DOTNEXT

2022 Spring

Денис Цветцих

DevBrothers

Аспектно-ориентированное программирование вчера, сегодня и завтра

Обо мне

Использую АОП с 2010 года

Поделюсь опытом

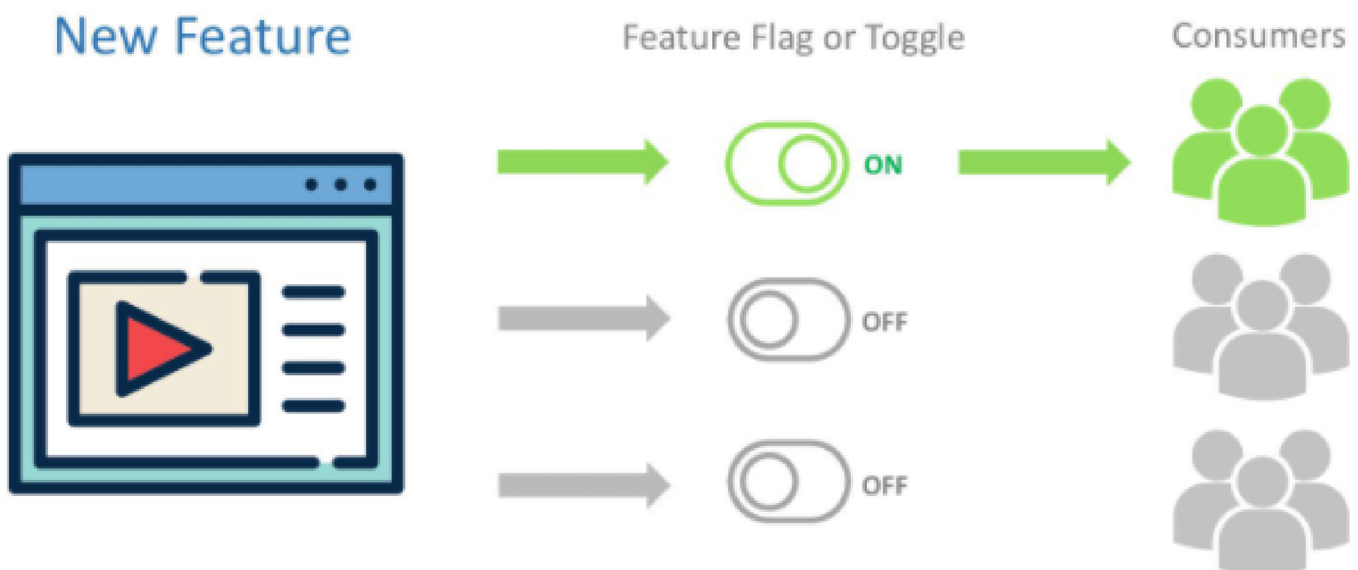
О чем поговорим

- Зачем нужно АОП
- Что такое АОП
- Варианты реализации АОП
- Библиотеки и фреймворки для АОП

Зачем нужно АОП

- Декомпозировать код
- Добавлять функциональность, не меняя модули

Пример: FeatureFlags



FeatureFlag

```
public class OrderService
{
    public Order GetOrder(int id)
    {
        FeatureManager.Check("Order");
        // get order
    }



}
```

FeatureFlag

```
public class OrderService
{
    public Order GetOrder(int id)
    {
        FeatureManager.Check("Order");
        // get order
    }

    public void UpdateOrder(int id, OrderDto dto)
    {
        FeatureManager.Check("Order");
        // update order
    }
}
```

Доступ к сущности

```
public class OrderService
{
    public Order GetOrder(int id)
    {
        FeatureManager.Check("Order");
        CurrentUserManager.CheckOrder(id));
        // get order
    }

    public void UpdateOrder(int id, OrderDto dto)
    {
        FeatureManager.Check("Order");
        CurrentUserManager.CheckOrder(id));
        // update order
    }
}
```

FeatureFlag – чего хочется

```
public class OrderService
{
    [Feature("Order")]
    public Order GetOrder(int id)
    {
        // get order
    }
    [Feature("Order")]
    public void UpdateOrder(int id, OrderDto dto)
    {
        // update order
    }
}
```

FeatureFlag – чего хочется

```
[Feature("Order")]
public class OrderService
{
    public Order GetOrder(int id)
    {
        // get order
    }

    public void UpdateOrder(int id, OrderDto dto)
    {
        // update order
    }
}
```

FeatureFlag – мечта

```
FeatureManager.AddFeature("Order", "OrderService.*Method[Enter]");

public class OrderService
{
    public Order GetOrder(int id)
    {
        // get order
    }

    public void UpdateOrder(int id, OrderDto dto)
    {
        // update order
    }
}
```

Аспектно-ориентированное программирование

Парадигма программирования,
предназначенная для
декомпозиции кода
при помощи специальных конструкций “аспектов”

