

Contents

[Docs 概览](#)

[什么是 NuGet？](#)

[入门](#)

[安装 NuGet 客户端工具](#)

[安装并使用包 \(dotnet CLI\)](#)

[安装并使用包 \(Visual Studio\)](#)

[安装并使用包 \(Visual Studio for Mac\)](#)

[创建并发布 .NET Standard 包 \(dotnet CLI\)](#)

[创建并发布 .NET Standard 包 \(Visual Studio\)](#)

[创建并发布 .NET Framework 包 \(Visual Studio\)](#)

[使用包](#)

[概述和工作流](#)

[查找和选择包](#)

[安装和管理包](#)

[Visual Studio](#)

[Visual Studio for Mac](#)

[dotnet CLI](#)

[nuget.exe CLI](#)

[包管理器控制台 \(PowerShell\)](#)

[配置 NuGet](#)

[包还原选项](#)

[还原包](#)

[故障排除](#)

[重新安装和更新包](#)

[管理全局包和缓存文件夹](#)

[管理包信任边界](#)

[使用已通过身份验证的源](#)

[使用源代码管理系统](#)

[常见的 NuGet 配置](#)

在项目中引用包

[项目文件中的包引用](#)

[将 packages.config 迁移到 PackageReference](#)

[packages.config](#)

创建包

[概述和工作流](#)

[创建包 \(dotnet CLI\)](#)

[创建包 \(nuget.exe CLI\)](#)

[创建包 \(MSBuild\)](#)

[支持项目文件中包含多个目标框架](#)

[构建预发行包](#)

[创建符号包](#)

[高级任务](#)

[支持多个目标框架](#)

[修改源代码和配置文件](#)

[选择项目引用的程序集](#)

[设置包类型](#)

[创建本地化的包](#)

[针对特定内容的指南](#)

[创建 UWP 包 \(C++\)](#)

[创建 UWP 包 \(C#\)](#)

[创建本机包](#)

[以 NuGet 包的形式创建 UI 控件](#)

[以 NuGet 包的形式创建分析器](#)

[使用 Visual Studio 2017 或 2019 为 Xamarin 创建包](#)

[使用 COM 互操作程序集创建包](#)

签名包

[对包进行签名](#)

[已签名的包签名和要求](#)

发布包

[发布到 NuGet.org](#)

[发布包](#)

[API 密钥](#)

[发布到私有源](#)

[概述](#)

[Azure Artifacts](#)

[NuGet.Server](#)

[本地源](#)

[概念](#)

[包安装过程](#)

[包版本控制](#)

[依赖项解析](#)

[参考](#)

[.nuspec](#)

[nuget.config 文件](#)

[目标框架](#)

[作为 MSBuild 目标的包和还原](#)

[dotnet CLI](#)

[nuget.exe CLI 参考](#)

[add](#)

[config](#)

[delete](#)

[help or ?](#)

[init](#)

[安装](#)

[list](#)

[locals](#)

[mirror](#)

[pack](#)

[push](#)

[还原 \(restore\)](#)

[setapikey](#)

[sign](#)

[sources](#)

[spec](#)
[update](#)
[verify](#)

[信任的签名者](#)

[环境变量](#)

[长路径支持](#)

[PowerShell 参考](#)

[Add-BindingRedirect](#)
[Find-Package](#)
[Get-Package](#)
[Get-Project](#)
[Install-Package](#)
[Open-PackagePage](#)
[Sync-Package](#)
[Uninstall-Package](#)
[Update-Package](#)

[NuGet 服务器 API](#)

[概述](#)
[资源](#)

[自动完成](#)
[目录](#)
[包内容](#)
[包详细信息 URL](#)
[包元数据](#)
[推送和删除](#)
[推送符号包](#)
[报告滥用 URL](#)
[存储库签名](#)
[搜索](#)
[服务索引](#)

[如何:使用 API 查询所有包](#)
[速率限制](#)

[nuget.org 协议](#)

[tools.json](#)

[NuGet 客户端 SDK](#)

[错误和警告](#)

[NU1000](#)

[NU1001](#)

[NU1002](#)

[NU1003](#)

[NU1100](#)

[NU1101](#)

[NU1102](#)

[NU1103](#)

[NU1104](#)

[NU1105](#)

[NU1106](#)

[NU1107](#)

[NU1108](#)

[NU1201](#)

[NU1202](#)

[NU1203](#)

[NU1401](#)

[NU1500](#)

[NU1501](#)

[NU1502](#)

[NU1503](#)

[NU1601](#)

[NU1602](#)

[NU1603](#)

[NU1604](#)

[NU1605](#)

[NU1608](#)

[NU1701](#)

NU1801

NU3000

NU3001

NU3002

NU3003

NU3004

NU3005

NU3006

NU3007

NU3008

NU3009

NU3010

NU3011

NU3012

NU3013

NU3014

NU3015

NU3016

NU3017

NU3018

NU3019

NU3020

NU3021

NU3022

NU3023

NU3024

NU3025

NU3026

NU3027

NU3028

NU3029

NU3030

NU3031

NU3032

NU3033

NU3034

NU3035

NU3036

NU3037

NU3038

NU3040

NU5000

NU5001

NU5002

NU5003

NU5004

NU5005

NU5007

NU5008

NU5009

NU5010

NU5011

NU5012

NU5013

NU5014

NU5015

NU5016

NU5017

NU5018

NU5019

NU5020

NU5021

NU5022

NU5023

NU5024

NU5025

NU5026

NU5027

NU5028

NU5029

NU5030

NU5031

NU5032

NU5033

NU5034

NU5035

NU5036

NU5037

NU5046

NU5047

NU5048

NU5100

NU5101

NU5102

NU5103

NU5104

NU5105

NU5106

NU5107

NU5108

NU5109

NU5110

NU5111

NU5112

NU5114

NU5115

[NU5116](#)

[NU5117](#)

[NU5118](#)

[NU5119](#)

[NU5120](#)

[NU5121](#)

[NU5122](#)

[NU5123](#)

[NU5124](#)

[NU5125](#)

[NU5127](#)

[NU5128](#)

[NU5129](#)

[NU5130](#)

[NU5131](#)

[NU5500](#)

存档内容

[project.json 管理格式](#)

[project.json 和 UWP](#)

[project.json impact](#)

扩展性

[可扩展性 - NuGet 插件](#)

[NuGet 跨平台插件](#)

[NuGet 跨平台身份验证插件](#)

[适用于 Visual Studio 的 NuGet 凭据提供程序](#)

[nuget.exe 凭据提供程序](#)

[Visual Studio 扩展性](#)

[Visual Studio 中的 NuGet API](#)

[项目系统支持](#)

[Visual Studio 模板](#)

资源

[策略](#)

[治理](#)

[生态系统](#)

[NuGet.org 策略](#)

[发行说明](#)

[已知问题](#)

[NuGet 5.x](#)

[NuGet 5.5](#)

[NuGet 5.4](#)

[NuGet 5.3](#)

[NuGet 5.2](#)

[NuGet 5.1](#)

[NuGet 5.0](#)

[NuGet 4.x](#)

[NuGet 4.9 RTM](#)

[NuGet 4.8 RTM](#)

[NuGet 4.7 RTM](#)

[NuGet 4.6 RTM](#)

[NuGet 4.5 RTM](#)

[NuGet 4.4 RTM](#)

[NuGet 4.3 RTM](#)

[NuGet 4.0 RTM](#)

[NuGet 4.0 RC](#)

[NuGet 3.x](#)

[NuGet 3.5 RTM](#)

[NuGet 3.5 RC](#)

[NuGet 3.5 Beta2](#)

[NuGet 3.5 Beta](#)

[NuGet 3.4.4](#)

[NuGet 3.4.3](#)

[NuGet 3.4.2](#)

[NuGet 3.4.1](#)

[NuGet 3.4](#)

[NuGet 3.4 RC](#)

[NuGet 3.3](#)

[NuGet 3.2.1](#)

[NuGet 3.2](#)

[NuGet 3.2 RC](#)

[NuGet 3.1.1](#)

[NuGet 3.1](#)

[NuGet 3.0.0](#)

[NuGet 3.0 RC2](#)

[NuGet 3.0 RC](#)

[NuGet 3.0 Beta](#)

[NuGet 3.0 Preview](#)

[NuGet 2.x](#)

[NuGet 2.12](#)

[NuGet 2.12 RC](#)

[NuGet 2.9 RC](#)

[NuGet 2.8.7](#)

[NuGet 2.8.6](#)

[NuGet 2.8.5](#)

[NuGet 2.8.3](#)

[NuGet 2.8.2](#)

[NuGet 2.8.1](#)

[NuGet 2.8](#)

[NuGet 2.7.2](#)

[NuGet 2.7.1](#)

[NuGet 2.7](#)

[NuGet 2.6.1-for-WebMatrix](#)

[NuGet 2.6](#)

[NuGet 2.5](#)

[NuGet 2.2.1](#)

[NuGet 2.2](#)

[NuGet 2.1](#)

[NuGet 2.0](#)

[NuGet 1.x](#)

[NuGet 1.8](#)

[NuGet 1.7](#)

[NuGet 1.6](#)

[NuGet 1.5](#)

[NuGet 1.4](#)

[NuGet 1.3](#)

[NuGet 1.2](#)

[NuGet 1.1](#)

[常见问题](#)

[项目格式](#)

[NuGet.org](#)

NuGet 简介

2020/4/8 • [Edit Online](#)

适用于任何现代开发平台的基本工具可充当一种机制，通过这种机制，开发人员可以创建、共享和使用有用的代码。通常，此类代码捆绑到“包”中，其中包含编译的代码（如 DLL）以及在使用这些包的项目中所需的其他内容。

对于 .NET（包括 .NET Core），共享代码的 Microsoft 支持的机制则为 NuGet，其定义如何创建、托管和使用面向 .NET 的包，并针对每个角色提供适用工具。

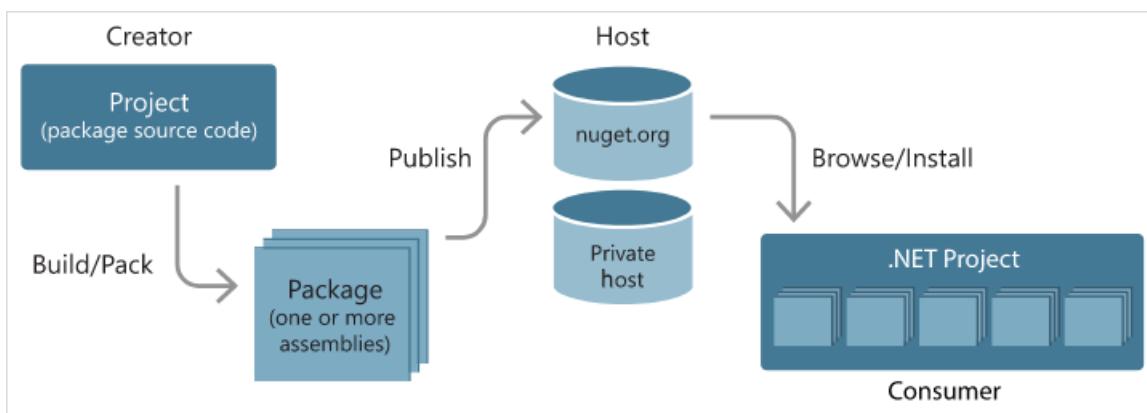
简单来说，NuGet 包是具有 `.nupkg` 扩展的单个 ZIP 文件，此扩展包含编译代码（DLL）、与该代码相关的其他文件以及描述性清单（包含包版本号等信息）。使用代码的开发人员共享创建包，并将其发布到公用或专用主机。包使用者从适合的主机获取这些包，将它们添加到项目，然后在其项目代码中调用包的功能。随后，NuGet 自身负责处理所有中间详细信息。

由于 NuGet 支持公用 nuget.org 主机旁边的专用主机，因此，可以使用 NuGet 包来共享组织或工作组专用的代码。此外，你还可以使用 NuGet 包作为一种便捷的方式，将自己的代码用于除你自己项目之外的任何其他项目。简而言之，NuGet 包是可共享的代码单元，但不需要暗示任何特定的共享方式。

包在创建者、主机和使用者之间的流

作为公用主机角色时，NuGet 自身负责在 [nuget.org](#) 中维护包含 100,000 多个唯一包的中央存储库。这些包每天供数以百万的 .NET/.Net Core 开发人员使用。NuGet 还支持在云中（如在 Azure DevOps 上）、在私有网络中或者甚至直接在本地文件系统以私密方式托管包。通过这样做，这些程序包仅对那些有权访问主机的开发人员可用，使你能够将程序包提供给特定的一组用户。[托管自己的 NuGet 源](#)中提供了对相关选项的说明。通过配置选项，你还可以精确控制任何给定计算机可以访问的主机，从而确保程序包是从特定源（而不是像 nuget.org 这样的公用存储库）获取的。

无论主机的本质是什么，它都可作为包创建者和包使用者之间的连接点。创建者生成有用的 NuGet 包并将其发布到主机。然后，使用者可以在可访问的主机上搜索有用且兼容的包，下载包并将其包含在项目中。在项目中安装包后，包的 API 将可用于其余项目代码。



包定向兼容性

“兼容”包指：此包所包含的程序集应至少针对与使用项目的目标框架兼容的一个目标 .NET Framework 而生成。与 UWP 控件一样，开发人员可以创建特定于一个框架的程序包，也可以支持更广泛的目标。为了最大限度地利用程序包的兼容性，开发人员的目标是所有 .NET 和 .NET Core 项目都可以使用的 [.NET Standard](#)。对于创建者和使用者而言，这是最有效的方式，因为单个包（通常包含单个程序集）适用于所有使用项目。

另一方面，需要 .NET Standard 之外的 API 的程序包开发人员会为他们希望支持的不同目标框架创建单独的程序集，并将所有这些程序集包含在同一个程序包中（称为“多目标”）。使用者安装此类包时，NuGet 将仅提取项目需要

的程序集。这能将包在该项目生成的最终应用程序和/或程序集中的占用量降到最低。当然，多目标包对创建者来说更难维护。

NOTE

目标 .NET Standard 取代了以前使用各种可移植类库 (PCL) 目标的方法。因此，此文档着重于为 .NET Standard 创建程序包。

NuGet 工具

除托管支持外，NuGet 还提供各种供创建者和使用者使用的工具。有关如何获取特定工具的信息，请参阅[安装 NuGet 客户端工具](#)。

工具	支持	功能	说明
dotnet CLI	全部	创建、使用	用于 .NET Core 和 .NET Standard 库，以及用于面向 .NET Framework 的 SDK 样式项目的 CLI 工具(请参阅 SDK 属性)。直接在 .NET Core 工具链中提供特定 NuGet CLI 功能。与 <code>nuget.exe</code> CLI 一样， <code>dotnet</code> CLI 不会与 Visual Studio 项目交互。
nuget.exe CLI	全部	创建、使用	用于 .NET Framework 库和面向 .NET Standard 库的非 SDK 样式项目的 CLI 工具。提供所有 NuGet 功能，包括一些专门适用于包创建者、仅适用于使用者和适用于两者的命令。例如，包创建者使用 <code>nuget pack</code> 命令通过各种程序集和相关文件创建包，包使用者使用 <code>nuget install</code> 在项目文件夹中包含包，而所有人都可使用 <code>nuget config</code> 设置 NuGet 配置变量。作为与平台无关的工具，NuGet CLI 不会与 Visual Studio 项目交互。
包管理器控制台	Windows 版 Visual Studio	使用	提供用于在 Visual Studio 项目中安装和管理包的 PowerShell 命令 。
包管理器 UI	Windows 版 Visual Studio	使用	提供用于在 Visual Studio 项目中安装和管理包的易用 UI。
管理 NuGet UI	Visual Studio for Mac	使用	提供用于在 Visual Studio for Mac 项目中安装和管理包的易用 UI。
MSBuild	Windows	创建、使用	支持创建包和还原项目中直接通过 MSBuild 工具链使用的包。

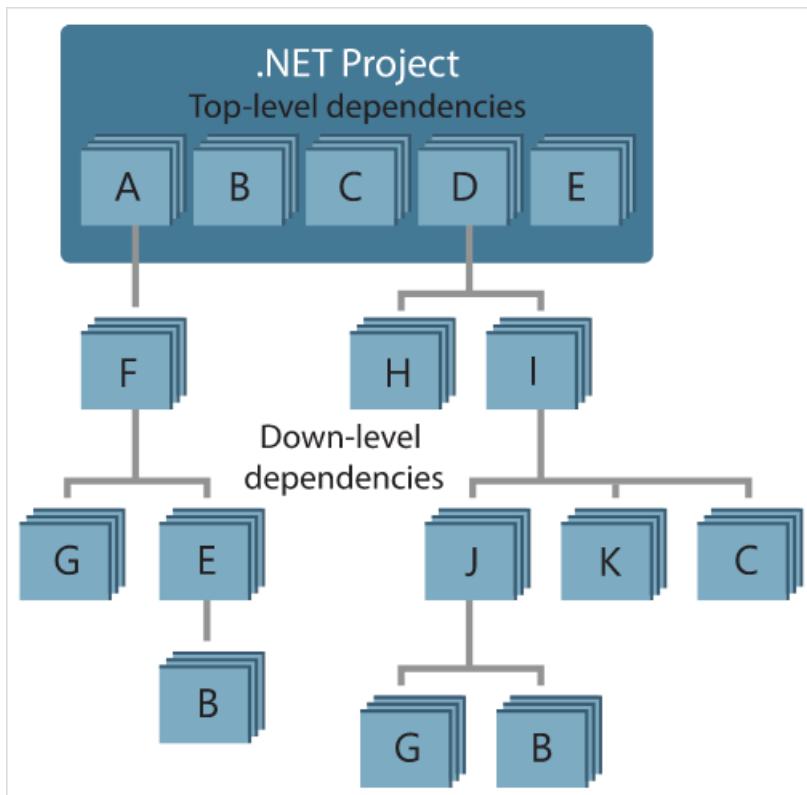
如上文所述，你使用的 NuGet 工具很大程度上取决于用户要创建、使用还是发布程序包以及你所使用的平台。包创建者通常也是使用者，因为他们以其他 NuGet 包中已存在的功能作为生成基础。当然，这些包反过来可能也需要依赖其他包。

有关详细信息，请从[包创建工作流](#)和[包使用工作流](#)文章开始。

管理依赖项

在其他人的工作基础上轻松生成，这是使程序包管理系统成为最强大功能的方法之一。相应地，大部分 NuGet 的用途就是代表项目管理该依赖关系树或“关系图”。简单来说，你仅需要关注在项目中直接使用的包。如果任何这些包本身使用其他包（这些包仍可以使用其他包），NuGet 将负责所有这些下层依赖项。

下图显示一个依赖于五个包的项目，这些包反过来也依赖于许多其他包。



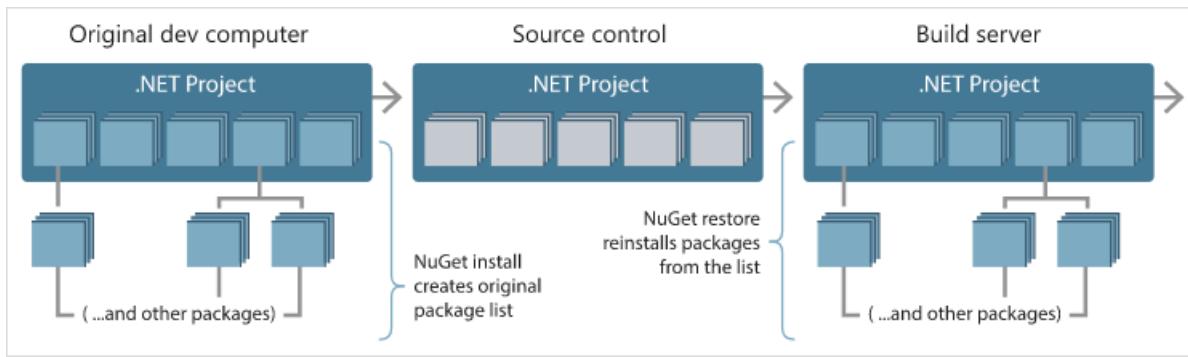
请注意，某些包在依赖项关系图中多次出现。例如，包 B 有三个不同的使用者，并且每个使用者可能为该包指定不同版本（未显示）。这是一种常见情况，特别是对于广泛使用的程序包。幸运的是，NuGet 将执行所有困难的工作来确定包 B 的哪一个版本可满足所有使用者。随后，无论依赖项关系图多么复杂，NuGet 都将对所有包执行相同操作。

有关 NuGet 如何实现此服务的更多详细信息，请参阅[依赖项解析](#)。

跟踪引用和还原包

项目可在开发人员计算机、源代码管理存储库、生成服务器等位置之间轻松移动，因此将 NuGet 包的二进制程序集直接绑定到项目非常不切实际。这样做不仅会使项目的每个副本出现不必要的膨胀（而且会浪费源代码管理存储库中的空间）。还会使包二进制文件难以更新到新版本（因为这需要更新项目的所有副本）。

而 NuGet 维护一个项目所依赖的包的简单引用列表，包括顶层和下层的依赖关系。也就是说，每当你将某个主机中的包安装到项目中时，NuGet 都将在此引用列表中记录包标识符和版本号。（当然，卸载包将从列表中删除此信息。）然后，NuGet 根据请求提供还原所有引用程序包的方法，如[软件包还原](#)中所述。



只需引用列表，NuGet 随后即可随时从公共和/或私有主机重新安装—即“还原”—所有这些包。将项目提交到源代码管理存储库或将其以其他方式进行共享时，只需包含引用列表，不需要包含任何包二进制文件（请参阅[包和源代码管理](#)）。

接收项目的计算机（如获得项目副本并将其作为自动部署系统的一部分的生成服务器）仅会在需要时要求 NuGet 还原依赖项。Azure DevOps 等生成系统会出于此确切目的提供“NuGet 还原”步骤。同样，当开发人员获取项目副本（如克隆存储库时），他们可以调用 `nuget restore` (NuGet CLI)、`dotnet restore` (dotnet CLI) 或 `Install-Package` (程序包管理器控制台) 类似的命令，以获得所有必要的程序包。对于 Visual Studio 来说，它将在生成项目时自动还原包（前提是启用了自动还原，如[包还原](#)中所述）。

显然，开发人员接下来关注的 NuGet 的主要角色则是代表项目维护该引用列表并提供高效还原（和更新）这些引用包的方法。该列表以两种“包管理格式”中的一种维护，因为将它们称为：

- [PackageReference](#)（或“项目文件中的包引用”）| (*NuGet 4.0+*) 维护直接位于项目文件中的项目顶层依赖项的列表，因此无需单独文件。关联文件 `obj/project.assets.json` 动态生成，以管理项目使用的包的总依赖项关系图以及所有下层依赖项。PackageReference 始终由 .NET Core 项目使用。
- [packages.config](#) : (*NuGet 1.0+*) 一种 XML 文件，用于维护项目中所有依赖项的简单列表，包括其他已安装包的依赖项。已安装或已还原的包存储在 `packages` 文件夹中。

任何特定项目中所用的包管理格式取决于项目类型以及 NuGet（和/或 Visual Studio）的可用版本。若要确认当前使用的格式，只需在安装第一个包后在项目根目录中查找 `packages.config`。如果没有该文件，请直接在项目文件中查找 `<PackageReference>` 元素。

当进行选择时，我们建议使用 PackageReference。出于与旧版兼容目的对 `packages.config` 进行维护，将不再主动对其进行开发。

TIP

各种 `nuget.exe` CLI 命令（如 `nuget install`）不会自动将包添加到引用列表中。当使用 Visual Studio 包管理器（UI 或控制台）并使用 `dotnet.exe` CLI 时，将更新此列表。

NuGet 的其他功能

到目前为止，你已经学习了 NuGet 的以下特征：

- NuGet 提供支持专用托管的中心 `nuget.org` 存储库。
- NuGet 为开发人员提供创建、发布和使用包所需的工具。
- 最重要的是，NuGet 能维护项目中所用包的引用列表，并且能够通过该列表还原和更新这些包。

为使这些进程高效运行，NuGet 执行了一些后台优化。最值得注意的是，NuGet 管理包缓存和全局包文件夹，使安装和重新安装过程更为快捷。缓存可避免下载已在计算机上安装的包。全局包文件夹允许多个项目共享同一个已安装的包，因此减少了计算机上的 NuGet 的总体占用。当在生成服务器等位置频繁还原大量包时，缓存和全局包文件夹也非常有帮助。有关这些机制的详细信息，请参阅[管理全局包和缓存文件夹](#)。

在一个单独的项目中，NuGet 管理整个依赖项关系图，它同样包括解析对同一个包的不同版本的多个引用。项目在具有相同依赖项的一个或多个包上选取依赖项是很常见的情况。nuget.org 上的某些最有用的实用程序包即由其他许多包使用。然后在整个依赖项关系图中，你可以对同一个包的不同版本轻松发起 10 种不同的引用。为避免将该包的多个版本引入应用程序本身，NuGet 会挑选出一个适合所有使用者的版本。（有关详细信息，请参阅[依赖项解析](#)。）

除此之外，NuGet 会维护与包的构造方式（包括[本地化](#)和[调试符号](#)）以及[引用方式](#)（包括[版本范围](#)和[预发行版本](#)）相关的所有规范。此外，NuGet 还提供了各种 API 以编程方式使用其服务，并可为编写 Visual Studio 扩展和项目模板的开发人员提供支持。

请花一点时间浏览本文档的目录，你会看到其中列出了所有这些功能，以及自 NuGet 首次发行起的发行说明。

相关视频

在[第 9 频道](#)和 [YouTube](#) 上查找更多 NuGet 视频。

评论、建议和问题

最后，我们非常欢迎针对此文档给出评论和建议 — 只需在任何页面上选择“反馈”和“编辑”命令，或访问 GitHub 上的[文档存储库](#)和[文档问题列表](#)。

我们同样欢迎通过[多种 GitHub 存储库](#)针对 NuGet 本身提出建议；有关 NuGet 的问题，请访问
<https://github.com/NuGet/home/issues>。

请尽情享受 NuGet 体验！

安装 NuGet 客户端工具

2020/4/8 • [Edit Online](#)

打算安装包？请参阅[安装 NuGet 包的方式](#)

要使用 NuGet，作为软件包使用者或创建者，可以使用命令行接口 (CLI) 工具以及 Visual Studio 中的 NuGet 功能。本文简要介绍了不同工具的功能，如何安装它们，以及它们功能可用性的相对优势。若要开始借助 NuGet 来使用包，请参阅[安装和使用包 \(dotnet CLI\)](#) 以及[安装和使用包 \(Visual Studio\)](#)。若要开始创建 NuGet 包，请参阅[创建和发布 .NET Standard 包 \(dotnet CLI\)](#) 以及[创建和发布 .NET Standard 包 \(Visual Studio\)](#)。

工具	功能	相关链接
dotnet.exe	适用于 .NET Core 和 .NET Standard 库，以及适用于任何 SDK 样式项目 （例如面向 .NET Framework 的项目）的 CLI 工具。包含在 .NET Core SDK 中，并在所有平台上提供核心 NuGet 功能。（从 Visual Studio 2017 开始，dotnet CLI 将自动随任何与 .NET Core 相关的工作负载一起安装。）	.NET Core SDK
nuget.exe	适用于 .NET Framework 库，以及适用于任何 非 SDK 样式项目 （例如面向 .NET Standard 库的项目）的 CLI 工具。提供 Windows 上的所有 NuGet 功能以及 Mac 和 Linux 上在 Mono 下运行时的大多数功能。	nuget.exe
Visual Studio	对于 Windows，Visual Studio 2012 及更高版本中都包括“NuGet 包管理器”。Visual Studio 提供 包管理器 UI 和 包管理器控制台 ，通过它可以运行大部分的 NuGet 操作。	Visual Studio
Visual Studio for Mac	对于 Mac，特定 NuGet 功能是直接内置的。包管理器控制台目前不可用。对于其他功能，请使用 dotnet.exe 或 nuget.exe CLI 工具。	Visual Studio for Mac
Visual Studio Code	对于 Windows、Mac 或 Linux，NuGet 功能可通过市场扩展提供，或者使用 dotnet.exe 或 nuget.exe CLI 工具。	Visual Studio Code

[MSBuild CLI](#) 还提供了还原和创建包的功能，该功能主要在生成服务器上使用。MSBuild 不是与 NuGet 一起使用的通用工具。

程序包管理器控制台命令只能在 Windows 的 Visual Studio 中工作，不能在其他 PowerShell 环境中工作。

Visual Studio

在 Visual Studio 2017 及更高版本上安装

从 Visual Studio 2017 开始，安装程序包括具有任何采用 .NET 的工作负载的 NuGet 包管理器。若要单独安装，

或验证是否已安装包管理器，运行 Visual Studio 安装程序，并检查“各个组件”>“代码工具”>“NuGet 包管理器”下的选项。

在 Visual Studio 2015 及更低版本上安装

适用于 Visual Studio 2013 和 2015 的 NuGet 扩展可以从 <https://dist.nuget.org/index.html> 下载。

对于 Visual Studio 2010 及更早版本，请安装“适用于 Visual Studio 的 NuGet 包管理器”扩展。请注意，如果在搜索结果的首页看不到该扩展，请尝试将“排序依据”下拉菜单改为“下载次数最多”或按字母顺序进行排序。

CLI 工具

可以使用 `dotnet` CLI 或 `nuget.exe` CLI 支持 IDE 中的 NuGet 功能。`dotnet` CLI 随某些 Visual Studio 工作负载一起安装，例如 .NET Core。如前面所述，必须单独安装 `nuget.exe` CLI。

两个 NuGet CLI 工具是 `dotnet.exe` 和 `nuget.exe`。请参阅[功能可用性](#)以进行比较。

- 若要面向 .NET Core 或 .NET Standard，请使用 `dotnet` CLI。`dotnet` CLI 是 SDK 样式项目格式所必需的，该格式使用 [SDK 属性](#)。
- 要面向 .NET Framework(仅限非 SDK 样式项目)，请使用 `nuget.exe` CLI。如果项目从 `packages.config` 迁移到 `PackageReference`，请使用 `dotnet` CLI。

dotnet.exe CLI

.NET Core 2.0 CLI `dotnet.exe` 适用于所有平台(Windows、Mac 和 Linux)，并提供核心的 NuGet 功能，例如安装、还原和发布程序包。`dotnet` 提供了与 .NET Core 项目文件(如 `.csproj`)的直接集成，这在大多数情况下都很有用。此外，`dotnet` 是直接为每个平台构建的，不需要你安装 Mono。

安装：

- 在开发人员计算机上，请安装 [.NET Core SDK](#)。从 Visual Studio 2017 开始，`dotnet` CLI 将自动随任何与 .NET Core 相关的工作负载一起安装。
- 对于生成服务器，请按照[在持续集成 \(CI\) 中使用 .NET Core SDK 和工具](#)中的说明进行操作。

要了解如何在 `dotnet` CLI 中使用基本命令，请参阅[使用 dotnet CLI 安装并使用包](#)。

nuget.exe CLI

`nuget.exe` CLI(即 `nuget.exe`)是适用于 Windows 的命令行实用工具，可提供所有 NuGet 功能；它也可以使用存在一些限制的 [Mono](#) 在 Mac OSX 和 Linux 上运行。

要了解如何在 `nuget.exe` CLI 中使用基本命令，请参阅[使用 nuget.exe CLI 安装并使用包](#)。

安装：

Windows

NOTE

NuGet.exe 5.0 及更高版本需要 .NET Framework 4.7.2 或更高版本才能执行。

- 请访问 nuget.org/downloads，并选择 NuGet 3.3 或更高版本(2.8.6 与 Mono 不兼容)。始终建议使用最新版。若要将包发布到 nuget.org，版本至少必须是 4.1.0。
- 每次下载都直接下载 `nuget.exe` 文件。让浏览器将文件保存到选定文件夹。此文件不是安装程序；如果直接在浏览器中运行，就不会看到任何内容。
- 将文件夹添加到 `nuget.exe` 中放置 PATH 环境变量的位置，这样就可以从任意位置使用 CLI 工具。

macOS/Linux

行为可能因 OS 分发版本略有不同。

1. 安装 Mono 4.4.2 或更高版本。

2. 在 shell 提示符处, 执行下列命令:

```
# Download the latest stable `nuget.exe` to `/usr/local/bin`  
sudo curl -o /usr/local/bin/nuget.exe https://dist.nuget.org/win-x86-commandline/latest/nuget.exe
```

3. 通过将以下脚本添加到 OS 的相应文件来创建别名(通常为 `~/.bash_aliases` 或 `~/.bash_profile`):

```
# Create alias for nuget  
alias nuget="mono /usr/local/bin/nuget.exe"
```

4. 重载 shell。通过输入 `nuget` (而不使用任何参数) 来测试安装。应该会看到 NuGet CLI 帮助。

TIP

在 Windows 上运行 `nuget update -self` 可以将现有 `nuget.exe` 更新为最新版本。

NOTE

<https://dist.nuget.org/win-x86-commandline/latest/nuget.exe> 中始终提供推荐的最新 NuGet CLI。为了实现与旧版持续集成系统的兼容性, 以前的 URL <https://nuget.org/nuget.exe> 当前提供弃用的 2.8.6 CLI 工具。

功能可用性

	DOTNET CLI	NUGET CLI (WINDOWS)	NUGET CLI (MONO)	VISUAL STUDIO (WINDOWS)	VISUAL STUDIO FOR MAC
搜索包		✓	✓	✓	✓
安装/卸载包	✓	✓(1)	✓	✓	✓
更新包	✓	✓		✓	✓
还原包	✓	✓	✓(2)	✓	✓
管理包源(来源)		✓	✓	✓	✓
在源上管理包	✓	✓	✓		
设置源的 API 密钥		✓	✓		
创建包(3)	✓	✓	✓(4)	✓	
发布包	✓	✓	✓	✓	
复制包		✓	✓		
管理 global-packages 文件夹和缓存文件夹。	✓	✓	✓		

II	DOTNET CLI	NUGET CLI (WINDOWS)	NUGET CLI (MONO)	VISUAL STUDIO (WINDOWS)	VISUAL STUDIO FOR MAC
----	------------	------------------------	---------------------	----------------------------	--------------------------

管理 NuGet 配置		✓	✓		
-------------	--	---	---	--	--

(1) 不影响项目文件; 改用 `dotnet.exe`。

(2) 仅适用于 `packages.config` 文件, 不适用于解决方案 (`.sln`) 文件。

(3) 只能通过 CLI 使用各种高级包功能, 因为 Visual Studio UI 工具中没有它们。

(4) 适用于 `.nuspec` 文件, 但不适用于项目文件。

即将推出的功能

如果希望预览即将推出的 NuGet 功能, 请安装 [Visual Studio 预览版](#), 该版本与 Visual Studio 稳定版本并行工作。若要报告问题或分享对预览版的看法, 请在 [NuGet GitHub 存储库](#) 上打开问题。

相关主题

- [使用 Visual Studio 安装和管理包](#)
- [使用 PowerShell 安装和管理包](#)
- [使用 dotnet CLI 安装和管理包](#)
- [使用 nuget.exe CLI 安装和管理包](#)
- [包管理器控制台 PowerShell 引用](#)
- [创建包](#)
- [发布包](#)

在 Windows 上工作的开发人员还可以浏览 [NuGet 包资源管理器](#), 它是可直观浏览、创建和编辑 NuGet 包的独立开源工具。它非常有用, 例如, 无需重新生成包即可对包结构进行实验性更改。

快速入门：使用 dotnet CLI 安装并使用包

2020/4/8 • [Edit Online](#)

NuGet 包包含其他开发人员提供的在项目中使用的可重用代码。请参阅[什么是 NuGet？](#)，了解背景信息。使用如本文所述的适用于常用 `dotnet add package Newtonsoft.Json` 包的 命令将包安装到 .NET Core 项目中。

安装完成后，请引用具有 `using <namespace>` 的代码中的包，其中 `<namespace>` 特定于正在使用的包。然后，可以使用包的 API。

TIP

[nuget.org](#) ■ 浏览 nuget.org 是 .NET 开发人员通常在自己的应用程序中查找可重用组件的方式。你可以直接搜索 nuget.org 或根据本文中的介绍，在 Visual Studio 中查找和安装包。

必备条件

- [.NET Core SDK](#)，提供 `dotnet` 命令行工具。从 Visual Studio 2017 开始，dotnet CLI 将自动随任何与 .NET Core 相关的工作负载一起安装。

创建一个项目

可以将 NuGet 包安装到某种类型的 .NET 项目。在本演练中，如下所示创建一个简单的 .NET Core 控制台项目：

1. 为项目创建文件夹。
2. 打开命令提示符并切换到新文件夹。
3. 请使用以下命令创建项目：

```
dotnet new console
```

4. 使用 `dotnet run` 来测试已正确创建的应用。

添加 Newtonsoft.Json Nuget 包

1. 运行以下命令安装 `Newtonsoft.json` 包：

```
dotnet add package Newtonsoft.Json
```

2. 该命令完成后，打开 `.csproj` 文件以查看所添加的引用：

```
<ItemGroup>
<PackageReference Include="Newtonsoft.Json" Version="12.0.1" />
</ItemGroup>
```

在应用中使用 Newtonsoft.Json API

1. 打开 `Program.cs` 文件，然后在文件的顶部添加以下行：

```
using Newtonsoft.Json;
```

2. 在 `class Program` 行的前面添加以下代码：

```
public class Account
{
    public string Name { get; set; }
    public string Email { get; set; }
    public DateTime DOB { get; set; }
}
```

3. 使用以下代码替换 `Main` 函数：

```
static void Main(string[] args)
{
    Account account = new Account
    {
        Name = "John Doe",
        Email = "john@nuget.org",
        DOB = new DateTime(1980, 2, 20, 0, 0, 0, DateTimeKind.Utc),
    };

    string json = JsonConvert.SerializeObject(account, Formatting.Indented);
    Console.WriteLine(json);
}
```

4. 使用 `dotnet run` 命令生成并运行应用。输出应是 `Account` 对象在代码中的 JSON 表示形式：

```
{
    "Name": "John Doe",
    "Email": "john@nuget.org",
    "DOB": "1980-02-20T00:00:00Z"
}
```

相关视频

在[第 9 频道](#)和[YouTube](#)上查找更多 NuGet 视频。

后续步骤

祝贺你安装并使用第一个 NuGet 包！

使用 dotnet CLI 安装并使用包

若要了解更多 NuGet 产品，请选择以下链接。

- [包使用的概述和工作流](#)
- [查找和选择包](#)
- [项目文件中的包引用](#)

快速入门：在 Visual Studio 中安装和使用包（仅适用于 Windows）

2020/4/8 • [Edit Online](#)

NuGet 包包含其他开发人员提供的在项目中使用的可重用代码。请参阅[什么是 NuGet？](#)，了解背景信息。使用 NuGet 包管理器、[包管理器控制台](#)或 [dotnet CLI](#) 在 Visual Studio 项目中安装包。本文介绍使用热门的 [Newtonsoft.Json](#) 包和 Windows Presentation Foundation (WPF) 项目的过程。相同的过程适用于任何其他 .NET 或 .NET Core 项目。

安装完成后，请引用具有 `using <namespace>` 的代码中的包，其中 `<namespace>` 特定于正在使用的包。建立引用后，可通过相应的 API 调用包。

TIP

[nuget.org](#) ■ 为查找可在自己的应用程序中重用的组件，.NET 开发人员通常都会浏览 nuget.org。可以直接搜索 nuget.org 或根据本文中的介绍，在 Visual Studio 中查找和安装包。有关一般信息，请参阅[查找和评估 NuGet 包](#)。

先决条件

- Visual Studio 2019 .NET 桌面开发工作流。

可以从 [visualstudio.com](#) 免费安装 2019 Community 版，或者使用 Professional 或 Enterprise 版。

如果使用的是 Visual Studio for Mac，请参阅[在 Visual Studio for Mac 中安装并使用包](#)。

创建项目

可将 NuGet 包安装到任何 .NET 项目，前提是包支持与项目相同的目标框架。

本演练使用简单的 WPF 应用。使用以下方法在 Visual Studio 中创建项目：单击“文件”>“新建项目”，在搜索框中键入“.NET”，然后选择“WPF 应用(.NET Framework)”。单击“下一步”。出现提示时，接受 Framework 的默认值。

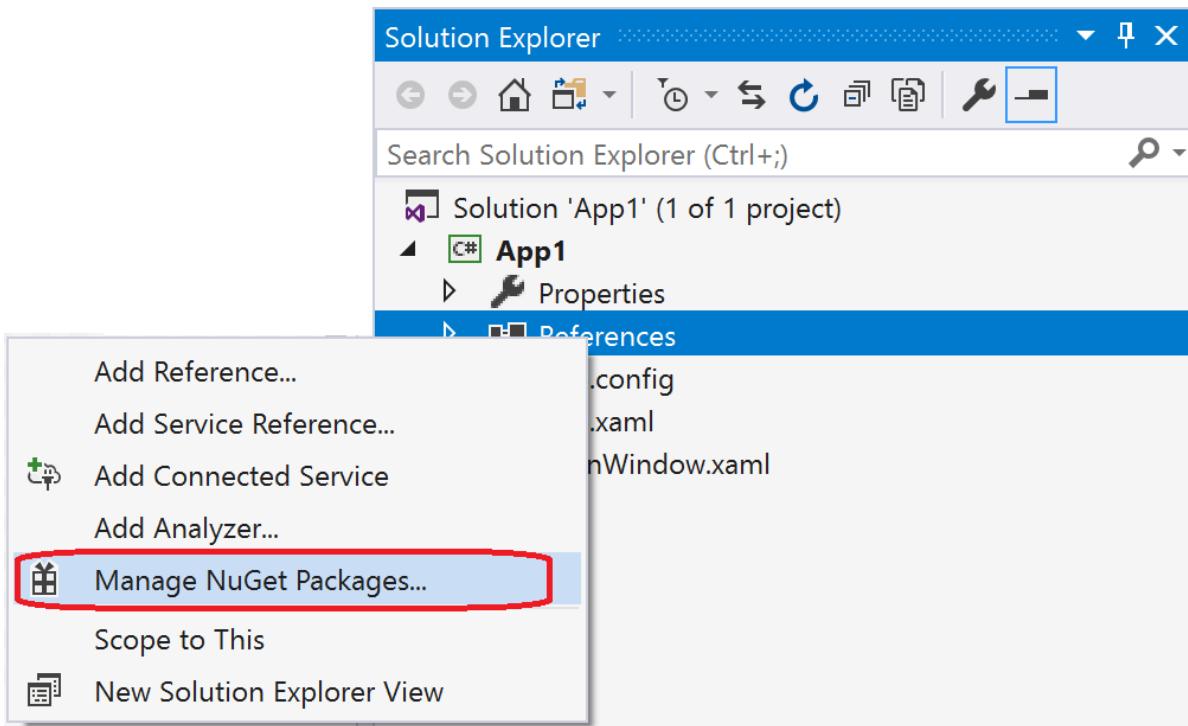
Visual Studio 创建项目，该项目将在解决方案资源管理器中打开。

添加 Newtonsoft.Json NuGet 包

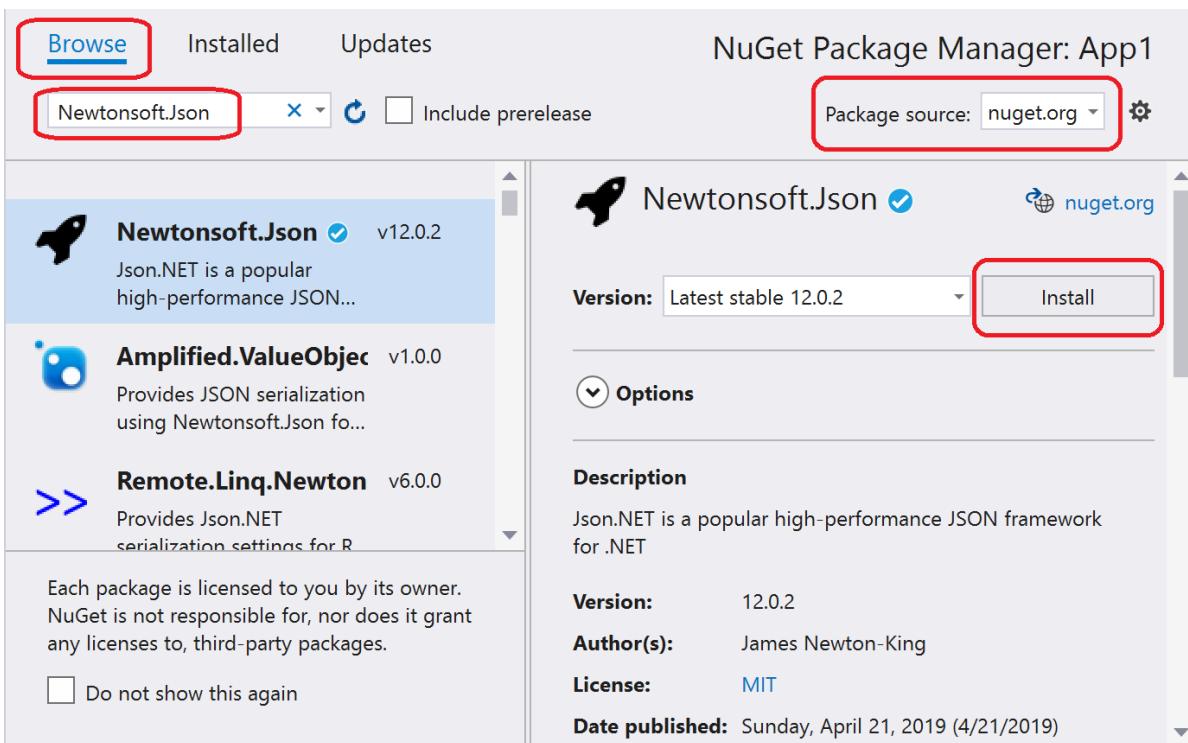
若要安装此包，可以使用 NuGet 包管理器或包管理器控制台。安装包时，NuGet 会将依赖项记录在项目文件或 `packages.config` 文件中（具体位置取决于项目格式）。有关详细信息，请参阅[包使用概述和工作流](#)。

NuGet 程序包管理器

- 在解决方案资源管理器中，右键单击“引用”，选择“管理 NuGet 包”。



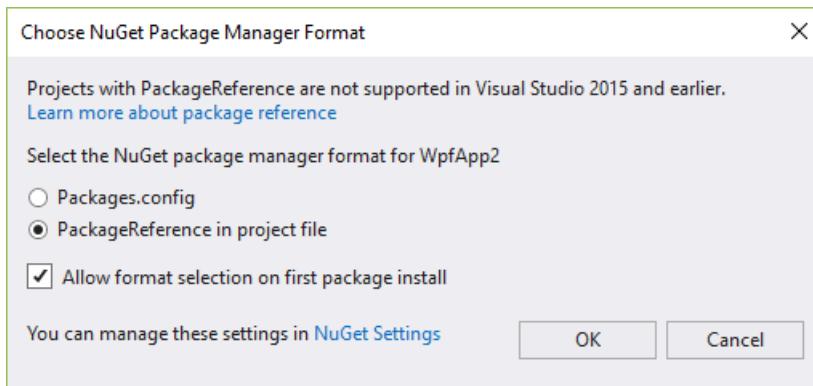
2. 将“nuget.org”选择为“包源”，选择“浏览”选项卡并搜索“Newtonsoft.Json”，在列表中选择该包，然后选择“安装”：



若要了解有关 NuGet 包管理器的详细信息，请参阅[使用 Visual Studio 安装和管理包](#)。

3. 接受任何许可证提示。

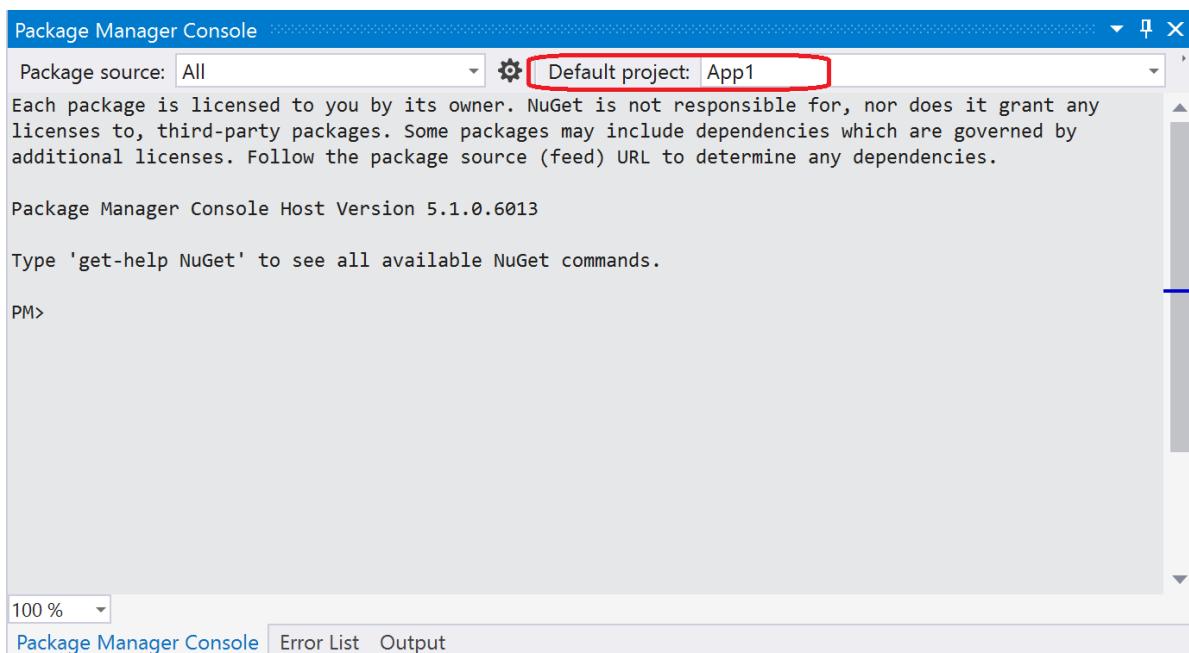
4. (仅适用于 Visual Studio 2017) 如果系统提示选择包管理格式，请选择“项目文件中的 PackageReference”：



5. 如果系统提示查看更改,请选择“确定”。

程序包管理器控制台

1. 选择“工具”>“NuGet 包管理器”>“包管理器控制台”菜单命令。
2. 控制台打开后,检查“默认项目”下拉列表中是否显示在程序包中要安装的项目。如果在解决方案中有一个项目,则它已被选中。



3. 输入命令 `Install-Package Newtonsoft.Json` (请参阅 [Install-Package](#))。控制台窗口会显示该命令的输出。错误通常指示程序包与项目的目标框架不兼容。

若要了解有关包管理器控制台的详细信息,请参阅[使用包管理器控制台安装和管理包](#)。

在应用中使用 Newtonsoft.Json API

使用项目中的 Newtonsoft.Json 包,可调用 `JsonConvert.SerializeObject` 方法将对象转换为可人工读取的字符串。

1. 打开 `MainWindow.xaml` 并将现有 `Grid` 元素替换为以下内容:

```
<Grid Background="White">
    <StackPanel VerticalAlignment="Center">
        <Button Click="Button_Click" Width="100px" HorizontalAlignment="Center" Content="Click Me" Margin="10"/>
        <TextBlock Name="TextBlock" HorizontalAlignment="Center" Text="TextBlock" Margin="10"/>
    </StackPanel>
</Grid>
```

2. 打开 `MainWindow.xaml.cs` 文件(位于 `MainWindow.xaml` 节点下的解决方案资源管理器中), 然后在 `MainWindow` 类中插入以下代码:

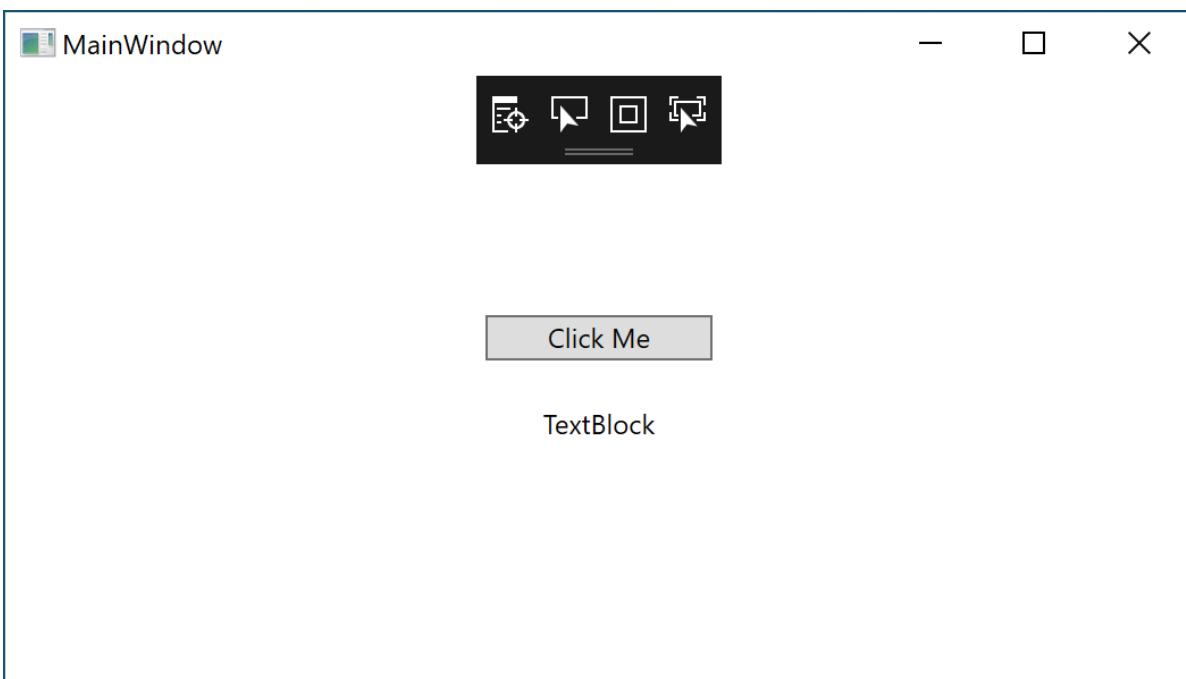
```
public class Account
{
    public string Name { get; set; }
    public string Email { get; set; }
    public DateTime DOB { get; set; }
}

private void Button_Click(object sender, RoutedEventArgs e)
{
    Account account = new Account
    {
        Name = "John Doe",
        Email = "john@microsoft.com",
        DOB = new DateTime(1980, 2, 20, 0, 0, 0, DateTimeKind.Utc),
    };
    string json = JsonConvert.SerializeObject(account, Formatting.Indented);
    TextBlock.Text = json;
}
```

3. 尽管已将 `Newtonsoft.Json` 包添加到项目中, 因为你需要使用代码文件最上方的 `using` 语句, 所以 `JsonConvert` 下仍会出现红色波形曲线:

```
using Newtonsoft.Json;
```

4. 要构建并运行应用, 请按 F5 或选择“调试”>“启动调试”:



5. 选择按钮, 查看替换为某些 JSON 文本的 TextBlock 的内容:



```
{  
    "Name": "John Doe",  
    "Email": "john@microsoft.com",  
    "DOB": "1980-02-20T00:00:00Z"  
}
```

相关视频

在[第 9 频道](#)和 [YouTube](#) 上查找更多 NuGet 视频。

后续步骤

祝贺你安装并使用第一个 NuGet 包！

[使用 Visual Studio 安装和管理包](#)

[使用包管理器控制台安装和管理包](#)

若要了解更多 NuGet 产品，请选择以下链接。

- [包使用的概述和工作流](#)
- [查找和选择包](#)
- [项目文件中的包引用](#)

快速入门：在 Visual Studio for Mac 中安装和使用包

2020/4/8 • [Edit Online](#)

NuGet 包包含其他开发人员提供的在项目中使用的可重用代码。请参阅[什么是 NuGet?](#)，了解背景信息。使用 NuGet 包管理器在 Visual Studio for Mac 项目中安装包。本文介绍使用热门的 [Newtonsoft.Json](#) 包和 .NET Core 控制台项目的过程。相同的过程适用于任何其他 Xamarin 或 .NET Core 项目。

安装完成后，请引用具有 `using <namespace>` 的代码中的包，其中 `<namespace>` 特定于正在使用的包。建立引用后，可通过相应的 API 调用包。

TIP

[nuget.org](#) ■ 为查找可在自己的应用程序中重用的组件，.NET 开发人员通常都会浏览 [nuget.org](#)。可以直接搜索 [nuget.org](#) 或根据本文中的介绍，在 Visual Studio 中查找和安装包。有关一般信息，请参阅[查找和评估 NuGet 包](#)。

先决条件

- Visual Studio 2019 for Mac。

可以从 [visualstudio.com](#) 免费安装 2019 Community 版，或者使用 Professional 或 Enterprise 版。

如果是在 Windows 上使用 Visual Studio，请参阅[在 Visual Studio 中安装并使用包\(仅适用于 Windows\)](#)。

创建项目

可将 NuGet 包安装到任何 .NET 项目，前提是包支持与项目相同的目标框架。

本演练使用简单的 .NET Core 控制台应用。通过以下方式在 Visual Studio for Mac 中创建项目：选择“文件”>“新建解决方案...”，然后选择“.NET Core”>“应用”>“控制台应用程序”模板。单击“下一步”。出现提示时，接受“目标框架”的默认值。

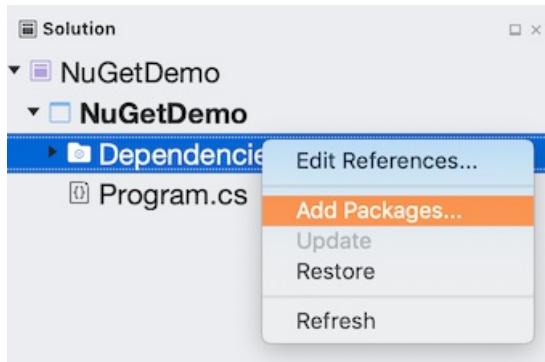
Visual Studio 创建项目，该项目将在解决方案资源管理器中打开。

添加 Newtonsoft.Json Nuget 包

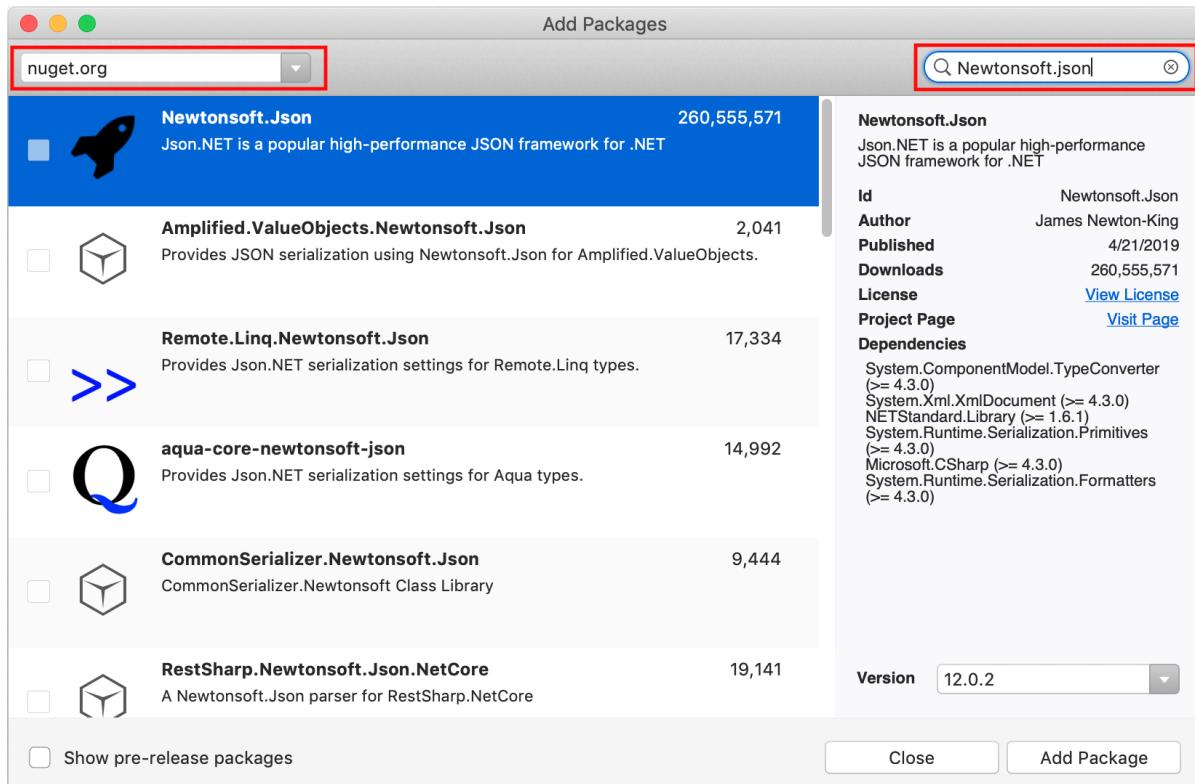
若要安装此包，请使用 NuGet 包管理器。安装包时，NuGet 会将依赖项记录在项目文件或 `packages.config` 文件中（具体位置取决于项目格式）。有关详细信息，请参阅[包使用概述和工作流](#)。

NuGet 程序包管理器

1. 在解决方案资源管理器中，右键单击“依赖项”，然后选择“添加包...”。



2. 在对话框的左上角，选择“nuget.org”作为“包源”，并搜索“Newtonsoft.Json”，在列表中选择该包，然后选择“添加包...”：



若要了解有关 NuGet 包管理器的详细信息，请参阅[使用 Visual Studio for Mac 安装和管理包](#)。

在应用中使用 Newtonsoft.Json API

使用项目中的 Newtonsoft.Json 包，可调用 `JsonConvert.SerializeObject` 方法将对象转换为可人工读取的字符串。

1. 打开 `Program.cs` 文件(位于 Solution Pad 中)，然后使用以下代码替换文件内容：

```
using System;
using Newtonsoft.Json;

namespace NuGetDemo
{
    public class Account
    {
        public string Name { get; set; }
        public string Email { get; set; }
        public DateTime DOB { get; set; }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Account account = new Account()
            {
                Name = "Joe Doe",
                Email = "joe@test.com",
                DOB = new DateTime(1976, 3, 24)
            };
            string json = JsonConvert.SerializeObject(account);
            Console.WriteLine(json);
        }
    }
}
```

2. 选择“运行”>“开始调试”，生成和运行应用：

3. 应用运行后，你将看到控制台中显示的序列化的 JSON 输出：



```
{"Name": "Joe Doe", "Email": "joe@test.com", "DOB": "1976-03-24T00:00:00"}  
Press any key to continue...
```

后续步骤

祝贺你安装并使用第一个 NuGet 包！

使用 Visual Studio for Mac 安装和管理包

若要了解更多 NuGet 产品, 请选择以下链接。

- [包使用的概述和工作流](#)
- [项目文件中的包引用](#)

快速入门：创建和发布包 (dotnet CLI)

2020/4/8 • [Edit Online](#)

从 .NET 类库创建 NuGet 包并使用 `dotnet` 命令行接口 (CLI) 将其发布到 nuget.org 是很简单的过程。

必备条件

- 安装包括 [CLI 的](#) .NET Core SDK `dotnet`。从 Visual Studio 2017 开始, dotnet CLI 将自动随任何与 .NET Core 相关的工作负载一起安装。
- 如果你还没有帐户, 请[在 nuget.org 上注册一个免费帐户](#)。创建新帐户会发送确认电子邮件。必须先确认该帐户, 才能上传包。

创建类库项目

你可以使用现有的 .NET 类库项目用于要打包的代码, 或者创建一个简单的项目, 如下所示:

- 创建名为 `AppLogger` 的文件夹。
- 打开命令提示符并切换到 `AppLogger` 文件夹。
- 类型 `dotnet new classlib`, 它使用项目当前文件夹的名称。

这会创建新项目。

将包元数据添加到项目文件

每个 NuGet 包都需要一个清单, 用以描述包的内容和依赖项。在最终包中, 清单是基于项目文件中包含的 NuGet 元数据属性生成的 `.nuspec` 文件。

- 打开项目文件 (`.csproj`), 并在现有 `<PropertyGroup>` 标记内至少添加以下属性, 同时根据需要更改值:

```
<PackageId>AppLogger</PackageId>
<Version>1.0.0</Version>
<Authors>your_name</Authors>
<Company>your_company</Company>
```

IMPORTANT

为包提供一个在 nuget.org 中唯一或你使用的任何主机的标识符。对于本次演练, 我们建议在名称中包含“Sample”或“Test”, 因为稍后的发布步骤确实会使该包公开显示(尽管实际上不太可能有人会使用它)。

- 添加 NuGet 元数据属性中描述的任何可选属性。

NOTE

对于面向公共使用而生成的包, 请特别注意 `PackageTags` 属性, 因为这些标记可帮助其他人查找包并了解其用途。

运行 pack 命令

若要从项目中生成 NuGet 包 (`.nupkg` 文件), 运行 `dotnet pack` 命令, 它也会自动生成项目:

```
# Uses the project file in the current folder by default  
dotnet pack
```

输出显示 `.nupkg` 文件的路径：

```
Microsoft (R) Build Engine version 15.5.180.51428 for .NET Core  
Copyright (C) Microsoft Corporation. All rights reserved.  
  
Restore completed in 29.91 ms for D:\proj\AppLoggerNet\AppLogger\AppLogger.csproj.  
AppLogger -> D:\proj\AppLoggerNet\AppLogger\bin\Debug\netstandard2.0\AppLogger.dll  
Successfully created package 'D:\proj\AppLoggerNet\AppLogger\bin\Debug\AppLogger.1.0.0.nupkg'.
```

在生成期间自动生成包

若要在运行 `dotnet pack` 时自动运行 `dotnet build`，请将以下行添加到 `<PropertyGroup>` 中的项目文件内：

```
<GeneratePackageOnBuild>true</GeneratePackageOnBuild>
```

发布包

有了 `.nupkg` 文件后，可以使用 `dotnet nuget push` 命令以及从 nuget.org 获取的 API 密钥将其发布到 nuget.org。

NOTE

所有上传到 nuget.org 的包都会进行病毒扫描，如果发现任何病毒，将拒绝包。此外，还会定期扫描 nuget.org 上列出的所有包。

发布到 nuget.org 的包也对其他开发者公开可见，除非你取消列出它们。若要专门托管包，请参阅[托管包](#)。

获取 API 密钥

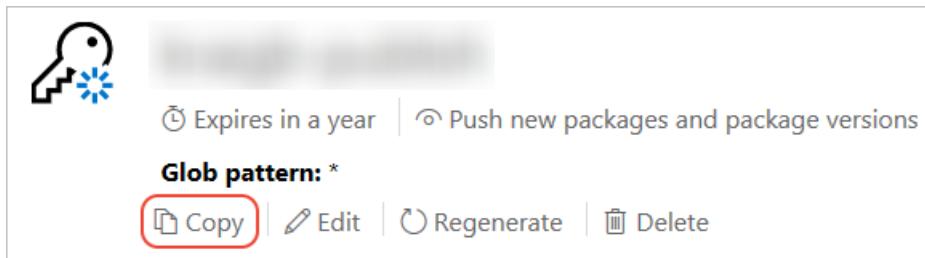
1. 登录你的 nuget.org 帐户，或创建一个帐户（如果你还没有帐户）。

有关创建帐户的详细信息，请参阅[个人帐户](#)。

2. 选择用户名（在右上角），然后选择“API 密钥”。

3. 选择“创建”，提供密钥名称，选择“选择范围”>“推送”。输入“Glob 模式”*，然后选择“创建”。（请参阅下面有关范围的详细信息。）

4. 创建密钥后，选择“复制”，检索需要在 CLI 中使用的访问密钥：



5. 重要事项：将你的密钥保存在安全位置，因为以后无法再次复制密钥。如果返回到 API 密钥页，则需要重新生成密钥以对其进行复制。如果不再希望通过 CLI 推送包，还可以删除 API 密钥。

范围允许创建针对不同用途的单独 API 密钥。每个密钥都有其过期时间，并且可以将范围限定为特定包（或 glob 模式）。每个密钥还将范围限定为特定操作：新包和更新推送、仅更新推送，或者从列表中删除。通过范围限定，可以为管理组织不同包的不同人员创建 API 密钥，这样他们就只有所需的权限。有关详细信息，请参阅[范围内的 API 密钥](#)。

用 dotnet nuget push 发布

- 更改到包含 `.nupkg` 文件的文件夹。
- 运行以下命令，指定包名称(唯一包 ID)并使用你的 API 密钥替换密钥值：

```
dotnet nuget push AppLogger.1.0.0.nupkg -k qz2jga8pl3dvn2akk9yquwcs9ygggg4exypy3bhxy6w6x6 -s https://api.nuget.org/v3/index.json
```

- dotnet 会显示发布过程的结果：

```
info : Pushing AppLogger.1.0.0.nupkg to 'https://www.nuget.org/api/v2/package'...
info : PUT https://www.nuget.org/api/v2/package/
info : Created https://www.nuget.org/api/v2/package/ 12620ms
info : Your package was pushed.
```

请参阅 [dotnet nuget push](#)。

发布错误

`push` 命令中的错误通常表示存在问题。例如，你可能会忘记更新项目中的版本号，因此尝试发布已存在的包。

尝试使用主机上已存在的标识符发布包时，你也会看到错误。例如，名称“AppLogger”已经存在。在这种情况下，`push` 命令会给出以下错误：

```
Response status code does not indicate success: 403 (The specified API key is invalid, has expired, or does not have permission to access the specified package.).
```

如果你使用的是刚刚创建的有效 API 密钥，则此消息表明存在命名冲突，并未从错误的“权限”部分中将其完全清除。更改包标识符，重建项目，重新创建 `.nupkg` 文件，然后重试 `push` 命令。

管理已发布的包

从 nuget.org 上的配置文件中，选择“管理包”，查看刚刚发布的包。同样也会收到确认电子邮件。请注意，包可能需要一些时间才能编入索引并显示在可供他人查看的搜索结果中。在该时间段，包页面会显示以下消息：

⚠ This package has not been published yet. It will appear in search results and will be available for install/restore after both validation and indexing are complete. Package validation and indexing may take up to an hour. [Read more](#).

这就是所有的操作！刚刚已将第一个 NuGet 包发布到 nuget.org，其他开发人员可在自己的项目中使用它。

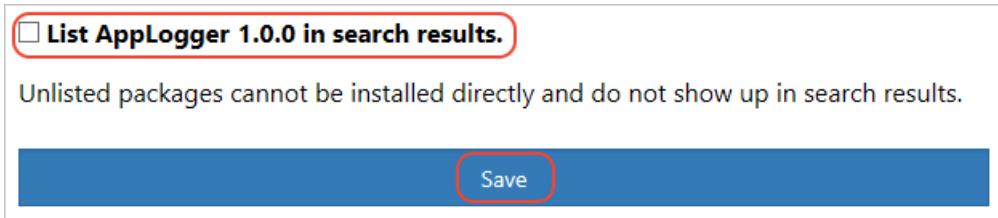
如果你已在本演练中创建一个实际上并不使用的包（例如使用空的类库创建的包），则应取消列出将在搜索结果中隐藏的包：

- 在 nuget.org 上，选择用户名（在该页的右上角），然后选择“管理包”。
- 找到你需要在“已发布”下取消列出的包，然后选择右侧的回收站图标：

The screenshot shows the 'Published' section of the nuget.org user profile. It displays the following information:

- You have 1 published package with a total of 0 downloads**
- Package ID**: AppLogger
- Owners**: [Redacted]
- Downloads**: 0
- Latest Version**: 1.0.0
- Action Buttons**: Edit, Details, Recycle Bin (highlighted with a red circle)

3. 在随后的页面上, 清除标记有“在搜索结果中列出(包名)”的框, 然后选择“保存”:



相关视频

在[第 9 频道](#)和 [YouTube](#) 上查找更多 NuGet 视频。

后续步骤

祝贺你创建第一个 NuGet 包 !

[创建包](#)

若要了解更多 NuGet 产品, 请选择以下链接。

- [发布包](#)
- [预发行包](#)
- [支持多个目标框架](#)
- [包版本控制](#)
- [创建本地化包](#)
- [创建符号包](#)
- [给包签名](#)

快速入门：使用 Visual Studio 创建和发布 NuGet 包 (仅限 .NET Standard 和 Windows)

2020/4/8 • [Edit Online](#)

从 Windows 上 Visual Studio 中的 .NET Standard 类库创建 NuGet 包，然后使用 CLI 工具将其发布到 nuget.org，这是一个很简单的过程。

NOTE

如果使用的是 Visual Studio for Mac，请参阅有关创建 NuGet 包的[以下信息](#)或使用 `dotnet CLI` 工具。

先决条件

1. 通过与 .NET Core 相关的工作负载从 [visualstudio.com](#) 安装任意版本的 Visual Studio 2019。
2. 如果尚未安装，则安装 `dotnet CLI`。

对于 `dotnet CLI`，从 Visual Studio 2017 开始，`dotnet CLI` 将自动随任何与 .NET Core 相关的工作负载一起安装。否则，请安装 [.NET Core SDK](#) 以获取 `dotnet CLI`。`dotnet CLI` 是使用 [SDK 样式格式](#)(SDK 属性)的 .NET Standard 项目所必需的。Visual Studio 2017 及更高版本中的默认 .NET Standard 类库模板(本文所用模板)使用 SDK 属性。

IMPORTANT

如果使用的是非 SDK 样式的项目，请改为按照[创建和发布 .NET Framework 包 \(Visual Studio\)](#)中的过程来创建和发布包。对于本文，建议使用 `dotnet CLI`。虽然可以使用 `nuget.exe CLI` 发布任何 NuGet 包，但本文中的某些步骤特定于 SDK 样式的项目和 `dotnet CLI`。`nuget.exe CLI` 用于[非 SDK 样式的项目](#)(通常为 .NET Framework)。

3. 如果你还没有帐户，请在 [nuget.org 上注册一个免费帐户](#)。创建新帐户会发送确认电子邮件。必须先确认该帐户，才能上传包。

创建类库项目

可以使用现有的 .NET Standard 类库项目用于要打包的代码，或者创建一个简单的项目，如下所示：

1. 在 Visual Studio 中，选择“文件”>“新建”>“项目”，展开“Visual C# > .NET Standard”节点，选择“类库 (.NET Standard)”模板，将项目命名为“AppLogger”，然后单击“确定”。

TIP

除非你有其他选择理由，否则 .NET Standard 是 NuGet 包的首选目标，因为它提供了与最广泛的使用项目的兼容性。

2. 右键单击生成的项目文件并选择“生成”，确保已正确创建项目。DLL 位于调试文件夹中(或发布中，如果生成的是该配置)。

当然，在实际的 NuGet 包中，可实现许多有用的功能，让其他人可通过这些功能生成应用程序。但是对于本演练，无需编写其他任何代码，因为模板的类库足以创建包。但是，如果你需要此程序包的某个功能代码，请使用以下命令：

```

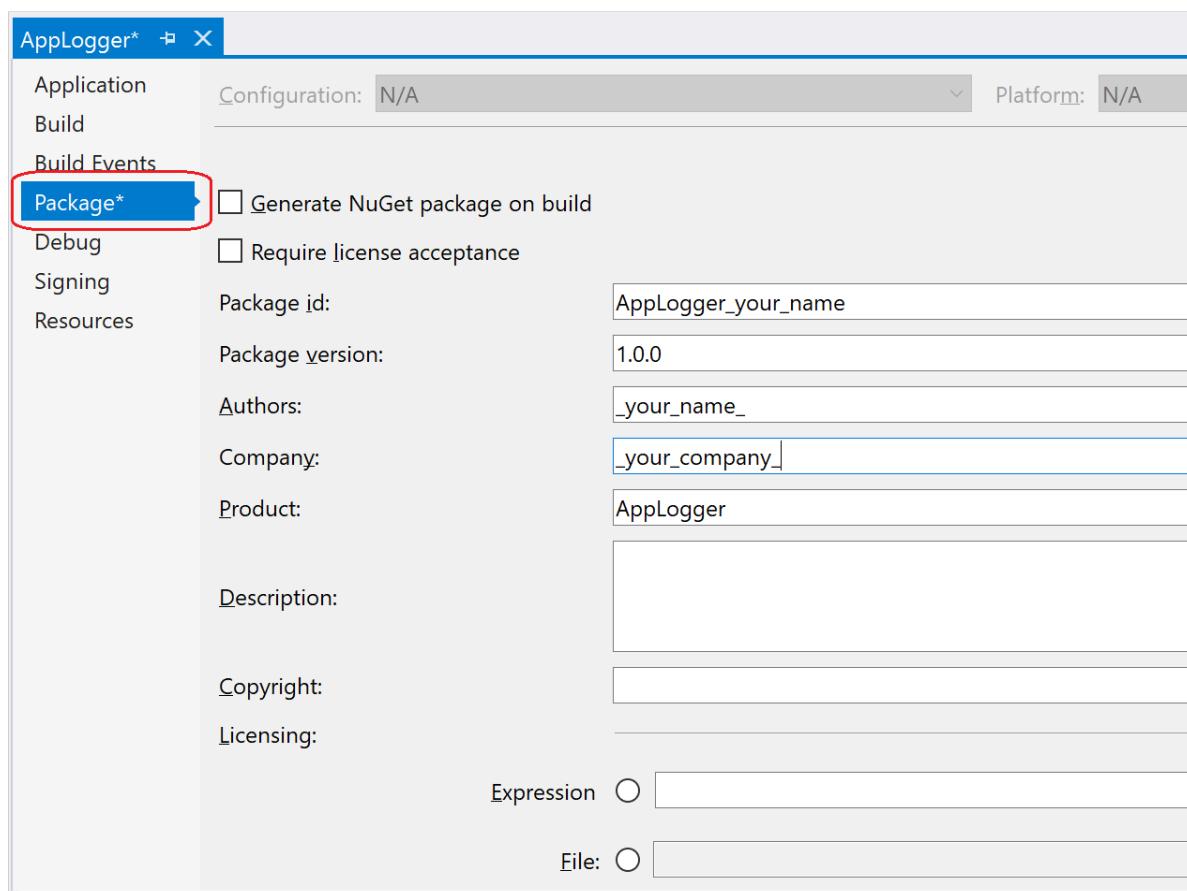
namespace AppLogger
{
    public class Logger
    {
        public void Log(string text)
        {
            Console.WriteLine(text);
        }
    }
}

```

配置包属性

- 在解决方案资源管理器中右键单击该项目，然后选择“属性”菜单命令，然后选择“包”选项卡。

“包”选项卡仅在 Visual Studio 的 SDK 样式项目中显示，通常是 .NET Standard 或 .NET Core 类库项目；如果要针对非 SDK 样式项目（通常是 .NET Framework），请迁移项目或者改为参阅[创建和发布 .NET Framework 包](#)，以获取分步说明。



NOTE

对于面向公共使用而生成的包，请特别注意 **Tags** 属性，因为这些标记可帮助其他人查找包并了解其用途。

- 为包提供一个唯一标识符，并填写任何其他所需的属性。若要将 MSBuild 属性（SDK 样式项目）映射到 **.nuspec** 中的属性，请参阅[包目标](#)。有关属性的说明，请参阅[.nuspec 文件引用](#)。这里的所有属性都列入 Visual Studio 为项目创建的 **.nuspec** 清单。

IMPORTANT

你必须为包提供一个在 nuget.org 中唯一或你使用的任何主机的标识符。对于本次演练，我们建议在名称中包含“Sample”或“Test”，因为稍后的发布步骤确实会使该包公开显示（尽管实际上不太可能有人会使用它）。

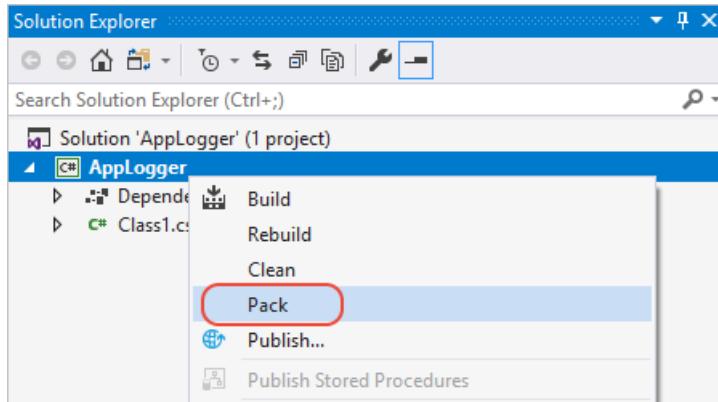
如果你尝试发布名称已存在的包，则会看到一个错误。

3. (可选) 若要直接查看项目文件中的属性，请右键单击“解决方案资源管理器”中的“项目”，然后选择“编辑 AppLogger.csproj”。

此选项从 Visual Studio 2017 开始仅对使用 SDK 样式属性的项目可用。否则，右键单击项目，并选择“卸载项目”。然后右键单击卸载的项目并选择“编辑 AppLogger.csproj”。

运行 pack 命令

1. 将此配置设置为“发布”。
2. 请在“解决方案资源管理器”中右键单击该项目，然后选择“Pack”命令：



如果没有看到“Pack”命令，那么项目可能不是 SDK 样式的项目，需要使用 `nuget.exe` CLI。[迁移项目](#) 并使用 `dotnet` CLI，或者改为参阅[创建和发布 .NET Framework 包](#)，以获取分步说明。

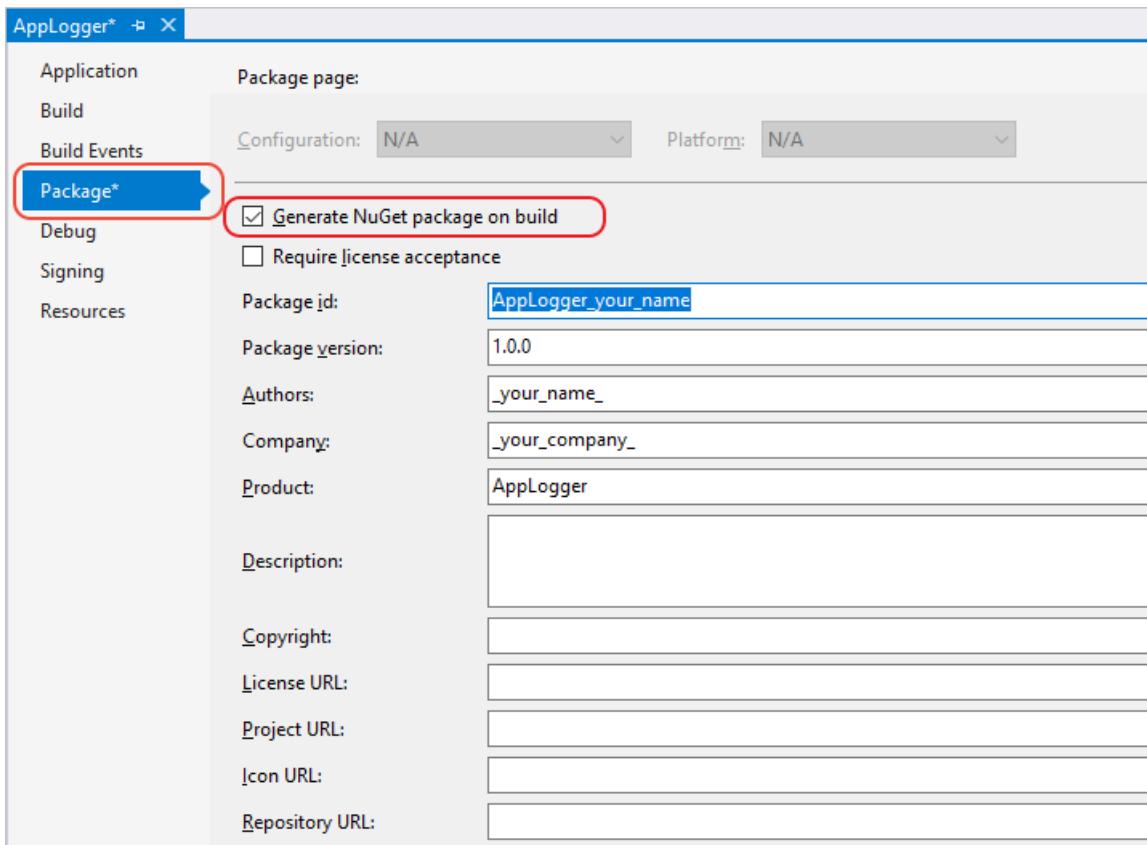
3. Visual Studio 构建项目并创建 `.nupkg` 文件。检查“输出”窗口以查看详细信息（类似于以下内容），其中包含包文件的路径。另请注意，生成的程序集位于适合 .NET Standard 2.0 目标的 `bin\Release\netstandard2.0` 中。

```
1>----- Build started: Project: AppLogger, Configuration: Release Any CPU -----
1>AppLogger -> d:\proj\AppLogger\AppLogger\bin\Release\netstandard2.0\AppLogger.dll
1>Successfully created package 'd:\proj\AppLogger\AppLogger\bin\Release\AppLogger.1.0.0.nupkg'.
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ======
```

(可选) 在生成期间生成包

可以将 Visual Studio 配置为在生成项目时自动生成 NuGet 包。

1. 在“解决方案资源管理器”中，右键单击项目，然后选择“属性”。
2. 在“包”选项卡中，选择“在生成期间生成 NuGet 包”。



NOTE

自动生成包时，打包时间会增加项目的生成时间。

(可选)使用 MSBuild 打包

作为使用“打包”菜单命令的备选项，当项目包含必要的包数据时，NuGet 4.x+ 和 MSBuild 15.1+ 支持 `pack` 目标。打开命令提示符，导航到项目文件夹并运行以下命令。（用户通常习惯从“开始”菜单中启动“适用于 Visual Studio 的开发人员命令提示符”，因为它将使用 MSBuild 的所有必需路径进行配置。）

有关详细信息，请参阅[使用 MSBuild 创建包](#)。

发布包

有了 `.nupkg` 文件后，可以使用 `nuget.exe` CLI 或 `dotnet.exe` CLI 以及从 nuget.org 获取的 API 密钥将其发布到 nuget.org。

NOTE

■:所有上传到 nuget.org 的包都会进行病毒扫描，如果发现任何病毒，将拒绝包。此外，还会定期扫描 nuget.org 上列出的所有包。

发布到 nuget.org 的包也对其他开发者公开可见，除非你取消列出它们。若要专门托管包，请参阅[托管包](#)。

获取 API 密钥

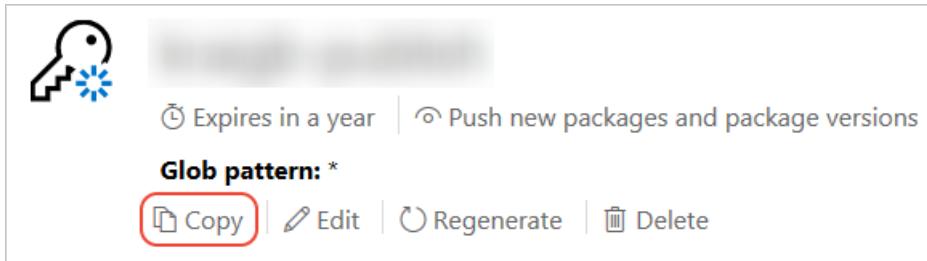
1. [登录你的 nuget.org 帐户](#)，或创建一个帐户（如果你还没有帐户）。

有关创建帐户的详细信息，请参阅[个人帐户](#)。

2. 选择用户名（在右上角），然后选择“API 密钥”。

3. 选择“创建”，提供密钥名称，选择“选择范围”>“推送”。输入“Glob 模式”*，然后选择“创建”。（请参阅下面有关范围的详细信息。）

4. 创建密钥后，选择“复制”，检索需要在 CLI 中使用的访问密钥：



5. 重要事项：将你的密钥保存在安全位置，因为以后无法再次复制密钥。如果返回到 API 密钥页，则需要重新生成密钥以对其进行复制。如果不再希望通过 CLI 推送包，还可以删除 API 密钥。

范围允许创建针对不同用途的单独 API 密钥。每个密钥都有其过期时间，并且可以将范围限定为特定包（或 glob 模式）。每个密钥还将范围限定为特定操作：新包和更新推送、仅更新推送，或者从列表中删除。通过范围限定，可以为管理组织不同包的不同人员创建 API 密钥，这样他们就只有所需的权限。有关详细信息，请参阅[范围内的 API 密钥](#)。

使用 dotnet CLI 或 nuget.exe CLI 发布

选择 CLI 工具 (.NET Core CLI (dotnet CLI) 或 NuGet (nuget.exe CLI)) 对应的选项卡。

- [.NET Core CLI](#)
- [NuGet](#)

此步骤是使用 `nuget.exe` 的推荐替代方法。

在发布包之前，必须先打开命令行。

1. 更改到包含 `.nupkg` 文件的文件夹。
2. 运行以下命令，指定包名称（唯一包 ID）并使用你的 API 密钥替换密钥值：

```
dotnet nuget push AppLogger.1.0.0.nupkg -k qz2jga8pl3dvn2akkysyquwcs9ygggg4exypy3bhxy6w6x6 -s https://api.nuget.org/v3/index.json
```

3. dotnet 会显示发布过程的结果：

```
info : Pushing AppLogger.1.0.0.nupkg to 'https://www.nuget.org/api/v2/package'...
info : PUT https://www.nuget.org/api/v2/package/
info : Created https://www.nuget.org/api/v2/package/ 12620ms
info : Your package was pushed.
```

请参阅 [dotnet nuget push](#)。

发布错误

`push` 命令中的错误通常表示存在问题。例如，你可能会忘记更新项目中的版本号，因此尝试发布已存在的包。

尝试使用主机上已存在的标识符发布包时，你也会看到错误。例如，名称“AppLogger”已经存在。在这种情况下，`push` 命令会给出以下错误：

```
Response status code does not indicate success: 403 (The specified API key is invalid,
has expired, or does not have permission to access the specified package.).
```

如果你使用的是刚刚创建的有效 API 密钥，则此消息表明存在命名冲突，并未从错误的“权限”部分中将其完全清除。更改包标识符，重建项目，重新创建 `.nupkg` 文件，然后重试 `push` 命令。

管理已发布的包

从 nuget.org 上的配置文件中，选择“管理包”，查看刚刚发布的包。同样也会收到确认电子邮件。请注意，包可能需要一些时间才能编入索引并显示在可供他人查看的搜索结果中。在该时间段，包页面会显示以下消息：

⚠ This package has not been published yet. It will appear in search results and will be available for install/restore after both validation and indexing are complete. Package validation and indexing may take up to an hour. [Read more.](#)

这就是所有的操作！刚刚已将第一个 NuGet 包发布到 nuget.org，其他开发人员可在自己的项目中使用它。

如果你已在本演练中创建一个实际上并不使用的包（例如使用空的类库创建的包），则应取消列出将在搜索结果中隐藏的包：

1. 在 nuget.org 上，选择用户名（在该页的右上角），然后选择“管理包”。
2. 找到你需要在“已发布”下取消列出的包，然后选择右侧的回收站图标：

You have 1 published package with a total of 0 downloads

Package ID	Owners	Downloads	Latest Version
AppLogger	[redacted]	0	1.0.0

3. 在随后的页面上，清除标记有“在搜索结果中列出(包名)”的框，然后选择“保存”：

List AppLogger 1.0.0 in search results.
Unlisted packages cannot be installed directly and do not show up in search results.

Save

添加自述文件和其他文件

若要直接指定要包含在包中的文件，请编辑项目文件并使用 `content` 属性：

```
<ItemGroup>
<Content Include="readme.txt">
<Pack>true</Pack>
<PackagePath>\</PackagePath>
</Content>
</ItemGroup>
```

这将在包根目录中包含一个名为 `readme.txt` 的文件。Visual Studio 在直接安装包之后立即将该文件的内容显示为纯文本。（对于安装为依赖项的包，不会显示自述文件）。例如，下面是 HtmlAgilityPack 包的自述文件的显示方式：

```
readme.txt ✘ X
1 -----
2 ----- Html Agility Pack Nuget Readme -----
3 -----
4
5 ----Silverlight 4 and Windows Phone 7.1+ projects-----
6 To use XPATH features: System.Xml.XPath.dll from the Silverlight 4 SDK must be referenced.
7 This is normally found at
8 %ProgramFiles(x86)%\Microsoft SDKs\Microsoft SDKs\Silverlight\v4.0\Libraries\Client
9 or
10 %ProgramFiles%\Microsoft SDKs\Microsoft SDKs\Silverlight\v4.0\Libraries\Client
11
12 ----Silverlight 5 projects-----
13 To use XPATH features: System.Xml.XPath.dll from the Silverlight 5 SDK must be referenced.
14 This is normally found at
15 %ProgramFiles(x86)%\Microsoft SDKs\Microsoft SDKs\Silverlight\v5.0\Libraries\Client
16 or
17 %ProgramFiles%\Microsoft SDKs\Microsoft SDKs\Silverlight\v5.0\Libraries\Client
18
```

NOTE

只在项目根目录添加 `readme.txt` 不会导致它被包含在生成的包中。

相关视频

在[第 9 频道](#)和[YouTube](#) 上查找更多 NuGet 视频。

相关主题

- [创建包](#)
- [发布包](#)
- [预发行包](#)
- [支持多个目标框架](#)
- [包版本控制](#)
- [创建本地化包](#)
- [.NET Standard 库文档](#)
- [从 .NET Framework 移植到 .NET Core](#)

快速入门：使用 Visual Studio 创建和发布包 (.NET Framework、Windows)

2020/4/8 • [Edit Online](#)

若要从 .NET Framework 类库创建 NuGet 包，需要在 Windows 上的 Visual Studio 中创建 DLL，然后使用 nuget.exe 命令行工具创建并发布包。

NOTE

本快速入门教程仅适用于 Windows 版的 Visual Studio 2017 和更高版本。Visual Studio for Mac 不包括此处所述的功能。改为使用 [dotnet CLI 工具](#)。

先决条件

1. 通过任何与 .NET 相关的工作负载从 [visualstudio.com](#) 安装任意版本的 Visual Studio 2017 或更高版本。安装 .NET 工作负载时，Visual Studio 2017 会自动包含 NuGet 功能。
2. 要安装 `nuget.exe` CLI，从 [nuget.org](#) 下载它，将 `.exe` 文件保存到合适的文件夹，然后将该文件夹添加到 PATH 环境变量中。
3. 如果你还没有帐户，请在 [nuget.org](#) 上注册一个免费帐户。创建新帐户会发送确认电子邮件。必须先确认该帐户，才能上传包。

创建类库项目

可以使用现有的 .NET Framework 类库项目用于要打包的代码，或者创建一个简单的项目，如下所示：

1. 在 Visual Studio 中，选择“文件”>“新建”>“项目”，再依次选择“Visual C#”节点、“类库(.NET Framework)”模板，将项目命名为“AppLogger”，然后单击“确定”。
2. 右键单击生成的项目文件并选择“生成”，确保已正确创建项目。DLL 位于调试文件夹中（或发布中，如果生成的是该配置）。

当然，在实际的 NuGet 包中，可实现许多有用的功能，让其他人可通过这些功能生成应用程序。也可以按个人喜欢的方式设置目标框架。有关示例，请参阅 [UWP](#) 和 [Xamarin](#) 指南。

但是对于本演练，无需编写其他任何代码，因为模板的类库足以创建包。但是，如果你需要此程序包的某个功能代码，请使用以下命令：

```
using System;

namespace AppLogger
{
    public class Logger
    {
        public void Log(string text)
        {
            Console.WriteLine(text);
        }
    }
}
```

TIP

除非你有其他选择理由, 否则 .NET Standard 是 NuGet 包的首选目标, 因为它提供了与最广泛的使用项目的兼容性。请参阅[使用 Visual Studio 创建和发布包 \(.NET Standard\)](#)。

配置包的项目属性

NuGet 包中包含清单(`.nuspec` 文件), 其中包含相关的元数据, 例如包标识符、版本号和描述等。其中一些元数据可直接从项目属性中获取, 避免在项目和清单中对其进行单独更新。本节介绍在何处设置适用的属性。

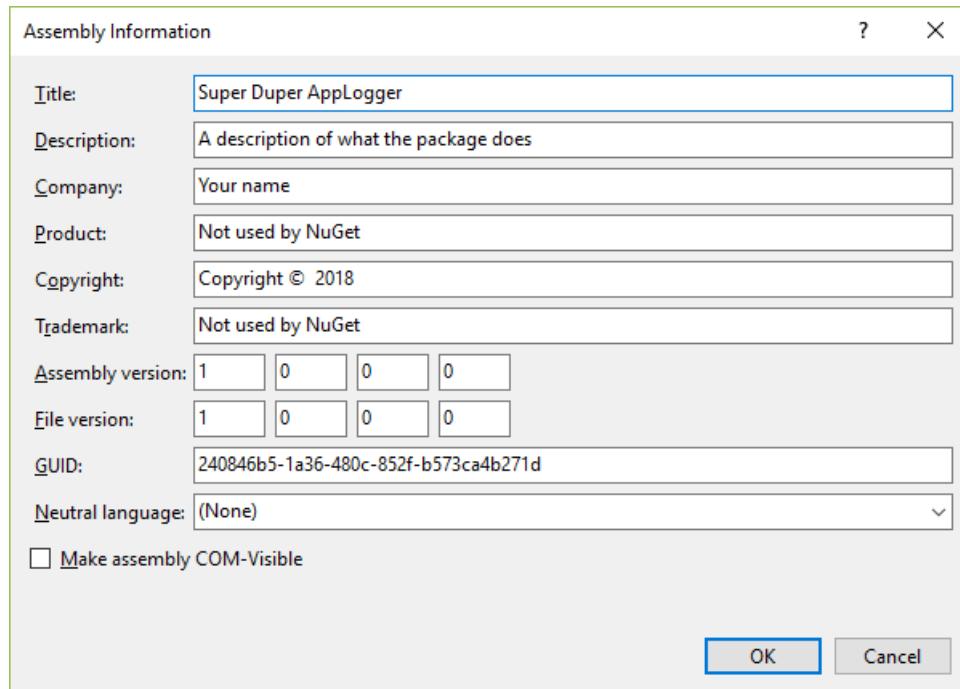
1. 选择“项目”>“属性”菜单命令, 然后选择“应用程序”选项卡。
2. 在“程序集名称”字段中, 为包提供唯一标识符。

IMPORTANT

你必须为包提供一个在 nuget.org 中唯一或你使用的任何主机的标识符。对于本次演练, 我们建议在名称中包含“Sample”或“Test”, 因为稍后的发布步骤确实会使该包公开显示(尽管实际上不太可能有人会使用它)。

如果你尝试发布名称已存在的包, 则会看到一个错误。

3. 选择“程序集信息...”按钮, 此时会出现一个对话框, 可在其中输入清单中的其他属性(请参阅[.nuspec 文件引用 - 替换令牌](#))。最常用的字段是“标题”、“描述”、“公司”、“版权所有”和“程序集版本”。这些属性最终会随包出现在主机上(例如 nuget.org), 因此请确保对其进行完整描述。



4. 可选:若要直接查看和编辑属性, 请打开项目中的 `Properties/AssemblyInfo.cs` 文件。
5. 设置属性时, 请将项目配置设置为“发布”并重新生成项目以生成更新的 DLL。

生成初始清单

既然已准备好 DLL 并已设置项目属性, 现在即可使用 `nuget spec` 命令从项目生成初始 `.nuspec` 文件。此步骤包括相关替换令牌, 便于从项目文件中提取信息。

仅运行一次 `nuget spec` 即可生成初始清单。更新包时, 可以更改项目中的值或直接编辑清单。

1. 打开命令提示符并导航到包含 `AppLogger.csproj` 文件的项目文件夹。
2. 运行以下命令：`nuget spec AppLogger.csproj`。通过指定项目，NuGet 会创建匹配项目名称的清单，在此示例中为 `AppLogger.nuspec`。它还会包括清单中的替换令牌。
3. 在文本编辑器中打开 `AppLogger.nuspec` 以检查其内容，其内容应如下所示：

```
<?xml version="1.0"?>
<package>
  <metadata>
    <id>Package</id>
    <version>1.0.0</version>
    <authors>YourUsername</authors>
    <owners>YourUsername</owners>
    <license type="expression">MIT</license>
    <projectUrl>http://PROJECT_URL_HERE_OR_DELETE_THIS_LINE</projectUrl>
    <iconUrl>http://ICON_URL_HERE_OR_DELETE_THIS_LINE</iconUrl>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>Package description</description>
    <releaseNotes>Summary of changes made in this release of the package.</releaseNotes>
    <copyright>Copyright 2019</copyright>
    <tags>Tag1 Tag2</tags>
  </metadata>
</package>
```

编辑清单

1. 如果尝试在 `.nuspec` 文件中创建包含默认值的包，NuGet 会产生错误，因此在继续操作之前必须编辑以下字段。请参阅 [.nuspec 文件引用 - 可选元数据元素](#)，了解如何使用它们。
 - `licenseUrl`
 - `projectUrl`
 - `iconUrl`
 - `releaseNotes`
 - 标记
2. 对于面向公共使用而生成的包，请特别注意 `Tags` 属性，因为这些标记可帮助其他人在源中（例如 `nuget.org`）查找包并了解其用途。
3. 也可以在此时向清单中添加任何其他元素，如 [.nuspec 文件引用](#) 中所述。
4. 在继续之前保存文件。

运行 pack 命令

1. 从包含 `.nuspec` 文件的文件夹中的命令提示符，运行命令 `nuget pack`。
2. NuGet 以 `identifier-version.nupkg` 形式生成 `.nupkg` 文件，你可在当前文件夹中找到该文件。

发布包

有了 `.nupkg` 文件后，可以使用 `nuget.exe` 以及从 `nuget.org` 获取的 API 密钥将其发布到 `nuget.org`。对于 `nuget.org`，必须使用 `nuget.exe` 4.1.0 或更高版本。

NOTE

所有上传到 nuget.org 的包都会进行病毒扫描，如果发现任何病毒，将拒绝包。此外，还会定期扫描 nuget.org 上列出的所有包。

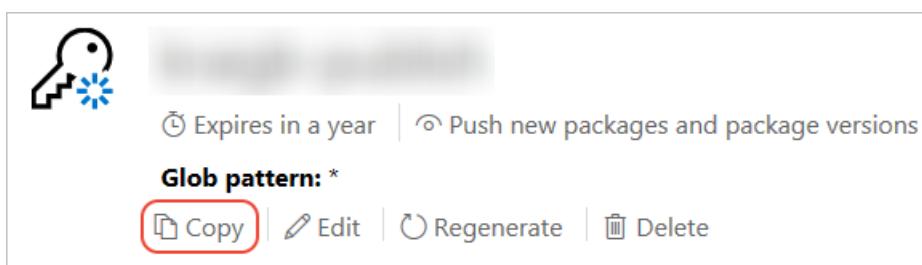
发布到 nuget.org 的包也对其他开发者公开可见，除非你取消列出它们。若要专门托管包，请参阅[托管包](#)。

获取 API 密钥

1. 登录你的 nuget.org 帐户，或创建一个帐户（如果你还没有帐户）。

有关创建帐户的详细信息，请参阅[个人帐户](#)。

2. 选择用户名（在右上角），然后选择“API 密钥”。
3. 选择“创建”，提供密钥名称，选择“选择范围”>“推送”。输入“Glob 模式”*，然后选择“创建”。（请参阅下面有关范围的详细信息。）
4. 创建密钥后，选择“复制”，检索需要在 CLI 中使用的访问密钥：



5. 重要事项：将你的密钥保存在安全位置，因为以后无法再次复制密钥。如果返回到 API 密钥页，则需要重新生成密钥以对其进行复制。如果不希望通过 CLI 推送包，还可以删除 API 密钥。

范围允许创建针对不同用途的单独 API 密钥。每个密钥都有其过期时间，并且可以将范围限定为特定包（或 glob 模式）。每个密钥还将范围限定为特定操作：新包和更新推送、仅更新推送，或者从列表中删除。通过范围限定，可以为管理组织不同包的不同人员创建 API 密钥，这样他们就只有所需的权限。有关详细信息，请参阅[范围内的 API 密钥](#)。

用 nuget push 发布

1. 打开命令行并更改到包含 .nupkg 文件的文件夹。
2. 运行以下命令，指定包名称并使用你的 API 密钥替换密钥值：

```
nuget push AppLogger.1.0.0.nupkg qz2jga8pl3dvn2akksyquwcs9ygggg4exypy3bhxy6w6x6 -Source https://api.nuget.org/v3/index.json
```

3. nuget.exe 会显示发布过程的结果：

```
Pushing AppLogger.1.0.0.nupkg to 'https://www.nuget.org/api/v2/package'...
PUT https://www.nuget.org/api/v2/package/
Created https://www.nuget.org/api/v2/package/ 6829ms
Your package was pushed.
```

请参阅[nuget push](#)。

发布错误

push 命令中的错误通常表示存在问题。例如，你可能会忘记更新项目中的版本号，因此尝试发布已存在的包。

尝试使用主机上已存在的标识符发布包时，你也会看到错误。例如，名称“AppLogger”已经存在。在这种情况下，

`push` 命令会给出以下错误：

```
Response status code does not indicate success: 403 (The specified API key is invalid, has expired, or does not have permission to access the specified package.).
```

如果你使用的是刚刚创建的有效 API 密钥，则此消息表明存在命名冲突，并未从错误的“权限”部分中将其完全清除。更改包标识符，重建项目，重新创建 `.nupkg` 文件，然后重试 `push` 命令。

管理已发布的包

从 nuget.org 上的配置文件中，选择“管理包”，查看刚刚发布的包。同样也会收到确认电子邮件。请注意，包可能需要一些时间才能编入索引并显示在可供他人查看的搜索结果中。在该时间段，包页面会显示以下消息：

⚠ This package has not been published yet. It will appear in search results and will be available for install/restore after both validation and indexing are complete. Package validation and indexing may take up to an hour. [Read more.](#)

这就是所有的操作！刚刚已将第一个 NuGet 包发布到 nuget.org，其他开发人员可在自己的项目中使用它。

如果你已在本演练中创建一个实际上并不使用的包（例如使用空的类库创建的包），则应取消列出将在搜索结果中隐藏的包：

1. 在 nuget.org 上，选择用户名（在该页的右上角），然后选择“管理包”。
2. 找到你需要在“已发布”下取消列出的包，然后选择右侧的回收站图标：

Published

You have 1 published package with a total of 0 downloads

Package ID	Owners	Downloads	Latest Version	
AppLogger	[Redacted]	0	1.0.0	

3. 在随后的页面上，清除标记有“在搜索结果中列出(包名)”的框，然后选择“保存”：

List AppLogger 1.0.0 in search results.

Unlisted packages cannot be installed directly and do not show up in search results.

Save

后续步骤

祝贺你创建第一个 NuGet 包！

创建包

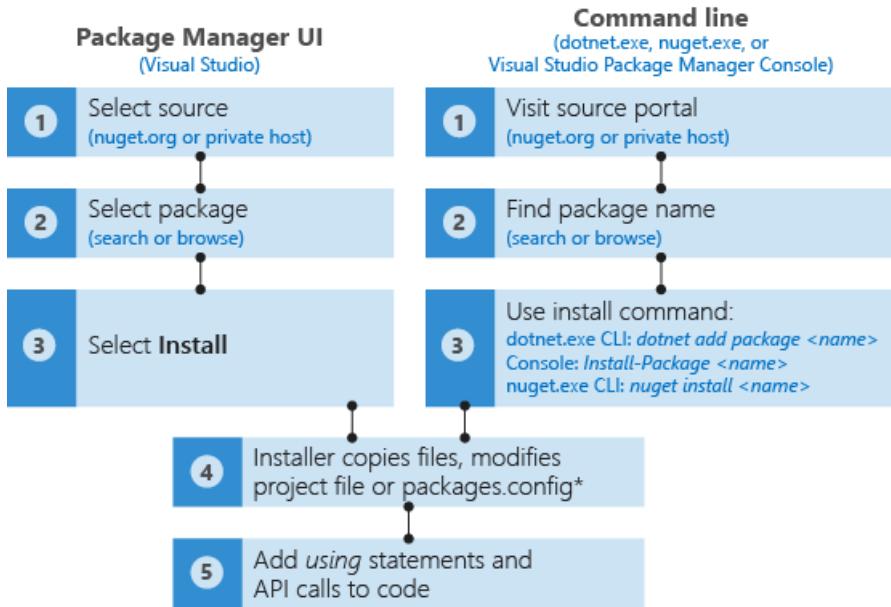
若要了解更多 NuGet 产品，请选择以下链接。

- [发布包](#)
- [预发行包](#)
- [支持多个目标框架](#)
- [包版本控制](#)
- [创建本地化包](#)

包使用工作流

2020/4/8 • [Edit Online](#)

在 nuget.org 和组织可能会建立的私有包库之间，可以找到数以万计的非常有用的包，这些包可在应用和服务中使用。无论源在哪里，使用包时都遵循相同的常规工作流。



有关详细信息，请参阅[查找和选择包](#)和[安装包时会发生什么情况？](#)。

NuGet 会记住每个已安装包的标识和版本号，并将其录制到项目文件（使用 [PackageReference](#)）或 `packages.config` 中，具体取决于项目类型和 NuGet 版本。若为 NuGet 4.0+，则 [PackageReference](#) 为首选方法，虽然这可在 Visual Studio 中通过[包管理器 UI](#) 进行配置。在任何情况下，可以随时在相应文件中进行搜索，查看项目依赖项的完整列表。

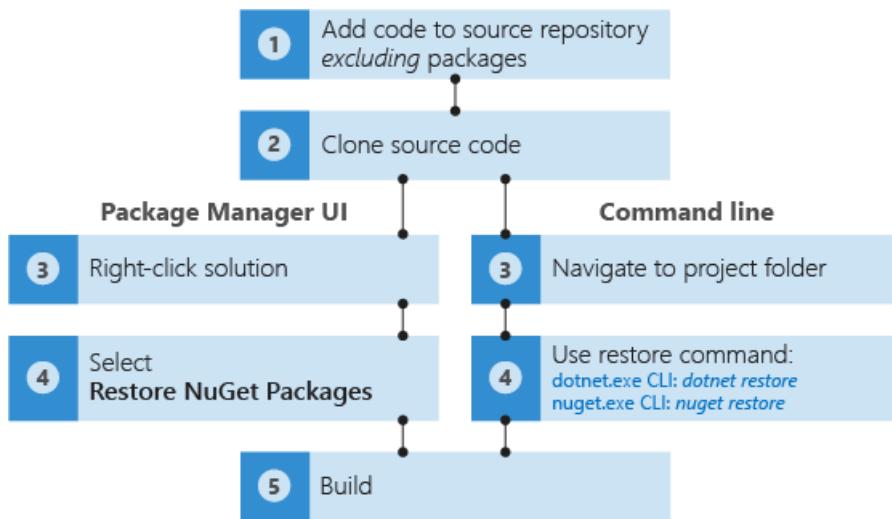
TIP

随时检查要在软件中使用的每个包的许可证是非常明智的做法。在 nuget.org 中，可以在每个包的说明页右侧找到“[许可证信息](#)”链接。如果包未指定许可条款，请直接通过包页面上的“联系所有者”链接与包所有者联系。Microsoft 不向用户授予任何第三方包提供程序的知识产权许可，同时不对第三方提供的信息承担任何责任。

安装包时，NuGet 通常会检查此包是否已存在于其缓存中。可手动从命令行中清除此缓存，如[管理全局包和缓存文件夹](#)中所述。

NuGet 还可以确保包支持的目标框架与你的项目兼容。如果包中不包含兼容的程序集，NuGet 将显示一个错误。请参阅[解决包不兼容错误](#)。

将项目代码添加到源存储库时，通常不需要包含 NuGet 包。如果用户要在后期克隆存储库或获取项目（包括 Visual Studio Team Services 等系统上的生成代理），则必须在运行生成前还原必要的包：



程序包还原使用项目文件中的信息或 `packages.config` 重新安装所有依赖项。请注意，此相关流程不尽相同，如[依赖项解析](#)中所述。此外，上图并未显示包管理器控制台的还原命令，因为已经在 Visual Studio 的上下文中使用了该控制台，它通常会自动还原包并提供所示的解决方案级别命令。

有时需要重新安装项目中已包含的包，这可能导致重新安装依赖项。使用 `nuget reinstall` 命令或使用 NuGet 包管理器控制台可轻松执行此操作。有关详细信息，请参阅[重新安装和更新包](#)。

最后一点，NuGet 的行为由 `Nuget.Config` 文件驱动。有多个文件可用于集中处理不同级别的特定设置，如[配置 NuGet 行为](#)中所述。

安装 NuGet 包的方式

使用下表中的任何方法下载和安装 NuGet 包。

<code>dotnet.exe CLI</code>	(所有平台) 用于 .NET Core 和 .NET Standard 库，以及用于面向 .NET Framework 的 SDK 样式项目的 CLI 工具(请参阅 SDK 属性)。检索由 <code><package_name></code> 标识的包，并添加对项目文件的引用。同时还要检索和安装依赖项。
Visual Studio	(Windows 和 Mac) 提供 UI，用户可以通过此 UI 浏览、选择包，并从指定包源将包及其依赖项安装到项目中。将对已安装的程序包引用添加到项目文件。 <ul style="list-style-type: none"> • 使用 Visual Studio 安装和管理包 • 在项目中包括 NuGet 包 (Mac)
包管理器控制台 (Visual Studio)	(仅限 Windows) 检索并将用 <code><package_name></code> 识别的包从所选源安装到解决方案的指定项目中，然后添加对项目文件的引用。同时还要检索和安装依赖项。
<code>nuget.exe CLI</code>	(所有平台) 用于 .NET Framework 库和面向 .NET Standard 库的非 SDK 样式项目的 CLI 工具。检索用 <code><package_name></code> 识别的包，将其内容展开到当前目录下的文件夹中；还可以检索 <code>packages.config</code> 文件中列出的所有包。同时还要检索和安装依赖项，但对项目文件或 <code>packages.config</code> 不作任何更改。

针对项目查找和评估 NuGet 包

2020/4/8 • [Edit Online](#)

启动任何 .NET 项目或识别应用或服务的功能需求时，使用可满足该需求的现有 NuGet 包可以大大地省时省力。这些包可能来自 [nuget.org](#) 中的公共集合，也可能来自组织或其他第三方提供的私有源。

查找包

访问 [nuget.org](#) 或打开 Visual Studio 中的程序包管理器 UI 时，其中会显示一个包列表，这些包按总下载次数排列。你可以立即看出在数以百万的 .NET 项目中使用次数最多的包。很可能前几页列出的某些包就适用于你的项目。

The screenshot shows the search results for NuGet packages. At the top, it says "There are 107,603 packages". There is a checked checkbox labeled "Include prerelease". Below this, three packages are listed:

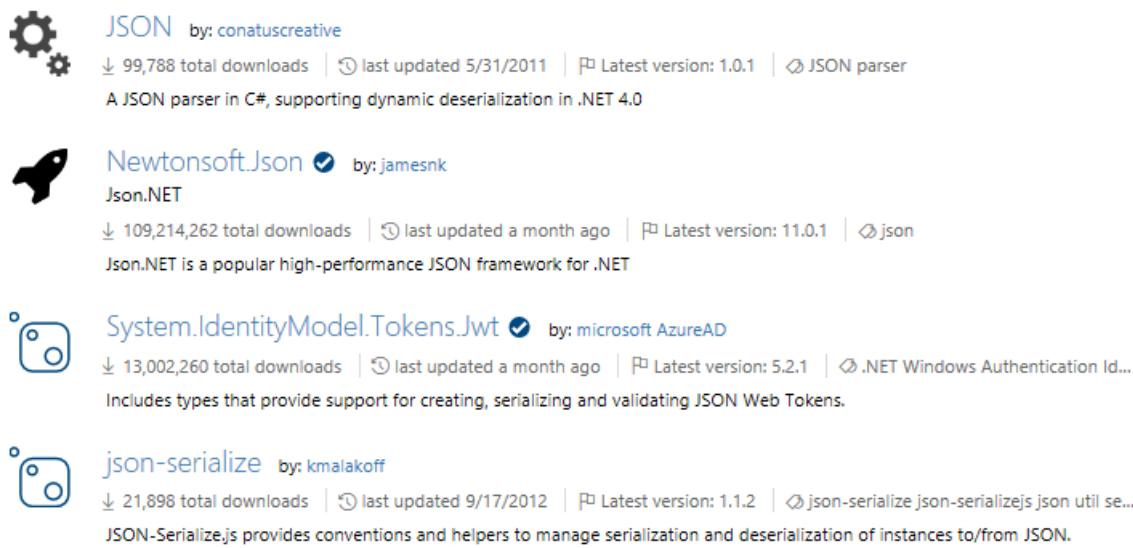
- Newtonsoft.Json** (by: jamesnk) - Json.NET: 109,214,262 total downloads, last updated a month ago, latest version: 11.0.1. Description: Json.NET is a popular high-performance JSON framework for .NET.
- NUnit** (by: rprouse charliepoole) - NUnit: 18,733,362 total downloads, last updated 4 days ago, latest version: 3.10.1. Description: NUnit features a fluent assert syntax, parameterized, generic and theory tests and is user-extensible. This package includes the NUnit 3 framework assembly, which is referenced by your tests. You will need to install version 3 of the nunit3-console program or a third-party runner that supports... [More information](#)
- .NET EntityFramework** (by: microsoft EntityFramework aspnet) - EntityFramework: 40,924,086 total downloads, last updated 5 months ago, latest version: 6.2.0. Description: Entity Framework is Microsoft's recommended data access technology for new applications.

请注意页面右上角的“包括预发行版”选项。勾选此选项时，[nuget.org](#) 将显示所有版本的包，包括 beta 版本和其他早期版本。若要仅显示稳定版本，请清除此选项。

对于特定需求，按标记搜索(在 Visual Studio 包管理器或在 [nuget.org](#) 等门户中)是发现适用包的最常用方法。例如，搜索“json”将列出具有该关键字标记的所有 NuGet 包，这些包必然与 JSON 数据格式存在某种关系。

2,335 packages returned for json

[Include prerelease](#)



The screenshot shows the search results for 'json' on nuget.org. There are four packages listed:

- JSON** by conatuscreative: A JSON parser in C#, supporting dynamic deserialization in .NET 4.0. Last updated 5/31/2011, version 1.0.1.
- Newtonsoft.Json** by jamesnk: Json.NET. Last updated a month ago, version 11.0.1. Description: Json.NET is a popular high-performance JSON framework for .NET.
- System.IdentityModel.Tokens.Jwt** by microsoft AzureAD: Includes types that provide support for creating, serializing and validating JSON Web Tokens. Last updated a month ago, version 5.2.1.
- json-serialize** by kmalakoff: JSON-Serialize.js provides conventions and helpers to manage serialization and deserialization of instances to/from JSON. Last updated 9/17/2012, version 1.1.2.

还可以使用包 ID 进行搜索(如果知道)。请参阅下面的[搜索语法](#)。

在此情况下, 搜索结果将仅按相关性进行排序, 因此通常至少需要浏览前几页结果才能找到满足需求的包;也可以优化搜索词, 使其更加具体。

包是否支持项目的目标框架?

仅当包支持的框架中包含该项目的目标框架时, NuGet 才会在项目中安装包。如果包不兼容, NuGet 将发出错误。

某些包会直接在 nuget.org 库中列出其支持的框架, 但由于这些不是必需数据, 因此许多包不会包含此列表。目前不支持在 nuget.org 中搜索支持特定目标框架的包(此功能处于考虑阶段, 请参阅 [NuGet 问题 2936](#))。

幸运的是, 你可以通过以下两种方法确定受支持的框架:

- 尝试在 NuGet 包管理器控制台中使用 [Install-Package](#) 命令将包安装到项目中。如果包不兼容, 此命令将显示包支持的框架。
- 在 nuget.org 中包的页面上, 使用“信息”下的“手动下载”链接来下载包。将扩展名从 `.nupkg` 更改为 `.zip`, 然后打开文件查看 `lib` 文件夹的内容。你会看到每个受支持框架的子文件夹, 每个子文件夹都以目标框架名字对象 (TFM)(请参阅[目标框架](#))命名。如果 `lib` 下没有任何子文件夹而只有一个 DLL, 此时则必须通过尝试将包安装到项目中来查看其是否兼容。

预发行包

许多包创建者会提供预览版和 beta 版, 他们会继续提供改进并收集其最新版本的反馈。

默认情况下, nuget.org 会在搜索结果中显示预发行包。若要仅搜索稳定版发布, 请清除页面右上角的“包括预发行版”选项

2,380 packages returned for logging

[Include prerelease](#)



NLog by: 304NotModified jkowalski Xharze

↓ 11,417,261 total downloads | last updated 17 days ago | Latest version: 4.4.13 | NLog logging log tracing logfile...

NLog is a logging platform for .NET with rich log routing and management capabilities. It can help you produce and manage high-quality logs for your application regardless of its size or complexity. This package installs NLog.dll with includes core logging functionality. For your main project also... [More information](#)



logging by: conatuscreative apitize

↓ 3,211 total downloads | last updated 6/14/2013 | Latest version: 0.9.1 | logging

A simple logging abstraction. You don't have the option of using immediate-mode logging, so you never accidentally kill performance.



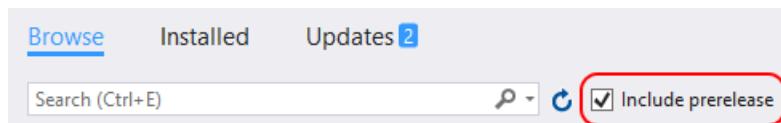
log4net by: cincura.net Apache.Logging

Apache log4net

↓ 14,841,688 total downloads | last updated 3/11/2017 | Latest version: 2.0.8 | logging log tracing logfiles

在 Visual Studio 中使用 NuGet 和 dotnet CLI 工具时，NuGet 默认不包括预发行版本。若要更改此行为，请执行以下操作：

- **Visual Studio 中的包管理器 UI:** 在“管理 NuGet 包”UI 中，勾选“包括预发行版”框。勾选或清除此框将刷新包管理器 UI 和可安装的可用版本列表。



- **包管理器控制台:** 将 `-IncludePrerelease` 开关与 `Find-Package`、`Get-Package`、`Install-Package`Sync-Package` 和 `Update-Package` 命令配合使用。请参阅 [PowerShell 参考](#)。
- **nuget.exe CLI:** 将 `-prerelease` 开关与 `install`、`update`、`delete` 和 `mirror` 命令配合使用。请参阅 [NuGet CLI 参考](#)
- **dotnet.exe CLI:** 使用 `-v` 参数指定确切的预发行版本。请参阅 [dotnet 添加包参考](#)。

本机 C++ 包

NuGet 支持本机 C++ 包，这些包可在 Visual Studio 的 C++ 项目中使用。这将启用项目的“管理 NuGet 包”上下文菜单命令、引入 `native` 目标框架，以及提供 MSBuild 集成。

若要在 [nuget.org](#) 中查找本机包，请搜索 `tag:native`。此类包通常提供 `.targets` 和 `.props` 文件，NuGet 可在将包添加到项目中时自动导入这些文件。

评估包

评估包用途的最佳方法是下载它并在你的代码中试用（顺便说一下，将对 nuget.org 的所有包进行常规病毒扫描）。每一个被广泛使用的包起初都仅被少数开发人员采用，你也可能成为早期采用者之一！

同时，使用 NuGet 包则意味着与此包建立依赖关系，因此使用者希望确保包具有耐用性和可靠性。安装和直接测试包非常耗时，所以还可以通过使用包清单页面中的信息深入了解包的质量：

- **下载统计信息:** 在 nuget.org 中，包页面上的“统计信息”部分显示总下载次数、最新版本下载次数以及日平均下载次数。数值较大则表示已有许多开发人员选择与此包建立依赖关系，这说明此包已获得认可。

Statistics

↓ 92,919,266 total downloads

↗ 10,344 downloads of latest version

↗ 36,785 downloads per day (avg)

[View full stats](#)

- GitHub 使用情况: 在包页面上, "GitHub 使用情况"部分列出了依赖此包且在 GitHub 上星级很高的公共 GitHub 存储库。GitHub 存储库的星级通常表示该存储库在 GitHub 用户当中的受欢迎程度(星级越高通常表示越受欢迎)。请访问 [GitHub 的入门页面](#), 详细了解 GitHub 的星级和存储库排名系统。

▽ GitHub Usage

Showing the top 10 GitHub repositories that depend on Microsoft.AspNetCore.Mvc:

Repository	Stars
aspnetboilerplate/aspnetboilerplate ASP.NET Boilerplate - Web Application Framework	★ 7.0K
aspnet/AspNetCore.Docs Documentation for ASP.NET Core	★ 5.4K

NOTE

包的"GitHub 使用情况"部分定期自动生成、无需人工审阅各个存储库且仅用于参考目的, 它是为了向你显示依赖此包且在 GitHub 用户当中很受欢迎的 GitHub 存储库。

- 版本历史记录: 在包页面上, 可在"信息"下查找最新更新的日期和查看"版本历史记录"。维护良好的包应具有最新更新和丰富的版本历史记录。疏于维护的包则仅具有几次更新, 并且通常已长时间未更新。

▽ Version History

Version	Downloads	Last updated
11.0.1-beta1 (current version)	11,667	12 days ago
10.0.3	5,781,529	6 months ago
10.0.2	4,745,507	8 months ago
10.0.1	11,320,059	9 months ago
9.0.1	11,284,341	6/22/2016

[+ Show more](#)

- 最近安装: 在包页面上, 选择"统计信息"下的"查看完整统计信息"。完整统计信息页面按版本号显示过去六周的包安装次数。其他开发人员频繁使用的包通常比鲜有使用的包更值得信赖。
- 支持: 在包页面上, 选择"信息"下的"项目网站" (如果可用)可查看作者提供的支持选项。具有专门网站的

项目通常具备更好的支持。

- **开发人员历史记录:** 在包页面上, 选择“所有者”下的某个所有者可查看其已发布的其他包。具有多个包的所有者更有可能在未来继续对其工作成果提供支持。
- **开源贡献:** 很多包都在开源存储库中进行维护, 这使依赖于包的开发人员可直接提供 bug 修补程序和功能改进。任何给定包的贡献历史记录也能反映出积极参与的开发人员的数量。
- **联系所有者:** 新晋开发人员同样可以制作出可供使用的优质包, 为他们提供向 NuGet 生态系统引入新内容的机会是非常好的做法。若有此想法, 使用清单页面中“信息”下的“联系所有者”选项即可直接联系包开发人员。他们很可能非常乐于与你合作来满足你的需求！
- **保留包 ID 前缀:** 许多包所有者已应用并授予了[保留包 ID 前缀](#)。如果 nuget.org 上或 Visual Studio 中的包 ID 旁有视觉对象复选标记, 这意味着包所有者已满足我们的 ID 前缀预留条件。这意味着包所有者清楚了解如何标识自己及其包。

NOTE

应时刻留意包的许可条款 - 在 nuget.org 中的包清单页面上选择“许可信息”即可查看。如果包未指定许可条款, 请直接通过包页面上的“联系所有者”链接与包所有者联系。Microsoft 不向用户授予任何第三方包提供程序的知识产权许可, 同时不对第三方提供的信息承担任何责任。

许可证 URL 弃用

在我们从 `licenseUrl` 切换到 `license` 时, 一些 NuGet 客户端和 NuGet 源可能在某些情况下尚无法提供授权信息。为了维护向后兼容性, 许可证 URL 会指向介绍如何在此类情况下检索许可证信息的文档。

如果通过单击包的许可证 URL 转到此页面, 表示包中有许可证文件; 并且

- 你已连接到尚不知道如何向客户端解释和显示新许可证信息的源; 或者
- 要使用的客户端尚不知道如何解释和读取源可能提供的新许可证信息; 或者
- 这两者的组合

下面介绍了如何读取包内的许可证文件信息:

1. 下载 NuGet 包, 并将它的内容解压缩到文件夹中。
2. 打开位于此文件夹的根目录中的 `.nuspec` 文件。
3. 它应有 `<license type="file">license\license.txt</license>` 等标记。这意味着, 许可证文件的命名为 `license.txt`, 且它位于 `license` 文件夹的根目录中。
4. 转到 `license` 文件夹, 并打开 `license.txt` 文件。

对于相当于在 `.nuspec` 中设置许可证的 MSBuild, 请查看[打包许可证表达式或许可证文件](#)。

搜索语法

NuGet 包搜索在 nuget.org 上、NuGet CLI 中和 Visual Studio 的 NuGet 包管理器扩展中具有相同的使用方法。通常可使用关键字和包说明进行搜索。

- **筛选:** 可以按照语法 `<property>:<term>` 使用搜索词来搜索特定属性, 其中, `<property>` (区分大小写) 可为 `id`、`packageid`、`version`、`title`、`tags`、`author`、`description`、`summary` 和 `owner`。可以同时搜索多个属性。按 `id` 属性搜索得到的是子字符串匹配项, 而按 `packageid` 和 `owner` 搜索将得到不区分大小写的确切匹配。示例:

使用 NuGet 包管理器在 Visual Studio 中安装和管理包

2020/4/8 • [Edit Online](#)

通过 Windows 版 Visual Studio 中的 NuGet 包管理器 UI，可轻松安装、卸载和更新项目和解决方案中的 NuGet 包。若要了解 Visual Studio for Mac 的使用体验，请参阅[在项目中包括 NuGet 包](#)。Visual Studio Code 中不包含包管理器 UI。

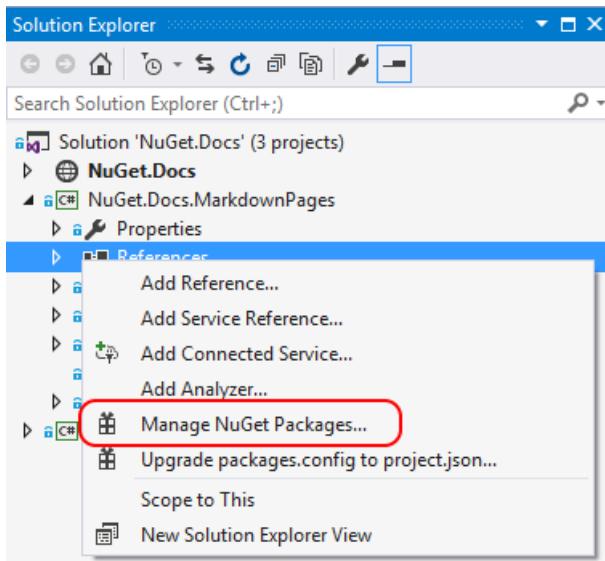
NOTE

如果 Visual Studio 2015 中缺少 NuGet 包管理器，请选中“工具”>“扩展和更新...”并搜索“NuGet 包管理器”扩展。如果无法在 Visual Studio 中使用扩展安装程序，请直接从 <https://dist.nuget.org/index.html> 下载扩展。

从 Visual Studio 2017 开始，NuGet 和 NuGet 包管理器会与任何 .NET 相关的工作负载一起自动安装。通过在 Visual Studio 安装程序中选择“单个组件”>“代码工具”>“NuGet 包管理器”选项，可以单独安装它。

查找和安装包

1. 在“解决方案资源管理器”中，右键单击“引用”或某个项目，然后选择“管理 NuGet 包...”。



2. “浏览”选项卡按当前所选来源的受欢迎程度显示包(请参阅[包源](#))。使用左上角的搜索框搜索特定包。从列表中选择一个包以显示其信息，此操作还会启用“安装”按钮以及版本选择下拉列表。

NuGet Package Manager: NuGet.Docs.MarkdownPages

Installed Updates 1

Json

Include prerelease

Package source: nuget.org

Newtonsoft.Json by James Newton-King, 32M v9.0.1
Json.NET is a popular high-performance JSON framework for .NET

JSON by Daniel Crenna, 35.2K downloads v1.0.1
A JSON parser in C#, supporting dynamic deserialization in .NET 4.0. A JSON parser in C#, su...

ServiceStack.Text by Service Stack, 1.48M dc v4.0.60
.NET's fastest JSON, JSV and CSV Text Serializers

RestSharp by John Sheehan, RestSharp Comm v105.2.3
Simple REST and HTTP API Client

json-serialize by Kevin Malakoff, 6.77K downlo v1.1.2
JSON-Serialize.js provides conventions and helpers

Newtonsoft.Json

Version: Latest stable 9.0.1 **Install**

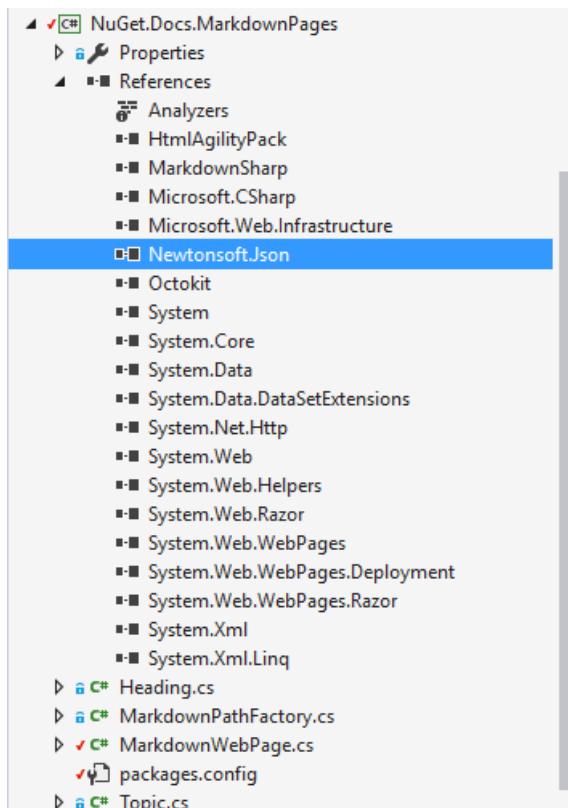
Options

Description
Json.NET is a popular high-performance JSON framework for .NET

Version: 9.0.1
Author(s): James Newton-King
License: <https://raw.githubusercontent.com/JamesNK/Newtonsoft.Json/master/LICENSE.md>

Date published: Wednesday, June 22, 2016 (6/22/2016)
Project URL: <http://www.newtonsoft.com/json>

- 从下拉列表中选择所需的版本，然后选择“安装”。Visual Studio 随即将包及其依赖项安装到项目中。系统可能会要求你接受许可条款。安装完成后，添加的包将显示在“已安装”选项卡上。包同时还列在解决方案资源管理器的“引用”节点中，表明可以使用 `using` 语句在项目中引用它们。



TIP

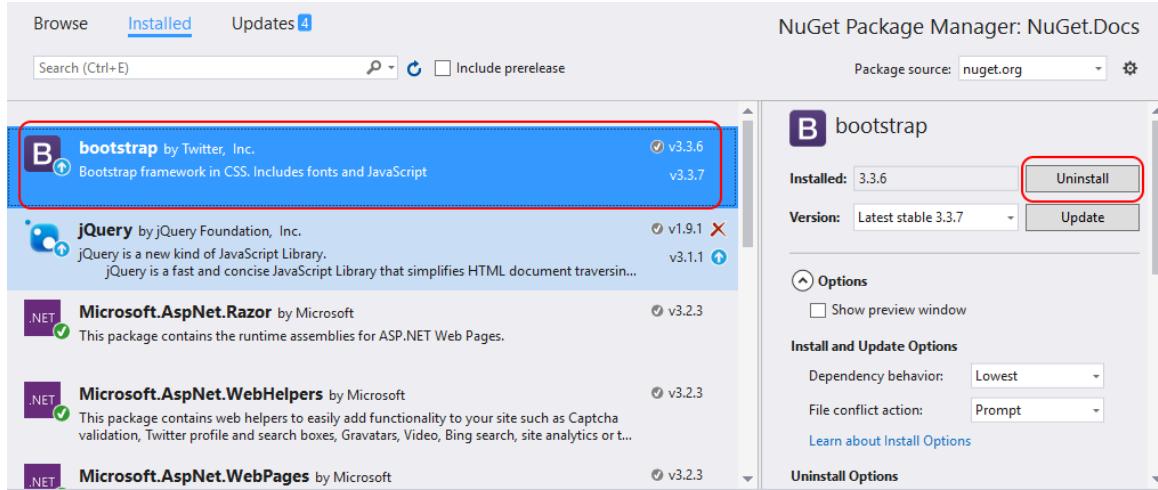
若要在搜索中包含预发布版本，并在版本下拉列表中提供预发布版本，请选中“包含预发布版本”选项。

NOTE

NuGet 提供项目可使用包的两种格式：`PackageReference` 和 `packages.config`。默认项可以在 Visual Studio 的选项窗口中设置。

卸载包

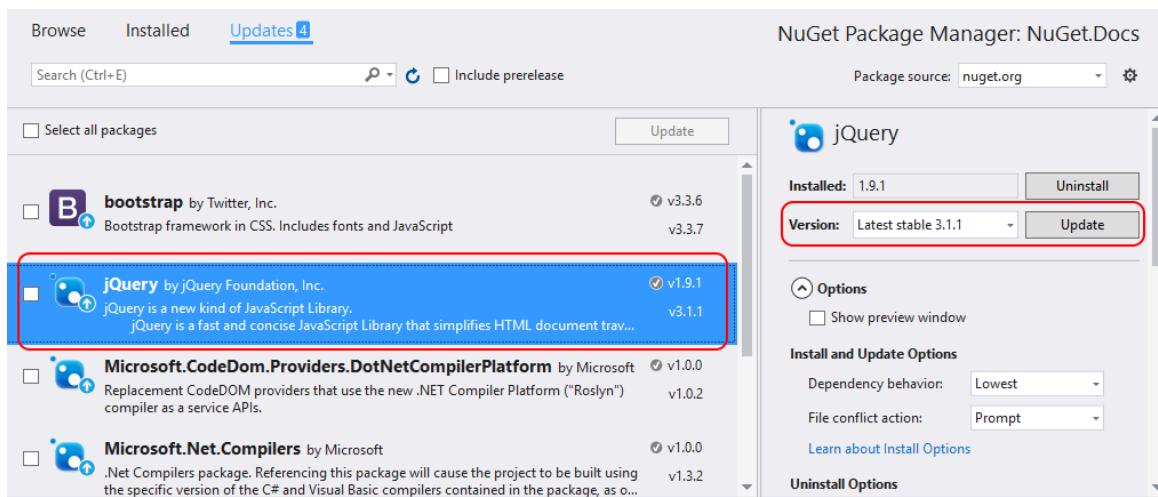
- 在“解决方案资源管理器”中，右键单击“引用”或所需项目，然后选择“管理 NuGet 包...”。
- 选择“已安装”选项卡。
- 选择要卸载的包(如有必要，使用搜索来筛选列表)并选择“卸载”。



- 请注意，在卸载包时，“包含预发布版本”和“包源”控件无效。

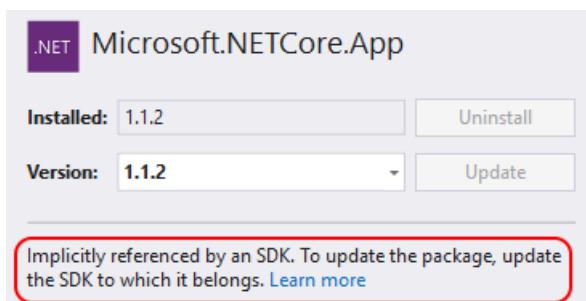
更新包

- 在“解决方案资源管理器”中，右键单击“引用”或所需项目，然后选择“管理 NuGet 包...”。(在网站项目中，右键单击“Bin”文件夹。)
- 选择“更新”选项卡，查看所选包源中包含可用更新的包。选中“包含预发布版本”，以便在更新列表中包含预发布版本的包。
- 选择要更新的包，从右侧的下拉列表中选择所需的版本，然后选择“更新”。



- 对于某些包，“更新”按钮处于禁用状态，并显示一条消息，指示它是“由 SDK 隐式引用”(或“AutoReferenced”)。此消息表明该包是较大框架或 SDK 的一部分，不能单独更新。(此类包在内部标有 `<IsImplicitlyDefined>True</IsImplicitlyDefined>`。)例如，`Microsoft.NETCore.App` 是 .NET Core SDK 的一部分，并且包版本与应用程序使用的运行时框架的版本不同。需要更新 .NET Core 安装以获取新版本的 ASP.NET Core 和 .NET Core 运行时。请参阅[本文档](#)，详细了解 .NET Core 元包和版本控制。这适用于以下常用包：
 - `Microsoft.AspNetCore.All`
 - `Microsoft.AspNetCore.App`

- Microsoft.NETCore.App
- NETStandard.Library

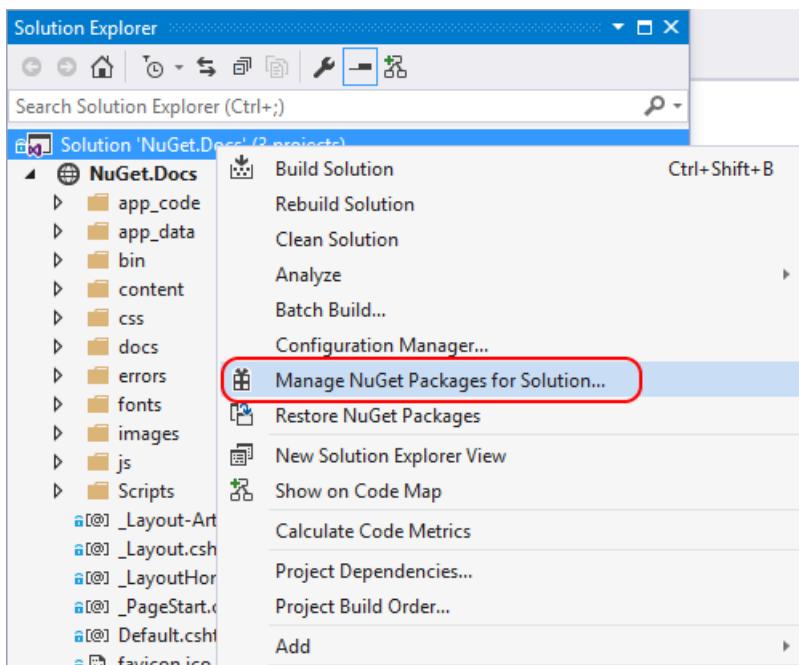


5. 若要将多个包更新到其最新版本, 请在列表中选中它们, 然后选择列表上方的“更新”按钮。
6. 还可以从“已安装”选项卡更新单个包。在这种情况下, 包的详细信息包括版本选择器(受“包含预发布版本”选项的约束)和“更新”按钮。

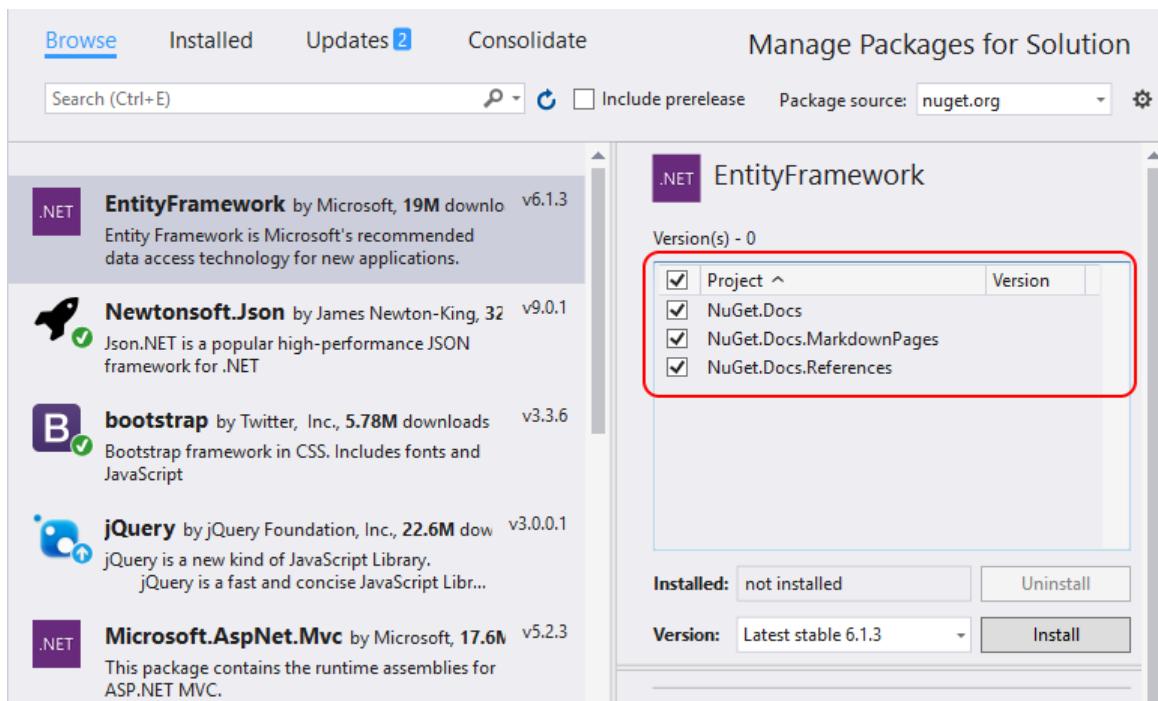
管理解决方案的包

管理解决方案的包是同时处理多个项目的便捷方式。

1. 选择“工具”>“NuGet 包管理器”>“管理解决方案的 NuGet 包...”菜单命令, 或右键单击解决方案, 然后选择“管理 NuGet 包...”:

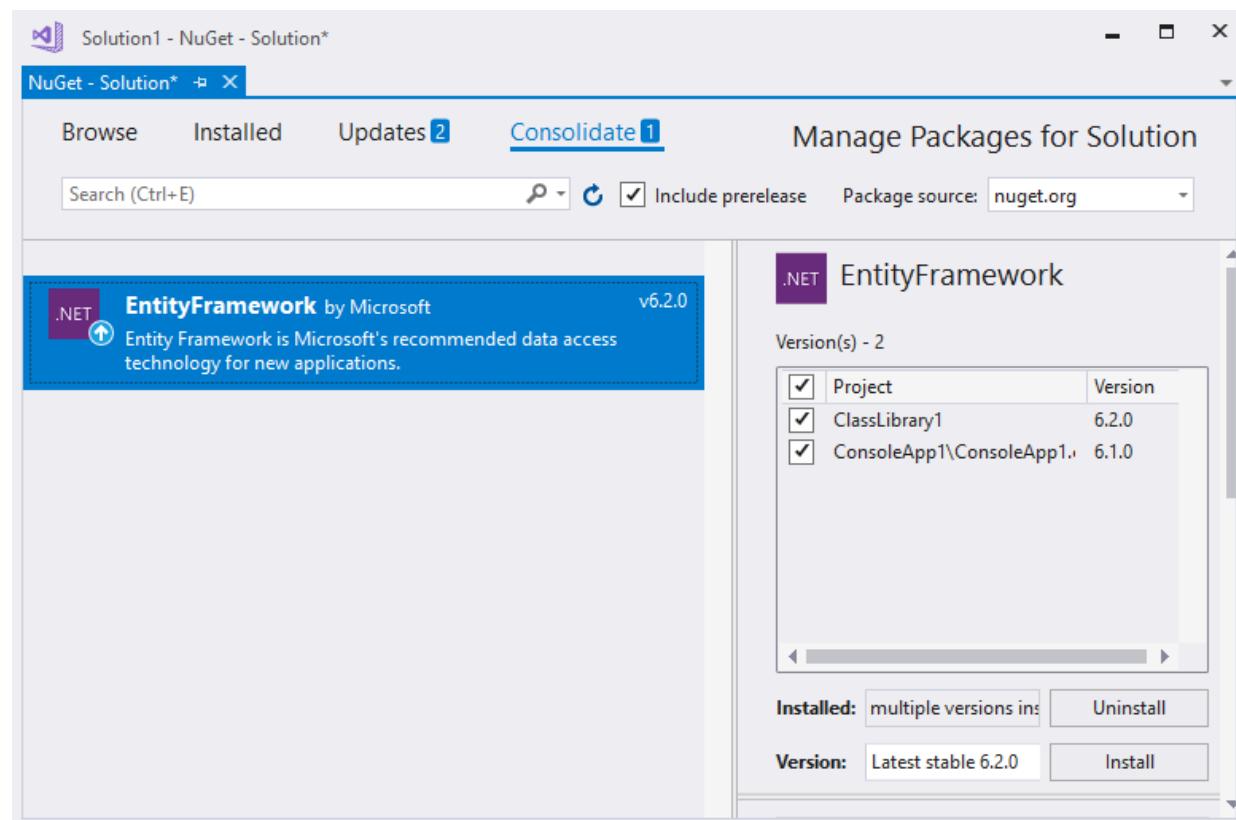


2. 管理解决方案的包时, UI 让你可以选择受操作影响的项目:



“合并”选项卡

开发人员通常认为，在同一解决方案的不同项目中使用相同 NuGet 包的不同版本的做法不好。当你选择管理解决方案的包时，包管理器 UI 提供了一个“合并”选项卡，让你可以轻松查看解决方案中不同项目使用的具有不同版本号的包：



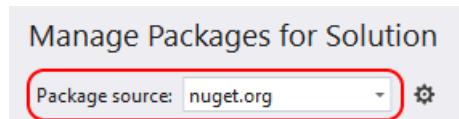
在本例中，ClassLibrary1 项目使用的是 EntityFramework 6.2.0，而 ConsoleApp1 使用的是 EntityFramework 6.1.0。若要合并包版本，请执行以下操作：

- 在项目列表中选择要更新的项目。
- 选择要在“版本”控件中的所有项目中使用的版本，例如 EntityFramework 6.2.0。
- 选择“安装”按钮。

包管理器将选定的包版本安装到所有选定的项目中，之后包不再显示在“合并”选项卡上。

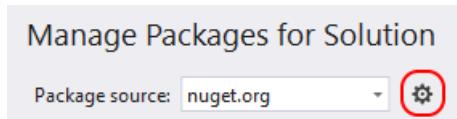
包源

若要更改 Visual Studio 从中获取包的源, 请从源选择器中选择一个源:

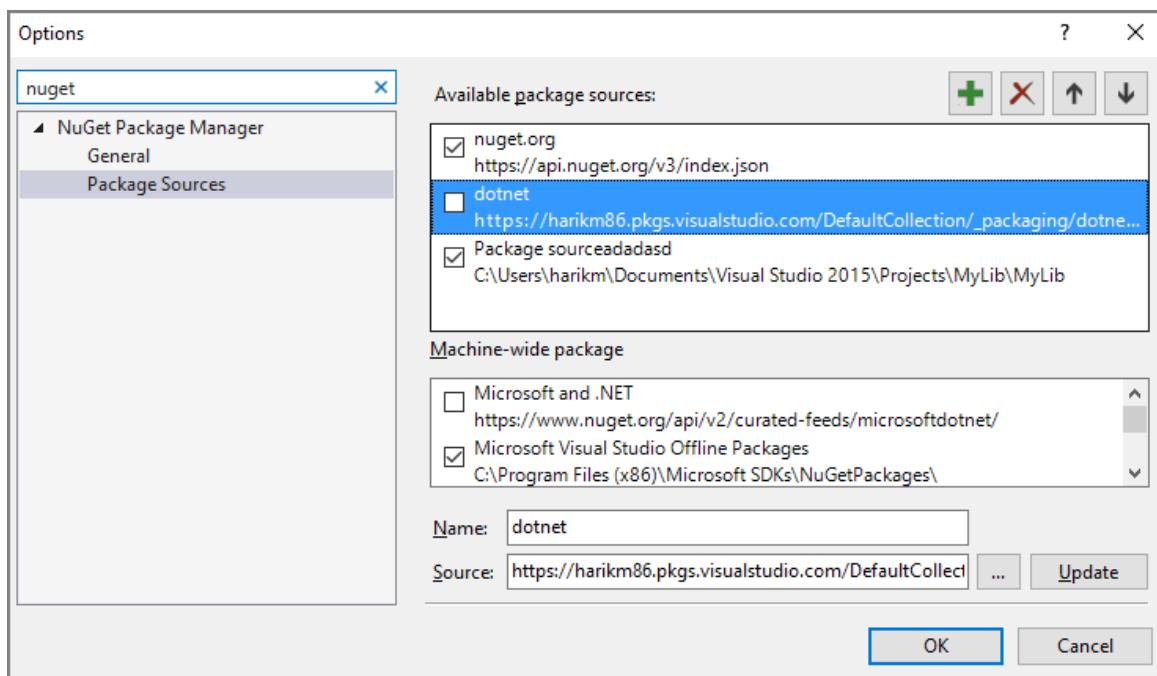


管理包源:

1. 在下面的包管理器 UI 中选择“设置”图标, 或使用“工具”>“选项”命令并滚动到“NuGet 包管理器”:



2. 选择“包源”节点:



3. 要添加源, 请选择“+”, 编辑名称, 在“源”控件中输入 URL 或路径, 然后选择“更新”。选择器下拉列表中现在显示源。
4. 若要更改包源, 请选中它, 在“名称”和“源”框中进行编辑, 然后选择“更新”。
5. 若要禁用包源, 请清除列表中名称左侧的框。
6. 若要删除包源, 请选中它, 然后选择“X”按钮。
7. 使用向上和向下箭头按钮不会更改包源的优先级顺序。Visual Studio 会忽略包源的顺序, 使用来自任何首先响应请求的源的包。有关详细信息, 请参阅[包源](#)。

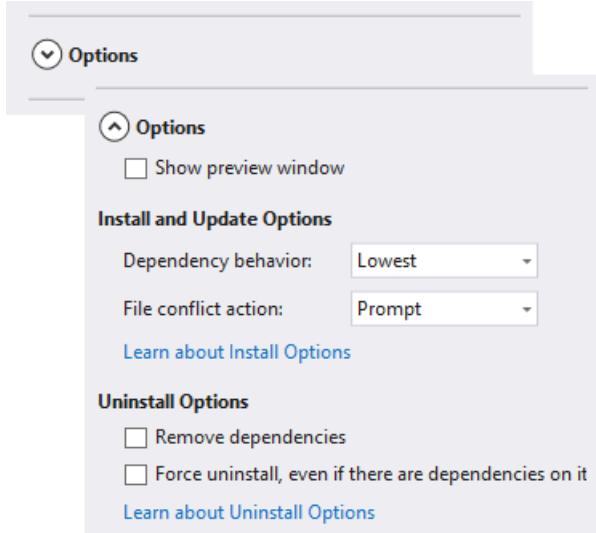
TIP

如果在删除某个包源后, 该包源重新出现, 则它可能会列在计算机级或用户级 `NuGet.Config` 文件中。有关这些文件的位置, 请参阅[常见 NuGet 配置](#), 然后通过手动编辑文件或使用 `nuget` 源命令删除源。

包管理器“选项”控件

选择包后, 包管理器 UI 会在版本选择器下方显示一个可扩展的“选项”小控件(此处显示为折叠和展开)。请注意

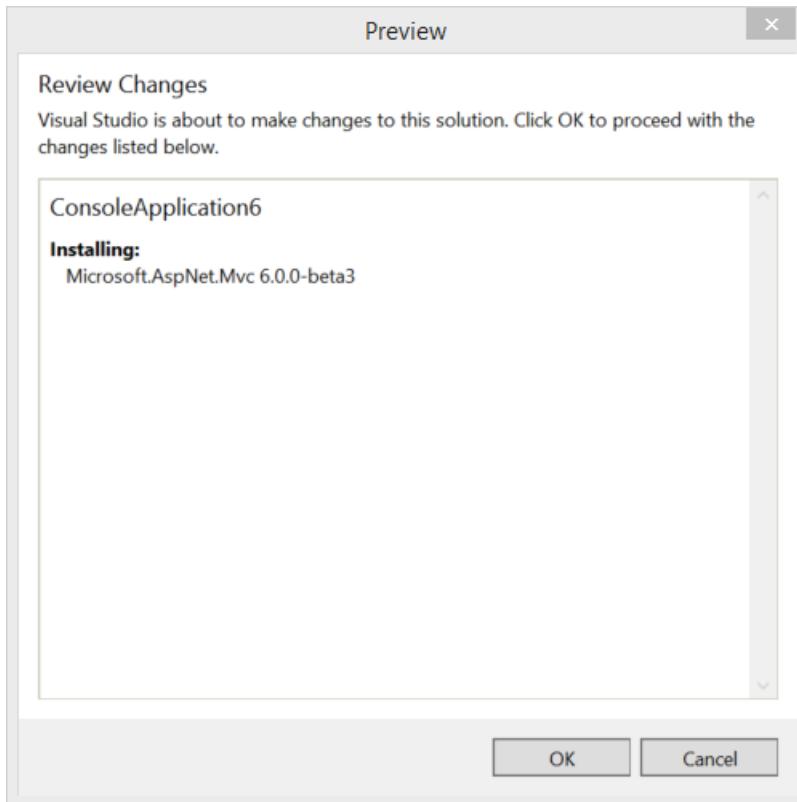
意，对于某些项目类型，仅提供“显示预览窗口”选项。



以下各节介绍了这些选项。

显示预览窗口

选中此选项后，模式窗口将显示安装包之前所选包的依赖项：



安装与更新选项

(并非适用于所有项目类型。)

“依赖项行为”用于配置 NuGet 如何决定要安装哪些版本的依赖包：

- “忽略依赖项”会跳过安装任何依赖项，这通常会破坏正在安装的软件包。
- 如果选择“最低”[默认选项]，则安装具有可满足主要选定包要求的最小版本号的依赖项。
- 如果选择“最高版本的修补程序”，则安装的版本的主要版本号和次要版本号相同，但修补程序版本号最高。
例如，如果指定版本 1.2.2，则会安装以 1.2 开头的最高版本
- 如果选择“最高次要版本”，则安装的版本的主要版本号相同，但次要版本号和修补程序版本号最高。如果指定版本 1.2.2，则会安装以 1 开头的最高版本

- 选择“最高”可安装包的最高可用版本。

“文件冲突操作”指定 NuGet 应如何处理项目或本地计算机中已存在的包：

- “提示”指示 NuGet 询问是保留还是覆盖现有包。
- “全部忽略”指示 NuGet 跳过覆盖任何现有包。
- “全部覆盖”指示 NuGet 覆盖任何现有包。

卸载选项

(并非适用于所有项目类型。)

“删除依赖项”：如果选中，则删除任何依赖包（如果它们未在项目中的其他位置引用）。

“在存在依赖项时仍强制卸载”：选中后，即使在项目中仍然引用了该包，也会卸载它。此选项通常与“删除依赖项”一起选中，用于删除包及其安装的任何依赖项。不过，使用此选项可能会导致项目中的引用中断。在这种情况下，可能需要[重新安装其他包](#)。

使用 dotnet CLI 安装和管理包

2020/4/8 • [Edit Online](#)

通过 CLI 工具可轻松安装、卸载和更新项目和解决方案中的 NuGet 包。它可在 Windows、Mac OS X 和 Linux 上运行。

dotnet CLI 适用于 .NET Core 和 .NET Standard 项目 (SDK 样式的项目类型)，以及任何其他 SDK 样式项目 (例如，面向 .NET Framework 的 SDK 样式项目)。有关更多信息，请参阅 [SDK 属性](#)。

本文介绍了一些最常见的 dotnet CLI 命令的基本用法。对于这些中的大多数命令，CLI 工具在当前目录中查找项目文件，除非在命令中指定了项目文件 (项目文件是一个可选开关)。如需获取命令的完整列表和可能使用的参数，请参阅 [.NET Core 命令行界面 \(CLI\) 工具](#)。

必备条件

- [.NET Core SDK](#) 提供 `dotnet` 命令行工具。从 Visual Studio 2017 开始，dotnet CLI 将自动随任何与 .NET Core 相关的工作负载一起安装。

安装包

`dotnet add package` 添加对项目文件的包引用，然后运行 `dotnet restore` 以安装包。

1. 打开命令行并切换到包含项目文件的目录。
2. 运行以下命令安装 Nuget 包：

```
dotnet add package <PACKAGE_NAME>
```

例如，若要安装 `Newtonsoft.Json` 包，请使用以下命令

```
dotnet add package Newtonsoft.Json
```

3. 命令完成后，查看项目文件以确保已安装该包。

可以打开 `.csproj` 文件以查看添加的引用：

```
<ItemGroup>
<PackageReference Include="Newtonsoft.Json" Version="12.0.1" />
</ItemGroup>
```

安装特定版本的包

如果未指定版本，NuGet 将安装最新版本的包。还可以使用 `dotnet add package` 命令安装特定版本的 Nuget 包：

```
dotnet add package <PACKAGE_NAME> -v <VERSION>
```

例如，要添加 `Newtonsoft.Json` 包的 12.0.1 版，请使用以下命令：

```
dotnet add package Newtonsoft.Json -v 12.0.1
```

NuGet 包引用

可以使用 [dotnet list package](#) 命令列出项目的包引用。

```
dotnet list package
```

删除包

使用 [dotnet remove package](#) 命令从项目文件中移除包引用。

```
dotnet remove package <PACKAGE_NAME>
```

例如, 要移除 `Newtonsoft.Json` 包, 请使用以下命令

```
dotnet remove package Newtonsoft.Json
```

更新包

除非指定包版本, 否则 NuGet 会在使用 [dotnet add package](#) 命令时安装最新版本的包 (`-v` 开关)。

还原包

使用 [dotnet restore](#) 命令还原项目文件中列出的包(请参阅 [PackageReference](#))。使用 .NET Core 2.0 和更高版本, 通过 `dotnet build` 和 `dotnet run` 自动完成还原。从 NuGet 4.0 开始, 它运行与 `nuget restore` 相同的代码。

与其他 `dotnet` CLI 命令一样, 先打开命令行并切换到包含项目文件的目录。

要使用 `dotnet restore` 还原包:

```
dotnet restore
```

使用 nuget.exe CLI 管理包

2020/4/8 • [Edit Online](#)

通过 CLI 工具可轻松更新和还原项目和解决方案中的 NuGet 包。该工具提供 Windows 上的所有 NuGet 功能以及 Mac 和 Linux 上在 Mono 下运行时的大多数功能。

nuget.exe CLI 适用于 .NET Framework 项目和非 SDK 样式项目(例如, 面向 .NET Standard 库的非 SDK 样式项目)。如果你使用的是已迁移到 PackageReference 的非 SDK 样式项目, 请改用 dotnet CLI。nuget.exe CLI 需要 packages.config 文件来进行包引用。

NOTE

在大多数情况下, 建议将使用 的非 SDK 样式项目迁移至 PackageReference packages.config , 然后可以使用 dotnet CLI 而不是 nuget.exe CLI。目前, C++ 和 ASP.NET 项目无法进行迁移。

本文介绍了一些最常见的 nuget.exe CLI 命令的基本用法。对于大多数这些命令, CLI 工具在当前目录中查找项目文件, 除非在命令中指定了项目文件。有关命令和可能使用的参数的完整列表, 请参阅 [nuget.exe CLI 参考](#)。

必备条件

- 要安装 nuget.exe CLI, 从 [nuget.org](#) 下载它, 将 .exe 文件保存到合适的文件夹, 然后将该文件夹添加到 PATH 环境变量中。

安装包

install 命令使用指定的包源将包下载并安装到项目中, 默认为当前文件夹。将新包安装到项目根目录的“包”文件夹中。

IMPORTANT

install 命令不会修改项目文件或 packages.config ;在这种方式下, 它类似于 restore , 因为它只向磁盘添加包, 而不更改项目的依赖项。要添加依赖项, 请通过 Visual Studio 中的包管理器 UI 或控制台添加包, 或修改 packages.config , 然后运行 install 或 restore 。

1. 打开命令行并切换到包含项目文件的目录。

2. 使用以下命令将 NuGet 包安装到“包”文件夹。

```
nuget install <packageID> -OutputDirectory packages
```

要将 Newtonsoft.Json 包安装到“包”文件夹, 请使用以下命令:

```
nuget install Newtonsoft.Json -OutputDirectory packages
```

或者可以使用以下命令将现有 packages.config 文件的 NuGet 包安装到“包”文件夹。该操作不会将包添加到项目依赖项中, 而是在本地安装它。

```
nuget install packages.config -OutputDirectory packages
```

安装特定版本的包

如果在使用 `install` 命令时未指定版本, NuGet 将安装最新版本的包。还可以安装特定版本的 NuGet 包:

```
nuget install <packageID | configFilePath> -Version <version>
```

例如, 要添加 `Newtonsoft.Json` 包的 12.0.1 版, 请使用以下命令:

```
nuget install Newtonsoft.Json -Version 12.0.1
```

有关 `install` 的限制和行为的详细信息, 请参阅[安装包](#)。

删除包

要删除一个或多个包, 请从“包”文件夹中删除要删除的包。

如果要重新安装包, 请使用 `restore` 或 `install` 命令。

列出包

可以使用 `list` 命令显示给定源的包列表。使用 `-Source` 选项限制搜索。

```
nuget list -Source <source>
```

例如, 列出“包”文件夹中的包。

```
nuget list -Source C:\Users\username\source\repos\MyProject\packages
```

如果使用搜索词, 则搜索包括包名称、标记和包描述。

```
nuget list <search term>
```

更新单个包

除非指定包版本, 否则 NuGet 会在使用 `install` 命令时安装最新版本的包。

更新所有包

使用 `update` 命令更新所有包。将项目中的所有包(使用 `packages.config`)更新为其最新可用版本。建议在运行 `restore` 之前运行 `update`。

```
nuget update
```

还原包

使用 `restore` 命令可下载并安装“包”文件夹中缺少的所有包。

对于迁移到 PackageReference 的项目，请使用 `msbuild -t:restore` 还原程序包。

`restore` 仅将包添加到磁盘，但不会更改项目的依赖项。要还原项目依赖项，请修改 `packages.config`，然后使用 `restore` 命令。

与其他 `nuget.exe` CLI 命令一样，先打开命令行并切换到包含项目文件的目录。

要使用 `restore` 还原包：

```
nuget restore MySolution.sln
```

获取 CLI 版本

使用此命令：

```
nuget help
```

帮助输出中的第一行显示版本。若要避免向上滚动，请改用 `nuget help | more`。

在 Visual Studio 中使用包管理器控制台安装和管理包 (PowerShell)

2020/4/8 • [Edit Online](#)

借助 NuGet 包管理器控制台，可以使用 [NuGet PowerShell 命令查找、安装、卸载和更新 NuGet 包](#)。如果包管理器 UI 未提供执行操作的方法，则必须使用控制台。若要在控制台中使用 `nuget.exe` CLI 命令，请参阅[在控制台中使用 nuget.exe CLI](#)。

Windows 版 Visual Studio 中内置了该控制台。Visual Studio for Mac 或 Visual Studio Code 中未提供该控制台。

查找和安装包

例如，通过三个简单的步骤查找和安装包：

1. 在 Visual Studio 中打开项目/解决方案，然后使用“工具”>“NuGet 包管理器”>“包管理器控制台”命令打开控制台。
2. 找到要安装的包。如果你已经知道此操作步骤，请跳至步骤 3。

```
# Find packages containing the keyword "elmah"
Find-Package elmah
```

3. 运行安装命令：

```
# Install the Elmah package to the project named MyProject.
Install-Package Elmah -ProjectName MyProject
```

IMPORTANT

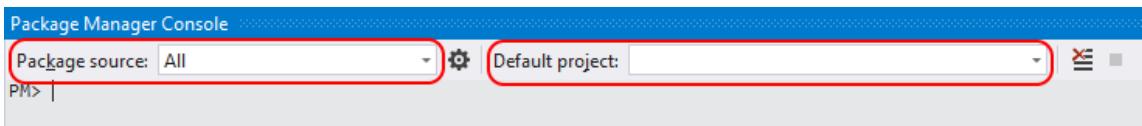
控制台中可用的全部操作也可以通过 [NuGet CLI](#) 完成。但是，控制台命令在 Visual Studio 和已保存的项目/解决方案的上下文中运行，并且通常比其等效的 CLI 命令完成更多操作。例如，通过控制台安装包会添加对项目的引用，而 CLI 命令则不会执行此操作。因此，在 Visual Studio 中工作的开发人员通常更愿意使用控制台而不是 CLI。

TIP

许多控制台操作依赖于在 Visual Studio 中通过已知路径名打开解决方案。如果你有未保存的解决方案或没有解决方案，可以看到错误：“解决方案未打开或未保存。请确保你有一个已打开且已保存的解决方案。”这表示控制台无法确定解决方案文件夹。保存未保存的解决方案，或者如果没有打开解决方案，则创建并保存解决方案，这些操作应该可以纠正错误。

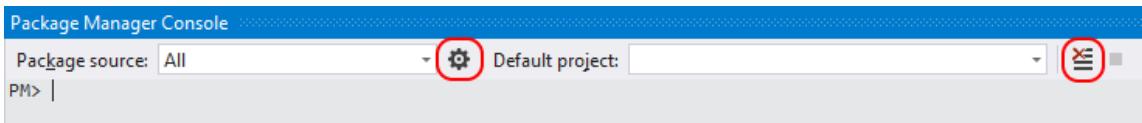
打开控制台和控制台控件

1. 在 Visual Studio 中使用“工具”>“NuGet 包管理器”>“包管理器控制台”命令打开控制台。控制台是一个 Visual Studio 窗口，可以根据需要进行排列和放置(请参阅[在 Visual Studio 中自定义窗口布局](#))。
2. 默认情况下，控制台命令针对窗口顶部控件中设置的特定包源和项目执行操作：

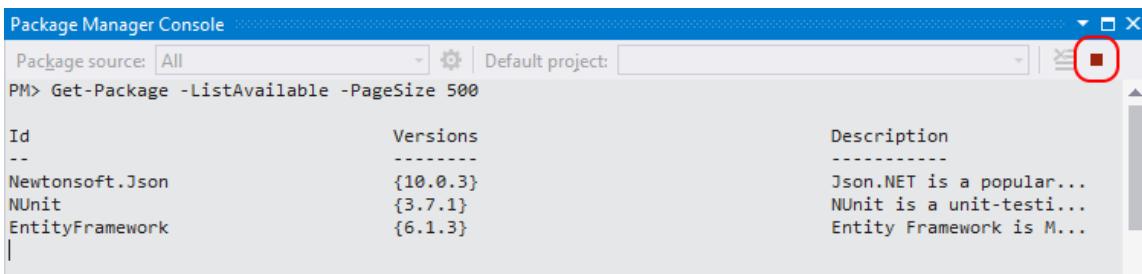


3. 选择不同的包源和/或项目会更改后续命令的默认值。要在不更改默认值的情况下覆盖这些设置，大多数命令都支持 `-Source` 和 `-ProjectName` 选项。

4. 若要管理包源，请选择齿轮图标。这是“工具”>“选项”>“NuGet 包管理器”>“包源”对话框的快捷方式，如[包管理器 UI](#)页中所述。此外，项目选择器右侧的控件可清除控制台的内容：



5. 最右边的按钮会中断长时间运行的命令。例如，运行 `Get-Package -ListAvailable -PageSize 500` 会列出默认源（例如 nuget.org）上的前 500 个包，这可能需要几分钟才能运行完毕。



安装包

```
# Add the Elmah package to the default project as specified in the console's project selector
Install-Package Elmah

# Add the Elmah package to a project named UtilitiesLib that is not the default
Install-Package Elmah -ProjectName UtilitiesLib
```

