Деревья выражений в enterprise-разработке

Максим Аршинов, HighTech

max@hightech.today

DotNext Piter 2018

История одного рефакторинга

Весь код вымышлен, любые совпадения случайны

Жил был интернет магазин

```
public class Product
{
    public string Name { get; set; }

    public decimal Price { get; set}

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

Получаем товары только для продажи

```
var products = _dbContext.Products
    .Where(x => x.IsForSale)
    .ToList();
```

Бизнес-правила изменчивы

Добавим свойство InStock

```
public class Product
{
    public string Name { get; set; }

    public decimal Price { get; set}

    public int InStock { get; set}

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

Добавим свойство IsAvailable

```
public class Product
{
    public string Name { get; set; }

    public decimal Price { get; set}

    public int InStock { get; set}

    public bool IsForSale { get; set; }

    public bool IsAvailable => IsForSale && InStock > 0;
}
```

Исправляем LINQ

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

DOH!

Исправляем LINQ

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

Как не дублировать код?

```
Where(x => x.IsForSale && x.InStock > 0)


IsAvailable => IsForSale && InStock > 0;
```

Что скрывает компилятор

```
var products = _dbContext.Products
    .ToList()
    .Where(x => x.IsAvailable) // Func<T,bool>
    .ToList();
```

Что скрывает компилятор

```
var products = _dbContext.Products
    .ToList()
    .Where(x => x.IsAvailable) // Func<T,bool>
    .ToList();
```

```
var products = _dbContext.Products
    .Where(x => x.IsAvailable) // Expression<Func<T,bool>>
    .ToList();
```

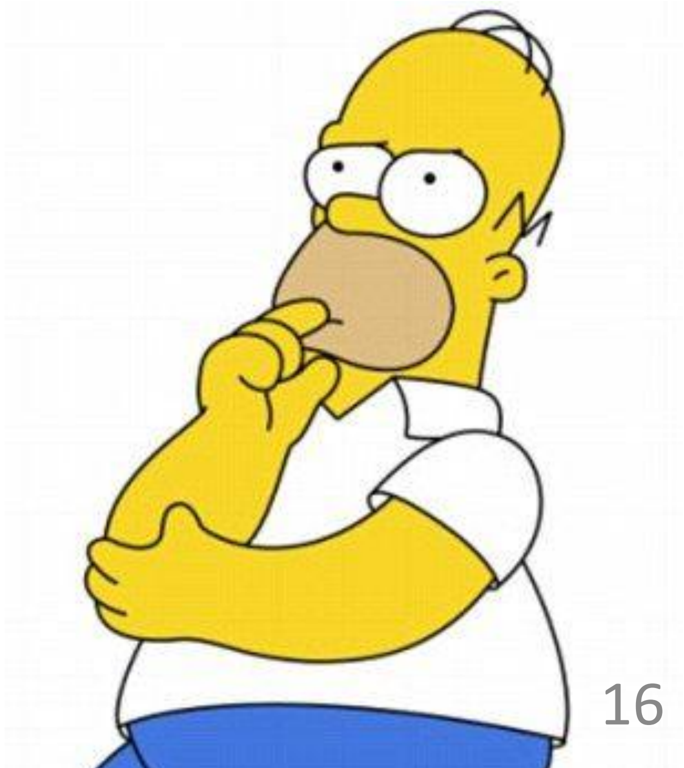
Лямбда-выражения Expression -> Delegate

```
Expression<Func<int, string>> expressionLambda =
    x => x.ToString();

Func<int, string> delegateLambda = expressionLambda.Compile();
```

Лямбда-выражения Delegate -> Expression?

Диалектика лямбда-выражений

- Выражения первичны, делегаты – вторичны?
- Делегаты первичны, выражения – вторичны?



Выражения первичны, делегаты – вторичны

```
// so slo-o-o-o-o-o-o-ow
var delegateLambda = expressionLambda.Compile();
```

Кеширование делегатов

```
internal class CompiledExpressions<TIn, TOut>
{
    private static readonly ConcurrentDictionary<
        Expression<Func<TIn, TOut>>,
        Func<TIn, TOut>> Cache
            = new ConcurrentDictionary<
                Expression<Func<TIn, TOut>>,
                Func<TIn, TOut>>();

    internal static Func<TIn, TOut> AsFunc(Expression<Func<TIn, TOut>> expr)
        => Cache.GetOrAdd(expr, k => k.Compile());
}
```

Кеширование делегатов

```
internal class CompiledExpressions<TIn, TOut>
{
    private static readonly ConcurrentDictionary<
        Expression<Func<TIn, TOut>>,
        Func<TIn, TOut>> Cache
            = new ConcurrentDictionary<
                Expression<Func<TIn, TOut>>,
                Func<TIn, TOut>>();

    internal static Func<TIn, TOut> AsFunc(Expression<Func<TIn, TOut>> expr)
        => Cache.GetOrAdd(expr, k => k.Compile());
}
```

Выражения первичны, делегаты – вторичны

```
public static readonly Expression<Func<Product, bool>>
    IsAvailableExpression = x => x.IsForSale && x.Price > 0;

public bool IsAvailable => IsAvailableExpression.AsFunc()(this);
```

Выражения первичны, делегаты – вторичны

- Microsoft.Linq.Translations
- Signum Framework

Делегаты первичны, выражения – вторичны

- Delegate Decompiler https://habrahabr.ru/post/155437/

Делегаты первичны, выражения – вторичны

- Delegate Decompiler https://habrahabr.ru/post/155437/
- `methodBody.GetILAsByteArray();`

