Классы типов на C#

Меньше кода большей производительности

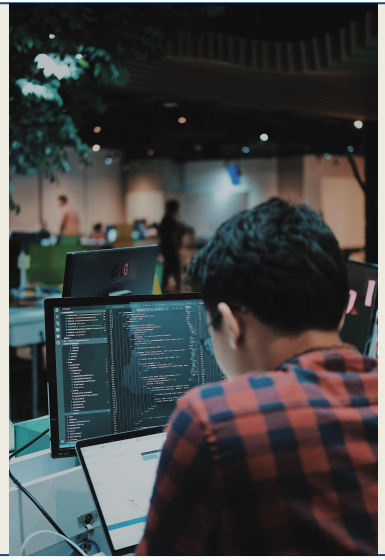
2020 Кирилл Маурин

План доклада

- **Дилемма**
- **Боль**
- **Идеал**
- **Решение**
- **Учения**
- **Бой**
- **Экспансия**
- **Эзотерика**

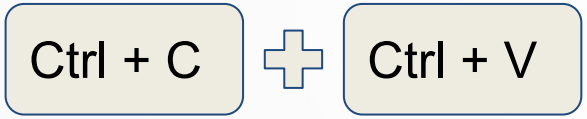
Дилемма

Сопровождаемость

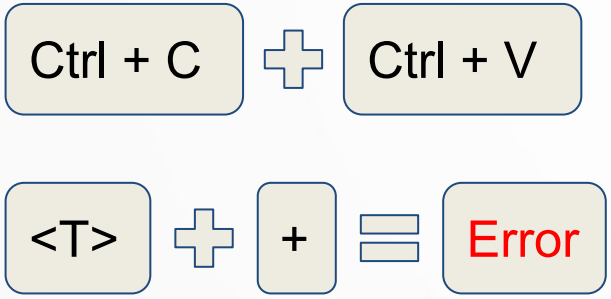


Производительность

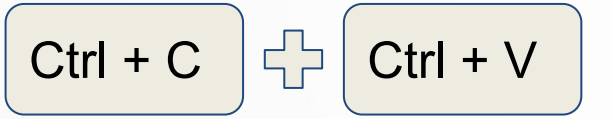
Боль

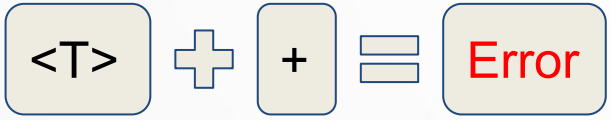


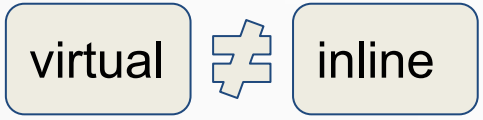
Боль



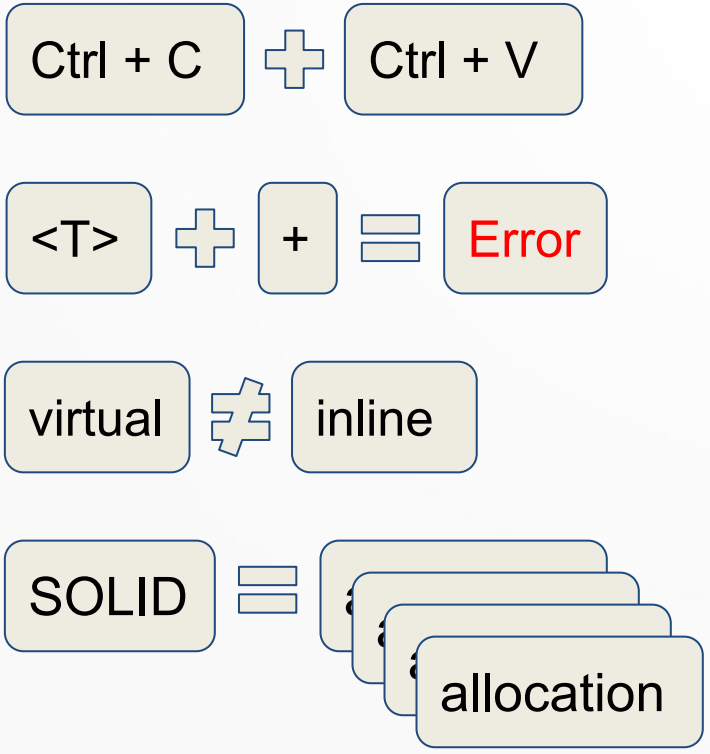
Боль







Боль



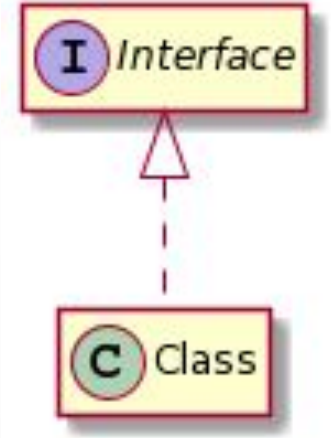
Боль

Ctrl + C + Ctrl + V

<T> + + = Error

virtual ≠ inline

SOLID = allocation



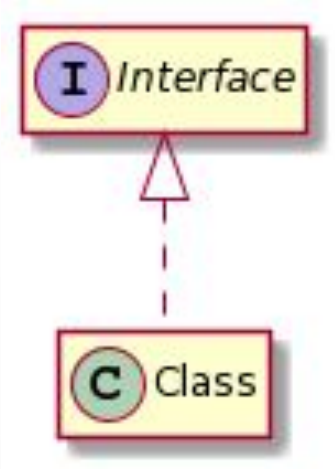
Боль

Ctrl + C + Ctrl + V

<T> + + = Error

virtual ≠ inline

SOLID = allocation



where + T = Error

На текущий момент проблем нет

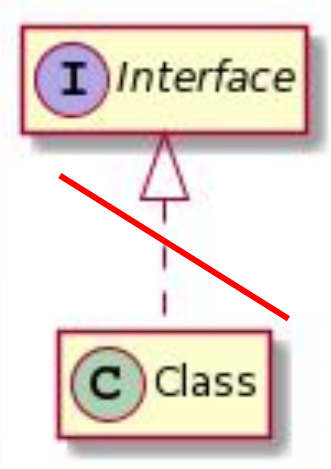
Идеал

Ctrl + C + Ctrl + V

<T> + + = Error

virtual ≠ inline

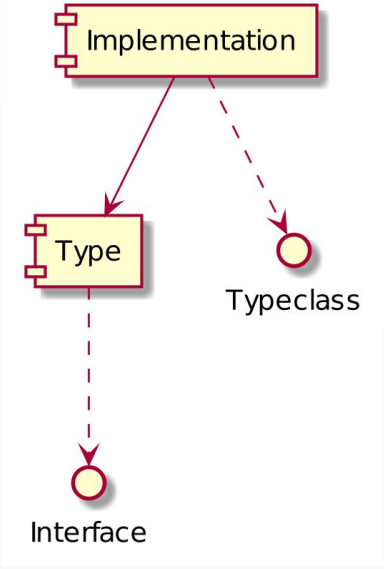
SOLID = allocation



where + T = Error

Решение

- Класс типов — набор операций, не привязанный к конкретному типу



Решение

- Класс типов — набор операций, не привязанный к конкретному типу
- Пример на Haskell:

```
class Listable a where -- Определение класса типов
  toList :: a -> [Int]
```

Классы типов

- Экземпляр класса типов — реализация операций класса типов для конкретного типа
- Пример на Haskell:

```
instance Listable (Tree Int) where -- Экземпляр класса типов
  toList Empty        = []
  toList (Node x l r) = toList l ++ [x] ++ toList
```

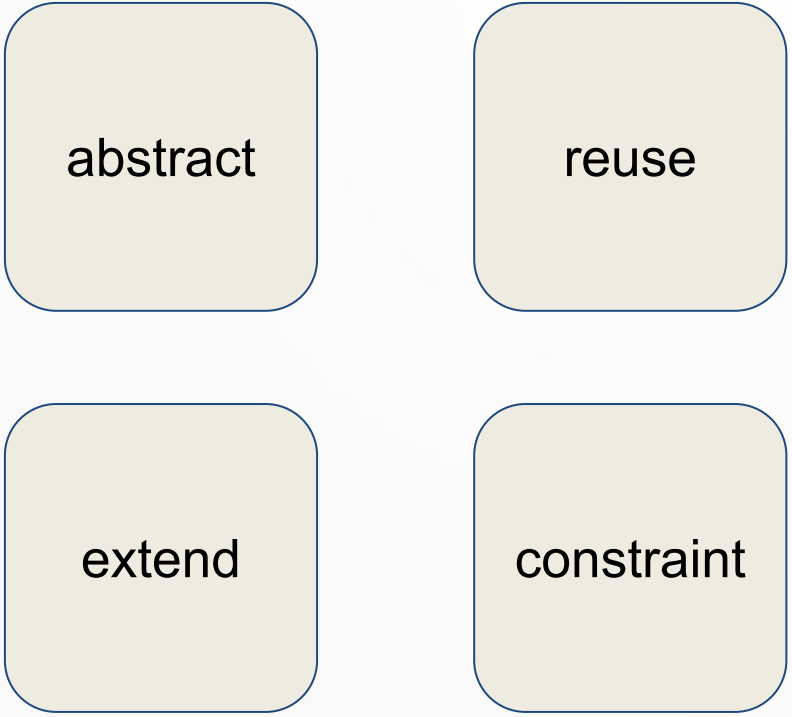
Классы типов

- Использование классов типов — просто вызов операций класса типов как обычных функций
- Пример на Haskell:

```
myTree :: Tree Int -- Создание значения типа
myTree = Node 1 (Node 2 Empty (Node 3 Empty Empty)) (Node 4 Empty Empty)

-- Использование класса типа
main = print (toList myTree)
```

Общее с интерфейсами



Различия

Класс типов	Интерфейс
Параметризуется типами	Реализуется типами
Категория типов	Тип
Статическая диспетчеризация	Динамическая диспетчеризация
Реализация отделена от типа	Реализация — часть типа

Классы типов есть в Haskell и Rust

В C# классов типов нет

В C# классов типов нет

Учения

- typeclass + C# = PROFIT
- ~~`var total = sum<TInt, FoldTuple<int>, Tuple<int, int, int>, int>(tup);`~~
- ~~performance penalty~~

https://github.com/louthy/language-ext

А на меньшее я не согласен!

```
{
    var result = initial;
    foreach (var i in enumerable)
        result += i;
    return result;
}
```

- ~~Copy & Paste~~
- TCollection<TItem>
- ~~virtual~~
- ~~allocation~~

Подсчет суммы 1000 элементов

```
_array.Sum();
```

LINQ

Это хорошо или плохо?