请参阅 [Install-Package](#)。

在控制台中安装包执行的步骤与[安装包时会发生什么情况](#)相同，只不过它添加了以下内容：

- 控制台在其窗口中显示适用的许可条款，并附带隐含协议。如果你不同意这些条款，应立即卸载包。
- 此外，对包的引用也会添加到项目文件中，并显示在“引用”节点下的“解决方案资源管理器”中，需要保存项目才能直接查看项目文件中的更改。

卸载包

```
# Uninstalls the Elmah package from the default project
Uninstall-Package Elmah

# Uninstalls the Elmah package and all its unused dependencies
Uninstall-Package Elmah -RemoveDependencies

# Uninstalls the Elmah package even if another package depends on it
Uninstall-Package Elmah -Force
```

请参阅 [Uninstall-Package](#)。如果需要查找标识符，请使用 `Get-Package` 查看当前安装在默认项目中的所有包。

卸载包将执行以下操作：

- 从项目中删除对包的引用(以及正在使用的任何管理格式)。引用不再出现在“解决方案资源管理器”中。
(可能需要重建项目才能看到它已从 Bin 文件夹中删除。)
- 安装包后，撤销对 `app.config` 或 `web.config` 的任何更改。
- 如果没有其余包使用这些依赖项，则删除以前安装的依赖项。

更新包

```
# Checks if there are newer versions available for any installed packages
Get-Package -updates

# Updates a specific package using its identifier, in this case jQuery
Update-Package jQuery

# Update all packages in the project named MyProject (as it appears in Solution Explorer)
Update-Package -ProjectName MyProject

# Update all packages in the solution
Update-Package
```

请参阅 [Get-Package](#) 和 [Update-Package](#)

查找包

```
# Find packages containing keywords
Find-Package elmah
Find-Package logging

# List packages whose ID begins with Elmah
Find-Package Elmah -StartWith

# By default, Get-Package returns a list of 20 packages; use -First to show more
Find-Package logging -First 100

# List all versions of the package with the ID of "jquery"
Find-Package jquery -AllVersions -ExactMatch
```

请参阅 [Find-Package](#)。在 Visual Studio 2013 及更早版本中，请改用 [Get-Package](#)。

控制台的可用性

从 Visual Studio 2017 开始，当你选择任何与 .NET 相关的工作负载时，会自动安装 NuGet 和 NuGet 包管理器；不过，也可以通过在 Visual Studio 安装程序中选中“单个组件”>“代码工具”>“NuGet 包管理器”选项来单独安装它。

另外，如果你在 Visual Studio 2015 及更早版本中缺少 NuGet 包管理器，请选中“工具”>“扩展和更新...”并搜索“NuGet 包管理器”扩展。如果无法在 Visual Studio 中使用扩展安装程序，可以直接从 <https://dist.nuget.org/index.html> 下载扩展。

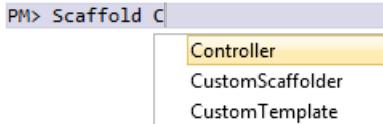
Visual Studio for Mac 目前不提供包管理器控制台。但是，可以通过 [NuGet CLI](#) 获取等效命令。Visual Studio for Mac 确实有一个用于管理 NuGet 包的 UI。请参阅[在项目中包括 NuGet 包](#)。

Visual Studio Code 中不包含包管理器控制台。

扩展包管理器控制台

某些包为控制台安装新命令。例如，`MvcScaffolding` 创建如下所示的 `Scaffold` 命令，用于生成 ASP.NET MVC 控制器和视图：

```
PM> Install-Package MvcScaffolding
'T4Scaffolding (>= 1.0.0)' not installed. Attempting to retrieve dependency from
source...
Done.
Successfully installed 'T4Scaffolding 1.0.0'.
Successfully installed 'MvcScaffolding 1.0.0'.
Successfully added 'T4Scaffolding 1.0.0' to MvcApplication5.
Successfully added 'MvcScaffolding 1.0.0' to MvcApplication5.
```



设置 NuGet PowerShell 配置文件

通过 PowerShell 配置文件，可以在每次使用 PowerShell 时使用常用命令。NuGet 支持通常位于以下位置的 NuGet 特定配置文件：

```
%UserProfile%\Documents\WindowsPowerShell\NuGet_profile.ps1
```

要查找配置文件，请在控制台中键入 `$profile`：

```
$profile
C:\Users<user>\Documents\WindowsPowerShell\NuGet_profile.ps1
```

有关更多详细信息，请参阅 [Windows PowerShell 配置文件](#)。

在控制台中使用 nuget.exe CLI

若要在包管理器控制台中使用 [nuget.exe CLI](#)，请从控制台安装 [NuGet.CommandLine](#) 包：

```
# Other versions are available, see https://www.nuget.org/packages/NuGet.CommandLine/
Install-Package NuGet.CommandLine -Version 4.4.1
```

使用“程序包还原”还原程序包

2020/4/8 • [Edit Online](#)

为了促成更干净的开发环境并减少存储库大小, NuGet“程序包还原”安装了项目文件或 `packages.config` 中列出的所有项目依赖项。.NET Core 2.0+ `dotnet build` 和 `dotnet run` 命令执行自动程序包还原。Visual Studio 可以在构建项目时自动还原包, 并且你可以随时通过 Visual Studio、`nuget restore`、`dotnet restore` 和 Mono 的 `xbuild` 来还原包。

程序包还原确保所有项目的依赖项都可用, 无需将其存储在源代码管理中。要配置源代码管理存储库以排除包二进制文件, 请参阅[包和源代码管理](#)。

程序包还原概述

程序包还原首先根据需要安装项目的直接依赖项, 然后在整个依赖项关系图中安装这些包的所有依赖项。

如果尚未安装包, NuGet 首先尝试从[缓存](#)中检索它。如果包不在缓存中, NuGet 将尝试从 Visual Studio 的“工具”>“选项”>“NuGet 包管理器”>“包源”下的列表中的所有已启用的源下载包。在还原期间, NuGet 会忽略包源的顺序, 使用来自最先响应请求的源的包。有关 NuGet 的行为方式的详细信息, 请参阅[常见的 NuGet 配置](#)。

NOTE

在检查完所有源之前, NuGet 不会指示包还原失败。届时, NuGet 仅会报告列表中最后一个源的失败。该错误说明包已不在其他任意源上, 尽管没有单独显示这些源的错误。

还原包

程序包还原试图将所有程序包依赖项安装到与项目文件 (.csproj) 或 packages.config 文件中的程序包引用相匹配的正确状态。(在 Visual Studio 中, 这些引用位于解决方案资源管理器的“依赖项 \ NuGet”或“引用”节点中。)

1. 若项目文件中的程序包引用正确, 则使用你的首选工具还原程序包。

- [Visual Studio \(自动还原或手动还原\)](#)
- [dotnet CLI](#)
- [nuget.exe CLI](#)
- [MSBuild](#)
- [Azure Pipelines](#)
- [Azure DevOps Server](#)

若项目文件 (.csproj) 或 packages.config 文件中的程序包引用不正确(它们与使用程序包还原时所需的状态不匹配), 则需要安装或更新程序包。

对于使用 PackageReference 的项目, 在成功还原后, global-packages 文件夹应显示此包, 并且会重新创建 `obj/project.assets.json` 文件。对于使用 `packages.config` 的项目, 项目的 `packages` 文件夹应显示此程序包。该项目现在应能成功生成。

2. 运行程序包还原后, 若仍出现程序包缺失的情况或与程序包相关的错误(如 Visual Studio 的解决方案资源管理器中出现错误图标), 可能需要按照[程序包还原错误疑难解答](#)中的步骤操作, 或[重新安装和更新程序包](#)。

Visual Studio 中的程序包管理器控制台提供了多个灵活的选项, 用于重新安装程序包。请参阅[使用 Package-Update](#)。

使用 Visual Studio 进行还原

在 Windows 上的 Visual Studio 中：

- 自动还原程序包，或
- 手动还原程序包

使用 Visual Studio 自动还原程序包

从模板创建项目或生成项目时，是否自动执行程序包还原取决于[启用和禁用程序包还原](#)中的选项。此外，在 NuGet 4.0+ 中，对 SDK 样式的项目（通常是 .NET Core 或 .NET Standard 项目）进行更改时还会自动进行还原。

1. 按照以下方式启用自动程序包还原：选择“工具”>“选项”>“NuGet 程序包管理器”，然后在“程序包还原”下选择“在 Visual Studio 中生成期间自动检查缺少的程序包”。

对于非 SDK 样式项目，首先需要选择“允许 NuGet 下载缺少的程序包”，以启动自动还原选项。

2. 生成项目。

如果仍未正确安装一个或多个单独的包，“解决方案资源管理器”会显示错误图标。右键单击并选择“管理 NuGet 包”，然后使用“包管理器”卸载并重新安装受影响的包。有关详细信息，请参阅[重新安装和更新包](#)。

如果看到错误“此项目引用此计算机上缺少的 NuGet 包”或者“一个或更多 NuGet 包需要还原但无法还原，因为未授予许可”，则[启用自动还原](#)。对于旧项目，亦请参阅[迁移到自动程序包还原](#)。另请参阅[程序包还原疑难解答](#)。

使用 Visual Studio 手动还原程序包

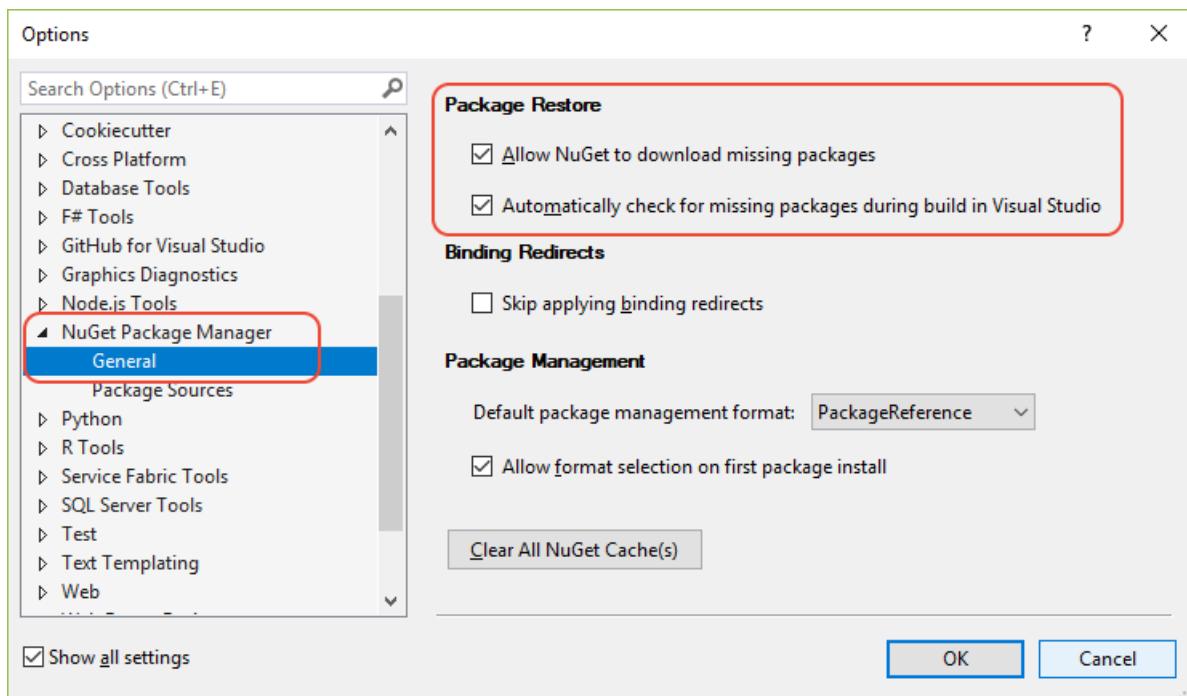
1. 选择“工具”>“选项”>“NuGet 程序包管理器”，以启用程序包还原。在“程序包还原”选项下，选择“允许 NuGet 下载缺少的程序包”。
2. 在“解决方案资源管理器”中，右键单击解决方案并选择“还原 NuGet 程序包”。

如果仍未正确安装一个或多个单独的包，“解决方案资源管理器”会显示错误图标。右键单击并选择“管理 NuGet 程序包”，然后使用“程序包管理器”卸载并重新安装受影响的程序包。有关详细信息，请参阅[重新安装和更新包](#)。

如果看到错误“此项目引用此计算机上缺少的 NuGet 包”或者“一个或更多 NuGet 包需要还原但无法还原，因为未授予许可”，则[启用自动还原](#)。对于旧项目，亦请参阅[迁移到自动程序包还原](#)。另请参阅[程序包还原疑难解答](#)。

在 Visual Studio 中启用和禁用程序包还原

在 Visual Studio 中，主要通过“工具”>“选项”>“NuGet 包管理器”控制程序包还原：



- “允许 NuGet 下载缺失的包”通过更改 `NuGet.Config` 文件中的 `packageRestore` 部分的 `packageRestore/enabled` 设置，控制程序包还原的所有窗体(该文件在 Windows 上位于 `%AppData%\NuGet\`，在 Mac/Linux 上位于 `~/.nuget/NuGet/`)。在 Visual Studio 中，此设置还可启用解决方案上下文菜单中的 Restore NuGet Packages 命令。

```
<configuration>
  <packageRestore>
    <!-- The 'enabled' key is True when the "Allow NuGet to download missing packages" checkbox
    is set.
    Clearing the box sets this to False, disabling command-line, automatic, and MSBuild-
    integrated restore. -->
    <add key="enabled" value="True" />
  </packageRestore>
</configuration>
```

NOTE

要全局重写 `packageRestore/enabled` 设置，请在启动 Visual Studio 或启动生成之前，将 `EnableNuGetPackageRestore` 环境变量设为 TRUE 或 FALSE 值。

- “生成期间在 Visual Studio 中自动检查缺失的包”通过更改 `NuGet.Config` 文件的 `packageRestore` 部分中的 `packageRestore/automatic` 设置控制自动还原。将该选项设置为 True 后，从 Visual Studio 运行生成会自动还原任何缺失的包。此设置不会影响从 MSBuild 命令行运行的生成。

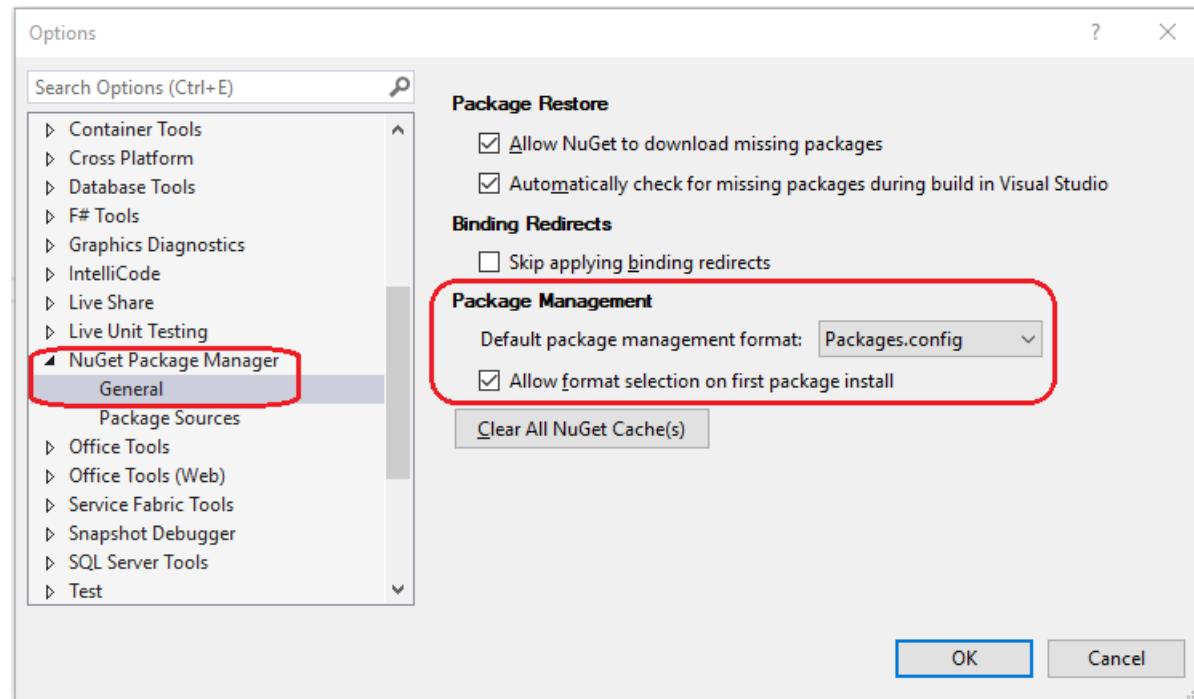
```
...
<configuration>
  <packageRestore>
    <!-- The 'automatic' key is set to True when the "Automatically check for missing packages
    during
    build in Visual Studio" checkbox is set. Clearing the box sets this to False and
    disables
    automatic restore. -->
    <add key="automatic" value="True" />
  </packageRestore>
</configuration>
```

要为计算机上的所有用户启用或禁用程序包还原，开发人员或公司可以将配置设置添加到全局 `nuget.config` 文件。全局 `nuget.config` 在 Windows 中位于 `%ProgramData%\NuGet\Config` 下，有时在特定的 `\{IDE\}\{Version\}\{SKU\}\Visual Studio` 文件夹下，或者在 Mac/Linux 中位于 `~/.local/share` 下。然后，各个用户可以按照项目级别的要求有选择地启用还原。有关 NuGet 如何设置多个配置文件优先级的详细信息，请参阅[常见 NuGet 配置](#)。

IMPORTANT

如果直接在 `nuget.config` 中编辑 `packageRestore` 设置，请重启 Visual Studio，以便“选项”对话框显示当前值。

选择默认包管理格式



NuGet 提供项目可使用包的两种格式：[PackageReference](#) 和 [packages.config](#)。默认格式可从“包管理”标题下的下拉菜单中选择。在项目中安装首个包时还会出现一个提示选项。

NOTE

如果这两种包管理格式均不受项目支持，则会使用与项目兼容的包管理格式，因此所用格式可能并非选项中的默认设置。此外，NuGet 不会在安装首个包时显示选择提示，即使选项窗口中已选中此选项时亦是如此。

如果使用包管理器控制台在项目中安装首个包，则即使选项窗口中已选中该选项，NuGet 也不会提示选择格式。

使用 dotnet CLI 进行还原

使用 `dotnet restore` 命令还原项目文件中列出的包（请参阅 [PackageReference](#)）。使用 .NET Core 2.0 和更高版本，通过 `dotnet build` 和 `dotnet run` 自动完成还原。从 NuGet 4.0 开始，它运行与 `nuget restore` 相同的代码。

与其他 `dotnet` CLI 命令一样，先打开命令行并切换到包含项目文件的目录。

要使用 `dotnet restore` 还原包：

```
dotnet restore
```

IMPORTANT

若要将缺少的程序包引用添加到项目文件，请使用 `dotnet add package`，它也会运行 `restore` 命令。

使用 nuget.exe CLI 进行还原

使用 `restore` 命令可下载并安装“包”文件夹中缺少的所有包。

对于迁移到 `PackageReference` 的项目，请使用 `msbuild -t:restore` 还原程序包。

`restore` 仅将包添加到磁盘，但不会更改项目的依赖项。要还原项目依赖项，请修改 `packages.config`，然后使用 `restore` 命令。

与其他 `nuget.exe` CLI 命令一样，先打开命令行并切换到包含项目文件的目录。

要使用 `restore` 还原包：

```
nuget restore MySolution.sln
```

IMPORTANT

`restore` 命令不会修改项目文件或 `packages.config`。要添加依赖项，请通过 Visual Studio 中的包管理器 UI 或控制台添加包，或修改 `packages.config`，然后运行 `install` 或 `restore`。

使用 MSBuild 进行还原

使用 `msbuild -t:restore` 命令还原项目文件中通过 `PackageReference` 列出的程序包。此命令仅适用于 Visual Studio 2017 及更高版本附带的 NuGet 4.x+ 和 MSBuild 15.1+。`nuget restore` 和 `dotnet restore` 均对适用项目使用此命令。

1. 打开开发人员命令提示符（在“搜索”框中，键入“开发人员命令提示符”）。

用户通常习惯从“开始”菜单中启动“适用于 Visual Studio 的开发人员命令提示符”，因为它将使用 MSBuild 的所有必需路径进行配置。

2. 切换到包含项目文件的文件夹，然后键入以下命令。

```
# Uses the project file in the current folder by default  
msbuild -t:restore
```

3. 键入以下命令以重新生成项目。

```
msbuild
```

确保 MSBuild 输出指示生成已成功完成。

使用 Azure Pipelines 进行还原

在 Azure Pipelines 上创建生成定义时，请在任意生成任务前将 NuGet 还原或 .NET Core 还原任务包括在定义中。默认情况下，某些生成模板包括还原任务。

使用 Azure DevOps Server 进行还原

如果你使用的是 TFS 2013 或更高版本的 Team Build 模板, Azure DevOps Server 和 TFS 2013 及更高版本会在生成期间自动还原包。对于较早的 TFS 版本, 可以包含一个生成步骤来运行命令行还原选项, 或者选择将生成模板迁移到更高版本。有关详细信息, 请参阅[使用 Team Foundation Build 设置程序包还原](#)。

使用还原约束包版本

NuGet 通过任意方法还原包时, 将遵守你在 `packages.config` 或项目文件中指定的任何约束:

- 在 `packages.config` 中, 可在依赖项的 `allowedVersion` 属性中指定版本范围。有关详细信息, 请参阅[约束升级版本](#)。例如:

```
<package id="Newtonsoft.json" version="6.0.4" allowedVersions="[6,7)" />
```

- 在项目文件中, 可以使用 `PackageReference` 直接指定依赖项的范围。例如:

```
<PackageReference Include="Newtonsoft.json" Version="[6, 7)" />
```

在所有情况下, 都使用[包版本控制](#)中介绍的表示法。

强制从包源还原

默认情况下, NuGet 还原操作使用 `global-packages` 和 `http-cache` 文件夹中的包, 如[管理全局包和缓存文件夹](#)中所述。

若要避免使用 `global-packages` 文件夹, 请执行下列操作之一:

- 使用 `nuget locals global-packages -clear` 或 `dotnet nuget locals global-packages --clear` 清除文件夹。
- 在进行还原操作之前, 使用下列方法之一暂时更改 `global-packages` 的位置:
 - 将 `NUGET_PACKAGES` 环境变量设置为其他文件夹。
 - 创建 `NuGet.Config` 文件, 将 `globalPackagesFolder` (如果使用 `PackageReference`) 或 `repositoryPath` (如果使用 `packages.config`) 设置为其他文件夹。有关详细信息, 请参阅[配置设置](#)。
 - 仅限 MSBuild: 指定具有 `RestorePackagesPath` 属性的其他文件夹。

若要避免将缓存用于 HTTP 源, 请执行下列操作之一:

- 将 `-NoCache` 选项与 `nuget restore` 结合使用, 或者将 `--no-cache` 选项与 `dotnet restore` 结合使用。这些选项不会影响通过 Visual Studio 包管理器或控制台执行的还原操作。
- 使用 `nuget locals http-cache -clear` 或 `dotnet nuget locals http-cache --clear` 清除缓存。
- 暂时将 `NUGET_HTTP_CACHE_PATH` 环境变量设置为其他文件夹。

迁移到自动程序包还原 (Visual Studio)

对于 NuGet 2.6 及更早版本, 以前支持集成 MSBuild 的包恢复, 但现在不再支持。(通常通过右键单击 Visual Studio 中的解决方案并选择“启用 NuGet 程序包还原”来启用)。如果项目使用已弃用的集成 MSBuild 的程序包还原, 请迁移到自动程序包还原。

使用集成 MSBuild 的程序包还原的项目通常包含带有三个文件的 `.nuget` 文件夹: `NuGet.config`、`nuget.exe` 和 `NuGet.targets`。`NuGet.targets` 文件的存在与否决定了 NuGet 是否继续使用集成 MSBuild 的方法, 因此在迁移期间必须删除此文件。

要迁移到自动程序包还原, 请执行以下操作:

- 关闭 Visual Studio。
- 删除 `.nuget/nuget.exe` 和 `.nuget/NuGet.targets`。

3. 对于每个项目文件，删除 `<RestorePackages>` 元素并删除对 NuGet.targets 的任何引用。

要测试自动程序包还原，请执行以下操作：

1. 删除解决方案中的“packages”文件夹。
2. 在 Visual Studio 中打开解决方案并开始生成。

自动程序包还原应下载并安装每个依赖包，而不将它们添加到源代码管理中。

疑难解答

请参阅[程序包还原疑难解答](#)。

包还原错误疑难解答

2020/4/8 • [Edit Online](#)

本文着重介绍还原包时遇到的常见错误以及相应的解决步骤。

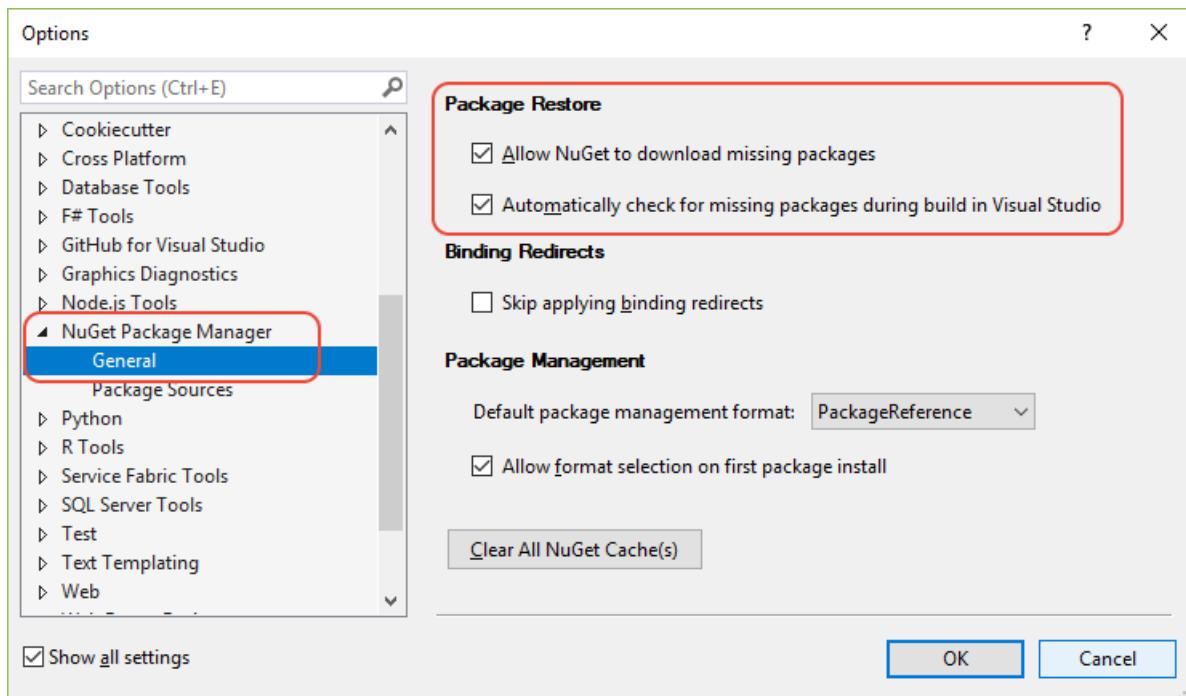
程序包还原试图将所有程序包依赖项安装到与项目文件 (.csproj) 或 packages.config 文件中的程序包引用匹配的相应状态。(在 Visual Studio 中, 这些引用位于解决方案资源管理器的“依赖项 \ NuGet”或“引用”节点中。) 请参阅[还原程序包](#), 以按照要求的步骤还原程序包。若项目文件 (.csproj) 或 packages.config 文件中的程序包引用不正确(它们与使用程序包还原时所需的状态不匹配), 则不要使用程序包还原, 而需要安装或更新程序包。

如果此处的说明对你没有帮助, [请在 GitHub 上提交问题](#), 以便我们可以更仔细地检查你的情况。请勿使用可能出现在该页面上的“此页面有帮助吗?”控件, 因为它无法让我们与你联系以获取详细信息。

适用于 Visual Studio 用户的快速解决方案

如果使用 Visual Studio, 请首先按如下方式启用包还原。否则, 请继续执行后面的部分。

1. 选择“工具”>“NuGet 包管理器”>“包管理器设置”菜单命令。
2. 在“包还原”下设置这两个选项。
3. 选择“确定”。
4. 再次生成项目。



也可以在 `NuGet.config` 文件中更改这些设置; 请参阅[同意](#)部分。如果项目是使用集成 MSBuild 的包恢复的较老项目, 那么可能需要[迁移到自动包恢复](#)。

此项目引用了此计算机上缺少的 NuGet 程序包

完整错误消息:

```
This project references NuGet package(s) that are missing on this computer.  
Use NuGet Package Restore to download them. The missing file is {name}.
```

当你尝试生成包含对一个或多个 NuGet 包的引用的项目，但这些包当前未安装在计算机上或项目中时，发生了此错误。

- 使用 [PackageReference](#) 管理格式时，此错误意味着程序包未安装在 global-packages 文件夹中，如[管理全局程序包和缓存文件夹](#) 中所述。
- 当使用 [packages.config](#) 时，此错误意味着程序包未安装在解决方案根目录中的 `packages` 文件夹中。

当你从源代码管理或其他下载获得项目的源代码时，通常会发生这种情况。包通常从源代码管理或下载中省略，因为它们可以从包源（例如 nuget.org）中还原（请参阅[包和源代码管理](#)）。否则，包含它们会导致存储库膨胀或创建不必要的大型 .zip 文件。

如果项目文件包含包位置的绝对路径，并移动项目，也会发生该错误。

使用下列方法之一还原包：

- 如果已移动项目文件，请直接编辑该文件以更新包引用。
- [Visual Studio（自动还原或手动还原）](#)
- [dotnet CLI](#)
- [nuget.exe CLI](#)
- [MSBuild](#)
- [Azure Pipelines](#)
- [Azure DevOps Server](#)

成功还原后，包应显示在 global-packages 文件夹中。对于使用 PackageReference 的项目，还原应重新创建 `obj/project.assets.json` 文件；对于使用 `packages.config` 的项目，包应显示在项目的 `packages` 文件夹中。该项目现在应能成功生成。如果没有，请在 [GitHub 上提交问题](#)，以便我们跟进。

找不到资产文件 project.assets.json

完整错误消息：

```
Assets file '<path>\project.assets.json' not found. Run a NuGet package restore to generate this file.
```

`project.assets.json` 文件在使用 PackageReference 管理格式时维护项目的依赖项关系图，用于确保在计算机上安装了所有必需的包。由于此文件是通过包还原动态生成的，它通常不添加到源代码管理中。因此，该错误在使用工具生成项目时出现，例如，不自动还原包的 `msbuild`。

在这种情况下，请运行 `msbuild -t:restore`，然后运行 `msbuild`，或使用 `dotnet build`（它会自动还原包）。也可以使用[上一节](#)中的任一包还原方法。

由于尚未获得同意，需要还原的一个或多个 NuGet 包无法进行还原

完整错误消息：

```
One or more NuGet packages need to be restored but couldn't be because consent has not been granted. To give consent, open the Visual Studio Options dialog, click on the NuGet Package Manager node and check 'Allow NuGet to download missing packages during build.' You can also give consent by setting the environment variable 'EnableNuGetPackageRestore' to 'true'. Missing packages: {name}
```

此错误表示你在 NuGet 配置中禁用了包还原。

可按照前文[适用于 Visual Studio 用户的快速解决方案](#)中所述来更改 Visual Studio 中的适用设置。

也可以直接在适用的 `nuget.config` 文件中编辑这些设置（通常，Windows 上为 `%AppData%\NuGet\NuGet.Config`）。

, Mac/Linux 上为 `~/.nuget/NuGet/NuGet.Config`)。确保将 `packageRestore` 下的 `enabled` 和 `automatic` 键设置为 `True`:

```
<!-- Package restore is enabled -->
<configuration>
  <packageRestore>
    <add key="enabled" value="True" />
    <add key="automatic" value="True" />
  </packageRestore>
</configuration>
```

IMPORTANT

如果直接在 `nuget.config` 中编辑 `packageRestore` 设置, 请重启 Visual Studio, 以便“选项”对话框显示当前值。

其他潜在条件

- 由于缺少文件, 你可能会遇到生成错误, 并看到提示使用 NuGet 还原来下载它们的消息。但是, 运行还原时可能会出现:“所有包都已安装, 无可还原项。”在这种情况下, 请删除 `packages` 文件夹(使用 `packages.config` 时)或 `obj/project.assets.json` 文件(使用 `PackageReference` 时)并再次运行还原。如果错误仍然存在, 请使用命令行中的 `nuget locals all -clear` 或 `dotnet locals all --clear` 以清除 `global-packages` 文件夹和缓存文件夹, 如[管理全局包和缓存文件夹](#)中所述。
- 从源代码管理获取项目时, 项目文件夹可能设置为只读。[更改文件夹权限并尝试重新还原包](#)。
- 你可能使用了旧版本的 NuGet。请访问 nuget.org/downloads 了解最新建议版本。对于 Visual Studio 2015, 建议使用 3.6.0。

如果遇到其他问题, 请[在 GitHub 上提交问题](#), 以便我们获得更多详细信息。

如何重新安装和更新包

2020/4/8 • [Edit Online](#)

[何时重新安装包](#) 中描述了大量有关对包的引用可能在 Visual Studio 项目中损坏的情况。在这些情况下，卸载并重新安装同一版本的包会将这些应用还原为正常工作状态。更新包仅意味着安装更新版本，这常常将包还原为正常工作状态。

Visual Studio 中的程序包管理器控制台提供了许多灵活的选项，用于更新和重新安装程序包。

更新和重新安装包按以下方式实现：

包管理器控制台 (使用 <code>Update-Package</code> 中所述)	<code>Update-Package</code> 命令	<code>Update-Package -reinstall</code> 命令
包管理器 UI	在“更新”选项卡上，选择一个或多个包并选择“更新”	在“已安装”选项卡上，选择一个包，记录其名称，然后选择“卸载”。切换到“浏览”选项卡，搜索包名称并选中，然后选择“安装”。
nuget.exe CLI	<code>nuget update</code> 命令	对于所有包，删除包文件夹，然后运行 <code>nuget install</code> 。对于单个包，删除包文件夹并使用 <code>nuget install <id></code> 再安装一个。

NOTE

对于 dotnet CLI，不需要相同的过程。在类似的情况下，可以使用 [dotnet CLI 还原包](#)。

本文内容：

- [何时重新安装包](#)
- [约束升级版本](#)

何时重新安装包

1. **包还原后的损坏引用**：如果已打开项目并还原了 NuGet 包，但仍看见了损坏的引用，请尝试重新安装每个包。
2. **项目因删除文件损坏**：NuGet 不会阻止删除从包添加的项，因此很容易在无意中修改从包安装的内容并损坏项目。要还原项目，请重新安装受影响的包。
3. **包更新损坏了项目**：如果包的更新损坏了项目，则故障通常由可能也已更新的依赖项包引起。要还原依赖项的状态，请重新安装该特定包。
4. **项目重定向或升级**：这在项目已重定向或升级时并且如果包因为更改目标框架需要重新安装时有用。NuGet 在项目重定向后立即显示该情况下的生成错误，后续生成警告会提醒你包可能需要重新安装。对于项目升级，NuGet 显示项目升级日志中的错误。
5. **开发期间重新安装包**：包创作者常常需要重新安装与他们开发来测试行为的包版本相同的包。
`Install-Package` 命令不提供强制重新安装选项，所以换成使用 `Update-Package -reinstall`。

约束升级版本

默认情况下，重新安装或更新包常常会安装包源中提供的最新版本。

但是，在使用 `packages.config` 管理格式的项目中，可以专门约束版本范围。例如，如果知道可能因为包 API 中存在重要更改，应用程序只能使用 1.x 版本的包，而不是 2.0 以及更高版本的包，则需要将升级约束为 1.x 版本。这会阻止可能损坏应用程序的意外更新。

要设置约束，在文本编辑器中打开 `packages.config`，找到有问题的依赖项，然后添加带版本范围的 `allowedVersions` 属性。例如，要将更新约束为版本 1.x，请将 `allowedVersions` 设为 `[1,2)`：

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="ExamplePackage" version="1.1.0" allowedVersions="[1,2)" />

  <!-- ... -->
</packages>
```

在所有情况下，都使用[包版本控制](#)中介绍的表示法。

使用 Update-Package

注意下述的[注意事项](#)，可以使用 Visual Studio 程序包管理器控制台中的 **Update-Package 命令**（“工具”“NuGet 程序包管理器”“程序包管理器控制台”）轻松重新安装任意程序包 >>。

```
Update-Package -Id <package_name> -reinstall
```

使用此命令比删除包然后尝试在 NuGet 库中找到同一版本的同一包更加简单。请注意，`-Id` 开关是可选的。

没有 `-reinstall` 更新的相同命令会将包更新为更新的版本（如果适用）。如果有问题的包未安装在项目中，则命令会生成一个错误；即 `Update-Package` 未直接安装包。

```
Update-Package <package_name>
```

默认情况下，`Update-Package` 会影响解决方案中的所有项目。要将操作限制于特定项目，请使用 `-ProjectName` 开关，使用出现在解决方案资源管理器中的项目的名称：

```
# Reinstall the package in just MyProject
Update-Package <package_name> -ProjectName MyProject -reinstall
```

要更新一个项目中的所有包（或者使用[重新安装](#)），请使用而无需指定任意特定的包 `-reinstall``-ProjectName`：

```
Update-Package -ProjectName MyProject
```

要更新一个解决方案中的所有包，只需使用 `Update-Package`，无需其他参数或开关。请谨慎使用此窗格，因为它可能需要大量时间来执行所有的更新：

```
# Updates all packages in all projects in the solution
Update-Package
```

使用 [PackageReference](#) 更新项目或解决方案中的包始终会更新为最新版本的包（不包括预发布包）。如果需要，使用 `packages.config` 的项目可以限制更新版本，如下方[约束更新版本](#)中所述。

有关命令的完整详细信息，请参阅 [Update-Package](#) 引用。

注意事项

重新安装包可能影响以下内容：

1. 根据项目目标框架重定向重新安装包

- 在简单的情况下，可以仅使用 `Update-Package -reinstall <package_name>` 重新安装包。会卸载根据旧的目标框架安装的包，同时会根据项目的当前目标框架安装相同的包。
- 在某些情况下，可能会有不支持新目标框架的包。
 - 如果包支持可移植类库 (PCL) 并且项目重定向到包不再支持的平台组合，则将在重新安装后会失去对包的引用。
 - 这可能会呈现直接使用的包或者作为依赖项安装的包。直接使用的包可能支持新的目标框架，而其依赖项不支持。
 - 如果重定向应用程序后重新安装包导致生成或运行时错误，则可能需要还原目标框架，或者搜索正确支持新目标框架的替换包。

2. 项目重定向或升级后添加到 packages.config 中的 requireReinstallation 属性

- 如果 NuGet 检测到包受到重定向或升级项目的影响，它会将 `requireReinstallation="true"` 中的 `packages.config` 属性添加到所有受影响的包引用。因此，Visual Studio 中的每个后续生成会引发这些包的生成警告，提醒你重新安装它们。

3. 使用依赖项重新安装包

- `Update-Package -reinstall` 重新安装与初始包相同的版本，但是，如果没有提供特定的版本约束，则安装依赖项的最新版本。这样可以按需仅更新依赖项以解决问题。但是，如果这将依赖项滚动回早期版本，则可以使用 `Update-Package <dependency_name>` 重新安装该依赖项，而不会影响依赖的包。
- `Update-Package -reinstall <packageName> -ignoreDependencies` 重新安装同一版本的初始包，但不重新安装依赖项。在更新包依赖项时使用可能会导致损坏状态

4. 涉及依赖版本时重新安装包

- 如上所述，重新安装包不更改依赖它的其他任何已安装包的版本。然后，重新安装依赖项可能会损坏依赖包。

管理全局包、缓存和临时文件夹

2020/4/8 • [Edit Online](#)

每当安装、更新或还原包时，NuGet 将管理项目结构多个文件夹之外的包和包信息：

“它”	描述(“它”)
global\packages	global-packages 文件夹是 NuGet 安装任何下载包的位置。每个包完全展开到匹配包标识符和版本号的子文件夹。使用 PackageReference 格式的项目始终直接从该文件夹中使用包。使用 packages.config 时，包将安装到 global-packages 文件夹，然后复制到项目的 packages 文件夹。 <ul style="list-style-type: none">Windows: <code>%userprofile%\.nuget\packages</code>Mac/Linux: <code>~/.nuget/packages</code>使用 NUGET_PACKAGES 环境变量、globalPackagesFolder 或 repositoryPath 配置设置(分别在使用 PackageReference 和 packages.config 时)或 RestorePackagesPath MSBuild 属性(仅限 MSBuild)进行替代。环境变量优先于配置设置。
http\cache	Visual Studio 包管理器 (NuGet 3.x+) 和 dotnet 工具存储此缓存中下载包的副本(另存为 .dat 文件)，这些副本被组织到每个包源的子文件夹中。未展开包，且缓存中有 30 分钟的到期时间。 <ul style="list-style-type: none">Windows: <code>%localappdata%\NuGet\v3-cache</code>Mac/Linux: <code>~/.local/share/NuGet/v3-cache</code>使用 NUGET_HTTP_CACHE_PATH 环境变量替代。
temp	NuGet 在各操作期间在其中存储临时文件的文件夹。 <ul style="list-style-type: none">Windows: <code>%temp%\NuGetScratch</code>Mac/Linux: <code>/tmp/NuGetScratch</code>
plugins-cache 4.8 +	NuGet 存储来自操作声明请求的结果的文件夹。 <ul style="list-style-type: none">Windows: <code>%localappdata%\NuGet\plugins-cache</code>Mac/Linux: <code>~/.local/share/NuGet/plugins-cache</code>使用 NUGET_PLUGINS_CACHE_PATH 环境变量替代。

NOTE

NuGet 3.5 和早期版本使用 `%localappdata%\NuGet\Cache` 中的 packages-cache 而不是 http-cache。

通过使用缓存和 global-packages 文件夹，NuGet 通常会避免下载计算机上已存在的包，以提高安装、更新和还原操作的性能。在使用 [PackageReference](#) 时，global-packages 文件夹还会避免在项目文件夹中保存下载的包，其中它们可能会在无意间被添加到源代码管理，并减少 NuGet 对计算机存储的总体影响。

当要求检索包时，NuGet 会首先查看 global-packages 文件夹。如果不存在包的确切版本，NuGet 将检查所有非 HTTP 包源。如果仍未找到包，NuGet 将查找 http-cache 中的包，除非使用 [dotnet.exe](#) 命令指定

--no-cache，或使用 nuget.exe 命令指定 -NoCache。如果包不在缓存中，或未使用缓存，那么 NuGet 将通过 HTTP 检索包。

有关详细信息，请参阅[安装包时会发生什么情况？](#)。

查看文件夹位置

可以使用 [nuget locals 命令](#) 查看位置：

```
# Display locals for all folders: global-packages, http cache, temp and plugins cache
nuget locals all -list
```

典型输出(Windows;“user1”为当前用户名)：

```
http-cache: C:\Users\user1\AppData\Local\NuGet\v3-cache
global-packages: C:\Users\user1\.nuget\packages\
temp: C:\Users\user1\AppData\Local\Temp\NuGetScratch
plugins-cache: C:\Users\user1\AppData\Local\NuGet\plugins-cache
```

(package-cache 在 NuGet 2.x 中使用，并在 NuGet 3.5 及更早版本中显示。)

还可以使用 [dotnet nuget locals 命令](#) 查看文件夹位置：

```
dotnet nuget locals all --list
```

典型输出(Mac/Linux;“user1”为当前用户名)：

```
info : http-cache: /home/user1/.local/share/NuGet/v3-cache
info : global-packages: /home/user1/.nuget/packages/
info : temp: /tmp/NuGetScratch
info : plugins-cache: /home/user1/.local/share/NuGet/plugins-cache
```

若要显示单个文件夹的位置，请使用 `http-cache`、`global-packages`、`temp` 或 `plugins-cache`，而不是 `all`。

清除本地文件夹

如果安装包时遇到问题或想要确保从远程库安装包，请使用 `locals --clear` 选项 (dotnet.exe) 或 `locals -clear` (nuget.exe)，指定要清除的文件夹，或使用 `all` 清除所有文件夹：

```

# Clear the 3.x+ cache (use either command)
dotnet nuget locals http-cache --clear
nuget locals http-cache -clear

# Clear the 2.x cache (NuGet CLI 3.5 and earlier only)
nuget locals packages-cache -clear

# Clear the global packages folder (use either command)
dotnet nuget locals global-packages --clear
nuget locals global-packages -clear

# Clear the temporary cache (use either command)
dotnet nuget locals temp --clear
nuget locals temp -clear

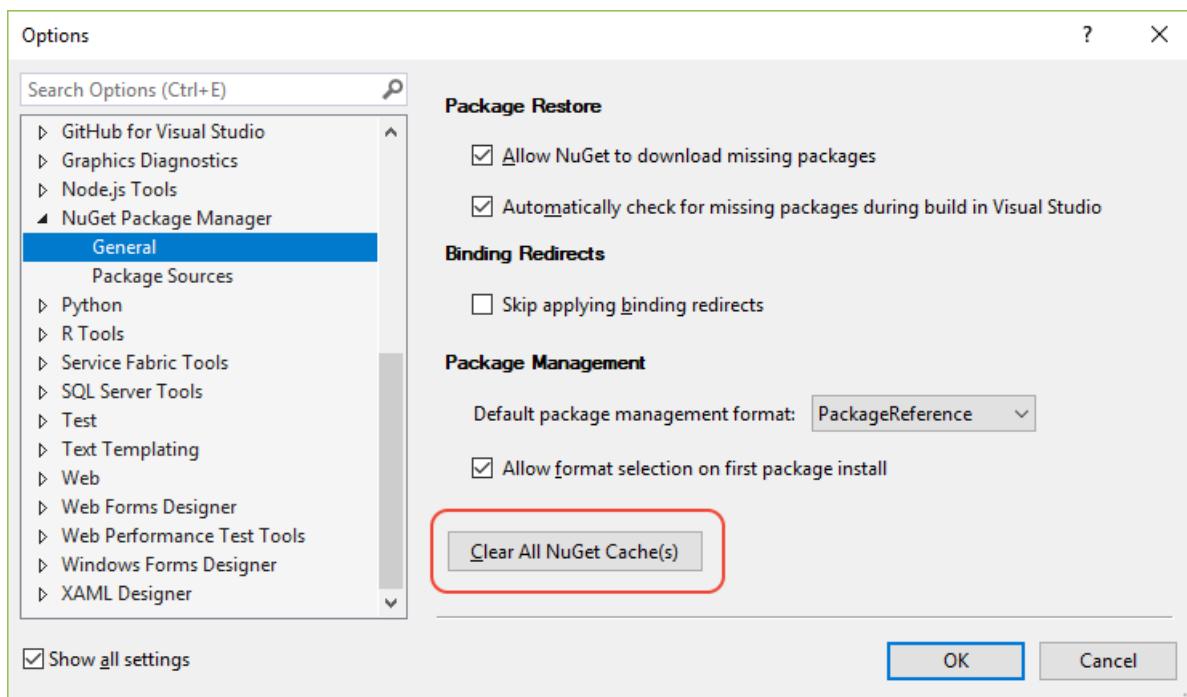
# Clear the plugins cache (use either command)
dotnet nuget locals plugins-cache --clear
nuget locals plugins-cache -clear

# Clear all caches (use either command)
dotnet nuget locals all --clear
nuget locals all -clear

```

目前在 Visual Studio 中打开的项目所使用的任何包都不会从 global-packages 文件夹中清除。

从 Visual Studio 2017 开始，使用“工具”>“NuGet 包管理器”>“包管理器设置”菜单命令，然后选择“清除所有 NuGet 缓存”。管理缓存目前不支持通过包管理器控制台提供。在 Visual Studio 2015 中，则改用 CLI 命令。



错误疑难解答

使用 `nuget locals` 或 `dotnet nuget locals` 时可能出现以下错误：

- 错误：进程无法访问文件，因为另一个进程正在使用该文件或清除本地资源失败：无法删除一个或多个文件

另一个进程正在使用文件夹中的一个或多个文件；例如，Visual Studio 项目处于打开状态，它指的是 global-packages 文件夹中的包。关闭这些进程，然后重试。

- 错误：访问路径被拒绝或目录不为空

你没有删除缓存文件的权限。如果可能，请更改文件夹权限，然后重试。否则，请与系统管理员联系。

- 错误：指定的路径和/或文件名太长。完全限定文件名必须少于 260 个字符，而目录名必须少于 248 个字符。

缩短文件夹名称，然后重试。

管理包信任边界

2020/4/8 • [Edit Online](#)

安装已签名的包不需要任何特定操作;但是,如果内容在签名后被修改,则安装会被阻止并引发[错误 NU3008](#)。

WARNING

使用不受信任的证书签名的包被视为未签名,并且在安装时不会像其他任何未签名的包一样发出任何警告或错误。

配置包签名要求

NOTE

在 Windows 上需要 NuGet 4.9.0+ 和 Visual Studio 版本 15.9 及更高版本

可使用 `signatureValidationMode``require` 命令,通过在 `nuget.config` 文件中将 `nuget config` 设置为,配置 NuGet 客户端包签名的验证方式。

```
nuget.exe config -set signatureValidationMode=require
```

```
<config>
  <add key="signatureValidationMode" value="require" />
</config>
```

此模式将验证所有包通过 `nuget.config` 文件中信任的任何证书进行签名。此文件允许基于证书的指纹指定信任哪些作者和/或存储库。

信任包作者

要基于作者签名信任包,请使用 `trusted-signers` 命令设置 `nuget.config` 中的 `author` 属性。

```
nuget.exe trusted-signers Add -Name MyCompanyCert -CertificateFingerprint
CE40881FF5F0AD3E58965DA20A9F571EF1651A56933748E1BF1C99E537C4E039 -FingerprintAlgorithm SHA256
```

```
<trustedSigners>
  <author name="MyCompanyCert">
    <certificate fingerprint="CE40881FF5F0AD3E58965DA20A9F571EF1651A56933748E1BF1C99E537C4E039"
hashAlgorithm="SHA256" allowUntrustedRoot="false" />
  </author>
</trustedSigners>
```

TIP

使用 `nuget.exe` 验证命令获取证书的指纹值 `SHA256`。

信任存储库中的所有包

要基于存储库签名信任包,请使用 `repository` 元素:

```
<trustedSigners>
  <repository name="nuget.org" serviceIndex="https://api.nuget.org/v3/index.json">
    <certificate fingerprint="0E5F38F57DC1BCC806D8494F4F90FBCEDD988B4676070...."
      hashAlgorithm="SHA256"
      allowUntrustedRoot="false" />
  </repository>
</trustedSigners>
```

信任包所有者

存储库签名包括用于确定提交时包的所有者的其他元数据。可以基于所有者列表限制存储库中的包：

```
<trustedSigners>
  <repository name="nuget.org" serviceIndex="https://api.nuget.org/v3/index.json">
    <certificate fingerprint="0E5F38F57DC1BCC806D8494F4F90FBCEDD988B4676070...."
      hashAlgorithm="SHA256"
      allowUntrustedRoot="false" />
    <owners>microsoft;nuget</owners>
  </repository>
</trustedSigners>
```

如果某个包具有多个所有者，并且受信任列表中包含这些所有者中的任何一个，包安装将成功。

不受信任的根证书

在某些情况下，你可能希望使用证书来启用验证，而证书并未链接到本地计算机的受信任根。可使用 `allowUntrustedRoot` 属性来自定义此行为。

同步存储库证书

包存储库应发布其在[服务索引](#)中使用的证书。存储库最终将更新这些证书，例如证书过期时。出现这种情况时，使用特定策略的客户端将需要更新配置，以包括新添加的证书。可以使用 `nuget.exe` [受信任签名程序同步命令](#)，轻松升级与存储库关联的受信任签名程序。

架构参考

有关客户端策略的完整架构参考，请参阅 [nuget.config 参考](#)

相关文章

- 对 NuGet 包进行签名
- 签名包引用

使用经过身份验证的源中的包

2020/4/8 • [Edit Online](#)

除了 nuget.org [公共源](#) 外, NuGet 客户端能够与文件源和专用 http 源交互。

要向专用 http 源进行身份验证, 可以使用以下两种方式:

- 在 [NuGet.config](#) 中添加凭据
- 根据所用的客户端, 使用一种扩展性模型进行身份验证。

NuGet 客户端的身份验证扩展性

对于各种 NuGet 客户端, 专用源提供程序本身负责进行身份验证。所有 NuGet 客户端都具有支持此方式的扩展性方法。这些方法包括 Visual Studio 扩展或可与 NuGet 通信以检索凭据的插件。

Visual Studio

在 Visual Studio 中, NuGet 公开了一个源提供程序可以实现并向其客户提供的接口。如需了解详情, 请参阅介绍[如何创建 Visual Studio 凭据提供程序](#)的文档。

Visual Studio 的可用 NuGet 凭据提供程序

Visual Studio 中有一个支持 Azure DevOps 的内置凭据提供程序。

可用的插件凭据提供程序包括:

- [适用于 Visual Studio 的 MyGet 凭据提供程序](#)

nuget.exe

当 `nuget.exe` 需要凭据来向源进行身份验证时, 它会采用以下方式查找凭据:

1. 在 `NuGet.config` 文件中查找凭据。
2. 使用 V2 插件凭据提供程序
3. 使用 V1 插件凭据提供程序
4. 然后, NuGet 会提示用户在命令行上提供凭据。

nuget.exe 和 V2 凭据提供程序

在 4.8 版中, NuGet 定义了一种新的身份验证插件机制(以下称为 V2 凭据提供程序)。若要发现和安装这些提供程序, 请参阅 [NuGet 跨平台插件](#)。

nuget.exe 和 V1 凭据提供程序

在 3.3 版本中, NuGet 引入了首版身份验证插件。若要发现和安装这些提供程序, 请参阅 [nuget.exe 凭据提供程序](#)

nuget.exe 的可用凭据提供程序

- [Azure DevOps V2 凭据提供程序](#) 或 [Azure Artifacts 凭据提供程序](#)

对于 Visual Studio 2017 15.9 及更高版本, Azure DevOps 凭据提供程序捆绑在 Visual Studio 中。如果 `nuget.exe` 使用来自该特定 Visual Studio 工具集的 MSBuild, 则系统会自动发现该插件。

dotnet.exe

当 `dotnet.exe` 需要凭据来向源进行身份验证时, 它会采用以下方式查找凭据:

1. 在 `NuGet.config` 文件中查找凭据。
2. 使用 V2 插件凭据提供程序

默认情况下, `dotnet.exe` 不是交互式的, 因此可能需要传递 `--interactive` 标志, 将工具指向用于身份验证的程序

块。

dotnet.exe 和 V2 凭据提供程序

在 SDK 2.2.100 版本中, NuGet 定义了一种适用于所有客户端的身份验证插件机制。若要发现和安装这些提供程序, 请参阅 [NuGet 跨平台插件](#)。

dotnet.exe 的可用凭据提供程序

- [Azure Artifacts 凭据提供程序](#)

MSBuild.exe

当 MSBuild.exe 需要凭据来向源进行身份验证时, 它会采用以下方式查找凭据:

1. 在 NuGet.config 文件中查找凭据
2. 使用 V2 插件凭据提供程序

默认情况下, MSBuild.exe 不是交互式的, 因此你可能需要设置 /p:NuGetInteractive=true 属性, 将工具指向用于身份验证的程序块。

MSBuild.exe 和 V2 凭据提供程序

在 Visual Studio 2019 Update 9 中, NuGet 定义了一种适用于所有客户端的身份验证插件机制。若要发现和安装这些提供程序, 请参阅 [NuGet 跨平台插件](#)。

MSBuild.exe 的可用凭据提供程序

- [Azure Artifacts 凭据提供程序](#)

对于 Visual Studio 2017 Update 9 及更高版本, Azure DevOps 凭据提供程序捆绑在 Visual Studio 中。无需执行其他步骤。

在源代码管理系统中省略 NuGet 包

2020/4/13 • [Edit Online](#)

开发人员通常省略源代码管理存储库中的 NuGet 包，且改为依赖包还原在生成前重新安装项目的依赖项。

以下是依赖包还原的原因：

1. 分布式版本控制系统(如 Git)包括存储库中每个文件每个版本的完整副本。频繁更新的二进制文件会造成大量膨胀并延长克隆存储库所需的时间。
2. 当存储库中包含包时，开发人员负责将引用直接添加到磁盘上的包内容中，而不是通过 NuGet 引用包(可导致项目中的硬编码路径名)。
3. 清理任何未使用包文件夹的解决方案会变得更困难，因为需要确保不删除仍在使用的任何包文件夹。
4. 通过省略包，可以在代码和所依赖的其他包之间维护清晰的所有权边界。许多 NuGet 包已在其自己的源代码管理存储库中进行了维护。

尽管包还原是 NuGet 的默认行为，但省略源代码管理中的包—即项目中的 `packages` 文件夹—仍需进行一些手动操作，详情请见本文。

使用 Git 省略包

使用 [.gitignore 文件](#)忽略 NuGet 包 (`.nupkg`)、`packages` 文件夹、`project.assets.json` 及其他内容。有关参考，请参阅 [Visual Studio 项目的示例 .gitignore](#)：

以下是 `.gitignore` 文件的重要部分：

```
# Ignore NuGet Packages
*.nupkg

# The packages folder can be ignored because of Package Restore
**/[Pp]ackages/*

# except build/, which is used as an MSBuild target.
!**/[Pp]ackages/build/

# Uncomment if necessary however generally it will be regenerated when needed
#!**/[Pp]ackages/repositories.config

# NuGet v3's project.json files produces more ignorable files
*.nuget.props
*.nuget.targets

# Ignore other intermediate files that NuGet might create. project.lock.json is used in conjunction
# with project.json (NuGet v3); project.assets.json is used in conjunction with the PackageReference
# format (NuGet v4 and .NET Core).
project.lock.json
project.assets.json
```

使用 Team Foundation 版本控制省略包

NOTE

向源代码管理添加项目之前，请尽量按照这些说明进行操作。否则，请手动删除存储库中的 `packages` 文件夹并签入该更改，然后才能继续。

要使用 TFVC 为选定的文件禁用源代码管理集成，请执行以下操作：

1. 在解决方案文件夹(.sln 文件所在的位置)中创建名为 .nuget 的文件夹。
 - 提示：对于 Windows，若要在 Windows 资源管理器中创建此文件夹，请使用具有尾随点的名称 .nuget.。
2. 在该文件夹中，创建名为 NuGet.Config 的文件，将其打开进行编辑。
3. 至少添加以下文本，其中 disableSourceControlIntegration 设置指示 Visual Studio 跳过 packages 文件夹中的所有内容：

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <solution>
        <add key="disableSourceControlIntegration" value="true" />
    </solution>
</configuration>
```

4. 如果使用的是 TFS 2010 或更早版本，请掩蔽工作区映射中的 packages 文件夹。
5. 对于 TFS 2012 或更高版本/Visual Studio Team Services，按照[将文件添加到服务器](#)中所述操作来创建 .tfignore 文件。在该文件中，包括以下内容以显式忽略对存储库级别上的 \packages 文件夹和其他几个中间文件的修改。(可以使用具有尾随点的 .tfignore. 名称在 Windows 资源管理器中创建文件，但可能需要首先禁用“隐藏已知文件扩展名”选项。)：

```
# Ignore NuGet Packages
*.nupkg

# Ignore the NuGet packages folder in the root of the repository. If needed, prefix 'packages'
# with additional folder names if it's not in the same folder as .tfignore.
packages

# Omit temporary files
project.lock.json
project.assets.json
*.nuget.props
```

6. 向源代码管理添加 NuGet.Config 和 .tfignore，并签入更改。

常见的 NuGet 配置

2020/4/8 • [Edit Online](#)

NuGet 的行为由一个或多个 `NuGet.Config` (XML) 文件 (可存在于项目范围、用户范围和计算机范围的级别) 中的累积设置驱动。还可以使用全局 `NuGetDefaults.Config` 文件专门配置包源。这些设置应用于 CLI、包管理器控制台和包管理器 UI 中发出的所有命令。

配置文件的位置和使用

位置	NUGET.CONFIG 文件	说明
解决方案	当前文件夹 (又称解决方案文件夹) 或上至驱动器根目录的任何文件夹。	在解决方案文件夹中，设置应用于子文件夹中的所有项目。请注意，如果配置文件位于项目文件夹中，则对该项目没有任何影响。
用户	Windows: <code>%appdata%\NuGet\NuGet.Config</code> Mac/Linux: <code>~/.config/NuGet/NuGet.Config</code> 或 <code>~/.nuget/NuGet/NuGet.Config</code> (因 OS 版本而异)	设置应用于所有操作，但可被任何项目级的设置替代。
计算机	Windows: <code>%ProgramFiles(x86)%\NuGet\Config</code> Mac/Linux: <code>\$XDG_DATA_HOME</code> 。如果 <code>\$XDG_DATA_HOME</code> 的值是 null 或为空，将使用 <code>~/.local/share</code> 或 <code>/usr/local/share</code> (因 OS 版本而异)	设置虽然适用于计算机上的所有操作，但会被任何用户级或项目级设置覆盖。

针对早期版本的 NuGet 的说明：

- NuGet 3.3 及更早版本使用 `.nuget` 文件夹作为解决方案范围的设置。NuGet 3.4+ 中不使用此文件夹。
- 对于 NuGet 2.6 到 3.x 版本，Windows 上的计算机级配置文件位于 `%ProgramData%\NuGet\Config\[\{IDE\}[\{Version\}[\{SKU\}]]]\NuGet.Config`，其中，`{IDE}` 可能为 `VisualStudio`，`{Version}` 为 `Visual Studio` 的版本 (如 14.0)，`{SKU}` 可能为 `Community`、`Pro` 或 `Enterprise`。若要将设置迁移到 NuGet 4.0+，只需将配置文件复制到 `%ProgramFiles(x86)%\NuGet\Config` 即可。在 Linux 上，此位置以前为 `/etc/opt`；在 Mac 上为 `/Library/Application Support`。

更改配置设置

`NuGet.Config` 文件是包含键/值对的简单 XML 文本文件，请参阅 [NuGet 配置设置](#) 主题。

设置通过 NuGet CLI `config` 命令进行管理：

- 默认情况下需更改用户级配置文件。
- 若要更改其他文件中的设置，请使用 `-configFile` 开关。在此情况下，文件可以使用任何文件名。
- 键始终需要区分大小写。
- 更改计算机级设置文件中的设置需要提升权限。

WARNING

尽管可以修改任何文本编辑器中的文件，但如果配置文件中包含格式不正确的 XML（不匹配的标记、无效引号等），NuGet（v3.4.3 及更高版本）将以无提示方式忽略整个配置文件。因此推荐使用 `nuget config` 管理设置。

设置值

Windows:

```
# Set repositoryPath in the user-level config file
nuget config -set repositoryPath=c:\packages

# Set repositoryPath in project-level files
nuget config -set repositoryPath=c:\packages -configfile c:\my.Config
nuget config -set repositoryPath=c:\packages -configfile .\myApp\NuGet.Config

# Set repositoryPath in the computer-level file (requires elevation)
nuget config -set repositoryPath=c:\packages -configfile %ProgramFiles(x86)%\NuGet\Config\NuGet.Config
```

Mac/Linux:

```
# Set repositoryPath in the user-level config file
nuget config -set repositoryPath=/home/packages

# Set repositoryPath in project-level files
nuget config -set repositoryPath=/home/projects/packages -configfile /home/my.Config
nuget config -set repositoryPath=/home/packages -configfile home/myApp/NuGet.Config

# Set repositoryPath in the computer-level file (requires elevation)
nuget config -set repositoryPath=/home/packages -configfile $XDG_DATA_HOME/NuGet.Config
```

NOTE

在 NuGet 3.4 及更高版本中，可以在任何值中使用环境变量，与在 `repositoryPath=%PACKAGEHOME%` (Windows) 和 `repositoryPath=$PACKAGEHOME` (Mac/Linux) 中类似。

删除值

若要删除值，请指定具有空值的键。

```
# Windows
nuget config -set repositoryPath= -configfile c:\my.Config

# Mac/Linux
nuget config -set repositoryPath= -configfile /home/my.Config
```

创建新配置文件

将下方的模板复制到新文件中，然后使用 `nuget config -configFile <filename>` 设置值：

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
</configuration>
```

如何应用设置

可使用多个 `NuGet.Config` 文件在不同位置存储设置，以便设置可应用于单个项目、一组项目或所有项目。这些设置共同应用于从命令行或 Visual Studio 调用的任何 NuGet 操作，并且优先应用“最靠近”项目或当前文件夹的设置。

具体而言，NuGet 将按照以下顺序从不同配置文件加载设置：

1. `NuGetDefaults.Config` 文件，仅包含与包源相关的设置。
2. 计算机级文件。
3. 用户级文件。
4. 用 `-configFile` 指定的文件。
5. 按照从驱动器根目录到当前文件夹（在此文件夹中调用 nuget.exe 或此文件夹包含 Visual Studio 项目）的路径，在其中的每个文件夹中找到的文件。例如，如果在 `c:\A\B\C` 中调用命令，NuGet 将先后查找并加载 `c:.`、`c:\A`、`c:\A\B` 和 `c:\A\B\C` 中的配置文件。

NuGet 在这些文件中找到设置时，设置将按如下方式应用：

1. 对于单项元素，NuGet 将替换以前找到的具有相同键的值。也就是说，“最靠近”当前文件夹或项目的设置将替代之前找到的任何其他设置。例如，如果 `NuGetDefaults.Config` 中的 `defaultPushSource` 设置存在于任何其他配置文件中，则此设置将被替代。
2. 对于集合元素（如 `<packageSources>`），NuGet 会将所有配置文件中的值合并到一个集合中。
3. 当给定节点中存在 `<clear />` 时，NuGet 将忽略之前为该节点定义的配置值。

设置演练

假设两个独立的驱动器上具有以下文件夹结构：

```
disk_drive_1
  User
disk_drive_2
  Project1
    Source
  Project2
    Source
  tmp
```

随后以下位置上将有 4 个具有给定内容的 `NuGet.Config` 文件。（此示例不包括计算机级文件，但其与用户级文件具有相似行为。）

文件 A. 用户级文件（Windows 上为 `%appdata%\NuGet\NuGet.Config`，Mac/Linux 上为 `~/.config/NuGet/NuGet.Config`）：

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <activePackageSource>
    <add key="NuGet official package source" value="https://api.nuget.org/v3/index.json" />
  </activePackageSource>
</configuration>
```

文件 B. `disk_drive_2/NuGet.Config`：

```

<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <config>
        <add key="repositoryPath" value="disk_drive_2/tmp" />
    </config>
    <packageRestore>
        <add key="enabled" value="True" />
    </packageRestore>
</configuration>

```

文件 C. disk_drive_2/Project1/NuGet.Config:

```

<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <config>
        <add key="repositoryPath" value="External/Packages" />
        <add key="defaultPushSource" value="https://MyPrivateRepo/ES/api/v2/package" />
    </config>
    <packageSources>
        <clear /> <!-- ensure only the sources defined below are used -->
        <add key="MyPrivateRepo - ES" value="https://MyPrivateRepo/ES/nuget" />
    </packageSources>
</configuration>

```

文件 D. disk_drive_2/Project2/NuGet.Config:

```

<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <packageSources>
        <!-- Add this repository to the list of available repositories -->
        <add key="MyPrivateRepo - DQ" value="https://MyPrivateRepo/DQ/nuget" />
    </packageSources>
</configuration>

```

接下来, NuGet 将按如下方式加载和应用设置, 具体取决于调用设置的位置:

- 从 disk_drive_1/users 调用:** 仅使用用户级配置文件 (A) 中列出的默认存储库, 因为这是 disk_drive_1 中的唯一文件。
- 从 disk_drive_2/ 或 disk_drive_2/tmp 调用:** 首先加载用户级文件 (A), 然后 NuGet 转到 disk_drive_2 的根目录并查找文件 (B)。NuGet 还将在 /tmp 中查找配置文件, 但不会找到此类文件。因此, 此时将使用 nuget.org 上的默认存储库、启用包还原并在 disk_drive_2/tmp 中展开包。
- 从 disk_drive_2/Project1 或 disk_drive_2/Project1/Source 调用:** 首先加载用户级文件 (A), 然后 NuGet 从 disk_drive_2 的根目录依次加载文件 (B) 和文件 (C)。(C) 中的设置会替代 (B) 和 (A) 中的设置, 因此安装包的 `repositoryPath` 将为 disk_drive_2/Project1/External/Packages, 而非 disk_drive_2/tmp。此外, 由于 (C) 清除了 `<packageSources>`, 因此 nuget.org 将不再可用作源, 并仅留下 `https://MyPrivateRepo/ES/nuget`。
- 从 disk_drive_2/Project2 或 disk_drive_2/Project2/Source 调用:** 首先加载用户级文件 (A), 然后依次加载文件 (B) 和文件 (D)。由于未清除 `packageSources`, 因此 `nuget.org` 和 `https://MyPrivateRepo/DQ/nuget` 都可用作源。按 (B) 中的指定, 包将在 disk_drive_2/tmp 中展开。

NuGet 默认文件

`NuGetDefaults.Config` 文件用于指定通过其安装和更新包的包源, 以及控制使用 `nuget push` 发布包的默认目标。由于管理员可以便捷地向开发人员和生成计算机部署一致的 `NuGetDefaults.Config` 文件(例如, 使用组策略), 因此他们可以确保组织中的每个人都在使用正确的包源, 而不是 nuget.org。

IMPORTANT

`NuGetDefaults.Config` 文件绝不会导致开发人员 NuGet 配置中的包源被删除。也就是说，如果开发人员已使用 NuGet，即意味着已注册 nuget.org 包源，创建 `NuGetDefaults.Config` 文件后将不会删除此包源。

此外，NuGet 中的 `NuGetDefaults.Config` 或任何其他机制都不会阻止访问 nuget.org 等包源。如果组织希望阻止此类访问，则必须通过其他方式（如防火墙）实现。

NuGetDefaults.Config 的位置

下表根据目标操作系统描述 `NuGetDefaults.Config` 文件应存储的位置：

操作系统	NUGETDEFAULTS.CONFIG 位置
Windows	Visual Studio 2017 ■ NuGet 4.x+ : %ProgramFiles(x86)%\NuGet\Config Visual Studio 2015 ■■■ NuGet 3.x ■■■ : %PROGRAMDATA%\NuGet
Mac/Linux	\$XDG_DATA_HOME (通常为 <code>~/.local/share</code> 或 <code>/usr/local/share</code> ，具体视 OS 版本而定)

NuGetDefaults.Config 的设置

- `packageSources`：此集合与常规配置文件中的 `packageSources` 具有相同含义，并可指定默认源。在使用 `packages.config` 管理格式的项目中安装或更新包时，NuGet 会按顺序使用源。对于使用 `PackageReference` 格式的项目，NuGet 会先使用本地源，再使用网络共享上的源，最后使用 HTTP 源，而不管配置文件中的顺序如何。NuGet 会始终忽略还原操作的源顺序。
- `disabledPackageSources`：此集合还与在 `NuGet.Config` 文件中时具有相同含义，集合中将列出每个受影响源的名称，并用 true/false 值指示源是否已禁用。这可使源名称和 URL 保留在 `packageSources` 中，但不会将其默认打开。开发人员随后可在其他 `NuGet.Config` 文件中将源的值设置为 false，以便重新启用源，而无需再次寻找正确的 URL。开发人员还可通过此方法获取组织的内部源 URL 完整列表，同时仅默认启用一个团队的源。
- `defaultPushSource`：指定 `nuget push` 操作的默认目标，同时替代 nuget.org 的内置默认值。管理员可部署此设置以避免误将内部包发布到公共 nuget.org，因为开发人员专门负责使用 `nuget push -Source` 向 nuget.org 发布。

示例 NuGetDefaults.Config 和应用程序

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <!-- defaultPushSource key works like the 'defaultPushSource' key of NuGet.Config files. -->
    <!-- This can be used by administrators to prevent accidental publishing of packages to nuget.org. -->
    <config>
        <add key="defaultPushSource" value="https://contoso.com/packages/" />
    </config>

    <!-- Default Package Sources; works like the 'packageSources' section of NuGet.Config files. -->
    <!-- This collection cannot be deleted or modified but can be disabled/enabled by users. -->
    <packageSources>
        <add key="Contoso Package Source" value="https://contoso.com/packages/" />
        <add key="nuget.org" value="https://api.nuget.org/v3/index.json" />
    </packageSources>

    <!-- Default Package Sources that are disabled by default. -->
    <!-- Works like the 'disabledPackageSources' section of NuGet.Config files. -->
    <!-- Sources cannot be modified or deleted either but can be enabled/disabled by users. -->
    <disabledPackageSources>
        <add key="nuget.org" value="true" />
    </disabledPackageSources>
</configuration>
```

项目文件中的包引用 (PackageReference)

2020/4/8 • [Edit Online](#)

使用 `PackageReference` 节点的包引用可直接在项目文件中管理 NuGet 依赖项(无需单独的 `packages.config` 文件)。使用所谓的 `PackageReference` 不会影响 NuGet 的其他方面;例如, 仍按照 `NuGet.config` 常规 NuGet 配置中的说明应用 文件(包括包源)中的设置。

借助 `PackageReference`, 还可使用 MSBuild 条件按目标框架或其他分组选择包引用。它还允许对依赖项和内容流实行精细控制。(有关更多详细信息, 请参阅[NuGet 打包和还原为 MSBuild 目标](#)。)

项目类型支持

默认情况下, `PackageReference` 用于 .NET Core 项目、.NET Standard 项目, 以及面向 Windows 10 Build 15063(创意者更新)及更高版本的 UWP 项目(C++ UWP 项目除外)。.NET 框架项目支持 `PackageReference`, 但当前默认为 `packages.config`。若要使用 `PackageReference`, 请将 [中的依赖项迁移](#) `packages.config` 到项目文件中, 然后删除 `packages.config`。

面向完整 .NET Framework 的 ASP.NET 应用仅包括对 `PackageReference` 的[有限支持](#)。不支持 C++ 和 JavaScript 项目类型。

添加 PackageReference

在项目文件中使用以下语法添加依赖项:

```
<ItemGroup>
  <!-- ... -->
  <PackageReference Include="Contoso.Utility.UsefulStuff" Version="3.6.0" />
  <!-- ... -->
</ItemGroup>
```

控制依赖项版本

用于指定包版本的约定与使用 `packages.config` 时的约定相同:

```
<ItemGroup>
  <!-- ... -->
  <PackageReference Include="Contoso.Utility.UsefulStuff" Version="3.6.0" />
  <!-- ... -->
</ItemGroup>
```

在上述示例中, `3.6.0` 指 $\geq 3.6.0$ 的任何版本(首选项为最低版本), 详见[包版本控制](#)。

对没有 PackageReferences 的项目使用 PackageReference

高级:如果项目中没有安装包(项目文件中没有 `PackageReference`, 也没有 `packages.config` 文件), 但想要项目还原为 `PackageReference` 样式, 则可以在项目文件中将项目属性 `RestoreProjectStyle` 设置为 `PackageReference`。

```

<PropertyGroup>
    <!-- ... -->
    <RestoreProjectStyle>PackageReference</RestoreProjectStyle>
    <!-- ... -->
</PropertyGroup>

```

如果引用 PackageReference 样式的项目(现有 csproj 或 SDK 样式的项目), 这可能很有用。这可让其他项目以“可传递”的方式引用这些项目引用的包。

PackageReference 和源

在 PackageReference 项目中, 在还原时解析可传递依赖项版本。因此, 在 PackageReference 项目中, 所有源必须可用于所有还原。

可变版本

支持可变版本 `PackageReference` :

```

<ItemGroup>
    <!-- ... -->
    <PackageReference Include="Contoso.Utility.UsefulStuff" Version="3.6.*" />
    <PackageReference Include="Contoso.Utility.UsefulStuff" Version="3.6.0-beta*" />
    <!-- ... -->
</ItemGroup>

```

控制依赖项资产

你可能只是将依赖项用作开发工具, 而不希望将它向会使用包的项目公开。在此情况下, 可使用 `PrivateAssets` 元数据控制此行为。

```

<ItemGroup>
    <!-- ... -->

    <PackageReference Include="Contoso.Utility.UsefulStuff" Version="3.6.0">
        <PrivateAssets>all</PrivateAssets>
    </PackageReference>

    <!-- ... -->
</ItemGroup>

```

以下元数据标记控制依赖项资产:

<code>IncludeAssets</code>	将使用这些资产	<code>all</code>
<code>ExcludeAssets</code>	不会使用这些资产	<code>none</code>
<code>PrivateAssets</code>	将使用这些资产, 但它们不会流入上级项目	<code>contentfiles;analyzers;build</code>

以下是这些标记的允许值, 其中用分号分隔多个值(但 `all` 和 `none` 必须单独显示):

编译	<code>lib</code> 文件夹的内容，控制项目能否对文件夹中的程序集进行编译
运行库	<code>lib</code> 和 <code>runtimes</code> 文件夹的内容，控制是否会复制这些程序集，以生成输出目录
contentFiles	<code>contentfiles</code> 文件夹中的内容
build	<code>.props</code> 文件夹中的 <code>.targets</code> 和 <code>build</code>
buildMultitargeting	(4.0) 文件夹中跨框架目标的和 <code>.props``.targets``buildMultitargeting</code>
buildTransitive	(5.0+) 以可传递的方式流入任意使用项目的资产的文件夹中的和 <code>.props``.targets``buildTransitive</code> 。请参阅 功能页 。
analyzers	.NET 分析器
本机	<code>native</code> 文件夹中的内容
none	不使用以上任何内容。
all	以上都是(除 <code>none</code> 之外)

在以下示例中，项目将使用除包中的内容文件之外的所有项，并且除内容文件和分析器之外的所有项均会流入上级项目。

```
<ItemGroup>
  <!-- ... -->

  <PackageReference Include="Contoso.Utility.UsefulStuff" Version="3.6.0">
    <IncludeAssets>all</IncludeAssets>
    <ExcludeAssets>contentFiles</ExcludeAssets>
    <PrivateAssets>contentFiles;analyzers</PrivateAssets>
  </PackageReference>

  <!-- ... -->
</ItemGroup>
```

请注意，因为 `build` 未包括 `PrivateAssets`，所以目标和属性将流入上级项目。例如，假设在生成名为 `AppLogger` 的 NuGet 包的项目中使用上述引用。`AppLogger` 可以使用 `Contoso.Utility.UsefulStuff` 中的目标和属性，使用 `AppLogger` 的项目也可以。

NOTE

在 `developmentDependency` 文件中将 `true` 设置为 `.nuspec` 时，会将包标记为仅开发依赖项，从而防止包作为依赖项包含到其他包中。利用 `PackageReference` (NuGet 4.8+)，此标志还意味着将从编译中排除编译时资产。有关详细信息，请参阅[PackageReference 的 DevelopmentDependency 支持](#)。

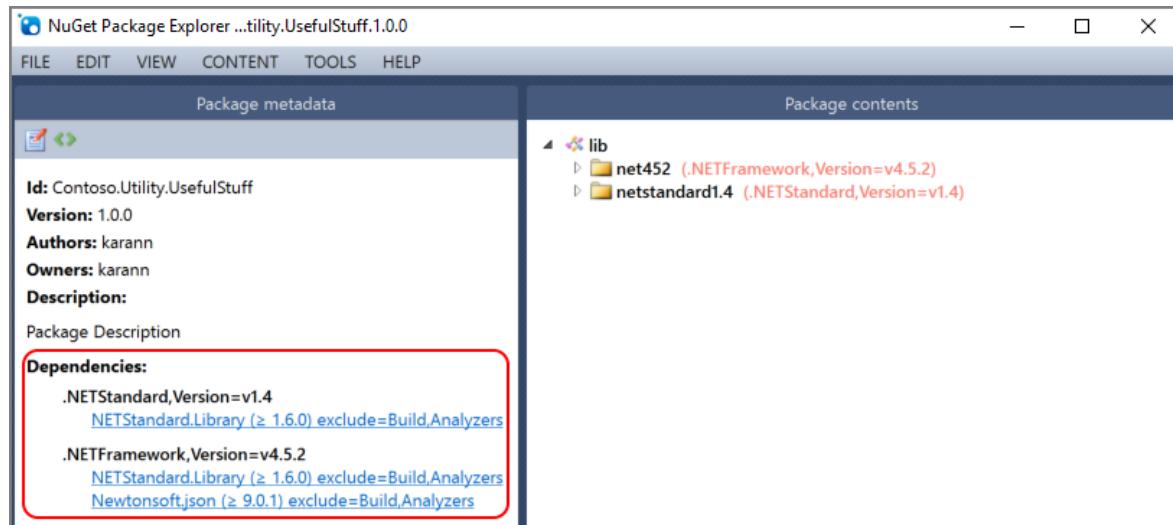
添加 PackageReference 条件

可使用条件控制是否包括包，在哪种条件可使用任何 MSBuild 变量或者目标或属性文件中定义的变量。但目前仅支持 `TargetFramework` 变量。

例如，假设目标为 `netstandard1.4` 和 `net452`，但具有的依赖项仅适用于 `net452`。在此情况下，你不希望使用包的 `netstandard1.4` 项目添加该不必要的依赖项。要防止此情况，请在 `PackageReference` 上指定条件，如下所示：

```
<ItemGroup>
  <!-- ... -->
  <PackageReference Include="Newtonsoft.Json" Version="9.0.1" Condition=" '$(TargetFramework)' == 'net452' " />
  <!-- ... -->
</ItemGroup>
```

使用此项目生成的包会显示，仅对于 `net452` 目标，`Newtonsoft.json` 包含为依赖项：



还可以在 `ItemGroup` 级别应用条件，并将其应用到所有子级 `PackageReference` 元素：

```
<ItemGroup Condition = "'$(TargetFramework)' == 'net452'">
  <!-- ... -->
  <PackageReference Include="Newtonsoft.Json" Version="9.0.1" />
  <PackageReference Include="Contoso.Utility.UsefulStuff" Version="3.6.0" />
  <!-- ... -->
</ItemGroup>
```

GeneratePathProperty

NuGet 5.0 或更高版本以及 Visual Studio 2019 16.0 或更高版本随附此功能。

有时需要从 MSBuild 目标引用包中的文件。在基于 `packages.config` 的项目中，包安装在项目文件所在的文件夹中。但是，在 `PackageReference` 中，包是在 `global-packages` 文件夹中使用的，此文件可能会因计算机而异。

为了消除这种差异，NuGet 引入了一个指向包使用位置的属性。

示例：

```
<ItemGroup>
  <PackageReference Include="Some.Package" Version="1.0.0" GeneratePathProperty="true" />
</ItemGroup>

<Target Name="TakeAction" AfterTargets="Build">
  <Exec Command="$(PkgSome_Package)\something.exe" />
</Target>
```

此外，NuGet 会对包含 tools 文件夹的包自动生成属性。

```
<ItemGroup>
  <PackageReference Include="Package.With.Tools" Version="1.0.0" />
</ItemGroup>

<Target Name="TakeAction" AfterTargets="Build">
  <Exec Command="$(PkgPackage_With_Tools)\tools\tool.exe" />
</Target>
```

MSBuild 属性和包标识不具有相同的限制，因此包标识需要改为一个 MSBuild 易记名称（以 `Pkg` 为前缀）。若要验证生成的属性的确切名称，请查看生成的 `nuget.g.props` 文件。

NuGet 警告和错误

NuGet 4.3 或更高版本以及 Visual Studio 2017 15.3 或更高版本随附此功能。

对于许多打包和还原方案，所有 NuGet 警告和错误都经过编码，且以 `NU****` 开头。所有 NuGet 警告和错误都列在[参考](#)文档中。

NuGet 遵循以下警告属性：

- `TreatWarningsAsErrors` 将所有警告视为错误
- `WarningsAsErrors` 将特定警告报告为错误
- `NoWarn` 隐藏项目范围或包范围内的特定警告。

示例：

```
<PropertyGroup>
  <TreatWarningsAsErrors>true</TreatWarningsAsErrors>
</PropertyGroup>
...
<PropertyGroup>
  <WarningsAsErrors>$($WarningsAsErrors);NU1603;NU1605</WarningsAsErrors>
</PropertyGroup>
...
<PropertyGroup>
  <NoWarn>$($NoWarn);NU5124</NoWarn>
</PropertyGroup>
...
<ItemGroup>
  <PackageReference Include="Contoso.Package" Version="1.0.0" NoWarn="NU1605" />
</ItemGroup>
```

阻止 NuGet 警告

虽然推荐在打包和还原操作期间解决所有 NuGet 警告，在某些情况下可以禁止显示警告。若要禁止显示项目范围内的警告，请考虑执行以下操作：

```
<PropertyGroup>
  <PackageVersion>5.0.0</PackageVersion>
  <NoWarn>$({NoWarn});NU5104</NoWarn>
</PropertyGroup>
<ItemGroup>
  <PackageReference Include="Contoso.Package" Version="1.0.0-beta.1"/>
</ItemGroup>
```

有时，警告仅适用于图中的特定包。通过对 PackageReference 项目添加 `NoWarn`，我们可以有更多地选择来禁止显示该警告。

```
<PropertyGroup>
  <PackageVersion>5.0.0</PackageVersion>
</PropertyGroup>
<ItemGroup>
  <PackageReference Include="Contoso.Package" Version="1.0.0-beta.1" NoWarn="NU1603" />
</ItemGroup>
```

在 Visual Studio 中禁止显示 NuGet 包警告

在 Visual Studio 中，还可以通过 IDE [禁止显示警告](#)。

锁定依赖项

NuGet 4.9 或更高版本以及 *Visual Studio 2017 15.9 或更高版本* 随附此功能。

对 NuGet 还原的输入是项目文件中的一组包引用（顶级或直接依赖项），而输出则是所有包依赖项的完整闭包，其中包括可传递依赖项。如果输入 PackageReference 列表尚未更改，则 NuGet 尝试始终生成相同的完整闭包。但是，在某些情况下，它无法执行此操作。例如：

- 在使用 `<PackageReference Include="My.Sample.Lib" Version="4.*"/>` 等浮动版本时。尽管在此处这样做的目的是浮动到每个包还原的最新版本，但是在某些情况下，用户需要在一个显式动作后，将图形锁定到某个最新版本并浮动到更高版本（如果有可用的更高版本）。
- 匹配 PackageReference 版本要求的较新版本已发布。例如
 - 第 1 天：如果指定了 `<PackageReference Include="My.Sample.Lib" Version="4.0.0"/>`，但在 NuGet 存储库上可用的版本为 4.1.0、4.2.0 和 4.3.0。在这种情况下，NuGet 将解析为 4.1.0（最接近的最低版本）
 - 第 2 天：版本 4.0.0 发布。NuGet 现在将查找完全匹配并开始解析到 4.0.0
- 给定包版本将从存储库中删除。尽管 nuget.org 不允许包删除，但并非所有包存储库都具有此约束。这导致 NuGet 在无法解析到已删除的版本时查找最佳匹配项。

启用锁定文件

为了保留包依赖项的完整闭包，可以选择锁定文件功能，方法是通过设置项目的 MSBuild 属性

`RestorePackagesWithLockFile`：

```
<PropertyGroup>
  <!-- ... -->
  <RestorePackagesWithLockFile>true</RestorePackagesWithLockFile>
  <!-- ... -->
</PropertyGroup>
```

如果设置此属性，NuGet 还原将生成一个锁定文件 - 项目根目录中列出所有包依赖项的 `packages.lock.json` 文件。

NOTE

一旦项目在其根目录中具有 `packages.lock.json` 文件，则即使在未设置属性 `RestorePackagesWithLockFile` 时，锁定文件仍将始终与还原配合使用。因此，选择加入此功能的另一个方法是在项目的根目录中创建一个虚拟空白 `packages.lock.json` 文件。

锁定文件的 `restore` 行为

如果项目存在锁定文件，则 NuGet 使用此锁定文件来运行 `restore`。NuGet 将执行一次快速检查，以确定在包依赖项中是否存在项目文件（或相关项目的文件）中所提及的任何更改，如果没有任何更改，则它将仅还原锁定文件中提及的包。不会对包依赖项进行重新评估。

如果 NuGet 检测到在定义的依赖项中存在一个或多个项目文件中提及的更改，它会重新评估包关系图，并将锁定文件更新为反映项目的新闭包。

对于 CI/CD 和其他方案（不想匆忙更改包依赖项），为此可以将 `lockedmode` 设置为 `true`：

对于 `dotnet.exe`，请运行：

```
> dotnet.exe restore --locked-mode
```

对于 `msbuild.exe`，请运行：

```
> msbuild.exe -t:restore -p:RestoreLockedMode=true
```

此外，还可以在项目文件中设置此条件 MSBuild 属性：

```
<PropertyGroup>
  <!-- ... -->
  <RestoreLockedMode>true</RestoreLockedMode>
  <!-- ... -->
</PropertyGroup>
```

如果锁定模式为 `true`，则还原将还原锁定文件中列出的完全匹配的包，或者，如果在锁定文件创建后为项目更新了定义的包依赖项，则还原将失败。

使锁定文件作为源存储库的一部分

若要生成应用程序，且可执行文件和相关项目位于依赖关系链的开头，请将锁定文件签入源代码存储库，以便 NuGet 能够在还原期间使用它。

但是，如果你的项目是不交付的库项目或其他项目依赖的常用代码项目，则不应将锁定文件作为源代码的一部分签入。保留锁定文件没有任何坏处，但在依赖于此常用代码项目的项目还原/生成期间，锁定文件中列出的常用代码项目的锁定的包依赖项可能无法使用。

例如

```
ProjectA
  |-----> PackageX 2.0.0
  |-----> ProjectB
    |-----> PackageX 1.0.0
```

如果 `ProjectA` 在 `PackageX` 版本 `2.0.0` 上具有依赖项并引用依赖于 `ProjectB` 版本 `PackageX` 的 `1.0.0`，则 `ProjectB` 的锁定文件将列出 `PackageX` 版本 `1.0.0` 的依赖项。但是，当生成 `ProjectA` 时，其锁定文件将包含 `PackageX` 锁定文件中列出的版本 `2.0.0`（而不是）上的依赖项 `1.0.0`ProjectB`。因此，常用代

码项目的锁定文件对依赖于它的项目进行解析的包几乎没有控制。

锁定文件可扩展性

可以使用以下所述的锁定文件控制各种还原行为：

NUGET.EXE	DOTNET	MSBUILD	
<code>-UseLockFile</code>	<code>--use-lock-file</code>	RestorePackagesWithLock File	选择使用锁定文件。
<code>-LockedMode</code>	<code>--locked-mode</code>	RestoreLockedMode	为还原启用锁定模式。这对于要获取可重复生成的 CI/CD 方案非常有用。
<code>-ForceEvaluate</code>	<code>--force-evaluate</code>	RestoreForceEvaluate	对于在项目中定义了浮动版本的包，此选项也非常有用。默认情况下，NuGet 还原不会在每次还原时自动更新包版本，除非使用此选项运行还原。
<code>-LockFilePath</code>	<code>--lock-file-path</code>	NuGetLockFilePath	为项目定义自定义锁定文件位置。默认情况下，NuGet 支持根目录中的 <code>packages.lock.json</code> 。如果在同一目录中具有多个项目，则 NuGet 支持特定于项目的锁定文件 <code>packages.<project_name>.lock.json</code>

从 packages.config 迁移到 PackageReference

2020/4/8 • [Edit Online](#)

Visual Studio 2017 版本 15.7 及更高版本支持将项目从 [packages.config](#) 管理格式迁移到 [PackageReference](#) 格式。

使用 PackageReference 的好处

- 在一个位置管理所有项目依赖项**: 与项目到项目的引用和程序集引用一样, NuGet 包引用(使用 `PackageReference` 节点)直接在项目文件中进行管理, 而不是使用单独的 `packages.config` 文件进行管理。
- 顶级依赖项的有序视图**: 与 `packages.config` 不同, `PackageReference` 仅列出那些直接安装在项目中的 NuGet 包。因此, NuGet 包管理器 UI 和项目文件不会与低级依赖项混在一起。
- 性能改进**: 使用 `PackageReference` 时, 包保留在 `global-packages` 文件夹中(而不是解决方案中的 `packages` 文件夹中), 如[管理全局包和缓存文件夹](#)中所述。因此, `PackageReference` 的执行速度更快, 但占用的磁盘空间更少。
- 更好地控制依赖项和内容流**: 使用 MSBuild 的现有功能可以有条件地引用 NuGet 包, 并选择每个目标框架、配置、平台或其他透视的包引用。
- PackageReference 正处于积极开发阶段**: 请参阅 [GitHub 上的 PackageReference 问题](#)。`packages.config` 不再处于积极开发阶段。

限制

- NuGet `PackageReference` 在 Visual Studio 2015 及更早版本中不可用。只能在 Visual Studio 2017 及更高版本中打开已迁移的项目。
- 目前, C++ 和 ASP.NET 项目无法进行迁移。
- 某些包可能与 `PackageReference` 不完全兼容。有关详细信息, 请参阅[包兼容性问题](#)。

此外, `PackageReferences` 的工作原理与 `packages.config` 相比存在一些差异。例如, `PackageReference` 不支持[约束升级版本](#), 但添加了对[可变版本](#)的支持。

已知问题

- `Migrate packages.config to PackageReference...` 选项在右键单击上下文菜单中不可用

问题

首次打开项目时, 在执行 NuGet 操作之前, NuGet 可能不会进行初始化。这将导致迁移选项不会显示在 `packages.config` 或 `References` 的右键单击上下文菜单中。

解决方法

执行以下 NuGet 操作之一:

- 打开包管理器 UI - 右键单击 `References` 并选择 `Manage NuGet Packages...`
- 打开“包管理器控制台” - 从 `Tools > NuGet Package Manager` 中选择 `Package Manager Console`
- 运行 NuGet 还原 - 在“解决方案资源管理器”中, 右键单击该解决方案节点, 然后选择 `Restore NuGet Packages`
- 生成还会触发 NuGet 还原的项目

现在应能够看到迁移选项。请注意, 此选项不受支持且不会对 ASP.NET 和 C++ 项目类型显示。

迁移步骤

NOTE

在开始迁移之前, Visual Studio 会创建项目的备份, 以便在必要时[回滚到 packages.config](#)。

1. 使用 `packages.config` 打开包含项目的解决方案。
2. 在“解决方案资源管理器”中, 右键单击“引用”节点或 `packages.config` 文件, 然后选择“将 `packages.config` 迁移到 PackageReference...”。
3. 迁移程序将分析项目的 NuGet 包引用并尝试将它们分类为“顶级依赖项”(直接安装的 NuGet 包)和“可传递的依赖项”(作为顶级包的依赖项安装的包)。

NOTE

PackageReference 支持可传递包还原并动态解析依赖项, 这意味着无需显式安装可传递的依赖项。

4. (可选)可以选择将分类为可传递的依赖项的 NuGet 包视为顶级依赖项, 方法是为包选择“顶级”选项。对于包含不以可传递的方式流式传输的资产(`build`、`buildCrossTargeting`、`contentFiles` 或 `analyzers` 文件夹中的资产)以及标记为开发依赖项(`developmentDependency = "true"`)的资产的包, 将自动设置此选项。
5. 查看任何[包兼容性问题](#)。
6. 选择“确定”, 开始迁移。
7. 迁移结束时, Visual Studio 会提供一个报表, 其中包含指向备份的路径、已安装包的列表(顶级依赖项)、作为可传递的依赖项引用的包的列表以及在开始迁移时确定的兼容性问题的列表。该报表将保存到备份文件夹。
8. 验证解决方案是否生成并运行。如果遇到问题, 请[在 GitHub 上提出问题](#)。

如何回滚到 packages.config

1. 关闭已迁移的项目。
2. 将项目文件和 `packages.config` 从备份(通常为 `<solution_root>\MigrationBackup\<unique_guid>\<project_name>\`)复制到项目文件夹。如果项目根目录中存在 `obj` 文件夹, 则删除该文件夹。
3. 打开项目。
4. 使用“工具”>“NuGet 包管理器”>“包管理器控制台”菜单命令打开包管理器控制台。
5. 在控制台中运行以下命令:

```
update-package -reinstall
```

迁移后创建包

迁移完成后, 建议你添加对 `nuget.build.tasks.pack` nuget 包的引用, 然后使用 `msbuild -t:pack` 来创建包。尽管在某些情况下可以使用 `dotnet.exe pack` 而不是 `msbuild -t:pack`, 但不建议使用。

包兼容性问题

PackageReference 不支持 `packages.config` 支持的某些方面。迁移程序将分析并检测此类问题。具有以下一个

或多个问题的任何包在迁移后可能无法按预期方式工作。

如果在迁移后安装了包，则将忽略“install.ps1”脚本

■	借助 PackageReference，安装或卸载包时不会执行 install.ps1 和 uninstall.ps1 PowerShell 脚本。
■	依赖于这些脚本来配置目标项目中的某些行为的包可能无法按预期方式工作。

如果在迁移后安装了包，则“内容”资产不可用

■	包的 <code>content</code> 文件夹中的资产不受 PackageReference 支持，并且将被忽略。PackageReference 添加了对 <code>contentFiles</code> 的支持，以便获得更好的可传递支持和共享内容。
■	<code>content</code> 中的资产不会复制到项目中，依赖于这些资产的项目代码需要重构。

如果在升级后安装了包，则不会应用 XDT 转换

■	PackageReference 不支持 XDT 转换，因此在安装或卸载包时将忽略 <code>.xdt</code> 文件。
■	XDT 转换不会应用于任何项目 XML 文件（最常见的是 <code>web.config.install.xdt</code> 和 <code>web.config.uninstall.xdt</code> ），这意味着在安装或卸载包时不会更新项目的 <code>web.config</code> 文件。

如果在迁移后安装了包，则将忽略 lib 根目录中的程序集

■	借助 PackageReference，将忽略无目标框架特定子文件夹的 <code>lib</code> 文件夹的根目录中的程序集。NuGet 会查找与对应于项目的目标框架的目标框架名字对象 (TFM) 相匹配的子文件夹，并将匹配的程序集安装到项目中。
■	没有与对应于项目的目标框架的目标框架名字对象 (TFM) 相匹配的子文件夹的包在迁移期间转换或安装失败后可能无法按预期方式工作

如果发现错误，请将其报告！

如果在迁移过程中遇到问题，请在 [NuGet GitHub 存储库上提出问题](#)。

packages.config 引用

2020/3/3 • [Edit Online](#)

某些项目类型中使用 `packages.config` 文件维护项目引用的包的列表。这允许 NuGet 在项目要传输到其他计算机（如生成服务器）上时轻松还原项目的依赖项，而无需所有这些包。

如果使用，`packages.config` 通常位于项目根目录中。它在第一个 NuGet 操作运行时自动创建，但也可以在运行任何命令（如 `nuget restore`）之前手动创建。

使用[PackageReference](#)的项目不使用 `packages.config`。

架构

架构十分简单：以下标准 XML 标头是一个 `<packages>` 节点，包含一个或多个 `<package>` 元素，每个元素用于一个引用。每个 `<package>` 元素可具有以下属性：

ATTRIBUTE	DEFINITION	DESCRIPTION
<code>id</code>	是	包的标识符，如 <code>Newtonsoft.json</code> 或 <code>Microsoft.AspNet.Mvc</code> 。
版本	是	要安装的包的确切版本，如 <code>3.1.1</code> 或 <code>4.2.5.11-beta</code> 。版本字符串必须至少具有三个数字，可以选择性添加第四个数字作为预发布后缀。不允许使用范围。
<code>targetFramework</code>	否	安装包时应用的 目标框架名字对象 (TFM) 。安装包时，此内容最初设置为项目目标。因此，不同的 <code><package></code> 元素可具有不同的 TFM。例如，如果创建面向 .NET 4.5.2 的项目，此时安装的包将使用 <code>net452</code> 的 TFM。如果稍后将项目重定向到 .NET 4.6 并添加更多包，这些包将使用 <code>net46</code> 的 TFM。项目目标和 <code>targetFramework</code> 属性之间的不匹配会生成警告，在此情况下，可以重新安装受影响的包。
<code>allowedVersions</code>	否	在包更新期间允许对此包应用的一系列版本（请参阅 约束升级版本 ）。这不影响安装或还原操作期间安装的包。有关语法的信息，请参阅 包版本控制 。PackageManager UI 还禁用允许范围之外的所有版本。
<code>developmentDependency</code>	否	如果使用项目本身创建 NuGet 包，针对依赖项将其设置为 <code>true</code> ，可防止在创建使用包时添加该包。默认为 <code>false</code> 。

示例

以下 `packages.config` 指的是两个依赖项：

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="jQuery" version="3.1.1" targetFramework="net46" />
  <package id="NLog" version="4.3.10" targetFramework="net46" />
</packages>
```

以下 `packages.config` 指的是九个包，但由于 `Microsoft.Net.Compilers` 属性，生成使用包时不会包括 `developmentDependency`。对 `Newtonsoft.Json` 的引用还将更新限制为仅 8.x 和 9.x 版本。

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="Microsoft.CodeDom.Providers.DotNetCompilerPlatform" version="1.0.0" targetFramework="net46"
/>
  <package id="Microsoft.Net.Compilers" version="1.0.0" targetFramework="net46"
developmentDependency="true" />
  <package id="Microsoft.Web.Infrastructure" version="1.0.0.0" targetFramework="net46" />
  <package id="Microsoft.Web.Xdt" version="2.1.1" targetFramework="net46" />
  <package id="Newtonsoft.Json" version="8.0.3" allowedVersions="[8,10)" targetFramework="net46" />
  <package id="NuGet.Core" version="2.11.1" targetFramework="net46" />
  <package id="NuGet.Server" version="2.11.2" targetFramework="net46" />
  <package id="RouteMagic" version="1.3" targetFramework="net46" />
  <package id="WebActivatorEx" version="2.1.0" targetFramework="net46" />
</packages>
```

包创建工作流

2020/4/8 • [Edit Online](#)

通过公共 nuget.org 库或组织内部的私有库，创建你要进行打包并与他人共享的、以编译代码（通常为 .NET 程序集）开头的包。此包还可以包含其他文件（如安装包时显示的自述文件）以及特定项目文件的转换文件。

包还可仅用于拉入任意数量的其他依赖项，但不包含其自己的任何代码。利用这种包可以轻松传递由多个独立包组成的 SDK。在其他情况下，包可能仅包含用于协助调试的符号（.pdb）文件。

NOTE

如果要创建供其他开发人员使用的包，则务必了解他们将依赖于你的工作成果。因此，创建并发布包也表示你将修复 bug 和提供其他更新，或者至少应以开源形式提供包，以便他人可帮助维护包。

无论哪种情况，都要从决定其标识符、版本号、许可证、版权信息和任何其他必要内容开始创建包。完成后，可以使用“pack”命令将所有内容放到 .nupkg 文件中。可以将此文件发布到 NuGet 源，如 nuget.org。

TIP

具有 .nupkg 扩展名的 NuGet 包只是一个 zip 文件。若要轻松查看任何包的内容，只需将扩展名更改为 .zip 并按常规方法展开内容。尝试将包上传到主机前，请务必将扩展名改回 .nupkg。

若想了解并掌握创建流程，请首先参阅[创建包](#)，其中针对适用于所有包的核心流程提供了完整说明。

接下来，你可以考虑对包应用其他多种选项：

- [支持多目标框架](#)，说明如何创建具有面向不同 .NET Framework 的多个变体的包。
- [创建本地化包](#)，说明如何构建具有多个语言资源的包以及如何使用独立的本地化附属包。
- [预发行包](#)，演示如何向感兴趣的客户发布 alpha、beta 和 rc 版本的包。
- [源和配置文件转换](#)，说明如何在已添加到项目的文件中执行两种单向令牌替换，以及如何修改 web.config 和 app.config（卸载包时还将舍弃这两者的设置）。
- [符号包](#)，说明如何为库提供符号以允许使用者在调试期间单步执行代码。
- [包版本控制](#)，说明如何识别依赖项（通过你的包所使用的其他包）所需的确切版本。
- [本机包](#)，说明面向 C++ 使用者创建包的流程。
- [签名包](#)，说明向包添加数字签名的流程。

当准备好将包发布到 nuget.org 时，请按照[发布包](#)中的简单流程执行操作。

如果希望使用私有源而非 nuget.org，请参阅[托管包概述](#)

使用 dotnet CLI 创建 NuGet 包

2020/4/8 • [Edit Online](#)

无论包是什么用途或者它包含什么代码，均使用其中一个 CLI 工具（`nuget.exe` 或 `dotnet.exe`）将该功能打包进可以与任意数量的其他开发人员共享且可以由其使用的组件中。本文介绍如何使用 dotnet CLI 创建包。若要安装 `dotnet` CLI，请参阅[安装 NuGet 客户端工具](#)。从 Visual Studio 2017 开始，dotnet CLI 包含在 .NET Core 工作负载中。

对于使用 [SDK 样式格式](#) 的 .NET Core 和 .NET Standard 项目，以及任何其他 SDK 样式项目，NuGet 直接使用项目文件中的信息创建包。若要查看分步教程，请参阅[使用 dotnet CLI 创建 .NET Standard 包](#)或[使用 Visual Studio 创建 .NET Standard 包](#)。

`msbuild -t:pack` 在功能上等效于 `dotnet pack`。若要使用 MSBuild 进行生成，请参阅[使用 MSBuild 创建 NuGet 包](#)。

IMPORTANT

本主题适用于 [SDK 样式](#) 的项目，通常是 .NET Core 和 .NET Standard 项目。

设置属性

创建包需要以下属性。

- `PackageId`，包标识符，在托管包的库中必须是唯一的。如果未指定，默认值为 `AssemblyName`。
- `Version`，窗体 Major.Minor.Patch[-Suffix] 中特定的版本号，其中 -Suffix 标识[预发布版本](#)。如果未指定，默认值为 1.0.0。
- 包标题应出现在主机上（例如 nuget.org）
- `Authors`，作者和所有者信息。如果未指定，默认值为 `AssemblyName`。
- `Company`，公司名称。如果未指定，默认值为 `AssemblyName`。

在 Visual Studio 中，可以在项目属性中设置这些值（在解决方案资源管理器中右键单击项目，选择“属性”，然后选择“包”选项卡）。也可以直接在项目文件（`.csproj`）中设置这些属性。

```
<PropertyGroup>
  <PackageId>AppLogger</PackageId>
  <Version>1.0.0</Version>
  <Authors>your_name</Authors>
  <Company>your_company</Company>
</PropertyGroup>
```

IMPORTANT

为包提供一个在 nuget.org 中唯一或你使用的任何包资源的标识符。

以下示例显示了包含这些属性的简单、完整的项目文件。（可以使用 `dotnet new classlib` 命令创建新的默认项目。）

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>netstandard2.0</TargetFramework>
    <PackageId>AppLogger</PackageId>
    <Version>1.0.0</Version>
    <Authors>your_name</Authors>
    <Company>your_company</Company>
  </PropertyGroup>
</Project>
```

另外，还可以设置可选属性，例如 `Title`、`PackageDescription` 和 `PackageTags`，如 [MSBuild 包目标](#)、[控制依赖项资产](#) 和 [NuGet 元数据属性](#) 中所述。

NOTE

对于面向公共使用而生成的包，请特别注意 `PackageTags` 属性，因为这些标记可帮助其他人查找包并了解其用途。

有关声明依赖项并指定版本号的详细信息，请参阅[项目文件中的包引用和包版本控制](#)。还可以使用

`<IncludeAssets>` 和 `<ExcludeAssets>` 特性直接在包中呈现依赖项的资产。有关详细信息，请参阅[控制依赖项资产](#)。

添加可选说明字段

在包的 NuGet.org 页面上所示的包可选说明从用于 `<description></description>` 文件中的 `.csproj` 拉取，或者通过 `$description` `.nuspec` 文件中的拉取。

有关 `description` 字段的示例，请参阅[.NET 包的文件中的以下 XML 文本 .csproj](#)：

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <PackageId>Azure.Storage.Blobs</PackageId>
    <Version>12.4.0</Version>
    <PackageTags>Microsoft Azure Storage
Blobs;Microsoft;Azure;Blobs;Blob;Storage;StorageScalable</PackageTags>
    <Description>
      This client library enables working with the Microsoft Azure Storage Blob service for storing binary and
      text data.
      For this release see notes - https://github.com/Azure/azure-sdk-for-
      net/blob/master/sdk/storage/Azure.Storage.Blobs/README.md and https://github.com/Azure/azure-sdk-for-
      net/blob/master/sdk/storage/Azure.Storage.Blobs/CHANGELOG.md
      in addition to the breaking changes https://github.com/Azure/azure-sdk-for-
      net/blob/master/sdk/storage/Azure.Storage.Blobs/BreakingChanges.txt
      Microsoft Azure Storage quickstarts and tutorials - https://docs.microsoft.com/en-us/azure/storage/
      Microsoft Azure Storage REST API Reference - https://docs.microsoft.com/en-us/rest/api/storageservices/
      REST API Reference for Blob Service - https://docs.microsoft.com/en-us/rest/api/storageservices/blob-
      service-rest-api
    </Description>
  </PropertyGroup>
</PropertyGroup>
```

选择唯一的包标识符并设置版本号

包标识符和版本号是项目中最重要的两个值，因为它们唯一标识包中包含的确切代码。

针对包标识符的最佳做法：

- **唯一性**：标识符必须在 nuget.org 或托管包的任意库中是唯一的。确定标识符之前，请搜索适用库以检查该名称是否已使用。为了避免冲突，最好使用公司名作为标识符的第一部分（例如 `Contoso.`）。

- **类似于命名空间的名称**: 遵循类似于 .NET 中命名空间的模式, 使用点表示法(而不是连字符)。例如, 使用 `Contoso.Utility.UsefulStuff` 而不是 `Contoso-Utility-UsefulStuff` 或 `Contoso_Utility_UsefulStuff`。当包标识符与代码中使用的命名空间相匹配时, 这个方法也很有用。
- **示例包**: 如果生成展示如何使用另一个包的示例代码包, 请附加 `.Sample` 作为标识符的后缀, 就像 `Contoso.Utility.UsefulStuff.Sample` 中一样。(当然, 示例包会在其他包上有依赖项。) 创建示例包时, 请在 `<IncludeAssets>` 中使用 `contentFiles` 值。在 `content` 文件夹中, 在名为 `\Samples\<identifier>` 的文件夹中排列示例代码, 与 `\Samples\Contoso.Utility.UsefulStuff.Sample` 中相似。

针对包版本的最佳做法:

- 一般情况下, 将包的版本设置为与项目(或程序集)相匹配, 但这不是必须的。如果将包限制为单个程序集, 那么这是一个简单的问题。总的来说, 请记住解析依赖项时, NuGet 自己处理包版本而不是程序集版本。
- 使用非标准版本方案时, 请确保考虑使用 NuGet 版本控制规则, 如[包版本控制](#)中所述。NuGet 主要与 [semver 2 兼容](#)。

有关依赖项解析的信息, 请参阅[使用 PackageReference 的依赖项解析](#)。对于可能有助于更好地理解版本控制的较旧信息, 请参阅此系列博客文章。

- [第 1 部分: 解决 DLL 地狱](#)
- [第 2 部分: 核心算法](#)
- [第 3 部分: 通过绑定重定向实现统一](#)

运行 pack 命令

若要从项目中生成 NuGet 包(`.nupkg` 文件), 运行 `dotnet pack` 命令, 它也会自动生成项目:

```
# Uses the project file in the current folder by default  
dotnet pack
```

输出将显示 `.nupkg` 文件的路径。

```
Microsoft (R) Build Engine version 15.5.180.51428 for .NET Core  
Copyright (C) Microsoft Corporation. All rights reserved.  
  
Restore completed in 29.91 ms for D:\proj\AppLoggerNet\AppLogger\AppLogger.csproj.  
AppLogger -> D:\proj\AppLoggerNet\AppLogger\bin\Debug\netstandard2.0\AppLogger.dll  
Successfully created package 'D:\proj\AppLoggerNet\AppLogger\bin\Debug\AppLogger.1.0.0.nupkg'.
```

在生成期间自动生成包

若要在运行 `dotnet build` 时自动运行 `dotnet pack`, 请将以下行添加到 `<PropertyGroup>` 中的项目文件内:

```
<GeneratePackageOnBuild>true</GeneratePackageOnBuild>
```

在解决方案上运行 `dotnet pack` 时, 会打包解决方案中可打包的所有项目(属性设置为 `true`)。

NOTE

自动生成包时, 打包时间会增加项目的生成时间。

测试包安装

发布包前, 通常需要测试将包安装到项目的过程。测试确保所有文件一定在项目中正确的位置结束。

可以在 Visual Studio 中手动测试安装, 或是使用常规[包安装步骤](#)在命令行上测试。

IMPORTANT

包是不可变的。如果更正了问题, 请更改包的内容并再次打包, 重新测试时, 仍将使用旧版本的包, 直到清除全局包文件夹。在测试每次生成时不使用唯一预发布标签的包时, 这尤为重要。

后续步骤

创建包(`.nupkg` 文件)后, 可以将其发布到选择的库, 如[发布包](#)中所述。

你可能还希望扩展包的功能, 或者支持其他方案, 如以下主题所述:

- [包版本控制](#)
- [支持多个目标框架](#)
- [添加包图标](#)
- [源和配置文件的转换](#)
- [本地化](#)
- [预发布版本](#)
- [设置包类型](#)
- [使用 COM 互操作程序集创建包](#)

最后, 有需要注意的其他包类型:

- [本机包](#)
- [符号包](#)

使用 nuget.exe CLI 创建包

2020/4/8 • [Edit Online](#)

无论包是什么用途或者它包含什么代码，均使用其中一个 CLI 工具（`nuget.exe` 或 `dotnet.exe`）将该功能打包进可以与任意数量的其他开发人员共享且可以由其使用的组件中。若要安装 NuGet CLI 工具，请参阅[安装 NuGet 客户端工具](#)。请注意，Visual Studio 不会自动包含 CLI 工具。

- 对于非 SDK 样式项目（通常为 .NET Framework 项目），按照本文中所述的步骤创建包。有关使用 Visual Studio 和 `nuget.exe` CLI 的分步说明，请参阅[创建和发布 .NET Framework 包](#)。
- 对于使用 [SDK 样式格式](#) 的 .NET Core 和 .NET Standard 项目，以及任何其他 SDK 样式项目，请参阅[使用 dotnet CLI 创建 NuGet 包](#)。
- 对于从 `packages.config` 迁移到 `PackageReference` 的项目，使用 `msbuild -t:pack`。

从技术角度来讲，NuGet 包仅仅是一个使用 `.nupkg` 扩展重命名且其中的内容与特定约定相匹配的 ZIP 文件。本主题介绍创建满足这些约定的包的详细流程。

包以编译的代码（程序集）、符号和/或需要作为包传送的其他文件开头（请参阅[概述和工作流](#)）。尽管可以使用项目文件中信息的描述来保持编译程序集和包的同步，但此流程独立于编译或生成进入包的文件。

IMPORTANT

本主题适用于非 SDK 样式项目，通常是除使用 Visual Studio 2017 和更高版本以及 NuGet 4.0+ 的 .NET Core 和 .NET Standard 项目之外的项目。

确定要打包的程序集

大多数通用包包含一个或多个其他开发人员可在其自己项目中使用的程序集。

- 通常情况下，最好每个 NuGet 包有一个程序集，前提是每个程序集可独立正常运行。例如，如果有一个 `Utilities.dll` 依赖于 `Parser.dll` 且 `Parser.dll` 可独立正常运行，则为每一个创建一个包。这样做使得开发人员能够独立于 `Utilities.dll` 使用 `Parser.dll`。
- 如果库由多个不能独立正常运行的程序集构成，那么可以将它们合并到一个包中。使用上面的示例，如果 `Parser.dll` 包含仅由 `Utilities.dll` 使用的代码，那么可以将 `Parser.dll` 保留在同一包中。
- 同样，如果 `Utilities.dll` 依赖于 `Utilities.resources.dll` 并且后者不能独立正常运行，则将两者放入同一包中。

事实上，资源是一种特殊情况。当包安装到项目中时，NuGet 自动将程序集引用添加到包的 DLL，不包含命名为 `.resources.dll` 的内容，因为它们被假定为本地化的附属程序集（请参阅[创建本地化包](#)）。为此，请避免对包含基本包代码的文件使用 `.resources.dll`。

如果库包含 COM 互操作程序集，请遵循[使用 COM 互操作程序集创建包](#)中的其他准则。

.nuspec 文件的角色和结构

一旦知道需要打包的文件后，下一步是在 `.nuspec` XML 文件中创建包清单。

清单：

- 介绍包的内容和自己是否包含在包中。

2. 驱动包的创建并且指示 NuGet 如何将包安装到项目。例如，清单识别其他包依赖项，这样，NuGet 还可以在安装主包时安装这些依赖项。
3. 如下所示，包含所需属性和可选属性。有关确切的详细信息（包括此处未提及的其他属性），请参阅 [.nuspec 引用](#)。

所需属性：

- 包标识符在承载包的库中必须是唯一的。
- 窗体 Major.Minor.Patch[-Suffix] 中特定的版本号，其中 -Suffix 标识[预发布版本](#)
- 包标题应出现在主机上（例如 nuget.org）
- 创建者和所有者信息。
- 对包的详细说明。

常见可选属性：

- 发行说明
- 版权信息
- 对[Visual Studio 中的包管理器 UI](#)的简短说明
- 区域设置 ID
- 项目 URL
- 作为表达式或文件的许可证（`licenseUrl` 将被弃用，请使用 [license nuspec 元数据元素](#)）
- 图标 URL
- 依赖项和引用的列表
- 协助库搜索的标记

以下是典型（但虚构的）`.nuspec` 文件，其中注释介绍属性：

```

<?xml version="1.0"?>
<package xmlns="http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd">
  <metadata>
    <!-- The identifier that must be unique within the hosting gallery -->
    <id>Contoso.Utility.UsefulStuff</id>

    <!-- The package version number that is used when resolving dependencies -->
    <version>1.8.3-beta</version>

    <!-- Authors contain text that appears directly on the gallery -->
    <authors>Dejana Tesic, Rajeev Dey</authors>

    <!--
        Owners are typically nuget.org identities that allow gallery
        users to easily find other packages by the same owners.
      -->
    <owners>dejanatc, rjdey</owners>

    <!-- Project URL provides a link for the gallery -->
    <projectUrl>http://github.com/contoso/UsefulStuff</projectUrl>

    <!-- License information is displayed on the gallery -->
    <license type="expression">Apache-2.0</license>

    <!-- The icon is used in Visual Studio's package manager UI -->
    <iconUrl>http://github.com/contoso/UsefulStuff/nuget_icon.png</iconUrl>

    <!--
        If true, this value prompts the user to accept the license when
        installing the package.
      -->
    <requireLicenseAcceptance>false</requireLicenseAcceptance>

    <!-- Any details about this particular release -->
    <releaseNotes>Bug fixes and performance improvements</releaseNotes>

    <!--
        The description can be used in package manager UI. Note that the
        nuget.org gallery uses information you add in the portal.
      -->
    <description>Core utility functions for web applications</description>

    <!-- Copyright information -->
    <copyright>Copyright ©2016 Contoso Corporation</copyright>

    <!-- Tags appear in the gallery and can be used for tag searches -->
    <tags>web utility http json url parsing</tags>

    <!-- Dependencies are automatically installed when the package is installed -->
    <dependencies>
      <dependency id="Newtonsoft.Json" version="9.0" />
    </dependencies>
  </metadata>

  <!-- A readme.txt to display when the package is installed -->
  <files>
    <file src="readme.txt" target="" />
  </files>
</package>

```

有关声明依赖项并指定版本号的详细信息，请参阅 [packages.config](#) 和 [包版本控制](#)。还可以使用 `dependency` 元素上的 `include` 和 `exclude` 特性直接在包中结合依赖项的资产。请参阅 [.nuspec 引用 - 依赖项](#)。

因为清单包含在从清单创建的包中，所以可以通过检查现有包查找任意数目的其他示例。计算机上的 global-packages 文件夹是一个不错的源，其位置由以下命令返回：

```
nuget locals -list global-packages
```

转到任意 package\version 文件夹, 将 `.nupkg` 文件复制到 `.zip` 文件, 然后打开该 `.zip` 文件并检查其中的 `.nuspec`。

NOTE

从 Visual Studio 项目创建 `.nuspec` 时, 清单包含生成包时被项目信息替换的令牌。请参阅[从 Visual Studio 项目创建 .nuspec](#)。

创建 .nuspec 文件

创建完整的清单通常以通过以下其中一种方法生成的基础 `.nuspec` 文件开始:

- [基于约定的工作目录](#)
- [程序集 DLL](#)
- [Visual Studio 项目](#)
- [包含默认值的新文件](#)

然后手动编辑文件, 使它可以在最终包中描述所需的确切内容。

IMPORTANT

生成 `.nuspec` 文件包含的占位符必须在使用 `nuget pack` 命令创建包前修改。如果 `.nuspec` 包含任意占位符, 则命令失败。

来自基于约定的工作目录

因为 NuGet 包仅仅是使用 `.nupkg` 扩展名重命名的 ZIP 文件, 所以在本地文件系统上创建所需文件夹结构, 然后从该结构直接创建 `.nuspec` 文件通常最简单。然后, `nuget pack` 命令自动将所有文件添加到该文件夹结构(不包含以 `.` 开头的文件夹, 从而在同一结构中保留专用文件)。

此方法的优势是无需在清单中指定需要包含在包中的文件(如本主题后面所述)。可以轻松使得生成进程生成进入包中的确切文件夹结构, 并且可以轻松包含不是项目一部分的其他文件:

- 应注入目标项目的内容和源代码。
- PowerShell 脚本
- 转换到现有配置和项目中的源代码文件。

文件夹约定如下所示:

约定	说明	示例
(根)	readme.txt 的位置	包安装时, Visual Studio 在包根目录中显示 readme.txt 文件。
lib/{tfm}	特定目标框架名字对象 (TFM) 的程序集 (<code>.dll</code>)、文档 (<code>.xml</code>) 和符号 (<code>.pdb</code>) 文件	程序集添加为引用, 以用于编译和运行时; <code>.xml</code> 和 <code>.pdb</code> 复制到项目文件夹中。有关创建框架特定目标子文件夹的详细信息, 请参阅 支持多个目标框架 。
ref/{tfm}	给定目标框架名字对象 (TFM) 的程序集 (<code>.dll</code>) 和符号 (<code>.pdb</code>) 文件	程序集添加为引用, 仅用于编译时; 因此, 不会向项目 Bin 文件夹复制任何内容。

runtimes	特定于体系结构的程序集 (.dll)、符号 (.pdb) 和本机源 (.pri) 文件	程序集添加为引用，仅用于运行时；其他文件复制到项目文件夹中。在 AnyCPU 文件夹下应始终具有特定于相应的 (TFM) /ref/{tfm} 的程序集，以提供相应的编译时程序集。请参阅 支持多个目标框架 。
内容	任意文件	内容复制到项目根目录。将“内容”文件夹视为最终使用包的目标应用程序的根目录。若要使包在应用程序的 /images 文件夹中添加图片，请将其置于包的 content/images 文件夹中。
生成	(3.x+) MSBuild .targets 和 .props 文件	自动插入到项目。
buildMultiTargeting	(4.0+) 跨框架目标的 MSBuild .targets 和 .props 文件	自动插入到项目。
buildTransitive	(5.0+) 以可传递的方式流入任意使用项目的 MSBuild .targets 和 .props 文件。请参阅 功能 页。	自动插入到项目。
工具	可从包管理器控制台访问 Powershell 脚本和程序	tools 文件夹添加到仅适用于包管理器控制台的 PATH 环境变量（尤其是不作为 MSBuild 集在生成项目时添加到 PATH）。

因为文件夹结构可能包含任意数量的目标框架的任意数量的程序集，所以此方法在创建支持多个框架的包时是必要的。

在任何情况下，一旦准备好所需文件夹结构，则在该文件夹中运行以下命令以创建 .nuspec 文件：

```
nuget spec
```

同样，生成的 .nuspec 不包含文件夹结构中的文件的显式引用。包创建时，NuGet 自动包含所有文件。但是，还需在清单的其他部分中编辑占位符。

来自程序集 DLL

在从程序集创建包的简单情况下，可以使用以下命令从程序集中的元数据生成 .nuspec 文件：

```
nuget spec <assembly-name>.dll
```

使用此窗体将清单中的一些占位符替换为程序集的特定值。例如，<id> 属性设为程序集名称，<version> 设为程序集版本。但是，清单中的其他属性在程序集中没有匹配值，因此仍包含占位符。

来自 Visual Studio 项目

从 .csproj 或 .vbproj 文件创建 .nuspec 较为方便，因为其他已安装到这些项目的包会自动引用为依赖项。只需使用与项目文件相同的文件夹中的命令：

```
# Use in a folder containing a project file <project-name>.csproj or <project-name>.vbproj
nuget spec
```

生成的 `<project-name>.nuspec` 文件包含在打包时替换为项目值的令牌，包含对所有其他已安装的包的引用。

若将程序包依赖项包含在 `.nuspec` 中，请使用 `nuget pack`，并从生成的 `.nupkg` 文件获取 `.nuspec` 文件。例如，使用以下命令。

```
# Use in a folder containing a project file <project-name>.csproj or <project-name>.vbproj
nuget pack myproject.csproj
```

令牌在项目属性的两侧由 `$` 符号分隔。例如，通过此方法生成的清单中的 `<id>` 值通常如下所示：

```
<id>$id$</id>
```

此令牌在打包时替换为项目文件的 `AssemblyName` 值。有关项目值到 `.nuspec` 令牌的确切映射，请参阅[替换令牌引用](#)。

借助令牌，更新项目时可以不再需要更新关键值，例如 `.nuspec` 中的版本号。（如果需要，随时可以使用文本值替换令牌）。

注意在 Visual Studio 项目中工作时，有多个可用的其他打包选项，如稍后的[运行 NuGet 包生成 .nupkg 文件](#)所述。

解决方案级别包

仅适用于 NuGet 2.x。在 NuGet 3.0+ 中不可用。

NuGet 2.x 支持为包管理器控制台（`tools` 文件夹的内容）安装工具或其他命令的解决方案级别包的概念，但是不会将引用、内容或生成自定义添加到解决方案中的任何项目中。该包在其直接 `lib`、`content` 或 `build` 文件夹中未包含文件，并且它的依赖项各自的 `lib`、`content` 或 `build` 文件夹中也没有文件。

NuGet 在 `.nuget` 文件夹的 `packages.config` 文件（而不是项目的 `packages.config` 文件）中，跟踪安装的解决方案级别包。

包含默认值的新文件

以下命令使用占位符创建默认清单，这可确保一开始就使用正确的文件结构：

```
nuget spec [<package-name>]
```

如果忽略 `<package-name>`，生成的文件是 `Package.nuspec`。如果提供 `Contoso.Utility.UsefulStuff` 等名称，则文件是 `Contoso.Utility.UsefulStuff.nuspec`。

生成的 `.nuspec` 包含值的占位符（例如 `projectUrl`）。请确保在使用文件创建最终 `.nupkg` 文件前编辑该文件。

选择唯一的包标识符并设置版本号

包标识符（`<id>` 元素）和版本号（`<version>` 元素）是清单中最重要的两个值，因为它们唯一标识包中包含的确切代码。

针对包标识符的最佳做法：

- 唯一性**：标识符必须在 `nuget.org` 或托管包的任意库中是唯一的。确定标识符之前，请搜索适用库以检查该名称是否已使用。为了避免冲突，最好使用公司名作为标识符的第一部分（例如 `Contoso.`）。
- 类似于命名空间的名称**：遵循类似于 .NET 中命名空间的模式，使用点表示法（而不是连字符）。例如，使用 `Contoso.Utility.UsefulStuff` 而不是 `Contoso-Utility-UsefulStuff` 或 `Contoso_Utility_UsefulStuff`。当包标识符与代码中使用的命名空间相匹配时，这个方法也很有用。
- 示例包**：如果生成展示如何使用另一个包的示例代码包，请附加 `.Sample` 作为标识符的后缀，就像 `Contoso.Utility.UsefulStuff.Sample` 中一样。（当然，示例包会在其他包上有依赖项。）创建示例包时，使用前

面介绍的基于约定的工作目录方法。在 `content` 文件夹中，在名为 `\Samples\<identifier>` 的文件夹中排列示例代码，与 `\Samples\Contoso.Utility.UsefulStuff.Sample` 中相似。

针对包版本的最佳做法：

- 一般情况下，将包的版本设置为与库相匹配，但这不是必须的。如之前在[确定要打包的程序集中所述](#)，将包限制到单个程序集是一个简单的问题。总的来说，请记住解析依赖项时，NuGet 自己处理包版本而不是程序集版本。
- 使用非标准版本方案时，请确保考虑使用 NuGet 版本控制规则，如[包版本控制](#)中所述。

以下一系列简短博客文章也有助于理解版本控制：

- [第 1 部分：解决 DLL 地狱](#)
- [第 2 部分：核心算法](#)
- [第 3 部分：通过绑定重定向实现统一](#)

添加自述文件和其他文件

若要直接指定要包含在包中的文件，请使用 `.nuspec` 中的 `<files>` 节点，其遵循 `<metadata>` 标记：

```
<?xml version="1.0"?>
<package xmlns="http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd">
  <metadata>
    <!-- ... -->
  </metadata>
  <files>
    <!-- Add a readme -->
    <file src="readme.txt" target="" />

    <!-- Add files from an arbitrary folder that's not necessarily in the project -->
    <file src="..\SomeRoot\**\*.*" target="" />
  </files>
</package>
```

TIP

使用基于约定的工作目录方法时，可以将 `readme.txt` 置于包根目录中并将其他内容置于 `content` 文件夹中。清单中不需要 `<file>` 元素。

如果包根目录中包含名为 `readme.txt` 的文件，Visual Studio 在直接安装包之后立即将该文件的内容显示为纯文本。（对于安装为依赖项的包，不会显示自述文件）。例如，下面是 `HtmlAgilityPack` 包的自述文件的显示方式：

```
readme.txt - X
1 -----
2 ----- Html Agility Pack Nuget Readme -----
3 -----
4 -----
5 -----Silverlight 4 and Windows Phone 7.1+ projects-----
6 To use XPATH features: System.Xml.XPath.dll from the Silverlight 4 SDK must be referenced.
7 This is normally found at
8 %ProgramFiles(x86)%\Microsoft SDKs\Microsoft SDKs\Silverlight\v4.0\Libraries\Client
9 or
10 %ProgramFiles%\Microsoft SDKs\Microsoft SDKs\Silverlight\v4.0\Libraries\Client
11 -----
12 -----Silverlight 5 projects-----
13 To use XPATH features: System.Xml.XPath.dll from the Silverlight 5 SDK must be referenced.
14 This is normally found at
15 %ProgramFiles(x86)%\Microsoft SDKs\Microsoft SDKs\Silverlight\v5.0\Libraries\Client
16 or
17 %ProgramFiles%\Microsoft SDKs\Microsoft SDKs\Silverlight\v5.0\Libraries\Client
18
```

NOTE

如果 `.nuspec` 文件中包含空 `<files>` 节点，则 NuGet 不会在包中包含除 `lib` 文件夹中的内容以外的任何其他内容。

将 MSBuild 属性和目标包含到包中

在某些情况下，可能需要在使用包的项目中添加自定义生成目标或属性，例如生成期间运行自定义工具或进程。

通过将文件置于项目的 `\build` 文件夹中的窗体 `<package_id>.targets` 或 `<package_id>.props`（例如 `Contoso.Utility.UsefulStuff.targets`）中执行此操作。

根 `\build` 文件夹中的文件被视为适用于所有目标框架。若要提供特定于框架的文件，首先将其放置到合适的子文件夹中，如下所示：

```
\build
\netstandard1.4
    \Contoso.Utility.UsefulStuff.props
    \Contoso.Utility.UsefulStuff.targets
\net462
    \Contoso.Utility.UsefulStuff.props
    \Contoso.Utility.UsefulStuff.targets
```

然后在 `.nuspec` 文件中，确保参阅 `<files>` 节点中的这些文件：

```
<?xml version="1.0"?>
<package >
  <metadata minClientVersion="2.5">
    <!-- ... -->
  </metadata>
  <files>
    <!-- Include everything in \build -->
    <file src="build\**" target="build" />

    <!-- Other files -->
    <!-- ... -->
  </files>
</package>
```

在包中包含 MSBuild 属性和目标是 [NuGet 2.5 中引入的功能](#)，因此建议将 `minClientVersion="2.5"` 属性添加到 `metadata` 元素，以指示使用该包所需的最低 NuGet 客户端版本。

当 NuGet 使用 `\build` 文件安装包时，它在指向 `.targets` 和 `.props` 文件的项目文件中添加 MSBuild `<Import>` 元素。（`.props` 添加到项目文件顶部；`.targets` 添加到底部。）为每个目标框架添加了单独的条件 MSBuild `<Import>` 元素。

可将跨框架目标的 MSBuild `.props` 和 `.targets` 文件置于 `\buildMultiTargeting` 文件夹中。在包安装期间，NuGet 将相应的 `<Import>` 元素添加到项目文件中，前提是目标框架未设置（MSBuild 属性 `$(TargetFramework)` 必须为空）。

对于 NuGet 3.x，目标不添加到项目，而是通过 `{projectName}.nuget.g.targets` 和 `{projectName}.nuget.g.props` 提供。

运行 nuget 包以生成 .nupkg 文件

使用程序集或基于约定的工作目录时，通过使用 `.nuspec` 文件运行 `nuget pack` 创建包，使用特定的文件名替换 `<project-name>`：

```
nuget pack <project-name>.nuspec
```

使用 Visual Studio 项目时，使用项目文件运行 `nuget pack`，该项目文件自动加载项目的 `.nuspec` 文件并使用项目文件中的值替换其中的任意令牌：

```
nuget pack <project-name>.csproj
```

NOTE

令牌替换需要直接使用项目文件，因为项目是令牌值的源。如果将 `nuget pack` 用于 `.nuspec` 文件，不会发生令牌替换。

在所有情况下，`nuget pack` 不包含句点开头的文件夹，例如 `.git` 或 `.hg`。

NuGet 指示需要更正的 `.nuspec` 文件中是否有错误，例如忘记更改清单中的占位符值。

一旦 `nuget pack` 成功，就有一个可以发布到合适库的 `.nupkg` 文件，如[发布包](#)中所述。

TIP

一个检查创建后的包的有用方法是在[包资源管理器](#)工具中打开它。这会为你提供包内容及其清单的图形视图。还可以将生成的 `.nupkg` 文件重命名为 `.zip` 文件并直接浏览其内容。

附加选项

可通过 `nuget pack` 使用多种命令行开关，以排除文件、替代清单中的版本号和更改输出文件夹等。有关完整列表，请参考[包命令引用](#)。

以下是 Visual Studio 项目中的一些常见选项：

- **已引用项目**：如果项目引用其他项目，可以使用 `-IncludeReferencedProjects` 选项，将已引用项目添加为包的一部分，或添加为依赖项：

```
nuget pack MyProject.csproj -IncludeReferencedProjects
```

此包含过程是递归的，所以如果 `MyProject.csproj` 引用项目 B 和 C，且这些项目引用 D、E 和 F，那么 B、C、D、E 和 F 中的文件包含在包中。

如果引用的项目包含其自身的 `.nuspec` 文件，那么 NuGet 将该引用的项目添加为依赖项。需要单独打包和发布该项目。

- **生成配置**: 默认情况下，NuGet 使用项目文件中的默认生成配置集(通常是“调试”)。若要从不同的生成配置(例如“发布”)中打包文件，使用包含以下配置的 `-properties` 选项：

```
nuget pack MyProject.csproj -properties Configuration=Release
```

- **符号**: 若要包含允许使用者在调试器中单步执行包代码的符号，请使用 `-symbols` 选项：

```
nuget pack MyProject.csproj -symbols
```

测试包安装

发布包前，通常需要测试将包安装到项目的过程。测试确保所有文件一定在项目中正确的位置结束。

可以在 Visual Studio 中手动测试安装，或是使用常规[包安装步骤](#)在命令行上测试。

对于自动测试，基本流程如下所示：

1. 将 `.nupkg` 文件复制到本地文件夹。
2. 使用 `nuget sources add -name <name> -source <path>` 命令将文件夹添加到包源(请参阅[nuget 源](#))。请注意，只需在任意给定计算机上设置一次此本地源。
3. 使用 `nuget install <packageID> -source <name>` (其中 `<name>` 与提供给 `nuget sources` 的源名称相匹配)从该源安装包。指定源可确保包从该源中单独安装。
4. 检查文件系统以检查文件是否正确安装。

后续步骤

创建包(`.nupkg` 文件)后，可以将其发布到选择的库，如[发布包](#)中所述。

你可能还希望扩展包的功能，或者支持其他方案，如以下主题所述：

- [包版本控制](#)
- [支持多个目标框架](#)
- [源和配置文件的转换](#)
- [本地化](#)
- [预发布版本](#)
- [设置包类型](#)
- [使用 COM 互操作程序集创建包](#)

最后，有需要注意的其他包类型：

- [本机包](#)
- [符号包](#)

使用 MSBuild 创建 NuGet 包

2020/4/8 • [Edit Online](#)

从代码中创建 NuGet 包时，会将该功能打包到一个组件中，该组件可以与任意数量的其他开发人员共享和使用。本文介绍如何使用 MSBuild 创建包。MSBuild 预安装了包含 NuGet 的所有 Visual Studio 工作负载。此外，还可以通过 dotnet CLI 借助 `dotnet msbuild` 来使用 MSBuild。

对于使用 [SDK 样式格式](#) 的 .NET Core 和 .NET Standard 项目，以及任何其他 SDK 样式项目，NuGet 直接使用项目文件中的信息创建包。对于使用 `<PackageReference>` 的非 SDK 样式的项目，NuGet 还使用项目文件来创建包。

默认情况下，SDK 样式的项目具有可用的包功能。对于非 SDK 样式的 PackageReference 项目，需要将 `NuGet.Build.Tasks.Pack` 包添加到项目依赖项中。有关 MSBuild 包目标的详细信息，请参阅[作为 MSBuild 目标的 NuGet 包和还原](#)。

用于创建包的命令 `msbuild -t:pack`，其功能相当于 `dotnet pack`。

IMPORTANT

本主题适用于 [SDK 样式](#) 的项目（通常是 .NET Core 和 .NET Standard 项目）以及使用 `PackageReference` 的非 SDK 样式的项目。

设置属性

创建包需要以下属性。

- `PackageId`，包标识符，在托管包的库中必须是唯一的。如果未指定，默认值为 `AssemblyName`。
- `Version`，窗体 Major.Minor.Patch[-Suffix] 中特定的版本号，其中 -Suffix 标识[预发布版本](#)。如果未指定，默认值为 1.0.0。
- 包标题应出现在主机上（例如 nuget.org）
- `Authors`，作者和所有者信息。如果未指定，默认值为 `AssemblyName`。
- `Company`，公司名称。如果未指定，默认值为 `AssemblyName`。

此外，如要打包使用 `PackageReference` 的非 SDK 类项目，则需要满足以下要求：

- `PackageOutputPath`（调用 `pack` 时生成的包的输出文件夹）。

在 Visual Studio 中，可以在项目属性中设置这些值（在解决方案资源管理器中右键单击项目，选择“属性”，然后选择“包”选项卡）。也可以直接在项目文件 (.csproj) 中设置这些属性。

```
<PropertyGroup>
  <PackageId>ClassLibDotNetStandard</PackageId>
  <Version>1.0.0</Version>
  <Authors>your_name</Authors>
  <Company>your_company</Company>
</PropertyGroup>
```

IMPORTANT

为包提供一个在 nuget.org 中唯一或你使用的任何包资源的标识符。

以下示例显示了包含这些属性的简单、完整的项目文件。

```
<Project Sdk="Microsoft.NET.Sdk">
<PropertyGroup>
<TargetFramework>netstandard2.0</TargetFramework>
<PackageId>ClassLibDotNetStandard</PackageId>
<Version>1.0.0</Version>
<Authors>your_name</Authors>
<Company>your_company</Company>
</PropertyGroup>
</Project>
```

另外，还可以设置可选属性，例如 `Title`、`PackageDescription` 和 `PackageTags`，如 [MSBuild 包目标、控制依赖项资产](#) 和 [NuGet 元数据属性](#) 中所述。

NOTE

对于面向公共使用而生成的包，请特别注意 `PackageTags` 属性，因为这些标记可帮助其他人查找包并了解其用途。

有关声明依赖项并指定版本号的详细信息，请参阅[项目文件中的包引用和包版本控制](#)。还可以使用

`<IncludeAssets>` 和 `<ExcludeAssets>` 特性直接在包中呈现依赖项的资产。有关详细信息，请参阅[控制依赖项资产](#)。

添加可选说明字段

在包的 NuGet.org 页面上所示的包可选说明从用于 `<description></description>` 文件中的 `.csproj` 拉取，或者通过 `$description.nuspec` 文件中的拉取。

有关 `description` 字段的示例，请参阅[.NET 包的文件中的以下 XML 文本 .csproj](#)：

```
<Project Sdk="Microsoft.NET.Sdk">
<PropertyGroup>
<PackageId>Azure.Storage.Blobs</PackageId>
<Version>12.4.0</Version>
<PackageTags>Microsoft Azure Storage
Blobs;Microsoft;Azure;Blobs;Blob;Storage;StorageScalable</PackageTags>
<Description>
    This client library enables working with the Microsoft Azure Storage Blob service for storing binary
    and text data.

    For this release see notes - https://github.com/Azure/azure-sdk-for-
    net/blob/master/sdk/storage/Azure.Storage.Blobs/README.md and https://github.com/Azure/azure-sdk-for-
    net/blob/master/sdk/storage/Azure.Storage.Blobs/CHANGELOG.md

    in addition to the breaking changes https://github.com/Azure/azure-sdk-for-
    net/blob/master/sdk/storage/Azure.Storage.Blobs/BreakingChanges.txt

    Microsoft Azure Storage quickstarts and tutorials - https://docs.microsoft.com/en-us/azure/storage/
    Microsoft Azure Storage REST API Reference - https://docs.microsoft.com/en-us/rest/api/storageservices/
    REST API Reference for Blob Service - https://docs.microsoft.com/en-us/rest/api/storageservices/blob-
    service-rest-api
</Description>
</PropertyGroup>
</PropertyGroup>
```

选择唯一的包标识符并设置版本号

包标识符和版本号是项目中最重要的两个值，因为它们唯一标识包中包含的确切代码。

针对包标识符的最佳做法：

- **唯一性**：标识符必须在 nuget.org 或托管包的任意库中是唯一的。确定标识符之前，请搜索适用库以检查该名称是否已使用。为了避免冲突，最好使用公司名作为标识符的第一部分（例如 `Contoso.`）。

- **类似于命名空间的名称**: 遵循类似于 .NET 中命名空间的模式, 使用点表示法(而不是连字符)。例如, 使用 `Contoso.Utility.UsefulStuff` 而不是 `Contoso-Utility-UsefulStuff` 或 `Contoso_Utility_UsefulStuff`。当包标识符与代码中使用的命名空间相匹配时, 这个方法也很有用。
- **示例包**: 如果生成展示如何使用另一个包的示例代码包, 请附加 `.Sample` 作为标识符的后缀, 就像 `Contoso.Utility.UsefulStuff.Sample` 中一样。(当然, 示例包会在其他包上有依赖项。) 创建示例包时, 请在 `<IncludeAssets>` 中使用 `contentFiles` 值。在 `content` 文件夹中, 在名为 `\Samples\<identifier>` 的文件夹中排列示例代码, 与 `\Samples\Contoso.Utility.UsefulStuff.Sample` 中相似。

针对包版本的最佳做法:

- 一般情况下, 将包的版本设置为与项目(或程序集)相匹配, 但这不是必须的。如果将包限制为单个程序集, 那么这是一个简单的问题。总的来说, 请记住解析依赖项时, NuGet 自己处理包版本而不是程序集版本。
- 使用非标准版本方案时, 请确保考虑使用 NuGet 版本控制规则, 如[包版本控制](#)中所述。NuGet 主要与 `semver 2` 兼容。

有关依赖项解析的信息, 请参阅[使用 PackageReference 的依赖项解析](#)。对于可能有助于更好地理解版本控制的较旧信息, 请参阅此系列博客文章。

- [第 1 部分: 解决 DLL 地狱](#)
- [第 2 部分: 核心算法](#)
- [第 3 部分: 通过绑定重定向实现统一](#)

添加 NuGet.Build.Tasks.Pack 包

如果将 MSBuild 与非 SDK 样式项目和 PackageReference 一起使用, 请将 NuGet.Build.Tasks.Pack 包添加到项目中。

1. 打开项目文件并在 `<PropertyGroup>` 元素后面添加以下内容:

```
<ItemGroup>
  <!-- ... -->
  <PackageReference Include="NuGet.Build.Tasks.Pack" Version="5.2.0"/>
  <!-- ... -->
</ItemGroup>
```

2. 打开开发人员命令提示符(在“搜索”框中, 键入“开发人员命令提示符”)。

用户通常习惯从“开始”菜单中启动“适用于 Visual Studio 的开发人员命令提示符”, 因为它将使用 MSBuild 的所有必需路径进行配置。

3. 切换到包含项目文件的文件夹, 然后键入以下命令以安装 NuGet.Build.Tasks.Pack 包。

```
# Uses the project file in the current folder by default
msbuild -t:restore
```

确保 MSBuild 输出指示生成已成功完成。

运行 msbuild -t:pack 命令

若要从项目中生成 NuGet 包(`.nupkg` 文件), 运行 `msbuild -t:pack` 命令, 它也会自动生成项目:

在 Visual Studio 开发人员命令提示处, 键入以下命令:

```
# Uses the project file in the current folder by default  
msbuild -t:pack
```

输出将显示 `.nupkg` 文件的路径。

```
Microsoft (R) Build Engine version 16.1.76+g14b0a930a7 for .NET Framework  
Copyright (C) Microsoft Corporation. All rights reserved.  
  
Build started 8/5/2019 3:09:15 PM.  
Project "C:\Users\username\source\repos\ClassLib_DotNetStandard\ClassLib_DotNetStandard.csproj" on node 1  
(pack target(s)).  
GenerateTargetFrameworkMonikerAttribute:  
Skipping target "GenerateTargetFrameworkMonikerAttribute" because all output files are up-to-date with  
respect to the input files.  
CoreCompile:  
...  
CopyFilesToOutputDirectory:  
Copying file from  
"C:\Users\username\source\repos\ClassLib_DotNetStandard\obj\Debug\netstandard2.0\ClassLib_DotNetStandard.dll"  
to "C:\Use  
rs\username\source\repos\ClassLib_DotNetStandard\bin\Debug\netstandard2.0\ClassLib_DotNetStandard.dll".  
ClassLib_DotNetStandard ->  
C:\Users\username\source\repos\ClassLib_DotNetStandard\bin\Debug\netstandard2.0\ClassLib_DotNetStandard.dll  
Copying file from  
"C:\Users\username\source\repos\ClassLib_DotNetStandard\obj\Debug\netstandard2.0\ClassLib_DotNetStandard.pdb"  
to "C:\Use  
rs\username\source\repos\ClassLib_DotNetStandard\bin\Debug\netstandard2.0\ClassLib_DotNetStandard.pdb".  
GenerateNuspec:  
Successfully created package  
'C:\Users\username\source\repos\ClassLib_DotNetStandard\bin\Debug\AppLogger.1.0.0.nupkg'.  
Done Building Project "C:\Users\username\source\repos\ClassLib_DotNetStandard\ClassLib_DotNetStandard.csproj"  
(pack target(s)).  
  
