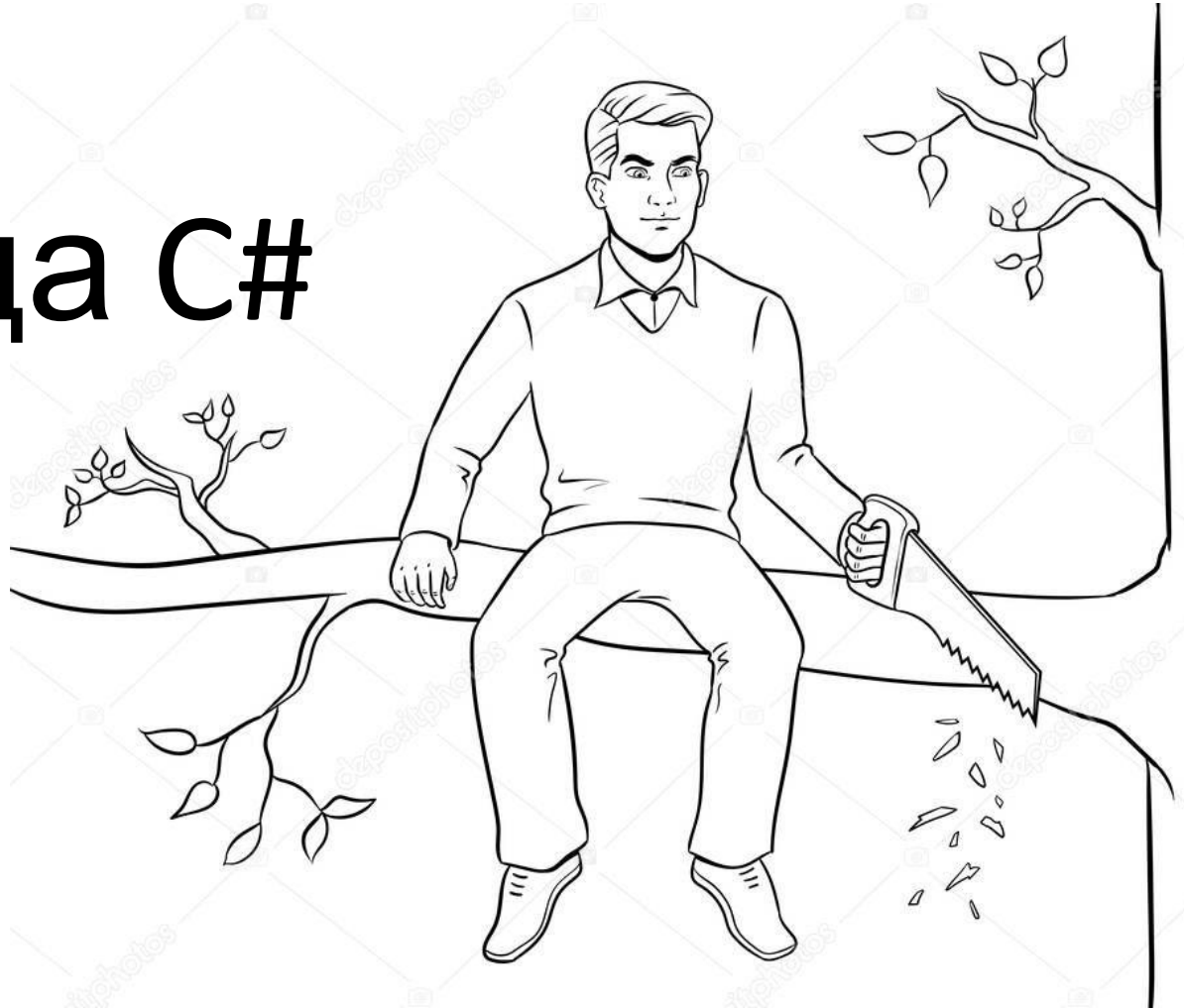
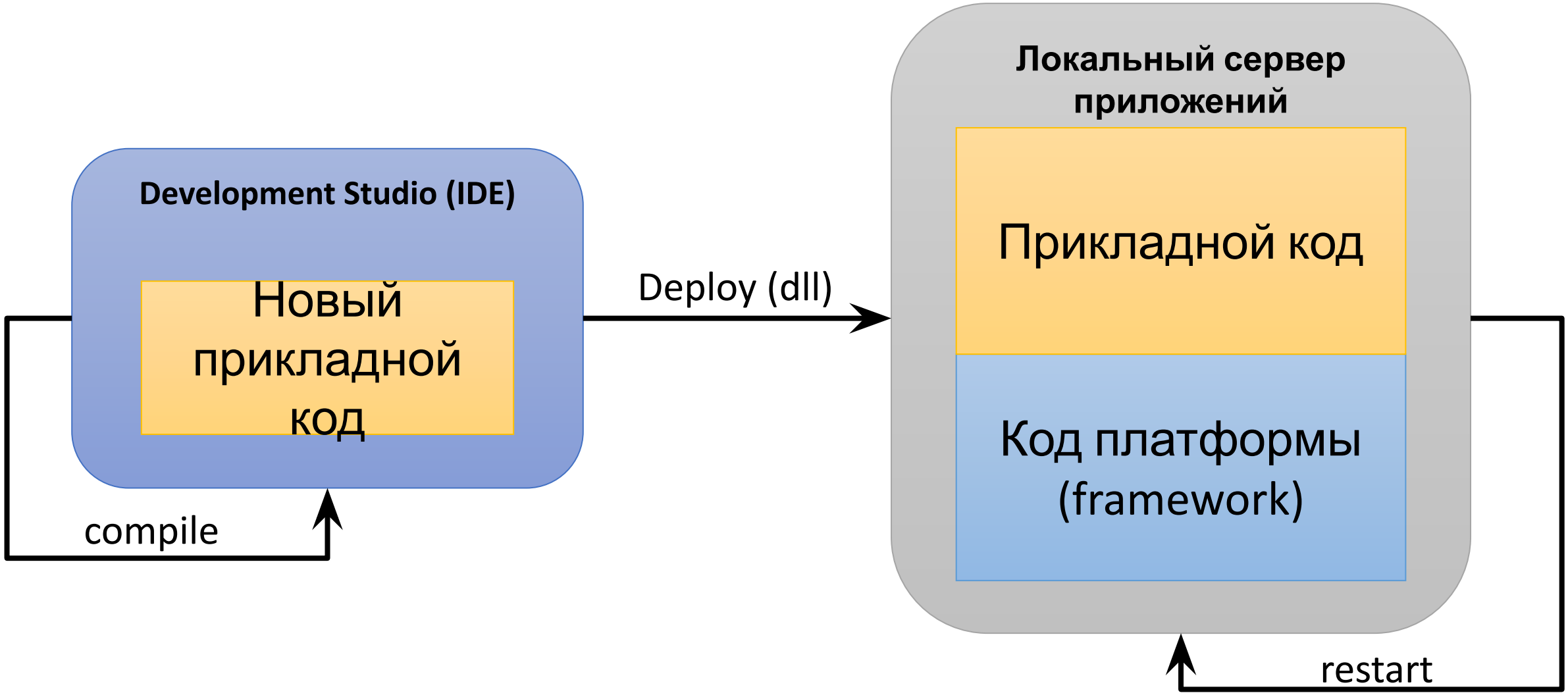
HotReload кода C#

