



ОНЛАЙН-ОБРАЗОВАНИЕ


Онлайн-образование

Проверить, идет ли запись!





Меня хорошо видно && слышно?

Ставьте , если все хорошо
Напишите в чат, если есть проблемы

Атрибуты



Виктор Дзицкий

TeamLead, Full Stack .Net Developer

SolarLab

Telegram: @Dzitskiy

Правила вебинара



Активно участвуем



Задаем вопрос в чат или голосом



Off-topic обсуждаем в Slack #канал группы или #general



Вопросы вижу в чате, могу ответить не сразу

Маршрут вебинара

Знакомимся с атрибутами (заново)



Встроенные атрибуты



Кастомные атрибуты



Реализуем свои атрибуты

Цели вебинара | После занятия вы сможете

1

Использовать существующие атрибуты

2

Проверять наличие атрибутов на классах, функциях, полях и т.п.

3

Создавать свои собственные атрибуты

Смысл вебинара | Зачем вам это уметь

1

Освоить инструмент для написания понятного, гибкого и удобного в сопровождении кода

2

Понимать что делают встречающиеся в коде атрибуты

Тестовый пример

<https://github.com/Dzitskiy/AttributesExamples>

Вопросы

1. Готовы ли кодить сегодня на занятии?
2. Какие Атрибуты Вы уже знаете?


Атрибуты

Что такое атрибут?

- класс-наследник `System.Attribute`, добавляющий дополнительную информацию или функциональность к элементу перед которым находится.

По соглашению, название атрибута заканчивается на `Attribute` ("`ObsoleteAttribute`").

```
[Obsolete("This method is obsolete. Use M2 instead.")]  
public void M1(int x)  
{  
    Console.WriteLine($"x = {x}");  
}
```



Использование атрибутов

- **Аннотация метаданных.** Атрибуты позволяют добавлять метаданные к элементам программы, таким как классы, методы, свойства, параметры и т. д. Эти метаданные могут передавать дополнительную информацию об этих элементах, помогая в документации кода, анализе и поведении во время выполнения.
- **Организация и ясность кода.** Атрибуты помогают организовать и уточнить код, прикрепляя описательные метки или поведение к конкретным элементам. Например, такие атрибуты, как [Serializable] или [Obsolete], передают важную информацию о сериализации классов или устаревших методах соответственно, что делает код более понятным для разработчиков.

Использование атрибутов

- **Использование платформы.** Многие платформы и библиотеки .NET широко используют атрибуты для управления поведением или предоставления параметров конфигурации. Понимание того, как использовать и интерпретировать эти атрибуты, имеет решающее значение при работе с такими платформами, как ASP.NET, Entity Framework или ASP.NET Core.
- **Настройка и расширяемость.** Атрибуты позволяют разработчикам расширять функциональность среды выполнения .NET, создавая собственные атрибуты, адаптированные к их конкретным требованиям. Пользовательские атрибуты можно использовать для реализации сквозных задач, таких как ведение журнала, кэширование, проверка, авторизация и многое другое.

Использование атрибутов

- **Reflection и метапрограммирование.** Reflection позволяет разработчикам проверять код и манипулировать им во время выполнения, а атрибуты играют центральную роль в Reflection. Понимание того, как эффективно использовать атрибуты, позволяет разработчикам использовать Reflection для таких задач, как динамический вызов методов, запросы метаданных и обработка на основе атрибутов.
- **Аспектно-ориентированное программирование (АОП).** Атрибуты являются ключевым компонентом АОП, парадигмы программирования, которая позволяет разработчикам модулизировать сквозные задачи. Применяя атрибуты к элементам программы, разработчики могут добавлять в приложение дополнительное поведение или функциональность, не изменяя его основную логику.
- **Тестирование и отладка.** Атрибуты могут помочь в тестировании и отладке, предоставляя метаданные, которые влияют на выполнение теста или поведение отладки.

