第 1 章

.NET 应用程序体系结构

本章要点

- 回顾.NET 的历史
- 理解.NET Framework 4.6 和.NET Core 1.0 之间的差异
- 程序集和 NuGet 包
- 公共语言运行库
- Windows 运行库的特性
- 编写“Hello, World!”程序
- 通用 Windows 平台
- 创建 Windows 应用程序的技术
- 创建 Web 应用程序的技术

本章源代码下载：

打开网页 www.wrox.com/go/professionalcsharp6，单击 Download Code 选项卡即可下载本章源代码。本章代码分为以下几个主要的示例文件：

- DotnetHelloWorld
- HelloWorldApp (.NET Core)

1.1 选择技术

近年来，.NET 已经成为在 Windows 平台上创建任意类型的应用程序的巨大生态系统。有了.NET，可以创建 Windows 应用程序、Web 服务、Web 应用程序以及用于 Microsoft Phone 的应用程序。

.NET 的最新版本对上一版进行了很大的修改——也许是.NET 自问世以来最大的修改。.NET 的大部分代码已开放，还可以为其他平台创建应用程序。.NET 的新版本(.NET Core)和 NuGet 包允许微软公司以更短的更新周期提供新特性。应该使用什么技术来创建应用程序并不容易决定。本章将

提供这方面的帮助。其中包含用于创建 Windows、Web 应用程序和服务的不同技术的信息，指导选择什么技术进行数据库访问，凸显了.NET 和.NET Core 之间的差异。

1.2　回顾.NET 历史

要更好地理解.NET 和 C#的可用功能，最好先了解它的历史。表 1-1 显示了.NET 的版本、对应的公共语言运行库(Common Language Runtime，CLR)的版本、C#的版本和 Visual Studio 的版本，并指出相应版本的发布年份。除了知道使用什么技术之外，最好也知道不推荐使用什么技术，因为这些技术会被代替。

表 1-1

.NET	CLR	C#	Visual Studio
1.0	1.0	1.0	2002
1.1	1.1	1.2	2003
2.0	2.0	2.0	2005
3.0	2.0	2.0	2005+扩展版
3.5	2.0	3.0	2008
4.0	4.0	4.0	2010
4.5	4.0	5.0	2012
4.5.1	4.0	5.0	2013
4.6	4.0	6	2015
.NET Core 1.0	CoreCLR	6	2015 +扩展版

下面各小节详细介绍表 1-1，以及 C#和.NET 的发展。

1.2.1　C# 1.0 —— 一种新语言

C# 1.0 是一种全新的编程语言，用于.NET Framework。开发它时，.NET Framework 由大约 3000 个类和 CLR 组成。

(创建 Java 的 Sun 公司申请)法庭判决不允许微软公司更改 Java 代码后，Anders Hejlsberg 设计了 C#。Hejlsberg 为微软公司工作之前，在 Borland 公司设计了 Delphi 编程语言(一种 Object Pascal 语言)。Hejlsberg 在微软公司负责 J++(Java 编程语言的微软版本)。鉴于 Hejlsberg 的背景，C#编程语言主要受到 C++、Java 和 Pascal 的影响。

因为 C#的创建晚于 Java 和 C++，所以微软公司分析了其他语言中典型的编程错误，完成了一些不同的工作来避免这些错误。这些不同的工作包括：

- 在 if 语句中，布尔(Boolean)表达式是必须的(C++也允许在这里使用整数值)。
- 允许使用 struct 和 class 关键字创建值类型和引用类型(Java 只允许创建自定义引用类型；在 C++中，struct 和 class 之间的区别只是访问修饰符的默认值不同)。
- 允许使用虚拟方法和非虚拟方法 (这类似于 C++，Java 总是创建虚拟方法)。

当然，阅读本书，你会看到更多的变化。

现在，C#是一种纯粹的面向对象编程语言，具备继承、封装和多态性等特性。C#也提供了基于组件的编程改进，如委托和事件。

在.NET 和 CLR 推出之前，每种编程语言都有自己的运行库。在 C++中，C++运行库与每个 C++程序链接起来。Visual Basic 6 有自己的运行库 VBRun。Java 的运行库是 Java 虚拟机(Java Virtual Machine，JVC)——可以与 CLR 相媲美。CLR 是每种.NET 编程语言都使用的运行库。推出 CLR 时，微软公司提供了 JScript .NET、Visual Basic .NET、 Managed C++ 和 C#。JScript .NET 是微软公司的 JavaScript 编译器，与 CLR 和.NET 类一起使用。Visual Basic .NET 是提供.NET 支持的 Visual Basic。现在再次简称为 Visual Basic。Managed C++是混合了本地 C++代码与 Managed .NET 代码的语言。今天与.NET 一起使用的新 C++语言是 C++/ CLR。

.NET 编程语言的编译器生成中间语言(Intermediate Language，IL)代码。IL 代码看起来像面向对象的机器码，使用工具 ildasm.exe 可以打开包含.NET 代码的 DLL 或 EXE 文件来检查 IL 代码。CLR 包含一个即时(Just-In-Time，JIT)编译器，当程序开始运行时，JIT 编译器会从 IL 代码生成本地代码。

注意：IL 代码也称为托管代码。

CLR 的其他部分是垃圾回收器(GC)、调试器扩展和线程实用工具。垃圾回收器负责清理不再引用的托管内存，这个安全机制使用代码访问安全性来验证允许代码做什么；调试器扩展允许在不同的编程语言之间启动调试会话 (例如，在 Visual Basic 中启动调试会话，在 C#库内继续调试)；线程实用工具负责在底层平台上创建线程。

.NET Framework 的第 1 版已经很大了。类在名称空间内组织，以便于导航可用的 3000 个类。名称空间用来组织类，允许在不同的名称空间中有相同的类名，以解决冲突。.NET Framework 的第 1 版允许使用 Windows Forms(名称空间 System.Windows.Forms)创建 Windows 桌面应用程序，使用 ASP.NET Web Forms (System.Web)创建 Web 应用程序，使用 ASP.NET Web Services 与应用程序和 Web 服务通信，使用.NET Remoting 在.NET 应用程序之间更迅速地通信，使用 Enterprise Services 创建运行在应用程序服务器上的 COM +组件。

ASP.NET Web Forms 是创建 Web 应用程序的技术，其目标是开发人员不需要了解 HTML 和 JavaScript。服务器端控件会创建 HTML 和 JavaScript，这些控件的工作方式类似于 Windows Forms 本身。

C# 1.2 和.NET 1.1 主要是错误修复版本，改进较小。

注意：继承在第 4 章中讨论，委托和事件在第 9 章中讨论。

注意：.NET 的每个新版本都有 Professional C#图书的新版本。对于.NET 1.0，这本书已经是第 2 版了，因为第 1 版是以.NET 1.0 的 Beta 2 为基础出版的。目前，本书是第 10 版。

1.2.2　带有泛型的 C# 2 和.NET 2

C# 2 和.NET 2 是一个巨大的更新。在这个版本中，改变了 C#编程语言，建立了 IL 代码，所以需要新的 CLR 来支持 IL 代码的增加。一个大的变化是泛型。泛型允许创建类型，而不需要知道使用什么内部类型。所使用的内部类型在实例化(即创建实例)时定义。

C#编程语言中的这个改进也导致了 Framework 中多了许多新类型，例如 System.Collections.Generic 名称空间中新的泛型集合类。有了这个类，1.0 版本定义的旧集合类就很少用在新应用程序中了。当然，旧类现在仍然在工作，甚至在新的.NET Core 版本中也是如此。

注意：本书一直在使用泛型，详见第 6 章。第 11 章介绍了泛型集合类。

1.2.3　.NET 3.0 ——Windows Presentation Foundation

发布.NET 3.0 时，不需要新版本的 C#。3.0 版本只提供了新的库，但它发布了大量新的类型和名称空间。Windows Presentation Foundation(WPF)可能是新框架最大的一部分，用于创建 Windows 桌面应用程序。Windows Forms 包括本地 Windows 控件，且基于像素；而 WPF 基于 DirectX，独立绘制每个控件。WPF 中的矢量图形允许无缝地调整任何窗体的大小。WPF 中的模板还允许完全自定义外观。例如，用于苏黎世机场的应用程序可以包含看起来像一架飞机的按钮。因此，应用程序的外观可以与之前开发的传统 Windows 应用程序非常不同。System.Windows 名称空间下的所有内容都属于 WPF，但 System.Windows.Forms 除外。有了 WPF，用户界面可以使用 XML 语法设计 XAML(XML for Applications Markup Language)。

.NET 3.0 推出之前，ASP.NET Web Services 和.NET Remoting 用于应用程序之间的通信。Message Queuing 是用于通信的另一个选择。各种技术有不同的优点和缺点，它们都用不同的 API 进行编程。典型的企业应用程序必须使用一个以上的通信 API，因此必须学习其中的几项技术。WCF(Windows Communication Foundation) 解决了这个问题。WCF 把其他 API 的所有选项结合到一个 API 中。然而，为了支持 WCF 提供的所有功能，需要配置 WCF。

.NET 3.0 版本的第三大部分是 Windows WF(Workflow Foundation)和名称空间 System.Workflow。微软公司不是为几个不同的应用程序创建自定义的工作流引擎(微软公司本身为不同的产品创建了几个工作流引擎)，而是把工作流引擎用作.NET 的一部分。

