POSITIVE TECHNOLOGIES

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

AOP в .NET

Как можно использовать аспекты в .NET

Игорь Яковлев
iyakovlev@ptsecurity.com

Обещание

Рассмотрим Аспектно-Ориентированную Парадигму

Узнаем как реализована эта парадигма в .NET

Рассмотрим основные фреймворки и библиотеки АОП для .NET на примерах

Почему АОП?

Мы научимся решать задачи декомпозиции кода

Узнаем как легко и просто добавлять функциональность в модули не меняя их

Узнаем как можно сократить и упростить наш код

Определения

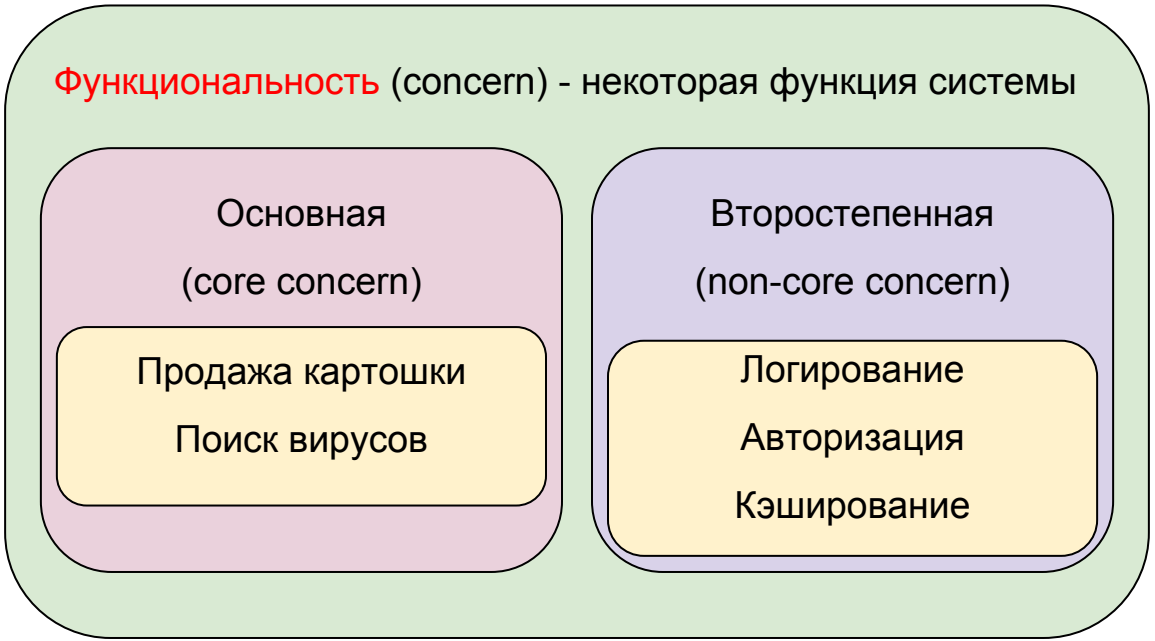
Те, что далеки от практики :)

АОП - аспектно ориентированное программирование

Парадигма программирования предназначенная для декомпозиции кода посредством использования конструкций АОП - “Аспектов”

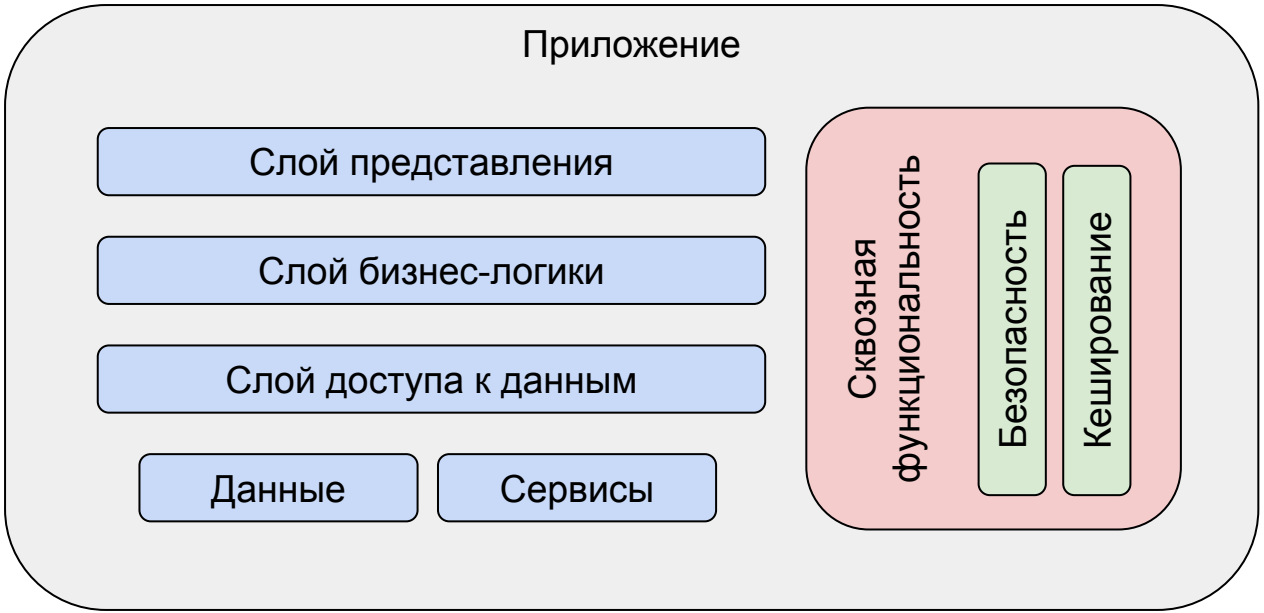
Следование парадигме может улучшить модульность программы, решая т.н. проблему “сквозной функциональности”

Функциональность (concern)

