Лекция № 6

I T E A

IT Education Academy

WWW.ITEA.UA

C# Base

Урок № 6

Рефлексия

План урока

- Общее понятие рефлексии и отражения
- Метаданные, манифест сборки, объект Type
- Позднее связывание. Класс Assembly
- Динамическая генерация кода. Класс Activator
- Генерация кода во время выполнения программы

Рефлексия

Рефлексия (отражение) представляет собой процесс выявления типов во время выполнения приложения. Каждое приложение содержит набор используемых классов, интерфейсов, а также их методов, свойств и прочих кирпичиков, из которых складывается приложение. И рефлексия как раз и позволяет определить все эти составные элементы приложения.

Парадигма программирования, положенная в основу отражения, **называется рефлексивным программированием**. Это один из видов метапрограммирования.

Основной функционал рефлексии сосредоточен в пространстве имен **System.Reflection.**

Класс Type

- **Класс Type** является корневым классом для функциональных возможностей рефлексии и основным способом доступа к метаданным.

- С помощью членов класса **Type** можно получить сведения об объявленных в типе элементах: конструкторах, методах, полях, свойствах и событиях класса, а также о модуле и сборке, в которых развернут данный класс.

- **Класс Type** представляет изучаемый тип, инкапсулируя всю информацию о нем.

- **Класс Type** является производным от абстрактного класса *System.Reflection.MemberInfo*

Способы получения экземпляра класса Type

- Вызов метода GetType() на экземпляре требуемого класса.

```
Type type = myClass.GetType();
```

- Вызов статического метода GetType() класса `Type`.

```
Type type = Type.GetType("ConsoleApp4.MyClass");
```

- Использование оператора `typeof()`.

```
Type type = typeof(MyClass);
```

В приведенных выше примерах результатом будет ссылка на объект `Type`, содержащий информацию о целевом типе.

Assembly

- При создании приложения в результате компиляции в Visual Studio или в консоли, результатом этой работы является файл **exe** или **dll** (в зависимости от выбранных настроек), который называется **сборкой приложения**. Сборка является базовой структурной единицей в .NET, на уровне которой проходит контроль версий, развертывание и конфигурация приложения.

- Сборки кристаллизуют всю библиотеку классов .NET - при написании кода и создании сборки своего приложения мы используем пространства имен, которые размещены в других сборках .NET.

- Класс **Assembly** представляет собой сборку, которая является модулем с возможностью многократного использования, поддержкой версий и встроенным механизмом описания общеязыковой исполняющей среды.

Assembly

- При создании приложения для него определяется набор сборок, которые будут использоваться. В проекте указываются ссылки на эти сборки, и когда приложение выполняется, при обращении к функционалу этих сборок они автоматически подгружаются.

- Но также мы можем сами динамически подгружать другие сборки, на которые в проекте нет ссылок.

- Для управления сборками в пространстве имен System.Reflection имеется класс Assembly. С его помощью можно загружать сборку, исследовать ее.

- Чтобы динамически загрузить сборку в приложение, надо использовать статические методы Assembly.LoadFrom() или Assembly.Load()

Однофайловая сборка

В однофайловой сборке все компоненты хранятся в одном файле.

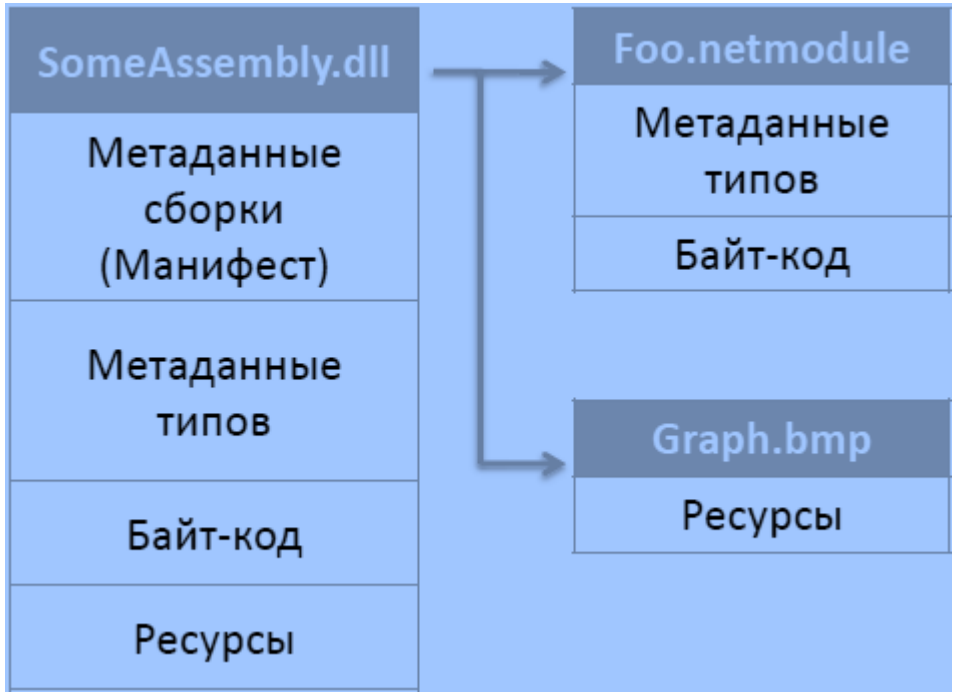
Метаданные сборки (манифест) состоят из описания сборки: имя, версия, строгое имя, информация о культуре.