Method	Mean us	Error us	StdDev us	Ratio	Allocated B
LINQ	9.781	0.1939	0.2522	1.00	32

Дедовский способ

```
var sum = 0;
for (var i = 0; i < _array.Length; i++)
    sum += _array[i];
```

LINQ
for

В чем подвох?

Method	Mean us	Error us	StdDev us	Ratio	Allocated B
LINQ	9.781	0.1939	0.2522	1.00	32
for	1.087	0.0204	0.0191	0.11	-

Современное суммирование

- https://github.com/dotnet/corefx/blob/release/2.2/src/System.Linq/src/System/Linq/Sum.cs

```
public static int Sum(this IEnumerable<int> source)
{
    if (source == null)
        ThrowHelper.ThrowArgumentNullException(
            ExceptionArgument.source);
    int sum = 0;
    checked
    {
        foreach (int v in source)
            sum += v;
    }
    return sum;
}
```

Декомпиляция

- https://github.com/dotnet/corefx/blob/release/2.2/src/System.Linq/src/System/Linq/Sum.cs

```
public static int Sum(this IEnumerable<int> source)
{
    if (source == null)
        ThrowHelper.ThrowArgumentNullException(
            ExceptionArgument.source);
    int sum = 0;
    checked
    {
        foreach (int v in source)
            sum += v;
    }
    return sum;
}
```

```
IEnumerator<int> e =
source.GetEnumerator();
try
{
    while (e.MoveNext())
        sum = checked(sum + e.Current);
    return sum;
}
finally
{
    if (e != null)
        e.Dispose();
}
```

Истинная сущность foreach

```
IEnumerator<int> e = source.GetEnumerator(); // Аллокация
try
{
   while (e.MoveNext())
       sum = checked(sum + e.Current);
   return sum;
}
finally
{
   if (e != null)
       e.Dispose();
}
```

Истинная сущность foreach

```
IEnumerator<int> e = source.GetEnumerator(); // Аллокация
try
{
   while (e.MoveNext())
       sum = checked(sum + e.Current); // Проверка переполнения
   return sum;
}
finally
{
   if (e != null)
       e.Dispose();
}
```

Истинная сущность foreach

```
IEnumerator<int> e = source.GetEnumerator(); // Аллокация Виртуальный вызов
try
{
    while (e.MoveNext()) // Виртуальный вызов
        sum = checked(sum + e.Current); // Проверка переполнения Виртуальный вызов
    return sum;
}
finally
{
    if (e != null)
        e.Dispose(); // Виртуальный вызов
}
```

Истинная сущность foreach

```
IEnumerator<int> e = source.GetEnumerator(); // Аллокация Виртуальный вызов
try
{
    while (e.MoveNext()) // Виртуальный вызов
        sum = checked(sum + e.Current); // Проверка переполнения Виртуальный вызов
    return sum;
}
finally // Блок финализации
{
    if (e != null)
        e.Dispose(); // Виртуальный вызов
}
```

Цена модернизации

Просто for

```
L0000: xor eax, eax
L0002: xor edx, edx
L0004: mov r8d, [rcx+0x8]
L0008: test r8d, r8d
L000b: jle L001c
L000d: movsxd r9, edx
L0010: add eax,[rcx+r9*4+0x10]
L0015: inc edx
L0017: cmp r8d, edx
L001a: jg L000d
L001c: ret
```

LINQ

```
L0000: push rbp
L0001: push rsi
L0002: sub rsp, 0x38
L0006: lea rbp, [rsp+0x40]
L000b: mov [rbp-0x20], rsp
L000f: xor esi, esi
L0011: mov r11, 0x7ffe711d7020
L001b: cmp [rcx], ecx
L001d: call qword [rip+0x5a1d]
L0023: mov [rbp-0x10], rax
L0027: mov rcx, [rbp-0x10]
L002b: mov r11, 0x7ffe711d7028
L0035: cmp [rcx], ecx
L0037: call qword [rip+0x5a0b]
L003d: test eax, eax
L003f: jz L007d
L0041: mov rcx, [rbp-0x10]
L0045: mov r11, 0x7ffe711d7030
L004f: cmp [rcx], ecx
L0051: call qword [rip+0x59f9]
L0057: add esi, eax
L0059: jo L0077
L005b: mov rcx, [rbp-0x10]
L005f: mov r11, 0x7ffe711d7028
L0069: cmp [rcx], ecx
L006b: call qword [rip+0x59d7]
L0071: test eax, eax
L0073: jnz L0041
L0075: jmp L007d
L0077: call 0x7ffec7fe1c50
L007c: int3
```

```
L007d: mov rcx, [rbp-0x10]
L0081: mov r11, 0x7ffe711d7038
L008b: cmp [rcx], ecx
L008d: call qword [rip+0x59c5]
L0093: mov eax, esi
L0095: lea rsp, [rbp-0x8]
L0099: pop rsi
L009a: pop rbp
L009b: ret
L009c: push rbp
L009d: push rsi
L009e: sub rsp, 0x28
L00a2: mov rbp, [rcx+0x20]
L00a6: mov [rsp+0x20], rbp
L00ab: lea rbp, [rbp+0x40]
L00af: cmp qword [rbp-0x10], 0x0
L00b4: jz L00cc
L00b6: mov rcx, [rbp-0x10]
L00ba: mov r11, 0x7ffe711d7038
L00c4: cmp [rcx], ecx
L00c6: call qword [rip+0x598c]
L00cc: nop
L00cd: add rsp, 0x28
L00d1: pop rsi
L00d2: pop rbp
L00d3: ret
```

А почему только для одного типа?

```
public static T Sum<T>(this T[] array, T initial)
{
    var result = initial;
    for (var i = 0; i < array.Length; i++)
        result += array[i];
    return result;
}
```

LINQ for generic

Столкновение с реальностью

- Error CS0019: Operator '+=' cannot be applied to operands of type 'T' and 'T'

```
public static T Sum<T>(this T[] array, T initial)
{
    var result = initial;
    for (var i = 0; i < array.Length; i++)
        result += array[i];
    return result;
}
```

А если сложение определено?

```
public static T Sum<T>(this T[] array, T initial)
    where T : +
{
    var result = initial;
    for (var i = 0; i < array.Length; i++)
        result += array[i];
    return result;
}
```

Так тоже не работает

- Error CS1031: Type expected

```
public static T Sum<T>(this T[] array, T initial)
    where T : +
{
    var result = initial;
    for (var i = 0; i < array.Length; i++)
        result += array[i];
    return result;
}
```

Обобщенное суммирование

```
public static T Sum<T>
    (this T[] array, T initial, ISummator<T> summator)
{
    var result = initial;
    for (var i = 0; i < array.Length; i++)
        result = summator.Add(result, array[i]);
    return result;
}
```

LINQ for generic

Сложение как интерфейс

```
public interface ISummator<T>
{
    T Add(T left, T right);
}
```

Сумматор как реализация

```
public class IntSummator : ISummator<int>
{
    public int Add(int left, int right) => left + right;
}
```

Обобщенное суммирование

- Использование: `_array.Sum(0, new IntSummator());`

```
public static T Sum<T>
    (this T[] array, T initial, ISummator<T> summator)
{
    var result = initial;
    for (var i = 0; i < array.Length; i++)
        result = summator.Add(result, array[i]);
    return result;
}
```

Насколько это быстро?

Method	Mean us	Error us	StdDev us	Ratio	Allocated B
LINQ	9.781	0.1939	0.2522	1.00	32
for	1.087	0.0204	0.0191	0.11	-
generic	5.915	0.1158	0.1334	0.61	24

Цена обобщения

```
public static T Sum<T>
    (this T[] array, T initial, ISummator<T> summator)
{
    var result = initial;
    for (var i = 0; i < array.Length; i++)
        result = summator.Add(result, array[i]); // Виртуальный вызов
    return result;
}
```

Цена обобщения

Просто for

```
L0000: xor eax, eax
L0002: xor edx, edx
L0004: mov r8d, [rcx+0x8]
L0008: test r8d, r8d
L000b: jle L001c
L000d: movsxd r9, edx
L0010: add eax,[rcx+r9*4+0x10]
L0015: inc edx
L0017: cmp r8d, edx
L001a: jg L000d
L001c: ret
```

С выделенным сумматором

```
L0000: push rdi
L0001: push rsi
L0002: push rbp
L0003: push rbx
L0004: sub rsp, 0x28
L0008: mov rsi, rcx
L000b: mov rdi, r8
L000e: xor ebx, ebx
L0010: mov ebp, [rsi+0x8]
L0013: test ebp, ebp
L0015: jle L003c
L0017: movsxd r8, ebx
L001a: mov r8d, [rsi+r8*4+0x10]
L001f: mov rcx, rdi
L0022: mov r11, 0x7ffe71217060
L002c: cmp [rcx], ecx
L002e: call qword [rip+0x584c]
```

```
L0034: mov edx, eax
L0036: inc ebx
L0038: cmp ebp, ebx
L003a: jg L0017
L003c: mov eax, edx
L003e: add rsp, 0x28
L0042: pop rbx
L0043: pop rbp
L0044: pop rsi
L0045: pop rdi
L0046: ret
```

Немного магии

```
public static T Sum<T, TSummator>
    (this T[] array, T initial, TSummator summator)
    where TSummator : struct, ISummator<T>
{
    var result = initial;
    for (var i = 0; i < array.Length; i++)
        result = summator.Add(result, array[i]);
    return result;
}
```

LINQ for generic magic

Что это было?

Method	Mean us	Error us	StdDev us	Ratio	Allocated B
LINQ	9.781	0.1939	0.2522	1.00	32
for	1.087	0.0204	0.0191	0.11	-
generic	5.915	0.1158	0.1334	0.61	24
magic	0.921	0.0017	0.0016	0.09	-

Цена обобщения — ноль?