Применение атрибутов

```
[Serializable][Flags]  
public void Method() { }
```

```
[FlagsAttribute, SerializableAttribute]  
public void Method() { }
```

```
[Serializable, Flags]  
public void Method() { }
```

```
[FlagsAttribute()][Serializable()]  
public void Method() { }
```

```
[RemarkAttribute("This class uses an attribute.")]  
[RemarkAttribute("This class uses an attribute #2.")]  
class UseAttribute { }
```

Применение атрибутов

```
[AttributeUsage(AttributeTargets.All, AllowMultiple = true)]
public class RemarkAttribute : Attribute
{
    string _remark;

    public RemarkAttribute(string comment)
    {
        _remark = comment;
    }

    public string Remark
    {
        get { return _remark; }
    }
}
```


Применение атрибутов

```
/// <summary>
/// Ремонт и строительство
/// </summary>
[Cities(Cities.Moscow, Cities.Piter)]
[Order(1), Description("TaskCategories_Clerical", true, "Tasks")]
[TaskPrice(1100, 1350, 2500, 1200, 1500, 500)]
[TaskPrice(900, 1200, 2500, 1000, 1500, 500, "Piter")]
Clerical = 32768,
```

```
/// <summary>
/// Получение списка мастеров
/// </summary>
[HttpPost]
[Route("masters")]
[ProducesResponseType(typeof(MastersModel), (int)HttpStatusCode.OK)]
[ProducesResponseType((int)HttpStatusCode.NotFound)]
? usages ? overrides ? Марат Гайфутдинов +3 * ? ext methods ? exposing APIs
public async Task<ActionResult> GetMasters([FromBody] MastersRequest request)
{
```

К чему применяются атрибуты?

```
using System;

[assembly: SomeAttr] // Применяется к сборке
[module: SomeAttr] // Применяется к модулю

[type: SomeAttr] // Применяется к типу
internal sealed class SomeType<[typevar: SomeAttr] T> { // Применяется
                                                         // к переменной обобщенного типа

[field: SomeAttr] // Применяется к полю
public Int32 SomeField = 0;

[return: SomeAttr] // Применяется к возвращаемому значению
[method: SomeAttr] // Применяется к методу
public Int32 SomeMethod(
    [param: SomeAttr] // Применяется к параметру Int32 SomeParam
    Int32 SomeParam) { return SomeParam; }

[property: SomeAttr] // Применяется к свойству
public String SomeProp {
    [method: SomeAttr] // Применяется к механизму доступа get
    get { return null; } }

[event: SomeAttr] // Применяется к событиям
[field: SomeAttr] // Применяется к полям, созданным компилятором
[method: SomeAttr] // Применяется к созданным
                   // компилятором методам add и remove
public event EventHandler SomeEvent;
}
```

Вопросы для проверки

Что такое атрибут в C#?

Как атрибуты используются при разработке на C#?

Можете ли вы привести примеры сценариев, в которых обычно используются атрибуты?

Вопросы для проверки

Что такое атрибут в C#?

Атрибут в C# — это декларативный тег, который предоставляет метаданные об элементах программы, таких как классы, методы, свойства, параметры и т. д. Он позволяет разработчикам добавлять к этим элементам дополнительную информацию или поведение без изменения их основных функций.

Как атрибуты используются при разработке на C#?

Атрибуты используются для передачи метаданных об элементах программы, которые могут использоваться компиляторами, средами выполнения, платформами и инструментами. Они облегчают выполнение различных задач, таких как генерация кода, сериализация, проверка, документирование и многое другое.

Можете ли вы привести примеры сценариев, в которых обычно используются атрибуты?

Общие сценарии использования атрибутов включают пометку методов как устаревших ([Obsolete]), указание поведения сериализации ([Serializable]), определение шаблонов маршрутов в ASP.NET MVC ([Route]), настройку внедрения зависимостей ([Inject]) и указание требования авторизации ([Authorize]).

