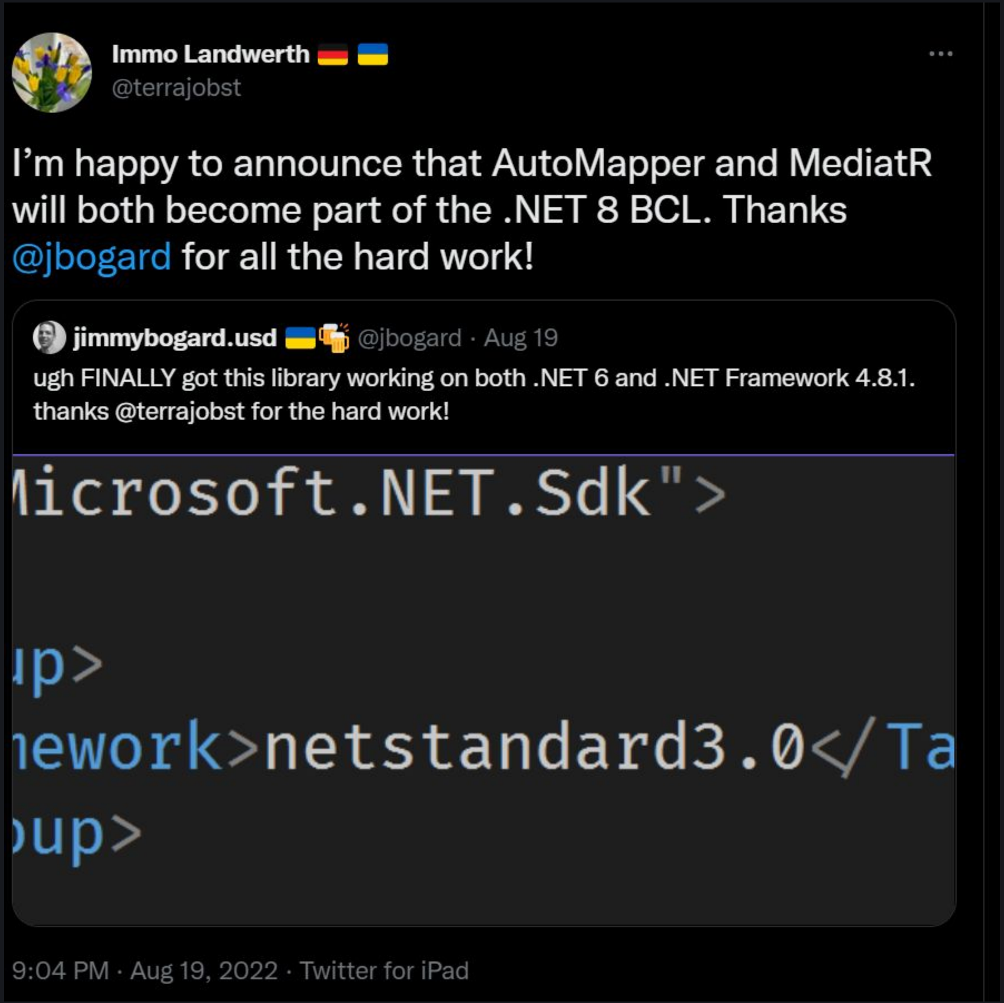
MediatR не нужен

Рассказ о том как страдают
2 principal engineer

DODO ENGINEERING

Андрей Парамонов
Антон Оникийчук


Immo Landwerth 🇩🇪 🇺🇦
@terrajobst

I'm happy to announce that AutoMapper and MediatR will both become part of the .NET 8 BCL. Thanks @jbogard for all the hard work!

jimmybogard.usd 🇺🇦🍻 @jbogard · Aug 19
ugh FINALLY got this library working on both .NET 6 and .NET Framework 4.8.1. thanks @terrajobst for the hard work!

```
icrosoft.NET.Sdk">


p>
ework>netstandard3.0</Ta
up>
```

9:04 PM · Aug 19, 2022 · Twitter for iPad

Link to tweet

Быстрорастворимое проектирование

Кривизна восприятия

Что сказал автор:

- Давайте делить код по фичам а не по слоям
- Давайте делить бизнес логику и технический код
- Давайте хорошо обрабатывать ошибки
- *В принципе MediatR может немного помочь*

Кривизна восприятия

Что сказал автор:

- Давайте делить код по фичам а не по слоям
- Давайте делить бизнес логику и технический код
- Давайте хорошо обрабатывать ошибки
- *В принципе MediatR может немного помочь*

