Тема 14. Атрибуты

Парадигмы программирования. C#

План

- Введение в атрибуты
- Создание собственных атрибутов
- Получение данных из атрибутов

Введение в атрибуты

Часть 1

Что такое атрибуты?

- Атрибуты это
  - Декларативные тэги, сообщающие информацию исполнительной среде
  - Данные, сохраняемые в метаданных элемента
- .NET Framework предоставляет предопределённые атрибуты
  - Исполнительная среда содержит код, проверяющий значения атрибутов и реагирует на эти значения

Применение атрибутов

- Синтаксис: используйте квадратные скобки для указания атрибута

```
[attribute(позиционные параметры,
  именованный параметр=значение, ...)]
Элемент
```

- Для применения нескольких атрибутов к одному элементу выберите один из вариантов
  - Укажите несколько атрибутов в отдельных квадратных скобках
  - Используйте единственные квадратные скобки и разделяйте атрибуты запятыми
  - Для некоторых элементов, например, сборок, явно укажите имя элемента, ассоциированного с атрибутом

Пример использования единственного атрибута

```
using System.ComponentModel;
...
[DefaultEvent("ShowResult")]
public class Calculator:
  System.Windows.Forms.UserControl
{
...
}
```

Пример использования нескольких атрибутов

```
[DefaultEvent("ShowResult")]
[DefaultProperty("Text")]
public class Calculator:
   System.Windows.Forms.UserControl
{
...
}

[DefaultEvent("Rotate"), DefaultProperty("Text")]
public class Rotator:
   System.Windows.Forms.UserControl
{
...
}
```

Пример задания атрибута сборки

```
using System;
[assembly:CLSCompliant(true)]
class MyClass
{
...
}
```

Общие предопределённые атрибуты

- Общие атрибуты
- Атрибуты взаимодействия с COM
- Атрибуты обработки транзакций
- Атрибуты управления визуальным дизайнером

Общие атрибуты

- `Conditional`
  - Применяется к методам
  - Проверяет, определён ли символ
  - Если символ определён, метод вызывается из кода, как обычно
  - Если символ не определён, вызовы метода не создаются
- `DllImport`
  - Применяется к методам
  - Показывает, что метод реализован в неуправляемом коде, в указанной DLL

Атрибуты взаимодействия с COM

- ComImport
- ComRegisterFunction
- ComUnregisterFunction
- DispId
- In
- MarshalAs
- ProgId
- Out
- InterfaceType

Атрибуты управления визуальным дизайнером

- `Bindable`
- `DefaultProperty`
- `DefaultValue`
- `Localizable`
- `DefaultEvent`
- `Category`
- `Description`

Использование условного атрибута

- Инструмент отладки
  - Вызывает условную компиляцию вызовов методов в зависимости от значения программно-задаваемого символа
  - Не приводит к условной компиляции метода целиком
- Ограничения на методы
  - Должны возвращать тип void
  - Не должны определяться, как override
  - Не должны реализовывать интерфейс

Пример использования условного атрибута

```
using System.Diagnostics;

class MyClass
{
    [Conditional ("DEBUGGING")]
    public static void MyMethod( )
    {
        ...
    }
}

#define DEBUGGING

class AnotherClass
{
    public static void Test( )
    {
        MyClass.MyMethod( );
    }
}
```

Использование атрибута DllImport

- Назначение атрибута DllImport
  - Осуществление вызова неуправляемого кода, расположенного в DLL, из управляемого кода
  - Пометить внешний метод, как функцию, расположенную в неуправляемой DLL

```
using System.Runtime.InteropServices;
...
public class MyClass( )
{
    [DllImport("MyDLL.dll", EntryPoint="MyFunction")]
    public static extern int MyFunction(string param1);
    ...
    int result = MyFunction("Hello Unmanaged Code");
    ...
}
```

Использование атрибута транзакции

- Для управления транзакциями в COM+
  - Укажите, что компонент будет включаться в транзакцию при необходимости её подтверждения
  - Используйте атрибут Transaction для класса, реализующего компонент

```
using System.EnterpriseServices;
...
[Transaction(TransactionOption.Required)]
public class MyTransactionalComponent
{
...
}
```

Создание собственных атрибутов

Часть 2

Как создать свой атрибут?

