عنوان: **فیلدهای پویا در NHibernate**
نویسنده: وحید نصیری
تاریخ: ۰۱:۰۸:۰۰ ۱۳۹۰/۰۳/۲۷
آدرس: www.dotnettips.info
برچسب‌ها: NHibernate

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 نگاشت آن‌را 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 نگاشت فوق است.
ابتدا از IgnoreProperty جهت ندید گرفتن Attributes استفاده کردیم. زیرا درغیراینصورت در حالت Auto mapping ، یک رابطه چند به یک به علت وجود IDictionary به صورت خودکار ایجاد خواهد شد که نیازی به آن نیست (یافتن این نکته نصف روز کار برد ....! چون مرتبا 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 باتوجه 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، اسکریپت 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: ۱۳۹۰/۰۴/۰۳ ۱۳:۵۸:۱۰

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: ۱۳۹۰/۰۴/۰۳ ۱۷:۱۱:۱۱

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: ۱۳۹۰/۰۴/۰۳ ۱۷:۱۶:۲۸

در مورد آشنایی کلی با طرز کارش هم می‌تونید به این پروژه مراجعه کنید: [+]

نویسنده: afsharm
تاریخ: ۱۳۹۰/۰۴/۰۶ ۰۹:۴۵:۲۴

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: ۱۳۹۰/۰۴/۰۶ ۱۰:۴۳:۴۳

بحث فوق را می‌شود با امکانات داینامیک سی شارپ 4 هم توسعه داد یعنی استفاده از اشیاء داینامیک 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.