А.В. Макаров, С.Ю. Скоробогатов, А.М. Чеповский

Учебный курс “CIL и системное программирование в Microsoft .NET”

Лекция 6.

Библиотеки для создания метаинструментов

Введение

- Метаинструменты для платформы .NET – это программы, рассматривающие сборки .NET в качестве объектов анализа, генерации или преобразования
- Сборка .NET – это совокупность метаданных, CIL-кода и, возможно, ресурсов. Библиотеки, необходимые для создания метаинструментов, должны обеспечивать чтение и генерацию этой информации
- В составе .NET Framework SDK поставляются две библиотеки, частично решающие эти задачи:
  - Metadata Unmanaged API
  - Reflection API

6.1. Metadata Unmanaged API

- Metadata Unmanaged API осуществляет импорт и генерацию метаданных
- Оно предназначено для использования в компиляторах и загрузчиках, для которых требуется высокая скорость доступа к метаданным и работа с метаданными на низком уровне
- Навигация через иерархию метаданных осуществляется через токены метаданных
- Metadata Unmanaged API предоставляет прямой доступ к таблицам метаданных, позволяет сливать несколько сборок в одну, но не содержит явных средств для работы с CIL-кодом

Взаимодействие с Metadata Unmanaged API

- Для того чтобы использовать Metadata Unmanaged API из программы, написанной на Visual C++, необходимо включить в программу следующие строки:

```
#include <corhlpr.h>

#pragma comment(lib, "format.lib")
```

- Взаимодействие с Metadata Unmanaged API осуществляется через набор COM-интерфейсов:

Интерфейс	Описание
IMetadataDispenserEx	Позволяет загружать образ исполняемого или объектного файла в память или создавать новый образ.
IMetadataImport	Предназначен для чтения метаданных.
IMetadataEmit	Предназначен для генерации метаданных.

Начало работы с Metadata Unmanaged API

- Работа с Metadata Unmanaged API начинается с инициализации системы COM и получения указателя на интерфейс IMetadataDispenserEx:

```
CoInitialize(NULL);

IMetaDataDispenser *dispenser;
HRESULT h = CoCreateInstance(
    CLSID_CorMetaDataDispenser,
    NULL,
    CLSCTX_INPROC_SERVER,
    IID_IMetaDataDispenserEx,
    (void **)&dispenser
    );

if (h)
    printf("Error");
```

Чтение метаданных

- Вызов метода OpenScope интерфейса IMetadataDispenserEx для чтения метаданных из сборки:

```
IMetaDataImport *mdimp;
HRESULT h = dispenser->OpenScope(
    L"test.exe",
    0,
    IID_IMetaDataImport,
    (IUnknown**)&mdimp
    );

if (h)
    printf("Error");
```

Завершение работы с Metadata Unmanaged API

- При завершении работы с Metadata Unmanaged API необходимо освободить указатели на полученные интерфейсы:

```
mdimp->Release();
dispenser->Release();
```

Основные группы методов интерфейса IMetadataImport

- Методы EnumXXX возвращают массивы токенов, описывающих определенную категорию элементов метаданных.
- Методы FindXXX предназначены для поиска элементов метаданных, удовлетворяющих некоторым критериям.
- Методы GetXXX используются для получения свойств элементов метаданных.

Пример: получение массива токенов, соответствующих типам, объявленным в сборке

```
mdTypeDef tmp;
mdTypeDef *tokens;
HCORENUM Enum = 0;
unsigned long tokenCount;

mdimp->EnumTypeDefs(&Enum,&tmp,1,&tokenCount);
mdimp->CountEnum(Enum,&tokenCount);

if (tokenCount > 0)
{
    tokens = new mdTypeDef [tokenCount];
    tokens[0] = tmp;
    if (tokenCount > 1)
        mdimp->EnumTypeDefs(
            &Enum,tokens+1,tokenCount-1,
            &tokenCount
            );
    mdimp->CloseEnum(Enum);
}
```

