开源.NET 生态软件开发

C# 11 和.NET 7 入门与跨平台开发（第 7 版）

[美] 马克 • J. 普莱斯(Mark J. Price) 著
叶伟民 译

清華大學出版社
北 京

北京市版权局著作权合同登记号　图字：01-2023-1356

图书在版编目(CIP)数据

C# 11 和.NET 7 入门与跨平台开发：第 7 版 / (美)马克 • J. 普莱斯 (Mark J. Price) 著；叶伟民译. —北京：清华大学出版社，2024.2
(开源.NET 生态软件开发)
书名原文：C# 11 and .NET 7 – Modern Cross-Platform Development Fundamentals—7th Edition
ISBN 978-7-302-65328-8

I. ①C… II. ①马…②叶… III. ①C 语言—程序设计 ②计算机网络—程序设计 IV. ①TP312.8 ②TP393.09

中国国家版本馆 CIP 数据核字(2024)第 015041 号

责任编辑：王　军
装帧设计：孔祥峰
责任校对：成凤进
责任印制：杨　艳

出版发行：清华大学出版社
网　　址：https://www.tup.com.cn，https://www.wqxuetang.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-83470000　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：三河市东方印刷有限公司
经　　销：全国新华书店
开　　本：170mm×240mm　　**印　　张**：41　　**字　　数**：1129 千字
版　　次：2024 年 2 月第 1 版　　**印　　次**：2024 年 2 月第 1 次印刷
定　　价：158.00 元

产品编号：100941-01

译 者 序

我因为掌握了一些新技术而生活得很不错，从而整个冬季都得以在三亚度过。所以在我看来，比别人更早掌握新技术是改变自己命运、摆脱经济困境的方法之一。

前几年，在东莞.NET 俱乐部举办的一场活动中，有人问我，大多数企业为稳妥起见，不会采用最新一代的技术，那么，学习新技术的意义是什么？

我是这样回答的：没错，事实的确如此。为了稳妥，企业所用技术一般会落后一两代，但是也只会落后一两代。如果一直不学习，就会慢慢从落后一两代变成落后三代、四代，这样不知不觉就落后了十年，慢慢就会被时代淘汰。

例如，最近我和特斯拉、汇丰、渣打等大型外企的朋友交流，发现这些大企业基本上都把项目从.NET Framework 等传统技术迁移到新一代的.NET 技术了，这个具有普遍性的事实再次印证了我上面的观点。

这也是本书的意义所在。本书很可能是市面上最新的.NET 技术图书了。当然，为了能让读者尽快阅读到本书，加上译者本身水平也有限，本书错漏在所难免，请多多包涵。在我过往翻译的图书中，的确有读者指出了少量译文错误，这些读者有一部分后来成了我的朋友和我其他译作和著作的试读者。所以如果你在阅读本书时发现书中内容有错漏，敬请指正，我将不胜感激。

叶伟民

2023 年 12 月

译者简介

叶伟民

拥有 19 年.NET 软件开发经验

《.NET 内存管理宝典》等 6 本书的译者

《精通 Neo4j》一书作者之一

广州.NET 俱乐部主席

全国各地.NET 社区名录维护者之一

广州神机妙算大数据科技有限公司 CEO

美国硅谷海归

作 者 简 介

Mark J. Price 是一位拥有20多年C#编程经验的微软认证技术专家，他专注于C#编程以及构建 Azure 云解决方案。自 1993 年以来，Mark 已通过了 80 多次微软编程考试，他特别擅长传道授业。从 2001 年到 2003 年，Mark 在美国雷德蒙德全职为微软编写官方课件。当 C#还处于 alpha 版本时，他的团队就为 C#编写了第一个培训教程。在微软任职期间，他为培训师上课，指导微软认证培训师快速掌握 C#和.NET。Mark 职业生涯中的大部分时间都在培训各类学生，从 16 岁的新人到 70 岁的退休人员，其中大部分是专业开发人员。Mark 拥有计算机科学学士学位。

审校者简介

Dave Brock 是一位开发主管，具有架构、设计和开发分布式、云原生应用程序的经验。他从 DePaul University 获得了软件工程硕士学位。Dave 关注微软的技术，如.NET 和 Azure，并在自己的网站上撰写关于微软技术的文章。由于对社区做出的贡献，他已经两次获得微软 MVP 奖项。他居住在威斯康星州的麦迪逊市，当没有审校图书时，他喜欢跑步、远足和演奏音乐。他是一位骄傲的父亲，有两个非常优秀的孩子 Emma 和 Colin。

前　言

有些 C#书籍长达数千页，旨在全面介绍 C#编程语言、.NET 库和应用程序模型(如网站、服务、桌面应用和移动应用)。

本书与众不同，内容简洁清晰、行文流畅，每个主题都配有实际动手演练项目。进行总体叙述的广度是以牺牲一定深度为代价的，但如果愿意，你就会发现许多主题都值得进一步探索。

本书也是一本循序渐进的学习指南，可用于通过跨平台的.NET 学习现代 C#实践，并简要介绍 Web 开发的基础知识，以及可以使用它们构建的网站和服务。本书最适合 C#和.NET 初学者阅读，也适合学过 C#但感觉在过去几年自身技术已落伍的程序员阅读。

如果有使用旧版本 C#语言的经验，那么可以在 2.1 节查看介绍新语言特性的表格，并直接跳到相应的部分阅读。

如果有使用较旧版本的.NET 库的经验，那么可以在 7.1 节查看介绍新库特性的表格，并直接跳到相应的部分阅读。

本书将指出 C#和.NET 的一些优缺点，让你在同事面前留下深刻的印象，并快速提高工作效率。本书的解释不会事无巨细，以免因放慢速度导致读者感到无聊，而是假设读者足够聪明，能够自行对一些初、中级程序员需要了解的主题进行搜索和解释。

本书内容

第 1 章介绍如何设置开发环境，并通过 C#和.NET，使用 Visual Studio 2022 或 Visual Studio Code 创建最简单的应用程序。对于简化的控制台应用程序，将使用 C# 9 中引入的顶级程序功能。在 C# 10 及更高版本中，默认项目模板使用了顶级程序功能。为了学习如何编写简单的语言构造和库特性，在一个在线小节中介绍了.NET Interactive Notebooks 的使用。该章还介绍了可以从哪里寻求帮助，以及与我联系的方法，以便在某个问题上获得帮助，或者向我提供反馈，使我能够在 GitHub 存储库或将来的印刷版本中改进本书。

第 2 章介绍 C#的版本，并通过一些表介绍各个版本的新特性，然后解释 C#日常用来为应用程序编写源代码的语法和词汇。特别是，该章将讲述如何声明和处理不同类型的变量，还将展示 C# 11 中的原始字符串字面值特性多么有用。

第 3 章讨论如何使用操作符对变量执行简单的操作，包括比较、编写决策，C# 7 到 C# 11 中的模式匹配，以及重复语句块和类型之间的转换。该章还介绍在不可避免地发生错误时，如何编写防御性代码来处理这些错误。

第 4 章讲述如何遵循 Don’t Repeat Yourself (不要重复自己，DRY)原则，使用命令式和函数式风格编写可重用的函数。你将学习使用调试工具来跟踪和删除 bug，利用热加载在应用程序运行过程中进行修改，在执行代码时监视代码以诊断问题，以及在将代码部署到生产环境之前严格测

试代码，以删除 bug 并确保稳定性和可靠性。

第 5 章讨论类可以拥有的所有不同类别的成员，包括存储数据的字段和执行操作的方法。涉及面向对象编程(Object-Oriented Programming，OOP)概念，如聚合和封装。你将学习一些语言特性，比如元组语法支持和 out 变量，运算符和局部函数，默认的字面值和推断出的元组名称，以及如何使用 C# 9 中引入的 record 关键字、init-only 属性和 with 表达式来定义和使用不可变类型。还将介绍 C# 11 引入的 required 关键字，它可以帮助避免过度使用构造函数来控制初始化。

第 6 章解释如何使用 OOP 从现有类型派生出新的类型。你将学习如何定义委托和事件，如何实现关于基类和派生类的接口，如何覆盖类型成员以及使用多态性，如何创建扩展方法，如何在继承层次结构中的类之间进行转换，以及 C# 8 中引入的可空引用类型带来的巨大变化，并且在 C# 10 及更高版本中使其成为了默认类型。你还将学习分析器如何帮助你编写更好的代码。

第 7 章介绍.NET 的版本，并给出一些表来说明哪些版本引入了一些新特性，然后介绍与.NET Standard 兼容的.NET 类型以及它们与 C#的关系。你将学习如何在任何受支持的操作系统(Windows、macOS 和 Linux 变体)上编写和编译代码。你将学习如何打包、部署和分发自己的应用程序和库。

第 8 章讨论允许代码执行的实际任务的类型，例如操作数字和文本、在集合中存储项和使用网络(在线小节)。 还将学习正则表达式，让正则表达式变得更容易编写的一些改进方法，以及在.NET 7 中，如何使用源代码生成器来提高它们的性能。

第 9 章讨论与文件系统的交互、对文件和流的读写、文本编码、诸如 JSON 和 XML 的序列化格式，还涉及改进的功能以及 System.Text.Json 类的性能问题。

第 10 章解释如何使用名为 Entity Framework Core (EF Core)的 ORM 技术来读写关系数据库，如 Microsoft SQL Server 和 SQLite。了解如何使用数据库优先模型定义映射到数据库中现有表的实体模型，以及如何定义可以在运行时创建表和数据库的 Code First 模型(在线小节)。

第 11 章介绍 LINQ。LINQ 扩展语言增加了处理条目序列、筛选、排序，以及将它们投影到不同输出的能力。介绍.NET 6 中新引入的 LINQ 方法，如 TryGetNonEnumeratedCount 和 DistinctBy，以及.NET 7 中新引入的 LINQ 方法，如 Order 和 OrderDescending。你将了解 LINQ to XML 的特殊功能。该章的一个在线小节介绍了如何使用并行 LINQ (PLINQ)提高效率。

第 12 章介绍可以使用 C#和.NET 构建的 Web 应用程序的类型。该章还将通过构建 EF Core 模型来表示虚构组织 Northwind 的数据库。Northwind 数据库将贯穿用于本书的剩余部分。最后，介绍了常用的 Web 技术。

第 13 章介绍在服务器端通过 ASP.NET Core 使用现代 HTTP 架构构建网站的基础知识。你将学习如何实现一种 ASP.NET Core 特性(Razor Pages)，从而简化为小型网站创建动态网页的过程，还将学习如何构建 HTTP 请求和响应管道，以及如何在网站项目中启用 HTTP/3。

第 14 章讨论程序员团队如何利用 ASP.NET Core MVC 以一种易于进行单元测试和管理的方式构建大型、复杂的网站。你将了解启动配置、身份验证、路由、模型、视图和控制器。还将了解一种.NET 社区热切期盼并最终在 ASP.NET Core 7 中实现的特性：输出缓存。

第 15 章解释如何使用 ASP.NET Core Web API 构建后端 REST 体系结构 Web 服务。讨论如何使用 OpenAPI 记录和测试它们，以及如何使用工厂实例化的 HTTP 客户端正确地使用它们。该章介绍 ASP.NET Core 6 中引入的最小 API，这种 API 减少了实现简单 Web 服务时需要的代码行数。

第 16 章介绍如何使用 Blazor 构建 Web 用户界面组件，这些组件既可以在服务器端执行，又可以在 Web 浏览器中执行。该章还讨论 Blazor 服务器和 Blazor WebAssembly 的区别，以及如何

构建能够更容易地在这两种托管模型之间进行切换的组件。

结语针对进一步学习 C#和.NET 提供了一些选项。

附录 A 中提供了各章练习的解决方案(在线提供)。

要做的准备工作

可在许多平台上使用 Visual Studio Code 和命令行工具开发和部署 C#和.NET 应用程序，包括 Windows、macOS 和各种 Linux 发行版。只需要一个支持 Visual Studio Code 和互联网连接的操作系统就可以学习本书的内容。

如果更喜欢在 Windows 或 macOS 上使用 Visual Studio 2022，或者使用像 JetBrains Rider 这样的第三方工具，那么也可以使用。

下载示例代码、彩色图片、附录 A

本书代码可通过扫描封底的二维码进行下载，也可从 GitHub 存储库中获取。书中的一些屏幕截图和图表用彩色效果可能更佳，为此，我们专门制作了一份 PDF 文件，读者可通过封底二维码下载该文件。另外，本书在线提供的附录 A 给出了各章练习的答案，读者也可通过扫描封底的二维码进行下载。

目　录

第1章
C#与.NET入门

本章的目标是建立开发环境，让你了解现代.NET、.NET Core、.NET Framework、Mono、Xamarin 和.NET Standard 之间的异同，使用各种代码编辑器通过 C# 11 和.NET 7 创建尽可能简单的应用程序。另外，本章还指出了寻求帮助的方式。

本书的 GitHub 存储库包含了解决方案，这些解决方案为所有代码任务和 Notebook 使用了完整的应用项目，GitHub 存储库的地址为 https://github.com/markjprice/cs11dotnet7。

只需要按下.(点)键或在上面的链接中将.com 更改为.dev，即可将 GitHub 存储库更改为使用 GitHub Codespaces 的、基于 Visual Studio Code 的实时编辑器，如图 1.1 所示。

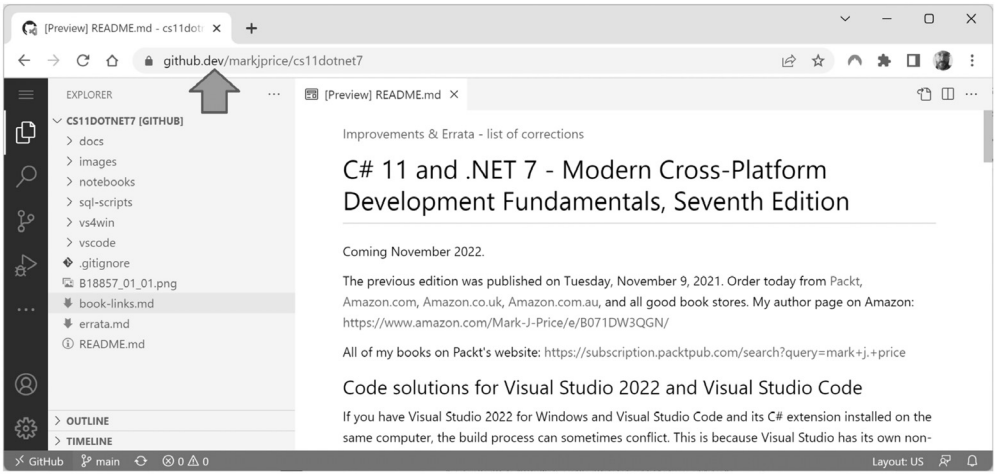

图 1.1　用于实时编辑本书的 GitHub 存储库的 GitHub Codespaces

注意：

我们提供了一个 PDF 文件，其中包含本书中使用的屏幕截图和图表的彩色版本。可以扫描本节封底的二维码下载此 PDF 文件。

完成本书的编码任务时，在浏览器中运行的 Visual Studio Code 非常适合与读者选择的代码编辑器一起运行。若有必要，可以比较自己的代码与解决方案代码，并轻松地复制和粘贴需要的部分。

注意：

读者不需要使用或者了解 Git，就能获得本书的解决方案代码。通过链接 https://github.com/markjprice/cs11dotnet7/archive/refs/heads/main.zip，可以直接下载包含所有代码解决方案的一个 ZIP 文件，然后将该 ZIP 文件解压到本地文件系统中。

本书用“现代.NET”来指代.NET 7 及其前身，如.NET 5 和.NET 6(来自.NET Core)。用“旧.NET”这个术语来指代.NET Framework、Mono、Xamarin 和.NET 标准。现代.NET 是这些传统平台和标准的统一体。

在第 1 章后，本书可以分为三大部分：第一大部分介绍 C#语言的语法和词汇；第二大部分介绍.NET 中用于构建应用程序功能的可用类型；第三大部分介绍可以使用 C#和.NET 构建的一些常见的跨平台网站、服务和浏览器应用程序。

大多数人学习复杂主题的最佳方式是模仿和重复，而不是阅读关于理论的详细解释。因此，本书不会对每一步都做详细解释，而是编写一些代码，然后观察代码的运行。

你不需要立即知道所有细节。随着时间的推移，你将学会创建自己的应用程序，你所获得的知识将超越任何书籍所能教你的。

借用 1755 年版《英语词典》的作者 Samuel Johnson 的话来说，我犯了“一些愚蠢的错误，书中有一些可笑的荒谬之处，这些错误和荒谬之处是任何综合性作品都无法避免的。”我对这些问题负全部责任，希望你能理解我面临的挑战；为了解决这些问题，我所编写的这本书涉及一些快速发展的技术(如 C#和.NET)，而读者可以用它们构建应用程序。

本章涵盖以下主题：

- 设置开发环境
- 理解.NET
- 使用 Visual Studio 2022 构建控制台应用程序
- 使用 Visual Studio Code 构建控制台应用程序
- 使用.NET Interactive Notebook 探索代码
- 检查项目的文件夹和文件
- 充分利用本书的 GitHub 存储库
- 寻求帮助

1.1 设置开发环境

在开始编程前，需要准备一款针对 C#的代码编辑器。微软提供了一系列代码编辑器和集成开发环境(Integrated Development Environment，IDE)，包括：

- Visual Studio 2022 for Windows
- Visual Studio 2022 for Mac
- Visual Studio Code (用于 Windows、macOS 或 Linux)
- Visual Studio Code for the Web
- GitHub Codespaces

第三方已经创建了自己的 C#代码编辑器，如 JetBrains Rider。

1.1.1　选择适合学习的工具和应用程序类型

学习 C#和.NET 最好的工具和应用程序类型是什么?