Добавляем атрибут для вычислимых полей

```
public class Product
{
    public string Name { get; set; }

    public decimal Price { get; set}

    public bool IsForSale { get; set; }

    [Computed]
    public bool IsAvailable => IsForSale && Price > 0;
}
```

Декомпилируем делегаты

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

Вернет IQueryable<Product>

Декомпилируем делегаты

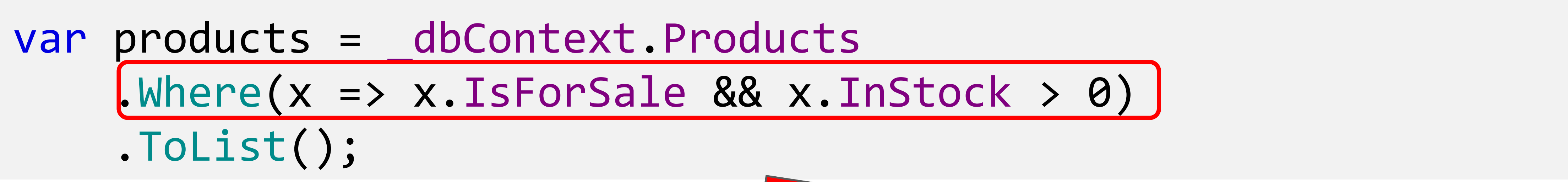
```
var products = _dbContext.Products
    .Where(x => x.IsAvailable)
    .Decompile()
    .ToList();
```

Вернет декоратор, «исправляющий» выражение

Delegate Decompiler 0.23.0 :(

Булевы операции

Булевы операции «И»



```
var products = _dbContext.Products
    .Where(x => x.IsForSale && x.InStock > 0)
    .ToList();
```

Булевы операции «ИЛИ»

```
var products = _dbContext.Products
    .Where(x => x.IsForSale || x.InStock > 0)
    .ToList();
```

Булевы операции «И» с выражениями

```
var products = _dbContext.Products
    .Where(Product.IsForSaleExpression)
    .Where(Product.InStockExpression)
    .ToList();
```

«ИЛИ» с выражениями

Compilation error

```
var products = _dbContext.Products
    .Where(Product.IsForSaleExpression
        || Product.InStockExpression)
    .ToList();
```

Спецификация

Шаблон проектирования, представляющий правила бизнес логики в виде объектов, связанных операциями булевой логики

Популяризованы в DDD

```
public class Spec<T> where T: class
{
    public static bool operator false(Spec<T> spec) => false;

    public static bool operator true(Spec<T> spec) => false;

    public static Spec<T> operator &(Spec<T> spec1, Spec<T> spec2)
        => new Spec<T>(spec1.Expression.And(spec2.Expression));

    public static Spec<T> operator |(Spec<T> spec1, Spec<T> spec2)
        => new Spec<T>(spec1.Expression.Or(spec2.Expression));

    public static Spec<T> operator !(Spec<T> spec)
        => new Spec<T>(spec.Expression.Not());
```

```
public static implicit operator Expression<Func<T, bool>>(Spec<T> spec)
    => spec.Expression;

public static implicit operator Func<T, bool>(Spec<T> spec)
    => spec.IsSatisfiedBy;
```

```
public static implicit operator Expression<Func<T, bool>>(Spec<T> spec)
    => spec.Expression;

public static implicit operator Func<T, bool>(Spec<T> spec)
    => spec.IsSatisfiedBy;
```

```
public Expression<Func<T, bool>> Expression { get; }

public bool IsSatisfiedBy(T obj) => Expression.AsFunc()(obj);
```

Объявим спецификации

```
public static readonly Spec<Product>
    IsForSaleSpec = new Spec<Product>(x => x.IsForSale);

public static readonly Spec<Product>
    IsInStockSpec = new Spec<Product>(x => x.InStock > 0);
```

«ИЛИ» с выражениями

```
var products = _dbContext.Products
    .Where(Product.IsForSaleSpec || Product.IsInStockSpec)
    .ToList();
```

And, or, not

```
public static Spec<T> operator &(Spec<T> spec1, Spec<T> spec2)
    => new Spec<T>(spec1.Expression.And(spec2.Expression));


public static Spec<T> operator |(Spec<T> spec1, Spec<T> spec2)
    => new Spec<T>(spec1.Expression.Or(spec2.Expression));


public static Spec<T> operator !(Spec<T> spec)
    => new Spec<T>(spec.Expression.Not());
```

Это extension-методы

Первый подход к снаряду

```
Expression<Func<int, bool>> e1 = x => x > 5;
Expression<Func<int, bool>> e2 = x => x / 2 == 5;

Expression combined = Expression.OrElse(e1, e2);
var lambda = Expression.Lambda<Func<int, bool>>(combined);
```

DOH!

Второй подход к снаряду

```
Expression<Func<int, bool>> e1 = x => x > 5;
Expression<Func<int, bool>> e2 = x => x / 2 == 5;

Expression combined = Expression.OrElse(e1.Body, e2.Body);
var lambda = Expression.Lambda<Func<int, bool>>(combined);
```

Второй подход к снаряду

```
Expression<Func<int, bool>> e1 = x => x > 5;
Expression<Func<int, bool>> e2 = x => x / 2 == 5;

Expression combined = Expression.OrElse(e1.Body, e2.Body);
var lambda = Expression.Lambda<Func<int, bool>>(combined);
```

Второй подход к снаряду

```
Expression<Func<int, bool>> e1 = x => x > 5;
Expression<Func<int, bool>> e2 = x => x / 2 == 5;

Expression combined = Expression.OrElse(e1.Body, e2.Body);
var lambda = Expression.Lambda<Func<int, bool>>(combined);
```

