

Приложение АОП в .net



Немного рекламы себя

- 1 Отвечаю за вертикаль .NET в компании Artsofte
- 2 Помогаю разрабатывать NOCODE.RU
- 3 Читаю лекции по программированию в УрФУ

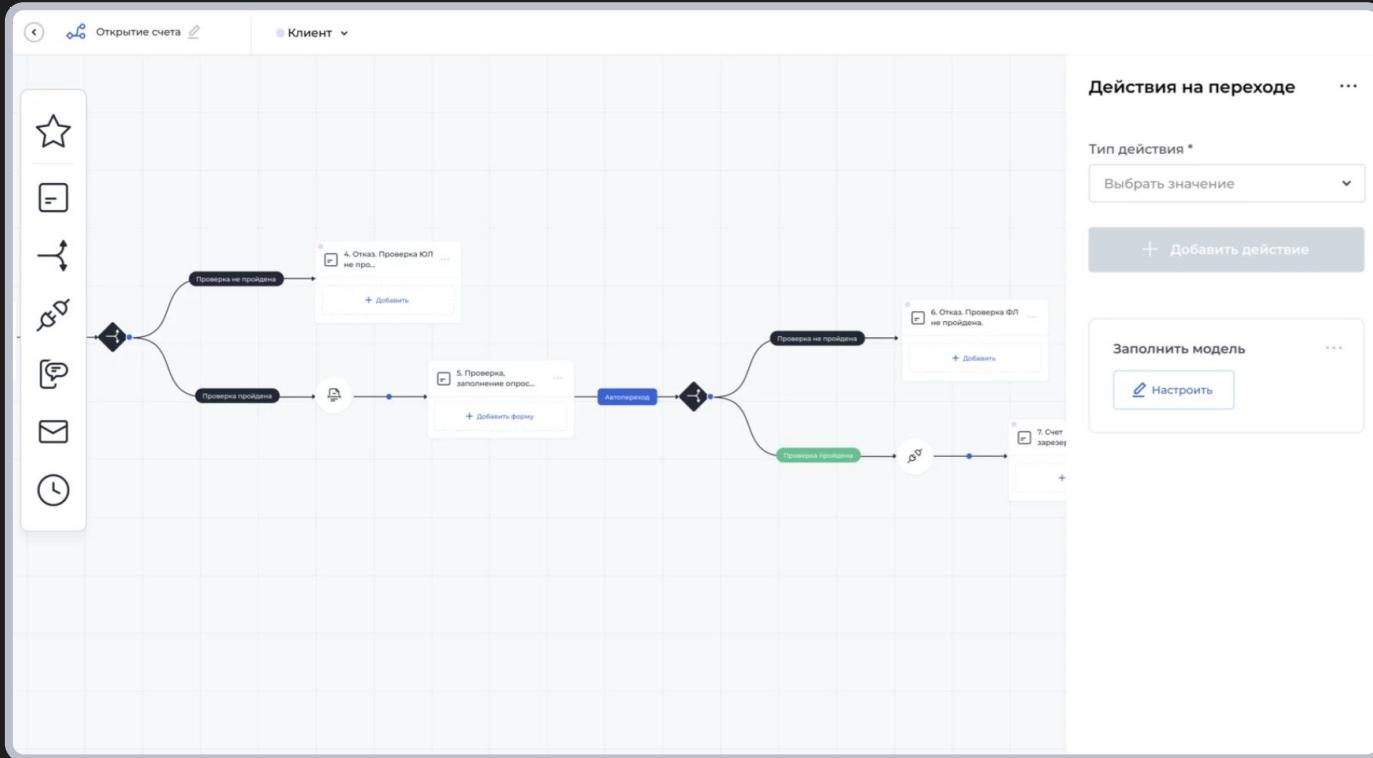
Подписывайтесь
и ставьте лайки

▶ [@ArtsofteEducation](#)

◀ [@dimoner1](#)



NOCODE



Как оно всё вместе?

Монорепозиторий



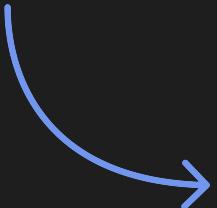
Команды в монорепозитории



Что в коде приводило к проблемам

Пример проблемного кода

Нужный технический код, который не влияет
на бизнес-логику, но захламляет всё



```
/// <inheritdoc />
0+1 usages
public async Task DeleteByTransitionIdListAsync(IEnumerable<Guid> transitionIdList)
{
    var logEnd = LogHelper.StartLog(transitionIdList);

    if (transitionIdList is null || !transitionIdList.Any())
    {
        return;
    }

    var transaction = _actionRepository.BeginTransactionOrDefault();
    var searchFieldList = new SearchFieldModel[]
    {
        new(nameof(ActionDal.LinkEntityId), transitionIdList)
        {
            SearchFieldComparerType = SearchFieldComparerType.IN
        }
    };
    await _actionRepository.DeleteByFieldListAsync(searchFieldList, transaction);

    logEnd();
}
```

Примеры «проблемного» кода

Логирование

```
public async Task<string[]> GetSimplePassListAsync()
{
    var logTitle = $"{GetType().Name}.{nameof(GetSimplePassListAsync)}";
    LogHelper.LogInfoWithPrefix(text:$"Start: {logTitle}");
    try
    {
        var result :string[] = await _simplePasswordRepository.GetAllSimplePasswordAsync();
        LogHelper.LogInfoWithPrefix(text:$"End: {logTitle} Result: {JsonAb.Serialize(result)}");
        return result;
    }
    catch (Exception e)
    {
        LogHelper.LogErrorWithPrefix(text:$"End ERROR: {logTitle} Result: {e.Message}");
        throw;
    }
}
```

Транзакции

Вход и выход из multitenant

Мониторинг

Примеры «проблемного» кода

Логирование

Транзакции

Вход и выход из multitenant

Мониторинг

```
public async Task<string[]> GetSimplePassListAsync()
{
    var logTitle = $"{GetType().Name}.{nameof(GetSimplePassListAsync)}";
    var result = new string[0];
    var dbTransaction = null;
    var id = Guid.Empty;

    try
    {
        var isNotTransactionExist = dbTransaction == null;
        if (isNotTransactionExist)
        {
            dbTransaction = await _stepRepository.BeginTransactionAsync();
        }

        var id : Guid = await _stepRepository.InsertAsync(stepDal, dbTransaction);

        if (isNotTransactionExist)
        {
            await dbTransaction.CommitAsync();
        }
    }
    catch (Exception ex)
    {
        result = new string[0];
        var errorMessage = $"Ошибка при выполнении запроса: {ex.Message}";
        await _logger.LogError(errorMessage);
    }
    finally
    {
        if (dbTransaction != null)
        {
            await dbTransaction.Dispose();
        }
    }
}

public async Task<Guid> CreateStepAsync(StepDal stepDal, DbTransaction dbTransaction = null)
{
    var isNotTransactionExist = dbTransaction == null;
    if (isNotTransactionExist)
    {
        dbTransaction = await _stepRepository.BeginTransactionAsync();
    }

    var id : Guid = await _stepRepository.InsertAsync(stepDal, dbTransaction);

    if (isNotTransactionExist)
    {
        await dbTransaction.CommitAsync();
    }

    return id;
}
```

Примеры «проблемного» кода

Логирование

Транзакции

Вход и выход из multitenant

Мониторинг

```
public async Task<string[]> GetSimplePassListAsync()
{
    var logTitle = $"{GetType().Name}.{nameof(GetSimplePassListAsync)}";
    public async Task<Guid> CreateStepAsync(StepDal stepDal, DbTransaction dbTransaction = null)
    {
        var isNotTransactionExist = dbTransaction == null;
        if (isNotTransactionExist)
            dbTransaction = await _dbConnection.BeginTransaction();
        await _stepDal.CreateStepAsync(stepDal, dbTransaction);
        return stepDal.Id;
    }
}

public async Task DeleteSubscriberAsync(string connectionId)
{
    _multiTenantConfigurationProvider.BlockConfiguration();
    try
    {
        await _subscriberModelRepository.DeleteAsync(connectionId);
    }
    catch (Exception e)
    {
        LogHelper.LogError(e);
    }
    finally
    {
        _multiTenantConfigurationProvider.UnblockConfiguration();
    }
}
```

Примеры «проблемного» кода

Логирование

Транзакции

Вход и выход из multitenant

Мониторинг

```
public async Task<string[]> GetSimplePassListAsync()
{
    var logTitle = $"{GetType().Name}.{nameof(GetSimplePassListAsync)}";
    var errorMessage :string = _generateTextMessageValidationManager
        .GenerateLockoutMessage(user,
        _dbSettingAccessor.Settings.IdentityLockoutSettings.Enable,
        out var blockEndTime :DateTimeOffset?);

    var logEvent = new MonitoringEvent(
        eventName: MonitoringEventConstants.BlockUser,
        user.Id.ToString(), LifetimeType.Short,
        eventType: MonitoringEventType.Start);
    _monitoringConnectionService.SendEvent(logEvent);

    throw new UserLogInLockedErrorException(errorMessage, blockEndTime);
}

public async Task<Guid> CreateStepAsync(StepDal stepDal, DbTransaction dbTransaction = null)
{
    var isNotTransactionExist = dbTransaction == null;
    if (isNotTransactionExist)
    {
        _multiTenantConfigurationProvider.BlockConfiguration();
        try
        {
            var logTitle = $"{GetType().Name}.{nameof(CreateStepAsync)}";
            var errorMessage :string = _generateTextMessageValidationManager
                .GenerateLockoutMessage(user,
                _dbSettingAccessor.Settings.IdentityLockoutSettings.Enable,
                out var blockEndTime :DateTimeOffset?);

            var logEvent = new MonitoringEvent(
                eventName: MonitoringEventConstants.BlockUser,
                user.Id.ToString(), LifetimeType.Short,
                eventType: MonitoringEventType.Start);
            _monitoringConnectionService.SendEvent(logEvent);

            throw new UserLogInLockedErrorException(errorMessage, blockEndTime);
        }
        finally
        {
            _multiTenantConfigurationProvider.UnblockConfiguration();
        }
    }
}
```

Декоратор для “группы” классов

- Если заголовок есть, то работа метода должна трекаться (event sourcing)
- Нужно делать что-то до и после работы метода



Декорирование группы классов (через ООП)

Много повторяющегося кода, надо
для каждого класса писать декоратор, а потом
делать сложные регистраций

```
services.TryAddScoped<StepRepository>();
services.TryAddScoped<IStepRepository>(implementationFactory: service =>
{
    var repository = service.GetRequiredService<IStepRepository>();

    // если мы не передали заголовок версии, то выходим
    var isNotVersionAction = service.CheckIsVersionAction();
    if (isNotVersionAction)
    {
        return repository;
    }

    return new DecorateRepository(repository);
});
```

```
public interface IStepRepository;

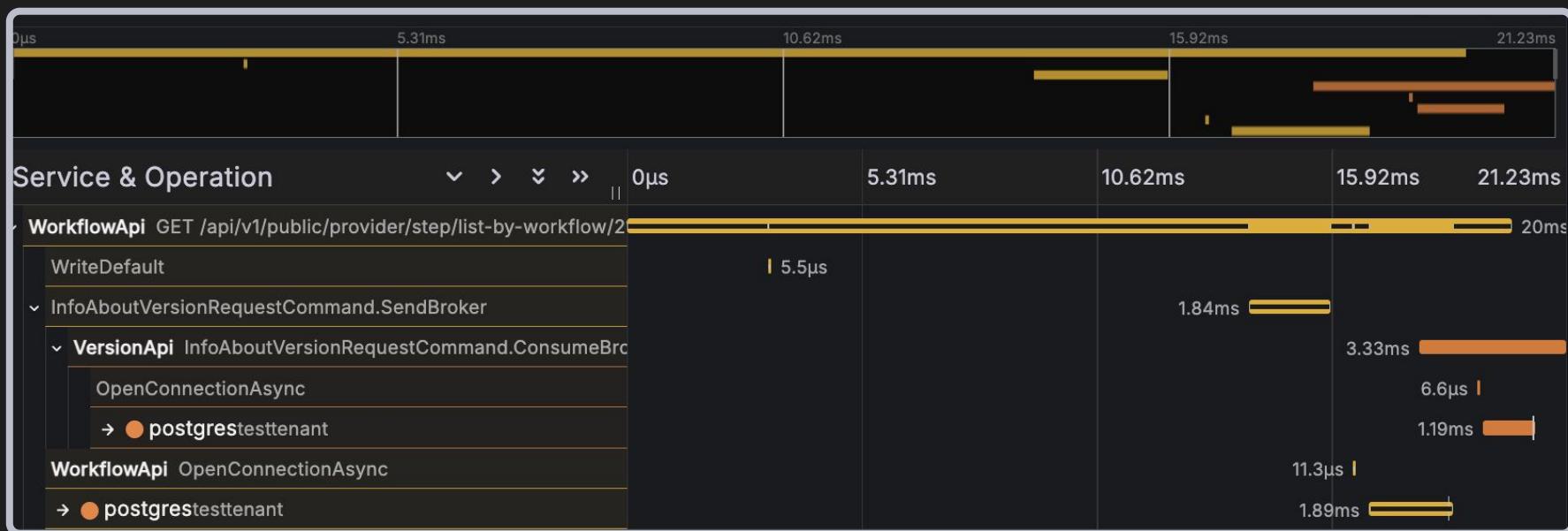
☒ 6 usages  ☒ new *
public class StepRepository : IStepRepository;

☒ new *
public class DecorateRepository : IStepRepository
{
    private readonly IStepRepository _stepRepository;

    ☒ new *
    public DecorateRepository(IStepRepository stepRepository)
    {
        _stepRepository = stepRepository;
    }
}
```