Просто for

```
L0000: xor eax, eax
L0002: xor edx, edx
L0004: mov r8d, [rcx+0x8]
L0008: test r8d, r8d
L000b: jle L001c
L000d: movsxd r9, edx
L0010: add eax,[rcx+r9*4+0x10]
L0015: inc edx
L0017: cmp r8d, edx
L001a: jg L000d
L001c: ret
```

С магическим сумматором

```
L0000: mov eax, edx
L0002: xor edx, edx
L0004: mov r8d, [rcx+0x8]
L0008: test r8d, r8d
L000b: jle L001f
L000d: movsxd r9, edx
L0010: mov r9d, [rcx+r9*4+0x10]
L0015: add eax, r9d
L0018: inc edx
L001a: cmp r8d, edx
L001d: jg L000d
L001f: ret
```

Сумматор - структура

```
public readonly struct IntSummator : ISummator<int>
{
    public int Add(int left, int right) => left + right;
}
```

Сумматор — параметр типа

```
public static T Sum<T, TSummator> // Параметр типа
    (this T[] array, T initial, TSummator summator)
    where TSummator : struct, ISummator<T>
{
    var result = initial;
    for (var i = 0; i < array.Length; i++)
        result = summator.Add(result, array[i]);
    return result;
}
```

Сумматор — параметр типа

```
public static T Sum<T, TSummator> // Параметр типа
    (this T[] array, T initial, TSummator summator)
    where TSummator : struct, ISummator<T> // Структура
{
    var result = initial;
    for (var i = 0; i < array.Length; i++)
        result = summator.Add(result, array[i]);
    return result;
}
```

Сумматор — параметр типа

```
public static T Sum<T, TSummator> // Параметр типа
    (this T[] array, T initial, TSummator summator)
    where TSummator : struct, ISummator<T> // Структура
{
    var result = initial;
    for (var i = 0; i < array.Length; i++)
        result = summator.Add(result, array[i]); // Inline
    return result;
}
```

Параметры обобщений в .NET

Ссылки	Значения
Один размер	Свой размер у каждого типа
Один общий код	Свой код под каждый тип
Виртуальные вызовы	Статические вызовы
Почти как в Java	Уникально для .NET

Сумматор — параметр типа

```
public static T Sum<T, TSummator> // Параметр типа
    (this T[] array, T initial, TSummator summator)
    where TSummator : struct, ISummator<T> // Структура
{
    var result = initial;
    for (var i = 0; i < array.Length; i++)
        result = summator.Add(result, array[i]); // Inline
    return result;
}
```

А нужен ли аргумент-сумматор?

- Использование: `_array.Sum<int, IntSummator>(0);`

```
public static T Sum<T, TSummator>
    (this T[] array, T initial, TSummator summator = default)
    where TSummator : struct, ISummator<T>
{
    var result = initial;
    for (var i = 0; i < array.Length; i++)
        result = summator.Add(result, array[i]);
    return result;
}
```

LINQ
for
generic
magic
default

А нужен ли аргумент-сумматор?

- Использование: `_array.Sum<int, IntSummator>(0);`

```
public static T Sum<T, TSummator>
    (this T[] array, T initial, TSummator summator = default) // Не нужен?
    where TSummator : struct, ISummator<T>
{
    var result = initial;
    for (var i = 0; i < array.Length; i++)
        result = summator.Add(result, array[i]);
    return result;
}
```

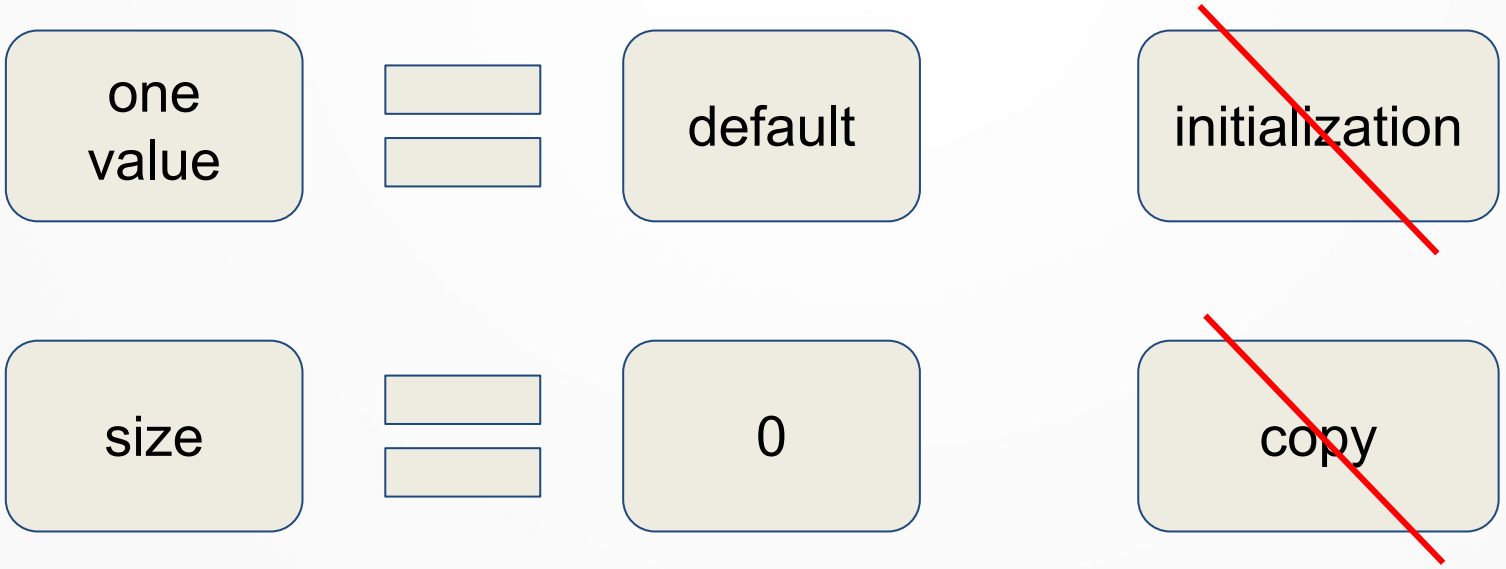
И так сойдет!

Method	Mean us	Error us	StdDev us	Ratio	Allocated B
LINQ	9.781	0.1939	0.2522	1.00	32
for	1.087	0.0204	0.0191	0.11	-
generic	5.915	0.1158	0.1334	0.61	24
magic	0.921	0.0017	0.0016	0.09	-
default	0.903	0.0017	0.0018	0.09	-

Сумматор — структура без полей

```
public struct IntSummator : ISummator<int>
{
    public int Add(int left, int right)
        => left + right;
}
```

Структура без полей



А что, так можно было?

- Использование:

```
_array.Sum(0.AsAdditive());
```

```
public static T Sum<T, TSummator>
    (this T[] array, Additive<T, TSummator> initial)
    where TSummator : struct, ISummator<T>
{
    var result = initial;
    for (var i = 0; i < array.Length; i++)
        result += array[i];
    return result;
}
```

LINQ for generic magic operator

А что, так можно было?

- Использование без явного указания типов-параметров:

```
_array.Sum<int, TSummator>(0.AsAdditive());

public static T Sum<T, TSummator>
    (this T[] array, Additive<T, TSummator> initial, TSummator summator =
default)
    where TSummator : struct, ISummator<T>
{
    var result = initial;
    for (var i = 0; i < array.Length; i++)
        result += array[i];
    return result;
}
```

А что, так можно было?

- Использование без явного указания типов-параметров:

```
_array.Sum<int, TSummator>(0.AsAdditive()); // Странный вызов

public static T Sum<T, TSummator>
    (this T[] array, Additive<T, TSummator> initial, TSummator summator =
default) // Странный тип
    where TSummator : struct, ISummator<T>
{
    var result = initial;
    for (var i = 0; i < array.Length; i++)
        result += array[i];
    return result;
}
```

А что, так можно было?

- Использование без явного указания типов-параметров:

```
_array.Sum<int, TSummator>(0.AsAdditive()); // Странный вызов

public static T Sum<T, TSummator>
    (this T[] array, Additive<T, TSummator> initial, TSummator summator =
default) // Странный тип
    where TSummator : struct, ISummator<T>
{
    var result = initial;
    for (var i = 0; i < array.Length; i++)
        result += array[i]; // Операция в обобщенном методе
    return result;
}
```

Как будто что-то плохое

```
public readonly ref struct Additive<T, TSummator>
    where TSummator : struct, ISummator<T>
{
    Additive(T value) => Value = value;

    public T Value { get; }

    public static Additive<T, TSummator> operator +
        (Additive<T, TSummator> left, Additive<T, TSummator> right)
        return default(TSummator).Add(left, right);

    public static implicit operator Additive<T, TSummator>(T value)
        => new Additive<T, TSummator>(value);
    public static implicit operator T(Additive<T, TSummator> value) => value.Value;
}
```

Как будто что-то плохое

```
public readonly ref struct Additive<T, TSummator>
   where TSummator : struct, ISummator<T>
{
   Additive(T value) => Value = value;

   public T Value { get; }

   public static Additive<T, TSummator> operator + // Операция для обобщенного
типа
        (Additive<T, TSummator> left, Additive<T, TSummator> right)
        return default(TSummator).Add(left, right);

   public static implicit operator Additive<T, TSummator>(T value)
       => new Additive<T, TSummator>(value);
   public static implicit operator T(Additive<T, TSummator> value) => value.Value;
}
```

Как будто что-то плохое

```
public readonly ref struct Additive<T, TSummator>
    where TSummator : struct, ISummator<T>
{
    Additive(T value) => Value = value;

    public T Value { get; } // Единственное поле для типа-обертки

    public static Additive<T, TSummator> operator + // Операция для обобщенного
типа
        (Additive<T, TSummator> left, Additive<T, TSummator> right)
        return default(TSummator).Add(left, right);

    public static implicit operator Additive<T, TSummator>(T value)
        => new Additive<T, TSummator>(value);
    public static implicit operator T(Additive<T, TSummator> value) => value.Value;
}
```

Как будто что-то плохое

```
public readonly ref struct Additive<T, TSummator>
   where TSummator : struct, ISummator<T>
{
   Additive(T value) => Value = value;

   public T Value { get; } // Единственное поле для типа-обертки

   public static Additive<T, TSummator> operator + // Операция для обобщенного
типа
        (Additive<T, TSummator> left, Additive<T, TSummator> right)
        return default(TSummator).Add(left, right);

   public static implicit operator Additive<T, TSummator>(T value)
       => new Additive<T, TSummator>(value); // Прозрачное заворачивание
   public static implicit operator T(Additive<T, TSummator> value) => value.Value;
}
```

Вишенка на торте

Вишенка на торте

```
public static Additive<int, IntSummator>
    AsAdditive(this int value)
    => value;
```

Синтаксический диабет

С магическим сумматором

```
L0000: mov eax, edx
L0002: xor edx, edx
L0004: mov r8d, [rcx+0x8]
L0008: test r8d, r8d
L000b: jle L001f
L000d: movsxd r9, edx
L0010: mov r9d, [rcx+r9*4+0x10]
L0015: add eax, r9d
L0018: inc edx
L001a: cmp r8d, edx
L001d: jg L000d
L001f: ret
```

С красивым сложением

```
L0000: push rax
L0001: xor eax, eax
L0003: mov [rsp], rax
L0007: mov eax, edx
L0009: xor edx, edx
L000b: mov r8d, [rcx+0x8]
L000f: test r8d, r8d
L0012: jle L0032
L0014: movsxd r9, edx
L0017: mov r9d, [rcx+r9*4+0x10]
L001c: mov byte [rsp], 0x0
L0020: lea r10, [rsp]
L0024: mov byte [r10], 0x0
L0028: add eax, r9d
L002b: inc edx
L002d: cmp r8d, edx
L0030: jg L0014
L0032: add rsp, 0x8
L0036: ret
```

