

# **ValueString**

## **Строки-значения**

**Юрий Малич**  
**NP4 GmbH**  
**[yurymalich@yandex.ru](mailto: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
- Занимаюсь импортом данных из текстовых, xml, json файлов и систем бронирования

# Что хочется улучшить в string

- String.Empty как значение по умолчанию (**default**)
- Обойтись без явных инициализаций свойств типа **string** в классах

```
public string Name { get; set; } = string.Empty;
```

- Уменьшить количество операторов слияния ??

```
x.Name = (name ?? string.Empty).Trim();
```

- Гарантировать отсутствие NullReferenceException при работе со строками

# Проблематика

## Рассмотрим пример

```
#nullable enable
```

```
public struct PassengerNameStruct
{
    public string LastName { get; set; } = string.Empty;

    public string FirstName { get; set; } = string.Empty;

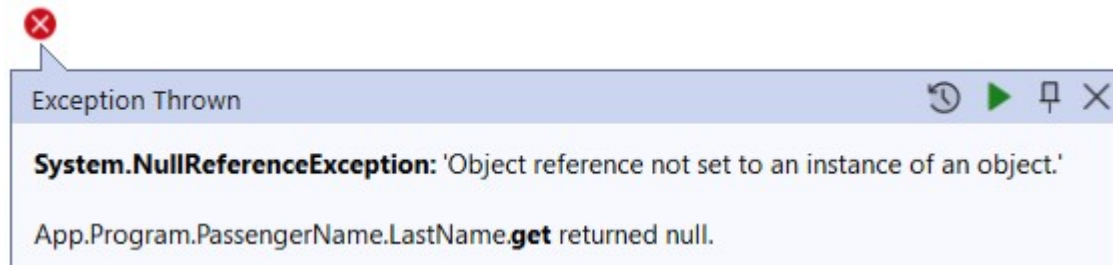
    public PassengerNameStruct(string lastName, string firstName)
    {
        LastName = lastName;
        FirstName = firstName;
    }
}
```

# Проблематика

Что может пойти не так?

```
PassengerNameStruct paxName1 = default;
```

```
Name = paxName1.LastName.ToUpper();
```



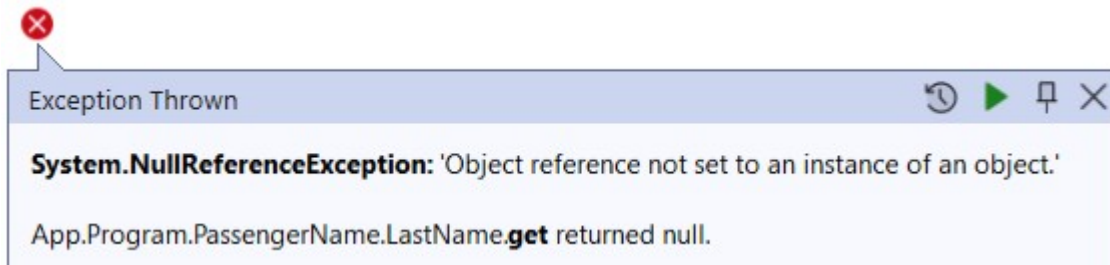
Name	Value
▾ paxName1	{App.PassengerName}
First Name	null
Last Name	null




# Проблематика

Что может пойти не так?

```
var paxName1 = new PassengerNameStruct();
```

```
Name = paxName1.LastName.ToUpper();
```



Name	Value
 paxName1	{App.PassengerName}
 FirstName	null
 LastName	null

# Проблематика

```
#nullable enable
```

```
public struct PassengerNameStruct
{
    public string LastName { get; set; } = string.Empty;

    public string FirstName { get; set; } = string.Empty;




    public PassengerNameStruct()
    {
    }

    public PassengerName(string lastName, string firstName)
    {
        LastName = lastName;
        FirstName = firstName;
    }
}
```

# Проблематика

```
var paxName1 = new PassengerNameStruct();
```

```
Name = paxName1.LastName.ToUpper(); // OK!
```

Name	Value
 paxName1	{App.PassengerName}
 FirstName	""
 LastName	""



# Проблематика

## Parameterless struct constructors

<https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/proposals/csharp-10.0/parameterless-struct-constructors>



### Summary

Support parameterless constructors and instance field initializers for struct types.

### Motivation

Explicit parameterless constructors would give more control over minimally constructed instances of the struct type. Instance field initializers would allow simplified initialization across multiple constructors. Together these would close an obvious gap between struct and class declarations.

Support for field initializers would also allow initialization of fields in record struct declarations without explicitly implementing the primary constructor.

```
C#Copyrecord struct Person(string Name)
{
    public object Id { get; init; } = GetNextId();
}
```

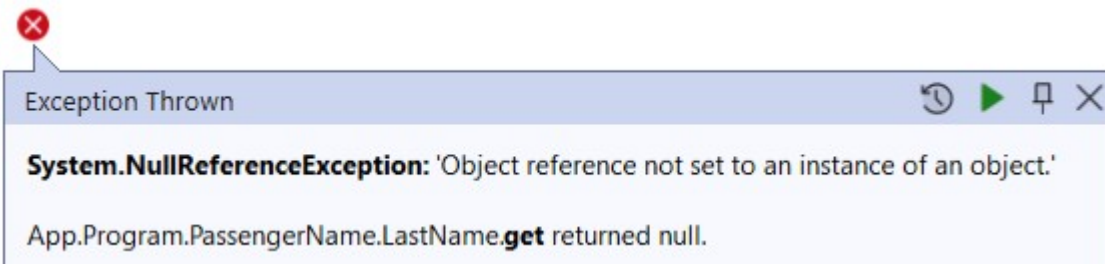
# Проблематика

## Возврат FirstOrDefault(...)

```
var paxNames = new List<PassengerNameStruct>();
```

```
PassengerNameStruct paxName1 = paxNames.FirstOrDefault(x => x.FirstName == "Yury");
```

```
Name = paxName1.LastName.ToUpper();
```



