# 6\_module

1. What is reflection in .NET?
2. What does reflection allow you to do?
3. What are fully qualified type names?
4. What examples of practical application of reflection can you imagine?
5. Is it possible to get information about private fields/methods using reflection?

**Рефлексия** 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. То есть основная задача рефлексии - это исследование типов.

Основной функционал рефлексии сосредоточен в пространстве имен **System.Reflection**. В нем мы можем выделить следующие основные классы:

* **Assembly**: класс, представляющий сборку и позволяющий манипулировать этой сборкой
* **AssemblyName**: класс, хранящий информацию о сборке
* **MemberInfo**: базовый абстрактный класс, определяющий общий функционал для классов EventInfo, FieldInfo, MethodInfo и PropertyInfo
* **EventInfo**: класс, хранящий информацию о событии
* **FieldInfo**: хранит информацию об определенном поле типа
* **MethodInfo**: хранит информацию об определенном методе
* **PropertyInfo**: хранит информацию о свойстве
* **ConstructorInfo**: класс, представляющий конструктор
* **ParameterInfo**: класс, хранящий информацию о параметре метода

Эти классы представляют составные блоки типа и приложения: методы, свойства и т.д. Но чтобы получить информацию о членах типа, нам надо воспользоваться классом **System.Type**.

Класс **Type** представляет изучаемый тип, инкапсулируя всю информацию о нем. С помощью его свойств и методов можно получить эту информацию. Некоторые из его свойств и методов:

* Метод **FindMembers()** возвращает массив объектов MemberInfo данного типа
* Метод **GetConstructors()** возвращает все конструкторы данного типа в виде набора объектов ConstructorInfo
* Метод **GetEvents()** возвращает все события данного типа в виде массива объектов EventInfo
* Метод **GetFields()** возвращает все поля данного типа в виде массива объектов FieldInfo
* Метод **GetInterfaces()** получает все реализуемые данным типом интерфейсы в виде массива объектов Type
* Метод **GetMembers()** возвращает все члены типа в виде массива объектов MemberInfo
* Метод **GetMethods()** получает все методы типа в виде массива объектов MethodInfo
* Метод **GetProperties()** получает все свойства в виде массива объектов PropertyInfo
* Свойство **Name** возвращает имя типа
* Свойство **Assembly** возвращает название сборки, где определен тип
* Свойство **Namespace** возвращает название пространства имен, где определен тип
* Свойство **IsArray** возвращает true, если тип является массивом
* Свойство **IsClass** возвращает true, если тип представляет класс
* Свойство **IsEnum** возвращает true, если тип является перечислением
* Свойство **IsInterface** возвращает true, если тип представляет интерфейс

### Получение типа

Чтобы управлять типом и получать всю информацию о нем, нам надо сперва получить данный тип. Это можно сделать тремя способами: с помощью оператора **typeof**, с помощью метода **GetType()** класса Object и применяя статический метод Type.GetType().

И третий способ получения типа - статический метод **Type.GetType()**:

|  |  |
| --- | --- |
| 1 | Type? myType = Type.GetType("Person", false, true); |

Первый параметр указывает на полное имя класса с пространством имен. Второй параметр указывает, будет ли генерироваться исключение, если класс не удастся найти. В данном случае значение false означает, что исключение не будет генерироваться. И третий параметр указывает, надо ли учитывать регистр символов в первом параметре. Значение true означает, что регистр игнорируется. Поскольку указанный тип может отсутствовать, то метод возвращает объект nullable-типа

В качестве альтернативы можно применять оператор typeof, передавая в него имя типа с указанием пространства имен:

|  |  |
| --- | --- |
| 1 | Type myType = typeof(PeopleTypes.Person); |

**Поиск реализованных интерфейсов**

Чтобы получить все реализованные типом интерфейсы, надо использовать метод **GetInterfaces()**, который возвращает массив объектов Type:

|  |  |
| --- | --- |
| 4  5  6  7  8  9  10 | Type myType = typeof(Person);    Console.WriteLine("Реализованные интерфейсы:");  foreach (Type i in myType.GetInterfaces())  {       Console.WriteLine(i.Name);  } |
|  |  |

2) Вы можете использовать рефлексию для **динамического создания экземпляра типа, получения типа из существующего объекта и вызова его методов или доступа к его полям и свойствам.** Если вы используете атрибуты в своем коде, рефлексия позволяет вам получить к ним доступ.

3) A fully qualified type name consists of an assembly name specification, a namespace specification, and a type name.

Полное имя типа состоит из спецификации названия сборки, пространства имен и названия самого типа.

Type name specifications are used by methods such as [Type.GetType](https://docs.microsoft.com/en-us/dotnet/api/system.type.gettype), [Module.GetType](https://docs.microsoft.com/en-us/dotnet/api/system.reflection.module.gettype), [ModuleBuilder.GetType](https://docs.microsoft.com/en-us/dotnet/api/system.reflection.emit.modulebuilder.gettype), and [Assembly.GetType](https://docs.microsoft.com/en-us/dotnet/api/system.reflection.assembly.gettype).

4) Загрузите сборку во время выполнения на основе ее имени.

Загружать объект из сборки во время выполнения на основе его имени.

Вызов конструктора объекта на основе сигнатуры объекта класса, неизвестного во время компиляции.

1. Сама утилита ildasm.exe подгружает сборку, определяет в ней методы, их параметры и тому подобное. Для определения метаданных сборки утилита использует рефлексию.