在学习时，最好使用能够帮助编写代码和配置，但不会隐藏实际情况的工具。IDE 提供了易用的图形用户界面，但它们到底在底层做了些什么工作呢？一个更接近实际操作、更基本的代码编辑器也可为编写代码提供帮助，在学习过程中效果更好。

话虽如此，可以认为最好的工具是已经熟悉的工具，或者团队用作日常开发工具的工具。出于这个原因，希望读者可以自由选择任何 C#代码编辑器或 IDE 来完成本书中的编码任务，包括 Visual Studio Code、Visual Studio for Windows、Visual Studio for Mac，甚至 JetBrains Rider。

第 1 章详细说明了如何在 Visual Studio 2022 for Windows 和 Visual Studio Code 中创建多个项目。之后，给出了与所有工具一起工作的项目名称和通用说明，以便你使用自己喜欢的任何工具。

为学习 C#语言构造和许多.NET 库，最好编写不会因不必要的代码而分心的应用程序。例如，不需要仅仅为了学习如何编写 switch 语句而创建整个 Windows 桌面应用程序或网站。

因此，学习第 1~11 章中的 C#和.NET 主题的最好方法是构建控制台应用程序。此后，在第 12~16 章，将构建网站、服务和 Web 浏览器应用程序。

1. .NET Interactive Notebooks 扩展的优缺点

Visual Studio Code 的另一个好处是引入了.NET Interactive Notebooks 扩展(注意，已被更名为 Polyglot Notebooks)。这个扩展为编写代码片段提供了一个简单、安全的地方，可用于实验和学习的目的。例如，数据科学家使用 Notebooks 分析和可视化数据。学生使用它们学习如何编写小段代码，从而熟悉语言构造和探索 API。

.NET Interactive Notebooks 能够创建一个简单的 Notebook 文件，混合了 Markdown 的“单元格”(格式丰富的文本)和 C#以及其他相关语言代码，如 PowerShell、F#和 SQL(用于数据库)。

然而，.NET Interactive Notebooks 确实存在一些限制：

- 无法用于创建网站、服务和应用。
- 无法读取用户输入，例如，不能使用 ReadLine 或 ReadKey。
- 不能将参数传递给它们。
- 不允许定义自己的名称空间。
- 没有任何调试工具(但将来会有)。

2. 使用 Visual Studio Code 进行跨平台开发

可以选择的最现代、最轻量级的代码编辑器是 Visual Studio Code，这也是唯一一个来自微软的跨平台代码编辑器。Visual Studio Code 可以运行在所有常见的操作系统中，包括 Windows、macOS 和许多 Linux 发行版，如 Red Hat Enterprise Linux (RHEL)和 Ubuntu。

Visual Studio Code 是现代跨平台开发代码的优秀选择，因为它提供了一个广泛的、不断增长的扩展集来支持除 C#外的多种语言。

Visual Studio Code 是跨平台的、轻量级的，可安装在所有平台上(应用程序将被部署到这些平台上)，可以快速修复 bug，等等。选择 Visual Studio Code 意味着开发者可以使用跨平台代码编辑器来开发跨平台应用程序。

Visual Studio Code 对 Web 开发提供了强大的支持，尽管它目前对移动和桌面开发的支持很弱。

ARM 处理器支持 Visual Studio Code，这样就可以在 Apple Silicon 计算机和 Raspberry Pi 上进行开发。

Visual Studio Code 也是目前最流行的集成开发环境，根据 Stack Overflow 在 2021 年所做的调查，超过 70%的专业开发者选择了它。

3. 使用 GitHub Codespaces 进行云开发

GitHub Codespaces 是一个基于 Visual Studio Code 的完全配置的开发环境，可以在云环境中运行，并通过任何 Web 浏览器访问。它支持 Git 存储库、扩展和内置命令行界面，因此可以从任何设备进行编辑、运行和测试。

4. 使用 Visual Studio for Mac 进行通用开发

Microsoft Visual Studio 2022 for Mac 版可以创建大多数类型的应用程序，包括控制台应用程序、网站、Web 服务、桌面应用程序和移动应用程序。

要为苹果操作系统(如 iOS)编译应用程序，以便在 iPhone 和 iPad 等设备上运行，就必须有 Xcode。Xcode 只能在 macOS 上运行。

5. 使用 Visual Studio for Windows 进行通用开发

Microsoft Visual Studio 2022 for Windows 可以创建大多数类型的应用程序，包括控制台应用程序、网站、Web 服务、桌面应用程序和移动应用程序。尽管可以使用 Visual Studio 2022 for Windows 及其 Xamarin 扩展来编写跨平台移动应用程序，但仍需要 macOS 和 Xcode 编译它。

Visual Studio for Windows 只能在 Windows 7 SP1 或更新版本上运行。必须在 Windows 10 或 Windows 11 上运行它才能创建通用 Windows 平台(Universal Windows Platform，UWP)应用程序，这些应用程序是从微软商店安装的，并在沙箱中运行，以保护计算机。

6. 我使用了什么

为了编写和测试本书的代码，使用的硬件如下：

- HP Spectre(英特尔)Notebook 计算机
- Apple Silicon Mac mini (M1)台式计算机
- Raspberry Pi 400 (ARM v8)台式计算机

使用的软件如下：

- Visual Studio Code
  - Apple Silicon Mac mini (M1)台式计算机上的 macOS 操作系统
  - HP Spectre(英特尔)笔记本计算机上的 Windows 11
  - Raspberry Pi 400 台式计算机上的 Ubuntu 64
- Visual Studio 2022 for Windows
  - HP Spectre(英特尔)Notebook 计算机上的 Windows 11
- Visual Studio 2022 for Mac
  - Apple Silicon Mac mini (M1)台式计算机上的 macOS 操作系统

希望读者可以访问各种各样的硬件和软件，因为明白了平台的差异可以加深对开发挑战的理

解，不过上述任何一个组合足以学习 C#和.NET 的基础知识并了解如何构建实际的应用程序和网站。

更多信息：

可以通过阅读我在以下链接撰写的文章，了解如何在安装了 Ubuntu Desktop 64 位的 Raspberry Pi 400 上使用 C#和.NET 编写代码：https://github.com/markjprice/cs11dotnet7/tree/main/docs/raspberry-pi-ubuntu64。

1.1.2　跨平台部署

为开发选择的代码编辑器和操作系统并不会限制代码的部署位置。

.NET 7 支持以下部署平台。

- **Windows**：Windows 7 SP1 或更新版本，Windows 8.1 或更新版本，Windows 10 版本 1607 或更新版本，Windows 11 版本 22000 或更新版本、Windows Server 2012 R2 SP1 或更新版本，Nano Server 版本 1809 或更新版本。
- **macOS**：macOS Catalina 版本 10.15 或更新版本。
- **Linux**：Alpine Linux 3.15 或更新版本，CentOS 7 或更新版本，Debian 10 或更新版本，Fedora 33 或更新版本，openSUSE 15 或更新版本，Red Hat Enterprise Linux (RHEL) 7 或更新版本，SUSE Enterprise Linux 12 SP2 或更新版本，Ubuntu 18.04 或更新版本。注意，现在 Ubuntu 22.04 中已经包含了.NET 6，可以阅读以下链接来了解更多信息：https://devblogs.microsoft.com/dotnet/dotnet-6-is-now-in-ubuntu-2204/。
- **Android**：API 21 或更新版本。
- **iOS**：10.0 或更新版本。

在.NET 5 及后续版本中支持 Windows ARM 64 意味着可以在微软 Surface Pro X 等 Windows ARM 设备上进行开发和部署，但在 Apple M1 Mac 上使用 Parallels 和 Windows 11 ARM 虚拟机进行开发的速度是前者的两倍!

更多信息：

通过以下链接可以了解最新支持的操作系统和版本：https://github.com/dotnet/core/blob/main/release-notes/7.0/supported-os.md。

1.1.3　下载并安装 Visual Studio 2022 for Windows

许多专业微软开发人员在日常开发工作中使用 Visual Studio 2022 for Windows。即使选择使用 Visual Studio Code 来完成本书中的编码任务，也可能需要熟悉 Visual Studio 2022 for Windows。只有当使用一个工具编写了一定数量的代码后，才能真正判断这个工具能否满足你的需求。

如果没有 Windows 计算机，那么可以跳过本节，继续学习下一节，在 macOS 或 Linux 操作系统中下载并安装 Visual Studio Code。

自 2014 年 10 月以来，微软已经为学生、开源贡献者和个人免费提供了专业的、高质量的 Visual Studio for Windows 版本。它被称为社区版。任何版本都适合本书。如果尚未安装，现在就安装它。

(1) 从以下链接下载 Microsoft Visual Studio 2022 17.4 或更新版本(用于 Windows)：https://visualstudio.microsoft.com/downloads/。

(2) 启动安装程序。

(3) 在 Workloads 选项卡上，选择以下内容：

- ASP.NET and web development
- .NET desktop development(它包括 Console Apps)

(4) 在 Individual components 选项卡的 Code tools 部分，选择以下内容：

- Git for Windows

(5) 单击 Install 并等待安装程序获取选定的软件并安装。

(6) 安装完成后，单击 Launch。

(7) 第一次运行 Visual Studio 时，系统会提示登录。如果你有微软账户，就使用该账户。如果没有微软账户，就通过以下链接注册一个新的账户：https://signup.live.com/。

(8) 第一次运行 Visual Studio 时，系统会提示配置环境。对于 Development Settings，选择 Visual C#。对于颜色主题，我选择了 Blue，但你可以选择任何你喜欢的颜色。

(9) 如果想自定义键盘快捷键，可导航到 Tools | Options…，然后选择 Keyboard 部分。

Microsoft Visual Studio for Windows 键盘快捷键

本书避免显示键盘快捷键，因为它们通常是自定义的。如果它们在代码编辑器中是一致的且经常使用，本书将尝试展示它们。若想识别和定制键盘快捷键，可访问如下链接了解相关信息：https://docs.microsoft.com/en-us/visualstudio/ide/identifying-and-customizing-keyboard-shortcutsin-visual-studio。

1.1.4 下载并安装 Visual Studio Code

在过去几年，Visual Studio Code 得到了极大改进，它的受欢迎程度让微软公司感到惊喜。如果读者很勇敢，喜欢挑战，那么有一个内部版可用，这是下一个版本的每日构建版。

即使计划只使用 Visual Studio 2022 for Windows 进行开发，也建议下载并安装 Visual Studio Code 并尝试使用它完成本章的编码任务，然后决定是否坚持在本书的剩余部分只使用 Visual Studio 2022。

现在，可下载并安装 Visual Studio Code、.NET SDK、C#和.NET Interactive Notebooks 扩展，步骤如下。

(1) 从以下链接下载并安装 Visual Studio Code 的稳定版本或内部版本：https://code.visualstudio.com/。

更多信息：

如果需要有关安装 Visual Studio Code 的更多帮助，可通过以下链接阅读官方安装指南：https://code.visualstudio.com/docs/setup/setup-overview。

(2) 从以下链接下载并安装.NET SDK 6.0 和 7.0：https://www.microsoft.com/net/download。

更多信息：

为了充分学习如何控制.NET SDK，需要安装多个版本。.NET 6.0 和.NET 7.0 是目前支持的两个版本。虽然.NET 6.0 不是最新版本，但它是最新的长期支持(Long-Term Support，LTS)版本，所以这是安装它的另一个好理由。

(3) 要安装 C#扩展，必须先启动 Visual Studio Code 应用程序。

(4) 在 Visual Studio Code 中，单击 Extensions 图标或导航到 View | Extensions。

(5) C#扩展是最流行的扩展之一，在列表的顶部应该能够看到它；也可以在搜索框中输入 C#。

(6) 单击 Install，等待下载和安装支持包。

(7) 在搜索框中输入.NET Interactive(注意，已更名为 Polyglot Notebooks)，找到.NET Interactive Notebooks 扩展。

(8) 单击 Install 并等待它安装。

1. 安装其他扩展

本书后续章节将使用更多 Visual Studio Code 扩展，如果想现在安装它们，可参照表 1-1。

表 1.1　本书用到的其他扩展

扩展名和标识符	说明
C# for Visual Studio Code (由 OmniSharp 提供支持) ms-dotnettools.csharp	提供 C#编辑支持。包括语法高亮，智能感知，Go To Definition，查找所有引用，对.NET 的调试支持，以及在 Windows、macOS 和 Linux 中对 csproj 项目的支持
.NET Interactive Notebooks ms-dotnettools.dotnet-interactive-vscode	这个扩展增加了在 Visual Studio Code Notebook 中使用.NET Interactive 的支持。它依赖于 Jupyter 扩展(ms-toolsai.jupyter)，Jupyter 扩展又有自己的依赖项
MSBuild 项目工具 tintoy.msbuild-project-tools	为 MSBuild 项目文件提供智能感知功能，包括<PackageReference>元素的自动完成
REST Client humao.rest-client	发送 HTTP 请求并在 Visual Studio Code 中直接查看响应
ilspy-vscode icsharpcode.ilspy-vscode	反编译 MSIL 程序集——支持现代.NET、.NET Framework、.NET Core 和.NET Standard