Гомеопатия

```
public readonly ref struct Additive<T, TSummator> where TSummator : struct, ISummator<T>
{
    Additive(T value) => Value = value;

    public T Value { get; }

    public static Additive<T, TSummator> operator +
        (Additive<T, TSummator> left, Additive<T, TSummator> right)
    {
        var summator = default(TSummator);
        return summator.Add(left, right);
    }

    public static implicit operator Additive<T, TSummator>(T value)
        => new Additive<T, TSummator>(value);
    public static implicit operator T(Additive<T, TSummator> value) => value.Value;
}
```

LINQ
for
generic
magic
operator
homeopathy

Гомеопатия

```
public readonly ref struct Additive<T, TSummator> where TSummator : struct, ISummator<T>
{
    Additive(T value) => Value = value;

    public T Value { get; }

    public static Additive<T, TSummator> operator +
        (Additive<T, TSummator> left, Additive<T, TSummator> right)
    {
        var summator = default(TSummator); // Ничего не меняющий лишний код
        return default(TSummator)summator.Add(left, right);
    }

    public static implicit operator Additive<T, TSummator>(T value)
        => new Additive<T, TSummator>(value);
    public static implicit operator T(Additive<T, TSummator> value) => value.Value;
}
```

Я не знаю как это работает

С магическим сумматором

```
L0000: mov eax, edx
L0002: xor edx, edx
L0004: mov r8d, [rcx+0x8]
L0008: test r8d, r8d
L000b: jle L001f
L000d: movsxd r9, edx
L0010: mov r9d, [rcx+r9*4+0x10]
L0015: add eax, r9d
L0018: inc edx
L001a: cmp r8d, edx
L001d: jg L000d
L001f: ret
```

С гомеопатическим сумматором

```
L0000: mov eax, edx
L0002: xor edx, edx
L0004: mov r8d, [rcx+0x8]
L0008: test r8d, r8d
L000b: jle L001f
L000d: movsxd r9, edx
L0010: mov r9d, [rcx+r9*4+0x10]
L0015: add eax, r9d
L0018: inc edx
L001a: cmp r8d, edx
L001d: jg L000d
L001f: ret
```

Класс типов на C#


```
public interface ISummator<T>
{
    T Add(T left, T right);
}
```

```
public readonly ref struct Additive<T, TSummator>
    where TSummator : struct, ISummator<T>
{
    Additive(T value) => Value = value;

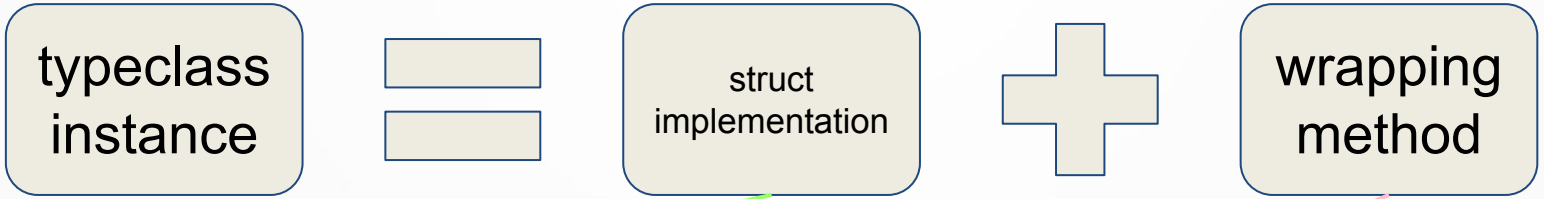
    public T Value { get; }

    public static Additive<T, TSummator> operator +
        (Additive<T, TSummator> left, Additive<T, TSummator> rig
        return default(TSummator).Add(left, right);

    public static implicit operator Additive<T, TSummator>(T val
        => new Additive<T, TSummator>(value);
    public static implicit operator T(Additive<T, TSummator> value) => value.Value;
}
```

LINQ
for
generic
magic
operator
homeopathy
typeclass

Экземпляр класса типов на C#

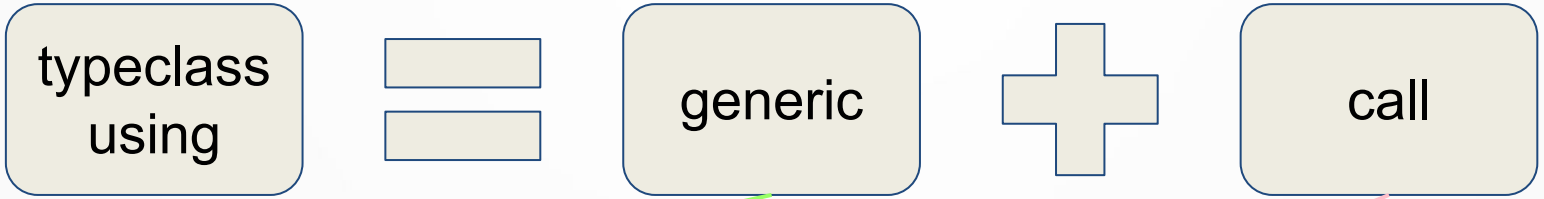



```
public readonly struct IntSummator :
    ISummator<int>
{
    public int Add(int left, int right)
        => left + right;
}
```

```
public static Additive<int, IntSummator>
    AsAdditive(this int value)
    => value;
```

interface
struct
instance
wrapping
generic
call

Применение классов типов на C#

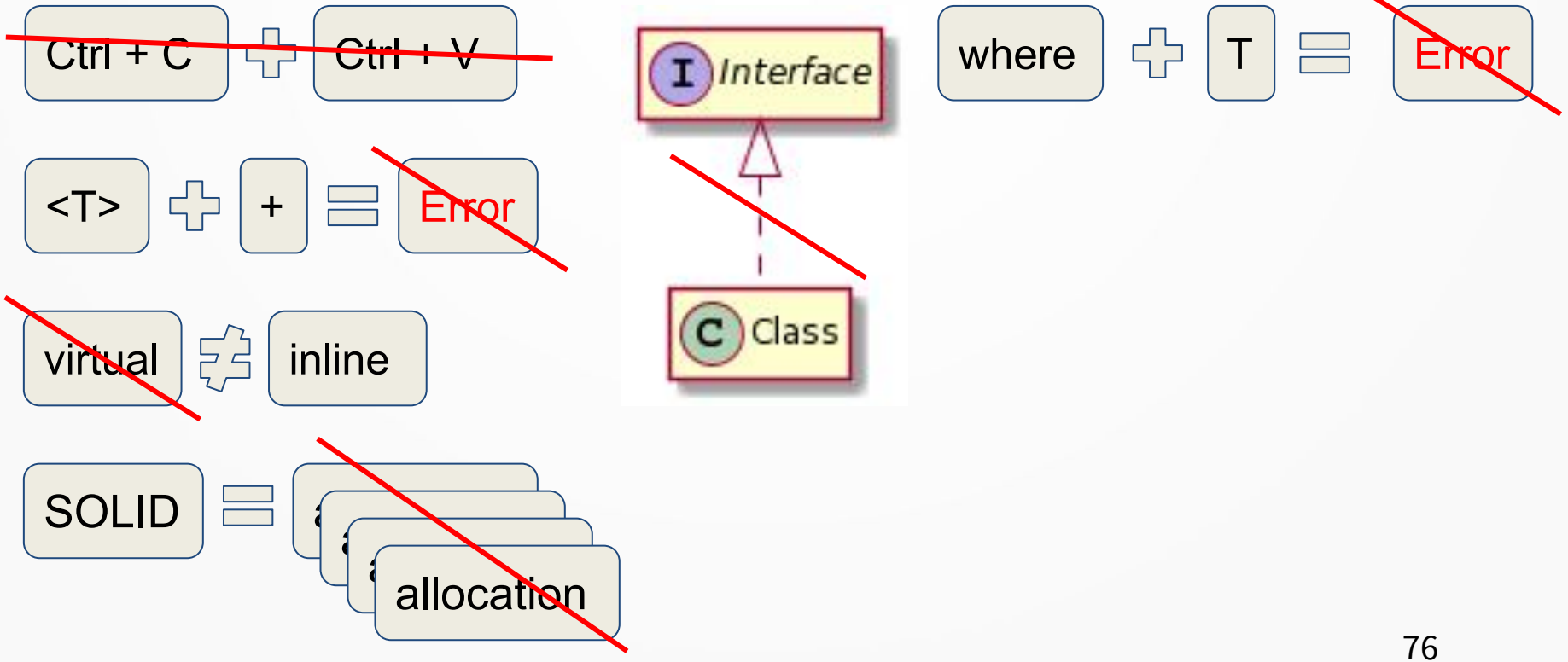



```
public static T Sum<T, TSummator>
(
    this T[] array,
    Additive<T, TSummator> initial
)
    where TSummator : struct, ISummator<T>
{
    var result = initial;
    for (var i = 0; i < array.Length; i++)
        result += array[i];
    return result;
}
```

```
_array.Sum(0.AsAdditive())
```

interface
struct
instance
wrapping
generic
call

Классы типов на C#



Цена классов типов на C#

- Haskell обходится без оберток

```
public interface ISummator<T>
{
    T Add(T left, T right);
}
```

```
public readonly ref struct Additive<T, TSummator>
    where TSummator : struct, ISummator<T>
{
    Additive(T value) => Value = value;

    public T Value { get; }

    public static Additive<T, TSummator> operator +
        (Additive<T, TSummator> left, Additive<T, TSummator> right)
        return default(TSummator).Add(left, right);

    public static implicit operator Additive<T, TSummator>(T value)
        => new Additive<T, TSummator>(value);
    public static implicit operator T(Additive<T, TSummator> value) => value.Value;
}
```

Цена классов типов на C#

- Haskell обходится без оберток

```
public readonly struct IntSummator :
    ISummator<int>
{
    public int Add(int left, int right)
        => left + right;
}
```

```
public static Additive<int, IntSummator>
    AsAdditive (this int value)
    => value;
```

Цена классов типов на C#

- Haskell обходится без оберток и явного указания экземпляра класса типов

```
public static T Sum<T, TSummator>
(
    this T[] array,
    Additive<T, TSummator> initial
)
    where TSummator : struct, ISummator<T>
{
    var result = initial;
    for (var i = 0; i < array.Length; i++)
        result += array[i];
    return result;
}
```

```
_array.Sum(0.AsAdditive())
```

