



LINQ Expressions: искусство запрашивать данные

Денис Цветчих | Тинькофф

Денис Цветцих

Тинькофф, DevBrothers



cRud



80% запросов - чтение



Запросы **частично** дублируются



А еще запросы меняются при изменении требований



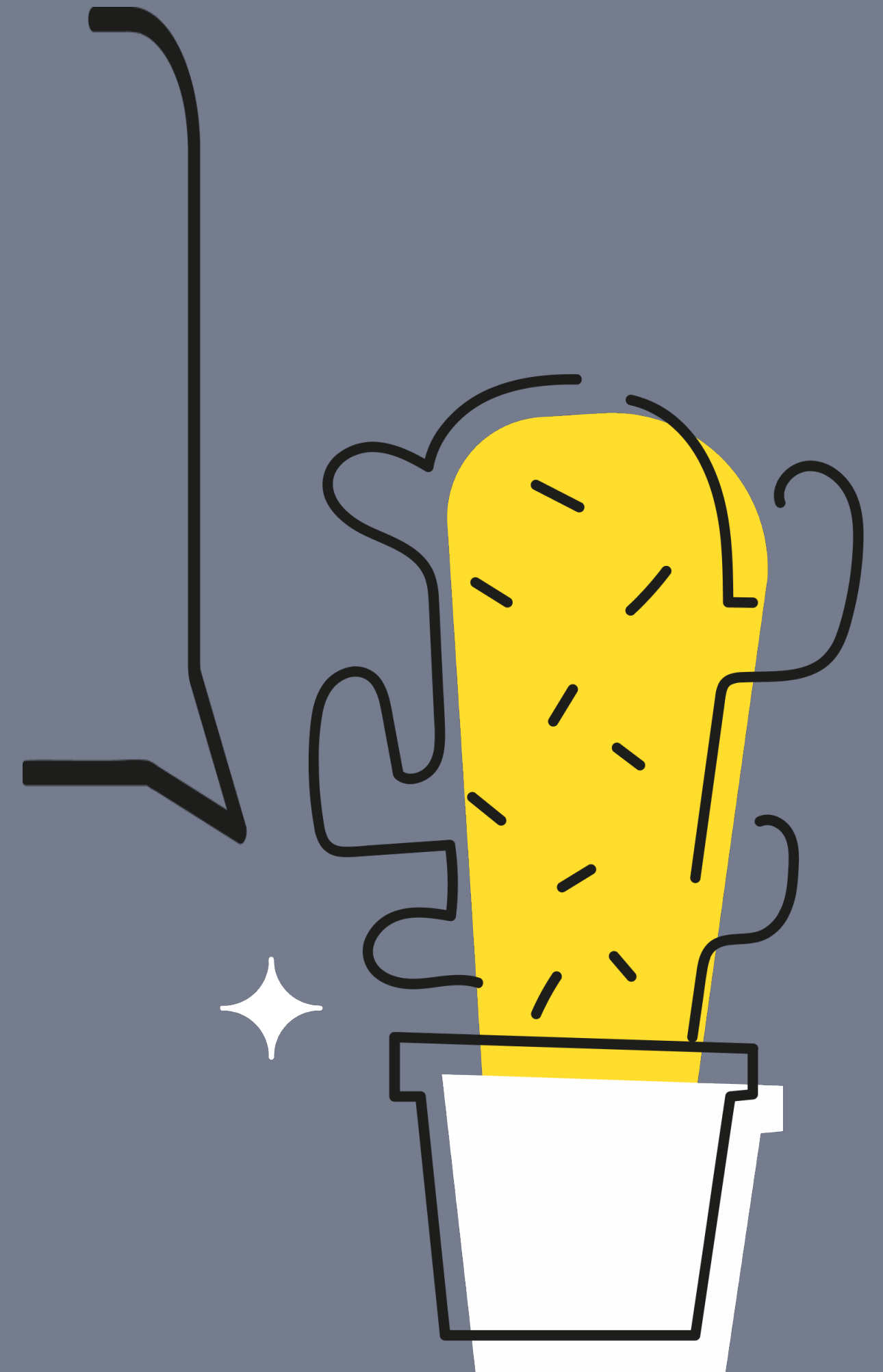
И много рутины для формирования запросов



GraphQL, OData, Json Api – не серебряная пуля

- Что такое спецификация
- Современная реализация спецификации
- Фильтрация вложенных коллекций
- Как упростить рутину генерации запросов

О чем поговорим



Интернет-магазин

```
public class Product
{
    public int Id { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }

    public bool IsSaleEnabled { get; set; }
}
```

СПИСОК ТОВАРОВ

```
var products = dbContext.Products
    .Where(x => x.IsSaleEnabled)
    .ToList();
```

УЧИТЫВАЕМ СКЛАД

```
public class Product
{
    public int Id { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }

    public bool IsSaleEnabled { get; set; }
    public int StockCount { get; set; }
}
```

Список товаров с учетом склада

```
var products = dbContext.Products
    .Where(x => x.IsSaleEnabled && x.StockCount > 0)
    .ToList();
```


Инкапсулируем

```
public class Product
{
    public int Id { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }

    public bool IsSaleEnabled { get; set; }
    public int StockCount { get; set; }

    public bool IsAvailable => IsSaleEnabled && StockCount > 0;
}
```