Использование для добавления трейсов

Типичным решением является опять же интеграция в работу Controller, например, через Filter и Item (пример движка перехвата)



Использование для добавления трейсов

Минусы

- Код повторяется между методами
- Долго так писать
- Массово изменять этот код - боль

```
private async Task OpenConnectionAsync(DbConnection dbConnection)
{
    var activityStatusCode = ActivityStatusCode.Ok;
    var activity :Activity = _serviceDiagnostic.StartActivity(nameof(OpenConnectionAsync), ActivityKind.Internal);
    try
    {
        await dbConnection.OpenAsync();
    }
    catch
    {
        activityStatusCode = ActivityStatusCode.Error;
        throw;
    }
    finally
    {
        activity?.SetStatus(activityStatusCode);
        activity?.Stop();
    }
}
```

Использование для добавления трейсов

Минусы

- Аллокация на func
- Может быть замыкание
- Лишний код

```
private async Task OpenConnectionAsync(DbConnection dbConnection)
{
    var activityStatusCode = ActivityStatusCode.Ok;
    var activityActivity = _serviceDiagnostic.StartActivity(nameof(OpenConnectionAsync), ActivityKind.Internal);
    try
    {
        await dbConnection.OpenAsync();
    }
    catch
    {
    }
}
```

```
private async Task OpenConnectionAsync(DbConnection dbConnection)
{
    await _serviceDiagnostic.StartActivity(func: async () =>
    {
        await dbConnection.OpenAsync();
    }, nameof(OpenConnectionAsync), ActivityKind.Internal); // Task
}
```

[CallerMemberName]

Управление инфраструктурой из CoreLib

- Переход с Start на StartAsync CoreTeam 100500 конфликтов (это самый простой и банальный пример)

```
private async Task<List<TableDependency>> GetTableSqlHierarchyAsync(IDbConnection connection)
{
    var connection = await connectionFactory.CreateConnectionAsync();
    var tableHierarchy :IEnumerable<TableDependency> = await connection.GetTableHierarchy();

    var tableSqlHierarchy :List<TableDependency> = tableHierarchy.Where(x :TableDependency)
        .Where(x :TableDependency)
        .Where(x :TableDependency);

    return tableSqlHierarchy;
}
```

Задачи, которые вытекают из этого кода



А как делают другие?

Использование декораторов в Angular

```
@Component({
  selector: 'sado-cl-cabinet-menu',
  templateUrl: './cabinet-menu.adaptive.component.html',
  styleUrls: ['./styles/cabinet-menu.master.adaptive.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush,
})
export class CabinetMenuAdaptiveComponent extends CabinetMenuBaseComponent implements AfterViewInit {
  @Output()
  public closeMenuEmitter: EventEmitter<void> = new EventEmitter();

  @ViewChild('container')
  public containerElement!: ElementRef<HTMLDivElement>;

  public isDefaultLogo: boolean = false;
```



Сразу видишь всё, что относится к поведению компоненты



Аналогичный опыт можно почерпнуть из UI/UX, плохо, когда пользователю надо что-то запоминать между страницами

Использование декораторов в Spring

Пример использования интерсепторов



```
@Service  
public class InvoiceService {  
  
    @Transactional(propagation = Propagation.REQUIRES_NEW)  
    public void createPdf() {  
        // ...  
    }  
}
```

- AОП -

АОП — аспектно-ориентированное программирование



```
public async Task<string[]> GetSimplePassListAsync()
{
    var logTitle = $"{GetType().Name}.{nameof(GetSimplePassListAsync)}";
    LogHelper.LogInfoWithPrefix(text: $"Start: {logTitle}");
    try
    {
        var result: string[] = await _simplePasswordRepository.GetAllSimplePasswordAsync();
        LogHelper.LogInfoWithPrefix(text: $"End: {logTitle} Result: {JsonAb.Serialize(result)}");
        return result;
    }
    catch (Exception e)
    {
        LogHelper.LogErrorWithPrefix(text: $"End ERROR: {logTitle} Result: {e.Message}");
        throw;
    }
}
```

Пример Аспекта

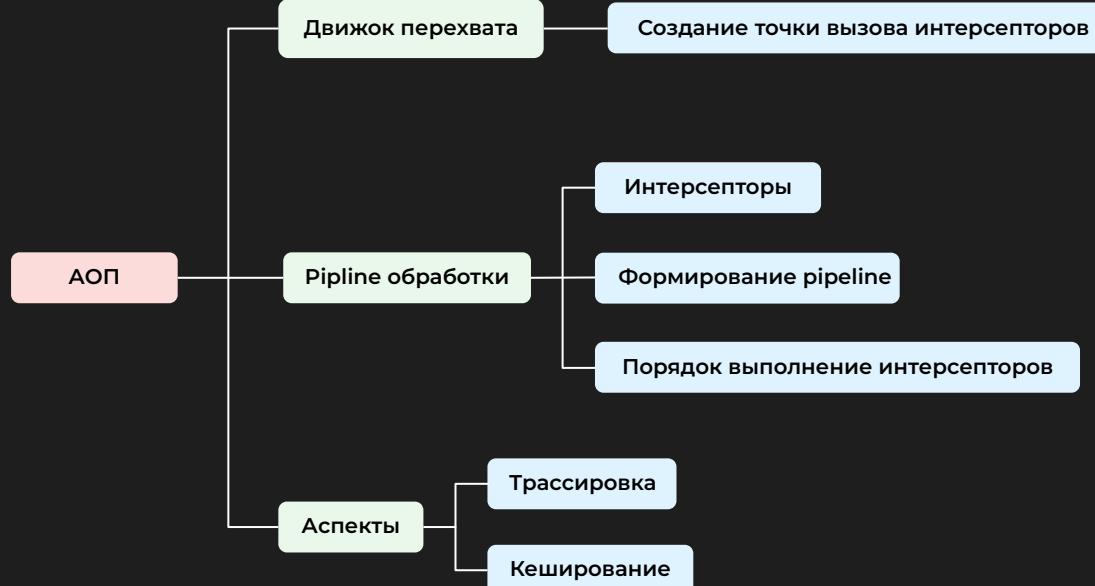
```
public async Task<IActionResult> TestCommonUseCaseAsync([FromBody] ExampleInternalRequest dto)
{
    var result:ExampleInternalResponse = await _exampleUseCaseManager.ControllerUseCaseAsync(dto);

    return Ok(result);
}
```

```
public async Task<ExampleInternalResponse> ControllerUseCaseAsync(ExampleInternalRequest dto)
{
    var logEnd:Func<ExampleInternalResponse,...> = LogHelper.StartLogWithReturnValue<ExampleInternalResponse>(dto);
    var result:int = await _exampleManager.CreateAsync(test: new ExampleDal() { Value = dto.Body });
    var response = new ExampleInternalResponse { Id = result };
    return logEnd(response);
}
```

```
public async Task<int> CreateAsync(ExampleDal test)
{
    var logEnd:Func<int,int> = LogHelper.StartLogWithReturnValue<int>(test);
    var res:int = await _testRepository.InsertOrUpdateAsync(test);
    return logEnd(res);
}
```

Из чего состоит АОП и его реализация



Применение

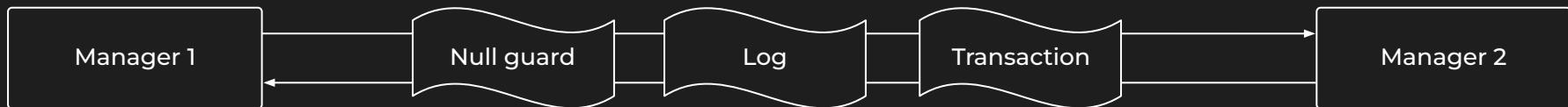
```
[Tracing]
✉ 0+3 usages  ↳ D Egorov +1 *
public async Task<string[]> GetSimplePassListAsync()
{
    var result :string[] = await _simplePasswordRepository
        .GetAllSimplePasswordAsync(); // Task<string[]>
    return result;
}
```

Бизнес-код

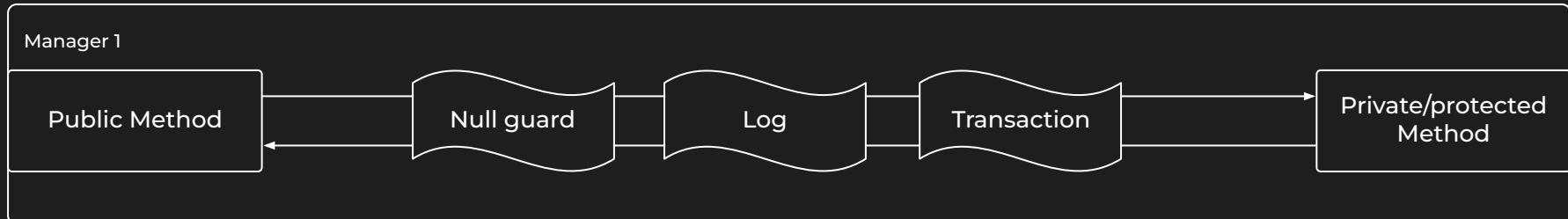
```
public void InterceptSynchronousHandlerVoid(IInvocation invocation)
{
    var activity = _serviceDiagnostic.StartActivity(invocation.Method.Name, ActivityKind.Internal);
    var activityStatusCode = ActivityStatusCode.Ok;
    try
    {
        invocation.Proceed();
    }
    catch
    {
        activityStatusCode = ActivityStatusCode.Error;
        throw;
    }
    finally
    {
        activity?.SetStatus(activityStatusCode);
        activity?.Stop();
    }
}
```

Сервис, инкапсулирующий логику

Типы вызова методов



Между классами



Между методами в одном классе

АОП в .NET - Filter Controller

Плюсы

- 1 Определяем атрибут
- 2 По атрибуту как маркеру вызывается сервис
- 3 Логика сервиса легко подключается и инкапсулируется

```
public class TestController : Controller
{
    [TestResourceFilter]
    public IActionResult Test()
    {
        return Ok();
    }
}
```

```
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme, Policy = RequireClientPolice.Name)]
public class ChatPublicController : BasePublicController
```

АОП в .NET - Diagnostic System

Пример перехвата начала и конца логики

```
private void StopActivity(Activity activity, HttpContext httpContext)
{
    if (activity.Duration == TimeSpan.Zero)
        activity.SetEndTime(DateTime.UtcNow);
    HostingApplicationDiagnostics.WriteDiagnosticEvent<HttpContext>((DiagnosticSource) this._diagnosticListener, name: "Microsoft.AspNetCore.Hosting.HttpRequestIn.Stop", httpContext);
    activity.Stop();
}
```

```
public void OnNext(KeyValuePair<string, object> value)
{
    if (value.Key == "Microsoft.AspNetCore.Hosting.HttpRequestIn.Start")
    {
        Console.WriteLine(value.Key);
        Console.WriteLine(value.Value);
        Console.WriteLine();
    }

    if (value.Key == "Microsoft.AspNetCore.Hosting.HttpRequestIn.Stop")
    {
        Console.WriteLine(value.Key);
        Console.WriteLine(value.Value);
        Console.WriteLine();
    }
}
```

АОП в .NET - Mock

Плюсы

- + Быстро и удобно описывает методы
- + Не нужно делать декоратор под каждый класс!!!

```
❖ IL code
private void InitializeInstance()
{
    // Determine the set of interfaces that the proxy object should additionally implement.
    var additionalInterfaceCount = this.AdditionalInterfaces.Count;
    var interfaces = new Type[1 + additionalInterfaceCount];
    interfaces[0] = typeof(IMocked<T>);
    this.AdditionalInterfaces.CopyTo(index: 0, interfaces, arrayIndex: 1, additionalInterfaceCount);

    this.instance = (T)ProxyFactory.Instance.CreateProxy(
        typeof(T),
        interceptor: this,
        interfaces,
        this.constructorArguments); // object
}
```

```
WorkflowApplicationFactory.ConnectMonitoringServiceMock // Mock<IMonitoringConnectionService>
    .Setup(expression: x :IMonitoringConnectionService => x.SendEvent(It.IsAny<MonitoringEvent>()));

WorkflowApplicationFactory.ConnectDocumentSchemaServiceMock // Mock<IConnectDocumentSchemaService>
    .Setup(expression: x :IConnectDocumentSchemaService => x.CreateDocumentSchemaAsync(request: It.IsAny<CreateDocumentSchemaBrokerRequest>()))
    .ReturnsAsync(new CreateDocumentSchemaBrokerResponse()
    {
        Id = ObjectId.GenerateNewId()
    });

```

xUnit DataAttribute

```
[Theory]
[TestMethod]
    connectionString: "Provider=SQLOLEDB;Server=(local);Database=TestDatabase;Trusted_Connection=yes;",
    selectStatement: $"select {nameof.FirstName} , ${nameof.LastName} from Users"]
    & new *
public void OleDbTests(string FirstName, string LastName)
{
    Assert.Equal(expected: "Peter Beardsley", actual: $"{FirstName} {LastName}");
}
```

Spring Query

```
public interface UserRepository extends JpaRepository<User, Long> {
    @Query("SELECT u FROM User u WHERE u.username = ?1")
    User findByUsername(String username);
}
```

