌کنند. در نگارش رسمی نهایی ارائه شده، تغییرات

بسیاری صورت گرفته که آن‌ها را [در این آدرس](#) می‌توان مشاهده کرد.

[دریافت سورس برنامه قسمت ششم](#)

ادامه دارد ...

نظرات خوانندگان

نویسنده: Majid

تاریخ: ۱۳۸۸/۱۲/۰۹ ۱۲:۴۱:۵۴

با سلام، می‌خواستم بدانم چطور می‌شود توسط FNH تعریف کرد که در یک جدول بیشتر از یک PK داشته باشیم. متشکرم.

نویسنده: وحید نصیری

تاریخ: ۱۳۸۸/۱۲/۰۹ ۱۴:۳۴:۲۱

سلام

نحوه پیاده سازی کامپوزیت کی در FHN:

http://devlicio.us/blogs/derik_whittaker/archive/2009/01/16/using-fluentnhibernate-to-map-composite-keys-for-a-table.aspx

و اینکه چرا کامپوزیت کی خوب نیست اساسا:

<http://codebetter.com/blogs/jeremy.miller/archive/2007/02/01/Composite-keys-are-evil.aspx>

نویسنده: Majid

تاریخ: ۱۳۸۸/۱۲/۰۹ ۱۷:۲۷:۰۲

سپاس بخاطر پاسخگویی سریع

نویسنده: Ahmad

تاریخ: ۱۳۸۹/۰۱/۰۵ ۱۶:۲۵:۰۳

اگر بخواهیم از Criteria استفاده کنیم (selectهای joinدار) روابط بین جداول را چگونه تشخیص می‌دهد؟ (با استفاده از AutoMapping) اگر ممکن است مثالی ارائه بفرمائید.

نویسنده: وحید نصیری

تاریخ: ۱۳۸۹/۰۱/۰۵ ۱۷:۰۷:۱۰

تشخیص روابط بین جداول یعنی همان mapping خودکار، یعنی همان نحوه‌ی تعریف کلاس‌های شما و برقراری روابطی که در طی چند قسمت مثال زده شد. سیستم پیش فرض NHibernate بر اساس اول طراحی کلاس‌ها و بعد ایجاد ارتباط با دیتابیس است که اینجا به صورت خودکار صورت می‌گیرد. برای مثال در قسمت هشتم یک سیستم many-to-many مثال زده شده است به همراه کوئری‌هایی از نوع Linq. اینجا فقط تعریف کلاس‌هایی که بیانگر روابط many-to-many باشند مهم است؛ نحوه‌ی نگاشت خودکار آن‌ها به دیتابیس کار Fluent NHibernate است. (از این نوع مثال‌ها در هر قسمت پیاده سازی شده) جزئیات ریز نحوه‌ی نگاشت خودکار با مطالعه‌ی سوره کتابخانه NHibernate و مشتقات آن قابل درک است (برای علاقمندان). ضمناً فرقی نمی‌کند از Linq قسمت پنجم استفاده کنید یا هر روش موجود دیگری برای کوئری گرفتن (زمانیکه Linq هست و نگارش‌های جدید آن برای NHibernate پیشرفت زیادی داشته، چرا روش‌های دیگر؟).

نویسنده: Ahmad

تاریخ: ۱۳۸۹/۰۱/۰۵ ۱۷:۵۳:۴۳

بسیار عالی

اما مثل اینکه Linq با NH مشکل داره: (در Join)

<http://guildsocial.web703.discountasp.net/dasblogce/2009/07/29/LinqToNHibernateJoins.aspx>

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۱/۰۵ ۱۹:۲۳:۵۹

یک سری وبلاگ در مورد NHibernate هست که مربوط به توسعه دهنده‌های اصلی آن است و مرجع محسوب می‌شوند. برای مثال:

<http://ayende.com/Blog>

در این پست زیر قید شده که هر آنچه که در criteria API پشتیبانی می‌شود در نگارش یک LINQ آن هم وجود دارد (بنابراین group joins or subqueries پشتیبانی نمی‌شود چون در criteria API وجود ندارد + join هم به نظر هنوز تکمیل نشده).

<http://ayende.com/Blog/archive/2009/07/26/nhibernate-linq-1.0-released.aspx>

وبلاگ توسعه دهنده‌ی اصلی LINQ برای NHibernate

<http://blogs.imeta.co.uk/sstrong/Default.aspx>

join هم اکنون در نگارش بتای جدید آن کار می‌کند:

<http://blogs.imeta.co.uk/sstrong/archive/2009/12/16/823.aspx>

وبلاگ یکی از مدیر پروژه‌های NHibernate

<http://fabiomaulo.blogspot.com>

نویسنده: Ahmad
تاریخ: ۱۳۸۹/۰۱/۰۵ ۲۳:۱۰:۳۷

Thanks

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 (WinForms، WPF 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.

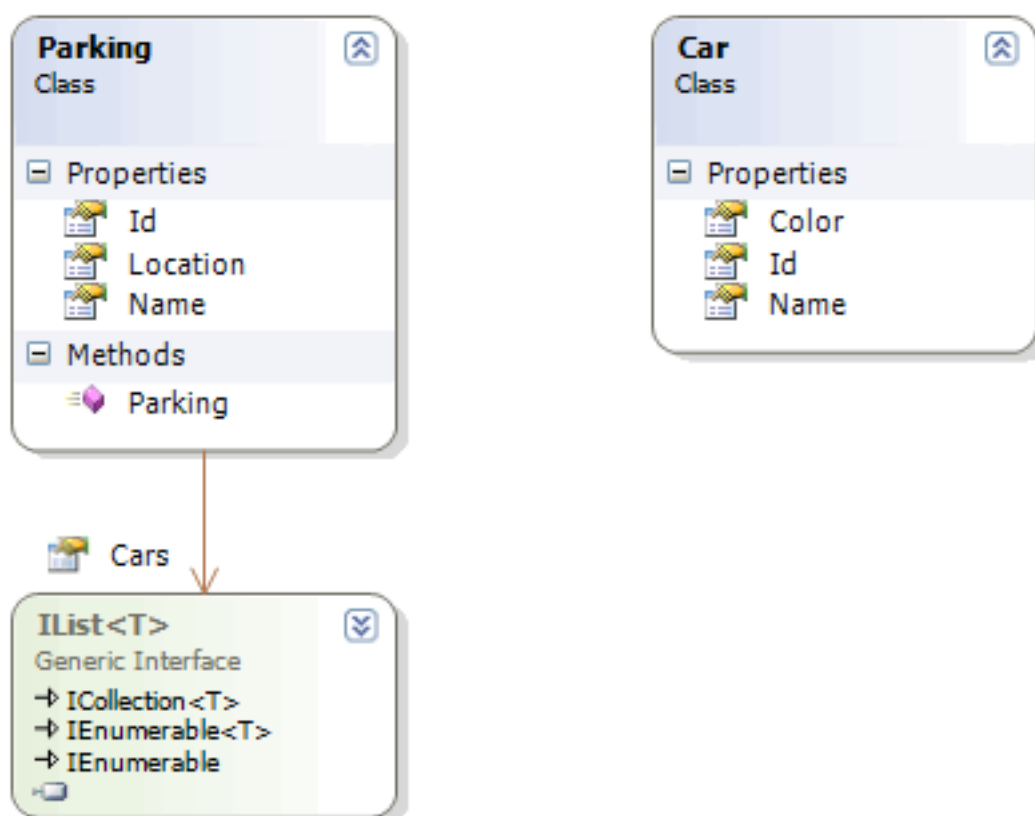
برای پیاده سازی شیء سشن فکتوری به صورتی که یکبار در طول برنامه ایجاد شود و بارها مورد استفاده قرار گیرد باید از یکی از الگوهای معروف طراحی برنامه نویسی شیء گرا به نام Singleton Pattern استفاده کرد. پیاده سازی نمونه‌ی thread safe آن که در برنامه‌های ذاتا چند ریسمانی وب و همچنین برنامه‌های معمولی ویندوزی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

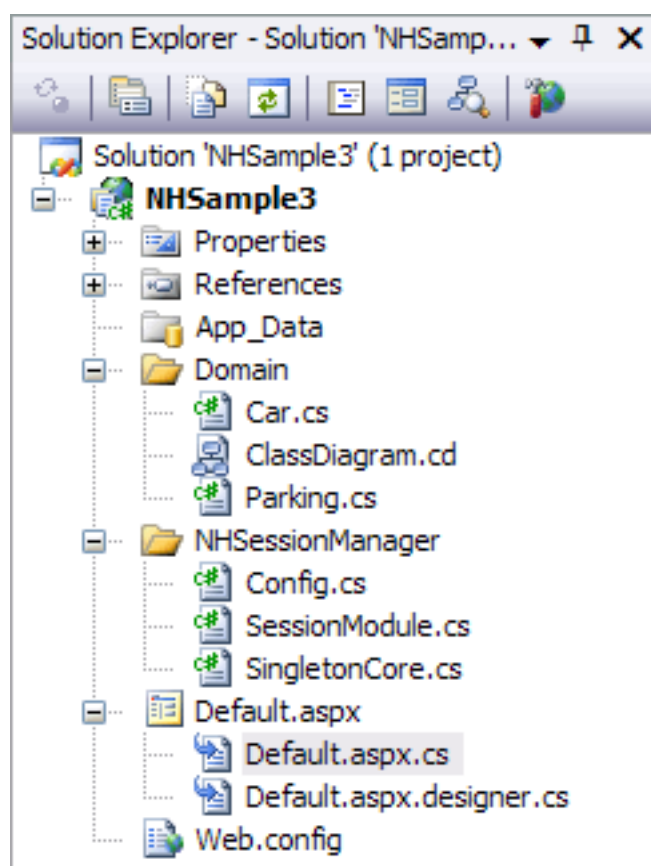
NHibernate.dll

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; //در کنسول هم نمایش داده شود
            bool export = true; //آیا بر روی دیتابیس هم اجرا شود
            bool dropTables = false; //آیا جداول موجود درآپ 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:

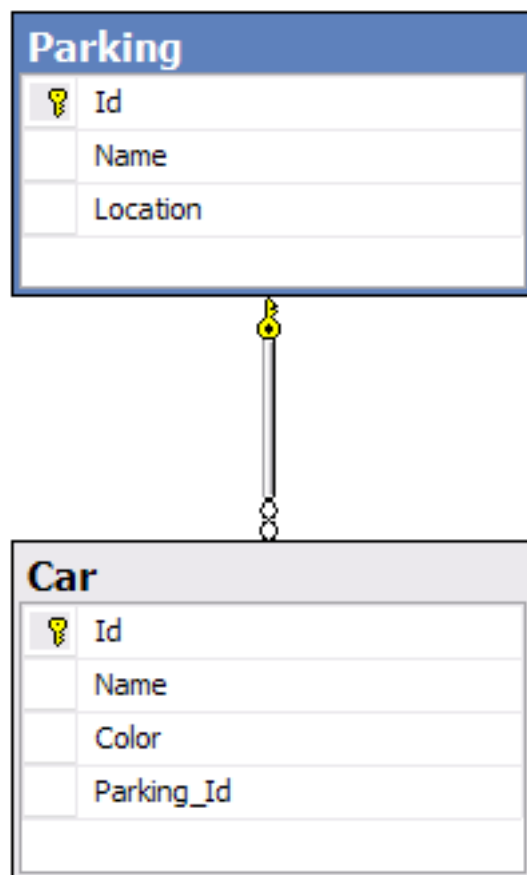
```
<connectionStrings>
  <!--NHSessionManager-->
  <add name="DbConnectionString"
    connectionString="Data Source=(local);Initial Catalog=HelloNHibernate;Integrated Security =
true" />
</connectionStrings>
```

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 باید 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 مطلوب تر 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 IObservable<Car> Cars
        {
            get { return Session.Linq<Car>(); }
        }

        public IObservable<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 ویو 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 ویو
            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: ۱۴:۰۱:۳۸ ۱۳۸۹/۰۹/۱۴

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

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

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 نامObjectContext 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: ۱۵:۳۰ ۱۳۹۱/۰۵/۲۷

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: ۱۶:۲۶ ۱۳۹۱/۰۵/۲۷

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.

معرفی الگوی Repository

- روش متداول کار با فناوری‌های مختلف دسترسی به داده‌ها عموماً بدین شکل است:
- (الف) یافتن رشته اتصال رمزنگاری شده به دیتابیس از یک فایل کانفیگ (در یک برنامه اصولی البته!)
- (ب) باز کردن یک اتصال به دیتابیس
- (ج) ایجاد اشیاء Command برای انجام عملیات مورد نظر
- (د) اجرا و فراخوانی اشیاء مراحل قبل
- (ه) بستن اتصال به دیتابیس و آزاد سازی اشیاء

اگر در برنامه‌های یک تازه کار به هر محلی از برنامه او دقت کنید این 5 مرحله را می‌توانید مشاهده کنید. همه جا! قسمت ثبت، قسمت جستجو، قسمت نمایش و ...

مشکلات این روش:

- 1- حجم کارهای تکراری انجام شده بالا است. اگر قسمتی از فناوری دسترسی به داده‌ها را به اشتباه درک کرده باشد، پس از مطالعه بیشتر و مشخص شدن نحوه رفع مشکل، قسمت عمده‌ای از برنامه را باید اصلاح کند (زیرا کدهای تکراری همه جای آن پراکنده‌اند).
 - 2- برنامه نویس هر بار باید این مراحل را به درستی انجام دهد. اگر در یک برنامه بزرگ تنها قسمت آخر در یکی از مراحل کاری فراموش شود دیر یا زود برنامه تحت فشار کاری بالا از کار خواهد افتاد (و متأسفانه این مساله بسیار شایع است).
 - 3- برنامه منحصر برای یک نوع دیتابیس خاص تهیه خواهد شد و تغییر این رویه جهت استفاده از دیتابیس دیگر (مثلاً کوچ برنامه از اکسس به اس کیوال سرور)، نیازمند بازنویسی کل برنامه می‌باشد.
- و ...

همین برنامه نویس پس از مدتی کار به این نتیجه می‌رسد که باید برای این کارهای متداول، یک لایه و کلاس دسترسی به داده‌ها را تشکیل دهد. اکنون هر قسمتی از برنامه برای کار با دیتابیس باید با این کلاس مرکزی که انجام کارهای متداول با دیتابیس را خلاصه می‌کند، کار کند. به این صورت کد نویسی یک نواختی با حذف کدهای تکراری از سطح برنامه و همچنین بدون فراموش شدن قسمت مهمی از مراحل کاری، حاصل می‌گردد. در اینجا اگر روزی قرار شد از یک دیتابیس دیگر استفاده شود فقط کافی است یک کلاس برنامه تغییر کند و نیازی به بازنویسی کل برنامه نخواهد بود.

این روزها تشکیل این لایه دسترسی به داده‌ها (data access layer یا DAL) نیز مرسوم است! و دلایل آن در مباحث چرا به یک ORM نیازمندیم برشمرده شده است. جهت کار با ORM ها نیز نیازمند یک لایه دیگر می‌باشیم تا یک سری اعمال متداول با آن‌ها را کپسوله کرده و از حجم کارهای تکراری خود بکاهیم. برای این منظور قبل از اینکه دست به اختراع بزنیم، بهتر است به الگوهای طراحی برنامه نویسی شیء گرا رجوع کرد و از رهنمودهای آن استفاده نمود.

الگوی Repository یکی از الگوهای برنامه نویسی با مقیاس سازمانی است. با کمک این الگو لایه نگاشت اشیاء برنامه به دیتابیس تشکیل شده و عملاً برنامه را مستقل از نوع ORM مورد استفاده می‌کند. به این صورت هم از تشکیل یک سری کدهای تکراری در سطح برنامه جلوگیری شده و هم از وابستگی بین مدل برنامه و لایه دسترسی به داده‌ها (که در اینجا همان NHibernate می‌باشد) جلوگیری می‌شود. الگوی Repository (مخزن)، کار ثبت، حذف، جستجو و به روز رسانی داده‌ها را با ترجمه آن‌ها به روش‌های بومی مورد استفاده توسط ORM مورد نظر، کپسوله می‌کند. به این شکل شما می‌توانید یک الگوی مخزن عمومی را برای کارهای خود تهیه کرده و به سادگی از یک ORM به ORM دیگر کوچ کنید؛ زیرا کدهای برنامه شما به هیچ ORM خاصی گره نخورده و این عملیات بومی کار با ORM توسط لایه‌ای که توسط الگوی مخزن تشکیل شده، صورت گرفته است.

طراحی کلاس مخزن باید شرایط زیر را برآورده سازد:

الف) باید یک طراحی عمومی داشته باشد و بتواند در پروژه‌های متعددی مورد استفاده مجدد قرار گیرد.

ب) باید با سیستمی از نوع اول طراحی و کد نویسی و بعد کار با دیتابیس، سازگاری داشته باشد.

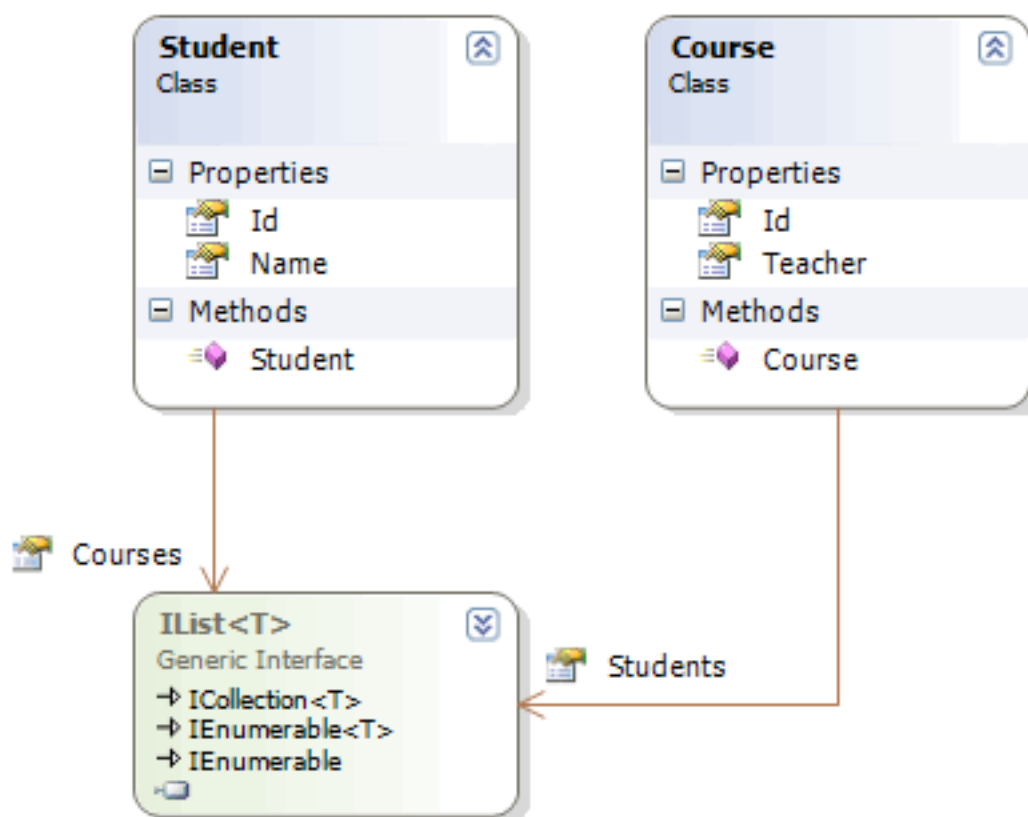
ج) باید امکان انجام آزمایشات واحد را سهولت بخشد.

د) باید وابستگی کلاس‌های دومین برنامه را به زیر ساخت ORM مورد استفاده قطع کند (اگر سال بعد به این نتیجه رسیدید که ORM ایی به نام XYZ برای کار شما بهتر است، فقط پیاده سازی این کلاس باید تغییر کند و نه کل برنامه).

ه) باید استفاده از کوثری‌هایی از نوع strongly typed را ترویج کند (مثل کوثری‌هایی از نوع LINQ).

بررسی مدل برنامه

مدل این قسمت (برنامه NHSample4 از نوع کنسول با همان ارجاعات متداول ذکر شده در قسمت‌های قبل)، از نوع many-to-many می‌باشد. در اینجا یک واحد درسی توسط چندین دانشجو می‌تواند اخذ شود یا یک دانشجو می‌تواند چندین واحد درسی را اخذ نماید که برای نمونه کلاس دیاگرام و کلاس‌های متشکل آن به شکل زیر خواهند بود:



```
using System.Collections.Generic;

namespace NHSample4.Domain
{
    public class Course
    {
        public virtual int Id { get; set; }
        public virtual string Teacher { get; set; }
        public virtual IList<Student> Students { get; set; }

        public Course()
        {
        }
    }
}
```

```

        Students = new List<Student>();
    }
}

```

```

using System.Collections.Generic;

namespace NHSample4.Domain
{
    public class Student
    {
        public virtual int Id { get; set; }
        public virtual string Name { get; set; }
        public virtual IList<Course> Courses { get; set; }

        public Student()
        {
            Courses = new List<Course>();
        }
    }
}

```

کلاس کانفیگ برنامه جهت ایجاد نگاشت‌ها و سپس ساخت دیتابیس متناظر

```

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"))
                    )
                    .Mappings(
                        m => m.AutoMappings.Add(
                            new AutoPersistenceModel()
                                .Where(x => x.Namespace.EndsWith("Domain"))
                                .AddEntityAssembly(typeof(NHSample4.Domain.Course).Assembly))
                            .ExportTo(System.Environment.CurrentDirectory)
                        );
        }

        public static void CreateDb()
        {
            bool script = false; // نمایش داده شود
            bool export = true; // اجرا شود
            bool dropTables = false; // شوند
            new SchemaExport(GetConfig().BuildConfiguration()).Execute(script, export, dropTables);
        }
    }
}

```

چند نکته در مورد این کلاس:

الف) با توجه به اینکه برنامه از نوع ویندوزی است، برای مدیریت صحیح کانکشن استرینگ، فایل App.Config را به برنامه افروده و محتویات آن را به شکل زیر تنظیم می‌کنیم (تا کلید DbConnectionString توسط متد GetConfig مورد استفاده قرار گیرد):

```

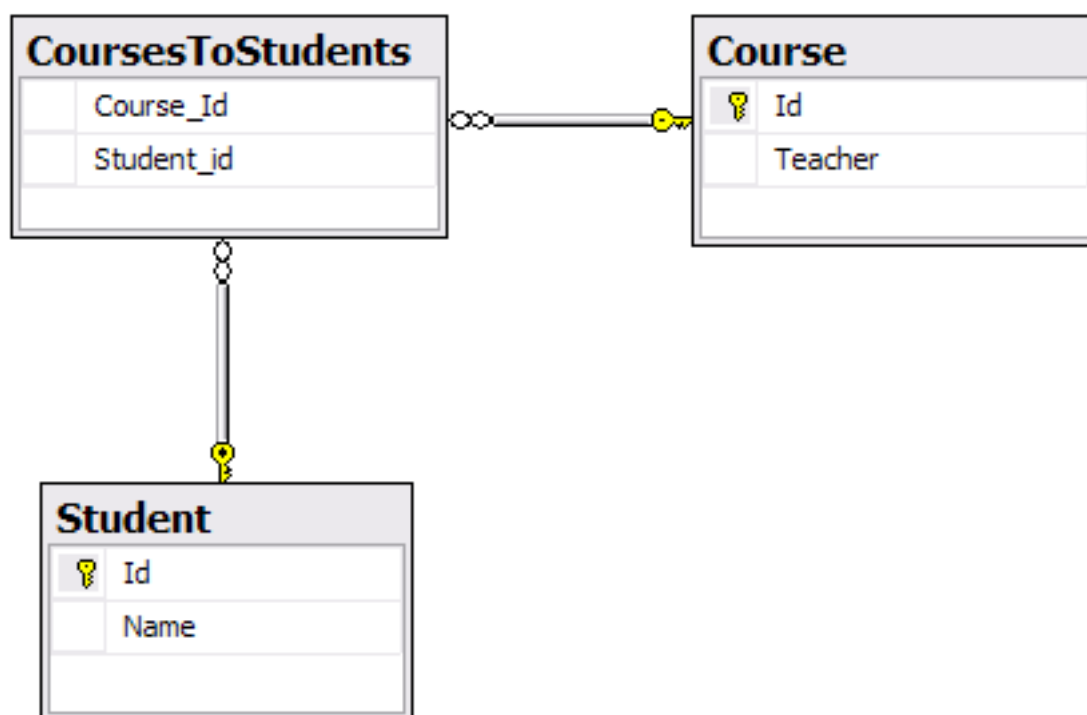
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<connectionStrings>
  <!--NHSessionManager-->

```

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

ب) در NHibernate سنتی (!) کار ساخت نگاشت‌ها توسط یک سری فایل xml صورت می‌گیرد که با معرفی فریم ورک Fluent NHibernate و استفاده از قابلیت‌های Auto Mapping آن، این کار با سهولت و دقت هر چه تمام‌تر قابل انجام است که توضیحات نحوه‌ی انجام آن‌را در قسمت‌های قبل مطالعه فرمودید. اگر نیاز بود تا این فایل‌های XML نیز جهت بررسی شخصی ایجاد شوند، تنها کافی است از متد ExportTo آن همانگونه که در متد GetConfig استفاده شده، کمک گرفته شود. به این صورت پس از ایجاد خودکار نگاشت‌ها، فایل‌های XML متناظر نیز در مسیری که به عنوان آرگومان متد ExportTo مشخص گردیده است، تولید خواهند شد (دو فایل NHSample4.Domain.Student.hbm.xml و NHSample4.Domain.Course.hbm.xml را در پوشه‌ای که محل اجرای برنامه است خواهید یافت).

با فراخوانی متد CreateDb این کلاس، پس از ساخت خودکار نگاشت‌ها، database schema متناظر، در دیتابیس‌ی که توسط کانکشن استرینگ برنامه مشخص شده، ایجاد خواهد شد که دیتابیس دیاگرام آن‌را در شکل ذیل مشاهده می‌نمائید (جداول دانشجویان و واحدها هر کدام به صورت موجودیتی مستقل ایجاد شده که ارجاعات آن‌ها در جدولی سوم نگهداری می‌شود).



پیاده سازی الگوی مخزن

اینترفیس عمومی الگوی مخزن به شکل زیر می‌تواند باشد:

```
using System;
using System.Linq;
using System.Linq.Expressions;
```

```
namespace NHSample4.NHRepository
{
    //Repository Interface
    public interface IRepository<T>
    {
        T Get(object key);

        T Save(T entity);
        T Update(T entity);
        void Delete(T entity);

        IQueryable<T> Find();
        IQueryable<T> Find(Expression<Func<T, bool>> predicate);
    }
}
```

سپس پیاده سازی آن با توجه به کلاس SingletonCore ایی که در قسمت قبل تهیه کردیم (جهت مدیریت صحیح سشن فکتوری)، به صورت زیر خواهد بود.

این کلاس کار آغاز و پایان تراکنش‌ها را نیز مدیریت کرده و جهت سهولت کار اینترفیس IDisposable را نیز پیاده سازی می‌کند :

```
using System;
using System.Linq;
using NHSessionManager;
using NHibernate;
using NHibernate.Linq;

namespace NHSample4.NHRepository
{
    public class Repository<T> : IRepository<T>, IDisposable
    {
        private ISession _session;
        private bool _disposed = false;

        public Repository()
        {
            _session = SingletonCore.SessionFactory.OpenSession();
            BeginTransaction();
        }

        ~Repository()
        {
            Dispose(false);
        }

        public T Get(object key)
        {
            if (!isSessionSafe) return default(T);

            return _session.Get<T>(key);
        }

        public T Save(T entity)
        {
            if (!isSessionSafe) return default(T);

            _session.Save(entity);
            return entity;
        }

        public T Update(T entity)
        {
            if (!isSessionSafe) return default(T);

            _session.Update(entity);
            return entity;
        }

        public void Delete(T entity)
        {
            if (!isSessionSafe) return;

            _session.Delete(entity);
        }

        public IQueryable<T> Find()
```



```

{
    if (!isSessionSafe) return null;
    return _session.Linq<T>();
}

public IQueryable<T> Find(System.Linq.Expressions.Expression<Func<T, bool>> predicate)
{
    if (!isSessionSafe) return null;
    return Find().Where(predicate);
}

void Commit()
{
    if (!isSessionSafe) return;

    if (_session.Transaction != null &&
        _session.Transaction.IsActive &&
        !_session.Transaction.WasCommitted &&
        !_session.Transaction.WasRolledBack)
    {
        _session.Transaction.Commit();
    }
    else
    {
        _session.Flush();
    }
}

void Rollback()
{
    if (!isSessionSafe) return;

    if (_session.Transaction != null && _session.Transaction.IsActive)
    {
        _session.Transaction.Rollback();
    }
}

private bool isSessionSafe
{
    get
    {
        return _session != null && _session.IsOpen;
    }
}

void BeginTransaction()
{
    if (!isSessionSafe) return;

    _session.BeginTransaction();
}

public void Dispose()
{
    Dispose(true);
    // tell the GC that the Finalize process no longer needs to be run for this object.
    GC.SuppressFinalize(this);
}

protected virtual void Dispose(bool disposeManagedResources)
{
    if (_disposed) return;
    if (!disposeManagedResources) return;
    if (!isSessionSafe) return;

    try
    {
        Commit();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
        Rollback();
    }
    finally
    {
        if (isSessionSafe)

```

```

        {
            _session.Close();
            _session.Dispose();
        }
    }
    _disposed = true;
}
}
}

```

اکنون جهت استفاده از این کلاس مخزن به شکل زیر می‌توان عمل کرد:

```

using System;
using System.Collections.Generic;
using NHSample4.Domain;
using NHSample4.NHRepository;

namespace NHSample4
{
    class Program
    {
        static void Main(string[] args)
        {
            //ایجاد دیتابیس در صورت نیاز
            //NHSessionManager.Config.CreateDb();

            //ابتدا یک دانشجو را اضافه می‌کنیم
            Student student = null;
            using (var studentRepo = new Repository<Student>())
            {
                student = studentRepo.Save(new Student() { Name = "Vahid" });
            }

            //سپس یک واحد را اضافه می‌کنیم
            using (var courseRepo = new Repository<Course>())
            {
                var course = courseRepo.Save(new Course() { Teacher = "Shams" });
            }

            //اکنون یک واحد را به دانشجو انتساب می‌دهیم
            using (var courseRepo = new Repository<Course>())
            {
                courseRepo.Save(new Course() { Students = new List<Student>() { student } });
            }

            //سپس شماره دروس استادی خاص را نمایش می‌دهیم
            using (var courseRepo = new Repository<Course>())
            {
                var query = courseRepo.Find(t => t.Teacher == "Shams");

                foreach (var course in query)
                    Console.WriteLine(course.Id);
            }

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

```

همانطور که ملاحظه می‌کنید در این سطح دیگر برنامه هیچ درکی از ORM مورد استفاده ندارد و پیاده سازی نحوه‌ی تعامل با NHibernate در پس کلاس مخزن مخفی شده است. کار آغاز و پایان تراکنش‌ها به صورت خودکار مدیریت گردیده و همچنین آزاد سازی منابع را نیز توسط اینترفیس IDisposable مدیریت می‌کند. به این صورت امکان فراموش شدن یک سری از اعمال متداول به حداقل رسیده، میزان کدهای تکراری برنامه کم شده و همچنین هر زمانیکه نیاز بود، صرفاً با تغییر پیاده سازی کلاس مخزن می‌توان به ORM دیگری کوچ کرد؛ بدون اینکه نیازی به بازنویسی کل برنامه وجود داشته باشد.

ادامه دارد ...

نظرات خوانندگان

نویسنده: iMAN

تاریخ: ۱۳۸۸/۰۷/۲۵ ۲۱:۵۵:۳۱

واقعاً خسته نباشید آقای نصیری، مجموعه مقالات آشنایی با NHibernate شما عالی است، باعث شد من شروع به یادگیری NHibernate کنم. ممنونم

نویسنده: Mahdi

تاریخ: ۱۳۸۸/۰۷/۲۸ ۱۰:۵۲:۲۴

دوست عزیز. سری آموزش NH واقعا یک کار نمونه و عالی هست که انجام دادید. خسته نباشید و ممنون.

راستی گویا پیدا کردن شما روی سایتهای Social مثل Twitter یا Facebook و ... کار ساده ای نیست! به هر حال از آشنایی بیشتر با شما خوشوقت خواهم شد.

نویسنده: وحید نصیری

تاریخ: ۱۳۸۸/۰۷/۲۸ ۱۵:۰۱:۲۹

با سلام

و با تشکر از لطف دوستان.

بله. بنده در سایتهای social به دلایل شخصی حضور ندارم. از لطف شما سپاسگزارم.

نویسنده: Majid

تاریخ: ۱۳۸۸/۱۱/۲۲ ۲۱:۲۸:۰۸

جناب نصیری عزیز

از شما به خاطر مطالب بسیار مفیدتان قدردانی می‌کنم

اگر امکان دارد طریقه استفاده از NHibernate در Windows Form (نمایش Objectها در Grid و کار با ابزارهای گزارش‌گیری و ...) را نیز آموزش دهید. پاینده باشید.

نویسنده: Ahmad

تاریخ: ۱۳۸۹/۰۷/۲۲ ۰۱:۴۳:۳۳

سلام.

فرض کنید همچین کدی نوشته ایم:

```
((using (var repository = new Repository
    }
    try
    }
    ... Some Code //
    ;(repository.Update(myClass
    Or //
    ;(repository.Delete(myClass
    ;return true
```

{

```
(catch (Exception
```

```
})
```

```
MessageBox.Show("خطا در ویرایش یا حذف");
```

```
{
```

```
{
```

در صورت بروز خطا هیچ وقت بلاک catch اجرا نمی شود.

البته هنگام return true متد Dispose کلاس Repository اجرا می شود.(نوش دارو پس از مرگ سهراب...)

نویسنده: وحید نصیری

تاریخ: ۱۳۸۹/۰۷/۲۲ ۱۱:۳۸:۴۵

سلام،

کاری را که شما دارید انجام می‌دید اشتباه است.

Using statement در سی شارپ به صورت خودکار به try/finally به همراه dispose شیء احاطه شده توسط آن ترجمه

می‌شود. (+)

به عبارتی شما یک try/finally را در یک سطح بالاتر دارید و داخل آن یک try/catch قرار داده‌اید. اینکار صحیح نیست. Using را حذف کنید و try/catch/finally را جایگزین تمام موارد اضافه شده کنید.

ضمناً توصیه من این است که فقط try/catch را حذف کنید. Using سرچایش باشد تا هدف اصلی آن یعنی dispose اشیاء مرتبط حتماً رخ دهد.

به تمام برنامه نویسی‌ها آموزش داده می‌شود که exception handling چیست اما در پایان فصل به آن‌ها آموخته نمی‌شود که لطفاً تا حد امکان از آن استفاده نکنید! بله، لطفاً در استفاده از آن خساست به خرج دهید. چرا؟ چون کرش بر خلاف تصور عمومی چیز خوبی است! زمانیکه شما این try/catch را قرار دادید، flow برنامه متوجه نخواهد شد که در مرحله‌ی قبل مشکلی رخ داده و از ادامه برنامه و خسارت وارد کردن به سیستم جلوگیری کند. ضمناً این را هم به خاطر داشته باشید که exception ها در دات نت حبایی هستند. یعنی به فراخوان خود منتشر خواهند شد و در یک سطح بالاتر هم قابل catch هستند (با تمام جزئیات).

نویسنده: Ahmad

تاریخ: ۱۳۸۹/۰۷/۲۲ ۱۹:۵۶:۴۳

با تشکر از تذکر شما. ولی کد زیر هم در صورت بروز خطا چیزی را برنمی گرداند. یعنی متد Delete چه با موفقیت به پایان برسد یا خطایی در حذف رکورد رخ دهد باز return true اجرا می شود.؟!

```
((using(var repository = new Repository
```

```
})
```

```
;(repository.Delete(myClass
```

```
;return true
```

```
{
```

نویسنده: وحید نصیری

تاریخ: ۱۳۸۹/۰۷/۲۲ ۲۱:۲۱:۵۷

این مورد اصلاً ربطی به try/catch , using و غیره ندارد. نیاز به solution کار شما است (با تمام کلاس‌ها و نگاشت و غیره) تا بتوان آن‌را دیباگ کرد. بهترین روش هم این است که خروجی SQL تولید شده را بررسی کنید تا متوجه شوید مشکل کار در کجاست.

نویسنده: وحید نصیری

تاریخ: ۱۳۸۹/۰۷/۲۲ ۲۲:۵۸:۲۲

ابزار حرفه‌ای مشاهده خروجی NHibernate برنامه زیر است (که در جهت دیباگ کار بسیار مفید است):

[NHProf](#)

کار کردن با آن هم بسیار ساده است. فایل How to use.txt آن را مطالعه کنید..

نویسنده: A

تاریخ: ۲۳:۵۱:۵۷ ۱۳۸۹/۰۹/۱۳

اگر نیاز به Transaction داشته باشیم در این مدل حتماً باید از TransactionScope استفاده کنیم؟ اگر این طور است فکر می‌کنم این مدل ضعیف باشد.

فکر می‌کنم (انتظار من این است) وقتی ORM وجود دارد باید بتوانیم کارها را در صف نگه داشته و یکجا اعمال کنیم.

البته منظورم این است که باید بتوان بین چند Table جداگانه (با ارتباط یا بی ارتباط) این کار را انجام داد.

من یک مدل در آورده‌ام که تا کنون نیاز من را بدون استفاده از TransactionScope برای کارهای تراکنشی برطرف کرده. شبیه همین مدل است با کمی تغییر. نمی‌دانم آیا می‌توانم آنرا مدل Repository بنامم یا خیر؟

نویسنده: وحید نصیری

تاریخ: ۰۰:۰۳:۰۹ ۱۳۸۹/۰۹/۱۴

حق با شما است. روش صحیح لایه بندی این قسمت بر اساس تعریف unit of work و سپس repository است. یک unit of work می‌تواند از اعمال حاصل چندین repository تشکیل شده و نهایتاً در پایان کار همه را یکجا اعمال کند. در مثال فوق این دو مفهوم با هم تلفیق شده.

