# .NET 术语表

2019/01/22 •

**本文内容**


AOT

ASP.NET

ASP.NET Core

程序集 (assembly)

CLR

CoreCLR

CoreFX

CoreRT

跨平台

生态系统

框架

“GC”

IL

JIT

.NET 实现

库

元包

Mono

.NET

.NET Core

.NET Core CLI

.NET Core SDK

.NET Framework

.NET Native

.NET Standard

NGEN

包

平台

Runtime — 运行时

堆栈

Target Framework — 目标 Framework

TFM

UWP

请参阅

此术语表的主要目的是阐明所选术语和缩写词的含义，这些词频繁出现在 .NET 文档中但没有定义。

# AOT

预编译器。

与 JIT 类似，此编译器还可将 IL 转换为机器代码。 与 JIT 编译相比，AOT 编译在应用程序执行前进行并且通常在不同计算机上执行。 AOT 工具不会在运行时进行编译，因此它们不需要最大程度地减少编译所花费的时间。 这意味着它们可花更多的时间进行优化。 由于 AOT 的上下文是整个应用程序，因此 AOT 编译器还会执行跨模块链接和全程序分析，这意味着之后会进行所有引用并会生成单个可执行文件。

请参阅 CoreRT 和 .NET Native。

# ASP.NET

随 .NET Framework 一起提供的原始 ASP.NET 实现。

有时 ASP.NET 是一个涵盖性术语，指包含 ASP.NET Core 在内的两个 ASP.NET 实现。 该术语在任何给定实例中的含义取决于上下文。 若要指明不要使用 ASP.NET 表示这两种实现，请参考 ASP.NET 4.x。

请参阅 ASP.NET 文档。

# ASP.NET Core

.NET Core 上生成的跨平台、高性能、 开放源 ASP.NET 实现。

请参阅 ASP.NET Core 文档。

# 程序集 (assembly)

.dll /.exe 文件，其中包含一组可由应用程序或其他程序集调用的 API。

程序集可以包括接口、类、结构、枚举和委托等类型。 有时，项目的 bin 文件夹中的程序集被称为二进制文件 。 另请参阅库。

# CLR

公共语言运行时。

确切含义取决于上下文，但它通常指 .NET Framework 的运行时。 CLR 处理内存分配和管理。 CLR 也是一个虚拟机，不仅可执行应用，还可使用 JIT 编译器快速生成和编译代码。 当前的 Microsoft CLR 实现仅限 Windows。

# CoreCLR

.NET Core 公共语言运行时。

此 CLR 是采用与 CLR 相同的基本代码生成的。 最初，CoreCLR 是 Silverlight 的运行时，专为在多个平台（特别是 Windows 和 OS X）上运行而开发。CoreCLR 现属于 .NET Core 并表示 CLR 的简化版本。 它仍是跨平台运行时，现包括针对许多 Linux 分发的支持。 CoreCLR 也是具有 JIT 和代码执行功能的虚拟机。

# CoreFX

.NET Core 基类库 (BCL)

一组构成 System.*（在一定的程度上构成 Microsoft.*）命名空间的库。 BCL 是用于生成 ASP.NET Core 等较高级应用程序框架的较低级通用框架。 .NET Core BCL 的源代码包含在 .NET Core 运行时存储库中。 但大部分 .NET Core API 也可在 .NET Framework 中使用，因此可将 CoreFX 视为 .NET Framework BCL 的一个分支。

# CoreRT

.NET Core 运行时。

与 CLR/CoreCLR 相比，CoreRT 不是虚拟机，这意味着它不包含用于快速生成并运行代码的功能，因为它不包括 JIT。 但它包含 GC 以及运行时类型标识 (RTTI) 和反射功能。 只是由于设计有类型系统，因此并不需要元数据反射功能。 包含这些功能使它具有 AOT 工具链，该工具链可去除多余的元数据，更重要的是可识别应用不使用的代码。 CoreRT 正在开发中。

请参阅 .NET Native 和 CoreRT 简介。

# 跨平台

能够开发并执行可在多个不同操作系统（如 Linux、Windows 和 iOS）上使用的应用程序，而无需专门针对每个操作系统进行重新编写。 这样，可以在不同平台上的应用程序之间重复使用代码并保持一致性。

# 生态系统

所有针对给定技术生成和运行应用程序的运行时软件、开发工具和社区资源。

在所包含的第三方应用和库方面，术语“.NET 生态系统”不同于类似的“.NET 堆栈”等术语。 请看以下这一句示例：

- “推出 .NET Standard 的背后动机是要提高 .NET 生态系统中的一致性。”