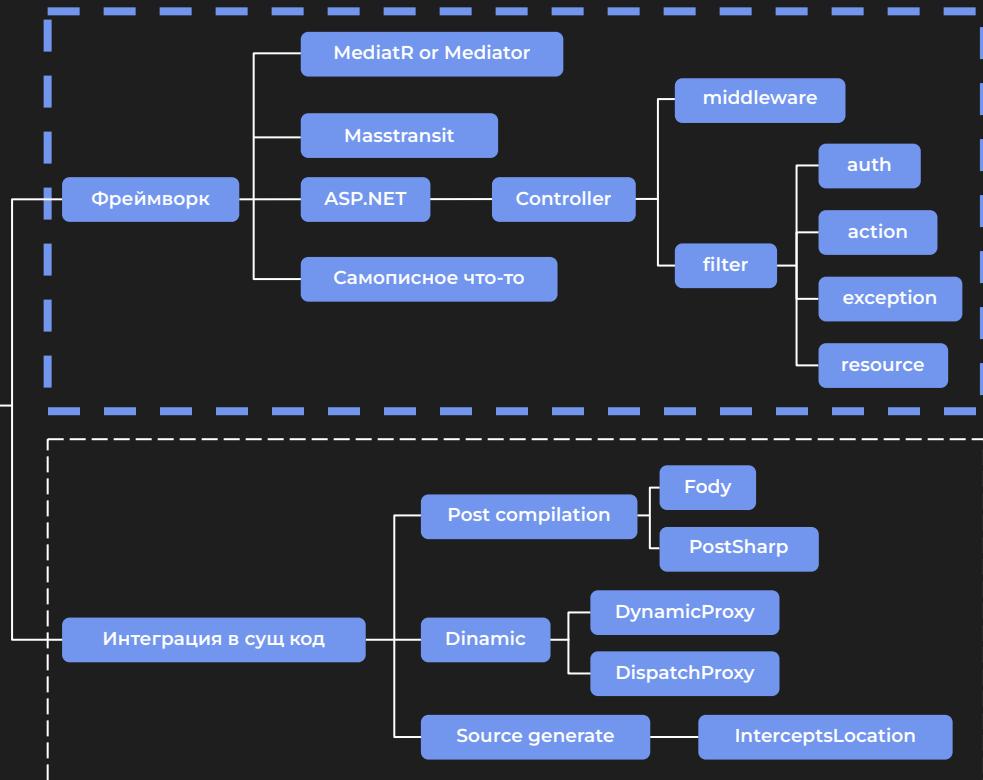
Какими путями можно внедрять АОП в код

Движок для создания точек перехвата

Фреймворк

- + Не сложно реализовать сам фреймворк, но легко туда впилить АОП
- Нужен Фреймворк

Варианты

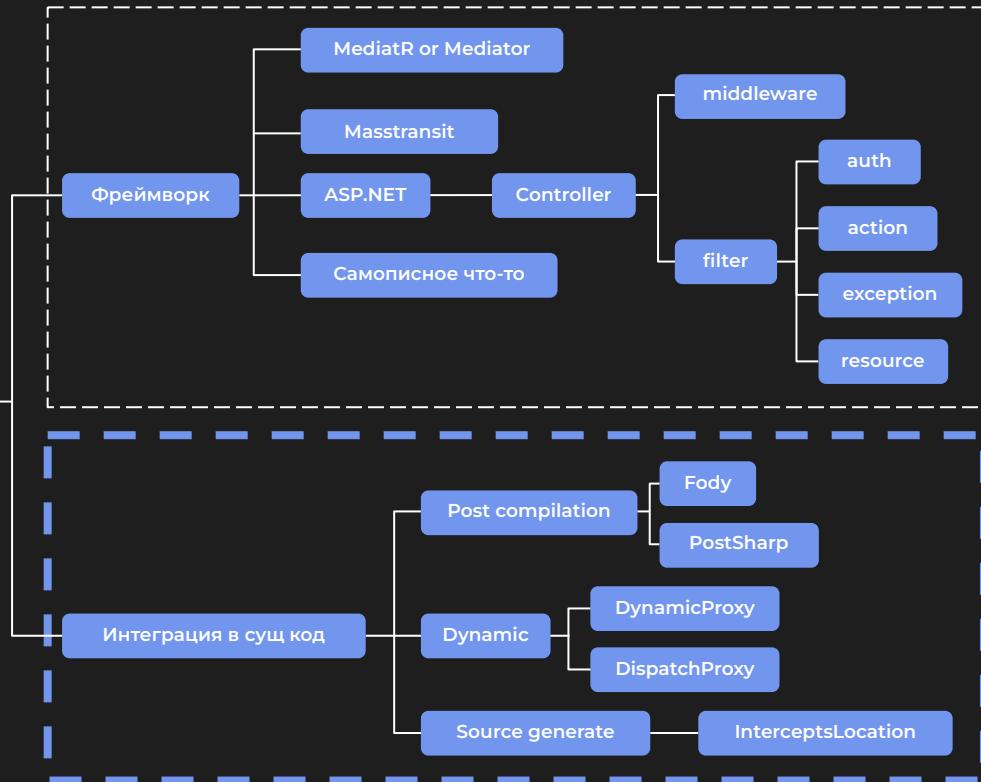


Движок для создания точек перехвата

Интеграция в код

- + Подходит для любой архитектуры
- Много нюансов при реализации

Варианты



Куда идет встраивание нашего кода ?



Manager1.Method1 вызывает Manager2.Method1

- 1) Внутри метода, который вызывает (Source generation Interceptors)
- 2) Прокси между менеджерами, обертка над вызываемым (DynamicProxy, DispatchProxy)
- 3) Внутри метода, который вызывают (Fody)

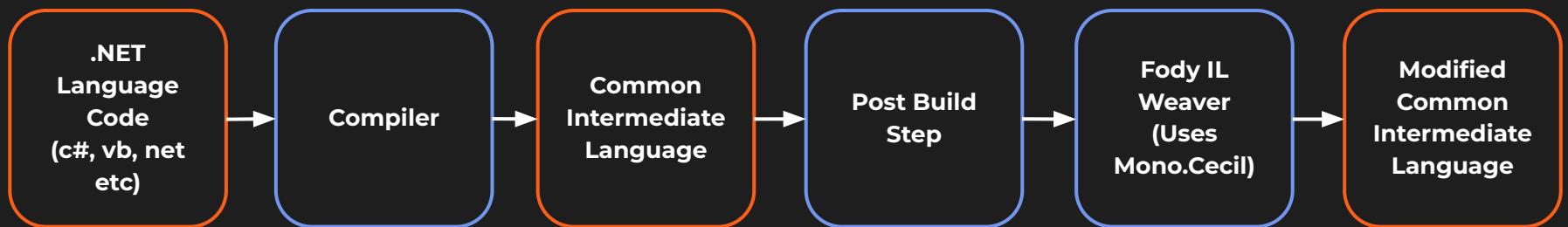
Технические требования



Post Compilation — Fody



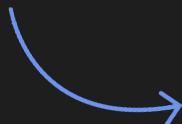
Fody



Плюсы

- + Хорошая поддержка сообществом
- + Много готовых расширений
- + Перехват любого типа методов
- + Может использоваться не только для перехвата методов

★ Дополнительно мы использовали библиотеку MethodBoundaryAspect.Fody



Fody: пример

```
File.cs[LogExample]
[LogExample]
✉ 3 usages
public class FodyExample
{
    ✉ 2 usages
    public async Task Action(int current)
    {
        Console.WriteLine(HelperGetInt());
    }
    ✉ 1 usage
    private int HelperGetInt() => 1;
}
```

```
File.cs[LogExample]
OnEntry Action
OnEntry MoveNext
OnEntry HelperGetInt
OnExit HelperGetInt
1
OnExit MoveNext
OnExit Action
```

Fody: до и после

Было

5 классов на запрос,
у каждого вызывает по 2 метода

```
public void Action(int current)
{
    Console.WriteLine(nameof(Examples));
}
```

Fody: до и после

Стало

$5 * 2 * 3 = 30$ классов на запрос



```
public void Action(int current)
public void Action(int current)
{
    object[] objArray = new object[1]{ (object) current };
    MethodExecutionArgs args = new MethodExecutionArgs();
    args.Arguments = objArray;
    MethodBase b01Ca1D2B354AbeC2B = MethodInfo._MethodInfo_8D48D52327
    args.Method = b01Ca1D2B354AbeC2B;
    FodyExample fodyExample1 = this;
    args.Instance = (object) fodyExample1;
    CustomLogAttribute customLogAttribute = new CustomLogAttribute();
    customLogAttribute.OnEntry(args);
    if (args.FlowBehavior == FlowBehavior.Return)
        return;
    FodyExample fodyExample2 = this;
    try
    {
        fodyExample2._executor_Action(current);
    }
    catch (Exception ex)
    {
        args.Exception = ex;
        customLogAttribute.OnException(args);
        switch (args.FlowBehavior)
        {
            case FlowBehavior.Continue:
                return;
            case FlowBehavior.Return:
                return;
            default:
                throw;
        }
    }
    customLogAttribute.OnExit(args);
}
```

Сам перехватчик

```
public sealed class CacheAttribute : OnMethodBoundaryAspect
public sealed class CacheAttribute : OnMethodBoundaryAspect
{
    public override void OnEntry(MethodExecutionArgs args)
    {
        // ...
    }

    public override void OnExit(MethodExecutionArgs args)
    {
        // ...
    }

    public override void OnException(MethodExecutionArgs args)
    {
        // ...
    }
}
```



Первый способ: ContinueWith

Минусы

- Не подходит, если нужно ожидание
- Остается проблема с Task<T>

```
public override void OnExit(MethodExecutionArgs args)
{
    if (args.ReturnValue is Task t && !args.ReturnValue.GetType().IsGenericType)
    {
        t.ContinueWith( continuationAction: () =>
        {
            Console.WriteLine($"OnExit {Thread.CurrentThread.ManagedThreadId}");
        }, TaskContinuationOptions.ExecuteSynchronously);
    }
    else
    {
        Console.WriteLine($"OnExit {Thread.CurrentThread.ManagedThreadId}");
    }
}
```

Второй способ: TaskCompletionSource

Минусы

- Производительность
- Если класс Singleton — будут проблемы с многопоточностью
- Остается проблема с Task<T>

```
private readonly TaskCompletionSource _source = new TaskCompletionSource();

public override void OnExit(MethodExecutionArgs args)
{
    if (args.ReturnValue is Task t && !args.ReturnValue.GetType().IsGenericType)
    {
        args.ReturnValue = _source.Task;
        t.ContinueWith(continuationFunction: async task =>
        {
            await task;
            Console.WriteLine($"OnExit {Thread.CurrentThread.ManagedThreadId}");
            _source.SetResult();
        }, TaskContinuationOptions.ExecuteSynchronously);
    }
    else
    {
        Console.WriteLine($"OnExit {Thread.CurrentThread.ManagedThreadId}");
    }
}
```

```
if (t.IsCompletedSuccessfully)
{
    //System.Threading.Tasks.VoidTaskResult - объект, который возвращает Task.Result без результата
    var isResultTask = !genericArguments.First().FullName.Equals("System.Threading.Tasks.VoidTaskResult");

    if (isResultTask)
    {
        var returnValueTask = arg.ReturnValue as Task;
        var result = type.GetProperty("Result")?.GetValue(returnValueTask);
        arg.ReturnValue = result;
    }
    else
    {
        //Если таска не возвращает результат, она возвращает AsyncStateMachineBox
        //Смысла в этом, думаю, нету
        arg.ReturnValue = null;
    }
    HandleExit(arg);
}
```

Делать так
не надо



Третий способ: Dynamic

Определяем, что это Task<T>

Создаём delegate под вызов

Логика создания delegate

Мы попали в Task<T>

Мы попали в Task<T>
с ожиданием

```
private static bool IsGenericTask(Type returnType)
{
    return returnType != typeof(void)
        && typeof(Task).IsAssignableFrom(returnType)
        && returnType.GetTypeInfo().IsGenericType;
```

3 способ: Dynamic

Определяем, что это Task<T>

Создаём delegate под вызов

Логика создания delegate

Мы попали в Task<T>

Мы попали в Task<T>
с ожиданием

```
private static bool IsGenericTask(Type returnType)
{
    return returnType != typeof(void)
}

private static readonly ConcurrentDictionary<Type, GenericAsyncHandler> GenericAsyncHandlers = new();

private static GenericAsyncHandler GetHandler(Type returnType)
{
    GenericAsyncHandler handler = GenericAsyncHandlers.GetOrAdd(returnType, CreateHandler);
    return handler;
}
```

3 способ: Dynamic

Определяем, что это Task<T>

Создаём delegate под вызов

Логика создания delegate

Мы попали в Task<T>

Мы попали в Task<T>
с ожиданием

```
private static bool IsGenericTask(Type returnType)
{
    return returnType != typeof(void)
        && returnType.GetGenericTypeDefinition() == typeof(Task<T>);
}

private static readonly ConcurrentDictionary<Type, GenericAsyncHandler> GenericAsyncHandlers = new();
private static GenericAsyncHandler GetHandler(Type returnType)
{
    private delegate void GenericAsyncHandler(MethodExecutionArgs methodExecutionArgs);
    private static GenericAsyncHandler CreateHandler(Type returnType)
    {
        Type taskReturnType = returnType.GetGenericArguments()[0];
        MethodInfo methodInfo = HandleMethodInfo.MakeGenericMethod(taskReturnType);
        return (GenericAsyncHandler)methodInfo.CreateDelegate(typeof(GenericAsyncHandler));
    }
}
```

