Чистый DI

Dependency Injection

Чистый DI

Контейнеры

Генератор кода

Пьяников Николай

GitHub /NikolayPianikov

Что такое DI?

stackoverflow/questions/1638919

Цель DI

Сделать код слабосвязанным

Задачи DI

Создавать композицию объектов

Управлять временем жизни

Перехватывать вызовы

Чистый DI

Чистый DI

```
var program = new Program(new Service(new Dependency()));

program.Run();

class Program
{
    private readonly Service _service;

    public Program(Service service) => _service = service;

    public void Run() => _service.DoSomething();
}

class Service
{
    private readonly Dependency _dependency;

    public Service(Dependency dependency) => _dependency = dependency;

    public void DoSomething() { }
}

class Dependency { }
```

Чистый DI

```
new Generator(
        new Logger<Generator>(),
        new ResolversFieldsBuilder(new Logger<ResolversFieldsBuilder>()),
        new ClassBuilder(new Logger<ClassBuilder>()),
        new ClassDiagramBuilder(new Logger<ClassDiagramBuilder>()),
        new MetadataValidator(new Logger<MetadataValidator>()),
        new CompositionBuilder(
                new Logger<CompositionBuilder>(),
                new []
                {
                        new UsingDeclarationsBuilder(new Logger<UsingDeclarationsBuilder>()),
                        new PrimaryConstructorBuilder(new Logger<PrimaryConstructorBuilder>()),
                        new DefaultConstructorBuilder(
                        new Logger<DefaultConstructorBuilder>(),
                        new ContractsBuilder(new RootDependencyNodeBuilder())),
                        new ChildConstructorBuilder(
                                new Logger<ChildConstructorBuilder>(),
                                new ContractsBuilder(new ConstructDependencyNodeBuilder(new Logger<ConstructDependencyNodeBuilder>()))),
                        new RootPropertiesBuilder(new Logger<RootPropertiesBuilder>()),
                        new ApiMembersBuilder(
                                new Logger<ApiMembersBuilder>(),
                                new RootDependencyNodeBuilder(new ResolversFieldsBuilder(new Logger<ResolversFieldsBuilder>()))),
                        new DisposeMethodBuilder(new Logger<DisposeMethodBuilder>()),
                        new SingletonFieldsBuilder(new Logger<SingletonFieldsBuilder>(new Marker(new Logger<Marker>()))),
                        new ArgFieldsBuilder(new Logger<ArgFieldsBuilder>()),
                        new StaticConstructorBuilder(
                                new Logger<StaticConstructorBuilder>(new RootDependencyNodeBuilder(new Logger<RootDependencyNodeBuilder>()))),
                        new UnboundTypeConstructor(), new ResolversFieldsBuilder(new Logger<ResolversFieldsBuilder>())),
                        new ResolverClassesBuilder(new Logger<ResolverClassesBuilder>()),
                        new ToStringMethodBuilder(new Logger<ToStringMethodBuilder>(), new ContractsBuilder(new RootDependencyNodeBuilder())))
                }),
        new FactoryDependencyNodeBuilder(
                new ImplementationVariantsBuilder(new Variator<Models.ImplementationVariant>(), new ArgFieldsBuilder()),
                new ImplementationDependencyNodeBuilder(
                new ArgDependencyNodeBuilder(new Logger<ArgDependencyNodeBuilder>()),
                new RootDependencyNodeBuilder(
                        new ResolversBuilder(new RootDependencyNodeBuilder(new Logger<RootDependencyNodeBuilder>())))),
                new ConstructDependencyNodeBuilder(new Logger<ConstructDependencyNodeBuilder>()),
                new Logger<FactoryDependencyNodeBuilder>(
                        new ResolversFieldsBuilder(new Logger<ResolversFieldsBuilder>()),
                        new MetadataValidator(new Logger<MetadataValidator>()),
                        new RootDependencyNodeBuilder(new ResolversFieldsBuilder(new Logger<ResolversFieldsBuilder>())))))
```

Эволюция DI контейнеров

```
Dictionary<Type, Func<object>>
```

Сложно управлять временем жизни

Сложно выполнять перехват

Зависимость от сигнатуры конструкторов

Эволюция DI контейнеров

.NET Reflection

1000 ns

Activator.CreateInstance

```
var type = Type.GetType("Sample.MyType");
var obj = Activator.CreateInstance(type);
```