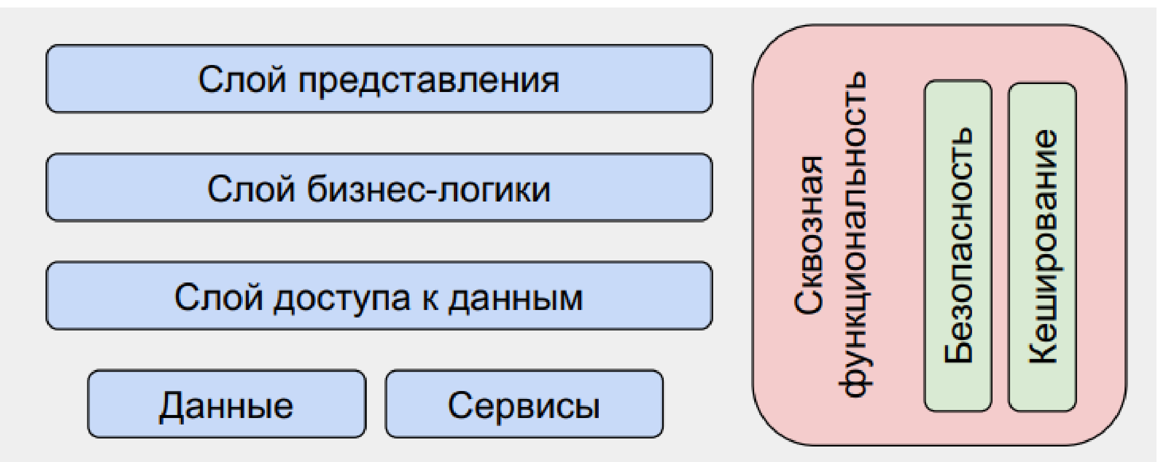
Решает проблему “сквозной функциональности”

Функциональность (concern)

- Основная (core concern)
  - Продажа товаров
- Второстепенная (non-core concern)
  - Логгирование
  - Кеширование

Сквозная функциональность

Cross-cutting concern - функциональность, реализация которой рассредоточена на несколько модулей или слоев приложения



Пример

```
public class OrderService
{
    public Order GetOrder(int id)
    {
        FeatureManager.Check("Order");

        return _dbContext.Orders.Find(id);
    }

}
```

Основная задача
(core concern)

Пример

Cross-cutting concern

```
public class OrderService
{
    public Order GetOrder(int id)
    {
        FeatureManager.Check("Order");

        return _dbContext.Orders.Find(id);
    }

}
```

Определения

- Аспект (aspect) - некоторая (не основная) задача программы, выделенная в отдельный модуль
- Совет (advice) - реализация задачи аспекта
- Срез (point cut) - (декларативное) описание всех мест действия аспекта
- Точка прикрепления (join point) - место, подходящее под описание среза

Срез – это важно!

Часто: Один аспект добавить для кучи классов на разных уровнях

Редко: для одного класса добавить несколько аспектов

Аспект = Совет + Срез [Точки прикрепления]

Аспект

```
class Aspect
{
    string PointCut =>
    "OrderService.*Method[Enter]";

    public void Advice()
    {
        FeatureManager.Check("Order");
    }
}
```

Исходный код

```
class OrderService
{
    public Order GetOrder(int id)
    {

    }
}
```

Совет

Срез

Точка прикрепления

Аспект = Совет + Срез [Точки прикрепления]

Аспект

```
class Aspect
{
    string PointCut =>
    "OrderService.*Method[Enter]";

    public void Advice()
    {
        FeatureManager.Check("Order");
    }
}
```

Исходный код

```
class OrderService
{
    public Order GetOrder(int id)
    {
      FeatureManager.Check("Order");
    }
}
```

Применение аспекта

Вынесение функциональности в аспекты

- Аспект “проверка фиче-флагов”
- Совет – проверка конкретного фиче-флага
- Срез – начало методов класса OrderService

Внедрение (introduction)

- Внедрение (introduction) - изменение структуры элемента программы
  - Добавление поддержки интерфейсов
  - Добавление новых методов, свойств, полей

- Примесь (MixIn) – совет, реализующий внедрение

Связывание (weaving)

- Связывание (weaving) - способ заставить выполнить совет в точках прикрепления
- Статическое (перед запуском программы)
- Динамическое (во время выполнения программы)

Связывание (weaving)

- Статическое
  - Переписывание исходного кода (макросы)
  - Переписывание IL (перекомпиляция)

- Динамическое
  - Архитектура (декораторы, пайплайны)
  - Проксирование (Генерация оберток)
  - Переписывание IL (JIT-перехват)

Примеры задач для АОП

- Профилирование
- Логгирование
- Кеширование
- Обработка ошибок
- Соответствие контракту

Динамическое связывание

Декоратор вручную

```
public class FeatureDecorator : IOrderService
{
    readonly IOrderService _service;

    public FeatureDecorator(IOrderService service)
        => _service = service;

    public Order GetOrder(int id)
    {
        FeatureManager.Check("Order");

        return _service.GetOrder(id);
    }
}
```

Scrutor decoration

```
var collection = new ServiceCollection();

collection.AddSingleton<IOrderService, OrderService>();

collection.Decorate<IOrderService, FeatureDecorator>();
collection.Decorate<IOrderService, AccessDecorator>();
```

