New Features in C# 13

日 程
Agena

- Params Collections
- OverloadResolutionPriority
- Partial Properties
- Ref Struct Interfaces
- Lock Object

Params Collections

C# 12
Collection Expression

```
IEnumerable<string> enumerable = ["Hello", "dotnet"];
IReadOnlyCollection<string> readOnlyCollection = ["He
ICollection<string> collection = ["Hello", "dotnet"];
```


```
int[] numArray = [1, 2, 3];
HashSet<int> numSet = [1, 2, 3];
List<int> numList = [1, 2, 3];
Span<char> charSpan = ['H', 'e', 'l', 'l', 'o'];
ReadOnlySpan<string> stringSpan = ["Hello", "World"];
ImmutableArray<string> immutableArray = ["Hello",
"World"];

int[] nums = [1, 1, ..numArray, 2, 2];
Console.WriteLine(string.Join(",", nums));
Console.ReadLine();

int[] row0 = [1, 2, 3];
int[] row1 = [4, 5, 6];
int[] row2 = [7, 8, 9];
int[] single = [..row0, ..row1, ..row2];
foreach (var element in single)
{
    Console.Write($"{element}, ");
}
```

Params Collections

```
void ParamsArrayMethod(params int[] array)
{
    foreach (var item in array)
    {
        Console.WriteLine(item);
    }
}

void ParamsReadOnlySpanMethod(params ReadOnlySpan<int> collection)
{
    foreach (var item in collection)
    {
        Console.WriteLine(item);
    }
}

void ParamsSpanMethod(params Span<int> collection)
{
    foreach (var item in collection)
    {
        Console.WriteLine(item);
    }
}

void ParamsListMethod(params List<int> list)
{
    foreach (var item in list)
    {
        Console.WriteLine(item);
    }
}

void ParamsEnumerableMethod(params IEnumerable<int> array)
{
    foreach (var item in array)
    {
        Console.WriteLine(item);
    }
}
```

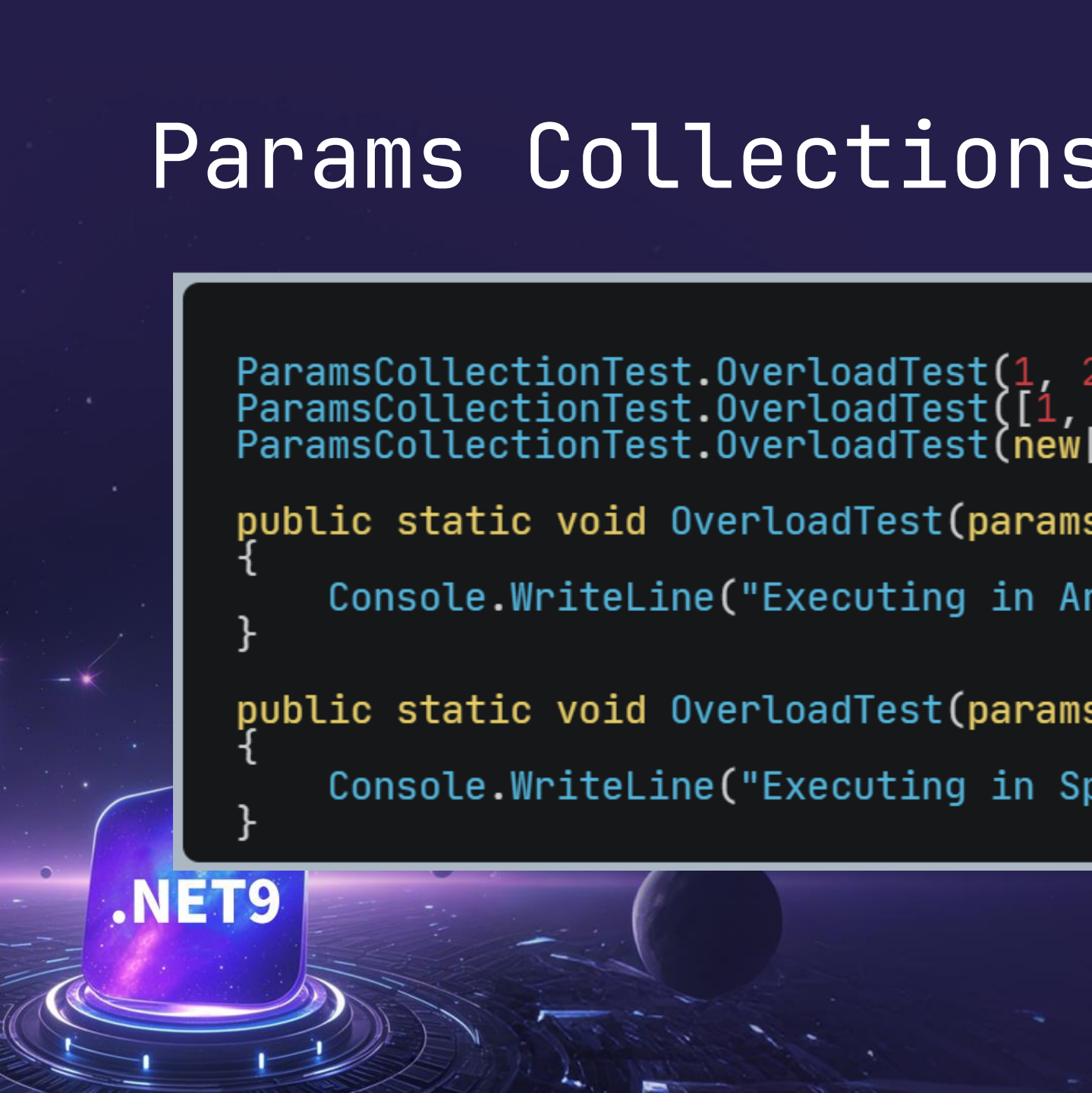
Params Collections

```
[CollectionBuilder(typeof(CustomCollectionBuilder),
nameof(CustomCollectionBuilder.CreateNumber))]
file sealed class CustomNumberCollection : IEnumerable<int>
{
    public required int[] Numbers { get; init; }
    public IEnumerator<int> GetEnumerator()
    {
        return Numbers.AsEnumerable().GetEnumerator();
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return (IEnumerator)Numbers.GetEnumerator();
    }
}

file static class CustomCollectionBuilder
{
    public static CustomNumberCollection CreateNumber(ReadOnlySpan<int>
elements)
    {
        return new CustomNumberCollection()
        {
            Numbers = elements.ToArray()
        };
    }
}
```

```
void ParamsCustomCollectionMethod
    (params CustomNumberCollection collection)
{
    foreach (var item in collection)
    {

        Console.WriteLine(item);
    }
}
```