Name	Value
▲ paxName1	{App.PassengerName}
FirstName	null
LastName	null

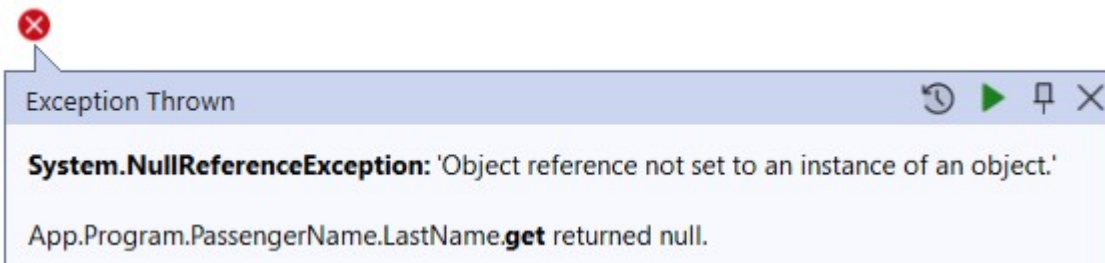
# Проблематика

## Кортежи

```
List<(string FirstName, string LastName)> names = new();
```

```
var (FirstName, LastName) paxName2 = names.FirstOrDefault(x => x.FirstName == "Yury");
```

```
Name = paxName2.LastName.ToUpper();
```



Name	Value
▾ paxName2	(null, null)
▾ FirstName	null
▾ LastName	null
▸ ▾ Raw View	(null, null)

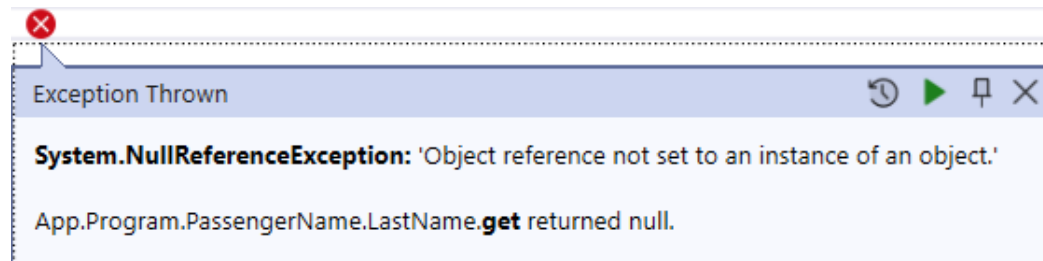
# Проблематика

## Структура в свойстве класса

#nullable enable

```
public class Passenger
{
    public PassengerNameStruct Name { get; set; }
}
```

```
var passenger = new Passenger();
name = passenger.Name.LastName.ToUpper();
```



# Проблематика

## Структура в свойстве класса

```
#nullable enable
```

```
public class Passenger  
{  
    public PassengerNameStruct Name { get; set; } = new();  
}
```

```
var passenger = new Passenger();
```

```
name = passenger.Name.LastName.ToUpper(); // ok!
```

# Проблематика

## Тип из legacy-библиотеки

```
#nullable disable
```

```
public class City  
{  
    public string Name { get; set; }  
  
    public string Code { get; set; }  
}
```

# Проблематика

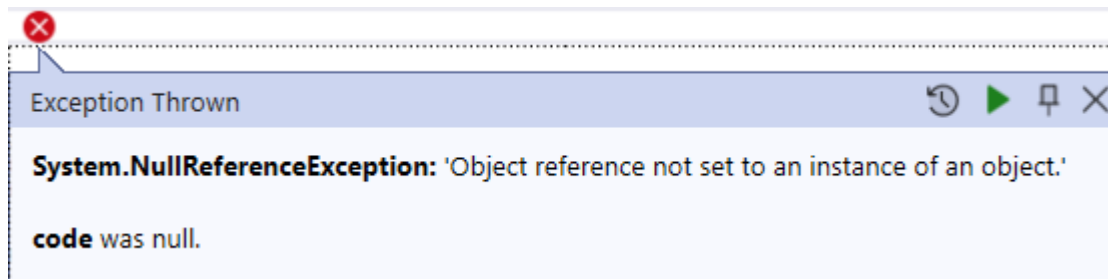
## Тип из legacy-библиотеки

```
#nullable enable
```

```
var city = new City();
```

```
string code = city.Code;
```

```
X.GetCityInfo(code.ToUpper());
```



# Проблематика

Другие случаи

Оператор !

```
List<string> paxName = items  
    .Where(x => x.Item is not null)  
    .Select(x => x.Item!)  
    .ToList();
```



# Проблематика

## Другие случаи

- C# классы, генерируемые SvcUtil.exe и xsd.exe
- C# классы, генерируемые MS Visual Studio из XML и Json
- Рефлексия

