Telerik Academy

Reflection

Examine and modify the structure and behavior of the program at runtime

Telerik Software Academy

Learning & Development Team

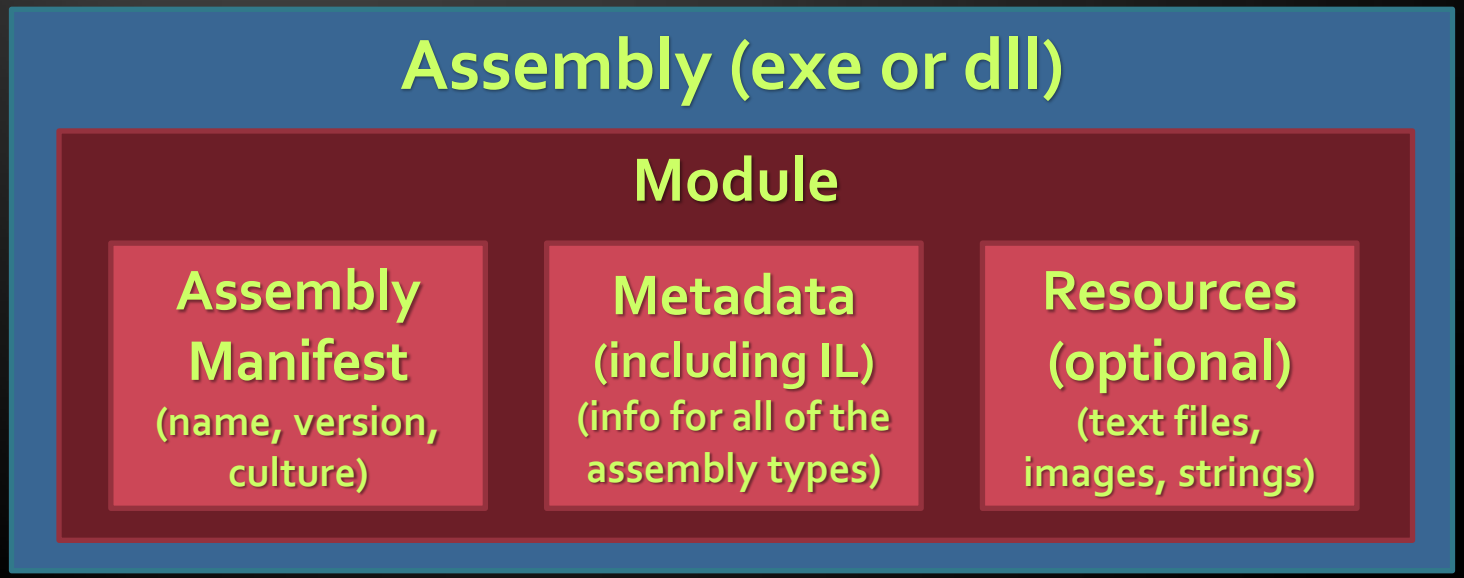
http://academy.telerik.com

Telerik Academy

Table of Contents

What is Reflection?

- Examine and modify the structure and behavior of the program at runtime
  - Inspecting the metadata and compiled code in an assembly
  - Reflection can be used for observing and/or modifying program execution at runtime



When We Use Reflection?

- Reflection is used when:
  - Examining assemblies' metadata
  - Examine assemblies' types
  - Dynamically invoking methods
  - Dynamically creating new assemblies, executing and storing them as a file
  - Inspecting an object at runtime without knowing its class
- The JustDecompile tool is an excellent example what .NET reflection can do

Drawbacks and Best Practices

- What are the costs:
  - More code than normal coding
  - Slower than normal coding
  - The "golden hammer" principle
  - Leads to a lot of casting and type checking
  - Possible magic strings
- When using Reflection
  - First try not to use it
  - Try to cache all reflected metadata

Telerik Academy

Practical Reflection Strategy

- Dynamically Load Assemblies
  - Happens one time (at start up)
- Dynamically Load Types
  - Happens one time (at start up)
- Cast Types to a Known Interface
  - All method calls go through the interface
  - No dynamic method calls – no Method.Invoke
  - Avoid interacting with private members
- Usually balance between performance, safety and flexibility

Telerik Academy

Loading Assemblies

Load libraries like a boss

System.Reflection.Assembly.Load(…)