Другим кодом C#

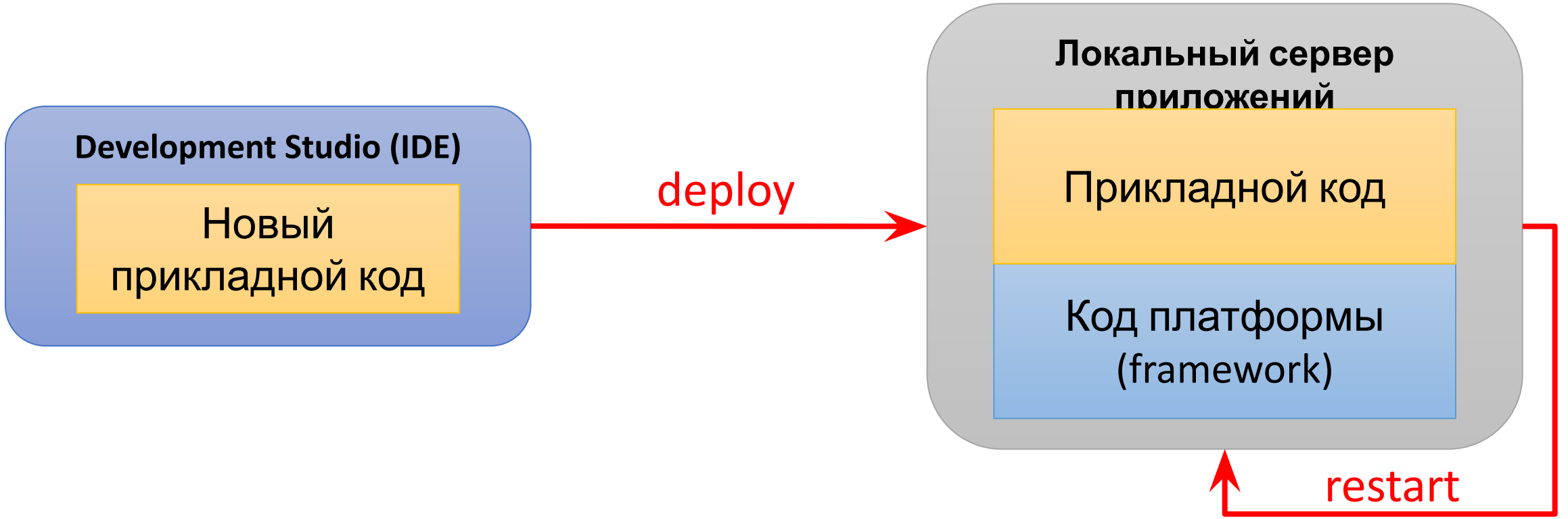


Прикладная разработка DirectumRX



Зачем нам HotReload

- Быстрая локальная отладка прикладного кода.
- Обычный deploy хорош и надёжен для прода, но медленный из-за перезапуска сервера.

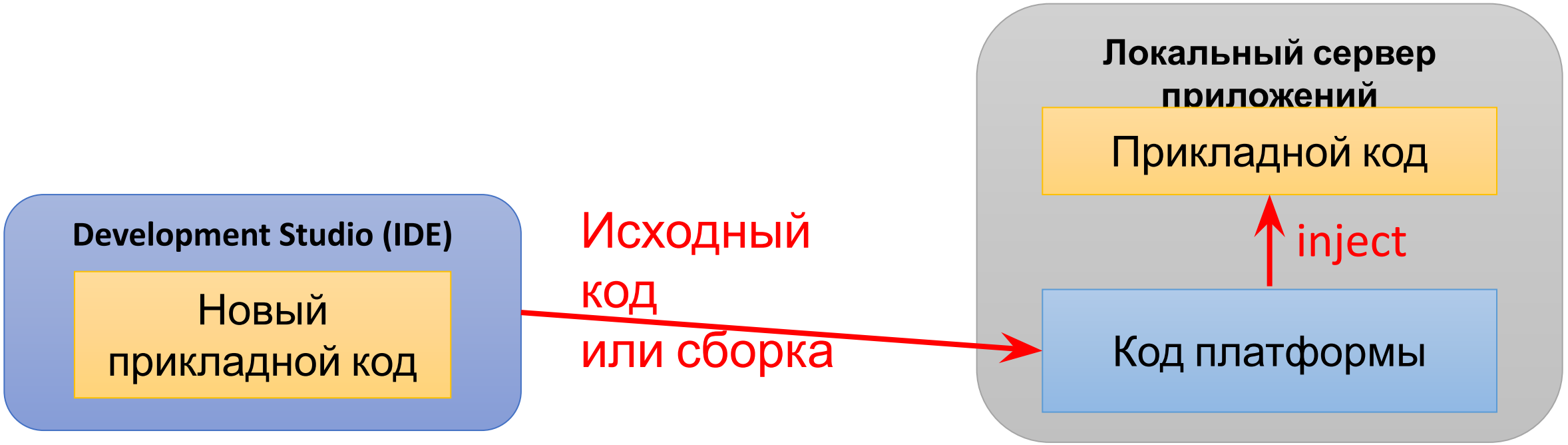


Дополнительные ограничения и требования

- Не спасёт, если поменялась структура БД.
- Надо, чтобы при Hot Reload могла работать отладка в Dev Studio.
- Хорошо бы, чтобы можно использовать не только для серверного кода, но и для клиента (который уже запущен).

Основная идея

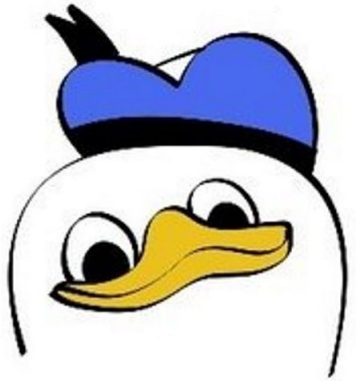
- **Избавиться от перезапуска сервера.**
- Делаем на стороне сервера точку (контроллер WebApi, например), которая встроит поданный код в работающее приложение.



Решения в лоб

- Возня с reflection – LoadAssembly (в т. ч. Shadow Copy Assemblies).
- Managed Extensibility Framework (MEF); в том числе VS-MEF.
- Mono.Cecil.