Маршрут вебинара

Знакомимся с атрибутами (заново)



Встроенные атрибуты



Кастомные атрибуты



Реализуем свои атрибуты

.NET Core

- **[Serializable]**: помечает класс как сериализуемый, позволяя сериализовать и десериализовать экземпляры класса.
- **[DataContract]** , **[DataMember]**: используются для определения контрактов данных в службах WCF (Windows Communication Foundation) для сериализации и десериализации данных
- **[JsonProperty]** (Newtonsoft.Json or System.Text.Json.Serialization): позволяет указывать имена свойств JSON во время сериализации и десериализации объектов.
- **[Required]** ([ValidationAttribute])(System.ComponentModel.DataAnnotations): указывает, что свойство должно иметь значение во время проверки модели.
- **[Obsolete]**: помечает элемент (например, свойство, метод) как устаревший или устаревший, указывая, что его больше не следует использовать.

.NET Core

- **[Conditional]**: позволяет условно компилировать методы на основе указанных символов предварительной обработки. Это позволяет разработчикам включать или исключать определенные методы из скомпилированного вывода в зависимости от определенных символов, что может быть полезно для управления кодом, специфичным для отладки, или реализации переключения функций.
- **[Flags]**: указывает, что перечисление представляет собой набор битовых полей или флагов. Это позволяет разработчикам определять перечислимые типы, в которых каждое значение перечисления представляет одну битовую позицию, что позволяет объединять несколько значений перечисления в одно значение с помощью побитовых операций.

AspNetCore.Mvc

- **[Route]** : определяет шаблон маршрута для действия контроллера MVC или обработчика Razor Pages.
- **[HttpGet]**, **[HttpPost]**, **[HttpPut]**, **[HttpDelete]**: HTTP-глаголы маршрутизации на основе атрибутов, используемые для указания метода HTTP для методов действия контроллера.
- **[ApiController]** : помечает контроллер как контроллер API, обеспечивая такие функции, как автоматическая проверка модели и автоматические ответы HTTP 400.
- **[FromQuery]**, **[FromRoute]**, **[FromBody]**, **[FromHeader]**: используется для привязки модели в ASP.NET Core MVC для привязки данных из разных источников (строка запроса, данные маршрута, тело запроса, заголовки) для параметров метода действия.

AspNetCore.Mvc

- **[Produces] [Consumes]** : используются для указания типов мультимедиа, которые действие контроллера может создавать или потреблять.
- **[BindProperty]**: используется в Razor Pages для указания того, что свойство должно быть привязано к данным входящего запроса.
- **[Authorize]** (Microsoft.AspNetCore.Authorization): указывает, что для доступа к контроллеру или методу действия требуется проверка подлинности.

Сколько у этого класса атрибутов?



30 секунд

```
class A { }
```

- **AutoLayout** - автоматическое размещение содержимого класса в памяти. Другие варианты - **ExplicitLayout** и **SequentialLayout**.

Размещение в памяти

```
public struct NotAlignedStruct
{
    public byte m_byte1;
    public int m_int;

    public byte m_byte2;
    public short m_short;
}
```

Size: 12. Paddings: 4 (%33 of empty space)

=====	
0:	Byte m_byte1 (1 byte)

1-3:	padding (3 bytes)

4-7:	Int32 m_int (4 bytes)

8:	Byte m_byte2 (1 byte)

9:	padding (1 byte)

10-11:	Int16 m_short (2 bytes)
=====	

Размещение в памяти

```
[StructLayout(LayoutKind.Auto)]  
public struct NotAlignedStructWithAutoLayout  
{  
    public byte m_byte1;  
    public int m_int;  
  
    public byte m_byte2;  
    public short m_short;  
}
```

Size: 8. Paddings: 0 (%0 of empty space)

```
|=====|  
| 0-3: Int32 m_int (4 bytes) |  
|-----|  
| 4-5: Int16 m_short (2 bytes) |  
|-----|  
| 6: Byte m_byte1 (1 byte) |  
|-----|  
| 7: Byte m_byte2 (1 byte) |  
|=====|
```


Сколько у этого класса атрибутов?

```
class A { }
```

- **AutoLayout** - автоматическое размещение содержимого класса в памяти. Другие варианты - **ExplicitLayout** и **SequentialLayout**.
- **AnsiClass** - означает, что при маршаллинге класса строки в нём необходимо интерпретировать как ANSI. Другие варианты - **UnicodeClass**, **AutoClass** и **CustomFormatClass**.
- **Class** - обозначение, что этот объект является классом.
- **NestedPrivate** - вложенный и приватный.
- **BeforeFieldInit** - означает, что объект класса может быть создан заранее, т.е. до того, как к нему обратились.