Build succeeded.  
0 Warning(s)  
0 Error(s)  
  
Time Elapsed 00:00:01.21
```

在生成期间自动生成包

若要在生成和还原项目时自动运行 `msbuild -t:pack`，请将以下行添加到 `<PropertyGroup>` 中的项目文件内：

```
<GeneratePackageOnBuild>true</GeneratePackageOnBuild>
```

在解决方案上运行 `msbuild -t:pack` 时，会打包解决方案中可打包的所有项目（属性设置为 `true`）。

NOTE

自动生成包时，打包时间会增加项目的生成时间。

测试包安装

发布包前，通常需要测试将包安装到项目的过程。测试确保所有文件一定在项目中正确的位置结束。

可以在 Visual Studio 中手动测试安装，或是使用常规[包安装步骤](#)在命令行上测试。

IMPORTANT

包是不可变的。如果更正了问题, 请更改包的内容并再次打包, 重新测试时, 仍将使用旧版本的包, 直到[清除全局包文件夹](#)。在测试每次生成时不使用唯一预发布标签的包时, 这尤为重要。

后续步骤

创建包([.nupkg](#) 文件)后, 可以将其发布到选择的库, 如[发布包](#)中所述。

你可能还希望扩展包的功能, 或者支持其他方案, 如以下主题所述:

- [作为 MSBuild 目标的 NuGet 包和还原](#)
- [包版本控制](#)
- [支持多个目标框架](#)
- [源和配置文件的转换](#)
- [本地化](#)
- [预发布版本](#)
- [设置包类型](#)
- [使用 COM 互操作程序集创建包](#)

最后, 有需要注意的其他包类型:

- [本机包](#)
- [符号包](#)

支持项目文件中的多个 .NET Framework 版本

2020/4/8 • [Edit Online](#)

首次创建项目时，建议创建 .NET Standard 类库，因为它提供了与最广泛使用项目的兼容性。使用 .NET Standard 可以默认向 .NET 库添加[跨平台支持](#)。但是，在某些情况下，可能还需要包含针对特定框架的代码。本文介绍如何针对 [SDK 样式](#) 的项目执行该操作。

对于 SDK 样式的项目，可以在项目文件中配置对多个目标框架 (TFM) 的支持，然后使用 `dotnet pack` 或 `msbuild /t:pack` 创建包。

NOTE

nuget.exe CLI 不支持打包 SDK 样式的项目，因此应只使用 `dotnet pack` 或 `msbuild /t:pack`。我们建议改为[包含所有属性](#)(项目文件的 `.nuspec` 文件中通常包含的所有属性)。要在非 SDK 样式的项目中以多个 .NET Framework 版本为目标，请参阅[支持多个 .NET Framework 版本](#)。

创建一个支持多个 .NET Framework 版本的项目

1. 在 Visual Studio 中或使用 `dotnet new classlib` 创建新的 .NET Standard 类库。

建议创建 .NET Standard 类库以获得最佳兼容性。

2. 编辑 `.csproj` 文件以支持目标框架。例如，更改

```
<TargetFramework>netstandard2.0</TargetFramework>
```

更改为：

```
<TargetFrameworks>netstandard2.0;net45</TargetFrameworks>
```

确保将 XML 元素从单数更改为复数(将“s”添加到开始和结束标记)。

3. 如果你有任何仅在一个 TFM 中工作的代码，则可以使用 `#if NET45` 或 `#if NETSTANDARD2_0` 分隔与 TFM 相关的代码。(有关详细信息，请参阅[如何实现多目标](#)。)例如，可以使用以下代码：

```
public string Platform {
    get {
        #if NET45
            return ".NET Framework"
        #elif NETSTANDARD2_0
            return ".NET Standard"
        #else
            #error This code block does not match csproj TargetFrameworks list
        #endif
    }
}
```

4. 将你需要的任何 NuGet 元数据添加到 `.csproj` 作为 MSBuild 属性。

有关可用包元数据和 MSBuild 属性名称的列表，请参阅[pack 目标](#)。另请参阅[控制依赖项资产](#)。

如果要将与生成相关的属性与 NuGet 元数据分开，可以使用不同的 `PropertyGroup`，或将 NuGet 属性放在另一个文件中，并使用 MSBuild 的 `Import` 指令将其包含在内。从 MSBuild 15.0. 开始，还支持 `Directory.Build.Props` 和 `Directory.Build.Targets`。

5. 现在, 使用 `dotnet pack` 和生成的 .nupkg 以 .NET Standard 2.0 和 .NET Framework 4.5 为目标。

以下是使用上述步骤和 .NET Core SDK 2.2 生成的 .csproj 文件。

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
  <TargetFrameworks>netstandard2.0;net45</TargetFrameworks>
  <Description>Sample project that targets multiple TFM</Description>
</PropertyGroup>

</Project>
```

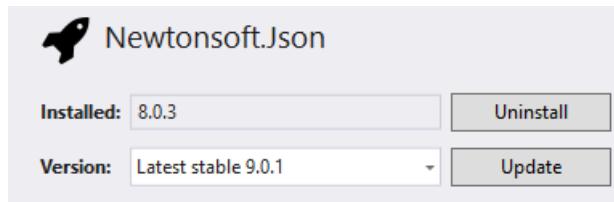
请参阅

- [如何指定目标框架](#)
- [跨平台定位](#)

生成预发行包

2020/4/8 • [Edit Online](#)

每当发布具有新版本号的更新包时, NuGet 将其视为所示的“最新稳定版本”, 以 Visual Studio 中的包管理器 UI 为例:



稳定版本是指被视为足够可靠, 可在生产中使用的版本。最新稳定版本也指作为包更新安装或在包还原期间安装的版本(受制于[重新安装和更新包](#)中所述的约束)。

为了支持软件的版本生命周期, NuGet 1.6 及更高版本允许分配预发行包, 其中的版本号包括语义化版本控制后缀, 如 `-alpha`、`-beta` 或 `-rc`。有关详细信息, 请参阅[包版本控制](#)。

可以使用以下方法之一来指定此类版本:

- 如果项目使用 `PackageReference` :在 `.csproj` 文件的 `PackageVersion` 元素中添加语义版本后缀:

```
<PropertyGroup>
  <PackageVersion>1.0.1-alpha</PackageVersion>
</PropertyGroup>
```

- 如果项目使用 `packages.config` 文件 :在 `.nuspec` 文件的 `version` 元素中添加语义版本后缀:

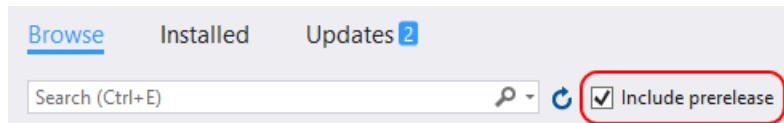
```
<version>1.0.1-alpha</version>
```

在准备好发布稳定版本后, 只需删除后缀, 此包便会优先于任何预发行版本。同样, 请参阅[包版本控制](#)。

安装和更新预发行包

默认情况下, NuGet 在处理包时不会包括预发行版本, 但按照下文所述方法可更改此行为:

- Visual Studio 中的包管理器 UI :在“管理 NuGet 包”UI 中, 选中“包括预发行版”框 :



设置或清除此框将刷新包管理器 UI 和可安装的可用版本的列表。

- 包管理器控制台 :将 `-IncludePrerelease` 开关与 `Find-Package`、`Get-Package`、`Install-Package`、`Sync-Package` 和 `Update-Package` 命令配合使用。请参阅 [PowerShell 参考](#)。
- NuGet CLI :将 `-prerelease` 开关与 `install`、`update`、`delete` 和 `mirror` 命令配合使用。请参阅 [NuGet CLI 参考](#)

语义化版本控制

[语义化版本控制](#)或 SemVer 约定介绍如何利用版本号中的字符串传达基础代码的含义。

在此约定中，每个版本由三部分组成，即 `Major.Minor.Patch`，其含义分别是：

- `Major`：重大更改
- `Minor`：新增功能，但可向后兼容
- `Patch`：仅可向后兼容的 bug 修复

然后，通过在修补程序编号后追加连字符和字符串表示预发行版本。从技术上说，可在连字符后使用任意字符串，NuGet 都会将包视为预发行版。然后，NuGet 在适用的 UI 中显示完整的版本号，供使用者自己解读其含义。

出于这种考虑，通常最好遵照如下所示的公认命名约定：

- `-alpha`：Alpha 版本，通常用于在制品和试验品
- `-beta`：Beta 版本，通常指可用于下一计划版本的功能完整的版本，但可能包含已知 bug。
- `-rc`：候选发布，通常可能为最终(稳定)版本，除非出现重大 bug。

NOTE

NuGet 4.3.0+ 支持[语义化版本控制 v2.0.0](#)，后者支持采用点表示法的预发布号，如 `1.0.1-build.23` 中所示。NuGet 4.3.0 之前的版本不支持点表示法。在 NuGet 的早期版本中，可以使用 `1.0.1-build23` 之类的格式，但这始终被视为预发布版本。

但是，无论使用何种后缀，NuGet 都会按照反向字母顺序向它们赋予优先级：

```
1.0.1
1.0.1-zzz
1.0.1-rc
1.0.1-open
1.0.1-beta.12
1.0.1-beta.5
1.0.1-beta
1.0.1-alpha.2
1.0.1-alpha
```

如上所示，没有任何后缀的版本始终优先于预发行版本。

semver2 不需要前导 0，但它们与旧版本架构一致。如果在可能使用两位数(或更多数字)的预发行版本标记中使用数字后缀，请使用前导零，如 beta.01 和 beta.05，以便确保数字增大时，可以正确地进行排序。此建议仅适用于旧版架构。

创建符号包 (.snupkg)

2020/4/8 • [Edit Online](#)

良好的调试体验依赖于调试符号的存在，因为它们提供了一些关键信息，例如已编译的代码与源代码之间的关联、局部变量的名称、堆栈跟踪等。你可以使用符号包 (.snupkg) 来分发这些符号，并改善 NuGet 包的调试体验。

必备条件

`nuget.exe v4.9.0 或更高版本` 或 `dotnet CLI v2.2.0 或更高版本`，它们实现了所需的 [NuGet 协议](#)。

创建符号包

如果使用 dotnet CLI 或 MSBuild，则除 .nupkg 文件外，还需要设置 `IncludeSymbols` 和 `SymbolPackageFormat` 属性以创建 .snupkg 文件。

- 要么将以下属性添加到 .csproj 文件：

```
<PropertyGroup>
  <IncludeSymbols>true</IncludeSymbols>
  <SymbolPackageFormat>snupkg</SymbolPackageFormat>
</PropertyGroup>
```

- 要么在命令行上指定这些属性：

```
dotnet pack MyPackage.csproj -p:IncludeSymbols=true -p:SymbolPackageFormat=snupkg
```

或

```
msbuild MyPackage.csproj /t:pack /p:IncludeSymbols=true /p:SymbolPackageFormat=snupkg
```

如果使用 NuGet.exe，除 .nupkg 文件外，可以使用以下命令创建一个 .snupkg 文件：

```
nuget pack MyPackage.nuspec -Symbols -SymbolPackageFormat snupkg
```

```
nuget pack MyPackage.csproj -Symbols -SymbolPackageFormat snupkg
```

`SymbolPackageFormat` 属性可以有下列两个值之一：`symbols.nupkg`（默认值）或 `snupkg`。如果未指定此属性，将创建旧的符号包。

NOTE

仍支持旧格式 `.symbols.nupkg`，但仅出于兼容性原因（请参阅[旧版符号包](#)）。NuGet.org 的符号服务器只接受新的符号包格式，即 `.snupkg`。

发布符号包

- 为方便起见，首先使用 NuGet 保存 API 密钥（请参阅[发布包](#)）。

```
nuget SetApiKey Your-API-Key
```

2. 将主包发布到 nuget.org 后, 按如下方式推送符号包。

```
nuget push MyPackage.snupkg
```

3. 还可以使用以下命令同时推送主包和符号包。当前文件夹中必须同时有 .nupkg 和 .snupkg 文件。

```
nuget push MyPackage.nupkg
```

NuGet 会将两个包发布到 nuget.org。`MyPackage.nupkg` 先发布, 随后 `MyPackage.snupkg` 发布。

NOTE

如果没有发布符号包, 请检查是否已将 NuGet.org 源配置为 <https://api.nuget.org/v3/index.json>。只有 [NuGet V3 API](#) 才支持符号包发布。

NuGet.org 符号服务器

NuGet.org 支持自己的符号服务器存储库, 只接受新的符号包格式 - `.snupkg`。包使用者可将 <https://symbols.nuget.org/download/symbols> 添加到 Visual Studio 中的符号源, 使用发布到 nuget.org 符号服务器的符号, 这允许在 Visual Studio 调试程序中单步执行包代码。有关该过程的详细信息, 请参阅[在 Visual Studio 调试程序中指定符号 \(.pdb\) 和源文件](#)。

NuGet.org 符号包约束

NuGet.org 对符号包具有以下约束:

- 符号包中仅允许使用以下文件扩展名: `.pdb`、`.nuspec`、`.xml`、`.psmdcp`、`.rels`、`.p7s`
- NuGet.org 符号服务器目前仅支持托管的[可移植 PDB](#)。
- 需要使用 Visual Studio 15.9 或更高版本中的编译器构建 PDB 及其关联的 nupkg DLL(请参阅 [PDB 加密哈希](#))

如果未满足这些约束, 则发布到 NuGet.org 的符号包将无法通过验证。

符号包验证和编制索引

发布到 [NuGet.org](#) 的符号包会接受多项验证, 包括恶意软件扫描。如果包未通过验证检查, 则其包详细信息页将显示错误消息。此外, 包的所有者还将收到一封电子邮件, 其中包含有关如何解决已识别问题的说明。

当符号包通过所有验证后, NuGet.org 的符号服务器将为其中的符号编制索引, 并且这些符号将可供使用。

包验证和编制索引所需的时间通常不超过 15 分钟。如果发布包所用时间超出预期, 请访问 [status.nuget.org](#) 检查 NuGet.org 是否遇到任何中断。如果所有系统均正常运行, 但一个小时之内还未成功发布包, 请登录 [nuget.org](#) 并使用包详细信息页面上的“联系支持人员”链接与我们联系。

符号包结构

符号包 (.snupkg) 具有以下特征:

1. .snupkg 与相应的 NuGet 包 (.nupkg) 具有相同的 ID 和版本。
2. .snupkg 具有与任何 DLL 或 EXE 文件的相应 .nupkg 相同的文件夹结构, 区别在于其相应的 PDB 将包含在同一文件夹层次结构中, 而不是 DLL/EXE 中。扩展名不是 PDB 的文件和文件夹将被排除在 snupkg 之外。

3. 符号包的 .nuspec 文件具有 `SymbolsPackage` 包类型：

```
<packageTypes>
  <packageType name="SymbolsPackage"/>
</packageTypes>
```

4. 如果创建者决定使用自定义 nuspec 来构建其 nupkg 和 snupkg，则 snupkg 应该具有 2 中详细描述的同一文件夹层次结构和文件)。
5. 将从 snupkg 的 nuspec 中排除以下字段：`authors`、`owners`、`requireLicenseAcceptance`、`license type`、`licenseUrl` 和 `icon`。
6. 不要使用 `<license>` 元素。`.snupkg` 与对应的 `.nupk` 位于同一个许可证中。

另请参阅

考虑使用源链接来启用 .NET 程序集的源代码调试。有关详细信息，请参阅[源链接指南](#)。

有关符号包的更多信息，请参阅[NuGet 包调试与符号改进设计规范](#)。

支持多个 .NET 版本

2020/4/8 • [Edit Online](#)

许多库以 .NET Framework 的特定版本为目标。例如，你可能拥有特定于 UWP 的一个库版本，而拥有的另一版本利用 .NET Framework 4.6 中的功能。为了适应这种情况，NuGet 支持将同一个库的多个版本放在单个包中。

本文介绍 NuGet 包的布局，但不考虑包或程序集的生成方式（也就是说，无论是使用多个非 SDK 样式的 .csproj 文件、自定义 .nuspec 文件，还是单个多目标 SDK 样式的 .csproj，布局都相同）。对于 SDK 样式的项目，NuGet pack 目标知道如何设计包的布局，并自动将程序集放在正确的 lib 文件夹中，并为每个目标框架 (TFM) 创建依赖项组。有关详细说明，请参阅[在项目文件中支持多个 .NET Framework 版本](#)。

使用[创建包](#)中介绍的基于约定的工作目录方法时，必须按照本文所述手动设计包的布局。对于 SDK 样式的项目，建议使用自动方法，但也可以选择按照本文所述手动设计包的布局。

框架版本的文件夹结构

生成仅包含一个库版本或面向多个框架的包时，始终根据以下约定使用区分大小写的不同框架名称在 lib 下创建子文件夹：

```
lib\{framework name}\{version}]
```

有关支持的名称的完整列表，请参阅[目标框架引用](#)。

所具有的库版本必须特定于某框架，且不应直接放置在根 lib 文件夹中。（此功能仅受 packages.config 支持。）此操作会导致库与任何目标框架兼容，并允许在任何位置上进行安装，很可能造成意外的运行时错误。已弃用在根文件夹（如 lib\abc.dll）或子文件夹（如 lib\abc\abc.dll）中添加程序集的操作，且在使用 PackagesReference 格式时忽略了此操作。

例如，以下文件夹结构支持特定于框架的程序集的四种版本：

```
\lib
  \net46
    \MyAssembly.dll
  \net461
    \MyAssembly.dll
  \uap
    \MyAssembly.dll
  \netcore
    \MyAssembly.dll
```

若要在生成包时轻松包括所有这些文件，请在 ** 的 <files> 部分中使用递归 .nuspec 通配符：

```
<files>
  <file src="lib\**" target="lib/{framework name}\{version}" />
</files>
```

特定于体系结构的文件夹

如果具有特定于体系结构的程序集，即面向 ARM、x86 和 x64 的单独程序集，必须将它们放置在名为 runtimes 或 {platform}-{architecture}\lib\{framework} 的子文件夹内名为 {platform}-{architecture}\native 的文件夹中。例如，以下文件夹结构可以容纳面向 Windows 10 的本机和托管 DLL 以及 uap10.0 框架：

```
\runtimes
  \win10-arm
    \native
    \lib\uap10.0
  \win10-x86
    \native
    \lib\uap10.0
  \win10-x64
    \native
    \lib\uap10.0
```

这些程序集将仅在运行时可用，因此，如果你也想要提供相应的编译时程序集，则在 `AnyCPU` 文件夹中设置 `/ref/{tfm}` 程序集。

请注意，NuGet 始终从一个文件夹选取这些编译或运行时资产，因此，如果在 `/ref` 中存在某些兼容资产，则将忽略 `/lib` 以添加编译时程序集。同样，如果在 `/runtimes` 中有某些兼容资产，则也将为运行时忽略 `/lib`。

有关在 [清单中引用这些文件的示例](#)，请参阅创建 UWP 包 `.nuspec`。

此外，请参阅[使用 NuGet 打包 Windows 存储应用组件](#)

将程序集版本与项目中的目标框架匹配

NuGet 在安装具有多个程序集版本的包时，会尝试将程序集的框架名称与项目的目标框架匹配。

如果未找到匹配项，NuGet 将复制小于或等于项目的目标框架的最高版本的程序集（如果可用）。如果未找到任何兼容的程序集，NuGet 将返回相应的错误消息。

例如，假设包中具有以下文件夹结构：

```
\lib
  \net45
    \MyAssembly.dll
  \net461
    \MyAssembly.dll
```

当在面向 .NET Framework 4.6 的项目中安装此包时，NuGet 将在 `net45` 文件夹中安装程序集，因为它是小于或等于 4.6 的最高可用版本。

但是，如果项目面向 .NET Framework 4.6.1，NuGet 将在 `net461` 文件夹中安装程序集。

如果项目面向 .NET Framework 4.0 和更早版本，NuGet 会发出相应的错误消息，因为找不到兼容的程序集。

通过框架版本对程序集进行分组

NuGet 仅从包中的单个库文件夹中复制程序集。例如，假设包具有以下文件夹结构：

```
\lib
  \net40
    \MyAssembly.dll (v1.0)
    \MyAssembly.Core.dll (v1.0)
  \net45
    \MyAssembly.dll (v2.0)
```

当在面向 .NET Framework 4.5 的项目中安装包时，将仅安装 `MyAssembly.dll` (v2.0) 程序集。不会安装 `MyAssembly.Core.dll` (v1.0)，因为它未在 `net45` 文件夹中列出。NuGet 执行此操作的原因是 `MyAssembly.Core.dll` 可能已合并到 `MyAssembly.dll` 的 2.0 版本中。

如果要为 .NET Framework 4.5 安装 `MyAssembly.Core.dll`，请在 `net45` 文件夹中放置副本。

通过框架配置文件对程序集进行分组

NuGet 还通过向文件夹末尾追加短划线和配置文件名称，支持以特定的框架配置文件为目标。

```
lib\{framework name}-{profile}
```

以下是支持的配置文件：

- `client` : 客户端配置文件
- `full` : 完整配置文件
- `wp` : Windows Phone
- `cf` : Compact Framework

声明依赖项(高级)

打程序包项目文件时，NuGet 尝试从项目自动生成依赖项。此部分中的信息介绍了如何使用 `.nuspec` 文件声明依赖项，通常仅高级方案需要使用此信息。

(版本 2.0+) 可以使用 `<group>` 元素中的 `<dependency>` 在目标项目的对应 `.nuspec` 中声明程序包依赖项。

`<group>`<dependencies>` 有关详细信息，请参阅[依赖项元素](#)。

每个组都有一个名为 `targetFramework` 的特性，并包含零个或多个 `<dependency>` 元素。当目标框架与项目的框架配置文件兼容时，将会一起安装这些依赖项。有关确切的框架标识符，请参阅[目标框架](#)。

建议每个目标框架名字对象 (TFM) 将一个组用于 lib/ 和 ref/ 文件夹中的文件。

以下示例显示了 `<group>` 元素的不同变体：

```
<dependencies>

<group targetFramework="net472">
    <dependency id="jQuery" version="1.10.2" />
    <dependency id="WebActivatorEx" version="2.2.0" />
</group>

<group targetFramework="net20">
</group>

</dependencies>
```

确定要使用的 NuGet 目标

当打包面向可移植类库的库时，不容易确定应在文件夹名称和 `.nuspec` 文件中使用的 NuGet 目标，尤其当仅面向 PCL 的子集时。以下外部资源有助于解决此问题：

- [.NET 中的框架配置文件](#) (stephenclearly.com)
- [可移植类库的配置文件](#) (plnkr.co)：枚举 PCL 配置文件及其等效 NuGet 目标的表
- [可移植类库的配置文件工具](#) (github.com)：用于确定系统上可用的 PCL 配置文件的命令行工具

内容文件和 PowerShell 脚本

WARNING

仅可通过 `packages.config` 格式使用可变的内容文件和脚本执行;这些内容在使用所有其他格式时已弃用,且不应用于任何新的包。

对于 `packages.config`, 可以使用 `content` 和 `tools` 文件夹中的相同文件夹约定, 通过目标框架对内容文件和 PowerShell 脚本进行分组。例如:

```
\content
  \net46
    \MyContent.txt
  \net461
    \MyContent461.txt
  \uap
    \MyUWPContent.html
  \netcore
  \tools
    init.ps1
  \net46
    install.ps1
    uninstall.ps1
  \uap
    install.ps1
    uninstall.ps1
```

如果框架文件夹保留为空, NuGet 不会添加程序集引用或内容文件, 也不会运行该框架的 PowerShell 脚本。

NOTE

因为 `init.ps1` 在解决方案级别执行, 且它不依赖于项目, 所以必须将其直接放置在 `tools` 文件夹下。如果将其放置在框架文件夹下, 它会被忽略。

转换源代码和配置文件

2020/4/8 • [Edit Online](#)

安装包时，“源代码转化”会对包的 `content` 或 `contentFiles` 文件夹(对于为 `PackageReference` 使用 `packages.config` 和 `contentFiles` 的用户，则为 `content`)中的文件应用单向令牌替换，其中令牌表示 Visual Studio 项目属性。这样就可以将文件插入到项目的命名空间中，或者自定义通常转到 ASP.NET 项目的 `global.asax` 中的代码。

通过“配置文件转换”可以修改目标项目中已存在的文件，例如 `web.config` 和 `app.config`。例如，包可能需要将某个项添加到配置文件中的 `modules` 部分。此转换通过将特殊文件包含在描述要添加到配置文件的部分的包中完成。卸载包时，会接着反转相同的更改，使其变为双向转换。

指定源代码转换

1. 需要从包插入到项目的文件必须位于包的 `content` 和 `contentFiles` 文件夹中。例如，如果希望将一个名为 `ContosoData.cs` 的文件安装在目标项目的 `Models` 文件夹中，则它必须位于包的 `content\Models` 和 `contentFiles\{lang}\{tfm}\Models` 文件夹中。
2. 若要指示 NuGet 在安装时应用令牌替换，请将 `.pp` 追加到源代码文件名。安装后，文件不会具有 `.pp` 扩展名。

例如，若要在 `ContosoData.cs` 中转换，请在包 `ContosoData.cs.pp` 中命名文件。安装后，它将以 `ContosoData.cs` 形式出现。

3. 在源代码文件中，使用 `$token$` 形式的不区分大小写令牌指示 NuGet 应使用项目属性替换的值：

```
namespace $rootnamespace$.Models
{
    public struct CategoryInfo
    {
        public string categoryid;
        public string description;
        public string htmlUrl;
        public string rssUrl;
        public string title;
    }
}
```

安装后，NuGet 将 `$rootnamespace$` 替换为 `Fabrikam`(假设目标项目的根命名空间为 `Fabrikam`)。

`$rootnamespace$` 令牌是最常用的项目属性；其他所有项目属性均列在[项目属性](#)中。请注意，有些属性可能特定于某一项目类型。

指定配置文件转换

如以下部分所述，可以通过两种方法完成配置文件转换：

- 在包的 `content` 文件夹中包含 `app.config.transform` 和 `web.config.transform` 文件，其中 `.transform` 扩展名告知 NuGet 这些文件包含在安装包时要与现有配置文件合并的 XML。卸载包时，该相同 XML 被删除。
- 在包的 `content` 文件夹中包含 `app.config.install.xdt` 和 `web.config.install.xdt` 文件，并使用 [XDT 语法](#) 描述所需更改。使用此选项还可以包含 `.uninstall.xdt` 文件，在从项目删除包时撤销更改。

NOTE

转换不适用于在 Visual Studio 中作为链接引用的 `.config` 文件。

使用 XDT 的优点是，它不是简单地合并两个静态文件，而是提供一种语法来操纵 XML DOM 的结构，这种语法使用元素和特性，支持完整的 XPath 路径。然后 XDT 可以添加、更新或删除元素、将新元素放在特定位置，或者替换/删除元素（包括子节点）。这使得创建卸载转换非常简单，卸载转换指退出包安装期间所完成的全部转换。

XML 转换

包的 `content` 文件夹中的 `app.config.transform` 和 `web.config.transform` 仅包含要合并到项目现有 `app.config` 和 `web.config` 文件中的元素。

例如，假设项目最初在 `web.config` 中包含以下内容：

```
<configuration>
  <system.webServer>
    <modules>
      <add name="ContosoUtilities" type="Contoso.Utilities" />
    </modules>
  </system.webServer>
</configuration>
```

若要在包安装期间将 `MyNuModule` 元素添加到 `modules` 部分，请在包的 `content` 文件夹中创建 `web.config.transform` 文件，如下所示：

```
<configuration>
  <system.webServer>
    <modules>
      <add name="MyNuModule" type="Sample.MyNuModule" />
    </modules>
  </system.webServer>
</configuration>
```

NuGet 安装包后，`web.config` 将显示为：

```
<configuration>
  <system.webServer>
    <modules>
      <add name="ContosoUtilities" type="Contoso.Utilities" />
      <add name="MyNuModule" type="Sample.MyNuModule" />
    </modules>
  </system.webServer>
</configuration>
```

注意，NuGet 未替换 `modules` 部分，只是通过仅添加新元素和特性合并了新条目。NuGet 不会更改任何现有元素或特性。

卸载包时，NuGet 会再次检查 `.transform` 文件，并从相应的 `.config` 文件中删除它包含的元素。注意，此过程不会影响包安装后修改的 `.config` 文件中的任何行。

一个更广泛的示例是[适用于 ASP.NET 的错误日志记录模块和处理程序 \(ELMAH\)](#) 包将多个条目添加到 `web.config` 中，这些条目在卸载时会再次删除。

若要检查其 `web.config.transform` 文件，请从上述链接下载 ELMAH 包，将包扩展名从 `.nupkg` 更改为 `.zip`，然后打开该 ZIP 文件中的 `content\web.config.transform`。

若要查看安装和卸载包的影响，请在 Visual Studio 中创建新的 ASP.NET 项目（模板位于“新建项目”对话框的“Visual

C#" > "Web" 下), 并选择一个空的 ASP.NET 应用程序。打开 `web.config` 查看其初始状态。然后右键单击项目, 选择“管理 NuGet 包”, 在 nuget.org 上浏览 ELMAH 并安装最新版本。注意对 `web.config` 进行的所有更改。现在卸载包, 你会看见 `web.config` 还原到其之前的状态。

XDT 转换

NOTE

如从 `packages.config` 迁移到 `PackageReference` 文档的兼容性问题一节所述, 下述 XDT 转换仅受 `packages.config` 支持。如果将以下文件添加到包中, 则使用者在使用具有 `PackageReference` 的包时无需应用转换(参考[本示例](#), 了解如何让 XDT 转换适用于 `PackageReference`)。

可以使用 [XDT 语法](#)修改配置文件。此外, 还可通过在 `$` 分隔符内包含属性名称(不区分大小写), 让 NuGet 将令牌替换为[项目属性](#)。

例如, 以下 `app.config.install.xdt` 文件会将 `appSettings` 元素插入到包含项目 `FullPath`、`FileName` 和 `ActiveConfigurationSettings` 值的 `app.config` 中:

```
<?xml version="1.0"?>
<configuration xmlns:xdt="http://schemas.microsoft.com/XML-Document-Transform">
    <appSettings xdt:Transform="Insert">
        <add key="FullPath" value="$FullPath$" />
        <add key="FileName" value="$filename$" />
        <add key="ActiveConfigurationSettings" value="$ActiveConfigurationSettings$" />
    </appSettings>
</configuration>
```

有关另一个示例, 假设项目最初在 `web.config` 中包含以下内容:

```
<configuration>
    <system.webServer>
        <modules>
            <add name="ContosoUtilities" type="Contoso.Utilities" />
        </modules>
    </system.webServer>
</configuration>
```

若要在安装包期间将 `MyNuModule` 元素添加到 `modules` 部分, 包的 `web.config.install.xdt` 应包含以下内容:

```
<?xml version="1.0"?>
<configuration xmlns:xdt="http://schemas.microsoft.com/XML-Document-Transform">
    <system.webServer>
        <modules>
            <add name="MyNuModule" type="Sample.MyNuModule" xdt:Transform="Insert" />
        </modules>
    </system.webServer>
</configuration>
```

安装包以后, `web.config` 将如下所示:

```
<configuration>
  <system.webServer>
    <modules>
      <add name="ContosoUtilities" type="Contoso.Utilities" />
      <add name="MyNuModule" type="Sample.MyNuModule" />
    </modules>
  </system.webServer>
</configuration>
```

若要在卸载包期间仅删除 `MyNuModule` 元素，`web.config.uninstall.xdt` 文件应包含以下内容：

```
<?xml version="1.0"?>
<configuration xmlns:xdt="http://schemas.microsoft.com/XML-Document-Transform">
  <system.webServer>
    <modules>
      <add name="MyNuModule" xdt:Transform="Remove" xdt:Locator="Match(name)" />
    </modules>
  </system.webServer>
</configuration>
```

选择项目引用的程序集

2020/4/8 • [Edit Online](#)

显式程序集引用允许将程序集的子集用于 IntelliSense 和编译，而所有程序集在运行时都可用。`PackageReference` 和 `packages.config` 的工作方式不同，因此包创建者需要注意创建与两种项目类型兼容的包。

NOTE

显式程序集引用与 .NET 程序集相关。它不是用于分发托管程序集调用的本机程序集的方法。

PackageReference 支持

当一个项目使用一个带有 `PackageReference` 的包，并且该包包含一个 `ref\<tfm>\` 目录时，NuGet 会将这些组件分类为编译时资产，而 `lib\<tfm>\` 程序集则归类为运行时资产。`ref\<tfm>\` 中的程序集不在运行时使用。这意味着 `ref\<tfm>\` 中的任何程序集都必须在 `lib\<tfm>\` 或相关的 `runtime\` 目录中具有匹配的程序集，否则可能会发生运行时错误。由于 `ref\<tfm>\` 中的程序集未在运行时使用，因此可以使用“[仅元数据”专用程序集](#)来减小包的大小。

IMPORTANT

如果包中包含 nuspec `<references>` 元素（由 `packages.config` 使用，请参见下文）并且不包含 `ref\<tfm>\` 中的程序集，NuGet 会将 nuspec `<references>` 元素中列出的程序集作为编译和运行时资产进行颁发。这意味着当引用的程序集需要加载 `lib\<tfm>\` 目录中的任何其他程序集时，将存在运行时异常。

NOTE

如果包包含 `runtime\` 目录，则 NuGet 可能不会使用 `lib\` 目录中的资产。

packages.config 支持

使用 `packages.config` 管理 NuGet 包的项目通常会向 `lib\<tfm>\` 目录中的所有程序集添加引用。添加 `ref\` 目录以支持 `PackageReference`，因此在使用 `packages.config` 时不考虑。要使用 `packages.config` 以显式方式设置项目引用的程序集，包必须使用 [nuspec 文件中的 <references> 元素](#)。例如：

```
<references>
  <group targetFramework="net45">
    <reference file="MyLibrary.dll" />
  </group>
</references>
```

NOTE

`packages.config` 项目使用名为 [ResolveAssemblyReference](#) 的进程将程序集复制到 `bin\<configuration>\` 输出目录。复制项目的程序集，然后构建系统查看引用程序集的程序集清单，然后复制这些程序集并递归重复所有程序集。这意味着，如果 `lib\<tfm>\` 目录中的任何程序集不在任何其他程序集的清单中作为依赖项列出（如果在运行时使用 `Assembly.Load`、MEF 或其他依赖项注入框架加载程序集）那么即使在 `bin\<configuration>\` 中，也不能将其复制到项目的 `bin\<tfm>\` 输出目录中。

示例

我的包将包含三个程序集 `MyLib.dll`、`MyHelpers.dll` 和 `MyUtilities.dll`，它们以 .NET Framework 4.7.2 为目标。`MyUtilities.dll` 包含仅供其他两个程序集使用的类，因此我不希望在 IntelliSense 中或在编译时将这些类用于使用我的包的项目。我的 `nuspec` 文件需要包含以下 XML 元素：

```
<references>
  <group targetFramework="net472">
    <reference file="MyLib.dll" />
    <reference file="MyHelpers.dll" />
  </group>
</references>
```

并且包中的文件将为：

```
lib\net472\MyLib.dll
lib\net472\MyHelpers.dll
lib\net472\MyUtilities.dll
ref\net472\MyLib.dll
ref\net472\MyHelpers.dll
```

设置 NuGet 包类型

2020/4/8 • [Edit Online](#)

通过 NuGet 3.5+ 可以使用特定的包类型标记包以指示其预期用途。未标记类型的包(包含使用更早版本的 NuGet 创建的所有包)默认为 `Dependency` 类型。

- `Dependency` 类型包将生成或运行时资产添加到库和应用程序，并且可以在任何项目类型中安装(假设它们相互兼容)。
- `DotnetTool` 类型包是 [dotnet CLI](#) 的扩展，并且从命令行中调用。该包仅在 .NET Core 项目中安装并且不影响还原操作。[.NET Core 扩展性](#) 文档中提供更多有关这些项目扩展的详细信息。
- `Template` 类型包提供[自定义模板](#)，这些模板可以用来创建文件或项目，例如应用、服务、工具或类库。
- 自定义类型包使用与包 ID 遵守相同格式规则的任意类型标识符。但是，任何不是 `Dependency` 和 `DotnetTool` 的类型不会被 Visual Studio 中的 NuGet 包管理器识别。

包类型在 `.nuspec` 文件中设置。后向兼容最好不显式设置 `类型`，而是依赖 NuGet 在没有指定类型时假设此类型 `Dependency`。

- `.nuspec` : 指示 `packageTypes\packageType` 元素的 `<metadata>` 节点中的包类型：

```
<?xml version="1.0" encoding="utf-8"?>
<package xmlns="http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd">
    <metadata>
        <!-- ... -->
    <packageTypes>
        <packageType name="DotnetTool" />
    </packageTypes>
    </metadata>
</package>
```

创建本地化 NuGet 包

2020/4/13 • [Edit Online](#)

有两种方法可创建库的本地化版本：

1. 将所有本地化资源程序集包含在一个包中。
2. 遵循一组严格的约定，创建单独的本地化附属包。

两种方法各有优缺点，如以下部分中所述。

本地化资源程序集位于一个包中

将本地化资源程序集包含在一个包中通常是最简单的方法。若要执行此操作，请在 `lib` 中创建文件夹用于支持的语言，而不是包默认语言（假定为 `en-us`）。在这些文件夹中，可以放置资源程序集和本地化 IntelliSense XML 文件。

例如，以下文件夹结构支持德语 (`de`)、意大利语 (`it`)、日语 (`ja`)、俄语 (`ru`)、简体中文 (`zh-Hans`) 和繁体中文 (`zh-Hant`)：

```
lib
└── net40
    ├── Contoso.Utilities.dll
    └── Contoso.Utilities.xml

    ├── de
    │   ├── Contoso.Utilities.resources.dll
    │   └── Contoso.Utilities.xml

    ├── it
    │   ├── Contoso.Utilities.resources.dll
    │   └── Contoso.Utilities.xml

    ├── ja
    │   ├── Contoso.Utilities.resources.dll
    │   └── Contoso.Utilities.xml

    ├── ru
    │   ├── Contoso.Utilities.resources.dll
    │   └── Contoso.Utilities.xml

    ├── zh-Hans
    │   ├── Contoso.Utilities.resources.dll
    │   └── Contoso.Utilities.xml

    └── zh-Hant
        ├── Contoso.Utilities.resources.dll
        └── Contoso.Utilities.xml
```

可以看到，这些语言全都列在 `net40` 目标框架文件夹下。如果 [支持多个框架](#)，则 `lib` 下将有每个变体的文件夹。

这些文件夹准备就绪后，即可在 `.nuspec` 中引用所有文件：

```
<?xml version="1.0"?>
<package>
  <metadata>...
  </metadata>
  <files>
    <file src="lib\***" target="lib" />
  </files>
</package>
```

使用此方法的一个包示例为 [Microsoft.Data.OData 5.4.0](#)。

优点和缺点(本地化资源程序集)

将所有语言捆绑到一个包中存在几个缺点：

- 共享元数据**: 由于 NuGet 包只可包含一个 `.nuspec` 文件，因此仅可为一种语言提供元数据。也就是说，NuGet 当前不支持本地化元数据。
- 包大小**: 根据支持的语言数量，库可能变得相当大，这会减慢包的安装和还原速度。
- 同时发布**: 将本地化文件捆绑到一个包中需要同时发布该包中的所有资产，而不是能够单独发布每个本地化。此外，对任何一个本地化的任何更新都需要整个包的新版本。

然而，它也存在几个优点：

- 简单**: 包的使用者安装一次即可获得支持的所有语言，而不必单独安装每种语言。在 [nuget.org](#) 上查找单个包也更加轻松。
- 耦合版本**: 由于所有资源程序集与主程序集都位于相同的包中，因此它们共享相同的版本号，且不存在错误分离的风险。

本地化附属包

类似于 .NET Framework 支持附属程序集的方式，此方法将本地化资源和 IntelliSense XML 文件分离为附属包。

若要执行此操作，主包会使用命名约定 `{identifier}.{version}.nupkg` 并包含默认语言(如 en-US)的程序集。例如，`ContosoUtilities.1.0.0.nupkg` 将包含以下结构：

```
lib
└── net40
    ContosoUtilities.dll
    ContosoUtilities.xml
```

附属程序集然后会使用命名约定 `{identifier}.{language}.{version}.nupkg`，如 `ContosoUtilities.de.1.0.0.nupkg`。标识符必须与主包完全匹配。

由于这是单独的包，因此它本身具有包含本地化元数据的 `.nuspec` 文件。请注意，`.nuspec` 中的语言必须与文件名中所使用的语言匹配。

附属程序集还必须使用 [] 版本表示法，将主包的确切版本声明为依赖项(请参阅[包版本控制](#))。例如，`ContosoUtilities.de.1.0.0.nupkg` 必须使用 `[1.0.0]` 表示法声明对 `ContosoUtilities.1.0.0.nupkg` 的依赖项。当然，附属包的版本号可以与主包不同。

然后，附属包结构必须将资源程序集和 XML IntelliSense 包含在与包文件名中的 `{language}` 匹配的子文件夹中：

```
lib
└── net40
    └── de
        ContosoUtilities.resources.dll
        ContosoUtilities.xml
```

注意：除非需要特定的子区域性（如 `ja-JP`），否则请始终使用更高级别的语言标识符，如 `ja`。

在附属程序集中，NuGet 仅会识别文件名与 `{language}` 匹配的文件夹中的这些文件。将忽略所有其他成员。

如果满足所有这些约定，NuGet 会将包识别为附属包并将本地化文件安装到主包的 `lib` 文件夹中，就好像它们最初就进行了捆绑。卸载附属程序包将从该相同文件夹中删除其文件。

你将以相同方式为支持的每种语言创建其他附属程序集。例如，检查一组 ASP.NET MVC 包：

- [Microsoft.AspNet.Mvc](#)（主要为英语）
- [Microsoft.AspNet.Mvc.de](#)（德语）
- [Microsoft.AspNet.Mvc.ja](#)（日语）
- [Microsoft.AspNet.Mvc.zh-Hans](#)（简体中文）
- [Microsoft.AspNet.Mvc.zh-Hant](#)（繁体中文）

所需约定摘要

- 主包必须命名为 `{identifier}.{version}.nupkg`
- 附属包必须命名为 `{identifier}.{language}.{version}.nupkg`
- 附属包的 `.nuspec` 必须指定其语言以与文件名匹配。
- 附属包必须在其 `.nuspec` 文件中使用 `[]` 表示法，声明对主包确切版本的依赖项。不支持范围。
- 附属包必须将文件放置于文件名与 `{language}` 完全匹配的 `lib\[{framework}\]\{language}` 文件夹中。

优点和缺点（附属包）

使用附属包有几个优点：

1. **包大小**：最大限度地减少主包的总内存占用量，并且使用者仅承担要使用的语言的费用。
2. **单独的元数据**：每个附属包都有自己的 `.nuspec` 文件，因此也拥有自己的本地化元数据。借此，一些使用者可通过使用本地化术语搜索 nuget.org，更加轻松地查找包。
3. **分离发布**：附属程序集可随着时间推移进行发布，而不是一次完成，从而分散了本地化工作量。

然而，附属包也有其自身的一些缺点：

1. **混乱**：许多包（而不是一个包）可能会导致 nuget.org 上的搜索结果混乱以及 Visual Studio 项目中的引用列表变长。
2. **严格的约定**：附属包必须完全遵循约定，否则将不会正确选取本地化版本。
3. **版本控制**：每个附属包必须对主包具有确切版本的依赖项。这意味着更新主包可能也需要更新所有附属程序包，即使未更改资源。

创建 UWP 包

2020/4/8 • [Edit Online](#)

通用 Windows 平台 (UWP) 为运行 Windows 10 的每台设备提供了一个通用应用平台。在此模型中, UWP 应用可以调用所有设备通用的 WinRT API, 以及特定于运行该应用的设备系列的 API(包括 Win32 和 .NET)。

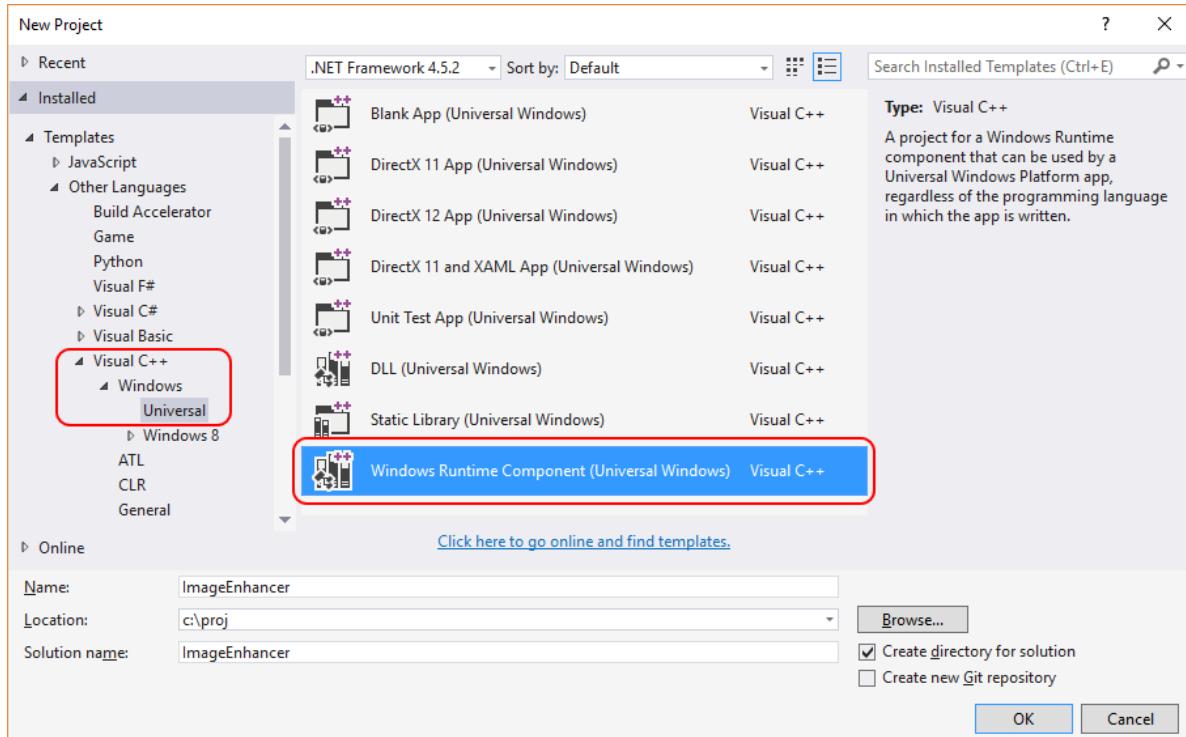
在本演练中, 将创建一个具有本机 UWP 组件(包括 XAML 控件)的 NuGet 包, 以便在托管项目和本机项目中使用。

先决条件

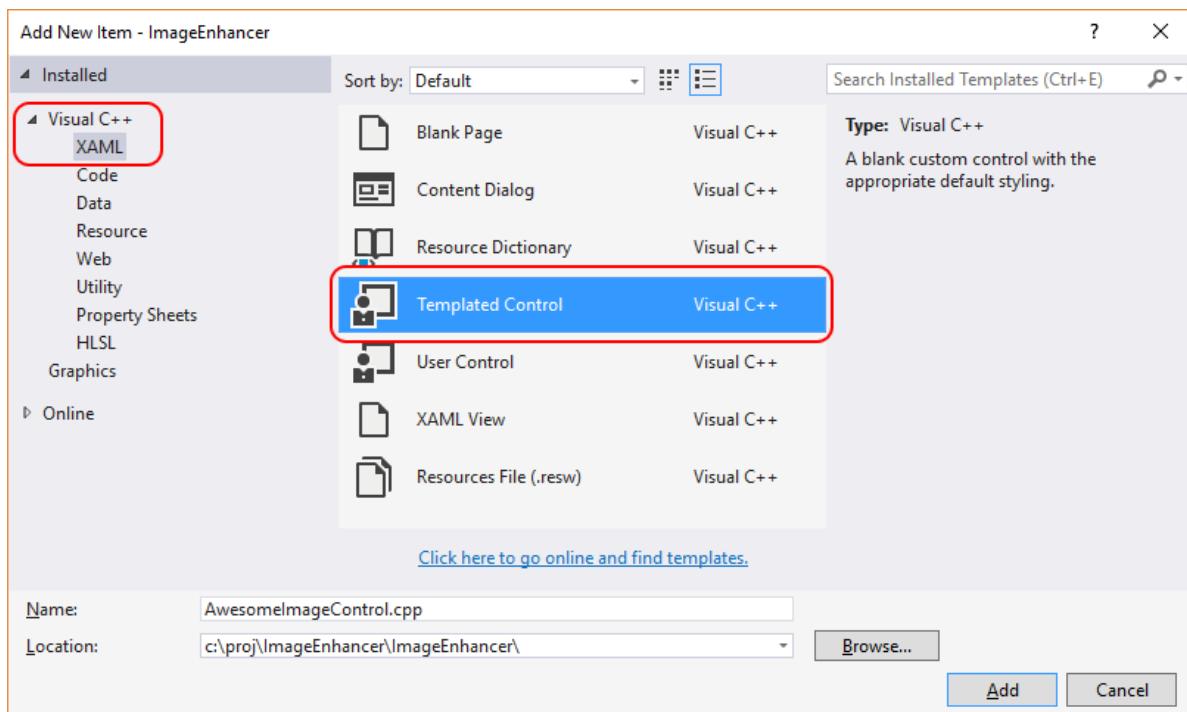
1. Visual Studio 2017 或 Visual Studio 2015。可以从 [visualstudio.com](#) 免费安装 2017 Community 版;也可以使用 Professional 和 Enterprise 版。
2. NuGet CLI。从 [nuget.org/downloads](#) 下载 `nuget.exe` 的最新版本, 将其保存到选择的位置(`.exe` 是直接下载的)。然后将该位置添加到 PATH 环境变量(如果尚未添加)。

创建 UWP Windows 运行时组件

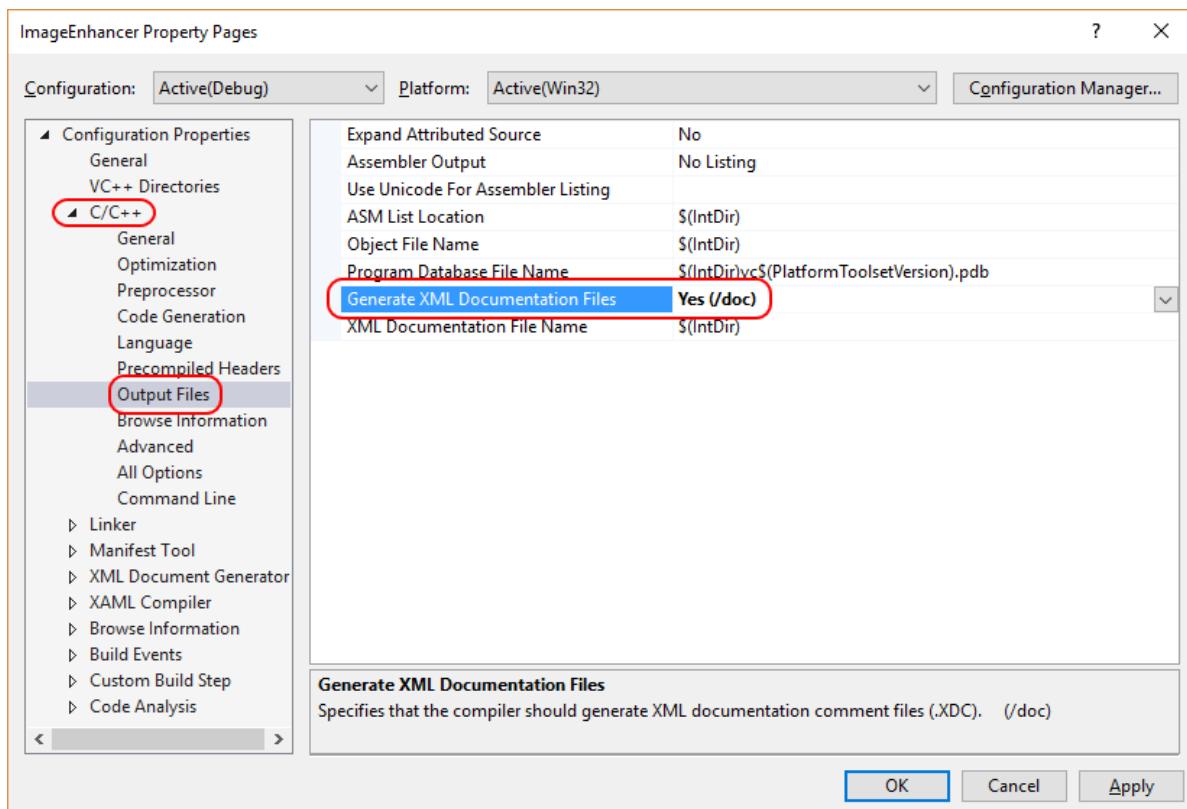
1. 在 Visual Studio 中, 选择“文件”>“新建”>“项目”, 展开“Visual C++”>“Windows”>“Universal”节点, 选择“Windows 运行时组件(通用 Windows)”, 将名称更改为“ImageEnhancer”, 然后单击“确定”。出现提示时, 接受“目标版本”和“最低版本”的默认值。



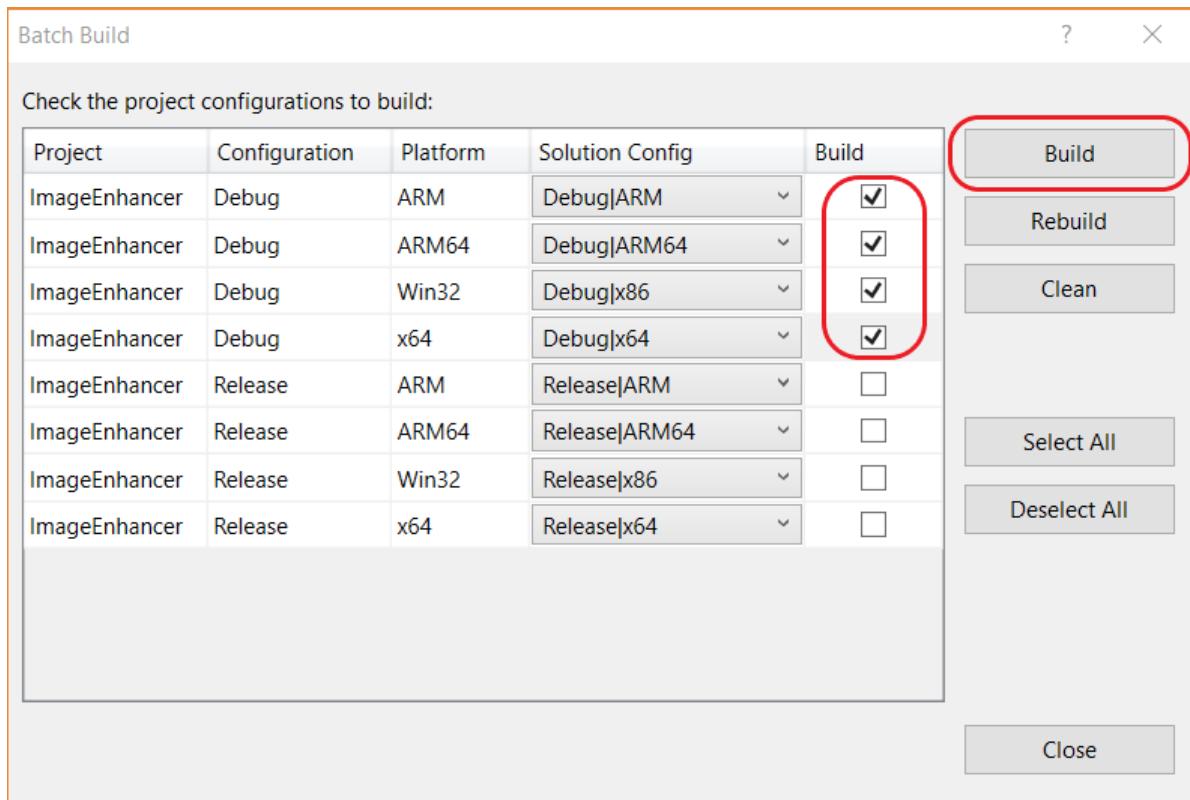
2. 右键单击“解决方案资源管理器”中的项目, 选择“添加”>“新建项目”, 单击“Visual C++”>“XAML”节点, 选择“模板控制”, 将名称更改为“AwesomeImageControl.cpp”, 然后单击“添加”:



3. 右键单击“解决方案资源管理器”中的项目，然后选择“属性”在“属性”页中，展开“配置属性”>“C/C++”，然后单击“输出文件”。在右侧窗格中，将“生成 XML 文档文件”的值更改为“Yes”：



4. 现在，右键单击“解决方案”，选择“批生成”，选中对话框中的三个 Debug 框，如下所示。这样可确保在生成时，将为 Windows 支持的每个目标系统生成一套完整的项目。



5. 在“批生成”对话框中，单击“生成”，验证项目并创建 NuGet 包所需的输出文件。

NOTE

在本演练中，将使用包的 Debug 项目。对于非调试包，请选中“批量生成”对话框中的 Release 选项，然后在以下步骤中引用生成的 Release 文件夹。

创建并更新 .nuspec 文件

若要创建初始 `.nuspec` 文件，请执行以下三个步骤。后续部分将引导你完成其他必要的更新。

1. 打开命令提示符并导航到包含 `ImageEnhancer.vcxproj` 的文件夹(这将是解决方案文件下方的子文件夹)。
2. 运行 NuGet `spec` 命令生成 `ImageEnhancer.nuspec` (文件的名称取自 `.vcxproj` 文件的名称)：

```
nuget spec
```

3. 在编辑器中打开 `ImageEnhancer.nuspec`，将其更新为与以下内容匹配，并将 YOUR_NAME 替换为适当的值。具体而言，`<id>` 值在 nuget.org 中必须是唯一的(请参阅[创建包中所述的命名约定](#))。另请注意，还必须更新创建者和说明标记，否则在打包步骤中会出现错误。

```
<?xml version="1.0"?>
<package>
  <metadata>
    <id>ImageEnhancer.YOUR_NAME</id>
    <version>1.0.0</version>
    <title>ImageEnhancer</title>
    <authors>YOUR_NAME</authors>
    <owners>YOUR_NAME</owners>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>Awesome Image Enhancer</description>
    <releaseNotes>First release</releaseNotes>
    <copyright>Copyright 2016</copyright>
    <tags>image enhancer imageenhancer</tags>
  </metadata>
</package>
```

NOTE

对于供公共使用而生成的包, 请特别注意 `<tags>` 元素, 因为这些标记可帮助其他人查找包并了解其用途。

将 Windows 元数据添加到包

Windows 运行时组件需要描述其所有公共可用类型的元数据, 这使得其他应用和库可使用该组件。该元数据包含在 .winmd 文件中, 此文件在编译项目时创建, 并且必须包含在 NuGet 包中。同时还生成具有 IntelliSense 数据的 XML 文件, 并且应该包含在内。

将以下 `<files>` 节点添加到 `.nuspec` 文件:

```
<package>
  <metadata>
    ...
  </metadata>

  <files>
    <!-- WinMd and IntelliSense files -->
    <file src=".\\Debug\\ImageEnhancer\\ImageEnhancer.winmd" target="lib\\uap10.0" />
    <file src=".\\Debug\\ImageEnhancer\\ImageEnhancer.xml" target="lib\\uap10.0" />
  </files>
</package>
```

添加 XAML 内容

若要在组件中包含 XAML 控件, 需要添加具有默认控件模板的 XAML 文件(由项目模板生成)。这也将进入

`<files>` 部分:

```
<?xml version="1.0"?>
<package>
  <metadata>
    ...
  </metadata>
  <files>
    ...

    <!-- XAML controls -->
    <file src="Themes\\Generic.xaml" target="lib\\uap10.0\\Themes" />

  </files>
</package>
```

添加本机实现库

在组件中, `ImageEnhancer` 类型的核心逻辑是本机代码, 包含在针对每个目标运行时(ARM、x86 和 x64)生成的各种 `ImageEnhancer.dll` 程序集中。若要在包中包含这些文件, 请在 `<files>` 部分中引用它们及其关联的 .pri 资源文件:

```
<?xml version="1.0"?>
<package>
  <metadata>
    ...
  </metadata>
  <files>
    ...

    <!-- DLLs and resources -->
    <file src="..\ARM\Debug\ImageEnhancer\ImageEnhancer.dll" target="runtimes\win10-arm\native"/>
    <file src="..\ARM\Debug\ImageEnhancer\ImageEnhancer.pri" target="runtimes\win10-arm\native"/>

    <file src="..\ARM64\Debug\ImageEnhancer\ImageEnhancer.dll" target="runtimes\win10-arm64\native"/>
    <file src="..\ARM64\Debug\ImageEnhancer\ImageEnhancer.pri" target="runtimes\win10-arm64\native"/>

    <file src="..\x64\Debug\ImageEnhancer\ImageEnhancer.dll" target="runtimes\win10-x64\native"/>
    <file src="..\x64\Debug\ImageEnhancer\ImageEnhancer.pri" target="runtimes\win10-x64\native"/>

    <file src="..\Debug\ImageEnhancer\ImageEnhancer.dll" target="runtimes\win10-x86\native"/>
    <file src="..\Debug\ImageEnhancer\ImageEnhancer.pri" target="runtimes\win10-x86\native"/>

  </files>
</package>
```

添加 .targets

接下来, 可能使用 NuGet 包的 C++ 和 JavaScript 项目需要一个 `.targets` 文件来标识必要的程序集和 `winmd` 文件。(C# 和 Visual Basic 项目自动执行此操作。)通过将下方文本复制到 `ImageEnhancer.targets` 创建此文件, 并将其保存在与 `.nuspec` 文件相同的文件夹中。*说明:* 此 `.targets` 文件需要与包 ID(例如 `.nuspec` 文件中的 `<Id>` 元素)名称相同:

```
<?xml version="1.0" encoding="utf-8"?>
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <ImageEnhancer-Platform Condition="'$(Platform)' == 'Win32'">x86</ImageEnhancer-Platform>
    <ImageEnhancer-Platform Condition="'$(Platform)' != 'Win32'">$(Platform)</ImageEnhancer-Platform>
  </PropertyGroup>
  <ItemGroup Condition="'$(TargetPlatformIdentifier)' == 'UAP'">
    <Reference Include="$(MSBuildThisFileDirectory)..\..\lib\uap10.0\ImageEnhancer.winmd">
      <Implementation>ImageEnhancer.dll</Implementation>
    </Reference>
    <ReferenceCopyLocalPaths Include="$(MSBuildThisFileDirectory)..\..\runtimes\win10-$(ImageEnhancer-Platform)\native\ImageEnhancer.dll" />
  </ItemGroup>
</Project>
```

然后在 `.nuspec` 文件中引用 `ImageEnhancer.targets`:

```
<?xml version="1.0"?>
<package>
  <metadata>
    ...
  </metadata>
  <files>
    ...
    <!-- .targets -->
    <file src="ImageEnhancer.targets" target="build\native"/>
  </files>
</package>
```

最终 .nuspec

现在，最终 `.nuspec` 文件应该如下所示，其中应再次将 YOUR_NAME 替换为适当的值：

```
<?xml version="1.0"?>
<package>
  <metadata>
    <id>ImageEnhancer.YOUR_NAME</id>
    <version>1.0.0</version>
    <title>ImageEnhancer</title>
    <authors>YOUR_NAME</authors>
    <owners>YOUR_NAME</owners>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>Awesome Image Enhancer</description>
    <releaseNotes>First Release</releaseNotes>
    <copyright>Copyright 2016</copyright>
    <tags>image enhancer imageenhancer</tags>
  </metadata>
  <files>
    <!-- WinMd and IntelliSense -->
    <file src=".\\Debug\\ImageEnhancer\\ImageEnhancer.winmd" target="lib\\uap10.0" />
    <file src=".\\Debug\\ImageEnhancer\\ImageEnhancer.xml" target="lib\\uap10.0" />

    <!-- XAML controls -->
    <file src="Themes\\Generic.xaml" target="lib\\uap10.0\\Themes" />

    <!-- DLLs and resources -->
    <file src=".\\ARM\\Debug\\ImageEnhancer\\ImageEnhancer.dll" target="runtimes\\win10-arm\\native" />
    <file src=".\\ARM\\Debug\\ImageEnhancer\\ImageEnhancer.pri" target="runtimes\\win10-arm\\native" />
    <file src=".\\ARM64\\Debug\\ImageEnhancer\\ImageEnhancer.dll" target="runtimes\\win10-arm64\\native" />
    <file src=".\\ARM64\\Debug\\ImageEnhancer\\ImageEnhancer.pri" target="runtimes\\win10-arm64\\native" />
    <file src=".\\x64\\Debug\\ImageEnhancer\\ImageEnhancer.dll" target="runtimes\\win10-x64\\native" />
    <file src=".\\x64\\Debug\\ImageEnhancer\\ImageEnhancer.pri" target="runtimes\\win10-x64\\native" />
    <file src=".\\Debug\\ImageEnhancer\\ImageEnhancer.dll" target="runtimes\\win10-x86\\native" />
    <file src=".\\Debug\\ImageEnhancer\\ImageEnhancer.pri" target="runtimes\\win10-x86\\native" />

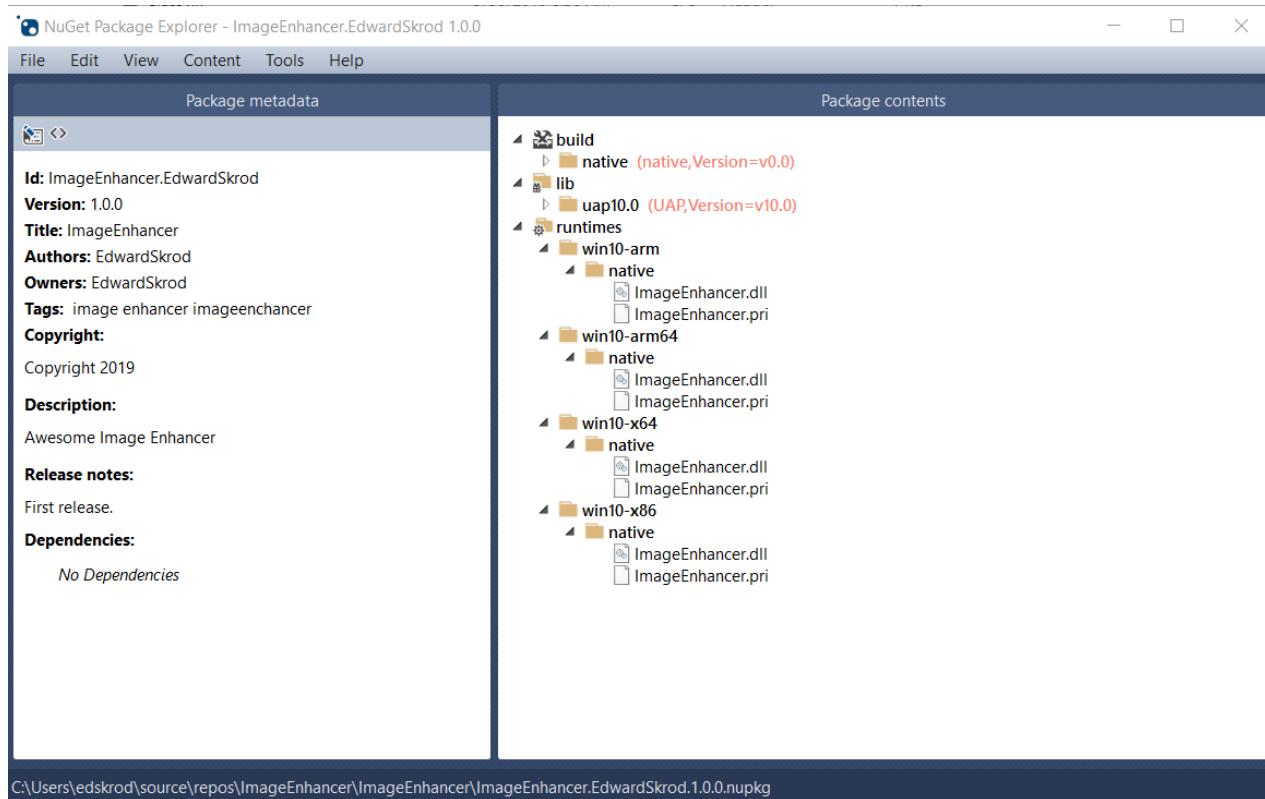
    <!-- .targets -->
    <file src="ImageEnhancer.targets" target="build\\native" />
  </files>
</package>
```

打包组件

如果已完成的 `.nuspec` 引用需要包含在包中的所有文件，便可运行 `pack` 命令：

```
nuget pack ImageEnhancer.nuspec
```

这将生成 `ImageEnhancer.YOUR_NAME.1.0.0.nupkg`。在类似 NuGet 包资源管理器的工具中打开此文件并展开所有节点，即可看到以下内容：



C:\Users\edskrod\source\repos\ImageEnhancer\ImageEnhancer\ImageEnhancer.EdwardSkrod.1.0.0.nupkg

TIP

`.nupkg` 文件实际上是具有其他扩展名的 ZIP 文件。然后，也可通过将 `.nupkg` 更改为 `.zip` 来检查包内容，但将包上传到 nuget.org 之前，请记得恢复扩展名。

若要使你的包可供其他开发人员使用，请按照[发布包](#)中的说明进行操作。

相关主题

- [.nuspec 引用](#)
- [符号包](#)
- [包版本控制](#)
- [支持多个 .NET Framework 版本](#)
- [将 MSBuild 属性和目标包含到包中](#)
- [创建本地化包](#)

创建 UWP 包 (C#)

2020/4/8 • [Edit Online](#)

通用 Windows 平台 (UWP) 为运行 Windows 10 的每台设备提供了一个通用应用平台。在此模型中, UWP 应用可以调用所有设备通用的 WinRT API, 以及特定于运行该应用的设备系列的 API(包括 Win32 和 .NET)。

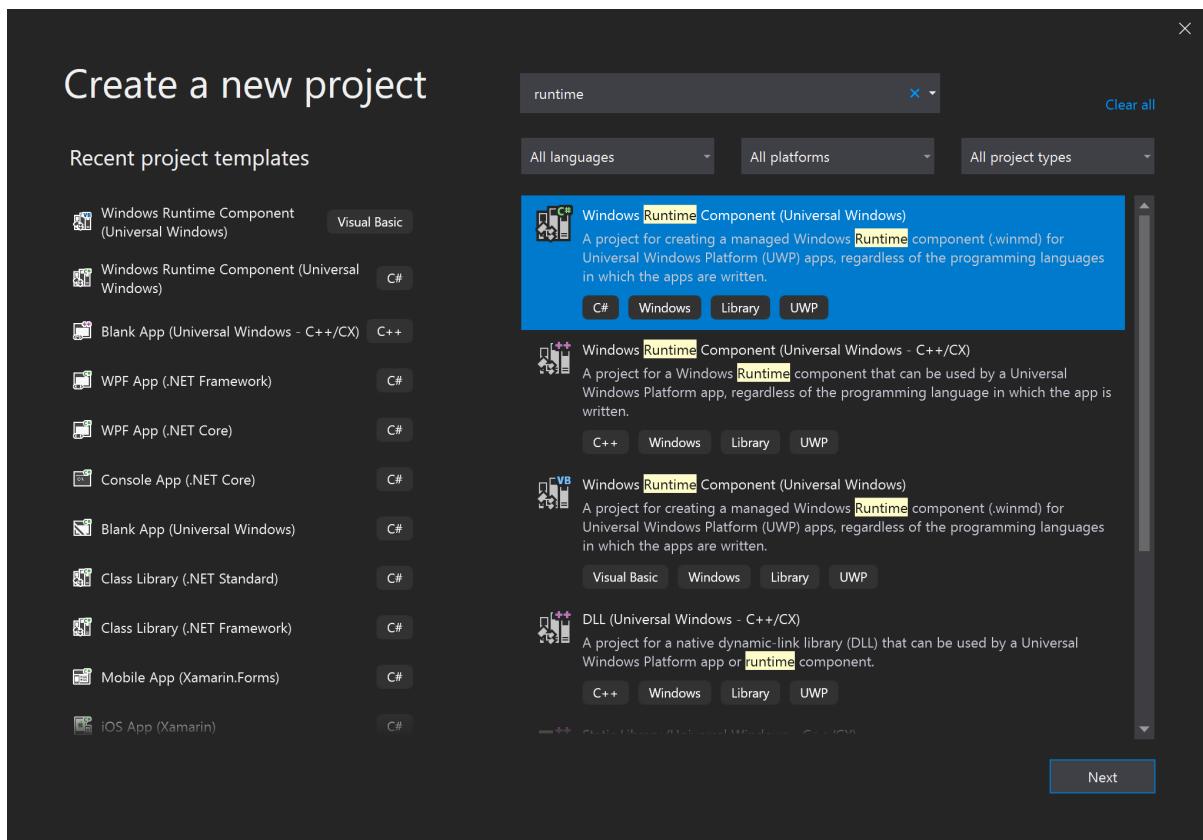
在本演练中, 将创建一个具有 C# UWP 组件(包括 XAML 控件)的 NuGet 包, 以便在托管项目和本机项目中使用。

必备条件

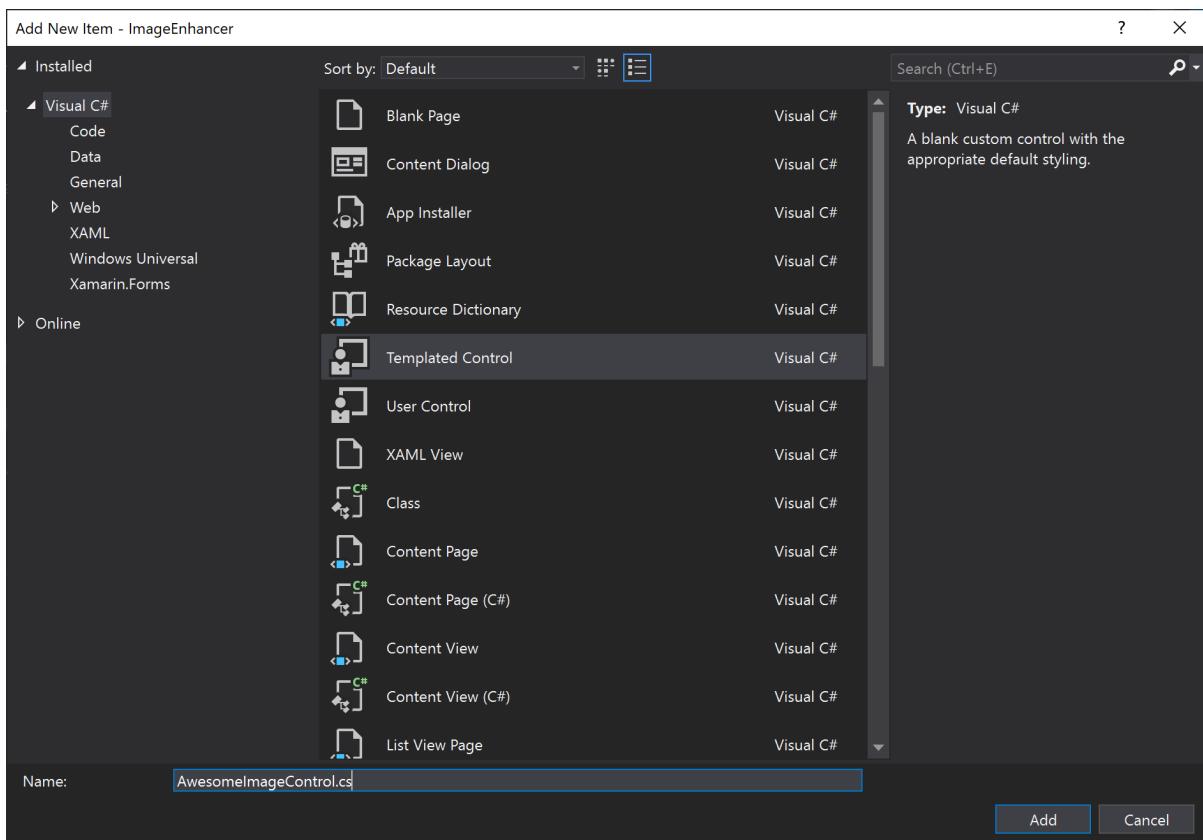
1. Visual Studio 2019。可以从 [visualstudio.com](#) 免费安装 2019 Community 版; 也可以使用 Professional 和 Enterprise 版。
2. NuGet CLI。从 `nuget.exe` [nuget.org/downloads](#) 下载 的最新版本, 将其保存到选择的位置(`.exe` 是直接下载的)。然后将该位置添加到 PATH 环境变量(如果尚未添加)。[更多详细信息](#)。

创建 UWP Windows 运行时组件

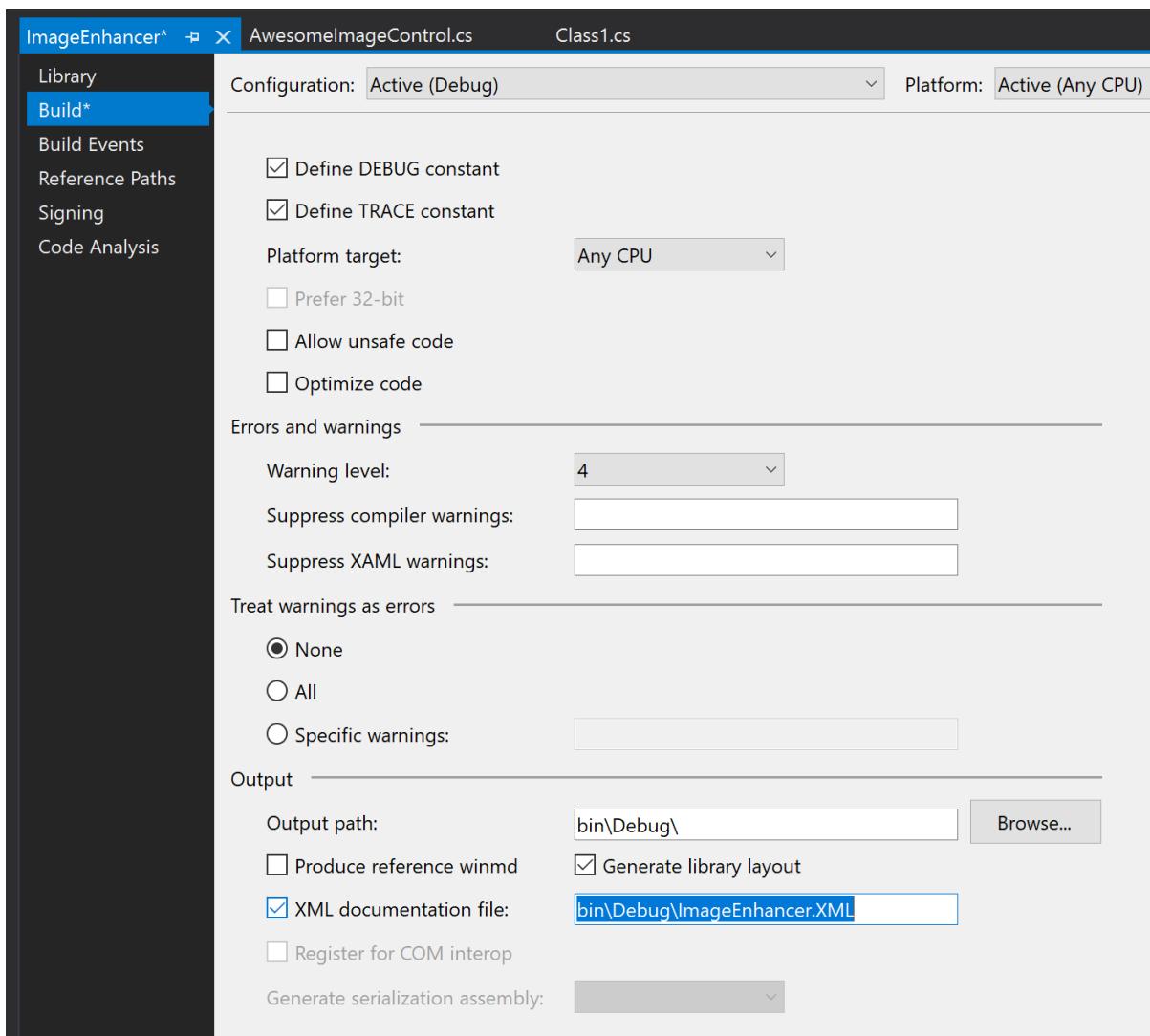
1. 在 Visual Studio 中, 选择“文件”>“新建”>“项目”, 搜索“uwp c#”, 然后选择“Windows 运行时组件(通用 Windows)”模板, 单击“下一步”, 将名称更改为 ImageEnhancer, 然后单击“创建”。出现提示时, 接受“目标版本”和“最低版本”的默认值。



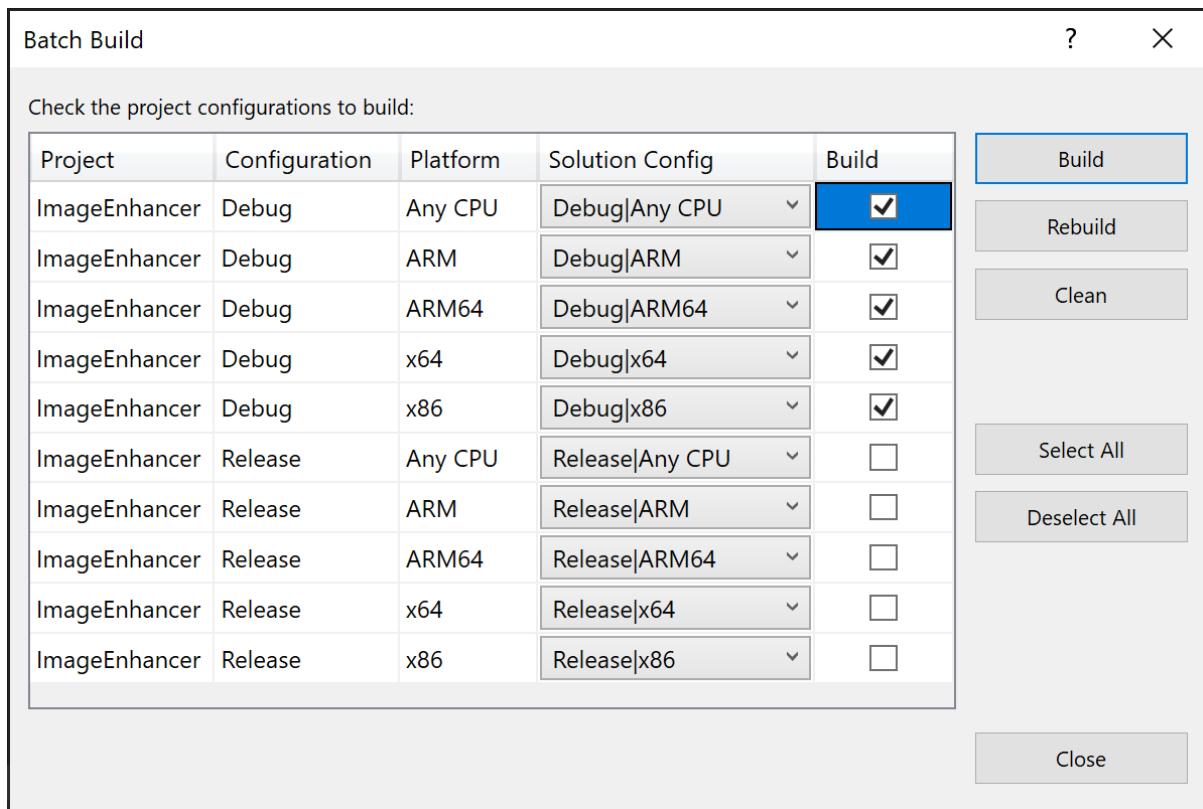
2. 右键单击“解决方案资源管理器”中的项目, 选择“添加”>“新建项目”, 选择“模板控制”, 将名称改为 AwesomelImageControl.cs, 然后单击“添加”:



3. 右键单击“解决方案资源管理器”中的项目，然后选择“属性”在“属性”页面上，选择“生成”选项卡并启用“XML 文档文件”：



4. 右键单击“解决方案”，选择“批生成”，选中对话框中的五个生成框，如下所示。这样可确保在生成时，将为 Windows 支持的每个目标系统生成一套完整的项目。



5. 在“批生成”对话框中，单击“生成”，验证项目并创建 NuGet 包所需的输出文件。

NOTE

在本演练中，将使用包的 Debug 项目。对于非调试包，请选中“批量生成”对话框中的 Release 选项，然后在以下步骤中引用生成的 Release 文件夹。

创建并更新 .nuspec 文件

若要创建初始 `.nuspec` 文件，请执行以下三个步骤。后续部分将引导你完成其他必要的更新。

1. 打开命令提示符并导航到包含 `ImageEnhancer.csproj` 的文件夹(这将是解决方案文件下方的子文件夹)。
2. 运行 `NuGet spec` 命令生成 `ImageEnhancer.nuspec` (此文件的名称取自 `.csproj` 文件的名称)：

```
nugget spec
```

3. 在编辑器中打开 `ImageEnhancer.nuspec`，将其更新为与以下内容匹配，并将 YOUR_NAME 替换为适当的值。
\$propertyName\$ 值均不得留空。具体而言，`<id>` 值在 nuget.org 中必须是唯一的(请参阅[创建包](#)中所述的命名约定)。另请注意，还必须更新创建者和说明标记，否则在打包步骤中会出现错误。

```
<?xml version="1.0"?>
<package>
  <metadata>
    <id>ImageEnhancer.YOUR_NAME</id>
    <version>1.0.0</version>
    <title>ImageEnhancer</title>
    <authors>YOUR_NAME</authors>
    <owners>YOUR_NAME</owners>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>Awesome Image Enhancer</description>
    <releaseNotes>First release</releaseNotes>
    <copyright>Copyright 2020</copyright>
    <tags>image enhancer imageenhancer</tags>
  </metadata>
</package>
```

NOTE

对于供公共使用而生成的包, 请特别注意 `<tags>` 元素, 因为这些标记可帮助其他人查找包并了解其用途。

将 Windows 元数据添加到包

Windows 运行时组件需要描述其所有公共可用类型的元数据, 这使得其他应用和库可使用该组件。该元数据包含在 .winmd 文件中, 此文件在编译项目时创建, 并且必须包含在 NuGet 包中。同时还生成具有 IntelliSense 数据的 XML 文件, 并且应该包含在内。

将以下 `<files>` 节点添加到 `.nuspec` 文件:

```
<package>
  <metadata>
    ...
  </metadata>

  <files>
    <!-- WinMd and IntelliSense files -->
    <file src="bin\Debug\ImageEnhancer.winmd" target="lib\uap10.0"/>
    <file src="bin\Debug\ImageEnhancer.xml" target="lib\uap10.0"/>
  </files>
</package>
```

添加 XAML 内容

若要在组件中包含 XAML 控件, 需要添加具有默认控件模板的 XAML 文件(由项目模板生成)。这也将进入 `<files>` 部分:

```
<?xml version="1.0"?>
<package>
  <metadata>
    ...
  </metadata>
  <files>
    ...

    <!-- XAML controls -->
    <file src="Themes\Generic.xaml" target="lib\uap10.0\Themes"/>

  </files>
</package>
```

添加本机实现库

在组件中, ImageEnhancer 类型的核心逻辑是本机代码, 包含在针对每个目标运行时(ARM、ARM64、x86 和 x64)生成的各种 `ImageEnhancer.winmd` 程序集中。若要在包中包含这些文件, 请在 `<files>` 部分中引用它们及其关联的 `.pri` 资源文件:

```
<?xml version="1.0"?>
<package>
  <metadata>
    ...
  </metadata>
  <files>
    ...

    <!-- WINMDs and resources -->
    <file src="bin\ARM\Debug\ImageEnhancer.winmd" target="runtimes\win10-arm"/>
    <file src="bin\ARM\Debug\ImageEnhancer.pri" target="runtimes\win10-arm"/>

    <file src="bin\ARM64\Debug\ImageEnhancer.winmd" target="runtimes\win10-arm64"/>
    <file src="bin\ARM64\Debug\ImageEnhancer.pri" target="runtimes\win10-arm64"/>

    <file src="bin\x64\Debug\ImageEnhancer.winmd" target="runtimes\win10-x64"/>
    <file src="bin\x64\Debug\ImageEnhancer.pri" target="runtimes\win10-x64"/>

    <file src="bin\x86\Debug\ImageEnhancer.winmd" target="runtimes\win10-x86"/>
    <file src="bin\x86\Debug\ImageEnhancer.pri" target="runtimes\win10-x86"/>

  </files>
</package>
```

最终 .nuspec

现在, 最终 `.nuspec` 文件应该如下所示, 其中应再次将 `YOUR_NAME` 替换为适当的值:

```
<?xml version="1.0"?>
<package>
  <metadata>
    <id>ImageEnhancer.YOUR_NAME</id>
    <version>1.0.0</version>
    <title>ImageEnhancer</title>
    <authors>YOUR_NAME</authors>
    <owners>YOUR_NAME</owners>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>Awesome Image Enhancer</description>
    <releaseNotes>First Release</releaseNotes>
    <copyright>Copyright 2020</copyright>
    <tags>image enhancer imageenhancer</tags>
  </metadata>
  <files>
    <!-- WinMd and IntelliSense -->
    <file src="bin\Debug\ImageEnhancer.winmd" target="lib\uap10.0"/>
    <file src="bin\Debug\ImageEnhancer.xml" target="lib\uap10.0"/>

    <!-- XAML controls -->
    <file src="Themes\Generic.xaml" target="lib\uap10.0\Themes"/>

    <!-- WINMDs and resources -->
    <file src="bin\ARM\Debug\ImageEnhancer.winmd" target="runtimes\win10-arm"/>
    <file src="bin\ARM\Debug\ImageEnhancer.pri" target="runtimes\win10-arm"/>

    <file src="bin\ARM64\Debug\ImageEnhancer.winmd" target="runtimes\win10-arm64"/>
    <file src="bin\ARM64\Debug\ImageEnhancer.pri" target="runtimes\win10-arm64"/>

    <file src="bin\x64\Debug\ImageEnhancer.winmd" target="runtimes\win10-x64"/>
    <file src="bin\x64\Debug\ImageEnhancer.pri" target="runtimes\win10-x64"/>

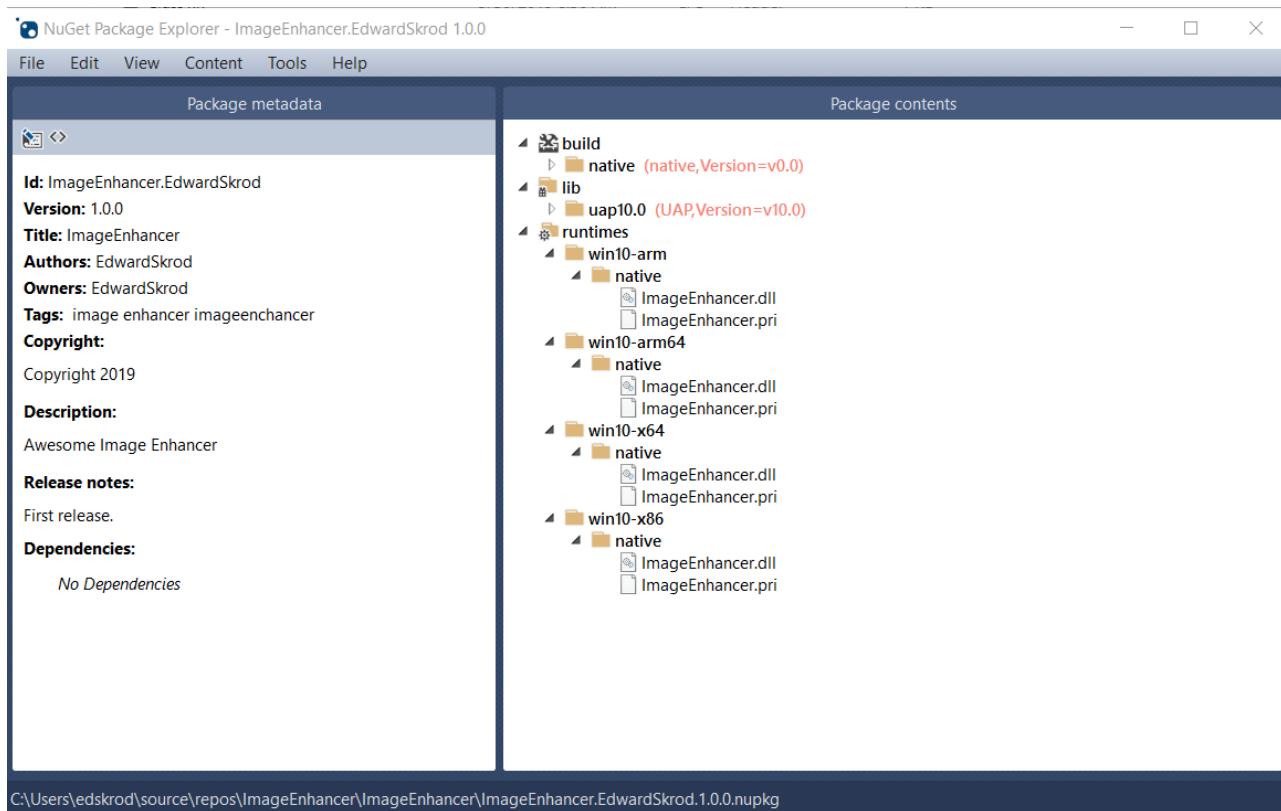
    <file src="bin\x86\Debug\ImageEnhancer.winmd" target="runtimes\win10-x86"/>
    <file src="bin\x86\Debug\ImageEnhancer.pri" target="runtimes\win10-x86"/>
  </files>
</package>
```

打包组件

如果已完成的 `.nuspec` 引用需要包含在包中的所有文件，便可运行 `nuget pack` 命令：

```
nuget pack ImageEnhancer.nuspec
```

这将生成 `ImageEnhancer.YOUR_NAME.1.0.0.nupkg`。在类似 [NuGet 包资源管理器](#) 的工具中打开此文件并展开所有节点，即可看到以下内容：



TIP

.nupkg 文件实际上是具有其他扩展名的 ZIP 文件。然后，也可通过将 .nupkg 更改为 .zip 来检查包内容，但将包上传到 nuget.org 之前，请记得恢复扩展名。

若要使你的包可供其他开发人员使用，请按照[发布包](#)中的说明进行操作。

相关主题

- [.nuspec 引用](#)
- [符号包](#)
- [包版本控制](#)
- [支持多个 .NET Framework 版本](#)
- [将 MSBuild 属性和目标包含到包中](#)
- [创建本地化包](#)

创建本机包

2020/4/8 • [Edit Online](#)

本机包包含本机二进制文件(而不是托管程序集), 因此可用于 C++(或类似)项目中。(请参阅“使用”部分中的[本机 C++ 包](#)。)

若要用于 C++ 项目中, 包必须面向 `native` 框架。目前不存在与此框架关联的任何版本号, 因为 NuGet 对所有 C++ 项目的处理方式相同。

NOTE

请确保在 `.nuspec` 的部分中包括本机, 以便其他开发人员通过搜索该标记查找包 `<tags>``.nuspec`。

然后, 面向 `native` 的本机 NuGet 包在 `\build`、`\content` 和 `\tools` 文件夹中提供文件;此事例中没有使用 `\lib` (NuGet 不能直接向 C++ 项目添加引用)。包还可能包括 `\build` 中的目标和属性文件, NuGet 会将这些内容自动导入到使用该包的项目中。这些文件的名称必须与包 ID 的名称相同, 包含 `.targets` 和/或 `.props` 扩展名。例如, `cpprestsdk` 包在 `cpprestsdk.targets` 文件夹中包括 `\build` 文件。

`\build` 文件夹可用于所有 NuGet 包, 并非仅用于本机包。与 `\build`、`\content` 和 `\lib` 文件夹相同, `\tools` 文件夹也遵照目标框架。这意味着可创建 `\build\net40` 文件夹和 `\build\net45` 文件夹, 且 NuGet 会向项目中导入相应的属性和目标文件。(无需使用 PowerShell 脚本导入 MSBuild 目标。)

以 NuGet 包形式创建 UI 控件

2020/4/8 • [Edit Online](#)

从 Visual Studio 2017 开始，可以利用在 NuGet 包中提供的 UWP 和 WPF 控件的附加功能。本指南使用 [Extension SDK as NuGet Package 示例](#) 来演练 UWP 控件上下文中的这些功能。这同样适用于 WPF 控件，除非另行指定。

先决条件

1. Visual Studio 2017
2. 了解如何[创建 UWP 包](#)

生成库布局

NOTE

这仅适用于 UWP 控件。

设置 `GenerateLibraryLayout` 属性可确保项目生成输出在已准备好打包的布局中生成，而无需 nuspec 中的单独文件项。

在项目属性中，转到“生成”选项卡并选中“生成库布局”复选框。这将修改项目文件，并针对当前所选的生成配置和平台将 `GenerateLibraryLayout` 标志设置为“true”。

或者，编辑项目文件以将 `<GenerateLibraryLayout>true</GenerateLibraryLayout>` 添加到第一个无条件属性组。无论生成配置和平台如何，这都将应用该属性。

添加对 XAML 控件的工具箱/资产窗格支持

要使 XAML 控件出现在 Visual Studio 中的 XAML 设计器工具箱和 Blend 的“资产”窗格中，请在包项目的 `tools` 文件夹根中创建 `VisualStudioToolsManifest.xml` 文件。如果不需要控件显示在工具箱或“资产”窗格中，则不需要此文件。

```
\build  
\lib  
\tools  
    VisualStudioToolsManifest.xml
```

文件的结构如下所示：

```
<FileList>  
  <File Reference = "your_package_file">  
    <ToolboxItems VSCategory="vs_category" BlendCategory="blend_category">  
      <Item Type="type_full_name_1" />  
  
      <!-- Any number of additional Items -->  
      <Item Type="type_full_name_2" />  
      <Item Type="type_full_name_3" />  
    </ToolboxItems>  
  </File>  
</FileList>
```

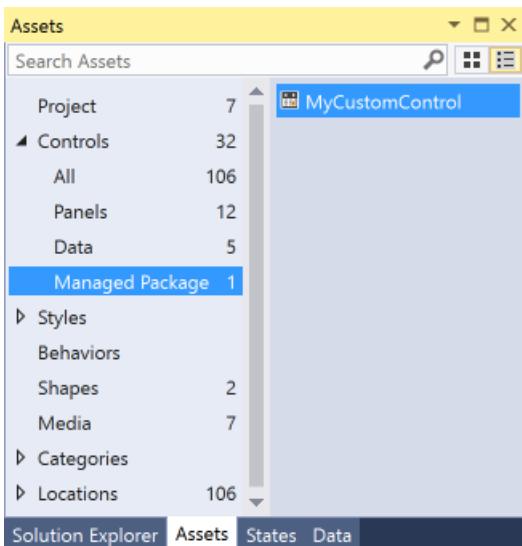
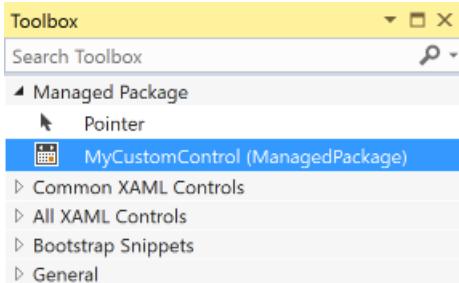
其中：

- *your_package_file*: 控件文件的名称, 例如 `ManagedPackage.winmd` ("ManagedPackage"是本示例中随意起的名称, 没有其他意义)。
- *vs_category*: Visual Studio 设计器工具箱中应出现控件的组的标签。 `VSCategory` 对控件出现在工具箱中是必要的。
- *blend_category*: Blend 设计器的"资产"窗格中应出现控件的组的标签。 `BlendCategory` 对控件出现在"资产"中是必要的。
- *type_full_name_n*: 每个控件的完全限定名称, 包括命名空间, 例如 `ManagedPackage.MyCustomControl`。注意, 点格式用于托管和本机类型。

在更高级的方案中, 当单个包包含多个控件程序集时, 还可以在 `<FileList>` 中包括多个 `<File>` 元素。如果需要将控件整理为单独的分类, 则还可以在单个 `<File>` 中有多个 `<ToolboxItems>` 节点。

在以下示例中, `ManagedPackage.winmd` 中实现的控件将出现在 Visual Studio 和 Blend 名为"托管包"的组中, "MyCustomControl"将出现在该组中。所有名称都是随意起的。

```
<FileList>
  <File Reference = "ManagedPackage.winmd">
    <ToolboxItems VSCategory="Managed Package" BlendCategory="Managed Package">
      <Item Type="ManagedPackage.MyCustomControl" />
    </ToolboxItems>
  </File>
</FileList>
```



NOTE

必须显式指定每个需要在工具箱/资产窗格中看见的控件。请确保以 `Namespace.ControlName` 格式指定它们。

将自定义图标添加到控件

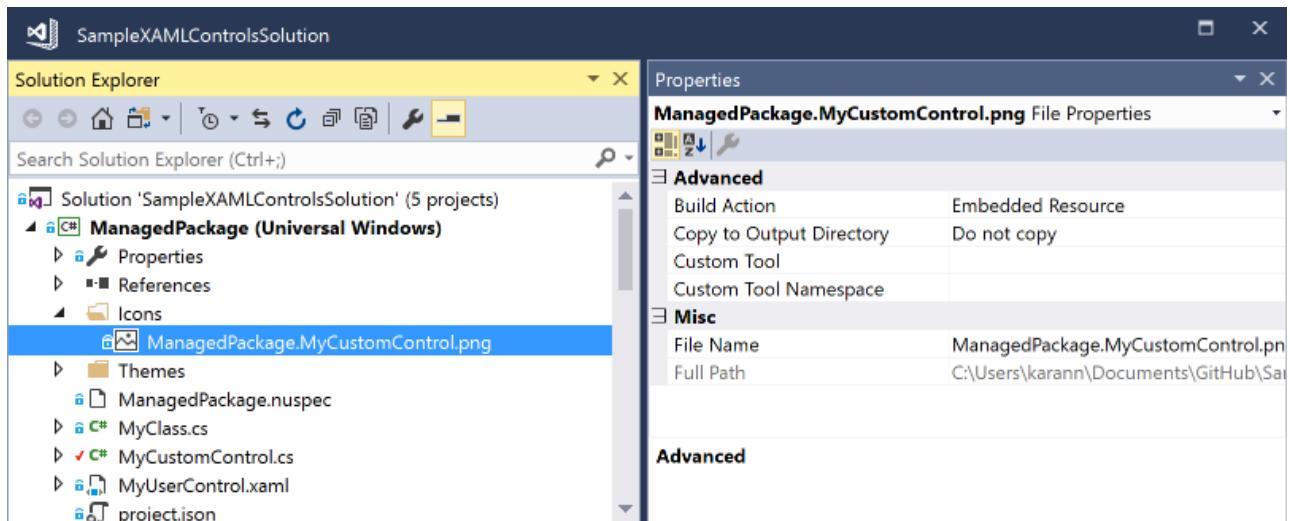
要在工具箱/资产窗格中显示自定义图标,请将图像添加到项目或名为“Namespace.ControlName.extension”的对应 `design.dll` 项目,并将生成操作设为“嵌入资源”。还必须确保关联的 `AssemblyInfo.cs` 指定 `ProvideMetadata` 属性
- `[assembly: ProvideMetadata(typeof(RegisterMetadata))]`。请参阅此[示例](#)。

支持的格式为 `.png`、`.jpg`、`.jpeg`、`.gif` 和 `.bmp`。推荐格式为 16 x 16 像素 BMP24。



运行时将替换粉色背景。更改 Visual Studio 主题且需要设置背景颜色时,图标会重新着色。有关详细信息,请参阅[Visual Studio 的图像和图标](#)。

在以下示例中,项目包含名为“ManagedPackage.MyCustomControl.png”的图像文件。



NOTE

对于本机控件,必须将图标以资源的形式放在 `design.dll` 项目中。

支持特定 Windows 平台版本

UWP 包有 `TargetPlatformVersion` (TPV) 和 `TargetPlatformMinVersion` (TPMinV), 定义了可以安装应用的 OS 版本的上限和下线。TPV 进一步指定生成应用的 SDK 的版本。创作 UWP 包时注意这些属性:使用应用定义的平台版本界限意外的 API 将导致生成失败或应用在运行时失败。

例如,假如已将控件包的 TPMinV 设为 Windows 10 Anniversary Edition(10.0;版本 14393),因此需要确保仅与下限相匹配的 UWP 项目使用此包。要使得包被 UWP 项目使用,你必须使用以下文件夹名称打包控件:

```
\lib\uap10.0.14393\*  
\ref\uap10.0.14393\*
```

NuGet 将自动检查正在使用项目的 TPMinV,如果低于 Windows 10 Anniversary Edition(10.0;版本 14393),则安装失败

在 WPF 中,我们假设你希望 WPF 控件包由面向 .NET Framework v4.6.1 或更高版本的项目使用。若要强制执行此操作,必须使用以下文件夹名称来打包控件:

```
\lib\net461\*  
\ref\net461\*
```

添加设计时支持

要配置控件属性在属性检查器中显示的位置、添加自定义装饰器等，请将 `design.dll` 文件放在目标平台对应的 `lib\uap10.0.14393\Design` 文件夹中。此外，要确保“[编辑模板](#)”>“[编辑副本](#)”功能正常工作，必须包含 `Generic.xaml` 及其在 `<your_assembly_name>\Themes` 文件夹中合并的任何资源字典(同样，使用实际的程序集名称)。(此文件对控件的运行时行为不产生影响。)文件夹结构将如下所示：

```
\lib
  \uap10.0.14393
    \Design
      \MyControl.design.dll
    \your_assembly_name
      \Themes
        Generic.xaml
```

对于 WPF，请继续上述示例，即希望由面向 .NET Framework v4.6.1 或更高版本的项目来使用 WPF 控件包：

```
\lib
  \net461
    \Design
      \MyControl.design.dll
    \your_assembly_name
      \Themes
        Generic.xaml
```

NOTE

默认情况下，控件属性将显示在属性检查器的“杂项”类别下。

使用字符串和资源

可以将字符串资源 (`.resw`) 嵌入在控件或者使用的 UWP 项目可使用的包中，将 `.resw` 文件的“生成操作”属性设为 `PRIResource`。

有关示例，请参考 [ExtensionSDKasNuGetPackage](#) 示例中的 `MyCustomControl.cs`。

NOTE

这仅适用于 UWP 控件。

请参阅

- [创建 UWP 包](#)
- [ExtensionSDKasNuGetPackage](#) 示例

分析器 NuGet 格式

2020/4/8 • [Edit Online](#)

借助 .NET Compiler Platform(也称为“Roslyn”), 开发人员可创建 [分析器](#), 用于在写入代码时检查其语法树和语义。这为开发人员提供了创建域特定的分析工具(如帮助指导如何使用特定 API 或库的工具)的方法。可以在 [.NET/Roslyn GitHub wiki](#) 上找到详细信息。另请参阅下文:MSDN 杂志中的[使用 Roslyn 编写 API 的实时代码分析器](#)

分析器本身通常作为实现 API 或相关库的 NuGet 包的一部分进行打包和分发。

有关典型的示例, 请参阅 [System.Runtime.Analyzers](#) 包, 其中包含以下内容:

- analyzers\dotnet\System.Runtime.Analyzers.dll
- analyzers\dotnet\cs\System.Runtime.CSharp.Analyzers.dll
- analyzers\dotnet\vb\System.Runtime.VisualBasic.Analyzers.dll
- build\System.Runtime.Analyzers.Common.props
- build\System.Runtime.Analyzers.props
- build\System.Runtime.CSharp.Analyzers.props
- build\System.Runtime.VisualBasic.Analyzers.props
- tools\install.ps1
- tools\uninstall.ps1

正如所见, 分析器 DLL 放置于包中的 `analyzers` 文件夹中。

属性文件放置于 `build` 文件夹中, 包括这些文件是为了禁用旧的 FxCop 规则以支持分析器实现。

支持使用 `packages.config` 的项目的安装和卸载脚本放置于 `tools` 中。

另请注意, 因为此包没有平台特定的需求, 因此 `platform` 文件夹已省略。

分析器路径格式

`analyzers` 文件夹的使用类似于用于 [框架目标](#), 只是路径中的说明符描述开发主机依赖项而不是生成时。常规格式如下所示:

```
$/analyzers/{framework_name}{version}/{supported_architecture}/{supported_language}/{analyzer_name}.dll
```

- **framework_name** 和**版本**: 所包含的 DLL 需要运行的 .NET framework 的可选 API 外围应用。`dotnet` 是目前唯一有效的值, 因为 Roslyn 是唯一可运行分析器的主机。如果未指定目标, 则假定 DLL 适用于所有目标。
- **supported_language**: DLL 适用的语言, `cs` (C#)、`vb` (Visual Basic) 和 `fs` 中的一种。语言表示应仅为使用该语言的项目加载分析器。如果未指定任何语言, 则假定 DLL 适用于支持分析器的所有语言。
- **analyzer_name**: 指定分析器的 DLL。如果需要 DLL 以外的其他文件, 必须通过目标或属性文件包括它们。

安装和卸载脚本

如果用户的项目使用的是 `packages.config`, 则选取分析器的 MSBuild 脚本不起作用, 因此应将具有以下所述内容的 `install.ps1` 和 `uninstall.ps1` 文件置于 `tools` 文件夹中。

install.ps1 文件内容

```

param($installPath, $toolsPath, $package, $project)

$analyzersPaths = Join-Path (Join-Path (Split-Path -Path $toolsPath -Parent) "analyzers" ) * -Resolve

foreach($analyzersPath in $analyzersPaths)
{
    # Install the language agnostic analyzers.
    if (Test-Path $analyzersPath)
    {
        foreach ($analyzerFilePath in Get-ChildItem $analyzersPath -Filter *.dll)
        {
            if($project.Object.AnalyzerReferences)
            {
                $project.Object.AnalyzerReferences.Add($analyzerFilePath.FullName)
            }
        }
    }
}

$project.Type # gives the language name like (C# or VB.NET)
$languageFolder = ""
if($project.Type -eq "C#")
{
    $languageFolder = "cs"
}
if($project.Type -eq "VB.NET")
{
    $languageFolder = "vb"
}
if($languageFolder -eq "")
{
    return
}

foreach($analyzersPath in $analyzersPaths)
{
    # Install language specific analyzers.
    $languageAnalyzersPath = join-path $analyzersPath $languageFolder
    if (Test-Path $languageAnalyzersPath)
    {
        foreach ($analyzerFilePath in Get-ChildItem $languageAnalyzersPath -Filter *.dll)
        {
            if($project.Object.AnalyzerReferences)
            {
                $project.Object.AnalyzerReferences.Add($analyzerFilePath.FullName)
            }
        }
    }
}

```

uninstall.ps1 文件内容

```

param($installPath, $toolsPath, $package, $project)

$analyzersPaths = Join-Path (Join-Path (Split-Path -Path $toolsPath -Parent) "analyzers" ) * -Resolve

foreach($analyzersPath in $analyzersPaths)
{
    # Uninstall the language agnostic analyzers.
    if (Test-Path $analyzersPath)
    {
        foreach ($analyzerFilePath in Get-ChildItem $analyzersPath -Filter *.dll)
        {
            if($project.Object.AnalyzerReferences)
            {
                $project.Object.AnalyzerReferences.Remove($analyzerFilePath.FullName)
            }
        }
    }
}

$project.Type # gives the language name like (C# or VB.NET)
$languageFolder = ""
if($project.Type -eq "C#")
{
    $languageFolder = "cs"
}
if($project.Type -eq "VB.NET")
{
    $languageFolder = "vb"
}
if($languageFolder -eq "")
{
    return
}

foreach($analyzersPath in $analyzersPaths)
{
    # Uninstall language specific analyzers.
    $languageAnalyzersPath = join-path $analyzersPath $languageFolder
    if (Test-Path $languageAnalyzersPath)
    {
        foreach ($analyzerFilePath in Get-ChildItem $languageAnalyzersPath -Filter *.dll)
        {
            if($project.Object.AnalyzerReferences)
            {
                try
                {
                    $project.Object.AnalyzerReferences.Remove($analyzerFilePath.FullName)
                }
                catch
                {

                }
            }
        }
    }
}

```

使用 Visual Studio 2017 或 2019 为 Xamarin 创建包

2020/4/8 • [Edit Online](#)

Xamarin 包包含在 iOS、Android 和 Windows 上使用本机 API 的代码，具体取决于运行时操作系统。虽然这很简单，但最好让开发人员通过通用的 API 外围应用从 PCL 或 .NET Standard 库中使用包。

在本演练中，将使用 Visual Studio 2017 或 2019 创建可在 iOS、Android 和 Windows 的移动项目中使用的跨平台 NuGet 包。

1. [先决条件](#)
2. [创建项目结构和抽象代码](#)
3. [编写平台特定的代码](#)
4. [创建并更新 .nuspec 文件](#)
5. [打包组件](#)
6. [相关主题](#)

必备条件

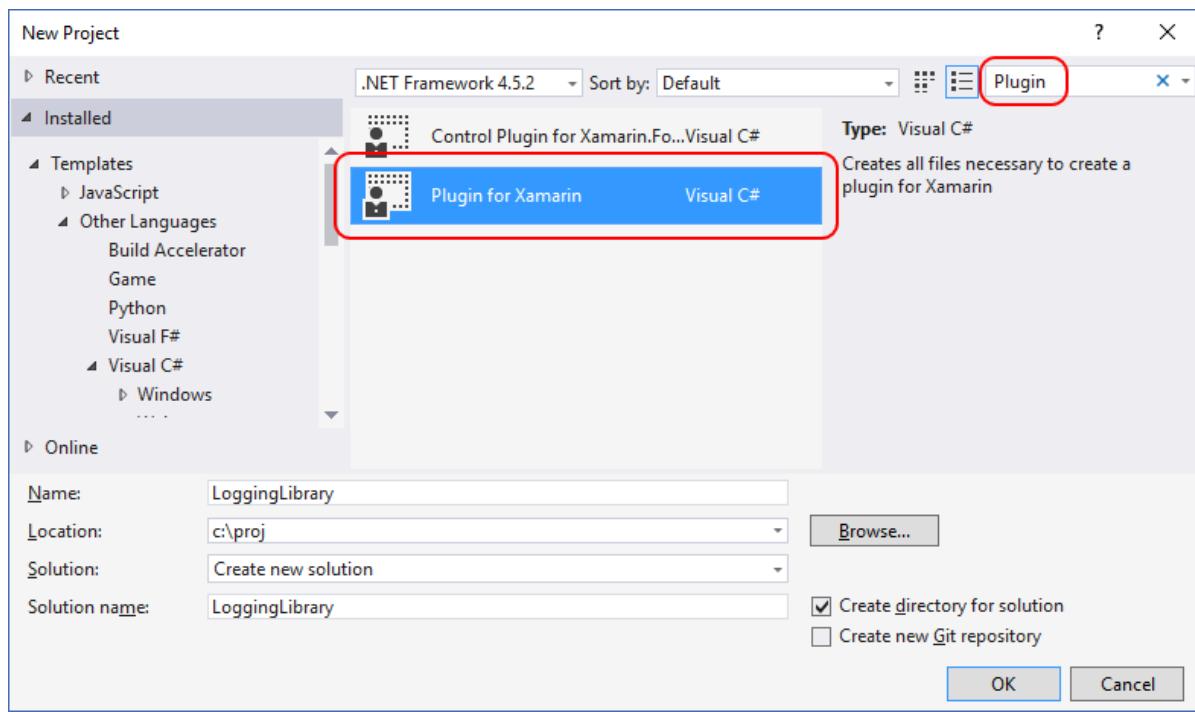
1. 在通用 Windows 平台 (UWP) 和 Xamarin 中使用 Visual Studio 2017 或 2019。可以从 [visualstudio.com](#) 免费安装 Community 版；当然，也可以使用 Professional 和 Enterprise 版。若要包含 UWP 和 Xamarin 工具，请选择自定义安装并选中相应的选项。
2. NuGet CLI。从 [nuget.org/downloads](#) 下载 nuget.exe 的最新版本，将其保存到选择的位置。然后将该位置添加到 PATH 环境变量（如果尚未添加）。

NOTE

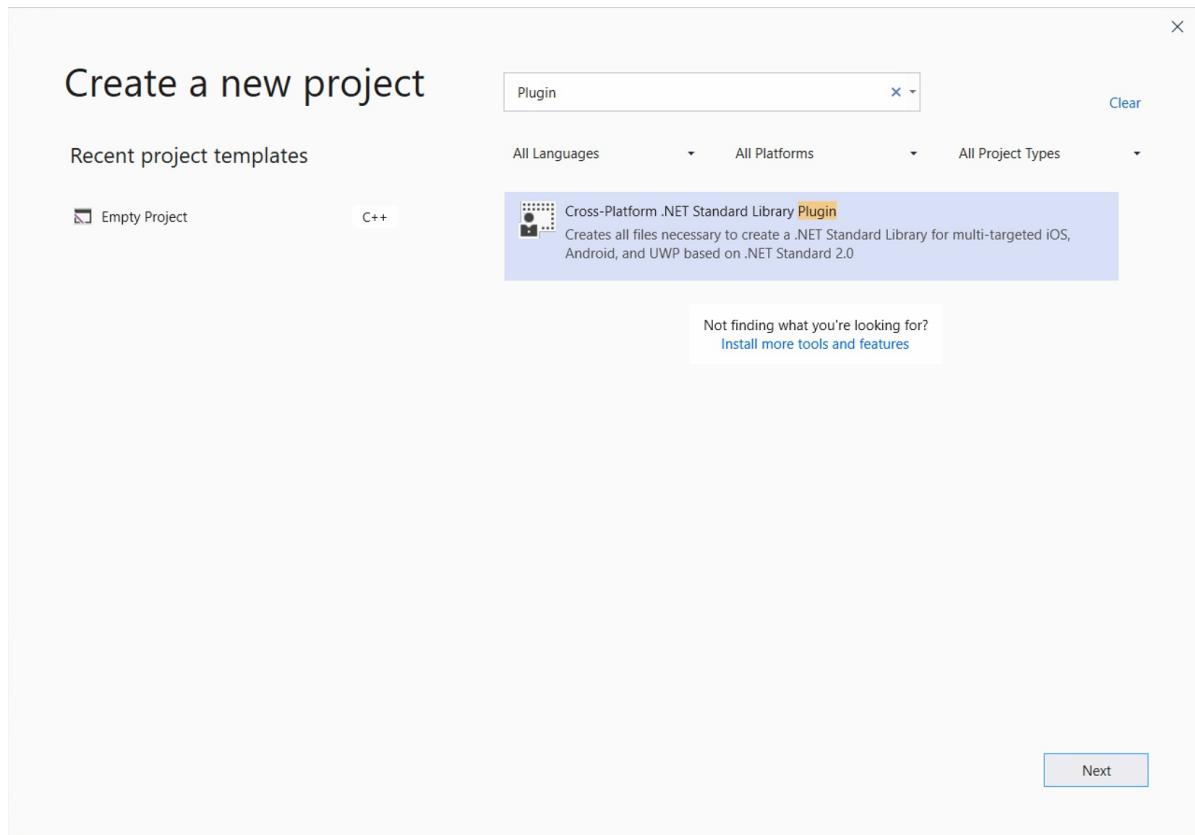
nuget.exe 是 CLI 工具本身，不是安装程序，因此一定要保存从浏览器下载的文件，而非运行。

创建项目结构和抽象代码

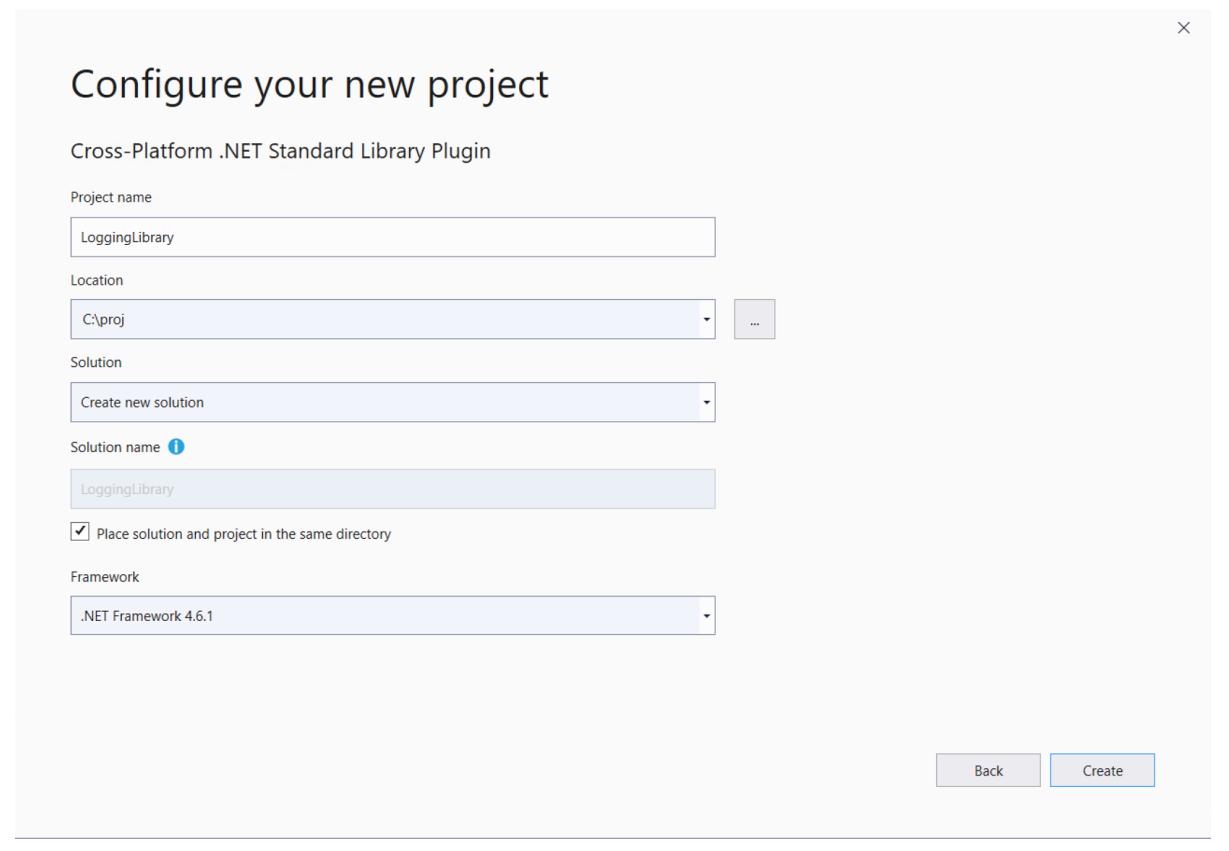
1. 下载并运行适用于 Visual Studio 的[跨平台 .NET Standard 插件模板扩展](#)。使用这些模板可轻松创建本演练所需的项目结构。
2. 在 Visual Studio 2017 中，选择“文件”>“新建”>“项目”，搜索，选择“跨平台 .NET Standard 库插件”模板，**将名称更改为“LoggingLibrary”**，然后单击“确定”**Plugin**。



在 Visual Studio 2019 中, 选择“文件”>“新建”>“项目”, 搜索, 选择“跨平台 .NET Standard 库插件”模板, 然后单击“下一步” `Plugin`。



将名称更改为 LoggingLibrary, 然后单击“创建”。



生成的解决方案包含两个共享项目，以及各种平台特定的项目：

- `ILoggingLibrary` 项目，该项目包含在 `ILoggingLibrary.shared.cs` 文件中，用于定义组件的公共接口（API 外围应用）。你将在此文件中定义库的接口。
- 另一个共享项目包含 `CrossLoggingLibrary.shared.cs` 中的代码，这些代码将在运行时定位抽象接口的平台特定实现。通常不需要修改此文件。
- 每个平台特定的项目（如 `LoggingLibrary.android.cs`）在其各自的 `LoggingLibraryImplementation.cs`（VS 2017）或 `LoggingLibrary.<PLATFORM>.cs`（VS 2019）文件中都包含该接口的本机实现。你将在此文件中生成库的代码。

默认情况下，`ILoggingLibrary` 项目的 `ILoggingLibrary.shared.cs` 文件包含接口定义，但不包含方法。为进行本演练，请按如下所示添加 `Log` 方法：

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Plugin.LoggingLibrary
{
    /// <summary>
    /// Interface for LoggingLibrary
    /// </summary>
    public interface ILoggingLibrary
    {
        /// <summary>
        /// Log a message
        /// </summary>
        void Log(string text);
    }
}
```

编写平台特定的代码

若要实现 `ILoggingLibrary` 接口及其方法的平台特定实现，请执行以下操作：

- 打开每个平台项目的 `LoggingLibraryImplementation.cs` (VS 2017) 或 `LoggingLibrary.<PLATFORM>.cs` (VS 2019) 文件并添加必要的代码。例如(使用 `Android` 平台项目):

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Plugin.LoggingLibrary
{
    /// <summary>
    /// Implementation for Feature
    /// </summary>
    public class LoggingLibraryImplementation : ILoggingLibrary
    {
        /// <summary>
        /// Log a message
        /// </summary>
        public void Log(string text)
        {
            throw new NotImplementedException("Called Log on Android");
        }
    }
}
```

- 在想要支持的每个平台的项目中重复此实现。
- 右键单击解决方案，选择“生成解决方案”，检查工作并生成接下来将要打包的项目。如果遇到关于缺少引用的错误，请右键单击解决方案，选择“还原 NuGet 包”，安装依赖项并重新生成。

NOTE

如果使用的是 Visual Studio 2019，则在选择“还原 NuGet 包”并尝试重新生成之前，需要将 `MSBuild.Sdk.Extras` 的版本更改为 `2.0.54` 中的 `LoggingLibrary.csproj`。只能通过以下方式访问此文件：首先右键单击该项目（在解决方案下方）并选择 `Unload Project`，然后右键单击卸载的项目并选择 `Edit LoggingLibrary.csproj`。

NOTE

若要为 iOS 生成，需要一台连接到 Visual Studio 的联网 Mac，如 [Introduction to Xamarin.iOS for Visual Studio](#) (Xamarin.iOS for Visual Studio 简介) 中所述。如果没有可用的 Mac，请清除配置管理器中的 iOS 项目（上面的步骤 3）。

创建并更新 .nuspec 文件

- 打开命令提示符，导航到 `LoggingLibrary` 文件下一级的 `.sln` 文件夹，然后运行 NuGet `spec` 命令，创建初始 `Package.nuspec` 文件：

```
nuget spec
```

- 将该文件重命名为 `LoggingLibrary.nuspec` 并在编辑器中打开。
- 将文件更新为与以下内容匹配，并将 `YOUR_NAME` 替换为适当的值。具体而言，`<id>` 值在 nuget.org 中必须是唯一的（请参阅[创建包](#)中所述的命名约定）。另请注意，还必须更新创建者和说明标记，否则在打包步骤中会出现错误。

```
<?xml version="1.0"?>
<package>
  <metadata>
    <id>LoggingLibrary.YOUR_NAME</id>
    <version>1.0.0</version>
    <title>Logginglibrary</title>
    <authors>YOUR_NAME</authors>
    <owners>YOUR_NAME</owners>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>Awesome application logging utility</description>
    <releaseNotes>First release</releaseNotes>
    <copyright>Copyright 2018</copyright>
    <tags>logger logging logs</tags>
  </metadata>
</package>
```

TIP

可使用 `-alpha`、`-beta` 或 `-rc` 为包版本添加后缀，将包标记为预发行版本。有关预发行版本的详细信息，请查阅[预发行版本](#)。

添加引用程序集

若要包含平台特定的引用程序集，请将以下内容添加到 `<files>` 的 `LoggingLibrary.nuspec` 元素，以适用于支持的平台：

```
<!-- Insert below <metadata> element -->
<files>
  <!-- Cross-platform reference assemblies -->
  <file src="Plugin.LoggingLibrary\bin\Release\Plugin.LoggingLibrary.dll"
target="lib\netstandard1.4\Plugin.LoggingLibrary.dll" />
  <file src="Plugin.LoggingLibrary\bin\Release\Plugin.LoggingLibrary.xml"
target="lib\netstandard1.4\Plugin.LoggingLibrary.xml" />
  <file src="Plugin.LoggingLibrary.Abstractions\bin\Release\Plugin.LoggingLibrary.Abstractions.dll"
target="lib\netstandard1.4\Plugin.LoggingLibrary.Abstractions.dll" />
  <file src="Plugin.LoggingLibrary.Abstractions\bin\Release\Plugin.LoggingLibrary.Abstractions.xml"
target="lib\netstandard1.4\Plugin.LoggingLibrary.Abstractions.xml" />

  <!-- iOS reference assemblies -->
  <file src="Plugin.LoggingLibrary.iOS\bin\Release\Plugin.LoggingLibrary.dll"
target="lib\Xamarin.iOS10\Plugin.LoggingLibrary.dll" />
  <file src="Plugin.LoggingLibrary.iOS\bin\Release\Plugin.LoggingLibrary.xml"
target="lib\Xamarin.iOS10\Plugin.LoggingLibrary.xml" />

  <!-- Android reference assemblies -->
  <file src="Plugin.LoggingLibrary.Android\bin\Release\Plugin.LoggingLibrary.dll"
target="lib\MonoAndroid10\Plugin.LoggingLibrary.dll" />
  <file src="Plugin.LoggingLibrary.Android\bin\Release\Plugin.LoggingLibrary.xml"
target="lib\MonoAndroid10\Plugin.LoggingLibrary.xml" />

  <!-- UWP reference assemblies -->
  <file src="Plugin.LoggingLibrary.UWP\bin\Release\Plugin.LoggingLibrary.dll"
target="lib\UAP10\Plugin.LoggingLibrary.dll" />
  <file src="Plugin.LoggingLibrary.UWP\bin\Release\Plugin.LoggingLibrary.xml"
target="lib\UAP10\Plugin.LoggingLibrary.xml" />
</files>
```

NOTE

若要缩短 DLL 和 XML 文件的名称，请右键单击任何给定项目，选择“库”选项卡，然后更改程序集名称。

添加依赖项

如果有特定的本机实现依赖项, 请使用带有 `<dependencies>` 元素的 `<group>` 元素来指定它们, 例如:

```
<!-- Insert within the <metadata> element -->
<dependencies>
    <group targetFramework="MonoAndroid">
        <!--MonoAndroid dependencies go here-->
    </group>
    <group targetFramework="Xamarin.iOS10">
        <!--Xamarin.iOS10 dependencies go here-->
    </group>
    <group targetFramework="uap">
        <!--uap dependencies go here-->
    </group>
</dependencies>
```

例如, 以下代码用于将 iTextSharp 设置为 UAP 目标的依赖项:

```
<dependencies>
    <group targetFramework="uap">
        <dependency id="iTextSharp" version="5.5.9" />
    </group>
</dependencies>
```

最终 .nuspec

现在, 最终 `.nuspec` 文件应该如下所示, 其中应再次将 YOUR_NAME 替换为适当的值:

```
<?xml version="1.0"?>
<package>
  <metadata>
    <id>LoggingLibrary.YOUR_NAME</id>
    <version>1.0.0</version>
    <title>LoggingLibrary</title>
    <authors>YOUR_NAME</authors>
    <owners>YOUR_NAME</owners>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>Awesome application logging utility</description>
    <releaseNotes>First release</releaseNotes>
    <copyright>Copyright 2018</copyright>
    <tags>logger logging logs</tags>
    <dependencies>
      <group targetFramework="MonoAndroid">
        <!--MonoAndroid dependencies go here-->
      </group>
      <group targetFramework="Xamarin.iOS10">
        <!--Xamarin.iOS10 dependencies go here-->
      </group>
      <group targetFramework="uap">
        <dependency id="iTextSharp" version="5.5.9" />
      </group>
    </dependencies>
  </metadata>
  <files>
    <!-- Cross-platform reference assemblies -->
    <file src="Plugin.LoggingLibrary\bin\Release\Plugin.LoggingLibrary.dll"
target="lib\netstandard1.4\Plugin.LoggingLibrary.dll" />
    <file src="Plugin.LoggingLibrary\bin\Release\Plugin.LoggingLibrary.xml"
target="lib\netstandard1.4\Plugin.LoggingLibrary.xml" />
    <file src="Plugin.LoggingLibrary.Abstractions\bin\Release\Plugin.LoggingLibrary.Abstractions.dll"
target="lib\netstandard1.4\Plugin.LoggingLibrary.Abstractions.dll" />
    <file src="Plugin.LoggingLibrary.Abstractions\bin\Release\Plugin.LoggingLibrary.Abstractions.xml"
target="lib\netstandard1.4\Plugin.LoggingLibrary.Abstractions.xml" />

    <!-- iOS reference assemblies -->
    <file src="Plugin.LoggingLibrary.iOS\bin\Release\Plugin.LoggingLibrary.dll"
target="lib\Xamarin.iOS10\Plugin.LoggingLibrary.dll" />
    <file src="Plugin.LoggingLibrary.iOS\bin\Release\Plugin.LoggingLibrary.xml"
target="lib\Xamarin.iOS10\Plugin.LoggingLibrary.xml" />

    <!-- Android reference assemblies -->
    <file src="Plugin.LoggingLibrary.Android\bin\Release\Plugin.LoggingLibrary.dll"
target="lib\MonoAndroid10\Plugin.LoggingLibrary.dll" />
    <file src="Plugin.LoggingLibrary.Android\bin\Release\Plugin.LoggingLibrary.xml"
target="lib\MonoAndroid10\Plugin.LoggingLibrary.xml" />

    <!-- UWP reference assemblies -->
    <file src="Plugin.LoggingLibrary.UWP\bin\Release\Plugin.LoggingLibrary.dll"
target="lib\UAP10\Plugin.LoggingLibrary.dll" />
    <file src="Plugin.LoggingLibrary.UWP\bin\Release\Plugin.LoggingLibrary.xml"
target="lib\UAP10\Plugin.LoggingLibrary.xml" />
  </files>
</package>
```

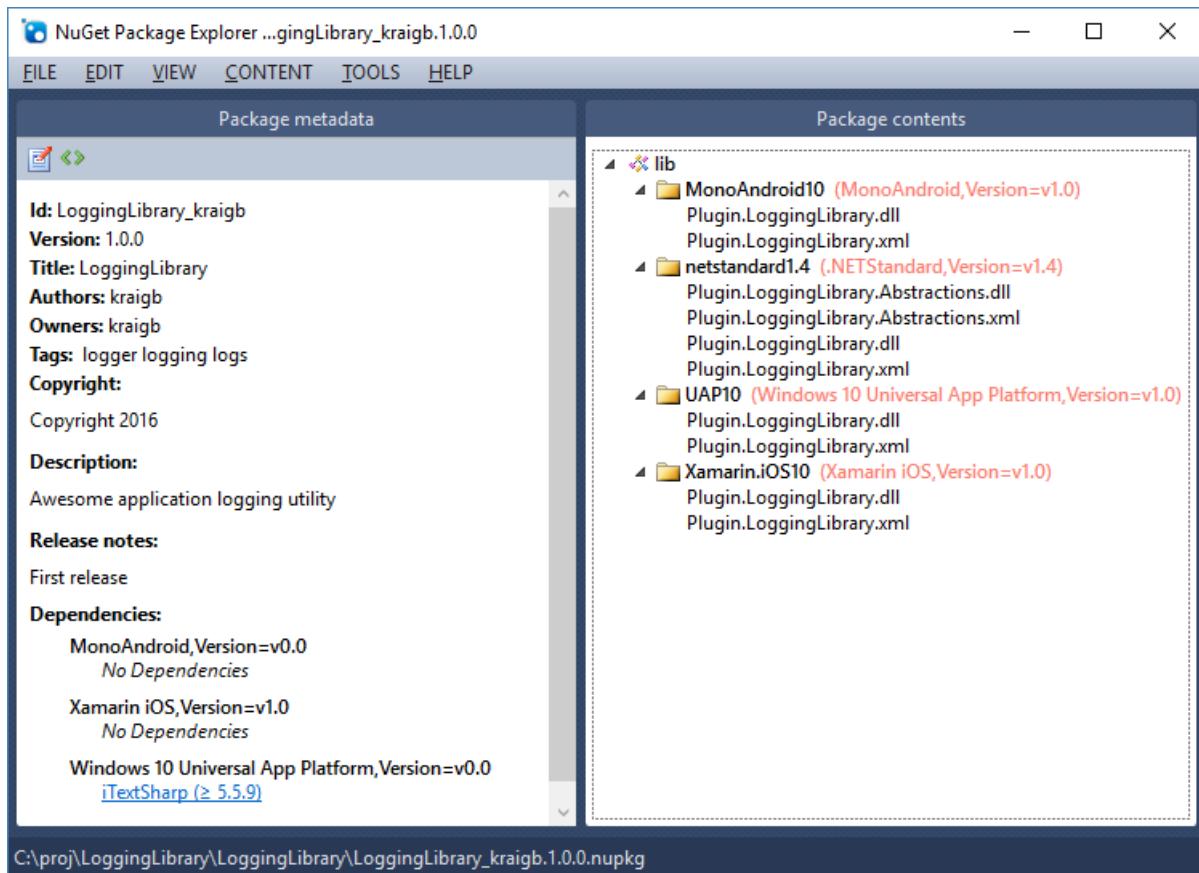
打包组件

如果已完成的 `.nuspec` 引用需要包含在包中的所有文件，便可运行 `pack` 命令：

```
nuget pack LoggingLibrary.nuspec
```

这将生成 `LoggingLibrary.YOUR_NAME.1.0.0.nupkg`。在类似 [NuGet 包资源管理器](#) 的工具中打开此文件并展开所有节

点，即可看到以下内容：



TIP

.nupkg 文件实际上是具有其他扩展名的 ZIP 文件。然后，也可通过将 .nupkg 更改为 .zip 来检查包内容，但将包上传到 nuget.org 之前，请记得恢复扩展名。

若要使你的包可供其他开发人员使用，请按照[发布包](#)中的说明进行操作。

相关主题

- [Nuspec 引用](#)
- [符号包](#)
- [包版本控制](#)
- [支持多个 .NET Framework 版本](#)
- [将 MSBuild 属性和目标包含到包中](#)
- [创建本地化包](#)

创建包含 COM 互操作程序集的 NuGet 包

2020/4/8 • [Edit Online](#)

包含 COM 互操作程序集的包必须包含合适的[目标文件](#), 使正确的 `EmbedInteropTypes` 元数据使用 `PackageReference` 格式添加到项目。默认情况下, 使用 `PackageReference` 时, `EmbedInteropTypes` 元数据对所有程序集一直为 `false`, 所以目标文件显式添加此元数据。为了避免冲突, 目标名称必须是唯一的; 理想情况下, 使用包名称和嵌入程序集的组合, 使用此值替换下例中的 `{InteropAssemblyName}`。(有关示例, 另请参阅 [NuGet.Samples.Interop](#)。)

```
<Target Name="Embedding**AssemblyName**From**PackageId**" AfterTargets="ResolveReferences"
BeforeTargets="FindReferenceAssembliesForReferences">
  <ItemGroup>
    <ReferencePath Condition=" '%(FileName)' == '{InteropAssemblyName}' AND '%(ReferencePath.NuGetPackageId)'
== '$(MSBuildThisFileName)' ">
      <EmbedInteropTypes>true</EmbedInteropTypes>
    </ReferencePath>
  </ItemGroup>
</Target>
```

请注意, 在使用 `packages.config` 管理格式时, 将引用从包添加到程序集会导致 NuGet 和 Visual Studio 检查 COM 互操作程序集并在项目文件中将 `EmbedInteropTypes` 设为 `true`。在这种情况下, 目标被替代。

此外, 默认情况下, [生成资产不以可传递的方式流动](#)。当此处所述的创作的包作为可传递的依赖项从项目拉取到项目引用时, 它们会以不同的方式工作。包使用者通过将 `PrivateAssets` 默认值修改为不包含生成使其流动。

对 NuGet 包进行签名

2020/4/8 • [Edit Online](#)

已签名的包允许进行内容完整性验证检查，可有效防止内容被篡改。此外，包的签名还可作为包的实际来源的单一可信来源，增强了对包使用者的身份验证。本指南假定你已[创建一个包](#)。

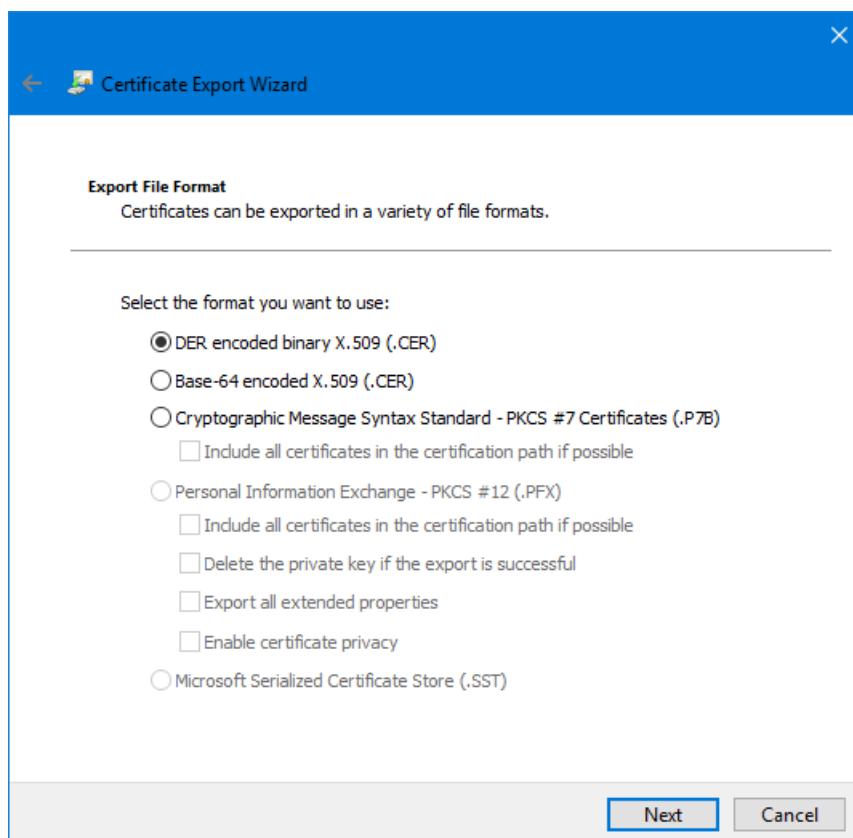
获取代码签名证书

可从以下公共证书颁发机构获取有效的证书：[Symantec](#)、[DigiCert](#)、[Go Daddy](#)、[Global Sign](#)、[Comodo](#)、[Certum](#)等。可从<http://aka.ms/trustcertpartners> 获取 Windows 信任的证书颁发机构完整列表。

可使用自颁发证书进行测试。但 NuGet.org 不接受使用自颁发证书签名的包。详细了解如何[创建测试证书](#)

导出证书文件

- 使用证书导出向导，可将现有的证书导出为二进制 DER 格式。



- 此外，还可使用 [Export-Certificate PowerShell 命令](#) 导出证书。

对包进行签名

NOTE

需要 nuget.exe 4.6.0 或更高版本。即将推出 dotnet.exe 支持 - [#7939](#)

可使用 [nuget sign](#) 对包进行签名：

```
nuget sign MyPackage.nupkg -CertificatePath <PathToTheCertificate> -Timestamper <TimestampServiceURL>
```

TIP

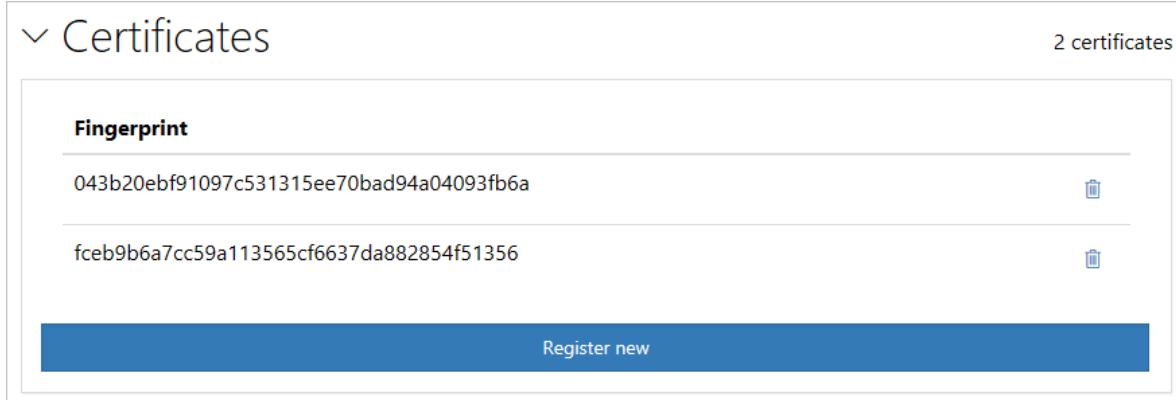
证书提供程序通常还提供时间戳服务器 URL，可用于如上所示的 `Timestamper` 可选参数。请参考提供程序文档和/或该服务 URL 的支持。

- 可使用证书存储中可用的证书或使用来自文件的证书。请参阅 [CLI 参考](#)，了解 `nuget sign`。
- 已签名包应包含时间戳，用于确保签名证书过期时签名仍有效。否则签名操作将引发一个警告。
- 使用 `nuget verify` 可查看给定包的签名详细信息。

使用 NuGet.org 注册证书

要发布已签名的包，必须先使用 NuGet.org 注册证书。需要将证书设置为二进制 DER 格式的 `.cer` 文件。

1. [登录](#) 到 NuGet.org。
2. 转到 `Account settings`（如果希望使用组织帐户注册证书，则转到 `Manage Organization` > `Edit Organization`）。
3. 展开 `Certificates` 部分，并选择 `Register new`。
4. 浏览并选择前面导出的证书文件。



注意

- 一个用户可以提交多个证书并且多个用户可以注册同一个证书。
- 用户注册证书之后，所有未来的包提交都必须 使用其中一个证书进行签名。请参阅[管理 NuGet.org 上的包的签名要求](#)
- 用户还可以从帐户中删除已注册的证书。删除证书后，使用该证书签名的新包将在提交时失败。现有包不会受到影响。

发布包

现在即可将包发布到 NuGet.org。请参阅[发布包](#)。

创建测试证书

可使用自颁发证书进行测试。要创建自颁发证书，请使用 `New-SelfSignedCertificate` PowerShell 命令。

```
New-SelfSignedCertificate -Subject "CN=NuGet Test Developer, OU=Use for testing purposes ONLY" ` 
    -FriendlyName "NuGetTestDeveloper" ` 
    -Type CodeSigning ` 
    -KeyUsage DigitalSignature ` 
    -KeyLength 2048 ` 
    -KeyAlgorithm RSA ` 
    -HashAlgorithm SHA256 ` 
    -Provider "Microsoft Enhanced RSA and AES Cryptographic Provider" ` 
    -CertStoreLocation "Cert:\CurrentUser\My"
```

此命令将在当前用户的个人证书存储中创建可用的测试证书。可通过运行 `certmgr.msc` 打开证书存储，查看新创建的证书。

WARNING

NuGet.org 不接受使用自颁发证书签名的包。

管理 NuGet.org 上的包的签名要求

1. 登录到 NuGet.org。

2. 转到 `Manage Packages`

Published Packages						2 packages / 10 downloads
Package ID	Owners	Signing Owner	Downloads	Latest Version		
Contoso.Controls	Contoso, JohnSmith	Any	10	1.0.0		
Contoso.Extensions.Data	Contoso	Contoso (1 certificate)	0	0.1.2		

- 如果你是包的唯一所有者，那么你就是所要求的签名者，即你可使用任何已注册的证书对包进行签名，然后将其发布到 NuGet.org。
- 如果包具有多个所有者，默认情况下，可以使用“任何”所有者的证书对包进行签名。作为包的共同所有者，你可将“任何”重写为你自己或任何其他共同所有者，作为所需的签名者。如果你让没有注册任何证书的人成为所有者，则将允许未签名的包。
- 同样，如果已选中某个包的默认“任何”选项，在该包中，有一个所有者已注册证书，而另一个所有者未注册任何证书，则 NuGet.org 接受已签名的包(由其中一个所有者注册签名)或接受未签名的包(因为其中一个所有者未注册任何证书)。

相关文章

- [管理包信任边界](#)
- [签名包引用](#)

已签名的包

2020/3/19 • [Edit Online](#)

NuGet 4.6.0 + 和 Visual Studio 2017 版本 15.6 及更高版本

NuGet 包可以包含一个数字签名，该签名提供针对篡改内容的保护。此签名是通过 x.509 证书生成的，该证书还将授权校样添加到包的实际来源。

签名包提供了最强大的端到端验证。NuGet 签名有两种不同的类型：

- **作者签名**。作者签名可保证自作者对包进行签名后包未修改，无论包传送到哪个存储库或传输方法。此外，创作签名包还为 nuget.org 发布管道提供额外的身份验证机制，因为必须提前注册签名证书。有关详细信息，请参阅[注册证书](#)。
- **存储库签名**。存储库签名是存储库中的所有包提供完整性保证，无论它们是否为作者签名，即使这些包是从其签名的原始存储库以外的位置获取的。

有关创建作者签名包的详细信息，请参阅[签署包和nuget 签名命令](#)。

IMPORTANT

目前仅在 Windows 上使用 nuget.exe 时才支持包签名。目前仅在使用 nuget.exe 或 Windows 上的 Visual Studio 时，才支持验证已签名的包。

证书要求

包签名需要一个代码签名证书，该证书是一种特殊类型的证书，适用于 `id-kp-codeSigning` 目的 [[RFC 5280 部分 4.2.1.12](#)]。此外，证书的 RSA 公钥长度必须为 2048 位或更高。

时间戳要求

签名包应包含 RFC 3161 时间戳，以确保签名有效期超出包签名证书的有效期。用于签署时间戳的证书必须对 `id-kp-timeStamping` 目的有效 [[RFC 5280 部分 4.2.1.12](#)]。此外，证书的 RSA 公钥长度必须为 2048 位或更高。

可以在[包签名技术规范](#)(GitHub)中找到其他技术详细信息。

NuGet.org 上的签名要求

nuget.org 对接受签名的包有其他要求：

- 主签名必须是作者签名。
- 主签名必须有一个有效的时间戳。
- 作者签名及其时间戳签名的 x.509 证书：
 - 必须具有 RSA 公钥 2048 或更高版本。
 - 在 nuget.org 上的包验证时，必须在其每个当前 UTC 时间的有效期内。
 - 必须链接到 Windows 上默认受信任的受信任根证书颁发机构。拒绝了用自行颁发的证书签名的包。
 - 的用途必须有效：
 - 作者签名证书必须有效，以便进行代码签名。
 - 对于时间戳，时间戳证书必须有效。
 - 不能在签名时撤销。(这可能不会在提交时 knowable，因此 nuget.org 会定期复查列为吊销状态)。

相关文章

- [对 NuGet 包进行签名](#)
- [管理包信任边界](#)

发布包

2020/4/8 • [Edit Online](#)

创建程序包并获得 `.nupkg` 文件后，即可轻松以公开或私密方式将其提供给其他开发人员：

- 根据本文中的介绍，可通过 [nuget.org](#) 将公共包全局提供给所有开发人员（需要 NuGet 4.1.0+）。
- 通过以下方式可以仅向团队或组织提供专用包：在文件共享、专用 NuGet 服务器、[Azure Artifacts](#) 或第三方存储库（如 myget、ProGet、Nexus 存储库和 Artifactory）上承载专用包。有关其他详细信息，请参阅[承载包概述](#)。

本文介绍如何发布到 nuget.org；有关发布到 Azure Artifacts 的信息，请参阅[包管理](#)。

发布到 nuget.org

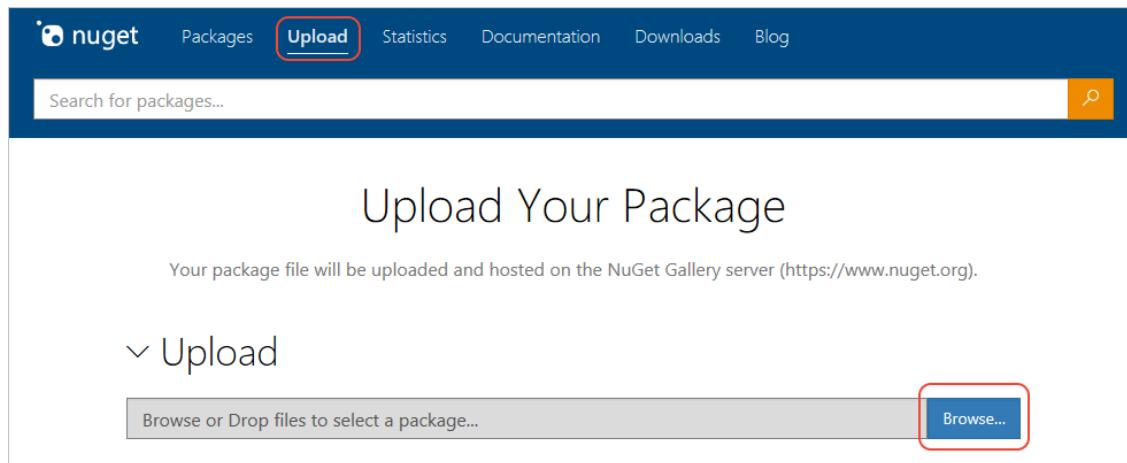
对于 nuget.org，必须使用 Microsoft 帐户进行登录，将需要在 nuget.org 中注册此帐户。此外可以使用较旧版本的门户创建的 nuget.org 帐户进行登录。



接下来，可根据以下各节中的介绍，通过 nuget.org Web 门户上传包、从命令行（需要 `nuget.exe` 4.1.0+）将包推送到 nuget.org 或通过 Azure DevOps Services 在 CI/CD 过程中发布包。

Web 门户：使用 nuget.org 上的“上传包”选项卡

- 选择 nuget.org 顶部菜单中的“上传”，并浏览到包位置。



- nuget.org 告知包名称是否可用。如果无法使用，则更改项目中的包标识符、重新生成，并重试上传。
- 如果包名称可用，nuget.org 将打开“验证”部分，可以在其中查看包清单中的元数据。若要更改任何元数据，请编辑项目（项目文件或 `.nuspec` 文件）、重新生成、重新创建包，然后再次上传。
- 在“导入文档”下，可以粘贴 Markdown、将 URL 指向文档，或上传文档文件。
- 当所有信息准备就绪后，选择“提交”按钮

命令行

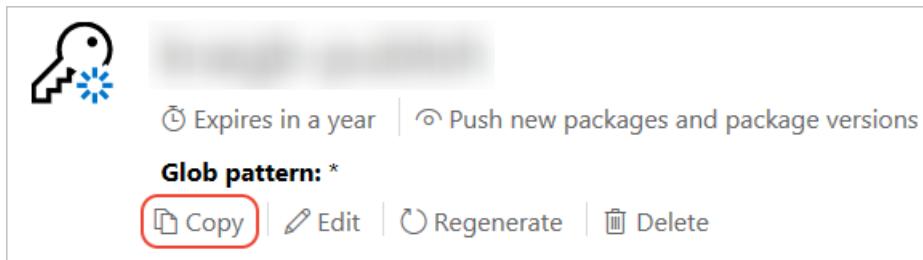
若要将包推送到 nuget.org，必须使用实现所需 [NuGet 协议](#) 的 `nuget.exe v4.1.0` 或更高版本。还需要在 nuget.org 上创建的 API 密钥。

创建 API 密钥

- 登录你的 nuget.org 帐户，或创建一个帐户（如果你还没有帐户）。

有关创建帐户的详细信息，请参阅[个人帐户](#)。

2. 选择用户名(在右上角)，然后选择“API 密钥”。
3. 选择“创建”，提供密钥名称，选择“选择范围”>“推送”。输入“Glob 模式” *，然后选择“创建”。(请参阅下面有关范围的详细信息。)
4. 创建密钥后，选择“复制”，检索需要在 CLI 中使用的访问密钥：



5. **重要事项**: 将你的密钥保存在安全位置，因为以后无法再次复制密钥。如果返回到 API 密钥页，则需要重新生成密钥以对其进行复制。如果不再希望通过 CLI 推送包，还可以删除 API 密钥。

范围允许创建针对不同用途的单独 API 密钥。每个密钥都有其过期时间，并且可以将范围限定为特定包(或 glob 模式)。每个密钥还将范围限定为特定操作:新包和更新推送、仅更新推送，或者从列表中删除。通过范围限定，可以为管理组织不同包的不同人员创建 API 密钥，这样他们就只有所需的权限。有关详细信息，请参阅[范围内的 API 密钥](#)。

用 `dotnet nuget push` 发布

1. 更改到包含 `.nupkg` 文件的文件夹。
2. 运行以下命令，指定包名称(唯一包 ID)并使用你的 API 密钥替换密钥值：

```
dotnet nuget push AppLogger.1.0.0.nupkg -k qz2jga8pl3dvn2akksyquwcs9ygggg4exypy3bhxy6w6x6 -s https://api.nuget.org/v3/index.json
```

3. dotnet 会显示发布过程的结果：

```
info : Pushing AppLogger.1.0.0.nupkg to 'https://www.nuget.org/api/v2/package'...
info :  PUT https://www.nuget.org/api/v2/package/
info :  Created https://www.nuget.org/api/v2/package/ 12620ms
info : Your package was pushed.
```

请参阅 [dotnet nuget push](#)。

用 `nuget push` 发布

1. 在命令提示符处运行以下命令，将 `<your_API_key>` 替换为从 nuget.org 获取的密钥：

```
nuget setApiKey <your_API_key>
```

此命令将 API 密钥存储在 NuGet 配置中，以便无需在同一台计算机上再次重复此步骤。

NOTE

API 密钥不用于向专用源进行身份验证。请参考 [nuget sources](#) 命令来管理用于向源进行身份验证的凭据。

API 密钥可以从单个 NuGet 服务器获取。要为 nuget.org 创建和管理 APIKey，请参阅 [publish-api-key](#)

2. 使用以下命令将包推送到 NuGet 库：

```
nuget push YourPackage.nupkg -Source https://api.nuget.org/v3/index.json
```

发布已签名的包

若要提交已签名的包，必须首先注册用于签名包的证书。

WARNING

nuget.org 会拒绝不满足签名包要求的包。

包验证和编制索引

推送到 nuget.org 的包会进行多项验证，如病毒检查。（定期扫描 nuget.org 上的所有包。）

包通过所有验证检查后，对包编制索引并将其显示在搜索结果中可能需要一些时间。编制索引完成后，你会收到一封确认已成功发布包的电子邮件。如果包未通过验证检查，将更新包详细信息页面以显示相关错误，同时你也会收到包含相关通知的电子邮件。

包验证和编制索引所需的时间通常不超过 15 分钟。如果发布包所用时间超出预期，请访问 [status.nuget.org](#) 检查 nuget.org 是否遇到任何中断。如果所有系统均正常运行，但一个小时之内还未成功发布包，请登录 nuget.org 并使用包页面上的“联系支持人员”链接与我们联系。

若要查看包状态，请选择 nuget.org 上帐户名称下的“管理包”。完成验证时，会收到确认电子邮件。

请注意，对包编制索引并将其显示在搜索结果中供其他人查找可能需要一些时间。在此期间，包页面上会出现以下消息：

⚠ This package has not been published yet. It will appear in search results and will be available for install/restore after both validation and indexing are complete. Package validation and indexing may take up to an hour. [Read more.](#)

Azure DevOps Services (CI/CD)

如果在持续集成/部署过程中使用 Azure DevOps Services 将包推送到 nuget.org，必须在 NuGet 任务中使用 `nuget.exe` 4.1 或更高版本。要了解详细信息，请参阅 [Using the latest NuGet in your build](#)（在生成中使用最新 NuGet）（Microsoft DevOps 博客）。

在 nuget.org 上管理包所有者

尽管每个 NuGet 包的 `.nuspec` 文件定义了该包的创建者，但 nuget.org 库不使用该元数据定义所有权。相反，nuget.org 将初始所有权分配给发布包的人员。他们通常是通过 nuget.org UI 上传包的已登录用户，或其 API 密钥与 `nuget SetApiKey` 或 `nuget push` 配合使用的用户。

所有包所有者均拥有包的完全权限，包括添加和删除其他所有者以及发布更新。

要更改包的所有权，请执行以下操作：

1. 使用包的当前所有者的帐户登录 nuget.org。
2. 选择帐户名称，选择“管理包”，然后展开“已发布的包”。
3. 选择要管理的包，然后在右侧选择“管理所有者”。

此时，你具有多个选择：

1. 删除“当前所有者”下列出的任何所有者。
2. 在“添加所有者”下添加所有者，方法是通过输入其用户名和一条消息，然后选择“添加”。此操作会向该新共有者发送包含确认链接的电子邮件。确认后，此人拥有添加和删除所有者的完全权限。（确认之前，“当前所有者”部分指示此人的状态为“待审批”。）

3. 要转让所有权(如变更所有权或在错误的帐户下发布包时), 请添加新所有者, 他们确认所有权后, 即可将你从列表中删除。

要将所有权分配给公司或组, 请使用转发到适当团队成员的电子邮件别名创建 nuget.org 帐户。例如, 各种 Microsoft ASP.NET 包由 [microsoft](#) 和 [aspnet](#) 帐户共同拥有, 这两个帐户就是此类别名。

恢复包的所有权

有时, 包所有者可能不太活跃。例如, 原始所有者可能已离开生产该包的公司, nuget.org 凭据丢失, 或库中的早期 bug 导致包不具有所有者。

如果你是包的合法所有者且需要重新获得所有权, 请使用 nuget.org 上的[联系表单](#)向 NuGet 团队解释相关情况。然后, 我们按照流程验证你对包的所有权, 包括通过包的项目 URL、Twitter、电子邮件或其他方式尝试找出所有者。但如果所有其他方式均失败, 我们会向你发送成为所有者的邀请。

范围内的 API 密钥

2020/4/8 • [Edit Online](#)

为了使 NuGet 成为一个更安全的包分发环境，可以通过添加范围来控制 API 密钥。

借助向 API 密钥提供范围的功能可更好地控制 API。可以：

- 创建多个范围内的 API 密钥，可用于具有不同到期时间表的不同包。
- 安全地获取 API 密钥。
- 编辑现有 API 密钥以更改包适用性。
- 刷新或删除现有 API 密钥，而不会妨碍使用其他密钥的操作。

为什么我们支持范围内的 API 密钥？

我们支持 API 密钥的范围是为了让你拥有更细粒度的权限。以前，NuGet 为一个帐户提供了一个 API 密钥，这种方法有几个缺点：

- **一个 API 密钥控制所有包。**若使用一个 API 密钥管理所有包，当多个开发人员处理不同包以及共享发布者帐户时，很难安全地共享密钥。
- **要么所有权限，要么无任何权限。**有权访问 API 密钥的任何人都拥有包的所有权限(发布、推送和取消列表)。在具有多个团队的环境中，这通常是不可取的。
- **单个故障点。**单个 API 密钥也意味着单个故障点。如果密钥已泄露，则与该帐户有关的所有包都可能存在危险。刷新 API 密钥是堵住漏洞，避免 CI/CD 工作流中断的唯一方法。此外，在某些情况下，你可能希望撤销某个人对 API 密钥的访问权限(例如，当员工离开组织时)。没有可以立即处理这种情况的有效方法。

使用范围内的 API 密钥，我们尝试解决这些问题，同时确保现有的工作流不会中断。

获取 API 密钥

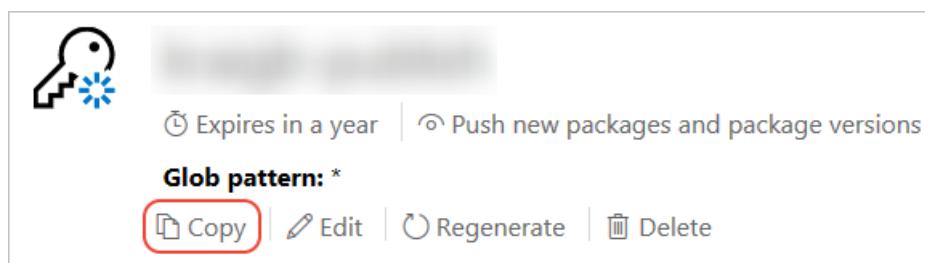
1. [登录你的 nuget.org 帐户](#)，或创建一个帐户（如果你还没有帐户）。

有关创建帐户的详细信息，请参阅[个人帐户](#)。

2. 选择用户名（在右上角），然后选择“API 密钥”。

3. 选择“创建”，提供密钥名称，选择“选择范围”>“推送”。输入“Glob 模式”*，然后选择“创建”。（请参阅下面有关范围的详细信息。）

4. 创建密钥后，选择“复制”，检索需要在 CLI 中使用的访问密钥：



5. **重要事项：**将你的密钥保存在安全位置，因为以后无法再次复制密钥。如果返回到 API 密钥页，则需要重新生成密钥以对其进行复制。如果不再希望通过 CLI 推送包，还可以删除 API 密钥。

范围允许创建针对不同用途的单独 API 密钥。每个密钥都有其过期时间，并且可以将范围限定为特定包（或 glob 模式）。每个密钥还将范围限定为特定操作：新包和更新推送、仅更新推送，或者从列表中删除。通过范围限定，

可以为管理组织不同包的不同人员创建 API 密钥，这样他们就只有所需的权限。有关详细信息，请参阅[范围内的 API 密钥](#)。

创建范围内的 API 密钥

可以根据需要创建多个 API 密钥。API 密钥可以应用于一个或多个包，具有授予特定权限的不同范围，并具有与之相关的到期日期。

在以下示例中，你有一个名为 `Contoso service CI` 的 API 密钥，可用于为特定 `Contoso.Service` 包推送包，并且有效期为 365 天。这是一个典型的情况，即同一组织中的不同团队处理不同的包，并且团队成员获得的密钥仅向他们授予正在处理的包的权限。到期用作一种防止密钥过时或遗忘的机制。

Manage API Keys

API Keys

Key name: Contoso service CI

Expires in: 365 days

Select scopes:

- Push
 - Push new packages and package versions
 - Push only new package versions
- Unlist package

Select packages:

To select which packages to associate with a key use a glob pattern, select individual packages, or both. Example [glob patterns](#).

Glob pattern: Example: The wildcard '*' will select all packages.

Available packages:

- Contoso.Service.API
- Contoso.Service.Extensions
- Contoso.Service.Framework
- Contoso.Service.Integration
- Contoso.UI.Extensions
- Contoso.UI.Framework
- Fabrikam.Service.API
- Fabrikam.Service.Extensions
- Fabrikam.UI.Extensions
- Fabrikam.UI.Framework

2/10 selected

Add key

使用 glob 模式

如果处理多个包并且需要管理大量包，则可以选择使用通配模式同时选择多个包。例如，如果希望为 ID 以 `Fabrikam.Service` 开头的所有包的键授予特定的范围，则可以在“Glob 模式”文本框中指定 `fabrikam.service.*`。

Select packages

To select which packages to associate with a key use a glob pattern, select individual packages, or both. [Example glob patterns](#).

Glob pattern

A glob pattern allows you to replace any sequence of characters with '*'.

Available packages

- Contoso.Service.API
- Contoso.Service.Extensions
- Contoso.Service.Framework
- Contoso.Service.Integration
- Contoso.UI.Extensions
- Contoso.UI.Framework
- Fabrikam.Service.API
- Fabrikam.Service.Extensions
- Fabrikam.UI.Extensions
- Fabrikam.UI.Framework

2/10 selected

Add key

使用 glob 模式确定 API 密钥权限也适用于与 glob 模式匹配的新包。例如，如果尝试推送名为 `Fabrikam.Service.Framework` 的新包，则可以使用之前创建的密钥执行此操作，因为该包与 glob 模式 `fabrikam.service.*` 匹配。

安全地获取 API 密钥

为了安全起见，屏幕上永远不会显示新创建的密钥，只能使用“复制”按钮。同样，刷新页面后无法访问该密钥。

Manage API Keys

API Keys

New API key

Keys

Contoso service CI   

Expires: in a year
Scopes: Push new packages and package versions
Packages: Contoso.Service.API, Contoso.Service.Extensions

Full access API key   

Expires: in a year
Scopes: All
Packages: All

编辑现有 API 密钥

你还可能希望在不更改密钥本身的情况下更新密钥权限和范围。如果有一个针对单个包的具有特定范围的密钥，则可以选择向一个或多个其他包应用同一范围。

Keys

Contoso service CI   

Expires: in a year
Scopes: Push new packages and package versions
Packages: Contoso.Service.API, Contoso.Service.Extensions

刷新或删除现有 API 密钥

帐户所有者可以选择刷新密钥，在这种情况下（对包的）权限、范围和到期日期保持不变，但是会发出一个新密钥，使旧密钥无法使用。这有助于管理过时密钥，或者有助于可能存在 API 密钥泄漏的情况。

Keys



Contoso service CI

Expires: in a year

Scopes: Push new packages and package versions

Packages: Contoso.Service.API, Contoso.Service.Extensions



如果不再需要这些密钥，也可以选择删除这些密钥。删除密钥将移除密钥并使其无法使用。

常见问题解答

我的旧(旧版)API 密钥会怎么样？

旧(旧版)API 密钥可继续使用，使用时间由你决定。但是，如果这些密钥已超过 365 天未被用于推送包，则将停用。有关详细信息，请参阅博客文章[更改即将到期的 API 密钥](#)。无法再刷新此密钥。需要删除旧密钥并改为创建新的范围内的密钥。

NOTE

此密钥具有所有包的所有权限，并且永不过期。应该考虑删除此密钥并创建具有范围内权限和明确期限的新密钥。

我可以创建多少个 API 密钥？

可以创建的 API 密钥数量没有限制。但我们建议保留易于管理的个数，这样你就不会获得许多过时密钥，而不知道密钥在何处以及谁在使用它们。

我是否可以删除旧 API 密钥或立即停止使用？

是的。你可以并且应该删除你的旧 API 密钥。

我是否可以找回错误删除的 API 密钥？

不是。删除后，你只能创建新密钥。意外删除的密钥无法恢复。

API 密钥刷新时是否可继续使用旧 API 密钥？

不是。刷新密钥后，将生成一个与旧密钥具有相同范围、权限和期限的新密钥。旧密钥不复存在。

我可以为现有 API 密钥授予更多权限吗？

你无法修改范围，但可以编辑适用的包列表。

如何知道密钥是已过期还是即将过期？

如果任何密钥过期，我们会通过页面顶部的警告消息通知你。我们还会在密钥到期前十天向帐户持有者发送警告电子邮件，以便能够提前做好准备。

承载自己的 NuGet 源

2020/4/8 • [Edit Online](#)

可能希望将包仅发布到有限受众(例如, 组织或工作组), 而不是将其公开发布。此外, 一些公司可能希望限制其开发人员可以使用的第三方库, 使他们从有限包源(而不是 nuget.org)中提取内容。

对于所有此类目的, NuGet 支持通过以下方式设置专用包源:

- **本地源**: 包只需放置在合适的网络文件共享中, 理想情况下使用 `nuget init` 和 `nuget add` 创建分层文件夹结构 (NuGet 3.3+)。有关详细信息, 请参阅 [本地源](#)。
- **NuGet.Server**: 通过本地 HTTP 服务器提供包。有关详细信息, 请参阅 [NuGet.Server](#)。
- **NuGet 库**: 在使用 [NuGet 库项目](#) (github.com) 的 Internet 服务器上承载包。NuGet 库提供用户管理和功能, 如丰富的 Web UI, 它允许从浏览器中搜索和浏览包, 这与 nuget.org 相似。

另外还有其他几种 NuGet 宿主产品支持远程专用源, 例如 [Azure Artifacts](#) 和 [GitHub 包注册表](#)。下面列出了此类产品:

- JFrog 的 [Artifactory](#)。
- [Azure Artifacts](#) 也可通过 Team Foundation Server 2017 以及更高版本获得。
- [BaGet](#) 构建与 ASP.NET Core 基础上的 NuGet V3 服务器的开源实现
- [Cloudsmith](#): 一种完全托管的包管理 SaaS
- [GitHub 包注册表](#)
- [LiGet](#): 在 docker 中 kestrel 上运行的 NuGet V2 服务器的开放源代码实现
- [MyGet](#)
- Sonatype 中的 [Nexus 存储库 OSS](#)。
- [NuGet 服务器\(开放源代码\)](#), 与 Inedo 的 NuGet 服务器相似的开放源代码实现
- [NuGet 服务器](#), Inedo 的社区项目
- Inedo 的 [ProGet](#)
- [Sleet](#)(开放源代码 NuGet V3 静态源生成器)
- JetBrains 的 [TeamCity](#)。

无论包是什么样的承载方式, 均可将其添加到 `NuGet.Config` 中的可用源列表以便访问。此操作可在 Visual Studio 中执行(如[包源](#)中所述), 或使用 `nuget sources` 从命令行执行操作。源的路径可以是本地文件夹路径名、网络名称或 URL。

NuGet.Server

2020/4/8 • [Edit Online](#)

NuGet.Server 是由 .NET Foundation 提供的包，其创建的 ASP.NET 应用程序可在运行 IIS 的任何服务器上托管包源。简而言之，NuGet.Server 通过 HTTP(尤其是 OData)在服务器上提供文件夹。其设置方法十分简单，最适用于简单的方案。

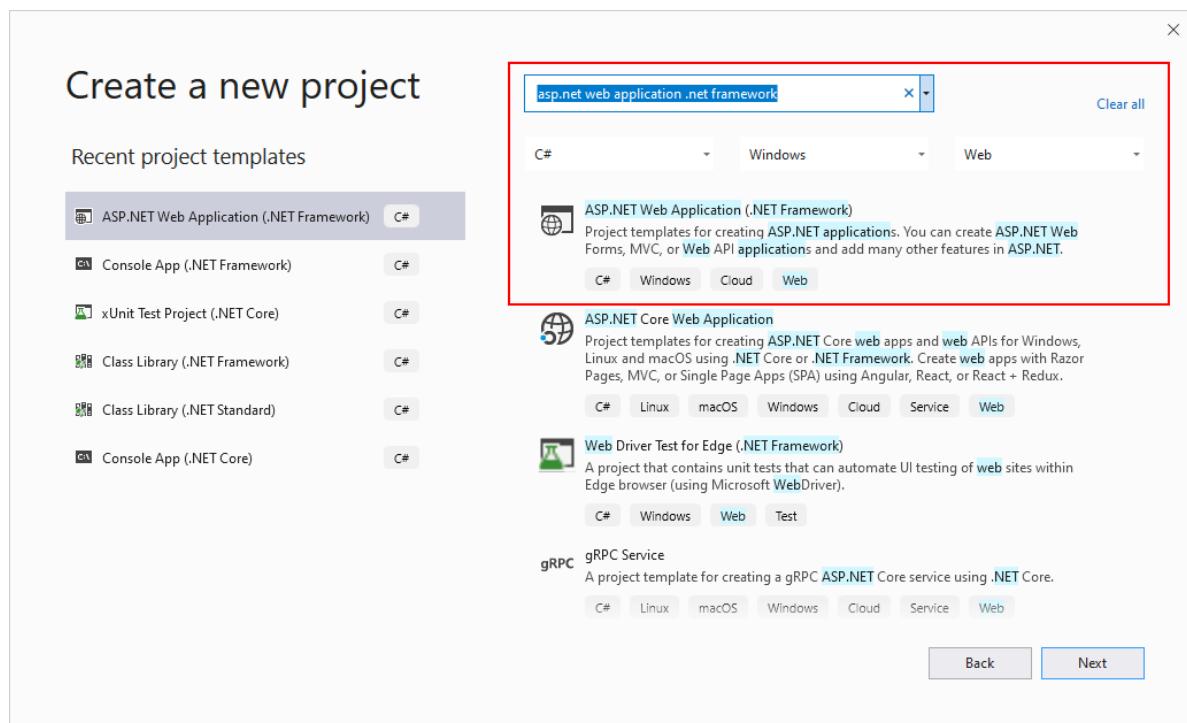
1. 在 Visual Studio 中创建空的 ASP.NET Web 应用程序并向其添加 NuGet.Server 包。
2. 配置应用程序中的 `Packages` 文件夹并添加包。
3. 将应用程序部署到适合的客户端。

以下各节使用 C# 详细演练此过程。

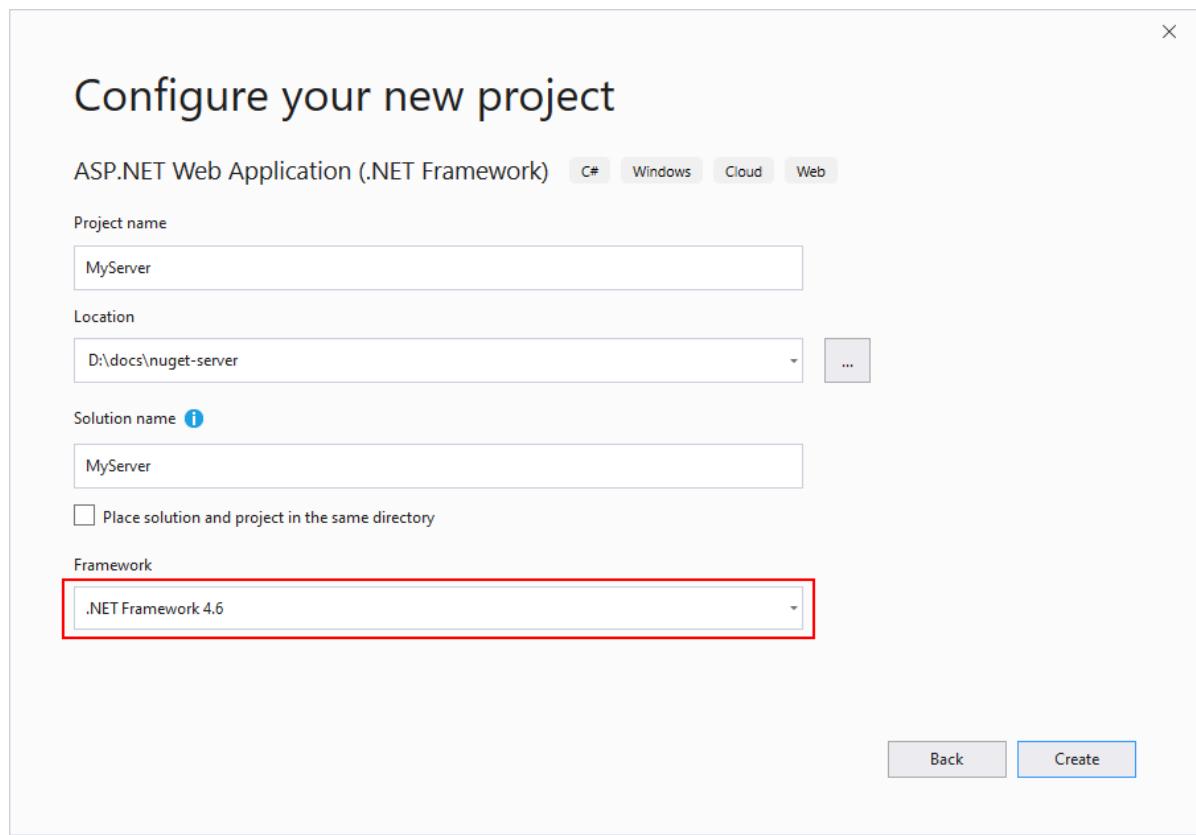
如果对 NuGet.Server 有进一步的疑问，请在 <https://github.com/nuget/NuGetGallery/issues> 上创建问题。

使用 NuGet.Server 创建和部署 ASP.NET Web 应用程序

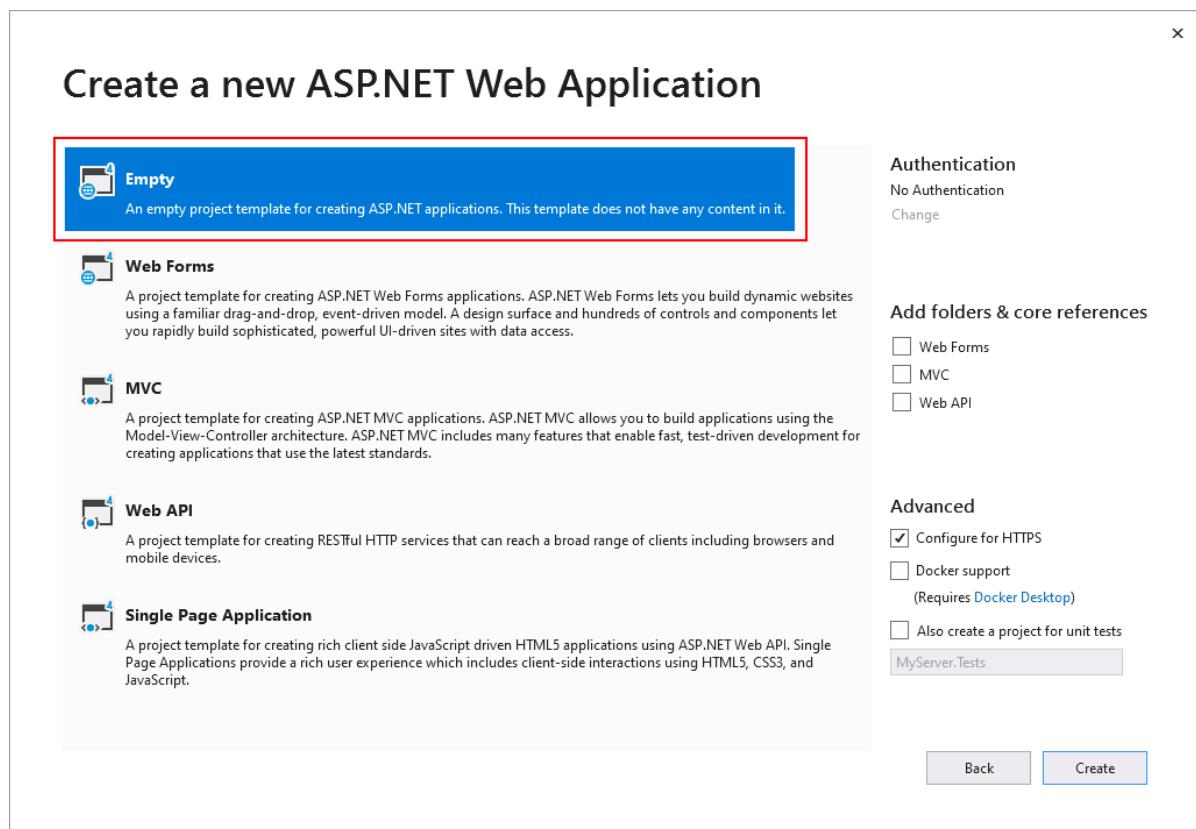
1. 在 Visual Studio 中，选择“文件”>“新建”>“项目”，搜索“ASP.NET Web 应用程序 (.NET Framework)”，选择适用于 C# 的匹配模板。



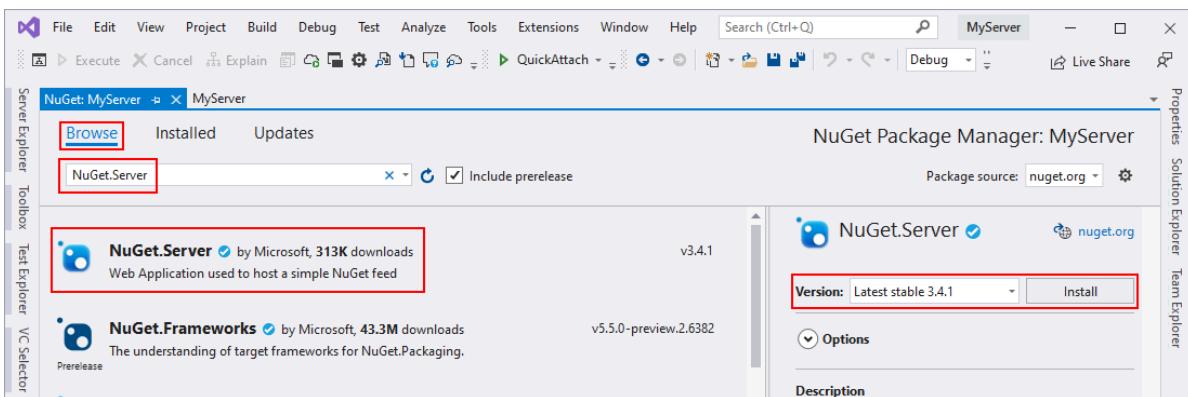
2. 将“框架”设置为“.NET Framework 4.6”。



3. 为应用程序提供除 NuGet.Server 之外的合适名称，选择“确定”，在接下来出现的对话框中选择“空”模板，然后选择“确定”。



4. 右键单击项目，选择“管理 NuGet 包”。
5. 如果面向 .NET Framework 4.6，请在“包管理器 UI”中，选择“浏览器”选项卡，然后搜索并安装 NuGet.Server 包的最新版本。（也可以使用 `Install-Package NuGet.Server` 从包管理器控制台安装。）如果出现提示，请接受此许可条款。



6. 安装 NuGet.Server 会将空 Web 应用程序转换成包源。此操作会安装各种其他包，在应用程序中创建 `Packages` 文件夹，并修改 `web.config` 以包括其他设置（请参阅该文件中的注释部分以获取详细信息）。

IMPORTANT

在 NuGet.Server 包完成对该文件的修改后，仔细检查 `web.config`。NuGet.Server 可能不会覆盖现有元素，而会创建重复元素。稍后尝试运行该项目时，这些重复项会导致“内部服务器错误”。例如，如果 `web.config` 在安装 NuGet.Server 之前包含 `<compilation debug="true" targetFramework="4.5.2" />`，则该包不会覆盖它，而是会插入另一个 `<compilation debug="true" targetFramework="4.6" />`。在这种情况下，请删除具有较旧框架版本的元素。

7. 在 Visual Studio 本地运行网站（使用“调试”>“开始执行（不调试）”或 Ctrl+F5）。主页提供包源 URL，如下所示。如果发现错误，请仔细检查 `web.config` 是否有重复元素（如前文所述）。

You are running NuGet.Server v3.4.1.0

Click [here](#) to view your packages.

Repository URLs

In the package manager settings, add the following URL to the list of Package Sources:

<https://localhost:44353/nuget>

To enable pushing packages to this feed using the [NuGet command line tool](#) (nuget.exe), set the `apiKey` appSetting in `web.config`.

Adding packages

To add packages to the feed put package files (.nupkg files) in the folder `D:\docs\NuGet-Server\MyServer\MyServer\Packages`

Click [here](#) to clear the package cache.