有了.NET 3.0，Framework 的类从.NET 2.0 的 8 000 个增加到约 12 000 个。

注意：在本书中，WPF 参见第 29、30、31、34、35 和 36 章。WCF 详见第 44 章。

1.2.4　C# 3 和.NET 3.5——LINQ

.NET 3.5 和新版本 C# 3 一起发布。主要改进是使用 C#定义的查询语法，它允许使用相同的语法来过滤和排序对象列表、XML 文件和数据库。语言增强不需要对 IL 代码进行任何改变，因为这里使用的 C#特性只是语法糖。所有的增强也可以用旧的语法实现，只是需要编写更多的代码。C#语言很容易进行这些查询。有了 LINQ 和 lambda 表达式，就可以使用相同的查询语法来访问对象集

合、数据库和 XML 文件。

为了访问数据库并创建 LINQ 查询，LINQ to SQL 发布为.NET 3.5 的一部分。在.NET 3.5 的第一个更新中，发布了 Entity Framework 的第一个版本。LINQ to SQL 和 Entity Framework 都提供了从层次结构到数据库关系的映射和 LINQ 提供程序。Entity Framework 更强大，但 LINQ to SQL 更简单。随着时间的推移，LINQ to SQL 的特性在 Entity Framework 中实现了，并且 Entity Framework 会一直保留这些特性(现在它看起来与第一个版本非常不同)。

另一种引入为.NET 3.5 一部分的技术是 System.AddIn 名称空间，它提供了插件模型。这个模型提供了甚至在过程外部运行插件的强大功能，但它使用起来也很复杂。

注意：LINQ 详见第 13 章，Entity Framework 的最新版本与.NET 3.5 版本有很大差别，参见第 38 章。

1.2.5　C# 4 和.NET 4.0——dynamic 和 TPL

C# 4 的主题是动态集成脚本语言，使其更容易使用 COM 集成。C#语法扩展为使用 dynamic 关键字、命名参数和可选参数，以及用泛型增强的协变和逆变。

其他改进在.NET Framework 中进行。有了多核 CPU，并行编程就变得越来越重要。任务并行库(Task Parallel Library，TPL)使用 Task 类和 Parallel 类抽象出线程，更容易创建并行运行的代码。

因为用.NET 3.0 创建的工作流引擎没有履行自己的诺言，所以全新的 Windows Workflow Foundation 成为.NET 4.0 的一部分。为了避免与旧工作流引擎冲突，新的工作流引擎是在 System.Activity 名称空间中定义的。

C# 4 的增强还需要一个新版本的运行库。运行库从版本 2 跳到版本 4。

发布 Visual Studio 2010 时，附带了一项创建 Web 应用程序的新技术：ASP.NET MVC 2.0。与 ASP.NET Web Forms 不同，这项技术需要编写 HTML 和 JavaScript，并使用 C#和.NET 的服务器端功能。ASP.NET MVC 是定期更新的。

注意：C# 4 的 dynamic 关键字参见第 16 章。任务并行库参见第 21 章。ASP.NET 5 和 ASP.NET MVC 6 参见第 40 和 41 章。

1.2.6　C# 5 和异步编程

C# 5 只有两个新的关键字：async 和 await。然而，它大大简化了异步方法的编程。在 Windows 8 中，触摸变得更加重要，不阻塞 UI 线程也变得更加重要。用户使用鼠标，习惯于花些时间滚动屏幕。然而，在触摸界面上使用手势时，反应不及时很不好。

Windows 8 还为 Windows Store 应用程序(也称为 Modern 应用程序、Metro 应用程序、通用 Windows 应用程序，最近称为 Windows 应用程序)引入了一个新的编程接口：Windows 运行库。这是一个本地运行库，看起来像是使用语言投射的.NET。许多 WPF 控件都为新的运行库重写了，.NET Framework 的一个子集可以使用这样的应用程序。

System.AddIn 框架过于复杂、缓慢，所以用.NET 4.5 创建了一个新的合成框架：Managed Extensibility Framework 和名称空间 System.Composition。

独立于平台的通信的新版本是由 ASP.NET Web API 提供的。WCF 提供有状态和无状态的服务，以及许多不同的网络协议，而 ASP.NET Web API 则简单得多，它是基于 Representational State Transfer(REST)软件架构风格的。

注意：C# 5 的 async 和 await 关键字在第 15 章中详细讨论。其中也介绍.NET 在不同时期使用的不同异步模式。

MEF 参见第 26 章。Windows 应用程序参见第 29～33 章，ASP.NET Web API 参见第 42 章。

1.2.7　C# 6 和.NET Core

C# 6 没有由泛型、LINQ 和异步带来的巨大改进，但有许多小而实用的语言增强，可以在几个地方减少代码的长度。很多改进都通过新的编译器引擎 Roslyn 来实现。

注意：Roslyn 参见第 18 章。

完整的.NET Framework 并不是近年来使用的唯一.NET Framework。有些场景需要较小的框架。2007 年，发布了 Microsoft Silverlight 的第一个版本(代码名为 WPF/E，即 WPF Everywhere)。Silverlight 是一个 Web 浏览器插件，支持动态内容。Silverlight 的第一个版本只支持通过 JavaScript 编程。第 2 个版本包含.NET Framework 的子集。当然，不需要服务器端库，因为 Silverlight 总是在客户端运行，但附带 Silverlight 的框架 Framework 也删除了核心特性中的类和方法，使其更简洁，便于移植到其他平台。用于桌面的 Silverlight 最新版本(第 5 版)在 2011 年 12 月发布。Silverlight 也用于 Windows Phone 的编程。Silverlight 8.1 进入 Windows Phone 8.1，但这个版本的 Silverlight 也不同于桌面版本。

在 Windows 桌面上，有如此巨大的.NET 框架，需要更快的开发节奏，也需要较大的改进。在 DevOps 中，开发人员和操作员一起工作，甚至是同一个人不断地给用户提供应用程序和新特性，需要使新特性快速可用。由于框架巨大，且有许多依赖关系，创建新的特性或修复缺陷是一项不容易完成的任务。

有了几个较小的.NET Framework(如 Silverlight、用于 Windows Phone 的 Silverlight)，在.NET 的桌面版本和较小版本之间共享代码就很重要。在不同.NET 版本之间共享代码的一项技术是可移植库。随着时间的推移，有了许多不同的.NET Framework 和版本，可移植库的管理已成为一场噩梦。

为了解决所有这些问题，需要.NET 的新版本(是的，的确需要解决这些问题)。Framework 的新版本命名为.NET Core。.NET Core 较小，带有模块化的 NuGet 包以及分布给每个应用程序的运行库是开源的，不仅可用于 Windows 的桌面版，也可用于许多不同的 Windows 设备，以及 Linux 和 OS X。

为了创建 Web 应用程序，完全重写了 ASP.NET Core 1.0。这个版本不完全向后兼容老版本，需要对现有的 ASP.NET MVC(和 ASP.NET MVC 6)代码进行一些修改。然而，与旧版本相比，它也有

很多优点，例如每一个网络请求的开销较低，性能更好，也可以在 Linux 上运行。ASP.NET Web Forms 不包含在这个版本中，因为 ASP.NET Web Forms 不是专为最佳性能而设计的，它基于 Windows Forms 应用程序开发人员熟悉的模式来提高开发人员的友好性。

当然，并不是所有的应用程序都很容易改为使用.NET Core。所以这个巨大的框架也会进行改进——即使这些改进的完成速度没有.NET Core 那么快，也是要改进的。.NET Framework 完整的新版本是 4.6。ASP.NET Web Forms 的小更新包在完整的.NET 上可用。

注意：Roslyn 参见第 18 章。C#语言的变化参见第 I 部分中所有的语言章节，例如，只读属性参见第 3 章，nameof 运算符和空值传播参见第 8 章，字符串插值参见第 10 章，异常过滤器参见第 14 章。本书尽可能使用.NET Core。.NET Core 和 NuGet 包的更多信息参见本章后面的内容。

1.2.8　选择技术，继续前进

知道框架内技术相互竞争的原因后，就更容易选择用于编写应用程序的技术。例如，如果创建新的 Windows 应用程序，使用 Windows Forms 就不好。而应该使用基于 XAML 的技术，例如 Windows 应用程序，或者使用 WPF 的 Windows 桌面应用程序。

如果创建 Web 应用程序，肯定应使用 ASP.NET Core 与 ASP.NET MVC 6。做这个选择时要排除 ASP.NET Web Forms。如果访问数据库，就应该使用 Entity Framework 而不是 LINQ to SQL，应该选择 Managed Extensibility Framework 而不是 System.AddIn。

旧应用程序仍在使用 Windows Forms、ASP.NET Web Forms 和其他一些旧技术。只为改变现有的应用程序而使用新技术是没有意义的。进行修改必须有巨大的优势，例如，维护代码已经是一个噩梦，需要大量的重构以缩短客户要求的发布周期，或者使用一项新技术可以减少更新包的编码时间。根据旧有应用程序的类型，使用新技术可能不值得。可以允许应用程序仍使用旧技术，因为在未来的许多年仍将支持 Windows Forms 和 ASP.NET Web Forms。

本书的内容以新技术为基础，展示创建新应用程序的最佳技术。如果仍然需要维护旧应用程序，可以参考本书的老版本，其中介绍了 ASP.NET Web Forms、Windows Forms、System.AddIn 和其他仍然在.NET Framework 中可用的旧技术。