Params Collections

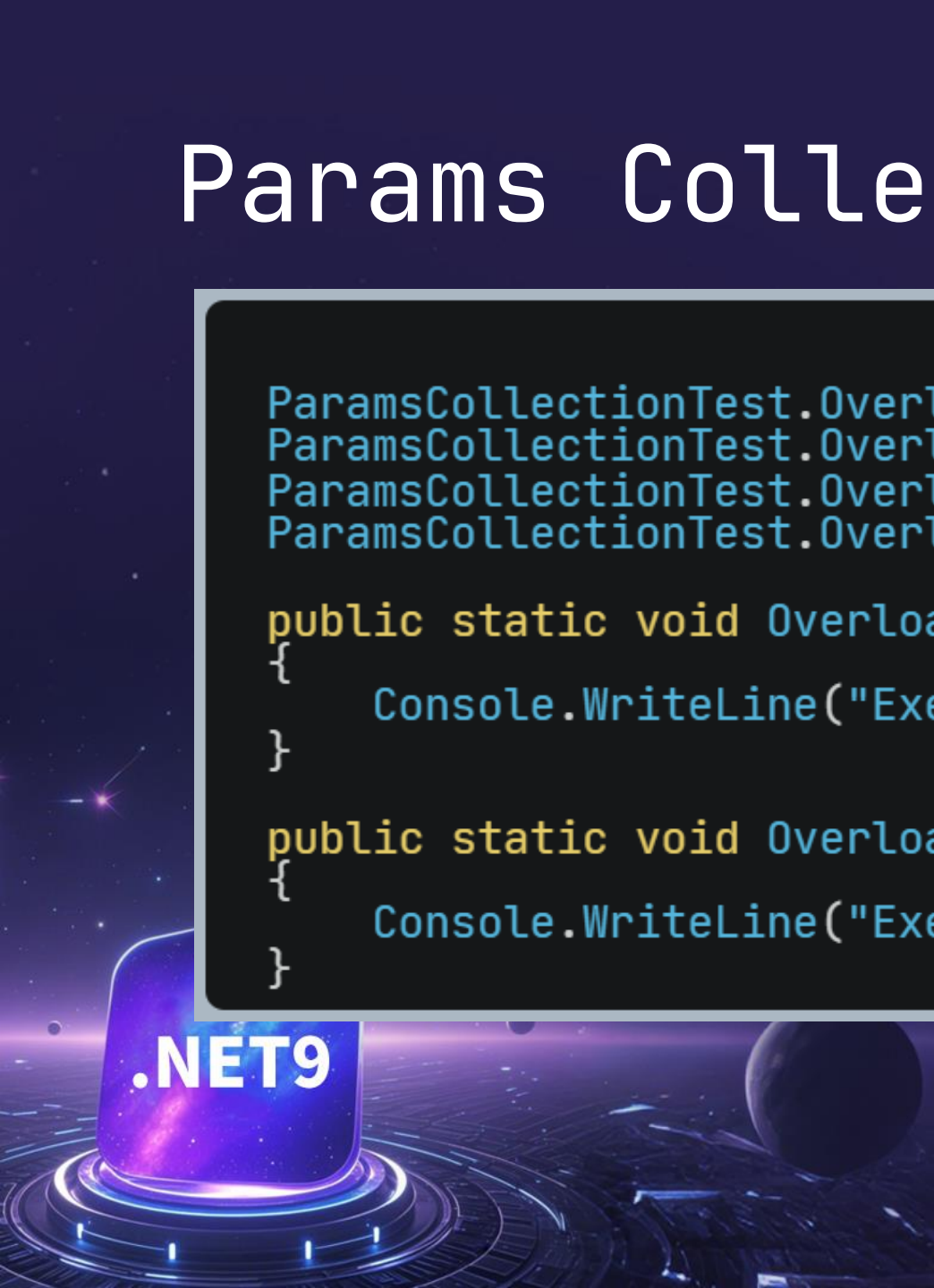
```
ParamsCollectionTest.OverloadTest(1, 2, 3);
ParamsCollectionTest.OverloadTest([1, 2, 3]);
ParamsCollectionTest.OverloadTest(new[] { 1, 2, 3 });

public static void OverloadTest(params int[] array)
{
    Console.WriteLine("Executing in Array method");
}

public static void OverloadTest(params ReadOnlySpan<int> span)
{
    Console.WriteLine("Executing in Span method");
}
```



Span 优先
Span First

Params Collections

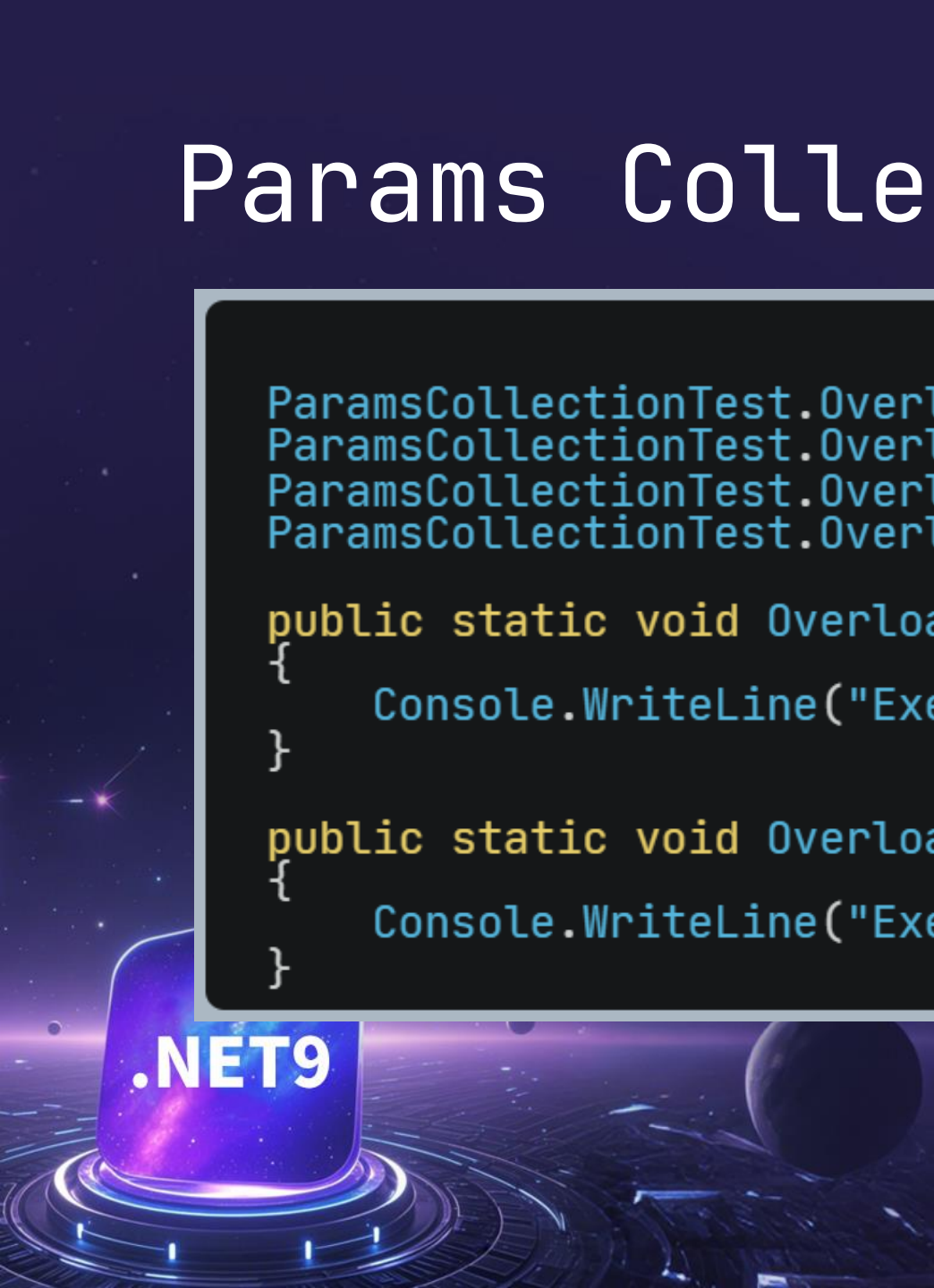
```
ParamsCollectionTest.OverloadTest2(1, 2, 3);
ParamsCollectionTest.OverloadTest2([1, 2, 3]);
ParamsCollectionTest.OverloadTest2(new[] { 1, 2, 3 });
ParamsCollectionTest.OverloadTest2(Enumerable.Range(1, 3));

public static void OverloadTest2(params int[] array)
{
    Console.WriteLine("Executing in Array method");
}

public static void OverloadTest3(params IEnumerable<int> values)
{
    Console.WriteLine("Executing in IEnumerable method");
}
```