- Атрибут – это класс, являющийся наследником System.Attribute
- Для создания атрибута следует
  - Создать класс-наследник System.Attribute
  - Задать область определения создаваемого атрибута
  - Задать параметры атрибута

Задание области определения атрибута

- Используйте атрибут AtributeUsage для задания области определения

```
[AttributeUsage(target_elements)]
public class MyAttribute: System.Attribute
{ ... }
```

- Для указания применимости создаваемого атрибута к нескольким элементам используйте несколько атрибутов AtributeUsage, разделённых символом «|»

```
[AttributeUsage(AttributeTargets.Method |
AttributeTargets.Constructor)]
public class MyAttribute: System.Attribute
{
...
}
```

Элементы, для которых применимы атрибуты

- Class
- Constructor
- Delegate
- Enum
- Event
- Field
- Interface
- Method
- Module
- Parameter
- Property
- ReturnValue
- Struct
- Assembly
- All

Создание класса атрибута

- Обязательно унаследовать класс атрибута
  - Прямое или косвенное наследование от System.Attribute
  - Суффикс названия класса атрибута должен быть Attribute
- Компоненты класса атрибута
  - Для позиционных атрибутов должен использоваться конструктор
  - Для задания именованных параметров используются свойства

Пример создания класса атрибута

```
public class DeveloperInfoAttribute:
   System.Attribute
{
     ...
     public DeveloperInfoAttribute(string developer)
     {
          ...
     }
     public string Date
     {
          get { ... }
          set { ... }
     }
}
```

Пример использования атрибута

```
[DeveloperInfoAttribute("Vasya",
  Date="26-11-2009")]
public class MyClass
{
    ...
}
```

Порядок обработки созданного (пользовательского) атрибута

1. Поиск класса атрибута
2. Проверка области определения атрибута
3. Проверка конструктора атрибута
4. Создание экземпляра объекта атрибута
5. Проверка именованных параметров
6. Установка значений свойств объекта атрибута в соответствии со значениями именованных параметров
7. Сохранение текущего состояния объекта атрибута в метаданных

Использование нескольких атрибутов

- Элемент может иметь несколько атрибутов
  - Определяйте каждый атрибут по отдельности

```
[Transaction(TransactionOption.Required)]
[DefaultProperty("Balance")]
public class FinancialComponent: System.Attribute
{
    ...
    public long Balance
    {
        ...
    }
}
```

- Элемент может иметь несколько экземпляров одного и того же атрибута
  - Укажите AllowMultiple = true

```
[AttributeUsage(AttributeTargets.Class,
  AllowMultiple=true)]
public class DeveloperInfoAttribute: System.Attribute
{
    ...
}
```

Получение данных из атрибутов

Часть 3

Проверка метаданных класса

- Осуществление запроса метаданных класса
  - Используется класс MemberInfo, размещённый в области имён System.Reflection
  - Заполните объект класса MemberInfo с использованием экземпляра класса System.Type
  - Создайте объект System.Type с помощью оператора typeof

```
System.Reflection.MemberInfo[ ]
  memberInfoArray;
memberInfoArray =
  typeof(MyClass).GetMembers( );
```

Запрос информации атрибутов

- Используйте метод GetCustomAttributes для извлечения всей информации об атрибуте в массив

```
System.Reflection.MemberInfo typeInfo;
typeInfo = typeof(MyClass);
object [ ] attrs =
  typeInfo.GetCustomAttributes(false);
foreach(Attribute atr in attrs)
{
    if (atr is DeveloperInfoAttribute)
    {
        DeveloperInfoAttribute dia =
            (DeveloperInfoAttribute)atr;
        Console.WriteLine("{0} {1}",
            dia.Developer, dia.Date);
    }
}
```

Проверка наличия атрибута у класса

- Используйте метод IsDefined для проверки того, был ли указан интересующий атрибут для данного класса

```
Type devInfoAttrType =
    typeof(DeveloperInfoAttribute);
if (typeInfo.IsDefined(devInfoAttrType, false))
{
    object [ ] attrs =
    typeInfo.GetCustomAttributes(
        devInfoAttrType, false);
    ...
}
```

Заключение

- Спасибо за внимание!