



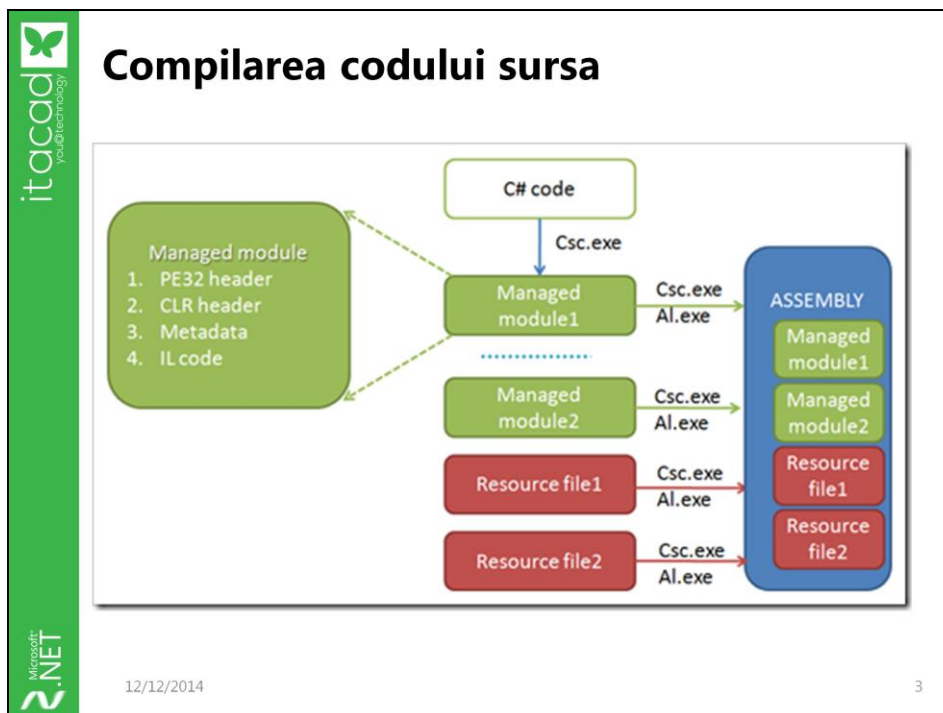
Modulul 9

Reflection

Overview

- Compilarea codului sursă
- Reflection
- Lucrul cu unități de asamblare
- Clasa Type
- Metode, proprietăți, câmpuri
- Crearea unei instanțe
- Invocarea unei metode





❑ Așa cum ați învățat în modulele de 1.Net, platforma .Net oferă mai multe limbaje pentru dezvoltarea aplicațiilor (C#, Visual Basic, Jscript...). Fiecare limbaj are propriul compilator care transformă codul sursă în IL (Intermediate Language) și creează un **Managed Module**.

❑ Un Managed Module este alcătuit din:

- ❖ PE (portable executable) header – indică tipul fișierului (GUI, CUI, dll), precum și un timestamp care indică momentul în care fișierul a fost creat.
- ❖ CLR header – conține informații despre versiunea de CLR necesară, despre metoda Main (punctul de pornire a modului), precum și locația și dimensiunea metadatei, a resurselor etc.
- ❖ Metadata – este alcătuită din mai multe tabele ce conțin informații despre tipurile definite în codul sursă și despre tipurile referite de cod.

Exemple de tabele conținute de metadata:

- TypeDef – conține câte o intrare pentru fiecare tip definit în cod. Sunt păstrate informații precum numele tipului, tipul de bază, flag-uri (public, private, static etc), precum și pointări către metodele conținute.

- MethodDef – conține o intrare pentru fiecare metodă definită în cod. Sunt

reținute date precum numele metodei, flag-uri, semnătura etc.

- PropertyDef – conține o intrare pentru fiecare proprietate definită în modul.

- EventDef – conține câte o intrare pentru fiecare eveniment definit.

