Как в C# 11 появилась обобщённая математика

Обо мне

Степан Минин

Senior C#

- BMSTU & HSE Alumni
- Телеграм канал @steponeit
- Топ-10 лучших авторов про C# на Хабр
- Два open source проекта на 80+ звёзд суммарно
- Спикер Стачки, DevFest, DotNext, MskDotNet Meetup

Предыстория

Митап сообщества MskDotNet

Что случилось?

.NET 7 + C# 11 + Generic Math

StepOne

Вы используете Generic Math или abstract static в продакшн коде на своём проекте?

Анонимный опрос

7%	Да
43%	Нет
4%	Планируем
51%	Не понимаем зачем нужен инструмент

204 голоса

О чём поговорим?

01 Где не хватало Generic Math?

02 Что мы получили?

03 Инструкция к применению

04 Плюсы и минусы

05 Альтернативы

06 Итоги

Давняя боль .NET разработчиков

```
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;
}
```

Давняя боль .NET разработчиков

```
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;
}
```

Error CS0019:

Operator '+=' cannot be applied to operands of type 'T' and 'T'

Давняя боль .NET разработчиков

```
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;
}
```

Давняя боль .NET разработчиков

```
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;
}
```

Error CS1031:

Type expected

Что мы получили?

Обобщённая математика

Впервые представлена 29 июня 2020

https://github.com/dotnet/csharplang/blob/main/meetings/2020/LDM-2020-06-29.md

Принятие и обсуждение здесь:

https://github.com/dotnet/csharplang/issues/4436

Обобщённая математика

- Абстрактные статические члены в интерфейсах

```
interface IAddable<T> where T : IAddable<T>
{
    static abstract T Zero { get; }
    static abstract T operator +(T t1, T t2);
}
```

Обобщённая математика

- Абстрактные статические члены в интерфейсах

```
struct Int32 : IAddable<Int32>
{
    static int operator +(int x, int y) => x + y;
    public static int Zero => 0;
}
```

Обобщённая математика

- Теперь можем написать сумматор

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

Обобщённая математика

- Теперь можем написать сумматор

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

Паттерн CRTP

Обобщённая математика

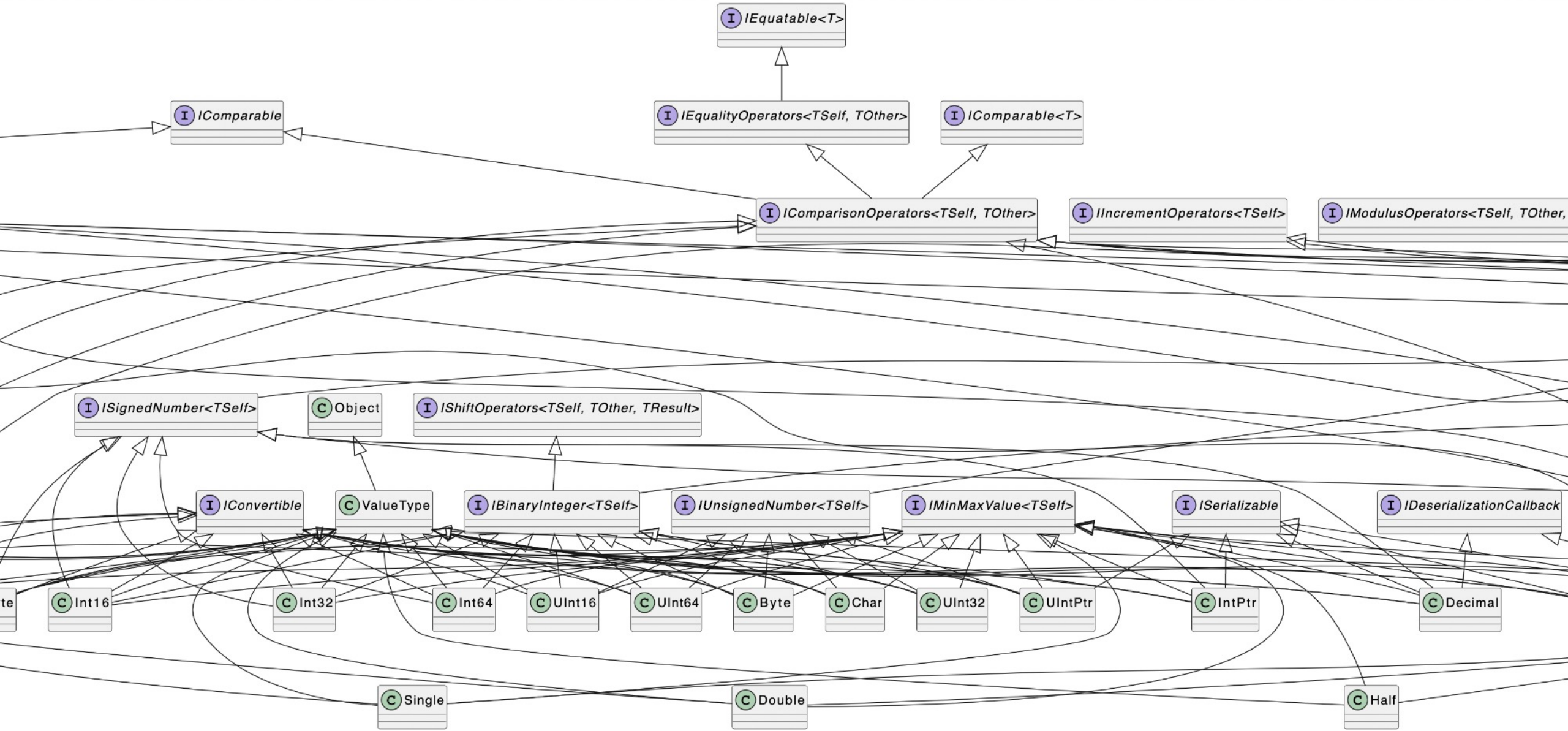
- На самом деле он будет выглядеть по-другому

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

Обобщённая математика

- На самом деле он будет выглядеть по-другому
- .NET Numerics!

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


IEquatable<T>
IComparable
IEqualityOperators<TSelf, TOther>
IComparable<T>
IComparisonOperators<TSelf, TOther>
IIncrementOperators<TSelf>
IModulusOperators<TSelf, TOther,
ISignedNumber<TSelf>
Object
IShiftOperators<TSelf, TOther, TResult>
IConvertible
ValueType
IBinaryInteger<TSelf>
IUnsignedNumber<TSelf>
IMinMaxValue<TSelf>
ISerializable
IDeserializationCallback
Int16
Int32
Int64
UInt16
UInt64
Byte
Char
UInt32
UIntPtr
IntPtr
Decimal
Single
Double
Half

Обобщённая математика

Гигансткий UML .NET Numerics

Не такая гигантская ссылка:
https://t.me/steponeitchat/3395

.NET Numerics

- Int32.cs

```
public readonly struct Int32 :
    IComparable,
    IConvertible,
    ISpanFormattable,
    IFormattable,
    IComparable<int>,
    IEquatable<int>,
    IBinaryInteger<int>,
    IBinaryNumber<int>,
    IBitwiseOperators<int, int, int>,
    INumber<int>,
```

.NET Numerics

- INumberBase.cs

```
public interface INumberBase<TSelf> :
  IAdditionOperators<TSelf, TSelf, TSelf>,
  IAdditiveIdentity<TSelf, TSelf>,
  IDecrementOperators<TSelf>,
  IDivisionOperators<TSelf, TSelf, TSelf>,
  IEquatable<TSelf>,
  IEqualityOperators<TSelf, TSelf, bool>,
  IIncrementOperators<TSelf>,
  IMultiplicativeIdentity<TSelf, TSelf>,
  IMultiplyOperators<TSelf, TSelf, TSelf>,
  ISpanFormattable,
  IFormattable,
  ISpanParsable<TSelf>,
  IParsable<TSelf>,
  ISubtractionOperators<TSelf, TSelf, TSelf>,
  IUnaryPlusOperators<TSelf, TSelf>,
  IUnaryNegationOperators<TSelf, TSelf>,
  IUtf8SpanFormattable,
  IUtf8SpanParsable<TSelf>
  where TSelf : INumberBase<TSelf>?
```

.NET Numerics

- IAdditionOperators.cs

```
public interface IAdditionOperators<TSelf, TOther,
TResult>
    where TSelf : IAdditionOperators<TSelf, TOther,
TResult>?
{
  static abstract TResult operator +(TSelf left, TOther
right);

  static virtual TResult operator checked +(TSelf left,
TOther right)
    => left + right;
}
```

.NET Numerics

- IAdditiveIdentity.cs

```
public interface IAdditiveIdentity<TSelf, TResult>
    where TSelf : IAdditiveIdentity<TSelf, TResult>?
{
  static abstract TResult AdditiveIdentity { get; }
}
```

.NET Numerics

- IUnaryNegationOperators.cs

```
public interface IUnaryNegationOperators<TSelf, TResult>
    where TSelf : IUnaryNegationOperators<TSelf,
TResult>?
{
  static abstract TResult operator -(TSelf value);

  static virtual TResult operator checked -(TSelf value)
    => -value;
}
```

.NET Numerics

- ISubstractionOperators.cs

```
public interface ISubtractionOperators<TSelf, TOther,
TResult>
    where TSelf : ISubtractionOperators<TSelf, TOther,
TResult>?
{
  static abstract TResult operator -(TSelf left, TOther
right);

  static virtual TResult operator checked -(TSelf left,
TOther right)
    => left - right;
}
```

.NET Numerics

- IMultiplyOperators.cs

```
public interface IMultiplyOperators<TSelf, TOther, TResult>
    where TSelf : IMultiplyOperators<TSelf, TOther,
TResult>?
{
  static abstract TResult operator *(TSelf left, TOther
right);

  static virtual TResult operator checked *(TSelf left,
TOther right)
    => left * right;
}
```