具体类型优先
Concrete Type First

Params Collections

```
ParamsCollectionTest.OverloadTest2(1, 2, 3);
ParamsCollectionTest.OverloadTest2([1, 2, 3]);
ParamsCollectionTest.OverloadTest2(new[] { 1, 2, 3 });
ParamsCollectionTest.OverloadTest2(Enumerable.Range(1, 3));

public static void OverloadTest2(params int[] array)
{
    Console.WriteLine("Executing in Array method");
}

public static void OverloadTest3(params IEnumerable<int> values)
{
    Console.WriteLine("Executing in IEnumerable method");
}
```



具体类型优先
Concrete Type First

Params Collections

```
ParamsCollectionTest.OverloadTest3(1, 2, 3);
ParamsCollectionTest.OverloadTest3([1, 2, 3]);
ParamsCollectionTest.OverloadTest3(new[] { 1, 2, 3 });
ParamsCollectionTest.OverloadTest3(new List<int> { 1, 2, 3 });
ParamsCollectionTest.OverloadTest3(Enumerable.Range(1, 3));

public static void OverloadTest3(params IEnumerable<int> values)
{
    Console.WriteLine("Executing in IEnumerable method");
}

public static void OverloadTest3(params int[] array)
{
    Console.WriteLine("Executing in Array method");
}

public static void OverloadTest3(params List<int> values)
{
    Console.WriteLine("Executing in List method");
}
```

Params Collections

```
ParamsCollectionTest.OverloadTest3(1, 2, 3);
ParamsCollectionTest.OverloadTest3([1, 2, 3]);
ParamsCollectionTest.OverloadTest3(new[] { 1, 2, 3 });
ParamsCollectionTest.OverloadTest3(new List<int> { 1, 2, 3 });
ParamsCollectionTest.OverloadTest3(Enumerable.Range(1, 3));
```

```
ParamsCollectionTest.OverloadTest3(1, 2, 3);
ParamsCollectionTest.OverloadTest3([1, 2, 3]);
ParamsCollectionTest.OverloadT
ParamsCollectionTest.OverloadT
ParamsCollectionTest.OverloadT
```

```
void ParamsCollectionTest.OverloadTest3(params IEnumerable<int> values) (+ 2 overloads)

CS0121: The call is ambiguous between the following methods or properties: 'ParamsCollectionTest.OverloadTest3(params int[])' and 'ParamsCollectionTest.OverloadTest3(params List<int>)'
```


```
public static void OverloadTest3(params int[] array)
{
    Console.WriteLine("Executing in Array method");
}

public static void OverloadTest3(params List<int> values)
{
    Console.WriteLine("Executing in List method");
}
```

Params Collections

```
ParamsCollectionTest.OverloadTest3(1, 2, 3);
ParamsCollectionTest.OverloadTest3([1, 2, 3]);
ParamsCollectionTest.OverloadTest3(new[] { 1, 2, 3 });
ParamsCollectionTest.OverloadTest3(new List<int> { 1, 2, 3 });
ParamsCollectionTest.OverloadTest3(Enumerable.Range(1, 3));

public static void OverloadTest3(params IEnumerable<int> values)
{
    Console.WriteLine("Executing in IEnumerable method");
}

public static void OverloadTest3(params int[] array)
{
    Console.WriteLine("Executing in Array method");
}

public static void OverloadTest3(params List<int> values)
{
    Console.WriteLine("Executing in List method");
}
```

Params Collections

```
ParamsCollectionTest.OverloadTest4(1, 2, 3);
ParamsCollectionTest.OverloadTest4([1, 2, 3]);
ParamsCollectionTest.OverloadTest4(Enumerable.Range(1, 3));

public static void OverloadTest4(params IEnumerable<int> values)
{
    Console.WriteLine("Executing in IEnumerable method");
}
```

```
ParamsCollectionTest.OverloadTest4(1, 2, 3);
ParamsCollectionTest.OverloadTest4([1, 2, 3]);
ParamsCollectionTest.Ove
```

void ParamsCollectionTest.OverloadTest4(params IList<int> values) (+ 2 overloads)

```
public static void OverloadTest4(params IList<int> values)
{
    Console.WriteLine("Executing in IList method");
}
```

Params Collections

```
public class ParamsCollectionBenchmark
{
    [Benchmark]
    public int ParamsSpanMethod()
    {
        return ParamsOverloadSpanMethod(1, 2, 3);
    }

    [Benchmark]
    public int ParamsArrayMethod()
    {
        return ParamsOverloadArrayMethod(1, 2, 3);
    }

    private int ParamsOverloadSpanMethod(params ReadOnlySpan<int> span)
        => span.Length;

    private int ParamsOverloadArrayMethod(params int[] array)
        => array.Length;
}
```

Params Collections

```
public class ParamsCollectionBenchmark
{
    [Benchmark]
    public int ParamsSpanMethod()
    {
        return Para
    }

    [Benchmark]
    public int Para
    {
        return Para
    }

    private int Par
        => span.Le

    private int Par
        => array.L
}
```

```
BenchmarkDotNet v0.14.0, Windows 11 (10.0.22631.4169/23H2/2023Update/SunValley3)
Intel Core Ultra 7 155H, 1 CPU, 22 logical and 16 physical cores
.NET SDK 9.0.100-rc.1.24452.12
  [Host]     : .NET 9.0.0 (9.0.24.43107), X64 RyuJIT AVX2
  DefaultJob : .NET 9.0.0 (9.0.24.43107), X64 RyuJIT AVX2


| Method           | Mean      | Error     | StdDev    | Median    | Ratio | Ra
 |
|----------------- |----------:|----------:|----------:|----------:|------:|---
:|
| ParamsSpanMethod | 0.0012 ns | 0.0032 ns | 0.0027 ns | 0.0000 ns |     ? |
 |
| ParamsArrayMethod | 4.0701 ns | 0.1103 ns | 0.1394 ns | 4.0730 ns |     ? |
 |
```