8. 首次运行应用程序时，NuGet.Server 会重新构建 `Packages` 文件夹，以包含每个包的文件夹。这符合 NuGet 3.3 中引入的用于提高性能的**本地存储布局**。添加更多包时，请继续遵照此结构。
9. 测试本地部署后，请根据需要将应用程序部署到任何其他内部或外部网站。
10. 部署到 `http://<domain>` 后，用于包源的 URL 将为 `http://<domain>/nuget`。

以外部方式向源添加包

NuGet.Server 站点运行后，就可以使用 `nuget push` 添加包，前提是在 `web.config` 中设置了 API 密钥值。

安装 NuGet.Server 包后，`web.config` 包含一个空 `appSetting/apiKey` 值：

```
<appSettings>
  <add key="apiKey" value="" />
</appSettings>
```

省略 `apiKey` 或将其留空时，会禁用向源推送包的功能。

要启用此功能，请设置 `apiKey` 的值（理想情况下为强密码），并添加值为 `true` 名为 `appSettings/requireApiKey` 的密钥：

```
<appSettings>
  <!-- Sets whether an API Key is required to push/delete packages -->
  <add key="requireApiKey" value="true" />

  <!-- Set a shared password (for all users) to push/delete packages -->
  <add key="apiKey" value="" />
</appSettings>
```

如果服务器已受保护或不需要其他 API 密钥（例如，在本地团队网络上使用专用服务器时），可将 `requireApiKey` 设置为 `false`。然后，有权访问服务器的所有用户均可推送包。

从 NuGet.Server 3.0.0 开始，推送包的 URL 更改为 `http://<domain>/nuget`。在 3.0.0 版本之前，推送 URL 为 `http://<domain>/api/v2/package`。

对于 NuGet 3.2.1 和更高版本，除 `/nuget` 外，默认还会通过启动配置（默认为 `NuGetODataConfig.cs`）中的 `enableLegacyPushRoute: true` 选项启用此旧 URL `/api/v2/package`。请注意，在同一项目中托管多个源时，此功能不适用。

从源中删除包

使用 NuGet.Server 时，`nuget delete` 命令会从存储库中删除一个包，但前提是包含 API 密钥和注释。

如果想要改变行为以从列表中删除包（将其保留为可用于包还原），请将 `web.config` 中的 `enableDelisting` 键更改 `true`。

配置包文件夹

对于 NuGet.Server 1.5 和更高版本，可使用 `web.config` 中的 `appSettings/packagesPath` 值自定义包文件夹：

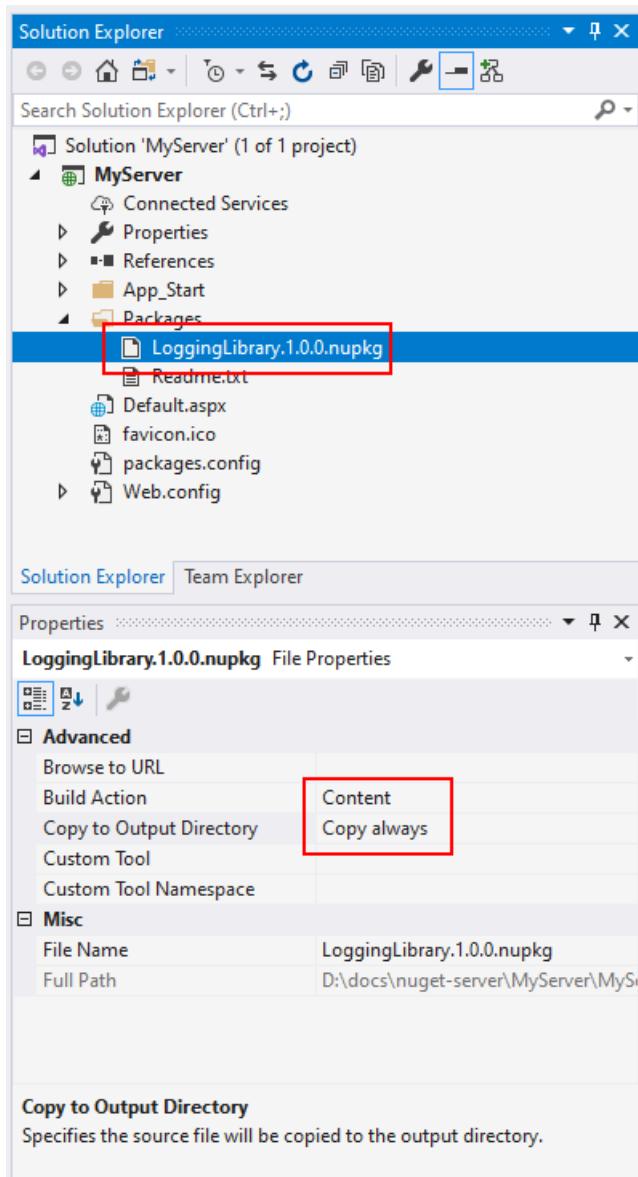
```
<appSettings>
  <!-- Set the value here to specify your custom packages folder. -->
  <add key="packagesPath" value="C:\MyPackages" />
</appSettings>
```

`packagesPath` 可以是绝对或虚拟路径。

省略 `packagesPath` 或将其留空时，包文件夹是默认的 `~/Packages`。

发布 Web 应用时使包可用

要在向服务器发布应用程序时在源中提供包，请将每个 `.nupkg` 文件添加到 Visual Studio 中的 `Packages` 文件夹，然后将每个文件的“生成操作”设置为“内容”，将“复制到输出目录”设置为“始终复制”：



发行说明

[GitHub 发布页](#)上提供了 NuGet 的发行说明。其中包括有关 bug 修复和新增功能的详细信息。

NuGet.Server 支持

有关使用 NuGet.Server 的其他帮助，请在 <https://github.com/nuget/NuGetGallery/issues> 上创建问题。

本地源

2020/4/8 • [Edit Online](#)

本地 NuGet 包源只是本地网络(甚至自己的计算机)上放置包的分层文件夹结构。然后，这些源可在使用 CLI、包管理器 UI 和包管理器控制台的所有其他 NuGet 操作中用作包源。

若要启用源，请使用 `\myserver\packages` 包管理器 UI 或 `nuget sources` 命令将其路径名(如)添加到源列表中。

NOTE

NuGet 3.3+ 中支持分层文件夹结构。较旧版本的 NuGet 仅使用包含包的一个文件夹，其性能远低于层次结构。

初始化和维护分层文件夹

分层版本控制的文件夹树具有以下常规结构：

```
\myserver\packages
└<packageID>
  └<version>
    ├<packageID>.<version>.nupkg
    └<other files>
```

当使用 `nuget add` 命令将包复制到源时，NuGet 自动创建此结构：

```
nuget add new_package.1.0.0.nupkg -source \myserver\packages
```

`nuget add` 命令一次仅用于一个包，这在设置具有多个包的源时会造成不便。

在此情况下，请使用 `nuget init` 命令将文件夹中的所有包复制到源，就像在每个源上单独运行 `nuget add`。例如，以下命令将所有包从 `c:\packages` 复制到 `\myserver\packages` 上的分层树中：

```
nuget init c:\packages \myserver\packages
```

与 `add` 命令一样，`init` 为每个包标识符创建文件夹，每个文件夹中包含具有相应包的版本号文件夹。

安装 NuGet 包时会发生什么情况？

2020/4/8 • [Edit Online](#)

简而言之，不同的 NuGet 工具通常会对项目文件或 `packages.config` 中的包创建引用，然后执行包还原，从而有效安装包。`nuget install` 除外，它仅将包展开到 `packages` 文件夹，而不修改任何其他文件。

一般流程如下：

1. (除 `nuget.exe` 之外的所有工具) 将包标识符和版本记录到项目文件或 `packages.config` 中。

如果安装工具是 Visual Studio 或 dotnet CLI，则该工具首先尝试安装包。如果不兼容，则不会将包添加到项目文件或 `packages.config` 中。

2. 获取包：

- 检查包是否(通过标识符和版本号)已安装在 `global-packages` 文件夹中，如[管理全局包和缓存文件夹](#)中所述。
- 如果包不在 `global-packages` 文件夹中，则尝试从[配置文件](#)中列出的源中检索它。对于在线源，请首先尝试从 HTTP 缓存中检索包，除非通过 `nuget.exe` 命令指定 `-NoCache` 或通过 `dotnet restore` 指定 `--no-cache`。(Visual Studio 和 `dotnet add package` 始终使用缓存。)如果从缓存中使用包，“缓存”将出现在输出中。缓存有 30 分钟的到期时间。
- 如果包不在 HTTP 缓存中，请尝试从配置中列出的源下载包。如果下载包，则会在输出中出现“GET”和“OK”。NuGet 以常规详细级别记录 http 流量。
- 如果无法从任何源成功获取包，安装将失败，并显示诸如 `NU1103` 之类的错误。注意，来自 `nuget.exe` 命令的错误仅显示最后检查的源，但意味着无法从任何源获取包。

获取包时，NuGet 配置中的源顺序可能适用：

- NuGet 将首先检查源本地文件夹和网络共享，然后再检查 HTTP 源。

3. 保存包副本和 `http-cache` 文件夹中的其他信息，如[管理全局包和缓存文件夹](#)中所述。
4. 如下载，请将包安装到每个用户的 `global-packages` 文件夹中。NuGet 创建每个包标识符的子文件夹，然后创建每个已安装包版本的子文件夹。
5. NuGet 安装所需的包依赖项。此过程可能会更新过程中的包版本，如[依赖项解析](#)中所述。
6. 更新其他项目文件和文件夹：

- 对于使用 `PackageReference` 的项目，更新存储在 `obj/project.assets.json` 中的包依赖项关系图。包内容本身不会复制到任何项目文件夹中。
- 如果包使用[源和配置文件转换](#)，则更新 `app.config` 和/或 `web.config`。

7. (仅适用于 Visual Studio) 如果可用，请在 Visual Studio 窗口中显示包的自述文件。

享受利用 NuGet 包高效处理代码的体验！

包版本控制

2020/4/8 • [Edit Online](#)

始终使用特定包的包标识符和确切的版本号来引用该包。例如, nuget.org 上的[实体框架](#)提供了数十个特定包, 范围从版本 4.1.10311 到版本 6.1.3(最新稳定版)以及各种预发布版本(例如 6.2.0-beta1)。

创建包时, 可以分配带有可选预发布文本后缀的特定版本号。另一方面, 使用包时, 可以指定确切的版本号或可接受的版本范围。

本主题内容:

- [版本基础知识](#), 包括预发布后缀。
- [版本范围](#)
- [规范化的版本号](#)

版本基础知识

特定版本号的格式为 Major.Minor.Patch[-Suffix] , 其中的组件具有以下含义:

- *Major*: 重大更改
- *Minor*: 新增功能, 但可向后兼容
- *Patch*: 仅可向后兼容的 bug 修复
- *-Suffix*(可选): 连字符后跟字符串, 表示预发布版本([遵循语义化版本控制或 SemVer 1.0 约定](#))。

示例:

```
1.0.1  
6.11.1231  
4.3.1-rc  
2.2.44-beta1
```

IMPORTANT

nuget.org 拒绝缺少确切版本号的任何包上传。必须在用于创建包的 `.nuspec` 或项目文件中指定版本。

预发布版本

从技术上讲, 包创建者可以将任何字符串用作后缀来表示预发布版本, 因为 NuGet 会将任何此类版本视为预发布版本, 而不进行任何其他解释。也就是说, NuGet 在任何涉及的 UI 中显示完整版本字符串, 并让使用者自行理解后缀的含义。

综上所述, 包开发人员通常遵循识别的命名约定:

- `-alpha`: Alpha 版本, 通常用于在制品和试验品。
- `-beta`: Beta 版本, 通常指可用于下一计划版本的功能完整的版本, 但可能包含已知 bug。
- `-rc`: 候选发布, 通常可能为最终(稳定)版本, 除非出现重大 bug。

NOTE

NuGet 4.3.0+ 支持 [SemVer 2.0.0](#), 后者支持采用点表示法的预发布号, 如 1.0.1-build.23 中所示。NuGet 4.3.0 之前的版本不支持点表示法。可以使用类似于 1.0.1-build23 的形式。

解析包引用时，如果多个包版本只有后缀不同，NuGet 会首先选择不带后缀的版本，然后按反向字母顺序来排列预发布版本的优先顺序。例如，将按显示的确切顺序选择以下版本：

```
1.0.1
1.0.1-zzz
1.0.1-rc
1.0.1-open
1.0.1-beta
1.0.1-alpha2
1.0.1-alpha
1.0.1-aaa
```

语义化版本控制 2.0.0

借助 NuGet 4.3.0+ 和 Visual Studio 2017 版本 15.3+，NuGet 支持[语义化版本控制 2.0.0](#)。

旧客户端不支持 SemVer v2.0.0 的某些语义。在以下任意一种情况下，NuGet 会将包版本视为特定于 SemVer v2.0.0：

- 预发布标签以点分隔，例如，1.0.0-alpha.1
- 版本具有生成元数据，例如，1.0.0+githash

对于 nuget.org，在以下任意一种情况下，会将包定义为 SemVer v2.0.0 包：

- 按照上述定义，包自己的版本与 SemVer v2.0.0 兼容，但与 SemVer v1.0.0 不兼容。
- 按照上述定义，包的任何依赖项版本范围都具有与 SemVer v2.0.0 兼容但与 SemVer v1.0.0 不兼容的最低版本或最高版本；例如，[1.0.0-alpha.1,)。

如果将 SemVer v2.0.0 特定包上传到 nuget.org，则该包对较旧的客户端不可见，并且仅可用于以下 NuGet 客户端：

- NuGet 4.3.0+
- Visual Studio 2017 版本 15.3+
- 带有[NuGet VSIX v3.6.0](#) 的 Visual Studio 2015
- dotnet
 - dotnetcore.exe (.NET SDK 2.0.0+)

第三方客户端：

- JetBrains Rider
- Paket 版本 5.0+

版本范围

引用包依赖项时，NuGet 支持使用间隔表示法来指定版本范围，汇总如下：

NOTATION	间隔表示法	说明
1.0	$x \geq 1.0$	最低版本(包含)
(1.0,)	$x > 1.0$	最低版本(独占)
[1.0]	$x == 1.0$	精确的版本匹配
(,1.0]	$x \leq 1.0$	最高版本(包含)

NOTATION	EXAMPLE	
(1.0)	$x < 1.0$	最高版本(独占)
[1.0,2.0]	$1.0 \leq x \leq 2.0$	精确范围(包含)
(1.0,2.0)	$1.0 < x < 2.0$	精确范围(独占)
[1.0,2.0)	$1.0 \leq x < 2.0$	混合了最低版本(包含)和最高版本(独占)
(1.0)	无效	无效

使用 PackageReference 格式时, NuGet 还支持使用浮点表示法 * 来表示版本号的主要、次要、修补程序和预发布后缀部分。 `packages.config` 格式不支持可变版本。

NOTE

PackageReference 中的版本范围包括预发布版本。按照设计, 可变版本不会解析预发布版本, 除非选择加入。有关相关功能请求的状态, 请参阅[问题 6434](#)。

示例

始终指定项目文件、`packages.config` 文件和 `.nuspec` 文件中的包依赖项的版本或版本范围。如果没有版本或版本范围, NuGet 2.8.x 及更早版本会在解析依赖项时选择最新的可用包版本, 而 NuGet 3.x 及更高版本会选择最低的包版本。指定版本或版本范围可以避免这种不确定性。

项目文件中的引用 (PackageReference)

```
<!-- Accepts any version 6.1 and above. -->
<PackageReference Include="ExamplePackage" Version="6.1" />

<!-- Accepts any 6.x.y version. -->
<PackageReference Include="ExamplePackage" Version="6.*" />
<PackageReference Include="ExamplePackage" Version="[6,7)" />

<!-- Accepts any version above, but not including 4.1.3. Could be
     used to guarantee a dependency with a specific bug fix. -->
<PackageReference Include="ExamplePackage" Version="(4.1.3,)" />

<!-- Accepts any version up below 5.x, which might be used to prevent pulling in a later
     version of a dependency that changed its interface. However, this form is not
     recommended because it can be difficult to determine the lowest version. -->
<PackageReference Include="ExamplePackage" Version="(<5.0)" />

<!-- Accepts any 1.x or 2.x version, but not 0.x or 3.x and higher. -->
<PackageReference Include="ExamplePackage" Version="[1,3)" />

<!-- Accepts 1.3.2 up to 1.4.x, but not 1.5 and higher. -->
<PackageReference Include="ExamplePackage" Version="[1.3.2,1.5)" />
```

packages.config 中的引用:

在 `packages.config` 中, 通过还原包时使用的确切 `version` 属性列出每个依赖项。`allowedVersions` 属性仅在更新操作过程中用于将版本约束到包可能更新到的版本。

```

<!-- Install/restore version 6.1.0, accept any version 6.1.0 and above on update. -->
<package id="ExamplePackage" version="6.1.0" allowedVersions="6.1.0" />

<!-- Install/restore version 6.1.0, and do not change during update. -->
<package id="ExamplePackage" version="6.1.0" allowedVersions="[6.1.0]" />

<!-- Install/restore version 6.1.0, accept any 6.x version during update. -->
<package id="ExamplePackage" version="6.1.0" allowedVersions="[6,7)" />

<!-- Install/restore version 4.1.4, accept any version above, but not including, 4.1.3.
     Could be used to guarantee a dependency with a specific bug fix. -->
<package id="ExamplePackage" version="4.1.4" allowedVersions="(4.1.3,)" />

<!-- Install/restore version 3.1.2, accept any version up below 5.x on update, which might be
     used to prevent pulling in a later version of a dependency that changed its interface.
     However, this form is not recommended because it can be difficult to determine the lowest version.
-->
<package id="ExamplePackage" version="3.1.2" allowedVersions="(,5.0)" />

<!-- Install/restore version 1.1.4, accept any 1.x or 2.x version on update, but not
     0.x or 3.x and higher. -->
<package id="ExamplePackage" version="1.1.4" allowedVersions="[1,3)" />

<!-- Install/restore version 1.3.5, accepts 1.3.2 up to 1.4.x on update, but not 1.5 and higher. -->
<package id="ExamplePackage" version="1.3.5" allowedVersions="[1.3.2,1.5)" />

```

.nuspec 文件中的引用

`<dependency>` 元素中的 `version` 属性描述了依赖项可接受的范围版本。

```

<!-- Accepts any version 6.1 and above. -->
<dependency id="ExamplePackage" version="6.1" />

<!-- Accepts any version above, but not including 4.1.3. Could be
     used to guarantee a dependency with a specific bug fix. -->
<dependency id="ExamplePackage" version="(4.1.3,)" />

<!-- Accepts any version up below 5.x, which might be used to prevent pulling in a later
     version of a dependency that changed its interface. However, this form is not
     recommended because it can be difficult to determine the lowest version. -->
<dependency id="ExamplePackage" version="(,5.0)" />

<!-- Accepts any 1.x or 2.x version, but not 0.x or 3.x and higher. -->
<dependency id="ExamplePackage" version="[1,3)" />

<!-- Accepts 1.3.2 up to 1.4.x, but not 1.5 and higher. -->
<dependency id="ExamplePackage" version="[1.3.2,1.5)" />

```

规范化的版本号

NOTE

这是 NuGet 3.4 及更高版本的重大更改。

当在安装、重新安装或还原操作过程中从存储库获取包时，NuGet 3.4+ 会按如下所示处理版本号：

- 从版本号中删除前导零：

```
1.00 is treated as 1.0  
1.01.1 is treated as 1.1.1  
1.00.0.1 is treated as 1.0.0.1
```

- 将忽略版本号第四部分中的零

```
1.0.0.0 is treated as 1.0.0  
1.0.01.0 is treated as 1.0.1
```

- 已删除 SemVer 2.0.0 生成元数据

```
1.0.7+r3456 is treated as 1.0.7
```

`pack` 和 `restore` 操作可尽可能规范化版本。对于已生成的包，此规范化不会影响包本身的版本号；仅影响 NuGet 在解析依赖项时匹配版本的方式。

但是，NuGet 包存储库必须以与 NuGet 相同的方式处理这些值，以防止包版本重复。因此，包含包版本 1.0 的存储库还不应将版本 1.0.0 作为单独的不同包进行托管。

NuGet 如何解析包依赖项

2020/4/8 • [Edit Online](#)

每当安装或重新安装包(包括在[还原](#)过程中安装)时, NuGet 还会安装第一个包所依赖的任何其他包。

这些直接依赖项可能本身也具有依赖项, 并可能继续延伸到任意深度。这便形成了所谓的“依赖项关系图”, 用于说明各级包之间的关系。

当多个包具有相同的依赖项时, 同一个包 ID 会在关系图中多次出现且可能具有不同的版本约束。但是, 一个项目中只能使用给定包的一个版本, 因此 NuGet 必须选择要使用的版本。确切流程取决于要使用的包管理格式。

利用 PackageReference 解析依赖项

当将包安装到使用 PackageReference 格式的项目中时, NuGet 将添加对相应文件中的平面包关系图的引用并提前解决冲突。此过程称为“传递还原”。重新安装或还原包指的是下载关系图中列出的包的过程, 此过程可加快生成的速度和提高其可预测性。还可以利用 2.8.* 等可变版本, 以避免为了能使用最新版本的包而修改项目。

当 NuGet 还原进程在生成之前运行时, 它将首先解析内存中的依赖项, 然后将生成的关系图写入名为 `project.assets.json` 的文件。如果 `packages.lock.json` 启用了锁定文件功能, 它还会将已解析的依赖项写入名为的锁定文件。资产文件位于 `MSBuildProjectExtensionsPath`, 它默认是项目的“obj”文件夹。MSBuild 随后将读取此文件并将其转换成一组文件夹(可在其中找到潜在引用), 然后将它们添加到内存中的项目树。

`project.assets.json` 文件是临时的, 不应添加到源代码管理中。默认情况下, 此文件将在 `.gitignore` 和 `.tfignore` 中列出。请参阅[包与源代码管理](#)。

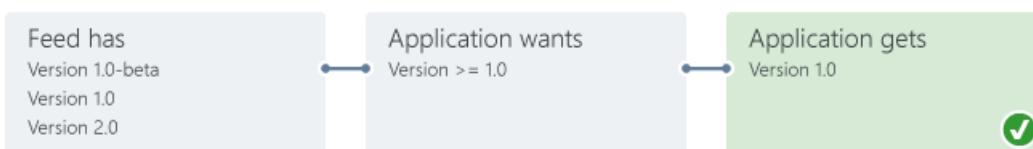
依赖项解析规则

传递还原应用 4 条主要规则来解析依赖项:最低适用版本、可变版本、选择最近项和等距依赖项。

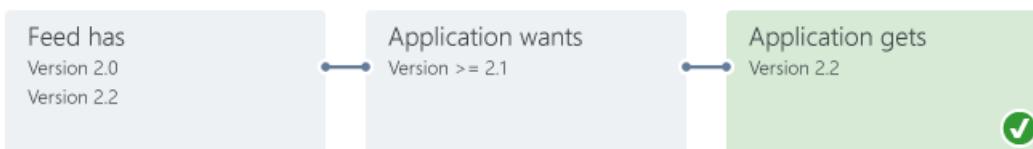
最低适用版本

最低适用版本规则根据包依赖项的定义还原包的最低可能版本。此规则还适用于应用程序上或类库上未声明为可变的依赖项。

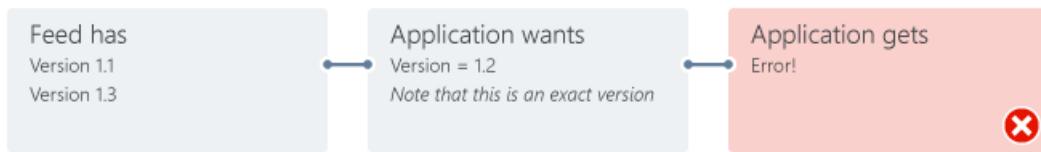
例如在下图中, 1.0 beta 版本低于 1.0, 因此 NuGet 选择 1.0 版本:



在下图中, 版本约束为 ≥ 2.1 , 但源上未提供版本 2.1, 因此 NuGet 选取能找到的下一个最低版本, 在此示例中即为 2.2:



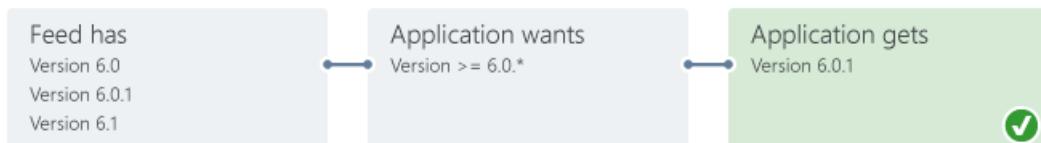
当源上未提供应用程序指定的确切版本号(如 1.2)时, NuGet 将在尝试安装或还原包时出错并导致失败:



可变版本

可变依赖项版本由 * 字符指定。例如，`6.0.*`。此版本规格表示“使用最新的 6.0.x 版本”；`4.*` 则表示“使用最新的 4.x 版本”。使用可变版本可减少对项目文件的更改，同时确保始终使用最新版本的依赖项。

使用可变版本时，NuGet 将解析与版本模式匹配的最高包版本，例如，请求 `6.0.*` 将获得以 6.0 开头的最高包版本：



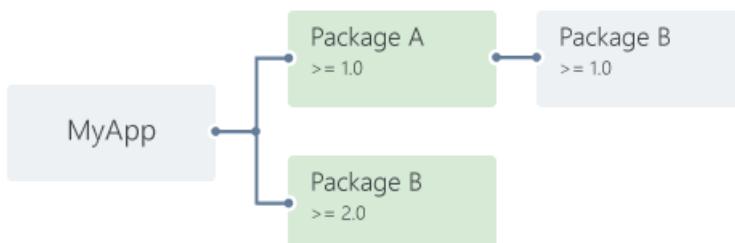
NOTE

有关可变版本和预发行版本的行为的信息，请参阅[包版本控制](#)。

选择最近项

当应用程序的包关系图包含相同包的不同版本时，NuGet 将选择关系图中最接近应用程序的包并忽略所有其他包。通过此行为，应用程序能够替代依赖项关系图中的任何特定包版本。

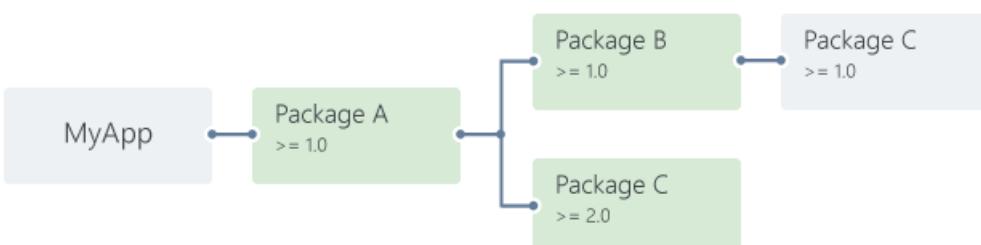
在下面的示例中，应用程序直接依赖于版本约束为 ≥ 2.0 的包 B。应用程序还依赖于版本约束为 ≥ 1.0 的包 A，此包依赖于包 B。在关系图中，由于包 B 2.0 上的依赖项更接近应用程序，因此将使用该版本：



WARNING

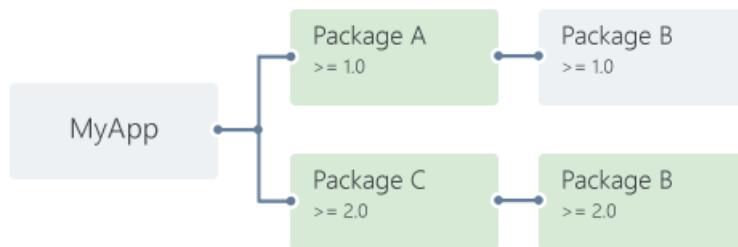
“选择最近项”规则可能导致包版本降级，可能破坏关系图中的其他依赖项。因此用户应用此规则时会收到警告提醒。

此规则还可提高大型依赖项关系图（如具有 BCL 包的关系图）的效率，因为忽略某个给定依赖项后，NuGet 还将忽略关系图中该分支上的所有其余依赖项。例如在下图中，由于使用了包 C 2.0，因此 NuGet 忽略了关系图中引用旧版包 C 的所有分支：

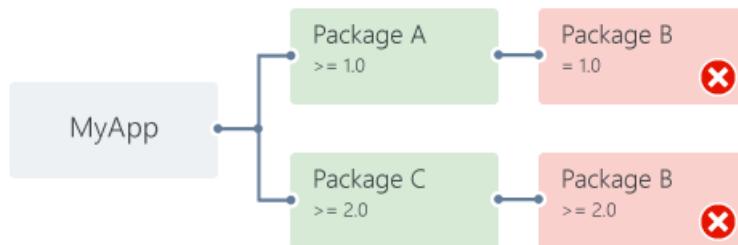


等距依赖项

当关系图中引用的不同包版本与应用程序具有相同距离时, NuGet 将使用满足所有版本要求的最低版本(与[最低适用版本](#)和[可变版本](#)规则相似)。例如在下图中, 2.0 版本的包 B 满足另一个 $>=1.0$ 约束, 因此使用此包:



某些情况下无法满足所有版本要求。如下所示, 如果包 A 明确要求包 B 1.0, 而包 C 要求包 B $>=2.0$, NuGet 则无法解析依赖项并显示错误。



在此情况下, 顶层使用者(应用程序或包)应在包 B 上添加其自己的直接依赖项, 以便应用[选择最近项](#)规则。

利用 packages.config 解析依赖项

利用 `packages.config`, 项目依赖项将作为简单列表写入 `packages.config`。这些包的所有依赖项也将写入同一列表。安装包时, NuGet 可能还会修改 `.csproj` 文件、`app.config`、`web.config` 和其他单独文件。

利用 `packages.config`, NuGet 可尝试解决在安装每个单独包期间出现的依赖项冲突。也就是说, 如果正在安装包 A 并且其依赖于包 B, 同时包 B 已作为其他项的依赖项在 `packages.config` 中列出, 则 NuGet 将比较所请求的包 B 版本, 并尝试找到满足所有版本约束的版本。具体而言, NuGet 将选择可满足依赖项的较低 major.minor 版本。

默认情况下, NuGet 2.8 会查找最低的修补程序版本(请参阅 [NuGet 2.8 发行说明](#))。可以通过 `DependencyVersion` 中的 `Nuget.Config` 属性和命令行上的 `-DependencyVersion` 开关控制此设置。

用于解析依赖项的 `packages.config` 进程会随着依赖项关系图的规模增大而愈加复杂。每次安装新包都需要遍历整个关系图, 并且可能引发版本冲突。发生冲突时, 安装将停止, 此时项目处于不确定状态, 很可能导致项目文件本身发生更改。使用其他包管理格式时则不会出现此问题。

管理依赖项资产

使用 `PackageReference` 格式时, 可以控制依赖项中的哪些资产可流入顶层项目。有关详细信息, 请参阅 [PackageReference](#)。

当顶层项目本身是一个包时, 还需要使用其依赖项在 `include` 文件中列出的 `exclude` 和 `.nuspec` 属性来控制此流。请参阅 [.nuspec 引用 - 依赖项](#)。

排除引用

对于有些方案, 同一项目中可能多次引用具有相同名称的程序集, 并因此生成设计时和生成时错误。例如, 某个项目既包含自定义版本的 `c.dll`, 又引用同样包含 `c.dll` 的包 C。同时, 该项目还依赖于同样依赖于包 C 和 `c.dll` 的包 B。在此情况下, NuGet 无法确定要使用哪一个 `c.dll`, 但你也不能直接删除包 C 上的项目依赖项, 因为包 B 也依赖于此依赖项。

要解决此问题，必须直接引用所需的 `c.dll`（或使用其他引用正确对象的包），然后在包 C 上添加不包括其所有资产的依赖项。此方法如下所示，具体取决于当前使用的包管理格式：

- `PackageReference`: 在依赖项中添加 `ExcludeAssets="All"` :

```
<PackageReference Include="PackageC" Version="1.0.0" ExcludeAssets="All" />
```

- `packages.config` : 从 `.csproj` 文件中删除对 `PackageC` 的引用，以便它仅引用所需版本的 `c.dll`。

在安装包期间更新依赖项

如果依赖项版本已满足，则在其他程序包安装期间不会更新依赖项。例如，假设包 A 依赖于包 B 并且指定版本号 1.0。源存储库包含程序包 B 的版本 1.0、1.1 和 1.2。如果将 A 安装在已包含 B 版本 1.0 的项目中，则 B 1.0 将保持使用状态，因为它满足版本限制。但是，如果包 A 请求 1.1 或更高版本的 B，则会安装 B 1.2。

解决包不兼容错误

在包还原操作期间，可能会看到错误“一个或多个包不兼容...”或包与项目的目标框架“不兼容”。

如果项目中引用的一个或多个包未指示其支持包的目标框架，则会出现此错误；即，包的 `lib` 文件夹中不包含与此项目兼容的目标框架的适用 DLL。（请参阅[目标框架](#)获取列表。）

例如，如果项目面向 `netstandard1.6`，并且你尝试安装仅在 `lib\net20` 和 `\lib\net45` 文件夹中包含 DLL 的包，则将看到类似如下针对包或其依赖项的消息：

```
Restoring packages for myproject.csproj...
Package ContosoUtilities 2.1.2.3 is not compatible with netstandard1.6 (.NETStandard,Version=v1.6). Package
ContosoUtilities 2.1.2.3 supports:
- net20 (.NETFramework,Version=v2.0)
- net45 (.NETFramework,Version=v4.5)
Package ContosoCore 0.86.0 is not compatible with netstandard1.6 (.NETStandard,Version=v1.6). Package
ContosoCore 0.86.0 supports:
- 11 (11,Version=v0.0)
- net20 (.NETFramework,Version=v2.0)
- sl3 (Silverlight,Version=v3.0)
- sl4 (Silverlight,Version=v4.0)
One or more packages are incompatible with .NETStandard,Version=v1.6.
Package restore failed. Rolling back package changes for 'MyProject'.
```

要解决不兼容问题，请执行下列操作之一：

- 将项目重定向到要使用的包所支持的框架。
- 联系包创建者并与其协作添加对所选框架的支持。[nuget.org](#) 中每个列出包的页面均针对此目的提供了“联系所有者”链接。

.nuspec 引用

2020/3/19 • [Edit Online](#)

.nuspec 文件是包含包元数据的 XML 清单。此清单同时用于生成包以及为使用者提供信息。清单始终包含在包中。

本主题内容：

- [常规形式和架构](#)
- [替换令牌\(用于 Visual Studio 项目时\)](#)
- [依赖项](#)
- [显式程序集引用](#)
- [Framework 程序集引用](#)
- [包括程序集文件](#)
- [包括内容文件](#)
- [示例 nuspec 文件](#)

项目类型兼容性

- 将 .nuspec 与使用 packages.config 的非 SDK 样式项目 nuget.exe pack 使用。
- 创建 sdk 样式项目包(通常是 .net Core 和使用 sdk 特性.NET Standard 项目)不需要 .nuspec 文件。(请注意, 当你创建包时, 将生成一个 .nuspec。)
如果要使用 dotnet.exe pack 或 msbuild pack target 创建包, 我们建议您改为在项目文件中的 .nuspec 文件中包含所有属性。不过, 您可以改为选择使用 .nuspec 文件, 以使用 dotnet.exe 或 msbuild pack target 进行打包。
- 对于从 packages.config 迁移到 PackageReference 的项目, 创建包不需要 .nuspec 文件。相反, 请使用 tpack。

常规形式和架构

当前 nuspec.xsd 架构文件可在 [NuGet GitHub 存储库](#) 中找到。

在此构架中, .nuspec 文件具有以下常规形式：

```
<?xml version="1.0" encoding="utf-8"?>
<package xmlns="http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd">
    <metadata>
        <!-- Required elements-->
        <id></id>
        <version></version>
        <description></description>
        <authors></authors>

        <!-- Optional elements -->
        <!-- ... -->
    </metadata>
    <!-- Optional 'files' node -->
</package>
```

有关架构的清晰可视表示形式, 请在 Visual Studio 中以“设计”模式打开架构文件, 然后单击“XML 架构资源管

理器”链接。或者，将该文件作为代码打开，在编辑器中右键单击，然后选择“显示 XML 架构资源管理器”。无论哪种方式，都会获得如下所示的视图（大多数部件都处于扩展状态）：



所需的元数据元素

尽管以下元素是包的最低要求，但应该考虑添加[可选元数据元素](#)以改善开发人员对包的整体体验。

这些元素必须出现在 `<metadata>` 元素中。

id

不区分大小写的包标识符，在 nuget.org 或包驻留的任意库中必须是唯一的。ID 不得包含空格或对 URL 无效的字符，通常遵循 .NET 命名空间规则。有关指南，请参阅[选择唯一的包标识符](#)。

版本

遵循 major.minor.patch 模式的包版本。版本号可能包括预发布后缀，如[包版本控制](#)中所述。

description

用于 UI 显示的包的说明。

authors

以逗号分隔的包作者列表，与 nuget.org 上的配置文件名称匹配。它们显示在 nuget.org 上的 NuGet 库中，并用于通过相同作者交叉引用包。

可选元数据元素

所有者

使用 nuget.org 上的配置文件名称的包创建者的逗号分隔列表。这通常与 `authors` 中的列表相同，并且在将包上传到 nuget.org 时将被忽略。请参阅[在 nuget.org 上管理包所有者](#)。

projectUrl

包的主页 URL，通常显示在 UI 中以及 nuget.org 中。

licenseUrl

IMPORTANT

licenseUrl 已被弃用。请改用许可证。

包的许可证的 URL，通常显示在 UI (如 nuget.org) 中。

license

包中的许可证文件的 SPDX 许可证表达式或路径，通常显示在 UI 中，如 nuget.org。如果要使用常见许可证（如 MIT 或 BSD-2 子句）来授权包，请使用关联的 [SPDX 许可证标识符](#)。例如：

```
<license type="expression">MIT</license>
```

NOTE

NuGet.org 仅接受开源计划或免费 Software Foundation 批准的许可表达式。

如果你的包在多个常用许可证下获得许可，则可以使用 [SPDX 表达式语法版本 2.0](#) 指定复合许可证。例如：

```
<license type="expression">BSD-2-Clause OR MIT</license>
```

如果使用许可证表达式不支持的自定义许可证，则可以使用许可证文本打包 `.txt` 或 `.md` 文件。例如：

```
<package>
  <metadata>
    ...
    <license type="file">LICENSE.txt</license>
    ...
  </metadata>
  <files>
    ...
    <file src="licenses\LICENSE.txt" target="" />
    ...
  </files>
</package>
```

对于 MSBuild 等效项，请查看 [打包许可证表达式或许可证文件](#)。

下面的 [ABNF](#) 中介绍了 NuGet 的许可证表达式的确切语法。

```
license-id          = <short form license identifier from https://spdx.org/spdx-specification-21-web-version#h.luq9dgcle9mo>

license-exception-id = <short form license exception identifier from https://spdx.org/spdx-specification-21-web-version#h.ruv3y18g6czd>

simple-expression = license-id / license-id"+"

compound-expression = 1*1(simple-expression /
                           simple-expression "WITH" license-exception-id /
                           compound-expression "AND" compound-expression /
                           compound-expression "OR" compound-expression ) /
                           "(" compound-expression ")" )

license-expression = 1*1(simple-expression / compound-expression / UNLICENSED)
```

iconUrl

IMPORTANT

iconUrl 已被弃用。改为使用图标。

具有透明度背景的 128x128 图像的 URL，该图像在 UI 显示中用作包的图标。请确保此元素包含直接图像 URL，而不是包含图像的网页的 URL。例如，若要使用 GitHub 中的映像，请使用 [等原始文件 URL](https://github.com/<username>/<repository>/raw/<branch>/<logo.png>)。

icon

它是指向包内的映像文件的路径，通常显示在 UI 中，如 nuget.org 作为包图标。图像文件大小限制为 1 MB。

支持的文件格式包括 JPEG 和 PNG。建议使用 128x128 的图像分辨率。

例如，使用 nuget.exe 创建包时，会将以下内容添加到 nuspec：

```
<package>
  <metadata>
    ...
    <icon>images\icon.png</icon>
    ...
  </metadata>
  <files>
    ...
    <file src="..\icon.png" target="images\" />
    ...
  </files>
</package>
```

包图标 nuspec 示例。

对于 MSBuild 等效项，请查看对图标图像文件进行打包。

TIP

您可以同时指定 `icon` 和 `iconUrl` 来维护与不支持 `icon` 的源的向后兼容性。在未来版本中，Visual Studio 将支持来自基于文件夹的源的包 `icon`。

requireLicenseAcceptance

一个布尔值，用于指定客户端是否必须提示使用者接受包许可证后才可安装包。

developmentDependency

(2.8+) 一个布尔值，用于指定包是否被标记为仅开发依赖项，从而防止包作为依赖项包含到其他包中。对于 PackageReference (NuGet 4.8+)，此标志还意味着它将从编译中排除编译时资产。请参阅 [DevelopmentDependency support For PackageReference](#)

摘要

IMPORTANT

即将弃用 `summary`。请改用 `description`。

用于 UI 显示的包的简要说明。如果省略，则使用 `description` 的截断版本。

releaseNotes

(1.5+) 此版本包中所作更改的说明，通常代替包说明用在 UI 中，如 Visual Studio 包管理器的“更新”选项卡。

copyright

(1.5+) 包的版权详细信息。

语言

包的区域设置 ID。请参阅 [创建本地化包](#)。

标记

以空格分隔的标记和关键字列表，描述包并通过搜索和筛选辅助包的可发现性。

可用

(3.3+) 仅限内部使用。

repository

存储库元数据，由四个可选属性组成：`type` 和 `url` (4.0+) 以及 `branch` 和 `commit` (4.6+)。通过这些属性，可以将 `.nupkg` 映射到生成它的存储库，并将其作为单独的分支名称和/或提交生成包的 SHA-1 哈希。这

应该是公开提供的 url，可由版本控制软件直接调用。它不应是 html 页面，因为这是用于计算机的。对于“链接到项目”页，请改用 `projectUrl` 字段。

例如：

```
<?xml version="1.0"?>
<package xmlns="http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd">
  <metadata>
    ...
    <repository type="git" url="https://github.com/NuGet/NuGet.Client.git" branch="dev"
commit="e1c65e4524cd70ee6e22abe33e6cb6ec73938cb3" />
    ...
  </metadata>
</package>
```

title

可在某些 UI 显示中使用的包的友好标题。(Visual Studio 中的 nuget.org 和包管理器不显示标题)

集合元素

packageTypes

(3.5+) 如果不是传统的依赖项包，则为指定包类型的包括零个或多个 `<packageType>` 元素的集合。每个 `packageType` 都具有 `name` 和 `version` 特性。请参阅[设置包类型](#)。

依赖关系

零个或多个 `<dependency>` 元素的集合，用来指定包的依赖项。每个 `dependency` 都具有 `id`、`version`、`include (3.x+)` 和 `exclude (3.x+)` 特性。请参阅下面的[依赖项](#)。

frameworkAssemblies

(1.2+) 零个或多个 `元素的集合`，用来标识此包要求的 .NET Framework 程序集引用，从而确保引用添加到使用该包的项目 `<frameworkAssembly>`。每个 `frameworkAssembly` 都具有 `assemblyName` 和 `targetFramework` 特性。请参阅下面的[指定 Framework 程序集引用 GAC](#)。

引用

(1.5+) 零个或多个 `元素的集合`，用来指定包的文件夹中添加为项目引用的程序集 `<reference>``lib`。每个 `reference` 都具有 `file` 特性。`<references>` 也可包含具有 `targetFramework` 特性的 `<group>` 元素，然后包含 `元素 <reference>`。如果省略，则包含 `lib` 中的全部引用。请参阅下面的[指定显式程序集引用](#)。

contentFiles

(3.3+) `元素的集合`，用来标识包含在使用项目中的内容文件 `<files>`。这些文件用一组特性指定，用于描述如何在项目系统中使用这些文件。请参阅下面的[指定包含在包中的文件](#)。

files

`<package>` 节点可能包含一个 `<files>` 节点作为 `<metadata>` 的同级，并且在 `<metadata>` 下的 `<contentFiles>` 子节点指定要包含在包中的程序集和内容文件。有关详细信息，请参阅本主题后面的[包含程序集文件](#)和[包含内容文件](#)。

metadata 特性

minClientVersion

指定可安装此包的最低 NuGet 客户端版本，并由 `nuget.exe` 和 Visual Studio 程序包管理器强制实施。只要包依赖于特定 NuGet 客户端版本中添加的 `.nuspec` 文件的特定功能，就会使用此功能。例如，使用 `developmentDependency` 特性的包应为 `minClientVersion` 指定“2.8”。同样，使用 `contentFiles` 元素（请参阅下一部分）的包应将 `minClientVersion` 设置为“3.3”。另请注意，早于 2.5 的 NuGet 客户端无法识别此标记，所以无论 包含什么内容，它们总是拒绝安装该包 `minClientVersion`。

```

<?xml version="1.0" encoding="utf-8"?>
<package xmlns="http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd">
  <metadata minClientVersion="100.0.0.1">
    <id>dasdas</id>
    <version>2.0.0</version>
    <title />
    <authors>dsadas</authors>
    <owners />
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>My package description.</description>
  </metadata>
  <files>
    <file src="content\one.txt" target="content\one.txt" />
  </files>
</package>

```

替换令牌

创建包时, `nuget pack` 命令使用来自项目文件的值或 `.nuspec` 命令的 `<metadata>` 开关来替换 `pack` 文件的 `-properties` 节点中带 \$ 分隔符的令牌。

在命令行中, 可使用 `nuget pack -properties <name>=<value>;<name>=<value>` 指定令牌值。例如, 可使用 `$owners$` 中的 `$desc$` 和 `.nuspec` 令牌, 并在封装时提供值, 如下所示:

```

nuget pack MyProject.csproj -properties
owners=janedoe,harikm,kimo,xiaop;desc="Awesome app logger utility"

```

如要使用项目中的值, 请指定下表中描述的令牌(`AssemblyInfo` 指的是 `Properties` 中的文件, 如 `AssemblyInfo.cs` 或 `AssemblyInfo.vb`)。

若要使用这些令牌, 请通过项目文件而不仅仅是 `nuget pack` 来运行 `.nuspec`。例如, 使用以下命令时, `id` 文件中的 `$version$` 和 `.nuspec` 令牌会被替换为项目的 `AssemblyName` 和 `AssemblyVersion` 值:

```

nuget pack MyProject.csproj

```

通常情况下, 如果已有项目, 最初会使用自动包含一些标准令牌的 `.nuspec` 创建 `nuget spec MyProject.csproj`。然而, 如果项目缺少要求的 `.nuspec` 元素的值, 那么 `nuget pack` 失败。此外, 如果更改项目值, 请确定在创建包之前重新生成, 可通过 `pack` 命令的 `build` 开关方便地完成此操作。

除 `$configuration$` 外, 项目中的值优先于在命令行上分配给相同令牌的任何值。

<code>\$id\$</code>	项目文件	项目文件中的 <code>AssemblyName</code> (<code>title</code>)
<code>\$version\$</code>	<code>AssemblyInfo</code>	<code>AssemblyInformationalVersion</code> (如果存在), 否则为 <code>AssemblyVersion</code>
<code>\$author\$</code>	<code>AssemblyInfo</code>	<code>AssemblyCompany</code>
<code>\$title \$</code>	<code>AssemblyInfo</code>	<code>AssemblyTitle</code>
<code>\$description\$</code>	<code>AssemblyInfo</code>	<code>AssemblyDescription</code>

``	``	``
\$copyright\$	AssemblyInfo	AssemblyCopyright
\$configuration\$	程序集 DLL	用于生成程序集的配置，默认为 Debug。请注意，若要使用 Release 配置创建包，应始终在命令行上使用 -properties Configuration=Release 。

包含 [程序集文件](#) 和 [内容文件](#) 时，令牌也可用于解析路径。这些令牌与 MSBuild 属性具有相同的名称，因此可根据当前生成配置来选择要包含的文件。例如，如果在 `.nuspec` 文件中使用以下令牌：

```
<files>
  <file src="bin\$configuration$\$id$.pdb" target="lib\net40" />
</files>
```

在 MSBuild 中生成具有 `AssemblyName` 配置且 `LoggingLibrary` 为 `Release` 的程序集，包中 `.nuspec` 文件中的结果行如下所示：

```
<files>
  <file src="bin\Release\LoggingLibrary.pdb" target="lib\net40" />
</files>
```

依赖项元素

`<dependencies>` 中的 `<metadata>` 元素包含任意数量的 `<dependency>` 元素，用来标识顶级包所依赖的其他包。每个 `<dependency>` 的特性如下所示：

ATTRIBUTE	``
<code>id</code>	(必须)依赖项的包 ID，如“EntityFramework”和“NUnit”，同时也是 nuget.org 在包页面上显示的包名称。
<code>version</code>	(必需)可接受作为依赖项的版本范围。有关准确语法，请参阅 包版本控制 。不支持浮动版本。
<code>include</code>	包括/排除标记的逗号分隔列表(见下文)，指示要包含在最终包中的依赖项。默认值是 <code>all</code> 。
<code>exclude</code>	包括/排除标记的逗号分隔列表(见下文)，指示要排除在最终包外的依赖项。默认值为 <code>build, analyzers</code> ，可以覆盖此值。但仍会在最终包中隐式排除 <code>content/ ContentFiles</code> ，这是无法覆盖的。用 <code>exclude</code> 指定的标记优先于用 <code>include</code> 指定的标记。例如， <code>include="runtime, compile" exclude="compile"</code> 和 <code>include="runtime"</code> 相同。
<code>contentFiles</code>	内容

[[/]]	[[/]]
运行库	运行时、资源和 FrameworkAssemblies
编译 (compile)	lib
build	生成 (MSBuild 属性和目标)
native	native
none	无文件夹
all	全部文件夹

例如，以下行指示 `PackageA` 版本 1.1.0 或更高版本，以及 `PackageB` 版本 1.x 的依赖项。

```
<dependencies>
  <dependency id="PackageA" version="1.1.0" />
  <dependency id="PackageB" version="[1,2)" />
</dependencies>
```

以下行指示相同包上的依赖项，但指定包括 `contentFiles` 的 `build` 和 `PackageA` 文件夹，以及除 `native` 的 `compile` 和 `PackageB` 文件夹以外的所有文件夹。

```
<dependencies>
  <dependency id="PackageA" version="1.1.0" include="contentFiles, build" />
  <dependency id="PackageB" version="[1,2)" exclude="native, compile" />
</dependencies>
```

IMPORTANT

使用 `nuget spec` 从项目创建 `.nuspec` 时，该项目中存在的依赖项不会自动包括在生成的 `.nuspec` 文件中。请改用 `nuget pack myproject.csproj`，并从 `nupkg` 文件中获取 `nuspec` 文件。*Nuspec* 包含依赖项。

依赖项组

版本 2.0+

作为单个简单列表的替代方法，可使用 `<group>` 中的 `<dependencies>` 元素根据目标项目的框架配置文件指定依赖项。

每个组都有一个名为 `targetFramework` 的特性，并包含零个或多个 `<dependency>` 元素。当目标框架与项目的框架配置文件兼容时，将会一起安装这些依赖项。

无 `<group>` 特性的 `targetFramework` 元素被用作依赖项的默认列表或回退列表。有关确切的框架标识符，请参阅[目标框架](#)。

IMPORTANT

组格式不能与简单列表混合使用。

NOTE

与 `dependency groups` 中使用的 TFM 相比, `lib/ref` 文件夹中使用的**目标框架名字对象 (TFM)** 的格式不同。如果在 `dependencies group` 中声明的目标框架和 `.nuspec` 文件的 `lib/ref` 文件夹没有完全匹配项, 则 `pack` 命令将引发 NuGet 警告 NU5128。

以下示例显示了 `<group>` 元素的不同变体:

```
<dependencies>
  <group>
    <dependency id="RouteMagic" version="1.1.0" />
  </group>

  <group targetFramework=".NETFramework4.7.2">
    <dependency id="jQuery" version="1.6.2" />
    <dependency id="WebActivator" version="1.4.4" />
  </group>

  <group targetFramework="netcoreapp3.1">
  </group>
</dependencies>
```

显式程序集引用

使用 `packages.config` 的项目使用 `<references>` 元素显式指定目标项目在使用包时应引用的程序集。显式引用通常用于仅设计时程序集。有关详细信息, 请参阅有关详细信息, 请参阅[选择项目引用的程序集](#)。

例如, 以下 `<references>` 元素指示 NuGet 仅对 `xunit.dll` 和 `xunit.extensions.dll` 添加引用, 即使包中还有其他程序集:

```
<references>
  <reference file="xunit.dll" />
  <reference file="xunit.extensions.dll" />
</references>
```

引用组

作为单个简单列表的替代方法, 可使用 `<group>` 中的 `<references>` 元素根据目标项目的框架配置文件指定引用。

每个组都有一个名为 `targetFramework` 的特性, 并包含零个或多个 `<reference>` 元素。当目标框架与项目的框架配置文件兼容时, 会将引用添加到项目中。

无 `<group>` 特性的 `targetFramework` 元素被用作引用的默认列表或回退列表。有关确切的框架标识符, 请参阅[目标框架](#)。

IMPORTANT

组格式不能与简单列表混合使用。

以下示例显示了 `<group>` 元素的不同变体:

```

<references>
  <group>
    <reference file="a.dll" />
  </group>

  <group targetFramework="net45">
    <reference file="b45.dll" />
  </group>

  <group targetFramework="netcore45">
    <reference file="bc45.dll" />
  </group>
</references>

```

Framework 程序集引用

Framework 程序集是 .NET Framework 的一部分，并已存在于任何给定计算机的全局程序集缓存 (GAC) 中。通过在 `<frameworkAssemblies>` 元素中标识这些程序集，包可确保在项目尚未具有此类引用的情况下，将必需的引用添加到项目中。当然，此类程序集不直接包含在包中。

`<frameworkAssemblies>` 元素包含零个或多个 `<frameworkAssembly>` 元素，这些元素指定以下特性：

ATTRIBUTE	说明
<code>assemblyName</code>	(必需) 完全限定程序集名称。
<code>targetFramework</code>	(可选) 指定此引用适用的目标框架。如果省略，则表示该引用适用于全部框架。有关确切的框架标识符，请参阅 目标框架 。

以下示例显示了对全部目标框架的 `System.Net` 的引用，以及对仅用于 .NET Framework 4.0 的 `System.ServiceModel` 的引用：

```

<frameworkAssemblies>
  <frameworkAssembly assemblyName="System.Net" />

  <frameworkAssembly assemblyName="System.ServiceModel" targetFramework="net40" />
</frameworkAssemblies>

```

包含程序集文件

如果遵循[创建包](#)中介绍的约定，则不必在 `.nuspec` 文件中显式指定文件列表。`nuget pack` 命令自动选取所需的文件。

IMPORTANT

当包安装到项目中时，NuGet 自动将程序集引用添加到包的 DLL，不包括命名为 的内容，因为它们被假定为本地化的附属程序集 `.resources.dll`。为此，请避免对包含基本包代码的文件使用 `.resources.dll`。

若要绕过此自动行为，并显式控制包中包含的文件，请将 `<files>` 元素作为 `<package>` 的子元素（和 `<metadata>` 的同级元素），并使用单独的 `<file>` 元素标识每个文件。例如：

```

<files>
  <file src="bin\Debug\*.dll" target="lib" />
  <file src="bin\Debug\*.pdb" target="lib" />
  <file src="tools\**\*.*" exclude="**\*.log" />
</files>

```

在 NuGet 2.x 及更早版本中，如果项目使用 `packages.config`，在安装包时，`<files>` 元素也用于包含不可变的内容文件。通过 NuGet 3.3+ 和项目 PackageReference，将改为使用 `<contentFiles>` 元素。有关详细信息，请参阅下面的[包含内容文件](#)。

文件元素特性

每个 `<file>` 元素指定以下特性：

ATTRIBUTE	DEFINITION
<code>src</code>	文件或要包含的文件位置，受 <code>exclude</code> 特性指定排除规则约束。路径是相对于 <code>.nuspec</code> 文件的路径，除非指定了绝对路径。允许使用通配符 <code>*</code> ，双通配符 <code>**</code> 意味着递归文件夹搜索。
■	放置源文件的包中文件夹的相对路径，必须以 <code>lib</code> 、 <code>content</code> 、 <code>build</code> 或 <code>tools</code> 开头。请参阅 从基于约定的工作目录创建 .nuspec 。
<code>exclude</code>	要从 <code>src</code> 位置排除的文件或文件模式的分号分隔列表。允许使用通配符 <code>*</code> ，双通配符 <code>**</code> 意味着递归文件夹搜索。

示例

单个程序集

```

Source file:
library.dll

.nuspec entry:
<file src="library.dll" target="lib" />

Packaged result:
lib\library.dll

```

特定于目标框架的单个程序集

```

Source file:
library.dll

.nuspec entry:
<file src="assemblies\net40\library.dll" target="lib\net40" />

Packaged result:
lib\net40\library.dll

```

使用通配符的 DLL 集

```

Source files:
  bin\release\libraryA.dll
  bin\release\libraryB.dll

.nuspec entry:
<file src="bin\release\*.dll" target="lib" />

Packaged result:
  lib\libraryA.dll
  lib\libraryB.dll

```

适用于不同框架的 DLL

```

Source files:
  lib\net40\library.dll
  lib\net20\library.dll

.nuspec entry (using ** recursive search):
<file src="lib\**" target="lib" />

Packaged result:
  lib\net40\library.dll
  lib\net20\library.dll

```

排除文件

```

Source files:
  \tools\fileA.bak
  \tools\fileB.bak
  \tools\fileA.log
  \tools\build\fileB.log

.nuspec entries:
<file src="tools\*.*" target="tools" exclude="tools\*.bak" />
<file src="tools\*\*.*" target="tools" exclude="**\*.log" />

Package result:
  (no files)

```

包含内容文件

内容文件是包需要包含在项目中的不可变文件。不可变文件指的是使用项目不会修改的文件。内容文件示例包括：

- 作为资源嵌入的图像
- 已编译的源文件
- 需要包含在项目的生成输出中的脚本
- 需要包含在项目中，但不需要进行任何项目特定的更改的包的配置文件

内容文件使用 `<files>` 元素包含在包中，并在 `content` 特性中指定 `target` 文件夹。但是，使用 `PackageReference`（而不是使用 `<contentFiles>` 元素）将包安装到项目中时，将忽略这些文件。

为了最大限度地兼容使用项目，一个包最好在两个元素中指定内容文件。

对内容文件使用文件元素

对于内容文件，只需使用与程序集文件相同的格式，但应在 `content` 属性中将 `target` 指定为基本文件夹，如以下示例所示。

基本内容文件

```
Source files:  
  css\mobile\style1.css  
  css\mobile\style2.css  
  
.nuspec entry:  
  <file src="css\mobile\*.css" target="content\css\mobile" />  
  
Packaged result:  
  content\css\mobile\style1.css  
  content\css\mobile\style2.css
```

具有目录结构的内容文件

```
Source files:  
  css\mobile\style.css  
  css\mobile\wp7\style.css  
  css\browser\style.css  
  
.nuspec entry:  
  <file src="css\**\*.css" target="content\css" />  
  
Packaged result:  
  content\css\mobile\style.css  
  content\css\mobile\wp7\style.css  
  content\css\browser\style.css
```

特定于目标框架的内容文件

```
Source file:  
  css\cool\style.css  
  
.nuspec entry  
  <file src="css\cool\style.css" target="Content" />  
  
Packaged result:  
  content\style.css
```

复制到名称中带点的文件夹的内容文件

在此情况下, NuGet 发现 `target` 中的扩展名与 `src` 中的扩展名不匹配, 因此将 `target` 中名称的该部分作为文件夹:

```
Source file:  
  images\picture.png  
  
.nuspec entry:  
  <file src="images\picture.png" target="Content\images\package.icons" />  
  
Packaged result:  
  content\images\package.icons\picture.png
```

不具有扩展名的内容文件

若要包含不具有扩展名的文件, 请使用 `*` 或 `**` 通配符:

```
Source file:  
  flags\installed  
  
.nuspec entry:  
  <file src="flags\**" target="flags" />  
  
Packaged result:  
  flags\installed
```

具有深层路径和深层目标的内容文件

在此情况下，由于源文件和目标文件的扩展名匹配，NuGet 假定目标是文件名而不是文件夹：

```
Source file:  
  css\cool\style.css  
  
.nuspec entry:  
  <file src="css\cool\style.css" target="Content\css\cool" />  
  or:  
  <file src="css\cool\style.css" target="Content\css\cool\style.css" />  
  
Packaged result:  
  content\css\cool\style.css
```

重命名包中的内容文件

```
Source file:  
  ie\css\style.css  
  
.nuspec entry:  
  <file src="ie\css\style.css" target="Content\css\ie.css" />  
  
Packaged result:  
  content\css\ie.css
```

排除文件

```
Source file:  
  docs\*.txt (multiple files)  
  
.nuspec entry:  
  <file src="docs\*.txt" target="content\docs" exclude="docs\admin.txt" />  
  or  
  <file src="*.txt" target="content\docs" exclude="admin.txt;log.txt" />  
  
Packaged result:  
  All .txt files from docs except admin.txt (first example)  
  All .txt files from docs except admin.txt and log.txt (second example)
```

对内容文件使用 `contentFiles` 元素

NuGet 4.0+ 与 `PackageReference`

默认情况下，包将内容放置在 `contentFiles` 文件夹中（见下文），`nuget pack` 使用默认特性将全部文件包含在该文件夹中。在此情况下，根本没有必要在 `contentFiles` 中包含 `.nuspec` 节点。

若要控制包含哪些文件，`<contentFiles>` 元素指定是 `<files>` 元素的集合，可标识包含的确切文件。

这些文件用一组特性指定，用于描述如何在项目系统中使用这些文件：

ATTRIBUTE	■
include	(必需)文件或要包含的文件位置, 受 exclude 特性指定的排除规则约束。除非指定了绝对路径, 否则路径相对于 contentFiles 文件夹。允许使用通配符 * , 双通配符 ** 意味着递归文件夹搜索。
exclude	要从 src 位置排除的文件或文件模式的分号分隔列表。允许使用通配符 * , 双通配符 ** 意味着递归文件夹搜索。
buildAction	要分配给 MSBuild 内容项的生成操作, 例如 Content 、 None 、 Embedded Resource 、 Compile 等。默认值为 Compile 。
copyToOutput	指示是否将内容项复制到生成(或发布)输出文件夹的布尔值。默认值为 false 。
flatten	一个布尔值, 用于指示是将内容项复制到生成输出中的单个文件夹 (true), 还是保留包中的文件夹结构 (false)。此标志仅在 copyToOutput 标志设置为 true 时才有效。默认值为 false 。

安装包时, NuGet 从上到下应用 `<contentFiles>` 的子元素。如果多个条目与相同的文件匹配, 那么应用全部条目。如果相同特性发生冲突, 则最上面的条目将替代靠下的条目。

包文件夹结构

包项目应使用以下模式构建内容结构:

```
/contentFiles/{codeLanguage}/{TxM}/{any?}
```

- `codeLanguages` 可以是 cs 、 vb 、 fs 、 any 或给定 \$(ProjectLanguage) 的小写等效形式
- `TxM` 是 NuGet 支持的任何合法目标框架名字对象(请参阅[目标框架](#))。
- 任何文件夹结构都可以附加到此语法的末尾。

例如:

```
Language- and framework-agnostic:  
/contentFiles/any/any/config.xml  
  
net45 content for all languages  
/contentFiles/any/net45/config.xml  
  
C#-specific content for net45 and up  
/contentFiles/cs/net45/sample.cs
```

空文件夹可以使用 . 来选择不提供特定语言和 TxM 组合的内容, 例如:

```
/contentFiles/vb/any/code.vb  
/contentFiles/cs/any/.
```

contentFiles 部分示例

```

<?xml version="1.0" encoding="utf-8"?>
<package xmlns="http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd">
    <metadata>
        ...
        <contentFiles>
            <!-- Embed image resources -->
            <files include="any-any/images/dnf.png" buildAction="EmbeddedResource" />
            <files include="any-any/images/ui.png" buildAction="EmbeddedResource" />

            <!-- Embed all image resources under contentFiles/cs/ -->
            <files include="cs/**/*.png" buildAction="EmbeddedResource" />

            <!-- Copy config.xml to the root of the output folder -->
            <files include="cs/uap/config/config.xml" buildAction="None" copyToOutput="true"
flatten="true" />

            <!-- Copy run.cmd to the output folder and keep the directory structure -->
            <files include="cs/commands/run.cmd" buildAction="None" copyToOutput="true" flatten="false"
/>

            <!-- Include everything in the scripts folder except exe files -->
            <files include="cs/net45/scripts/*" exclude="**/*.exe" buildAction="None"
copyToOutput="true" />
        </contentFiles>
        </metadata>
    </package>

```

框架引用组

仅限版本 5.1 + 同时 `PackageReference`

框架引用是一个 .NET Core 概念，表示 WPF 或 Windows 窗体等共享框架。通过指定共享框架，包可确保其所有框架依赖项都包括在引用项目中。

每个 `<group>` 元素都需要一个 `targetFramework` 属性和零个或多个 `<frameworkReference>` 元素。

下面的示例演示如何为 .NET Core WPF 项目生成 nuspec。请注意，不建议使用包含框架引用的手动创作 nuspecs。请考虑改用目标包，这会自动从项目推断它们。

```

<package xmlns="http://schemas.microsoft.com/packaging/2012/06/nuspec.xsd">
    <metadata>
        <dependencies>
            <group targetFramework=".NETCoreApp3.1" />
        </dependencies>
        <frameworkReferences>
            <group targetFramework=".NETCoreApp3.1">
                <frameworkReference name="Microsoft.WindowsDesktop.App.WPF" />
            </group>
        </frameworkReferences>
    </metadata>
</package>

```

示例 nuspec 文件

未指定依赖项或文件的简单 `.nuspec`

```
<?xml version="1.0" encoding="utf-8"?>
<package xmlns="http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd">
  <metadata>
    <id>sample</id>
    <version>1.2.3</version>
    <authors>Kim Abercrombie, Franck Halmaert</authors>
    <description>Sample exists only to show a sample .nuspec file.</description>
    <language>en-US</language>
    <projectUrl>http://xunit.codeplex.com/</projectUrl>
    <license type="expression">MIT</license>
  </metadata>
</package>
```

具有依赖项的 [.nuspec](#)

```
<?xml version="1.0" encoding="utf-8"?>
<package xmlns="http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd">
  <metadata>
    <id>sample</id>
    <version>1.0.0</version>
    <authors>Microsoft</authors>
    <dependencies>
      <dependency id="another-package" version="3.0.0" />
      <dependency id="yet-another-package" version="1.0.0" />
    </dependencies>
  </metadata>
</package>
```

具有文件的 [.nuspec](#)

```
<?xml version="1.0"?>
<package xmlns="http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd">
  <metadata>
    <id>routedebugger</id>
    <version>1.0.0</version>
    <authors>Jay Hamlin</authors>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>Route Debugger is a little utility I wrote...</description>
  </metadata>
  <files>
    <file src="bin\Debug\*.dll" target="lib" />
  </files>
</package>
```

具有 Framework 程序集的 [.nuspec](#)

```
<?xml version="1.0"?>
<package xmlns="http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd">
  <metadata>
    <id>PackageWithGacReferences</id>
    <version>1.0</version>
    <authors>Author here</authors>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>
      A package that has framework assemblyReferences depending
      on the target framework.
    </description>
    <frameworkAssemblies>
      <frameworkAssembly assemblyName="System.Web" targetFramework="net40" />
      <frameworkAssembly assemblyName="System.Net" targetFramework="net40-client, net40" />
      <frameworkAssembly assemblyName="Microsoft.Devices.Sensors" targetFramework="sl4-wp" />
      <frameworkAssembly assemblyName="System.Json" targetFramework="sl3" />
    </frameworkAssemblies>
  </metadata>
</package>
```

在此示例中，为特定的项目目标安装了以下内容：

- .NET4 -> `System.Web` `System.Net`
- .NET4 Client Profile -> `System.Net`
- Silverlight 3 -> `System.Json`
- WindowsPhone -> `Microsoft.Devices.Sensors`

nuget.exe 引用

2020/3/19 • [Edit Online](#)

NuGet 行为由不同 `NuGet.Config` 或 `nuget.config` 文件中的设置控制，如[常见 NuGet 配置](#)中所述。

`nuget.config` 是包含顶级 `<configuration>` 节点的 XML 文件，而该节点包含本主题中所述的节元素。每节都包含零个或多个项。请参阅[示例配置文件](#)。设置名称不区分大小写，并且值可以使用[环境变量](#)。

配置节

包含杂项配置设置，可使用 `nuget config` 命令设置。

`dependencyVersion` 和 `repositoryPath` 仅适用于使用 `packages.config` 的项目。`globalPackagesFolder` 仅适用于使用 `PackageReference` 格式的项目。

节	描述
<code>dependencyVersion</code> (仅限于 <code>packages.config</code>)	包安装、还原和更新的默认 <code>DependencyVersion</code> 值(未直接指定 <code>-DependencyVersion</code> 开关时)。NuGet 包管理器 UI 也使用此值。值为 <code>Lowest</code> 、 <code>HighestPatch</code> 、 <code>HighestMinor</code> 、 <code>Highest</code> 。
<code>globalPackagesFolder</code> (仅限使用 <code>PackageReference</code> 的项目)	默认全局包文件夹的位置。默认值为 <code>%userprofile%\.nuget\packages</code> (Windows) 或 <code>~/.nuget/packages</code> (Mac/Linux)。相对路径可在项目特定的 <code>nuget.config</code> 文件中使用。此设置由 <code>NUGET_PACKAGES</code> 环境变量重写，该变量优先。
<code>repositoryPath</code> (仅限于 <code>packages.config</code>)	安装 NuGet 包的位置，而非默认的 <code>\$(Solutiondir)/packages</code> 文件夹。相对路径可在项目特定的 <code>nuget.config</code> 文件中使用。此设置由 <code>NUGET_PACKAGES</code> 环境变量重写，该变量优先。
<code>defaultPushSource</code>	如果操作未找到任何其他包源，则会标识应用作默认值的包源 URL 或路径。
<code>http_proxy</code> <code>http_proxy.user</code> <code>http_proxy.password</code> <code>no_proxy</code>	连接到包源时要使用的代理设置： <code>http_proxy</code> 应为 <code>http://<username>:<password>@<domain></code> 格式。密码已加密，且不能手动添加。对于 <code>no_proxy</code> ，该值是绕过代理服务器的域的列表(以逗号分隔)。可将 <code>http_proxy</code> 和 <code>no_proxy</code> 环境变量交替用于这些值。有关其他详细信息，请参阅 NuGet 代理设置 (skolima.blogspot.com)。
<code>signatureValidationMode</code>	指定用于验证包签名以便安装和还原的验证模式。值为 <code>accept</code> 、 <code>require</code> 。默认为 <code>accept</code> 。

示例：

```
<config>
  <add key="dependencyVersion" value="Highest" />
  <add key="globalPackagesFolder" value="c:\packages" />
  <add key="repositoryPath" value="c:\installed_packages" />
  <add key="http_proxy" value="http://company-squid:3128@contoso.com" />
  <add key="signatureValidationMode" value="require" />
</config>
```

bindingRedirects 节

在安装包时，配置 NuGet 是否执行自动绑定重定向。

“

”

skip

指示是否跳过自动绑定重定向的布尔。默认值为 false。

示例：

```
<bindingRedirects>
  <add key="skip" value="True" />
</bindingRedirects>
```

packageRestore 节

在生成期间控制包还原。

“

”

已启用

指示 NuGet 是否可执行自动还原的布尔。还可以使用 `EnableNuGetPackageRestore` 的值设置 `True` 环境变量，而不是在配置文件中设置此密钥。

automatic

指示 NuGet 是否应在生成期间检查缺少的包。

示例：

```
<packageRestore>
  <add key="enabled" value="true" />
  <add key="automatic" value="true" />
</packageRestore>
```

解决方案节

控制解决方案的 `packages` 文件夹是否包括在源代码管理中。此节仅适用于解决方案文件夹中的 `nuget.config` 文件。

“

”

disableSourceControlIntegration

指示在使用源代码管理时是否忽略包文件夹的布尔。默认值是 False。

示例：

```
<solution>
  <add key="disableSourceControlIntegration" value="true" />
</solution>
```

包源节

`packageSources`、`packageSourceCredentials`、`apikeys`、`activePackageSource`、`disabledPackageSources` 和 `trustedSigners` 一起使用，以配置在安装、还原和更新操作过程中 NuGet 如何与包存储库一起工作。

`nuget sources` 命令通常用于管理这些设置，但使用 `nuget setapikey` 命令管理的 `apikeys` 和使用 `nuget trusted-signers` 命令管理的 `trustedSigners` 除外。

请注意，`nuget.org` 的源 URL 是 `https://api.nuget.org/v3/index.json`。

packageSources

列出所有已知包源。在还原操作和任何使用 `PackageReference` 格式的项目中，将忽略此顺序。NuGet 遵循使用 `packages.config` 的项目进行安装和更新操作的源顺序。

“	”
(要分配给包源的名称)	包源的路径或 URL。

示例：

```
<packageSources>
  <add key="nuget.org" value="https://api.nuget.org/v3/index.json" protocolVersion="3" />
  <add key="Contoso" value="https://contoso.com/packages/" />
  <add key="Test Source" value="c:\packages" />
</packageSources>
```

TIP

当给定节点中存在 `<clear />` 时，NuGet 将忽略之前为该节点定义的配置值。阅读有关如何应用设置的详细信息。

packageSourceCredentials

存储源的用户名和密码，通常通过 `-username` 使用 `-password` 和 `nuget sources` 开关指定。默认情况下密码会进行加密，除非还使用了 `-storepasswordincleartext` 选项。

“	”
username	纯文本形式的源用户名。
password	源的加密密码。
cleartextpassword	源的未加密密码。

示例：

在配置文件中，`<packageSourceCredentials>` 元素包含每个适用源名称的子节点（名称中的空格被替换为 `_x0020_`）。也就是说，对于名为“Contoso”和“测试源”的源，使用加密密码时，配置文件包含以下内容：

```
<packageSourceCredentials>
  <Contoso>
    <add key="Username" value="user@contoso.com" />
    <add key="Password" value="..." />
  </Contoso>
  <Test_x0020_Source>
    <add key="Username" value="user" />
    <add key="Password" value="..." />
  </Test_x0020_Source>
</packageSourceCredentials>
```

使用未加密密码时：

```
<packageSourceCredentials>
  <Contoso>
    <add key="Username" value="user@contoso.com" />
    <add key="ClearTextPassword" value="33f!!lloppa" />
  </Contoso>
  <Test_x0020_Source>
    <add key="Username" value="user" />
    <add key="ClearTextPassword" value="hal+9ooo_da!sY" />
  </Test_x0020_Source>
</packageSourceCredentials>
```

apikeys

存储使用 API 密钥身份验证的源的密钥，如使用 [nuget setapikey 命令](#) 设置。

II

I

(源 URL)

加密的 API 密钥。

示例：

```
<apikeys>
  <add key="https://MyRepo/ES/api/v2/package" value="encrypted_api_key" />
</apikeys>
```

disabledPackageSources

标识当前已禁用的源。可能为空。

II

I

(源名称)

指示源是否禁用的布尔。

示例：

```
<disabledPackageSources>
  <add key="Contoso" value="true" />
</disabledPackageSources>

<!-- Empty list -->
<disabledPackageSources />
```

activePackageSource

(仅限于 2.x; 3.x+ 中已弃用)

标识到当前活动的源或指示所有源的聚合。

II

(源名称)或 `All`

如果密钥是源的名称，则值为源路径或 URL。如果为 `All`，值应为 `(Aggregate source)`，从而组合其他未禁用的所有包源。

I

示例：

```
<activePackageSource>
    <!-- Only one active source-->
    <add key="nuget.org" value="https://api.nuget.org/v3/index.json" />

    <!-- All non-disabled sources are active -->
    <add key="All" value="(Aggregate source)" />
</activePackageSource>
```

trustedSigners 部分

存储用于在安装或还原时允许包的可信签名者。当用户将 `signatureValidationMode` 设置为 `require` 时，此列表不能为空。

可以通过 [nuget trusted-signers](#) 命令更新此部分。

架构：

受信任的签名者具有 `certificate` 项的集合，这些项将登记标识给定签名者的所有证书。受信任的签名者可以是 `Author` 或 `Repository`。

受信任的存储库还指定存储库的 `serviceIndex`（必须是有效的 `https` uri），并可以选择指定以分号分隔的 `owners` 列表，以限制更多的受信任的用户。

用于证书指纹的受支持的哈希算法 `SHA256`、`SHA384` 和 `SHA512`。

如果 `certificate` 指定 `allowUntrustedRoot` `true`，则在将证书链作为签名验证的一部分生成时，允许给定的证书链接到不受信任的根。

示例：

```
<trustedSigners>
    <author name="microsoft">
        <certificate fingerprint="3F9001EA83C560D712C24CF213C3D312CB3BFF51EE89435D3430BD06B5D0EECE"
hashAlgorithm="SHA256" allowUntrustedRoot="false" />
    </author>
    <repository name="nuget.org" serviceIndex="https://api.nuget.org/v3/index.json">
        <certificate fingerprint="0E5F38F57DC1BCC806D8494F4F90FBCEDD988B46760709CBEEC6F4219AA6157D"
hashAlgorithm="SHA256" allowUntrustedRoot="false" />
        <owners>microsoft;aspnet;nuget</owners>
    </repository>
</trustedSigners>
```

fallbackPackageFolders 部分

(3.5+) 提供了一种预安装包的方法，以便在回退文件夹中发现包时无需执行任何操作。回退包文件夹与全局包文件夹具有完全相同的文件夹和文件结构：。`nupkg` 存在，并提取所有文件。

此配置的查找逻辑为：

- 查看全局包文件夹，查看是否已下载包/版本。
- 查看后备文件夹中是否有包/版本匹配。

如果查找成功，则无需下载。

如果找不到匹配项，NuGet 将检查文件源，然后检查 http 源，然后下载包。

<code><packageManagement></code>	<code></packageManagement></code>
(后备文件夹的名称)	回退文件夹的路径。

示例：

```
<fallbackPackageFolders>
  <add key="XYZ Offline Packages" value="C:\somePath\someFolder\"/>
</fallbackPackageFolders>
```

packageManagement 部分

设置默认包管理格式，即 `package` 或 `PackageReference`。SDK 样式项目始终使用 `PackageReference`。

<code><packageManagement></code>	<code></packageManagement></code>
<code>format</code>	指示默认包管理格式的布尔值。如果 <code>1</code> ，格式为 <code>PackageReference</code> 。如果 <code>0</code> ，则 <code>format</code> 为包。
<code>disabled</code>	指示是否在第一次安装包时显示提示选择默认包格式的布尔值。 <code>False</code> 隐藏提示。

示例：

```
<packageManagement>
  <add key="format" value="1" />
  <add key="disabled" value="False" />
</packageManagement>
```

使用环境变量

可以在 `nuget.config` 值中使用环境变量 (NuGet 3.4+) 在运行时应用设置。

例如，如果 Windows 上的 `HOME` 环境变量设置为 `c:\users\username`，则配置文件中 `%HOME%\NugetRepository` 的值解析为 `c:\users\username\NugetRepository`。

请注意，必须使用 Windows 样式的环境变量(开头和结尾均为%)即使在 Mac/Linux 上也是如此。配置文件中的 `$HOME/NugetRepository` 将无法解析。在 Mac/Linux 上，`%HOME%\NugetRepository` 的值将解析为 `/home/myStuff/NugetRepository`。

如果未找到环境变量，NuGet 会使用配置文件中的文本值。

示例配置文件

下面是一个示例 `nuget.config` 文件示例，演示了一些设置，包括可选的设置：

```
<?xml version="1.0" encoding="utf-8"?>
```

```

<configuration>
  <config>
    <!--
      Used to specify the default location to expand packages.
      See: nuget.exe help install
      See: nuget.exe help update

      In this example, %PACKAGEHOME% is an environment variable. On Mac/Linux,
      use $PACKAGE_HOME/External as the value.
    -->
    <add key="repositoryPath" value="%PACKAGEHOME%\External" />

    <!--
      Used to specify default source for the push command.
      See: nuget.exe help push
    -->

    <add key="defaultPushSource" value="https://MyRepo/ES/api/v2/package" />

    <!-- Proxy settings -->
    <add key="http_proxy" value="host" />
    <add key="http_proxy.user" value="username" />
    <add key="http_proxy.password" value="encrypted_password" />
  </config>

  <packageRestore>
    <!-- Allow NuGet to download missing packages -->
    <add key="enabled" value="True" />

    <!-- Automatically check for missing packages during build in Visual Studio -->
    <add key="automatic" value="True" />
  </packageRestore>

  <!--
    Used to specify the default Sources for list, install and update.
    See: nuget.exe help list
    See: nuget.exe help install
    See: nuget.exe help update
  -->
  <packageSources>
    <add key="NuGet official package source" value="https://api.nuget.org/v3/index.json" />
    <add key="MyRepo - ES" value="https://MyRepo/ES/nuget" />
  </packageSources>

  <!-- Used to store credentials -->
  <packageSourceCredentials />

  <!-- Used to disable package sources -->
  <disabledPackageSources />

  <!--
    Used to specify default API key associated with sources.
    See: nuget.exe help setApiKey
    See: nuget.exe help push
    See: nuget.exe help mirror
  -->
  <apikeys>
    <add key="https://MyRepo/ES/api/v2/package" value="encrypted_api_key" />
  </apikeys>

  <!--
    Used to specify trusted signers to allow during signature verification.
    See: nuget.exe help trusted-signers
  -->
  <trustedSigners>
    <author name="microsoft">
      <certificate fingerprint="3F9001EA83C560D712C24CF213C3D312CB3BFF51EE89435D3430BD06B5D0EECE"
        hashAlgorithm="SHA256" allowUntrustedRoot="false" />
    </author>
  </trustedSigners>

```

```
<repository name="nuget.org" serviceIndex="https://api.nuget.org/v3/index.json">
    <certificate fingerprint="0E5F38F57DC1BCC806D8494F4F90FBCEDD988B46760709CBEEC6F4219AA6157D"
hashAlgorithm="SHA256" allowUntrustedRoot="false" />
    <owners>microsoft;aspnet;nuget</owners>
</repository>
</trustedSigners>
</configuration>
```

目标框架

2020/3/3 • [Edit Online](#)

NuGet 在各个地方使用目标框架引用，以特别标识和隔离包的框架依赖组件：

- [项目文件](#): 对于 SDK 样式项目，`.csproj` 包含目标框架引用。
- [.nuspec 清单](#): 根据项目的目标框架，包可以指示要包含在项目中的不同包。
- [.nupkg 文件夹名称](#): 包的 `lib` 文件夹内的文件夹可根据目标框架进行命名，每个文件夹都包含适合该框架的 DLL 及其他内容。
- [packages.config](#): 依赖项的 `targetframework` 特性指定要安装的包的变体。

NOTE

计算下方表格的 NuGet 客户端源代码位于以下位置：

- 支持的框架名称：[FrameworkConstants.cs](#)
- Framework 优先级和映射：[DefaultFrameworkMappings.cs](#)

支持的框架

通常按简短的目标框架名字对象或 TFM 引用框架。在 .NET Standard 这也通用化到了 `TM`，以允许单个引用多个框架。

NuGet 客户端支持下表中的框架。等效项显示在括号内 []。请注意，某些工具（如 `dotnet`）可能会在某些文件中使用规范的 TFM 变体。例如，`dotnet pack` 在 `.NETCoreApp2.0` 文件中使用 `.nuspec`，而非 `netcoreapp2.0` 文件。各种 NuGet 客户端工具正确处理这些变体，但是在直接编辑文件时，应始终使用规范的 TFM。

II	II	TFM/TXM
.NET Framework	net	net11
		net20
		net35
		net40
		net403
		net45
		net451
		net452
		net46
		net461

		TFM/TXM
		net462
		net47
		net471
		net472
		net48
Microsoft Store(Windows 应用商店)	netcore	netcore [netcore45]
		netcore45 [win, win8]
		netcore451 [win81]
		netcore50
.NET MicroFramework	netmf	netmf
Windows	win	win [win8, netcore45]
		win8 [netcore45, win]
		win81 [netcore451]
		win10(Windows 10 平台不支持)
Silverlight	sl	sl4
		sl5
Windows Phone (SL)	wp	wp [wp7]
		wp7
		wp75
		wp8
		wp81
Windows Phone (UWP)		wpa81
通用 Windows 平台	uap	uap [uap10.0]
		uap10.0
		uap 10.0 (其中10.0 是使用应用的 目标平台最小版本)

		TFM/TXM
.NET Standard	netstandard	netstandard1.0
		netstandard1.1
		netstandard1.2
		netstandard1.3
		netstandard1.4
		netstandard1.5
		netstandard1.6
		netstandard2.0
		netstandard2.1
.NET Core 应用	netcoreapp	netcoreapp1.0
		netcoreapp1.1
		netcoreapp2.0
		netcoreapp2.1
		netcoreapp2.2
		netcoreapp3.0
		netcoreapp3.1
Tizen	tizen	tizen3
		tizen4

弃用的框架

以下框架已弃用。定位这些框架的包应迁移到指明的替代框架。

aspnet50	netcoreapp
aspnetcore50	
dnxcore50	
dnx	

dnx45	
dnx451	
dnx452	
dotnet	netstandard
dotnet50	
dotnet51	
dotnet52	
dotnet53	
dotnet54	
dotnet55	
dotnet56	
winrt	win

优先级

许多框架相互关联且彼此兼容，但不一定是等效的：

II	III
uap(通用 Windows 平台)	win81
	wpa81
	netcore50
win (Microsoft Store)	winrt

NET Standard

.NET Standard 简化了二进制兼容框架之间的引用，允许单个目标框架引用其他框架的组合。（有关背景信息，请参阅 [.NET 入门](#)。）

[NuGet 获取最新框架工具](#) 模拟用于从基于项目框架的包中的许多可用框架资产中选择一个框架的 NuGet 逻辑。

NuGet 3.3 及更早版本应该使用 `dotnet` 系列的名字对象；v3.4 及更高版本应该使用 `netstandard` 名字对象语法。

可移植类库

WARNING

建议不要使用 PCL。尽管支持 PCL，但包创建者反而应支持 netstandard。.NET 平台标准是 PCL 的发展，它使用与便携 $a + b + c$ 名字对象相关的单个名字对象来表示跨平台的二进制可移植性。

若要定义一个引用多个子目标框架的目标框架，请使用 `portable` 关键字作为所引用框架列表的前缀。避免人为地包含非直接编译的额外框架，因为可能会导致这些框架中出现意外的负面效果。

由第三方定义的附加框架提供了与其他环境的兼容性，这些环境可通过此方式进行访问。此外，还有速记配置文件编号，可用于以 `Profile#` 形式引用相关框架的组合，但不建议通过此方法引用这些编号，因为这会降低文件夹和 `.nuspec` 的可读性。

子目标框架	框架	前缀	.NET STANDARD
Profile2	.NETFramework 4.0	portable-net40+win8+sl4+wp7	
	Windows 8.0		
	Silverlight 4.0		
	WindowsPhone 7.0		
Profile3	.NETFramework 4.0	portable-net40+sl4	
	Silverlight 4.0		
Profile4	.NETFramework 4.5	portable-net45+sl4+win8+wp7	
	Silverlight 4.0		
	Windows 8.0		
	WindowsPhone 7.0		
Profile5	.NETFramework 4.0	portable-net40+win8	
	Windows 8.0		
Profile6	.NETFramework 4.0.3	portable-net403+win8	
	Windows 8.0		
Profile7	.NETFramework 4.5	portable-net45+win8	netstandard1.1
	Windows 8.0		
Profile14	.NETFramework 4.0	portable-net40+sl5	
	Silverlight 5.0		

			.NET STANDARD
Profile18	.NETFramework 4.0.3	portable-net403+sl4	
	Silverlight 4.0		
Profile19	.NETFramework 4.0.3	portable-net403+sl5	
	Silverlight 5.0		
Profile23	.NETFramework 4.5	portable-net45+sl4	
	Silverlight 4.0		
Profile24	.NETFramework 4.5	portable-net45+sl5	
	Silverlight 5.0		
Profile31	Windows 8.1	portable-win81+wp81	netstandard1.0
	WindowsPhone 8.1 (SL)		
Profile32	Windows 8.1	portable-win81+wpa81	netstandard1.2
	WindowsPhone 8.1 (UWP)		
Profile36	.NETFramework 4.0	portable-net40+sl4+win8+wp8	
	Silverlight 4.0		
	Windows 8.0		
	WindowsPhone 8.0 (SL)		
Profile37	.NETFramework 4.0	portable-net40+sl5+win8	
	Silverlight 5.0		
	Windows 8.0		
Profile41	.NETFramework 4.0.3	portable-net403+sl4+win8	
	Silverlight 4.0		
	Windows 8.0		
Profile42	.NETFramework 4.0.3	portable-net403+sl5+win8	
	Silverlight 5.0		

			.NET STANDARD
	Windows 8.0		
Profile44	.NETFramework 4.5.1	portable-net451+win81	netstandard1.2
	Windows 8.1		
Profile46	.NETFramework 4.5	portable-net45+sl4+win8	
	Silverlight 4.0		
	Windows 8.0		
Profile47	.NETFramework 4.5	portable-net45+sl5+win8	
	Silverlight 5.0		
	Windows 8.0		
Profile49	.NETFramework 4.5	portable-net45+wp8	netstandard1.0
	WindowsPhone 8.0 (SL)		
Profile78	.NETFramework 4.5	portable-net45+win8+wp8	netstandard1.0
	Windows 8.0		
	WindowsPhone 8.0 (SL)		
Profile84	WindowsPhone 8.1	portable-wp81+wpa81	netstandard1.0
	WindowsPhone 8.1 (UWP)		
Profile88	.NETFramework 4.0	portable-net40+sl4+win8+wp75	
	Silverlight 4.0		
	Windows 8.0		
	WindowsPhone 7.5		
Profile92	.NETFramework 4.0	portable-net40+win8+wpa81	
	Windows 8.0		
	WindowsPhone 8.1 (UWP)		
Profile95	.NETFramework 4.0.3	portable-net403+sl4+win8+wp7	

			.NET STANDARD
	Silverlight 4.0		
	Windows 8.0		
	WindowsPhone 7.0		
Profile96	.NETFramework 4.0.3	portable-net403+sl4+win8+wp75	
	Silverlight 4.0		
	Windows 8.0		
	WindowsPhone 7.5		
Profile102	.NETFramework 4.0.3	portable-net403+win8+wpa81	
	Windows 8.0		
	WindowsPhone 8.1 (UWP)		
Profile104	.NETFramework 4.5	portable-net45+sl4+win8+wp75	
	Silverlight 4.0		
	Windows 8.0		
	WindowsPhone 7.5		
Profile111	.NETFramework 4.5	portable-net45+win8+wpa81	netstandard1.1
	Windows 8.0		
	WindowsPhone 8.1 (UWP)		
Profile136	.NETFramework 4.0	portable-net40+sl5+win8+wp8	
	Silverlight 5.0		
	Windows 8.0		
	WindowsPhone 8.0 (SL)		
Profile143	.NETFramework 4.0.3	portable-net403+sl4+win8+wp8	
	Silverlight 4.0		

			.NET STANDARD
	Windows 8.0		
	WindowsPhone 8.0 (SL)		
Profile147	.NETFramework 4.0.3	portable-net403+sl5+win8+wp8	
	Silverlight 5.0		
	Windows 8.0		
	WindowsPhone 8.0 (SL)		
Profile151	.NETFramework 4.5.1	portable-net451+win81+wpa81	netstandard1.2
	Windows 8.1		
	WindowsPhone 8.1 (UWP)		
Profile154	.NETFramework 4.5	portable-net45+sl4+win8+wp8	
	Silverlight 4.0		
	Windows 8.0		
	WindowsPhone 8.0 (SL)		
Profile157	Windows 8.1	portable-win81+wp81+wpa81	netstandard1.0
	WindowsPhone 8.1 (SL)		
	WindowsPhone 8.1 (UWP)		
Profile158	.NETFramework 4.5	portable-net45+sl5+win8+wp8	
	Silverlight 5.0		
	Windows 8.0		
	WindowsPhone 8.0 (SL)		
Profile225	.NETFramework 4.0	portable-net40+sl5+win8+wpa81	
	Silverlight 5.0		
	Windows 8.0		

			.NET STANDARD
	WindowsPhone 8.1 (UWP)		
Profile240	.NETFramework 4.0.3	portable-net403+sl5+win8+wpa8	
	Silverlight 5.0		
	Windows 8.0		
	WindowsPhone 8.1 (UWP)		
Profile255	.NETFramework 4.5	portable-net45+sl5+win8+wpa81	
	Silverlight 5.0		
	Windows 8.0		
	WindowsPhone 8.1 (UWP)		
Profile259	.NETFramework 4.5	portable-net45+win8+wpa81+wp8	netstandard1.0
	Windows 8.0		
	WindowsPhone 8.1 (UWP)		
	WindowsPhone 8.0 (SL)		
Profile328	.NETFramework 4.0	portable-net40+sl5+win8+wpa81+wp8	
	Silverlight 5.0		
	Windows 8.0		
	WindowsPhone 8.1 (UWP)		
	WindowsPhone 8.0 (SL)		
Profile336	.NETFramework 4.0.3	portable-net403+sl5+win8+wpa81+wp8	
	Silverlight 5.0		
	Windows 8.0		
	WindowsPhone 8.1 (UWP)		

			.NET STANDARD
	WindowsPhone 8.0 (SL)		
Profile344	.NETFramework 4.5	portable-net45+sl5+win8+wpa81+wp8	
	Silverlight 5.0		
	Windows 8.0		
	WindowsPhone 8.1 (UWP)		
	WindowsPhone 8.0 (SL)		

另外，面向 Xamarin 的 NuGet 包可以使用 Xamarin 定义的其他框架。请参阅[创建适用于 Xamarin 的 NuGet 包](#)。

		.NET STANDARD
monoandroid	Mono 支持 Android OS	netstandard1.4
monotouch	Mono 支持 iOS	netstandard1.4
monomac	Mono 支持 OSX	netstandard1.4
xamarinios	支持适用于 iOS 的 Xamarin	netstandard1.4
xamarinmac	支持适用于 Mac 的 Xamarin	netstandard1.4
xamarinpsthree	支持 Playstation 3 上的 Xamarin	netstandard1.4
xamarinpsthreefour	支持 Playstation 4 上的 Xamarin	netstandard1.4
xamarinpssvita	支持 PS Vita 上的 Xamarin	netstandard1.4
xamarinwatchos	适用于 Watch OS 的 Xamarin	netstandard1.4
xamarintvos	适用于 TV OS 的 Xamarin	netstandard1.4
xamarinxboxthreesixty	适用于 XBox 360 的 Xamarin	netstandard1.4
xamarinxboxone	适用于 XBox One 的 Xamarin	netstandard1.4

NOTE

Stephen Cleary 创建了一种工具，用于列出支持的 PCL，该工具可在其文章 [Framework profiles in .NET \(.NET 中的框架配置文件\)](#) 中找到。

作为 MSBuild 目标的 NuGet 包和还原

2020/3/19 • [Edit Online](#)

NuGet 4.0 +

使用 [PackageReference](#) 格式, NuGet 4.0 + 可以直接将所有清单元数据存储在项目文件中, 而不是使用单独的 `.nuspec` 文件。

如下所述, 对于 MSBuild 15.1+, NuGet 还是具有 `pack` 目标和 `restore` 目标的一等 MSBuild 公民。借助这些目标, 你可以像使用任何其他 MSBuild 任务或目标一样使用 NuGet。有关使用 MSBuild 创建 NuGet 包的说明, 请参阅[使用 Msbuild 创建 nuget 包](#)。(对于 Nuget 3.x 及更早版本, 通过 NuGet CLI 使用 `pack` 和 `restore` 命令。)

目标生成顺序

由于 `pack` 和 `restore` 为 MSBuild 目标, 因此可以访问它们以增强工作流。例如, 假设希望在打包后将包复制到网络共享。此操作可通过向项目文件中添加以下内容来完成:

```
<Target Name="CopyPackage" AfterTargets="Pack">
  <Copy
    SourceFiles="$(OutputPath)..\$(PackageId).$(PackageVersion).nupkg"
    DestinationFolder="\\myshare\packageshare\"/>
</Target>
```

同样, 你可以编写 MSBuild 任务、编写自己的目标并使用 MSBuild 任务中的 NuGet 属性。

NOTE

`$(OutputPath)` 是相对的, 需要从项目根目录运行命令。

包目标

对于使用 [PackageReference](#) 格式的 .NET Standard 项目, 使用 `msbuild -t:pack` 从项目文件中绘制输入, 以用于创建 NuGet 包。

下表描述了可以添加到项目文件第一个 `<PropertyGroup>` 节点中的 MSBuild 属性。在 Visual Studio 2017 及更高版本中, 通过右键单击项目并选择上下文菜单上的“编辑 {project_name}”, 即可轻松进行这些编辑。为方便起见, 表由 `.nuspec` 文件中的等效属性组织。

请注意, `Owners` 的 `Summary` 和 `.nuspec` 属性不受 MSBuild 支持。

MSBUILD	MSBUILD	MSBUILD	MSBUILD
ID	Packageld	AssemblyName	MSBuild 的 \$(AssemblyName)
版本	PackageVersion	版本	这与 SemVer 兼容, 例 如, “1.0.0”、“1.0.0- beta”或“1.0.0-beta-00345”

NUSPEC 属性	MSBuild 属性	值	说明
VersionPrefix	PackageVersionPrefix	empty	设置 PackageVersion 会覆盖 PackageVersionPrefix
VersionSuffix	PackageVersionSuffix	empty	MSBuild 的 \$(VersionSuffix)。设置 PackageVersion 会覆盖 PackageVersionSuffix
Authors	Authors	当前用户的用户名	
所有者	空值	NuSpec 中不存在	
标题	标题	Packageld	
说明	说明	“包描述”	
版权信息	版权信息	empty	
RequireLicenseAcceptance	PackageRequireLicenseAcceptance	false	
license	PackageLicenseExpression	empty	对应于 <code><license type="expression"></code>
license	PackageLicenseFile	empty	对应到 <code><license type="file"></code> 。 需要显式打包所引用的许可证文件。
LicenseUrl	PackageLicenseUrl	empty	<code>PackageLicenseUrl</code> 已弃用, 请使用 PackageLicenseExpression 或 PackageLicenseFile 属性
ProjectUrl	PackageProjectUrl	empty	
图标	PackageIcon	empty	需要显式打包所引用的图标图像文件。
IconUrl	PackageIconUrl	empty	为了获得最佳的下层体验, 除了 <code>PackageIcon</code> 之外, 还应指定 <code>PackageIconUrl</code> 。 更长的期限, <code>PackageIconUrl</code> 将被弃用。
Tags	PackageTags	empty	使用分号分隔标记。
ReleaseNotes	PackageReleaseNotes	empty	

NUSPEC 属性	MSBUILD 属性	示例	说明
存储库/Url	RepositoryUrl	empty	用于克隆或检索源代码的存储库 URL。示例： https://github.com/NuGet/NuGet.Client.git
存储库/类型	RepositoryType	empty	存储库类型。示例：git、tfs。
存储库/分支	RepositoryBranch	empty	可选存储库分支信息。还必须为此属性指定要包含的 RepositoryUrl。示例： master (NuGet 4.7.0 +)
存储库/提交	RepositoryCommit	empty	可选的存储库提交或更改集，指示针对其生成包的源。还必须为此属性指定要包含的 RepositoryUrl。示例： 0e4d1b598f350b3dc67501 8d539114d1328189ef (NuGet 4.7.0 +)
PackageType	<pre><PackageType>DotNetCliTool, 1.0.0.0;Dependency, 2.0.0.0</PackageType></pre>		
总结	不支持		

包目标输入

- IsPackable
- SuppressDependenciesWhenPacking
- PackageVersion
- PackageId
- Authors
- 说明
- 版权信息
- PackageRequireLicenseAcceptance
- DevelopmentDependency
- PackageLicenseExpression
- PackageLicenseFile
- PackageLicenseUrl
- PackageProjectUrl
- PackageIconUrl
- PackageReleaseNotes
- PackageTags
- PackageOutputPath
- IncludeSymbols
- IncludeSource
- PackageTypes
- IsTool
- RepositoryUrl

- RepositoryType
- RepositoryBranch
- RepositoryCommit
- NoPackageAnalysis
- MinClientVersion
- IncludeBuildOutput
- IncludeContentInPack
- BuildOutputTargetFolder
- ContentTargetFolders
- NuspecFile
- NuspecBasePath
- NuspecProperties

包方案

禁止依赖项

若要取消生成的 NuGet 包中的包依赖项，请将 `SuppressDependenciesWhenPacking` 设置为 `true` 这样将允许跳过生成的 nupkg 文件中的所有依赖项。

PackageIconUrl

`PackageIconUrl` 将被弃用，以支持新的 `PackageIcon` 属性。

从 NuGet 5.3 & Visual Studio 2019 版本16.3 开始，如果包元数据仅指定 `PackageIconUrl`，`pack` 将引发 [NU5048](#) 警告。

PackageIcon

TIP

应同时指定 `PackageIcon` 和 `PackageIconUrl`，以保持与尚不支持 `PackageIcon` 的客户端和源的向后兼容性。在未来版本中，Visual Studio 将支持来自基于文件夹的源的包 `PackageIcon`。

打包图标图像文件

打包图标图像文件时，需要使用 `PackageIcon` 属性来指定包路径，相对于包的根。此外，还需要确保文件已包含在包中。图像文件大小限制为 1 MB。支持的文件格式包括 JPEG 和 PNG。建议使用128x128 的图像分辨率。

例如：

```
<PropertyGroup>
  ...
  <PackageIcon>icon.png</PackageIcon>
  ...
</PropertyGroup>

<ItemGroup>
  ...
  <None Include="images\icon.png" Pack="true" PackagePath="/" />
  ...
</ItemGroup>
```

包图标示例。

对于 nuspec 等效项，请查看 [图标的 nuspec 参考](#)。

输出程序集

`nuget pack` 会复制扩展名为 `.exe`、`.dll`、`.xml`、`.winmd`、`.json` 和 `.pri` 的输出文件。复制的输出文件取决于 MSBuild 从 `BuiltOutputProjectGroup` 目标提供的内容。

在项目文件或命令行中，有两种 MSBuild 属性可用于控制输出程序集的位置：

- `IncludeBuildOutput`：确定包中是否应包括生成输出程序集的布尔值。
- `BuildOutputTargetFolder`：指定应放置输出程序集的文件夹。输出程序集（和其他输出文件）会复制到各自的框架文件夹中。

包引用

请参阅[项目文件中的包引用](#)

项目到项目的引用

默认情况下，项目到项目的引用被视为 nuget 包引用，例如：

```
<ProjectReference Include=".\\UwpLibrary2\\UwpLibrary2.csproj"/>
```

也可将以下元数据添加到项目引用：

```
<IncludeAssets>
<ExcludeAssets>
<PrivateAssets>
```

在包中添加内容

若要添加内容，请将额外的元数据添加到现有的 `<Content>` 项。默认情况下，类型“Content”的所有内容都包括在包中，除非使用以下条目替代：

```
<Content Include=".\\win7-x64\\libuv.txt">
<Pack>false</Pack>
</Content>
```

默认情况下，所有内容都添加到包中 `content` 和 `contentFiles\\any\\<target_framework>` 文件夹根目录，并保留相对文件夹结构，除非指定包路径：

```
<Content Include=".\\win7-x64\\libuv.txt">
<Pack>true</Pack>
<PackagePath>content\\myfiles\\</PackagePath>
</Content>
```

如果仅希望将所有内容都复制到特定根文件夹（而不是 `content` 和 `contentFiles`），可使用 MSBuild 属性 `ContentTargetFolders`，其默认值为“`content;contentFiles`”，但可以设置为任何其他文件夹名称。请注意，基于 `ContentTargetFolders`，仅在 `contentFiles\\any\\<target_framework>` 中指定“`contentFiles`”会将文件置于 `contentFiles\\language\\<target_framework>` 或 `buildAction` 下。

`PackagePath` 可以是一组以分号分隔的目标路径。指定空的包路径会将文件添加到包的根目录。例如，以下操作会将 `libuv.txt` 添加到 `content\\myfiles`、`content\\samples` 和包的根目录：

```
<Content Include=".\\win7-x64\\libuv.txt">
<Pack>true</Pack>
<PackagePath>content\\myfiles;content\\sample;;</PackagePath>
</Content>
```

另外还有 MSBuild 属性 `$(IncludeContentInPack)`，默认值为 `true`。如果在任何项目中将此值设置为 `false`，则该项目的内容不会包括在 nuget 包中。

其他可在任何上述项上设置的特定于包的元数据包括 `<PackageCopyToOutput>` 和 `<PackageFlatten>`，后者在输出 nuspec 中的 `CopyToOutput` 条目上设置 `Flatten` 和 `contentFiles` 值。

NOTE

除了“Content”项，`<Pack>` 和 `<PackagePath>` 元数据还可以在具有 `Compile`、`EmbeddedResource`、`ApplicationDefinition`、`Page`、`Resource`、`SplashScreen`、`DesignData`、`DesignDataWithDesignTimeCreateableTypes`、`CodeAnalysisDictionary`、`AndroidAsset`、`AndroidResource`、`BundleResource` 或 `None` 生成操作的文件上进行设置。

使用通配模式时，对于将包的文件名追加到包路径，包路径必须以文件夹分隔符结尾，否则包路径将被视为包括文件名的完整路径。

IncludeSymbols

使用 `MSBuild -t:pack -p:IncludeSymbols=true` 时，相应的 `.pdb` 文件将随其他输出文件 (`.dll`、`.exe`、`.winmd`、`.xml`、`.json`、`.pri`) 一起复制。请注意，设置 `IncludeSymbols=true` 会创建常规包和符号包。

IncludeSource

这与 `IncludeSymbols` 相同，但它还会连同 `.pdb` 文件复制源文件。类型为 `Compile` 的所有文件都会复制到 `src\<ProjectName>\`，并保留生成包中的相对路径文件夹结构。对于将 `ProjectReference` 设置为 `TreatAsPackageReference` 的任何 `false` 的源文件也是如此。

如果类型为 `Compile` 的文件位于项目文件夹外，则它只添加到了 `src\<ProjectName>\`。

打包许可证表达式或许可证文件

使用许可证表达式时，应使用 `PackageLicenseExpression` 属性。[许可证表达式示例](#)。

```
<PropertyGroup>
  <PackageLicenseExpression>MIT</PackageLicenseExpression>
</PropertyGroup>
```

[了解有关 NuGet.org 所接受的许可证表达式和许可证的详细信息。](#)

打包许可证文件时，需要使用 `PackageLicenseFile` 属性来指定相对于包根的包路径。此外，还需要确保文件已包含在包中。例如：

```
<PropertyGroup>
  <PackageLicenseFile>LICENSE.txt</PackageLicenseFile>
</PropertyGroup>

<ItemGroup>
  <None Include="licenses\LICENSE.txt" Pack="true" PackagePath="" />
</ItemGroup>
```

[许可证文件示例。](#)

IsTool

使用 `MSBuild -t:pack -p:IsTool=true` 时，所有输出文件（如[输出程序集](#)方案中所指定）都被复制到 `tools` 文件夹，而不是 `lib` 文件夹。请注意，这不同于 `DotNetCliTool`，后者通过在 `PackageType` 文件中设置 `.csproj` 进行指定。

使用 .nuspec 打包

尽管建议你改为在项目文件中包含通常位于 `.nuspec` 文件中的所有属性，但你可以选择使用 `.nuspec` 文件来

打包项目。对于使用 `PackageReference` 的非 SDK 样式项目，必须导入 `NuGet.Build.Tasks.Pack.targets` 以便可以执行 pack 任务。你仍需要还原项目，然后才能打包 nuspec 文件。(默认情况下，SDK 样式项目包括包目标。)

项目文件的目标框架不相关，并且不会在对 nuspec 进行打包时使用。以下三个 MSBuild 属性与使用 `.nuspec` 的打包相关：

1. `NuspecFile`：用于打包的 `.nuspec` 文件的相对或绝对路径。
2. `NuspecProperties`：键=值对的分号分隔列表。由于 MSBuild 命令行分析工作的方式，必须指定多个属性，如下所示：`-p:NuspecProperties=\"key1=value1;key2=value2\"`。
3. `NuspecBasePath`：`.nuspec` 文件的基路径。

如果使用 `dotnet.exe` 打包项目，请使用类似于下面的命令：

```
dotnet pack <path to .csproj file> -p:NuspecFile=<path to nuspec file> -p:NuspecProperties=<> -p:NuspecBasePath=<Base path>
```

如果使用 MSBuild 打包项目，请使用类似于下面的命令：

```
msbuild -t:pack <path to .csproj file> -p:NuspecFile=<path to nuspec file> -p:NuspecProperties=<> -p:NuspecBasePath=<Base path>
```

请注意，使用 dotnet 或 msbuild 打包 nuspec 在默认情况下也会导致生成项目。可以通过将 `--no-build` 属性传递到 dotnet 来避免这种情况，这相当于在项目文件中设置 `<NoBuild>true</NoBuild>`，以及在项目文件中设置 `<IncludeBuildOutput>false</IncludeBuildOutput>`。

用于打包 nuspec 文件的 `.csproj` 文件的示例如下：

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>netstandard2.0</TargetFramework>
    <NoBuild>true</NoBuild>
    <IncludeBuildOutput>false</IncludeBuildOutput>
    <NuspecFile>PATH_TO_NUSPEC_FILE</NuspecFile>
    <NuspecProperties>add nuspec properties here</NuspecProperties>
    <NuspecBasePath>optional to provide</NuspecBasePath>
  </PropertyGroup>
</Project>
```

用于创建自定义包的高级扩展点

`pack` 目标提供了两个扩展点，这些点在内部特定于目标框架的内部版本中运行。扩展点支持包括特定于目标框架的内容和程序集到包中：

- `TargetsForTfmSpecificBuildOutput` 目标：用于 `lib` 文件夹内的文件或使用 `BuildOutputTargetFolder` 指定的文件夹。
- `TargetsForTfmSpecificContentInPackage` 目标：用于 `BuildOutputTargetFolder` 以外的文件。

TargetsForTfmSpecificBuildOutput

编写一个自定义目标并将其指定为 `$(TargetsForTfmSpecificBuildOutput)` 属性的值。对于需要进入 `BuildOutputTargetFolder` 的任何文件(默认情况下 lib)，目标应将这些文件写入 `ItemGroup BuildOutputInPackage` 并设置以下两个元数据值：

- `FinalOutputPath`：文件的绝对路径；如果未提供，则标识用于计算源路径。
- `TargetPath`：(可选)当文件需要进入到 `lib\<TargetFramework>` 中的子文件夹时设置，如在其各自的区域性文件夹下的附属程序集。默认为文件的名称。

示例：

```
<PropertyGroup>

<TargetsForTfmSpecificBuildOutput>$(TargetsForTfmSpecificBuildOutput);GetMyPackageFiles</TargetsForTfmSpec
ificBuildOutput>
</PropertyGroup>

<Target Name="GetMyPackageFiles">
  <ItemGroup>
    <BuildOutputInPackage Include="$(OutputPath)cs\$(AssemblyName).resources.dll">
      <TargetPath>cs</TargetPath>
    </BuildOutputInPackage>
  </ItemGroup>
</Target>
```

TargetsForTfmSpecificContentInPackage

编写一个自定义目标并将其指定为 `$(TargetsForTfmSpecificContentInPackage)` 属性的值。对于要包含在包中的任何文件，目标应将这些文件写入 `ItemGroup TfmspecificPackageFile` 并设置以下可选的元数据：

- `PackagePath`：应在包中输出文件的路径。如果将多个文件添加到同一个包路径，NuGet 将发出警告。
- `BuildAction`：要分配给文件的生成操作，只有当包路径在 `contentFiles` 文件夹中时才是必需的。默认值为 "None"。

示例：

```
<PropertyGroup>

<TargetsForTfmSpecificContentInPackage>$(TargetsForTfmSpecificContentInPackage);CustomContentTarget</TargetsForTfmSpecificContentInPackage>
</PropertyGroup>

<Target Name="CustomContentTarget">
  <ItemGroup>
    <TfmspecificPackageFile Include="abc.txt">
      <PackagePath>mycontent/$(TargetFramework)</PackagePath>
    </TfmspecificPackageFile>
    <TfmspecificPackageFile Include="Extensions/ext.txt" Condition="'$(>TargetFramework)' == 'net46'">
      <PackagePath>net46content</PackagePath>
    </TfmspecificPackageFile>
  </ItemGroup>
</Target>
```

还原目标

`MSBuild -t:restore` (`nuget restore` 和 `dotnet restore` 与 .NET Core 项目一起使用) 会还原项目文件中引用的包，如下所示：

1. 读取所有项目到项目的引用
2. 读取项目属性以查找中间文件夹和目标框架
3. 将 MSBuild 数据传递到 NuGet。生成 .dll
4. 运行还原
5. 下载包
6. 编写资产文件、目标和属性

`restore` 目标仅适用于使用 `PackageReference` 格式的项目。它不适用于使用 `packages.config` 格式的项目；请改用 [nuget 还原](#)。

还原属性

其他的还原设置可能来自项目文件中的 MSBuild 属性。还可以从命令行使用 `-p:` 关开关设置值(请参阅以下示例)。

PROPERTIES	
RestoreSources	以分号分隔的包源列表。
RestorePackagesPath	用户包文件夹路径。
RestoreDisableParallel	将下载限制为一次一个。
RestoreConfigFile	要应用的 <code>Nuget.Config</code> 文件的路径。
RestoreNoCache	如果为 <code>true</code> , 则避免使用缓存的包。请参阅 管理全局包和缓存文件夹 。
RestoreIgnoreFailedSources	如果为“ <code>true</code> ”, 则忽略失败或丢失的包源。
RestoreFallbackFolders	后备文件夹, 其使用方式与使用“用户包”文件夹的方式相同。
RestoreAdditionalProjectSources	要在还原期间使用的其他源。
RestoreAdditionalProjectFallbackFolders	要在还原期间使用的其他回退文件夹。
RestoreAdditionalProjectFallbackFoldersExcludes	排除 <code>RestoreAdditionalProjectFallbackFolders</code> 中指定的回退文件夹
RestoreTaskAssemblyFile	<code>NuGet.Build.Tasks.dll</code> 的路径。
RestoreGraphProjectInput	要还原的以分号分隔的项目列表, 其中应包含绝对路径。
RestoreUseSkipNonexistentTargets	通过 MSBuild 收集项目时, 它将确定是否使用 <code>SkipNonexistentTargets</code> 优化来收集这些项目。如果未设置, 则默认为 <code>true</code> 。如果无法导入项目的目, 则结果是一个故障快速的行为。
MSBuildProjectExtensionsPath	Output 文件夹, 默认为 <code>BaseIntermediateOutputPath</code> 和 <code>obj</code> 文件夹。
RestoreForce	在基于 <code>PackageReference</code> 的项目中, 将强制解析所有依赖项, 即使上次还原已成功。指定此标志与删除 <code>project.assets.json</code> 文件类似。这不会绕过 http 缓存。
RestorePackagesWithLockFile	选择使用锁定文件。
RestoreLockedMode	在锁定模式下运行还原。这意味着, 还原将不会重新评估依赖关系。
NuGetLockFilePath	锁定文件的自定义位置。默认位置为项目旁边, 名为 <code>packages.lock.json</code> 。

PROPERTIES

RestoreForceEvaluate

强制执行还原，重新计算依赖项并更新锁定文件，而不会出现任何警告。

示例

命令行：

```
msbuild -t:restore -p:RestoreConfigFile=<path>
```

项目文件：

```
<PropertyGroup>
    <RestoreIgnoreFailedSource>true</RestoreIgnoreFailedSource>
</PropertyGroup>
```

还原输出

还原会在生成 `obj` 文件夹中创建以下文件：

project.assets.json

包含所有包引用的依赖项关系图。

{projectName}.projectFileExtension.nuget.g.props

包中包含的对 MSBuild 属性的引用

{projectName}.projectFileExtension.nuget.g.targets

包中包含的对 MSBuild 目标的引用

通过一个 MSBuild 命令进行还原和生成

由于 NuGet 可以还原引入 MSBuild 目标和属性的包，因此，还原和生成评估将以不同的全局属性运行。这意味着，以下操作将具有不可预测且通常不正确的行为。

```
msbuild -t:restore,build
```

相反，推荐的方法是：

```
msbuild -t:build -restore
```

相同的逻辑也适用于类似于 `build` 的其他目标。

PackageTargetFallback

`PackageTargetFallback` 元素可用于指定要在还原包时使用的一组兼容目标。旨在允许使用 dotnet TxM 的包与未声明 dotnet TxM 的兼容包结合使用。也就是说，如果项目使用 dotnet TxM，那么所依赖的所有包也必须有 dotnet TxM，除非将 `<PackageTargetFallback>` 添加到项目中，以允许非 dotnet 平台与 dotnet 兼容。

例如，如果项目使用的是 `netstandard1.6` TxM，且从属包仅包含 `lib/net45/a.dll` 和 `lib/portable-net45+win81/a.dll`，则项目将无法生成。如果需要的是后一种 DLL，则可以按照如下所示添加 `PackageTargetFallback`，声明 `portable-net45+win81` DLL 是兼容的：

```
<PackageTargetFallback Condition="'$(TargetFramework)'=='netstandard1.6'">
    portable-net45+win81
</PackageTargetFallback>
```

若要声明项目中所有目标的回退, 请停止 `Condition` 属性。还可以通过包括 `PackageTargetFallback` 扩展任何现有 `$(PackageTargetFallback)`, 如下所示:

```
<PackageTargetFallback>
    $(PackageTargetFallback);portable-net45+win81
</PackageTargetFallback >
```

从还原图中替换一个库

如果还原引入了错误的程序集, 可以排除包默认选项, 并将其替换为自己的选项。首先使用顶级 `PackageReference`, 排除所有资产:

```
<PackageReference Include="Newtonsoft.Json" Version="9.0.1">
    <ExcludeAssets>All</ExcludeAssets>
</PackageReference>
```

接下来, 将自己的引用添加到适当的 DLL 本地副本:

```
<Reference Include="Newtonsoft.Json.dll" />
```

dotnet CLI 命令

2019/7/19 • [Edit Online](#)

dotnet 命令行接口 (CLI) 在 Windows、Mac OS X 和 Linux 上运行，它提供了许多必要的命令，例如安装、还原和发布包。如果 dotnet 满足你的需求，则无需使用 nuget.exe。

有关使用这些命令来使用包的示例，请参阅[使用 DOTNET CLI 安装和管理包](#)。有关使用这些命令创建包的示例，请参阅[使用 DOTNET CLI 创建和发布包](#)。

有关 dotnet CLI 的完整命令参考，请参阅[.net Core 命令行接口 \(CLI\) 工具](#)。

包消耗

- [dotnet 添加包](#):向项目文件添加包引用，然后运行 dotnet restore 以安装包。
- [删除包 dotnet](#):从项目文件中删除包引用。
- [dotnet restore](#):还原项目的依赖项和工具。从 NuGet 4.0 开始，它运行与 nuget restore 相同的代码。
- [dotnet nuget 局部变量](#):列出全局包、*http* 缓存和临时文件夹的位置，并清除这些文件夹的内容。
- [dotnet new nugetconfig](#): nuget.config 创建文件以配置 NuGet 的行为。

包创建

- [dotnet 包](#):将代码打包到 NuGet 包中。
- [dotnet nuget 推送](#):将包发布到 NuGet 服务器。适用于 nuget.org、Azure Artifacts 和[第三方 nuget 服务器](#)。
- [dotnet nuget 删除](#):从 NuGet 服务器删除或取消列出包。适用于 nuget.org、Azure Artifacts 和[第三方 nuget 服务器](#)。

NuGet CLI 参考

2019/7/19 • [Edit Online](#)

NuGet 命令行接口 (CLI) `nuget.exe` 提供了完整的 nuget 功能, 可用于安装、创建、发布和管理包, 而无需对项目文件进行任何更改。

若要使用任何命令, 请打开命令窗口或 bash shell, 然后 `nuget` 运行命令和相应的选项, `nuget help pack` 如 (以查看 pack 命令的帮助)。

本文档反映了最新版本的 NuGet CLI。有关所使用的任何给定版本的确切详细信息, 请 `nuget help` 运行以获取所需的命令。

要了解如何在 `nuget.exe` CLI 中使用基本命令, 请参阅[使用 nuget.exe CLI 安装并使用包](#)。

安装 nuget.exe

Windows

NOTE

NuGet.exe 5.0 及更高版本需要 .NET Framework 4.7.2 或更高版本才能执行。

1. 请访问 [nuget.org/downloads](https://www.nuget.org/downloads), 并选择 NuGet 3.3 或更高版本 (2.8.6 与 Mono 不兼容)。始终建议使用最新版。若要将包发布到 nuget.org, 版本至少必须是 4.1.0。
2. 每次下载都直接下载 `nuget.exe` 文件。让浏览器将文件保存到选定文件夹。此文件不是安装程序; 如果直接在浏览器中运行, 就不会看到任何内容。
3. 将文件夹添加到 `nuget.exe` 中放置 PATH 环境变量的位置, 这样就可以从任意位置使用 CLI 工具。

macOS/Linux

行为可能因 OS 分发版本略有不同。

1. 安装 [Mono 4.4.2 或更高版本](#)。
2. 在 shell 提示符处, 执行下列命令:

```
# Download the latest stable `nuget.exe` to `/usr/local/bin`
sudo curl -o /usr/local/bin/nuget.exe https://dist.nuget.org/win-x86-commandline/latest/nuget.exe
```

3. 通过将以下脚本添加到 OS 的相应文件来创建别名(通常为 `~/.bash_aliases` 或 `~/.bash_profile`):

```
# Create as alias for nuget
alias nuget="mono /usr/local/bin/nuget.exe"
```

4. 重载 shell。通过输入 `nuget` (而不使用任何参数) 来测试安装。应该会看到 NuGet CLI 帮助。

TIP

若要使 NuGet CLI 在 Visual Studio 的包管理器控制台内可用, 请参阅[控制台中的使用 NUGETEXE cli](#)。

可用性

有关详细信息,请参阅[功能可用性](#)。

- 所有命令都在 Windows 上可用。
- 除为、和 `pack` `update` 指示的情况外, `restore` 所有命令都使用在 Mono 上运行的 nuget.exe。
- `locals` `restore` `delete` 在 Mac 和 Linux 上 `push`, 通过 dotnet CLI 还可以使用、、、和命令。`pack`

命令和适用性

可用命令和对包创建、包使用和/或将包发布到主机的适用性:

命令	功能	NUGET 版本	说明
pack	建立	2.7+	从 <code>.nuspec</code> 或项目文件创建 NuGet 包。在 Mono 上运行时, 不支持从项目文件创建包。
push	发布	全部	将包发布到包源。
config	全部	全部	获取或设置 NuGet 配置值。
help or ?	全部	全部	显示命令的帮助信息或帮助。
locals	使用	3.3+	列出全局包、 <code>http</code> 缓存和临时文件夹的位置, 并清除这些文件夹的内容。
restore	使用	2.7+	还原使用中的包管理格式所引用的所有包。在 Mono 上运行时, 不支持使用 <code>PackageReference</code> 格式还原包。
setapikey	发布, 消耗	全部	当包源需要访问密钥时, 保存给定包源的 API 密钥。
spec	建立	全部	如果从 <code>.nuspec</code> Visual Studio 项目生成文件, 则使用标记生成一个文件。

命令	功能	NUGET 版本	说明
add	发布	3.3+	使用分层布局将包添加到非 HTTP 包源。对于 HTTP 源, 请使用 <code>push</code> 。
delete	发布	全部	从包源中删除或取消列出包。
init	建立	3.3+	使用分层布局, 将文件夹中的包添加到包源。
install	使用	全部	将包安装到当前项目中, 但不修改项目或引用文件。

命令	功能	NUGET 命令	描述
list	消耗, 可能正在发布	全部	显示来自给定源的包。
mirror	发布	3.2 + 中弃用	将包及其依赖项从源存储库镜像到目标存储库。
sources	消耗, 发布	全部	管理配置文件中的包源。
update	使用	全部	将项目的包更新为最新的可用版本。在 Mono 上运行时不支持。

不同的命令使用各种[环境变量](#)。

按适用角色的 NuGet CLI 命令:

角色	命令
使用	config、help、install、list、locals、restore、setapikey、sources、update
建立	config、help、init、pack、spec
发布	add、config、delete、help、list、push、setapikey、sources

例如, 仅关注使用包的开发人员只需了解 NuGet 命令的子集。

NOTE

命令选项名称不区分大小写。不推荐使用的选项不包括在此引用中, 如 `NoPrompt` (`NonInteractive` 替换为 `Verbosity`) 和 `Verbose` (替换为)。

add 命令 (NuGet CLI)

2019/7/19 • [Edit Online](#)

适用于: 包发布 • 支持的版本: 3.3+

将指定的包添加到分层布局中的非 HTTP 包源 (文件夹或 UNC 路径), 其中为包 ID 和版本号创建了文件夹。例如:

```
\myserver\packages
  <packageID>
    <version>
      <packageID>.<version>.nupkg
      <packageID>.<version>.nupkg.sha512
      <packageID>.nuspec
```

针对包源还原或更新时, 分层布局提供明显更好的性能。

若要将包中的所有文件展开到目标包源, 请使用 `-Expand` 开关。这通常会导致在目标中出现其他子文件夹, 例如 `tools` 和 `lib`。

用法

```
nuget add <packagePath> -Source <sourcePath> [options]
```

其中 `<packagePath>`, 是要添加的包的路径名, 并 `<sourcePath>` 指定要将包添加到的基于文件夹的包源。HTTP 源不受支持。

选项

II	II
ConfigFile	要应用的 NuGet 配置文件。如果未指定, <code>%AppData%\NuGet\NuGet.Config</code> 则使用 (Windows <code>~/.nuget/NuGet/NuGet.Config</code>) 或 (Mac/Linux)。
Expand	将包中的所有文件添加到包源。
ForceEnglishOutput	(3.5+) 使用固定的、基于英语的区域性强制执行 nuget.exe。
Help	显示命令的帮助信息。
NonInteractive	取消显示提示用户输入或确认。
Verbosity	指定在输出中显示的详细信息量: "正常"、"静默"、"详细"。

另请参阅 [环境变量](#)

示例

```
nuget add foo.nupkg -Source c:\bar\  
nuget add foo.nupkg -Source \\bar\packages\
```

config 命令 (NuGet CLI)

2019/7/24 • [Edit Online](#)

适用于: 所有 • 受支持的版本: 全部

获取或设置 NuGet 配置值。有关其他用法, 请参阅[常见 NuGet 配置](#)。有关允许的密钥名称的详细信息, 请参阅[NuGet 配置文件参考](#)。

用法

```
nuget config -Set <name>=[<value>] [<name>=<value> ...] [options]  
nuget config -AsPath <name> [options]
```

其中 `<name>`, `<value>` 并指定要在配置中设置的键值对。您可以根据需要指定任意数量对。若要删除值, 请指定名称和 `=` 符号但不指定值。

有关允许的密钥名称, 请参阅[NuGet 配置文件参考](#)。

在 NuGet 3.4 + 中 `<value>`, 可以使用[环境变量](#)。

选项

示例	说明
<code>AsPath</code>	将配置值作为路径返回, 使用时 <code>-Set</code> 将忽略此值。
<code>ConfigFile</code>	要修改的 NuGet 配置文件。如果未指定, 则使用默认文件- <code>%AppData%\NuGet\NuGet.Config</code> (Windows) 或 <code>~/.config/NuGet/NuGet.Config</code> (Mac/Linux) 或 <code>~/.nuget/NuGet/NuGet.Config</code> (因 OS 分发而异)。
<code>ForceEnglishOutput</code>	(3.5 +) 使用固定的、基于英语的区域性强制执行 nuget.exe。
<code>Help</code>	显示命令的帮助信息。
<code>NonInteractive</code>	取消显示提示用户输入或确认。
<code>Verbosity</code>	指定在输出中显示的详细信息量: "正常"、"静默"、"详细"。

另请参阅[环境变量](#)

示例

```
nuget config -Set repositoryPath=c:\packages -configfile c:\my.config  
nuget config -Set repositoryPath=  
nuget config -Set repositoryPath=%PACKAGE_REPO% -configfile %ProgramData%\NuGet\NuGetDefaults.Config  
nuget config -Set HTTP_PROXY=http://127.0.0.1 -set HTTP_PROXY.USER=domain\user
```

delete 命令 (NuGet CLI)

2019/7/19 • [Edit Online](#)

适用于: 包发布 • 支持的版本: 全部

从包源中删除或取消列出包。对于 nuget.org, delete 命令取消列出包。

用法

```
nuget delete <packageID> <packageVersion> [options]
```

where `<packageID>` 和 `<packageVersion>` 确定要删除或取消列出的确切包。具体的行为取决于源。例如, 对于本地文件夹, 将删除包;对于 nuget.org, 包未列出。

选项

参数	描述
ApiKey	目标存储库的 API 密钥。如果不存在, 则使用配置文件中指定的一个。
ConfigFile	要应用的 NuGet 配置文件。如果未指定, <code>%AppData%\NuGet\NuGet.Config</code> 则使用 (Windows <code>~/.nuget/NuGet/NuGet.Config</code>) 或 (Mac/Linux)。
ForceEnglishOutput	(3.5+) 使用固定的、基于英语的区域性强制执行 nuget.exe。
Help	显示命令的帮助信息。
NonInteractive	取消显示提示用户输入或确认。
Source	指定服务器 URL。Nuget.org 的 URL 是 <code>https://api.nuget.org/v3/index.json</code> 。对于专用源, 请替换主机名, 例如, <code>%hostname%/api/v3</code> 。
Verbosity	指定在输出中显示的详细信息量: "正常"、"静默"、"详细"。

另请参阅[环境变量](#)

示例

```
nuget delete MyPackage 1.0
```

```
nuget delete MyPackage 1.0 -Source http://package.contoso.com/source -apikey A1B2C3
```

help or ? 命令 (NuGet CLI)

2019/7/19 • [Edit Online](#)

适用于: 所有 • 受支持的版本: 全部

显示常规帮助信息和有关特定命令的帮助信息。

用法

```
nuget help [command] [options]  
nuget ? [command] [options]
```

其中, [命令] 标识要显示其帮助的特定命令。

WARNING

对于某些命令, 请注意首先指定帮助, `nuget help install` 因为在 nuget.org 中有一个名为 "help" 的包。如果你指定了命令 `nuget install help`, 则将不会获得有关安装命令的帮助, 但会改为安装名为 help 的程序包。

选项

参数	描述
全部	打印所有可用命令的详细帮助;如果给定了特定命令, 则忽略。
ConfigFile	要应用的 NuGet 配置文件。如果未指定, <code>%AppData%\NuGet\NuGet.Config</code> 则使用 (Windows <code>~/.nuget/NuGet/NuGet.Config</code>) 或 (Mac/Linux)。
ForceEnglishOutput	(3.5+) 使用固定的、基于英语的区域性强制执行 nuget.exe。
Help	显示 help 命令本身的帮助信息。
Markdown	当与 <code>-All</code> 一起使用时, 打印 markdown 格式的详细帮助。否则忽略。
NonInteractive	取消显示提示用户输入或确认。
Verbosity	指定在输出中显示的详细信息量: "正常"、"静默"、"详细"。

另请参阅[环境变量](#)

示例

```
nuget help
nuget help push
nuget ?
nuget push -?
nuget help -All -Markdown
```

init 命令 (NuGet CLI)

2019/7/19 • [Edit Online](#)

适用于: 创建•包的支持版本: 3.3+

使用 "添加" 命令所述的分层布局, 将平面文件夹中的所有包复制到目标文件夹。也就是说, 使用 `init` 等效于对文件夹中的 `add` 每个包使用命令。

对于 `add`, 目标必须是本地文件夹或 UNC 路径;不支持 HTTP 包存储库, 如 nuget.org 或专用服务器。

用法

```
nuget init <source> <destination> [options]
```

其中 `<source>`, 是包含包的文件夹 `<destination>`, 是要将包复制到的本地文件夹或 UNC 路径名。

选项

参数	描述
<code>ConfigFile</code>	要应用的 NuGet 配置文件。如果未指定, <code>%AppData%\NuGet\NuGet.Config</code> 则使用 (Windows <code>~/.nuget/NuGet/NuGet.Config</code>) 或 (Mac/Linux)。
<code>ForceEnglishOutput</code>	(3.5+) 使用固定的、基于英语的区域性强制执行 nuget.exe。
<code>Expand</code>	添加添加到包源的每个包中的所有文件; <code>-Expand</code> 与 <code>add</code> 命令相同。
<code>Help</code>	显示命令的帮助信息。
<code>NonInteractive</code>	取消显示提示用户输入或确认。
<code>Verbosity</code>	指定在输出中显示的详细信息量: "正常"、"静默"、"详细"。

另请参阅[环境变量](#)

示例

```
nuget init c:\foo c:\bar
nuget init \\foo\packages \\bar\packages -Expand
```

install 命令 (NuGet CLI)

2020/2/6 • [Edit Online](#)

适用于：• 支持的版本的包使用：全部

使用指定的包源，将包下载并安装到项目中，默认为当前文件夹。

TIP

若要直接在项目上下文之外下载包，请访问[nuget.org](#)上的包页面，并选择“■”链接。

如果未指定任何源，将使用全局配置文件中列出的 `%appdata%\NuGet\NuGet.Config` (Windows) 或 `~/.nuget/NuGet.Config` (Mac/Linux)。有关更多详细信息，请参阅[常见 NuGet 配置](#)。

如果未指定特定的包，`install` 将安装项目 `packages.config` 文件中列出的所有包，使其类似于 `restore`。

`install` 命令不会修改项目文件或 `packages.config`；通过这种方式，它与 `restore` 类似，只是将包添加到磁盘，但不会更改项目的依赖项。

若要添加依赖项，请在 Visual Studio 中通过包管理器 UI 或控制台添加包，或修改 `packages.config` 然后运行 `install` 或 `restore`。

用法

```
nuget install <packageID | configFilePath> [options]
```

其中 `<packageID>` 命名要安装的包（使用最新版本），或 `<configFilePath>` 标识列出要安装的包的 `packages.config` 文件。您可以使用 `-Version` 选项指示特定版本。

Options

参数	描述
<code>ConfigFile</code>	要应用的 NuGet 配置文件。如果未指定，则使用 <code>%AppData%\NuGet\NuGet.Config</code> (Windows) 或 <code>~/.nuget/NuGet.Config</code> (Mac/Linux)。
<code>DependencyVersion</code>	(4.4+) 要使用的依赖项包的版本，可以是下列项之一： <ul style="list-style-type: none">最低(默认值)：最低版本<i>HighestPatch</i>：最小主要、次要和最高修补程序的版本<i>HighestMinor</i>：最小主要版本号最高的版本，最高修补程序最高：最高版本忽略：将不使用任何依赖项包
<code>DisableParallelProcessing</code>	禁止并行安装多个包。
<code>ExcludeVersion</code>	将包安装到仅带有包名称而不是版本号的名为的文件夹中。

FallbackSource	(3.2+) 在主或默认源中找不到包时要用作回退的包的列表。
ForceEnglishOutput	(3.5+) 使用固定的、基于英语的区域性强制执行 nuget.exe。
框架	(4.4+) 用于选择依赖项的目标框架。如果未指定，默认值为 "Any"。
帮助	显示命令的帮助信息。
NoCache	阻止 NuGet 使用缓存的包。请参阅 管理全局包和缓存文件夹 。
NonInteractive	取消显示提示用户输入或确认。
OutputDirectory	指定在其中安装包的文件夹。如果未指定文件夹，则使用当前文件夹。
PackageSaveMode	指定包安装后要保存的文件的类型： <code>nuspec</code> 、 <code>nupkg</code> 或 <code>nuspec; nupkg</code> 之一。
PreRelease	允许安装预发行程序包。在 <code>packages.config</code> 还原包时，不需要此标志。
RequireConsent	验证在下载和安装包之前是否启用了还原包。有关详细信息，请参阅 包还原 。
SolutionDirectory	指定要为其还原包的解决方案的根文件夹。
源	指定要使用的包源（作为 Url）的列表。如果省略，则该命令使用配置文件中提供的源，请参阅 常见的 NuGet 配置 。
详细程度	指定在输出中显示的详细信息量：“正常”、“静默”、“详细”。
版本	指定要安装的程序包的版本。

另请参阅[环境变量](#)

示例

```
nuget install elmah
nuget install packages.config
nuget install ninject -OutputDirectory c:\proj
```

list 命令 (NuGet CLI)

2020/1/31 • [Edit Online](#)

适用于：包使用情况、发布 • 支持的版本：全部

显示来自给定源的包的列表。如果未指定任何源，将使用全局配置文件中定义的所有源

`%AppData%\NuGet\NuGet.Config` (Windows) 或 `~/.nuget/NuGet/NuGet.Config`。如果 `NuGet.Config` 未指定源，则 `list` 使用默认源 (nuget.org)。

用量

```
nuget list [search terms] [options]
```

其中，可选搜索词将筛选显示的列表。搜索词应用于包、标记和包说明的名称，就像在 nuget.org 上使用时一样。

选项

参数	描述
<code>AllVersions</code>	列出包的所有版本。默认情况下，仅显示最新的包版本。
<code>ConfigFile</code>	要应用的 NuGet 配置文件。如果未指定，则使用 <code>%AppData%\NuGet\NuGet.Config</code> (Windows) 或 <code>~/.nuget/NuGet/NuGet.Config</code> (Mac/Linux)。
<code>ForceEnglishOutput</code>	(3.5+) 使用固定的、基于英语的区域性强制执行 nuget.exe。
<code>Help</code>	显示命令的帮助信息。
<code>IncludeDelisted</code>	(3.2+) 显示未列出的包。
<code>NonInteractive</code>	取消显示提示用户输入或确认。
<code>PreRelease</code>	在列表中包括预发行程序包。
<code>Source</code>	指定要搜索的包源列表。
<code>Verbose</code>	指定在输出中显示的详细信息量：“正常”、“静默”、“详细”。

另请参阅[环境变量](#)

示例

列出配置的源中的所有包：

```
nuget list
```

列出与 Azure 相关的包，详细说明：

```
nuget list Azure -Verbosity detailed
```

列出来自配置的源的 Azure 相关包的所有版本：

```
nuget list Azure -AllVersions
```

列出指定源/源中与 JSON 相关的所有包的所有版本：

```
nuget list JSON -AllVersions -Source "https://nuget.org/api/v2"
```

列出来自多个源/源的 JSON 相关包：

```
nuget list JSON -Source "https://nuget.org/api/v2" -Source "https://other-feed-url-goes-here"
```

locals 命令 (NuGet CLI)

2019/7/19 • [Edit Online](#)

适用于: 包 • 使用受支持的版本: 3.3+

清除或列出本地 NuGet 资源, 例如 `http` 缓存、全局包文件夹和临时文件夹。`locals` 命令也可用于显示这些位置的列表。有关详细信息, 请参阅[管理全局包和缓存文件夹](#)。

用法

```
nuget locals <folder> [options]
```

其中 `<folder>` `all`, 是 `plugins-cache`、`global-packages`、`temp`、(3.5 及更早版本)、(3.4+) 和 (4.8+) 之一。
`packages-cache` | `http-cache`

选项

参数	说明
<code>-clear</code>	清除指定的文件夹。
<code>-ConfigFile</code>	要应用的 NuGet 配置文件。如果未指定, <code>%AppData%\NuGet\NuGet.Config</code> 则使用 (Windows <code>~/.nuget/NuGet/NuGet.Config</code>) 或 (Mac/Linux)。
<code>-ForceEnglishOutput</code>	(3.5+) 使用固定的、基于英语的区域性强制执行 nuget.exe。
<code>-Help</code>	显示命令的帮助信息。
<code>-List</code>	列出指定文件夹的位置或所有文件夹在与 <code>all</code> 一起使用时的位置。
<code>-NonInteractive</code>	取消显示提示用户输入或确认。
<code>-Verbosity</code>	指定在输出中显示的详细信息量: "正常"、"静默"、"详细"。

另请参阅[环境变量](#)

示例

```
nuget locals all -list
nuget locals http-cache -clear
```

有关其他示例, 请参阅[管理全局包和缓存文件夹](#)。

mirror 命令 (NuGet CLI)

2019/8/13 • • [Edit Online](#)

适用于: 包发布• 支持的版本: 3.2 + 中弃用

将包及其依赖项从指定的源存储库镜像到目标存储库。

NOTE

ServerExtensions 和 NuGet-Signed 以前在 NuGet 2.x 中支持此命令 (通过将 NuGet-Signed 重命名为 nuget.exe) 已不再可供下载。若要使用与此类似的命令, 请尝试[NuGetMirror](#)。

用法

```
nuget mirror <packageID | configFilePath> <listUrlTarget> <publishUrlTarget> [options]
```

其中 `<packageID>`, 是要镜像的包, `<configFilePath>` 或标识 `packages.config` 列出要镜像的包的文件。

指定源存储库, 并 `<publishUrlTarget>` 指定目标存储库。 `<listUrlTarget>`

如果目标存储库在运行 `https://machine/repo` [NuGet 服务器](#) 的上, 则列表和推送 url 将分别为

`https://machine/repo/nuget` 和 `https://machine/repo/api/v2/package`。

选项

参数	说明
ApiKey	目标存储库的 API 密钥。如果不存在, 则使用配置文件中指定的一个 (<code>%AppData%\NuGet\NuGet.Config</code> (Windows) 或 <code>~/.nuget/NuGet/NuGet.Config</code> (Mac/Linux))。
Help	显示命令的帮助信息。
NoCache	阻止 NuGet 使用缓存的包。请参阅 管理全局包和缓存文件夹 。
Noop	记录将执行的操作, 但不执行操作;假定推送操作成功。
Early	在镜像操作中包括预发行程序包。
Source	要镜像的包源的列表。如果未指定任何源, 将使用配置文件中定义的源 (请参阅上面的 ApiKey), 如果未指定, 则默认为 <code>nuget.org</code> 。
Timeout	指定推送到服务器的超时值 (以秒为单位)。默认值为300秒 (5分钟)。
Version	要安装的包的版本。如果未指定, 则将镜像最新版本。

另请参阅[环境变量](#)

示例

```
nuget mirror packages.config https://MyRepo/nuget https://MyRepo/api/v2/package -source  
https://nuget.org/api/v2 -apikey myApiKey -nocache  
  
nuget mirror Microsoft.AspNet.Mvc https://MyRepo/nuget https://MyRepo/api/v2/package -version 4.0.20505.0  
  
nuget mirror Microsoft.Net.Http https://MyRepo/nuget https://MyRepo/api/v2/package -prerelease
```

pack 命令 (NuGet CLI)

2020/3/3 • [Edit Online](#)

适用于：包创建 • 支持的版本：2.7 +

基于指定的 `nuspec` 或项目文件创建 NuGet 包。`dotnet pack` 命令 (请参阅[Dotnet 命令](#)) 和 `msbuild -t:pack` (请参阅[MSBuild 目标](#)) 可以用作备用项。

IMPORTANT

为基于 `PackageReference` 的项目使用 `dotnet pack` 或 `msbuild -t:pack`。在 Mono 下，不支持从项目文件创建包。还需要将 `.nuspec` 文件中的非本地路径调整为 Unix 样式路径，因为 `nuget.exe` 不会转换 Windows 路径名本身。

使用情况

```
nuget pack <nuspecPath | projectPath> [options] [-Properties ...]
```

其中 `<nuspecPath>` 和 `<projectPath>` 分别指定 `.nuspec` 或项目文件。

选项

II	II
BasePath	设置在 <code>nuspec</code> 文件中定义的文件的基路径。
构建	指定应在生成包之前生成项目。
排除	指定创建包时要排除的一个或多个通配符模式。若要指定多个模式，请重复 <code>-Exclude</code> 标志。请参阅以下示例。
ExcludeEmptyDirectories	在生成包时阻止包含空目录。
ForceEnglishOutput	(3.5 +) 使用固定的、基于英语的区域性强制执行 <code>nuget.exe</code> 。
ConfigFile	指定 <code>pack</code> 命令的配置文件。
帮助	显示命令的帮助信息。
IncludeReferencedProjects	指示生成的包应包含作为依赖项或包的一部分的引用项目。如果引用的项目具有与项目同名的相应 <code>.nuspec</code> 文件，则会将引用项目作为依赖项添加。否则，被引用项目作为包的一部分添加。
MinClientVersion	设置所创建的包的 <code>minClientVersion</code> 属性。此值将覆盖 <code>.nuspec</code> 文件中现有 <code>minClientVersion</code> 属性的值 (如果有)。
MSBuildPath	(4.0 +) 指定要与命令一起使用的 MSBuild 的路径，其优先级高于 <code>-MSBuildVersion</code> 。

MSBuildVersion	(3.2+) 指定要与此命令一起使用的 MSBuild 版本。支持的值为 4、12、14、15.1、15.3、15.4、15.5、15.6、15.7、15.8、15.9。默认情况下，将选取路径中的 MSBuild，否则默认为已安装的最高版本的 MSBuild。
NoDefaultExcludes	禁止默认排除 NuGet 包文件以及以点(例如 <code>.svn</code> 和 <code>.gitignore</code>)开头的文件和文件夹。
NoPackageAnalysis	指定 pack 不应在生成包后运行包分析。
OutputDirectory	指定在其中存储创建的包的文件夹。如果未指定文件夹，则使用当前文件夹。
属性	应在其后出现在命令行的最后。指定重写项目文件中的值的属性的列表;请参阅属性名称的 常用 MSBuild 项目属性 。此处的 Properties 参数是由分号分隔的标记 = 值对列表，其中每个 <code>\$token\$</code> 出现在 <code>.nuspec</code> 文件中，都将替换为给定的值。值可以是用引号引起起来的字符串。请注意，对于 "配置" 属性，默认值为 "调试"。若要更改为发布配置，请使用 <code>-Properties Configuration=Release</code> 。 ■ 情况下，属性应该与在相应 <code>nuget build</code> 中使用的属性相同，以避免潜在的异常行为。
Suffix	(3.4.4+) 将后缀追加到内部生成的版本号，通常用于追加生成或其他预发行标识符。例如，使用 <code>-suffix nightly</code> 会创建一个包，其中包含 <code>1.2.3-nightly</code> 等版本号。后缀必须以字母开头，以避免与不同版本的 NuGet 和 NuGet 包管理器的警告、错误和可能的不兼容性。
"符号"	指定包包含源和符号。与 <code>.nuspec</code> 文件一起使用时，将创建一个常规 NuGet 包文件和相应的符号包。默认情况下，它会创建 旧的符号包 。符号包的新推荐格式为 <code>.snupkg</code> 。请参阅 创建符号包 (.snupkg) 。
工具	指定应将项目的输出文件放在 <code>tool</code> 文件夹中。
详细程度	指定在输出中显示的详细信息量："正常"、"静默"、"详细"。
版本	覆盖 <code>.nuspec</code> 文件中的版本号。

另请参阅[环境变量](#)

排除开发依赖项

某些 NuGet 包可用作开发依赖项，它们可帮助你创作自己的库，但不一定是实际的包依赖项。

`pack` 命令将忽略 `packages.config` 中将 `developmentDependency` 特性设置为 `true` 的 `package` 项。这些项不会作为依赖项包含在所创建的包中。

例如，请考虑源项目中的以下 `packages.config` 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="jQuery" version="1.5.2" />
  <package id="netfx-Guard" version="1.3.3.2" developmentDependency="true" />
  <package id="microsoft-web Helpers" version="1.15" />
</packages>
```

对于此项目，`nuget pack` 创建的包将依赖于 `jQuery` 和 `microsoft-web-helpers` 但不 `netfx-Guard`。

抑制包警告

尽管建议你在包操作期间解决所有 NuGet 警告，但在某些情况下禁止它们是保证的。

可以通过以下方式实现此目的：

```
nuget.exe 包 nuspec-Properties NoWarn = NU5104
```

示例

```
nuget pack

nuget pack foo.nuspec

nuget pack foo.csproj

nuget pack foo.csproj -Properties Configuration=Release

nuget pack foo.csproj -Build -Symbols -Properties owners=janedoe,xiaop;version="1.0.5"

# Create a package from project foo.csproj, using MSBuild version 12 to build the project
nuget pack foo.csproj -Build -Symbols -MSBuildVersion 12 -Properties owners=janedoe,xiaop;version="1.0.5"

# Create a package from project foo.nuspec and the corresponding symbol package using the new recommended
format .snupkg
nuget pack foo.nuspec -Symbols -SymbolPackageFormat snupkg

nuget pack foo.nuspec -Version 2.1.0

nuget pack foo.nuspec -Version 1.0.0 -MinClientVersion 2.5

nuget pack Package.nuspec -exclude "*.exe" -exclude "*.bat"
```

push 命令 (NuGet CLI)

2019/7/19 • [Edit Online](#)

适用于: 包发布 • 支持的版本: 全部; 4.1.0 + 必需的 nuget.org

IMPORTANT

若要将包推送到 nuget.org, 必须使用 4.1.0 +, 后者实现所需的nuget 协议。

将包推送到包源并发布包。

NuGet 的默认配置是通过 `%AppData%\NuGet\NuGet.Config` 加载 (Windows) 或 `~/.nuget/NuGet/NuGet.Config` (Mac/Linux `Nuget.Config`) `.nuget\Nuget.Config`, 然后从驱动器的根目录启动并在当前目录中结束 (请参阅[常用 NuGet 配置](#))

用法

```
nuget push <packagePath> [options]
```

其中 `<packagePath>` 标识要推送到服务器的包。

选项

II	II
ApiKey	目标存储库的 API 密钥。如果不存在, 则使用配置文件中指定的一个。
ConfigFile	要应用的 NuGet 配置文件。如果未指定, <code>%AppData%\NuGet\NuGet.Config</code> 则使用 (Windows <code>~/.nuget/NuGet/NuGet.Config</code>) 或 (Mac/Linux)。
DisableBuffering	在推送到 HTTP (s) 服务器时禁用缓冲, 以减少内存使用量。注意: 使用此选项时, 集成的 Windows 身份验证可能不起作用。
ForceEnglishOutput	(3.5 +) 使用固定的、基于英语的区域性强制执行 nuget.exe。
Help	显示命令的帮助信息。
NonInteractive	取消显示提示用户输入或确认。
NoSymbols	(3.5 +) 如果符号包存在, 则不会将其推送到符号服务器。
Source	指定服务器 URL。NuGet 标识 UNC 或本地文件夹, 只是复制该文件, 而不是使用 HTTP 推送它。此外, 从 NuGet 3.4.2 开始, 这是必需的参数, 除非 <code>NuGet.Config</code> 该文件指定了 <code>DefaultPushSource</code> 值 (请参阅 配置 NuGet 行为)。

SkipDuplicate	(5.1+) 如果包和版本已存在，则跳过它，并继续执行推送中的下一个包（如果有）。
SymbolSource	(3.5+) 指定符号服务器 URL；推送到 nuget.org 时使用 nuget.smbsrc.net
SymbolApiKey	(3.5+) 为中 <code>-SymbolSource</code> 指定的 URL 指定 API 密钥。
Timeout	指定推送到服务器的超时值（以秒为单位）。默认值为300秒（5分钟）。
Verbosity	指定在输出中显示的详细信息量：“正常”、“静默”、“详细”。

另请参阅[环境变量](#)

示例

```

nuget push foo.nupkg

nuget push foo.symbols.nupkg

nuget push foo.nupkg -Timeout 360

nuget push *.nupkg

nuget.exe push -source \\mycompany\repo\ mypackage.1.0.0.nupkg

nuget push foo.nupkg 4003d786-cc37-4004-bfdf-c4f3e8ef9b3a -Source https://api.nuget.org/v3/index.json

nuget push foo.nupkg 4003d786-cc37-4004-bfdf-c4f3e8ef9b3a

nuget push foo.nupkg 4003d786-cc37-4004-bfdf-c4f3e8ef9b3a -src https://customsource/

:: In the example below -SkipDuplicate will skip pushing the package if package "Foo" version "5.0.2" already
exists on NuGet.org
nuget push Foo.5.0.2.nupkg 4003d786-cc37-4004-bfdf-c4f3e8ef9b3a -src https://api.nuget.org/v3/index.json -
SkipDuplicate

```

restore 命令 (NuGet CLI)

2020/3/3 • [Edit Online](#)

适用于：包消耗 • 支持的版本：2.7 +

下载并安装 `packages` 文件夹中缺少的任何程序包。与 NuGet 4.0+ 和 `PackageReference` 格式一起使用时，会在 `obj` 文件夹中生成 `<project>.nuget.props` 文件（如果需要）。（可以从源代码管理中省略该文件。）

在 Mono 上带有 CLI 的 Mac OS X 和 Linux 上，`PackageReference` 不支持还原包。

使用情况

```
nuget restore <projectPath> [options]
```

其中 `<projectPath>` 指定解决方案或 `packages.config` 文件的位置。有关行为的详细信息，请参阅下面的备注。

选项

参数	描述
<code>ConfigFile</code>	要应用的 NuGet 配置文件。如果未指定，则使用 <code>%AppData%\NuGet\NuGet.Config</code> （Windows）或 <code>~/.nuget/NuGet/NuGet.Config</code> （Mac/Linux）。
<code>DirectDownload</code>	(4.0+) 直接下载包，而不填充包含任何二进制文件或元数据的缓存。
<code>DisableParallelProcessing</code>	禁用并行还原多个包。
<code>FallbackSource</code>	(3.2+) 在主或默认源中找不到包时要用作回退的包的列表。使用分号分隔列表项。
<code>Force</code>	在基于 <code>PackageReference</code> 的项目中，将强制解析所有依赖项，即使上次还原已成功。指定此标志与删除 <code>project.assets.json</code> 文件类似。这不会绕过 http 缓存。
<code>ForceEnglishOutput</code>	(3.5+) 使用固定的、基于英语的区域性强制执行 <code>nuget.exe</code> 。
<code>ForceEvaluate</code>	强制还原以重新计算所有依赖项，即使已经存在一个锁定文件。
<code>Help</code>	显示命令的帮助信息。
<code>LockFilePath</code>	写入项目锁定文件的输出位置。默认情况下，这是 " <code>PROJECT_ROOT\packages.lock.json</code> "。
<code>LockedMode</code>	不允许更新项目锁定文件。
<code>MSBuildPath</code>	(4.0+) 指定要与命令一起使用的 MSBuild 的路径，其优先级高于 <code>-MSBuildVersion</code> 。

MSBuildVersion	(3.2+) 指定要与此命令一起使用的 MSBuild 版本。支持的值为 4、12、14、15.1、15.3、15.4、15.5、15.6、15.7、15.8、15.9。默认情况下，将选取路径中的 MSBuild，否则默认为已安装的最高版本的 MSBuild。
NoCache	阻止 NuGet 使用缓存的包。请参阅 管理全局包和缓存文件夹 。
NonInteractive	取消显示提示用户输入或确认。
OutputDirectory	指定在其中安装包的文件夹。如果未指定文件夹，则使用当前文件夹。当使用 <code>packages.config</code> 文件进行还原时必需，除非使用 <code>PackagesDirectory</code> 或 <code>SolutionDirectory</code> 。
PackageSaveMode	指定包安装后要保存的文件的类型： <code>nuspec</code> 、 <code>nupkg</code> 或 <code>nuspec; nupkg</code> 之一。
PackagesDirectory	与 <code>OutputDirectory</code> 相同。当使用 <code>packages.config</code> 文件进行还原时必需，除非使用 <code>OutputDirectory</code> 或 <code>SolutionDirectory</code> 。
Project2ProjectTimeOut	解析项目到项目引用的超时时间(秒)。
Recursive	(4.0+) 还原 UWP 和 .NET Core 项目的所有引用项目。不适用于使用 <code>packages.config</code> 的项目。
RequireConsent	验证在下载和安装包之前是否启用了还原包。有关详细信息，请参阅 包还原 。
SolutionDirectory	指定解决方案文件夹。还原解决方案的包时无效。当使用 <code>packages.config</code> 文件进行还原时必需，除非使用 <code>PackagesDirectory</code> 或 <code>OutputDirectory</code> 。
源	指定要用于还原的包源(作为 Url)的列表。如果省略，则该命令将使用配置文件中提供的源，请参阅 配置 NuGet 行为 。使用分号分隔列表项。
UseLockFile	允许生成项目锁定文件并将其与 restore 一起使用。
详细程度	指定在输出中显示的详细信息量：“正常”、“静默”、“详细”。

另请参阅[环境变量](#)

备注

Restore 命令执行以下步骤：

- 确定 restore 命令的操作模式。

PROJECTPATH

解决方案(文件夹)

II

NuGet 查找 `.sln` 文件，如果找到，则使用该文件；否则，将产生错误。`(SolutionDir)\.nuget` 用作起始文件夹。

PROJECTPATH	
.sln 文件	还原解决方案标识的包;如果使用 -SolutionDirectory , 则会出现错误。\$(SolutionDir)\.nuget 用作起始文件夹。
packages.config 或项目文件	还原文件中列出的包, 解析和安装依赖项。
其他文件类型	文件被假定为上述 .sln 文件;如果它不是解决方案, NuGet 将会出现错误。
(未指定 projectPath)	<ul style="list-style-type: none"> NuGet 在当前文件夹中查找解决方案文件。如果找到一个文件, 则使用该文件还原包;如果找到多个解决方案, NuGet 将提供错误。 如果没有解决方案文件, NuGet 将查找 packages.config , 并使用该文件还原包。 如果未找到任何解决方案或 packages.config 文件, NuGet 将提供错误。

2. 使用以下优先级顺序确定包文件夹(如果找不到这些文件夹, 则 NuGet 会出现错误):

- PackagesDirectory 指定的文件夹。
- NuGet.Config 中的 repositoryPath 值
- 用 -SolutionDirectory 指定的文件夹
- \$(SolutionDir)\packages

3. 还原解决方案的包时, NuGet 会执行以下操作:

- 加载解决方案文件。
- 将 \$(SolutionDir)\.nuget\packages.config 中列出的解决方案级别包还原到 packages 文件夹。
- 将 \$(ProjectDir)\packages.config 中列出的包还原到 packages 文件夹。对于指定的每个包, 请并行还原包, 除非指定 -DisableParallelProcessing 。

示例

```
# Restore packages for a solution file
nuget restore a.sln

# Restore packages for a solution file, using MSBuild version 14.0 to load the solution and its project(s)
nuget restore a.sln -MSBuildVersion 14

# Restore packages for a project's packages.config file, with the packages folder at the parent
nuget restore proj1\packages.config -PackagesDirectory ..\packages

# Restore packages for the solution in the current folder, specifying package sources
nuget restore -source "https://api.nuget.org/v3/index.json;https://www.myget.org/F/nuget"
```

setapikey 命令 (NuGet CLI)

2020/3/3 • [Edit Online](#)

适用于：包使用情况、发布 • 支持的版本：全部

将给定服务器 URL 的 API 密钥保存到 `NuGet.Config`，以便不需要为后续命令输入该 URL。

使用情况

```
nuget setapikey <key> -Source <url> [options]
```

其中 `<source>` 标识服务器，`<key>` 是要保存的密钥。如果省略 `<source>`，则采用 nuget.org。

NOTE

API 密钥不用于通过专用源进行身份验证。请参阅 [nuget sources 命令](#)，管理凭据以便通过源进行身份验证。可以从单个 NuGet 服务器获取 API 密钥。若要创建和管理 nuget.org 的 APIKeys，请参阅[发布 api 密钥](#)

选项

参数	描述
<code>ConfigFile</code>	要应用的 NuGet 配置文件。如果未指定，则使用 <code>%AppData%\NuGet\NuGet.Config</code> (Windows) 或 <code>~/.nuget/NuGet/NuGet.Config</code> (Mac/Linux)。
<code>ForceEnglishOutput</code>	(3.5+) 使用固定的、基于英语的区域性强制执行 nuget.exe。
<code>Help</code>	显示命令的帮助信息。
<code>NonInteractive</code>	取消显示提示用户输入或确认。
<code>Verbosity</code>	指定在输出中显示的详细信息量：“正常”、“静默”、“详细”。

另请参阅[环境变量](#)

示例

```
nuget setapikey 4003d786-cc37-4004-bfdf-c4f3e8ef9b3a  
nuget setapikey 4003d786-cc37-4004-bfdf-c4f3e8ef9b3a -source https://example.com/nugetfeed
```

sign 命令 (NuGet CLI)

2020/3/3 • [Edit Online](#)

适用于：包创建 • 支持的版本：4.6 +

使用证书对匹配第一个参数的所有包进行签名。可以通过提供使用者名称或指纹，从文件或证书存储中安装的证书获取带有私钥的证书。

NOTE

.NET Core、Mono 或非 Windows 平台上尚不支持包签名。

使用情况

```
nuget sign <package(s)> [options]
```

其中 `<package(s)>` 是一个或多个 `.nupkg` 文件。

选项

参数	说明
<code>CertificateFingerprint</code>	指定用于搜索证书的本地证书存储区的证书的 SHA-1 指纹。
<code>CertificatePassword</code>	如果需要，指定证书密码。如果证书受密码保护，但未提供密码，则该命令将在运行时提示输入密码，除非传递了-非交互式选项。
<code>CertificatePath</code>	指定用于对包进行签名的证书的文件路径。
<code>CertificateStoreLocation</code>	指定用于搜索证书的 x.509 证书存储区的名称。默认为 "CurrentUser"，当前用户使用的 x.509 证书存储。当通过-CertificateSubjectName 或-CertificateFingerprint 选项指定证书时，应使用此选项。
<code>CertificateStoreName</code>	指定用于搜索证书的 x.509 证书存储区的名称。默认值为 "My"，适用于个人证书的 x.509 证书存储。当通过-CertificateSubjectName 或-CertificateFingerprint 选项指定证书时，应使用此选项。
<code>CertificateSubjectName</code>	指定用于在本地证书存储区中搜索证书的证书的使用者名称。搜索是使用提供的值进行区分大小写的字符串比较，它将查找使用者名称包含该字符串的所有证书，而与其他使用者值无关。可以通过-CertificateStoreName 和-CertificateStoreLocation 选项指定证书存储区。
<code>ConfigFile</code>	要应用的 NuGet 配置文件。如果未指定，则使用 <code>%AppData%\NuGet\NuGet.Config</code> (Windows) 或 <code>~/.nuget/NuGet/NuGet.Config</code> (Mac/Linux)。

ForceEnglishOutput	使用固定的、基于英语的区域性强制执行 nuget.exe。
HashAlgorithm	用于对包进行签名的哈希算法。默认值为 SHA256。可能的值为 SHA256、SHA384 和 SHA512。
帮助	显示命令的帮助信息。
NonInteractive	取消显示提示用户输入或确认。
OutputDirectory	指定应将已签名的包保存到的目录。默认情况下，已签名的包将覆盖原始包。
Overwrite	切换以指示是否应覆盖当前签名。默认情况下，如果包已有签名，则该命令将失败。
Timestamper	RFC 3161 时间戳服务器的 URL。
TimestampHashAlgorithm	RFC 3161 时间戳服务器使用的哈希算法。默认值为 SHA256。
详细程度	指定在输出中显示的详细信息量："正常"、"静默"、"详细"。

示例

```
nuget sign MyPackage.nupkg -Timestamper http://timestamp.test
nuget sign .\..\MyPackage.nupkg -Timestamper http://timestamp.test -OutputDirectory .\..\Signed
```

sources 命令 (NuGet CLI)

2019/7/19 • [Edit Online](#)

适用于: 包消耗, 发布 • 支持的版本: 全部

管理位于用户范围配置文件或指定配置文件中的源的列表。用户范围配置文件位于 `%appdata%\NuGet\NuGet.Config` (Windows) 和 `~/.nuget/NuGet/NuGet.Config` (Mac/Linux)。

请注意, nuget.org 的源 URL 是 <https://api.nuget.org/v3/index.json>。

用法

```
nuget sources <operation> -Name <name> -Source <source>
```

其中 `<operation>` 是 *List*、*Add*、*Remove*、*Enable*、*Disable* 或 *Update*，`<name>` 中的一个，是源的名称，`<source>` 是源的 URL。一次只能在一个源上操作。

选项

参数	描述
ConfigFile	要应用的 NuGet 配置文件。如果未指定, <code>%AppData%\NuGet\NuGet.Config</code> 则使用 (Windows) <code>~/.nuget/NuGet/NuGet.Config</code> 或 (Mac/Linux)。
ForceEnglishOutput	(3.5+) 使用固定的、基于英语的区域性强制执行 nuget.exe。
格式	适用于 <code>Detailed</code> <code>Short</code> 操作, 可以为 (默认值) 或 <code>list</code>
Help	显示命令的帮助信息。
NonInteractive	取消显示提示用户输入或确认。
Password	指定用于对源进行身份验证的密码。
StorePasswordInPlainText	指示以未加密的文本而不是存储加密的窗体的默认行为来存储密码。
UserName	指定用于对源进行身份验证的用户名。
Verbosity	指定在输出中显示的详细信息量: "正常"、"静默"、"详细"。

NOTE

请确保在与 nuget.exe 一起用于访问包源的用户上下文中添加源密码。密码将以加密形式存储在配置文件中, 并且只能在对其进行加密的用户上下文中解密。例如, 在使用生成服务器还原 NuGet 包时, 必须使用运行生成服务器任务的相同 Windows 用户对密码进行加密。

另请参阅 [环境变量](#)

示例

```
nuget sources Add -Name "MyServer" -Source \\myserver\packages  
nuget sources Disable -Name "MyServer"  
nuget sources Enable -Name "nuget.org"  
  
nuget sources add -name foo.bar -source C:\NuGet\local -username foo -password bar -StorePasswordInClearText  
-configfile %AppData%\NuGet\my.config
```

spec 命令 (NuGet CLI)

2019/7/19 • [Edit Online](#)

适用于: 创建•包的支持版本: 全部

为新包生成文件。`.nuspec` 如果在与项目文件相同的文件夹中运行 (`.csproj`、`.vbproj`、`.fsproj`), `spec` 则将创建 `.nuspec` 一个标记化文件。有关其他信息, 请参阅[创建包](#)。

用法

```
nuget spec [<packageID>] [options]
```

其中 `<packageID>` 是要保存到 `.nuspec` 文件中的可选包标识符。

选项

参数	描述
<code>AssemblyPath</code>	指定要用于元数据的程序集的路径。
<code>Team</code>	覆盖任何现有 <code>.nuspec</code> 文件。
<code>ForceEnglishOutput</code>	(3.5+) 使用固定的、基于英语的区域性强制执行 nuget.exe。
<code>Help</code>	显示命令的帮助信息。
<code>NonInteractive</code>	取消显示提示用户输入或确认。
<code>Verbosity</code>	指定在输出中显示的详细信息量: "正常"、"静默"、"详细"。

另请参阅[环境变量](#)

示例

```
nuget spec  
nuget spec MyPackage  
nuget spec -AssemblyPath MyAssembly.dll
```

update 命令 (NuGet CLI)

2019/7/19 • [Edit Online](#)

适用于: 包•使用受支持的版本: 全部

将项目中的所有包(使用 `packages.config`)更新为其最新可用版本。建议在运行 `update` 之前运行“[还原](#)”。(若要更新单个包, 请 `nuget install` 使用而不指定版本号, 在这种情况下, NuGet 安装最新版本。)

注意: `update` 不适用于在 Mono (Mac OSX 或 Linux) 下运行的 CLI 或使用 PackageReference 格式。

`update` 命令还会更新项目文件中的程序集引用, 前提是这些引用已存在。如果更新的包具有添加的程序集, 则不会添加新引用。新的包依赖项也没有添加其程序集引用。若要将这些操作包括为更新的一部分, 请使用包管理器 UI 或程序包管理器控制台在 Visual Studio 中更新包。

此命令还可用于使用 `-self` 标志更新 nuget.exe 本身。

用法

```
nuget update <configPath> [options]
```

其中 `<configPath>` 标识 `packages.config` 列出项目依赖项的或解决方案文件。

选项

参数	描述
<code>ConfigFile</code>	要应用的 NuGet 配置文件。如果未指定, <code>%AppData%\NuGet\NuGet.Config</code> 则使用 (Windows <code>~/.nuget/NuGet/NuGet.Config</code>) 或 (Mac/Linux)。
<code>FileConflictAction</code>	指定在要求覆盖或忽略项目引用的现有文件时要执行的操作。 值为 <code>overwrite</code> , <code>ignore</code> , <code>none</code> 。
<code>ForceEnglishOutput</code>	(3.5+) 使用固定的、基于英语的区域性强制执行 nuget.exe。
<code>Help</code>	显示命令的帮助信息。
<code>Id</code>	指定要更新的包 Id 列表。
<code>MSBuildPath</code>	(4.0+) 指定要与命令一起使用的 MSBuild 的路径, 优先于 <code>-MSBuildVersion</code> 。
<code>MSBuildVersion</code>	(3.2+) 指定要与此命令一起使用的 MSBuild 版本。支持的值 为 4、12、14、15.1、15.3、15.4、15.5、15.6、15.7、15.8、15.9。 默认情况下, 将选取路径中的 MSBuild, 否则默认为已安装的最 高版本的 MSBuild。
<code>NonInteractive</code>	取消显示提示用户输入或确认。

早期	允许更新到预发布版本。更新已安装的预发布包时, 不需要此标志。
RepositoryPath	指定安装包的本地文件夹。
防	指定仅安装具有与安装包相同的主要版本和次要版本中可用的最高版本的更新。
解压	将 nuget.exe 更新到最新版本;忽略所有其他参数。
Source	指定要用于更新的包源 (作为 Url) 的列表。如果省略, 则该命令使用配置文件中提供的源, 请参阅 常见的 NuGet 配置 。
Verbosity	指定在输出中显示的详细信息量: "正常"、"静默"、"详细"。
Version	与一个包 ID 一起使用时, 指定要更新的包的版本。

另请参阅[环境变量](#)

示例

```
nuget update

# update packages installed in solution.sln, using MSBuild version 14.0 to load the solution and its
# project(s).
nuget update solution.sln -MSBuildVersion 14

nuget update -safe

nuget update -self
```

verify 命令 (NuGet CLI)

2019/7/19 • [Edit Online](#)

适用于: 包•使用受支持的版本: 4.6 +

验证包。

在 .NET Core、Mono 或非 Windows 平台上, 尚不支持验证已签名的包。

用法

```
nuget verify <-All|-Signatures> <package(s)> [options]
```

其中 `<package(s)>`, 是一个或 `.nupkg` 多个文件。

nuget 验证-全部

指定应在包上执行所有验证。

nuget 验证-签名

指定应执行的包签名验证。

用于 "验证-签名" 的选项

■	■
CertificateFingerprint	指定一个或多个证书的一个或多256个证书指纹, 证书必须用来签名包。证书 256 SHA-1 指纹是证书的 SHA-256 哈希。多个输入应以分号分隔。

选项

■	■
ConfigFile	要应用的 NuGet 配置文件。如果未指定, <code>%AppData%\NuGet\NuGet.Config</code> 则使用 (Windows <code>~/.nuget/NuGet/NuGet.Config</code>) 或 (Mac/Linux)。
ForceEnglishOutput	使用固定的、基于英语的区域性强制执行 nuget.exe。
Help	显示命令的帮助信息。
Verbosity	指定在输出中显示的详细信息量: "正常"、"静默"、"详细"。

示例

```
nuget verify -Signatures .\..\MyPackage.nupkg -CertificateFingerprint  
"CE40881FF5F0AD3E58965DA20A9F571EF1651A56933748E1BF1C99E537C4E039;5F874AAF47BCB268A19357364E7FBB09D6BF9E8A93E  
1229909AC5CAC865802E2" -Verbosity detailed  
  
nuget verify -Signatures c:\packages\MyPackage.nupkg -CertificateFingerprint  
CE40881FF5F0AD3E58965DA20A9F571EF1651A56933748E1BF1C99E537C4E039  
  
nuget verify -Signatures MyPackage.nupkg -Verbosity quiet  
  
nuget verify -Signatures .\*.nupkg  
  
nuget verify -All .\*.nupkg
```

受信任的签名者命令 (NuGet CLI)

2019/11/5 • [Edit Online](#)

适用于：• 支持的版本的包使用: 4.9.1 +

获取或设置 NuGet 配置的受信任签署者。有关其他用法，请参阅[常见 NuGet 配置](#)。有关 nuget.exe 架构的外观的详细信息，请参阅[nuget 配置文件参考](#)。

用法

```
nuget trusted-signers <list|add|remove|sync> [options]
```

如果未指定 `list|add|remove|sync`，则该命令将默认为 `list`。

nuget 可信-签署者列表

列出配置中的所有受信任签署者。此选项将包含每个签署者的所有证书(具有指纹和指纹算法)。如果证书具有前面的 `[U]`，则表示证书条目的 `allowUntrustedRoot` 设置为 `true`。

下面是此命令的示例输出：

```
$ nuget trusted-signers
Registered trusted signers:

1. nuget.org [repository]
Service Index: https://api.nuget.org/v3/index.json
Certificate fingerprint(s):
SHA256 - 0E5F38F57DC1BCC806D8494F4F90FBCEDD988B46760709CBEEC6F4219AA6157D

2. microsoft [author]
Certificate fingerprint(s):
SHA256 - 3F9001EA83C560D712C24CF213C3D312CB3BFF51EE89435D3430BD06B5D0EECE

3. myUntrustedAuthorSignature [author]
Certificate fingerprint(s):
[U] SHA256 - 518F9CF082C0872025EFB2587B6A6AB198208F63EA58DD54D2B9FF6735CA4434
```

nuget 可信-签署者添加 [选项]

向配置添加具有给定名称的受信任的签署者。此选项具有不同的手势来添加受信任的作者或存储库。

基于包的 "添加" 选项

```
nuget trusted-signers add <package(s)> -Name <name> [options]
```

其中 `<package(s)>` 是一个或多个 `.nupkg` 文件。

作者	指定应信任包的作者签名。
存储库	指定应信任包的存储库签名或副署。
AllowUntrustedRoot	指定是否允许受信任的签名者的证书链接到不受信任的根。
Owners	以分号分隔的受信任所有者列表，进一步限制了存储库的信任。仅在使用 <code>-Repository</code> 选项时有效。

不支持同时提供 `-Author` 和 `-Repository`。

基于服务索引添加的选项

```
nuget trusted-signers add -Name <name> [options]
```

注意:此选项将只添加受信任的存储库。

ServiceIndex	指定要信任的存储库的 V3 服务索引。此存储库必须支持存储库签名资源。如果未提供，则该命令将查找具有相同 <code>-Name</code> 的包源，并从中获取服务索引。
AllowUntrustedRoot	指定是否允许受信任的签名者的证书链接到不受信任的根。
Owners	以分号分隔的受信任所有者列表，进一步限制了存储库的信任。

基于证书信息添加的选项

```
nuget trusted-signers add -Name <name> [options]
```

注意:如果已存在具有给定名称的受信任的签名者，则会将证书项目添加到该签名者。否则，将创建一个受信任的作者，其中包含来自给定证书信息的证书项目。

CertificateFingerprint	指定证书的证书指纹，必须使用该证书对签名的包进行签名。证书指纹是证书的哈希。用于计算此哈希的哈希算法应在 <code>FingerprintAlgorithm</code> 选项中指定。
FingerprintAlgorithm	指定用于计算证书指纹的哈希算法。默认为 <code>SHA256</code> 。支持的值为 <code>SHA256</code> 、 <code>SHA384</code> 和 <code>SHA512</code>
AllowUntrustedRoot	指定是否允许受信任的签名者的证书链接到不受信任的根。

nuget 可信-签名者删除名称 <名称>

删除任何与给定名称匹配的受信任签署者。

nuget 可信-签署者同步-名称 <名称>

请求当前受信任的存储库中使用的最新证书列表，以更新受信任的签署者中的现有证书列表。

注意:此笔势将删除当前的证书列表，并将其替换为存储库中的最新列表。

选项

参数	说明
Read-configfile	要应用的 NuGet 配置文件。如果未指定，则使用 %AppData%\NuGet\NuGet.Config (Windows) 或 ~/ .nuget/NuGet/NuGet.Config (Mac/Linux)。
ForceEnglishOutput	使用固定的、基于英语的区域性强制执行 nuget.exe。
帮助	显示命令的帮助信息。
详细级别	指定在输出中显示的详细信息量：“正常”、“静默”、“详细”。

示例

```
nuget trusted-signers list

nuget trusted-signers Add -Name existingSource

nuget trusted-signers Add -Name trustedRepo -ServiceIndex https://trustedRepo.test/v3ServiceIndex

nuget trusted-signers Add -Name author1 -CertificateFingerprint
CE40881FF5F0AD3E58965DA20A9F571EF1651A56933748E1BF1C99E537C4E039 -FingerprintAlgorithm SHA256

nuget trusted-signers Add -Repository ..\..\MyRepositorySignedPackage.nupkg -Name TrustedRepo

nuget-trusted-signers Remove -Name TrustedRepo

nuget-trusted-signers Sync -Name TrustedRepo
```

NuGet CLI 环境变量

2019/7/19 • [Edit Online](#)

可以通过多个环境变量来配置 nuget.exe CLI 的行为, 这会影响计算机范围、用户或进程级别上的 nuget.exe。环境变量始终覆盖文件中 `NuGet.Config` 的任何设置, 允许生成服务器更改适当的设置, 而无需修改任何文件。

一般情况下, 直接在命令行上或在 NuGet 配置文件中指定的选项优先, 但有一些例外, 如`FORCE_NUGET_EXE_INTERACTIVE`。如果在不同的计算机之间发现 nuget.exe 的行为不同, 则可能是环境变量。例如, 在部署期间使用的 Azure Web Apps Kudu 将`NUGET_XMLDOC_MODE`设置为 "跳过", 以加快包还原性能并节省磁盘空间。

NuGet CLI 使用 MSBuild 读取项目文件。在 MSBuild 计算期间, 所有环境变量都作为属性提供。[NuGet 包和还原为 MSBuild 目标](#)中记录的属性列表也可以设置为环境变量。

