عنوان: QueryOver در NHibernate و تفاوت‌های آن با LINQ to NH
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۲/۲۴ ۱۱:۳۶:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: NHibernate

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 عملا تعریف نشده است و نباید آن‌را با LINQ اشتباه گرفت) و سعی در استفاده از آن‌ها به استثناهای 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 ، EndsWith (MatchMode.End) ، StartsWith (MatchMode.Start) 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 accountsContianX = session.QueryOver<Account>()
                                        .Where(x => x.Name.IsLike("%X%"))
                                        .List();
```

این فضای نام شامل چهار extension method به نام‌های IsIn ، IsInsensitiveLike ، IsLike و IsBetween 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 آن‌را 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: (+).