Решения в лоб. Проблемы



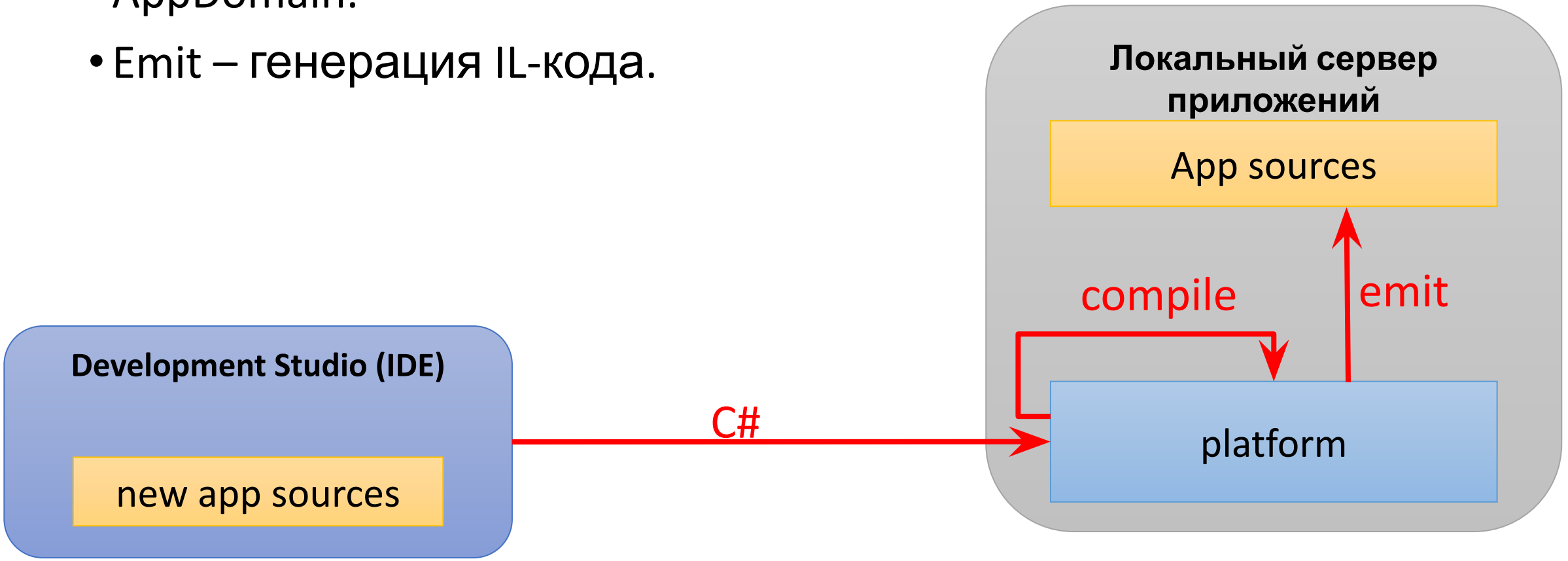
- Никак не повлияем на уже существующие в памяти объекты,
- Новые объекты, создаваемые через *new* в прикладном коде будут ссылаться на старые типы.

Чтобы это работало нужно:

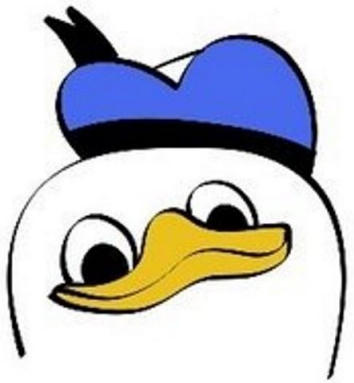
- писать свой загрузчик типов. Если где-то в прикладном коде какая-то прикладная сущность создаётся через *new*, то придётся модифицировать прикладной код (это плохо);
- в случае с MEF по коду надо либо раскидывать специальные атрибуты, либо наследоваться от нужных интерфейсов.

Компилять на сервере и Emit (v1)

- CodeDom – компиляция c# в Assembly сразу в память в AppDomain.
- Emit – генерация IL-кода.



Компилять на сервере и Emit



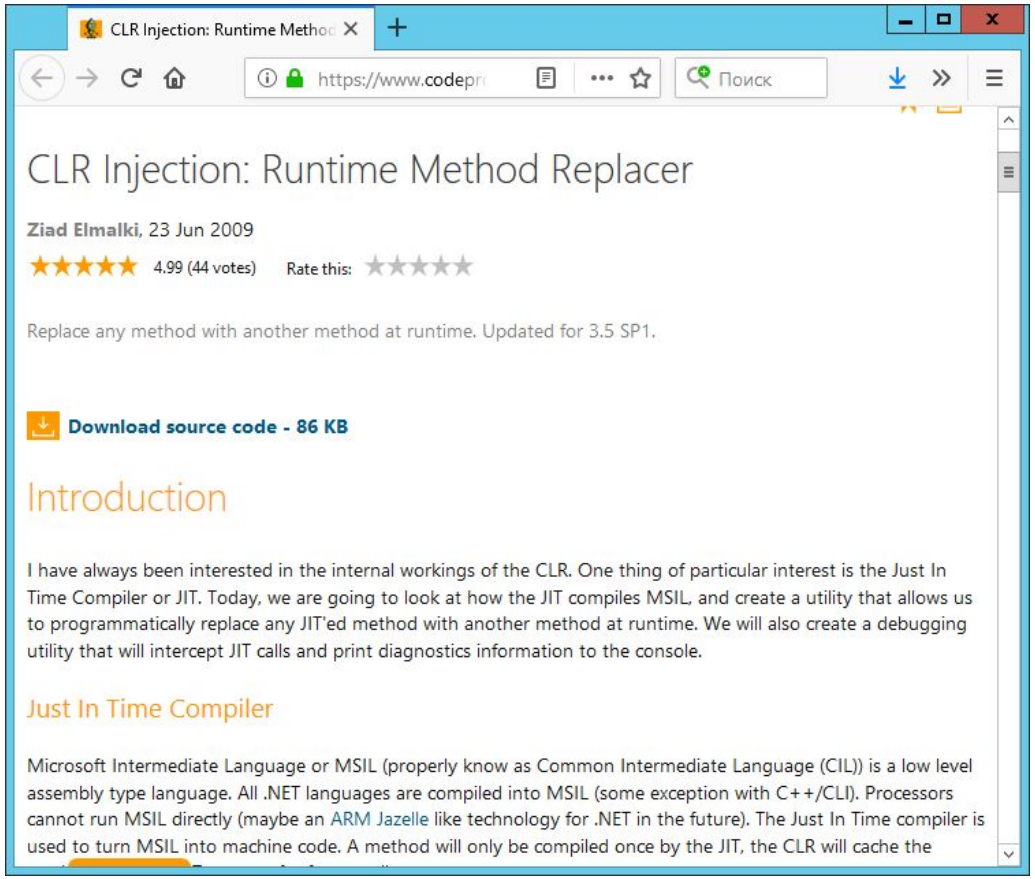
Проблемы:

- Emit работает только с DynamicAssembly (соответствввенно – DynamicMethod). А в результатет работы CodeDom и компиляции мы получаем, по сути, обычную Assembly.
- Чтобы динамический класс отправить в работу, нужно вызвать у него CreateType(). Это блокирует дальнейшие его модификации

Edit and Continue

- Встроенный в Visual Studio хитрый механизм, генерирующий некоторые дельты.
- Общедоступного API нет.
- Даже в самой VS механизм не работает в ряде случаев.

