Язык C#

Рефлексия типов и программирование с использованием атрибутов

Лекция #6

Что такое рефлексия типов

- В .NET рефлексия типов – это процесс обнаружения типов во время работы программы
- Класс Type из System
- Пространство имен System.Reflection

Члены класса Type

IsAbstract, IsArray, IsClass, IsSealded, IsInterface, IsValueType	Позволяют определить основные характеристики конкретного типа в программе
GetConstructors(), GetEvents(), GetFileds(), GetInterfaces, GetProperties, GetMembers()	Возвращают массив с набором интересующих пользователя элементов
FindMembers()	Возвращает массив типов MemeberInfo
GetType	Возвращает объект Type по строковому имени
InvokeMember()	Используется для позднего связывания

Получение объекта класса Type

- Foo theFoo = new Foo();
  Type t = theFoo.GetType();

- Type t = null;
  t = Type.GetType("Foo");

- Type t = typeof(Foo)

Тестовый класс Foo

```
// Мы сможем получить разнообразную информацию об этом классе во время выполнения
namespace TheType
{
// Два интерфейса
public interface IFaceOne { void MethodA(); }
public interface IFaceTwo { void MethodB(); }

public class Foo: IFaceOne, IFaceTwo // Класс Foo поддерживает эти два интерфейса
{
     public int myIntField;        // Поля
     public string myStringField;

     public void myMethod(int p1, string p2) {...} // Метод

     public int MyProp   // Свойство
     {    get { return myIntField; }
          set { myIntField = value; }
     }

     // Методы интерфейсов IFaceOne и IFaceTwo
     public void MethodA() {...}
     public void MethodB() {...}
 }
}
```

Получаем информацию о полях и методах

```
public static void ListMethods (Foo f)
{
 Console.WriteLine("***** Methods of Foo *****");
 Type t = f.GetType();
 MethodInfo[] mi = t.GetMethods();
 foreach(MethodInfo m in mi)
     Console.WriteLine("Method: {0}", m.Name);
 Console.WriteLine("*************************\n");
}

public static void ListFields(Foo f)
{
 Console.WriteLine("***** Fields of Foo *****");
 Type t = f.GetType();
 FieldInfo[] fi = t.GetFields();
 foreach(FieldInfo field in fi)
     Console.WriteLine("Field: {0}", field.Name);
 Console.WriteLine("*************************\n");
}
```

Выводим разную информацию о классе

```
// Выводим разную информацию о Foo
public static void ListVariosStats(Foo f)
{
   Console.WriteLine("***** Various stats about Foo *****");
   Type t = f.GetType();

   Console.WriteLine("Full name is: {0}", t.FullName);
   Console.WriteLine("Base is: {0}", t.BaseType);
   Console.WriteLine("Is it abstract? {0}", t.IsAbstract);
   Console.WriteLine("Is it a COM object? {0}", t.IsCOMObject);
   Console.WriteLine("Is it sealed? {0}", t.IsSealed);
   Console.WriteLine("Is it a class? {0}", t.IsClass);

   Console.WriteLine("**************************\n");
}
```

Выводим список свойств

```
// Выводим список всех свойств
public static void ListPorps(Foo f)
{
   Console.WriteLine("***** Properties of Foo *****");

   Type t = f.GetType();
   PropertyInfo[] pi = t.GetProperties();
   foreach(PropertyInfo prop in pi)
    Console.WriteLine("Prop: {0}", prop.Name);

   Console.WriteLine("*****************************\n");
}
```

Выводим список интерфейсов

```
// Выводим список всех интерфейсов, поддерживаемых Foo
public static void ListInterfaces(Foo f)
{
   Console.WriteLine("***** Interfaces of Foo *****");

   Type t = f.GetType();
   Type[] ifaces = t.GetInterfaces();
   foreach(Type i in ifaces)
    Console.WriteLine("Interface: {0}", i.Name);

   Console.WriteLine("*************************\n");
}
```