1.3　.NET 2015

.NET 2015 是所有.NET 技术的总称。图 1-1 给出了这些技术的总图。左边代表.NET Framework 4.6 技术，如 WPF 和 ASP.NET 4。ASP.NET Core 1.0 也可以在.NET Framework 4.6 上运行。右边代表新的.NET Core 技术。ASP.NET Core 1.0 和 UWP 运行在.NET Core 上。还可以创建在.NET Core 上运行的控制台应用程序。

.NET Core 的一个组成部分是一个新的运行库 CoreCLR。这个运行库从 ASP.NET Core 1.0 开始使用。不使用 CoreCLR 运行库，.NET 也可以编译为本地代码。UWP 自动利用这个特性，这些.NET 应用程序编译为本地代码之后，在 Windows Store 中提供。也可以把其他.NET Core 应用程序以及运

行在 Linux 上的应用程序编译为本地代码。

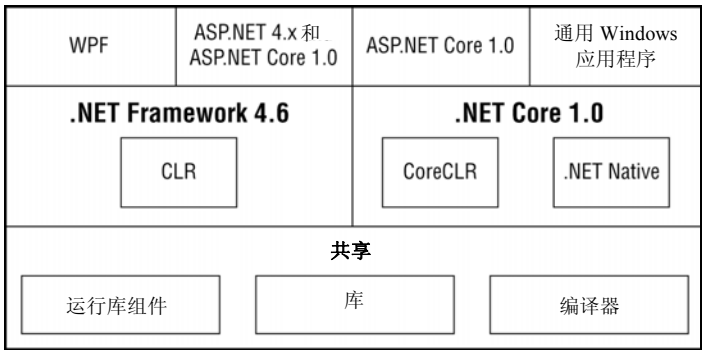

图 1-1

在图 1-1 的下方，.NET Framework 4.6 和.NET Core 1.0 之间还有一些共享的内容。运行库组件是共享的，如垃圾回收器的代码和 RyuJIT(这是一个新的 JIT 编译器，把 IL 代码编译为本地代码)。垃圾回收器由 CLR、CoreCLR 和.NET Native 使用。RyuJIT 即时编译器由 CLR 和 CoreCLR 使用。库可以在基于.NET Framework 4.6 和.NET Core 1.0 的应用程序之间共享。NuGet 包的概念帮助把这些库放在一个在所有.NET 平台上都可用的公共包上。当然，所有这些技术都使用新的.NET 编译器平台。

1.3.1　.NET Framework 4.6

.NET Framework 4.6 是.NET Framework 在过去 10 年不断增强的结果。1.2 节讨论的许多技术都基于这个框架。这个框架用于创建 Windows Forms 和 WPF 应用程序。此外，ASP.NET 5 可以在.NET Core 上运行，也可以在.NET Framework 4.6 上运行。

如果希望继续使用 ASP.NET Web Forms，就应选择 ASP.NET 4.6 和.NET Framework 4.6。ASP.NET 4.6 与 4.5 版本相比，也有新特性，比如支持 HTTP2(HTTP 协议的一个新版本，参见第 25 章)，用 Roslyn 编译器编译，以及异步模型绑定。然而，不能把 ASP.NET Web Forms 切换到.NET Core。

在目录%windows%\Microsoft.NET\Framework\v4.0.30319 下可以找到框架的库以及 CLR。

可用于.NET Framework 的类组织在 System 名称空间中。表 1-2 描述的名称空间提供了层次结构的思路。

表 1-2

名 称 空 间	说　明
System.Collections	这是集合的根名称空间。子名称空间也包含集合，如 System.Collections.Concurrent 和 System.Collections. Generic
System.Data	这是访问数据库的名称空间。System.Data.SqlClient 包含访问 SQL Server 的类
System.Diagnostics	这是诊断信息的根名称空间，如事件记录和跟踪(在 System.Diagnostics.Tracing 名称空间中)
System.Globalization	该名称空间包含的类用于全球化和本地化应用程序
System.IO	这是文件 IO 的名称空间，其中的类访问文件和目录，包括读取器、写入器和流
System.Net	这是核心网络的名称空间，比如访问 DNS 服务器，用 System.Net.Sockets 创建套接字
System.Threading	这是线程和任务的根名称空间。任务在 System.Threading.Tasks 中定义

(续表)

名 称 空 间	说　明
System.Web	这是 ASP.NET 的根名称空间。在这个名称空间下面定义了许多子名称空间，如 System.Web.UI、System.Web.UI.WebControls 和 System.Web.Hosting
System.Windows	这是用于带有 WPF 的 Windows 桌面应用程序的根名称空间。子名称空间有 System.Windows.Shapes、System.Windows.Data 和 System.Windows.Documents

注意：一些新的.NET 类使用以 Microsoft 开头而不是以 System 开头的名称空间，比如用于 Entity Framework 的 Microsoft.Data.Entity，用于新的依赖关系注入框架的 Microsoft.Extensions.DependencyInjection。

1.3.2　.NET Core 1.0

.NET Core 1.0 是新的.NET，所有新技术都使用它，是本书的一大关注点。这个框架是开源的，可以在 http://www.github.com/dotnet 上找到它。运行库是 CoreCLR 库；包含集合类的框架、文件系统访问、控制台和 XML 等都在 CoreFX 库中。

.NET Framework 要求必须在系统上安装应用程序需要的特定版本，而在.NET Core 1.0 中，框架(包括运行库)是与应用程序一起交付的。以前，把 ASP.NET Web 应用程序部署到共享服务器上有时可能有问题，因为提供程序安装了旧版本的.NET。这种情况已经一去不复返了。现在可以同时提交应用程序和运行库，而不依赖服务器上安装的版本。

.NET Core 1.0 以模块化的方式设计。该框架分成数量很多的 NuGet 包。根据应用程序决定需要什么包。添加新功能时，.NET Framework 就变得越来越大。删除不再需要的旧功能是不可能的，比如添加了泛型集合类，旧的集合类就是不必要的。.NET Remoting 被新的通信技术取代，或 LINQ to SQL 已经更新为 Entity Framework。删除某个功能，会破坏应用程序。这不适用于.NET Core，因为应用程序会发布它需要的部分框架。

目前.NET Core 的框架与.NET Framework 4.6 一样庞大。然而，这可以改变，它可以变得更大，但因为模块化，其增长潜力不是问题。.NET Core 已经如此之大，本书不可能包括每个类型。在 http://www.github.com/dotnet /corefx 中可以看到所有的源代码。例如，旧的非泛型集合类已被包含在.NET Core 中，使旧代码更容易进入新平台。

.NET Core 可以很快更新。即使更新运行库，也不影响现有的应用程序，因为运行库与应用程序一起安装。现在，微软公司可以增强.NET Core，包括运行库，发布周期更短。

注意：为了使用.NET Core 开发应用程序，微软公司创建了新的命令行实用程序.NET Core Command Line (CLI)。

1.3.3　程序集

.NET 程序的库和可执行文件称为程序集(assembly)。程序集是包含编译好的、面向.NET Framework 的代码的逻辑单元。

程序集是完全自描述性的，它是一个逻辑单元而不是物理单元，这意味着它可以存储在多个文件中(动态程序集存储在内存中，而不是存储在文件中)。如果一个程序集存储在多个文件中，其中就会有一个包含入口点的主文件，该文件描述了程序集中的其他文件。

可执行代码和库代码使用相同的程序集结构。唯一的区别是可执行的程序集包含一个主程序入口点，而库程序集不包含。

程序集的一个重要特征是它们包含的元数据描述了对应代码中定义的类型和方法。程序集也包含描述程序集本身的程序集元数据，这种程序集元数据包含在一个称为“清单(manifest)”的区域中，可以检查程序集的版本及其完整性。

由于程序集包含程序的元数据，因此调用给定程序集中的代码的应用程序或其他程序集不需要引用注册表或其他数据源就能确定如何使用该程序集。

在.NET Framework 4.6 中，程序集有两种类型：私有程序集和共享程序集。共享程序集不适用于 UWP，因为所有代码都编译到一个本机映像中。

1. 私有程序集

私有程序集一般附带在某个软件上，且只能用于该软件。附带私有程序集的常见情况是，以可执行文件或许多库的方式提供应用程序，这些库包含的代码只能用于该应用程序。

系统可以保证私有程序集不被其他软件使用，因为应用程序只能加载位于主执行文件所在文件夹或其子文件夹中的私有程序集。

用户一般会希望把商用软件安装在它自己的目录下，这样软件包不存在覆盖、修改或在无意间加载另一个软件包的私有程序集的风险。私有程序集只能用于自己的软件包，这样，用户对什么软件使用它们就有了更大的控制权。因此，不需要采取安全措施，因为这没有其他商用软件用某个新版本的程序集覆盖原来私有程序集的风险(但软件专门执行怀有恶意的损害性操作的情况除外)。名称也不会有冲突。如果私有程序集中的类正巧与另一个人的私有程序集中的类同名，是不会有问题的，因为给定的应用程序只能使用它自己的一组私有程序集。

因为私有程序集是完全自包含的，所以部署它的过程就很简单。只需要把相应的文件放在文件系统的对应文件夹中即可(不需要注册表项)，这个过程称为“0 影响(xcopy)安装”。

2. 共享程序集

共享程序集是其他应用程序可以使用的公共库。因为其他软件可以访问共享程序集，所以需要采取一定的保护措施来防止以下风险：

- 名称冲突，指另一个公司的共享程序集实现的类型与自己的共享程序集中的类型同名。因为客户端代码理论上可以同时访问这些程序集，所以这是一个严重的问题。
- 程序集被同一个程序集的不同版本覆盖—— 新版本与某些已有的客户端代码不兼容。