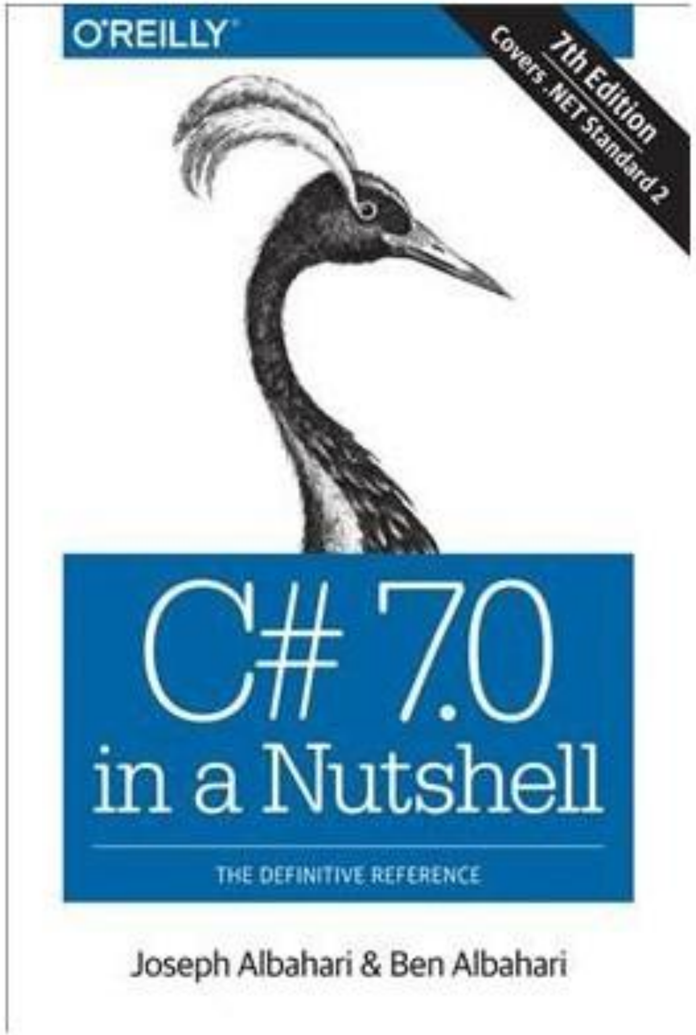
Второй подход к снаряду

```
Expression<Func<int, bool>> e1 = x => x > 5;
Expression<Func<int, bool>> e2 = x => x / 2 == 5;

Expression combined = Expression.OrElse(e1.Body, e2.Body);
var lambda = Expression.Lambda<Func<int, bool>>(combined);
```

Второй подход к снаряду

```
Expression<Func<int, bool>> e1 = x => x > 5;
Expression<Func<int, bool>> e2 = x => x / 2 == 5;

Expression combined = Expression.OrElse(e1.Body, e2.Body);
var lambda = Expression.Lambda<Func<int, bool>>(combined);
```

DOH!



O'REILLY

7th Edition

Covers .NET Standard 2

C# 7.0 in a Nutshell

THE DEFINITIVE REFERENCE

Joseph Albahari & Ben Albahari

```
public static Expression<Func<T, bool>> Or<T> (
    this Expression<Func<T, bool>> expr1, Expression<Func<T, bool>> expr2)
{
    var invokedExpr = Expression.Invoke (expr2,
        expr1.Parameters.Cast<Expression>());

    return Expression.Lambda<Func<T, bool>>(
        Expression.OrElse (expr1.Body, invokedExpr), expr1.Parameters);
}
```

Expression.Invoke

Expression.Invoke

- Creates an InvocationExpression that applies a delegate or lambda expression to a list of argument expressions.

Expression.Invoke

- Creates an InvocationExpression that applies a delegate or lambda expression to a list of argument expressions.
- The interesting work takes place inside the And and Or methods. We start by invoking the second expression with the first expression’s parameters. **An Invoke expression calls another lambda expression using the given expressions as arguments. We can create the conditional expression from the body of the first expression and the invoked version of the second.** The final step is to wrap this in a new lambda expression.

```
public static Expression<Func<T, bool>> And<T> (
    this Expression<Func<T, bool>> expr1, Expression<Func<T, bool>> expr2)
{
    var invokedExpr = Expression.Invoke (expr2,
        expr1.Parameters.Cast<Expression>());

    return Expression.Lambda<Func<T, bool>>(
        Expression.AndAlso (expr1.Body, invokedExpr), expr1.Parameters);
}
```

Снова
Expression.Invoke

AsExpandable

```
var products = _dbContext.Products
    .AsExpandable()
    .Where(x => x.IsForSale && x.InStock > 0)
    .ToList();
```

Подозрительно

LinqKit

```
public string[] QueryCustomers (
    Expression<Func<Purchase, bool>> purchaseCriteria)
{
    var query =
        from c in _dbContext.Customers.AsExpandable()
        // will be stripped by AsExpandable
        where c.Purchases.Any (purchaseCriteria.Compile())
        select c.Name;

    return query.ToArray();
}
```