Декораторы DI (Autofac)

```
var builder = new ContainerBuilder();

builder.RegisterGeneric(typeof(OrderService))
    .As(IOrderService);

builder.RegisterGenericDecorator(typeof(FeatureDecorator),
    typeof(IOrderService));
builder.RegisterGenericDecorator(typeof(AccessDecorator),
    typeof(IOrderService));

var container = builder.Build();

var service = container.Resolve<IOrderService>();
```

Масштабирование – GOD Object

```
public class FeatureDecorator : IOrderService, IProfileService
{
    readonly IOrderService _orderService;
    readonly IProfileService _profileService;

    public FeatureDecorator(IOrderService orderService, IProfileService profileService)
    { _orderService = orderService; _profileService = profileService;

    public Order GetOrder(int id) {
        FeatureManager.Check("Order");
        return _service.GetOrder(id);
    }

    public Profile GetProfile(int id) {
        FeatureManager.Check(“Profile");
        return _service.GetProfile(id);
    }
}
```

Масштабирование – дублирование

```
public class FeatureDecorator : IProfileService
{
    readonly IProfileService _service;

    public FeatureDecorator(IProfileService service)
        => _service = service;

    public Order GetOrder(int id)
    {
        FeatureManager.Check(“Profile");

        return _service.GetOrder(id);
    }
}
```

Декораторы

- Нет проблем добавлять
- Но на каждую задачу – свой декоратор!
- Их нужно генерировать автоматически

Castle.Core

- Библиотека создания динамических прокси-оберток
- Реализована через динамические модули .NET

Castle.DynamicProxy

```
public class FeatureAdvice : IInterceptor
{
    public void Intercept(IInvocation invocation)
    {
        FeatureManager.Check(GetFeatureName());

        invocation.Proceed();
    }
}
```

Связывание

```
static class FeatureAspect
{
    public static TYPE Apply<TYPE>()
        where TYPE : class
    {
        var generator = new ProxyGenerator();
        var proxy = generator.CreateClassProxy<TYPE>(new FeatureAdvice());
        return proxy;
    }
}
```

Создание прокси

```
var service = FeatureAspect.Apply<OrderService>();

service.GetOrder(id);
```

Name	Value
service	{Castle.Proxies.OrderServiceProxy}
Add item to watch	

Виртуальный метод

```
public class OrderService
{
    public virtual Order GetOrder(int id)
    {
        return _dbContext.Orders.Find(id);
    }
}
```

EF LazyLoading

```
public class Order
{
    public virtual Category Category { get; set; }
}
```

NHibernate ChangeTracking

```
public class Order
{
    public virtual string Name { get; set; }

    public virtual Category Category { get; set; }
}
```

Autofac

```
var builder = new ContainerBuilder();

builder.RegisterType<OrderService>()
    .EnableClassInterceptors()
    .InterceptedBy(typeof(FeatureAdvice));
builder.Register(c => new FeatureAdvice());


var container = builder.Build();
var willBeIntercepted = container.Resolve<OrderService>();
```

MS DI

```
sc.AddScoped<OrderService>();
sc.AddScoped<IOrderService>(sp =>
{
    var service = sp.GetRequiredService<OrderService>();
    var advice = sp.GetRequiredService<FeatureAdvice>();
    var generator = new ProxyGenerator();
    return generator.CreateClassProxyWithTarget(service, advice);
});
```

Async Interceptor (январь 2021)

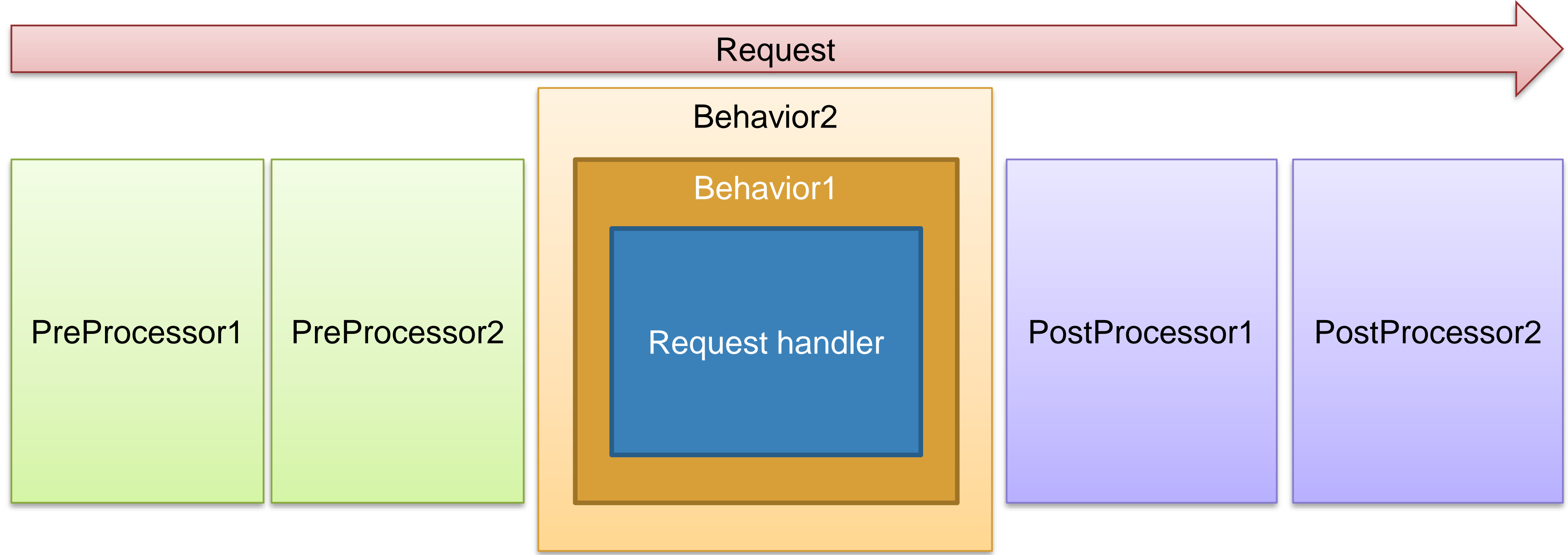
```
public class FeatureAsyncInterceptor : AsyncInterceptorBase
{
    protected override async Task InterceptAsync
        (IInvocation invocation, Func<IInvocation, Task> proceed)
    {
        await proceed(invocation);
    }

    protected override async Task<T> InterceptAsync<T>
        (IInvocation invocation, Func<IInvocation, Task<T>> proceed)
    {
        return await proceed(invocation);
    }
}
```