Эволюция DI контейнеров

ILGenerator

10 ns

```
var cctorGen = cctor.GetILGenerator();
cctorGen.Emit(OpCodes.Ldc_I4, 1970);
cctorGen.Emit(OpCodes.Ldc_I4_1);
cctorGen.Emit(OpCodes.Ldc_I4_1);
cctorGen.Emit(OpCodes.Newobj, dateTimeCtor);
```

Эволюция DI контейнеров

Lambda Expressions

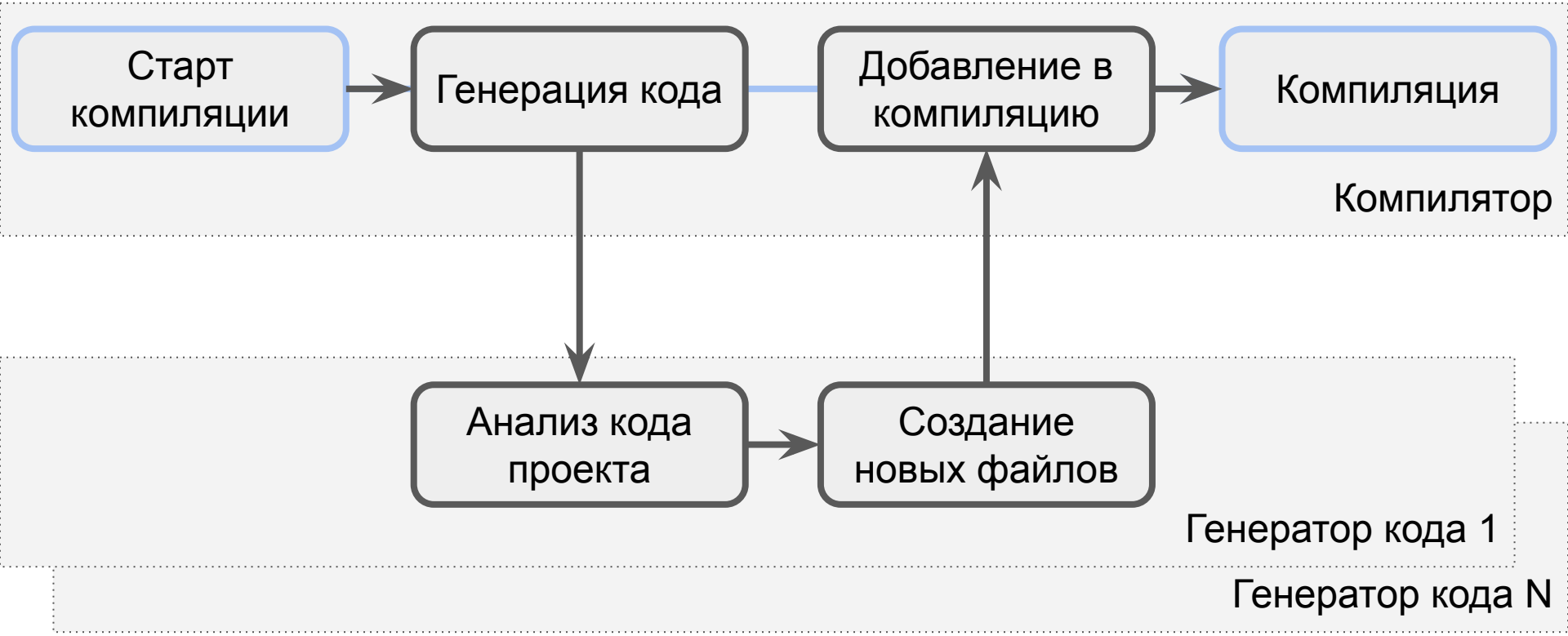
10 ns

```
var ctor = typeof(T).GetConstructors().First();
var newExpression = Expression.New(ctor);
Expression.Lambda<Func<T>>(newExpression)
    .Compile();
```

Недостатки DI контейнеров

- Ошибки на этапе выполнения
- Обращение к отражению типов .NET
- Работают не везде
- Потенциально менее эффективны

Генераторы кода



Чистый DI + Генератор кода

- Определение графа зависимостей
- Эффективное создание композиции
- Выявление проблем на этапе компиляции
- Минимум усилий на поддержку