Сколько у этого метода атрибутов?

```
public void M1(int x)
{
    Console.WriteLine($"x = {x}");
}
```

- **PrivateScope** - указывает, что невозможно создать ссылку на этот член
- **Public** - метод находится в публичном доступе.
- **HideBySig** - указывает, что метод скрывается на основе оценки имени и сигнатуры; в обратном случае метод скрывается только по имени.

А сколько атрибутов здесь? ;-)

```
[Serializable]
class TestClass
{
    private int _a;

    private bool B { get; set; }

    public void M2(int x = 15)
    {
        Console.WriteLine($"x = {x}");
    }

    public void M3(params int[] x)
    {
        Console.WriteLine($"x = {x.Length}");
    }

    [Obsolete("This method is obsolete. Use method New() instead.")]
    public void Obsolete() {}

    [Conditional("TEST")]
    public void C()
    {
        Console.WriteLine("Test passed");
    }
}
```

Атрибут [Flags]

```
[Flags]
internal enum Weekdays
{
    None          = 0x0000, // Зачем None?
    Monday        = 0x0001,
    Tuesday       = 0x0002,
    Wednesday     = 0x0004,
    Thursday      = 0x0008,
    Friday        = 0x0010, // Почему 10, ведь должны быть степени двойки?
    Saturday      = 0x0020,
    Sunday        = 0x0030,
    Workdays     = Monday | Tuesday | Wednesday | Thursday | Friday,
    Holidays      = Saturday | Sunday,
    All           = Workdays | Holidays
}
```


Где используются атрибуты?

- Тестирование (xUnit, nUnit)
- Валидация
- Сериализация
- ASP.NET Controllers
- etc.

Задание

Создать перечисление для Ролей (Roles) в котором есть роли со стороны Клиента (User, Customer, Moderator), и со стороны саппорта (Admin, Employee, Manager) для удобного использования.

Вопросы для проверки

Объясните назначение атрибута `[Obsolete]`.

Как и почему вы будете использовать его в своем коде?

Что делает атрибут `[Serializable]` и когда его можно использовать?

Обсудите значение атрибута `[Conditional]` в C#.

Вопросы для проверки

Объясните назначение атрибута [Obsolete]. Как и почему вы будете использовать его в своем коде?

Атрибут [Obsolete] помечает программный элемент (обычно метод или свойство) как устаревший, указывая, что он устарел или больше не должен использоваться. При его применении компилятор генерирует предупреждение или ошибку, чтобы предупредить разработчиков об устаревшем элементе, побуждая их соответствующим образом обновить свой код.

Что делает атрибут [Serializable] и когда его можно использовать?

Атрибут [Serializable] помечает класс как сериализуемый, позволяя конвертировать экземпляры класса в поток байтов для хранения или передачи. Этот атрибут обычно используется в распределенных системах, операциях файлового ввода-вывода и сценариях сохранения объектов.

Обсудите значение атрибута [Conditional] в C#.

Атрибут [Conditional] позволяет условно компилировать методы на основе указанных символов предварительной обработки. Это позволяет разработчикам включать или исключать определенные методы из скомпилированного вывода в зависимости от определенных символов, что может быть полезно для управления кодом, специфичным для отладки, или реализации переключения функций.

Маршрут вебинара

Знакомимся с атрибутами (заново)



Встроенные атрибуты



Кастомные атрибуты



Реализуем свои атрибуты

Типы которые можно использовать

В C# атрибуты могут иметь свойства, также называемые параметрами. Однако существуют ограничения на типы, которые можно использовать для этих свойств.