- ❖ Intermediate language – codul rezultat în urma compilării codului sursă. Acest cod va fi transformat în instrucțiuni CPU.

❑ Totuși, CLR-ul nu lucrează direct cu un managed module, ci cu un assembly.

Un **assembly** reprezintă o grupare logică ce conține unul sau mai multe managed module și resurse. De asemenea, el este cea mai mică unitate care poate fi utilizată, securizată și versionată.

❑ În continuare, vom prezenta o parte din metadata obținută din compilarea programului *HelloWorld*:

```
using System;
public class Program
{
    public static void Main(String[] args)
    {
        Console.WriteLine("Hello World!");
    }
}
```

TypeDef #1 (02000002)

TypeDefName: Program (02000002)

Flags : [Public] [AutoLayout] [Class] [AnsiClass] [BeforeFieldInit](00100001)

Extends : 01000001 [TypeRef] System.Object

Method #1 (06000001) [ENTRYPOINT]

MethodName: Main (06000001)

Flags : [Public] [Static] [HideBySig] [ReuseSlot] (00000096)

RVA : 0x00002050

ImplFlags : [IL] [Managed] (00000000)

CallCnvtn: [DEFAULT]

ReturnType: Void

1 Arguments

Argument #1: SZArray String

1 Parameters

(1) ParamToken : (08000001) Name : args flags: [none] (00000000)

- ❑ În exemplul de mai sus putem observa că, în urma compilării programului, a fost creată o intrare în tabela TypeDef din metadată. Aceasta conține informații despre tipul **Program** definit în codul nostru: numele tipului (Program), flag-uri (public, class etc), tipul de bază (Object), precum și pointer către metoda Main.

Definiție

- ▶ Reflection – reprezintă procesul prin care o aplicație își poate observa și își poate modifica structura și comportamentul
- ▶ Este abilitate codului de a-și citi metadata pentru a obține informații despre unități de asamblare, module și tipuri, la runtime

12/12/2014


4

❑ În domeniul calculatoarelor, **reflection** poate fi văzut ca un proces prin care programul poate să își analizeze și să își modifice structura și comportamentul.

❑ Un program reflectă asupra sa prin extragerea metadatai din unitatea de asamblare și prin folosirea acesteia fie pentru a informa utilizatorul, fie pentru a-și modifica comportamentul.

❑ Folosind **Reflection**, utilizatorul poate afla detalii despre un obiect, o metodă, poate crea obiecte și poate invoca metode la runtime.

❑ Namespace-ul **System.Reflection** conține clase și interfețe care permit observarea tipurilor, metodelor și câmpurilor încărcate, cu posibilitatea de a crea și invoca dinamic tipuri.



Încărcarea unei unități de asamblare

- ▶ Înainte de a obține informații despre tipurile conținute de un assembly, acesta trebuie să fie încărcat:
 - ▶ `Assembly myAssem = Assembly.LoadFrom(„Employee.dll”);`
 - ▶ `Assembly execAssem = Assembly.GetExecutingAssembly();`

Microsoft .NET

12/12/2014

5

❑ Pentru a putea obține informații despre tipurile conținute într-un assembly, utilizatorul trebuie să îl încarce. Acest lucru se poate realiza folosind metoda statică **LoadFrom** din clasa **Assembly**:

❖ `Assembly assembly = Assembly.LoadFrom(dll_path);`

❑ Un obiect **Assembly** este folosit pentru a explora metadata, pentru a descoperi tipurile conținute și pentru a crea instanțe ale acelor tipuri.