Чистый DI + Генератор = Pure.DI

Не фреймворк и не библиотека

Генерирует частичный класс .NET

Простой и привычный API

Pure.DI

1 ns

- Не добавляет зависимостей в код
- Предсказуемая и проверенная композиция
- Высокая производительность
- Работает везде

Pure.DI

- Прост в использовании
- Тонкая настройка обобщенных типов
- Поддержка BCL из коробки
- Для библиотек и фреймворков

Кот/д

```
15
16  public interface ICat { State State { get; } }
17
18  public enum State { Alive, Dead }
19
20  public class ShroedingersCat : ICat
21  {
22      private readonly Lazy<State> _superposition;
23
24      public ShroedingersCat(Lazy<State> superposition)
25          => _superposition = superposition;
26
27      public State State => _superposition.Value;
28  }
29
```

Коробка

```
30  public interface IBox<out T> { T Content { get; } }
31
32  public class CardboardBox<T> : IBox<T>
33  {
34      public CardboardBox(T content) => Content = content;
35
36      public T Content { get; }
37  }
38
39  public partial class Program
40  {
41      private readonly IBox<ICat> _box;
42
43      internal Program(IBox<ICat> box) => _box = box;
44
45      private void Run() => Console.WriteLine(_box);
46  }
```

API Pure.DI

```
1  using Pure.DI;
2
3  new Composition().Root.Run();
4
5  DI.Setup("Composition")
6      .Bind<Random>().As(Lifetime.Singleton).To<Random>()
7      .Bind<State>().To(ctx =>
8      {
9          ctx.Inject<Random>(out var random);
10         return (State)random.Next(2);
11     })
12     .Bind<ICat>().To<ShroedingersCat>()
13     .Bind<IBox<TT>>().To<CardboardBox<TT>>()
14     .Root<Program>("Root");
```

Привязки

```
1  using Pure.DI;
2
3  new Composition().Root.Run();
4
5  DI.Setup("Composition")
6      .Bind<Random>().As(Lifetime.Singleton).To<Random>()
7      .Bind<State>().To(ctx =>
8      {
9          ctx.Inject<Random>(out var random);
10         return (State)random.Next(2);
11     })
12     .Bind<ICat>().To<ShroedingersCat>()
13     .Bind<IBox<TT>>().To<CardboardBox<TT>>()
14     .Root<Program>("Root");
```

Обобщенные типы

```
1  using Pure.DI;
2
3  new Composition().Root.Run();
4
5  DI.Setup("Composition")
6      .Bind<Random>().As(Lifetime.Singleton).To<Random>()
7      .Bind<State>().To(ctx =>
8      {
9          ctx.Inject<Random>(out var random);
10         return (State)random.Next(2);
11     })
12     .Bind<ICat>().To<ShroedingersCat>()
13     .Bind<IBox<TT>>().To<CardboardBox<TT>>()
14     .Root<Program>("Root");
```

Маркерные типы

```
Bind(typeof(IMap<,>)).To(typeof(Map<,>))

Map<TV, TK>: IMap<TKey, TValue>

Bind<IMap<TT1, TT2>>().To<Map<TT2, TT1>>()

[GenericTypeArgument]
abstract class TT1 { }
```

Ручное внедрение

```
1  using Pure.DI;
2
3  new Composition().Root.Run();
4
5  DI.Setup("Composition")
6      .Bind<Random>().As(Lifetime.Singleton).To<Random>()
7      .Bind<State>().To(ctx =>
8      {
9          ctx.Inject<Random>(out var random);
10         return (State)random.Next(2);
11     })
12     .Bind<ICat>().To<ShroedingersCat>()
13     .Bind<IBox<TT>>().To<CardboardBox<TT>>()
14     .Root<Program>("Root");
```

Корень композиции

```
1  using Pure.DI;
2
3  new Composition().Root.Run();
4
5  DI.Setup("Composition")
6      .Bind<Random>().As(Lifetime.Singleton).To<Random>()
7      .Bind<State>().To(ctx =>
8      {
9          ctx.Inject<Random>(out var random);
10         return (State)random.Next(2);
11     })
12     .Bind<ICat>().To<ShroedingersCat>()
13     .Bind<IBox<TT>>().To<CardboardBox<TT>>()
14     .Root<Program>("Root");
```