Возможные типы, которые можно использовать в качестве свойств атрибутов:

- Примитивные типы: `bool`, `byte`, `char`, `short`, `int`, `long`, `float`, `double`
- `string:string`
- Enum (Перечисления): Любой тип перечисления
- Тип: `System.Type`
- Массивы объектов:

Одномерные массивы любого из вышеперечисленных типов, включая массивы этих типов (`int[]`, `string[]`, `bool[]`, `Type[]` и т. д.)

Почему только такие

Атрибуты предназначены для хранения метаданных о коде, и эти метаданные должны быть доступны во время компиляции, чтобы их можно было внедрить в сборку и впоследствии использовать инструментами и платформами, которые читают эти метаданные во время выполнения.

Типы, разрешенные в качестве параметров атрибута (примитивные типы, строки, перечисления, `Туре` и статические массивы), — это все типы, которые могут быть представлены как константы во время компиляции.

В C# экземпляр класса атрибута создается средой выполнения .NET, когда это необходимо, обычно когда отражение используется для проверки атрибутов, примененных к элементам кода (таким как классы, методы, свойства и т. д.). (Такие методы, как `GetCustomAttributes`, `IsDefined` и `GetCustomAttribute`, запускают создание экземпляров классов атрибутов.)

Пример Кастомного Атрибута

```
[RemarkAttribute("This class uses an attribute.")]  
class UseAttribute { }
```

```
[AttributeUsage(AttributeTargets.All)]  
public class RemarkAttribute : Attribute  
{  
    string _remark;  
  
    public RemarkAttribute(string comment)  
    {  
        _remark = comment;  
    }  
  
    public string Remark  
    {  
        get { return _remark; }  
    }  
}
```


Области применения

```
[assembly: SomeAttr]    // Applied to assembly
[module: SomeAttr]      // Applied to module

[type: SomeAttr]        // Applied to type
class SomeType<[typevar: SomeAttr] T> // Applied to generic type variable
{
    [field: SomeAttr]    // Applied to field
    public int SomeField = 0;

    [return: SomeAttr]   // Applied to return value
    [method: SomeAttr]   // Applied to method
    public int SomeMethod([param: SomeAttr] int SomeParam) // Applied to parameter
    {
        return SomeParam;
    }

    [property: SomeAttr] // Applied to property
    public String SomeProp
    {
        [method: SomeAttr]
        get { return null; } // Applied to get accessor method
    }

    [event: SomeAttr]    // Applied to event
    [field: SomeAttr]    // Applied to compiler-generated field
    [method: SomeAttr]   // Applied to compiler-generated add & remove methods
    public event EventHandler SomeEvent;
}
```

Области применения

```
[Flags, Serializable]
public enum AttributeTargets
{
    Assembly           = 0x0001,
    Module             = 0x0002,
    Class              = 0x0004,
    Struct              = 0x0008,
    Enum               = 0x0010,
    Constructor         = 0x0020,
    Method              = 0x0040,
    Property            = 0x0080,
    Field               = 0x0100,
    Event               = 0x0200,
    Interface           = 0x0400,
    Parameter           = 0x0800,
    Delegate            = 0x1000,
    ReturnValue         = 0x2000,
    GenericParameter    = 0x4000,
    All                 = Assembly | Module | Class | Struct | Enum | Constructor
                      | Method | Property | Field | Event | Interface | Parameter | Delegate | ReturnValue | GenericParameter
}
```

Способы получения атрибутов

- IsDefined*
- GetCustomAttribute
- GetCustomAttributes
- GetCustomAttributesData

```
typeof(UseAttribute).IsDefined(typeof(RemarkAttribute)).Dump();  
(typeof(UseAttribute).GetCustomAttribute(typeof(RemarkAttribute)) as RemarkAttribute).Remark.Dump();  
(typeof(UseAttribute).GetCustomAttributes(typeof(RemarkAttribute)).Single() as RemarkAttribute).Remark.Dump();  
typeof(UseAttribute).GetCustomAttributesData().Single(x => x.AttributeType == typeof(RemarkAttribute)).ConstructorArguments.First().Value.Dump();
```

Results SQL IL Tree

Format Export

True
This class uses an attribute.
This class uses an attribute.
This class uses an attribute.