Extension method

```
public static class Queries
{
    public static IQueryable<Product> Available(this IQueryable<Product> query)
        => query.Where(x => x.IsSaleEnabled && x.StockCount > 0);
}
```

```
var products = dbContext.Products
    .Available()
    .ToList();
```

Группируем по И

```
var products = dbContext.Products  
    .Available()  
    .HasDiscount()  
    .ToList();
```



Спецификация

Представляет бизнес-правила в виде цепочки объектов, связанных бизнес-логикой

01

Инкапсуляция правил

02

Комбинация правил

Спецификация

```
public interface ISpecification
{
    bool IsSatisfiedBy(object candidate);
}
```

```
Func<T, bool>
```

Делегат не работает

```
public static class Queries
{
    public static bool IsAvailable(Product product)
        => product.IsSaleEnabled && product.StockCount > 0;
}

var products = dbContext.Products
    .Where(x => Queries.IsAvailable(x))
    .ToList();
```

Фильтр в памяти – не вариант

```
var products = dbContext.Products
    .AsEnumerable()
    .Where(x => Queries.IsAvailable(x))
    .ToList();
```

Спецификация как выражение

`Func<T, bool>` - IEnumerable

`Expression<Func<T, bool>>` - IQueryable

`Func<T, bool>` \Rightarrow `Expression<Func<T, bool>>` ?

Expression первичен, делегат вторичен

```
Func<Product, bool> delegate = expression.Compile();
```

CompileFast: 10-40 раз быстрее

```
Func<Product, bool> delegate = expression.CompileFast();
```



<https://github.com/dadhi/FastExpressionCompiler>

Delegate Decompiler (0.32)

```
public class Product
{
    public int Id { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }

    public bool IsSaleEnabled { get; set; }
    public int StockCount { get; set; }

    [Computed]
    public bool IsAvailable => IsSaleEnabled && StockCount > 0;
}
```



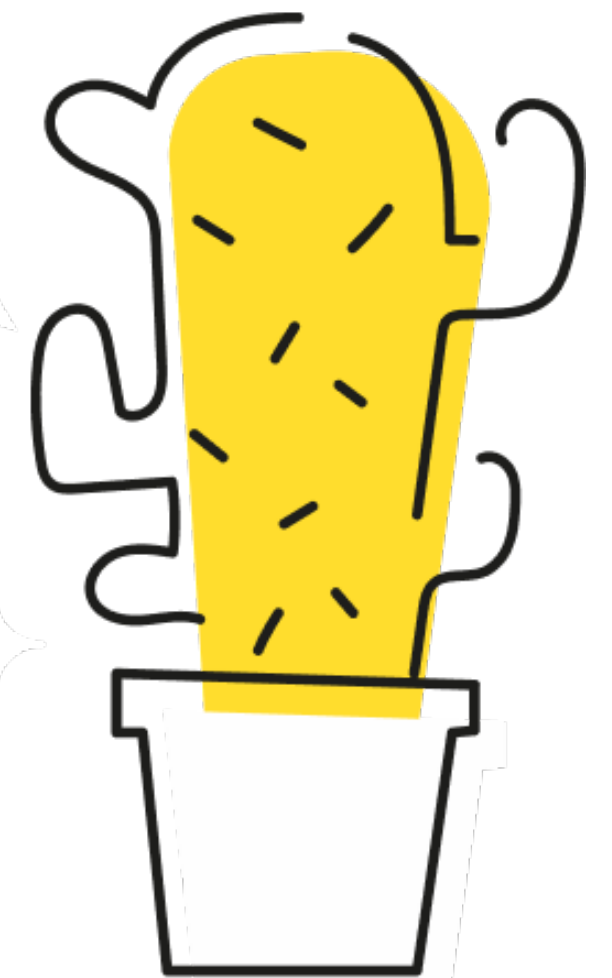
<https://github.com/hazzik/DelegateDecompiler>

Декоратор Queryable

```
var products = dbContext.Products
    .Where(x => x.IsAvailable)
    .Decompile()
    .ToList();
```

А нельзя побыстрее
и поменьше магии?

/else



Linq Spec

```
public abstract class Specification<T>
{
    public abstract Expression<Func<T, bool>> ToExpression();

    public static implicit operator Expression<Func<T, bool>>(Specification<T> spec)
        => spec?.ToExpression() ?? throw new ArgumentNullException(nameof(spec));
}
```



<https://github.com/navozenko/LinqSpecs>

Спецификация → LinqExpression

```
public class IsProductAvailableSpecification : Specification<Product>
{
    public override Expression<Func<Product, bool>> ToExpression()
        => x => x.IsSaleEnabled && x.StockCount > 0;
}
```

```
public class Product
{
    public static readonly Specification<Product> IsAvailable
        = new IsProductAvailableSpecification();
}
```

```
var products = dbContext.Products
    .Where(Product.IsAvailable)
    .ToList();
```

AdHocSpecification без отдельного класса

```
public class Product
{
    public static readonly Specification<Product> HasDiscount =
        new AdHocSpecification<Product>(x => x.IsDiscount && x.Price > 100);
}

var products = dbContext.Products
    .Where(Product.IsAvailable || Product.HasDiscount)
    .ToList();
```

