Unity 和 C#游戏编程入门

(第 7 版)

[美] 哈里森 • 费隆(Harrison Ferrone) 著
王 冬 殷崇英 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：01-2023-1362

图书在版编目（CIP）数据

Unity 和 C#游戏编程入门 : 第 7 版 / (美) 哈里森·费隆 (Harrison Ferrone) 著 ; 王冬, 殷崇英译.
北京 : 清华大学出版社, 2024. 8. -- ISBN 978-7-302-66802-2

Ⅰ. TP311.5

中国国家版本馆 CIP 数据核字第 2024TH7741 号

责任编辑：王　军
装帧设计：孔祥峰
责任校对：马遥遥
责任印制：丛怀宇

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：148mm×210mm　　**印　　张：**12.125　　**字　　数：**436 千字
版　　次：2024 年 8 月第 1 版　　**印　　次：**2024 年 8 月第 1 次印刷
定　　价：69.80 元

产品编号：100940-01

推荐一

游戏被很多人称为“第九艺术”，近年来更与传统的文学、音乐、建筑、雕塑、绘画、舞蹈、电影和戏剧这“八大艺术”开始并驾齐驱。

游戏之所以变得越来越受欢迎，我觉得与其丰富的体验方式和内容形式息息相关。我们不仅可以在手机上畅游《原神》这样的二次元开放式大世界，也可以在配备了高性能显卡的主机设备(如 Xbox 和 PlayStation)上，使用附带力反馈功能的手柄体验与游戏世界中超写实类角色深入互动，更可以戴上 VR 头盔感受真正的沉浸式 3D 互动体验。

而要做出好的游戏，游戏引擎是核心开发工具之一。Unity 作为目前世界范围内市场占有率第一的游戏引擎，经过十几年的快速发展，我们已经可以使用它为将近 30 个计算平台开发互动式内容。无论你是开发 2D、3D，还是 VR/AR/MR 互动式内容，Unity 都可以提供完整的开发工具链。近些年来，这些内容已经超出游戏的领域，进入影视动画、汽车制造、建筑建造这些非游戏领域。

因为游戏本质上是实时渲染出来的互动式内容，所以游戏的一个基本功能是可以接受玩家的输入信息(来自鼠标、键盘、手柄等)，并对其进行处理，然后实时生成相关的内容。因此对于游戏开发人员来说，使用编程语言开发相关的游戏逻辑就是其中必不可少的一环。

本书使用通俗易懂的语言，深入浅出地为想要使用 Unity 开发互动式内容(不仅仅是游戏)的读者，提供了非常系统性的学习资料。配合书中的实例项目，一步一个脚印，按部就班地学习，相信大家很快可以掌握在 Unity 中使用 C#编程语言的基础知识，开始自己的游戏开发之旅！

杨栋
Unity 大中华区平台技术总监
《创造高清 3D 虚拟世界：Unity 引擎 HDRP 高清渲染管线实战》的作者

推荐二

随着新一代信息技术的日新月异，以数字化、网络化、智能化、虚拟化为特征的信息化浪潮已经蔚然兴起，人们对信息内容的生产、传播和消费也从传统的单一渠道、单一媒介、单一体验升级为对多元、多维、多态的全域融合媒体的新需求。数字游戏融合了丰富的感官体验与高度的交互性，具有极佳的内容叙事能力，已成为当前最受欢迎、最具影响力的数字内容表达与传播形式之一；虚拟现实、增强现实、混合现实等人机交互技术更是为用户提供了虚实融合、沉浸全息的极致体验，进一步拉近了用户与内容的距离；以“万物智联”“虚实互映”为目标，数字孪生技术通过构建智慧工业、智慧城市、智慧校园等方式，加快了政治、经济、生产、教育、文旅、传媒等各领域数字产业化和产业数字化的进程 —— 而所有这一切，组成了今天备受人们关注和热议的元宇宙领域的重要基石。

与由短视频、自媒体的兴起会带来影视编辑学习门槛的降低，进而引领大众学习并参与内容生产一样，数字内容的消费升级也亟需低门槛、高效率、开放式的工具，使得万千大众能够积极参与数字内容的创作与开发，同时也为创意工作者、独立开发者以及中大型工作室提供更便利、更强大、更丰富的创作和开发环境，而 Unity 便是当前数字内容创作与开发领域中最受欢迎、最热门、最广受好评的引擎之一。

人工智能技术的快速发展与广泛应用，要求当代人应具备基本的编程能力和算法思维，以培养面向未来智能社会所需的信息素养。C#语言作为一种面向对象、类型安全、表达自然的编程语言，同时也作为 Unity 的脚本语言，是零基础初学者学习编程的绝佳选择，也便于已具备 C、C++、Java 和 JavaScript 经验的成熟编程者快速上手 Unity 的脚本编写。

本书从学习者的角度出发，将以往同类教材中晦涩难懂的概念与日常生活中的事物结合起来，用通俗平实的语言循序渐进地将编程的相关知识点娓娓道来，并通过案例实践的方式，与学习者一起制作游戏来达到学习 Unity 的目的，是一本适合各专业背景的学习者和数字内容创作爱好者学习与了解 Unity 游戏开发与 C#编程的入门教材和工具书。

曹三省 教授
中国传媒大学媒体融合与传播国家重点实验室党政班子成员
协同创新中心副主任、互联网信息研究院副院长
中国电子学会虚拟现实分会副主任委员

推荐三

首先非常感谢两位老师为本书的翻译付出的辛劳，很荣幸我能够有机会提前看到这本书，内容真的非常棒。本书充分地结合了 Unity 来讲解 C#的知识体系。相比单纯的 C#教学来说，更实用也更有意思！书中非常全面地介绍了 Unity 游戏开发过程中大家几乎一定会用到的 C#知识，并使用通俗易懂的方式来讲解原理和实际运用。对于我个人来说，在我的 Unity 项目中常常会运用很多 C#内容，但往往我只是停留在会用的程度，对其定义，或者说对其背后的基本原理和概念缺乏透彻的理解。这本书能够帮助我更全面地了解这些内容。特别值得一提的是，本书中的每一个关键知识点，除了结合了非常棒的案例以外，也有很多需要特别“注意”的提醒！书中还包括很多 C#技术文档的链接，可以帮助想要更深入学习知识的朋友快速定位到需要查阅的内容。事实上，本书更像一本参考书或字典，弥补了很多平时会被忽略的小技巧和知识点。想要在 Unity 开发过程中学习 C#，看这本书就足够了！

当然，这本书除了 C#语言的介绍讲解以外，也涵盖了对几乎所有 Unity 常用主要功能的介绍，组件的使用方法等。如果全部认真看完并且自己动手完成操作的话，那么你与做出一个小游戏 Demo 就只差一套美术素材的距离了。

总之，这本书让我收获很多。只有打好基础才能发挥更多创意。本书也是我看过的唯一一本讲解 C#的书，推荐给所有刚刚上手 Unity 或已经可以做一些小 Demo 的朋友来阅读学习，相信你一定会跟我一样收获良多。

Michael Wang
bilibili 知名 UP 主：M_Studio
2022 年度 Unity 最具社区影响人物

推荐四

社区里面经常会被问到：Unity 学习过程中需要了解哪些知识？这是一个很好的问题，也是一个很难回答的问题。幸运的是，本书很好地回答了这个问题。本书从 C#开始，涵盖 Unity 常用模块的使用，能够很好地帮助有兴趣的开发者深入浅出、系统性地学习 Unity。祝你开卷有益。

Unity 大中华区资深技术经理
高川

推荐五

从虚拟社区到游戏、元宇宙概念，虚拟体验实现了人们超越现实的想象，而 3D 引擎是链接虚拟体验的重要工具。这是一本非常好的入门教材，当你有一定的程序基础后，基于一个通用成熟的商业引擎，可以让你快速地了解 3D 游戏世界的基础结构搭建，从校门走向行业，将兴趣变成现实，帮助你完成自我的学习和实践。国内关于游戏的书籍一直非常少，更多的内容仅存在于行业内的交流。感谢分享经验和辛勤翻译的译者，让更多人有机会加入创造全新世界的体验中来。

徐振华
苏州游戏蜗牛 九阴工作室负责人

推荐六

游戏、互动式电影、扩展现实等一直深受青年一代的喜爱，尤其在当今全面数字化的时代中，正逐渐成为文化创意、国际文化交流、文化遗产保护等众多领域最重要的信息载体和有效传播手段之一。Unity 使用 C#作为脚本编程语言，是青年人初次接触和尝试数字创意与开发的首选。本书条理清晰，语言表达通俗易懂，真正面向零起点学习者，且对美术、艺术等非计算机相关专业背景的读者也非常友好。通过对本书的学习，读者既能掌握 Unity 的基本使用和操作技巧，同时也能掌握和理解 C#编程基础以及通用的编程理论知识。

王科
大连外国语大学创新创业学院院长、
软件学院院长、大数据产业学院院长

贡献者

关于作者

Harrison Ferrone 出生并成长于美国伊利诺伊州的芝加哥市。他曾工作于微软、普华永道和一些小型的初创公司，但大多数日子里，他在为 LinkedIn Learning 和 Pluralsight 创建教学内容，或者为 Ray Wenderlich 网站进行技术编辑工作。

他拥有来自科罗拉多大学博尔德分校和芝加哥哥伦比亚学院的多个看起来很高级的文凭，尽管他是这些学校值得骄傲的校友，但这些文凭都被存放在某个地下室里。

在做了几年全职 iOS 和 Unity 开发者后，他投身于教学生涯，直至今日。一路走来，他买过很多书，养过几只猫，也在国外工作过，并一直对于为什么《神经漫游者》没有出现在更多课程的教学大纲上感到疑惑。

本书的完成离不开来自 Kelsey 的爱与支持，她是我的妻子，也是我一路走来的伙伴。

关于审稿人

Simon Jackson 是一名资深的软件工程师和架构师，他拥有多年的 Unity 游戏开发经验，也是几本 Unity 游戏开发书籍的作者。他热爱创建 Unity 项目，同时也乐于为学习者伸出援手，无论是通过博客、Vlog、用户小组还是大型演讲活动。

他目前的主要工作是 XRTK(混合现实工具包)项目，该项目旨在构建一个跨平台的混合现实框架，以便 VR 和 AR 开发者能够在 Unity 中更高效地构建解决方案，然后将它们构建和分发到尽可能多的平台上。

向本书的试读者 Kyle Quesada、Karen Stingel、Laksh M.及其他试读者表示感谢，感谢你们花时间审阅本书的内容。你们的评论对我们的工作非常有帮助和重要；在你们的帮助下，我们才能制作出尽可能高品质的书籍——一本贴近目标受众且产生共鸣的书。真诚地感谢你们的参与，并期待能在未来再次合作！