Method inject v1



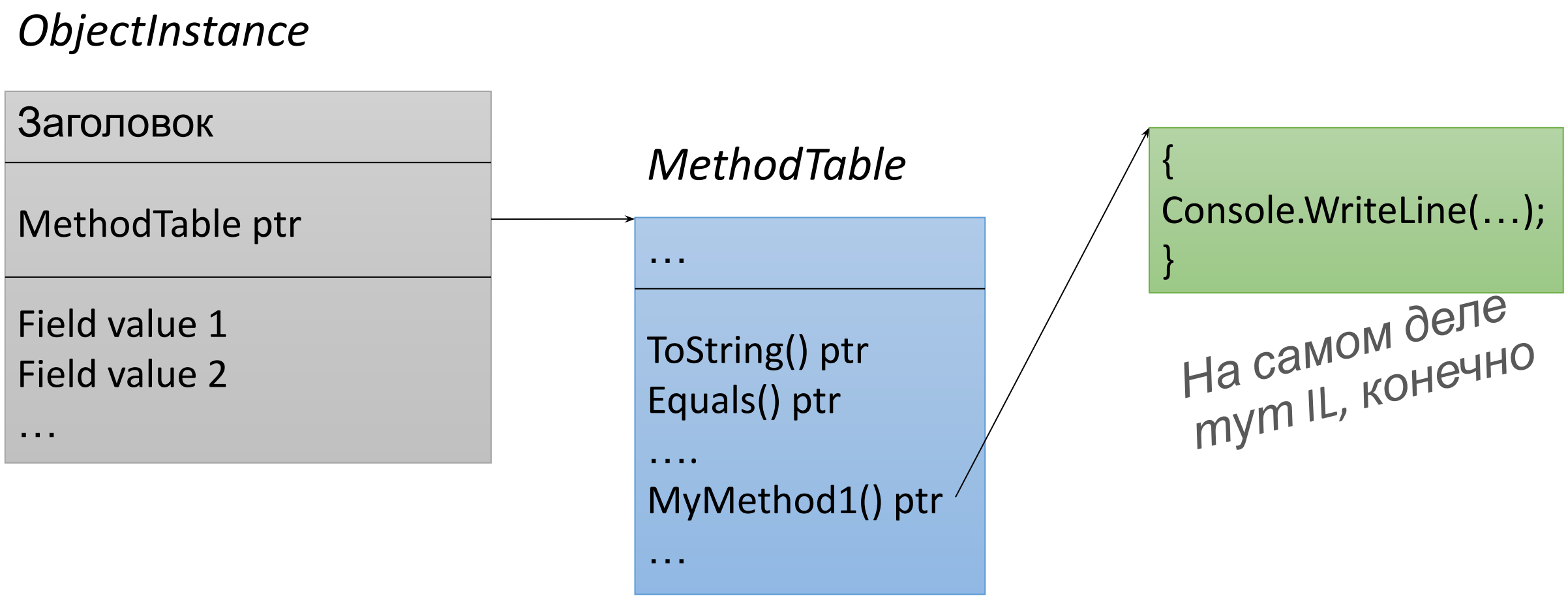
Замена указателя на метод:

```
MethodInfo methodToReplace = … ;
MethodInfo methodToInject = … ;


unsafe
{
  long* inject = (long*)methodToInject.MethodHandle.Value.ToPointer();
  long* target =
(long*)methodToReplace.MethodHandle.Value.ToPointer();
  *tar = *inj;
}
```

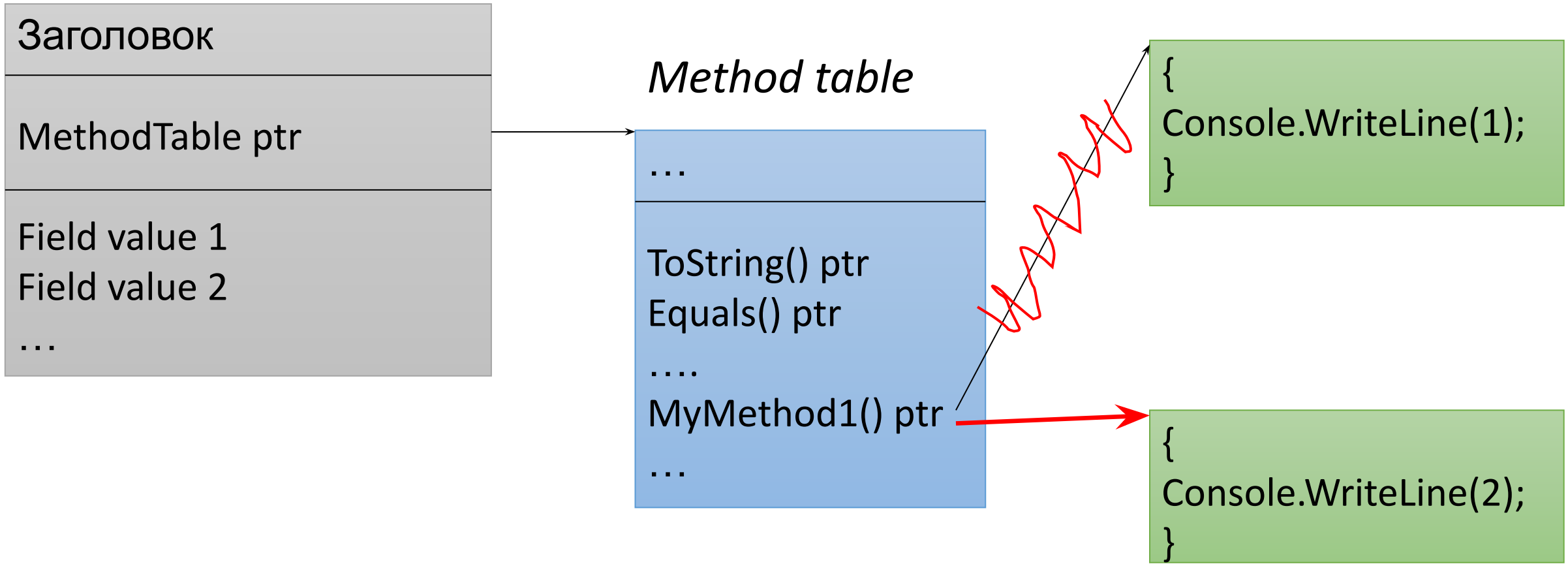
В реальности чуток сложнее, потому что надо учесть x86/x64, Debug/Release.

