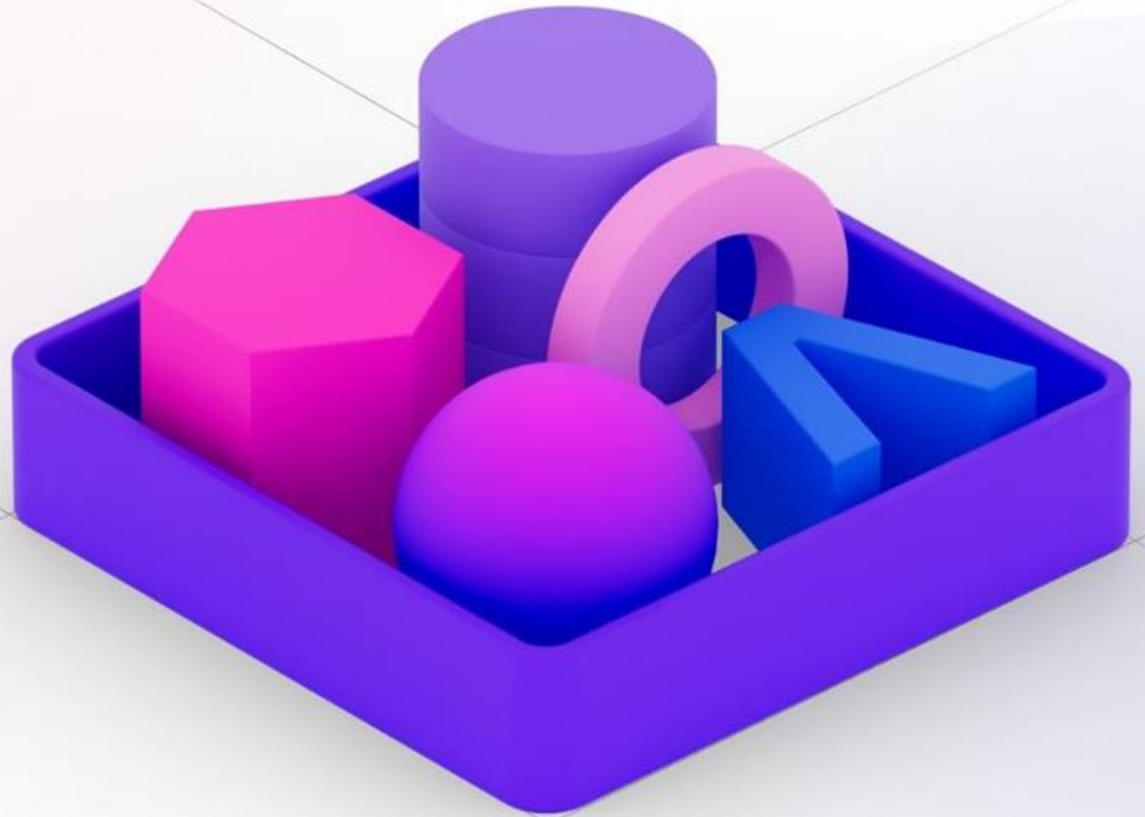


.NET Conf
China 2024
.NET Talks



.NET Talks | .NET 9 AOT 的突破

姚圣伟 | Conan Yao

<https://github.com/JaneConan/>

<https://www.linkedin.com/in/conanyao/>



个人介绍



<https://github.com/JaneConan>

姚圣伟

在职软件工程师，微软 Insider Dev Tour China 、.NET Conf China 讲师，盛派开发者社区主创人员，Ant Design Blazor 社区贡献者，中国 DevOps 社区理事会成员天津地区核心组织者，华为云云享专家，腾讯腾源会开源创新100人。首届 .NET Conf 黑客松北京赛区汗八里小队队长、.NET 20周年云原生开发挑战赛获奖团队选手。

热衷于学习和分享可落地的新技术和新文化。目前致力于在国产化项目中实践新技术，并立志于将所习得的新技术和思想分享给更多开发者去解决实际问题。

什么是 JIT 和 AOT ?

语言在运行之前通常都需要编译，最常见的两种编译模式：

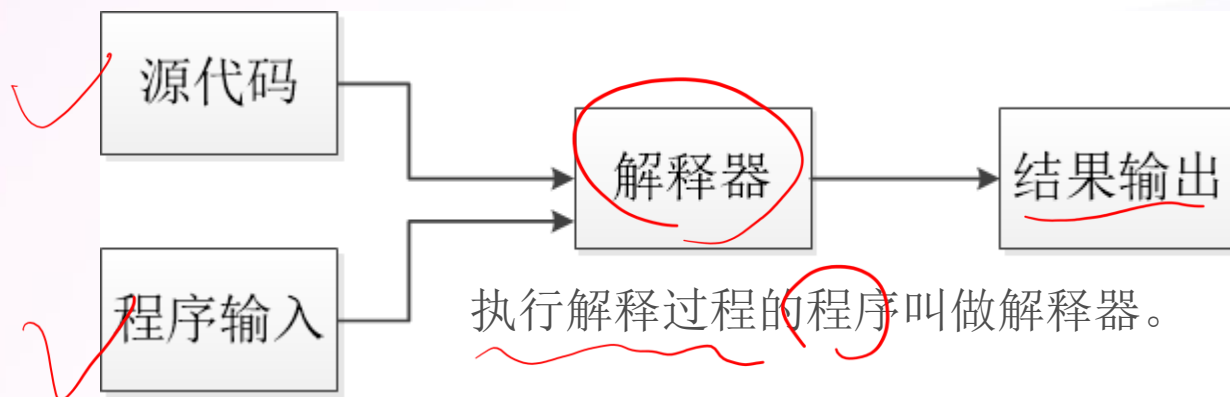
1、即时编译 **JIT(Just-in-Time)** —— 典型代表是 JavaScript、Python 等几乎所有的脚本语言和 **C#**