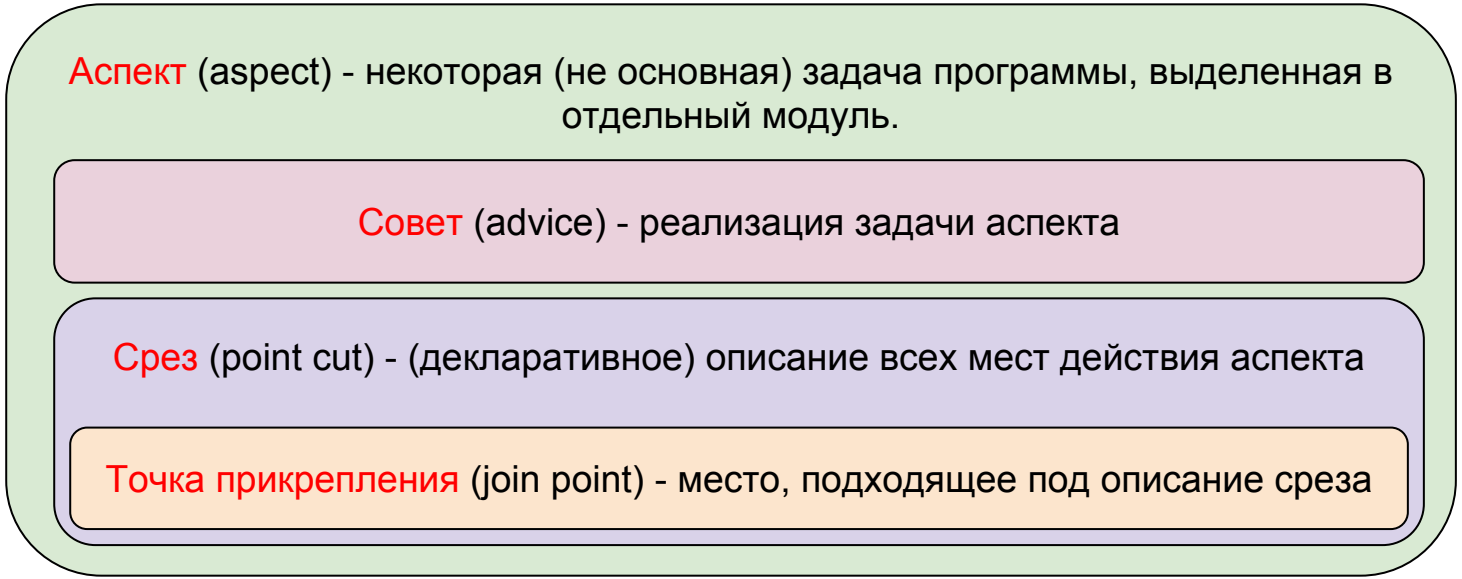


Сквозная функциональность (cross-cutting concern)

Сквозная функциональность - функциональность, реализация которой рассредоточена на несколько модулей или слоев приложения



Аспект


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

Аспект = Совет + Срез [Точки прикрепления]

Аспект

```
class Aspect
{
    public string PointCut => "*Class.*Mehtod[Enter]";

    public void Advice()
    {
        Console.WriteLine("Entering");
    }
}
```

Исходный код

```
public class MyClass
{
    public void MyMethod()
    {
        //...
    }
}
```

Аспект = Совет + Срез [Точки прикрепления]

Аспект

```
class Aspect
{
    public string PointCut => "*Class.*Mehtod[Enter]";

    public void Advice()
    {
        Console.WriteLine("Entering");
    }
}
```