Объединение Expression

```
Expression<Func<Product, bool>> saleEnabled = x => x.IsSaleEnabled;
```

```
Expression<Func<Product, bool>> onStock = y => y.StockCount > 0;
```

parameter body

```
var isAvailableBody = Expression.And(saleEnabled.Body, onStock.Body);
```

```
// x.IsSaleEnabled And y.StockCount > 0
```

```
var isAvailable = Expression.Lambda<Func<Product, bool>>(isAvailableBody, parameter);
```

System.InvalidOperationException: 'The LINQ expression 'x' could not be translated

Подмена параметра

```
public class ReplaceVisitor : ExpressionVisitor
{
    private readonly Expression _from, _to;
    public ReplaceVisitor(ParameterExpression from, ParameterExpression to)
    {
        _from = from;
        _to = to;
    }

    protected override Expression VisitParameter(ParameterExpression node)
    {
        return node == _from ? _to : base.Visit(node);
    }
}
```


Делаем один параметр

```
var saleParam = saleEnabled.Parameters.Single(); // x
var stockParam = onStock.Parameters.Single(); // y

var replace = new ReplaceVisitor(saleParam, stockParam);
var fixedSaleEnabled = replace.Visit(saleEnabled.Body)!; // x.IsSaleEnabled -> y.IsSaleEnabled

var isAvailableBody = Expression.And(fixedSaleEnabled, onStock.Body);
// y.IsSaleEnabled && y.StockCount > 0

var isAvailable = Expression.Lambda<Func<Product, bool>>(isAvailableBody, stockParam);
// y => y.IsSaleEnabled && y.StockCount > 0
```

PredicateBuilder (LINQKit)

```
Expression<Func<Product, bool>> saleEnabled = x => x.IsSaleEnabled;  
  
Expression<Func<Product, bool>> onStock    = y => y.StockCount > 0;  
  
var combined = saleEnabled.And(onStock); //Or, Not
```

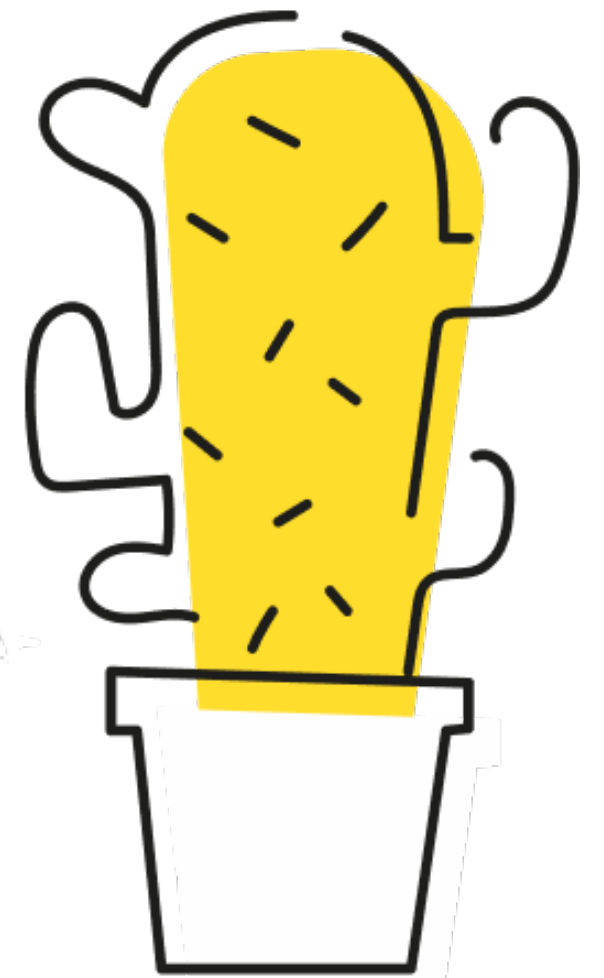


Спецификация не дружит с выражением

```
var products = dbContext.Products
    .Where(x => Product.IsAvailable || x.Price > 0) // Compilation error
    .ToList();
```

Дублирующихся выражений
немного, заворачивать абсолютно
все в спеки не хочется

А можно как-то гибче?



NeinLinq

```
public static class Ext
{
    [InjectLambda]
    public static bool IsAvailable(this Product product)
        => throw new NotImplementedException();

    public static Expression<Func<Product, bool>> IsAvailable()
        => p => p.IsSaleEnabled && p.StockCount > 0;
}
```



<https://github.com/axelheer/nein-linq>

Снова декоратор Queryable

```
var products = dbContext.Products
    .ToInjectable()
    .Where(x => x.IsAvailable() && x.Price > 0)
    .ToList();
```

Не только агрегаты, но и значения

```
public static class Ext
{
    [InjectLambda]
    public static bool IsPositive(this decimal value)
        => value > 0;

    public static Expression<Func<decimal, bool>> IsPositive()
        => (p) => p > 0;
}

var products = dbContext.Products
    .Where(x => x.IsAvailable() && x.Price.IsPositive())
    .ToList();
```

