

C#の新機能勉強会

～ C#7、#8の新機能を活用して速く安全なプログラムを書こう ～



2020/02/14
小島 富治雄

アジェンダ



C# の歴史



C# Ver.	主な新機能	登場時期	.NET	Visual Studio
1.0, 1.1, 1.2	オブジェクト指向	2002年	.NET Framework 1.0,1.1	.NET, .NET 2003
2.0	ジェネリック	2005年	.NET Framework 2.0	2005
3.0	関数型	2007年	.NET Framework 2.0, 3.0, 3.5	2008, 2010
4.0	動的	2010年	.NET Framework 4	2010
5.0	非同期	2012年	.NET Framework 4.5	2012, 2013
6.0	Roslyn (コンパイラーをC#で実装しオープンソース化)	2015年	.NET Framework 4.6 .NET Core 1.0	2015
7.0, 7.1, 7.2, 7.3	パターン マッチング、値型に関する改良	2017年	.NET Framework 4.6.2, 4.7, 4.7.1, 4.7.2 .NET Core 2.0, 2.1, 2.2	2017
8.0	値型、参照型に関する改良	2019年	.NET Core 3.0	2019 Ver.16.3

C# 7~8



C# Ver.	Visual Studio
7.0	Visual Studio 2017
7.1	Visual Studio 2017 バージョン 15.3
7.2	Visual Studio 2017 バージョン 15.5
7.3	Visual Studio 2017 バージョン 15.7
8.0	Visual Studio 2019 16.3

C# 7~8



ターゲット フレーム	バージョン	C# 言語の既定のバージョン
.NET Core	3.x	C# 8.0
.NET Core	2.x	C# 7.3
.NET Standard	2.1	C# 8.0
.NET Standard	2.0	C# 7.3
.NET Standard	1.x	C# 7.3
.NET Framework	all	C# 7.3

プロジェクト ファイルでの C#のバージョン指定

```
<Project .....>
  <PropertyGroup>
    <OutputType>.....</OutputType>
    <TargetFramework>.....</TargetFramework>
    <!--.....中略.....-->

    <LangVersion>8.0</LangVersion>

    <Nullable>enable</Nullable>
  </PropertyGroup>
  <!--.....中略.....-->
</Project>
```

LangVersion	説明
preview	最新プレビュー バージョン
latest	最新リリース バージョン (マイナー バージョンを含む)
latestMajor	最新リリースの メジャー バージョン
8.0	C# 8.0

C# 8.0

- C#の最新を全部使えるのは、.NET Core 3 以降
と .NET Standard 2.1 以降
 - (.NET Framework では一部使用不可)

参考: .NET Framework と .NET Core

- .NET Core と .NET Framework
どちらを使えばよい？

参考: 今の .NET (2019年9月以降)



.NET Framework 4.8.X

- WPF
- Windows Forms
- ASP.NET

.NET Core 3.X

- WPF (Windows)
- Windows Forms (Windows)
- UWP (Windows)
- ASP.NET

Xamarin

- iOS
- Android
- Windows
- MacOS

.NET Standard Library

参考: 近未来の .NET (2020年11月予定)



.NET Framework 4.8.X

- WPF
- Windows Forms
- ASP.NET

保守フェーズに

.NET 5.0

- WPF (Windows)
- Windows Forms (Windows)
- UWP (Windows)
- ASP.NET

Xamarin

- iOS
- Android
- Windows
- MacOS

.NET Standard Library

C# 7~8の新機能の例



- 分解と Deconstruct **タプル**
- タプル (ValueTuple)
- ValueTask
- 参照戻り値
- In 引数
- readonly struct/ref
readonly
- Span **値型 (struct)**

- ローカル関数
- null 許容参照型
- 型 switch
- switch 式 **パターン マッチング**
- インターフェイスのデフォルト実装
- 非同期ストリーム/非同期イテレーター/非同期foreach

C# 7～8の新機能の例

- 値型 (struct) や参照型 (class) に関する
改良がたくさん



C#に潜むstructの罠？

- [C#に潜むstructの罠 – KAYAC engineers ' blog](#)
- 「お急ぎの方のために結論を申しあげますと、
structを使うなとなります。」