# 框架

一般指一个综合 API 集合，便于开发和部署基于特定技术的应用程序。 从此常规意义上来说，ASP.NET Core 和 Windows 窗体都是示例应用程序框架。 另请参阅库。

“框架”一词在以下术语中有更具体的技术含义：

- .NET Framework
- 目标框架
- TFM（目标框架名字对象）

在现有的文档中，“框架”有时指 .NET 的实现。 例如，某文章可能会将 .NET Core 称为框架。 我们计划从文档中去掉这种令人困惑的用法。

# “GC”

垃圾回收器。

垃圾回收器是自动内存管理的实现。 GC 可释放对象占用的不再使用的内存。

请参阅垃圾回收。

# IL

中间语言。

C# 等较高级的 .NET 语言编译为称为中间语言 (IL) 的硬件无关性指令集。 IL 有时被称为 MSIL (Microsoft IL) 或 CIL（通用 IL）。

# JIT

实时编译器。

与 AOT 类似，此编译器将 IL 转换为处理器可理解的计算机代码。 与 AOT 不同，JIT 编译在需要运行代码的同一台计算机上按需执行。 由于 JIT 编译在应用程序的执行过程中发生，因此编译时是运行时的一部分。 因此，JIT 编译器需要平衡优化代码所花费的时间与生成代码时可节约的时间。 但 JIT 知道实际硬件，这样开发人员就无需提供不同的实现。

# .NET 实现

.NET 的实现包括以下项：

- 一个或多个运行时。 示例：CLR、CoreCLR、CoreRT。
- 实现 .NET Standard 的某版本并且可能包含其他 API 的类库。 示例：.NET Framework 基类库、.NET Core 基类库。
- 可选择包含一个或多个应用程序框架。 示例：ASP.NET、Windows 窗体和 WPF 包含在 .NET Framework 中。
- 可包含开发工具。 某些开发工具在多个实现之间共享。

.NET 实现的示例：

- .NET Framework
- .NET Core
- 通用 Windows 平台 (UWP)

# 库

可由应用或其他库调用的 API 集合。 .NET 库由一个或多个程序集组成。

词库和框架通常作同义词使用。

# 元包

一个 NuGet 包，没有自己的库，而只是一个依赖项列表。 所含包可选择建立目标框架的 API。

请参阅包、元包和框架

# Mono

Mono 是主要在需要小型运行时使用的开放源、跨平台 .NET 实现。 它是运行时，在 Android、Mac、iOS、tvOS 和 watchOS 上为 Xamarin 应用程序提供技术支持，主要针对内存占用少的应用程序。

它支持所有当前已发布的 .NET Standard 版本。

以前，Mono 实现更大的 .NET Framework API 并模拟一些 Unix 上最常用的功能。 有时使用它运行依赖 Unix 上的这些功能的 .NET 应用程序。

Mono 通常与实时编译器一起使用，但它也提供在 iOS 之类的平台使用的完整静态编译器（预先编译）。

若要了解有关 Mono 的详细信息，请参阅 Mono 文档。

# .NET

.NET Standard 和所有 .NET 实现及工作负荷的涵盖性术语。 始终采用大写形式，请勿使用“.Net”。

请参阅 .NET 指南

# .NET Core

一种跨平台、高性能的开放源 .NET 实现。 包括 Core 公共语言运行时 (CoreCLR)、Core AOT 运行时（正在开发的 CoreRT）、Core 基类库和 Core SDK。

请参阅 .NET Core。

# .NET Core CLI

用于开发 .NET Core 应用程序的跨平台工具链。

请参阅 .NET Core CLI。

# .NET Core SDK

一组库和工具，开发人员可用其创建 .NET Core 应用程序和库。 包括用于生成应用的 .NET Core CLI、用于生成和运行应用的 .NET Core 库以及运行 CLI 命令和运行应用程序的 dotnet 可执行文件 (dotnet.exe )。

请参阅 .NET Core SDK 概述。

# .NET Framework

仅在 Windows 上运行的 .NET 实现。 包括公共语言运行时 (CLR)、基类库和 ASP.NET、Windows 窗体和 WPF 之类的应用程序框架库。

请查阅 .NET Framework 指南。

# .NET Native

编译器工具链，可预先 (AOT) 生成，而非实时 (JIT) 生成本机代码。

编译采用与 C++ 编译器和链接器类似的工作方式在开发人员计算机上进行。 它删除了未使用的代码，留出更多时间进行优化。 它从库中提取代码，将它们合并到可执行文件中。 结果是表示整个应用的单个模块。