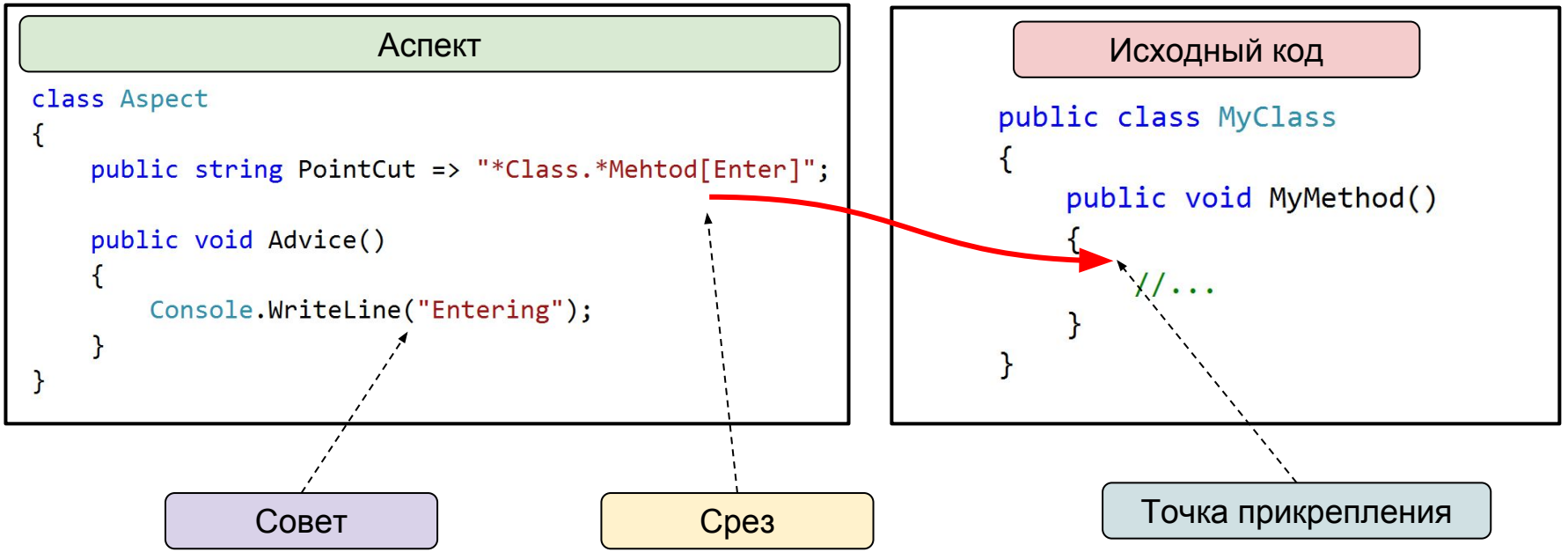
Совет

Срез

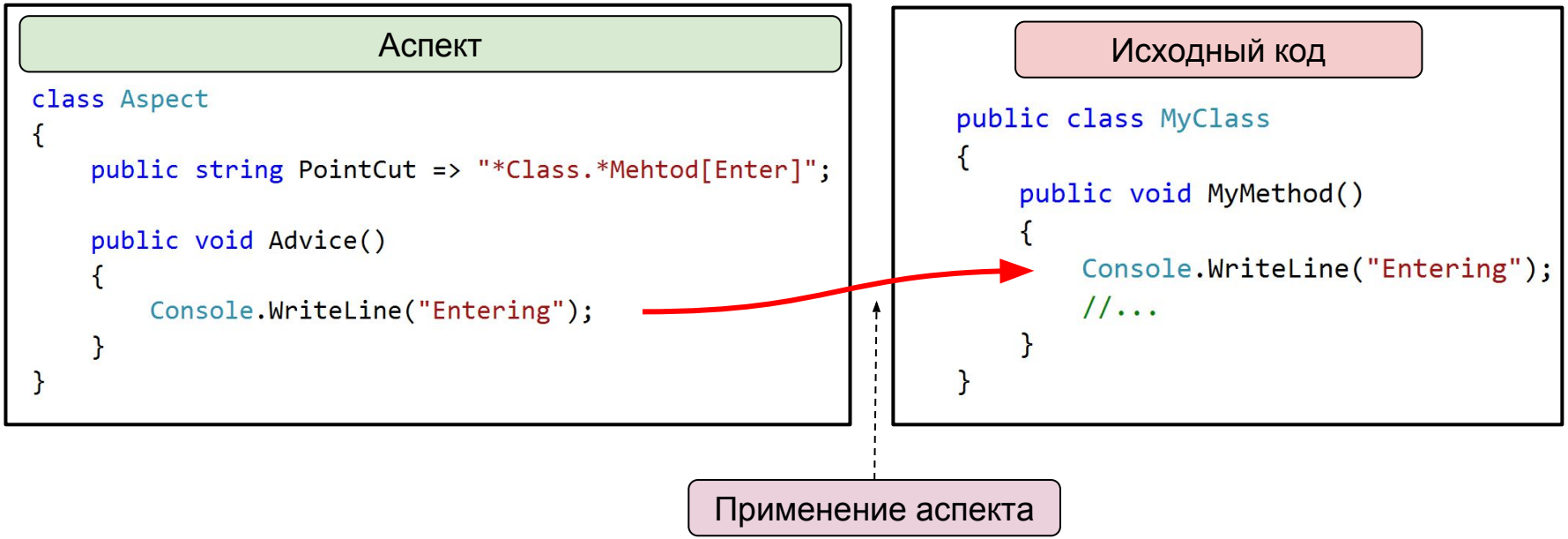
Исходный код

```
public class MyClass
{
    public void MyMethod()
    {
        //...
    }
}
```

Аспект = Совет + Срез [Точки прикрепления]



Аспект = Совет + Срез [Точки прикрепления]



Сквозная функциональность - пример

```
class PotatoModel {
    public int Price { get; set; }
}
```

Сквозная функциональность - пример

```
class PotatoModel {
    private int _price;

    public int Price {
        get { return _price; }
        set {
            _price = value;
            Logger.Instance.Log($"Price is changed to {_price}");
        }
    }
}
```

Сквозная функциональность - пример

```
class PotatoModel : INotifyPropertyChanged {
    public event PropertyChangedEventHandler PropertyChanged;
    private int _price;

    public int Price {
        get { return _price; }
        set {
            _price = value;
            PropertyChanged?.Invoke(
                    this, new PropertyChangedEventArgs(nameof(Price)));
            Logger.Instance.Log($"Price is changed to {_price}");
        }
    }
}
```

Сквозная функциональность - пример

```
class PotatoModel : INotifyPropertyChanged {
    public event PropertyChangedEventHandler PropertyChanged;
    private int _price;

    public int Price {
        get { return _price; }
        set {
            _price = value;
            PropertyChanged?.Invoke(
                    this, new PropertyChangedEventArgs(nameof(Price)));
            Logger.Instance.Log($"Price is changed to {_price}");
        }
    }
}
```

Основная задача (core-concern)

Вынесение функциональности в аспекты

```
class PotatoModel {
    public int Price { get; set; }
}
```

Аспект “логирование сеттеров”

Совет - запись в лог

Срез - вызов сеттеров в классах Models.*Model

Аспект “поддержки view-модели”

Совет - Реализовать интерфейс INotifyPropertyChanged

Срез - Классы Models.*Model

Внедрение (introduction)

Внедрение (introduction) - изменение структуры элемента программы т.е. добавление к нему поддержки интерфейсов, новых методов, свойств, полей, параметров и тп.

Совет реализующий внедрение часто называют примесью (MixIn)

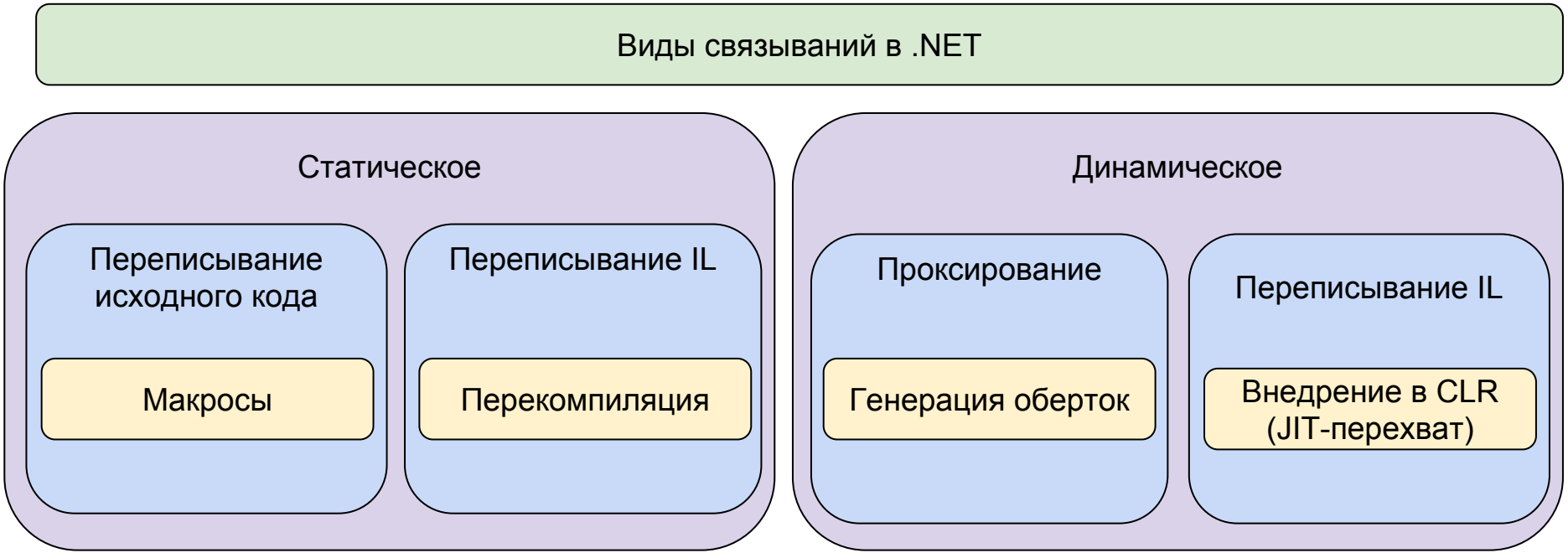
Связывание (weaving)