お急ぎの方のために
結論を申しあげますと、
そんなわけありません。

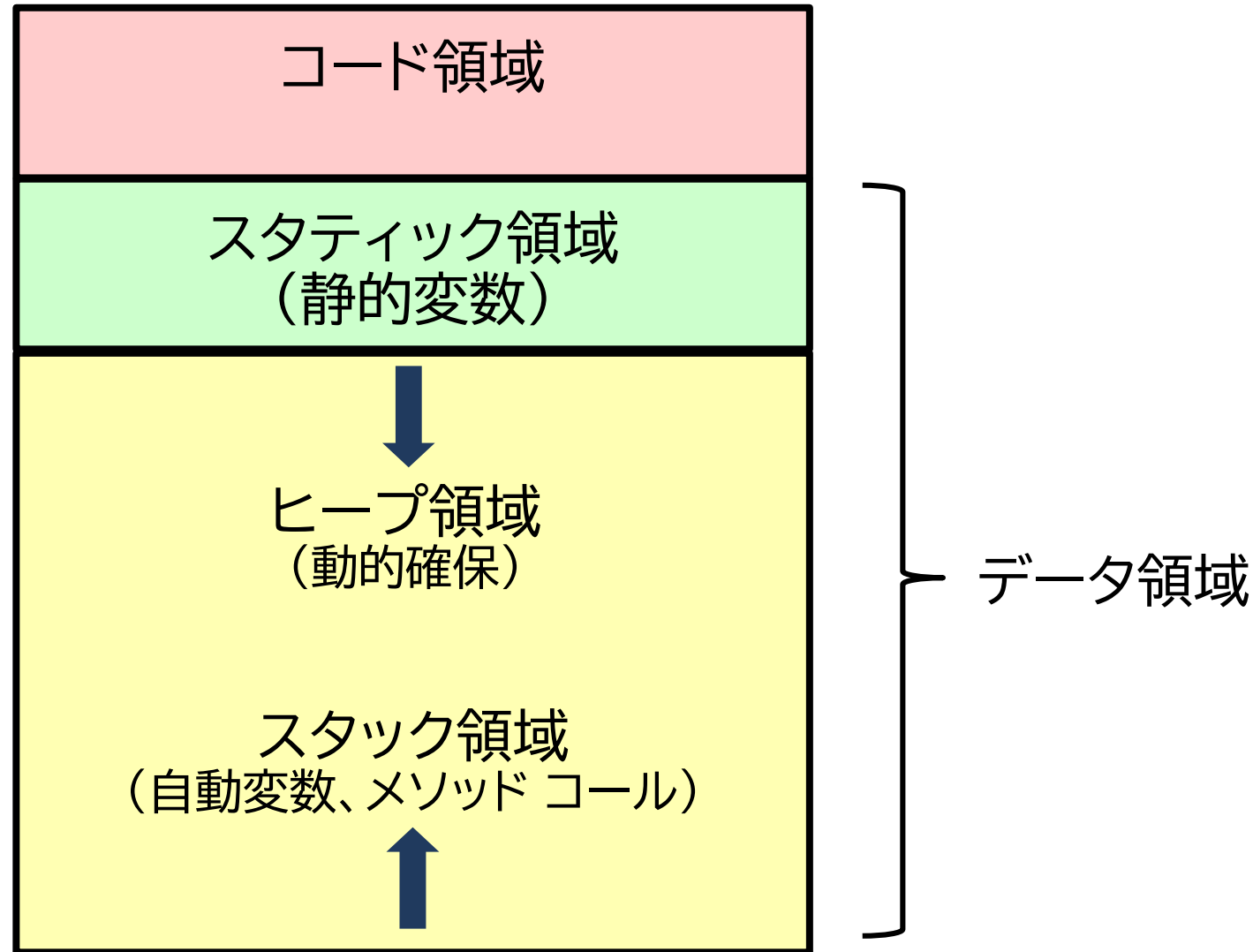
復習: structを使うときの注意点



コードを参考

- https://github.com/Fujiwo/CSharp7_8NewFeatures
 - https://github.com/Fujiwo/CSharp7_8NewFeatures/blob/master/struct%E3%82%92%E4%BD%BF%E3%81%86%E3%81%A8%E3%81%8D%E3%81%AE%E6%B3%A8%E6%84%8F%E7%82%B9/Program.cs

復習: スタック領域とヒープ領域



復習: スタック領域

```
int Add(int x, int y) // C++、x64、最適化なし
```

```
{
```

```
00007FF765AD1000  mov          dword ptr [rsp+10h],edx
```

```
00007FF765AD1004  mov          dword ptr [rsp+8],ecx
```

```
00007FF765AD1008  sub          rsp,18h
```

```
    auto answer = x + y;
```

```
00007FF765AD100C  mov          eax,dword ptr [y]
```

```
00007FF765AD1010  mov          ecx,dword ptr [x]
```

```
00007FF765AD1014  add          ecx,eax
```

```
00007FF765AD1016  mov          eax,ecx
```

```
00007FF765AD1018  mov          dword ptr [rsp],eax
```

```
    return answer;
```

```
00007FF765AD101B  mov          eax,dword ptr [rsp]
```

```
}
```

```
00007FF765AD101E  add          rsp,18h
```

```
00007FF765AD1022  ret
```


復習: スタック領域とヒープ領域

- スタック領域かヒープ領域かを意識する
- なるべくスタック領域を使う

参照型 (class)

- (必ず) ヒープ領域
- new および 暗黙の new がハイコスト
- コピーはローコスト (Shallow Copy)

値型 (struct)

- スタック領域にもおける（参照型のメンバーのときはヒープ領域）
 - （それ自体を）new するかどうかによらない
- いちいち動的確保しないのでローコスト
- コピーがハイコスト（16バイトくらいから）
 - 参照渡し/参照返しすればローコスト

値型 (struct) と参照型 (class) の例



- タプル
 - Tuple (System 名前空間) は参照型
 - ValueTuple (System 名前空間) は値型
- タスク
 - Task (System.Threading.Tasks 名前空間) は参照型
 - ValueTask (System.Threading.Tasks 名前空間) は値型
- 列挙型
 - Enum (System 名前空間) は参照型
 - enum は値型

値型 (struct) を使うコツ

- スタック領域をいたおす
- 値型を使ってもヒープ領域が使われちゃう場合に注意
- コピーをさける (16バイトを目途に)
 - 参照渡し・参照渡し
- なるべく immutable に使う