- `Assembly.Load(…)` loads existing assembly in the .NET CLR by given
  - Assembly name or `AssemblyName` object
- It searches for an assembly with the given description (probing) and loads it if it is found
  - If the assembly is not found throws a `FileNotFoundException`

```
Assembly.Load("SampleAssembly, Version=1.0.2004.0,
  Culture=neutral, PublicKeyToken=8744b20f8da049e3");
```

System.Reflection.Assembly.LoadFrom(…)

- `Assembly.LoadFrom(…)` loads assembly from existing local file
  - Takes the path to the assembly as a parameter
  - Reads the given file and loads it in the CLR
  - If the assembly is not found throws a `FileNotFoundException`

```
Assembly.LoadFrom(@"C:\Tools\MyAssembly.dll");
```

System.Reflection.Assembly – Properties

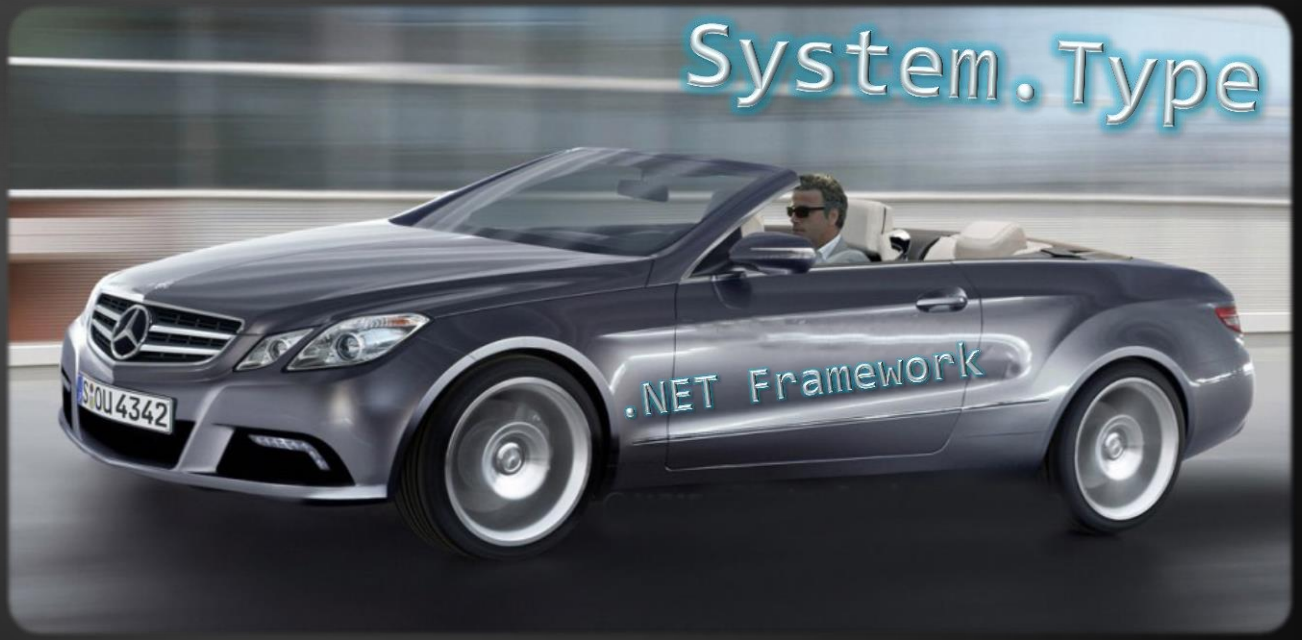
- `FullName` – the assembly's full name
  - Including version, culture and key (Public Key Token)
- `Location` – the file name from which the assembly is loaded
- `EntryPoint` – the method by which the assembly will start (`Main()` method)
- `GlobalAssemblyCache` – indicates whether the assembly is loaded from the GAC

Telerik Academy

Loading Assemblies

Live Demo

Telerik Academy



The System.Type Class

The System.Type Class

- `System.Type`
  - A starting point for inspecting .NET types
  - Provides access to all type members:
    - Fields
    - Methods
    - Properties
    - Events
    - Inner types
    - …

The System.Type Class (2)

- `Assembly.GetTypes()` allows listing all types contained in a given assembly
- Properties:

```
BaseType, Attributes, FullName, IsAbstract, IsArray,
IsByRef, IsClass, IsCOMObject, IsEnum, IsInterface,
IsPublic, IsSealed, IsValueType, Name, …
```