UWP 是 .NET Native 支持的首个应用程序框架。 现在，我们支持为 Windows、macOS 和 Linux 生成本机控制台应用。

请参阅 .NET Native 和 CoreRT 简介

# .NET Standard

在每个 .NET 实现中都可用的 .NET API 正式规范。

.NET Standard 规范有时被称为文档中的库。 由于库不仅包括规范（接口），还包括 API 实现，所以会误将 .NET Standard 称为“库”。 我们计划从本文档中去除该用法，引用 .NET Standard 元包 (`NETStandard.Library`) 的名称除外。

请参阅 .NET Standard。

# NGEN

本机（映像）生成。

可将此方法视为永久性 JIT 编译器。 它通常在执行代码的计算机上编译该代码，但通常在安装时进行编译。

# 包

NuGet 包 — 或只是一个包 — 是一个 .zip 文件，其中具有一个或多个名称相同的程序集以及作者姓名等其他元数据。

.zip 文件的扩展名为 .nupkg ，且可以包含在多个目标框架和版本中使用的资产（如 .dll 文件和 .xml 文件）。 在应用或库中安装时，会根据应用或库指定的目标框架选择相应的资产。 定义接口的资产位于 ref 文件夹，而定义实现的资产位于 lib 文件夹。

# 平台

操作系统以及运行它的硬件，例如 Windows、macOS、Linux、iOS 和 Android。

下面是在句子中使用的示例：

- ".NET Core 是一个跨平台 .NET 实现。"
- "PCL 配置文件代表 Microsoft 平台，而 .NET Standard 与平台无关。"

.NET 文档经常使用".NET 平台"表示一个 .NET 实现或包括所有实现的 .NET 堆栈。 这两种用法往往会与主（OS/硬件）含义混淆，因此我们计划从文档中去除这些用法。

# Runtime — 运行时

用于托管程序的执行环境。

操作系统属于运行时环境，但不属于 .NET 运行时。 下面是 .NET 运行时的一些示例：

- 公共语言运行时 (CLR)
- Core 公共语言运行时 (CoreCLR)
- .NET Native（适用于 UWP）
- Mono 运行时

.NET 文档有时使用"运行时"表示 .NET 的实现。 例如，在以下句子中应使用"实现"替换"运行时"：

- "各种 .NET 运行时实现特定版本的 .NET Standard。"
- "计划在多个运行时上运行的库应将此框架作为目标。"（参阅 .NET Standard）
- "各种 .NET 运行时实现特定版本的 .NET Standard。 ... 每个 .NET 运行时版本都将会公布它所支持的最高 .NET Standard 版本..."

我们计划消除此不一致的用法。

# 堆栈

一组编程方法，一起用于生成并运行应用程序。

".NET 堆栈"指 .NET Standard 和所有 .NET 实现。 短语"一个 .NET 堆栈"可能指一种 .NET 实现。

# Target Framework — 目标 Framework

.NET 应用或库依赖的 API 集合。

应用或库可将某版本的 .NET Standard（例如 .NET Standard 2.0）作为目标，这是所有 .NET 实现中一组标准化 API 的规范。 应用或库还能以特定 .NET 的某版本实现为目标，

这样便可获得特定于实现的 API 的访问权限。 例如，面向 Xamarin.iOS 的应用有权访问 Xamarin 提供的 iOS API 包装器。

对于某些目标框架（例如 .NET Framework），可用 API 由 .NET 实现在系统上安装的程序集定义，其中可能包括应用程序框架 API（例如 ASP.NET、WinForms）。 对于基于包的目标框架（例如 .NET Standard 和 .NET Core），框架 API 由安装在应用或库中的包定义。 在这种情况下，目标框架隐式指定一个元包，该元包引用一起构成框架的所有包。

请参阅目标框架。

## TFM

目标框架名字对象。

一个标准化令牌格式，用于指定 .NET 应用或库的目标框架。 目标框架通常由短名称（如 `net462`）引用。 存在长格式的 TFM（如 .NETFramework,Version=4.6.2），但通常不用来指定目标框架。

请参阅目标框架。

## UWP

通用 Windows 平台。

用于为物联网 (IoT) 生成新式触控 Windows 应用程序和软件的 .NET 实现。 它旨在统一可能想要以其为目标的不同类型的设备，包括电脑、平板电脑、平板手机、电话，甚至 Xbox。 UWP 提供许多服务，如集中式应用商店、执行环境 (AppContainer) 和一组 Windows API（用于代替 Win32 (WinRT)）。 应用可采用 C++、C#、Visual Basic 和 JavaScript 编写。 使用 C# 和 Visual Basic 时，.NET API 由 .NET Core 提供。

# 请参阅

- .NET 指南
- .NET Framework 指南
- .NET Core
- ASP.NET 概述
- 概述