بنابراین اگر علمی‌تر می‌خواهید کار کنید در مورد unit of work تحقیق کنید (در سایت nhforge.org).

نویسنده: A

تاریخ: ۰۸:۴۵:۰۳ ۱۳۸۹/۰۹/۱۴

بله در پشت صحنه مدلی که گفتم شبیه UoW است. اما برنامه‌نویس تقریباً مانند Repository با آن کار می‌کند و خیلی از مسائل در پشت صحنه برای او حل می‌شود.

خیلی ممنون از اطلاعات. وبلاگتان هم بسیار عالیست.

نویسنده: shayan

تاریخ: ۱۶:۵۶:۴۸ ۱۳۸۹/۱۰/۲۸

با سلام،
فرض کنید در Table CoursesToStudents فیلدی به نام IsApproved را می‌خواهیم داشته باشیم، در اینصورت کلاس‌های نگاشت به چه صورت خواهد بود؟ در کدام کلاس نگاشت پیاده سازی می‌شود؟ اگر کلاس جداگانه ایی تعریف کنیم آیا باز هم رابطه ManyToMany برقرار خواهد بود؟

با تشکر

نویسنده: وحید نصیری

تاریخ: ۱۸:۱۵:۵۳ ۱۳۸۹/۱۰/۲۸

سلام،
زمانیکه با ORM هایی از نوع Code First کار می‌کنید مثل NHibernate یا مثل نگارش بعدی Entity framework، ذهن خودتون رو از وجود جداول حاضر در بانک اطلاعاتی خالی کنید. جدولی به نام CoursesToStudents توسط ساز و کار درونی NHibernate مدیریت خواهد شد و لزومی ندارد برنامه در مورد آن اطلاعاتی داشته باشد.

موردی را که شما نیاز دارید کلاسی است به نام نتایج دوره؛ مثلاً چیزی به نام CourseResult. این کلاس ارجاعاتی را به شیء دانشجو و شیء دوره دارد، به همراه نمره نهایی یا مثلاً خاصیت قبول شده و برای مثال تاریخ امتحان و خواص دیگری که صلاح

می‌دانید.

زمانیکه NHibernate اسکریپت اعمال این نگاشت‌ها را تشکیل دهد (توسط امکانات کلاس SchemaExport که در مطلب بالا ذکر شده)، در جدول نهایی بانک اطلاعاتی شما به ازای ارجاعات به اشیاء یاد شده، یک کلید خارجی خواهید داشت. این کلاس جدید تاثیری روی سایر روابط ندارد.

نویسنده: Amir

تاریخ: ۱۳۸۹/۱۲/۰۱ ۱۳:۲۴:۳۷

سلام با تشکر از مطلب خوبتون

لینک زیر نحوه ترکیب Repository با EF رو توضیح داده که البته ادامه داره و در پست های بعدی مطلب رو تکمیل و به روز کرده
/http://huyrua.wordpress.com/2010/07/13/entity-framework-4-poco-repository-and-specification-pattern

نویسنده: وحید نصیری

تاریخ: ۱۳۸۹/۱۲/۰۱ ۱۵:۲۴:۳۴

ممنون. الگوهای طراحی برنامه نویسی شیءگرا یک حالت عمومی دارند. یعنی مختص به یک فناوری یا زبان خاص یا حتی یک محصول خاص نیستند. بگردید برای LINQ to SQL هم پیاده سازی الگوی Repository وجود دارد. کلا استفاده‌ی از هر کدام از ORMs موجود بدون پیاده سازی الگوی Repository اشتباه است. به چند دلیل:

- مخفی کردن ساز و کار درونی یک ORM: برای مثال من جدا قصد ندارم این رو حفظ کنم که فلان ORM خاص چطور Insert انجام می‌دهد. من فقط می‌خواهم یک متد Insert داشته باشم. یکبار این رو در الگوی Repository پیاده سازی می‌کنم و بعد فراموش می‌کنم که این ORM الان EF است یا NH یا هرچی
- امکان تعویض کلی یک ORM: زمانیکه من در کدهای BLL خودم فقط از متد Insert پیاده سازی شده مطابق رهنمون‌های الگوی Repository استفاده کردم، دیگر BLL درکی از ORM نخواهد داشت. برای کوچ کردن به یک ORM دیگر فقط کافی است تا Repository را عوض کرد. مابقی برنامه دست نخورده باقی می‌ماند.
- نوشتن Unit test با استفاده از الگوی Repository ساده‌تر است: این الگو چون بر مبنای یک Interface پیاده سازی می‌شود، امکان Mocking این Interface در Unit tests ساده‌تر است.

نویسنده: شاهین کیاست

تاریخ: ۱۳۸۹/۱۲/۰۳ ۲۰:۴۹:۳۱

سلام.

من تا قبل فکر می کردم برای گرفتن کارایی از یک کلاس که اعمال آن پیچیده هست مثل همین Insert در یک ORM که مثال زدید باید از لایه Facade استفاده کرد.

لطفا اگر ممکن هست در رابطه با فرق این 2 یک توضیح بدید.

ممنون

نویسنده: وحید نصیری

تاریخ: ۱۳۸۹/۱۲/۰۳ ۲۲:۰۳:۲۳

برای اینکه از بحث دور نشیم، در NHibernate این الگوها قابل مشاهده است:

الگوی Facade که همان session و ISession معروف آن است و کار آن فراخوانی تعداد قابل ملاحظه‌ای زیر سیستم و مخفی کردن آن‌ها از دید کاربر نهایی است. به این صورت شما توانایی کار کردن با انواع بانک‌های اطلاعاتی را بدون درگیر شدن با جزئیات آن‌ها از طریق یک اینترفیس عمومی پیدا می‌کنید.

الگوی proxy که پایه و اساس lazy loading آن است.

الگوی object pool جهت مدیریت اتصالات آن به بانک اطلاعاتی

الگوی Interpreter جهت مدیریت کوثری‌های ویژه آن

و ...

الگوی Repository هم یک نوع نگارش سفارشی الگوی Facade است. در اینجا یک سیستم پیچیده (یک سطح بالاتر است از ISession که کارش مخفی کردن ساختار داخلی NHibernate است) یا همان ORM مورد نظر را دریافت کرده و یک اینترفیس ساده، قابل درک و عمومی را از آن را ارائه می‌دهد. هدف آن مخفی کردن ریز جزئیات روش کار با یک ORM خاص است. به همین جهت به آن Persistence Ignorance هم گفته می‌شود.

الگوی Repository یکی از الگوهای اصلی Domain driven design یا DDD است.

نویسنده: Hamidrezabina
تاریخ: ۱۳۹۰/۰۲/۰۱ ۱۹:۴۰:۴۲

با سلام . . .

چطوری میشه تمام مواردی که می‌خواهیم ثبت کنیم بر اساس یه transaction کار کنند ؟
مثلا من یه فاکتور فروش دارم که تمام موارد باید با هم ثبت بشوند یا هیچکدوم نشوند.
میشه بفهمائید چطوری باید پیاده سازیش کرد ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۲/۰۱ ۲۰:۴۵:۵۷

پیاده سازی الگوی مخزن در مطلب بالا از دیدی که مطرح کردید ایراد دارد. چون به ازای هر موجودیت یک تراکنش لحاظ می‌کند. روش صحیح پیاده سازی مورد نظر شما استفاده از الگوی unit of work است این الگو یک سطح بالاتر از الگوی مخزن قرار می‌گیرد اگر می‌خواهید با نحوه پیاده سازی آن آشنا شوید به این پروژه مراجعه کنید

[/http://efrepository.codeplex.com](http://efrepository.codeplex.com)

هر چند برای EF نوشته شده ولی از دیدگاه طراحی اینترفیس و روابط نهایی برای تمام ORM های دیگر هم صادق است و فرقی نمی‌کند

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۲/۰۳ ۱۶:۳۸:۵۸

جهت یادآوری...

در مورد unit of work در سایت <http://nhforge.org> جستجو کنید.

استفاده از Log4Net جهت ثبت خروجی‌های SQL حاصل از NHibernate

هنگام استفاده از NHibernate، پس از افزودن ارجاعات لازم به اسمبلی‌های مورد نیاز آن به برنامه، یکی از اسمبلی‌هایی که به پوشه build برنامه به صورت خودکار کپی می‌شود، فایل log4net.dll است (حتی اگر ارجاعی را به آن اضافه نکرده باشیم) که جهت ثبت وقایع مرتبط با NHibernate مورد استفاده قرار می‌گیرد. خوب اگر مجبوریم که این وابستگی کتابخانه NHibernate را نیز در پروژه‌های خود داشته باشیم، چرا از آن استفاده نکنیم؟! شرح مفصل استفاده از این کتابخانه سورس باز را در سایت اصلی آن می‌توان مشاهده کرد:

[Log4Net](#)

برای اینکه از این کتابخانه در برنامه خود جهت ثبت عبارات SQL تولیدی توسط NHibernate استفاده کنیم، باید مراحل زیر طی شوند:

الف) ارجاعی را به اسمبلی log4net.dll اضافه نمائید (کنار اسمبلی NHibernate در پوشه build برنامه باید موجود باشد)
 ب) فایل app.config برنامه را (برنامه ویندوزی) به صورت زیر ویرایش کرده و چند سطر مربوطه را اضافه نمائید (در مورد برنامه‌های وب هم به همین شکل است). configSections فایل web.config تنظیم شده و سپس تنظیمات log4net را قبل از بسته شدن تگ configuration اضافه نمائید) :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="log4net"
      type="log4net.Config.Log4NetConfigurationSectionHandler,log4net" />
  </configSections>

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

  <log4net>
    <appender name="rollingFile"
      type="log4net.Appender.RollingFileAppender,log4net" >
      <param name="File" value="NHibernate_Log.txt" />
      <param name="AppendToFile" value="true" />
      <param name="DatePattern" value="yyyy.MM.dd" />
      <rollingStyle value="Size" />
      <maxSizeRollBackups value="10" />
      <maximumFileSize value="500KB" />
      <staticLogFileName value="true" />
      <layout type="log4net.Layout.PatternLayout,log4net">
        <conversionPattern value="%d %p %m%n" />
      </layout>
    </appender>
    <logger name="NHibernate.SQL">
      <level value="ALL" />
      <appender-ref ref="rollingFile" />
    </logger>
  </log4net>
</configuration>
```

ج) سپس باید فراخوانی زیر نیز در ابتدای کار برنامه صورت گیرد:

```
log4net.Config.XmlConfigurator.Configure();
```

در یک برنامه ASP.Net این فراخوانی باید در Application_Start فایل Global.asax.cs صورت گیرد. یا در یک برنامه از نوع WinForms تنها کافی است سطر زیر را به فایل AssemblyInfo.cs برنامه اضافه کرد:

```
// Configure log4net using the .config file
[assembly: log4net.Config.XmlConfigurator(Watch = true)]
```

یا این سطر را به فایل Global.asax.cs یک برنامه ASP.Net نیز می‌توان اضافه کرد. Watch=true آن، با کمک FileSystemWatcher تغییرات فایل کانفیگ را تحت نظر داشته و هر بار که تغییر کند بلافاصله، تغییرات جدید را اعمال خواهد کرد.

د) هنگام استفاده از کتابخانه Fluent NHibernate حتما باید متد ShowSql در جایی که دیتابیس برنامه را تنظیم می‌کنیم (Fluently.Configure()).Database) ذکر گردد (که نمونه آن را در مثال‌های قسمت‌های قبل مشاهده کرده‌اید).

توضیحاتی در مورد تنظیمات فوق:

configSections حتما باید در ابتدای فایل app.config ذکر شود و گر نه برنامه کار نخواهد کرد.

سپس کانکشن استرینگ مورد استفاده در قسمت کانفیگ برنامه ذکر شده است.

در ادامه تنظیمات استاندارد مربوط به log4net را مشاهده می‌کنید.

در تنظیمات این کتابخانه، appender مشخص کننده محل ثبت وقایع است. زمانی که از RollingFileAppender استفاده کنیم، اطلاعات را در یک سری فایل ذخیره خواهد کرد (امکان ثبت وقایع در EventLog ویندوز، ارسال از طریق ایمیل و غیره نیز میسر است که جهت توضیحات بیشتر می‌توان به مستندات آن رجوع نمود).

سپس نام فایلی که اطلاعات وقایع در آن ثبت خواهند شد ذکر شده است (برای مثال NHibernate_Log.txt)، در ادامه مشخص گردیده که اطلاعات باید هر بار به این فایل Append و اضافه شوند. سطرهای بعدی مشخص می‌کنند که هر زمانیکه این لاگ فایل به 10 مگابایت رسید، یک فایل جدید تولید کن و هر بار 10 فایل آخر را نگه دار و مابقی فایل‌های قدیمی را حذف کن. در قسمت PatternLayout مشخصات می‌کنیم که خروجی ثبت شده با چه فرمتی باشد. برای مثال یک سطر خروجی مطابق با تنظیمات فوق به شکل زیر خواهد بود:

```
2009-10-18 20:03:54,187 DEBUG INSERT INTO [Student] (Name) VALUES (@p0); select SCOPE_IDENTITY();@p0 = 'Vahid'
```

در قسمت Logger یک نام دلخواه ذکر شده و میزان اطلاعاتی که باید درج شود، از طریق مقدار level مورد نظر، قابل تنظیم است که می‌تواند یکی از مقادیر ALL, DEBUG, INFO, WARN, ERROR, FATAL باشد. اینجا level در نظر گرفته شده ALL است که تمامی اطلاعات مرتبط با اعمال پشت صحنه NHibernate را لاگ خواهد کرد. توسط appender-ref آن appender ای را که در ابتدای کار تعریف و تنظیم کردیم، مشخص خواهیم کرد.

اگر هم با برنامه نویسی بخواهیم اطلاعاتی را به این لاگ فایل اضافه کنیم تنها کافی است بنویسیم:

```
log4net.LogManager.GetLogger("NHibernate.SQL").Info("test1");
```

[اطلاعات بیشتر](#)

ادامه دارد ...

عنوان: آشنایی با NHibernate - قسمت دهم

نویسنده: وحید نصیری

تاریخ: ۱۳۸۸/۰۷/۲۸ ۱۸:۰۳:۰۰

آدرس: www.dotnettips.info

برچسب‌ها: NHibernate

آشنایی با کتابخانه NHibernate Validator

پروژه جدیدی به پروژه NHibernate Contrib در سایت سورس فورج اضافه شده است به نام NHibernate Validator که از آدرس زیر قابل دریافت است:

<http://sourceforge.net/projects/nhcontrib/files/NHibernate.Validator>

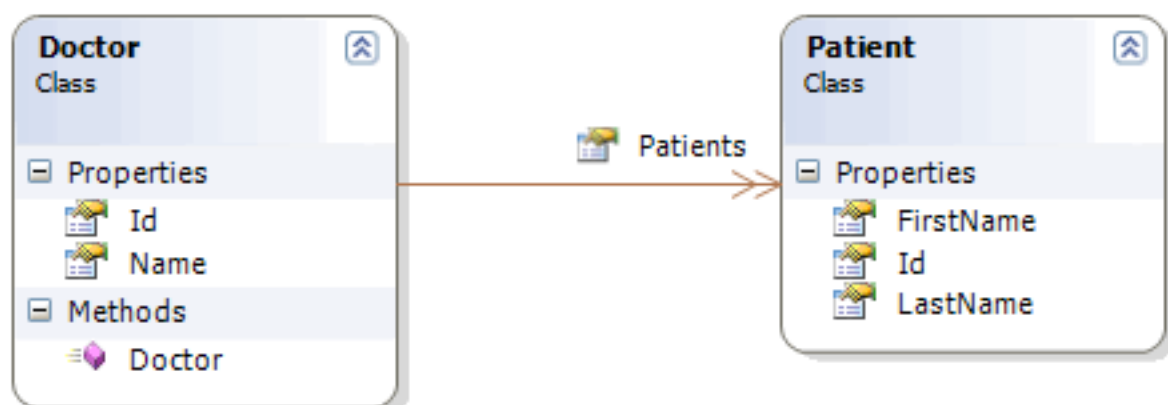
این پروژه که توسط [Dario Quintana](#) توسعه یافته است، امکان اعتبار سنجی اطلاعات را پیش از افزوده شدن آن‌ها به دیتابیس به دو صورت دستی و یا خودکار و یکپارچه با NHibernate فراهم می‌سازد؛ که امروز قصد بررسی آن‌را داریم.

کامپایل پروژه اعتبار سنجی NHibernate

پس از دریافت آخرین نگارش موجود کتابخانه NHibernate Validator از سایت سورس فورج، فایل پروژه آن‌را در VS.Net گشوده و یکبار آن‌را کامپایل نمائید تا فایل اسمبلی NHibernate.Validator.dll حاصل گردد.

بررسی مدل برنامه

در این مدل ساده، تعدادی پزشک داریم و تعدادی بیمار. در سیستم ما هر بیمار تنها توسط یک پزشک مورد معاینه قرار خواهد گرفت. رابطه آن‌ها را در کلاس دیاگرام زیر می‌توان مشاهده نمود:



به این صورت پوشه دومین برنامه از کلاس‌های زیر تشکیل خواهد شد:

```
namespace NHSample5.Domain
{
    public class Patient
```

```
{
    public virtual int Id { get; set; }
    public virtual string FirstName { get; set; }
    public virtual string LastName { get; set; }
}
```

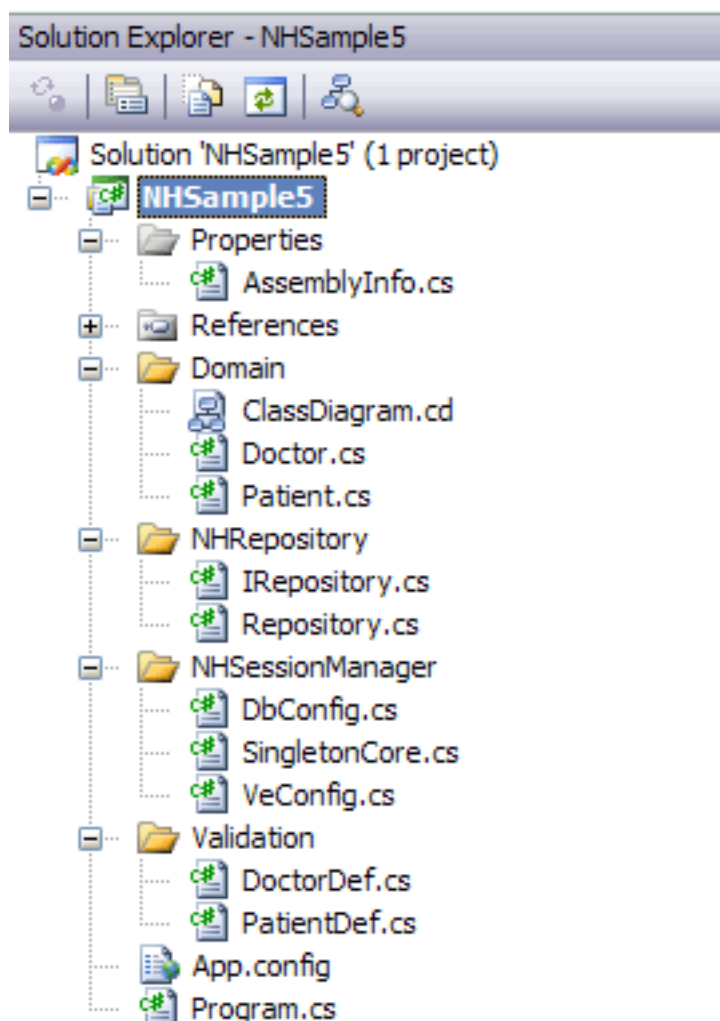
```
using System.Collections.Generic;

namespace NHSample5.Domain
{
    public class Doctor
    {
        public virtual int Id { get; set; }
        public virtual string Name { get; set; }
        public virtual IList<Patient> Patients { get; set; }

        public Doctor()
        {
            Patients = new List<Patient>();
        }
    }
}
```

برنامه این قسمت از نوع کنسول با ارجاعاتی به اسمبلی‌های ،NHibernate.dll ،log4net.dll ،FluentNHibernate.dll و NHibernate.ByteCode.Castle.dll ،NHibernate.Linq.dll ،NHibernate.Validator.dll است.

ساختار کلی این پروژه را در شکل زیر مشاهده می‌کنید:



اطلاعات این برنامه بر مبنای NHRepository و NHSessionManager ایی است که در قسمت‌های قبل توسعه دادیم و پیشنهاد ضروری مطالعه آن می‌باشند (سورس پیوست شده شامل نمونه تکمیل شده این موارد نیز هست). همچنین از قسمت ایجاد دیتابیس از روی مدل نیز صرف‌نظر می‌شود و همانند قسمت‌های قبل است.

تعریف اعتبار سنجی دومین با کمک ویژگی‌ها (attributes)

فرض کنید می‌خواهیم بر روی طول نام و نام خانوادگی بیمار محدودیت قرار داده و آن‌ها را با کمک کتابخانه NHibernate Validator، اعتبار سنجی کنیم. برای این منظور ابتدا فضای نام NHibernate.Validator.Constraints به کلاس بیمار اضافه شده و سپس با کمک ویژگی‌هایی که در این کتابخانه تعریف شده‌اند می‌توان قیود خود را به خواص کلاس تعریف شده اعمال نمود که نمونه‌ای از آن را مشاهده می‌نمائید:

```
using NHibernate.Validator.Constraints;

namespace NHSample5.Domain
{
    public class Patient
    {
        public virtual int Id { get; set; }

        [Length(Min = 3, Max = 20, Message = "طول نام باید بین 3 و 20 کاراکتر باشد")]
        public virtual string FirstName { get; set; }

        [Length(Min = 3, Max = 60, Message = "طول نام خانوادگی باید بین 3 و 60 کاراکتر باشد")]
        public virtual string LastName { get; set; }
    }
}
```

اعمال این قیود از این جهت مهم هستند که نباید وقت برنامه و سیستم را با دریافت خطای نهایی از دیتابیس تلف کرد. آیا بهتر نیست قبل از اینکه اطلاعات به دیتابیس وارد شوند و رفت و برگشتی در شبکه صورت گیرد، مشخص گردد که این فیلد حتما نباید خالی باشد یا طول آن باید دارای شرایط خاصی باشد و امثال آن؟

مثالی دیگر:

جهت اجباری کردن و همچنین اعمال Regular expressions برای اعتبار سنجی یک فیلد می‌توان دو ویژگی زیر را به بالای آن فیلد مورد نظر افزود:

```
[NotNull]
[Pattern(Regex = "[A-Za-z0-9]+")]
```

تعریف اعتبار سنجی با کمک کلاس ValidationDef

راه دوم تعریف اعتبار سنجی، کمک گرفتن از کلاس ValidationDef این کتابخانه و استفاده از روش fluent configuration است. برای این منظور، پوشه جدیدی را به برنامه به نام Validation اضافه خواهیم کرد و سپس دو کلاس DoctorDef و PatientDef را به آن به صورت زیر خواهیم افزود:

```
using NHibernate.Validator.Cfg.Loquacious;
using NHSample5.Domain;

namespace NHSample5.Validation
{
    public class DoctorDef : ValidationDef<Doctor>
    {
        public DoctorDef()
        {
            Define(x => x.Name).LengthBetween(3, 50);
            Define(x => x.Patients).NotNullableAndNotEmpty();
        }
    }
}
```

```

    }
}

using NHSample5.Domain;
using NHibernate.Validator.Cfg.Loquacious;

namespace NHSample5.Validation
{
    public class PatientDef : ValidationDef<Patient>
    {
        public PatientDef()
        {
            Define(x => x.FirstName)
                .LengthBetween(3, 20)
                .WithMessage("طول نام باید بین 3 و 20 کاراکتر باشد");

            Define(x => x.LastName)
                .LengthBetween(3, 60)
                .WithMessage("طول نام خانوادگی باید بین 3 و 60 کاراکتر باشد");
        }
    }
}

```

استفاده از قیودات تعریف شده به صورت دستی

می‌توان از این کتابخانه اعتبار سنجی به صورت مستقیم نیز اضافه کرد. روش انجام آن‌را در متد زیر مشاهده می‌نمائید.

```

/// <summary>
/// استفاده از اعتبار سنجی ویژه به صورت مستقیم
/// در صورت استفاده از ویژگی‌ها
/// </summary>
static void WithoutConfiguringTheEngine()
{
    // تعریف یک بیمار غیر معتبر
    var patient1 = new Patient() { FirstName = "V", LastName = "N" };
    var ve = new ValidatorEngine();
    var invalidValues = ve.Validate(patient1);
    if (invalidValues.Length == 0)
    {
        Console.WriteLine("patient1 is valid.");
    }
    else
    {
        Console.WriteLine("patient1 is NOT valid!");
        // نمایش پیغام‌های تعریف شده مربوط به هر فیلد
        foreach (var invalidValue in invalidValues)
        {
            Console.WriteLine(
                "{0}: {1}",
                invalidValue.PropertyName,
                invalidValue.Message);
        }
    }

    // تعریف یک بیمار معتبر بر اساس قیودات اعمالی
    var patient2 = new Patient() { FirstName = "وحید", LastName = "نصیری" };
    if (ve.IsValid(patient2))
    {
        Console.WriteLine("patient2 is valid.");
    }
    else
    {
        Console.WriteLine("patient2 is NOT valid!");
    }
}

```

ابتدا شیء ValidatorEngine تعریف شده و سپس متد Validate آن بر روی شیء بیماری غیر معتبر فراخوانی می‌گردد. در صورتیکه این اعتبار سنجی با موفقیت روبرو نشود، خروجی این متد آرایه‌ای خواهد بود از فیلدهای غیرمعتبر به همراه پیغام‌هایی که برای آن‌ها تعریف کرده‌ایم. یا می‌توان به سادگی همانند بیمار شماره دو، تنها از متد IsValid آن نیز استفاده کرد.

در اینجا اگر سعی در اعتبار سنجی یک پزشک نمائیم، نتیجه‌ای حاصل نخواهد شد زیرا هنگام استفاده از کلاس `ValidationDef`، باید نگاشت لازم به این قیودات را نیز دقیقاً مشخص نمود تا مورد استفاده قرار گیرد که نحوه‌ی انجام این عملیات را در متد زیر می‌توان مشاهده نمود.

```
public static ValidatorEngine GetFluentlyConfiguredEngine()
{
    var vtor = new ValidatorEngine();
    var configuration = new FluentConfiguration();
    configuration
        .Register(
            Assembly
                .GetExecutingAssembly()
                .GetTypes()
                .Where(t => t.Namespace.Equals("NHSample5.Validation"))
                .ValidationDefinitions()
        )
        .SetDefaultValidatorMode(ValidatorMode.UseExternal);
    vtor.Configure(configuration);
    return vtor;
}
```

`FluentConfiguration` آن مجزا است از نمونه مشابه کتابخانه `Fluent NHibernate` و نباید با آن اشتباه گرفته شود (در فضای نام `NHibernate.Validator.Cfg.Loquacious` تعریف شده است).

در این متد کلاس‌های قرار گرفته در پوشه `Validation` برنامه که دارای فضای نام `NHSample5.Validation` هستند، به عنوان کلاس‌هایی که باید اطلاعات لازم مربوط به اعتبار سنجی را از آنان دریافت کرد معرفی شده‌اند. همچنین `ValidatorMode` نیز به صورت `External` تعریف شده و منظور از `External` در اینجا هر چیزی بجز استفاده از روش بکارگیری `attributes` است (علاوه بر امکان تعریف این قیودات در یک پروژه `class library` مجزا و مشخص ساختن اسمبلی آن در اینجا).

اکنون جهت دسترسی به این موتور اعتبار سنجی تنظیم شده می‌توان به صورت زیر عمل کرد:

```
/// <summary>
/// استفاده از اعتبار سنجی ویژه به صورت مستقیم
/// در صورت تعریف آن‌ها با کمک
/// ValidationDef
/// </summary>
static void WithConfiguringTheEngine()
{
    var ve2 = VeConfig.GetFluentlyConfiguredEngine();
    var doctor1 = new Doctor() { Name = "S" };
    if (ve2.IsValid(doctor1))
    {
        Console.WriteLine("doctor1 is valid.");
    }
    else
    {
        Console.WriteLine("doctor1 is NOT valid!");
    }

    var patient1 = new Patient() { FirstName = "وحید", LastName = "نصیری" };
    if (ve2.IsValid(patient1))
    {
        Console.WriteLine("patient1 is valid.");
    }
    else
    {
        Console.WriteLine("patient1 is NOT valid!");
    }

    var doctor2 = new Doctor() { Name = "شمس", Patients = new List<Patient>() { patient1 } };
    if (ve2.IsValid(doctor2))
    {
        Console.WriteLine("doctor2 is valid.");
    }
    else
    {

```

```

        Console.WriteLine("doctor2 is NOT valid!");
    }
}

```

نکته مهم:

فراخوانی `GetFluentlyConfiguredEngine` نیز باید یکبار در طول برنامه صورت گرفته و سپس حاصل آن بارها مورد استفاده قرار گیرد. بنابراین نحوه‌ی صحیح دسترسی به آن باید حتماً از طریق الگوی `Singleton` که در قسمت‌های قبل در مورد آن بحث شد، انجام شود.

استفاده از قیودات تعریف شده و سیستم اعتبار سنجی به صورت یکپارچه با NHibernate

کتابخانه `NHibernate Validator` زمانی که با NHibernate یکپارچه گردد دو رخداد `PreInsert` و `PreUpdate` آن را به صورت خودکار تحت نظر قرار داده و پیش از اینکه اطلاعات ثبت و یا به روز شوند، ابتدا کار اعتبار سنجی خود را انجام داده و اگر اعتبار سنجی مورد نظر با شکست مواجه شود، با ایجاد یک `exception` از ادامه برنامه جلوگیری می‌کند. در این حالت استثنای حاصل شده از نوع `InvalidStateException` خواهد بود.

برای انجام این مرحله یکپارچه سازی ابتدا متد `BuildIntegratedFluentlyConfiguredEngine` را به شکل زیر باید فراخوانی نمائیم:

```

/// <summary>
/// از این کانفیگ برای آغاز سشن فکتوری باید کمک گرفته شود
/// </summary>
/// <param name="nhConfiguration"></param>
public static void BuildIntegratedFluentlyConfiguredEngine(ref Configuration nhConfiguration)
{
    var vtor = new ValidatorEngine();
    var configuration = new FluentConfiguration();
    configuration
        .Register(
            Assembly
                .GetExecutingAssembly()
                .GetTypes()
                .Where(t => t.Namespace.Equals("NHSample5.Validation"))
                .ValidationDefinitions()
        )
        .SetDefaultValidatorMode(ValidatorMode.UseExternal)
        .IntegrateWithNHibernate
        .ApplyingDDLConstraints()
        .And
        .RegisteringListeners();
    vtor.Configure(configuration);

    //Registering of Listeners and DDL-applying here
    ValidatorInitializer.Initialize(nhConfiguration, vtor);
}

```

این متد کار دریافت `Configuration` مرتبط با NHibernate را جهت اعمال تنظیمات اعتبار سنجی به آن انجام می‌دهد. سپس از `nhConfiguration` تغییر یافته در این متد جهت ایجاد سشن فکتوری استفاده خواهیم کرد (در غیر اینصورت سشن فکتوری درکی از اعتبار سنجی‌های تعریف شده نخواهد داشت). اگر قسمت‌های قبل را مطالعه کرده باشید، کلاس `SingletonCore` را جهت مدیریت بهینه‌ی سشن فکتوری به خاطر دارید. این کلاس اکنون باید به شکل زیر وصله شود:

```

SingletonCore()
{
    Configuration cfg = DbConfig.GetConfig().BuildConfiguration();
    VeConfig.BuildIntegratedFluentlyConfiguredEngine(ref cfg);
    //همان کانفیگ تنظیم شده برای اعتبار سنجی باید کار شروع شود
    _sessionFactory = cfg.BuildSessionFactory();
}

```


از این لحظه به بعد، نیاز به فراخوانی متدهای Validate و IsValid نبوده و کار اعتبار سنجی به صورت خودکار و یکپارچه با NHibernate انجام می‌شود. لطفاً به مثال زیر دقت فرمائید:

```
/// <summary>
/// استفاده از اعتبار سنجی یکپارچه و خودکار
/// </summary>
static void tryToSaveInvalidPatient()
{
    using (Repository<Patient> repo = new Repository<Patient>())
    {
        try
        {
            var patient1 = new Patient() { FirstName = "V", LastName = "N" };
            repo.Save(patient1);
        }
        catch (InvalidOperationException ex)
        {
            Console.WriteLine("Validation failed!");
            foreach (var invalidValue in ex.GetInvalidValues())
                Console.WriteLine(
                    "{0}: {1}",
                    invalidValue.PropertyName,
                    invalidValue.Message);
            log4net.LogManager.GetLogger("NHibernate.SQL").Error(ex);
        }
    }
}

/// <summary>
/// استفاده از اعتبار سنجی یکپارچه و خودکار
/// </summary>
static void tryToSaveValidPatient()
{
    using (Repository<Patient> repo = new Repository<Patient>())
    {
        var patient1 = new Patient() { FirstName = "Vahid", LastName = "Nasiri" };
        repo.Save(patient1);
    }
}
```

در اینجا از کلاس Repository که در قسمت‌های قبل توسعه دادیم، استفاده شده است. در متد tryToSaveInvalidPatient، بدلیل استفاده از تعریف بیماری غیرمعتبر، پیش از انجام عملیات ثبت، استثنایی حاصل شده و پیش از هرگونه رفت و برگشتی به دیتابیس، سیستم از بروز این مشکل مطلع خواهد شد. همچنین پیغام‌هایی را که هنگام تعریف قیودات مشخص کرده بودیم را نیز توسط آرایه ex.GetInvalidValues می‌توان دریافت کرد.

نکته:

اگر کار ساخت database schema را با کمک کانفیگ تنظیم شده توسط کتابخانه اعتبار سنجی آغاز کنیم، طول فیلدها دقیقاً مطابق با حداکثر طول مشخص شده در قسمت تعاریف قیود هر یک از فیلدها تشکیل می‌گردد (حاصل از اعمال متد ApplyingDDLConstraints در متد BuildIntegratedFluentlyConfiguredEngine ذکر شده می‌باشد).