# Что ещё хочется улучшить в string

- Улучшить синтаксис (свойства вместо `string.IsNullOrEmpty`)
- Расширить функциональность
- Сохранить обратную совместимость с `System.String`
- Сохранить производительность

# Структура ValueString

```
public struct ValueString :  
    IEquatable<ValueString>, IEquatable<string>, IComparable<string>  
{  
    private string? _value;  
  
    [XmlText]  
    public string Value  
    {  
        get => _value ?? string.Empty;  
        set => _value = value;  
    }  
  
    public char this[int index] => Value[index];  
  
    public int Length => Value.Length;  
  
    [Pure]  
    public override int GetHashCode() => Value.GetHashCode();  
  
    [Pure]  
    public int IndexOf(char value) => Value.IndexOf(value);
```

# Структура ValueString

- Никогда не вызывает `NullReferenceException`
- Значение default - `string.Empty`
- Реализует публичные методы класса `System.String`
- Содержит операторы неявного присваивания в/из `System.String`
- Размер структуры 8 байт (в 64-битном режиме) равен размеру ссылки
- При оптимизации минимальные накладные расходы

# Структура ValueString

## Методы для обратной совместимости со String

- *CompareTo*
- *Contains*
- *CopyTo*
- *EndsWith*
- *Equals*
- *GetHashCode*
- *IndexOf*
- *IndexOfAny*
- *Insert*
- *IsNormalized*
- *LastIndexOf*
- *LastIndexOfAny*
- *Normalize*
- *PadLeft*
- *PadRight*
- *Remove*
- *Replace*
- *Split*
- *StartsWith*
- *Substring*
- *ToCharArray*
- *ToLower*
- *ToLowerInvariant*
- *ToString*
- *ToUpper*
- *ToUpperInvariant*
- *Trim*
- *TrimEnd*
- *TrimStart*

# Структура ValueString

## Методы для обратной совместимости со String

```
public bool Contains(string substring) => Value.Contains(substring);
```

```
public bool Contains(string substring, StringComparison comparisonType)  
    => Value.IndexOf(substring, comparisonType) >= 0;
```

```
public bool EndsWith(string value, bool ignoreCase, CultureInfo culture)  
    => Value.EndsWith(value, ignoreCase, culture);
```

```
public bool Equals(string? value) => Value.Equals(value);
```

```
public override int GetHashCode() => Value.GetHashCode();
```

```
public int IndexOf(char value) => Value.IndexOf(value);
```

```
public int IndexOf(char value, int startIndex) => Value.IndexOf(value, startIndex);
```

# Структура ValueString

## Конструкторы

```
public ValueString()
{
    _value = string.Empty;
}

public ValueString(string? value)
{
    _value = value ?? string.Empty;
}

public ValueString(char[] value)
{
    _value = new string(value);
}

public ValueString(char c, int count)
{
    _value = new string(c, count);
}
```

# Структура ValueString

## Операции неявного преобразования ValueString <-> String

```
public static implicit operator ValueString(string? value) => new ValueString(value);
```











```
public static implicit operator string(ValueString val) => val.Value;
```

```
public static implicit operator ValueString(char[] value) => new ValueString(value);
```

```
string? strNull = null;  
string strNotNull = strNull!;  
string strHelloWorld = "Hello, World!";  
  
ValueString defaultValueString = default;  
ValueString defaultValueString2 = null;  
ValueString ValueStringStrNull = strNull;  
ValueString valueStrHelloWorld = strHelloWorld;  
ValueString text3 = "Text3";
```

```
string stringText3 = valueStringText3;  
string? strNotNull2 = defaultValueString;
```

```
char[] charArray = ['A', 'r', 'r'];
```

 strNull	null
 strNotNull	null
 strHelloWorld	"Hello, World!"
▶  defaultValueString	""
▶  ValueStringStrNull	""
▶  valueStrHelloWorld	"Hello, World!"
▶  valueStringText3	"Text3"
 stringText3	"Text3"
 strNotNull2	""
▶  fromArray	"Arr"



# Структура ValueString

## Передача параметров и возврат значений

```
private static ValueString Method1(string text)
{
    return text;
}
```

```
private static string Method2(ValueString ValueString)
{
    return ValueString;
}
```

```
string x1 = Method1("11111");
```

```
ValueString x2 = Method2("11112");
```

# Структура ValueString

## Модифицированная структура PassengerName

```
#nullable enable
```

```
public struct PassengerNameStruct
{
    public ValueString LastName { get; set; }

    public ValueString FirstName { get; set; }




    public PassengerNameStruct(string lastName, string firstName)
    {
        LastName = lastName;
        FirstName = firstName;
    }
}
```

# Структура ValueString

## Модифицированная структура PassengerName

```
PassengerNameStruct paxName = default;
```

```
name = paxName.LastName.ToUpper();
```

 paxName2	{App.PassengerName
▶  FirstName	...
▶  LastName	...