Обертки DynamicProxy

- Достоинства
  - Простота реализации
  - Поддержка DI контейнерами
  - Внедрение зависимостей в конструктор аспекта
  - Асинхронность

- Недостатки
  - Нет срезов

Пайплайн (конвейер)



MediatR - пайплайн обработки запроса

- PreProcessor
- Behavior
- PostProcessor

Унифицированный Request и Handler

```
public interface IRequest<TResponse>
{
}

public interface IRequestHandler<TRequest, TResponse>
    where TRequest : IRequest<TResponse>
{
    Task<TResponse> Handle(TRequest request);
}
```

Behavior

```
public class FeatureBehavior<TRequest, TResponse> :
    IPipelineBehavior<TRequest, TResponse>
    where TRequest : IFeatureRequest
{
     public async Task<TResponse> Handle(TRequest request,
        RequestHandlerDelegate<TResponse> next)
     {
        FeatureManager.Check(GetFeatureName());
        var response = await next();


        return response;
     }
}
```

Срез

Совет

EditEntityCommand

```
public class EditEntityCommand : IFeatureRequest
{
    public int Id { get; set; }

    public EditDto Dto { get; set; }
}
```

IFeatureRequest – пустой маркер

```
public interface IFeatureRequest
{
}
```

Как делать срезы – много интерфейсов

```
public class EditOrderCommand : IEditRequest, IOrderRequest
{
    public int Id { get; set; }

    public EditOrderDto Dto { get; set; }
}
```

Как добавить Behavior для кучи хендлеров

```
public interface IEditRequest : IFeatureRequest
```

Тесты через MediatR

```
[Fact]
public async Task Test() {
    // Arrange
    var mediatr = CreateMediatr(); // using DI Container

    // Act
    var result = await mediatr.Send(new EditEntityCommand{Id = 1});

    // Assert
    Assert.Equal(1, result.Id);
}
```

Любой (почти) DI контейнер

Container / Feature	ASP.NET Core DI	Autofac	Dryloc	DrylocZero	LightInject
Request Handler	✅	✅	✅	✅	✅
Void Handler	✅	✅	✅	✅	✅
Pipeline Behavior	✅	✅	✅	✅	✅
Pre-Processor	✅	✅	✅	✅	✅
Post-Processor	✅	✅	✅	✅	✅
Constrained Post-Processor	✅	✅	✅	✅	✅

https://github.com/jbogard/MediatR/wiki/Container-Feature-Support

MediatR под капотом

```
private Task<TResponse> Handle<TRequest, TResponse>(TRequest reques
    where TRequest : IRequest<TResponse>
{
    // обработчик
    var handler = _serviceProvider
        .GetRequiredService<IRequestHandler<TRequest, TResponse>>();

    // аспекты
    var behaviors = _serviceProvider
        .GetServices<IPipelineBehavior<TRequest, TResponse>>();
```