解释：

将源代码逐条转换成目标代码，

同时逐条运行的过程

类似英语中的同声传译

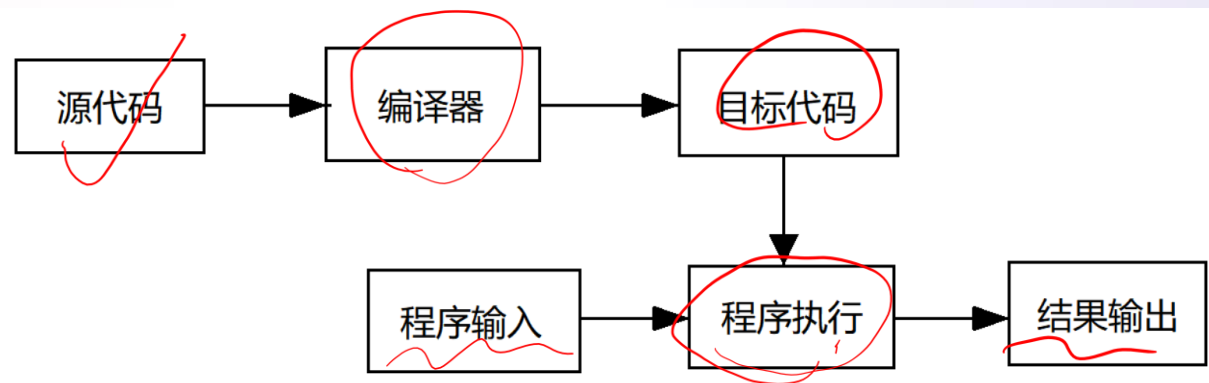


2、提前编译 **AOT(Ahead-of-Time)** —— 典型代表是 C/C++，必须在执行前编译成机器码

编译：

将源代码一次性转换成目标代码的过程

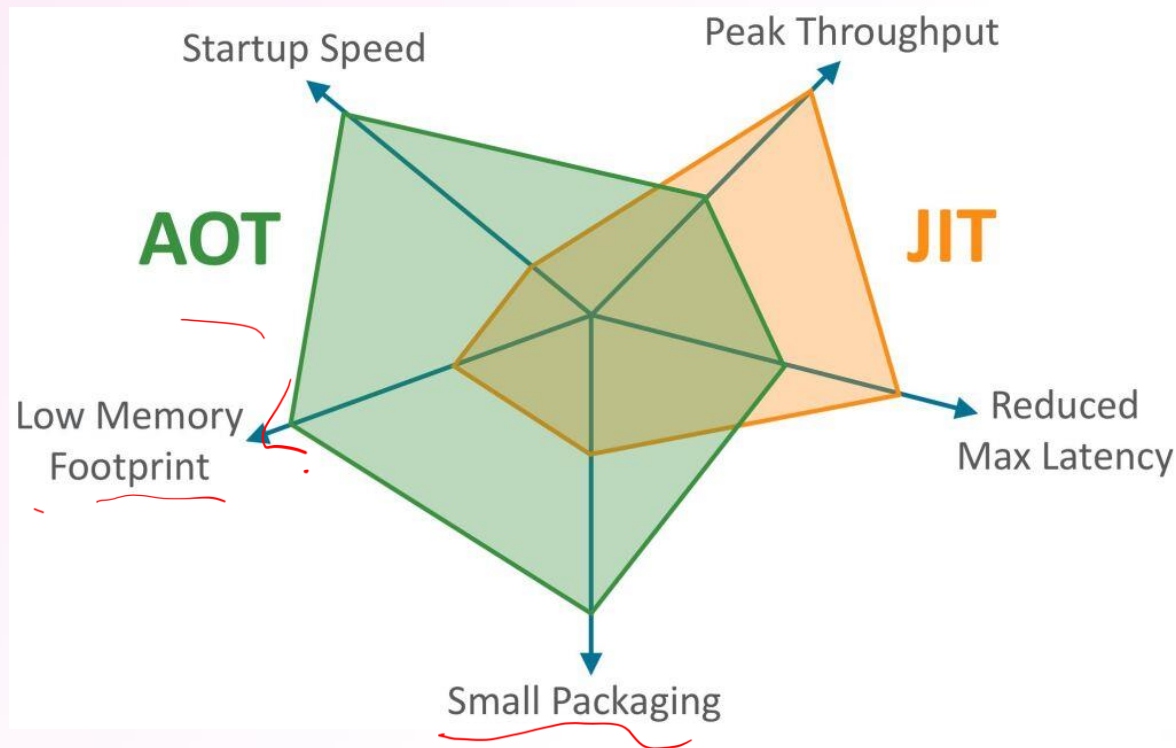
类似英语中的全文翻译



JIT 和 AOT 的差异

AOT 编译器

1. 启动时间：
较快
2. 运行时性能：
缺乏动态优化
稳定性
3. 内存占用：
较低



JIT 编译器

1. 启动时间：
较慢
2. 运行时性能：
动态优化
自适应优化
3. 内存占用：
较高

这两种编译方式的主要区别在于是否在“运行时”进行编译

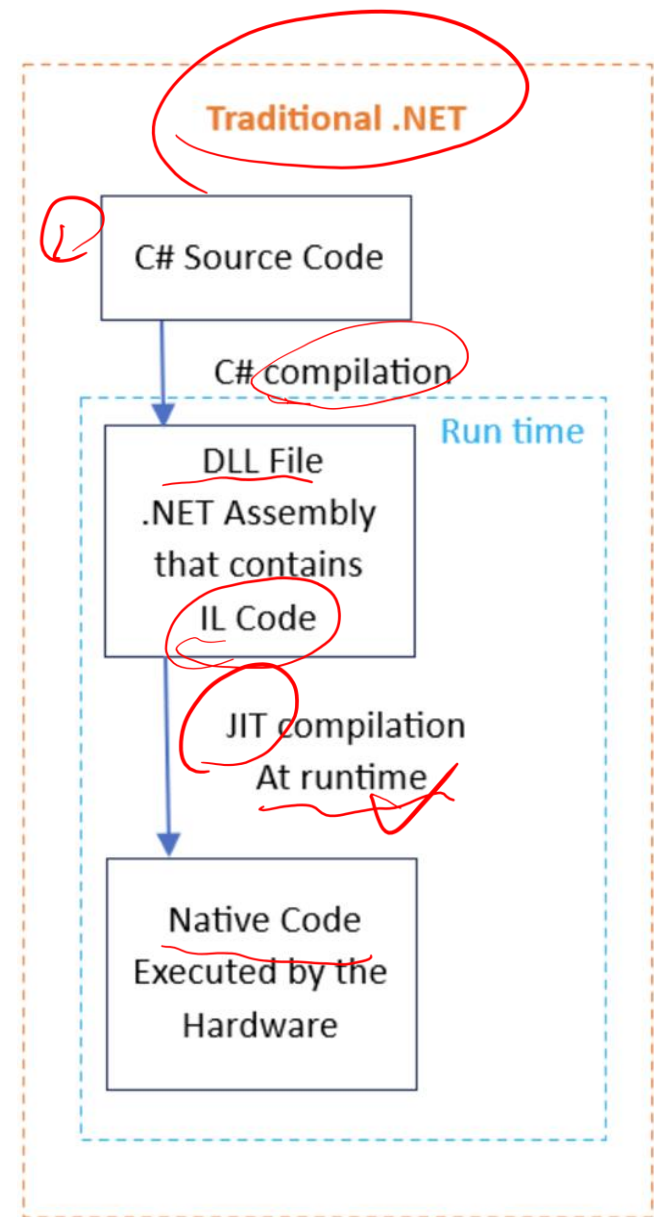
.NET 传统编译方式 JIT

1、C# 编译生成时：

包含中间语言 (IL) 代码的 DLL 文件
此类 DLL 称为 .NET 程序集 ~~XXXX~~ *dll*

2、执行 .NET 程序时：

在 .NET 运行时 (CLR 公共语言运行时)
将加载 .NET 程序集 (DLL) CLR 的子系统
此子系统称为 JIT (Just-In-Time) 编译器
负责将 IL 代码编译为由 CPU 能直接执行的本机代码



程序的执行 [编辑源代码]

C#通常不被编译成为能够直接在计算机上执行的二进制本地代码。与Java类似，它被编译成为中间代码（Microsoft Intermediate Language），然后通过.NET Framework的虚拟机——被称为通用语言运行库——执行。

所有的.Net编程语言都被编译成这种被称为通用中间语言的中间代码。因此虽然最终的程序在表面上仍然与传统意义上的可执行文件都具有“.exe”的后缀名。如果计算机上没有安装.Net Framework，那么这些程序会弹出对话框，要求用户下载.net framework。

在程序执行时，.Net Framework将中间代码翻译成为二进制机器码，从而使它得到正确的运行。最终的二进制代码被存储在一个缓冲区（Buffer）中。所以一旦程序使用了相同的代码，那么将会调用缓冲区中的版本。这样如果一个.Net程序第二次被运行，那么这种翻译不需要进行第二次，速度明显加快。

实现 [编辑源代码]