# Структура ValueString

- IEquatable<ValueString>, IEquatable<string>
- IComparable<ValueString>, IComparable<string>
- operator ==, operator !=

```
if (valueStrHelloWorld == "Hello, World!")  
{  
  
}  
  
if ("Hello, World!" == valueStrHelloWorld)  
{  
  
}
```

# Структура ValueString

## Новые свойства для удобства

```
public bool IsEmpty => string.IsNullOrEmpty(_value);
```

```
public bool IsNotEmpty => !string.IsNullOrEmpty(_value);
```

```
public bool IsEmptyOrWhiteSpace => string.IsNullOrEmpty(_value);
```

```
public bool IsNotEmptyOrWhiteSpace => !string.IsNullOrEmpty(_value);
```

# Структура ValueString

- НОВЫЕ СВОЙСТВА

```
if (valueStringStrNull.IsEmpty)
{
    Console.WriteLine("valueStringStrNull.IsEmpty");
}
```

```
if (valueStrHelloWorld.IsNotEmpty)
{
    Console.WriteLine("valueStrHelloWorld.IsEmpty");
}
```

- ВМЕСТО КЛАССИЧЕСКИХ

```
if (string.IsNullOrEmpty(valueStringStrNull))
{
    Console.WriteLine("valueStringStrNull.IsEmpty");
}
```

```
if (!string.IsNullOrEmpty(valueStringStrNull))
{
    Console.WriteLine("valueStringStrNull.IsEmpty");
}
```

# Структура ValueString

## Дополнительные методы расширения

```
public static class ValueStringExt
{
    public static ReadOnlySpan<char> AsSpan(this string? Text)

    public static ValueString ToValueString(this string? text)

    public static string? NullIfEmpty(this ValueString value)

    public static ValueString NotEmptyOr(this ValueString value, ValueString defaultValue)

    public static int? TryParseInt(this ValueString value)

    public static long? TryParseLong(this ValueString value)

    public static double? TryParseDouble(this ValueString value)

    public static decimal? TryParseDecimal(this ValueString value)

    public ValueString Left(int charCount)
```

# Структура ValueString

## Дополнительные методы расширения

```
string strNumber1 = "1112";
```

```
int? int1 = strNumber1.ToValueString().TryParseInt();
```

```
long? int64 = strNumber1.ToValueString().TryParseLong();
```

```
decimal? decimal1 = strNumber1.ToValueString().TryParseDecimal();
```

```
double? double1 = strNumber1.ToValueString().TryParseDouble();
```

```
int int2 = valueStringStrNull.TryParseInt() ?? 0;
```



# Структура ValueString

## Поддержка коллекций

```
public struct ValueString : IEquatable<ValueString>, IEquatable<string>
{
    }
}
```

```
Dictionary<ValueString, ValueString> dict = new();
```

```
dict["111"] = "222";
```

```
ValueString val = dict["111"]; // val == "222"
```

# Нюансы совместимости

- Сравнение с **null** оператором **is**

```
if (str is null)
{
    throw new NullReferenceException("str is null");
}
```

Ошибка компиляции *"Cannot convert null to 'ValueString' because it is a non-nullable value type"*

# Нюансы совместимости

- Сравнение с **null** оператором **==**

```
if (str == null)
{
    throw new NullReferenceException("str is null");
}
```

Всегда false

```
public static bool operator ==(ValueString left, string? right)
{
    return left.Equals(right);
}
```

```
public bool Equals(string? value)
{
    return Value.Equals(value);
}
```

# Нюансы совместимости

- Методы расширения

```
public static string Format(this string format, string arg0)
    => string.Format(format, arg0);
```

```
ValueString myFormat = "Hi {0}";
```

```
myFormat.Format("Yury").
```



CS1929

'ValueString' does not contain a definition for 'Format' and the best extension method overload  
'ValueStringExt.Format(string, string)' requires a receiver of type 'string'

- Исправление – новый метод расширения или использование свойства .Value

```
myFormat.Value.Format("Yury");
```

```
MyFormat.ToString().Format("Yury");
```

# Нюансы совместимости

- Метод с параметром типа Object, без перегрузок

```
private static string? Method3(object? obj)
{
    return obj as string;
}
```

```
// string text = "Text";
ValueString text = "Text";
```

```
var x = Method3(text);
// !boxing
```

# Нюансы совместимости

- Метод с параметром типа Object, без перегрузок

```
private static string? Method3(object? obj)
{
    return obj as string;
}
```

```
// string text = "Text";
ValueString text = "Text";
```

```
var x = Method3(text);
```

```
var x = Method3(text.Value);
```

# Нюансы совместимости

- Работа с IEnumerable<char>, LINQ-запросы

```
int countA = text.Count(x => x == 'A'); // !boxing
```

# Нюансы совместимости

- Работа с IEnumerable<char>, LINQ-запросы

```
int countA = text.Count(x => x == 'A');
```

```
int countA = text.Value.Count(x => x == 'A');
```



# Нюансы совместимости

## Сериализация XML

```
[XmlText]
public string Value

public class TestClass
{
    public string? Str1 { get; set; }

    public string Str2 { get; set; } = string.Empty;

    public ValueString ValStr1 { get; set; }

    public ValueString ValStr2 { get; set; }
}

var myClass = new TestClass { Str1 = "s1", ValStr1 = "s2" };
```

# Нюансы совместимости

## Сериализация XML

```
var xs = new XmlSerializer(typeof(TestClass));
```

```
xs.Serialize(sw, myClass);
```

```
<TestClass >  
  <Str1>s1</Str1>  
  <Str2 />  
  <ValStr1>s2</ValStr1>  
  <ValStr2></ValStr2>  
</TestClass>
```

