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

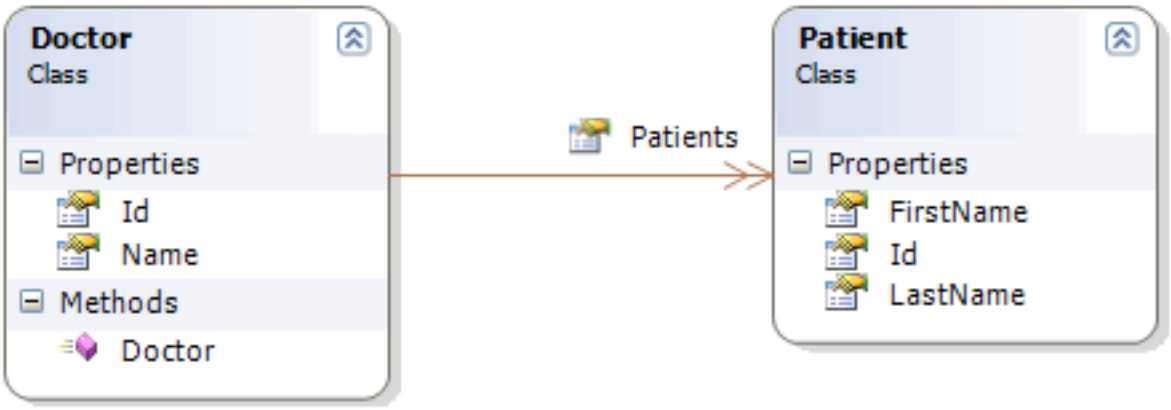
این پروژه که توسط Dario Quintana توسعه یافته است، امکان اعتبار سنجی اطلاعات را پیش از افزوده شدن آن‌ها به دیتابیس به دو صورت دستی و یا خودکار و یکپارچه با NHibernate فراهم می‌سازد؛ که امروز قصد بررسی آن‌را داریم.

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; }
    }
}
```

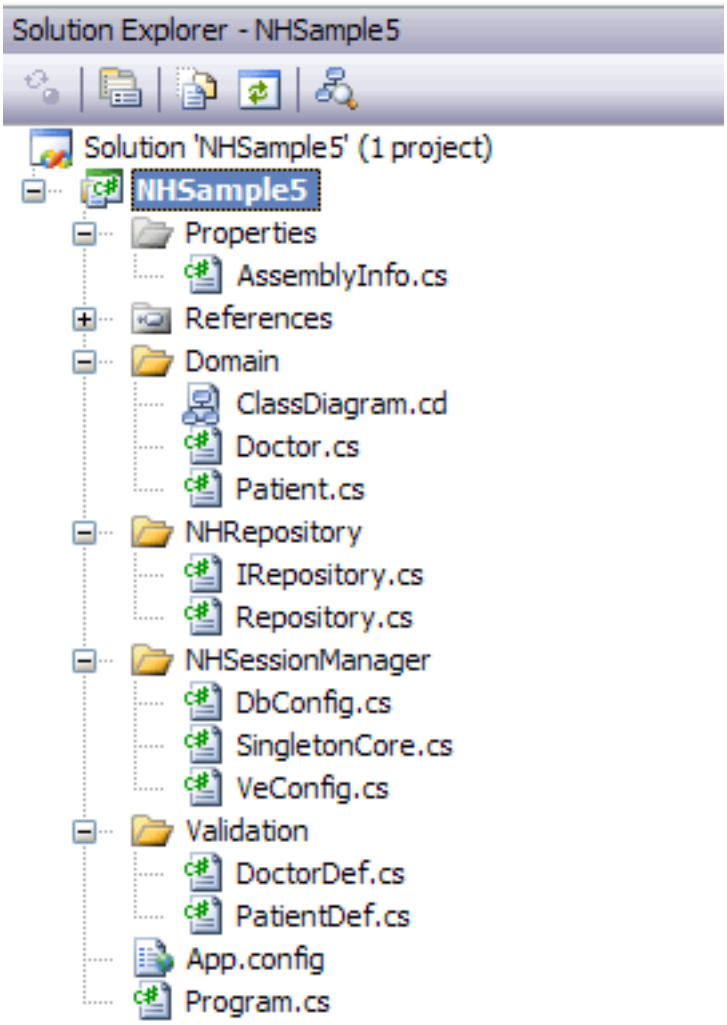
```
using System.Collections.Generic;