```
public static void CreateValidDb()
{
    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

    Configuration cfg = DbConfig.GetConfig().BuildConfiguration();
    VeConfig.BuildIntegratedFluentlyConfiguredEngine(ref cfg);
    // با همان کانفیگ تنظیم شده برای اعتبار سنجی باید کار شروع شود

    new SchemaExport(cfg).Execute(script, export, dropTables);
}
```


نظرات خوانندگان

نویسنده: Nima

تاریخ: ۱۳۸۸/۰۸/۰۳ ۰۹:۱۰:۴۸

سلام.

خیلی سیستم گنگی هستش اما به نظر کارامد و سریع میاد... تمام سیمو دارم میکنم تا بفهمم قضیش چیه ;)

اگر میشد همین مثال (سفارش و مشتری ...) روکه زدید توی یک پروژه واقعی پیاده می کردید خیلی عالی میشد. آخه الان جایگاه و نحوه استفاده از این ORM بین لایه های برنامه برام جای سواله...

بهرحال از اینکه سرخ رو به دستمون دادی ممنونم

نویسنده: مهدی پایروند

تاریخ: ۱۳۸۸/۰۸/۰۹ ۱۱:۴۷:۵۸

سلام، توی <http://vahid.nasiri.googlepages.com/NH-links.txt> یه فایل به اسم Summer_20ofNHibernate_20Session_2005.avi از ریپیدشر پاک شده اگه ممکنه لینک دانلود مستقیمشو تو وبلاگ بذارین! ممنون

نویسنده: وحید نصیری

تاریخ: ۱۳۸۸/۰۸/۰۹ ۱۳:۰۵:۲۱

سلام،

خودتون هم میتونید اینکار را انجام بدید.

یک فایل در ریپیدشر آپلود کنید. سپس یک لینک به شما می دهد جهت ایجاد collectors account . این اکانت کالکتور را که ایجاد کردید، امکان remote upload هم دارد. (این نوع اکانت ها پولی نیست اما امکانات خوبی دارد و کار راه انداز است)

نویسنده: مهدی پایروند

تاریخ: ۱۳۸۸/۰۸/۰۹ ۱۳:۴۲:۰۷

منظورم اینه که از کجا میتونم خود ویدئو رو دانلود کنم. فکر کنم منظورمو خوب بیان نکردم، فایل از ریپید پاک شده!

نویسنده: وحید نصیری

تاریخ: ۱۳۸۸/۰۸/۰۹ ۱۴:۳۶:۰۱

- اشکالی نداره که پاک شده. خودتون فایل رو ترنس لود کنید. یک اکانت کالکتور در ریپید شیر درست کنید. بعد لاگین کنید. سپس به قسمت remote upload آن مراجعه کرده و لینک های زیر را بدهید تا برای شما بدون مشکل دانلود کند:

<http://vahid.nasiri.googlepages.com/summerNH.txt>

- ضمنا این بحث ارتباطی به قسمت دهم فوق ندارد...

نویسنده: مهدی پایروند

تاریخ: ۱۳۸۸/۰۸/۰۹ ۱۴:۴۳:۲۰

خیلی ممنون بابت فایل

نویسنده: Iman

تاریخ: ۱۳۸۹/۰۱/۰۴ ۰۵:۱۰:۳۶

اول سلام و خسته نباشید
مطالب این orm دنبال کردم و به این نتیجه رسیدم هنر اصلیش در ایجاد کوئری بدون توجه به نوع دیتابیس هستش و خیلی استفاده از رویه های ذخیره شده در sql پیش بینی نشده و این برای برنامه نویسی هایی که با sql و sp کار می کنند خوشایند نیست.
و در انتها یک سوال دارم.
جناب نصیری شما خودتون برای لایه دسترسی داده در پروژه شخصی خودتون از چه مودلی استفاده می کنید.

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۱/۰۴ ۱۰:۰۹:۵۴

چرا. امکان استفاده از رویه ذخیره شده رو هم داره. من در موردش مطلب ننوشتم و گرنه برای پوشش کامل آن باید کتاب تهیه می شد.

نویسنده: Majid
تاریخ: ۱۳۸۹/۰۱/۰۴ ۱۰:۴۶:۴۱

با سلام
لطفا نحوه معرفی اسمبلی در حالتیکه قیودات در یک پروژه class library مجزا تعریف شده باشد را بفرمایید.
سپاس

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۱/۰۴ ۱۱:۴۱:۴۲

تفاوتی نمی کنه. همان AddMappingsFromAssembly و FluentMappings.AddFromAssembly از قسمت چهارم به بعد است. اساس کار آن هم reflection است. در این حالت چه این تعاریف در خود پروژه باشد یا در هر پروژه دیگری که ارجاعی از آن در برنامه وجود دارد، دسترسی به آن از طریق reflection است و نه parse مستقیم کلاس های cs یا vb شما.

نویسنده: Majid
تاریخ: ۱۳۸۹/۰۱/۰۴ ۱۴:۱۱:۵۸

از پاسخگویی شما ممنونم.
آیا امکان اعتبار سنجی بصورت یکپارچه با FNH در حالتی که از ویژگی ها استفاده نمی کنیم وجود دارد؟ (در مثال شما این کار انجام نمی شود)

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۱/۰۴ ۱۶:۲۰:۳۸

لطفا قسمت آخر را مطالعه بفرمائید (استفاده از قیودات تعریف شده و سیستم اعتبار سنجی به صورت یکپارچه با NHibernate).
هم FNH است و هم با تزریقی که صورت گرفته یکپارچه شده و هم از ویژگی ها استفاده نشده.

نویسنده: Majid
تاریخ: ۱۳۸۹/۰۱/۰۴ ۱۶:۵۳:۲۹

با عرض معذرت، منظور من این است که اگر ویژگی های کلاس Patient را حذف کرده و متد tryToSaveInvalidPatient را اجرا نمایید بدون خطا ثبت می شود.

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۱/۰۴ ۱۷:۴۰:۱۰

در هر حال به یک طریقی شما باید به این کتابخانه اعلام کنید که چه چیزی را اعتبار سنجی کند. درست است؟

یا از طریق ویژگی‌ها یا به صورت دستی.
اگر ویژگی‌ها را حذف کنید اعتبار سنجی رخ نخواهد داد چون اعلامی در این زمینه صورت نگرفته. در این حالت از روش دستی یکپارچه شده می‌توان استفاده کرد (همان حالت آخر).

نویسنده: وحید نصیری
تاریخ: ۱۷:۴۸:۱۶ ۱۳۸۹/۰۱/۰۴

ضمناً یک مورد را هم اضافه کنم. این قسمت ValidationDefinitions و همچنین BuildIntegratedFluentlyConfiguredEngine بسیار مهم هستند و اگر فراموش شوند ممکن است مدتی وقت شما را برای عیب یابی تلف کنند.

نویسنده: Majid
تاریخ: ۱۸:۵۷:۱۱ ۱۳۸۹/۰۱/۰۴

از راهنمایی خوبتان متشکرم.

نویسنده: Alex
تاریخ: ۱۷:۳۶:۴۰ ۱۳۸۹/۰۱/۱۵

لام آقای نصیری
یه زمانی که شروع کردید به نوشتن در مورد NHibernate، اصلن فکر نمی‌کردم که روزی به درد من هم بخوره چون استفاده ازش نمی‌کردیم اما روزگار چرخید و چرخید تا رسید به الان و دیدم که مجبورم یادش بگیرم. امروز کل 10 قسمت مربوط به NHibernate رو خوندم و استفاده کردم. بی نهایت ممنون.

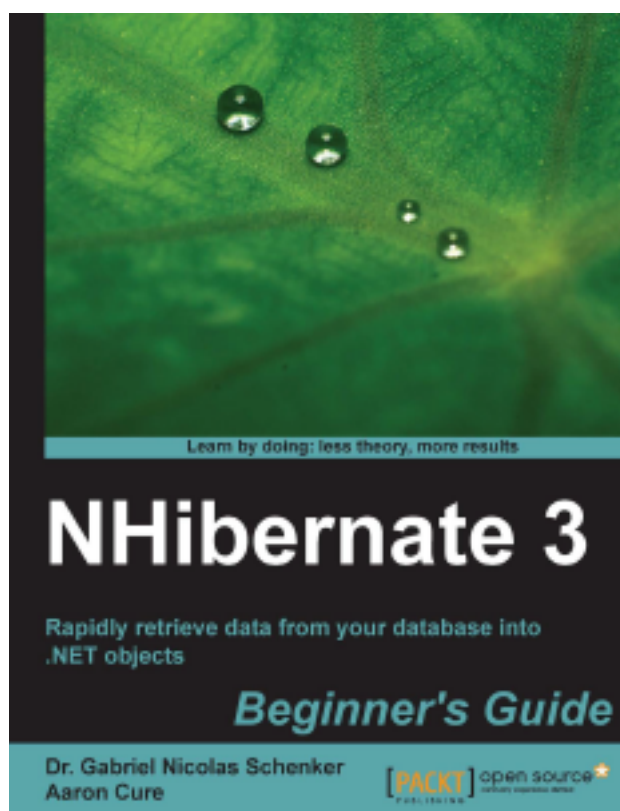
نویسنده: Hamidrezabina
تاریخ: ۲۳:۳۰:۴۶ ۱۳۹۰/۰۱/۱۶

با سلام . . .
من به آدرس گفته شده برای دانلود Nhibernate.validator رجوع کردم اما بعد از دریافت NHCH-3.1.0.GA-bin.zip هیچ فایل VS یا Nhibernate.validator.dll در اون وجود نداشت ؟؟؟!!

نویسنده: وحید نصیری
تاریخ: ۰۰:۱۶:۵۳ ۱۳۹۰/۰۱/۱۷

اون‌هایی که bin در اسمشون دارند به معنای نگارش بایناری یا فقط فایل کامپایل شده نهایی هستند و اون‌هایی که src داخل اسم فایل zip آن‌ها است، شامل سورس هستند. مثلاً
[/http://sourceforge.net/projects/nhcontrib/files/NHibernate.Validator/1.3.0%20GA](http://sourceforge.net/projects/nhcontrib/files/NHibernate.Validator/1.3.0%20GA)

در بیشتر مواردی که یک تکنولوژی جدید را برای یادگیری انتخاب می‌کنیم در اولین فرصت سراغ منابع آنلاین از قبیل کتابها و یا ویدئوهای موجود بر روی نت می‌رویم و در این بین ممکن است با محدودیت هایی از قبیل کیفیت بد اتصال به اینترنت و یا حجم مربوط به فایل‌های موجود مواجه شویم. خوب چاره و نکته در اینجا است که با انتخاب یک کتاب مفید در این زمینه می‌توان تا حدود زیادی این محدودیت‌ها را برطرف کرد. در ادامه برای شروع کار با NHibernate که روز به روز در حال توسعه است، میتوان کتاب زیر را شروع بسیار خوبی برای کار دانست:



NHibernate 3 Beginners Guide, Aug 2011

در این کتاب بصورتی بسیار جامع از ابتدایی‌ترین مسئله تا فنی‌ترین مسائلی که در هر پروژه‌ای عملی هر توسعه دهنده ای با هر سطحی امکان مواجه شدن با این مشکلات را دارد به تفصیل بررسی شده. این کتاب شامل 12 فصل بوده که مطالب آن به شرح زیر ارائه شده است:

1- فصل اول - نگاه اولیه:

NHibernate چیست

موارد تازه در آخرین نسخه NHibernate

چرا ما استفاده کنیم و چه کسانی دیگری استفاده میکنند

زمانیکه به مشکل برخوردیم از کجا کمک بگیریم یا حتی نسخه ای تجاری تهیه کنیم

2- فصل دوم - اولین مثال کامل:

آماده سازی سیستم برای توسعه برنامه‌ها با استفاده از NHibernate

- ایجاد یک مدل ساده از مشکل موجود
- ایجاد بانک و برپایی یک نگاشت (Map) بین مدل و بانک
- نوشتن و خواندن داده از و به بانک
- بحث در مورد بدست آوردن نتیجه معادل بدون استفاده از NHibernate و یا ORM دیگر
- 3- فصل سوم - ایجاد یک مدل:
- مدل چیست؟
- عوامل اصلی موجود در ایجاد یک مدل چیست؟
- چطور میتوان مدل ساخت؟
- 4- فصل چهارم - ایجاد شمای بانک:
- یادگیری جدول چیست؟
- یادگیری چطور جدولها به هم مرتبط شود؟
- بحث در مورد استراتژی‌های تحمیلی ای که کدام داده میتواند ذخیره شود
- نمایش امکانات موجود برای بهبود کارایی دسترسی به داده
- ایجاد بانک داده برای سیستم سفارش (Ordering System)
- 5- فصل پنجم - نگاشت مدل به بانک داده:
- بدست آوردن یک درک درست درباره نگاشت و پیش نیازهای آن
- بحث در مورد ریزه کاری‌های چهار تکنیک پر استفاده معمول نگاشت
- توصیف و توسعه قراردادهای کاهش تقلا در کدنویسی
- ایجاد خودکار اسکریپت برای ایجاد شمای بانک دیتا از روی نگاشت مان
- توصیف و نگاشت مدل دامنه سیستم سفارش مان
- 6- فصل ششم - وهله‌ها و تراکنش‌ها
- بحث در مورد اشیاء وهله و تراکنش
- معرفی شیء session factory
- پیاده سازی برنامه ای که دیتا ذخیره و بازخوانی میکند
- تجزیه و تحلیل متدهای گوناگون برای مدیریت وهله‌ها در پر استفاده‌ترین انواع برنامه
- 7- فصل هفتم - آزمایش کردن، نمایه سازی، نظارت، واقع نگاری
- پیاده سازی یک بستر پایه برای ساخت آزمایش ساده کد دسترسی به بانک داده
- ایجاد آزمایش‌ها برای تایید کد دسترسی به بانک داده مان
- تجزیه و تحلیل ارتباط بین NHibernate و بانک داده
- پیکربندی NHibernate برای واقع نگاری داده‌های مورد توجه
- 8- فصل هشتم - پیکر بندی
- بحث در مورد پیکربندی قبل از شروع
- آشنا شدن با لیست مولفه‌های NHibernate که میتوان پیکربندی کرد
- یادگیری چهار روش متفاوت پیکربندی که چگونه میتوان در برنامه هایمان استفاده کرد
- 9- فصل نهم - نوشتن پرس و جو
- یادگیری چگونگی استفاده از LINQ (Language Integrated Query) در NHibernate برای دریافت داده
- پرس و جو با استفاده از criteria query API
- استفاده از گویش object-oriented اصلی SQL بنام Hibernate Query Language HQL
- بحث در مورد موجودیت هایی با خواصی که توان lazy load دارند
- مقابله با بارگذاری حریصانه با lazy loading بطوریکه بصورت دسته ای از پرس و جو بنظر آید
- 10- فصل دهم - اعتبار سنجی داده برای نگهداری (ذخیره)
- 11- فصل یازدهم - اشتباهات متداول - چیزهایی برای جلوگیری

نظرات خوانندگان

نویسنده: علی قمشلویی
تاریخ: ۱۰:۳۱۳۹۱/۰۳/۲۹

سلام با تشکر از معرفی کتاب لطف کنید لینک دانلود هم بذارید

نویسنده: امین
تاریخ: ۱۷:۱۱۱۳۹۱/۰۳/۳۰

سلام
بابت سایت جدیدتون و همچنین قالبش بهتون تبریک میگم.
موفق باشید.

نویسنده: مسعود زیانی
تاریخ: ۱۷:۳۷۱۳۹۱/۰۳/۳۰

خیلی ممنون بابت معرفی کتاب
اگه امکانش هست لینک دانلود رو هم قرار بزارید....با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۷:۵۹۱۳۹۱/۰۳/۳۰

لطفا در گوگل جستجو کنید.

نویسنده: امیرحسین جلوداری
تاریخ: ۱۶:۱۳۱۳۹۱/۰۵/۰۹

آیا این کتاب رو برا کسی که EF رو بلده ، پیشنهاد میکنید؟!

نویسنده: مهدی پایروند
تاریخ: ۲۳:۱۸۱۳۹۱/۰۵/۰۹

آشنایی با EF رو اگر با CodeFirst همراه کنید میتونید جواب کارتون رو بدید، ولی کلا برای یاد گیری پیش نیازی نداره. پیشنهاد میکنم برای شروع [سری آموزشی](#) این وبلاگ رو مطالعه کنید.

امروز : یکشنبه مورخ 28 آذر 1389 ساعت : 05:19
نام کاربر : کاربر مهمان (مهمان 0) [\[خروج\]](#)

شناسه : رمز : ورود

مرکز آزمون دانشگاه



روابط عمومی



اخبار



صفحه اصلی

تکمیل ظرفیت آزمون سراسری 88

تاریخ اعلام نتیجه 29 مهرماه سال 1388



داوطلب گرامی : به دلایل فنی ، حتی الامکان از اقلامی استفاده نمایید که فاقد حروف «ی» یا «ک» باشند .
وارد کردن حداقل یکی از اقلام برای جستجو ، لازم است .

نام :

تصویر فوق، یکی از تصویرهایی است که شاید از طریق ایمیل‌هایی تحت عنوان "فقط در ایران!" به دست شما هم رسیده باشد. تصور کاربر نهایی (که این ایمیل را با تعجب ارسال کرده) این است که در اینجا به او گفته شده مثلا "مرتضی" را جستجو نکنید و امثال آن. چون برای او تفاوتی بین ی و ی وجود ندارد. همچنین بکار بردن "اقلامی" هم کمی غلط انداز است و بیشتر ذهن را به سمت کلمه سوق می‌دهد تا حرف.

در ادامه‌ی بحث آلرژي مزمن به وجود انواع "ی" و "ک" در بانک اطلاعاتی (+ و + و +)، اینبار قصد داریم این اطلاعات را به NHibernate بسط دهیم. شاید یک روش اعمال یک دست سازی "ی" و "ک" این باشد که در کل برنامه هر جایی که قرار است update یا insert ایی صورت گیرد، خواص رشته‌ای را یافته و تغییر دهیم. این روش "کار می‌کنه" ولی ایده آل نیست؛ چون حجم کار تکراری در برنامه زیاد خواهد شد و نگهداری آن هم مشکل می‌شود. همچنین امکان فراموش کردن اعمال آن هم وجود دارد. در NHibernate یک سری [EventListener وجود دارند](#) که کارشان گوش فرا دادن به یک سری رخدادها مانند مثلا update یا insert است. این رخدادها می‌توانند پیش یا پس از هرگونه ثبت یا ویرایشی در برنامه صادر شوند. بنابراین بهترین جایی که جهت اعمال این نوع ممیزی (Auditing) بدون بالا بردن حجم برنامه یا اضافه کردن بیش از حد یک سری کد تکراری در حین کار با NHibernate می‌توان یافت، روال‌های مدیریت کننده‌ی همین [EventListener](#) ها هستند.

کلاس YeKeAuditorEventListener نهایی با پیاده سازی IPreInsertEventListener و IPreUpdateEventListener به شکل زیر خواهد بود:

```
using NHibernate.Event;

namespace NHYeKeAuditor
{
    public class YeKeAuditorEventListener : IPreInsertEventListener, IPreUpdateEventListener
    {
        // Represents a pre-insert event, which occurs just prior to performing the
        // insert of an entity into the database.
        public bool OnPreInsert(PreInsertEvent preInsertEvent)
        {
            var entity = preInsertEvent.Entity;
            CorrectYeKe.ApplyCorrectYeKe(entity);
            return false;
        }
    }
}
```

```

    }

    // Represents a pre-update event, which occurs just prior to performing the
    // update of an entity in the database.
    public bool OnPreUpdate(PreUpdateEvent preUpdateEvent)
    {
        var entity = preUpdateEvent.Entity;
        CorrectYeKe.ApplyCorrectYeKe(entity);
        return false;
    }
}
}

```

در کدهای فوق روال‌های OnPreInsert و OnPreUpdate پیش از ثبت و ویرایش اطلاعات فراخوانی می‌شوند (همواره و بدون نیاز به نگرانی از فراموش شدن فراخوانی کدهای مربوطه). اینجا است که فرصت داریم تا تغییرات مورد نظر خود را جهت یکسان سازی "ی" و "ک" دریافتی اعمال کنیم (کد کلاس CorrectYeKe را در پیوست خواهید یافت).

تا اینجا فقط تعریف YeKeAuditorEventListener انجام شده است. اما NHibernate چگونه از وجود آن مطلع خواهد شد؟ برای تزریق کلاس YeKeAuditorEventListener به تنظیمات برنامه باید به شکل زیر عمل کرد:

```

using System;
using System.Linq;
using FluentNHibernate.Cfg;
using NHibernate.Cfg;

namespace NHYeKeAuditor
{
    public static class MappingsConfiguration
    {
        public static FluentConfiguration InjectYeKeAuditorEventListener(this FluentConfiguration fc)
        {
            return fc.ExposeConfiguration(configListeners());
        }

        private static Action<Configuration> configListeners()
        {
            return
                c =>
                {
                    var listener = new YeKeAuditorEventListener();
                    c.EventListeners.PreInsertEventListeners =
                        c.EventListeners.PreInsertEventListeners
                            .Concat(new[] { listener })
                            .ToArray();
                    c.EventListeners.PreUpdateEventListeners =
                        c.EventListeners.PreUpdateEventListeners
                            .Concat(new[] { listener })
                            .ToArray();
                };
        }
    }
}

```

به این معنا که FluentConfiguration خود را [همانند قبل](#) ایجاد کنید. درست در زمان پایان کار تنها کافی است متد InjectYeKeAuditorEventListener فوق بر روی آن اعمال گردد و بس (یعنی پیش از فراخوانی BuildSessionFactory).

کدهای NHYeKeAuditor را [از اینجا](#) می‌توانید دریافت کنید.

نظرات خوانندگان

نویسنده: Ramin

تاریخ: ۱۳۸۹/۰۹/۲۸ ۱۹:۳۲:۴۰

دست شما درد نکنه.

اگه امکان داره در مورد NH3 که به تازگی ریلیز شده مطلب بنویسید

نویسنده: A

تاریخ: ۱۳۸۹/۰۹/۲۸ ۱۹:۴۳:۴۰

هیچ وقت دنبال این قضیه رو نگرفتم که خود SQL این قضیه رو می‌تونه حل کنه یا نه ولی احتمالاً خود MS باید یک فکری برای این قضیه کرده باشه. شاید با Collation ها!

شما در این مورد اطلاع دارید؟

نویسنده: وحید نصیری

تاریخ: ۱۳۸۹/۰۹/۲۸ ۱۹:۵۲:۵۸

نه. از Collation ها هم کاری ساخته نیست. در این مورد اینجا بیشتر بحث شده : [\(+\)](#)

ضمناً این رو هم در نظر داشته باشید که بانک اطلاعاتی‌های گوناگونی داریم (فقط SQL Server را نباید مد نظر داشت) و ... هنگام تصمیم‌گیری باید یک راه حل کلی که همانا یکسان سازی دریافتی است را اعمال کرد.

نویسنده: A

تاریخ: ۱۳۸۹/۰۹/۲۸ ۲۲:۰۵:۵۵

بله مسلماً من در اینجا به SQL به عنوان نمونه اشاره کردم کما اینکه اکثر DBهای مدرن از چیزی شبیه این پشتیبانی می‌کنند.

نویسنده: وحید نصیری

تاریخ: ۱۳۸۹/۰۹/۲۸ ۲۲:۴۸:۳۲

بله. Collation نحوه‌ی ذخیره سازی و همچنین مرتب سازی و مقایسه‌ی اطلاعات رشته‌ای را مشخص می‌کند (کاری با تاریخ ندارد).

اما چندتا بحث هست. Collation های فعلی یا Collation های آتی.

مدل توسعه‌ی SQL Server باز نیست. به این معنا که مثلاً تا سال 2008 طول کشید تا Persian Collation به آن اضافه شد (Persian_100_CS_AS, آن هم فقط با هدف گیری قسمت مرتب سازی صحیح رشته‌ها) یا Collation فعلی SQLite در مرتب سازی یک سری حروف فارسی مشکل دارد (بر اساس کدهای اسکی حروف عمل می‌کند، چیزی که در SQL Server حل شده است به لطف وجود Collation مناسب). البته مدل توسعه‌ی آن باز است ولی ... من ندیم کسی این مورد را اصلاح کند و یک patch ارائه دهد.

هنوز کسی در مورد قسمت و مفهوم مقایسه‌ای رشته‌ها در Collations کاری نکرده است که جای کار دارد (مثلاً ی و ی یکی در نظر گرفته شود).

ولی باز هم به عنوان راه حل جامع قابل قبول نیست. چون گیرم به نگارش بعدی اضافه شد، نگارش‌های قبلی چه کنند؟ سایر بانک‌های اطلاعاتی چکار کنند؟

به همین جهت یک راه حل ساده و بدون نیاز به منتظر ماندن و سازگار با تمام بانک‌های اطلاعاتی، یکسان سازی اطلاعات دریافتی است.

نویسنده: A

تاریخ: ۱۳۸۹/۰۹/۲۹ ۱۲:۳۳:۲۴

من هم کاری به تاریخ ندارم. فعلاً ما تنهای چیزی که در این زمینه نیاز داریم این است که همانطور که SQL میتواند در هنگام Query گرفتن بین "a" و "A" فرقی نگذارد، برای «ی» و «ی» نیز به همین ترتیب عمل کند.

البته صحبت شما در مورد نگارش های قبلی بانکها درست است. اما حس می‌کنم این قضیه مانند استفاده از CSS 3.0 خواهد شد. این چیزی است که نبود آن برای یک Database مدرن یک کمبود واقعیست. البته من سعی خواهم کرد در Connect.Microsoft این مورد را گزارش دهم. مثلاً یک Collation به نام "Perisan_X_CI_AI_YI_KI" دیده شود که:

YI: Yeh Insensitive

KI: Keh Insensitive

مثلاً !!!

ظاهراً میکروسافت _ حداقل بیشتر از غولهای دیگر _ به زبان ما و شبیه آن اهمیت می‌دهد.

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۹/۲۹ ۱۲:۳۷:۲۱

اتفاقاً اضافه شدن Persian_100_CS_AS هم بر اساس پیگیری کاربران ایرانی بوده؛ بنابراین این نوع نامه نگاریها واقعا تاثیر دارد و اهمیت می‌دهند.

نویسنده: Ali
تاریخ: ۱۳۸۹/۰۹/۲۹ ۱۸:۴۸:۴۶

سلام مهندس

برای Update عمل نمی‌کنه !!!

نویسنده: Payamin
تاریخ: ۱۳۸۹/۰۹/۲۹ ۱۹:۲۲:۱۵

میکروسافت مورد مشابهی در مورد الف و همزه برای عربی زبانها دارد که مشابه اون رو می‌شه برای ی/ی/ک/ک ما هم می‌شه درخواست داد.

در سیستمهای مدیریت محتوای (غالبن متن‌باز) با پشتیبانی فارسی مقل جوملا و سایر خلنواده‌هاش تبدیل خودکار حروف عربی به فارسی معمولن ساپورت می‌شه و از جمله مثلن یکی از پلاگینهای پیش‌فرض وردپرس تبدیل خودکار ی/ک به ی/ک هستش.

+ به موردی که بارها خواسته بودم ازتون سوال کنم و بحثش پیش نیومده بود اینه که شما خودتون از kbdfa عربی پورت شده از روی ویندوز 98 هنوز استفاده می‌کنید که "ک" و "ی" عربیه.

چرا از برنامه‌های زیادی که برای این کار است استفاده نمی‌کنید یا خودتون dll مذکور رو جوری تغییر نمی‌دید که ک و ی هاتون فارسی باشه؟

یادمه چند سال پیش در جایی صحبت شد و توجیه‌تون این بود که مطالب با ی عربی ایندکس شده توی گوگل بیشتره و این استاندارد رایج وب فارسی هستش. برای اون موقع شاید این مطلب درستی بود ولی الان هر کلمه‌ای که خواستید رو سرچ کنید می‌بینید نتایج جستجوی فارسی‌ش برای "ی/ک" بیشتر از "ی/ک" هستش.

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۹/۲۹ ۲۲:۴۰:۱۳

Ali@

خیلی امیدوار شدم ... :

بله. ممنون. این یک باگ در پیاده سازی بود. هنگام استفاده از این گوش فرا دهنده‌ها، تنها به روز رسانی خواص یک موجودیت کافی نیست؛ بلکه باید حالت آنرا هم به روز کرد تا NHibernate واقعا متوجه شود که چیزی تغییر کرده.

نسخه‌ی جدید را از همان آدرس قبلی ذکر شده دریافت کنید.

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۹/۲۹ ۲۲:۴۹:۱۳

Payamin@

من چون خودم این kbdfa مخصوص ویندوز 7 64 بیتی رو درست کردم (+) ، به همین جهت یک نوع علاقه است :)

نویسنده: شنگول
تاریخ: ۱۳۸۹/۱۰/۰۱ ۱۴:۳۹:۰۴

سلام مهندس

در یکی از مقاله‌های قبلیتون که راجع به همین بحث ی/ی/ک/ک بود، من کامنتی گذاشتم در مورد مشکلی که بعد از این یکسان سازی در هنگام مرتب کردن سطرهای یک جدول پیش خواهد اومد. در کامنتهای مربوط به این پستتون توضیح داده بودم:

<http://www.dotnettips.info/2010/08/wcf-ria-services.html>

بعد متأسفانه دیگه فرصت نشد و بعدشم من فراموش کردم که این بحث را پیگیری کنم. لطفاً تصویر زیر را که توی مدیافایر آپ شده ملاحظه کنید:

<http://www.mediafire.com/imageview.php?quickkey=fnyc6ilu6i6n6ur>

یه جدول ساده‌ست که توی اکسس ۲۰۱۰ ساخته شده و حروف ی/ی/ک/ک در اون وارد شده و بعد با استفاده از خود اکسس (یعنی کلیک بر ستون) مرتب شده. (برای اینکار کوئری نوشته نشده)
همونطور که در تصویر ملاحظه میکنید بعد از مرتب سازی، ک (فارسی) بعد از ی (عربی) قرار گرفته. این مشکل در اکسس ۲۰۰۳ هم بود ولی اکسس ۲۰۰۷ را در دسترس نداشتم.
نمیدونم که این مشکل بخاطر سلختار دیتابییی اکسس به وجود میاد یا اینکه مشکل مربوط به Data Grid اون میشه، ولی در برنامه‌های تجاری دیگه‌ای هم این مشکل را دیده‌ام.
اگر شما این مشکل را بررسی کردید و دلیلی براش پیدا کردید خوشحال میشم نتایج را منتشر کنید.

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۱۰/۰۱ ۱۹:۱۴:۲۸

سلام

کاری که اکسس در اینجا کرده یا SQLite ایی که مثال زدم، مرتب سازی بر اساس کدهای یونیکد این حروف است و کار صحیحی (در بدو امر حداقل) انجام شده: (+)

کد یونیکد "ک" عربی = 1603
کد یونیکد "ی" عربی = 1610
کد یونیکد "ک" فارسی = 1705
کد یونیکد "ی" فارسی = 1740

یعنی این مرتب سازی بر اساس منطق ریاضی صحیح است؛ اما بر اساس فرهنگ ایران خیر. به همین جهت توسعه دهنده‌های بانک‌های اطلاعاتی مجبور شده‌اند تا مفهومی به نام Collation را ارائه بدهند که در بالا در مورد آن بحث شد. این Collation دیگر صرفاً بر اساس منطق ریاضی کدهای یونیکد حروف، مرتب سازی را انجام نمی‌دهد، بلکه بر اساس ادبیات و فرهنگ زبان‌های مختلف کار مرتب سازی را انجام خواهد داد. SQL Server در این زمینه حداقل برای فارسی زبان‌ها یک Collation مخصوص را در نگارش 2008 خودش ارائه داده تا مرتب سازی صورت گرفته روی رشته‌ها دقیقاً مطابق فرهنگ و ادبیات ایرانی باشد (برای سایر کشورها هم این نوع Collation ها پیش بینی شده).

در اکسس که مد نظر شما است این Collation به نام General Sort order مهیا است (در اکسس‌های جدید در قسمت فایل،

options و سپس برگه‌ی general قسمتی هست به نام new database sort order که همین collation است (+) و اگر در این مورد خاص درست کار نمی‌کند باید با مایکروسافت مکاتبه کرد و این مسایل را توضیح داد.

نویسنده: برای مصطفی کرمی
تاریخ: ۱۱:۵۲:۱۷ ۱۳۸۹/۱۰/۰۲

سلام و عرض ادب
اگر کلیت مشکل فارسی و عربی را بررسی کنیم به گمانم به 3 سطح زیر میرسیم

1- تبدیل در ویندوز

2- تبدیل در UI

3- تبدیل در بانک اطلاعاتی

در تایید فرمایش شما، شخصا چون صدها کاربر با میلیونها رکورد پراکند در سطح کشور دارم که همگی از هر دو ویندوز XP و 7 می‌خواهند استفاده کنند و هزینه تبدیل اطلاعات قبلی به فارسی بسیار بالاست، ترجیح میدهم ادامه ورود اطلاعاتم روی عربی باشد.

اما در سطح بانک اطلاعاتی بدلیل کنترلهای فراوان و احتمال خطا زیاد موافق نیستم (اگر نیاز است میتوانم توضیح بیشتری بدهم)

در سطح UI هم همانگونه که فرمودید مشکلات عدیده ای وجود دارد

در سطح ویندوز بهترین راه حل است.. اما نمیدانم چگونه میتوانم انرا پیاده کنم. اگر بتوانید راهنمایی بفرمایید من در مورد پیاده سازی و یا سرمایه گذاری روی پیاده سازی مشکلی ندارم

مرسی

وکیلی

Javan_Soft@Yahoo.com

نویسنده: وحید نصیری
تاریخ: ۱۲:۵۶:۵۴ ۱۳۸۹/۱۰/۰۲

سلام، در سطح ویندوز استفاده از kbdfa.dll استاندارد و اصلاح شده کفایت می‌کند (و مزیتی هم که دارد این است که به تمام برنامه‌های موجود به صورت یک دست اعمال می‌شود؛ بدون نیاز به کد نویسی). یعنی اگر این مورد همه‌گیر شود اصلا مشکلی نخواهیم داشت.

این فایل را هم از اینجا می‌توانید دریافت کنید: (+)

نویسنده: وحید نصیری
تاریخ: ۱۲:۵۹:۲۲ ۱۳۸۹/۱۰/۰۲

اضافه کنم که فایل بالا مربوط به ی و ک فارسی است. اگر ی و ک عربی مد نظر است به این مطلب مراجعه کنید: (+)

نویسنده: برای مصطفی کرمی
تاریخ: ۱۸:۲۲:۰۸ ۱۳۸۹/۱۰/۰۲

مرسی از سرعت جواب
مشکل در این است که این فایل را چگونه در ویندوز 7 بدون مشکل امنیتی نصب کنم
میدانید که کاربر را نمیتوان در گیر SafeMode و یا پیدا کردن شاخه های ویندوز نمود

در عین حال هم قبلا برخی از DLLهای موجود در اینترنت را استفاده کرده اما جوابگو نبوده است . امیدوارم این مورد به سرچ من پایان دهد
باز هم از جوابگویی متشکرم

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۱۰/۰۲ ۱۹:۲۱:۵۵

از این نصاب استفاده کنید: (+)
سپس از قسمت regional settings کنترل پنل، صفحه کلید اضافه شده باید انتخاب شود.

نویسنده: برای مصطفی کرمی
تاریخ: ۱۳۸۹/۱۰/۰۵ ۱۱:۵۴:۴۲

از برنامه فوق جهت نصب استفاده کردم و صفحه کلید درست شد. (Persian-2901)
اما در برخی قسمتهای برنامه ام فونت ها بصورت علامت سوال نمایش داده می شود. برای درست شدن علامت سوالها در
Region-> admin-> Change System local
Current System Local را به Persian تبدیل کردم . با این کار مشکل علامت سوالها حل شد ولی دوباره ک و ی فارسی به
سیستم برگشت . پس از اینکه همین بخش را به English تبدیل کردم دوباره ک و ی عربی را داشتم .
آیا مشکل علامت سوالها در برخی موارد میتواند بدلیل برنامه نویسی (نسخه دلفی / کامپوننت مورد استفاده و یا ترکیب فونت
(باشد ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۱۰/۰۵ ۱۲:۴۹:۵۵

این مشکلات زمان VB6 (مرحوم) هم بود (مثلا هنگام انتخاب فونت برای یک متن فارسی باید script آن را در صفحه انتخاب فونت
روی Arabic گذاشت تا درست نمایش داده شود). قبل از دات نت. قبل از یونیکد شدن رشته ها در سیستم های متداول دات نت
به صورت پیش فرض از نگارش یک آن.
دلفی های جدید هم به نظر رشته یونیکد را پیش فرض خود کرده اند (نگارش های بعد از 2007). بهتر است برنامه خودتون رو به
این نگارش ها ارتقاء بدید (تا به صورت خودکار همه چیز منجمله کامپوننت ها (ی جدید) بر مبنای رشته های یونیکد کار کنند)، همچنین
بانک اطلاعاتی هم باید واقعا رشته های یونیکد را ساپورت کند. مثلا در SQL Server ، بین نوع های varchar و nvarchar تفاوت وجود
دارد.
در کل من با این صفحه کلید و برنامه های دات نت، نه مشکلی در ثبت دارم و نه مشکلی در نمایش (چند سال هست). همچنین نیم
فاصله هم جهت تایپ فارسی پشتیبانی می شود + ساپورت فونت های قدیمی هم لحاظ شده.

نویسنده: برای مصطفی کرمی
تاریخ: ۱۳۸۹/۱۰/۰۵ ۱۳:۳۱:۵۷

البته فرمایش شما صحیح است . این مشکلات قبلا هم وجود داشته است . اما با توجه به اینکه بنده از دلفی 2010 و کامپونتهای
استاندار آن استفاده میکنم و کد پیچ را هم روی عربیک گذاشته ام فکر میکنم این نظر مشکل داشته باشد .
با این وجود تست کرده و نتیجه را مجددا خدمت شما اعلام خواهم کرد
متشکرم

نویسنده: برای مصطفی کرمی
تاریخ: ۱۳۸۹/۱۰/۰۶ ۱۲:۳۸:۵۹

یک روش ساده پیدا کردم که اگر استاده هم در مورد آن نظر بدهند ممنون میشوم
پس از نصب ویندوز 7 هیچ نوع فارسی سازی روی آن نصب نکردم .
تنها در بخش

Region-> admin-> Change System local
Current System Local را به Persian تبدیل کردم

پس از ری استارت ویندوز تمامی علامت سوالها فارسی شد.
سپس در برنامه ام کد پیچ فارسی (عربی)
; (LoadKeyboardLayout('00000429', 1
را لود کردم .

براحتی کیبوردی که پ و ژ آن صحیح است ارائه شد. ی و ک هم عربی خورد . نیم فاصله هم کاملاً صحیح عمل میکند.
توضیح در مورد ویندوز مورد استفاده ام
Product Id : 00426-OEM-8992662-00497
System Type : 32-bit

در بخش Regional and Language
Format : English
Location : United states
Keyboard : English-US (فقط. هیچ فارسی اضافه نشده)
System Locale : Persian

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۱۰/۰۶ ۱۲:۴۴:۵۷

این روش هم برای برنامه‌های دسکتاپ خوبه. ولی بحث اصلی این تایپیک تقریباً به تمام برنامه‌هایی که می‌توانند از NHibernate استفاده کنند و الزاماً هم توانایی دخل و تصرف در سیستم را ندارند (مانند برنامه‌های وب)، قابل بسط و استفاده است.

ICriteria API در NHibernate پیاده سازی الگوی [Query Object](#) است. مشکلی هم که این روش دارد استفاده از رشته‌ها جهت ایجاد کوئری‌های متفاوت است؛ به عبارتی Type safe نیست. ایرادی هم به آن وارد نیست چون پیاده سازی اولیه آن از جاوا صورت گرفته و مباحث Lambda Expressions و Extension Methods هنوز در آن زبان به صورت رسمی ارائه نشده است (در JDK 7 تحت عنوان Closures [قرار است](#) اضافه شود). [NHibernate 3.0](#) از ویژگی‌های جدید زبان‌های دات نت جهت ارائه‌ی محصور کننده‌ای Type safe حول ICriteria API استاندارد به نام QueryOver سود جسته است. این پیاده سازی بسیار شبیه به عبارات LINQ است اما نباید با آن اشتباه گرفته شود زیرا LINQ to NHibernate یک ویژگی دیگر جدید، یکپارچه و استاندارد NHibernate 3.0 به شمار می‌رود.

برای نمونه در یک ICriteria query متداول، فراخوانی‌های ذیل متداول است:

```
Add(Expression.Eq("Name", "Smith"))
```

اکنون شما در NHibernate 3.0 می‌توانید دستورات فوق را به صورت ذیل وارد نمایید:

```
.Where<Person>(p => p.Name == "Smith")
```

مزیت‌های این روش (strongly-typed fluent API) به شرح زیر است:

- خبری از رشته‌ها جهت استفاده از یک خاصیت وجود ندارد. برای مثال در اینجا خاصیت Name کلاس Person تحت کنترل کامپایلر قرار می‌گیرد و اگر در کلاس Person تغییراتی حاصل شود، برای مثال Name به LName تغییر کند، برنامه دیگر کامپایل نخواهد شد. اما در حالت ICriteria API یا باید به نتایج حاصل از Unit testing مراجعه کرد یا باید به نتایج بازخورد کاربران برنامه مانند: "باز برنامه رو تغییر دادی، یکجای دیگر از کار افتاد!" دقت نمود!
- اگر در حین ویرایش کلاس Person از ابزارهای Refactoring استفاده شود، تغییرات حاصل به صورت خودکار به تمام برنامه نیز اعمال خواهد شد. بدیهی است این اعمال تغییرات تنها در صورتی میسر است که خاصیت مورد نظر به صورت رشته معرفی نگردیده و ارجاعات به اشیاء تعریف شده به سادگی قابل parse باشند.
- در این حالت امکان بررسی نوع خواص تغییر کرده نیز توسط کامپایلر به سادگی میسر است و اگر ارجاعات تعریف شده به نحو صحیحی از این نوع جدید استفاده نکنند باز هم برنامه تا رفع این مشکلات کامپایل نخواهد شد که این هم مزیت مهمی در نگهداری ساده‌تر یک برنامه است.
- با بکارگیری Extension methods و پیاده سازی Fluent API جدید، مدت زمان یادگیری این روش نیز به شدت کاهش یافته، زیرا Intellisense موجود در VS.NET بهترین راهنمای استفاده از امکانات فراهم شده است. برای مثال جهت استفاده از ویژگی جدید QueryOver به سادگی می‌توان پس از ساختن یک session جدید به صورت زیر عمل نمود:

```
IList<Cat> cats = session.QueryOver<Cat>().Where(c => c.Name == "Max").List();
```

در اینجا اگر متدهای نمایش داده شده توسط Intellisense را دنبال کنیم دیگر حتی نیازی به مراجعه به مستندات QueryOver در مورد اینکه چه متدها و امکاناتی را فراهم کرده است نیز نخواهد بود.

جهت مشاهده‌ی معرفی کامل آن می‌توان به [مستندات](#) NHibernate 3.0 مراجعه کرد.

نظرات خوانندگان

نویسنده: Meysam

تاریخ: ۱۶:۰۳:۵۲ ۱۳۸۹/۰۹/۳۰

Oren Eini یک روش شبیه به این برای INPC هم داده که جالبه

نویسنده: A

تاریخ: ۱۶:۱۰:۰۹ ۱۳۸۹/۰۹/۳۰

گذشته از بحث NHibernate، اینطوری که پیداست واقعاً جاوا Follower سی شارپ شده!!!

نویسنده: ghafoori

تاریخ: ۱۸:۵۶:۳۴ ۱۳۸۹/۱۰/۰۱

سلام آقای نصیری

من از کد زیر برای جستجو در بانکم بوسیله nhibernate ویرایش 3 استفاده می کنم اما خطا میدهد

(Dim Query = ServerRepo.Find(Function(x) x.Country Is Country And x.ServerName Is ServerName

خطای زیر

Unable to cast object of type 'NHibernate.Hql.Ast.HqlBitwiseAnd' to type

."NHibernate.Hql.Ast.HqlBooleanExpression

وقتی بجای Is از = استفاده می کنم خطای زیر را می دهد

(Int32 CompareString(System.String, System.String, Boolean

فکر می کنید مشکل از کجاست با تشکر

نویسنده: وحید نصیری

تاریخ: ۱۹:۵۸:۳۴ ۱۳۸۹/۱۰/۰۱

مشکل مرتبط است با زبان VB.NET، جهت توضیحات بیشتر و ارائه راه حل (که باید کمی کدهای اصلی NHibernate را ویرایش (جایگزینی VBStringComparisonExpression با BinaryExpression) و سپس کامپایل کنید) این دو مقاله را مطالعه کنید:

[\(+\)](#) و [\(+\)](#)

نویسنده: ghafoori

تاریخ: ۱۲:۴۲:۳۱ ۱۳۸۹/۱۰/۰۲

اقای نصیری بابت راهنمایی خیلی متشکر

من ترسیدم برم سراغ کدهای nhibernate و کلا پروژه ام را از وی بی به سی شارپ تغییر دادم مشکل هم حل شد

مخصوصاً با این تبدیل کننده شرکت تلریک

<http://converter.telerik.com/batch.aspx>

سریع تونستم هسته پروژه که با nhibernate سرو کار داره را از وی بی به سی شارپ انتقال بدم

باز هم بابت راهنمایی ممنونم

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۱۱/۰۸ ۱۵:۰۸:۱۳

یک مورد شبیه به این QueryOver را هم اینجا می‌توانید پیدا کنید:
[nhflowquery blog](#) , [nhflowquery](#)

اولین موردی که پس از دریافت [NHibernate 3.0](#) ممکن است به چشم بخورد، نبود اسمبلی Log4Net است. مطابق [درخواست‌های کاربران](#)، ارجاع مستقیم به این کتابخانه حذف شده و با یک اینترفیس عمومی به نام `IInternalLogger` جایگزین گشته است (قرار گرفته در فضای نام `NHibernate.Logging`). به این صورت می‌توان از انواع و اقسام کتابخانه‌های ثبت وقایع نوشته شده برای دات نت استفاده کرد؛ مانند: `log4net`، `Nlog`، `EntLib Logging` و غیره. البته لازم به ذکر است که همان [روش قبلی](#) استفاده از `Log4Net` هنوز هم پشتیبانی می‌شود (بدون نیاز به تغییر خاصی در کدهای خود)، زیرا پیاده سازی اینترفیس جدید `IInternalLogger` برای استفاده از آن به صورت پیش فرض توسط `NHibernate` انجام شده است.

یک مثال:

کتابخانه‌ی سورس باز [Common.Logging](#) واقع شده در سورس فورج، در واقع یک `logging abstraction framework` است. به این معنا که تا به حال کتابخانه‌های ثبت وقایع مختلف و متعددی برای دات نت فریم ورک نوشته شده است و هر کدام راه و روش و پیاده سازی خاص خود را دارند. کتابخانه‌ی `Common.Logging` لایه‌ای است عمومی بر روی تمام این کتابخانه‌ها مانند `Log4Net`، `Nlog`، `Enterprise Library Logging` و غیره که برنامه‌ی شما را از وابستگی مستقیم به هر کدام از موارد ذکر شده رها می‌سازد. اکنون با توجه به وجود اینترفیس `IInternalLogger` در `NHibernate 3.0`، تنها کافی است این اینترفیس جهت استفاده از کتابخانه‌ی `Common.Logging` پیاده سازی شود (`abstraction` اندر `abstraction`!). البته نیازی نیست اینکار انجام شود، زیرا پیشتر توسط پروژه‌ی [NHibernate.Logging](#) در سایت کدپلکس اینکار صورت گرفته است. بنابراین تنها کاری که باید انجام داد این است که:

الف) ارجاعاتی را به اسمبلی‌های `Common.Logging.dll` (واقع در سورس فورج) و `NHibernate.Logging.CommonLogging.dll` (واقع در کدپلکس) به پروژه‌ی خود اضافه کنید.

ب) ارجاعی را نیز به اسمبلی کتابخانه‌ی ثبت وقایع مورد نظر خود نیز باید اضافه نمایید (مثلاً [NLog](#)).

ج) سپس باید چند سطر زیر را به فایل `app.config` خود اضافه کنید:

```
<appSettings>
  <add key="nhibernate-logger"
        value="NHibernate.Logging.CommonLogging.CommonLoggingLoggerFactory,
        Hibernate.Logging.CommonLogging"/>
</appSettings>
```

`NHibernate.Logging.CommonLogging.dll` وقایع داخلی `NHibernate` را با پیاده سازی `IInternalLogger` به `Common.Logging.dll` منتقل می‌کند. سپس `Common.Logging.dll` این وقایع را به زبان کتابخانه‌ی ثبت وقایع مورد نظر ترجمه خواهد کرد.

اطلاعات بیشتر: ([+](#))

احتمالا مطلب " [دات نت 4 و کلاس Lazy](#) " را پیشتر مطالعه کرده‌اید. هر چند NHibernate 3.0 بر اساس دات نت فریم ورک 3 و نیم تهیه شده، اما شبیه به این مفهوم را در مورد بارگذاری به تاخیر افتاده‌ی مقادیر خواص یک کلاس که به ندرت مورد استفاده قرار می‌گیرند، پیاده سازی کرده است. Lazy را در اینجا تنبل، به تعویق افتاده، با تاخیر و شبیه به آن می‌توان ترجمه کرد؛ خواص معوقه!

برای مثال فرض کنید یکی از خواص کلاس مورد استفاده، متن، تصویر یا فایلی حجیم است. در مکانی هم که قرار است وهله‌ای از این کلاس مورد استفاده قرار گیرد نیازی به این اطلاعات حجیم نیست؛ با سایر خواص آن سر و کار داریم و نیازی به اشغال حافظه و منابع سیستم در این مورد خاص نیست.

سؤال: چگونه آن‌را تعریف کنیم؟

اگر از NHibernate سنتی استفاده می‌کنید (یا به عبارتی فایل‌های hbm.xml را دستی تهیه می‌کنید)، ویژگی Lazy را به صورت زیر می‌توان مشخص کرد:

```
<property name="Text" lazy="true"/>
```

اگر از Fluent NHibernate استفاده می‌کنید (و فایل‌های hbm.xml به صورت خودکار از کلاس‌های شما تهیه خواهند شد)، روش کار به صورت زیر است (فراخوانی متد LazyLoad روی خاصیت مورد نظر):

```
public class Post
{
    public virtual int Id { set; get; }
    public virtual string PostText { set; get; }
}
```

```
public class PostMappings : ClassMap<Post>
{
    public PostMappings()
    {
        Id(p => p.Id, "PostId").GeneratedBy.Identity();
        Map(p => p.PostText).LazyLoad();
        //...
        Table("Posts");
    }
}
```

در این حالت در پشت صحنه در مورد خاصیت PostText چنین نگاهی تعریف خواهد شد:

```
<property name="PostText" type="System.String, mscorlib, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" lazy="true" />
```

سؤال: چه زمانی نباید از این روش استفاده کرد؟

فرض کنید در شرایطی دیگر نیاز است تا اطلاعات تمام رکوردهای جدول مذکور به همراه مقدار PostText نمایش داده شود. در این حالت بسته به تعداد رکوردها، ممکن است هزاران هزار کوئری به دیتابیس ایجاد شود که مطلوب نیست (به ازای هربار دسترسی به خاصیت PostText یک کوئری تولید می‌شود).

البته امکان لغو موقت این روش تنها در حین استفاده از HQL (یکی دیگر از روش‌های دسترسی به داده‌ها در NHibernate) میسر است. اطلاعات بیشتر: [\(+\)](#)

نظرات خوانندگان

نویسنده: مهدی پایروند
تاریخ: ۱۳۸۹/۱۰/۰۵ ۰۹:۲۳:۳۴

بتازگی انتشارات Packtpub یک کتاب با موضوع NHibernate 3.0 چاپ کرده:

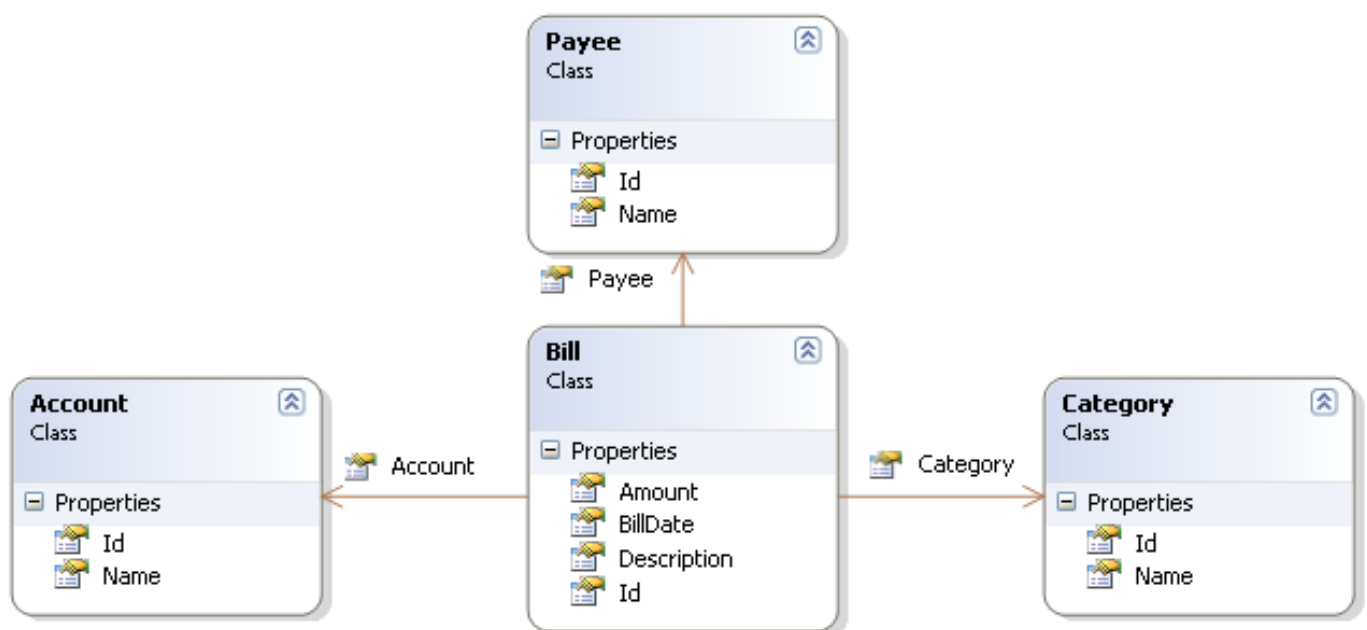
<https://www.packtpub.com/nhibernate-3-0-cookbook/book>

ORM های Entity framework و NHibernate روش‌های متفاوتی را برای به روز رسانی کلید خارجی با حداقل رفت و برگشت به دیتابیس ارائه می‌دهند که در ادامه معرفی خواهند شد.

صورت مساله:

فرض کنید می‌خواهیم برنامه‌ای را بنویسیم که ریز پرداخت‌های روزانه‌ی ما را ثبت کند. برای اینکار حداقل به یک جدول گروه‌های اقلام خریداری شده، یک جدول حساب‌های تامین کننده‌ی مخارج، یک جدول فروشنده‌ها و نهایتاً یک جدول صورتحساب‌های پرداختی بر اساس جداول ذکر شده نیاز خواهد بود.

الف) بررسی مدل برنامه



در اینجا جهت تعریف ویژگی‌ها یا Attributes تعریف شده در این کلاس‌ها از NHibernate validator استفاده شده (+). مزیت اینکار هم علاوه بر اعتبارسنجی سمت کلاینت (پیش از تبادل اطلاعات با بانک اطلاعاتی)، تولید جداولی با همین مشخصات است. برای مثال Fluent NHibernate بر اساس ویژگی Length تعریف شده با طول حداکثر 120، یک فیلد nvarchar با همین طول را ایجاد می‌کند.

```

public class Account
{
    public virtual int Id { get; set; }

    [NotNullNotEmpty]
    [Length(Min = 3, Max = 120, Message = "طول نام باید بین 3 و 120 کاراکتر باشد")]
    public virtual string Name { get; set; }
}
  
```



```

public class Category
{
    public virtual int Id { get; set; }

    [NotNullNotEmpty]
    [Length(Min = 3, Max = 130, Message = "طول نام باید بین 3 و 130 کاراکتر باشد")]
    public virtual string Name { get; set; }
}

public class Payee
{
    public virtual int Id { get; set; }

    [NotNullNotEmpty]
    [Length(Min = 3, Max = 120, Message = "طول نام باید بین 3 و 120 کاراکتر باشد")]
    public virtual string Name { get; set; }
}

public class Bill
{
    public virtual int Id { get; set; }

    [NotNull]
    public virtual Account Account { get; set; }

    [NotNull]
    public virtual Category Category { get; set; }

    [NotNull]
    public virtual Payee Payee { get; set; }

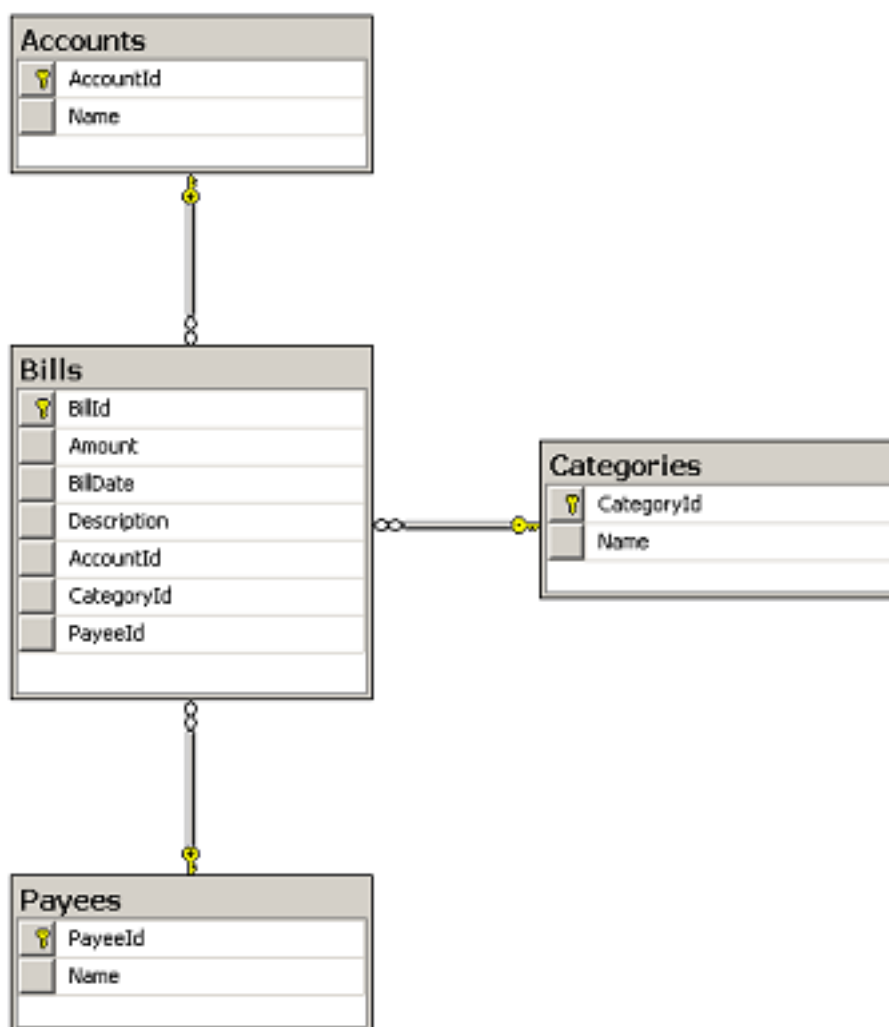
    [NotNull]
    public virtual decimal Amount { set; get; }

    [NotNull]
    public virtual DateTime BillDate { set; get; }

    [NotNullNotEmpty]
    [Length(Min = 1, Max = 500, Message = "طول توضیحات باید بین 1 و 500 کاراکتر باشد")]
    public virtual string Description { get; set; }
}

```

ب) ساختار جداول متناظر (تولید شده به صورت خودکار توسط Fluent NHibernate در اینجا)



در مورد نحوه‌ی استفاده از ویژگی AutoMapping و همچنین تولید خودکار ساختار بانک اطلاعاتی از روی جداول در NHibernate [قبلا](#) توضیح داده شده است. البته بدیهی است که ترکیب مقاله‌ی Validation و آشنایی با AutoMapping در اینجا جهت اعمال ویژگی‌ها باید بکار گرفته شود که در [همان](#) مقاله‌ی Validation مفصل توضیح داده شده است. نکته‌ی مهم database schema تولیدی، کلیدهای خارجی (foreign key) تعریف شده بر روی جدول Bills است (همان AccountId, CategoryId و PayeeId تعریف شده) که به primary key جداول متناظر اشاره می‌کند.

```

create table Accounts (
    AccountId INT IDENTITY NOT NULL,
    Name NVARCHAR(120) not null,
    primary key (AccountId)
)

create table Bills (
    BillId INT IDENTITY NOT NULL,
    Amount DECIMAL(19,5) not null,
    BillDate DATETIME not null,
    Description NVARCHAR(500) not null,
    AccountId INT not null,
    CategoryId INT not null,
    PayeeId INT not null,
    primary key (BillId)
)

create table Categories (
    CategoryId INT IDENTITY NOT NULL,
    Name NVARCHAR(130) not null,
    primary key (CategoryId)
)
  
```

```

create table Payees (
    PayeeId INT IDENTITY NOT NULL,
    Name NVARCHAR(120) not null,
    primary key (PayeeId)
)

alter table Bills
    add constraint fk_Account_Bill
    foreign key (AccountId)
    references Accounts

alter table Bills
    add constraint fk_Category_Bill
    foreign key (CategoryId)
    references Categories

alter table Bills
    add constraint fk_Payee_Bill
    foreign key (PayeeId)
    references Payees

```

ج) صفحه‌ی ثبت صورتحساب‌ها

صفحات ثبت گروه‌های اقلام، حساب‌ها و فروشنده‌ها، نکته‌ی خاصی ندارند. چون این جداول وابستگی خاصی به جایی نداشته و به سادگی اطلاعات آن‌ها را می‌توان ثبت یا به روز کرد.

صفحه‌ی مشکل در این مثال، همان صفحه‌ی ثبت صورتحساب‌ها است که از سه کلید خارجی به سه جدول دیگر تشکیل شده است.

عموماً برای طراحی این نوع صفحات، کلیدهای خارجی را با drop down list نمایش می‌دهند و اگر در جهت سهولت کار کاربر قدم برداشته شود، باید از یک Auto complete drop down list استفاده کرد تا کاربر برنامه جهت یافتن آیتم‌های از پیش تعریف شده کمتر سختی بکشد.

اگر از Silverlight یا WPF استفاده شود، امکان بایند یک لیست کامل از اشیاء با تمام خواص مرتبط به آن‌ها وجود دارد (هر رکورد نمایش داده شده در دراپ داون لیست، دقیقا معادل است با یک شیء متناظر با کلاس‌های تعریف شده است). اگر از ASP.NET استفاده شود (یعنی یک محیط بدون حالت که پس از نمایش یک صفحه دیگر خبری از لیست اشیاء بایند شده وجود نخواهد داشت و همگی توسط وب سرور جهت صرفه جویی در منابع تخریب شده‌اند)، بهتر است datatextfield را با فیلد نام و datavaluefield را با فیلد Id مقدار دهی کرد تا کاربر نهایی، نام را جهت ثبت اطلاعات مشاهده کند و برنامه از Id موجود در لیست جهت ثبت کلیدهای خارجی استفاده نماید.

و نکته‌ی اصلی هم همینجا است که چگونه؟! چون ما زمانیکه با یک ORM سر و کار داریم، برای ثبت یک رکورد در جدول Bills باید یک وهله از کلاس Bill را ایجاد کرده و خواص آن‌را مقدار دهی کنیم. اگر به تعریف کلاس Bill مراجعه کنید، سه خاصیت آن از نوع سه کلاس مجزا تعریف شده است. به عبارت دیگر با داشتن فقط یک id از رکوردهای این کلاس‌ها باید بتوان سه وهله‌ی متناظر آن‌ها را از بانک اطلاعاتی خواند و سپس به این خواص انتساب داد:

```
var newBill = new Bill
{
    Account = accountRepository.GetByKey(1),
    Amount = 1,
    BillDate = DateTime.Now,
    Category = categoryRepository.GetByKey(1),
    Description = "testtest...",
    Payee = payeeRepository.GetByKey(1)
};
```

یعنی برای ثبت یک رکورد در جدول Bills فوق، چهار بار رفت و برگشت به دیتابیس خواهیم داشت:

- یکبار برای دریافت رکورد متناظر با گروه‌ها بر اساس کلید اصلی آن (که از دراپ داون لیست مربوطه دریافت می‌شود)
- یکبار برای دریافت رکورد متناظر با فروشنده‌ها بر اساس کلید اصلی آن (که از دراپ داون لیست مربوطه دریافت می‌شود)
- یکبار برای دریافت رکورد متناظر با حساب‌ها بر اساس کلید اصلی آن (که از دراپ داون لیست مربوطه دریافت می‌شود)
- یکبار هم ثبت نهایی اطلاعات در بانک اطلاعاتی

متد GetByKey فوق همان متد session.Get استاندارد NHibernate است (چون به primary key ها از طریق drop down list دسترسی داریم، به سادگی می‌توان بر اساس متد Get استاندارد ذکر شده عمل کرد).

SQL نهایی تولیدی هم به صورت واضحی این مشکل را نمایش می‌دهد (4 بار رفت و برگشت؛ سه بار select یکبار هم insert نهایی):

```
SELECT account0_.AccountId as AccountId0_0_, account0_.Name as Name0_0_
FROM Accounts account0_ WHERE account0_.AccountId=@p0;@p0 = 1 [Type: Int32 (0)]

SELECT category0_.CategoryId as CategoryId2_0_, category0_.Name as Name2_0_
FROM Categories category0_ WHERE category0_.CategoryId=@p0;@p0 = 1 [Type: Int32 (0)]

SELECT payee0_.PayeeId as PayeeId3_0_, payee0_.Name as Name3_0_
FROM Payees payee0_ WHERE payee0_.PayeeId=@p0;@p0 = 1 [Type: Int32 (0)]

INSERT INTO Bills (Amount, BillDate, Description, AccountId, CategoryId, PayeeId)
VALUES (@p0, @p1, @p2, @p3, @p4, @p5);
select SCOPE_IDENTITY();
@p0 = 1 [Type: Decimal (0)],
@p1 = 2010/12/27 11:48:33 ق.ظ [Type: DateTime (0)],
@p2 = 'testtest...' [Type: String (500)],
@p3 = 1 [Type: Int32 (0)],
@p4 = 1 [Type: Int32 (0)],
@p5 = 1 [Type: Int32 (0)]
```

کسانی که قبلا با رویه‌های ذخیره شده کار کرده باشند (stored procedures) احتمالا الان خواهند گفت؛ ما که گفتیم این روش کند است! سربار زیادی دارد! فقط کافی است یک SP بنویسید و کل عملیات را با یک رفت و برگشت انجام دهید. اما در ORMs نیز برای انجام این مورد در طی یک حرکت یک ضرب راه حل‌هایی وجود دارد که در ادامه بحث خواهد شد:

د) پیاده سازی با NHibernate

برای حل این مشکل در NHibernate با داشتن primary key (برای مثال از طریق datavaluefield ذکر شده)، بجای session.Get از session.Load استفاده کنید.

session.Get یعنی همین الان برو به بانک اطلاعاتی مراجعه کن و رکورد متناظر با کلید اصلی ذکر شده را بازگشت بده و یک شیء از آن را ایجاد کن (حالت‌های دیگر دسترسی به اطلاعات مانند استفاده از LINQ یا Criteria API یا هر روش مشابه دیگری نیز در اینجا به همین معنا خواهد بود).

session.Load یعنی فعلا دست نگه دار! مگر در جدول نهایی نگاشت شده، اصلا چیزی به نام شیء مثلا گروه وجود دارد؟ مگر این مورد واقعا یک فیلد عددی در جدول Bills بیشتر نیست؟ ما هم که الان این عدد را داریم (به کمک عناصر دراپ داون لیست)، پس لطفا در پشت صحنه یک پروکسی برای ایجاد شیء مورد نظر ایجاد کن (uninitialized proxy to the entity) و سپس عملیات مرتبط را در حین تشکیل SQL نهایی بر اساس این عدد موجود انجام بده. یعنی نیازی به رفت و برگشت به بانک اطلاعاتی نیست. در این حالت اگر SQL نهایی را بررسی کنیم فقط یک سطر زیر خواهد بود (سه select ذکر شده حذف خواهند شد):

```
INSERT INTO Bills (Amount, BillDate, Description, AccountId, CategoryId, PayeeId)
VALUES (@p0, @p1, @p2, @p3, @p4, @p5);
select SCOPE_IDENTITY();
@p0 = 1 [Type: Decimal (0)],
@p1 = 2010/12/27 11:58:22 ق.ظ [Type: DateTime (0)],
@p2 = 'testtest...' [Type: String (500)],
@p3 = 1 [Type: Int32 (0)],
@p4 = 1 [Type: Int32 (0)],
@p5 = 1 [Type: Int32 (0)]
```

ه) پیاده سازی با Entity framework

Entity framework زمانیکه بانک اطلاعاتی فوق را (به روش database first) به کلاس‌های متناظر تبدیل/نگاشت می‌کند، حاصل نهایی مثلا در مورد کلاس Bill به صورت خلاصه به شکل زیر خواهد بود:

```
public partial class Bill : EntityObject
{
    public global::System.Int32 BillId {set;get;}
    public global::System.Decimal Amount {set;get;}
    public global::System.DateTime BillDate {set;get;}
    public global::System.String Description {set;get;}
    public global::System.Int32 AccountId {set;get;}
    public global::System.Int32 CategoryId {set;get;}
    public global::System.Int32 PayeeId {set;get;}
    public Account Account {set;get;}
    public Category Category {set;get;}
}
```

به عبارتی فیلدهای کلیدهای خارجی، در تعریف نهایی این کلاس هم مشاهده می‌شوند. در اینجا فقط کافی است سه کلید خارجی، از نوع int مقدار دهی شوند (و نیازی به مقدار دهی سه شیء متناظر نیست). در این حالت نیز برای ثبت اطلاعات، فقط یکبار رفت و برگشت به بانک اطلاعاتی خواهیم داشت.

نظرات خوانندگان

نویسنده: Afshar Mohebbi
تاریخ: ۱۳۸۹/۱۰/۰۶ ۲۳:۲۰:۲۴

سلام،

من هم این روزها خیلی درگیر این مسئله با NH هستم. تا حالا دو تا راه پیدا کردم. یکی استفاده از HQL برای update کردن و دیگری استفاده از خاصیت Future برای کاهش رفت و آمدها به دیتابیس. البته تا حالا از هیچ کدام اونها به طور عملی استفاده نکردم. ولی با این راه حلی که شما گفتید تعداد راه حل ها سه تا می شود.

فقط مسئله کوچکی که می ماند این است که من عمدتاً از Castle ActiveRecord و Linq-to-NH استفاده می کنم و نمی دانم با نبود Load (به جای Get) در این حالت چه کار کنم.

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۱۰/۰۷ ۰۰:۳۴:۴۰

سلام، به نظر مطابق مستندات آن [\(+\)](#) اگر SessionScope تعریف شود و داخل آن کار کنید، متد Find شبیه به همان Load ذکر شده در مطلب فوق عمل می کند. تست کنید ببینید در این حالت تعداد کوئری ها چه فرقی می کند.

نویسنده: iMAN
تاریخ: ۱۳۸۹/۱۰/۰۸ ۱۱:۴۸:۳۹

برای Castle ActiveRecord استفاده از SessionScope همونطور که اشاره کردین موضوع را حل می کند.

همانطور که در مطلب " [NHibernate 3.0 و عدم وابستگی مستقیم به Log4Net](#) " عنوان شد، از اینترفیس جدید IInternalLogger آن می‌توان جهت ثبت وقایع داخلی NHibernate استفاده کرد. اگر در این بین صرفاً بخواهیم SQL های تولیدی را لاگ کنیم، خلاصه‌ی آن به صورت زیر خواهد بود:

```
public class LoggerFactory : ILoggerFactory
{
    public IInternalLogger LoggerFor(System.Type type)
    {
        if (type == typeof(NHibernate.Tool.hbm2ddl.SchemaExport))
            //log it
    }

    public IInternalLogger LoggerFor(string keyName)
    {
        if (keyName == "NHibernate.SQL")
            //log it
    }
}
```

یا کلید NHibernate.SQL باید پردازش شود (جهت ثبت SQL های کوئری‌ها) یا نوع NHibernate.Tool.hbm2ddl.SchemaExport جهت ثبت SQL ساخت ساختار جداول بانک اطلاعاتی باید بررسی گردد. [سورس](#) کامل این کتابخانه‌ی کوچک را [از اینجا](#) می‌توانید دریافت کنید. جهت استفاده از آن تنها کافی است چند سطر زیر به فایل app.config یا web.config برنامه‌ی شما اضافه شوند:

```
<appSettings>
    <add key="nhibernate-logger" value="NH3SQLLogger.LoggerFactory, NH3SQLLogger" />
</appSettings>
```

کلید nhibernate-logger ، به صورت مستقیم توسط NHibernate بررسی می‌شود و صرف نظر از اینکه از کدامیک از مشتقات NHibernate استفاده می‌کنید، با تمام آن‌ها کار خواهد کرد. لازم به ذکر است که اگر برنامه‌ی شما از نوع ASP.NET است، این کتابخانه اطلاعات را در پوشه‌ی استاندارد App_Data ثبت خواهد کرد؛ در غیراینصورت فایل‌ها در کنار فایل اجرایی برنامه تشکیل خواهند شد.

نظرات خوانندگان

نویسنده: A

تاریخ: ۲۳:۳۳:۴۵ ۱۳۸۹/۱۰/۰۸

شاید این موضوع بی ربط به پست جاری باشه. از این بابت عذر خواهی می‌کنم.

اینطور که در کدهای شما دیدم شما اغلب از `CodeStyle` یی شبیه `Developer` های سیستم‌های میکروسافت استفاده میکنید. که بارزترین تفاوت آن استفاده از `Underline` در ابتدای فیلدهای تعریف شده در کلاس است. حتماً می‌دانید که `StyleCop` نیز که توسط خود میکروسافت ارائه شده، این روش را رد می‌کند و می‌گوید `Underline` زیبایی را کاهش می‌دهد و به جای آن از `this`. استفاده شود که محاسن دیگری نیز دارد.

البته به قول نویسنده `StyleCop` اینکه بچه‌های میکروسافت اینطور کد می‌نویسند یک دلیل منطقی دارد؛ و آن این است که آنها بیشتر برنامه‌نویسان `C++` بوده‌اند و در `C++` شبیه این سبک بیشتر رایج است (درست است که این روش، روش مجارستانی نیست ولی حداقل استفاده از `this` به هیچ عنوان در `C++` رایج نبود).

خوب، سوال اینجاست. نظر شما چیست؟

نویسنده:

وحید نصیری

تاریخ: ۲۳:۵۰:۴۸ ۱۳۸۹/۱۰/۰۸

خوانایی استفاده از `Underline` برای معرفی `private field names` از حالت استفاده از `this` بیشتر است (فقط با یک نگاه مشخص می‌شود). همچنین امکان فراموش شدن استفاده از `this` هم ممکن است باشد که این مورد سبب بروز احتمال تداخل متغیرهای یک متد با نمونه‌ای که به صورت `private field` تعریف شده می‌گردد. فقط موردی را که باید در نظر داشت، یکپارچگی است. یعنی در کل کدهای شما یک روش باید وجود داشته باشد. همچنین چون من برای بازبینی کدها یکبار از `resharper` هم استفاده می‌کنم، توصیه `resharper` استفاده از `Underline` هست و `this` را به صورت زاید (`redundant`) معرفی می‌کند.

نویسنده: A

تاریخ: ۰۲:۳۴:۳۷ ۱۳۸۹/۱۰/۰۹

البته یک چیزی خیلی برایم جالب است که ایشان به `Underscore` معتقد است:

<http://10rem.net/articles/net-naming-conventions-and-programming-standards---best-practices>

ولی در MSDN ذکر شده که

"Do not apply a prefix to field names or static field names"

منبع: پاراگراف آخر [http://msdn.microsoft.com/en-us/library/ta31s3bc\(v=vs.71\).aspx](http://msdn.microsoft.com/en-us/library/ta31s3bc(v=vs.71).aspx)

اگر مطلب "[ذخیره سازی SQL تولیدی در NH3](#)" را دنبال کرده باشید که یک مثال عملی از "[NHibernate 3.0 و عدم وابستگی مستقیم به Log4Net](#)" بود، خروجی حاصل از آن به صورت زیر است:

```
---+ 12/29/2010 05:35:59.75 +---
SQL ...