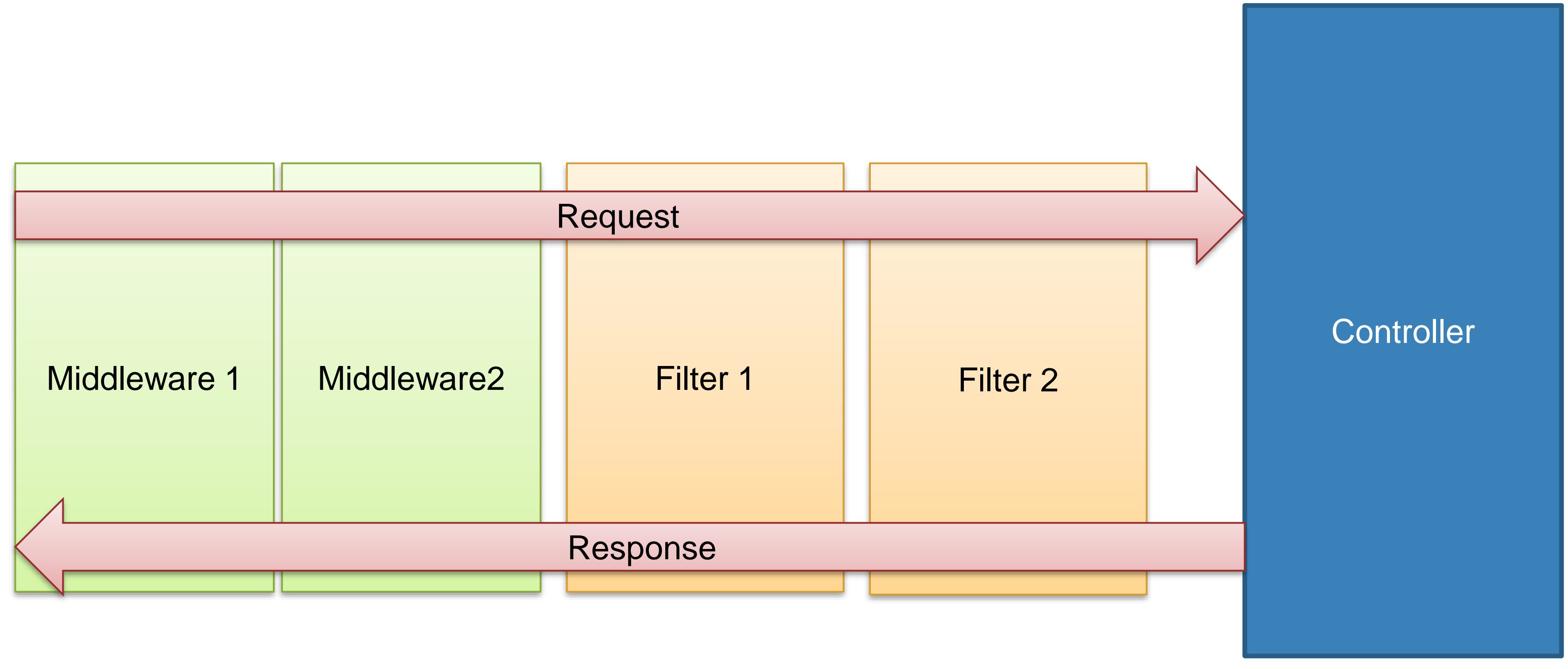
MediatR под капотом

```
    // начальное значение
    HandlerDelegate<TResponse> handlerDelegate =
        () => handler.HandleAsync(request);

    // пайплайн из делегатов
    var resultDelegate = middlewares
        .Aggregate(handlerDelegate, (next, middleware) =>
                () => middleware.HandleAsync(request, next));

    // запуск пайплайна
    return resultDelegate(); // Invoke
}
```

MediatR

- Достоинства
  - Асинхронность
  - DI
  - Тесты без HTTP запросов
  - Срезы через generic constraint

ASP.NET Core Pipeline



Middleware

Срез – все HTTP запросы

```
public class ExceptionHandlerMiddleware {
    public async Task Invoke(HttpContext httpContext) {
        try {
            // before
            await next(httpContext);
            // after
        }
        catch (EntityNotFoundException ex) {
            // 404 NotFound HTTP Code
        }
    }
}
```

Совет

Filter

```
public class SimpleResourceFilter : IActionFilter
{
    public void OnActionExecuting(ActionExecutingContext context)
    {
        // before
    }

    public void OnActionExecuted(ActionExecutedCont          ntext)
    {
        // after
    }
}
```

Совет

Использование фильтра

```
[SimpleResourceFilter]
public class HomeController : Controller
{
    [SimpleResourceFilter]
    public IActionResult Index()
    {
        return View();
    }
}
```