Типы пространства имен System.Reflection

Assembly	Класс содержит методы для загрузки и изучения сборки, а также выполнения с ней различных операций
AssemblyName	Класс позволяет получить информацию о версии, естественном языке и т.п.
EventInfo	Хранит информацию о событии
FieldInfo	Хранит информацию о поле
MemberInfo	Абстрактный базовый класс для всех *Info
MethodInfo	Хранит информацию о методе
Module	Позволяет обратиться к модулю в многофайловой сборке
ParameterInfo	Хранит информацию о параметре
PropertyInfo	Хранит информацию о свойстве

Загрузка сборки

```
namespace CarReflector  // Получаем информацию о сборке CarLibrary
{
  using System;
  using System.Reflection;
  using System.IO;    // Нужно для использования FileNotFoundException

public class CarReflector
{
   public static int Main(string[] args)
   {
   // Используем метод Assembly.Load() для загрузки сборки
   Assembly a = null;

   try
    {  a=Assembly.Load("CarLibrary");
    }

   catch (FileNotFoundException e)
    {Console.WriteLine(e.Message);}
        return 0;
    }
}
}
```

a=Assembly.Load(@“CarLibrary, Ver=1.0.454.30104, SN=null, Loc=“”“);

Вывод информации о типах в сборке

```
public class CarReflector
{
  public static int Main(string[] args)
  {
   Assembly a = null;
   try       { a=Assembly.Load("CarLibrary"); }
   catch(FileNotFoundException e) {Console.WriteLine(e.Message);}

   ListAllTypes(a);
   return 0;
  }

  // Выводим информацию о всех типах в сборке
  private static void ListllTypes(Assembly a)
  {
    Console.WriteLine("Listing all types in {0}", a.FullName);
    Type[] types = a.GetTypes();
    foreach(Type t in types)
       Console.WriteLine("Type: {0}", t);
   }
}
```

Вывод информации о членах класса

```
private static void ListAllMembers(Assembly a)
{
   Type miniVan = a.GetType("CarLibrary.MiniVan");

   Console.WriteLine("Listing all members for {0}", miniVan);

   MemberInfo[] mi = miniVan.GetMembers();
   foreach(MemberInfo m in mi)
    Console.WriteLine("Type {0}: {1} ",
           m.MemberType.ToString(), m);
}
```

Вывод информации о параметрах метода

```
private static void GetParams(Assembly a)
{
   Type miniVan = a.GetType("CarLibrary.MiniVan");
   MethodInfo mi = miniVan.GetMethod("TurnOnRadio");
   Console.WriteLine("Here are the params for {0}", mi.Name);

   // Show number of params.
   ParameterInfo[] myParams = mi.GetParameters();
   Console.WriteLine("Method has {0} params",myParams.Length);

   // Show info about param.
   foreach(ParameterInfo pi in myParams)
   {
   Console.WriteLine("Param name: {0}",pi.Name);
   Console.WriteLine("Position in method: {0}",pi.Position);
   Console.WriteLine("Param type: {0}",pi.ParameterType);
   }
}
```

Позднее связывание и класс System.Activator

```
// Создаем объект выбранного нами типа "на лету"
public class LateBind
{
   public static int Main(string[] args)
   {
    // Используем класс Assembly для загрузки сборки
    Assembly a = null;
    try
    {
        a = Assembly.Load("CarLibrary");
    }
    catch(FileNotFoundException e)
    {Console.WriteLine(e.Message);}

    // Получаем объект Type для класса MiniVan
    Type miniVan = a.GetType("CarLibrary.MiniVan");

    // Создаем объект класса MiniVan "на лету"
    object obj = Activator.CreateInstance(miniVan);
   }
}
```

Использование позднего связывания

```
public static int Main(string[] args)
{
   // Загружаем CarLibrary при помощи класса Assembly
   ...

   // Получаем объект типа Type
   Type miniVan = a.GetType("CarLibrary.MiniVan");

   // Создаем объект класса MiniVan "на лету"
   object obj = Activator.CreateInstance(miniVan);

   // Получаем объект класса MethodInfo для метода TurboBoost()
   MethodInfo mi = miniVan.GetMethod("TurboBoost");

   // Вызываем метод (передаем null вместо параметров)
   mi.invoke(obj, null);
   return 0;
}
```

