عنوان: **Static Reflection**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۵/۰۹ ۲۰:۵۲:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: #C

قابلیت Dynamic reflection یا به اختصار همان reflection متداول، از اولین نگارش‌های دات نت فریم در دسترس است و امکان دسترسی به اطلاعات مرتبط با کلاس‌ها، متدها، خواص و غیره را در زمان اجرا مهیا می‌سازد. تابحال به کمک این قابلیت، امکان تهیه‌ی ابزارهای پیشرفته‌ی زیر مهیا شده است:

انواع و اقسام
- فریم ورک‌های آزمون واحد
- code generators
- ORMs
- ابزارهای آنالیز کد
و ...

برای مثال فرض کنید که می‌خواهید برای یک کلاس به صورت خودکار، متدهای آزمون واحد تهیه کنید (تهیه یک code generator ساده). اولین نیاز این برنامه، دسترسی به امضای متدها به همراه نام آرگومان‌ها و نوع آن‌ها است. برای حل این مساله باید برای مثال یک parser زبان سی شارپ یا اگر بخواهید کامل‌تر کار کنید، به ازای تمام زبان‌های قابل استفاده در دات نت فریم ورک باید parser تهیه کنید که ... کار ساده‌ای نیست. اما با وجود reflection به سادگی می‌توان به این نوع اطلاعات دسترسی پیدا کرد و نکته‌ی مهم آن هم این است که مستقل است از نوع زبان مورد استفاده. به همین جهت است که این نوع ابزارها را در فریم ورک‌هایی که فاقد امکانات reflection هستند، کمتر می‌توان یافت. برای مثال کیفیت کتابخانه‌های آزمون واحد CPP در مقایسه با آنچه که در دات نت مهیا هستند، اصلا قابل مقایسه نیستند. برای نمونه به یکی از معظم‌ترین فریم ورک‌های آزمون واحد CPP که توسط گوگل تهیه شده مراجعه کنید : (+)
قابلیت Reflection ، مطلب جدیدی نیست و برای مثال زبان جاوا هم سال‌ها است که از آن پشتیبانی می‌کند. اما نگارش سوم دات نت فریم ورک با معرفی LINQ ، lambda expressions و Expressions در یک سطح بالاتر از این Dynamic reflection متداول قرار گرفت.

تعریف Static Reflection :
استفاده از امکانات Reflection API بدون بکارگیری رشته‌ها، به کمک قابلیت اجرای به تعویق افتاده‌ی LINQ، جهت دسترسی به متادیتای المان‌های کد، مانند خواص، متدها و غیره.
برای مثال کد زیر را در نظر بگیرید:

```
//dynamic reflection
PropertyInfo property = typeof (MyClass).GetProperty("Name");
MethodInfo method = typeof (MyClass).GetMethod("SomeMethod");
```

این کد، یک نمونه از دسترسی به متادیتای خواص یا متدها را به کمک Reflection متداول نمایش می‌دهد. مهم‌ترین ایراد آن استفاده از رشته‌ها است که تحت نظر کامپایلر نیستند و تنها زمان اجرا است که مشخص می‌شود آیا MyClass واقعا خاصیتی به نام Name داشته است یا خیر.
چقدر خوب می‌شد اگر این قابلیت بجای dynamic بودن (مشخص شدن در زمان اجرا)، استاتیک می‌بود و در زمان کامپایل قابل بررسی می‌شد. این امکان به کمک lambda expressions و expression trees دات نت سه بعد، میسر شده است. کلیدهای اصلی Static Reflection کلاس‌های Func و Expression هستند. با استفاده از کلاس Func می‌توان lambda expression ایی را تعریف کرد که مقداری را بر می‌گرداند و توسط کلاس Expression می‌توان به محتوای یک delegate دسترسی یافت. ترکیب این دو، قدرت دستیابی به اطلاعاتی مانند PropertyInfo را در زمان طراحی کلاس‌ها، می‌دهد؛ با توجه به اینکه:
- کاملا توسط intellisense موجود در VS.NET پشتیبانی می‌شود.

- با استفاده از ابزارهای refactoring قابل کنترل است.
- از همه مهم‌تر، دیگری خبری از رشته‌ها نبوده و همه چیز تحت کنترل کامپایلر قرار می‌گیرد.