微软正在引领开源参考 C# 编译器和工具集的开发。第一个编译器 Roslyn 编译成中间语言 (IL)，第二个编译器 RyuJIT，^[47] 是一个 JIT（即时）编译器，它是动态的，进行动态优化并编译将 IL 转换为 CPU 前端的本机代码。^[48] RyuJIT 是开源的，用 C++ 编写。^[49] Roslyn 完全是用 托管代码 (C#) 编写的，已经开放并且功能以 API 的形式出现。因此，它使开发人员能够创建重构和诊断工具。^{[2][50]} 官方实现的两个分支是 .NET Framework（闭源，仅限 Windows）和 .NET Core（开源，跨平台）；它们最终融合为一个开源实现：.NET 5.0。^[51] 在 .NET Framework 4.6 中，新的 JIT 编译器取代了前者。^{[47][52]}

其他 C# 编译器（其中一些包括公共语言基础结构和 .NET 类库的实现）：

- 微软的Rotor项目（Rotor Project，目前称为Shared Source Common Language Infrastructure），提供了通用语言运行库（Common Language Runtime）的实作与C# 编译器。但是Shared Source Common Language Infrastructure在2006年的2.0版后就停止了。
- 由Microsoft赞助的Mono 项目提供了C# 编译器，它提供了一个开源 C# 编译器、一个完整的 CLI 开源实现，同时也接近百分之百地实作了.NET Framework类库。而Mono后来行伸出由微软认可的第三方套件Xamarin。
- Dot GNU 专案（现已停产）也提供了另一个自由版本的C# 编译器，也提供了.NET Framework类库的实作。

游戏引擎 Unity 使用C# 作为其主要脚本语言。由于Microsoft 捐赠了 24,000 美元，Godot 游戏引擎实现了一个可选的 C# 模块。

维基百科 C#

<https://zh.wikipedia.org/zh-hans/C%E2%99%AF>

什么是 .NET Native AOT

.NET Native AOT (**Ahead-of-Time**)提前编译

是 .NET 平台中的一项前沿进步

使用 AOT 时, C# 代码在开发人员计算机上被编译为本机代码

.NET Native AOT 编译由 一个步骤 组成:

将 C# 源代码编译为开发人员计算机上的本机代码

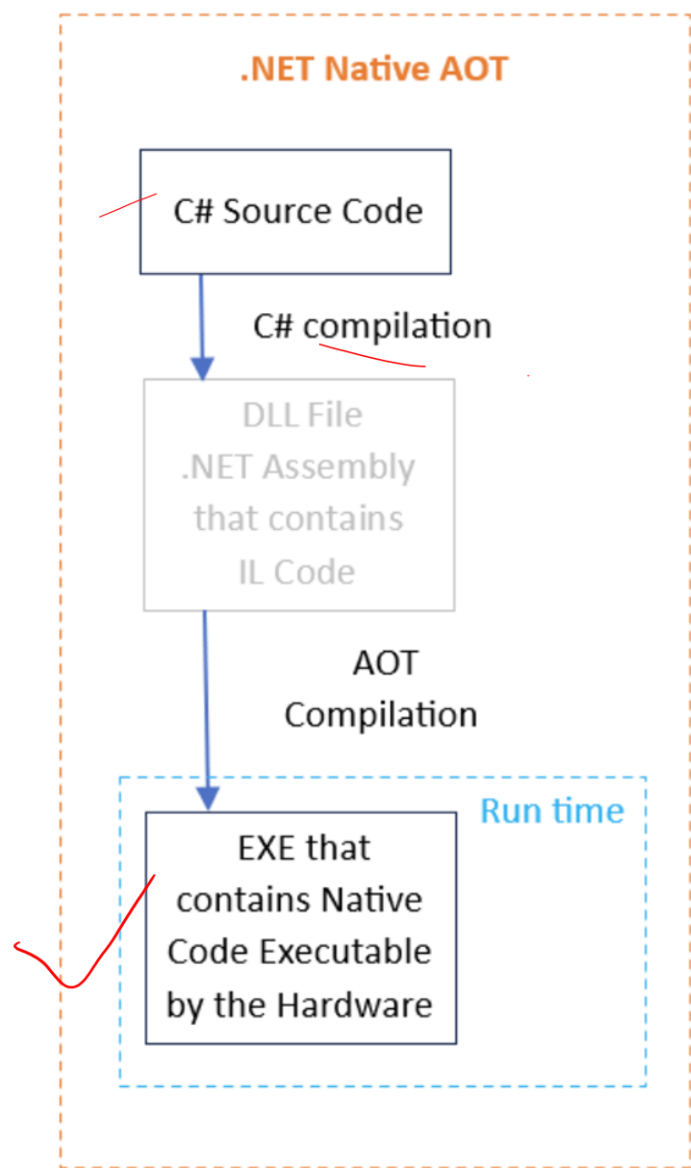
此过程包括将 C# 代码转换为 IL (中间语言) 代码, 然后转换为 Native 代码, 形成一个两步编译过程。但这是一个实现细节。
这就是 AOT .NET 程序集 (DLL) 框在右侧的架构中为灰色的原因