译者序

随着游戏、影视动画、扩展现实、数字孪生乃至元宇宙等相关技术与应用的发展，世人对优质的视听、人机交互、虚拟仿真等相关需求愈加旺盛，数字内容的种类与形式也越发广泛和丰富，越来越多来自不同领域与专业的创意人员参与创作与开发。这便需要一种简单、快捷、高效的工具与工作流来满足不同领域的内容创作者与开发者的需求，Unity 便是其中最流行与优质并存的选择之一。

在 Unity 与 C#教学过程中，我们发觉现有教材或教程大多延续了传统计算机语言的语言范式和内容编排模式，虽严谨、专业度高，但言语晦涩难懂，更与应用和现实生活脱节。尤其是对于艺术、动画等非工科背景的学习者来说，专业语言成为一条难以逾越的门槛，很多 Unity 爱好者和学习者往往都因此中道而止。而且，随着人工智能技术的发展，像 ChatGPT 这类的 AI 工具也悄然影响着编程领域的学习，开发者可以通过自然语言直接生成 C#代码，此时只需要具备 C#基础知识，就可以快速判断 AI 生成代码的正确性，或者与 ChatGPT 反复推敲来调整代码，使之更适用于自己的项目。

本书原著作者用极其通俗生活化的语言和比喻为读者诠释了 Unity 与 C#语言的基础知识与使用方法，并结合了项目实践、说明与提示、小测验等模块进一步帮助读者理解和灵活运用 C#与 Unity。作者将创建的游戏项目命名为“Hero Born”(英雄诞生)，寄托了他对读者踏上学习征程的期许，还配有“Hero's Trial”(英雄的试炼)环节，鼓励读者接受章节中的挑战。

我们在翻译过程中也追求尽力还原和贴合原作者口语化的叙述风格，希望能用平易近人的语言，为来自任意领域任何背景的读者清晰诠释相关专业概念。期待本书可以成为帮助零基础，尤其是无编程背景的艺术、影视动画等学习者了解 Unity 与 C#的有利教程与工具书。

前言

Unity 是世界上最受欢迎的游戏引擎之一，它适用于业余爱好者、专业 3A 工作室和电影制作公司。尽管 Unity 常以其作为 3D 工具的使用而闻名，但它还有诸多专有功能，为 2D 游戏、虚拟现实、影视后期制作、跨平台发布等内容生产提供支持。

开发者喜欢 Unity 的拖放界面和内置功能，但真正让 Unity 更上一层楼的，是因其能为行为和游戏机制编写自定义 C#脚本。学习编写 C#代码对于已熟悉其他语言、经验丰富的程序员来说可能不是一个巨大的障碍，但对于那些没有编程经验的人来说，可能会是令人生畏的。本书的意义在，将带你从零开始学习编程的和 C#语言的基本知识，并同时在 Unity 中构建一个有趣且可玩的游戏原型。

本书读者对象

本书主要面向没有编程基础或 C#语言经验的人群。然而，无论你是 C#语言或其他编程语言的编程老手，还是编程初学者，只要想尝试在 Unity 中动手实践进行游戏开发，那么本书就正是你所需要的。

本书主要内容

第 1 章“了解开发环境”，介绍了 Unity 安装过程、Unity 编辑器的主要功能、如何查阅 C#和 Unity 特定主题的文档。以及如何从 Unity 编辑器创建 C#脚本，并初步了解如何使用 Visual Studio，Visual Studio 是进行所有的代码编辑工作的地方。

第 2 章“编程的基本构建块”，从列举编程的原子级概念开始，将变量、方法和类与日常生活中的事物联系起来。然后，介绍了简单的调试技巧、如何进行正确的格式设置和注释，以及 Unity 是如何将 C#脚本转换为组件的。

第 3 章“深入了解变量、类型和方法”，在第 2 章的基础上进行更深入的讲

解，内容包括 C#的数据类型、命名规则、访问修饰符等其他编程基础内容。还介绍了如何更有效地编写方法、使用参数和返回类型。最后，对属于 MonoBehaviour 类的标准 Unity 方法进行了概述。

第 4 章“控制流和集合类型”，介绍了在代码中做决策的常用方法，包括 if-else 和 switch 语句。然后介绍了如何使用数组、列表和字典，并结合迭代语句来循环遍历集合类型。在本章的最后，介绍了条件循环语句和一个称为枚举的特殊 C#数据类型。

第 5 章“使用类、结构体以及面向对象编程”，在本章首次接触如何构建和实例化类和结构体。还介绍创建构造函数、添加变量和方法的基本步骤，以及如何使用子类和继承的基础知识。最后，全面诠释了面向对象编程及其在 C#中的应用。

第 6 章“亲手实践 Unity”，标志着我们从 C#语法进入游戏设计、关卡构建和 Unity 特色工具的世界。首先从了解游戏设计文档的基础知识开始，然后进行游戏关卡的几何布局，并添加光照和简单的粒子系统。

第 7 章“角色移动、摄像机以及碰撞”，讲解了使玩家对象移动和设置第三人称摄像机的不同方法，还介绍了如何通过 Unity 物理引擎获得更逼真的运动效果，如何使用碰撞体组件以及如何捕获场景中的交互。

第 8 章“游戏机制脚本编写”，介绍了游戏机制的概念以及如何有效地实现游戏机制。从添加简单的跳跃动作开始，然后创建射击机制，并基于前几章的代码添加处理道具收集的逻辑。

第 9 章“AI 基础与敌人行为”，简要概述了游戏中的人工智能以及应用于 Hero Born 示例中的相关概念。涵盖的主题包括 Unity 中的地图导航、使用关卡几何结构和导航网格、智能代理，以及敌人自动移动等。

第 10 章“重睹类型、方法和类”，更深入地讲解了有关数据类型、中级的方法特性以及可用于更复杂类的附加行为。本章带领读者进一步了解 C#语言的功能多样性和广度。

第 11 章“特殊集合类型和 LINQ”，深入探讨了栈、队列和哈希集，以及它们分别适合的不同开发场景。此外，本章还探讨了如何使用 LINQ(语言集成查询)对数据集合进行过滤、排序、变换等。

第 12 章“保存、加载与数据序列化”，着重介绍了如何处理游戏信息。话题涵盖了如何操作文件系统，以及如何创建、删除、更新文件等。此外还介绍了如何处理如 XML、JSON、二进制数据等不同的数据类型，并在最后实践部分讨论如何将 C#对象直接序列化为对应的数据格式。

第 13 章“探索泛型、委托以及更多”，详细介绍了 C#语言的中级特性以及如何在实战中应用它们。从泛型编程的概述开始，逐步介绍了委托、事件和异常处理等概念。

第 14 章“旅程继续”，回顾了整本书中所学的主要主题，并提供了进一步学习 C#和 Unity 的资源。这些资源包括在线阅读材料、认证信息和一些推荐的视频教程频道。

如何充分地利用好本书

为了能在即将到来的 C#和 Unity 旅程中获得最大收益，你唯一需要的是保持好奇心和学习意愿。话虽如此，如果你希望巩固所学知识，那么完成所有的代码练习、“英雄的试炼”和“小测验”等部分是必须的。最后，在继续学习新知识之前，重温已学过的知识点或整个章节来刷新或巩固理解不失为一个好主意。毕竟，在不稳定的地基上建造房子是没有意义的。

此外，还需要在计算机上安装当前版本的 Unity——建议使用 2022 或更新版本。所有代码示例均已在 Unity 2022.1 上进行了测试，它们应该可以在未来的版本中正常运行。

本书涉及的软件和硬件
Unity 2022.1 或更高版本
Visual Studio 2019 或更高版本
C# 8.0 或更高版本

在开始之前，请参考以下链接，检查计算机设置是否满足 Unity 的系统要求：https://docs.unity3d.com/2022.1/Documentation/Manual/system-requirements.html。

本书的代码库和彩图可扫描封底二维码下载。

目录

第 1 章　了解开发环境 1
1.1　技术要求 1
1.2　Unity 2022 入门 2
1.2.1　使用 macOS 7
1.2.2　创建新项目 7
1.2.3　编辑器界面导航 9
1.3　在 Unity 中使用 C# 11
1.3.1　使用 C#脚本 11
1.3.2　介绍 Visual Studio 编辑器 12
1.3.3　同步 C#文件 14
1.4　浏览技术文档 15
1.4.1　访问 Unity 的技术文档 15
1.4.2　查找 C#资源 18
1.5　本章小结 19
1.6　小测验 —— 关于脚本 20

第 2 章　编程的基本构建块 21
2.1　定义变量 21
2.1.1　名字很重要 22
2.1.2　变量充当占位符 23
2.2　了解方法 26
2.2.1　方法驱使行为 26
2.2.2　方法也是占位符 27
2.3　类 29
2.3.1　一个常见的 Unity 类 30
2.3.2　类就像蓝图 30
2.3.3　类之间的通信 31
2.4　使用注释 32

2.4.1 单行注释……32
2.4.2 多行注释……32
2.4.3 添加注释……33
2.5 整合基本构建块……33
2.5.1 将脚本变成组件……34
2.5.2 来自 MonoBehaviour 的助力……36
2.6 本章小结……37
2.7 小测验——C#的基本构建块……37

第 3 章 深入了解变量、类型和方法……39
3.1 编写符合规范的 C#代码……40
3.2 调试代码……41
3.3 深入了解变量……42
3.3.1 声明变量……42
3.3.2 使用访问修饰符……44
3.3.3 了解数据类型……46
3.3.4 命名变量……51
3.3.5 了解变量的作用域……52
3.4 运算符……53
3.5 定义方法……55
3.5.1 声明方法……56
3.5.2 命名约定……58
3.5.3 方法是逻辑上的绕行道……58
3.5.4 指定参数……59
3.5.5 指定返回值……60
3.5.6 使用返回值……61
3.6 本章小结……64
3.7 小测验——变量和方法……65

第 4 章 控制流和集合类型……67
4.1 选择语句……67
4.1.1 if-else 语句……68
4.1.2 switch 语句……77
4.1.3 小测验 1——if 语句与逻辑运算符……82

4.2 初识集合……82
4.2.1 数组……82
4.2.2 列表……86
4.2.3 字典……89
4.2.4 小测验 2—— 关于集合的一切……92
4.3 迭代语句……92
4.3.1 for 循环……92
4.3.2 foreach 循环……96
4.3.3 while 循环……98
4.3.4 超越无限……100
4.4 本章小结……101

第 5 章 使用类、结构体以及面向对象编程……103
5.1 什么是面向对象编程……104
5.2 定义类……104
5.2.1 实例化类对象……105
5.2.2 添加类字段……106
5.2.3 使用构造方法……107
5.2.4 声明类方法……110
5.3 声明结构体……111
5.4 了解引用类型和值类型……114
5.4.1 引用类型……114
5.4.2 值类型……115
5.5 植入面向对象的思维……116
5.5.1 封装……117
5.5.2 继承……118
5.5.3 组合……120
5.5.4 多态……121
5.6 在 Unity 中应用面向对象编程……122
5.6.1 对象是类的行动……123
5.6.2 访问组件……124
5.7 本章小结……128
5.8 小测验——面向对象编程的那些事……128