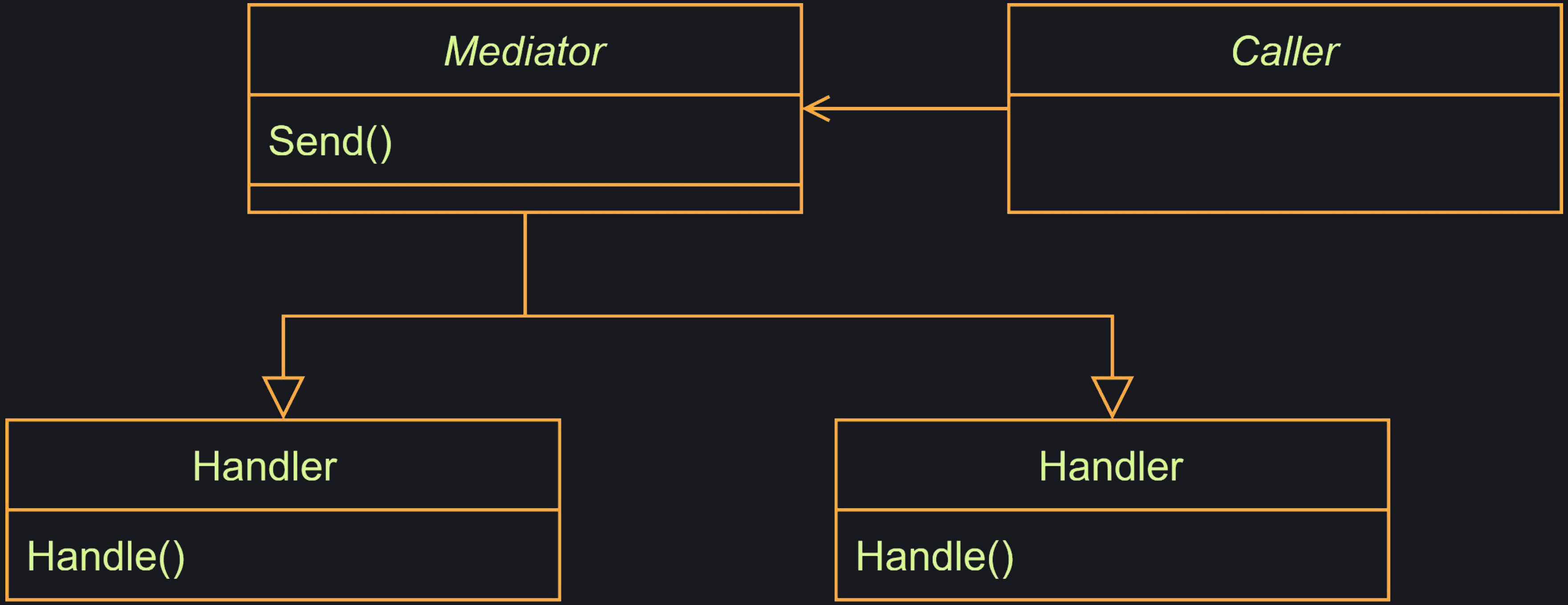
Что услышали/прочитали:

- НАМ НУЖЕН MEDIATR!!!
- Ну и код по фичам и папочкам можно разложить

МЫ БУДЕМ
МЫ БУДЕМ
ПИСАТЬ
ПИСАТЬ
ПОНЯТНЫЙ КОД
ПОНЯТНЫЙ КОД
МЫ БУДЕМ ПИСАТЬ ПОНЯТНЫЙ КОД
ВОЗЬМЕМ MEDIATR
imgflip.com

Что такое MediatR?

Реализация паттерна Mediator в .net



How-to

```
record CalculationRequest(int Target) : IRequest<CalculationResponse>;

record CalculationResponse(int Factorial, int FibonacciNumber);

class Handler : IRequestHandler<CalculationRequest, CalculationResponse>
{

   public CalculationResponse Handle(CalculationRequest request)
   {
        return new CalculationResponse(0,0);
   }
}

_mediator.Send(new CalculationRequest(0));


Services.AddMediatR(typeof(App));
```

C Pipeline

```
class LogBehavior<TRequest, TResponse>: IPipelineBehavior<TRequest, TResponse>
   where TRequest : IRequest<TResponse>
{
   public async Task<TResponse> Handle(TRequest request,
       RequestHandlerDelegate<TResponse> next, CancellationToken ct)
   {
       _logger.LogInformation($"Handling {typeof(TRequest).Name}");
       var response = await next();
       _logger.LogInformation($"Handled {typeof(TResponse).Name}");

       return response;
   }
}
```

C Notification

```
class Ping : INotification { }
class PongHandler1 : INotificationHandler<Ping>
{
   public Task Handle(Ping notification, CancellationToken ct)
   {
       // do some work
       return Task.CompletedTask;
   }
}
class PongHandler2 : INotificationHandler<Ping>
{
   public Task Handle(Ping notification, CancellationToken ct)
   {
       // do some other work
       return Task.CompletedTask;
   }
}
await mediator.Publish(new Ping());
```

