



C# 10. Interpolated string handlers

Зачем и для чего?

О чем поговорим



Что?



Зачем?



Как?



Что нам с этим делать?

1. Что?

Возможность контролировать процесс создания интерполируемой строки

```
var a = 15;  
var b = 1;  
var c = $"This is A: {a}, and this is B: {b}";
```



Не нашего ума дело



"This is A: 15, and this is B: 1"

Здесь будут все изменения

2. Зачем?

C# 9. Декомпилируем интерполяцию (sharplab.io или dotPeek)

```
var a = 15;  
var b = 1;  
var c = $"This is A: {a}, and this is B: {b}";
```



```
var a = 15;  
var b = 1;  
var c = string.Format("This is A: {0}, and this is B: {1}", a, b);  
  
Console.WriteLine(c);
```

Недостатки:

- params
- боксинг
- сложно контролировать
- строка создаётся всегда

2. Зачем?

C# 10. Декомпилируем интерполяцию

```
var a = 15;  
var b = 1;  
var c = $"This is A: {a}, and this is B: {b}";
```



```
var a = 15;  
var b = 1;  
var handler = new DefaultInterpolatedStringHandler(28, 2);  
handler.AppendLiteral("This is A: ");  
handler.AppendFormatted(a);  
handler.AppendLiteral(", and this is B: ");  
handler.AppendFormatted(b);  
string c = handler.ToStringAndClear();  
  
Console.WriteLine(c);
```

Недостатки:

- ~~params~~
- ~~боксинг~~
- ~~сложно контролировать~~
- строка создаётся всегда ...

3. Как?

Сигнатура дефолтного хэндлера

```
[InterpolatedStringHandler]
public ref struct DefaultInterpolatedStringHandler
{
    public DefaultInterpolatedStringHandler(int literalLength, int formattedCount);
    public DefaultInterpolatedStringHandler(int literalLength, int formattedCount, System.IFormatProvider? provider);
    public DefaultInterpolatedStringHandler(int literalLength, int formattedCount, System.IFormatProvider? provider,
        System.Span<char> initialBuffer);

    public void AppendLiteral(string value);

    public void AppendFormatted<T>(T value);
    public void AppendFormatted<T>(T value, string? format);
    public void AppendFormatted<T>(T value, int alignment);
    public void AppendFormatted<T>(T value, int alignment, string? format);

    public void AppendFormatted(ReadOnlySpan<char> value);
    public void AppendFormatted(ReadOnlySpan<char> value, int alignment = 0, string? format = null);

    public void AppendFormatted(string? value);
    public void AppendFormatted(string? value, int alignment = 0, string? format = null);
    public void AppendFormatted(object? value, int alignment = 0, string? format = null);

    public string ToStringAndClear();
}
```

3. Как?

Минимально необходимый набор

```
[InterpolatedStringHandler]  
public ref struct DefaultInterpolatedStringHandler  
{  
    public DefaultInterpolatedStringHandler(int literalLength, int formattedCount);  
  
    public void AppendLiteral(string value);  
    public void AppendFormatted<T>(T value);  
  
    public string ToStringAndClear();  
}
```

Что потребуется:

1. Атрибут `[InterpolatedStringHandler]`
2. Конструктор с параметрами `literalLength` и `formattedCount`
3. Методы `AppendLiteral` и `AppendFormatted`
4. Любое количество перегрузок `AppendFormatted` с нужными типами параметров

3. Как?

Наш собственный хэндлер

```
[InterpolatedStringHandler]
public ref struct MyInterpolatedStringHandler
{
    private readonly StringBuilder _stringBuilder = new();

    public MyInterpolatedStringHandler(int literalLength, int formattedCount){}

    public void AppendLiteral(string value) => _stringBuilder.Append(value);

    public void AppendFormatted<T>(T value)
    {
        _stringBuilder.Append("\x1b[31;1m");
        _stringBuilder.Append(value);
        _stringBuilder.Append("\x1b[0m");
    }

    public string GetString() => _stringBuilder.ToString();
}
```


3. Как?

Используем наш хэндлер

```
void UseHandlerForSomething(ref MyInterpolatedStringHandler handler)
{
    var text = handler.GetString();
    // ...
}

var a = 15;
var b = 1;

UseHandlerForSomething($"This is A: {a}, and this is B: {b}");
```

Или даже проще

```
var a = 15;
var b = 1;
MyInterpolatedStringHandler handler = $"This is A: {a}, and this is B: {b}";
string c = handler.GetString();
```

3. Как?

