.NET 标准介绍

2016-10-04 葡萄城控件技术译 [dotNET跨平台](javascript:void(0);) dotNET跨平台

**dotNET跨平台**

微信号 opendotnet

功能介绍 在这里你可以谈微软.NET,Mono的跨平台开发技术，也可以谈谈其他的跨平台技术。  
在这里可以让你的.NET项目有新的思路，不局限于微软的技术栈，横跨Windows，Linux 主流平台

本文介绍如何使用 .NET 标准，更容易地实现向 .NET Core 迁移。文中会讨论计划包含的 APIs，跨构架兼容性如何工作以及这对 .NET Core 意味着什么。

如果你对细节感兴趣，这篇文章正是为你准备的；如果你没有那么多时间或者对细节并不感兴趣，你可以仅仅只阅读 TL;DR 章节。

**TL;DR**

对于跨平台的 .NET 开发者来说，.NET 标准解决了编码共享的问题。.NET 标准带来了所有你所需要的和期待的，跨环境的 APIs：桌面应用，移动应用/游戏和云服务。

* .NET 标准是一组所有 .NET 平台必须实现的 APIs。这就统一了 .NET 平台并防止平台在未来分离。
* .NET 2.0 标准将由 .NET 框架，.NET Core 和 Xamarin 来实现。对于 .NET Core，这将会增加许多现有的被期待的 APIs。
* 对于 .NET Framework 的二进制文件，.NET 2.0 标准包含了一个兼容性的功能，显著地增加库类集，这个你可以参考 .NET 标准库。
* .NET 标准将取代便携式类库（PCLs）作为构建多平台 .NET 库的工具集。
* 你可以看到 .NET APIs 标准定义在 dotnet/standard GitHub 上，同时这也以在 GitHub 上获得。

**为什么我们需要一个标准？**

在文章中的介绍 .NET Core 部分会详细解释，.NET 平台已经分离开很多年了。

一方面，这其实是一个很好的事情。它允许根据需求来裁剪 .NET，这是一个单平台做不到的。例如，.NET Compact Framework 的创建是为了适应2000年代手机发展的足迹。今天同样也是这样：统一集合运行在20多个平台上。对于任何期待的技术来说，能够分离和定制是一项很重要的能力。

但另一方面，平台分离也给 .NET 多平台的开发者编写代码带来了很大的问题，因为没有一个统一的库类来使用：

目前 .NET 有三种主要的风格，这意味着你必须要掌握三种不同的基类库，来编写跨三种风格的代码。比起 .NET 被创立之初时，现状已变得更加的多样。微软或者其它人将会创建新的 .NET 风格，来支持新的操作系统或者裁剪 .NET 来适应特殊设备的兼容。

**.NET标准产生的原因：**

对于开发者来说，这意味着他们只需要掌握一个基础类库。针对 .NET 标准的库类，将能够在所有的 .NET 平台上运行。平台提供者不需要再猜测他们需要提供哪些 APIs ，来对应 NuGet 上获取的库类。

