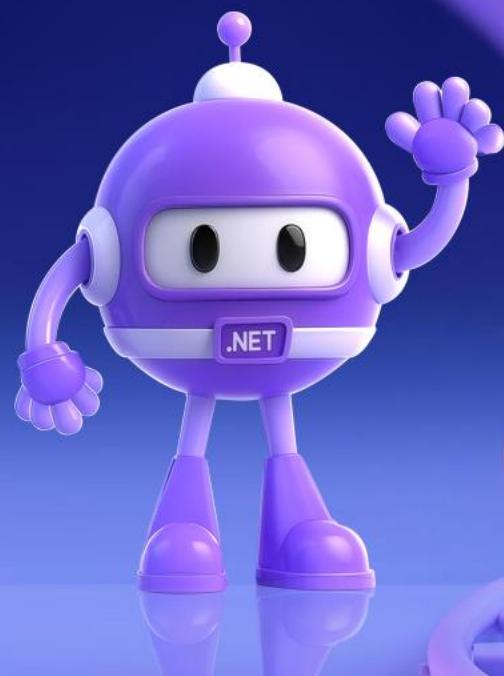


.NET Conf China 2025

改变世界 改变自己

2025 年 11 月 30 日 | 中国 上海



C#14中新的扩展方法语法 带来的崭新面貌



汪好盛
Microsoft MVP /
Pokemonisshoni开发者





全新的扩展方法语法

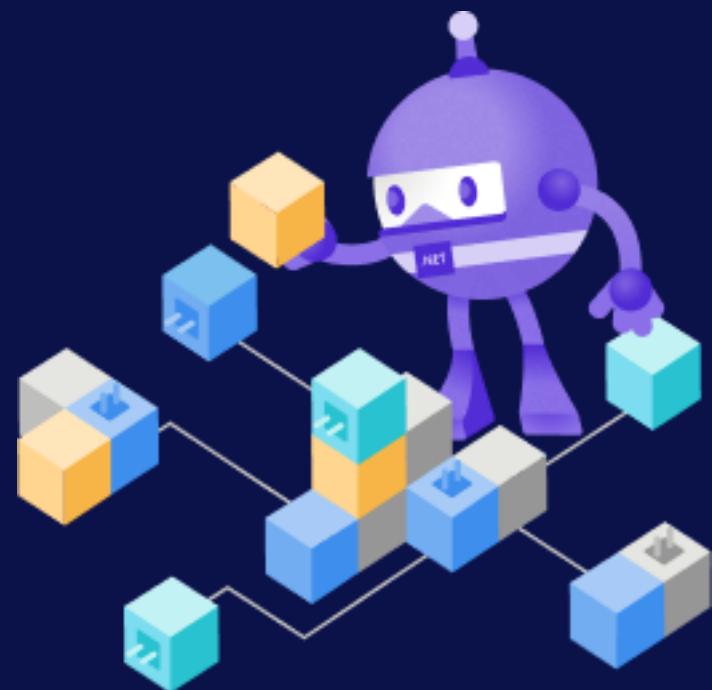
- 早在C#3时，我们就拥有了扩展方法
- 终于如今发布的C#14中，我们迎来了扩展方法一个非常重要的更新
- 扩展块语法更新以及其能力都得到了加强
- 得益于这些的加强，我们能获得更清晰的代码结构的同时，也能实现一些全新的可能性



新语法 扩展块

- 原本的语法中，我们需要在一个静态类中定义静态的扩展方法，还需要把方法的第一个参数设为this修饰的参数，我们才正常定义了一个扩展方法。这其实是噪音很大的一件事。
- 同时初学者看到的时候可能也会一头雾水，“欸，这个方法是怎么回事？，怎么这里写了那边就突然能用了？”
- 不过在C#14中，我们引入了全新的语法

```
public static class OldExtension
{
    public static int Add(this int x, int y) => x + y;
}
```



新语法 扩展块

- 首先我们将使用明确意义的extension关键字，声明以下将会是一个扩展方法。这时我们发现

- 好像代码反而变多了???

- 好吧我坦白新语法只是为了让我们多写两行代码

```
public static class NewExtension
{
    extension(int x)
    {
        public int Add2(int y)
        {
            return x + y;
        }
    }
}
```





新语法 扩展块

- 首先我们将使用明确的关键字，会是一系列扩展方法，并且同一方法将在同一个扩展块中。
- 其次，我们不再需要再每个方法添加static修饰
- 之前所有this修饰的参数不再需要入参之中，它被提升到了扩展块
- 当扩展块中的方法越来越多时，码量也会显著减少
- 泛型参数也理所应当的可以提升

```
public static TSource First<TSource>(this IEnumerable<TSource> source)
{
    foreach (var item in source)
    {
        return item;
    }
    throw new InvalidOperationException("Sequence contains no elements");
}

extension<TSource>(IEnumerable<TSource> source)
{
    public TSource First1()
    {
        foreach (var item in source)
        {
            return item;
        }
        throw new InvalidOperationException("Sequence contains no elements");
    }

    public IEnumerable<TResult> Select<TResult>(Func<TSource, TResult> selector)
    {
        foreach(var item in source)
        {
            yield return selector(item);
        }
    }
}
```