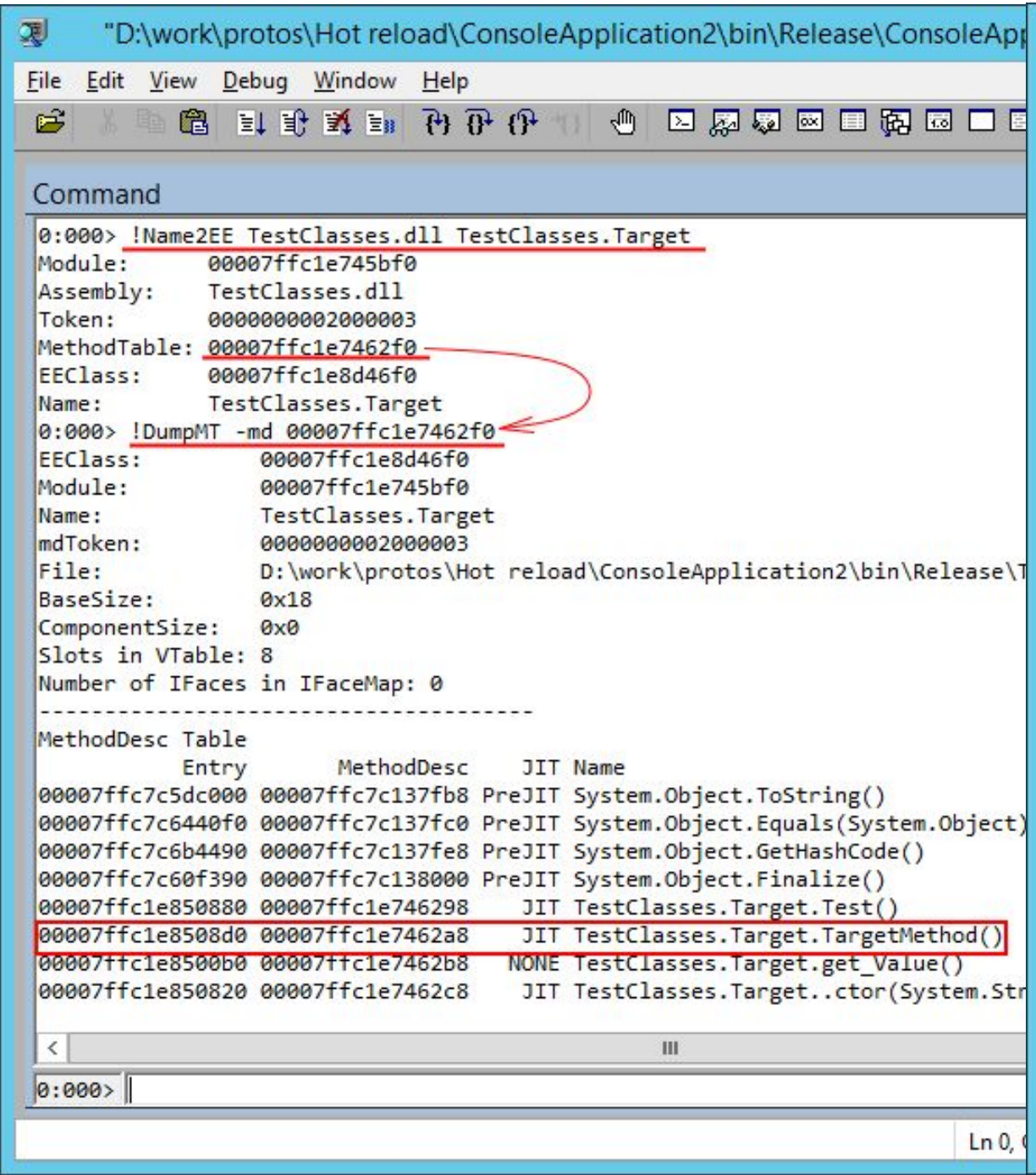
Как хранятся описания классов в .NET

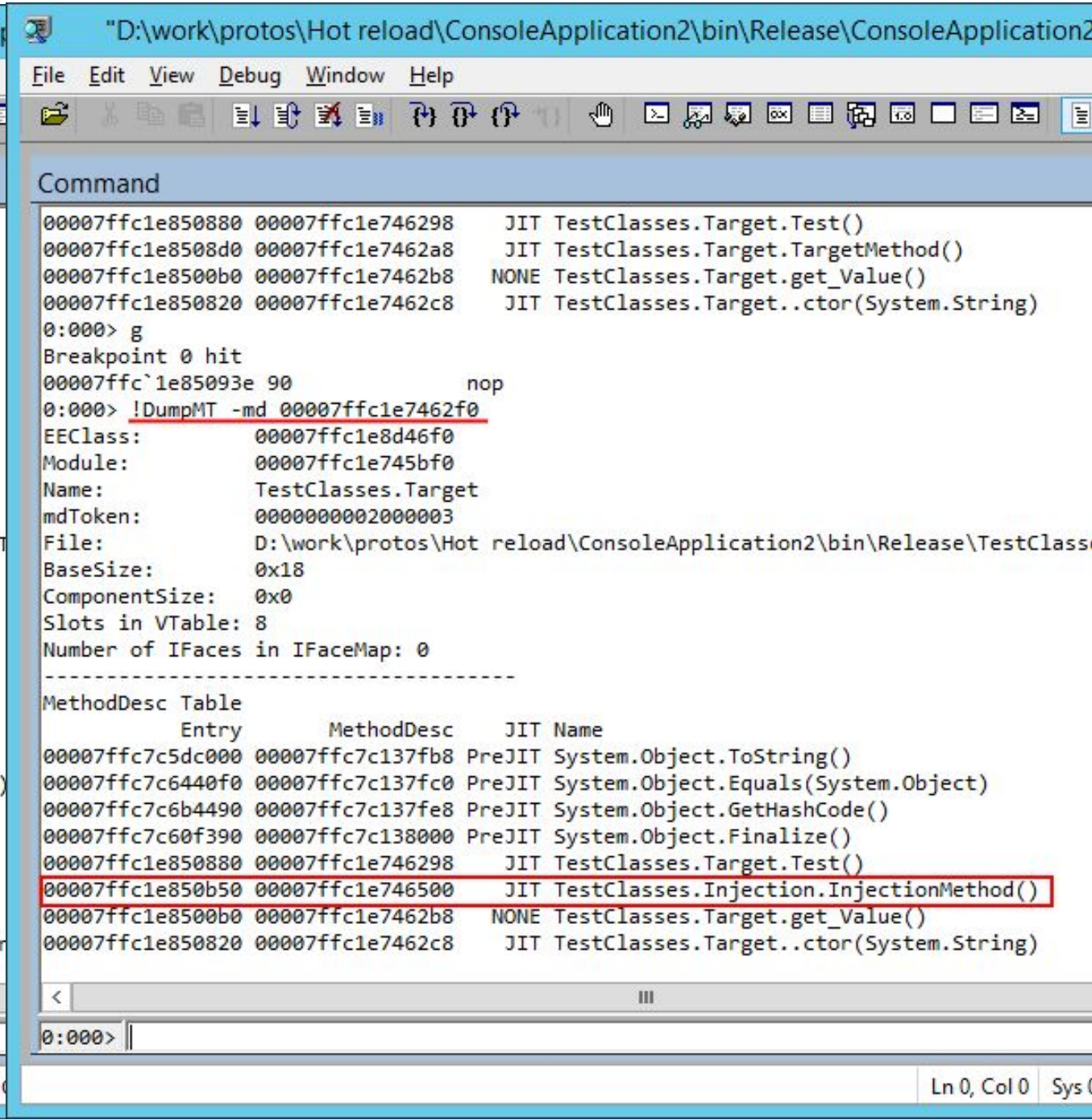


Суть Method Inject v1

Object instance



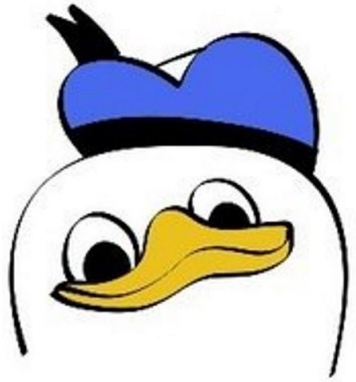

"D:\work\protos\Hot reload\ConsoleApplication2\bin\Release\ConsoleApp
File Edit View Debug Window Help
Command
0:000> !Name2EE TestClasses.dll TestClasses.Target
Module: 00007ffc1e745bf0
Assembly: TestClasses.dll
Token: 0000000002000003
MethodTable: 00007ffc1e7462f0
EEClass: 00007ffc1e8d46f0
Name: TestClasses.Target
0:000> !DumpMT -md 00007ffc1e7462f0
EEClass: 00007ffc1e8d46f0
Module: 00007ffc1e745bf0
Name: TestClasses.Target
mdToken: 0000000002000003
File: D:\work\protos\Hot reload\ConsoleApplication2\bin\Release\T
BaseSize: 0x18
ComponentSize: 0x0
Slots in VTable: 8
Number of IFaces in IFaceMap: 0
MethodDesc Table
Entry MethodDesc JIT Name
00007ffc7c5dc000 00007ffc7c137fb8 PreJIT System.Object.ToString()
00007ffc7c6440f0 00007ffc7c137fc0 PreJIT System.Object.Equals(System.Object)
00007ffc7c6b4490 00007ffc7c137fe8 PreJIT System.Object.GetHashCode()
00007ffc7c60f390 00007ffc7c138000 PreJIT System.Object.Finalize()
00007ffc1e850880 00007ffc1e746298 JIT TestClasses.Target.Test()
00007ffc1e8508d0 00007ffc1e7462a8 JIT TestClasses.Target.TargetMethod()
00007ffc1e8500b0 00007ffc1e7462b8 NONE TestClasses.Target.get_Value()
00007ffc1e850820 00007ffc1e7462c8 JIT TestClasses.Target..ctor(System.Str
0:000>
Ln 0,


"D:\work\protos\Hot reload\ConsoleApplication2\bin\Release\ConsoleApplication2
File Edit View Debug Window Help
Command
00007ffc1e850880 00007ffc1e746298 JIT TestClasses.Target.Test()
