

Чистый DI

Пьяников Николай

<https://github.com/NikolayPianikov>



Цели и задачи DI

DI - это инструмент чтобы сделать код слабо связанными и легким в сопровождении

- Создавать граф объектов
- Управлять временем жизни объектов
- Перехватывать вызовы к объектам графа

Чистый DI

Чистый DI

```
1 var program = new Program(new Service(new Dependency()));
2
3 program.Run();
4
5 class Program
6 {
7     private readonly Service _service;
8
9     public Program(Service service) => _service = service;
10
11     public void Run() => _service.DoSomething();
12 }
13
14 class Service
15 {
16     private readonly Dependency _dependency;
17
18     public Service(Dependency dependency) => _dependency = dependency;
19
20     public void DoSomething() { }
21 }
22
23 class Dependency { }
```

Эволюция DI контейнеров

Dictionary<Type, Func<object>>

Эволюция DI контейнеров

1000 ns

Activator.CreateInstance

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

Эволюция DI контейнеров

10 ns

ILGenerator

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


Эволюция DI контейнеров

10 ns

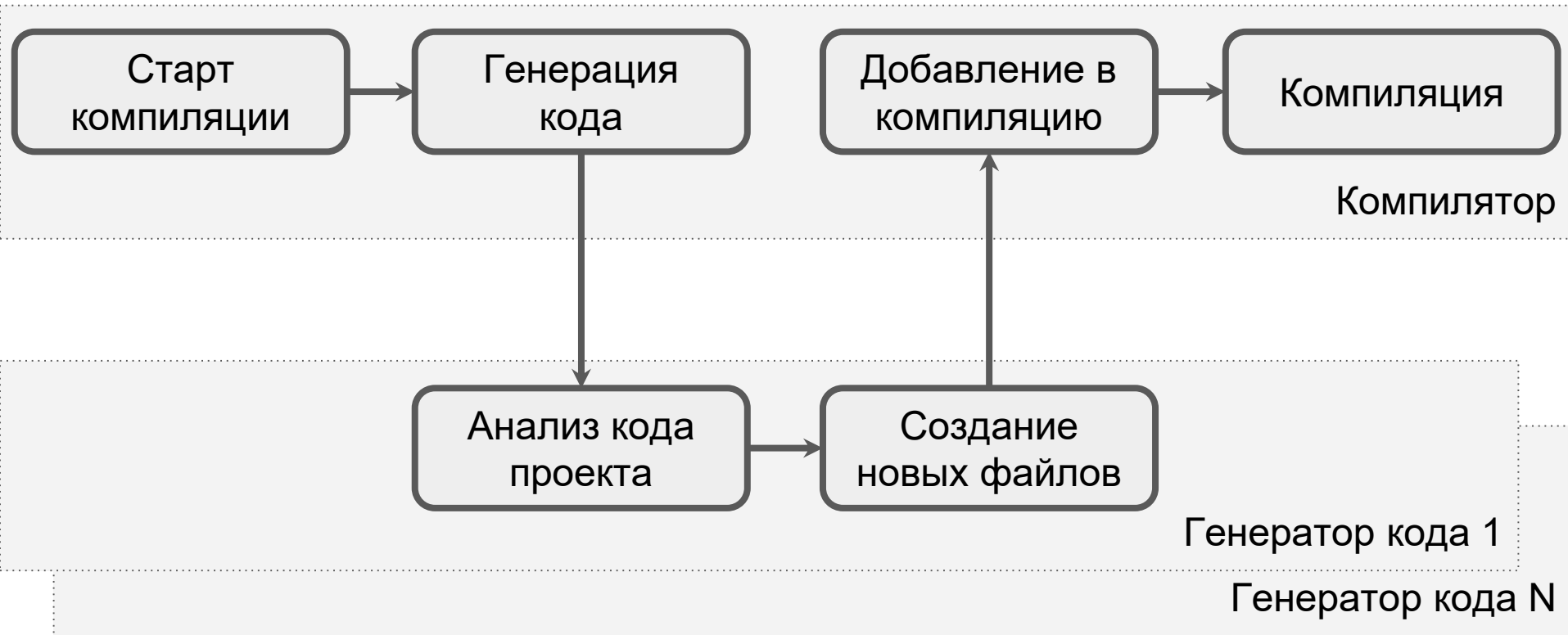
Lambda Expressions

```
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
- IL генерация не всегда возможна
- DI контейнер потенциально менее эффективен чем чистый DI

Генераторы кода



Чистый DI + Генераторы кода

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

Чистый DI + Генераторы кода = Pure.DI

- Не добавляет зависимостей в код
- Предсказуемый и проверенный граф
- Высокая производительность
- Работает везде
- Прост в использовании
- Тонкая настройка обобщенных типов
- Поддержка BCL из коробки
- Для библиотек и фреймворков

1 ns