Params Collections

```
[Benchmark(Baseline = true)]
public int ParamsSpanMethod()
{
    var num = 0;
    for (var i = 0; i < 10_000; i++)
    {
        num += ParamsOverloadSpanMethod(1, 2, 3);
    }
    return num;
}
```

```
BenchmarkDotNet v0.14.0, Windows 11 (10.0.22631.4169/23H2/2023Update/SunValley3)
Intel Core Ultra 7 155H, 1 CPU, 22 logical and 16 physical cores
.NET SDK 9.0.100-rc.1.24452.12
  [Host]     : .NET 9.0.0 (9.0.24.43107), X64 RyuJIT AVX2
  DefaultJob : .NET 9.0.0 (9.0.24.43107), X64 RyuJIT AVX2


| Method             | Mean      | Error     | StdDev    | Ratio | RatioSD | Gen0    | Allocated | Alloc Ratio |
|------------------- |----------:|----------:|----------:|------:|--------:|--------:|----------:|------------:|
| ParamsSpanMethod   |  2.756 us | 0.0548 us | 0.1225 us |  1.00 |    0.07 |       - |         - |          NA |
| ParamsArrayMethod | 43.228 us | 0.8613 us | 0.7192 us | 15.72 |    0.87 | 31.8604 |  400000 B |          NA |
```

Overload Resolution Priority

```
namespace System.Runtime.CompilerServices
{
    /// <summary>
    /// Specifies the priority of a member in overload resolution. When unspecified, the default priority is 0.
    /// </summary>
    [AttributeUsage(AttributeTargets.Method | AttributeTargets.Constructor | AttributeTargets.Property,
AllowMultiple = false, Inherited = false)]
    public sealed class OverloadResolutionPriorityAttribute : Attribute
    {
        /// <summary>
        /// Initializes a new instance of the <see cref="OverloadResolutionPriorityAttribute"/> class.
        /// </summary>
        /// <param name="priority">The priority of the attributed member. Higher numbers are prioritized, lower
numbers are deprioritized. 0 is the default if no attribute is present.</param>
        public OverloadResolutionPriorityAttribute(int priority)
        {
            Priority = priority;
        }

        /// <summary>
        /// The priority of the member.
        /// </summary>
        public int Priority { get; }
    }
}
```

Overload Resolution Priority

```
PrintNumbers(1, 2, 3);
PrintNumbers([1, 2, 3]);

private static void PrintNumbers(params int[] numbers)
{
    Console.WriteLine("PrintNumbers in Array overload");
    foreach (var item in numbers)
    {
        Console.WriteLine(item);
    }
}

private static void PrintNumbers(params ReadOnlySpan<int>
numbers)
{
    Console.WriteLine("PrintNumbers in ReadOnlySpan overload");
    foreach (var item in numbers)
    {
        Console.WriteLine(item);
    }
}
```

```
PrintNumbers1(1, 2, 3);
PrintNumbers1([1, 2, 3]);


[OverloadResolutionPriority(1)]
private static void PrintNumbers1(params int[] numbers)
{
    Console.WriteLine("PrintNumbers1 in Array overload");
    foreach (var item in numbers)
    {
        Console.WriteLine(item);
    }
}

private static void PrintNumbers1(params ReadOnlySpan<int>
numbers)
{
    Console.WriteLine("PrintNumbers1 in ReadOnlySpan overload");
    foreach (var item in numbers)
    {
        Console.WriteLine(item);
    }
}
```

Overload Resolution Priority

```
[Conditional("DEBUG")]
[OverloadResolutionPriority(-1)] // lower priority than (bool, string) overload so that the compiler prefers using CallerArgumentExpression
public static void Assert([DoesNotReturnIf(false)] bool condition) =>
    Assert(condition, string.Empty, string.Empty);

[Conditional("DEBUG")]
public static void Assert([DoesNotReturnIf(false)] bool condition, [CallerArgumentExpression(nameof(condition))] string? message = null) =>
    Assert(condition, message, string.Empty);
```

```
---- DEBUG ASSERTION FAILED ----
---- Assert Short Message ----
numbers.Length != numbersSpan.Length
---- Assert Long Message ----

   at CSharp13Samples.OverloadResolutionPrioritySample.MainTest() in C:\projects\source\SamplesInPractice\net9sample\CSharp13Samples\OverloadResolutionPrioritySample.cs:line 24
   at Program.<Main>$(String[] args) in C:\projects\source\SamplesInPractice\net9sample\CSharp13Samples\Program.cs:line 4
```

https://github.com/dotnet/runtime/pull/104942

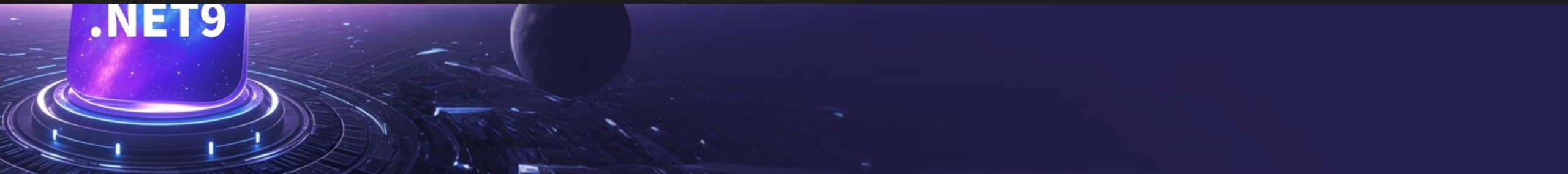
Partial Properties

```
file partial class PartialPropertyClass
{
    /// <summary>
    /// Number comment on declaration
    /// </summary>
    public partial int Num { get; set; }

    private int _num = 1;

    /// <summary>
    /// Number comment on implementation
    /// </summary>
    public partial int Num
    {
        get => _num;
        set => _num = value;
    }
}
```

```
file partial struct PartialPropertyStruct
{
    [DisplayName("Number")]
    public partial int Num
    {
        get;
    }
}
file partial struct PartialPropertyStruct
{
    [JsonPropertyName("num")]
    public partial int Num => 2;
}
```

Partial Properties

```
internal struct
  <PartialPropertySample>FB5A6CA4358E61135AEB1ED6E263D9A5FE244A78163BB9065CE18CE587E1F2AF9__PartialPropertyStru
  ct
{
    // Token: 0x17000006 RID: 6
    // (get) Token: 0x06000048 RID: 72 RVA: 0x00002C46 File Offset: 0x00000E46
    [DisplayName("Number")]
    [JsonPropertyName("num")]
    public int Num
    {
        get
        {
            return 2;
        }
    }
}
```



Partial Properties

```
internal class
  <PartialPropertySample>FB5A6CA4358E61135AEB1ED6E263D9A5FE244A78163BB9065CE18CE587E1F2AF9__PartialPropertyClas
  s
{
    // Token: 0x17000005 RID: 5
    // (get) Token: 0x06000045 RID: 69 RVA: 0x00002C25 File Offset: 0x00000E25
    // (set) Token: 0x06000046 RID: 70 RVA: 0x00002C2D File Offset: 0x00000E2D
    public int Num
    {
        get
        {
            return this._num;
        }
        set
        {
            this._num = value;
        }
    }

    // Token: 0x04000007 RID: 7
    private int _num = 1;
}
```

```
<member name="P:CSharp13Samples.PartialPropertyClass.Num">
    <summary>
    Number comment on implementation
    </summary>
</member>
```