❑ Pentru a obține obiectul de tip **Assembly** care conține codul ce se află în execuție, se folosește metoda **GetExecutingAssembly()**

❖ `Assembly assembly = Assembly.GetExecutingAssembly();`

❑ De asemenea, această clasă oferă o metodă care poate fi utilizată pentru a obține informații despre numele unui assembly:

❖ `GetName();`

❑ Următorul exemplu arată cum se pot obține numele, versiunea și punctul de intrare ale unității de asamblare curente:

```
using System.Reflection;
```

```
using System.Security.Permissions;
```

```
public class Example
```

```
{
```

```
    public static void Main()
```

```
    {
```

```
        // Obținerea assembly-ului care se află în execuție
```

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

```
        Console.WriteLine("Assembly Full Name:");
```

```
        Console.WriteLine(assem.FullName);
```

```
        // Obținerea obiectului AssemblyName, care conține informații despre  
        nume și despre versiune
```

```
        AssemblyName assemName = assem.GetName();
```

```
        Console.WriteLine("\nName: {0}", assemName.Name);
```

```
        Console.WriteLine("Version: {0}.{1}", assemName.Version.Major,  
        assemName.Version.Minor);
```

```
        Console.WriteLine("\nAssembly entry point:");
```



```
Console.WriteLine(assem.EntryPoint);  
  
}  
  
}
```

Tipurile conținute de assembly

- ▶ Pentru a obține toate tipurile din assembly:
 - ▶ `Type[] myTypes = myAssem.GetTypes();`
- ▶ Pentru a obține un anumit tip:
 - ▶ `Type myType = myAssem.GetType("Company.Employee");`

12/12/2014

6

❑ Există două metode de a obține informații despre tipuri:

❖ **GetTypes()** – returnează un vector de **System.Types**

❖ **GetType("Company.Employee")** - returnează un obiect **System.Type**, ce va conține informații despre clasa dată ca parametru (**Company.Employee**). Observați ca numele clasei conține și namespace-ul din care face parte.

namespace Company

```
{  
    public class Employee  
    {  
        private string _name;  
        private string _id, _age;  
        public string Name { get { return _name;} set { _name = value;} }  
        public int Id { ....}  
        public int Age { ...}
```

```
        public double GetSalary() { ... }  
    }  
}
```

```
using System;
```

```
using System.Reflection;
```

```
public class Example
```

```
{  
    public static void Main()  
    {  
        Assembly assem = Assembly.GetExecutingAssembly();  
        Type t = assem.GetType("Company.Employee");  
        if (t != null)  
        {  
            Console.WriteLine("We have found type {0}:", type.FullName);  
        }  
    }  
}
```

Type

- ▶ Pentru a obține informații despre un anumit tip, se folosesc următoarele proprietăți:
 - ▶ `Type myType = myAssem.GetType(„Employee”);`
 - ▶ `myType.Name`
 - ▶ `myType.FullName`
 - ▶ `myType.Namespace`
 - ▶ `myType.IsClass`
 - ▶ `myType.IsInterface`
 - ▶ `myType.IsEnum`
 - ▶ `myType.IsAbstract`
 - ▶ `myType.IsPublic`

12/12/2014

7

□ Odată obținut obiectul `Type`, acesta permite utilizatorului să afle informații despre acest tip:

- ❖ **Name**
- ❖ **FullName**
- ❖ **Assembly**
- ❖ **Module**
- ❖ **IsClass**
- ❖ **IsInterface**
- ❖ **IsEnum**
- ❖ **IsAbstract**
- ❖ **IsPublic**
- ❖ **IsSealed**

```
using System;
```

```
using System.Collections.Generic;
```

```
public class Example
```

```
{
```

```
    public static void Main()
```

```
    {
```

```
        Assembly myAssem = Assembly.LoadFrom(„Employee.dll”);
```

```
        Type myType = myAssem.GetType(„Employee”);
```

```
        Console.WriteLine(“FullName: {0}”, myType.FullName);
```

```
        Console.WriteLine(“Name: {0}”, myType.Name);
```

```
    }
```

```
}
```

Metode, proprietăți, câmpuri

▶ Metode

- ▶ `MethodInfo[] myMethods = myType.GetMethods();`
- ▶ `MethodInfo myMeth = myType.GetMethod("GetSalary");`

▶ Proprietăți

- ▶ `PropertyInfo[] myProps = myType.GetProperties();`
- ▶ `PropertyInfo myProp = myType.GetProperty("Name");`

