Теория и практика программирования

Шпилёв Пётр Валерьевич

Санкт-Петербургский государственный университет
Математико-механический факультет
Кафедра статистического моделирования

Лекция 10

Санкт-Петербург
2015 г.

e-mail: pitshp@hotmail.com

Создание объекта класса Assembly

static Assembly LoadFrom(string файл_сборки)
где *файл_сборки* обозначает конкретное имя файла сборки.

Создание объекта класса Assembly

static Assembly LoadFrom(string файл_сборки)
где *файл_сборки* обозначает конкретное имя файла сборки.

Получение типов данных, определенных в объекте класса Assembly

Type[] Имя имя_объекта.GetTypes()

Создание объекта класса Assembly

static Assembly LoadFrom(string файл_сборки)
где *файл_сборки* обозначает конкретное имя файла сборки.

Получение типов данных, определенных в объекте класса Assembly

Type[] Имя имя_объекта.GetTypes()

Пример:

```
// Загрузить сборку MyClasses.exe.
Assembly asm = Assembly.LoadFrom("MyClasses.exe");
// Обнаружить типы, содержащиеся в сборке MyClasses.exe.
Type[] alltypes = asm.GetTypes();
```

Создание объекта класса Assembly

static Assembly LoadFrom(string файл_сборки)
где *файл_сборки* обозначает конкретное имя файла сборки.

Получение типов данных, определенных в объекте класса Assembly

Type[] Имя имя_объекта.GetTypes()

Пример:

```
// Загрузить сборку MyClasses.exe.
Assembly asm = Assembly.LoadFrom("MyClasses.exe");
// Обнаружить типы, содержащиеся в сборке MyClasses.exe.
Type[] alltypes = asm.GetTypes();
```

Упражнение 10.1

Обнаружить сборку (сделать для своего класса), определить типы и создать объект с помощью рефлексии.

Создание объекта класса Assembly

static Assembly LoadFrom(string файл_сборки)
где *файл_сборки* обозначает конкретное имя файла сборки.

Получение типов данных, определенных в объекте класса Assembly

Type[] Имя имя_объекта.GetTypes()

Пример:

```
// Загрузить сборку MyClasses.exe.
Assembly asm = Assembly.LoadFrom("MyClasses.exe");
// Обнаружить типы, содержащиеся в сборке MyClasses.exe.
Type[] alltypes = asm.GetTypes();
```

Упражнение 10.1

Обнаружить сборку (сделать для своего класса), определить типы и создать объект с помощью рефлексии.

Упражнение 10.2

Выполнить предыдущее упражнение для dll-библиотеки созданной одногруппником. Вызвать несколько методов определенных в классах библиотеки.

Атрибут

С помощью атрибута(т.е. специального класса) определяются дополнительные сведения, связанные с классом, структурой, методом и т.д.

Атрибут

С помощью атрибута(т.е. специального класса) определяются дополнительные сведения, связанные с классом, структурой, методом и т.д.

Пример: [AttributeUsage(AttributeTargets.All)]

```
public class RemarkAttribute : Attribute
{
    string pri_remark; // базовое поле свойства Remark
    public RemarkAttribute(string comment)
    {
        pri_remark = comment;
    }
    public string Remark
    {
        get
        {
            return pri_remark;
        }
    }
}
```

Замечание

Атрибут указывается перед тем элементом, к которому он присоединяется, и для этого его конструктор заключается в квадратные скобки.

Замечание

Атрибут указывается перед тем элементом, к которому он присоединяется, и для этого его конструктор заключается в квадратные скобки.

Пример:

```
[RemarkAttribute("В этом классе используется атрибут.")]
class UseAttrib
{
// ...
}
```

Замечание

Атрибут указывается перед тем элементом, к которому он присоединяется, и для этого его конструктор заключается в квадратные скобки.