Цена классов типов на C#

- Сквозь тернии к производительности

```
public readonly ref struct Additive<T, TSummator>
   where TSummator : struct, ISummator<T>
{
   Additive(T value) => Value = value;

   public T Value { get; }

   public static Additive<T, TSummator> operator +
       (Additive<T, TSummator> left, Additive<T, TSummator> right)
   {
       var summator = default(TSummator); // Ничего не меняющий лишний
код
       return default(TSummator)summator.Add(left, right);
   }

   public static implicit operator Additive<T, TSummator>(T value)
       => new Additive<T, TSummator>(value);
   public static implicit operator T(Additive<T, TSummator> value)
      => value.Value;
}
```

Добавление классов типов в C#

- Класс типов:

```
concept Num<A>
{
   A operator + (A a, A b);
   A operator * (A a, A b);
   A operator - (A a, A b);
   implicit operator A(int i);
}
```

interface
struct
instance
wrapping
generic
call

Добавление классов типов в C#

- Экземпляр:

```
instance NumInt
{
    int operator + (A a, A b) => a + b;
    int operator * (A a, A b) => a * b;
    int operator - (A a, A b) => -a;
    implicit operator int(int i) => i;
}
```

interface
struct
instance
wrapping
generic
call

Добавление классов типов в C#

- Использование:

```
public static A F<A, implicit NumA>(A x)
    where NumA : Num<A>
    => x*x + x + 666;
```

interface
struct
instance
wrapping
generic
call

Добавление классов типов в C#

```
concept Num<A> { // Класс типов
   A operator + (A a, A b);
   A operator * (A a, A b);
   A operator - (A a, A b);
   implicit operator A(int i);
}

instance NumInt { // Экземпляр
   int operator + (A a, A b) => a + b;
   int operator * (A a, A b) => a * b;
   int operator - (A a, A b) => -a;
   implicit operator int(int i) => i;
}

publix static A F<A, implicit NumA>(A x) where NumA : Num<A> // использование
   => x*x + x + 666;
```

interface
struct
instance
wrapping
generic
call

Ложка дегтя

2017

C# 10

Первоисточник

- https://github.com/dotnet/csharplang/issues/110

Как учесть переполнение?

```
public struct IntCheckedSummator : ISummator<int>
{
    public int Add(int left, int right)
    {
        checked
        {
            return left + right;
        }
    }
}

public static Additive<int, IntCheckedSummator> AsCheckedAdditive
    (this int value)
    => value;
```

interface
struct
instance
wrapping
generic
call

Цена проверки - минимум

С красивым сложением

```
L0000: mov eax, edx
L0002: xor edx, edx
L0004: mov r8d, [rcx+0x8]
L0008: test r8d, r8d
L000b: jle L001f
L000d: movsxd r9, edx
L0010: mov r9d, [rcx+r9*4+0x10]
L0015: add eax, r9d
L0018: inc edx
L001a: cmp r8d, edx
L001d: jg L000d
L001f: ret
```

С проверкой на переполнение

```
L0000: sub rsp, 0x28
L0004: mov eax, edx
L0006: xor edx, edx
L0008: mov r8d, [rcx+0x8]
L000c: test r8d, r8d
L000f: jle L0025
L0011: movsxd r9, edx
L0014: mov r9d, [rcx+r9*4+0x10]
L0019: add eax, r9d
L001c: jo L002a
L001e: inc edx
L0020: cmp r8d, edx
L0023: jg L0011
L0025: add rsp, 0x28
L0029: ret
L002a: call 0x7ffec7fe1c50
L002f: int3
```

Как исключить переполнение?

```
public interface ISummator<T, TIncrement>
{
    T Add(T left, TIncrement right);
}
```

LINQ
for
generic
magic
operator
homeopathy
typeclass
overflow

Как исключить переполнение?

```
public readonly ref struct Additive<TAccumulator, TIncrement, TSummator>
   where TSummator : struct, ISummator<TAccumulator, TIncrement>
{
   Additive(TAccumulator value) => Value = value;
   public TAccumulator Value { get; }

   public static Additive<TAccumulator, TIncrement, TSummator> operator +
       (Additive<TAccumulator, TIncrement, TSummator> left, TIncrement right)
   {
       var summator = default(TSummator);
       return summator.Add(left, right);
   }

   public static implicit operator Additive<TAccumulator, TIncrement, TSummator>(TAccumulator value)
       => new Additive<TAccumulator, TIncrement, TSummator>(value);
   public static implicit operator TAccumulator(Additive<TAccumulator, TIncrement, TSummator> value)
       => value.Value;
}
```

interface
struct
instance
wrapping
generic
call

Как исключить переполнение?

```
public struct LongIntSummator : ISummator<long, int>
{
    public long Add(long left, int right)
        => left + right;
}

public static Additive<long, int, LongIntSummator>
   AsAccumulator(this long value) => value;
```

interface
struct
instance
wrapping
generic
call

Как исключить переполнение?

```
public static TAccumulator Sum<T, TAccumulator, TSummator>
    (this T[] array, Additive<TAccumulator, T, TSummator> initial)
    where TSummator : struct, ISummator<TAccumulator, T>
{
    var result = initial;
    for (var i = 0; i < array.Length; i++)
        result += array[i];
    return result;
}
```

interface
struct
instance
wrapping
generic
call

Ожидаемый результат

С красивым сложением

```
L0000: mov eax, edx
L0002: xor edx, edx
L0004: mov r8d, [rcx+0x8]
L0008: test r8d, r8d
L000b: jle L001f
L000d: movsxd r9, edx
L0010: mov r9d, [rcx+r9*4+0x10]
L0015: add eax, r9d
L0018: inc edx
L001a: cmp r8d, edx
L001d: jg L000d
L001f: ret
```

Без риска переполнения

```
L0000: mov rax, rdx
L0003: xor edx, edx
L0005: mov r8d, [rcx+0x8]
L0009: test r8d, r8d
L000c: jle L0023
L000e: movsxd r9, edx
L0011: mov r9d, [rcx+r9*4+0x10]
L0016: movsxd r9, r9d
L0019: add rax, r9
L001c: inc edx
L001e: cmp r8d, edx
L0021: jg L000e
L0023: ret
```

Не только лишь массивы

```
public interface IIndexator<out T, TIndex, in TIndexable>
{
    TIndex GetCount(TIndexable indexable);
    T GetItem(TIndexable indexable, TIndex index);
}
```

LINQ
for
generic
magic
operator
homeopathy
typeclass
overflow
indexable

Не только лишь массивы

```
public struct Indexable<T, TIndex, TIndexable, TIndexator>
    where TIndexator : struct, IIndexator<T, TIndex, TIndexable>
    . . .
    public TIndexable Value { get; }
    public T this[TIndex index]
    {
        get
        {
            var indexator = default(TIndexator);
            return indexator.GetItem(Value, index);
        }
    }
    public TIndex Count
    {
        get
        {
            var indexator = default(TIndexator);
            return indexator.GetCount(Value);
        }
    }
```

interface
struct
instance
wrapping
generic
call

Не только лишь массивы

```
public struct ArrayIndexator<T> : IIndexator<T, int, T[]>
{
    public T GetItem(T[] array, int index) => array[index];

    public int GetCount(T[] array) => array.Length;
}
. . .
public static Indexable<T, int, T[], ArrayIndexator<T>>
     AsGenericIndexable<T>(this T[] array) => array;
```

interface
struct
instance
wrapping
generic
call

Не только лишь массивы

```
public static TAccumulator Sum<T, TAccumulator, TSummator, TIndexable, TIndexator>
(
    this Indexable<T, int, TIndexable, TIndexator> indexable,
    Additive<TAccumulator, T, TSummator> initial
)
    where TIndexator : struct, IIndexator<T, int, TIndexable>
    where TSummator : struct, ISummator<TAccumulator, T>
{
    var result = initial;
    for (var i = 0; i < indexable.Count; i++)
        result += indexable[i];
    return result;
}
```

interface
struct
instance
wrapping
generic
call

Фиаско

Method	Mean us	Error us	StdDev us	Ratio	Allocated B
LINQ	9.781	0.1939	0.2522	1.00	32
for	1.087	0.0204	0.0191	0.11	-
generic	5.915	0.1158	0.1334	0.61	24
magic	0.921	0.0017	0.0016	0.09	-
fiasco	9.600	0.1626	0.1521	0.98	-

Корень зла

```
public struct ArrayIndexator<T> :
    IIndexator<T, int, T[]>
{
    public T GetItem(T[] array, int index) => array[index];
    public int GetCount(T[] array) => array.Length;
}
. . .
```

Корень зла

```
public struct ArrayIndexator<T> :
    IIndexator<T, int, T[]> // Ссылочный тип
{
    public T GetItem(T[] array, int index) => array[index];
    public int GetCount(T[] array) => array.Length;
}
. . .
```

Волшебство

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static TAccumulator Sum<T, TAccumulator, TSummator, TIndexable, TIndexator>
(
   this Indexable<T, int, TIndexable, TIndexator> indexable,
   Additive<TAccumulator, T, TSummator> initial
)
   where TIndexator : struct, IIndexator<T, int, TIndexable>
   where TSummator : struct, ISummator<TAccumulator, T>
{
   var result = initial;
   for (var i = 0; i < indexable.Count; i++)
       result += indexable[i];
   return result;
}
```

Волшебство

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static TAccumulator Sum<T, TAccumulator, TSummator, TIndexable, TIndexator>
(
    this Indexable<T, int, TIndexable, TIndexator> indexable,
    Additive<TAccumulator, T, TSummator> initial
)
    where TIndexator : struct, IIndexator<T, int, TIndexable>
    where TSummator : struct, ISummator<TAccumulator, T>
{
    var result = initial;
    for (var i = 0; i < indexable.Count; i++)
        result += indexable[i];
    return result;
}
```

LINQ
for
generic
magic
operator
homeopathy
typeclass
overflow
indexable
inline

Чудо инлайна

Без риска переполнения

```
L0000: mov rax, rdx
L0003: xor edx, edx
L0005: mov r8d, [rcx+0x8]
L0009: test r8d, r8d
L000c: jle L0023
L000e: movsxd r9, edx
L0011: mov r9d, [rcx+r9*4+0x10]
L0016: movsxd r9, r9d
L0019: add rax, r9
L001c: inc edx
L001e: cmp r8d, edx
L0021: jg L000e
L0023: ret
```

С обобщением на индексированные коллекции

```
L0000: xor eax, eax
L0002: xor edx, edx
L0004: jmp L0016
L0006: movsxd r8, edx
L0009: mov r8d, [rcx+r8*4+0x10]
L000e: movsxd r8, r8d
L0011: add rax, r8
L0014: inc edx
L0016: cmp [rcx+0x8], edx
L0019: jg L0006
L001b: ret
```

Победа!

