

Source Generators в действии

Andrey Dyatlov
Software engineer at JetBrains
Working on ReSharper / Rider



@a_tessenr



TessenR



В докладе

- Генераторы – как это работает и как с этим работать
- Live Demo x2!
- Сравнение со старыми технологиями
- Продвинутые сценарии работы
- Best practices

Какие задачи должны решить генераторы?

- Генерация шаблонного кода
 - Шаблонная реализация `Equals`, `GetHashCode`, `==`, `ToString`
 - Генерация типов по схеме
 - Object mapping (например, AutoMapper)
 - Материализация объектов БД

Какие задачи должны решить генераторы?

- Генерация шаблонного кода
 - Шаблонная реализация `Equals`, `GetHashCode`, `==`, `ToString`
 - Генерация типов по схеме
 - Object mapping (например, AutoMapper)
 - Материализация объектов БД
- Оптимизация приложения
 - Регистрация типов для dependency injection без рефлексии
 - Методы сериализации без рефлексии

Давай на примере!

```
public interface INotifyPropertyChanged
{
    event PropertyChangedEventHandler PropertyChanged;
}
```

Давай на примере!

```
public class CarModel : INotifyPropertyChanged
{
    public double SpeedKmPerHour { get; set; }
    public event PropertyChangedEventHandler? PropertyChanged;
}
```

И начался бойлерплейт...

```
public class CarModel : INotifyPropertyChanged
{
    private double SpeedKmPerHourBackingField;
    public double SpeedKmPerHour
    {
        get => SpeedKmPerHourBackingField;
        set
        {
            SpeedKmPerHourBackingField = value;
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(nameof(SpeedKmPerHour)));
        }
    }
    public event PropertyChangedEventHandler? PropertyChanged;
}
```

И начался бойлерплейт...

```
public class CarModel : INotifyPropertyChanged
{
    private double SpeedKmPerHourBackingField;
    public double SpeedKmPerHour
    {
        get => SpeedKmPerHourBackingField;
        set
        {
            SpeedKmPerHourBackingField = value;
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(nameof(SpeedKmPerHour)));
        }
    }
    public event PropertyChangedEventHandler? PropertyChanged;
}
```


И чем тут помогут генераторы?

Код сейчас

```
class CarModel : INotifyPropertyChanged
{
    private double SpeedKmPerHourBackingField;
    public double SpeedKmPerHour
    {
        get => SpeedKmPerHourBackingField;
        set
        {
            SpeedKmPerHourBackingField = value;
            PropertyChanged?.Invoke(this,
                new PropertyChangedEventArgs(nameof(SpeedKmPerHour)));
        }
    }
    public event PropertyChangedEventHandler? PropertyChanged;
}
```

Код с генераторами

```
partial class CarModel : INotifyPropertyChanged
{
    private double SpeedKmPerHourBackingField;
}
```

И чем тут помогут генераторы?

Код сейчас

```
class CarModel : INotifyPropertyChanged
{
    private double SpeedKmPerHourBackingField;
    private string ModelBackingField = "";

    public double SpeedKmPerHour
    {
        get => SpeedKmPerHourBackingField;
        set
        {
            SpeedKmPerHourBackingField = value;
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(nameof(SpeedKmPerHour)));
        }
    }

    public string Model
    {
        get => ModelBackingField;
        set
        {
            ModelBackingField = value;
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(nameof(Model)));
        }
    }

    public event PropertyChangedEventHandler? PropertyChanged;
}
```

Код с генераторами

```
partial class CarModel : INotifyPropertyChanged
{
    private double SpeedKmPerHourBackingField;
    private string ModelBackingField = "";
}
```

И чем тут помогут генераторы?

Код сейчас

```
class CarModel : INotifyPropertyChanged
{
    private double SpeedKmPerHourBackingField;
    private int NumberOfDoorsBackingField;
    private string ModelBackingField = "";

    public double SpeedKmPerHour
    {
        get => SpeedKmPerHourBackingField;
        set
        {
            SpeedKmPerHourBackingField = value;
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(nameof(SpeedKmPerHour)));
        }
    }

    public int NumberOfDoors
    {
        get => NumberOfDoorsBackingField;
        set
        {
            NumberOfDoorsBackingField = value;
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(nameof(NumberOfDoors)));
        }
    }

    public string Model
    {
        get => ModelBackingField;
        set
        {
            ModelBackingField = value;
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(nameof(Model)));
        }
    }

    public event PropertyChangedEventHandler? PropertyChanged;
}
```

Код с генераторами

```
partial class CarModel : INotifyPropertyChanged
{
    private double SpeedKmPerHourBackingField;
    private int NumberOfDoorsBackingField;
    private string ModelBackingField = "";
}
```

И чем тут помогут генераторы?

Код сейчас

```
class CarModel : INotifyPropertyChanged
{
    private double SpeedKmPerHourBackingField;
    private int NumberOfDoorsBackingField;
    private string ModelBackingField = "";

    public double SpeedKmPerHour
    {
        get => SpeedKmPerHourBackingField;
        set
        {
            SpeedKmPerHourBackingField = value;
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(nameof(SpeedKmPerHour)));
        }
    }

    public int NumberOfDoors
    {
        get => NumberOfDoorsBackingField;
        set
        {
            NumberOfDoorsBackingField = value;
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(nameof(NumberOfDoors)));
        }
    }

    public string Model
    {
        get => ModelBackingField;
        set
        {
            ModelBackingField = value;
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(nameof(Model)));
        }
    }

    public void SpeedUp() => SpeedKmPerHour *= 1.1;

    public event PropertyChangedEventHandler? PropertyChanged;
}
```