Метаданные типов включают пространство имен и имя типа, члены типа и параметры, если имеются.

Байт-код (псевдокод) — машинно-независимый код низкого уровня, генерируемый транслятором и исполняемый интерпретатором.

Ресурсы – это объекты, которые используются кодом: строки, изображения, различные файлы.

Многофайловая сборка



В многофайловой сборке метаданные сборки обязательно должны находится в главном файле.

Метаданные типов, код и ресурсы могут храниться как в главном файле сборки, так и во вспомогательных файлах.

Класс Activator

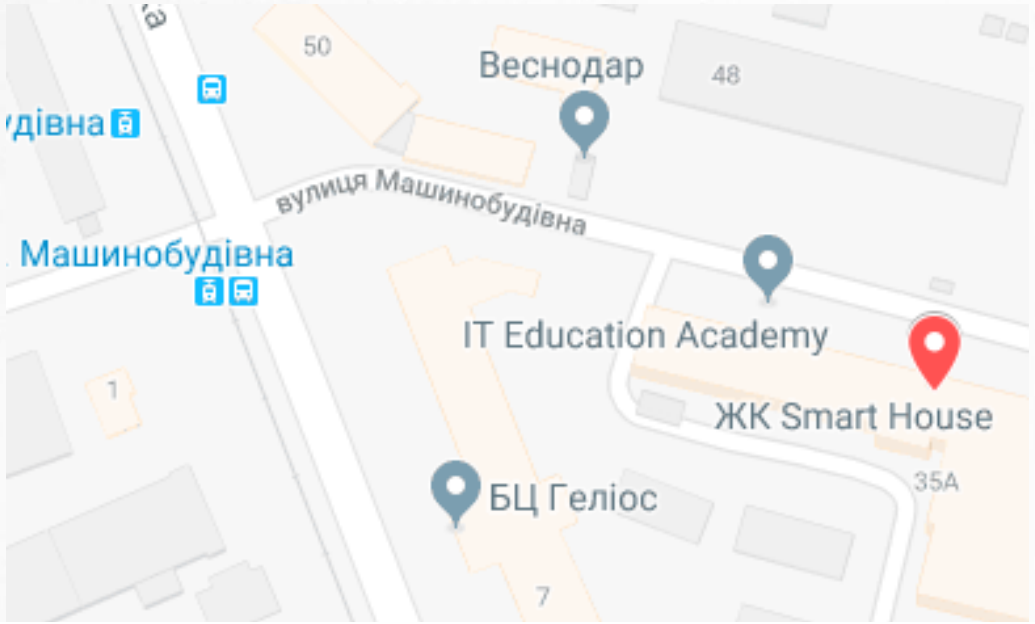
- С помощью динамической загрузки мы можем реализовать технологию позднего связывания. Позднее связывание позволяет создавать экземпляры некоторого типа, а также использовать его во время выполнения приложения.

- Ключевую роль в позднем связывании играет класс System.Activator.

- Класс Activator содержит методы для локального создания типов объектов.

- Метод CreateInstance() создает экземпляр типа, определенного в сборке путем вызова конструктора, который наилучшим образом соответствует заданным аргументам.


Презентация окончена.
Спасибо за внимание!



I T E A

КОНТАКТНЫЕ ДАННЫЕ

ITEA

ЖК “Smart House”, ул. Машиностроительная, 41 (м.Берестейская)

ЖК «Корона» улица Срибнокильская,1 м. Позняки

+38 (044) 599-01-79
facebook.com/Itea
info@itea.ua
itea.ua