* **应用**。在应用程序方面你不需要直接使用 .NET 标准。不过，你还会间接地受益。首先，.NET 标准会确保所有的 .NET 平台共享具有相同 APIs 的基础类库。一旦你学会了如何在桌面应用程序中使用它，你知道如何在移动应用程序或云服务中使用它。其次，.NET 标准中的大部分类库会变得随处可见，这意味着基础层的一致性也将适用于更大规模的 .NET 库生态系统。
* **便携式类库**。让我们来和便携式类库（PCLs）如何工作做个对比。使用 PCLs，你可以选择你想要运行的平台，同时你也可以选用的 APIs 呈现给你的工具。因此，当工具帮助生成了能在多平台上运行的二进制文件时，它也会迫使你去考虑不同的基础库类。使用 .NET 标准，你有一个单一的基础类库。库类中的所有，将会在全部的 .NET 平台中获得支持--那些当前的以及未来的。另一个重要方面是，.NET 标准的 APIs 的可用性是可预测的：高版本意味着更多的 APIs。使用 PCLs，以下情况不是必然的：一组可用的 APIs ，这将是所选择的平台之间的交集。
* **一致性的APIs** 。如果你比较. NET 框架、.NET Core和Xamarin/Mono，你会发现 .NET Core 提供了最小的 APIs 界面（不含特定操作系统的 APIs ）。第一个问题是基础性的 APIs 的可用性有大幅差异（如 networking 和加密的 APIs ）。第二个问题 .NET Core 的引入在 APIs 方面有很大的不同，尤其是在反应方面。这两个问题是将代码移植到 .NET Core 上很难的主要原因。通过创建 .NET 标准，我们正在设定具有跨所有 .NET 平台的一致性 APIs 的要求，这也包括可用性和 APIs 的形式。
* **版本控制和工具**。正如我在介绍 .NET Core 时提到的，我们的目标是为了奠定一个便携式 .NET 平台的基础，这样就可以统一 APIs 的信息和实现。我们预计它会成为下一个便携式库类的版本。不幸的是，它没有以一个巨大的工具体验作为结果。由于我们的目标是代表任何 .NET 平台，我们不得不将它分解成更小的 NuGet 包。如果所有这些组件都可以部署到应用程序上，那么将会工作得很好，这样你也可以独立的更新他们。但是，如果针对的是抽象的规范，例如像 PCLs 或者 .NET 标准，这将不会工作的很好。因为有一套非常具体的组合版本，来确保能够在正确的平台上运行。为了避免这个问题，我们定义了 .NET 标准作为一个单独的 NuGet 包。因为它仅表示必需那组 APIs，也没有将它继续分解的必要，因为所有 .NET 平台必须以所有的方式支持它。唯一重要的方面是它的版本，它扮演的像一个 APIs 的等级：较高的版本，有更多的 APIs ，但较低的版本，更多的 .NET 平台已经实现了它。

总而言之，我们需要 .NET 标准，原因有二：

1. **驱动力的一致性。**我们希望拥有一套需求一致的，在所有的 .NET 平台上都实现了的 APIs ，来获得 .NET 库的生态系统的访问。
2. **跨平台工具的基础**。我们希望有一个简单的工具体验，允许通过选择一个单独的版本号，来制定所有 .NET 平台的公共的目标。

**.NET 2.0 标准有哪些新功能？**

当我们发布 .NET Core 1.0 时，我们还推出了 .NET 标准。还有很多个 .NET 标准的版本，来表示跨当前所有平台的 APIs 的可用性。下表显示了一个现有的平台的版本，与 .NET 标准的一个给定版本的兼容：

| .NET平台 | .NET标准 | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1.0 | 1.1 | 1.2 | 1.3 | 1.4 | 1.5 | 1.6 | 2.0 |
| .NET Core | → | → | → | → | → | → | 1.0 | vNext |
| .NET框架 | → | 4.5 | 4.5.1 | 4.6 | 4.6.1 | 4.6.2 | vNext | 4.6.1 |
| Xamarin.iOS | → | → | → | → | → | → | → | vNext |
| Xamarin.Android | → | → | → | → | → | → | → | vNext |
| 通用的Windows平台 | → | → | → | → | 10.0 | → | → | vNext |
| 视窗 | → | 8 | 8.1 |  |  |  |  |  |
| Windows手机 | → | → | 8.1 |  |  |  |  |  |
| Windows Phone的Silverlight的 | 8 |  |  |  |  |  |  |  |

箭头表示，该平台支持更高版本的 .NET 标准。例如，.NET Core 1.0 支持 .NET 标准1.6版，这就是为什么有指向的较低版本1.0 - 1.5的右箭头。

你可以借此来了解 .NET 标准的最高版本，以便根据你计划运行的 .NET 平台，更有针对性的选择。举例来说，如果你想在 .NET Framework  4.5 和 .NET Core 1.0 上运行，你可以最多可以选择 .NET 标准1.1。