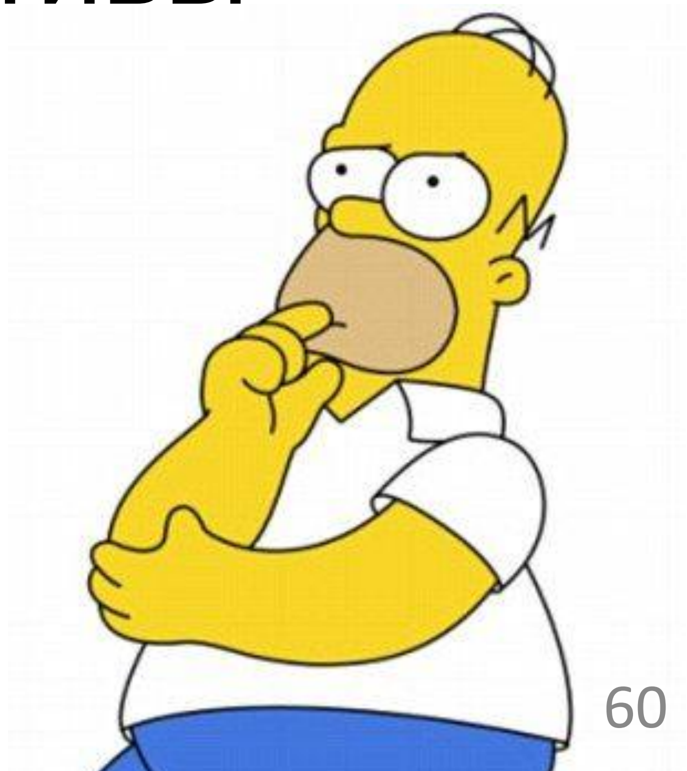
```
public static IQueryable<T> AsExpandable<T>(this IQueryable<T> query)
{
    return AsExpandable(query, LinqKitExtension.QueryOptimizer);
}
```

```
IQueryable<TElement> IQueryProvider.CreateQuery<TElement>(Expression expression)
{
    var expanded = expression.Expand();
    var optimized = _queryOptimizer(expanded);

    return _query.InnerQuery.Provider.CreateQuery<TElement>(optimized)
         .AsExpandable();
}
```

```
IQueryable<TElement> IQueryProvider.CreateQuery<TElement>(Expression expression)
{
    var expanded = expression.Expand();
    var optimized = _queryOptimizer(expanded);

    return _query.InnerQuery.Provider.CreateQuery<TElement>(optimized)
        .AsExpandable();
}
```

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

    return input;
}
```

В поисках альтернативы



PredicateBuilder by Pete Mongomery

https://petemontgomery.wordpress.com/2011/02/10/a-universal-predicatebuilder/

```
public static Expression<T> Compose<T>(this Expression<T> first,
    Expression<T> second, Func<Expression, Expression, Expression> merge)
{
    var map = first.Parameters
        .Select((f, i) => new { f, s = second.Parameters[i] })
        .ToDictionary(p => p.s, p => p.f);

    var secondBody = ParameterRebinder.ReplaceParameters(map, second.Body);

    return Expression.Lambda<T>(merge(first.Body, secondBody), first.Parameters);
}
```

```
public static Expression<T> Compose<T>(this Expression<T> first,
    Expression<T> second, Func<Expression, Expression, Expression> merge)
{
    var map = first.Parameters
        .Select((f, i) => new { f, s = second.Parameters[i] })
        .ToDictionary(p => p.s, p => p.f);

    var secondBody = ParameterRebinder.ReplaceParameters(map, second.Body);

    return Expression.Lambda<T>(merge(first.Body, secondBody), first.Parameters);
}
```

```
public static Expression<T> Compose<T>(this Expression<T> first,
    Expression<T> second, Func<Expression, Expression, Expression> merge)
{
    var map = first.Parameters
        .Select((f, i) => new { f, s = second.Parameters[i] })
        .ToDictionary(p => p.s, p => p.f);

    var secondBody = ParameterRebinder.ReplaceParameters(map, second.Body);

    return Expression.Lambda<T>(merge(first.Body, secondBody), first.Parameters);
}
```

```
protected override Expression VisitParameter(ParameterExpression p)
{
    ParameterExpression replacement;

    if (_map.TryGetValue(p, out replacement))
    {
        p = replacement;
    }

    return base.VisitParameter(p);
}
```

```
public static Expression<T> Compose<T>(this Expression<T> first,
    Expression<T> second, Func<Expression, Expression, Expression> merge)
{
    var map = first.Parameters
        .Select((f, i) => new { f, s = second.Parameters[i] })
        .ToDictionary(p => p.s, p => p.f);

    var secondBody = ParameterRebinder.ReplaceParameters(map, second.Body);

    return Expression.Lambda<T>(merge(first.Body, secondBody), first.Parameters);
}
```

Без AsExpandable

```
public static Expression<Func<T, bool>> And<T>(
    this Expression<Func<T, bool>> first,
    Expression<Func<T, bool>> second)
{
    return first.Compose(second, Expression.AndAlso);
}
```

Спецификации и агрегаты

Спецификация для категории

```
public class Category
{
    public static readonly Spec<Category> NiceRating
        = new Spec<Category>(x => x.Rating > 50);

    public int Rating { get; set; }
}
```

Работает для целевой сущности

```
var niceCategories = _dbContext
     .Categories
     .Where(Category.NiceRating)
     .ToList();
```

Но не для товаров

```
var niceCategories = _dbContext
    .Categories
    .Where(Category.NiceRating)
    .ToList();
```