Пример:

```
[Remark("В этом классе используется атрибут.")]
class UseAttrib
{
// ...
}
```

Замечание

Атрибут указывается перед тем элементом, к которому он присоединяется, и для этого его конструктор заключается в квадратные скобки.

Пример:

```
[Remark("В этом классе используется атрибут.")]
class UseAttrib
{
// ...
}
```

Два метода получения атрибутов объекта

[1] object[] GetCustomAttributes(bool наследование)
Если наследование имеет логическое значение true, то в список включаются атрибуты всех базовых классов, наследуемых по иерархической цепочке. В противном случае атрибуты извлекаются только из тех классов, которые определяются указанным типом.

[2] static Attribute.GetCustomAttribute(MemberInfo элемент, Type тип_а)
где элемент обозначает объект класса MemberInfo, описывающий тот элемент, для которого создаются атрибуты, тогда как тип_а — требуемый атрибут.

Замечание

Первый метод определяется в классе MemberInfo и наследуется классом Type. Второй метод определяется в классе Attribute.

Два метода получения атрибутов объекта

[1] object[] GetCustomAttributes(bool наследование)
Если наследование имеет логическое значение true, то в список включаются атрибуты всех базовых классов, наследуемых по иерархической цепочке. В противном случае атрибуты извлекаются только из тех классов, которые определяются указанным типом.

[2] static Attribute.GetCustomAttribute(MemberInfo элемент, Type тип_а)
где элемент обозначает объект класса MemberInfo, описывающий тот элемент, для которого создаются атрибуты, тогда как тип_а — требуемый атрибут.

Замечание

Первый метод определяется в классе MemberInfo и наследуется классом Type. Второй метод определяется в классе Attribute.

Пример:

```
//Получить экземпляр объекта класса MemberInfo, связанного
//с классом, содержащим атрибут RemarkAttribute.
Type t = typeof(UseAttrib);
// Извлечь атрибут RemarkAttribute.
Type RemAt = typeof(RemarkAttribute);
RemarkAttribute ra = (RemarkAttribute)Attribute.GetCustomAttribute(t, RemAt);
```

Замечание

Первый метод определяется в классе MemberInfo и наследуется классом Type. Второй метод определяется в классе Attribute.

Пример:

```
//Получить экземпляр объекта класса MemberInfo, связанного
//с классом, содержащим атрибут RemarkAttribute.
Type t = typeof(UseAttrib);
// Извлечь атрибут RemarkAttribute.
Type RemAt = typeof(RemarkAttribute);
RemarkAttribute ra = (RemarkAttribute)Attribute.GetCustomAttribute(t, RemAt);
```

Упражнение 10.3

Реализовать программу демонстрирующую применение атрибута.

Общая форма

[attrib(список_позиционных_параметров,
именованный_параметр_1 = значение,
именованный_параметр_2 = значение, ...)]

Общая форма

[attrib(список_позиционных_параметров,
именованный_параметр_1 = значение,
именованный_параметр_2 = значение, ...)]

Замечание

Первыми указываются позиционные параметры, если они существуют. Далее следуют именованные параметры с присваиваемыми значениями. Порядок следования именованных параметров особого значения не имеет. Именованным параметрам не обязательно присваивать значение, и в этом случае используется значение, устанавливаемое по умолчанию.

Пример:

```
[AttributeUsage(AttributeTargets.All)]
public class RemarkAttribute : Attribute
{
    string pri_remark; // базовое поле свойства Remark
    // Это поле можно использовать в качестве именованного параметра.
    public string Supplement;
    public RemarkAttribute(string comment)
    {
        pri_remark = comment;
        Supplement = "Отсутствует";
    }
    public string Remark
    {
        get
        {
            return pri_remark;
        }
    }
}
```