《.NET Native AOT 的发展现状如何?》

https://mp.weixin.qq.com/s/jnrT--EO_zV8e_uYx_xCnA ✓

《.NET 8.0 本机 AOT》

https://blog.csdn.net/hefeng_aspnet/article/details/135400417 ✓



.NET Native AOT 编译过程

JIT

C#



IL



linux-x64/arm64
osx-x64/arm64
win-x64/arm64

AOT

C#



IL



linux-x64/arm64
osx-x64/arm64
win-x64/arm64

Demo

ASP.NET Core Web API (native AOT) with Minimal APIs

. Net 8 & . Net 9

微软官方容器镜像

<https://mcr.microsoft.com/zh-cn>



Demo

ASP.NET Core Web API (native AOT) with Minimal APIs

. Net 9

跨平台编译

正在完善中

```
Windows PowerShell
> [build 6/8] COPY . . 0.1s
> ERROR [build 7/8] RUN dotnet publish -r linux-amd64 --no-restore -o /app releasesapi.csproj 33.5s

-----
> [build 7/8] RUN dotnet publish -r linux-amd64 --no-restore -o /app releasesapi.csproj:
1.848 /usr/share/dotnet/sdk/9.0.100-rc.2.24468.2/Sdks/Microsoft.NET.Sdk/targets/Microsoft.NET.RuntimeIdentifierInference
.targets(326,5): message NETSDK1057: You are using a preview version of .NET. See: https://aka.ms/dotnet-support-policy
[/source/releasesapi.csproj]
8.181 releasesapi -> /source/bin/Release/net9.0/linux-x64/releasesapi.dll
8.914 Generating native code
33.13 /usr/bin/ld.bfd: cannot find crtbeginS.o: No such file or directory
33.34 /usr/bin/ld.bfd: cannot find -lgcc: No such file or directory
33.37 /usr/bin/ld.bfd: cannot find -lgcc: No such file or directory
33.41 clang : error : linker command failed with exit code 1 (use -v to see invocation) [/source/releasesapi.csproj]
33.41 /root/.nuget/packages/microsoft.dotnet.ilcompiler/9.0.0-rc.2.24463.7/build/Microsoft.NETCore.Native.targets(376,5)
: error MSB3073: The command "clang" "obj/Release/net9.0/linux-x64/native/releasesapi.o" -o "bin/Release/net9.0/linux-x
64/native/releasesapi" -WL,--version-script=obj/Release/net9.0/linux-x64/native/releasesapi.exports -WL,--export-dynamic
-gz=zlib -fuse-ld=bfd /root/.nuget/packages/runtime.linux-x64.microsoft.dotnet.ilcompiler/9.0.0-rc.2.24463.7/sdk/libboo
tstrapper.o /root/.nuget/packages/runtime.linux-x64.microsoft.dotnet.ilcompiler/9.0.0-rc.2.24463.7/sdk/libRuntime.Server
GC.a /root/.nuget/packages/runtime.linux-x64.microsoft.dotnet.ilcompiler/9.0.0-rc.2.24463.7/sdk/libeventpipe-enabled.a /
root/.nuget/packages/runtime.linux-x64.microsoft.dotnet.ilcompiler/9.0.0-rc.2.24463.7/sdk/libRuntime.VxsortEnabled.a /ro
ot/.nuget/packages/runtime.linux-x64.microsoft.dotnet.ilcompiler/9.0.0-rc.2.24463.7/sdk/libstandalonegc-disabled.a /root
/.nuget/packages/runtime.linux-x64.microsoft.dotnet.ilcompiler/9.0.0-rc.2.24463.7/sdk/libstdc++compat.a /root/.nuget/pac
kages/runtime.linux-x64.microsoft.dotnet.ilcompiler/9.0.0-rc.2.24463.7/framework/libSystem.Native.a /root/.nuget/package
s/runtime.linux-x64.microsoft.dotnet.ilcompiler/9.0.0-rc.2.24463.7/framework/libSystem.IO.Compression.Native.a /root/.nu
get/packages/runtime.linux-x64.microsoft.dotnet.ilcompiler/9.0.0-rc.2.24463.7/framework/libSystem.Net.Security.Native.a
/root/.nuget/packages/runtime.linux-x64.microsoft.dotnet.ilcompiler/9.0.0-rc.2.24463.7/framework/libSystem.Security.Cryp
tography.Native.OpenSsl.a /root/.nuget/packages/runtime.linux-x64.microsoft.dotnet.ilcompiler/9.0.0-rc.2.24463.7/sdk/lib
z.a --target=x86_64-linux-gnu -g -WL,-rpath,'$ORIGIN' -WL,--build-id=sha1 -WL,--as-needed -pthread -ldl -lrt -lm -pie -W
l,-pie -WL,-z,relro -WL,-z,now -WL,--eh-frame-hdr -WL,--discard-all -WL,--gc-sections" exited with code 1. [/source/rele
asesapi.csproj]
```

为什么要将 .NET Native AOT 与 ASP.NET Core 配合使用

发布

发布

连接

设置

FolderProfile *

配置(C): Release

目标框架(F): net9.0

部署模式(M): 独立
[了解有关部署模式的信息](#)

目标运行时(U): win-x64

文件发布选项

- ☒ 生成单个文件
- ☒ 启用 ReadyToRun 编译
- ☒ 裁剪未使用的代码
- ☐ 在发布前删除所有现有文件(A)

数据库

i 未在项目中找到数据库

< 上一页(R) 下一页(X) > 保存

本机 AOT 发布

本机 AOT 通过 `PublishAot` MSBuild 属性启用。以下示例演示如何在项目文件中启用本机 AOT:

```
XML
<PropertyGroup>
  <PublishAot>true</PublishAot>
</PropertyGroup>
```

此设置将在发布期间启用本机 AOT 编译，并在生成和编辑期间启用动态代码使用情况分析。在本地运行时，使用本机 AOT 发布的项目将使用 JIT 编译。AOT 应用与 JIT 编译的应用有以下区别：

