Native AOT

Возможности и ограничения

Андрей Порожняков

.NET разработчик в SKAI

О чём поговорим

- Виды компиляций. Их отличия
- Native AOT. Требования и ограничения
- Метрики
- Что поддерживает Native AOT
- Поддержка Minimal API
- Плюсы, минусы и практическая польза

Разные подходы к компиляции

Момент компиляции
compile time

— синтаксический и семантический анализ кода

— оптимизация кода для повышения эффективности

— генерация компилянта

Момент выполнения
runtime

— загрузка программы в память компьютера

— выполнение программы

Компиляция в .NET

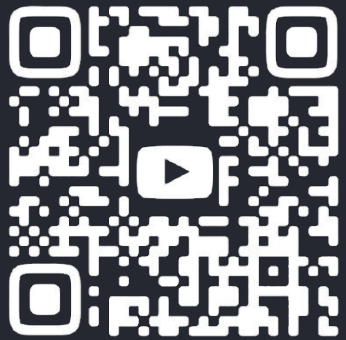
JIT-компиляция
just-in-time

В момент компиляции исходный код преобразуется в IL-код

В момент выполнения IL-код преобразуется в машинный код

Промежуточные варианты

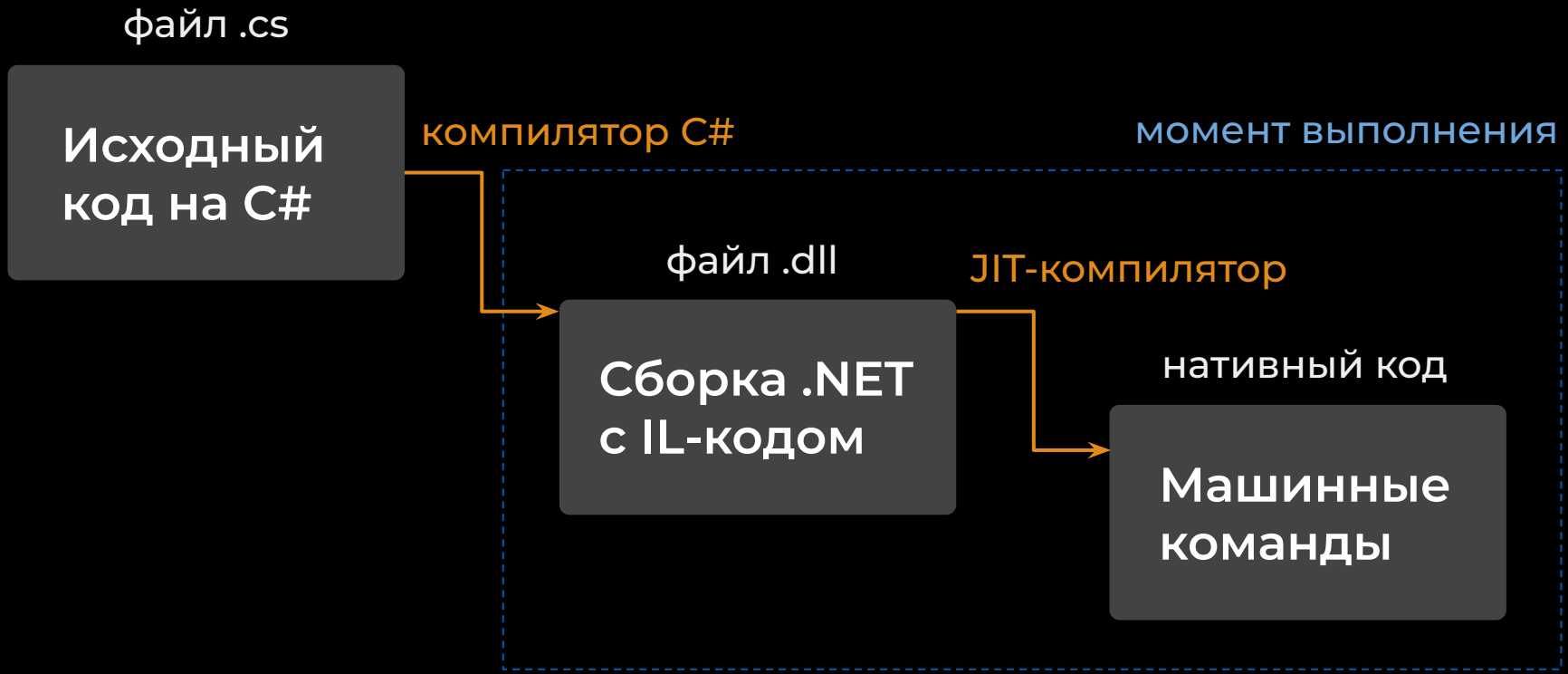
Например, Crossgen2



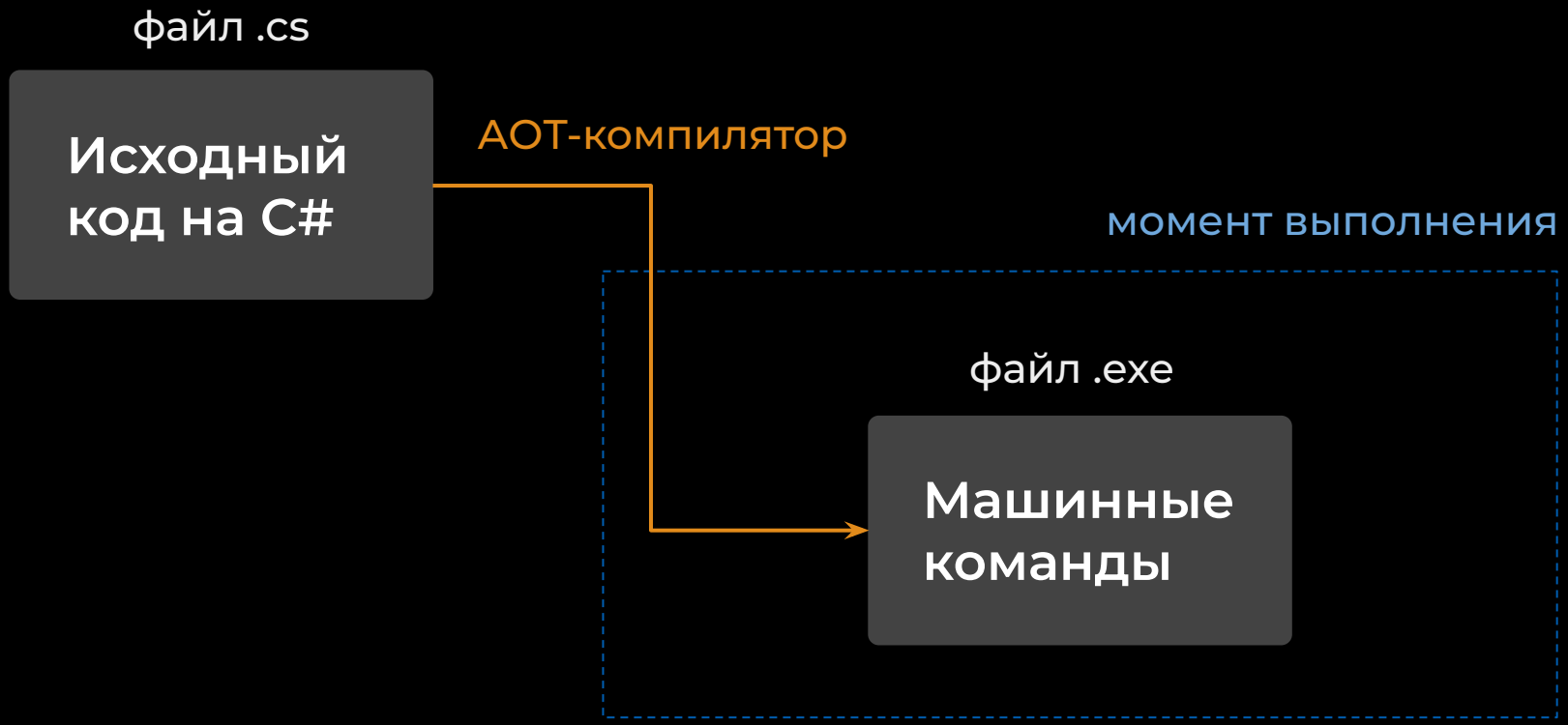
AOT-компиляция
ahead-of-time

В момент компиляции исходный код преобразуется в машинные команды

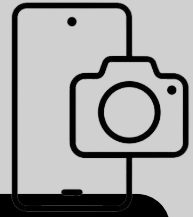
JIT-компиляция



AOT-компиляция



Важное



JIT-компилятор

Переводит IL-код
в машинный на лету
прямо перед его
выполнением

AOT-компилятор

Генерирует машинный
код в момент компиляции.
В момент выполнения
дополнительной
компиляции нет