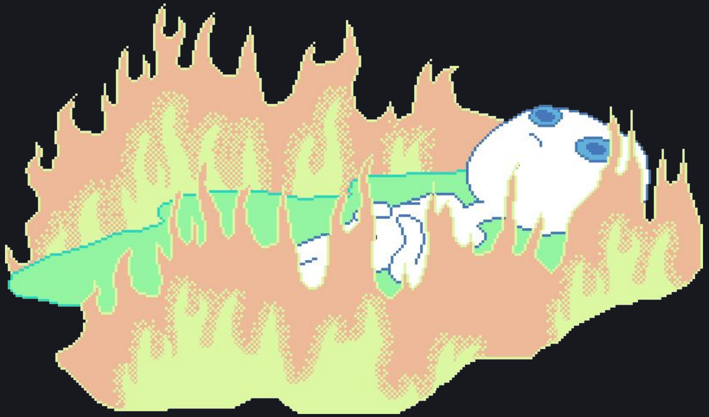
Про Performance

Microbenchmark

Method	Mean	Error	Ratio	Allocated	Alloc Ratio
JustCall	98.29 ns	0.780 ns	1.00	40 B	1.00
NativeTransient	94.66 ns	0.713 ns	0.96	192 B	4.80
MediatrTransient	875.23 ns	8.894 ns	8.91	1720 B	43.00
NativeSingleton	88.64 ns	0.712 ns	0.90	144 B	3.60
MediatrSingleton	869.52 ns	4.677 ns	8.84	1672 B	41.80

Benchmark ближе к реальности

name	http_req_duration avg	http_req_duration p(95)	vus_max max	http_reqs count
sunny-day-native	1.32 ms	3.00 ms	20	862939
sunny-day-mediatr	1.24 ms	2.96 ms	20	915294
rainy-day-native	10.40 ms	30.64 ms	5000	297647
rainy-day-mediatr	13.93 ms	41.40 ms	5000	296746

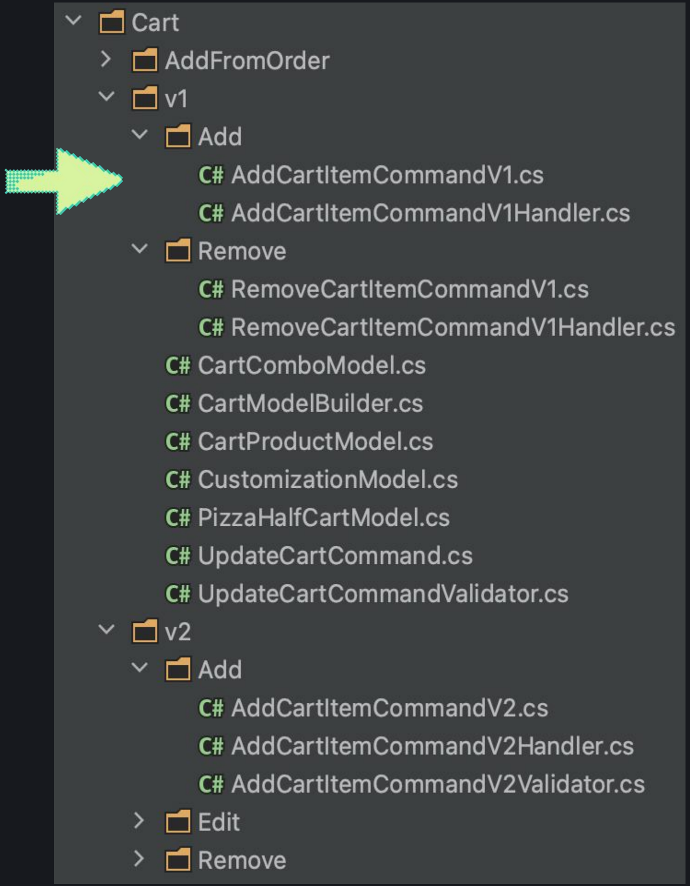


Что не так с MediatR?

Как понять что происходит?

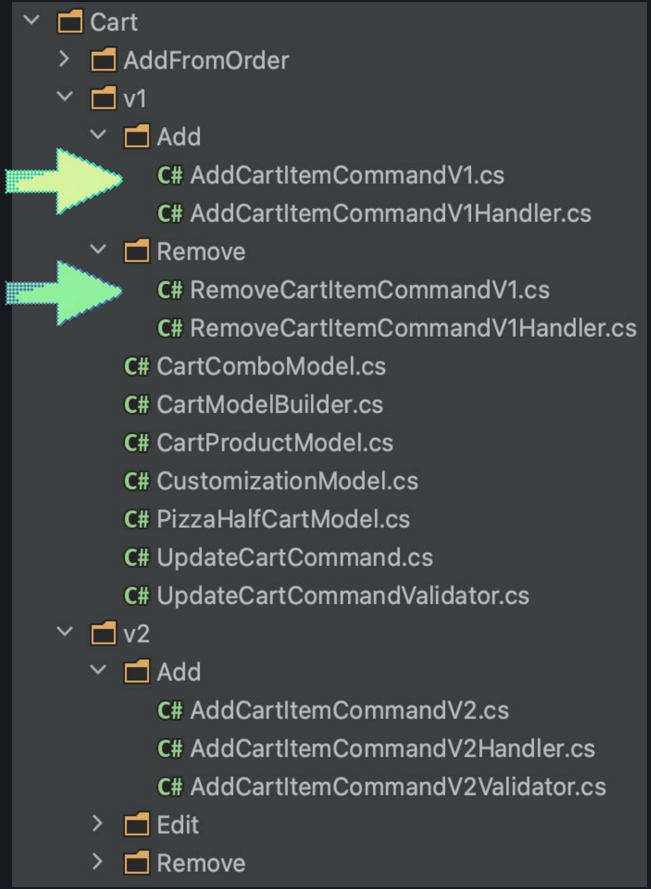
```
[HttpPost("calculate")]
public async Task<CalculationResponse>
    Calculate([FromBody] CalculateInput input)
{
   return await _mediator.Send(new CalculationRequest(input.target));
}
```