3 способ: Dynamic

Определяем, что это Task<T>

Создаём delegate под вызов

Логика создания delegate

Мы попали в Task<T>

Мы попали в Task<T>
с ожиданием

```
private static bool IsGenericTask(Type returnType)
{
    return returnType != typeof(void);

private static readonly ConcurrentDictionary<Type, GenericAsyncHandler> GenericAsyncHandlers = new();
    ↳ usage

private static GenericAsyncHandler GetHandler(Type returnType)
{
    ↳ usage

private delegate void GenericAsyncHandler(MethodExecutionArgs methodExecutionArgs);
    ↳ usage

private static void HandleAsyncResult<TResult>(MethodExecutionArgs invocation)
{
    if (invocation.ReturnValue is Task<TResult> taskWithResult)
    {
        taskWithResult.ContinueWith(async val:Task<TResult> =>
        {
            var res:TResult = await val;
            Console.WriteLine(res);
            Console.WriteLine($"OnExit Task<T>");
        });
    }
}
```

3 способ: Dynamic

Определяем, что это Task<T>

Создаём delegate под вызов

Логика создания delegate

Мы попали в Task<T>

Мы попали в Task<T>
с ожиданием

```
private static bool IsGenericTask(Type returnType)
{
    return returnType != typeof(void);

private static readonly ConcurrentDictionary<Type, GenericAsyncHandler> GenericAsyncHandlers = new();
    ↳ 1 usage

private static GenericAsyncHandler GetHandler(Type returnType)
{
    ↳ 1 usage

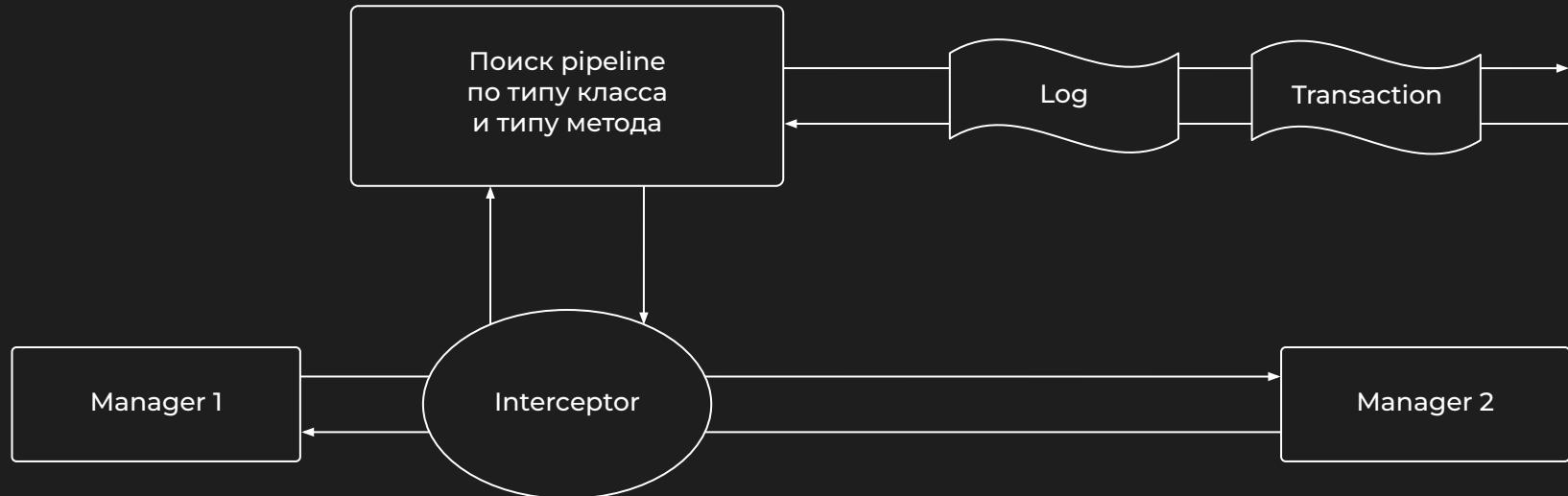
private delegate void GenericAsyncHandler(MethodExecutionArgs methodExecutionArgs);
    ↳ 1 usage

private static void HandleAsyncWithResult<TResult>(MethodExecutionArgs invocation)
{
    ↳ 1 usage
    {
        if (invocation.ReturnValue is Task<TResult> taskWithResult)
        {
            invocation.ReturnValue = HandlerAsync(taskWithResult, invocation);
        }
    }
    ↳ 1 usage
}

private static async Task<T> HandlerAsync<T>(Task<T> task, MethodExecutionArgs invocation)
{
    var res:T = await task;
    return res;
}
```

Формирование pipeline

Pipeline + interceptor chain



А как делает MediatR? Черпаем вдохновение

```
public override Task<TResponse> Handle(  
    IRequest<TResponse> request,  
    IServiceProvider serviceProvider,  
    CancellationToken cancellationToken)  
{  
    // все эти вычисления каждый раз  
    return serviceProvider  
        .GetServices<IPipelineBehavior<TRequest, TResponse>>() // находим все обработчики между вызовами  
        .Reverse() // определение порядка через порядок регистрации !!!  
        .Aggregate( // ЗАМЫКАНИЕ  
            () => serviceProvider  
                .GetRequiredService<IRequestHandler<TRequest, TResponse>>() // ПЛЮС что есть DI  
                .Handle((TRequest) request, cancellationToken),  
            (Func<RequestHandlerDelegate<TResponse>, IPipelineBehavior<TRequest, TResponse>, RequestHandlerDelegate<TResponse>>)  
            ((next :RequestHandlerDelegate<TResponse> , pipeline) => (RequestHandlerDelegate<TResponse>) (() => pipeline.Handle((TRequest) request, next, cancellationToken))))();  
}
```

1

Аллокации/замыкания

2

Каждый раз пайплайн в рантайме

Хотелось бы не создавать pipeline каждый раз (ASP)

```
public RequestDelegate Build()
{
    RequestDelegate requestDelegate = (RequestDelegate) (context =>
    {
        Endpoint endpoint = context.GetEndpoint();
        if (!context.Response.HasStarted)
            context.Response.StatusCode = 404;
        context.Items[(object) "_RequestUnhandled"] = (object) true;
        return Task.CompletedTask;
    });
    for (int index = this._components.Count - 1; index >= 0; --index)
        requestDelegate = this._components[index](requestDelegate);
    return requestDelegate;
}
```

```
public class Middleware
{
    private readonly RequestDelegate _next;

    public Middleware(RequestDelegate next)
    {
        this._next = next;
    }

    public async Task InvokeAsync(HttpContext context)
    {
        var timer = context.RequestServices.GetRequiredService<ITimer>();
        await _next.Invoke(context);
    }
}
```

- 1 ASP.NET: один раз при Build формируем pipeline

- 2 Контекст передается через параметры (доступ в DI через параметр контекста)

★ Но у него один конвейер, а как сделать несколько?

Реализация через runtime

Анализ всех
Assembly при старте

Dictionary<Type, Dictionary<MethodInfo, List<BaseAbankingInterceptorAttribute>>>

По аналогии с ASP
создаёт вложенную
цепочку методов

Dictionary<Type, FrozenDictionary<MethodInfo, InterceptDelegate>>

C .NET 8 FrozenDictionary



Реализация (Как middleware)

```
if (typeof(Task).IsAssignableFrom(methodType.ReturnType) && methodType.ReturnType.GetTypeInfo().IsGenericType)
{
    var zero = (InterceptTask)(static async arg :MethodExecutionArgs =>
    {
        if (arg.ReturnValue is Task task)
        {
            await task;
        }
    });

    foreach (var interceptor :BaseAbankingInterceptorAttribute? in interceptorAttributeList)
    {
        // тут кеш
        var interceptorInstance :object? = Activator.CreateInstance(interceptor.InterceptorType, zero);

        zero = (InterceptTask)interceptor
            .InterceptorType?
            .GetMethod(nameof(ICustomInterceptor.InterceptSynchronousHandlerRes))!
            .MakeGenericMethod(methodType.ReturnType.GetGenericArguments()[0])
            .CreateDelegate(typeof(InterceptTask), interceptorInstance)!;
    }

    dictionary.Add(methodType, zero);
}
```

Вызов внутри Fody

```
private static async Task<T> HandlerAsync<T>(Task<T> task, MethodExecutionArgs invocation)
private static async Task<T> HandlerAsync<T>(Task<T> task, MethodExecutionArgs invocation)
{
    if (InterceptPipelineCreator.Dictionary.TryGetValue(
        invocation.Method.DeclaringType,
        out var methodList :Dictionary<MethodInfo,InterceptTask>? ))
    {
        if (methodList.TryGetValue((MethodInfo)invocation.Method, out var intercept))
        {
            await intercept(invocation);
        }
        else
        {
            return await task;
        }
    }
    else
    {
        return await task;
    }

    if (task.IsCompleted)
    {
        return task.Result;
    }

    throw new Exception();
}
```

Вид интерсептора (OnExit)

```
public class Test1Interceptor : IBankingInterceptor
{
    private readonly InterceptTask _interceptTask;
    private readonly InterceptVoid _interceptVoid;

    public Test1Interceptor() { }
    public Test1Interceptor(InterceptVoid interceptVoid) { _interceptVoid = interceptVoid; }
    public Test1Interceptor(InterceptTask interceptTask) { _interceptTask = interceptTask; }

    public void InterceptSynchronousHandlerVoid(MethodExecutionArgs arg) { _interceptVoid(arg); }
    public async Task InterceptSynchronousHandler(MethodExecutionArgs arg) { await _interceptTask(arg); }

    □ 0+1 usages
    public async Task InterceptSynchronousHandlerRes<TResult>(MethodExecutionArgs arg)
    {
        await _interceptTask(arg);
        if (arg.ReturnValue is Task<TResult> returnValue)
        {
            var resData :TResult = await returnValue;
            Console.WriteLine(resData);
        }
    }
}
```

Стек трейс

- DebuggerStepThroughAttribute отдаёт отладчику указание о сквозной обработке кода (вместо обработки изнутри)

```
[DebuggerStepThrough]
[DebuggerStepThrough]
[0+1 usages & degorov +1 *]
protected override async Task<TResult> InterceptSynchronousHandler<TResult>(Func<Task<TResult>> next, IInvocation invocation)
{
    var actionResult :Func<Task<...>> = next;
    var interceptor :List<IAbankingInterceptor> = GetCurrentForMethodList(invocation);

    for (var c :int = interceptor.Count - 1; c >= 0; c--)
    {
        actionResult = interceptor[c].InterceptSynchronousHandler(actionResult, invocation);
    }

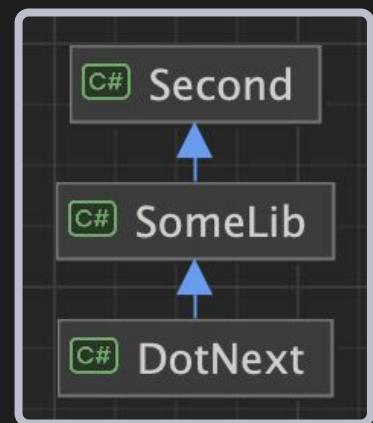
    if (actionResult == null)
    {
        var res :TResult = await next();
        return res;
    }

    return await actionResult();
}
```

А почему мой класс не интерсептируется?

- Assembly Lazy, соответственно, если на момент построения дерева не было обращения к типу, то сборка просто не подгрузится!

```
foreach (var name in Assembly.GetEntryAssembly()!.GetReferencedAssemblies()!){  
    var res :Assembly = Assembly.Load(name);  
    foreach (var nameInner in res.GetReferencedAssemblies()!)  
    {  
        Assembly.Load(nameInner);  
    }  
}
```



Просто добавили это в CoreLib

Fody DI

Как получить DI/внешнее состояние?

1 Чтобы работать с DI, нужно получить доступ к IServiceProvider

2 Делать это через Fody мы не можем, так как сам обработчик создается через new

```
public void Action(int current)
{
    object[] objArray = new object[1]{ (object) current };
    MethodExecutionArgs args = new MethodExecutionArgs();
    args.Arguments = objArray;
    MethodBase b01Ca1D2B354AbeC2B = MethodInfo._MethodInfo_8D48D52327;
    args.Method = b01Ca1D2B354AbeC2B;
    FodyExample fodyExample1 = this;
    args.Instance = (object) fodyExample1;
    CustomLogAttribute customLogAttribute = new CustomLogAttribute();
```

Состояние

DI

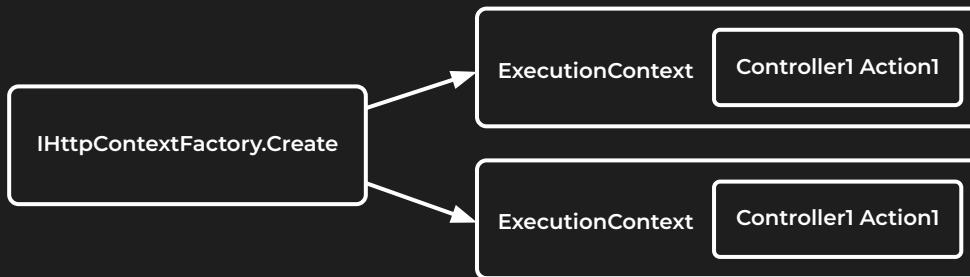
Static AsyncLocal

Пример решения проблемы из ASP

!