.NET Numerics

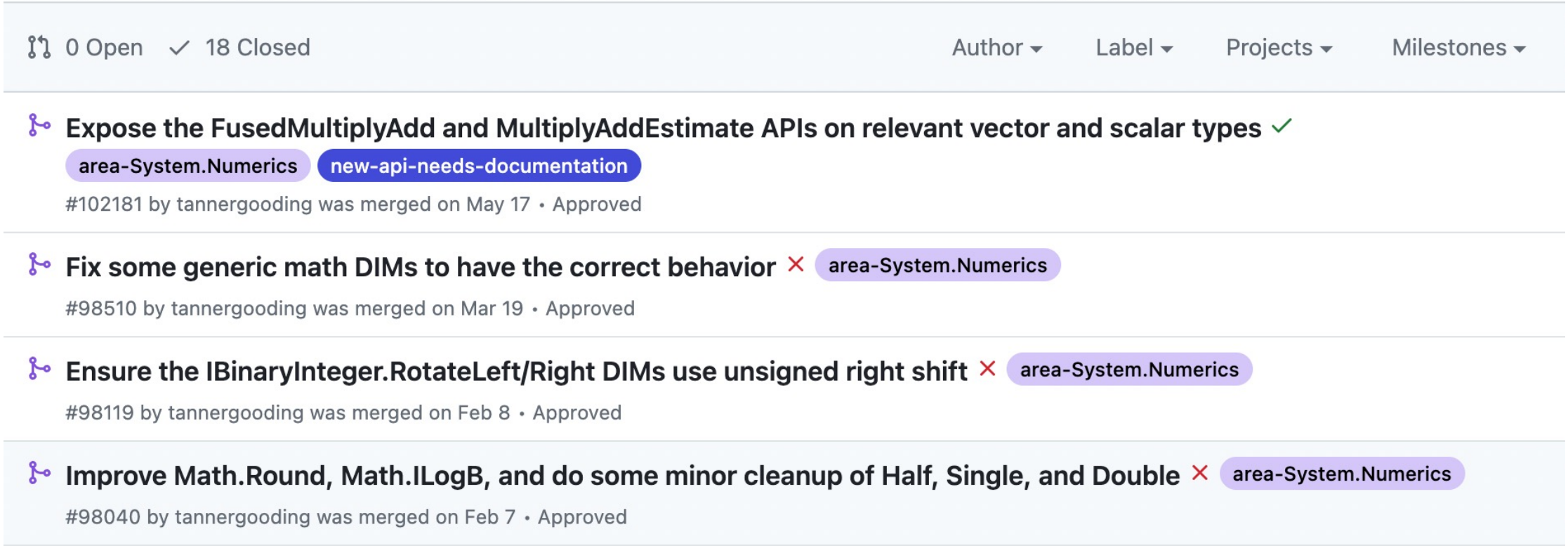
- IMultiplicativeIdentity.cs

```
public interface IMultiplicativeIdentity<TSelf, TResult>
    where TSelf : IMultiplicativeIdentity<TSelf, TResult>?
{
    static abstract TResult MultiplicativeIdentity { get; }
}
```

Как появился .NET Numerics?

Расширили BCL

https://github.com/dotnet/runtime/pulls?q=is%3Apr+author%3Atannergooding+generic+math+label%3Aarea-System.Numerics



Расширение BCL

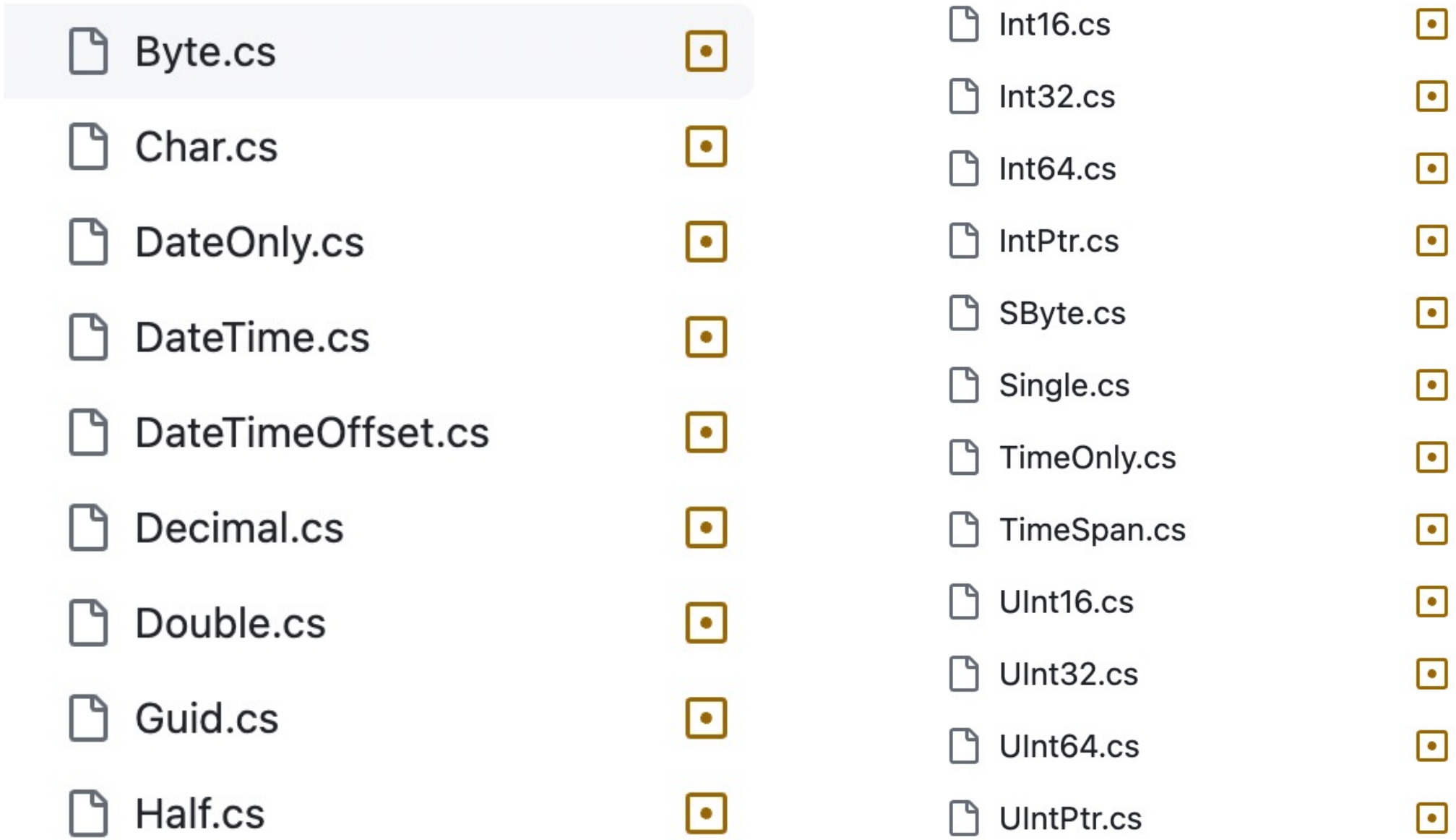
https://github.com/dotnet/runtime/pull/54650

Expose and implement generic math interfaces on core types #54650

Merged **tannergooding** merged 15 commits into `dotnet:main` from `tannergooding:generic-math` on Jul 3, 2021

Files changed 52 +15,921 −27

Расширение BCL



Расширение BCL

```
    public readonly partial struct DateTime : IComparable, ISpanFormattable, IConvertible, IComparable<DateTime>, IEquatable<DateTime>, ISerializable
+ #if FEATURE_GENERIC_MATH
+ #pragma warning disable SA1001
+         , IAdditionOperators<DateTime, TimeSpan, DateTime>,
+           IAdditiveIdentity<DateTime, TimeSpan>,
+           IComparisonOperators<DateTime, DateTime>,
+           IMinMaxValue<DateTime>,
+           ISpanParseable<DateTime>,
+           ISubtractionOperators<DateTime, TimeSpan, DateTime>,
+           ISubtractionOperators<DateTime, DateTime, TimeSpan>
+ #pragma warning restore SA1001
+ #endif // FEATURE_GENERIC_MATH
```

```
[RequiresPreviewFeatures]
static DateTime IParseable<DateTime>.Parse(string s, IFormatProvider? provider)
    => Parse(s, provider);

[RequiresPreviewFeatures]
static bool IParseable<DateTime>.TryParse([NotNullWhen(true)] string? s, IFormatProvider? provider, out DateTime result)
    => TryParse(s, provider, DateTimeStyles.None, out result);
```

Расширение BCL

- IAdditionOperators.cs
- IAdditiveIdentity.cs
- IBitwiseOperators.cs
- IComparisonOperators.cs
- IDecrementOperators.cs
- IDivisionOperators.cs
- IEqualityOperators.cs
- IFloatingPoint.cs
- IIncrementOperators.cs
- IInteger.cs
- IMinMaxValue.cs
- IModulusOperators.cs
- IMultiplicativeIdentity.cs
- IMultiplyOperators.cs
- INumber.cs
- IParseable.cs
- IShiftOperators.cs
- ISpanParseable.cs
- ISubtractionOperators.cs
- IUnaryNegationOperator...
- IUnaryPlusOperators.cs

Расширение BCL

2,787 src/libraries/System.Runtime/ref/System.Runtime.cs

Viewed

Load diff

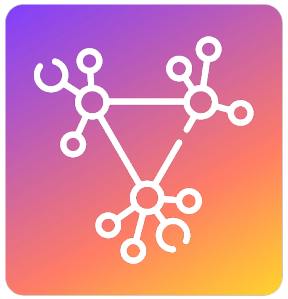
Large diffs are not rendered by default.

Расширение BCL

```
707         public readonly partial struct Byte : 
708   + #if FEATURE_GENERIC_MATH
709   + #pragma warning disable SA1001
710   +         , System.IBinaryInteger<byte>,
711   +           System.IMinMaxValue<byte>,
712   +           System.IUnsignedNumber<byte>
713   + #pragma warning restore SA1001
714   + #endif // FEATURE_GENERIC_MATH
715         {
```

```
754   +
755   + #if FEATURE_GENERIC_MATH
756   +         [System.Runtime.Versioning.RequiresPreviewFeaturesAttribute]
757   +         static byte IAdditiveIdentity<byte, byte>.AdditiveIdentity { get { throw null; } }
758   +
759   +         [System.Runtime.Versioning.RequiresPreviewFeaturesAttribute]
760   +         static byte IMinMaxValue<byte>.MinValue { get { throw null; } }
761   +         [System.Runtime.Versioning.RequiresPreviewFeaturesAttribute]
762   +         static byte IMinMaxValue<byte>.MaxValue { get { throw null; } }
763   +
764   +         [System.Runtime.Versioning.RequiresPreviewFeaturesAttribute]
765   +         static byte IMultiplicativeIdentity<byte, byte>.MultiplicativeIdentity { get { throw null; } }
766   +
767   +         [System.Runtime.Versioning.RequiresPreviewFeaturesAttribute]
768   +         static byte INumber<byte>.One { get { throw null; } }
769   +         [System.Runtime.Versioning.RequiresPreviewFeaturesAttribute]
770   +         static byte INumber<byte>.Zero { get { throw null; } }
771   +
772   +         [System.Runtime.Versioning.RequiresPreviewFeaturesAttribute]
773   +         static byte IAdditionOperators<byte, byte, byte>.operator +(byte left, byte right) { throw null; }
774   +
```

Как всё это работает?