# Нюансы совместимости

## Сериализация XML

```
public class TestClass
{
    [XmlElement(IsNullable = true)]
    public string? Str1 { get; set; }

    public string Str2 { get; set; } = string.Empty;

    public ValueString ValStr1 { get; set; }

    public ValueString ValStr2 { get; set; }
}

var myClass = new TestClass { Str2 = "s2", ValStr2 = "s2" };
```

# Нюансы совместимости

## Сериализация XML

```
var xs = new XmlSerializer(typeof(TestClass));
```

```
xs.Serialize(sw, myClass);
```

```
<TestClass >
```

```
<Str1 xsi:nil="true" />
```

```
<Str2>s2</Str2>
```

```
<ValStr1>s2</ValStr1>
```

```
<ValStr2></ValStr2>
```

```
</TestClass>
```

# Нюансы совместимости

## Сериализация XML

```
public class TestClass
{
    [XmlElement(IsNullable = true)]
    public string? Str1 { get; set; }

    public string Str2 { get; set; } = string.Empty;

    [XmlElement(IsNullable = true)]
    public ValueString ValStr1 { get; set; }

    public ValueString ValStr2 { get; set; }
}

var myClass = new TestClass { Str2 = "s2", ValStr2 = "s2" };
```

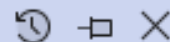
# Нюансы совместимости

## Сериализация XML

```
var xs = new XmlSerializer(typeof(TestClass));
```



Exception Unhandled



**System.InvalidOperationException:** 'There was an error reflecting type 'ValueStringTestApp.Objects.TestClass'.'

◀ 2 of 2 Inner Exceptions ▶

InvalidOperationException: IsNullable may not be 'true' for value type MY.ValueString. Please consider using Nullable<MY.ValueString>

# Нюансы совместимости

## Сериализация XML

```
public class TestClass
{
    public string? Str1 { get; set; }

    [XmlAttribute]
    public string Str2 { get; set; } = string.Empty;

    public ValueString ValStr1 { get; set; }

    [XmlAttribute]
    public ValueString ValStr2 { get; set; }
}

var myClass = new TestClass { Str2 = "s2", ValStr2 = "s2" };
```

# Нюансы совместимости

## Сериализация XML

```
var xs = new XmlSerializer(typeof(TestClass));
```



Exception Unhandled

◀ 2 of 2 Inner Exceptions ▶

InvalidOperationException: Cannot serialize member 'ValStr2' of type MY.ValueString. XmlAttribute/XmlText cannot be used to encode complex types.



# Нюансы совместимости

## Сериализация XML

```
public class TestClass
{
    private ValueString _valStr1;
    private ValueString _valStr2;

    public string? Str1 { get; set; }

    [XmlAttribute]
    public string Str2 { get; set; } = string.Empty;

    public string ValStr1 { get => _valStr1; set => _valStr1 = value; }

    [XmlAttribute]
    public string ValStr2 { get => _valStr2; set => _valStr2 = value; }
}
```

# Нюансы совместимости

## Сериализация JSON

```
public sealed class JsonValueStringConverter : JsonConverter<ValueString>
{
    public override ValueString Read(ref Utf8JsonReader reader,
                                     typeToConvert,
                                     JsonSerializerOptions options)
    {
        return reader.TokenType switch
        {
            JsonTokenType.Null => ValueString.Empty,
            _ => reader.GetString(),
        };
    }

    public override void Write(Utf8JsonWriter writer,
                               ValueString value,
                               JsonSerializerOptions options)
    {
        writer.WriteStringValue(value);
    }
}
```

# Нюансы совместимости

## Сериализация JSON

```
var options = new System.Text.Json.JsonSerializerOptions();  
options.Converters.Add(new JsonValueStringConverter());
```

```
var json = System.Text.JsonSerializer.Serialize(myClass);  
{  
    "Str1": "s1",  
    "Str2": "",  
    "ValStr1": "s1",  
    "ValStr2": "",  
}
```

# Нюансы совместимости

## EntityFramework

```
public class ValueStringConverter : ValueConverter<ValueString, string>
{
    public ValueStringConverter()
        : base(v => v.Value, v => new ValueString(v))
    {
    }
}

public class ValueStringNullableConverter : ValueConverter<ValueString, string?>
{
    public ValueStringNullableConverter()
        : base(v => v.NullIfEmpty(), v => new ValueString(v))
    {
    }
}
```

# Нюансы совместимости

## EntityFramework

```
public class Hotel
{
    public long HotelId { get; set; }

    public ValueString HotelName { get; set; } // not null

    public ValueString HotelPhone { get; set; } // can be null
}

public class HotelDataContext : DbContext
{
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Hotel>(entity =>
        {
            entity.HasKey(e => e.HotelId);
            entity.Property(x => x.HotelName)
                                .HasConversion(new
ValueStringConverter());
            entity.Property(x => x.HotelPhone)
```

# Тестирование производительности в BenchmarkDotNet

```
// * Summary *  
  
BenchmarkDotNet v0.14.0, Windows 11 (10.0.26100.2605)  
AMD Ryzen 5 5600U with Radeon Graphics, 1 CPU, 12 logical and 6 physical cores  
.NET SDK 9.0.101  
  [Host]           : .NET 9.0.0 (9.0.24.52809), X64 RyuJIT AVX2  
  NET 9.0 RyuJIT : .NET 9.0.0 (9.0.24.52809), X64 RyuJIT AVX2  
  
Job=NET 9.0 RyuJIT  PowerPlanMode=3acc4505-bf6e-4630-9a5f-a4637dd5f458  Runtime=.NET 9.0
```