Method	Mean us	Error us	StdDev us	Ratio	Allocated B
LINQ	9.781	0.1939	0.2522	1.00	32
for	1.087	0.0204	0.0191	0.11	-
generic	5.915	0.1158	0.1334	0.61	24
magic	0.921	0.0017	0.0016	0.09	-
fiasco	9.600	0.1626	0.1521	0.98	-
inline	1.077	0.0104	0.0092	0.11	-

А если без волшебства?

```
public interface IIndexable<out T, TIndex, TIndexator>
{
    TIndex Count { get; }
    T this[TIndex index] { get; }
}
```

LINQ
for
generic
magic
operator
homeopathy
typeclass
overflow
indexable
inline
no magic

А если без волшебства?

```
public readonly struct Indexable<T, TIndex, TIndexable> : IIndexable<T, TIndex>
   where TIndexable : struct, IIndexable<T, TIndex>
{
   public Indexable(TIndexable indexable) => Value = indexable;

   public TIndexable Value { get; }

   public T this[TIndex index] => Value[index];

   public TIndex Count => Value.Count;
}
```

interface
struct
instance
wrapping
generic
call

А если без волшебства?

```
public struct ArrayIndexable<T> : IIndexable<T, int>
{
   internal ArrayIndexable(T[] value) => Value = value;

   public T[] Value { get; }

   public T this[int index] => Value[index];

   public int Count => Value.Length;

   public static implicit operator ArrayIndexable<T>(T[] array)
       => new ArrayIndexable<T>(array);

   public static implicit operator T[](ArrayIndexable<T> value) => value.Value;
}

. . .
public static Indexable<T, int, ArrayIndexable<T>> AsIndexable<T>(this T[] array)
   => new Indexable<T, int, ArrayIndexable<T>>(array);
```

interface
struct
instance
wrapping
generic
call

А если без волшебства?

```
public struct ArrayIndexable<T> : IIndexable<T, int, T[]> // Массива в параметрах нет
{
    internal ArrayIndexable(T[] value) => Value = value;
    public T[] Value { get; } // Структура-обертка вокруг ссылочного типа

    public T this[int index] => Value[index];
    public int Count => Value.Length;

    public static implicit operator ArrayIndexable<T>(T[] array)
        => new ArrayIndexable<T>(array);
    public static implicit operator T[](ArrayIndexable<T> value) => value.Value;
}
. . .
public static Indexable<T, int, ArrayIndexable<T>> AsIndexable<T>(this T[] array)
    => new Indexable<T, int, ArrayIndexable<T>>(array);
```

А если без волшебства?

```
public static TAccumulator Sum<T, TAccumulator, TSummator, TIndexable>
(
    this Indexable<T, int, TIndexable> indexable,
    Additive<TAccumulator, T, TSummator> initial
)
    where TIndexable : struct, IIndexable<T, int>
    where TSummator : struct, ISummator<TAccumulator, T>
{
    var result = initial;
    for (var i = 0; i < indexable.Count; i++)
        result += indexable[i];
    return result;
}
```

interface
struct
instance
wrapping
generic
call

Инлайн или не инлайн?

С волшебством

```
L0000: xor eax, eax
L0002: xor edx, edx
L0004: jmp L0016
L0006: movsxd r8, edx
L0009: mov r8d, [rcx+r8*4+0x10]
L000e: movsxd r8, r8d
L0011: add rax, r8
L0014: inc edx
L0016: cmp [rcx+0x8], edx
L0019: jg L0006
L001b: ret
```

Без волшебства

```
L0000: mov rax, rdx
L0003: xor edx, edx
L0005: jmp L0017
L0007: movsxd r8, edx
L000a: mov r8d, [rcx+r8*4+0x10]
L000f: movsxd r8, r8d
L0012: add rax, r8
L0015: inc edx
L0017: cmp [rcx+0x8], edx
L001a: jg L0007
L001c: ret
```

Класс типов: итерируемое

```
public interface IEnumerable
    <out T, out TEnumerator>
    : IEnumerable<T>
    where TEnumerator : IEnumerator<T>
{
    new TEnumerator GetEnumerator();
}
```

LINQ
for
generic
magic
operator
homeopathy
typeclass
overflow
indexable
inline
no magic
enumerable

Класс типов: итерируемое

```
public interface IEnumerable
    <out T, out TEnumerator> // Параметр-итератор
    : IEnumerable<T>
    where TEnumerator : IEnumerator<T>
{
    new TEnumerator GetEnumerator();
}
```

Класс типов: итерируемое

```
public interface IEnumerable
    <out T, out TEnumerator> // Параметр-итератор
    : IEnumerable<T>
    where TEnumerator : ~~struct,~~ IEnumerator<T>
{
    new TEnumerator GetEnumerator();
}
```

Класс типов: итерируемое

```
public interface IEnumerable
    <out T, out TEnumerator> // Параметр-итератор
    : IEnumerable<T>
    where TEnumerator : struct, IEnumerator<T>
{
    new TEnumerator GetEnumerator(); // Новый метод
}
```

Обертка для итерируемых

```
public readonly ref struct Enumerable<T, TEnumerator, TEnumerable>
   where TEnumerator : IEnumerator<T>
   where TEnumerable : IEnumerable<T, TEnumerator>
{
   Enumerable(TEnumerable value) => Value = value;

   public TEnumerable Value { get; }

   public TEnumerator GetEnumerator() => Value.GetEnumerator();

   public static implicit operator Enumerable<T, TEnumerator, TEnumerable>
       (TEnumerable value)
       => new Enumerable<T, TEnumerator, TEnumerable>(value);
   public static implicit operator TEnumerable
       (Enumerable<T, TEnumerator, TEnumerable> value)
       => value.Value;
}
```

interface
struct
instance
wrapping
generic
call

Итератор для массива

```
public struct ArrayEnumerator<T> : IEnumerator<T>
{
   internal ArrayEnumerator(T[] array) => (_i, _array, Current) = (-1, array, default);

   readonly T[] _array;
   int _i;

   public T Current { get; private set; }
   object IEnumerator.Current => Current;
   public void Dispose() { }
   public void Reset() => throw new NotImplementedException();

   [MethodImpl(MethodImplOptions.AggressiveInlining)]
   public bool MoveNext()
   {
       if ((uint)++_i >= (uint)_array.Length)
           return false;
       Current = _array[_i];
       return true;
   }
}
```

Экземпляр для массива

```
public readonly struct ArrayEnumerable<T>
    : IEnumerable<T, ArrayEnumerator<T>>
{
    internal ArrayEnumerable(T[] value) => Value = value;

    public T[] Value { get; }

    public ArrayEnumerator<T> GetEnumerator()
        => new ArrayEnumerator<T>(Value);

    IEnumerator<T> IEnumerable<T>.GetEnumerator() => GetEnumerator();
    IEnumerator IEnumerable.GetEnumerator() => GetEnumerator();
}
```

interface
struct
instance
wrapping
generic
call

Оборачивание массива

```
public static Enumerable<T, ArrayEnumerator<T>, ArrayEnumerable<T>>
    AsStructEnumerable<T>(this T[] list)
    => new ArrayEnumerable<T>(list)
```

interface
struct
instance
wrapping
generic
call

Сумма без индекса

```
public static TAccumulator Sum<T, TAccumulator, TSummator, TEnumerable, TEnumerator>
(
    this Enumerable<T, TEnumerator, TEnumerable> enumerable,
    Additive<TAccumulator, T, TSummator> initial
)
    where TEnumerator : IEnumerator<T>
    where TEnumerable : IEnumerable<T, TEnumerator>
    where TSummator : struct, ISummator<TAccumulator, T>
{
    var result = initial;
    foreach (var i in enumerable)
        result += i;
    return result;
}
```

interface
struct
instance
wrapping
generic
call

Почему так долго?

Method	Mean us	Error us	StdDev us	Ratio	Allocated B
LINQ	9.781	0.1939	0.2522	1.00	32
for	1.087	0.0204	0.0191	0.11	-
generic	5.915	0.1158	0.1334	0.61	24
magic	0.921	0.0017	0.0016	0.09	-
enumerable	2.693	0.0248	0.0220	0.30	-

Граница на замке

По индексу

```
L0000: mov rax, rdx
L0003: xor edx, edx
L0005: jmp L0017
L0007: movsxd r8, edx
L000a: mov r8d, [rcx+r8*4+0x10]
L000f: movsxd r8, r8d
L0012: add rax, r8
L0015: inc edx
L0017: cmp [rcx+0x8], edx
L001a: jg L0007
L001c: ret
```

С итератором

```
L0000: push rbp
L0001: sub rsp, 0x30
L0005: lea rbp, [rsp+0x30]
L000a: mov [rbp-0x10], rsp
L000e: mov rax, rdx
L0011: mov edx, 0xffffffff
L0016: xor r8d, r8d
L0019: jmp L0021
L001b: movsxd r9, r8d
L001e: add rax, r9
L0021: inc edx
L0023: cmp [rcx+0x8], edx
L0026: jg L0032
L0028: xor r9d, r9d
L002b: jmp L0045
L002d: call 0x7ffec7fe1e00
L0032: cmp edx, [rcx+0x8]
L0035: jae L002d
L0037: movsxd r8, edx
L003a: mov r8d, [rcx+r8*4+0x10]
L003f: mov r9d, 0x1
L0045: test r9d, r9d
L0048: jnz L001b
L004a: lea rsp, [rbp]
L004e: pop rbp
L004f: ret
```

А может, договоримся?

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public bool MoveNext()
{
    if ((uint)++_i >= (uint)_array.Length)
        return false;
    Current = _array[_i];
    return true;
}
```

А может, договоримся?

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public bool MoveNext()
{
    if ((uint)++_i >= (uint)_array.Length)
        return false;
    Current = _array[_i];
    return true;
}
```

LINQ
for
generic
magic
operator
homeopathy
typeclass
overflow
indexable
inline
no magic
enumerable
no borders

Таможня дает добро!

Method	Mean us	Error us	StdDev us	Ratio	Allocated B
LINQ	9.781	0.1939	0.2522	1.00	32
for	1.087	0.0204	0.0191	0.11	-
generic	5.915	0.1158	0.1334	0.61	24
magic	0.921	0.0017	0.0016	0.09	-
enumerable	2.693	0.0248	0.0220	0.30	-
enumerable2	1.136	0.0061	0.0054	0.12	-

Таможня дает добро!