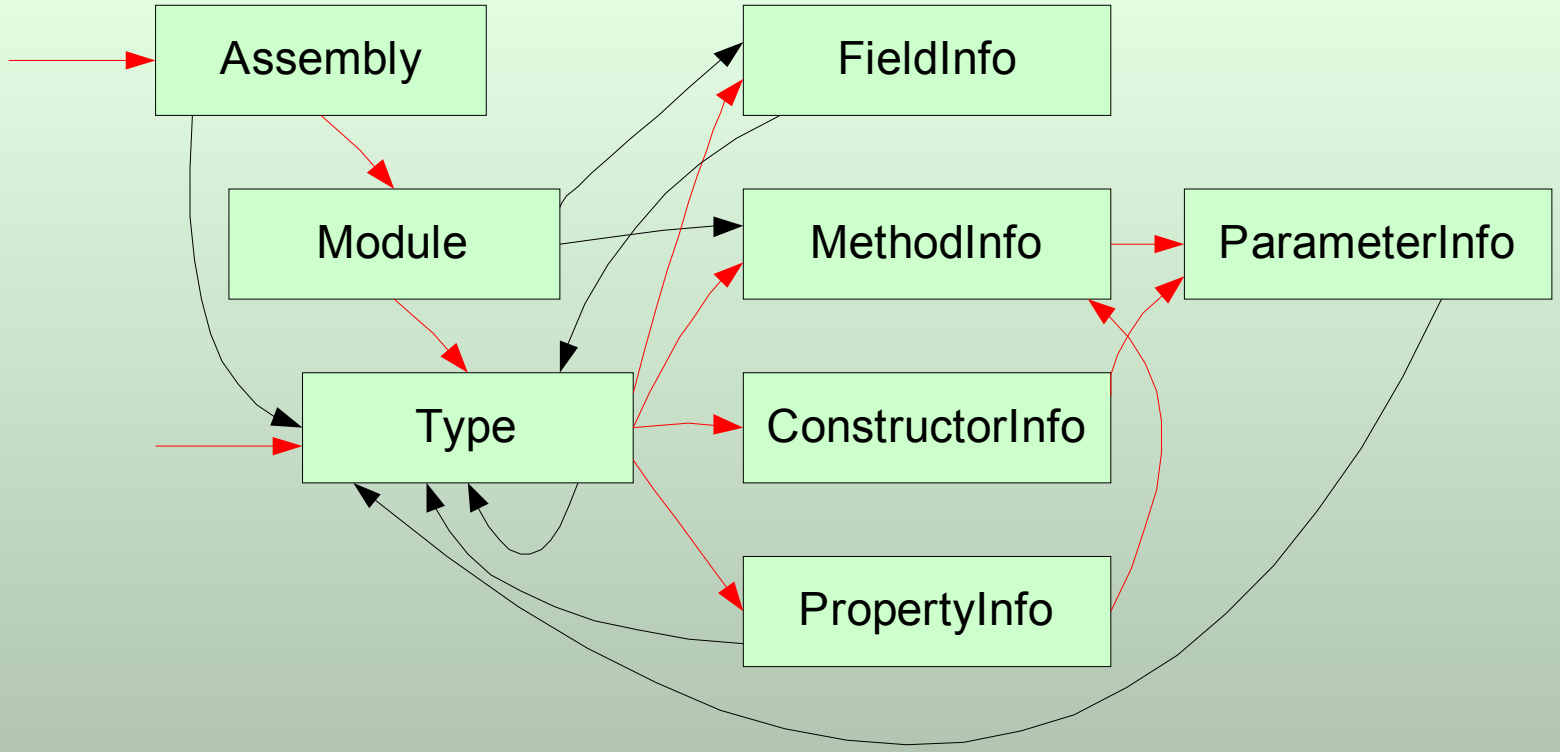
Интерфейс IMetadataEmit

- Генерация метаданных осуществляется через методы интерфейса IMetadataEmit, которые можно условно разделить на две группы:
  - Методы DefineXXX добавляют новые элементы метаданных
  - Методы SetXXX устанавливают свойства элементов метаданных
- Сгенерированные метаданные могут быть сохранены на диске при помощи метода Save

6.2. Reflection API

- Библиотека рефлексии содержит классы для работы с метаданными на высоком уровне
- Пространства имен, в которых расположены классы библиотеки рефлексии:
  - System.Reflection
  - System.Reflection.Emit
- Библиотека рефлексии соответствует спецификации CLS, поэтому использующий ее метаинструмент может быть написан на любом языке платформы .NET, который является потребителем CLS-библиотек

6.2.1. Чтение метаданных через рефлексию



- Информацию о любом элементе метаданных можно прочитать из свойств соответствующего ему объекта рефлексии

Доступ к сборкам (получение объекта класса Assembly)

- Доступ к текущей сборке

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

- Доступ к сборке, содержащей указанный тип

```
Assembly assembly = Assembly.GetAssembly
  (typeof(int));
```

- Загрузка сборки

```
Assembly assembly = Assembly.LoadFrom
  ("test.exe");
```

Доступ к модулям (получение объекта класса Module)

- Доступ к модулю по имени

```
Module module = assembly.GetModule
   ("somemodule.exe");
```

- Получение массива модулей сборки

```
Module[] modules = assembly.GetModules(false);
```

Доступ к глобальным полям

- Доступ к глобальному полю по имени

```
FieldInfo field =
    module.GetField("FieldName");
```