```
var niceProducts = _dbContext
    .Products
    .Where(Category.NiceRating) // wrong type
    .ToList();
```

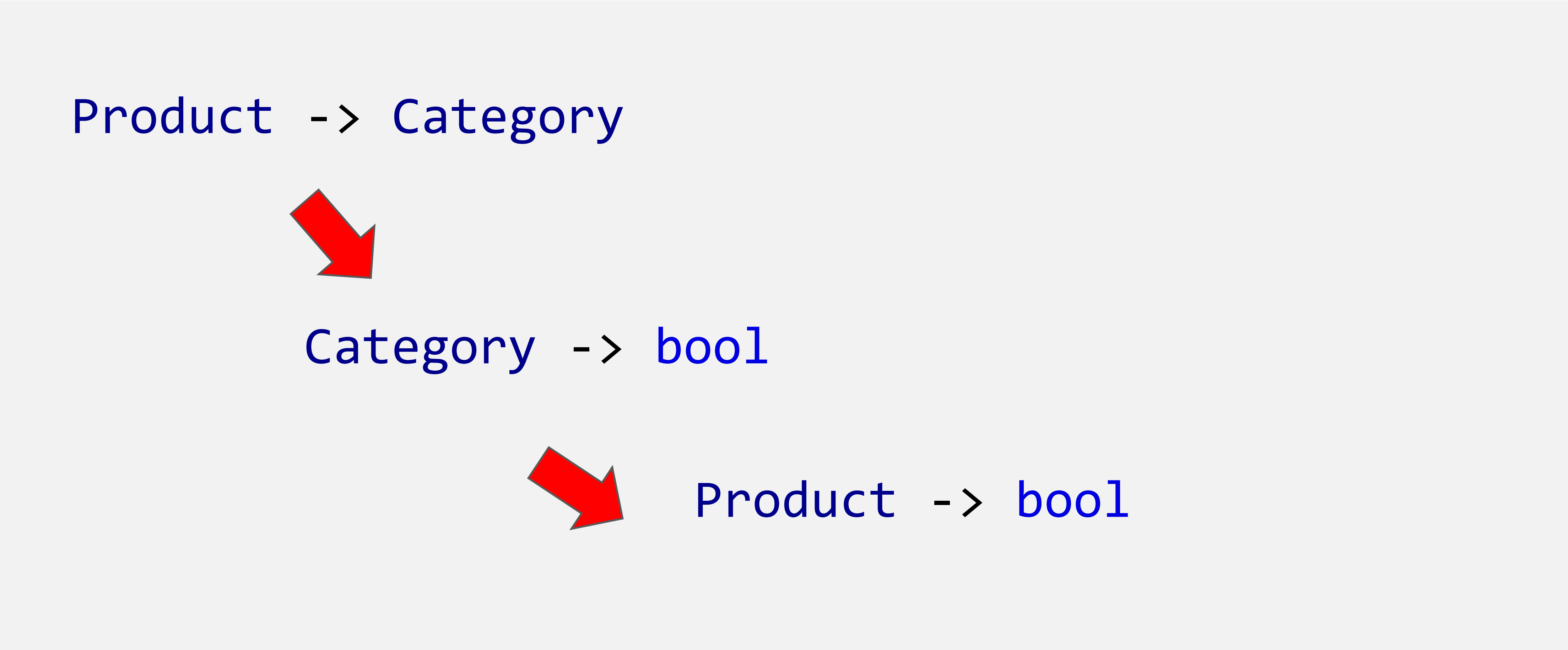
Compilation error

Фиксим

```
var niceProducts = _dbContext
    .Categories
    .Where(Category.NiceRating)
    .SelectMany(x => x.Products)
    .ToList();
```

Композиция

Композиция



```
Product -> Category

          Category -> bool

                         Product -> bool
```

Хочется так

```
var niceProducts = _dbContext
    .Products
    .Where(x => x.Category, Category.NiceRating)
    .ToList();
```

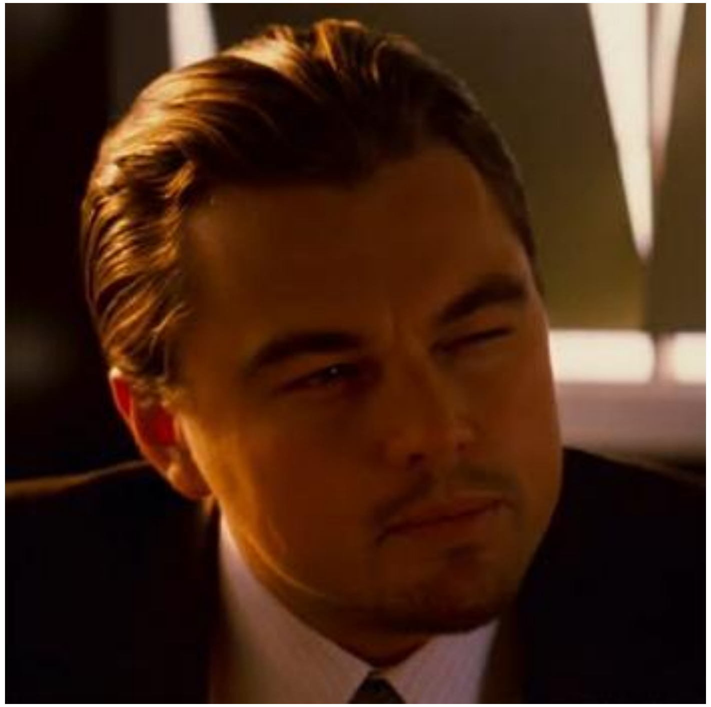
Compose + Where

```
public static IQueryable<T> Where<T, TParam>(this IQueryable<T> queryable,
    Expression<Func<T, TParam>> prop, Expression<Func<TParam, bool>> where)
{
    return queryable.Where(prop.Compose(where));
}
```

Получилось!