Код с генераторами

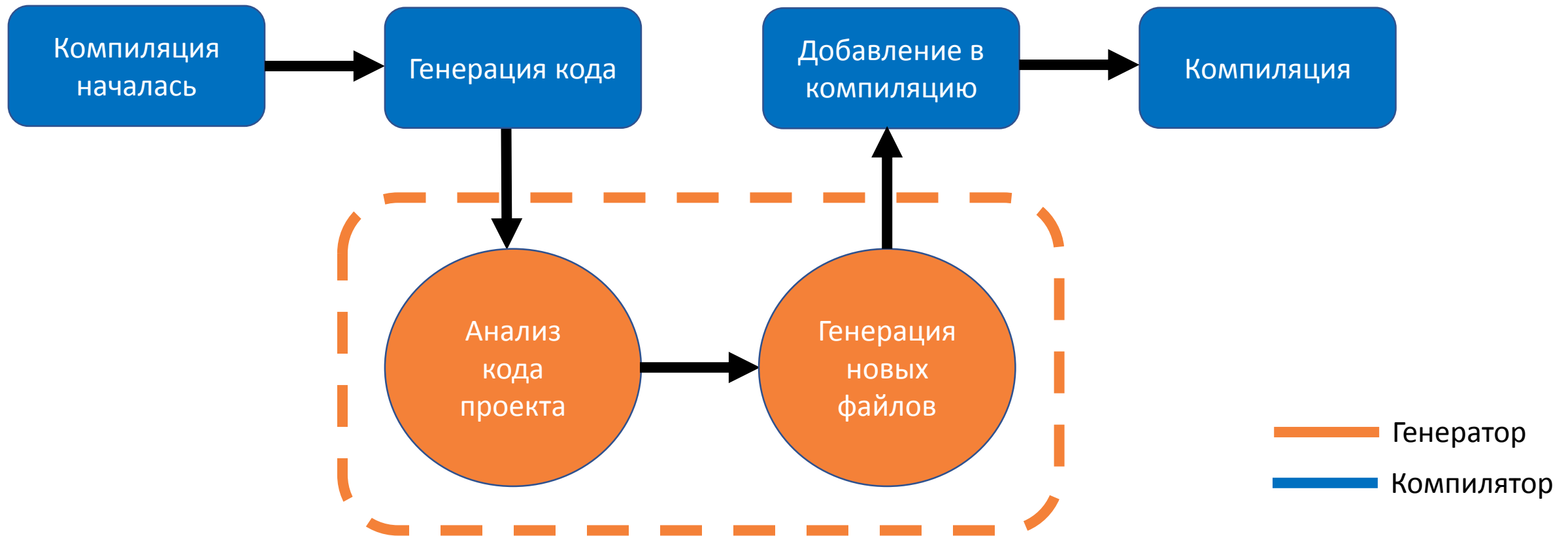
```
partial class CarModel : INotifyPropertyChanged
{
    private double SpeedKmPerHourBackingField;
    private int NumberOfDoorsBackingField;
    private string ModelBackingField = "";

    public void SpeedUp() => SpeedKmPerHour *= 1.1;
}
```

Так что такое эти генераторы?

- Новая технология метапрограммирования от Microsoft
- Часть процесса компиляции
- Доступ к модели вашего кода
- Результат генератора – новые файлы

Как это работает?



```
goto demo;
```

I KNOW
SOURCE GENERATORS!



Это же уже было в Fody?

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

    public string GivenNames { get; set; }
    public string FamilyName { get; set; }
    public string FullName => $"{GivenNames} {FamilyName}";
}
```



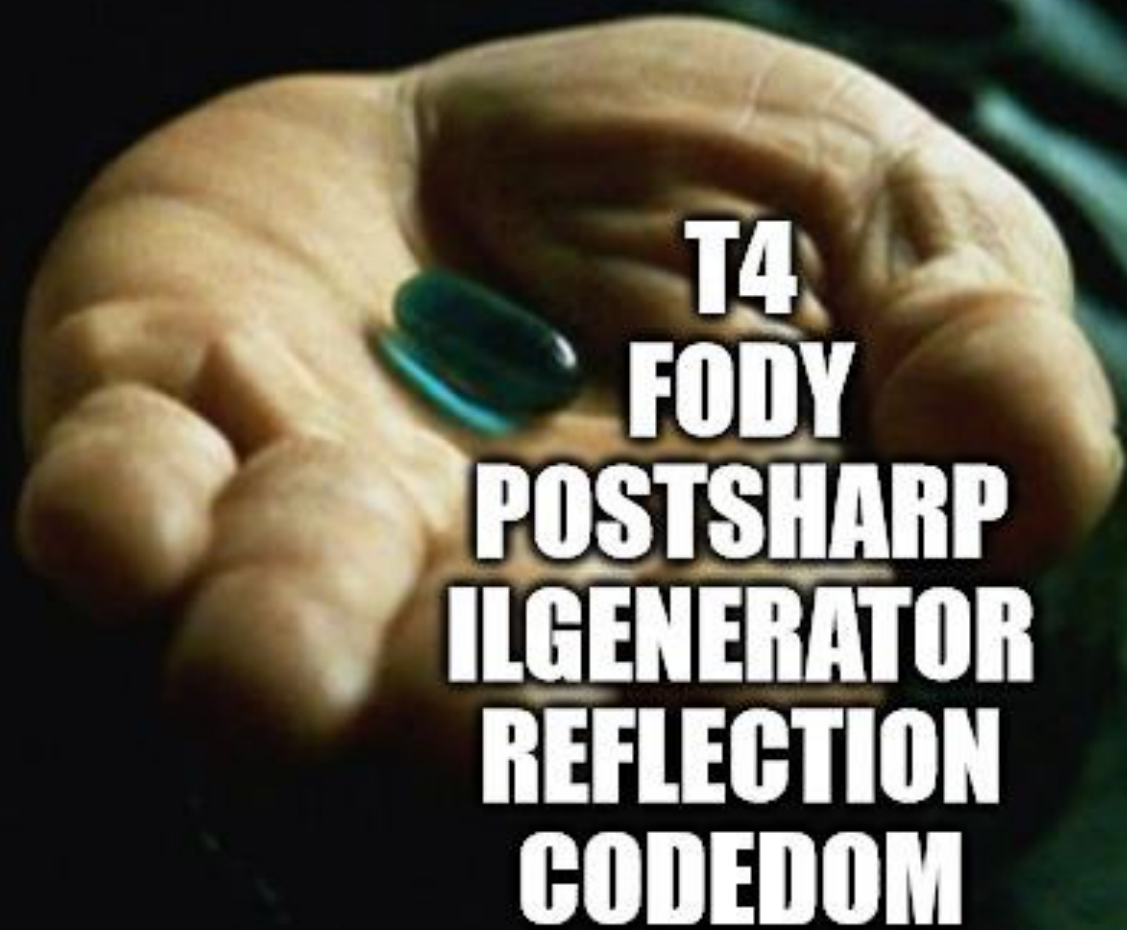
bit.ly/3iWIda9

Это же уже было в Fody?