- Получение массива глобальных полей

```
FieldInfo[] fields =
    module.GetFields();
```

Доступ к глобальным методам

- Доступ к глобальному методу по имени

```
MethodInfo method =
    module.GetMethod("MethodName");
```

- Получение массива глобальных методов

```
MethodInfo[] methods =
    module.GetMethods();
```

Пример: вывод на экран сигнатур всех глобальных методов сборки

```
Assembly assembly = Assembly.LoadFrom("test.exe");
Module[] modules = assembly.GetModules(false);

foreach (Module mod in modules)
{
    MethodInfo[] methods = mod.GetMethods();

    foreach (MethodInfo met in methods)
        Console.WriteLine(met);
}
```

Доступ к типам (получение объекта класса Type)

- Использование метода GetType

```
object o = new String("qwerty");
Type t = o.GetType();
Console.WriteLine(t);
```

- Использование оператора typeof

```
Type t = typeof(string[]);
```

- Доступ к типу по имени
  - `Type t = Type.GetType("System.Char");`
  - `Type t = module.GetType("Class1");`
  - `Type t = assembly.GetType("Class1");`
- Получение массива типов через вызов метода GetTypes () объекта Assembly или Module

Пример: вывод на экран списка типов из сборки

```
Assembly assembly = Assembly.LoadFrom("test.exe");
Type[] types = assembly.GetTypes();

foreach (Type t in types)
    Console.WriteLine(t);
```

Доступ к содержимому типов через методы класса Type

- Доступ по имени: GetField, GetConstructor, GetMethod, GetNestedType и GetProperty
- Получение массивов: GetFields, GetConstructors, GetMethods, GetNestedTypes и GetProperties

Пример: вывод на экран списка типов сборки вместе с их методами

```
Assembly assembly = Assembly.LoadFrom("test.exe");
Type[] types = assembly.GetTypes();

foreach (Type t in types)
{
    Console.WriteLine(""+t+":");
    MethodInfo[] methods = t.GetMethods();

    foreach (MethodInfo m in methods)
        Console.WriteLine("    "+m);
}
```

Доступ к параметрам методов и конструкторов

- Получение массива параметров метода или конструктора осуществляется путем вызова метода GetParameters объекта MethodInfo или ConstructorInfo

```
ParameterInfo[] parms =
    method.GetParameters();
```

6.2.2. Управление объектами

- Кроме чтения метаданных библиотека рефлексии позволяет создавать экземпляры типов, входящих в обрабатываемую сборку, вызывать методы этих типов, читать и изменять значения полей
- Рассмотрим класс TestClass:

```
class TestClass
{
    private int val;
    public TestClass() { val = 7; }
    protected void print()
        { Console.WriteLine(val); }
}
```

Пример: вызов методов через рефлексию

```
static void CallMethodDemo()
{
    TestClass a = new TestClass();

    BindingFlags flags = (BindingFlags)
        (BindingFlags.NonPublic |
         BindingFlags.Instance);
    MethodInfo printer =
        typeof(TestClass).GetMethod("print",flags);

    printer.Invoke(a, null);
}
```

Пример: создание объекта через рефлексию путем вызова конструктора

```
static void CallMethodDemo2()
{
    Type t = typeof(TestClass);
    BindingFlags flags = (BindingFlags)
        (BindingFlags.NonPublic |
         BindingFlags.Instance);
    ConstructorInfo ctor =
        t.GetConstructor(new Type[0]);
    MethodInfo printer =
        t.GetMethod("print",flags);

    object a = ctor.Invoke(null);
    printer.Invoke(a, null);
}
```

