### 1. Что такое рефлексия?

Рефлексия в .NET — это механизм, который позволяет программно исследовать метаданные типов в сборках, а также создавать и изменять объекты во время выполнения, обращаться к их свойствам, методам и полям. С помощью рефлексии можно получить информацию о типах, методах, свойствах, конструкторах и атрибутах, а также динамически создавать экземпляры объектов и вызывать их методы.

### 2. Какие есть способы получения объекта типа `Type`? Приведите три способа получения типа.

Существует несколько способов получить объект типа `Type`:

1. \*\*Через метод `GetType()`\*\* — используется для получения типа экземпляра объекта:

```csharp

Type type = myObject.GetType();

```

2. \*\*Через оператор `typeof()`\*\* — используется для получения типа, зная имя класса на момент компиляции:

```csharp

Type type = typeof(MyClass);

```

3. \*\*Через метод `Type.GetType()`\*\* — используется для получения типа по строковому имени класса:

```csharp

Type type = Type.GetType("Namespace.MyClass");

```

### 3. Охарактеризуйте классы из пространства имен `System.Reflection`.

Пространство имен `System.Reflection` содержит несколько важных классов, которые обеспечивают доступ к метаданным сборки, типам и членам типов:

- \*\*`Type`\*\* — представляет метаданные типа (классов, интерфейсов, структур и делегатов). Предоставляет информацию о членах типа, таких как методы, свойства, поля и конструкторы.

- \*\*`MethodInfo`\*\* — предоставляет информацию о методах типа, включая их параметры, возвращаемое значение и другие атрибуты.

- \*\*`PropertyInfo`\*\* — предоставляет информацию о свойствах типа.

- \*\*`FieldInfo`\*\* — предоставляет информацию о полях типа.

- \*\*`ConstructorInfo`\*\* — предоставляет информацию о конструкторах типа.

- \*\*`Assembly`\*\* — представляет сборку .NET и позволяет работать с метаданными, находящимися в сборке.

- \*\*`Module`\*\* — представляет модуль сборки и позволяет исследовать типы, объявленные в этом модуле.

- \*\*`ParameterInfo`\*\* — предоставляет информацию о параметрах метода.

- \*\*`CustomAttributeData`\*\* — используется для работы с аттрибутами, привязанными к типам и членам типов.

### 4. Как можно использовать класс `System.Type`? Перечислите его свойства и методы.

Класс `Type` представляет метаданные типа в .NET и предоставляет множество полезных свойств и методов для работы с типами. Вот некоторые из них:

#### Свойства:

- \*\*`FullName`\*\* — строковое имя типа, включая его пространство имен.

- \*\*`Namespace`\*\* — пространство имен, в котором определен тип.

- \*\*`Name`\*\* — имя типа (без пространства имен).

- \*\*`IsClass`\*\* — указывает, является ли тип классом.

- \*\*`IsInterface`\*\* — указывает, является ли тип интерфейсом.

- \*\*`IsAbstract`\*\* — указывает, является ли тип абстрактным.

- \*\*`BaseType`\*\* — возвращает базовый тип типа (для классов это базовый класс).

- \*\*`IsEnum`\*\* — указывает, является ли тип перечислением.

- \*\*`IsValueType`\*\* — указывает, является ли тип значимым типом (например, структурами).

- \*\*`IsArray`\*\* — указывает, является ли тип массивом.

#### Методы:

- \*\*`GetMethods()`\*\* — возвращает массив методов, объявленных для типа.

- \*\*`GetProperties()`\*\* — возвращает массив свойств типа.

- \*\*`GetFields()`\*\* — возвращает массив полей типа.

- \*\*`GetInterfaces()`\*\* — возвращает массив интерфейсов, реализованных типом.

- \*\*`GetConstructors()`\*\* — возвращает массив конструкторов типа.

- \*\*`GetCustomAttributes()`\*\* — возвращает массив атрибутов, примененных к типу.

- \*\*`IsAssignableFrom()`\*\* — проверяет, может ли объект данного типа быть присвоен объекту другого типа.

- \*\*`GetTypeInfo()`\*\* — получает объект, который предоставляет метаданные типа.

### 5. Что такое позднее и раннее связывание?

- \*\*Раннее связывание\*\* (early binding) — процесс, при котором компилятор связывает вызовы методов и доступ к свойствам с их реальными реализациями в момент компиляции. Это обеспечивает большую производительность, но меньше гибкости, так как все зависимости должны быть известны заранее.

- \*\*Позднее связывание\*\* (late binding) — процесс, при котором связывание методов и свойств с их реализациями происходит в момент выполнения программы. Это обычно используется при работе с рефлексией, динамическими типами или COM-объектами, когда тип или метод неизвестен до выполнения.

Пример позднего связывания с использованием рефлексии:

```csharp

Type type = Type.GetType("Namespace.MyClass");

MethodInfo method = type.GetMethod("MyMethod");

method.Invoke(Activator.CreateInstance(type), null);

```

### 6. Как динамически загрузить сборку в приложение?

Для динамической загрузки сборки в приложение можно использовать метод `Assembly.Load` или `Assembly.LoadFrom`. Пример загрузки сборки из файла:

```csharp

Assembly assembly = Assembly.LoadFrom("path\_to\_assembly.dll");

```

После загрузки сборки можно исследовать типы, методы и другие члены сборки с помощью рефлексии.

### 7. Что такое позднее (раннее) связывание?

Вопрос 7 аналогичен вопросу 5, т.к. здесь подразумеваются те же определения позднего и раннего связывания, описанные выше.

### 8. Для чего предназначены `BindingFlags`? Какую комбинацию флагов необходимо использовать, чтобы иметь возможность получать приватные члены класса?

`BindingFlags` — это перечисление, которое используется для указания флагов при поиске членов (методов, свойств, полей и т.д.) типа с помощью рефлексии. Оно позволяет настроить поведение поиска.

Вот комбинации флагов для различных целей:

- \*\*`BindingFlags.Public`\*\* — искать только публичные члены.

- \*\*`BindingFlags.NonPublic`\*\* — искать только непубличные члены.

- \*\*`BindingFlags.Instance`\*\* — искать члены экземпляра (не статические).

- \*\*`BindingFlags.Static`\*\* — искать только статические члены.

- \*\*`BindingFlags.FlattenHierarchy`\*\* — искать члены, унаследованные от базовых типов.

Чтобы получить приватные члены класса, нужно использовать флаги `BindingFlags.NonPublic | BindingFlags.Instance` для экземплярных членов или `BindingFlags.NonPublic | BindingFlags.Static` для статических:

```csharp

Type type = typeof(MyClass);

var privateFields = type.GetFields(BindingFlags.NonPublic | BindingFlags.Instance);

```