Настройка декоратора

```
services.AddDbContext<AppDbContext>(options =>  
    options.UseSqlOrTheLike("...")  
        .WithLambdaInjection());
```

```
var products = dbContext.Products  
    .ToInjectable()  
    .Where(x => x.IsAvailable() && x.Price > 0)  
    .ToList();
```

Projectable

```
services.AddDbContext<AppDbContext>(options => options  
    .UseSqlServer("").UseProjectables());
```



<https://github.com/koenbeuk/EntityFrameworkCore.Projectables>

СВОЙСТВО

```
public class Product
{
    [Projectable]
    public bool IsAvailable => IsSaleEnabled && StockCount > 0;
}

var products = dbContext.Products
    .Where(x => x.IsAvailable)
    .ToList();
```

Extension метод

```
public static class Ext
{
    [Projectable]
    public static bool IsAvailable(this Product p)
        => p.IsSaleEnabled && p.StockCount > 0;
}
```

```
var products = dbContext.Products
    .Where(x => x.IsAvailable())
    .ToList();
```

Projectable

А можно
без декоратора?

/else



01

Source Generator для LINQ

02

Декоратор для замены
выражения

EF Core 7: IQueryableExpressionInterceptor

```
public class SpecificationInterceptor : IQueryableExpressionInterceptor
{
    public Expression QueryCompilationStarting(Expression queryExpression, QueryExpressionEventData eventData)
        => new SpecificationVisitor().Visit(queryExpression);

    private class SpecificationVisitor : ExpressionVisitor
    {
        protected override Expression VisitMethodCall(MethodCallExpression? methodCallExpression)
        {
            var methodInfo = methodCallExpression!.Method;
            if (methodInfo.Name != nameof(Ext.IsAvailable))
                return base.VisitMethodCall(methodCallExpression);

            var argument = methodCallExpression.Arguments.First();
            return Expression.Property(argument, nameof(Product.IsSaleEnabled));
        }
    }
}
```



linq2db

Все в одном



<https://github.com/linq2db/linq2db>



LINQ to DB is
type-safe SQL



Нет ChangeTracking



Документация
не очень

Спецификация – Extension method

```
[ExpressionMethod("IsAvailable")]  
public static bool IsAvailable(this Product p)  
    => p.IsSaleEnabled && p.StockCount > 0;  
  
public static Expression<Func<Product, bool>> IsAvailable()  
    => (p) => p.IsSaleEnabled && p.StockCount > 0;  
  
var res = db.Product.Where(x => x.IsAvailable())  
    .ToList();
```



<http://blog.linq2db.com/2016/06/how-to-teach-linq-to-db-convert-custom.html>

Спецификация – свойство класса

```
public class Product
{
    [ExpressionMethod("IsAvailableExpr")]
    public bool IsAvailable => IsSaleEnabled && StockCount > 0;

    private static Expression<Func<Product, bool>> IsAvailableExpr()
        => (p) => p.IsSaleEnabled && p.StockCount > 0;
}

var res = db.Product.Where(x => x.IsAvailable)
    .ToList();
```