Partial Properties

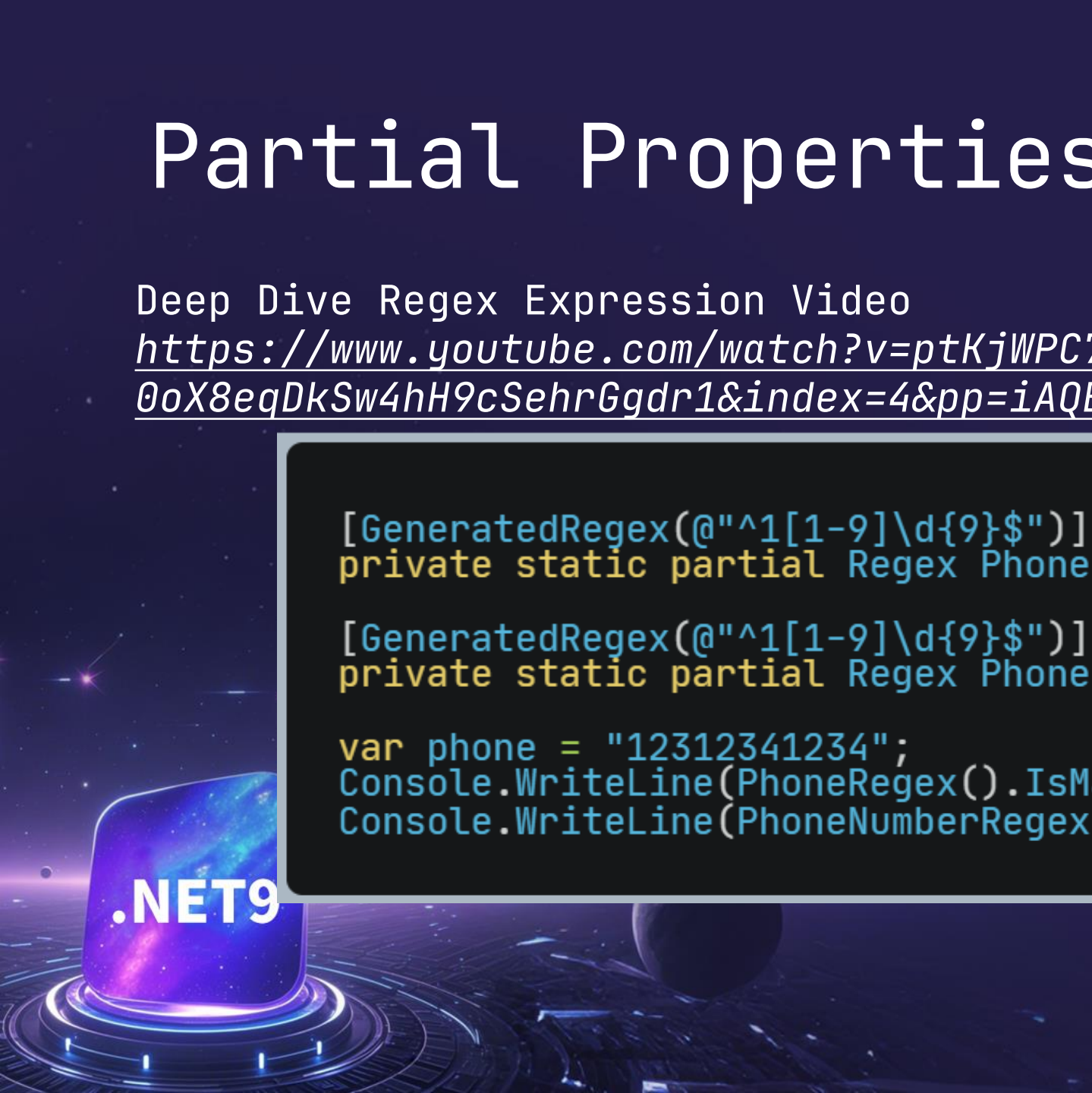
```
file partial class PartialIndexer
{
    public partial string this[int index] { get; }
}

file partial class PartialIndexer
{
    public partial string this[int index]
    {
        get => index.ToString();
    }
}
```

```
internal class
  <PartialPropertySample>FB5A6CA4358E61135AEB1ED6E263D9A5FE244A78163BB9065CE18CE587E1F2AF9__PartialIndexer
{
    // Token: 0x17000007 RID: 7
    public string this[int index]
    {
        get
        {
            return index.ToString();
        }
    }
}
```

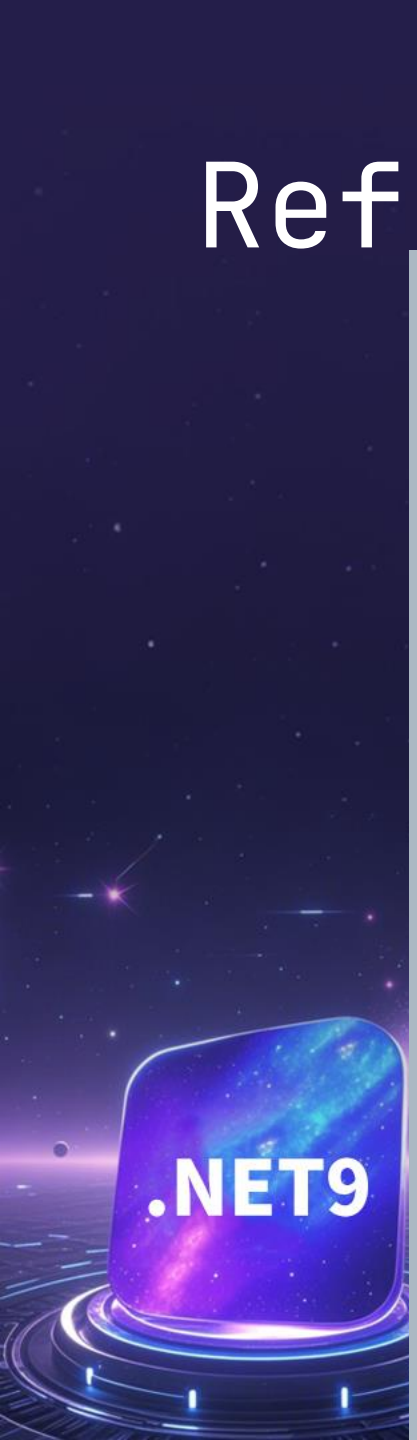
Partial Properties

Deep Dive Regex Expression Video
https://www.youtube.com/watch?v=ptKjWPC7pqw&list=PLdo4fOcmZ0oX8eqDkSw4hH9cSehrGgdr1&index=4&pp=iAQB

```
[GeneratedRegex(@"^1[1-9]\d{9}$")]
private static partial Regex PhoneRegex();

[GeneratedRegex(@"^1[1-9]\d{9}$")]
private static partial Regex PhoneNumberRegex { get; }

var phone = "12312341234";
Console.WriteLine(PhoneRegex().IsMatch(phone));
Console.WriteLine(PhoneNumberRegex.IsMatch(phone));
```