Связывание (weaving) - способ заставить выполнить совет в точках прикрепления

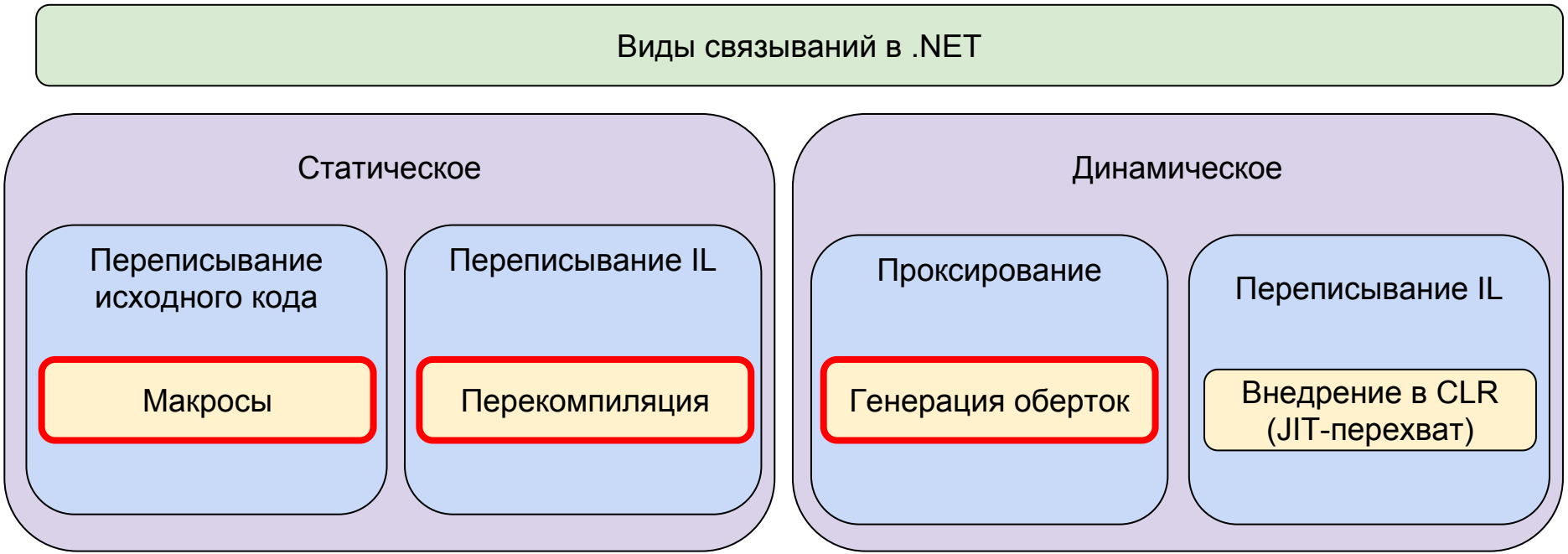
Статическое
(перед запуском программы)

Динамическое
(во время выполнения программы)

Связывание (weaving)



Связывание (weaving)



Закрепим

Функциональность (concern) - некоторая функция системы

Сквозная функциональность (cross-cutting concern) - фун-ть распределенная по модулям

Аспект (aspect) - совет + срез

Совет (advice) - ЧТО делает аспект

Срез (point cut) - ГДЕ действует аспект

Точка прикрепления (join point) - ГДЕ КОНКРЕТНО действует аспект

Внедрение (introduction) - ИЗМЕНЕНИЕ В СТРУКТУРЕ программы (типе, функции, …)

Связывание (weaving) - СПОСОБ применения совета в точке прикрепления

Закрепим

Функциональность (concern) - некоторая функция системы

Сквозная функциональность (cross-cutting concern) - фун-ть распределенная по модулям

Аспект (aspect) - совет + срез

Совет (advice) - ЧТО делает аспект

Срез (point cut) - ГДЕ действует аспект

Точка прикрепления (join point) - ГДЕ КОНКРЕТНО действует аспект

Внедрение (introduction) - ИЗМЕНЕНИЕ В СТРУКТУРЕ программы (типе, функции, …)

Связывание (weaving) - СПОСОБ применения совета в точке прикрепления

Закрепим

Функциональность (concern) - некоторая функция системы

Сквозная функциональность (cross-cutting concern) - фун-ть распределенная по модулям

Аспект (aspect) - совет + срез

Совет (advice) - ЧТО делает аспект

Срез (point cut) - ГДЕ действует аспект

Точка прикрепления (join point) - ГДЕ КОНКРЕТНО действует аспект

Внедрение (introduction) - ИЗМЕНЕНИЕ В СТРУКТУРЕ программы (типе, функции, …)

Связывание (weaving) - СПОСОБ применения совета в точке прикрепления

Закрепим

Функциональность (concern) - некоторая функция системы

Сквозная функциональность (cross-cutting concern) - фун-ть распределенная по модулям

Аспект (aspect) - совет + срез

Совет (advice) - ЧТО делает аспект

Срез (point cut) - ГДЕ действует аспект

Точка прикрепления (join point) - ГДЕ КОНКРЕТНО действует аспект

Внедрение (introduction) - ИЗМЕНЕНИЕ В СТРУКТУРЕ программы (типе, функции, …)

