DYNAMIC: DON’T BE AFRAID

Hadi Hariri
JetBrains
The What, the Why, the How
A tale as old as time…
Static vs Dynamic
Discussion
Threads

Blog Posts
and more blog posts

THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S COMPILING."

HEY! GET BACK TO WORK!

COMPILING!

OH. CARRY ON.
In the Static World

• Types can be implicit or explicit (var)

• Compiler Safety

• Early Binding
In a Dynamic World

- Types defined at runtime
- No Compiler (Usually)
- Late Binding
- Interpreted (Not always)
What Dynamic Developers think of Static Developers…
I need my compiler!
What Static Developers think of Dynamic Developers…
"You cannot build serious business applications in dynamic languages"
They both have Good and Bad Things
• **dynamic** keyword

• Classes/Binders and interfaces to work with dynamic types
Hosting API
Debugging API
Interop Binders
Dynamic Objects
Call-Site Caching
Expressions
Expression Compiler / Interpreter
IL Code Generator

DLR added…
The Big Picture

- IronRuby
- IronPython
- C#
- VB.NET

DLR

Bindings

- .NET
- Ruby
- Python
- Office
GET YOUR STINKING DYNAMIC TYPES OFF OF MY STATIC LANGUAGE
IS THERE A NEED?
Readability
```csharp
var assembly = Assembly.LoadFrom(@"..\..\..\..\..\ExternalTypes.dll");

var typeInfo = assembly.GetType("ExternalTypes.CustomerService");

var methodInfo = typeInfo.GetMethod("MakeCustomerPreferred");

var customerService = Activator.CreateInstance(typeInfo);

methodInfo.Invoke(customerService, null);

var customerService = new CustomerService();

customerService.MakeCustomerPreferred();
```
Interoperability
Interoperability with other languages

- IronPython
  - Interpreted
  - Can be compiled
- IronRuby
  - Interpreted
- Your own language
object calculatorType = ruby.Runtime.Globals.GetVariable("Calculator");

object calculator = ruby.Operations.CreateInstance(calculatorType);

object sum = ruby.Operations.InvokeMember(calculator, "add", 20, 30);

Console.WriteLine(String.Format("The sum is \{0\}", sum));

Console.ReadLine();
```javascript
  dynamic scope = ruby.Runtime.Globals;

  var calculator = scope.Calculator.@new();

  var sum = calculator.add(20, 30);
```
Interoperability

• Talking to COM
  • Need a type-library beforehand
  • Use Method Invocation
The Case of the DTO
Creating Dynamic Objects in C#
Options

- ExpandObject
- DynamicObject
- IDynamicMetaObjectProvider
Expando Object

• Built-in Dynamic Object. Works out of the box

• Benefits over Dictionary
  • More Fluent
  • Support for Methods
  • Supports Hierarchies
  • Implements INotifyPropertyChanged

• Limitations
  • Index Access
DEMO

ON THE FLY: BASICS OF DYNAMIC
DynamicObject

- Moving Beyond an Expando

- Built-in class which implements IDynamicMetaObjectProvider

- Allows easy creation of Dynamic types
IDynamicMetaObjectProvider

- Meta Object that performs binding
- Allows decoupling from class
- Uses DLR Expressions
- Returns DynamicObject
DEMO

DYNAMICPROVIDER
Undetermined API
Aspects of MetaProgramming

- Adding / Removing Methods
- Creating Instance Methods
- Creating Static / Class Methods
- Query Classes
public class Employee
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    public string City { get; set; }
    public string State { get; set; }
    public string Country { get; set; }

    public Employee FindById(int id) ... 

    public Employee FindByName(string name) ... 

    public Employee FindByEmail(string email) ... 

}
DEMO

DYNAMIC METHOD MISSING – SIMPLE DATA
Consuming the ever-changing
DSL's and Fluent API's
A QUICK LOOK UNDER THE HOOD...
The backbone of dynamic support

- DLR
  - Language Semantics via DLR EXpression
  - Define Late Binding Logic
DLR Expression

- Superset of Linq.Expression
- Common to multiple Languages
- DLR Expression is to DLR Languages what IL is to CLR languages

C#  VB.NET  IL  CLR

IronPython  IronRuby  DLR Expression  DLR Runtime
Late Binding

- We only know the types at runtime
- We have to figure out how to call those methods at runtime
- It’s not embedded in the “IL”
- It’s potentially slower
Late Binding

• Using Binders and Call Sites

• Using Dynamic Expression (uses former internally)

```csharp
DynamicExpression expression = Expression.Dynamic(binder, typeof(object), Expression.Constant(2), Expression.Constant(3));
```
Late Binding

```csharp
static void Main(string[] args)
{
    dynamic value = "Hello";
    Console.WriteLine(value.ToString());
}
```

```csharp
private static void Main(string[] args)
{
    object value = "Hello";
    if (<Main>o__SiteContainer0.<>p__Site1 == null)
    {
        <Main>o__SiteContainer0.<>p__Site1 = CallSite<Action<CallSite, Type, object>>.Create(Binder.InvokeMember(CSharpBinder.NullableReceiver, MethodDesc.GetCurrentMethod, typeof(CallSite)), typeof(CallSite), TypeDesc.Any);
    }
    if (<Main>o__SiteContainer0.<>p__Site2 == null)
    {
        <Main>o__SiteContainer0.<>p__Site2 = CallSite<Func<CallSite, object, object>>.Create(Binder.InvokeMember(CSharpBinder.NullableReceiver, MethodDesc.GetCurrentMethod, typeof(CallSite)), typeof(CallSite), TypeDesc.Any);
    }
    <Main>o__SiteContainer0.<>p__Site1.Target(<Main>o__SiteContainer0.<>p__Site1, typeof(Console), <Main>o__SiteContainer0.<>p__Site2);
}
```
DEMO

DYNAMICCONVERSION
SUMMING UP…
The Disadvantages

- There is no compile type-checking*
- Potentially slower (even with caching)
- There is no Intellisense*

* Partially incorrect – It’s about the tooling
Reasons to not not use dynamic
Reasons to not use dynamic

- There’s no compiler

- There’s no intellisense (Emphasis on Unit Tests)

- You shouldn’t mix dynamic and static languages
Reasons to use dynamic
Reasons to use Dynamic

- Interoperability
  - COM

- Consuming Dynamic Languages
  - Ruby
  - JavaScript

- Fluent API’s and DSL

- Consuming the *unknown*
  - Dynamic Structures

- Avoiding unnecessary “class explosion”
Thank you

Hadi Hariri
Technology Evangelist

http://hadihariri.com
http://twitter.com/hhariri
e-mail: hadi@jetbrains.com