Но ведь у нас все по папочкам



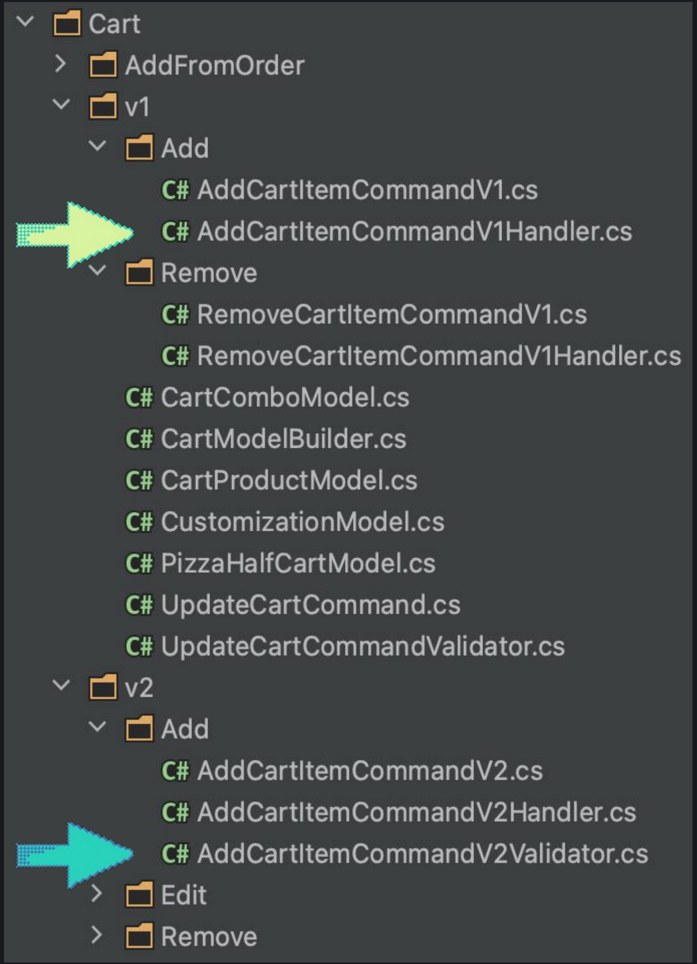
```
Cart
  AddFromOrder
  v1
    Add
      AddCartItemCommandV1.cs
      AddCartItemCommandV1Handler.cs
    Remove
      RemoveCartItemCommandV1.cs
      RemoveCartItemCommandV1Handler.cs
    CartComboModel.cs
    CartModelBuilder.cs
    CartProductModel.cs
    CustomizationModel.cs
    PizzaHalfCartModel.cs
    UpdateCartCommand.cs
    UpdateCartCommandValidator.cs
  v2
    Add
      AddCartItemCommandV2.cs
      AddCartItemCommandV2Handler.cs
      AddCartItemCommandV2Validator.cs
    Edit
    Remove
```