第 6 章 亲手实践 Unity ····· 129
6.1 游戏设计入门 ····· 129
6.1.1 游戏设计文档 ····· 130
6.1.2 Hero Born 游戏单页文档 ····· 131
6.2 构建关卡 ····· 131
6.2.1 创建原始对象 ····· 132
6.2.2 用 3D 思维思考 ····· 134
6.2.3 材质 ····· 135
6.2.4 白盒 ····· 138
6.3 光照基础 ····· 147
6.3.1 创建光源 ····· 147
6.3.2 Light 组件属性 ····· 148
6.4 在 Unity 中制作动画 ····· 149
6.4.1 使用代码创建动画 ····· 150
6.4.2 在 Unity 中使用 Animation 窗口创建动画 ····· 151
6.4.3 录制关键帧 ····· 154
6.4.4 曲线和切线 ····· 156
6.5 本章小结 ····· 158
6.6 小测验——Unity 的基本功能 ····· 159

第 7 章 角色移动、摄像机以及碰撞 ····· 161
7.1 管理玩家移动 ····· 161
7.2 使用 Transform 组件移动玩家 ····· 162
7.2.1 理解向量 ····· 164
7.2.2 获取玩家输入 ····· 166
7.2.3 角色移动 ····· 167
7.3 编写摄像机行为脚本 ····· 170
7.4 使用 Unity 物理系统 ····· 172
7.4.1 刚体运动 ····· 174
7.4.2 碰撞体和碰撞 ····· 178
7.4.3 使用碰撞触发器 ····· 181
7.4.4 物理系统综述 ····· 185
7.5 本章小结 ····· 186

7.6 小测验——玩家控制和物理系统……186

第 8 章 游戏机制脚本编写……187

8.1 添加跳跃……187

8.1.1 枚举介绍……188

8.1.2 使用层遮罩……191

8.2 发射子弹……196

8.2.1 实例化对象……196

8.2.2 添加射击机制……198

8.2.3 管理对象堆积……201

8.3 创建游戏管理器……202

8.3.1 追踪玩家属性……202

8.3.2 get 和 set 属性……204

8.3.3 更新道具收集……206

8.4 创建 GUI……208

8.4.1 显示玩家数据……208

8.4.2 胜负条件……216

8.4.3 使用指令和命名空间暂停和重启游戏……221

8.5 本章小结……225

8.6 小测验——游戏机制……225

第 9 章 AI 基础与敌人行为……227

9.1 在 Unity 中导航 3D 空间……227

9.1.1 导航组件……228

9.1.2 设置敌人代理……230

9.2 移动敌人代理……231

9.2.1 面向过程编程……231

9.2.2 引用巡逻点……232

9.2.3 移动敌人……234

9.3 敌人游戏机制……238

9.3.1 寻找和销毁：改变代理的目标……238

9.3.2 减少玩家生命值……240

9.3.3 检测子弹碰撞……241

9.3.4 更新游戏管理器……244

9.4 重构并保持“DRY”……246
9.5 本章小结……248
9.6 小测验——AI 和导航……248

第 10 章 重睹类型、方法和类……249
10.1 再谈访问修饰符……249
10.1.1 常量和只读属性……250
10.1.2 使用 static 关键字……250
10.2 重睹方法……252
10.2.1 重载方法……252
10.2.2 ref 参数……254
10.2.3 out 参数……256
10.3 中级的面向对象编程……257
10.3.1 接口……257
10.3.2 抽象类……261
10.3.3 类的扩展……263
10.4 命名空间冲突和类型别名……266
10.5 本章小结……267
10.6 小测验——升级……268

第 11 章 特殊集合类型和 LINQ……269
11.1 栈……269
11.1.1 出栈(Pop)和查看栈顶(Peek)……273
11.1.2 常用方法……274
11.2 队列……275
11.3 哈希集……277
11.4 关于中级集合的小结……279
11.5 用 LINQ 查询数据……280
11.5.1 LINQ 基础……280
11.5.2 lambda 表达式……283
11.5.3 链式查询……284
11.5.4 将数据转换为新类型……285
11.5.5 用可选语法化简……286
11.6 本章小结……288

11.7 小测验——中级的集合……288

第 12 章 保存、加载与数据序列化……289
12.1 数据格式……289
12.1.1 解构 XML……290
12.1.2 解构 JSON……291
12.2 了解文件系统……293
12.2.1 使用资产路径……295
12.2.2 创建或删除目录……297
12.2.3 创建、更新、删除文件……299
12.3 使用数据流……305
12.3.1 管理数据流资源……306
12.3.2 使用 StreamWriter 与 StreamReader……306
12.3.3 创建一个 XMLWriter……310
12.3.4 自动关闭数据流……313
12.4 数据序列化……314
12.4.1 序列化和反序列化 XML……315
12.4.2 序列化和反序列化 JSON……319
12.5 数据汇总……325
12.6 本章小结……326
12.7 小测验——数据管理……326

第 13 章 探索泛型、委托以及更多……327
13.1 泛型……327
13.1.1 泛型类……328
13.1.2 泛型方法……330
13.1.3 约束类型参数……333
13.1.4 向 Unity 对象添加泛型……336
13.2 委托行动……337
13.2.1 创建调试委托……337
13.2.2 将委托作为参数类型……339
13.3 触发事件……340
13.3.1 创建并调用事件……341
13.3.2 处理事件订阅……342

13.3.3 清理事件订阅……344
13.4 处理异常……345
13.4.1 抛出异常……345
13.4.2 使用 try-catch 语句……347
13.5 本章小结……350
13.6 小测验——中级 C#……350

第 14 章 旅程继续……351
14.1 有待更深入的知识……351
14.2 牢记面向对象编程……352
14.3 设计模式入门……353
14.4 走进 Unity 项目……353
14.5 本书未提及的 Unity 特性……354
14.6 下一步的学习……354
14.6.1 C#资源……354
14.6.2 Unity 资源……355
14.6.3 Unity 认证……355
14.7 英雄的试炼 —— 向全世界展示……356
14.8 本章小结……357

小测验答案……359

第1章
了解开发环境

流行文化喜欢将程序员描绘为局外人、独行侠或极客黑客，这些人拥有算法思维的天赋，但缺乏社交和情商。虽然事实并非如此，但学习编程的确会从根本上改变一个人看待世界的方式。好在出于好奇的天性，人类的头脑会很快适应这种看待世界的新方式，甚至还可能会喜欢上这种新的思维方式。

从清晨睁开眼睛的一霎那直至睡前望着吊扇的最后一瞥，我们都在不知不觉中使用了可以直接转化为编程的分析技巧，只是缺乏正确的语言和语法将这些生活技能映射为代码。例如，现实中每个人的年龄，可以看作编程中的变量；再如，现实中我们习惯在过马路前观察道路两边，这相当于在编程中评估不同的条件，专业术语称为控制流；又如，现实中看到一罐汽水，我们会本能地认出它具有形状、重量和内容物等属性，这就好比编程中的一个类对象！以此类推，这样的例子还有很多。

有了这些真实世界的经验，你已经完全准备好进入编程的世界。为了更好地开始旅程，还需要知道如何设置开发环境，如何使用涉及的应用程序，并确切地知道该去哪里寻求帮助。

为此，我们将从以下几个C#主题开始。

- Unity 2022 入门
- 在 Unity 中使用 C#
- 探索技术文档

让我们开始吧！

1.1 技术要求

有时候，使用反向排除胜过正向列举。本书的主要目标不是掌握 Unity 游

戏引擎或游戏开发中的种种细节。但在旅程开始之前，有必要对 Unity 和游戏开发有一个基础的认知，更多详情会在第 6 章中展开。这些主题会帮助我们以一种有趣轻松的方式从零开始学习 C#语言，但不是深入的 Unity 操作教程。因为本书以编程作为主要目标，所以有时会更倾向于选择一个基于代码的解决方案，即使 Unity 可能有一个特定的功能可以在不需要任何代码的情况下做同样的事情。

本书面向编程初学者，特别适合没有任何 C#或 Unity 使用经验的读者！也适合没有编程经验但对 Unity 编辑器有些了解的读者。此外，即便是对上述两方面均有涉猎，但希望探索一些中高级主题的读者，也同样可以从本书的后几个章节的相关内容中获益。

如果已经熟练掌握其他编程语言，可略过初学者理论，直接跳至感兴趣的部分。当然，也可选择从头学起，重温基础知识。

学习中除了需要运行 Unity 2022 以外，还将使用 C# 8.0 和 Visual Studio 来编写游戏代码。

1.2 Unity 2022 入门

如果你还没有安装 Unity，或者正在运行一个较早的版本，可按照以下步骤设置你的环境。

(1) 访问网址：https://unity.cn/。

(2) 单击“下载 Unity”按钮(如图 1-1 所示)。

图 1-1 Unity 首页

(3) 这将跳转到 Unity 下载页面。不必惊慌，Unity 可以完全免费使用！

说明：

如果 Unity 首页看起来和截图中看到的不同，你可以直接进入 https://unity.cn/releases。

(4) 单击“下载 Unity Hub”按钮(见图 1-2)。

图 1-2　下载 Unity

(5) 选择适合的系统下载 Unity Hub 应用程序，如图 1-3 所示。尽管本书使用的是 Mac 系统，但几乎所有操作与在 Windows 系统上是相同的。

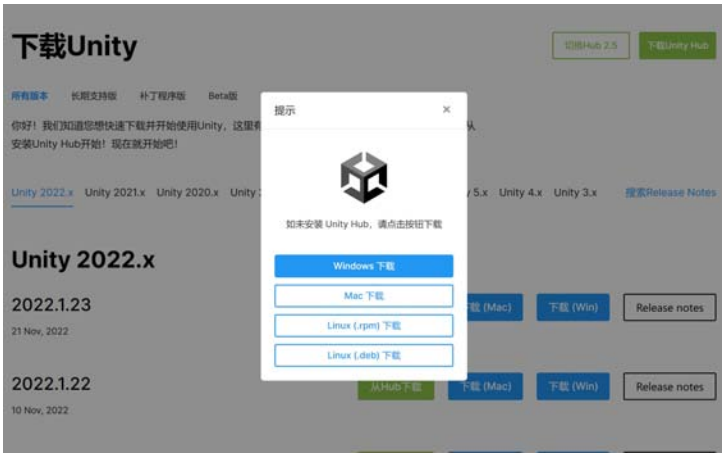

图 1-3　下载 Unity Hub

(6)下载完成后，按照以下步骤继续。

① 打开安装包(双击它)。

② 接受用户协议。

③ 按照安装说明执行。

(7) 安装成功后，启动 Unity Hub 应用程序！

(8) 如果 Unity 提示选择许可证类型，请选择个人版许可类型(它是完全免费的)，并根据指引设置你的账户。

(9) 最新版的 Unity Hub 会鼓励安装最新的 LTS(长期支持)版本的 Unity 编辑器，如图 1-4 所示。但在撰写本书时 Unity 2022 仍处于预览版，因此如果在阅读本教程时默认的版本是 Unity 2022，请直接选择并单击 Install Unity Editor(安装 Unity 编辑器)。