2. 使用命令行管理 Visual Studio Code 扩展

可以在命令行或终端安装 Visual Studio Code 扩展，如表 1.2 所示。

表 1.2　安装 Visual Studio Code 的命令及说明

命令	说明
code --list-extensions	列举已安装的扩展
code --install-extension <extension-id>	安装指定的扩展
code --uninstall-extension <extension-id>	卸载指定的扩展

例如，要安装 C#扩展，可以在命令行或终端输入下面的命令：

```
code --install-extension ms-dotnettools.csharp
```

更多信息：

我创建了一个 PowerShell 脚本，用于安装前面的表中列出的所有 Visual Studio Code 扩展。你可以通过下面的链接找到这个脚本: https://github.com/markjprice/cs11dotnet7/blob/main/vscode/Scripts/install-vs-code-extensions.ps1。

3. 理解 Microsoft Visual Studio Code 版本

微软公司几乎每个月都会发布 Visual Studio Code 的新特性版本，并且会更频繁地发布 bug 修复版本。例如：

- 1.64.0 版，2022 年 2 月发布的新特性版本。
- 1.64.1 版，2021 年 8 月发布的 bug 修复版本。

本书使用的是 1.71.0 版，是 2022 年 9 月发布的新特性版本，但是 Visual Studio Code 版本不如稍后安装的 C# for Visual Studio Code 扩展版本重要。

C#扩展虽然不是必需的，但它提供了输入时的智能感知、代码导航和调试等功能，因此十分有必要安装。所以安装和更新它非常方便，以支持最新的 C#语言特性。

4. Microsoft Visual Studio Code 快捷键

本书将避免显示用于“创建新文件”等任务的键盘快捷键，因为它们在不同的操作系统中通常是不同的。书中显示键盘快捷键的情景是需要重复按下相应键时，例如调试时；这些快捷键也尽可能在操作系统之间保持一致。

如果想为 Visual Studio Code 定制键盘快捷键，可访问如下链接来学习如何定制：

https://code.visualstudio.com/docs/getstarted/keybindings。

建议根据使用的操作系统下载一份 PDF 格式的操作系统快捷键。

- Windows：https://code.visualstudio.com/shortcuts/keyboard-shortcuts-windows.pdf。
- macOS：https://code.visualstudio.com/shortcuts/keyboard-shortcuts-macos.pdf。
- Linux：https://code.visualstudio.com/shortcuts/keyboard-shortcuts-linux.pdf。

1.2 理解.NET

.NET 7、.NET Core、.NET Framework 和 Xamarin 是相关的，它们是开发人员用来构建应用程序和服务的平台。本节将介绍这些.NET 概念。

1.2.1 理解.NET Framework

.NET Framework 开发平台包括公共语言运行库(Common Language Runtime，CLR)和基类库(Base Class Library，BCL)，前者负责管理代码的执行，后者提供了丰富的类库来构建应用程序。

微软公司最初设计.NET Framework 是为了使应用具有跨平台的可能性，但是微软公司在实现过程中，发现这一平台在 Windows 上工作得最好。

自.NET Framework 4.5.2 成为 Windows 操作系统的官方组件以来，由于组件与父产品的支持相同，因此 4.5.2 及后续版本的组件遵循其所在 Windows 操作系统的生命周期策略。.NET Framework 已经安装在超过 10 亿台计算机上，所以对它的改动必须尽可能少。即使是修复 bug 也

会导致问题，所以更新频率很低。

对于.NET Framework 4.0 或更新版本，在计算机中，为.NET Framework 编写的所有应用程序都共享相同版本的 CLR 以及存储在全局程序集缓存(Global Assembly Cache，GAC)中的库，如果其中一些应用程序需要特定版本以保证兼容性，就会出问题。

最佳实践：

实际上，.NET Framework 仅适用于 Windows，因为是旧平台，所以不建议使用它创建新的应用程序。

1.2.2　理解 Mono、Xamarin 和 Unity 项目

一些第三方开发了名为 Mono 项目的 .NET Framework 实现。Mono 是跨平台的，但是它远远落后于.NET Framework 的官方实现。

Mono 作为 Xamarin 移动平台以及 Unity 等跨平台游戏开发平台的基础，已经找到了自己的价值所在。

微软公司在 2016 年收购了 Xamarin，并在 Visual Studio 中免费提供曾经昂贵的 Xamarin 扩展。微软公司将只能创建移动应用程序的 Xamarin Studio 开发工具更名为 Visual Studio for Mac，并赋予它创建其他类型的应用程序(如控制台应用程序和 Web 服务)的能力。

有了 Visual Studio 2022 for Mac，微软公司就能将 Xamarin Studio 编辑器的部分功能替换为 Visual Studio 2022 for Windows 的部分功能，以提供更接近的体验和性能。Visual Studio 2022 for Mac 也被重写为一个真正的本地 macOS UI 应用程序，以提高可靠性并与 macOS 的内置辅助技术一起工作。

1.2.3　理解.NET Core

如今，我们生活在真正跨平台的世界里，现代移动技术和云计算的发展使得 Windows 作为操作系统变得不那么重要了。正因为如此，从 2015 年开始，微软公司一直致力于将.NET 从它与 Windows 的紧密联系中分离出来。在将.NET Framework 重写为真正跨平台的同时，微软公司也利用这次机会重构并删除了不再被认为核心的主要部分。

这个新的、现代化的产品一开始被命名为.NET Core，其中包括名为 CoreCLR 的 CLR 跨平台实现和名为 CoreFX 的流畅 BCL。

微软公司负责.NET 的项目经理 Scott Hunter 认为：“.NET Core 客户中有 40%是全新的平台开发人员，这正是我们想要的结果。我们想引进新人。”

.NET Core 的变化很快，因为可以与应用程序并行部署，所以.NET Core 可以频繁地更改，因为这些更改不会影响同一台计算机上的其他.NET Core 应用程序。微软公司对.NET Core 和现代.NET 所做的大多数改进都无法轻易添加到.NET Framework 中。

1.2.4　了解走向.NET 的过程

在 2020 年 5 月的 Microsoft Build 开发者大会上，.NET 团队宣布，.NET 的统一化延迟了。他们说，将会在 2020 年 11 月 10 日发布.NET 5，它会将除移动平台的所有.NET 平台统一起来。直到 2021 年 11 月发布的.NET 6，统一的.NET 平台才支持移动设备。但是，在 2021 年 9 月，他们

宣布将.NET MAUI 延迟 6 个月，这是他们针对移动和桌面应用开发提供的新的、跨平台的平台。.NET MAUI 最终在 2022 年 5 月发布公众可用(General Availability，GA)版本。从以下链接可阅读关于 MAUI 的公告：https://devblogs.microsoft.com/dotnet/introducing-dotnet-maui-one-codebase-manyplatforms/。

.NET Core 已重命名为.NET，主版本号则跳过了数字 4，以免与.NET Framework 4.x 混淆。微软公司计划每年 11 月发布主版本，就像苹果在每年 9 月发布 iOS 的主版本一样。

表 1.3 显示了现代.NET 的重要版本是什么时候发布的，计划什么时候发布未来的版本，以及本书的各个版本使用的是哪个.NET 版本。

表 1.3 对比.NET 的不同版本

.NET 版本	发布日期	本书的版本	本书英文版的出版日期
.NET Core RC1	2015 年 11 月	第 1 版	2016 年 3 月
.NET Core 1.0	2016 年 6 月		
.NET Core 1.1	2016 年 11 月		
.NET Core 1.0.4 和.NET Core 1.1.1	2017 年 3 月	第 2 版	2017 年 3 月
.NET Core 2.0	2017 年 8 月		
.NET Core for UWP in Windows 10 Fall Creators Update	2017 年 10 月	第 3 版	2017 年 11 月
.NET Core 2.1 (LTS)	2018 年 5 月		
.NET Core 2.2 (Current)	2018 年 12 月		
.NET Core 3.0 (Current)	2019 年 9 月	第 4 版	2019 年 10 月
.NET Core 3.1 (LTS)	2019 年 12 月		
Blazor WebAssembly 3.2 (Current)	2020 年 5 月		
.NET 5.0 (Current)	2020 年 11 月	第 5 版	2020 年 11 月
.NET 6.0 (LTS)	2021 年 11 月	第 6 版	2021 年 11 月
.NET 7.0 (Standard)	2022 年 11 月	第 7 版	2022 年 11 月
.NET 8.0 (LTS)	2023 年 11 月	第 8 版	2023 年 11 月
.NET 9.0 (Standard)	2024 年 11 月	第 9 版	2024 年 11 月
.NET 10.0 (LTS)	2025 年 11 月	第 10 版	2025 年 11 月

了解 Blazor WebAssembly 的版本

.NET Core 3.1 包含了用于构建 Web 组件的 Blazor 服务器。微软公司也曾计划在该版本中包含 Blazor WebAssembly，但被推迟了。Blazor WebAssembly 后来作为.NET Core 3.1 的可选附加组件发布。我把它包含在上表中，是因为它被版本化为 3.2，以便从.NET Core 3.1 的 LTS 中排除。

1.2.5 了解.NET 支持

.NET 版本可以是长期支持的(LTS)版本，也可以是标准支持的版本，即以前的“当前(Current)版本”，还可以是预览的(Preview)版本。下面解释了这 3 种版本：

- LTS 版本是稳定的，在其生命周期中很少需要更新。对于不打算频繁更新的应用程序，这是不错的选择。LTS 版本将在 GA 版本发布后的 3 年内受到微软支持，或下一个 LTS 版本发布后的 1 年内受到支持(以二者中较长的为准)。

- Standard 或 Current 版本包含可根据反馈进行更改的功能。对于正在积极开发的应用程序，这是很好的选择，因为它们提供了最新的改进。Standard 版本将在 GA 版本发布后的 18 个月内受到微软公司支持，或下一个 Standard 或 LTS 版本发布后的 6 个月内受到支持(以二者中较长的为准)。
- Preview 版本用于公众测试。对于想要使用最新技术的有冒险精神的程序员，或者想要及早了解新的语言特性、库和应用平台的编程图书作者来说，这是很好的选择。微软公司不为 Preview 版本提供支持，但是 Preview 或 Release Candicate(RC)版本可能被宣布为 Go Live(上线)，这意味着微软公司会在生产环境中为它们提供支持。

.NET 在整个生命周期中，都要接受安全性和可靠性方面的关键补丁。必须更新最新的补丁才能获得支持。例如，如果系统运行的是 1.0.0 版本，但微软公司已发布了 1.0.1 版本，就需要安装 1.0.1 版本来获得支持。

更多信息：

End of support 或 End of life(结束支持)指的是在这个日期之后，微软公司不再提供 bug 修复、安全更新和技术帮助。

为了帮助你更好地理解 Standard/Current 和 LTS 版本，使用色条对它们进行直观的表示是很有帮助的。对于 LTS 版本，使用 3 年长的黑色条；对于 Standard/Current 版本，使用可变长度的灰色条，其结尾的网纹表示在发布新的主版本或小版本后，对它们继续提供支持的 6 个月时间，如图 1.2 所示。

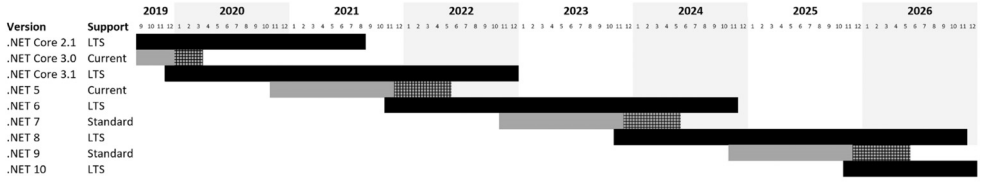

图 1.2　Standard 和 LTS 版本的支持时间

例如，如果使用.NET 5.0 创建项目，而微软公司在 2021 年 11 月 8 日发布了.NET 6.0，就需要在 2022 年 5 月 8 日之前将项目升级到.NET 6.0，这样才能获得.NET 的更新和修复，并得到微软公司的支持。

如果需要微软公司的长期支持，那么现在选择.NET 6.0(直到.NET 8.0 发布)，即使微软公司已发布了.NET 7.0 也是如此。这是因为.NET 7.0 是 Standard 版本，因此它将先于.NET 6.0 失去支持。只要记住，即使使用 LTS 版本，也必须升级到 bug 修复版本，如.NET 6.0.9 和.NET SDK 6.0.401(它们发布于 2022 年 9 月 13 日)，因为微软公司每个月都会发布更新。

在本书英文版于 2022 年 11 月出版时，除了下面列出的版本(按停止支持的日期排序)，其他.NET Core 和现代.NET 版本都已走到了尽头：

- .NET Core 3.1 于 2022 年 12 月 3 日停止支持。
- .NET 7.0 将于 2024 年 5 月 14 日停止支持。
- .NET 6.0 将于 2024 年 11 月 12 日停止支持。

更多信息：

通过访问以下链接，可以了解当前支持的.NET 版本，以及它们将在什么时候停止支持：https://github.com/dotnet/core/blob/main/releases.md。

1. 了解.NET Runtime 和.NET SDK 版本

.NET Runtime 版本控制遵循语义版本控制。也就是说，主版本表示非常大的更改，次版本表示新特性，而补丁版本表示 bug 的修复。

.NET SDK 版本控制不遵循语义版本控制。主版本号和次版本号与匹配的运行时版本绑定。补丁版本遵循的约定指明了.NET SDK 的主版本和次版本，如表 1.4 所示。

表 1.4 .NET SDK 版本不遵循语义版本控制

变更	运行时	SDK
初始版本	7.0.0	7.0.100
SDK bug 修复	7.0.0	7.0.101
运行时和 SDK bug 修复	7.0.1	7.0.102
SDK 新特性	7.0.1	7.0.200

2. 列举和删除.NET 的版本

.NET Runtime 更新与主版本(如 7.x 版)兼容。.NET SDK 的更新版本保留了构建适用于旧版运行时的应用程序的能力，这使得安全删除旧版.NET 成为可能。

执行以下命令后，就可以看到目前安装了哪些 SDK 和运行时：

```
dotnet --list-sdks
dotnet --list-runtimes
dotnet --info
```

在 Windows 上，可使用 App & features 部分删除.NET SDK。

在 macOS 或 Windows 上，可使用 dotnet-core-uninstall 工具删除.NET SDK。这个工具默认没有安装。

例如，在编写本书第 4 版时，笔者每个月都执行以下命令：

```
dotnet-core-uninstall remove --all-previews-but-latest --sdk
```

1.2.6 现代.NET 的区别

与单一的传统.NET Framework 相比，现代的.NET 是模块化的，是开源的，微软公司决定在开放中加以改进。微软公司在改进现代.NET 的性能方面付出了特别多的努力。

现代.NET 比.NET Framework 的最新版本要小，因为非跨平台的旧技术已被移除。例如，Windows Forms 和 Windows Presentation Foundation (WPF) 可用于构建 GUI 应用程序，但它们与 Windows 生态系统紧密相连，因此已从 macOS 和 Linux 的.NET 中移除。

1. Windows 桌面开发

现代.NET 的一大特性就是支持使用 Windows Desktop Pack 运行旧的 Windows 窗体和 WPF 应用程序；Windows Desktop Pack 是.NET Core 3.1 或更新版本的 Windows 版本附带的组件，这也就是它比用于 macOS 和 Linux 的 SDK 更大的原因。如有必要，可对旧的 Windows 桌面应用做一些小的改动；还可以为现代.NET 重新构建应用程序，以利用新的特性和性能改进。本书不讨论 Windows 桌面开发。

