Рефлексия

№ урока: 6 **Курс:** C# Professional

Средства обучения: Компьютер с установленной Visual Studio

Обзор, цель и назначение урока

Понимание принципов рефлексии и использование в своей работе классов `Type` и `Assembly` позволяют программисту получать доступ к информации, относящейся к определению любого типа данных, а также выгружать или загружать информацию из внешней сборки в свою программу.

Изучив материал данного занятия, учащийся сможет:

- Использовать объект `Type`.
- Используя позднее связывание, получать информацию о типе переданного объекта.
- Динамически загружать сборки.
- Генерировать код во время выполнения программы.
- Определять члены, извлекаемые из `Type` с помощью перечисления `BindingFlags`.

Содержание урока

1. Общее понятие рефлексии и отражения.
2. Метаданные, манифест сборки, объект `Type`.
3. Позднее связывание. Класс `Assembly`.
4. Динамическая генерация кода. Класс `Activator`.
5. Генерация кода во время выполнения программы.

Резюме

- Рефлексией (reflection) называется процесс обнаружения типов во время выполнения.
- Рефлексия позволяет: перечислять члены типа, создавать новые экземпляры объекта, запускать на выполнение члены объекта, извлекать информацию о типе, извлекать информацию о сборке, исследовать пользовательские атрибуты, примененные к типу, создавать и компилировать новые сборки.
- Метаданные описывают все классы и члены классов, определённые в сборке, а также классы и члены классов, которые текущая сборка вызывает из другой сборки.
- Манифест сборки – коллекция данных, с описанием того, как элементы любой сборки (статической или динамической) связаны друг с другом. Манифест сборки содержит все метаданные, необходимые для задания требований сборки к версиям и удостоверения безопасности, а также все метаданные, необходимые для определения области действия сборки и разрешения ссылок на ресурсы и классы. Манифест сборки может храниться в PE-файле (EXE или DLL) с кодом MSIL или же в отдельном PE-файле, содержащем только данные манифеста.
- Модули — это контейнеры типов, расположенные внутри сборки. Модуль может быть контейнером в простой, или многофайловой сборке. Несколько модулей в одной сборке применяются в редких случаях, когда нужно добавить код на разных языках в одну сборку или обеспечить поддержку выборочной загрузки модулей.
- Пространство имен `System.Reflection` содержит типы, предназначенные для извлечения сведений о сборках, модулях, членах, параметрах и других объектах в управляемом коде путем обработки их метаданных. Эти типы также можно использовать для работы с экземплярами загруженных типов, например, для подключения событий или вызова методов.
- Класс `Type` является корневым классом для функциональных возможностей пространства имен `System.Reflection` и основным способом доступа к метаданным. При помощи членов класса `Type` можно получить сведения об объявленных в типе

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com

элементах: конструкторах, методах, полях, свойствах и событиях класса, а также о модуле и сборке, в которых развернут данный класс.