Отличия JIT и AOT

Применимость в .NET

Как сравнивать JIT и AOT

Генерируемый код

Независимость от платформы

Момент компиляции

Использование памяти

Момент выполнения

Производительность

Генерируемый код

C#

```
System.Console.WriteLine("Hello, World!");
```

IL-код

```
ldstr "Hello, World!"
call void
System.Console::WriteLine(string)
nop
ret
```

Машинные команды

```
push ebp
mov ebp, esp
push edi
push eax
mov [ebp-8], ecx
cmp dword ptr [0x1b88c18c], 0
je short L0016
call 0x79671960
mov ecx, [0x8e70d0c]
call dword ptr [0x1370e5b0]
nop
nop
pop ecx
pop edi
pop ebp
ret
```

Момент компиляции

JIT

Исходный код → IL-код

AOT

Исходный код → машинный

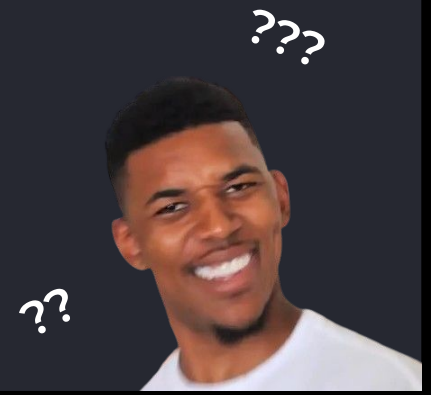
Момент выполнения

JIT

IL-код → машинный код

AOT

Ни-че-го



Независимость от платформы

JIT

Независимый от платформы IL-код в момент выполнения преобразуется в машинный код, адаптированный к окружению

AOT

Нужна отдельная компиляция под каждую целевую платформу. Код будет оптимизирован под неё

Использование памяти

JIT

В момент выполнения используется больше памяти из-за генерации машинного кода

AOT

Нагрузка на память в момент выполнения не растёт

Производительность

JIT

Запуск дольше, но в момент выполнения оптимизирует производительность — адаптирует сгенерированный код к характеристикам конкретного окружения

AOT

Более быстрый запуск, ведь программа компилируется заранее

Что в итоге

JIT и AOT — это разные стратегии перевода исходного кода в машинный с компромиссом между временем запуска, производительностью, использованием памяти и зависимостью от платформы

Выбор зависит от требований, ограничений приложения и сценариев использования

Зачем AOT в .NET

1

Сокращает время запуска приложения

Идеально для FaaS и бессерверных вычислений

Запросы обслуживаются быстрее

Оркестраторы контейнеров эффективнее управляют переходами между версиями

Зачем AOT в .NET

2

Занимает меньше места на диске

Образы контейнеров меньшего размера

Сокращается время развертывания

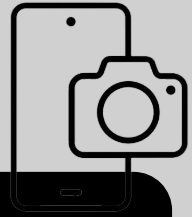
Зачем AOT в .NET

3

Использует меньше памяти во время работы

Большая плотность развертывания и улучшение масштабируемости

Важное



JIT-компиляция

Не зависит от платформы. Повышает производитель-ность, так как компилятор оптимизирует код в зависимости от аппаратной и программной среды

AOT-компиляция

Сокращает время запуска, потому что исходный код уже скомпилирован в машинные команды

Публикация Native AOT

Требования и ограничения

Варианты публикации через dotnet publish

Командная строка	Файл .csproj
-p:PublishSingleFile=true	<PublishSingleFile>true</PublishSingleFile>
-r linux-x64 --self-contained	<RuntimeIdentifier>linux-x64</RuntimeIdentifier> <SelfContained>true</SelfContained>
-p:PublishTrimmed=true	<PublishTrimmed>true</PublishTrimmed>
-p:PublishAot=true	<PublishAot>true</PublishAot>