您还可以看到哪些平台将支持 .NET 2.0 标准：

* 我们将会更新 .NET Core，Xamarin 和 UWP 的版本，这样将会添加所有支持 .NET 2.0 标准的必要的 APIs。
* .NET Framework 4.6.1已经实现了所有的 APIs，这也是 .NET 2.0 标准的一部分。需要注意的是这个版本出现了两次; 后来我将介绍这是为什么以及它是如何工作的。

.NET 标准也与便携式类库兼容。从 PCLs 属性到 .NET 标准版本的映射列在我们的文档。

从一个目标 .NET 标准库类中，你就可以引用两个其它的类库：

* **.NET 标准**，如果它们的版本是低于或等于你的目标版本。
* **便携式类库**，如果它们的属性可以映射到一个版本低于或等于你的目标版本的 .NET 标准版本。

从图形上看，就像这样：

不幸的是，NuGet 上 PCLs 和 .NET 标准的采用，并不是那么高。这是一个给定的目标包中有多少次发生在 NuGet.org：

| 目标 | 出现次数 |
| --- | --- |
| .NET框架 | 46894 |
| .NET标准 | 1886 |
| Portable | 4501 |

正如你所看到的，这是很清楚， NuGet 上的绝大多数的类库，都是以 .NET 框架为目标的。但是，我们也知道那些库类中的很大一部分，都只使用了我们在 .NET2.0 标准中提供的 APIs。

在 .NET 2.0 标准中，我们将有可能使用以 .NET 标准为目标的库类，同样也可以通过兼容性的功能，实现现有的 .NET框架二进制文件的引用：

当然，这只是在 .NET 框架库使用 .NET 标准中可用的 APIs 时，才会起作用。这就是为什么这不是首选方式，来创建跨不同的 .NET 平台使用的库。然而，这种兼容性功能提供了一个桥梁，使你可以转换库类到 .NET 标准，而不必放弃那些没有被转换，却还在引用的现有库类。

如果您想了解更多关于兼容性功能是如何工作的，请看的规范 .NET 2.0 标准。

**.NET 2.0 标准的重大更改：添加的 .NET Framework 4.6.1 的兼容性**

一个标准只有当平台实现时才有用。与此同时，我们也希望使得 .NET 标准有用并有意义，这是因为对以标准为目标的库类来说，这些 APIs 是可以用的：

* **.NET 框架**。.NET FrameWork 4.6.1具有最高的采用，这使得它成为了最吸引人的 .NET 框架版本。因此，我们要确保它可以在 .NET 2.0 标准中实现。
* **.NET Core**。如上所述，.NET Core 拥有小得多的 APIs 集，对比于 .NET 框架和 Xamarin。支持 .NET 2.0 标准意味着我们需要显著增加界面。由于 .NET Core 不与操作系统配套，但与应用程序配套，所以支持 .NET 2.0 标准只需要更新 SDK 和 NuGet 包。
* **Xamarin**。Xamarin 已经支持大部分的 APIs，这些 APIs 也是 .NET 标准的一部分。更新的工作原理类似于 .NET Core --我们希望可以更新到包含所有 APIs 的 Xamarin。事实上，APIs 中的绝大多数，已经加入到稳定的 Cycle 8 release/Mono 4.6.0 中了。

下面列出了支持.NET标准的 .NET Framework 版本：

|  | 1.4 | 1.5 | 1.6 | 2.0 |
| --- | --- | --- | --- | --- |
| .NET Framework | 4.6.1 | 4.6.2 | vNext | 4.6.1 |

为了让 .NET FrameWork  4.6.1 支持 .NET 2.0 标准，我们不得不删除所有的 .NET 标准中引入的 APIs。紧随普通版本规则之后，可以预料，.NET 2.0 标准将只会被一个更新的 .NET Framework 所支持。考虑到 .NET FrameWork  4.6.2 的最新版本只支持 .NET 1.5 标准。这意味着，针对 .NET 2.0 标准编制的库，不会运行在绝大多数 .NET 框架的安装上。

