

您的潜力，我们的动力

**Microsoft**  
微软(中国)有限公司

# C#锐利体验2.0:

## 杂项技术，以及C#语言的未来发展

李建忠

[www.lijianzhong.com](http://www.lijianzhong.com)

上海祝成科技 高级讲师

# Agenda

- 属性访问器的保护级别
- 命名空间别名限定符
- Pragma 指示符
- Conditional特性类
- 定长buffer
- C# 3.0与未来发展
- Q&A

# 属性访问器保护级别的变化

- C# 2.0允许我们对一个属性的get和set访问器使用不同的访问级别修饰符：

```
class Customer {  
    private string name;  
    .....  
    public string Name {  
        get { return this.name; }  
        internal set { this.name=value;}  
    }  
    .....  
}
```

## 一些注意点

- 属性访问器（**get**或**set**）上应用的访问修饰符 必须“小于”属性上应用的访问修饰符；“小于”的意思即“更严格”，例如**protected**小于**public**。
- 只能在一个属性访问器（**get**或**set**）上指定 比 属性上的访问修饰符“更小”的访问修饰符。
- 对于接口中属性的声明，不能给属性访问器（**get**或**set**）指定任何访问修饰符，只能默认为**public**。
- 属性访问器保护级别的变化规则完全适用于C#的索引器。



# Agenda

- 属性访问器的保护级别
- 命名空间别名限定符
- Pragma 指示符
- Conditional特性类
- 定长buffer
- C# 3.0与未来发展
- Q&A

# 命名空间别名限定符的引入

- **C# 2.0**允许我们使用命名空间别名限定符 (::) 来避免不同命名空间中类型名称冲突的问题：

```
using SC = System.Collections;
using ZC = Zhucheng.Collections;
class Program {
    static void Main() {
        ZC::ArrayList list = new ZC::ArrayList();
        ...
    }
}
```

## 一些注意点

- 当使用命名空间别名限定符 (::) 时，如 `ZC::ArrayList`，编译器可以确保这是一个只适用于“命名空间别名”的限定符，不会辨析为其他类型、或者成员限定符 (.)。
- 关键字 `global` 可以放在命名空间别名限定符 (::) 的左边，它使得编译器只去搜索那些所有的命名空间，而不会去搜索其他的类型、或者成员。
- 尽可能地使用命名空间别名限定符 (::)，而减少使用点号 (.) 这样的通用限定符。

# Agenda

- 属性访问器的保护级别
- 命名空间别名限定符
- Pragma 指示符
- Conditional特性类
- 定长buffer
- C# 3.0与未来发展
- Q&A



# Pragma 指示符的引入

- C# 2.0允许我们使用命名空间别名限定符 (::) 来避免不同命名空间中类型名称冲突的问题:

```
class Program{  
    [Obsolete]  
    static void Foo() { ..... }  
    static void Main() {  
        #pragma warning disable 612  
        Foo();  
        #pragma warning restore 612  
    }  
}
```

## 几个注意点

- 目前Pragma 指示符只支持# pragma warning
- #pragma warning disable可以禁掉任何编译器警告信息。
- #pragma warning restore可以恢复被disable掉的任何编译器警告信息。
- 可以在disable和restore后面跟上具体的警告代码号，从而来禁止或者恢复 特定的警告信息。
- #pragma 是一个编译预处理功能，不影响任何代码运行机制。

# Agenda

- 属性访问器的保护级别
- 命名空间别名限定符
- Pragma 指示符
- Conditional特性类
- 定长buffer
- C# 3.0与未来发展
- Q&A

# Conditional特性类的引入

- C# 2.0允许我们使用Conditional特性类来告诉编译器根据“特定的预定义指示符条件”来在类上应用特性：

```
#define DEBUG
using System;
using System.Diagnostics;

[Conditional("DEBUG")]
public class TestAttribute : Attribute {}

[Test]
class MyClass {}
```



## 几个注意点

- 如果定义了条件指示符，如`#define DEBUG`，那么编译器将在`MyClass`类上应用`TestAttribute`特性。
- 如果没有定义条件指示符，如`#undef DEBUG`，`MyClass`类照样可以正常使用，但是其上将不再应用`TestAttribute`特性。
- 注意区别C# 2.0中的`Conditional` 特性类和C# 1.0中的`Conditional`特性方法。

# Agenda

- 属性访问器的保护级别
- 命名空间别名限定符
- Pragma 指示符
- Conditional特性类
- 定长buffer
- C# 3.0与未来发展
- Q&A

## 定长buffer的引入

- C# 2.0引入定长buffer来使得我们可以在结构里声明C风格的数组，从而更加方便地实现托管代码和非托管代码的互操作：

```
unsafe struct MyClass {  
    public fixed int x[5];  
    public fixed int y[10];  
    public fixed int z[100];  
}
```

## 几个注意点

- 定长**buffer**只能使用在**unsafe**代码的上下文中，不可以非**unsafe**的代码中使用。
- 使用定长**buffer**所定义的字段在结构类型的实例对象中将按照它们的声明顺序来进行内存布局。
- 注意区别**unsafe**代码中的定长**buffer**和我们通常使用的托管数组。
- 定长**buffer**主要应用在托管代码和非托管代码互操作的情况，除此之外，我们一般使用托管数组。



# Agenda

- 属性访问器的保护级别
- 命名空间别名限定符
- Pragma 指示符
- Conditional特性类
- 定长buffer
- C# 3.0与未来发展
- Q&A

# C# 3.0与未来发展


- C# 2.0的核心机制在于泛型编程的引入，它赋予了类型参数式多态的能力，将对今后的C#代码构造有重要影响。
- 研发中的C# 3.0将XML, SQL两种数据处理技术引入到C#这样的强类型语言中，极大地丰富了C#语言的数据处理能力，是一个极具远见的创新。
- C#语言的发展越来越多地体现 融合“设计模式+库”的思想。“语言的发展就是库的发展！”

# Agenda


- 属性访问器的保护级别
- 命名空间别名限定符
- Pragma 指示符
- Conditional特性类
- 定长buffer
- C# 3.0与未来发展
- Q&A

## Question & Answer

如需提出问题，请单击“提问”按钮并在随后显示的浮动面板中输入问题内容。一旦完成问题输入后，请单击“提问”按钮。

 **问题和解答 (无问题)** ▲ ×

在此会议中尚未解答任何问题。

要向演示者提问，请在此处键入问 

提问(A)

删除(D)

问题管理器(Q)



您的潜力，我们的动力

**Microsoft®**  
微软(中国)有限公司

**Microsoft®**

msdn  


**MSDN Webcasts**