```

15 public interface ICat { State State { get; } }
16
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 }
```

Pure.DI

```
1 using Pure.DI;  
2  
3 new Composition().Root.Run();
```

```
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>>()`

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();
```

```
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");
```

Код

```
1 partial class Composition
2 {
3     private object _lockObject = new object();
4     private Random _randomSingleton;
5
6     public Program Root
7     {
8         get
9         {
10             Func<State> stateFunc = new Func<State>(() =>
11             {
12                 if (_randomSingleton == null)
13                     lock (_lockObject)
14                     if (_randomSingleton == null)
15                         _randomSingleton = new Random();
16
17                 return (State)_randomSingleton.Next(2);
18             });
19
20             return new Program(
21                 new CardboardBox<ICat>(
22                     new ShroedingersCat(
23                         new Lazy<Sample.State>(
24                             stateFunc))));
25         }
26     }
27
28     public T Resolve<T>() { ... }
29
30     public object Resolve(Type type) { ... }
31 }
```

```
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))));
```

Получение корня

```
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");
```


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
- сложная логика для 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);
```

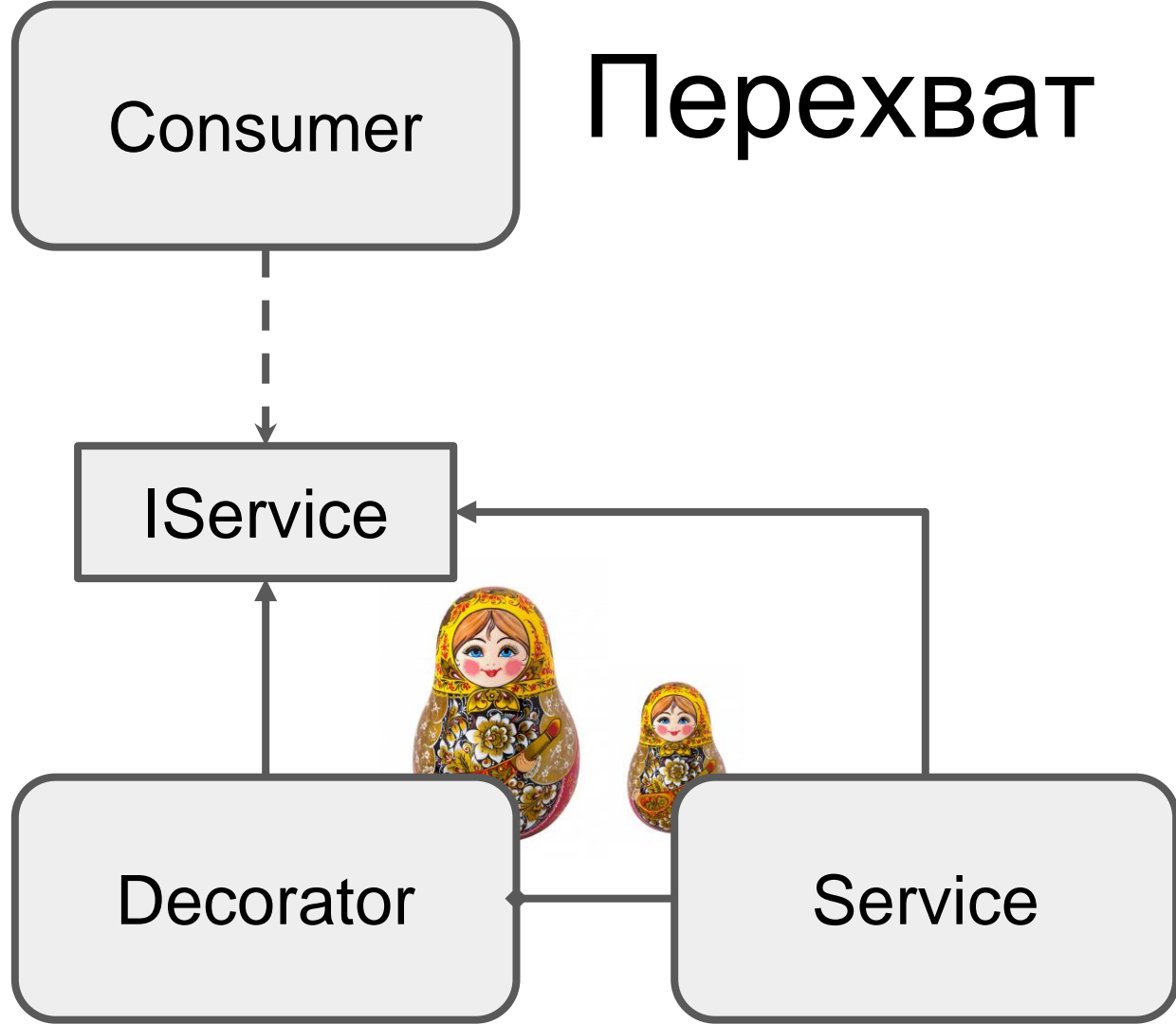
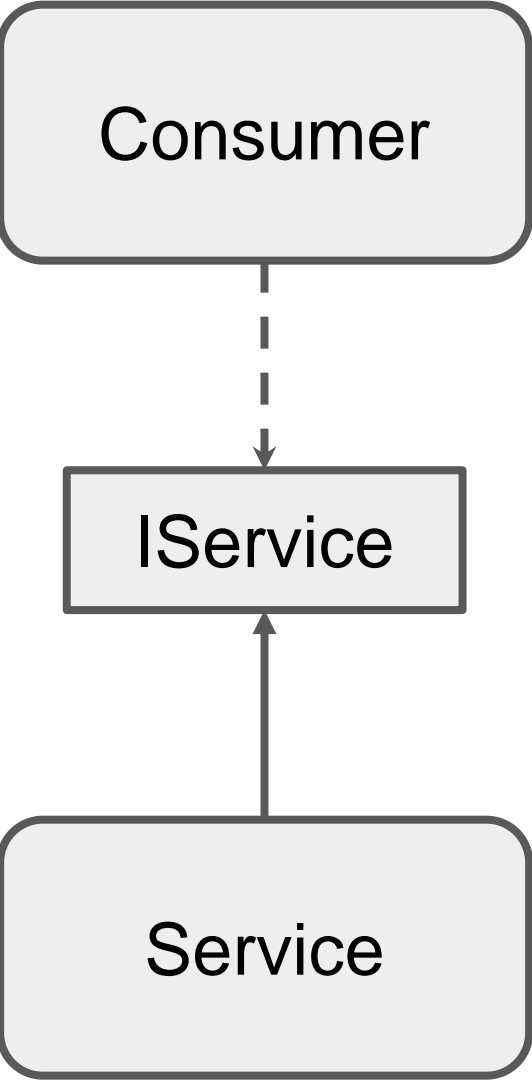
```
    }
```

```
}
```

Transient

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

- лишние расходы памяти
- каждый созданный объект должен быть утилизирован
- плохо спроектированные конструкторы могут работать медленно