---+ 12/29/2010 05:35:59.75 +---
SQL ...
```

و پس از مدتی این فایل هیچ حسی را منتقل نمی‌کند! یک سری SQL که لاگ شده‌اند. مشخص نیست کدام متد در کدام کلاس و کدام فضای نام، سبب صدور این عبارت SQL ثبت شده، گردیده است. خوشبختانه در دات نت فریم ورک می‌توان با بررسی Stack trace، رد کاملی را از فراخوان‌های متدها یافت:

```
StackTrace stackTrace = new StackTrace();
StackFrame stackFrame = stackTrace.GetFrame(1);
MethodBase methodBase = stackFrame.GetMethod();
Type callingType=methodBase.DeclaringType;
```

با بررسی StackFrame ها امکان یافتن نام متدها، فضاهای نام و غیره میسر است. مثلاً یکی از کاربردهای مهم این روش، ثبت فراخوان‌های متدی است که استثنایی را ثبت کرده است. بر این اساس سورس مثال قبل را جهت درج اطلاعات فراخوان‌های متدها تکمیل کرده‌ام، که [از این آدرس](#) قابل دریافت است.

اکنون اگر از این ماژول جدید استفاده کنیم، خروجی نمونه‌ی آن به صورت زیر خواهد بود:

```
---+ 01/02/2011 02:19:24.98 +---
-- Void ASP.feedback_aspx.ProcessRequest(System.Web.HttpContext) [File=App_Web_4nvdip40.5.cs, Line=0]
-- Void Prog.Web.UserCtrls.FeedbacksList.Page_Load(System.Object, System.EventArgs)
[File=FeedbacksList.ascx.cs, Line=23]
---- Void Prog.Web.UserCtrls.FeedbacksList.BindTo() [File=FeedbacksList.ascx.cs, Line=43]
----- System.Collections.Generic.IList`1[Feedback] Prog.GetAllUserFeedbacks(Int32)
[File=FeedbackWebRepository.cs, Line=66]

SELECT ...
@p0 = 3 [Type: Int32 (0)]
```

به این معنا که عبارت SQL ثبت شده، حاصل از پردازش صفحه‌ی feedback.aspx، سپس متد Page_Load آن که از یوزر کنترل FeedbacksList.ascx استفاده می‌کند، می‌باشد. در اینجا فراخوانی متد BindTo سبب فراخوانی متد GetAllUserFeedbacks در فایل FeedbackWebRepository.cs واقع در سطر 66 آن گردیده است. اینطوری حداقل می‌توان دریافت که SQL تولیدی دقیقاً به کجا بر می‌گردد و چه متدی سبب صدور آن شده است.

ملاحظات:

این ماژول تنها در صورت وجود فایل pdb معتبر کنار اسمبلی‌های شما این خروجی مفصل را تولید خواهد کرد. در غیر اینصورت از آن صرف‌نظر می‌کند. (برای مثال نام فایل سورس فراخوان، شماره‌ی سطر فراخوان، حتی محل قرارگیری آن فایل بر روی کامپیوتر شما در فایل‌های pdb ثبت می‌گردند؛ به همین جهت توصیه اکید حین ارائه‌ی نهایی برنامه، حذف این نوع فایل‌ها است)

نظرات خوانندگان

نویسنده: Meysam

تاریخ: ۱۳:۲۱:۲۴ ۱۳۸۹/۱۰/۱۳

استفاده از StackTrace برای پیاده سازی INPC به نظرتون مشکل Performance ایجاد میکنه؟

<http://csharperimage.jeremylikness.com/2010/12/jounce-part-8-raising-property-changed.html>

نویسنده: وحید نصیری

تاریخ: ۱۶:۵۱:۰۲ ۱۳۸۹/۱۰/۱۳

بله، در تعداد رکورد بالا مثلاً در یک گرید در صفحه، حتماً مشکل‌زا است. کلاً روش در این مورد زیاد هست، منجمله روشی که در قسمت 5 آموزش MVVM در سایت جاری هست. یا روشی که شما ذکر کردید، یا یک روش دیگر هم استفاده از فریم ورک‌های AOP است. این‌ها روی کد IL نهایی تاثیر می‌گذارند.

ولی در نهایت همان روش سنتی استفاده از رشته‌ها، هر چند کمی طولانی‌تر است، اما بهترین کارایی و کمترین سربار را هم دارد.

مطلبی هم که من در اینجا عنوان کردم در مورد دیباگ یک سیستم مبتنی بر NHibernate هست و بدیهی است قرار نیست در محیط کاری از آن استفاده شود.

نویسنده: وحید نصیری

تاریخ: ۱۸:۰۴:۰۷ ۱۳۸۹/۱۰/۳۰

نگارش کامل شده‌ی این پروژه را از آدرس زیر دریافت کنید:

[/http://nh3sqllogger.codeplex.com](http://nh3sqllogger.codeplex.com)

عنوان: آیا دیتابیس مورد استفاده در NHibernate با نگاشت‌های تعریف شده همخوانی دارد؟

نویسنده: وحید نصیری

تاریخ: ۱۳۸۹/۱۰/۱۵ ۲۳:۱۹:۰۰

آدرس: www.dotnettips.info

برچسب‌ها: NHibernate

زمانیکه خاصیتی به یکی از کلاس‌های نگاشت‌های تعریف شده اضافه می‌شود یا حذف می‌گردد، دقیقاً باید این به روز رسانی در سمت بانک اطلاعاتی هم انجام شود. امکان تهیه و همچنین اعمال اسکریپت نهایی تولید database schema مهیا است، اما ممکن است به هر علتی این کار فراموش شود. اکنون سؤال این است که آیا می‌توان سریع بررسی کرد که دیتابیس مورد استفاده با نگاشت‌های برنامه همخوانی و تطابق دارد؟

جهت پاسخ به این سؤال بهترین راه ایجاد یک کوئری Select بر اساس تمام خواص تعریف شده در یک کلاس است. اگر یکی از خواص یا حتی خود جدول وجود نداشته باشد، انجام این کوئری خودبخود با شکست مواجه شده و یک استثناء صادر خواهد شد. همین ایده را به سادگی می‌توان با NHibernate هم پیاده سازی کرد:

```
public class ConfirmDatabaseMatchesMappings
{
    public static void ValidateDatabaseSchemaAgainstMappings()
    {
        //در اینجا باید سشن فکتوری سراسری تعریف شده را دریافت و استفاده کرد
        using (var session = sessionManager.OpenSession())
        {
            var allClassMetadata = session.SessionFactory.GetAllClassMetadata();

            foreach (var entry in allClassMetadata)
            {
                session.CreateCriteria(entry.Value.GetMappedClass(EntityMode.Poco))
                    .SetMaxResults(0).List();
            }
        }
    }
}
```

برای مثال اگر فیلدی در کلاس‌های برنامه موجود باشد اما در بانک اطلاعاتی خیر، استثنای حاصل شبیه به عبارات ذیل خواهد بود:

```
NHibernate.Exceptions.GenericADOException was unhandled
Message=could not execute query
...
```

و اگر کمی سایر اطلاعات این استثناء را بررسی کنیم، به همان عبارات آشنای فلان فیلد یافت نشد یا فلان جدول وجود ندارد، می‌رسیم.

نظرات خوانندگان

نویسنده: ghafoori
تاریخ: ۱۳۸۹/۱۰/۱۶ ۰۹:۵۲:۱۲

سلام آقای نصیری
قبلا داخل وبلاگ شما یا دیگران دقیق یادم نیست با کلاس
SchemaValidator آشنا شدم این کلاس برای چک کردن وجود دیتابیس برای nhibernate بود
میخواستم ببینم از همین کلاس می تونه جایگزین همین کد شما در این پست باشه یا فقط برای چک کردن وجود بانک بکار میره و
آیا این کد الان شما فقط برای چک تغییرات است یا برای چک کردن خود بانک هم می تونه بکار بره
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۱۰/۱۶ ۱۰:۳۴:۰۸

بله، این کد برای چک کردن خود بانک است. آیا فیلدی فراموش نشده. آیا جدولی از قلم نیفتاده.
موردی هم که اشاره کردید جزو ابزارهای NHibernate است و شبیه به همین کار را انجام می‌دهد: [\(+\)](#)

نویسنده: Meysam
تاریخ: ۱۳۸۹/۱۰/۱۶ ۲۲:۱۸:۳۶

همچین چیزی واسه EF سراغ دارین؟

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۱۰/۱۶ ۲۳:۰۰:۴۵

به صورت توکار و آماده، خیر اما یک ابزار غیر رایگان برای اینکار هست: [\(+\)](#)
و همچنین یک مورد هم اینجا ذکر شده: [\(+\)](#)

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۱/۲۱ ۱۲:۳۳:۳۷

این هم نسخه‌ی مربوط به EF
[Unit Test to verify Entity Framework Model \(EDMX\) is accurate](#)

مباحث eager fetching/loading (واکشی حریصانه) و lazy loading/fetching (واکشی در صورت نیاز، با تاخیر، تنبل) جزو نکات کلیدی کار با ORM های پیشرفته بوده و در صورت عدم اطلاع از آن‌ها و یا استفاده‌ی ناصحیح از هر کدام، باید منتظر از کار افتادن زود هنگام سیستم در زیر بار چند کاربر همزمان بود. به همین جهت تصور اینکه "با استفاده از ORMs دیگر از فراگیری SQL راحت شدیم!" یا اینکه "به من چه که پشت صحنه چه اتفاقی می‌افته!" بسی مهلك و نادرست است! در ادامه به تفصیل به این موضوع پرداخته خواهد شد.

ابزار مورد نیاز

در این مطلب از برنامه‌ی [NHProf](#) استفاده خواهد شد. اگر مطالب NHibernate این سایت را دنبال کرده باشید، در مورد لاگ کردن SQL تولیدی به اندازه‌ی کافی توضیح داده شده یا حتی یک مازول جمع و جور هم برای مصارف دم دستی [نوشته شده است](#). این موارد شاید این ایده را به همراه داشته باشند که چقدر خوب می‌شد یک برنامه‌ی جامع‌تر برای این نوع بررسی‌ها تهیه می‌شد. حداقل SQL نهایی فرمت می‌شد (یعنی برنامه باید مجهز به یک SQL Parser تمام عیار باشد که کار چند ماهی هست ...؛ با توجه به اینکه مثلاً NHibernate از افزونه‌های SQL ویژه بانک‌های اطلاعاتی مختلف هم پشتیبانی می‌کند، مثلاً T-SQL مایکروسافت با یک سری ریزه کاری‌های منحصر به MySQL متفاوت است)، یا پس از فرمت شدن، syntax highlighting به آن اضافه می‌شد، در ادامه مشخص می‌کرد کدام کوئری‌ها سنگین‌تر هستند، کدامیک نشانه‌ی عدم استفاده‌ی صحیح از ORM مورد استفاده است، چه مشکلی دارد و از این موارد. خوشبختانه این ایده‌ها یا آرزوها با برنامه‌ی NHProf محقق شده است. این برنامه برای استفاده‌ی یک ماه اول آن رایگان است (آدرس ایمیل خود را وارد کنید تا یک فایل مجوز رایگان یک ماهه برای شما ارسال گردد) و پس از یک ماه، باید حداقل 300 دلار هزینه کنید.

واکشی حریصانه و غیرحریصانه چیست؟

رفتار یک ORM جهت تعیین اینکه آیا نیاز است برای دریافت اطلاعات بین جداول Join صورت گیرد یا خیر، واکشی حریصانه و غیرحریصانه را مشخص می‌سازد. در حالت واکشی حریصانه به ORM خواهیم گفت که لطفاً جهت دریافت اطلاعات فیلدهای جداول مختلف، از همان ابتدای کار در پشت صحنه، Join های لازم را تدارک بین. در حالت واکشی غیرحریصانه به ORM خواهیم گفت به هیچ عنوان حق نداری Join ایی را تشکیل دهی. هر زمانی که نیاز به اطلاعات فیلدی از جدولی دیگر بود باید به صورت مستقیم به آن مراجعه کرده و آن مقدار را دریافت کنی. به صورت خلاصه برنامه نویسی در حین کار با ORM های پیشرفته نیازی نیست Join بنویسد. تنها باید ORM را طوری تنظیم کند که آیا اینکار را حتماً خودش در پشت صحنه انجام دهد (واکشی حریصانه)، یا اینکه خیر، به هیچ عنوان SQL های تولیدی در پشت صحنه نباید حاوی Join باشند (lazy loading).

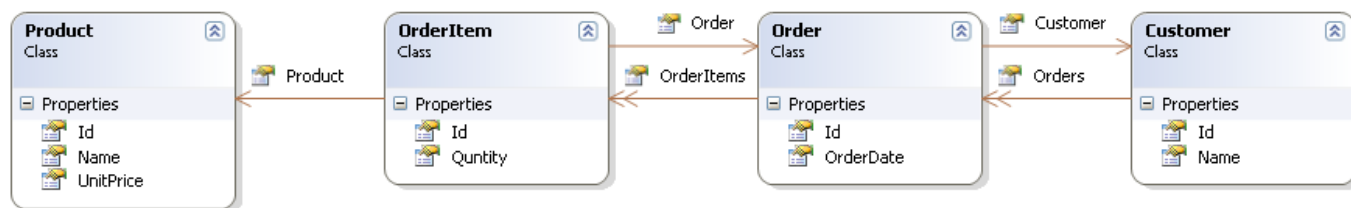
چگونه واکشی حریصانه و غیرحریصانه را در NHibernate 3.0 تنظیم کنیم؟

در NHibernate اگر تنظیم خاصی را تدارک ندیده و خواص جداول خود را به صورت virtual معرفی کرده باشید، تنظیم پیش فرض دریافت اطلاعات همان lazy loading است. به مثالی در این زمینه توجه بفرمائید:

مدل برنامه:

مدل برنامه همان مثال کلاسیک مشتری و سفارشات او می‌باشد. هر مشتری چندین سفارش می‌تواند داشته باشد. هر سفارش به

یک مشتری وابسته است. هر سفارش نیز از چندین قلم جنس تشکیل شده است. در این خرید، هر جنس نیز به یک سفارش وابسته است.



```
using System.Collections.Generic;
namespace CustomerOrdersSample.Domain
{
    public class Customer
    {
        public virtual int Id { get; set; }
        public virtual string Name { get; set; }
        public virtual IList<Order> Orders { get; set; }
    }
}
```

```
using System;
using System.Collections.Generic;
namespace CustomerOrdersSample.Domain
{
    public class Order
    {
        public virtual int Id { get; set; }
        public virtual DateTime OrderDate { set; get; }
        public virtual Customer Customer { get; set; }
        public virtual IList<OrderItem> OrderItems { set; get; }
    }
}
```

```
namespace CustomerOrdersSample.Domain
{
    public class OrderItem
    {
        public virtual int Id { get; set; }
        public virtual Product Product { get; set; }
        public virtual int Quantity { get; set; }
        public virtual Order Order { set; get; }
    }
}
```

```
namespace CustomerOrdersSample.Domain
{
    public class Product
    {
        public virtual int Id { set; get; }
        public virtual string Name { get; set; }
        public virtual decimal UnitPrice { get; set; }
    }
}
```

که جداول متناظر با آن به صورت زیر خواهند بود:

```

create table Customers (
    CustomerId INT IDENTITY NOT NULL,
    Name NVARCHAR(255) null,
    primary key (CustomerId)
)

create table Orders (
    OrderId INT IDENTITY NOT NULL,
    OrderDate DATETIME null,
    CustomerId INT null,
    primary key (OrderId)
)

create table OrderItems (
    OrderItemId INT IDENTITY NOT NULL,
    Quantity INT null,
    ProductId INT null,
    OrderId INT null,
    primary key (OrderItemId)
)

create table Products (
    ProductId INT IDENTITY NOT NULL,
    Name NVARCHAR(255) null,
    UnitPrice NUMERIC(19,5) null,
    primary key (ProductId)
)

alter table Orders
    add constraint fk_Customer_Order
    foreign key (CustomerId)
    references Customers

alter table OrderItems
    add constraint fk_Product_OrderItem
    foreign key (ProductId)
    references Products

alter table OrderItems
    add constraint fk_Order_OrderItem
    foreign key (OrderId)
    references Orders

```

همچنین یک سری اطلاعات آزمایشی زیر را هم در نظر بگیرید: (بانک اطلاعاتی انتخاب شده SQL CE است)

```

SET IDENTITY_INSERT [Customers] ON;
GO
INSERT INTO [Customers] ([CustomerId],[Name]) VALUES (1,N'Customer1');
GO
SET IDENTITY_INSERT [Customers] OFF;
GO
SET IDENTITY_INSERT [Products] ON;
GO
INSERT INTO [Products] ([ProductId],[Name],[UnitPrice]) VALUES (1,N'Product1',1000.00000);
GO
INSERT INTO [Products] ([ProductId],[Name],[UnitPrice]) VALUES (2,N'Product2',2000.00000);
GO
INSERT INTO [Products] ([ProductId],[Name],[UnitPrice]) VALUES (3,N'Product3',3000.00000);
GO
SET IDENTITY_INSERT [Products] OFF;
GO
SET IDENTITY_INSERT [Orders] ON;
GO
INSERT INTO [Orders] ([OrderId],[OrderDate],[CustomerId]) VALUES (1,{ts '2011-01-07 11:25:20.000'},1);
GO
SET IDENTITY_INSERT [Orders] OFF;
GO
SET IDENTITY_INSERT [OrderItems] ON;
GO
INSERT INTO [OrderItems] ([OrderItemId],[Quantity],[ProductId],[OrderId]) VALUES (1,10,1,1);
GO
INSERT INTO [OrderItems] ([OrderItemId],[Quantity],[ProductId],[OrderId]) VALUES (2,5,2,1);
GO
INSERT INTO [OrderItems] ([OrderItemId],[Quantity],[ProductId],[OrderId]) VALUES (3,20,3,1);
GO
SET IDENTITY_INSERT [OrderItems] OFF;

```

GO

دریافت اطلاعات :

می‌خواهیم نام کلیه محصولات خریداری شده توسط مشتری‌ها را به همراه نام مشتری و زمان خرید مربوطه، نمایش دهیم (دریافت اطلاعات از 4 جدول بدون join نویسی):

```
var list = session.QueryOver<Customer>().List();
foreach (var customer in list)
{
    foreach (var order in customer.Orders)
    {
        foreach (var orderItem in order.OrderItems)
        {
            Console.WriteLine("{0}:{1}:{2}", customer.Name, order.OrderDate,
orderItem.Product.Name);
        }
    }
}
```

خروجی به صورت زیر خواهد بود:

```
Customer1:2011/01/07 11:25:20 :Product1
Customer1:2011/01/07 11:25:20 :Product2
Customer1:2011/01/07 11:25:20 :Product3
```

اما بهتر است نگاهی هم به پشت صحنه عملیات داشته باشیم:

The screenshot shows the NHibernateProfiler interface. On the left, there's a sidebar with 'Sessions' and 'Analysis' tabs. Under 'Sessions', 'Session #2' is selected. The main area shows 'Statements' for 'Session #2'. It lists several SQL queries with their row counts and durations. The queries are:

- begin transaction with isolation level: ReadCommitted
- SELECT ... FROM Customers this_ (1 row, 1 ms / 240 ms)
- SELECT ... FROM Orders orders0_ WHERE orders0_CustomerId = 1 (1 row, 84 ms / 201 ms)
- SELECT ... FROM OrderItems orderitems0_ WHERE orderitems0_OrderId = 1 (3 rows, 8 ms / 12 ms)
- SELECT ... FROM Products product0_ WHERE product0_ProductId = 1 (1 row, 32 ms / 43 ms)
- SELECT ... FROM Products product0_ WHERE product0_ProductId = 2 (1 row, 1 ms / 1 ms)
- SELECT ... FROM Products product0_ WHERE product0_ProductId = 3 (1 row, 0 ms / 2 ms)
- commit transaction

Below the statements, there's a 'Details' tab showing the SQL query for the last statement (SELECT ... FROM Products product0_ WHERE product0_ProductId = 3). The query is:

```
1 SELECT product0_.ProductId as ProductId3_0_,
2 product0_.Name as Name3_0_,
3 product0_.UnitPrice as UnitPrice3_0_
4 FROM Products product0_
5 WHERE product0_.ProductId = 3 /* @p0 */
```

On the right, there's a 'Param Value' table showing the parameter values for the query:

Param	Value
@p0	3

همانطور که مشاهده می‌کنید در اینجا اطلاعات از 4 جدول مختلف دریافت می‌شوند اما ما Join ایی را ننوشته‌ایم. ORM هر جایی که به اطلاعات فیلدهای جداول دیگر نیاز داشته، به صورت مستقیم به آن جدول مراجعه کرده و یک کوئری، حاصل این عملیات خواهد بود (مطابق تصویر جمعا 6 کوئری در پشت صحنه برای نمایش سه سطر خروجی فوق اجرا شده است).

این حالت فقط و فقط با تعداد رکورد کم بهینه است (و به همین دلیل هم تدارک دیده شده است). بنابراین اگر برای مثال قصد نمایش اطلاعات حاصل از 4 جدول فوق را در یک گرید داشته باشیم، بسته به تعداد رکوردها و تعداد کاربران همزمان برنامه (خصوصاً در برنامه‌های تحت وب)، بانک اطلاعاتی باید بتواند هزاران هزار کوئری رسیده حاصل از lazy loading را پردازش کند و این یعنی مصرف بیش از حد منابع (IO بالا، مصرف حافظه بالا) به همراه بالا رفتن CPU usage و از کار افتادن زود هنگام سیستم. کسانی که پیش از این با SQL نویسی خو گرفته‌اند احتمالاً الان منابع موجود را در مورد نحوه‌ی نوشتن Join در NHibernate زیر و رو خواهند کرد؛ زیرا پیش از این آموخته‌اند که برای دریافت اطلاعات از دو یا چند جدول مرتبط باید Join نوشت. اما همانطور که پیشتر نیز عنوان شد، اگر با جزئیات کار با NHibernate آشنا شویم، نیازی به Join نویسی نخواهیم داشت. اینکار را خود ORM در پشت صحنه باید و می‌تواند مدیریت کند. اما چگونه؟

در NHibernate 3.0 با معرفی QueryOver که جایگزینی از نوع strongly typed همان ICriteria API قدیمی است، یا با معرفی Query که همان LINQ to NHibernate می‌باشد، متدی به نام Fetch نیز تدارک دیده شده است که استراتژی‌های lazy loading و eager loading را به سادگی توسط آن می‌توان مشخص نمود.

مثال: دریافت اطلاعات با استفاده از QueryOver

```
var list = session
    .QueryOver<Customer>()
    .Fetch(c => c.Orders).Eager
    .Fetch(c => c.Orders.First().OrderItems).Eager
    .Fetch(c => c.Orders.First().OrderItems.First().Product).Eager
    .List();

foreach (var customer in list)
{
    foreach (var order in customer.Orders)
    {
        foreach (var orderItem in order.OrderItems)
        {
            Console.WriteLine("{0}:{1}:{2}", customer.Name, order.OrderDate,
            orderItem.Product.Name);
        }
    }
}
```

پشت صحنه:

NHibernateProfiler FILE OPTIONS REPORTS HELP

Recording Filter Inactive

Sessions Analysis

Recent Statements

☆ Session #2 [1]

Session #2

Statements Entities Session Usage

Short SQL

Short SQL	Row Count	Duration	Alerts
begin transaction with isolation level: ReadCommitted			
SELECT ... FROM Customers this_ left outer join Orders orders2_...		4 ms	

Details Stack Trace

```

1 SELECT this_.CustomerId as CustomerId0_3_,
2        this_.Name as Name0_3_,
3        orders2_.CustomerId as CustomerId5_,
4        orders2_.OrderId as OrderId5_,
5        orders2_.OrderId as OrderId1_0_,
6        orders2_.OrderDate as OrderDate1_0_,
7        orders2_.CustomerId as CustomerId1_0_,
8        orderitems3_.OrderId as OrderId6_,
9        orderitems3_.OrderItemId as OrderItem1_6_,
10       orderitems3_.OrderItemId as OrderItem1_2_1_,
11       orderitems3_.Quantity as Quantity2_1_,
12       orderitems3_.ProductId as ProductId2_1_,
13       orderitems3_.OrderId as OrderId2_1_,
14       product4_.ProductId as ProductId3_2_,
15       product4_.Name as Name3_2_,
16       product4_.UnitPrice as UnitPrice3_2_
17 FROM
18 Customers this_
19 left outer join Orders orders2_
20 on this_.CustomerId = orders2_.CustomerId
21 left outer join OrderItems orderitems3_
22 on orders2_.OrderId = orderitems3_.OrderId
23 left outer join Products product4_
24 on orderitems3_.ProductId = product4_.ProductId

```

Session factory Statistics

unnamed

Close Statement Count	0
Collection Fetch Count	0
Collection Load Count	0

اینبار فقط یک کوئری حاصل عملیات بوده و join ها به صورت خودکار با توجه به متدهای Fetch ذکر شده که حالت eager loading آن‌ها صریحا مشخص شده است، تشکیل شده‌اند (6 بار رفت و برگشت به بانک اطلاعاتی به یکبار تقلیل یافت).

نکته 1: نتایج تکراری

اگر حاصل join آخر را نمایش دهیم، نتایجی تکراری خواهیم داشت که مربوط است به مقدار دهی customer با سه وهله از شیء مربوطه تا بتواند واکنشی حریصانه‌ی مجموعه اشیاء فرزند آن‌را نیز پوشش دهد. برای رفع این مشکل یک سطر TransformUsing باید اضافه شود:

```

...
.TransformUsing(NHibernate.Transform.Transformers.DistinctRootEntity)
.List();

```

دریافت اطلاعات با استفاده از LINQ to NHibernate3.0

برای اینکه بتوان متدهای Fetch ذکر شده را به LINQ to NHibernate 3.0 اعمال نمود، ذکر فضای نام NHibernate.Linq ضروری است. پس از آن خواهیم داشت:

```
var list = session
```

```
.Query
```

اینبار از FetchMany، سپس ThenFetchMany (برای واکنشی حریصانه مجموعه‌های فرزند) و در آخر از ThenFetch استفاده خواهد شد.

همانطور که ملاحظه می‌کنید حاصل این کوئری، با کوئری قبلی ذکر شده یکسان است. هر دو، اطلاعات مورد نیاز از دو جدول مختلف را نمایش می‌دهند. اما یکی در پشت صحنه شامل چندین و چند کوئری برای دریافت اطلاعات است، اما دیگری تنها از یک کوئری Join دار تشکیل شده است.

نکته 2: خطاهای ممکن

ممکن است حین تعریف متدهای Fetch در زمان اجرا به خطاهای `Antlr.Runtime.MismatchedTreeNodeException` و یا `Specified method is not supported` و یا موارد مشابهی برخورد نمائید. تنها کاری که باید انجام داد جابجا کردن مکان بکارگیری extension methods است. برای مثال متد Fetch باید پس از Where در حالت استفاده از LINQ ذکر شود و نه قبل از آن.

نظرات خوانندگان

نویسنده: مهدی پایروند
تاریخ: ۱۳۸۹/۱۰/۱۹ ۲۳:۱۳:۲۹

سلام، مطلب جالبی بود، البته تا اونجایی که من میدونم برای مثال در بانک اطلاعاتی اوراکل اگر از جویین بیشتر از 3 تا استفاده کنید سرعت دریافت اطلاعات به شدت پایین میاد

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۱۰/۲۰ ۰۰:۲۷:۳۵

برای دریافت مجوز یک ماهه برنامه‌ی NHProf به همان صفحه <http://nhprof.com/Trial> مراجعه کرده و ایمیل خود را وارد کنید.

در زمان اولین بارگذاری NHibernate ، ساخت تمام نگاشته‌ها صورت گرفته و همچنین session factory ایجاد می‌گردد. به همین جهت به کمک الگوی [thread safe singleton](#) نسبت به کش کردن آن در طول عمر یک برنامه استفاده می‌گردد. در برنامه‌ای که در یک محیط کاری مورد استفاده قرار می‌گیرد این زمان اصلا مهم نیست، زیرا تنها یکبار باید انجام شود. اما به عنوان یک برنامه نویسی شاید در طول روز صدها بار نیاز به باز و بسته کردن برنامه جهت آزمودن آن داشته باشیم و این مورد پس از مدتی تبدیل به عذاب می‌شود! خوشبختانه امکان serialize نمودن تنظیمات تولیدی session factory به فایل و سپس خواندن از آن نیز وجود دارد که این امر در حین توسعه‌ی برنامه بسیار ارزشمند است. جهت مطالعه بیشتر می‌توان به مطالب زیر مراجعه کرد:

[Speed up nHibernate startup with object serialization](#)

[An improvement on SessionFactory Initialization](#)

[Optimizing application startup time with Fluent NHibernate and uNhAddIns](#)

[NHibernate Hidden Gems - Speed up NHibernate startup time](#)

و حاصل تمام این مقالات در پروژه‌ی [Effectus](#) ، فایل Effectus\Infrastructure\BootStrapper.cs آن گردآوری شده است.

نظرات خوانندگان

نویسنده: hamidrezabina
تاریخ: ۱۳۸۹/۱۰/۲۷ ۱۲:۴۶:۳۱

با سلام. . .
میشه فایل PDF بخش NHibernate رو برای دانلود بزارید.

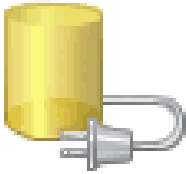
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۱۰/۲۷ ۱۷:۰۵:۰۹

فایل CHM وبلاگ قابل دریافت است. لطفا از آن استفاده کنید (همان فایل خلاصه وبلاگ در قسمت گزیده‌های ستون سمت راست سایت).

عنوان: سرویس جمع و مفرد سازی اسامی
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۱۰/۲۸ ۱۰:۲۰:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: NHibernate

اگر به Entity data model wizard در VS.Net 2010 دقت کرده باشید، گزینه‌ی "Pluralize or singularize generated object names" نیز به آن اضافه شده است:

Entity Data Model Wizard



Choose Your Database Objects

Which database objects do you want to include in your model?

☐

Tables

☐

Views

☐

Stored Procedures

☒

Pluralize or singularize generated object names

Model Namespace:

NorthwindModel

< Previous

Next >

این مورد از این جهت حائز اهمیت است که عموماً نام جداول در بانک اطلاعاتی، جمع است و نام کلاس متناظر ایجاد شده برای آن در کدهای برنامه بهتر است مفرد باشد. برای مثال نام جدول، Customers است و نام کلاس آن بهتر است Customer تعریف گردد. به این صورت کار کردن با آن توسط یک ORM با معناتر خواهد بود؛ زیرا زمانیکه یک وهله از شیء Customer ایجاد می‌شود، فقط یک رکورد از بانک اطلاعاتی مد نظر است؛ در حالیکه یک جدول مجموعه‌ای است از رکوردها.

زبان انگلیسی هم پر است از اسامی جمع و مفرد باقاعده و بی‌قاعده و کل عملیات با اضافه و حذف کردن یک s و یا es پایان نمی‌یابد؛ برای مثال phenomenon و phenomena را در نظر بگیرد تا Money و Moneys.

این امکان مهیا شده توسط Entity Framework 4.0 یا همان EF v2 با برنامه نویسی هم [قابل دسترسی است](#) و در اسمبلی System.Data.Entity.Design.dll و فضای نام System.Data.Entity.Design.PluralizationServices قرار گرفته است.

این اسمبلی جزئی از دات نت 4 است و اگر آن را توسط گزینه‌ی Add references در VS.NET مشاهده نمی‌کنید، علت آن است که در تنظیمات پروژه جاری، گزینه‌ی Target framework بر روی Client profile قرار گرفته است که باید به دات نت 4 کامل تغییر یابد.

استفاده از آن هم به صورت زیر است:

```
using System;
using System.Data.Entity.Design.PluralizationServices;
using System.Globalization;

namespace PluralizationServicesTest
{
    class Program
    {
        static void Main(string[] args)
        {
            var service = PluralizationService.CreateService(CultureInfo.GetCultureInfo("en"));
            Console.WriteLine(service.Pluralize("mouse"));
            Console.WriteLine(service.IsPlural("phenomena"));
        }
    }
}
```

ملاحظات:

این روش فعلاً به زبان انگلیسی محدود است و اگر Culture را به مورد دیگری تنظیم کنید با خطای "We don't support locales other than English yet" متوقف خواهید شد.

روش دیگر:

کتابخانه‌ی سورس باز Castle ActiveRecord نیز دارای کلاسی است به نام Inflector که برای همین منظور طراحی شده است:

[Inflector.cs](#)

کاربرد آن در Fluent NHibernate

در Fluent NHibernate کار نگاشت کلاس‌ها به جداول به صورت خودکار صورت می‌گیرد و همچنین تولید ساختار بانک اطلاعاتی نیز به همین نحو می‌باشد. اما می‌توان تولید نام جداول را سفارشی نیز نمود. برای مثال از کلاس Book به صورت خودکار ساختار جدولی به نام Books را تولید کند:

```
using FluentNHibernate.Conventions;
using FluentNHibernate.Conventions.Instances;
using NHibernate.Helper.Toolkit;

namespace NHibernate.Helper.MappingConventions
{
    public class TableNameConvention : IClassConvention
    {
        public void Apply(IClassInstance instance)
        {
            instance.Table(Inflector.Pluralize(instance.EntityType.Name));
        }
    }
}
```

و برای تزریق آن خواهیم داشت:

```
... = new AutoPersistenceModel()  
    .Where(...)  
    .Conventions.Setup(c =>c.Add<TableNameConvention>())  
    .AddEntityAssembly(...)  
...
```

این روزها هیچکدام از فناوری‌های دسترسی به داده بدون امکان یکپارچگی آن‌ها با سیستم‌ها و روش‌های متفاوت caching ، مطلوب شمرده نمی‌شوند. ایده اصلی caching هم به زبان ساده به این صورت است : فراهم آوردن روش‌هایی جهت میسر ساختن دسترسی سریعتر به داده‌هایی که به صورت متناوب در برنامه مورد استفاده قرار می‌گیرند، بجای مراجعه مستقیم به بانک اطلاعاتی و خواندن اطلاعات از دیسک سخت.

یکی از تفاوت‌های مهم NHibernate با اکثر ORM های موجود داشتن دو سطح متفاوت cache است : first level cache & second level cache .

برای نمونه Entity framework (در زمان نگارش این مطلب) تنها first level caching را پشتیبانی می‌کند و پروایدر توکار و یکپارچه‌ای را جهت second level caching ارائه نمی‌دهد.

در این قسمت قصد داریم First Level Cache را بررسی کنیم.

سطح اول caching در NHibernate چیست؟

سطح اول caching در تمام ORM هایی که آن‌را پشتیبانی می‌کنند مانند NHibernate ، در طول عمر یک تراکنش تعریف می‌گردد. در این حالت در طی یک تراکنش و طول عمر یک سشن، دریافت اطلاعات هر رکورد از بانک اطلاعاتی، تنها یکبار انجام خواهد شد؛ صرفنظر از اینکه کوئری دریافت اطلاعات آن چندبار فراخوانی می‌گردد. یکی از دلایل این روش هم آن است که هیچ دو شیء متفاوتی که هم اکنون در حافظه قرار دارند نباید بیانگر یک رکورد واحد از بانک اطلاعاتی باشند.

در NHibernate به صورت پیش فرض هر زمانیکه از شیء استاندارد session استفاده می‌کنید، سطح اول caching نیز فعال است. درست در زمانیکه سشن خاتمه می‌یابد، این سطح از caching نیز به صورت خودکار تخلیه خواهد گردید.

به first level caching اصطلاحاً thought-out cache system یا Cache Through pattern و یا identity map هم گفته می‌شود.

مثال:

روش متداول و استاندارد کار با NHibernate عموماً به صورت زیر است:

الف) دریافت شیء Session از Session Factory

ب) شروع یک تراکنش با فراخوانی متد BeginTransaction شیء Session

ج) برای مثال دریافت اطلاعات رکوردی با ID مساوی یک به کمک متد Get مرتبط با شیء Session : این اطلاعات مستقیماً از بانک اطلاعاتی دریافت خواهد شد.

د) سپس مجدداً سعی در دریافت رکوردی با ID مساوی یک. اینبار اطلاعات این شیء مستقیماً از cache خوانده می‌شود و رفت و برگشتی به بانک اطلاعاتی نخواهیم داشت. به همین جهت به این روش identity map هم گفته می‌شود، زیرا NHibernate بر اساس ID منحصر بفرد این اشیاء ، identity map خود را تشکیل می‌دهد.

ه) خاتمه‌ی سشن با فراخوانی متد Close آن

بلافاصله

الف) دریافت شیء Session از Session Factory

ب) شروع یک تراکنش با فراخوانی متد BeginTransaction شیء Session

ج) برای مثال دریافت اطلاعات رکوردی با ID مساوی یک به کمک متد Get مرتبط با شیء Session : این اطلاعات مستقیماً از بانک اطلاعاتی دریافت خواهد شد (زیرا در یک سشن جدید قرار داریم و همچنین سشن قبلی بسته شده و کش آن تخلیه گشته است).

د) خاتمه‌ی سشن با فراخوانی متد Close آن

سؤال: آیا استفاده از یک سشن سراسری در برنامه صحیح است؟

پاسخ: خیر!

توضیحات: زمانیکه از یک سشن سراسری استفاده می‌کنید، کش NHibernate را در اختیار تمام کاربران همزمان سیستم قرار داده‌اید. در طی یک سشن، همانطور که عنوان شد، بر اساس IDهای اشیاء، یک identity map تشکیل می‌شود و در این حالت به ازای هر رکورد بانک اطلاعاتی فقط و فقط یک شیء در حافظه وجود خواهد داشت که این روش در محیط‌های چندکاربره مانند برنامه‌های وب به زودی تبدیل به نشت اطلاعات و یا تخریب اطلاعات می‌گردد. به همین جهت در این نوع برنامه‌ها روش session-per-request بهترین حالت کاری است.

سؤال: حین به روز رسانی اشیاء جدید، به خطا بر می‌خورم. مشکل در کجاست؟

فرض کنید شیء مفروض Customer را توسط متد session.Get از بانک اطلاعاتی دریافت و تعدادی از خواص آن را جهت ساخت شیء جدیدی از کلاس Customer استفاده کرده‌ایم. اکنون اگر بخواهیم این شیء جدید را در بانک اطلاعاتی ذخیره یا به روز رسانی کنیم، NHibernate این اجازه را نمی‌دهد! چرا؟

پاسخ:

خطای متداول این حالت عموماً به صورت زیر است:

a different object with the same identifier value was already associated with the session

اگر شخصی با مکانیزم سطح اول caching در NHibernate آشنایی نداشته باشد، شاید ساعاتی را در انجمن‌های مرتبط، جهت یافتن روش حل خطای فوق سپری کند.

همانطور که عنوان شد، در طول یک سشن، نمی‌توان دو شیء با یک ID را به عنوان یک رکورد بانک اطلاعاتی مورد استفاده قرار داد. اولین فراخوانی Get، سبب کش شدن آن شیء در identity map سطح اول caching می‌گردد. راه حل:

الف) از چندین و چند شیء استفاده نکنید. هر رکورد باید تنها با یک وهله از شیء‌ای متناظر باشد.

ب) می‌توان پیش از update، کش سطح اول را به صورت دستی خالی کرد. برای این منظور از متد Clear شیء سشن استفاده کنید.

ج) بجای استفاده از متد saveOrUpdate شیء سشن، از متد Merge آن استفاده کنید. به این صورت شیء جدید ایجاد شده با شیء موجود در کش یکی خواهد شد.

د) می‌توان بجای تخلیه کل کش (حالت ب)، کش مرتبط با شیء Customer را به صورت دستی خالی کرد. برای این منظور از متد Evict شیء سشن استفاده نمایید.

و لازم به ذکر است که متد Flush سبب تخلیه کش نمی‌گردد. کار این متد اعمال کلیه تغییرات اعمالی موجود در کش به بانک اطلاعاتی است و بیشتر جهت هماهنگ سازی این دو مورد استفاده قرار می‌گیرد.

سؤال: آیا می‌توان سطح اول caching را غیرفعال کرد؟

پاسخ: بله.

توضیحات:

عموماً کلیه ORMs جهت Batching یا Bulk data operations (برای مثال ثبت تعداد زیادی رکورد یا به روز رسانی تعداد بالایی از آن‌ها، یا نمایش فقط خواندنی تعداد زیادی رکورد و گزارشگیری از آن‌ها) کارایی مطلوبی ندارند. نمونه‌ای از آن‌را در مبحث جاری ملاحظه کرده‌اید. هر شیء‌ای که به نحوی به سشن جاری وارد می‌شود تحت نظر قرار می‌گیرد و این مورد در تعداد بالای ثبت یا به روز رسانی رکوردها، یعنی کاهش سرعت و کارایی، به علاوه مصرف بالای حافظه. به همین جهت باید به خاطر داشت که ORMs جهت سناریوهای [OLTP](#) مناسب هستند و کسانی که سرعت و کارایی ORMs را با Batch processing اندازه گیری می‌کنند، کلاً درکی از فلسفه وجودی ORMs و ساختار درونی آن‌ها ندارند!

خوشبختانه NHibernate با معرفی Stateless Sessions بر این مشکل فائق آمده است. در اینجا بجای ISession تنها کافی است از [IStatelessSession](#) استفاده گردد:

```
using (IStatelessSession statelessSession = sessionFactory.OpenStatelessSession())
using (ITransaction transaction = statelessSession.BeginTransaction())
```

```
{  
  //now insert 1,000,000 records!  
}
```

در این حالت سیستم دو مزیت عمده را تجربه خواهد کرد: سرعت بالای ثبت اطلاعات با تعداد زیاد رکورد و همچنین مصرف پایین حافظه از آنجائیکه یک `ISession` ارجاعی را به اشیایی که بارگذاری می‌کند، در خود نگهداری نخواهد کرد. تنها باید به خاطر داشت که در این حالت `lazy loading` پشتیبانی نمی‌شود و همچنین رخدادهای درونی NHibernate نیز لغو خواهند شد.

نظرات خوانندگان

نویسنده: Afshar Mohebbi
تاریخ: ۰۹:۱۸:۴۳ ۱۳۸۹/۱۱/۱۷

سپاس

عموما دو الگوی اصلی caching در برنامه‌ها وجود دارند: cache aside و cache trough . در الگوی cache trough ، سیستم caching داخل DAL (که در اینجا همان NHibernate است)، تعبیه می‌شود؛ مانند سطح اول caching که پیشتر در مورد آن صحبت شد. در این حالت cache از دید سایر قسمت‌های برنامه مخفی است و DAL به صورت خودکار آن را مدیریت می‌کند. در الگوی cache aside ، کار مدیریت سیستم caching دستی است و خارج از NHibernate قرار می‌گیرد و DAL هیچگونه اطلاعی از وجود آن ندارد. در این حالت لایه caching موظف است تا هنگام به روز شدن بانک اطلاعاتی، اطلاعات خود را نیز به روز نماید. این لایه عموماً توسط سایر شرکت‌ها یا گروه‌ها برنامه نویسی تهیه می‌شود. NHibernate جهت سهولت کار با این نوع cache providers خارجی، نقاط تزریق ویژه‌ای را تدارک دیده است که به second level cache معروف است. هدف از second level cache فراهم آوردن دیدی کش شده از بانک اطلاعاتی است تا فراخوانی‌های کوثری‌ها به سرعت و بدون تماس با بانک اطلاعاتی صورت گیرد. در حال حاضر (زمان نگارش این مطلب)، entity framework این لایه‌ی دوم caching یا به عبارتی دیگر، امکان تزریق ساده‌تر cache providers خارجی را به صورت توکار ارائه نمی‌دهد. در NHibernate طول عمر second level cache در سطح session factory (یا به عبارتی طول عمر تمام برنامه) تعریف می‌شود و برخلاف سطح اولیه caching محدود به یک سشن نیست. در این حالت هر زمانیکه یک موجودیت به همراه ID منحصر بفرد آن تحت نظر NHibernate قرارگیرد و همچنین سطح دوم caching نیز فعال باشد، این موجودیت در تمام سشن‌های برنامه بدون نیاز به مراجعه به بانک اطلاعاتی در دسترس خواهد بود (بنابراین باید دقت داشت که هدف از این سیستم، کار سریعتر با اطلاعاتی است که سطح دسترسی عمومی دارند).

در ادامه لیستی از cache providers خارجی مهیا جهت استفاده در سطح دوم caching را ملاحظه می‌نمائید:

AppFabric Caching Services : بر اساس Microsoft's AppFabric Caching Services که یک پلتفرم caching محسوب می‌شود (+). (این پروژه پیشتر به نام Velocity معروف شده بود و قرار بود تنها برای ASP.NET ارائه شود که سیاست آن به گونه‌ای جامع‌تر تغییر کرده است)

MemCache : بر اساس سیستم معروف MemCached تهیه شده است (+).

NCache : (+)

ScaleOut : (+)

Prevalence : (+)

SysCache : بر اساس همان روش آشنای متداول در برنامه‌های ASP.NET به کمک System.Web.Caching.Cache کار می‌کند؛ یا به قوی همان IIS caching

SysCache2 : همانند SysCache است با این تفاوت که SQL dependencies ویژه SQL Server را نیز پشتیبانی می‌کند.

SharedCache : یک سیستم distributed caching نوشته شده برای دات نت است (+).

این موارد و پروایدها جزو پروژه‌ی nhcontrib در سایت سورس فورج هستند (+).

مطالب تکمیلی:

[مستندات NHibernate](#)

[توضیحات مفصلی در مورد سطح اول و دوم caching در NHibernate](#)

[مقایسه‌ای در مورد مبحث caching در EF و NHibernate](#)

[چگونه fluent NHibernate را جهت استفاده از سطح دوم caching تنظیم کنیم؟](#)
[توضیحات جامعی در مورد استفاده از SysCache](#)

نظرات خوانندگان

نویسنده: Ahmad

تاریخ: ۱۳۹۰/۰۱/۱۲ ۱۸:۲۷:۰۱

سلام

سطح دوم فقط برای وب استفاده میشه یا برای ویندوز اپلیکیشن هم کاربرد داره؟
درک کردن سطح دوم یه کمی سخته!!!

نویسنده: وحید نصیری

تاریخ: ۱۳۹۰/۰۱/۱۲ ۳۰:۰۶:۰۱

برای برنامه‌های تک کاربره شاید لزومی نداشته باشه چون هدف اصلی آن کم کردن بار سرور است
یک سری از موارد سایت عمومی است مثلاً آمار سایت حالا سوال اینجا است که اگر 10 هزار نفر هم اکنون به سایت مراجعه
کردند باید 10 هزار بار به بانک اطلاعاتی جهت دریافت یک سری اطلاعات با سطح دسترسی عمومی مراجعه کرد؟ اینجا است که
سطح دوم کش ذکر شده معنا پیدا می‌کند

با کمک Fluent NHibernate می‌توان نگاشت‌ها را به دو صورت [خودکار](#) و یا [دستی](#) تعریف کرد. در حالت خودکار، روابط بین کلاس‌ها بررسی شده و بدون نیاز به تعریف هیچگونه ویژگی (attribute) خاصی بر روی فیلدها، امکان تشخیص خودکار حالت‌های کلید خارجی، روابط یک به چند، چند به چند و امثال آن وجود دارد. یا اگر نیاز باشد تا اسکرپیت تولیدی جهت به روز رسانی بانک اطلاعاتی، طول خاصی را به فیلدی اعمال کند می‌توان از ویژگی‌های [NHibernate validator](#) استفاده کرد؛ مانند تعریف طول و نال نبودن یک فیلد که علاوه بر بکارگیری اطلاعات آن در حین تعیین اعتبار ورودی دریافتی، بر روی نحوه‌ی به روز رسانی بانک اطلاعاتی هم تاثیر گذار است:

```
public class Product
{
    public virtual int Id { set; get; }

    [Length(120)]
    [NotNullNotEmpty]
    public virtual string Name { get; set; }

    public virtual decimal UnitPrice { get; set; }
}
```

این نگاشت خودکار یا AutoMapper، تقریباً در 90 درصد موارد کافی است. فیلد Id را بر اساس یک سری پیش فرض‌هایی که این مورد هم قابل تنظیم است به صورت primary key تعریف می‌کند، طول فیلدها و نحوه‌ی پذیرفتن نال آن‌ها، از ویژگی‌های NHibernate validator گرفته می‌شود و روابط بین کلاس‌ها به صورت خودکار به روابط یک به چند و مانند آن ترجمه می‌شود و نیازی نیست تا کلاسی جداگانه را جهت مشخص سازی صریح این موارد تهیه کرد، یا ویژگی مشخص کننده‌ی دیگری را به فیلدها افزود. اما اگر برای مثال بخواهیم در این کلاس فیلد Name را به صورت Unique معرفی کنیم چه باید کرد؟ به عبارتی تمام آنچه که ویژگی AutoMapper در Fluent NHibernate انجام می‌دهد، بسیار هم عالی؛ اما فقط می‌خواهیم مقادیر یک فیلد منحصر بفرد باشد. برای این منظور اینترفیس [IAutoMappingOverride](#) تدارک دیده شده است:

```
public class ProductCustomMappings : IAutoMappingOverride<Product>
{
    public void Override(AutoMapping<Product> mapping)
    {
        mapping.Id(u => u.Id).GeneratedBy.Identity(); // ضروری است
        mapping.Map(p => p.Name).Unique();
    }
}
```

در حالت استفاده از اینترفیس IAutoMappingOverride مشخص سازی نحوه‌ی تولید primary key ضروری است و سپس برای نمونه، فیلد Name به صورت منحصر بفرد تعریف می‌گردد. در اینجا کل عملیات هنوز از روش AutoMapper پیروی می‌کند اما فیلد Name علاوه بر اعمال ویژگی‌های NHibernate validator، به صورت منحصر بفرد نیز معرفی خواهد شد.

اگر با SQL Server کار کرده باشید حتما با مفهوم و امکان [Computed columns](#) (فیلدهای محاسبه شده) آن آشنایی دارید. چقدر خوب می‌شد اگر این امکان برای سایر بانک‌های اطلاعاتی که از تعریف فیلدهای محاسبه شده پشتیبانی نمی‌کنند، نیز مهیا می‌شد. زیرا یکی از اهداف مهم استفاده‌ی صحیح از ORMs، مستقل شدن برنامه از نوع بانک اطلاعاتی است. برای مثال امروز می‌خواهیم با MySQL کار کنیم، ماه بعد شاید بخواهیم یک نسخه‌ی سبک‌تر مخصوص کار با SQLite را ارائه دهیم. آیا باید قسمت دسترسی به داده برنامه را از نو بازنویسی کرد؟ اینکار در NHibernate فقط با تغییر نحوه‌ی اتصال به بانک اطلاعاتی میسر است و نه بازنویسی کل برنامه (و صد البته شرط مهم و اصلی آن هم این است که از امکانات ذاتی خود NHibernate استفاده کرده باشید. برای مثال وسوسه‌ی استفاده از رویه‌های ذخیره شده را فراموش کرده و به عبارتی ORM مورد استفاده را به امکانات ویژه‌ی یک بانک اطلاعاتی گره نزنید).

خوشبختانه در NHibernate امکان تعریف فیلدهای محاسباتی با کمک تعریف نگاشت خواص به صورت فرمول مهیا است. برای توضیحات بیشتر لطفا به مثال ذیل دقت بفرمائید:

در ابتدا کلاس کاربر تعریف می‌شود:

```
using System;
using NHibernate.Validator.Constraints;

namespace FormulaTests.Domain
{
    public class User
    {
        public virtual int Id { get; set; }

        [NotNull]
        public virtual DateTime JoinDate { set; get; }

        [NotNullNotEmpty]
        [Length(450)]
        public virtual string FirstName { get; set; }

        [NotNullNotEmpty]
        [Length(450)]
        public virtual string LastName { get; set; }

        [Length(900)]
        public virtual string FullName { get; private set; } // می‌گردد
        public virtual int DayOfWeek { get; private set; } // می‌گردد
    }
}
```

در این کلاس دو خاصیت FullName و DayOfWeek به صورت فقط خواندنی به کمک private set ذکر شده، تعریف گردیده‌اند. قصد داریم روی این دو خاصیت فرمول تعریف کنیم:

```
using FluentNHibernate.Automapping;
using FluentNHibernate.Automapping.Alterations;

namespace FormulaTests.Domain
{
    public class UserCustomMappings : IAutoMappingOverride<User>
    {
        public void Override(AutoMapping<User> mapping)
        {
            mapping.Id(u => u.Id).GeneratedBy.Identity(); // ضروری است
            mapping.Map(x => x.DayOfWeek).Formula("DATEPART(dw, JoinDate) - 1");
            mapping.Map(x => x.FullName).Formula("FirstName + ' ' + LastName");
        }
    }
}
```

نحوه‌ی انتساب فرمول‌های مبتنی بر SQL را در نگاشت فوق ملاحظه می‌نمائید. برای مثال FullName از جمع دو فیلد نام و نام خانوادگی حاصل خواهد شد و DayOfWeek از طریق فرمول SQL دیگری که ملاحظه می‌نمائید (یا هر فرمول SQL دلخواه دیگری که صلاح می‌دانید).

اکنون اگر Fluent NHibernate را وادار به تولید اسکریپت متناظر با این دو کلاس کنیم حاصل به صورت زیر خواهد بود:

```
create table Users (
    UserId INT IDENTITY NOT NULL,
    JoinDate DATETIME not null,
    FirstName NVARCHAR(450) not null,
    LastName NVARCHAR(450) not null,
    primary key (UserId)
)
```

همانطور که ملاحظه می‌کنید در اینجا خبری از دو فیلد محاسباتی تعریف شده نیست. این فیلدها در تعاریف نگاشت‌ها به صورت خودکار ظاهر می‌شوند:

```
<hibernate-mapping xmlns="urn:hibernate-mapping-2.2"
  default-access="property" auto-import="true" default-cascade="none" default-lazy="true">
  <class xmlns="urn:hibernate-mapping-2.2" mutable="true"
    name="FormulaTests.Domain.User, FormulaTests, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"
    table="Users">
    <id name="Id" type="System.Int32" unsaved-value="0">
      <column name="UserId" />
      <generator class="identity" />
    </id>
    <property name="DayOfWeek" formula="DATEPART(dw, JoinDate) - 1" type="System.Int32" />
    <property name="FullName" formula="FirstName + ' ' + LastName" type="System.String" />
    <property name="JoinDate" type="System.DateTime">
      <column name="JoinDate" />
    </property>
    <property name="FirstName" type="System.String">
      <column name="FirstName" />
    </property>
    <property name="LastName" type="System.String">
      <column name="LastName" />
    </property>
  </class>
</hibernate-mapping>
```

اکنون اگر کوئری زیر را در برنامه اجرا نمائیم:

```
var list = session.Query<User>.ToList();
foreach (var item in list)
{
    Console.WriteLine("{0}:{1}", item.FullName, item.DayOfWeek);
}
```

به صورت خودکار به SQL ذیل ترجمه خواهد شد و اکنون نحوه‌ی بکارگیری فیلدهای فرمول، بهتر مشخص می‌گردد:

```
select
  user0_.UserId as UserId0_,
  user0_.JoinDate as JoinDate0_,
  user0_.FirstName as FirstName0_,
  user0_.LastName as LastName0_,
  DATEPART(user0_.dw, user0_.JoinDate) - 1 as formula0_, --- همان فرمول تعریف شده است
  user0_.FirstName + ' ' + user0_.LastName as formula1_ --- از طریق فرمول تعریف شده حاصل گردیده
است
from
  Users user0_
```

نظرات خوانندگان

نویسنده: Anonymous
تاریخ: ۲۰:۲۵:۵۹ ۱۳۸۹/۱۲/۰۵

سلام آقای نصیری.
فرض کنید کلاسی مانند زیر وجود دارد:

```
public class Project
{
    { ;public virtual int Id { get; set
    { ;public virtual long ProjectCode { get; set
    { ;public virtual string ProjectName { get; set
    { ;public virtual int CreateDate { get; set
    {
```

در فیلد CreateDate مقادیر زیر وجود دارد:

```
CreateDate
890102
891210
... و
```

که تاریخ شروع پروژه ها می باشد. سوال من اینجاست که در NH کجا باید این تاریخ ها رو به 02/01/89 و 10/12/89 تبدیل کنم و در UI به کاربر نشون بدم.
با تشکر فراوان.

نویسنده: وحید نصیری
تاریخ: ۲۰:۳۵:۰۰ ۱۳۸۹/۱۲/۰۵

سلام

دقیقا مانند مثال فوق عمل کنید. یک خاصیت private set دار را همانند مثال فوق اضافه کنید، مثلا PersianDate از نوع string . سپس فرمولی را باید به آن در قسمت CustomMappings ذکر شده انتساب داد. برای اینکار از همان روش‌های مرسوم cast استفاده کنید به همراه substring تا بشود ابتدا مقدار عددی را به رشته تبدیل کرد و سپس با substring قسمت‌های مختلف را جدا کرد و نهایتا به هم چسباند. فقط باید دقت داشت که این فرمول باید یک فرمول معتبر SQL ایی باشد.

نویسنده: Anonymous
تاریخ: ۲۳:۳۶:۱۱ ۱۳۸۹/۱۲/۰۵

ممنون کاملا متوجه شدم.

حالا اگر بخواهیم از توابع غیر SQL استفاده کنیم باید چکار کنیم؟ برای مثال بخواهیم همین مثال بالا رو با توابع نوشته شده توسط خودمون انجام بدیم.

نویسنده: وحید نصیری
تاریخ: ۰۱:۵۱:۲۶ ۱۳۸۹/۱۲/۰۶

ببینید، توابع ویژه نمایشی سی شارپ شما، یعنی سمت کلاینت. موضوع بحث فوق سمت سرور بانک اطلاعاتی است. مقادیر در سمت سرور مطابق فرمول شما تشکیل می‌شوند. به آخرین کوئری ذکر شده در مطلب فوق دقت کنید. در حال حاضر فقط SQL Server است که امکان استفاده از توابع دات نت را هم سمت سرور میسر کرده (از نگارش 2005 به بعد). بنابراین اگر می‌خواهید توابع ویژه‌ای را در همان سمت سرور اعمال کنید که منطق آن مثلا با سی شارپ پیاده سازی شده، باید یک CLR function مخصوص اس کیوال سرور درست کنید. بعد فرمول نگاشت فوق را بر اساس این CLR function تعیین کنید و کار می‌کند. چیزی

شبيه به همان آخرين كوئري تشكيل شده را خواهيد داشت. خلاصه اينكه به نحوی بايد اين پياده سازی دات نتی خودتون رو به سمت سرور ببريد.

اما سمت كلاينت شما هر کاری را می‌توانيد انجام دهيد. برای مثال زمان نمایش اطلاعات در WPF یا سیلورلايت از یک Converter استاندارد آن (با پياده سازی اينترفيس IValueConverter) در حين Binding استفاده كنيد. اگر با ASP.NET Webforms کار می‌کنيد حين نمایش اطلاعاتی که هم اکنون در سمت كلاينت مهيا است، مثلا جهت نمایش در یک GridView یا موارد مشابه شما خواهيد داشت myFunc(Eval("field")) و شبيه به اين که myFunc بايد در کديهيانده شما پياده سازی شود. در ساير فناوری‌ها که می‌تواند شامل موارد قبل هم باشند، نهايتا شما یک ليست دريافتی از سرور را داريد، یک حلقه با LINQ یا حالت معمولی تشكيل شده و مقادير مدل مورد نظر ويرايش می‌شوند تا جهت نمایش مناسب شوند. تمام این‌ها در حالتی است که قصد شما فقط و فقط تغيير نحوه‌ی نمایش است. به عبارتی الان کل دیتای فیلتر شده سمت کاربر مهيا است. شما می‌خواهيد به آن شکل دهيد.

حالت ديگر (حالت غير نمایشی و استفاده در كوئري‌ها):

اگر با LINQ کمی بیشتر از اطلاعات موجود در وب کار کرده باشید احتمالا به این سوال رسيده‌ايد که آیا می‌شود متد سفارشی خودمان را هم حين تهيه كوئري‌هایی از این دست استفاده كنيم؟ چون فقط یک سری extension method مشخص بیشتر وجود ندارند. اگر من extension method سفارشی خودم را تهيه کردم چگونه؟

این سوال دو پاسخ دارد:

- متدهای سفارشی شما حتما روی کل اطلاعات دريافتی از سرور کار می‌کنند؛ اما بهينه نیستند. چون برای مثال myFunc سی شارپ من معادل SQL ایی ندارد که بتوانم مستقيما آن‌را سمت سرور اجرا کنم. چون نهايتا LINQ to NHibernate بايد به SQL یا T-SQL ترجمه شود. به همین جهت مجبورم کل اطلاعات را دريافت کنم، مثلا 100 هزار رکورد، حالا که اشیاء دات نتی من تشكيل و کامل شده، متد سفارشی LINQ خودم را بر روی این‌ها اجرا می‌کنم. این روش کار می‌کنه ولی از لحاظ کارآیی فاجعه است.

- روش ديگر: در NH 3.0 این امکان وجود دارد ... بسط پروايدر LINQ آن با صور مختلف. که اگر وقت شد یک مطلب کامل در مورد آن خواهم نوشت.

نویسنده: Anonymous

تاریخ: ۱۳۸۹/۱۲/۰۶ ۱۵:۱۵:۲۲

از پاسخگویی شما بسیار ممنونم. من هر روز از شما مطلب جدیدی یاد می‌گیرم. من قصد كشدار کردن بحث رو ندارم و اينم آخرين ارسال من در مورد این بحث است. فکر می‌کنم نتونستم منظورم رو واضح برسونم. فرض كنيم كلاس زیر وجود داره:

```
public class Project
{
    public virtual int Id { get; set }
    public virtual long ProjectCode { get; set }
    public virtual string Name { get; set }
    public virtual int CreateDate { get; set }

    public virtual string SepratedDate
    {
        get { return myFunc(CreateDate); }
        private set
        {
        }
```

من می‌خواهم در متد زیر لیستی از كلاس بالا رو به DataSet تبدیل کنم:

```
(public DataSet dsGetAll(bool includeArchived
{
}
```

```

using (var repository = new Repository
    }

var projects = repository.Find(x => x.IsArchive == includeArchived
    );

var ds = new CollectionToDataSet>(projects.ToList
    );

return ds.CreateDataSet
    {
    {

ولی خطا می ده که SepratedDate در جدول وجود نداره!!!
        {".Invalid column name 'SepratedDate'"}
        could not execute query

select project0_Id as Id15_, project0_ProjectCode as ProjectC2_15_, project0_Name as Name15_, ]
project0_IsArchive as IsArchive15_, project0_CreateDate as CreateDate15_, project0_SepratedDate as
Seprated6_15_ from tblProject project0_ where case when project0_IsArchive=1 then 'true' else 'false'
end=case when @p0='true' then 'true' else 'false' end
    
```

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۱۲/۰۶ ۱۷:۲۱:۱۳

- راه یک: مطالب مقاله فوق. یک قسمت آن custom mapping است که می‌گه لطفا این فیلد رو در کوئری با فرمول تشکیل بده نه با همین فیلدی که من اینجا اضافه کردم. این رو ندید بگیر، بجاش در SQL نهایی یک فرمول بذار، نه صاف همین فیلد رو تا من به خطا برخورد کنم.

- راه دو: مطالب کامنت قبل. (یعنی از زمان داشتن ToList که همه چیز سمت کلاینت است به بعد ... هر کاری دوست داشتید با این اطلاعات انجام دهید)

- راه سه: در همان قسمت custom mappings می‌شود نوشت map.IgnoreProperty الی آخر. به این صورت خاصیت تعریف شده شما در کوئری SQL ظاهر نمی‌شود تا مشکل درست کند. اطلاعات بیشتر: [\(+\)](#)

نویسنده: Anonymous
تاریخ: ۱۳۸۹/۱۲/۰۶ ۱۹:۰۵:۰۷

!!!Thanks. Excellent

عنوان: به روز رسانی ارجاعات یک اسمبلی دارای امضای دیجیتال بدون کامپایل مجدد

نویسنده: وحید نصیری

تاریخ: ۱۳۹۰/۰۱/۱۰ ۰۹:۰۹:۰۰

آدرس: www.dotnettips.info

برچسب‌ها: NHibernate

سؤال: امروز NHibernate به روز شده اما Fluent NHibernate خیر! چکار باید کرد؟!

Fluent NHibernate کتابخانه‌ای است جهت رهایی برنامه نویس‌ها از نوشتن فایل‌های XML نگاشت کلاس‌ها به جداول به همراه قابلیت‌های دیگری مانند نگاشت خودکار و غیره. بنابراین این کتابخانه بدون NHibernate اصلی بدون کاربرد است. تیم توسعه آن هم با تیم اصلی NHibernate یکی نیست. عموماً NHibernate به روز می‌شود اما Fluent NHibernate ممکن است تا دو ماه بعد از آن هم به روز نشود. در این مواقع چه باید کرد؟

دو کار را می‌توان انجام داد:

الف) سورس Fluent NHibernate را دریافت کنیم و سپس ارجاعات قبلی به NHibernate قدیمی را حذف و ارجاعاتی را به اسمبلی‌های جدید آن اضافه و پروژه را کامپایل کنیم. Fluent NHibernate در طی این مدت به اندازه کافی رشد کرده و به قولی پخته است. کاری را هم که ادعا می‌کند به خوبی انجام می‌دهد. اما چون اسمبلی‌های اصلی NHibernate و همچنین Fluent NHibernate دارای امضای دیجیتال هستند، نمی‌توان از اسمبلی‌های جدید NHibernate به همراه Fluent NHibernate قدیمی استفاده کرد. خطای حاصل شبیه به عبارات ذیل خواهد بود:

```
System.IO.FileLoadException: Could not load file or assembly 'nameOfAssembly', Version=specificVersion, Culture=neutral, PublicKeyToken=publicKey' or one of it's dependencies. The located assembly's manifest definition does not match the assembly reference. (Exception from HRESULT: 0x80131040)
```

حذف ارجاعات به NHibernate قدیمی و افزودن مجدد ارجاعات به فایل‌های جدید و کامپایل نهایی پروژه یک راه حل است.

ب) راه حل دیگر استفاده از ویژگی [bindingRedirect](#) است بدون دریافت سورس، حذف و افزودن ارجاعات و کامپایل مجدد:

```
<runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
    <dependentAssembly>
      <assemblyIdentity name="NHibernate"
        publicKeyToken="aa95f207798dfdb4"
        culture="neutral" />
      <bindingRedirect oldVersion="3.0.0.4000"
        newVersion="3.1.0.4000"/>
    </dependentAssembly>
  </assemblyBinding>
</runtime>
```

در این مثال، پس از افزودن تعاریف فوق به فایل config برنامه، به سادگی می‌توان از اسمبلی اصلی NHibernate دارای نگارش 3.1.0.4000 به جای اسمبلی قدیمی‌تر 3.0.0.4000 آن استفاده کرد (همان نگارشی که Fluent NHibernate ما بر اساس آن کامپایل شده) و دیگر نیازی هم به کامپایل مجدد پروژه‌ای که از یک اسمبلی قدیمی Fluent NHibernate استفاده می‌کند، نخواهد بود.

اگر علاقمند باشید که به آخرین نگارش‌های Fluent NHibernate دسترسی داشته باشید، مکان اصلی نگهداری و Build آن‌ها در سایت teamcity.codebetter.com می‌باشد. ثبت نام در آن رایگان است و سپس در آدرس ذیل می‌توانید آخرین Build ها را مشاهده و دریافت کنید:

Fluent NHibernate > Fluent NHibernate 1.x (NH3.x)





























Overview History Change Log Issue Log Statistics Compatible Agents (3) Pending Changes (0)

Current status

No pending changes
Idle

Recent history

[\[all history\]](#)

#	Results	Artifacts	Changes	Started
#1.2.0.705	✓ Success	 View	james (1)	25 Mar 11 0
#1.2.0.704	✓ Success	 fluentnhibernate-docs-1.2.0.695.zip	1.32Mb	
#1.2.0.703	✓ Success	 fluentnhibernate-docs-1.2.0.696.zip	1.32Mb	
#1.2.0.702	✓ Success	 fluentnhibernate-docs-1.2.0.697.zip	1.32Mb	
#1.2.0.701	✓ Success	 fluentnhibernate-docs-1.2.0.698.zip	1.32Mb	
#1.2.0.701	✗ Failure	 fluentnhibernate-docs-1.2.0.702.zip	1.32Mb	
#1.2.0.700	✗ Failure	 fluentnhibernate-docs-1.2.0.703.zip	1.33Mb	
#1.2.0.699	✗ Failure	 fluentnhibernate-docs-1.2.0.704.zip	1.33Mb	
#1.2.0.699	✗ Failure	 fluentnhibernate-docs-1.2.0.705.zip	1.33Mb	
#1.2.0.698	✓ Success	 fluentnhibernate-NH3.0-binary-1.2.0.695.zip	4.0Mb	
#1.2.0.697	✓ Success	 fluentnhibernate-NH3.0-binary-1.2.0.696.zip	4.0Mb	
#1.2.0.696	✓ Success	 fluentnhibernate-NH3.0-binary-1.2.0.697.zip	4.0Mb	
#1.2.0.696	✓ Success	 fluentnhibernate-NH3.0-binary-1.2.0.698.zip	4.01Mb	
Showing 10 builds, see entire history				
Permalinks You can bookmark these links for quicker navigation  Last successful build  Last pinned build				
 Subscribe to finished builds or customize a feed				
Help Feedback				
		 fluentnhibernate-NH3.0-source-1.2.0.695.zip	10.97Mb	
		 fluentnhibernate-NH3.0-source-1.2.0.696.zip	10.97Mb	
		 fluentnhibernate-NH3.0-source-1.2.0.697.zip	10.97Mb	
		 fluentnhibernate-NH3.0-source-1.2.0.698.zip	10.97Mb	
		 fluentnhibernate-NH3.1-binary-1.2.0.702.zip	3.97Mb	
		 fluentnhibernate-NH3.1-binary-1.2.0.703.zip	3.97Mb	
		 fluentnhibernate-NH3.1-binary-1.2.0.704.zip	3.97Mb	
		 fluentnhibernate-NH3.1-binary-1.2.0.705.zip	3.96Mb	
		 fluentnhibernate-NH3.1-source-1.2.0.702.zip	10.63Mb	
		 fluentnhibernate-NH3.1-source-1.2.0.703.zip	10.63Mb	
		 fluentnhibernate-NH3.1-source-1.2.0.704.zip	10.64Mb	
		 fluentnhibernate-NH3.1-source-1.2.0.705.zip	10.62Mb	

[\(Fluent NHibernate v1.x \(NH3.x](#)

برای نمونه:

[fluentnhibernate-NH3.1-source-1.2.0.705.zip](#)

[fluentnhibernate-NH3.1-binary-1.2.0.705.zip](#)

[fluentnhibernate-docs-1.2.0.705.zip](#)

عنوان: NH 3.2 و تاثیر آن بر آینده‌ی FHN

نویسنده: وحید نصیری

تاریخ: ۱۳۹۰/۰۱/۲۷ ۱۲:۰۸:۰۰

آدرس: www.dotnettips.info

برچسب‌ها: NHibernate

در این عنوان، NH همان NHibernate است و FHN همان [Fluent NHibernate](#)

نگارش آزمایشی NH 3.2 هم اکنون در دسترس است و یکی از مهمترین مباحثی را که پوشش داده، جایگزین کردن فایل‌های XML [تهیه نگاشت‌ها با کدنویسی](#) است. دقیقا چیزی شبیه به Fluent NHibernate البته اینبار از یک کتابخانه دیگر به نام [ConfOrm](#) کدها یکی شده‌اند.

باید توجه داشت که نگارش 3.2 خاصیت [AutoMapping](#) مربوط به FHN را پشتیبانی نمی‌کند (یا هنوز در این نگارش به این حد نرسیده است)، بنابراین نمی‌تواند جایگزین صد در صدی برای FHN باشد اما باز هم تا حدود 75 درصد کار FHN را پوشش می‌دهد و می‌تواند علاقمندان را از این وابستگی خارجی (!) نجات دهد.

و ... این مساله نویسنده‌ی اصلی FHN را کمی دلگیر کرده است که آیا FHN را ادامه دهد یا خیر. اصل مطلب رو می‌تونید [اینجا](#) [بخوانید](#).

نظر بعضی‌ها هم در این بین این بوده!

ConfOrm looks like lipstick on a pig as far as fluent interfaces go

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۱/۲۹ ۱۲:۱۲:۲۳

ادامه در اینجا!

[me on Fluent NHibernate](#)

سطح اول کش در NHibernate در یک تراکنش معنا پیدا می‌کند ([+](#))؛ اما نتایج حاصل از اعمال سطح دوم ([+](#)) آن، در اختیار تمام تراکنش‌های جاری برنامه خواهند بود. در ادامه قصد داریم نحوه فعال سازی سطح دوم کش NHibernate را توسط Fluent NHibernate بررسی کنیم.

الف) دریافت کش پروایدر
برای این منظور به صفحه اصلی آن در سایت سورس فورج مراجعه نمائید ([+](#)). اگر به علت تحریم‌ها امکان دریافت فایل‌های مرتبط را نداشتید از این برنامه استفاده کنید ([+](#)). پس از دریافت، می‌خواهیم نحوه فعال سازی NHibernate.Caches.SysCache.dll را بررسی کنیم (این اسمبلی، در برنامه‌های وب و دسکتاپ بدون مشکل کار می‌کند).

ب) اعمال به قسمت تعاریف اولیه
پس از دریافت اسمبلی NHibernate.Caches.SysCache.dll و افزودن ارجاعی به آن، اکنون نوبت به معرفی آن به تنظیمات Fluent NHibernate می‌باشد. این کار هم بسیار ساده است:

```
...
.ConnectionString(x => x.FromConnectionStringWithKey(...))
.Cache(x => x.UseQueryCache()
    .UseMinimalPuts()
    .ProviderClass<NHibernate.Caches.SysCache.SysCacheProvider>())
...
```

ج) تعریف نوع کش در هنگام ایجاد نگاشت‌ها
اگر از ClassMap‌ها برای تعریف نگاشت‌ها استفاده می‌کنید، در انتهای تعاریف یک سطر Cache.ReadWrite را اضافه کنید.
اگر از AutoMapping استفاده می‌کنید، نیاز است تا با استفاده از IAutoMappingOverride ([+](#)) سطر یاد شده اضافه گردد؛ برای مثال:

```
using FluentNHibernate.Automatic.Alterations;
namespace NH3Test.MappingDefinitions.Domain
{
    public class AccountOverrides : IAutoMappingOverride<Account>
    {
        public void Override(FluentNHibernate.Automatic.AutoMapping<Account> mapping)
        {
            mapping.Cache.ReadWrite();
        }
    }
}
```

تعریف یک سطر فوق هم مهم است؛ زیرا در غیراینصورت فقط primary key حاصل از بار اول فراخوانی کوئری‌های مرتبط کش می‌شوند؛ نه نتیجه عملیات. هرچند این مورد هم یک قدم مثبت به شمار می‌رود از این لحاظ که برای مثال تهیه نتایج کوئری بر روی فیلدی که ایندکس بر روی آن تعریف نشده است همیشه از حالت تهیه کوئری بر روی فیلد دارای ایندکس کندتر است. اما هدف ما در اینجا این است که پس از بار اول فراخوانی کوئری، بارهای دوم و بعدی دیگر کوئری خاصی را به بانک اطلاعاتی ارسال نکرده و نتایج از کش خوانده شوند (جهت استفاده عموم کاربران در کلیه تراکنش‌های جاری برنامه).

د) اعمال متد Cacheable به کوئری‌ها
سه مرحله قبل نحوه برپایی مقدماتی سطح دوم کش را بیان می‌کنند و تنها یکبار نیاز است انجام شوند. در ادامه هر جایی که نیاز

داشتیم نتایج کوئری مورد نظر کش شوند (و باید دقت داشت که این کش شدن سطح دوم به معنی در دسترس بودن نتایج آن جهت تمام کاربران برنامه در تمام تراکنش‌های جاری برنامه هستند؛ برای مثال نتایج آمار سایت که دسترسی عمومی دارد) تنها کافی است متد Cacheable را به کوئری مورد نظر اضافه کرد؛ برای مثال:

```
var data = session.QueryOver<Account>()
    .Where(s => s.Name == "name")
    .Cacheable()
    .List();
```

ه) چگونه صحت اعمال سطح دوم کش را بررسی کنیم؟

برای بررسی این امر باید به خروجی SQL نهایی مراجعه کرد ([+](#)). سه تراکنش مجزا را تعریف کنید. در تراکنش اول یک insert ساده، در تراکنش دوم کوئری از اطلاعات قبل (به همراه اعمال متد Cacheable) و در تراکنش سوم مجددا همان کوئری تراکنش دوم را (به همراه اعمال متد Cacheable) تکرار کنید. حاصل کار تنها باید دو عبارت SQL باشند. یک مورد جهت insert و یک مورد هم select. در تراکنش سوم، از نتایج کش شده تراکنش دوم استفاده خواهد شد؛ به همین جهت دیگری کوئری سوم به بانک اطلاعاتی ارسال نخواهد شد.

اگر اعمال مورد (ج) فوق را فراموش کنید، سه کوئری را مشاهده خواهید کرد، اما کوئری سوم با کوئری دوم اندکی متفاوت خواهد بود و بهینه‌تر؛ چون به صورت هوشمند بر اساس جستجوی بر روی primary key تغییر کرده است (صرفنظر از اینکه قسمت where کوئری شما چیست).

نظرات خوانندگان

نویسنده: Ahmadxm1

تاریخ: ۱۷:۳۸:۳۰ ۱۳۹۰/۰۲/۱۱

سلام آقای نصیری