Вызов метода с параметрами

```
object[] paramArray = new object[2];
paramArray[0]="Fred";
paramArray[1]= 4;

MethodInfo mi = miniVan.GetMethod("TellChildToBeQuiet");

// Вызываем метод
mi.invoke(obj, paramArray);
```

Динамические сборки

- Динамические сборки создаются «на лету»
- Ее можно сохранить на диск
- Пространство имен System.Reflection.Emit
- Класс AssemblyBuilder
- Создание кода IL «на лету» в процессе выполнения программы

Подрбнее см. Э.Троелсен «C# и платформа .NET»

Атрибуты в .NET

- Атрибуты – это аннотации, который могут быть применены к типу (интерфейсу, структуре и т.п.), члену класса (полю, свойству, методу)
- Многие атрибуты предназначены для «хитрых» целей: отладки, взаимодействию с COM

Некоторые встроенные атрибуты

CLSCompilant	Определяет совместимость всех типов сборки с Common Language Specification
DllImport	Для вызова традиционных файлов dll
StructLayout	Для определения внутреннего представления структуры
Serializable	Помечает класс или структуру как *сериализуемые* (доступные для сохранения на диск и восстановления с него)
NonSerialized	Помечает класс или структуру как *несериализуемые*

Работа с существующими атрибутами

```
// Этот класс можно сохранять на диске
[Serializable]
public class Motorcycle
{
  bool hasRadioSystem;
  bool hasHeadSet;
  bool hasBar;

  // Однако незачем утруждать себя сохранением этого поля
  [NonSerialized]
  float weightOfCurrentPassangers;
}
```

Создание пользовательских атрибутов

```
public class VehicleDescriptionAttribute : System.Attribute
{
    private string description;
    public string Desc
   {
    get { return description; }
    set { description = value; }
   }

    public VehicleDescriptionAttribute() {}
    public VehicleDescriptionAttribute(string desc)
        { description = desc;}
}
```

Применение пользовательских атрибутов

```
[VehicleDescriptionAttribute("A very long, slow but feature rich auto")]
public class WinExample
{
        public WinExample()
        {
        }
}
```

```
[VehicleDescription ("A very long, slow but feature rich auto")]
public class WinExample
  {
        public WinExample()
        {
        }

  }
```

Ограничение использования атрибута

```
// Это перечисление позволяет определить, к чему можно будет
// применять пользовательский атрибут
public enum AttributeTargets
{
    All,
    Assembly,
    Class,
    Constructor,
    Delegate,
    Enum,
    Event,
    Field,
    Interface,
    Method,
    Module,
    Parameter,
    Property,
    ReturnValue,
    Struct,
}
```

Ограничение использования атрибута

```
namespace CustomAtt
{
using System;

[AttributeUsage(AttributeTargets.Class | AttributeTargets.Struct)]

public class VehicleDescriptionAttribute : System.Attribute
{
    private string description;
    public string Desc
   {
    get { return description; }
    set { description = value; }
   }

    public VehicleDescriptionAttribute() {}
    public VehicleDescriptionAttribute(string desc)
        { description = desc;}
 }
}
```

Аттрибуты уровня сборки и модуля

```
// Гарантируем совместимость с CLS
using System;

[assembly:System.ClsCompliantAttribute(true)]

namespace MyAttributes
{
[VehicleDescription ("A very long, slow but feature rich auto")]
public class WinExample
{
    public WinExample(){}

            // public ulong notCompilant

}
}
```

Файл AssemblyInfo.cs

AssemblyCompanyAttribute	Информация о компании
AssemblyDescriptionAttribute	Дружественное текстовое описание сборки
AssemblyProcessorAttribute	Информация о процессоре, для которого рассчитана данная сборка
AssemblyVersionAttribute	Определяет номер версии сборки

Аттрибуты уровня сборки и модуля

```
// Рефлексия для пользовательских атрибутов
public class AttReader
{
   public static int Main(string[] args)
   {
    // Получаем объект класса Type для Winnebago
    Type t = typeof(WinExample);

    // Получаем все атрибуты данной сборки
    object[] customAtts = t.GetCustomAttributes(false);

    // Выводим информацию о каждом атрибуте
    foreach(VehicleDescriptionAttribute v in customAtts)
            Console.WriteLine(v.Desc);

    return 0;
   }
}
```