```
string givenNames;  
public string GivenNames  
{  
    get => givenNames;  
    set  
    {  
        if (value != givenNames)  
        {  
            givenNames = value;  
            PropertyChanged?.Invoke(this, InternalEventArgsCache.GivenNames);  
            PropertyChanged?.Invoke(this, InternalEventArgsCache.FullName);  
        }  
    }  
}
```



**SOURCE
GENEARTORS**



**T4
FODY
POSTSHARP
ILGENERATOR
REFLECTION
CODEDOM**

Генераторы vs IL weaving

Генераторы

+/- Только добавляют файлы

IL weaving

+/- Переписывает байткод

Как это выглядит с Fody?

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

    public string GivenNames { get; set; }
    public string FamilyName { get; set; }
    public string FullName => $"{GivenNames} {FamilyName}";
}
```



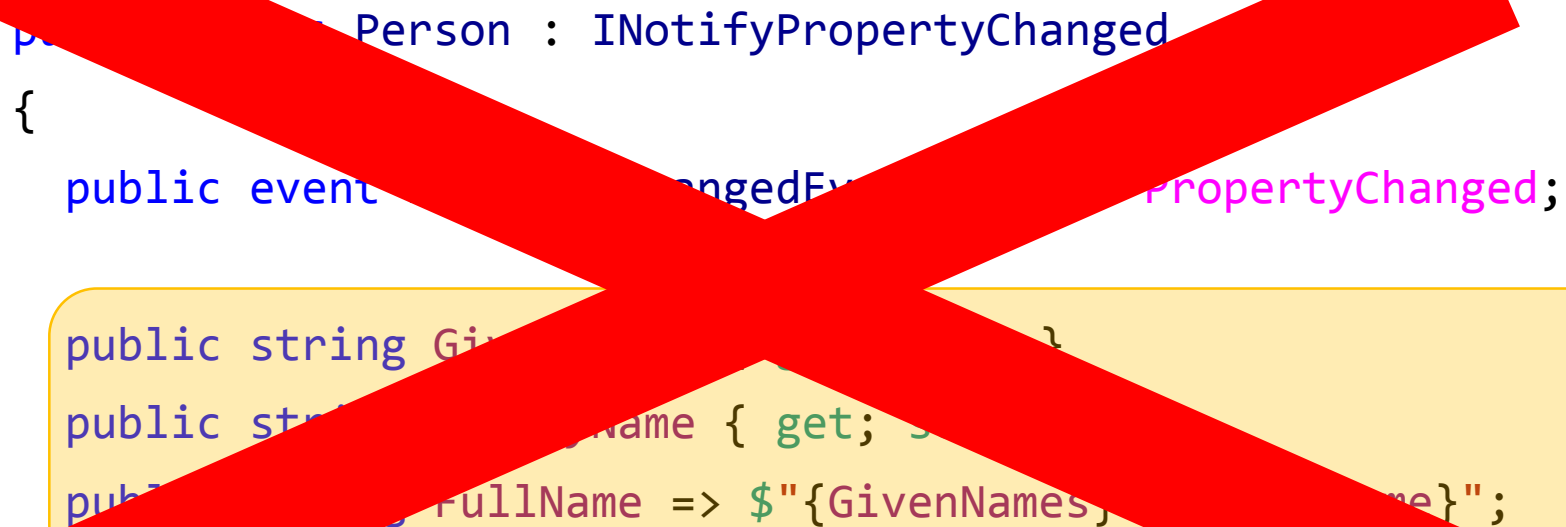
bit.ly/3iWIda9

Как это выглядит с Fody?

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

    public string GivenNames { get; set; }
    public string FamilyName { get; set; }
    public string FullName => $"{GivenNames} {FamilyName}";
}
```

Как это выглядит с Fody?



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

    public string GivenName { get; }
    public string Surname { get; }
    public string FullName => $"{GivenName} {Surname}";
}
```

Какие опции есть у генераторов?