Связывание (weaving) - СПОСОБ применения совета в точке прикрепления

Закрепим

Функциональность (concern) - некоторая функция системы

Сквозная функциональность (cross-cutting concern) - фун-ть распределенная по модулям

Аспект (aspect) - совет + срез

Совет (advice) - ЧТО делает аспект

Срез (point cut) - ГДЕ действует аспект

Точка прикрепления (join point) - ГДЕ КОНКРЕТНО действует аспект

Внедрение (introduction) - ИЗМЕНЕНИЕ В СТРУКТУРЕ программы (типе, функции, …)

Связывание (weaving) - СПОСОБ применения совета в точке прикрепления

Закрепим

Функциональность (concern) - некоторая функция системы

Сквозная функциональность (cross-cutting concern) - фун-ть распределенная по модулям

Аспект (aspect) - совет + срез

Совет (advice) - ЧТО делает аспект

Срез (point cut) - ГДЕ действует аспект

Точка прикрепления (join point) - ГДЕ КОНКРЕТНО действует аспект

Внедрение (introduction) - ИЗМЕНЕНИЕ В СТРУКТУРЕ программы (типе, функции, …)

Связывание (weaving) - СПОСОБ применения совета в точке прикрепления

Закрепим

Функциональность (concern) - некоторая функция системы

Сквозная функциональность (cross-cutting concern) - фун-ть распределенная по модулям

Аспект (aspect) - совет + срез

Совет (advice) - ЧТО делает аспект

Срез (point cut) - ГДЕ действует аспект

Точка прикрепления (join point) - ГДЕ КОНКРЕТНО действует аспект

Внедрение (introduction) - ИЗМЕНЕНИЕ В СТРУКТУРЕ программы (типе, функции, …)

Связывание (weaving) - СПОСОБ применения совета в точке прикрепления

Закрепим

Функциональность (concern) - некоторая функция системы

Сквозная функциональность (cross-cutting concern) - фун-ть распределенная по модулям

Аспект (aspect) - совет + срез

Совет (advice) - ЧТО делает аспект

Срез (point cut) - ГДЕ действует аспект

Точка прикрепления (join point) - ГДЕ КОНКРЕТНО действует аспект

Внедрение (introduction) - ИЗМЕНЕНИЕ В СТРУКТУРЕ программы (типе, функции, …)

Связывание (weaving) - СПОСОБ применения совета в точке прикрепления

Закрепим

Функциональность (concern) - некоторая функция системы

Сквозная функциональность (cross-cutting concern) - фун-ть распределенная по модулям

Аспект (aspect) - совет + срез

Совет (advice) - ЧТО делает аспект

Срез (point cut) - ГДЕ действует аспект

Точка прикрепления (join point) - ГДЕ КОНКРЕТНО действует аспект

Внедрение (introduction) - ИЗМЕНЕНИЕ В СТРУКТУРЕ программы (типе, функции, …)

Связывание (weaving) - СПОСОБ применения совета в точке прикрепления

Примеры

АОП в примерах и задачах

Пример - профайлинг

```
public void Function()
{
    var begin = DateTime.Now.Ticks;
    //...
    var end = DateTime.Now.Ticks;
    Logger.Instance.Log($"Ticks for {nameof(Function)}: {end - begin}");
}
```

Пример - профайлинг

```
public void Function()
{
    var begin = DateTime.Now.Ticks;
    //...
    var end = DateTime.Now.Ticks;
    Logger.Instance.Log($"Ticks for {nameof(Function)}: {end - begin}");
}
```

Пример - кэширование

```
private readonly Dictionary<object, object> cache =
    new Dictionary<object, object>();

public object Function(object argument)
{
    object result;
    if (cache.TryGetValue(argument, out result))
    {
        return result;
    }

    //...

    cache.Add(argument, result);
    return result;
}
```

Пример - кэширование

```
private readonly Dictionary<object, object> cache =
    new Dictionary<object, object>();

public object Function(object argument)
{
    object result;
    if (cache.TryGetValue(argument, out result))
    {
        return result;
    }

    //...

    cache.Add(argument, result);
    return result;
}
```

Пример - контракты

```
public void Function(object argument1, object argument2)
{
    Debug.Assert(argument1 != null);
    Debug.Assert(argument2 != null);
    //...
}
```

Пример - контракты

```
public void Function(object argument1, object argument2)
{
    Debug.Assert(argument1 != null);
    Debug.Assert(argument2 != null);
    //...
}
```

Пример - исключения

```
public void Function()
{
    try
    {
        //...
    }catch(Exception e) when(!(e is VeryBadException))
    {
        //...
    }
}
```

Пример - исключения

```
public void Function()
{
    try
    {
        //...
    }catch(Exception e) when(!(e is VeryBadException))
    {
        //...
    }
}
```

Пример - дата-классы

```
class DataClass
{
    public string StringProperty { get; set; }
    public int IntProperty { get; set; }

    protected bool Equals(DataClass other)
        => string.Equals(StringProperty, other.StringProperty) && IntProperty == other.IntProperty;

    public override bool Equals(object obj)
    {
        if (ReferenceEquals(null, obj)) return false;
        if (ReferenceEquals(this, obj)) return true;
        if (obj.GetType() != this.GetType()) return false;
        return Equals((DataClass) obj);
    }

    public override int GetHashCode()
    {
        unchecked
        {
            return ((StringProperty != null ? StringProperty.GetHashCode() : 0)*397) ^ IntProperty;
        }
    }
}
```

Пример - дата-классы

```
class DataClass
{
    public string StringProperty { get; set; }
    public int IntProperty { get; set; }

    protected bool Equals(DataClass other)
        => string.Equals(StringProperty, other.StringProperty) && IntProperty == other.IntProperty;

    public override bool Equals(object obj)
    {
        if (ReferenceEquals(null, obj)) return false;
        if (ReferenceEquals(this, obj)) return true;
        if (obj.GetType() != this.GetType()) return false;
        return Equals((DataClass) obj);
    }

    public override int GetHashCode()
    {
        unchecked
        {
            return ((StringProperty != null ? StringProperty.GetHashCode() : 0)*397) ^ IntProperty;
        }
    }
}
```

