New C#4 Features – Part V

Nico Ludwig (@ersatzteilchen)

TOC

- New C#4 Features – Part V
  - Dynamic COM Automation with C#
  - Dynamic Objects in JavaScript
  - Expando Objects in C#

- Sources:
  - Jon Skeet, CSharp in Depth

Dynamic Coding – A better Experience with COM

- You no longer need interop types:
  - Just don't use them:
    - if you want lazy dispatching and you don't bother finding method signatures yourself...
    - or if you simply don't want to or can't use type infos (w/o type library)! – You just use dynamic.
  - (Use them if you want to exploit early dispatching, e.g. if you want to use IntelliSense.)
  - To have no need to create code as an extra step is a key benefit of dynamic coding.

- But if you use interop types w/o PIAs, the DLR still makes your life easier:
  - No-PIA deployed types provide the type dynamic instead of object in their APIs.
  - E.g. *IDispatch* and *VARIANT*s are exposed as dynamics, reducing cast operations.

- With dynamic typing and COM you now have a "natural" coding experience:
  - "Plain old CLR objects" (POCOs) and COM objects can be programmed like peers.
  - During debugging the new "Dynamic View" assists you watching COM objects.

- COM as "peer objects" is one advantage of VB and PowerShell programming.

Example with COM Automation

See accompanying project <DynamicComInteropCSharp>
This example shows the application of C# dynamic coding to allow simple COM automation.

Example: Transportation of Anonymous Types' Instances

- Passing or returning anonymous types' instances from or to methods is limited.
  - Feasible solutions like generic methods or using the type *Object* are quite limiting...
  - You could only call *Object*'s methods within such a method, not the anonymous type's ones (well, you might use reflection...).
- With dynamic you can overcome these limits.
- You'll get following features:
  - Easy to use adapters.
  - But data-binding, e.g. from *IList*<*dynamic*> to your controls is still fully functional (Windows Forms and WPF).
- !!You can't access properties of anonymous types in non-friend assemblies!!
  - Anonymous types are internal and accessing them publicly may fail.
  - => Keep using them in the same assembly!

- This is just one selected example of how dynamic typing simplifies coding problems. Passing anonymous types' instances to methods accepting dynamic arguments is used with view models in ASP.NET MVC.
- Basically dynamic makes it possible to use otherwise only locally known anonymous types in other methods.
- Databinding in WPF is less "sensitive", because it also accepts different types having equally named properties. The default behavior in Windows Forms requires identical types.
- Situation in non-friend assemblies: On accessing the anonymous type's instance's properties (via a dynamically typed symbol) you'll get a *RuntimeBinderException* on the caller side, because the properties couldn't be bound.
- More:
  - The type dynamic enables us to solve a problem on programming of generics: we can call arbitrary methods on unconstrained types' instances! Esp. we can call operators in generics with dynamic dispatching!
  - .NET remoting: Also can we use dynamic dispatch to access the interface of proxy objects on marshaled instances w/o having the object's type in avail.

Leering at JavaScript: Dynamic Objects

See accompanying project <Safari Web Inspector as REPL>
Shows how we can exploit dynamic objects with dynamic properties in JavaScript.

Key aspects of dynamic programming:
(1. Dynamic Typing)
(2. Late Binding)
(3. Duck Typing)
4. Dynamic Objects

- The ability to modify objects during run time is another aspect of dynamic programming that is shown here. -> Dynamic objects.

Dynamic Objects with .NET ExpandoObjects

- Dynamic programming allows creating types, whose interfaces change at run time.
  - These interface changes are based on how you use instances of such dynamic types.
- Dynamic objects are objects that have their own late binding logic!
  - (In contrast to static objects that get bound by the compiler (early binding).)
  - We can create own dynamic APIs to put mighty data driven APIs into effect!
  - The easiest way to get this is a dynamic property bag, via the type *ExpandoObject*.
  - *ExpandoObject*s are similar to JavaScript's dynamic objects that work like dictionaries.
  - In many dynamic languages (e.g. JavaScript, Python) all objects are property bags.
- How to make use of .NET's *ExpandoObject*:
  - Import the namespace *System.Dynamic*.
  - Create a new *ExpandoObject* and assign it to a dynamic to enable dynamic dispatch.
  - Add/modify members during run time and exploit *ExpandoObject*s as dictionaries.

- In static typing types are immutable, if you don’t have the source code. – In dynamic typing this is no longer the case, because the dynamic type information of an object is part of the object's value!
- You should also check, whether a member of a dynamic object is really present, before accessing it (*RuntimeBinderException*). – This is a duck-typing aspect (Does "it" lay eggs?)!
- As a matter of fact in .NET there also exists the type *IExpando*, which is implemented by JScript .NET objects (and by dynamic objects in some situations).
- *ExpandoObject*s can also have functions as properties of a Delegate type!
- Interoperability when hosting a DLR scripting language: Expose your C# objects to IronRuby as *ExpandoObject*s!
- The class *ExpandoObject* is sealed. The discussion of more flexible dynamic objects is discussed in the next lecture.

ExpandoObjects in Action

See accompanying project <DynamicObjects>
Shows the basic usage of ExpandoObjects.

Key aspects of dynamic programming:
(1. Dynamic Typing)
(2. Late Binding)
(3. Duck Typing)
4. Dynamic Objects

Thank you!