图 1-4 Install Unity Editor 窗口

(10) 如果此时你看到的默认版本并非 Unity 2022，请选择窗口右下角的跳过安装(Skip Installation)，如图 1-5 所示。

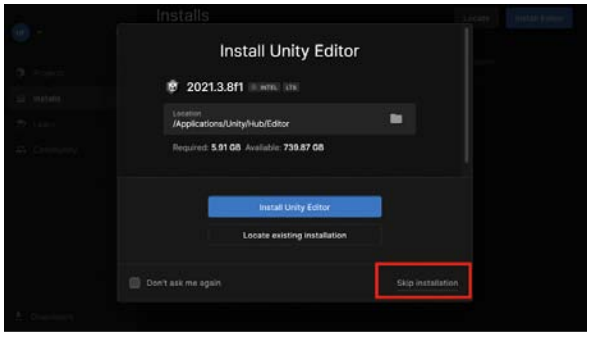

图 1-5 安装引导

(11) 从左侧列表中选择并切换到安装(Installs)标签，选择安装编辑器(Install Editor)，如图 1-6 所示。

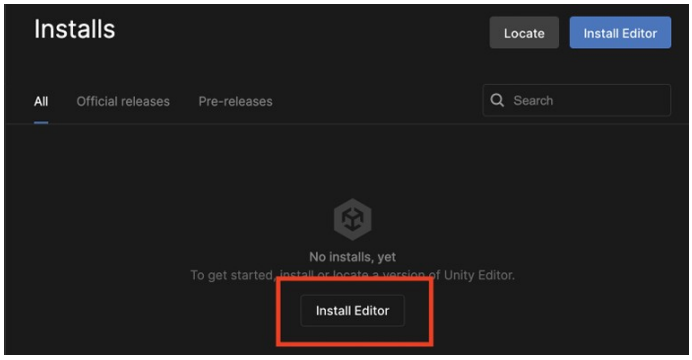

图 1-6　Unity Hub 的安装面板

(12) 选择希望安装的 Unity 版本，然后单击 Install“安装”进行安装。在撰写本书时，Unity 2022 还被列在 Official releases(正式发行版本)标签下的 OTHER VERSIONS(其他版本)中，但也许在你阅读本书的时刻，应该已经可以在 Official releases(正式发行版本)的列表中找到 2022 版本了(见图 1-7)。

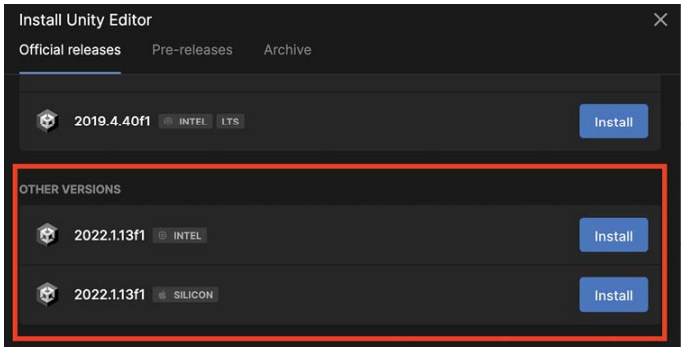

图 1-7　在弹出窗口中添加 Unity 版本

(13) 接下来将会有机会选择多种多样的模块并将其一同添加到安装过程中。务必确保 Visual Studio 模块(Windows 或 Mac 版本)被选中，然后单击“继续”(Continue)按钮，如图 1-8 所示。

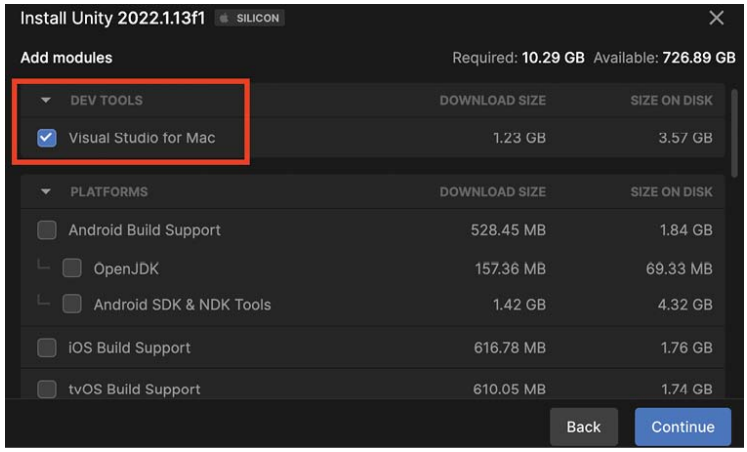

图 1-8　添加要安装的模块

(14) 如果之后任何时刻想要添加模块，可以在“安装”(Installs)窗口中单击任何已安装 Unity 编辑器版本右侧的齿轮图标。

当安装完成后，将会在“安装”(Installs)面板中看到新的版本，如图 1-9 所示。

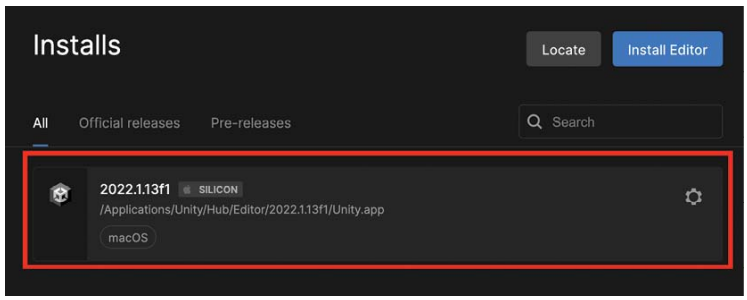

图 1-9　“安装”面板与 Unity 版本

说明：

有关 Unity Hub 应用程序的其他信息和资源可访问 https://docs.unity3d.com/Manual/GettingStartedInstallingHub.html。

有些出入是难免的，如果你使用的是 Mac OS Catalina 或更高版本的 Mac 系统，一定要查看接下来的内容。

1.2.1　使用 macOS

如果你正在使用 Catalina 或更高版本的 Mac 操作系统，那么使用 Unity Hub 根据上述步骤安装 Unity 会存在已知问题。如果发生了这样的情况，没关系，放轻松，可以进入 Unity 下载页面并获取所需版本(https://unity.cn/releases)。记住使用“下载(Mac)”或“下载(Win)”选项而不是“从 Hub 下载”选项进行下载，如图 1-10 所示。

图 1-10　Unity 下载页面

说明：

在应用程序安装包下载完毕后打开并根据安装指引进行安装。本书的所有示例和屏幕截图都是使用 Unity 2022.1.13f1 版本创建和截取的。如果你正在使用较新的版本，那么在 Unity 编辑器中的情况可能会略有不同，但这通常不会影响后续的学习和操作。

至此，Unity Hub 和 Unity 2022 已经安装好了，是时候创建一个新项目了！

1.2.2　创建新项目

启动 Unity Hub 应用程序，我们将从这里“登场”。在这里可以查看所有项目的列表和 Unity 版本，而且可以访问学习资源和社区功能等。接下来，请根据下方的步骤操作。

(1) 首先，单击右上角的“New project”选项，如图 1-11 所示。

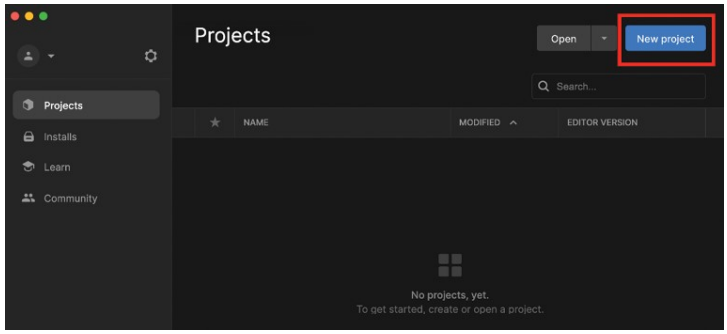

图 1-11　Unity Hub 的项目管理面板

(2) 确保将位于顶端的编辑器版本设置为 2022 版本，并根据下方指引设置相关字段区域。

- 模板(Templates)：此项目将默认选择 3D 核心模板。
- 项目名称(Project name)：将项目命名为“Hero Born”。
- 位置(Location)：选择希望保存此项目的路径。

(3) 当上述都设置完成后，单击“创建项目”(Create Project)。

创建项目后，便可以开始探索 Unity 界面了！在任何时刻，我们都可以从 Unity Hub 的“项目”(Projects)面板重新打开项目，但如果感觉同时打开并运行 Unity 和 Unity Hub 时计算机运行有点缓慢，则可以随时关闭 Unity Hub(见图 1-12)。

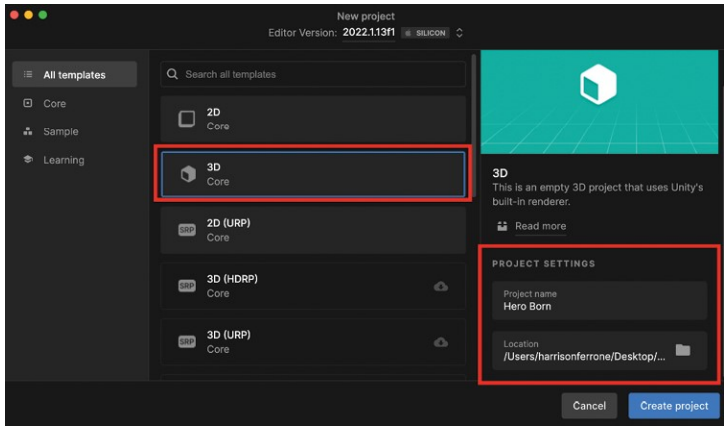

图 1-12　Unity Hub 中新项目设置的弹出窗口

1.2.3　编辑器界面导航

新项目完成初始化后，将看到漂亮的 Unity 编辑器界面！图 1-13 中对重要的选项卡(也称面板)进行了标注。

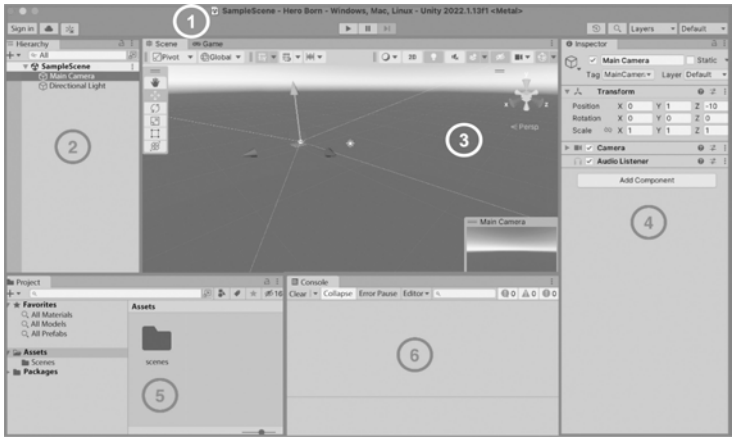

图 1-13　Unity 界面

因为一口气记住所有面板需要花一点时间，所以先详细地介绍每个面板。

(1) Toolbar(工具栏)面板位于Unity编辑器的最顶部。在这里，可以登录Unity账号、管理服务、与团队协作(最左边的按钮组)，以及运行和暂停游戏(位于中央的几个按钮)。最右边的按钮组包含了搜索功能、图层遮罩和布局方案等功能，但由于它们在学习 C#的过程中不会被用到，因此我们在本书中将不会用到它们。