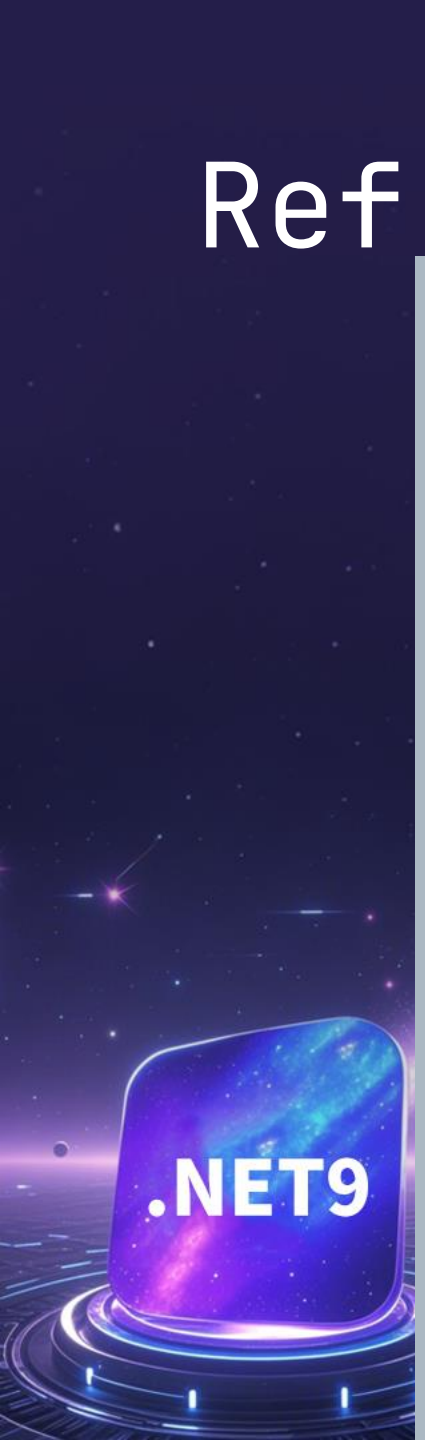


Ref Struct Interfaces




```
internal interface IAge
{
    int AgeNum { get; }
    int GetAge();
}

internal readonly ref struct RefStructAge(int age) : IAge
{
    public int AgeNum => age;
    public int GetAge() => AgeNum;
}

internal readonly struct StructAge(int age) : IAge
{
    public int AgeNum => age;
    public int GetAge() => AgeNum;
}

internal sealed class ClassAge(int age) : IAge
{
    public int AgeNum => age;
    public int GetAge() => AgeNum;
}
```

Ref Struct Interfaces

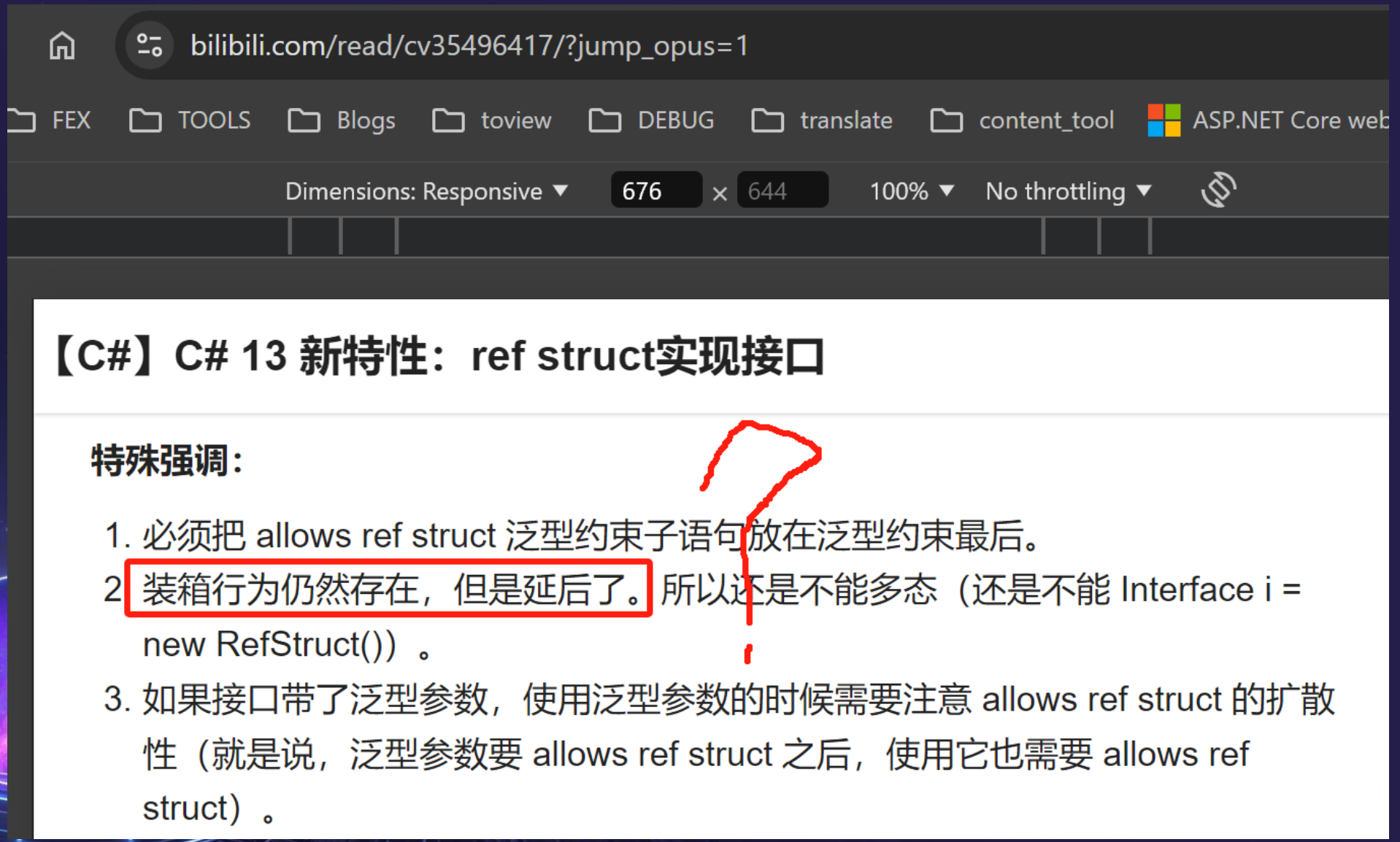



```
var classAge = new ClassAge(1);
var structAge = new StructAge(1);
var refStructAge = new RefStructAge(1);

PrintAge0(classAge);
PrintAge0(structAge);
// CS1503: Argument 1: cannot convert from
'CSharp13Samples.RefStructAge' to 'CSharp13Samples.IAge'
// PrintAge0(refStructAge);

PrintAge(classAge);
PrintAge(structAge);
PrintAge(refStructAge);

private static void PrintAge0(IAge age)
{
    Console.WriteLine(age.GetAge());
}

private static void PrintAge<TAge>(TAge age)
    where TAge : IAge, allows ref struct
{
    Console.WriteLine($"GetAge {age.GetAge()}");
    Console.WriteLine($"AgeNum {age.AgeNum}");
}
```