- 与本机 AOT 不兼容的功能会被禁用，并在运行时引发异常。
- 启用源分析器以突出显示与本机 AOT 不兼容的代码。在发布时，将再次分析整个应用（包括 NuGet 包）是否兼容。

本机 AOT 分析包括应用的所有代码和应用依赖的库。查看本机 AOT 警告并采取纠正措施。建议经常发布应用，以在开发生命周期的早期发现问题。

在 .NET 8 中，以下 ASP.NET Core 应用类型支持本机 AOT：

- 最小 API - 有关详细信息，请参阅本文后面的 [Web API（本机 AOT）模板部分](#)。
- gRPC - 有关详细信息，请参阅 [gRPC 和本机 AOT](#)。
- 辅助角色服务 - 有关详细信息，请参阅 [辅助角色服务模板中的 AOT](#)。

为什么要将 .NET Native AOT 与 ASP.NET Core 配合使用



.NET Native AOT 和 ASP.NET Core 功能的兼容性

ASP.NET Core 和本机 AOT 兼容性

目前，并非所有 ASP.NET Core 功能都与本机 AOT 兼容。下表汇总了 ASP.NET Core 功能与本机 AOT 的兼容性：

功能	完全支持	部分支持	不支持
gRPC	✓		
最小 API		✓	
MVC			✗
Blazor Server			✗
SignalR		✓	
JWT 身份验证	✓		
其他身份验证			✗
CORS	✓		
HealthChecks	✓		
HttpLogging	✓		

功能	完全支持	部分支持	不支持
本地化	✓		
OutputCaching	✓		
RateLimiting	✓		
RequestDecompression	✓		
ResponseCaching	✓		
ResponseCompression	✓		
Rewrite	✓		
会话			✗
Spa			✗
StaticFiles	✓		
WebSockets	✓		

.NET Native AOT的优势

- 1、增强性能：通过将代码提前编译为本机计算机指令，显著缩短了启动时间并提高了应用程序的整体性能。在无服务器方案中，如果应用程序针对每个请求启动，这可能会产生重大差异。运行时没有 **JIT** 编译开销，执行速度更快，提供更流畅的用户体验。
- 2、简化部署：**AOT** 编译的应用程序通常会导致依赖项为零或较少的独立可执行文件。这简化了部署过程，可以更轻松地各种平台和设备之间分发应用程序，而无需额外的安装或运行时组件。
- 3、更小的应用程序大小：通过 **修剪** 不必要的代码，**AOT** 可以大大减小应用程序的大小。节省存储空间，优化应用程序的内存占用，这在**移动设备或 IoT 设备**等资源受限的环境中尤为重要。
- 4、增强知识产权的保护：**AOT** 编译将源代码转换为优化的机器代码，这使得**逆向工程**尝试更具挑战性。生成的本机代码比 **IL** 代码更加模糊，并且**难以破译**，因为 **IL** 代码可以轻松反编译为原始 **C#** 代码。这增强了应用程序中嵌入的敏感算法、业务逻辑和专有方法的安全性。

《.NET Native AOT 的发展现状如何？》

https://mp.weixin.qq.com/s/jnrT--EO_zV8e_uYx_xCnA

.NET Native AOT的缺点

1、特定于平台的编译：AOT 生成特定于平台的本机代码，针对特定体系结构或操作系统进行定制。

例如，与常规 .NET 程序集不同，在 Windows 上使用 AOT 生成的可执行文件在 Linux 上不起作用。

2、本机 AOT 确实对跨体系结构编译提供有限的支持。例如，从 Linux 针对 Windows 进行编译，或者从 x64 针对 ~~Arm64~~ 进行编译。在 Linux 上，差异还可能见于标准 C 库实现 - **glibc**（例如 Ubuntu Linux）或 musl（例如 Alpine Linux）。

<https://learn.microsoft.com/zh-cn/dotnet/core/deploying/native-aot/cross-compile#linux>

筆記 - 在 Linux 平台使用 .NET Native AOT

<https://blog.darkthread.net/blog/publish-native-aot-on-linux/>

《.NET Native AOT 的发展现状如何？》

https://mp.weixin.qq.com/s/jnrT--EO_zV8e_uYx_xCnA

.NET Native AOT的缺点

3、对 Reflection 的部分支持：反射依赖于动态代码生成和运行时类型发现，这与 AOT 编译代码的提前编译和静态性质相冲突。

Ⅱ

例如：

```
var type = Type.GetType("Foo");  
Activator.CreateInstance(type);  
  
class Foo  
{  
    public Foo() => Console.WriteLine("Foo instantiated");  
}
```

输出
Foo instantiated。

《.NET Native AOT 的发展现状如何？》

https://mp.weixin.qq.com/s/jnrT--EO_zV8e_uYx_xCnA

.NET Native AOT的缺点

3、对 **Reflection** 的部分支持：反射依赖于动态代码生成和运行时类型发现，这与 **AOT** 编译代码的提前编译和静态性质相冲突。

但是如果我们把代码改为如下：

```
var type = Type.GetType(Console.ReadLine());  
Activator.CreateInstance(type);  
  
class Foo  
{  
    public Foo() => Console.WriteLine("Foo instantiated");  
}
```

现在让我们用 NativeAOT 构建并运行这个程序，然后输入 **Foo** 来创建一个 **Foo** 的实例。

《.NET Native AOT 的发展现状如何？》
https://mp.weixin.qq.com/s/jnrT--EO_zV8e_uYx_xCnA

.NET Native AOT的缺点

3、对 Reflection 的部分支持：反射依赖于动态代码生成和运行时类型发现，这与 AOT 编译代码的提前编译和静态性质相冲突。

输出

你会立刻得到一个异常：

```
Unhandled Exception: System.ArgumentNullException: Value cannot be null. (Parameter 'type')
  at System.ArgumentNullException.Throw(String) + 0x2b
  at System.ActivatorImplementation.CreateInstance(Type, Boolean) + 0xe7
  ...
```

这是因为编译器无法看到你在哪里使用了 **Foo**，所以它根本不会为 **Foo** 生成任何代码，导致这里的 **type** 为 **null**。

因为**依赖图**是在**编译期间静态构建**的，这也意味着任何**无法静态分析**的东西都不会被编译。不幸的是，反射，即在不事先告诉编译器的情况下在运行时获取东西，正是编译器无法弄清楚的一件事。