В IL теперь допускается static abstract

Ранее это было запрещено

Платформа была готова со времён C# 8

Как всё это работает?

- Генерируется constrained. call

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

```
// [91 9 - 91 41]
IL_0000: constrained. !!0/*T*/
IL_0006: call         !1/*T*/ class [System.Runtime]System.Numerics.IAdditiveIdentity`2<!!0/*T*/, !!0/*T*/>::get_AdditiveIdentity()
IL_000b: stloc.0      // result
```

Где применять кроме математики?

Инструкция к применению

IParsable.cs

```
public interface IParsable<TSelf>
    where TSelf : IParsable<TSelf>?
{
  static abstract TSelf Parse(string s, IFormatProvider?
provider);

  static abstract bool TryParse(
    [NotNullWhen(true)] string? s,
    IFormatProvider? provider,
    [MaybeNullWhen(false)] out TSelf result);
}
```

Value object ASP.NET

```
public class DateRange
{
    public DateOnly? From { get; init; }
    public DateOnly? To { get; init; }
}
```

ASP.NET old model binding

```
internal class DateRangeModelBinder : IModelBinder
{
    public Task BindModelAsync(ModelBindingContext bindingContext)
    {
        ArgumentNullException.ThrowIfNull(bindingContext);
        var fieldName = bindingContext.FieldName;
        var valueProviderResult = bindingContext.ValueProvider.GetValue(fieldName);

        if (valueProviderResult == ValueProviderResult.None)
            return Task.CompletedTask;
        bindingContext.ModelState.SetModelValue(fieldName, valueProviderResult);
        var value = valueProviderResult.FirstValue;

        if (string.IsNullOrEmpty(value))
        {
            bindingContext.Result = ModelBindingResult.Failed();
            return Task.CompletedTask;
        }

        //...

    }
}
```

ASP.NET old model binding

```
internal class DateRangeModelBinder : IModelBinder
{
    public Task BindModelAsync(ModelBindingContext bindingContext)
    {
        //...
        var segments = value.Split(',',
           StringSplitOptions.RemoveEmptyEntries |
           StringSplitOptions.TrimEntries);
        var dtfi = new DateTimeFormatInfo { DateSeparator = "/" };
        if (segments.Length == 2 &&
            DateOnly.TryParse(segments[0], dtfi, out var fromDate) &&
            DateOnly.TryParse(segments[1], dtfi, out var toDate))
        {
            var dateRange = new DateRange { From = fromDate, To = toDate };
            bindingContext.Result = ModelBindingResult.Success(dateRange);
            return Task.CompletedTask;
        }
        bindingContext.Result = ModelBindingResult.Failed();
        return Task.CompletedTask;
    }
}
```

ASP.NET old model binding

```
// GET /WeatherForecast/ByRange?range=7/24/2022,07/26/2022
public IActionResult ByRange(
    [ModelBinder<DateRangeModelBinder>] DateRange range)
{
    // ...
}
```

ASP.NET new model binding

```
public class DateRange : IParsable<DateRange>
{
    public DateOnly? From { get; init; }
    public DateOnly? To { get; init; }

    public static DateRange Parse(string value, IFormatProvider? provider)
    {
        if (!TryParse(value, provider, out var result))
        {
            throw new ArgumentException(
                "Could not parse supplied value.", nameof(value));
        }

        return result;
    }
}
```

ASP.NET new model binding

```
public class DateRange : IParsable<DateRange>
{
    public static bool TryParse(string? value,
        IFormatProvider? provider, out DateRange dateRange)
    {
        var segments = value?.Split(
            ' ',
            StringSplitOptions.RemoveEmptyEntries |
            StringSplitOptions.TrimEntries);

        if (segments?.Length == 2
            && DateOnly.TryParse(segments[0], provider, out var fromDate)
            && DateOnly.TryParse(segments[1], provider, out var toDate))
        {
            dateRange = new DateRange { From = fromDate, To = toDate };
            return true;
        }

        dateRange = new DateRange { From = default, To = default };
        return false;
    }
}
```

ASP.NET new model binding

```
// GET /WeatherForecast/ByRange?range=7/24/2022,07/26/2022
public IActionResult ByRange([FromQuery] DateRange range)
{
    // ...
}
```

Варианты применения

- Задача: спроектировать cache-like хранилище, где ключ зависит от типа хранимого элемента

```
class Cache
{
    public void Set<TResource>(TResource resource)
    {
        // string cacheKey = ???
        // ключ зависит от TResource
    }
}
```

Варианты решения без GM

- Экземплярная абстракция

```
interface IResource
{
    string CacheKey { get; }
}
```

- А как реализовать операцию Get?

Варианты решения без GM

- Атрибут-контейнер + рефлексия

```
[CacheKey(nameof(MyResource) + "suffix")]
record MyResource;

class CacheKeyAttribute(string value) : Attribute
{
    public string Value { get; } = value;
}
```

Варианты решения без GM

- Выделение логики в сервис
- Как распространять и предоставлять?

```
abstract class KeyProvider
{
    public abstract string Key { get; }
}

abstract class Resource<TKeyProvider>
    where TKeyProvider : KeyProvider, new()
{
    public string CacheKey =>
        new TKeyProvider().Key;
}
```

Варианты решения без GM

- Явно отобразить типы на ключи

```
class Cache
{
    private readonly IReadOnlyDictionary<Type, string>
_resourceTypeToKeyMap;

    public Cache(IReadOnlyDictionary<Type, string>
resourceTypeToKeyMap) =>
        _resourceTypeToKeyMap = resourceTypeToKeyMap;
}
```

Решение с GM

```
interface IResource
{
    static abstract string CacheKey { get; }
}

class Cache
{
    public void Set<TResource>(TResource resource)
        where TResource : IResource
    {
        string cacheKey = TResource.CacheKey;
        // ...
    }
}
```

Новый подход к реализации паттернов

```
public interface IAsyncFactory<T>
{
    static abstract Task<T> CreateAsync();
}

public interface IExampleStrategy
{
    static abstract bool IsEnabled(string foo);

    void DoStuff(string foo);
}
```

Обобщённые атрибуты

```
public AutoDataAttribute()
   : this(
         (Func<IFixture>) (() =>
             (IFixture) new AutoFixture.Fixture()))
{
}
```

Обобщённые атрибуты

```
public interface IFixtureCustomizer
{
    static abstract void Customize(IFixture fixture);
}
```

Обобщённые атрибуты

```
public class AutoDataAttribute<TFixtureCustomizer> :
AutoDataAttribute
    where TFixtureCustomizer : IFixtureCustomizer
{
    public AutoDataAttribute() : base(
        fixtureFactory: () =>
        {
            var fixture = new Fixture();
            TFixtureCustomizer.Customize(fixture);
            return fixture;
        })
    {
    }
}
```

Обобщённые атрибуты

```
public class TenElementsInCollection : IFixtureCustomizer
{
    public static void Customize(IFixture fixture) =>
        fixture.RepeatCount = 10;
}


public class Tests
{
    [Theory, AutoData<TenElementsInCollection>]
    public void SomeTest(IReadOnlyCollection<int> ints)
    {
        ints.Count.Should().Be(10);
    }
}
```

Новый приём для DDD

```
interface IDomainEvent;

interface IEntityCreated<T> : IDomainEvent
{
    T Entity { get; }
}


record CustomerCreated(Customer Entity) :
    IEntityCreated<Customer>;
```

Новый приём для DDD

```
interface IFactory<TEntity, TEntityCreateDto>
    where TEntity : class, IFactory<TEntity,
TEntityCreateDto>
    where TEntityCreateDto : class
{
    static abstract IEntityCreated<TEntity> Create(
        TEntityCreateDto createDto);
}
```

Новый приём для DDD

```
record CustomerCreateDto(string Name, DateTimeOffset DateOfBirth);

class Customer(Guid id, string name, DateTimeOffset? dateOfBirth) :
    IFactory<Customer, CustomerCreateDto>
{
    // ...

    public static IEntityCreated<Customer> Create(
        CustomerCreateDto createDto) =>
        new CustomerCreated(
            new Customer(
                id: Guid.NewGuid(),
                createDto.Name,
                createDto.DateOfBirth));
}
```

Новый приём для DDD

```
record CustomerCreateDto(string Name, DateTimeOffset DateOfBirth);
record CustomerCreateWithoutDateOfBirthDto(string Name);


class Customer(Guid id, string name, DateTimeOffset? dateOfBirth) :
    IFactory<Customer, CustomerCreateDto>,
    IFactory<Customer, CustomerCreateWithoutDateOfBirthDto>
{

    // ...

    public static IEntityCreated<Customer> Create(
        CustomerCreateWithoutDateOfBirthDto createDto) =>
        new CustomerCreated(
            new Customer(
                id: Guid.NewGuid(),
                createDto.Name,
                dateOfBirth: null));
}
```

Выделение контрактов сгенерированных моделей таблиц БД

- Document.cs

```
partial class Document
{
    public required string Name { get; set; }

    // ...

    public static int NameLengthConstraint { get; }
}
```

Выделение контрактов сгенерированных моделей таблиц БД

- Экземплярный контракт

```
interface IDocument
{
    // ...
    public string Name { get; set; }
}
```

- А статические constraint'ы?

Выделение контрактов сгенерированных моделей таблиц БД

- Обобщённая математика пришла на помощь!