- Methods:

```
GetConstructors(), GetEvents(), GetFields(),
GetInterfaces(), GetMembers(), GetMethods(),
GetNestedTypes(), GetProperties(), InvokeMember(),
IsInstanceOfType()
```

Inspecting Type Members

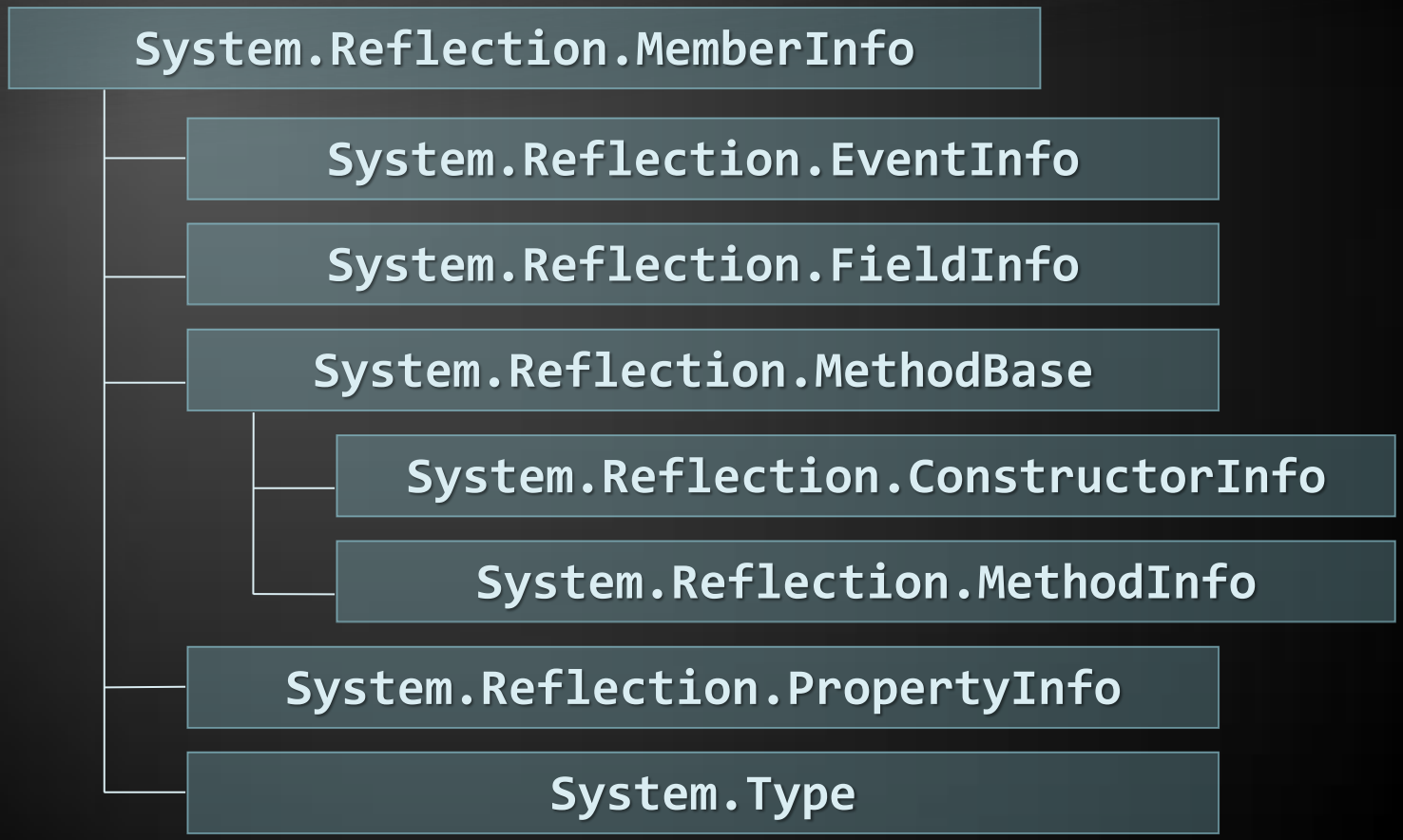
```
Assembly currAssembly =
  Assembly.GetExecutingAssembly();

foreach(Type type in currAssembly.GetTypes())
{
   foreach(MemberInfo member in type.GetMembers())
   {
      Console.WriteLine("{0}.{1}",
        member.MemberType, member.Name);
   }
}
```