Частичный класс

```
1  using Pure.DI;
2
3  new Composition().Root.Run();
4
5  DI.Setup("Composition")
6      .Bind<Random>().As(Lifetime.Singleton).To<Random>()
7      .Bind<State>().To(ctx =>
8      {
9          ctx.Inject<Random>(out var random);
10         return (State)random.Next(2);
11     })
12     .Bind<ICat>().To<ShroedingersCat>()
13     .Bind<IBox<TT>>().To<CardboardBox<TT>>()
14     .Root<Program>("Root");
```

Частичный класс

```
partial class Composition
{
    private object _lockObject = new object();
    private Random _randomSingleton;

    public Program Root
    {
      get
      {
        Func<State> stateFunc = new Func<State>(() =>
        {
          if (_randomSingleton == null)
            lock (_lockObject)
              if (_randomSingleton == null)
                _randomSingleton = new Random();

          return (State)_randomSingleton.Next(2);
        });

        return new Program(
          new CardboardBox<ICat>(
            new ShroedingersCat(
              new Lazy<Sample.State>(
                stateFunc))));
      }
    }

    public T Resolve<T>() { ... }

    public object Resolve(Type type) { ... }
}
```

Композиция

```
1  using Pure.DI;
2
3  new Composition().Root.Run();
4
5  DI.Setup("Composition")
6      .Bind<Random>().As(Lifetime.Singleton).To<Random>()
7      .Bind<State>().To(ctx =>
8      {
9          ctx.Inject<Random>(out var random);
10         return (State)random.Next(2);
11     })
12     .Bind<ICat>().To<ShroedingersCat>()
13     .Bind<IBox<TT>>().To<CardboardBox<TT>>()
14     .Root<Program>("Root");
```

T Resolve<T>()

```
using Pure.DI;

new Composition().Resolve<Program>().Run();

DI.Setup("Composition")
    .Bind<Random>().As(Lifetime.Singleton).To<Random>()
    .Bind<State>().To(ctx =>
    {
        ctx.Inject<Random>(out var random);
        return (State)random.Next(2);
    })
    .Bind<ICat>().To<ShroedingersCat>()
    .Bind<IBox<TT>>().To<CardboardBox<TT>>()
    .Root<Program>("Root");
```

T Resolve<T>()

- Неограниченный набор зависимостей
- Может приводить к исключениям
- Снижение производительности

T Resolve<T>()

```
1  partial class Composition
2  {
3      public T Resolve<T>() => Resolver<T>.Value.Resolve(this);
4
5      static Composition()
6      {
7          Resolver<Program>.Value = new ProgramResolver();
8      }
9
10     private class Resolver<T>: IResolver<Composition, T>
11     {
12         public static IResolver<Composition, T> Value;
13     }
14
15     private class ProgramResolver: IResolver<Composition, Program>
16     {
17         public Program Resolve(Composition composition) => composition.Root;
18     }
19 }
```

object Resolve(Type type)

```
public object Resolve(Type type)
{
    var hash = Runtime.CompilerServices.RuntimeHelpers.GetHashCode(type);
    var index = (int)(_bucketSize * ((uint) hash % 4));
    var finish = index + _bucketSize;
    do {
        ref var pair = ref _buckets[index];
        if (ReferenceEquals(pair.Key, type))
        {
            return pair.Value.Resolve(this);
        }
    } while (++index < finish);

    throw new InvalidOperationException(
        $"Cannot resolve composition root of type {type}.");
}
```

Жизненный цикл

```
1  using Pure.DI;
2
3  new Composition().Root.Run();
4
5  DI.Setup("Composition")
6      .Bind<Random>().As(Lifetime.Singleton).To<Random>()
7      .Bind<State>().To(ctx =>
8      {
9          ctx.Inject<Random>(out var random);
10         return (State)random.Next(2);
11     })
12     .Bind<ICat>().To<ShroedingersCat>()
13     .Bind<IBox<TT>>().To<CardboardBox<TT>>()
14     .Root<Program>("Root");
```