و شاید هیچ قابلیتی به اندازه‌ی Static Reflection در این چندسال اخیر بر روی اکوسیستم دات نت فریم ورک تاثیرگذار نبوده باشد. این روزها کمتر کتابخانه یا فریم ورکی را می‌توانید پیدا کنید که از Static Reflection استفاده نکند. سرآغاز استفاده گسترده از آن به Fluent NHibernate بر می‌گردد؛ سپس در انواع و اقسام ORMs ، mocking frameworks و غیره استفاده شد و مدتی است که در ASP.NET MVC نیز مورد استفاده قرار می‌گیرد (برای مثال TextBoxFor معروف آن):

```
public string TextBoxFor<T>(Expression<Func<T,object>> expression);
```

به این ترتیب حین استفاده از آن دیگری نیازی نخواهد بود تا نام خاصیت مدل مورد نظر را به صورت رشته وارد کرد:

```
<%= this.TextBoxFor(model => model.FirstName); %>
```

یک مثال ساده از تعریف و بکارگیری Static Reflection :

```
public PropertyInfo GetProperty<T>(Expression<Func<T, object>> expression)
{
 var memberExpression = expression.Body as MemberExpression;

 if (memberExpression == null)
     throw new InvalidOperationException("Not a member access.");

  return memberExpression.Member as PropertyInfo;
}
```

همانطور که عنوان شد کلیدهای اصلی بهره‌گیری از امکانات Static reflection ، استفاده از کلاس‌های Expression و Func هستند که در آرگومان متد فوق بکارگرفته شده‌اند و در حقیقت یک expression of a delegate است که به آن Lambdas as Data نیز گفته می‌شود. این delegate پارامتری از نوع T را دریافت کرده و سپس مقداری از نوع object را بر می‌گرداند. اما زمانیکه از کلاس Expression در اینجا استفاده می‌شود، این Func دیگر اجرا نخواهد شد، بلکه از آن به عنوان قطعه کدی که اطلاعاتش قرار است استخراج شود (Lambdas as Data) استفاده می‌شود.
برای نمونه Fluent NHibernate در پشت صحنه متد Map ، به کمک متدی شبیه به GetProperty فوق، a => a.Address1 را به رشته متناظر خاصیت Address1 تبدیل کرده و جهت تعریف نگاشت‌ها مورد استفاده قرار می‌دهد:

```
public class AddressMap : DomainMap<Address>
{
 public AddressMap()
 {
    Map(a => a.Address1);
 }
}
```

جهت اطلاع؛ قابلیت استفاده از «کد به عنوان اطلاعات» هم مفهوم جدیدی نیست و برای مثال زبان Lisp چند دهه است که آن‌را ارائه داده است!

برای مطالعه بیشتر:

Expression Tree Basics

Functional Programming for Everyday .NET Development

Introduction to static reflection

The basics behind static reflection

Dynamic reflection versus static reflection

Static Reflection of property names

Lisp is sin

نظرات خوانندگان

نویسنده: afsharm
تاریخ: ۱۳۹۰/۰۵/۱۰ ۰۸:۳۷:۵۸

سلام،

من همیشه اینجا چیزهای جدید یاد می‌گیرم.

نویسنده: Nima
تاریخ: ۱۳۹۰/۰۵/۱۰ ۱۱:۳۳:۰۲

سلام آقای نصیری
باز هم انگار در این پست مشکل اخیر من رو آموزش دادین.یه جورایی خیلی جالبه
من یه قطعه کد دیده بودم برای پیاده سازی INotifyPropertyChanged که در اینجا پرسیدم :
http://stackoverflow.com/questions/6829099/how-this-code-works-for-handling-inotifypropertychanged
البته آقای مارک گراول گفته که این روش سرعتش پایینه. البته من همچنان از کد خیلی سر در نیاوردم. یعنی سلسله مراتبی که انجام داده رو متوجه نمیشم از کجا نشات میگیره

ممنون و موفق باشی

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۵/۱۰ ۱۲:۱۱:۰۱

بله. این هم یکی از کاربردهای static reflection در عمل است که در WPF و سیلورلایت می‌تونه مورد استفاده قرار بگیره.
هدف هم حذف رشته ذکر شده در متدهای متداول و اجباری PropertyChanged است که باید به ازای هر خاصیت نوشته شود.
این رشته‌ها (آرگومان‌های PropertyChanged) چون دقیقا همان نام خاصیت‌های تعریف شده در کلاس جاری هستند، بنابراین با استفاده از lambda به عنوان داده (توسط کلاس expression و func) به صورت strongly typed و همچنین قابل تشخیص توسط intellisense می‌توانند تفسیر و قابل دسترسی شوند. زمانیکه Expression Func of T را بجای آرگومان رشته‌ای تعریف کردید، خواص این T توسط intellisense و lambda expression ظاهر می‌شوند. تا اینجا یک مرحله پیشرفت است (شما دیگر رشته ننوشته‌اید و کد هست به عنوان داده). مرحله بعد ترجمه این کد هست به همان رشته. نهایتا متد PropertyChanged نیاز به رشته دارد. اینجا است که کلاس Expression وارد عمل می‌شود و کد را به داده مورد نظر ترجمه می‌کند.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۵/۱۰ ۱۲:۲۰:۵۸

جهت تکمیل بحث این کتاب هم اخیرا چاپ شده و بگردید می‌تونید پیداش کنید ....
Functional Programming in C#: Classic Programming Techniques for Modern Projects

نویسنده: Nima
تاریخ: ۱۳۹۰/۰۵/۱۰ ۱۲:۵۳:۳۸

بسیار عالی بود متوجه شدم.بسیار لطف کردین