这些问题的解决方法是把共享程序集放在文件系统的特定子目录树中，称为全局程序集缓存(Global Assembly Cache，GAC)。与私有程序集不同，不能简单地把共享程序集复制到对应的文件夹

中，而需要专门安装到缓存中。有许多.NET 工具可以完成这个过程，并要求对程序集进行检查，在程序集缓存中设置一个小的文件夹层次结构，以确保程序集的完整性。

为了避免名称冲突，应根据私钥加密法为共享程序集指定一个名称(而对于私有程序集，只需要指定与其主文件名相同的名称即可)。该名称称为强名(strong name)，并保证其唯一性，它必须由要引用共享程序集的应用程序来引用。

与覆盖程序集的风险相关的问题，可以通过在程序集清单中指定版本信息来解决，也可以通过同时安装来解决。

1.3.4　NuGet 包

在早期，程序集是应用程序的可重用单元。添加对程序集的一个引用，以使用自己代码中的公共类型和方法，此时，仍可以这样使用(一些程序集必须这样使用)。然而，使用库可能不仅意味着添加一个引用并使用它。使用库也意味着一些配置更改，或者可以通过脚本来利用的一些特性。这是在 NuGet 包中打包程序集的一个原因。

NuGet 包是一个 zip 文件，其中包含程序集(或多个程序集)、配置信息和 PowerShell 脚本。

使用 NuGet 包的另一个原因是，它们很容易找到，它们不仅可以从微软公司找到，也可以从第三方找到。NuGet 包很容易在 NuGet 服务器 http://www.nuget.org 上获得。

在 Visual Studio 项目的引用中，可以打开 NuGet 包管理器(NuGet Package Manager，见图 1-2)，在该管理器中可以搜索包，并将其添加到应用程序中。这个工具允许搜索还没有发布的包(包括预发布选项)，定义应该在哪个 NuGet 服务器中搜索包。

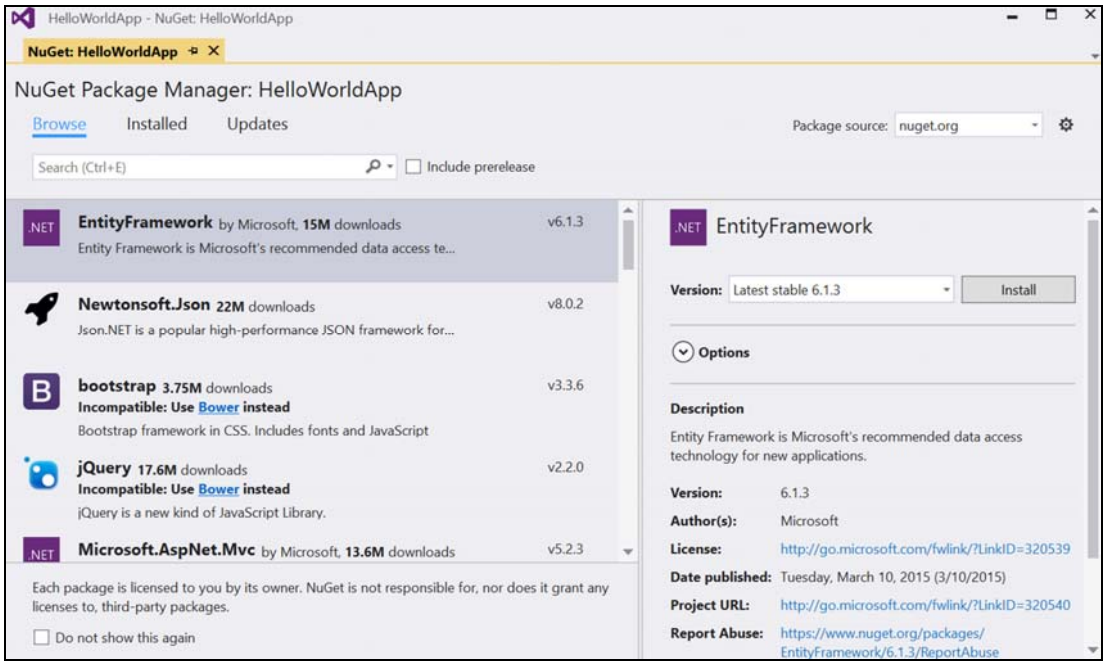

图 1-2

注意：使用 NuGet 服务器中的第三方包时，如果一个包以后才能使用，就总是有风险。还需要检查包的支持可用性。使用包之前，总要检查项目的链接信息。对于包的来源，可以选择 Microsoft and .NET，只获得微软公司支持的包。第三方包也包括在 Microsoft and .NET 部分中，但它们是微软公司支持的第三方包。

也可以让开发团队使用自己的 NuGet 服务器。可以定义开发团队只允许使用自己服务器中的包。

因为.NET Core 是模块化的，所以所有应用程序(除了最简单的应用程序)都需要额外的 NuGet 包。为了更容易找到包，本书使用.NET Core 构建的每个示例应用程序都显示了一个表格，列出需要添加的包和名称空间。

注意：NuGet 包管理器的更多信息参见第 17 章。

1.3.5　公共语言运行库

UWP(通用 Windows 平台)利用 Native .NET 把 IL 编译成本地代码。在所有其他场景中，使用.NET Framework 4.6 的应用程序和使用.NET Core 1.0 的应用程序都需要 CLR(Common Language Runtime，公共语言运行库)。然而，.NET Core 使用 CoreCLR，而.NET Framework 使用 CLR。

在 CLR 执行应用程序之前，编写好的源代码(使用 C#或其他语言编写的代码)都需要编译。在.NET 中，编译分为两个阶段：

(1) 将源代码编译为 Microsoft 中间语言(Intermediate Language，IL)。

(2) CLR 把 IL 编译为平台专用的本地代码。

IL 代码在.NET 程序集中可用。在运行时，JIT 编译器编译 IL 代码，创建特定于平台的本地代码。

新的 CLR 和 CoreCLR 包括一个新的 JIT 编译器 RyuJIT。新的 JIT 编译器不仅比以前的版本快，还在用 Visual Studio 调试时更好地支持 Edit & Continue 特性。Edit & Continue 特性允许在调试时编辑代码，可以继续调试会话，而不需要停止并重新启动过程。

CLR 还包括一个带有类型加载器的类型系统，类型加载器负责从程序集中加载类型。类型系统中的安全基础设施验证是否允许使用某些类型系统结构，如继承。

创建类型的实例后，实例还需要销毁，内存也需要回收。CLR 的另一个功能是垃圾回收器。垃圾回收器从托管堆中清除不再引用的内存。第 5 章解释其工作原理和执行的时间。

CLR 还负责线程的处理。在 C#中创建托管的线程不一定来自底层操作系统。线程的虚拟化和管理由 CLR 负责。

注意：如何在 C#中创建和管理线程参见第 21 章和第 22 章。

1.3.6　.NET Native

.NET Native 是.NET 2015 的一个新特性，它将托管程序编译成本地代码。对于 Windows 应用程序，这会生成优化的代码，其启动时间可以缩短 60%，内存的使用减少 15%～20%。

最初，.NET Native 把 UWP 应用编译为本地代码，以部署到 Windows Store。现在，.NET Native 将来也可以用于其他 .NET Core 应用程序，不过它目前还不能用于.NET Core 1.0 版本中，但可用于.NET Core 的将来版本中。可以把运行在 Windows 和 Linux 上的.NET Core 应用程序编译为本地代码。当然，在每一个平台上需要不同的本地映像。在后台.NET Native 共享 C++优化器，以生成本地代码。

1.3.7　Windows 运行库

从 Windows 8 开始，Windows 操作系统提供了另一种框架：Windows 运行库(Windows Runtime)。这个运行库由 WUP(Windows Universal Platform，Windows 通用平台)使用，Windows 8 使用第 1 版，Windows 8.1 使用第 2 版，Windows 10 使用第 3 版。

与.NET Framework 不同，这个框架是使用本地代码创建的。当它用于.NET 应用程序时，所包含的类型和.NET 类似。在语言投射的帮助下，Windows 运行库可以用于 JavaScript、C++和.NET 语言，它看起来像编程环境的本地代码。不仅方法因区分大小写而行为不同；方法和类型也可以根据所处的位置有不同的名称。

Windows 运行库提供了一个对象层次结构，它在以 Windows 开头的名称空间中组织。这些类没有复制.NET Framework 的很多功能；相反，提供了额外的功能，用于在 UWP 上运行的应用程序。如表 1-3 所示。

表 1-3

名 称 空 间	说　　明
Windows.ApplicationModel	这个名称空间及其子名称空间(如 Windows.ApplicationModel.Contracts)定义了类，用于管理应用程序的生命周期，与其他应用程序通信
Windows.Data	Windows.Data 定义了子名称空间，来处理文本、JSON、PDF 和 XML 数据
Windows.Devices	地理位置、智能卡、服务设备点、打印机、扫描仪等设备可以用 Windows.Devices 子名称空间访问
Windows.Foundation	Windows.Foundation 定义了核心功能。集合的接口用名称空间 Windows.Foundation.Collections 定义。这里没有具体的集合类。相反，.NET 集合类型的接口映射到 Windows 运行库类型
Windows.Media	Windows.Media 是播放、捕获视频和音频、访问播放列表和语音输出的根名称空间
Windows.Networking	这是套接字编程、数据后台传输和推送通知的根名称空间
Windows.Security	Windows.Security.Credentials 中的类提供了密码的安全存储区；Windows.Security.Credentials.UI 提供了一个选择器，用于从用户处获得凭证
Windows.Services.Maps	这个名称空间包含用于定位服务和路由的类
Windows.Storage	有了 Windows.Storage 及其子名称空间，就可以访问文件和目录，使用流和压缩
Windows.System	Windows.System 名称空间及其子名称空间提供了系统和用户的信息，也提供了一个启动其他应用程序的启动器
Windows.UI.Xaml	在这个名称空间中，可以找到很多用于用户界面的类型