eShopOnWeb - Ardalis.Specification

```
public interface ISpecification<T>
{
    bool IsSatisfiedBy(T entity);

    IEnumerable<WhereExpressionInfo<T>> WhereExpressions { get; }
    IEnumerable<OrderExpressionInfo<T>> OrderExpressions { get; }
    IEnumerable<IncludeExpressionInfo> IncludeExpressions { get; }

    int? Take { get; }
    int? Skip { get; }

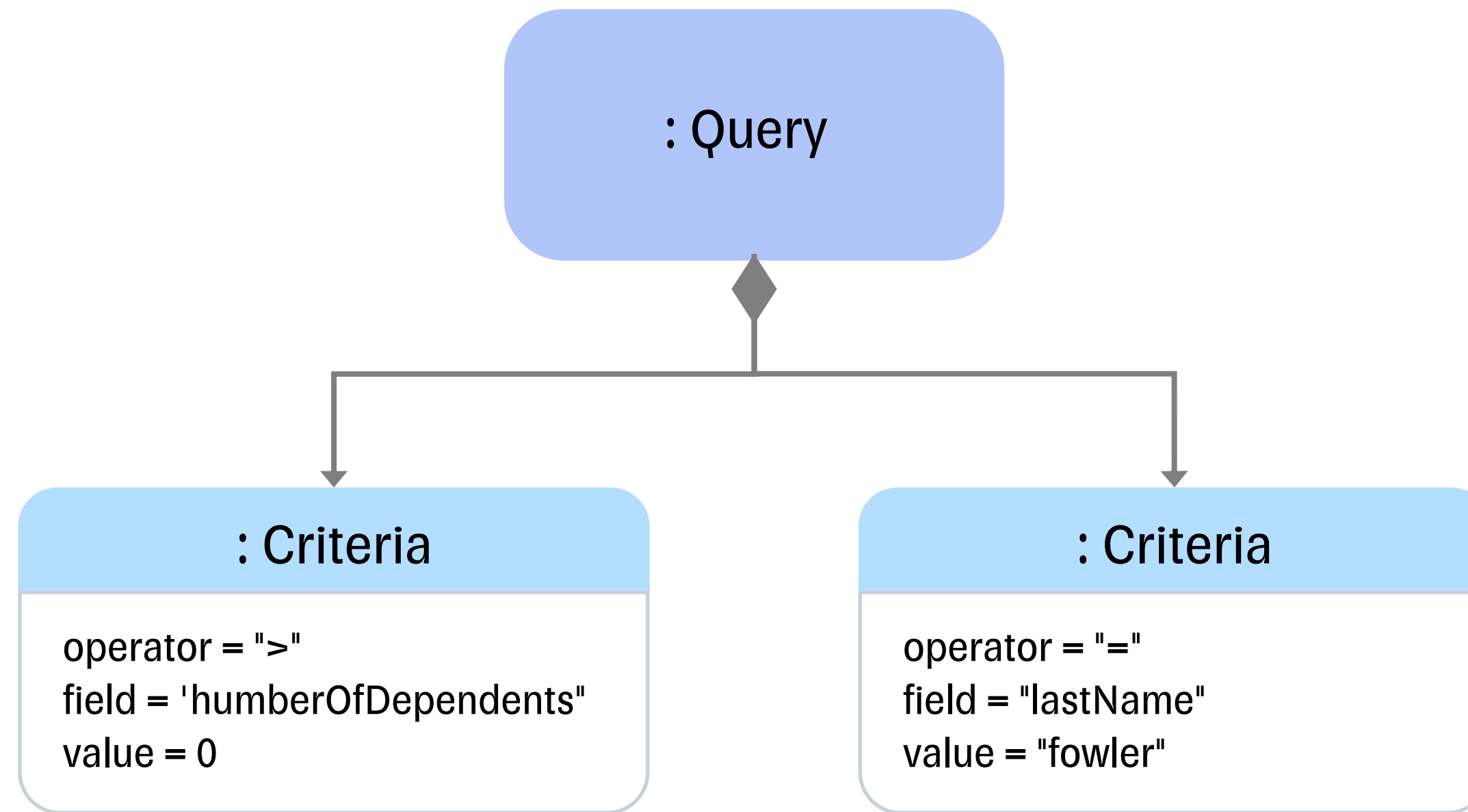
    bool CacheEnabled { get; }
    string? CacheKey { get; }

    bool AsNoTracking { get; }
    bool AsSplitQuery { get; }
    bool AsNoTrackingWithIdentityResolution { get; }
    bool IgnoreQueryFilters { get; }
}
```



<https://github.com/ardalis/Specification>

Query Object



<https://www.martinfowler.com/eaCatalog/queryObject.html>

Ardalis.Specification.EntityFrameworkCore

```
public sealed class ProductsAvailableSpecification : Specification<Product>
{
    public ProductsAvailableSpecification()
    {
        Query.Where(x => x.IsSaleEnabled && x.StockCount > 0);
    }
}
```

```
var products = dbContext.Products
    .WithSpecification(new ProductsAvailableSpecification())
    .ToList();
```

QueryObject нельзя комбинировать

```
queryObject.Take = 20 && 30;
```

```
queryObject.OrderBy = "FirstName DESC" || "Date ASC";
```

QueryObject vs Specification

```
var products = dbContext.Products
    .WithQueryObject(new QueryObject())
    .ToList();
```

```
var products = dbContext.Products
    .WithSpecification(new Specification())
    .OrderBy(x => x.Name)
    .Skip(pageNumber * pageSize).Take(pageSize)
    .ToList();
```