```
;()var q1 = session.QueryOver().Where(x => x.Id > 3).Cacheable().List
```

```
;()var q2 = session.QueryOver().Where(x => x.Id == 4).List
```

چرا با اینکه نتیجه کوئری دوم در کوئری اول وجود داره اما باز به دیتابیس مراجعه می کنه؟

نویسنده: وحید نصیری

تاریخ: ۲۰:۲۸:۵۱ ۱۳۹۰/۰۲/۱۱

سلام،

کلا سطح دوم کش در NH بر اساس 4 مکانیزم در طول یک سشن فکتوری عمل می کند:

- کش مربوط به موجودیت ها (entities cache) که بر اساس متد session.Get یا Load فعال می شود

و همچنین Collections cache (متدهای List و Enumerable)

- کش مربوط به کوئری ها (queries cache) با اعمال متد Cacheable به کوئری مورد نظر.

- timestamp cache که به معنای آخرین زمان نوشتن در یک جدول می باشد (و این timestamp فقط و فقط بر اساس وجود

تراکنش ها عمل می کند). به این ترتیب در زمان insert/update/delete به صورت خودکار کش موجود منقضی اعلام می شود تا

اطلاعات قدیمی به کاربر تحویل داده نشود و کش سطح دوم جهت کوئری های بعدی بازسازی خواهد شد.

و در مثال شما:

- در کوئری دوم هم باید متد Cacheable ذکر شود اگر نشود به صورت متداول با آن برخورد خواهد شد.

- زمانیکه از متد Cacheable استفاده می شود، حالت queries cache فعال می شود. چون در مثال شما دو کوئری مختلف داریم،

پس به کش مربوط به کوئری اول مراجعه نخواهد شد. این کوئری کش، با تغییر مقادیر پارامترهای یک کوئری هم مجدداً به روز

می شود. (این حالت برای کوئری هایی که با پارامترهای یکسان به طور متناوب فراخوانی می شوند، بسیار مناسب است)

- سطح دوم کش فقط پس از commit یک تراکنش معنا پیدا می کند. بنابراین اگر جهت آزمایش داخل یک تراکنش، پشت سر هم

کوئری ها را نوشته اید ... در این لحظه از سطح دوم کش بی بهره خواهید بود (فقط سطح اول کش فعال است) و کوئری های پس از

پایان تراکنش جاری، از نتیجه کش آن می توانند استفاده کنند.

- در مورد کش مربوط به موجودیت ها و تفاوت آن با کش کوئری ها در بالا صحبت شد (شما در یک جا کش کوئری را فعال کرده اید

در جای دیگر کش entities را طلب می کنید که نمی شود).

نویسنده: وحید نصیری

تاریخ: ۱۶:۲۸:۴۳ ۱۳۹۰/۰۲/۱۳

آقای پایروند این مجموعه رو تبدیل به فایل پی دی اف کردند برای کسانی که می خواهند ساده تر آن را مطالعه یا حتی پرینت بگیرند

https://rapidshare.com/files/460383624/NHibernate_VN_.pdf

در NHibernate چندین و چند روش، جهت تهیه کوئری‌ها وجود دارد که QueryOver یکی از آن‌ها است ([+](#)). QueryOver نسبت به LINQ to NH سازگاری بهتری با ساز و کار درونی NHibernate دارد؛ برای مثال امکان یکپارچگی آن با سطح دوم کش. هر چند ظاهر QueryOver با LINQ یکی است، اما در عمل متفاوتند و راه و روش خاص خودش را طلب می‌کند. برای مثال در LINQ to NH می‌تواند نوشت `x.Property.Contains` اما در QueryOver متدی به نام `contains` قابل استفاده نیست (هر چند در Intellisense ظاهر می‌شود اما عملاً تعریف نشده است و نباید آن را با LINQ اشتباه گرفت) و سعی در استفاده از آن‌ها به استثنای زیر ختم می‌شوند:

```
Unrecognised method call: System.String: Boolean StartsWith(System.String)
Unrecognised method call: System.String: Boolean Contains(System.String)
```

برای مثال کلاس زیر را در نظر بگیرید؛ کوئری‌های مطلب جاری بر این اساس تهیه خواهند شد:

```
using NHibernate.Validator.Constraints;

namespace NH3Test.MappingDefinitions.Domain
{
    public class Account
    {
        public virtual int Id { get; set; }

        [NotNullNotEmpty]
        [Length(Min = 3, Max = 120, Message = "طول نام باید بین 3 و 120 کاراکتر باشد")]
        public virtual string Name { get; set; }

        [NotNull]
        public virtual int Balance { set; get; }
    }
}
```

1 (یافتن رکوردهایی که در یک مجموعه‌ی مشخص قرار دارند. برای مثال `balance` آن‌ها مساوی 10 و 12 است:

```
var list = new[] { 12,10};
var resultList = session.QueryOver<Account>()
    .WhereRestrictionOn(p => p.Balance)
    .IsIn(list)
    .List();
```

```
SELECT
    this_.AccountId as AccountId0_0_,
    this_.Name as Name0_0_,
    this_.Balance as Balance0_0_
FROM
    Accounts this_
WHERE
    this_.Balance in (
        @p0 /* = 10 */, @p1 /* = 12 */
    )
```

2 (پیاده سازی همان متد `Contains` ذکر شده، در QueryOver:

```
var accountsContianX = session.QueryOver<Account>()
    .WhereRestrictionOn(x => x.Name)
```



```
.IsLike("X", NHibernate.Criterion.MatchMode.Anywhere)
.List();
```

```
SELECT
  this_.AccountId as AccountId0_0_,
  this_.Name as Name0_0_,
  this_.Balance as Balance0_0_
FROM
  Accounts this_
WHERE
  this_.Name like @p0 /* = %X% */
```

در اینجا بر اساس مقادیر مختلف MatchMode می‌توان متدهای StartsWith (MatchMode.Start) ، EndsWith (MatchMode.End) ، Equals (MatchMode.Exact) را نیز تهیه نمود.

انجام مثال دوم راه ساده‌تری نیز دارد. قسمت WhereRestrictionOn و IsLike به صورت یک سری extension متد ویژه در فضای نام NHibernate.Criterion تعریف شده‌اند. ابتدا این فضای نام را به کلاس جاری افزوده و سپس می‌توان نوشت :

```
using NHibernate.Criterion;
...
var accountsContainingX = session.QueryOver<Account>()
    .Where(x => x.Name.IsLike("%X%"))
    .List();
```

این فضای نام شامل چهار extension method به نام‌های IsBetween و IsLike ، IsInsensitiveLike ، IsIn است.

چگونه extension method سفارشی خود را تهیه کنیم؟

بهترین کار این است که به سورس NHibernate ، فایل‌های RestrictionsExtensions.cs و ExpressionProcessor.cs که تعاریف متد IsLike در آن‌ها وجود دارد مراجعه کرد. در اینجا می‌توان با نحوه‌ی تعریف و سپس ثبت آن در رجیستری extension methods مرتبط با QueryOver توسط متد عمومی RegisterCustomMethodCall آشنا شد. در ادامه سه کار را می‌توان انجام داد:

- متد مورد نظر را در کدهای خود (نه کدهای اصلی NH) اضافه کرده و سپس با فراخوانی RegisterCustomMethodCall آن را قابل استفاده نمائید.

-متد خود را به سورس اصلی NH اضافه کرده و کامپایل کنید.

-متد خود را به سورس اصلی NH اضافه کرده و کامپایل کنید (بهتر است همان روش نامگذاری بکار گرفته شده در فایل‌های ذکر شده رعایت شود). یک تست هم برای آن بنویسید (تست نویسی هم یک سری اصولی دارد ([+](#))). سپس یک patch از آن روی آن ساخته ([+](#)) و برای تیم NH ارسال نمائید (تا جایی که دقت کردم از کلیه ارسال‌هایی که آزمون واحد نداشته باشند، صرفنظر می‌شود).

مثال:

می‌خواهیم extension متد جدیدی به نام Year را به QueryOver اضافه کنیم. این متد را هم بر اساس توابع توکار بانک‌های اطلاعاتی، تهیه خواهیم نمود. لیست کامل این نوع متدهای بومی SQL را در فایل Dialect.cs سورس‌های NH می‌توان یافت (البته به صورت پیش فرض از متد extract برای جداسازی قسمت‌های مختلف تاریخ استفاده می‌کند. این متد در فایل‌های Dialect مربوط به بانک‌های اطلاعاتی مختلف، متفاوت است و برحسب بانک اطلاعاتی جاری به صورت خودکار تغییر خواهد کرد).

```
using System;
using System.Linq.Expressions;
using NHibernate;
using NHibernate.Criterion;
using NHibernate.Impl;
```

```

namespace NH3Test.ConsoleApplication
{
    public static class MyQueryOverExts
    {
        public static bool YearIs(this DateTime projection, int year)
        {
            throw new Exception("Not to be used directly - use inside QueryOver expression");
        }

        public static ICriterion ProcessAnsiYear(MethodCallExpression methodCallExpression)
        {
            string property =
                ExpressionProcessor.FindMemberExpression(methodCallExpression.Arguments[0]);
            object value = ExpressionProcessor.FindValue(methodCallExpression.Arguments[1]);
            return Restrictions.Eq(
                Projections.SqlFunction("year", NHibernateUtil.DateTime, Projections.Property(property)),
                value);
        }
    }

    public class QueryOverExtsRegistry
    {
        public static void RegistrMyQueryOverExts()
        {
            ExpressionProcessor.RegisterCustomMethodCall(
                () => MyQueryOverExts.YearIs(DateTime.Now, 0),
                MyQueryOverExts.ProcessAnsiYear);
        }
    }
}

```

اکنون برای استفاده خواهیم داشت:

```

QueryOverExtsRegistry.RegistrMyQueryOverExts(); //یکبار در ابتدای اجرای برنامه باید ثبت شود
...
var data = session.QueryOver<Account>()
    .Where(x => x.AddDate.YearIs(2010))
    .List();

```

برای مثال اگر بانک اطلاعاتی انتخابی از نوع SQLite باشد، خروجی SQL مرتبط به شکل زیر خواهد بود:

```

SELECT
    this_.AccountId as AccountId0_0_,
    this_.Name as Name0_0_,
    this_.Balance as Balance0_0_,
    this_.AddDate as AddDate0_0_
FROM
    Accounts this_
WHERE
    strftime("%Y", this_.AddDate) = @p0 /* =2010 */

```

هر چند ما تابع year را در متد ProcessAnsiYear ثبت کرده‌ایم اما بر اساس فایل SQLiteDialect.cs، تعاریف مرتبط و مخصوص این بانک اطلاعاتی (مانند متد strftime فوق) به صورت خودکار دریافت می‌گردد و کد ما مستقل از نوع بانک اطلاعاتی خواهد بود.

نکته جالب!

LINQ to NH هم قابل بسط است؛ کاری که در ORM های دیگر به این سادگی نیست. چند مثال در این زمینه:

چگونه تابع سفارشی SQL Server خود را به صورت یک extension method تعریف و استفاده کنیم: ([+](#)) ، یک نمونه دیگر: ([+](#)) و نمونه‌ای دیگر: ([+](#)).

عنوان:	QueryOver Extensions
نویسنده:	وحید نصیری
تاریخ:	۱۳۹۰/۰۲/۲۷ ۰۹:۱۹:۰۰
آدرس:	www.dotnettips.info
برچسب‌ها:	NHibernate

جهت تکمیل مطلب قبل ([+](#))، می‌توان به ازای تمام توابع SQL موجود و همچنین تمام حالت‌های اعمال محدودیت مانند مساوی، بزرگتر، کوچکتر و امثال آن، extension method نوشت. یا اینکه یک متد داشت که بتوان پارامترهای آن را تنظیم کرد. به همین جهت کتابخانه زیر را تهیه کرده‌ام که از آدرس زیر قابل دریافت است:

[QueryOverSqlFuncsExts](#)

نحوه استفاده:

ابتدا باید به NH معرفی شود (یکبار در ابتدای کار برنامه):

```
RegistrExt.RegistrMyQueryOverExts();
```

سپس استفاده از آن به سادگی زیر خواهد بود:

```
using QueryOverSqlFuncsExts;
...
var data = session.QueryOver<Account>()
    .Where(x => x.Name.Evaluate(new SqlFunc().CharIndex("a",
1).IsEqualTo(2)))
    .List();
```

مثال‌های بیشتر را در پوشه تست پروژه می‌توانید پیدا کنید.

نظرات خوانندگان

نویسنده: Ahmadxm1
تاریخ: ۰۸:۵۶:۵۷ ۱۳۹۰/۰۲/۲۸

سلام مهندس
من وقتی طبق چند پست قبلی شما از جوین استفاده میکنم این ارور رو میده
'Invalid column name 'Customer_id'
در صورتی که من اصلا این فیلد رو ندارم و فقط جدول والد به نام
Customers
است و در جدول
Order
نام کلید خارجی
Customer
است

نویسنده: وحید نصیری
تاریخ: ۰۹:۳۵:۵۷ ۱۳۹۰/۰۲/۲۸

برای پاسخ دقیق نیاز هست روش مپ کردن شما رو ببینم.
اگر دستی است احتمالا این نام تعریف شده (به ستون Id به صورت صریح انتساب داده شده): بررسی کنید
اگر از روش AutoMap استفاده می کنید حذسم این است که یک Convection جایی تعریف کردید که فیلد آی دی رو به این صورت
مپ کنه. (من خودم شبیه به این Convection را تهیه کردم).

نویسنده: Ahmadxm1
تاریخ: ۱۰:۳۲:۲۱ ۱۳۹۰/۰۲/۲۸

<https://rapidshare.com/files/3718254390/Test1.rar>
ممنون

نویسنده: Amir
تاریخ: ۱۱:۲۰:۰۵ ۱۳۹۰/۰۲/۲۸

See <http://tirania.org/blog/archive/2011/May-16.html>

نویسنده: وحید نصیری
تاریخ: ۱۱:۳۷:۴۰ ۱۳۹۰/۰۲/۲۸

در مورد شرکت جدید مونو در همان مطلب مخصوص آن به صورت کامنت لینک شما دیروز اضافه شد
+
مثال بالایی شما رو اصلاح کردم از این آدرس قابل دریافت است: [\(+\)](#)

نویسنده: Ahmadxm1
تاریخ: ۱۳:۴۲:۲۴ ۱۳۹۰/۰۲/۲۸

!!!Thanks

نویسنده: وحید نصیری
تاریخ: ۱۹:۳۳:۲۱ ۱۳۹۰/۰۳/۰۱

مبحث جاری در مورد "QueryOver Extensions" الان در trunk پروژه NHibernate قرار گرفته و از نگارش جدید آن در دسترس خواهد بود. البته syntax آن کمی تغییر کرده و مثلا شده `x.DateProp.DatePart()` و امثال آن، در فضای نام `NHibernate.Criterion`

Dialects در NHibernate کلاس‌هایی هستند جهت معرفی ویژگی‌های خاص بانک‌های اطلاعاتی مختلف؛ مثلاً SQL Server 2008 چه ویژگی‌های جدیدی دارد یا SQL Server CE 4.0 که جدیداً ارائه شده، امکان تعریف offset را در کوئری‌های خود میسر کرده (چیزی که قرار است در نگارش بعدی SQL Server اصلی(!) در دسترس باشد)، اکنون چگونه می‌توان این ویژگی را فعال کرد (باید Dialect آن به روز شود و ... همین). یک سری Dialect از پیش تعریف شده هم برای اکثر بانک‌های اطلاعاتی در NHibernate وجود دارد. ممکن است این Dialects پیش فرض الزاماً خواسته شما را برآورده نکنند یا مو به مو مستندات بانک اطلاعاتی مرتبط را پیاده سازی نکرده باشند و سؤال این است که اکنون چه باید کرد؟ آیا باید حتماً سورس‌ها را دستکاری و بعد کامپایل کرد؟ به این صورت هر بار با ارائه یک نگارش جدید NHibernate به مشکل برخورد چون باید کل عملیات تکرار شود.

خبر خوب اینکه می‌توان از همین Dialects موجود ارث بری کرد، سپس مواردی را که نیاز داریم override کرده یا به کلاس مشتق شده افزود. اکنون می‌توان از این Dialect سفارشی به جای Dialect اصلی استفاده کرد. در ادامه با یک نمونه آشنا خواهیم شد.

فرض کنید Dialect انتخابی مرتبط است با SQL Server CE استاندارد. کوئری ساده زیر را می‌نویسیم، به ظاهر باید کار کند:

```
var list = session.Query<SomeClass>().Where(x=>x.Date.Year==2011).ToList();
```

اما کار نمی‌کند! علت این است که تمام Dialects در NHibernate از یک Dialect پایه مشتق شده‌اند. در این Dialect پایه، تعریف تابع استخراج year از یک تاریخ به نحو زیر است:

```
extract(year, ?1)
```

اما در SQL CE این تابع باید به صورت زیر تغییر کند تا کار کند:

```
datepart(year, ?1)
```

و ... این Override انجام نشده (تا نگارش فعلی آن). مهم نیست! خودمان انجام خواهیم داد! به صورت زیر:

```
using NHibernate;
using NHibernate.Dialect;
using NHibernate.Dialect.Function;

namespace Test1
{
    public class CustomSqlCeDialect : MsSqlCeDialect
    {
        public CustomSqlCeDialect()
        {
            RegisterFunction("year", new SQLFunctionTemplate(NHibernateUtil.Int32, "datepart(year,
?1)"));
        }
    }
}
```

خوب تا اینجا ما یک Dialect جدید را با ارث بری از MsSqlCeDialect اصلی تولید کرده‌ایم. مرحله بعد نوبت به معرفی آن به NHibernate است. اینکار توسط Fluent NHibernate به سادگی زیر است:

```
var dbType = MsSqlCeConfiguration.Standard
...
```

```
.Dialect<CustomSqlCeDialect>();
```

پس از آن کوئری LINQ ابتدای بحث بدون مشکل اجرا خواهد شد چون اکنون می‌داند که بجای `extract year` ، باید از تابع `datepart` استفاده کند.

مرحله بعد هم می‌تواند تهیه یک `patch` و ارسال به گروه اصلی برای به روز رسانی پروژه NH باشد.

یکی از قابلیت‌های جالب NHibernate امکان تعریف فیلدها به صورت پویا هستند. به این معنا که زیرساخت طراحی یک برنامه "فرم ساز" هم اکنون در اختیار شما است! سیستمی که امکان افزودن فیلدهای سفارشی را دارا است که توسط برنامه نویس در زمان طراحی اولیه آن ایجاد نشده‌اند. در ادامه نحوه‌ی تعریف و استفاده از این قابلیت را توسط Fluent NHibernate بررسی خواهیم کرد.

در اینجا کلاسی که قرار است توانایی افزودن فیلدهای سفارشی را داشته باشد به صورت زیر تعریف می‌شود:

```
using System.Collections;

namespace TestModel
{
    public class DynamicEntity
    {
        public virtual int Id { get; set; }
        public virtual IDictionary Attributes { set; get; }
    }
}
```

Attributes در عمل همان فیلدهای سفارشی مورد نظر خواهند بود. جهت معرفی صحیح این قابلیت نیاز است تا نگاهی به آن را از نوع dynamic component تعریف کنیم:

```
using FluentNHibernate.Automapping;
using FluentNHibernate.Automapping.Alterations;

namespace TestModel
{
    public class DynamicEntityMapping : IAutoMappingOverride<DynamicEntity>
    {
        public void Override(AutoMapping<DynamicEntity> mapping)
        {
            mapping.Table("tblDynamicEntity");
            mapping.Id(x => x.Id);
            mapping.IgnoreProperty(x => x.Attributes);
            mapping.DynamicComponent(x => x.Attributes,
                c =>
                {
                    c.Map(x => (string)x["field1"]);
                    c.Map(x => (string)x["field2"]).Length(300);
                    c.Map(x => (int)x["field3"]);
                    c.Map(x => (double)x["field4"]);
                });
        }
    }
}
```

و مهم‌ترین نکته‌ی این بحث هم همین نگاهی است.

ابتدا از IgnoreProperty جهت ندید گرفتن Attributes استفاده کردیم. زیرا در غیر این صورت در حالت Auto mapping، یک رابطه چند به یک به علت وجود IDictionary خواهد شد که نیازی به آن نیست (یافتن این نکته نصف روز کار

برد! چون مرتباً خطای: An association from the table DynamicEntity refers to an unmapped class

System.Collections.IDictionary ظاهر می‌شد و مشخص نبود که مشکل از کجاست).

سپس Attributes به عنوان یک DynamicComponent معرفی شده است. در اینجا چهار فیلد سفارشی را اضافه کرده‌ایم. به این معنا که اگر نیاز باشد تا فیلد سفارشی دیگری به سیستم اضافه شود باید یکبار Session factory ساخته شود و SchemaUpdate

فراخوانی گردد و خوشبختانه با وجود Fluent NHibernate، تمام این تغییرات در کدهای برنامه قابل انجام است (بدون نیاز به سر و کار داشتن با فایل‌های XML نگاشت‌ها و ویرایش دستی آن‌ها). از تغییر نام جدول که برای مثال در اینجا `tblDynamicEntity` در نظر گرفته شده تا افزودن فیلدهای دیگر در قسمت `DynamicComponent` فوق. همچنین با توجه به اینکه این نوع تغییرات (ساخت دوبار سشن فکتوری) مواردی نیستند که قرار باشد هر ساعت انجام شوند، بنابراین سربار آنچنانی را به سیستم تحمیل نمی‌کنند.

```
drop table tblDynamicEntity

create table tblDynamicEntity (
    Id INT IDENTITY NOT NULL,
    field1 NVARCHAR(255) null,
    field2 NVARCHAR(300) null,
    field3 INT null,
    field4 FLOAT null,
    primary key (Id)
)
```

اگر با `SchemaExport`، اسکرپت خروجی معادل با نگاشت فوق را تهیه کنیم به جدول فوق خواهیم رسید. نوع و طول این فیلدهای سفارشی بر اساس نوعی که برای اشیاء دیکشنری مشخص می‌کنید، تعیین خواهند شد.

چند مثال جهت کار با این فیلدهای سفارشی یا پویا :

نحوه‌ی افزودن رکوردهای جدید بر اساس خاصیت‌های سفارشی:

```
//insert
object savedId = 0;
using (var session = sessionFactory.OpenSession())
{
    using (var tx = session.BeginTransaction())
    {
        var obj = new DynamicEntity();
        obj.Attributes = new Hashtable();
        obj.Attributes["field1"] = "test1";
        obj.Attributes["field2"] = "test2";
        obj.Attributes["field3"] = 1;
        obj.Attributes["field4"] = 1.1;

        savedId = session.Save(obj);
        tx.Commit();
    }
}
```

با خروجی

```
INSERT
INTO
    tblDynamicEntity
    (field1, field2, field3, field4)
VALUES
    (@p0, @p1, @p2, @p3);
    @p0 = 'test1' [Type: String (0)], @p1 = 'test2' [Type: String (0)], @p2 = 1
    [Type: Int32 (0)], @p3 = 1.1 [Type: Double (0)]
```

نحوه‌ی کوئری گرفتن از این اطلاعات (فعلا پایدارترین روشی را که برای آن یافته‌ام استفاده از HQL می‌باشد ...):

```
//query
using (var session = sessionFactory.OpenSession())
{
    using (var tx = session.BeginTransaction())
    {
```

```

//using HQL
var list = session
    .CreateQuery("from DynamicEntity d where d.Attributes.field1=:p0")
    .SetString("p0", "test1")
    .List<DynamicEntity>();

if (list != null && list.Any())
{
    Console.WriteLine(list[0].Attributes["field2"]);
}
tx.Commit();
}
}

```

با خروجی:

```

select
    dynamicent0_.Id as Id1_,
    dynamicent0_.field1 as field2_1_,
    dynamicent0_.field2 as field3_1_,
    dynamicent0_.field3 as field4_1_,
    dynamicent0_.field4 as field5_1_
from
    tblDynamicEntity dynamicent0_
where
    dynamicent0_.field1=@p0;
@p0 = 'test1' [Type: String (0)]

```

استفاده از HQL هم یک مزیت مهم دارد: چون به صورت رشته قابل تعریف است، به سادگی می‌توان آنرا داخل دیتابیس ذخیره کرد. برای مثال یک سیستم گزارش ساز پویا هم در این کنار طراحی کرد

نحوه‌ی به روز رسانی و حذف اطلاعات بر اساس فیلدهای پویا:

```

//update
using (var session = sessionFactory.OpenSession())
{
    using (var tx = session.BeginTransaction())
    {
        var entity = session.Get<DynamicEntity>(savedId);
        if (entity != null)
        {
            entity.Attributes["field2"] = "new-val";
            tx.Commit(); // Persist modification
        }
    }
}

//delete
using (var session = sessionFactory.OpenSession())
{
    using (var tx = session.BeginTransaction())
    {
        var entity = session.Get<DynamicEntity>(savedId);
        if (entity != null)
        {
            session.Delete(entity);
            tx.Commit();
        }
    }
}

```

نظرات خوانندگان

نویسنده: A. Karimi
تاریخ: ۱۳:۵۸:۱۰ ۱۳۹۰/۰۴/۰۳

این طور که به نظر می‌رسد برای مثال در صورت اضافه نمودن یک فیلد جدید به Attributes در این مثال، باید بانک اطلاعات مجدداً ایجاد شود. آیا این طور است؟
به علت خروجی دستور Insert و Select به این نتیجه رسیدم.

نویسنده: وحید نصیری
تاریخ: ۱۷:۱۱:۱۱ ۱۳۹۰/۰۴/۰۳

بله. اما مشکلی نیست چون NH مکانیزم به روز رسانی خودکار دیتابیس را هم دارد؛ به کمک کلاس SchemaUpdate واقع شده در فضای نام NHibernate.Tool.hbm2ddl.

نویسنده: وحید نصیری
تاریخ: ۱۷:۱۶:۲۸ ۱۳۹۰/۰۴/۰۳

در مورد آشنایی کلی با طرز کارش هم می‌تونید به این پروژه مراجعه کنید: [\[+\]](#)

نویسنده: afsharm
تاریخ: ۰۹:۴۵:۲۴ ۱۳۹۰/۰۴/۰۶

راه حل فیلدهای دینامیک راه جالبی است. اما من به دنبال راهی برای دینامیک کردن entity هستم. یعنی بشه چند تا entity که از قبل وجود ندارند را در زمان اجرا به برنامه اضافه کرد.

نویسنده: وحید نصیری
تاریخ: ۱۰:۴۳:۴۳ ۱۳۹۰/۰۴/۰۶

بحث فوق را می‌شود با امکانات دینامیک سی شارپ 4 هم توسعه داد یعنی استفاده از اشیاء دینامیک بجای استفاده از HashTable. دو مطلب در این مورد موجود است:

[Support dynamic fields with NHibernate and .NET 4.0](#)

[Duck Typing with NHibernate Reloaded](#)

اگر این موارد مد نظر نبودند باید به سراغ مطالبی مانند این [\(+\)](#) بروید. می‌شود کلاس را در زمان اجرا اضافه کرد، در حافظه کامپایل کرد (بجای کامپایل روی سخت دیسک) و سپس استفاده کرد.

در مورد طراحی یک برنامه "فرم ساز" در [مطلب قبلی](#) بحث شد ... حدودا سه سال قبل اینکار را برای شرکتی انجام دادم. یک برنامه درخواست خدمات نوشته شده با ASP.NET که مدیران برنامه می‌توانستند برای آن فرم طراحی کنند؛ فرم درخواست پرینت، درخواست نصب نرم افزار، درخواست وام، درخواست پیک، درخواست آژانس و ... فرم‌هایی که تمامی نداشتند! آن زمان برای حل این مساله از فیلدهای XML استفاده کردم.

فیلدهای XML قابلیت نه چندان جدیدی هستند که از SQL Server 2005 به بعد اضافه شده‌اند. مهم‌ترین مزیت آن‌ها هم امکان ذخیره سازی اطلاعات هر نوع شیء‌ای به عنوان یک فیلد XML است. یعنی همان زیرساختی که برای ایجاد یک برنامه فرم ساز نیاز است. ذخیره سازی آن هم آداب خاصی را طلب نمی‌کند. به ازای هر فیلد مورد نظر کاربر، یک نود جدید به صورت رشته معمولی باید اضافه شود و نهایتا رشته تولیدی باید ذخیره گردد. از دید ما یک رشته است، از دید SQL Server یک نوع XML واقعی؛ به همراه این مزیت مهم که به سادگی می‌توان با T-SQL/XQuery/XPath از جزئیات اطلاعات این نوع فیلدها کوئری گرفت و سرعت کار هم واقعا بالا است؛ به علاوه بر خلاف مطلب قبلی در مورد dynamic components، اینبار نیازی نیست تا به ازای هر یک فیلد درخواستی کاربر، واقعا یک فیلد جدید را به جدول خاصی اضافه کرد. داخل این فیلد XML هر نوع ساختار دلخواهی را می‌توان ذخیره کرد. به عبارتی به کمک فیلدهایی از نوع XML می‌توان داخل یک سیستم بانک اطلاعاتی رابطه‌ای، schema-less کار کرد (un-typed XML) و همچنین از این اطلاعات ویژه، کوئری‌های پیچیده هم گرفت.

تا جایی که اطلاع دارم، چند شرکت دیگر هم در ایران دقیقا از همین ایده فیلدهای XML برای ساخت برنامه فرم ساز استفاده کرده‌اند ... البته مطلب جدیدی هم نیست؛ برنامه‌های فرم ساز اوراکل و IBM هم سال‌ها است که از XML برای همین منظور استفاده می‌کنند. مایکروسافت هم به همین دلیل (شاید بتوان گفت مهم‌ترین دلیل وجودی فیلدهای XML در SQL Server)، پشتیبانی توکاری از XML به عمل آورده است.

یا روش دیگری را که برای طراحی سیستم‌های فرم ساز پیشنهاد می‌کنند استفاده از بانک‌های اطلاعاتی مبتنی بر key-value مانند [Redis](#) یا [RavenDb](#) است؛ یا استفاده از بانک‌های اطلاعاتی schema-less واقعی مانند [CouchDb](#).

خوب ... اکنون سؤال این است که NHibernate برای کار با فیلدهای XML چه تمهیداتی را در نظر گرفته است؟ برای این منظور خاصیتی را که قرار است به یک فیلد از نوع XML نگاشت شود، با نوع XDocument مشخص خواهیم ساخت:

```
using System.Xml.Linq;

namespace TestModel
{
    public class DynamicTable
    {
        public virtual int Id { get; set; }
        public virtual XDocument Document { get; set; }
    }
}
```

سپس باید جهت معرفی این نوع ویژه، به صورت صریح از XDocType استفاده کرد؛ یعنی نکته‌ی اصلی، استفاده از CustomType مرتبط است:

```
using FluentNHibernate.Automapping;
using FluentNHibernate.Automapping.Alterations;
using NHibernate.Type;

namespace TestModel
{
    public class DynamicTableMapping : IAutoMappingOverride<DynamicTable>
    {
        public void Override(AutoMapping<DynamicTable> mapping)
        {
        }
    }
}
```

```

        mapping.Id(x => x.Id);
        mapping.Map(x => x.Document).CustomType<XDocType>();
    }
}

```

البته لازم به ذکر است که دو نوع `NHibernate.Type.XmlDocType` و `NHibernate.Type.XDocType` برای کار با فیلدهای XML در NHibernate وجود دارند. `XDocType` برای کار با نوع `System.Xml.Linq.XDocument` طراحی شده است و `XmlDocType` مخصوص نگاشت نوع `System.Xml.XmlDocument` است.

اکنون اگر به کمک کلاس `SchemaExport`، اسکریپت تولید جدول متناظر با اطلاعات فوق را ایجاد کنیم به حاصل زیر خواهیم رسید:

```

if exists (select * from dbo.sysobjects
           where id = object_id(N'[DynamicTable]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
    drop table [DynamicTable]

create table [DynamicTable] (
    Id INT IDENTITY NOT NULL,
    Document XML null,
    primary key (Id)
)

```

یک سری اعمال متداول ذخیره سازی اطلاعات و تهیه کوئری نیز در ادامه ذکر شده‌اند:

```

//insert
object savedId = 0;
using (var session = sessionFactory.OpenSession())
{
    using (var tx = session.BeginTransaction())
    {
        var obj = new DynamicTable
        {
            Document = System.Xml.Linq.XDocument.Parse(
                @"<Doc><Node1>Text1</Node1><Node2>Text2</Node2></Doc>"
            );
        };
        savedId = session.Save(obj);
        tx.Commit();
    }
}

//simple query
using (var session = sessionFactory.OpenSession())
{
    using (var tx = session.BeginTransaction())
    {
        var entity = session.Get<DynamicTable>(savedId);
        if (entity != null)
        {
            Console.WriteLine(entity.Document.Root.ToString());
        }
        tx.Commit();
    }
}

//advanced query
using (var session = sessionFactory.OpenSession())
{
    using (var tx = session.BeginTransaction())
    {
        var list = session.CreateSQLQuery("select [Document].value('(/Doc/Node1)[1]', 'nvarchar(255)')
from [DynamicTable] where id=:p0")
            .SetParameter("p0", savedId)
            .List();

        if (list != null)

```

```
    {  
        Console.WriteLine(list[0]);  
    }  
    tx.Commit();  
}
```

و در پایان بدیهی است که جهت کار با امکانات پیشرفته‌تر موجود در SQL Server در مورد فیلدهای XML (برای نمونه: [+](#) و [+](#)) باید مثلاً رویه ذخیره شده تهیه کرد (یا مستقیماً از متد CreateSQLQuery همانند مثال فوق کمک گرفت) و آن‌را در NHibernate مورد استفاده قرار داد. البته به این صورت کار شما محدود به SQL Server خواهد شد و باید در نظر داشت که در کل تعداد کمی بانک اطلاعاتی وجود دارند که نوع‌های XML را به صورت توکار پشتیبانی می‌کنند.

نظرات خوانندگان

نویسنده: Afshar Mohebbi
تاریخ: ۱۳۹۰/۰۳/۳۰ ۲۱:۲۳:۲۷

جالب بود. خصوصاً این که در مورد «فرم ساز»ها صحبت می‌کرد.

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۰/۰۳/۳۱ ۱۰:۴۰:۴۱

سلام.

من nHibernate بلد نیستم اما قسمتی که درباره فرم سازها صحبت کردید جالب بود.
این فرم سازی که شما ازش صحبت کردید User Mode بود حالا اگر فرم سازی بخواهیم توسعه بدیم که Developer Mode باشه نظرتون چیه؟
مثلا از روی یک Table فرم خام رو صورت Html در بیاورد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۳/۳۱ ۱۰:۴۵:۱۳

به ASP.NET MVC کوچ کنید. از روی مدل شما فرم‌های insert/delete/update به همراه اعتبار سنجی و غیره رو همه رو یکجا با ابزارهای توکاری که داره تولید می‌کنه. حتی مقادیر وارد شده توسط کاربر رو هم به صورت خودکار به فیلدهای مدل انتساب می‌ده (model binding).

عنوان: پیاده سازی الگوی واحد کار در NHibernate

نویسنده: وحید نصیری

تاریخ: ۱۳۹۰/۰۳/۳۱ ۲۰:۳۵:۰۰

آدرس: www.dotnettips.info

برچسب‌ها: NHibernate

جهت تکمیل بحث [الگوی مخزن در NH](#) ، می‌توان به مطالب ذیل نیز مراجعه کرد:

[Unit of work and Repository Pattern with nHibernate and linq - I](#)

[Unit of work and Repository Pattern with nHibernate and linq - II](#)

همچنین پروژه مرتبط با مطالب فوق هم [از این](#) آدرس قابل دریافت است.

نگاشت خودکار مجموعه‌ها در Fluent NHibernate ساده است و نیاز به تنظیم خاصی ندارد. برای مثال IList به صورت خودکار به Bag ترجمه می‌شود و الی آخر.

البته شاید سؤال بپرسید که این Bag از کجا آمده؟ کلاً 6 نوع مجموعه در NHibernate پشتیبانی می‌شوند [که شامل](#) Array، List، Set، Bag، Primitive-Array و Map هستند؛ این اسامی هم جهت حفظ سازگاری با جاوا تغییر نکرده‌اند و گر نه معادل‌های آن‌ها در دات نت به این شرح هستند:

```
Bag=IList
Set=Iesi.Collections.ISet
List=IList
Map=IDictionary
```

البته در دات نت 4، [ISet](#) هم به صورت توکار اضافه شده، اما NHibernate از مدت‌ها قبل آن‌را از کتابخانه‌ی Iesi.Collections به عاریت گرفته است. مهم‌ترین تفاوت‌های این مجموعه‌ها هم در پذیرفتن یا عدم پذیرش اعضای تکراری است. Set و Map اعضای تکراری نمی‌پذیرند.

در ادامه می‌خواهیم طرز کار با Map یا همان IDictionary دات نت را بررسی کنیم:

الف) حالتی که نوع کلید و مقدار (در یک عضو Dictionary تعریف شده)، Entity نیستند

```
using System.Collections.Generic;

namespace Test1.Model12
{
    public class User
    {
        public virtual int Id { get; set; }
        public virtual string Name { get; set; }
        public virtual IDictionary<string, string> Preferences { get; set; }
    }
}
```

نحوه تعریف نگاشت که مبتنی است بر مشخص سازی تعاریف کلید و مقدار آن جهت تشکیل یک Map یا همان Dictionary :

```
using FluentNHibernate.Automapping;
using FluentNHibernate.Automapping.Alterations;

namespace Test1.Model12
{
    public class UserMapping : IAutoMappingOverride<User>
    {
        public void Override(AutoMapping<User> mapping)
        {
            mapping.Id(x => x.Id);
            mapping.HasMany(x => x.Preferences)
                .AsMap<string>("FieldKey")
                .Element("FieldValue", x => x.Type<string>().Length(500));
        }
    }
}
```

خروجی SQL متناظر:

```
create table "User" (
    Id INT IDENTITY NOT NULL,
    Name NVARCHAR(255) null,
    primary key (Id)
)

create table Preferences (
    User_id INT not null,
    FieldValue NVARCHAR(500) null,
    FieldKey NVARCHAR(255) not null,
    primary key (User_id, FieldKey)
)

alter table Preferences
    add constraint FKD6CB18523B1FD789
    foreign key (User_id)
    references "User"
```

ب) حالتی که مقدار، Entity است

```
using System.Collections.Generic;

namespace Test1.Model13
{
    public class User
    {
        public virtual int Id { get; set; }
        public virtual string Name { get; set; }
        public virtual IDictionary<string, Property> Properties { get; set; }
    }

    public class Property
    {
        public virtual int Id { get; set; }
        public virtual string Name { get; set; }
        public virtual string Value { get; set; }
        public virtual User User { get; set; }
    }
}
```

نحوه تعریف نگاشت:

```
using FluentNHibernate.Automapping;
using FluentNHibernate.Automapping.Alterations;

namespace Test1.Model13
{
    public class UserMapping : IAutoMappingOverride<User>
    {
        public void Override(AutoMapping<User> mapping)
        {
            mapping.Id(x => x.Id);
            mapping.HasMany<Property>(x => x.Properties)
                .AsMap<string>("FieldKey")
                .Component(x => x.Map(c => c.Id));
        }
    }
}
```

خروجی SQL متناظر:

```
create table "Property" (
    Id INT IDENTITY NOT NULL,
    Name NVARCHAR(255) null,
    Value NVARCHAR(255) null,
    User_id INT null,
```