新语法 扩展块

- 当然，也可以直接定义类型的静态扩展方法

```
int.MyRange(1, 3);
IEnumerable<int>.MyRange(1, 3);
```

```
extension(int x)
{
    public int Add2(int y) => x + y;
    public static IEnumerable<int> MyRange(int start, int end)
    {
        for (int i = start; i < end; i += 1)
        {
            yield return i;
        }
    }
}

extension<T>(IEnumerable<T>) where T: INumber<T>
{
    public static IEnumerable<T> MyRange(T start, T end)
    {
        for (T i = start; i < end; i += T.One)
        {
            yield return i;
        }
    }
}
```

扩展属性

- 得益于更统一的语法，更多的扩展功能也得以实现
- 扩展属性也是其一
- 不过目前set访问器其实没有办法去真正意义上的赋值一个新的幕后变量，仅仅能让其拥有进行赋值操作的可能性，未来存在这个功能的可能性也并不高。

```
extension<TSource>(IEnumerable<TSource> source)
{
    public TSource FirstProp
    {
        get
        {
            foreach (var item in source)
            {
                return item;
            }
            throw new
                InvalidOperationException
                ("Sequence contains no elements");
        }
        set { }
    }
}
```



扩展运算符

- 与扩展属性同样的理由，扩展运算符在 C#14 中也得到了支持
- 这也是扩展方法的另一种写法，当你不需要使用变量时，你可以仅仅只声明其类型。
- 扩展运算符可以很好的给一些我们无法修改，又不想创建新类型的类型，构建独有的运算

```
extension<T>(IEnumerable<T>) where T: INumber<T>
{
    public static IEnumerable<T>
        operator *(IEnumerable<T> source, T scalar)
        => source.Select(s => s * scalar);
}

extension<T>(T[] array) where T: INumber<T>
{
    public void operator *=(T scalar)
    {
        for (int i = 0; i < array.Length; i++)
        {
            array[i] = array[i] * scalar;
        }
    }
}
```

```
Console.WriteLine(string.Join(", ", v2));
// output: 3, 6, 9
```

```
var arr4 = Enumerable.Range(1, 3).ToArray();
arr4 *= 3;
```

```
Console.WriteLine(string.Join(", ", arr4));
// output: 3, 6, 9
```



更多的可能性...

- 扩展运算符，可能比想象中的意义更为重大，
我们甚至可以编写出如下代码

```
_ = "Hello, World!" >> Console.WriteLine;
```

```
extension<T>(T)
{
    public static T operator >>(T source, Action<T> action)
    {
        action(source);
        return source;
    }
}
```

```
var add = (int x, int y) => x + y;
var add3 = (int x, int y, int z) => x + y + z;

var addCurry = (int x) => (int y) => x + y;
var add3Curry = (int x) => (int y) => (int z) => x + y + z;

_ = 1
>> add << 2
>> add3 << 3 << 3
>> add3Curry << 7 << 9
>> Console.WriteLine;

// output: 25
```



更多的可能性…

- 以上代码模拟了一种C#未曾出现过的运算符，管道运算符。但这类运算符在支持函数式编程的语言中是非常常见的（例如F#）
- 扩展运算符让我们有足够的能力直接用现有的运算符一定程度上的实现这一操作。
- 在c#中曾经只能通过Pipe的扩展方法实现类似的操作，而如今我们能将其直接化为运算符，更方便的使用它



更多的可能性…

- 现代C#发展中，不断的在吸收函数式编程中一些优秀的特性，例如记录，模式匹配，甚至扩展方法本身也一定程度上与管道有关。（右侧为F#的代码）
- 目前 C# 不允许自定义新的运算符，因此只能用已有符号（如 |, >>, << 等）来模拟，但这些技巧依然提供了非常大的表达空间，让库作者可以根据需要构建接近领域语言（DSL）的写法
- 不过当然，这可能不是该类语法的最佳实现，在一些特殊的领域使用尚可，直接在通用开发中使用那都算得上防御性编程了（笑）

```
let add x y = x + y
let add3 x y z = x + y + z

1
▷ add 2
▷ add3 3 3
▷ add3 7 9
▷ printfn "Result: %d"
```



更多的可能性...

```
var arr2Linq = arr.Where(isMod(2));  
  
var arr3 = arr  
    >> filter << (isMod << 2)  
    >> select << (x => x * 10)  
    >> joinInts << ", "  
    >> Console.WriteLine  
;  
// output: 20, 40, 60, 80  
  
_ = 1 >> Add1 >> Console.WriteLine;  
// output: 2  
  
  
int result = 1 >> (x => x * 2)  
    >> Console.WriteLine  
    >> (x => x - 3)  
    >> Console.WriteLine;  
// output: 2  
// output: -1  
  
var someData = Option<int>.Some(5)  
    * (x => x + 1)  
    * (x => x * 2)  
    >> Console.WriteLine  
;  
// output: SomeOption { Value = 12 }  
  
var s = OptionExtensions.Bind<int, int>;  
var mod = ((int x) => x % 2 == 1 ? Option<int>.Some(x) : Option<int>.None());  
var cc =  
    Option<int>.Some(5)  
    * (x => x * 2)  
    >> s << mod  
    >> Console.WriteLine;  
// output: NoneOption { }
```



扩展块的未来

- 扩展块将会是一个长期更新的功能项，在未来可能会有类似扩展索引器等更多的内容
- 也许明年，C#就会迎来一次更激进的变革
- Future Awaits !

演讲代码

<https://github.com/ssccinng/NETConf2025ChinaCode>

.NET Conf China 2025

改变世界 改变自己

THANK YOU