1.4　Hello, World

下面进入编码，创建一个 Hello, World 应用程序。自 20 世纪 70 年代以来，Brian Kernighan 和 Dennis Ritchie 编写了 *The C Programming Language* 一书，使用 Hello,World 应用程序开始学习编程语言就成为一个传统。有趣的是，自 C#发明以来，Hello, World 的语法用 C# 6 改变后，这一简单的程序第一次看起来非常不同。

创建第一个示例不借助于 Visual Studio，而是使用命令行工具和简单的文本编辑器(如记事本)，以便看到后台会发生什么。之后，就转而使用 Visual Studio，因为它便于编程。

在文本编辑器中输入以下源代码，并用.cs 作为扩展名(如 HelloWorld.cs)保存它。Main()方法是.NET 应用程序的入口点。CLR 在启动时调用一个静态的 Main()方法。Main()方法需要放到一个类中。这里的类称为 Program，但是可以给它指定任何名字。WriteLine 是 Console 类的一个静态方法。Console 类的所有静态成员都用第一行中的 using 声明打开。using static System.Console 打开 Console 类的静态成员，所以不需要输入类名，就可以调用方法 WriteLine()(代码文件 Dotnet / HelloWorld.cs)：

```
using static System.Console;

class Program
{
  static void Main()
  {
    WriteLine("Hello, World!");
  }
}
```

如前所述，Hello, World 的语法用 C# 6 略加改变。在 C# 6 推出之前，using static 并不可用，只能通过 using 声明打开名称空间。当然，下面的代码仍然适用于 C# 6(代码文件 Dotnet/ HelloWorld2.cs)：

```
using System;

class Program
{
  static void Main()
  {
    Console.WriteLine("Hello, World!");
  }
}
```

using 声明可以减少打开名称空间的代码。编写 Hello,World 程序的另一种方式是删除 using 声明，在调用 WriteLine()方法时，给 Console 类添加 System 名称空间(代码文件 Dotnet/HelloWorld3.cs)：

```
class Program
{
  static void Main()
  {
    System.Console.WriteLine("Hello, World!");
  }
}
```

编写源代码之后，需要编译代码来运行它。

1.5　用.NET 4.6 编译

对源文件运行 C#命令行编译器(csc.exe)，就可以编译这个程序，如下所示：

```
csc HelloWorld.cs
```

如果想使用 csc 命令在命令行上编译代码，就应该知道，.NET 命令行工具(包括 csc)只有设置了某些环境变量后才可用。根据安装.NET(和 Visual Studio)的方式，计算机可能设置了这些环境变量，也可能没有设置。

注意：如果没有设置环境变量，则有 3 个选择。第一个选择是在调用 csc 可执行文件时添加路径。它位于% ProgramFiles %\MsBuild\14.0\Bin\csc.exe。如果安装了 dotnet 工具，则 csc 在%ProgramFiles%\dot.net\bin\csc.exe 上。第二个选择是在运行 csc 前，从命令提示符下运行批处理文件% Microsoft Visual Studio 2015%\Common7\vsvars32.bat，其中%Microsoft Visual Studio 2015%是安装 Visual Studio 2015 的文件夹。第三个选择、也是最容易的方式，是使用 Visual Studio 2015 命令提示符代替 Windows 命令提示符。要在“开始”菜单中找到 Visual Studio 2015 命令提示符，选择 Programs｜Microsoft Visual Studio 2015｜Visual Studio Tools。Visual Studio 2015 命令提示符只是一个命令提示符窗口，它打开时会自动运行 vsvars32.bat。

编译代码，生成一个可执行文件 HelloWorld.exe，在命令行上可以运行它。也可以在 Windows 资源管理器中运行它，就像运行任何其他可执行文件一样。试一试：

```
> csc HelloWorld.cs
Microsoft (R) Visual C# Compiler version 1.1.0.51109
Copyright (C) Microsoft Corporation. All rights reserved.
> HelloWorld
Hello World!
```

以这种方式编译可执行程序，会生成一个程序集，其中包含 IL(中间语言)代码。程序集可以使用中间语言反汇编程序(Intermediate Language Disassembler，IL DASM)工具读取。如果运行 ildasm.exe，打开 HelloWorld.exe，会发现程序集包含一个 Program 类型和一个 Main()方法，如图 1-3 所示。

双击树视图中的 MANIFEST 节点，显示程序集的元数据信息(如图 1-4 所示)。这个程序集会利用 mscorlib 程序集(因为 Console 类位于 mscorlib 程序集里)、一些配置和 HelloWorld 程序集的版本。

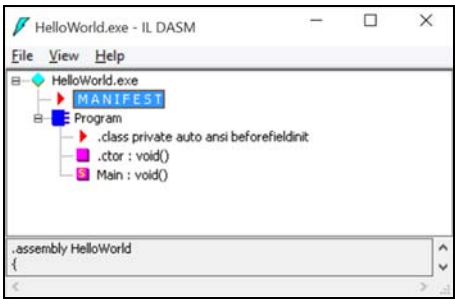

图 1-3

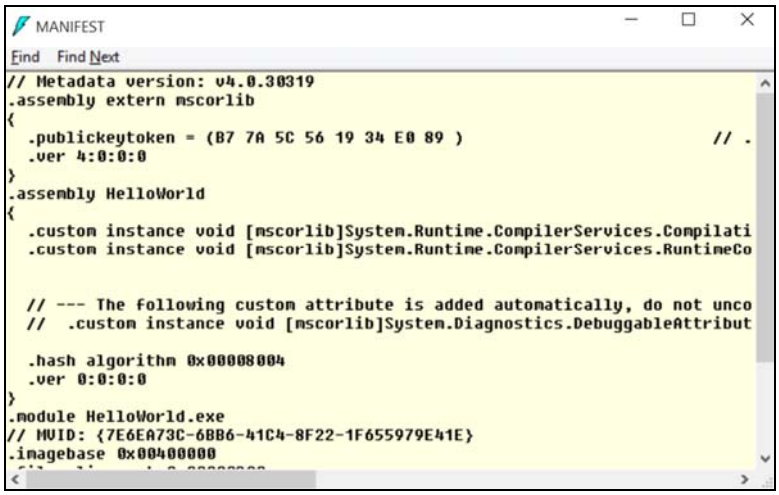

图 1-4

双击 Main()方法，显示该方法的 IL 代码(如图 1-5 所示)。不管编译 Hello, World 代码的什么版本，结果都是一样的。都是调用 mscorlib 程序集中定义的 System.Console.WriteLine 方法，传递字符串，之后加载字符串 Hello, World!。CLR 的一个特性是 JIT 编译器。运行该应用程序时，JIT 编译器把 IL 代码编译为本地代码。

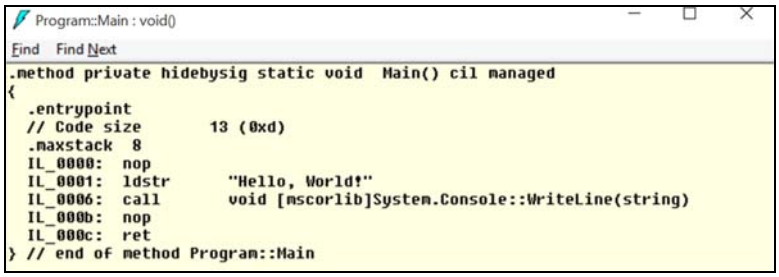

图 1-5

1.6　用.NET Core CLI 编译

如果没有 Visual Studio 的帮助，使用新的.NET Core CLI(Command Line，命令行)就需要做一些准备，才能编译应用程序。下面看看编译 Hello, World 示例应用程序的新工具。

1.6.1　设置环境

安装了 Visual Studio 2015 和最新的更新包后，就可以立即启动 CLI 工具。否则，就需要安装.NET Core 和 CLI 工具。Windows、Linux 和 OS X 的下载指令可以从 http://dotnet.github.io 获取。

在 Windows 上，不同版本的.NET Core 运行库以及 NuGet 包安装在用户配置文件中。使用.NET 时，这个文件夹的大小会增加。随着时间的推移，会创建多个项目，NuGet 包不再存储在项目中，而是存储在这个用户专用的文件夹中。这样做的优势在于，不需要为每个不同的项目下载 NuGet 包。这个 NuGet 包下载后，它就在系统上。因为不同版本的 NuGet 包和运行库都是可用的，所有的不同版本都存储在这个文件夹中。不时地检查这个文件夹，删除不再需要的旧版本，可能很有趣。

安装.NET Core CLI 工具，要把 dotnet 工具作为入口点来启动所有这些工具。开始时：

```
> dotnet
```

会看到，dotnet 工具的所有不同选项都可用。

repl(read、eval、print、loop)命令适合于学习和测试简单的 C#特性，而无须创建程序。用 dotnet 工具启动 repl：

```
> dotnet repl
```