Но ведь у нас все по папочкам



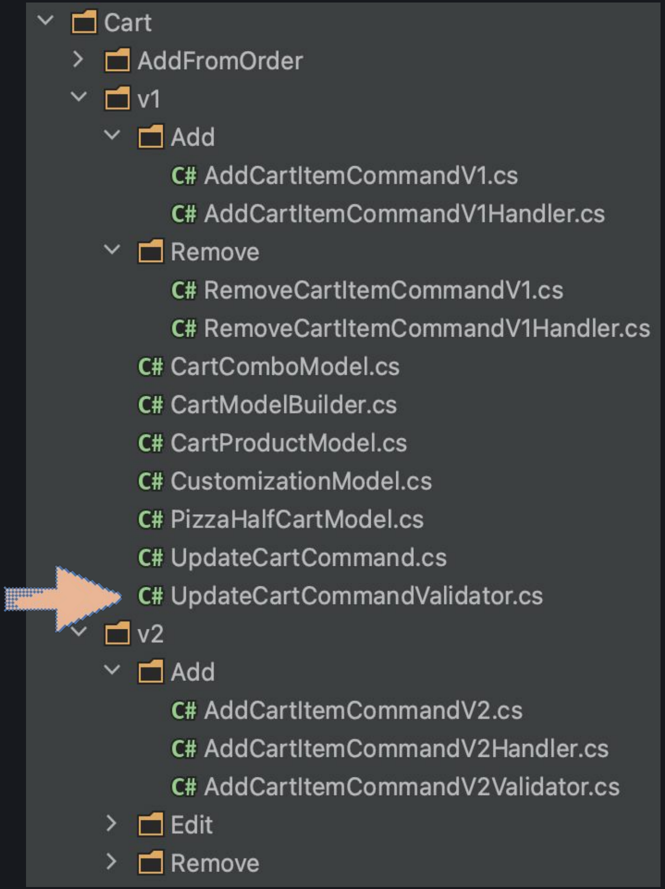
```
Cart
  AddFromOrder
  v1
    Add
      AddCartItemCommandV1.cs
      AddCartItemCommandV1Handler.cs
    Remove
      RemoveCartItemCommandV1.cs
      RemoveCartItemCommandV1Handler.cs
    CartComboModel.cs
    CartModelBuilder.cs
    CartProductModel.cs
    CustomizationModel.cs
    PizzaHalfCartModel.cs
    UpdateCartCommand.cs
    UpdateCartCommandValidator.cs
  v2
    Add
      AddCartItemCommandV2.cs
      AddCartItemCommandV2Handler.cs
      AddCartItemCommandV2Validator.cs
    Edit
    Remove
```

Сюрприз!



```
Cart
  AddFromOrder
  v1
    Add
      AddCartItemCommandV1.cs
      AddCartItemCommandV1Handler.cs
    Remove
      RemoveCartItemCommandV1.cs
      RemoveCartItemCommandV1Handler.cs
    CartComboModel.cs
    CartModelBuilder.cs
    CartProductModel.cs
    CustomizationModel.cs
    PizzaHalfCartModel.cs
    UpdateCartCommand.cs
    UpdateCartCommandValidator.cs
  v2
    Add
      AddCartItemCommandV2.cs
      AddCartItemCommandV2Handler.cs
      AddCartItemCommandV2Validator.cs
    Edit
    Remove
```

Но ведь переиспользование кода!



```
⌄ Cart
  › AddFromOrder
  ⌄ v1
    ⌄ Add
        C# AddCartItemCommandV1.cs
        C# AddCartItemCommandV1Handler.cs
    ⌄ Remove
        C# RemoveCartItemCommandV1.cs
        C# RemoveCartItemCommandV1Handler.cs
      C# CartComboModel.cs
      C# CartModelBuilder.cs
      C# CartProductModel.cs
      C# CustomizationModel.cs
      C# PizzaHalfCartModel.cs
      C# UpdateCartCommand.cs
      C# UpdateCartCommandValidator.cs
  ⌄ v2
    ⌄ Add
        C# AddCartItemCommandV2.cs
        C# AddCartItemCommandV2Handler.cs
        C# AddCartItemCommandV2Validator.cs
    › Edit
    › Remove
```

Pipeline против читабельности

F# pipeline vs C# pipeline

```fsharp
let updateEmailHandler =
    validateEmail
    |> updateEmail
    |> saveChanges
    |> logAction
```

```csharp
services.AddScoped<IEmailValidator,
    EmailValidator>();
services.AddScoped<IUpdateEmailService,
    UpdateEmailService>();
services.Decorate<IUpdateEmailService,
    SaveChangesService>();
services.Decorate<IUpdateEmailService,
    LogActionService>();
```

MediatR pipeline в реальности

```
public static void AddApplication(this IServiceCollection services)
{
   services.AddMediatR(typeof(DependencyMarker));
   services.Scan(
       selector => selector
           .FromAssemblyOf<DependencyMarker>()
           .AddClasses(classes => classes.AssignableTo(typeof(IPipelineBehavior<,>)))
           .AsImplementedInterfaces()
           .AddClasses(classes => classes.AssignableTo(typeof(IValidator<>)))
           .AsImplementedInterfaces());
}
```

Наследование реализации

```
class Formatter
{
   protected virtual string GetFormat() => "{0}";

   public string Format(string src) => String.Format(GetFormat(), src);
}

class CoolFormatter : Formatter
{
   protected override string GetFormat() => "Cool {0}";
}
```

Наследование реализации

```
interface IFormattingStrategy
{
    string GetFormat();
}

class DefaultFormat : IFormattingStrategy
{
    public string GetFormat() => "{0}";
}

class CoolFormat : IFormattingStrategy
{
    public string GetFormat() => "Cool {0}";
}
```

Наследование реализации

```
interface IFormattingStrategy
{
    string GetFormat();
}

class DefaultFormat : IFormattingStrategy
{
    public string GetFormat() => "{0}";
}

class CoolFormat : IFormattingStrategy
{
    public string GetFormat() => "Cool {0}";
}
```

Наследование реализации

```
new CoolFormatter().Format("Anton")
```

```
String.Format(strategy.GetFormat(), "Anton");
```

А что делать?