# Тестирование производительности в BenchmarkDotNet

## IndexOf()

```
[Benchmark(Baseline = true)]
public void StringIndexOf()
{
    res = StringToTest.IndexOf(StringToSearch, StringComparison.OrdinalIgnoreCase);
}

[Benchmark]
public void ValueStringIndexOf()
{
    res = ValueStringToTest.IndexOf(StringToSearch, StringComparison.OrdinalIgnoreCase);
}
```

The object **type** is an alias for `System.Object` in .NET. In the unified type system of C#, all types, predefined and user-defined, reference types and value types, inherit directly or indirectly from `System.Object`. You can assign values of any type (except `ref struct`, see `ref struct`) to variables of type `object`. Any object variable can be assigned to its default value using the literal `null`. When a variable of a value type is converted to `object`, it's said to be boxed. When a variable of type `object` is converted to a value type, it's said to be **unboxed**.

# Тестирование производительности в BenchmarkDotNet

## IndexOf()

Method	StringToSearch	Mean	Error	Ratio	RatioSD	Allocated
StringIndexOf	type	22.11 ns	0.324 ns	1.00	0.03	-
ValueStringIndexOf	type	22.26 ns	0.454 ns	1.01	0.04	-
StringIndexOf	unboxed	77.14 ns	0.661 ns	1.00	0.02	-
ValueStringIndexOf	unboxed	77.35 ns	1.205 ns	1.00	0.03	-



# Тестирование производительности в BenchmarkDotNet

## IndexOf()

## .NET 9.0.0 (9.0.24.52809), X64 RyuJIT AVX2 ```assembly ; Benchmarks.StringIndexOfBench.SearchInValueStringToTest()	=	## .NET 9.0.0 (9.0.24.52809), X64 RyuJIT AVX2 ```assembly ; Benchmarks.StringIndexOfBench.SearchInString()
push rbx	=	push rbx
sub rsp,30		sub rsp,30
mov rbx,rcx		mov rbx,rcx
lea r9,[rbx+20]	<>	mov rcx,[rbx+8]
mov rdx,[rbx+10]	=	mov rdx,[rbx+10]
mov rcx,[r9] mov r9,10C00100008 test rcx,rcx cmove rcx,r9	+-	
mov dword ptr [rsp+20],5 mov r9d,[rcx+8] xor r8d,r8d	=	mov dword ptr [rsp+20],5 mov r9d,[rcx+8] xor r8d,r8d
call qword ptr [7FFCDFE353E0]; System.String.IndexOf(System.String, Int32, Int32, System.StringComparison)	<>	call qword ptr [7FFCDFE55488]; System.String.IndexOf(System.String, Int32, Int32, System.StringComparison)
mov [rbx+18],eax add rsp,30 pop rbx ret	=	mov [rbx+18],eax add rsp,30 pop rbx ret
; Total bytes of code 66	<>	; Total bytes of code 46

# Тестирование производительности в BenchmarkDotNet

## Substring(16)

```
[Benchmark(Baseline = true)]  
public void StringSubString()  
{  
    res = StringToTest.Substring(16);  
}
```

```
[Benchmark]  
public void ValueStringSubString()  
{  
    res = ValueStringToTest.Substring(16);  
}
```

# Тестирование производительности в BenchmarkDotNet

## Substring(16)

Method	Mean	Error	Ratio	RatioSD	Allocated	Alloc Ratio
StringSubString	106.45 ns	9.054 ns	1.01	0.16	1.08 KB	1.00
ValueStringSubstring	97.42 ns	2.708 ns	0.93	0.11	1.08 KB	1.00

# Тестирование производительности в BenchmarkDotNet

## List<string>, List<ValueString> Add(1000)

```
[Benchmark(Baseline = true)]
public void AddString()
{
    List<string> stringList = new();
    for (int i = 0; i < Count; i++)
    {
        var str = Values[i];
        stringList.Add(str);
    }
}

[Benchmark]
public void AddValueString()
{
    List<ValueString> valueStringList = new();
    for (int i = 0; i < Count; i++)
    {
        var str = Values[i];
        valueStringList.Add(str);
    }
}
```

# Тестирование производительности в BenchmarkDotNet

**List<string>, List<ValueString> Add(1000)**

Method	Mean	Error	Ratio	RatioSD	Allocated	Alloc Ratio
-----	-----:	-----:	-----:	-----:	-----:	-----:
AddString	7.073 us	0.0909 us	1.00	0.03	16.21 KB	1.00
AddValueString	6.901 us	0.0644 us	0.98	0.02	16.21 KB	1.00

# Тестирование производительности в BenchmarkDotNet

## Dictionary<T1, T2> Add [1000]

```
Dictionary<string, string> Str_StrDict = new();
```

```
Dictionary<ValueString, ValueString> ValStr_ValStrDict = new();
```

```
Dictionary<string, ValueString> Str_ValStrDict = new();
```

```
Dictionary<ValueString, string> ValStr_StrDict = new();
```