値型 (struct) を使ってもヒープ領域が使われてしまう例

- 参照型が new されてヒープ領域が使われてしまう例
 - ボックス化
 - ラムダ式とキャプチャー
 - yield return

暗黙の new

- object や interface への代入・初期化
- メソッドを引数で渡す
 - delegate が new される
- ラムダ式などでのメモリ空間のキャプチャー
 - => にカーソルを合わせて確認
- yield return
- async

値型 (struct) はなるべく immutable に使う

- readonly ref や readonly struct、
readonly メソッド
 - in からの Defensive Copy を防ぐ
- 参考: C++ の const T& 渡し/返し、const メンバー関数などと同じ

高速化

- 80:20の法則
- ボトルネックの解消を繰り返す
 - パフォーマンス プロファイラー
 - 余計な実行コードがないか ILSpy や LINQPad や SharpLab で確認

ツール: パフォーマンス プロファイラー

The screenshot displays the Visual Studio Performance Profiler interface. The top menu bar includes options like File (F), Edit (E), View (V), Project (P), Build (B), Debug (D), Test (S), Analyze (N), Tools (T), Extensions (X), Window (W), and Help (H). The toolbar shows various icons for file operations and debugging. The main window is titled "SpeedApp" and shows the "SpeedApp.Console" window. The "CPU Usage - Report2...1507.diagsession" is selected, and the "SpeedApp.dll" is loaded. The "SpeedApp.PlayerKojima::GetCommand" function is highlighted in the call stack. The call stack shows the following functions and their execution counts and percentages:

呼び出す関数	現在の関数	呼び出される関数
SpeedAppTest.Table::Game (389 (1.65%))	SpeedApp.PlayerKojima::GetCommand (390 (1.65%))	System.Collections.Generic.Dictionary'2+Enumerator[S... 87 (0.37%)
[ウォーク不可] (1 (0.00%))	関数本体 (266 (1.13%))	[外部コード] (30 (0.13%))
		System.Collections.Generic.Dictionary'2[System._Canon...7 (0.03%)

The code editor shows the implementation of the `GetCommand` function in `PlayerKojima.cs`. The code is as follows:

```
// var myOpenCard = playerOpenCard_NakamiwoKakikaetaraKorosu[this],
var myOpenCardCount = myOpenCard.Count;

foreach (var tableCard in tableCard_NakamiwoKakikaetaraKorosu) {
    for (var index = 0; index < myOpenCardCount; index++) {
        var cpuCard = myOpenCard[index];
        var difference = tableCard.Value.CardNumber - cpuCard.CardNumber;
        if (difference == 1 || difference == -1 || difference == 12 || difference == -12) {
            cpuPutCardCommand.PutCard = cpuCard;
            cpuPutCardCommand.PutPosition = tableCard.Key;
        }
    }
}

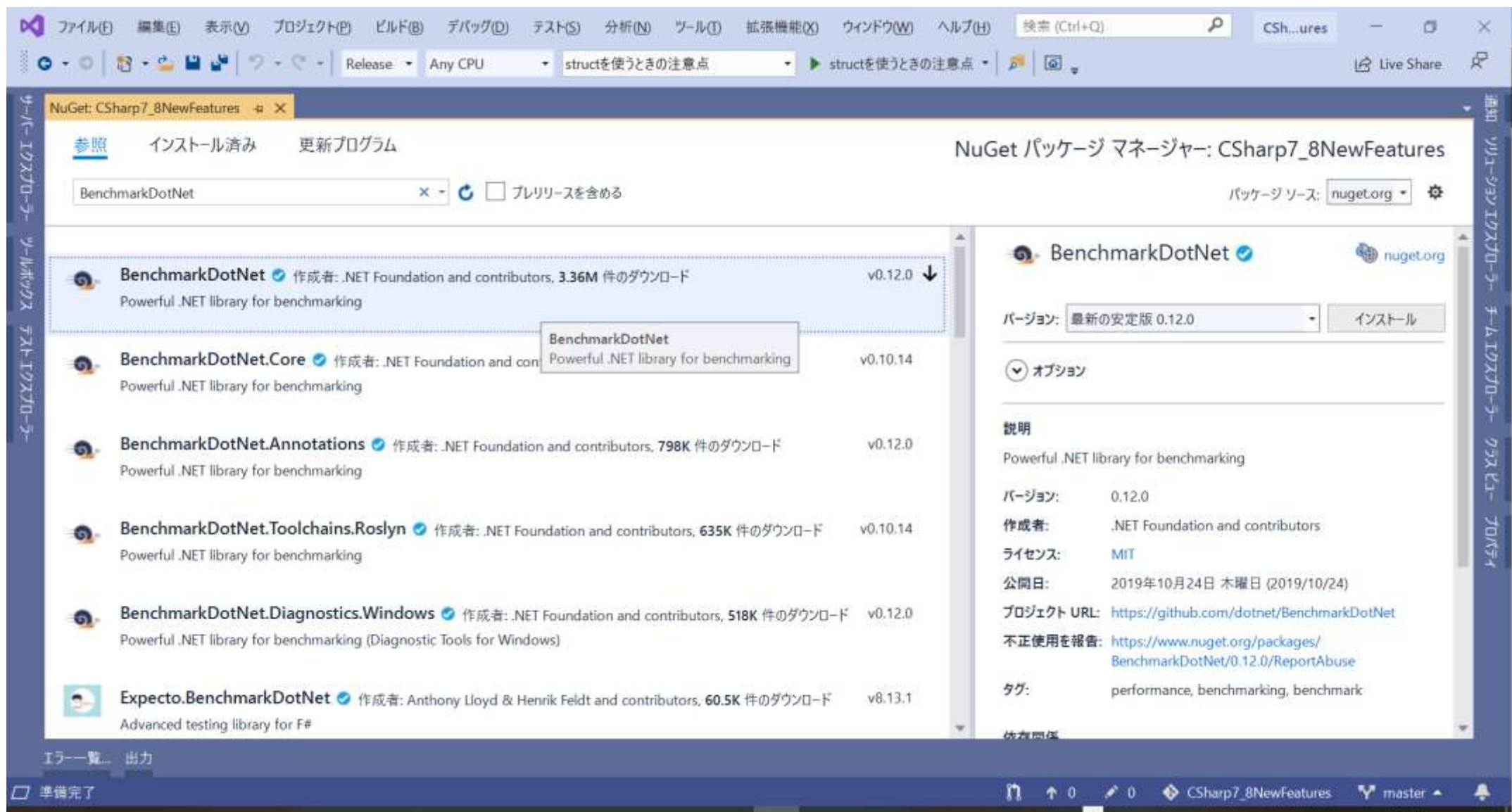
#if ATTACK_CPU_AND_MEMORY
    CpuMemoryAttacker.Start();
#endif // ATTACK_CPU_AND_MEMORY

return cpuPutCardCommand;
```