Где используются кастомные атрибуты?

- Кастомная валидация
- Аспектно-ориентированное программирование
- Собственные мапперы
- Любые места, где нужно работать с классами, а не их экземплярами.

Задание

1. Создать атрибут для валидации номера банковской карты
2. Создать Атрибут для проверки авторизации Ролей (Roles) и применить его так, что контроллер ClientController доступен только стороне клиента (User, Customer, Moderator). При этом метод AddUser доступен только модератору. SupportController доступен только Саппорту (Admin, Employee, Manager). При этом метод AddEmployee доступен менеджеру и админу.

Вопросы для проверки

Что такое пользовательские атрибуты и зачем их создавать?

Можете ли вы объяснить шаги, необходимые для создания настраиваемого атрибута?

Приведите пример сценария, в котором вы будете использовать настраиваемый атрибут в проекте C#.

Вопросы для проверки

Что такое пользовательские атрибуты и зачем их создавать?

Пользовательские атрибуты — это определяемые пользователем теги метаданных, созданные на основе базового класса `System.Attribute`. Разработчики создают пользовательские атрибуты, чтобы аннотировать свой код метаданными, специфичными для предметной области.

Можете ли вы объяснить шаги, необходимые для создания настраиваемого атрибута?

Чтобы создать настраиваемый атрибут, вы определяете новый класс, который наследуется от `System.Attribute`, и украшаете его атрибутом `[AttributeUsage]`, чтобы указать его целевые элементы. При желании вы можете определить свойства, поля или методы в классе атрибутов для хранения дополнительной информации.

Приведите пример сценария, в котором вы будете использовать настраиваемый атрибут в проекте C#.

Распространенным сценарием использования настраиваемого атрибута является реализация правил декларативной авторизации в веб-приложении. Вы можете создать собственный атрибут `[AuthorizeRole]`, который принимает параметр роли и применяет его к действиям или методам контроллера, упрощая логику управления доступом и повышая читаемость кода.

Маршрут вебинара

Знакомимся с атрибутами (заново)



Встроенные атрибуты



Кастомные атрибуты



Реализуем свои атрибуты




Практика

The background of the slide is an aerial photograph of a city skyline, likely New York City, with numerous skyscrapers. The entire image is filtered with a blue color scheme. A horizontal band across the middle of the image features a network of white lines connecting dots, creating a digital or web-like pattern. The word "Тестирование" is written in large, bold, white Cyrillic letters across this band.