Сам ASP.NET позволяет получать доступ из статики к текущему контексту запроса:

```
public class HttpContextAccessor : IHttpContextAccessor
{
    #nullable disable
    ◆ IL code
    private static readonly AsyncLocal<HttpContextAccessor.HttpContextHolder> _httpContextCurrent = new AsyncLocal<HttpContextAccessor.HttpContextHolder>();
```



Тоже самое с Activity

```
private static readonly AsyncLocal<Activity> s_current;
```

```
public static Activity? Current
{
    get => Activity.s_current.Value;
    set
    {
        if (!Activity.ValidateSetCurrent(value))
            return;
        Activity.SetCurrent(value);
    }
}
```

```
public void InterceptSynchronousHandlerVoid(IInvocation invocation)
{
    var activity = _serviceDiagnostic.StartActivity(invocation.Method.Name, ActivityKind.Internal);
    var activityStatusCode = ActivityStatusCode.Ok;
    try
    {
        invocation.Proceed();
    }
    catch
    {
        activityStatusCode = ActivityStatusCode.Error;
        throw;
    }
    finally
    {
        activity?.SetStatus(activityStatusCode);
        activity?.Stop();
    }
}
```

Serilog

```
public static class LogContext
{
    // IL code
    private static readonly AsyncLocal<EnricherStack?> Data = new AsyncLocal<EnricherStack>();
```

- ❖ Хранит свойства, которые подставляет в строку лога в AsyncLocal

AsyncLocal

Если кратко

- 1) ExecutionContext - это механизм, с помощью которого реализуется AsyncLocal<T>
- 2) ExecutionContext иммутабельный, он не копируется, если не изменился!!!

```
internal static void SetLocalValue(
    IAsyncLocal local,
    object newValue,
    bool needChangeNotifications)
{
    ExecutionContext executionContext = Thread.CurrentThread._executionContext;
```

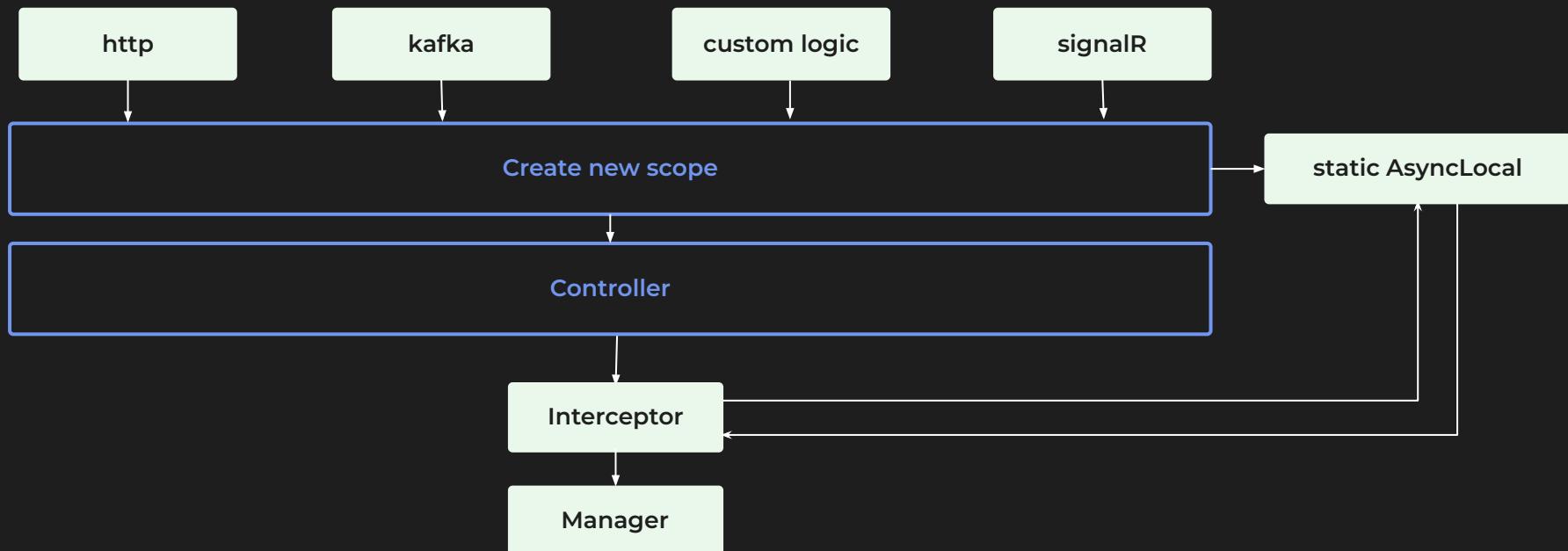
```
Thread.CurrentThread._executionContext =
    isFlowSuppressed || !AsyncLocalValueMap.IsEmpty(asyncLocalValueMap)
    ? new ExecutionContext(asyncLocalValueMap, array, isFlowSuppressed)
    : (ExecutionContext) null;
```

Выдержка из его работы



Как на самом деле работает
Async/Await в C# (Часть 6)

Реализация



Почему AsyncLocal может нас подвести

- ★ Метод Void завладеет ExecutionContext из метода MainAsync
- ★ Метод Task без async завладеет ExecutionContext из метода MainAsync



Почему AsyncLocal может нас подвести

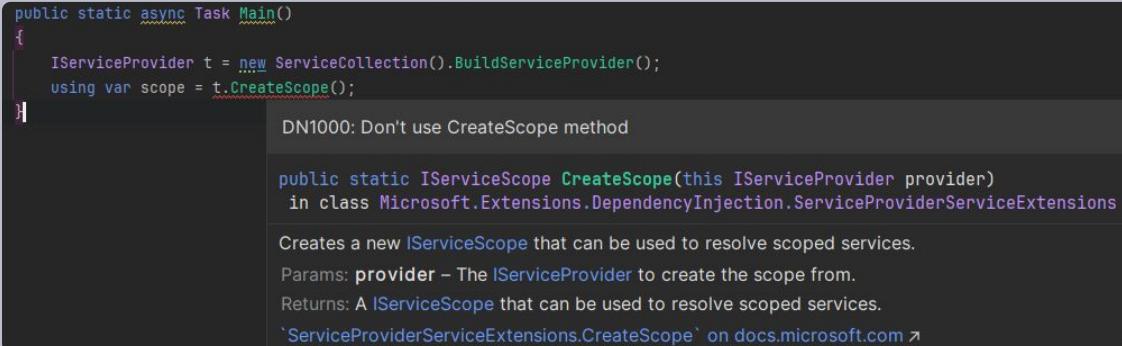
```
private static readonly AsyncLocal<IServiceProvider> _asyncLocal = new();
private static readonly AsyncLocal<IServiceProvider> _asyncLocal = new();
private async Task ActionTask()
{
    var scopeOne = _serviceProvider.CreateScope();
    _asyncLocal.Value = scopeOne.ServiceProvider;
    var scopeTwo = _serviceProvider.CreateScope();
    _asyncLocal.Value = scopeTwo.ServiceProvider;
}
```

Вывод

Если мы хотим получать доступ до DI через AsyncLocal,
то нужно запретить CreateScope, **проксировать тоже не вариант**

Как запретить создавать scope

```
private static void Analyze(SyntaxNodeAnalysisContext context)
{
    var memberAccess = (MemberAccessExpressionSyntax)context.Node;
    if (memberAccess.Name.ToString() == "CreateScope" && memberAccess.Expression is IdentifierNameSyntax identifier)
    {
        if (context.SemanticModel.GetSymbolInfo(identifier).Symbol is ILocalSymbol variableSymbol && variableSymbol.Type.ToString() != "IServiceProvider")
        {
            var diagnostic = Diagnostic.Create(_createScopeMethodUsageIsProhibitedDescriptor, memberAccess.GetLocation(), identifier.Identifier.Text);
            context.ReportDiagnostic(diagnostic);
        }
    }
}
```



Нас начали засыпать в Jira вот таким

```
13:00:36:036 LEVEL:[Information] Task faulted. Exceptions:  
InvalidProgramException => Common Language Runtime detected an invalid program.
```



Итог с Fody:

- 1 Баги с IL
(аналогичное видели в dapper)
- 2 Накладные расходы: +3 класса на каждый метод (Object Pool туда не добавишь)
- 3 Не очень так и бесшовно его устанавливать приходится (править каждый csproj)
- 4 DI: если запретить создавать scope разработчикам — мы не умели и не решились
- 5 Async/await: можно сделать
- 6 Тип перехвата: любой

Вывод

Из-за совокупности этих проблем мы применили его только на части ненагруженных сервисов и **двинулись искать новое решение**

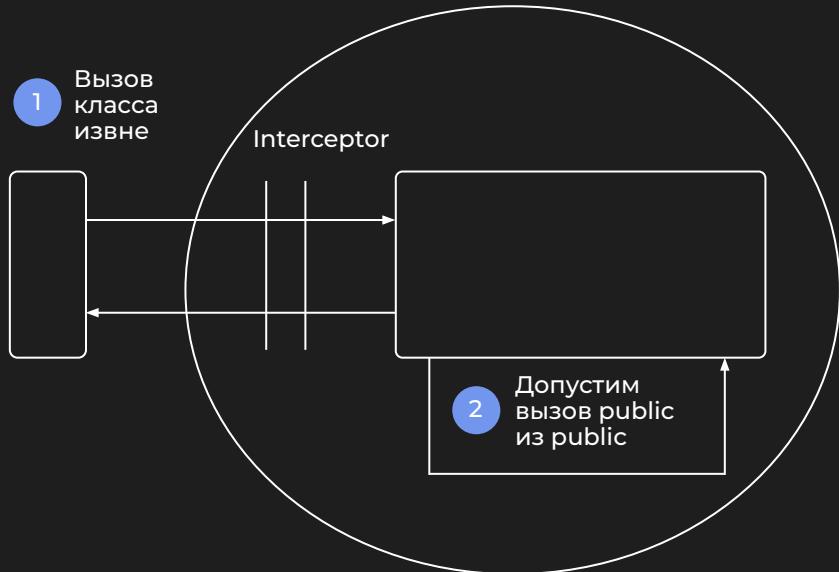
Генерация Type в Runtime — DynamicProxy и DispatchProxy



Proxy Interceptor



крутая статья,
как этим пользоваться



Важно

В отличие от посткомпиляции методы, реализованные через такие интерсепторы, работают только на публичные вызовы!!

Главная разница DynamicProxy и DispatchProxy

	DynamicProxy	DispatchProxy
Возможность перехвата	По интерфейсу, по классу с virtual methods.	По интерфейсу
Дополнительные возможности	<ol style="list-style-type: none">1) Можно настраивать фильтр на то, какие методы интерсептировать и какими перехватчиками из указанных2) Можно использовать слабые ссылки3) Можно явно кидать ошибку, если пометили Private метод (рантайм)	Нет сложных опций в коробке

Как это работает ? (DispatchProxy)

```
var allAssembly :int = AppDomain.CurrentDomain.GetAssemblies().Length;  allAssembly: 110
var test1 :IDispatchProxyExample = DispatchProxyLoggingDecorator<IDispatchProxyExample>.Decorate( target: new DispatchProxyExample());  test1: generatedProxy_1
var allAssembly2 :int = AppDomain.CurrentDomain.GetAssemblies().Length;  allAssembly2: 113
var test2 :IDispatchProxyExample = DispatchProxyLoggingDecorator<IDispatchProxyExample>.Decorate( target: new DispatchProxyExample());  test2: generatedProxy_1
var allAssembly3 :int = AppDomain.CurrentDomain.GetAssemblies().Length;  allAssembly3: 113
var test11 :IDispatchProxyExample2 = DispatchProxyLoggingDecorator2<IDispatchProxyExample2>.Decorate( target: new DispatchProxyExample2());  test11: generatedProxy_2
var allAssembly4 :int = AppDomain.CurrentDomain.GetAssemblies().Length;  allAssembly4: 113
```

>  [110] = {System.Reflection.RuntimeAssembly} ProxyBuilder, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null

▽  DefinedTypes = {System.RuntimeType[]} System.RuntimeType[2]

>  [0] = {System.RuntimeType} generatedProxy_1

>  [1] = {System.RuntimeType} generatedProxy_2

Как это работает ? (DispatchProxy)

Надо исследовать Type

```
foreach (MethodInfo runtimeMethod in iface.GetRuntimeMethods())
{
    if (runtimeMethod.IsVirtual && !runtimeMethod.IsFinal)
    {
        int count = this._methodInfos.Count;
        this._methodInfos.Add(runtimeMethod);
    }
}
```

```
foreach ( PropertyInfo runtimeProperty in iface.GetRuntimeProperties())
{
    DispatchProxyGenerator.ProxyBuilder.PropertyAccessorInfo propertyAccessor;
    if (runtimeProperty.GetMethod != (MethodInfo) null)
        dictionary1[runtimeProperty.GetMethod] = propertyAccessor;
    if (runtimeProperty.SetMethod != (MethodInfo) null)
        dictionary1[runtimeProperty.SetMethod] = propertyAccessor;
}
```

Как это работает ? (DispatchProxy)

```
// 1. Создаем динамическую сборку
AssemblyName assemblyName = new AssemblyName("MyDynamicAssembly");
AssemblyBuilder assemblyBuilder = AssemblyBuilder.DefineDynamicAssembly(assemblyName, AssemblyBuilderAccess.Run);
// 2. Создаем модуль
ModuleBuilder moduleBuilder = assemblyBuilder.DefineDynamicModule(name: "MyDynamicModule");
// 3. Создаем новый тип
TypeBuilder typeBuilder = moduleBuilder.DefineType(name: "MyDynamicType", attr: TypeAttributes.Public);
// 4. Добавляем поле к типу
FieldBuilder fieldBuilder = typeBuilder.DefineField("MyField", typeof(int), FieldAttributes.Public);
// 5. Создаем метод
MethodBuilder methodBuilder = typeBuilder.DefineMethod(name: "MyMethod", MethodAttributes.Public, typeof(int), Type.EmptyTypes);
// 6. Генерируем код для метода (возвращаем значение поля)
ILGenerator ilGenerator = methodBuilder.GetILGenerator();
ilGenerator.Emit(opcode: OpCodes.Ldarg_0); // Загружаем ссылку на текущий объект
ilGenerator.Emit(opcode: OpCodes.Ldfld, fieldBuilder); // Загружаем значение поля
ilGenerator.Emit(opcode: OpCodes.Ret); // Возвращаем значение
// 7. Завершаем создание типа
Type dynamicType = typeBuilder.CreateType();
```