你可能想知道这一决定带来的影响是什么。我们针对 .NET1.5 标准和更高的版本，使用所有的 APIs 在 NuGet.org 上对所有的包，进行分析。在写这篇文章的时候，只发现了6个非微软的包做到这一点。我们会接触到这些包的拥有者，并与他们合作，以解决这个问题。从看它们的用来看途，很显然，他们的调用能够被 .NET 2.0 标准的 APIs 所代替。

为了使这些包能支持 .NET 1.5 标准，1.6 和 2.0，他们需要针对这些版本进行交叉编译。或者，他们可以选择对 .NET2.0 标准或更高版本给予广泛平台的支持。

**.NET标准有什么？**

为了决定哪些 APIs 会成为 .NET 标准的一部分，我们使用下面的方法：

* **输入**。我们是以所有 .NET 框架和 Xamarin 中可用的 APIs 开始的。
* **评估**。我们所有的这些 APIs 分为两个部分：

1. **必需**。我们希望所有的平台都提供，并且我们相信的可以实现跨平台的 APIs，我们将此视为必需。
2. **可选**。特殊平台或者属于传统技术的一部分的 APIs，我们将此视为可选。

可选 APIs 不是 .NET 标准的一部分，但可作为单独的 NuGet 包。我们尝试针对 .NET 标准，作为库类创建他们，以至于他们的实现可以根据平台的不同而定制，但对于平台特殊的 APIs，这不总是可行的。

为了使一些 APIs 可选，我们不得不删除这是必需 APIs 集的一部分其他 APIs。例如，我们决定在 .NET 标准中具有 AppDomain，而代码访问安全性（CAS）是一个传统部件。这就要求我们删除 AppDomain 中使用 CAS 类型包含的所有成员，如创建域中的重载。

.NET 标准 APIs 集，以及我们可选 APIs 的提议，将会被 .NET 标准的审查机构审阅。

这里是 .NET 2.0 标准的 APIs 界面的高度概括：

如果你想看看 .NET 2.0 标准特定的 APIs 集，你可以看看 .NET 标准 GitHub 的信息库。请注意，.NET 2.0 标准是一项正在进行的工作，这意味着一些 APIs 可能会增加，而另一些可能会被删除。

**我们还可以使用特定平台的 APIs 吗？**

创建多平台库类的经验中的一个最大挑战，就是避免只有大众化的东西，同时确保你不会意外地创建原本不打算创建的库类。

在 PCLs，我们通过拥有多个配置文件（其中每一个代表着一套平台的交集），解决了这个问题。这样做的好处是，允许你在一组目标之间，最大输出该 APIs 界面。.NET 标准代表了所有 .NET 平台必须要实现的一组 APIs。

这也带来了问题，那就是如何定义那些无法在全部平台上实现的 APIs：

* **运行时特定APIs**。例如，运用反射发出生成和运行代码的能力。因为没有一个 JIT 编译器，所以这是不能在 .NET 平台上起作用的，如 UWP 上的 .NET 原生或 Xamarin 的 iOS 工具链。
* **操作系统特定APIs**。在 .NET 中，我们已经从 Win32 中暴露了许多 APIs，以使他们能够更容易被使用。一个很好的例子就是 Windows 注册表。实现依赖于在其他操作系统上不具备的等同的底层 Win32 APIs 。

对于这些APIs，我们有以下的选项：

1. **使得APIs 不可用**。不能使用不能在跨所有 .NET 平台上工作的 APIs 。
2. **使得APIs 可用，但却丢弃PlatformNotSupportedException**。这意味着不论它们是否是支持或不支持，我们将会公开所有的 APIs 。不支持它们的平台会提供 APIs ，但是会丢弃 PlatformNotSupportedException。
3. **模拟APIs** 。Mono 作为一个 APIs 通过 APIs  .ini 文件，实现了注册表。这虽然对于使用注册表来读取操作系统信息的应用不起作用，但是对于简单的使用注册表，来存储自己状态和用户设置的应用来说，却起了很好的作用。

我们相信，最好的选择是一个组合。正如上面提到的，我们希望 .NET 标准代表一组所有 .NET 平台都要求实现的 APIs。我们要让这个设定合理地实施，同时确保流行的 APIs 也都存在，这样编写跨平台的库会非常简单，直观。