```
var niceProducts = _dbContext
    .Products
    .Where(x => x.Category, Category.NiceRating)
    .ToList();
```

Мы можем писать код, который пишет код

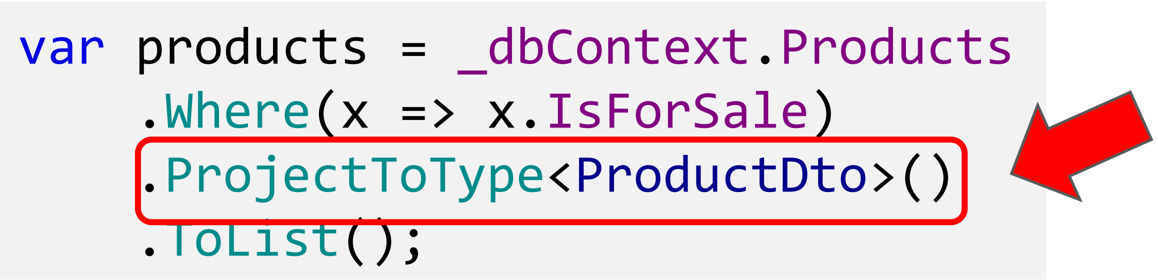


Проекции

Не сложно, но нудно

```
var products = _dbContext.Products
    .Where(x => x.IsForSale)
    .Select(x => new ProductDto()
    {
        Id = x.Id,
        Price = x.Price
    })
    .ToList();
```

Предоставьте это Data Mapper



```
var products = _dbContext.Products
    .Where(x => x.IsForSale)
    .ProjectToType<ProductDto>()
    .ToList();
```

Фильтрация

Снова не сложно, но нудно

```
public IActionResult GetProducts(ProductFilter filter)
{
    IQueryable<Product> products = _dbContext.Products;

    if (filter.Price.HasValue)
    {
        products = products.Where(x => x.Price < filter.Price.Value);
    }

    if (!string.IsNullOrEmpty(filter.Name))
    {
        products = products.Where(x => x.Name.StartsWith(filter.Name));
    }

    return Ok(products.ToList());
}
```

Снова не сложно, но нудно

```
public IActionResult GetProducts(ProductFilter filter)
{
    IQueryable<Product> products = _dbContext.Products;

    if (filter.Price.HasValue)
    {
        products = products.Where(x => x.Price < filter.Price.Value);
    }

    if (!string.IsNullOrEmpty(filter.Name))
    {
        products = products.Where(x => x.Name.StartsWith(filter.Name));
    }

    return Ok(products.ToList());
}
```

Автоматизируем

```
public IActionResult GetProducts(AutoFilter<ProductDto> filter)
{
    var products = _dbContext.Products
        .ProjectToType<ProductDto>()
        .Where(filter)
        .ToList();

    return Ok(products);
}
```

Автоматизируем

```
public IActionResult GetProducts(AutoFilter<ProductDto> filter)
{
    var products = _dbContext.Products
        .ProjectToType<ProductDto>()
        .Where(filter)
        .ToList();

    return Ok(products);
}
```

Строим деревья выражений

```
Expression property = Expression.Property(parameter, x.Property);
Expression value = Expression.Constant(x.Value);

value = Expression.Convert(value, property.Type);

var body = (Expression) Expression.Equal(property, value);

return Expression.Lambda<Func<TSubject, bool>>(body, parameter);
```

Строим деревья выражений

```
Expression property = Expression.Property(parameter, x.Property);
Expression value = Expression.Constant(x.Value);

value = Expression.Convert(value, property.Type);

var body = (Expression) Expression.Equal(property, value);

return Expression.Lambda<Func<TSubject, bool>>(body, parameter);
```

Строим деревья выражений

```
Expression property = Expression.Property(parameter, x.Property);
Expression value = Expression.Constant(x.Value);

value = Expression.Convert(value, property.Type);

var body = (Expression) Expression.Equal(property, value);

return Expression.Lambda<Func<TSubject, bool>>(body, parameter);
```

Строим деревья выражений

```
Expression property = Expression.Property(parameter, x.Property);
Expression value = Expression.Constant(x.Value);

value = Expression.Convert(value, property.Type);

var body = (Expression) Expression.Equal(property, value);

return Expression.Lambda<Func<TSubject, bool>>(body, parameter);
```

Строим деревья выражений

```
Expression property = Expression.Property(parameter, x.Property);
Expression value = Expression.Constant(x.Value);

value = Expression.Convert(value, property.Type);

var body = (Expression) Expression.Equal(property, value);

return Expression.Lambda<Func<TSubject, bool>>(body, parameter);
```

Объединяем

```
if (!props.Any()) return new AutoFilter<T>(x => true);

var expr = compose == Compose.And
    ? props.Aggregate((c, n) => c.And(n))
    : props.Aggregate((c, n) => c.Or(n));

return new AutoFilter<T>(expr);
```

Объединяем

```
if (!props.Any()) return new AutoFilter<T>(x => true);

var expr = compose == Compose.And
    ? props.Aggregate((c, n) => c.And(n))
    : props.Aggregate((c, n) => c.Or(n));

return new AutoFilter<T>(expr);
```

Сортировка чуть сложнее

```
var lambda = FastTypeInfo<Expression>
    .PublicMethods
    .First(x => x.Name == "Lambda");

lambda = lambda.MakeGenericMethod(typeof(Func<,>)
    .MakeGenericType(typeof(TSubject), property.PropertyType));

var expression = lambda.Invoke(null, new object[] {body, new[] {parameter}});

var orderBy = typeof(Queryable)
    .GetMethods()
    .First(x => x.Name == "OrderBy" && x.GetParameters().Length == 2)
    .MakeGenericMethod(typeof(TSubject), property.PropertyType);
```

Сортировка чуть сложнее

