ref-структуры в C# 13

Юрий Малич
NP4 GmbH
yurymalich@yandex.ru

О себе

- Закончил ПГУПС (Автоматика и телемеханика на ж/д транспорте)
- Программирую примерно с 1996 года, с MS DOS на Intel i386
- На C# .NET с первых версий. Microsoft Certified Professional 2016
- Писал статьи по архитектуре микропроцессоров для сайтов iXBT.com, fcenter.ru, 3dnews.ru
- Принимал участие в разработке системных утилит для Windows: в Nero AG (Nero Burning Rom) и в TuneUp (TuneUp Utilities)
- Работаю в NP4 GmbH, разрабатываю Desktop и Backend приложения

ref-структуры

- Структура с модификатором ref – value type (переменные сохраняются по значению)
- Ограничение: значения переменных типа ref-структур можно хранить **только в стеке (или в регистрах)**
- Могут (но не обязаны) содержать поля-ссылки (ref fields)

Ограничения ref-структур

- ref-структуру можно хранить только в стеке
- Не может быть полем или свойством класса

```
class Pax
{
   public Span<char> Name;
}
```

- Нельзя присваивать значение переменной типа Object или интерфейс

```
object? span = "number 123456".AsSpan()
```

- Нельзя (было) реализовывать интерфейсы (до C# 13)
- Нельзя (было) использовать в качестве аргумента дженерик-функций (до C# 13)
- Нельзя (было) использовать в async-функциях (до C# 13)

Возможности ref-структур

- Может содержать ref-поля
- Может быть локальной переменной (но не в async-функциях)
- Может быть полем другой ref-структуры

```
ref struct Pax
{
    public Span<char> Name;
}
```

- Можно передавать как параметр в функцию

```
static Span<char> Method(Span<char> t)
{
    return t;
}
```

- Можно возвращать по значению из функции

ref-структуры

```
public readonly ref struct Span<T>
{
    /// <summary>A byref or a native ptr.</summary>
    internal readonly ref T _reference;

    /// <summary>The number of elements this Span contains.</summary>
    private readonly int _length;

}
```

ref-структуры

```
string str = "number 123456";

var subStr = str.Substring(7);  // subStr == "123456"

int.TryParse(subStr, out var number); // number == 123456
```

ref-структуры

```
string str = "number 123456";

ReadOnlySpan<char> span = str.AsSpan(7);  // span == "123456"

int.TryParse(span, out var number); // number == 123456
```

Ограничения C# 12

```
private T Method<T>(T t)
{
    return t;
}

Method(str.AsSpan(7));

//  error CS0306: The type 'ReadOnlySpan<char>' may not be used as a type argument
```

Ограничения C# 12

```
private ReadOnlySpan<char> Method(ReadOnlySpan<char> t)
{
    return t;
}

Method(str.AsSpan(7)); // OK
```

Ограничения C# 12

```
public void DoSomething(List<string> list, Func<string, bool> predicate)
{
    foreach (var str in list)
    {
        var subStr = str.Substring(2, 2);
        if (predicate(subStr))
        {
            // do!
        }
    }
}

DoSomething(list, x => x[0] == '#' && x[1] == '1');
```

Ограничения C# 12

```
public void DoSomething(List<string> list, Func<ReadOnlySpan<char>, bool> predicate)
{
    foreach (var str in list)
    {
        var span = str.AsSpan(2, 2);
        if (predicate(span))
        {
            // do!
        }
    }
}

// error CS0306: The type 'ReadOnlySpan<char>' may not be used as a type argument
```

Ограничения C# 12

```
delegate bool SpanDelegate(ReadOnlySpan<char> x);

public void DoSomething(List<string> list, SpanDelegate predicate)
{
    foreach (var str in list)
    {
        var span = str.AsSpan(2, 2);
        if (predicate(span))
        {
            // do!
        }
    }
}

DoSomething(list, x => x[0] == '#' && x[1] == '1');
```

C# 13 allows ref struct

```
private static T Method<T>(T t) where T : allows ref struct
{
    // object? obj = t;
    // T[] array;
    return t;
}

Method(str.AsSpan(7));
```

C# 13 allows ref struct

```
namespace System
{
    public delegate TResult Func<out TResult>()
        where TResult : allows ref struct;

    public delegate TResult Func<in T, out TResult>(T arg)
        where T : allows ref struct
        where TResult : allows ref struct;

    public delegate TResult Func<in T1, in T2, out TResult>(T1 arg1, T2 arg2)
        where T1 : allows ref struct
        where T2 : allows ref struct
        where TResult : allows ref struct;



}
```