- Конвенции
 - Реализованные интерфейсы
 - Схема имен бэкинг филдов
- Атрибуты
 - Указать имя свойства
 - Дополнительные нотификации
- Конфигурационные файлы

Генератор с помощью атрибутов

```
public partial class Person : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    [AutoNotify("GivenNames", alsoNotify: "FullName")]
    private string givenNames;

    [AutoNotify("FamilyName", alsoNotify: "FullName")]
    private string familyName;

    public string FullName => $"{GivenNames} {FamilyName}";
}
```

Генератор с помощью атрибутов

```
[PropertyWithNotification(typeof(string), "GivenNames", alsoNotify: "FullName")]  
[PropertyWithNotification(typeof(string), "FamilyName", alsoNotify: "FullName")]  
public partial class Person : INotifyPropertyChanged  
{  
    public event PropertyChangedEventHandler PropertyChanged;  
  
    public string FullName => $"{GivenNames} {FamilyName}";  
}
```

В любом случае в компиляции
будет и свойство и поле

```
public partial class Person {  
    string givenNames;  
    public string GivenNames {  
        get => givenNames;  
        set {  
            givenNames = value;  
            PropertyChanged?.Invoke(this, nameof(GivenNames));  
        }  
    }  
}
```

Мусор или исходники для дебага?

```
public string SayHello()  
=> $"Hello, I'm {FullName},"  
+ $" but you can call me {given}";
```



Мусор или исходники для дебага?

```
public string SayHello()  
=> $"Hello, I'm {FullName}, "  
+ $" but you can call me {given}";
```



- Можно случайно использовать поле вместо свойства
- [Obsolete] + #pragma warning disable в сгенерированном коде

Генераторы vs IL weaving

Генераторы

- +/- Только добавляют файлы
- + Компилятор проверяет код
- + Можно смотреть и дебажить код
- + Можно протестировать
- + Ниже порог вхождения
- Требуется partial
- Нельзя поменять существующий код

IL weaving

- +/- Переписывает байткод
- InvalidProgramException
- Счастливого дебага
- Код должен компилироваться до модификации
- + Могут удалять и менять любой код, например, для оптимизации

Так IL weaving мертв?

- Многие модификации легко заменить
 - Fody:
 - PropertyChanged
 - Equatable
 - ToString
 - Visualize
 - With...
 - PostSharp
 - ToString
 - IStructuralEquality
 - DependencyProperty...

Не так быстро...

- IL-weaving может модифицировать методы
 - Fody
 - Undisposed
 - MethodDecorator
 - Tracer
 - SwallowExceptions...
 - PostSharp
 - Caching
 - ICommand
 - Logging

Так IL weaving мертв? Не так быстро...

Code rewriting

- Optimization
- Logging injection
- IL Weaving
- Call site re-writing

While these techniques have many valuable use cases,
they do not fit into the idea of *source generation*.



bit.ly/3kIBzET

LoggingInjection

Если нельзя но очень хочется...

- В начале метода
 - информация о вызове
 - логирование аргументов
- В конце метода
 - Затраченное время
 - Возвращенное значение (если есть)
 - Информация об исключениях

PostSharp Logging injection

```
[Log]
public Request(int id)
{
    Id = id;
}
```



bit.ly/32TVNpj



bit.ly/2G33sJ6

PostSharp Logging injection

```
public Request(int id) {  
    if (localState.IsEnabled(LogLevel.Debug)) {  
        logRecordInfo = new LogRecordInfo(MethodEntry, ...);  
        recordBuilder1.SetParameter<int>(..., id);  
    }  
    try {  
        this.Id = id;  
        logRecordInfo = new LogRecordInfo(MethodSuccess, ...);  
    }  
    catch (Exception ex) {  
        logRecordInfo = new LogRecordInfo(MethodException, ...);  
        recordBuilder1.SetException(ex);  
        throw;  
    }  
}
```

75 строк кода

PostSharp Logging injection

```
public Request(int id) {  
    if (localState.IsEnabled(LogLevel.Debug)) {  
        logRecordInfo = new LogRecordInfo(MethodEntry, ...);  
        recordBuilder1.SetParameter<int>(..., id);  
    }  
    try {  
        this.Id = id;  
        logRecordInfo = new LogRecordInfo(MethodSuccess, ...);  
    }  
    catch (Exception ex) {  
        logRecordInfo = new LogRecordInfo(MethodException, ...);  
        recordBuilder1.SetException(ex);  
        throw;  
    }  
}
```

75 строк кода

PostSharp Logging injection

```
public Request(int id) {  
    if (localState.IsEnabled(LogLevel.Debug)) {  
        logRecordInfo = new LogRecordInfo(MethodEntry, ...);  
        recordBuilder1.SetParameter<int>(..., id);  
    }  
    try {  
        this.Id = id;  
        logRecordInfo = new LogRecordInfo(MethodSuccess, ...);  
    }  
    catch (Exception ex) {  
        logRecordInfo = new LogRecordInfo(MethodException, ...);  
        recordBuilder1.SetException(ex);  
        throw;  
    }  
}
```

75 строк кода

PostSharp Logging injection

```
public Request(int id) {  
    if (localState.IsEnabled(LogLevel.Debug)) {  
        logRecordInfo = new LogRecordInfo(MethodEntry, ...);  
        recordBuilder1.SetParameter<int>(..., id);  
    }  
    try {  
        this.Id = id;  
        logRecordInfo = new LogRecordInfo(MethodSuccess, ...);  
    }  
    catch (Exception ex) {  
        logRecordInfo = new LogRecordInfo(MethodException, ...);  
        recordBuilder1.SetException(ex);  
        throw;  
    }  
}
```