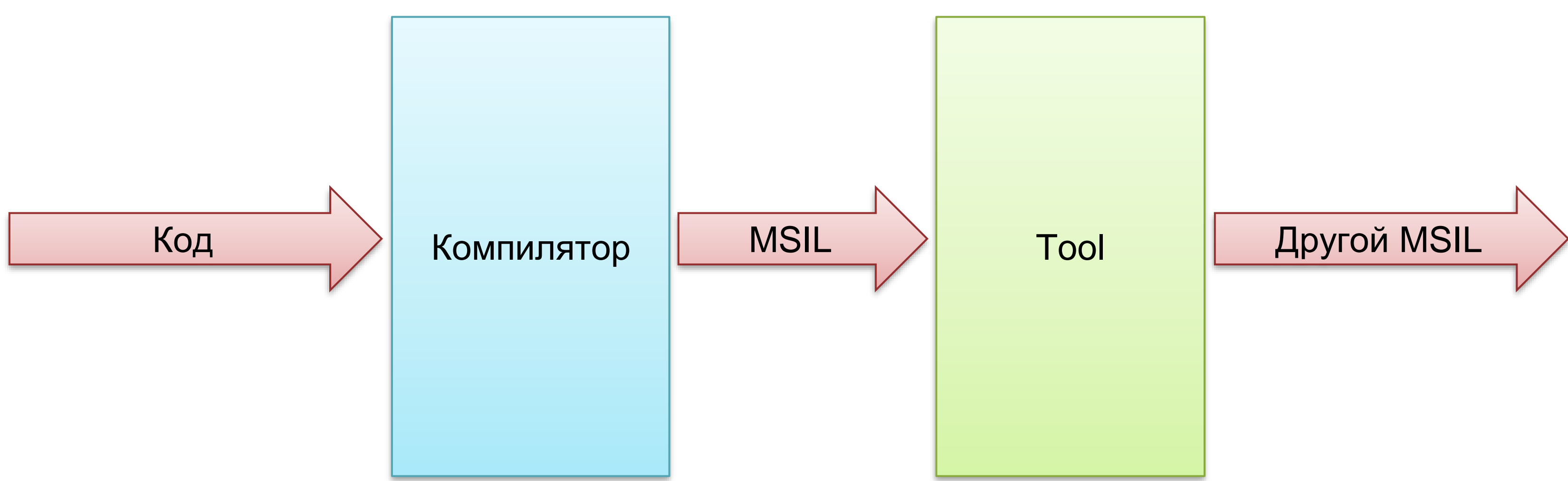
Интеграционные тесты (TestServer)

```
public class BasicTests :
    IClassFixture<WebApplicationFactory<Startup>>
{
    [Theory, InlineData("/Index")]
    public async Task Get_EndpointsReturnSuccess(string url) {
        // Arrange
        var client = _factory.CreateClient();

        // Act
        var response = await client.GetAsync(url);

        // Assert
        response.EnsureSuccessStatusCode(); // Status Code 200-299
    }
}
```

ASP.NET Core Pipeline

- Достоинства
  - Асинхронный код
  - DI
- Недостатки
  - Тесты через HTTP запросы
  - Нет срезов

Статическое связывание


Код
Компилятор
MSIL
Tool
Другой MSIL

Fody

Утилита, реализующая АОП с помощью перекомпиляции сборок .NET

- MethodDecorator
- MethodBoundaryAspect

MethodDecorator

```
public class FeatureAttribute : Attribute, IMethodDecorator
{
    public void OnEntry() {
        FeatureManager.Check(GetFeatureName());
    }

    public void OnExit() {
        // after
    }
}
```

Совет

Применение

```
public class OrderService
{
    [Feature]
    public Order GetOrder(int id)
    {
        return _dbContext.Orders.Find(id);
    }
}
```

Какой код получим

```
var attribute = Activator.CreateInstance(typeof(FeatureAttribute));
MethodBase method = GetMethod();
object[] args = new object[1] { 1 };
Order result;

attribute.Init(service, method, args);
attribute.OnEntry();
try {
    result = await service.GetOrder(1);
    attribute.OnExit();
}
catch (Exception exception) {
    attribute.OnException(exception);
    throw;
}
```

Fody

- Достоинства
- Недостатки
  - Синхронный
  - Нет DI для аспектов (ServiceLocator)
  - Нет срезов
  - Платный (5$ за рабочее место в мес)
  - ~~Замедление сборки (небольшое)~~

PostSharp - RegExp срез

```
[assembly: FeatureAspect(AttributeTargetTypes = "regex:.*Service")]
```

OnMethodBoundaryAspect – синхронный

```
public virtual void OnSuccess(
        MethodExecutionArgs args
)
```