R

```
//     RefStructInterfaceSample.PrintAge0(structAge);

/* (13,9)-(13,30) C:\projects\source\SamplesInPractice\net9sample\CSharp13Samples
  \RefStructInterfaceSample.cs */
/* 0x00000E87 07          */ IL_001F: ldloc.1
/* 0x00000E88 8C1A000002  */ IL_0020: box        CSharp13Samples.StructAge
/* 0x00000E8D 284E000006  */ IL_0025: call       void CSharp13Samples.RefStructInterfaceSample::PrintAge0
  (class CSharp13Samples.IAge)
/* 0x00000E92 00          */ IL_002A: nop

//     RefStructInterfaceSample.PrintAge<ClassAge>(classAge);

/* (17,9)-(17,28) C:\projects\source\SamplesInPractice\net9sample\CSharp13Samples
  \RefStructInterfaceSample.cs */
/* 0x00000E93 06          */ IL_002B: ldloc.0
/* 0x00000E94 280900002B  */ IL_002C: call       void
  CSharp13Samples.RefStructInterfaceSample::PrintAge<class CSharp13Samples.ClassAge>(!!0)
/* 0x00000E99 00          */ IL_0031: nop

//     RefStructInterfaceSample.PrintAge<StructAge>(structAge);

/* (18,9)-(18,29) C:\projects\source\SamplesInPractice\net9sample\CSharp13Samples
  \RefStructInterfaceSample.cs */
/* 0x00000E9A 07          */ IL_0032: ldloc.1
/* 0x00000E9B 280A00002B  */ IL_0033: call       void
  CSharp13Samples.RefStructInterfaceSample::PrintAge<valuetype CSharp13Samples.StructAge>(!!0)
/* 0x00000EA0 00          */ IL_0038: nop

//     RefStructInterfaceSample.PrintAge<RefStructAge>(refStructAge);

/* (19,9)-(19,32) C:\projects\source\SamplesInPractice\net9sample\CSharp13Samples
  \RefStructInterfaceSample.cs */
/* 0x00000EA1 08          */ IL_0039: ldloc.2
/* 0x00000EA2 280B00002B  */ IL_003A: call       void
  CSharp13Samples.RefStructInterfaceSample::PrintAge<valuetype CSharp13Samples.RefStructAge>(!!0)
/* 0x00000EA7 00          */ IL_003F: nop

//     int[] numbers = new int[] { 1, 2, 3, 4 };
```

Ref Struct Interfaces



Ref


```
[SimpleJob]
[MemoryDiagnoser]
public class RefStructInterfaceBenchmark
{
    [Benchmark(Baseline = true)]
    public int RefStructInterface()
    {
        var num = 0;
        for (var i = 0; i < 10_000; i++)
        {
            Aggregate(ref num, new RefStructAge(1));
        }
        return num;
    }

    [Benchmark]
    public int ClassInterface()
    {
        var num = 0;
        for (var i = 0; i < 10_000; i++)
        {
            Aggregate(ref num, new ClassAge(1));
        }
        return num;
    }

    private static void Aggregate<T>(ref int init, T t)
        where T : IAge, allows ref struct
    {
        init += t.GetAge();
    }
}
```

Ref Struct Interfaces

```
[Benchmark(Baseline = true)]
public int RefStructInterface()
{
```

```
BenchmarkDotNet v0.14.0, Windows 11 (10.0.22631.4169/23H2/2023Update/SunValley3)
Intel Core Ultra 7 155H, 1 CPU, 22 logical and 16 physical cores
.NET SDK 9.0.100-rc.1.24452.12
  [Host]     : .NET 9.0.0 (9.0.24.43107), X64 RyuJIT AVX2
  DefaultJob : .NET 9.0.0 (9.0.24.43107), X64 RyuJIT AVX2
```

Method	Mean	Error	StdDev	Ratio	RatioSD	Gen0	Allocated	Alloc Ratio
RefStructInterface	5.234 us	0.2491 us	0.7227 us	1.04	0.35	-	-	NA
ClassInterface	26.331 us	1.4789 us	4.2905 us	5.25	1.81	19.1040	240000 B	NA


```
            Aggregate(ref num, new ClassAge(1));
        }
        return num;
    }

    private static void Aggregate<T>(ref int init, T t)
        where T : IAge, allows ref struct
    {
        init += t.GetAge()
    }
```

Ref Struct Interfaces

```
    public delegate TResult Func<in T1, in T2, in T3, in T4, in T5, in T6, in T7, in T8, in T9, in T10, in T11, in T12, i
+       where T1 : allows ref struct
+       where T2 : allows ref struct
+       where T3 : allows ref struct
+       where T4 : allows ref struct
+       where T5 : allows ref struct
+       where T6 : allows ref struct
+       where T7 : allows ref struct
+       where T8 : allows ref struct
+       where T9 : allows ref struct
+       where T10 : allows ref struct
+       where T11 : allows ref struct
+       where T12 : allows ref struct
+       where T13 : allows ref struct
+       where T14 : allows ref struct
+       where T15 : allows ref struct
+       where T16 : allows ref struct
+       where TResult : allows ref struct;

    public delegate bool Predicate<in T>(T obj)
+       where T : allows ref struct;

    public sealed class String
    {
        public static string Create<TState>(int length, TState state, SpanAction<char, TState> action)
+           where TState : allows ref struct
    }
}

namespace System.Buffers
{
    public delegate void SpanAction<T, in TArg>(Span<T> span, TArg arg)
+       where TArg : allows ref struct;

    public delegate void ReadOnlySpanAction<T, in TArg>(ReadOnlySpan<T> span, TArg arg)
+       where TArg : allows ref struct;
}
```

Lock Object

sharplab.io/#v2:EYLgtghglgdgNAExAagD4AEBMBGAsAKHQAYACdbAVgG4CD0BmMzEgYRIG8CTuzH0AWEgFkAFAEoOXHtIBuEAE4koJALwkiNfNNkKSAGw...

