The Dynamic module works by using Roslyn API to generate CSharp code. Roslyn is powerful enough to provide us with a lot of metadata regarding the generated code. The generated code is compiled into an assembly and then executed using Reflection. The executed code can be flushed from the memory to help reduce the load on the server afterwards. The one thing I’ve noticed is the first dynamic code being generated has the largest code startup time because it takes a few milliseconds for the Roslyn engine to startup.

Features:

1. Compile & execute dynamic C# code.
2. Supports latest C# version (C#10)
3. Able to write code using any standard .net standard packages
4. Able to write code custom to the assembly this dynamic code will be executed in (i.e., referencing and/or using classes from the executing assembly)
5. Use references from Targeting assembly
6. Supports executing multiple methods or different classes within the C# code
7. Flushes the assembly from memory after the code executes to prevent memory leaks.
8. Can treat warnings as errors
9. Injects services from executing assembly to dynamic code's methods/constructors as required.
10. Retrieve data from the executing assembly’s HttpContext as required.
11. Run code analysis (basic for now) to prevent things like blacklisted namespaces.
12. Pass custom arguments to the code being compiled & executed

There are two things that needs to be considered when executing dynamic code:

1. Global usings cannot be used. It seems that Roslyn does not have the functionality to determine this.
2. When writing top-level-statement code (ConsoleApplication) the using System; namespace needs to be included.

I’ve written unit tests to demonstrate most of the potential ways you can use the code as well as errors that can occur (I followed a Test-Driven development for this). Each unit test’s name should provide you the general idea as to what I’m testing.

There is room for improvement, but this should provide you the general idea on how it works and the capability.

You should notice that the library has two attributes included:

* DontInvokeAttribute
  + Useful if you need to include custom DTOs in the assembly but prevent them from being captured and invoked using Reflection.
* ExecuteOrderAttribute
  + Define a custom order on methods & classes in the assembly’s defined types to determine the order of execution. Handy in cases where we have multiple methods or so that needs to be executed in a sequential order.

How to use the library:

1. Write dynamic code in a file or a raw string.
2. Register the RoslynCodeService implementation and the IDynamicCodeService definition onto your project’s startup.
3. Inject the IDynamicCodeService into your class
4. Use the appropriate overload ExecuteCode method for your dynamic code.
5. Use the Action<SourceCodeConfig> to specify any custom configuration required.