Как это работает ? (DispatchProxy)

```
private void Complete()
{
    Type[] parameterTypes = new Type[this._fields.Count];
    for (int index = 0; index < parameterTypes.Length; ++index)
        parameterTypes[index] = this._fields[index].FieldType;
    ILGenerator ilGenerator = this._tb.DefineConstructor(MethodAttributes.Public,
gConvention: CallingConventions.HasThis,
parameterTypes).GetILGenerator();
    ConstructorInfo constructor = this._proxyBaseType.GetConstructor(Type.EmptyTypes);
    ilGenerator.Emit(opcode: OpCodes.Ldarg_0);
    ilGenerator.Emit(opcode: OpCodes.Call, constructor);
    for (int index = 0; index < parameterTypes.Length; ++index)
    {
        ilGenerator.Emit(opcode: OpCodes.Ldarg_0);
        ilGenerator.Emit(opcode: OpCodes.Ldarg, arg: index + 1);
        ilGenerator.Emit(opcode: OpCodes.Stfld, (FieldInfo) this._fields[index]);
    }
    ilGenerator.Emit(opcode: OpCodes.Ret);
}
```

Что на выходе?

```
dispatchProxyExample = generatedProxy_1
  > Target = DotNext.Examples.DispatchProxyExample
  > _methodInfos = {System.Reflection.MethodInfo[]} System.Reflection.MethodInfo[1]
```

```
dispatchProxyExample.GetType().GetMethods()
{System.Reflection.MethodInfo[6]}
[0]: {Int32 Action()}
[1]: {DotNext.Examples.IDispatchProxyExample get_Target()}
[2]: {System.Type GetType()}
[3]: {System.String ToString()}
[4]: {Boolean Equals(System.Object)}
[5]: {Int32 GetHashCode()}
```

Где это может еще пригодиться ?

```
private Type BuildDynamicTableTypeHandler(Guid tableViewId, Dictionary<string, TableColumnDefinitionDal> metadata, bool isUseExtendedMode)

public async Task<List<object>> ExecuteSqlAsync(string sql, DynamicParameters dbParams, Type type)
{
    var connection = await GetConnectionAsync();
    var query :IEnumerable<object> = await connection.QueryAsync(type, sql, dbParams);
    return query.AsList();
}
```

Билдим Type для работы dapper, например чтобы правильно интерпретировать типы колонок

Плюс класс

```
proxy.GetType().GetField("__target", BindingFlags.Instance | BindingFlags.NonPublic)
```

```
> $result = {System.Reflection.RtFieldInfo} IdentityDalLib.Repositories.Common.SimplePasswordRepository __target
```

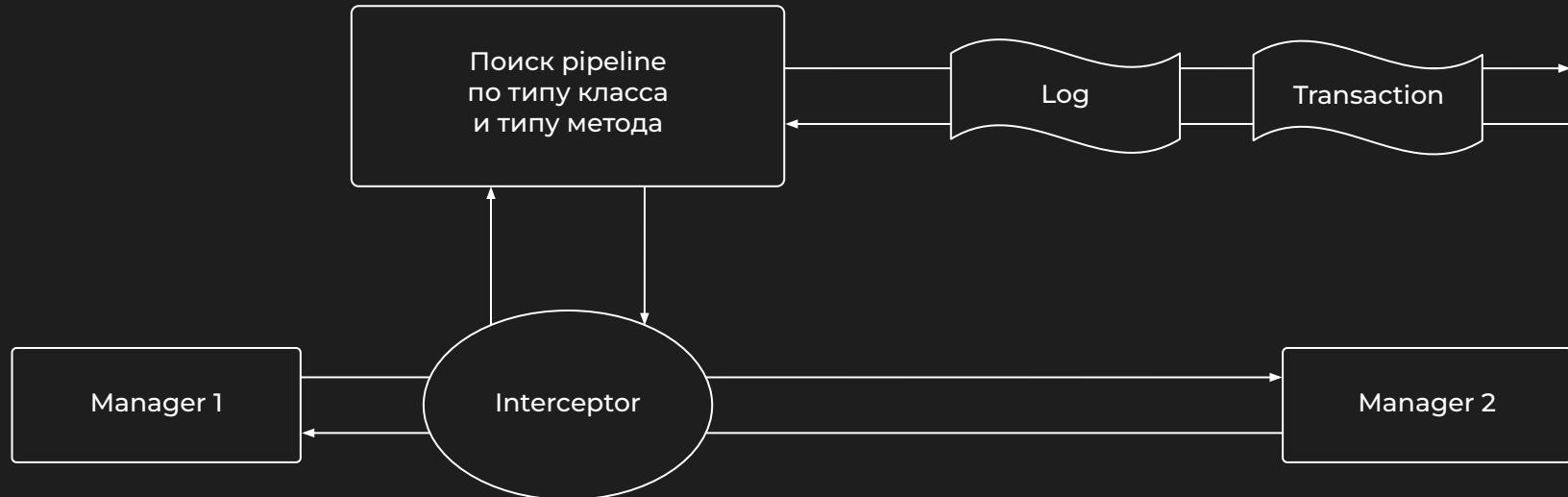
- 1 Через рефлексию/expression/Emit (количество компиляций будет конечным) можно заменять значения
- 2 Необходим ObjectPool. Пока класс, захваченный прокси, не произведет Dispose, мы можем создать гонку
- 3 Нет смысла использовать это для Singleton классов

Разные типы оберток

```
if (ensureRes == EnsureInterfaceInterceptionAppliesType.ByInterface)
{
    Очень выжный код
    var proxy :object = _proxyGenerator.CreateInterfaceProxyWithTarget(
        theInterface,
        interfaces,
        context.Instance,
        params interceptors: new Interceptor(existAttribute, serviceProvider));
    context.Instance = proxy;
}

^ if (ensureRes == EnsureInterfaceInterceptionAppliesType.ByClass)
{
    var proxy :object = _proxyGenerator.CreateClassProxyWithTarget(
        classToProxy: context.Instance.GetType(),
        context.Instance,
        params interceptors: new Interceptor(existAttribute, serviceProvider));
    context.Instance = proxy;
}
```

Pipeline + interceptor chain



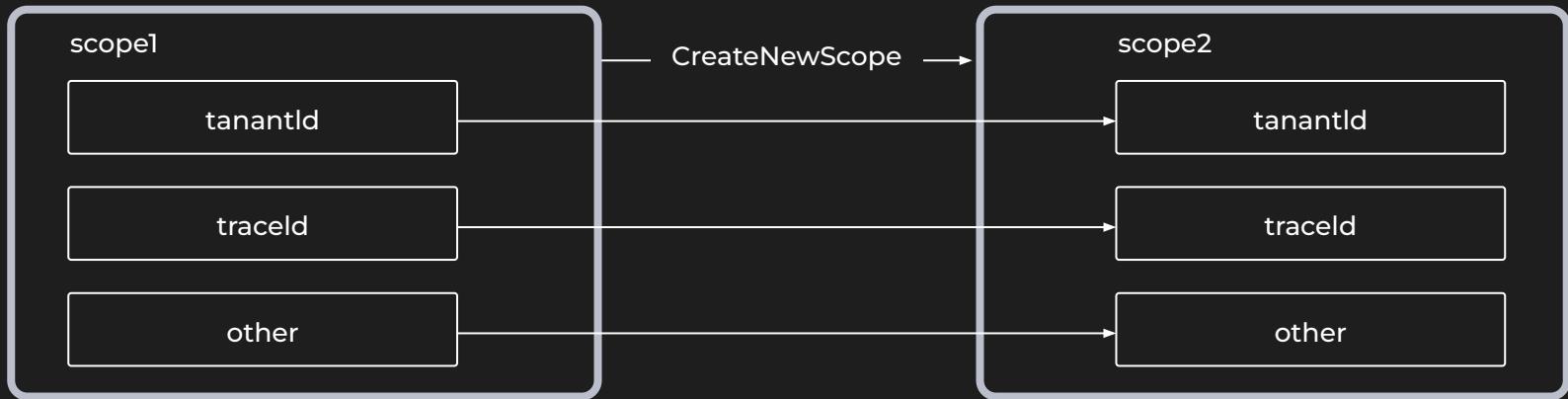


DI чтобы применить
DynamicProxy

Что нам в совокупе нужно от DI?

(начало 2023 года)

- 1 Доступ к CreateScope для переноса трассировочных значений



- 2 Доступ к классу на выходе, чтобы обернуть его в проксю
- 3 Injection Key из коробки (делали его сами еще до .NET 8)

Как мы хотели ?

Наивные

```
services.AddScoped<IServiceProvider, CustomServiceProvider>();
```



```
public class CustomServiceProvider : IServiceProvider, ISupportRequiredService
{
    private readonly IServiceProvider _serviceProvider;

    & degorov *
    public CustomServiceProvider(IServiceProvider serviceProvider)
    {
        _serviceProvider = serviceProvider;
    }

    ☐ 2 usages & degorov
    internal IServiceProvider ServiceProvider => _serviceProvider;

    /// <inheritdoc />
    & degorov *
    public object GetService(Type serviceType) ~
        => GetServiceInner(_serviceProvider.GetService(serviceType));

    /// <inheritdoc />
    & degorov *
    public object GetRequiredService(Type serviceType) ~
        => GetServiceInner(((ISupportRequiredService) serviceProvider).GetRequir

    /// <summary>
    /// Внутренняя логика создания сервиса
    /// </summary>
    & 2 usages & degorov *
    private object GetServiceInner(object service)
    {
        if (service is IServiceScopeFactory serviceScopeFactory)
        {
            var handlerServiceScopeFactories : IEnumerable<IHandlerServiceScopeFactory> =
                _serviceProvider.GetServices<IHandlerServiceScopeFactory>();
            return new CustomServiceScopeFactory(
                serviceScopeFactory, ~
                handlerServiceScopeFactories);
        }

        return service;
    }
}
```

Базовый DI внутри себя через IL Emit создаёт всё дерево

Получили

Вообще в базовом DI
нет логики декорирования
или интерсептирования!!!

Нет такого места, где можно
перехватить получение
внутренних зависимостей

```
protected override object? VisitConstructor(  
    ConstructorCallSite constructorCallSite,  
    ILEmitResolverBuilderContext argument)  
{  
    foreach (ServiceCallSite parameterCallSite in constructorCallSite.ParameterCallSites)  
    {  
        this.VisitCallSite(parameterCallSite, argument);  
        if (parameterCallSite.ServiceType.IsValueType)  
            argument.Generator.Emit(opcode: OpCodes.Unbox_Any, parameterCallSite.ServiceType);  
    }  
    argument.Generator.Emit(opcode: OpCodes.Newobj, constructorCallSite.ConstructorInfo);  
    if (constructorCallSite.ImplementationType.IsValueType)  
        argument.Generator.Emit(opcode: OpCodes.Box, constructorCallSite.ImplementationType);  
    return (object) null;  
}
```

А ЧТО С ДЕКОРИРОВАНИЕМ САМОГО IServiceProvider?

- ❖ Нельзя декорировать IServiceProvider (можно частично, но легко ломается)

```
internal ServiceProvider(
    ICollection<ServiceDescriptor> serviceDescriptors,
    ServiceProviderOptions options)
{
    this.Root = new ServiceProviderEngineScope(provider: this, isRootScope: true);
    this._engine = this.GetEngine();
    this._createServiceAccessor = new Func<ServiceIdentifier, ServiceProvider.ServiceAccessor>(this.Crea
    this._serviceAccessors = new ConcurrentDictionary<ServiceIdentifier, ServiceProvider.ServiceAccessor>
    this.CallSiteFactory = new CallSiteFactory(serviceDescriptors);
    this.CallSiteFactory.Add(ServiceIdentifier.FromServiceType(typeof(IServiceProvider)), (ServiceCalls
    this.CallSiteFactory.Add(ServiceIdentifier.FromServiceType(typeof(IServiceScopeFactory)), (ServiceCalls
    this.CallSiteFactory.Add(ServiceIdentifier.FromServiceType(typeof(IServiceProviderIsService))), (Ser
    this.CallSiteFactory.Add(ServiceIdentifier.FromServiceType(typeof(IServiceProviderIsKeyedService))),
```

А ЧТО С ДЕКОРИРОВАНИЕМ САМОГО IServiceProvider?