```
var lambda = FastTypeInfo<Expression>
    .PublicMethods
    .First(x => x.Name == "Lambda");

lambda = lambda.MakeGenericMethod(typeof(Func<,>)
    .MakeGenericType(typeof(TSubject), property.PropertyType));

var expression = lambda.Invoke(null, new object[] {body, new[] {parameter}});

var orderBy = typeof(Queryable)
    .GetMethods()
    .First(x => x.Name == "OrderBy" && x.GetParameters().Length == 2)
    .MakeGenericMethod(typeof(TSubject), property.PropertyType);
```

Where до или после Select?

Можно и до и после

```
public IActionResult GetProducts(AutoFilter<ProductDto> filter)
{
    var products = _dbContext.Products
        .Where(Product.Specs.IsForSale)
        .ProjectToType<ProductDto>()
        .Where(filter)
        .ToList();

    return Ok(products);
}
```

Можно и до и после

```
public IActionResult GetProducts(AutoFilter<ProductDto> filter)
{
    var products = _dbContext.Products
        .Where(Product.Specs.IsForSale)
        .ProjectToType<ProductDto>()
        .Where(filter)
        .ToList();

    return Ok(products);
}
```

Live Demo

Валидация

TComb.validation

- https://github.com/gcanti/tcomb-validation
- Основана на системе типов
- Пользовательские типы определяются на основе предикатов

Примитивы

```
// null and undefined
validate('a', t.Nil).isValid(); // => false
validate(null, t.Nil).isValid(); // => true
validate(undefined, t.Nil).isValid(); // => true

// strings
validate(1, t.String).isValid(); // => false
validate('a', t.String).isValid(); // => true

// numbers
validate('a', t.Number).isValid(); // => false
validate(1, t.Number).isValid(); // => true
```

Уточнения

```
// a predicate is a function with signature: (x) -> boolean
var predicate = function (x) { return x >= 0; };

// a positive number
var Positive = t.refinement(t.Number, predicate);

validate(-1, Positive).isValid(); // => false
validate(1, Positive).isValid(); // => true
```

Переносим в C#

```
public Refinement<T>(Expression<Func<T, bool>> expression, string errorMessage)
{
    Expression = expression ?? throw new ArgumentNullException(nameof(expression));
    ErrorMessage = errorMessage;
}

public bool IsValid(object obj) => Expression.AsFunc()(obj);
```

Добавляем атрибут валидации

```
public class RefinementAttribute: ValidationAttribute
{
    public IValidator<object> Refinement { get; }

    public RefinementAttribute(Type refinmentType)
    {
        Refinement = (IValidator<object>)
            Activator.CreateInstance(refinmentType);
    }

    public override bool IsValid(object value)
        => Refinement.Validate(value).IsValid();
}
```

Используем

```
public class User
{
    [Refinement(typeof(AdultRefinement))]
    public int Age { get; set; }
}
```

Пишем Visitor для JS

```
switch (node.NodeType)
{
    case ExpressionType.Add:
        _stringBuilder.Append(" + ");
        break;

    case ExpressionType.Divide:
        _stringBuilder.Append(" / ");
        break;

    case ExpressionType.Subtract:
        _stringBuilder.Append(" - ");
        break;

    case ExpressionType.Multiply:
        _stringBuilder.Append(" * ");
        break;

    case ExpressionType.GreaterThan:
        _stringBuilder.Append(" > ");
        break;
```

```
 case ExpressionType.GreaterThanOrEqual:
        _stringBuilder.Append(" >= ");
        break;

    case ExpressionType.LessThan:
        _stringBuilder.Append(" < ");
        break;

    case ExpressionType.LessThanOrEqual:
        _a.Append(" <= ");
        break;

    case ExpressionType.And:
    case ExpressionType.AndAlso:
        _stringBuilder.Append(" && ");
        break;

    case ExpressionType.Or:
    case ExpressionType.OrElse:
        _stringBuilder.Append(" || ");
        break;
}
```

Особое внимание регулярным выражениям

```
protected override Expression VisitMethodCall(MethodCallExpression node)
{
    if (node.Method.DeclaringType == typeof(Regex) && node.Method.Name == "Match")
    {
        var value = ((node.Object as MemberExpression)?.Expression as ConstantExpression)?.Value;

        var regex = value
            .GetType()
            .GetFields(BindingFlags.Instance | BindingFlags.Public).First()
            .GetValue(value);

        _stringBuilder.Append($"!!/{regex}/.exec(x)");

        return node;
    }

    return base.VisitMethodCall(node);
}
```

Особое внимание регулярным выражениям

```
protected override Expression VisitMethodCall(MethodCallExpression node)
{
    if (node.Method.DeclaringType == typeof(Regex) && node.Method.Name == "Match")
    {
        var value = ((node.Object as MemberExpression)?.Expression as ConstantExpression)?.Value;

        var regex = value
            .GetType()
            .GetFields(BindingFlags.Instance | BindingFlags.Public).First()
            .GetValue(value);

        _stringBuilder.Append($"!!/{regex}/.exec(x)");

        return node;
    }

    return base.VisitMethodCall(node);
}
```

Live Demo

Тестирование

Moq

```
var mock = new Mock<ILoveThisFramework>();

Mock
    .Setup(framework => framework.DownloadExists("2.0.0.0"))
    .Returns(true);

ILoveThisFramework lovable = mock.Object;
bool download = lovable.DownloadExists("2.0.0.0");

mock.Verify(
    framework => framework.DownloadExists("2.0.0.0"),
    Times.AtMostOnce());
```

Moq

```
var mock = new Mock<ILoveThisFramework>();

Mock
    .Setup(framework => framework.DownloadExists("2.0.0.0"))
    .Returns(true);

ILoveThisFramework lovable = mock.Object;
bool download = lovable.DownloadExists("2.0.0.0");

mock.Verify(
    framework => framework.DownloadExists("2.0.0.0"),
    Times.AtMostOnce());
```

Мы можем также

