Webinar #1
7PM CEST
May 6

Topics:

- `ReadOnlySpan` **trick with** `byte[]`
- `SkipLocalsInit` **and** `Unsafe.SkipInit`

@konradkokosa / @dotnetosorg

About me

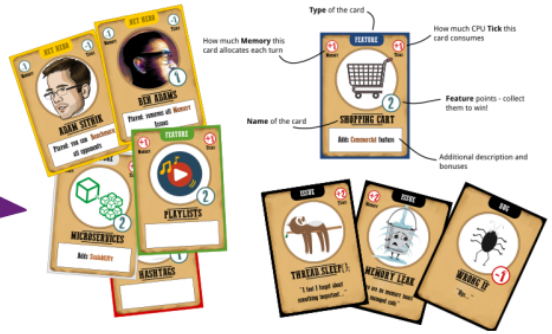
.NET freelancer/ trainer/ consultant/ speaker/ ...

.NET meetups/ conferences/ webinars/ ...

diagnosticsexpert.com

aSync expert();

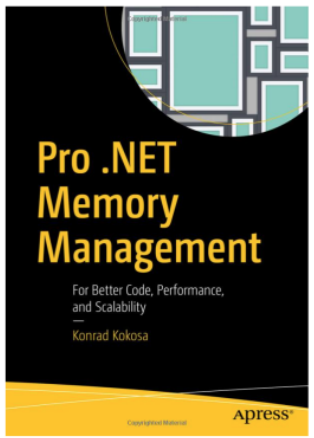
asyncexpert.com



OutOfMemory card game

(still prototyping)

playoutofmemory.com



prodotnetmemory.com

Microsoft® MVP Most Valuable Professional

TooSlowException.com

@konradkokosa

.NET GC Tips & Tricks series

- what it is NOT...?

.NET GC Tips & Tricks series

- what it is NOT...?
  - in DEPTH explanation of the most important .NET GC parts

.NET GC Tips & Tricks series

- what it is NOT...?
  - in DEPTH explanation of the most important .NET GC parts . I've done that already! .NET GC Internals took ~14h

.NET GC Tips & Tricks series

- what it is NOT...?
  - in DEPTH explanation of the most important .NET GC parts . I've done that already! .NET GC Internals took ~14h
- what it is...?

.NET GC Tips & Tricks series

- what it is NOT...?
  - in DEPTH explanation of the most important .NET GC parts . I've done that already! .NET GC Internals took ~14h
- what it is...?
  - practical debugging/diagnosing/measuring series 😇

.NET GC Tips & Tricks series

- what it is NOT...?
  - in DEPTH explanation of the most important .NET GC parts . I've done that already! .NET GC Internals took ~14h
- what it is...?
  - practical debugging/diagnosing/measuring series 😇
  - practical best-practices programming series

.NET GC Tips & Tricks series

- what it is NOT...?
  - in DEPTH explanation of the most important .NET GC parts . I've done that already! .NET GC Internals took ~14h
- what it is...?
  - practical debugging/diagnosing/measuring series 😇
  - practical best-practices programming series
  - **interactive** ~1h webinars - *ad hoc* drawings, live coding, **do not hesitate** to Q&A

.NET GC Tips & Tricks series

- what it is NOT...?
  - in DEPTH explanation of the most important .NET GC parts . I've done that already! .NET GC Internals took ~14h
- what it is...?
  - practical debugging/diagnosing/measuring series 😇
  - practical best-practices programming series
  - **interactive** ~1h webinars - *ad hoc* drawings, live coding, **do not hesitate** to Q&A
  - **various levels** - from beginners to advanced ones

Today's agenda

Explaining my two recent infographics:

- **`ReadOnlySpan`** trick with **`byte[]`** - 🌶️🌶️🌶️🌶️
- **`SkipLocalsInit`** and **`Unsafe.SkipInit`** - 🌶️🌶️🌶️🌶️🌶️

ReadOnlySpan trick with byte[]

```
[Benchmark]
public byte Create1()
{
  var array = new byte[] { 1, 2, 3, 4 };
  return array[0];
}
```

Method	Mean	Allocated
Create1	6.9263 ns	32 B

```
[Benchmark]
public byte Create2()
{
    ReadOnlySpan<byte> array = new byte[] { 1, 2, 3, 4 };
    return array[0];
}
```