《.NET Native AOT 的发展现状如何？》

https://mp.weixin.qq.com/s/jnrT--EO_zV8e_uYx_xCnA/

.NET Native AOT的缺点

- 4、需要 AOT 兼容的依赖项：AOT 编译要求项目中使用的所有库和依赖项都与 AOT 兼容。依赖于反射、运行时代码生成或其他动态行为的库可能与 AOT 不兼容，这可能会导致冲突或运行时错误。
- 5、增加构建时间：AOT 编译涉及在构建过程中预先生成本机代码。这个额外的步骤会显著增加构建时间，特别是对于大型项目或具有大量代码库的应用程序。
- 6、需要适用于 C++ 的桌面开发工具：AOT 只能在安装这些工具的情况下进行编译，这些工具在您的硬盘驱动器上最多可占用 7GB 硬盘空间。

《.NET Native AOT 的发展现状如何？》

https://mp.weixin.qq.com/s/jnrT--EO_zV8e_uYx_xCnA

.NET 9 Preview 1 库和运行时的更新

<https://github.com/dotnet/runtime/discussions/98372>

运行时更新中

重点提及

Native AOT

JIT

Libraries and runtime updates in .NET 9 Preview 1 #98372

richlander announced in Announcements



richlander on Feb 14 Collaborator

edited by jamesmontemagno ...

.NET 9 Preview 1 includes several new libraries and runtime features. We focused on the following areas:

Libraries:

- System.Collections
- System.Linq
- System.Reflection
- System.Security.Cryptography
- System.Text.Json

Runtime:

- Native AOT
- JIT

Release notes:

- [Libraries](#)
- [Runtime](#)
- [What's new in the .NET in .NET 9](#) documentation.
- [Breaking Changes in .NET 9](#)

.NET 9 Preview 1:

- [Discussion](#)
- [Release notes](#)

7 4 5 7

.NET 9 中 Native AOT 的突破

.NET 对于 64 位 JIT 编译器内联方的目标之一是尽可能多地消除阻止方法被内联的限制。

.NET 9 支持在 Windows x64、Linux x64 和 Linux Arm64 上内联对线程本地静态的访问。

对于 static 类成员，该成员的一个实例存在于该类的所有实例中，这些实例“共享”该成员。

如果 **static** 成员的值对于每个线程都是唯一的，则将该值设为线程本地可以提高性能，

因为这样并发原语便无需从其包含的线程安全访问 **static** 成员。

以前，访问本机 AOT 编译的程序中的线程本地静态数据需要 64 位 JIT 编译器向运行时发出调用以获取线程本地存储的基地址。GitHub: [在 NativeAOT 中使用 RyuJIT 作为 IL 扫描器](#)

现在，编译器可以内联这些调用，从而减少了访问这些数据的指令。

<https://learn.microsoft.com/zh-cn/dotnet/core/whats-new/dotnet-9/overview#inlining-improvements-for-native-aot>

.NET 9 中 Native AOT 的突破

在.NET 9中，微软已经将Native AOT作为提升性能的关键点之一[2]。

此外，微软还宣布了对通用Windows平台（UWP）[1]的初步支持，允许开发者使用.NET 9和Native AOT技术来现代化改造现有的UWP应用。这一举措为UWP开发者提供了一条升级路径，使他们能够利用最新的.NET和Native AOT技术来改进其应用程序。

.NET 9的Native AOT不仅限于UWP平台，它还支持老旧的Windows 7和XP环境，这标志着AOT技术在兼容性方面的突破。然而，对于Android平台的Native AOT支持，目前尚未完成，尤其是JNI（Java Native Interface）支持，这被认为是一个较大的功能需求，还有 WPF/WinForms 的Native AOT支持也需要在 .NET 10 才能够完成。

[1]UWP 通过 .NET 9 和Native AOT 的支持实现 UWP 应用的现代化：

<https://mp.weixin.qq.com/s/lCgDOeaTuwmGagZWZG2AmQ>

[2]在.NET 9中，微软已经将Native AOT作为提升性能的关键点之一：

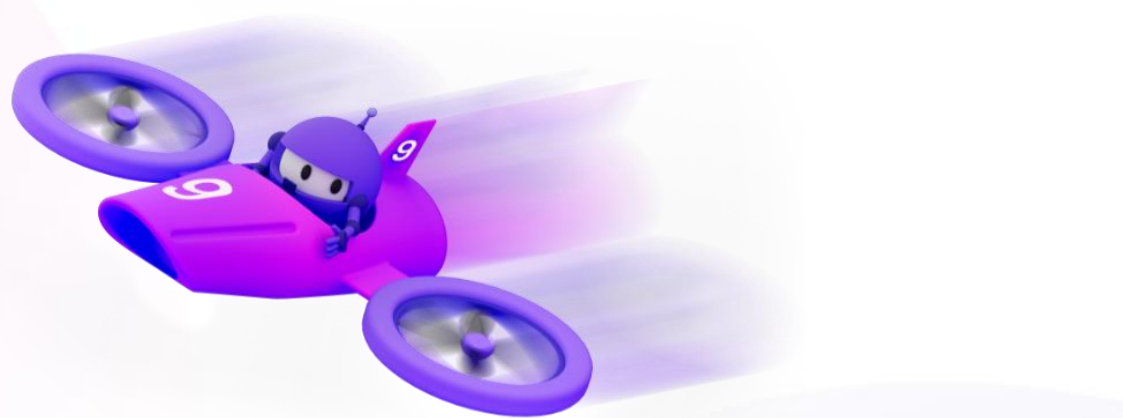
<https://www.cnblogs.com/shanyou/p/18015105>