С хитрым итератором

```
L0000: push rbp
L0001: sub rsp, 0x10
L0005: lea rbp, [rsp+0x10]
L000a: mov [rbp-0x10], rsp
L000e: mov rax, rdx
L0011: mov edx, 0xffffffff
L0016: xor r8d, r8d
L0019: jmp L0021
L001b: movsxd r9, r8d
L001e: add rax, r9
L0021: inc edx
L0023: cmp [rcx+0x8], edx
L0026: ja L002d
L0028: xor r9d, r9d
L002b: jmp L003b
L002d: movsxd r8, edx
L0030: mov r8d, [rcx+r8*4+0x10]
L0035: mov r9d, 0x1
L003b: test r9d, r9d
L003e: jnz L001b
L0040: lea rsp, [rbp]
L0044: pop rbp
L0045: ret
```

С обычным итератором

```
L0000: push rbp
L0001: sub rsp, 0x30
L0005: lea rbp, [rsp+0x30]
L000a: mov [rbp-0x10], rsp
L000e: mov rax, rdx
L0011: mov edx, 0xffffffff
L0016: xor r8d, r8d
L0019: jmp L0021
L001b: movsxd r9, r8d
L001e: add rax, r9
L0021: inc edx
L0023: cmp [rcx+0x8], edx
L0026: jg L0032
L0028: xor r9d, r9d
L002b: jmp L0045
L002d: call 0x7ffec7fe1e00
L0032: cmp edx, [rcx+0x8]
L0035: jae L002d
L0037: movsxd r8, edx
L003a: mov r8d, [rcx+r8*4+0x10]
L003f: mov r9d, 0x1
L0045: test r9d, r9d
L0048: jnz L001b
L004a: lea rsp, [rbp]
L004e: pop rbp
L004f: ret
```

Обещание выполнено!

```
public static TAccumulator Sum<T, TAccumulator, TSummator, TEnumerable, TEnumerator>
(
    this Enumerable<T, TEnumerator, TEnumerable> enumerable,
    Additive<TAccumulator, T, TSummator> initial
)
    where TEnumerator : IEnumerator<T>
    where TEnumerable : IEnumerable<T, TEnumerator>
    where TSummator : struct, ISummator<TAccumulator, T>
{
    var result = initial;
    foreach (var i in enumerable)
        result += i;
    return result;
}
```

Обещание выполнено!

```
public static TAccumulator Sum<T, TAccumulator, TSummator, TEnumerable, TEnumerator>
(
    this Enumerable<T, TEnumerator, TEnumerable> enumerable,
    Additive<TAccumulator, T, TSummator> initial
)
    where TEnumerator : IEnumerator<T>
    where TEnumerable : IEnumerable<T, TEnumerator>
    where TSummator : struct, ISummator<TAccumulator, T>
{
    var result = initial;
    foreach (var i in enumerable)
        result += i;
    return result;
}
```

Обещание выполнено!

```
public static TAccumulator Sum<T, TAccumulator, TSummator, TEnumerable, TEnumerator>
(
    this Enumerable<T, TEnumerator, TEnumerable> enumerable,
    Additive<TAccumulator, T, TSummator> initial
)
    where TEnumerator : IEnumerator<T>
    where TEnumerable : IEnumerable<T, TEnumerator>
    where TSummator : struct, ISummator<TAccumulator, T>
{
    var result = initial;
    foreach (var i in enumerable)
        result += i;
    return result;
}
```

LINQ
for
generic
magic
operator
homeopathy
typeclass
overflow
indexable
inline
no magic
enumerable
PROFIT

Теперь комфортно!

Бой

- SequenceEqual
  <
    TLeftIndexable<TItem>,
    TRightIndexable<TItem>,
    TComparator<TItem>
  >

База для сравнения

```
static int[] _array = Enumerable.Range
    (0, 1000).ToArray();

static int[] _array2 = Enumerable.Range
    (0, 999).Concat(new int[] { 0 }).ToArray();
```

LINQ

```
_array.SequenceEqual(_array2);
```

Дедовский способ

```
bool ForSequenceEqual(int[] left, int[] right)
{
    if (left == null | right == null || left.Length != right.Length)
        return false;
    for (var c = 0; c < left.Length; c++)
        if (left[c] != right[c])
            return false;
    return true;
}
```

Класс типов: сравниваемое

```
public interface IEqualityComparer<T>
{
    bool Equals(T left, T right);
    public int GetHashCode(T obj);
}
```

interface
struct
instance
wrapping
generic
call

Сравниваемое: обертка

```
public struct Equatable<T, TComparer> : IEquatable<Equatable<T, TComparer>>
    where TComparer : struct, IEqualityComparer<T>
{
    public override int GetHashCode() => default(TComparer).GetHashCode(Value);

    public static bool operator ==(Equatable<T, TComparer> left, Equatable<T, TComparer> right)
    {
        var comparer = default(TComparer);
        return comparer.Equals(left, right);
    }

    public static bool operator !=(Equatable<T, TComparer> left, Equatable<T, TComparer> right)
    {
        var comparer = default(TComparer);
        return !comparer.Equals(left, right);
    }
}
```

interface
struct
instance
wrapping
generic
call

Экземпляр для целых

```
public struct IntComparer : IEqualityComparer<int>
{
    public bool Equals(int left, int right) => left == right;

    public int GetHashCode(int obj) => obj.GetHashCode();
}
```

interface
struct
instance
wrapping
generic
call

Оборачивание

```
public static Equatable<T, TComparer> AsEquatable<T, TComparer>
    (this T value, TComparer _)
    where TComparer: struct, IEqualityComparer<T>
    => value;
```

interface
struct
instance
wrapping
generic
call

Обобщенный SequenceEqual

```
public static bool SequenceEqual<T, TLeftIndexable, TRightIndexable, TComparer>
(
    this Indexable<T, int, TLeftIndexable> left,
    Indexable<T, int, TRightIndexable> right,
    TComparer comparer = default
)
    where TLeftIndexable : struct, IIndexable<T, int>
    where TRightIndexable : struct, IIndexable<T, int>
    where TComparer : struct, IEqualityComparer<T>
{
    if (left.Count != right.Count) return false;
    for (var i = 0; i < left.Count; i++)
    {
        Equatable<T, TComparer> l = left[i];
        if (l != right[i]) return false;
    }
    return true;
}
```

interface
struct
instance
wrapping
generic
call

Использование

```
_array.AsIndexable().SequenceEqual(_array2.AsIndexable(), default(IntComparer))
```

interface
struct
instance
wrapping
generic
call

Хорошо, но мало

Method	Mean us	Error us	StdDev us	Ratio	Allocated B
LINQ	13.691	0.1469	0.1302	1.00	-
for	0.894	0.0172	0.0162	0.07	-
indexable	1.775	0.0127	0.0106	0.13	-

Цена одной команды

Цикл for

```
L0055: movsxd r10, eax
L0058: mov r11d, [rcx+r10*4+0x10]
L005d: cmp eax, r9d
L0060: jae L0081
L0062: cmp r11d, [rdx+r10*4+0x10]
L0067: jnz L007a
L0069: inc eax
L006b: cmp r8d, eax
L006e: jg L0055
L0070: mov eax, 0x1
L0075: add rsp, 0x28
L0079: ret
L007a: xor eax, eax
L007c: add rsp, 0x28
L0080: ret
L0081: call 0x7ffec7fe1e00
L0086: int3
```

Обобщенный метод

```
L001f: movsxd r10, eax
L0022: mov r11d, [rcx+r10*4+0x10]
L0027: cmp eax, r9d
L002a: jae L004e
L002c: mov r10d, [rdx+r10*4+0x10]
L0031: cmp r11d, r10d
L0034: jnz L0047
L0036: inc eax
L0038: cmp r8d, eax
L003b: jg L001f
L003d: mov eax, 0x1
L0042: add rsp, 0x28
L0046: ret
L0047: xor eax, eax
L0049: add rsp, 0x28
L004d: ret
L004e: call 0x7ffec7fe1e00
L0053: int3
```

Универсальный компаратор

```
public struct DefaultComparer<T> : IEqualityComparer<T>
{
    public bool Equals(T left, T right)
        => EqualityComparer<T>.Default.Equals(left, right);

    public int GetHashCode(T obj)
        => EqualityComparer<T>.Default.GetHashCode(obj);
}
```

interface
struct
instance
wrapping
generic
call

Как это работает?

Method	Mean us	Error us	StdDev us	Ratio	Allocated B
LINQ	13.691	0.1469	0.1302	1.00	-
for	0.894	0.0172	0.0162	0.07	-
indexable	1.775	0.0127	0.0106	0.13	-
default	2.015	0.0398	0.0609	0.15	-

Компараторная магия

DefaultComparer<int>

```
L001f: movsxd r10, eax
L0022: mov r11d, [rcx+r10*4+0x10]
L0027: cmp eax, r9d
L002a: jae L0059
L002c: mov r10d, [rdx+r10*4+0x10]
L0031: cmp r11d, r10d
L0034: setz r10b
L0038: movzx r10d, r10b
L003c: test r10d, r10d
L003f: jz L0052
L0041: inc eax
L0043: cmp r8d, eax
L0046: jg L001f
L0048: mov eax, 0x1
L004d: add rsp, 0x28
L0051: ret
L0052: xor eax, eax
L0054: add rsp, 0x28
L0058: ret
L0059: call 0x7ffec7e51e00
L005e: int3
```

IntComparer

```
L001f: movsxd r10, eax
L0022: mov r11d, [rcx+r10*4+0x10]
L0027: cmp eax, r9d
L002a: jae L004e
L002c: mov r10d, [rdx+r10*4+0x10]
L0031: cmp r11d, r10d
L0034: jnz L0047
L0036: inc eax
L0038: cmp r8d, eax
L003b: jg L001f
L003d: mov eax, 0x1
L0042: add rsp, 0x28
L0046: ret
L0047: xor eax, eax
L0049: add rsp, 0x28
L004d: ret
L004e: call 0x7ffec7fe1e00
L0053: int3
```

Секрет магии

```
public abstract partial class EqualityComparer<T>
    : IEqualityComparer, IEqualityComparer<T>
{
// public static EqualityComparer<T> Default is
runtime-specific
```

Компаратор для объектов

```
public struct ObjectComparer : IEqualityComparer<object>
{
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    public new bool Equals(object left, object right)
        => left == right;

    public int GetHashCode(object obj) => obj.GetHashCode();
}
```

interface
struct
instance
wrapping
generic
call

Магия не сработала?