The status bar at the bottom indicates "52 %", "問題は見つかりませんでした", and "行: 255 文字: 26 SPC CRLF".

ツール: BenchmarkDotNet

NuGet



ツール: BenchmarkDotNet



```
using BenchmarkDotNet.Attributes;
using System.Linq;

class Program
{
    static void Main()
        => BenchmarkRunner.Run<SampleBenchMark>();
}

[ShortRunJob][HtmlExporter][CsvExporter]
public class SampleBenchMark
{
    const int count = 10000;
    int[] array = new int[0];

    [GlobalSetup]
    public void Setup()
        => array = Enumerable.Range(0, count).ToArray();
}
```

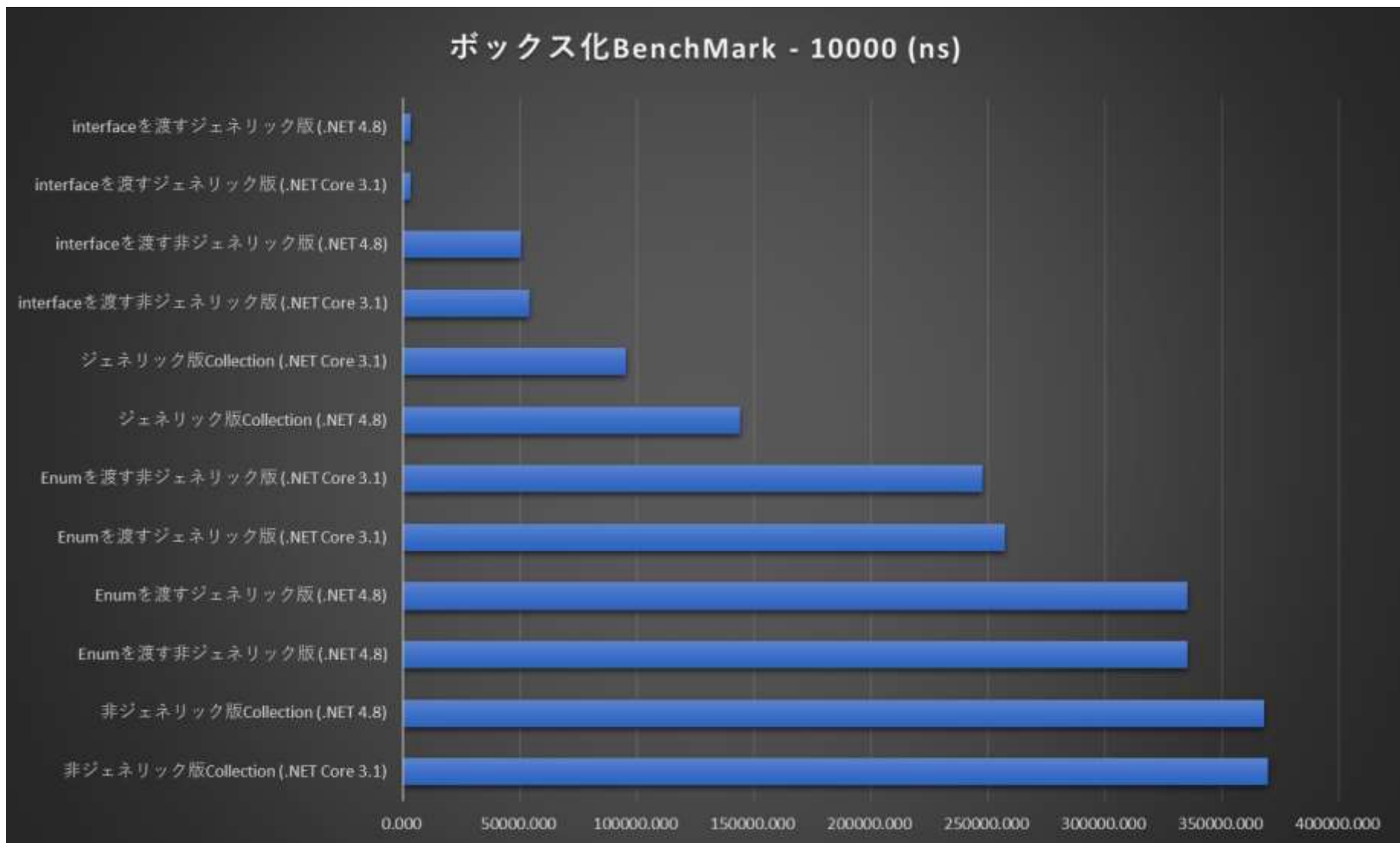
```
[Benchmark]
public int 配列をforして合計を求める()
{
    var sum = 0;
    for (var index = 0; index < array.Length; index++)
        sum += array[index];
    return sum;
}

[Benchmark]
public int 配列をforeachして合計を求める()
{
    var sum = 0;
    foreach (var element in array)
        sum += element;
    return sum;
}
}
```