Dynamic Linq

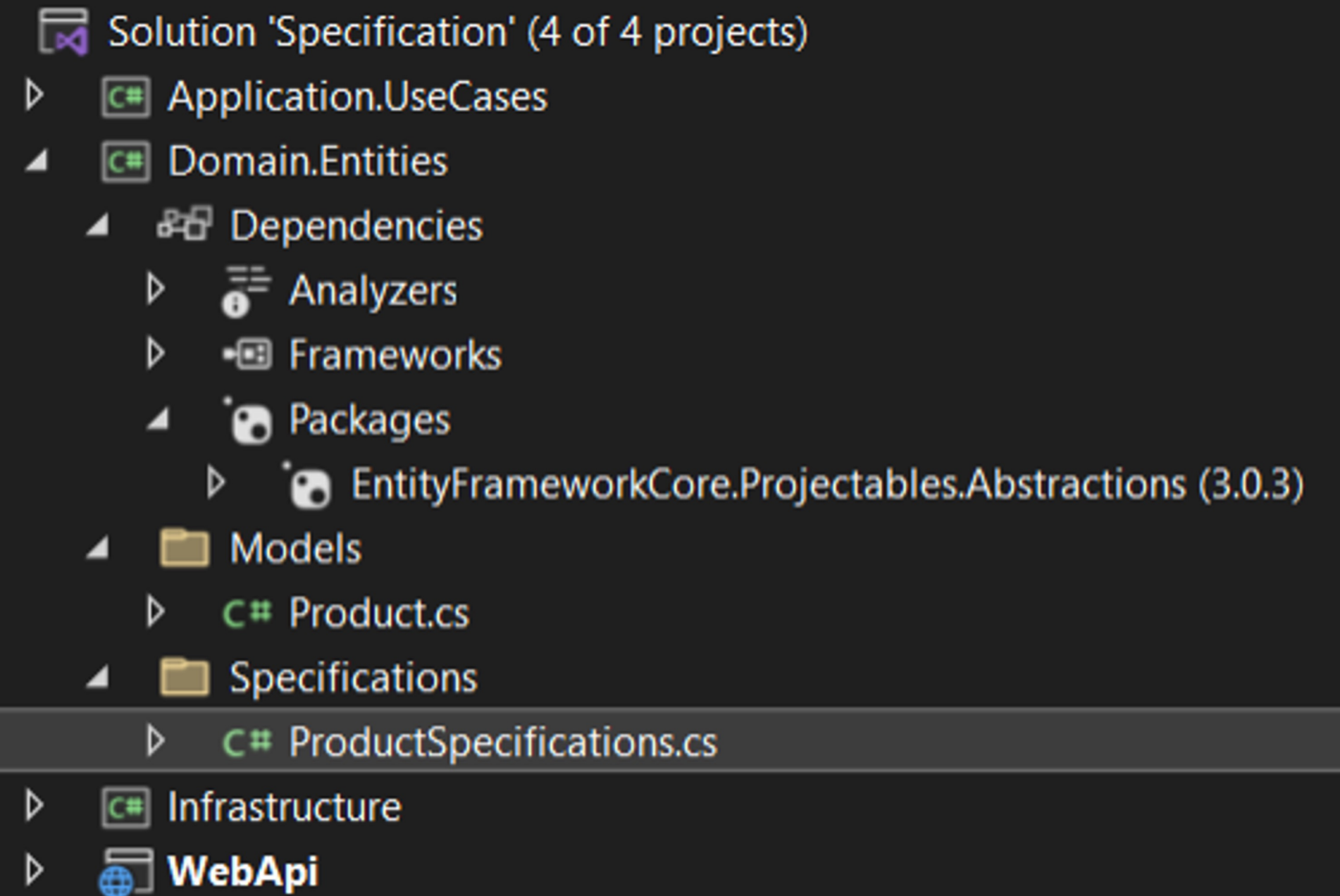
```
Expression<Func<Product, bool>> saleEnabled = x => x.IsSaleEnabled;  
Expression<Func<Product, bool>> onStock = x => x.StockCount > 0;
```

```
var products = dbContext.Products  
    .Where("@0(it) and @1(it)", saleEnabled, onStock)  
    .ToList();
```



<https://github.com/zzzprojects/System.Linq.Dynamic.Core>

Спецификации в Domain.Entities



**фильтрация
по вложенным
коллекциям**



В категории много товаров

```
public class Product
{
    public int CategoryId { get; set; }
    public Category Category { get; set; }
}

public class Category
{
    public int Id { get; set; }

    public string Name { get; set; }

    public List<Product> Products { get; set; }
}
```


Фильтрация вложенных коллекций

```
var categories = dbContext.Categories
    .Where(x => x.Products.Any(p => p.IsSaleEnabled && p.StockCount > 0))
    .ToList();
```

LINQ-спецификация не подходит

```
Expression<Func<Product, bool>> specification =  
    x => x.IsSaleEnabled && x.StockCount > 0;
```

```
var categories = dbContext.Categories  
    .Where(x => x.Products.Any(specification)) // compilation error  
    .ToList();
```

LINQKit Core

```
Expression<Func<Product, bool>> specification =  
    x => x.IsSaleEnabled && x.StockCount > 0;  
  
var categories = dbContext.Categories  
    .AsExpandable()  
    .Where(x => x.Products.Any(specification.Compile()))  
    .ToList();
```



<https://github.com/scottksmith95/LINQKit>

Нет компиляции спецификации, но...

```
private Expression TryVisitExpressionFunc(MemberExpression input, FieldInfo field)
{
    var propertyInfo = input.Member as PropertyInfo;
    if (field.FieldType.GetTypeInfo().IsSubclassOf(typeof(Expression)) || propertyInfo != null && propertyInfo.F
    {
        return Visit(Expression.Lambda<Func<Expression>>(input).Compile());
    }

    return input;
}
```

Projectable спешит на помощь

```
public class Product
{
    [Projectable]
    public bool IsAvailable => IsSaleEnabled && StockCount > 0;
}

var categories = dbContext.Categories
    .Where(x => x.Products.Any(p => p.IsAvailable))
    .ToList();
```

Extension-метод тоже работает

```
public static class Ext
{
    [Projectable]
    public static bool IsAvailable(this Product p)
        => p.IsSaleEnabled && p.StockCount > 0;
}

var categories = dbContext.Categories
    .Where(x => x.Products.Any(p => p.IsAvailable()))
    .ToList();
```

Рутина фильтрации

ОПЕРАЦИИ

Текущий квартал ▾ Все проекты ▾ Все контрагенты ▾ 🔍

Плановые Фактические Все Приходы Расходы Переводы Все

+ Приход - Расход 📄 Импорт 📄 Экспорт ⚙️

Рутина филтрации

```
public class Filter
{
    public string? Name { get; set; }
    public decimal? PriceFrom { get; set; }
    public decimal? PriceTo { get; set; }
}
```