Method	Mean us	Error us	StdDev us	Ratio	Allocated B
LINQ	43.214	0.8190	0.9103	1.00	-
for	0.916	0.0165	0.0093	0.02	-
indexable	45.750	0.4960	0.2106	1.06	-
inline	1.848	0.0259	0.0216	0.04	-
default	11.355	0.3031	0.2531	0.26	-

Цена двух команд

Цикл for

```
L0055: movsxd r10, eax
L0058: mov r11, [rcx+r10*8+0x10]
L005d: cmp eax, r9d
L0060: jae L0081
L0062: cmp r11, [rdx+r10*8+0x10]
L0067: jnz L007a
L0069: inc eax
L006b: cmp r8d, eax
L006e: jg L0055
```

Обобщенный метод

```
L001c: movsxd r10, r8d
L001f: mov r10, [rcx+r10*8+0x10]
L0024: cmp r8d, r9d
L0027: jae L004c
L0029: movsxd r11, r8d
L002c: mov r11, [rdx+r11*8+0x10]
L0031: cmp r10, r11
L0034: jnz L0045
L0036: inc r8d
L0039: cmp eax, r8d
L003c: jg L001c
```

Магия бессильна

DefaultComparer<object>

```
L0025: movsxd rcx, r14d
L0028: mov rdx, [rsi+rcx*8+0x10]
L002d: cmp r14d, ebp
L0030: jae L006d
L0032: movsxd rcx, r14d
L0035: mov r8, [rdi+rcx*8+0x10]
L003a: mov rcx, 0x293cbaf9120
L0044: mov rcx, [rcx]
L0047: call qword [rip-0xb1e6fbd]
L004d: test eax, eax
L004f: jz L0060
L0051: inc r14d
L0054: cmp ebx, r14d
L0057: jg L0025
```

ObjectComparer

```
L001c: movsxd r10, r8d
L001f: mov r10, [rcx+r10*8+0x10]
L0024: cmp r8d, r9d
L0027: jae L004c
L0029: movsxd r11, r8d
L002c: mov r11, [rdx+r11*8+0x10]
L0031: cmp r10, r11
L0034: jnz L0045
L0036: inc r8d
L0039: cmp eax, r8d
L003c: jg L001c
```

SequenceEquals на классах типов

- Оптимизированный перебор обеих сравниваемых коллекций, включая разнотипные

SequenceEquals на классах типов

- Оптимизированный перебор обеих сравниваемых коллекций, включая разнотипные
- DefaultComparator vs Copy & Paste

SequenceEquals на классах типов

- Оптимизированный перебор обеих сравниваемых коллекций, включая разнотипные
- DefaultComparator vs Copy & Paste
- Элементы-ссылки - выделенный компаратор и агрессивный инлайн

Экспансия

- Цель: LINQ без аллокаций

Select без аллокаций

```
public struct SelectEnumerator<T, TOut, TAtor, TSelector> : IEnumerator<TOut>
    where TAtor : IEnumerator<T>
    where TSelector : IFunc<T, TOut>
{
    internal SelectEnumerator(in TAtor enumerator, TSelector selector)
        => (_enumerator, _selector, Current) = (enumerator, selector, default);

    TAtor _enumerator;
    readonly TSelector _selector;

    public TOut Current { get; private set; }
    public void Dispose() { }
    object IEnumerator.Current => Current;
    public void Reset() => _enumerator.Reset();

    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    public bool MoveNext()
    {
        if (!_enumerator.MoveNext()) return false;
        var s = _selector;
        Current = s.Invoke(_enumerator.Current);
        return true;
    }
}
```

Select без аллокаций

```
public readonly struct SelectEnumerable<T, TOut, TAtor, TAble, TSelector>
    : IEnumerable<TOut, SelectEnumerator<T, TOut, TAtor, TSelector>>
    where TAtor : IEnumerator<T>
    where TAble : IEnumerable<T, TAtor>
    where TSelector : IFunc<T, TOut>
{
    internal SelectEnumerable(in Enumerable<T, TAtor, TAble> unwrap, TSelector selector)
        => (Value, _selector) = (unwrap.Value, selector);

    public TAble Value { get; }

    readonly TSelector _selector;

    public SelectEnumerator<T, TOut, TAtor, TSelector> GetEnumerator()
        => new SelectEnumerator<T, TOut, TAtor, TSelector>(Value.GetEnumerator(), _selector);

    IEnumerator<TOut> IEnumerable<TOut>.GetEnumerator() => GetEnumerator();

    IEnumerator IEnumerable.GetEnumerator() => GetEnumerator();
}
```

Использование

- LINQ

```
_array.Select(n => n + 5).Sum()
```

- LINQ без аллокаций

```
_array.AsStructEnumerable().Select(n => n + 5).Sum()
```

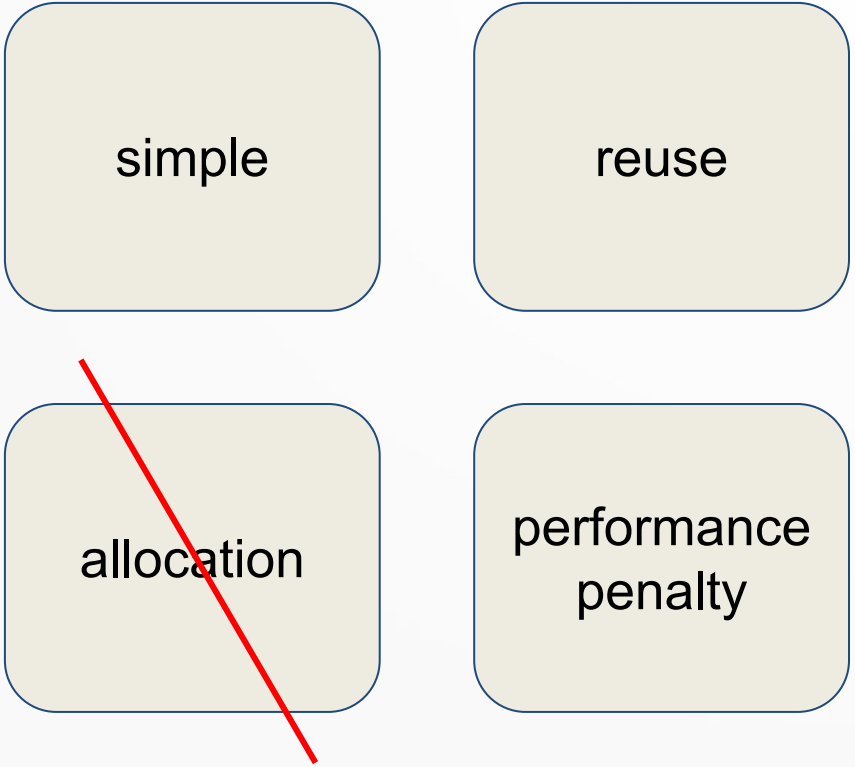
- Лобовой вариант

```
public static int ForeachSum(int[] array)
{
    var sum = 0;
    foreach (var n in array)
        sum += n + 5;
    return sum;
}
```

Частичный успех

Method	Mean us	Error us	StdDev us	Ratio	Allocated B
LINQ	12.530	0.2504	0.5282	1.00	48
foreach	1.337	0.0260	0.0329	0.11	-
noalloc	8.618	0.1710	0.3127	0.69	-

LINQ без аллокаций



Эзотерика

- Цель: LINQ без вызова делегатов в цикле

Использование

- LINQ

```
_array.Select(n => n + 5).Sum();
```

- LINQ без аллокаций

```
_array.AsStructEnumerable().Select(n => n + 5).Sum();
```

- LINQ без вызова делегатов в цикле

```
_array.AsStructLinqable().Select(n => n + 5).Sum(0);
```

Частичный успех

Method	Mean us	Error us	StdDev us	Ratio	Allocated B
LINQ	12.530	0.2504	0.5282	1.00	48
foreach	1.337	0.0260	0.0329	0.11	-
noalloc	8.618	0.1710	0.3127	0.69	-
exotic	4.316	0.0858	0.2448	0.35	-

Чем еще можно улучшить LINQ?

- LINQ Optimizer
- LINQ Rewriter

LINQ Optimizer

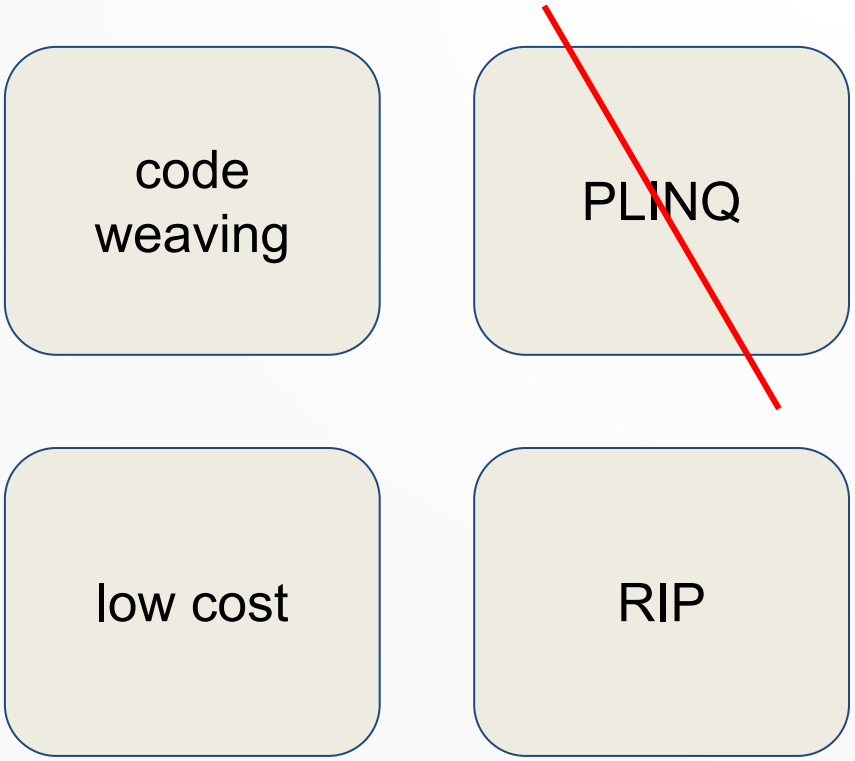
on the fly

PLINQ

F#

huge constant cost

LINQ Rewriter



Итоги

- `Generic<T> where T : struct`
- `struct Wrapper { public Class Value { get; set; }}`
- `[MethodImpl(MethodImplOptions.AggressiveInlining)]`
- ```
  public static bool SequenceEqual<T, TLeftIndexable, TRightIndexable, TComparer>
  (
      this Indexable<T, int, TLeftIndexable> left,
      Indexable<T, int, TRightIndexable> right,
      TComparer comparer = default
  )
      where TLeftIndexable : struct, IIndexable<T, int>
      where TRightIndexable : struct, IIndexable<T, int>
      where TComparer : struct, IEqualityComparer<T>
  {
      if (left.Count != right.Count) return false;
      for (var i = 0; i < left.Count; i++)
      {
          Equatable<T, TComparer> l = left[i];
          if (l != right[i]) return false;
      }
      return true;
  }
  ```
  ```

Что просилось в доклад

monad ≠ C# typeclass

LINQ <TaskLike>

ref struct ≠ first class citizen

Вопросы?

leo.bonart@gmail.com

https://github.com/Kirill-Maurin/Sample.Struct