00007ffc1e8508d0 00007ffc1e7462a8 JIT TestClasses.Target.TargetMethod()
00007ffc1e8500b0 00007ffc1e7462b8 NONE TestClasses.Target.get_Value()
00007ffc1e850820 00007ffc1e7462c8 JIT TestClasses.Target..ctor(System.String)
0:000> g
Breakpoint 0 hit
00007ffc`1e85093e 90 nop
0:000> !DumpMT -md 00007ffc1e7462f0
EEClass: 00007ffc1e8d46f0
Module: 00007ffc1e745bf0
Name: TestClasses.Target
mdToken: 0000000002000003
File: D:\work\protos\Hot reload\ConsoleApplication2\bin\Release\TestClass
BaseSize: 0x18
ComponentSize: 0x0
Slots in VTable: 8
Number of IFaces in IFaceMap: 0
MethodDesc Table
Entry MethodDesc JIT Name
00007ffc7c5dc000 00007ffc7c137fb8 PreJIT System.Object.ToString()
00007ffc7c6440f0 00007ffc7c137fc0 PreJIT System.Object.Equals(System.Object)
00007ffc7c6b4490 00007ffc7c137fe8 PreJIT System.Object.GetHashCode()
00007ffc7c60f390 00007ffc7c138000 PreJIT System.Object.Finalize()
00007ffc1e850880 00007ffc1e746298 JIT TestClasses.Target.Test()
00007ffc1e850b50 00007ffc1e746500 JIT TestClasses.Injection.InjectionMethod()
00007ffc1e8500b0 00007ffc1e7462b8 NONE TestClasses.Target.get_Value()
00007ffc1e850820 00007ffc1e7462c8 JIT TestClasses.Target..ctor(System.String)
0:000>
Ln 0, Col 0
Sys

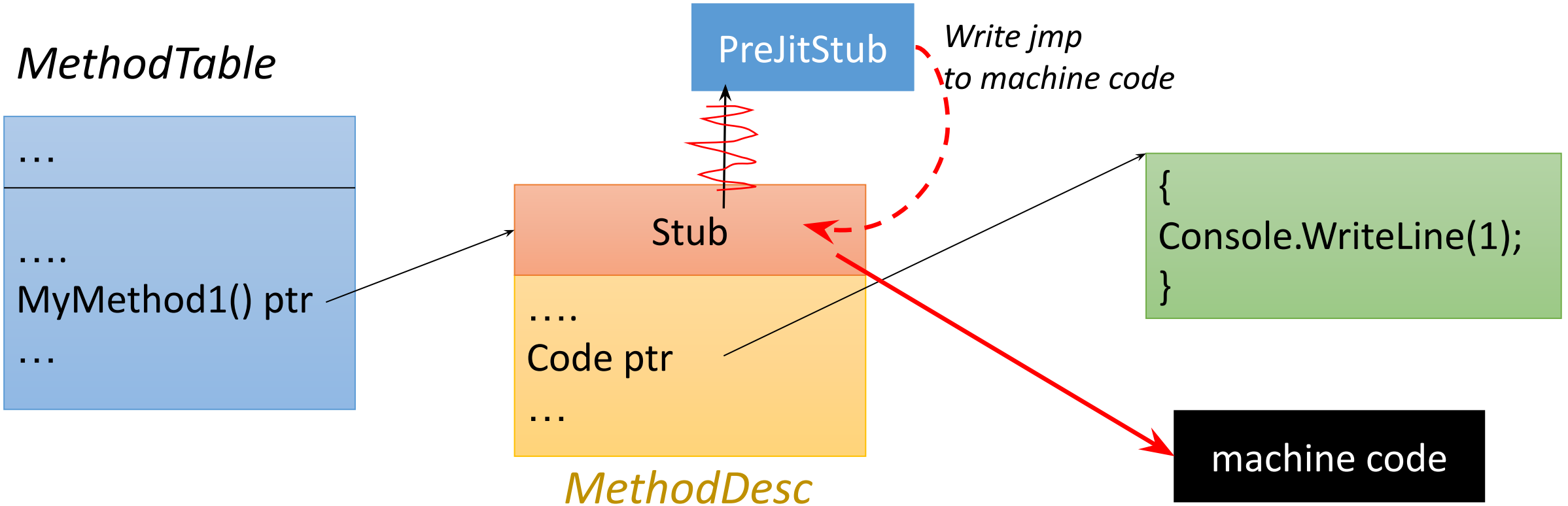
Method Inject v1. Что нам надо сделать ещё

- Сравнить прикладные исходники и найти изменившиеся методы.
- Передать исходные тексты методов (+ доп. инфа) на сервер
- Создать новый класс, засунуть в него методы и скомпилировать в память
- *(Перед компиляцией надо заменить все обращения к this на обращения через Reflection)*
- Найти старый метод и сделать MethodInject на новый