Пример:

```
[AttributeUsage(AttributeTargets.All)]
public class RemarkAttribute : Attribute
{
    string pri_remark; // базовое поле свойства Remark
    // Это поле можно использовать в качестве именованного параметра.
    public string Supplement;
    public RemarkAttribute(string comment)
    {
        pri_remark = comment;
        Supplement = "Отсутствует";
    }
[RemarkAttribute("В этом классе используется атрибут.",
Supplement = "Это дополнительная информация.")]
class UseAttrib
{
    //...
}
```

Пример:

```
[AttributeUsage(AttributeTargets.All)]
public class RemarkAttribute : Attribute
{
    string pri_remark; // базовое поле свойства Remark
    // Это поле можно использовать в качестве именованного параметра.
    public string Supplement;
    public RemarkAttribute(string comment)
    {
        pri_remark = comment;
        Supplement = "Отсутствует";
    }
[RemarkAttribute("В этом классе используется атрибут.",
Supplement = "Это дополнительная информация.")]
class UseAttrib
{
    //...
}
```

Упражнение 10.4

Реализовать программу демонстрирующую применение атрибута с именованными параметрами (использовать поля и свойства).

Пример:

```
[AttributeUsage(AttributeTargets.All)]
public class RemarkAttribute : Attribute
{
    string pri_remark; // базовое поле свойства Remark
    // Это поле можно использовать в качестве именованного параметра.
    public string Supplement;
    public RemarkAttribute(string comment)
    {
        pri_remark = comment;
        Supplement = "Отсутствует";
    }
```

Замечание

Тип параметра атрибута (как позиционного, так и именованного) должен быть одним из встроенных простых типов, object, Type, перечислением или одномерным массивом одного из этих типов.

Атрибут AttributeUsage

Атрибут AttributeUsage определяет типы элементов, к которым может быть применен объявляемый атрибут. У него имеется следующий конструктор:
AttributeUsage(AttributeTargets validOn)
где validOn обозначает один или несколько элементов, к которым может быть применен объявляемый атрибут, тогда как AttributeTargets — перечисление, в котором определяются приведенные в Таблице 1 значения.

Атрибут AttributeUsage

Атрибут AttributeUsage определяет типы элементов, к которым может быть применен объявляемый атрибут. У него имеется следующий конструктор:
AttributeUsage(AttributeTargets validOn)
где validOn обозначает один или несколько элементов, к которым может быть применен объявляемый атрибут, тогда как AttributeTargets — перечисление, в котором определяются приведенные в Таблице 1 значения.

Таблица 1: Значения перечисления AttributeTargets

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

Атрибут Conditional

Атрибут Conditional позволяет создавать условные методы, которые вызываются только в том случае, если с помощью директивы #define определен конкретный идентификатор, а иначе метод пропускается.

Атрибут Conditional

Атрибут Conditional позволяет создавать условные методы, которые вызываются только в том случае, если с помощью директивы #define определен конкретный идентификатор, а иначе метод пропускается.

Замечание

Для применения атрибута Conditional в исходный код программы следует включить пространство имен System.Diagnostics.

Атрибут Conditional

Атрибут Conditional позволяет создавать условные методы, которые вызываются только в том случае, если с помощью директивы #define определен конкретный идентификатор, а иначе метод пропускается.

Замечание

Для применения атрибута Conditional в исходный код программы следует включить пространство имен System.Diagnostics.

Пример:

```
#define тест
using System;
using System.Diagnostics;
class Test
{
    [Conditional("тест")]
    void Method {...}
}
```

Атрибут Conditional

Атрибут Conditional позволяет создавать условные методы, которые вызываются только в том случае, если с помощью директивы #define определен конкретный идентификатор, а иначе метод пропускается.

Замечание

Для применения атрибута Conditional в исходный код программы следует включить пространство имен System.Diagnostics.

Пример:

```
#define тест
using System;
using System.Diagnostics;
class Test
{
    [Conditional("тест")]
    void Method {...}
}
```

Упражнение 10.5

Продемонстрировать применение встроенного атрибута Conditional.