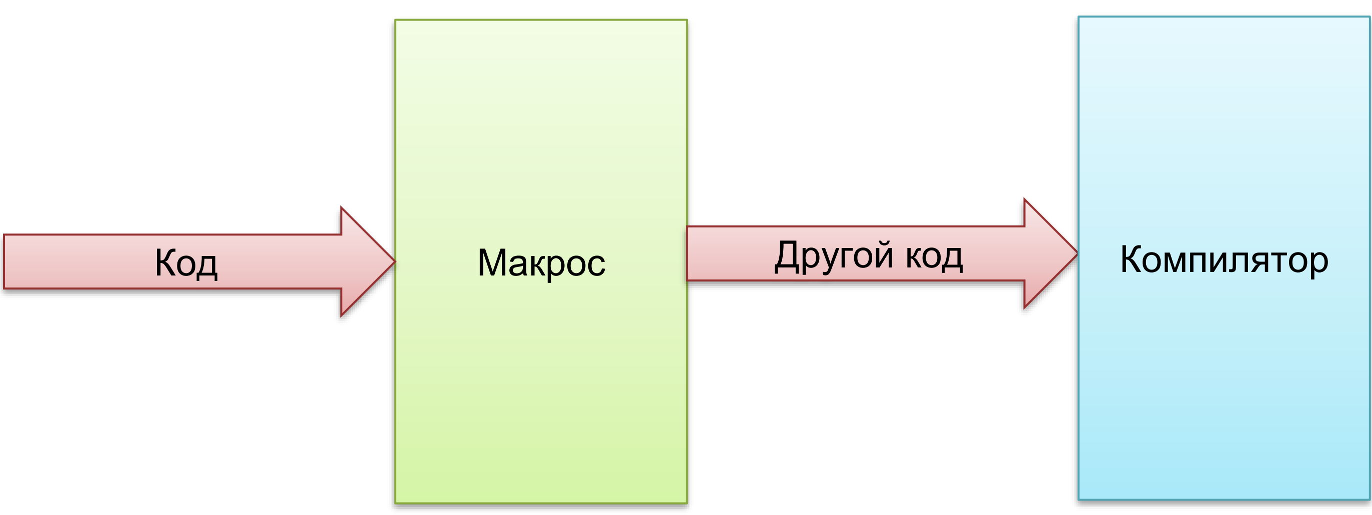
PostSharp

- Достоинства
    - Есть срезы

- Недостатки
    - Синхронный
    - Платный (600$ за рабочее место в год)
    - Ручная установка на рабочее место
    - Нет DI для аспектов
    - ~~Замедление сборки (небольшое)~~

#define true false /* счастливой отладки */

Макросы



Макросы в C#

- Any plan to support AOP directly in a compiler?
- Probably not.

Anders Hejlsberg

Nemerle – язык для CLR

```
macro FeatureAdvice(typeBuilder : TypeBuilder)
{
    foreach(member is ClassMember.Function in typeBuilder.Ast.GetMembers())
    {
        member.Body =
        <[
            FeatureManager.Check(GetFeatureName());
            def result = $(member.Body);
            result
        ]>
    }
}
```

Совет

Regex срезы

```
[assembly: RegexPointCutAspect("*Service", FeatureAdvice)]
```

Source generators

```
[Generator]
public class MyGenerator : ISourceGenerator
{
    public void Execute(GeneratorExecutionContext context)
    {
        var sourceBuilder = new StringBuilder();
        foreach (SyntaxTree tree in context.Compilation.SyntaxTrees)
        {
            sourceBuilder.AppendLine(@"Console.WriteLine(@"" - {tree.FilePath}"");");
        }

        context.AddSource("helloWorldGenerator",
                  SourceText.From(sourceBuilder.ToString()));
    }
}
```

Совет

Применение

Применение

```
[MyGenerator]
public class Person
{
    public string FirstName { get; set;}
    public Address Address { get; set;}
}
```

Source generators

```
public class Person
{
    public Person WithFirstName(string firstName)
    {
        FirstName = firstName;
        return this;
    }
    public Person WithAddress(Address address)
    {
        Address = address;
        return this;
    }
}
```

