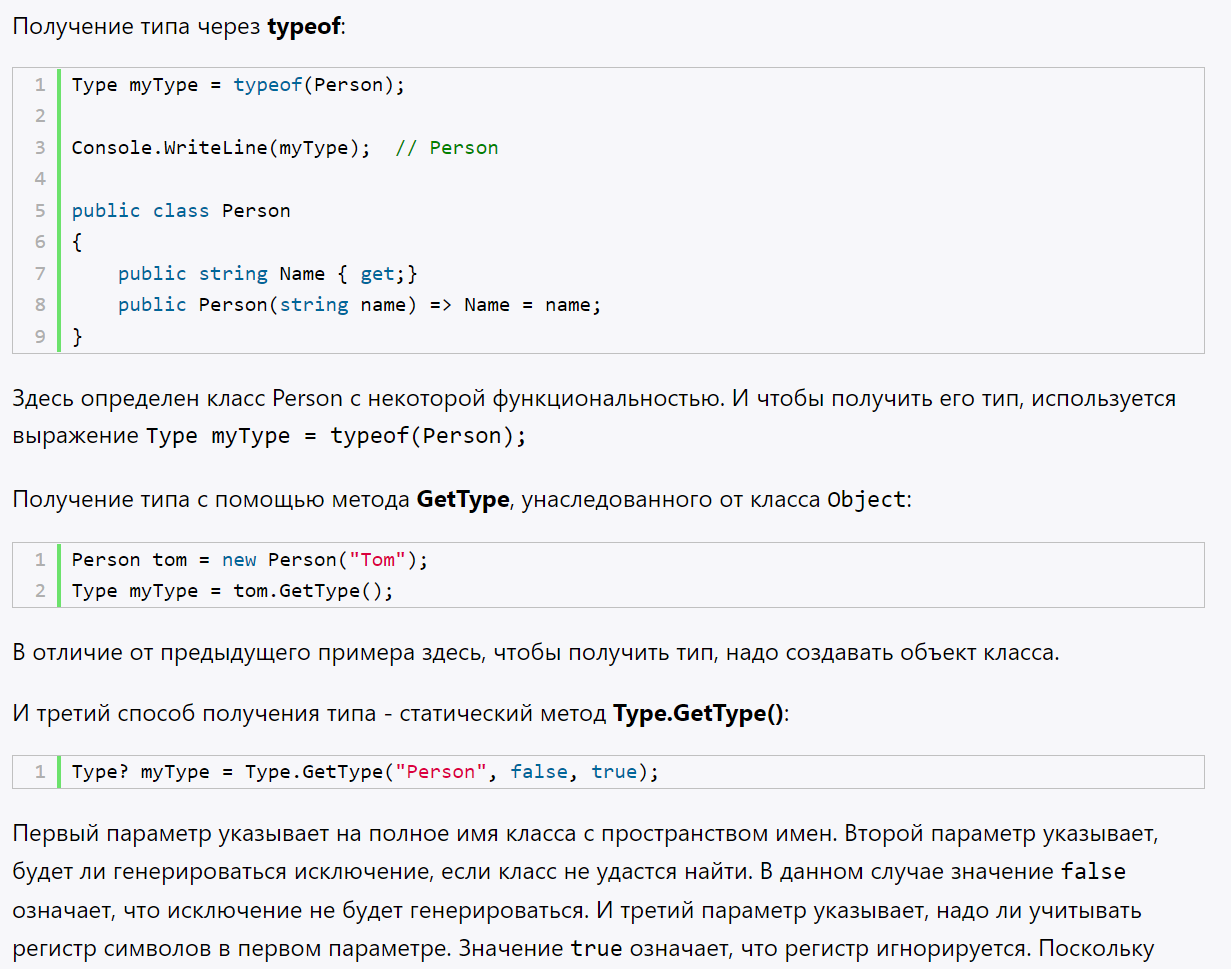
1. Что такое рефлексия?

Рефлексия C# — это процесс выявления типов (объектов типа Type) во время выполнения приложения (run-time). С помощью классов, содержащихся в пространстве имен System.Reflection мы можем получать информацию о сборке, используемых классах, свойствах классов и прочую информацию для использования в нашем же приложении.

2. Какие есть способы получения объекта типа Type. Приведите три способа получения 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().



3. Охарактеризуйте классы из пространства имен *System*.Reflection.

Класс System.Type служит основой рефлексии. Благодаря методам класса System.Type можно определять информацию об используемых типах во время выполнения программы. Затем эту информацию можно анализировать, обрабатывать в зависимости от поставленной задачи.

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
* **Module**: класс, позволяющий получить доступ к определенному модулю внутри сборки
* **ParameterInfo**: класс, хранящий информацию о параметре метода

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

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

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().

\*Подробнее в п.2

5. Что такое позднее и раннее связывание?

Раннее связывание - это когда метод, который будет вызван, известен во время компиляции, например, вызов статического метода. Позднее связывание - это когда вызов метода может быть осуществлен только во время выполнения и у компилятора нет информации, чтобы проверить корректность такого вызова. Использование позднего связывания менее безопасно в том плане, что при жестком кодировании всех типов (ранее связывание) на этапе компиляции мы можем отследить многие ошибки. В то же время позднее связывание позволяет создавать расширяемые приложения, когда дополнительный функционал программы неизвестен, и его могут разработать и подключить сторонние разработчики. Ключевую роль в позднем связывании играет класс **System.Activator**. С помощью его статического метода **Activator.CreateInstance()** можно создавать экземпляры заданного типа.

6. Как динамически загрузить сборку в приложение?

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()**.

Метод LoadFrom() принимает в качестве параметра путь к сборке.

7. Что такое позднее (раннее) связывание?

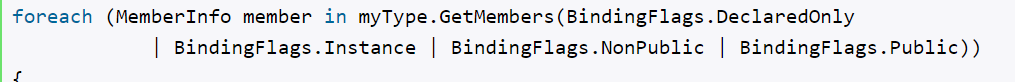
==п.5

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

Сущетвукет простая форма метода GetMembers(), которая извлекает все общедоступные публичные методы. Но мы можем использовать и другую форму метода: MembersInfo[] GetMembers(BindingFlags). Перечисление **BindingFlags** может принимать различные значения:

* **DeclaredOnly**: получает только методы непосредственно данного класса, унаследованные методы не извлекаются
* **Instance**: получает только методы экземпляра
* **NonPublic**: извлекает не публичные методы
* **Public**: получает только публичные методы
* **Static**: получает только статические методы

Объединяя данные значения с помощью побитовой операции ИЛИ можно комбинировать вывод. Например, получим только компоненты непосредственно самого класса без унаследованных, как публичные, так и все остальные:



Для получения одного компонента можно использовать метод GetMember(), в который передается имя компонента. И опционально можно передать флаги BindingFlags.