Написать простой код

```
public async Task<OrderWorkflowStateModel> Handle(Command request)
{
  await using var transaction = await Transaction<Command>.Begin(request);

  await transaction.Workflow.Validate(cancellationToken);

  await DoAction(transaction.Workflow);

  await transaction.Commit();

  return new(
     await WorkflowModelBuilder.Build(
        transaction.Workflow.GetState(),
        request.WorkflowRequest.ClientVersion,
        cancellationToken)
  );
}
```

Как убрать неявный pipeline?

Non functional concerns

Logging

```csharp
public async Task<TResponse> Handle(TRequest request,
   RequestHandlerDelegate<TResponse> next, CancellationToken ct)
{
   using (_logger.BeginScope(_requestScopeGenerators.Generate(request)))
   {
       _logger.LogInformation("Start handling");
       try
       {
           var result = await next();
           using (_logger.BeginScope(_responseScopeGenerators.Generate(result)))
           {
               _logger.LogInformation("Request handled");
               return result;
           }
       }
       catch (Exception exception)
       {
           _logger.LogError(exception, "Fail to handle request");
           throw;
       }
   }
```

Logging

```
builder.Services.AddHttpLogging(opts =>
    opts.LoggingFields = opts.LoggingFields |
                         HttpLoggingFields.RequestBody |
                         HttpLoggingFields.ResponseBody);
```

```
app.UseHttpLogging();
```

Logging

```
public async Task Invoke(HttpContext context)
{
    using (_logger.BeginScope(_requestGenerator.Generate(context.Request)))
    {
        _logger.LogInformation("Start handling http request");
        try
        {
            await _next(context);

            using (_logger.BeginScope(_responseGenerator.Generate(context.Response)))
            {
                _logger.LogInformation("Http request handled");
            }
        }
        catch (Exception e)
        {
            _logger.LogError(e, "Fail to handle http request");
            throw;
        }
    }
}
```

Logging

```
public interface IScope
{
   IDisposable WithScope<T>(T state);
}


public async Task<(int,int)> Calculate([FromBody] CalculateInput input)
{
   using var _ = _scope.WithScope(input);
   return await _unit.DoCalulate(input.target.Value);
}
```

Metrics

```
var stopwatch = Stopwatch.StartNew();
_calls.Add(1,RequestTag);
try
{
    var response = await next();
    _success.Add(1,RequestTag);
    return response;
}
catch
{
    _errors.Add(1,RequestTag);
    throw;
}
finally
{
    stopwatch.Stop();
    _time.Record(stopwatch.ElapsedMilliseconds,RequestTag);
}
```

Metrics

```
using(Elapsed.WithMeter<MediatRController.CalculateInput>())
{
   return await _unit.DoCalulate(input.target.Value);
}
```

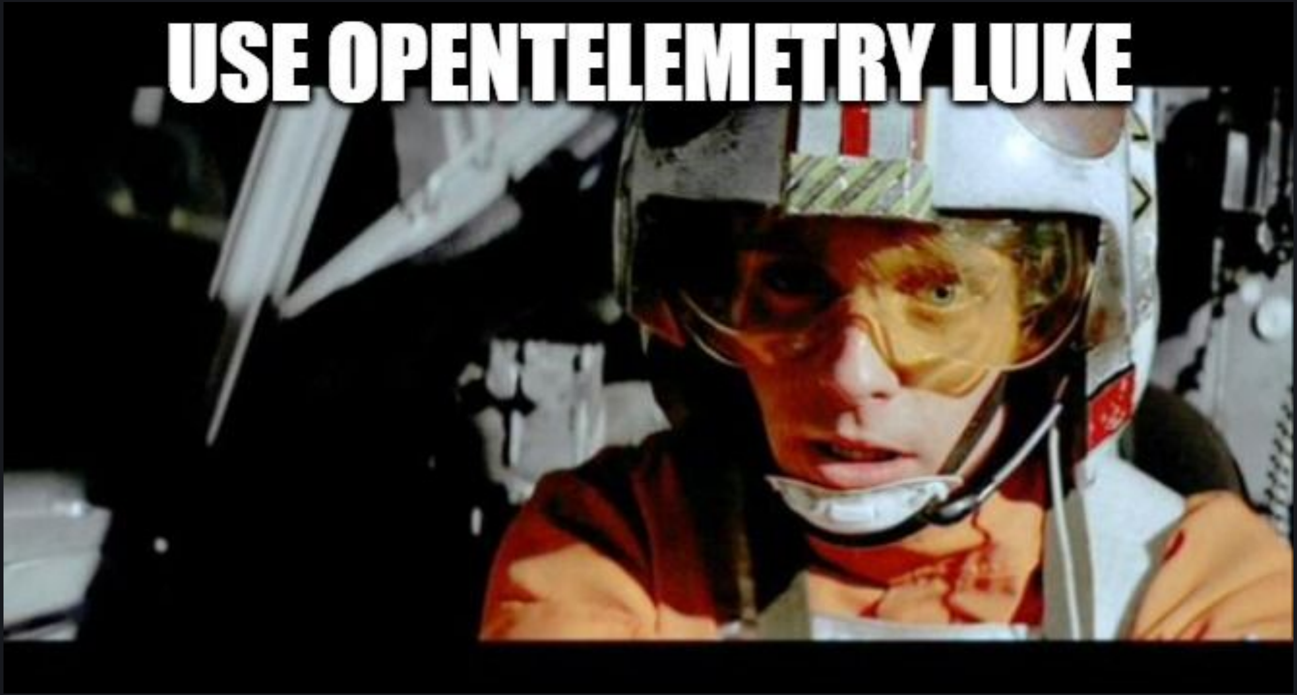
Tracing

```
using var activity = ActivitySource.StartActivity(ActivityKind.Internal);
activity?.SetTag("Request", nameof(TRequest));
return await next();
```