```

    primary key (Id)
)
create table "User" (
    Id INT IDENTITY NOT NULL,
    Name NVARCHAR(255) null,
    primary key (Id)
)
create table Properties (
    User_id INT not null,
    Id INT null,
    FieldKey NVARCHAR(255) not null,
    primary key (User_id, FieldKey)
)
alter table "Property"
    add constraint FKF9F4D85A3B1FD7A2
    foreign key (User_id)
    references "User"
alter table Properties
    add constraint FK63646D853B1FD7A2
    foreign key (User_id)
    references "User"

```

ج) حالتی که کلید، Entity است

```

using System;
using System.Collections.Generic;

namespace Test1.Model14
{
    public class FormData
    {
        public virtual int Id { get; set; }
        public virtual DateTime? DateTime { get; set; }
        public virtual IDictionary<FormField, string> FormPropertyValues { get; set; }
    }

    public class FormField
    {
        public virtual int Id { get; set; }
        public virtual string Name { get; set; }
    }
}

```

نحوه تعریف نگاشت:

```

using FluentNHibernate.Automapping;
using FluentNHibernate.Automapping.Alterations;

namespace Test1.Model14
{
    public class FormDataMapping : IAutoMappingOverride<FormData>
    {
        public void Override(AutoMapping<FormData> mapping)
        {
            mapping.Id(x => x.Id);
            mapping.HasMany<FormField>(x => x.FormPropertyValues)
                .AsEntityMap("FieldId")
                .Element("FieldValue", x => x.Type<string>().Length(500))
                .Cascade.All();
        }
    }
}

```

خروجی SQL متناظر:

```

create table "FormData" (
  Id INT IDENTITY NOT NULL,
  DateTime DATETIME null,
  primary key (Id)
)

create table FormPropertyValues (
  FormData_id INT not null,
  FieldValue NVARCHAR(500) null,
  FieldId INT not null,
  primary key (FormData_id, FieldId)
)

create table "FormField" (
  Id INT IDENTITY NOT NULL,
  Name NVARCHAR(255) null,
  primary key (Id)
)

alter table FormPropertyValues
  add constraint FKB807B9C090849E
  foreign key (FormData_id)
  references "FormData"

alter table FormPropertyValues
  add constraint FKB807B97165898A
  foreign key (FieldId)
  references "FormField"

```

یک مثال عملی:

امکانات فوق جهت طراحی قسمت ثبت اطلاعات یک برنامه «فرم ساز» مبتنی بر Key-Value بسیار مناسب هستند؛ برای مثال: برنامه‌ای را در نظر بگیرید که می‌تواند تعدادی خدمات داشته باشد که توسط مدیر برنامه قابل اضافه شدن است؛ برای نمونه خدمات درخواست نصب نرم افزار، خدمات درخواست تعویض کارت پرسنلی، خدمات درخواست مساعده، خدمات ... :

```

public class Service
{
  public virtual int Id { get; set; }
  public virtual string Name { get; set; }
  public virtual IList<ServiceFormField> Fields { get; set; }
  public virtual IList<ServiceFormData> Forms { get; set; }
}

```

برای هر خدمات باید بتوان یک فرم طراحی کرد. هر فرم هم از یک سری فیلد خاص آن خدمات تشکیل شده است. برای مثال:

```

public class ServiceFormField
{
  public virtual int Id { get; set; }
  public virtual string Name { get; set; }
  public virtual bool IsRequired { get; set; }
  public virtual Service Service { get; set; }
}

```

در اینجا نیازی نیست به ازای هر فیلد جدید واقعا یک فیلد متناظر به دیتابیس اضافه شود و ساختار آن تغییر کند (برخلاف حالت dynamic components که پیشتر در مورد آن بحث شد).

اکنون با داشتن یک خدمات و فیلدهای پویای آن که توسط مدیربرنامه تعریف شده‌اند، می‌توان اطلاعات وارد کرد. مهم‌ترین نکته‌ی آن هم IDictionary تعریف شده است که حاوی لیستی از فیلدها به همراه مقادیر وارد شده توسط کاربر خواهد بود:

```

public class ServiceFormData
{
  public virtual int Id { get; set; }
  public virtual IDictionary<ServiceFormField, string> FormPropertyValues { get; set; }
  public virtual DateTime? DateTime { get; set; }
}

```

```
public virtual Service Service { get; set; }  
}
```

در مورد نحوه نگاشت آن هم در حالت «ج» فوق توضیح داده شد.

عنوان: به روز رسانی اسمبلی‌های دارای امضای دیجیتال در VS.NET

نویسنده: وحید نصیری

تاریخ: ۱۳۹۰/۰۵/۲۱ ۱۰:۲۳:۰۰

آدرس: www.dotnettips.info

برچسب‌ها: NHibernate

زمانیکه در VS.NET یک اسمبلی دارای امضای دیجیتال را اضافه می‌کنیم، در فایل پروژه برنامه مدخلی شبیه به عبارت زیر اضافه می‌شود:

```
<Reference Include="NHibernate, Version=2.1.0.4000, Culture=neutral, PublicKeyToken=aa95f207798dfdb4, processorArchitecture=MSIL">
```

همانطور که ملاحظه می‌کنید، شماره نگارش فایل، PublicKeyToken و غیره دقیقاً ذکر می‌شوند. حال اگر همین پروژه را بخواهید به نگارش 3.2 ارتقاء دهید، احتمالاً به روش متداول کپی اسمبلی جدید در پوشه bin برنامه اکتفاء خواهید کرد. برنامه هم پس از یک Rebuild، به خوبی کامپایل می‌شود و مشکلی ندارد. اما به محض اجرا و دیباگ در VS.NET، با خطای زیر مواجه خواهید شد:

```
Could not load file or assembly 'NHibernate, Version=2.0.0.4000, Culture=neutral, PublicKeyToken=aa95f207798dfdb4'
```

```
or one of its dependencies. The located assembly's manifest definition does not match the assembly reference.
```

```
(Exception from HRESULT: 0x80131040)
```

بله! هنوز به دنبال نگارش 2 می‌گردد و به نظر، نگارش 3.2 جدید را ندید گرفته است. مشکل هم به همان مدخل دقیق موجود در فایل پروژه برنامه، مرتبط است. این مدخل صرفاً با copy/paste فایل‌های جدید در پوشه bin برنامه یا rebuild پروژه، «به روز نمی‌شود»!

یا باید دستی این فایل csproj یا vbproj را ویرایش کنید، یا یکبار باید از داخل VS.NET این ارجاعات را حذف کرده و مجدداً بر اساس فایل‌های جدید ایجاد کنید تا فایل پروژه برنامه بر این اساس به روز شود.

این مشکلی هست که حداقل با تمام مثال‌های NHibernate دریافتی از این سایت خواهید داشت.

روش دیگر حل این مشکل، مراجعه به خواص اسمبلی اضافه شده در لیست ارجاعات پروژه در VS.NET و خاموش کردن گزینه‌ی " [Specific Version](#) " آن است.

به صورت خلاصه حین به روز رسانی اسمبلی‌های دارای امضای دیجیتال:

یا باید ارجاعات دارای امضای دیجیتال را حذف و بار دیگر اضافه کنید.

یا باید فایل پروژه برنامه را با یک ویرایشگر متنی ساده باز کرده و شماره نگارش‌ها را اصلاح کنید. (ساده‌ترین روش ممکن) یا خاموش کردن بررسی Specific Version را هم آزمایش کنید.

عنوان: آدرس جدید مخزن کد NHibernate

نویسنده: وحید نصیری

تاریخ: ۱۳۹۰/۰۶/۰۷ ۰۰:۴۲:۰۰

آدرس: www.dotnettips.info

برچسب‌ها: NHibernate

تیم NHibernate از سیستم SVN سورس فورج، به سورس کنترل Git در سایت GitHub نقل مکان کرده است: [[^](#)]

همچنین Issue tracker آن‌ها هم مدتی است که به آدرس جدیدی منتقل شده است: [[^](#)]

و ... اگر علاقمند باشید که از آخرین تغییرات این کتابخانه آگاه شوید، زیاد به دنبال وبلاگ یا سایت خاصی نگردید. روش متداول کار با کتابخانه‌های سورس باز، دنبال کردن change log ارسالی آن‌ها به سیستم‌های سورس کنترل است (همان متنی که حین commit ارسال می‌کنند). برای مثال جهت آگاه شدن از آخرین تغییرات NHibernate مشترک این فید شوید: [[^](#)]

نظرات خوانندگان

نویسنده: Ahmad_Akbari2008
تاریخ: ۱۶:۱۱:۵۰ ۱۳۹۰/۰۶/۰۷

همه دارن به Git کوچ میکنن. مثل اینکه Git داره از SVN رو کنار میزنه!

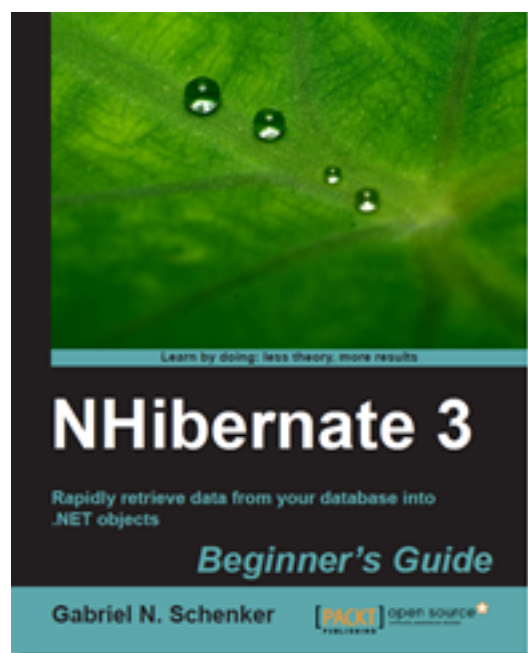
نویسنده: وحید نصیری
تاریخ: ۱۸:۵۹:۳۰ ۱۳۹۰/۰۶/۰۷

Git به علت ساختار توزیع شده آن از SVN پیشرفته تر است. البته مرکوریال هم در همین رده قرار می گیرد. تا جایی که می دونم در گوگل از mercurial زیاد استفاده می شود [\[+\]](#). لینوکسی ها هم از Git، چون خود Linus Torvalds اون رو درست کرده (یا حداقل شروعش توسط اون بوده). ولی ... در کل مهم این است که از یک سیستم سورس کنترل استفاده شود. من با SVN راحتتم!

نویسنده: Farhad Yazdan-Panah
تاریخ: ۰۰:۳۸:۱۳ ۱۳۹۰/۰۶/۰۹

به نظر من در مورد Git (و همچنین خانواده لینوکس) یه چیز فراموش شده، "سادگی".

کتاب جدیدی در مورد NHibernate 3 ماه قبل توسط انتشارات Packt منتشر گردید، که توسط آقای [دکتر Schenker](#) نوشته شده و از همه مهم‌تر توسط تیم NHibernate بازخوانی و رفع اشکال شده است.



قسمتی از این کتاب مقدماتی را [اینجا](#) می‌توانید مطالعه کنید.

و... یکی دو روزی است که فایل PDF کامل آن در اکثر سایت‌ها قابل دریافت است.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۶/۱۸ ۲۱:۰۱:۲۶

کدهای کتاب رو از سایت اصلی میشه دریافت کرد: [^]

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۶/۱۹ ۲۱:۳۰:۲۷

این کتاب تاکید زیادی روی Fluent NHibernate دارد و تقریباً تمام مثال‌های آن با Fluent NHibernate پیاده سازی شده. همچنین ConfOrm رو هم توضیح داده. فوق العاده کتاب پرمحتوایی است! مخصوصاً فصل unit test و فصل خطاهای متداول با NH آن عالی است.

نویسنده: Mehdi Payervand
تاریخ: ۱۳۹۰/۰۶/۲۰ ۰۸:۱۰:۵۱

مطمئنم بعد از کتاب آقای مهندس راد این کتاب رو میخونم، خیلی وقته منتظرش هستم

نویسنده: Ahmad Alavy
تاریخ: ۱۳۹۰/۰۶/۲۰ ۲۱:۲۱:۳۱

Hello
?Fluent NHibernate or ConfOrm

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۶/۲۰ ۲۱:۵۳:۵۷

تمرکز اصلی آن روی FHN است. خلاصه‌ای از ConfOrm رو هم گفته. البته می‌دونید که NH 3.2 قسمتی از ConfOrm رو به ارث برده، بنابراین یادگیری آن مفید است. اشاره‌ای هم به سیستم تعریف نگاشت XML ایی متداول داشته.

نویسنده: amir hosein jelodari
تاریخ: ۱۳۹۰/۰۶/۲۱ ۱۹:۵۹:۱۰

سلام ... خوشحال میشم اگه مطلبی در باب مقایسه ی EF و NHibernate بنویسید ... البته چیزی که تا الان پیش رفتن!

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۶/۲۱ ۲۰:۰۷:۰۹

در همین رابطه: [^]

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۶/۲۲ ۲۰:۰۷:۱۶

یک کتاب دیگر هم به نظر در زمینه NH 3.0 منتشر شده:

[Working with NHibernate 3.0](#)

نویسنده: Mohammad Kalhori
تاریخ: ۱۳۹۰/۰۷/۱۸ ۲۱:۱۹:۱۴

با سلام و خسته نباشید

مهندس مطالب سایتتون عالی هست واقعا متشکرم.

من به تازگی به دنبال یک ORM خوب می گشتم که تو سایت Codeplex به nhydrate برخوردی ویژگی های خوبی داره <http://nhhydrate.codeplex.com> ولی خیلی جویونه. چون زیاد با ORM ها کار نکردم و benchmark خوبی هم در این باره پیدا نکردم اگر راهنمایی بفرمایید که کار کردن با همچین ORM بجای NHibernate یا EF و ... کار درستی هست و در دراز مدت قابل توسعه و عیب یابی , خوبی می تونم از پروژم داشته باشم بسیار ممنون میشم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۷/۱۸ ۲۲:۴۰:۴۸

سورس این نوع پروژه ها برای کسانی که EF یا NH رو نوشتن خوبه. به اون ها ایده می ده؛ ولی به درد من و شما نمی خوره و وقت تلف کردن است. از هر چیزی که GUI داره برای طراحی دوری کنید! ORM ها زمانیکه قسمت عمده کار را از شما مخفی می کنند شاید به ظاهر برای یک تازه کار جذاب باشند ولی هر چقدر که جلوتر می رن محبوب تر می شن به پشت صحنه بیشتر رجوع کنند. NH از روز اول همون پشت صحنه رو نشون می ده. صادق هست. ضمنا برای انتخاب هر کتابخانه ی نسبتا پایهای باید به این مسایل دقت کنید:

- چه تعداد کتاب چاپ شده در این مورد هست؟
 - چه تعداد انجمن برای رفع اشکال آن وجود دارد؟
 - چه تعداد بلاگ و سایت در این زمینه مطلب و مقاله منتشر می کنند؟
 - آخرین تاریخ به روز رسانی آن کی بوده؟
- benchmark مهم نیست. گیرم این یکی دو ثانیه زودتر از اون یکی جواب بده. چه اهمیتی داره؟ مهم این است که چند نفر با این کتابخانه کار می کنند، چه تعداد منبع دارد و آیا می شود در روزهای سخت دنبال پاسخگو گشت؟
- NH یک خوبی دیگر هم دارد. در سایت های دات نت مطلب پیدا نکردید، معادل یک به یک جاوا هم دارد.
- EF هم داره خوب میشه. این نگارش 4.1 آن خیلی شبیه به Fluent NHibernate شده.

یکی از دردهای عظمایی که حین کار با بانک‌های اطلاعاتی رابطه‌ای وجود دارد، هماهنگ نبودن دیتابیس توسعه، با دیتابیس کاری است. البته ابزارهای متعددی برای تهیه Diff بین این دو وجود دارند. ولی زمانیکه قرار باشد این کار را در چندجا هم انجام دهیم، باز هم مشکل خواهد بود.

با NHibernate می‌شود کل این مساله را فراموش کرد! می‌شود راحت خاصیتی را به کلاسی اضافه کرد و در اولین بار اجرای برنامه، خود NHibernate هماهنگ سازی‌ها را انجام دهد. فیلد اضافه کند. جدول اضافه کند. روابط مرتبط را اضافه کند. یعنی تا این حد که ما فقط فایل اجرایی برنامه را به روز کنیم، کافی باشد. البته در لابلای مطالبی که تا به حال در مورد NHibernate در این سایت منتشر شده به این موضوع هم پرداخته شده و مطلب جاری، خلاصه‌ی بزرگمایی شده آن‌ها است.

اولین قدم: آیا ساختار دیتابیس جاری، با مدل برنامه تطابق دارد؟

قبل از اینکه از NHibernate بخواهیم ساختار بانک اطلاعاتی ما را تغییر دهید، باید بدانیم که آیا واقعا نیازی به اینکار هست یا خیر؟ توضیحات بیشتر در مورد روش تشخیص در اینجا: ([^](#))

قدم دوم: اگر ساختار دیتابیس جاری با مدل برنامه تطابق ندارد، چگونه باید آن را به صورت خودکار به روز کرد؟

متد زیر بر اساس Configuration ابتدایی بانک اطلاعاتی و نگاشت‌های شما، کار به روز رسانی خودکار ساختار بانک اطلاعاتی را انجام خواهد داد:

```
public void UpdateDatabaseSchema(NHibernate.Cfg.Configuration config)
{
    var schemaUpdate = new NHibernate.Tool.hbm2ddl.SchemaUpdate(config);
    schemaUpdate.Execute(script: false, doUpdate: true);
}
```

یک نکته را هم باید در نظر داشت. در این روش هیچ فیلد و جدولی حذف نمی‌شود و به این ترتیب، جهت امنیت بیشتر طراحی شده. اگر واقعا نیاز داشتید فیلد یا جدولی را حذف کنید باید دستی، همانند سابق اقدام کنید.

قدم سوم: چگونه و در کجا، دو قدم قبل را با برنامه یکپارچه کنیم؟

بلافاصله پس از ایجاد SessionFactory در برنامه، متد زیر را فراخوانی کنید:

```
public void TryValidateAndUpdateDatabaseSchema(NHibernate.Cfg.Configuration config)
{
    try
    {
        ValidateDatabaseSchemaAgainstMappings();
    }
    catch
    {
        UpdateDatabaseSchema(config);
    }
}
```

متد [ValidateDatabaseSchemaAgainstMappings](#) در صورت عدم تطابق مدلی با بانک اطلاعاتی، یک exception را صادر می‌کند. بنابراین در اینجا کافی است متد UpdateDatabaseSchema را در قسمت catch فراخوانی کرد.

و از این پس دیگر می‌توانید به روز رسانی ساختار بانک اطلاعاتی برنامه را فراموش کنید! فیلد اضافه کنید، کلاس اضافه کنید، تمام این‌ها در اولین بار اجرای برنامه به روز شده، به صورت خودکار به بانک اطلاعاتی اعمال خواهند شد.

نظرات خوانندگان

نویسنده: MehdiPayervand

تاریخ: ۰۹:۴۶:۵۳ ۱۳۹۰/۱۲/۰۲

در بیشتر شرکت هایی که به طریقی میخوان از بانک هم ورژن داشته باشند معمولا کنار هر پروژه یک پروژه از نوع بانک ساخته میشه (فقط MS SQL) ولی با این امکان NHibernate دیگه احتیاجی به نگهداری یک پروژه موازی نیست، همه تسک ها بصورت مجتمع و مدیریت شده در کلاسهای سطح سلوشن نگهداری میشه.
واقعا پست هاتون برای توسعه دهنده ها یه نوع واکسیناسیون میمونه ;)

نگارش نهایی NHibernate 3.2 مدتی است که ارائه شده و به همراه آن قابلیت‌هایی همانند Fluent NHibernate جهت حذف فایل‌های XML ایی تعریف نگاشت‌ها به کمک کد نویسی هم وجود دارد. در حال حاضر آنچنان مطالب خودآموز قابل توجهی را در این مورد نمی‌توان یافت ولی در کل دو ویدیوی مقدماتی زیر می‌توانند کمک خوبی جهت شروع به کار با این امکان جدید باشند:

[Part 1\) - NHibernate Mapping by Code, SchemaExport and SchemaValidate\)](#)

[Part 2\) - NHibernate Mapping by Code, SchemaExport and SchemaValidate\)](#)

ماخذ

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۵/۱۶ ۰۸:۴۷:۵۰

اگر به آدرس‌های فوق دسترسی ندارید می‌تونید از لینک‌های زیر هم استفاده کنید:

[part-1](#)

[part-2](#)

نویسنده: Mehdi Payervand
تاریخ: ۱۳۹۰/۰۵/۲۳ ۲۳:۲۰:۳۲

ممنون، مثل همیشه مفید بود، پس با این اوصاف دیگه باید Fluent رو کنار بزنه.

شروع به کار با NH به دو قسمت تقسیم می‌شود. یک قسمت نگاشت کلاس‌ها است و قسمت دوم سشن گردانی آن. قسمت دوم آن به همان مباحث کلاس‌های singleton ایی که بحث آن‌ها در سایت هست بر می‌گردد. یا حتی استفاده از کتابخانه‌های IOC برای مدیریت آن (که این پیاده سازی را به صورت توکار هم دارند).

قسمت نگاشت کلاس‌ها در NH انواع و اقسامی دارد:

ابتدا همان فایل‌های XML مدل Hibernate جاوا بود.

بعد شد مدل annotation ایی به نام [Castle ActiveRecord](#) . (این پروژه آنچنان فعال نیست و علتش به این بر می‌گردد که نویسنده اصلی جذب مایکروسافت شده)

سپس [Fluent NHibernate](#) پدید آمد. (این پروژه هم پس از NH 3.2 ، سرد شده و به نظر آنچنان فعال نیست)

الان هم مدل جدیدی به صورت توکار و بدون نیاز به کتابخانه‌های جانبی از NH 3.2 به بعد به آن اضافه شده به نام mapping-by-code .

بنابراین روش مرجع از NH 3,2 به بعد، همین روش mapping-by-code توکار آن است. خصوصا اینکه نیاز به وابستگی خارجی ندارد. برای مثال به دلیل عدم فعال بودن پروژه‌هایی که نام برده شد، مثلا NH 3,3 امروز ارائه می‌شود، شاید دو ماه بعد، این کتابخانه‌های جانبی ساده سازی نگاشت‌ها، به روز شوند یا نشوند.

و ... خبر خوب اینکه شخصی در 18 قسمت به توضیح این قابلیت جدید mapping by code پرداخته و روش‌های نگاشت مرتبط رو با مثال توضیح داده که در آدرس زیر می‌تونید اون‌ها رو پیدا کنید:

[NHibernate's mapping-by-code - the summary](#)

نظرات خوانندگان

نویسنده: محمد صاحب
تاریخ: ۱۵:۵۳:۱۸ ۱۳۹۰/۱۱/۳۰

ممنون.
بنظرتون همین تغییرات میتونه دلیل(بهبود) خوبی برای استفاده از EF باشه.
بنظرتون به ریسکش می ارزه که وقت بذاری و فردا روزی این روش هم ...
من سایت کارویس رو که دنبال میکنم زیاد آگهی که NHibernate کار; بخوان ندیدم. بنظرتون این دلیل استقبال کم(البته با استناد به فید های سایت کارویس) چی میتونه باشه؟

نویسنده: وحید نصیری
تاریخ: ۱۶:۲۰:۵۷ ۱۳۹۰/۱۱/۳۰

- این بهبود چطوره: EF هم به سرنوشت LINQ to SQL در یک تا دو سال بعد مبتلا بشه. ولی این امر در مورد NH صادق نیست.
- این روش چون جزو خود کتابخانه پایه شده و نویسنده اصلی آن هم خود مدیر پروژه NH است (فابیو مالو)، احتمال کنار گذاشته شدنش کم است.

نویسنده: Javad Darvish Amiry
تاریخ: ۲۰:۲۰:۴۳ ۱۳۹۰/۱۱/۳۰

با نکته اول حرف آقای نصیری خیلی خیلی موافقم (:
اینکه به قول دوست و استاد عزیزم جناب صاحب mapping-by-code فردا روزی حذف شه، خیلی فرق میکنه با اینکه EF فردا روزی کنار گذاشته بشه! همونطور که L2S کنار گذاشته شد و وب-فرم ها دارن میشن. اگه mapping-by-code نبود، از xml استفاده میکنیم و اگه اون حذف شد از Fluent و اگه اون حذف شد از x و y و z. ولی مسلما nh کماکان زنده میمونه! چیزی که در مورد موضوعات میکروسافتی خیلی باید با تردید با قضیه مواجه شد.
من پیشنهاد میکنم انرژی اصلی رو روی nh بذارید ولی از ef هم غافل نشید. اکثر پروژه های کد-باز بزرگ و معروف هم با nh نوشته شدن که خودش یه منبع عالی برای تمرین و تسلط هست. ولی تقریباً برای ef سمپل های ابتدایی و کوچیکی میشه فقط پیدا کرد. زنده باشین.

نویسنده: Nasr
تاریخ: ۰۸:۵۵:۰۶ ۱۳۹۰/۱۲/۰۱

با عرض سلام
برای آموزش NHibernate از کجا باید شروع کرد؟

ممنون

نویسنده: MehdiPayervand
تاریخ: ۱۰:۴۱:۳۱ ۱۳۹۰/۱۲/۰۱

ممنون، 1یه مشکلی توی برنامه BloggetToCHM دیدم، زمانیکه این بلاگ رو آپدیت میکنم لینکهای داخلی فایل به صفحه هدایت نمیشه با اینکه خود صفحات رو میشه مستقیم از داخل فایل دید، باز هم ممنون

نویسنده: MehdiPayervand
تاریخ: ۱۰:۴۶:۳۱ ۱۳۹۰/۱۲/۰۱

برای شروع کتابهای زیادی هست ولی من خودم بشخصه راهنمایی که جناب نصیری توی این وبلاگ گذاشتن رو بهترین شروع میدونم

در صورتیکه تمایل داشتید فایل پی دی اف این مجموعه تهیه شده که از لینک زیر میتونید دریافتش کنید، البته بعد از این فایل بنظرم فصل یازدهم کتاب NHibernate 3 Beginners Guide - Aug.2011 هم برای آشنایی با مشکلاتی که بطور معمول برنامه نویسان مرتکب میشن خیلی خوبه.

لینک: <http://www.mediafire.com/?4a3ajd1t787ha6n>

پیشتر مطلبی را در مورد 18 مقاله‌ای که اکثر حالت‌های Mapping موجود در NHibernate را خلاصه کرده بود، [مطالعه کردید](#). یک مورد هم در این مطلب به نظر در مقایسه با Fluent NHibernate در نظر گرفته نشده است و آن هم بحث [AutoMapping](#) است. Fluent NHibernate این قابلیت را دارد که بر اساس تعاریف کلاس‌های شما و روابط بین آن‌ها به صورت خودکار نگاشت‌ها را تشکیل دهد. یعنی خودش مباحث ارتباط‌های یک به چند و چند به چند و غیره را در پشت صحنه به صورت خودکار تولید کند؛ بدون حتی یک سطر کدنویسی اضافی. فقط حداکثر یک سری [IAutoMappingOverride](#) و همچنین تعدادی [Convention](#) نامگذاری را هم می‌توان جهت تنظیمات این سیستم تمام خودکار اعمال کرد. مثلاً توسط [IAutoMappingOverride](#)، یکی از خاصیت‌های کلاس را به صورت Unique معرفی کرد و مابقی هم به صورت خودکار توسط قابلیت Automapping نگاشت می‌شوند. یا توسط Convention نامگذاری سفارشی خود، به Fluent NHibernate اعلام می‌کنیم که من علاقمندم نام مثلاً کلیدهای خارجی تشکیل شده بجای اعدادی منحصر بفرد، از روش ویژه‌ای که تعیین می‌کنم، ساخته شوند. اینجا است که به نظر من کار با NHibernate حتی از Entity framework هم ساده‌تر است (یا ساده‌تر شده است).

قابلیت AutoMapping یاد شده، در سیستم جدید توکار Mapping by code هم وجود دارد. فقط چون جایی به صورت درست و درمان مستند نشده، همه دور خودشان می‌چرخند! به همین جهت مثالی را در این زمینه آماده کردم که موارد زیر را پوشش می‌دهد:

- نحوه اعمال تنظیمات بانک اطلاعاتی با کدنویسی در NH3,2
- نحوه یکپارچه سازی این روش جدید با کتابخانه [NHibernate Validator](#)
- استفاده از NHibernate Validator جهت تنظیم طول فیلدهای بانک اطلاعاتی تولیدی به صورت خودکار
- نحوه تعریف قراردادهای نامگذاری ویژه (مثلاً نام جداول تولید شده به صورت خودکار، برخلاف نام موجودیت‌ها، جمع باشد نه مفرد)
- اضافه کردن قابلیت تشخیص many-to-many به auto-mapping موجود در NH3,2 (برای تشخیص این مورد به کمک امکانات مهبیای پیش فرض NH3,2، باید اندکی کدنویسی کرد)
- نحوه بازنویسی قراردادهای پیش فرض auto-mapping. مثلاً اگر قرار باشد همه چیز خودکار شود اما یکی از فیلدها به صورت unique معرفی شود چکار باید کرد.
- [نحوه](#) ذخیره اطلاعات mapping کامپایل شده در فایل فایل و سپس بارگذاری خودکار آن در حین اجرای برنامه جهت بالا بردن سرعت بارگذاری اولیه برنامه.

[NHibernate 3.2 Auto-Mapping Sample](#)

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۲/۱۰ ۲۲:۴۰:۰۱

این مثال رو مجدداً به روز کردم. این قابلیت‌ها به آن اضافه شده:
- استفاده از یک کلاس پایه جهت قرار دادن یک سری خواص تکراری در آن، بدون لحاظ شدن این کلاس پایه به عنوان یک موجودیت مستقل.
- ارث بری از یک کلاس پایه و نگاشت خودکار آن.
- پشتیبانی از موجودیت‌های خود ارجاعی (سلسله مراتبی).

نویسنده: MehdiPayervand
تاریخ: ۱۳۹۰/۱۲/۱۱ ۱۲:۴۷:۴۰

ازتون بابت اشتراک دانشتون ممنونم

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۲/۱۲ ۱۰:۳۹:۴۵

EnumConvention رو هم به مثال اضافه کردم. به صورت پیش فرض Enumها در NH به اعداد Map می‌شوند. اگر نیاز بود به رشته‌های معادل مپ شوند، این EnumConvention خودکار مفید خواهد بود.

نویسنده: MehdiPayervand
تاریخ: ۱۳۹۰/۱۲/۱۲ ۱۲:۲۸:۳۷

اتفاقاً دیروز بدنال نمونه مناسب از استفاده مپ برای Enum می‌گشتم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۲/۱۶ ۰۹:۱۸:۴۸

مثال رو مجدداً به روز کردم. Component mapping و همچنین یک روال نامگذاری سفارشی خودکار هم برای آن اضافه شد.
Component mapping مربوط به حالتی است که شما یک سری خواص را در یک کلاس تعریف می‌کنید اما قصد ندارید این‌ها واقعاً تبدیل به یک جدول مجزا در بانک اطلاعاتی شوند. می‌خواهید این خواص دقیقاً در همان جدول اصلی کنار مابقی خواص قرار گیرد. اما در طرف کدهای ما به شکل یک کلاس مجزا تعریف شود.

نویسنده: علی رضا71
تاریخ: ۱۳۹۲/۱۱/۰۶ ۲۳:۰۹

چرا به این بخش رسیدگی نمی‌کنید؟ آیا این روش منسوخ شده؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۰۶ ۲۳:۱۱

نه. [زمانه تغییر کرده](#).

Second Level Cache In NHibernate 4

همان طور که می‌دانیم کش در NHibernate در دو سطح قابل انجام می‌باشد:

- کش سطح اول که همان اطلاعات سشن، در تراکنش جاری هست و با اتمام تراکنش، محتویات آن خالی می‌گردد. این سطح همیشه فعال می‌باشد و در این بخش قصد پرداختن به آن را نداریم.
- کش سطح دوم که بین همه‌ی تراکنش‌ها مشترک و پایدار می‌باشد. این مورد به طور پیش فرض فعال نمی‌باشد و می‌بایستی از طریق کانفیگ برنامه فعال گردد.

جهت پیاده سازی باید قسمت‌های ذیل را در کانفیگ مربوط به NHibernate اضافه نمود:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
<configSections>
    <section name="hibernate-configuration" type="NHibernate.Cfg.ConfigurationSectionHandler,
NHibernate"/>
    <section name="syscache" type="NHibernate.Caches.SysCache.SysCacheSectionHandler,
NHibernate.Caches.SysCache" requirePermission="false"/>
</configSections>

<hibernate-configuration xmlns="urn:hibernate-configuration-2.2" >
<session-factory>
    <property name="connection.provider">NHibernate.Connection.DriverConnectionProvider</property>
    <property name="dialect">NHibernate.Dialect.MsSql2005Dialect</property>
    <property name="connection.driver_class">NHibernate.Driver.SqlClientDriver</property>
    <property name="connection.connection_string_name">LocalSqlServer</property>
    <property name="show_sql">false</property>
    <property name="hbm2ddl.keywords">none</property>
    <property name="cache.use_second_level_cache">true</property>
    <property name="cache.use_query_cache">true</property>
    <property name="cache.provider_class">NHibernate.Caches.SysCache.SysCacheProvider,
NHibernate.Caches.SysCache</property>
</session-factory>
</hibernate-configuration>

<syscache>
    <cache region="LongExpire" expiration="3600" priority="5"/>
    <cache region="ShortExpire" expiration="600" priority="3"/>
</syscache>
</configuration>
```

پیاده سازی Caching در NHibernate در سه مرحله قابل اعمال می‌باشد :

- کش در سطح Load موجودیت‌های مستقل
- کش در سطح Load موجودیت‌های وابسته Set , List , Bag , ...
- کش در سطح Query ها

Providerهای مختلفی برای اعمال و پیاده سازی آن وجود دارند که معروف‌ترین آن‌ها SysCache بوده و ما هم از همان استفاده می‌نماییم.

- مدت زمان پیش فرض کش سطح دوم، ۵ دقیقه می‌باشد و در صورت نیاز به تغییر آن، باید تگ مربوط به SysCache را تنظیم نمود. محدودیتی در تعریف تعداد متفاوتی از زمان‌های خالی شدن کش وجود ندارد و مدت زمان آن بر حسب ثانیه مشخص می‌گردد. نحوه‌ی تخصیص زمان انقضای کش به هر مورد بدین شکل صورت می‌گیرد که region مربوطه در آن معرفی می‌گردد.

جهت اعمال کش در سطح Load موجودیت‌های مستقل، علاوه بر کانفیگ اصلی، می‌بایستی کدهای زیر را به Mapping موجودیت اضافه نمود مانند :

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2" assembly="Core.Domain"
namespace="Core.Domain.Model">
  <class name="Organization" table="Core_Enterprise_Organization">
    <cache usage="nonstrict-read-write" region="ShortExpire"/>
    <id name="Id" >
      <generator/>
    </id>
    <version name="Version"/>
    <property name="Title" not-null="true" unique="true"/>
    <property name="Code" not-null="true" unique="true"/>
  </class>
</hibernate-mapping>
```

این مورد برای موجودیت‌های وابسته هم نیز صادق است؛ به شکل کد زیر:

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2" assembly="Core.Domain"
namespace="Core.Domain.Model">
  <class name="Party" table="Core_Enterprise_Party">
    <id name="Id" >
      <generator />
    </id>
    <version name="Version"/>
    <property name="Username" unique="true"/>
    <property name="DisplayName" not-null="true"/>
    <bag name="PartyGroups" inverse="true" table="Core_Enterprise_PartyGroup" cascade="all-delete-
orphan">
      <cache usage="nonstrict-read-write" region="ShortExpire"/>
      <key column="Party_id_fk"/>
      <one-to-many/>
    </bag>
  </class>
</hibernate-mapping>
```

ویژگی usage نیز با مقادیر زیر قابل تنظیم است:

- read-only : این مورد جهت موجودیت‌هایی مناسب است که امکان بروزرسانی آن‌ها توسط کاربر وجود ندارد. این مورد بهترین کارایی را دارد.

- read-write : این مورد جهت موجودیت‌هایی بکار می‌رود که امکان بروزرسانی آن‌ها توسط کاربر وجود دارد. این مورد کارایی پایین‌تری دارد.

- nonstrict-read-write : این مورد جهت موجودیت‌هایی مناسب می‌باشد که امکان بروزرسانی آن‌ها توسط کاربر وجود دارد؛ اما امکان همزمان بروز کردن آن‌ها توسط چندین کاربر وجود نداشته باشد. این مورد در قیاس، کارایی بهتر و بهینه‌تری نسبت به مورد قبل دارد.

جهت اعمال کش در کوئری‌ها نیز باید مراحل خاص خودش را انجام داد. به عنوان مثال برای یک کوئری Linq به شکل زیر خواهیم داشت:

```
public IList<Organization> Search(QueryOrganizationDto dto)
{
    var q = SessionInstance.Query<Organization>();
    if (!String.IsNullOrEmpty(dto.Title))
        q = q.Where(x => x.Title.Contains(dto.Title));
    if (!String.IsNullOrEmpty(dto.Code))
        q = q.Where(x => x.Code.Contains(dto.Code));
    q = q.OrderBy(x => x.Title);
    q = q.CacheRegion("ShortExpire").Cacheable();
    return q.ToList();
}
```

در واقع کد اضافه شده به کوئری بالا، قابل کش بودن کوئری را مشخص می‌نماید و مدت زمان کش شدن آن نیز از طریق کانفیگ مربوطه مشخص می‌گردد. این نکته را هم در نظر داشته باشید که کش در سطح کوئری برای کوئری‌هایی که دقیقا مثل هم هستند اعمال می‌گردد و با افزوده یا کاسته شدن یک شرط جدید به کوئری، مجددا کوئری سمت پایگاه داده ارسال می‌گردد.

در انتها لینک‌های زیر هم جهت مطالعه بیشتر پیشنهاد می‌گردند:

<http://www.nhforge.org/doc/nh/en/index.html#performance-cache-readonly>

<http://nhforge.org/blogs/nhibernate/archive/2009/02/09/quickly-setting-up-and-using-nhibernate-s-second-level-cache.aspx>

<http://www.klopfenstein.net/lorenz.aspx/using-syscache-as-secondary-cache-in-nhibernate>

<http://stackoverflow.com/questions/1837651/nhibernate-cache-strategy>