Creating Custom Dynamic Classes

Jason Roberts
.NET MVP

@robertsjason   dontcodetired.com

Overview

Why custom dynamic Classes?

The IDynamicMetaObjectProvider interface

The DynamicObject base class

Number of virtual methods

Custom dynamic HtmlElement class

dynamic image = new HtmlElement("img");

image.src = "car.png";

string html = image.ToString();

Why Custom Dynamic Classes?

Non static structures
Highly fluid / untyped
Unknown to compiler
Known during runtime
ViewBag.Title = “xyz”;

Improved readability
(non-dynamic) clutter
Clearer intent

Interoperability
IronPython
IronRuby

The IDynamicMetaObjectProvider Interface

The IDynamicMetaObjectProvider Interface

```
public sealed class ExpandoObject :
    IDynamicMetaObjectProvider,
    IDictionary<string, object>,
    INotifyPropertyChanged


private sealed class DapperRow :
    IDynamicMetaObjectProvider,
    IDictionary<string, object>,
    ICollection<KeyValuePair<string, object>>,
    IEnumerable<KeyValuePair<string, object>>,
    IEnumerable
```

The IDynamicMetaObjectProvider Interface

```
public sealed class ExpandoObject :
    IDynamicMetaObjectProvider,
    IDictionary<string, object>,
    INotifyPropertyChanged


private sealed class DapperRow :
    IDynamicMetaObjectProvider,
    IDictionary<string, object>,
    ICollection<KeyValuePair<string, object>>,
    IEnumerable<KeyValuePair<string, object>>,
    IEnumerable
```

The DynamicObject Base Class

```
[SerializableAttribute]
public class DynamicObject : IDynamicMetaObjectProvider
```

DynamicObject

The DynamicObject class enables you to define which operations can be performed on dynamic objects and how to perform those operations. For example, you can define what happens when you try to get or set an object property, call a method, or perform standard mathematical operations such as addition and multiplication. [MSDN]

DynamicObject Virtual Methods

public virtual bool TryInvokeMember(...)	Calling a method
public virtual bool TryGetMember(...)	Getting property/field value
public virtual bool TrySetMember(...)	Setting property/field value
public virtual bool TryGetIndex(...)	Getting value by index
public virtual bool TrySetIndex(...)	Setting value by index
public virtual bool TryUnaryOperation(...)	Unary operators, e.g. !
public virtual bool TryBinaryOperation(...)	Binary operators, e.g. +
public virtual bool TryConvert(...)	Converting (casting) to other types
public virtual bool TryInvoke(...)	Invoking the object

TryGetMember

```
private readonly Dictionary<string, string> _attributes =
                          new Dictionary<string, string>();



public override bool TryGetMember(GetMemberBinder binder,
                                  out object result)
{
   string attribute = binder.Name;

   result = _attributes[attribute];

   return true;
}
```

Demo

Creating A Dynamic HtmlElement Class

```
dynamic image = new HtmlElement("img");
image.src = "car.png";
```

ShouldStoreTagName

ShouldAddAttributeNameAndValueDynamically

ShouldErrorIfAttributeNotSet

ShouldReturnDynamicMemberNames

ShouldOutputTagHtml

ShouldBeCastableToDictionary

ShouldBeEnumerable

ShouldRenderHtml

ShouldRenderHtmlOnInvoke

Summary

Why custom dynamic Classes?

The IDynamicMetaObjectProvider interface

The DynamicObject base class

Overrode DynamicObject methods

TrySetMember() & TryGetMember()

GetDynamicMemberNames()

IDictionary<string, object>

TryInvokeMember()

TryInvoke()

Next:

Interoperating with Dynamic Languages