环境变量	描述	示例
<code>http_proxy</code>	用于 NuGet HTTP 操作的 Http 代理。	这将指定为 <code>http://<username>:<password>@proxy.com</code> 。
<code>no_proxy</code>	将使用代理的域配置为绕过。	指定为以逗号 (,) 分隔的域。
<code>EnableNuGetPackageRestore</code>	如果 NuGet 在还原时需要此权限, NuGet 应隐式授予同意。	指定标志被视为 <code>true</code> 或 1, 其他任何值视为标志未设置。
<code>NUGET_EXE_NO_PROMPT</code>	阻止 exe 提示输入凭据。	Null 或空字符串以外的任何值都将被视为此标志设置/ <code>true</code> 。
<code>FORCE_NUGET_EXE_INTERACTIVE</code>	强制交互模式的全局环境变量。	Null 或空字符串以外的任何值都将被视为此标志设置/ <code>true</code> 。
<code>NUGET_PACKAGES</code>	用于全局包文件夹的路径, 如 管理全局包和缓存文件夹 中所述。	指定为绝对路径。
<code>NUGET_FALLBACK_PACKAGES</code>	全局备用包文件夹。	绝对文件夹路径, 用分号 (;) 分隔。
<code>NUGET_HTTP_CACHE_PATH</code>	用于 <code>http</code> 缓存文件夹的路径, 如 管理全局包和缓存文件夹 中所述。	指定为绝对路径。
<code>NUGET_PERSIST_DG</code>	指示是否应持久保存 dg 文件 (从 MSBuild 收集的数据) 的标志。	指定为 <code>true</code> 或 <code>false</code> (默认值), 如果未设置 <code>NUGET_PERSIST_DG_PATH</code> , 则将存储到临时目录 (当前环境 Temp 目录中的 NuGetScratch 文件夹)。
<code>NUGET_PERSIST_DG_PATH</code>	持久保存 dg 文件的路径。	指定为绝对路径, 仅当 <code>NUGET_PERSIST_DG</code> 设置为 <code>true</code> 时才使用此选项。

NUGET_RESTORE_MSBUILD_ARGS	设置其他 MSBuild 参数。	传递参数的方式与将它们传递给 msbuild.exe 的方式相同。将项目属性 "Foo" 从命令行设置为值栏的示例如下所示: Foo = Bar
NUGET_RESTORE_MSBUILD_VERBOSITY	设置 MSBuild 日志详细级别。	默认值为 <i>quiet</i> ("v: q")。可能的值 <i>quiet</i> 、 <i>m [minimal]</i> 、 <i>n [normal]</i> 、 <i>d [detailed]</i> 和 <i>诊断 [nistic]</i> 。
NUGET_SHOW_STACK	确定是否应向用户显示完全异常 (包括堆栈跟踪)。	指定为 <i>true</i> 或 <i>false</i> (默认值)。
NUGET_XMLDOC_MODE	确定如何处理程序集 XML 文档文件提取。	支持的模式为 <i>skip</i> (不提取 xml 文档文件), <i>压缩</i> (将 xml 文档文件存储为 zip 存档) 或 <i>none</i> (默认情况下, 将 xml 文档文件视为常规文件)。
NUGET_CERT_REVOCATION_MODE	确定用于对包进行签名的证书的吊销状态检查如何在安装或还原已签名的包时执行。如果未设置, 则默认 <i>online</i> 为。	可能的值 <i>联机</i> (默认值), <i>脱机</i> 。 与 NU3028 相关

长路径支持 (NuGet CLI)

2020/2/6 • [Edit Online](#)

适用于：所有 • 支持的版本：4.8 +

Nuget.exe 4.8 和更高版本支持文件和目录的长路径，适用于打包、还原、安装和需要文件路径的大多数其他方案。

所需操作系统

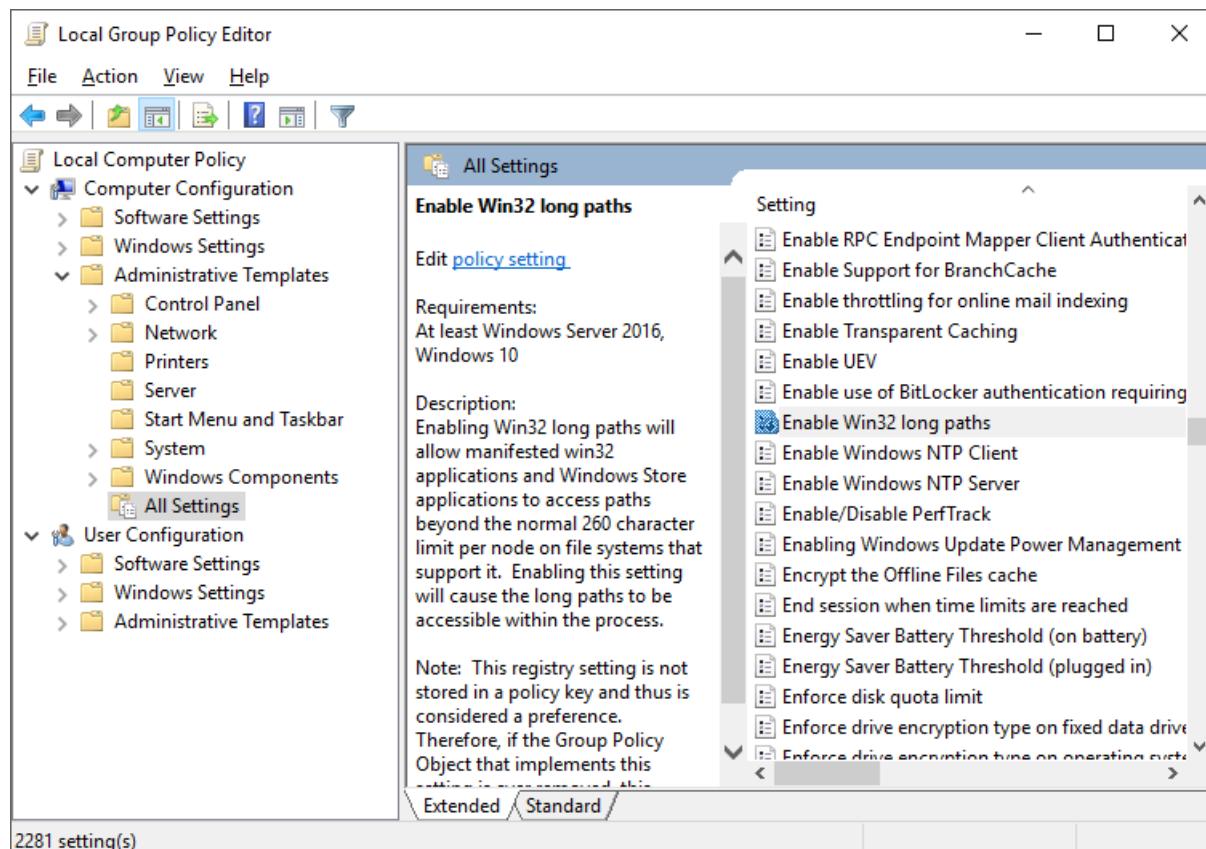
- Windows 10 (版本1607或更高版本)
- 如果将 .NET Framework 升级到版本4.6.2 或更高版本，则 Windows 10 (2015年7月版或版本1511)。
- Windows Server 2016 (所有版本)

启用 "Win32 长路径" 组策略

需要通过设置组策略在这些系统上启用长路径支持。

步骤：

- 启动组策略编辑器-在 "开始" 搜索栏中键入 "编辑组策略"，或从 "运行" 命令运行 "gpedit.msc" (Windows-R)。
- 在本地组策略编辑器中，启用 "本地计算机策略/计算机配置/管理模板/所有设置/启用 Win32 长路径"。



NOTE

启用其他 NuGet 工具以支持长路径

- Dotnet CLI 支持长路径，而不考虑操作系统或版本。
- Visual Studio 或 `msbuild -t:restore` 尚不支持长路径。
- 使用 NuGet 库执行还原和其他命令的软件将支持 Nuget.exe 使用的同一系统上的长路径，前提是它们还在 windows 清单中设置 `longPathAware`，并通过 App.config 将 `useLegacyPathHandling` 配置为 `false` 的[详细信息](#)

PowerShell 参考

2020/1/8 • [Edit Online](#)

程序包管理器控制台在 Windows 上的 Visual Studio 中提供了一个 PowerShell 接口，可通过下面列出的特定命令与 NuGet 交互。（目前，控制台目前不可用于 Visual Studio for Mac。）有关使用控制台的指南，请参阅[使用程序包管理器控制台安装和管理包](#)主题。

TIP

所有 PowerShell 命令仅与包消耗相关。除了包还可以是其他包的使用者以外，没有任何 PowerShell 命令都与创建和发布包相关。

IMPORTANT

此处列出的命令特定于 Visual Studio 中的包管理器控制台，不同于常规 PowerShell 环境中提供的[包管理模块命令](#)。具体而言，每个环境都有一些命令，这些命令在其他环境中不可用，并且其特定参数的名称相同的命令也可能不同。使用 Visual Studio 中的包管理控制台时，适用于本主题中所述的命令和参数。

命令	描述	NuGet 命令
Install-Package	将程序包及其依赖项安装到项目中。	全部
Update-Package	更新包及其依赖项或项目中的所有包。	全部
Find-Package	使用包 ID 或关键字搜索包源。	3.0+
Get-Package	检索本地存储库中安装的包的列表，或列出包源中可用的包。	全部
命令	描述	NuGet 命令
Add-BindingRedirect	检查项目的输出路径中的所有程序集，并在必要时将绑定重定向添加到 <code>app.config</code> 或 <code>web.config</code> 。	全部
Get-Project	显示有关默认项目或指定项目的信息。	3.0+
Open-PackagePage	用指定包的项目、许可证或报表滥用 URL 启动默认浏览器。	3.0+ 中弃用
Register-TabExpansion	为命令的参数注册选项卡展开，使你可以为常用参数值创建自定义扩展。	全部
Sync-Package	获取指定项目中已安装包的版本，并将该版本同步到解决方案中项目的其余部分。	3.0+
Uninstall-Package	从项目中删除包，并可以选择删除其依赖项。	全部

若要获取有关控制台中任何这些命令的完整详细信息，只需运行以下命令名称：

```
Get-Help <command> -full
```

所有程序包管理器控制台命令都支持以下[常见的 PowerShell 参数](#)：

- 调试
- ErrorAction
- ErrorVariable
- OutBuffer
- OutVariable
- PipelineVariable
- 详细
- WarningAction
- WarningVariable

有关详细信息，请参阅 PowerShell 文档中的[about_CommonParameters](#)。

Add-BindingRedirect (Visual Studio 中的程序包管理器控制台)

2020/1/8 • [Edit Online](#)

仅在 Windows 上的 Visual Studio 中的[包管理器控制台](#)内可用。

检查项目的输出路径中的所有程序集，并在必要时向应用程序或 web 配置文件添加绑定重定向。安装包时，会自动运行此命令。

有关绑定重定向及其使用原因的详细信息，请参阅 .NET 文档中的[重定向程序集版本](#)。

语法

```
Add-BindingRedirect [-ProjectName] <string> [<CommonParameters>]
```

参数

■

■

ProjectName

请求要向其添加绑定重定向的项目。-项目开关本身是可选的。

这些参数都不接受管道输入或通配符。

通用参数

Add-BindingRedirect 支持以下[常见的 PowerShell 参数](#)：调试、错误操作、ErrorVariable、OutBuffer、OutVariable、PipelineVariable、Verbose、WarningAction 和 WarningVariable。

示例

```
Add-BindingRedirect MyProject
```

```
Add-BindingRedirect -ProjectName MyProject
```

Find-Package (Visual Studio 中的程序包管理器控制台)

2020/1/8 • [Edit Online](#)

版本 3.0 +; 本主题介绍 Windows 上 Visual Studio 中的包管理器控制台内的命令。对于“通用 PowerShell 查找包”命令，请参阅[PowerShell PackageManagement 参考](#)。

从包源中获取具有指定 ID 或关键字的一组远程包。

语法

```
Find-Package [-Id] <关键字> -Source <string> [-AllVersions] [-First [<int>]]  
[-Skip <int>] [-IncludePrerelease] [-ExactMatch] [-StartWith] [<CommonParameters>]
```

参数

参数	描述
-Id <关键字>	请求搜索包源时要使用的关键字。使用-ExactMatch 仅返回其包 ID 与关键字匹配的那些包。如果未提供任何关键字， <code>Find-Package</code> 将返回前20个包的列表(通过下载)或由-First 指定的数字。请注意，-Id 是可选的，而不是操作。
-Source	要搜索的包源的 URL 或文件夹路径。本地文件夹路径可以是绝对路径，也可以是相对于当前文件夹的路径。如果省略，则 <code>Find-Package</code> 搜索当前选定的包源。
-AllVersions	显示每个包的所有可用版本，而不是仅显示最新版本。
-First	要从列表开头返回的包数;默认值为20。
-Skip	省略所显示列表中的第一个 <int> 包。
-IncludePrerelease	在结果中包括预发布包。
-ExactMatch	指定为使用 <关键字> 为区分大小写的包 ID。
-StartWith	返回包 ID 以 <关键字开头> 的包。

这些参数都不接受管道输入或通配符。

通用参数

`Find-Package` 支持以下[常见的 PowerShell 参数](#):调试、错误操作、ErrorVariable、OutBuffer、OutVariable、PipelineVariable、Verbose、WarningAction 和 WarningVariable。

示例

```
# Find packages containing keywords
Find-Package elmah
Find-Package logging

# List packages whose ID begins with Elmah
Find-Package Elmah -StartWith

# By default, Get-Package returns a list of 20 packages; use -First to show more
Find-Package logging -First 100

# List all versions of the package with the ID of "jquery"
Find-Package jquery -AllVersions -ExactMatch
```

Get-Package (Visual Studio 中的程序包管理器控制台)

2020/1/8 • [Edit Online](#)

本主题介绍 Windows 上 Visual Studio 中的包管理器控制台内的命令。对于“通用 PowerShell 获取包”命令，请参阅[PowerShell PackageManagement 参考](#)。

检索本地存储库中安装的包的列表，列出包源中与 -ListAvailable 开关一起使用时可用的包，或在与 -Update 开关一起使用时列出可用的更新。

语法

```
Get-Package -Source <string> [-ListAvailable] [-Updates] [-ProjectName <string>]
[-Filter <string>] [-First <int>] [-Skip <int>] [-AllVersions] [-IncludePrerelease]
[-PageSize] [<CommonParameters>]
```

如果没有参数，`Get-Package` 将显示默认项目中安装的包的列表。

参数

参数	说明
Source	包的 URL 或文件夹路径。本地文件夹路径可以是绝对路径，也可以是相对于当前文件夹的路径。如果省略，则 <code>Get-Package</code> 搜索当前选定的包源。当与 -ListAvailable 一起使用时，默认为 nuget.org。
ListAvailable	列出包源中可用的包，默认为 nuget.org。显示默认的 50 包，除非指定了 -PageSize 和/或 -First。
更新	列出具有包源中可用更新的包。
ProjectName	要从中获取已安装包的项目。如果省略，则返回整个解决方案的已安装项目。
筛选器筛选器	一个筛选器字符串，用于通过将包列表应用到包 ID、说明和标记来缩小包列表范围。
First	要从列表开头返回的程序包数。如果未指定，则默认为 50。
Skip	省略所显示列表中的第一个 <int> 包。
AllVersions	显示每个包的所有可用版本，而不是仅显示最新版本。
IncludePrerelease	在结果中包括预发布包。
PageSize	(3.0+) 当与 -ListAvailable (必需) 一起使用时，会在提示继续操作之前列出要列出的包数。

这些参数都不接受管道输入或通配符。

通用参数

`Get-Package` 支持以下[常见的 PowerShell 参数](#): 调试、错误操作、ErrorVariable、OutBuffer、OutVariable、PipelineVariable、Verbose、WarningAction 和 WarningVariable。

示例

```
# Lists the packages installed in the current solution
Get-Package

# Lists the packages installed in a project
Get-Package -ProjectName MyProject

# Lists packages available in the current package source
Get-Package -ListAvailable

# Lists 30 packages at a time from the current source, and prompts to continue if more are available
Get-Package -ListAvailable -PageSize 30

# Lists packages with the Ninject keyword in the current source, up to 50
Get-Package -ListAvailable -Filter Ninject

# List all versions of packages matching the filter "jquery"
Get-Package -ListAvailable -Filter jquery -AllVersions

# Lists packages installed in the solution that have available updates
Get-Package -Updates

# Lists packages installed in a specific project that have available updates
Get-Package -Updates -ProjectName MyProject
```

Get-Project (Visual Studio 中的程序包管理器控制台)

2020/1/8 • [Edit Online](#)

仅在 Windows 上的 Visual Studio 中的包管理器控制台内可用。

显示有关默认项目或指定项目的信息。Get-Project 专门返回对项目的 Visual Studio DTE (开发工具环境) 对象的引用。

语法

```
Get-Project [[-Name] <string>] [-All] [<CommonParameters>]
```

参数

参数	描述
Name	指定要显示的项目， 默认为在包管理器控制台中选择的默认项目。-Name 开关本身是可选的。
全部	显示解决方案中每个项目的信息；项目的顺序不确定。

这些参数都不接受管道输入或通配符。

通用参数

Get-Project 支持以下常见的 PowerShell 参数：调试、错误操作、ErrorVariable、OutBuffer、OutVariable、PipelineVariable、Verbose、WarningAction 和 WarningVariable。

示例

```
# Displays information for the default project
Get-Project

# Displays information for a project in the solution
Get-Project MyProjectName

# Displays information for all projects in the solution
Get-Project -All
```

Install-Package (Visual Studio 中的程序包管理器控制台)

2020/1/8 • [Edit Online](#)

本主题介绍 Windows 上 Visual Studio 中的包管理器控制台内的命令。对于“通用 PowerShell 安装包”命令，请参阅[PowerShell PackageManagement 参考](#)。

将包及其依赖项安装到项目中。

语法

```
Install-Package [-Id] <string> [-IgnoreDependencies] [-ProjectName <string>] [[-Source] <string>]
    [[-Version] <string>] [-IncludePrerelease] [-FileConflictAction] [-DependencyVersion]
    [-WhatIf] [<CommonParameters>]
```

在 NuGet 2.8+ 中，`Install-Package` 可以在项目中降级现有包。例如，如果安装了 5.1.0-rc1，以下命令会将其降级到 5.0.0：

```
Install-Package Microsoft.AspNet.Mvc -Version 5.0.0.
```

参数

参数	说明
<code>-Id</code>	请求要安装的包的标识符。 <code>(3.0+)</code> 标识符可以是 <code>packages.config</code> 文件或 <code>.nupkg</code> 文件的路径或 URL。 <code>-Id</code> 开关本身是可选的。
<code>-IgnoreDependencies</code>	仅安装此程序包，而不安装其依赖项。
<code>-ProjectName</code>	要在其中安装包的项目，默认为默认项目。
<code>-Source</code>	要搜索的包源的 URL 或文件夹路径。本地文件夹路径可以是绝对路径，也可以是相对于当前文件夹的路径。如果省略，则 <code>Install-Package</code> 搜索当前选定的包源。
<code>{2>版本<2}</code>	要安装的包的版本，默认为最新版本。
<code>-IncludePrerelease</code>	考虑安装的预发行程序包。如果省略，则只考虑稳定程序包。
<code>-FileConflictAction</code>	当要求覆盖或忽略项目引用的现有文件时要执行的操作。可能的值包括 <code>Overwrite</code> 、 <code>Ignore</code> 、 <code>None</code> 、 <code>OverwriteAll</code> 和 <code>(3.0+)</code> <code>IgnoreAll</code> 。

<p>DependencyVersion</p>	<p>要使用的依赖项包的版本，可以是下列项之一：</p> <ul style="list-style-type: none"> • 最低(默认值): 最低版本 • HighestPatch: 最小主要、次要和最高修补程序的版本 • HighestMinor: 最小主要版本号最高的版本，最高修补程序 • 最高(不带参数的更新包的默认值): 最高版本 <p>您可以使用 <code>Nuget.Config</code> 文件中的 <code>dependencyVersion</code> 设置设置默认值。</p>
<p>WhatIf</p>	<p>显示运行命令时，如果不实际执行安装，会发生什么情况。</p>

这些参数都不接受管道输入或通配符。

通用参数

`Install-Package` 支持以下常见的 PowerShell 参数：调试、错误操作、ErrorVariable、OutBuffer、OutVariable、PipelineVariable、Verbose、WarningAction 和 WarningVariable。

示例

```
# Installs the latest version of Elmah from the current source into the default project
Install-Package Elmah

# Installs Glimpse 1.0.0 into the MvcApplication1 project
Install-Package Glimpse -Version 1.0.0 -Project MvcApplication1

# Installs Ninject.Mvc3 but not its dependencies from c:\temp\packages
Install-Package Ninject.Mvc3 -IgnoreDependencies -Source c:\temp\packages

# Installs the package listed on the online packages.config into the current project
# Note: the URL must end with "packages.config"
Install-Package https://raw.githubusercontent.com/linked-data-dotnet/json-
ld.net/master/.nuget/packages.config

# Installs jquery 1.10.2 package, using the .nupkg file under local path of c:\temp\packages
Install-Package c:\temp\packages\jQuery.1.10.2.nupkg

# Installs the specific online package
# Note: the URL must end with ".nupkg"
Install-Package https://globalcdn.nuget.org/packages/microsoft.aspnet.mvc.5.2.3.nupkg
```

Open-PackagePage (Visual Studio 中的程序包管理器控制台)

2020/1/8 • [Edit Online](#)

3.0+ 中弃用仅在 Windows 上的 Visual Studio 中的包管理器控制台内可用。

用指定包的项目、许可证或报表滥用 URL 启动默认浏览器。

语法

```
Open-PackagePage [-Id] <string> [-Version] [-Source] [-License] [-ReportAbuse]
                  [-PassThru] [<CommonParameters>]
```

参数

参数	说明
-Id	所需包的包 ID。-Id 开关本身是可选的。
{2>版本<2}	包的版本，默认为最新版本。
Source	包源，默认为“源”下拉选项中的所选源。
许可证	打开浏览器，直至包的许可证 URL。如果未指定-License 和-ReportAbuse，则浏览器将打开包的项目 URL。
ReportAbuse	打开浏览器，指向包的“报告滥用 URL”。如果未指定-License 和-ReportAbuse，则浏览器将打开包的项目 URL。
PassThru	显示 URL；使用-WhatIf 来禁止打开浏览器。

这些参数都不接受管道输入或通配符。

通用参数

Open-PackagePage 支持以下常见的 PowerShell 参数：调试、错误操作、ErrorVariable、OutBuffer、OutVariable、PipelineVariable、Verbose、WarningAction 和 WarningVariable。

示例

```
# Opens a browser with the Ninject package's project page
Open-PackagePage Ninject

# Opens a browser with the Ninject package's license page
Open-PackagePage Ninject -License

# Opens a browser with the Ninject package's report abuse page
Open-PackagePage Ninject -ReportAbuse

# Assigns the license URL to the variable, $url, without launching the browser
$url = Open-PackagePage Ninject -License -PassThru -WhatIf
```

Sync-Package (Visual Studio 中的程序包管理器控制台)

2020/1/8 • [Edit Online](#)

版本 3.0 +; 仅在 Windows 上的 Visual Studio 中的包管理器控制台内可用。

从指定的(或默认的)项目获取已安装包的版本，并将该版本同步到解决方案中项目的其余部分。

语法

```
Sync-Package [-Id] <string> [-IgnoreDependencies] [-ProjectName <string>] [[-Version] <string>]
    [[-Source] <string>] [-IncludePrerelease] [-FileConflictAction] [-DependencyVersion]
    [-WhatIf] [<CommonParameters>]
```

参数

参数	说明
-Id	请求要同步的包的标识符。-Id 开关本身是可选的。
IgnoreDependencies	仅安装此程序包，而不安装其依赖项。
ProjectName	要从中同步包的项目，默认为默认项目。
{2>版本<2}	要同步的包版本，默认为当前安装的版本。
Source	要搜索的包源的 URL 或文件夹路径。本地文件夹路径可以是绝对路径，也可以是相对于当前文件夹的路径。如果省略，则 Sync-Package 搜索当前选定的包源。
IncludePrerelease	包括同步中的预发行包。
FileConflictAction	当要求覆盖或忽略项目引用的现有文件时要执行的操作。可能的值包括 Overwrite、Ignore、None、OverwriteAll 和 (3.0+) IgnoreAll。
DependencyVersion	要使用的依赖项包的版本，可以是下列项之一： <ul style="list-style-type: none">最低(默认值): 最低版本HighestPatch: 最小主要、次要和最高修补程序的版本HighestMinor: 最小主要版本号最高的版本，最高修补程序最高(不带参数的更新包的默认值): 最高版本 您可以使用 Nuget.Config 文件中的 dependencyVersion 设置设置默认值。
WhatIf	显示运行命令时，不实际执行同步的情况。

这些参数都不接受管道输入或通配符。

通用参数

`Sync-Package` 支持以下[常见的 PowerShell 参数](#): 调试、错误操作、ErrorVariable、OutBuffer、OutVariable、PipelineVariable、Verbose、WarningAction 和 WarningVariable。

示例

```
# Sync the Elmah package installed in the default project into the other projects in the solution
Sync-Package Elmah

# Sync the Elmah package installed in the ClassLibrary1 project into other projects in the solution
Sync-Package Elmah -ProjectName ClassLibrary1

# Sync Microsoft.AspNet.package but not its dependencies into the other projects in the solution
Sync-Package Microsoft.AspNet.Mvc -IgnoreDependencies

# Sync jQuery.Validation and install the highest version of jQuery (a dependency) from the package source
Sync-Package jQuery.Validation -DependencyVersion highest
```

Uninstall-Package (Visual Studio 中的程序包管理器控制台)

2020/1/8 • [Edit Online](#)

本主题介绍 Windows 上 Visual Studio 中的包管理器控制台内的命令。对于“通用 PowerShell 卸载包”命令，请参阅[PowerShell PackageManagement 参考](#)。

从项目中删除包，并可以选择删除其依赖项。如果其他程序包依赖于该程序包，则除非指定了 -Force 选项，否则此命令将失败。

语法

```
Uninstall-Package [-Id] <string> [-RemoveDependencies] [-ProjectName <string>] [-Force]
[-Version <string>] [-WhatIf] [<CommonParameters>]
```

如果其他程序包依赖于该程序包，则除非指定了 -Force 选项，否则此命令将失败。

参数

参数	说明
-Id	请求要卸载的包的标识符。-Id 开关本身是可选的。
{2> 版本 <2}	要卸载的包的版本，默认为当前安装的版本。
RemoveDependencies	卸载包及其未使用的依赖项。也就是说，如果任何依赖项具有依赖于它的其他包，则会跳过它。
ProjectName	要从中卸载包的项目，默认为默认项目。
Force	强制卸载包，即使其他程序包依赖于它。
WhatIf	显示运行命令时将发生的情况，但不实际执行卸载。

这些参数都不接受管道输入或通配符。

通用参数

`Uninstall-Package` 支持以下[常见的 PowerShell 参数](#)：调试、错误操作、ErrorVariable、OutBuffer、OutVariable、PipelineVariable、Verbose、WarningAction 和 WarningVariable。

示例

```
# Uninstalls the Elmah package from the default project
Uninstall-Package Elmah

# Uninstalls the Elmah package and all its unused dependencies
Uninstall-Package Elmah -RemoveDependencies

# Uninstalls the Elmah package even if another package depends on it
Uninstall-Package Elmah -Force
```

Update-Package (Visual Studio 中的程序包管理器控制台)

2020/1/8 • [Edit Online](#)

仅在 Windows 上的 Visual Studio 中的 NuGet 包管理器控制台内可用。

将包及其依赖项或项目中的所有包更新到较新的版本。

语法

```
Update-Package [-Id] <string> [-IgnoreDependencies] [-ProjectName <string>] [-Version <string>]
[-Safe] [-Source <string>] [-IncludePrerelease] [-Reinstall] [-FileConflictAction]
[-DependencyVersion] [-ToHighestPatch] [-ToHighestMinor] [-WhatIf] [<CommonParameters>]
```

在 NuGet 2.8 + 中，可以使用 `Update-Package` 来降级项目中的现有包。例如，如果安装了 5.1.0-rc1，以下命令会将其降到 5.0.0：

```
Update-Package Microsoft.AspNet.Mvc -Version 5.0.0.
```

参数

参数	说明
<code>-Id</code>	要更新的包的标识符。如果省略，则更新所有包。 <code>-Id</code> 开关本身是可选的。
<code>-IgnoreDependencies</code>	跳过更新包的依赖项。
<code>-ProjectName</code>	包含要更新的包的项目的名称，默认为 "所有项目"。
<code>{2>版本<2}</code>	要用于升级的版本，默认为最新版本。在 NuGet 3.0 + 中，版本值必须为 最低 、 最高 、 HighestMinor 或 HighestPatch （等效于 <code>-Safe</code> ）之一。
<code>-Safe</code>	仅将升级限制为与当前安装的包具有相同的主版本和次版本的版本。
<code>-Source</code>	要搜索的包源的 URL 或文件夹路径。本地文件夹路径可以是绝对路径，也可以是相对于当前文件夹的路径。如果省略，则 <code>Update-Package</code> 搜索当前选定的包源。
<code>-IncludePrerelease</code>	包含更新的预发布包。
<code>-Reinstall</code>	使用当前安装的版本重装所有包。请参阅 重新安装和更新包 。

FileConflictAction	当要求覆盖或忽略项目引用的现有文件时要执行的操作。可能的值包括 <i>Overwrite</i> 、 <i>Ignore</i> 、 <i>None</i> 、 <i>OverwriteAll</i> 和 <i>IgnoreAll</i> (3.0+)。
DependencyVersion	<p>要使用的依赖项包的版本，可以是下列项之一：</p> <ul style="list-style-type: none"> • 最低(默认值): 最低版本 • HighestPatch: 最小主要、次要和最高修补程序的版本 • HighestMinor: 最小主要版本号最高的版本，最高修补程序 • 最高(不带参数的更新包的默认值): 最高版本 <p>您可以使用 <code>Nuget.Config</code> 文件中的 <code>dependencyVersion</code> 设置设置默认值。</p>
ToHighestPatch	等效于 <code>-Safe</code> 。
ToHighestMinor	仅将升级限制为版本与当前安装的包相同的版本。
WhatIf	显示运行命令时，不实际执行更新的情况。

这些参数都不接受管道输入或通配符。

通用参数

`Update-Package` 支持以下常见的 PowerShell 参数：调试、错误操作、ErrorVariable、OutBuffer、OutVariable、PipelineVariable、Verbose、WarningAction 和 WarningVariable。

示例

```
# Updates all packages in every project of the solution
Update-Package

# Updates every package in the MvcApplication1 project
Update-Package -ProjectName MvcApplication1

# Updates the Elmah package in every project to the latest version
Update-Package Elmah

# Updates the Elmah package to version 1.1.0 in every project showing optional -Id usage
Update-Package -Id Elmah -Version 1.1.0

# Updates the Elmah package within the MvcApplication1 project to the highest "safe" version.
# For example, if Elmah version 1.0.0 of a package is installed, and versions 1.0.1, 1.0.2,
# and 1.1 are available in the feed, the -Safe parameter updates the package to 1.0.2 instead
# of 1.1 as it would otherwise.
Update-Package Elmah -ProjectName MvcApplication1 -Safe

# Reinstall the same version of the original package, but with the latest version of dependencies
# (subject to version constraints). If this command rolls a dependency back to an earlier version,
# use Update-Package <dependency_name> to reinstall that one dependency without affecting the
# dependent package.
Update-Package Elmah -reinstall

# Reinstall the Elmah package in just MyProject
Update-Package Elmah -ProjectName MyProject -reinstall

# Reinstall the same version of the original package without touching dependencies.
Update-Package Elmah -reinstall -ignoreDependencies
```

NuGet 服务器 API

2020/3/19 • [Edit Online](#)

NuGet 服务器 API 是一组 HTTP 终结点，可用于下载包、提取元数据、发布新包以及执行官方 NuGet 客户端中提供的大多数其他操作。

此 API 由 NuGet 客户端在 Visual Studio、nuget.exe 和 .NET CLI 中用来执行 NuGet 操作，例如 `dotnet restore`、在 VISUAL Studio UI 中搜索和 `nuget.exe push`。

请注意，在某些情况下，nuget.org 具有其他包源未强制执行的其他要求。[Nuget.org 协议](#)记录了这些差异。

有关可用的 nuget.exe 版本的简单枚举和下载，请参阅[node.js 终结点](#)。

服务索引

该 API 的入口点是众所周知的位置中的 JSON 文档。此文档称为 **服务索引**。Nuget.org 的服务索引的位置是 <https://api.nuget.org/v3/index.json>。

此 JSON 文档包含资源列表，这些资源提供不同的功能并实现不同的用例。

支持 API 的客户端应接受一个或多个服务索引 URL 作为连接到相应包源的方法。

有关服务索引的详细信息，请参阅[它的 API 参考](#)。

版本控制

API 是 NuGet 的 HTTP 协议的版本3。此协议有时称为 "V3 API"。这些参考文档将此版本的协议称为 "API"。

服务索引架构版本由服务索引中的 `version` 属性表示。API 规定版本字符串具有 `3` 的主要版本号。由于对服务索引架构进行了非重大更改，将增加版本字符串的次要版本。

较旧的客户端(如 nuget.exe 2.x)不支持 V3 API，并且仅支持旧版 V2 API，此处未介绍。

NuGet V3 API 命名为，因为它是 V2 API 的后续版本，后者是基于 OData 的协议，该协议是由8.x 版的官方 NuGet 客户端实现的。第一版的官方 NuGet 客户端3.0 支持 V3 API，但它仍然是 NuGet 客户端、4.0 和上支持的最新主要协议版本。

自首次发布以来，已对 API 进行了非重大协议更改。

资源和架构

服务索引描述了各种资源。当前支持的资源集如下所示：

资源	方法	描述
目录	否	所有包事件的完整记录。
<code>PackageBaseAddress</code>	是	获取包内容(. nupkg)。
<code>PackageDetailsUriTemplate</code>	否	构造用于访问包详细信息网页的 URL。
<code>PackagePublish</code>	是	推送和删除(或取消列出)包。

RegistrationsBaseUrl	是	获取包元数据。
ReportAbuseUriTemplate	否	构造用于访问报表滥用网页的 URL。
RepositorySignatures	否	获取用于存储库签名的证书。
SearchAutocompleteService	否	按子字符串发现包 Id 和版本。
SearchQueryService	是	按关键字筛选和搜索包。
SymbolPackagePublish	否	推送符号包。

通常，API 资源返回的所有非二进制数据都使用 JSON 进行序列化。服务索引中每个资源返回的响应架构分别为该资源定义。有关每个资源的详细信息，请参阅上面列出的主题。

将来，随着协议的发展，可能会向 JSON 响应中添加新的属性。为了使客户端能够进行进一步的验证，实现不应假定响应架构是最终的，不能包含额外的数据。应忽略实现不理解的所有属性。

NOTE

如果源未实现 `SearchAutocompleteService` 应正常禁用任何自动完成行为。如果未实现 `ReportAbuseUriTemplate`，官方 NuGet 客户端将回退到 nuget。组织的报表滥用 URL（由[NuGet/Home # 4924](#)跟踪）。其他客户端可能选择只是不向用户显示报表滥用 URL。

Nuget.org 上未记录的资源

Nuget.org 上的 V3 服务索引有一些未在上面列出的资源。不记录资源的原因有很多。

首先，我们不会将使用的资源记录为 nuget.org 的实现细节。`SearchGalleryQueryService` 属于此类别。

`NuGetGallery` 使用此资源将一些 V2 (OData) 查询委托给搜索索引，而不是使用数据库。之所以引入此资源是为了实现可伸缩性的原因，不能供外部使用。

其次，我们不会记录从未在官方客户端 RTM 版本中提供的资源。`PackageDisplayMetadataUriTemplate` 和 `PackageVersionDisplayMetadataUriTemplate` 属于此类别。

脾气，我们不会记录与 V2 协议紧密耦合的资源，这本身就是有意未记录的。`LegacyGallery` 资源属于此类别。此资源允许 V3 服务索引指向相应的 V2 源 URL。此资源支持 `nuget.exe list`。

如果此处未记录资源，强烈建议您不要对其进行依赖。我们可能会删除或更改这些未记录资源的行为，这些资源可能会以意外的方式中断您的实现。

时间戳

API 返回的所有时间戳均为 UTC，或者使用[ISO 8601](#)表示形式指定。

HTTP 方法

GET	执行只读操作，通常检索数据。
HEAD	获取相应 <code>GET</code> 请求的响应标头。

PUT	创建不存在的资源，如果该资源存在，则将其更新。某些资源可能不支持更新。
DELETE	删除或取消列出资源。

HTTP 状态代码

200	成功，并且存在响应正文。
201	成功，并创建了资源。
202	成功，请求已被接受，但一些工作可能仍未完成并以异步方式完成。
204	成功，但没有响应正文。
301	永久性重定向。
302	临时重定向。
400	URL 或请求正文中的参数无效。
401	提供的凭据无效。
403	给定提供的凭据不允许执行该操作。
404	请求的资源不存在。
409	请求与现有资源冲突。
500	服务遇到意外错误。
503	服务暂时不可用。

对 API 终结点发出的任何 `GET` 请求可能会返回 HTTP 重定向(301或302)。客户端应通过观察 `Location` 标头并发出后续 `GET` 来正常处理此类重定向。与特定终结点相关的文档将不会显式调用可以使用重定向的位置。

对于500级别的状态代码，客户端可以实现合理的重试机制。如果遇到任何500级状态代码或 TCP/IP 错误，官方 NuGet 客户端将重试三次。

HTTP 请求标头

X-NuGet-ApiKey	需要进行推送和删除，请参阅 PackagePublish 资源
X-NuGet-Client-Version	■并替换为 <code>X-NuGet-Protocol-Version</code>

X-NuGet-Protocol-Version	仅在某些情况下在 nuget.org 上是必需的。请参阅 nuget.org 协议
X-NuGet-Session-Id	可选。NuGet 客户端 4.7+ 识别属于同一 NuGet 客户端会话的 HTTP 请求。

`X-NuGet-Session-Id` 对于与 `PackageReference` 中的单个还原相关的所有操作都有一个值。对于其他方案，如自动完成和 `packages.config` 还原，可能有几个不同的会话 ID，原因是代码的分解方式。

Authentication

身份验证留给包源实现定义。对于 nuget.org，只有 `PackagePublish` 资源需要通过一个特殊 API 密钥标头进行身份验证。有关详细信息，请参阅 [PackagePublish 资源](#)。

自动完成

2019/8/16 • • [Edit Online](#)

可以使用 V3 API 构建包 ID 和版本自动完成体验。用于使自动完成查询的资源是在 [SearchAutocompleteService 服务索引](#) 中找到的资源。

版本管理

使用以下 `@type` 值:

<code>@TYPE</code>	
SearchAutocompleteService	初始版本
SearchAutocompleteService/3.0.0-beta	别名 <code>SearchAutocompleteService</code>
SearchAutocompleteService/3.0.0-rc	别名 <code>SearchAutocompleteService</code>

基 URL

以下 api 的基 URL 是与上述某个资源 `@id` `@type` 值关联的属性的值。在下面的文档中, 将使用占位符 `{@id}` 基 URL。

HTTP 方法

注册资源中找到的所有 url 都支持 HTTP 方法 `GET` 和 `HEAD`。

搜索包 Id

第一个自动完成 API 支持搜索包 ID 字符串的一部分。如果要在与 NuGet 包源集成的用户界面中提供包 typeahead 功能, 这非常有用。

只有未列出的版本的包将不会出现在结果中。

```
GET {@id}?q={QUERY}&skip={SKIP}&take={TAKE}&prerelease={PRERELEASE}&semVerLevel={SEMVERLEVEL}
```

请求参数

NAME				
q	URL	string	否	要与包 Id 进行比较的字符串
skip	URL	integer	否	要跳过的结果数, 用于分页
take	URL	integer	否	要返回的结果数, 用于分页

NAME				
早期	URL	boolean	否	<code>true</code> 或 <code>false</code> 确定是否包括 预发布包
semVerLevel	URL	string	否	SemVer 1.0.0 版本字符串

自动完成查询 `q` 按服务器实现所定义的方式进行分析。nuget.org 支持查询包 ID 令牌的前缀，该前缀是由拆分以大小写字符和符号字符为原始而生成的 ID 的组成部分。

`skip` 参数默认为 0。

`take` 参数应为大于零的整数。服务器实现可能会施加最大值。

如果 `prerelease` 未提供，则将排除预发布包。

Query 参数用于选择 SemVer 2.0.0 包。`semVerLevel` 如果排除此查询参数，则将仅返回带有 SemVer 1.0.0 兼容版本的包 Id (带有[标准的 NuGet 版本控制注意事项](#)，如具有 4 个整数部分的版本字符串)。如果 `semVerLevel=2.0.0` 提供了，则将返回 SemVer 1.0.0 和 SemVer 2.0.0 兼容包。有关详细信息，请参阅[SemVer 2.0.0 support for nuget.org](#)。

响应

响应是最多 `take` 包含自动完成结果的 JSON 文档。

根 JSON 对象具有以下属性：

NAME			
totalHits	integer	是	匹配项的总数， <code>skip</code> 忽略和 <code>take</code>
数据	字符串数组	是	请求匹配的包 Id

示例请求

```
GET https://api-v2v3search-0.nuget.org/autocomplete?q=storage&prerelease=true
```

示例响应

```
{
  "totalHits": 571,
  "data": [
    "WindowsAzure.Storage",
    "Storage.Net",
    "CK.Storage",
    "NCL.Storage",
    "DK.Storage",
    "Nine.Storage.Test",
    "Touch.Storage.Aws",
    "StorageAPIClient",
    "StorageAccess",
    "Storage.Net.Microsoft.Azure.Storage",
    "UnofficialAzure.StorageClient",
    "StorageAccess12",
    "AWSSDK.StorageGateway",
    "StorageExtensions",
    "Cloud.Storage",
    "lighthouse.storage",
    "ZU.Storage.Redis",
    "Magicodes.Storage",
    "Masticore.Storage",
    "hq.storage"
  ]
}
```

枚举包版本

使用以前的 API 发现包 ID 后, 客户端可以使用自动完成 API 来枚举提供的包 ID 的包版本。

未列出的包版本将不会出现在结果中。

```
GET {@id}?id={ID}&prerelease={PRERELEASE}&semVerLevel={SEMVERLEVEL}
```

请求参数

NAME	¶	¶	¶	¶
id	URL	string	是	要提取其版本的包 ID
早期	URL	boolean	否	<code>true</code> 或 <code>false</code> 确定是否包括 预发布包
semVerLevel	URL	string	否	SemVer 2.0.0 版本字符串

如果 `prerelease` 未提供, 则将排除预发布包。

`semVerLevel` Query 参数用于选择 SemVer 2.0.0 包。如果排除此查询参数, 则仅返回 SemVer 1.0.0 版本。如果 `semVerLevel=2.0.0` 提供了, 则将返回 SemVer 1.0.0 和 SemVer 2.0.0 版本。有关详细信息, 请参阅[SemVer 2.0.0 support for nuget.org](#)。

响应

响应是一个 JSON 文档, 其中包含所提供的包 ID 的所有包版本, 并按给定的查询参数进行筛选。

根 JSON 对象具有以下属性:

NAME	¶	¶	¶
数据	字符串数组	是	与请求匹配的包版本

如果 `1.0.0+metadata` `data` 查询字符串中提供了,则数组中的包版本可能包含SemVer2.0.0生成元数据(例如)。

`semVerLevel=2.0.0`

示例请求

```
GET https://api-v2v3search-0.nuget.org/autocomplete?id=nuget.protocol&prerelease=true
```

示例响应

```
{
  "data": [
    "4.3.0-preview3-4168",
    "4.3.0-preview4",
    "4.3.0-rtm-4324",
    "4.3.0",
    "4.4.0-preview3-4475",
    "4.4.0"
  ]
}
```

Catalog

2019/7/2 • [Edit Online](#)

目录是记录之类的创建和删除的包源上的所有包操作的资源。目录资源具有 Catalog 中键入 [服务索引](#)。可以使用此资源，使 [查询所有已发布的包](#)。

NOTE

由于该目录不由官方 NuGet 客户端，并非所有包源都实现目录。

NOTE

目前，nuget.org 目录不是在中国推出的。有关更多详细信息，请参阅[NuGet/NuGetGallery #4949](#)。

版本管理

以下 @type 使用值：

@TYPE	示例
Catalog/3.0.0	初始版本

基 URL

以下 API 的入口点 URL 是的值 @id 属性与前面提到的资源相关联 @type 值。本主题使用占位符 URL {@id}。

HTTP 方法

仅 HTTP 方法位于目录资源支持的所有 URL `GET` 和 `HEAD`。

目录索引

目录索引是具有一系列目录项，按时间顺序排列的已知位置中的文档。它是目录资源的入口点。

索引组成目录页。每个目录页包含目录项。每个目录项表示有关时间点的单个包的事件。目录项可以表示已创建、未列出、重新列出或删除从包源的包。通过处理中按时间顺序的目录项，客户端可以生成 V3 包源上存在的每个包的最新视图。

简单地说，目录的 blob 具有以下层次结构：

- 索引：目录的入口点。
- 页：目录项的分组。
- 叶：文档，表示一个目录项，它是单个包的状态的快照。

每个目录对象具有一个名为属性 `commitTimeStamp` 表示项添加到目录中时。目录项添加到名为提交的批次中的目录页。在同一提交中的所有目录项都具有相同提交时间戳 (`commitTimeStamp`) 和提交 ID (`commitId`)。目录项放置在同一提交中表示时间上的包源围绕同一点发生的事件。目录提交中没有排序。

因为每个包 ID 和版本是唯一的将永远不会有多个目录项的提交中。这可确保，单个包的目录项可始终明确地排序

方面提交时间戳。

没有永远不会为每个目录的多个提交 `commitTimeStamp`。换而言之，`commitId` 是冗余的 `commitTimeStamp`。

与此相反包元数据资源、包 ID 索引、目录是索引（和可查询）只能由时间。

目录项总是会添加到的目录中的单调递增的、按时间顺序。这意味着，如果在时间 X 添加目录提交则没有目录提交会不断添加时间小于或等于 X。

以下请求提取目录索引。

```
GET {@id}
```

目录索引是包含具有以下属性的对象的 JSON 文档：

属性	类型	是否必需	说明
<code>commitId</code>	string	是	与最新提交关联的唯一 ID
<code>commitTimeStamp</code>	string	是	最新的提交时间戳
<code>count</code>	整数	是	在索引中的页面数
项	对象的数组	是	对象，表示页的每个对象的数据组

在每个元素 `items` 数组是具有一些有关每个页面的最小的详细信息的对象。这些页面对象不包含目录叶（项）。未定义此数组中元素的顺序。可按内存使用中的客户端排序页及其 `commitTimeStamp` 属性。

由于新的页进行介绍，`count` 递增和新对象将显示在 `items` 数组。

将项目添加到目录，该索引的 `commitId` 将更改和 `commitTimeStamp` 会增加。这两个属性是通过所有页实质上是摘要 `commitId` 并 `commitTimeStamp` 中的值以 `items` 数组。

在索引中的目录页对象

目录索引中找到的目录页对象 `items` 属性具有以下属性：

属性	类型	是否必需	说明
<code>@id</code>	string	是	提取目录页面的 URL
<code>commitId</code>	string	是	使用此页中的最新提交关联的唯一 ID
<code>commitTimeStamp</code>	string	是	在此页中的最新提交的时间戳
<code>count</code>	整数	是	在目录页中的项的数目

与此相反包元数据资源在某些情况下，内嵌元素离开到索引中，目录叶是永远不会内联到索引，始终必须通过使用页面的提取 `@id` URL。

示例请求

```
GET https://api.nuget.org/v3/catalog0/index.json
```

示例响应

```
{  
  "commitId": "3d698852-eefb-48ed-8f55-9ee357540d20",  
  "commitTimeStamp": "2017-10-31T23:33:17.0954363Z",  
  "count": 3,  
  "items": [  
    {  
      "@id": "https://api.nuget.org/v3/catalog0/page0.json",  
      "commitId": "3a4df280-3d86-458e-a713-4c91ca261fef",  
      "commitTimeStamp": "2015-02-01T06:30:11.7477681Z",  
      "count": 540  
    },  
    {  
      "@id": "https://api.nuget.org/v3/catalog0/page1.json",  
      "commitId": "8bcd3cbf-74f0-47a2-a7ae-b7ecc50005d3",  
      "commitTimeStamp": "2015-02-01T06:39:53.9553899Z",  
      "count": 540  
    },  
    {  
      "@id": "https://api.nuget.org/v3/catalog0/page2.json",  
      "commitId": "3d698852-eefb-48ed-8f55-9ee357540d20",  
      "commitTimeStamp": "2017-10-31T23:33:17.0954363Z",  
      "count": 47  
    }  
  ]  
}
```

目录页

目录页是目录项的集合。这是使用其中一个提取的文档 `@id` 目录索引中找到的值。目录页面的 URL 不应为可预测，并应使用仅目录索引发现。

新目录项添加到目录索引只能与最高的提交时间戳中的页或到新页面。一旦具有更高版本的提交时间戳的页面添加到目录中，较早的页面永远不会添加或更改。

目录页文档是具有以下属性的 JSON 对象：

属性	类型	是否必需	说明
commitId	string	是	使用此页中的最新提交关联的唯一 ID
commitTimeStamp	string	是	在此页中的最新提交的时间戳
count	整数	是	在页中的项的数目
项	对象的数组	是	此页中的目录项
父级 (parent)	string	是	到目录索引 URL

在每个元素 `items` 数组是具有一些有关目录项的最小的详细信息的对象。这些项对象不包含的所有目录项的数据。在页面的项的顺序 `items` 数组未定义。可按内存使用中的客户端排序项及其 `commitTimeStamp` 属性。

在页面中的目录项的数目由服务器实现定义。对于 nuget.org，最多有 550 项在每个页中，尽管的实际数目的时间可能会更小，具体取决于在下一步提交批大小某些页面。

引入了新项，如 `count` 递增和新的目录项对象出现在 `items` 数组。

将项目添加到页上, `commitId` 更改和 `commitTimeStamp` 会增加。这两个属性对所有是实质上是摘要 `commitId` 并 `commitTimeStamp` 中的值以 `items` 数组。

目录项在页面中的对象

在目录页中找到的目录项对象 `items` 属性具有以下属性:

属性	类型	是否必需	说明
<code>@id</code>	string	是	要提取的目录项的 URL
<code>@type</code>	string	是	目录项的类型
<code>commitId</code>	string	是	与此目录项关联的提交 ID
<code>commitTimeStamp</code>	string	是	此目录项的提交时间戳
<code>nuget:id</code>	string	是	该叶与包 ID
<code>nuget:version</code>	string	是	该叶与包版本

`@type` 值将是以下两个值之一:

1. `nuget:PackageDetails` : 这对应于 `PackageDetails` 目录叶文档中的类型。
2. `nuget:PackageDelete` : 这对应于 `PackageDelete` 目录叶文档中的类型。

有关详细信息意味着每个类型, 请参阅[相对应的项类型](#)下面。

示例请求

```
GET https://api.nuget.org/v3/catalog0/page2926.json
```

示例响应

```
{
  "commitId": "616117f5-d9dd-4664-82b9-74d87169bbe9",
  "commitTimeStamp": "2017-10-31T23:30:32.4197849Z",
  "count": 5,
  "parent": "https://api.nuget.org/v3/catalog0/index.json",
  "items": [
    {
      "@id": "https://api.nuget.org/v3/catalog0/data/2017.10.31.23.30.32/util.biz.payments.0.0.4-preview.json",
      "@type": "nuget:PackageDetails",
      "commitId": "616117f5-d9dd-4664-82b9-74d87169bbe9",
      "commitTimeStamp": "2017-10-31T23:30:32.4197849Z",
      "nuget:id": "Util.Biz.Payments",
      "nuget:version": "0.0.4-preview"
    },
    {
      "@id": "https://api.nuget.org/v3/catalog0/data/2017.10.31.23.28.02/util.biz.0.0.4-preview.json",
      "@type": "nuget:PackageDetails",
      "commitId": "820340b2-97e3-4f93-b82e-bc85550a6560",
      "commitTimeStamp": "2017-10-31T23:28:02.788239Z",
      "nuget:id": "Util.Biz",
      "nuget:version": "0.0.4-preview"
    },
    {
      "@id": "https://api.nuget.org/v3/catalog0/data/2017.10.31.22.31.22/sourcecode.clay.data.1.0.0-preview1-00258.json",
      "@type": "nuget:PackageDetails",
      "commitId": "cae34527-ffc7-4e96-884f-7cf95a32dbdd",
      "commitTimeStamp": "2017-10-31T22:31:22.5169519Z",
      "nuget:id": "SourceCode.Clay.Data",
      "nuget:version": "1.0.0-preview1-00258"
    },
    {
      "@id": "https://api.nuget.org/v3/catalog0/data/2017.10.31.22.31.22/sourcecode.clay.1.0.0-preview1-00258.json",
      "@type": "nuget:PackageDetails",
      "commitId": "cae34527-ffc7-4e96-884f-7cf95a32dbdd",
      "commitTimeStamp": "2017-10-31T22:31:22.5169519Z",
      "nuget:id": "SourceCode.Clay",
      "nuget:version": "1.0.0-preview1-00258"
    },
    {
      "@id": "https://api.nuget.org/v3/catalog0/data/2017.10.31.22.31.22/sourcecode.clay.json.1.0.0-preview1-00258.json",
      "@type": "nuget:PackageDetails",
      "commitId": "cae34527-ffc7-4e96-884f-7cf95a32dbdd",
      "commitTimeStamp": "2017-10-31T22:31:22.5169519Z",
      "nuget:id": "SourceCode.Clay.Json",
      "nuget:version": "1.0.0-preview1-00258"
    }
  ]
}
```

目录叶

目录叶包含有关特定包 ID 和版本在某个时间点的元数据的时间。这是使用提取的文档 `@id` 目录页中找到的值。为目录叶的 URL 不应为可预测，并且应使用目录页发现。

目录叶文档是具有以下属性的 JSON 对象：

属性	类型	是否必需	说明
<code>@type</code>	字符串或字符串数组	是	目录项的类型

属性	类型	是否	说明
catalog:commitId	string	是	一个与此目录项关联的提交 ID
catalog:commitTimeStamp	string	是	此目录项的提交时间戳
id	string	是	目录项的包 ID
已发布	string	是	包的目录项的发布的日期
version	string	是	目录项的包版本

项类型

@type 属性为字符串数组。为方便起见，如果 @type 值是一个字符串，它应视为一个大小的任何数组。并非所有可能值 @type 记录。但是，每个目录项有两个以下字符串类型值之一：

1. PackageDetails : 表示包元数据的快照
2. PackageDelete : 表示已删除的包

包的详细信息的目录项

目录项类型的 PackageDetails 包含特定包 (ID 和版本组合) 的包元数据的快照。如果包源遇到的任何以下情况下，则会生成包目录项的详细信息：

1. 包是推送。
2. 包是列出。
3. 包是未列出。
4. 包是重排。

包重排是实质上是生成包本身的虚设推送，无需更改现有包的管理手势。在 nuget.org 中，使用目录的后台作业之一修复 bug 后使用重排。

客户端使用的目录项不应尝试确定哪种方案生成的目录项目。相反，客户端应只需更新任何维护的视图或索引使用的目录项目中包含的元数据。此外，应妥善处理重复或冗余目录项 (幂等方式)。

包详细信息的目录项具有以下属性除了 [包含在所有目录叶](#)。

属性	类型	是否	说明
作者	string	否	
created	string	否	首次创建包的时间戳。回退属性： published 。
dependencyGroups	对象的数组	否	按目标框架的包的依赖项分组 (与包元数据资源相同的格式)
不推荐使用	object	否	不推荐使用与包相关联 (与包元数据资源相同的格式)
说明	string	否	
iconUrl	string	否	

isPrerelease	boolean	否	是否是预发行包版本。可以从检测到 <code>version</code> 。
语言	string	否	
licenseUrl	string	否	
列出	boolean	否	该程序包是否将列
minClientVersion	string	否	
packageHash	string	是	使用编码的包的哈希 标准 base64
packageHashAlgorithm	string	是	
packageSize	整数	是	包.nupkg 以字节为单位的大 小
projectUrl	string	否	
releaseNotes	string	否	
requireLicenseAgreement	boolean	否	假定 <code>false</code> 如果排除
摘要	string	否	
标记	字符串数组	否	
标题	string	否	
verbatimVersion	string	否	因为它的版本字符串最初位 于.nuspec

包 `version` 属性是在执行规范化后的完整版本字符串。这意味着，SemVer 2.0.0 生成数据可以包含此处。

`created` 包已首次接收到包源，这通常是短时间之前目录项的提交时间戳时时间戳。

`packageHashAlgorithm` 是由服务器实现表示哈希算法用于生成定义的字符串 `packageHash`。始终使用 nuget.org `packageHashAlgorithm` 的值 `SHA512`。

`published` 时间戳是上次列出包的时间。

NOTE

在 nuget.org 中，`published` 值设置为 1900 年时取消列出包。

示例请求

获取 <https://api.nuget.org/v3/catalog0/data/2015.02.01.11.18.40/windowsazure.storage.1.0.0.json>

示例响应

```
{
```

```

"@type": [
    "PackageDetails",
    "catalog:Permalink"
],
"authors": "NuGet.org Team",
"catalog:commitId": "49fe04d8-5694-45a5-9822-3be61bda871b",
"catalog:commitTimeStamp": "2015-02-01T11:18:40.8589193Z",
"created": "2011-12-02T20:21:23.74Z",
"description": "This package is an example for the V3 protocol.",
"deprecation": {
    "reasons": [
        "Legacy",
        "HasCriticalBugs",
        "Other"
    ],
    "message": "This package is an example--it should not be used!",
    "alternatePackage": {
        "id": "Newtonsoft.JSON",
        "range": "12.0.2"
    }
},
"iconUrl": "https://www.nuget.org/Content/gallery/img/default-package-icon.svg",
"id": "NuGet.Protocol.V3.Example",
"isPrerelease": false,
"language": "en-US",
"licenseUrl": "http://www.opensource.org/licenses/ms-pl",
"packageHash": "2edCwKLcbcFJpsAwa883BLt0y8bZpWwbQpiIb71E74k5t2f2WzXEGWbPwntrleUEgSrcxJrh90rm/TAmg04NQ==",
"packageHashAlgorithm": "SHA512",
"packageSize": 118348,
"projectUrl": "https://github.com/NuGet/NuGetGallery",
"published": "1900-01-01T00:00:00Z",
"requireLicenseAcceptance": false,
"title": "<span data-ttu-id=\"63192-101\">NuGet V3 协议示例</span><span class=\"sxs-lookup\"><span data-stu-id=\"63192-101\">NuGet V3 Protocol Example</span></span>",
"version": "1.0.0",
"dependencyGroups": [
{
    "@id": "https://api.nuget.org/v3/catalog0/data/2015.02.01.11.18.40/windowsazure.storage.1.0.0.json#dependencygroup",
    "@type": "PackageDependencyGroup",
    "dependencies": [
        {
            "@id": "https://api.nuget.org/v3/catalog0/data/2015.02.01.11.18.40/windowsazure.storage.1.0.0.json#dependencygroup/aspNet.suppressformsredirect",
            "@type": "PackageDependency",
            "id": "aspnet.suppressformsredirect",
            "range": "[0.0.1.4, )"
        },
        {
            "@id": "https://api.nuget.org/v3/catalog0/data/2015.02.01.11.18.40/windowsazure.storage.1.0.0.json#dependencygroup/webactivator",
            "@type": "PackageDependency",
            "id": "WebActivator",
            "range": "[1.4.4, )"
        },
        {
            "@id": "https://api.nuget.org/v3/catalog0/data/2015.02.01.11.18.40/windowsazure.storage.1.0.0.json#dependencygroup/webapi.all",
            "@type": "PackageDependency",
            "id": "WebApi.All",
            "range": "[0.5.0, )"
        }
    ],
    "targetFramework": ".NETFramework4.6"
}
]

```

```
    "tags": [
      "NuGet",
      "V3",
      "Protocol",
      "Example"
    ]
}
```

包删除目录项

目录项类型的 `PackageDelete` 包含少量的包从包源已被删除并且不再可用于任何包的操作（如还原），指示向目录客户端的信息。

NOTE

很可能要删除的包和更高版本重新发布使用的相同包 ID 和版本。在 nuget.org 中，这是极少数情况下，因为这会破坏的包 ID 和版本表示特定包内容的官方客户端的假设。在 nuget.org 上包删除的详细信息，请参阅[我们的策略](#)。

包删除目录项不包含其他属性除了[包含在所有目录叶](#)。

`version` 属性是在包.nuspec 中找到的原始版本字符串。

`published` 属性是包被删除时，这通常是作为目录项的提交时间截之前的时间。

示例请求

获取 https://api.nuget.org/v3/catalog0/data/2017.11.02.00.40.00/netstandard1.4_lib.1.0.0-test.json

示例响应

```
{
  "@type": [
    "PackageDelete",
    "catalog:Permalink"
  ],
  "catalog:commitId": "19fec5b4-9335-4e4b-bd50-8d5d3f734597",
  "catalog:commitTimeStamp": "2017-11-02T00:40:00.1969812Z",
  "id": "netstandard1.4_lib",
  "originalId": "netstandard1.4_lib",
  "published": "2017-11-02T00:37:43.7181952Z",
  "version": "1.0.0-test"
}
```

Cursor

概述

本部分介绍的客户端概念，尽管不一定是托管协议，但应为任何实际目录客户端实现的一部分。

因为目录是时间索引的仅限追加的数据结构，应存储在客户端游标本地，最多的时间点在客户端表示已处理目录项。请注意，应永远不会使用客户端的计算机时钟生成此游标值。相反，值应来自于目录对象的 `commitTimestamp` 值。

每次客户端想要处理的包源上的新事件，它需要仅查询目录的提交时间截的所有目录项大于其存储的游标。客户端已成功处理所有新的目录项后，它会记录作为它的新游标值只是处理目录项的最新的提交时间截。

使用此方法，而客户端可以确保永远不会错过包源发生的任何包事件。此外，客户端绝不会重新处理在游标的记录的提交时间截之前的旧事件。

这一功能强大的游标概念用于许多 nuget.org 后台作业，用于使 V3 API 本身保持最新。

初始值

当目录客户端第一次启动（并因此具有任何游标值）时，它应使用默认游标的值。NET 的 `System.DateTimeOffset.MinValue` 或最小可表示的时间戳的一些此类类似概念。

循环访问目录项

若要查询下一组要处理的目录项，客户端应：

1. 从本地存储区中提取记录的游标值。
2. 下载并反序列化目录索引。
3. 查找所有目录页提交时间戳大于光标。
4. 声明目录项，以便处理空的列表。
5. 为在步骤 3 中匹配每个目录页：
 - a. 下载并反序列化目录页。
 - b. 查找所有目录项提交时间戳大于光标。
 - c. 将所有匹配的目录项添加到在步骤 4 中声明的列表。
6. 对目录项列表按提交时间戳进行排序。
7. 按顺序处理每个目录项：
 - a. 下载并反序列化的目录项目。
 - b. 相应地作出反应到目录项的类型。
 - c. 处理目录项文档以特定于客户端的方式。
8. 记录为的新游标值的最后一个目录项的提交时间戳。

使用此基本算法，客户端实现可以构建包源上可用的所有包的完整视图。客户端仅需要执行此算法会定期以始终要清楚的包源的最新更改。

NOTE

这是算法的 nuget.org 用来保持[包元数据](#), [包内容](#), [搜索](#)并[记忆式键入功能](#)最新的资源。

依赖的游标

假设有两个目录客户端具有固有的依赖关系，其中一个客户端的输出取决于另一个客户端的输出。

示例

例如，在 nuget.org 上的新发布的包不应出现在搜索资源才会出现在包元数据资源。这是因为“还原”操作执行的官方 NuGet 客户端使用的包元数据资源。如果客户将发现使用搜索服务包，他们应能够成功还原该包使用的包元数据资源。换而言之，搜索资源依赖于包元数据资源。每个资源都有目录客户端后台作业，更新该资源。每个客户端具有其自己的光标。

由于这两个资源是从目录更新搜索资源的目录客户端游标必须不超过包元数据目录客户端游标。

算法

若要实现这一限制，只需修改上面为算法：

1. 从本地存储区中提取记录的游标值。
2. 下载并反序列化目录索引。
3. 查找所有目录页提交时间戳大于光标小于或等于依赖项的游标。
4. 声明目录项，以便处理空的列表。
5. 为在步骤 3 中匹配每个目录页：
 - a. 下载并反序列化目录页。
 - b. 查找所有目录项提交时间戳大于光标小于或等于依赖项的游标。
 - c. 将所有匹配的目录项添加到在步骤 4 中声明的列表。
6. 对目录项列表按提交时间戳进行排序。
7. 按顺序处理每个目录项：

- a. 下载并反序列化的目录项目。
 - b. 相应地作出反应到目录项的类型。
 - c. 处理目录项文档以特定于客户端的方式。
8. 记录为的新游标值的最后一个目录项的提交时间戳。

使用此修改后的算法，您可以构建系统的依赖目录客户端的所有生成其自己的特定索引、项目等。

包内容

2019/9/19 • [Edit Online](#)

可以使用 V3 API 生成 URL 来提取任意包的内容(nupkg 文件)。用于提取包内容的资源是在 [PackageBaseAddress 服务索引](#) 中找到的资源。此资源还允许发现包的所有版本、列出或未列出。

此资源通常称为 "包基址" 或 "平面容器"。

版本管理

使用以下 `@type` 值：

<code>@TYPE</code>	
PackageBaseAddress/3.0.0	初始版本

基 URL

以下 api 的基 URL 是与上述资源 `@id` `@type` 值关联的属性的值。在下面的文档中，将使用占位符 `{@id}` 基 URL。

HTTP 方法

注册资源中找到的所有 url 都支持 HTTP 方法 `GET` 和 `HEAD`。

枚举包版本

如果客户端知道包 ID 并想要发现包源有哪些包版本可用，则客户端可以构造可预测的 URL 来枚举所有包版本。此列表应为下述包内容 API 的 "目录列表"。

NOTE

此列表包含列出和未列出的包版本。

```
GET {@id}/{LOWER_ID}/index.json
```

请求参数

NAME				
LOWER_ID	URL	string	是	包 ID lowercased

`LOWER_ID` 值是所需的包 ID lowercased，它使用由实现的规则。NET 的 [System.String.ToLowerInvariant\(\)](#) 方法。

响应

如果包源没有提供的包 ID 版本，则返回 404 状态代码。

如果包源有一个或多个版本，则返回 200 状态代码。响应正文是包含以下属性的 JSON 对象：

NAME				
版本	字符串数组	是	可用版本	

`versions` 数组中的字符串是所有 lowercased 的规范化 NuGet 版本字符串。版本字符串不包含任何 SemVer 2.0.0 生成元数据。

目的在于，在此数组中找到的版本字符串可以逐字 `LOWER_VERSION` 用于以下终结点中的标记。

示例请求

```
GET https://api.nuget.org/v3-flatcontainer/owin/index.json
```

示例响应

```
{
  "versions": [
    "0.5.0",
    "0.7.0",
    "0.11.0",
    "0.12.0",
    "0.14.0",
    "1.0.0"
  ]
}
```

下载包内容 (.nupkg)

如果客户端知道包 ID 和版本并想要下载包内容，则它们只需构造以下 URL：

```
GET {@id}/{LOWER_ID}/{LOWER_VERSION}/{LOWER_ID}.{LOWER_VERSION}.nupkg
```

请求参数

NAME				
LOWER_ID	URL	string	是	包 ID, 小写
LOWER_VERSION	URL	string	是	包版本(正常化和 lowercased)

`LOWER_ID` 和 `LOWER_VERSION` 都是使用实现的规则 lowercased 的。网络 [System.String.ToLowerInvariant\(\)](#) 方法。

是使用 NuGet 的版本规范化规则规范化的所需包版本。`LOWER_VERSION` 这意味着，在这种情况下必须排除 SemVer 2.0.0 规范允许的生成元数据。

响应正文

如果包源上存在包，则返回 200 状态代码。响应正文将是包内容本身。

如果包源中不存在包，则返回 404 状态代码。

示例请求

```
GET https://api.nuget.org/v3-flatcontainer/newtonsoft.json/9.0.1/newtonsoft.json.9.0.1.nupkg
```

示例响应

Newtonsoft.json 的 nupkg 的二进制流。

下载包清单 (.nuspec)

如果客户端知道包 ID 和版本并想要下载包清单，则它们只需构造以下 URL：

```
GET {@id}/{LOWER_ID}/{LOWER_VERSION}/{LOWER_ID}.nuspec
```

请求参数

NAME	TYPE	DESCRIPTION	REQUIRED	EXAMPLE
LOWER_ID	URL	string	是	包 ID, 小写
LOWER_VERSION	URL	string	是	包版本(正常化和 lowercased)

`LOWER_ID` 和 `LOWER_VERSION` 都是使用实现的规则 lowercased 的。NET 的 [System.String.ToLowerInvariant\(\)](#) 方法。

是使用 NuGet 的版本[规范化规则](#)规范化的所需包版本。`LOWER_VERSION` 这意味着，在这种情况下必须排除 SemVer 2.0.0 规范允许的生成元数据。

响应正文

如果包源上存在包，则返回 200 状态代码。响应正文将为包清单，即 nuspec 中包含的 nupkg。Nuspec 是一个 XML 文档。

如果包源中不存在包，则返回 404 状态代码。

示例请求

```
GET https://api.nuget.org/v3-flatcontainer/newtonsoft.json/6.0.4/newtonsoft.json.nuspec
```

示例响应

```
<?xml version="1.0"?>
<package xmlns="http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd">
  <metadata>
    <id>Newtonsoft.Json</id>
    <version>6.0.4</version>
    <title>Json.NET</title>
    <authors>James Newton-King</authors>
    <owners>James Newton-King</owners>
    <licenseUrl>https://raw.github.com/JamesNK/Newtonsoft.Json/master/LICENSE.md</licenseUrl>
    <projectUrl>http://james.newtonking.com/json</projectUrl>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>Json.NET is a popular high-performance JSON framework for .NET</description>
    <language>en-US</language>
    <tags>json</tags>
  </metadata>
</package>
```

包详细信息 URL 模板

2020/1/31 • [Edit Online](#)

客户端可能会生成一个 URL，用户可以使用该 URL 在其 web 浏览器中查看更多包详细信息。当包源要显示与 NuGet 客户端应用程序所显示的范围不匹配的包的其他信息时，这非常有用。

用于生成此 URL 的资源是在服务索引中找到的 `PackageDetailsUriTemplate` 资源。

版本管理

使用以下 `@type` 值：

<code>@TYPE</code>	示例
<code>PackageDetailsUriTemplate/5.1.0</code>	初始版本

URL 模板

以下 API 的 URL 是 `@id` 属性的值，该属性与前面提到的一个资源 `@type` 值相关联。

HTTP 方法

尽管客户端不打算代表用户向包详细信息 URL 发出请求，但网页应支持 `GET` 方法，以便在 web 浏览器中轻松地打开已单击的 URL。

构造 URL

给定已知的包 ID 和版本，客户端实现可以构造用于访问 web 界面的 URL。客户端实现应向用户显示此构造的 URL（或可单击的链接），以允许用户通过 web 浏览器打开 URL 并了解有关包的详细信息。包详细信息页的内容由服务器实现确定。

URL 必须是绝对 URL，方案（协议）必须为 HTTPS。

服务索引中 `@id` 的值是包含以下任何占位符标记的 URL 字符串：

URL 占位符

NAME	示例	示例	示例
<code>{id}</code>	<code>string</code>	<code>no</code>	要获取其详细信息的包 ID
<code>{version}</code>	<code>string</code>	<code>no</code>	要获取其详细信息的包版本

服务器应接受任意大小写形式的 `{id}` 和 `{version}` 值。此外，服务器不应对版本是否规范化敏感。换句话说，服务器还应接受非标准化版本。

例如，nuget 的包详细信息模板如下所示：

```
https://www.nuget.org/packages/{id}/{version}
```

如果客户端实现需要显示4.3.0 的包详细信息的链接，它将生成以下 URL，并将其提供给用户：

<https://www.nuget.org/packages/NuGet.Versioning/4.3.0>

包元数据

2020/3/19 • [Edit Online](#)

可以使用 NuGet V3 API 提取有关包源上可用包的元数据。可以使用在[服务索引](#)中找到的 `RegistrationsBaseUrl` 资源来提取此元数据。

在 `RegistrationsBaseUrl` 下找到的文档的集合通常称为 "注册" 或 "注册 blob"。单个 `RegistrationsBaseUrl` 下的文档集称为 "注册配置单元"。注册配置单元包含有关包源上可用的每个包的所有元数据。

版本控制

使用以下 `@type` 值：

<code>@TYPE</code>	
<code>RegistrationsBaseUrl</code>	初始版本
<code>RegistrationsBaseUrl/3.0.0-beta</code>	<code>RegistrationsBaseUrl</code> 的别名
<code>RegistrationsBaseUrl/3.0.0-rc</code>	<code>RegistrationsBaseUrl</code> 的别名
<code>RegistrationsBaseUrl/3.4.0</code>	Gzip 压缩过响应
<code>RegistrationsBaseUrl/3.6.0</code>	包括 SemVer 2.0.0 包

这表示可用于各种客户端版本的三个不同的注册配置单元。

`RegistrationsBaseUrl`

这些注册不会压缩(也就是说, 它们使用隐含的 `Content-Encoding: identity`)。此配置单元中排除了 SemVer 2.0.0 包。

`RegistrationsBaseUrl/3.4.0`

这些注册使用 `Content-Encoding: gzip` 进行压缩。此配置单元中排除了 SemVer 2.0.0 包。

`RegistrationsBaseUrl/3.6.0`

这些注册使用 `Content-Encoding: gzip` 进行压缩。此配置单元中包含 SemVer 2.0.0 包。有关 SemVer 2.0.0 的详细信息, 请参阅[nuget.org 的 SemVer 2.0.0 支持](#)。

基 URL

以下 Api 的基 URL 是与前面提到的资源 `@type` 值相关联的 `@id` 属性的值。在下面的文档中, 将使用占位符基 URL `{@id}`。

HTTP 方法

注册资源中找到的所有 Url 都支持 `GET` 和 `HEAD` 的 HTTP 方法。

注册索引

注册资源组按包 ID 包元数据。一次不能获取有关多个包 ID 的数据。此资源不提供任何方式来发现包 Id。相反,

假设客户端已经知道所需的包 ID。每个包版本的可用元数据因服务器实现而异。包注册 blob 具有以下层次结构：

- **索引**: 包元数据的入口点, 由具有相同包 ID 的源中的所有包共享。
- **页面**: 包版本的分组。页面中的包版本数由服务器实现定义。
- **叶**: 特定于单个包版本的文档。

注册索引的 URL 可预测, 并且可由客户端从服务索引中给定包 ID 和注册资源的 `@id` 值确定。通过检查注册索引发现注册页和叶的 Url。

注册页面和叶

尽管服务器实现将注册 leafs 存储在单独的注册页文档中并不是绝对必需的, 但建议使用这种方法来节省客户端内存。建议服务器实现定义一些试探法, 以根据包版本的数量在这两种方法之间进行选择, 而不是将所有注册留在索引中, 也不会立即存储在页面文档中。包叶的累计大小。

将所有包版本(叶)存储在注册索引中可节省提取包元数据所需的 HTTP 请求数, 但这意味着必须下载更大的文档, 并且必须分配更多的客户端内存。另一方面, 如果服务器实现立即将注册留在单独的页面文档中, 则客户端必须执行更多 HTTP 请求以获取所需的信息。

Nuget.org 使用的试探法如下: 如果包有128或更多版本, 请将叶分成大小为64的页。如果版本低于128, 则将所有行都置于注册索引中。请注意, 这意味着65到127版本的包在索引中将有两页, 但这两个页将被内联。

```
GET {@id}/{LOWER_ID}/index.json
```

请求参数

名称	类型	描述	是否必需	说明
LOWER_ID	代码	字符串	是	包 ID lowercased

`LOWER_ID` 值是所需的包 ID lowercased, 它使用由实现的规则。NET 的 [System.String.ToLowerInvariant\(\)](#) 方法。

响应

响应是一个 JSON 文档, 其中包含具有以下属性的根对象:

名称	类型	描述	说明
count	integer	是	索引中注册页的数目
items	对象数组	是	注册页的数组

Index 对象的 `items` 数组中的每一项都是一个表示注册页的 JSON 对象。

注册页对象

在注册索引中找到的注册页对象具有以下属性:

名称	类型	描述	说明
@id	字符串	是	注册页的 URL
count	integer	是	页面中的注册离开次数
items	对象数组	否	注册叶数组及其关联元数据

lower	字符串	是	页面中的最低 SemVer 2.0.0 版本(包含)
父级 (parent)	字符串	否	注册索引的 URL
upper	字符串	是	页面中的 SemVer 2.0.0 的最高版本(包含)

当需要特定页面版本的元数据时, `page` 对象的 `lower` 和 `upper` 界限非常有用。这些界限可用于提取所需的唯一注册页面。版本字符串遵循[NuGet 的版本规则](#)。版本字符串已规范化, 不包括生成元数据。与 NuGet 生态系统中的所有版本一样, 版本字符串的比较是使用[SemVer 2.0.0 的版本优先规则](#)实现的。

仅当注册页对象具有 `items` 属性时才会显示 "parent" 属性。

如果注册页对象中不存在 `items` 属性, 则必须使用 `@id` 中指定的 URL 来获取有关各个包版本的元数据。有时 `items` 数组作为优化从页面对象中排除。如果单个包 ID 的版本数非常大, 则对于只关心特定版本或少量版本的客户端, 注册索引文档将会很大, 并且会浪费大量的信息。

请注意, 如果存在 `items` 属性, 则不需要使用 `@id` 属性, 因为所有页面数据都已在 `items` 属性中内联。

Page 对象的 `items` 数组中的每一项都是一个表示注册叶及其关联元数据的 JSON 对象。

页中的注册叶对象

在注册页中找到的注册叶对象具有以下属性:

@id	字符串	是	注册叶的 URL
catalogEntry	对象 (object)	是	包含包元数据的目录条目
packageContent	字符串	是	包内容的 URL (.nupkg)

每个注册叶对象均表示与单个包版本关联的数据。

目录条目

注册叶对象中的 `catalogEntry` 属性具有以下属性:

@id	字符串	是	用于生成此对象的文档的 URL
authors	字符串或字符串数组	否	
dependencyGroups	对象数组	否	包的依赖项, 按目标框架分组
弃用	对象 (object)	否	与包关联的弃用
description	字符串	否	
iconUrl	字符串	否	

属性	类型	必需	说明
id	字符串	是	包的 ID
licenseUrl	字符串	否	
licenseExpression	字符串	否	
列出的	boolean	否	如果不存在，则应将其视为列出
minClientVersion	字符串	否	
projectUrl	字符串	否	
published	字符串	否	一个字符串，其中包含发布包时的 ISO 8601 时间戳
requireLicenseAcceptance	boolean	否	
摘要	字符串	否	
标记	字符串或字符串数组	否	
title	字符串	否	
版本	字符串	是	规范化后的完整版本字符串

Package `version` 属性是规范化后的完整版本字符串。这意味着，可以在此处包括 SemVer 2.0.0 生成数据。

`dependencyGroups` 属性是对象的数组，这些对象表示包的依赖项，按目标框架分组。如果包没有依赖项，则缺少 `dependencyGroups` 属性、空数组或所有组的 `dependencies` 属性为空或缺失。

`licenseExpression` 属性的值符合[NuGet 许可证表达式语法](#)。

NOTE

在 nuget.org 上，当包未列出时，`published` 值设置为1900年。

包依赖关系组

每个依赖项组对象具有以下属性：

属性	类型	必需	说明
<code>targetFramework</code>	字符串	否	这些依赖关系适用的目标框架
依赖关系	对象数组	否	

`targetFramework` 字符串使用 NuGet 的 .NET 库[nuget](#)所实现的格式。如果未指定 `targetFramework`，则依赖项组适用于所有目标框架。

`dependencies` 属性是对象的数组，每个对象表示当前包的包依赖项。

包依赖关系

每个包依赖项都具有以下属性：

属性	类型	是否必需	说明
id	字符串	是	包依赖项的 ID
range	对象 (object)	否	依赖项的允许 版本范围
注册	字符串	否	此依赖项的注册索引的 URL

如果已排除 range 属性或空字符串，则客户端应默认为版本范围 (,)。也就是说，允许使用任何版本的依赖项。range 属性不允许 * 的值。

包弃用

每个包弃用都具有以下属性：

属性	类型	是否必需	说明
原因	字符串数组	是	弃用包的原因
message	字符串	否	有关此弃用的其他详细信息
alternatePackage	对象 (object)	否	应改为使用的备用包

reasons 属性必须包含至少一个字符串，并且只应包含下表中的字符串：

原因	说明
旧的	不再维护包
CriticalBugs	包中的 bug 使其不适用于使用
其他	由于未在此列表中的原因，包已弃用

如果 reasons 属性包含不是来自已知集的字符串，则应将其忽略。字符串不区分大小写，因此应将 legacy 视为与 Legacy 相同。数组没有排序限制，因此字符串可以任意顺序排列。此外，如果属性仅包含不来自已知集的字符串，则应将其视为只包含 "其他" 字符串。

备用包

备用包对象具有以下属性：

属性	类型	是否必需	说明
id	字符串	是	备用包的 ID
range	对象 (object)	否	允许的 版本范围 ，或在允许任何版本的情况下 *

示例请求

```
GET https://api.nuget.org/v3/registration3/nuget.server.core/index.json
```

示例响应

```
{
  "count": 1,
  "items": [
    {
      "@id": "https://api.nuget.org/v3/registration3/nuget.server.core/index.json#page/3.0.0-beta/3.0.0-
beta",
      "count": 1,
      "items": [
        {
          "@id": "https://api.nuget.org/v3/registration3/nuget.server.core/3.0.0-beta.json",
          "catalogEntry": {
            "@id": "https://api.nuget.org/v3/catalog0/data/2017.10.05.18.41.33/nuget.server.core.3.0.0-
beta.json",
            "authors": ".NET Foundation",
            "dependencyGroups": [
              {
                "@id": "https://api.nuget.org/v3/catalog0/data/2017.10.05.18.41.33/nuget.server.core.3.0.0-
beta.json#dependencygroup",
                "dependencies": [
                  {
                    "@id":
"https://api.nuget.org/v3/catalog0/data/2017.10.05.18.41.33/nuget.server.core.3.0.0-
beta.json#dependencygroup/nuget.core",
                    "id": "NuGet.Core",
                    "range": "[2.14.0, )",
                    "registration": "https://api.nuget.org/v3/registration3/nuget.core/index.json"
                  }
                ]
              }
            ],
            "description": "Core library for creating a Web Application used to host a simple NuGet feed",
            "iconUrl": "",
            "id": "NuGet.Server.Core",
            "language": "",
            "licenseUrl": "https://raw.githubusercontent.com/NuGet/NuGet.Server/dev/LICENSE.txt",
            "listed": true,
            "minClientVersion": "2.6",
            "packageContent": "https://api.nuget.org/v3-flatcontainer/nuget.server.core/3.0.0-
beta/nuget.server.core.3.0.0-beta.nupkg",
            "projectUrl": "https://github.com/NuGet/NuGet.Server",
            "published": "2017-10-05T18:40:32.43+00:00",
            "requireLicenseAcceptance": false,
            "summary": "",
            "tags": [ "" ],
            "title": "",
            "version": "3.0.0-beta"
          },
          "packageContent": "https://api.nuget.org/v3-flatcontainer/nuget.server.core/3.0.0-
beta/nuget.server.core.3.0.0-beta.nupkg",
          "registration": "https://api.nuget.org/v3/registration3/nuget.server.core/index.json"
        }
      ],
      "lower": "3.0.0-beta",
      "upper": "3.0.0-beta"
    }
  ]
}
```

在这种特定情况下，注册索引会内嵌注册页面，因此，提取有关各个包版本的元数据不需要额外的请求。

注册页

注册页面包含注册离开。用于提取注册页的 URL 由前面提到的[注册页对象](#)中的 `@id` 属性确定。此 URL 并不是可预测的，应始终通过索引文档来发现。

WARNING

在 nuget.org 上，注册页文档恰巧的 URL 包含该页的下限和上限。不过，客户端永远不应执行此假设，因为只要索引文档具有有效的链接，服务器实现就可以随意更改 URL 的形状。

如果注册索引中未提供 `items` 数组，则 `@id` 值的 HTTP GET 请求将返回一个 JSON 文档，该文档具有作为其根的对象。对象具有以下属性：

属性	类型	是否	说明
<code>@id</code>	字符串	是	注册页的 URL
<code>count</code>	integer	是	页面中的注册离开次数
<code>items</code>	对象数组	是	注册叶数组及其关联元数据
<code>lower</code>	字符串	是	页面中的最低 SemVer 2.0.0 版本(包含)
父级 (parent)	字符串	是	注册索引的 URL
<code>upper</code>	字符串	是	页面中的 SemVer 2.0.0 的最高版本(包含)

注册叶对象的形状与 [上面的注册索引](#) 相同。

示例请求

```
GET https://api.nuget.org/v3/registration3/ravendb.client/page/1.0.531/1.0.729-unstable.json
```

示例响应

```
{
  "count": 2,
  "lower": "1.0.531",
  "parent": "https://api.nuget.org/v3/registration3/nuget.protocol.v3.example/index.json",
  "upper": "1.0.729-unstable",
  "items": [
    {
      "@id": "https://api.nuget.org/v3/registration3/nuget.protocol.v3.example/1.0.531.json",
      "@type": "Package",
      "commitId": "e0b9ca79-75b5-414f-9e3e-de9534b5cf1",
      "commitTimeStamp": "2017-10-26T14:12:19.3439088Z",
      "catalogEntry": {
        "@id": "https://api.nuget.org/v3/catalog0/data/2015.02.01.11.38.37/nuget.protocol.v3.example.1.0.531.json",
        "@type": "PackageDetails",
        "authors": "NuGet.org Team",
        "iconUrl": "https://www.nuget.org/Content/gallery/img/default-package-icon.svg",
        "id": "NuGet.Protocol.V3.Example",
        "licenseUrl": "http://www.opensource.org/licenses/ms-pl",
        "listed": false,
        "packageContent": "https://api.nuget.org/v3-flatcontainer/nuget.protocol.v3.example/1.0.531/nuget.protocol.v3.example.1.0.531.nupkg",
        "projectUrl": "https://github.com/NuGet/NuGetGallery",
        "published": "1900-01-01T00:00:00+00:00",
        "requireLicenseAcceptance": true,
      }
    }
  ]
}
```

```
        "title": "NuGet V3 协议示例",
        "version": "1.0.531"
    },
    "packageContent": "https://api.nuget.org/v3-flatcontainer/nuget.protocol.v3.example/1.0.531/nuget.protocol.v3.example.1.0.531.nupkg",
    "registration": "https://api.nuget.org/v3/registration3/nuget.protocol.v3.example/index.json"
},
{
    "@id": "https://api.nuget.org/v3/registration3/nuget.protocol.v3.example/1.0.729-unstable.json",
    "@type": "Package",
    "commitId": "e0b9ca79-75b5-414f-9e3e-de9534b5cf1",
    "commitTimeStamp": "2017-10-26T14:12:19.3439088Z",
    "catalogEntry": {
        "@id":
"https://api.nuget.org/v3/catalog0/data/2015.02.01.18.22.05/nuget.protocol.v3.example.1.0.729-unstable.json",
        "@type": "PackageDetails",
        "authors": "NuGet.org Team",
        "deprecation": {
            "reasons": [
                "CriticalBugs"
            ],
            "message": "This package is unstable and broken!",
            "alternatePackage": {
                "id": "Newtonsoft.JSON",
                "range": "12.0.2"
            }
        },
        "iconUrl": "https://www.nuget.org/Content/gallery/img/default-package-icon.svg",
        "id": "NuGet.Protocol.V3.Example",
        "licenseUrl": "http://www.opensource.org/licenses/ms-pl",
        "listed": false,
        "packageContent": "https://api.nuget.org/v3-flatcontainer/nuget.protocol.v3.example/1.0.729-unstable/nuget.protocol.v3.example.1.0.729-unstable.nupkg",
        "projectUrl": "https://github.com/NuGet/NuGetGallery",
        "published": "1900-01-01T00:00:00+00:00",
        "requireLicenseAcceptance": true,
        "summary": "This package is an example for the V3 protocol.",
        "title": "NuGet V3 协议示例",
        "version": "1.0.729-Unstable"
    },
    "packageContent": "https://api.nuget.org/v3-flatcontainer/nuget.protocol.v3.example/1.0.729-unstable/nuget.protocol.v3.example.1.0.729-unstable.nupkg",
    "registration": "https://api.nuget.org/v3/registration3/nuget.protocol.v3.example/index.json"
}
]
```

注册叶

注册叶包含有关特定包 ID 和版本的信息。有关特定版本的元数据在本文档中可能不可用。应该从[注册索引](#)或[注册页](#)(使用注册索引发现)提取包元数据。

提取注册叶的 URL 是从注册索引或注册页中注册叶对象的 `@id` 属性获取的。与页面文档一样。此 URL 并不是可预测的，应始终通过[注册页](#)对象来发现。

WARNING

在 nuget.org 上，注册叶文档恰巧的 URL 包含包版本。不过，客户端永远不应执行此假设，因为只要父文档具有有效的链接，服务器实现就可以随意更改 URL 的形状。

注册叶是包含具有以下属性的根对象的 JSON 文档：

@id	字符串	是	注册叶的 URL
catalogEntry	字符串	否	生成这些叶的目录条目的 URL
列出的	boolean	否	如果不存在，则应将其视为列出
packageContent	字符串	否	包内容的 URL (.nupkg)
published	字符串	否	一个字符串，其中包含发布包时的 ISO 8601 时间戳
注册	字符串	否	注册索引的 URL

NOTE

在 nuget.org 上，当包未列出时，`published` 值设置为1900年。

示例请求

```
GET https://api.nuget.org/v3/registration3/nuget.versioning/4.3.0.json
```

示例响应

```
{
  "@id": "https://api.nuget.org/v3/registration3/nuget.versioning/4.3.0.json",
  "catalogEntry": "https://api.nuget.org/v3/catalog0/data/2017.08.11.18.24.22/nuget.versioning.4.3.0.json",
  "listed": true,
  "packageContent": "https://api.nuget.org/v3-flatcontainer/nuget.versioning/4.3.0/nuget.versioning.4.3.0.nupkg",
  "published": "2017-08-11T18:24:14.36+00:00",
  "registration": "https://api.nuget.org/v3/registration3/nuget.versioning/index.json"
}
```

推送和删除

2019/6/28 • [Edit Online](#)

可以推送、删除（或取消列出，具体取决于服务器实现），并重新列出包使用 NuGet V3 API。这些操作所基于的中断 `PackagePublish` 资源中找到[服务索引](#)。

版本管理

以下 `@type` 使用值：

<code>@TYPE</code>	
PackagePublish/2.0.0	初始版本

基 URL

以下 API 的基 URL 是的值 `@id` 的属性 `PackagePublish/2.0.0` 包源中的资源[服务索引](#)。以下文档中，使用 `nuget.org` 的 URL。请考虑 <https://www.nuget.org/api/v2/package> 作为占位符 `@id` 服务索引中找到的值。

请注意此 URL 指向与旧的 V2 推送终结点相同的位置，因为协议是相同。

HTTP 方法

`PUT`，`POST` 和 `DELETE` 此资源支持 HTTP 方法。有关每个终结点上支持哪些方法，请参阅下面。

将包推送

NOTE

nuget.org 已[其他要求](#)与推送终结点进行交互。

nuget.org 支持推送新的包使用以下 API。如果已存在具有提供的 ID 和版本的包，nuget.org 将拒绝推送。其他包源可能支持替换现有的包。

```
PUT https://www.nuget.org/api/v2/package
```

请求参数

X-NuGet-ApiKey	Header	string	是	例如， <code>X-NuGet-ApiKey: {USER_API_KEY}</code>

API 密钥是从包源获得由用户和配置到客户端不透明的字符串。强制要求任何特定字符串格式，但 API 密钥的长度不应超过合理的大小，为 HTTP 标头值。

请求正文

请求正文必须采用以下形式：

多部分窗体数据

请求标头 Content-Type 是 multipart/form-data 和请求正文中的第一项是推送.nupkg 的原始字节。多部分正文中后面的项将被忽略。将忽略的文件的名称或任何其他标头的多部分项。

响应

HTTP 状态	描述
201, 202	已成功推送包
400	提供的包无效
409	已存在具有提供的 ID 和版本的包

服务器实现上成功状态代码返回成功推送包时存在差异。

删除包

nuget.org 将解释为包删除请求的"取消列出"。这意味着包仍可用于包的现有使用者，但包不会再出现在搜索结果中或 web 界面中。有关这种做法的详细信息，请参阅[删除包策略](#)。其他服务器实现可以自由地解释为硬删除此信号，软删除，或取消列出。例如，[NuGet.Server](#)（仅支持较旧的 V2 API 的服务器实现）支持处理此请求作为未列出或硬删除基于配置选项。

```
DELETE https://www.nuget.org/api/v2/package/{ID}/{VERSION}
```

请求参数

参数	类型	说明	是否必需	示例
Id	URL	string	是	要删除的包 ID
VERSION	URL	string	是	要删除的包的版本
X-NuGet-ApiKey	Header	string	是	例如, X-NuGet-ApiKey: {USER_API_KEY}

响应

HTTP 状态	描述
204	删除此包。
404	利用所提供的任何包 ID 和 VERSION 存在

重新列出包

如果某个包未列出，则就可以使该包在使用"重新列出"终结点的搜索结果中再次可见。此终结点有形状[删除（取消列出）终结点](#)但使用 POST HTTP 方法，而不是 DELETE 方法。

如果包已列出，请求仍会成功。

```
POST https://www.nuget.org/api/v2/package/{ID}/{VERSION}
```

请求参数

参数	类型	描述	是否必需	说明
Id	URL	string	是	重新列出包的 ID
VERSION	URL	string	是	重新列出包的版本
X-NuGet-ApiKey	Header	string	是	例如, X-NuGet-ApiKey: {USER_API_KEY}

响应

状态码	描述
200	现在, 该包列
404	利用所提供的任何包 <code>ID</code> 和 <code>VERSION</code> 存在

推送符号包

2019/9/4 • [Edit Online](#)

可以使用 NuGet V3 API 推送符号包([snupkg](#))。这些操作基于 `SymbolPackagePublish` 在[服务索引](#)中找到的资源。

版本管理

使用以下 `@type` 值：

<code>@TYPE</code>	
<code>SymbolPackagePublish/4.9.0</code>	初始版本

基 URL

以下 api 的基 URL 是包源的[服务索引](#)中 `@id` `SymbolPackagePublish/4.9.0` 资源的属性的值。对于以下文档，使用 nuget 的 URL。请 <https://www.nuget.org/api/v2/symbolpackage> 考虑在服务索引中 `@id` 找到的值的占位符。

HTTP 方法

此资源支持 HTTP 方法。[PUT](#)

推送符号包

nuget.org 支持使用以下 API 推送新的符号包格式([snupkg](#))。

```
PUT https://www.nuget.org/api/v2/symbolpackage
```

可以多次提交具有相同 ID 和版本的符号包。在以下情况下，将拒绝符号包。

- ID 和版本相同的包不存在。
- 已推送具有相同 ID 和版本的符号包，但尚未发布。
- 符号包([snupkg](#))无效(请参阅[符号包约束](#))。

请求参数

NAME				
X-NuGet-ApiKey	Header	string	是	例如， <code>X-NuGet-ApiKey: {USER_API_KEY}</code>

API 密钥是用户从包源中获得的不透明字符串，配置到客户端。不会强制执行任何特定的字符串格式，但 API 密钥的长度不应超过 HTTP 标头值的合理大小。

请求正文

符号推送的请求正文与包推送请求的请求正文相同(请参阅[包推送和删除](#))。

响应

HTTP 状态码	错误描述
201	已成功推送符号包。
400	提供的符号包无效。
401	用户无权执行此操作。
404	具有提供的 ID 和版本的对应包不存在。
409	已推送具有提供的 ID 和版本的符号包, 但它尚不可用。
413	包太大。

报告滥用 URL 模板

2018/9/5 • [Edit Online](#)

就可以生成可由用户报告滥用行为有关特定包的 URL 的客户端。当包源想要启用所有客户端体验（甚至是第三方）委派对包源的滥用报告时，这很有用。

用于生成此 URL 使用的资源是 `ReportAbuseUriTemplate` 资源中找到[服务索引](#)。

版本管理

以下 `@type` 使用值：

<code>@TYPE</code>	
<code>ReportAbuseUriTemplate/3.0.0-beta</code>	初始版本
<code>ReportAbuseUriTemplate/3.0.0-rc</code>	别名 <code>ReportAbuseUriTemplate/3.0.0-beta</code>

URL 模板

以下 API 的 URL 是的值 `@id` 属性与前面提到的资源之一相关联 `@type` 值。

HTTP 方法

尽管客户端不应代表用户向报告滥用 URL 发出请求，但应支持 web 页 `GET` 方法，以允许单击的 URL 来轻松地在 web 浏览器中打开。

构造的 URL

提供了已知的包 ID 和版本，客户端实现可以构造用于访问 web 界面的 URL。客户端实现应显示给用户允许应用来打开 web 浏览器到 URL 并进行任何必要的滥用报告此构造的 URL（或可单击的链接）。滥用报告表单的实现取决于服务器实现。

值 `@id` 是 URL 的字符串包含任何以下占位符标记：

URL 占位符

NAME			
<code>{id}</code>	字符串	否	将包 ID 与举报违规帖子
<code>{version}</code>	字符串	否	将包版本到举报违规帖子

`{id}` 和 `{version}` 解释服务器实现的值必须是不区分大小写和不敏感是否规范化版本。

例如，nuget.org 的报告滥用行为模板如下所示：

```
https://www.nuget.org/packages/{id}/{version}/ReportAbuse
```

如果需要有关 NuGet.Versioning 4.3.0 报告滥用行为窗体显示的链接的客户端实现，它将产生以下 URL 并将其提供

给用户：

<https://www.nuget.org/packages/NuGet.Versioning/4.3.0/ReportAbuse>

存储库签名

2019/4/23 • [Edit Online](#)

如果包源支持添加存储库发布的包签名，就可以确定所用的包源的签名证书的客户端。此资源允许客户端以检测包是否已签名的存储库已被篡改或包含意外的签名证书。

此存储库签名信息中提取的使用的资源是 `RepositorySignatures` 资源中找到[服务索引](#)。

版本管理

以下 `@type` 使用值：

<code>@TYPE</code>	
RepositorySignatures/4.7.0	初始版本
RepositorySignatures/4.9.0	NuGet v4.9 + 客户端支持
RepositorySignatures/5.0.0	允许启用 <code>allRepositorySigned</code> 。NuGet 5.0 版 + 客户端支持

基 URL

以下 API 的入口点 URL 是的值 `@id` 属性与前面提到的资源相关联 `@type` 值。本主题使用占位符 URL `{@id}`。

请注意，与其他资源不同 `{@id}` URL，才能通过 HTTPS 提供服务。

HTTP 方法

在存储库签名资源支持的 HTTP 方法中找到的所有 URL `GET` 和 `HEAD`。

存储库签名索引

存储库签名索引包含两条信息：

- 所有包源上可以都找到的是受此包的来源签署的存储库。
- 包源来进行包所用证书的列表。

在大多数情况下，证书的列表将只会追加到。以前的签名证书已过期，但包源需要开始使用新的签名证书时，会将新证书添加到列表。如果从列表中删除证书，这意味着，使用已删除的签名证书创建的所有包签名应不再被都视为有效客户端。在这种情况下，包的签名（但不是一定是包）是无效的。客户端策略可能会允许安装未签名包。

在证书吊销（例如密钥泄漏）的情况下应将包的源文件重新签名由受影响的证书签名的所有包。此外，包源应从签名的证书列表中删除受影响的证书。

以下请求获取存储库签名索引。

```
GET {@id}
```

存储库签名索引是包含具有以下属性的对象的 JSON 文档：

参数	类型	是否必需	说明
allRepositorySigned	boolean	是	必须为 <code>false</code> 4.7.0 和 4.9.0 资源
signingCertificates	对象的数组	是	

`allRepositorySigned` 布尔值设置为 `false`, 如果包源具有一些具有任何存储库的签名的包。如果布尔值设置为 `true`, 可在上找到的所有包源必须由其中一个签名证书中所述的存储库签名 `signingCertificates`。

WARNING

`allRepositorySigned` 布尔值必须为假的 4.7.0 和 4.9.0 资源上。NuGet v4.7、v4.8 和 v4.9 客户端不能从具有的源安装包 `allRepositorySigned` 设置为 `true`。

应在一个或多个签名证书 `signingCertificates` 数组如果 `allRepositorySigned` 布尔值设置为 `true`。如果数组为空并 `allRepositorySigned` 设置为 `true`, 来自源的所有包应都视为无效, 尽管客户端策略可能仍允许使用包。此数组中的每个元素均具有以下属性的 JSON 对象。

参数	类型	是否必需	说明
contentUrl	string	是	绝对 URL 的 DER 编码的公用证书
指纹	object	是	
主题	string	是	从证书使用者可分辨的名称
issuer	string	是	证书的颁发者的可分辨的名称
notBefore	string	是	证书的有效期的起始时间戳
notAfter	string	是	证书的有效期的结束时间戳

请注意, `contentUrl` 需要通过 HTTPS 提供服务。此 URL 具有任何特定的 URL 模式, 并且必须使用此存储库签名索引文档动态地发现。

此对象中的所有属性 (除了 `contentUrl`) 必须是从证书, 请参阅派生 `contentUrl`。为方便起见, 尽量减少往返过程提供了这些派生的属性。

`fingerprints` 对象具有以下属性:

参数	类型	是否必需	说明
2.16.840.1.101.3.4.2.1	string	是	SHA-256 指纹

密钥名称 `2.16.840.1.101.3.4.2.1` 是 SHA-256 哈希算法的 OID。

所有哈希值必须为小写字符、十六进制编码的字符串表示形式的哈希摘要。

示例请求

```
GET https://api.nuget.org/v3-index/repository-signatures/index.json
```

示例响应

```
{  
    "allRepositorySigned": false,  
    "signingCertificates": [  
        {  
            "fingerprints": {  
                "2.16.840.1.101.3.4.2.1": "0e5f38f57dc1bcc806d8494f4f90fbcedd988b46760709cbeec6f4219aa6157d"  
            },  
            "subject": "CN=NuGet.org Repository by Microsoft, O=NuGet.org Repository by Microsoft, L=Redmond,  
S=Washington, C=US",  
            "issuer": "CN=DigiCert SHA2 Assured ID Code Signing CA, OU=www.digicert.com, O=DigiCert Inc, C=US",  
            "notBefore": "2018-04-10T00:00:00.000000Z",  
            "notAfter": "2021-04-14T12:00:00.000000Z",  
            "contentUrl": "https://api.nuget.org/v3-index/repository-  
signatures/certificates/0e5f38f57dc1bcc806d8494f4f90fbcedd988b46760709cbeec6f4219aa6157d.crt"  
        }  
    ]  
}
```

搜索

2019/9/4 • [Edit Online](#)

可以使用 V3 API 搜索包源上可用的包。用于搜索的资源是在 `SearchQueryService` 服务索引中找到的资源。

版本管理

使用以下 `@type` 值：

<code>@TYPE</code>	
SearchQueryService	初始版本
SearchQueryService/3.0.0-beta	别名 <code>SearchQueryService</code>
SearchQueryService/3.0.0-rc	别名 <code>SearchQueryService</code>

基 URL

以下 API 的基 URL 是与上述某个资源 `@id` `@type` 值关联的属性的值。在下面的文档中，将使用占位符 `{@id}` 基 URL。

HTTP 方法

注册资源中找到的所有 url 都支持 HTTP 方法 `GET` 和 `HEAD`。

搜索包

搜索 API 允许客户端查询与指定的搜索查询匹配的包页面。搜索查询(如搜索词的词汇切分)的解释由服务器实现确定，但一般的期望是搜索查询用于匹配包 Id、标题、说明和标记。还可以考虑其他包元数据字段。

未列出的包决不会出现在搜索结果中。

```
GET {@id}?q={QUERY}&skip={SKIP}&take={TAKE}&prerelease={PRERELEASE}&semVerLevel={SEMVERLEVEL}
```

请求参数

NAME				
q	URL	string	否	用于筛选包的搜索词
skip	URL	integer	否	要跳过的结果数，用于分页
长	URL	integer	否	要返回的结果数，用于分页
早期	URL	boolean	否	<code>true</code> 或 <code>false</code> 确定是否包括预发布包

NAME	类型	描述	必需	示例
semVerLevel	URL	string	否	SemVer 1.0.0 版本字符串

搜索查询 `q` 按服务器实现所定义的方式进行分析。nuget.org 支持针对 [各种字段](#) 的基本筛选。如果未提供 `q`，则应在 `skip` 和 `take` 施加的边界内返回所有包。这会启用 NuGet Visual Studio 体验中的“浏览”选项卡。

`skip` 参数默认为 0。

`take` 参数应为大于零的整数。服务器实现可能会施加最大值。

如果 `prerelease` 未提供，则将排除预发布包。

Query 参数用于选择 [SemVer 2.0.0 包](#)。`semVerLevel` 如果此查询参数被排除，则将仅返回 SemVer 1.0.0 兼容版本的包（其中包含 [标准 NuGet 版本控制注意事项](#)，如具有 4 个整数部分的版本字符串）。如果 `semVerLevel=2.0.0` 提供了，则将返回 SemVer 1.0.0 和 SemVer 2.0.0 兼容包。有关详细信息，请参阅 [SemVer 2.0.0 support for nuget.org](#)。

响应

响应是最多 `take` 包含搜索结果的 JSON 文档。搜索结果按包 ID 分组。

根 JSON 对象具有以下属性：

NAME	类型	必需	描述
totalHits	integer	是	匹配项的总数。 <code>skip</code> 忽略和 <code>take</code>
数据	对象数组	是	请求匹配的搜索结果

搜索结果

`data` 数组中的每一项都是一个 JSON 对象，其中包含一组共享同一包 ID 的包版本。对象具有以下属性：

NAME	类型	必需	描述
id	string	是	匹配包的 ID
version	string	是	包的完整 SemVer 2.0.0 版本字符串（可能包含元数据）
description	string	否	
版本	对象数组	是	与 <code>prerelease</code> 参数匹配的包的所有版本
作者	字符串或字符串数组	否	
iconUrl	string	否	
licenseUrl	string	否	
所有者	字符串或字符串数组	否	
projectUrl	string	否	

NAME	类型	必需	
注册	string	否	关联 注册索引 的绝对 URL
摘要	string	否	
标记	字符串或字符串数组	否	
标题	string	否	
totalDownloads	integer	否	此值可由 <code>versions</code> 数组中的下载内容和
得到	boolean	否	一个 JSON 布尔值, 指示是否 验证 了包

在 nuget.org 上, 已验证的包是指其包 ID 与保留的 ID 前缀匹配并由保留前缀的一个所有者拥有的包 ID。有关详细信息, 请参阅[有关 ID 前缀保留的文档](#)。

搜索结果对象中包含的元数据取自最新的包版本。`versions` 数组中的每一项都是具有以下属性的 JSON 对象:

NAME	类型	必需	
@id	string	是	关联 注册叶 的绝对 URL
version	string	是	包的完整 SemVer 2.0.0 版本字符串(可能包含生成元数据)
下载	integer	是	此特定包版本的下载数

示例请求

```
GET https://azuresearch-usnc.nuget.org/query?q=NuGet.Versioning&prerelease=false&semVerLevel=2.0.0
```

示例响应

```
{
  "totalHits": 2,
  "data": [
    {
      "registration": "https://api.nuget.org/v3/registration3/nuget.versioning/index.json",
      "id": "NuGet.Versioning",
      "version": "4.4.0",
      "description": "NuGet's implementation of Semantic Versioning.",
      "summary": "",
      "title": "<span data-ttu-id=\"20b01-101\">NuGet. 版本控制</span><span class=\"sxs-lookup\"><span data-stu-id=\"20b01-101\">NuGet.Versioning</span></span>",
      "licenseUrl": "https://raw.githubusercontent.com/NuGet/NuGet.Client/dev/LICENSE.txt",
      "tags": [
        "semver",
        "semantic",
        "versioning"
      ],
      "authors": [
        "NuGet"
      ],
      "totalDownloads": 141896,
      "...": ...
    }
  ]
}
```

```
"verified": true,
"versions": [
    {
        "version": "3.3.0",
        "downloads": 50343,
        "@id": "https://api.nuget.org/v3/registration3/nuget.versioning/3.3.0.json"
    },
    {
        "version": "3.4.3",
        "downloads": 27932,
        "@id": "https://api.nuget.org/v3/registration3/nuget.versioning/3.4.3.json"
    },
    {
        "version": "4.0.0",
        "downloads": 63004,
        "@id": "https://api.nuget.org/v3/registration3/nuget.versioning/4.0.0.json"
    },
    {
        "version": "4.4.0",
        "downloads": 617,
        "@id": "https://api.nuget.org/v3/registration3/nuget.versioning/4.4.0.json"
    }
]
},
{
    "@id": "https://api.nuget.org/v3/registration3/nerdbank.gitversioning/index.json",
    "@type": "Package",
    "registration": "https://api.nuget.org/v3/registration3/nerdbank.gitversioning/index.json",
    "id": "Nerdbank.GitVersioning",
    "version": "2.0.41",
    "description": "Stamps your assemblies with semver 2.0 compliant git commit specific version information and provides NuGet versioning information as well.",
    "summary": "Stamps your assemblies with semver 2.0 compliant git commit specific version information and provides NuGet versioning information as well.",
    "title": "<span data-ttu-id=\"20b01-102\">Nerdbank.GitVersioning</span><span class=\"sxs-lookup\"><span data-stu-id=\"20b01-102\">Nerdbank.GitVersioning</span></span>",
    "licenseUrl": "https://raw.githubusercontent.com/AArnott/Nerdbank.GitVersioning/ed547462f7/LICENSE.txt",
    "projectUrl": "http://github.com/aarnott/Nerdbank.GitVersioning",
    "tags": [
        "git",
        "commit",
        "versioning",
        "version",
        "assemblyinfo"
    ],
    "authors": [
        "Andrew Arnott"
    ],
    "totalDownloads": 11906,
    "verified": false,
    "versions": [
        {
            "version": "1.6.35",
            "downloads": 10229,
            "@id": "https://api.nuget.org/v3/registration3/nerdbank.gitversioning/1.6.35.json"
        },
        {
            "version": "2.0.41",
            "downloads": 1677,
            "@id": "https://api.nuget.org/v3/registration3/nerdbank.gitversioning/2.0.41.json"
        }
    ]
}
]
```

服务索引

2019/1/16 • [Edit Online](#)

服务索引是一个 JSON 文档是 NuGet 包源的入口点，并允许客户端实现，若要了解包源的功能。服务索引是一个具有两个必需属性的 JSON 对象：`version`（服务索引的架构版本）和 `resources`（终结点或功能的包源）。

nuget.org 的服务索引位于 <https://api.nuget.org/v3/index.json>。

版本管理

`version` 值是 SemVer 2.0.0 版本解析，因此字符串指示服务索引的架构版本。API 要求的版本字符串有主版本号的 `3`。服务索引架构做出非重大更改后，会增加版本字符串的次要版本。

服务索引的每个资源是版本控制独立于服务索引的架构版本。

当前架构版本是 `3.0.0`。`3.0.0` 版本在功能上等效于较旧 `3.0.0-beta.1` 版本但只是首选，因为它更清楚地传达稳定、定义架构。

HTTP 方法

服务索引是使用 HTTP 方法可以访问 `GET` 和 `HEAD`。

资源

`resources` 属性包含此包源支持的资源数组。

资源

资源是中的对象 `resources` 数组。它表示包源的版本控制功能。每个资源有以下属性：

NAME	■	■	■
<code>@id</code>	字符串	是	资源的 URL
<code>@type</code>	字符串	是	一个字符串常数，表示资源类型
注释	字符串	否	资源的人工可读说明

`@id` 是 URL 必须是绝对路径，并且必须具有 HTTP 或 HTTPS 架构。

`@type` 用于标识要与资源交互时使用的特定协议。资源类型是不透明的字符串，但通常采用格式：

```
{RESOURCE_NAME}/{RESOURCE_VERSION}
```

客户端应进行硬编码 `@type` 他们了解并查找包源的服务索引中的值。确切 `@type` 枚举中列出的单个资源引用文档上目前所用的值 [API 概述](#)。

本文档中，为不同的资源有关的文档将实质上是按分组 `{RESOURCE_NAME}` 它类似于按方案进行分组的服务索引中找到。

没有任何要求每个资源具有一个唯一 `@id` 或 `@type`。负责要确定哪些资源通过另一个首选的客户端实现。一个可能的实现是相同或兼容的资源 `@type` 可以以轮循机制方式发生连接故障或服务器错误时使用。

示例请求

```
GET https://api.nuget.org/v3/index.json
```

示例响应

```
{
  "version": "3.0.0",
  "resources": [
    {
      "@id": "https://api.nuget.org/v3-flatcontainer/",
      "@type": "PackageBaseAddress/3.0.0",
      "comment": "基 URL 的 Azure 存储.NET Core 的 NuGet 包注册信息，以存储格式 https://api.nuget.org/v3-flatcontainer/ {低 id} / {id 低}。{版本较低}.nupkg"
    },
    {
      "@id": "https://www.nuget.org/api/v2/package",
      "@type": "PackagePublish/2.0.0"
    },
    {
      "@id": "https://api-v2v3search-0.nuget.org/query",
      "@type": "SearchQueryService/3.0.0-rc",
      "comment": "查询 RC 客户端使用的（主）NuGet 搜索服务的终结点"
    },
    {
      "@id": "https://api-v2v3search-0.nuget.org/autocomplete",
      "@type": "SearchAutocompleteService/3.0.0-rc",
      "comment": "记忆式键入功能服务终结点的 NuGet 搜索（主）RC 客户端使用"
    },
    {
      "@id": "https://api.nuget.org/v3/registration2/",
      "@type": "RegistrationsBaseUrl/3.0.0-rc",
      "comment": "基 URL 的 Azure 存储 NuGet 包注册信息的存储位置 RC 客户端使用。此基 URL 不包括 SemVer 2.0.0 包。"
    }
  ]
}
```

查询所有发布到 nuget.org 的包

2020/4/13 • [Edit Online](#)

旧版 OData V2 API 上一个常见查询模式为枚举所有发布到 nuget.org 的包，按照包发布的时间排序。需要这类针对 nuget.org 的查询的情况多种多样：

- 完全复制 nuget.org
- 检测包发布新版本的时间
- 查找依赖包的包

执行此操作的老式方法通常依靠按照时间戳对 OData 包实体进行排序并且用 `skip` 和 `top` (页大小) 参数跨大型结果集进行分页。遗憾的是，这种方法有一些缺点：

- 可能会丢失包，因为查询的对象为常常更改顺序的数据
- 查询响应时间长，因为查询未优化（最优化的查询支持官方 NuGet 客户端的主流场景）
- 使用弃用和未记录的 API，意味着将来不保证支持此类查询
- 不能按照发生的确切顺序重播历史记录

因此，可以遵循以下指南以更加可靠新颖的方法解决上述情况。

概述

本指南的中心是 [NuGet API](#) 中称为“目录”的资源。目录是仅限追加的 API，它允许调用方查看在 nuget.org 中添加、修改和删除的包的完整历史记录。如果对发布到 nuget.org 的所有包甚至是包的子集感兴趣，则最好用目录来跟进了解当前可用的包集。

本指南旨在提供高级演练，如果你对目录的详情细节感兴趣，请参阅它的 [API 引用文档](#)。

以下步骤可以选择任意的编程语言来执行。如果需要完整的运行示例，请查看下面提及的 [C# 示例](#)。

反之，按照以下指南生成可靠的目录阅读器。

初始化游标

生成可靠目录阅读器的第一步是实现游标。有关目录游标设计的完整详细信息，请参阅[目录引用文档](#)。简而言之，游标是时间中的某一点，你在目录中处理事件直到该点。目录中的事件表示包发布和其他包更改。如果你关注（从一开始）发布到 NuGet 的所有包，那么将游标初始化为“最小值”时间戳（例如 .NET 中的 `DateTime.MinValue`）。如果仅关注现在开始发布的包，则使用当前时间戳作为初始游标值。

对于本指南，我们要将游标初始化为一小时前的时间戳。现在，只需在内存中保存时间戳。

```
DateTime cursor = DateTime.UtcNow.AddHours(-1);
```

确定目录索引 URL

NuGet API 中每个资源（终结点）的位置应使用[服务索引](#)发现。因为本指南重点介绍 nuget.org，所以我们使用 nuget.org 的服务索引。

```
GET https://api.nuget.org/v3/index.json
```

服务文档是包含 nuget.org 上所有资源的 JSON 文档。查找具有 `Catalog/3.0.0` 的 `@type` 属性值的资源。相关联的 `@id` 属性值是到目录索引本身的 URL。

查找新的目录叶

使用前面步骤中找到的 `@id` 属性值下载目录索引：

```
GET https://api.nuget.org/v3/catalog0/index.json
```

反序列化目录索引。筛选出 `commitTimeStamp` 小于或等于当前游标值的所有目录页对象。

对于每个剩余的目录页，使用 `@id` 属性下载完整文档。

```
GET https://api.nuget.org/v3/catalog0/page2926.json
```

反序列化目录页。筛选出 `commitTimeStamp` 小于或等于当前游标值的所有目录叶对象。

在下载没有筛选出的所有目录页之后，你会具有目录叶对象集，用于表示在游标时间戳和现在时间之间已发布的、未列出、已列出或删除的包。

处理目录叶

此时，可以对目录项执行任何自定义处理。如果只需要包的 ID 和版本，则可以在页中找到的目录项对象上检查 `nuget:id` 和 `nuget:version` 属性。请务必查看 `@type` 属性，了解目录项是否涉及现有包或者删除包。

如果对有关包的元数据感兴趣（例如描述、依赖项、.nupkg 大小等），则可以使用 `@id` 属性提取目录叶文档。

```
GET https://api.nuget.org/v3/catalog0/data/2015.02.01.11.18.40/windowsazure.storage.1.0.0.json
```

本文档不仅有包元数据资源中包含的所有元数据，而且还有更多内容！

通过该步骤可以实现自定义逻辑。无论要使用目录叶来做什么，本指南中的其他步骤也以大致相同的方法实现。

下载 .nupkg

如果对下载目录中找到的包的 .nupkg 感兴趣，那么可以使用包内容资源。但请注意，在目录中找到包和包在包内容资源中可用之间有短暂的延迟。所以，如果在尝试下载目录中找到的包的 .nupkg 时遇到 `404 Not Found`，只需稍后进行重试。由 GitHub 问题 [NuGet/NuGetGallery#3455](#) 跟踪此延迟的修复。

向前移动游标

成功处理目录项后，需要确定要保存的新游标值。为了执行此操作，请找到处理的所有目录项的最大（按最新时间顺序）`commitTimeStamp`。这是新的游标值。将其保存到某些永久存储中，例如数据库、文件系统或 blob 存储。当需要获取更多目录项时，只需通过从此永久存储中初始化游标值从第一步开始。

如果应用程序引发异常或错误，请勿向前移动游标。向前移动游标意味着不再需要处理游标前的目录项。

如果某些原因导致处理目录叶的方式有错误，只需及时向后移动游标并允许代码重新处理旧的目录项。

C# 示例代码

因为目录是可以通过 HTTP 获得的 JSON 文档集，所以可以使用有 HTTP 客户端和 JSON 反序列化程序的任意编程语言与之交互。

C# 示例可在 [NuGet/示例存储库](#) 中获得。

```
git clone https://github.com/NuGet/Samples.git
```

目录 SDK

使用目录最简单的方法是使用预发布 .NET 目录 SDK 包：[NuGet.Protocol.Catalog](#)。此包可在 `nuget-build` MyGet 源 <https://dotnet.myget.org/F/nuget-build/api/v3/index.json> 上获得，可以使用 NuGet 包源 URL 找到该源。

可以将此包安装到与 `netstandard1.3` 或更高版本（例如 .NET Framework 4.6）兼容的项目。

使用此包的示例可在 [NuGet.Protocol.Catalog.Sample](#) 项目中的 GitHub 上获得。

示例输出

```
2017-11-10T22:16:44.8689025+00:00: Found package details leaf for xSkrape.APIWrapper.REST 1.0.2.
2017-11-10T22:16:54.6972769+00:00: Found package details leaf for xSkrape.APIWrapper.REST 1.0.1.
2017-11-10T22:19:20.6385542+00:00: Found package details leaf for Platform.EnUnity 1.0.8.
...
2017-11-10T23:05:04.9695890+00:00: Found package details leaf for xSkrape.APIWrapper.Base 1.0.1.
2017-11-10T23:05:04.9695890+00:00: Found package details leaf for xSkrape.APIWrapper.Base 1.0.2.
2017-11-10T23:07:23.1303569+00:00: Found package details leaf for VeiculoX.Model 0.0.15.
Processing the catalog leafs failed. Retrying.
fail: NuGet.Protocol.Catalog.LoggerCatalogLeafProcessor[0]
    10 catalog commits have been processed. We will now simulate a failure.
warn: NuGet.Protocol.Catalog.CatalogProcessor[0]
    Failed to process leaf
https://api.nuget.org/v3/catalog0/data/2017.11.10.23.veiculox.model.0.0.15.json (VeiculoX.Model 0.0.15,
PackageDetails).
warn: NuGet.Protocol.Catalog.CatalogProcessor[0]
    13 out of 59 leaves were left incomplete due to a processing failure.
warn: NuGet.Protocol.Catalog.CatalogProcessor[0]
    1 out of 1 pages were left incomplete due to a processing failure.
2017-11-10T23:07:23.1303569+00:00: Found package details leaf for VeiculoX.Model 0.0.15.
2017-11-10T23:07:33.0212446+00:00: Found package details leaf for VeiculoX.Model 0.0.14.
2017-11-10T23:07:41.6621837+00:00: Found package details leaf for VeiculoX.Model 0.0.13.
2017-11-10T23:09:58.5728614+00:00: Found package details leaf for CreaSoft.Composition.Web.Extensions 1.1.0.
2017-11-10T23:09:58.5728614+00:00: Found package details leaf for DarkXahTeP.Extensions.Configuration.Consul
0.0.4.
2017-11-10T23:09:58.5728614+00:00: Found package details leaf for xSkape.APIWrapper.REST.Sample 1.0.3.
2017-11-10T23:10:09.0574930+00:00: Found package details leaf for Microsoft.VisualStudio.Imaging 15.4.27004.
2017-11-10T23:10:09.0574930+00:00: Found package details leaf for
Microsoft.VisualStudio.Imaging.Interop.14.0.DesignTime 14.3.25407.
2017-11-10T23:10:09.0574930+00:00: Found package details leaf for Microsoft.VisualStudio.Language.Intellisense
15.4.27004.
2017-11-10T23:10:09.0574930+00:00: Found package details leaf for
Microsoft.VisualStudio.Language.StandardClassification 15.4.27004.
2017-11-10T23:10:09.0574930+00:00: Found package details leaf for Microsoft.VisualStudio.ManagedInterfaces
8.0.50727.
2017-11-10T23:10:09.0574930+00:00: Found package details leaf for xSkape.APIWrapper.REST.Sample 1.0.2.
2017-11-10T23:10:09.0574930+00:00: Found package details leaf for xSkape.APIWrapper.REST.Sample 1.0.3.
```

最小示例

有关详细说明与目录交互的较少依赖项的示例，请参阅 [CatalogReaderExample](#) 示例项目。项目面向 `netcoreapp2.0` 并依赖 [NuGet.Protocol 4.4.0](#)（适用于解析服务索引）和 [Newtonsoft.Json 9.0.1](#)（适用于 JSON 序列化）。

代码的主要逻辑在 [Program.cs](#) 文件中可见。

示例输出

```
No cursor found. Defaulting to 11/2/2017 9:41:28 PM.  
Fetched catalog index https://api.nuget.org/v3/catalog0/index.json.  
Fetched catalog page https://api.nuget.org/v3/catalog0/page2935.json.  
Processing 69 catalog leaves.  
11/2/2017 9:32:35 PM: DotVVM.Compiler.Light 1.1.7 (type is nuget:PackageDetails)  
11/2/2017 9:32:35 PM: Momentum.Pm.Api 5.12.181-beta (type is nuget:PackageDetails)  
11/2/2017 9:32:44 PM: Momentum.Pm.PortalApi 5.12.181-beta (type is nuget:PackageDetails)  
11/2/2017 9:35:14 PM: Genesys.Extensions.Standard 3.17.11.40 (type is nuget:PackageDetails)  
11/2/2017 9:35:14 PM: Genesys.Extensions.Core 3.17.11.40 (type is nuget:PackageDetails)  
11/2/2017 9:35:14 PM: Halforbit.DataStores.FileStores.Serialization.Bond 1.0.4 (type is nuget:PackageDetails)  
11/2/2017 9:35:14 PM: Halforbit.DataStores.FileStores.AmazonS3 1.0.4 (type is nuget:PackageDetails)  
11/2/2017 9:35:14 PM: Halforbit.DataStores.DocumentStores.DocumentDb 1.0.6 (type is nuget:PackageDetails)  
11/2/2017 9:35:14 PM: Halforbit.DataStores.FileStores.BlobStorage 1.0.5 (type is nuget:PackageDetails)  
...  
11/2/2017 10:23:54 PM: Cake.GitPackager 0.1.2 (type is nuget:PackageDetails)  
11/2/2017 10:23:54 PM: UtilPack.NuGet 2.0.0 (type is nuget:PackageDetails)  
11/2/2017 10:23:54 PM: UtilPack.NuGet.AssemblyLoading 2.0.0 (type is nuget:PackageDetails)  
11/2/2017 10:26:26 PM: UtilPack.NuGet.Deployment 2.0.0 (type is nuget:PackageDetails)  
11/2/2017 10:26:26 PM: UtilPack.NuGet.Common.MSBuild 2.0.0 (type is nuget:PackageDetails)  
11/2/2017 10:26:36 PM: InstaClient 1.0.2 (type is nuget:PackageDetails)  
11/2/2017 10:26:36 PM: SecureStrConvertor.VARUN_RUSIYA 1.0.0.5 (type is nuget:PackageDetails)  
Writing cursor value: 11/2/2017 10:26:36 PM.
```

速率限制

2020/1/31 • [Edit Online](#)

NuGet.org API 强制实施速率限制以防止滥用。超出速率限制的请求会返回以下错误：

```
{  
  "statusCode": 429,  
  "message": "Rate limit is exceeded. Try again in 56 seconds."  
}
```

除了使用速率限制进行请求限制之外，某些 API 还强制执行配额。超过配额的请求会返回以下错误：

```
{  
  "statusCode": 403,  
  "message": "Quota exceeded."  
}
```

下表列出了 NuGet.org API 的速率限制。

包搜索

NOTE

建议使用 NuGet 的[V3 搜索 api](#)，因为它当前未进行速率限制。对于 V1 和 V2 搜索 API，以下限制适用：

API	HTTP 方法	速率	API 描述
■ /api/v1/Packages	IP	1000/分钟	通过 v1 OData 查询 NuGet 包元数据 Packages 收集
■ /api/v1/Search()	IP	3000/分钟	通过 v1 搜索终结点搜索 NuGet 包
■ /api/v2/Packages	IP	20000/分钟	通过 v2 OData Packages 收集查询 NuGet 包元数据
■ /api/v2/Packages/\$count	IP	100/分钟	通过 v2 OData 查询 NuGet 包计数 Packages 收集

包推送和取消列出

API	HTTP 方法	速率	API 描述
PUT /api/v2/package	API 密钥	350/小时	通过 v2 推送终结点上传新的 NuGet 包(版本)
■ /api/v2/package/{id}/{version}	API 密钥	250/小时	通过 v2 终结点取消列出 NuGet 包(版本)

nuget.org 协议

2018/9/5 • [Edit Online](#)

若要与 nuget.org 进行交互，客户端需要遵守某些协议。由于这些协议，让不断演变，客户端必须标识调用特定 nuget.org API 时，它们使用的协议版本。这样，nuget.org 的旧客户端不间断的方式引入的更改。

NOTE

此页上所述的 API 是特定于 nuget.org 并且没有任何其他 NuGet 服务器上实现来引入这些 API 的假定条件。

有关在 NuGet 生态系统中实施广泛 NuGet API 的信息，请参阅[API 概述](#)。

本主题列出了作为各种协议和时它们可以直接对存在。

NuGet 协议版本 4.1.0

4.1.0 协议指定验证范围密钥与 nuget.org 中，若要验证包对 nuget.org 帐户以外的服务进行交互的用法。请注意，
4.1.0 数目是不透明的字符串，但恰好官方 NuGet 客户端支持此协议的第一个版本的版本。

验证可确保用户创建 API 密钥仅用于 nuget.org 中，并通过使用一次验证作用域键处理该验证或通过第三方服务的验证。这些验证范围密钥可以用于验证包属于 nuget.org 上的特定用户（帐户）。

客户端要求

要求客户端时它们向发出 API 调用传递以下标头**推送到** nuget.org 的包：

X-NuGet-Protocol-Version: 4.1.0

请注意，**X-NuGet-Client-Version** 标头具有类似语义，但将其保留仅供官方 NuGet 客户端。第三方客户端应使用
X-NuGet-Protocol-Version 标头和值。

推送 协议本身中的文档所述 [PackagePublish](#) 资源。

如果客户端与外部服务，需要验证是否属于特定用户（帐户）的包进行交互，它应使用以下协议，并使用作用域的验证密钥和不从 nuget.org 的 API 密钥。

API 请求作用域的验证密钥

此 API 用于获取 nuget.org 作者可以验证拥有的他/她的包的作用域的验证密钥。

POST api/v2/package/create-verification-key/{ID}/{VERSION}

请求参数

NAME	¶	¶	¶	¶
Id	URL	字符串	是	为其请求验证作用域键包 identifier
VERSION	URL	字符串	否	包版本

NAME				
X-NuGet-ApiKey	Header	字符串	是	例如, X-NuGet-ApiKey: {USER_API_KEY}

响应

```
{
  "Key": "{Verify scope key from nuget.org}",
  "Expires": "{Date}"
}
```

若要验证验证作用域键的 API

此 API 用于验证归 nuget.org 作者的包的作用域的验证密钥。

```
GET api/v2/verifykey/{ID}/{VERSION}
```

请求参数

NAME				
Id	URL	字符串	是	为其请求验证作用域 键包标识符
VERSION	URL	字符串	否	包版本
X-NuGet-ApiKey	Header	字符串	是	例如, X-NuGet-ApiKey: {VERIFY_SCOPE_KEY}

NOTE

此验证作用域 API 密钥将在某一天的时间后过期，或在首次使用准。

响应

HTTP	
200	API 密钥无效
403	API 密钥是无效或未授权，以对包推送
404	由包引用 ID 和 VERSION (可选) 不存在

用于发现 nuget.exe 版本的工具

2019/11/5 • [Edit Online](#)

目前，有几种方法可以在计算机上以脚本方式获取 nuget.exe 的最新版本。例如，你可以从 nuget.org 下载并提取 `NuGet.CommandLine` 包。这种方法有一定的复杂性，因为它要求您已经具有 nuget.exe（对于 `nuget.exe install`），或者必须使用基本解压缩工具来解压缩 nupkg，并在内部查找二进制文件。

如果你已有 nuget.exe，还可以使用 `nuget.exe update -self`，但这也需要具有 nuget.exe 的现有副本。此方法还会更新到最新版本。它不允许使用特定版本。

`tools.json` 终结点可用于解决启动问题，并可控制所下载的 nuget.exe 版本。这可以用于 CI/CD 环境或自定义脚本，以发现和下载任何发布版本的 nuget.exe。

可以使用未经身份验证的 HTTP 请求获取 `tools.json` 终结点（例如，在 PowerShell 中 `Invoke-WebRequest` 或 `wget`）。使用 JSON 反序列化程序可以对其进行分析，后续 nuget.exe 下载 URL 也可以使用未经身份验证的 HTTP 请求提取。

可以使用 `GET` 方法获取终结点：

```
GET https://dist.nuget.org/tools.json
```

此处提供终结点的 JSON 架构：

```
GET https://dist.nuget.org/tools.schema.json
```

响应

响应是一个 JSON 文档，其中包含 nuget.exe 的所有可用版本。

根 JSON 对象具有以下属性：

“ <code>”</code>	“ <code>”</code>	“ <code>”</code>
nuget.exe	对象数组	是

`nuget.exe` 数组中的每个对象具有以下属性：

“ <code>”</code>	“ <code>”</code>	“ <code>”</code>	“ <code>”</code>
版本	string	是	SemVer 2.0.0 字符串
URL	string	是	下载此版本的 nuget.exe 的绝对 URL
结束	string	是	枚举字符串
上载	string	是	可用版本的大约 ISO 8601 时间戳

数组中的项将按降序、SemVer 2.0.0 顺序排序。这一保证旨在降低对最高版本号感兴趣的客户端的负担。但这意味

着列表不按时间顺序进行排序。例如，如果在较高的主要版本之前的日期提供较低的主要版本，则此服务版本将不会显示在列表的顶部。如果需要由时间戳发布的最新版本，只需按 `uploaded` 字符串对数组进行排序即可。这是因为 `uploaded` 时间戳采用 ISO 8601 格式，可以使用字典排序（即简单字符串排序）按时间顺序进行排序。

`stage` 属性指示此版本工具的审查。

阶段	说明
EarlyAccessPreview	在 下载 网页上还不可见，应由合作伙伴验证
Released(已释放)	在下载站点上可用，但尚不建议用于跨范围使用
ReleasedAndBlessed	在下载站点提供，建议用于使用

使用最新的建议版本的一种简单方法是使用列表中具有 `ReleasedAndBlessed` `stage` 值的第一个版本。这是因为版本按 SemVer 2.0.0 顺序排序。

Nuget.org 上的 `NuGet.CommandLine` 包通常仅更新 `ReleasedAndBlessed` 版本。

示例请求

```
GET https://dist.nuget.org/tools.json
```

示例响应

```
{
  "nuget.exe": [
    {
      "version": "4.8.0-preview3",
      "url": "https://dist.nuget.org/win-x86-commandline/v4.8.0-preview3/nuget.exe",
      "stage": "EarlyAccessPreview",
      "uploaded": "2018-07-06T23:00:00.000000Z"
    },
    {
      "version": "4.7.1",
      "url": "https://dist.nuget.org/win-x86-commandline/v4.7.1/nuget.exe",
      "stage": "ReleasedAndBlessed",
      "uploaded": "2018-08-10T23:00:00.000000Z"
    },
    {
      "version": "4.6.1",
      "url": "https://dist.nuget.org/win-x86-commandline/v4.6.1/nuget.exe",
      "stage": "Released",
      "uploaded": "2018-03-22T23:00:00.000000Z"
    },
    {
      "version": "3.5.0",
      "url": "https://dist.nuget.org/win-x86-commandline/v3.5.0/nuget.exe",
      "stage": "ReleasedAndBlessed",
      "uploaded": "2016-12-19T15:30:00.000000-08:00"
    },
    {
      "version": "2.8.6",
      "url": "https://dist.nuget.org/win-x86-commandline/v2.8.6/nuget.exe",
      "stage": "ReleasedAndBlessed",
      "uploaded": "2015-09-01T12:30:00.000000-07:00"
    }
  ]
}
```

NuGet 客户端 SDK

2020/3/3 • [Edit Online](#)

*NuGet 客户端 SDK*引用一组 NuGet 包：

- `NuGet.Protocol` - 用于与基于 HTTP 和文件的 NuGet 源交互
- `NuGet.Packaging` - 用于与 NuGet 包交互。`NuGet.Protocol` 依赖于此包

你可以在[NuGet/nuget。客户端](#)GitHub 存储库中找到这些包的源代码。

NOTE

有关 NuGet 服务器协议的文档，请参阅[NuGet 服务器 API](#)。

入门

安装包

```
dotnet add package NuGet.Protocol
```

示例

你可以在 GitHub 上的[nuget.exe](#)项目上找到这些示例。

列出包版本

使用[NuGet V3 包内容 API](#)查找 `newtonsoft.json` 的所有版本：

WARNING

It looks like the sample you are looking for does not exist.

下载包

使用[NuGet V3 包内容 API](#)下载 `newtonsoft.json v 12.0.1`：

WARNING

It looks like the sample you are looking for does not exist.

获取包元数据

使用[NuGet V3 包元数据 API](#)获取 `"newtonsoft.json"` 包的元数据：

WARNING

It looks like the sample you are looking for does not exist.

搜索包

使用[NuGet V3 搜索 API](#)搜索 `"json"` 包：

WARNING

It looks like the sample you are looking for does not exist.

第三方文档

可以在以下博客系列中的 Dave Glick (已发布2016:

- 浏览 NuGet v3 库, 第1部分:简介和概念
- 浏览 NuGet v3 库, 第2部分:搜索包
- 浏览 NuGet v3 库, 第3部分:安装包

NOTE

3.4.3版本的 NuGet 客户端 SDK 包发布后, 不久就会编写这些博客文章。较新版本的包可能与博客文章中的信息不兼容。

圣马丁 Björkström 对 Dave Glick 的博客系列进行了跟进博客文章, 其中介绍了使用 NuGet 客户端 SDK 安装 NuGet 包的不同方法:

- 重新进行 NuGet v3 库

错误和警告

2019/9/26 • [Edit Online](#)

在 NuGet 4.3.0 + 中，错误和警告按照本主题中的说明进行编号，并提供详细信息来帮助解决所涉及的问题。

此处列出的错误和警告仅适用于[基于 PackageReference 的项目](#)和 NuGet 4.3.0 +。NuGet 还遵循 MSBuild 属性来禁止显示警告或将警告提升为错误。有关详细信息，请参阅[如何：在 Visual Studio 文档中取消显示编译器警告](#)。

错误

GROUP	III
输入错误无效	NU1001 、 NU1002 、 NU1003
缺少包和项目错误	NU1100 、 NU1101 、 NU1102 、 NU1103 、 NU1104 、 NU1105 、 NU1106 、 NU1107 、 NU1108
兼容性错误	NU1201 、 NU1202 、 NU1203 、 NU1401
NuGet 内部错误	NU1000
签名包错误(创建和验证)	NU3001 、 NU3004 、 NU3005 、 NU3008 、 NU3034
包错误	NU5000 、 NU5001 、 NU5002 、 NU5003 、 NU5004 、 NU5005 、 NU5007 、 NU5008 、 NU5009 、 NU5010 、 NU5011 、 NU5012 、 NU5013 、 NU5014 、 NU5015 、 NU5016 、 NU5017 、 NU5018 、 NU5019 、 NU5020 、 NU5021 、 NU5022 、 NU5023 、 NU5024 、 NU5025 、 NU5026 、 NU5027 、 NU5028 、 NU5029 、 NU5036
许可证特定的包错误	NU5030 、 NU5031 、 NU5032 、 NU5033 、 NU5034 、 NU5035

警告

GROUP	
输入警告无效	NU1501 、 NU1502 、 NU1503
意外的包版本警告	NU1601 、 NU1602 、 NU1603 、 NU1604 、 NU1605 、 NU1606 、 NU1607
冲突解决程序冲突警告	NU1608
包回退警告	NU1701
源警告	NU1801
NuGet 内部警告	NU1500

GROUP	EEEE
签名包警告(创建和验证)	NU3000、NU3002、NU3003、NU3006、NU3007、NU3009、NU3010、NU3011、NU3012、NU3013、NU3014、NU3015、NU3016、NU3017、NU3018、NU3019、NU3020、NU3021、NU3022、NU3023、NU3024、NU3025、NU3026、NU3027、NU3028、NU3029、NU3030、NU3031、NU3032、NU3033、NU3035、NU3036、NU3037、NU3038、NU3040
包警告	NU5100、NU5101、NU5102、NU5103、NU5104、NU5105、NU5106、NU5107、NU5108、NU5109、NU5110、NU5111、NU5112、NU5114、NU5115、NU5116、NU5117、NU5118、NU5119、NU5120、NU5121、NU5122、NU5123、NU5127、NU5128、NU5129、NU5130、NU5131、NU5500
许可证特定包警告	NU5124、NU5125
图标特定包警告	NU5046、NU5047、NU5048

NuGet 错误 NU1000

2018/9/5 • [Edit Online](#)

问题

NuGet 操作时出现问题。我们尚未尚未分配唯一的错误代码针对该问题时，使用 NU1000。可以帮助我们改进，请随时[提出问题](#)与您的错误的详细信息。

解决方案

检查输出窗口（在 Visual Studio）或控制台输出（与 NuGet 命令行工具）有关详细信息。

NuGet 错误 NU1001

2018/9/5 • [Edit Online](#)

The project 'Project' does not specify any target frameworks in 'ProjectFile'

问题

项目不包含一个或多个框架。

解决方案

添加 `TargetFramework` 或 `TargetFrameworks` 属性设置为指定的项目文件。

NuGet 错误 NU1002

2018/9/5 • [Edit Online](#)

'CLEAR' cannot be used in conjunction with other values

问题

无效的输入以及清除关键字的组合。

解决方案

清除单独使用，并忽略所有其他输入。

NuGet 错误 NU1003

2018/9/5 • [Edit Online](#)

PackageTargetFallback 和 AssetTargetFallback 不能一起使用。从项目环境中移除 PackageTargetFallback(deprecated) 参考。

问题

PackageTargetFallback 和 AssetTargetFallback 用于选择资产提供不同的行为，并且不能一起使用。

解决方案

删除不推荐使用 PackageTargetFallback 项目中的元素。

NuGet 错误 NU1100

2018/9/5 • [Edit Online](#)

```
Unable to resolve 'Dependency dll' for 'TargetFramework'
```

问题

依赖关系组不被解析。这是不是包或项目的类型的一般问题。

解决方案

打开项目文件并检查其依赖项的列表。检查每个依赖项存在的包源使用的，以及包是否支持项目的目标框架。

NuGet 错误 NU1101

2019/8/16 • [Edit Online](#)

```
Unable to find package 'PackageName'. No packages exist with this id in source(s): 'sourceA',  
'sourceB', 'sourceC'
```

问题

在任何源上都找不到该包。

解决方案

在 Visual Studio 中检查项目的依赖项，以确保使用的是正确的包标识符和版本号。另外，请检查[NuGet 配置](#)是否确定了应使用的包源。如果使用具有[语义版本控制 2.0.0](#)的包，请确保在 `https://api.nuget.org/v3/index.json` [NuGet 配置](#) 中使用 V3 馈送。

NuGet 错误 NU1102

2018/11/30 • [Edit Online](#)

```
Unable to find package 'PackageName' with version (>= 9.0.1)
- Found 30 version(s) in 'sourceA' [ Nearest version: '4.0.0' ]
- Found 10 version(s) in 'sourceB' [ Nearest version: '4.0.0-rc-2129' ]
- Found 9 version(s) in 'sourceC' [ Nearest version: '3.0.0-beta-00032' ]
- Found 0 version(s) in 'sourceD'
- Found 0 version(s) in 'sourceE'
```

问题

找到的包标识符，但任何源上找不到指定的依赖项范围内的版本。可能会通过一个包并不是用户指定范围。

解决方案

编辑项目文件以更正包版本。此外检查[NuGet 配置](#)标识包源你希望进行使用。您可能需要更改请求的版本，如果此包被直接引用该项目。

NuGet 错误 NU1103

2019/8/16 • [Edit Online](#)

```
Unable to find a stable package 'PackageId' with version (>= 3.0.0)
- Found 10 version(s) in 'sourceA' [ Nearest version: '4.0.0-rc-2129' ]
- Found 9 version(s) in 'sourceB' [ Nearest version: '3.0.0-beta-00032' ]
- Found 0 version(s) in 'sourceC'
- Found 0 version(s) in 'sourceD'
```

问题

项目为依赖关系范围指定了稳定的版本,但在该范围内找不到稳定的版本。找到了预发布版本,但不允许这样做。

解决方案

编辑项目文件中的版本范围,以包括预发布版本。请参阅[包版本控制](#)。

NuGet 错误 NU1104

2018/9/5 • [Edit Online](#)

Project reference does not exist 'ProjectFile'. Check that the project reference is valid and that the project file exists.

问题

ProjectReference 指向不存在的文件。

解决方案

编辑项目文件以更正被引用项目的路径，或若要删除的引用完全不再需要。

NuGet 错误 NU1105

2020/3/3 • [Edit Online](#)

Unable to read project information for 'ProjectFile'. The project file may be invalid or missing targets required for restore.

问题

项目文件存在，但没有为其提供还原信息。

解决方案

命令行

在命令行中，这可能表示该文件已损坏或未导入 `NuGet.targets`。若要导入 `NuGet.targets`，通常建议导入 `Microsoft.Common.targets`。

Visual Studio

在 Visual Studio 中，错误可能表示目标不是类似于命令行方案导入的。

此错误还可能表示项目已卸载。

- 如果你使用的是 Visual Studio 2019 或更高版本，NuGet 将能够重新使用以前还原的已卸载项目的项目。若要使此方案正常工作，需要确保解决方案中的所有项目都已从命令行还原，然后再将其加载到 Visual Studio 中。
- 或者，重新加载项目。

NuGet 错误 NU1106

2018/9/5 • [Edit Online](#)

```
Unable to satisfy conflicting requests for 'PackageId': 'Conflict path' Framework: 'Target graph'
```

问题

无法解析的依赖项约束。

解决方案

编辑项目文件以指定的依赖项而不是确切版本更加开放范围。

NuGet 错误 NU1107

2019/8/16 • [Edit Online](#)

```
Version conflict detected for 'PackageA'. Install/reference 'PackageA' v4.0.0 directly to resolve this issue.
```

```
'PackageB' 3.5.0 -> 'PackageA' (= 3.5.0)  
'PackageC' 4.0.0 -> 'PackageA' (= 4.0.0)
```

问题

无法解析包之间的依赖关系约束。两个不同的包正在请求两个不同版本的 "PackageA"。项目需要选择要使用的 "PackageA" 版本。

解决方案

以所选的确切版本直接 (在项目文件中) 安装/引用 "PackageA"。通常, 选取较高版本是正确的选择。

若要安装特定版本, 请参阅所使用工具的信息:

- [Visual Studio](#)
- [dotnet CLI](#)
- [nuget.exe CLI](#)
- [包管理器控制台](#)

注释

早期版本的 Visual Studio 2017 可能会将其报告为警告 (NU1607)。

NuGet 错误 NU1108

2018/9/5 • [Edit Online](#)

Cycle detected: A -> B -> A

问题

检测到循环依赖项。

解决方案

不正确; 创作包请联系软件包所有者以更正 bug。

说明

Visual Studio 2017 的早期版本可能会报告过此作为警告 (NU1606)。

NuGet 错误 NU1201

2019/5/10 • [Edit Online](#)

示例 1

```
Project 'ProjectA' is not compatible with 'TargetFramework'. Project 'ProjectA' supports:  
- 'TargetFrameworkA'  
- 'TargetFrameworkB'
```

问题

依赖项项目不包含与当前项目兼容的框架。通常情况下，项目的目标框架是比使用项目的更高版本。

解决方案

将项目的目标框架更改为比使用项目的相等或更低版本。

示例 2-NetStandard 已定位项目不能引用 NetCoreApp 已定位项目

```
Project 'ProjectB' is not compatible with netstandard2.0 (.NETStandard,Version=v2.0). Project  
'ProjectB' supports: netcoreapp2.0 (.NETCoreApp,Version=v2.0)
```

问题

这种情况下：

- 项目面向 NetStandard 2.0
- 是 ProjectB 面向 NetCoreApp 2.0
- 项目具有对项目 b 的项目引用

NetStandard 项目永远不会依赖于 NetCoreApp 项目。

解决方案

二者之一：

- 将项目更改为目标 NetCoreApp 2.0

或

- 将项目 b 更改到目标 NetStandard 2.0

NuGet 错误 NU1202

2018/9/5 • [Edit Online](#)

```
Package 'PackageName' 4.0.11 is not compatible with 'TargetFramework'. Package 'PackageName' 4.0.11
supports:
- 'TargetFrameworkA'
- 'TargetFrameworkB'
- 'TargetFrameworkC'
```

问题

依赖项包不包含任何与此项目兼容的资产。

解决方案

将项目的目标框架更改为另一个包支持。

NuGet 错误 NU1203

2018/9/5 • [Edit Online](#)

```
'PackageId' 1.0.0 provides a compile-time reference assembly for 'Foo.dll' on 'TargetFramework',  
but there is no compatible run-time assembly.
```

问题

包不支持项目的 `RuntimeIdentifier`。

解决方案

更改 `RuntimeIdentifier` 根据需要在项目中使用的值。

NuGet 错误 NU1401

2018/9/5 • [Edit Online](#)

The 'PackageId' package requires NuGet client version '5.0.0' or above, but the current NuGet version is '4.3.0'.

问题

包需要功能或框架当前不支持安装的 NuGet 版本。

解决方案

安装最新版 NuGet。请参阅[安装 NuGet 客户端工具](#)。

NuGet 警告 NU1500

2018/9/5 • [Edit Online](#)

问题

NuGet 操作时出现问题。我们尚未尚未分配唯一的警告代码针对该问题时，使用 NU1500。可以帮助我们改进，请随时[提出问题](#)与您的错误的详细信息。

解决方案

检查输出窗口（在 Visual Studio）或控制台输出（与 NuGet 命令行工具）有关详细信息。

NuGet 警告 NU1501

2018/9/5 • [Edit Online](#)

The folder 'FolderPath' does not contain a project to restore.

问题

项目还原尝试操作上找不到。

解决方案

在包含项目的文件夹中运行 nuget 还原。

NuGet 警告 NU1502

2018/9/5 • [Edit Online](#)

Unknown Compatibility Profile: 'aaa'

问题

`RuntimeSupports` 包含无效的配置文件。通常情况下，支持配置文件中找不 `runtime.json` 从当前的依赖项包文件。

解决方案

检查 `RuntimeSupports` 在项目中的值。

NuGet 警告 NU1503

2018/9/5 • [Edit Online](#)

```
Skipping restore for project 'c:\a.csproj'. The project file may be invalid or missing targets required for restore.
```

问题

依赖项项目不能导入的 NuGet 还原目标。这类似于 NU1105 但此处跳过项目，而不是导致还原失败的所有忽略。在复杂的解决方案通常有其他类型的可能不支持还原的项目中。

解决方案

发生这种情况不导入常见属性/目标的自动导入还原的项目。如果项目不需要还原可以忽略此。否则，编辑受影响的项目添加为还原的目标。

NuGet 警告 NU1601

2018/9/5 • [Edit Online](#)

```
Dependency specified was 'PackageId' (>= 3.5.0) but ended up with 'PackageId' 4.0.0.
```

问题

直接的项目依赖项是已升级到更高版本比指定的项目。

解决方案

更新到适当版本的项目中的依赖项。

NuGet 警告 NU1602

2018/9/5 • [Edit Online](#)

'PackageA' 4.0.0 does not provide an inclusive lower bound for dependency 'PackageB' (> 3.5.0). An approximate best match of 3.6.0 was resolved.

问题

包依赖项缺少下限。这不允许还原，以确定最佳匹配项。每个还原将浮动向下尝试查找可以使用较低版本。这意味着该还原将进入联机状态，而不是使用用户包文件夹中已存在的包每次检查所有源。

解决方案

这通常是创作错误的包。请联系包的作者来解决此问题。

NuGet 警告 NU1603

2018/9/5 • [Edit Online](#)

```
'PackageA' 4.0.0 depends on 'PackageB' (>= 4.0.0) but 4.0.0 was not found. An approximate best match of 5.0.0 was resolved.
```

问题

包依赖项指定未找到的版本。通常情况下，包源不包含预期更低绑定版本。与它不同的包针对创作相反，使用更高版本。

这意味着找不到该还原最佳匹配项。每个还原将浮动向下尝试查找可以使用较低版本。这意味着该还原将进入联机状态，而不是使用用户包文件夹中已存在的包每次检查所有源。

解决方案

如果预期包尚未发布，这可能是创作错误的包。请联系包的作者来解决此问题。如果包已发布，然后检查，该命令适用于正在使用的包源。如果使用的专用源，可能需要更新上的源的包。

NuGet 警告 NU1604

2020/3/3 • [Edit Online](#)

Project dependency 'PackageA' ($\leq 9.0.0$) does not contain an inclusive lower bound. Include a lower bound in the dependency version to ensure consistent restore results.

问题

项目依赖项未定义下限。

这意味着还原没有找到最佳匹配项。每个还原将浮动，尝试查找可使用的较低版本。这意味着每次还原都将联网，而不是使用用户包文件夹中已有的包。

解决方案

更新项目的 `PackageReference` `Version` 属性，使其包含下限。例如，从以下内容更改：

```
<PackageReference Version="(9.0.0, )"
```

修改为：

```
<PackageReference Version="[9.0.0, )"
```

或

```
<PackageReference Version="9.0.0"
```

这表示下限。

NuGet 警告 NU1605

2019/10/18 • [Edit Online](#)

示例 1

```
Detected package downgrade: 'PackageB' from 4.0.0 to 3.5.0. Reference the package directly from the project to select a different version.  
'PackageA' 3.5.0 -> 'PackageB' 3.5.0  
'PackageC' 4.0.0 -> 'PackageD' 4.0.0 -> 'PackageB' 4.0.0
```

问题

依赖项包在包的更高版本上指定了版本约束，而不是最终解析还原。这是由于最接近的 wins 规则-解析包时，图形中的“即将出现的包”的版本将覆盖具有相同 ID 的远处包的版本。

解决方案

在项目中出现还原错误时，将包引用添加到包的更高版本。

在上面的示例中，您将向包 B 版本4.0.0 添加包引用：

```
'PackageA' 3.5.0 -> 'PackageB' 3.5.0  
  
'PackageC' 4.0.0 -> 'PackageD' 4.0.0 -> 'PackageB' 4.0.0  
  
'PackageB' 4.0.0
```

由于最接近 wins，对 PackageB v 4.0.0 的直接包引用将优先于对 PackageB v 3.5.0 的传递引用。

示例 2

```
Detected package downgrade: System.IO.FileSystem.Primitives from 4.3.0 to 4.0.1. Reference the package directly from the project to select a different version.  
  
Project -> System.IO.FileSystem 4.0.1 -> runtime.win.System.IO.FileSystem 4.3.0 ->  
System.IO.FileSystem.Primitives (>= 4.3.0)  
  
Project -> System.IO.FileSystem 4.0.1 -> System.IO.FileSystem.Primitives (>= 4.0.1)
```

问题

当在 .NET Core 3.0 或更高版本的项目中同时引用时，与 .NET Core 1.0 和1.1 随附的某些包组合不兼容。问题包通常以 `System.` 或 `Microsoft.` 开头，并具有4.0.0 和4.3.1 之间的版本号。在这种情况下，降级消息将具有从运行时开始的包。依赖关系链。

解决方案

若要解决此问题，请添加以下 PackageReference：

```
<PackageReference Include="Microsoft.NETCore.Targets" Version="3.0.0" PrivateAssets="all" />
```

示例 3

```
Detected package downgrade: Microsoft.NETCore.App from 2.1.8 to 2.1.0. Reference the package directly from the project to select a different version.
```

```
test -> mvc -> Microsoft.NETCore.App (>= 2.1.8)
```

```
test -> Microsoft.NETCore.App (>= 2.1.0)
```

问题

Mvc 项目在包的更高版本上指定了版本约束，而不是最终解析还原。这是由于最接近的 [wins](#) 规则-解析包时，图形中的“即将出现的包”的版本将覆盖具有相同 ID 的远处包的版本。

解决方案

这一特定的错误(NETCore 包)已改进，我将 .NET Core SDK 移到2.2.100 或更高版本。NETCore 是一个自动引用的包，在版本3.0.100 之前 .NET Core SDK 会自动导入。另请参阅[自包含部署运行时前滚](#)。

NuGet 警告 NU1608

2018/9/5 • [Edit Online](#)

```
Detected package version outside of dependency constraint: 'PackageA' 1.0.0 requires 'PackageB' (= 1.0.0) but version 'PackageB' 2.0.0 was resolved.
```

问题

解决的包是不是依赖项约束允许更高版本。这意味着直接引用项目的包将其他包中的依赖项约束重写。

解决方案

在某些情况下，这是有意为之，可以禁止显示警告。否则，请更改项目的引用包以扩大其版本约束。

NuGet 警告 NU1701

2020/3/25 • [Edit Online](#)

Package 'packageId' was restored using 'TargetFrameworkA' instead the project target framework 'TargetFrameworkB'. This package may not be fully compatible with your project.

问题

`PackageTargetFallback` / `AssetTargetFallback` 用于从包中选择资产。警告使用户知道资产可能不是100% 兼容。

解决方案

将项目的目标框架更改为包支持的目标框架。

NuGet 警告 NU1801

2018/9/5 • [Edit Online](#)

问题

读取源时出现错误时 `IgnoreFailedSources` 设置为 `true`, 将其转换为非致命警告。这可以包含任何消息, 是泛型。

解决方案

编辑你的配置来指定有效的源。

NuGet 警告 NU3000

2019/7/24 • [Edit Online](#)

TIP

使用 "验证" 命令可查看包签名和时间戳。

方案 1

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The primary signature does not have a timestamp.
```

问题

包具有不具有时间戳的主签名。

解决方案

若要在签名证书过期后启用长期签名有效性, 请确保包签名时间为时间戳。

方案 2

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': Multiple timestamps are not accepted.
```

问题

包具有具有多个时间戳的签名。

解决方案

请确保每个包签名最多包含1个时间戳。

方案3

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The repository countersignature does not have a timestamp.
```

问题

包中的存储库计数器签名没有时间戳。

解决方案

对于长期签名有效性, 请确保任何包签名都有时间戳。

NuGet 错误 NU3001

2020/3/3 • [Edit Online](#)

方案 1

Invalid password was provided for the certificate file 'certificate.pfx'. Provide a valid password using the '-CertificatePassword' option.

问题

为 NuGet 签名操作提供了密码保护的证书文件。但提供的密码无效或未提供密码。

解决方案

如果使用受密码保护的证书文件对 NuGet 包进行签名，请使用 `-CertificatePassword` 选项传递正确的密码。

方案 2

Certificate file 'certificate.pfx' not found. For a list of accepted ways to provide a certificate, visit <https://docs.nuget.org/docs/reference/command-line-reference>.

问题

为 NuGet 签名操作提供了一个证书文件。但磁盘上不存在该文件。

解决方案

请确保用于对 NuGet 包进行签名的任何证书文件都存在于磁盘上。

方案 3

Certificate file 'random_file.txt' is invalid. For a list of accepted ways to provide a certificate, visit <https://docs.nuget.org/docs/reference/command-line-reference>.

问题

为 NuGet 签名操作提供了一个证书文件，但该文件不是有效的证书文件。

解决方案

请确保用于对 NuGet 包进行签名的任何证书文件都是有效的证书文件。

方案 4

Multiple certificates were found that meet all the given criteria. Use the '`-CertificateFingerprint`' option with the hash of the desired certificate.

问题

使用 `-CertificateSubjectName` 选项向 NuGet 签名命令建议使用证书。但发现多个证书与 windows 证书存储中的证书使用者名称匹配。

解决方案

请将包含所需证书哈希的 "`-CertificateFingerprint`" 选项传递到 NuGet Sign 命令，以唯一标识证书。

方案 5

No certificates were found that meet all the given criteria. For a list of accepted ways to provide a certificate, visit <https://docs.nuget.org/docs/reference/command-line-reference>.

问题

使用 `-CertificateSubjectName` 选项向 NuGet 签名命令建议使用证书。但找不到与 windows 证书存储中的证书使用者名称匹配的证书。

解决方案

请确保传递正确的使用者名称筛选器，否则，请将 `-CertificateFingerprint` 选项与所需证书的哈希传递到 NuGet Sign 命令，以唯一标识证书。

方案6

The following certificate cannot be used for package signing as the private key provider is unsupported:

```
Subject Name: CN=Certificate Subject Name  
SHA1 hash: HASH  
SHA256 hash: HASH  
Issued by: Issuer Subject Name  
Valid from: 4/9/2016 5:00:00 PM to 4/14/2020 5:00:00 AM
```

问题

已将证书传递到不支持 pribate 密钥提供程序的 NuGet 签名命令。

解决方案

目前，由于框架限制，NuGet 签名命令不支持 CNG 密钥私钥提供程序。请使用具有 CAPI 私钥提供程序的证书。

方案7

The package already contains a signature. Remove the existing signature before adding a new signature.

问题

NuGet Sign 命令用于对已具有包签名的包进行签名。

解决方案

请确保对未签名的包进行签名。如果已对包进行签名，请使用 `-Overwrite` 选项覆盖现有签名。

NuGet 警告 NU3002

2019/4/23 • [Edit Online](#)

Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The '-Timestamper' option was not provided. The signed package will not be timestamped. To learn more about this option, please visit <https://docs.nuget.org/docs/reference/command-line-reference>.

问题

`-Timestamper` 未提供到 NuGet Sign 命令选项。

解决方案

若要启用长期签名有效性签名证书过期后, 请确保包的签名加盖时间戳。

NuGet 警告 NU3003

2020/1/31 • [Edit Online](#)

方案 1

Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The package is not signed. Unable to verify signature from an unsigned package.

问题

NuGet 客户端尝试验证不包含包签名的包。

解决方案

请在[NuGet/Home](#)上发布问题以及生成此问题的包。

方案 2

Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The package signature is invalid or cannot be verified on this platform.

问题

NuGet 客户端尝试验证包含无效包签名的包或在当前平台上验证 connot 的签名。

解决方案

请在[NuGet/Home](#)上提出问题，以及生成此问题的包和遇到该问题的平台。

NOTE

当 NuGet 的[签名验证模式](#)设置为 accept (默认值)时，将引发 NU3003 作为警告。当 NuGet 的签名验证模式设置为 "必需" 时，或者在运行 `nuget verify -signatures` 命令时，NU3003 将从警告提升为错误。

NuGet 错误 NU3004

2019/4/22 • [Edit Online](#)

Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The package is not signed.

Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json':
signatureValidationMode is set to require, so packages are allowed only if signed by trusted
signers; however, this package is unsigned.

Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': This repository
indicated that all its packages are repository signed; however, this package is unsigned.

问题

如果来自 `nuget verify -signatures`

NuGet 客户端尝试验证未签名的包。

如果从还原或安装时指定 `signatureValidationMode` **到** `require`

`require` 验证模式不支持未签名的包和未签名的包尝试安装。

解决方案

请确保任何包用于安装或传递给 `nuget verify -signatures` 命令包含包的签名。

NuGet 错误 NU3005

2020/1/31 • [Edit Online](#)

方案 1

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The package contains an invalid package signature file.
```

问题

NuGet 客户端尝试使用具有无效本地文件头的签名文件验证包。

解决方案

如[NuGet 文档](#)中所述, 请使用 `nuget sign` 命令请求包作者对包进行重新签名。如果问题仍然存在, 请将[NuGet/Home](#)上的问题以及生成此问题的包引起。

方案 2

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The package contains multiple package signature files.
```

问题

NuGet 客户端尝试验证包含多个签名文件的包。

解决方案

如[NuGet 文档](#)中所述, 请使用 `nuget sign` 命令请求包作者对包进行重新签名。如果问题仍然存在, 请将[NuGet/Home](#)上的问题以及生成此问题的包引起。

方案3

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The package does not contain a valid package signature file.
```

问题

NuGet 客户端尝试验证不包含包签名文件的包。

解决方案

请在[NuGet/Home](#)上发布问题以及生成此问题的包。

NuGet 警告 NU3006

2019/4/23 • [Edit Online](#)

Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': Signed Zip64 packages are not supported.

问题

NuGet 客户端尝试验证 Zip64 包。

解决方案

NuGet 客户端不支持 Zip64 已签名的包。请确保正在验证的任何包不是 Zip64 包。你可以阅读更多有关中的 [Zip64 PKWARE Zip 规范](#)。

NuGet 警告 NU3007

2019/4/23 • [Edit Online](#)

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The package signature format version is not supported. Updating your client may solve this problem.
```

问题

NuGet 客户端尝试验证的包具有更高的包签名版本而不是此客户端支持的。

解决方案

请确保你使用最新版本的 NuGet 客户端。你可以阅读更多关于在版本[NuGet.org](#)。

NuGet 错误 NU3008

2019/4/23 • [Edit Online](#)

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The package integrity check failed.
```

问题

正在验证的 NuGet 包签名后发生更改。

解决方案

请确保包自签名以来未被篡改。如果这是临时问题，则可以解决此问题通过清除本地缓存，通过运行 `nuget locals -Clear all` 命令。但是，如果问题仍然存在然后，请告知包源和包的作者。

如果此问题发生的包的来源 [nuget.org](#) 然后请文件时出现问题 [NuGet/Home](#) 以及导致此问题的包。

NuGet 警告 NU3009

2020/1/31 • [Edit Online](#)

Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The package signature file does not contain exactly one primary signature.

问题

NuGet 客户端尝试验证包含多个 `SignerInfo` 字段的包签名。

解决方案

如[NuGet 文档](#)中所述, 请使用 `nuget sign` 命令请求包作者对包进行重新签名。如果问题仍然存在, 请将[NuGet/Home](#)上的问题以及生成此问题的包引起。

NOTE

当 NuGet 的[签名验证模式](#)设置为 `accept` (默认值)时, 将引发 NU3009 作为警告。当 NuGet 的签名验证模式设置为 "必需" 时, 或者在运行 `nuget verify -signatures` 命令时, NU3009 将从警告提升为错误。

NuGet 警告 NU3010

2020/1/31 • [Edit Online](#)

Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The primary signature does not have a signing certificate.