解决只在一些 .NET 平台可用的技术的一般策略是：提供给他们 .NET 标准上的 NuGet 包。所以，如果你创建一个基于 .NET 标准的库时，它会默认不引用这些 APIs 。你必须添加一个 NuGet 包进来。

对于自包含的并且可以整理成独立包的 APIs 来说，这种策略工作得很好。对于单个类型成员不能在所有环境下实现的情况，我们将使用第二和第三种方法：平台必须有这些成员，但他们可以决定丢弃或模仿他们。

让我们看几个例子，了解我们是计划如何模拟它们：

* **注册**。Windows 注册表是一个自包含的控件，将作为一个单独的 NuGet 包被提供。你可以从 .NET Core 中使用它，但它只能在 Windows 上运行。从任何其他操作系统调用 APIs 的注册表，将会导致 PlatformNotSupportedException。你希望适当地保护你的调用或者确保你的代码只运行在 Windows 上。我们正在考虑改善我们的工具，来帮助你检测这些情况。
* **AppDomain**。该 AppDomain 类型有很多的 APIs ，不依赖于创建应用程序域，如获取加载的程序集列表或登记未处理的异常处理。这些 APIs 是整个 .NET 库生态系统中大量使用的。对于这种情况，我们决定添加这种类型到 .NET 标准，让少量的 APIs 来应对平台上应用程序域创建时，不支持抛出的异常要好的多，如 .NET Core。
* **反射发出**。反射发出是合理的自包含。因此，我们计划按照模型进行注册。有一个另外的 APIs 逻辑上依赖于 emit 代码，例如表达式树形的编译或者编译正则表达式的能力。在某些情况下，我们会模仿他们的行为，而在其他情况下，我们会丢弃。

一般情况下，就像你今天做的，你可以通过以特殊 .NET 平台为目标，使用 .NET 标准中还没可用的 APIs 工作。我们正在考虑如何才能改善我们的工具，来帮助特殊平台与位置平台之间迁移地更加流畅，你可以根据你的情境做出最好的选择，不必考虑早先设计的选择。

总结：

* 我们将揭露一些并不适用于所有.NET平台的概念。
* 我们将会让他们成为你必须明确引用的独立的包。
* 在极少数情况下，个别成员可能会抛出异常。

我们的目标是让 .NET 基础标准库尽可能强大的并具有表现力，同时让你了解到你所依赖的技术并不是在任何环境下都起作用。

**.NET Core意味着什么？**

我们设计 .NET Core，是为了它的引用程序集是 .NET 可移植的。这使得它很难增加新的 APIs ，因为在 .NET Core 中添加这些 APIs，取代了决定这些 APIs 是否在任何环境下都可用。更糟的是，由于版本规则，这也意味着我们必须决定哪些 APIs 的组合在老版本中是可用的。

* **带外数据递送**。我们试图通过这些可用 APIs 来解决这个带外数据，这意味着要做出新的可以安置在现有 APIs 顶端的组件。对于技术，这是容易做到的，也是首选的方法，因为这也意味着任何 .NET 开发人员可以使用这些 APIs 并且给我们反馈。对于这些不可改变的集合，我们已经做出了巨大的成功。
* **运行时启示功能**。但是，对于需要运行时工作的特性，这是更难的，因为我们不能只给你一个起作用的 NuGet 包。我们必须给你一种方式来获得一个更新的运行时。这对具有全系统运行时的平台，这个更难。因为对于不同的目的，我们有多种运行时。这是不实际的，一次跨越所有范围去创新。有关 .NET Core 的好处是，这个平台的设计是完全独立的。因此，对于未来，我们就更有可能将此功能用于实验和预览。
* **从.NET Core拆分.NET标准**。为了能够从其他 .NET 平台独立地发展 .NET Core，我们已经从 .NET Core 分析了便携性机制。.NET 标准被定义为满足所有的 .NET 平台的一个独立引用集合。每 .NET 平台使用一套不同的引用程序集，因此可以自由地在他们选择的部分增添新的 APIs。然后我们也可以决定向 .NET 标准添加哪些 APIs，让其成为通用的。