▶ Câmpuri



- ▶ `FieldInfo[] myFields = myType.GetFields();`
- ▶ `FieldInfo myField = myType.GetField("m_name");`

12/12/2014

8

❑ Fiecare tip poate conține câmpuri, proprietăți și metode. Aceste informații despre tipuri pot fi obținute prin folosirea următoarelor metode:

- ❖ `GetMembers()` – returnează toți membrii publici ai tipului curent
- ❖ `GetMember(string name)` – caută membrul public cu numele dorit
- ❖ `GetProperties()` – returnează toate proprietățile publice ale tipului curent
- ❖ `GetProperty(string name)` – caută și returnează proprietatea cu numele specificat
- ❖ `GetFields()` – returnează toate câmpurile publice ale tipului curent
- ❖ `GetField(string name)` – returnează câmpul cu numele specificat
- ❖ `GetEvents()` – returnează evenimentele publice ale tipului
- ❖ `GetEvent(string name)` – returnează evenimentul specificat



MethodInfo

- ▶ Pentru a obține informații despre un obiect **MethodInfo**, sunt disponibile următoarele proprietăți și metode:
 - ▶ Name
 - ▶ IsPrivate / IsPublic / IsStatic
 - ▶ IsConstructor
 - ▶ ReturnType
 - ▶ GetParameters()
 - ▶ Invoke()

12/12/2014 9

- ❑ Metodele **GetMethods()** și **GetMethod()** întorc un vector de obiecte/ un obiect **MethodInfo**. Acesta poate fi folosit pentru a obține informații despre metoda conținută de tip-ul curent.

```
using System;
```

```
using System.Reflection;
```

```
public class Example
```

```
{
```

```
    public static void Main()
```

```
    {
```

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

```
        Type t = assem.GetType("Company.Employee");
```

```
        MethodInfo getSalary = t.GetMethod("GetSalary");
```

```
Console.WriteLine("Return type: {0} ", getSalary.ReturnType);  
}  
}
```


PropertyInfo

- ▶ Pentru a obține informații despre un obiect PropertyInfo, sunt disponibile următoarele proprietăți și metode:
 - ▶ Name
 - ▶ CanRead / CanWrite
 - ▶ PropertyType
 - ▶ GetValue()
 - ▶ SetValue()

12/12/2014

10

❑ Exemplu de utilizare a unui obiect PropertyInfo:

```
using System;
```

```
using System.Reflection;
```

```
public class Example
```

```
{
```

```
    public static void Main()
```

```
    {
```

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

```
        Type t = assem.GetType("Company.Employee");
```

```
        object employee = Activator.CreateInstance(t);
```

```
        PropertyInfo id = t.GetProperty("Id");
```

```
        Console.WriteLine("Id: {0} ", id.GetValue(employee));
```

}

}

FieldInfo

- ▶ Pentru a obține informații despre un obiect FieldInfo, sunt disponibile următoarele proprietăți și metode:
 - ▶ Name
 - ▶ FieldType
 - ▶ IsPublic / IsPrivate / IsStatic
 - ▶ GetValue()
 - ▶ SetValue()

12/12/2014

11

❑ Exemplu de utilizare a unui obiect FieldInfo:

```
using System;
```

```
using System.Reflection;
```

```
public class Example
```

```
{
```

```
    public static void Main()
```

```
    {
```

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

```
        Type t = assem.GetType("Company.Employee");
```

```
        object employee = Activator.CreateInstance(t);
```

```
        FieldInfo id = t.GetField("_id");
```

```
        if(id != null)
```

```
        Console.WriteLine("Id: {0} ", id.GetValue(employee));  
    else  
        Console.WriteLine("Nu a fost găsit câmpul id");  
    }  
}
```

☐ De ce credeți că nu a fost găsit câmpul id?