(2) Hierarchy(层级)面板显示了当前游戏场景中的每个物体。在新建的入门项目中，这里默认只有摄像机和方向光。但是当我们创建原型环境时，此面板将会被我们在场景中创建的物体对象填充。

(3) Game(游戏)和 Scene(场景)视图面板是编辑器最直观的区域。可以将场景视图视为舞台，我们可以在其中移动和摆放 2D 和 3D 物体。当按下 Play(运行)按钮时，将自动切换至游戏视图，在其中展示渲染后的场景视图和所有已编程的交互。在按下 Play 按钮后仍然可以使用“场景”视图。

(4) Inspector(检视)面板是一站式查看和编辑所选对象属性的区域。例如，当在 Hierarchy 中选中 Main Camera(主摄像机)时(在图 1-13 中被蓝色高亮显示)，该面板会显示它包含的几个“部分”，这些“部分”在 Unity 中称为组件，所有组件都可以从此面板访问。

(5) Project(项目)面板包含项目中当前存在的每个资产。可以将其视为项目文件夹和文件的展示区域。

(6) 在 Console(控制台)面板上，将显示我们希望脚本打印的任何输出。从现在开始，如果我们提及控制台或调试输出，便到该面板查看相关结果。

如果不小心关闭上述任一面板，则可以随时通过 Unity 菜单 > Window > General 再次打开它们。有关面板功能的深入解析可查看 Unity 文档：https://docs.unity3d.com/Manual/UsingTheEditor.html。

在继续之前，有必要明确 Visual Studio 已经被设置为项目的脚本编辑器。通过 Unity 菜单选择 Edit > Preferences > External Tools，并检查 External Script Editor(外部脚本编辑器)是否已设置为 Visual Studio for Mac(或 for Windows)。见图 1-14。

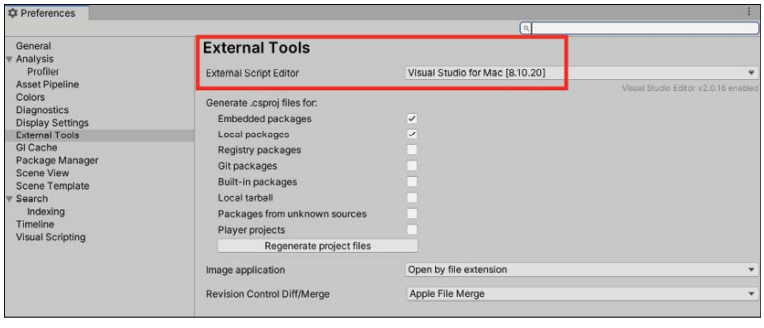

图 1-14　将 External Script Editor(外部脚本编辑器)设置为 Visual Studio

小提示：

如果希望在浅色模式(light mode)或深色模式(dark mode)之间切换，可以通过 Unity 菜单选择 Edit > Preferences > General 并改变 Editor Theme(编辑器主题)来实现，如图 1-15 所示。

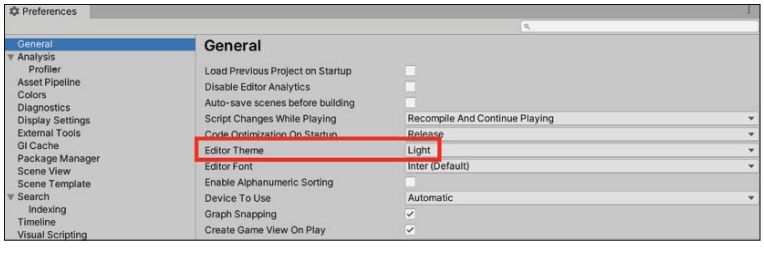

图 1-15　Unity 的一般偏好设置面板

对于 Unity 初学者，的确有很多东西要记忆和消化，但请放心，在后续的所有内容中我们都会对必要步骤进行指引，避免使你犹豫不决或不知道该单击哪个按钮。排除了这些顾虑之后，就让我们开始创建一些真正的 C#脚本吧。

1.3　在 Unity 中使用 C#

从现在开始，很有必要将 Unity 和 C#视为相互共生的关系。在 Unity 引擎中可以创建脚本和游戏对象(Game Objects)，但实际的编程却是在另一个名为 Visual Studio 的软件中进行的。

1.3.1　使用 C#脚本

尽管尚未介绍任何的基本编程概念，但作为基础中的基础，应该先知道如何在 Unity 中创建实际的 C#脚本。C#脚本是一种特殊的 C#文件，在其中可以编写 C#代码。这些脚本可以被用于 Unity 以虚拟地实现任何事情，从通过键盘控制游戏内角色到控制关卡中物体的动画等。

从 Unity 编辑器中创建 C#脚本的方法有以下几种。

- 选择 Assets > Create > C# Script。
- 在 Project 选项卡中，选择加号图标并选择 C# Script。
- 右击 Project 选项卡中的 Assets(资产)文件夹，然后在弹出菜单中选择 Create > C# Script。
- 在 Hierarchy 窗口中选择一个 GameObject(游戏对象)，在 Inspector 中，单击 Add Component > New Script。

之后每当提到创建 C#脚本时，请使用以上任何一种方法。

也可以使用上述方法在编辑器中创建 C#脚本以外的资源和对象。每当我们创建新内容时，并不会明确指定使用这些方法中的哪一种变化形式，因此有必要将这些方法的变化形式都熟练记住。

为了便于组织和管理，通常习惯将各种资产和脚本存储在其对应名称的文件夹中。这虽不是 Unity 要求的做法，但尽早养成好的习惯会受到同行的欢迎和感谢。

(1) 选择 Assets > Create > Folder，并将其命名为 Scripts，如图 1-16 所示。

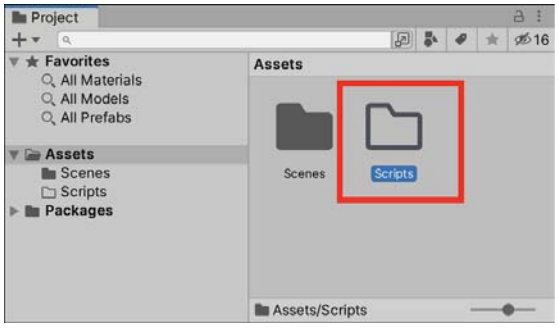

图 1-16　创建一个 C#脚本

(2) 双击 Scripts 文件夹并在其中创建一个新的 C#脚本。默认情况下，该脚本将被命名为 NewBehaviourScript，且文件名被高亮显示，此时可以立即对其进行重命名。输入 LearningCurve，然后按下键盘上的 Enter 键。见图 1-17。

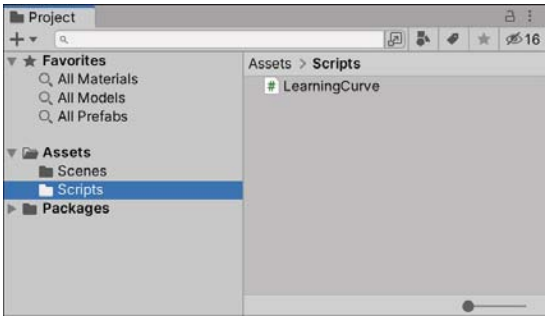

图 1-17　Project(项目)面板，当 Scripts 文件夹被选中时

(3) 可以使用 Project(项目)面板右下角的小滑杆来调节文件呈现的方式。

我们刚刚创建了一个名为 Scripts 的子文件夹，如前面的屏幕截图所示。在该文件夹内，我们创建了一个名为 LearningCurve.cs 的 C#脚本(.cs 文件类型代表 C-Sharp，即 C#)，且已将其保存为项目 Hero Born 资产中的一部分。剩下要做的就是在 Visual Studio 中打开它！

1.3.2　介绍 Visual Studio 编辑器

虽然在 Unity 中可以创建和存储 C#脚本，但是需要使用 Visual Studio 对其进行编辑。Visual Studio 与 Unity 被预捆绑在一起，当从编辑器内部双击任何 C#

脚本时，将自动通过 Visual Studio 打开。

1. 打开 C#文件

首次打开文件时，Unity 将与 Visual Studio 同步。最简单的方法是从 Projects 选项卡中选择脚本。具体步骤如下。

(1) 双击 C#脚本文件“LearningCurve.cs”，会在 Visual Studio 中打开它，见图 1-18。

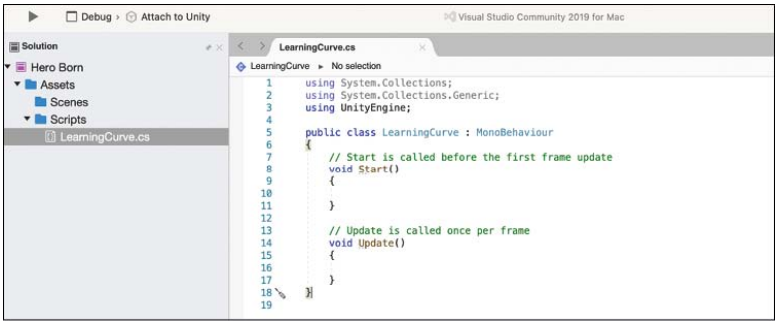

图 1-18　在 Visual Studio 中打开的 LearningCurve.cs 脚本

(2) 可以随时通过 Visual Studio > View > Layout 来改变 Visual Studio 的标签。本书将使用 Design 布局以便可以在编辑器的左侧看到项目文件目录。

(3) 在 Visual Studio 界面的左侧会看到一个文件夹结构，该文件夹结构是 Unity 中文件夹结构的镜像，可以用访问其他文件夹同样的方式访问它。右侧便是生成一切魔法的代码编辑器。 Visual Studio 应用程序还有许多功能，但目前这就是使程序运行所需要了解的全部内容。