2. Web 开发

ASP.NET Web Forms 和 Windows Communication Foundation (WCF) 是旧的 Web 应用开发和服务技术，现在很少有开发人员选择在新的开发项目中使用它们，所以它们也从现代.NET 中被移除。相反，开发人员更喜欢使用 ASP.NET MVC、ASP.NET Web API、SignalR 和 gRPC。这些技术已经重组并结合成一个运行在现代.NET 上的新产品，名为 ASP.NET Core。

第 12~16 章将介绍 Web 开发技术。

更多信息：

一些.NET Framework 开发人员对现代.NET 中没有 ASP.NET Web Forms、WCF 和 Windows Workflow (WF) 感到非常失望，并且希望微软公司能改变这一状况。一些开源项目支持将 WCF 和 WF 迁移到现代.NET。可通过以下链接阅读更多相关内容：https://devblogs.microsoft.com/dotnet/supporting-the-community-with-wf-and-wcf-oss-projects/。CoreWCF 1.0 于 2022 年 4 月发布：https://devblogs.microsoft.com/dotnet/corewcfv1-released/。以下链接提供一个使用了 Blazor Web Forms 组件的开源项目：https://github.com/FritzAndFriends/BlazorWebFormsComponents。

3. 数据库开发

Entity Framework 6 是一种对象-关系映射技术，用于处理存储在关系数据库(如 Oracle 和 Microsoft SQL Server)中的数据。多年来，Entity Framework 一直背负着沉重的包袱，因此这一跨平台 API 被精简了，并将支持非关系数据库(如 Azure Cosmos DB)，微软公司将之重命名为 Entity Framework Core，详见第 10 章。

如果现有的应用程序使用旧的 Entity Framework，那么可以知道，.NET Core 3.0 或更新版本支持 Entity Framework 6.3。

1.2.7　了解.NET Standard

2019 年，.NET 的情况是，微软公司控制着三个.NET 平台分支，如下所示。

- .NET Core：用于跨平台和新应用程序。
- .NET Framework：用于旧应用程序。
- Xamarin：用于移动应用程序。

以上每种.NET 平台都有优点和缺点，它们都是针对不同的场景设计的。这导致如下问题：开发人员必须学习三个.NET 平台，每个.NET 平台都有令人讨厌的怪癖和限制。

因此，微软公司定义了.NET Standard，这是所有.NET 平台都可以实现的一套 API 规范，用于指示兼容性级别。例如，与.NET Standard 1.4 兼容的平台表明提供基本的支持。

在.NET Standard 2.0 及后续版本中，微软公司已将这三个.NET 平台融合到现代的最低标准，这使开发人员可以更容易地在任何类型的.NET 之间共享代码。

在.NET Core 2.0 及后续版本中，微软公司增加了许多缺失的 API，开发人员需要将你为.NET Framework 编写的旧代码移植到跨平台的.NET Core 中。有些 API 已经实现了，但会抛出异常，指示开发人员不应该实际使用它们！这通常是由于运行.NET 的操作系统不同而导致的。第 2 章将介绍如何处理这些特定于平台的异常。

理解.NET Standard 只是一种标准是很重要的。你不能安装.NET Standard，就像不能安装 HTML5 一样。要使用 HTML5，就必须安装实现了 HTML5 标准的 Web 浏览器。

要使用.NET Standard，就必须安装实现了.NET Standard 规范的.NET 平台。.NET Standard 2.1 是由.NET Core 3.0、Mono 和 Xamarin 实现的。C# 8.0 的一些特性需要.NET Standard 2.1，.NET Framework 4.8 没有实现.NET Standard 2.1。

随着.NET 5 和更新版本的发布，对.NET Standard 的需求大大减少，因为有了适用于所有平台(包括移动平台)的.NET。现代.NET 有一个基类库和两个运行时：CoreCLR 针对服务器或桌面场景(如网站和 Windows 桌面应用)进行了优化，Mono 运行时针对资源有限的移动和 Web 浏览器应用进行了优化。

2021 年 8 月，Stephen Toub(软件工程师合作伙伴，.NET 方向)撰写了文章“Performance Improvements in .NET 6。”在这篇文章中关于 Blazor 和 Mono 的部分，他写道：

运行时自身被编译为 WASM，下载到浏览器，并用于执行应用程序及其依赖的库代码。对于.NET 来说，运行时实际上有好几种。在.NET 6 中，针对各种.NET 应用模型(无论是控制台应用、ASP.NET Core 应用、Blazor WASM 还是移动应用)的所有.NET 核心库都来自 dotnet/runtime 中的共同来源，但 dotnet/runtime 中实际上有 coreclr 和 mono 两种运行时实现。

更多信息：

从以下链接可以阅读关于这两个运行时的更多信息：https://devblogs.microsoft.com/dotnet/performance-improvements-in-net-6/#blazor-and-mono。

即使是现在，为.NET Framework 创建的应用程序和网站仍需要得到支持，因此，理解如何创建.NET Standard 2.0 类库很重要，这些类库要向后兼容旧的.NET 平台。

更多信息：

.NET Standard 现在已正式成为遗留技术。不会再有新的.NET Standard 版本，所以它的 GitHub 存储库已被封存，从以下推文可以了解更多信息：https://twitter.com/dotnet/status/1569725004690128898。

1.2.8 本书使用的.NET 平台和工具

本书的第 1 版写于 2016 年 3 月，作者主要关注.NET Core 功能，但对于.NET Core 还没有实现重要或有用的功能，作者使用了.NET Framework，因为那时还没有发布.NET Core 1.0 的最终版本。书中的大多数例子都使用了 Visual Studio 2015，只是简单地展示了 Visual Studio Code。

本书的第 2 版(几乎)完全清除了所有.NET Framework 代码示例，以便读者能够关注真正跨平台运行的.NET Core 1.1 示例，并且.NET Core 1.1 是 LTS 版本。

本书的第 3 版完成了转换，所有代码都是完全使用.NET Core 2.0 编写的。但是，由于要为 Visual Studio Code 和 Visual Studio 2017 中的所有任务提供详细的指令，因此增加了复杂性。

在本书的第 4 版中，除了最后两章，只展示如何使用 Visual Studio Code 编写代码示例，从而延续这一趋势。第 20 章需要使用运行在 Windows 10 上的 Visual Studio，第 21 章则需要使用运行在 macOS 上的 Visual Studio。

在本书的第 5 版中，原来的第 20 章变成了在线附录 B，以便为介绍 Blazor 的新内容腾出空间。Blazor 项目可以使用 Visual Studio Code 创建。

在第 6 版中，第 19 章已更新，以展示如何使用 Visual Studio 2022 和.NET MAUI(Multi-platform App UI，多平台应用程序 UI)创建移动和桌面跨平台应用程序。

在这个第 7 版中，作者重新让本书关注语言、库和 Web 开发 3 个领域的基础知识。读者能够为书中的所有示例使用 Visual Studio Code，但也可使用自己选择的其他任何代码编辑器。

1.2.9　Apps and Services with .NET 7 中涵盖的主题

作者的新书 *Apps and Services with .NET 7* 中涵盖了以下主题。

- 数据：SQL Server、Azure Cosmos DB。
- 库：日期、时间、时区和国际化；反射和源代码生成器；用于图片处理、日志、映射和生成 PDF 文件的第三方库；性能基准测试和多任务等。
- 服务：gRPC、OData、GraphQL、Azure Functions、SignalR、Minimal Web API。
- 用户界面：ASP.NET Core、Blazor WebAssembly、.NET MAUI。

1.2.10　理解中间语言

dotnet CLI 工具使用的 C#编译器(名为 Roslyn)会将 C#源代码转换成中间语言(Intermediate Language，IL)代码，并将 IL 存储在程序集(DLL 或 EXE 格式的文件)中。IL 代码语句就像汇编语言指令，由.NET 的虚拟机 CoreCLR 执行。

在运行时，CoreCLR 从程序集中加载 IL 代码，再由 JIT 编译器将 IL 代码编译成本机 CPU 指令，最后由机器上的 CPU 执行。

以上两步编译过程带来的好处是，微软公司能为 Linux、macOS 以及 Windows 创建 CLR。在编译过程中，相同的 IL 代码能够在各个地方运行，因为编译过程的第二步将为本地操作系统和 CPU 指令集生成代码。

不管源代码是用哪种语言(如 C#、Visual Basic 或 F#)编写的，所有.NET 应用程序都会为存储在程序集中的指令使用 IL 代码。使用微软和其他公司提供的反汇编工具(如.NET 反编译工具 ILSpy)可以打开程序集并显示 IL 代码。

1.2.11　比较.NET 技术

表 1.5 对.NET 技术进行了总结和比较。

表 1.5 比较.NET 技术

.NET 技术	说明	驻留的操作系统
现代.NET	现代特性集，完全支持 C# 8 到 C# 11，支持移植现有应用程序，可用于创建新的桌面、移动和 Web 应用程序及服务。可以支持旧的.NET 平台	Windows、macOS、Linux、Android、iOS、Tizen
.NET Framework	旧的特性集，提供有限的 C# 8.0 支持，不支持 C# 9 到 C# 11，用于维护现有的应用程序	只用于 Windows
Xamarin	用于移动和桌面应用程序	Android、iOS 和 macOS

1.3 使用 Visual Studio 2022 构建控制台应用程序

本节的目的是展示如何使用 Visual Studio 2022 for Windows 构建控制台应用程序。

如果没有 Windows 计算机，或你想使用 Visual Studio Code，那么可以跳过这一节，因为代码是相同的，只是工具体验不同。

1.3.1 使用 Visual Studio 2022 管理多个项目

Visual Studio 2022 有一个名为“解决方案”的概念，允许同时打开和管理多个项目。我们将使用一个解决方案来管理本章创建的两个项目。

1.3.2 使用 Visual Studio 2022 编写代码

开始编写代码吧!

(1) 启动 Visual Studio 2022。

(2) 在 Start 窗口中，单击 Create a new project。

(3) 在 Create a new project 对话框中，在 Search for templates 框中输入 console，并选择 Console App，确保选择 C#项目模板而不是其他语言，如 Visual Basic 或 C++，并且项目模板是跨平台的，而不是只针对 Windows 的.NET Framework，如图 1.3 所示。

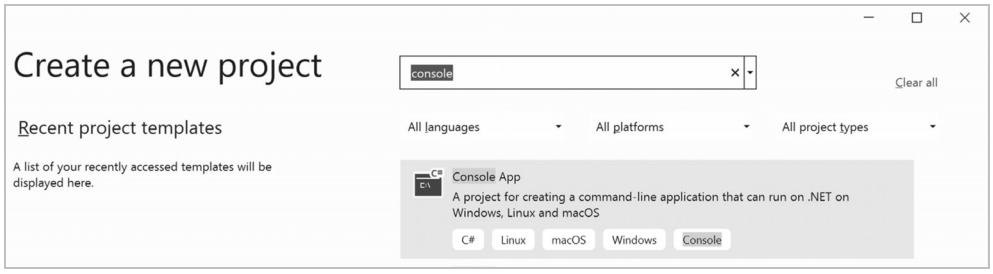

图 1.3 为现代跨平台.NET 选择 Console App 项目模板

(4) 单击 Next 按钮。

(5) 在 Configure your new project 对话框中，为项目名称输入 HelloCS，为位置输入 C:\cs11dotnet7，为解决方案名称输入 Chapter01。

(6) 单击 Next 按钮。

(7) 在 Additional information 对话框的 Target Framework 下拉列表中，注意.NET 的短期支持和长期支持版本的选择，然后选择.NET 6.0 或.NET 7.0。对于本章的项目，我们不需要任何.NET 7 特定的功能。

更多信息：

如果缺少.NET SDK 版本，可以从以下链接安装它们：https://dotnet.microsoft.com/en-us/download/dotnet。

(8) 保持 Do not use top-level statement 复选框的未选中状态，然后单击 Create。

(9) 如果未显示代码，就在 Solution Explorer 中，双击以打开名为 Program.cs 的文件，请注意 Solution Explorer 显示了 HelloCS 项目，如图 1.4 所示。

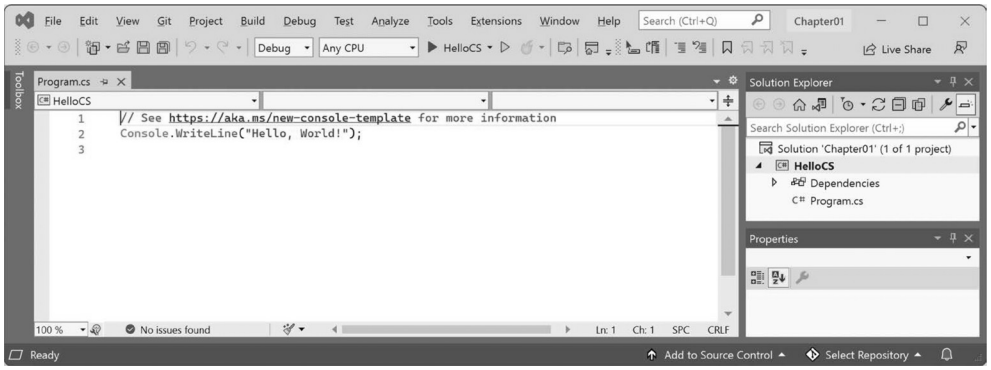

图 1.4　在 Visual Studio 2022 中编辑 Program.cs 文件

(10) 注意，Program.cs 的代码只包含一个注释和一条语句，因为它使用了 C# 9 中引入的顶级程序特性，如下面的代码段所示：

```
// See https://aka.ms/new-console-template for more information
Console.WriteLine("Hello, World!");
```

更多信息：

如代码中的注释所述，从以下链接可以阅读关于这个模板的更多信息：https://aka.ms/new-console-template。

(11) 在 Program.cs 中，修改第 2 行，以便写入控制台的文本是 Hello, C#!