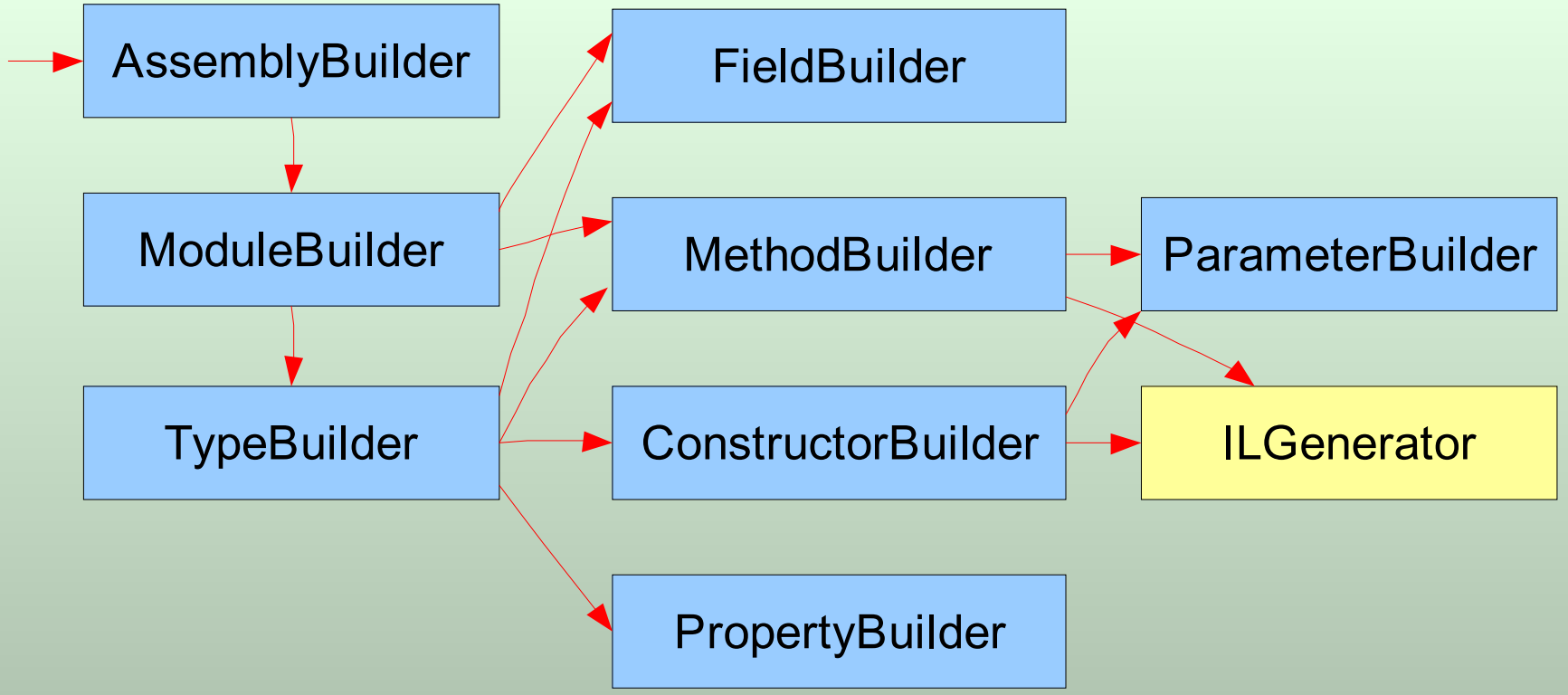
Пример: доступ к полям объекта

```
static void GetFieldDemo()
{
    TestClass a = new TestClass();

    BindingFlags flags = (BindingFlags)
        (BindingFlags.NonPublic |
         BindingFlags.Instance);
    FieldInfo val =
        typeof(TestClass).GetField("val",flags);

    Console.WriteLine(val.GetValue(a));
}
```

6.2.3. Генерация метаданных и CIL-кода



- В отличие от Metadata Unmanaged API, библиотека рефлексии содержит средства для генерации CIL-кода. Для этого предназначен класс ILGenerator

Пример: генерация сборки hello.exe (часть 1)

```
using System;
using System.Threading;
using System.Reflection;
using System.Reflection.Emit;
class HelloGenerator
{
    static void Main(string[] args)
    {
        AppDomain appDomain = Thread.GetDomain();
        AssemblyName assemblyName = new AssemblyName();
        assemblyName.Name = "hello.exe";
        AssemblyBuilder assembly =
            appDomain.DefineDynamicAssembly(
                assemblyName,
                AssemblyBuilderAccess.RunAndSave
                );
        ModuleBuilder module =
            assembly.DefineDynamicModule(
                "hello.exe", "hello.exe"
                );
```

Пример: генерация сборки hello.exe (часть 2)

```
        MethodBuilder mainMethod =
            module.DefineGlobalMethod(
                "main",
                MethodAttributes.Static |
                    MethodAttributes.Public,
                typeof(void), null
                );
        MethodInfo ConsoleWriteLineMethod =
            ((typeof(Console)).GetMethod("WriteLine",
                new Type[] { typeof(string) } ));
        ILGenerator il = mainMethod.GetILGenerator();
        il.Emit(OpCodes.Ldstr,"Hello, World!");
        il.Emit(OpCodes.Call,ConsoleWriteLineMethod);
        il.Emit(OpCodes.Ret);
        module.CreateGlobalFunctions();
        assembly.SetEntryPoint(
            mainMethod,PEFileKinds.ConsoleApplication
            );
        assembly.Save("hello.exe");
    }
}
```

6.3. Сравнение возможностей библиотек

	Metadata Unmanaged API	Reflection API
Чтение метаданных	+	+
Генерация метаданных	+	+
Чтение CIL-кода	–	–
Генерация CIL-кода	–	+