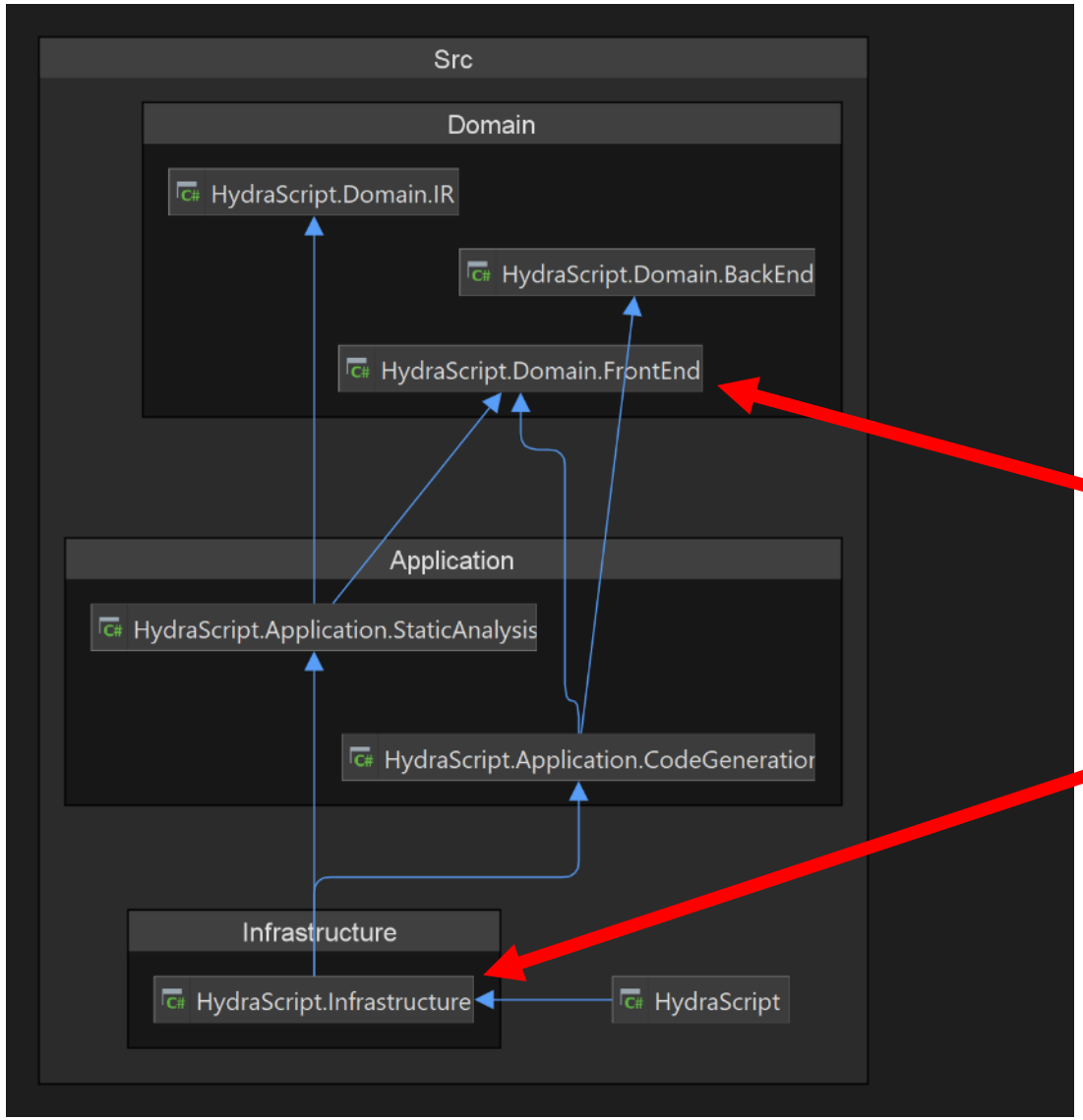
```
interface IDocumentConstraints<TDocument>
    where TDocument :
    IDocumentConstraints<TDocument>
{
    static abstract int NameLengthConstraint { get; }
}
```

Выделение контрактов сгенерированных моделей таблиц БД

- Document.Contract.cs

```
partial class Document :
    IDocument,
    IDocumentConstraints<Document>;
```

Архитектурная граница слоёв



https://github.com/Stepami/hydrascript

Проект – интерпретатор

1. Дано: DDD + Clean Architecture
2. Лексическая структура, описываемая регуляркой – сущность в домене
3. Регулярка конструируется служебным сурс генератором

Что делать?

Архитектурная граница слоёв

```
public interface IStructure : IEnumerable<TokenType>
{
    public Regex Regex { get; }

    public TokenType FindByTag(string tag);
}
```

Архитектурная граница слоёв

- Тот же тактический паттерн, но уже с GM

```
public interface IGeneratedRegexContainer
{
    public static abstract Regex GetRegex();
}
```

Архитектурная граница слоёв

- Тот же тактический паттерн, но уже с GM

```
public class Structure<TContainer>(ITokenTypesProvider provider) : IStructure
    where TContainer : IGeneratedRegexContainer
{
    //...

    public Regex Regex { get; } = TContainer.GetRegex();
}


services.AddSingleton<IStructure, Structure<GeneratedRegexContainer>>();
```

Немного про ML.NET

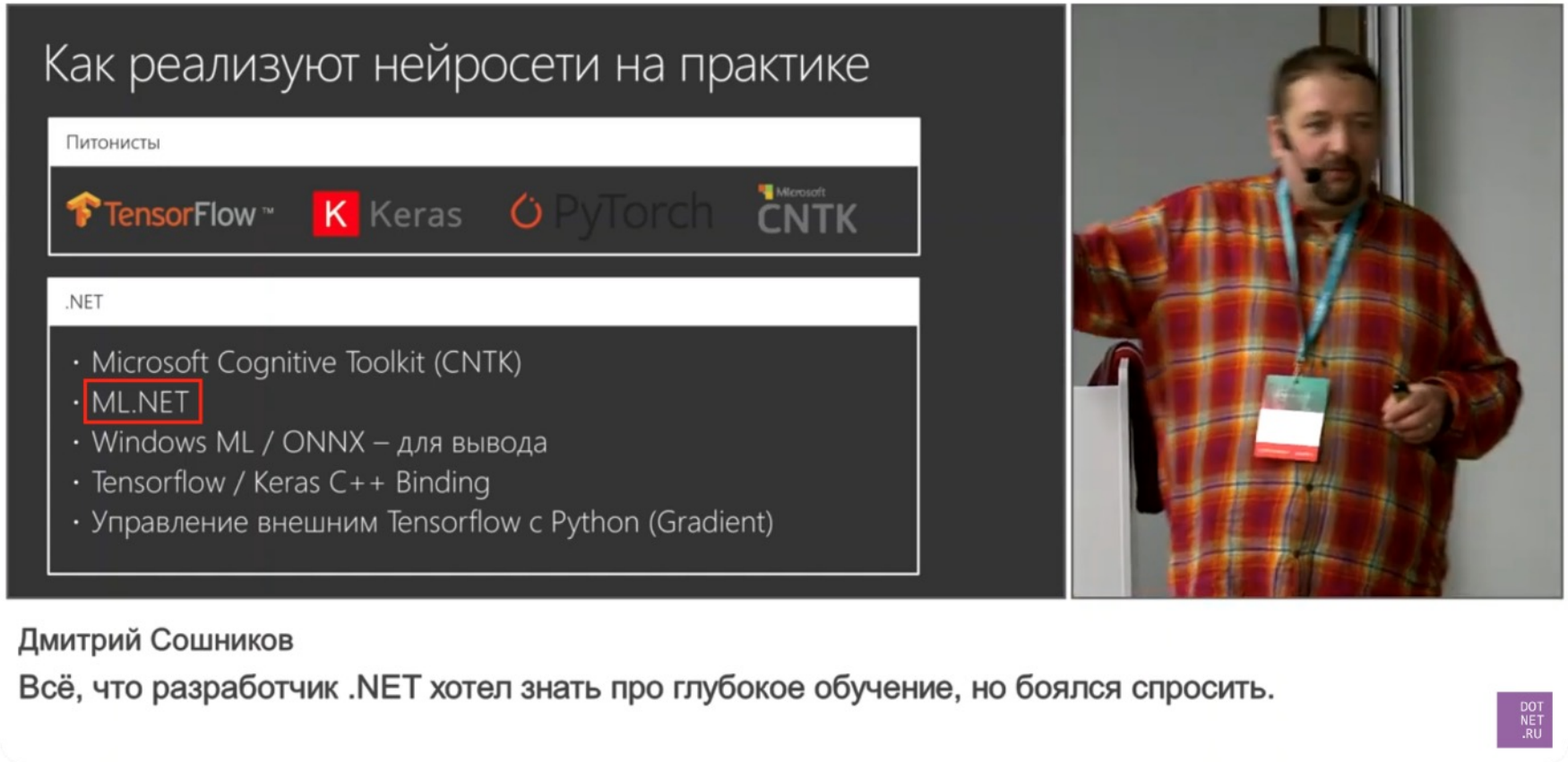
ML.NET

это инструмент, который привносит машинное обучение в .NET приложения как в online, так и offline сценариях

Центральным понятием для ML.NET является модель машинного обучения

Немного про ML.NET

Вместе с ML.NET можно обучать собственную модели или импортировать предобученные модели TensorFlow и ONNX

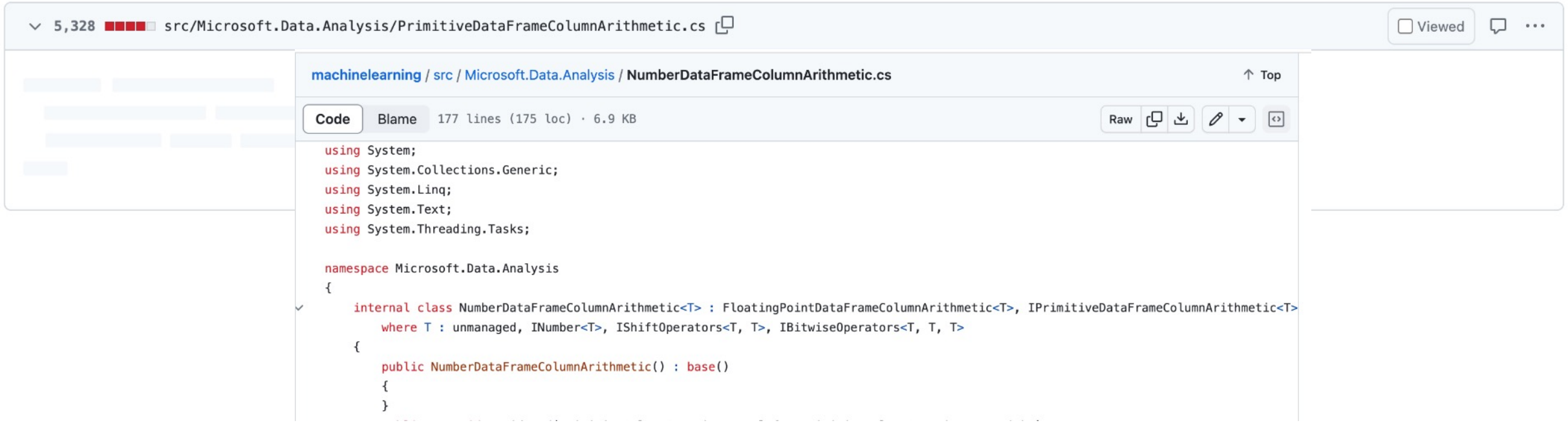


Рефакторинг ML.NET

+2,615 −43,918

Теперь для ML.NET открыта дверь в мир Глубокого Обучения

https://github.com/dotnet/machinelearning/pull/6664



Рефакторинг ML.NET

```
internal interface IPrimitiveDataFrameColumnArithmetic<T>
    where T : unmanaged
{
    void Add(PrimitiveColumnContainer<T> left, PrimitiveColumnContainer<T> right);
    void Add(PrimitiveColumnContainer<T> column, T scalar);
    void Add(T scalar, PrimitiveColumnContainer<T> column);
    void Subtract(PrimitiveColumnContainer<T> left, PrimitiveColumnContainer<T> right);
    void Subtract(PrimitiveColumnContainer<T> column, T scalar);
    void Subtract(T scalar, PrimitiveColumnContainer<T> column);
    void Multiply(PrimitiveColumnContainer<T> left, PrimitiveColumnContainer<T> right);
    void Multiply(PrimitiveColumnContainer<T> column, T scalar);
    void Multiply(T scalar, PrimitiveColumnContainer<T> column);
    void Divide(PrimitiveColumnContainer<T> left, PrimitiveColumnContainer<T> right);
    void Divide(PrimitiveColumnContainer<T> column, T scalar);
    void Divide(T scalar, PrimitiveColumnContainer<T> column);
```