这会启动一个交互式 repl 会话。使用一个变量，可以给 Hello, World 输入以下语句：

```
> using static System.Console;
> var hello = "Hello, World!";
> WriteLine(hello);
```

输入最后一条语句后，输出就是 Hello, World!字符串。

dotnet repl 命令不可用于 dotnet 工具的 Preview 2 版本，但在将来可以作为其扩展用于该工具。

1.6.2　构建应用程序

dotnet 工具提供了一种简单的方法来创建 Hello, World 应用程序。创建一个新的目录 HelloWorldApp，用命令提示符进入这个目录。然后输入如下命令：

```
> dotnet new
```

这个命令创建一个 Program.cs 文件(其中包括 Hello, World 应用程序的代码)、一个 NuGet.config 文件(定义了应该加载 NuGet 包的 NuGet 服务器)和新的项目配置文件 project.json。

注意：使用 dotnet new，还可以创建库和 ASP.NET Web 应用程序所需要的初始文件(使用选项--type)。也可以选择其他编程语言，如 F#和 Visual Basic(使用选项--lang)。

创建的项目配置文件是 project.json。这个文件采用 JavaScript Object Notation(JSON)格式，定义了框架应用程序信息，如版本、描述、作者、标签、依赖的库和应用程序支持的框架。生成的项目配置文件如下面的代码片段所示(代码文件 HelloWorldApp/project.json)：

```
{
  "version": "1.0.0-*",
  "buildOptions": {
```

```
    "emitEntryPoint": true
  },

  "dependencies": {
    "NETStandard.Library": "1.0.0-*"
  },

  "frameworks" : {
    "netstandardapp1.5": {
      "imports": "dnxcore50"
    }
  },
  "runtimes" : {
    "ubuntu.14.04-x64": { },
    "win7-x64": { },
    "win10-x64": { },
    "osx.10.10-x64": { },
    "osx.10.11-x64": { }
  }
}
```

通过 compilationOptions 设置来设置 emitEntryPoint。如果把 Main()方法创建为程序的入口点，这就是必要的。这个 Main()方法在运行应用程序时调用。库不需要这个设置。

对于依赖关系部分，可以添加程序的所有依赖项，如需要编译程序的额外 NuGet 包。默认情况下，NetStandard.Library 添加为一个依赖项。NetStandard.Library 是一个引用的 NuGet 包，这个包引用了其他 NuGet 包。有了它，就可以避免添加很多其他的包，如 Console 类的 System.Console、泛型集合类的 System.Collections 等。NetStandard.Library 1.0 是一个标准，定义了所有.NET 平台必须支持的一组程序集，在网站 https://github.com/dotnet/corefx/blob/master/Documentation/project-docs/standard-platform.md 上，可以找到一长串.NET 1.0 中的程序集及其版本号，以及.NET 标准的 1.1、1.2、1.3 和 1.4 版本中添加的程序集。

NetStandard.Library 1.0 有一个依赖项，可以支持.NET Framework 4.5.2 及以上版本(对.NET 4、4.5、4.5.1 的支持结束于 2016 年 1 月)、.NET Core 1.0、UWP 10.0 和其他.NET Framework，如 Windows Phone Silverlight 8.0、Mono 和 Mono/Xamarin。对 1.3 版本的修改有限支持.NET 4.6、.NET Core 1.0、UWP 10.0 和 Mono/Xamarin 平台。1.4 版本有限支持.NET 4.6.1、.NET Core 1.0 和 Mono/Xamarin 平台，但版本越新，可用的程序集列表就越大。

project.json 中的 frameworks 部分列出了应用程序支持的.NET Framework。默认情况下，应用程序只为.NET Core 1.0 构建为 netstandardapp1.5 moniker 指定的名称。netstandardapp1.5 与.NET Core 1.0 所构建的应用程序一起使用。通过库可以使用 netstandard1.0 moniker。这允许在.NET Core 应用程序和使用.NET Framework 的应用程序中使用库。netstandardapp1.5 内的 imports 部分引用旧名称 dnxcore50，该旧名称映射到新名字上。这允许使用仍在使用旧名称的包。

.NET Core 是框架的新开源版本，可用于 Windows、Linux 和 OS X。应该支持的运行库需要添加到 runtimes 部分。前面的代码片段显示了对 Ubuntu Linux 发行版、Windows 7(也允许在 Windows 8 上运行应用程序)、Windows 10 和 OS X 的支持。

添加字符串 net46，为 .NET Framework 4.6 构建程序：

```
"frameworks" : {
  "netstandardapp1.5" : { }
  "net46" : { }
}
```

给 frameworks 部分添加 net46，就不再支持非 Windows 运行库，因此需要删除这些运行库。

还可以添加额外的元数据，比如描述、作者信息、标签、项目和许可 URL：

```
"version": "1.0.0-*",
"description": "HelloWorld Sample App for Professional C#",
"authors": [ "Christian Nagel" ],
"tags": [ "Sample", "Hello", "Wrox" ],
"projectUrl": "http://github.com/professionalCSharp/",
"licenseUrl": "",
```

给 project.json 文件添加多个框架时，可以在 frameworks 下面的 dependencies 部分中为每个框架指定专门的依赖项。dependencies 部分中指定的依赖项，若处于 frameworks 部分所在的层次上，就指定了所有框架共同的依赖项。

有了项目结构后，就可以使用如下命令下载应用程序的所有依赖项：

```
> dotnet restore
```

此时，命令提示符定位在 project.json 文件所在的目录。这个命令会下载应用程序所需要的所有依赖项，即 project.json 文件中定义的项。指定版本 1.0.0 - *，会得到版本 1.0.0，*表示可用的最新版本。在 project.lock.json 文件中，可以看到检索了什么 NuGet 包的哪个版本，包括依赖项的依赖项。记住，包存储在用户专门的文件夹中。

要编译应用程序，启动命令 dotnet build，输出如下——为.NET Core 1.0 和.NET Framework 4.6 编译：

```
> dotnet build
Compiling HelloWorldApp for .NETStandardApp, Version=1.5"
Compilation succeeded.
  0 Warning(s)
  0 Error(s)
Time elapsed 00:00:02.6911660

Compiling HelloWorldApp for .NETFramework,Version=v4.6
Compilation succeeded.
  0 Warning(s)
  0 Error(s)
Time elapsed 00:00:03.3735370
```

编译过程的结果是 bin/debug/[netstandardapp1.5|net46]文件夹中包含 Program 类的 IL 代码的程序集。如果比较.NET Core 与.NET 4.6 的编译结果，会发现一个包含 IL 代码和.NET Core 的 DLL，和包含 IL 代码和.NET 4.6 的 EXE。为.NET Core 生成的程序集有一个对 System.Console 程序集的依赖项，而.NET 4.6 程序集在 mscorlib 程序集中找到 Console 类。

还可以使用下面的命令行把程序编译成本地代码：

```
> dotnet build --native
```

编译为本地代码，会加快应用程序的启动，消耗更少的内存。本地编译过程会把应用程序的 IL 代码以及所有依赖项编译为单一的本地映像。别指望.NET Core 的所有功能都可用于编译为本地代码，但是随着时间的推移，微软公司的继续发展，越来越多的应用程序可以编译为本地代码。

要运行应用程序，可以使用 dotnet 命令：

```
> dotnet run
```

要使用特定版本的框架启动应用程序，可以使用-framework 选项。这个框架必须用 project.json 文件配置：

```
> dotnet run --framework net46
```

启动 bin/debug 目录中的可执行程序，也可以运行应用程序。

注意：前面是在 Windows 上构建和运行 Hello, World 应用程序，而 dotnet 工具在 Linux 和 OS X 上的工作方式是相同的。可以在这两个平台上使用相同的 dotnet 命令。在使用 dotnet 命令之前，只需要准备基础设施：为 Ubuntu Linux 使用 sudo 实用工具，在 OS X 上安装一个 PKG 包，参见 http://dotnet.github。安装好.NET Core CLI 后，就可以通过本小节介绍的方式使用 dotnet 工具——只是.NET Framework 4.6 是不可用的。除此之外，可以恢复 NuGet 包，用 dotnet restore、dotnet build 和 dotnet run 编译和运行应用程序。

本书的重点是 Windows，因为 Visual Studio 2015 提供了一个比其他平台更强大的开发平台，但本书的许多代码示例是基于.NET Core 的，所以也能够在其他平台上运行。还可以使用 Visual Studio Code(一个免费的开发环境)，直接在 Linux 和 OS X 上开发应用程序，参见 1.8 节，了解 Visual Studio 不同版本的更多信息。

1.6.3　打包和发布应用程序

使用 dotnet 工具还可以创建一个 NuGet 包，发布应用程序，以进行部署。

命令 dotnet pack 创建了可以放在 NuGet 服务器上的 NuGet 包。开发人员现在可以使用下面的命令来引用包：

```
> dotnet pack
```

用 HelloWorldApp 运行这个命令，会创建文件 HelloWorldApp.1.0.0.nupkg，其中包含用于所有支持框架的程序集。NuGet 包是一个 ZIP 文件。如果用 zip 扩展名重命名该文件，就可以轻松地查看它的内容。对于示例应用程序，创建了两个文件夹 dnxcore500 和 net46，其中包含各自的程序集。文件 HelloWorldApp.nuspec 是一个 XML 文件，它描述了 NuGet 包，列出了支持框架的内容，还列出了安装 NuGet 包之前所需要的程序集依赖项。