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 ، FluentNHibernate.dll ،log4net.dll ،NHibernate.dll NHibernate.Validator.dll ،NHibernate.Linq.dll ،NHibernate.ByteCode.Castle.dll و System.Data.Services.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 پیشنیاز 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 عتبار سنجی با موفقیت روبر 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 دقیقا 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 حتما 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 زمانیکه با NHibernate یکپارچه گردد دو رخداد PreInsert و PreUpdate آن‌را 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);
            //با 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 یا IsValid نبوده و کار اعتبار سنجی به صورت خودکار و یکپارچه با NHibernate انجام می‌شود. لطفا به مثال زیر دقت بفرمائید:

```
/// <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 (InvalidStateException 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 دقیقا 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);
       }
```

دریافت سورس کامل قسمت دهم

نظرات خوانندگان

نویسنده: Nima
تاریخ: ۰۹:۱۰:۴۸ ۱۳۸۸/۰۸/۰۳

سلام.

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

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

بهرحال از اینکه سرنخ رو به دستمون دادی ممنونم

نویسنده: مهدی پایروند
تاریخ: ۱۱:۴۷:۵۸ ۱۳۸۸/۰۸/۰۹

سلام، توی http://vahid.nasiri.googlepages.com/NH-links.txt یه فایل به اسم Summer_20ofNhibernate_20Session_2005.avi از رپیدشر پاک شده اگه ممکنه لینک دانلود مستقیمشو تو وبلاگ بذارین!
ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳:۰۵:۲۱ ۱۳۸۸/۰۸/۰۹

سلام،
خودتون هم می‌تونید اینکار را انجام بدید.
یک فایل در رپیدشیر آپلود کنید. سپس یک لینک به شما می‌دهد جهت ایجاد collectors account . این اکانت کالکتور را که ایجاد کردید، امکان remote upload هم دارد. (این نوع اکانت‌ها پولی نیست اما امکانات خوبی دارد و کار راه انداز است)

نویسنده: مهدی پایروند
تاریخ: ۱۳:۴۲:۰۷ ۱۳۸۸/۰۸/۰۹

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

نویسنده: وحید نصیری
تاریخ: ۱۴:۳۶:۰۱ ۱۳۸۸/۰۸/۰۹

- اشکالی نداره که پاک شده. خودتون فایل رو ترنس لود کنید. یک اکانت کالکتور در رپید شیر درست کنید. بعد لاگین کنید. سپس به قسمت remote upload آن مراجعه کرده و لینک‌های 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: ۱۴:۴۳:۲۰ ۱۳۸۸/۰۸/۰۹

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 نویس 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 می‌شد.

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 نمی‌کنه. 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 پروژه‌ی دیگری که ارجاعی از آن در برنامه وجود دارد، دسترسی به آن از طریق reflection است و نه parse مستقیم کلاس‌های 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 ویژگی‌ها استفاده نمی‌کنیم وجود دارد؟ (در مثال شما این کار انجام نمی‌شود)

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 FHN است و هم با تزریقی که صورت گرفته یکپارچه شده و هم از ویژگی‌ها استفاده نشده.

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

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

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: ۱۳۸۹/۰۱/۰۴ ۱۷:۴۸:۱۶

ضمنا 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: ۱۳۸۹/۰۱/۰۴ ۱۸:۵۷:۱۱

از راهنمایی خوبتان متشکرم.

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

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: ۱۳۹۰/۰۱/۱۶ ۲۳:۳۰:۴۶

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: ۱۳۹۰/۰۱/۱۷ ۰۰:۱۶:۵۳

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