Рефакторинг ML.NET

```
internal class BoolArithmetic : IPrimitiveDataFrameColumnArithmetic<bool>
{

    public void Add(PrimitiveColumnContainer<bool> left, PrimitiveColumnContainer<bool> right)
    {
        throw new NotSupportedException();
    }

    public void Add(PrimitiveColumnContainer<bool> column, bool scalar)
    {
        throw new NotSupportedException();
    }

    public void Add(bool scalar, PrimitiveColumnContainer<bool> column)
    {
        throw new NotSupportedException();
    }
```

Рефакторинг ML.NET

```
else if (typeof(T) == typeof(byte))
{
    return (IPrimitiveDataFrameColumnArithmetic<T>)new ByteArithmetic();
    return (IPrimitiveDataFrameColumnArithmetic<T>)new NumberDataFrameColumnArithmetic<byte>();
}
else if (typeof(T) == typeof(char))
{
    return (IPrimitiveDataFrameColumnArithmetic<T>)new CharArithmetic();
    return (IPrimitiveDataFrameColumnArithmetic<T>)new NumberDataFrameColumnArithmetic<char>();
}
else if (typeof(T) == typeof(decimal))
{
    return (IPrimitiveDataFrameColumnArithmetic<T>)new DecimalArithmetic();
    return (IPrimitiveDataFrameColumnArithmetic<T>)new FloatingPointDataFrameColumnArithmetic<decimal>();
}
else if (typeof(T) == typeof(double))
{
    return (IPrimitiveDataFrameColumnArithmetic<T>)new DoubleArithmetic();
    return (IPrimitiveDataFrameColumnArithmetic<T>)new FloatingPointDataFrameColumnArithmetic<double>();
}
else if (typeof(T) == typeof(float))
{
    return (IPrimitiveDataFrameColumnArithmetic<T>)new FloatArithmetic();
    return (IPrimitiveDataFrameColumnArithmetic<T>)new FloatingPointDataFrameColumnArithmetic<float>();
}
else if (typeof(T) == typeof(int))
{
    return (IPrimitiveDataFrameColumnArithmetic<T>)new IntArithmetic();
    return (IPrimitiveDataFrameColumnArithmetic<T>)new NumberDataFrameColumnArithmetic<int>();
}
```

Рефакторинг ML.NET

```
internal class FloatingPointDataFrameColumnArithmetic<T> : IPrimitiveDataFrameColumnArithmetic<T>
    where T : unmanaged, INumber<T>
{
    public FloatingPointDataFrameColumnArithmetic() : base()
    {
    }
```

```
internal class NumberDataFrameColumnArithmetic<T> : FloatingPointDataFrameColumnArithmetic<T>, IPrimitiveDataFrameColumnArithmetic<T>
    where T : unmanaged, INumber<T>, IShiftOperators<T, T>, IBitwiseOperators<T, T, T>
{
    public NumberDataFrameColumnArithmetic() : base()
    {
    }
```

Плюсы и минусы

Плюсы

Решена старая насущная проблема

Мощный инструмент для продвинутого проектирования

Нет просадки в перфомансе (пруф позже)

Минусы

Использовать GM можно только в рамках типа полного цикла (свой контракт – своя реализация)

Нельзя перенести ответственность за реализацию на класс

Первый минус

- А что если я хочу сделать так?

```
interface IAdditive<TAdditive> :
    IAdditiveIdentity<TAdditive, TAdditive>,
    IAdditionOperators<TAdditive, TAdditive, TAdditive>
    where TAdditive : IAdditive<TAdditive>;
```

Первый минус

- Взяли пользовательский тип данных

```
class AdditiveString(string s) : IAdditive<AdditiveString>
{
    private readonly string _string = s;

    public static AdditiveString AdditiveIdentity => new(string.Empty);

    public static AdditiveString operator +(
        AdditiveString left,
        AdditiveString right) =>
        new(left._string + right._string);
}
```

Первый минус

- Но теперь забудьте про встроенные типы данных

```
IAdditive<int>
```

The type 'int' must be convertible to 'IAdditive<int>' in order to use it as parameter 'TAdditive' in the generic interface 'IAdditive<TAdditive>'

```
public readonly struct Int32
  : IComparable,
    ISpanFormattable,
    IUtf8SpanFormattable,
    IConvertible,
    IComparable<int>,
    IEquatable<int>,
    IMinMaxValue<int>,
    IBinaryInteger<int>,
    ISignedNumber<int>
 in namespace System
 in assembly System.Runtime (System.Runtime.dll)
```

Represents a 32-bit signed integer.

`Int32` on docs.microsoft.com ↗

Первый минус

- Временные решения – писать обёртки или
- большие constraint'ы
- Решение, которое могло бы быть – implicit interface
- https://github.com/dotnet/roslyn/issues/2146
- Начали в 2015, заглохло в 2020

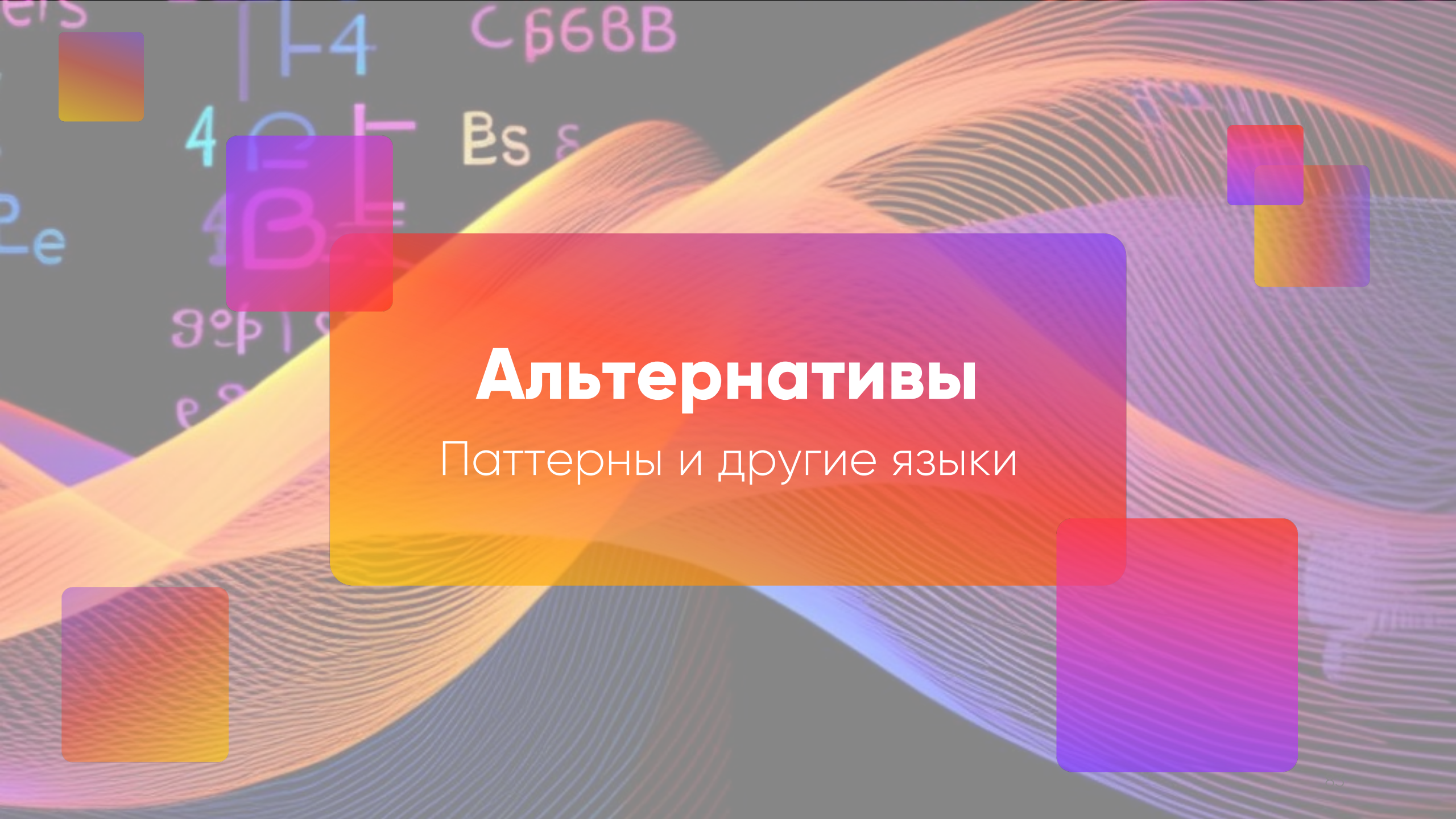
```
implicit interface IAdditive<TAdditive> :
    IAdditiveIdentity<TAdditive, TAdditive>,
    IAdditionOperators<TAdditive, TAdditive, TAdditive>
    where TAdditive : IAdditive<TAdditive>;
```

Второй минус

- Нельзя сделать так

```
abstract class AdditiveBase<TAdditive> : IAdditive<TAdditive>
    where TAdditive : IAdditive<TAdditive>
{
    public abstract static TAdditive AdditiveIdentity { get; }

    public abstract static TAdditive operator +(
        TAdditive left,
        TAdditive right);
}
```


Альтернативы
Паттерны и другие языки

Три вида полиморфизма

ЗАДАЧА

создать принтер, распечатывающий несколько видов объектов

оказывается, что у неё есть несколько решений

Три вида полиморфизма

- Полиморфизм подтипов

```
interface IPrintable
{
    string Content { get; }
}

interface IPrinter
{
    void Print(IPrintable printable);
}
```