Tracing

```
using(Trace.WithTrace<MediatRController.CalculateInput>())
{
    return await _unit.DoCalculate(target);
}
```

OpenTelemetry



OpenTelemetry

Распределенный трейсинг OpenTelemetry вместо логирования всего подряд

Business concerns

Validation

```
public Task<TResponse> Handle(TRequest request, CancellationToken ct,
  RequestHandlerDelegate<TResponse> next)
{
  var validationContext = new ValidationContext<TRequest>(request);
  var validationResults = _validators.SelectMany(validator =>
      validator.Validate(validationContext).Errors);

  var failures = validationResults.WhereNotNull().ToList();

  if (failures.Count != 0)
     throw new ValidationException(failures);

  return next();
}
```

Validation

```
record Request(
    string? SourceAccountId,
    string? TargetAccountId,
    int? Amount,
    string? Currency
    ) : IRequest<Response>;
```

Validation

```
[HttpPost("calculate")]
public async Task<(int, int)> Calculate([FromBody] CalculateInput input)
{
    if (input.target == null)
    {
        throw new ValidationException(nameof(input.target), "should present");
    }

    return await _unit.DoCalculate(input.target!);
}
```

Transaction

```
public async Task<TResponse> Handle(TRequest request, CancellationToken ct,
   RequestHandlerDelegate<TResponse> next)
{
   var response = await next();
   var context = _commandContextAccessor.Context;
   var changes = context.Aggregates.SelectMany(a => a.Changes);
   if (!changes.Any() && !context.Projections.Any())
       return response;

   using var transaction = await _dbSessionFactory.OpenTransactionAsync();
   await AppendChanges(transaction, changes, ct);
   await ProcessProjections(transaction, context.Projections, ct);
   await transaction.CommitAsync();

   return response;
}
```

Transaction

```
public Task<Result> DoStuff()
{
   using var transaction = await transactionManager.OpenTransactionAsync();
   // do some work
   await transaction.Commit(ct);
   return result;
}
```

Propagate changes

```
public async Task<TResponse> Handle(TRequest request,CancellationToken ct,
   RequestHandlerDelegate<TResponse> next)
{
   var result = await next();
   await _eventPublisher.PublishAllPendingEvents(cancellationToken);
   return result;
}
```

Propagate changes

```
public async Task Notify(IEnumerable<EntityBase> entities)
{
    foreach (var entity in entities)
    {
        var events = entity.GetChanges.ToArray();
        foreach (var domainEvent in events)
        {
            await _mediator.Publish(domainEvent);
        }
        entity.Commit();
    }
}
```

Propagate changes

```
public Task<Result> DoStuff()
{
   using var transaction = await transactionManager.OpenTransactionAsync();
   //do some work
   await transaction.Commit(ct);
   await _eventPublisher.Notify(transaction.Changes);
   return result;
}
```

Propagate changes

```
public Task<Result> DoStuff()
{
   using var transaction = await transactionManager.OpenTransactionAsync();
   //do some work
   await _eventPublisher.Notify(transaction.Changes);
   await transaction.Commit(ct);
   return result;
}

Services.AddTransient(typeof(IPipelineBehavior<,>),
                      typeof(NotificationBehavior<,>));
Services.AddTransient(typeof(IPipelineBehavior<,>),
                      typeof(TransactionBehavior<,>));
```