- ❖ Можно пойти следующим образом

```
var builder = WebApplication.CreateBuilder(args);
builder.Host.UseServiceProviderFactory(new CustomServiceProviderFactory());
```

IServiceScopeFactory, IServiceProvider, IServiceScope, ISupportRequiredService

Переопределение IServiceProvider



Не получат нашу реализацию:
Controller, Endpoint, либы — в общем, все,
у кого инжекция не от корня идёт



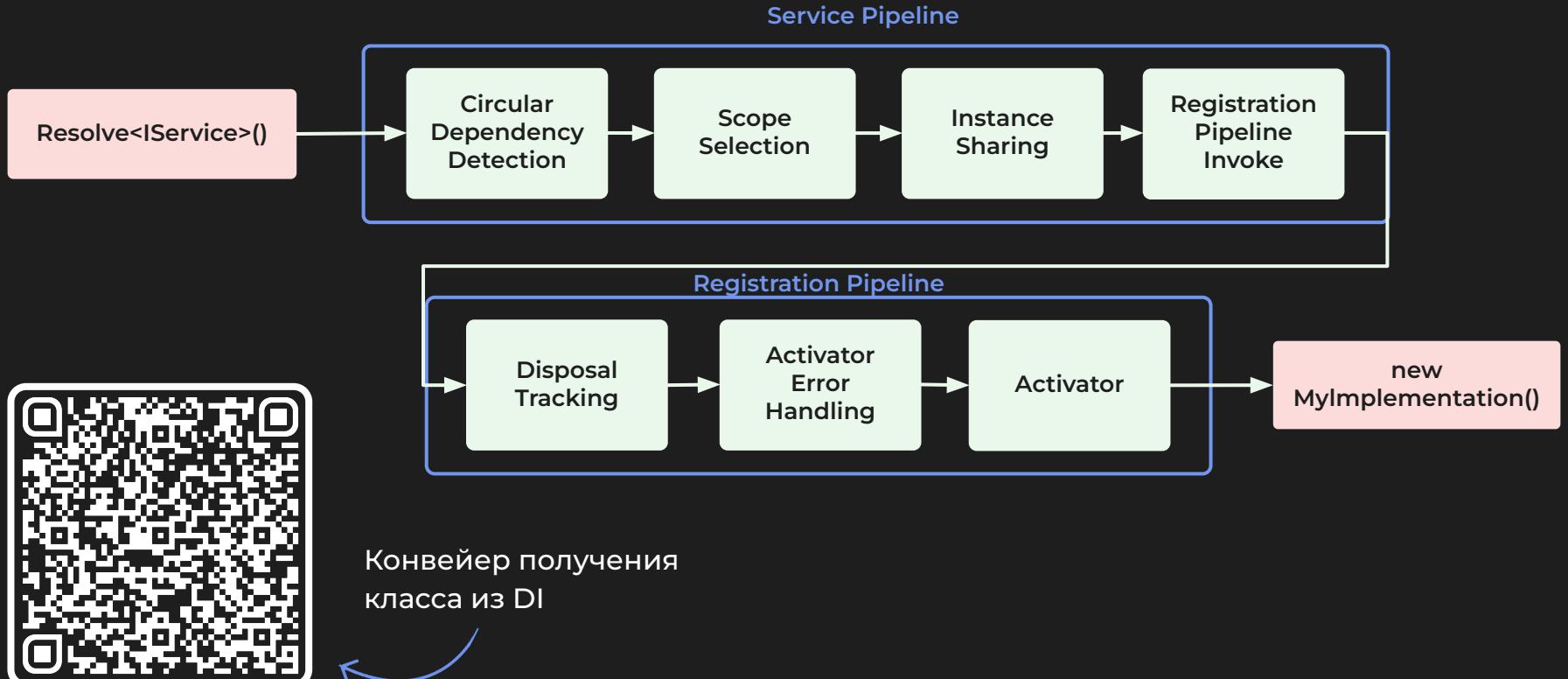
Контролируемую область кода можно
обезопасить через анализатор!!!
Запрет инжекции IServiceProvider

```
[Route( template: "setting-password")]
public class PasswordSettingInternalController : ControllerBase
{
    private readonly IServiceProvider _serviceProvider; _servic

    public PasswordSettingInternalController(IServiceProvider serviceProvider)
    {
        _serviceProvider = serviceProvider;
    }

    [HttpGet]
    public async Task<ActionResult> GetExamplePasswordAsync()
    {
        return Ok();
    }
}
```

Мы пошли в сторону autofac



Pure Di

```
partial class Composition: IInterceptor
{
    private static readonly ProxyGenerator ProxyGenerator
        = new();

    # 1 usage
    private partial T OnDependencyInjection<T>(
        in T value,
        object? tag,
        Lifetime lifetime)
    {
        return (T)ProxyGenerator // ProxyGenerator
            .CreateInterfaceProxyWithTargetInterface(
                interfaceToProxy: typeof(T),
                value,
                params interceptors: this); // object
    }

    public void Intercept(IInvocation invocation)
    {
        invocation.Proceed();
    }
}
```

Доклад того
года об этом



```
DI.Setup(nameof(Composition)).Bind().To<Service>().Root<IService>( name: "Root");
```

Далее мы уже умели
анализировать код

Проблема регистрации и не virtual методов

1

Пусть у нас есть
такой класс
и интерфейс

```
☒ 1 usage  ☐ 1 implementation
public interface ISomeManager { Task ActionAsync(); }

☒ 1 usage
public class SomeManager : ISomeManager
{
    [Cache]
    public Task ActionAsync() { throw new NotImplementedException(); }
}
```

2

И такая
регистрация

```
public static void TryAddDi(this IServiceCollection serviceCollection)
{
    serviceCollection.TryAddTransient<SomeManager>();
}
```

Пример генерации правил с DiagnosticAnalyzer

```
public override void Initialize(AnalysisContext context)
{
    context.ConfigureGeneratedCodeAnalysis(GeneratedCodeAnalysisFlags.None);
    context.EnableConcurrentExecution();
    context.RegisterSymbolAction>AnalyzeMethod, params symbolKinds: SymbolKind.Method);
}

☒ 1 usage  ↗ new *

private static void AnalyzeMethod(SymbolAnalysisContext context)
{
    var methodSymbol = (IMethodSymbol)context.Symbol;

    // Проверяем, есть ли у метода атрибут InterseptAttribute
    var hasInterseptAttribute = methodSymbol.GetAttributes().Any(attr:AttributeData => attr.AttributeClass.Name == "InterseptAttribute");

    if (hasInterseptAttribute)
    {
        // Проверяем, является ли метод virtual или реализует интерфейс
        var isVirtual = methodSymbol.IsVirtual;
        var implementsInterface:bool = methodSymbol.ContainingType.AllInterfaces // ImmutableList<INamedTypeSymbol>
            .Any(i:INamedTypeSymbol => i.GetMembers().Any(m:ISymbol => methodSymbol.Equals(other: methodSymbol.ContainingType.FindImplementationForInterfaceMember(m))));

        if (!isVirtual && !implementsInterface)
        {
            var diagnostic = Diagnostic.Create(descriptor:_rule, methodSymbol.Locations[0], methodSymbol.Name);
            context.ReportDiagnostic(diagnostic);
        }
    }
}
```

Пример генерации правил с DiagnosticAnalyzer

```
context.EnableSourceFileExecution();
context.RegisterOperationAction>AnalyzeInvocation
```

params operationKinds: OperationKind.Invocation);
}

[1 usage 2 new *
private static void AnalyzeInvocation(OperationAnalysisContext context)
{
 var invocation = (IInvocationOperation)context.Operation;

 // Проверяем, является ли метод вызовом TryAddTransient с дженериками
 if (invocation.TargetMethod.Name == "TryAddTransient" && invocation.TargetMethod.IsGenericMethod)
 {
 // Получаем типы TService и TImplementation, если они существуют
 var genericArguments:ImmutableArray<ITypeSymbol> = invocation.TargetMethod.TypeArguments;

 if (genericArguments.Length == 1 || genericArguments.Length == 2)
 {
 var implementationType = genericArguments[genericArguments.Length - 1]; // Берем последний дженерик-аргумент как реализацию

 // Проверяем, реализует ли класс интерфейсы
 if (implementationType.AllInterfaces.Any())
 {
 // Проверяем наличие атрибута InterseptAttribute на методах класса
 var hasInterseptAttribute = implementationType.GetMembers() // ImmutableArray<ISymbol>
 .OfType<IMethodSymbol>() // IEnumerable<IMethodSymbol>
 .Any(method => method.GetAttributes().Any(attr:AttributeData => attr.AttributeClass.Name == "InterseptAttribute"));

 if (hasInterseptAttribute && genericArguments.Length == 1) // Если регистрируется только реализация без интерфейса
 {
 var diagnostic = Diagnostic.Create(descriptor: Rule, invocation.Syntax.GetLocation(), implementationType.Name);
 context.ReportDiagnostic(diagnostic);
 }
 }
 }
 }
}

Проблема: Двойная прокся

```
public static void TryAddDi(this IServiceCollection serviceCollection)
{
    serviceCollection.TryAddTransient<SomeManager>();
    serviceCollection.TryAddTransient<ISomeManager>(
        implementationFactory: sp : IServiceProvider => sp.GetRequiredService<SomeManager>());
}
```

Аналогично тому, чтобы сделать вот так:

```
var proxyGenerator = new ProxyGenerator();
var castleProxyClass : object? = proxyGenerator.CreateClassProxyWithTarget(typeof(DispatchProxyExample), target: new DispatchProxyExample(), interceptors: new []{ new LogStandardInterceptor()});
var castleProxyInterface : object? = proxyGenerator.CreateInterfaceProxyWithTarget(typeof(IDispatchProxyExample), castleProxyClass, interceptors: new []{ new LogStandardInterceptor()});
```

Проблема: Класс из DI извлекается по ключу типа, а не интерфейса

→ перешли на базовый `InjectionKey`

```
public static class UserManagerInjector
{
    public static class UserManagerInjector
    {
        private static readonly Dictionary<UserType, Type> _userManagerDictionary = new();

        [2 usages & degorov *]
        public static void AddUserManager<TRealization>(this IServiceCollection serviceCollection)
            where TRealization : class, IUserManager, new()
        {
            var userManagerType = DynamicHelper.GetDynamicFieldValue<TRealization, UserType>(nameof(IUserManager.UserType));
            _userManagerDictionary.Add(userManagerType, typeof(TRealization));

            serviceCollection.AddTransient<TRealization>();
            serviceCollection.AddTransient<IUserManager>(service => service.GetRequiredService<TRealization>());
            serviceCollection.AddUserManager();
        }

        /// <summary>
        /// Создание кол бэка для поиска сервиса нужного
        /// </summary>
        [1 usage & degorov *]
        private static void AddUserManager(this IServiceCollection services)
        {
            services.TryAddSingleton<GetUserManager>(<implementationFactory: provider => key: UserType =>
            {
                var type :KeyValuePair<UserType,Type> = _userManagerDictionary.FirstOrDefault(value => value.Key == key);
                var service = (IUserManager)provider.GetService(type.Value);
                return service;
            });
        }
    }
}
```

Проблема: Анализ через рефлексию

```
foreach (var externalActionFactory in _externalActionFactoryList)
{
    var actionName :string = externalActionFactory.Name;
    var action = externalActionFactory.GetExternalAction();
    var actionType = action.GetType();

    var actionContextType = ActionContextHelper.GetActionContextTypeOrDefault(actionType);
```

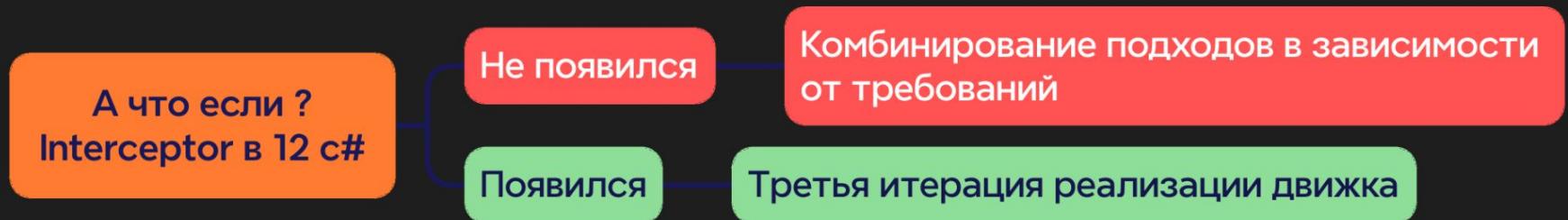
Вместо нужного типа мы получили тип обертки

```
var field = proxyType.GetField(name: "__target", bindingAttr: BindingFlags.Instance | BindingFlags.NonPublic);
var type = field.GetMemberType();
```

Итоге по DynamicProxy и Autofac

- 1 Есть полная поддержка DI
- 2 Async/await — решение аналогично Fody
- 3 Перехват только публичных методов между классами
- 4 Встраивание через прокси над классом, нет безумных багов с IL, но есть баги с Reflection и DI
- 5 Накладные расходы при transient: +1 класс прокси (решается через ObjectPool), +1 класс для Invocation с параметрами и ещё внутренние классы. В общем, лучше по числу не стало(
- 6 Один pipeline, возможность работать с async до старта метода

А что дальше?