Рутина филтрации

```
var filter = request.Filter;

IQueryable<Product> query = dbContext.Products;

if (filter.Name != null)
    query = query.Where(x => x.Name.Contains(filter.Name));
if (filter.PriceFrom != null)
    query = query.Where(x => x.Price >= filter.PriceFrom);
if (filter.PriceTo != null)
    query = query.Where(x => x.Price <= filter.PriceTo);

var products = query.ToList();
```

Sieve

```
public class Filter
{
    [Sieve(CanFilter = true, CanSort = true)]
    public string? Name { get; set; }
}

_sieveProcessor.Apply(sieveModel, products); //IQueryable
```



<https://github.com/Biarity/Sieve>

SieveModel

```
?sorts= Price  
&filters= Price<100, Name@=some name,  
&page= 1,  
&pageSize= 10
```

```
public class SieveModel  
{  
    public string Filters { get; set; }  
  
    public string Sorts { get; set; }  
  
    public int? Page { get; set; }  
  
    public int? PageSize { get; set; }  
}
```

Autofilterer

```
public class ProductFilter : FilterBase
{
    public Range<double>? Price { get; set; }

    [ToLowerContainsComparison]
    public string? Name { get; set; }
}

var products = dbContext.Products.ApplyFilter(filter).ToList();
```



<https://github.com/enisn/AutoFilterer>

AutoFilter.Sql

```
public class Filter
{
    [FilterProperty(IgnoreCase = true, StringFilter = StringFilterCondition.Contains)]
    public string? Name { get; set; }

    public Range<decimal>? PriceFrom { get; set; }
}

var products = dbContext.Products.AutoFilter(filter).ToList();
```



<https://github.com/denis-tsv/AutoFilter>

Под капотом AutoFilter

```
public static IQueryable<TItem> AutoFilter<TItem>(this IQueryable<TItem> items, object filter)
{
    var parameter = Expression.Parameter(typeof(TItem), "x");

    var propertyExpressions = filter.GetType().GetProperties()
        .Select(x => new { Name = x.Name, Value = x.GetValue(filter) })
        .Where(x => x.Value != null)
        .Select(x =>
        {
            var property = Expression.Property(parameter, x.Name);
            Expression value = Expression.Constant(x.Value, property.Type);

            return Expression.Equal(property, value);
        })
        .ToList();

    var body = propertyExpressions.Aggregate(Expression.AndAlso);

    var result = Expression.Lambda<Func<TItem, bool>>(body, parameter);

    return items.Where(result);
}
```

Оптимизируем через компиляцию выражений

```
public static IQueryable<TItem> AutoFilter<TItem>(this IQueryable<TItem> items, object filter)
{
    var parameter = Expression.Parameter(typeof(TItem), "x");

    var propertyExpressions = filter.GetType().GetProperties()
        .Select(x => new { Name = x.Name, ValueGetter = CreateValueGetter(filter, x) })
        .Select(x => new { Name = x.Name, Value = x.ValueGetter.Invoke(filter) })
        .Where(x => x.Value != null)
        .Select(x =>
        {
            var property = Expression.Property(parameter, x.Name);
            Expression value = Expression.Constant(x.Value, property.Type);

            return Expression.Equal(property, value);
        })
        .ToList();

    var body = propertyExpressions.Aggregate(Expression.AndAlso);

    var result = Expression.Lambda<Func<TItem, bool>>(body, parameter);

    return items.Where(result);
}
```

Компилируем выражение в делегат

```
public static Func<Filter, object> CreateValueGetter(Filter filter, PropertyInfo x)
{
    var parameter = Expression.Parameter(typeof(Filter), "x");
    Expression body = Expression.Property(parameter, x);
    body = Expression.Convert(body, typeof(object));

    var lambda = Expression.Lambda<Func<Filter, object>>(body, parameter);
    return lambda.Compile();
}
```



Доклад про сериализацию с бенчмарками <https://youtu.be/DhzT8Abpsso>

Подмена параметра и компиляция выражения

```
BenchmarkDotNet=v0.13.5, OS=Windows 10 (10.0.19044.2965/21H2/November2021Update)
AMD Ryzen 7 3700X, 1 CPU, 16 logical and 8 physical cores
.NET SDK=7.0.202
[Host] : .NET 7.0.4 (7.0.423.11508), X64 RyuJIT AVX2
DefaultJob : .NET 7.0.4 (7.0.423.11508), X64 RyuJIT AVX2
```

Method	Mean	Error	StdDev
RebindParameter	370.6 ns	4.34 ns	4.06 ns
RebindParameter_PredicateBuilder	340.1 ns	3.58 ns	3.18 ns
CompileFast	4,223.3 ns	96.39 ns	284.21 ns
Compile	67,990.3 ns	514.81 ns	456.36 ns

Автофилтр

```
BenchmarkDotNet=v0.13.5, OS=Windows 10 (10.0.19044.2965/21H2/November2021Update)
AMD Ryzen 7 3700X, 1 CPU, 16 logical and 8 physical cores
.NET SDK=7.0.202
[Host]      : .NET 7.0.4 (7.0.423.11508), X64 RyuJIT AVX2
DefaultJob : .NET 7.0.4 (7.0.423.11508), X64 RyuJIT AVX2
```