问题

NuGet 客户端尝试使用不包含签名证书的 `SignerInfo` 条目来验证包签名。

解决方案

如[NuGet 文档](#)中所述, 请使用 `nuget sign` 命令请求包作者对包进行重新签名。如果问题仍然存在, 请将[NuGet/Home](#)上的问题以及生成此问题的包引起。

NOTE

当 NuGet 的[签名验证模式](#)设置为 `accept` (默认值)时, 将引发 NU3010 作为警告。当 NuGet 的签名验证模式设置为 "必需" 时, 或者在运行 `nuget verify -signatures` 命令时, NU3010 将从警告提升为错误。

NuGet 警告 NU3011

2020/1/31 • [Edit Online](#)

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The primary signature is invalid.
```

问题

NuGet 客户端无法读取用于对包进行签名的签名证书的证书链。

解决方案

如[NuGet 文档](#)中所述, 请使用 `nuget sign` 命令请求包作者对包进行重新签名。如果问题仍然存在, 请将[NuGet/Home](#)上的问题以及生成此问题的包引起。

NOTE

当 NuGet 的[签名验证模式](#)设置为 `accept` (默认值)时, 将引发 NU3011 作为警告。当 NuGet 的签名验证模式设置为 "必需" 时, 或者在运行 `nuget verify -signatures` 命令时, NU3011 将从警告提升为错误。

NuGet 警告 NU3012

2020/1/31 • [Edit Online](#)

方案 1

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The primary signature validation failed.
```

问题

NuGet 客户端无法验证包中 NuGet 签名中存在的 `SignedCms` 签名。

解决方案

您可以通过查看调试日志来获取有关该问题的更多详细信息。如果问题仍然存在, 请将[NuGet/Home](#)上的问题以及生成此问题的包引起。

方案 2

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The primary signature found a chain building issue: A certificate chain processed, but terminated in a root certificate which is not trusted by the trust provider.
```

问题

NuGet 客户端无法验证用于对包进行签名的签名证书的证书链。

解决方案

请确保包签名具有有效的证书链。可以通过对包运行 `nuget verify -signatures` 命令来验证包签名。如果问题仍然存在, 请将[NuGet/Home](#)上的问题以及生成此问题的包引起。

NOTE

当 NuGet 的[签名验证模式](#)设置为 `accept` (默认值)时, 在大多数情况下, NU3012 将作为警告引发。当 NuGet 的签名验证模式设置为 "必需" 时, 或者在运行 `nuget verify -signatures` 命令时, NU3012 将从警告提升为错误。

NuGet 警告 NU3013

2020/1/31 • [Edit Online](#)

方案 1

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The signing certificate has an unsupported signature algorithm.
```

问题

用于对包进行签名的证书具有不受支持的签名算法。

解决方案

请确保签名证书具有以下签名算法之一-

- `sha256WithRSAEncryption`
- `sha384WithRSAEncryption`
- `sha512WithRSAEncryption`

方案 2

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The primary signature's certificate has an unsupported signature algorithm.
```

问题

用于对包进行签名的证书具有不受支持的签名算法。

解决方案

请确保已使用具有以下签名算法之一的证书对包进行签名-

- `sha256WithRSAEncryption`
- `sha384WithRSAEncryption`
- `sha512WithRSAEncryption`

NOTE

当 NuGet 的[签名验证模式](#)设置为 `accept` (默认值)时, 将引发 NU3013 作为警告。当 NuGet 的签名验证模式设置为 "必需" 时, 或者在运行 `nuget verify -signatures` 命令时, NU3013 将从警告提升为错误。

NuGet 警告 NU3014

2020/1/31 • [Edit Online](#)

方案 1

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The signing certificate does not meet a minimum public key length requirement.
```

问题

用于对包进行签名的证书不满足最小公钥长度要求。

解决方案

请确保签名证书具有长度为 ≥ 2048 位的 RSA 公钥。

方案 2

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The primary signature's certificate does not meet a minimum public key length requirement.
```

问题

用于对包进行签名的证书不满足最小公钥长度要求。

解决方案

请确保已使用签名证书对包进行签名，该签名证书的 RSA 公钥长度 ≥ 2048 位。

NOTE

当 NuGet 的[签名验证模式](#)设置为 accept (默认值)时，将引发 NU3014 作为警告。当 NuGet 的签名验证模式设置为 "必需" 时，或者在运行 `nuget verify -signatures` 命令时，NU3014 将从警告提升为错误。

NuGet 警告 NU3015

2020/1/31 • [Edit Online](#)

方案 1

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The lifetime signing EKU in the primary signature's certificate is not supported.
```

问题

用于对包进行签名的证书具有不受支持的扩展密钥用法。

解决方案

请确保签名证书没有生存期签名扩展密钥用法。

方案 2

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The lifetime signing EKU in the signing certificate is not supported.
```

问题

用于对包进行签名的证书具有不受支持的扩展密钥用法。

解决方案

请确保使用不具有生存期签名扩展密钥用法的签名证书对包进行签名。

NOTE

当 NuGet 的[签名验证模式](#)设置为 accept (默认值)时, 将引发 NU3015 作为警告。当 NuGet 的签名验证模式设置为 "必需" 时, 或者在运行 `nuget verify -signatures` 命令时, NU3015 将从警告提升为错误。

NuGet 警告 NU3016

2020/1/31 • [Edit Online](#)

Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The package hash uses an unsupported hash algorithm.

问题

NuGet 客户端尝试验证使用不受支持的哈希算法签名的包。

解决方案

请确保已用下列哈希算法之一对包进行了签名 -

- sha256
- sha384
- sha512

NOTE

当 NuGet 的[签名验证模式](#)设置为 accept (默认值)时, 将引发 NU3016 作为警告。当 NuGet 的签名验证模式设置为 "必需" 时, 或者在运行 `nuget verify -signatures` 命令时, NU3016 将从警告提升为错误。

NuGet 警告 NU3017

2020/1/31 • [Edit Online](#)

方案 1

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The signing certificate is not yet valid.
```

问题

用于对包进行签名的证书在将来有效，但目前无效。

解决方案

请确保签名证书当前有效。

方案 2

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The primary signature's certificate is not yet valid.
```

问题

用于对包进行签名的证书在将来有效，但目前无效。

解决方案

请根据使用当前有效的签名证书的[NuGet 文档](#)中所述，请求包作者对 `nuget sign` 包进行重新签名。

NOTE

当 NuGet 的[签名验证模式](#)设置为 `accept` (默认值)时，将引发 NU3017 作为警告。当 NuGet 的签名验证模式设置为 "必需" 时，或者在运行 `nuget verify -signatures` 命令时，NU3017 将从警告提升为错误。

NuGet 警告 NU3018

2020/1/31 • [Edit Online](#)

Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The primary signature found a chain building issue: A certificate chain processed, but terminated in a root certificate which is not trusted by the trust provider.

问题

NuGet 客户端无法验证用于对包进行签名的签名证书的证书链。

解决方案

请确保包签名具有有效的证书链。可以通过对包运行 `nuget verify -signatures` 命令来验证包签名。如果问题仍然存在, 请将[NuGet/Home](#)上的问题以及生成此问题的包引起。

NOTE

当 NuGet 的[签名验证模式](#)设置为 accept (默认值)时, 将引发 NU3018 作为警告。当 NuGet 的签名验证模式设置为 "必需" 时, 或者在运行 `nuget verify -signatures` 命令时, 在大多数情况下, NU3018 将从警告提升为错误。

NuGet 警告 NU3019

2020/1/31 • [Edit Online](#)

方案 1

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The timestamp integrity check failed.
```

问题

包签名上的时间戳在由时间戳颁发机构生成后已发生更改。

解决方案

请尝试对包重新签名并加盖时间戳。如果问题仍然存在，请与时间戳颁发机构联系，以发现问题的根源。

方案 2

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The primary signature's timestamp integrity check failed.
```

问题

包签名上的时间戳在由时间戳颁发机构生成后已发生更改。

解决方案

如[NuGet 文档](#)中所述，请使用 `nuget sign` 命令请求包作者对包进行重新签名。如果问题仍然存在，请请求包作者与时间戳机构联系，以发现问题的根源。

NOTE

运行 `nuget verify -signatures` 命令时，NU3019 将作为错误引发。否则，NU3019 将作为警告引发。

NuGet 警告 NU3020

2020/1/31 • [Edit Online](#)

方案 1

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The timestamp does not have a signing certificate.
```

问题

包签名上的时间戳不包含签名证书。

解决方案

请尝试对包重新签名并加盖时间戳。如果问题仍然存在，请与时间戳颁发机构联系，以发现问题的根源。

方案 2

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The primary signature's timestamp does not have a signing certificate.
```

问题

包签名上的时间戳不包含签名证书。

解决方案

如[NuGet 文档](#)中所述，请使用 `nuget sign` 命令请求包作者对包进行重新签名。如果问题仍然存在，请请求包作者与时间戳机构联系，以发现问题的根源。

NOTE

运行 `nuget verify -signatures` 命令时，NU3020 将作为错误引发。否则，NU3020 将作为警告引发。

NuGet 警告 NU3021

2020/1/31 • [Edit Online](#)

方案 1

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The primary signature's timestamp signature validation failed.
```

问题

NuGet 客户端无法验证包签名上时间戳内的 `SignedCms` 对象。

解决方案

如[NuGet 文档](#)中所述, 请使用 `nuget sign` 命令请求包作者对包进行重新签名。如果问题仍然存在, 请请求包作者与时间戳机构联系, 以发现问题的根源。

方案 2

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The timestamp signature validation failed.
```

问题

包签名中时间戳内的 `SignedCms` 对象无法进行验证。

解决方案

请尝试对包重新签名并加盖时间戳。如果问题仍然存在, 请与时间戳颁发机构联系, 以发现问题的根源。

NOTE

运行 `nuget verify -signatures` 命令时, NU3021 将作为错误引发。否则, NU3021 将作为警告引发。

NuGet 警告 NU3022

2020/1/31 • [Edit Online](#)

方案 1

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The primary signature's timestamp certificate has an unsupported signature algorithm.
```

问题

用于对包签名进行时间戳的证书具有不受支持的签名算法。

解决方案

请确保时间戳颁发机构的签名证书具有以下签名算法之一-

- sha256WithRSAEncryption
- sha384WithRSAEncryption
- sha512WithRSAEncryption

方案 2

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The timestamp certificate has an unsupported signature algorithm (SHA1). The following algorithms are supported: SHA256RSA, SHA384RSA, SHA512RSA.
```

问题

用于对包签名进行时间戳的证书具有不受支持的签名算法。

解决方案

请按照使用 `-Timestamper` 选项的[NuGet 文档](#) 中所述, 请求包作者对 `nuget sign` 包进行重新签名, 使时间戳颁发机构签名证书具有以下签名算法之一-

- sha256WithRSAEncryption
- sha384WithRSAEncryption
- sha512WithRSAEncryption

NOTE

运行 `nuget verify -signatures` 命令时, NU3022 将作为错误引发。否则, NU3022 将作为警告引发。

NuGet 警告 NU3023

2019/4/23 • [Edit Online](#)

方案 1

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The timestamp certificate does not meet a minimum public key length requirement.
```

问题

证书使用时间戳执行包签名不符合最小公钥长度要求。

解决方案

请确保时间戳颁发机构的签名证书具有 RSA 公钥的长度 \geq 2048 位。

方案 2

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The primary signature's timestamp certificate does not meet a minimum public key length requirement.
```

问题

证书使用时间戳执行包签名不符合最小公钥长度要求。

解决方案

请确保包的签名是使用具有长度的 RSA 公钥签名证书的时间戳的 \geq 2048 位。

NOTE

运行时 `nuget verify -signatures` 命令, NU3023 引发错误。否则, NU3023 引发作为警告。

NuGet 警告 NU3024

2020/1/31 • [Edit Online](#)

方案 1

Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The timestamp signature has an unsupported digest algorithm. The following algorithms are supported: : SHA-2-256, SHA-2-384, SHA-2-512.

问题

时间戳的签名具有不支持的摘要算法。

解决方案

确保时间戳颁发机构的签名具有以下摘要算法之一-

- SHA-2-256
- SHA-2-384
- SHA-2-512

方案 2

Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The primary signature's timestamp signature has an unsupported digest algorithm.

问题

时间戳的签名具有不支持的摘要算法。

解决方案

请求包作者使用 `-Timestamper` 选项的 [NuGet 文档](#) 中所述的 `nuget sign` 命令对包进行重新签名，使时间戳颁发机构签名证书具有以下摘要算法之一-

- SHA-2-256
- SHA-2-384
- SHA-2-512

NOTE

运行 `nuget verify -signatures` 命令时，NU3024 将作为错误引发。否则，NU3024 将作为警告引发。

NuGet 警告 NU3025

2020/1/31 • [Edit Online](#)

方案 1

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The timestamp signing certificate is not yet valid.
```

问题

用于给包签名加盖时间戳的证书在将来具有有效性，但目前无效。

解决方案

请确保时间戳颁发机构的签名证书当前有效。如果问题仍然存在，请与时间戳颁发机构联系，以发现问题的根源。

方案 2

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The primary signature's timestamp signing certificate is not yet valid.
```

问题

用于对包进行签名的证书在将来有效，但目前无效。

解决方案

如[NuGet 文档](#)中所述，请使用 `nuget sign` 命令请求包作者对包进行重新签名。如果问题仍然存在，请请求包作者与时间戳机构联系，以发现问题的根源。

NOTE

运行 `nuget verify -signatures` 命令时，NU3025 将作为错误引发。否则，NU3025 将作为警告引发。

NuGet 警告 NU3026

2019/4/23 • [Edit Online](#)

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The timestamp response is invalid. Nonces did not match.
```

问题

时间戳颁发机构未返回预期的 nonce 其响应;因此，其响应是无效的。

解决方案

尝试重新签名和时间戳包。如果问题仍然存在，请联系的时间戳颁发机构来发现问题的原因。

NuGet 警告 NU3027

2020/1/31 • [Edit Online](#)

Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The signature should be timestamped to enable long-term signature validity after the certificate has expired.

问题

包签名不包含时间戳。

解决方案

对于长期签名有效性, [请使用 "-Timestamper"](#) 选项, 请求包作者使用 `nuget sign` 命令对包进行重新签名。

NuGet 警告 NU3028

2020/1/31 • [Edit Online](#)

NuGet 4.6.0+

The author primary signature's timestamp found a chain building issue: The revocation function was unable to check revocation because the revocation server could not be reached. For more information, visit <https://aka.ms/certificateRevocationMode>

问题

用于生成时间戳签名的证书链生成失败。时间戳签名证书不受信任、被吊销或吊销信息不可用。

解决方案

使用可信证书和有效证书。检查 internet 连接。位数

吊销检查模式 (4.8.1 +)

如果计算机具有受限的 internet 访问权限(例如 CI/CD 方案中的生成计算机), 安装/还原已签名 nuget 包将导致此警告, 因为吊销服务器不可访问。这是预期情况。但是, 在某些情况下, 这可能会产生意外的结果, 如安装/还原包花费的时间比平常长。如果发生这种情况, 可以通过将 `NUGET_CERT_REVOCATION_MODE` 环境变量设置为 `offline` 来解决该问题。这将强制 NuGet 仅针对缓存的证书吊销列表检查证书的吊销状态, NuGet 不会尝试访问吊销服务器。

WARNING

不建议在正常情况下将吊销检查模式切换到脱机。执行此操作将导致 NuGet 跳过联机吊销检查, 并只对可能过期的缓存证书吊销列表执行脱机吊销检查。这意味着将继续安装/还原签名证书可能已经吊销的包, 否则吊销检查将失败, 并且不会安装。

如果吊销检查模式设置为 `offline`, 警告将降级为信息。

The author primary signature's timestamp found a chain building issue: The revocation function was unable to check revocation because the certificate is not available in the cached certificate revocation list and `NUGET_CERT_REVOCATION_MODE` environment variable has been set to `offline`. For more information, visit <https://aka.ms/certificateRevocationMode>.

NOTE

大多数情况下, NU3028 将作为错误引发。如果 NuGet 的 [签名验证模式](#) 设置为 `accept` (默认值), 则在某些情况下, NU3028 将作为警告引发。

NuGet 警告 NU3029

2019/4/23 • [Edit Online](#)

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The timestamp  
signature is invalid.
```

问题

时间戳签名无效。

解决方案

尝试重新签名和时间戳有效的时间戳的包。如果问题仍然存在, 请联系的时间戳颁发机构来发现问题的原因。

NuGet 警告 NU3030

2020/1/31 • [Edit Online](#)

Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The primary signature's timestamp's message imprint uses an unsupported hash algorithm.

问题

主签名的 timestamp's 消息使用了不支持的哈希算法。

解决方案

请求包作者使用 `nuget sign` 命令(如使用 `-Timestamper` 选项的[NuGet 文档](#)中所述)对包进行重新签名, 以便时间戳的消息标记使用以下哈希算法之一-

- `SHA-2-256`
- `SHA-2-384`
- `SHA-2-512`

NOTE

运行 `nuget verify -signatures` 命令时, NU3030 将作为错误引发。否则, NU3030 将作为警告引发。

NuGet 警告 NU3031

2019/4/23 • [Edit Online](#)

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The repository  
countersignature is invalid.
```

问题

存储库副署无效。

解决方案

请联系副署包的存储库。

NuGet 警告 NU3032

2019/4/23 • [Edit Online](#)

方案 1

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The package already contains a repository countersignature. Please remove the existing signature before adding a new repository countersignature.
```

问题

已签名的包不能包含多个存储库副署。在尝试添加新的存储库副署时，该包已包含存储库副署。

解决方案

添加新的存储库副署之前删除现有的签名。

方案 2

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The package signature contains multiple repository signatures.
```

问题

包的签名包含多个存储库副署。

解决方案

请联系副署包的存储库。

NuGet 警告 NU3033

2019/4/23 • [Edit Online](#)

方案 1

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': A repository primary signature must not have a repository countersignature.
```

问题

主签名应为作者签名或存储库签名。存储库主签名不能具有存储库副署。

解决方案

尝试重新签名的包的作者签名或存储库签名作为主签名，然后重新进行副署包。

方案 2

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': A repository primary signature must not have a repository countersignature.
```

问题

主签名应为作者签名或存储库签名。存储库主签名不能具有存储库副署。

解决方案

请联系副署包的存储库。

NuGet 错误 NU3034

2019/4/23 • [Edit Online](#)

Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json':
signatureValidationMode is set to require, so packages are allowed only if signed by trusted
signers; however, no trusted signers were specified.

Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The package signature
certificate fingerprint does not match any certificate fingerprint in the allow list.

Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': This repository
indicated that all its packages are repository signed; however, it listed no signing certificates.

Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': This package was not
repository signed with a certificate listed by this repository.

问题

没有缺少允许列表，或包签名者与列表中的任何签名者不匹配。此列表可能会发送由存储库或中指定
`trustedSigners` 一部分 `nuget.config`。

解决方案

如果在 `require` 模式下，只有受信任的签名者的包将通过验证。否则，请联系，这从下载的让他们知道它们具有不符合存储库签名安全指导原则的包的存储库。

NuGet 警告 NU3035

2019/4/23 • [Edit Online](#)

问题

为存储库副署的签名证书，可以生成完整的证书链。

解决方案

联系时间戳颁发机构来发现问题的原因。

NuGet 警告 NU3036

2020/1/31 • [Edit Online](#)

方案 1

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The timestamp's
generalized time is outside the timestamping certificate's validity period.
```

问题

时间戳的一般化时间超出了时间戳证书的有效期。

解决方案

尝试对包进行重新签名并加盖时间戳。如果问题仍然存在，请与时间戳颁发机构联系，以发现问题的根源。

方案 2

```
Package 'SamplePackage v1.0.0' from source 'https://contoso.com/index.json': The primary
signature's timestamp's generalized time is outside the timestamping certificate's validity
period.
```

问题

用于对包签名进行时间戳的证书无效，因为时间戳的一般化时间超出了时间戳证书的有效期。

解决方案

如[NuGet 文档](#)中所述，请求包作者使用 `nuget sign` 命令对包进行重新签名并加盖时间戳。如果问题仍然存在，请请求包作者与时间戳机构联系，以发现问题的根源。

NOTE

运行 `nuget verify -signatures` 命令时，NU3036 将作为错误引发。否则，NU3036 将作为警告引发。

NuGet 警告 NU3037

2020/1/31 • [Edit Online](#)

问题

NuGet 包签名已过期。包签名与用于生成签名的证书共享同一有效期。包签名在该有效期之外无效。若要确保长期有效---甚至超出签名证书的有效期---应使用受信任时间戳为包签名加盖时间戳。如果包签名仍有效且未过期，则必须添加可信时间戳。

解决方案

- 使用未过期的证书放弃包。(可选)在签名时添加受信任的时间戳，以确保签名的长期有效。
- 如果仅用于接受模式，则忽略此警告。

NOTE

当 NuGet 的[签名验证模式](#)设置为 "接受" (默认值)时，具有过期包签名的包将被视为未签名的包，并且仍将安装。NU3037 将作为警告引发。当 NuGet 的签名验证模式设置为 "必需" 时，或者在运行 `nuget verify -signatures` 命令时，NU3037 将从警告提升为错误。

NuGet 警告 NU3038

2019/4/23 • [Edit Online](#)

问题

验证设置需要存储库副署，但包没有存储库副署。

解决方案

使用副署的存储库包的包源。

NuGet 警告 NU3040

2018/11/28 • [Edit Online](#)

There are two certificates with conflicting allowUntrustedRoot attributes in the computed settings. The allowUntrustedRoot attribute is going to be set to false. Certificate: SHA256-3F9001EA83C560D712C24CF213C3D312CB3BFF51EE89435D3430BD06B5D0EECE

问题

冲突的属性中有证书项 nuget.config 中。证书的两个项共用同一个 `fingerprint` 并 `hashAlgorithm`，但具有不同 `allowUntrustedRoot`。

解决方案

NuGet 需要这些设置的最具限制性 (`allowUntrustedRoot=false`)，用于删除警告，请确保重复数据删除的证书项，或者设置 `allowUntrustedRoot` 为上都相同的值。

NuGet 错误 NU5000

2018/9/5 • [Edit Online](#)

问题

NuGet 包操作时出现问题。我们尚未尚未分配唯一的错误代码针对该问题时，使用 NU5000。可以帮助我们改进，请随时提出此问题与您的错误的详细信息。

解决方案

检查输出窗口（在 Visual Studio）或控制台输出（通过 NuGet 或 dotnet 命令行工具）有关详细信息。

NuGet 错误 NU5001

2018/9/5 • [Edit Online](#)

```
Unable to output resolved nuspec file because it would overwrite the original at  
'F:\project\project.nuspec'.
```

问题

调用 NuGet 包操作时使用了 `-InstallPackageToOutputPath` 选项，但输出路径已包含 nuspec 文件。

解决方案

请确保输出路径传递到 NuGet pack 命令尚未包含 nuspec 文件。

NuGet 错误 NU5002

2018/9/5 • [Edit Online](#)

Please specify a nuspec, project.json, or project file to use.

问题

为 NuGet 包操作不提供的任何输入的文件。

解决方案

请使用 nuspec、project.json 或项目文件路径到 pack 命令作为参数。

NuGet 错误 NU5003

2018/9/5 • [Edit Online](#)

```
Failed to build package because of an unsupported targetFramework value on 'System.Net'.
```

问题

Nuspec 文件中指定的 framework 程序集不包含有效框架。

解决方案

请修复中引用的程序集指定的目标框架。

NuGet 错误 NU5004

2018/9/5 • [Edit Online](#)

```
Failed to build package. Ensure 'F:\project\project.nuspec' includes assembly files. For help on building symbols package, visit http://docs.nuget.org/.
```

问题

使用 NuGet 包操作已执行 `-Symbol` 选项，但包含 nuspec 文件的文件夹包含任何程序集文件。

解决方案

在生成符号包时，请确保包含 nuspec 文件的文件夹包含符号需要打包的程序集文件。

NuGet 错误 NU5005

2018/9/5 • [Edit Online](#)

Ensure 'F:\project\project.nuspec' includes source and symbol files. For help on building symbols package, visit <http://docs.nuget.org/>.

问题

调用 NuGet 包操作时使用了 `-Symbols` 选项，但 nuspec 文件不包含任何源或符号文件。

解决方案

在生成符号包时，请确保包含 nuspec 文件的文件夹包含符号需要打包的程序集文件。

NuGet 错误 NU5007

2018/9/5 • [Edit Online](#)

```
No build found in F:\project\bin\Debug\net461\project.dll. Use the -Build option or build the project.
```

问题

调用 NuGet 包操作时使用了 `-Symbols` 选项，但要打包的项目尚未生成，因此不能打包。

解决方案

请运行 NuGet 包操作之前生成项目或使用 `-Build` 选项来打包前生成项目。

NuGet 错误 NU5008

2018/9/5 • [Edit Online](#)

```
Manifest file not found at 'F:\project\project.nuspec'
```

问题

传递到 NuGet 包操作的 nuspec 文件不存在。

解决方案

请确保传递到 NuGet pack 命令的 nuspec 文件存在磁盘上。

NuGet 错误 NU5009

2018/9/5 • [Edit Online](#)

```
Cannot find version of msbuild.
```

问题

NuGet 包操作找不到 msbuild。

解决方案

请传递 `-MsBuildPath` 到 NuGet pack 命令。

NuGet 错误 NU5010

2018/9/5 • [Edit Online](#)

```
Version string specified for package reference '9.9.9.9.9' is invalid.
```

问题

提供给 NuGet 包操作版本字符串不是有效的字符串。

解决方案

请确保传递到 NuGet 包操作版本字符串是有效的 SemVer2 字符串。

NuGet 错误 NU5011

2018/9/5 • [Edit Online](#)

Unable to extract metadata from 'project.csproj'.

问题

从程序集或项目文件中提取元数据时出现运行时异常。

解决方案

请创建时出现问题 <https://github.com/NuGet/Home/issues>。

NuGet 错误 NU5012

2018/9/5 • [Edit Online](#)

```
Unable to find 'F:\project\bin\Debug\net461\project.dll'. Make sure the project has been built.
```

问题

被打包项目尚未生成，因此不能进行打包。

解决方案

请运行 NuGet 包操作之前生成项目或使用 `-Build` 选项来打包前生成项目。

NuGet 错误 NU5013

2018/9/5 • [Edit Online](#)

```
Failed to build 'project.csproj'
```

问题

项目未能运行使用 NuGet 包操作时生成 `-Build` 选项。

解决方案

请在项目中修复生成错误，然后重试。应在控制台日志中显示失败的原因。

NuGet 错误 NU5014

2018/9/5 • [Edit Online](#)

```
Error occurred when processing file 'F:\project2\project2.csproj': The 'id' start tag on line 4  
position 10 does not match the end tag of 'ids'. Line 4, position 20.
```

问题

NuGet 包操作已运行与 `-IncludeReferencedProjects` 选项和读取与引用的项目的 nuspec 文件时出错。

解决方案

请修复 nuspec 错误根据错误消息中列出的详细信息。

NuGet 错误 NU5015

2018/9/5 • [Edit Online](#)

```
project.json cannot contain multiple Target Frameworks.
```

问题

传递到 NuGet 包操作的 project.json 文件包含多个目标框架。

解决方案

请修复将包含单个目标框架的 project.json 文件。

NuGet 错误 NU5016

2018/9/5 • [Edit Online](#)

Package version constraints for 'NuGet.Versioning' return a version range that is empty.

问题

项目具有的包依赖项，它不具有有效的版本。

解决方案

请修复要符合 semver 的依赖项版本。

NuGet 错误 NU5017

2018/9/5 • [Edit Online](#)

```
Cannot create a package that has no dependencies nor content.
```

问题

正在保存的包不包含任何包引用、程序集或框架引用。

解决方案

请确保所创建的包包含程序集或包引用。

NuGet 错误 NU5018

2018/9/5 • [Edit Online](#)

```
Invalid assembly reference 'xunit.dll'. Ensure that a file named 'xunit.dll' exists in the lib directory.
```

问题

用于创建 NuGet 包操作包含的 nuspec 文件 `references` 中不存在的 `.\lib` 项目目录中的文件夹。

解决方案

请确保将任何引用的程序集放在 `.\lib` 文件夹所在的文件夹 nuspec 中用来创建包文件。

NuGet 错误 NU5019

2018/9/5 • [Edit Online](#)

```
File not found: 'bad_file.path'
```

问题

用于创建 NuGet 包操作包含的 nuspec 文件 `files` 不存在。

解决方案

请确保任何 `file` 中的条目 `files` nuspec 文件中的元素将位于指定为路径 `src`。

NuGet 错误 NU5020

2018/11/28 • [Edit Online](#)

A source file was added with an empty path.

问题

已将源文件传递到 `msbuild -t:pack` 带有空路径的操作。

解决方案

请确保所有源文件传递给 `msbuild -t:pack` 操作具有有效路径，并在磁盘上存在。

NuGet 错误 NU5021

2018/9/5 • [Edit Online](#)

The project directory for the source file 'src/Project/Code.cs' could not be found.

问题

调用 NuGet 包操作时使用了 `-Symbols` 未找到选项，但源文件的项目目录。

解决方案

请确保源文件 idsk 上存在。否则请文件时出现问题[NuGet/主页](#)

NuGet 错误 NU5022

2018/11/28 • [Edit Online](#)

MinClientVersion 属性传递给 msbuild t: 包操作不是有效的版本字符串。

问题

`MinClientVersion` 属性传递给 msbuild t: 包操作不是有效的版本字符串。

解决方案

请修复作为传递的版本字符串 `MinClientVersion` 是有效的 SemVer 版本。

NuGet 错误 NU5023

2018/11/28 • [Edit Online](#)

The assets file produced by restore does not exist. Try restoring the project again. The expected location of the assets file is F:\project\obj\project.assets.json.

问题

项目传递给 `msbuild -t:pack` 不包含 obj 目录中的资产文件。

解决方案

请运行 `msbuild -t:restore` 运行包操作之前打包项目上的操作。

NuGet 错误 NU5024

2018/11/28 • [Edit Online](#)

```
PackageVersion string specified '9.9.9.9.9' is invalid.
```

问题

`PackageVersion` 属性传递给 msbuild t: 包操作不是有效的版本字符串。

解决方案

请修复作为传递的版本字符串 `PackageVersion` 是有效的 SemVer 版本。

NuGet 错误 NU5025

2018/11/28 • [Edit Online](#)

The assets file found does not contain a valid package spec. Try restoring the project again. The location of the assets file is F:\project\obj\project.assets.json.

问题

项目传递给 `msbuild -t:pack` 不包含 obj 目录中的有效资产文件。

解决方案

请运行 `msbuild -t:restore -p:restoreforce=true` 运行包操作之前打包项目上的操作。

NuGet 错误 NU5026

2019/8/17 • [Edit Online](#)

```
The file ''F:\project\bin\Debug\net461\project.exe' to be packed was not found on disk.
```

问题

尚未生成要打包的项目，因此无法将其打包。

解决方案

请在运行 `dotnet pack` 操作之前生成项目，或者不使用 `--no-build` 选项允许 `dotnet pack` 在打包之前生成项目。

您可能已编写了一个不输出程序集的项目。如果打算交付无程序集的 NuGet 包，请禁用 `dotnet pack` 对输出程序集的要求。为此，可以在项目文件 `IncludeBuildOutput` 中将 `false` 属性设置为。

另请参阅[相关 mbuild 属性](#)。

NuGet 错误 NU5027

2018/9/5 • [Edit Online](#)

```
Invalid target framework for the file 'F:\project\project.dll'.
```

问题

打包程序集不包含有效的目标 framework 元数据。

解决方案

请确保要打包的程序集包含一个有效 `TargetFramework` 属性。

NuGet 错误 NU5028

2018/11/28 • [Edit Online](#)

No project was provided to the PackTask.

问题

没有项目文件指定给 `msbuild -t:pack` 操作。

解决方案

请指定要打包操作的项目。您可以指定它在 command (`msbuild -t:pack project.csproj`) 或运行 `msbuild -t:pack` 包含项目文件的文件夹中的操作。

NuGet 错误 NU5029

2018/11/28 • [Edit Online](#)

NuspecProperties should be in the form of 'key1=value1;key2=value2'.

问题

属性传递给 `msbuild -t:pack -p:NuspecFile=project.nuspec` 命令未采用正确的格式进行分析。

解决方案

请传递任何 nuspec 属性使用 `-p:NuspecProperties` 采用的格式 `key=value`。例如

```
msbuild -t:pack -p:NuspecFile=project.nuspec -p:NuspecProperties="configuration=release"
```

NuGet 错误 NU5030

2019/12/5 • [Edit Online](#)

```
The license file 'LICENSE.txt' does not exist in the package.
```

问题

此许可证文件在元数据中引用，`PackageLicenseFile` 中的任意一个是 `.csproj` 或 `nuspec` 中的 `license` 元素，但文件本身并未包含在包中的预期位置。

解决方案

将文件包括在包中，例如：

如果包包含目标：

```
<PropertyGroup>
    <PackageLicenseFile>LICENSE.txt</PackageLicenseFile>
</PropertyGroup>

<ItemGroup>
    <None Include="licenses\LICENSE.txt" Pack="true" PackagePath="" />
</ItemGroup>
```

如果使用 `nuspec` 打包：

```
<package>
    <metadata>
        <license type="file">LICENSE.txt</license>
    </metadata>
    <files>
        <file src="licenses\LICENSE.txt" target="" />
    </files>
</package>
```

NuGet 错误 NU5031

2018/11/28 • [Edit Online](#)

```
The license file 'LICENSE.txt' has an invalid extension. Valid options are .txt, .md or none.
```

问题

允许仅列出的文件扩展。

解决方案

使用文件扩展名从允许列表。

NuGet 错误 NU5032

2019/12/5 • [Edit Online](#)

The license expression 'MIT OR OR Apache-2.0' cannot be parsed successfully. The license expression is invalid.

问题

许可证表达式不符合 NuGet 许可证表达式语法。

解决方案

在这种情况下，有2个 "OR" 运算符。运算符必须用操作数据括起来。例如：

```
MIT OR Apache-2.0
```

NuGet 错误 NU5033

2018/11/28 • [Edit Online](#)

Invalid metadata. Cannot specify both a License Expression and a License File.

问题

PackageLicenseExpression 和 PackageLicenseFile 属性互斥。

解决方案

使用一个或另一个。如果使用的是标准许可证列表的一部分的知名许可证，请考虑使用一个表达式。否则请使用的许可证文件。

NuGet 错误 NU5034

2018/12/4 • [Edit Online](#)

The PackageLicenseExpressionVersion string '2.0.0-InvalidSystemVersion' is not a valid version.

问题

PackageLicenseExpressionVersion 字符串是 System.Version。这意味着允许没有标签。

解决方案

将属性值更改为

NuGet 错误 NU5035

2018/11/28 • [Edit Online](#)

The PackageLicenseUrl cannot be used in conjunction with the PackageLicenseFile and PackageLicenseExpression.

问题

使用 PackageLicenseFile 或 PackageLicenseExpression 时，不应将 PackageLicenseUrl 属性进行设置。LicenseUrl 元数据将由低级别友好的 url 的客户端工具自动生成。

解决方案

未设置 PackageLicenseUrl。

NuGet 错误 NU5036

2019/1/16 • [Edit Online](#)

This package has an improperly escaped Url in LicenseUrl

问题

应正确转义 LicenseUrl nuspec 文件中的元数据值。某些版本的 nuget 包功能有问题将尽快更新。

解决方案

使用 nuget 包功能的修复的版本：

- "NuGet.exe 包"-在 4.9.1 中修复
- "dotnet.exe 包"-在 2.1.500 中无效。没有可用的修补程序尚未。(请勿使用新的许可证表达式或文件功能, 以避免出现问题。)
- "msbuild /t: pack"-在 Visual Studio 15.9.1 中无效。没有可用的修补程序尚未。(请勿使用新的许可证表达式或文件功能, 以避免出现问题。)

NuGet 错误 NU5037

2019/12/6 • [Edit Online](#)

The package is missing the required nuspec file.

问题

当源中的包或包的全局包文件夹中缺少[nuspec](#)文件时，还原将失败。

方案 1

包(nupkg)文件中缺少 nuspec 文件。

解决方案

请与包作者联系。

方案 2

对于使用 `PackageReference` 格式管理的项目，还原将失败。例如：

The package is missing the required nuspec file. Path: C:\..\..\nuget\packages\x\1.0.0.'x' represents package name and '1.0.0' represents package version.

解决方案

删除错误消息中指定的包文件夹，然后再次运行还原。例如：

Consider sample error message specified in the above section.
Delete the package folder '1.0.0' from 'C:\..\..\nuget\packages\x' directory and run the restore again.

场景 3

对于使用 `packages.config` 格式管理的项目，还原将失败。

解决方案

识别缺少 nuspec 文件的包需要手动探测依赖项。如果缺少 nuspec 文件的包是未知的，则删除已损坏的包文件夹或整个解决方案包文件夹，然后再次运行还原。

NuGet 错误 NU5046

2019/9/6 • [Edit Online](#)

The icon file 'icon.png' does not exist in the package.

问题

NuGet 找不到包中的图标文件。

解决方案

- 请确保在源中存在标记为包图标的文件，该文件是可读的，并且目标与 `icon` 属性所需的路径匹配。
- 确保在 nuspec 或项目文件中引用该文件。
 - 从 MSBuild 项目文件创建包时，请确保引用项目中的图标文件，如下所示：

```
<Project Sdk="Microsoft.NET.Sdk">
<PropertyGroup>
    ...
    <PackageIcon>icon.png</PackageIcon>
    ...
</PropertyGroup>

<ItemGroup>
    ...
    <None Include="images\icon.png" Pack="true" PackagePath="" />
    ...
</ItemGroup>
</Project>
```

- 从 nuspec 文件创建包时，请确保在 `<files/>` 部分中包含图标文件：

```
<package>
<metadata>
    ...
    <icon>images\icon.png</icon>
    ...
</metadata>
<files>
    ...
    <file src="..\icon.png" target="images\" />
    ...
</files>
</package>
```

[了解有关打包图标图像文件的详细信息。](#)

NuGet 错误 NU5047

2019/9/6 • [Edit Online](#)

The icon file size must not exceed 1 megabyte.

问题

指定为包图标的文件大于 1 mb。NuGet 仅允许其文件大小小于 1 MB 的图标。

解决方案

使用图像编辑器程序来减小包图标文件的大小。

NuGet 警告 NU5048

2019/9/10 • [Edit Online](#)

The 'PackageIconUrl'/'iconUrl' element is deprecated. Consider using the 'PackageIcon'/'icon' element instead. Learn more at <https://aka.ms/deprecateIconUrl>

问题

在 NuGet 包中嵌入图标时，图标 URL 已被弃用。可能的原因有：

- 从 nuspec 文件创建包时，它包含一个 `<iconUrl/>` 条目。
- 从 MSBuild 项目文件创建包时，它包含 `<PackageIconUrl>` 属性。

解决方案

若要停止看到此警告，请将嵌入的图标添加到包。

对于 MSBuild 项目文件，请添加 `<PackageIcon/>` 属性，如下所示：

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    ...
    <PackageIcon>icon.png</PackageIcon>
    ...
  </PropertyGroup>

  <ItemGroup>
    ...
    <None Include="images\icon.png" Pack="true" PackagePath="" />
    ...
  </ItemGroup>
</Project>
```

对于 "nuspec 文件"，`<icon/>` 添加指向将作为包图标的文件的条目：

```
<package>
  <metadata>
    ...
    <icon>images\icon.png</icon>
    ...
  </metadata>
  <files>
    ...
    <file src="..\icon.png" target="images\" />
    ...
  </files>
</package>
```

[了解有关打包图标图像文件的详细信息。](#)

NuGet 警告 NU5100

2019/8/15 • [Edit Online](#)

The assembly 'bin\Debug\net461\project.dll' is not inside the 'lib' folder and hence it won't be added as a reference when the package is installed into a project. Move it into the 'lib' folder if it needs to be referenced.

问题

正在打包的文件夹包含不在文件夹中 `lib` 的程序集文件。

解决方案

打包文件夹时, 请确保将所有程序集文件放在 `lib` 文件夹下的特定于框架的文件夹中。有关文件夹结构的详细信息, 请参阅[框架版本文件夹结构](#)。

NuGet 警告 NU5101

2020/1/31 • [Edit Online](#)

方案 1

The assembly 'lib\project.dll' is placed directly under 'lib' folder. It is recommended that assemblies be placed inside a framework-specific folder. Move it into a framework-specific folder.

问题

正在打包的文件夹包含直接位于 `lib` 文件夹下的程序集文件。

解决方案

打包文件夹时, 请确保将所有程序集文件放在 `lib` 文件夹下特定于框架的文件夹中。

方案 2

The assembly 'lib\project.dll' will be ignored when the package is installed after the migration.

问题

包中包含直接位于 `lib` 文件夹下的程序集文件。包将不与项目的包引用类型兼容。

解决方案

请请求包作者修复包, 以便将所有程序集放在 `lib` 文件夹下特定于框架的文件夹中。有关详细信息, 请参阅 [NuGet 文档](#)。

NuGet 警告 NU5102

2018/9/5 • [Edit Online](#)

The value "http://project_url_here_or_delete_this_line/" for ProjectUrl is a sample value and should be removed. Replace it with an appropriate value or remove it and rebuild your package.

问题

Nuspec 属性已使用默认值。

解决方案

请更改指定的属性的值。

NuGet 警告 NU5103

2018/9/5 • [Edit Online](#)

The folder 'lib\random_tfm\temp.dll' under 'lib' is not recognized as a valid framework name or a supported culture identifier. Rename it to a valid framework name or culture identifier.

问题

检测到程序集的 lib 文件夹下的无效的目标框架或区域性标识符文件夹下。

解决方案

请重命名文件夹到有效框架名称或区域性标识符。

NuGet 警告 NU5104

2018/9/5 • [Edit Online](#)

A stable release of a package should not have a prerelease dependency. Either modify the version spec of dependency "NuGet.Versioning [4.7.0-preview4.5065,)" or update the version field in the nuspec.

问题

项目或打包的 nuspec 包含预发行包上的依赖项。

解决方案

如果你想要创建预发行包，然后请 SemVer2 准则，请参阅并[将预发行版标记即添加到版本属性](#)

`<version>1.0.0-pre</version>`。如果你想要创建稳定的包，然后请更新指定的依赖项版本为稳定版本。

NuGet 警告 NU5105

2019/9/25 • • [Edit Online](#)

The package version '1.2.3+semver2.metadata' uses SemVer 2.0.0 or components of SemVer 1.0.0 that are not supported on legacy clients. Change the package version to a SemVer 1.0.0 string. If the version contains a release label it must start with a letter. This message can be ignored if the package is not intended for older clients.

问题

指定 `version` 的属性包含 SemVer 2.0.0 组件或 SemVer 1.0.0 组件，这些组件在之前的 NuGet 客户端上可能不受支持。

解决方案

如果你希望你的包可供版本低于 v4.0 的 NuGet 客户端使用，请修改 `version` 属性，使其与旧版客户端兼容。可[在此处](#)阅读有关 SemVer 2.0.0 支持的详细信息。

NuGet 警告 NU5106

2018/9/5 • [Edit Online](#)

The file at 'lib\WinRT\temp.dll' uses the obsolete 'WinRT' as the framework folder. Replace 'WinRT' or 'WinRT45' with 'NetCore45'.

问题

检测到文件中进行 `WinRT` 或 `WinRT45` lib 文件夹下的目标框架文件夹。但是, `WinRT` 和 `WinRT45` 目标框架现已过时。

解决方案

请重命名目标框架文件夹到 `NetCore45`。

NuGet 警告 NU5107

2018/9/5 • [Edit Online](#)

The file 'tools/subfolder/init.ps1' will be ignored by NuGet because it is not directly under 'tools' folder. Place the file directly under 'tools' folder.

问题

`init.ps1` 下的文件夹中检测到文件 `tools` 文件夹。

解决方案

请将放 `init.ps1` 文件直接下 `tools` 它是与 NuGet 工具兼容的文件夹。

NuGet 警告 NU5108

2018/9/5 • [Edit Online](#)

The transform file 'other\code.pp' is outside the 'content' folder and hence will not be transformed during installation of this package. Move it into the 'content' folder.

问题

.pp 或 .transform 检测到文件的文件夹中而不 content 文件夹。

解决方案

请将放 .pp 或 .transform 文件下 content 它是与 NuGet 工具兼容的文件夹。

NuGet 警告 NU5109

2018/9/5 • [Edit Online](#)

The file at 'tools/_.' uses the symbol for empty directory '_.', but it is present in a directory that contains other files. Please remove this file from directories that contain other files.

问题

具有文件名称的文件 `_.` 检测到包含其他文件的文件夹中。

解决方案

文件名称。用作空的目录的符号。具有该文件名称的任何文件应会出现在一个空的目录。请从包含其他文件的目录，删除此文件。

NuGet 警告 NU5110

2018/9/5 • [Edit Online](#)

The script file 'other\init.ps1' is outside the 'tools' folder and hence will not be executed during installation of this package. Move it into the 'tools' folder.

问题

.ps1 检测到文件的文件夹中而不 tools 文件夹。

解决方案

请将放 .ps1 文件下 tools 它是与 NuGet 工具兼容的文件夹。

NuGet 警告 NU5111

2018/9/5 • [Edit Online](#)

The script file 'tools\random.ps1' is not recognized by NuGet and hence will not be executed during installation of this package. Rename it to install.ps1, uninstall.ps1 or init.ps1 and place it directly under 'tools'.

问题

无法识别 `.ps1` 检测到文件打包。

解决方案

请重命名的文件 `init.ps1` 并将其直接下放置 `tools` 它是与 NuGet 工具兼容的文件夹。

NuGet 警告 NU5112

2018/9/5 • [Edit Online](#)

The version of dependency 'NuGet.Versioning' is not specified. Specify the version of dependency and rebuild your package.

问题

依赖项中指定不带版本的 nuspec 文件即- `<dependency id="NuGet.Versioning" />`

解决方案

请将版本添加到指定打包 nuspec 中的所有依赖项。

NuGet 警告 NU5114

2018/11/28 • [Edit Online](#)

```
'SolutionDir' key already exists in Properties collection. Overriding value.
```

问题

属性已在项目文件中指定和通过 pack 命令使用还传递 `-Properties` 或 `-p:NuspecProperties` 选项。

解决方案

请确保该项目文件中定义的属性，则不传递通过 pack 命令使用相同的属性 `-Properties` 或 `-p:NuspecProperties` 选项。

NuGet 警告 NU5115

2018/9/5 • [Edit Online](#)

Description was not specified. Using 'Description'.

问题

包命令未指定属性，因此改为选择默认值。

解决方案

请 specify 提到属性作为 msbuild 属性，如果使用的项目文件的 csproj 或 nuspec 文件中。

NuGet 警告 NU5116

2018/9/5 • [Edit Online](#)

```
'Content\sample.txt' was included in the project but doesn't exist. Skipping...
```

问题

打包项目中指定的内容文件不存在磁盘上，正在跳过。

解决方案

请确保所有项目中指定的内容文件在磁盘上存在。

NuGet 警告 NU5117

2018/9/5 • [Edit Online](#)

```
'$(MSBuildProjectDirectory)/tools/sample.txt' was included in the project but the path could not  
be resolved. Skipping...
```

问题

在项目文件中添加了文件，但包操作期间无法解析文件路径。

解决方案

包操作无法解析为 msbuild 变量已添加的文件。请更新为完全解析的路径的路径。

NuGet 警告 NU5118

2018/9/5 • [Edit Online](#)

```
File 'F :\validation\test\proj\tools\readme.1.txt' is not added because the package already contains file 'tools\readme.txt'
```

问题

为添加的文件 Content 项目中可能不文件添加到包正在生成, 因为另一个具有相同 PackagePath 已添加到包。

解决方案

请确保任何 Content 文件添加到项目文件, 使用对包具有唯一 PackagePath 元数据。

NuGet 警告 NU5119

2018/9/5 • [Edit Online](#)

File 'F:\project\binary\Libuv.1.10.0.nupkg' was not added to the package. Files and folders starting with '.' or ending with '.nupkg' are excluded by default. To include this file, use -NoDefaultExcludes from the commandline

问题

找到 NuGet 包操作 `.nupkg` 文件 / 文件夹开头 `.` 若要添加到包。

解决方案

如果您对包意向 `.nupkg` 文件 / 文件夹开头 `.` 请使用 `-NoDefaultExcludes` 属性, 以允许这些文件/文件夹的打包。

NuGet 警告 NU5120

2020/1/31 • [Edit Online](#)

`install.ps1` script will be ignored when the package is installed after the migration.

问题

包中包含 `install.ps1` 文件。如果将包安装到项目的包引用类型，则将不执行该文件。

解决方案

请请求包作者修复包，使其不再包含 `install.ps1` 文件。有关详细信息，请参阅[NuGet 文档](#)。

NuGet 警告 NU5121

2020/1/31 • [Edit Online](#)

```
'content' assets will not be available when the package is installed after the migration.
```

问题

包包含 `Content` 文件夹下的文件。如果将包安装到项目的包引用类型，这些资源将不可用。

解决方案

请请求包作者修复包，使其不再包含 `Content` 的文件夹。有关详细信息，请参阅[NuGet 文档](#)。

NuGet 警告 NU5122

2020/1/31 • [Edit Online](#)

XDT transform file 'tools/transform.xdt' will not be applied when the package is installed after the migration.

问题

包中包含的程序集文件不在特定于目标框架的文件夹下。包将不与项目的包引用类型兼容。

解决方案

请请求包作者修复包，使其不再包含 `install.ps1` 文件。有关详细信息，请参阅[NuGet 文档](#)。

NuGet 警告 NU5123

2018/9/5 • [Edit Online](#)

The file 'content//readme.txt' path, name, or both are too long. Your package might not work without long file path support. Please shorten the file path or file name.

问题

检测到一个文件来安装路径的长度超过 200 个字符。安装的路径指 `<package_id>/<version>/target_file_path`，其中 `target_file_path` 中定义 `target` 属性的 `<files>` nuspec 文件中的部分。

解决方案

请确保路径 `<package_id>/<version>/target_file_path` 为包含在包中的所有文件都都小于 200 个字符，其中 `target_file_path` 中定义 `target` 属性的 `<files>` nuspec 文件中的部分。

NuGet 警告 NU5124

2018/11/28 • [Edit Online](#)

The license identifier 'Microsoft-SpecialLicense' is not recognized by the current toolset.

问题

NuGet 客户端工具有一个已知的许可证标识符列表。有时这一知识包含在产品中，如版本的工具并不知道许可证的标准化。其他情况下，该许可证只是不是标准化的许可证，与以往变为 1 没有清除时间线。

解决方案

使用更新的客户端能够理解想要使用的许可证标识符。如果没有时间线所述的许可证，以成为 SPDX 标准许可证，请使用的许可证文件而不是表达式。

NuGet 警告 NU5125

2019/4/23 • [Edit Online](#)

The 'licenseUrl' element will be deprecated. Consider using the 'license' element instead.

问题

`licenseUrl` 元素替换为 `license` 元素。

解决方案

如果您创建 NuGet 包使用 `dotnet pack` 或 `msbuild -t:pack`，并遵照文档[打包许可证表达式或使用 MSBuild 目标的许可证文件](#)。

如果您使用 `.nuspec` 文件，请使用 `license` 元素。

NuGet 警告 NU5127

2019/9/26 • [Edit Online](#)

This package does not contain a lib/ or ref/ folder, and will therefore be treated as compatible for all frameworks. Since framework specific files were found under the build/ directory for net45, netstandard2.0, consider creating the following empty files to correctly narrow the compatibility of the package:

-lib/net45/_._
-lib/netstandard2.0/_._

问题

使用包的 `PackageReference` 项目仅使用 `lib/` 和 `ref/` 程序集来确定包兼容性。因此，没有任何 `lib/` 或 `ref/` 文件的包将被视为与所有项目兼容。但是，如果此包包含特定于一个或多个目标框架名字对象(`tfm`)的生成文件，则包使用者可能希望包在未使用任何生成文件时失败。

解决方案

如警告消息中所示，请在 `lib` 文件夹 `_._` 中为所列的 `tfm` 创建一个名为的空文件。当项目与包不兼容时，这 `PackageReference` 将允许 NuGet 使项目还原失败。

NuGet 警告 NU5128

2019/9/26 • [Edit Online](#)

方案 1

Some target frameworks declared in the dependencies group of the nuspec and the lib/ref folder do not have exact matches in the other location. Consult the list of actions below:

- Add a dependency group for .NETStandard2.0 to the nuspec

问题

包 `lib/<tfm>/` 中 `ref/<tfm>/` 的或目录至少包含警告消息中指定的[目标框架名字对象\(TFM\)](#)的一个文件。但是，此 TFM 在 `nuspec` 文件中不存在依赖关系组。这可能会导致包使用者相信包与 TFM 不兼容，即使包没有依赖项也是如此。如果包具有未声明的依赖项，则使用该包的项目将会遇到运行时错误。

解决方案

- 在项目上运行 NuGet 的 pack 目标

如果可能，请使用[NuGet 的 MSBuild 包目标](#)，因为它会自动将程序集 tfm 与项目的目标框架中的依赖项组匹配。请注意，你的项目 `PackageReference` 必须使用其自己的 NuGet 依赖项。如果你的项目使用包 `.config`，则需要使用 `nuget.exe pack nuspec` 和文件。

- 手动编辑 `nuspec` 的文件

如果你使用的是自 `nuspec` 定义文件，则建议每个包含 `lib/` 或 `ref/` 程序集的 TFM 都应有一个匹配的依赖项组，即使这些依赖项与下一个兼容的 TFM 相同。例如，如果包包含 `netstandard1.0` 和 `netstandard2.0` 程序集，且这些依赖项对于两者都是相同的，则建议这两个 tfm 都列为具有重复依赖项的依赖项组。

请注意，在程序集路径中使用的 TFM 标识符使用的格式不同于在依赖项组中使用的 TFM 标识符。警告消息指定要在依赖项组中使用的正确名称。如果包对于该目标框架没有任何依赖项，请使用空组。例如：

```
<package>
  <metadata>
    ...
    <dependencies>
      <group targetFramework=".NETFramework4.7.2" />
    </dependencies>
  </metadata>
  ...
</package>
```

- `lib/` 删除或文件 `ref/`

如果你不希望包与报告的 TFM 兼容，请修改你的项目，以使该 TFM `lib/<tfm>/` 的 `ref/<tfm>/` 包中不存在或文件。例如，如果警告指出向添加 `.NETFramework4.7.2` 的依赖项组 `nuspec`，则从包中删除任何 `lib/net472/*` 和 `ref/net472/*` 文件。

方案 2

Some target frameworks declared in the dependencies group of the nuspec and the lib/ref folder do not have exact matches in the other location. Consult the list of actions below:

- Add lib or ref assemblies for the netstandard2.0 target framework

问题

该文件具有所报告的目标框架名字对象(TFM)的依赖项组, 但 lib/ 在或 ref/ 中不存在此 TFM 的程序集。 nuspec 如果存在兼容 TFM 的程序集, 则包仍将安装, 但对于编译时使用的程序集, 依赖项可能不正确, 并且可能会导致项目在运行时失败。

解决方案

- 在项目上运行 NuGet 的 pack 目标

如果可能, 请使用[NuGet 的 MSBuild 包目标](#), 因为它会自动将程序集 tfm 与项目的目标框架中的依赖项组匹配。请注意, 你的项目 PackageReference 必须使用其自己的 NuGet 依赖项。如果你的项目使用包 .config, 则需要使用 nuget.exe pack nuspec 和文件。

- 手动编辑 nuspec 文件

添加报告的 TFM 作为为其编译项目的附加目标框架, 并将程序集添加到包。如果使用 SDK 样式项目多目标多个 tfm, NuGet 的 MSBuild 包目标可以自动将程序集添加到正确 lib/<tfm>/ 的文件夹中, 并创建具有正确 tfm 和依赖项的依赖项组。如果你使用的是非 SDK 样式项目, 你可能需要为其他 TFM 创建附加的项目文件, 并修改 nuspec 文件以将输出程序集复制到包中正确的位置。

- 添加空 ___.文件

如果包不包含任何程序集(如元包), 请考虑将空 ___.文件添加 lib/<tfm>/ 到警告消息中列出的 tfm 的目录中。例如, 如果警告指出要为 netstandard2.0 目标框架添加程序集, 请在包中创建一个空 lib/netstandard2.0/___.文件。

- 删除依赖项组

如果使用自定义 nuspec 文件, 请删除所报告 TFM 的依赖项组, 只保留 tfm 的依赖项组, 其中 ref/<tfm>/ lib/<tfm>/ 或文件存在。

- 删除与程序集无关的包的所有依赖项

如果你的包不包含任何 lib/ 或 ref/ 文件, 并且不是元包, 则可能没有包使用者所需的任何依赖项。如果用 NuGet 的 MSBuild Pack 目标进行打包, 可以在项目文件

<SuppressDependenciesWhenPacking>true</SuppressDependenciesWhenPacking> 中的 PropertyGroup 任意处设置。如果使用的是自定义 nuspec 文件, 请 <dependencies> 删除元素。

- 其他方案

此警告是在 NuGet 5.3 的开发过程中添加的, 第一条功能在 .NET Core SDK 3.0 预览版9中提供。[NuGet/Home #8583](#) 跟踪在太多方案中引发警告的问题。可以使用 MSBuild 属性 NoWarn (添加

<NoWarn>\$ (NoWarn);NU5128</NoWarn> 到项目文件中 PropertyGroup 的任何)。如果有多个受影响的项目, 则 [Directory.Build.targets](#) 可以使用自动 NoWarn 添加到所有项目。

NuGet 警告 NU5129

2020/1/31 • [Edit Online](#)

At least one .<extension> file was found in '<build_folder>/<tfm>/' , but '<build_folder>/<tfm>/<package_id>. <extension>' was not.

<extension> 是: targets 、 props 之一。 <build_folder> 是以下其中之一: build 、 buildTransitive 、 buildCrossTargeting 、 buildMultiTargeting 。 <tfm> 是目标框架名字对象 , 或可能不存在。 <package_id> 是包的包标识符。

例如:

```
At least one .targets file was found in 'build/netstandard2.0/' , but 'build/netstandard2.0/MyPackage.targets' was not.  
At least one .props file was found in 'build/netstandard2.0/' , but 'build/netstandard2.0/MyPackage.props' was not.  
At least one .props file was found in 'buildTransitive/net472/' , but 'buildTransitive/net472/My.Package.Id.props' was not.  
At least one .targets file was found in 'buildMultitargeting/netcoreapp3.0/' , but 'buildMultitargeting/netcoreapp3.0/Contoso.Utilities.targets' was not.  
At least one .props file was found in 'build/' , but 'build/AdventureWorks.Tools.props' was not.
```

问题

包含 MSBuild 属性和目标 的包需要遵循在 .props 或 .targets 扩展之前使用包 id 的命名约定。不遵循此约定的文件将不会导入到使用该包的项目。

示例: 如果包 id Contoso.Utilities 并且包含 build/Contoso.Utilities.props 和 build/Utilities.targets 的文件 , 则将仅使用包将 Contoso.Utilities.props 文件导入项目中。 NuGet 不会导入 Utilities.targets 。

解决方案

重命名该文件以满足约定。

在上面的示例中 , 应将 build/netstandard2.0/Utilities.targets 重命名为 NuGet build/netstandard2.0/Contoso.Utilities.targets , 以开始导入它。如果要在 Contoso.Utilities.props 中导入 Utilities.targets , 请将该文件重命名为使用 .props 扩展。

NuGet 警告 NU5130

2019/9/26 • [Edit Online](#)

Some target frameworks declared in the dependencies group of the nuspec and the lib/ref folder have compatible matches, but not exact matches in the other location. Unless intentional, consult the list of actions below:

- Add a dependency group for .NETFramework4.7.2 to the nuspec

问题

包 `lib/<tfm>/` 中 `ref/<tfm>/` 的或目录至少包含警告消息中指定的 [目标框架名字对象 \(TFM\)](#) 的一个文件。但是，此 TFM 在 `nuspec` 文件中不存在依赖关系组。这可能会导致包使用者相信包与 TFM 不兼容。如果指定 TFM 的程序集与依赖关系组中定义的兼容 TFM 具有不同的依赖关系，则使用包的项目可能会遇到运行时失败。

解决方案

- 在项目上运行 NuGet 的 pack 目标

如果可能，请使用[NuGet 的 MSBuild 包目标](#)，因为它会自动将程序集 tfm 与项目的目标框架中的依赖项组匹配。请注意，你的项目 `PackageReference` 必须使用其自己的 NuGet 依赖项。如果你的项目使用包 `.config`，则需要使用 `nuget.exe pack nuspec` 和文件。

- 手动编辑 `nuspec` 文件

如果你使用的是自 `nuspec` 定义文件，则建议每个包含 `lib/` 或 `ref/` 程序集的 TFM 都应有一个匹配的依赖项组，即使这些依赖项与下一个兼容的 TFM 相同。例如，如果包包含 `netstandard1.0` 和 `netstandard2.0` 程序集，且这些依赖项对于两者都是相同的，则建议这两个 tfm 都列为具有重复依赖项的依赖项组。

请注意，在程序集路径中使用的 TFM 标识符使用的格式不同于在依赖项组中使用的 TFM 标识符。警告消息指定要在依赖项组中使用的正确名称。如果包对于该目标框架没有任何依赖项，请使用空组。例如：

```
<package>
  <metadata>
    ...
    <dependencies>
      <group targetFramework=".NETFramework4.7.2" />
    </dependencies>
  </metadata>
  ...
<package>
```

- `lib/` 删除或文件 `ref/`

如果你不希望包与报告的 TFM 兼容，请修改你的项目，以使该 TFM `lib/<tfm>/` 的 `ref/<tfm>/` 包中不存在或文件。例如，如果警告指出向添加 `.NETFramework4.7.2` 的依赖项组 `nuspec`，则从包中删除任何 `lib/net472/*` 和 `ref/net472/*` 文件。

NuGet 警告 NU5131

2020/1/8 • [Edit Online](#)

References were found in the nuspec, but some reference assemblies were not found in both the nuspec and ref folder. Add the following reference assemblies:

- Add AssemblyName.dll to the ref/net472/ directory

问题

NuGet 提供一项功能，允许包作者选择在编译时将在使用包的项目中使用的程序集。

如果未遵循所需的约定，则使用带有 `PackageReference` 的包的项目在运行时可能会因缺少程序集而失败。

解决方案

Nuspec 文件的 `<references>` 部分中的程序集列表应在 `ref/<tfm>/` 中具有匹配的程序集。

例如，请考虑具有以下文件的包：

```
lib\net472\MyLib.dll  
lib\net472\MyHelpers.dll  
lib\net472\MyUtilities.dll
```

包作者要阻止包使用者编写直接调用 `MyUtilities.dll` 的代码，因此它们会将以下内容添加到 nuspec 文件中：

```
<references>  
  <group targetFramework="net472">  
    <reference file="MyLib.dll" />  
    <reference file="MyHelpers.dll" />  
  </group>  
</references>
```

使用 `PackageReference` 时，此包将无法按预期工作。若要解决此问题，包还必须包含以下文件：

```
ref\net472\MyLib.dll  
ref\net472\MyHelpers.dll
```

NuGet 警告 NU5500

2018/9/5 • [Edit Online](#)

问题

NuGet 包操作时出现问题。我们尚未尚未分配唯一的警告代码针对该问题时，使用 NU5000。可以帮助我们改进，请随时提出此问题与您的错误的详细信息。

解决方案

检查输出窗口（在 Visual Studio）或控制台输出（通过 NuGet 或 dotnet 命令行工具）有关详细信息。

project.json 引用

2020/4/8 • [Edit Online](#)

NuGet 3.x+

`project.json` 文件维护项目中使用的包列表，称为包管理格式。它会取代 `packages.config`，但在 NuGet 4.0+ 中又被 `PackageReference` 取代。

`project.lock.json` 文件(如下所述)也用于采用 `project.json` 的项目。

`project.json` 具有以下基本结构，其中四个顶层对象均可拥有任意数量的子对象：

```
{  
  "dependencies": {  
    "PackageID" : "{version_constraint}"  
  },  
  "frameworks" : {  
    "TxM" : {}  
  },  
  "runtimes" : {  
    "RID": {}  
  },  
  "supports" : {  
    "CompatibilityProfile" : {}  
  }  
}
```

依赖项

以下的形式列出项目的 NuGet 包依赖项：

```
"PackageID" : "version_constraint"
```

例如：

```
"dependencies": {  
  "Microsoft.NETCore": "5.0.0",  
  "System.Runtime.Serialization.Primitives": "4.0.10"  
}
```

`dependencies` 部分是“NuGet 包管理器”对话框向项目中添加包依赖项的位置。

包 ID 对应于 nuget.org 上包的 ID，与包管理器控制台中使用的 ID 相同：`Install-Package Microsoft.NETCore`。

还原包时，`"5.0.0"` 版本约束意味着 `>= 5.0.0`。也就是说，如果 5.0.0 在服务器上不可用，但 5.0.1 可用，则 NuGet 安装 5.0.1，并发出有关升级的警告。否则，NuGet 尽可能在服务器上选取最低的版本，以与约束匹配。

请参阅[依赖项解析](#)深入了解解析规则。

管理依赖项资产

依赖项的哪些资产流入顶级项目是通过在依赖项引用的 `include` 和 `exclude` 属性中指定一组以逗号分隔的标记来控制的。下表中列出了这些标记：

contentFiles	内容
运行库	运行时、资源和 FrameworkAssemblies
编译	lib
build	生成(MSBuild 属性和目标)
本机	本机
none	无文件夹
all	全部文件夹

用 `exclude` 指定的标记优先于用 `include` 指定的标记。例如, `include="runtime, compile" exclude="compile"` 和 `include="runtime"` 相同。

例如, 若要包括依赖项的 `build` 和 `native` 文件夹, 请使用以下内容:

```
{
  "dependencies": {
    "packageA": {
      "version": "1.0.0",
      "include": "build, native"
    }
  }
}
```

若要排除依赖项的 `content` 和 `build` 文件夹, 请使用以下内容:

```
{
  "dependencies": {
    "packageA": {
      "version": "1.0.0",
      "exclude": "contentFiles, build"
    }
  }
}
```

框架

列出项目运行的框架, 如 `net45`、`netcoreapp`、`netstandard`。

```
"frameworks": {
  "netcore50": {}
}
```

`frameworks` 部分仅允许一个条目。(例外情况是用于 ASP.NET 项目的 `project.json` 文件, 它们通过弃用的 DNX 工具链生成, 可允许多个目标。)

运行时

列出应用运行的操作系统和体系结构, 如 `win10-arm`、`win8-x64`、`win8-x86`。

```
"runtimes": {
    "win10-arm": { },
    "win10-arm-aot": { },
    "win10-x86": { },
    "win10-x86-aot": { },
    "win10-x64": { },
    "win10-x64-aot": { }
}
```

可在任何运行时上运行的包含 PCL 的包不需要指定运行时。这还必须对任何依赖项适用，否则必须指定运行时。

支持

为包依赖项定义一组检查。可以定义期望 PCL 或应用运行的位置。定义限制并不严格，因为代码可在其他位置运行。但指定这些检查会让 NuGet 检查所有依赖项在所列 TxM 上是否满足。此支持的值的示例有：`net46.app`、`uwp.10.0.app` 等。

在“可移植类库目标”对话框中选择条目时，此部分应自动填充。

```
"supports": {
    "net46.app": {},
    "uwp.10.0.app": {}
}
```

导入

导入旨在允许使用 `dotnet` TxM 的包处理未声明 `dotnet` TxM 的包。如果项目使用的是 `dotnet` TxM，那么依赖的所有包也必须有 `dotnet` TxM，除非将以下内容添加到 `project.json` 中，以允许非 `dotnet` 平台与 `dotnet` 兼容：

```
"frameworks": {
    "dotnet": { "imports" : "portable-net45+win81" }
}
```

如果使用的是 `dotnet` TxM，那么 PCL 项目系统会基于支持的目标添加相应的 `imports` 语句。

可移植应用和 Web 项目的区别

NuGet 使用的 `project.json` 文件是 ASP.NET Core 项目中该文件的子集。在 ASP.NET Core 中，`project.json` 用于项目元数据、编译信息和依赖项。在其他项目系统中使用时，这三项内容则拆分为单独的文件，并且 `project.json` 包含的信息更少。明显的区别包括：

- `frameworks` 部分仅可有一个框架。
- 该文件不能包含 DNX `project.json` 文件中可看到的依赖项、编译选项等。鉴于只能有一个框架，输入框架特定的依赖项毫无意义。
- 编译由 MSBuild 处理，因此编译选项、预处理器定义等都属于 MSBuild 项目文件而不是 `project.json`。

在 NuGet 3+ 中，不需要开发人员手动编辑 `project.json`，因为 Visual Studio 中的包管理器 UI 会操作内容。即便如此，当然还是可以编辑文件，但必须生成项目以启动包还原或以其他方式调用还原。请参阅[包还原](#)。

project.lock.json

在还原使用 `project.lock.json` 的项目中的 NuGet 包的过程中，会生成 `project.json` 文件。它拥有 NuGet 在走包的关系图时生成的所有信息的快照，并包括项目中所有包的版本、内容和依赖项。生成项目时，生成系统根据此快

照从相关的全局位置选择包，而不是依靠项目本身中的本地包文件夹。这使得生成性能更快，因为仅需读取 `project.lock.json` 而不是许多单独的 `.nuspec` 文件。

`project.lock.json` 在包还原时自动生成，因此可通过将其添加到 `.gitignore` 和 `.tfignore` 文件，将其从源代码管理去掉（请参阅[包和源代码管理](#)）。但是，如果将其包括在源代码管理中，更改历史记录会显示随着时间推移解析的依赖项的更改。

project.json 和 UWP

2020/4/13 • [Edit Online](#)

IMPORTANT

此内容已弃用。项目应使用 `packages.config` 或 `PackageReference` 格式。

本文档介绍使用 NuGet 3+ (Visual Studio 2015 及更高版本) 中功能的包结构。可以通过将 `.nuspec` 的 `minClientVersion` 设置为 3.1 来声明你需要此处介绍的功能。

向现有包添加 UWP 支持

如果有一个现有包，并且想添加对 UWP 应用程序的支持，则不需要采用此处介绍的打包格式。如果需要此处介绍的功能并且只愿意使用已更新到 3+ 版 NuGet 的客户端，只需采用此格式即可。

已面向 netcore45

如果已面向 `netcore45`，并且不需要利用此处介绍的功能，则无需执行任何操作。UWP 应用程序可以使用 `netcore45` 包。

希望利用 Windows 10 特定的 API

在这种情况下，需要将 `uap10.0` 目标框架名字对象 (TFM 或 TxM) 添加到包。在包中创建一个新文件夹，并将已编译为与 Windows 10 一起使用的程序集添加到该文件夹中。

不需要 Windows 10 特定的 API，但需要新的 .NET 功能或者尚没有 netcore45

在这种情况下，需要将 `dotnet` TxM 添加到包。与其他 TxM 不同，`dotnet` 并不意味着外围应用或平台。也就是说，包适用于依赖项适用的任何平台。使用 `dotnet` TxM 生成包时，`.nuspec` 中可能存在许多 TxM 特定的依赖项，因为需要定义所依赖的 BCL 包，如 `System.Text`、`System.Xml` 等。这些依赖项适用的位置即是包的工作位置。

如何查找依赖项

有两种方法可以查找要列出的依赖项：

1. 使用 [NuSpec 依赖项生成器](#) 第三方工具。该工具自动执行该过程，并使用生成时依赖的包更新 `.nuspec` 文件。该工具通过 NuGet 包 `NuSpec.ReferenceGenerator` 提供。
2. (复杂方法) 使用 `ILDasm` 检查 `.dll`，查看运行时实际需要的程序集。然后确定这些程序集分别来自哪个 NuGet 包。

请参阅 [project.json](#) 主题，详细了解可帮助创建支持 `dotnet` TxM 的包的功能。

IMPORTANT

如果打算将包用于 PCL 项目，强烈建议创建一个 `dotnet` 文件夹，以避免警告和潜在的兼容性问题。

目录结构

使用此格式的 NuGet 包具有以下已知文件夹和行为：

III	II
生成	此文件夹中包含 MSBuild 目标和属性文件将以不同的方式集成到项目中，但除此之外没有更改。
工具	<code>install.ps1</code> 和 <code>uninstall.ps1</code> 不运行。 <code>init.ps1</code> 像往常一样工作。
内容	内容不会自动复制到用户的项目中。后续发行版本计划支持将内容包含在项目中。
Lib	对于许多包而言， <code>lib</code> 的工作方式与 NuGet 2.x 相同，只不过其中使用的名称具有扩展选项，并且在使用包时具有更好的逻辑来选取正确的子文件夹。但是，与 <code>ref</code> 一起使用时， <code>lib</code> 文件夹包含如下程序集：实现由 <code>ref</code> 文件夹中的程序集定义的外围应用。
引用	<code>ref</code> 是一个可选文件夹，其中包含 .NET 程序集，用于定义应用程序编译的公共外围（公共类型和方法）。此文件夹中的程序集可能没有实现，只是用于为编译器定义外围应用。如果包没有 <code>ref</code> 文件夹，那么 <code>lib</code> 同时为引用程序集和实现程序集。
运行时	<code>runtimes</code> 是一个可选文件夹，其中包含操作系统特定的代码，如 CPU 体系结构以及操作系统特定的或依赖于其他平台的二进制文件。

包中的 MSBuild 目标和属性文件

NuGet 包可能包含 `.targets` 和 `.props` 文件，这些文件被导入到包安装到的任何 MSBuild 项目中。在 NuGet 2.x 中，此操作是通过向 `.csproj` 文件注入 `<Import>` 语句而完成的；在 NuGet 3.0 中，没有特定的“安装到项目”操作。而是包还原过程写入 `[projectname].nuget.props` 和 `[projectname].NuGet.targets` 两个文件。

MSBuild 知道查找这两个文件，并在项目生成过程开始和结束时自动导入它们。此行为与 NuGet 2.x 非常相似，但有一个主要区别：在此情况下，无法保证目标/道属性文件的顺序。但是，MSBuild 通过 `<Target>` 定义的 `BeforeTargets` 和 `AfterTargets` 特性（请参阅 [Target 元素 \(MSBuild\)](#)）提供了对目标进行排序的方法。

Lib 和引用

在 NuGet v3 中，`lib` 文件夹的行为没有重大变化。但是，所有程序集都必须位于以 TxM 命名的子文件夹内，不再直接放置在 `lib` 文件夹下。TxM 是包中给定资产适用的平台名称。逻辑上，这些是目标框架名字对象 (TFM) 的扩展，例如，`net45`、`net46`、`netcore50` 和 `dnxcore50` 都是 TxM 的示例（请参阅 [目标框架](#)）。TxM 可以指框架 (TFM) 以及其他平台特定的外围应用。例如，UWP TxM (`uap10.0`) 表示 .NET 外围应用以及适用于 UWP 应用程序的 Windows 外围应用。

Lib 结构示例：

```
lib
├── net40
│   └── MyLibrary.dll
└── wp81
    └── MyLibrary.dll
```

`lib` 文件夹包含在运行时使用的程序集。对于大多数包而言，每个目标 TxM 的 `lib` 下的文件夹全都是必需的。

引用

有时在编译期间应使用不同的程序集(如 .NET 引用程序集)。对于这些情况,请使用名为 `ref` (“引用程序集”的缩写)的顶层文件夹。

大多数包创建者都不需要 `ref` 文件夹。这对以下包非常有用:需要为编译和 IntelliSense 提供一致的外围应用,但对不同的 TxM 具有不同的实现。最大的用例是 `System.*` 包,这些包是在 NuGet 上提供 .NET Core 的过程中生成的。这些包具有各种实现,这些实现由一组一致的引用程序集统一。

从表面上讲, `ref` 文件夹中包含的程序集是传递给编译器的引用程序集。对于使用过 `csc.exe` 的用户来说,这些是我们传递给 [C# /reference 选项](#) 开关的程序集。

`ref` 文件夹的结构与 `lib` 的结构相同,例如:

```
└── MyImageProcessingLib
    ├── lib
    │   ├── net40
    │   │   └── MyImageProcessingLibrary.dll
    │   ├── net451
    │   │   └── MyImageProcessingLibrary.dll
    │   └── win81
    │       └── MyImageProcessingLibrary.dll
    └── ref
        ├── net40
        │   └── MyImageProcessingLibrary.dll
        └── portable-net451-win81
            └── MyImageProcessingLibrary.dll
```

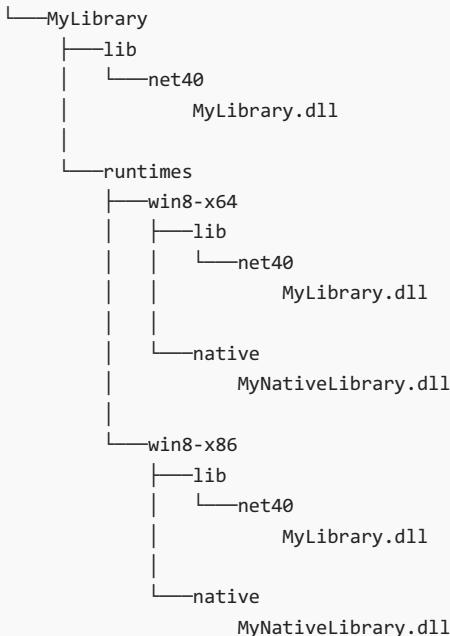
在本例中, `ref` 目录中的程序集是相同的。

运行时

运行时文件夹包含在特定“运行时”上运行所需的程序集和本机库,通常由操作系统和 CPU 体系结构定义。这些运行时使用[运行时标识符 \(RID\)](#) 进行标识,如 `win`、`win-x86`、`win7-x86`、`win8-64` 等。

本机帮助程序使用特定于平台的 API

以下示例显示这样的包:拥有适用于多个平台的纯托管实现,但可在 Windows 8 上使用本机帮助程序以调用 Windows 8 特定的本机 API。



提供上述包时，将发生以下情况：

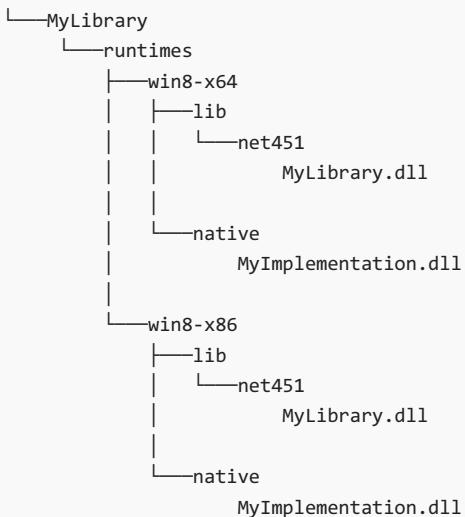
- 当不在 Windows 8 上，将使用 `lib/net40/MyLibrary.dll` 程序集。
- 在 Windows 8 上时，将使用 `runtimes/win8-<architecture>/lib/MyLibrary.dll`，并且 `native/MyNativeHelper.dll` 将复制到生成的输出。

在上面的示例中，`lib/net40` 程序集是纯托管代码，而运行时文件夹中的程序集将平台调用到本机帮助程序程序集中，以调用特定于 Windows 8 的 API。

始终仅选取单个 `lib` 文件夹，因此如果有运行时特定的文件夹，将选择通过非运行时特定的 `lib`。本机文件夹是累加式的，如果存在，将被复制到生成的输出。

托管包装器

使用运行时的另一种方式是将属于纯托管包装器的包提供到本机程序集上。在此方案中，将创建如下所示的包：



在此情况下，没有顶层 `lib` 文件夹作为该文件夹，因为此包的实现不依赖于相应的本机程序集。如果托管程序集 `MyLibrary.dll` 在这两种情况下完全相同，则可将它放在顶层 `lib` 文件夹中；但是，由于缺少本机程序集并不会导致将包安装到非 win-x86 或 win-x64 平台上时失败，因此将使用顶层 `lib`，但不会复制本机程序集。

创作适用于 NuGet 2 和 NuGet 3 的包

如果想创建一个可供使用 `packages.config` 的项目使用的包以及使用 `project.json` 的包，则以下内容适用：

- 引用和运行时只适用于 NuGet 3。在 NuGet 2 中将被忽略。
- 不能依靠 `install.ps1` 或 `uninstall.ps1` 来运行。这些文件在使用 `packages.config` 时执行，但使用 `project.json` 时会忽略。因此包需要不运行它们时可用。`init.ps1` 仍在 NuGet 3 上运行。
- 目标与属性安装不同，因此请确保包在两个客户端上按预期工作。
- 在 NuGet 3 中，`lib` 的子目录必须是一个 TxM。不能将库放置在 `lib` 文件夹的根目录下。
- NuGet 3 将不会自动复制内容。包的使用者可以自己复制文件，或使用任务运行程序之类的工具来自动复制文件。
- NuGet 3 不会运行源和配置文件转换。

如果支持 NuGet 2 和 3，`minClientVersion` 应该是包适用的最低版本的 NuGet 2 客户端。如果存在现有包，则不需要更改。

创建包时 project.json 产生的影响

2020/4/8 • [Edit Online](#)

IMPORTANT

此内容已弃用。项目应使用 `packages.config` 或 `PackageReference` 格式。

NuGet 3+ 中使用的 `project.json` 系统通过多种方式影响包作者，这些内容将在后续部分介绍。

影响现有包使用的更改

传统的 NuGet 包支持一系列不会传送到可传递体系的功能。

忽略了安装和卸载脚本

[依赖项解析](#) 中介绍的可传递还原模型不具有“包安装时间”概念。包的状态为存在或不存在，但安装包后不会出现一致的进程。

此外，仅 Visual Studio 支持安装脚本。其他 IDE 必须模拟 Visual Studio 扩展性 API 以尝试支持此类脚本，且在常见编辑器和命令行工具中不提供支持。

不支持转换内容。

与安装脚本相似，转换在包安装上运行，且通常不是幂等的。由于不再存在安装时间，XDT 转换和类似功能不受支持，并且如果在可传递方案中使用了此类包，则会忽略前述转换和功能。

内容

传统的 NuGet 包传送源代码和配置文件之类的内容文件。通常用于以下两个方案：

- 将初始文件放置到项目中，以便用户稍后进行编辑。常见示例是默认配置文件。
- 将内容文件用作项目中所安装程序集的辅助文件。示例是程序集使用的徽标图像。

目前禁用了对内容的支持，其原因与不支持脚本和转换的原因相似，但我们正在设计对内容的支持。

内容文件仍可以在包中进行传递，且此类文件目前被忽略，但是最终用户仍可以将其复制到适当的位置。

可在此处查看恢复内容文件的其中一项建议并跟踪其进度：<https://github.com/NuGet/Home/issues/627>。

对包作者的影响

使用上述功能的包必须使用不同的机制。对此最常用的有用机制是继续受到完全支持的 MSBuild 目标/属性。生成系统可以选择在包中选取其他约定。这样便可实现 MSBuild 目标和 Roslyn 分析器支持。可以生成支持适用于 `packages.config` 和 `project.json` 方案的目标和分析器的包。

尝试修改项目以简化启动的包通常用于一组非常有限的方案，且应改为提供自述文件或包的使用方法指南。

大多数现有包都无需使用下文介绍的包格式。

格式使得本机内容作为优先方案。这意味着，托管程序集依赖近硬件实现基于目标平台随托管程序集提供二进制实现。例如，`System.IO.Compression` 包便是利用此技术。<https://www.nuget.org/packages/System.IO.Compression>

总之，如果上述功能不是绝对必要，我们建议继续使用现有的包格式，因为仅 NuGet 3.x+ 支持此处介绍的格式。

可通过填充生成适用于 `packages.config` 和 `project.json` 方案的包，但以传统方式组织包的结构通常会更简单，无需上述弃用的功能。

3.x 包格式

3.x 包格式允许使用 NuGet 2.x 之外的其他几项功能：

1. 定义一个引用程序集和一组实现程序集，前者用于编译，后者用于不同平台/设备上的运行时。这样，你就可
以利用平台特定的 API，同时为使用者提供常见的外围应用。具体而言，这有助于更轻松地编写中间可移植
库。
2. 允许包在操作系统或 CPU 体系结构等平台上透视。
3. 允许将平台特定的实现分离到辅助包中。
4. 支持本机依赖项作为优先选项。

NuGet 跨平台插件

2020/3/19 • [Edit Online](#)

已添加 NuGet 4.8 + 跨平台插件支持。这是通过生成新的插件扩展性模型实现的，该模型必须符合一组严格的操作规则。这些插件是独立的可执行文件 (.NET Core world 中的 runnables)，NuGet 客户端在单独的进程中启动。这是一次真正的写入，可在任何位置运行插件。它将适用于所有 NuGet 客户端工具。插件可以是 .NET Framework (Nuget.exe、Msbuild.exe 和 Visual Studio)，也可以是 .NET Core (dotnet)。定义 NuGet 客户端与插件之间的版本控制通信协议。在启动握手期间，2个进程会协商协议版本。

为了涵盖所有 NuGet 客户端工具方案，同时需要 .NET Framework 和 .NET Core 插件。下面介绍插件的客户端/框架组合。

客户端	框架
Visual Studio	.NET Framework
dotnet.exe	.NET Core
Nuget.exe	.NET Framework
Msbuild.exe	.NET Framework
Mono 上的 Nuget.exe	.NET Framework

工作原理

高级别工作流的描述如下所示：

1. NuGet 发现可用插件。
2. 如果适用，NuGet 将按优先级顺序循环访问插件，并逐个启动插件。
3. NuGet 将使用第一个可为请求服务的插件。
4. 插件不再需要时将关闭。

一般插件要求

当前协议版本为 2.0.0。在此版本下，要求如下所示：

- 具有有效的受信任 Authenticode 签名程序集，将在 Windows 和 Mono 上运行。尚无对 Linux 和 Mac 上运行的程序集的特殊信任要求。[相关问题](#)
- 支持在 NuGet 客户端工具的当前安全上下文中进行无状态启动。例如，NuGet 客户端工具不会在稍后所述的插件协议之外执行提升或其他初始化。
- 除非显式指定，否则为非交互式。
- 遵循协商插件协议版本。
- 在合理的时间段内响应所有请求。
- 处理任何正在进行的操作的取消请求。

以下规范更详细地介绍了技术规范：

- [NuGet 包下载插件](#)
- [NuGet 跨 x-plat 身份验证插件](#)

客户端-插件交互

NuGet 客户端工具和插件通过标准流(stdin、stdout、stderr)与 JSON 通信。所有数据必须采用 UTF-8 编码。插件用参数 "-插件" 启动。如果用户在没有此参数的情况下直接启动了插件可执行文件，则该插件可以提供信息性消息，而不是等待协议握手。协议握手超时值为5秒。插件应尽可能简短地在中完成设置。NuGet 客户端工具将通过传入 NuGet 源的服务索引来查询插件支持的操作。插件可以使用服务索引来检查是否存在受支持的服务类型。

NuGet 客户端工具与插件之间的通信是双向的。每个请求的超时为5秒。如果操作需要更长的时间，则应发送进度消息，以防止请求超时。1分钟处于非活动状态后，插件将被视为空闲状态并关闭。

插件安装和发现

插件将通过基于约定的目录结构来发现。CI/CD 方案和超级用户可以使用环境变量来重写此行为。使用环境变量时，只允许使用绝对路径。请注意，`NUGET_NETFX_PLUGIN_PATHS` 和 `NUGET_NETCORE_PLUGIN_PATHS` 仅适用于 5.3 + 版本的 NuGet 工具和更高版本。

- `NUGET_NETFX_PLUGIN_PATHS` - 定义将由基于 .NET Framework 的工具(Nuget.exe/Msbuild.exe/Visual Studio)使用的插件。优先于 `NUGET_PLUGIN_PATHS`。(仅 NuGet 版本 5.3 +)
- `NUGET_NETCORE_PLUGIN_PATHS` - 定义将由基于 .NET Core 的工具(dotnet)使用的插件。优先于 `NUGET_PLUGIN_PATHS`。(仅 NuGet 版本 5.3 +)
- `NUGET_PLUGIN_PATHS` - 定义将用于 NuGet 进程的插件，保留优先级。如果设置了此环境变量，它将覆盖基于约定的发现。如果指定了任何一个框架特定的变量，则忽略。
- 用户-位置，`%UserProfile%/.nuget/plugins` 中的 NuGet 主页位置。此位置不能被重写。.NET Core 和 .NET Framework 插件将使用不同的根目录。

框架	插件目录
.NET Core	<code>%UserProfile%/.nuget/plugins/netcore</code>
.NET Framework	<code>%UserProfile%/.nuget/plugins/netfx</code>

每个插件都应安装在其自己的文件夹中。插件入口点将是已安装文件夹的名称，其扩展名为 .NET Core, .exe 扩展名用于 .NET Framework。

```
.nuget
  plugins
    netfx
      myPlugin
        myPlugin.exe
        nuget.protocol.dll
      ...
    netcore
      myPlugin
        myPlugin.dll
        nuget.protocol.dll
      ...
```

NOTE

当前没有用于安装插件的用户情景。只需将所需文件移动到预定位置即可。

支持的操作

新的插件协议支持两个操作。

		NUGET
下载包	1.0.0	4.3.0
身份验证	2.0.0	4.8.0

在正确的运行时下运行插件

对于 dotnet 方案中的 NuGet，插件需要能够在 dotnet 的特定运行时执行。它在插件提供程序和使用者上，以确保使用兼容的 dotnet/插件组合。用户位置插件可能会出现问题，例如，2.0 运行时下的 dotnet 尝试使用为2.1 运行时编写的插件。

功能缓存

插件的安全验证和实例化成本高昂。类似于身份验证操作，下载操作的执行频率更高，但平均 NuGet 用户只可能具有身份验证插件。为了改进体验，NuGet 将缓存给定请求的操作声明。此缓存是每个插件，其中插件键是插件路径，此功能缓存的过期时间为30天。

缓存位于 `%LocalAppData%/NuGet/plugins-cache` 中，并通过环境变量 `NUGET_PLUGINS_CACHE_PATH` 进行重写。若要清除此缓存，可以运行带有 `plugins-cache` 选项的局部变量命令。现在，`all` 局部变量 "选项也将删除插件缓存。

协议消息索引

协议版本 1.0.0 消息：

1. 关闭

- 请求方向：NuGet > 插件
- 请求不包含有效负载
- 不需要响应。适当的响应是为了使插件进程及时退出。

2. 复制包中的文件

- 请求方向：NuGet > 插件
- 该请求将包含：
 - 包 ID 和版本
 - 包源存储库位置
 - 目标目录路径
 - 包中要复制到目标目录路径的文件的可枚举
- 响应将包含：
 - 指示操作结果的响应代码
 - 如果操作成功，则为目标目录中复制的文件的完整路径的可枚举

3. 复制包文件 (.nupkg)

- 请求方向：NuGet > 插件
- 该请求将包含：
 - 包 ID 和版本
 - 包源存储库位置
 - 目标文件路径
- 响应将包含：
 - 指示操作结果的响应代码

4. 获取凭据

- 请求方向: 插件-> NuGet
- 该请求将包含:
 - 包源存储库位置
 - 使用当前凭据从包源存储库获取的 HTTP 状态代码
- 响应将包含:
 - 指示操作结果的响应代码
 - 用户名(如果可用)
 - 密码(如果可用)

5. 获取包中的文件

- 请求方向: NuGet > 插件
- 该请求将包含:
 - 包 ID 和版本
 - 包源存储库位置
- 响应将包含:
 - 指示操作结果的响应代码
 - 如果操作成功, 则为包中的文件路径的可枚举

6. 获取操作声明

- 请求方向: NuGet > 插件
- 该请求将包含:
 - 包源的服务索引 json
 - 包源存储库位置
- 响应将包含:
 - 指示操作结果的响应代码
 - 如果操作成功, 则为支持的操作(如包下载)的可枚举。如果插件不支持包源, 则该插件必须返回一组支持操作的空集。

NOTE

此消息已在版本 2.0.0 中更新。它在客户端上, 用于保留向后兼容性。

7. 获取包哈希

- 请求方向: NuGet > 插件
- 该请求将包含:
 - 包 ID 和版本
 - 包源存储库位置
 - 哈希算法
- 响应将包含:
 - 指示操作结果的响应代码
 - 如果操作成功, 则使用所请求的哈希算法的包文件哈希

8. 获取包的版本

- 请求方向: NuGet > 插件
- 该请求将包含:
 - 包 ID
 - 包源存储库位置
- 响应将包含:

- 指示操作结果的响应代码
- 如果操作成功，则为包版本的可枚举

9. 获取服务索引

- 请求方向：插件-> NuGet
- 该请求将包含：
 - 包源存储库位置
- 响应将包含：
 - 指示操作结果的响应代码
 - 如果操作成功，则为服务索引

10. 握手

- 请求方向：NuGet < > 插件
- 该请求将包含：
 - 当前插件协议版本
 - 支持的最低插件协议版本
- 响应将包含：
 - 指示操作结果的响应代码
 - 如果操作成功，则协商协议版本。失败将导致插件终止。

11. Initialize

- 请求方向：NuGet > 插件
- 该请求将包含：
 - NuGet 客户端工具版本
 - NuGet 客户端工具的有效语言。如果使用，此操作将考虑 ForceEnglishOutput 设置。
 - 默认请求超时值，它取代协议默认值。
- 响应将包含：
 - 指示操作结果的响应代码。失败将导致插件终止。

12. 日志

- 请求方向：插件-> NuGet
- 该请求将包含：
 - 请求的日志级别
 - 要记录的消息
- 响应将包含：
 - 指示操作结果的响应代码。

13. 监视 NuGet 进程退出

- 请求方向：NuGet > 插件
- 该请求将包含：
 - NuGet 进程 ID
- 响应将包含：
 - 指示操作结果的响应代码。

14. 预提取包

- 请求方向：NuGet > 插件
- 该请求将包含：
 - 包 ID 和版本
 - 包源存储库位置
- 响应将包含：

- 指示操作结果的响应代码

15. 设置凭据

- 请求方向: NuGet > 插件
- 该请求将包含:
 - 包源存储库位置
 - 最后一个已知包源用户名(如果可用)
 - 最后一个已知包源密码(如果可用)
 - 最后一个已知代理用户名(如果可用)
 - 最后一个已知代理密码(如果可用)
- 响应将包含:
 - 指示操作结果的响应代码

16. 设置日志级别

- 请求方向: NuGet > 插件
- 该请求将包含:
 - 默认日志级别
- 响应将包含:
 - 指示操作结果的响应代码

协议版本2.0.0消息

17. 获取操作声明

- 请求方向: NuGet > 插件
- 该请求将包含:
 - 包源的服务索引 json
 - 包源存储库位置
- 响应将包含:
 - 指示操作结果的响应代码
 - 如果操作成功, 则为支持的操作的可枚举。如果插件不支持包源, 则该插件必须返回一组支持操作的空集。

如果服务索引和包源为 null, 则该插件可以通过身份验证进行应答。

18. 获取身份验证凭据

- 请求方向: NuGet > 插件
- 该请求将包含:
 - URI
 - isRetry
 - NonInteractive
 - CanShowDialog
- 响应将包含
 - 用户名
 - Password
 - Message
 - 身份验证类型列表
 - MessageResponseCode

NuGet 跨平台身份验证插件

2019/7/19 • [Edit Online](#)

在版本 4.8+ 中, 所有 NuGet 客户端 (Nuget.exe、Visual Studio、dotnet 和 Msbuild.exe) 都可以使用在[NuGet 跨平台插件](#)模型之上构建的身份验证插件。

Dotnet 中的身份验证

默认情况下, Visual Studio 和 Nuget.exe 是交互式的。Nuget.exe 包含用于使其成为[非交互式](#)的开关。此外, Nuget.exe 和 Visual Studio 插件会提示用户输入。在 dotnet 中没有提示, 默认值为非交互式。

Dotnet 中的身份验证机制是设备流。如果 "还原" 或 "添加包" 操作以交互方式运行, 则在命令行上将会阻止操作并向用户提供如何完成身份验证的说明。用户完成身份验证后, 操作将继续。

若要使操作具有交互性, 应通过 `--interactive` 一个。目前只有显式 `dotnet restore` 和 `dotnet add package` 命令支持交互式切换。`dotnet build` 和 `dotnet publish` 上没有交互式开关。

MSBuild 中的身份验证

与 dotnet 类似, Msbuild.exe 默认情况下不交互, msbuild.exe 身份验证机制是设备流。若要允许还原暂停并等待身份验证, 请调用 restore with `msbuild -t:restore -p:NuGetInteractive="true"`。

创建跨平台身份验证插件

可以在[Microsoft 凭据提供程序插件](#)中找到示例实现。

插件必须符合 NuGet 客户端工具所规定的安全要求, 这一点非常重要。作为身份验证插件的插件所需的最低版本为 2.0.0。NuGet 将通过插件执行握手, 并查询支持的操作声明。有关特定消息的详细信息, 请参阅[NuGet 跨平台插件协议消息](#)。

NuGet 会将日志级别设置为 "可用", 并向该插件提供代理信息。仅在 NuGet 将日志级别设置为插件后, 才可以接受日志记录到 NuGet 控制台。

- .NET Framework 插件身份验证行为

在 .NET Framework 中, 允许插件以对话框的形式提示用户输入。

- .NET Core 插件身份验证行为

在 .NET Core 中, 无法显示对话框。插件应该使用设备流进行身份验证。该插件可以向 NuGet 发送日志消息, 并向用户发送说明。请注意, 在日志级别设置为插件后, 日志记录可用。NuGet 不会从命令行获取任何交互式输入。

当客户端使用获取身份验证凭据调用插件时, 插件需要符合交互交换机, 并遵循对话开关。

下表总结了插件对于所有组合的行为方式。

ISNONINTERACTIVE	CANSHOWDIALOG	
true	true	IsNonInteractive 开关优先于对话框开关。不允许插件弹出对话框。此组合仅对 .NET Framework 插件有效

ISNONINTERACTIVE	CANSHOWDIALOG	
true	False	IsNonInteractive 开关优先于对话框开关。不允许插件阻止。此组合仅适用于 .NET Core 插件
False	true	插件应该会显示一个对话框。此组合仅对 .NET Framework 插件有效
False	False	插件应/不能显示对话框。该插件应该使用设备流通过日志记录指令消息进行身份验证。此组合仅适用于 .NET Core 插件

请参阅以下规范，然后再编写插件。

- [NuGet 包下载插件](#)
- [NuGet 跨 x-plat 身份验证插件](#)

在 Visual Studio 中通过 NuGet 凭据提供程序对源进行身份验证

2020/3/3 • [Edit Online](#)

NuGet Visual Studio 扩展 3.6+ 支持凭据提供程序，这些提供程序允许 NuGet 使用经过身份验证的源。安装适用于 Visual Studio 的 NuGet 凭据提供程序后，NuGet Visual Studio 扩展会根据需要自动获取和刷新经过身份验证的源的凭据。

可在[VsCredentialProvider 示例](#)中找到示例实现。

在 Visual Studio 中，NuGet 使用内部 `VsCredentialProviderImporter`，这也会扫描插件凭据提供程序。这些插件凭据提供程序必须可被视为 `IVsCredentialProvider` 类型的 MEF 导出。

从 Visual Studio 中的 4.8+ NuGet 开始，还支持新的跨平台身份验证插件，但出于性能方面的原因，不建议采用这些方法。

NOTE

用于 Visual Studio 的 NuGet 凭据提供程序必须安装为常规 Visual Studio 扩展，并需要[Visual Studio 2017](#)或更高版本。

用于 Visual Studio 的 NuGet 凭据提供程序仅适用于 Visual Studio（不在 dotnet restore 或 nuget.exe 中）。有关 nuget.exe 的凭据提供程序，请参阅[Nuget.exe 凭据提供程序](#)。对于 dotnet 和 msbuild 中的凭据提供程序，请参阅[NuGet 跨平台插件](#)

为 Visual Studio 创建 NuGet 凭据提供程序

NuGet Visual Studio 扩展 3.6+ 实现了用于获取凭据的内部 CredentialService。CredentialService 包含内置和插件凭据提供程序的列表。每个提供程序按顺序尝试，直到获取凭据。

获取凭据时，凭据服务将按以下顺序尝试凭据提供程序，获取凭据后立即停止：

1. 将从 NuGet 配置文件（使用内置 `SettingsCredentialProvider`）中提取凭据。
2. 如果包源位于 Visual Studio Team Services 上，则将使用 `VisualStudioAccountProvider`。
3. 将按顺序尝试所有其他插件 Visual Studio 凭据提供程序。
4. 尝试按顺序使用所有 NuGet 跨平台凭据提供程序。
5. 如果尚未获取任何凭据，则系统会提示用户使用标准的基本身份验证对话框来输入凭据。

实现 `IVsCredentialProvider`.`GetCredentialsAsync`

若要为 Visual Studio 创建 NuGet 凭据提供程序，请创建一个 Visual Studio 扩展，该扩展公开实现 `IVsCredentialProvider` 类型的公共 MEF 导出，并遵循下面所述的原则。

```
public interface IVsCredentialProvider
{
    Task<ICredentials> GetCredentialsAsync(
        Uri uri,
        IWebProxy proxy,
        bool isProxyRequest,
        bool isRetry,
        bool nonInteractive,
        CancellationToken cancellationToken);
}
```

可在[VsCredentialProvider 示例](#)中找到示例实现。

适用于 Visual Studio 的每个 NuGet 凭据提供程序必须:

1. 确定是否可以在启动凭据获取之前为目标 URI 提供凭据。如果提供程序无法为目标源提供凭据，则它应返回 `null`。
2. 如果提供程序处理目标 URI 的请求，但无法提供凭据，则应引发异常。

Visual Studio 的自定义 NuGet 凭据提供程序必须实现[VisualStudio 包](#)中提供的 `IVsCredentialProvider` 接口。

GetCredentialAsync

参数	说明
<code>Uri uri</code>	正在为其请求凭据的包源 Uri。
<code>IWebProxy 代理</code>	网络上通信时要使用的 Web 代理。如果未配置代理身份验证，则为 <code>Null</code> 。
<code>bool isProxyRequest</code>	如果此请求将获取代理身份验证凭据，则为 <code>True</code> 。如果实现对于获取代理凭据无效，则应返回 <code>null</code> 。
<code>bool isRetry</code>	如果以前已为此 Uri 请求凭据，但提供的凭据不允许授权访问，则为 <code>True</code> 。
<code>bool 非交互式</code>	如果为 <code>true</code> ，则凭据提供程序必须禁止显示所有用户提示并改用默认值。
<code>CancellationToken cancellationToken</code>	应检查此取消标记以确定请求凭据的操作是否已取消。

返回值: 实现 [System.Net.ICredentials](#) 接口的凭据对象。

用 nuget.exe 凭据提供程序对源进行身份验证

2020/3/19 • [Edit Online](#)

在版本 3.3 为 nuget.exe 特定凭据提供程序添加了支持。自那时起, 在版本 4.8 支持跨所有命令行方案 (nuget.exe、dotnet.exe`msbuild.exe) 的凭据提供程序。

有关 nuget.exe 的所有身份验证方法的详细信息, 请参阅[使用已通过身份验证的源中的包](#)

nuget.exe 凭据提供程序发现

可以通过3种方式使用 nuget.exe 凭据提供程序:

- **全局**: 若要使凭据提供程序可用于在当前用户的配置文件下运行 nuget.exe 的所有实例, 请将其添加到 %LocalAppData%\NuGet\CredentialProviders 中。可能需要创建 CredentialProviders 文件夹。凭据提供程序可以安装在 CredentialProviders 文件夹的根目录或子文件夹中。如果凭据提供程序具有多个文件/程序集, 则可以使用子文件夹来保持提供程序的组织。
- **从环境变量中**: 凭据提供程序可以存储在任何位置, 并可通过将 %NUGET_CREDENTIALPROVIDERS_PATH% 环境变量设置为提供程序位置来访问 nuget.exe。如果有多个位置, 则此变量可以是以分号分隔的列表(例如 path1;path2)。
- 除了nuget.exe, 还可以将 nuget.exe 凭据提供程序放在与 nuget.exe 相同的文件夹中。

加载凭据提供程序时, nuget.exe 会按顺序搜索前面的任何名为 credentialprovider*.exe 的文件, 然后按找到的顺序加载这些文件。如果同一文件夹中存在多个凭据提供程序, 则按字母顺序加载它们。

创建 nuget.exe 凭据提供程序

凭据提供程序是一个命令行可执行文件, 它以 CredentialProvider*.exe 格式指定, 可收集输入, 根据需要获取凭据, 然后返回相应的退出状态代码和标准输出。

提供程序必须执行以下操作:

- 确定是否可以在启动凭据获取之前为目标 URI 提供凭据。如果不是, 则它应返回无凭据的状态代码1。
- 不修改 Nuget.Config (例如在此处设置凭据)。
- 自行处理 HTTP 代理配置, 因为 NuGet 不向插件提供代理信息。
- 使用 UTF-8 编码将 JSON 响应对象(见下文)写入 stdout, 从而将凭据或错误详细信息返回给 nuget.exe。
- (可选)向 stderr 发出附加跟踪日志记录。不应将任何机密写入到 stderr, 因为在详细级别 "正常" 或 "详细" 时, 此类跟踪会回显到控制台。
- 应忽略意外参数, 以提供与 NuGet 的未来版本的向前兼容性。

输入参数

参数	说明
Uri {value}	需要凭据的包源 URI。
NonInteractive	如果存在, 则提供程序不会发出交互式提示。

/	
IsRetry	如果存在，则指示此尝试将重试先前失败的尝试。提供程序通常使用此标志来确保它们绕过任何现有缓存，并在可能的情况下提示输入新凭据。
详细程度 {值}	如果存在，则为以下值之一：“normal”、“quiet”或“详细”。如果未提供任何值，则默认为“normal”。提供程序应使用此选项来指示要发出到标准错误流的可选日志记录级别。

退出代码

0	成功	已成功获取凭据并已将其写入 stdout。
1	ProviderNotApplicable	当前提供程序不提供给定 URI 的凭据。
2	失败	提供程序是给定 URI 的正确提供程序，但不能提供凭据。在这种情况下，nuget.exe 不会重试身份验证，并且会失败。典型的示例是用户取消交互式登录时。

标准输出

用户名	经过身份验证的请求的用户名。
Password	经过身份验证的请求的密码。
Message	有关响应的可选详细信息，仅用于显示故障情况下的其他详细信息。

示例 stdout:

```
{ "Username" : "freddy@example.com",
  "Password" : "bwm3bcx6txhprzmxhl2x63mdsul6grctazoomtdb6kfb0f7m3a3z",
  "Message"  : "" }
```

凭据提供程序故障排除

目前，NuGet 并不能直接支持调试自定义凭据提供程序；[问题 4598](#) 正在跟踪此工作。

也可以执行以下操作：

- 运行带有 `-verbosity` 开关的 nuget.exe 以检查详细的输出。
- 将调试消息添加到 `stdout` 在适当的位置。
- 请确保使用的是 nuget.exe 3.3 或更高版本。
- 在启动时将调试器附加到以下代码片段：

```
while (!Debugger.IsAttached)
{
    System.Threading.Thread.Sleep(100);
}
Debugger.Break();
```

Visual Studio 中的 NuGet API

2020/4/8 • [Edit Online](#)

除了 Visual Studio 中的包管理器 UI 和控制台, NuGet 还会通过 [Managed Extensibility Framework \(MEF\)](#) 导出一些有用服务。此接口允许 Visual Studio 中的其他组件与 NuGet 交互, 可用于安装和卸载包, 以及获取有关已安装包的信息。

从 NuGet 3.3+ 开始, NuGet 会导出以下所有服务, 它们全部驻留于 `NuGet.VisualStudio` 程序集中的 `NuGet.VisualStudio.dll` 命名空间:

- `IRegistryKey`: 检索注册表子项中的值的方法。
- `IVsPackageInstaller`: 将 NuGet 包安装到项目中的方法。
- `IVsPackageInstallerEvents`: 包安装/卸载的事件。
- `IVsPackageInstallerProjectEvents`: 包安装/卸载的 Batch 事件。
- `IVsPackageInstallerServices`: 检索当前解决方案中安装的包及检查项目中是否已安装给定包的方法。
- `IVsPackageManagerProvider`: 提供 NuGet 包的替代包管理器建议的方法。
- `IVsPackageMetadata`: 检索已安装包相关信息的方法。
- `IVsPackageProjectMetadata`: 检索正在执行 NuGet 操作的项目相关信息的方法。
- `IVsPackageRestorer`: 还原项目中已安装包的方法。
- `IVsPackageSourceProvider`: 检索 NuGet 包源列表的方法。
- `IVsPackageUninstaller`: 从项目中卸载 NuGet 包的方法。
- `IVsTemplateWizard`: 专为项目/项模板设计, 用以包括预安装的包; 此接口并非要从代码中调用, 且没有任何公共方法。

使用 NuGet 服务

1. 将 `NuGet.VisualStudio` 包安装到项目中, 其中包含 `NuGet.VisualStudio.dll` 程序集。

安装后, 包会自动将程序集引用的“嵌入互操作类型”属性设置为“True”。当用户更新到较新版本的 NuGet 时, 代码能够更适应版本更改。

WARNING

除了公共接口, 请不要在代码中使用任何其他类型, 也不要引用任何其他 NuGet 程序集, 包括 `NuGet.Core.dll`。

1. 要使用服务, 请通过 MEF 导入属性, 或通过 `IComponentModel` 服务将其导入。

```
//Using the Import attribute
[Import(typeof(IVsPackageInstaller2))]
public IVsPackageInstaller2 packageInstaller;
packageInstaller.InstallLatestPackage(null, currentProject,
    "Newtonsoft.Json", false, false);

//Using the IComponentModel service
var componentModel = (IComponentModel)GetService(typeof(SComponentModel));
IVsPackageInstallerServices installerServices =
    componentModel.GetService<IVsPackageInstallerServices>();

var installedPackages = installerServices.GetInstalledPackages();
```

对于引用，NuGet.VisualStudio 的源代码包含在 [NuGet.Clients 存储库](#) 中。

IRegistryKey 接口

```
/// <summary>
/// Specifies methods for manipulating a key in the Windows registry.
/// </summary>
public interface IRegistryKey
{
    /// <summary>
    /// Retrieves the specified subkey for read or read/write access.
    /// </summary>
    /// <param name="name">The name or path of the subkey to create or open.</param>
    /// <returns>The subkey requested, or null if the operation failed.</returns>
    IRegistryKey OpenSubKey(string name);

    /// <summary>
    /// Retrieves the value associated with the specified name.
    /// </summary>
    /// <param name="name">The name of the value to retrieve. This string is not case-sensitive.</param>
    /// <returns>The value associated with name, or null if name is not found.</returns>
    object GetValue(string name);

    /// <summary>
    /// Closes the key and flushes it to disk if its contents have been modified.
    /// </summary>
    void Close();
}
```

IVsPackageInstaller 接口

```
public interface IVsPackageInstaller
{
    /// <summary>
    /// Installs a single package from the specified package source.
    /// </summary>
    /// <param name="source">
    /// The package source to install the package from. This value can be <c>null</c>
    /// to indicate that the user's configured sources should be used. Otherwise,
    /// this should be the source path as a string. If the user has credentials
    /// configured for a source, this value must exactly match the configured source
    /// value.
    /// </param>
    /// <param name="project">The target project for package installation.</param>
    /// <param name="packageId">The package ID of the package to install.</param>
    /// <param name="version">
    /// The version of the package to install. <c>null</c> can be provided to
    /// install the latest version of the package.
    /// </param>
    /// <param name="ignoreDependencies">
    /// A boolean indicating whether or not to ignore the package's dependencies
    /// during installation.
    /// </param>
    void InstallPackage(string source, Project project, string packageId, Version version, bool
ignoreDependencies);

    /// <summary>
    /// Installs a single package from the specified package source.
    /// </summary>
    /// <param name="source">
    /// The package source to install the package from. This value can be <c>null</c>
    /// to indicate that the user's configured sources should be used. Otherwise,
```

```

///>     /// this should be the source path as a string. If the user has credentials
///>     /// configured for a source, this value must exactly match the configured source
///>     /// value.
///>     /// </param>
///>     /// <param name="project">The target project for package installation.</param>
///>     /// <param name="packageId">The package ID of the package to install.</param>
///>     /// <param name="version">
///>     /// The version of the package to install. <c>null</c> can be provided to
///>     /// install the latest version of the package.
///>     /// </param>
///>     /// <param name="ignoreDependencies">
///>     /// A boolean indicating whether or not to ignore the package's dependencies
///>     /// during installation.
///>     /// </param>
void InstallPackage(string source, Project project, string packageId, string version, bool
ignoreDependencies);

///>     /// <summary>
///>     /// Installs a single package from the specified package source.
///>     /// </summary>
///>     /// <param name="repository">The package repository to install the package from.</param>
///>     /// <param name="project">The target project for package installation.</param>
///>     /// <param name="packageId">The package id of the package to install.</param>
///>     /// <param name="version">
///>     /// The version of the package to install. <c>null</c> can be provided to
///>     /// install the latest version of the package.
///>     /// </param>
///>     /// <param name="ignoreDependencies">
///>     /// A boolean indicating whether or not to ignore the package's dependencies
///>     /// during installation.
///>     /// </param>
///>     /// <param name="skipAssemblyReferences">
///>     /// A boolean indicating if assembly references from the package should be
///>     /// skipped.
///>     /// </param>
void InstallPackage(IPackageRepository repository, Project project, string packageId, string version, bool
ignoreDependencies, bool skipAssemblyReferences);

///>     /// <summary>
///>     /// Installs one or more packages that exist on disk in a folder defined in the registry.
///>     /// </summary>
///>     /// <param name="keyName">
///>     /// The registry key name (under NuGet's repository key) that defines the folder on disk
///>     /// containing the packages.
///>     /// </param>
///>     /// <param name="isPreUnzipped">
///>     /// A boolean indicating whether the folder contains packages that are
///>     /// pre-unzipped.
///>     /// </param>
///>     /// <param name="skipAssemblyReferences">
///>     /// A boolean indicating whether the assembly references from the packages
///>     /// should be skipped.
///>     /// </param>
///>     /// <param name="project">The target project for package installation.</param>
///>     /// <param name="packageVersions">
///>     /// A dictionary of packages/versions to install where the key is the package id
///>     /// and the value is the version.
///>     /// </param>
///>     /// <remarks>
///>     /// If any version of the package is already installed, no action will be taken.
///>     /// </para>
///>     /// Dependencies are always ignored.
///>     /// </para>
///>     /// </remarks>
void InstallPackagesFromRegistryRepository(string keyName, bool isPreUnzipped, bool skipAssemblyReferences,
Project project, IDictionary<string, string> packageVersions);

///>     /// <summary>
///>     /// Installs one or more packages that exist on disk in a folder defined in the registry.

```

```
/// </summary>
/// <param name="keyName">
/// The registry key name (under NuGet's repository key) that defines the folder on disk
/// containing the packages.
/// </param>
/// <param name="isPreUnzipped">
/// A boolean indicating whether the folder contains packages that are
/// pre-unzipped.
/// </param>
/// <param name="skipAssemblyReferences">
/// A boolean indicating whether the assembly references from the packages
/// should be skipped.
/// </param>
/// <param name="ignoreDependencies">A boolean indicating whether the package's dependencies should be
ignored</param>
/// <param name="project">The target project for package installation.</param>
/// <param name="packageVersions">
/// A dictionary of packages/versions to install where the key is the package id
/// and the value is the version.
/// </param>
/// <remarks>
/// If any version of the package is already installed, no action will be taken.
/// </remarks>
void InstallPackagesFromRegistryRepository(string keyName, bool isPreUnzipped, bool skipAssemblyReferences,
bool ignoreDependencies, Project project, IDictionary<string, string> packageVersions);

/// <summary>
/// Installs one or more packages that are embedded in a Visual Studio Extension Package.
/// </summary>
/// <param name="extensionId">The Id of the Visual Studio Extension Package.</param>
/// <param name="isPreUnzipped">
/// A boolean indicating whether the folder contains packages that are
/// pre-unzipped.
/// </param>
/// <param name="skipAssemblyReferences">
/// A boolean indicating whether the assembly references from the packages
/// should be skipped.
/// </param>
/// <param name="project">The target project for package installation</param>
/// <param name="packageVersions">
/// A dictionary of packages/versions to install where the key is the package id
/// and the value is the version.
/// </param>
/// <remarks>
/// If any version of the package is already installed, no action will be taken.
/// </para>
/// Dependencies are always ignored.
/// </para>
/// </remarks>
void InstallPackagesFromVSExtensionRepository(string extensionId, bool isPreUnzipped, bool
skipAssemblyReferences, Project project, IDictionary<string, string> packageVersions);

/// <summary>
/// Installs one or more packages that are embedded in a Visual Studio Extension Package.
/// </summary>
/// <param name="extensionId">The Id of the Visual Studio Extension Package.</param>
/// <param name="isPreUnzipped">
/// A boolean indicating whether the folder contains packages that are
/// pre-unzipped.
/// </param>
/// <param name="skipAssemblyReferences">
/// A boolean indicating whether the assembly references from the packages
/// should be skipped.
/// </param>
/// <param name="ignoreDependencies">A boolean indicating whether the package's dependencies should be
ignored</param>
/// <param name="project">The target project for package installation</param>
/// <param name="packageVersions">
/// A dictionary of packages/versions to install where the key is the package id
```

```

/// A parameter of type string is used to indicate which key to use for the package -- 
/// and the value is the version.
/// </param>
/// <remarks>
/// If any version of the package is already installed, no action will be taken.
/// </remarks>
void InstallPackagesFromVSExtensionRepository(string extensionId, bool isPreUnzipped, bool
skipAssemblyReferences, bool ignoreDependencies, Project project, IDictionary<string, string> packageVersions);
}

```

IVsPackageInstallerEvents 接口

```

public interface IVsPackageInstallerEvents
{
    /// <summary>
    /// Raised when a package is about to be installed into the current solution.
    /// </summary>
    event VsPackageEventHandler PackageInstalling;

    /// <summary>
    /// Raised after a package has been installed into the current solution.
    /// </summary>
    event VsPackageEventHandler PackageInstalled;

    /// <summary>
    /// Raised when a package is about to be uninstalled from the current solution.
    /// </summary>
    event VsPackageEventHandler PackageUninstalling;

    /// <summary>
    /// Raised after a package has been uninstalled from the current solution.
    /// </summary>
    event VsPackageEventHandler PackageUninstalled;

    /// <summary>
    /// Raised after a package has been installed into a project within the current solution.
    /// </summary>
    event VsPackageEventHandler PackageReferenceAdded;

    /// <summary>
    /// Raised after a package has been uninstalled from a project within the current solution.
    /// </summary>
    event VsPackageEventHandler PackageReferenceRemoved;
}

```

IVsPackageInstallerProjectEvents 接口

```

public interface IVsPackageInstallerProjectEvents
{
    /// <summary>
    /// Raised before any IVsPackageInstallerEvents events are raised for a project.
    /// </summary>
    event VsPackageProjectEventHandler BatchStart;

    /// <summary>
    /// Raised after all IVsPackageInstallerEvents events are raised for a project.
    /// </summary>
    event VsPackageProjectEventHandler BatchEnd;
}

```

IVsPackageInstallerServices 接口

```

public interface IVsPackageInstallerServices
{
    // IMPORTANT: do NOT rearrange the methods here. The order is important to maintain
    // backwards compatibility with clients that were compiled against old versions of NuGet.

    /// <summary>
    /// Get the list of NuGet packages installed in the current solution.
    /// </summary>
    IEnumerable<IVsPackageMetadata> GetInstalledPackages();

    /// <summary>
    /// Checks if a NuGet package with the specified Id is installed in the specified project.
    /// </summary>
    /// <param name="project">The project to check for NuGet package.</param>
    /// <param name="id">The id of the package to check.</param>
    /// <returns><c>true</c> if the package is install. <c>false</c> otherwise.</returns>
    bool IsPackageInstalled(Project project, string id);

    /// <summary>
    /// Checks if a NuGet package with the specified Id and version is installed in the specified project.
    /// </summary>
    /// <param name="project">The project to check for NuGet package.</param>
    /// <param name="id">The id of the package to check.</param>
    /// <param name="version">The version of the package to check.</param>
    /// <returns><c>true</c> if the package is install. <c>false</c> otherwise.</returns>
    bool IsPackageInstalled(Project project, string id, SemanticVersion version);

    /// <summary>
    /// Checks if a NuGet package with the specified Id and version is installed in the specified project.
    /// </summary>
    /// <param name="project">The project to check for NuGet package.</param>
    /// <param name="id">The id of the package to check.</param>
    /// <param name="versionString">The version of the package to check.</param>
    /// <returns><c>true</c> if the package is install. <c>false</c> otherwise.</returns>
    /// <remarks>
    /// The reason this method is named IsPackageInstalledEx, instead of IsPackageInstalled, is that
    /// when client project compiles against this assembly, the compiler would attempt to bind against
    /// the other overload which accepts SemanticVersion and would require client project to reference
    NuGet.Core.
    /// </remarks>
    bool IsPackageInstalledEx(Project project, string id, string versionString);

    /// <summary>
    /// Get the list of NuGet packages installed in the specified project.
    /// </summary>
    /// <param name="project">The project to get NuGet packages from.</param>
    IEnumerable<IVsPackageMetadata> GetInstalledPackages(Project project);
}

```

IVsPackageManagerProvider 接口

```
public interface IVsPackageManagerProvider
{
    /// <summary>
    /// Localized display package manager name.
    /// </summary>
    string PackageManagerName { get; }

    /// <summary>
    /// Package manager unique id.
    /// </summary>
    string PackageManagerId { get; }

    /// <summary>
    /// The tool tip description for the package
    /// </summary>
    string Description { get; }

    /// <summary>
    /// Check if a recommendation should be surfaced for an alternate package manager.
    /// This code should not rely on slow network calls, and should return rapidly.
    /// </summary>
    /// <param name="packageId">Current package id</param>
    /// <param name="projectName">Unique project name for finding the project through VS dte</param>
    /// <param name="token">Cancellation Token</param>
    /// <returns>return true if need to direct to integrated package manager for this package</returns>
    Task<bool> CheckForPackageAsync(string packageId, string projectName, CancellationToken token);

    /// <summary>
    /// This Action should take the user to the other package manager.
    /// </summary>
    /// <param name="packageId">Current package id</param>
    /// <param name="projectName">Unique project name for finding the project through VS dte</param>
    void GoToPackage(string packageId, string projectName);
}
```

IVsPackageMetadata 接口

```

public interface IVsPackageMetadata
{
    /// <summary>
    /// Id of the package.
    /// </summary>
    string Id { get; }

    /// <summary>
    /// Version of the package.
    /// </summary>
    /// <remarks>
    /// Do not use this property because it will require referencing NuGet.Core.dll assembly. Use the
    VersionString
    /// property instead.
    /// </remarks>
    [Obsolete("Do not use this property because it will require referencing NuGet.Core.dll assembly. Use the
    VersionString property instead.")]
    NuGet.SemanticVersion Version { get; }

    /// <summary>
    /// Title of the package.
    /// </summary>
    string Title { get; }

    /// <summary>
    /// Description of the package.
    /// </summary>
    string Description { get; }

    /// <summary>
    /// The authors of the package.
    /// </summary>
    IEnumerable<string> Authors { get; }

    /// <summary>
    /// The location where the package is installed on disk.
    /// </summary>
    string InstallPath { get; }

    // IMPORTANT: This property must come LAST, because it was added in 2.5. Having it declared
    // LAST will avoid breaking components that compiled against earlier versions which doesn't
    // have this property.
    /// <summary>
    /// The version of the package.
    /// </summary>
    /// <remarks>
    /// Use this property instead of the Version property because with the type string,
    /// it doesn't require referencing NuGet.Core.dll assembly.
    /// </remarks>
    string VersionString { get; }
}

```

IVsPackageProjectMetadata 接口

```

public interface IVsPackageProjectMetadata
{
    /// <summary>
    /// Unique batch id for batch start/end events of the project.
    /// </summary>
    string BatchId { get; }

    /// <summary>
    /// Name of the project.
    /// </summary>
    string ProjectName { get; }
}

```

IVsPackageRestorer 接口

```

public interface IVsPackageRestorer
{
    /// <summary>
    /// Returns a value indicating whether the user consent to download NuGet packages
    /// has been granted.
    /// </summary>
    /// <returns>true if the user consent has been granted; otherwise, false.</returns>
    bool IsUserConsentGranted();

    /// <summary>
    /// Restores NuGet packages installed in the given project within the current solution.
    /// </summary>
    /// <param name="project">The project whose NuGet packages to restore.</param>
    void RestorePackages(Project project);
}

```

IVsPackageSourceProvider 接口

```

public interface IVsPackageSourceProvider
{
    /// <summary>
    /// Provides the list of package sources.
    /// </summary>
    /// <param name="includeUnOfficial">Unofficial sources will be included in the results</param>
    /// <param name="includeDisabled">Disabled sources will be included in the results</param>
    /// <returns>Key: source name Value: source URI</returns>
    IEnumerable<KeyValuePair<string, string>> GetSources(bool includeUnOfficial, bool includeDisabled);

    /// <summary>
    /// Raised when sources are added, removed, disabled, or modified.
    /// </summary>
    event EventHandler SourcesChanged;
}

```

IVsPackageUninstaller 接口

```
public interface IVsPackageUninstaller
{
    /// <summary>
    /// Uninstall the specified package from a project and specify whether to uninstall its dependency packages
    /// too.
    /// </summary>
    /// <param name="project">The project from which the package is uninstalled.</param>
    /// <param name="packageId">The package to be uninstalled</param>
    /// <param name="removeDependencies">
    /// A boolean to indicate whether the dependency packages should be
    /// uninstalled too.
    /// </param>
    void UninstallPackage(Project project, string packageId, bool removeDependencies);
}
```

IVsTemplateWizard 接口

```
/// <summary>
/// Defines the logic for a template wizard extension.
/// </summary>

public interface IVsTemplateWizard : IWizard
{}
```

NuGet 对 Visual Studio 项目系统的支持

2020/4/8 • [Edit Online](#)

为在 Visual Studio 中支持第三方项目类型, NuGet 3.x+ 支持[公共项目系统 \(CPS\)](#), 且 NuGet 3.2+ 也支持非 CPS 项目系统。

若要与 NuGet 集成, 项目系统必须播发自己对本主题中介绍的所有项目功能的支持。

NOTE

不声明项目实际上没有的功能, 以便在项目中安装程序包。除 NuGet 客户端之外, Visual Studio 的许多功能和其他扩展还依赖于项目功能。项目的错误播发功能会导致这些组件出现故障, 同时对用户体验产生负面影响。

播发项目功能

NuGet 客户端基于[项目功能](#)确定与项目类型兼容的包, 如下表所述。

“	“
AssemblyReferences	指示项目支持程序集引用(与 WinRTReferences 不同)。
DeclaredSourceItems	指示项目是一个典型的 MSBuild 项目(不是 DNX), 因为它在项目本身中声明源项目。
UserSourceItems	指示允许用户向其项目添加任意文件。

有关基于 CPS 的项目系统, 本节剩余部分中介绍了项目功能的实现详细信息。请参阅[声明 CPS 项目中的项目功能](#)。

实现 VsProjectCapabilitiesPresenceChecker

`VsProjectCapabilitiesPresenceChecker` 类必须实现 `IVsBooleanSymbolPresenceChecker` 接口, 定义如下:

```

public interface IVsBooleanSymbolPresenceChecker
{
    /// <summary>
    /// Checks whether the symbols defined may have changed since the last time
    /// this method was called.
    /// </summary>
    /// <param name="versionObject">
    /// The response version object assigned at the last call.
    /// May be null to get the initial version.
    /// At the conclusion of this method call, the reference may be changed so that on a subsequent call
    /// we know what version was last observed. The caller should treat this value as an opaque object,
    /// and should not assume any significance from whether the pointer changed or not.
    /// </param>
    /// <returns>
    /// <c>true</c> if the symbols defined may have changed since the last call to this method
    /// or if <paramref name="versionObject"/> was <c>null</c> upon entering this method.
    /// <c>false</c> if the responses would all be the same.
    /// </returns>
    bool HasChangedSince(ref object versionObject);

    /// <summary>
    /// Checks for the presence of a given symbol.
    /// </summary>
    /// <param name="symbol">The symbol to check for.</param>
    /// <returns><c>true</c> if the symbol is present; <c>false</c> otherwise.</returns>
    bool IsSymbolPresent(string symbol);
}

```

此接口的示例实现是：

```

class VsProjectCapabilitiesPresenceChecker : IVsBooleanSymbolPresenceChecker
{
    /// <summary>
    /// The set of capabilities this project system actually has.
    /// This should be kept current, and honestly reflect what the project can do.
    /// </summary>
    private static readonly HashSet<string> ActualProjectCapabilities = new HashSet<string>
(StringComparer.OrdinalIgnoreCase)
    {
        "AssemblyReferences",
        "DeclaredSourceItems",
        "UserSourceItems",
    };

    public bool HasChangedSince(ref object versionObject)
    {
        // If your project capabilities do not change over time while the project is open,
        // you may simply `return false;` from your `HasChangedSince` method.
        return false;
    }

    public bool IsSymbolPresent(string symbol)
    {
        return ActualProjectCapabilities.Contains(symbol);
    }
}

```

请记住，[基于项目系统实际支持的内容添加/删除](#) `ActualProjectCapabilities` 集的功能。若要了解完整说明，请参阅[项目功能文档](#)。

对查询作出响应

项目通过 `VSHPROPID_ProjectCapabilitiesChecker` 支持 `IVsHierarchy::GetProperty` 属性来声明此功能。它应返回

`Microsoft.VisualStudio.Shell.Interop.IVsBooleanSymbolPresenceChecker` 的实例,

`Microsoft.VisualStudio.Shell.Interop.14.0.DesignTime.dll` 程序集中对其进行定义。通过安装[相应的 NuGet 包](#)引用此程序集。

例如, 可将以下 `case` 语句添加到 `IVsHierarchy::GetProperty` 方法的 `switch` 语句中:

```
case __VSHPROPID8.VSHPROPID_ProjectCapabilitiesChecker:  
    propVal = new VsProjectCapabilitiesPresenceChecker();  
    return VSConstants.S_OK;
```

DTE 支持

NuGet 通过调用到 [DTE](#)(Visual Studio 的顶级自动化接口), 使项目系统添加引用、内容项和 MSBuild 导入。DTE 是一组可能已经实现的 COM 接口。

如果项目类型基于 CPS, 则已实现 DTE。

Visual Studio 模板中的包

2020/4/8 • [Edit Online](#)

创建项目或项时，Visual Studio 项目和项模板通常需要确保某些包已安装。例如，ASP.NET MVC 3 模板安装 jQuery、Modernizr 和其他包。

若要支持此功能，模板作者可指示 NuGet 安装必需的包，而不是单独的库。此后，开发人员可随时轻松更新这些包。

要了解有关创作模板本身的详细信息，请参阅[如何：创建项目模板或创建自定义项目和项模板](#)。

本节其余部分介绍创作模板以正确添加 NuGet 包时应采取的具体步骤。

- [将包添加到模板](#)
- [最佳做法](#)

有关示例，请参阅[NuGetInVsTemplates 示例](#)。

将包添加到模板

实例化模板后，会调用[模板向导](#)加载要安装的包列表以及有关在何处查找这些包的信息。包可嵌入 VSIX，嵌入模板，或者位于本地硬盘上（在这种情况下，请使用注册表项引用文件路径）。本节稍后将提供这些位置的详细信息。

预安装的包使用[模板向导](#)工作。模板实例化后会调用特殊向导。该向导会加载需要安装的包列表，并将该信息传递到相应的 NuGet API。

将包添加到模板的步骤：

1. 在 `vstemplate` 文件中，通过添加 `WizardExtension` 元素向 NuGet 模板向导添加引用：

```
<WizardExtension>
  <Assembly>NuGet.VisualStudio.Interop, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a</Assembly>
  <FullClassName>NuGet.VisualStudio.TemplateWizard</FullClassName>
</WizardExtension>
```

`NuGet.VisualStudio.Interop.dll` 为仅包含 `TemplateWizard` 类的程序集，该类是调用 `NuGet.VisualStudio.dll` 中实际实现的简单包装器。程序集版本永远不会更改，因此项目/项模板可继续用于新版本的 NuGet。

2. 在项目中添加要安装的包列表：

```
<WizardData>
  <packages>
    <package id="jQuery" version="1.6.2" />
  </packages>
</WizardData>
```

向导支持多个 `<package>` 元素，从而支持多个包源。`id` 和 `version` 属性都是必需的，这意味着即使可使用较新版本，仍将安装包的特定版本。这可以防止包更新中断模板，并将是否更新包的选择权留给使用模板的开发人员。

3. 指定 NuGet 可找到包的存储库，如以下各节中所述。

VSIX 包存储库

建议用于 Visual Studio 项目/项模板的部署方法是 [VSIX 扩展](#)，因为借助它，可将多个项目/项模板打包在一起，而且开发人员可使用 VS 扩展管理器或 Visual Studio 库轻松发现模板。也可使用 [Visual Studio 扩展管理器自动更新机制](#) 轻松部署扩展更新。

VSIX 本身可用作模板所需包的源：

1. 修改 `<packages>` 文件中的 `.vstemplate` 元素，如下所示：

```
<packages repository="extension" repositoryId="MyTemplateContainerExtensionId">
    <!-- ... -->
</packages>
```

`repository` 属性将存储库类型指定为 `extension`，而 `repositoryId` 是 VSIX 本身的唯一标识符（它是扩展的 `ID` 文件中 `vsixmanifest` 属性的值，请参阅 [VSIX 扩展架构 2.0 参考](#)）。

2. 将 `nupkg` 文件放置于 VSIX 中名为 `Packages` 的文件夹中。
3. 添加必要的包文件作为 `<Asset>` 文件中的 `vsixmanifest`（请参阅 [VSIX 扩展架构 2.0 参考](#)）：

```
<Asset Type="Moq.4.0.10827.nupkg" d:Source="File" Path="Packages\Moq.4.0.10827.nupkg"
d:VsixSubPath="Packages" />
```

4. 请注意，可在项目模板所在的同一 VSIX 中提供包，或者可将其放置在单独的 VSIX 中（如果这样对于你的方案更合理）。但是，请不要引用任何你无法控制的 VSIX，因为对该扩展进行更改可能会中断模板。

模板包存储库

如果仅分发一个项目/项模板且无需将多个模板打包在一起，则可使用更为简单但受限更多的方法，该方法直接在项目/项模板 ZIP 文件中添加包：

1. 修改 `<packages>` 文件中的 `.vstemplate` 元素，如下所示：

```
<packages repository="template">
    <!-- ... -->
</packages>
```

`repository` 属性具有值 `template`，不需要 `repositoryId` 属性。

2. 将包放置在项目/项模板 ZIP 文件的根文件夹中。

请注意，如果一个或多个包对模板通用，则在包含多个模板的 VSIX 中使用此方法会导致不必要的膨胀。在这种情况下，请将 [VSIX 用作存储库](#)，如上一节中所述。

注册表指定的文件夹路径

使用 MSI 安装的 SDK 可直接在开发人员计算机上安装 NuGet 包。这使其在使用项目或项模板时立即可用，而不必在此期间进行提取。ASP.NET 模板使用此方法。

1. 利用 MSI 将包安装到计算机。可仅安装 `.nupkg` 文件，也可将其与扩展内容一起安装，这可在使用模板时减少额外的步骤。在这种情况下，请遵循 NuGet 的标准文件夹结构，其中 `.nupkg` 文件位于根文件夹中，然后每个包都有子文件夹，子文件夹使用 id/版本对作为名称。

2. 写入注册表项以标识包位置：

- 项位置：如果是计算机范围内的 `HKEY_LOCAL_MACHINE\SOFTWARE[\Wow6432Node]\NuGet\Repository`，或为每个用户安装的模板和包，则使用 `HKEY_CURRENT_USER\Software\NuGet\Repository`
- 项名称：使用唯一的名称。例如，适用于 VS 2012 的 ASP.NET MVC 4 模板使用 `AspNetMvc4VS11`。

- 值：包文件夹的完整路径。
3. 在 `<packages>` 文件的 `.vstemplate` 元素中，添加属性 `repository="registry"` 并在 `keyName` 属性中指定注册表项名称。
- 如果已预先解压缩包，请使用 `isPreunzipped="true"` 属性。
 - (NuGet 3.2+) 如果希望在包安装结束时强制执行设计时生成，请添加 `forceDesignTimeBuild="true"` 属性。
 - 作为优化，添加 `skipAssemblyReferences="true"`，因为模板本身已包括必要的引用。

```
<packages repository="registry" keyName="AspNetMvc4VS11" isPreunzipped="true">
    <package id="EntityFramework" version="5.0.0" skipAssemblyReferences="true" />
    <!-- ... -->
</packages>
```

最佳实践

1. 通过在 VSIX 清单中添加对其的引用，声明 NuGet VSIX 上的依赖项：

```
<Reference Id="NuPackToolsVsix.Microsoft.67e54e40-0ae3-42c5-a949-fdd5739e7a5"
MinVersion="1.7.30402.9028">
    <Name>NuGet Package Manager</Name>
    <MoreInfoUrl>http://docs.microsoft.com/nuget/</MoreInfoUrl>
</Reference>
<!-- ... -->
```

2. 通过在 [文件中包括](#) `<PromptForSaveOnCreation>true</PromptForSaveOnCreation>` `.vstemplate`，要求在创建时保存项目/项模板。
3. 模板不包括 `packages.config` 文件，也不包括安装 NuGet 包时将添加的任何引用或内容。

NuGet 治理

2020/4/8 • [Edit Online](#)

本文档基于牛津大学的[仁慈独裁者治理模型](#)。它根据 [Creative Commons Attribution-ShareAlike 2.0 UK: England & Wales License](#) 获得许可。

NuGet 项目由仁慈独裁者领导并由社区进行管理。也就是说，社区积极参与项目的日常维护，但总体策略规划由仁慈独裁者制定。如果出现争议，仁慈独裁者拥有最终裁量权。

仁慈独裁者的职责是解决社区内部争议，并确保项目能够协调开展。而社区的职责是通过积极参与和贡献来指导仁慈独裁者的决策。

角色和职责

此处介绍四种角色：仁慈独裁者、提交者、参与者和用户。

仁慈的独裁者

NuGet 核心团队自任命为仁慈独裁者或项目领导。但是，由于社区总是会出现分歧，因此团队应完全对社区负责。项目领导应了解整个社区，并努力尽量满足发生冲突的需求，同时确保项目长期开展下去。

在许多方面，仁慈独裁者的角色更偏向协调，而不是独裁。关键的一点是，随着项目展开，确保相关人员受到积极影响，社区在项目领导的愿景下共同协作。领导的职责是确保提交者（见下文）代表项目做出正确的决策。一般而言，只要提交者与项目的策略保持一致，项目领导会允许他们按自己的方式行事。

此外，.NET Foundation 员工将项目领导视为 NuGet 的主要或首个联系点，用于域注册和技术服务（如代码签名）等业务操作。

提交者

提交者是为 NuGet 持续做出有价值贡献的参与者，由仁慈独裁者任命。经任命后，将代码直接写入存储库及筛选其他人的贡献均依赖于提交者。提交者通常是开发人员，但可以通过其他方式参与。

通常，提交者专注于项目的特定方面，其具有一定的专业知识和经验，因此受到社区和项目领导的尊重。提交者并非一种正式角色，它仅仅是影响力大的社区成员在项目领导向其寻求指导和支持时担当的职位。

从 NuGet 的总体方向来看，提交者不具备权威。但项目领导会听取他们的意见。提交者的职责是确保领导了解社区的需求和集体目标，并帮助开发或引导对项目的适当贡献。通常，提交者可对其特定负责部分进行非正式管理，并且分配有可直接修改源代码的特定部分的权限。也就是说，尽管提交者没有明确的决策权，但他们通常会发现其操作与领导做出的决策一致。

参与者

参与者是向 NuGet 提交修补程序的社区成员。这些修补程序可能只发生一次或者不断发生。根据预期，参与者提交的修补程序起初很小，随着参与者、提交者和项目领导对其质量建立起信心而逐渐变大。参与者在关联的产品发布说明文档中获得认可。

将参与者的第一个修补程序放入存储库前，他们必须向 .NET Foundation 签署[参与者许可协议](#)或分配协议。修补程序可提交和讨论，但如果没有任何相应的书面批准，实际上不能提交到存储库。若要获取参与者许可协议，请通过电子邮件将请求发送至 contributions@nuget.org。

若要成为参与者，请向以下存储库之一提交拉取请求：

- [NuGet 客户端](#)
- [NuGet 库](#)

- [NuGet Docs](#)

提交拉取请求的详细过程因存储库而异：

- [NuGet 客户端和 NuGet 库的参与说明](#)
- [NuGet Docs 的参与说明](#)

用户

用户是作为包使用者和/或创建者需要并使用 NuGet 的社区成员。用户是社区最重要的成员：没有他们，项目就没有用。任何人都可成为用户；没有特定要求。

应鼓励用户尽量参与 NuGet 和社区的活动。用户的参与使项目团队确保满足用户的需求。常见的用户活动包括但不限于以下几种：

- 提倡使用该项目
- 从新用户的角度告知开发人员项目的优缺点
- 提供精神支持(感谢的话语为开发人员带来工作的动力)
- 编写文档和教程
- 填写 bug 报告和功能请求
- 参与社区活动，如 bug 清除
- 参与讨论板或论坛

通常，持续参与项目及其社区的用户的参与度会不断提高。这类用户以后可以成为参与者，如上文所述。

特殊情况下的包继承

在 NuGet 帐户持有者无行为能力或死亡等不幸情况下，我们将与社区协作，将适当的所有者添加到包，其中上述帐户对包具有唯一所有权，并且包根据 [OSI 批准许可](#) 发布。若要请求所有权，必须向我们发送以下文档：

1. 政府颁发的带照片身份证复印件。
2. 可证明以前帐户持有者状态的以下文档之一：
 - 政府颁发的官方死亡证明（如果以前的帐户持有者已死亡），或
 - 公证文件，例如负责照顾无行为能力帐户持有者的医疗专业人员签名的证明。
3. 可证明你拥有所有权的以下文档之一：
 - 可证明你是帐户持有者的依然健在的配偶/伴侣的结婚证，
 - 签名委托书，
 - 将你命名为执行人或受益人的遗嘱或信托文件复印件，
 - 帐户持有者的出生证明（如果你是其父母），或
 - 监护人身份书面证明（如果你是帐户持有者的法定监护人）。

如果你发现自己需要使用此政策，请通过电子邮件将包的 ID 和版本发送至 support@nuget.org。

透明度

在开放源代码项目治理中构建社区信任对其成功至关重要。为此，决策制定必须采取透明、开放的方式。有关项目方向的讨论必须公开进行。社区不应对仁慈独裁者的决策措手不及。此外，有关项目决策的讨论必须存档，以便社区成员可了解决策及其上下文的整个历史记录。

NuGet 生态系统概述

2020/4/8 • [Edit Online](#)

自 2010 年推出以来, NuGet 提供了良好的机遇来改善和自动化开发过程的不同方面。

由于 NuGet 是根据限制性弱的 [Apache v2 许可证](#) 许可的开放源代码, 因此其他项目可以利用 NuGet, 公司也可在其产品中生成对 NuGet 的支持。无论是对于开源项目还是企业应用程序开发, NuGet 和以 NuGet 为基础生成的其他应用程序提供了广泛的工具生态系统, 以便改善软件开发过程。

由于开发人员的贡献, 所有这些项目都能够进行创新。正如参与到 NuGet 本身中去, 也通过报告缺陷和新功能创意、提供反馈、编写文档以及参与代码, 尽力对这些项目做出贡献。

.NET Foundation 项目

NuGet 为 Microsoft 开发平台提供了免费的开放源代码程序包管理系统。它包含一些客户端工具以及一组服务, 这些工具和服务构成了[正式 NuGet 库](#)。一经组合, 便形成受 .NET Foundation 控制的 NuGet 项目。

NuGet 组织包含 GitHub 上的各种存储库。<https://github.com/Nuget/Home> 概述了所有存储库以及可在何处找到各种 NuGet 组件。

Microsoft 项目

Microsoft 对 NuGet 的开发做出了巨大的贡献。Microsoft 员工做出的所有贡献也均为开放源代码, 并将捐赠(包括版权)给 .NET Foundation。

非 Microsoft 项目

许多其他个人和公司也为 NuGet 生态系统做出了重要贡献。此处所列每个项目的许可证可能与核心 NuGet 组件的不同, 因此请在使用前确认许可条款是否可接受:

- [AppVeyor CI](#)
- [Artifactory](#)
- [BoxStarter](#)
- [Chocolatey](#)
- [CoApp](#)
- [JetBrains ReSharper](#)
- [JetBrains TeamCity](#)
- [Klondike](#)
- [MinimalNugetServer](#)
- [MyGet\(或 NuGet 即服务\)](#)
- [NuGet 包资源管理器](#)
- [NuGet 服务器](#)
- [OctopusDeploy](#)
- [Paket](#)
- [ProGet \(Inedo\)](#)
- [scriptcs](#)
- [SharpDevelop](#)
- [Sonatype Nexus](#)
- [SymbolSource](#)

- [Xamarin 和 MonoDevelop](#)

其他基于 NuGet 的实用程序

以下是在 NuGet 基础上生成的工具和实用程序：

- [Glimpse 扩展](#)(插件均为包)
- [NuGetMustHaves.com](#)
- [Orchard](#)(CMS 模块从 Orchard 库中托管的 v1 NuGet 源提取)
- [NuGet 服务器的 Java 实现](#)
- [NuGetLatest](#)(发表新包发布推文的 Twitter 自动程序)
- [DefinitelyTyped](#)(发布到 NuGet 的[自动化](#) TypeScript 类型定义)

培训材料和参考资料

使用新工具或技术通常伴有学习曲线。幸运的是，NuGet 完全没有陡峭的学习曲线！事实上，任何人都可以快速[入门包的使用](#)。

即便如此，创作包(尤其是好的包)以及在自动生成和部署过程中利用 NuGet，需要在以下资源上花费更多时间：

- [NuGet 博客](#)
- [Twitter 上的 NuGet 团队, @nuget](#)
- 书籍：
 - [Apress Pro NuGet](#)
 - [NuGet 2 基础知识](#)

单个包的文档

[NuDoq](#) 提供了适用于 NuGet 包的直接访问、更新和文档。

NuDoq 定期轮询 nuget.org 库服务器以获取最新的包更新，解压缩并处理库文档文件，以及相应地对站点进行更新。

添加项目

如果有 NuGet 生态系统项目，且是对本页有价值的补充，请向本页提交带编辑的拉取请求。

用户数据请求

2020/4/8 • [Edit Online](#)

nuget.org 用户可通过 [nuget.org](#) 提交信息删除请求和信息导出请求。这两种类型均以支持请求的形式提交，并由 nuget.org 管理员在 30 天内执行。

可通过 nuget.org 直接访问以下用户数据：

- 与电子邮件地址、登录帐户、个人资料图片、电子邮件通知设置等数据相关的帐户
- 拥有的 API 密钥
- 拥有包列表

此数据不包含在通过支持请求导出的数据中。

标识客户数据

可以将客户数据标识为 nuget.org 用户帐户名。

删除客户数据

请求从 nuget.org 删除用户数据：

- 用户必须登录到 [nuget.org](#)
- 用户必须通过 [nuget.org/account/delete](#) 提交请求，才能删除帐户

若用户为包的唯一所有者，建议其在请求删除帐户前找到新的所有者。如果不转移包所有权，则不会列出 NuGet 包，因此，它不会再显示在 Visual Studio 的搜索查询中或 nuget.org 网站上。删除帐户前，nuget.org 管理员需与用户协作，为其所拥有的包寻找新的所有者。

nuget.org 管理员将在请求提交后 30 天内完成帐户删除操作。

帐户删除后，所有用户数据都将从 nuget.org 系统中删除，并会执行以下操作：

- 从 nuget.org 中注销已删除的帐户
- 删除所有拥有的 API 密钥
- 释放所有保留命名空间
- 删除所有包所有权

不会删除拥有包。虽然搜索结果中未列出这些包，但依赖于它们的项目仍可通过包还原进行使用。

导出客户数据

登录到 nuget.org 后，用户可通过 [nuget.org/policies/Contact](#) 提交导出请求

用户可以通过 Azure Blob 在 48 小时内下载导出的数据。48 小时后，访问过期，用户必须根据需要提交新的导出请求。

导出的数据包括：

- 用户的支持请求
- 审核日志中保存的用户操作(发布包、创建帐户)
- IIS 日志中保存的任何用户信息

NuGet 的已知问题

2020/4/8 • [Edit Online](#)

这些是反复报告的 NuGet 最常见已知问题。如果在安装 NuGet 或管理包时遇到问题, 请查看这些已知问题及其解决方案。

NOTE

从 NuGet 4.0 开始, 各版本的发行说明中均包含已知问题。

在 nuget.exe v3.4.3 中, 向 VSTS 中的 NuGet 源进行身份验证的问题

问题:

当我们使用以下命令来存储凭据时, 我们最终会对个人访问令牌双重加密。

```
$PAT = "Your personal access token" $Feed = "Your url" .\nuget.exe sources add -Name Test -Source $Feed -UserName $UserName -Password $PAT
```

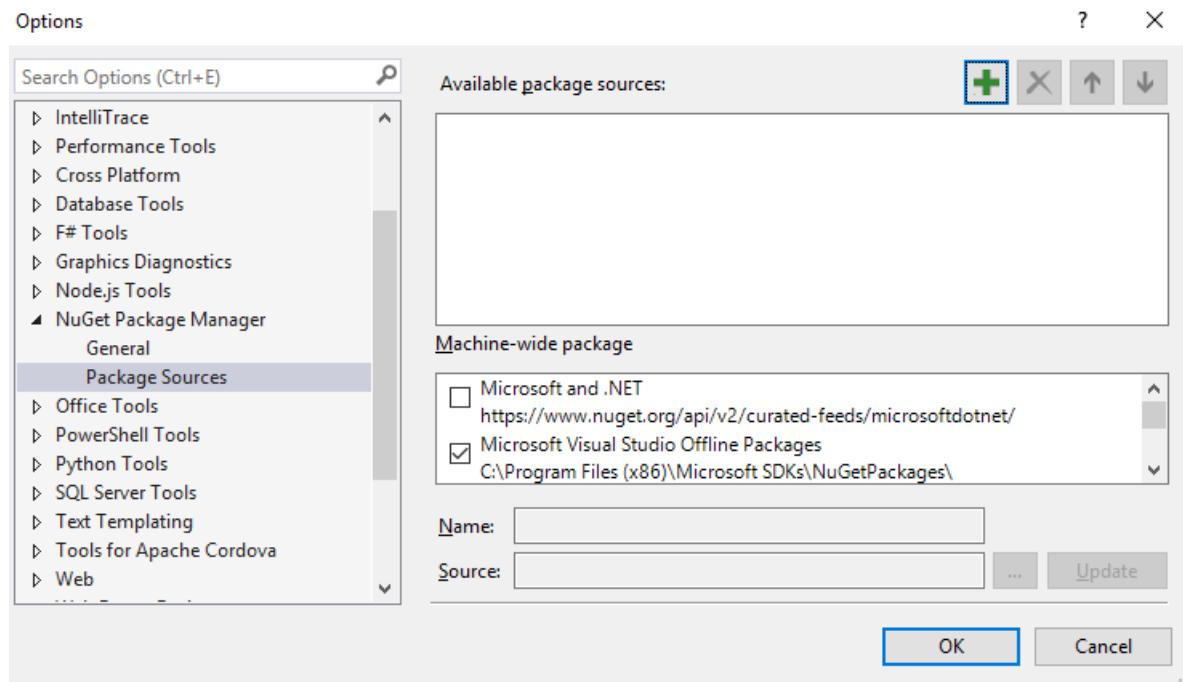
解决方法:

使用 [-StorePasswordInClearText](#) 选项以明文形式存储密码。

在 NuGet 3.4、3.4.1 中安装包时出现出错

问题:

在 NuGet 3.4 和 3.4.1 中, 使用 NuGet 加载项时, 没有源报告为可用, 且无法在配置窗口添加新源。结果与下图类似:



`NuGet.Config` (Windows) 或 `%AppData%\NuGet\` (Mac/Linux) 文件夹中的 `~/.nuget/` 文件意外被清空。若要解决此问题, 请执行以下操作:关闭 Visual Studio(适用于 Windows), 删除 `NuGet.Config` 文件, 然后再次尝试操作。
NuGet 生成了新的 `NuGet.Config` 文件, 你应该可以继续操作。

在 NuGet 2.7 中安装包时出现错误

问题：

在 NuGet 2.7 或更高版本中，尝试安装包含程序集引用的任何包时，可能会收到错误消息“输入字符串的格式不正确。”，如下所示：

```
install-package log4net
  Installing 'log4net 2.0.0'.
  Successfully installed 'log4net 2.0.0'.
  Adding 'log4net 2.0.0' to Tyson.OperatorUpload.
  Install failed. Rolling back...
install-package : Input string was not in a correct format.
At line:1 char:1
  install-package log4net
  ~~~~~
CategoryInfo : NotSpecified: (:) [Install-Package], FormatException
FullyQualifiedErrorId : NuGetCmdletUnhandledException,NuGet.PowerShell.Commands.InstallPackageCommand
```

这是由于未在系统上注册 `VSLangProj.dll` COM 组件的类型库引起的。这在以下情况下可能发生，例如：并行安装两个版本的 Visual Studio，然后卸载旧版本。此操作可能会无意中取消注册上述 COM 库。

解决方案：

从提升的提示符运行此命令，重新注册 的类型库 `VSLangProj.dll`

```
regsvr32 "C:\Program Files (x86)\Common Files\microsoft shared\MSEnv\VSLangproj.olb"
```

如果该命令失败，请检查文件是否存在该位置。

有关此错误的详细信息，请参阅此[工作项](#)。

在 VS 2012 中更新包之后生成失败

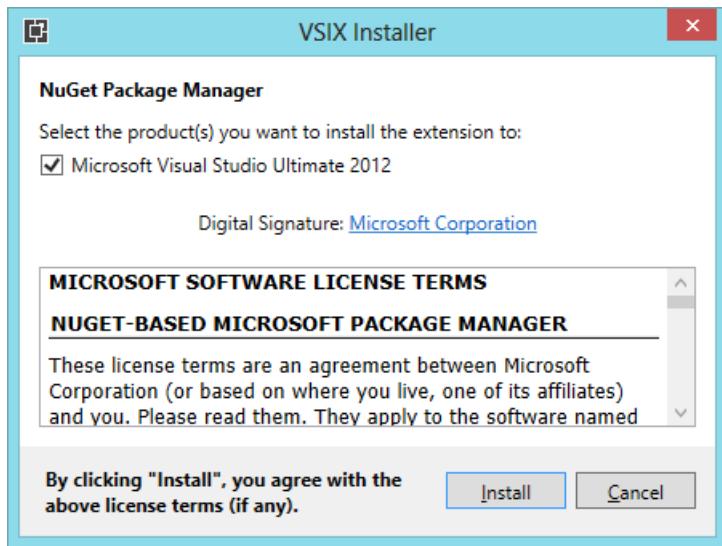
问题：使用 VS 2012 RTM。更新 NuGet 包时，出现此消息：“无法完全卸载一个或多个包。”，并提示你重启 Visual Studio。VS 重启后，收到奇怪的生成错误。

原因是旧包中的某些文件被后台 MSBuild 进程锁定。即使 VS 重启，后台 MSBuild 进程仍在使用旧包中的文件，导致生成失败。

解决方法是安装 VS 2012 Update，例如 VS 2012 Update 2。

从旧版升级到最新版 NuGet 时出现签名验证错误

由于安装了旧版本，尝试升级 NuGet 时，如果运行 VS 2010 SP1，可能遇到以下错误消息。



查看日志时，可能会看到提到 `SignatureMismatchException`。

若要防止此情况发生，可安装 [Visual Studio 2010 SP1 修补程序](#)。或者，解决方法是简单地卸载 NuGet（以管理员身份运行 Visual Studio 时），然后从 VS 扩展库安装。有关详细信息，请参阅<https://support.microsoft.com/kb/2581019>。

当安装了 Reflector Visual Studio 加载项时，包管理器控制台也会引发异常。

在运行包管理器控制台时，如果安装了 Reflector VS 加载项，则可能会遇到以下异常消息。

```
The following error occurred while loading the extended type data file:  
Microsoft.PowerShell.Core, C:\Windows\SysWOW64\WindowsPowerShell\v1.0\types.ps1xml(2950) :  
Error in type "System.Security.AccessControl.ObjectSecurity":  
Exception: Cannot convert the "Microsoft.PowerShell.Commands.SecurityDescriptorCommandsBase"  
value of type "System.String" to type "System.Type".  
System.Management.Automation.ActionPreferenceStopException:  
Command execution stopped because the preference variable "ErrorActionPreference" or common parameter  
is set to Stop: Unable to find type
```

或

```
System.Management.Automation.CmdletInvocationException: Could not load file or assembly 'Scripts\nuget.psm1' or
one of its dependencies. <br />The parameter is incorrect. (Exception from HRESULT: 0x80070057 (E_INVALIDARG))
---&gt; System.IO.FileLoadException: Could not load file or <br />assembly 'Scripts\nuget.psm1' or one of its
dependencies. The parameter is incorrect. (Exception from HRESULT: 0x80070057 (E_INVALIDARG)) <br />---&gt;
System.ArgumentException: Illegal characters in path.
   at System.IO.Path.CheckInvalidPathChars(String path)
   at System.IO.Path.Combine(String path1, String path2)
   at Microsoft.VisualStudio.Platform.VsAppDomainManager.<AssemblyPaths>d__1.MoveNext()
   at Microsoft.VisualStudio.Platform.VsAppDomainManager.InnerResolveHandler(String name)
   at Microsoft.VisualStudio.Platform.VsAppDomainManager.ResolveHandler(Object sender, ResolveEventArgs args)
   at System.AppDomain.OnAssemblyResolveEvent(RuntimeAssembly assembly, String assemblyFullName)
   --- End of inner exception stack trace ---
   at Microsoft.PowerShell.Commands.ModuleCmdletBase.LoadBinaryModule(Boolean trySnapInName, String moduleName,
String fileName, <br />Assembly assemblyToLoad, String moduleBase, SessionState ss, String prefix, Boolean
loadTypes, Boolean loadFormats, Boolean& found)
   at Microsoft.PowerShell.Commands.ModuleCmdletBase.LoadModuleNamedInManifest(String moduleName, String
moduleBase, <br />Boolean searchModulePath, <br />String prefix, SessionState ss, Boolean loadTypesFiles,
Boolean loadFormatFiles, Boolean& found)
   at Microsoft.PowerShell.Commands.ModuleCmdletBase.LoadModuleManifest(ExternalScriptInfo scriptInfo,
ManifestProcessingFlags <br />manifestProcessingFlags, Version version)
   at Microsoft.PowerShell.Commands.ModuleCmdletBase.LoadModule(String fileName, String moduleBase, String
prefix, SessionState ss, <br />Boolean& found)
   at Microsoft.PowerShell.Commands.ImportModuleCommand.ProcessRecord()
   at System.Management.Automation.Cmdlet.DoProcessRecord()
   at System.Management.Automation.CommandProcessor.ProcessRecord()
   --- End of inner exception stack trace ---
   at System.Management.Automation.Runspaces.PipelineBase.Invoke(IEnumerable input)
   at System.Management.Automation.Runspaces.Pipeline.Invoke()
   at NuGetConsole.Host.PowerShell.Implementation.PowerShellHost.Invoke(String command, Object input, Boolean
outputResults)
   at NuGetConsole.Host.PowerShell.Implementation.PowerShellHostExtensions.ImportModule(PowerShellHost host,
String modulePath)
   at NuGetConsole.Host.PowerShell.Implementation.PowerShellHost.LoadStartupScriptScripts()
   at NuGetConsole.Host.PowerShell.Implementation.PowerShellHost.Initialize()
   at NuGetConsole.Implementation.Console.Dispatcher.Start()
   at NuGetConsole.Implementation.PowerConsoleToolWindow.MoveFocus(FrameworkElement consolePane)
```

我们已联系了加载项的创建者，希望能够制定一个解决方案。

更新：我们已验证了最新版本的 Reflector (6.5 版) 不再在控制台中导致此异常。

打开包管理器控制台失败，出现 ObjectSecurity 异常

尝试打开包管理器控制台时，可能会看到以下错误：

```
The following error occurred while loading the extended type data file: Microsoft.PowerShell.Core,
C:\Windows\SysWOW64\WindowsPowerShell\v1.0\types.ps1xml(2977) : Error in type
"System.Security.AccessControl.ObjectSecurity": Exception: The getter method should be public, non void,
static, and have one parameter of type PSObject.
The following error occurred while loading the extended type data file: Microsoft.PowerShell.Core,
C:\Windows\SysWOW64\WindowsPowerShell\v1.0\types.ps1xml(2984) : Error in type
"System.Security.AccessControl.ObjectSecurity": Exception: The getter method should be public, non void,
static, and have one parameter of type PSObject.
The following error occurred while loading the extended type data file: Microsoft.PowerShell.Core,
C:\Windows\SysWOW64\WindowsPowerShell\v1.0\types.ps1xml(2991) : Error in type
"System.Security.AccessControl.ObjectSecurity": Exception: The getter method should be public, non void,
static, and have one parameter of type PSObject.
The following error occurred while loading the extended type data file: Microsoft.PowerShell.Core,
C:\Windows\SysWOW64\WindowsPowerShell\v1.0\types.ps1xml(2998) : Error in type
"System.Security.AccessControl.ObjectSecurity": Exception: The getter method should be public, non void,
static, and have one parameter of type PSObject.
The following error occurred while loading the extended type data file: Microsoft.PowerShell.Core,
C:\Windows\SysWOW64\WindowsPowerShell\v1.0\types.ps1xml(3005) : Error in type
"System.Security.AccessControl.ObjectSecurity": Exception: The getter method should be public, non void,
static, and have one parameter of type PSObject.
The term 'Get-ExecutionPolicy' is not recognized as the name of a cmdlet, function, script file, or operable
program. Check the spelling of the name, or if a path was included, verify that the path is correct and try
again.
```

如果看到此错误, 请按照 [StackOverflow 上讨论的解决方案](#) 来修复错误。

如果解决方案包含 InstallShield Limited Edition 项目, 则“添加包库引用”对话框会引发异常

我们发现, 如果解决方案包含一个或多个 InstallShield Limited Edition 项目, 打开时, “添加包库引用”对话框将引发异常。除了删除或卸载 InstallShield 项目, 目前尚无解决方法。

“卸载”按钮变灰? 需要管理权限才能安装/卸载 NuGet

如果尝试通过 Visual Studio Extension Manager 卸载 NuGet, 可能会注意到“卸载”按钮被禁用。需要管理员访问权限才能安装和卸载 NuGet。以管理员身份重启 Visual Studio, 卸载扩展。NuGet 不需要管理员访问权限即可使用。

在 Windows XP 中打开包管理器控制台时崩溃。为什么会这样?

NuGet 需要 Powershell 2.0 运行时。默认情况下, Windows XP 没有 Powershell 2.0。可从 <https://support.microsoft.com/kb/968929> 下载 Powershell 2.0 运行时。安装后, 重启 Visual Studio, 应该就能打开包管理器控制台。

如果包管理器控制台处于打开状态, Visual Studio 2010 SP1 Beta 在退出时崩溃。

如果已安装了 Visual Studio 2010 SP1 Beta 版, 可能会注意到, 当打开包管理器控制台并关闭 Visual Studio 时, 它会崩溃。这是 Visual Studio 的一个已知问题, 将在 SP1 RTM 版本中修复。临时解决方法是直接忽略崩溃或卸载 SP1 Beta 版(如果可以)。

元素“元数据”...发生无效的子元素异常

如果安装了使用 NuGet 预发布版本生成的包, 当对该项目运行 RTM 版本的 NuGet 时, 可能会遇到错误消息“命名空间'schemas.microsoft.com/packaging/2010/07/nuspec.xsd'中的元素‘元数据’具有无效的子元素”。将需要使用 RTM 版本的 NuGet 卸载并重新安装每个包。

尝试安装或卸载导致错误“文件已存在时，无法创建该文件。”

出于某种原因，Visual Studio 扩展呈现奇怪的状态，即已卸载了 VSIX 扩展，但遗留了一些文件。若要解决此问题，请执行以下操作：

1. 退出 Visual Studio
2. 打开以下文件夹(可能位于计算机上的其他驱动器上)
C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE\Extensions\Microsoft Corporation\NuGet Package Manager<version>\
3. 删除带有 .deleteme 扩展名的所有文件。
4. 重新打开 Visual Studio

执行这些步骤后，继续操作。

在极少数情况下，当代码分析处于开启状态时，编译会导致错误。

如果使用包管理器控制台安装 FluentNHibernate，然后在代码分析处于开启状态时编译项目，可能会收到以下错误。

```
Error 3 CA0058 : The referenced assembly  
'NHibernate, Version=3.0.0.2001, Culture=neutral, PublicKeyToken=aa95f207798dfdb4'  
could not be found. This assembly is required for analysis and was referenced by:  
C:\temp\Scratch\src\MyProject.UnitTests\bin\Debug\MyProject.UnitTests.dll.  
MyProject.UnitTests
```

默认情况下，FluentNHibernate 需要 NHibernate 3.0.0.2001。但是，NuGet 会故意在项目中安装 NHibernate 3.0.0.4000，并添加相应的绑定重定向，使其工作。如果没有开启代码分析，项目将正常编译。与编译器不同，代码分析工具不正确地遵循绑定重定向来使用 3.0.0.4000 而非 3.0.0.2001。可通过安装 NHibernate 3.0.0.2001 或者通过执行以下操作，指示代码分析工具与编译器行为相同，来解决这个问题：

1. 转到 %PROGRAMFILES%\Microsoft Visual Studio 10.0\Team Tools\Static Analysis Tools\FxCop
2. 打开 FxCopCmd.exe.config 并将 `AssemblyReferenceResolveMode` 从 `StrongName` 更改为 `StrongNameIgnoringVersion`。
3. 保存更改并重新生成项目。

Write-Error 命令在 install.ps1/uninstall.ps1/init.ps1 中不起作用

这是一个已知问题。请调用 `throw`，而非调用 `Write-Error`。

```
throw "My error message"
```

在 Windows 2003 上安装具有受限访问权限的 NuGet 可能会导致 Visual Studio 崩溃

尝试使用 Visual Studio Extension Manager 安装 NuGet 并且未以管理员身份运行时，将显示“运行方式”对话框，并且默认选中对话框中标记为“以受限访问运行此程序”的复选框。



选中该复选框后，单击“确定”会导致 Visual Studio 崩溃。确保先取消选中该选项，然后再安装 NuGet。

无法卸载适用于 Windows Phone 工具的 NuGet

Windows Phone 工具不支持 Visual Studio Extension Manager。若要卸载 NuGet，请运行以下命令。