Атрибут Obsolete

Атрибут Obsolete позволяет пометить элемент программы как устаревший. Вот общая форма этого атрибута:

```
[Obsolete("сообщение")]
```

где сообщение выводится при компилировании элемента программы, помеченного как устаревший.

Атрибут Obsolete

Атрибут Obsolete позволяет пометить элемент программы как устаревший. Вот общая форма этого атрибута:
[Obsolete("сообщение")]
где сообщение выводится при компилировании элемента программы, помеченного как устаревший.

Вторая форма атрибута Obsolete

[Obsolete("сообщение", ошибка)]
где ошибка обозначает логическое значение. Если это значение истинно (true), то при использовании устаревшего элемента формируется сообщение об ошибке компиляции вместо предупреждения.

Атрибут Obsolete

Атрибут Obsolete позволяет пометить элемент программы как устаревший. Вот общая форма этого атрибута:
[Obsolete("сообщение")]
где сообщение выводится при компилировании элемента программы, помеченного как устаревший.

Вторая форма атрибута Obsolete

[Obsolete("сообщение", ошибка)]
где ошибка обозначает логическое значение. Если это значение истинно (true), то при использовании устаревшего элемента формируется сообщение об ошибке компиляции вместо предупреждения.

Упражнение 10.6

Продемонстрировать применение атрибута Obsolete.

Обобщение

Обобщение это параметризированный тип. Обобщения позволяют создавать классы, структуры, интерфейсы, методы и делегаты, в которых обрабатываемые данные указываются в виде параметра.

Пример:

```
using System;
//В приведенном ниже классе Gen параметр типа T заменяется
//реальным типом данных при создании объекта типа Gen.
class Gen<T>
{
    T ob; //объявить переменную типа T
    public Gen(T o) //У конструктора имеется параметр типа T.
    {
        ob = o;
    }
    //Возвратить переменную экземпляра ob, которая относится к типу T.
    public T GetOb()
    {
        return ob;
    }
    public void ShowType()//Показать тип T.
    {
        Console.WriteLine("К типу T относится " + typeof(T));
    }
}
```

Упражнение 10.7

Продемонстрировать применение обобщенного класса.

Упражнение 10.7

Продемонстрировать применение обобщенного класса.

Упражнение 10.8

Написать "необобщенный" аналог класса Gen<T>* и переделать под него пример из предыдущего упражнения.
**Подсказка.* *Использовать переменную типа object.*

Упражнение 10.7

Продемонстрировать применение обобщенного класса.

Упражнение 10.8

Написать "необобщенный" аналог класса Gen<T>* и переделать под него пример из предыдущего упражнения.
**Подсказка.* *Использовать переменную типа object.*

Замечание

Обобщения позволяют создавать типизированный код, в котором ошибки несоответствия типов выявляются во время компиляции.

Обобщенный класс с несколькими параметрами

```
class имя_класса<список_параметров_типа> {//...}
```

Обобщенный класс с несколькими параметрами

```
class имя_класса<список_параметров_типа> {//...}
```

Общая форма объявления ссылки на обобщенный класс

```
имя_класса<список_аргументов_типа> имя_переменной =
new имя_класса<список_параметров_типа>(список_аргументов);
```

Обобщенный класс с несколькими параметрами

```
class имя_класса<список_параметров_типа> {//...}
```

Общая форма объявления ссылки на обобщенный класс

```
имя_класса<список_аргументов_типа> имя_переменной =
new имя_класса<список_параметров_типа>(список_аргументов);
```

Ограниченные типы

```
имя_класса<список_параметров_типа>
where параметр1_типа : ограничения
where параметр2_типа : ограничения
...
where параметрN_типа : ограничения { где ограничения указываются
списком через запятую.
```

В C# предусмотрен ряд ограничений на типы данных.