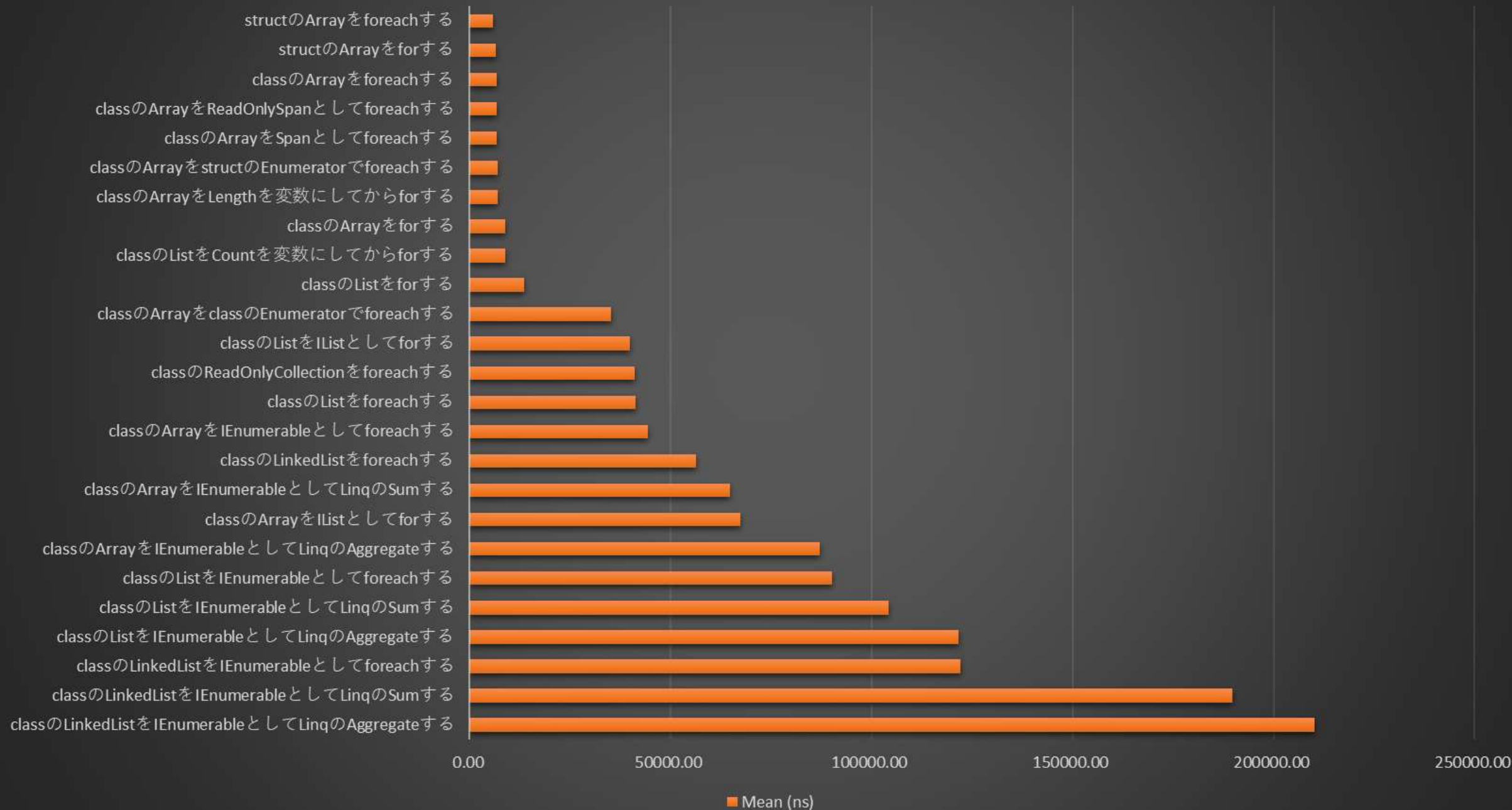
ツール: BenchmarkDotNet



- 測定結果1
- 測定結果2