Требования к публикации Native AOT

Desktop Development for C++ для Windows

Command Line Tools for XCode для MacOS

Clang и zlib1g-dev для Ubuntu

Соответствие среды публикации и целевой платформы

Ограничения публикации Native AOT

Тримминг

— нет динамической загрузки, например, Assembly.LoadFile

— нет генерации кода с помощью JIT в момент выполнения, например, System.Reflection.Emit

— нет C++ и CLI

Публикация в один файл

— нет встроенного COM для Windows

— приложения содержат все нужные библиотеки. Как и для self-contained, их размер увеличивается

Ограничения тримминга

Проблема:

Получение информации о типе

Решение:

Атрибут [DynamicallyAccessedMembers]

Ограничения тримминга

Проблема:

Генерация кода в момент выполнения.
Например, Type.MakeGenericType

Решение:

Генераторы исходного кода Roslyn

Ограничения публикации в один файл

Член класса	Результат обращения
Assembly.CodeBase	Бросает PlatformNotSupportedException
Assembly.EscapedCodeBase	Бросает PlatformNotSupportedException
Assembly.GetFile()	Бросает IOException
Assembly.GetFiles()	Бросает IOException
Assembly.Location	Возвращает пустую строку
AssemblyName.CodeBase	Возвращает null
AssemblyName.EscapedCodeBase	Возвращает null
Module.FullyQualifiedName	Возвращает строку со значением <Unknown> или вызывает исключение
Module.Name	Возвращает строку со значением <Unknown>
Marshal.GetHINSTANCE()	Возвращает значение -1

Важное

Метрики

Как собирал метрики

HelloWorld-приложение, использующее GoogleApi

Сравнение показателей Self-contained single file и Native AOT

Запуск приложения в Yandex Cloud Functions, среда выполнения bash

Сравнение холодного старта и подготовленных экземпляров

Показатели

Среднее время работы функции за 100 запусков

Объём выделенной памяти

Размер файла

Запуск в Yandex Cloud Functions

Измеряемый показатель	Self-contained single file	Native AOT
Среднее время работы HW-приложения за 100 запусков	51.52 ms / 100.8 ms	3.62 ms / 28.82 ms
Размер HW-приложения / память в YCF при выполнении	13 106 KB / 58 MB	1 540 KB / 43 MB
Среднее время работы реального приложения за 100 запусков	1343.15ms / 1420.56 ms	486.87 ms / 583.52 ms
Размер реального приложения / память в YCF при выполнении	15 835 KB / 91 MB	9 840 KB / 74 MB

Что поддерживает Native AOT

Поддержка в ASP.NET Core

🟢 Полностью

gRPC, CORS, JWT Authentication, HealthChecks, Rewrite, HttpLogging, Localization, OutputCaching, RateLimiting, RequestDecompression, ResponseCaching, ResponseCompression, StaticFiles, WebSockets

🟡Частично

Minimal APIs

🔴Нет поддержки

MVC, Blazor Server, Razor Pages, SignalR, Session, Spa

Поддержка в других решениях

PostgreSQL

OpenTelemetry

SQLite

EF Core

Dapper AOT

❓ MediatR

Поддержка Minimal API

Шаблон ASP.NET Core Web API (Native AOT)

Проект создаётся командой dotnet new webapiaot

Значимые изменения в файле .csproj

- <InvariantGlobalization>true </InvariantGlobalization>
- <PublishAot>true</PublishAot>

Значимые изменения в файле Program.cs

- Использует .CreateSlimBuilder(args) вместо .CreateBuilder(args)
- Подключает генератор исходного кода JSON
- Каждый сериализуемый объект должен быть явно прописан в JsonSerializerContext

Пример Minimal API-приложения

```
1.  var builder = WebApplication.CreateSlimBuilder(args);
2.  builder.Services.ConfigureHttpJsonOptions(options ⇒
3.  {
4.      options.SerializerOptions.TypeInfoResolverChain.Insert(0,
    AppJsonSerializerContext.Default);
5.  });
6.  var app = builder.Build();
7.  app.MapGet("/", () ⇒ new MyObj("Hello!"))
8.  app.Run();
9.
10. [JsonSerializable(typeof(MyObj))]
11. internal partial class AppJsonSerializerContext :
    JsonSerializerContext { }
```