Три вида полиморфизма

- Параметрический полиморфизм

```
interface IPrinter<in T>
{
    void Print(T item);
}
```

Три вида полиморфизма

- Ad-hoc полиморфизм

```
interface IPrinter
{
    void Print(int i);

    void Print(string s);

    void Print(bool b);
}
```

Ad-hoc полиморфизм

- Ad-hoc полиморфизм – привязка к конкретному типу данных
- В зависимости от типа используются разные реализации
- Перегрузка методов – один из примеров такого полиморфизма

Классы типов

- Как заставить компилятор продолжать помогать и создать один универсальный контракт вместо множества перегрузок?
- Отделить операцию от данных

```
interface IPrintable<T>
{
    string GetContent(T item);
}
```

Классы типов

```
class Printer
{
    public void Print<T, TPrintable>(T item)
        where TPrintable : struct, IPrintable<T>
    {
        var content = default(TPrintable).GetContent(item);
        Console.WriteLine(content);
    }
}
```

Классы типов

```
struct PrintableInt : IPrintable<int>
{
    public string GetContent(int item) =>
        item.ToString();
}

struct PrintableBool : IPrintable<bool>
{
    public string GetContent(bool item) =>
        item ? "true" : "false";
}
```

Пробуем решить боль дотнет разработчиков

```
interface IAdder<T>
{
    T Zero { get; }

    T Plus(T left, T right);
}


public static T SumTypeClass<T, TAdder>(
    this IReadOnlyList<T> array,
    TAdder adder = default)
    where TAdder : struct, IAdder<T>
{
    var result = adder.Zero;
    var count = array.Count;
    for (var i = 0; i < count; i++)
        result = adder.Plus(result, array[i]);
    return result;
}
```

Пробуем решить боль дотнет разработчиков

```
int[] array = [1, 2, 3];
var sum = array.SumTypeClass<int, IntAdder>();

struct IntAdder : IAdder<int>
{
    public int Zero => 0;

    public int Plus(int left, int right) =>
        left + right;
}
```

Как быстро это работает?

- Что будем сравнивать?

```
public static T SumTypeClass<T, TAdder>(
    this IReadOnlyList<T> array,
    TAdder adder = default)
    where TAdder : struct, IAdder<T>
{
    var result:T = adder.Zero;
    var count:int = array.Count;
    for (var i = 0; i < count; i++)
        result = adder.Plus(left:result, right:array[i]);
    return result;
}
```

```
public static T SumGenericMath<T>(
    this IReadOnlyList<T> array) where T :
    IAdditiveIdentity<T, T>,
    IAdditionOperators<T, T, T>
{
    var result:T = T.AdditiveIdentity;
    var count:int = array.Count;
    for (var i = 0; i < count; i++)
        result += array[i];
    return result;
}
```

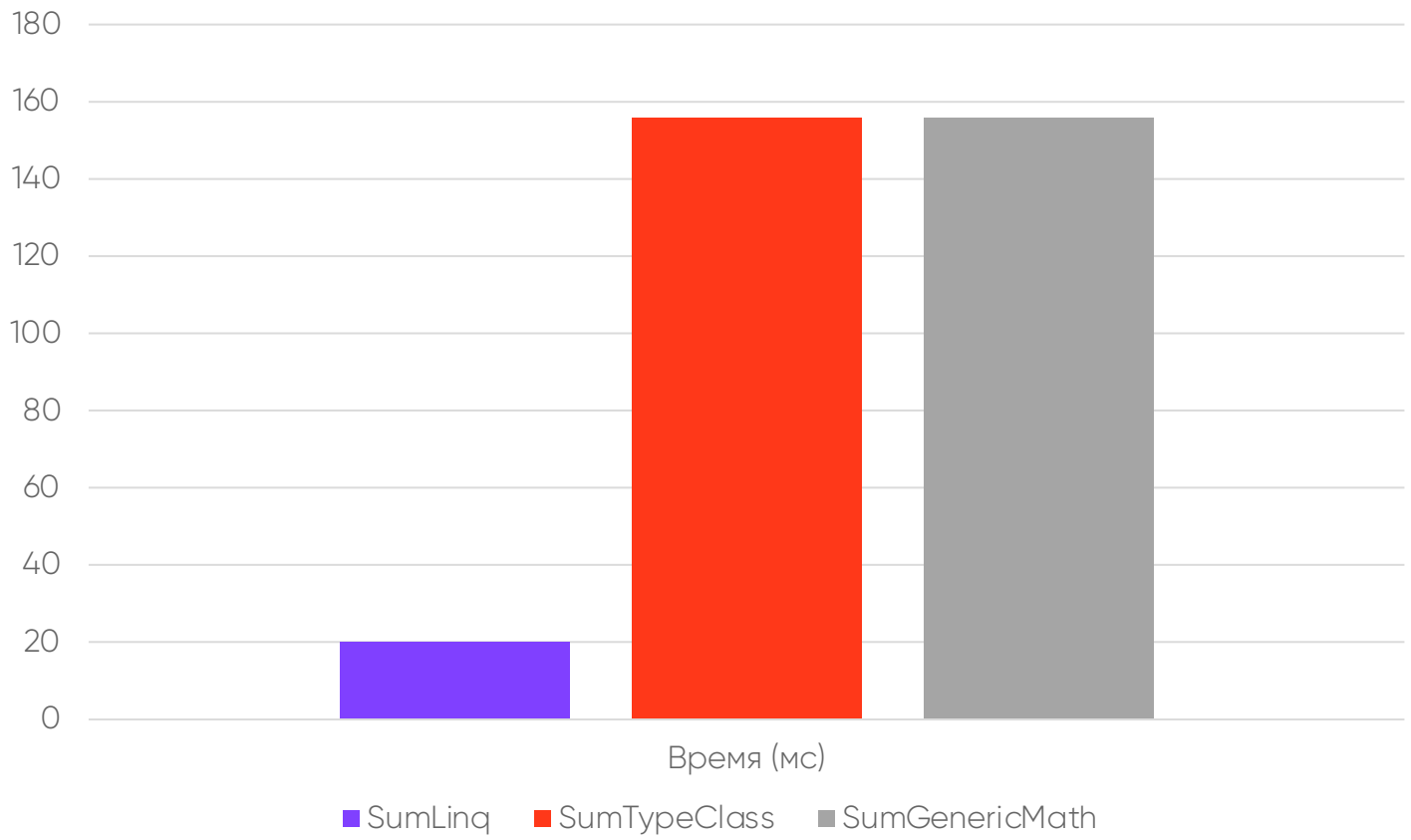
Как быстро это работает?

Конфигурация бенчмарка:

- BenchmarkDotNet v0.13.12
- macOS Monterey 12.3
- Apple M1 Pro, 1 CPU, 10 logical and 10 physical cores
- .NET SDK 8.0.100

Как быстро это работает?

Результат бенчмарка:



Как быстро это работает?

Результат бенчмарка:

```
private static TResult Sum<TSource, TResult>(this
IEnumerable<TSource> source)
  where TSource : struct, INumber<TSource>
  where TResult : struct, INumber<TResult>
{
  ReadOnlySpan<TSource> span;
  if (source.TryGetSpan<TSource>(out span))
    return Enumerable.Sum<TSource, TResult>(span);
  TResult zero = TResult.Zero;
  foreach (TSource source1 in source)
    checked { zero +=
TResult.CreateChecked<TSource>(source1); }
  return zero;
}
```

Как быстро это работает?

Результат бенчмарка:

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
private static bool TryGetSpan<TSource>(
  this IEnumerable<TSource> source,
  out ReadOnlySpan<TSource> span)
  where TSource : struct
{
  if (source == null)
    ThrowHelper.ThrowArgumentNullException(ExceptionArgument.source);
  bool span1 = true;
  if (source.GetType() == typeof (TSource[]))
    span = (ReadOnlySpan<TSource>) Unsafe.As<TSource[]>((object) source);
  else if (source.GetType() == typeof (List<TSource>))
  {
    span = (ReadOnlySpan<TSource>) CollectionsMarshal.AsSpan<TSource>(Unsafe.As<List<TSource>>((object) source));
  }
  else
  {
    span = new ReadOnlySpan<TSource>();
    span1 = false;
  }
  return span1;
}
```

Как быстро это работает?

Итоги бенчмарка:

- .NET 8 поднял производительность LINQ
- Используйте по возможности

  `T[]` или `List<T>`
- Если не можете – обобщайте с помощью GM

Пробуем решить боль дотнет разработчиков

- Почему так?

```
int[] array = [1, 2, 3];
var sum = array.SumTypeClass<int, IntAdder>();

struct IntAdder : IAdder<int>
{
    public int Zero => 0;

    public int Plus(int left, int right) =>
        left + right;
}
```

Как работает вывод типов в C#?

- Через анализ входящего потока данных

```
var id = Guid.NewGuid();
var dateTask = Task.FromResult(DateTime.Now);
var strings = new List<int> { 1, 2, 3 }
    .Select(x => x.ToString())
    .ToList();
```

Будут ли в C# классы типов?

- 2017: Concept C#: Type Classes for the Masses
- https://github.com/dotnet/csharplang/issues/110
- https://github.com/MattWindsor91/roslyn/blob/master/concepts/docs/csconcepts.md

C# concept

```
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);
}
```

C# instance

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

C# implicit type argument

- Неявный формальный типовой параметр

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

Вдохновение от Scala

- Не лучшая идея)