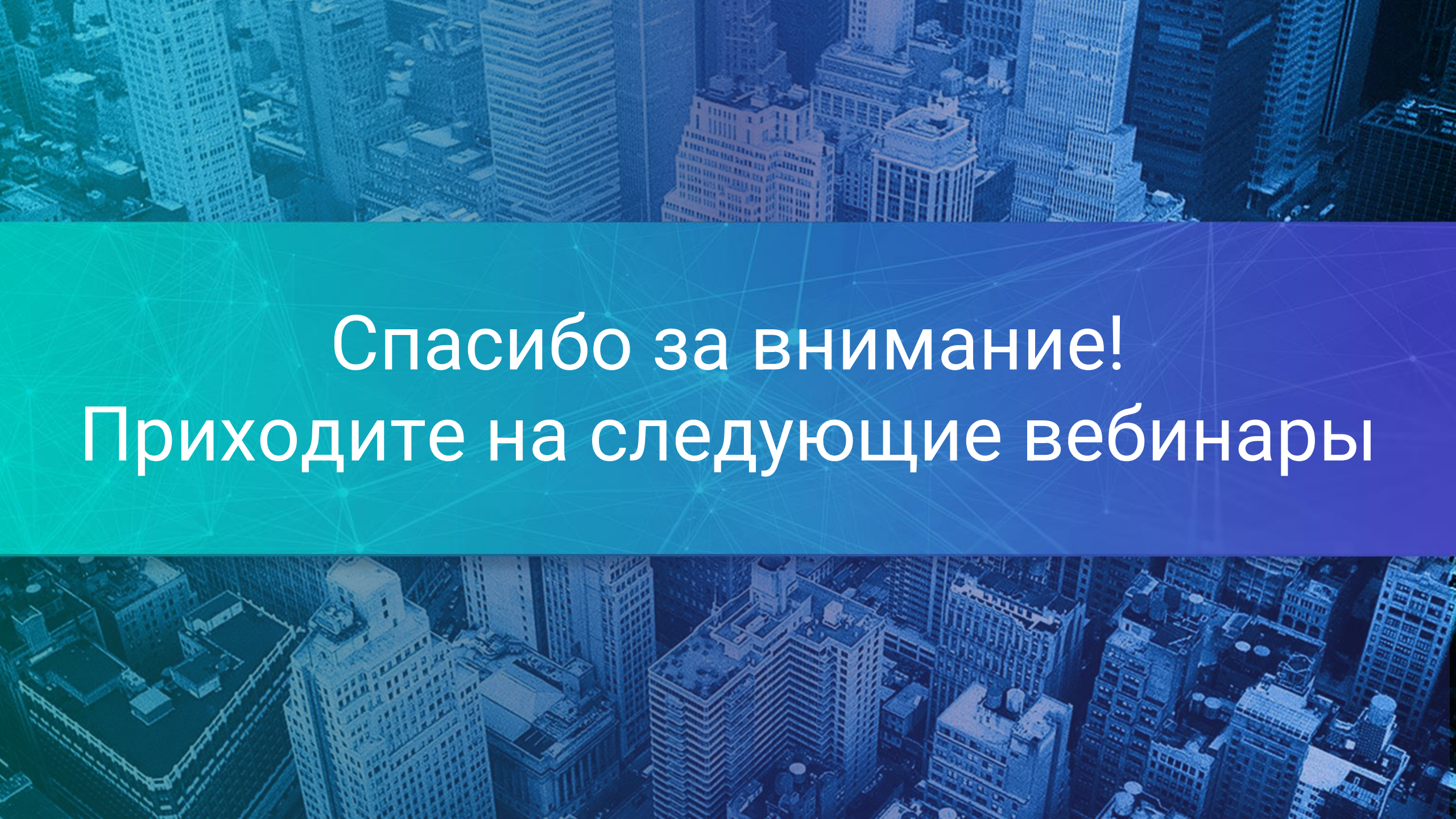
Тестирование

Полезные ссылки

- <https://learn.microsoft.com/en-us/dotnet/csharp/advanced-topics/reflection-and-attributes/>
- <http://csharpindepth.com/Articles/General/Beforefieldinit.aspx> - Статья, где Джон Скит рассуждает про поведение BeforeFieldInit.
- <https://github.com/SergeyTeplyakov/ObjectLayoutInspector> - утилита для просмотра расположения объектов в памяти.
- <http://sergeyteplyakov.blogspot.com/> - блог Сергея Теплякова, где ещё много всего интересного.
- <https://referencesource.microsoft.com/#mscorlib/system/reflection/methodattributes.cs,22> - Исходный код с PrivateScope.
- <https://www.youtube.com/watch?v=zAh0xraBsi0> - Обсуждение изменений в атрибутах в Microsoft

The background of the image is an aerial photograph of a city skyline, likely New York City, with numerous skyscrapers. The image is overlaid with a semi-transparent blue layer that features a white geometric network pattern of interconnected lines and dots. The text is centered within this blue layer.

Заполните, пожалуйста,
опрос о занятии по ссылке в чате

The background of the entire image is an aerial photograph of a city skyline, likely New York City, with numerous skyscrapers. The image is overlaid with a semi-transparent blue layer that features a white geometric network pattern of dots and lines. The text is centered within this blue layer.

Спасибо за внимание!
Приходите на следующие вебинары