Задачи

Две невероятно стандартных АОП-задачи

Задача 1 - трассировка (before-after)

```
public interface ICommand
{
    string Execute();
}

public class CommandService : ICommand
{
    public virtual string Execute()
    {
        //...
    }
}
```

Задача 1: трассировка вызова Execute в консоль

Задача 2 - добавить INPC (INPC-mixin)

```
class PotatoModel
{
    public virtual int Price { get; set; }
}
```

Задача 2: Добавить к поддержку интерфейса INotifyPropertyChanged

Динамическое связывание

Как завязывать шнурки на бегу

Прокси вручную (before-after)

```
class CommandWrapper : ICommand {
    private readonly ICommand target;
    public CommandWrapper(ICommand target) {
        this.target = target;
    }
    public string Execute() {
        Console.WriteLine("Before");
        string returnValue = target.Execute();
        Console.WriteLine("After");
        return returnValue;
    }
}
```

Решение не совершенно расширяемо:
Каждому интерфейсу свой враппер! :/

Прокси вручную (before-after)

```
class CommandWrapper : ICommand {
    private readonly ICommand target;
    public CommandWrapper(ICommand target) {
        this.target = target;
    }
    public string Execute() {
        Console.WriteLine("Before");
        string returnValue = target.Execute();
        Console.WriteLine("After");
        return returnValue;
    }
}
```

Было бы здорово!

Создавать произвольные обертки

Реализовывать их интерфейсы

ПРЯМО В РАНТАЙМЕ!

Обертки Castle.DynamicProxy

Castle.DynamicProxy библиотека создания динамических прокси-оберток.

Генерация обертки в рантайме реализована через динамические модули .NET.

Обертки DynamicProxy (before-after)

```
class BeforeAfterAdvice : IInterceptor {
    public void Intercept(IInvocation invocation) {
        Console.WriteLine("Before");
        invocation.Proceed();
        Console.WriteLine("After");
    }
}
```

Обертки DynamicProxy (before-after)

```
class BeforeAfterAdvice : IInterceptor {
    public void Intercept(IInvocation invocation) {
        Console.WriteLine("Before");
        invocation.Proceed();
        Console.WriteLine("After");
    }
}
```

Совет

Обертки DynamicProxy (before-after)

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

Обертки DynamicProxy (before-after)

Создание сервиса с поддержкой трассировки

```
var command = BeforeAfterAspect.Apply<CommandService>();
command.Execute();
```

Метод с трассировкой в консоль

Обертки DynamicProxy (before-after)

Создание сервиса с поддержкой трассировки

```
var command = BeforeAfterAspect.Apply<CommandService>();
command.Execute();
```

Среза нет, точка прикрепления?

Обертки DynamicProxy (INPC-mixin)

Объявим INotifyPropertyChanged примесь

```
public class INPCMixIn : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    public void RaisePropertyChanged(object sender, string name)
        => PropertyChanged?.Invoke(sender, new PropertyChangedEventArgs(name));
}
```

Обертки DynamicProxy (INPC-mixin)

Объявим совет который при вызове сеттера вызовет вызовет событие в примеси MotifyerMixIn

```
public class PropertySetterAdvice : IInterceptor {

    readonly INPCMixIn linkedMixIn;
    public PropertySetterAdvice(INPCMixIn linkedMixIn)
    {
        this.linkedMixIn = linkedMixIn;
    }

    public void Intercept(IInvocation invocation) {
        invocation.Proceed();
        if (invocation.Method.Name.StartsWith("set_")) {
            linkedMixIn.RaisePropertyChanged(invocation.Proxy, invocation.Method.Name);
        }
    }
}
```

Обертки DynamicProxy (INPC-mixin)

```
class NotifyPropertyChangedAspect {
    public static TYPE Apply<TYPE>()
        where TYPE : class {

        var mixIn = new INPCMixIn();
        var setterAdvice = new PropertySetterAdvice(mixIn);

        var generator = new ProxyGenerator();
        var options = new ProxyGenerationOptions();
        options.AddMixinInstance(mixIn);

        var proxy = generator.CreateClassProxy<TYPE>(
            options, setterAdvice);

        return proxy;
    }
}
```

Обертки DynamicProxy (INPC-mixin)

Создание модели PotatoModel с примесью INotifyPropertyChanged

```
var model = NotifyPropertyChangedAspect.Apply<PotatoModel>();

((INotifyPropertyChanged)model)
    .PropertyChanged += (sender, args) =>
       Console.WriteLine($"Notification {args.PropertyName}");

model.Price = 4;
```

Вызов события в сеттере

Обертки DynamicProxy (INPC-mixin)

Создание модели PotatoModel с примесью INotifyPropertyChanged

```
var model = NotifyPropertyChangedAspect.Apply<PotatoModel>();

((INotifyPropertyChanged)model)
    .PropertyChanged += (sender, args) =>
       Console.WriteLine($"Notification {args.PropertyName}");

model.Price = 4;
```

Среза нет, точка прикрепления?

Обертки DynamicProxy

Достоинства:

- Динамические прокси
- Поддержка внедрения
- Поддержка IOC-контейнерами и mock-фреймворками (autofac, Windsor, moq, spring.net…)

Недостатки:

- Сложность добавления (фабрики, обертки)
- Часто подстраивать код под “аспект”
- Некоторые ограничения для внедрения
- Это все еще не АОП (нет срезов)

Статическое связывание

Как бегать с прибитыми к полу ногами

PostSharp

PostSharp - утилита реализующая АОП с помощью перекомпиляции сборок .NET.