```scala
trait Comparator[A] {
  def compare(x: A, y: A): Int
}

object Comparator {
  implicit object IntComparator extends Comparator[Int] {
    def compare(x: Int, y: Int): Int = Integer.compare(x, y)
  }

  implicit object StringComparator extends Comparator[String] {
    def compare(x: String, y: String): Int = x.compareTo(y)
  }
}

def max[A](x: A, y: A)(implicit comparator: Comparator[A]): A =
  if (comparator.compare(x, y) >= 0) x
  else y

println(max(10, 6))             // 10
println(max("hello", "world"))  // world
```

Что ещё?

- C++ concepts
- Scala implicits
- Rust traits
- Swift protocols

Но не C# или F#...

Haskell практически идеал

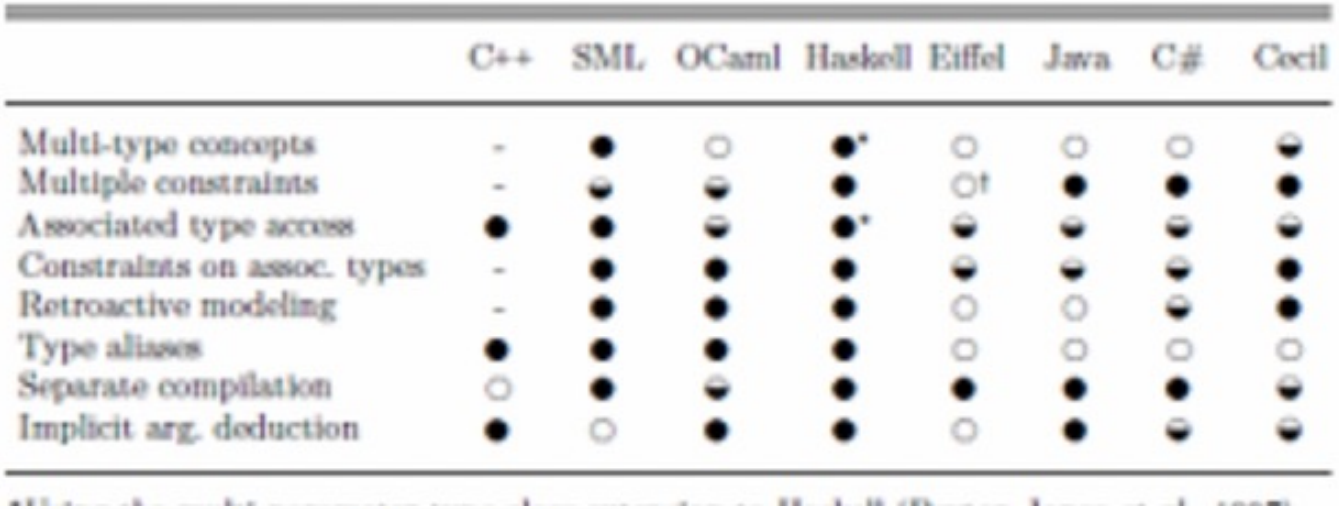
An Extended Comparative Study of Language Support for Generic Programming

RONALD GARCIA[1], JAAKKO JÄRVI[2], ANDREW LUMSDAINE[1]

JEREMY SIEK[3], JEREMIAH WILLCOCK[1]

[1] *Open Systems Lab, Indiana University, Bloomington, IN USA*
[2] *Texas A&M University, Computer Science, College Station, TX USA*
[3] *Rice University, Computer Science, Houston, TX USA*
(*e-mail:* {garcia,lums,jewillco}@osl.iu.edu, jeremy.g.siek@rice.edu, jarvi@cs.tamu.ed

Haskell практически идеал



	C++	SML	OCaml	Haskell	Eiffel	Java	C#	Cecil
Multi-type concepts	-	●	○	●*	○	○	○	◒
Multiple constraints	-	◒	◒	●	○†	●	●	●
Associated type access	●	●	◒	●*	◒	◒	◒	◒
Constraints on assoc. types	-	●	●	●	◒	◒	◒	●
Retroactive modeling	-	●	●	●	○	○	◒	●
Type aliases	●	●	●	●	○	○	○	○
Separate compilation	○	●	◒	●	●	●	●	◒
Implicit arg. deduction	●	○	●	●	○	●	◒	◒

*Using the multi-parameter type class extension to Haskell (Peyton Jones *et al.*, 1997).
*Using the functional dependencies extension to Haskell (Jones, 2000).
†Planned language additions.
Table 1: *The level of support for important properties for generic programming in the evaluated languages. A black circle indicates full support, a white circle indicates poor support, and a half-filled circle indicates partial support. The rating*

An Extended Comparative Study of Language Support for Generic Programming

RONALD GARCIA[1], JAAKKO JÄRVI[2], ANDREW LUMSDAINE[1]
JEREMY SIEK[3], JEREMIAH WILLCOCK[1]
[1] *Open Systems Lab, Indiana University, Bloomington, IN USA*
[2] *Texas A&M University, Computer Science, College Station, TX USA*
[3] *Rice University, Computer Science, Houston, TX USA*
(*e-mail:* {garcia,lums,jewillco}@osl.iu.edu, jeremy.g.siek@rice.edu, jarvi@cs.tamu.ed

We also identify eight language properties that support this broader view of generic programming: support for *multi-type concepts*, multiple constraints on type parameters, convenient *associated type access*, *constraints on associated types*, *retroactive modeling*, type aliases, *separate compilation* of algorithms and data structures, and *implicit argument type deduction* for generic algorithms. We find that these features are necessary to avoid awkward designs, poor maintainability, and painfully verbose code.

Many
type-s
and in
paper
progra
(with i
in each of these languages, we illustrate how the basic roles of generic programming can be represented in each language. We also identify eight language properties that support this broader view of generic programming: support for multi-type concepts, multiple constraints on type parameters, convenient associated type access, constraints on associated types, retroactive modeling, type aliases, separate compilation of algorithms and data structures, and implicit argument type deduction for generic algorithms. We find that these features are necessary to avoid awkward designs, poor maintainability, and painfully verbose code. As languages increasingly support generics, it is important that language designers understand the features necessary to enable the effective use of generics and that their absence can cause difficulties for programmers.

Haskell vs C#

```haskell
class Eq a where
    (==) :: a -> a -> Bool

(==) :: (Eq a) => a -> a ->
Bool
```

```csharp
interface Eq<A>
{
    bool Equal(A a, A b);
}

static class Overloads
{
    public static bool
Equals<A, EqA>(A a, A b)
        where EqA : struct,
Eq<A> =>

default(EqA).Equal(a, b);
}
```

Больше про ad-hoc полиморфизм

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

README.md

language-ext

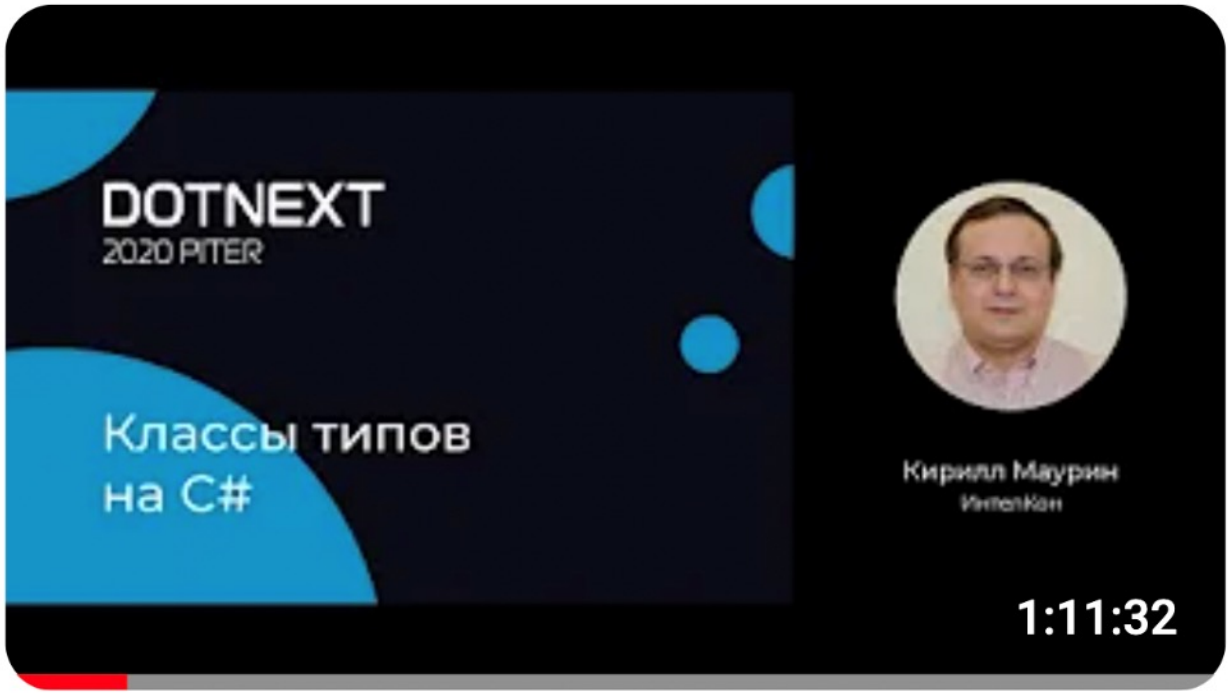
C# Functional Programming Language Extensions

This library uses and abuses the features of C# to provide a functional-programming 'base class library' that, if you squint, can look like extensions to the language itself. The desire here is to make programming in C# much more reliable and to make the engineer's inertia flow in the direction of declarative and functional code rather than imperative.

https://habr.com/ru/companies/ruvds/articles/757118

Больше про классы типов

https://www.youtube.com/watch?v=W64C3FsxjXE



Кирилл Маурин — Классы типов на C#

3,9 тыс. просмотров • 3 года назад

Альтернатива от Swift

- protocol

```
protocol Animal {
    var maxBabiesCount: Int { get }

    func getSound() -> String
}

class Cat : Animal {
    let maxBabiesCount: Int = 5

    func getSound() -> String {
        "Meow"
    }
}
```

Альтернатива от Swift

- associatedtype

```
protocol Animal {
    associatedtype BabyType
    var babies: [BabyType] { get set }
}

class Cat : Animal {
    var babies = [Kitten]()
}

class Kitten { }
```

Альтернатива от Swift

- Self

```
protocol Animal {
    func mate(with: Self)
}

class Cat : Animal {
    func mate(with: Cat) {
        print("mating with another cat")
    }
}
```

Альтернатива от Swift

https://github.com/apple/swift-evolution/blob/main/proposals/0233-additive-arithmetic-protocol.md