Перехват

- логирование
- регистрация действий
- мониторинг производительности
- обеспечение безопасности
- кэширование
- обработка ошибок
- обеспечение устойчивости к сбоям

Перехват

```
1 var service = new Composition().Root;
2 service.GetMessage().ShouldBe("Hello World !!!");
3
4 DI.Setup("Composition")
5     .Bind<IService>("base").To<Service>()
6     .Bind<IService>().To<GreetingService>().Root<IService>("Root");
7
8 internal interface IService { string GetMessage(); }
9
10 internal class Service : IService
11 {
12     public string GetMessage() => "Hello World";
13 }
14
15 internal class GreetingService : IService
16 {
17     private readonly IService _baseService;
18
19     public GreetingService([Tag("base")] IService baseService) =>
20         _baseService = baseService;
21
22     public string GetMessage() => $"{_baseService.GetMessage()} !!!";
23 }
```



Теги

```
1 var service = new Composition().Root;
2 service.GetMessage().ShouldBe("Hello World !!!");
3
4 DI.Setup("Composition")
5     .Bind<IService>("base").To<Service>()
6     .Bind<IService>().To<GreetingService>().Root<IService>("Root");
7
8 internal interface IService { string GetMessage(); }
9
10 internal class Service : IService
11 {
12     public string GetMessage() => "Hello World";
13 }
14
15 internal class GreetingService : IService
16 {
17     private readonly IService _baseService;
18
19     public GreetingService([Tag("base")] IService baseService) =>
20         _baseService = baseService;
21
22     public string GetMessage() => $"{_baseService.GetMessage()} !!!";
23 }
```


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  
9 Nikolay Pyanikov  
10 public interface IService { string GetMessage(); }  
11  
12 Nikolay Pyanikov  
13 ✓ internal class Service : IService  
14 {  
15     Nikolay Pyanikov  
16     public string GetMessage() => "Hello World";  
17 }  
18
```

Hints - OnDependencyInjection

```
16  internal partial class Composition: IInterceptor
17  {
18      private static readonly ProxyGenerator ProxyGenerator = new();
19
20       new *
21      private partial T OnDependencyInjection<T>(in T value, object? tag, Lifetime lifetime) =>
22          (T)ProxyGenerator.CreateInterfaceProxyWithTargetInterface(typeof(T), value, this);
23
24       Nikolay Pyanikov
25      public void Intercept(IInvocation invocation)
26      {
27          invocation.Proceed();
28          if (invocation.Method.Name == nameof(IService.GetMessage)
29              && invocation.ReturnValue is string message)
30          {
31              invocation.ReturnValue = $"{message} !!!";
32          }
33      }
34  }
```

Hints

- ToString
- OnDependencyInjection
- OnNewInstance
- Resolve
- OnCannotResolve
- OnNewRoot
- ThreadSafe
- Другие

Производительность

	Mean 20+1	Ratio 20+1	Mean 21	Ratio 21
Обычный код	3.786 ns	1.00	3.866 ns	1.00
Pure.DI корень композиции	4.313 ns	1.14	3.664 ns	0.95
Pure.DI <i>Resolve<T>()</i>	5.427 ns	1.43	5.021 ns	1.32
Pure.DI <i>Resolve(Type)</i>	8.801 ns	2.34	8.258 ns	2.12
Dryloc 5.4.1	22.165 ns	5.88	20.525 ns	5.36
Simple Injector 5.4.1	26.368 ns	6.96	25.713 ns	6.68
Microsoft DI 7.0.0	27.638 ns	7.33	24.907 ns	6.50
Light Inject 6.6.4	40.876 ns	10.82	11.880 ns	3.09
Autofac 7.0.1	7,092.382 ns	1,879.76	11,836.198 ns	3,192.58

Другая функциональность

<https://github.com/DevTeam/Pure.DI>

- Arg<T>(string argName)
- RootArg<T>(string argName)
- Disposable singletons
- Дочерние композиции
- Scopes
- Аллокации Span/ReadOnlySpan на стеке
- Переиспользование BCL объектов