Source generators

Недостатки

- Нет срезов
- Дописывать, но не переписывать код

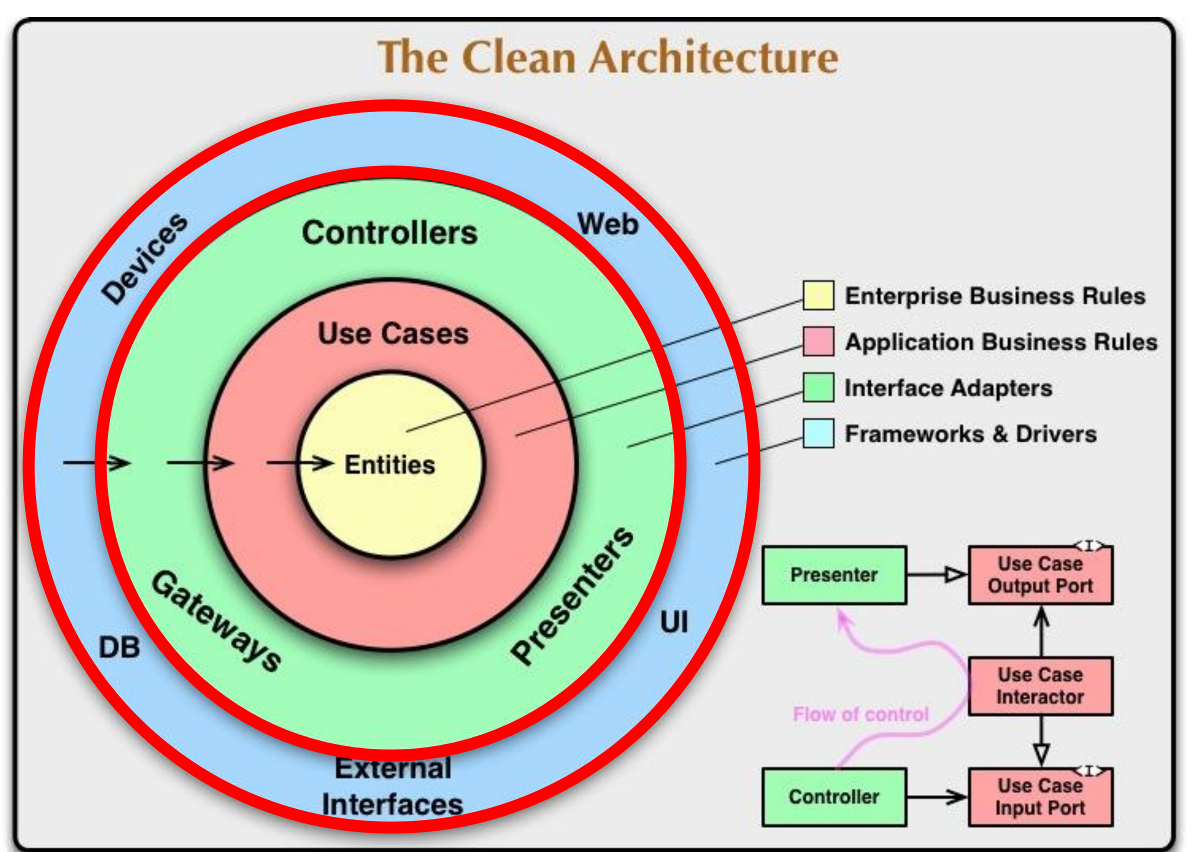
Мой опыт

- DynamicProxy – не использую
- Fody, PostSharp - не использую

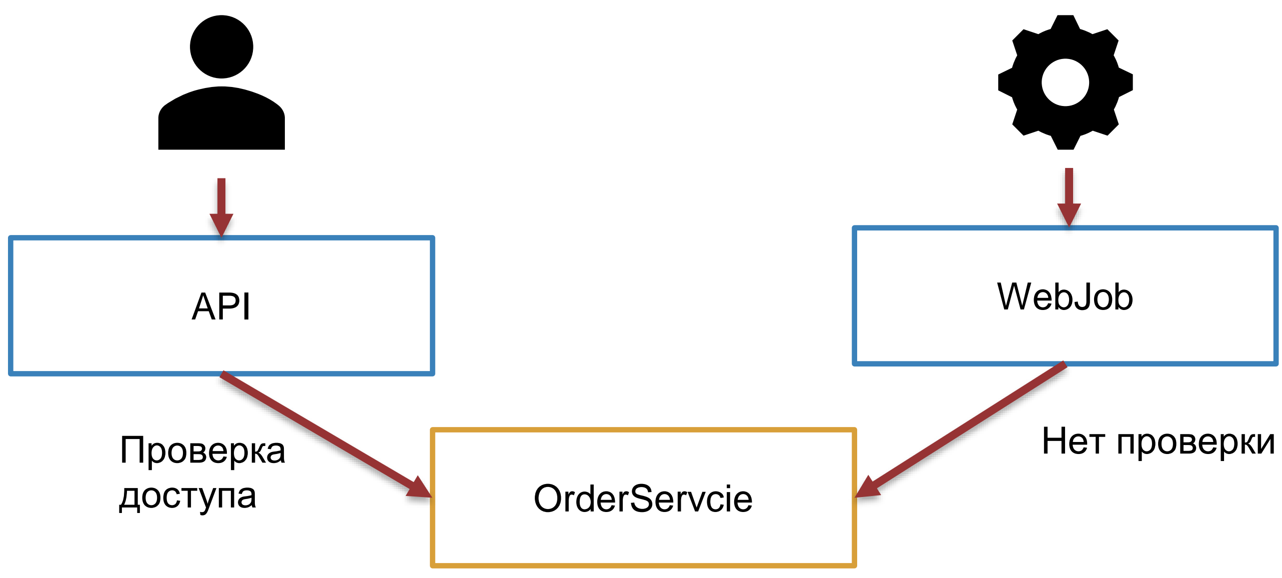
- ASP.NET Middleware (или глобальный фильтр)
  - HttpCode для ошибки (NotFound, а не InternalServerError)
  - CorrelationId – отдать в хедере Http Response

- MediatR – application логика, а не web
  - Feature Flags – для разработчиков
  - Блокировка интеграций – для админов
  - Есть ли доступ к сущности

Настройка аспектов – на уровне Frameworks



В разных хостах – разные аспекты



Собираем все вместе

- Динамическое связывание
  1. Castle.DymanicProxy (DI)
  2. **Пайплайн MediatR**
  3. Пайплайн ASP.NET Core

- Статическое связывание
  4. Изменение MSIL Fody
  5. Изменение MSIL PostSharp
  6. Генерация кода Source Generators

Материалы о пайплайнах MediatR

- Jimmy Bogard. Vertical Slice Architecture
  https://youtu.be/SUiWfhAhgQw

- Максим Аршинов. Быстрорастворимое проектирование
  https://habr.com/ru/company/jugru/blog/447308/

Главная мысль

Мир изменился!
Вчера аспекты были магией компиляции или динамических прокси.
Сегодня – часть веб-фреймворка ASP.NET и MediatR.

MediatR

- 60 млн скачиваний (год назад было 20 млн)
- eShopOnWeb - официальный сэмпл Microsoft
- Попробуйте)

https://www.udemy.com/course/cqrs-architecture-csharp-ru

Development › Software Engineering › C#

Как улучшить Enterprise архитектуру при помощи CQRS

8 преимуществ CQRS хендлеров и вертикальных слайсов по сравнению с привычной слоистой архитектурой

5.0 ★★★★★ (9 ratings) 64 students

Created by Denis Tsvettsikh

Last updated 2/2021 Russian

Спасибо за внимание

Денис Цветцих
Software Architect
DevBrothers

den.tsvettsih@yandex.ru
@den_tsvettsikh
https://github.com/denis-tsv