Множество способов задать сложные срезы (декларативные и провайдеры аспектов)

Множество аспектов доступных из коробки

Postsharp (before-after)

Совет - трассировка в консоль

```
[PSerializable]
class PostSharpSimpleAspectAttribute : OnMethodBoundaryAspect {
    public override void OnEntry(MethodExecutionArgs args) {
        Console.WriteLine("Before");
        base.OnEntry(args);
    }

    public override void OnExit(MethodExecutionArgs args) {
        Console.WriteLine("After");
        base.OnExit(args);
    }
}
```

Postsharp (before-after)

Совет - трассировка в консоль

```
[PSerializable]
class PostSharpSimpleAspectAttribute : OnMethodBoundaryAspect {
    public override void OnEntry(MethodExecutionArgs args) {
        Console.WriteLine("Before");
        base.OnEntry(args);
    }

    public override void OnExit(MethodExecutionArgs args) {
        Console.WriteLine("After");
        base.OnExit(args);
    }
}
```

Совет

Postsharp (before-after)

```
[PostSharpSimpleAspect]
class CommandService : ICommand {
    public string Execute() {
        //Реализация
    }
}
```

Точка прикрепления

```
var command = new CommandService();
command.Execute();
```

Использование

Postsharp (INPC-mixin)

Для реализации INotifyPropertyChanged уже есть аспект NotifyPropertyChanged!

```
[NotifyPropertyChanged(AttributeTargetMembers = nameof(Price))]
class PotatoModel {
    public int Price { get; set; }
}
```

Postsharp (INPC-mixin)

Для реализации INotifyPropertyChanged уже есть аспект NotifyPropertyChanged!

```
[NotifyPropertyChanged(AttributeTargetMembers = nameof(Price))]
class PotatoModel {
    public int Price { get; set; }
}
```

Точка прикрепления

Postsharp (INPC-mixin)

Для реализации INotifyPropertyChanged уже есть аспект NotifyPropertyChanged!

```
[NotifyPropertyChanged(AttributeTargetMembers = nameof(Price))]
class PotatoModel {
    public int Price { get; set; }
}
```

Как реализовать срезы?

Postsharp - RegExp срез (before-after)

```
[PSerializable]
public class BeforeAfterAspect : TypeLevelAspect {

    [OnMethodEntryAdvice]
    [MulticastPointcut(Targets = MulticastTargets.Method)]
    public void OnEntry(MethodExecutionArgs args)
    {
        Console.WriteLine("Before");
    }

    [OnMethodExitAdvice]
    [MulticastPointcut(Targets = MulticastTargets.Method)]
    public void OnExit(MethodExecutionArgs args)
    {
        Console.WriteLine("After");
    }
}
```

Postsharp - RegExp срез (before-after)

```
[PSerializable]
public class BeforeAfterAspect : TypeLevelAspect {

    [OnMethodEntryAdvice]
    [MulticastPointcut(Targets = MulticastTargets.Method)]
    public void OnEntry(MethodExecutionArgs args)
    {
        Console.WriteLine("Before");
    }

    [OnMethodExitAdvice]
    [MulticastPointcut(Targets = MulticastTarget
    public void OnExit(MethodExecutionArgs args)
    {
        Console.WriteLine("After");
    }
}
```

Совет

Postsharp - RegExp срез

Применим Before-After аспект для всех моделей

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

Все сервисы теперь имеют трассировку методов!

PostSharp - аспекты

Кроме собственных аспектов можно использовать те что идут “из коробки”

Многопоточность - аспекты синхронизации и поиска тупиковых ситуаций

Архитектура - определяет видимость и отношения между компонентами

Контракты - различные декларативные контракты параметров методов

И многое другое :)

PostSharp

Достоинства:

- Сложные срезы (Regex-срезы, провайдеры аспектов и тп)
- Возможность конфигурации через XML
- Статическое связывание
- Поддержка внедрения
- Точки прикрепления через атрибуты
- Набор удобных аспектов из коробки

Недостатки:

- Замедление сборки
- Необходимо наличие вручную установленного PostSharp
- Цена 600 зеленых за одно рабочее место

Fody

Fody - утилита реализующая элементы АОП с помощью перекомпиляции сборок .NET.

“Аспекты” Fody распространяются через nuget в виде плагинов

Советы применяются с помощью мета-атрибутов

На данный момент на официальном сайте Fody и в nuGet распространяется более 100 плагинов

Fody - плагины

Более 100 плагинов на официальном GitHub и доступных из nuGet

WeakEvents - создает слабое событие вместо сильного

Obsolete - останавливает сборку при использовании Obsolete для определенных версий

Janitor - добавляет реализацию IDisposable

Equals - добавляет deep-Equals и deep-GetHashCode реализации

Mixins - добавляет примеси реализаций интерфейсов к классам

Как сделать настоящий AOP?

Где и как в .NET можно реализовать настоящие аспекты?

Не использовать сторонние библиотеки

Реализовывать любые советы

Использовать декларативно заданные срезы

Срезы для любых точек прикрепления

Nemerle

Nemerle - язык для платформы .NET с развитой системой макросов

Система макросов в Nemerle позволяют реализовать сложные трансформации кода на этапе компиляции.

Кроме того, imho, это просто замечательный язык (в сравнении с C#)

Nemerle

Цитирование - способ получить синтаксическое дерево из строкового представления

Квази-цитирование - цитирование, имеющее переменные внутри цитаты

Квази-цитирование в Nemerle выражается через конструкцию <[expression]>

Nemerle

Выражение языка

```
Console.WriteLine(40 + 2);
```

Nemerle

Выражение языка

```
Console.WriteLine(40 + 2);
```

```
Console.WriteLine(
        40
    +
        2
);
```

```
Expression.Call(
    ConsoleWriteLineMethodInfo,
    Expression.Add(
        Expression.Constant(40),
        Expression.Constant(2)
    )
);
```

Nemerle

Выражение языка

```
Console.WriteLine(40 + 2);
```

```
Console.WriteLine(
        40
    +
        2
);
```

```
Expression.Call(
    ConsoleWriteLineMethodInfo,
    Expression.Add(
        Expression.Constant(40),
        Expression.Constant(2)
    )
);
```

Nemerle

Выражение языка

```
Console.WriteLine(40 + 2);
```

```
Console.WriteLine(
        40
    +
        2
);
```

```
Expression.Call(
    ConsoleWriteLineMethodInfo,
    Expression.Add(
        Expression.Constant(40),
        Expression.Constant(2)
    )
);
```

Nemerle

Выражение языка

```
Console.WriteLine(40 + 2);
```

```
Console.WriteLine(
        40
    +
        2
);
```

```
Expression.Call(
    ConsoleWriteLineMethodInfo,
    Expression.Add(
        Expression.Constant(40),
        Expression.Constant(2)
    )
);
```