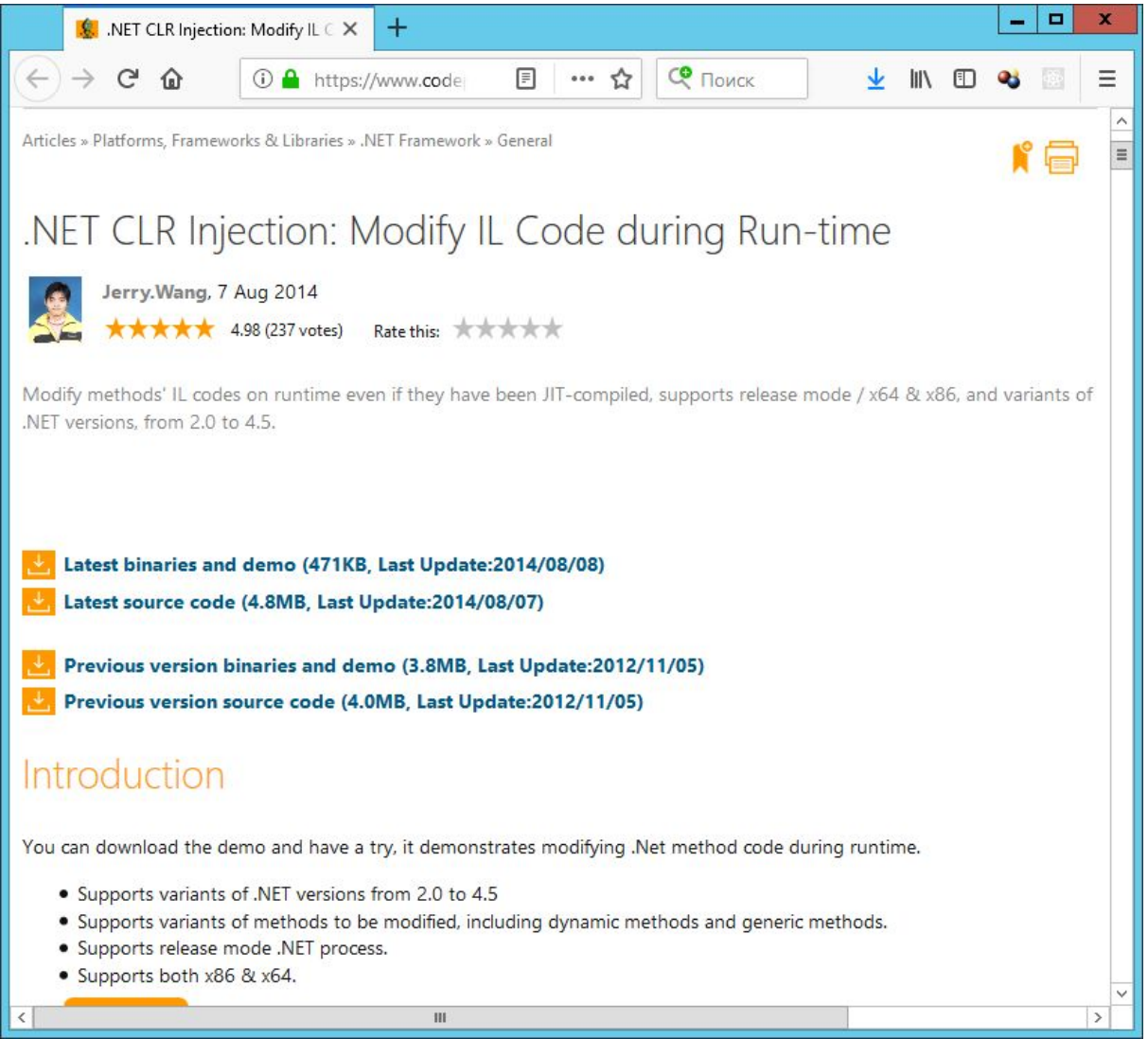
Method Inject v1. ПРОБЛЕМА!



Прекрасно работает, пока мы не вызываем метод.
После первого вызова срабатывает JIT и вся магия пропадает.



Method Inject v2



Использование EasyHook

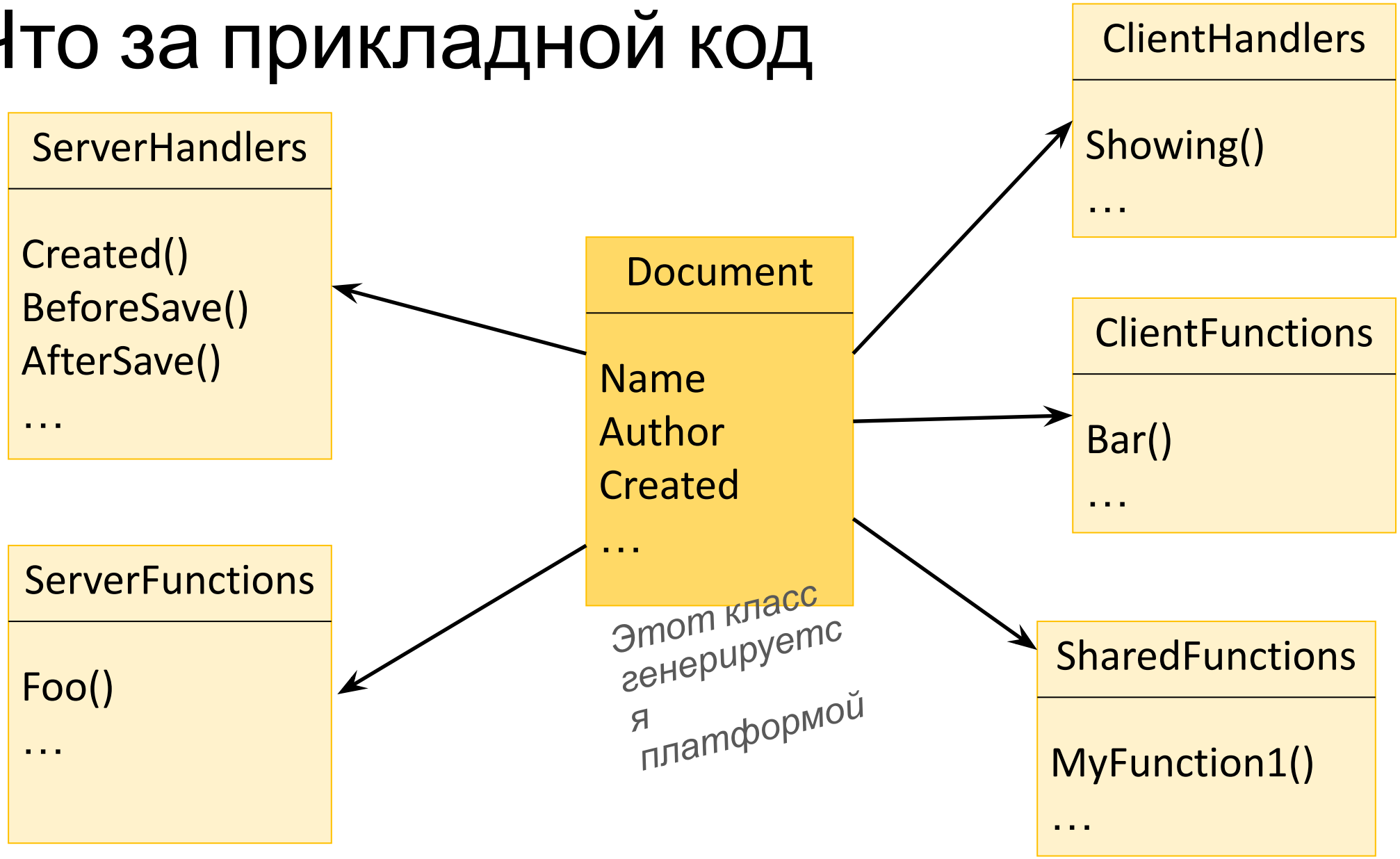
Основная идея:

- Переопределение compileMethod,
- Сброс состояния Jitted с помощью MethodDesc.Reset()

Method Inject v2. Проблемки

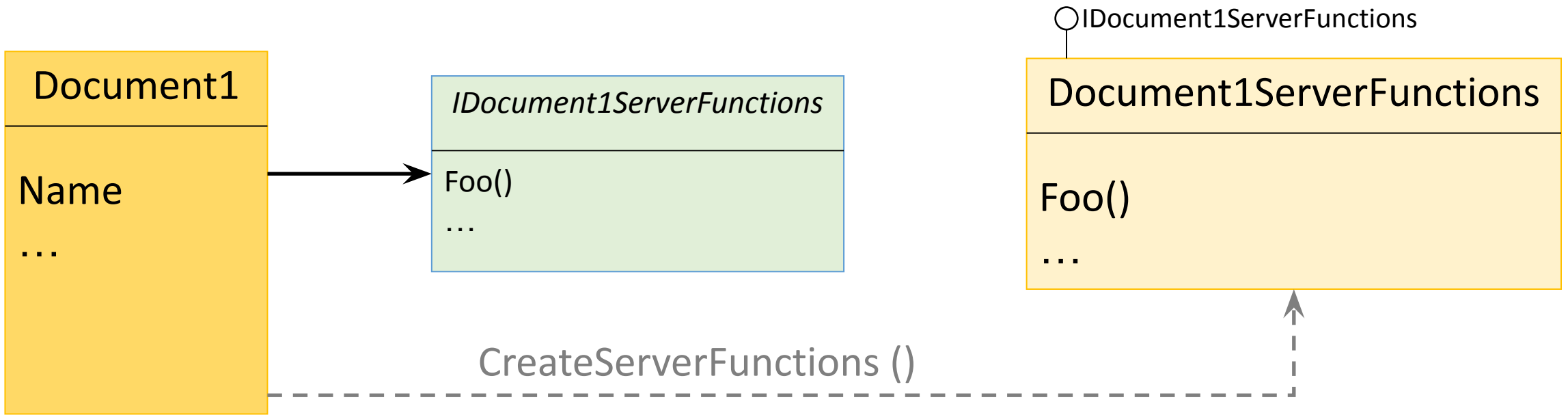
- Нормально заработало только на .NET Framework 3.5.
- Уж очень рискованно такое применять в продакшене.
- Поддерживать такое от версии к версии .NET – хз, как, потому что влезли конкретно внутрь кухни CLR

Что за прикладной код

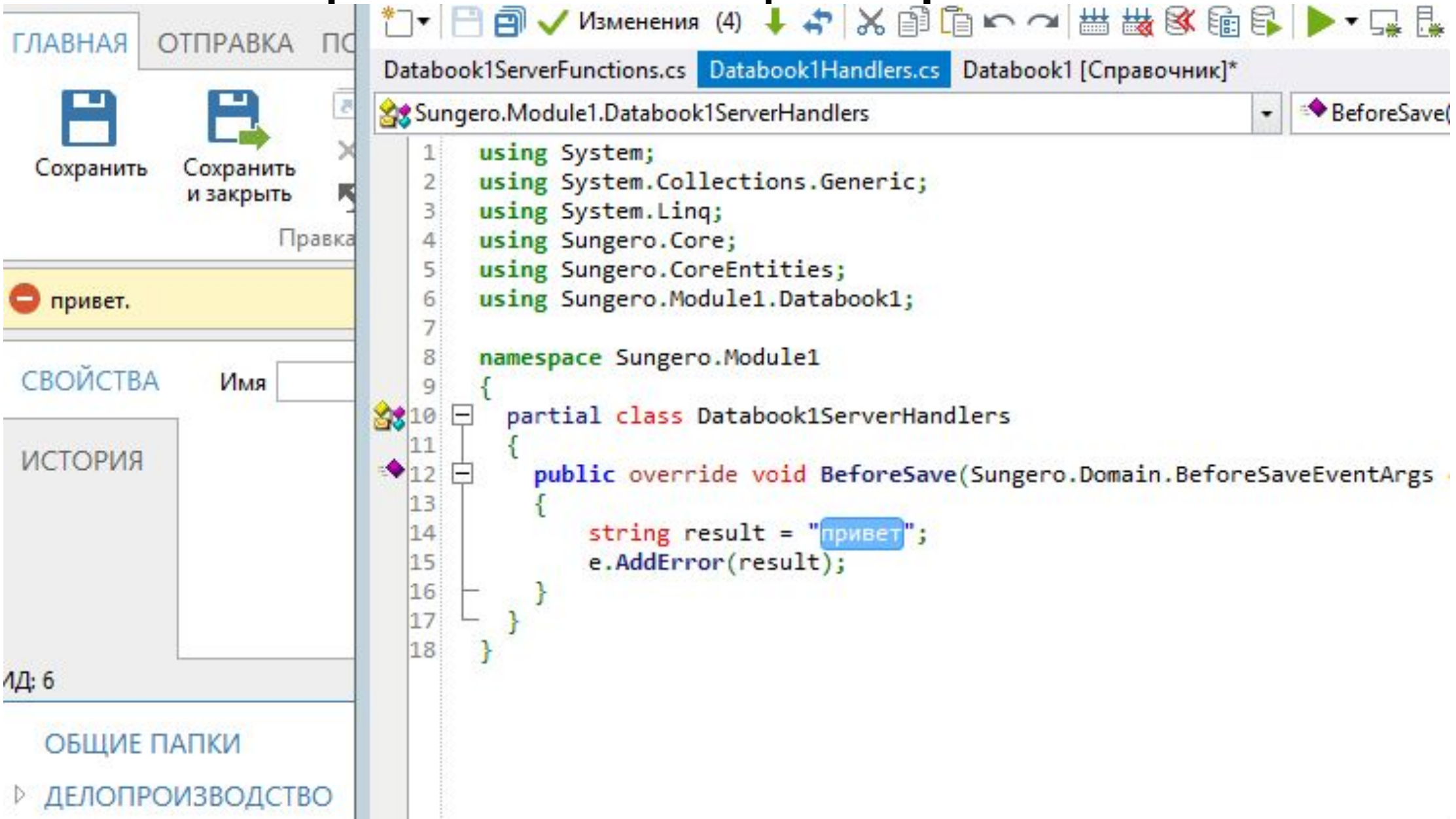


Class swap

- Платформенный код создаёт объекы, содержащие прикладной код.
- Зачем менять изворачиваться с методами, если можно просто подсунуть объект



Class swap. Работающий прототип



Мораль

Исследования – офигенная тема

- Правильно, что не стали бросаться делать первое попавшееся.
- Нужно смотреть шире. Упёрлись в подмену кода на низком уровне.
- Документирование исследования – вообще тема.
- Фича не забыта и может быть когда-нибудь...