75 строк кода

Можно ли это реализовать генераторами?

```
interface IAccountingService
```

```
{
```

```
    AccountsSet GetAccounts(Client client);
```

```
    decimal GetTotalBalance(AccountsSet accounts);
```

```
}
```

```
class AccountingServiceCore : IAccountingService
```

```
{
```

```
    public AccountsSet GetAccounts(Client client) => ...
```

```
    public decimal GetTotalBalance(AccountsSet accounts) => ...
```

```
}
```


Сгенерировать декоратор

```
class AccountingServiceLoggingProxy : IAccountingService {  
    private readonly ILogger _logger = GetLogger();  
    private readonly IAccountingService _businessLogic;  
  
    public AccountingServiceLoggingProxy(IAccountingService businessLogic)  
        => _businessLogic = businessLogic;  
}
```

Сгенерировать декоратор

```
public AccountsSet GetAccounts(Client client) {  
    _logger.Log(LogLevel.Info, $"nameof(GetAccounts)} started with Client = {client}...");  
    var sw = new Stopwatch(); sw.Start();  
    try {  
        var result = _businessLogic.GetAccounts(client);  
        sw.Stop();  
        _logger.Log(LogLevel.Info, $"nameof(GetAccounts)}  
                                returned {result} in {sw.ElapsedMilliseconds} ms");  
        return result;  
    }  
    catch (Exception e) {  
        _logger.Log(LogLevel.Error, $" Exception {e}"); throw;  
    }  
}
```

Но всё заменить не выйдет

- Fody
 - NullGuard / RuntimeNullable
 - Caseless
 - ConfigureAwait
 - LoggerIsEnabled
- PostSharp:
 - Threading – deadlock detection
 - UnsafeMemoryChecker

Или всё же?...

- Генератор видит весь исходный код
- Вы можете создать «шаблон» метода или всего типа
- Исследовать его генератором
- Сгенерировать дубликат метода \ типа
- Вставить в дубликат нужные строчки

Или все же?...

В исходном коде

```
public class MyTypeTemplate {  
    public void DoSomething() {  
        lock (typeof(MyTypeTemplate)) {  
            Console.WriteLine("Written under lock");  
            LogicUnderLock();  
        }  
    }  
}
```

Или все же?...

В исходном коде

```
public class MyTypeTemplate {  
    public void DoSomething() {  
        lock (typeof(MyTypeTemplate)) {  
            Console.WriteLine("Written under lock");  
            LogicUnderLock();  
        }  
    }  
}
```

Создано генератором

```
public class MyType {  
    public void DoSomething() {  
        lock (typeof(MyType)) {  
            Console.WriteLine("MyType lock taken");  
            Console.WriteLine("Written under lock");  
            LogicUnderLock();  
            Console.WriteLine("MyType lock released");  
        }  
    }  
}
```

ILWeaving становится нишевым

- Большое число заменяемых плагинов
- Небольшая разница в реализации
 - ILWeaving сложнее дебажить и тестировать
 - На этапе компиляции не видно деталей реализации
- Часть возможностей уникальна и не заменяется генераторами

Рантайм генерация - ILGenerator

Можно заменить генераторами

- Генерация шаблонного кода чтобы сэкономить время
 - Логирование
 - Сериализация
 - Материализация объектов БД
 - Автомапперы
 - Шифрование

Зависит от рантайм данных

- Обработка рантайм аргументов
 - `Expression<T>.Compile()`
 - Генерация кода для проверки регулярного выражения

Как насчет рефлексии?

Рефлексия часто просто сокращает код

- Сериализация
 - Поиск полей и свойств типа рефлексией
 - Newtonsoft.Json, System.Text.Json, DataContractSerializer...

Рефлексия часто просто сокращает код

- Сериализация
 - Поиск полей и свойств типа рефлексией
 - Newtonsoft.Json, System.Text.Json, DataContractSerializer...

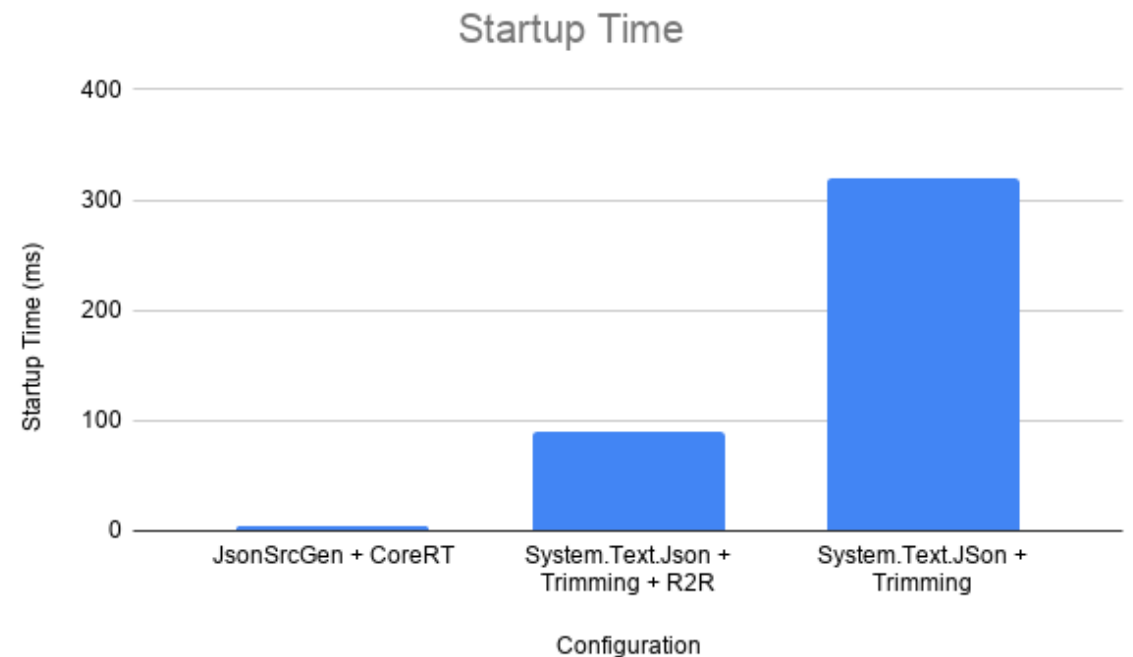
- А что если просто написать код сериализации для каждого типа?