# Тестирование производительности в BenchmarkDotNet

## Dictionary<T1, T2> Add [1000]

```
[Benchmark(Baseline = true)]
public void Add_String_String()
{
    Dictionary<string, string> Str_StrDict = new();
    for (int i = 0; i < Values.Length; i++)
    {
        var str = Values[i];
        Str_StrDict[str] = str;
    }
}
```

```
[Benchmark]
public void Add_ValString_ValString()
{
    Dictionary<ValueString, ValueString> ValStr_ValStrDict = new();
    for (int i = 0; i < Values.Length; i++)
    {
        var str = Values[i];
        ValStr_ValStrDict[str] = str;
    }
}
```

# Тестирование производительности в BenchmarkDotNet

## Dictionary<T1, T2> Add [1000]

Method	Mean	Error	Ratio	RatioSD	Allocated	Alloc Ratio
Add_ValString_String	42.93 us	0.567 us	1.16	0.13	99.82 KB	1.00
Add_ValString_ValString	43.27 us	1.170 us	1.16	0.14	99.82 KB	1.00
Add_String_String	37.65 us	2.993 us	1.01	0.17	99.82 KB	1.00
Add_String_ValString	33.32 us	0.689 us	0.90	0.11	99.82 KB	1.00

## ConcurrentDictionary<T1, T2> Add [1000]

Method	Mean	Error	Ratio	RatioSD	Allocated	Alloc Ratio
Add_ValString_String	140.8 us	10.05 us	1.05	0.11	197.97 KB	0.91
Add_ValString_ValString	129.2 us	10.54 us	0.96	0.12	206.03 KB	0.95
Add_String_String	134.0 us	3.33 us	1.00	0.05	217.14 KB	1.00
Add_String_ValString	139.7 us	3.67 us	1.04	0.05	217.14 KB	1.00



# Тестирование производительности в BenchmarkDotNet

## Dictionary<T1, T2>, ConcurrentDictionary<T1, T2> FindValue(str)

```
[Benchmark(Baseline = true)]
public void Str_Str_DictFindValue()
{
    Value = Str_StrDict[KeyToSearch];
}

[Benchmark]
public void ValStr_ValStr_DictFindValue()
{
    Value = ValStr_ValStrDict[KeyToSearch];
}

[Benchmark]
public void Str_ValStr_DictFindValue()
{
    Value = Str_ValStrDict[KeyToSearch];
}

[Benchmark]
public void ValStr_Str_DictFindValue()
{

```

# Тестирование производительности в BenchmarkDotNet

## Dictionary<T1, T2> FindValue(str)

Method	Mean	Error	Ratio	RatioSD
ValStr_Str_DictFindValue	23.29 ns	0.341 ns	1.04	0.02
ValStr_ValStr_DictFindValue	23.48 ns	0.216 ns	1.05	0.02
Str_Str_DictFindValue	22.34 ns	0.403 ns	1.00	0.02
Str_ValStr_DictFindValue	22.42 ns	0.354 ns	1.00	0.02

## ConcurrentDictionary<T1, T2> FindValue(str)

Method	Mean	Error	Ratio	RatioSD	Allocated	Alloc Ratio
ValStr_Str_DictFindValue	27.48 ns	0.580 ns	1.06	0.05	-	NA
ValStr_ValStr_DictFindValue	28.30 ns	0.593 ns	1.09	0.04	-	NA
Str_Str_DictFindValue	25.88 ns	0.471 ns	1.00	0.04	-	NA
Str_ValStr_DictFindValue	26.12 ns	0.549 ns	1.01	0.04	-	NA

# Тестирование производительности в BenchmarkDotNet

## XML Serialization / Deserialization

```
public class PassengerName1
{
    public string LastName { get; set; } = string.Empty;

    public string FirstName { get; set; } = string.Empty;
}

public class Passenger1
{
    public PassengerName1? Name { get; set; }

    public string Text { get; set; } = string.Empty;
}

public class PassengerList1
{
    public List<Passenger1> List { get; set; } = new();
}
```

# Тестирование производительности в BenchmarkDotNet

## XML Serialization / Deserialization

```
public class PassengerName2
{
    public ValueString LastName { get; set; }

    public ValueString FirstName { get; set; }
}

public class Passenger2
{
    public PassengerName2? Name { get; set; }

    public ValueString Text { get; set; }
}

public class PassengerList2
{
    public List<Passenger2> List { get; set; } = new();
}
```

# Тестирование производительности в BenchmarkDotNet

## XML Serialization / Deserialization

```
public class PassengerName3
{
    private ValueString _lastName;
    private ValueString _firstName;

    public string LastName { get => _lastName; set => _lastName = value; }

    public string FirstName { get => _firstName; set => _firstName = value; }
}
```

```
public class Passenger3
{
    private ValueString _text;

    public PassengerName3? Name { get; set; }

    public string Text { get => _text; set => _text = value; }
}
```

```
public class PassengerList3
{
```

# Тестирование производительности в BenchmarkDotNet

## XML Serialization

```
[Benchmark(Baseline = true)]
public void SerializeXmlPropertiesStr()
{
    using var cachedStream = RecyclableMemoryStreamManager.GetStream("1");
    XmlSerializer1?.Serialize(cachedStream, PassengerList1);
}
```

```
[Benchmark]
public void SerializeXmlPropertiesValStr()
{
    using var cachedStream = RecyclableMemoryStreamManager.GetStream("1");
    XmlSerializer2?.Serialize(cachedStream, PassengerList2);
}
```

```
[Benchmark]
public void SerializeXmlBackFieldsValStr()
{
    using var cachedStream = RecyclableMemoryStreamManager.GetStream("1");
    XmlSerializer3?.Serialize(cachedStream, PassengerList3);
}
```