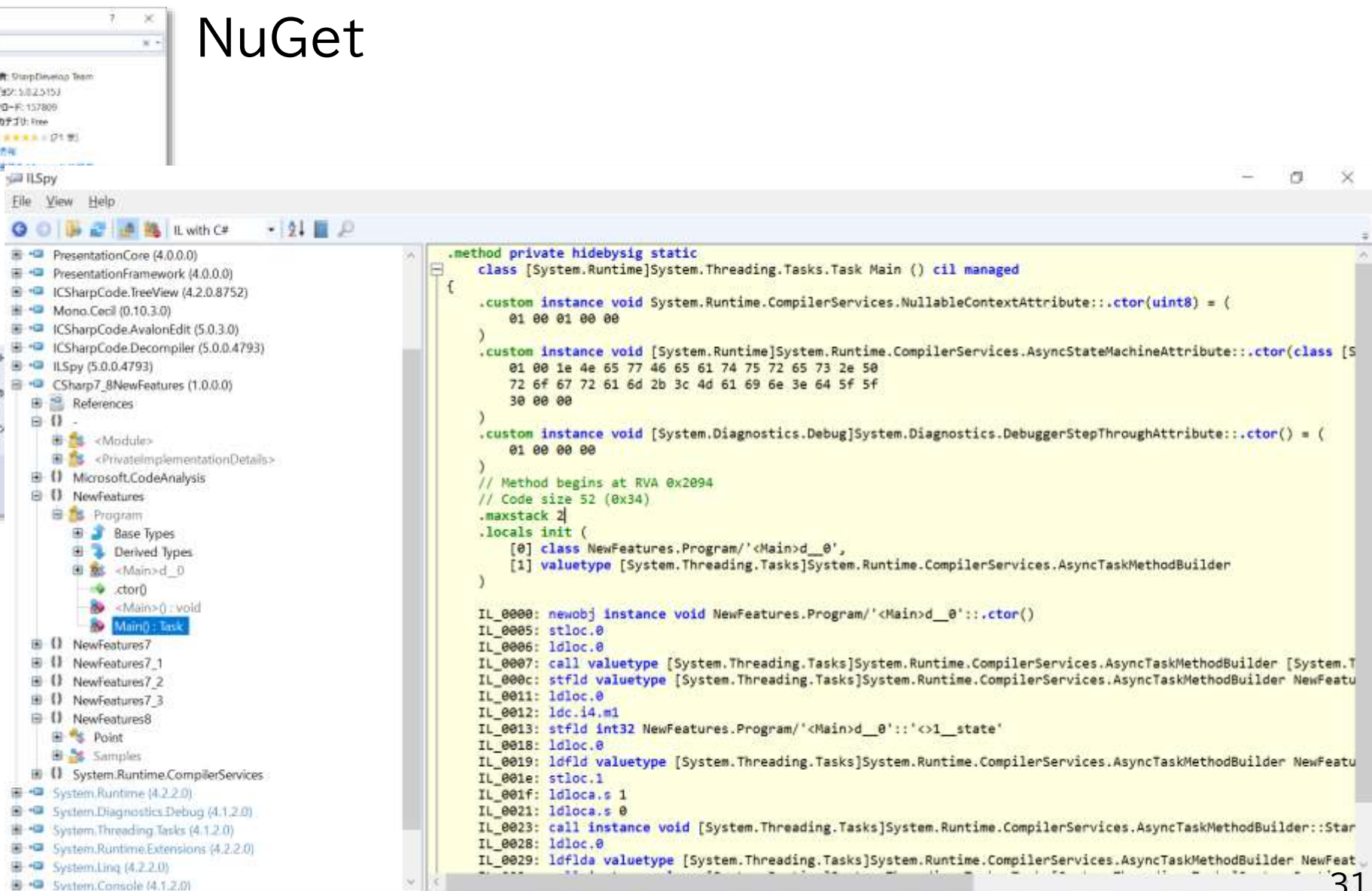
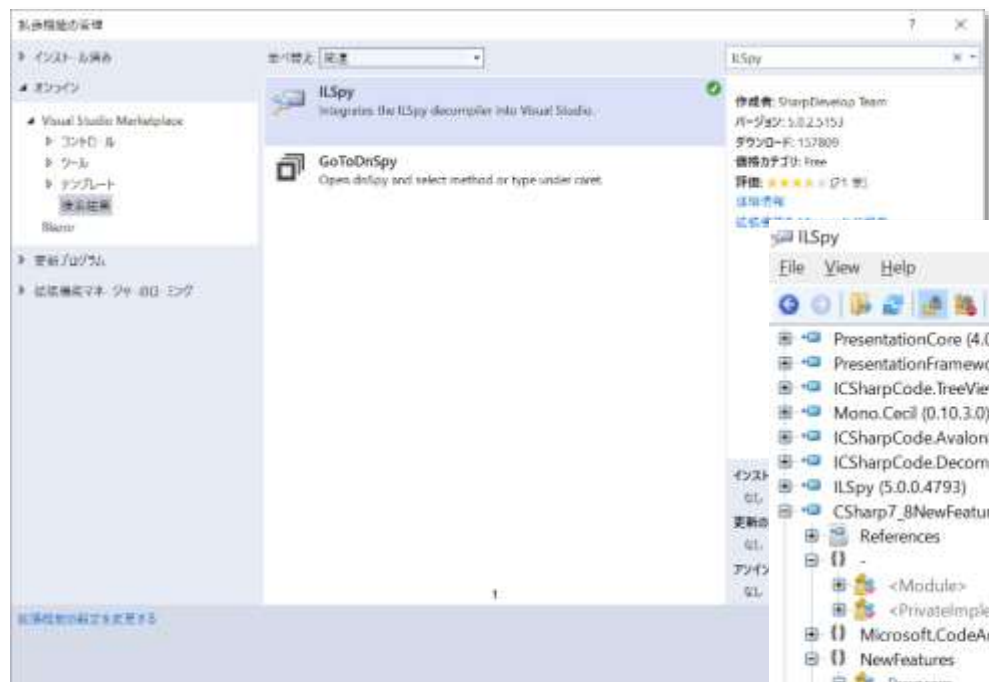