更多信息：

读者必须查看或者键入的所有代码示例和命令都以纯文本形式显示，如步骤(10)中所示。你不需要从图 1.4 那样的屏幕截图中阅读代码或命令，因为在纸上显示时，屏幕截图中的代码不够清晰。

1.3.3　使用 Visual Studio 编译和运行代码

下一个任务是编译并运行代码，步骤如下。

(1) 在 Visual Studio 中，导航到 Debug | Start Without Debugging。

最佳实践：

在 Visual Studio 2022 中启动一个项目时，可以选择是否附加调试器。如果不需要调试，则最好不要附加调试器，因为附加调试器需要更多资源，从而会拖慢所有东西。而且，附加调试器后，只能启动一个项目。如果你想运行一个以上的项目，每个项目都附加调试器，就必须启动 Visual Studio 的多个实例。在工具栏中，单击绿色的空心三角形按钮，可以直接启动项目，而不进行调试；单击绿色实心三角形按钮，可以在调试模式下启动项目。

(2) 控制台窗口的输出将显示应用程序的运行结果，如图 1.5 所示。

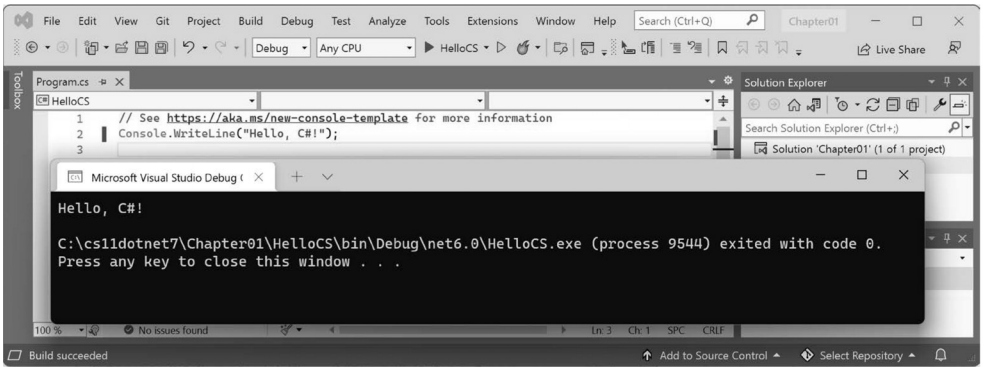

图 1.5　在 Windows 上运行控制台应用程序

(3) 按任意键关闭控制台窗口并返回 Visual Studio。

(4) 可以选择关闭 Properties 窗格，为 Solution Explorer 留出更多纵向空间。

(5) 双击 HelloCS 项目，打开它的项目文件。注意，取决于你在创建项目时选择的 SDK 版本，HelloCS.csproj 项目文件会显示这个控制台应用程序针对的是 net6.0 还是 net7.0，如图 1.6 所示。

(6) 在 Solution Explorer 工具栏中，切换 Show All Files 按钮，注意编译器生成的 bin 和 obj 文件夹是可见的，如图 1.6 所示。

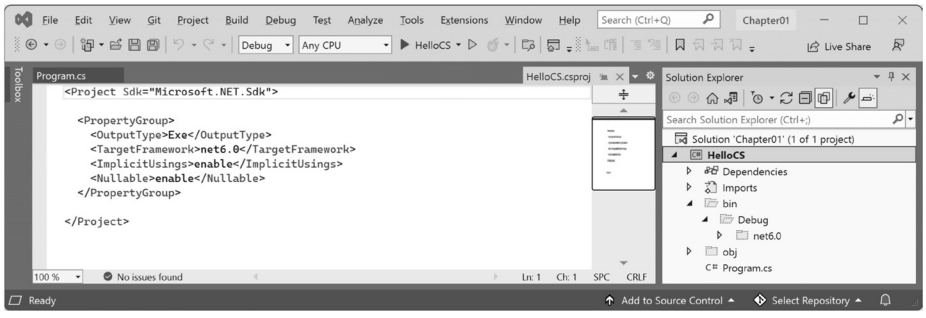

图 1.6　显示编译器生成的文件夹和文件

理解编译器生成的文件夹和文件

前面创建了两个编译器生成的文件夹，分别为 obj 和 bin。你不需要查看这些文件夹或了解它

们的文件(但如果你感到好奇，可以查看它们)。

注意，编译器需要创建临时文件夹和文件来完成工作。可以删除这些文件夹及其文件，下一次“构建”或运行项目的时候会自动重建它们。开发人员经常删除这些临时文件夹和文件来“清理”项目。Visual Studio 甚至在 Build 菜单上有一个名为 Clean Solution 的命令，它可以删除一些临时文件。Visual Studio Code 的等效命令是 dotnet clean。下面简单描述这两个文件夹中包含的内容：

- obj 文件夹为每个源代码文件包含一个已编译的目标文件。这些对象还没有被链接到最终的可执行文件。
- bin 文件夹包含应用程序或类库的二进制可执行文件，相关内容详见第 7 章。

1.3.4　理解顶级程序

如果你见过以前的.NET 项目，可能会期望看到更多的代码，哪怕只是为了输出一条简单的消息。这里的代码很少，是因为在针对.NET 6 及更新版本时，编译器会自动编写一些代码。

如果你使用.NET SDK 5.0 或更早版本创建这个项目，或者如果你选中了 Do not use top-level statements 复选框，那么 Program.cs 文件将包含更多语句，如下所示：

```
using System; // Not needed in .NET 6 or later.

namespace HelloCS
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Hello, World!");
    }
  }
}
```

在.NET SDK 或更新版本的编译期间，所有用于定义名称空间、Program 类及其 Main 方法的样板代码都会自动生成并封装我们编写的语句。这使用了一种称为顶级程序的功能，.NET 5 中已经引入该功能，但直到.NET 6，微软才更新了控制台应用程序的项目模板，在默认情况下使用顶级程序。

关于顶级程序，需要记住的要点包括：

- 项目中只能有一个这样的文件。
- 任何 using 语句仍然必须放在文件的顶部。
- 如果你声明任何类或其他类型，必须把它们放到文件底部。
- 尽管在显式定义时，必须使用 Main 这个方法名称，但当编译器创建该方法时，会将其命名为<Main>$。

文件顶部的 using System;语句用于导入 System 名称空间。这将启用 Console.WriteLine 语句。为什么我们不需要把它导入项目中呢？

1. 隐式导入名称空间

诀窍在于，仍然需要导入 System 名称空间，但现在使用 C# 10 和.NET 6 中引入的特性就可

以完成了。如下所示：

(1) 在 Solution Explorer 中，依次展开 obj 文件夹、Debug 文件夹和 net6.0(或 net7.0)文件夹，打开文件 HelloCS.GlobalUsings.g.cs。

(2) 注意，这个文件是由编译器为面向.NET 6 或更新版本的项目自动创建的，它使用了 C# 10 中引入的一个叫作全局名称空间导入的特性，该特性导入了一些常用的名称空间，如 System，以便在所有代码文件中使用，代码如下所示：

```
// <autogenerated />
global using global::System;
global using global::System.Collections.Generic;
global using global::System.IO;
global using global::System.Linq;
global using global::System.Net.Http;
global using global::System.Threading;
global using global::System.Threading.Tasks;
```

第 2 章将详细解释这个特性。现在请注意.NET 5 和.NET 6 之间的一个重大变化是，许多项目模板(如控制台应用程序的模板)使用新的语言特性来隐藏实际发生的事情。

(3) 在 Solution Explorer 中，单击 Show All Files 按钮隐藏 bin 和 obj 文件夹。

第 2 章将详细解释隐式导入特性。现在请注意.NET 5 和.NET 6 之间的一个重大变化：许多项目模板(如控制台应用程序的模板)使用新的 SDK 和语言特性来隐藏实际发生的事情。

2. 通过抛出异常显示隐藏的代码

现在来看看隐藏的代码是如何写入的：

(1) 在 Program.cs 中输出消息的语句之后，添加一条语句来抛出一个异常，如下面的代码所示：

```
throw new Exception();
```

(2) 在 Visual Studio 中，导航到 Debug | Start Without Debugging(不要在调试模式下启动项目，否则异常会被调试器捕获)。

(3) 控制台窗口的输出将显示运行程序的结果，可以看到编译器定义了一个隐藏的 Program 类，其中包含一个名为<Main>$的方法，它有一个名为 args 的参数，用于传入实参，如图 1.7 所示。

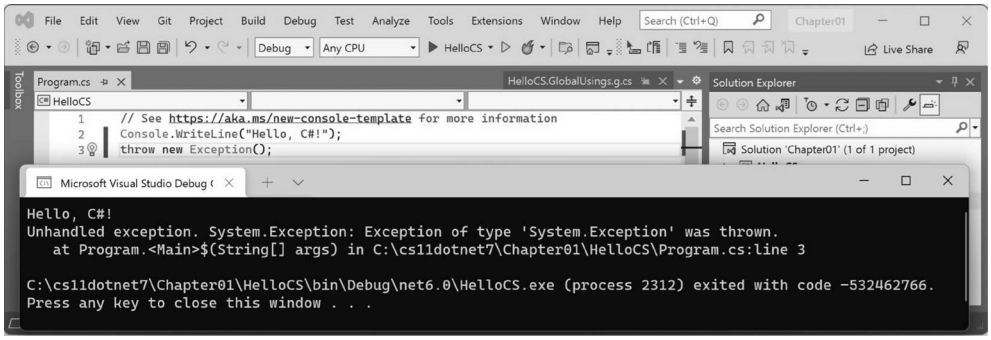

图 1.7 通过抛出异常来揭示隐藏的 Program.<Main>$方法

(4) 按任意键关闭控制台应用的窗口，返回到 Visual Studio。

1.3.5　使用 Visual Studio 2022 添加第二个项目

在解决方案中添加第二个项目，探索如何管理多个项目。

(1) 在 Visual Studio 中，导航到 File | Add | New Project。

(2) 在 Add a new project 对话框，在 Recent project templates 中选择 Console Application [C#]，然后单击 Next 按钮。

(3) 在 Configure your new project 对话框中，输入项目名称 AboutMyEnvironment，保留位置为 C:\cs11dotnet7\Chapter01，然后单击 Next 按钮。

(4) 在 Additional information 对话框中，选择.NET 6.0 或.NET 7.0，然后单击 Create 按钮。

(5) 在 AboutMyEnvironment 项目中，修改 Program.cs 文件中的语句，输出当前目录以及操作系统的版本，如下面突出显示的代码所示：

```
// See https://aka.ms/new-console-template for more information
Console.WriteLine(Environment.CurrentDirectory);
Console.WriteLine(Environment.OSVersion.VersionString);
```

(6) 在 Solution Explorer 中，右击 Chapter01 解决方案，选择 Set Startup Projects…，设置 Current selection，然后单击 OK。

(7) 在 Solution Explorer 中，单击 AboutMyEnvironment 项目(或该项目中的任意文件或文件夹)，注意 Visual Studio 会以粗体显示项目名称，指出 AboutMyEnvironment 现在是启动项目。

(8) 导航到 Debug | Start Without Debugging，运行 AboutMyEnvironment 项目，并注意结果，如图 1.8 所示。

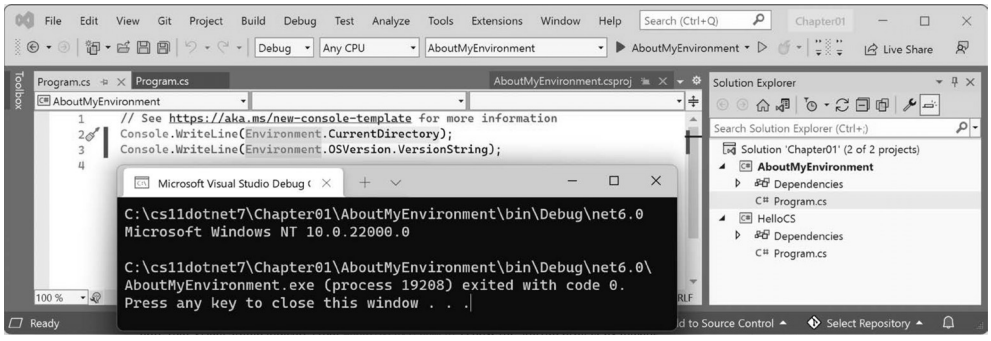

图 1.8　在 Windows 11 上，在包含两个项目的 Visual Studio 解决方案中运行一个顶级程序

更多信息：

Windows 11 只是一种品牌宣传。它的正式主版本号仍然是 10！但是，它的补丁版本是 22000 或更高版本。

(9) 按任意键关闭控制台应用的窗口，返回到 Visual Studio。

注意：

当使用 Visual Studio 2022 for Windows 运行控制台应用程序时，会从 <projectname>\bin\Debug\net7.0 或 net6.0 文件夹中执行该应用。在后续章节中使用文件系统时，记住这一点很重要。当使用 Visual Studio Code 时，或者更准确地说，使用 dotnet CLI 时，行为是不同的，后面将介绍这一点。

1.4 使用 Visual Studio Code 构建控制台应用程序

本节的目标是展示如何使用 Visual Studio Code 和 dotnet 命令行(CLI)构建控制台应用程序。

如果不想尝试 Visual Studio Code 或.NET Interactive Notebooks，那么请随意跳过此节和下一节，然后继续 1.1.3 节。

本节中的指令和屏幕截图都是针对 Windows 的，但是相同的操作也适用于 macOS 和 Linux 发行版的 Visual Studio Code。

主要区别在于本机命令行操作，比如在 Windows、macOS 和 Linux 上，删除文件时使用的命令和路径就可能不同。幸运的是，dotnet 命令行工具在所有平台上都是相同的。

1.4.1 使用 Visual Studio Code 管理多个项目

Visual Studio Code 有一个名为工作区的概念，允许同时打开和管理多个项目。我们将使用工作区来管理本章创建的两个项目。

1.4.2 使用 Visual Studio Code 编写代码

下面开始编写代码吧!

(1) 启动 Visual Studio Code。

(2) 确保没有任何打开的文件、文件夹或工作区。

(3) 导航到 File | Save Workspace As....

(4) 在对话框中，导航到 Windows 上的 C:驱动器，macOS 上的用户文件夹(我的文件夹名为 markjprice)，或者想要保存项目的任何目录或驱动器。

(5) 单击 New Folder 按钮，并将文件夹命名为 cs11dotnet7(如果你完成了 Visual Studio 2022 的部分，那么该文件夹已经存在)。

(6) 在 cs11dotnet7 文件夹中，创建一个名为 Chapter01-vscode 的新文件夹。

(7) 在 Chapter01-vscode 文件夹中，将工作区保存为 Chapter01.code-workspace。

(8) 导航到 File | Add Folder to Workspace…或单击 Add Folder 按钮。

(9) 在 Chapter01-vscode 文件夹中，创建一个名为 HelloCS 的新文件夹。

(10) 选择 HelloCS 文件夹并单击 Add 按钮。

(11) 看到对话框询问是否信任这个文件夹中的文件的作者时，单击 Yes 按钮，如图 1.9 所示。

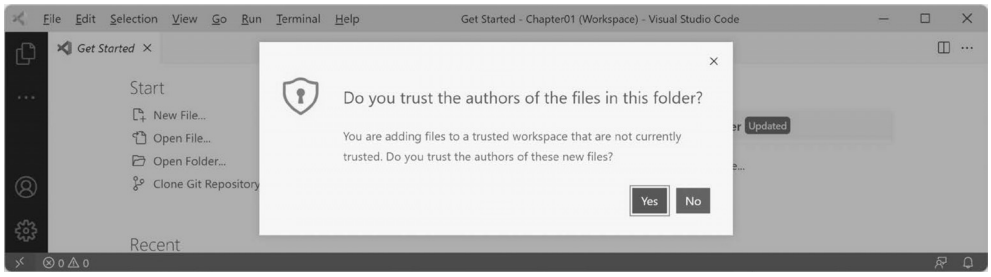

图 1.9　在 Visual Studio Code 中信任文件夹

(12) 在菜单栏中，注意指出了 Restricted Mode 的蓝色条，在这里单击 Manage 按钮。

(13) 在 Workspace Trust 选项卡中，单击 Trust 按钮，如图 1.10 所示。

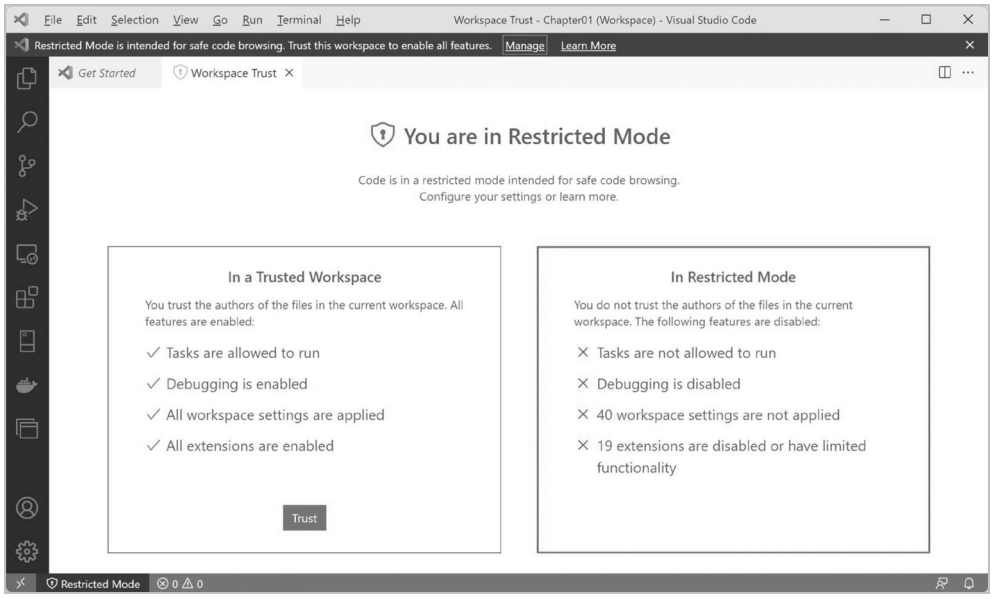

图 1.10　在 Visual Studio Code 中信任工作区

注意：

在将来使用 Visual Studio Code 时，如果某个功能看起来有问题，可能是因为你的工作区或文件夹处于受限模式。可以在 Workspace Trust 窗口中向下滚动，手动管理你当前信任哪些文件夹或工作区。