Что ещё могут хэндлеры

- Передача контекста при вызове хэндлера
- Условное выполнение

3. Как?

Декомпилируем Debug.Assert

```
var a = 15;  
var b = 1;  
Debug.Assert(a == b, $"This is A: {a}, and this is B: {b}");
```



```
var a = 15;  
var b = 1;  
var condition = a == b;  
var msg = new AssertInterpolatedStringHandler(28, 2, condition, out var shouldAppend);  
if (shouldAppend)  
{  
    msg.AppendLiteral("This is A: ");  
    msg.AppendFormatted(a);  
    msg.AppendLiteral(", and this is B: ");  
    msg.AppendFormatted(b);  
}  
Debug.Assert(condition, ref msg);
```

3. Как?

Тело метода Debug.Assert

```
public static void Assert(  
    bool condition,  
    [InterpolatedStringHandlerArgument("condition")] ref AssertInterpolatedStringHandler message)  
{  
    Assert(condition, message.ToStringAndClear());  
}
```

3. Как?

Конструктор хэндлера `AssertInterpolatedStringHandler`

```
public AssertInterpolatedStringHandler(  
    int literalLength,  
    int formattedCount,  
    bool condition,  
    out bool shouldAppend)  
{  
    if (condition)  
    {  
        shouldAppend = false;  
        _stringBuilder = null;  
    }  
    else  
    {  
        shouldAppend = true;  
    }  
}
```

Недостатки:

- ~~params~~
- ~~боксинг~~
- ~~сложно контролировать~~
- ~~строка создается всегда~~

4. А что нам с этим делать?

Примеры применения

- Контроль длины передаваемого в плейсхолдер значения
- Экранирование параметров SQL запроса
- Логирование
- Структурное логирование

4. А что нам с этим делать?

Структурное логирование

```
public void AppendFormatted<T>(
    T value,
    [CallerArgumentExpression("value")] string name = "")
{
    _arguments[name] = value.ToString();
    _builder.Append(value);
}
```

Использование хэндлера

```
var a = 15;
var b = 1;

logger.Log(
    LogLevel.Information,
    $"This is A: {a}, and this is B: {b}");
```

4. А что нам с этим делать?

Примеры применения

- Контроль длины передаваемого в плейсхолдер значения
- Экранирование параметров SQL запроса
- Логирование
- Структурное логирование
- Бонусный пример



4. А что нам с этим делать?

Бонус. Парсинг строк как в scanf

```
string input = "Name: Andrew; Age: 31";
```

```
string? name = null;
```

```
int age = 0;
```

```
if (input.TryParse($"Name: {{Placeholder(ref name)}}; Age: {{Placeholder(ref age)}}"))  
{  
    Console.WriteLine($"{name} {age}");  
}  
else  
{  
    Console.WriteLine("Does not match :(");  
}
```

Спасибо Андрею Карпову (<https://twitter.com/akarpov89>)

Немного полезных ссылок

- Мануал по созданию своего хэндлера: <https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/tutorials/interpolated-string-handler>
- Proposal фичи: <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/proposals/csharp-10.0/improved-interpolated-strings>
- Крутая статья про изменения в интерполируемых строках: <https://devblogs.microsoft.com/dotnet/string-interpolation-in-c-10-and-net-6/>
- Ещё одна крутая статья по особенностям работы интерполируемых строк: <https://www.meziantou.net/interpolated-strings-advanced-usages.htm>
- И ещё одна статья с несколькими примерами применения: <https://btburnett.com/csharp/2021/12/17/string-interpolation-trickery-and-magic-with-csharp-10-and-net-6>
- Развёрнутый пример применения самописного хэндлера в структурном логировании: <https://habr.com/ru/post/590069/>
- Тот самый твит Андрея Карпова из последнего примера: <https://twitter.com/akarpov89/status/1443957219834400773>
- Репозиторий со сниппетами кода из презентации: <https://github.com/Alano13/interpolation-handlers>



Спасибо за внимание!

Вопросы?

Вадим Нестеров
Telegram: [@Alano13](https://t.me/@Alano13)