Visual Studio 的界面在 Windows 和 Mac 环境中有所不同，但是本书中将使用的代码在两种环境下均能很好地工作。本书中的所有屏幕截图均是在 Mac 环境中截取的，因此，如果你在计算机上看到的外观有所不同，不必担心。

2. 当心命名不匹配

困扰编程新手的一个常见的陷阱是对文件命名。更准确地说，是命名不匹配。我们可以使用之前 Visual Studio 中 C#文件的屏幕截图 1-18 中代码的第 5 行来说明。

```
public class LearningCurve : MonoBehaviour
```

LearningCurve 这个类名称与 LearningCurve.cs 文件名相同。这是一个基本要求。如果你还不知道什么是类，那没关系。但要记住的重点是，在 Unity 中文件名和类名必须相同。如果在 Unity 之外使用 C#，则文件名和类名并不必须匹配。

在 Unity 中创建 C#脚本文件时，Project 面板中的文件名自动处于 Edit(编辑)模式，意味着已为重命名做好准备。因此，在此处重命名是一个好习惯。如果稍后重命名脚本，则文件名和类名可能会不匹配。

如果选择了稍后更改文件名，尽管文件名会发生改变，但此时第 5 行会变成类似下方所示的样子。

```
public class NewBehaviourScript : MonoBehaviour
```

如果不小心这样操作了，也并不是什么末日灾难，所需要做的就是在 Project 面板中右击该脚本并选择 Rename(重命名)，见图 1-19。

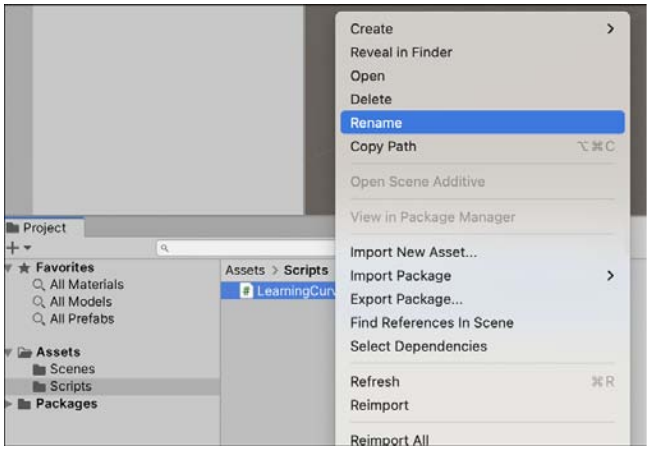

图 1-19　重命名 C#脚本

1.3.3　同步 C#文件

作为它们共生关系的一部分，Unity 和 Visual Studio 彼此保持联系以同步其内容。这意味着，如果在其中一个应用程序中添加、删除或更改脚本文件，则另一个应用程序将自动接收到所做的更改。

正如墨菲定律中说的“凡是可能出错的事，就一定会出错”，万无一失的事物并不存在，那么当脚本没能正常同步时，会怎样呢？如果遇到这种情况，不

必惊慌，选中那个出现麻烦的脚本，单击鼠标右键，然后选择 Refresh(刷新)，如图 1-20 所示。

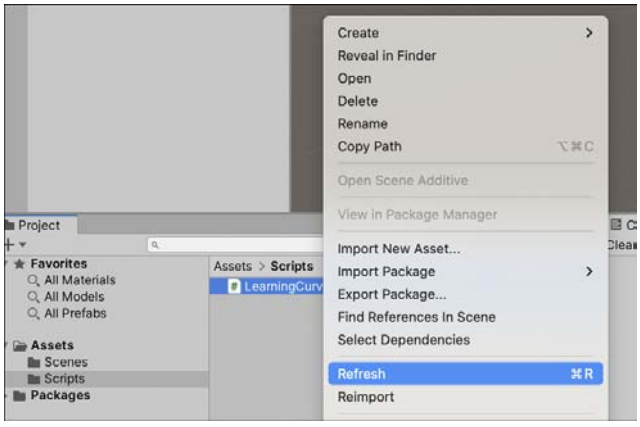

图 1-20　刷新 C#脚本

现在我们已经掌握了脚本创建的基础知识，该是讨论如何查找和更有效率地使用帮助资源的时候了。

1.4　浏览技术文档

在对 Unity 与 C#脚本的首次尝试中，要涉及的最后一个话题是浏览技术文档。这个话题并不吸引人，但在与新的编程语言或开发环境打交道前培养一个良好的习惯很重要。

1.4.1　访问 Unity 的技术文档

一旦开始认真编写脚本，就会经常使用 Unity 的技术文档，因此尽早了解如何访问它是有益的。参考手册(Reference Manual)为我们提供了某一组件或主题的概述，而特定的编程示例可以在脚本参考(Scripting Reference)中找到。

场景中的每个游戏对象 GameObject(Hierarchy 面板中的一个物体)都具有一个控制其 Position(位置)、Rotation(旋转)和 Scale(缩放比例)的 Transform(变换)组件。为简单起见，我们将在参考手册中查看摄像机的 Transform 组件。

(1) 在 Hierarchy 面板中，选中 Main Camera(主摄像机)游戏对象。

(2) 将视线移至 Inspector 面板，单击 Transform 组件右边的信息图标(？)，见图 1-21。

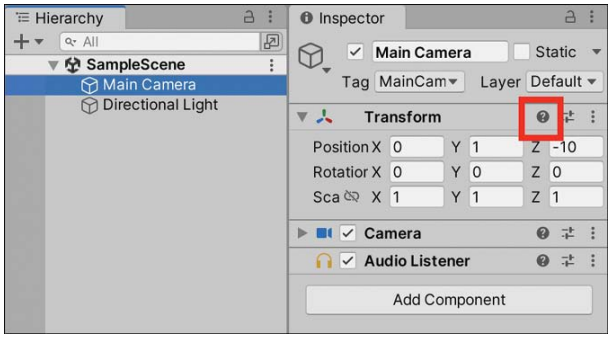

图 1-21 在 Inspector 中选中 Main Camera 游戏对象

(3) 网络浏览器会打开并显示参考手册中的 Transforms 页面，见图 1-22。

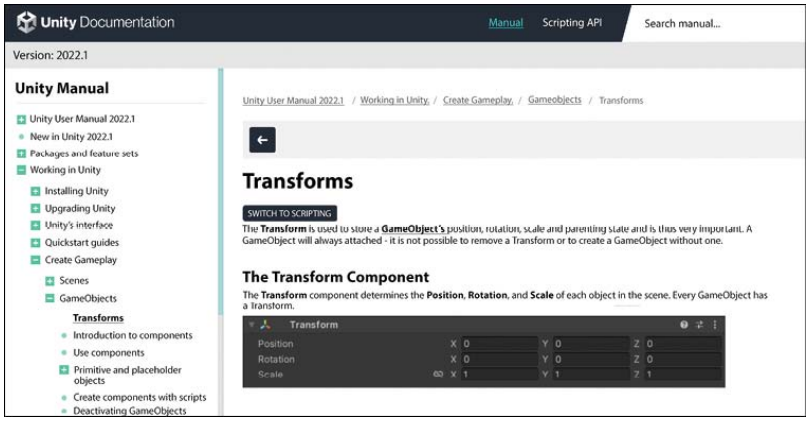

图 1-22 Unity 的参考手册

提示说明：

Unity 中的所有组件都具有此功能，因此，每当我们想进一步了解某件事情的工作原理时，便可以用该方式查看。

至此，我们已经打开了参考手册，但如果想要查看与 Transform 组件相关的具体代码示例，该怎么办呢？这很简单，所要做的就是查询脚本参考。

(1) 单击组件或类名称(在本例中为 Transforms)下方的 SWICTH TO

SCRIPTING(切换到脚本)链接，见图 1-23。

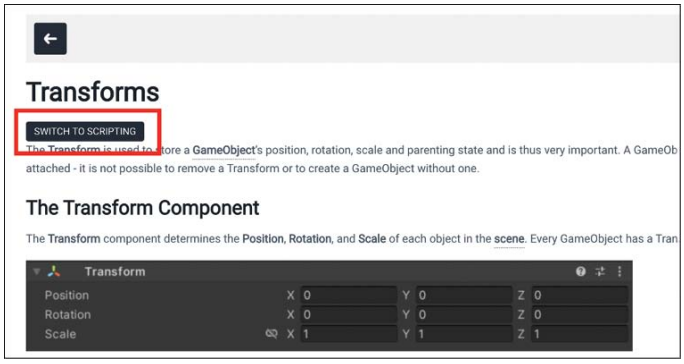

图 1-23　Unity 参考手册，SWICTH TO SCRIPTING 按钮高亮显示

(2) 操作后，参考手册会自动切换到 Transform 组件的脚本参考页面，见图 1-24。

图 1-24　Unity 脚本参考文档，此时按钮变为 SWITCH TO MANUAL

(3) 可以看到，在必要时可以通过 SWITCH TO MANUAL 选项切换回参考手册。

脚本参考是一个庞大的文档，因为需要囊括大量内容。但这并不意味着我们必须记住或熟悉其所有内容才能开始编写脚本。如其名字所示，它只是一个参考，而不是考试。

提示说明：

如果发现自己迷失在文档中，或者不知从何处看起，还可以在 Unity 开发社区中的以下位置找到丰富的解决方案。

- Unity Forum (https://forum.unity.com/)
- Unity Answers (https://answers.unity.com/index.html)
- Unity Discord (https://discord.com/invite/unity)

另一方面，还需要知道在哪里可以找到可以解答任何 C#问题的资源，我们将在接下来的内容中进行介绍。

1.4.2 查找 C#资源

我们已经了解如何查看 Unity 资源，下面看一下 Microsoft 的一些 C#资源。首先，位于 https://docs.microsoft.com/en-us/dotnet/csharp 的微软学习文档包含了大量优秀的教程、快速开始指引以及指南文章。你也可以在下面这个地址找到针对单独 C#话题的极佳概述：https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/index。

然而，如果想要了解某一特定 C#语言功能的详细信息，参考指南可以派上用场。这些参考指南对任何 C#编程者而言都是非常重要的资源，但鉴于它们不总是最易于定位的，因此这里让我们花一点时间学习如何找到我们想要的内容。