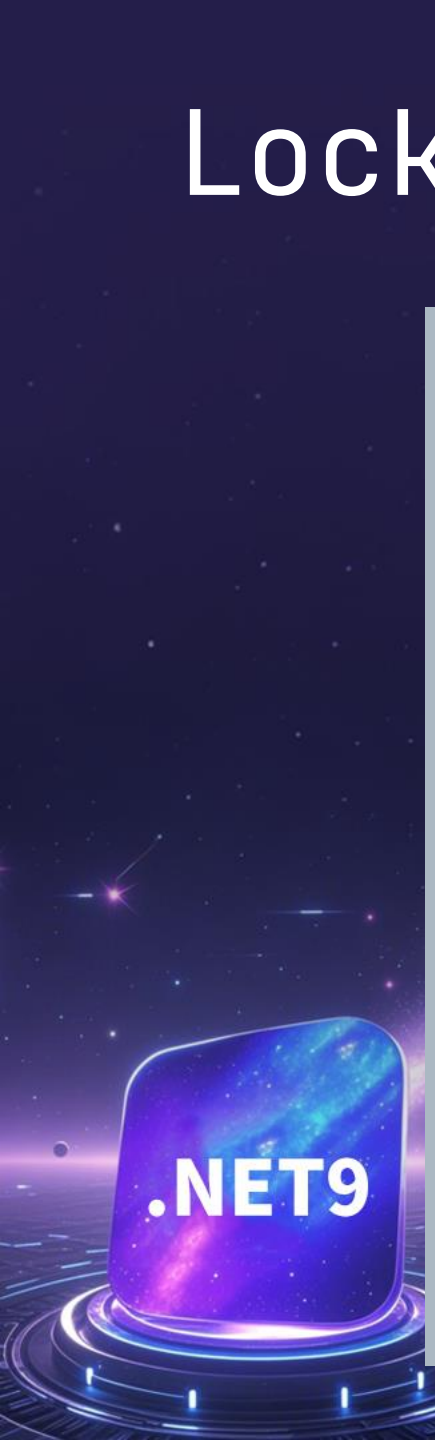
DOTNET | FEX | TOOLS | Blogs | toview | DEBUG | translate | content_tool | ASP.NET Core web...

Code C# Create Gist main (11 Jun 2024) Results C#

```
using System;
using System.Threading.Tasks;

public class C {
    public void M() {
        var i = 0;
        var locker = new object();
        Parallel.For(1, 100, _ =>
        {
            lock (locker)
            {
                i++;
            }
        });
        Console.WriteLine(i);
    }
}
```

```
[CompilerGenerated]
private sealed class <>c__DisplayClass0_0
{
    public object locker;

    public int i;

    internal void <M>b__0(int _)
    {
        object obj = locker;
        bool lockTaken = false;
        try
        {
            Monitor.Enter(obj, ref lockTaken);
            i++;
        }
        finally
        {
            if (lockTaken)
            {
                Monitor.Exit(obj);
            }
        }
    }
```

Lock Object




```
namespace System.Threading;

public sealed class Lock
{
    public Scope EnterScope();

    public void Enter();
    public void Exit();
    public bool TryEnter();
    public bool TryEnter(TimeSpan timeout);
    public bool TryEnter(int millisecondsTimeout);

    public bool IsHeldByCurrentThread { get; }

    public struct Scope : IDisposable
    {
        public void Dispose();
    }
}
```

Lock Object

```
var i = 0;
var locker = new Lock();
Parallel.For(1, 100, _ ⇒
{
    lock (locker)
    {
        i++;
    }
});
Console.WriteLine(i);
```

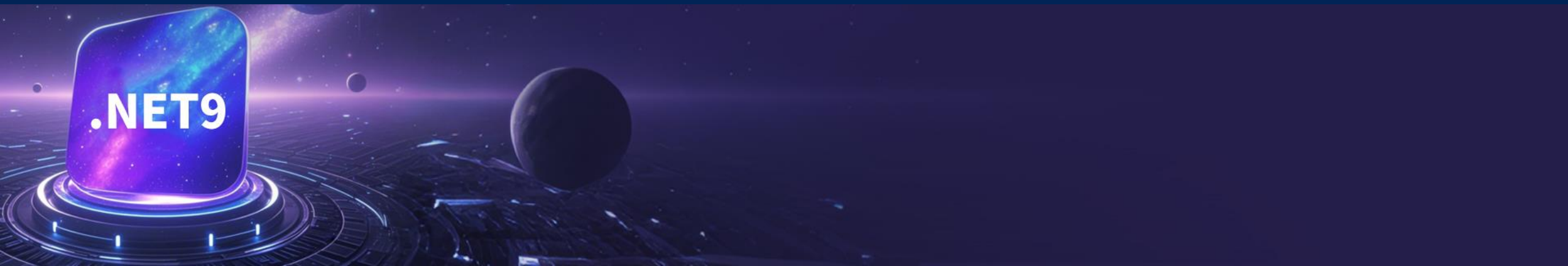
```
int i = 0;
Lock locker = new Lock();
Parallel.For(1, 100, delegate(int _)
{
    using (locker.EnterScope())
    {
        int j = i;
        i = j + 1;
    }
});
Console.WriteLine(i);
```

Lock


```
[SimpleJob]
[MemoryDiagnoser]
public class LockObjectBenchmark
{
    private readonly object _lock0 = new();
    private readonly Lock _lock1 = new();

    [Benchmark(Baseline = true)]
    public int NewLockObject()
    {
        var i = 0;
        Parallel.For(1, 1000, _ ⇒
        {
            lock (_lock1)
            {
                Interlocked.Increment(ref i);
            }
        });
        return i;
    }

    [Benchmark]
    public int TraditionalLock()
    {
        var i = 0;
        Parallel.For(1, 1000, _ ⇒
        {
            lock (_lock0)
            {
                Interlocked.Increment(ref i);
            }
        });
        return i;
    }
}
```

Lock Object

```
BenchmarkDotNet v0.14.0, Windows 11 (10.0.22631.4169/23H2/2023Update/SunValley3)
Intel Core Ultra 7 155H, 1 CPU, 22 logical and 16 physical cores
.NET SDK 9.0.100-rc.1.24452.12
  [Host]     : .NET 9.0.0 (9.0.24.43107), X64 RyuJIT AVX2
  DefaultJob : .NET 9.0.0 (9.0.24.43107), X64 RyuJIT AVX2


| Method          | Mean     | Error   | StdDev   | Ratio | RatioSD | Gen0   | Allocated | Alloc Ratio |
|---------------- |---------:|--------:|---------:|------:|--------:|-------:|----------:|------------:|
| NewLockObject   | 280.0 us | 5.57 us | 11.99 us |  1.00 |    0.06 |      - |   5.71 KB |        1.00 |
| TraditionalLock | 303.3 us | 6.01 us | 13.45 us |  1.09 |    0.07 | 0.4883 |   5.99 KB |        1.05 |
```



Lock Object

```
private static readonly Lock Locker = new();

public static void LogInit(LoggerConfiguration loggerConfiguration)
{
    lock (Locker)
    {
        Log.Logger = loggerConfiguration.CreateLogger();
    }
}
```

```
<Project>

  <ItemGroup>
      <Using Include="System.Object" Alias="Lock"
Condition="!$([MSBuild]::IsTargetFrameworkCompatible('$(TargetFramework)', 'net9.0'))" />
  </ItemGroup>

</Project>
```

More

New Escape Sequence \e

Collection Expression Better Conversion

Ref Safe Extension

Field Keyword??

References

- https://learn.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-13
- https://devblogs.microsoft.com/dotnet/csharp-13-explore-preview-features/
- https://github.com/dotnet/roslyn/blob/main/docs/Language%20Feature%20Status.md
- https://github.com/dotnet/csharplang/blob/main/proposals/csharp-13.0/
- https://github.com/WeihanLi/SamplesInPractice/tree/main/net9sample/CSharp13Samples
- https://mp.weixin.qq.com/mp/appmsgalbum?__biz=MzAxMjE2NTMxMw==&action=getalbum&album_id=3513771389918691333#wechat_redirect

Thank You