Interceptors in C# 12

Interceptors in C# 12

Еще до появления interceptor можно было использовать source generate для реализации АОП

Но тут были минусы



- partial class



Interceptors in C# 12

```
var interceptorExample = new InterceptorExample();
await interceptorExample.Action2Async();
```

```
[InterceptsLocation( filePath: "/Users/dmitrijegorov/Desktop/DotNext/DotNext/Program.cs", line: 66, character: 26)]
```

```
2 usages
```

```
public static async Task Action2Async(this InterceptorExample example)
{
    Console.WriteLine(nameof(Action2Async) + "1");
    await example.Action2Async();
    Console.WriteLine(nameof(Action2Async) + "2");
}
```

- 1 Поддержка async/await
- 2 Работают через указание path line char
- 3 Если вы используете Rider есть интеграция

Никаких лишних классов не создаётся, всё максимально экономично через статику

```
var interceptorExample = new InterceptorExample();
await interceptorExample.ActionAsync();
```

до

```
InterceptorExample interceptorExample = new InterceptorExample();
await GeneratedCode.ActionAsync(interceptorExample);
```

после

- 1 Конечно минус, что тут ситуация с DI аналогично Fody
- 2 Перехват только публичных методов, зато как между, так и внутри классов (в теории можно вообще любые)

Простой пример (source generate)

```
context.RegisterPostInitializationOutput(ctx :IncrementalGeneratorPostInitializationContext =>
{
    var sourceText = $"""
        using System.Runtime.CompilerServices;

        namespace SourceGeneratorInCSharp;

        public static class GeneratedCodeTwo
        {
            [InterceptsLocation("/Users/dmitrijegorov/Desktop/CodeMaz
            public static string InterceptorMethod(this Example example)
            {
                return $"{text}, YES";
            }
        }
    """;

    ctx.AddSource( hintName: "Inter.g", SourceText.From(sourceText, Encoding.UTF8));
});
```

Чуть сложнее

Синтаксический
анализ

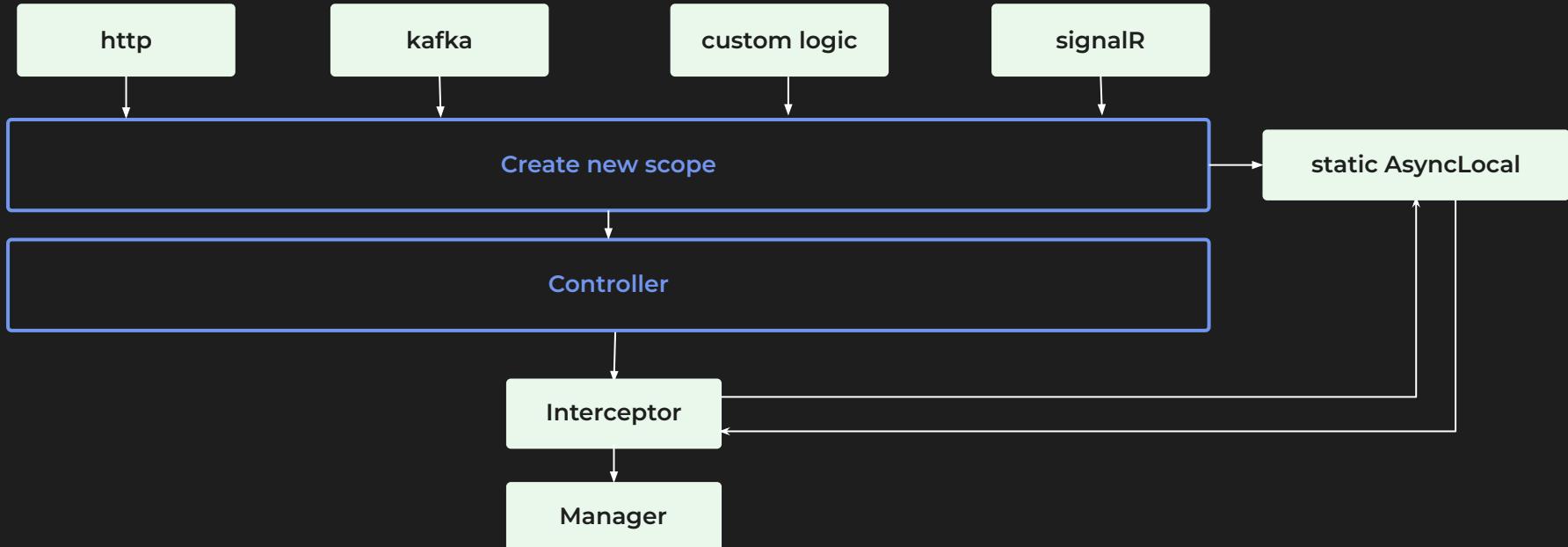
Генерация
интерсепторов

```
var providerInvocationExp = context.SyntaxProvider.CreateSyntaxProvider(  
    predicate: (n : SyntaxNode, _) => n is MemberAccessExpressionSyntax,  
    transform: (n : GeneratorSyntaxContext, cp : CancellationToken) =>  
{  
    var node = (MemberAccessExpressionSyntax)n.Node;  
    var semanticModel = n.SemanticModel;  
    var location = node.Name.GetLocation();  
    var lineSpan = location.GetLineSpan();  
    var startLine : int = lineSpan.StartLinePosition.Line + 1;  
    var startColumn : int = lineSpan.StartLinePosition.Character + 1;  
    var symbol = semanticModel.GetSymbolInfo(node, cp).Symbol;  
    var hasAttribute : bool? = symbol?.GetAttributes() // ImmutableArray<AttributeData>  
        .Any(attribute => attribute.AttributeClass?.Name == "InterceptAttribute")  
    if (hasAttribute != true)  
    {  
        return null;  
    }  
    var typeInfo = semanticModel.GetTypeInfo(node.Expression);  
    var className = "";  
    var namespaceName = "";  
    if (typeInfo.Type is INamedTypeSymbol namedTypeSymbol)  
    {  
        className = namedTypeSymbol.Name;  
        namespaceName = namedTypeSymbol.ContainingNamespace.ToString()  
    }  
    var filePath : string = location.SourceTree?.FilePath;  
    return new MethodInfoToIntercept(className, namespaceName, methodName: node.Identifier.Text,  
        filePath: filePath);  
});
```

**От создателей таких хитов,
как «ваше приложение больше
не скомпилируется с namespace
Newtonsoft»**

— Мы запрещаем использовать —
/ CreateScope вне CoreLib \

И возвращаемся к этой концепции



Итоговый вид перехватчика

```
// ReSharper disable once InconsistentNaming
private static readonly Type _interceptorExampleType = typeof(InterseptorExample);
// ReSharper disable once InconsistentNaming
private static readonly MethodInfo _interceptorExampleTypeAction = typeof(InterseptorExample).GetMethod( name: "ActionAsync" );
[<InterceptsLocation( filePath: "/Users/dmitrijegorov/Desktop/DotNext/DotNext/Program.cs", line: 66, character: 26 )>]
public static async Task ActionAsync(this InterseptorExample example)
{
    if (InterceptPipelineCreator.Store.TryGetValue(_interceptorExampleType, out var methodList : Dictionary<MethodInfo, Intercept>))
    {
        if (methodList.TryGetValue(_interceptorExampleTypeAction, out var intercept))
        {
            var executeContext = ExecuteInterceptorContextPool.Pool.Get();
            try
            {
                executeContext.ServiceProvider = ServiceProviderStore.ServiceProvider.Value;
                await intercept(executeContext);
            }
            finally
            {
                ExecuteInterceptorContextPool.Pool.Return(executeContext);
            }
        }

        return;
    }

    await example.ActionAsync();
}
```

ExecuteInterceptorContext

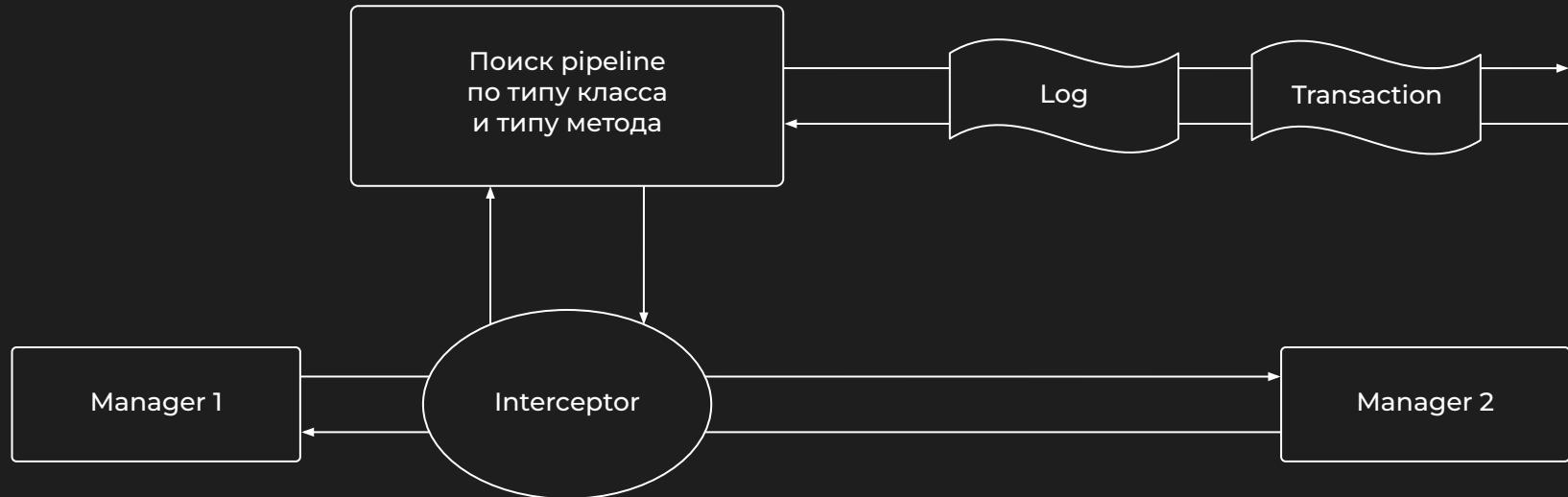
```
public class ExecuteInterceptorContext
{
    ↗ 2 usages
    public required IServiceProvider ServiceProvider { get; set; }

    internal object ExecuteClass { get; set; }
}
```

```
public class ExecuteInterceptorContext<T, U> : ExecuteInterceptorContext
{
    public T Param1;

    public U Param2;
}
```

Pipeline + interceptor chain



Итоги по Interceptor

- 1 Накладные расходы — сам по себе движок не создает их
- 2 Async/await — поддержка из коробки без магии
- 3 Перехват любых методов вызываемых из любых методов (expression)



Без трудностей не обошлось



Частичное использование интерсепторов

1 Было

```
public async Task DeleteActionAsync(Guid actionId)
{
    await using var lockHandler :IAsyncDisposable = await _keyedSemaphoresCollection.LockAsync(actionId, timeout: null);
    var transaction = await _dbConnectionFactory.StartTransactionAsync();
```

2 Сломали

```
[Transaction]
└ 0+1 usages
public async Task DeleteActionAsync(Guid actionId)
{
    await using var lockHandler :IAsyncDisposable = await _keyedSemaphoresCollection.LockAsync(actionId, timeout: null);
```

3 Итог

```
[Transaction]
[Lock(nameof(actionId))]
└ 0+1 usages
public async Task DeleteActionAsync(Guid actionId)
{
```

Лейзи поведение для атрибута

Транзакция и соединение должны создаваться в момент первого обращения в БД, а не заранее, иначе нельзя делать так:

```
[ Transaction ]
public async Task<UpdateOnAuthIdentityResponse> UpdateOnAuthIdentityAsync(UpdateOnAuthIdentityRequest request)
{
    var identityUser = await _identityConnectionService.GetUserByIdAsync(new GetUserByIdRabbitRequest(request.Id));
    var isProvider = await _providerProfileRepository.ExistByAsync(nameof(ProviderProfileDal.IdentityUserId), request.Id);
```

Не всегда получается гладко использовать атрибуты

```
/// <inheritDoc/>
public async Task SaveFileAsync(byte[] fileContent, string fileNameWithExtension, string directoryPath)
{
    var endLog = LogHelper.StartLog(fileNameWithExtension, directoryPath, _fileStoreDirectoryAccess.DirectoryStore);

    var contentPath = Path.Combine(_fileStoreDirectoryAccess.DirectoryStore, directoryPath);
    await CreateFile(fileContent, fileNameWithExtension, contentPath);

    endLog();
}
```

- ❖ Как правило, это говорит о неправильной декомпозиции методов

Пример невалидного кода

```
[  
{  
    public async Task DeleteAsync(ObjectId id)  
    {  
        var transaction = await _conditionRepository.BeginTransactionAsync();  
        var entity :ConditionSchemaDal = await _conditionRepository.GetAsync(id);  
  
        await _conditionRepository.DeleteAsync(entity.Id, transaction);  
  
        await transaction.CommitAsync();  
  
        var conditionSchemaDeletedEvent = new ConditionSchemaDeletedEvent  
        {  
            ConditionSchemaId = id  
        };  
  
        _brokerSenderService.SendEvent(conditionSchemaDeletedEvent);  
    }  
}
```

[Transaction]

```
public async Task DeleteAsync(ObjectId id)  
{  
    var entity :ConditionSchemaDal = await _conditionRepository.GetAsync(id);  
  
    await _conditionRepository.DeleteAsync(entity.Id);  
  
    var conditionSchemaDeletedEvent = new ConditionSchemaDeletedEvent  
    {  
        ConditionSchemaId = id  
    };  
  
    _brokerSenderService.SendEvent(conditionSchemaDeletedEvent);  
}
```

Подводим итоги

Сравнительная таблицы функциональных возможностей

Тип	Плюс классов	Внешнее состояние	Async /await	Какие вызовы можно перехватить?	Где идет встраивание?
Fody + Default DI	3 + x	AsyncLocal	через delegate	любые	метод, который вызываем
DispatchProxy + Autofac	2 + n + x	DI	через delegate	public interface & virtual class	прокся
Interceptor source gen	1 (objectPool) + x	AsyncLocal	+	любые	метод, который вызывает

Итоговая схема



- Движок: Interceptor c#12
- Async/Await: проблем нет
- Pipeline: Как middleware
- Накладные расходы: ExecuteInterceptorContext для pipeline (ObjectPool)
- Di в interceptor: AsyncLocal
- Di движок: Откатились к базовому (переходим на Pure Di)
- Запретили: CreateScope
- Перехват любых методов из любых методов

★ Есть проблема, мы не можем инверсировать внутри сторонних либ, которые расширяем через инверсию (но такое встречается у нас редко)