```
public protocol AdditiveArithmetic : Equatable {
  static var zero: Self { get }

  static func + (lhs: Self, rhs: Self) -> Self
}
```

Альтернатива от Swift

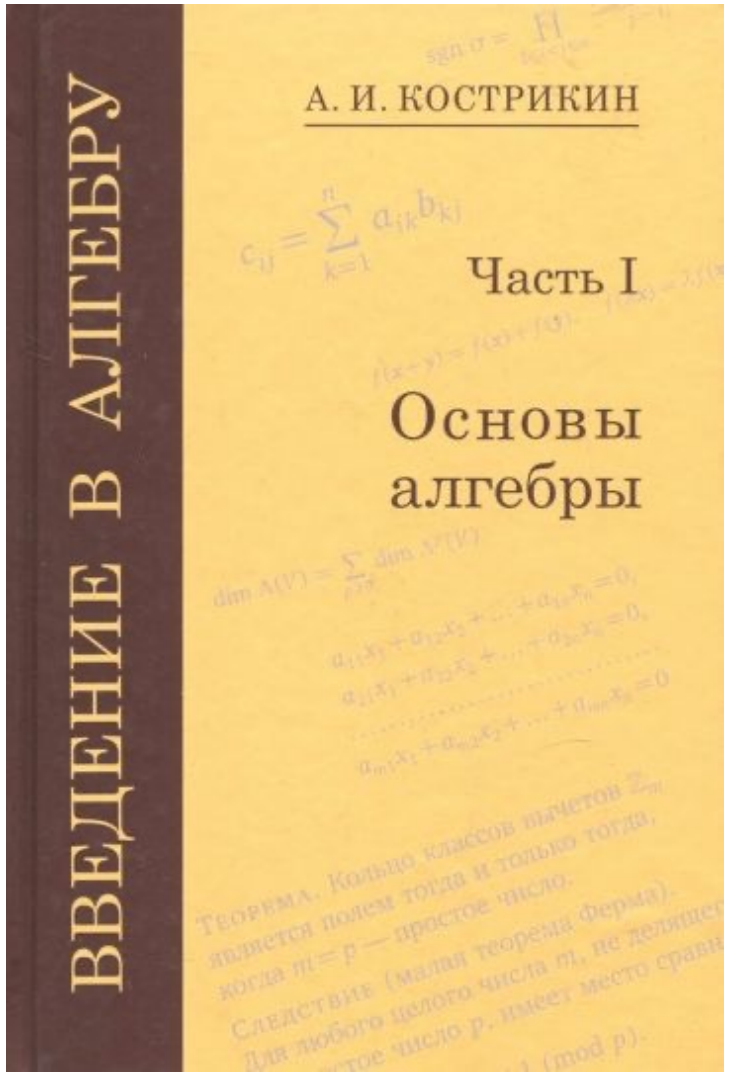
- Swift Numerics

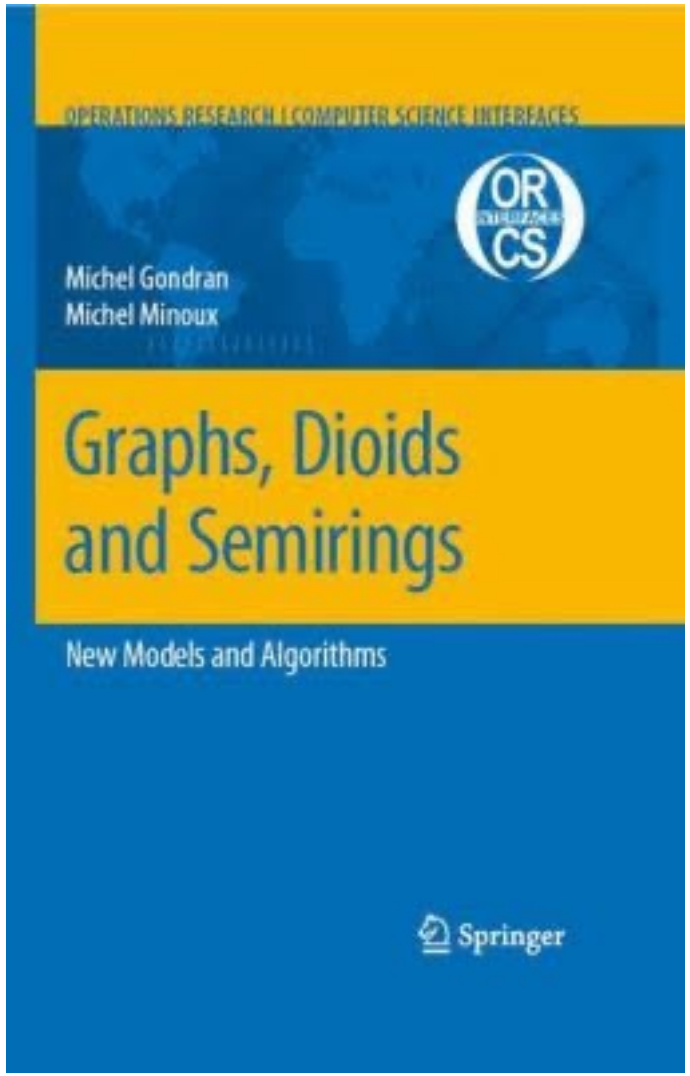
Итоги

- Что, как и для чего мы узнали
- Язык двигается вперёд
- Открылось много новых возможностей

Post Scriptum

Что почитать





Что почитать

https://habr.com/ru/articles/655059

Stefanio 10 мар 2022 в 22:44

Абстрактная алгебра в действии

6 мин 26K

Программирование*, .NET*, C#*, Математика*, Функциональное программирование*

https://habr.com/ru/articles/656919

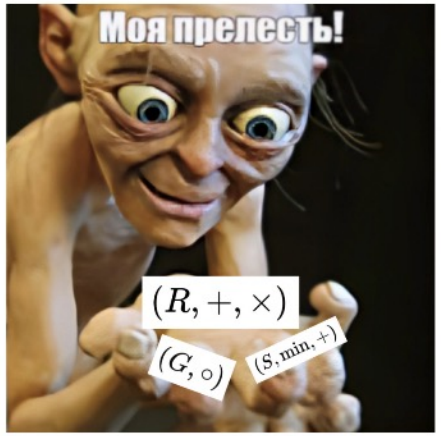
Stefanio 8 апр 2022 в 20:30

Властелин структур

15 мин 10K

Программирование*, .NET*, C#*, Математика*, Функциональное программирование*

Технотекст 2021



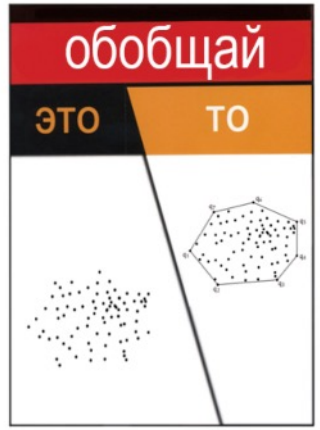
https://habr.com/ru/companies/ruvds/articles/726368

Stefanio 25 апр 2023 в 12:00

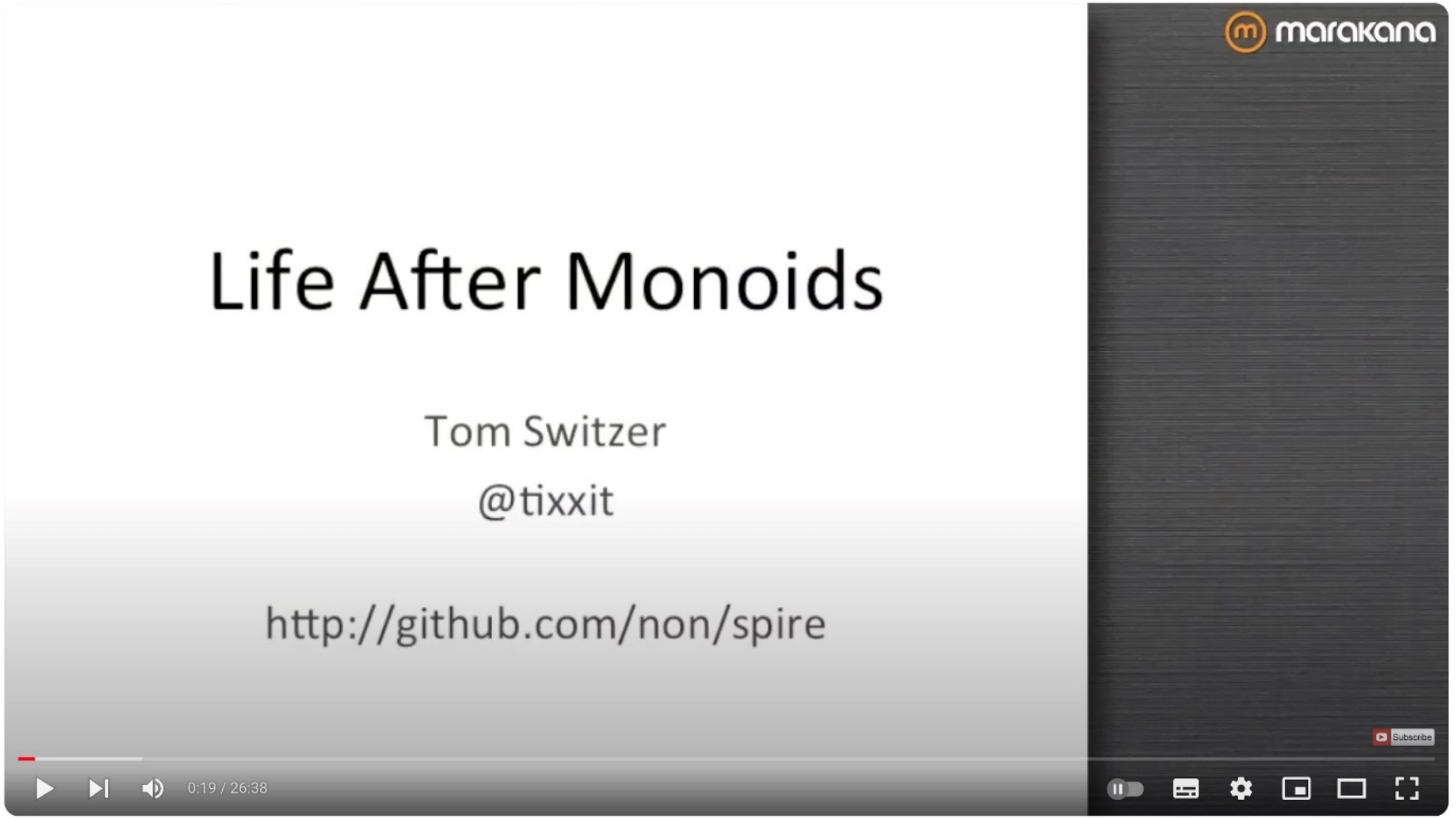
Обобщай это, обобщай то

Средний 7 мин 6.1K

Блог компании RUVDS.com, .NET*, C#*, Математика*, Функциональное программирование*



Что посмотреть



Life After Monoids

InfoQ
229 тыс. подписчиков
Подписаться
80
Поделиться
Создать клип
Сохранить

Что посмотреть

"Add ALL the things: abstract algebra meets analytics" by Avi Bryant (2013)

Strange Loop Conference
81 тыс. подписчиков
Подписаться
Нравится
Поделиться
Создать клип

Что посмотреть

- https://github.com/twitter/algebird
- https://www.michael-noll.com/blog/2013/12/02/twitter-algebird-monoid-monad-for-large-scala-data-analytics/

Спасибо за внимание!

Контакты

Степан Минин

Senior C#

https://t.me/steponeit

https://youtube.com/@steponeit

https://habr.com/ru/users/Stefanio/

https://github.com/Stepami