(14) 关闭 Workspace Trust 选项卡。

(15) 导航到 View | Terminal。

(16) 在 TERMINAL 中，确保位于 HelloCS 文件夹中，然后使用 dotnet 命令行工具创建一个新的控制台应用程序，如下所示：

```
dotnet new console
```

注意：

dotnet new console 命令默认情况下针对的是最新版本的.NET SDK。要针对一个不同的版本，可以使用-f 开关指定目标框架，如下面的命令指定了目标框架为.NET 6.0：

```
dotnet new console -f net6.0
```

(17) dotnet 命令行工具在当前文件夹中创建了一个新的 Console App 项目，并且 EXPLORER 窗口显示了创建的两个文件 HelloCS.csproj 和 Program.cs，以及 obj 文件夹，如图 1.11 所示。

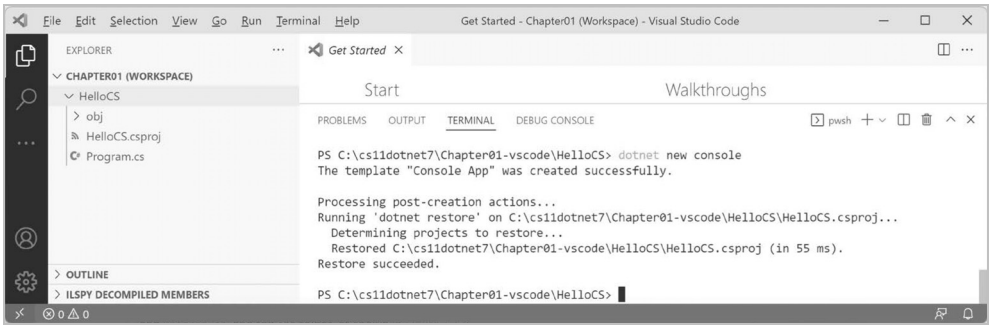

图 1.11　EXPLORER 窗口将显示已创建了两个文件和一个文件夹

更多信息：

默认情况下，项目的名称将匹配文件夹的名称。前面的 dotnet new console 命令在 HelloCS 文件夹中创建了一个名为 HelloCS.csproj 的项目文件。如果你想同时创建文件夹和项目，可以使用-o 开关。例如，如果你在 Chapter01-vscode 文件夹中，想要创建一个名为 HelloCS 的子文件夹和项目，可以输入下面的命令：

```
dotnet new console -o HelloCS
```

(18) 在 EXPLORER 中，单击名为 Program.cs 的文件，在编辑器窗口中打开它。起初，如果安装 C#扩展时没有这样做，或者它们需要更新，那么 Visual Studio Code 可能不得不下载并安装 C#依赖项，如 OmniSharp、.NET Core Debugger 和 Razor Language Server。Visual Studio Code 将在 Output 窗口中显示进度，最终显示消息 Finished，如下所示：

```
Installing C# dependencies...
Platform: win32, x86_64
Downloading package 'OmniSharp for Windows (.NET 4.6 / x64)' (36150
KB)..................... Done!
Validating download...
Integrity Check succeeded.
Installing package 'OmniSharp for Windows (.NET 4.6 / x64)'
Downloading package '.NET Core Debugger (Windows / x64)' (45048
KB)..................... Done!
Validating download...
Integrity Check succeeded.
Installing package '.NET Core Debugger (Windows / x64)'
Downloading package 'Razor Language Server (Windows / x64)' (52344
KB)..................... Done!
Installing package 'Razor Language Server (Windows / x64)'
Finished
```

上述输出来自 Windows 上的 Visual Studio Code。在 macOS 或 Linux 上运行时，输出略有不同，但将下载和安装对应操作系统的等效组件。

(19) 名为 obj 和 bin 的文件夹将被创建，若看到一个通知指出需要的资产(asset)不见了，则单击 Yes 按钮，如图 1.12 所示。

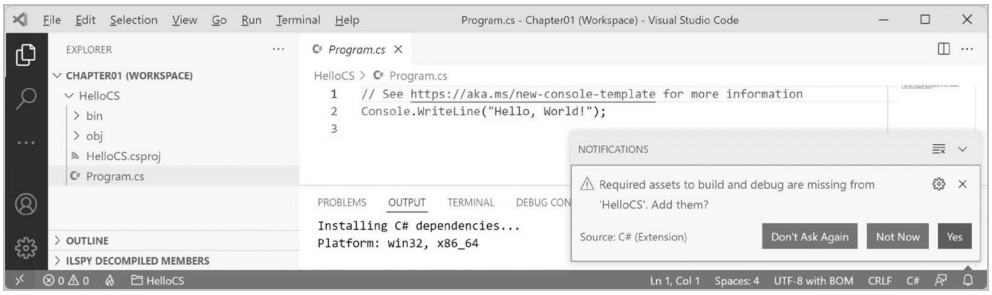

图 1.12　警告消息指出需要添加所需的构建和调试资产

如果通知在你与之交互之前就消失了，那么可以单击状态栏右下角的铃声图标，再次显示它。

(20) 几秒钟后，创建另一个名为.vscode 的文件夹，其中包含一些文件，Visual Studio Code 使用这些文件来提供调试期间的智能感知功能，详见第 4 章。

(21) 在 Program.cs 中，修改第 2 行，以便写入控制台的文本是 Hello, C#!

最佳实践：

导航到 File | Auto Save，从而省去每次重新构建应用程序之前都要保存的麻烦。

1.4.3　使用 dotnet CLI 编译和运行代码

下一个任务是编译和运行代码。

(1) 导航到 View | Terminal 并输入以下命令：

```
dotnet run
```

(2) TERMINAL 窗口中的输出将显示运行应用程序的结果，如图 1.13 所示。

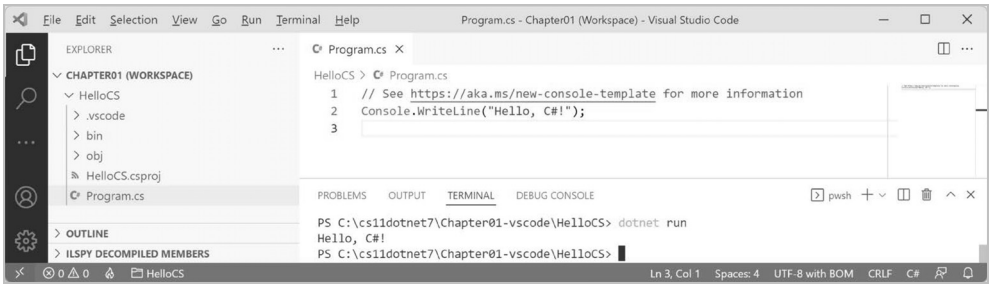

图 1.13　运行第一个控制台应用程序的输出

(3) 在 Program.cs 中输出消息的语句的后面，添加一条语句抛出一个异常，如下面的代码所示：

```
throw new Exception();
```

(4) 导航到 View | Terminal，输入下面的命令：

```
dotnet run
```

注意：

在 TERMINAL 中，可以按上下方向键遍历之前执行的命令，按左右方向键编辑命令，最后按 Enter 键运行命令。

(5) Terminal 窗口的输出将显示运行程序的结果，可以看到编译器定义了一个隐藏的 Program 类，其中包含一个名为<Main>$的方法，它有一个名为 args 的参数，用于传入实参，如下面的输出所示：

```
Hello, C#!
   at Program.<Main>$(String[] args) in C:\cs11dotnet7\Chapter01-vscode\
HelloCS\Program.cs:line 3
```

1.4.4 使用 Visual Studio Code 添加第二个项目

在工作区中添加第二个项目，以探索如何管理多个项目。

(1) 在 Visual Studio Code 中，导航到 File | Add Folder to Workspace...。

(2) 在 Chapter01-vscode 文件夹中，使用 New Folder 按钮创建一个名为 AboutMyEnvironment 的新文件夹，选择它，然后单击 Add 按钮。

(3) 当被询问是否信任文件夹时，单击 Yes 按钮。

(4) 导航到 Terminal | New Terminal，在显示的下拉列表中，选择 AboutMyEnvironment。或者，在 EXPLORER 中，右击 AboutMyEnvironment 文件夹，然后选择 Open in Integrated Terminal。

(5) 在 TERMINAL 中，确认位于 AboutMyEnvironment 文件夹，然后输入命令创建一个新的控制台应用程序，如下所示：

```
dotnet new console
```

最佳实践：

在 TERMINAL 中输入命令时要小心。在输入可能具有破坏性的命令之前，请确保处于正确的文件夹中！这就是在发出创建新控制台应用程序的命令之前为 AboutMyEnvironment 创建一个新终端的原因。

(6) 导航到 View | Command Palette。

(7) 输入 omni，然后在出现的下拉列表中选择 OmniSharp: Select Project。

(8) 在两个项目的下拉列表中，选择 AboutMyEnvironment 项目，当出现提示时，单击 Yes 按钮添加需要调试的资产。

最佳实践：

为了启用调试和其他有用的功能，如代码格式化和 Go To Definition，必须告诉 OmniSharp，你正在 Visual Studio Code 中开发哪个项目。可通过单击状态栏左侧火焰图标右边的项目/文件夹快速切换活动项目。

(9) 在 EXPLORER 的 AboutMyEnvironment 文件夹中，选择 Program.cs，然后更改现有语句

以输出当前目录和操作系统版本字符串，代码如下所示：

```
Console.WriteLine(Environment.CurrentDirectory);
Console.WriteLine(Environment.OSVersion.VersionString);
```

(10) 在 TERMINAL 中输入命令运行程序，如下所示：

```
dotnet run
```

(11) 注意 TERMINAL 窗口的输出，如图 1.14 所示。

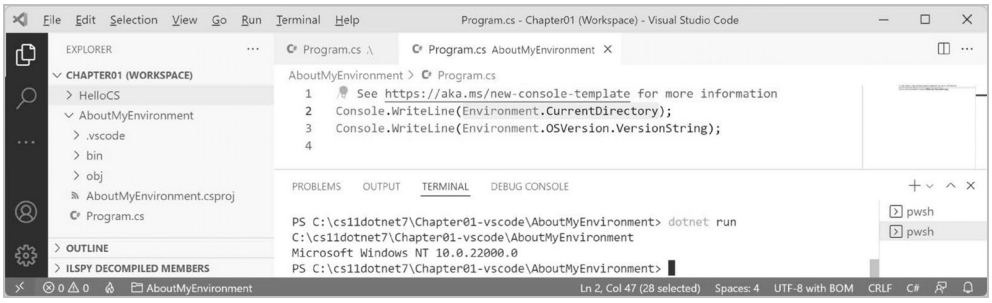

图 1.14　在 Visual Studio Code 工作区中运行带有两个项目的顶级程序

注意：

当使用 Visual Studio Code 运行控制台应用程序时，或者更准确地说，使用 dotnet 命令行来运行时，会从<projectname>文件夹中执行该应用程序。当使用 Visual Studio 2022 for Windows 时，会从<projectname>\bin\Debug\net7.0 或 net6.0 文件夹中执行该应用程序。在后面的章节中使用文件系统时，记住这一点很重要。

如果在 macOS Big Sur 上运行这个程序，环境操作系统会有所不同，如下所示：

```
Unix 11.2.3
```

1.5　使用.NET Interactive Notebook 探索代码

这是本章的一个附加小节，可以在线阅读，网址为 https://github.com/markjprice/cs11dotnet7/blob/main/docs/bonus/notebooks.md。

为本书中的代码使用.NET Interactive Notebook

后面的章节中不会给出使用 Notebook 的明确说明，但本书的 GitHub 存储库在适当的时候提供了解决方案 Notebook。我猜想很多读者想要为第 1~11 章介绍的语言和库功能运行预先创建的 Notebook，他们想要看到实际操作，而不需要编写完整的应用程序：

https://github.com/markjprice/cs11dotnet7/tree/main/notebooks

1.6 检查项目的文件夹和文件

本章创建了两个项目：HelloCS 和 AboutMyEnvironment。

Visual Studio Code 使用工作区文件管理多个项目。Visual Studio 2022 使用解决方案文件管理多个项目。你可能还创建了一个.NET Interactive Notebook。

创建完项目后，将得到一个文件夹结构和一些文件，后续章节中的项目将具有相似的文件夹结构和文件，只不过项目的数量不只是两个，如图 1.15 所示。

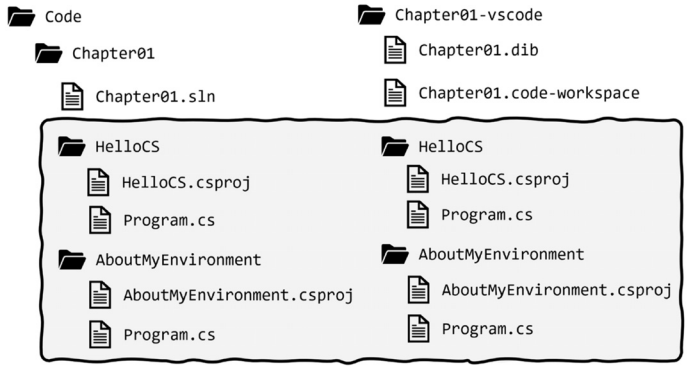

图 1.15 本章中两个项目的文件夹结构和文件

1.6.1 了解常见的文件夹和文件

尽管.code-workspace 和.sln 文件是不同的，但项目文件夹和文件(如 HelloCS 和 AboutMyEnvironment)对于 Visual Studio 2022 和 Visual Studio Code 是相同的。这意味着如果愿意，可以混合使用这两种代码编辑器。

- 在 Visual Studio 2022 中，打开解决方案，导航到 File | Add Existing Project...，添加由另一个工具创建的项目文件。
- 在 Visual Studio Code 中，打开工作区，导航到 File | Add Folder to Workspace...，添加由另一个工具创建的项目文件夹。

最佳实践：

虽然源代码是相同的，如.csproj 和.cs 文件，但 bin 和 obj 文件夹是由编译器自动生成的。文件版本可能不匹配，这就会出错。如果想在 Visual Studio 2022 和 Visual Studio Code 中打开相同的项目，则可以删除临时的 bin 和 obj 文件夹，然后在另一个代码编辑器中打开项目。这是在本章的解决方案中为 Visual Studio Code 创建另一个文件夹的原因。

1.6.2 理解 GitHub 中的解决方案代码

本书的 GitHub 存储库中的解决方案代码包括 Visual Studio Code、Visual Studio 2022 和.NET Interactive Notebook 文件的单独文件夹。

- Visual Studio 2022 解决方案：https://github.com/markjprice/cs11dotnet7/tree/main/vs4win