```
vsixinstaller.exe /uninstall:NuPackToolsVsix.Microsoft.67e54e40-0ae3-42c5-a949-fddff5739e7a5
```

更改 NuGet 包 ID 的大写会中断包还原

正如在[此 GitHub 问题](#)上详细讨论的，可通过 NuGet 支持来完成 NuGet 包的大写更改，但对于在 global-packages 文件夹中现有不同包的用户来说，包还原期间会导致复杂情况。建议仅在有办法与包的现有用户交流生成时包还原可能发生的中断情况时，才请求更改事例。

报告问题

若要报告 NuGet 问题，请访问 <https://github.com/nuget/home/issues>。

NuGet 5.5 发行说明

2020/3/25 • [Edit Online](#)

NuGet 分发车辆:

NUGET	VISUAL STUDIO	.NET SDK
5.5.0	Visual Studio 2019 版本16.5	3.1.200 ¹

¹随 Visual Studio 2019 with .NET Core 工作负载一起安装

摘要 : 5.5 中的新增功能

- 改进了 Visual Studio 中 NuGet 包管理器 UI 的辅助功能和屏幕阅读器体验
 - 屏幕阅读器体验中的辅助功能问题，缺少已安装文本框等的 altText 和可访问的名称，#9059
 - 包中的屏幕阅读器体验中的辅助功能问题列表 - #9077
 - 屏幕阅读器体验中与 "浏览"、"安装" 和 "更新" 选项卡相关的辅助功能问题 - #9078
 - 讲述人不会公告 "空白"、"无 Dependencies" "、" "nuget"、"MIT" 链接标签 #9157
- 支持在 Visual Studio 包管理器 UI 中为本地源上承载的包提供自包含的图标 - #8189
- 使用 `RestoreUseStaticGraphEvaluation` 提高了无操作还原性能，从而可通过调用 MSBuild 静态图形 API 来加快评估 - 8791
- 通过跨平台身份验证插件改进 dotnet 的可靠性
 - dotnet restore 失败，TaskCanceledException - #7842
 - 插件："已取消任务"-由于此原因，ADO 身份验证存在问题。 - #8528
- 添加 `dotnet nuget <add|remove|update|disable|enable|list> source` 命令 - #4126
- 使用 dotnet nuget 推送 #8778 的 `--skip-duplicate` 的支持
- 支持 `packages.config` 与 msbuild/restore - #8506

此版本中已修复的问题

Bug

- 通过 V3 API 改编自我更新程序 - #4197
- 如果包依赖项版本设置为 "*", 则包依赖项版本错误 - #6697
- ErrorUnsafePackageEntry 错误消息不指向问题的根源 - #7505
- 锁定文件不采用 "*" 方案 - #8073
- 在 PackageReference 中使用 * 时，Nuget.exe 不会解析为包的最新版本 (MSBuild/Dotnet/VS restore do) - #8432
- 具有多个目标 WPF 项目的 dotnet 列表包 - #8463
- 改善 ConcurrencyUtilities (降低 CPU 使用率) - #8653
- 不应在预览还原中编写已卸载项目方案的 DG 规范 - #8793
- Visual Studio NuGet 包 (RestoreManagerPackage) 需要自动加载解决方案生成事件 - #8796

- .Vssettings 初始化中的死锁- #8842
- 如果项目位于解决方案文件夹中，则不会从 NuGet 包填充 VisualStudio 工具箱- #8868
- VS：由于争用条件，解决方案还原永久失败- #8881
- "正在安装" 选项卡上的 "正在加载 ..." 和 "搜索 ..." 更新选项卡- #8890
- 缓存过期后，VS PM UI 中缺少嵌入的图标- #9069
- FireAndForget PM UI 启动- #9112
- Restore: IncludeExcludeFiles (...) 实现不正确- #9167
- Restore: PackageSpec () 创建不相等的克隆 #9211
- 未选中错误列表 "如果生成完成但有错误，则显示错误列表" 的错误列表- #8190
- 静态图形还原不应传递空的 SolutionPath #9061
- 还原：计算每个项目4次的结束时间- #9042
- Restore: DependencyGraphSpec (...) 不需要 JObject- #9040
- 还原：在大型对象堆(LOH)上创建的大型字符串- #9031
- 新 mono 上的自定义 nuget.exe 可能因 MSBuild SDK 解析程序而中断- 8848
- dgspec "由其他进程使用" 时还原失败- 8692

Dcr

- _GetRestoreProjectStyle 中的逻辑应在任务 #8804
- 默认情况下，在 "已安装" 选项卡上添加弃用信息- #8541

此版本中已修复的所有问题的列表-5。5

NuGet 5.4 发行说明

2020/1/8 • [Edit Online](#)

NuGet 分发车辆:

NUGET	VISUAL STUDIO	.NET SDK
5.4.0	Visual Studio 2019 版本16.4	3.1.100 ¹

¹随 Visual Studio 2019 with .NET Core 工作负载一起安装

摘要 : 5.4 中的新增功能

- 更快的解决方案加载时间-在第一个解决方案加载过程中，运行 NuGet 代码的开销已通过部分 ngen 降低，以减少 JIT 成本[#6007](#)
- 新建帮助程序函数-给定包 id 和版本的列表，获取可能的顶级包。[- #8316](#)
- 新 [nuget/setup-nuget](#) 操作，用于在[GitHub 操作](#)上安装和配置 nuget.exe。[- #8818](#)

此版本中已修复的问题

Bug

- 插件：linux/Mac 上的日志记录时间准确性已关闭- [#8747](#)
- 释放插件有时会引发并使整个操作失败。- [#8732](#)
- 停止在 PMUI 中的允许和阻止版本列表中显示版本重复项- [#8679](#)
- 锁定文件未正确生成-框架排序不应影响 restore with lockedmode- [#8645](#)
- SDK 3.0.100 中设置的项目的 LockFile 验证失败- [#8639](#)
- 签名验证现在会正确拒绝带有时间戳的签名，该时间戳在同一 OID [#8629](#) 下具有2个值
- 更新许可证列表- [#8544](#)

Dcr

- 将诊断文件载入 IFeedbackDiagnosticFileProvider- [#8535](#)

此版本中已修复的所有问题的列表-5。4

NuGet 5.3 发行说明

2020/1/31 • [Edit Online](#)

NuGet 分发车辆:

NUGET	VISUAL STUDIO	.NET SDK
5.3.0	Visual Studio 2019 版本16.3	3.0.100 ¹
5.3.1	Visual Studio 2019 版本16.3.6	未来版本:3.0.101

¹随 Visual Studio 2019 with .NET Core 工作负载一起安装

摘要 : 5.3 中的新增功能

- 包图标可以嵌入到包中, 而无需使用外部 URL。- #352
- 提高了对包的 SHA 跟踪和强制执行的安全性- #7281
- 支持弃用过时/旧 NuGet 包[#2867](#) | 博客文章 | 文档

此版本中已修复的问题

Bug

- 使用 3.0.100-preview9 SDK 生成的 NuGet 包不能由 2.2 SDK 用户使用。。具体取决于时区[#8603](#)
- 引号 "路径中的字符导致" 路径中的非法字符 "`nuget restore`" 中的失败[#8168](#)
- VS:程序集是完全 ngen-ed, 而不是部分 ngen- [#8513](#)
- 减少内存使用量(取消订阅事件)- [#8471](#)
- "Error_UnableToFindProjectInfo" 消息的语法不正确- [#8441](#)
- NU1403 改进-验证所有包, 包括预期/实际 sha 值- [#8424](#)
- `NuGetPackageManager.PreviewUpdatePackagesAsync` - 中的多个枚举[#8401](#)
- 恢复 Pluginprocess.exe 中的 "公共 > 内部" 更改- [#8390](#)
- IVsPackageSourceProvider. GetSources (...) 的异常行为定义错误- [#8383](#)
- 使 PluginManager 构造函数再次公开- [#8379](#)
- 用于跟踪 PM UI 的刷新速率的指标- [#8369](#)
- 通过包管理器 UI 安装时减少 UI 刷新的次数- [#8358](#)
- 遥测: datetime 值使用特定于区域性的格式- [#8351](#)
- 减少包管理器 UI 的浏览选项卡中的 UI 刷新 #6570- [#8339](#)
- [测试失败]"无法解析配置文件" 将提示两次[#8320](#)
- 向说明客户修补程序的良好文档页引发 NU5037 错误(包缺少所需的 nuspec 文件)- [#8291](#)
- 更改项目的 RuntimeIdentifiers 时, 锁定模式还原失败- [#8260](#)

- 在 VS 懒惰#8156中进行设置读取
- `Nuget sources add` 中的回归导致 ":" 字符(十六进制值0x3A)不能包含在名称 "errors- #7948
- NuGet 插件凭据提供程序-隐藏进程窗口- #7511
- 强制 PackagePathResolver 是绝对路径#7349
- 减少安装和更新包管理器 UI 的选项卡中的 UI 刷新- #6570

DCR:

- 更新 Xamarin framework 以映射到 NetStandard 2.1- #8368
- 为安装/更新#8324启用包管理器 "预览窗口" 的内容复制
- 对 proj 文件启用还原- #8212
- 引入 `NUGET_NETFX_PLUGIN_PATHS` 和 `NUGET_NETCORE_PLUGIN_PATHS` 同时支持这两种方法的配置- #8151
- 通过版本属性为 PackageDownload 启用多个版本- #8074
- 将-SolutionDirectory 和-PackageDirectory 选项添加到 nuget.exe 包- #7163

此版本中已修复的所有问题的列表-5。3

摘要: 5.3.1 中的新增功能

- 插件:任务被取消-不允许取消影响插件实例化- #8648
- 无法在一个进程中安全地运行两次还原任务(使用凭据提供程序)- #8688

NuGet 5.2 发行说明

2019/7/26 • [Edit Online](#)

NuGet 分发车辆:

NUGET	VISUAL STUDIO	.NET SDK
5.2.0	Visual Studio 2019 版本16.2	2.1.80 X ¹ , 2.2.40 X ²

¹随 Visual Studio 2019 with .NET Core 工作负载一起安装

²可用作 Visual Studio 2019 with .NET Core 工作负载的可选安装

摘要:5.2 中的新增功能

- 修复了由于 Linux & Mac 上的路径问题而导致不定期 NuGet 操作失败的严重 bug [#7341](#)
- 提高了使用 Visual Studio 中的 NuGet 包管理器 UI 浏览包时的 UI 响应能力, 尤其是对于慢速源而言, [#8039](#)
- 用于锁定文件的大量可靠性修补程序 ([#8187](#)、[#8160](#)、[#8114](#)、[#7840](#)) 和身份验证插件 ([#8300](#)、[#8271](#)、[#8269](#)、[#8210](#)、[#8198](#)、[#7845](#))

此版本中已修复的问题

Bug

- 性能程序包管理器控制台:UI 延迟更新 "默认项目" combobox 选定的值- [#8235](#)
- 性能PM UI 中的性能改进- [#8039](#)
- 性能在 PMC 中读取默认项目时的 UI 延迟- [#6824](#)
- Perf: [vsfeedback] 针对本地包源的 "NuGet 更新" 选项卡冻结- [#6470](#)
- 插件如果插件无法启动或提前终止, NuGet 将等待完全握手超时[#8300](#)
- 插件: 提高诊断插件启动故障- [#8271](#)
- 插件内置插件的 nuget.exe 发现问题- [#8269](#)
- 插件: 缓存文件不是只读的[#8210](#)
- 插件"任务已取消。" 在还原过程中具有身份验证插件的错误- [#8198](#)
- 无法在 linux 平台上间歇地发现插件缓存- [#7845](#)
- LockFile: 使用 ATF 时, 由于目标 framework 相等性检查错误, 它具有 false NU1004 [#8187](#)
- LockFile: 如果锁定文件为空或格式不正确, 则不遵循 "--锁定模式" 还原标志[#8160](#)
- LockFile: 不包含包中的自定义程序集名称的小写项目锁定文件- [#8114](#)
- LockFile: 在锁定文件中使项目引用小写[#7840](#)
- 还原: 安装篡改后的签名包将导致多次失败的安装尝试 (使用重复输出)- [#8175](#)
- VS: 解决方案用户选项在 NuGet 更新后未能反序列化- [#8166](#)
- UnitTest 项目中的 dotnet 包返回错误- [#8154](#)

- 为 VS 安装程序创建 NuGet 包组-修复某些 VSIX 安装问题- #8033
- GeneratePackageOnBuild 不应设置 NoBuild。 - #7801
- 当 nuspec 文件包含显式程序集引用元素#7638时, 新选项 "-SymbolPackageFormat snupkg" 将生成错误。
- NuGet (498, 5): 错误:找不到路径 "/tmp/NuGetScratch- #7341

DCR:

- 添加一个 msbuild 属性, 该属性指示支持 PackageDownload- #8106
- FrameworkReference 通过 FrameworkReference 禁用依赖项流- #7988
- 用于在包外提供 runtime #7351的机制

[此版本中已修复的所有问题的列表-5.2 RTM](#)

NuGet 5.1 发行说明

2019/7/12 • [Edit Online](#)

NuGet 分发车辆:

NUGET	VISUAL STUDIO	.NET SDK
5.1.0	Visual Studio 2019 版本 16.1	2.1.70 ¹ , 2.2.30 ²

¹随 Visual Studio 2019 一起安装.NET Core 工作负载

²作为可选安装随.NET Core 工作负载的 Visual Studio 2019

摘要:什么是 5.1 中的新增功能

- 支持要跳过包推送, 如果已存在, 更好地集成 CI/CD 工作流- #1630年
- Visual Studio 现在提供了方便指向包的 nuget.org 库页- #5299
- 新的.NET Core 3.0 是资产的支持, 如[目标包](#)和[运行时包](#)
 - NuGet 包和还原的支持 FrameworkReferences 启用目标和运行时包引用- #7342
 - 支持"仅下载"包方案具有 PackageDownload- #7339
 - 排除运行时和目标从搜索结果和还原的包关系图使用 PackageType- #7337

此版本中已修复的问题

Bug

- 插件: 插件期间的异常详细信息丢失#8057
- PackageReference 范围排他下限不适如果下限中存在, 其中一个源。 - #8054
- Improve IsPackableFalseError message - #8021
- 包锁定文件的项目图发生更改时, 请重新生成锁文件- #8019
- ProjectSystem bug:Nuget 包获取自动删除- #8017
- 添加用于返回 FrameworkReference 目标类似于 CollectPackageDownloads 和 CollectPackageReferences- #8005
- HTTP 缓存:RepositoryResources 资源未缓存的版本控制的方式- #7997
- 日志记录: 异常调用堆栈未报告详细- #7955
- 更改为使用 HTTPS-的所有 NuGet Docs Url #7950
- 提高 NU3024 警告消息- #7933
- 锁定文件未更新时删除了 -packagereference #7930
- 验证 licenseurl 和许可证 nuspec-中的元素时改进错误 case 处理#7915
- PM UI-右键单击选项卡头, 并单击"打开文件位置"错误-生成#7913
- 插件: 日志时插件进程退出的#7907

- 插件: 在日志记录的日期时间值的高冲突率 #7899
- 上与 LicenseExpression-任何 nuspec 失败 Manifest.ReadFrom #7894
- RestoreLockedMode:意外的 NU1004 ProjectReference 引用具有自定义程序集名称的项目时 #7889
- 更详细的错误消息时插件启动失败, 异常- #7857
- 执行操作时 NoOp 还原, 避免 *。在 obj 目录-dgspec.json 写 #7854
- GeneratePathProperty = true 无法生成在大小写不匹配的属性 #7843
- 设置: 在包源路径中的非法字符可以导致 VS 崩溃- #7820
- 如果删除锁定文件, 还原不会生成 NoOp-上的锁定文件 #7807
- 许可证 URL 和许可证会导致读取错误与元数据- #7547
- 未经处理的异常中 V2FeedParser- #7523
- nuget.exe 返回退出代码为零的参数无效- #7178
- 更新错误和警告文档以反映相关的签名方案- #6498
- 资产文件应该使用相对路径来更轻松地-启用移动项目 #4582

DCRs

- 插件: 启用诊断日志记录- #7859
- 请 Tizen 6 映射到 NetStandard 2.1- #7773

在此版本的 5.1 RTM 中已解决所有问题的列表

NuGet 5.0 发行说明

2019/11/5 • [Edit Online](#)

NuGet 分发车辆:

NUGET	VISUAL STUDIO	.NET SDK
5.0.0	Visual Studio 2019 版本16.0	2.1.602 ¹ 、2.2.202 ²
5.0.2	Visual Studio 2019 版本16.0。4	2.1.60 x ¹ 、2.2.20 x ²

¹随 Visual Studio 2019 with .NET Core 工作负载一起安装

²可用作 Visual Studio 2019 with .NET Core 工作负载的可选安装

摘要 : 5.0 中的新增功能

- 支持在 Visual Studio 2019 中还原筛选的解决方案 - [#5820](#)
- `BuildTransitive` 文件夹使包能够以可传递的目标/属性到主机项目 - [#6091](#)
- 更好地支持 NuGet IVs API 中的 PackageReference 方案 - [#7005](#)、[#7493](#)
- `nuget.exe pack project.json` 已弃用 - [#7928](#)
- 第1代凭据提供程序插件已被[gen 2](#)取代, 即将弃用 - [#7819](#)

此版本中已修复的问题

Bug

- 执行 NoOp 还原时, 请避免在 obj 目录中写入 dgspec - [#7854](#)
- 在 ~/nuget 中创建的文件的权限过于打开 - [#7673](#)
- `dotnet list package --outdated` 不适用于需要身份验证 [#7605](#) 的源
- VisualStudio。IVsPackageInstaller 对没有包引用的项目调用始终使用 package, 即使默认设置为 PackageReference - [#7005](#)
- PMC: 更新包重新安装在 delisted 包上失败 ("找不到包")。 - [#7268](#)
- 在我们的存储库和 VSIX 中添加第三方通知 - [#7409](#)
- 如果未指定版本, 则 VisualStudio 应安装最新版本 [#7493](#)
- 交互式支持 dotnet nuget 推送 - [#7519](#)
- 当还原时, 不应引发 NU1603 警告。 - [#7529](#)
- 在还原过程中 NuGet 不应打印项目路径 - [#7647](#)
- 交互式支持 dotnet 删除包 - [#7727](#)
- 添加 back Nuget.exe with TypeForwardedTo attrs - [#7768](#)
- plugins_cache 需要更短的路径才能正常工作 - [#7770](#)
- 如果用户没有请求特定的 msbuild 版本, 则首选 msbuild 发现的路径 - [#7786](#)

- `nuget.exe /?` 应列出正确的 msbuild 版本- #7794
- NuGet (498, 5): 错误: 找不到路径 "/tmp/NuGetScratch" 的一部分#7793
- 还原不必要地枚举计算机缓存中引用包的所有版本的内容- #7639
- 在安装 VS 2019 Preview 后, MSBuild 自动检测始终会选择 16.0 #7621
- 解决方案中的 dotnet 列表包输出框架#7607的重复条目
- 在旧项目和包文件夹中调用 IVsPackageInstaller 时, 异常 "空路径名称不合法"。 - #5936
- msbuild/t: 还原最少详细级别应该更小- #4695
- VS 16.0 的 NuGet UI 由于颜色问题而无法读取选项卡- #7735
- Nuget.exe & NuGet。客户端许可证 .txt 说明- #7629
- 还原不必要地枚举全局包文件夹, 尝试确定类型#7596
- 锁定文件强制执行的错误应显示在错误列表窗口- #7429
- 修复 NuGet。配置问题- #7326
- 适应 MSBuild 更新其安装位置- #7325
- Nuget.exe 应为开发依赖项- #7249
- 添加包扩展点以包括调试符号- #7234
- `dotnet pack` 应保留创建的 nupkg 中的依赖项版本范围(即使使用了浮动版本)- #7232
- 当用户级配置还具有源#7209时, 对通过身份验证的源的 `dotnet restore` 失败
- 包不应限制内容文件的 BuildActions 集- #7155
- 使用需要 AssetTargetFallback 成功的 ProjectReference 会发出警告。 - #7137
- 由于调用 CPS (CommonProjectSystem) 时线程问题导致死锁- #7103
- `dotnet add package` 不会对本地配置中指定的源使用来自全局配置的凭据- #6935
- 在异步代码路径上调用 MEF 的线程问题- #6771
- 签名: 错误报告了两次, 没有调用堆栈- #6455
- 安装带有不受信任签名证书的签名包应显示错误- #6318
- 当2个项目共享 obj 目录时, NuGet 还原 Noop 错误- #6114
- 在 Linux 上, 不能将 PAT `dotnet restore` 与来自已验证源的包结合使用- #5651
- dotnet restore 因禁用了计算机范围的源而失败- #5410

Dcr

- 以后删除 "dotnet pack" 时警告- #7928
- 为 Gen1 credential 插件添加弃用警告- #7819
- 签名: 启用的存储库需要将每个包的客户端验证为签名#7759的存储库
- 通过 NuGet 限制每个源的 http 请求数- #4538
- NuGet 应面向 Net472 (以帮助清理 VSIX 的16.0 版本)- #7143

- PMC: 删除 OpenPackagePage 命令 - #7384
- 将 NetCoreApp 3.0 映射到 NetStandard 2.1 - #7762
- 将 netstandard 2.0 支持添加到 NuGet。* 包 - #6516
- 允许包作者定义生成资产可传递行为 - #6091
- 支持 VS 2019 解决方案筛选器功能。还支持不在解决方案中的项目或已卸载的项目。需要首先还原完整解决方案(通过 CLI 或 VS) #5820
- NuGet 5.0 程序集需要 .NET 4.7.2 (通过 TFM 更改) - #7510
- 从 NuGet NuGetLicenseData。打包应为公共类型。从 spdx 更新许可证元数据引入。 - #7471
- 删除过时的设置 Api- #7294
- 具有1个 cpu #6742的系统上的解决方法还原超时
- NuGet 首选 NTLM 身份验证, 即使在 NuGet 中有凭据-添加配置选项来筛选凭据的身份验证类型- #5286
- 启用用于 PackageReference 的 EmbedInteropTypes (匹配的包 .Config 功能) - #2365

此版本中已修复的所有问题的列表-5.0 RTM

摘要: 5.0.2 中的新增功能

- 安全(当通过 dotnet 或 cscript.exe 运行时)-应创建具有正确权限的 obj 文件夹#7908
- mono/MacOS 上的 nuget.exe 还原失败, 并出现自定义 nuget .config 和 `PackageSignatureValidity: False` #8011

已知问题

FallbackFolders 中由 .NET Core SDK 安装的包是自定义安装的, 无法通过签名验证。 - #7414

问题

使用 dotnet.exe 2.x 还原多重定位 netcoreapp 1.x 和 netcoreapp 2.x 的项目时, 回退文件夹被视为文件源。也就是说, 在还原时, NuGet 将从回退文件夹中选取包, 并尝试将它安装到全局包文件夹, 再执行失败的常规签名验证。

解决方法

通过将 `RestoreAdditionalProjectSources` 设置为 "无", 禁用回退文件夹的使用:

```
<RestoreAdditionalProjectSources></RestoreAdditionalProjectSources>
```

请谨慎使用这一点, 因为现在将从 NuGet.org 下载从回退文件夹还原的包。

NuGet 4.9 发行说明

2020/4/8 • [Edit Online](#)

NuGet 分发车辆:

NUGET	VISUAL STUDIO	.NET SDK
4.9.0	Visual Studio 2017 版本 15.9.0	2.1.500、2.2.100
4.9.1	n/a	n/a
4.9.2	Visual Studio 2017 版本 15.9.4	2.1.502, 2.2.101
4.9.3	Visual Studio 2017 版本 15.9.6	2.1.504, 2.2.104

摘要:4.9.0 版中的新增功能

- 签名:允许 ClientPolicies 要求必须使用 NuGet.Config 中列出的一组受信任作者和存储库 - [#6961, 博客文章](#)
- 创建".snupkg"文件以将符号包含在包中, 即将 push 增强为了解 nuget 协议, 以接受符号服务器的 snupkg 文件 - [#6878, 博客文章](#)
- NuGet 凭据插件 V2 - [#6642](#)
- 独立式 NuGet 包 - 许可证 - [#4628, 公告](#)
- 支持选择对 PackageReference 启用"GeneratePathProperty"元数据, 以将每个包的 MSBuild 属性生成到"Foo.Bar\1.0" 目录中 - [#6949](#)
- 改进了客户成功执行 NuGet 操作的体验 - [#7108](#)
- 使用锁定文件启用可重复的包还原 - [#5602, 公告, 博客文章](#)

此版本中已修复的问题

- (通过 WarnAsErrors)提升为 PackageExtraction 抛出的错误的警告绝不得留下已提取包 - [#7445](#)
- 错误签名的包最终不得位于全局包文件夹中 - [#7423](#)
- 绑定重定向生成不得跳过外观程序集 - [#7393](#)
- VersionRange Equals 不比较浮动范围 - [#7324](#)
- 还原:使用新 .NET Core 2.1 HTTP 堆栈进行性能回归 - [#7314](#)
- 包更新不得修改 PackageReference 的 PrivateAssets - [#7285](#)
- 签名:如果包内的条目太多 (> 65534), 签名应该会失败 - [#7248](#)
- "dotnet nuget push"代码路径应支持新凭据提供程序 - [#7233](#)
- 支持执行具有固定区域性的插件(就像在 Docker 中一样) - [#7223](#)
- nuget sources add 不得从 NuGet.config 中删除凭据 - [#7200](#)
- devDependency PackageReference 安装应默认采用 excludeassets=compile - [#7084](#)

- 将迁移程序选项修复为，在项目不兼容时对所有项目显示并显示错误 - #6958
- “dotnet add package”应提交对资产文件执行的还原 - #6928
- 签名：改进了与错误消息相关的签名 - #6906
- [测试失败][zh-TW]字符串“包管理器控制台”在包管理器控制台中未本地化 - #6381
- 与“找不到项目信息”相关的错误消息应在 VS 中更具体一点 - #5350
- 错误使用 nuget 包的 nuspec 版本标记时，错误消息毫无益处 - #2714
- DCR - 签名：支持 NuGet 协议：RepositorySignatures/4.9.0 资源 - #7421
- DCR - 现在 .nupkg.metadata 文件在包提取期间创建 - 包含“内容哈希”- #7283
- DCR - 在 Mono 上执行时，跳过插件的验证码验证 - #7222

版本 4.9.0 中所有已修复问题的列表

摘要:4.9.1 版中的新增功能

- 现在支持通过新命令 trusted-signers 读取对 nuget.config 执行的写入操作 - #7480

此版本中已修复的问题

- 修复了许可证链接生成 - #7515
- 用于验证签名的错误代码回归 - #7492
- NuGet.Build.Tasks.Pack 包没有许可证信息 - #7379

版本 4.9.1 中所有已修复问题的列表

摘要:4.9.2 版中的新增功能

此版本中已修复的问题

- 如果源名称包含空格，VS/dotnet.exe/nuget.exe/msbuild.exe 还原不使用凭据 - #7517
- LicenseAcceptanceWindow 和 LicenseFileDialog 辅助功能问题 - #7452
- 修复 DateTimeConverter 中 DateTime.Parse 中的 FormatException - #7539

版本 4.9.2 中所有已修复问题的列表

摘要:4.9.3 中的新增功能

此版本中已修复的问题

“使用锁定文件启用可重复的包还原”问题

- 锁定模式不工作，因为未对以前缓存的包正确计算哈希 - #7682
- 还原解析为与 `packages.lock.json` 文件中定义的版本不同的版本 - #7667
- 涉及 ProjectReference 时，“--locked-mode / RestoreLockedMode”导致虚假的还原失败 - #7646
- MSBuild SDK 解析程序尝试验证在使用 packages.lock.json 时未能还原的 SDK 包的 SHA - #7599

“使用可配置的信任策略锁定依赖关系”问题

- dotnet.exe 不应在签名包不受支持时评估受信任的签名者 - #7574
- 配置文件中 trustedSigners 的顺序影响信任评估 - #7572

- 无法实现 ISettings [由设置 API 重构导致以支持信任策略功能] - #7614

“提升了调试体验”问题

- 无法发布 .NET Core 全局工具的符号包 - #7632

“独立 NuGet 包 - 许可证”问题

- 使用嵌入的许可证文件时，生成符号 .snupkg 包出错 - #7591

版本 4.9.3 中所有已修复问题的列表

摘要:4.9.4 版中的新增功能

- 安全修复:~/.nuget 中针对所创建文件的权限过于开放 #7673 CVE-2019-0757

已知问题

dotnet nuget push --interactive 在 Mac 上抛出错误。- #7519

问题

--interactive 参数无法由 dotnet cli 转发，因此导致错误 error: Missing value for option 'interactive' 出现

解决方法

将 interactive 选项与其他任何 dotnet 命令一起运行(如 dotnet restore --interactive)，并进行身份验证。凭据提供程序可能会缓存身份验证。然后，运行 dotnet nuget push。

FallbackFolders 中由 .NET Core SDK 安装的包是自定义安装的，无法通过签名验证。- #7414

问题

使用 dotnet.exe 2.x 还原多重定位 netcoreapp 1.x 和 netcoreapp 2.x 的项目时，回退文件夹被视为文件源。也就是说，在还原时，NuGet 将从回退文件夹中选取包，并尝试将它安装到全局包文件夹，再执行失败的常规签名验证。

解决方法

通过将 RestoreAdditionalProjectSources 设置为无，禁用回退文件夹。请谨慎使用

<RestoreAdditionalProjectSources/>，因为它会导致从 NuGet.org 下载许多包，否则将从回退文件夹中还原这些包。

NuGet 4.8 发行说明

2020/4/8 • [Edit Online](#)

Visual Studio 2017 15.8 RTW 附带 NuGet 4.8 功能。

此外提供了相同功能的命令行版本：

- NuGet.exe 4.8 - [nuget.org/downloads](#)
- DotNet.exe - [.NET Core SDK 2.1.400](#)

摘要:4.8.0 版中的新增功能

- NuGet.exe 现支持 Windows 10 上的 longfilenames - [#6937](#)
- 身份验证插件现适用于 MsBuild、DotNet.exe、NuGet.exe 和 Visual Studio(包括跨平台使用)。MsBuild 的 DotNet.exe 中不支持第一代身份验证插件。注意:VS 2017 15.9 预览内部版本包含 VSTS 身份验证插件。[#6486](#)
- MsBuild 的 SDK 解析程序现作为 NuGet 的一部分生成，并通过 VS 的 NuGet 工具安装。这将避免版本无法同步的问题。[#6799](#)
- PackageReference 现支持 DevelopmentDependency 元数据 - [#4125](#)

摘要:4.8.2 版中的新增功能

- 安全修复:~/.nuget 中针对所创建文件的权限过于开放 [#7673 CVE-2019-0757](#)

已知问题

在 CI 计算机或在脱机环境中安装签名包所需的时间要比平常长

问题

如果计算机的 Internet 访问受限(如 CI/CD 方案中的生成计算机)，安装/还原签名的 nuget 包将导致警告([NU3028](#))，因为吊销服务器不可访问。这是预期情况。但是，在某些情况下，这可能会产生意外的结果，如安装/还原包花费的时间比平常长。

解决方法

更新到 Visual Studio 15.8.4 和 NuGet.exe 4.8.1，我们在其中引入了一个环境变量来切换吊销检查模式。将 `NUGET_CERT_REVOCATION_MODE` 环境变量设置为 `offline` 将迫使 NuGet 仅针对缓存证书吊销列表检查证书吊销状态，NuGet 不会尝试访问吊销服务器。如果吊销检查模式设置为 `offline`，警告将降级为信息。

WARNING

不建议在正常情况下将吊销检查模式切换到脱机。执行此操作将导致 NuGet 跳过联机吊销检查，并只对可能过期的缓存证书吊销列表执行脱机吊销检查。这意味着将继续安装/还原签名证书可能已经吊销的包，否则吊销检查将失败，并且不会安装。

`Migrate packages.config to PackageReference...` 选项在右键单击上下文菜单中不可用

问题

首次打开项目时，在执行 NuGet 操作之前，NuGet 可能不会进行初始化。这将导致迁移选项不会显示在 `packages.config` 或 `References` 的右键单击上下文菜单中。

解决方法

执行以下 NuGet 操作之一：

- 打开包管理器 UI - 右键单击 `References` 并选择 `Manage NuGet Packages...`

- 打开“包管理器控制台” - 从 `Tools > NuGet Package Manager` 中选择 `Package Manager Console`
- 运行 NuGet 还原 - 在“解决方案资源管理器”中，右键单击该解决方案节点，然后选择 `Restore NuGet Packages`
- 生成还会触发 NuGet 还原的项目

现在应能够看到迁移选项。请注意，此选项不受支持且不会对 ASP.NET 和 C++ 项目类型显示。注意：此问题已在 VS 2017 15.9 预览版 3 中修复

此版本中已修复的问题

Bug

签名

- 签名：在脱机环境中安装签名包 [#7008](#) -- 已在 4.8.1 中修复
- 签名：不正确的 URL 检查 - [#7174](#)
- 签名：当包是存储库副署时，检查 `RepositorySignatureVerifier` 中包的完整性 - [#6926](#)
- “包完整性检查失败。”消息中应具有包 ID（和错误代码） - [#6944](#)
- 存储库签名包验证允许由不同证书对包进行签名 - [#6884](#)
- NuGet - 签名 - 时间戳 URL 不能是 `https://?` - [#6871](#)
- 不要在 NuSpec 封装方案中使用 NULL 引用，还可以改进选项 - [#6866](#)
- 如果将时间戳添加到副署，更新签名者信息时内存无效 - [#6840](#)
- 签名：删除 CTL 异常 - [#6794](#)
- 签名：`contentUrl` 必须是 HTTPS - [#6777](#)
- 签名：未使用 `SignedPackageVerifierSettings.VSClientDefaultPolicy` - [#6601](#)

打包

- 使用 `dotnet.exe` 打包 `nuspec` 时不需要还原和生成 - [#6866](#)
- 允许在 `NuspecProperties` 中使用空的替换令牌 - [#6722](#)
- 当指定 `NuspecProperties` 时，`PackTask` 将引发 `NullReferenceException` - [#4649](#)

可访问性

- [可访问性] 包按钮下的字符串“预发行版”在 PM UI 中被包描述覆盖 - [#4504](#)
- [可访问性] 当选择 PM UI 中的“Microsoft Visual Studio 脱机包”时，包源下拉列表和设置按钮被截断 - [#4502](#)

Powershell 管理控制台 (PMC)

- `Update-Package` 忽略 `PackageReference` 版本范围 - [#6775](#)
- `Update-Package -reinstall` 解决方案级别的问题 - [#3127](#)
- `Update-Package [packagename] -reinstall` 重新安装所有包，而不仅仅是指定包 - [#737](#)
- 可以从包管理器控制台中更新到未列出的 NuGet 包 - [#4553](#)

杂项

- 若要修复 `NuGet update self`，`NuGet.Commandline.nupkg` 不应为 semver2.0 - [#7116](#)
- 改进 NU1107 安装失败的体验 - [#7107](#)
- `GetAuthenticationCredentialRequest` 的序列化不正确 - [#6983](#)
- 关闭 UI 线程初始化时 NuGet Visual Studio AsyncPackage 加载失败 - [#6976](#)
- 还原操作报告误导性错误，指示不需要 `project.json` - [#6959](#)
- 包管理器 UI：预览更改，“确定”按钮不可自动通过键盘使用 - [#6893](#)
- 对于使用 p2p 引用的项目，会忽略 `RestoreSources` - [#6776](#)
- 使用 .NET Framework 模板创建单元测试项目将使用 `packages.config` 打开较旧的项目模型 - [#6736](#)
- 允许项目引用重写包依赖项 - [#6536](#)
- 在 MSBuild 任务中公开 `NoDefaultExcludes` - [#6450](#)
- 调整窗口大小时，可以隐藏“清除所有 NuGet 缓存”的状态消息 - [#5938](#)

此版本中所有已修复问题的列表

NuGet 4.7 发行说明

2020/4/8 • [Edit Online](#)

Visual Studio 2017 15.7 RTW 附带 NuGet 4.7.0。

摘要:4.7.0 版中的新增功能

- 我们已增强包签名，以便启用[存储库签名包](#)
- 在 Visual Studio 版本 15.7 中，我们已引入功能改为[迁移使用 packages.config 格式的现有项目以使用 PackageReference](#)。

摘要:4.7.2 版中的新增功能

- 安全修复:~/.nuget 中针对所创建文件的权限过于开放 [#7673 CVE-2019-0757](#)

摘要:4.7.3 版中的新增功能

- 安全修复:NUPKGs 中的文件可以具有高于 NUPKG 目录的相对路径 [#7906](#)

已知问题

`Migrate packages.config to PackageReference...` 选项在右键单击上下文菜单中不可用

问题

首次打开项目时，在执行 NuGet 操作之前，NuGet 可能不会进行初始化。这将导致[迁移选项不会显示在 packages.config 或 References 的右键单击上下文菜单中](#)。

解决方法

执行以下 NuGet 操作之一：

- 打开包管理器 UI - 右键单击 `References` 并选择 `Manage NuGet Packages...`
- 打开“包管理器控制台” - 从 `Tools > NuGet Package Manager` 中选择 `Package Manager Console`
- 运行 NuGet 还原 - 在“解决方案资源管理器”中，右键单击该解决方案节点，然后选择 `Restore NuGet Packages`
- 生成还会触发 NuGet 还原的项目

现在应能够看到迁移选项。请注意，此选项[不受支持且不会对 ASP.NET 和 C++ 项目类型显示](#)。

使用 .NET Framework 和 NuGet 的 .NET Standard 2.0 的问题

.NET Standard 及其工具旨在使面向 .NET Framework 4.6.1 的项目可使用面向 .NET Standard 2.0 或更早版本的 NuGet 包和项目。[本文档](#)总结了该方案相关问题、解决问题的计划以及可使用当前工具状态部署的变通方法。

此版本中已修复的主要问题

Bug

- NuGet 在 .Net Core 项目系统中遇到死锁(新回归)。 - [#6733](#)
- 包:如果 TfmSpecificPackageFile 与组合路径一起使用，则 PackagePath 构造不正确 - [#6726](#)
- 包:除非显式设置 ispackable，否则 Web API 项目无法创建包。 - [#6156](#)
- VS UI 和 PMC 需要 30 分钟才能看到新包(nuget.exe 可以立即看到) - [#6657](#)
- 签名:SignatureUtility.GetCertificateChain(...) 不会检查所有链状态 - [#6565](#)
- 签名:改进 DER GeneralizedTime 处理 - [#6564](#)

- 签名: 安装经篡改的包时, VS 不显示 NU3002 错误 - [#6337](#)
- lockFile.GetLibrary 区分大小写 - [#6500](#)
- 安装/更新还原代码和还原代码路径不一致 - [#3471](#)
- 解决方案 PackageManager 版本组合框可以通过键盘选择分隔符 - [#2606](#)
- 无法加载源的服务索引 `https://www.myget.org/F/<id>` ---> System.Net.Http.HttpRequestException: 响应状态代码未指示成功:403(已禁止) - [#2530](#)

DCR

- 发出 X-NuGet-Session-Id 标头以跨请求关联[功能建议] - [#5330](#)
- 公开方法来等待运行通过 IV API 在 Visual Studio 中运行的还原操作。 - [#6029](#)
- NuGet.exe -NoServiceEndpoint 将避免追加服务 URL 后缀 - [#6586](#)
- 将提交哈希添加到信息版本 - [#6492](#)
- 签名: 启用存储库签名/副署删除 - [#6646](#)
- 签名: 去除存储库签名/副署的 API - [#6589](#)
- VS 中的登录源信息 - [#6527](#)
- 仅在 TFM 和 RID 上筛选 /tools, 以便可将设置 XML 置于 /tools 文件夹中 - [#6197](#)
- 当包命令排除开头的文件时发出警告。 - [#3308](#)

[此版本中所有已修复问题的列表](#)

NuGet 4.6 发行说明

2020/4/8 • [Edit Online](#)

Visual Studio 2017 15.6 RTW 附带 NuGet 4.6.0。

摘要:4.6.0 版中的新增功能

- 添加了对[给包签名](#)的支持。
- Visual Studio 2017 和 nuget.exe 现将在安装包、为[已签名包](#)还原包之前，验证包的完整性。
- 改进了连续还原的性能。

摘要:4.6.3 版中的新增功能

- 安全修复:~/.nuget 中针对所创建文件的权限过于开放 #7673 CVE-2019-0757

摘要:4.6.4 版中的新增功能

- 安全修复:NUPKGs 中的文件可以具有高于 NUPKG 目录的相对路径 #7906

已知问题

使用 .NET Framework 和 NuGet 的 .NET Standard 2.0 的问题

.NET Standard 及其工具旨在使面向 .NET Framework 4.6.1 的项目可使用面向 .NET Standard 2.0 或更早版本的 NuGet 包和项目。[本文档](#)总结了该方案相关问题、解决问题的计划以及可使用当前工具状态部署的变通方法。

此版本中已修复的主要问题

性能修复

- 没有更改时，无法写入资产文件 - #6491
- 当子项目的 TFM 与父项目的 TFM 不匹配时，还原会导致额外的 MSBuild 计算 - #6311
- 通过优化依赖项关系图创建规范来提高 NoOp 还原性能 - #6252

Bug

- 推送到本地文件夹导致 nupkg 锁定 - #6325
- NuGet 插件实现:多个问题 - #6149
- UIHang - 从 VSSolutionManager 的 MEF 初始化中删除查询服务调用 - #6110
- 错误报告已取消的包下载任务发生异常 - #6096
- NuGet.exe 在程序集名称中用“%2B”替换“+”- #5956
- Fn+F1 无法转到 PM UI 和控制台的正确帮助页面 - #5912
- VS NuGet 在特定情况下会将绝对路径写入项目文件 - #5888
- 修复 4.3 回归 - 无法通过转换在内容文件中替换占位符 \$product\$ 和 \$AssemblyGuid\$ - #5880
- dotnet 还原出现多个源故障 - #5817
- 包应重新评估项目版本以允许 git 版本控制 - #4790
- 改善在安装不兼容包时出现的难以理解的错误 - #4555
- TemplateWizard 需要将包安装为 PackageReferences 的选项 - #4549
- 从开发人员命令提示符以外运行 MSBuild.exe 时，包传递的属性文件被忽略 - #4530

- 修复在引用不适用于项目的.NET Standard 库时出现的不当错误消息 - [#4423](#)
- dotnet add package 无法为面向可移植配置文件的包提供指导 - [#4349](#)
- dotnet 包 - ProjectReference 缺少版本后缀 - [#4337](#)
- 有关 .NET Core 模板的生成错误和 VS 崩溃 - [#3973](#)
- 无法加载源 https:* 的服务索引 - [#3681](#)
- nuget.exe list -allversions 无法正常工作 - [#3441](#)
- 误导性的依赖项解析错误信息 - [#2984](#)
- nuget.exe 还原不会为 .msbuildproj 生成 .props 和 .targets 文件(在 v3.3.0-3.4.4 升级中回归) - [#2921](#)
- 更新打开了 XAML 文件的 NuGet 包时出现 UI 延迟 - [#2878](#)
- 来自 IIS 的 WebSite 项目失败, 路径中包含非法字符 - [#2798](#)
- Nuget 添加在 CentOS 上挂起 - [#2708](#)
- 使用 packagesavemode -nupkg 还原 json.net 失败 - [#2706](#)
- 包管理器筛选器在 vs 输出窗口中不可用于还原命令 - [#2704](#)

[此版本中所有已修复问题的列表](#)

NuGet 4.5 发行说明

2020/4/8 • [Edit Online](#)

Visual Studio 2017 15.5 RTW 附带有 NuGet 4.5 RTM。

摘要:4.5.0 版中的新增功能

摘要:4.5.2 版中的新增功能

- 安全修复:~/.nuget 中针对所创建文件的权限过于开放 [#7673 CVE-2019-0757](#)

摘要:4.5.3 版中的新增功能

- 安全修复:NUPKGs 中的文件可以具有高于 NUPKG 目录的相对路径 [#7906](#)

已知问题

使用 .NET Framework 和 NuGet 的 .NET Standard 2.0 的问题

.NET Standard 及其工具旨在使面向 .NET Framework 4.6.1 的项目可使用面向 .NET Standard 2.0 或更早版本的 NuGet 包和项目。本文档总结了该方案相关问题、解决问题的计划以及可使用当前工具状态部署的变通方法。

无法使用 NuGet 包管理器查看、添加或更新 DotNetCLITools

问题

NuGet 包管理器不显示，且不允许添加/更新 DotNetCLITools。[NuGet#4256](#)

解决方法

必须在项目文件中手动编辑 DotNetCLIToolReferences。

对目标框架版本重定目标可能会导致 Intellisense 不完整

问题

在 Visual Studio 中对目标框架版本重定目标可能会导致 Intellisense 不完整。将 PackageReferences 用作包管理器格式时可能出现这种情况。[NuGet#4216](#)

解决方法

手动进行还原。

.NET Core 项目中的某个包内附具有无效签名的程序集，该包可触发还原操作无限循环

问题

有时，如果所用包内附具有无效签名的程序集或者包版本设置有“DateTime”贴标，这会导致包自动还原操作无限循环 [dotnet/project-system#1457](#)。

解决方法

目前没有解决方法。

NuGet 4.5 RTM 时间框架中已修复的问题

有关 NuGet 4.4 RTM 中已修复问题的信息，请参阅[NuGet 4.4 RTM 发行说明](#)

特征

- 禁用符号包的自动推送 - [#6113](#)

Bug

- [回归]在 15.5p1 中:Portable0.0 被跳过 - [#6105](#)
- 还原后包中的资产丢失 - [#5995](#)
- 插件凭据提供程序不支持包含空格的 URI - [#5982](#)
- 如果包无法还原, 应在输出中打印错误, 即使最低详细级别为“ON”- [#5658](#)
- dotnet
 - 解决方案级别的 dotnetcore restore 不遵循 ProjectReference, 且 ReferenceOutputAssembly 为 false, 导致随机生成失败 - [#5490](#)
- PMC 中的自动完成无法与对象方法正确协作 - [#4800](#)
- 使用 Visual Studio 2015 工具集时 nuget.exe 还原失败 - [#4713](#)
- 在 vs2017 中实例化 perf - pmc 成本高昂 - [#4205](#)
- 慢速连接时获取依赖项信息速度缓慢 - [#4089](#)
- 如果多个包共享通用的依赖项, 带 -RemoveDependencies 的 uninstall-package 将失败 - [#4026](#)
- 完成 NuGet.Core.nupkg 以便发布 - [#3581](#)
- 将 -IncludeProjectReferences 用于 csproj + project.json 时, NuGet 包会解析目录名称中的依赖项 ID - [#3566](#)
- “NuGet.ProxyCache”的类型初始值设定项引发异常 - [#3144](#)
- 使用 kudu 时的 nuget 还原性能问题 - [#3087](#)
- 搜索超出注册 blob 时, UI 客户端无法显示任何错误或警告 - [#2149](#)
- Get-Packages -Updates 生成不正确的查询 - [#2135](#)

4.5 RTM 中已修复 GitHub 问题的链接

[问题列表](#)

NuGet 4.4 发行说明

2020/4/13 • [Edit Online](#)

Visual Studio 2017 15.4 RTW 附带 NuGet 4.4 RTM。

摘要:4.4.0 版中的新增功能

摘要:4.4.2 版中的新增功能

- 安全修复:~/.nuget 中针对所创建文件的权限过于开放 #7673 CVE-2019-0757

摘要:4.4.3 版中的新增功能

- 安全修复:NUPKGs 中的文件可以具有高于 NUPKG 目录的相对路径 #7906

已知问题

使用 .NET Framework 和 NuGet 的 .NET Standard 2.0 的问题

.NET Standard 及其工具旨在使面向 .NET Framework 4.6.1 的项目可使用面向 .NET Standard 2.0 或更早版本的 NuGet 包和项目。本文档总结了该方案相关问题、解决问题的计划以及可使用当前工具状态部署的变通方法。

使用包管理器控制台时，“Enter”键可能不起作用

问题

有时无法在包管理器控制台中使用 Enter 键。如果看到此内容，请在修补程序上签出进程，并提供有关重现步骤的其他任何有用信息。[NuGet#4204](#) [NuGet#4570](#)

解决方法

打开该解决方案之前，重启 Visual Studio 并打开 PMC。或者，请尝试删除 `project.lock.json`，然后再次还原。

无法使用 NuGet 包管理器查看、添加或更新 DotNetCLITools

问题

NuGet 包管理器不显示，且不允许添加/更新 DotNetCLITools。[NuGet#4256](#)

解决方法

必须在项目文件中手动编辑 DotNetCLIToolReferences。

对目标框架版本重定目标可能会导致 Intellisense 不完整

问题

在 Visual Studio 中对目标框架版本重定目标可能会导致 Intellisense 不完整。将 PackageReferences 用作包管理器格式时可能出现这种情况。[NuGet#4216](#)

解决方法

手动进行还原。

.NET Core 项目中的某个包内附具有无效签名的程序集，该包可触发还原操作无限循环

问题

有时，如果所用包内附具有无效签名的程序集或者包版本设置有“DateTime”贴标，这会导致包自动还原操作无限循环 (`dotnet/project-system#1457`)。

解决方法

目前没有解决方法。

NuGet 4.4 RTM 时间框架中已修复的问题

[NuGet 4.3 RTM 发行说明](#)- 列出了 NuGet 4.3 RTM 已修复的所有问题

特征

- 支持 PMC 和 NuGet PM UI 方案中的轻量型解决方案加载 - [#5180](#)
- msbuild 包目标应具有公共挂钩，以用于在其本身之前运行用户目标 - [#5143](#)
- 功能: 将 dependencyVersion 开关添加到 nuget install - [#1806](#)
- uap10.0.TODO.0 应映射到适用于 NuGet 的 .NET Standard 2.0 - [#5684](#)
- 支持使用 msbuild/t:restore 的 Visual Studio 生成工具 SKU - [#5562](#)
- 还原期间，如果需要适用于 .NET Standard 2.0 的 .NET 4.6.1 支持但尚未安装，则会出现错误 - [#5325](#)
- 包 ID 前缀预留客户端 UI - [#5572](#)
- 提供本地化 nuget 组件以支持 dotnet.exe 本地化 - [#4336](#)

Bug

- 项目路径大小写不同可能导致还原丢失 PackageReferences - [#5855](#)
- 将带警告编号的错误代码移动到错误范围 - [#5824](#)
- 未知 .NET Standard 版本与目标框架兼容时，出现误导性错误 - [#5818](#)
- 测试文件的许可证混乱 - [#5776](#)
- EndToEnd 测试模板中缺少许可证标头 - [#5774](#)
- packages.config restore 显示错误为 NU1000 - [#5743](#)
- nuget.exe install 应在 mono 上具有 DisableParallelProcessing - [#5741](#)
- nuget.exe install 不正确地禁用缓存 - [#5737](#)
- 禁用 Restore 时，运行 packages.config 的还原命令的 VS 会显示不正确消息 - [#5718](#)
- VS; 禁用 Restore 时，运行还原命令显示令人混乱的消息 - [#5659](#)
- 缺少版本元数据时，GetRestoreDotnetCliToolsTask 失败 - [#5716](#)
- dotnet
 - dotnetcore 添加包可从 csproj 清除空行 - [#5697](#)
- NuGet.Config 中凭据设置的源名称要区分大小写 - [#5695](#)
- 启用 GeneratePackageOnBuild 删除了包的整个历史记录 - [#5676](#)
- Restore 不会还原 mono.cecil 或 semver 包，但所有其他包都会还原。 - [#5649](#)
- 错误和警告 - 源不可用时发生严重错误。 - [#5644](#)
- [DesignConsistency] NuGet 安装状态文本目前在深色主题方面看起来不正确。 - [#5642](#)
- 解决方案中的更新包会对所有项目进行更新/安装 - [#5508](#)
- dotnet
 - 根据 TargetFramework 与 TargetFrameworks，dotnetcore pack 的行为有所不同 - [#5281](#)
- “工具”文件夹中包含的 DLL 引发警告 - [#5020](#)

- NuGet.ContentModel 将过多内存用于字符串操作 - #4714
- 对于 OSX, RuntimeEnvironmentHelper.IsLinux 返回 true - #4648
- “dotnet 包”将 nuspec 置于 obj 下, 而不是 obj\Debug 下 - #4644
- Nuget 极其缓慢的包升级 - #4534
- CPS 与尚未开启 LSL(轻量型解决方案还原)的较大型解决方案中的 Restore 不同步 - #4307
- SemVer 2.0 - 具有提供版本的 nuget pack 忽略元数据 (3.5.0-rtm-1938) - #3643
- 带版本号和 ExcludeVersion 标志的 Nuget.exe (3.+) 安装包不能将包更新到较新版本 - #2405
- 顶级包违反约束时, Project.json restore 应发出警告 - #2358
- -ConfigFile 不在安装命令上设置自定义配置 - #1646
- nuget.exe install 不遵循“-DisableParallelProcessing”开关 - #1556
- DotNet.exe 或 msbuild.exe 仍在使用已禁用的源 - #5704
- 修复在 LSL 方案中挂起 - #5685

DCR

- nuget.exe install TargetFramework 支持 - #5736
- 添加不同的 msbuild 任务 UserAgent 字符串 (netcore 与桌面 msbuild) - #5709
- PackagePathResolver.GetPackageDirectoryName 应为虚拟 - #5700
- [DesignConsistency] 添加 NuGet 包时, 消息混乱 - #5641
- [警告和错误] NoWarn 不通过 P2P 引用间接流动 - #5501
- 轻量级解决方案加载:PM UI、PMC 和 IV 的共同核心 - #5057
- 轻量级解决方案加载:支持 - PMC - #5053
- 添加对 Visual Studio 触发的还原前 MSBuild 目标的 support - #4781
- 将公共目标添加到可使用 BeforeTargets 引用的 NuGet.targets - #4634
- 包目标不能正确使用生成操作创建 contentFile - #4166
- RestoreOperationLogger.Do 会阻止线程池线程 - #5663

Docs

- 用于安装命令 DependencyVersion 和 Framework 标志的文档 - #5858
- 文档中有关 NuGet 警告和错误的更新 - #5857

4.4 RTM 中已修复 GitHub 问题的链接

[问题列表 1](#)

[问题列表 2](#)

[问题列表 3](#)

NuGet 4.3 发行说明

2020/4/13 • [Edit Online](#)

[Visual Studio 2017 15.3 RTW](#) 附带 NuGet 4.3 RTM, NuGet 4.3 RTM 添加了对新方案(例如 .NET Standard 2.0/.NET Core 2.0)的支持、包含多种质量修复和提升性体能。此版本还提供多方面的提升, 例如对语义化版本控制 2.0.0 的支持、NuGet 警告和错误的 MSBuild 集成等等。

摘要:4.3.0 版中的新增功能

摘要:4.3.1 版中的新增功能

- 安全修复: ~/nuget 中针对所创建文件的权限过于开放 [#7673 CVE-2019-0757](#)
- 安全修复: NUPKGs 中的文件可以具有高于 NUPKG 目录的相对路径 [#7906](#)

已知问题

在某些情况下, NuGet 还原操作可能会将禁用的包源视作已启用进行处理

问题

以下还原命令行方法将被禁用的包源视为已启用。[NuGet#5704](#)

- `msbuild /t:restore`
- `dotnet restore` (不管是使用 VS 随附的 dotnet.exe, 还是使用 NetCore SDK 2.0.0 随附的文件)

解决方法

- 使用 Visual Studio(2017 15.3 或更高版本)或 NuGet.exe(v4.3.0 或更高版本)
- 删除禁用的源, 并继续使用 msbuild 或 dotnet.exe。
- 对于解决方案, 可以在 NuGet.config 中使用“Clear”, 再定义此解决方案所必需的源。

使用包管理器控制台时, “Enter”键可能不起作用

问题

有时无法在包管理器控制台中使用 Enter 键。如果看到此内容, 请在修补程序上签出进程, 并提供有关重现步骤的其他任何有用信息。[NuGet#4204](#) [NuGet#4570](#)

解决方法

打开该解决方案之前, 重启 Visual Studio 并打开 PMC。或者, 请尝试删除 `project.lock.json`, 然后再次还原。

无法使用 NuGet 包管理器查看、添加或更新 DotNetCLITools

问题

NuGet 包管理器不显示, 且不允许添加/更新 DotNetCLITools。[NuGet#4256](#)

解决方法

必须在项目文件中手动编辑 DotNetCLIToolReferences。

对目标框架版本重定目标可能会导致 Intellisense 不完整

问题

在 Visual Studio 中对目标框架版本重定目标可能会导致 Intellisense 不完整。将 PackageReferences 用作包管理器格式时可能出现这种情况。[NuGet#4216](#)

解决方法

手动进行还原。

NuGet 4.3 RTM 时间框架中已修复的问题

[NuGet 4.0 RTM 发行说明](#) - 列出所有 NuGet 4.0 RTM 修复的问题

特征

- 提升 NuGet 还原性能 - 为命令行还原和 VS 实现更小的 NoOp - [#5080](#)
- NET Core 2.0:VS/Dotnet CLI 应开始使用现有的 NuGet 功能:FallBack 文件夹 - [#4939](#)
- NET Core 2.0:使用户可以忽略特定还原警告(或提升为错误) - [#4898](#)
- NET Core 2.0:CLI 本地化程序集 - [#4896](#)
- NET Core 2.0:将所有警告/错误注册到资产文件(包括 PackageTargetFallback) - [#4895](#)
- 启用 TFM 支持:NetStandard2.0, Tizen - [#4892](#)
- 减少 NuGet.Core 和 NuGet.Client 项目(和 DLL)的数量 - [#2446](#)
- 添加将 NuGet 警告标记为错误的功能 - [#2395](#)

Bug

- msbuild /t:pack 失败, "PackTask"任务不支持"DevelopmentDependency"参数 - [#5584](#)
- 如果未在 PackagePath 末尾添加 Windows 目录分隔符, 则内容文件的目录结构会合并 - [#4795](#)
- netcore 项目不支持设置为 developmentDependency - [#4694](#)
- 同步加载的 RestoreManagerPackage 锁定 UI 线程和死锁 VS - [#4679](#)
- dotnet
 - dotnetcore Restore(和 msbuild /t:restore)跳过有显式解决方案项目依赖项的项目 [#4578](#)
- 如果解决方案有引用大小写不同的同一项目的 projectreference, 则还原可能不起作用。这还会影响大小写相同的相对路径 - [#4574](#)
- NuGet 包还原的可执行文件不能再使用 .NET Core 2.0 执行 - [#4424](#)
- NuGet.exe 在分析解决方案文件时吞并异常的详细信息 - [#4411](#)
- 如果 ContentTargetFolders 在 Windows 上包含以"/"结尾的路径, 则包会将内容文件放在错误的位置 - [#4407](#)
- 不能还原面向 netcoreapp1.1 的工具包的 DotNetCliToolReference - [#4396](#)
- Nuget 更新 CLI 将旧包版本条件留在项目文件中 (C++) - [#2449](#)

DCR

- 从CPS nomination 读取 DotnetCliToolTargetFramework - [#5397](#)
- TPMinV check 应适用于 pj 样式 UWP - [#4763](#)
- 提升 AutoReferenced 包的 UI 说明 - [#4471](#)
- NuGet restore 从运行时部分选择编译资产。 - [#4207](#)
- 将依赖项诊断放在锁定文件中 - [#1599](#)

4.3 RTM 中已修复 GitHub 问题的链接

[问题列表](#)

NuGet 4.0 RTM 发行说明

2020/4/13 • [Edit Online](#)

Visual Studio 2017 采用的 NuGet 4.0 添加了 .NET Core 支持，拥有大量优质修补程序并可提高性能。此版本还具有一些改进，如支持 PackageReference、作为 MSBuild 目标的 NuGet 命令、后台包还原等等。

已知问题

当有多个项目引用解决方案中的另一项目时，NuGet 还原可能会失败

问题

如果解决方案中对同一项目的项目引用的大小写或路径不同，则 NuGet 还原可能会不起作用。[NuGet#4574](#)

解决方法

将所有项目引用的大小写和相对路径修复为相同。

使用包管理器控制台时，“Enter”键可能不起作用

问题

有时无法在包管理器控制台中使用 Enter 键。如果看到此内容，请在修补程序上签出进程，并提供有关重现步骤的其他任何有用信息。[NuGet#4204](#) [NuGet#4570](#)

解决方法

打开该解决方案之前，重启 Visual Studio 并打开 PMC。或者，请尝试删除 `project.lock.json`，然后再次还原。

在 .NET Core 项目中，使用包含具有无效签名的程序集的包时，可能会出现无限还原循环

问题

有时，使用包含具有无效签名的程序集的包或包版本设置有 'DateTime' 贴标时，会导致包自动还原无限循环运行。

[NuGet#4542](#)

解决方法

目前没有解决方法。

无法使用 NuGet 包管理器查看、添加或更新 DotNetCLITools

问题

NuGet 包管理器不显示，且不允许添加/更新 DotNetCLITools。[NuGet#4256](#)

解决方法

必须在项目文件中手动编辑 DotNetCLIToolReferences。

如果对项目设置 PackageId 属性，则 NuGet 包还原会失败

问题

对于 .NET Core 项目，Visual Studio 中的 NuGet 还原不遵从项目的 PackageId 属性。[NuGet#4586](#)

解决方法

使用命令行运行还原。

项目不包含 'obj' 文件夹时，包还原可能失败

问题

'obj' 文件夹被删除后 Visual Studio 无法还原 PackageReferences。[NuGet#4528](#)

解决方法

手动创建 'obj' 文件夹后应可正常进行还原。

在控制台中使用 Update-Package 手动更新包可能会失败

问题

在控制台中手动使用 Update-Package 对刚刚转换的 PackageReferences 项目仅适用一次。NuGet#4431

解决方法

目前没有解决方法。

对目标框架版本重定目标可能会导致 Intellisense 不完整

问题

在 Visual Studio 中对目标框架版本重定目标可能会导致 Intellisense 不完整。将 PackageReferences 用作包管理器格式时可能出现这种情况。NuGet#4216

解决方法

手动进行还原。

面向 .NET461 的项目引用面向 .NETStandard 的另一项目时 msbuild /t:restore 出现失败

问题

面向 .NET461 的基于 PackageReference 的项目引用面向 .NETStandard 的另一基于 PackageReference 的项目时 msbuild /t:restore 出现失败。NuGet#4532

解决方法

目前没有解决方法。

NuGet 4.0 RTM 时间框架中已修复的问题

[NuGet 4.0 RC 发行说明](#) - 列出了所有 NuGet 4.0 RC 中已修复的问题

特征

- 对 NuGet.Core.sln 中的字符串进行本地化 - [#2041](#)
- Nuget 强制加载 LSL 模式下的 Web 应用程序项目 - [#4258](#)
- 对于“已安装 SDK”的包，用于阻止 UI 中的版本更改的 AutoReferenced PackageReference 支持 - [#4044](#)
- 为任何项目依赖项 (PackageRef) 正确传达 PackageSpec.Version - [#3902](#)
- 支持从命令行删除引用到 `.csproj` 中 - [#4101](#)
- 支持对 PackageReference 项目 (常规和 xplat) 和轻型解决方案加载的还原 - [#4003](#)
- 支持将引用从命令行添加到 `.csproj` 中 - [#3751](#)
- 对于 `packages.config` 或 `project.json`，支持轻型解决方案加载的 NuGet 还原 - [#3711](#)
- 由 nuget 生成的目标文件中的 contentFiles 支持 - [#3683](#)
- 使用 MSBuild 在 Mac 上建立用于 nuget.exe 验证的 Mono CI - [#3646](#)
- 从 v2 NuGet.Core 依赖项中移出 NuGet - [#3645](#)

Bug

- Visual Studio 中的 NuGet 还原不遵从项目的 PackagId 属性 - [#4586](#)
- 在 vsix 包中添加包时，NuGet ProjectSystemCache 出现错误 - [#4545](#)
- 如果在具有多个 TFM 的项目中使用了 IncludeSource，包会引发异常 - [#4536](#)
- 使用解决方案范围的包管理的更新时，VS 2017 RC3 出现故障 - [#4474](#)
- 无法卸载新安装的包 - [#4435](#)
- 迁移到 PackageRef 时，混合解决方案出现异常还原行为 - [#4433](#)
- 如果在启动 NuGet 操作 (安装、更新或还原) 后立即进行生成操作，可能导致 VS 挂起 - [#4420](#)

- UI 挂起 - 死锁初始化 NuGet.SolutionRestoreManager.RestoreManagerPackage #4371
- add package 命令应将版本添加为属性(而不是元素) - #4325
- dotnet
 - dotnetcore Restore foo.sln - SLN 中的配置导致还原图中出现重复(但配置不同)项目时, 此操作失败 - #4316
- 仅包含内容的包 - #3668
- 默认情况下, 选择退出包格式选择器选项 - #4468
- 性能: CreateUAP_CSharp_VS.01.1.Create 项目使 Duration_TotalElapsedTime 回退了 3,153.570 毫秒 (149.1%)。基线 26129.02 - #4452
- 性能: ManagedLangs_CS_DDRIT.0300.Rebuild 解决方案使 Duration_TotalElapsedTime 回退了 1.5 秒。基线 26105 - #4441
- 多个 TFM 项目中出现提名失败 - #4419
- 性能: WebForms_DDRIT.1200.Close 解决方案使 VM_ImagesInMemory_Total_devenv 回退了 3.000 计数 (0.5%)。基线 26123.04 - #4408
- vsfeedback - 面向 netcoreapp1.1 时出现包警告 - #4397
- 尝试将 NuGet 包添加到空 ASP.NET Core Web 应用程序中时, 出现 PathTooLongException - #4391
- 包的运行过于频繁 -- dotnet
 - 目标依赖项关系图中存在涉及目标“包”的循环依赖时, dotnetcore 包失败 - #4381
- 包的运行过于频繁 - 生成 NuGet 包不包括所有配置 - #4380
- 使用 packageref 在 C++ 项目中添加 nuget 时, 出现 NullReferenceException - #4378
- 辅助功能: 讲述人不读出用于选择要将包安装到的项目的复选框 - #4366
- NuGet VS17 偶尔无法连接到 VSO/VSTS 源 - VS Bug 365798 - #4365
- 如果 PackagePath 将路径指定为“contentFiles”, contentFiles 会输出到错误位置 - #4348
- 包目标将 VersionSuffix 追加到 PackageVersion 属性中 - #4324
- 指定包路径操作不适用于 dotnet 包 - #4321
- 在还原期间, NuGet 输出大量有关重复导入的警告 - #4304
- 在深色主题下, 选择“NuGet 包管理器格式”对话框的显示不清晰 - #4300
- 生成还原时, VS 出现故障 - #4298
- 如果在 targetframeworks 中对 TFM 进行添加、保存和生成操作, Visual Studio 出现死锁。10% 的时间 - #4295
- nuget 包不输出有关成功打包项目的成功消息 - #4294
- 由于找不到 System.IO.Compression 4.1, PackTask 失败 - #4290
- 包的运行过于频繁 - 由于文件访问冲突, PackTask 频繁失败 - #4289
- 在后台还原期间, NuGet 打开输出窗口 - #4274
- 将 ServiceProvider 作为危险编码模式删除(这可能导致挂起)- #4268
- 性能/UIHang - 改善 DownloadTimeoutStream 的读取 - #4266

- 若在完成 NuGet 还原之前尝试关闭项目, Visual Studio 出现死锁 - #4257
- 有关 PackTask 和打包 `.nuspec` 的问题 - #4250
- [vsfeedback] 无法在新项目上解析 nuget 包(需要重启 Visual Studio)- #4217
- [vsfeedback] 显示可用包版本的“版本”下拉列表难以与所选的 nuget 包同步... - #4198
- 与 CPS 交互时, Nuget.Client 应使用 CPS JoinableTaskFactory 以防止死锁 - #4185
- NuGet 3.5.0 不从包中解压缩 `.targets` - #4171
 - dotnet
 - dotnetcore 包不支持 `.csproj` 中的标题 - #4150
- 安装包导致 VS2017 RC 中出现错误对话框 - #4127
- 针对 .net core 项目的更新包操作似乎不起作用, 因为 UI 没有从提名中获取 CPS 更新。 - #4035
- 改进未解析的引用警告 - #3955
- dotnet
 - dotnetcore 包 - ProjectReference 丢失版本信息 - #3953
- 创建 UWP 应用、创建项目和重新生成的总运行时间回归 - #3873
- 尽管还原期间出现错误, 仍显示成功还原消息。 - #3799
- 将 NugetCommandLine 3.4.4 重新发布到 Nuget.org - # 2931
- 在迁移期间, 项目从 `project.json` 更改为 `.csproj` --- 还原失败 - #4297
- 在新创建的 xunit 测试项目上, 还原失败 - #4296
- 核心项目可在打开状态下挂起和锁定 UI - #4269
- 修复生成任务的目标文件 - #4267
- 生成卸载引用的项目的解决方案后, 错误列表中出现错误 - #4208
- MSB4057:项目中不存在目标“_GenerateRestoreGraphProjectEntry”。 - #4194
- vsfeedback:选择所有项目时, 解决方案的 nuget 管理器 UI 出现故障 - #4191
- 存在尾随斜杠时, nuget.exe msbuildpath 失败 - #4180
- vsfeedback:NuGet 还原提供 LinqToTwitter 项目的多个项目引用警告 - #4156
- `.csproj` 的包不包括 minClientVersion 属性 - #4135
- 运送的 NuGet.Build.Tasks.Pack.dll 延迟了 VS2017 (d15rel 26014.00) 中的签名 - #4122
- VSFeedback:使用 CMake 3.7.1 生成的 VS 2015 项目还原失败 - #4114
- VSFeedback:还原错误可能掩盖生成可提供的更完整的错误消息 - #4113
- [VSFeedback] 还原网站项目的 NuGet 包时出现错误:值不能为空。 - #4092
- 迁移在 NuGet.PackageManagement.VisualStudio.SolutionRestoreWorker 中引发“对象引用异常”- #4067
- dotnet
 - dotnetcore 包应使用生成包时所用的版本来打包工具 - #4063
- 还原需要花费数秒时, 新的后台还原将毫秒写入状态栏 - #4036

- 由于拼写错误，未能解析所有项目引用 - #4018
- 在包引用方案中启用 PCM 工作流 - #4016
- 在包管理器 UI 中找不到已安装的包 - #4015
- dotnet
 - PackagePath 为空时，dotnetcore 包失败 - #3993
- 多用户方案中的还原任务失败 - #3897
- 使用 NuGet 包任务打包时，无法更改内容类型 - #3895
- 对于 MsBuild /t:pack, ContentFiles 的默认副本错误 - #3894
- 安装包还原将还原包的消息记录了两次 - #3785
- 删除 Guardrails - “运行时”部分的还原应仅应用于当前项目 - #3768
- 包任务在“content/”和“contentFiles/”中均放置了内容文件 - #3718
- dotnet
 - dotnetcore pack3 执行额外标记拆分操作 - #3701
- dotnet
 - dotnetcore 包：打包具有包引用的项目导致重复导入警告 - #3665
- VS 中并非始终显示还原日志记录 - #3633
- nuget 局部变量可帮助文本仍提到的包缓存 - #3592
- Restore3 将 PackageReferences 与 TargetFrameworks 结合。 - #3504
- 在 VS“15”Preview 4 dev. 命令提示符中，Nuget 选取意外的 MSBuild 版本 - #3408
- 还原失败时，写出目标/属性文件 - #3399
- 在 VS 15 命令提示符中运行时，还原期间的 NuGet 所遵循的兼容性填充码与 MSBuild 不同 - #3387
- 为 VS15 重新启用 PackFromProjectWithDevelopmentDependencySet - #3272
- NuGet 的 Blend 出现问题 - #4043
- 将 4.0.0.2067 集成到 CLI 和 SDK 存储库以与 RC2 共同传送 - #4029
- 创建新核心控制台应用、关闭解决方案、打开解决方案和关闭解决方案时，VS 挂起 - #4008
- 针对 d15prerel.25916.01 打开项目时，点击出现挂起 - #3982
- 修复 dotnet/nuget.exe 局部变量的 doc/help 消息 - #3919
- 检查 PackTask 是否存在尾随或前导空格的问题 - #3906
- dotnet
 - dotnetcore 包从 obj(而不是 bin)进行打包 - #3880
- dotnet
 - dotnetcore 包似乎始终将 ProjectReference 版本设置为 1.0.0 - #3874
- dotnet
 - 由于项目引用和，dotnetcore 包失败 - #3865
- ProjectSystemCache.TryGetProjectNameByShortName 中的 LockRecursionException - #3861

- 删除 MSBuild 属性中的空格 - [#3819](#)
- 合并在项目加载上引发的两个项目事件 - [#3759](#)
- `project.assets.json` 文件中的 P2P 库版本不正确 - [#3748](#)
- 由于源无响应且包不可用, 还原出现故障 - [#3672](#)
- 出现大量 MSBuild 错误输出时, nuget.exe 可能挂起 - [#3572](#)
- Blend 的 Restore-on-build 第一次失败, 第二次成功(修复了 VS 方案) - [#2121](#)

DCR

- 将 vsix 从 v2 vsix 迁移到 v3 vsix - [#4196](#)
- NuGet 应具有一个机制, 该机制用于获取指向 MSBuild 中的锁定文件的路径 - [#3351](#)
- 将生成资产添加到 TFM 兼容性检查和资产文件 - [#3296](#)
- 在包目标中定义新的 ProjectCapability“包”, 用于启用与包相关的能力 - [#4146](#)
- 以“GeneratePackageOnBuild”MSBuild 属性为条件, 将包作为后期生成目标运行 - [#4145](#)
- 使用 NuGet 属性 RestoreProjectStyle 创建特定的 NuGet 项目 - [#4134](#)
- 为可传递的项目引用更改调整还原 - [#4076](#)
- 在非 UWP 项目的目标文件中添加 NuGet 属性 - [#4030](#)
- UWP TargetPlatformVersion 支持 - [#3923](#)
- 将项目引用元数据传达到 NuGet 项目系统 - [#3922](#)
- 针对打包模式添加 UI - [#3921](#)
- 旧版 `.csproj` 需要在项目/目标中设置 NugetTargetMoniker 和 RuntimeIdentifiers - [#3854](#)
- 安装包可能与自动还原重叠 - [#3836](#)
- 未加载 VSPackage 时, 不会发生上下文菜单 QueryStatus - [#3835](#)
- 解决方案还原和生成还原时仍显示对话框 - [#3789](#)
- 在 NuGet.Clients 解决方案生成中隔离 VSSDK 版本 - [#3890](#)

RTM 中已修复 GitHub 问题的链接

[问题列表 1](#)

[问题列表 2](#)

[问题列表 3](#)

[问题列表 4](#)

[问题列表 5](#)

NuGet 4.0 RC 发行说明

2020/4/8 • [Edit Online](#)

NuGet 3.5 RTM 发行说明

NuGet 4.0 RC for Visual Studio 2017 重点增加了对 .NET Core 方案的支持，解决了主要客户反馈，并提高了各种方案的性能。此版本引入了一些改进，如支持 PackageReference、作为 MSBuild 目标的 NuGet 命令、后台包还原等。

Bug 修复

- `dotnet pack --version-suffix foo` 中的行为更改 - [#3838](#)
- VS“15”计算机上的 nuget.exe restore 总是失败 - [#3834](#)
- .NET Core“文件”>“新建项目”在还原过程中应阻止生成 - [#3780](#)
- 从 VS2015 迁移到 VS“15”的 ASP.NET Core Web 应用无法还原 - [#3773](#)
- [测试失败]“jQuery 验证”包无法通过 PM UI 卸载 - [#3755](#)
- 将包安装到 UWP `project.json` 时，还应还原父项目 - [#3731](#)
- 修改 NuGet 目标以将包源记录为高度详细而非普通 - [#3719](#)
- dotnet
 - dotnetcore pack3 应默认包含 XML 文档 - [#3698](#)
- 当第一个为无包的源且选中全部源时，从 UI 的批量更新失败 - [#3696](#)
- Nuget pack 命令不包含所有文件 - [#3678](#)
- OOM 问题 - [#3661](#)
- 资产文件的 ProjectFileDependencyGroups 部分应使用项目的库名称 - [#3611](#)
- “dotnet restore”和递归目录 - [#3517](#)
- Restore3 失败记录为警告而非错误 - [#3503](#)
- TFS 问题：“在工作区中找不到[文件]，或者你无权访问”- [#2805](#)
- 在 VS 快速启动搜索框中键入“nuget”，会保留“nuget”前缀 - [#2719](#)
- System.Xml.XmlException：核心属性部分存在无法识别的根元素。行:2, 位置:2 - [#2718](#)
- 文本字段中具有转义 `.nuspec` 或 < 的 > 不再生成 - [#2651](#)
- nuget.exe delete 不会提示输入凭据(处于非交互模式) - [#2626](#)
- nuget.exe delete 警告本地源的 API 密钥(即使没有任何意义) - [#2625](#)
- 安装 EF -pre 包时出现错误，体验不佳 - [#2566](#)
- 在包管理器中更改选择后，Visual Studio 崩溃 - [#2551](#)
- dotnet
 - 使用可变版本时，dotnetcore restore 在简单列表本地存储库上执行区分大小写的 ID 查找 - [#2516](#)
- 对于 V2 源，nuget.exe delete 已被破坏 - [#2509](#)

- nuget.exe push timeout 需要更好的错误消息 - #2503
- 不包含正确导入的工具还原失败且没有任何提示 - #2462
- 即使从 nuget.org 进行安装, NuGet 也会在有专用源时提示输入凭据 - #2346
- ApplicationInsights 2.0 包已列出但尚不存在 - #2317
- VS“15”预览 5 分支中的 UI 延迟 - #3500
- 在针对 UWP 生成的过程中, 首个 OnBuild 事件未还原 - #3489
- PowerShell5 中断 EntityFramework 安装 - #3312
- 将源添加到详细日志记录(考虑为 3.5) - #3294
- NoCache 参数在 nuget 客户端版本 3.4+ 中无效 - #3074
- 当凭据提供程序在 VS 中加载失败时, 请勿中断 NuGet - #2422

功能

- 设置 CI 以运行 x86 - #3868
- 自动还原 3/3: 非阻止 UI - #3658
- 自动还原 2/3: 在指定的计算机上进行后台还原 - #3657
- 还原项目引用以匹配生成行为(递归) - #3615
- VS“15”中支持 DPL - minbar - #3614
- 将设置文件移动到 Program Files - #3613
- 生成的还原属性和目标需要跨目标参与支持 - #3496
- NuGet 还原支持 PackageTargetFallback(以前称为导入) - #3494
- ToolsRef 实现 - #3472
- RID 的 Restore3 - #3465
- 支持添加/删除/更新 PackageRefs 的 NuGet UI - #3457
- 自动还原 1/3: 通过缓存项目还原信息实现指定 API - #3456
- [0] NuGet 还原任务和目标 - #2994
- [1] 在 MSBuild 中启用解决方案级别还原 - #2993
- 在 Visual Studio 中支持凭据提供程序公共扩展性 - #2909
- 递归 NuGet 还原 - #2533
- 无法在 dev15 上加载 Microsoft.TeamFoundation.Client, 对于 VS“15”预览, 需要将 Microsoft.TeamFoundation.Client 版本更新为 15.0 - #2392
- 无法在 VS“15”预览中将 C++ 包安装到 C++ UWP 项目 - #2369
- Nupkg 需要支持 \buildCrossTargeting\ 文件夹 - 并导入 `.targets` / `.props` 来跨越 MSBuild 范围目标 - #3499
- ToolsReference 设计 - #3462
- 修复 NuGet UI 以支持使用 `.csproj` 中的 PackageReferences 进行还原 - #3455

- 将清除缓存按钮添加到 VS 包管理器设置 - #3289

DCR

- 发生自动还原时，应阻止解决方案还原 - #3797
- 从 NuGet 包管理器 UI 进行的 NetCore 安装将安装到每个 TFM，而不是该包支持的那个 TFM - #3721
- 还原指定 API 还需要支持 DotNetCliToolsReferences - #3702
- 将我们的 VS“15”vsix 标记为 systemcomponent - #3700
- 从引用 MS.VS.Services.Client 迁移到 MS.VS.Services.Client.Interactive - #3670
- 应按还原在项目级别考虑 \$(RestoreLegacyPackagesDirectory) - #3618
- 使用单个 TargetFramework 还原到项目不得调整属性 - #3588
- dotnet
 - dotnetcore restore3 foo.csproj 应遵循 projectref 依赖项，同时还原那些依赖项。如生成 - #3577
- “类型”：“平台”依赖项在锁定文件中表示为“类型”：“包” - #2695
- nuget.exe 详细模式应显示下载 URL - #2629
- 将 NuGet xplat 移动到 Microsoft.NetCore.App 和 netcoreapp1.0 - #2483
- 推送 - 从命令行推送时，应该可以替代符号服务器 - #2348
- 合并用于查找全局包路径的代码 - #2296
- 需要一个比 suppressParent 更好的名称 - #2196
- 决定用于 MSBuild 项目的 `project.json` 依赖项名称 - #1914
- 将 SemVer 2.0.0 支持添加到 NuGet.Core - #3383
- 允许具有 `.targets` 的传递依赖项 NuPkgs 在 MSBuild 中可用 - #3342
- 命令行中的 NuGet 还原明显慢于 VS - #3330
- 使包 ID 和版本比较不区分大小写 - #2522
- NoCache 选项不适用于基于 `packages.config` 的还原/安装 (GlobalPackagesFolder) - #1406
- FindPackageByIdResource 资源需要一个默认的缓存上下文和记录器 - #1357

NuGet 3.5 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 3.5 RC 发行说明](#) | [NuGet 4.0 RC 发行说明](#)

Bug 修复

- 包不在 mono-使用 MSBuild 14.1 [#3550](#)
- 更新选项卡上不会选择最新可用版本更新改为选择的当前已安装的版本- [#3498](#)
- MyGet 源的专用 v2 进行身份验证, 并单击"显示更多结果 x"之后修复故障- [#3469](#)
- 日志消息似乎是按相反的顺序为 UI- [#3446](#)
- v3.4.4-Nuget 还原将引发"不支持给定的路径的格式"- [#3442](#)
- NuGet 命令行 3.6 beta 不会遵循-Prop Configuration = Release- [#3432](#)
- 在大型项目-上安装的 Nuget IKVM 慢速[#3428](#)
- nuget.exe 更新-自不断在更新本身- [#3395](#)
- 从 UNC 共享的 3.5 安装/还原已从 3.4.4-性能回归[#3355](#)
- 错误时从包管理 UI net451 项目-安装 Moq [#3349](#)
- 在解决方案级别的安装选项卡不会显示包的版本- [#3339](#)
- xproj `project.json` 从已安装选项卡上的更新将失去状态- [#3303](#)
- 上的 NuGet 包 `.csproj` 会忽略空文件中的元素 `.nuspec` 文件- [#3257](#)
- 在 IIS 中承载的网站项目并不会导致还原失败- [#3235](#)
- 凭据时 v3 终结点将重定向到 v2-不检索从 Nuget.Config [#3179](#)
- NuGet 包未能解析程序集时检索可移植程序集元数据- [#3128](#)
- 找不到 Nuget `msbuild.exe` Mono-上[#3085](#)
- nuget.exe 包不允许预发布标记开头数字- [# 1743年](#)
- 在上 VS2015E-nuget 包安装失败[# 1298年](#)
- allowedVersions 筛选不在解决方案级别的工作[#333](#)
- 还原失败, 随机并具有相同的项已添加密钥。- [#2646](#)
- 不能安装在 Nuget.Common `.csproj` - [# 2635年](#)
- 为每个 ID-两次时使用用户界面来搜索 V2 源, 调用 FindPackagesById [# 2517年](#)
- 包不能依赖于项目- [# 2490年](#)
- nuget.exe 包-排除是记录, 但不是支持- [# 2284年](#)
- 出现错误问题的消息时的 contentFiles 部分 `.nuspec` 无效- [# 1686年](#)
- 推送始终发送整个包两次使用进行身份验证包源- [# 1501年](#)

- 时调用 nuget.exe 更新 *.csproj 项目时没有提供任何信息 `packages.config` - # 1496年
- `packages.config` 从 V2 源-5xx 状态代码, 还原不重试 # 1217年
- 在文件中的 src 两点 `.nuspec` 不起作用- # 2947年
- CoreCLR 还原需要忽略与加密-源 # 2942年
- nuget.exe 推送 403 处理的错误会提示输入凭据- # 2910年
- NuGet 更新通过包管理器中删除属性从 `project.json` - # 2888年
- 尝试加载"NuGet.TeamFoundationServer14", 但 DLL 名称更改为"NuGet.TeamFoundationServer"-
`NuGet.PackageManagement.VisualStudio` # 2857年
- 包管理器 UI 不会显示新更新版本- # 2828年
- 更新包尝试使用包 id, 而不是 package.version-版本 # 2771年
- 如果项目不使用 nuget, nuget 还原 csproj 应错误 (`packages.config` 或 `project.json`)- # 2766年
- TFS 错误"[文件] 不是位于工作区中, 或您没有权限访问该"期间升级或卸载解决方案/项目绑定到 TFS 源代码
管理- # 2739年
- 更新包不会为非目标包的依赖项 # 2724年
- 没有任何方法可设置 Nuget 包管理器 UI 操作的日志详细级别 # 2705年
- nuget 配置无效-VS 2015 VSIX (v3.4.3)- # 2667年
- 在 DefaultPushSource `NuGetDefaults.Config` (`ProgramData\NuGet`) 不起作用- # 2653年
- nuget 3.4.3 发行版-获取值不能为 null 上包内部版本- # 2648年
- 还原时未使用存储的凭据从 Nuget.Config VSTS 源- # 2647年
- [dotnet 还原]-configfile 是相对于项目而不是 cmd 目录中-dir # 2639年
- 版本比较代码-的过多分配 # 2632年
- Nuget.exe 尝试安装相同的多个实例包中并行原因双的写入 - # 2628年
- 依赖关系信息不会缓存的多项目操作- # 2619年
- 安装和更新下载包, 而不首先检查包文件夹 # 2618年
- 如果包源列表为空, 无法添加包源通过 UI (NuGet 3.4.x)- # 2617年
- 如果试图安装包依赖于设计时外观的误导性错误 # 2594年
- 将包安装从 PackageManager 控制台中设置"All"会尝试仅第一个源- # 2557年
- 最新测试版不解压缩 ModernHttpClient- # 2518年
- 在启动时使用自生成 NuGet 3.4.1-VS2015 崩溃 # 2419年
- 更新命令可能是更详细, 如果 i 要求它是这样...- # 2418年
- 本地生成的 VSIX 应作为 CI 生成具有相同的 DLL 和文件。 - #2401
- 修复 NuGet 降级警告中生成- # 2396年
- 无法进行身份验证包源 (3 次) 被永久-阻止 # 2362年
- 从 nuget v3.3 + 安装的包源具有参数时, 不正确还原包的内容时包中包含的-NoCache `.nupkg` 文件- # 2354

年

- 与所有包源，但包缺少从 1 源，Nuget 安装失败- # 2322 年
- [PerfWatson] UI Delay:
nuget.packagemanagement.visualstudio.dll!NuGet.PackageManagement.VisualStudio.VSMSBuildNuGetProjectSystem+*It;>c__DisplayClass_0_+<<AddReference>b__>d.MoveNext - #2285
- 如果单个源失败授权-安装块# 2034 年
- .nuspec 版本范围应重写-IncludeReferencedProjects 版本- # 1983 年
- 更新包非常缓慢-"尝试收集依赖项信息"- # 1909 年
- 隐藏降级 NuGet 包时批处理更新及其依赖项- # 1903 年
- nuget.exe 更新中删除程序集强名称和专用属性。- #1778
- 相对文件路径"DefaultPushSource"- # 1746 年
- 提高冲突解决程序失败消息- # 1373 年
- v3 中的更新包失败，并不在指定的源的包# 1013 年
- 对于包源中使用相对路径会产生问题，若要使用- #865
- 缺少 NUPKG 文件的间接依赖关系已存在具有较低的版本要求-如果从项目生成中的依赖项#759
- 删除项目关闭相应 UI 窗口中，但重命名项目不会重命名 UI 窗口。请注意 PMC 倾听项目重命名和项目删除事件- #670
- [Willow Web 工作负载]创建 Razor v3 WSP 挂起- #3241
- 为特定包的安装/还原失败，出现"包包含多个 nuspec 文件。"- #3231
- 小写 Id & packages.config 方案- #3209
- [3.5-beta2]无法还原"传统"包的包还原#3208
- nuget 包强制将.tt 文件添加到 content 文件夹，无论什么- #3203
- 更新包的 ASP.NET web 应用的生成与文件相关的警告：源- #3194
- nuget 包 csproj (使用 project.json) 如果没有 packOptions 和在 JSON 文件的所有者将崩溃#3180
- 适用于 nuget 包 project.json 将忽略摘要、作者和所有者等-packOptions 标记#3161
- NullReferenceException via NuGet.Packaging.PhysicalPackageFile.GetStream - #3160
- NuGet pack 忽略输出中的依赖关系 .nuspec 有关 project.json - #3145
- 使用回滚更新多个包使项目处于中断状态- #3139
- 任何下的内容文件不会添加为 netstandard 项目- #3118
- 无法打包库面向 .Net 标准正确- #3108
- 文件-> 新建项目-> 类库（可移植）项目失败 VS2015 和 Dev15- #3094
- NuGet 错误-1.0.0-* 不是有效的版本字符串- #3070
- 查找包失败到显示，但安装包的工作原理- #3068
- 错误时在 dev15- 上的"安装包 jquery.validation" #3061

- nuget 包的 xproj 默认设置为无效的目标路径- #3060
- 当已安装的 VS 2015 更新 3 使用的 NuGet 版本 3.5.0 错误发生的对和#3053
- "阻止 packages.config" `project.json` (UWP, 即生成集成) 项目 - #3046
- 更新安装到 preview2-003121, 这是官方 preview2 生成的生成脚本通过 dotnet cli。 - #3045
- 包管理器 UI: 在更新包后不会显示新版本- #3041
- -ApiKey delete 命令行上不读取/发送中 3.5.0-beta- #3037
- 不正确的字符串: 稳定的发行版的包不应具有在预发行版的依赖关系。 - #3030
- OptimizedZipPackage 缓存离开空文件夹- #3029
- 正在创建 PCL (net46 和 windows 10) 项目 get NullRef 异常。 - #3014
- NuGet 更新的 allowedVersions 约束的限制更高版本时应提供信息性消息#3013
- NuGet v3 还原问题- # 2891年
- 凭据插件已退出, 错误为-1 / 错误下载包时使用多个源的凭据提供程序# 2885年
- `project.json` nuget 还原导致重新编译时执行任何操作更改- # 2817年
- 符号包不应在安装或更新-中使用# 2807年
- 与不支持在 repositoryPath 中环境变量 (nuget.exe 执行)- # 2763年
- 使用的可访问性的标签在包管理器用户界面中未标记的 UIElements # 2745年
- 可移植框架使用连字符的由配置文件将被拒绝。 - #2734
- NuGet 包管理器应使它清楚该选项列表中包详细信息不适用于 `project.json` - # 2665年
- nuget.exe 推送/删除不会使用 API 密钥- # 2627年
- 从锁定文件中的删除锁定的属性# 2379年
- NuGet 3.3.0 更新失败, 出现...更多约束中定义 packages.config 阻止此操作。 - #1816
- 从本地源不存在, 则会引发错误的邮件-安装包# 1674年
- "可用的升级"筛选器将显示与版本约束的冲突的升级# 1094年
- 无法更新本机包- # 1291年

功能

- 引用添加 nuget 支持设置为 false 的 CopyLocal #329
- MSBuild 15-nuget.exe 支持# 1937年
- 对包支持。 `csproj` + `project.json` - #1689
- 禁用用户执行任何操作时没有用户正在执行的操作- # 1440年
- NuGet 应添加对的支持 `runtimes/{rid}/nativeassets/{txm}/` - # 2782年
- 添加 NuGet 中缺少的框架兼容性 (这已在 3.x) 的 2.x- # 2720年
- 支持回退的包文件夹- # 2899年
- 设计和实现包类型的概念, 以支持工具的包- # 2476年

- 添加一个 API 来获取全局包文件夹的路径 # 2403 年

- 启用在包的 SemVer 2.0.0 #3356

DCR

- nuget.exe 推送-超时参数不起作用- # 2785 年
- 包描述文本应为可选择- # 1769 年
- 启用生成的 nuget.exe `.props` 并 `.targets` 文件, 以 `.nuproj` 项目 # 2711 年
- 将可扩展性 API 进行比较与导入的框架添加 # 2633 年
- 使用时隐藏依赖关系选项 `project.json` - # 2486 年
- 打印出 nuget.exe 版本标头中详细输出- # 1887 年
- NuGet 需要让用户知道, 在基于 dotnet tfm 升级/安装 PCL 可能会导致问题- #3138
- 错误安装/升级带 tfm 的项目, 则发出警告 = "dotnet" - #3137
- 修复性能问题与 ReShaper 和 NuGet 更新- #3044
- 添加了 netcoreapp11 和 netstandard17 支持- # 2998 年
- 利用 AssemblyMetadata 属性 `.nuspec` 令牌替换- # 2851 年

NuGet 3.5 RC 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 3.5 Beta2 发行说明](#) | [NuGet 3.5 的 RTM 发行说明](#)

3.5 版本侧重于提高质量和性能的 NuGet 客户端。此外，我们对发布了一些功能，例如，为支持[Fallback 文件夹](#)，[PackageType](#)中支持 `.nuspec` 和的详细信息。

问题列表

Bug 修复

- 包的安装/还原失败，出现“包包含多个 `.nuspec` 文件。” - [#3231](#)
- nuget 包添加了强制 `.tt` 文件复制到 content 文件夹，无论什么 - [#3203](#)
- nuget 包 csproj (使用 `project.json`) 如果没有 packOptions 和在 JSON 文件的所有者将崩溃 [#3180](#)
- 适用于 nuget 包 `project.json` 将忽略摘要、作者和所有者等 packOptions 标记 [#3161](#)
- nuget pack 忽略输出中的依赖关系 `.nuspec` 有关 `project.json` - [#3145](#)
- 使用回滚更新多个包使项目处于中断状态 - [#3139](#)
- 任何下的内容文件不会添加为 netstandard 项目 - [#3118](#)
- 无法打包库面向 .Net 标准正确 - [#3108](#)
- 文件 -> 新建项目 -> 类库（可移植）项目失败 VS2015 和 Dev15 - [#3094](#)
- NuGet 错误 1.0.0-* 不是有效的版本字符串 - [#3070](#)
- 查找包失败到显示，但安装包的工作原理 - [#3068](#)
- 错误时在 dev15 上的“安装包 jquery.validation” - [#3061](#)
- 当已安装的 VS 2015 更新 3 使用的 NuGet 版本 3.5.0 错误发生的对和 [#3053](#)
- 包管理器 UI：在更新包后不会显示新版本 - [#3041](#)
- ApiKey delete 命令行上不读取/发送中 3.5.0-beta - [#3037](#)
- 不正确的字符串：稳定的发行版的包不应具有在预发行版的依赖关系。 - [#3030](#)
- 正在创建 PCL (net46 和 windows 10) 项目 get NullRef 异常。 - [#3014](#)
- Nuget 更新的 allowedVersions 约束的限制更高版本时应提供信息性消息 [#3013](#)
- 凭据插件已退出，错误为 -1 / 错误下载包时使用多个源的凭据提供程序 # 2885 年
- nuget 包 - 缺少 Newtonsoft.Json 包依赖项 - [# 2876 年](#)
- 在 Linux/MacOS + Mono-ExecuteSynchronizedCore bug # 2860 年
- 与不支持在 repositoryPath 中环境变量 (nuget.exe 执行) - [# 2763 年](#)
- 修复可访问性问题 - [# 2745 年](#)
- 可移植框架使用连字符的由配置文件将被拒绝。 - [#2734](#)

- NuGet 包管理器应使它清楚该选项列表中包详细信息不适用于 `project.json` - # 2665 年
- NuGet 3.3.0 更新失败, 出现...更多约束中定义 `packages.config` 阻止此操作。- #1816
- 从本地源不存在, 则会引发错误的邮件-安装包# 1674 年
- "升级版提供"筛选器将显示与版本约束的冲突的升级# 1094 年

性能改进

- 性能: 改善 ContentModel 目标框架分析- #3162
- 性能: 避免读取 `runtime.json` 不具有 Rid 还原的文件#3150。CI 计算机上的 ASP.NET Web 应用程序减少了超过 15 秒到 3 秒的示例中进行还原。
- 性能: 程序包管理器控制台 `init.ps1` 加载时间# 2956 年。时间才能打开 `PackageManagerConsole` 改进了在某些情况下从 132s 年到 10 秒。
- 解决在 NuGet 更新-ReSharper 性能问题#3044: 上一个示例项目, 安装包所花的时间减少从 140s 年到 68s。

DCR

- NuGet 需要让用户知道, 在基于 `dotnet tfm` 升级/安装 PCL 可能会导致问题- #3138
- 错误安装/升级带 `tfm` 的项目, 则发出警告 = "dotnet"- #3137
- 添加了 `netcoreapp11` 和 `netstandard17` 支持- # 2998 年
- 打印 NuGet 警告标头内容, 控制台中 `nuget.exe`- # 2934 年
- 利用 `AssemblyMetadata` 属性 `.nuspec` 令牌替换- # 2851 年
- 从锁定文件中的删除锁定的属性# 2379 年
- 符号包应不曾经用于安装或更新 #2807

功能

- 支持回退的包文件夹- # 2899 年
- 设计和实现包类型的概念, 以支持工具的包- # 2476 年
- 若要获取的全局包文件夹的路径的 API # 2403 年
- 本机包更新支持- # 1291 年

NuGet 3.5 Beta2 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 3.5 测试版发行说明](#) | [NuGet 3.5 RC 发行说明](#)

NuGet 3.5 Beta 2 RTM 发布了 2016 年 6 月 27 日, Visual Studio 2013 和 nuget.exe

[完整更改日志](#)

[问题列表](#)

Bug 修复

- 更新的错误消息到不支持的已验证的源-.NET Core 中的密码 decryption # 2942年
- 如果.NET Core 项目处于打开状态的程序包管理器控制台获取的包失败# 2932年
- 在 NuGet push 命令中修复的 403 错误处理# 2910年
- 修复在卸载时 disableSourceControlIntegration 设置绑定到 TFS 源代码管理解决方案中的包的问题为 true 的# 2739年
- 修复包更新要考虑到帐户非目标包- # 2724年
- 使用 MSBuild 详细级别设置为 Nuget 程序包管理器的记录器级别 UI 操作- # 2705年
- 修复 NuGet 配置是在网站项目的 VS 2015 VSIX (v3.4.3)-无效的错误# 2667年
- 修复包期刊 `.csproj` 时将包含的内容文件# 2658年
- 在 DefaultPushSource `NuGetDefaults.Config` (`ProgramData\NuGet`) 不起作用- # 2653年
- 在 Nuget 3.4.3 版本中的值不能为 null 在包创建-修复问题# 2648年
- 还原为 VSTS 源-使用存储的凭据从 Nuget.Config # 2647年
- 性能-版本比较代码中修复过多分配- # 2632年
- Nuget.exe 的多个实例会尝试并行的安装相同包时, 解决问题# 2628年
- 性能-缓存依赖项信息的多项目操作- # 2619年
- 解决问题的包源无法从设置时要从源列表为空-添加# 2617年
- 尝试安装包依赖于设计时外观-时修复令人误解错误# 2594年
- 将包安装从 PackageManager 控制台中设置"All"会尝试仅第一个源- # 2557年
- 修复文件写入时间在将来有 (Mono) 的包的问题# 2518年
- 失败时查找项目中更新命令-显示异常# 2418年
- 从 nuget v3.3 + 安装的包源具有参数时, 不正确还原包的内容时包中包含的-NoCache `.nupkg` 文件- # 2354年
- 修复问题, 与包安装 (所有源) 时从 1 源的包缺少# 2322年
- 如果单个源失败授权-安装块# 2034年

- `.nuspec` 版本范围应重写-IncludeReferencedProjects 版本- # 1983年
- NuGet 3.3.0 更新失败, 出现...更多约束中定义 packages.config 阻止此操作。 - #1816
- nuget.exe 更新中删除程序集强名称和专用属性。 - #1778
- 使用相对文件路径为"DefaultPushSource"-修复问题# 1746年
- 提高更新冲突解决程序失败消息- # 1373年

功能及行为变化

- nuget.exe 推送-超时参数不起作用- # 2785年
- nuget.exe 还原不会产生 `.props` 并 `.targets` 文件, 以 `.nuproj` 项目 (v3.4.3.855 中回归)- # 2711年
- 需要扩展性 API 以比较框架与导入- # 2633年
- 使用时隐藏依赖关系选项 `project.json` - # 2486年
- 打印出 nuget.exe 版本标头中详细输出- # 1887年
- NuGet 应添加对 `/runtimes/ {rid}` /`nativeassets/ {txm}` 的支持 /- # 2782年

NuGet 3.5 测试版发行说明

2018/9/5 • [Edit Online](#)

[NuGet 3.4 发行说明](#) | [NuGet 3.5 Beta2 发行说明](#)

NuGet 3.5 Beta 已于 2016 年 5 月 16, ASP.NET Core 预览工具批的一部分发布。此版本添加了对.NET Core RC2 和 ASP.NET Core RC2 的支持。有关此版本有关的详细信息请参阅 <http://dot.net>。

您可以下载的 VSIX 和 nuget.exe[此处](#)。

NuGet 3.5 Beta 是 3.4.3 中引入的更改的超集版本。有关完整的更改集, 请参阅 3.4.3 发行说明[此处](#)和 3.5 测试版下面的说明。

更新和改进

- 添加了对 ASP.NET Core RC2 和.NET Core rc2 现支持

修补程序

- 给出的修复和改进, 在此版本中, 列表[此处](#)。

NuGet 3.4.4 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 3.4.3 发行说明](#) | [NuGet 3.5 测试版发行说明](#)

此版本的主要重点是 3.4.3 质量的改进与几个修补程序，用于 Visual Studio 扩展插件的 nuget.exe 版本。

您可以下载的 VSIX 和 nuget.exe [此处](#)。

3.4.4-rtm (2016年-05-19)

[完整更改日志](#)

[问题列表](#)

更改

- 包改进：改进的打包符号，与封装 `project.json` 和更多 #606
- 未找到项目更新命令中时显示异常 [#605] (<https://github.com/NuGet/NuGet.Client/pull/605>)
- 从输入读取包类型 `.nuspec` 并 `project.json` 打包时 #603
- 请 NuGet.Shared 不是项目。#602
- 使用推送超时作为 HTTP 响应超时 #599
- 将来时间与包文件不会使用其时间 #597
- 正在更新 `NuGet.Core.dll` 2.12.0 若要解决 XML 问题的版本 #594
- 支持`./NuGet.CommandLine.XPlat-v<详细程度><模式>` #593
- 显示错误还原，而无需 `project.json` 或 `packages.config` #590
- 修复依赖项版本，如果所需的版本不同 #559

NuGet 3.4.3 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 3.4.2 发行说明](#) | [NuGet 3.4.4 发行说明](#)

NuGet 3.4.3 发布于 2016 年 4 月 22，若要解决在 3.4 及后续版本中已发现的几个问题。

您可以下载的 VSIX 和 nuget.exe[此处](#)。

更新和改进

- 改进了 Visual Studio 可靠性。我们已在 Visual Studio 中导致崩溃的 NuGet 中解决一些问题。

修补程序

- 修复了一些授权问题与受密码保护私有 nuget 源。
- 修复了围绕无法还原 PCL 从 `project.json` 与指定的运行时。
- 安装包时，某些客户已遇到间歇性故障。此问题现在已在此版本中修复。
- 修复了导致还原失败，在 C++/cli 项目与 `project.json`。
- 在其中不解压缩正确在 mono 中使用 nuget 时某些包（例如 ModernHttpClient）。此问题现在已在此版本中修复。

有关此版本中的修补程序和改进的完整列表，请参阅问题列表[此处](#)。

NuGet 3.4.2 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 3.4.1 发行说明](#) | [NuGet 3.4.3 发行说明](#)

NuGet 3.4.2 已于 2016 年 4 月 8, 若要解决几个在 3.4 和 3.4.1 中发现的问题, 发布版本。

nuget.exe 3.4.2 RC 现已推出

您可以下载 release candidate 的 nuget.exe 3.4.2[此处](#)。

更新和改进

- 我们显著改进性能的特定方案中, 其中有深入的依赖项关系图的包上的更新需要花费很长时间, 挂起 Visual Studio 中的更新。
- nuget 还原, 而无需网络流量是 2.5 倍 – Visual Studio 中更快 3 倍。
- 除了此更改后, 我们已解决了问题, 我们已达到网络, 两次时 VS UI 中提取更新计数。这是部分负责方面具有丰富经验 3.4/3.4.1 某些超时问题客户。
- 添加了的对 no_proxy 设置

修补程序

- 修复了其中 nuget.org 源中缺少 NuGet 设置或配置更新到 3.4.1 后。
- 修复了在何处对 FindPackagesById 3.4.1 中的大小写更改断开 Artifactory。
- 更正了 FIPS 使用 nuget.exe 的 NuGet 还原导致失败的问题。
- 浏览具有无效的图标 URL 源时, 修复了故障。
- 修复了与合并版本和所有源中的条目的问题。

上面 3.4.2 中的已知问题 Windows x86 命令行 (版本 RC)

将早期的下一步一周前我们碰到 RTM 修复这些问题。

- 如果解决方案文件放置在与项目文件的较低的文件夹层次结构中, 对解决方案运行 nuget 还原将失败。
- 使用 V2 源的包上运行 nuget delete 命令将失败。改为使用 V3 源。

有关此版本中的修补程序和改进的完整列表, 请参阅问题列表[此处](#)。

NuGet 3.4.1 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 3.4 发行说明](#) | [NuGet 3.4.2 发行说明](#)

NuGet 3.4.1 已于 2016 年 3 月 30 日作为 Visual Studio 2015 Update 2 和 Visual Studio 15 预览版版本 3.4 版本中已发现的几个问题同时发布。

更新和改进

- 更正了阻止浏览问题包从具有最小的 Visual Studio 安装的 Visual Studio UI
- 更正了 Visual Studio 查找出现问题 `lucene.net.dll`
- NuGet 扩展安装或更新之后，所有源不应为默认存储库源。可以选择在此功能的配置设置。

我们继续来跟踪我们的 GitHub 问题中，可以在中找到的问题：<http://github.com/nuget/home/issues>

NuGet 3.4 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 3.4 RC 发行说明](#) | [NuGet 3.4.1 发行说明](#)

NuGet 3.4 已于 2016 年 3 月 30 日发布的 Visual Studio 2015 Update 2 和 Visual Studio 15 预览版的一部分，并且已生成具有几项基本原则在脑海中：

- 跨平台支持
- 性能改进
- 次要 UI 改进

以下功能以前在 RC 中添加和已更新或完成 3.4 版：

新增功能

- NuGet 客户端现在支持 gzip 内容编码从存储库
- 从 xproj 项目包中支持 Pdb
- 支持 iOS 和 Android 生成 contentFiles 元素中的操作
- 支持 netstandard 和 netstandardapp 框架名字对象

新用户界面功能

- 尤其是在已安装、更新和合并选项卡上的显著的性能改进
- 聚合的所有包源，同时提供正确的搜索结果合并
- 安装和更新选项卡现在按字母顺序排序
- 添加允许搜索要刷新的刷新按钮
- 在版本列表顶部的最新生成选项

更新和改进

- 中引用的包 `project.json` 具有浮点版本将不会更新在每次生成。相反，它们将在其中更新仅当强制还原、清理、重新生成或修改时 `project.json`。
- 使用 NuGet 配置 UI 时，不能再强制 nuget.org 存储库源到项目配置。
- NuGet 不能再还原在共享项目中的包，也不写入锁定文件。
- 我们改进了网络故障，然后重试处理无法访问或为响应速度缓慢的服务器。
- 键盘和鼠标行为是改进了 Visual Studio 包管理器 UI 中。
- 我们现在支持最新 `project.json` DNX 中的架构。

重大更改

- 包的版本号现在规范化为格式主要。次要。修补程序-预发行版每个主要版本号、次要版本号和补丁都被视为整数并删除任何前导零。作为字符串处理的预发布信息并向其应用任何更改。通过 NuGet 客户端并由 nuget.org 服务提供的搜索查询中使用的这些数字。可以在 NuGet 文档中找到更多详细信息[预发行版版本](#)。

已知问题

- 问题：Windows 10 v1511 用户可能会遇到问题或甚至在 Visual Studio 系统崩溃时 Visual Studio 中的 Powershell 在以下方案中：

- 安装 / 卸载包具有 install.ps1 / uninstall.ps1 脚本
 - 正在加载具有 init.ps1 脚本（如 EntityFramework）的项目
 - 发布的 web 内容
- **解决方法：**确保 Windows 10 安装有最新的修补程序应用、专用 (KB 3124263) 的 2016 年 1 月或更高版本的更新。更多详细信息位于[GitHub 问题 #1638](#)。
 - 问题：NuGet v2 协议重定向已断开。将请求重定向到备用主机的自定义 NuGet 存储库不接受重定向请求。
 - **解决方法：**若要解决此问题，请将存储库 URI 配置设置来重定向的服务器位置中。有关详细信息，请参阅[GitHub 拉取请求 #387](#)。

我们继续来跟踪我们的 GitHub 问题中，可以在[中找到的问题：<http://github.com/nuget/home/issues>](http://github.com/nuget/home/issues)

NuGet 3.4 RC 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 3.3 发行说明](#) | [NuGet 3.4 发行说明](#)

NuGet 3.4 RC 已发布于 2016 年 3 月 3 日与 Visual Studio 2015 Update 2 RC 一起，并且已生成具有几项基本原则在脑海中：

- 跨平台支持
- 性能改进
- 次要 UI 改进

以下功能都在此 RC 中，提供了计划 3.4 的最终版本的详细信息。

新增功能

- NuGet 客户端现在支持 gzip 内容编码从存储库
- 从 xproj 项目包中支持 Pdb
- 支持 iOS 和 Android 生成 contentFiles 元素中的操作
- 支持 netstandard 和 netstandardapp 框架名字对象

新用户界面功能

- 尤其是在已安装、更新和合并选项卡上的显著的性能改进
- 安装和更新选项卡现在按字母顺序排序
- 添加允许搜索要刷新的刷新按钮

更新和改进

- 中引用的包 `project.json` 具有浮点版本将不会更新在每次生成。相反，它们将在其中更新仅当强制还原、清理、重新生成或修改时 `project.json`。
- 使用 NuGet 配置 UI 时，不能再强制 nuget.org 存储库源到项目配置。
- NuGet 不能再还原在共享项目中的包，也不写入锁定文件。
- 我们改进了网络故障，然后重试处理无法访问或为响应速度缓慢的服务器。
- 键盘和鼠标行为是改进了 Visual Studio 包管理器 UI 中。
- 我们现在支持最新 `project.json` DNX 中的架构。

已知问题

我们继续来跟踪我们的 GitHub 问题中，可以在中找到的问题：<http://github.com/nuget/home/issues>

NuGet 3.3 发行说明

2020/1/31 · · [Edit Online](#)

[NuGet 3.2.1 发行说明](#) | [NUGET 3.4-RC 发行说明](#)

NuGet 3.3 于2015年11月30日发布，具有大量用户界面更新和命令行功能，以及对 NuGet 客户端的有用修补程序的集合。

新增功能

- 引入了凭据提供程序，使 NuGet 命令行客户端能够与经过身份验证的源无缝协作。NuGet 文档中提供了有关[如何安装 Visual Studio Team Services 凭据提供程序](#)以及[如何配置 nuget 客户端以使用该提供程序](#)的说明。

新的用户界面功能

- 单独的浏览、已安装和更新可用选项卡
- 更新可用徽章，指示包含可用更新的包数
- 包列表中的包徽章，用来指示包是否已安装或是否有可用更新
- 添加到包列表的下载计数和作者
- 包列表中可用的最高版本号和当前安装的版本号
- 允许从包列表快速安装、更新和卸载的操作按钮
- 包详细信息面板中更清晰的操作按钮
- 包更新日期包详细信息面板
- 解决方案视图中的合并面板
- 项目的可排序网格和解决方案视图上安装的版本号

新的命令行功能

在此版本中，我们引入了 `add` 和 `init` 命令来初始化基于文件夹的存储库，如[nuget.exe 引用](#)中所述。通过此文件夹结构构造和维护的存储库将[提供重要的性能优势](#)，如博客中所述。

ContentFiles

现在，通过新的 `contentFiles` 文件夹 `project.json` 托管项目支持内容，并 `.nuspec` `contentFiles` 元素表示法。包作者可以更直接指定此内容以与项目系统交互。有关如何在 `.nuspec` 文档中配置 `contentFiles` 的详细信息，请参阅[Nuspec 引用](#)。

NuGet 局部变量缓存管理

NuGet 命令行已更新，以包括有关如何管理工作站上的本地缓存的信息。[NuGet 命令行参考](#)中提供了有关局部变量命令的详细信息。

修复的问题

值得注意的问题

- NuGet 命令行已还原对在 Mono 上使用解决方案文件还原包的支持- [1543](#)

3.3 版本中解决的问题的完整列表可在 GitHub 上的[3.3 里程碑](#)下找到。

3.3 命令行版本中修复的问题列表记录在[3.3 命令行里程碑](#)中。

已知问题

我们会继续跟踪 GitHub 问题列表中的问题, 可在以下位置找到: <http://github.com/nuget/home/issues>

NuGet 3.2.1 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 3.2 发行说明](#) | [NuGet 3.3 发行说明](#)

有关命令行中的 NuGet 3.2.1 已于 2015 年 10 月 12 日发布了几个优化和 3.2 版本的修补程序，可从dist.nuget.org。

改进

- NuGet 现在使用配置文件时使用的原始大小写 `NuGet.Config`。这一点在区分大小写的操作系统上 [1427 年](#)
- NuGet 还原现在将忽略 dnx 项目 (`*.xproj`) 应具有处理 `dnu` [1227 年](#)
- 使用时，优化网络利用率 `index.json` 并将其打包注册数据 [1426 年](#)
- 处理，与 v2 服务更加稳定可靠的改进的资源下载 [1448 年](#)

修补程序

- NuGet 更新是否能够正确更新 `.csproj` / `.vcxproj` 引用 [1483 年](#)
- 现在阻止本地.nuget 文件夹时 `SpecialFolders.UserProfile` 找不到创建 [1531 年](#)
- 改进了在下载过程中损坏的包在本地缓存中的处理 [1405 年](#) [1157 年](#)

可以在 NuGet GitHub 中找到的命令行和 Visual Studio 扩展已解决的问题的完整列表 [3.2.1 里程碑](#)

已知问题

我们继续来跟踪我们的 GitHub 问题中，可以在中找到的问题：<http://github.com/nuget/home/issues>

NuGet 3.2 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 3.2 RC 发行说明](#) | [NuGet 3.2.1 发行说明](#)

发布 NuGet 3.2 2015 年 9 月 16 日作为一系列改进和修复了 3.1.1 发布并且可从这两个[dist.nuget.org](#)并[Visual Studio 库](#)。

新增功能

- 位于相同的文件夹的项目现在可以具有不同 `project.json` 特定于每个项目的文件夹中的文件。对于每个项目，命名 `project.json` 文件 `{ProjectName}.project.json`，NuGet 会相应地提供首选项设置为每个项目的配置。与安装的 Windows 10 工具 1.1 版才支持此[1102年](#)
- NuGet 客户端支持指定全局 NUGET_PACKAGES 环境变量指定的位置中使用的共享的全局包文件夹 `project.json` 托管使用 Windows 10 工具 1.1 版的项目。

命令行的更新

这是支持 NuGet v3 服务器的 nuget.exe 客户端的第一个版本，使用还原项目包管理 `project.json` 文件。

没有大量的经过身份验证源的问题已解决在此版本中以提高与客户端之间的交互。

- 安装 / 还原交互只能提交到经过身份验证数据源的初始请求凭据[1300年](#), [456](#)
- Push 命令不能解决从配置的凭据[1248年](#)
- 用户代理和标头现在提交到 NuGet 存储库，以便统计信息跟踪- [929](#)

我们做了大量改进，可尝试使用远程 NuGet 存储库时更好地处理网络故障：

- 改进了错误消息时无法连接到远程源- [1238年](#)
- 更正了 NuGet 还原命令时将发生错误情况的正确返回 1 [1186年](#)
- 现在重试网络连接最多 5 次尝试 HTTP 5xx 故障-的情况下每隔 200 毫秒[1120年](#)
- 改进的服务器重定向响应的处理期间推送命令- [1051年](#)
- `nuget install -source` 现在支持作为自变量的 Nuget.Config 中的 URL 或存储库名称[1046年](#)
- 在还原过程中已不在存储库位于的缺失包现在报告为错误而不是警告[1038年](#)
- 更正的 Unix/Linux 方案-\r\n multipartwebrequest 处理[776](#)

有大量的各种命令的问题的修补程序：

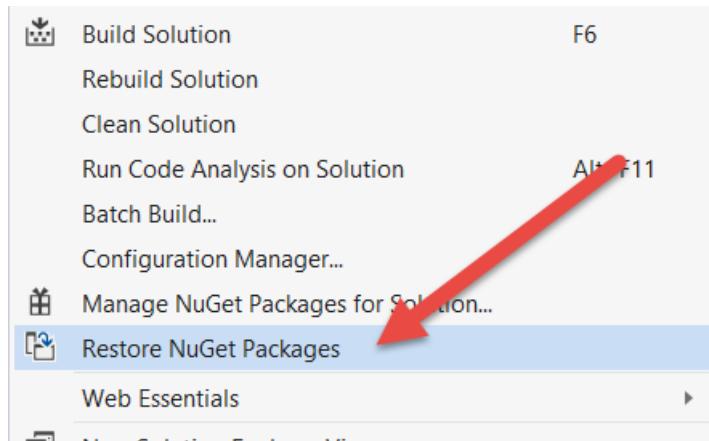
- Push 命令不会再对包源的 PUT 前 GET [1237年](#)
- 列表命令不再重复版本号- [1185年](#)
- 包使用的生成参数现在正确地支持 C# 6.0- [1107年](#)
- 已更正的问题尝试打包 F # 项目使用 Visual Studio 2015-生成[1048年](#)
- 不还原现在进行任何操作时的包已经存在- [1040年](#)
- 改进的错误消息何时 `packages.config` 文件的格式不正确- [1034年](#)
- 更正了带-SolutionDirectory 开关还原命令，可以使用相对路径- [992](#)
- 改进了更新命令，以支持解决方案级更新- [924](#)

在此版本中已解决的问题的完整列表可在 NuGet GitHub[命令行里程碑](#)。

Visual Studio 扩展更新

Visual Studio 中的新增功能

- 新的上下文菜单项已添加到解决方案资源管理器允许包在还原时不用生成解决方案的解决方案节点上 ([1274年](#))。



更新和修补程序在 Visual Studio 中

已验证的源的修补程序已累加起来，并且也在扩展中解决。以下身份验证项还已扩展中得到解决：

- 现在正确，正确处理 NuGet v3 已验证的源而不是 v2 已经过身份验证源- [1216年](#)
- 在项目中使用的身份验证凭据的已更正的请求 `project.json` 以及与 v2 源-通信 [1082年](#)

网络连接有影响 Visual Studio 中的用户界面和我们解决了此与以下修补程序：

- 改进的包版本的本地缓存维护 [1096年](#)
- 连接到源到无法再尝试将其视为 v2 源-v3 时更改失败行为 [1253年](#)
- 用多个包源的安装包时，现在阻止安装故障 [1183年](#)

我们改进了与生成操作之间的交互的处理：

- 现在，继续生成项目，如果还原包，为单个项目失败- [1169年](#)
- 将包安装到解决方案中的另一个项目依赖的项目强制解决方案重新生成- [981](#)
- 更正失败的包安装到项目的正确回滚更改 [1265年](#)
- 更正了无意中的删除 `developmentDependency` 属性中的包 `packages.config` - [1263年](#)
- 调用 `install.ps1` 现在具有适当 `$package.AssemblyReferences` 传递的对象 [1245年](#)
- 不再阻止卸载包在 UWP 项目中的项目时处于错误状态- [1128年](#)
- 其中包含的各种解决方案 `packages.config` 并 `project.json` 而无需第二个现在正确地生成项目生成操作- [1122年](#)
- 如果链接或位于不同的文件夹的正确查找 `app.config` 文件 [1111年](#), [894](#)
- UWP 项目现在可以安装取消列出的包- [1109年](#)
- 当解决方案处于不处于已保存的状态-现在允许包还原 [1081年](#)

处理的配置文件已更正的更新：

- 从在后续版本中的包不能再删除目标文件传送 `project.json` 托管的项目- [1288年](#)
- 在 ASP.NET 5 解决方案生成的过程不能再修改 Nuget.Config 文件 [1201年](#)
- 允许在包更新的过程不能再更改版本约束 [1130年](#)
- 锁定文件现在保持锁定状态期间生成的 [1127年](#)
- 现在修改 `packages.config` 并不在更新的过程中重写该 [585](#)

与 TFS 源代码管理的交互得到改进：

- 不能再失败的包的绑定到 TFS 的安装 [1164年](#), [980](#)

- 更正了的 NuGet 用户界面, 以允许 TFS 2013 集成- [1071年](#)
- 更正了对已还原到正确来自的 packages 文件夹中的包的引用[1004年](#)

最后, 我们还改进了这些项:

- 日志消息的详细级别可以减少 `project.json` 托管项目- [1163年](#)
- 现在, 正确的用户界面中显示安装的包版本[1061年](#)
- 现在指定其 nuspec 中的依赖项范围的包具有稳定的包版本-安装这些依赖项的预发布版本[1304年](#)

可以在 NuGet GitHub 中找到 Visual Studio 扩展的已解决的问题的完整列表[3.2 里程碑](#)

已知问题

我们继续来跟踪我们的 GitHub 问题中, 可以在中找到的问题: <http://github.com/nuget/home/issues>

NuGet 3.2 RC 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 3.1.1 发行说明](#) | [NuGet 3.2 发行说明](#)

发布 NuGet 3.2 的候选发布版本 2015 年 9 月 2 日作为一系列改进和修复了 3.1.1 版本。此外，它们被首次发布到新 dist.nuget.org 存储库的第一个版本。

新增功能

- 位于相同的文件夹的项目现在可以具有不同 `project.json` 特定于每个项目的文件夹中的文件。对于每个项目，命名 `project.json` 文件 `{ProjectName}.project.json` 和 NuGet 将正确引用并相应地为每个项目中使用该内容。这支持一项新功能[1102年](#)
- `NuGet.Config` 现在支持为相对路径`-globalPackagesFolder` [1062年](#)

命令行的更新

这是支持 NuGet v3 服务器的 `nuget.exe` 客户端的第一个版本，使用还原项目包管理 `project.json` 文件。

没有大量的经过身份验证源的问题已解决在此版本中以提高与客户端之间的交互。

- 安装 / 还原交互只能提交到经过身份验证数据源的初始请求凭据[1300年, 456](#)
- Push 命令不能解决从配置的凭据[1248年](#)
- 用户代理和标头现在提交到 NuGet 存储库，以便统计信息跟踪- [929](#)

我们做了大量改进，可尝试使用远程 NuGet 存储库时更好地处理网络故障：

- 改进了错误消息时无法连接到远程源- [1238年](#)
- 更正了 NuGet 还原命令时将发生错误情况的正确返回 1 [1186年](#)
- 现在重试网络连接最多 5 次尝试 HTTP 5xx 故障-的情况下每隔 200 毫秒[1120年](#)
- 改进的服务器重定向响应的处理期间推送命令- [1051年](#)
- `nuget install -source` 现在支持作为自变量的 Nuget.Config 中的 URL 或存储库名称[1046年](#)
- 在还原过程中已不在存储库位于的缺失包现在报告为错误而不是警告[1038年](#)
- 更正的 Unix/Linux 方案-\r\n multipartwebrequest 处理[776](#)

有大量的各种命令的问题的修补程序：

- Push 命令不会再对包源的 PUT 前 GET [1237年](#)
- 列表命令不再重复版本号- [1185年](#)
- 包使用的生成参数现在正确地支持 C# 6.0- [1107年](#)
- 已更正的问题尝试打包 F# 项目使用 Visual Studio 2015-生成[1048年](#)
- 不还原现在进行任何操作时的包已经存在- [1040年](#)
- 改进的错误消息何时 `packages.config` 文件的格式不正确- [1034年](#)
- 更正了与 `restore` 命令 `-SolutionDirectory` 开关以使用相对路径- [992](#)
- 改进了更新命令，以支持解决方案级更新- [924](#)

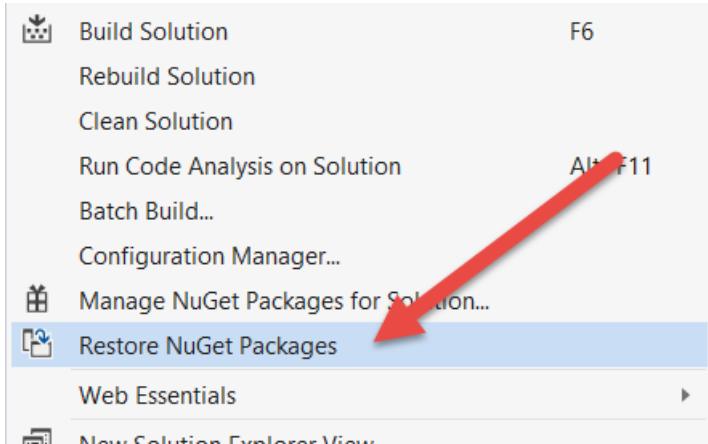
在此版本中已解决的问题的完整列表可在 NuGet GitHub[命令行里程碑](#)。

Visual Studio 扩展更新

Visual Studio 中的新增功能

- 新的上下文菜单项已添加到解决方案资源管理器允许包在还原时不用生成解决方案的解决方案节点上 ([1274](#))

年)。



更新和修补程序在 Visual Studio 中

已验证的源的修补程序已累加起来，并且也在扩展中解决。以下身份验证项还已扩展中得到解决：

- 现在正确，正确处理 NuGet v3 已验证的源而不是 v2 已经过身份验证源- 1216 年
- 在项目中使用的身份验证凭据的已更正的请求 `project.json` 以及与 v2 源-通信 1082 年

网络连接有影响 Visual Studio 中的用户界面和我们解决了此与以下修补程序：

- 改进的包版本的本地缓存维护 1096 年
- 连接到源到无法再尝试将其视为 v2 源-v3 时更改失败行为 1253 年
- 用多个包源的安装包时，现在阻止安装故障 1183 年

我们改进了与生成操作之间的交互的处理：

- 现在，继续生成项目，如果还原包，为单个项目失败- 1169 年
- 将包安装到解决方案中的另一个项目依赖的项目强制解决方案重新生成- 981
- 更正失败的包安装到项目的正确回滚更改 1265 年
- 更正了无意中的删除 `developmentDependency` 属性中的包 `packages.config` - 1263 年
- 调用 `install.ps1` 现在具有适当 `$package.AssemblyReferences` 传递的对象 1245 年
- 不再阻止卸载包在 UWP 项目中的项目时处于错误状态- 1128 年
- 其中包含的各种解决方案 `packages.config` 并 `project.json` 而无需第二个现在正确地生成项目生成操作- 1122 年
- 如果链接或位于不同的文件夹的正确查找 `app.config` 文件 1111 年， 894
- UWP 项目现在可以安装取消列出的包- 1109 年
- 当解决方案处于不处于已保存的状态-现在允许包还原 1081 年

处理的配置文件已更正的更新：

- 从在后续版本中的包不能再删除目标文件传送 `project.json` 托管的项目- 1288 年
- 在 ASP.NET 5 解决方案生成的过程不能再修改 Nuget.Config 文件 1201 年
- 允许在包更新的过程不能再更改版本约束 1130 年
- 锁定文件现在保持锁定状态期间生成的 1127 年
- 现在修改 `packages.config` 并不在更新的过程中重写该 585

与 TFS 源代码管理的交互得到改进：

- 不能再失败的包的绑定到 TFS 的安装 1164 年， 980
- 更正了的 NuGet 用户界面，以允许 TFS 2013 集成- 1071 年
- 更正了对已还原到正确来自的 packages 文件夹中的包的引用 1004 年

最后，我们还改进了这些项：

- 日志消息的详细级别可以减少 `project.json` 托管项目- 1163 年

- 现在，正确的用户界面中显示安装的包版本1061年

可以在 NuGet GitHub 中找到 Visual Studio 扩展的已解决的问题的完整列表[3.2 里程碑](#)

已知问题

我们继续来跟踪我们的 GitHub 问题中，可以在[中找到的问题](#): <http://github.com/nuget/home/issues>

NuGet 3.1.1 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 3.1 发行说明](#) | [NuGet 3.2 RC 发行说明](#)

发布 NuGet 3.1.1 2015 年 7 月 27 日作为修补程序更新为 3.1 VSIX 通过特定于受影响的 Powershell 策略实现的 bug 修复。<https://github.com/NuGet/Home/issues/974>

NuGet 3.1 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 3.0 发行说明](#) | [NuGet 3.1.1 发行说明](#)

NuGet 3.1 2015 年 7 月 27 日作为发布对通用 Windows 平台 SDK 的捆绑扩展插件用于 Visual Studio 2015。我们提供了此版本中的使用 Windows 平台 SDK，以便 Windows 开发体验无法利用以前已经开始的 NuGet 跨平台工作。此 NuGet 扩展版本才可用于 Visual Studio 2015。

我们建议有权访问不可用，因为我们始终要发布包含 bug 修复和新功能更新的最新版本的 Visual Studio 库更新这些开发人员。

NuGet Visual Studio 扩展

问题和在此版本中的功能标记使用 GitHub 上["3.1 RTM UWP 可传递支持"里程碑](#)总的来说，我们已关闭的 67 3.1 发行版中的问题。

新增功能

- `project.json` 支持 Windows UWP 和 ASP.NET 5 支持
- 可传递的包安装

说明和定义的这些功能可以将其他位置的文档中找到。

不推荐使用

以下功能将不再可用于 Visual Studio 2015：

- 不能再将解决方案级别包安装

以下功能将不再适用于 Visual Studio 2015 和使用的项目 `project.json` 规范

- `install.ps1` 和 `uninstall.ps1` - 这些脚本将包安装过程中忽略、还原、更新和卸载
- 配置转换将被忽略
- 将执行，但不是复制到项目的内容。
 - 团队正在努力重新实现此功能、关注讨论和在进度：<https://github.com/NuGet/Home/issues/627>

已知问题

没有随此版本的已知问题的数量。

- 版本与 Windows 10 SDK 的版本 3.1 安装将以前安装的 NuGet 任何的扩展版本降级

NuGet 命令行

NuGet 命令行可执行文件已更新，并且移动到新的可分发位置，以便历史版本的 `nuget.exe` 继续可用。可以为 Windows 在下载 `nuget.exe` 3.1 beta 的版本：<http://dist.nuget.org/win-x86-commandline/v3.1.0-beta/nuget.exe>

新的可分发位置位于在 `dist.nuget.org` 主机上，遵循此模板的文件夹结构：

```
{platform supported}/{version}/nuget.exe
```

新增功能

- `nuget.exe` 可以恢复，并将包安装到使用的项目 `project.json` 文件。

- nuget.exe 可以连接到并使用在 NuGet v3 协议: <https://api.nuget.org/v3/index.json>

已知问题

1. 无法执行包针对 `project.json` 文件 - [928](#)
2. 不支持 Mono - [1059年](#)
3. 未本地化 - [1058年](#), [1057年](#)
4. 未签名, 就像现有 <http://nuget.org/nuget.exe> - [1073年](#)

NuGet 3.0 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 3.0 RC2 发行说明](#) | [NuGet 3.1 发行说明](#)

NuGet 3.0 作为捆绑扩展到 Visual Studio 2015 发布于 2015 年 7 月 20 日。我们推送来提供此版本中的使用 Visual Studio，以便完成更新后的 NuGet 3.0 体验将可用于新的 Visual Studio 用户。此 NuGet 扩展版本才可用于 Visual Studio 2015。

我们建议有权访问不可用，因为我们要发布包含对 Windows 10 开发的支持的 Visual Studio 2015 发布更新的最新版本的 Visual Studio 库更新这些开发人员。

总的来说，我们已关闭的 240 问题在 3.0 版本中，并可以检查[GitHub 上的问题的完整列表](#)。

已知问题

出了很多已知问题随此版本中，且所有这些项在我们计划的 3.1 版本，以便与 Windows 10 版本保持一致上年 7 月 29, 中修复。您将能够更新你的 Visual Studio 扩展从库或之后的日期来解决这些已知的问题。

- 在预览窗口上的“不再显示此信息”标签和包描述窗口中的“作者”标签不提供翻译。
- 当你使用 TFS 处理项目源代码管理时，NuGet 不能显示包管理器用户界面如果 Nuget.Config 文件标记为只读的。
 - **解决方法**签出该文件从 TFS。
- 使用 Visual Studio 深色主题时，黄色 NuGet Powershell 窗口中的“重新启动栏”中的文本不可见。
 - **解决方法**使用 Visual Studio 浅色主题。

已解决的热门问题的摘要

- [刷新包管理器窗口时将调用频繁网络更新](#)
- [更改为安装在包管理器中的视图时，延迟滚动](#)
- [网络调用应在后台线程上运行](#)
- [添加了不显示预览窗口复选框](#)
- [添加了的过程限制以减少处理器使用情况](#)
- 改进的可移植类库参考处理
 - <https://github.com/NuGet/Home/issues/562>
 - <https://github.com/NuGet/Home/issues/454>
 - <https://github.com/NuGet/Home/issues/440>
- [记忆式键入功能服务是区分大小写](#)
- [更新重新引入基本身份验证凭据](#)
- [改进的错误日志记录](#)
- [改进了的 powershell 调用更新包时的错误消息](#)
- [修复了了解有关选项的信息链接，以防止在 Windows 10 上崩溃，](#)
- [请记住预发行版复选框设置](#)
- [改进了的收集性能的方式跨解决方案中的项目缓存结果](#)
- [可以并行收集多个包](#)
- [删除安装包的强制命令](#)

请密切关注[我们的博客](#)的更多进度和公告如我们准备好提供对 Windows 10 开发的支持。

NuGet 3.0 RC2 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 3.0 RC 发行说明](#) | [NuGet 3.0 发行说明](#)

作为临时版本可从 Visual Studio 2015 扩展库 2015 年 6 月 3 日发布 NuGet 3.0 RC2 并[Codeplex](#)。此版本中有许多重要的 bug 修复和性能改进，我们认为已重要释放之前已完成的 Visual Studio 2015 版本。此 NuGet 扩展版本才可用于 Visual Studio 2015。

总的来说，我们已关闭的 158 问题在此版本中，并可以检查[GitHub 上的问题的完整列表](#)。

已解决的热门问题的摘要

- 刷新包管理器窗口时将调用频繁网络更新
- 更改为安装在包管理器中的视图时，延迟滚动
- 网络调用应在后台线程上运行
- 添加了不显示预览窗口复选框
- 添加了的过程限制以减少处理器使用情况
- 改进的可移植类库参考处理
 - <https://github.com/NuGet/Home/issues/562>
 - <https://github.com/NuGet/Home/issues/454>
 - <https://github.com/NuGet/Home/issues/440>
- 记忆式键入功能服务是区分大小写
- 更新重新引入基本身份验证凭据
- 改进的错误日志记录
- 改进了的 powershell 调用更新包时的错误消息

下载这更新到 [NuGet 扩展](#) 从 [Codeplex](#) 和请密切关注[我们的博客](#)详细进度和 NuGet 3.0 的公告！

NuGet 3.0 RC 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 3.0 测试版发行说明](#) | [NuGet 3.0 RC2 发行说明](#)

NuGet 3.0 RC 与 Visual Studio 2015 RC 版本发布于 2015 年 4 月 29 日。此版本包含重要的 bug 修复、性能改进和更新以支持新框架的数量。它是仅适用于 Visual Studio 2015。

对性能方面的不懈关注

稳定性和 NuGet 查询的性能仍是我们的重心一个热门的话题。此版本中，您将会看到非常快速的搜索操作中的 NuGet UI 和网站。我们正在监视服务以及如何使用该服务，以便我们可以继续调整这些操作。

已解决的重要问题

为了达到稳定状态的 NuGet 客户端，作为此发布的一部分解决许多问题。下面是只是简要的一些更重要的问题已解决的列表：

- ASP.NET 5 K framework 的重命名的一部分，已更新了框架名字对象来处理 dnx 和 dnxcore[链接](#)
- 从 Visual Studio UI 中的链接添加帮助文档[链接](#)
- 更好地处理中的复杂引用 `.nuspec` 使用以逗号分隔的框架引用[链接](#)
- 修复了对日语区域性的支持[链接](#)
- 更新的客户端以允许 ASP.NET 5 项目使用新的 v3 终结点[链接](#)
- 更新为更好的句柄与源代码管理的包文件夹[链接](#)
- 修复了对附属包的支持[链接](#)
- 更正了对特定于框架的内容文件的支持[链接](#)

GitHub 存在革新

我们做出了一些更改到我们[源代码存储库在 GitHub 上的](#)。如果使用 Visual Studio 的 NuGet 客户端、Powershell 命令或命令行中有任何问题可执行文件可以记录这些问题和监视其进度上我们[主页 GitHub 存储库问题列表](#)。我们在跟踪中的库的问题我们[GitHub NuGetGallery 存储库](#)。

请继续关注

请密切关注[我们的博客](#)详细进度和 NuGet 3.0 的公告！

NuGet 3.0 测试版发行说明

2018/9/5 • [Edit Online](#)

[NuGet 3.0 预览版发行说明](#) | [NuGet 3.0 RC 发行说明](#)

Visual Studio 2015 CTP 6 版本 2015 年 2 月 23 日发布了 NuGet 3.0 试用版。此版本是指大量与我们的团队，我们有了大量的体系结构和性能改进，若要共享，并且我们很高兴以开始优化我们 nuget.org 的服务上的性能设置。

我们强烈建议您在安装此新版本之前卸载任何以前版本的 Visual Studio 2015 的 NuGet 扩展。如果有此版本的扩展中的任何问题，我们建议你还原到[早期版本](#)以用于 Visual Studio 2015 预览版。

Visual Studio 2012 和更高

可在 Visual Studio 2015 CTP 6 扩展库中安装此 NuGet 3.0 试用版。我们正在努力推出预览版删除 Visual Studio 2012 和 Visual Studio 2013 很快。我们以前共享到我们的意图[停止更新适用于 Visual Studio 2010](#)，和我们未做出难以决定。

新的客户端/服务器 API

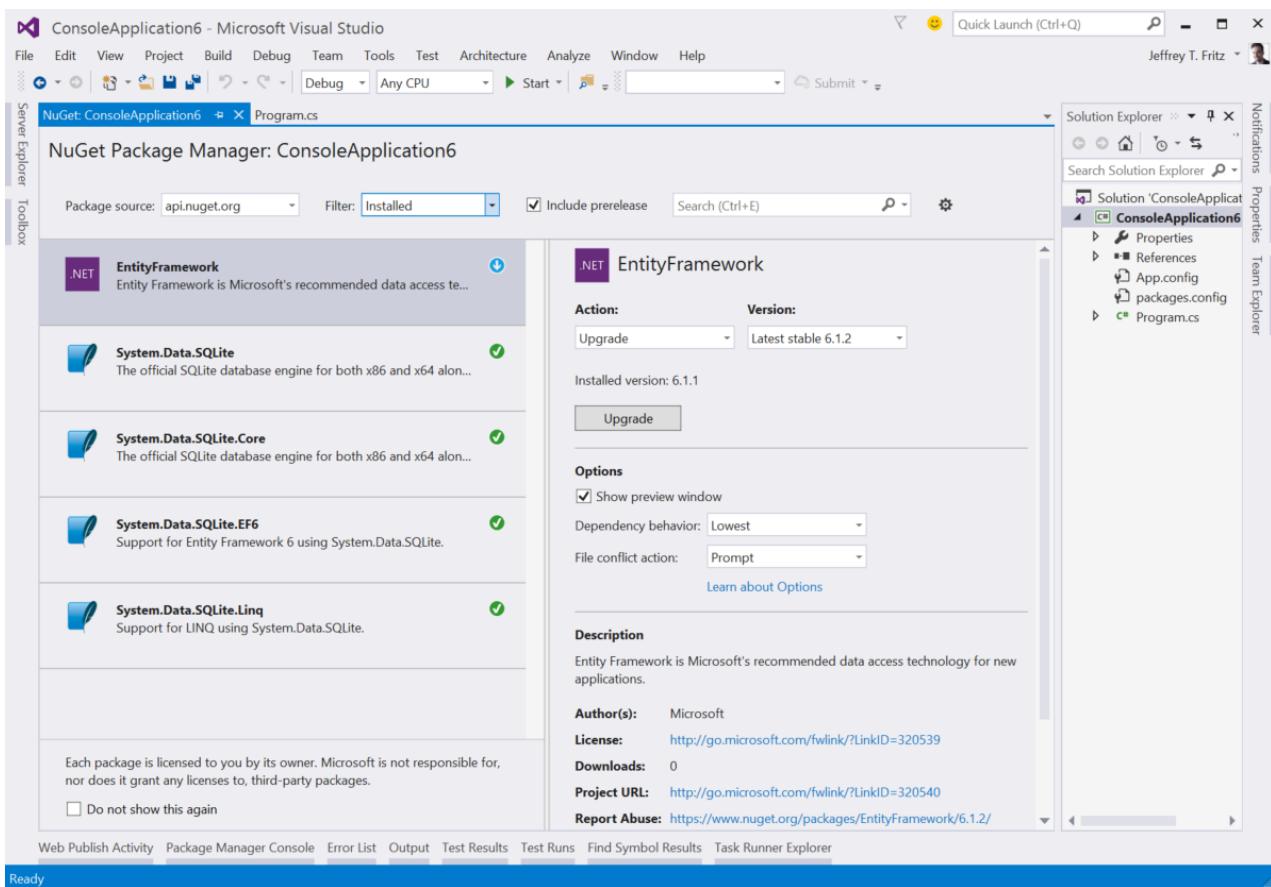
我们一直在 NuGet 的客户端/服务器协议一些实现细节。我们所做的工作是为 NuGet，专为高可用性的关键方案，例如包还原和安装包创建"API v3"。新的 API 基于 REST 和超媒体和我们所选[JSON-LD](#)作为我们的资源格式。

在 NuGet 3.0 试用版位中，可以看到称为"api.nuget.org"包源下拉列表中的新包源。如果您选择的包源，我们将使用我们的新 API，而是要连接到 nuget.org。在 NuGet 3.0 RC 中，此新 API 基于 v3 的包源将替换基于 v2"nuget.org"包源。我们建议禁用所有其他公共包源，并将仅 api.nuget.org 保留为仅对公共包存储库。

我们已投入了大量的时间构建我们的 v3 API，并将继续为旧客户端查找来访问公共存储库保持标准 v2 API。

更新后的 UI

我们已改进在此版本中包括的组合框，将允许您选择要与包执行的操作和转换到屏幕的选项区域中的复选框的预览按钮的用户界面。选项区域不再可折叠，现在提供描述可用的选项的帮助链接。



操作日志记录

我们删除包含的日志记录信息的快速的显示效果并在安装或卸载时隐藏模式窗口。此窗口时真正想要查看的信息，否则将无法从其复制和粘贴添加任何值。相反，我们现在重定向所有输出记录到输出窗口的包管理器窗格中。我们认为这是更舒适和类似于你想要检查的典型生成报表。

专注于性能

我们做了大量的更改提高性能的 NuGet 搜索和提取操作的名称。这是从我们的客户，我们首要问题，我们希望确保我们在此版本中解决。我们已经优化我们的服务器，构建新的 CDN，并改进了匹配逻辑为您的关系更密切希望提供的查询和更快包搜索结果。

随着我们逐步开发的 NuGet 3.0 的这个阶段，我们将优化和监视 nuget.org 服务，以确保我们提供改进的体验。我们不要计划参与任何停机时间，但将会添加和更改服务中的资源。请关注我们[twitter 订阅源](#)有关当我们更改的服务配置的详细信息。

使用 NuGet 生成 NuGet

我们现在已为多个组件内置到 NuGet 包本身被重建我们 NuGet 客户端。此重复使用我们自己的库的迫使我们构建的是重复使用，可以正确打包的组件。我们已能够消除重复的代码，了解到如何更好地配置我们的开发过程，以支持需要生成在我们的解决方案整个包。查找博客文章很快我们将讨论有关代码项目的构建方式我们生成过程的工作原理。

请继续关注

请密切关注[我们的博客](#)详细进度和 NuGet 3.0 的公告！

NuGet 3.0 预览版发行说明

2018/9/5 • [Edit Online](#)

[NuGet 2.9 RC 发行说明](#) | [NuGet 3.0 测试版发行说明](#)

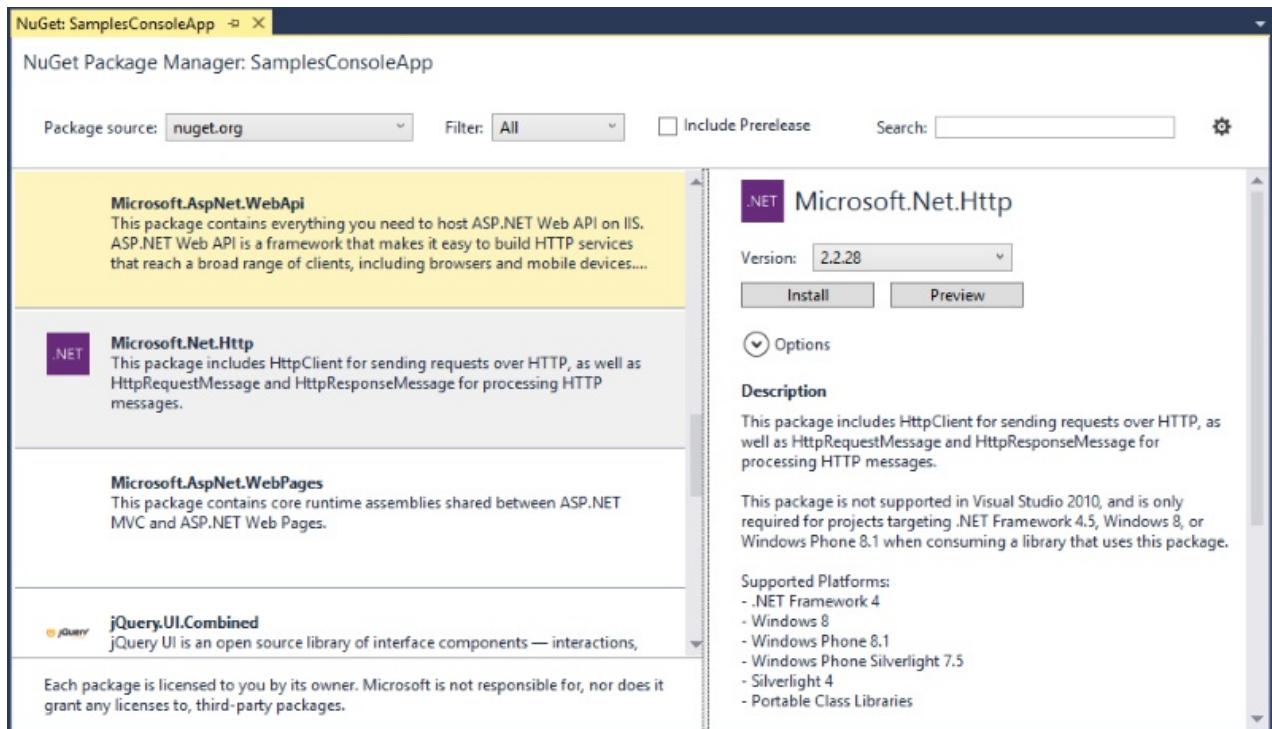
NuGet 3.0 预览版发布于 2014 年 11 月 12 日作为 Visual Studio 2015 预览版发布的一部分。我们发布了 NuGet 3.0 预览版。这是对我们很重要的版本（尽管预览版），和我们高兴地开始获取反馈，我们更改。

Visual Studio 2012 和更高

此 NuGet 3.0 预览版包括在 Visual Studio 2015 预览版中。我们正在努力推出预览版删除 Visual Studio 2012 和 Visual Studio 2013 很快。我们以前共享到我们的意图[停止更新适用于 Visual Studio 2010](#)，和我们未做出难以决定。

全新的 UI

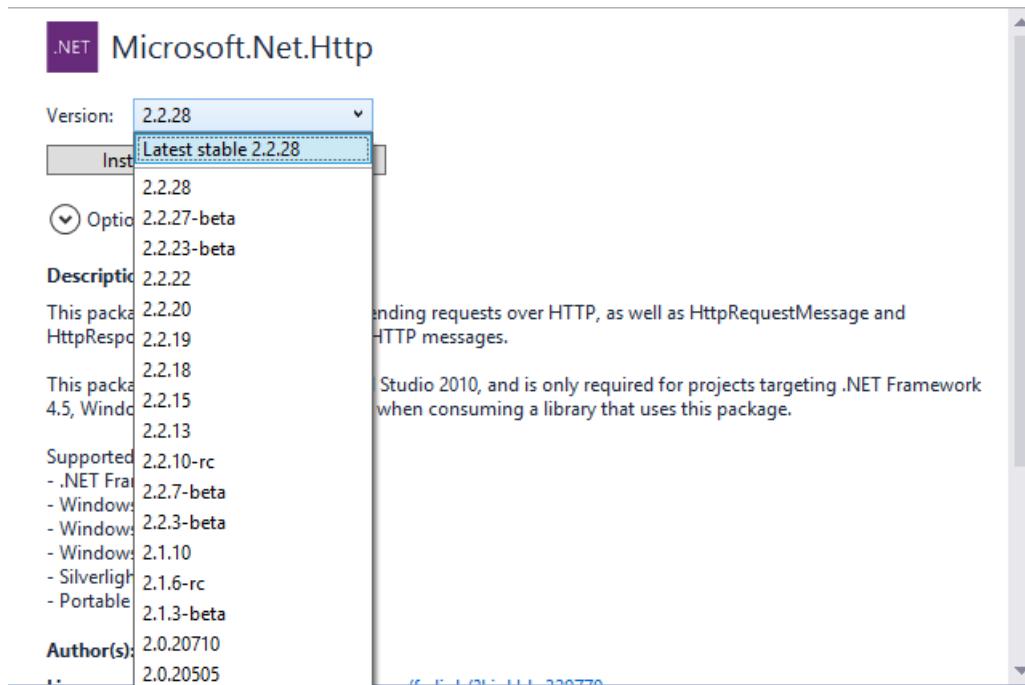
你注意到有关 NuGet 3.0 Preview 的第一件事是我们全新的 UI。它不再是一个模式对话框；它现在是一个完整的 Visual Studio 文档窗口。这允许你在一次打开多个项目（和/或解决方案）的 UI，拉出到另一个监视器窗口中，将其固定，但您会喜欢，等等。



除了可用性差异由于放弃模式对话框中，我们还在新 UI 有很多新功能。

版本选择

可能是最常请求的 UI 功能是允许版本选择的包的安装和更新-此功能现提供。



无论您是安装还是更新包，版本下拉列表中，可查看所有与提升轻松选择列表的顶部为一些值得注意版本可用于包的版本。不再需要使用 PowerShell 控制台来获取不是最新的特定版本。

合并已安装/联机/更新工作流

我们以前的 UI 更新已安装，联机，以及有 3 个选项卡。列出的包是特定于这些工作流，并且特定于工作流以及可用的操作。虽然这看起来逻辑，我们听说过很多人通常会获取掣肘：这种分离。

现在，我们有组合的体验，可以在其中安装、更新或卸载包而不考虑如何达到选择的包。为了帮助与特定工作流，我们现在有一个筛选器下拉列表，可以筛选包可见，但然后可用包的操作保持一致。

Package source: nuget.org Filter: Installed Include Prerelease Search:

Antlr	Newtonsoft.Json
ANother Tool for Language Recognition, is a language tool that provides a framework for constructing recognizers, interpreters, compilers and generators.	Version: 5.0.4 Uninstall Preview
Newtonsoft.Json	Description Json.NET is a popular high-performance JSON framework for .NET
Ninject	Author(s): James Newton-King License: http://json.codeplex.com/license Downloads: 8808784 Date Published: 4/25/2013 4:48 AM -07:00 Project URL: http://james.newtonking.com/projects/json-net.aspx Tags: json

通过使用“已安装”筛选器，然后，您可以轻松地看到哪些功能具有可用的更新，你已安装的包，然后你可以卸载或更改版本选择以查看更新包更改可用的操作。

版本合并

它是常见的同一个包安装到您的解决方案内的多个项目。有时安装到每个项目的版本可以偏离和需要合并中使用的版本。NuGet 3.0 Preview 引入了新功能，可解决此类问题。

可以通过右键单击解决方案并选择为解决方案管理 NuGet 包访问解决方案级包管理窗口。在这里，如果您选择安装到多个项目，但在使用中，不同版本的包的新的“合并”操作将变为可用。在以下屏幕截图，`Newtonsoft.Json` 安装到 `SamplesClassLibrary` 版本 `6.0.4` 且已安装到 `SamplesConsoleApp` 版本 `5.0.4`。

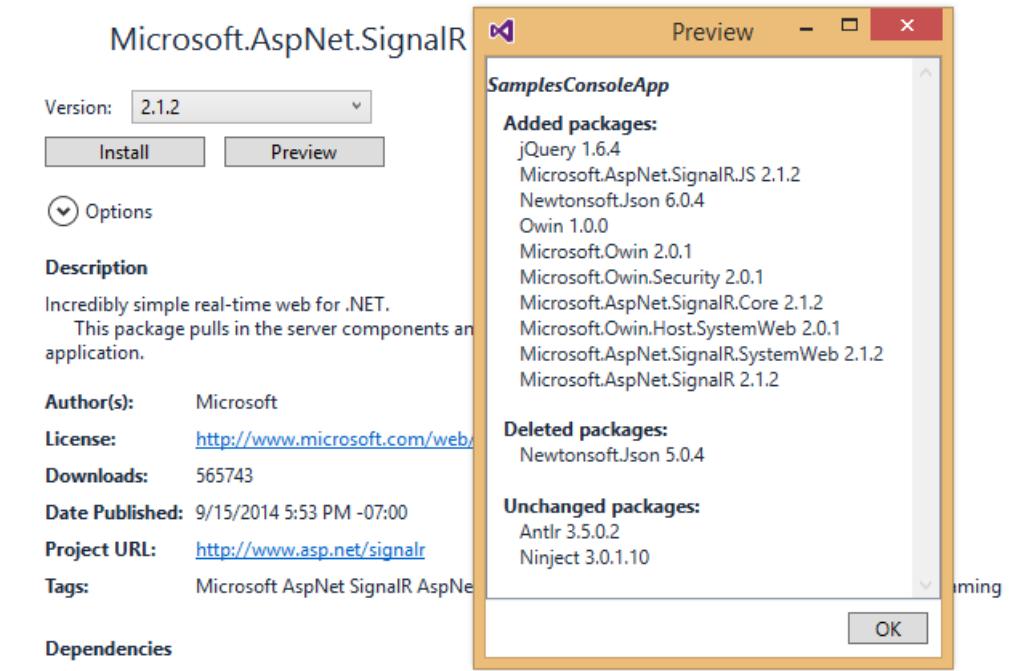
下面是将合并到单个版本上的工作流。

1. 选择 `Newtonsoft.Json` 列表中的包
2. 选择 `Consolidate` 从 `Action` 下拉列表中
3. 使用 `Version` 下拉列表中选择要合并到的版本
4. 选中的复选框应合并到该版本（请注意，已在所选的版本上的项目将灰显）的项目
5. 单击 `Consolidate` 按钮以执行合并

操作预览

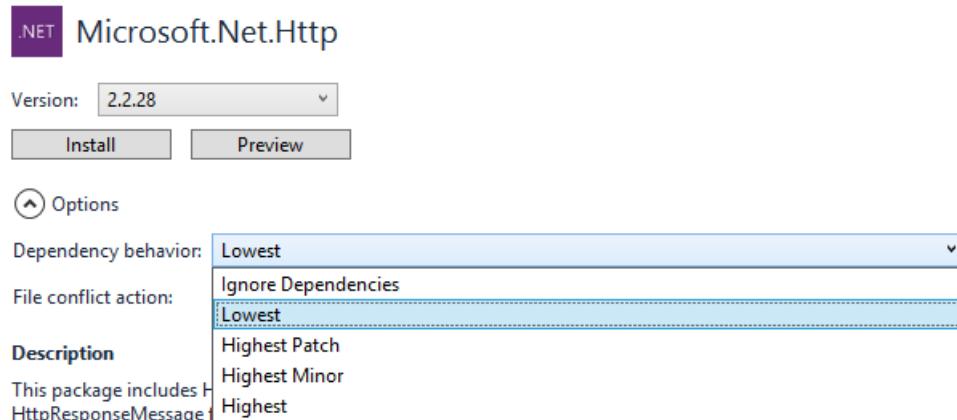
无论哪个操作在执行-安装/更新/卸载-新用户界面现在提供了一种方法，若要预览将会对项目所做的更改。此预览会显示任何新的包将安装包，将被更新，并且包将被卸载，以及在操作期间将保持不变的包。

在下面的示例中，我们可以看到，安装 `Microsoft.AspNet.SignalR` 会导致相当多的更改到项目。



安装选项

使用 PowerShell 控制台，在过去的几个值得注意的安装选项控制。现在，我们已引入 UI 还将这些功能。现在可以控制如何选择版本的依赖项的依赖关系解析行为。



此外可以指定当包中的内容文件与你的项目中的现有文件冲突时要执行的操作。

.NET Microsoft.Net.Http

Version: 2.2.28

Install Preview

Options

Dependency behavior: Lowest

File conflict action: Prompt

Description

This package includes [HttpMessageHandler](#) and [HttpResponseMessage](#) for processing HTTP messages.

This package is not supported in Visual Studio 2010, and is only required for projects targeting .NET Framework 4.5, Windows 8, or Windows Phone 8.1 when consuming a library that uses this package.

Supported Platforms:

- .NET Framework 4
- Windows 8
- Windows Phone 8.1
- Windows Phone Silverlight 7.5
- Silverlight 4

无限滚动

我们用于获取大量的反馈对我们的 UI 具有这两个滚动和列出的包时分页模式。这是相当常见的要滚动到底部的短列表，请单击下一步的页码，再然后向下滚动。使用新的用户界面，我们已实现无限滚动包列表中，以便只需要进行滚动-没有更多的分页。

Package source: nuget.org Filter: All Include Prerelease

Microsoft.Owin.Security.Cookies
Middleware that enables an application to use cookie based authentication, similar to ASP.NET's forms authentication.

Unity
The Unity Application Block (Unity) is a lightweight extensible dependency injection container with support for constructor, property, and method call injection. It facilitates loosely-coupled design. Declarative configuration and r...

WindowsAzure.Storage
This client library enables working with the Microsoft Azure storage services which include the blob and file service for storing binary and text data, the table service for storing structured non-relational data, and the queue service for st...

Loading...

Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.

使其工作，使其快速，使其美观

我们很高兴为您尝试推出此新 UI。在此预览阶段，我们已经遵循良好老话“使其工作，使其快速、使其美观。”在此预览版中，我们已完成大多数的这个第一个目标-其工作原理。我们知道它非常快速还不是，我们知道它非常非常还不是。我们将使用这些目标现在和 RC 版本之间的信任。在此期间，我们期待倾听您的反馈意见可用性的新 UI-工作流、操作、以及如何它感觉以使用新的用户界面。

有几个我们将删除与旧 UI 相比的函数。其中一种是特意的和另一个只是未完成的时间。

搜索“所有”包源

旧 UI 允许您执行针对包源的所有包的搜索。我们在 UI 中删除了该功能，我们将不会将其返回。此功能，用于执行搜索操作针对所有包源，归纳到一起，结果，然后尝试基于排序所选对结果进行排序。

我们发现很难归纳到一起，搜索相关性。可以想象得到执行针对 Google 和必应搜索，以及一起编辑结果？此外，此功能是速度较慢且易于意外使用和我们认为它很少实际上非常有用。由于问题的功能中引入，我们收到数可能永远不会得到了修复的 bug 报告在其上。

全部更新

我们使用旧 ui 中有新的用户界面中尚不具有"更新全部"按钮。我们将于我们的 RC 版本恢复此功能。

新的客户端/服务器 API

除了所有我们新的包管理 UI 中的新功能，我们也一直在 NuGet 的客户端/服务器协议一些实现细节。我们所做的工作是为 NuGet，专为高可用性的关键方案，例如包还原和安装包创建"API v3"。新的 API 基于 REST 和超媒体和我们所选[JSON-LD](#)作为我们的资源格式。

在 NuGet 3.0 预览版位中，可以看到称为"preview.nuget.org"包源下拉列表中的新包源。如果您选择的包源，我们将使用我们的新 API，而是要连接到 nuget.org。我们继续测试、修订和改进的新 API 时，我们已使预览源 UI 中提供。在 NuGet 3.0 RC 中，此新 API 基于 v3 的包源将替换基于 v2"nuget.org"包源。

尽管我们正在将它们放入 API v3 的投资，我们已使所有这些新功能也适用于我们现有的 API v2 协议，这意味着它们将与现有以外还 nuget.org 的包源。

新功能

从现在到 3.0 的 RTM，我们还致力于一些基本之外的新 NuGet 功能，在 UI 中看到的内容。下面是一个重要投资领域的简短列表：

1. 我们正与 Visual Studio 合作，并由 MSBuild 团队以获取[深入到该平台的 NuGet](#)。
2. 我们正在放弃安装时包约定，然后改为在打包时应用这些约定，通过引入一个新"权威"包清单。
3. 我们正在努力 NuGet 基本代码，以使 Visual Studio 中的包管理范围之外的不同域中的客户端和服务器组件可重用的重构。
4. 我们正在研究的"专用依赖项"其中一个包可以指示它概念实现详细信息，其他包上具有依赖关系，这些依赖项不应显示为顶级依赖项。

请继续关注

请密切关注[我们的博客](#)详细进度和 NuGet 3.0 的公告！

NuGet 2.12 发行说明

2018/9/5 • [Edit Online](#)

NuGet 2.12 RTM 发布了 2016 年 6 月 27 日, Visual Studio 2013

此版本中的更新

- 完整 NetStandard 和 NetCoreApp 支持 VS2013。
- 添加到包括/排除 `.nuspec` 依赖项。
- 添加对"no_proxy"若要指定代理的异常支持。
- TFS 相关的修补程序。

可以在 GitHub 上找到的在此版本中的修补程序列表[2.12 里程碑](#)

下载从工具扩展-> 扩展和 Visual Studio 中的更新

NuGet 2.12 RC 发行说明

2018/9/5 • [Edit Online](#)

NuGet 2.12 RC 已于 2016 年 6 月 22 日发布作为对 2.12.0-rc VSIX 的更新, Visual Studio 2013。

此版本中的更新

- 完整 NetStandard 和 NetCoreApp 支持 VS2013。
- 添加到包括/排除 `.nuspec` 依赖项。
- 添加对"no_proxy"若要指定代理的异常支持。
- TFS 相关的修补程序。

可以在 GitHub 上找到的在此版本中的修补程序列表[2.12 里程碑](#)

下载适用于的扩展:

- [Visual Studio 2013](#)

NuGet 2.9 RC 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 2.8.7 发行说明](#) | [NuGet 3.0 预览版发行说明](#)

NuGet 2.9 已于 2015 年 9 月 10 日发布作为到 2.8.7 更新 VSIX 中的进行 Visual Studio 2012 和 2013 年。

此版本中的更新

- 现在跳过处理包, 如果其包含 `.nuspec` 文档的格式不正确- [PR8](#)
- 更正的 Unix/Linux 方案-`\r\n` multipartwebrequest 处理[776](#)
- 更正了与 Visual Studio 2013 Community edition-中的生成事件的集成[1180年](#)

在此版本中的修补程序的完整列表可在 GitHub 上的[2.8.8 里程碑](#)

NuGet 2.8.7 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 2.8.6 发行说明](#) | [NuGet 2.9 RC 发行说明](#)

发布 NuGet 2.8.7 2015 年 7 月 27 日作为修补程序更新到 2.8.6 VSIX 通过特定于受影响的 Powershell 策略实现的 bug 修复。<https://github.com/NuGet/Home/issues/974>

NuGet 2.8.6 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 2.8.5 发行说明](#) | [NuGet 2.8.7 发行说明](#)

发布 NuGet 2.8.6 2015 年 7 月 20 日作为一项次要更新到我们 2.8.5 VSIX 某些目标修复和改进功能以支持具有支持的 Windows 10 UWP 开发模型可能会发送的包。

此版本的 NuGet 包管理器扩展提供仅支持 Visual Studio 2013。

此版本中，在 NuGet 包管理器对话框中必须添加对的支持：

- 引入了 UAP 目标框架名字对象以支持 Windows 10 应用程序开发。
- NuGet 的协议版本 3 终结点
- 为支持[Nuget.Config](#) protocolVersion 属性存储库源。默认值为"2"
- 回退到远程存储库如果所需的包版本在本地缓存中不可用

NuGet 2.8.5 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 2.8.3 发行说明](#) | [NuGet 2.8.6 发行说明](#)

NuGet 2.8.5 已于 2015 年 3 月 30 日发布。它是一项次要更新我们 2.8.3 VSIX 某些目标修补程序。

在此版本中，NuGet 包管理器对话框添加了对支持[DNX 目标框架名字对象](#)。支持这些新框架名字对象包括：

- `core50` -'base' 的目标框架名字对象 (TFM) 与核心 CLR 兼容。
- `dnx452` -使用完整 4.5.2 TFM 特定于基于 DNX 的应用程序的 framework 版本
- `dnx46` -使用完整 4.6 版本的 framework TFM 特定于基于 DNX 的应用程序
- `dnxcore50` -使用核心 5.0 版本的 framework TFM 特定于基于 DNX 的应用程序

该阻止的包安装到 FSharp 项目正确修复一个 bug:

<https://nuget.codeplex.com/workitem/4400>

NuGet 2.8.3 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 2.8.2 发行说明](#) | [NuGet 2.8.5 发行说明](#)

NuGet 2.8.3 已于 2014 年 10 月 17 日发布。它是一项次要更新我们 2.8.1 VSIX 某些目标修补程序。

在此版本中，NuGet 包管理器对话框添加了对支持[ASP.NET vNext](#), [DevExtreme](#)并[BizTalk \(.btproj\)](#)项目类型。它还包括可靠性与启用包还原和保存包管理器选项的方案相关的 bug 修复。

NuGet 2.8.2 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 2.8.1 发行说明](#) | [NuGet 2.8.3 发行说明](#)

NuGet 2.8.2 已于 2014 年 5 月 22, 发布。此版本中, 仅包含 nuget.exe 命令行, NuGet.Server 包与其他 NuGet 包的变化。发布未包含的更新的 Visual Studio 扩展或 WebMatrix 扩展。

值得注意的更新

在 nuget.exe 命令行和 NuGet.Server 包（适用于自承载的 NuGet 源）中介绍了最值得注意的更新。

重要的 nuget.exe Bug 修复

1. [nuget.exe 推送会失败, 并保留重试](#)
2. [nuget.exe 推送不会正确发送基本身份验证凭据](#)
3. [nuget.exe 推送不遵循临时重定向](#)

重要 NuGet.Server 的 Bug 修复

1. [IsAbsoluteLatestVersion NuGet.Server 返回的值不正确](#)

更新包

Nuget.exe 命令行和 NuGet.Server 修补程序附带 NuGet 包更新。没有使用 2.8.2 以及更新其他包。

下面是更新后的包的列表：

1. [NuGet.Core](#)
2. [NuGetCommandLine](#)
3. [NuGet.Server](#)
4. [NuGet.Build](#)
5. [NuGet.VisualStudio](#) (包, 不扩展)

所有更改

出现了 10 个版本中解决的问题。有关完整列表的工作项中已修复 NuGet 2.8.2, 请查看[对于此版本的 NuGet 问题跟踪程序](#)。

NuGet 2.8.1 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 2.8 发行说明](#) | [NuGet 2.8.2 发行说明](#)

NuGet 2.8.1 已于 2014 年 4 月 2 日发布。

在版本中值得注意的功能

对 Windows Phone 8.1 项目的支持

此版本现在支持以下新目标框架名字对象可以用于面向 Windows Phone 8.1 项目记录的：

- WindowsPhone81 / (适用于基于 Silverlight 的 Windows Phone 项目) 和 WP81
- WindowsPhoneApp81 / WPA81 (对于基于 WinRT 的 Windows Phone 应用程序项目)

NuGet WebMatrix 扩展的更新

此版本更新到 WebMatrix 中找到的 NuGet 客户端 [NuGet.Core](#) 2.6.1，并使其如 XDT 转换使用的功能。更重要的是 2.6.1 核心更新，WebMatrix 客户端将安装包含较新版本的 NuGet 包 [.nuspec](#) 架构，其中包括 ASP.NET NuGet 包。

有关 WebMatrix 扩展更新的详细信息，请参阅的那些特定发行说明。

Bug 修复

除了这些功能，此版本的 NuGet 包括其他 bug 修复。出现的版本中解决的 16 总问题。有关完整列表的工作项中已修复 NuGet 2.8.1，请查看[对于此版本的 NuGet 问题跟踪程序](#)。

使用 Visual Studio"14"CTP reshipping

在 Visual Studio"14"CTP 中发布的 2014 年 6 月第三，NuGet 2.8.1 包装盒中。它支持这些功能仍保持在 par 与其他 2.8.1 VSIXes 类似于 Visual Studio 2013。

NuGet 2.8 发行说明

2020/3/19 • [Edit Online](#)

[Nuget 2.7.2 发行说明](#) | [Nuget 2.8.1 发行说明](#)

NuGet 2.8 于2014年1月29日发布。

致谢

1. [Llewellyn Pritchard \(@leppie\)](#)
 - [#3466](#) -打包时，验证依赖项包的 Id。
2. [圣马丁 Balliauw \(@maartenballiauw\)](#)
 - [#2379](#) -删除 persistening 源凭据时 \$metadata 后缀。
3. [Filip De Vos \(@foxtricks\)](#)
 - [#3538](#) -支持为 nuget.exe update 命令指定项目文件。
4. [Juan Gonzalez](#)
 - 不通过-IncludeReferencedProjects 传递[#3536](#)替换标记。
5. [David Poole \(@Sarkie_Dave\)](#)
 - [#3677](#)修复了推送大包时的 OutOfMemoryException 引发。
6. [Wouter Ouwens](#)
 - [#3666](#) -在项目引用其他 CLI/C++ 项目时修复不正确的目标路径。
7. [Adam Ralph \(@adamralph\)](#)
 - [#3639](#) -默认情况下，允许将包安装为开发依赖项
8. [David Fowler \(@davidfowl\)](#)
 - [#3717](#) -删除到最新修补程序版本的隐式升级
9. [Gregory Vandenbrouck](#)
 - NuGet.Server、nuget.exe 镜像命令及其他程序的几个 bug 修复和改进。
 - 这项工作是在数个月内完成的，Gregory 与我们一起使用，以将其集成到2.8 的主节点。

依赖项的修补程序解析

解析包依赖关系时，NuGet 一直实现了一个策略，该策略选择满足包的依赖项的最低主要和次要包版本。但与主要和次要版本不同的是，修补程序版本总是解析为最高版本。尽管此行为是善意的，但它却缺乏了安装具有依赖项的包的确定性。请考虑以下示例：

```
PackageA@1.0.0 -[ >=1.0.0 ]-> PackageB@1.0.0

Developer1 installs PackageA@1.0.0: installed PackageA@1.0.0 and PackageB@1.0.0

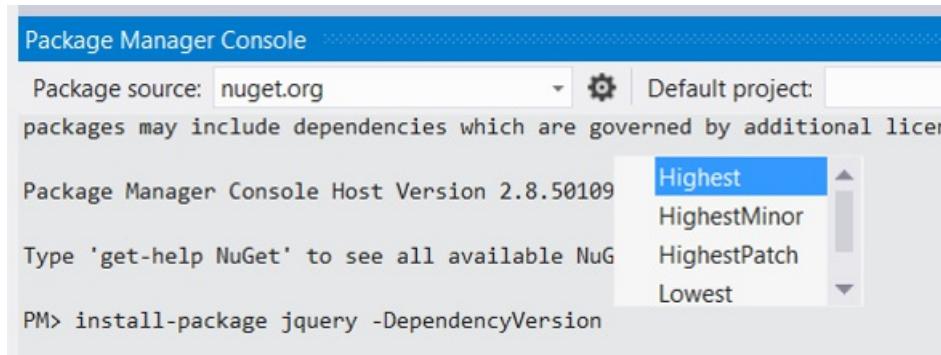
PackageB@1.0.1 is published

Developer2 installs PackageA@1.0.0: installed PackageA@1.0.0 and PackageB@1.0.1
```

在此示例中，即使 Developer1 和 Developer2 安装 PackageA@1.0.0，每个都最终使用不同版本的 PackageB。NuGet 2.8 更改此默认行为，以便修补程序版本的依赖关系解析行为与主版本和次要版本的行为一致。在上面的示例中，PackageB@1.0.0 将因安装 PackageA@1.0.0 而安装，而不考虑较新的修补程序版本。

-DependencyVersion 开关

尽管 NuGet 2.8 更改了用于解决依赖项的默认行为，但它还在包管理器控制台中通过 -DependencyVersion 开关更精确地控制依赖项解析过程。开关允许将依赖项解析为可能的最低版本（默认行为）、可能的最高版本或最高的次要版本或修补程序版本。此开关仅适用于 powershell 命令中的安装包。



DependencyVersion 特性

除了上面详细说明的 -DependencyVersion 开关以外，还允许 NuGet 在 NuGet 文件中设置新属性定义默认值的功能，如果未在调用中指定 -DependencyVersion 开关安装包。任何安装包操作的“NuGet 包管理器”对话框也会遵守此值。若要设置此值，请将以下属性添加到 NuGet.Config 文件：

```
<config>
  <add key="dependencyversion" value="Highest" />
</config>
```

预览具有 -whatif 的 NuGet 操作

某些 NuGet 包可能具有深层依赖项关系图，因此，在安装、卸载或更新操作过程中，它可帮助您首先了解将发生的情况。NuGet 2.8 将标准 PowerShell whatif 交换机添加到安装包、卸载包和更新包命令中，以使将应用该命令的包的整个关闭可视化。例如，在空 ASP.NET Web 应用程序中运行 `install-package Microsoft.AspNet.WebApi -whatif` 将产生以下结果。

```
PM> install-package Microsoft.AspNet.WebApi -whatif
Attempting to resolve dependency 'Microsoft.AspNet.WebApi.WebHost (>= 5.0.0)'.
Attempting to resolve dependency 'Microsoft.AspNet.WebApi.Core (>= 5.0.0)'.
Attempting to resolve dependency 'Microsoft.AspNet.WebApi.Client (>= 5.0.0)'.
Attempting to resolve dependency 'Newtonsoft.Json (>= 4.5.11)'.
Install Newtonsoft.Json 4.5.11
Install Microsoft.AspNet.WebApi.Client 5.0.0
Install Microsoft.AspNet.WebApi.Core 5.0.0
Install Microsoft.AspNet.WebApiWebHost 5.0.0
Install Microsoft.AspNet.WebApi 5.0.0
```

降级包

安装包的预发行版本是为了调查新功能，然后决定回滚到最后一个稳定的版本，这种情况并不常见。在 NuGet 2.8 之前，这是卸载预发行包及其依赖项，然后安装早期版本的多步骤过程。不过，对于 NuGet 2.8，更新包现在会将整个包关闭（例如包的依赖关系树）回滚到以前的版本。

开发依赖项

许多不同类型的功能可作为 NuGet 包提供，包括用于优化开发过程的工具。这些组件虽然可以有助于开发新包，但在以后发布时，不应将其视为新包的依赖项。使用 NuGet 2.8，包可以将 .nuspec 文件中的自身标识为 developmentDependency。安装后，还会将此元数据添加到安装包的项目的 packages.config 文件中。以后在 `nuget.exe pack` 期间为 NuGet 依赖项分析该 packages.config 文件时，它将排除标记为开发依赖项的依赖项。

不同平台的单独包 .config 文件

为多个目标平台开发应用程序时，通常会为各自的每个生成环境使用不同的项目文件。在不同的项目文件中使用不同的 NuGet 包也很常见，因为包具有不同平台的不同级别的支持。NuGet 2.8 通过为不同的特定于平台的项目文件创建不同的 `packages.config` 文件，为此方案提供了改进的支持。

 packages.reactiveUI.config	1/12/2014 10:55 PM	CONFIG File	1 KB
 packages.reactiveUI_Monoandroid.config	1/12/2014 10:55 PM	CONFIG File	1 KB
 packages.ReactiveUI_MonoMac.config	1/12/2014 10:55 PM	CONFIG File	1 KB
 packages.ReactiveUI_Monotouch.config	1/12/2014 10:55 PM	CONFIG File	1 KB
 packages.ReactiveUI_WP8.config	1/12/2014 10:55 PM	CONFIG File	1 KB

回退到本地缓存

尽管 NuGet 包通常是从使用网络连接的远程库使用，但在很多情况下客户端未连接。如果没有网络连接，NuGet 客户端将无法成功安装包-即使这些包已在本地 NuGet 缓存中的客户端计算机上。NuGet 2.8 将自动缓存回退添加到程序包管理器控制台。例如，断开网络适配器和安装 jQuery 时，控制台将显示以下内容：

```
PM> Install-Package jquery
The source at nuget.org [https://www.nuget.org/api/v2/] is unreachable. Falling back to NuGet Local Cache at
C:\Users\me\AppData\Local\NuGet\Cache
Installing 'jQuery 2.0.3'.
Successfully installed 'jQuery 2.0.3'.
Adding 'jQuery 2.0.3' to WebApplication18.
Successfully added 'jQuery 2.0.3' to WebApplication18.
```

缓存回退功能不需要任何特定的命令参数。此外，缓存回退当前仅在包管理器控制台中运行-"包管理器"对话框中当前未使用该行为。

WebMatrix NuGet 客户端更新

除了 NuGet 2.8，还更新了 WebMatrix 的 NuGet 扩展，以包含[nuget 2.5](#)提供的很多主要功能。新功能包括 "全部更新"、"最低 NuGet 版本" 等，并允许覆盖内容文件。

在 WebMatrix 3 中更新 NuGet 包管理器扩展：

1. 打开 WebMatrix 3
2. 单击功能区中的扩展图标
3. 选择 "更新" 选项卡
4. 单击以将 NuGet 包管理器更新到2.5。0
5. 关闭并重新启动 WebMatrix 3

这是 NuGet 团队发布的 NuGet 包管理器扩展的第一个版本。Microsoft 最近向开源 NuGet 项目提供了该代码。以前，NuGet 集成内置于 WebMatrix 中，无法在 WebMatrix 中进行带外更新。现在，我们可以将其与 NuGet 的客户端工具的其余部分进行进一步更新。

Bug 修复

在更新包-重新安装命令中，所做的一个主要 bug 修复是性能改进。

除了这些功能和前面提到的性能修复外，此版本的 NuGet 还包括许多其他 bug 修复。此版本中解决了181的总问题。有关 NuGet 2.8 中已修复工作项的完整列表，请查看[此版本的 NuGet 问题跟踪程序](#)。

NuGet 2.7.2 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 2.7.1 发行说明](#) | [NuGet 2.8 发行说明](#)

NuGet 2.7.2, 它已于 2013 年 11 月 11 日发布。

值得注意的 Bug 修复和功能

许可证文本

一段时间, Microsoft 已包含在 Visual Studio 为 Web 应用程序项目的默认模板的一部分的几个常用的开源库的 NuGet 包。jQuery 可能是库的这种类型的最常见示例。由于与产品一起提供的组件关联的支持协议, 包的脚本文件包含不同的许可证文本中比在同一个包中找到公共 nuget.org 库上的脚本文件。这种文本的差异可以防止由于导致要具有不同的内容哈希值的脚本文件的其他许可证文本块继续包更新 (并因此被视为在项目中修改)。

若要缓解此问题, NuGet 2.7.2, 它允许脚本作者以包括如下所示的特殊标记部分中的许可证文本块。