Method	Mean	Error	StdDev
Reflection	2.476 us	0.0440 us	0.0411 us
Delegates	2.417 us	0.0315 us	0.0295 us
AutoFilterSql	2.742 us	0.0284 us	0.0266 us
AutoFilterer	4.272 us	0.0421 us	0.0394 us

→ Современная спецификация – extension метод + сорсгены (Projectable)

→ Подходит для фильтрации вложенных коллекций (Projectable)

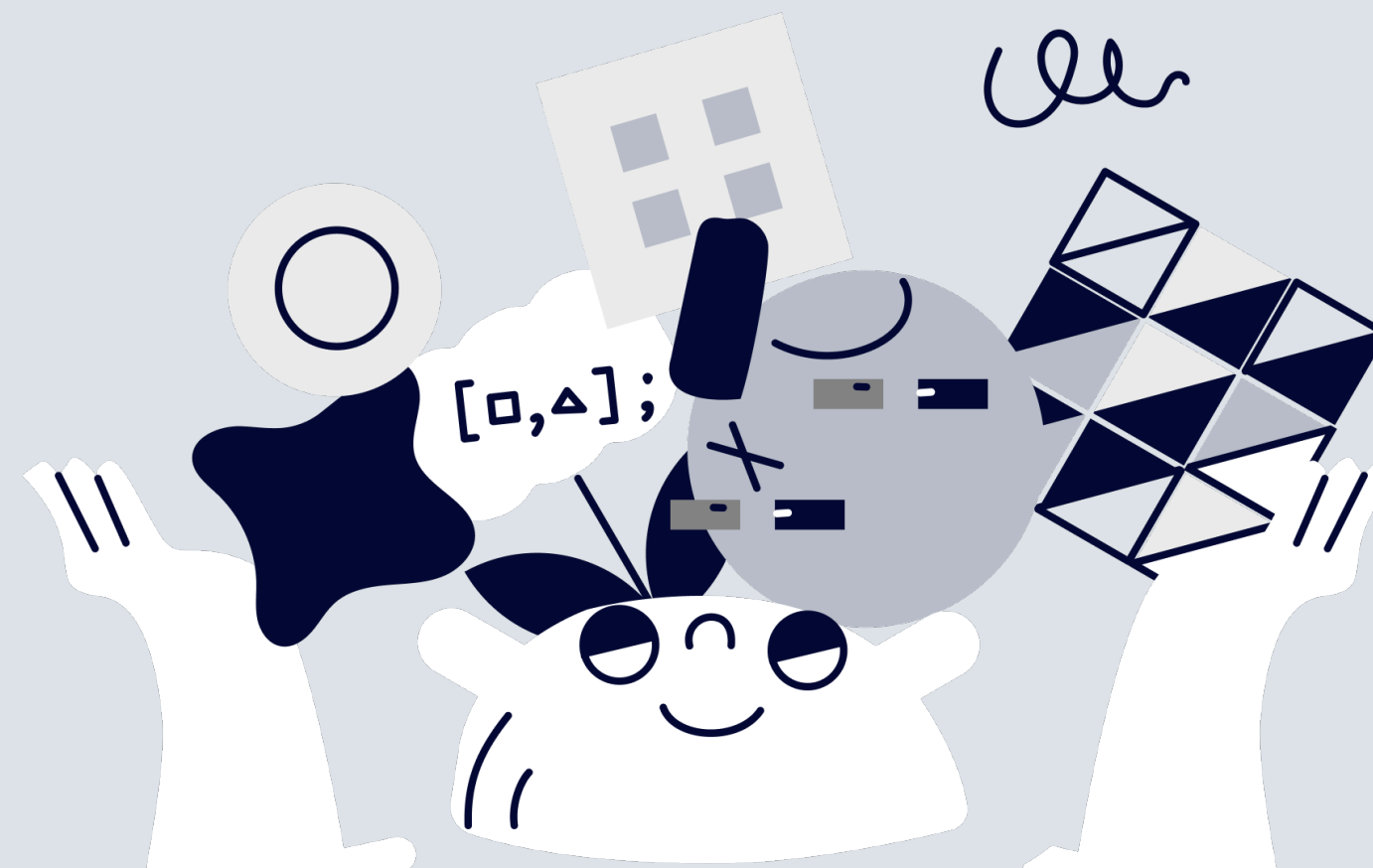
→ Автофильтр для автоматизации рутины фильтрации

→ Компиляция Expression в делегат – доли процента от работы запроса



О чем поговорили

Главная мысль



Expression, сорсгены,
интерсепторы – не просто
хайп, а полезные инструменты!



Но надо знать когда
их применять 😊

Полезные ССЫЛКИ



EF Core Extensions

<https://learn.microsoft.com/en-us/ef/core/extensions/>



EntityFramework: (анти)паттерн Repository

<https://habr.com/ru/articles/335856/>



Максим Аршинов. Деревья выражений
в Enterprise разработке (2018)

<https://youtu.be/J2XzsCoJM4o>





Денис Цветцих



den.tsvettsih@yandex.ru



@den_tsvettsikh



<https://github.com/denis-tsv>

**Спасибо
за внимание!
Есть вопросы?**