Singleton

```
public IService Root
{
    get
    {
        if (object.ReferenceEquals(_singleton, null))
        {
            lock (_lockObject)
            {
                if (object.ReferenceEquals(_singleton, null))
                {
                    _singleton = new Dependency();
                }
            }
        }

        return new Service(_singleton, _singleton);
    }
}
```

- Дополнительная логика создания
- Контроль потокобезопасности
- Контроль потокобезопасности зависимостей
- Логика контроля потокобезопасности
- Может сохранять ссылки на зависимости дольше их ожидаемого времени жизни
- Логика по утилизации

Per Resolve

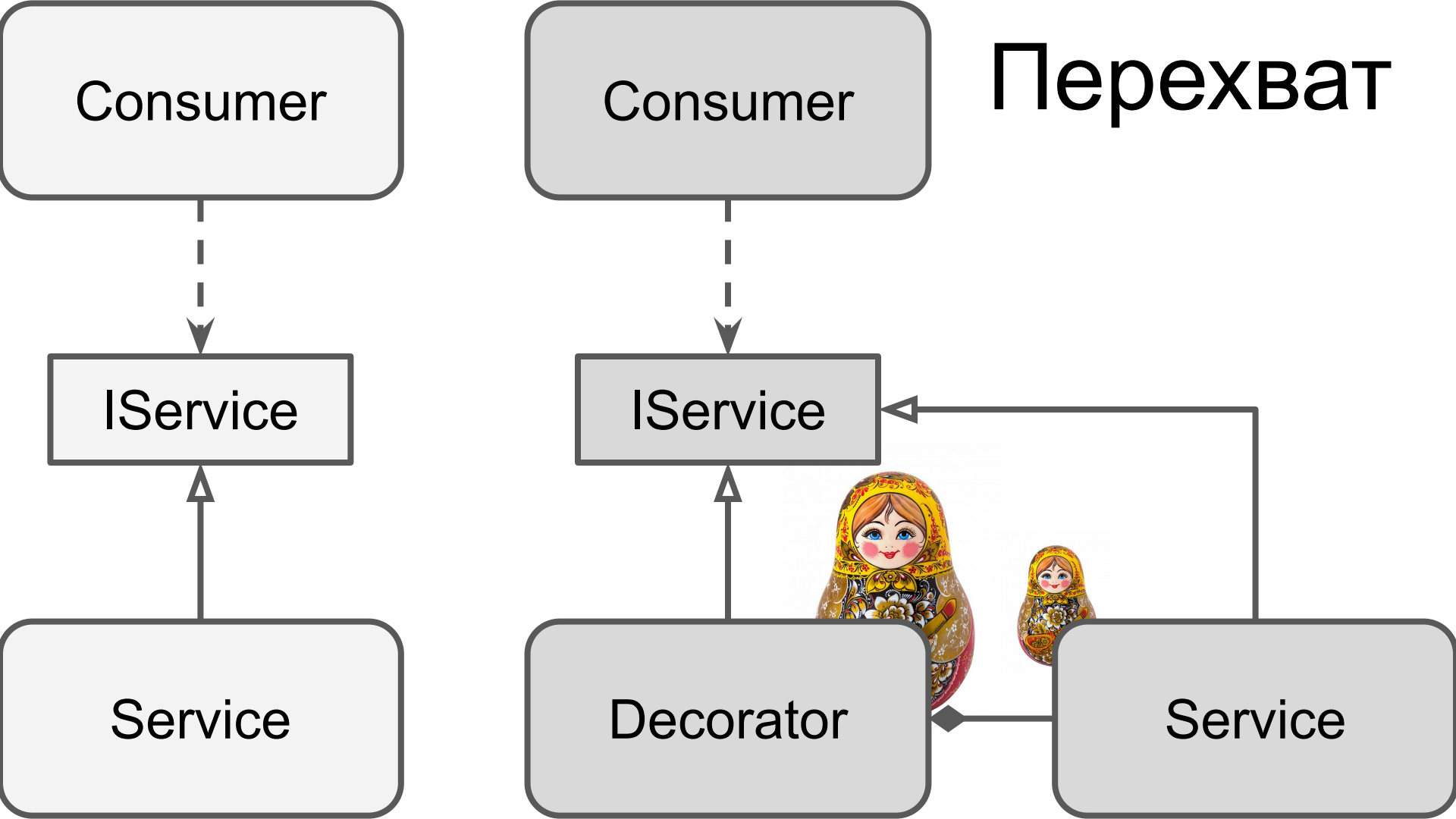
```
public IService Root
{
    get
    {
        Dependency perResolve = new Dependency();
        return new Service(perResolve, perResolve);
    }
}
```