Demo

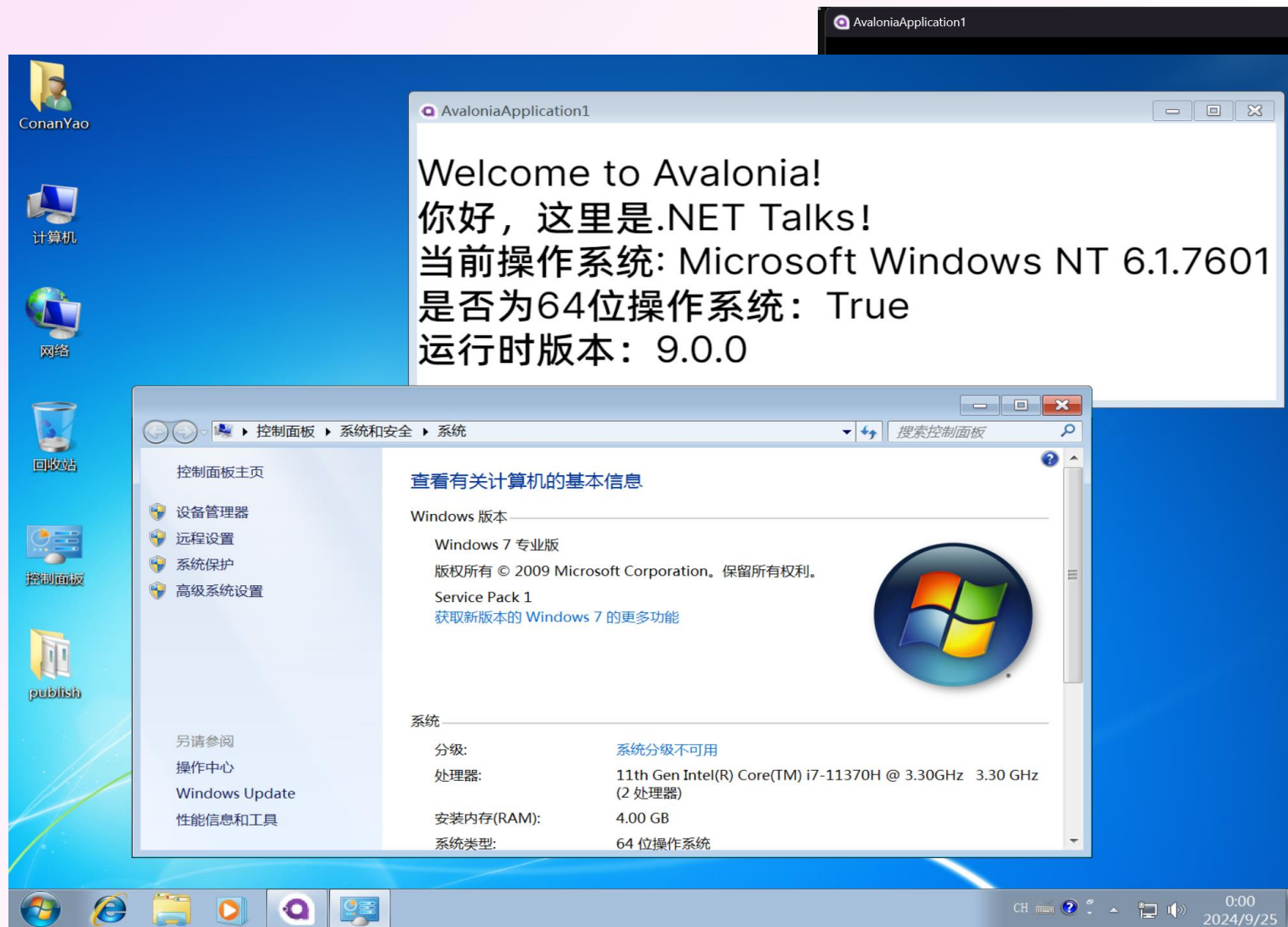
WinExe (Publish Native AOT) with Avalonia UI

Try Run On Windows 7

名称	修改日期	类型	大小
AvaloniaApplication1.Desktop.exe	2024/9/24 23:32	应用程序	97,220 KB
AvaloniaApplication1.DesktopAOT.exe	2024/9/24 15:33	应用程序	31,529 KB
AvaloniaApplication1.DesktopJIT.exe	2024/9/24 15:33	应用程序	31,529 KB



