Programowanie wizualne
.NET System Reflections

Programowanie Wizualne

Paweł Wojciechowski

Instytut Informatyki, Politechniki Poznańskiej

2023

System.Reflection

- każda biblioteka .NET zawiera dokładny opis zdefiniowanych typów.
- `System.Reflection` to biblioteka, która umożliwia uzyskanie danych dostępnych w aplikacji ildasm z poziomu kodu.
- Biblioteka ta, ma następujące klasy:
  - Assembly – klasa abstrakcyjna umożliwiająca wczytanie i przeglądanie pakietu.
  - klasy przechowujące informacje o: metodach (`MethodInfo`), polach (`FieldInfo`), właściwościach (`PropertyInfo`), parametrach (`ParameterInfo`), zdarzeniach (`EventInfo`), konstruktorach (`ConstructorInfo`)
- klasa System.Type umożliwia pobranie informacji o typie.

```
Type t1 = pe.GetType();
Type t2 = typeof(Person);
Type t3 = Type.GetType("Namespace.Klasa");
```

System.Type

- klasa Type ma następujące metody:

  GetConstructors(), GetEvents(), GetFields(), GetInterfaces(), GetMethods(), GetNestedTypes(), GetProperties()

- każda z nich zwraca tablicę obiektów `Info` z przestrzeni nazw `System.Reflection`

```
foreach( var m in t1.GetMethods())
{
    Console.WriteLine(m.Name);
}
MethodInfo mi = t1.GetMethod("ShowInfo");
Console.WriteLine( mi.ReturnType);

Type[] ii = t2.GetInterfaces();
```

System.Type (2)

- wywołanie metody – metoda Invoke(object obj, object[] parameters)

```
Person pe = new Person() { ID = 0, Name = "Grzegorz" };
Type t1 = pe.GetType();

MethodInfo methodInfo = t1.GetMethod("ToString");
Console.WriteLine(methodInfo.ReturnType);
string s = (string) methodInfo.Invoke(pe, null);
Console.WriteLine(s);

methodInfo = t1.GetMethod("StaticMethod");
s = (string)methodInfo.Invoke(null, null);
Console.WriteLine(s);
```

```
public class Person
{
    private int value = 3;
    public string Name { get; set; }
    public int ID { get; set; }

    public int GetValue() => value;

    public override string ToString()
    {
        return Name;
    }

    public static string StaticMethod()
    {
        return "This is static!";
    }
}
```

System.Type (3)

- dostęp do pól prywatnych

```
foreach( var t in t1.GetFields(BindingFlags.NonPublic|BindingFlags.Instance))
{
    Console.WriteLine(t.Name);
}

FieldInfo fieldInfo = t1.GetField("value", BindingFlags.Instance|BindingFlags.NonPublic);

fieldInfo.SetValue(pe, 1);
Console.WriteLine(pe.GetValue());
```

Tworzenie obiektów – System.Reflection

- wczytanie pakietu

```
Assembly a = Assembly.LoadFrom(@"d:\work\programowanie_wizualne\2013\Test.dll");
Assembly a = Assembly.Load("Test,PublicKeyToken=38cacc69c67a0deb");
Assembly a = Assembly.UnsafeLoadFrom(@"d:\work\programowanie_wizualne\2013\Test.dll");
```

- przeglądanie dostępnych typów

```
foreach (var t in a.GetTypes())
```

- tworzenie obiektów – klasa Activator (trzeba znać jakie konstruktory są dostępne)

```
Type ti = a.GetType("Person");
var o = Activator.CreateInstance(ti, new object[]{ ""});
```

- obiekt można również utworzyć pobierając informacje o konstruktorach typu GetConstructor() - jako parametr musimy podać listę argumentów konstruktora

```
ConstructorInfo constructorInfo = ti.GetConstructor(new Type[] { typeof(string) });
```

- …i wywołując go metodą Invoke.

```
var o2 = constructorInfo.Invoke(new object[] { "Nazwa" });
```