عنوان: **نگاشت IDictionary در Fluent NHibernate**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۴/۰۴ ۲۱:۱۰:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: NHibernate

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؟ کلا 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 گرنه 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 { set; get; }
       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.