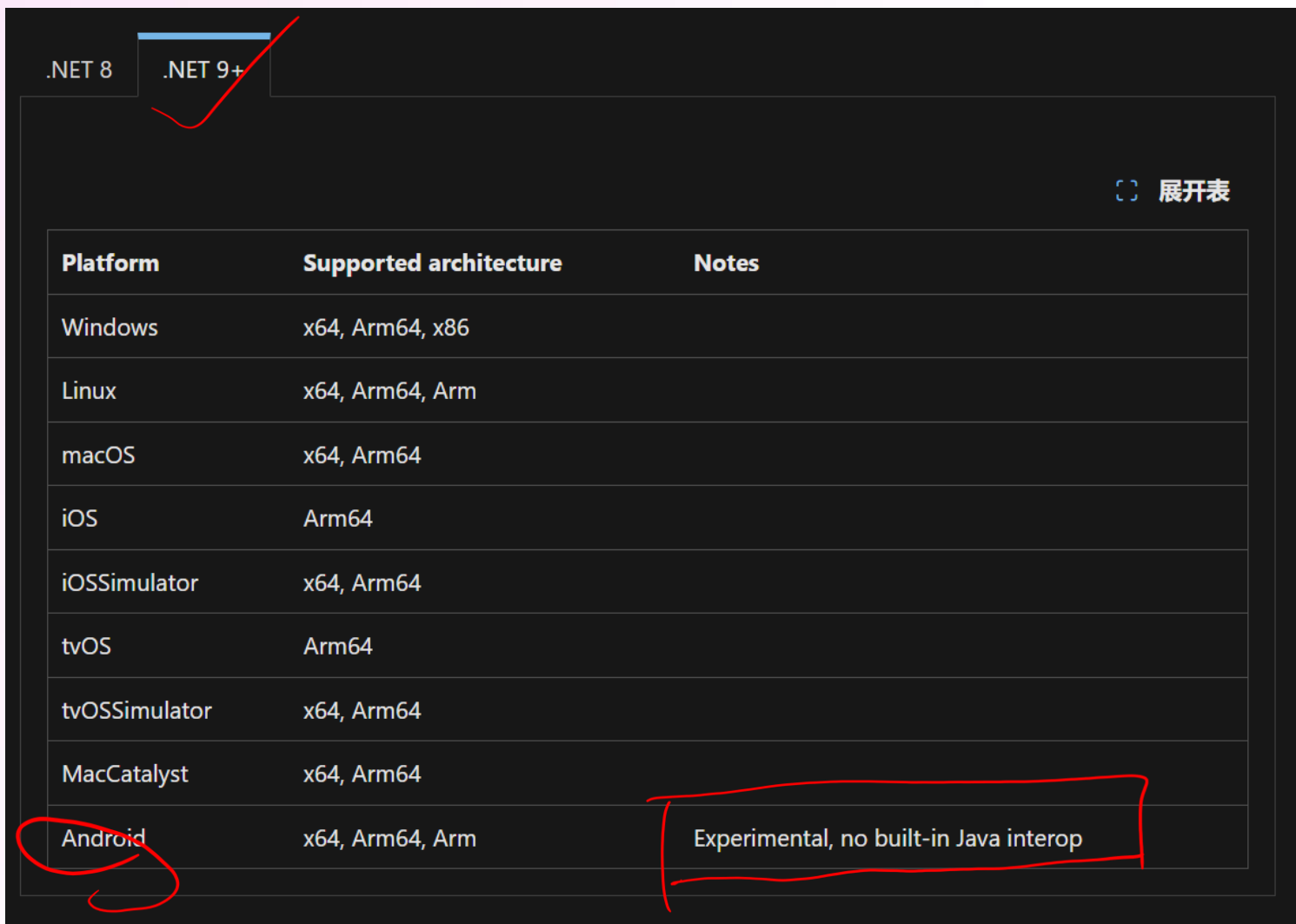
.NET 9 Native AOT 的突破



Windows 11

a!
alks!
soft Windows NT 10.0.22631.0
: True

.NET 9 Native AOT 的突破 平台/架构

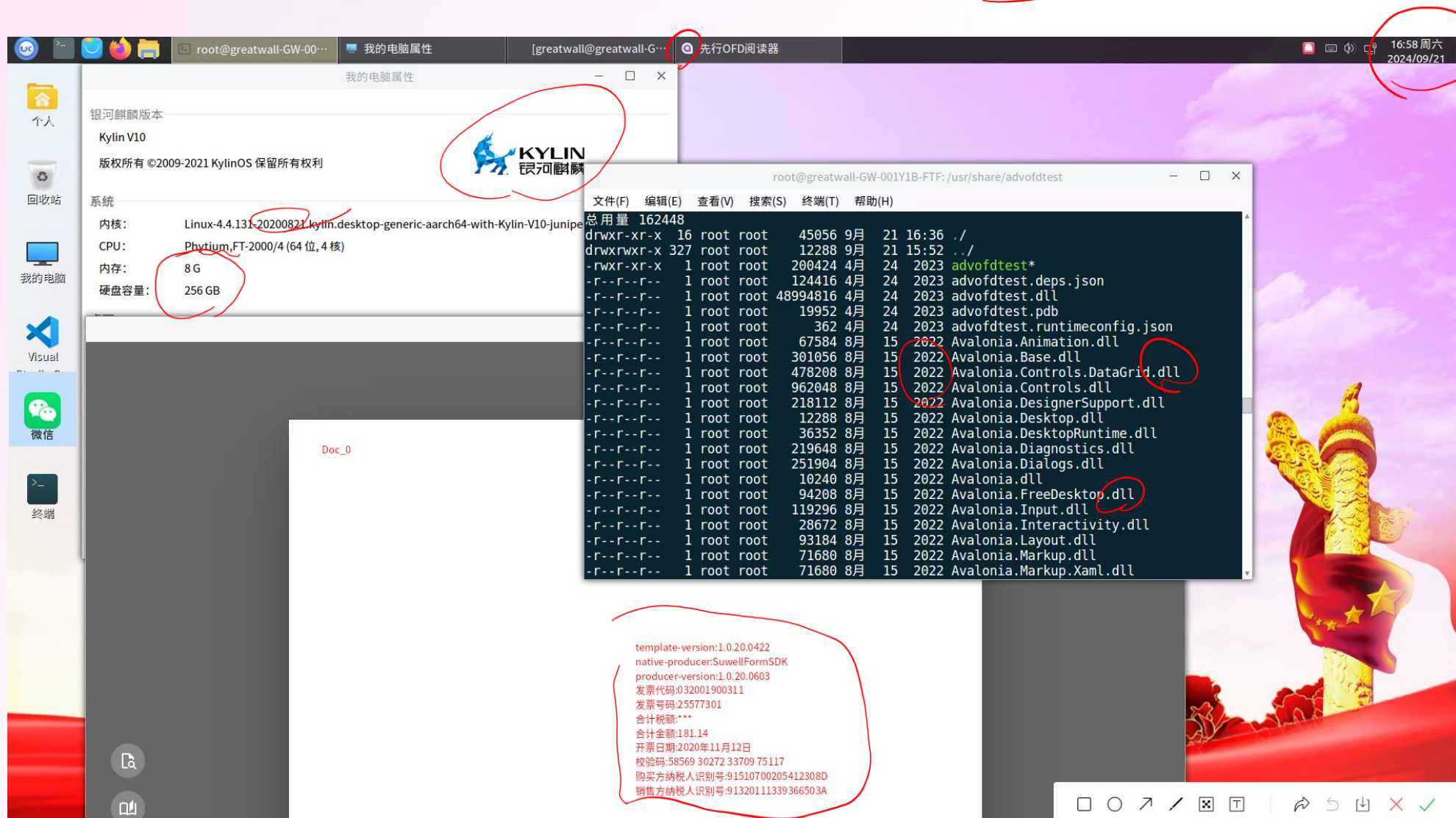


The screenshot shows a web interface for .NET 9 Native AOT. At the top, there are two tabs: ".NET 8" and ".NET 9+", with the latter selected and underlined in red. To the right of the tabs is a link "展开表" (Expand Table). Below the tabs is a table with three columns: "Platform", "Supported architecture", and "Notes". The table lists various platforms and their supported architectures. The "Android" row is circled in red, and its "Notes" cell, which says "Experimental, no built-in Java interop", is also enclosed in a red rectangle.

Platform	Supported architecture	Notes
Windows	x64, Arm64, x86	
Linux	x64, Arm64, Arm	
macOS	x64, Arm64	
iOS	Arm64	
iOSSimulator	x64, Arm64	
tvOS	Arm64	
tvOSSimulator	x64, Arm64	
MacCatalyst	x64, Arm64	
Android	x64, Arm64, Arm	Experimental, no built-in Java interop

<https://learn.microsoft.com/zh-cn/dotnet/core/deploying/native-aot>

信创 + .NET + Avalonia UI + 未来 Native AOT





.NET Conf 2024 .NET 9 在 .NET Conf 2024 上发布!在 11 月 12-14 日与 .NET 社区一起参与庆祝并了解新版本。

即将举办的活动

.NET Conf 2024

Celebrate and learn about what you can do with .NET 9 at the biggest .NET virtual event

November 12-14

.NET Conf 2024

2024年11月12日

<https://www.dotnetconf.net/>
<https://dotnet.microsoft.com/zh-cn/platform/community>



现在就下载 .NET 9 ！

[https://dotnet.microsoft.com/
zh-cn/download/dotnet/9.0](https://dotnet.microsoft.com/zh-cn/download/dotnet/9.0)