# Тестирование производительности в BenchmarkDotNet

## XML Deserialization

```
[Benchmark(Baseline = true)]
public void DeserXmlPropStr()
{
    PassengerList1 = (PassengerList1)XmlSerializer1!.Deserialize(new StringReader(Xml1))!;
}

[Benchmark]
public void DeserXmlPropValStr()
{
    PassengerList2 = (PassengerList2)XmlSerializer2!.Deserialize(new StringReader(Xml2))!;
}

[Benchmark]
public void DeserXmlBackFieldValStr()
{
    PassengerList3 = (PassengerList3)XmlSerializer3!.Deserialize(new StringReader(Xml3))!;
}
```

# Тестирование производительности в BenchmarkDotNet

## XML Serialization

Method	Mean	Error	Ratio	RatioSD	Gen0	Allocated	Alloc Ratio
SerXmlPropertiesStr	1.901 ms	0.0164 ms	1.00	0.02	3.9063	8.77 KB	1.00
SerXmlPropertiesValStr	3.164 ms	0.0250 ms	1.66	0.03	62.5000	149.39 KB	17.04
SerXmlBackFieldsValStr	1.855 ms	0.0069 ms	0.98	0.01	3.9063	8.77 KB	1.00

## XML Deserialization

Method	Mean	Error	Ratio	RatioSD	Gen0	Gen1	Allocated	Alloc Ratio
DeserXmlPropStr	2.723 ms	0.0291 ms	1.00	0.02	46.8750	15.6250	225.11 KB	1.00
DeserXmlPropValStr	4.885 ms	0.0276 ms	1.79	0.03	93.7500	62.5000	576.72 KB	2.56
DeserXmlBackFieldValStr	2.726 ms	0.0305 ms	1.00	0.02	31.2500	15.6250	225.11 KB	1.00



# Тестирование производительности в BenchmarkDotNet

## Json Serialization

```
[Benchmark(Baseline = true)]
public void SerJsonPropsStr()
{
    Json = JsonSerializer.Serialize(PassengerList1);
}

[Benchmark]
public void SerJsonPropsValStr()
{
    //[GlobalSetup]: options.Converters.Add(new JsonValueStringConverter());
    Json = JsonSerializer.Serialize(PassengerList2, options);
}

[Benchmark]
public void SerJsonBackFieldsValStr()
{
    Json = JsonSerializer.Serialize(PassengerList3);
}
```

# Тестирование производительности в BenchmarkDotNet

## Json Serialization

```
[Benchmark(Baseline = true)]
public void DeserJsonPropsStr()
{
    PassengerList1 = JsonSerializer.Deserialize<PassengerList1>(Json1);
}

[Benchmark]
public void DeserJsonPropsValStr()
{
    //[GlobalSetup]: options.Converters.Add(new JsonValueStringConverter());
    PassengerList2 = JsonSerializer.Deserialize<PassengerList2>(Json2, options);
}

[Benchmark]
public void DeserJsonBackFieldsValStr()
{
    PassengerList3 = JsonSerializer.Deserialize<PassengerList2>(Json3);;
}
```

# Тестирование производительности в BenchmarkDotNet

## Json Serialization

Method	Mean	Error	Ratio	RatioSD	Gen0	Gen1	Gen2	Allocated	Alloc Ratio
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----
SerJsonPropsStr	677.9 us	24.49 us	1.00	0.08	49.8047	49.8047	49.8047	157.96 KB	1.00
SerJsonPropsValStr	674.9 us	7.39 us	1.00	0.06	49.8047	49.8047	49.8047	157.96 KB	1.00
SerJsonBackFieldsValStr	681.5 us	11.20 us	1.01	0.06	49.8047	49.8047	49.8047	157.96 KB	1.00

## Json Deserialization

Method	Mean	Error	Ratio	RatioSD	Gen0	Gen1	Allocated	Alloc Ratio
-----	-----	-----	-----	-----	-----	-----	-----	-----
DeserJsonPropsStr	1.177 ms	0.0145 ms	1.00	0.03	39.0625	31.2500	212.46 KB	1.00
DeserJsonPropsValStr	1.188 ms	0.0070 ms	1.01	0.02	39.0625	33.2031	212.46 KB	1.00
DeserJsonBackFieldsValStr	1.161 ms	0.0254 ms	0.99	0.04	41.0156	29.2969	212.46 KB	1.00

# Выводы

**ValueString** - тонкая структура-обёртка над типом **string**

**Pro:**

- + Ненулевые значения независимо от способа инициализации
- + Позволяет прозрачно заменить **string** в большинстве случаев
- + Расширяет функциональность типа **string** дополнительными методами и свойствами
- + Можно свободно сочетать с типом **string** при работе с другими библиотеками
- + Производительность при вызове базовых функций практически не меняется
- + Подходит для (де)сериализации JSON (и условно для XML)

# Выводы

## Contra:

- При командной разработке потребуется однообразие подхода, однотипное использование строк всей командой
- Для моделей EF и других ORM, а также сериализации JSON требуется конвертер
- Сериализация свойств **ValueString** в XML медленнее из-за боксинга (лучше использовать в полях)

# Спасибо за внимание!

**Вопросы? Отзывы? Предложения?**

<https://github.com/ymalich/ValueString>



<https://gitverse.ru/ymalich/ValueString>