- Ограничение на базовый класс, требующее наличия определенного базового класса в аргументе типа. Это ограничение накладывается указанием имени требуемого базового класса. Разновидностью этого ограничения является неприкрытое ограничение типа, при котором на базовый класс указывает параметр типа, а не конкретный тип. Благодаря этому устанавливается взаимосвязь между двумя параметрами типа.
- Ограничение на интерфейс, требующее реализации одного или нескольких интерфейсов аргументом типа. Это ограничение накладывается указанием имени требуемого интерфейса.
- Ограничение на конструктор, требующее предоставить конструктор без параметров в аргументе типа. Это ограничение накладывается с помощью оператора new().
- Ограничение ссылочного типа, требующее указывать аргумент ссылочного типа с помощью оператора class.
- Ограничение типа значения, требующее указывать аргумент типа значения с помощью оператора struct.

В C# предусмотрен ряд ограничений на типы данных.

- Ограничение на базовый класс, требующее наличия определенного базового класса в аргументе типа. Это ограничение накладывается указанием имени требуемого базового класса. Разновидностью этого ограничения является неприкрытое ограничение типа, при котором на базовый класс указывает параметр типа, а не конкретный тип. Благодаря этому устанавливается взаимосвязь между двумя параметрами типа.
- Ограничение на интерфейс, требующее реализации одного или нескольких интерфейсов аргументом типа. Это ограничение накладывается указанием имени требуемого интерфейса.
- Ограничение на конструктор, требующее предоставить конструктор без параметров в аргументе типа. Это ограничение накладывается с помощью оператора new().
- Ограничение ссылочного типа, требующее указывать аргумент ссылочного типа с помощью оператора class.
- Ограничение типа значения, требующее указывать аргумент типа значения с помощью оператора struct.

В C# предусмотрен ряд ограничений на типы данных.

- Ограничение на базовый класс, требующее наличия определенного базового класса в аргументе типа. Это ограничение накладывается указанием имени требуемого базового класса. Разновидностью этого ограничения является неприкрытое ограничение типа, при котором на базовый класс указывает параметр типа, а не конкретный тип. Благодаря этому устанавливается взаимосвязь между двумя параметрами типа.
- Ограничение на интерфейс, требующее реализации одного или нескольких интерфейсов аргументом типа. Это ограничение накладывается указанием имени требуемого интерфейса.
- Ограничение на конструктор, требующее предоставить конструктор без параметров в аргументе типа. Это ограничение накладывается с помощью оператора new().
- Ограничение ссылочного типа, требующее указывать аргумент ссылочного типа с помощью оператора class.
- Ограничение типа значения, требующее указывать аргумент типа значения с помощью оператора struct.

В C# предусмотрен ряд ограничений на типы данных.

- Ограничение на базовый класс, требующее наличия определенного базового класса в аргументе типа. Это ограничение накладывается указанием имени требуемого базового класса. Разновидностью этого ограничения является неприкрытое ограничение типа, при котором на базовый класс указывает параметр типа, а не конкретный тип. Благодаря этому устанавливается взаимосвязь между двумя параметрами типа.
- Ограничение на интерфейс, требующее реализации одного или нескольких интерфейсов аргументом типа. Это ограничение накладывается указанием имени требуемого интерфейса.
- Ограничение на конструктор, требующее предоставить конструктор без параметров в аргументе типа. Это ограничение накладывается с помощью оператора new().
- Ограничение ссылочного типа, требующее указывать аргумент ссылочного типа с помощью оператора class.
- Ограничение типа значения, требующее указывать аргумент типа значения с помощью оператора struct.

В C# предусмотрен ряд ограничений на типы данных.

- Ограничение на базовый класс, требующее наличия определенного базового класса в аргументе типа. Это ограничение накладывается указанием имени требуемого базового класса. Разновидностью этого ограничения является неприкрытое ограничение типа, при котором на базовый класс указывает параметр типа, а не конкретный тип. Благодаря этому устанавливается взаимосвязь между двумя параметрами типа.
- Ограничение на интерфейс, требующее реализации одного или нескольких интерфейсов аргументом типа. Это ограничение накладывается указанием имени требуемого интерфейса.
- Ограничение на конструктор, требующее предоставить конструктор без параметров в аргументе типа. Это ограничение накладывается с помощью оператора new().
- Ограничение ссылочного типа, требующее указывать аргумент ссылочного типа с помощью оператора class.
- Ограничение типа значения, требующее указывать аргумент типа значения с помощью оператора struct.