Пример Minimal API-приложения

```
1.  var builder = WebApplication.CreateSlimBuilder(args);
2.  builder.Services.ConfigureHttpJsonOptions(options ⇒
3.  {
4.      options.SerializerOptions.TypeInfoResolverChain.Insert(0,
    AppJsonSerializerContext.Default);
5.  });
6.  var app = builder.Build();
7.  app.MapGet("/", () ⇒ new MyObj("Hello!"))
8.  app.Run();
9.
10. [JsonSerializable(typeof(MyObj))]
11. internal partial class AppJsonSerializerContext :
    JsonSerializerContext { }
```

Что не поддерживает CreateSlimBuilder

- Hosting startup assemblies и вызов .UseStartup<Startup>()
- Загрузку статических ассетов из подключенных проектов и пакетов через вызов .UseStaticWebAssets()
- Интеграцию с IIS
- HTTPS и Quic (HTTP/3)
- RegEx-выражения в роутинге
- Провайдеры логирования Windows event log, Windows Event Tracing и в debugger console

Генерация делегатов запросов

RequestDelegateFactory превращает вызовы Map*()
для конкретных маршрутов в RequestDelegate
с помощью System.Reflection.Emit

RequestDelegateGenerator — это генератор исходного кода. Выполняет аналогичную работу, но в момент компиляции

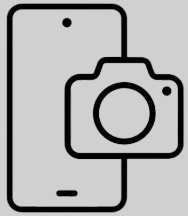
Можно использовать для проектов без Native AOT

Генерация делегатов запросов

EnableRequestDelegateGenerator	false	true
Время первого запроса для приложения с 1002-мя маршрутами	1082 ms	176 ms

Источник: Performance Improvements in ASP.NET Core 8

Важное



Native AOT подходит не только для FaaS-решений, но и для Web API

При разработке необходимо следить, чтобы каждый сериализуемый объект был явно прописан в JsonSerializerContext

Приложения без AOT-компиляции работают быстрее с помощью <EnableRequestDelegate Generator>true</EnableRequestDelegateGenerator>

Плюсы, минусы и практическая польза

Плюсы

+ быстрый старт приложения
+ хорошо подходит для небольших проектов

Минусы

– не всё поддерживается
– перевод проекта требует дополнительных затрат

Польза Native AOT

Эффективно решает определённые задачи

Активно развивается и получает всё больше возможностей

Приносит интересные фичи в проекты с JIT-компиляцией

Источники

How to make libraries compatible with native AOT

Performance Improvements in ASP.NET Core 8

ASP.NET Core support for Native AOT

ASP.NET Core Request Delegate Generator (RDG) for Native AOT

Yandex Cloud Functions

Andrew Lock about .NET 8

GitHub - dotnet/aspnetcore

Мои контакты

Порожняков Андрей

📧aporozhniakov@gmail.com

хабр, линк, почта

Конфигурация Native AOT-приложения

Полезные настройки

<PublishAot>

Назначение

Выполняет компиляцию AOT
во время публикации. Включает анализ
использования динамического кода
во время сборки и редактирования

Параметры

true false

<InvariantGlobalization>

Назначение

Удаляет зависимости приложения
от данных и поведения глобализации

Параметры

true false

<ServerGarbageCollection>

Назначение

Включает или отключает сборщик мусора сервера. Отключение уменьшает потребление памяти

Параметры

true false

<EmitCompilerGeneratedFiles>

Назначение

Сохраняет файлы генератора исходного кода в файловой системе

Параметры

true false

<OptimizationPreference>

Назначение

Компромисс между размером исполняемого файла и созданием теоретически более быстрого исполняемого файла

Параметры

Size Speed

<IsAotCompatible>

Назначение

Указывает, совместима ли библиотека
с Native AOT

Параметры

true false

<StripSymbols>

Назначение

Включает отладочную информацию в двоичный файл на Unix-подобных платформах

Параметры

true false

<RuntimeIdentifier>, <RuntimeIdentifiers>

Назначение

Указывает идентификатор среды выполнения

Параметры

linux-x64 win-x64

Все идентификаторы

<SelfContained>

Назначение

Позволяет исполнять приложение
без использования системного .NET

Параметры

true false

<PublishTrimmed>

Назначение

Включает тримминг для приложений
с автономным развёртыванием

Параметры

true false

<PublishSingleFile>

Назначение

Включает публикацию одного файла

Параметры

true false

<EnableRequestDelegateGenerator>

Назначение

Включает преобразование методов map
в делегаты запросов с помощью
генератора исходного кода

Параметры

true false