Nemerle

Выражение языка

```
Console.WriteLine(40 + 2);
```

```
Console.WriteLine(
        40
    +
        2
);
```

```
Expression.Call(
    ConsoleWriteLineMethodInfo,
    Expression.Add(
        Expression.Constant(40),
        Expression.Constant(2)
    )
);
```

Nemerle

Выражение языка

```
Console.WriteLine(40 + 2);
```

Синтаксическое дерево (например, в формате Linq Expressions Tree)

```
Expression.Call(ConsoleWriteLineMethodInfo,
    Expression.Add(Expression.Constant(40), Expression.Constant(2)));
```

Nemerle

Выражение языка

```
Console.WriteLine(40 + 2);
```

Цитата выражения в Nemerle (в формате Nemerle-Ast)

```
<[ Console.WriteLine(40 + 2); ]>;
```

Nemerle

Выражение языка

```
Console.WriteLine(40 + 2);
```

Квази-цитата выражения в Nemerle (в формате Nemerle-Ast)

```
def sum = <[ 40 + 2 ]>;
<[ Console.WriteLine($sum); ]>;
```

Nemerle (before-after)

```
macro BeforeAfterAdvice(typeBuilder : TypeBuilder)
{
  foreach (member is ClassMember.Function in typeBuilder.Ast.GetMembers()){
  member.Body = <[
       System.Console.WriteLine("Before");
       def result = $(member.Body);
       System.Console.WriteLine("After");
       result]>;
  }
}
```

Во все функции добавим совет трассировки

Nemerle (INPC-mixin)

Добавим интерфейс INPC и его реализацию

```
macro INPCAdvice(typeBuilder : TypeBuilder)
{
    typeBuilder.AddImplementedInterface(<[ INotifyPropertyChanged ]>);
    typeBuilder.Define(
          <[decl: public event PropertyChanged : PropertyChangedEventHandler; ]>);
}
```

Nemerle (INPC-mixin)

Во все авто-сеттеры добавим вызов INPC-события

```
macro INPCAdvice(typeBuilder : TypeBuilder)
{
    foreach(property is PropertyBuilder in typeBuilder.GetProperties()){
        when(property.IsAutoProperty && property.CanWrite)
          when(property.Setter is MethodBuilder as setter)
              setter.Body = <[ $(setter.Body);
              when(PropertyChanged != null)
                  PropertyChanged(this, PropertyChangedEventArgs($(property.Name : string))); ]>;
    }
}
```

Nemerle

```
[BeforeAfterAdvice]
class CommandService : ICommand{
    public Execute() : string{
        //реализация Execute()
    }
}

[INPCAdvice]
class PotatoModel{
    public Price : int { get; set; }
}
```

Nemerle

```
[BeforeAfterAdvice]
class CommandService : ICommand{
    public Execute() : string{
        //реализация Execute()
    }
}

[INPCAdvice]
class PotatoModel{
    public Price : int { get; set; }
}
```

Применим макросы к типам
(точки прикрепления)

Окей, а как декларативные срезы?!

Nemerle - regex срезы

Добавим advice ко всем классам в сборке удовлетворяющим pointCut

```
macro RegexPointCutAspect(pointCut : string, advice : PExpr)
{
  def regex = Regex(pointCut);
  def typeBuilders = Macros.ImplicitCTX().Env.NameTree.NamespaceTree.GetTypeBuilders();
  foreach(typeBuilder in typeBuilders){
      when(regex.IsMatch(typeBuilder.Name))
          typeBuilder.Ast.AddCustomAttribute(advice);
  }
}
```

Nemerle - regex срезы

Аспекты с RegEx срезами

```
[assembly: RegexPointCutAspect(".*Model", INPCAdvice)]

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

Nemerle - regex срезы

Аспекты с RegEx срезами

Все типы моделей поддерживают INPC

```
[assembly: RegexPointCutAspect(".*Model", INPCAdvice)]
```

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

Все типы сервисов поддерживают трассировку

Nemerle - regex срезы

Использование

```
def command  = CommandService();
_ = command.Execute();

def model = PotatoModel();
model.PropertyChanged += fun(_, arg) { WriteLine(arg.PropertyName) };
model.Price = 42;
```

Nemerle - regex срезы

Использование

```
def command  = CommandService();
_ = command.Execute();

def model = PotatoModel();
model.PropertyChanged += fun(_, arg) { WriteLine(arg.PropertyName) };
model.Price = 42;
```

Трассировка

Оповещение

Nemerle

Достоинства:

- Очень гибкая система макросов позволяют реализовать аспекты любой сложности

Недостатки:

- Замедление сборки
- Не распространенный язык
- Гибкая но сложная подсистема макросов

Заключение

...и мораль

Когда и где?

Мне нужно быстро добавить функциональность во множество классов или методов

Я могу легко описать декларативно срез для всех точек прикрепления

Я могу легко написать один совет для целого среза

Заключение

ООП языки и, хуже того, статически-компилируемые языки плохо подходят для хорошей реализации АОП!

ООП языки с поддержкой метапрограммирования лучше подходят для реализации АОП

Однако, уже с тем, что есть, можно:

- Сильно сократить код
- Произвести декомпозицию
- В краткие сроки добавлять функциональность

Ссылки

- en.wikipedia.org/wiki/Aspect-oriented_programming - АОП в Википедии
- ibm.com/developerworks/ru/library/j-aopwork15/ - Мифы и реальности АОП
- castleproject.org/projects/dynamicproxy/ - Динамические прокси
- github.com/Fody/Fody - Fody
- postsharp.net/ - PostSharp
- nemerle.org/About - Nemerle
- Архив с примерами

Спасибо

Благодарю за внимание!

Контейнеры Autofac (before-after)


```
static class ContainerCreator
{
    public static IContainer CreateContainter()
    {
        var builder = new ContainerBuilder();

        builder.Register(c => new BeforeAfterAdvice());

        builder.RegisterType<CommandService>()
               .As<ICommand>()
               .InterceptedBy(typeof(BeforeAfterAdvice));

        return builder.Build();
    }
}
```

Контейнеры Autofac (before-after)

Получение сервиса с поддержкой трассировки

```
var container = ContainerCreator.CreateContainter();

var command = container.Resolve<ICommand>();
command.Execute();
```

АОП-фреймворки