- JsonSrcGen



bit.ly/3erWUR6

bit.ly/33dFZgV



Как насчет рефлексии?

Можно заменить генераторами


- Входные данные доступны на этапе компиляции
 - Информация для сериализации
 - Dependency injection

```
var builder = new ContainerBuilder();  
var asm = GetExecutingAssembly();  
builder.RegisterAssemblyTypes(asm)  
    .AsImplementedInterfaces();
```

Зависит от рантайм данных

- Исследование объектов в рантайме
- Обход ограничений компилятора
 - Работа с приватными методами \ полями \ свойствами

Так как поменяется мир метапрограммирования?

- ILWeaving – Fody, PostSharp
 - Становится нишевым но имеет преимущества
- ILGenerator, Reflection
 - Все зависит от ваших сценариев
- T4 – скорее всего уйдет
 - Как и генераторы только создает новые файлы
 - Свой синтаксис не имеющий прямой поддержки в IDE
 - Рослин уже использует генераторы где мог бы быть T4
 -  bit.ly/3hXk0zt
 - Может создавать не только C# файлы и вы контролируете время запуска

Проблемы при использовании генераторов

- Как не создать проблем потребителям вашего генератора?
- Что если нужны дополнительные данные?
- Как быть если генераторов несколько?
- Как подсказать коллегам что будет добавлено генератором?

Вам не нужно писать генератор самим!

- Можно просто подключить генератор как обычный Nuget пакет
- Поделиться генератором не сложнее чем Roslyn-анализатором

Помните что пишете код для чужого проекта

- Проверяйте версию языка целевого проекта
- Думайте обо всех сценариях

Помните что пишете код для чужого проекта

- Проверяйте версию языка целевого проекта
- Думайте обо всех сценариях
- AutoNotify пример от Microsoft:

```
// begin building the generated source
StringBuilder source = new StringBuilder($"
namespace {namespaceName}
{{
    public partial class {classSymbol.Name} : {notifySymbol.ToDisplayString()}
    {{
");
```



bit.ly/360h6Gb

Укажите зависимости генерируемого кода

- Укажите зависимость от пакета

```
<PackageReference Include="Newtonsoft.Json" Version="12.0.1" />
```

- Проверьте что он есть!

```
if (!context.Compilation.ReferencedAssemblyNames.Any(ai =>  
    ai.Name.Equals("Newtonsoft.Json", StringComparison.OrdinalIgnoreCase)))  
{  
    context.ReportDiagnostic(...);  
}
```



bit.ly/2TTGR5l

C# кода часто недостаточно

- Генератору могут потребоваться дополнительные данные
 - xml-файл с настройками
 - схема генерируемых типов
 - конфигурационная информация
- Они не являются частью C# проекта и должны быть подключены отдельно

Дополнительные файлы для генератора

- Просто добавьте их с .csproj в теге

```
<AdditionalFiles Include="Syntax\Syntax.xml" />
```

- И вы сможете прочитать их из контекста генератора

- `var syntaxXml = context.AdditionalFiles.SingleOrDefault(a => Path.GetFileName(a.Path) == "Syntax.xml");`



bit.ly/38rDv23

Дополнительные файлы для генератора

- Также можно прикрепить к файлу свойства доступные генератору

```
<CompilerVisibleItemMetadata Include="AdditionalFiles"  
                             MetadataName="NamespaceName" />
```

```
<CompilerVisibleItemMetadata Include="AdditionalFiles"  
                             MetadataName="ClassName" />
```

```
<ItemGroup>  
  <AdditionalFiles Include="Assets/Sample.svg"  
                  NamespaceName="Assets" ClassName="Sample" />  
</ItemGroup>
```



bit.ly/3cwYYXi

Доступ к MSBuild-свойствам

- Любая топ-уровневая конфигурация
 - Включить\отключить генератор
 - Выдавать ли диагностическую информацию из генератора и куда
 - Заходить в `Debugger.Launch()` только при компиляции с флагом
- Конфигурация созданного кода
 - Неймспейс для сгенерированных типов
 - Выбор логгинг фреймворка
 - Уровень логирования отдельных элементов `trace / info / etc`
 - Что именно логировать (параметры, время, возвращенное значение etc)