配列(10000要素)の順次アクセスの速度比較 (.NET Core)



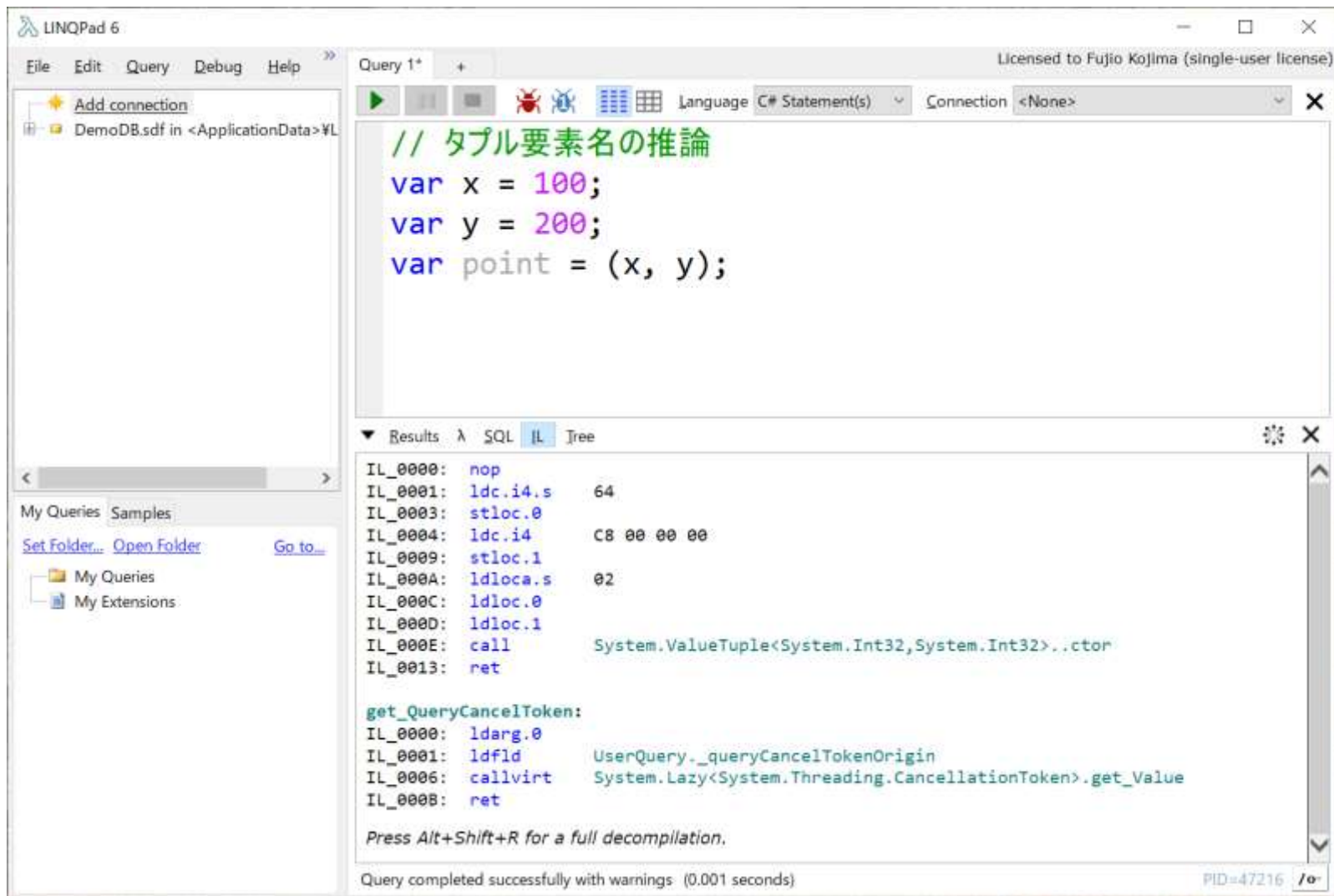
ツール: ILSpy

NuGet



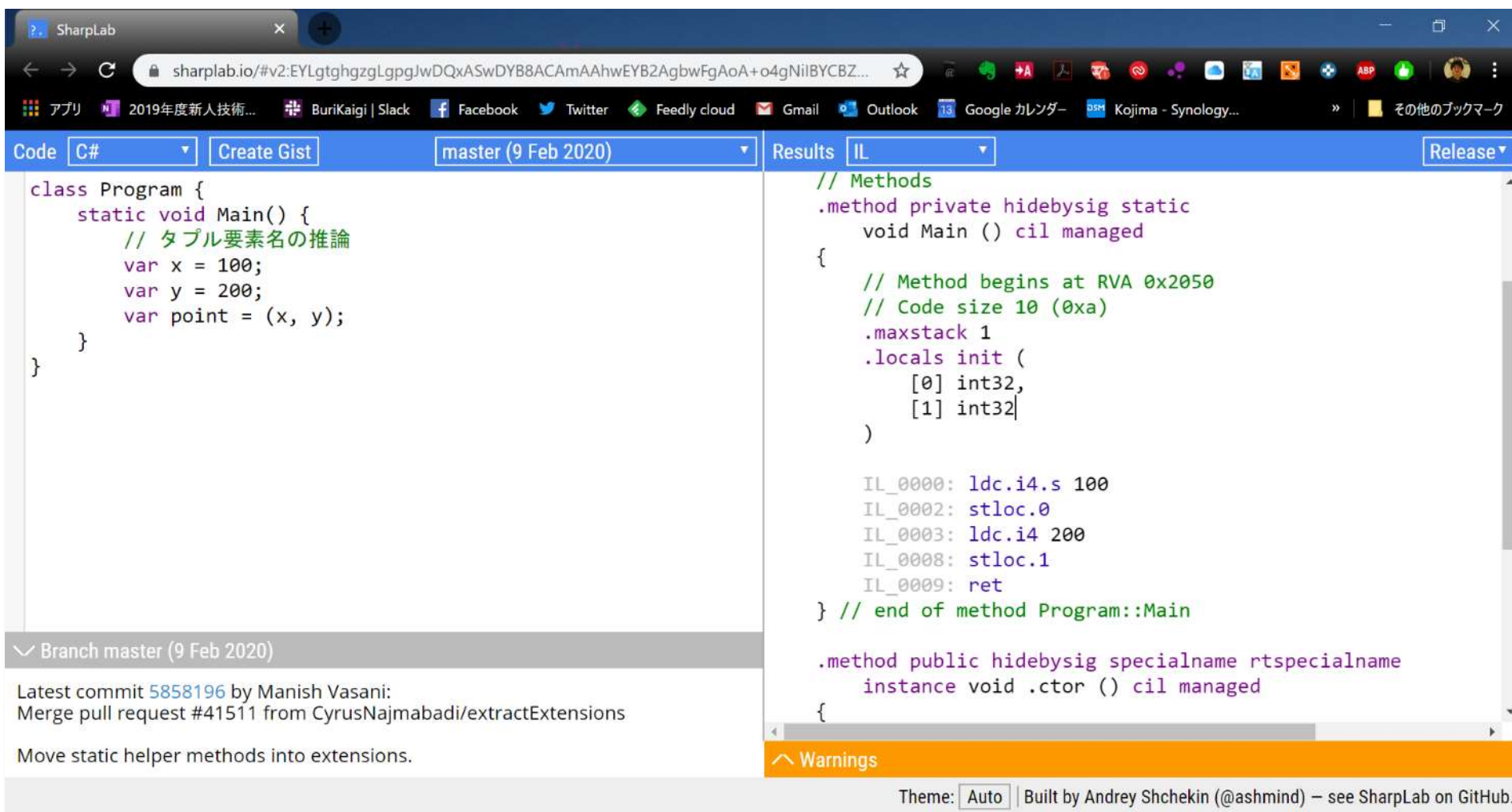
ツール: LINQPad

<https://www.linqpad.net>



ツール: SharpLab

- <https://sharplab.io>



The screenshot displays the SharpLab web application interface. The browser address bar shows the URL `sharplab.io/#v2:EYLgtghgzgJpggJwDQxASwDYB8ACAmAAhwEYB2AgbwFgAoA+o4gNilBYCBZ...`. The interface is divided into two main panels: 'Code' on the left and 'Results' on the right.

Code Panel (C#):

```
class Program {  
    static void Main() {  
        // タプル要素名の推論  
        var x = 100;  
        var y = 200;  
        var point = (x, y);  
    }  
}
```

Results Panel (IL):

```
// Methods  
.method private hidebysig static  
    void Main () cil managed  
{  
    // Method begins at RVA 0x2050  
    // Code size 10 (0xa)  
    .maxstack 1  
    .locals init (  
        [0] int32,  
        [1] int32  
    )  
  
    IL_0000: ldc.i4.s 100  