В C# предусмотрен ряд ограничений на типы данных.

- Ограничение на базовый класс, требующее наличия определенного базового класса в аргументе типа. Это ограничение накладывается указанием имени требуемого базового класса. Разновидностью этого ограничения является неприкрытое ограничение типа, при котором на базовый класс указывает параметр типа, а не конкретный тип. Благодаря этому устанавливается взаимосвязь между двумя параметрами типа.
- Ограничение на интерфейс, требующее реализации одного или нескольких интерфейсов аргументом типа. Это ограничение накладывается указанием имени требуемого интерфейса.
- Ограничение на конструктор, требующее предоставить конструктор без параметров в аргументе типа. Это ограничение накладывается с помощью оператора new().
- Ограничение ссылочного типа, требующее указывать аргумент ссылочного типа с помощью оператора class.
- Ограничение типа значения, требующее указывать аргумент типа значения с помощью оператора struct.

Общая форма наложения ограничения на базовый класс

where T : имя_базового_класса
где T обозначает имя параметра типа, а имя_базового_класса — конкретное имя ограничиваемого базового класса.

Общая форма наложения ограничения на базовый класс

where T : имя_базового_класса
где T обозначает имя параметра типа, а имя_базового_класса — конкретное имя ограничиваемого базового класса.

Упражнение 10.9

На примере своих классов (A,B:A и C) продемонстрировать механизм наложения ограничения на базовый класс.

Общая форма наложения ограничения на базовый класс

where T : имя_базового_класса
где T обозначает имя параметра типа, а имя_базового_класса — конкретное имя ограничиваемого базового класса.

Упражнение 10.9

На примере своих классов (A,B:A и C) продемонстрировать механизм наложения ограничения на базовый класс.

Упражнение 10.10

Реализовать механизм управления списками телефонных номеров, чтобы можно было пользоваться разными категориями таких списков, в частности отдельными списками для друзей, поставщиков, клиентов.

Общая форма наложения ограничения на интерфейс

where T : имя_интерфейса
где T — это имя параметра типа, а имя_интерфейса — конкретное имя ограничиваемого интерфейса.

Общая форма наложения ограничения на интерфейс

where T : имя_интерфейса
где T — это имя параметра типа, а имя_интерфейса — конкретное имя ограничиваемого интерфейса.

Упражнение 10.11

Базовый класс для телефонных номеров (PhoneNumber) из предыдущего упражнения представить в форме интерфейса. Выполнить предыдущее упражнение используя данный интерфейс.

Ограничение new()

Ограничение new() на конструктор позволяет получать экземпляр объекта обобщенного типа.

Ограничение new()

Ограничение new() на конструктор позволяет получать экземпляр объекта обобщенного типа.

Пример:

```
class Test<T> where T : new()
{
    T obj;
    public Test()
    {
      //Этот код работоспособен благодаря наложению ограничения new().
      obj = new T(); //создать объект типа T
    }
    //...
}
```

Ограничение new()

Ограничение new() на конструктор позволяет получать экземпляр объекта обобщенного типа.

Пример:

```
class Test<T> where T : new()
{
    T obj;
    public Test()
    {
      //Этот код работоспособен благодаря наложению ограничения new().
      obj = new T(); //создать объект типа T
    }
    //...
}
```

Замечание

Ограничение new() можно использовать вместе с другими ограничениями, но последним по порядку. Ограничение new() позволяет конструировать объект, используя только конструктор без параметров. Ограничение new() нельзя использовать одновременно с ограничением типа значения.