要发布应用程序，还需要在目标系统上有运行库。发布所需要的文件可以用 dotnet publish 命令创建：

```
> dotnet publish
```

使用可选参数，可以指定用于发布的特定运行库(选项-r)或不同的输出目录(选项-o)。在 Windows 系统上运行这个命令后，会创建 win7-x64 文件夹，其中包含目标系统上需要的所有文件。注意，在.NET Core 中包含运行库，因此不管安装的运行库是什么版本都没有关系。

1.7 应用程序类型和技术

可以使用 C#创建控制台应用程序，本章的大多数示例都是控制台应用程序。对于实际的程序，控制台应用程序并不常用。使用 C#创建的应用程序可以使用与.NET 相关的许多技术。本节概述可以用 C#编写的不同类型的应用程序。

1.7.1 数据访问

在介绍应用程序类型之前，先看看所有应用程序类型都使用的技术：数据访问。

文件和目录可以使用简单的 API 调用来访问，但简单的 API 调用对于有些场景而言不够灵活。使用流 API 有很大的灵活性，流提供了更多的特性，例如加密或压缩。阅读器和写入器简化了流的使用。所有可用的不同选项都包含在第 23 章中。也可能以 XML 或 JSON 格式序列化完整的对象。第 27 章讨论了这些选项。

为了读取和写入数据库，可以直接使用 ADO.NET(参见第 37 章)；也可以使用抽象层：ADO.NET Entity Framework (参见第 38 章)。Entity Framework 提供了从对象层次结构到数据库关系的映射。

ADO.NET Entity Framework 进行了若干次迭代。Entity Framework 的不同版本值得讨论，以很好地理解 NuGet 包的优点。还要了解不应继续使用 Entity Framework 的哪些部分。

表 1-4 描述了 Entity Framework 的不同版本和每个版本的新特性。

表 1-4

Entity Framework	说　明
1.0	.NET 3.5 SP1 可用。这个版本提供了通过 XML 文件把表映射到对象的映射
4.0	在.NET 4 中，Entity Framework 从第 1 版跳到第 4 版
4.1	支持代码优先
4.2	修复错误
4.3	添加了迁移
5.0	与.NET 4.5 一起发布，提供性能改进，支持新的 SQL Server 特性
6.0	移动到 NuGet 包中
7.0	完全重写，也支持 NoSQL、也运行在 Windows 应用程序上

下面介绍一些细节。Entity Framework 最初发布为.NET Framework 类的一部分，和.NET Framework 一起预装。Entity Framework 1 是.NET 3.5 第一个服务包的一部分，它有一个特性更新：.NET 3.5 Update 1。

第 2 版有许多新特性，因此决定和.NET 4 一起跳到第 4 版。之后，Entity Framework 以比.NET Framework 更快的节奏发布新版本。要得到 Entity Framework 的更新版本，必须把一个 NuGet 包添加到应用程序中(版本 4.1、4.2、4.3)。这种方法有一个问题。已经通过.NET Framework 发布的类必须用以前的方式使用。只有新添加的功能，如代码优先，用 NuGet 包添加。

Entity Framework 5.0 与.NET 4.5 一起发布。其中的一些类附带在预装的.NET Framework 中，而附加的特性是 NuGet 包的一部分。NuGet 包也允许给 Entity Framework 5.0 和.NET 4.0 应用程序安装 NuGet 包。然而在现实中，Entity Framework 5.0 添加到.NET 4.0 项目中时，(通过脚本)决定的包就是 Entity Framework 4.4，因为一些类型必须属于.NET 4.5，而不是.NET 4。

下一个版本的 Entity Framework 解决了这个问题，它把所有 Entity Framework 类型移动到 NuGet 包中；忽略框架本身附带的类型。这允许同时使用 Framework 6.0 与旧版本，而不会局限于 Framework 4.5。为了不与框架的类冲突，有些类型移到不同的名称空间中。对此，ASP.NET Web Forms 的一些特性有一个问题，因为它们使用了 Entity Framework 的原始类，而这些类映射到新类没有那么轻松。

在发布不同版本的过程中，Entity Framework 提供了不同的选项，把数据库表映射到类。前两个选项是 Database First 和 Model First。在这两个选项中，映射是通过 XML 文件完成的。XML 文件通过图形设计器表示，可以把实体从工具箱拖动到设计器中，进行映射。

在 4.1 版本中，添加了通过代码来映射的选项：代码优先(Code First)。代码优先并不意味着数据库不能事先存在。两者都是可能的：数据库可以动态地创建，数据库也可以在编写代码之前存在。使用代码优先，不通过 XML 文件进行映射。相反，属性或流利的 API 可以以编程方式定义映射。

Entity Framework Core 1.0 是 Entity Framework 的完全重新设计，新名称反映了这一点。代码需要更改，把应用程序从 Entity Framework 的旧版本迁移到新版本。旧的映射变体，如 Database First 和 Model First 已被删除，因为代码优先是更好的选择。完全重新设计也支持关系数据库和 NoSQL。Azure Table Storage 现在是可以使用 Entity Framework 的一个选项。

1.7.2　Windows 桌面应用程序

为了创建 Windows 桌面应用程序，可以使用两种技术：Windows Forms 和 Windows Presentation Foundation。Windows Forms 包含包装本地 Windows 控件的类；它基于像素图形。Windows Presentation Foundation(WPF)是较新的技术，基于矢量图形。

WPF 在构建应用程序时利用了 XAML。XAML 是指可扩展的应用程序标记语言(Extensible Application Markup Language)。这种在微软环境中创建应用程序的方式在 2006 年引入，是.NET Framework 3.0 的一部分。.NET 4.5 给 WPF 引入了新特性，如功能区控件和实时塑造。

XAML 是用来创建表单的 XML 声明，表单代表 WPF 应用程序的所有可视化方面和行为。虽然可以以编程方式使用 WPF 应用程序，但 WPF 是迈向声明性编程方向的一步，软件业正转向这个方向。声明性编程意味着，不使用 C#、Visual Basic 或 Java 等编译语言通过编程方式创建对象，而是通过 XML 类型的编程方式声明一切对象。第 29 章介绍了 XAML(使用 XML Paper Specification、Windows Workflow Foundation 和 Windows Communication Foundation)。第 30 章涵盖了 XAML 样式和资源。第 34 章提供控件、布局和数据绑定的细节。打印和创建文档是 WPF 的另一个重要方面，参见第 35 章。

WPF 的未来是什么？ UWP 是用于未来的新应用程序的 UI 平台吗？UWP 的优点是也支持移动设备。只要一些用户没有升级到 Windows 10，就需要支持旧的操作系统，如 Windows 7。UWP

应用程序不在 Windows 7/8 上运行。可以使用 WPF。如果还愿意支持移动设备，最好共享尽可能多的代码。通过支持 MVVM 模式，可以使用尽可能多的通用代码，通过 WPF 和 UWP 创建应用程序。这种模式参见第 31 章。

1.7.3　UWP

UWP(Universal Windows Platform，通用 Windows 平台)是微软公司的一个战略平台。使用 UWP 创建 Windows 应用程序时，只能使用 Windows 10 和 Windows 的更新版本，但不限于 Windows 的桌面版本。使用 Windows 10 有很多不同的选择，如 Phone、Xbox、Surface Hub、HoloLens 和 IoT。还有一个适用于所有这些设备的 API!

一个适用于所有这些设备的 API？是的！每个设备系列都可以添加自己的软件开发工具包(Software Development Kit，SDK)，来添加不是 API 的一部分、但对所有设备可用的功能。添加这些 SDK 不会破坏应用程序，但需要以编程方式检查，这种 SDK 中的 API 在运行应用程序的平台上是否可用。根据需要区分的 API 调用数，代码可能变得混乱，依赖注入可能是更好的选择。

注意：第 31 章讨论了依赖注入，以及对基于 XAML 的应用程序有用的其他模式。

可以决定什么设备系列支持应用程序。并不是所有的设备系列都可用于每个应用程序。

在 Windows 10 后，会有新版本的 Windows 吗？Windows 11 尚未计划。对于 Windows 应用程序(也称为 Metro 应用程序、Windows Store 应用程序、Modern 应用程序和 UWP)，应针对 Windows 8 或 Windows 8.1。Windows 8 应用程序通常也在 Windows 8.1 上运行，但 Windows 8.1 应用程序不在 Windows 8 上运行。这是非常不同的。为 UWP 创建应用程序时，目标版本是 10.0.10130.0，定义了可用的最低版本和要测试的最新版本，并假设它在未来版本上运行。根据可以为应用程序使用的功能，以及希望用户拥有的版本，可以决定要支持的最低版本。个人用户通常会自动更新版本；企业用户可能会坚持使用旧版本。

运行在 UWP 上的 Windows 应用程序利用了 Windows 运行库和.NET Core。讨论这些应用程序类型的最重要部分是第 32 章和第 33 章。

1.7.4　SOAP 服务和 WCF

Windows Communication Foundation(WCF)是一个功能丰富的技术，旨在取代在 WCF 以前可用的通信技术，它为基于标准的 Web 服务使用的所有特性提供基于 SOAP 的通信，如安全性、事务、双向和单向通信、路由、发现等。WCF 允许一次性构建服务，然后在一个配置文件中进行更改，让这个服务用于许多方面(甚至在不同的协议下)。WCF 是一种强大而复杂的方式，用来连接完全不同的系统。第 44 章将介绍它。

1.7.5　Web 服务和 ASP.NET Web API

ASP.NET Web API 是非常容易通信的一个选项，能满足分布式应用程序 90%以上的需求。这项技术基于 REST (Representational State Transfer)，它定义了无状态、可伸缩的 Web 服务的指导方针和最佳实践。