- Visual Studio Code 解决方案：https://github.com/markjprice/cs11dotnet7/tree/main/vscode
- .NET Interactive Notebook 解决方案：https://github.com/markjprice/cs11dotnet7/tree/main/notebooks

最佳实践：

可随时温习本章内容，回顾如何在所选的代码编辑器中创建和管理多个项目。GitHub 存储库提供了 4 种代码编辑器(Visual Studio 2022 for Windows、Visual Studio Code、Visual Studio 2022 for Mac 和 JetBrains Rider)的步骤说明，以及附加的截图：https://github.com/markjprice/cs11dotnet7/blob/main/docs/code-editors/。

1.7　充分利用本书的 GitHub 存储库

Git 是一种常用的源代码管理系统。GitHub 是一个公司、网站和桌面应用程序，它使 Git 管理更容易。微软在 2018 年收购了 GitHub，所以它将继续与微软的工具进行更紧密的整合。

我为本书创建了一个 GitHub 库，用它来做以下事情。

- 保存图书的解决方案代码，可在出版后进行维护。
- 提供额外的材料来扩展本书，比如勘误表的修正、微小的改进、有用的链接列表，以及纸质书中无法容纳的较长文章。
- 如果读者有关于本书的问题，可通过这个存储库与我联系。

1.7.1　对本书提出问题

如果困惑于本书的任何说明，或者发现正文或者解决方案中存在代码错误，请在 GitHub 存储库中提出问题。

(1) 使用喜欢的浏览器导航到以下链接：https://github.com/markjprice/cs11dotnet7/issues。

(2) 单击 New Issue。

(3) 输入尽可能多的细节，以帮助诊断问题。例如：

- 对于图中的错误，请提供页码和小节标题。
- 操作系统，如 Windows 11(64 位)或 macOS Big Sur 11.2.3 版本。
- 硬件，如英特尔、Apple Silicon 或 ARM CPU。
- 代码编辑器，如 Visual Studio 2022、Visual Studio Code 等，包括版本号在内。
- 相关的、必要的、尽可能详明的代码和配置。
- 描述预期行为和所经历的行为。
- 截图(可以把图片文件拖放到问题输入框中)。

我不能总是立即对问题给出答复。但我希望所有读者都能通过本书获得成功，所以我很乐意在力所能及的范围内帮助读者。

1.7.2　反馈

如果你想为我提供关于本书的更加一般性的反馈，可以向我发邮件。另外，GitHub 存储库中的 README.md 页面上有一些调查的链接。你可以匿名提供反馈。如果想得到回复，可以提供一个电子邮件地址；我只会用这个邮箱地址进行回复。

我喜欢听读者说他们喜欢本书的哪些内容，关于改进的建议，以及他们是如何使用 C#和.NET 的。不要害羞，请联系我!

提前感谢你的深思熟虑和建设性反馈意见。

1.7.3 从 GitHub 存储库下载解决方案代码

我使用 GitHub 存储各个章节中实用的、详细的编码示例和章末练习的解决方案。你将在以下链接中找到该存储库：https://github.com/markjprice/cs11dotnet7。

建议将前面的链接添加到书签中。

如果想在不使用 Git 的情况下下载所有解决方案文件，请单击 Code 按钮，然后选择 Download ZIP，如图 1.16 所示。

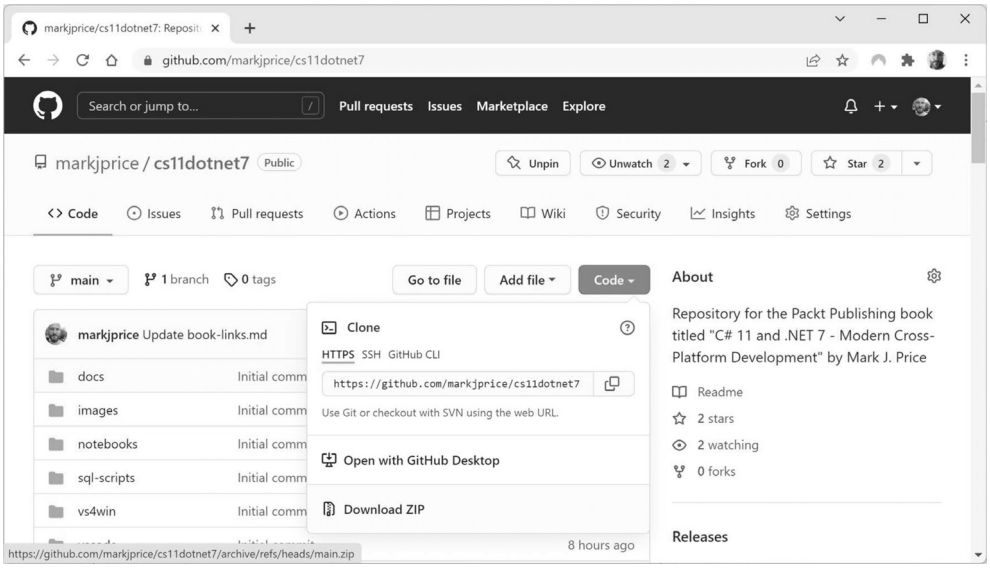

图 1.16 将存储库下载为 ZIP 文件

最佳实践：

最好将代码解决方案克隆或下载到一个短的文件夹路径中，如 C:\cs11dotnet7\或 C:\book\，以避免构建过程生成的文件超出最大路径长度。还应该避免特殊字符(如#)，例如，不要使用 C:\C# projects\。对于简单的控制台应用项目，这个文件夹名称可能不会造成问题，但一旦你开始添加自动生成代码的功能，就可能遇到奇怪的问题。应该让文件夹的名称保持简短。

1.7.4 在 Visual Studio Code 和命令行中使用 Git

Visual Studio Code 支持 Git，但需要使用操作系统的 Git 安装，所以必须先安装 Git 2.0 或更新版本。

可通过以下链接安装 Git：https://git-scm.com/download。

如果喜欢使用图形用户界面，可从以下链接下载 GitHub Desktop：https://desktop.github.com。

更多信息：

并不需要使用 Git，或者了解关于 Git 的任何信息，就可以获得本书的解决方案代码。通过访问下面的链接，可直接下载一个 ZIP 压缩文件，然后将其解压到本地文件系统中：https://github.com/markjprice/cs11dotnet7/archive/refs/heads/main.zip。

克隆本书解决方案代码存储库

下面克隆本书解决方案代码存储库。在接下来的步骤中，将使用 Visual Studio Code 终端，但也可在任何命令提示符或终端窗口中输入命令。

(1) 在用户文件夹或 Documents 文件夹中创建名为 Repos-vscode 的文件夹，也可在希望存储 Git 存储库的任何地方创建该文件夹。

(2) 在 Visual Studio Code 中打开 Repos-vscode 文件夹。

(3) 导航到 View | Terminal，输入以下命令：

```
git clone https://github.com/markjprice/cs11dotnet7.git
```

注意：

备份各个章节的所有解决方案需要一分钟左右的时间，所以请耐心等待。

1.8　寻求帮助

本节主要讨论如何在网络上查找关于编程的高质量信息。

1.8.1　阅读微软文档

关于微软开发工具和平台帮助的权威资源是 Microsoft Docs，参见 https://docs.microsoft.com/。

1.8.2　获取关于 dotnet 工具的帮助

在命令行，可以向 dotnet 工具请求有关 dotnet 命令的帮助。

(1) 要在浏览器窗口中打开 dotnet new 命令的官方文档，请在命令行或 Visual Studio Code 终端输入以下命令：

```
dotnet help new
```

(2) 要在命令行中获得帮助输出，可使用-h 或--help 标志，命令如下所示。

```
dotnet new console -h
```

(3) 部分输出如下：

```
Console App (C#)
Author: Microsoft
Description: A project for creating a command-line application that can
run on .NET Core on Windows, Linux and macOS
Options:
  -f|--framework. The target framework for the project.
```

```
                      net7.0         - Target net7.0
                      net6.0         - Target net6.0
                      net5.0         - Target net5.0
                      netcoreapp3.1. - Target netcoreapp3.1
                   Default: net7.0
  --langVersion   Sets langVersion in the created project file
                   text - Optional

  --no-restore If specified, skips the automatic restore of the
project on create.
                  bool - Optional
                  Default: false

To see help for other template languages (F#, VB), use --language option:
    dotnet new console -h --language F#
```

1.8.3 获取类型及其成员的定义

代码编辑器最有用的特性之一是Go To Definition。它在Visual Studio Code和Visual Studio 2022中可用。它将通过读取已编译程序集中的元数据，或者从源链接加载元数据，以显示类型或成员的公共定义。

有些工具(如.NET 反编译工具 ILSpy)甚至可将元数据和 IL 代码反向工程化为 C#或另一种语言。

下面看看如何使用 Go To Definition 特性：

(1) 在 Visual Studio 2022 或 Visual Studio Code 中打开 Chapter01 解决方案/工作区。

(2) 打开 HelloCS 项目，在 Program.cs 的底部输入以下语句，声明一个名为 z 的整型变量：

```
int z;
```

(3) 单击 int 内部，右击并从弹出的菜单中选择 Go To Definition。

(4) 在新出现的代码窗口中，可以看到 int 数据类型是如何定义的，如图 1.17 所示。

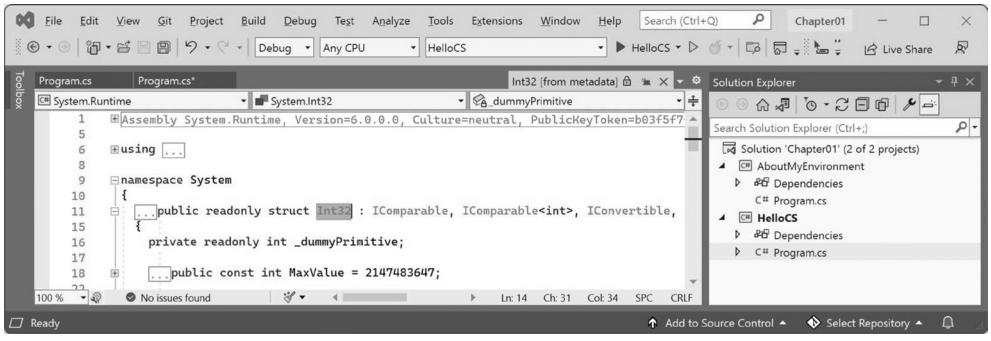

图 1.17 int 数据类型的元数据

可以看到，int 数据类型具有以下特点：

- 是用 struct 关键字定义的
- 在 System.Runtime 程序集中
- 在 System 名称空间中

- 被命名为 Int32
- 是 System.Int32 的别名
- 实现了 IComparable 等接口
- 最大值和最小值为常数
- 拥有 Parse 等方法

最佳实践：

当尝试在 Visual Studio Code 中使用 Go To Definition 特性时，有时会看到错误消息，指出没有找到定义。这是因为 C#扩展不知道当前项目。要修正这个错误，可导航到 View | Command Palette，输入 Omni 并选择 Omni Sharp: Select Project，然后选择要使用的正确项目。

现在，Go To Definition 特性似乎不是很有用，因为你还不知道这些术语的含义。

等到阅读完本书的第一部分(包括第 2~6 章)，你就会对这个特性有足够的了解，使用它时也会变得非常顺手。

(5) 在代码编辑器窗口中，向下滚动，找到带单个 string 参数的 Parse 方法，如下面的代码所示：

```
public static int Parse(string s)
```

(6) 展开代码，查看描述这个方法的注释，如图 1.18 所示。

更多信息：

图 1.23 显示了从包含注释的元数据生成的代码。Visual Studio 2022 也可能显示来自不包含注释的源链接的代码。

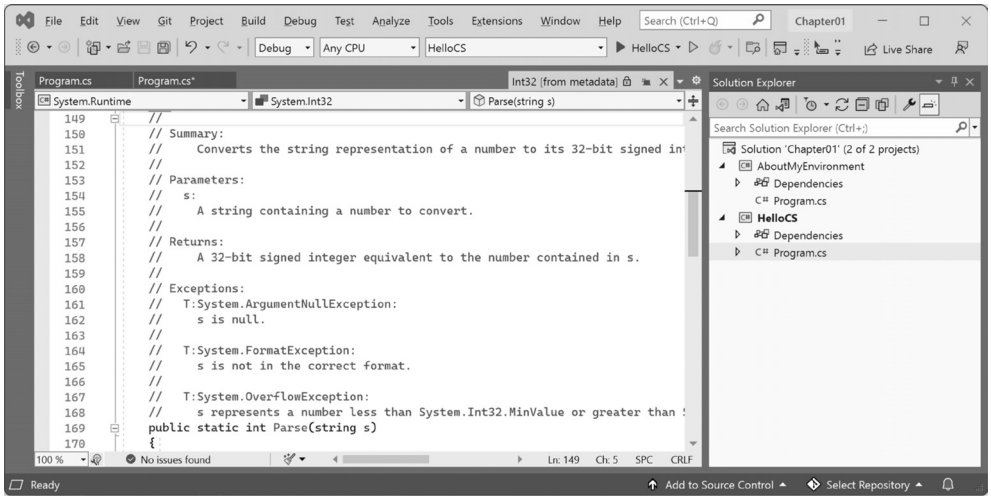

图 1.18　带单个 string 参数的 Parse 方法的注释

在注释中，微软记录了以下内容：

- 描述方法的摘要。

- 参数，比如可以传递给方法的 string 值。
- 方法的返回值，包括它的数据类型。
- 如果调用这个方法，可能会发生三个异常，包括 ArgumentNullException、FormatException 和 OverflowException。现在，我们知道了可以在 try 语句中封装对这个方法的调用，并且知道了要捕获哪些异常。

你可能已经迫不及待地想要了解这一切意味着什么!