Telerik Academy

Inspecting Type Members

Live Demo

Classes in the MemberInfo Hierarchy



Inspecting Methods and their Parameters

- Type.GetMethod()
  - Returns the reflection of a given method
- MethodInfo.GetParameters()
  - Extracts the method's parameters

```
MethodInfo someMethod =
  myType.GetMethod("SomeMethod");
foreach(ParameterInfo param in
  someMethod.GetParameters())
{
   Console.WriteLine(param.ParameterType);
}
```

Telerik Academy

Dynamic Method Invocation (Late Binding)

- We create type object instance by the `Activator` class
  - `CreateInstance(…)`
    - Creates an instance of given type specified as `Type` object or `string`
  - `CreateComInstanceFrom(…)`
    - Creates COM object instance
- `System.MethodInfo.Invoke(…)`
  - Dynamically invokes a method

Telerik Academy

Dynamic Method Invocation

Live Demo

Reflection and Generics

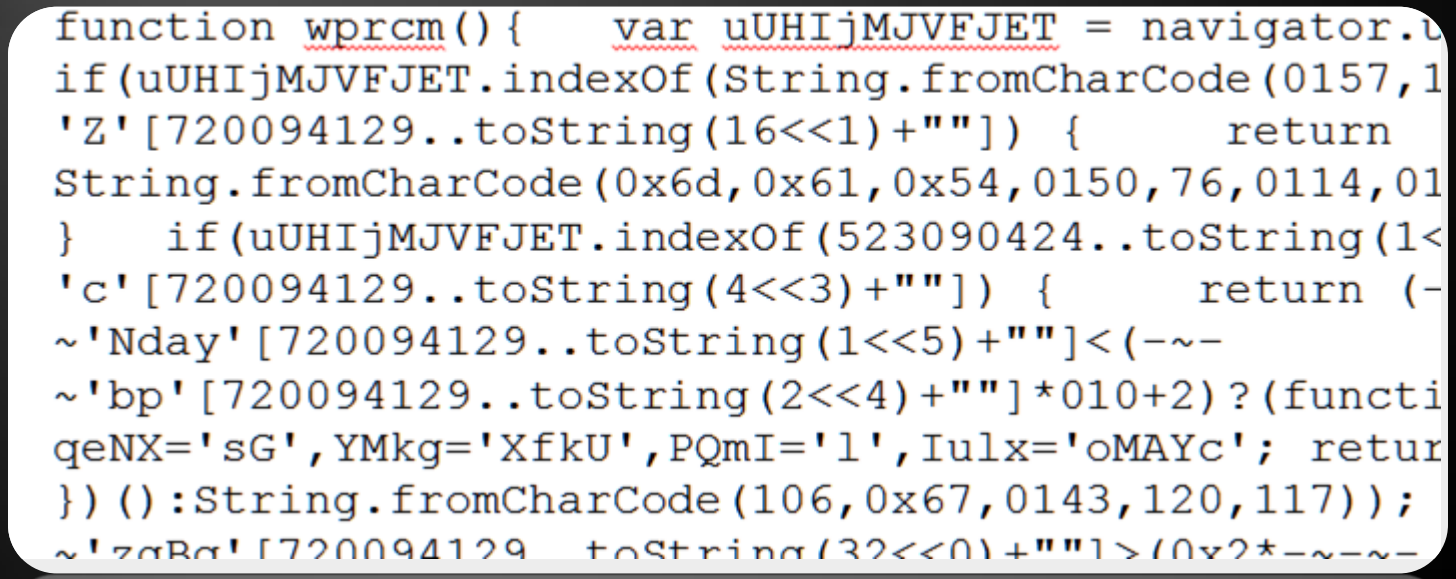
- .NET reflection supports generics
  - Can get the generic parameters at runtime
- Some of the generic reflection APIs:
  - `MethodInfo.IsGenericMethod()`
  - `MethodInfo.GetGenericArguments()`
  - `Type.IsGenericType()`
  - `Type.GetGenericTypeDefinition()`
  - …

Obfuscation

- A way to make IL code less readable
  - Private variables, fields and parameters renames to "nonsense" or unprintable names
  - Other techniques that breaks decompilers but the IL code is still valid
  - Public member names remain unchanged
- Similar to JavaScript minification
- Ultimately, this is an ongoing battle between the obfuscators and the decompilers

Obfuscation

Live Demo with Confuser



Telerik Academy

Real World Examples

Plugin System, Validator, Serialization, Unit Testing

Telerik Academy

Reflection