```
***** NUGET: BEGIN LICENSE TEXT *****
* The following code is licensed under the MIT license
* Additional license information below is informational
* only.
***** NUGET: END LICENSE TEXT *****
```

当更新包的内容文件, 其中包含此块中, NuGet 不会不块的内容, 应考虑使用 NuGet 库上的版本比较和可以因此删除和更新内容文件, 就好像它匹配的原始副本。

由文本"NUGET:: BEGIN 许可证 TEXT"和"NUGET: 结束许可证 TEXT"发生任意位置上开始和结束行标识此块。没有其他格式设置要求存在, 可让在任何类型的文本与语言无关的文件中使用此功能。

添加绑定重定向为非 Framework 程序集

对于属于.NET Framework 的程序集, NuGet 将跳过添加绑定重定向到应用程序的配置文件更新包时。此修补程序地址, 由此绑定重定向不添加对于某些程序集, 即使这些程序集不是 NuGet 2.7 中的回归视为.NET Framework 的一部分。NuGet 2.7.2, 它将还原以前的 NuGet 2.5 和 2.6 行为, 并将添加绑定重定向。

使用安装了 Xamarin 工具安装可移植库

当在计算机上安装 Xamarin 的开发工具时, 它们修改要指定现有的目标框架组合和 Xamarin 框架之间的兼容性的支持的框架配置数据。2.7.2, 它在版本, NuGet 会了解这样的隐式的兼容性规则, 并因此可以轻松面向 Xamarin 平台的开发人员能够在包中这种情况下安装 Xamarin 兼容, 但未显式标记的可移植库元数据本身。

接受的计算机范围的配置设置

使用分层 Nuget.Config 文件时, repositoryPath 密钥不被接受 Nuget.Config 文件接近解决方案根目录。在 Visual Studio 2013 中, NuGet 安装自定义在

%ProgramData%\NuGet\Config\VisualStudio\12.0\Microsoft.VisualStudio.config Nuget.Config 文件以添加"Microsoft 和.NET"包源。因此, 解决在解决方案中使用自定义 repositoryPath 是删除计算机级别 Nuget.Config-这也意味着删除"Microsoft 和.NET"包源。使用分层 Nuget.Config 文件时, NuGet 2.7.2, 它现在遵循 repositoryPath 的优先顺序规则。

所有更改

有关完整列表的工作项中已修复 NuGet 2.7.2, 它, 请查看[对于此版本的 NuGet 问题跟踪程序](#)。

NuGet 2.7.1 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 2.7 发行说明](#) | [NuGet 2.7.2 发行说明](#)

NuGet 2.7.1 已于 2013 年 10 月 7 日发布。这是一项次要更新到某些目标修补，以改善 2.7 的新功能的体验与我们新 2.7 版本。有关工作的列表项中已修复 NuGet 2.7.1，请查看[对于此版本的 NuGet 问题跟踪程序](#)。

完整的 2.7 中的功能集可在[发行说明](#)此处。

NuGet 2.7 发行说明

2019/7/26 • [Edit Online](#)

[Nuget 2.6.1 for WebMatrix 发行说明](#) | [nuget 2.7.1 发行说明](#)

NuGet 2.7 于8月 22 2013 日发布。

致谢

我们想感谢以下外部参与者对 NuGet 2.7 的重大贡献:

1. [Mike Roth\(@mxrss\)](#)

- 列出包时显示许可证 url, 详细信息详细说明。

2. [Adam Ralph\(@adamralph\)](#)

- [#1956](#) -将 developmentDependency 属性添加 `packages.config` 到中, 并将其用于 pack 命令中, 以仅包含运行时包

3. [Rafael Nicoletti\(@tkrafael\)](#)

- 避免在 nuget.exe 包命令中出现重复的属性键。

4. [Ben Phegan\(@BenPhegan\)](#)

- [#2610](#) -将计算机缓存大小增加到200。

5. [Slava Trenigin\(@derigel\)](#)

- [#3217](#) -修复在错误的选项卡中显示更新的 NuGet 对话框
- 修复项目。在 ProjectManager 中, TargetFramework 可以为 null
- [#3248](#)修复了不存在的 packageId 上的 SharedPackageRepository FindPackage/FindPackagesById 将失败

6. [古柯 Boyle\(@kevfromireland\)](#)

- [#3234](#) -启用对 Nomad 项目的支持

7. [Corin Blaikie\(@corinblaikie\)](#)

- 在文件不存在时, [#3252](#)修复推送命令失败, 退出代码为0。

8. [圣马丁 Vesely](#)

- 当项目引用数据库项目时, [#3226](#) -修复 BindingRedirect 命令中的 bug。

9. [Miroslav Bajtos\(@bajtos\)](#)

- [#2891](#) -修复 nuget 的 bug。不正确地对 "exclude" 特性中的通配符进行分析。

10. [Justin Dearing\(@zippy1981\)](#)

- 在还原包时 `NuGet.targets` , #3307 修复 bug 不会将 \$(Platform) 传递给 nuget.exe。

11. [Brian Federici](#)

- [#3294](#) -修复 nuget.exe 包命令中的 bug, 该命令允许添加名称相同但大小写不同的文件, 最后导致 "项已存在" 异常。

12. [Daniel Cazzulino\(@kzu\)](#)

- [#2990](#) -将版本属性添加到 NetPortableProfile 类。

13. [David Simner](#)

- [#3460](#) -修复 bug NullReferenceException if requireApiKey = true, 但标头 X-APIKEY 不存在

14. [Michael Friis\(@friism\)](#)

- [#3278](#) -将 NuGet 版本文件修复为, 以使其在 MonoDevelop 上正常工作

15. [Pranav Krishnamoorthy\(@pranav_km\)](#)

- 通过提高并行度提高还原命令性能

版本中值得注意的功能

默认情况下，包还原（具有隐式同意）

NuGet 2.7 引入了一种新的包还原方法，还克服了重大障碍：包还原许可默认情况下已启用！新方法和隐式许可的组合可大大简化包还原方案。

隐式同意

对于 NuGet 版本 2.0、2.1、2.2、2.5 和 2.6，用户需要在生成过程中显式允许 NuGet 下载缺少的包。如果未显式指定此许可，则已启用包还原的解决方案在用户授予同意之前无法生成。

从 NuGet 2.7 开始，默认启用包还原许可，同时允许用户在需要时使用 NuGet 的“设置”中的复选框明确选择退出包还原。此隐式同意更改会影响以下环境中的 NuGet：

- Visual Studio 2013 预览版
- Visual Studio 2012
- Visual Studio 2010
- nuget.exe 命令行实用程序

Visual Studio 中的自动包还原

从 NuGet 2.7 开始，NuGet 将在 Visual Studio 中生成期间自动下载缺少的包，即使未对该解决方案显式启用包还原也是如此。在生成项目或解决方案时，但在调用 MSBuild 之前，将在 Visual Studio 中执行此自动包还原。这产生了几个重要的好处：

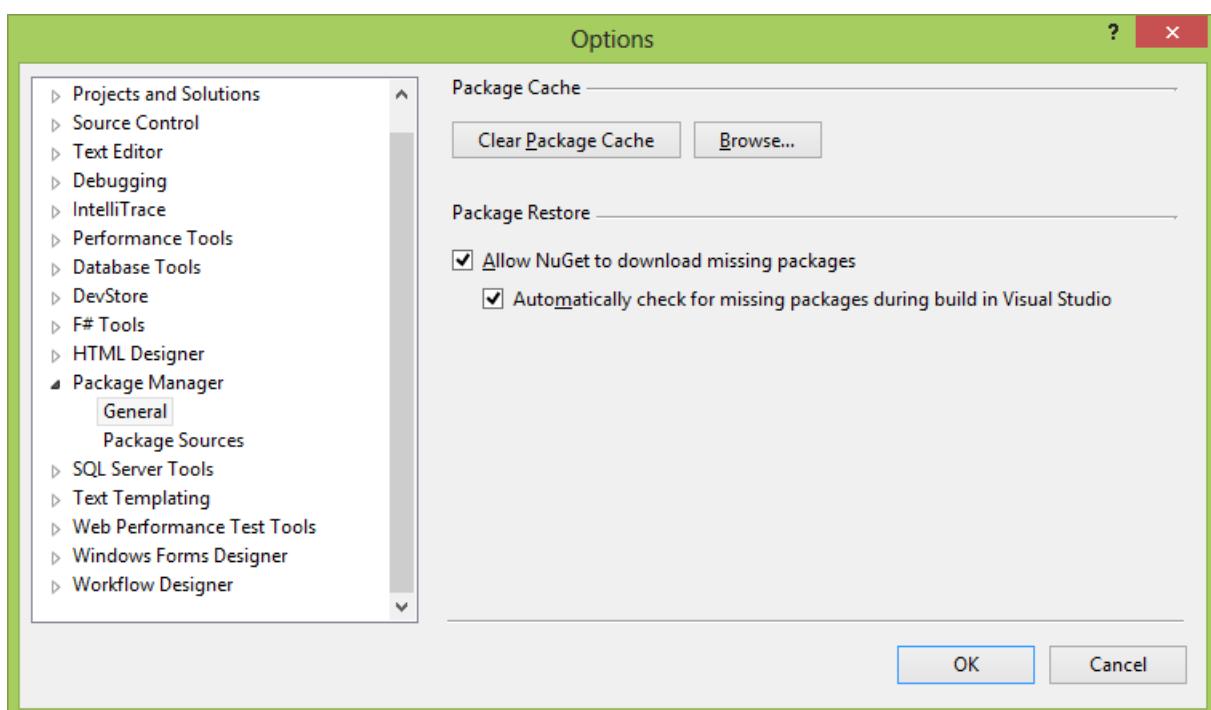
1. 不再需要在解决方案中使用“启用 NuGet 包还原”手势
2. 不需要修改项目，并且 NuGet 不会对项目进行更改，以确保启用包还原
3. 所有 NuGet 包（包括包含属性/目标文件的 MSBuild 导入的文件）将在调用 MSBuild 之前还原，并确保在生成过程中正确识别这些属性/目标

若要在 Visual Studio 中使用自动包还原，只需执行一个（在中）操作：

1. 不签入您 `packages` 的文件夹

有多种方法可以从源代码管理 `packages` 中省略文件夹。有关详细信息，请参阅[包和源代码管理](#)主题。

虽然所有用户都隐式选择自动包还原许可，但你可以轻松地通过 Visual Studio 中的包管理器设置进行选择。



从命令行简化包还原

NuGet 2.7 为 nuget.exe 引入了一项新功能: `nuget.exe restore`

这一新的 "还原" 命令可让你使用单个命令轻松还原解决方案的所有包, 方法是接受解决方案文件或文件夹作为参数。此外, 当当前文件夹中只有一个解决方案时, 该参数是隐含的。这意味着, 从包含单个解决方案文件 (MySolution) 的文件夹中执行以下所有操作:

1. `nuget.exe restore MySolution`
2. `nuget.exe` 还原。
3. `nuget.exe` 还原

Restore 命令将打开解决方案文件并查找解决方案中的所有项目。然后, 它将查找 `packages.config` 每个项目的文件, 并还原找到的所有包。它还将还原在 `.nuget\packages.config` 文件中找到的解决方案级别包。有关新还原命令的详细信息, 请参阅[命令行参考](#)。

新的包还原工作流

我们对包还原进行了这些更改, 因为它引入了一个新的工作流。如果要从源代码管理中省略包, 只需提交 `packages` 文件夹。打开并生成解决方案的 Visual Studio 用户将看到包自动还原。对于命令行生成, 只需在 `nuget.exe restore` 调用 `msbuild` 之前调用。不再需要记得在解决方案上使用 "启用 NuGet 包还原" 手势, 而不再需要修改项目来更改生成。这还为包含 MSBuild 导入的包带来了更好的体验, 特别是通过 NuGet 的最新功能添加的导入, 可自动导入 `\build` 文件夹中的[属性/目标文件](#)。

除了我们亲自完成的工作外, 我们还与一些重要合作伙伴合作, 将此新方法舍入。对于其中的任何一种情况, 我们都还没有具体的时间表, 但每个合作伙伴都非常兴奋。

- Team Foundation Service-他们正在努力将对 `nuget.exe restore` 的调用集成到默认生成方案。
- Microsoft Azure 网站-他们正在努力, 使你能够将项目推送到 Azure, 并 `nuget.exe restore` 在构建你的网站之前调用它。
- TeamCity-他们正在为 TeamCity 3.x 更新 NuGet 安装程序插件
- AppHarbor-它们的作用是使你能够将存储库推送到 AppHarbor, `nuget.exe restore` 并在构建解决方案之前调用它。

对于上述每个合作伙伴, 他们将使用自己的 nuget.exe 副本, 而不需要在解决方案中携带 nuget.exe。

已知问题

使用初始2.7 版本时, nuget.exe 还原有两个已知问题, 但在9/6/2013 上已修复了这些问题, 并更新了[NuGet 命令行包](#)。CodePlex 上的[NuGet 2.7 下载页面](#)上也提供了此更新。运行 `nuget.exe update -self` 将更新到最新版本。

固定的是:

1. [使用 SLN 文件时, 新包还原不适用于 Mono](#)
2. [新包还原不适用于 Wix 项目](#)

新的包还原工作流也存在一个已知问题, 即[对于解决方案文件夹下的项目, 自动包还原不起作用](#)。此问题已在 NuGet 2.7.1 中修复。

项目重定目标和升级生成错误/警告

在重定目标或升级项目后多次, 你会发现某些 NuGet 包不能正常工作。遗憾的是, 没有任何迹象, 就如何解决此情况没有任何帮助。使用 NuGet 2.7, 我们现在使用一些 Visual Studio 事件来识别你的项目重定向或升级的方式, 这种方式会影响你安装的 NuGet 包。

如果检测到你的任何包受重定目标或升级的影响, 我们将生成即时生成错误以通知你。除了立即生成错误外, 还会在 `requireReinstallation="true"` `packages.config` 文件中为受重定目标影响的所有包保留一个标志, 并且 Visual Studio 中的每个后续生成将引发这些包的生成警告。

尽管 NuGet 不能执行自动操作来重新安装受影响的包, 但我们希望这一指示和警告将引导你在需要重新安装包时了解相关信息。我们还致力于在[包重新安装指南文档](#)中, 将这些错误消息定向到。

NuGet 配置默认值

许多公司都在内部使用 NuGet，但我们已在很大时间向开发人员提供了使用内部包源（而不是 nuget.org）的指导。

NuGet 2.7 引入了“配置默认值”功能，该功能允许为以下项指定计算机范围的默认值：

1. 启用的包源
2. 已注册，但已禁用包源
3. 默认的 nuget.exe 推送源

现在，其中的每个可在位于 `%ProgramData%\NuGet\NuGetDefaults.Config` 的文件中进行配置。如果此配置文件指定了包源，则将不会自动注册默认的 nuget.org 包源，而是改 `NuGetDefaults.Config` 为注册中的包源。

尽管不需要使用此功能，但我们希望公司使用组策略 `NuGetDefaults.Config` 部署文件。

请注意，此功能永远不会导致包源从开发人员的 NuGet 设置中删除。这意味着，如果开发人员已使用 NuGet，并因此注册了 nuget.org 包源，则不会在创建 `NuGetDefaults.Config` 文件后将其删除。

有关此功能的详细信息，请参阅[NuGet 配置默认值](#)。

重命名默认包源

NuGet 始终注册一个指向 nuget.org 的默认包源，名为“NuGet 官方包源”。该名称是详细的，它也不指定实际指向的位置。若要解决这两个问题，我们已将此包源重命名为 UI 中的“nuget.org”。包源的 URL 也已更改为包含“www”。前缀。使用 NuGet 2.7 后，现有的“NuGet 官方包源”将自动更新为“nuget.org”作为其名称，将 [“https://www.nuget.org/api/v2/”](https://www.nuget.org/api/v2/) 作为其 URL。

性能改进

我们在 2.7 中进行了一些性能改进，从而降低了内存占用，减少了磁盘使用量，并加快了包安装。我们还为基于 OData 的源进行了更智能化的查询，这将减少总体负载。

新的扩展性 API

我们向扩展性服务添加了一些新的 API，以弥补以前版本中缺少的功能的差距。

IVsPackageInstallerServices

```
```cs
// Checks if a NuGet package with the specified Id and version is installed in the specified project.
bool IsPackageInstalledEx(Project project, string id, string versionString);

// Get the list of NuGet packages installed in the specified project.
IEnumerable<IVsPackageMetadata> GetInstalledPackages(Project project);
```
```

IVsPackageInstaller

```
```cs
// Installs one or more packages that exist on disk in a folder defined in the registry.
void InstallPackagesFromRegistryRepository(string keyName, bool isPreUnzipped, bool skipAssemblyReferences,
Project project, IDictionary<string, string> packageVersions);

// Installs one or more packages that are embedded in a Visual Studio Extension Package.
void InstallPackagesFromVSExtensionRepository(string extensionId, bool isPreUnzipped, bool
skipAssemblyReferences, Project project, IDictionary<string, string> packageVersions);
```
```

仅限开发的依赖项

此功能由 [Adam Ralph](#) 提供，它允许包作者声明仅在开发时使用的依赖项，而无需包依赖项。通过将 `developmentDependency="true"` 属性添加到中 `packages.config` 的包，`nuget.exe pack` 将不再包含该包作为依赖项。

删除了对 Visual Studio 2010 Express for Windows Phone 的支持

2.7 中的新包还原模型是通过与主 NuGet VSPackage 不同的新 VSPackage 实现的。由于技术问题, 在 *Visual studio 2010 Express for Windows Phone* SKU 中, 此新的 VSPackage 无法正常工作, 因为我们与其他受支持的 Visual studio sku 共享相同的代码库。因此, 从 NuGet 2.7 开始, 我们将从已发布的扩展中删除对 *Visual Studio 2010 Express for Windows Phone* 的支持。支持 *Visual studio 2010 Express For Web* 仍包含在发布到 Visual Studio 扩展库的主扩展中。

由于我们不确定开发人员在该版本的 Visual Studio 中仍使用 NuGet 的多少开发人员, 我们将专门为这些用户发布单独的 Visual Studio 扩展, 并将其发布到 CodePlex (而不是 Visual Studio 扩展库). 我们不打算继续维护该扩展, 但如果这样会影响你, 请在 CodePlex 上存档问题, 告诉我们。

若要下载 NuGet 包管理器 (对于 Visual Studio 2010 Express for Windows Phone), 请访问[nuget 2.7 下载页](#)。

Bug 修复

除了这些功能以外, 此版本的 NuGet 还包括许多其他 bug 修复。此版本中解决了97的总问题。有关 NuGet 2.7 中已修复的工作项的完整列表, 请查看[此版本的 NuGet 问题跟踪程序](#)。

NuGet 2.6.1 for WebMatrix 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 2.6 发行说明](#) | [NuGet 2.7 发行说明](#)

NuGet 团队于 2014 年 3 月 26 日发布的 WebMatrix 的更新的 NuGet 包管理器扩展。可以从安装此更新[WebMatrix 扩展库](#)使用以下步骤：

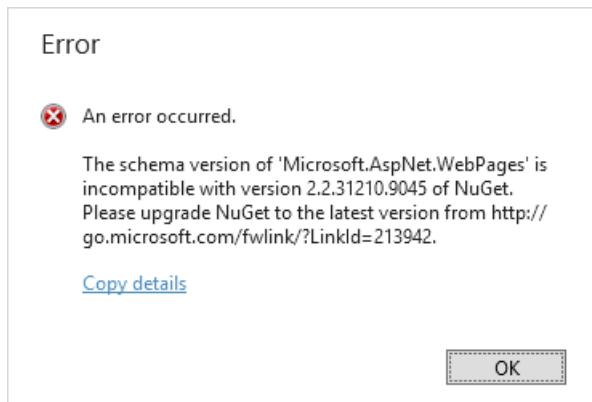
1. 打开 WebMatrix 3
2. 单击主页功能区中的扩展图标
3. 选择更新选项卡
4. 单击此项可更新到 2.6.1 的 NuGet 包管理器
5. 关闭并重新启动 WebMatrix 3

值得注意的更改

最大的问题用户此扩展更新解决两个已经遇到过 WebMatrix 中的使用 NuGet 包。第一种是 NuGet 架构版本错误和第二个是指向零字节 DLL 错误 `bin` 文件夹。

NuGet 架构版本错误

WebMatrix 3 已自发布以来，到适用于 NuGet 包需要新的架构版本的 NuGet 已引入新功能。在尝试管理 NuGet 程序包在您的网站，这些新的包可能会导致你在 WebMatrix 中看到的错误。



此最新版本提供了与最新的 NuGet 包，防止发生此错误的兼容性。现在可以在 WebMatrix 中安装新版本的包包括 Microsoft.AspNet.WebPages。这些包的一些已使用的 NuGet 功能，如[XDT 配置转换](#)，这不受支持的在 WebMatrix 中到目前为止。

Bin 文件夹中的零字节 DLL

某些用户报告的后安装 NuGet 中的包中包含 DLL 复制到 bin 的 WebMatrix DLL 显示 `bin` 0 字节的文件的文件夹。这将在运行时中断该应用程序。

[此问题现已修复。](#)

其他最新的改进

NuGet 包管理器 2.8 for Visual Studio 发布时，我们还为 WebMatrix 发布 NuGet 包管理器 2.5.0。虽然这中所述[NuGet 2.8 发行说明](#)，我们没有提到特定的新功能引入了该更新。

全部更新

现在，您可以更新所有在一个步骤中的 web 站点的包！在 WebMatrix 中打开 NuGet 扩展时，所有程序包的列表

上看到库、安装，和具有可用的更新。以前，必须单独更新每个包，但现在没有显示在更新选项卡的有用“全部更新”按钮。

The screenshot shows the NuGet Gallery interface. The left sidebar has tabs for 'All (9052)', 'Updates (8)', and 'Installed (11)'. The 'Updates (8)' tab is selected. The main area displays eight package updates:

- jQuery**: jQuery 2.1.0 Installed Version 1.8.2. Description: jQuery is a new kind of JavaScript Library. jQuery is a fast and concise JavaScript Library that simplifies HTML document tra...
- jQuery Mobile**: jquery.mobile 1.4.0 Installed Version 1.2.0. Description: A unified user interface system across all popular mobile device platforms, built on the rock-solid jQuery and jQuery UI foundation.
- jQuery Validation**: jQuery.Validation 1.11.1 Installed Version 1.10.0. Description: This jQuery plugin makes simple clientside form validation trivial, while offering lots of option for customization. That makes a good choice if you're building something ne...
- Microsoft ASP.NET Razor 2**: Microsoft.AspNet.Razor 2.0.30506.0 Installed Version 2.0.20715.0. Description: This package contains the runtime assemblies for ASP.NET Web Pages. ASP.NET Web Pages and the new Razor syntax provide a fast, terse, clean and lightweight way to co...
- Microsoft ASP.NET Web Pages 2**: Microsoft.AspNet.WebPages 2.0.30506.0 Installed Version 2.0.20710.0. Description: This package contains core runtime assemblies shared between ASP.NET MVC and ASP.NET Web Pages.

At the bottom, there are filters for 'WebMatrix Package' and 'Stable Only', and buttons for 'Update All', 'Update', 'Uninstall', and 'Close'.

覆盖现有文件

安装时包含您的网站中已存在的文件的包，NuGet 已始终只以无提示方式忽略这些文件（保留不动现有文件）。这可能导致包已安装，或当事实上没有正确更新的印象。NuGet 现在将提示输入覆盖文件。

A screenshot of a 'File Conflict Resolution' dialog box. It shows a code editor with the file 'jquery.validate.unobtrusive.js' containing the following text:

```
jquery.validate.unobtrusive.js
1 // This file already exists and therefore NuGet should prompt me to overwrite it
```

The dialog message says: 'File 'Scripts\jquery.validate.unobtrusive.js' already exists in project 'C:\Users\jeffhand\Documents\My Web Sites\Photo Gallery1'. Do you want to overwrite it?' Below the message are four buttons: 'Yes', 'Yes to All', 'No', and 'No to All'.

NuGet 2.6 发行说明

2020/1/8 • [Edit Online](#)

[NuGet 2.5 发行说明](#) | [NuGet 2.6.1 For WebMatrix 发行说明](#)

NuGet 2.6 于2013年6月26日发布。

版本中值得注意的功能

支持 Visual Studio 2013

NuGet 2.6 是为 Visual Studio 2013 提供支持的第一个版本。和 Visual Studio 2012 一样，每个版本的 Visual Studio 中都包含 NuGet 包管理器扩展。

若要为 Visual Studio 2013 提供尽可能好的支持，同时仍然支持 Visual Studio 2010 和 Visual Studio 2012，并尽可能缩小扩展大小，我们将为 Visual Studio 2013 生成一个单独的扩展，同时原始扩展将继续面向 Visual Studio 2010 和 2012。

从 NuGet 2.6 开始，我们将发布两个扩展，如下所示：

1. [NuGet 包管理器](#) (适用于 Visual Studio 2010 和 2012)
2. [Visual Studio 2013 的 NuGet 包管理器](#)

使用此拆分时，[nuget.org](#) 主页的 "安装 nuget" 按钮将转到 "安装 nuget" 页，您可以在其中找到有关安装不同 nuget 客户端的详细信息。

XDT Web.config 转换支持

NuGet 客户端最常请求的功能之一就是使用在 Visual Studio 生成配置转换中使用的 XDT 转换引擎，支持更强大的 XML 转换。

2013 年 4 月，我们为 XDT 提供了两个有关 NuGet 支持的重要公告。第一种是，XDT 库本身[作为 NuGet 包发布，并在 CodePlex 上开放](#)。此步骤启用了其他开源软件（包括 NuGet 客户端）可自由使用的 XDT 引擎。第二个公告是在 NuGet 客户端中支持使用 XDT 引擎进行转换的计划。NuGet 2.6 包括此集成。

工作原理

若要利用 NuGet 的 XDT 支持，该结构与[当前配置转换功能](#)的外观类似。转换文件被添加到包的内容文件夹中。但是，虽然配置转换使用单个文件进行安装和卸载，但 XDT 转换使用以下文件实现对这两个过程的精细控制：

- Web.config.install.xdt
- Web.config.uninstall.xdt

此外，NuGet 使用文件后缀来确定要为转换运行的引擎，以便使用现有 web.config 的包将继续工作。XDT 转换还可以应用于任何 XML 文件（而不仅仅是 web.config），因此可以将此应用于项目中的其他应用程序。

可以通过 XDT 执行的操作

XDT 的最大优势之一是它的[简单但功能强大的语法](#)，用于处理 XML DOM 的结构。XDT 可以通过多种方式（从简单的属性名称匹配到完整的 XPath 支持）为匹配元素提供控件，而不是只是将一个固定文档结构覆盖到另一结构。一旦找到匹配的元素或一组元素，XDT 就提供了一组丰富的用于操作元素的函数，无论是指添加、更新或删除属性、将新元素放在特定位置还是替换或删除整个元素及其子元素。

计算机范围的配置

NuGet 的优点之一是，它将其他大的可执行文件或库分解为一组模块化组件，这些组件可进行集成，并且最重要的维护和版本控制。不过，这种情况的一个副作用是，产品或产品系列的传统思路可能会产生更多的碎片。NuGet 的自定义包源功能提供了一种组织包的方式；但是，自定义包源本身无法发现。

NuGet 2.6 通过在路径% ProgramData%/NuGet/Config. 下搜索文件夹层次结构来扩展用于配置 NuGet 的逻辑。产品安装程序可在此文件夹下添加自定义 NuGet 配置文件，以便为其产品注册自定义包源。此外，文件夹结构还支持 IDE 的产品、版本甚至 SKU 的语义。这些目录中的设置将按以下顺序应用，并具有 "last in wins" 优先级策略。

1. %ProgramData%\NuGet\Config*.config
2. %ProgramData%\NuGet\Config{IDE}*.config
3. %ProgramData%\NuGet\Config{IDE}{Version}*.config
4. %ProgramData%\NuGet\Config{IDE}{Version}{SKU}*.config

在此列表中，{IDE} 占位符特定于运行 NuGet 的 IDE，因此，在 Visual Studio 中，它将为 "VisualStudio"。IDE（例如 "11.0" 和 "WDEnvironment"、"VWDExpress" 和 "Pro"）提供了 {Version} 和 {SKU} 占位符。然后，文件夹可以包含许多不同的 *.config 文件。因此，ACME 组件公司可以作为其产品安装程序的一部分，添加自定义包源，仅在 Visual Studio 2012 的专业版和旗舰版（通过创建以下文件路径）中可见：

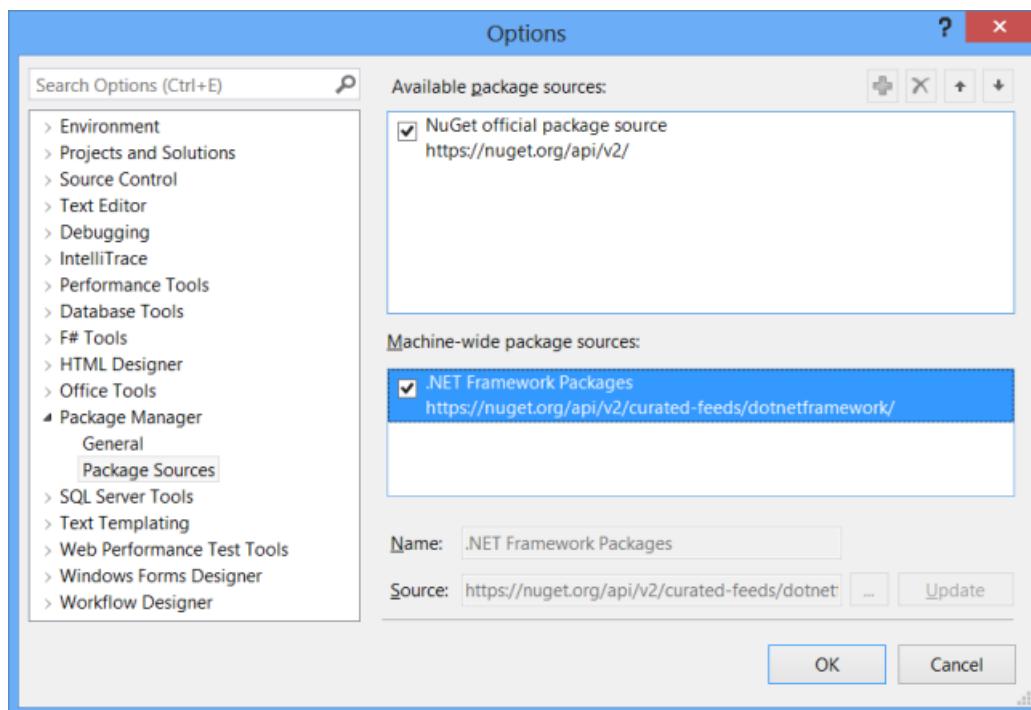
```
%ProgramData%\NuGet\Config\VisualStudio\11.0\Pro\acme.config
```

虽然文件夹结构使软件安装程序等程序可以简单地将计算机范围的包源添加到 NuGet 的配置，但还更新了 NuGet 配置对话框，以允许将包源注册为用户特定的（例如，在% AppData%/NuGet/NuGet.Config 中注册）或计算机范围。

此功能的使用 Visual Studio 2013，其中的文件安装位置如下：

```
%ProgramData%\NuGet\Config\VisualStudio\12.0\Microsoft.VisualStudio.config
```

在此文件中，将配置名为 ".NET Framework 包" 的新包源。



Contextualizing 搜索

随着 NuGet 库所提供的包数量不断增长，在 NuGet 优先级列表的顶部，提高搜索范围。NuGet 的计划功能之一是上下文搜索，也就是说，NuGet 将使用有关所使用的 Visual Studio 版本和 SKU 的信息，以及要构建为确定潜在搜索相关性的条件的项目类型后果。

从 NuGet 2.6 开始，每次安装包时，安装的上下文都将记录为安装操作数据的一部分。搜索还会发送相同的上下文信息，使 NuGet 库能够按上下文安装趋势增加搜索结果。对 NuGet 库的未来更新将实现此上下文相关的相关性提升。

跟踪直接安装与依赖项安装

包作者依赖于 NuGet 库上提供的包统计信息。作者要求的一个重要的数据点是直接包安装和依赖项安装之间的差异。到现在为止，NuGet 客户端不会在安装操作周围发送任何上下文，无论开发人员是否直接安装了包，或者是否

已安装以满足依赖项的需要。从 NuGet 2.6 开始，现在将为安装操作发送数据。NuGet 库中的包统计信息会将这些数据作为单独的安装操作公开，并带有 "-依赖项" 后缀。

- 安装
- 安装-依赖项
- 更新
- 更新-依赖项
- 重新安装
- 重新安装-依赖项

除了不同的操作名称外，还会为安装记录依赖程序包 id。对 NuGet 库的将来更新将在报表中公开这些数据，以使包创作者能够充分了解开发人员安装包的方式。

Bug 修复

NuGet 2.6 还包括多个 bug 修复。有关 NuGet 2.6 中已修复的工作项的完整列表，请查看[此版本的 NuGet 问题跟踪程序](#)。

NuGet 2.5 发行说明

2020/3/19 • [Edit Online](#)

[Nuget 2.2.1 发行说明](#) | [Nuget 2.6 发行说明](#)

NuGet 2.5 于2013年4月25日发布。此版本非常大，我们认为我们会跳过版本2.3 和2.4！迄今为止，这是我们为 NuGet 提供的最大版本，其中包含160多个[工作项](#)。

致谢

我们想感谢以下外部参与者对 NuGet 2.5 的重大贡献：

1. [Daniel Plaisted \(@dsplaisted\)](#)
 - #2847 -将 MonoAndroid、Monotouch.dialog 和 MonoMac 添加到已知目标框架标识符的列表。
2. [Andres Aragoneses \(@knocte\)](#)
 - #2865 -修复区分大小写的操作系统的 `NuGet.targets` 的拼写
3. [David Fowler \(@davidfowl\)](#)
 - 使解决方案在 Mono 上构建。
4. [Andrew Theken \(@atheken\)](#)
 - 修复 Mono 上失败的单元测试。
5. [Marc-olivier Dagenais \(@OlilsCool\)](#)
 - #2920 -nuget.exe 包命令未将属性传播到 MSBuild
6. [Miroslav Bajtos \(@bajtos\)](#)
 - 用于保留格式的#1511修改的 XML 处理代码。
7. [Adam Ralph \(@adamralph\)](#)
 - 已将识别的字词添加到自定义字典，以允许成功生成 .cmd。
8. [Bruno Roggeri](#)
 - 在本地化和中运行时修复单元测试
9. [Gareth Evans](#)
 - 已从 PackageService 中提取接口
10. [Maxime Brugidou \(@brugidou\)](#)
 - #936 -在打包时处理项目依赖项
11. [Xavier Decoster \(@XavierDecoster\)](#)
 - #2991, 在 config 文件中存储包源凭据时，#3164支持明文密码
12. [James Manning \(@manningj\)](#)
 - #3190, #3191修复获取包帮助说明

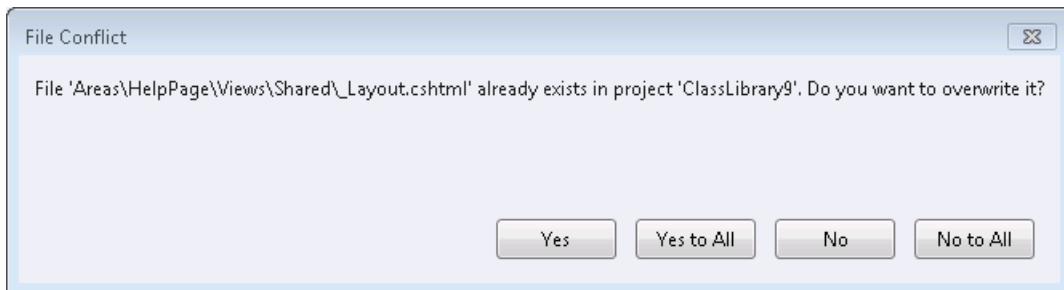
此外，我们还会感谢以下人员查找在最终版本之前批准和修复的 NuGet 2.5 Beta/RC bug：

1. [Tony 墙\(@CodeChief\)](#)
 - #3200 -MSTest 中断了最新 NuGet 2.4 和2.5 版本

版本中值得注意的功能

允许用户覆盖已存在的内容文件

所有时间最常请求的功能之一是能够覆盖包含在 NuGet 包中的已存在于磁盘上的内容文件。从 NuGet 2.5 开始，会识别这些冲突，并提示您覆盖这些文件，而以前这些文件始终被跳过。



"nuget.exe update" 和 "安装包" 现在都有一个新选项 "-FileConflictAction" 用于为命令行方案设置某些默认值。

当目标项目中已存在来自包的文件时，设置默认操作。设置为 "覆盖" 以始终覆盖文件。设置为 "Ignore" 可跳过文件。如果未指定，则会提示输入每个冲突的文件。

MSBuild 目标和属性文件的自动导入

已在 NuGet 包的顶层创建了一个新的传统文件夹。作为对等的 `\lib`、`\content` 和 `\tools`，你现在可以在包中包含 `\build` 文件夹。在此文件夹下，可以将两个具有固定名称的文件、`{packageid}.targets` 或 `{packageid}.props`。这两个文件可以直接位于 `build` 下，也可以在特定于框架的文件夹下直接进行，就像其他文件夹一样。选择最佳匹配框架文件夹的规则与这些文件夹的规则完全相同。

当 NuGet 安装带有 `\build` 文件的包时，它会将项目文件中的 MSBuild `<Import>` 元素添加到 `.targets` 并 `.props` 文件中。`.props` 文件会添加到顶部，而 `.targets` 文件会添加到底部。

使用 `<References/>` 元素为每个平台指定不同的引用

在 2.5 之前，用户只能在 `.nuspec` 文件中指定要为所有框架添加的引用文件。现在，在 2.5 中提供了这项新功能，用户可以为每个受支持的平台创作 `<reference/>` 元素，例如：

```
<references>
  <group targetFramework="net45">
    <reference file="a.dll" />
  </group>
  <group targetFramework="netcore45">
    <reference file="b.dll" />
  </group>
  <group>
    <reference file="c.dll" />
  </group>
</references>
```

下面是 NuGet 如何根据 `.nuspec` 文件添加对项目的引用的流程：

1. 查找适用于目标框架的 `lib` 文件夹，并从该文件夹获取程序集列表
2. 分别查找适用于目标框架的引用组，并从该组获取程序集列表。未指定目标框架的引用组是回退组。
3. 查找两个列表的交集，并将其用作要添加的引用

如果需要在多个 `lib` 文件夹中包含重复的程序集，则此新功能将允许包作者使用 "引用" 功能将程序集的子集应用于不同框架。

注意：你目前必须使用 `nuget.exe` 包才能使用此功能；NuGet 包资源管理器尚不支持。

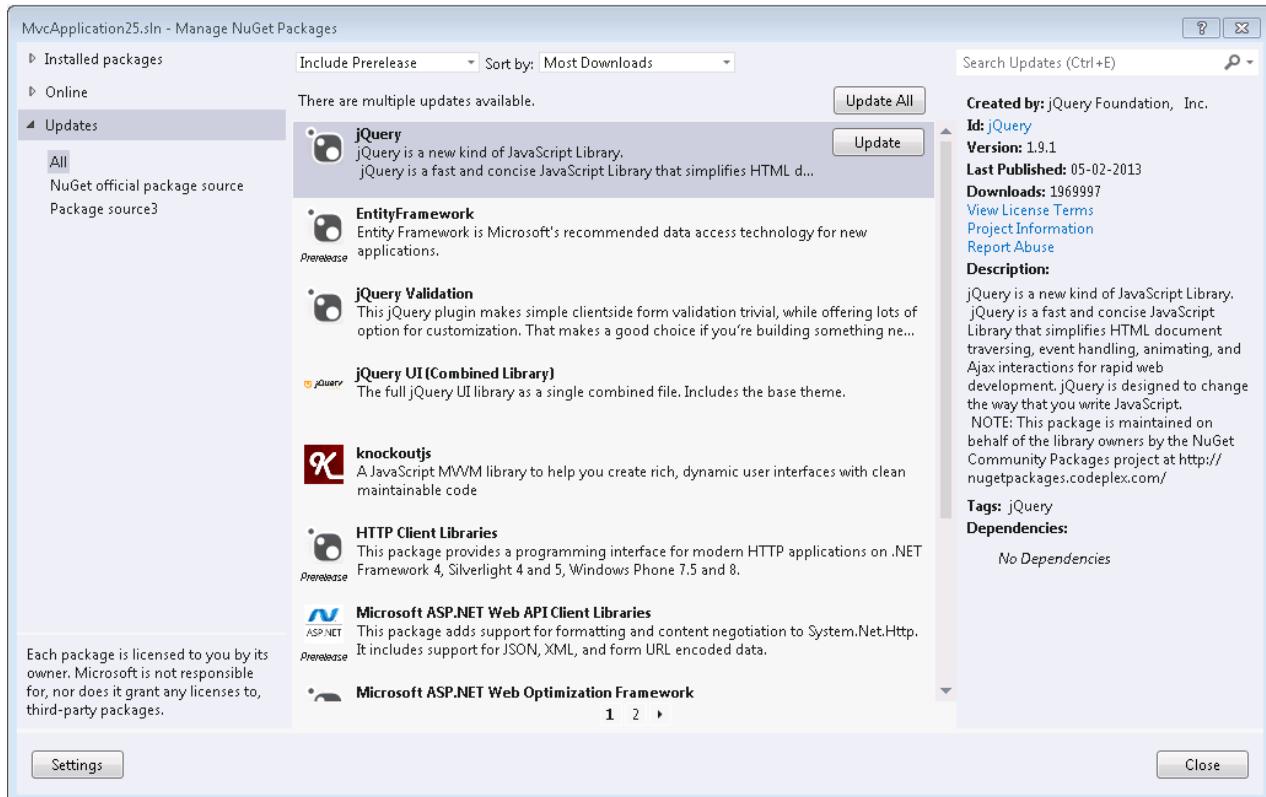
"全部更新" 按钮允许同时更新所有包

许多人都知道，"更新包" PowerShell cmdlet 可用于更新所有包；现在，还可以通过 UI 轻松完成此操作。

要尝试此功能，请执行以下操作：

1. 新建 ASP.NET MVC 应用程序
2. 启动 "管理 NuGet 包" 对话框
3. 选择 "更新"

4. 单击 "全部更新" 按钮



改进了对 nuget 包的项目引用支持

现在, nuget.exe 包命令处理具有以下规则的引用项目 :

- 如果引用的项目具有相应的 `.nuspec` 文件(例如, 在与 `proj1.csproj` 相同的文件夹中有一个名为 `proj1.nuspec` 的文件, 则会使用从 `.nuspec` 文件中读取的 id 和版本将此项目作为依赖项添加到包中。)
- 否则, 被引用项目的文件捆绑到包中。然后, 将以递归方式使用 `sames` 规则处理此项目引用的项目。
- 添加所有 DLL、`.pdb` 和 `.exe` 文件。
- 添加了其他所有内容文件。
- 所有依赖项都将合并。

这允许将引用的项目视为依赖项(如果存在 `.nuspec` 文件), 否则它将成为包的一部分。

更多详细信息: <http://nuget.codeplex.com/workitem/936>

向包中添加 "最小 NuGet 版本" 属性

名为 "minClientVersion" 的新元数据属性现在可以指示使用包所需的最小 NuGet 客户端版本。

此功能有助于包作者指定仅在特定版本的 NuGet 后使用包。由于新的 `.nuspec` 功能将在 NuGet 2.5 后添加, 因此包将能够声明最低版本的 NuGet。

```
<metadata minClientVersion="2.6">
```

如果用户安装了 NuGet 2.5, 并且包被标识为需要2.6, 则将向用户提供视觉提示, 指示将无法安装包。然后, 用户将指导更新其 NuGet 的版本。

这会改进现有的体验, 其中包开始安装, 但随后会失败, 指示标识了无法识别的架构版本。

包安装过程中不再需要再更新依赖项

在 NuGet 2.5 之前, 当安装了依赖于项目中已安装的包的包时, 该依赖项将作为新安装的一部分进行更新, 即使现有版本满足了依赖关系也是如此。

从 NuGet 2.5 开始, 如果依赖项版本已满足, 则不会在其他包安装期间更新依赖项。

方案:

1. 源存储库包含版本 1.0.0 和 1.0.2 的包 B。它还包含在 B (> = 1.0.0) 上依赖的包 A。
2. 假定当前项目已安装了包 B 1.0.0 版。现在, 你想要安装包 A。

在 NuGet 2.2 和更早版本中:

- 安装包 A 时, NuGet 将自动更新 B 到 1.0.2, 即使现有版本 1.0.0 已经满足依赖项版本约束, 该约束 > = 1.0.0。

在 NuGet 2.5 和更高版本中:

- NuGet 将不再更新 B, 因为它检测到现有版本 1.0.0 满足依赖项版本约束。

有关此更改的更多背景信息, 请阅读详细的[工作项](#)和相关的[讨论线索](#)。

nuget.exe 输出 http 请求以及详细详细信息

如果你正在排查 nuget.exe 问题, 或者只是想要在操作过程中发出 HTTP 请求, "-详细信息" 开关现在会输出发出的所有 HTTP 请求。

```
C:\>nuget install jquery -verbosity detailed
GET https://nuget.org/api/v2/FindPackagesById()?id='jquery'
GET https://nuget.org/api/v2/package/jquery/1.9.1
Installing 'jQuery 1.9.1'.
Successfully installed 'jQuery 1.9.1'.
```

nuget.exe 推送现在支持 UNC 和文件夹源

在 NuGet 2.5 之前, 如果尝试根据 UNC 路径或本地文件夹将 "nuget.exe push" 运行到包源, 则推送会失败。使用最近添加的层次结构配置功能, nuget.exe 需要将 UNC/文件夹源或基于 HTTP 的 NuGet 库作为目标。

从 NuGet 2.5 开始, 如果 nuget.exe 标识 UNC/文件夹源, 将对源执行文件复制。

现在可以使用以下命令:

```
nuget push -source \\mycompany\repo\ mypackage.1.0.0.nupkg
```

nuget.exe 支持显式指定的配置文件

访问配置(除 "spec" 和 "pack" 之外的所有文件)的 nuget.exe 命令现在支持新的 "-Read-configfile" 选项, 该选项强制使用特定的配置文件来替代%Appdata%\nuget\nuget.config 上的默认配置文件。

示例:

```
nuget sources add -name test -source http://test -ConfigFile C:\test\.nuget\Nuget.Config
```

对本机项目的支持

使用 NuGet 2.5, NuGet 工具现在可用于 Visual Studio 中的本机项目。我们预计大多数本机包都将使用[CoApp 项目](#)创建的工具来利用上述 MSBuild 导入功能。有关详细信息, 请参阅 coapp.org 网站上[有关该工具的详细信息](#)。

当包安装到本机项目中时, 将为包引入 "本机" 的目标框架名称, 以包括 \build、\content 和 \tools 中的文件。'lib' 文件夹不用于本机项目。

NuGet 2.2.1 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 2.2 发行说明](#) | [NuGet 2.5 发行说明](#)

NuGet 2.2.1 已于 2013 年 2 月 15 日发布。在 VS 扩展版本编号是 2.2.40116.9051。

本地化刷新

当 NuGet 作为 Visual Studio 2012 的一部分提供时，它已完全本地化为英语 + 13 种其他语言。从那时起，NuGet 2.1 和 2.2 已装运，但尚未刷新本地化。NuGet 2.2.1 版本刷新本地化。

NuGet 的 UI 和 PowerShell 控制台已本地化为以下语言版本：

1. 中文(简体)
2. 和 SharePoint 2010 显示的“中文(繁体)”
3. 捷克语
4. 英语
5. 法语
6. 德语
7. 意大利语
8. 日语
9. 朝鲜语
10. 波兰语
11. 葡萄牙语(巴西)
12. 俄语
13. 西班牙语
14. 土耳其语

Visual Studio 模板支持多个预安装的包存储库

如果生成 Visual Studio 模板，则可以使用 NuGet 预安装包作为模板的一部分。到目前为止，此功能有限制的所有包需要来自同一源。使用 NuGet 2.2.1 不过，您可以从多个存储库（在模板中，VSIX 或在注册表中定义的磁盘上的文件夹）安装的包。

此功能的主要方案是自定义 ASP.NET 项目模板。内置的 ASP.NET 模板使用预安装的包，提取程序包从本地磁盘。现在，您可以创建使用 ASP.NET 所安装的现有包的自定义 ASP.NET 项目模板，但将额外的 NuGet 包添加到你的模板。

Bug 修复

NuGet 2.2.1 包含几个目标的 bug 修补程序。有关工作的列表项中已修复 NuGet 2.2.1，请查看[对于此版本的 NuGet 问题跟踪程序](#)。

已知问题

当扩展 ASP.NET 项目模板，所有预安装的包存储库必须使用相同的值为 `isPreunzipped` 属性。

NuGet 2.2 发行说明

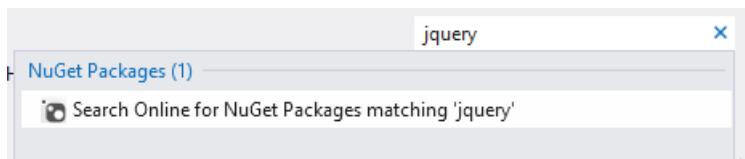
2018/9/5 • [Edit Online](#)

[NuGet 2.1 发行说明](#) | [NuGet 2.2.1 发行说明](#)

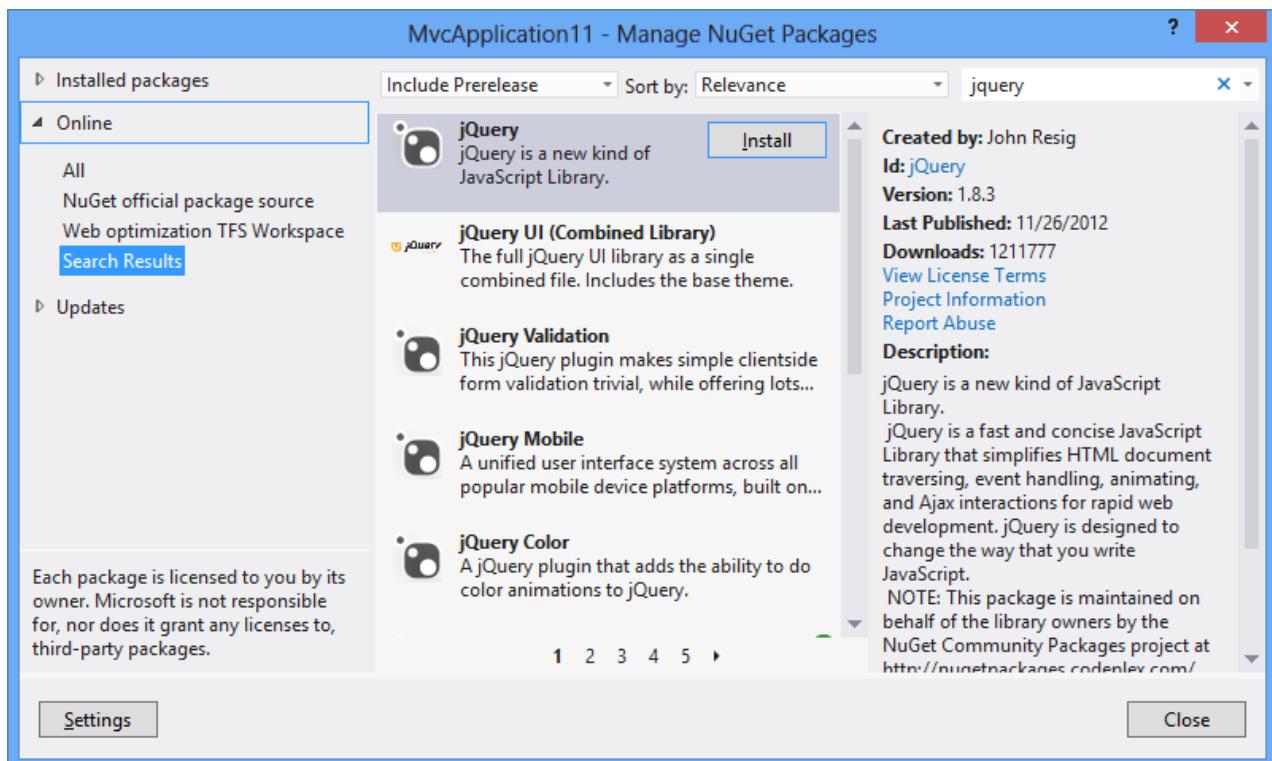
NuGet 2.2 已于 2012 年 12 月 12 日发布。

Visual Studio 快速启动

在 Visual Studio 2012 中已添加的新功能之一是[快速启动对话框](#)。NuGet 2.2 扩展了此对话框中，使其能够与在快速启动栏中输入的搜索条件初始化包管理器对话框。例如，现在在快速启动中输入 jquery 将包括在结果中搜索匹配 jquery NuGet 包的选项。



选择此选项将启动术语 jquery 的标准 NuGet 包管理器搜索的体验。



指定包内容的整个文件夹

NuGet 2.2 现在允许你指定的整个文件夹中 `<file>` 元素的 `.nuspec` 文件来包含所有该文件夹的内容。例如，以下将导致所有脚本添加到 `content\scripts` 文件夹中，当包安装到项目的包的脚本文件夹中。

```
<file src="scripts\" target="content\scripts"/>
```

更新 6/24/16：安装包时，“内容”文件夹中的空文件夹将被忽略。

已知问题

使用包管理器控制台时，软件包安装失败的 F# 项目

当尝试将 NuGet 程序包安装到 F# 项目使用包管理器控制台时，将引发 InvalidOperationException。我们正在积极与 F# 团队来解决此问题，但在此期间，解决方法是将 NuGet 包安装到 F# 项目的 NuGet 包管理器对话框而不是在控制台通过。[CodePlex 上提供了详细信息。](#)

Bug 修复

NuGet 2.2 包括许多 bug 修复。有关工作的完整列表项中已修复 NuGet 2.2，请查看[对于此版本的 NuGet 问题跟踪程序。](#)

NuGet 2.1 发行说明

2018/9/5 • [Edit Online](#)

[NuGet 2.0 发行说明](#) | [NuGet 2.2 发行说明](#)

NuGet 2.1 已于 2012 年 10 月 4 日发布。

分层 Nuget.Config

NuGet 2.1 为您提供更灵活地控制通过以递归方式查找的文件夹结构向上遍历的 NuGet 设置 `NuGet.Config` 文件，然后生成中找到的所有文件组的配置。作为示例，请考虑其中一个团队具有的其他内部的依赖项的 CI 生成的内部包存储库的方案。单个项目的文件夹结构可能类似以下形式：

```
C:\\\nmyteam\\\nmyteam\\solution1\\\nmyteam\\solution1\\project1
```

此外，如果解决方案启用了包还原，也将存在以下文件夹：

```
C:\\myteam\\solution1\\.nuget
```

为了使团队的内部包存储库可用的所有项目的团队处理，同时不使它可为每个项目的计算机上，我们可以创建一个新的 `Nuget.Config` 文件，并将其放在 `c:\\myteam` 文件夹中。无法为指定每个项目的 `packages` 文件夹。

```
<configuration>\n<packageSources>\n<add key="Official project team source" value="http://teamserver/api/v2/" />\n</packageSources>\n<disabledPackageSources />\n<activePackageSource>\n<add key="Official project team source" value="http://teamserver/api/v2/" />\n</activePackageSource>\n</configuration>
```

现在，我们可以看到源被添加，如下所示的 `c:\\myteam` 下任何文件夹中运行 `nuget.exe` 源命令：

```
1. NuGet official package source [Enabled]\nhttps://nuget.org/api/v2/\n2. Official project team source [Enabled]\nhttp://teamserver/api/v2/
```

`NuGet.Config` 按以下顺序搜索了文件：

1. `.nuget\\Nuget.Config`
2. 递归遍历到根项目文件夹中
3. 全局 `Nuget.Config` (`%appdata%\\NuGet\\Nuget.Config`)

配置是不是应用于反转顺序，也就是说，取决于上述顺序，全局 `Nuget.Config` 将是首先应用后，跟已发现的 `Nuget.Config` 文件从根项目文件夹，然后是通过 `.nuget\\Nuget.Config`。这一点特别重要，如果您使用的 `<clear/>` 要从配置中删除一组项的元素。

指定包文件夹位置

在过去, NuGet 具有托管解决方案的根文件夹下找到已知的包文件夹中的解决方案包。对于具有许多不同的解决方案的已安装的 NuGet 包的开发团队, 这可能导致在文件系统上的多个不同位置中安装在同一包中。

NuGet 2.1 提供了更精细地控制通过包文件夹的位置 `repositoryPath` 中的元素 `NuGet.Config` 文件。生成的 `NuGet.Config` 的层次结构支持以上一个示例, 假设我们想要拥有 `C:\myteam\` 共享相同的包文件夹下的所有项目。若要完成此操作, 只需添加到以下一项 `c:\myteam\Nuget.Config`。

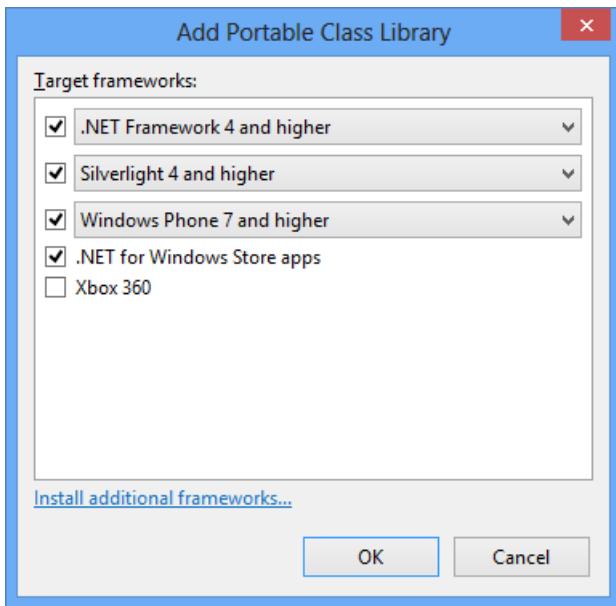
```
<configuration>
  <config>
    <add key="repositoryPath" value="C:\myteam\teampackages" />
  </config>
  ...
</configuration>
```

在此示例中, 共享 `Nuget.Config` 文件指定用于创建下方 `C:\myteam`, 而不考虑深度的每个项目的共享的包文件夹。请注意, 是否解决方案根目录下有一个现有的包文件夹, 您需要 NuGet 会将包放置在新位置之前将其删除。

对可移植库的支持

可移植库是首次随.NET 4 中, 可用于生成可以无需修改即可在不同的 Microsoft 平台, 从.net Framework 对 Windows Phone 和甚至 Xbox silverlight 版本上的程序集的功能360 (尽管在此期间, NuGet 不支持 Xbox 可移植库目标)。通过扩展打包约定 framework 版本和配置文件, NuGet 2.1 现在支持可移植库, 使您创建具有复合框架和配置文件目标的包 `lib` 文件夹。

作为示例, 请考虑以下可移植库的可用目标平台。



构建库后, 该命令 `nuget.exe pack MyPortableProject.csproj` 运行时, 新的可移植库包文件夹结构可以通过检查生成的 NuGet 包的内容发现。

```
▲ lib
  ▲ portable-win+net40+sl40+wp
    PortableClassLibrary1.dll
```

正如您所看到的可移植库文件夹名称约定遵循的模式可移植的 {framework 1} + {framework n} 的框架标识符遵循现有 **framework 名称和版本约定**。用于 Windows Phone 的框架标识符中找到了一个异常的名称和版本的约定。此名字对象应使用的框架名称 wp (wp7、wp71 或 wp8)。使用 silverlight-wp7, 例如, 将导致错误。

在安装时创建此文件夹结构中的包, NuGet 现在可以将其框架和配置文件规则应用于多个目标, 文件夹名称中指

定。NuGet 的匹配规则是，"更多特定"目标将优先于"不太特定"的原则。这意味着，面向特定平台的名字对象将始终是首选通过可移植的如果它们都与一个项目兼容。此外，如果多个可移植目标兼容使用项目，NuGet 将首选支持的平台设置为"最靠近"项目引用包。

面向 Windows 8 和 Windows Phone 8 项目

除了添加支持面向可移植库项目，NuGet 2.1 所提供的 Windows 8 应用商店和 Windows Phone 8 项目，以及为 Windows 应用商店和 Windows Phone 项目将一些新常规名字对象的新框架名字对象更轻松地跨各自的平台的未来版本管理。

对于 Windows 8 应用商店应用程序中，标识符如下所示：

NUGET 2.0	NUGET 2.1
winRT45, .NETCore45	Windows, Windows8, win win8

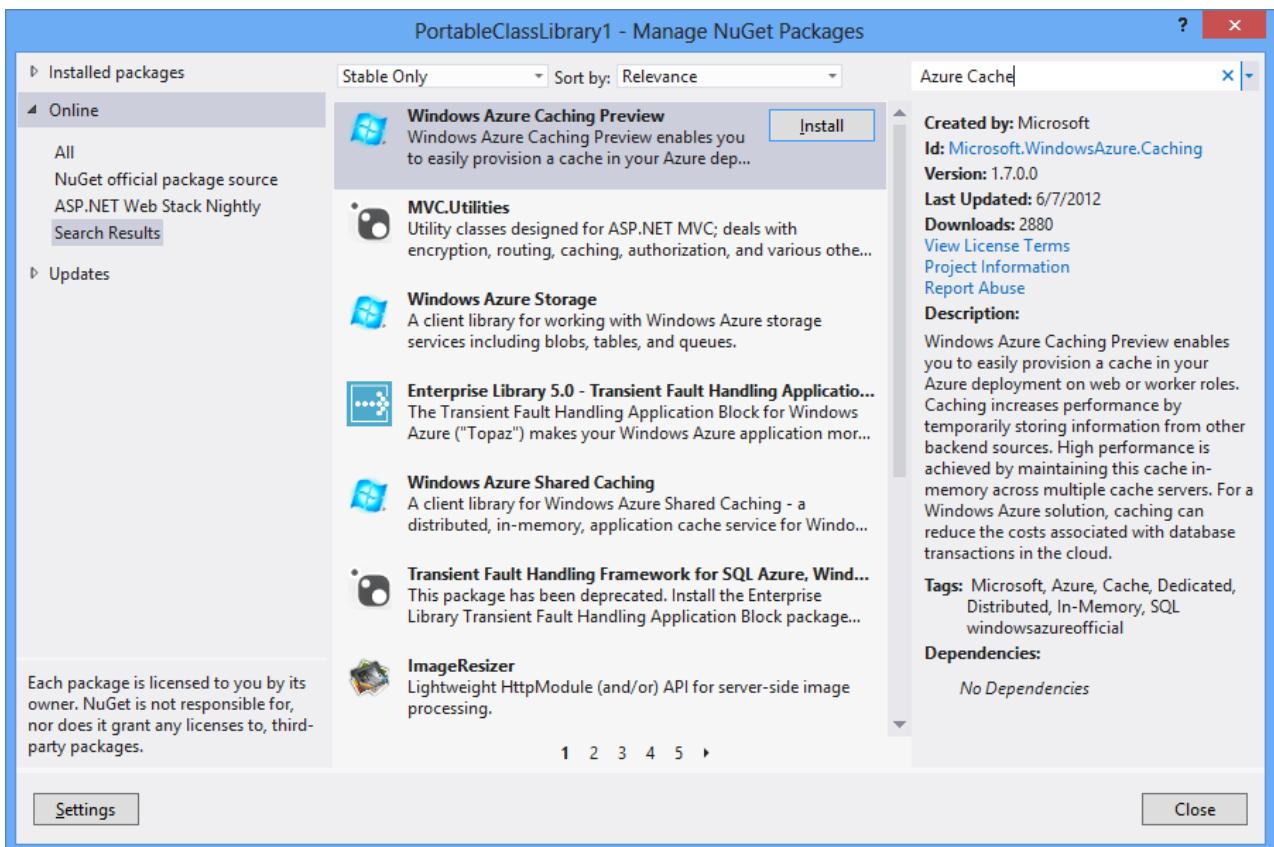
对于 Windows Phone 项目，标识符如下所示：

PHONE	NUGET 2.0	NUGET 2.1
Windows Phone 7	silverlight3 wp	wp wp7, WindowsPhone WindowsPhone7
Windows Phone 7.5 (Mango)	silverlight4 wp71	wp71 WindowsPhone71
Windows Phone 8	(不支持)	wp8 WindowsPhone8

在所有上述更改，将继续得到的 NuGet 2.1 完全支持旧的框架名称。展望未来，新的名称应将更稳定，将来版本的各自的平台。新名称将*不*是支持在 2.1 之前的 NuGet 版本，但是，因此相应地规划时切换。

在包管理器对话框改进的搜索

在过去的几个迭代，引入了更改到 NuGet 库，大大提高速度和包搜索相关性。但是，这些改进的限制为 nuget.org Web 站点。NuGet 2.1 可通过 NuGet 包管理器对话框提供改进的搜索体验。例如，假设您想要找到 Windows Azure Caching 预览包。此包的合理的搜索查询可能是"Azure 缓存"。在以前版本的包管理器对话框中，所需的包将不甚至会列出结果的第一页。但是，在 NuGet 2.1 所需的包现在显示在搜索结果顶部。



强制执行包更新

在 NuGet 2.1 之前 NuGet 将跳过更新包时出现不高的版本号。这就产生了某些情况下-尤其是对于生成或 CI 方案，团队没有不需要每次生成数字将包版本递增的冲突。所需的行为是强制进行更新，而不考虑。NuGet 2.1 可解决这的重新安装标志。例如，尝试更新不具有最新包版本的包时 NuGet 的早期版本将导致以下：

```
PM> Update-Package Moq
No updates available for 'Moq' in project 'MySolution.MyConsole'.
```

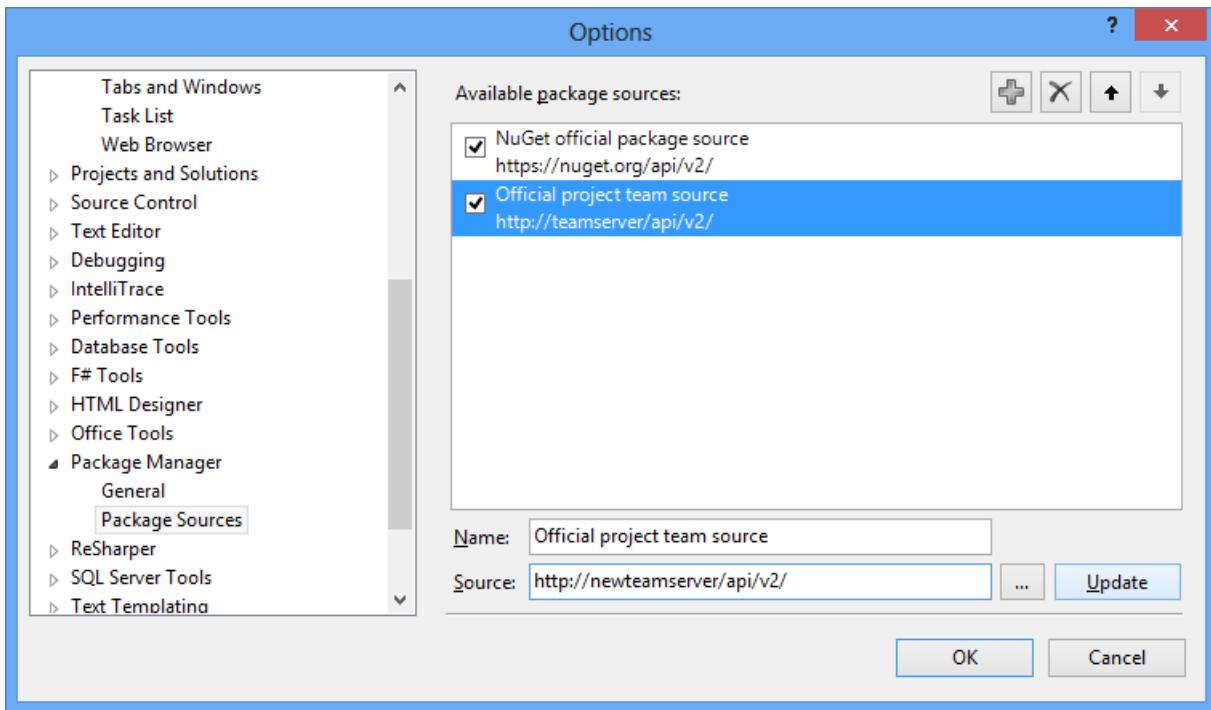
通过重新安装标志，包将更新无论是否还有一个较新版本。

```
PM> Update-Package Moq -Reinstall
Successfully removed 'Moq 4.0.10827' from MySolution.MyConsole.
Successfully uninstalled 'Moq 4.0.10827'.
Successfully installed 'Moq 4.0.10827'.
Successfully added 'Moq 4.0.10827' to MySolution.MyConsole.
```

重新安装标志证明有益的另一种方案是 framework 重定目标。更改（例如，从.NET 4 中为.NET 4.5），一个项目的目
标框架时更新包的重新安装可以更新到正确的程序集的所有项目中安装的 NuGet 包的引用。

编辑 Visual Studio 中的包源

在以前版本的 NuGet，正在更新从 Visual Studio 选项对话框所需删除并重新添加包源中的包源。NuGet 2.1 通过支持更新作为配置用户界面的第一类函数来改进此工作流。



Bug 修复

NuGet 2.1 包括许多 bug 修复。有关工作的完整列表项中已修复 NuGet 2.0, 请查看[对于此版本的 NuGet 问题跟踪程序](#)。

NuGet 2.0 发行说明

2020/1/8 • [Edit Online](#)

[NuGet 1.8 发行说明](#) | [NuGet 2.1 发行说明](#)

NuGet 2.0 于2012年6月19日发布。

已知安装问题

如果你运行的是 VS 2010 SP1，但如果你安装了较旧的版本，你可能会遇到安装错误。

解决方法是仅卸载 NuGet，然后从 VS 扩展库安装它。有关详细信息，请参阅 <https://support.microsoft.com/kb/2581019>，或直接跳到 [VS 修补程序](#)。

注意：如果 Visual Studio 不允许你卸载扩展（“卸载”按钮已禁用），则可能需要使用“以管理员身份运行”重启 Visual Studio。

包还原许可现在处于活动状态

如本[文章在包还原许可](#)中所述，NuGet 2.0 现在要求授予许可以使包还原联机并下载包。请确保已通过“程序包管理器配置”对话框或 `EnableNuGetPackageRestore` 环境变量提供同意。

按目标框架对依赖项进行分组

从版本2.0 开始，根据目标项目的框架配置文件，包依赖项可能会有所不同。这是使用更新的 `.nuspec` 架构来完成的。`<dependencies>` 元素现在可以包含一组 `<group>` 元素。每个组包含零个或多个 `<dependency>` 元素和 `targetFramework` 属性。如果目标框架与目标项目框架配置文件兼容，则组中的所有依赖项都将一起安装。例如：

```
<dependencies>
  <group>
    <dependency id="RouteMagic" version="1.1.0" />
  </group>

  <group targetFramework="net40">
    <dependency id="jQuery" />
    <dependency id="WebActivator" />
  </group>

  <group targetFramework="sl30">
  </group>
</dependencies>
```

请注意，一个组可以包含零个依赖项。在上述示例中，如果将包安装到面向 Silverlight 3.0 或更高版本的项目中，则不会安装任何依赖项。如果将包安装到面向 .NET 4.0 或更高版本的项目中，将安装两个依赖项（jQuery 和 WebActivator）。如果将包安装到以这两个框架的早期版本为目标的项目或任何其他框架，将安装 RouteMagic 1.1.0。组之间没有继承。如果项目的目标框架与组的 `targetFramework` 属性匹配，则只安装该组中的依赖关系。

包可以使用以下两种格式中的一种来指定包依赖项：`<dependency>` 元素或组的简单列表的旧格式。如果使用 `<group>` 格式，则不能将包安装在版本早于2.0 的 NuGet 中。

请注意，不允许混合使用这两种格式。例如，下面的代码段无效并将被 NuGet 拒绝。

```
<dependencies>
  <dependency id="jQuery" />
  <dependency id="WebActivator" />

  <group>
    <dependency id="RouteMagic" version="1.1.0" />
  </group>
</dependencies>
```

按目标框架对内容文件和 PowerShell 脚本进行分组

除程序集引用外，还可以按目标框架对内容文件和 PowerShell 脚本进行分组。现在，`lib` 文件夹中发现的相同文件夹结构可用于指定目标框架，现在可以通过相同的方式应用到 `content` 和 `tools` 文件夹。例如：

```
\content
  \net11
    \MyContent.txt
  \net20
    \MyContent20.txt
  \net40
  \sl40
    \MySilverlightContent.html

\tools
  \init.ps1
  \net40
    \install.ps1
    \uninstall.ps1
  \sl40
    \install.ps1
    \uninstall.ps1
```

注意：由于 `init.ps1` 是在解决方案级别执行的，并且不依赖于任何单个项目，因此它必须直接置于 `tools` 文件夹下。如果放置在框架特定的文件夹中，则将被忽略。

此外，NuGet 2.0 中的一项新功能是框架文件夹可以是空的，在这种情况下，NuGet 不会添加程序集引用、添加内容文件或针对特定的框架版本运行 PowerShell 脚本。在上面的示例中，文件夹 `content\net40` 为空。

改善了选项卡完成性能

NuGet 包管理器控制台中的选项卡完成功能已更新，以显著提高性能。按下 `tab` 键时，延迟时间会大大减少，直到出现建议下拉列表。

Bug 修复

NuGet 2.0 包含多个 bug 修复，重点介绍包还原同意和性能。有关 NuGet 2.0 中已修复的工作项的完整列表，请查看[此版本的 NuGet 问题跟踪程序](#)。

NuGet 1.8 发行说明

2020/1/8 • [Edit Online](#)

[Nuget 1.7 发行说明](#) | [Nuget 2.0 发行说明](#)

NuGet 1.8 于年5月 23 2012 日发布。

已知安装问题

如果你运行的是 VS 2010 SP1, 但如果你安装了较旧的版本, 你可能会遇到安装错误。

解决方法是仅卸载 NuGet, 然后从 VS 扩展库安装它。有关详细信息, 请参阅
<https://support.microsoft.com/kb/2581019>, 或直接跳到 VS 修补程序。

注意:如果 Visual Studio 不允许你卸载扩展("卸载" 按钮已禁用), 则可能需要使用 "以管理员身份运行" 重启 Visual Studio。

NuGet 1.8 与 Windows XP 兼容, 已发布修补程序

在 NuGet 1.8 发布之后不久, 我们已了解到1.8 中的密码更改在 Windows XP 上中断了用户。

我们已经发布了解决此问题的修补程序。通过在 Visual Studio 扩展库中更新 NuGet, 你会收到此修补程序。

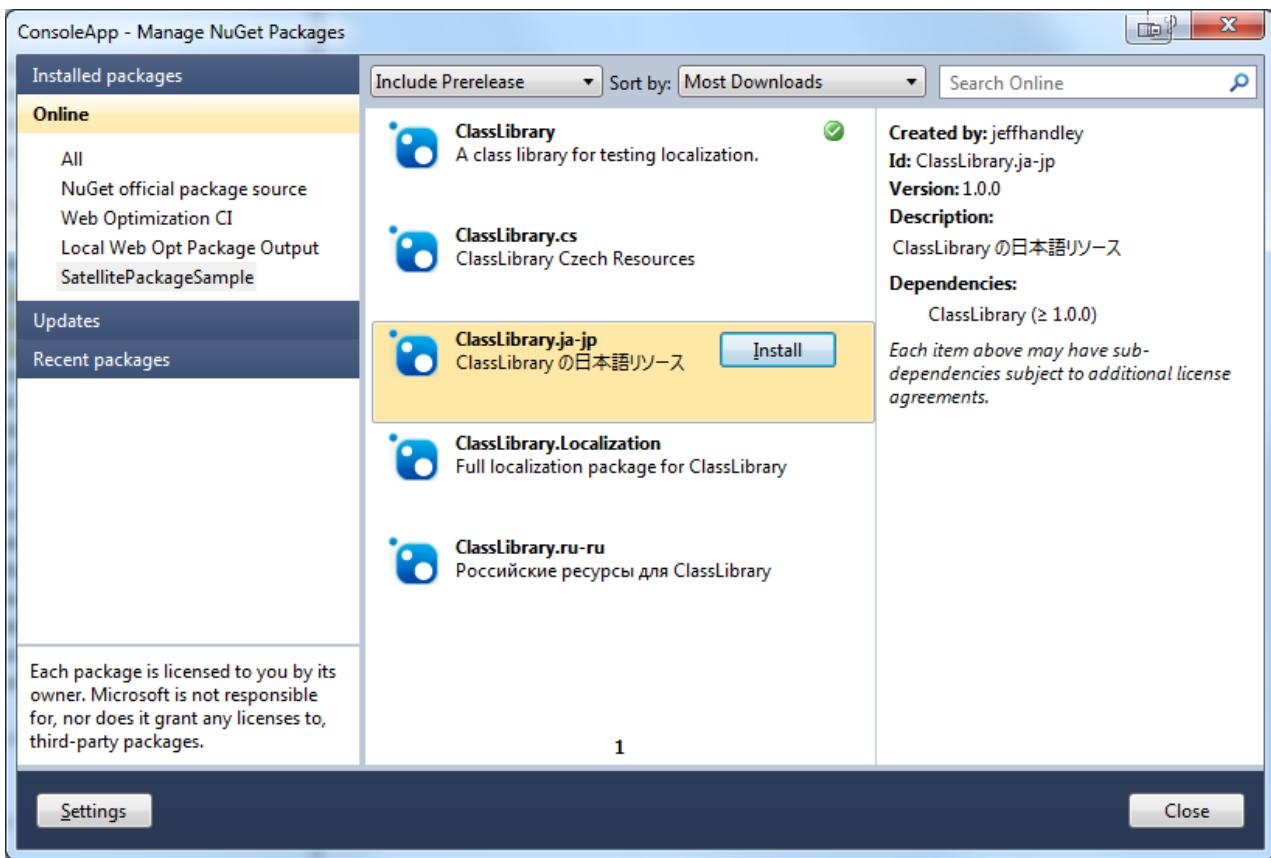
特征

本地化资源的附属包

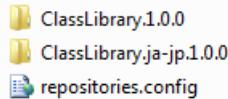
NuGet 1.8 现在支持为本地化资源创建单独包的功能, 类似于.NET Framework 的附属程序集功能。附属包的创建方式与任何其他 NuGet 包相同, 只是添加了一些约定:

- 附属包 ID 和文件名应包含与[.NET Framework 使用的标准区域性字符串](#)之一匹配的后缀。
- 在其 `.nuspec` 文件中, 附属包应定义一个语言元素, 该元素具有在 ID 中使用的相同区域性字符串
- 附属包应将其 `.nuspec` 文件中的依赖项定义为其核心包, 这只是 ID 与语言后缀相同的包。核心包需要在存储库中可用, 才能成功安装。

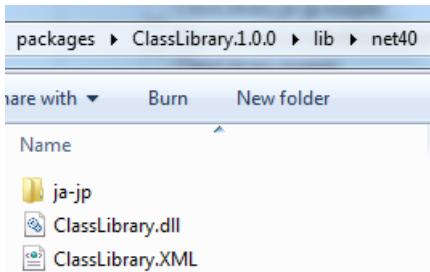
若要安装包含本地化资源的包, 开发人员需要从存储库中显式选择本地化包。目前, NuGet 库不会向附属包提供任何类型的特殊处理。



由于附属包列出了其核心包的依赖项，因此附属包和核心包都将提取到 NuGet 包文件夹中并安装。



此外，在安装附属包时，NuGet 还识别区域性字符串命名约定，然后将本地化的资源程序集复制到核心包内的正确子文件夹中，以便 .NET Framework 可以对其进行选取。



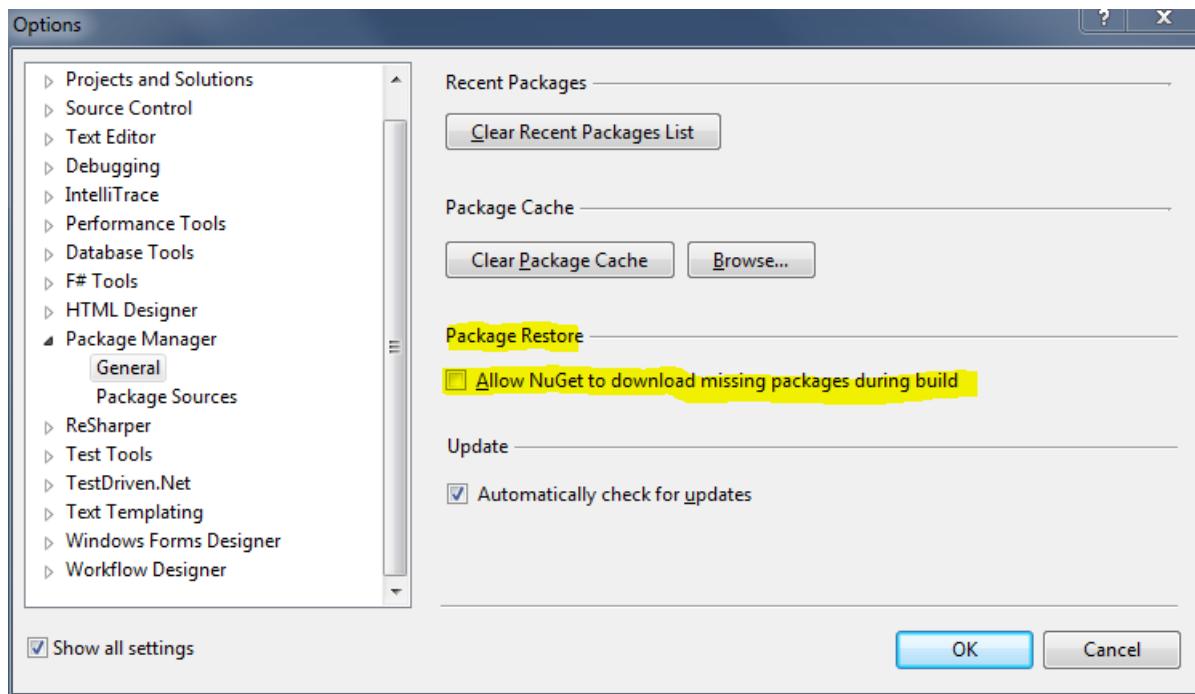
附属包需要注意的一个现有 bug 是 NuGet 不会将本地化的资源复制到网站项目的 `bin` 文件夹中。此问题将在 NuGet 的下一版本中得到解决。

有关演示如何创建和使用附属包的完整示例，请参阅<https://github.com/NuGet/SatellitePackageSample>。

包还原许可

在 NuGet 1.8 中，我们为支持包还原中的重要约束设定了基础，以保护用户隐私。此约束要求开发人员生成项目和解决方案，这些项目和解决方案使用包还原来显式同意包还原的联机状态，以便从配置的包源下载包。

有两种方法可提供此许可。第一个可以在“包管理器配置”对话框中找到，如下所示。此方法主要用于开发人员计算机。



第二种方法是将环境变量 "EnableNuGetPackageRestore" 设置为值 "true"。此方法适用于无人参与的计算机，例如 CI 或生成服务器。

现在，如上所述，我们只在 NuGet 1.8 中为此功能奠定了基础。实际上，这意味着虽然我们添加了所有逻辑来启用此功能，但当前未在此版本中强制执行此功能。但在 NuGet 的下一个版本中会启用此功能，因此，我们希望你尽快了解该功能，以便你可以适当地配置环境，从而在我们开始强制许可约束时不会受到影响。

有关更多详细信息，请参阅[团队博客文章](#)此功能。

nuget.exe 性能改进

通过修改安装命令以并行下载和安装包，NuGet 1.8 使 nuget.exe 的性能得到显著改善，并通过扩展包还原提高了性能。高级别测试表明，将6个包安装到项目中的性能将在 NuGet 1.8 中提高大约35%。将包数增加到25会显示大约60% 的性能提升。

Bug 修复

NuGet 1.8 包含非常多的 bug 修复，其中重点介绍了包管理器控制台和包还原工作流，尤其是在它与包还原许可和 Windows 8 Express 集成相关时。有关 NuGet 1.8 中已修复的工作项的完整列表，请查看[此版本的 NuGet 问题跟踪程序](#)。

NuGet 1.7 发行说明

2020/1/8 • [Edit Online](#)

[Nuget 1.6 发行说明](#) | [Nuget 1.8 发行说明](#)

NuGet 1.7 于年4月 4 2012 日发布。

已知安装问题

如果你运行的是 VS 2010 SP1, 但如果你安装了较旧的版本, 你可能会遇到安装错误。

解决方法是仅卸载 NuGet, 然后从 VS 扩展库安装它。有关更多信息, 请参见<https://support.microsoft.com/kb/2581019>。

注意:如果 Visual Studio 不允许你卸载扩展("卸载" 按钮已禁用), 则可能需要使用 "以管理员身份运行" 重启 Visual Studio。

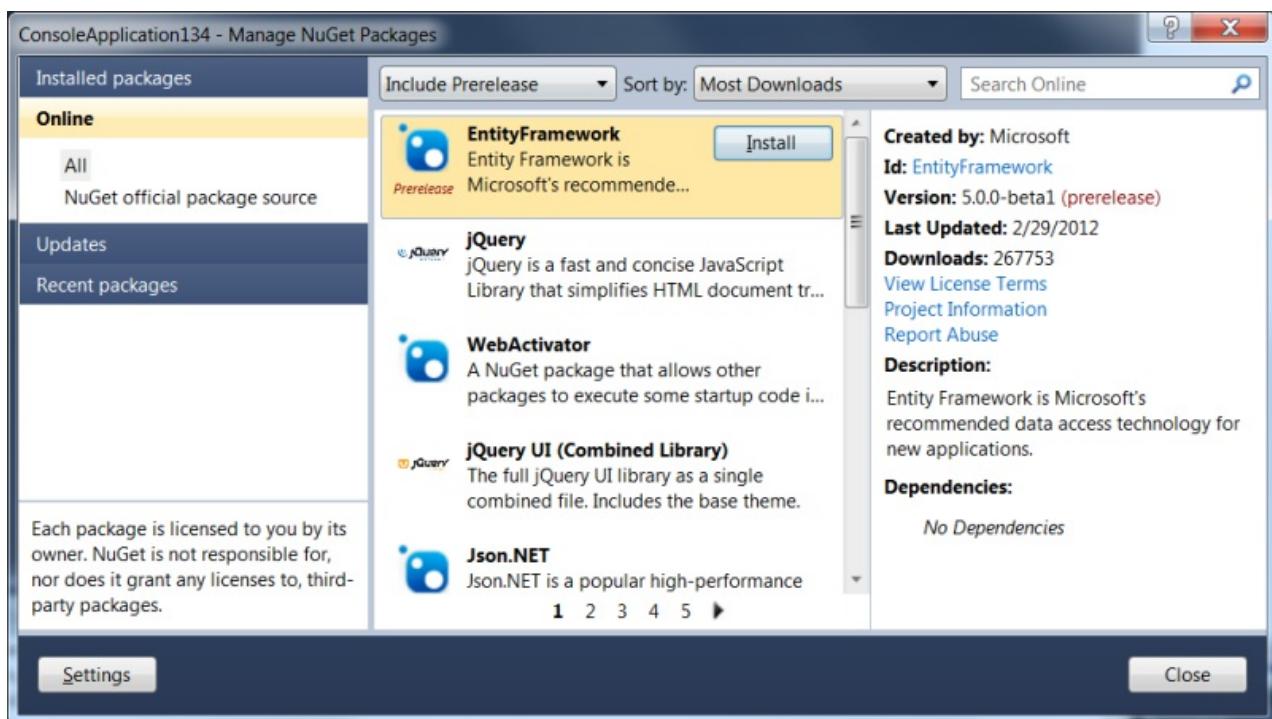
特征

支持在安装后打开 readme.txt 文件

1.7 中的新增内容, 如果你的包在包的根目录中包含一个 `readme.txt` 文件, 则在安装完包之后, NuGet 将自动打开此文件。

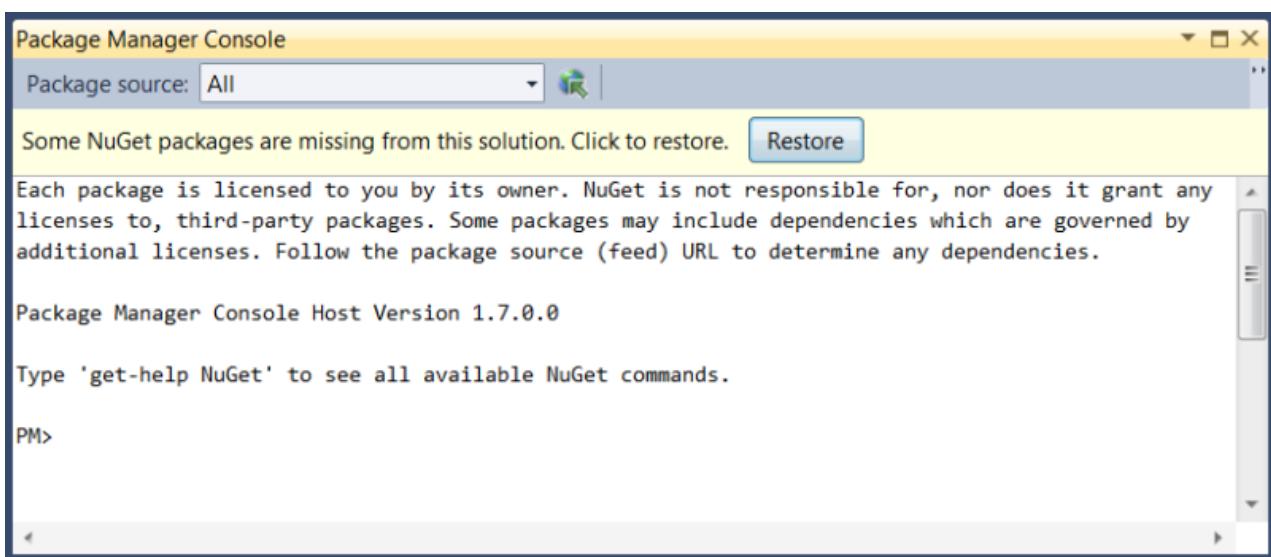
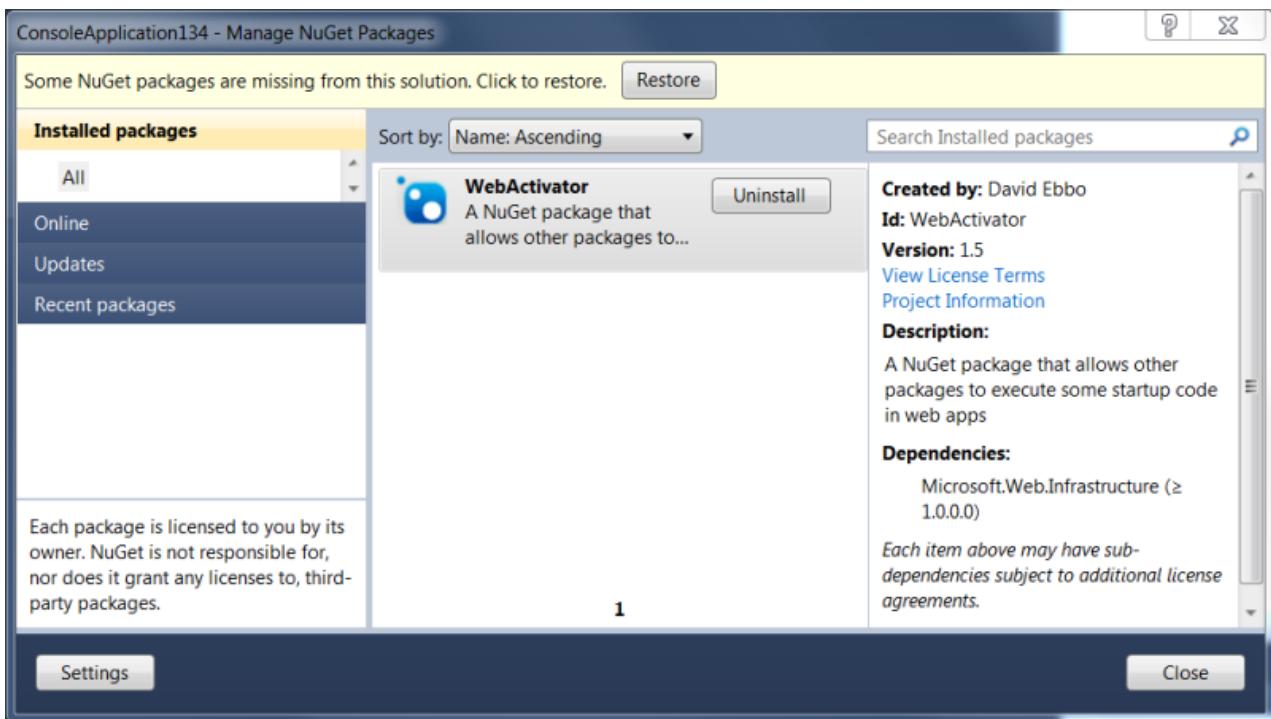
在 "管理 NuGet 包" 对话框中显示预发行包

"管理 NuGet 包" 对话框现在包含一个下拉列表, 其中提供了显示预发行包的选项。



缺少包文件时显示 "包还原" 按钮

打开 "程序包管理器控制台" 或 "管理器 NuGet 包" 对话框时, NuGet 将检查当前解决方案是否已启用包还原模式, 以及 `packages` 文件夹中是否缺少任何包文件。如果满足这两个条件, NuGet 会通知你, 并显示一个便利的 "还原" 按钮。单击此按钮将触发 NuGet 以还原所有缺少的包。



添加解决方案级包 .config 文件

在 NuGet 的早期版本中，每个项目都有一个 `packages.config` 文件，该文件跟踪在该项目中安装的 NuGet 包。不过，解决方案级别没有类似的文件来跟踪解决方案级包。因此，无法还原解决方案级包。此功能现已在 NuGet 1.7 中实现。解决方案级 `packages.config` 文件位于解决方案根下的 `.nuget` 文件夹下，并且将仅存储解决方案级包。

删除新包命令

由于使用率低，已删除新包命令。建议开发人员使用 `nuget.exe` 或方便的 NuGet 包资源管理器来创建包。

Bug 修复

NuGet 1.7 在包还原工作流和网络/源代码管理方案周围修复了很多 bug。

有关 NuGet 1.7 中已修复的工作项的完整列表，请查看[此版本的 NuGet 问题跟踪程序](#)。

NuGet 1.6 发行说明

2020/1/8 • [Edit Online](#)

[Nuget 1.5 发行说明](#) | [Nuget 1.7 发行说明](#)

NuGet 1.6 于2011年12月13日发布。

已知安装问题

如果你运行的是 VS 2010 SP1，但如果你安装了较旧的版本，你可能会遇到安装错误。

解决方法是仅卸载 NuGet，然后从 VS 扩展库安装它。有关更多信息，请参见<https://support.microsoft.com/kb/2581019>。

注意：如果 Visual Studio 不允许你卸载扩展（“卸载”按钮已禁用），则可能需要使用“以管理员身份运行”重启 Visual Studio。

特征

支持语义版本控制和预发行包

NuGet 1.6 引入了对语义版本控制(SemVer)的支持。有关如何使用 SemVer 的更多详细信息，请参阅[版本控制文档](#)。

使用 NuGet 而不签入包(包还原)

NuGet 1.6 现在提供对工作流的第一类支持，其中 NuGet 包未添加到源代码管理中，但在生成时将在缺少时还原。有关更多详细信息，请阅读[使用 NuGet 但不将包提交到源代码管理](#)主题。

安装 NuGet 包的项模板

基于支持预安装 NuGet 包的工作以支持 Visual Studio 项目模板，NuGet 1.6 还添加了对 Visual Studio 项模板的支持。项模板可以具有在调用模板时安装的关联 NuGet 包。

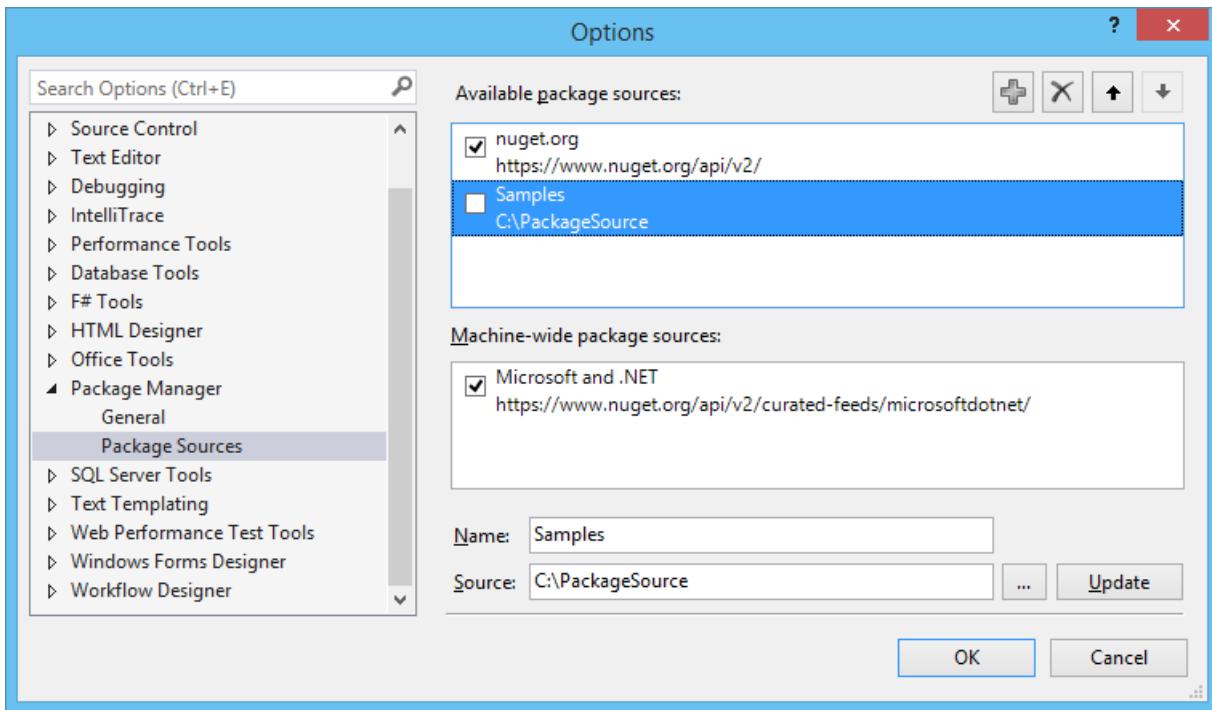
有关如何更改项目/项模板以安装 NuGet 包的详细信息，请阅读[Visual Studio 模板中的包](#)主题。

支持禁用包源

当配置了多个包源时，NuGet 将在包及其依赖项的安装过程中为包查找每个源。出于某种原因关闭的包源可能严重降低了 NuGet 速度。

在 NuGet 1.6 之前，你可以删除包源，但当你想要将其重新添加时，必须记住的详细信息。

NuGet 1.6 允许不选中包源来禁用它，但要进行保留。



Bug 修复

NuGet 1.6 总共修复了106个工作项。其中的95归类为 bug, 其中10个是功能。

有关 NuGet 1.6 中已修复的工作项的完整列表, 请查看[此版本的 NuGet 问题跟踪程序](#)。

NuGet 1.5 发行说明

2020/1/8 • [Edit Online](#)

[NuGet 1.4 发行说明](#) | [NuGet 1.6 发行说明](#)

NuGet 1.5 于2011年8月30日发布。

特征

预安装了 NuGet 包的项目模板

创建新的 ASP.NET MVC 3 项目模板时，项目中包含的 jQuery 脚本库实际上是通过安装 NuGet 包放置在那里。

ASP.NET MVC 3 项目模板包含一组在调用项目模板时安装的 NuGet 包。此功能能够将 NuGet 包包含到项目模板中，它现在是 NuGet 的一项功能，现在任何项目模板都可以利用。

有关此功能的更多详细信息，请阅读此博客文章，了解该[功能的开发人员](#)。

显式程序集引用

添加了新的 `<references />` 元素，用于显式指定应引用包中的程序集。

例如，如果添加以下内容：

```
<references>
  <reference file="xunit.dll" />
  <reference file="xunit.extensions.dll" />
</references>
```

然后，将仅从 `lib` 文件夹的相应[框架/配置文件子](#)文件夹中引用 `xunit.dll` 和 `xunit.extensions.dll`，即使该文件夹中存在其他程序集。

如果省略此元素，则会应用常用行为，即引用 `lib` 文件夹中的每个程序集。

此功能的用途是什么？

此功能支持仅设计时程序集。例如，使用代码协定时，协定程序集需要位于它们增加的运行时程序集旁边，以便 Visual Studio 可以找到它们，但是协定程序集实际上不应被项目引用，因此不应复制到 `bin` 文件夹中。

同样，此功能可用于单元测试框架（如 XUnit），这需要将其工具程序集放在运行时程序集旁边，但从项目引用中排除。

添加了排除 nuspec 中的文件的功能

`.nuspec` 文件中的 `<file>` 元素可用于包含特定文件或使用通配符的一组文件。使用通配符时，无法排除包含的文件的特定子集。例如，假设您希望文件夹中的所有文本文件除外。

```
<files>
  <file src="*.txt" target="content\docs" exclude="admin.txt" />
</files>
```

使用分号指定多个文件。

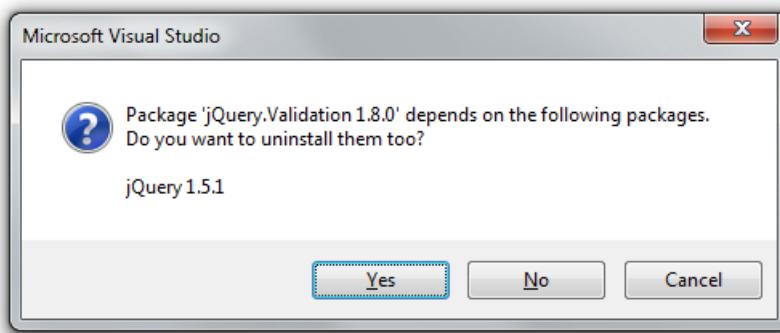
```
<files>
  <file src="*.txt" target="content\docs" exclude="admin.txt;log.txt" />
</files>
```

或使用通配符排除一组文件，如所有备份文件

```
<files>
  <file src="tools\*.*" target="tools" exclude="*.bak" />
</files>
```

使用对话框删除包会提示删除依赖项

卸载具有依赖项的包时，NuGet 会提示，允许删除包的依赖项和包。



Get-Package 命令改进

Get-Package 命令现在支持 **-ProjectName** 参数。因此命令

```
Get-Package -ProjectName A
```

将列出在项目 A 中安装的所有包。

支持需要身份验证的代理

使用需要身份验证的代理后面的 NuGet 时，NuGet 现在会提示输入代理凭据。输入凭据后，NuGet 即可连接到远程存储库。

支持需要身份验证的存储库

NuGet 现在支持连接到需要基本或 NTLM 身份验证的[专用存储库](#)。

未来版本中将添加对摘要式身份验证的支持。

Nuget.org 存储库的性能改进

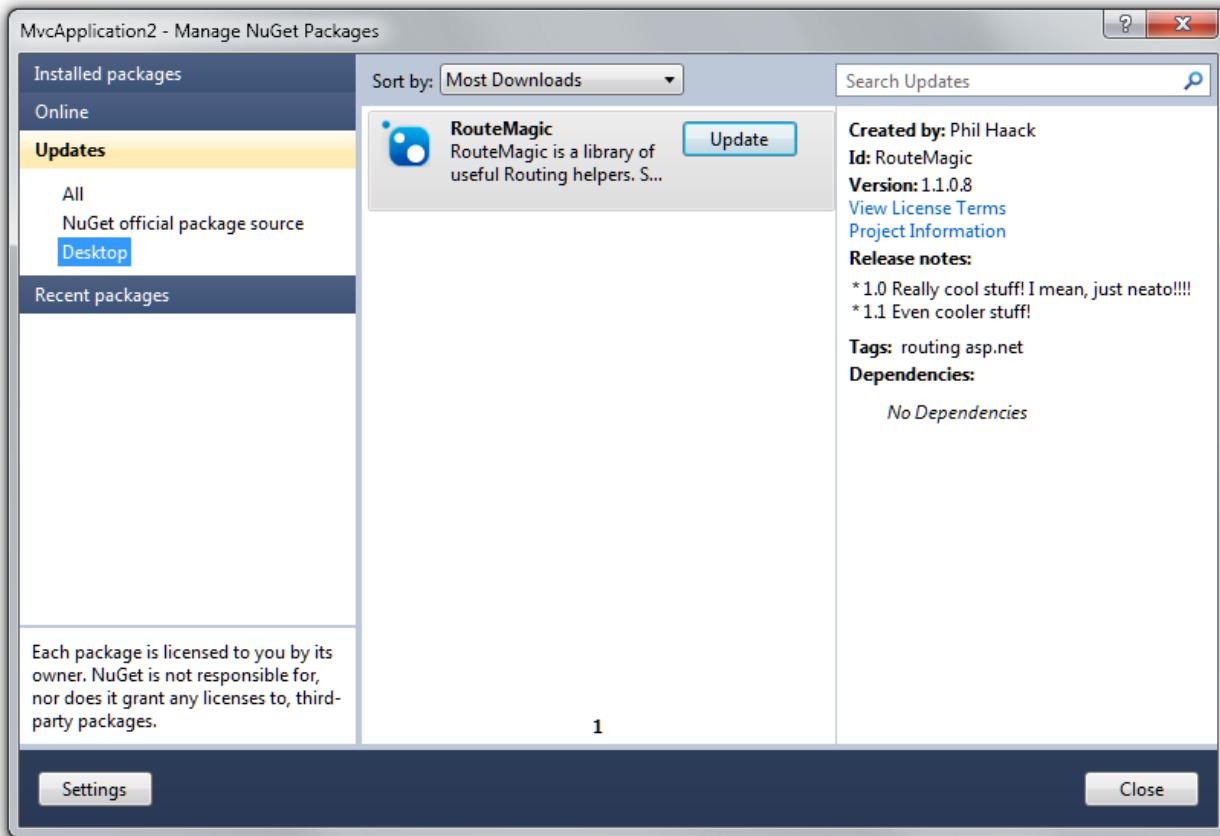
我们对 nuget.org 库进行了几项性能改进，使包列表和搜索速度更快。

解决方案对话框项目筛选

在解决方案级对话框中，当提示您要安装的项目时，我们只显示与所选包兼容的项目。

包发行说明

NuGet 包现在包括对发行说明的支持。仅当查看包的 更新 时才会显示发行说明，因此，将其添加到第一个版本中并没有什么意义。



若要向包添加发行说明, 请使用 NuSpec 文件中的新 `<releaseNotes />` metadata 元素。

.nuspec & Itfiles/> 改善

.nuspec 文件现在允许空 `<files />` 元素, 这会告诉 nuget.exe 不要包含包中的任何文件。

Bug 修复

NuGet 1.5 总共修复了107个工作项。103标记为 bug。

有关 NuGet 1.5 中已修复的工作项的完整列表, 请查看[此版本的 NuGet 问题跟踪程序](#)。

需要注意的 Bug 修复 :

- [问题 1273](#):通过按字母顺序对包进行排序并删除额外的空白, 使 `packages.config` 更好地控制版本。
- [问题 844](#):版本号现已规范化, 以便 `Install-Package 1.0` 适用于 `1.0.0` 版本的包。
- [问题 1060](#):使用 nuget.exe 创建包时, `-Version` 标志将重写 `<version />` 元素。

NuGet 1.4 发行说明

2020/3/3 • [Edit Online](#)

[NuGet 1.3 发行说明](#) | [NuGet 1.5 发行说明](#)

NuGet 1.4 于2011年6月17日发布。

功能

更新-包改进

NuGet 1.4 为更新包命令引入了很多改进，使解决方案中多个项目的包保持同一版本。例如，将包升级到最新版本时，需要将安装该程序包的所有项目更新为相同的 verision，这种情况很常见。

`Update-Package` 命令现在可以更轻松地执行以下操作：

更新单个项目中的所有包

```
Update-Package -Project MvcApplication1
```

更新所有项目中的包

```
Update-Package PackageId
```

更新所有项目中的所有包

```
Update-Package
```

在所有包上执行“安全”更新

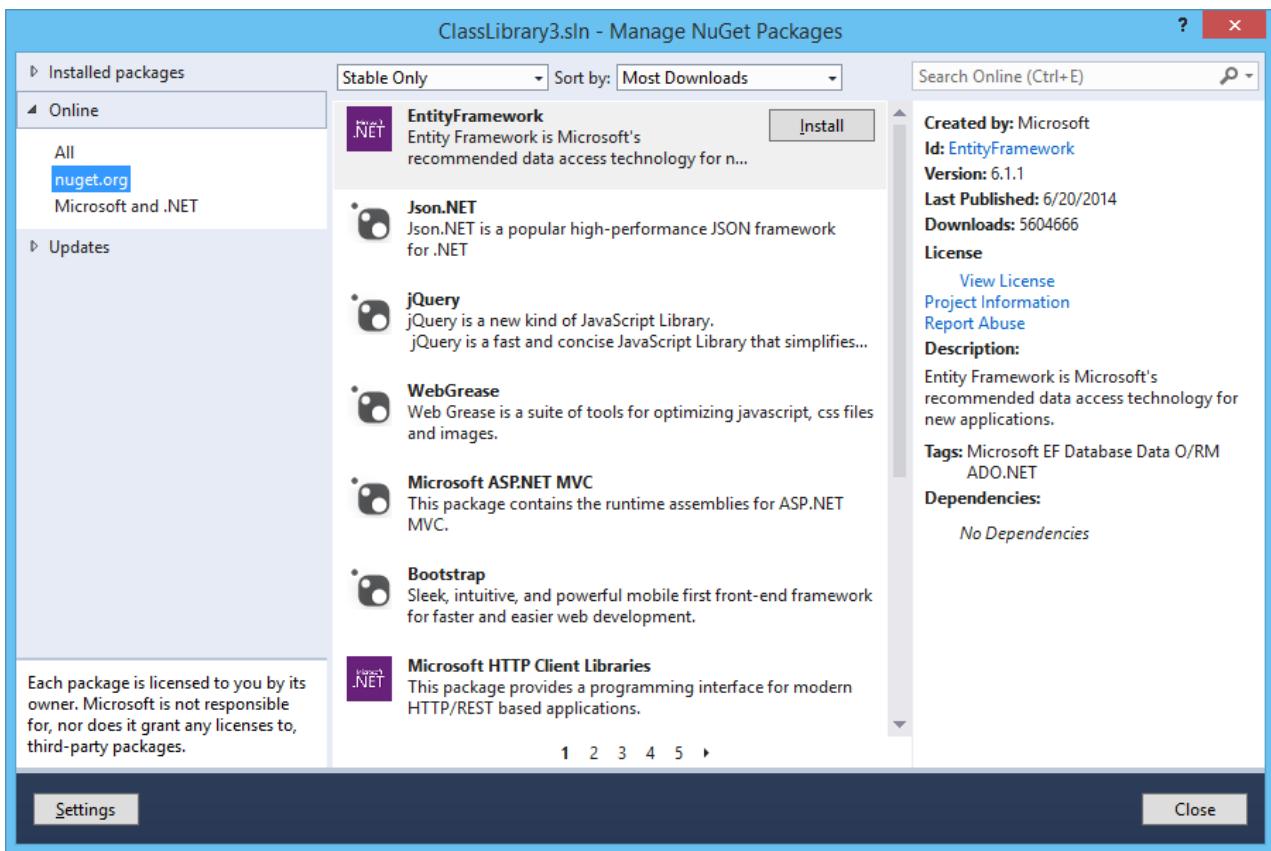
`-Safe` 标志仅将升级限制为具有相同的主要版本和次要版本组件的版本。例如，如果安装了包的1.0.0 版，并且源中提供了版本1.0.1、1.0.2 和1.1，则 `-Safe` 标志会将包更新到1.0.2。如果升级时没有 `-Safe` 标志，会将包升级到最新版本1.1。

```
Update-Package -Safe
```

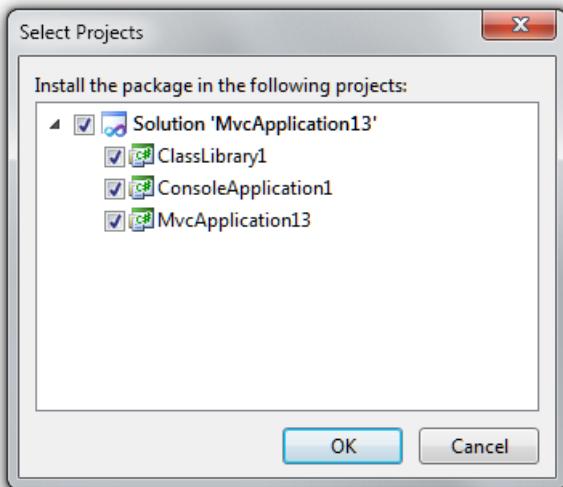
在解决方案级别管理包

在 NuGet 1.4 之前，使用对话框将包安装到多个项目中会很繁琐。每个项目都需要启动一次此对话。

NuGet 1.4 添加了对在多个项目中同时安装/卸载/更新包的支持。只需通过右键单击解决方案并选择“**管理 NuGet 包**”菜单选项即可启动。



请注意，在对话框的标题栏中显示解决方案的名称，而不是项目的名称。包操作现在提供了一个复选框列表，其中包含操作应应用到的项目的列表。



有关更多详细信息，请参阅有关[管理解决方案的包](#)的主题。

限制升级到允许的版本

默认情况下，在包上运行 `Update-Package` 命令（或使用对话框更新包）时，它将更新为源中的最新版本。使用更新所有包的新支持，可能需要将包锁定到特定的版本范围。例如，你可能事先知道，你的应用程序将仅适用于包的版本 2.*，而不是 3.0 和更高版本。为了防止意外将包更新到 3，NuGet 1.4 添加了对约束可升级到的版本范围的支持，方法是使用新的 `allowedVersions` 属性手动编辑 `packages.config` 文件。

例如，下面的示例演示如何将 `SomePackage` 包锁定为 2.0-3.0（独占）版本。`allowedVersions` 属性接受使用[版本范围格式](#)的值。

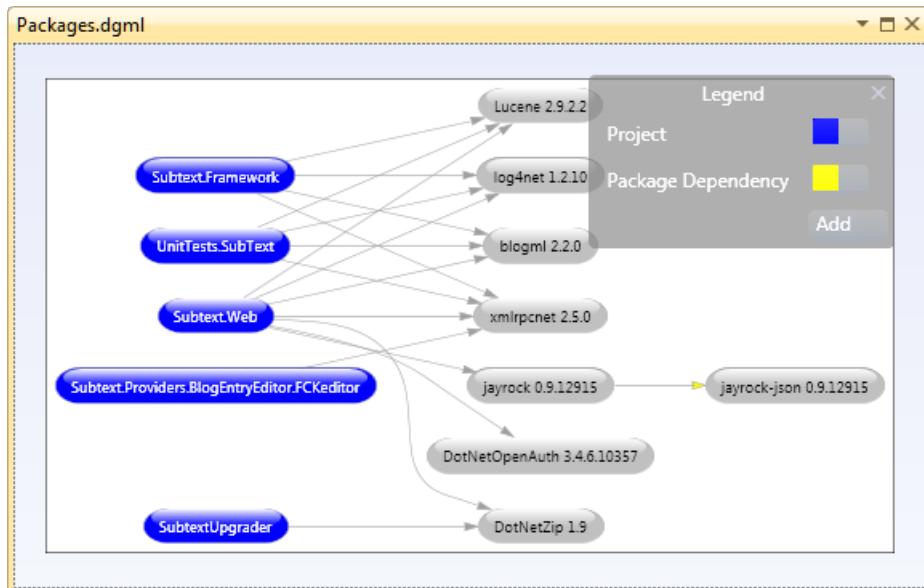
```
<?xml version="1.0" encoding="utf-8"?>
<packages>
    <package id="SomePackage" version="2.1.0" allowedVersions="[2.0, 3.0)" />
</packages>
```

请注意，在1.4中，必须手动编辑将包锁定到特定的版本范围。在NuGet 1.5中，我们计划添加对通过
Install-Package命令放置此范围的支持。

包可视化工具

通过“工具”->“库包管理器”->“包可视化工具”菜单选项启动的新包可视化工具，可以轻松地将所有项目及其包依赖关系可视化到解决方案中。

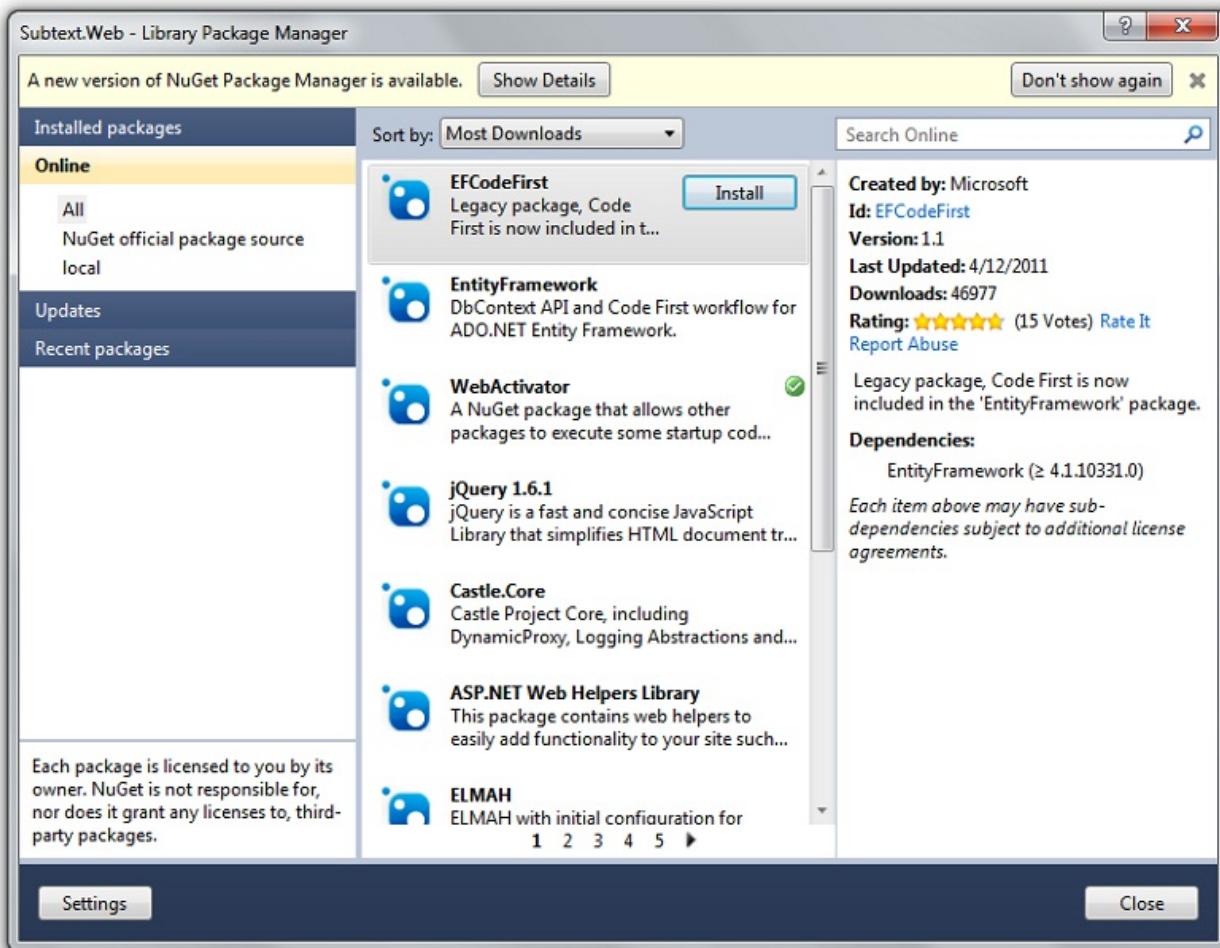
重要说明：此功能利用Visual Studio中的DGML支持。仅Visual Studio Ultimate支持创建可视化效果。仅Visual Studio Premium或更高版本中支持查看DGML关系图。



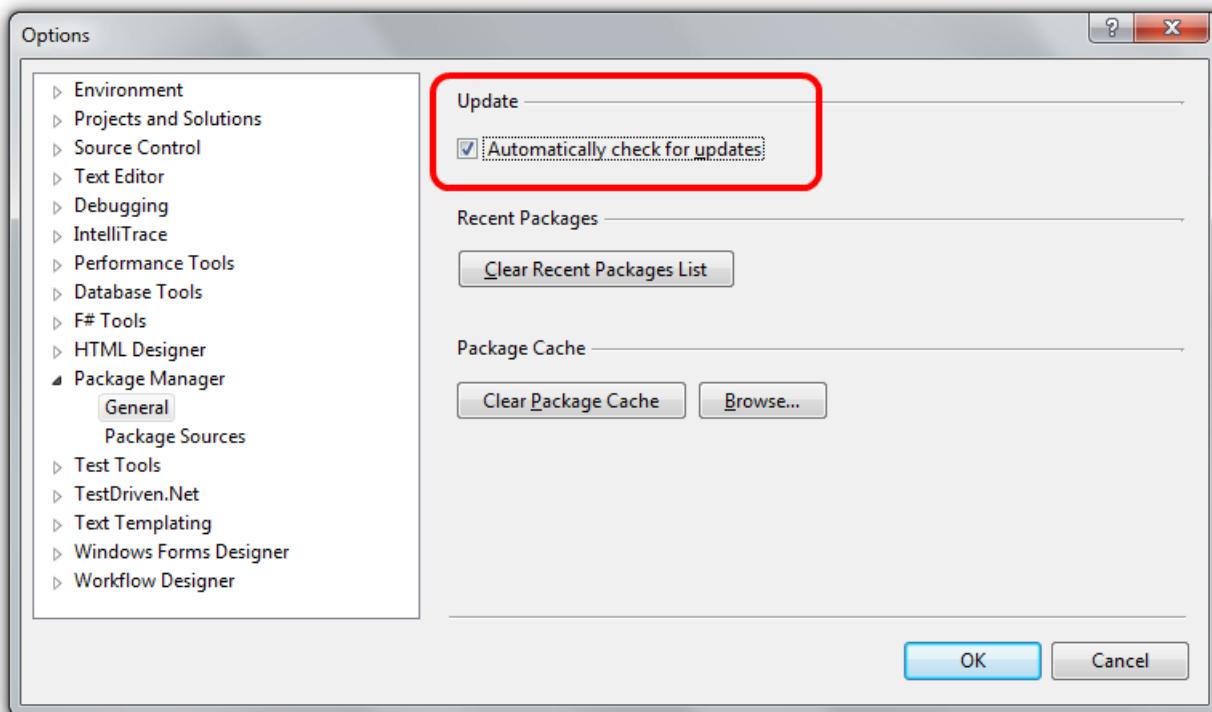
NuGet 对话框的自动更新检查

某些版本的NuGet引入通过.nuspec文件表示的新功能，这些新功能在NuGet对话框的较旧版本中不被理解。其中一个示例是NuGet 1.4中用于[指定框架程序集](#)的简介。因此，请务必使用最新版本的NuGet，以确保可以使用包利用最新功能。为了更直观地更新NuGet，NuGet对话框包含了在更新版本可用时要突出显示的逻辑。

注意：仅当在当前会话中选择了“联机”选项卡时，才进行此检查。



若要禁用自动检查更新，请转到 NuGet 设置对话框，并取消选中“自动检查更新”。



此功能实际上已添加到 NuGet 1.3 中，但在更新到1.3（如 NuGet 1.4）之前，将不会显示此功能。

程序包管理器对话框改进

- **菜单名称改进**: 为了清楚起见，已重命名用于启动对话框的菜单选项。菜单选项现在为“管理 NuGet 包”。
- **详细信息窗格显示最新更新日期**: 在选择“联机”或“更新”选项卡时，NuGet 对话框显示包的详细信息窗格中

的最新更新的日期。

- **显示的标记列表**: Nuget 对话框显示标记。

Powershell 改进

- **已签名的 powershell 脚本**: NuGet 包括已签名的 powershell 脚本, 以实现更严格的环境中的使用。
- **提示支持**: 程序包管理器控制台现在支持通过 `$host.ui.Prompt` 和 `$host.ui.PromptForChoice` 命令发出提示。
- **包源名称**: 在使用 `-Source` 标志指定包源时支持提供包源的名称。

nuget.exe 命令行改进

- **Nuget 自定义命令**: 使用 MEF 通过自定义命令可扩展 nuget.exe。
- **更简单地创建符号包的工作流**: `-Symbols` 标志可应用于基于标准的基于约定的文件夹结构, 该结构仅通过在文件夹中包含源和 `.pdb` 文件来创建符号包。
- **指定多个源**: `NuGet install` 命令支持使用分号作为分隔符或多次指定 `-Source` 来指定多个源。
- **代理身份验证支持**: 在需要身份验证的代理后面使用 nuget 时, nuget 1.4 添加了对提示用户凭据的支持。
- **Nuget.exe 更新重大更改**: 现在, nuget.exe 需要 `-Self` 标志才能更新自身。`nuget.exe Update` 现在采用 `packages.config` 文件的路径, 并将尝试更新包。请注意, 此更新有限, 因为它不会: ** 更新、添加、删除项目文件中的内容。** 在包中运行 Powershell 脚本。

NuGet 服务器支持使用 nuget.exe 推送包

NuGet 包含一种通过 `NuGet.Server` NuGet 包来托管**基于轻型 web 的 nuget 存储库**的简单方式。使用 NuGet 1.4, 轻型服务器支持使用 nuget.exe 推送和删除包。最新版本的 `NuGet.Server` 添加名为 `apiKey` 的新 `appSetting`。如果省略该密钥或将其保留为空, 则会禁用向该源推送包。将 `apiKey` 设置为值(理想情况下为强密码)可启用使用 nuget.exe 推送包。

```
<appSettings>
    <!-- Set the value here to allow people to push/delete packages from the server.
        NOTE: This is a shared key (password) for all users. -->
    <add key="apiKey" value="" />
</appSettings>
```

支持 Windows Phone Tools Mango Edition

Mango 的 Windows Phone 工具的候选发布版本中现在支持 NuGet。目前, Windows Phone 工具不支持 Visual Studio 扩展管理器, 因此若要为 Windows Phone 工具安装 NuGet, 你可能需要手动下载并运行该 VSIX。

若要卸载 Windows Phone 工具的 NuGet, 请运行以下命令。

```
vsixinstaller.exe /uninstall:NuPackToolsVsix.Microsoft.67e54e40-0ae3-42c5-a949-fddff5739e7a5
```

Bug 修复

NuGet 1.4 总共修复了88个工作项。71标记为 bug。

有关 NuGet 1.4 中已修复的工作项的完整列表, 请查看[此版本的 NuGet 问题跟踪程序](#)。

需要注意的 Bug 修复:

- [问题 603](#): 指定特定包源时, 跨不同存储库的包依赖项解析正确。
- [问题 1036](#): 将 `NuGet Pack SomeProject.csproj` 添加到生成后事件不再导致无限循环。
- [问题 961](#): `-Source` 标志支持相对路径。

NuGet 1.4 更新

在 NuGet 1.4 发布之后不久，我们发现了一些很重要的问题需要解决。此更新到1.4 的特定版本号为 1.4.20615.9020。

Bug 修复

- [问题 1220](#): 如果有多个项目，则更新包不会在所有项目中执行 `install.ps1` / `uninstall.ps1`
- [问题 1156](#): 程序包管理器控制台在 W2K3/XP 上停滞(未安装 Powershell 2 时)

NuGet 1.3 发行说明

2019/12/6 • [Edit Online](#)

[NuGet 1.2 发行说明](#) | [NuGet 1.4 发行说明](#)

NuGet 1.3 于2011年4月25日发布。

新增功能

通过符号服务器集成简化包创建

NuGet 团队与 [SymbolSource.org](#) 的用户进行了合作，提供了一种非常简单的方式来将源和 PDB 与包一起发布。这允许你的包的使用者在调试器中单步执行包的源。有关更多详细信息，请参阅[创建和发布符号包](#)使用源发布 NuGet 包的简单方法。你还可以观看此功能的实时演示，作为 NuGet 深入了解 Mix11。此功能在视频的20分钟标记处开始完全演示。

`Open-PackagePage` 命令

使用此命令可以轻松地从包管理器控制台中访问包的项目页。它还提供了用于打开包的 "许可证 URL" 和 "报告滥用" 页的选项。命令的语法为：

```
Open-PackagePage -Id <string> [-Version] [-Source] [-License] [-ReportAbuse] [-PassThru]
```

`-PassThru` 选项用于返回指定 URL 的值。

例如：

```
PM> Open-PackagePage Ninject
```

打开浏览器，指向在 Ninject 包中指定的项目 URL。

```
PM> Open-PackagePage Ninject -License
```

打开浏览器，指向在 Ninject 包中指定的许可证 URL。

```
PM> Open-PackagePage Ninject -ReportAbuse
```

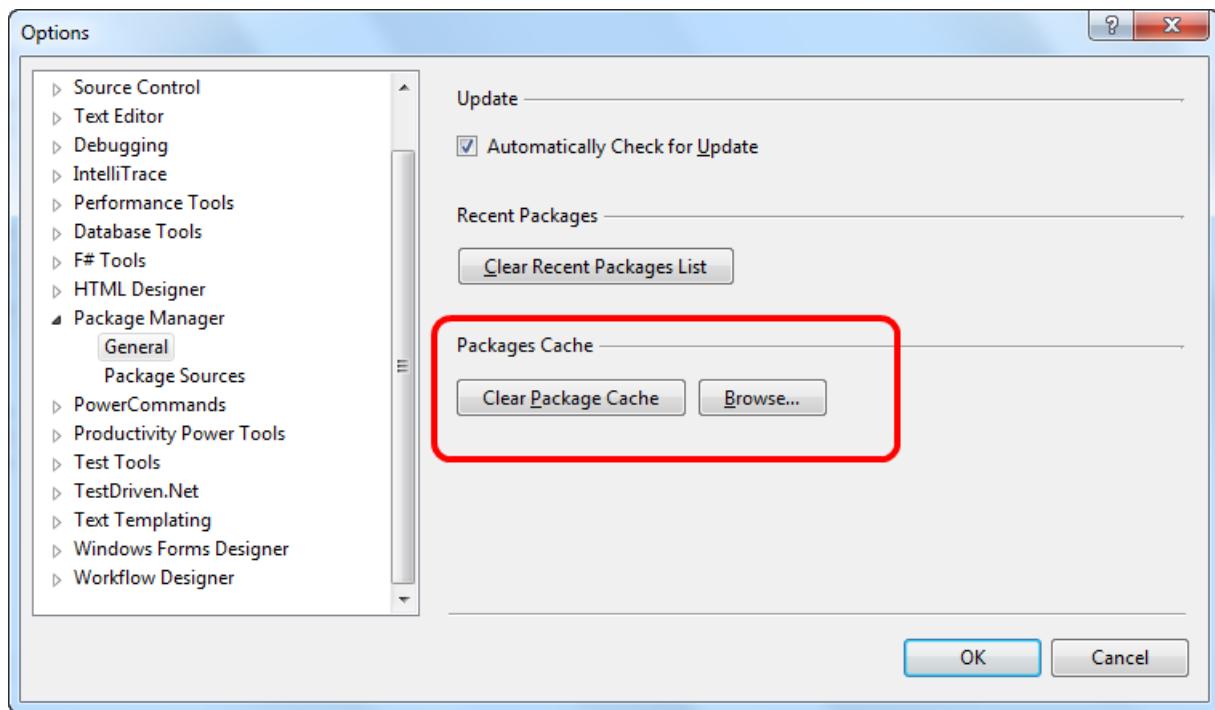
打开浏览器，指向当前包源中用于报告指定包的滥用的 URL。

```
PM> $url = Open-PackagePage Ninject -License -WhatIf -PassThru
```

将许可证 URL 分配给变量 \$url，而无需在浏览器中打开该 URL。

性能改进

NuGet 1.3 引入了大量性能改进。NuGet 1.3 通过包含本地的每用户缓存避免多次下载包的同一版本。可以通过 "包管理器设置" 对话框访问和清除缓存：



其他性能改进包括添加对 HTTP 压缩的支持和在 Visual Studio 中提高包的安装速度。

Visual Studio 和 nuget.exe 使用相同的包源列表

在 NuGet 1.3 之前, nuget.exe 和 NuGet Visual Studio 外接程序使用的包源列表未存储在同一位置。NuGet 1.3 现在同时在这两个位置使用同一列表。该列表存储在 `NuGet.Config` 中并存储在 AppData 文件夹中。

默认情况下, nuget.exe 忽略以 "." 开头的文件和文件夹

为了使 NuGet 适用于源控制系统(如 Subversion 和 Mercurial), nuget.exe 会忽略在创建包时以 "." 字符开头的文件夹和文件。这可以使用两个新标志进行重写:

- `-NoDefaultExcludes` 用于覆盖此设置并包括所有文件。
- `-Exclude` 用于使用模式添加要排除的其他文件/文件夹。例如, 若要排除文件扩展名为 ".bak" 的所有文件

```
nuget Pack MyPackage.nuspec -Exclude **\*.bak
```

注意:默认情况下, 该模式不是递归的。

对 WiX 项目和 .NET 微框架的支持

由于社区贡献, NuGet 包含对 WiX 项目类型和 .NET 微框架的支持。

Bug 修复

有关 bug 修复的完整列表, 请查看[此版本的 NuGet 问题跟踪程序](#)。

Bug 修复值得注意

- 包含源文件的包可在网站和 Web 应用程序项目中使用。对于网站, 源文件将复制到 `App_Code` 文件夹中

NuGet 1.2 发行说明

2020/3/19 • [Edit Online](#)

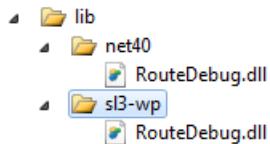
[NuGet 1.0 和 1.1 发行说明](#) | [NuGet 1.3 发行说明](#)

NuGet 1.2 于年3月 30 2011 日发布。

新功能

框架配置文件支持

从开始, NuGet 支持的库以不同的框架为目标。但现在, 包可能包含面向特定配置文件(如 Windows Phone 配置文件)的程序集。若要以框架的特定配置文件为目标, 请追加一条短划线, 后跟配置文件缩写。例如, 若要面向在 Windows Phone 上运行的 SilverLight (也称为 Windows Phone 7), 可以将程序集放入 sl3 wp 文件夹中, 如以下屏幕截图所示。



您可能会问为什么我们不只选择使用 "wp7" 作为名字对象。在部分中, 我们将预测到 Windows Phone 7 以后可能会运行较新版本的 Silverlight, 在这种情况下, 你可能需要更具体地了解要针对哪个框架配置文件。

自动添加绑定重定向

当安装具有强名称程序集的包时, NuGet 现在可以检测项目需要将绑定重定向添加到配置文件中的情况, 以便项目自动编译和添加它们。"David Ebbo 的博客文章系列" 中的第3部分 "NuGet 版本管理" "[通过绑定重定向](#)" 包含此功能的更多详细信息。

指定框架程序集引用 (GAC)

在某些情况下, 包可能依赖于 .NET Framework 中的程序集。严格地说, 包的使用者并非始终需要引用框架程序集。但在某些情况下, 这一点很重要, 例如当开发人员需要针对该程序集中的类型编写代码时, 才使用您的包。新的 `frameworkAssemblies` 元素 (metadata 元素的子元素) 允许您指定一组指向 GAC 中的框架程序集的 `frameworkAssembly` 元素。请注意框架程序集的重点。这些程序集不包含在包中, 因为它们被假定为在每台计算机上作为 .NET Framework 的一部分。下表列出了 `frameworkAssembly` 元素的特性。

ATTRIBUTE	DEFINITION
<code>assemblyName</code>	必需。程序集的名称, 例如 <code>System.Net</code> 。
<code>targetFramework</code>	可选。允许指定此框架程序集应用到的框架和配置文件名称 (或别名), 例如 "net40" 或 "sl4"。使用的格式与 支持多个目标框架 中所述的格式相同。

```
<frameworkAssemblies>
  <frameworkAssembly assemblyName="System.ComponentModel.DataAnnotations" targetFramework="net40" />
  <frameworkAssembly assemblyName="System.ServiceModel" targetFramework="net40" />
</frameworkAssemblies>
```

现在, `nuget.exe` 可以存储 API 密钥凭据

使用 nuget.exe 命令行工具时, 现在可以使用 SetApiKey 命令来存储 API 密钥。这样, 每次推送包时都无需指定它。

有关使用 nuget.exe 保存 API 密钥的详细信息, 请阅读[有关发布包的文档](#)。

包资源管理器

包资源管理器已更新为支持 NuGet 1.2。有关详细信息, 请参阅[包资源管理器发行说明](#)。

其他功能/修补程序

之前的列表是我们实现的许多功能和修复的 bug 中最明显的一项。毕竟, 我们在此版本中实现/修复了[59 工作项](#)。

已知问题

- **1.2 包不兼容**: 使用最新版本的命令行工具生成的包, nuget.exe (> 1.2) 将不适用于 NuGet 和外接程序的较旧版本(例如1.1)。如果遇到一条错误消息, 指出不兼容的架构, 则会遇到此错误。请将 NuGet 更新到最新版本。
- **NuGet 服务器不兼容性**: 如果你要使用 nuget.exe 服务器项目托管内部 nuget 源, 则需要使用最新版本的 nuget 更新该项目。
- **签名不匹配错误**: 如果在升级过程中遇到与签名不匹配有关的消息, 则需要先卸载 NuGet, 然后再进行安装。这会在我们的 "[已知问题](#)" 页中列出, 其中提供了更多详细信息。此问题只会影响那些运行 Visual Studio 2010 SP1 的版本, 并且安装的 NuGet 1.0 版本未正确签名。此版本仅在 CodePlex 网站中提供一小段时间, 因此, 此问题不会影响太多人。

NuGet 1.0 和1.1 发行说明

2020/1/8 • [Edit Online](#)

NuGet 1.2 发行说明

NuGet 1.0 于2011年1月13日发布。NuGet 1.1 于2011年2月12日发布。

概述

本文档包含根据主要预览版进行分组的各种版本的 NuGet 1.0 发行说明。

NuGet 包含以下组件：

- *NuGet .vsix**, 其中包括：
 - 在 Visual Studio 中的 "添加库包" 对话框中, 用于浏览和安装包。
 - Visual Studio 中的包管理器控制台* 基于 Powershell 的控制台。
- 用于创建(最终发布)包的*NuGet 命令行工具**工具。

NuGet 工具 Visual Studio 扩展(*nugget*)要求：

- Visual Studio 2010 或 Visual Web Developer 2010 Express。

NuGet 命令行工具需要：

- .NET Framework 版本4

安装

使用此[最新版本](#)：

- 首先卸载旧版本。需要以管理员身份运行 VS 才能执行此操作。
- 删除所有现有源。
- 添加指向 <https://go.microsoft.com/fwlink/?LinkId=206669>的新源。

NuGet 1.1

[可在此处找到](#)此版本中已修复问题的列表

NuGet 1.0 RTM

自 RC 以来, 为 RTM 修复了一个问题。

- [问题474:删除包会影响解决方案中的所有项目](#)

候选发布版本

自 CTP 2 起, 此候选发布版中的更改如下所述。请访问问题跟踪程序, 查看完整的错误列表。

- [从控制台更新包不会更新依赖项。](#)
- [添加包提取 bin not package reference \(CTP1\)](#)
- [更新包会保留损坏的引用](#)
- [获取包-对话框中的更新失败, 或在控制台中选择 "全部" 聚合源时失败](#)
- [获取包验证错误](#)

- 当无法通过 "添加包" 对话框安装包时警告用户
- 获取包-更新大量包时引发更新
- 当错误编写 nuspec 文件时改进错误处理
- NuGet 包忽略指定文件
- 删除第二个到最后一个包源, 然后单击 "下移" 崩溃与
- 安装包时删除程序集引用
- 打开设置对话框时的 InvalidOperationException
- 包源在包管理器控制台中的访问密钥无效
- NuGet VS 设置对话框访问键将焦点放在错误字段上
- 包 ID intellisense 不应查询太多项目
- 将包添加到项目名称中包含点字符的项目失败
- Nuspec 中指定文件的问题
- 使用较新版本时应注册正确的官方源
- 标记应使用空格而不是#
- IPackageMetadata 缺少某些有用信息
- 向对话添加 "报告滥用行为" 链接
- 在 Visual Studio 中使用 App_Data 解压缩包中断
- 实现标记
- PackageBuilder 允许没有要创建的依赖项的空包
- 为包添加所有者字段
- 更新 VSIX 清单以表示 NuGet 包管理器, 而不是 VSIX 工具
- 选择 "所有源" 时, "获取包" 命令会引发错误
- 允许在 "选项" 对话框中排序包源
- 更新-包不会删除较旧的版本
- 实现依赖项的版本范围规范
- 单击 "添加新包" 时 Visual Studio 崩溃
- 显示 "添加包" 对话框中的下载和分级
- 对话框中的包源之间的更改不会更新活动源
- 删除程序包管理器控制台窗口的键绑定
- 安装包未被识别为 cmdlet 的名称 ..。
- 从本地源安装包未解析常规源上的依赖项
- RemoveDependencies 应跳过仍在使用的依赖项
- 如果取消页面导航, 则当原始页面请求返回时, 用户无法导航到其他页面
- 调查 NuPack 的性能, 以便为具有大量包的源提供服务。
- 第二次筛选包时, 它将使用 "新" 包源, 而不是以前选择的源。
- 选择对话框上的 "联机" 选项卡时, 应选择默认的包源。
- 默认情况下, 列表包应显示已安装的包
- 程序集引用 HintPaths
- 打开程序包管理器控制台时出现异常
- 控制台 intellisense 下载整个源
- "Default" 包源应重命名为 "Active"
- 包源 UI: 如果名称/源字段非空, 请按 "确定" 以添加新源
- 当已安装的包的数量较大时, 对话框变慢
- 支持强名称程序集的绑定重定向
- 添加包引用 ..。用于包含包源的下拉的 UI
- NuPack 需要支持配置文件名称的配置转换 agnoscitally

- 允许在 NuPack 中替代 BasePath
- 包源回退行为
- GUI 崩溃
- 添加排序选项以添加包对话框
- 用于清除程序包管理器控制台的快捷键
- PowerConsole 导致 NuPack 控制台失败
- 控制台和 "添加包" 对话框应在请求中设置用户代理
- 在生成中设置 VSIX 和 NuPack 的版本号。
- 从 -? 隐藏公共 PowerShell 参数
- 为控制台命令添加详细帮助
- "添加包" 对话框应该允许选择当前包源
- 将 NuPack 类移到不同的命名空间
- 向 cmdlet 添加帮助
- 下载包后验证源中的哈希

CTP 2

下面是 CTP 2 中最重要的更改：

- 将包源从 ATOM 切换到 OData 服务终结点：如果升级到 CTP2 版本的 NuGet，请确保将以下 URL 添加为包源：
`https://feed.nuget.org/ctp2/odata/v1/`。
- 已将添加包命令重命名为安装包。
- 已更新 `.nuspec` 格式。`.nuspec` 格式现在包含用于指定 32x32 png 图标的 `iconUrl` 字段，该图标将显示在 "添加包" 对话框中。因此，请确保将其设置为区分包。`.nuspec` 格式还包含新的 `projectUrl` 字段，可用于指向提供有关包的详细信息的网页。

此生成不适用于旧的 `.nupkg` 文件。如果收到空引用异常，则使用的是旧的 `.nupkg` 文件，需要使用更新后的 [NuGet 命令行工具](#) 重新生成它。

下面列出了针对 NuGet CTP 2 修复的功能和 bug（不包括用于次代码清理等的错误）。

- 为程序集指定 TargetFramework 时，解压缩包程序集时出错。
- 使 NuPack 控制台窗口更易发现
- ILMerge nupack 版本
- 更好的错误/异常处理
- [Nupack]：PackageManager 应适当处理与源相关的错误
- 需要控制台的新图标
- 在对话框中本地化字符串
- NuPack 在内存中缓存下载的 NuPack 文件
- NuPack 控制台：更改显示控制台的默认快捷方式
- ProjectSystem 应支持通用属性的默认值
- 在仅包含一个 nuspec 文件的文件夹中运行 nupack 应使用该 nuspec
- 即使没有加载任何项目/解决方案，"项目" 菜单也会显示
- 在基本代码的干净克隆上生成 .cmd 失败
- 更新可用功能
- 对话框：通过对话框添加包将删除控制台中的提示
- 通过单击 "安装" 添加包的速度通常较慢，无可视反馈
- 无法发现哪些已安装的包有更新。
- 无法在对话框中更新已安装的包。
- 无法在对话框中卸载已安装的包

- 添加包引用 “...” 出现在已安装引用的上下文菜单上
- 从控制台更新包后，它会将旧版本和新版本显示为已安装
- 使用对话框时，控制台中的活动会在使用后消失
- 清除 nupack 中的命令行分析
- 将友好名称添加到包源
- 要支持包括包图标的 nuspec
- 源 UI 不允许复制 URL
- 更好的删除包错误处理。
- 在控制台窗口中键入依赖于游标焦点
- 错误消息的外观
- 未安装的包的删除包性能错误
- 如果没有包源，则删除包会失败
- 当包源不可用时，删除包失败
- 将标题添加到包元数据和源。
- 将-Source 参数添加回添加包
- 列表-包应具有-Source 参数
- 更新 NuPack 以要求 NuPack 用户代理下载包
- “许可证接受”对话框必须列出所有需要接受的依赖项的许可证
- 在源中引发包时记录错误
- NuPack 不应允许空的 <licenseurl> 元素
- 重命名列表-将包添加到包，将包添加到安装包，将包删除到卸载包
- 使用解决方案导航器中的“添加包引用”菜单项导致 Visual Studio 崩溃
- “可用的包源”标签缺少冒号
- 使 nuspec xml 元素大小写一致大小写
- NuPack VSIX 的清单需要启用 “admin” 位
- 如果运行不带源的列表包，则会收到空引用错误
- nuget.exe: 指定目标路径
- 在 WinXP 上打开包管理控制台时出现 Powershell 错误
- 尝试加载包列表时 VS 崩溃
- 允许元包(无文件，仅依赖项)
- 将 Powershell 脚本转换为 Powershell 2.0 模块
- 指定 target 后，PathResolver 应丢弃前面的通配符字符的路径部分
- 无依赖项
- 安装 Elmah 时出错
- <configsections>，配置转换不能正常工作
- 无法检索变量 "\$global:projectCache"，因为它尚未设置
- 添加用于创建 NuPack 包的 MSBuild 任务
- 列表-包需要支持搜索/筛选
- 如果包作者提供许可证 URL，则始终显示许可证的链接
- 使用删除包偶尔出现“拒绝访问”异常
- 单元测试失败：InvalidPackagesExcludedFromFeedItems & CreatingFeedConvertsPackagesToAtomEntries
- 如果找不到针对特定 framework 版本，则允许使用回退/默认文件集
- 添加包引用 .. 。 UI 无法删除包
- 卸载一个或多个项目时添加包引用崩溃 studio
- Config 转换似乎在 web.config 文件中不起作用
- init.ps1 未在自定义包上触发

- 将路径添加到 feedlist 时， 默认按钮设置为 "确定"，因此，如果我按 ENTER 自动关闭
- 尝试卸载依赖项将崩溃，并在行中尝试2次
- 在 "添加包" 对话框中显示项目 URL
- 默认情况下，已安装包的 "添加包" 对话框
- 更改 "添加包" 对话框菜单项。
- 重命名命名空间和程序集
- 将 NuPack 项目重命名为 NuGet
- 将以下文本添加到依赖项列表下
- 在 "许可证接受" 对话框中更改许可证接受文本
- 更改包列表上方的 "许可证接受" 对话框中的文本
- OData 不适用于 fwlink URL
- 包管理器 UI:通过积极缓存用于分页的包计数
- NuPack/NuGet-> 程序包管理器控制台错误
- "添加包" 对话框显示已安装包的许可证接受情况

CTP 1

下面列出了针对 NuGet CTP 1 修复的功能和 bug。

- 包扩展应重命名为。nupack
- 将包文件移动到文件夹
- 合并安装 & 添加 PS 命令
- 创建谓词的别名-名词 cmdlet
- 在 VS 中切换解决方案时，NuPack 混淆
- 默认情况下，应隐藏 "包" 解决方案文件夹
- 添加对内容项中的标记替换的支持。
- NuPack 应使用 Register-packagesource API
- [Nupack]: PackageManager 在安装包之前将它们标记为已安装
- 从解决方案中删除默认项目仍会将已删除的项目显示为默认项目
- 新包失败，出现 "无法为指定的 URI 添加部分，因为它已在包中"。
- 从 Visual Studio GUI 中删除 "NuPack" 字符串
- 将 Apache 标头添加到版权所有 .txt 文件
- 删除 Register-packagesource 命令
- 加载配置文件时程序包管理器不可用引发异常
- init.ps1, install.ps1 和 uninstall 需要接收附加状态
- 将控制台和 GUI 包合并到一个包中
- 如果应用于不在根位置的 XML，则 Xml 转换逻辑不起作用
- 管理包源设置对话框未更新 NuPack 控制台
- NuPack 控制台 UI:将 "包源" 下拉列表重命名为 "包源"
- NuPack 控制台选项:将 "存储库 UI" 重命名为与 NuPack 控制台一致
- 对于从 IIS 或 URL 打开的网站，添加包失败
- 程序包管理器源不适用于 FwLink
- 设置默认包源
- 如果在选项中添加包源，则仅提供一个源时，假定它是默认值。
- 对话框 UI 显示虚假的 "最近的" 包
- 选项:单击 "取消" 不会取消更改
- 添加包引用对话框搜索应该不区分大小写
- 修复 AssemblyInfo.cs 文件中的公司元数据

- VSIX 的版本号
- 删除包: 使用 -? 显示帮助两次
- 执行项目级包的安装/卸载包
- 服务器无法在验证失败时创建源 nupack
- 需要替换 NuPack 图标
- NTLM http 代理不对包源进行身份验证。
- 对话框并不总是在 VS 窗口中居中
- 包中的许多字段未在对话框中填充
- 对话框 UI 不显示作者姓名
- 原因-删除包的版本
- 删 除对话框 UI 上的 "最近" 选项卡
- 当在对话框 UI 至少打开一次后右键单击解决方案文件夹时, VS 崩溃。
- 将列表包的本地参数更改为已安装
- 将包重命名为 NuPack
- 控制台强制将光标移到行尾
- 删 除-包 intellisense 已中断
- 向 nuspec 和源添加 RequireLicenseAcceptance 标志
- 将 LicenseUrl 添加到 nuspec 格式和包源
- 单击 "安装" 需要接受的包应显示 "接受" 对话框
- 将免责声明文本添加到 "添加包" 对话框
- 首次运行包控制台时添加声明
- 在控制台中安装包后显示声明
- 将 nupack 扩展名重命名为 nupkg。

NuGet 常见问题

2020/4/8 • [Edit Online](#)

有关 NuGet.org 相关常见问题(如 NuGet.org 帐户问题), 请参阅[NuGet.org 常见问题解答](#)。

运行 NuGet 有哪些要求？

[安装指南](#)中提供了有关 UI 和命令行工具的所有信息。

NuGet 是否支持 Mono？

命令行工具 `nuget.exe` 在 Mono 3.2+ 下生成和运行, 并且可以在 Mono 中创建包。

尽管 `nuget.exe` 在 Windows 上可充分发挥功能, 但用于 Linux 和 OS X 时则存在已知问题。请参阅 GitHub 上的 [Mono 问题](#)。

[图形化客户端](#)以 MonoDevelop 外接程序的形式提供。

如何确定包的内容及其是否适用于应用程序？

了解包的主要来源为 nuget.org(或其他私有源)上的包清单页。nuget.org 上的每个包页面都包含包的说明、其版本历史记录和使用情况统计信息。包页面中的“信息”部分还包含项目网站的链接, 此网站通常提供大量实例和其他文档, 帮助你了解包的使用方法。

有关详细信息, 请参阅[查找和选择包](#)。

Visual Studio 中的 NuGet

[NuGet 在不同 Visual Studio 产品中的受支持情况](#)

- Windows 版 Visual Studio 支持[包管理器 UI](#) 和[包管理器控制台](#)。
- Visual Studio for Mac 具有内置 NuGet 功能, 如[在项目中包括 NuGet 包](#)中所述。
- Visual Studio Code(所有平台)与 NuGet 不存在任何直接集成。请使用 [NuGet CLI](#) 或 [dotnet CLI](#)。
- Azure DevOps 提供[还原 NuGet 包的生成步骤](#)。还可以在[Azure DevOps 上托管私有 NuGet 包源](#)。

如何查看安装的 NuGet 工具的确切版本？

在 Visual Studio 中, 请使用“帮助”>“关于 Microsoft Visual Studio”命令, 然后查看“NuGet 包管理器”旁显示的版本。

或者, 启动“包管理器控制台”(“工具”>“NuGet 包管理器”>“包管理器控制台”), 输入 `$host`, 然后查看包括版本信息在内的 NuGet 的相关信息。

NuGet 支持哪些编程语言？

NuGet 旨在将 .NET 库引入项目, 通常适用于 .NET 语言。由于 NuGet 还在某些项目类型中支持 MSBuild 和 Visual Studio 自动化, 因此它也在不同程度上支持其他项目和语言。

最新版 NuGet 支持 C#、Visual Basic、F#、WiX 和 C++。

NuGet 支持哪些项目模板？

NuGet 对多种项目模板提供完整支持, 如 Windows、Web、Cloud、SharePoint 和 Wix 等。

如何更新 Visual Studio 模板中的包？

在包管理器 UI, 转到“更新”选项卡, 选择“全部更新”; 或者使用包管理控制台中的 [Update-Package](#) 命令。

若要更新模板本身，则需要手动更新模板存储库。请参阅与该主题相关的 [Xavier Decoster 博客](#)。请注意，此操作的风险由自己承担，因为如果所有依赖项的最新版本不相互兼容，手动更新可能会损坏该模板。

是否可以在 Visual Studio 之外使用 NuGet？

可以，NuGet 可直接在命令行中使用。请参阅[安装指南](#)和[CLI 参考](#)。

NuGet 命令行

如何获取最新版本的 NuGet 命令行工具？

请参阅[安装指南](#)。若要检查当前安装的工具版本，请使用 `nuget help`。

`nuget.exe` 的许可证是什么？

可以根据 MIT 许可条款重新分发 `nuget.exe`。你有责任更新和维护你选择重新分发的所有 `nuget.exe` 副本。

是否可以扩展 NuGet 命令行工具？

是的，可以向 `nuget.exe` 添加自定义命令，如 [Rob Reynold 博客](#) 中所述。

NuGet 包管理器控制台 (Windows 版 Visual Studio)

如何在包管理器控制台中访问 DTE 对象？

Visual Studio 自动化对象模型中的顶层对象称为 DTE (开发工具环境) 对象。控制台通过名为 `$DTE` 的变量提供此对象。有关详细信息，请参阅 Visual Studio 扩展性文档中的[自动化模型概述](#)。

尝试将 `$DTE` 变量强制转换为 `DTE2` 类型时出现错误：无法将“`EnvDTE.DTEClass`”类型的“`EnvDTE.DTEClass`”值转换为类型“`EnvDTE80.DTE2`”。为什么会这样？

这是 PowerShell 与 COM 对象交互的已知问题。请尝试如下方法：

```
`$dte2 = Get-Interface $dte ([EnvDTE80.DTE2])`
```

`Get-Interface` 是 NuGet PowerShell 主机添加的帮助程序函数。

创建和发布包

如何在源中列出我的包？

请参阅[创建和发布包](#)。

我有面向不同版本的 .NET Framework 的多个版本的库。如何才能生成一个支持此方案的包？

请参阅[支持多个 .NET Framework 版本和配置文件](#)。

如何设置自己的存储库或源？

请参阅[托管包概述](#)。

如何将包批量上传到我的 NuGet 源？

请参阅[批量发布 NuGet 包](#) (jeffhandly.com)。

使用包

项目级包和解决方案级包之间有何差异？

解决方案级包 (NuGet 3.x+) 只需在解决方案上安装一次，随后即可用于该解决方案中的所有项目。项目级包需在使

用它的每个项目中进行安装。解决方案级包还可以安装可从包管理器控制台中调用的新命令。

是否可以在没有 Internet 连接时安装 NuGet 包？

可以，请参阅 Scott Hanselman 的博客文章 [How to access NuGet when nuget.org is down \(or you're on a plane\)](#)(如何在 nuget.org 不可用(或用户在飞机上)时访问 NuGet)(hanselman.com)。

如何在默认包文件夹以外的其他位置安装包？

请使用 `nuget config -set repositoryPath=<path>` 设置 `Nuget.Config` 中的 `repositoryPath` 设置。

如何避免将 NuGet 包文件夹中添加到源代码管理中？

请将 `Nuget.Config` 中的 `disableSourceControlIntegration` 设置为 `true`。此密钥适用于解决方案级别，因此需要添加到 `$(Solutiondir)\.nuget\Nuget.Config` 文件中。从 Visual Studio 中启用包还原将自动创建此文件。

如何关闭包还原？

请参阅[启用和禁用包还原](#)。

安装具有远程依赖项的本地包时，为何会出现“无法解析依赖项”错误？

将本地包安装到项目中时，需要选择“所有”源。这样可以聚合所有源，而不只是一个源。本地存储库用户通常不希望因为公司政策而意外安装远程包，因此才会出现此错误。

如果同一文件夹中有多个项目，如何使用每个项目单独的 packages.config 文件？

对于各个项目位于单独文件夹中的大多数项目，用户不必考虑此问题，因为 NuGet 会识别每个项目中的 `packages.config` 文件。如果使用 NuGet 3.3+ 并且同一文件夹中有多个项目，可将项目名称插入 `packages.config` 文件名(使用 `packages.{project-name}.config` 模式)，然后 NuGet 将使用该文件。

使用 PackageReference 时不必考虑此问题，因为每个项目文件仅包含自己的依赖项列表。

存储库列表中未显示 nuget.org，应如何让其重新显示？

- 将 `https://api.nuget.org/v3/index.json` 添加到源列表；或
- 删除 `%appdata%\nuget\NuGet.Config` (Windows) 或 `~/.nuget/NuGet/NuGet.Config` (Mac/Linux) 并指示 NuGet 重新创建它。

标识项目格式

2020/4/8 • [Edit Online](#)

NuGet 适用于所有 .NET 项目。但是，项目格式（SDK 样式或非 SDK 样式）决定了使用和创建 NuGet 包所需的一些工具和方法。SDK 样式的项目使用 [SDK 属性](#)。确定项目类型非常重要，因为用于使用和创建 NuGet 包的方法和工具都依赖于项目格式。对于非 SDK 样式的项目，方法和工具还取决于项目是否迁移到 [PackageReference](#) 格式。

项目是否为 SDK 样式取决于用于创建项目的方法。下表显示了使用 Visual Studio 2017 及更高版本创建项目时的默认项目格式和相关的 CLI 工具。

项目	项目文件	CLI 工具	说明
.NET Standard	SDK 样式	dotnet CLI	在 Visual Studio 2017 之前创建的项目为非 SDK 样式。使用 nuget.exe CLI。
.NET Core	SDK 样式	dotnet CLI	在 Visual Studio 2017 之前创建的项目为非 SDK 样式。使用 nuget.exe CLI。
.NET Framework	非 SDK 样式	nuget.exe CLI	使用其他方法创建的 .NET Framework 项目可能是 SDK 样式项目。对于这种情况，请改为使用 dotnet CLI 。
迁移的 .NET 项目	非 SDK 样式	要创建包，请使用 msbuild -t:pack 创建包。	要创建包，建议使用 msbuild -t:pack 。否则，使用 dotnet CLI 。迁移的项目是非 SDK 样式的项目。

检查项目格式

如果不确定该项目是否是 SDK 样式的格式，请在项目文件中的 `<Project>` 元素中查找 [SDK 属性](#)（对于 C#，这是 `*.csproj` 文件）。如果存在，则该项目是一个 SDK 样式的项目。

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
  <TargetFramework>netstandard2.0</TargetFramework>
  <Authors>authorname</Authors>
  <PackageId>mypackageid</PackageId>
  <Company>mycompanyname</Company>
</PropertyGroup>

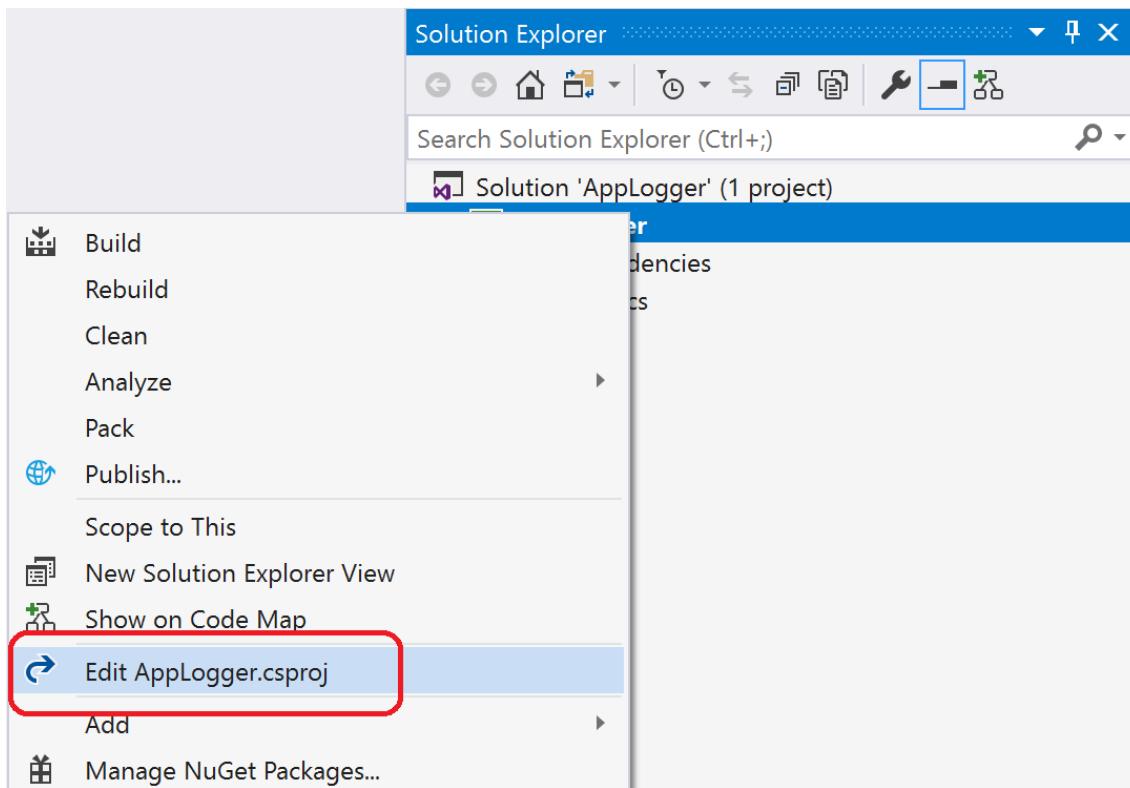
</Project>
```

在 Visual Studio 中检查项目格式

如果在 Visual Studio 中进行操作，可以使用以下方法之一快速检查项目格式：

- 在解决方案资源管理器中右键单击该项目，然后选择“编辑 myprojectname.csproj”。

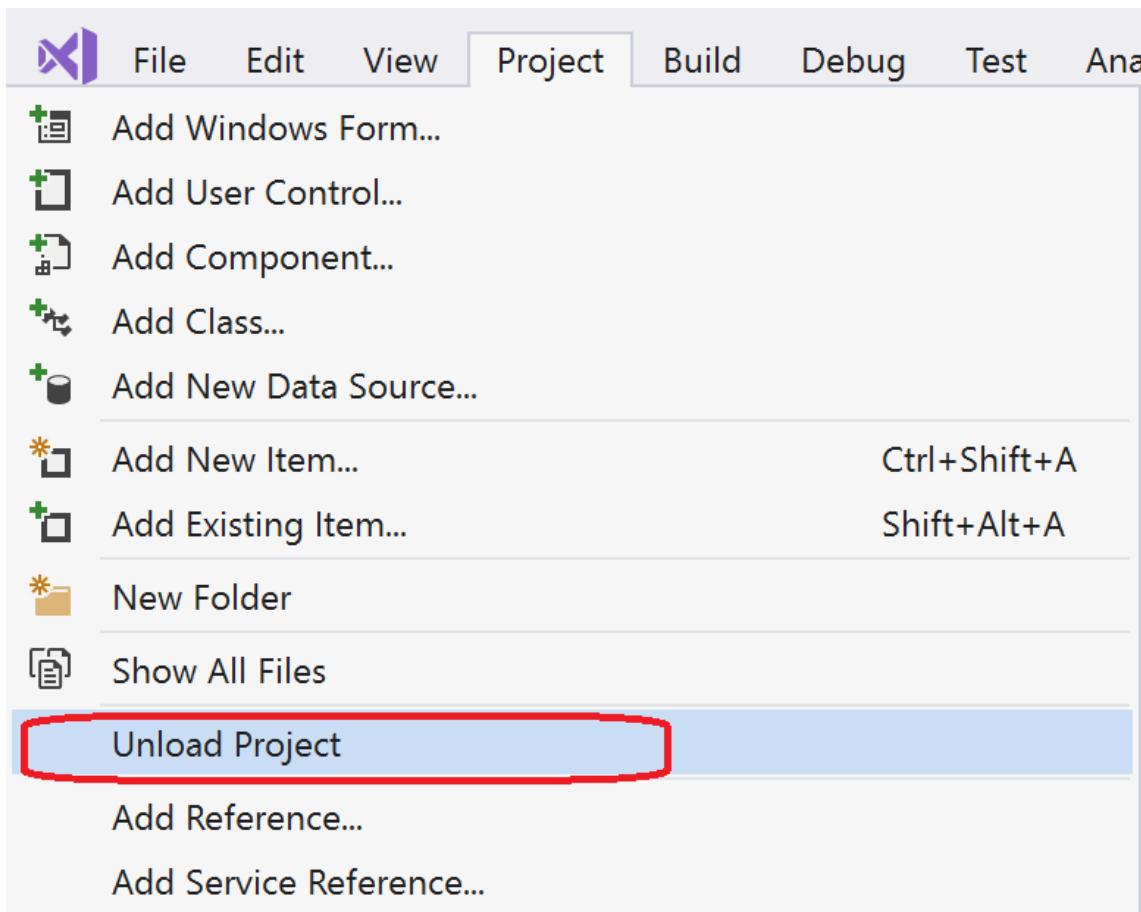
此选项从 Visual Studio 2017 开始仅对使用 SDK 样式属性的项目可用。否则，请使用其他方法。



一个 SDK 样式的项目会在项目文件中显示 [SDK 属性](#)。

- 在“项目”菜单中选择“卸载项目”（或右键单击项目并选择“卸载项目”）。

该项目将不在项目文件中包含 SDK 属性。它不是 SDK 样式的项目。



然后，右键单击卸载的项目并选择“编辑 myprojectname.csproj”。

另请参阅

- [使用 dotnet CLI 创建 .NET Standard 包](#)
- [使用 Visual Studio 创建 .NET Standard 包](#)
- [创建并发布 .NET Framework 包 \(Visual Studio\)](#)
- [作为 MSBuild 目标的 NuGet 包和还原](#)

NuGet.org 概述

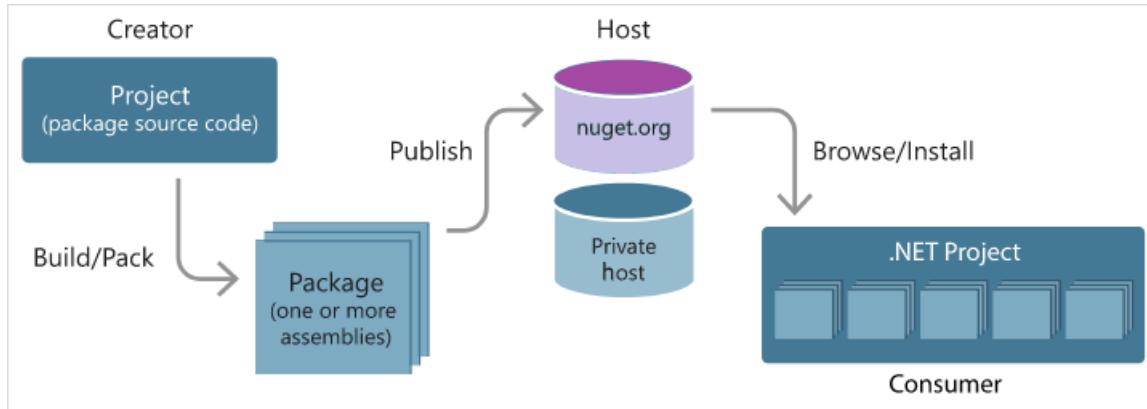
2020/4/8 • [Edit Online](#)

NuGet.org 是 NuGet 包的公用主机，每天都有数百万 .NET 和 .NET Core 开发人员使用它。

NuGet.org 在 NuGet 生态系统中的角色

作为公用主机角色时，NuGet.org 自身负责在 nuget.org 中维护包含 100,000 多个唯一包的中央存储库。NuGet.org 不是唯一可能的包主机。NuGet 技术还支持在云中（如在 Azure DevOps 上）、在私有网络中或者甚至直接在本地文件系统以私密方式托管包。如果对其他主机或承载选项感兴趣，请参阅[承载自己的 NuGet 源](#)。

与 NuGet 包的任何主机一样，NuGet.org 充当包创建者和包消费者之间的连接点。创建者生成有用的 NuGet 包并将其发布。然后，使用者可以在可访问的主机上搜索有用且兼容的包，下载包并将其包含在项目中。在项目中安装包后，包的 API 将可用于其余项目代码。



帐户

要在 NuGet.org 上发布包，首先要创建一个[个人\(用户\)帐户](#)。这将成为你在 NuGet.org 上的标识。

通过 NuGet.org 还可创建[组织帐户](#)。组织帐户的成员可以是一个或多个个人帐户。成员可以管理一组包，同时保持所有权的单一标识。通过你的个人帐户，你可以成为任意数量组织的成员。

包可以属于组织帐户，就像它可以属于个人帐户一样。包使用者看不到个人帐户或组织帐户之间的任何差异：两者都显示为包 `owners`。

API 密钥

具有要发布的 NuGet 包 (.nupkg 文件) 后，可以使用 nuget.exe CLI 或 dotnet.exe CLI 以及从 NuGet.org 获得的 [API 密钥](#) 将其发布到 NuGet.org。

在[发布包](#)时，请在 CLI 命令中包含 API 密钥值。

ID 前缀

发布包时，可以通过[保留 ID 前缀](#)来保留和保护标识。安装包时，包使用者会收到附加信息，其中指出他们使用的包在标识属性中并不具有欺骗性。

适用于 NuGet.org 的 API 终结点

若要将 NuGet.org 用作 NuGet 客户端的包存储库，应使用以下 V3 API 终结点：

<https://api.nuget.org/v3/index.json>

较旧版本的客户端仍然可以使用 V2 协议来访问 NuGet.org。但是, 请注意, NuGet 客户端 3.0 或更高版本在使用 V2 协议时将导致服务的速度更慢且不太可靠:

<https://www.nuget.org/api/v2> (**V2 protocol 已弃用!**)