Crearea unei instanțe

Exemplu instanțiere obiect:

```
Assembly a=Assembly.LoadFrom("employee.dll");  
Type t=a.GetType("Company.Employee");  
Employee obj=(Employee) Activator.CreateInstance(t);
```

12/12/2014

12

- ❑ În slide-urile anterioare am lucrat cu tipuri și obiecte deja instanțiate și am arătat cum se poate inspecta structura acestora. În continuare, vom folosi reflection pentru a instanția obiecte la runtime, cunoscând doar numele clasei pe care o vom instanția.
- ❑ Există mai multe metode prin care se poate instanția o clasă, una dintre acestea fiind folosirea clasei statice **Activator**.
- ❑ Activator este o clasă ce conține metode folosite pentru a crea instanțe de obiecte. În acest modul, noi vom folosi metoda **CreateInstance(Type t)**, ce va folosi constructorul default (dacă acesta nu există, se va arunca o excepție). Metoda va returna o referință a obiectului.

```
Assembly a=Assembly.LoadFrom("employee.dll");  
Type t=a.GetType("Company.Employee");  
Employee obj=(Employee) Activator.CreateInstance(t);
```

Invocarea unei metode

Exemplu invocare metodă:

```
Assembly a=Assembly.LoadFrom("employee.dll");  
Type t=a.GetType("Company.Employee");  
Employee emp=(Employee) Activator.CreateInstance(t);  
MethodInfo getSalary=t.GetMethod(„GetSalary”);  
object result = getSalary.Invoke(emp, null);
```

12/12/2014

13

- ❑ Am învățat cum să obținem un obiect **MethodInfo**, folosind numele metodei dorite. În continuare, vom chema această metodă și îi reținem rezultatul.
- ❑ Pentru aceasta, vom folosi metoda **Invoke** din **MethodInfo**:
 - ❖ `object result = getSalary.Invoke(emp, null);`
- ❑ Aceasta primește ca parametri, instanța obiectului pentru care se va apela metoda și un vector cu parametri pe care îi primește metoda. În exemplul curent, metoda `GetSalary` nu primește nici un parametru (`null`).
- ❑ Exemplu de metodă care primește parametri:
`setSalary.Invoke (emp, new object[] {100});`



Instanțiere folosind constructorul

Exemplu:

```
Assembly a=Assembly.LoadFrom(„Employee.dll”);  
Type t=a.GetType("Company.Employee");  
ConstructorInfo ctor = t.GetConstructor(Type.EmptyTypes);  
object instance = ctor.Invoke(null);
```



12/12/2014

14

- ☐ O altă modalitate de a instanția un obiect este aceea de a-i folosi constructorul. Pentru aceasta, se folosește metoda **GetCostructor(Type[] types)**.
- ☐ Această metodă primește ca parametru un vector de obiecte Type, reprezentând numărul, ordinea și tipul parametrilor ale constructorului dorit.
- ☐ Dacă acel constructor nu primește nici un parametru, se folosește **Type.EmptyTypes**.
- ☐ Exemplu:

```
namespace Company
```

```
{
```

```
    public class Employee
```

```
{  
    public Employee() {...}  
    public Employee(string name) {...}  
}  
}
```

```
using System;
```

```
using System.Reflection;
```

```
public class Example
```

```
{  
    public static void Main()  
    {  
        Assembly assem = Assembly.GetExecutingAssembly();  
        Type t = assem.GetType("Company.Employee");  
  
        // Obținem constructorul fara parametri  
        ConstructorInfo ctor1 = t.GetConstructor(Type.EmptyTypes);  
        object instance1 = ctor1.Invoke(null);  
  
        // Obținem constructorul cu parametri  
        Type[] types = new Type[1];  
        types[0] = typeof(string);
```



```
ConstructorInfo ctor2 = t.GetConstructor(Type.EmptyTypes);  
    object instance2 = ctor2.Invoke(new object[0]);  
}  
}
```

Sumar

- Compilarea codului sursă
- Reflection
- Lucrul cu unități de asamblare
- Clasa Type
- Metode, proprietăți, câmpuri
- Crearea unei instanțe
- Invocarea unei metode