Доступ к MSBuild-свойствам

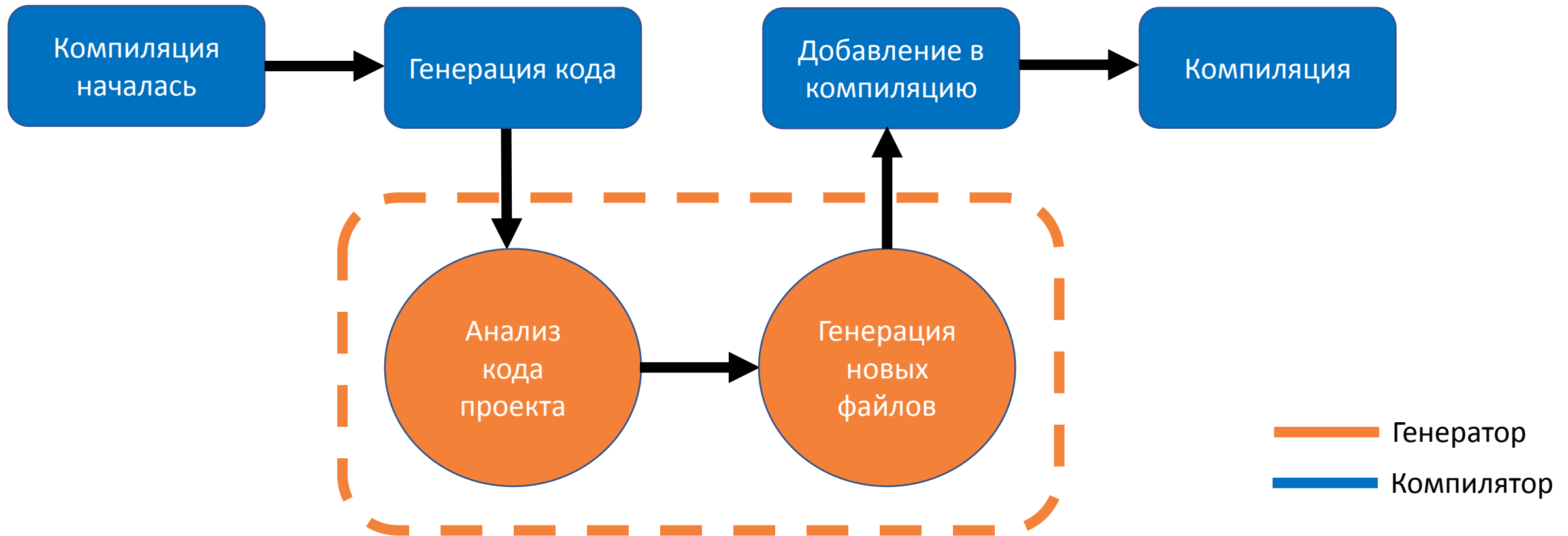
- Объявите свойства в .csproj
 - `<CompilerVisibleProperty Include="EnableLogging" />`
- Прочитать свойство из контекста в генераторе
 - `context.AnalyzerConfigOptions.GlobalOptions
 .TryGetValue("build_property.EnableLogging",
 out var emitLoggingSwitch);`



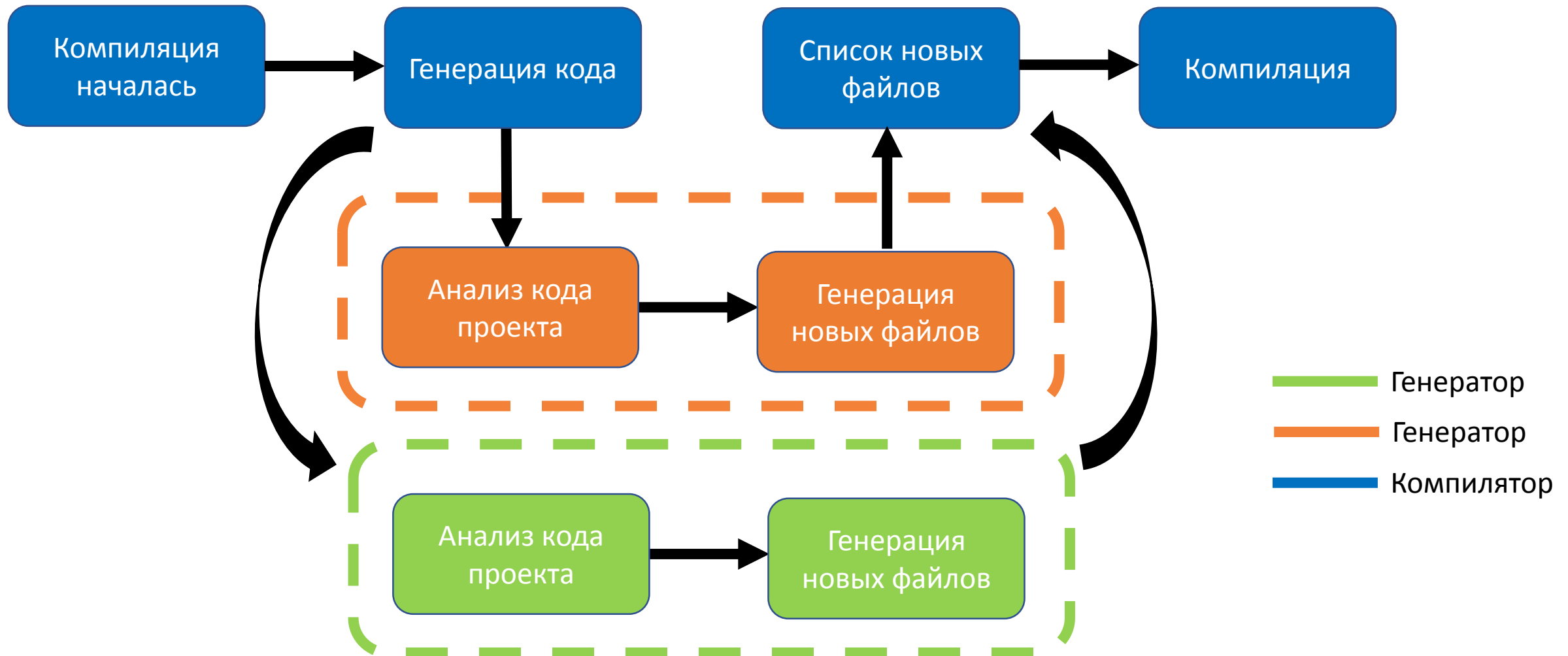
bit.ly/301UHpK

bit.ly/2HrYztF

А с чем будет работать второй генератор?