Transient

```
public IService Root
{
    get
    {
        return new Service(new Dependency(), new Dependency());
    }
}
```

- Лишние расходы памяти
- Каждый созданный объект должен быть утилизирован
- Плохо спроектированные конструкторы могу работать медленно

Перехват


Consumer
IService
Service
Consumer
IService
Decorator
Service

Перехват

- Логирование
- Регистрация действий
- Мониторинг производительности
- Обеспечение безопасности
- Кэширование
- Обработка ошибок
- Обеспечение устойчивости к сбоям

Перехват

```
var service = new Composition().Root;
service.GetMessage().ShouldBe("Hello World !!!");

DI.Setup("Composition")
    .Bind<IService>("base").To<Service>()
    .Bind<IService>().To<GreetingService>().Root<IService>("Root");

internal interface IService { string GetMessage(); }

internal class Service : IService
{
    public string GetMessage() => "Hello World";
}

internal class GreetingService : IService
{
    private readonly IService _baseService;

    public GreetingService([Tag("base")] IService baseService) =>
        _baseService = baseService;

    public string GetMessage() => $"{_baseService.GetMessage()} !!!";
}
```

Теги

```
var service = new Composition().Root;
service.GetMessage().ShouldBe("Hello World !!!");

DI.Setup("Composition")
    .Bind<IService>("base").To<Service>()
    .Bind<IService>().To<GreetingService>().Root<IService>("Root");

internal interface IService { string GetMessage(); }

internal class Service : IService
{
    public string GetMessage() => "Hello World";
}

internal class GreetingService : IService
{
    private readonly IService _baseService;

    public GreetingService([Tag("base")] IService baseService) =>
        _baseService = baseService;

    public string GetMessage() => $"{_baseService.GetMessage()} !!!";
}
```

Hints - OnDependencyInjection

```
1   var service = new Composition().Root;
2   service.GetMessage().ShouldBe("Hello World !!!");
3
4   // OnDependencyInjection = On
5   // OnDependencyInjectionContractTypeNameRegularExpression = IService
6   DI.Setup("Composition")
7       .Bind<IService>().To<Service>().Root<IService>("Root");
8
        Nikolay Pyanikov
9   public interface IService { string GetMessage(); }
10
        Nikolay Pyanikov
11  internal class Service : IService
12  {
            Nikolay Pyanikov
13      public string GetMessage() => "Hello World";
14  }
15
```

Hints - OnDependencyInjection

```
16  internal partial class Composition: IInterceptor
17  {
18      private static readonly ProxyGenerator ProxyGenerator = new();
19
            new *
20      private partial T OnDependencyInjection<T>(in T value, object? tag, Lifetime lifetime) =>
21          (T)ProxyGenerator.CreateInterfaceProxyWithTargetInterface(typeof(T), value, this);
22
            Nikolay Pyanikov
23      public void Intercept(IInvocation invocation)
24      {
25          invocation.Proceed();
26          if (invocation.Method.Name == nameof(IService.GetMessage)
27              && invocation.ReturnValue is string message)
28          {
29              invocation.ReturnValue = $"{message} !!!";
30          }
31      }
32  }
```