Method	Mean	Allocated
Create2	0.0178 ns	-

Did you know...? The byte array assigned to **ReadOnlySpan<byte>** is a magic C# compiler trick to not allocate array at all.

dotnetmemoryexpert.com

`ReadOnlySpan` trick with `byte[]`

- Roslyn implementation

`ReadOnlySpan` trick with `byte[]`

- Roslyn implementation
- Use case #1 - memory-free initialization helper - we can see an example here

`ReadOnlySpan` trick with `byte[]`

- Roslyn implementation
- Use case #1 - memory-free initialization helper - we can see an example here
- Use case #2 - it works for `static` ROS too - we can see an example here

`ReadOnlySpan` trick with `byte[]`

- Roslyn implementation
- Use case #1 - memory-free initialization helper - we can see an example here
- Use case #2 - it works for `static` ROS too - we can see an example here
- Use case #3..64 - Look for `ReadOnlySpan<.*>.*(=|=>)\s+new (byte|sbyte)` in `dotnet/runtime`

`ReadOnlySpan` trick with `byte[]`

- Roslyn implementation
- Use case #1 - memory-free initialization helper - we can see an example here
- Use case #2 - it works for `static` ROS too - we can see an example here
- Use case #3..64 - Look for `ReadOnlySpan<.*>.*(=|=>)\s+new (byte|sbyte)` in `dotnet/runtime`
- Use case... yours? - Look for `(\sSpan<.*>|var|byte\[\]).*(=|=>)\s+new (byte|sbyte)\[\d*\]\s+\{` in your project :)

`ReadOnlySpan` trick with `byte[]`

- Roslyn implementation
- Use case #1 - memory-free initialization helper - we can see an example here
- Use case #2 - it works for `static` ROS too - we can see an example here
- Use case #3..64 - Look for `ReadOnlySpan<.*>.*(=|=>)\s+new (byte|sbyte)` in `dotnet/runtime`
- Use case... yours? - Look for `(\sSpan<.*>|var|byte\[\]).*(=|=>)\s+new (byte|sbyte)\[\d*\]\s+\{` in your project :)

  Trick spiciness: 🌶🌶🌶🌶

SkipLocalsInit and Unsafe.SkipInit

```
[SkipLocalsInit]
public bool TryDoSomething(out Guid value, out int consumed)
{
    Unsafe.SkipInit(out value);
    Unsafe.SkipInit(out consumed);
    if (DoMagic())
    {
        value = Guid.NewGuid();
        consumed = 16;
        return true;
    }
    return false;
}
```

Benchmark assumng DoMagic fails:

Method	Mean
Normal code	5.050 ns
SkipInit & attribute	3.243 ns

Did you know...? In C# you can relax the assigment requirement with **Unsafe.SkipInit** and also get rid of zero initialization with **SkipLocalsInit**.

dotnetmemoryexpert.com

`SkipLocalsInit` and `Unsafe.SkipInit`

- Use case #1 - don't initialize in case of failure - we can see an example here

`SkipLocalsInit` and `Unsafe.SkipInit`

- Use case #1 - don't initialize in case of failure - we can see an example here
  - BTW, F# allows that out of the box

`SkipLocalsInit` and `Unsafe.SkipInit`

- Use case #1 - don't initialize in case of failure - we can see an example here
  - BTW, F# allows that out of the box
- Use case #2 - help the compiler while doing `Unsafe` magic - here's simplified

`SkipLocalsInit` and `Unsafe.SkipInit`

- Use case #1 - don't initialize in case of failure - we can see an example here
  - BTW, F# allows that out of the box
- Use case #2 - help the compiler while doing `Unsafe` magic - here's simplified
- Use case #3 - fooling struct constructor which is mostly useful in structs as unions

`SkipLocalsInit` and `Unsafe.SkipInit`

- Use case #1 - don't initialize in case of failure - we can see an example here
  - BTW, F# allows that out of the box
- Use case #2 - help the compiler while doing `Unsafe` magic - here's simplified
- Use case #3 - fooling struct constructor which is mostly useful in structs as unions

Trick spiciness: 🌶🌶🌶🌶🌶

Thank you! Any questions?!