А с чем будет работать второй генератор?



Зависимости между генераторами

- Генераторы только создают код в текстовом виде
 - Если зависимостей еще нет это не проблема
- Файлы от всех генераторов появляются одновременно
- Вы только добавляете новые файлы
 - Можно абстрагировать любой вызов за интерфейсом

Logging + Caching

```
interface ILogic { ... }  
class LogicImpl : ILogic { ... }
```

Исходный код

Logging + Caching

```
interface ILogic { ... }  
class LogicImpl : ILogic { ... }
```

```
class LoggingLogic : ILogic {  
    private readonly ILogger _logger = GetLogger();  
    private readonly ILogic _coreLogic;  
}
```

Исходный код

Генератор 1

Logging + Caching

```
interface ILogic { ... }  
class LogicImpl : ILogic { ... }
```

```
class LoggingLogic : ILogic {  
    private readonly ILogger _logger = GetLogger();  
    private readonly ILogic _coreLogic;  
}
```

```
class CachingLogic : ILogic {  
    private readonly Dictionary<IArgument, IResult> _cache = new();  
    private readonly ILogic _coreLogic;  
}
```

- Исходный код
- Генератор 1
- Генератор 2

Logging + Caching

- Собрать правильную последовательность в контейнере зависимостей
- Собрать вручную

```
var logsAllCalls = new LoggingLogic(new CachingLogic(new LogicImpl()));
```

```
var logsUniqueArguments = new CachingLogic(new LoggingLogic(new LogicImpl()));
```

Используйте новый partial

```
partial class MyType
{
    partial void OnModelCreating(string input); // C# 8
}
```



bit.ly/33VwEK6

Используйте новый partial

```
partial class MyType
{
    partial void OnModelCreating(string input); // C# 8

    public partial bool IsPetMatch(string input); // C# 9
}
```



bit.ly/33VwEK6

Используйте новый partial

```
partial class MyType
{
    partial void OnModelCreating(string input); // C# 8

    public partial bool IsPetMatch(string input); // C# 9
}
```

```
partial class MyType
{
    public partial bool IsPetMatch(string input)
        => input is "dog" or "cat" or "fish";
}
```



bit.ly/33VwEK6

Используйте новый partial

```
partial class MyType
{

    [RegexGenerated("(dog|cat|fish)")]
    public partial bool IsPetMatch(string input); // C# 9

}
```

```
partial class MyType
{
    public partial bool IsPetMatch(string input)
        => input is "dog" or "cat" or "fish";
}
```

Source Generated



bit.ly/33VwEK6

Best practices

- Как сделать генератор конфигурируемым
- Как предоставить пользователю необходимые атрибуты
- Как сообщить потребителю генератора о проблеме
- Какие проблемы нужно предусмотреть в генераторе
- Скорость работы – переиспользуйте обход кода компилятором

goto demo;





Best practices

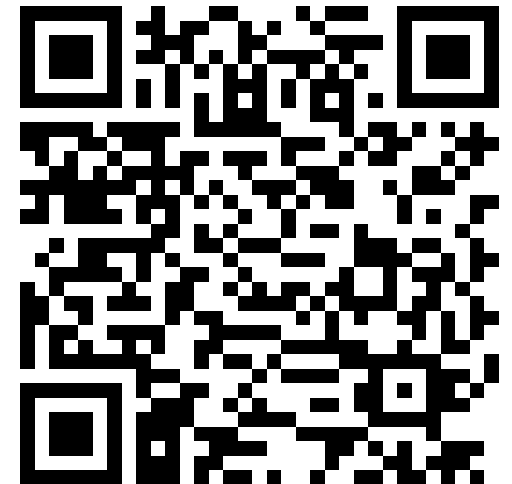
- Поднимите CS8785 до ошибки
- Проверяйте файлы добавленные генераторами
- Проверяйте `context.CancellationToken`
- Используйте `ISyntaxReceiver`
- Выдавайте диагностики
- Предоставьте необходимые атрибуты как часть генератора
- Делайте генераторы конфигурируемыми

Что в итоге?






- Генераторы делают создание шаблонного кода еще проще
- Исчезают многие типичные проблемы метапрограммирования
 - Навигация
 - Дебаг
 - Тестирование
 - Порог вхождения
- Возможности по оптимизации старта приложения
- Многие виды кодогенерации становятся более нишевыми
- Генераторы только дополняют код

Примеры генераторов

- Svg to C# SourceGenerator
 - Преобразует .svg файлы в классы возвращающие SKPicture
 -  bit.ly/32dqtBi
- JsonSrcGen
 - Сериализация в JSON без рефлексии
 -  bit.ly/2I6SKCk
- ThisAssembly
 - Константы текущей сборки – версия, название сборки, продукта
 -  bit.ly/2TSKVTr
- Win32 P/Invoke generator
 - Добавляет Win32 P/Invoke методы и необходимые типы
 -  bit.ly/2ZMYMxn



Я хочу написать генератор! С чего начать?

- Source generators cookbook – общая информация
 -  bit.ly/3mFxR0G
- Посмотреть примеры от Microsoft
 -  bit.ly/2FLih3e
- Детальный разбор нескольких примеров генераторов
 -  bit.ly/2HrZ2fp
- Примеры из презентации, как тестировать
 -  github.com/TessenR/NotifyPropertyChangedDemomo
 -  github.com/TessenR/BestPracticesSourceGeneratorsDemo
- Поиграться онлайн
 - sourcegen.dev



Спасибо за внимание



tessenr@gmail.com



github.com/TessenR



twitter.com/a_tessenr

Ссылки использованные в
докладе