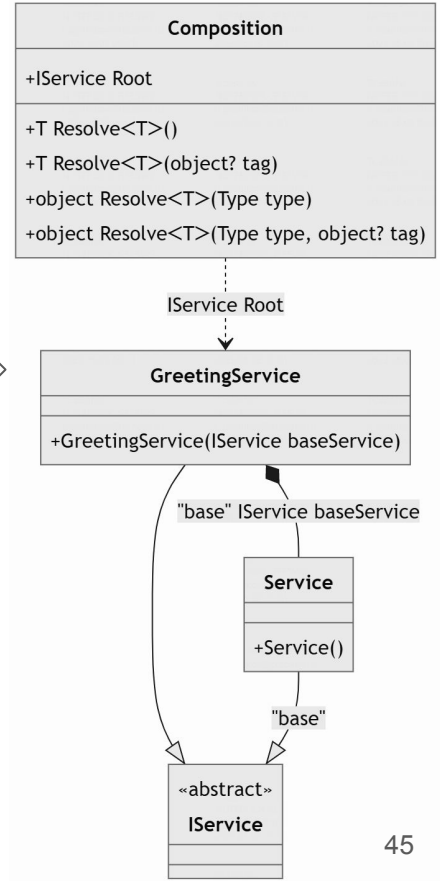
Hints - ToString

```
var service = new Composition().Root;
service.GetMessage().ShouldBe("Hello World !!!");
// ToString = On
DI.Setup("Composition")
    .Bind<IService>("base").To<Service>()
    .Bind<IService>().To<GreetingService>().Root<IService>("Root");

internal interface IService { string GetMessage(); }

internal class Service : IService
{
    public string GetMessage() => "Hello World";
}

internal class GreetingService : IService
{
    private readonly IService _baseService;

    public GreetingService([Tag("base")] IService baseService) =>
        _baseService = baseService;

    public string GetMessage() => $"{_baseService.GetMessage()} !!!";
}
```

Hints - ToString

```
public override string ToString()
{
    return
        "classDiagram\n" +
        "  class Composition {\n" +
        "    +IService Root\n" +
        "    + T Resolve<T>()\n" +
        "    + T Resolve<T>(object? tag)\n" +
        "    + object Resolve(Type type)\n" +
        "    + object Resolve(Type type, object? tag)\n" +
        "  }\n" +
        "  GreetingService --|> IService : \n" +
        "  class GreetingService {\n" +
        "    +GreetingService(IService baseService)\n" +
        "  }\n" +
        "  Service --|> IService : \"base\" \n" +
        "  class Service {\n" +
        "    +Service()\n" +
        "  }\n" +
        "  class IService {\n" +
        "    <<abstract>>\n" +
        "  }\n" +
        "  GreetingService *--  Service : \"base\"  IService baseService\n" +
        "  Composition ..> GreetingService : IService Root";
}
```

mermaid.live



Hints - ThreadSafe

```
using Pure.DI;

new Composition().Root.Run();

// ThreadSafe = Off
DI.Setup("Composition")
    .Bind<Random>().As(Lifetime.Singleton).To<Random>()
    .Bind<State>().To(ctx =>
    {
        ctx.Inject<Random>(out var random);
        return (State)random.Next(2);
    })
    .Bind<ICat>().To<ShroedingersCat>()
    .Bind<IBox<TT>>().To<CardboardBox<TT>>()
    .Root<Program>("Root");
```

```
Program Root
{
    get
    {
        Func<State> stateFunc = new Func<State>(() =>
        {
            if (_randomSingleton == null)

                        _randomSingleton = new Random()

            return (State)_randomSingleton.Next(2);
        });

        return new Program(
            new CardboardBox<ICat>(
                new ShroedingersCat(
                    new Lazy<Sample.State>(
                        stateFunc))));
    }
}
```

Hints

OnDependencyInjection

ToString

ThreadSafe

OnNewInstance

Resolve

OnCannotResolve

OnNewRoot

… 30+

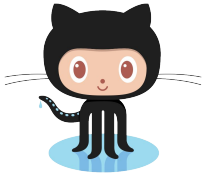
Производительность

	20T + 1S	21T
Код написанный вручную	**3.786 ns**	3.866 ns
Pure.DI Корень композиции	4.313 ns	**3.664 ns**
Pure.DI *T Resolve<T>()*	5.427 ns	5.021 ns
Pure.DI *object Resolve(Type)*	8.801 ns	8.258 ns
Dryloc 5.4.1	22.165 ns	20.525 ns
Simple Injector 5.4.1	26.368 ns	25.713 ns
Microsoft DI 7.0.0	27.638 ns	24.907 ns
Light Inject 6.6.4	40.876 ns	11.880 ns
Autofac 7.0.1	7,092.382 ns	11,836.198 ns

.NET SDK 7.0.306, Intel Core i7-10875H CPU 2.30GHz

Другая функциональность

- Arg<T>(string argName)
- RootArg<T>(string argName)
- Утилизируемые одиночки
- Дочерние композиций
- Сессии
- Span/ReadOnlySpan на стеке
- Переиспользование объектов



GitHub/DevTeam/Pure.DI