打开编程指南链接并查找 C#中的 String 类。可执行以下任一操作。

- 在网页左上角的搜索栏中输入 Strings。
- 向下滚动至 Language Sections，然后直接单击 Strings 链接。见图 1-25。

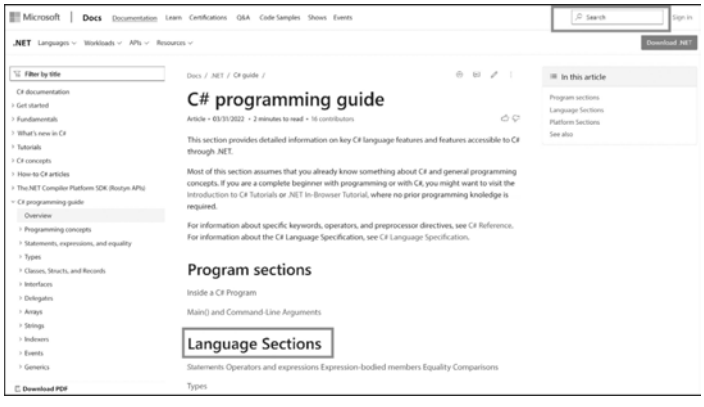

图 1-25 在微软 C#参考指南中定位

应该在类描述页面上看到类似以下页面的内容，如图 1-26 所示。

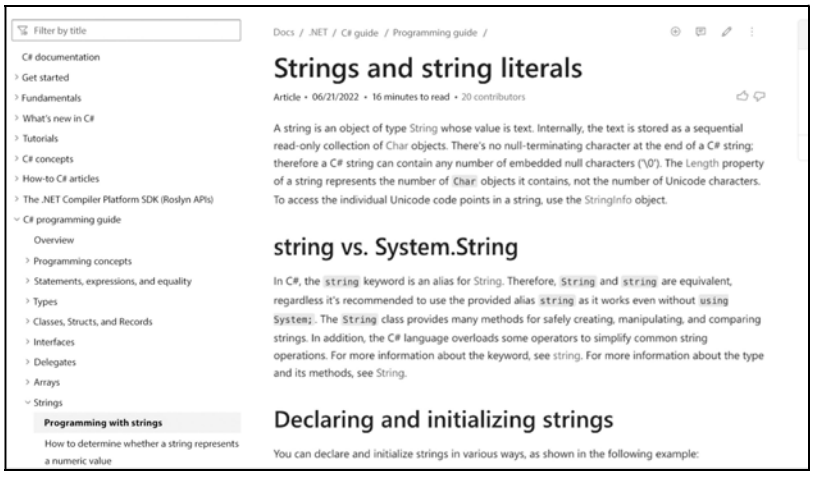

图 1-26　微软的 Strings(C#编程指南)页面

与 Unity 文档不同，C#参考文档和脚本示例信息全部整合在一起，但是幸好有其右侧的子主题列表，应该充分地利用它！遇到困难或有疑问时，知道在哪里寻求帮助是极其重要的，因此在遇到障碍时请务必重温本节。

1.5　本章小结

本章介绍了很多预备知识，想必你已经迫不及待地想要编写一些代码了。在接下来令人兴奋的旅程中，如何新建一个项目、创建文件夹和脚本以及如何访问技术文档常常容易被遗忘。只要记住，本章为接下来的学习提供了很多资源，因此不要怕返回这里重新查看。编程思维如同肌肉记忆：训练得越多，就会变得越强大。

在第 2 章中，我们将开始为大脑铺垫编程所需的理论、词汇和主要概念。我们会在 LearningCurve 脚本中编写第一行代码，尽管内容是概念性的。做好准备吧！

1.6 小测验 —— 关于脚本

(1) Unity 和 Visual Studio 之间存在怎样的关系？

(2) 脚本参考提供了有关使用特定 Unity 组件或功能的示例代码。在哪里可以找到有关 Unity 组件的更详细(非代码相关)的信息？

(3) 脚本参考是一个大型文档。在尝试编写脚本之前你必须记住多少？

(4) 命名一个 C#脚本的最佳时机是什么时候？

第2章 编程的基本构建块

任何一种编程语言对于不熟悉它们的人来说，起初看着都像外星文，C#也不例外。但好消息是，潜藏在这些字符谜团之下，所有编程语言都由相同的基本构建块构成。一般来说，变量、方法和类(或对象)构成了编程的 DNA；理解这些简单的概念将为我们开启应用程序多样且复杂的世界。毕竟，尽管地球上每人只有四种不同的 DNA 核酸碱基，但它们却为每个人构成了独一无二的有机生命体。

如果你是编程新手，在本章中你将迎来许多新知识，这可能标志着你将编写人生的第一行代码。本章并非试图通过事实和图表使你的大脑超载，而是希望通过对照日常生活中的实例，为你提供一个了解编程基本构建块的全局视角。

本章从高层次的视角审视构成一个程序的细枝末节。在直接开始写代码之前先掌握其工作原理，不仅可以帮助编程新手找到立足之地，还可以通过易于记忆的参考案例来巩固知识点。本章我们将聚焦在以下话题：

- 定义变量
- 了解方法
- 了解类
- 如何使用注释
- 如何将各基本构建块整合在一起

2.1 定义变量

让我们从一个简单的问题开始：什么是变量？从不同的视角可以有以下几种不同的方式回答该问题。

- 从概念上讲，变量是编程的最基本单位。一切都始于变量，没有变量，程序就不可能存在。
- 从技术上讲，变量是计算机内存中保存某一赋值的一块小区域。每个变量都会跟踪其信息的存储位置(这称为内存地址)、值及其类型(例如数字、字符或列表)。
- 从实践应用上讲，变量是一个容器。我们可以随意创建新的变量，给它们填入内容，移动它们，更改它们的内容并根据需要引用它们。它们甚至可以是空的且依然有用处。

提示说明：

可以通过以下地址从微软的 C#文档中更深入地了解变量：https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/variables。

变量的一个比较贴近真实生活的示例是信箱。你还记得它们吗(见图 2-1)?

图 2-1　一排各种颜色的信箱

信箱可以存放各种东西，比如信件、账单，或来自 Mabel 姨妈的照片。其关键是邮箱中的东西可以是不同的：它们有一个名称且存放了信息(实体邮件)，并且如果拥有正确的安全权限，甚至可以改变它们的内容。类似地，变量也可以存放不同种类的信息。C#中的变量可以存放字符串(文本)、整型值(整数)以及布尔值(二元值，“真”或“假”中的一个)。

2.1.1　名字很重要

参照图 2-1，如果我要你打开信箱，你可能首先会问：打开哪一个？如果我告诉你的是史密斯家的信箱或印有向日葵的信箱或最右边那个摇摇欲坠的信箱，那么你便获得了打开我指定的信箱所必需的背景信息。相似地，在创建变

量时，也必须给它们提供唯一的名称，以供之后指明时引用。我们将在第 3 章中探讨命名的恰当格式和描述性命名的更多细节。

2.1.2　变量充当占位符

创建和命名变量时，其实是为要存储的值创建了一个占位符。以下面的简单数学等式为例：

```
2 + 9 = 11
```

好吧，这很浅显易懂。但是如果希望把数字 9 换成一个变量，该怎么办？思考以下代码：

```
MyVariable = 9
```

现在，便可以在任何需要的地方使用变量名 MyVariable 指代数值 9。

```
2 + MyVariable = 11
```

变量有其他规则或规定吗？答案是有。在第 3 章中将介绍这些内容，敬请期待。

上述示例虽不是真正的 C#代码，却也体现了变量的能力以及将其视为占位符并引用它的用法。接下来，我们将动手创建一个变量，让我们继续！

关于变量的理论已经足够，让我们在第 1 章创建的 LearningCurve 脚本中创建一个真正的变量。

(1) 在 Unity 中从 Project 面板找到并双击 LearningCurve.cs，以便在 Visual Studio 中将其打开。

(2) 在第 6 行和第 7 行之间添加一个空行，并参考下面的代码在该行中声明一个新变量。

```
public int CurrentAge = 30;
```

(3) 在 Start 方法中，添加两条调试日志语句，以打印输出以下计算的结果。

```
Debug.Log(30 + 1);
Debug.Log(CurrentAge + 1);
```

下面对上述添加的代码进行解构。首先创建了一个命名为 CurrentAge 的新变量，并同时给它赋了一个值，为 30。然后，为了显示变量是如何作为值的容器存储值的，我们添加了两条调试日志语句，用来将计算 30 + 1 与计算 CurrentAge + 1 的结果打印输出。可以发现，变量的使用方式与数值相同。

另外一个需要注意的重点是，public 的变量会出现在 Unity 的 Inspector 上，然而 private 的变量却不会。目前暂时不需要担心语法，只需要确保你的脚本与图 2-2 所示的脚本相同即可。

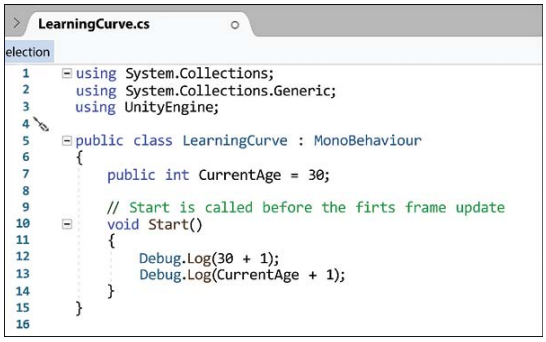

图 2-2 在 Visual Studio 中打开的 LearningCurve 脚本

最后，可以通过 Editor > File > Save 命令保存文件，或用任何操作系统支持的快捷键组合来保存文件。保存文件对于编辑脚本而言至关重要，因为对于脚本的变更，只会在脚本文件保存后再切换回 Unity 编辑器时 Unity 才会识别脚本的变化。如果在 Visual Studio 中向一个脚本添加了新的代码，却没有保存的话，Unity 将不会识别。

为了使脚本能在 Unity 中运行，必须将它们附着到场景中的 GameObjects 上。Unity 将游戏中的任何事物都视为 GameObject(游戏对象)，例如光源、玩家角色、物品、建筑，一切的一切都是游戏对象。

Hero Born 项目默认在初始仅有一个用于渲染场景的摄像机，以及一个方向光，该光源为场景提供照明。简单起见，让我们将 LearningCurve 脚本附着到摄像机上。

(1) 将 LearningCurve.cs 脚本拖放到 Main Camera 上。见图 2-3。

(2) 选中 Main Camera，使其出现在 Inspector 面板中，并查验是否已正确地给主摄像机加上了 LearningCurve.cs(Script)组件。