Locking

```
internal sealed class StrictCommandOrderBehavior<TRequest, TResponse> :
IPipelineBehavior<TRequest, TResponse>
   where TRequest : class, IRequireStrictCommandOrder
{

   public async Task<TResponse> Handle(TRequest request, CancellationToken ct,
       RequestHandlerDelegate<TResponse> next)
   {
       await using var @lock = await _distributedLockAcquirer.AcquireLock(
           request.Id,
           StrictCommandOrderBehaviorUtilities.DefaultTimeout,
           cancellationToken);
       return await next();
   }
}
```

Locking

```
public Task<Result> DoStuff()
{
    await using var _ = await _lockAcquirer.AcquireLock(
        "calculate",
        TimeSpan.FromMinutes(1),
        ct);
    //do some work
    return result;
}
```

Retry

```
public class OptimisticConcurrencyRetryBehavior<TResponse>
   : IPipelineBehavior<IOptimisticConcurrencyRetriable, TResponse>
{
   private readonly ICosmosRetryPolicyFactory _cosmosRetryPolicyFactory;

   public async Task<TResponse> Handle(
       IOptimisticConcurrencyRetriable request,
       CancellationToken ct,
       RequestHandlerDelegate<TResponse> next)
   {
       return await _cosmosRetryPolicyFactory.DefaultRetryPolicy
           .ExecuteAsync(async () => await next());
   }
}
```

Retry

```
public class OptimisticConcurrencyRetryBehavior<TResponse>
   : IPipelineBehavior<IOptimisticConcurrencyRetriable, TResponse>
{
   private readonly ICosmosRetryPolicyFactory _cosmosRetryPolicyFactory;

   public async Task<TResponse> Handle(
       IOptimisticConcurrencyRetriable request,
       CancellationToken ct,
       RequestHandlerDelegate<TResponse> next)
   {
       return await _cosmosRetryPolicyFactory.DefaultRetryPolicy
           .ExecuteAsync(async () => await next());
   }
}
```

Retry

```
[HttpPost("calculate")]
public async Task<(int, int)> Calculate([FromBody] CalculateInput input)
{
   var retryPolicy = _retriesFactory.DefaultStoragePolicy;
   return await retryPolicy.ExecuteAsync(
       async () => await _unit.DoCalculate(input.target));
}
```

Brave New World

```
public async Task<Result> DoStuff(Request request, CancellationToken ct)
{
    await ValidateRequest(request);
    await using var _ = _lockProvider.For<UnitOfWork>(ct);
    await using var transaction = await _transactionManager.OpenTransactionAsync(ct)

    var data = await _retryProvider.For(
        ct => _external.Fetch(new DataRequest(), ct), ct);
    var state = await transaction.Get<State>();

    state.Apply(data);

    await transaction.Commit(ct);
    await _eventPublisher.Notify(transaction.Changes, ct);
    return CreateResponse(state);
}
```

VS MediatR

```
Services.AddTransient(typeof(IPipelineBehavior<,>), typeof(ValidationBehavior<,>));

Services.AddTransient(typeof(IPipelineBehavior<,>), typeof(LockingBehavior<,>));

Services.AddTransient(typeof(IPipelineBehavior<,>), typeof(RetryBehavior<,>));

Services.AddTransient(typeof(IPipelineBehavior<,>), typeof(TransactionBehavior<,>));

Services.AddTransient(typeof(IPipelineBehavior<,>), typeof(NotificationBehavior<,>));
```

Выводы

Выводы

- Pipeline ненужен тк усложняет понимание кода
- Без pipeline ненужен IRequest<TResponse>
- INotificationHandler все еще крут (хотя есть MulticastDelegate)

Strange concerns

Data parsing and enrichment

```
public async Task<TResponse> Handle(IAppRequest request, CancellationToken ct,
  RequestHandlerDelegate<TResponse> next)
{
   var contextRequest = _httpContextAccessor.HttpContext!.Request;
   var platform = contextRequest.Headers.GetPlatform();
   var deviceId = contextRequest.Headers.GetDeviceId();

   request.AppInfo = new AppInfo(
      platform,
      deviceId
   );

   return await next();
}
```

Data parsing and enrichment

```
public async Task<TResponse> Handle(IAppRequest request, CancellationToken ct,
  RequestHandlerDelegate<TResponse> next)
{
   var platform = GetPlatform();
   var deviceId = GetDeviceId();

   request.AppInfo = new AppInfo(
      platform,
      deviceId
   );

   return await next();
}
```

Data parsing and enrichment

```
interface IAppRequest
{
    AppInfo AppInfo { get; set; }
}
```

Control flow

```
public class CountrySpecificEventBehavior<TRequest, TResponse> :
IPipelineBehavior<TRequest, TResponse> where TRequest : ICountrySpecific
  where TResponse : new()
{
  private readonly CountryOptions _countryOptions;

  public async Task<TResponse> Handle(TRequest request, CancellationToken ct,
     RequestHandlerDelegate<TResponse> next)
  {
     if (request.CountryId == null || request.CountryId == _countryOptions.Code)
     {
        return await next();
     }

     return new TResponse();
  }
}
```

Directed by
ROBERT B. WEIDE