- Две ссылки на объект `Type` указывают на один и тот же объект тогда и только тогда, когда они представляют один и тот же тип.
- Экземпляр класса `Type` можно получить несколькими способами. Единственное, что нельзя делать — так это напрямую создавать объект `Type` с помощью ключевого слова `new`, потому что класс `Type` является абстрактным.
- Класс `Assembly` используется для загрузки сборок, изучения метаданных и компонентов сборок, выявления содержащихся в сборках типов и создания экземпляров этих типов.
- Класс `Module` можно использовать для извлечения или поиска типов в заданном модуле. Для сборок, изначально написанных на языке с поддержкой модулей, данный класс также поддерживает методы `GetField, GetFields, GetMethod` и `GetMethods`. К модулям данного типа поля и методы можно подключать непосредственно.
- Для загрузки сборок рекомендуется использовать метод `Load`, который идентифицирует загружаемые сборки по отображаемому имени. Методы `LoadFile` и `LoadFrom` предоставляются для редко используемых скриптов, в которых сборка должна определяться по пути.
- Чтобы получить объект `Assembly` для выполняемой в текущий момент сборки, следует воспользоваться методом `GetExecutingAssembly`.
- Метод `GetName` возвращает объект `AssemblyName`, который обеспечивает доступ к отображаемому имени сборки.
- Метод `GetCustomAttributes` используется для вывода атрибутов, примененных для сборки.
- Метод `GetFiles` обеспечивает доступ к файлам в манифесте сборки.
- Метод `GetManifestResourceNames` предоставляет имена всех ресурсов в манифесте сборки.
- Метод `GetTypes` выводит все типы, содержащиеся в сборке. Метод `GetExportedTypes` выводит типы, которые видимы вызывающим объектам, находящимся вне сборки.
- Метод `GetType` может использоваться для поиска конкретного типа в сборке.
- Метод `CreateInstance` может использоваться для поиска и создания экземпляров ряда типов в сборке.
- Перечисление `BindingFlags` управляет извлечением членов типа с помощью методов `GetMembers` и других методов, специфичных для членов типа. Перечисление `BindingFlags` поддерживает флаги, то есть, принимает несколько значений.
- Поздним связыванием (late binding) называется технология, которая позволяет создавать экземпляр определенного типа и вызывать его члены во время выполнения без кодирования факта его существования жестким образом на этапе компиляции. При создании приложения, в котором предусмотрено позднее связывание с типом из какой-то внешней сборки, добавлять ссылку на эту сборку нет никакой причины, и потому в манифесте вызывающего кода она непосредственно не указывается.
- Класс `System.Activator` (определенный в сборке mscorlib.dll) играет ключевую роль в процессе позднего связывания в .NET. Его метод `CreateInstance`, позволят создавать экземпляр подлежащего позднему связыванию типа. Этот метод имеет несколько перегруженных версий и потому обеспечивает довольно высокую гибкость. В самой простой версии `CreateInstance` принимает действительный объект `Type`, описывающий сущность, которая должна размещаться в памяти на лету.
- Динамическая генерация кода предназначена для динамической загрузки и исполнения сборки без предварительного обращения к ней.
- Динамический код создается, только если загружается код, на который приложение раньше не ссылалось.
- Для обеспечения динамической генерации кода, необходимо получить информацию о типах и запросить объект `ConstructorInfo` для конструирования нового типа. После получения `ConstructorInfo` для создания требуемого объекта достаточно вызвать конструктор.
- Класс `PropertyInfo` поддерживает получение и установку отдельных свойств. В данном случае метод `GetValue` класса `PropertyInfo` вызывается для получения значения свойства `Count`.

CyberBionic systematics

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com

- Класс `MemberInfo` предназначен для исполнения произвольного кода над заданным экземпляром типа также для исполнения статического кода, связанного с определенным типом.
- Генерация кода во время выполнения – техника, которая позволяет определить информацию о сборках и типах, или даже создать собственную сборку (хранимую в памяти, на диске) непосредственно в момент выполнения.

Закрепление материала

- Что такое рефлексия?
- Что такое метаданные и манифест сборки?
- Что такое метаданные типа?
- Что такое модуль?
- Какие ограничения действуют на многофайловые сборки?
- Какими способами можно получать объекты `Type`?
- Какое назначение перечисления `BindingFlags` и как его использовать?
- Как динамически создать сборку? Какие шаги являются обязательными?
- В чем отличие загрузки сборки только для получения информации от обычной загрузки для выполнения?

Дополнительное задание

Создайте программу-рефлектор, которая позволит получить информацию о сборке и входящих в ее состав типах. Для основы можно использовать программу-рефлектор из урока.

Самостоятельная деятельность учащегося

Задание 1
Выучите основные конструкции и понятия, рассмотренные на уроке.

Задание 2
Создайте свою пользовательскую сборку по примеру сборки `CarLibrary` из урока, сборка будет использоваться для работы с конвертером температуры.

Задание 3
Создайте программу, в которой предоставьте пользователю доступ к сборке из Задания 2. Реализуйте использование метода конвертации значения температуры из шкалы Цельсия в шкалу Фаренгейта. Выполняя задание используйте только рефлексию.

Задание 4
Зайдите на сайт MSDN.
Используя поисковые механизмы MSDN, найдите самостоятельно описание темы по каждому примеру, который был рассмотрен на уроке, так, как это представлено ниже, в разделе «Рекомендуемые ресурсы», описания данного урока. Сохраните ссылки и дайте им короткое описание.

Рекомендуемые ресурсы

MSDN: класс `Type`
http://msdn.microsoft.com/ru-ru/library/system.type.aspx

MSDN: класс `Assembly`
http://msdn.microsoft.com/ru-ru/library/system.reflection.assembly.aspx

MSDN: пространство имен `System.Reflection`
http://msdn.microsoft.com/ru-ru/library/system.reflection.aspx

MSDN: класс `Activator`

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com

http://msdn.microsoft.com/ru-ru/library/system.activator(VS.95).aspx

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com