    IL_0002: stloc.0  
    IL_0003: ldc.i4 200  
    IL_0008: stloc.1  
    IL_0009: ret  
} // end of method Program::Main  
  
.method public hidebysig specialname rtspecialname  
    instance void .ctor () cil managed  
{  
}
```

Branch master (9 Feb 2020)

Latest commit [5858196](#) by Manish Vasani:
Merge pull request #41511 from CyrusNajmabadi/extractExtensions

Move static helper methods into extensions.

Warnings

Theme: [Auto](#) | Built by Andrey Shchekin (@ashmind) – see SharpLab on GitHub.

まとめ



参考文献

- [C# | Wikipedia](#)
- [C# の歴史 - C# ガイド | Microsoft Docs](#)
- [C# 7 の新機能 - C# によるプログラミング入門 | ++C++; // 未確認飛行 C](#)
- [C# 7.1 の新機能 - C# によるプログラミング入門 | ++C++; // 未確認飛行 C](#)
- [C# 7.2 の新機能 - C# によるプログラミング入門 | ++C++; // 未確認飛行 C](#)
- [C# 7.3 の新機能 - C# によるプログラミング入門 | ++C++; // 未確認飛行 C](#)
- [C# 8.0 の新機能 - C# によるプログラミング入門 | ++C++; // 未確認飛行 C](#)
- [今日からできる! 簡単 .NET 高速化 Tips | slideshare](#)
- [foreach の掛け方いろいろ | ++C++; // 未確認飛行 C ブログ](#)