再忍耐一会儿。本章差不多结束了，第 2 章将深入介绍 C#语言的细节。下面我们再看看还可从哪里寻求帮助。

1.8.4 在 Stack Overflow 上寻找答案

Stack Overflow 是最受欢迎的第三方网站，可以在上面找到编程难题的答案。Stack Overflow 非常受欢迎，像 DuckDuckGo 这样的搜索引擎提供了一种特殊的方式来编写搜索该网站的查询。具体执行步骤如下。

(1) 启动喜欢的 Web 浏览器。

(2) 进入 DuckDuckGo.com，输入以下查询，并注意搜索结果，如图 1.19 所示。

```
!so securestring
```

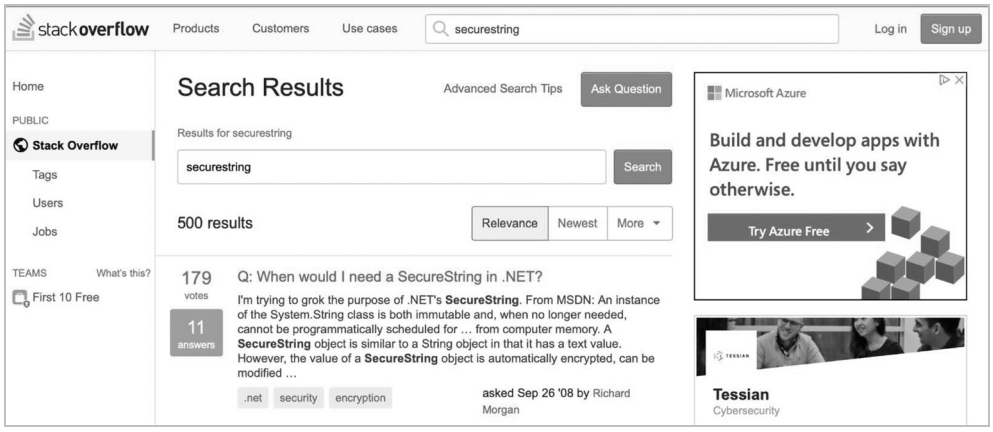

图 1.19 在 Stack Overflow 网站上搜索 securestring

1.8.5 使用谷歌搜索答案

可使用谷歌提供的高级搜索选项，以增大找到答案的可能性。具体执行步骤如下。

(1) 导航到谷歌。

(2) 使用简单的谷歌查询搜索关于 garbage collection(垃圾回收)的信息。请注意，你可能会先看到一堆与本地区垃圾回收服务相关的广告，然后才能看到维基百科针对垃圾回收在计算机科学领域的定义。

(3) 可通过将搜索结果限制在有用的站点(如 Stack Overflow)、删除我们可能不关心的语言(如 C++、Rust 和 Python)或显式地添加 C#和.NET 来改进搜索功能，如下所示：

```
garbage collection site:stackoverflow.com +C# -Java
```

1.8.6　订阅官方的.NET 博客

要跟上.NET 的最新动态，值得订阅的优秀博客就是.NET 工程团队编写的官方.NET 博客，网址为 https://devblogs.microsoft.com/dotnet/。

1.8.7　观看 Scott Hanselman 的视频

来自微软的 Scott Hanselman 在 YouTube 上有一个很好的频道，网址为 http://computerstuffftheydidntteachyou.com/。

我建议每个从事计算机工作的人关注这个频道。

1.8.8　本书的配套图书

我撰写了另外一本图书，帮助你在学习旅途中继续前行，所以可以把它作为本书的配套图书。

本书针对 Web 开发，介绍了 C#、.NET 和 ASP.NET Core 的基础知识。第二本书介绍了更加具体的主题，如在关系数据库和云数据库中管理数据，以及使用 Minimal API、OData、GraphQL、gRPC、SignalR 和 Azure Functions 构建服务。你将学习如何使用 ASP.NET Core、Blazor 和.NET MAUI 为网站、桌面应用和移动应用构建图形用户界面，如图 1.20 所示。

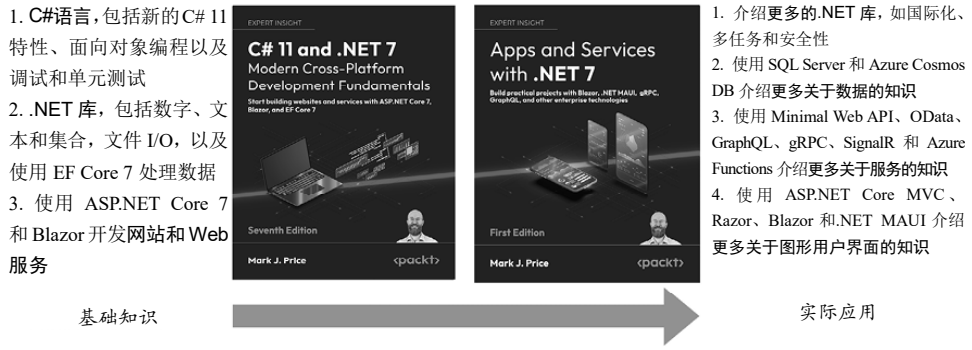

图 1.20　学习 C#和.NET 的配套图书

第一本书最好按顺序逐章阅读，因为它介绍基础知识，打造基本技能。

第二本书可以像阅读食谱那样阅读，所以如果你对构建 gRPC 服务特别感兴趣，可以先阅读该章。但第 2 章(介绍了 SQL Server)是一个例外，因为该章的一个编码任务介绍了如何在其他多个章节使用的类库中创建一个 EF Core 模型。

要查看我在 Packt 出版的所有图书的列表，可以访问下面的链接：

https://subscription.packtpub.com/search?query=mark+j.+price

Amazon 上也有类似的列表，当然你也可以在其他图书网站上搜索我的图书：

https://www.amazon.com/Mark-J-Price/e/B071DW3QGN/

1.9　实践和探索

现在尝试回答一些问题，从而测试自己对知识的理解程度，获得一些实际操作经验，并对本

章涉及的主题进行更深入的研究。

1.9.1 练习 1.1：测试你掌握的知识

试着回答以下问题，记住，虽然大多数答案可在本章中找到，但你需要进行一些在线研究或编写一些代码来回答其他问题：

(1) Visual Studio 2022 比 Visual Studio Code 更好吗?

(2) .NET 5 和更新版本比.NET Framework 更好吗?

(3) .NET Standard 是什么？为什么它很重要？

(4) 为什么程序员可使用不同的语言(如 C#和 F#)编写运行在.NET 上的应用程序？

(5) 什么是顶级程序?如何访问命令行参数?

(6) .NET 控制台应用程序的入口点方法是什么？如果你没有使用顶级程序特性，应该如何显式地声明它？

(7) 在编译和执行 C#源代码时，应在命令行输入什么?

(8) 使用.NET Interactive Notebook 编写 C#代码有什么好处?

(9) 可在哪里寻找关于 C#关键字的帮助?

(10) 可在哪里寻找常见编程问题的解决方案？

提示：

可从 GitHub 存储库的 README 文件中的链接下载附录 A(英文版)：https://github.com/markjprice/cs11dotnet7。

1.9.2 练习 1.2：使用浏览器在任何地方练习 C#

不需要下载并安装 Visual Studio Code，甚至不需要 Visual Studio 2022 for Windows 或 Visual Studio 2022 for Mac 就可以编写 C#代码。可以在以下任何链接开始在线编码：

- Visual Studio Code for Web：https://vscode.dev/
- SharpLab：https://sharplab.io/
- C# Online Compiler | .NET Fiddle：https://dotnetfiddle.net/
- W3Schools C# Online Compiler：https://www.w3schools.com/cs/cs_compiler.php

1.9.3 练习 1.3：探索主题

撰写一本书是一段精心策划的历程。笔者试图在印刷书中找到适当的主题平衡。我所写的其他内容可以在本书的 GitHub 存储库中找到。

相信本书涵盖了 C#和.NET 开发人员应该拥有或知道的所有基本知识和技能。一些较长的例子可在微软文档或第三方作者的文章中找到，本书提供了相关的链接。

请通过以下网页了解本章所涵盖主题的更多详情：

https://github.com/markjprice/cs11dotnet7/blob/main/book-links.md#chapter-1---helloc-welcome-net

1.9.4　练习 1.4：探索现代.NET 的主题

微软使用 Blazor 创建了一个网站，显示了现代.NET 的主要主题：https://themesof.net/。

1.10　本章小结

本章主要内容：

- 设置了开发环境。
- 讨论了现代.NET、.NET Core、.NET Framework、Xamarin 和.NET Standard 之间的异同。
- 使用带.NET SDK 的 Visual Studio 2022 for Windows 和 Visual Studio Code，在解决方案或工作区中创建了一些简单的控制台应用程序。
- 使用.NET Interactive Notebook 执行代码段。
- 学习如何从 GitHub 存储库下载本书的解决方案代码。
- 最重要的是，学会了如何寻求帮助。

第 2 章将学习 C#。

第2章 C#编程基础

本章主要介绍 C#编程语言的基础知识。你将学习如何使用 C#语法编写语句，还将了解一些几乎每天都会用到的常用词汇。除此之外，到本章结束时，你将对在计算机内存中临时存储和处理信息充满信心。

本章涵盖以下主题：

- 介绍 C#
- 理解 C#语法和词汇
- 使用变量
- 进一步探索控制台应用程序
- 理解 async 和 await

2.1 介绍 C#

本书的这一部分是关于 C#语言的——每天用来编写应用程序源代码的语法和词汇。

编程语言与人类语言有很多相似之处，但有一点除外：在编程语言中，可以自己创建单词！

在 Seuss 博士于 1950 年撰写的 *If I Ran the Zoo*(《若我管理动物园》)一书中，他写道:

然后，为了让他们看看，我要前往 Kar-Troo，带回一个 It-Kutch、一个 Preep、一个 Proo、一个 Nerkle、一个 Nerd，还有一个 Seersucker!

2.1.1 理解语言版本和特性

本书的这一部分主要是为初学者编写的，因此涵盖了所有开发人员都需要知道的基本主题，从声明变量到存储数据，再到如何自定义数据类型。

本节涵盖 C#语言从版本 1 到最新版本 11 的所有特性。

如果你对较旧版本的 C#有些熟悉，并且有兴趣了解最新版本的新特性，下面列出了该语言的版本及其重要的新特性，以及可以了解它们的章节编号和主题标题，以方便你阅读。

1. 项目 COOL

在 C#首次发布前，其代号为 COOL(类似 C 的面向对象语言)。

2. C# 1

C# 1 于 2002 年 2 月发布，其中包含静态类型的面向对象现代编程语言的所有重要特性，本书第 2~6 章将介绍这些特性。

3. C# 1.2

C# 1.2 与 Visual Studio .NET 2003 一起发布，其中包含一些小的改进，如 foreach 语句末尾的自动处理功能。

4. C# 2

C# 2 于 2005 年发布，其重点是使用泛型实现强数据类型，以提高代码的性能、减少类型错误，其中包含的主题如表 2.1 所示。

表 2.1　C# 2 中包含的主题

特性	涉及的章节	主题
可空的值类型	第 6 章	使值类型为空
泛型	第 6 章	通过泛型使类型的可重用性更好

5. C# 3

C# 3 于 2007 年发布，其重点是使用 LINQ(Language Integrated Queries)、匿名类型和 lambda 表达式等相关特性支持声明式编程，其中包含的主题如表 2.2 所示。

表 2.2　C# 3 中包含的主题

特性	涉及的章节	主题
隐式类型的局部变量	第 2 章	推断局部变量的类型
LINQ	第 11 章	所有的主题详见第 11 章

6. C# 4

C# 4 于 2010 年发布，其重点是利用 F#和 Python 等动态语言改进互操作性，其中包含的主题如表 2.3 所示。

表 2.3　C# 4 中包含的主题

特性	涉及的章节	主题
动态类型	第 2 章	存储动态类型
命名/可选参数	第 5 章	可选参数和命名参数

7. C# 5

C# 5 于 2012 年发布，其重点是简化异步操作支持，从而在编写类似于同步语句的语句时自动实现复杂的状态机，其中包含的主题如表 2.4 所示。

表 2.4　C# 5 中包含的主题

特性	涉及的章节	主题
简化的异步任务	第 2 章	理解 async 和 await

8. C# 6

C# 6 于 2015 年发布，专注于对 C#语言的细微改进，其中包含的主题如表 2.5 所示。

表 2.5　C# 6 中包含的主题

特性	涉及的章节	主题
静态导入	第 2 章	简化了控制台的使用
内插字符串	第 2 章	向用户显示输出
表达式体成员	第 5 章	定义只读属性

9. C# 7.0

C# 7.0 于 2017 年 3 月发布，其重点是添加了功能语言特性，如元组和模式匹配，以及对语言的细微改进，其中包含的主题如表 2.6 所示。

表 2.6　C# 7.0 中包含的主题

特性	涉及的章节	主题
二进制字面值和数字分隔符	第 2 章	存储整数
模式匹配	第 3 章	利用 if 语句进行模式匹配
out 变量	第 5 章	控制参数的传递方式
元组	第 5 章	将多个值与元组组合在一起
局部函数	第 6 章	定义局部函数

10. C# 7.1

C# 7.1 于 2017 年 8 月发布，其重点是对 C#语言做了细微改进，其中包含的主题如表 2.7 所示。

表 2.7　C# 7.1 中包含的主题

特性	涉及的章节	主题
async Main	第 2 章	改进对控制台应用程序的响应
默认字面值表达式	第 5 章	使用默认字面值设置字段
推断元组元素的名称	第 5 章	推断元组名称

11. C# 7.2

C# 7.2 于 2017 年 11 月发布，其重点是对 C#语言做了细微改进，其中包含的主题如表 2.8 所示。

表 2.8　C# 7.2 中包含的主题

特性	涉及的章节	主题
数字字面值中的前导下画线	第 2 章	存储整数
非尾随命名参数	第 5 章	可选参数和命名参数
private protected 访问修饰符	第 5 章	理解访问修饰符
可以使用元组类型测试==和!=	第 5 章	比较元组

12. C# 7.3

C# 7.3 于 2018 年 5 月发布，主要关注以性能为导向的安全代码，并且改进了 ref 变量、指针和 stackalloc。这些都是高级功能，大多数开发人员都很少使用，因此本书不涉及它们。

13. C# 8

C# 8 于 2019 年 9 月发布，主要关注 C#中与空处理相关的重大变化，其中包含的主题如表 2.9 所示。

表 2.9　C# 8 中包含的主题

特性	涉及的章节	主题
switch 表达式	第 3 章	使用 switch 表达式简化 switch 语句
可空引用类型	第 6 章	使引用类型可空
默认的接口方法	第 6 章	了解默认的接口方法

14. C# 9

C# 9 于 2020 年 11 月发布，关注记录类型、模式匹配的细化以及极简代码(Minimal-Code)控制台应用程序，其中包含的主题如表 2.10 所示。

表 2.10　C# 9 中包含的主题

特性	涉及的章节	主题
极简代码控制台应用程序	第 1 章	顶级程序
Target-typed new	第 2 章	使用 Target-typed new 实例化对象
改进的模式匹配	第 5 章	与对象的模式匹配
记录	第 5 章	处理记录

15. C# 10

C# 10 于 2021 年 11 月发布，主要关注那些将常见场景中所需代码量最小化的特性，其中包含的主题如表 2.11 所示。

表 2.11　C# 10 中包含的主题

特性	涉及的章节	主题
导入全局名称空间	第 2 章	导入名称空间
常量字符串字面值	第 2 章	使用内插字符串进行格式化

(续表)

特性	涉及的章节	主题
文件范围的名称空间	第 5 章	简化名称空间声明
记录结构	第 6 章	使用记录结构类型
ArgumentNullException.ThrowIfNull	第 6 章	检查方法参数是否为空

16. C# 11

C# 11 发布于 2022 年 11 月，主要关注那些简化代码的特性，其中包含的主题如表 2.12 所示。

表 2.12　C# 11 中包含的主题

特性	涉及的章节	主题
原始字符串字面值	第 2 章	理解原始字符串字面值
内插字符串表达式中的换行符	第 2 章	使用内插字符串进行格式化
需要的属性	第 5 章	要求在实例化期间设置属性

2.1.2　了解 C#标准

多年来，微软已经向标准组织提交了一些 C#版本，如表 2.13 所示。

表 2.13　C#标准

C#版本	ECMA 标准	ISO/IEC 标准
1.0	ECMA-334:2003	ISO/IEC 23270:2003
2.0	ECMA-334:2006	ISO/IEC 23270:2006
5.0	ECMA-334:2017	ISO/IEC 23270:2018

更多信息

C# 6 的 ECMA 标准仍然是一个草案，添加 C# 7 特性的工作正在进行中。微软是在 2014 年将 C#开源。可通过如下链接阅读 C#的 ECMA 标准文档：https://www.ecma-international.org/publications-and-standards/standards/ecma-334/。

比 ECMA 标准更实用的是一些公共的 GitHub 库，它们可以使 C#和相关技术的工作尽可能开放，如表 2.14 所示。

表 2.14　公共的 GitHub 库

说明	链接
C#语言设计	https://github.com/dotnet/csharplang
编译器实现	https://github.com/dotnet/roslyn
描述语言的标准	https://github.com/dotnet/csharpstandard