C# 13 allows ref struct

```
public static void DoSomething(List<string> list, Func<ReadOnlySpan<char>, bool> predicate)
{
    foreach (var str in list)
    {
        ReadOnlySpan<char> span = str.AsSpan(1, 2);
        if (predicate(span))
        {
            // do!
        }
    }
}

DoSomething(list, x => x[0] == '#' && x[1] == '1'); // OK!
```

C# 13 allows ref struct

```
public class ConcurrentDictionary<TKey, TValue>
{
        // ……. //
        public TValue GetOrAdd<TArg>(TKey key,
                                     Func<TKey, TArg, TValue> valueFactory,
                                     TArg factoryArgument)
            where TArg : allows ref struct
        {
            // ……. //
        }
}

ConcurrentDictionary<string, string> cDict = new();
var span = "123".AsSpan(1);
cDict.GetOrAdd("111", (key, arg) => arg.ToString(), span);
```

C# 13 allows ref struct

```
namespace System
{
    public interface IEquatable<T> where T : allows ref struct
    {
        bool Equals(T? other);
    }
}
```

Наследование и реализация интерфейсов

```
public int Compare<T>(T str, T str2) where T : IComparable<T>
{
    return str.CompareTo(str2);
}

string str1 = "111";
string str2 = "112";

Compare(str1, str2);

var span1 = str1.AsSpan();
var span2 = str2.AsSpan();

Compare(span1, span2); // Не компилируется
```

Наследование и реализация интерфейсов C# 13

```
public ref struct ComparableSpanChar :
     IComparable<ComparableSpanChar>, IEquatable<ComparableSpanChar>
{
    private ReadOnlySpan<char> _value;

    public ComparableSpanChar(ReadOnlySpan<char> val)
    {
        _value = val;
    }

    public static implicit operator ComparableSpanChar(ReadOnlySpan<char> val)
                                    => new ComparableSpanChar(val);

    public int CompareTo(ComparableSpanChar other)
    {
        return _value.SequenceCompareTo(other._value);
    }

    public bool Equals(ComparableSpanChar other)
    {
        return _value.SequenceEqual(other._value);
    }
}
```

Наследование и реализация интерфейсов C# 13

```
public int Compare<T>(T str, T str2) where T : IComparable<T>, allows ref struct
{
    // IComparable<T>? x = str;
    return str.CompareTo(str2);
}

Compare((ComparableSpanChar)span1, (ComparableSpanChar)span2); // OK!


IComparable c = (IComparable)new ComparableSpanChar(span1);
// Ошибка Cannot convert type 'ComparableSpanChar' to 'System.IComparable'
```

ref в асинхронных методах C# 12

```
public static async void ReadFileAsync(string path)
{
    var str = await File.ReadAllTextAsync(path);

    ReadOnlySpan<char> span = str.AsSpan(); // error CS4012:
     // Parameters or locals of type 'ReadOnlySan<char>' cannot be declared in async
     // methods or async lambda expressions


     // Reader(str)
}
```

ref в асинхронных методах C# 13

```
public static async void ReadFileAsync(string path)
{
    var str = await File.ReadAllTextAsync(path);

    ReadOnlySpan<char> span = str.AsSpan();

    await Task.CompletedTask;

    span=span.Slice(1); // Ошибка «Instance of type 'System.ReadOnlySpan<char>' cannot
                                    // be preserved across 'await' or 'yield' boundary.»
}
```

ref в методах-итераторах C# 12

```
private IEnumerable<int> TestIterator1(byte[] values)
{
    for (int i = 0; i < values.Length; i += 4)
    {
        ref int x = ref Unsafe.As<byte, int>(ref values[i]); // error CS4012
        yield return x;
    }
}
```

ref в методах-итераторах C# 13

```
private IEnumerable<int> TestIterator1(byte[] values)
{
    for (int i = 0; i < values.Length; i += 4)
    {
        ref int x = ref Unsafe.As<byte, int>(ref values[i]);
        yield return x;
    }
}
```

Заключение

- ref-структуры теперь можно использовать в обобщённых функциях и делегатах
- ref-структуры могут реализовывать интерфейсы
- Некоторые библиотечные функции System.Runtime теперь позволяют принимать переменные типа Span<T> (и других ref-структур)
- Меньше ограничений на использование в асинхронных методах и итераторах

Спасибо за внимание!

What's new in C# 13

https://learn.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-13#allows-ref-struct

C# language specification

16.2.3 Ref modifier

https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/structs#1623-ref-modifier