客户端可以接收 JSON 或 XML 数据。JSON 和 XML 也可以格式化来使用 Open Data 规范(OData)。

这个新 API 的功能是便于通过 JavaScript 和 UWP 使用 Web 客户端。

ASP.NET Web API 是创建微服务的一个好方法。创建微服务的方法定义了更小的服务，可以独立运行和部署，可以自己控制数据存储。

ASP.NET 5 是 ASP.NET Web API 的旧版本，从 ASP.NET MVC 分离而来，现在与 ASP.NET MVC 6 合并，使用相同的类型和特征。

注意：ASP.NET Web API 和微服务的更多信息参见第 42 章。

1.7.6　WebHooks 和 SignalR

对于实时 Web 功能以及客户端和服务器端之间的双向通信，可以使用的 ASP.NET 技术是 WebHooks 和 SignalR。

只要信息可用，SignalR 就允许将信息尽快推送给连接的客户。SignalR 使用 WebSocket 技术，在 WebSocket 不可用时，它可以回退到基于拉的通信机制。

WebHooks 可以集成公共服务，这些服务可以调用公共 ASP.NET Web API 服务。WebHooks 技术从 GitHub 或 Dropbox 和其他服务中接收推送通知。

1.7.7　Windows 服务

Web 服务无论是通过 WCF 完成还是通过 ASP.NET Web 服务完成，都需要一个主机才能运行。IIS(Internet Information Server，互联网信息服务器)通常是一个很好的选择，因为它提供了所有的服务，但它也可以是自定义程序。使用自定义选项创建一个后台进程，在运行 Windows 时启动的是 Windows 服务。这个程序设计为在基于 Windows NT 内核的操作系统的后台运行。希望程序持续运行，做好响应事件的准备，而不是让用户显式地启动时，就可以使用服务。一个很好的例子是 Web 服务器上的 World Wide Web 服务，它监听来自客户端的 Web 请求。

很容易用 C#编写服务。.NET Framework 基类在 System.ServiceProcess 名称空间中，处理与服务相关的许多样板任务。此外，Visual Studio .NET 允许创建 C# Windows 服务(Service)项目，它给基本 Windows 服务使用 C#源代码。第 39 章探讨了如何编写 C# Windows 服务。

1.7.8　Web 应用程序

最初引入 ASP.NET 1，从根本上改变了 Web 编程模型。ASP.NET 5 是新的主要版本，允许使用.NET Core 提高性能和可伸缩性。这个新版本也可以在 Linux 系统上运行，这个需求很高。

在 ASP.NET 5 中，不再包含 ASP.NET Web Forms(它仍然可以使用，在.NET 4.6 中更新)，所以本书关注现代技术 ASP.NET MVC 6，它是 ASP.NET 5 的一部分。

ASP.NET MVC 基于著名的 MVC(模型-视图-控制器)模式，更容易进行单元测试。它还允许把编写用户界面代码与 HTML、CSS、JavaScript 清晰地分离，它只在后台使用 C#。

注意：第 41 章介绍了 ASP.NET MVC 6。

1.7.9　Microsoft Azure

现在，在考虑开发图片时不能忽视云。虽然没有专门的章节讨论云技术，但在本书的几章中都引用了 Microsoft Azure。

Microsoft Azure 提供了软件即服务(Software as a Service，SaaS)、基础设施即服务(Infrastructure as a Service，IaaS)和平台即服务(Platform as a Service，PaaS)。下面介绍这些 Microsoft Azure 产品。

1. SaaS

SaaS 提供了完整的软件，不需要处理服务器的管理和更新等。Office 365 是一个 SaaS 产品，它通过云产品使用电子邮件和其他服务。与开发人员相关的 SaaS 产品是 Visual Studio Online，它不是在浏览器中运行的 Visual Studio。Visual Studio Online 是云中的 Team Foundation Server，可以用作私人代码库，跟踪错误和工作项，以及构建和测试服务。

2. IaaS

另一个服务产品是 IaaS。这个服务产品提供了虚拟机。用户负责管理操作系统，维护更新。当创建虚拟机时，可以决定不同的硬件产品，从共享核心开始，到最多 32 核(这个数据会很快改变)。32 核、448 GB 的 RAM 和 6144 GB 的本地 SSD 属于计算机的“G 系列”，命名为哥斯拉。

对于预装的操作系统，可以在 Windows、Windows Server、Linux 和预装了 SQL Server、BizTalk Server、SharePoint 和 Oracle 的操作系统之间选择。

作者经常给一周只需要几个小时的环境使用虚拟机，因为虚拟机按小时支付费用。如果想尝试在 Linux 上编译和运行.NET Core 程序，但没有 Linux 计算机，在 Microsoft Azure 上安装这样一个环境是很容易的。

3. PaaS

对于开发人员来说，Microsoft Azure 最相关的部分是 PaaS。可以为存储和读取数据而访问服务，使用应用程序服务的计算和联网功能，在应用程序中集成开发者服务。

为了在云中存储数据，可以使用关系数据存储 SQL Database。SQL Database 与 SQL Server 的本地版本大致相同。也有一些 NoSQL 解决方案，例如，DocumentDB 存储 JSON 数据，Storage 存储 blob(如图像或视频)和表格数据(这是非常快的，提供了大量的数据)。

Web 应用程序可以用于驻留 ASP.NET MVC 解决方案，API 应用程序可以用来驻留 ASP.NET Web API 服务。

Visual Studio Online 是开发者服务(Developer Service)产品的一部分。在这里也可以找到 Visual Studio Application Insights。它的发布周期更短，对于获得用户如何使用应用程序的信息越来越重要。

用户因为可能找不到哪些菜单，而从未使用过它们？用户在应用程序中使用什么路径来完成任务？在 Visual Studio Application Insights 中，可以得到良好的匿名用户信息，找出用户关于应用程序的问题，使用 DevOps 可以快速解决这些问题。

注意：第 20 章介绍了跟踪特性以及如何使用 Microsoft Azure 的 Visual Studio Application Insights 产品。第 45 章不仅说明了如何部署到本地 IIS 上，还描述了如何部署到 Microsoft Azure Web Apps 上。

1.8 开发工具

第 2 章会讨论很多 C#代码，而本章的最后一部分介绍开发工具和 Visual Studio 2015 的版本。

1.8.1 Visual Studio Community

这个版本的 Visual Studio 是免费的，具备以前专业版的功能。使用时间有许可限制。它对开源项目和培训、学术和小型专业团队是免费的。Visual Studio Express 版本以前是免费的，但该产品允许在 Visual Studio 中使用插件。

1.8.2 Visual Studio Professional with MSDN

这个版本比 Community 版包括更多的功能，例如 CodeLens 和 Team Foundation Server，来进行源代码管理和团队协作。有了这个版本，也会得到 MSDN 订阅，其中包括微软公司的几个服务器产品，用于开发和测试。

1.8.3 Visual Studio Enterprise with MSDN

Visual Studio 2013 有高级版和旗舰版。而 Visual Studio 2015 有企业版。这个版本提供了旗舰版的功能，但采用高级版的价格。与专业版一样，这个版本包含很多测试工具，如 Web 负载和性能测试、使用 Microsoft Fakes 进行单元测试隔离，以及编码的 UI 测试(单元测试是所有 Visual Studio 版本的一部分)。通过 Code Clone 可以找到解决方案中的代码克隆。Visual Studio 企业版还包含架构和建模工具，以分析和验证解决方案体系结构。

注意：有了 MSDN 订阅，就有权免费使用 Microsoft Azure，每月具体的数量视 MSDN 订阅的类型而定。

注意：第 17 章详细介绍了 Visual Studio 2015 几个特性的使用。第 19 章阐述单元测试、Web 测试和创建编码的 UI 测试。

注意：本书中的一些功能，如编码的 UI 测试，需要 Visual Studio 企业版。使用 Visual Studio Community 版可以完成本书的大部分内容。

1.8.4　Visual Studio Code

与其他 Visual Studio 版本相比，Visual Studio Code 是一个完全不同的开发工具。Visual Studio 2015 提供了基于项目的特性以及一组丰富的模板和工具，而 Visual Studio Code 是一个代码编辑器，几乎不支持项目管理。然而，Visual Studio Code 不仅在 Windows 上运行，也在 Linux 和 OS X 上运行。

对于本书的许多章节，可以使用 Visual Studio Code 作为开发编辑器。但不能创建 WPF、UWP 或 WCF 应用程序，也无法获得第 17 章介绍的特性。Visual Studio Code 代码可以用于.NET Core 控制台应用程序，以及使用.NET Core 的 ASP.NET Core 1.0 Web 应用程序。

可以从 http://code.visualstudio.com 下载 Visual Studio Code。

1.9　小结

本章涵盖了很多重要的技术和技术的变化。了解一些技术的历史，有助于决定新的应用程序应该使用哪些技术，现有的应用程序应该如何处理。

.NET Framework 4.6 和.NET Core 1.0 是有差异的。本章讨论了如何在这两种环境中创建并运行“Hello,World!”应用程序，但没有使用 Visual Studio。

本章阐述了公共语言运行库(CLR)的功能，介绍了用于访问数据库和创建 Windows 应用程序的技术。论述了 ASP.NET Core 1.0 的优点。

第 2 章开始使用 Visual Studio 创建“Hello,World!”应用程序，继续讨论 C#语法。