```
var resp = GetResponse<ProductController>(
    c => c.Index(new ProductFilter(){Name = "Stuff"}));
```

Есть Intellisense!

Оптимизация Reflection

Альтернатива Activator.CreateInstance

```
var newExp = Expression.New(ctor,argsExp);

var lambda = Expression.Lambda(typeof(ObjectActivator<T>),
    newExp, param);

var compiled = (ObjectActivator<T>)lambda.Compile();
return compiled;
```

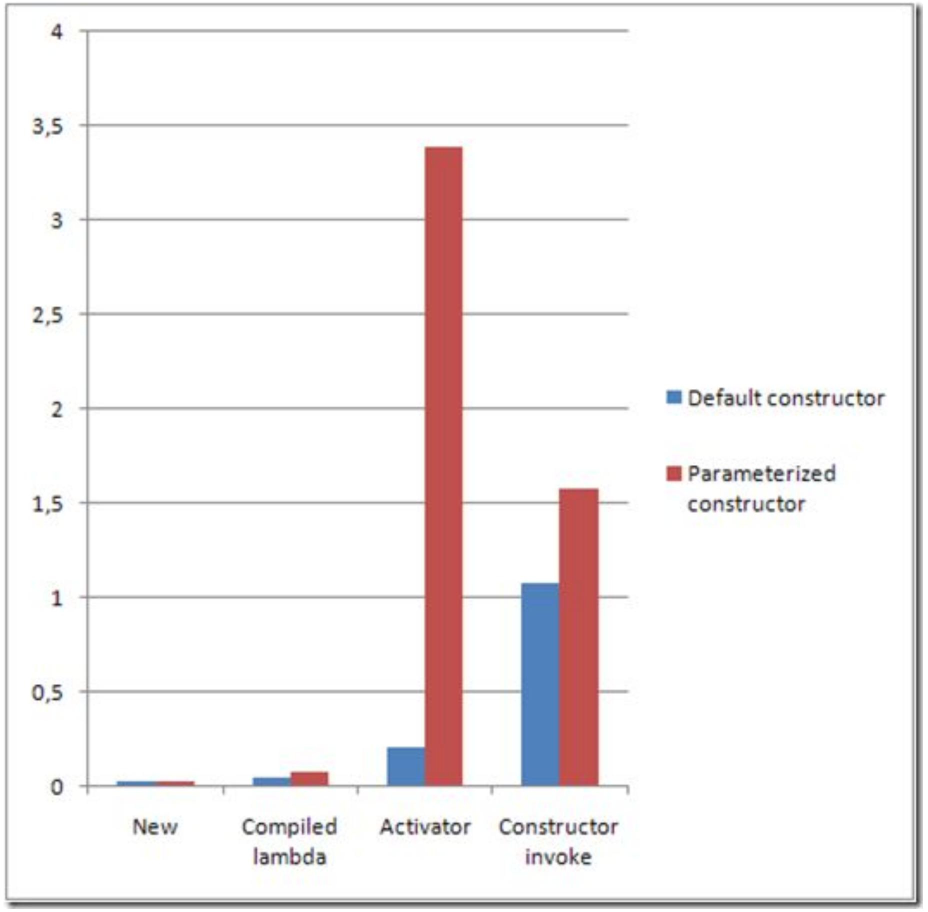
Сравнение производительности

DefaultConstructor_Activator: (0,20 ms per 1000 calls)
DefaultConstructor_CompiledExpression: (0,04 ms per 1000 calls)

DefaultConstructor_Invoke: (1,07 ms per 1000 calls)
DefaultConstructor_New: (0,02 ms per 1000 calls)
DefaultConstructor_NotCompiledExpression: (169,00 ms per 1000 calls)

NonDefaultConstructor_Activator: (3,39 ms per 1000 calls)
NonDefaultConstructor_CompiledExpression: (0,07 ms per 1000 calls)

NonDefaultConstructor_Invoke: (1,57 ms per 1000 calls)
NonDefaultConstructor_New: (0,02 ms per 1000 calls)
NonDefaultConstructor_NotCompiledExpression:
(293,00 ms per 1000 calls)



Компилируем getter’ы

```
public static Func<TObject, TProperty> PropertyGetter<TObject, TProperty>(
     string propertyName)
{
    var paramExpression = Expression.Parameter(typeof(TObject), "value");

    var propertyGetterExpression = Expression.Property(
        paramExpression, propertyName);

    var result = Expression.Lambda<Func<TObject, TProperty>>(
        propertyGetterExpression, paramExpression)
        .Compile();

    return result;
}
```

Компилируем setter’ы

```
var propertyExpression = Expression.Property(
        paramExpression, propertyName);

var result = Expression.Lambda<Action<TObject, TProperty>>
(
    Expression.Assign(propertyExpression, paramExpression2),
        paramExpression, paramExpression2
).Compile();
```

И делегаты

```
public static Delegate CreateMethod(MethodInfo method)
{
    var parameters = method.GetParameters()
        .Select(p => Expression.Parameter(p.ParameterType, p.Name))
        .ToArray();

    var call = Expression.Call(null, method, parameters);
    return Expression.Lambda(call, parameters).Compile();
}
```

Feedback о внедрении

- Автоматизировали рутину, повысили производительность на типовых задачах
- Снизились требования к квалификации команды для решения типовых задач
- Повысились требования к квалификации проектировщика
- Получили деградацию производительности из-за expression.compile, но быстро поправили
- Код стал менее идиоматическим

Спасибо

- max@hightech.today
- https://habrahabr.ru/users/marshinov/posts/
- https://github.com/hightechtoday/force