从 .NET Core 中分离便携性，帮助我们加快 .NET Core 的发展，使得新特性的实验更加的容易。除了人工设计位于现存平台顶端的特性，我们可以简单的修改底层，以便支持这个特性。我们还可以将 APIs 添加到逻辑上归属的类型中，而不必担心类型是否已经在其它平台上拓展过。

给 .NET Core 中添加新的 APIs 已经不是一个陈述了，我们对 .NET 标准的目标，是创造 .NET 平台之间的一致性，所以新的类型成员成为标准的一部分，在标准更新时已经被自动考虑了。

**作为一个库类开发者，你可以做些什么？**

作为一个库类开发者，你应该考虑切换到 .NET 标准，因为以多 .NET 平台为目标，它会取代便携式库类。

在 .NET 1.x 标准下，可用的 APIs 集合与 PCLs 非常相似。但是，.NET 2.x 标准将会有更大的 APIs 集，这也允许你依赖于 .NET 框架的库类。

PCLs和 .NET 标准之间的主要区别是：

* **平台搭配**。PCLs的一个挑战是，当你目标是多个平台时，它仍然是一组特殊集合。对于 NuGet 包，可以确认的是，你必须列出库文件名中的平台。当新的平台出现并支持相同的 APIs 时，这将导致问题。.NET 标准不存在这样的问题，因为你的目标版本不包含任何的平台问题。
* **平台的可用性**。目前 PCLs支持较为宽泛的平台，但是并不是所有的配置文件都具有相应的 .NET 标准版本。
* **库的可用性**。PCLs的设计是为了那些你无法依赖的、在选择的平台上无法运行的 APIs 和库。因此，PCLs 项目将只允许引用其它的PCLs。.NET 标准是相似的，但它增加了对 .NET 框架的二进制文件的引用。因此，使用 .NET 2.0 的标准，你将有机会获得更大的库。

为了做出明智的决定，我建议你：

1. 使用APIs 端口来查看你的代码库，是如何与各种版本的.NET标准兼容的。
2. 看.NET标准文档，以确保你选用的平台.NET标准是可用的。

例如，如果你想确认你是否能够使用 .NET2.0 标准，你可以通过以下的 APIs 文件命令行工具并且像这样运行你的库类，来检测应该使用 .NET1.6 标准还是 .NET2.0 标准：

> APIs port analyze -f C:\src\mylibs\ -t ".NET Standard,Version=1.6"^

-t ".NET Standard,Version=2.0"

注： .NET 标准2.0 仍然在制定中，因此 APIs 的可用性随时可能更改。我也建议你注意，那些在 .NET1.6 标准中可用，但是在 .NET2.0 标准中移除的APIs。

**总结**

我们已经创建了 .NET 标准，以便使得多个 .NET 平台之间代码的共享和复用变得更加容易。

在 .NET 2.0 标准中，我们更关注于兼容性。为了在 .NET Core 和 UWP 中支持 .NET 2.0 标准，我们将扩展这些平台，以便包括更多的现有的 APIs 。这也包含了兼容性功能，这种兼容性功能允许引用 .NET 框架中无法编译的二进制文件。

展望未来，我们建议你使用 .NET 标准，而不是便携式类库。.NET 2.0 标准也会在即将到来的 Visual Studio “Dev 15”发布的同时，进行推广。你可以以一个 NuGet 包的形式，来引用 .NET 标准。对于 Visual Studio, VS Code 和 Xamarin Studio，将会有一流的库类支持。

文章来源：by Immo Landwerth

英文原文链接：https://blogs.msdn.microsoft.com/dotnet/2016/09/26/introducing-net-standard/

原文链接：http://www.cnblogs.com/powertoolsteam/p/introducing\_net\_standard.html

**.NET社区新闻，深度好文，微信中搜索dotNET跨平台或扫描二维码关注**