(3) 单击 Play 按钮，并在 Console(控制台)面板中查看输出。

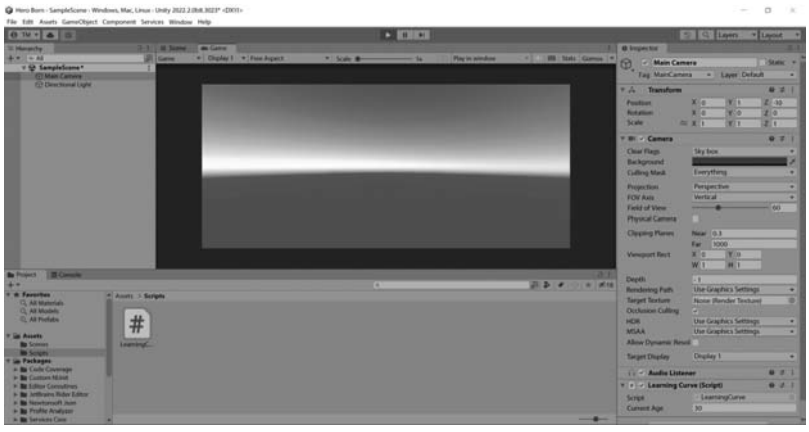

图 2-3　在 Unity 编辑器中拖曳并释放脚本

此时你可能已经注意到，在游戏运行时编辑器界面变得略暗，且 Play 按钮变为蓝色。这是因为 Unity 具有两种状态：编辑器状态和运行时(runtime)状态。编辑脚本或向场景中添加物体时，应处于编辑器状态下。在此状态下，任何的改变都将保存到项目中。然而，在按下了 Play 按钮之后，Unity 将切换到运行时状态。在游戏运行时所做的任何修改都将不会保存，因此在此状态下需要额外注意所做的更改。

Debug.Log()语句会打印输出放在括号内的简单数学表达式的结果。如下面的 Console(控制台)屏幕截图(见图 2-4)所示，使用变量 CurrentAge 的表达式与使用数字的表达式结果相同。

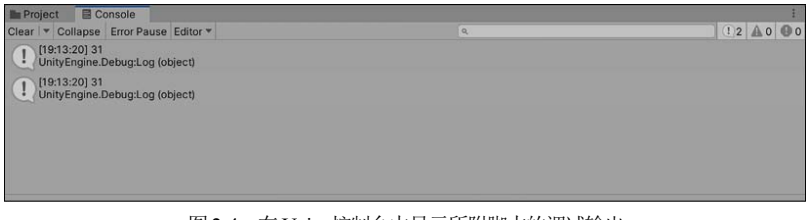

图 2-4　在 Unity 控制台中显示所附脚本的调试输出

在 2.5.1 节“将脚本变成组件”中，将介绍 Unity 是如何将 C#脚本转换为组件的，但是先让我们更改一个变量的值。

如图 2-2 所示，由于 CurrentAge 在第 7 行被声明为变量，因此其存储的值可以在脚本内或在 Unity 的 Inspector 中被更改(因为是 public 变量)。更新后的值将

至此向后被代码中任何使用该变量的地方沿用。让我们通过行动进一步了解。

(1) 如果场景仍在运行，请通过单击 Play 按钮停止游戏。

(2) 在 Inspector 面板中将 Current Age 更改为 18，然后再次运行场景，并在 Console 面板中查看新输出，如图 2-5 所示。

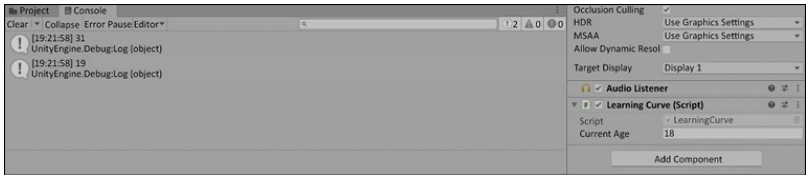

图 2-5　在 Unity 的控制台中查看附加在主摄像机上的 LearningCurve 脚本的调试信息

第一个输出仍然是 31，因为我们并没有对脚本做任何的修改。但是第二个输出现在是 19，因为我们在 Inspector 中更改了变量 CurrentAge 的值。

本节的目标不是要全面地介绍变量的语法，而是展示变量如何作为容器，如何可以一次创建并在其他位置多次被引用。既然知道了如何在 C#中创建变量并为其分配值，就意味着已准备好深入下一个重要的编程基本构建块：方法！

2.2　了解方法

只依赖变量自身，变量除了跟踪其分配的值之外并不能做太多的事情。尽管这也很重要，但是就创建有意义的应用程序而言，单凭变量并不足够。那么，如何在代码中创建动作和驱使行为呢？简单的回答就是要使用方法。

在了解什么是方法以及如何使用它们之前，我们应该清楚一些术语。在编程圈子里，特别是在 Unity 中，通常会看到“方法”和“函数”两种等价的称呼。

由于 C#是一种面向对象的语言(第 5 章中将涉及此部分内容)，本书其余部分将使用“方法”这一称呼，以遵循标准的 C#术语准则。

当在脚本参考或任何其他文档中遇到“函数”这一称呼时，请将其视为“方法”。

2.2.1　方法驱使行为

与变量类似，给编程方法下定义可以是乏味且冗长的，也可以是异常简短的，同样可以从以下三个层面去理解。

- 从概念上讲，方法是应用程序内部各项业务运作的方式。

- 从技术上讲，方法是包含可执行语句的代码块，当方法名被调用时，这些可执行语句便运行。方法还可以接受参数，参数仅在方法自身的作用域内生效。
- 从实践应用上讲，方法是每次被执行时都会运行的一组指令的容器。该容器可以将变量作为输入，且只能在该方法本身内部被引用。

综上所述，方法就如同程序的骨骼，它们将一切连接在一起，几乎所有内容都基于方法的结构构建起来。

可以在 Microsoft C# 文档 https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/methods 中找到关于方法的深入指南。

2.2.2　方法也是占位符

让我们举一个简单的将两个数字相加的例子来全面理解方法的概念。编写脚本，其本质上是在罗列一行行的代码，以便计算机按顺序执行它们。在第一次需要将两个数相加时，可以像下面的代码块所示将它们相加。

```
SomeNumber + AnotherNumber
```

但是之后我们发现这些数字在其他地方也需要做同样的加法，加在一起。

复制并粘贴相同的一行代码，会导致代码臃肿或产生称为“意大利面式”的代码，应尽力避免此情形的发生。可以创建一个具有专属命名的方法来专门执行此操作。

```
AddNumbers ()
{
    SomeNumber + AnotherNumber
}
```

这样，AddNumbers 方法便在内存中占有一席之地，就如同一个变量。但是，除了值之外，它还包含一个指令块。在脚本中的任何位置使用该方法的名称(称为调用方法)，会让已被存储的指令唾手可得，而不必再重复编写任何相同的代码。

```
AddNumbers()
```

如果发现自己一遍又一遍地编写相同的代码，可能意味着你错失了将重复的操作化简或凝练为通用方法的机会。

这种情况通常被程序员戏称为“意大利面式”代码，因为这样的代码会变得非常凌乱。程序员常提及一种称为“不要重复自己(DRY)”的原则来避免出现这种情况，建议时刻牢记此原则。

和之前一样，在通过伪代码学习了新的概念后，最好是将其付诸实践，我们在下一节中便会这么做来强化理解。

让我们再次打开 LearningCurve，看看方法是如何在 C#中运作的。就像变量的那个示例一样，请参照图 2-6 所示的代码一比一复制到脚本中。为了看起来更整洁，其中已删除了上个示例的代码，但是你当然也可以将其保留在脚本中以供参考。

(1) 在 Visual Studio 中打开 LearningCurve。

(2) 在第 8 行添加一个新的变量：

```
public int addedAge = 1;
```

(3) 在第 16 行添加一个新方法，将 currentAge 和 addedAge 相加，并打印出结果。

```
void ComputeAge()
{
    Debug.Log(currentAge + addedAge);
}
```

(4) 在 Start 内通过下面的代码调用新方法。

```
void Start()
{
    ComputeAge();
}
```

(5) 在 Unity 中运行脚本之前，再次检查你的代码是否如图 2-6 所示。

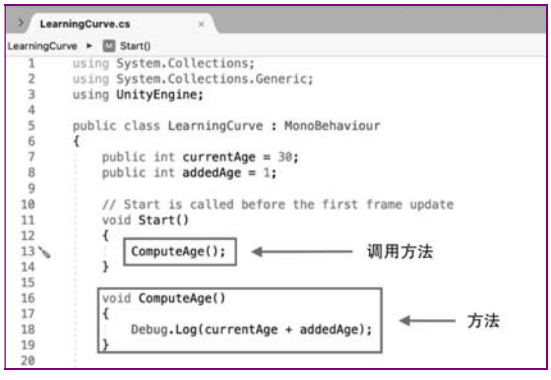

图 2-6　在 LearningCurve 脚本中创建 ComputeAge 方法

(6) 保存文件，然后返回 Unity，单击 Play 按钮并在控制台中查看新的输出。

我们在第 16 至 19 行中定义了第一个方法，并在第 13 行中对其进行了调用。现在，无论在何处调用 ComputeAge()方法，两个变量都将相加并且相加的结果会打印输出到控制台，即使它们的值发生了变化。请记住，我们之前在 Unity 的 Inspector 中将 CurrentAge 设置为 18，而 Inspector 中设置的值将总会优先覆盖 C#脚本中赋予的值(见图 2-7)。

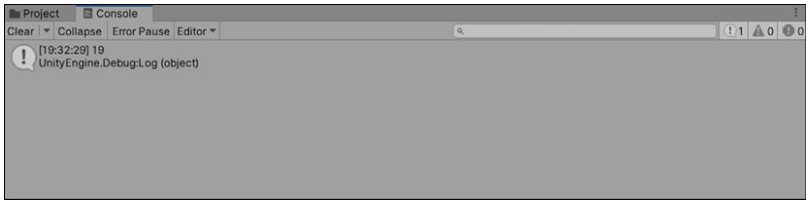

图 2-7　在 Inspector 中更改变量值并在控制台输出

可以继续在 Inspector 面板中尝试不同的变量值，并查看结果！有关上面编写代码的具体语法将在下一章讲解。

现在已掌握了有关方法的知识，下面介绍编程领域中最大的知识点——类！

2.3 类

虽然已经学习了变量如何存储信息以及方法如何执行动作，但是我们掌握的编程工具箱依然很有限。我们还需要掌握一种方法来创建某种超级容器，它拥有自身的变量和方法，且这些变量和方法可以通过该容器从其自身内部进行调用。这便是类。

- 从概念上讲，类将相关信息、动作和行为保存在单个容器中，它们之间还可以相互通信。
- 从技术上讲，类是一种数据结构。它可以包含变量、方法和其他程序信息，这些信息皆可在该类的对象被创建后被引用。
- 从实践上讲，类就像一个蓝图。它为任何参照该蓝图创建的对象(称为实例)设定了规则和规定。

可能你已发觉并意识到，不仅在 Unity 中，在现实世界中我们同样被各种各样的类包围着。接下来，我们将审视一个最常见的 Unity 类以及现实生活中的类是如何起作用的。

可以在 Microsoft C#文档 https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/types/classes 中找到关于类的更深入的指南。

2.3.1 一个常见的 Unity 类

在知道 C#中的类是什么样子之前，应该知道我们其实在本章中一直在使用一个类。默认情况下，在 Unity 中创建的每个脚本都是一个类，这一点可以从代码第 5 行的 class 关键字中看出来。

```
public class LearningCurve: MonoBehaviour
{

}
```

MonoBehaviour 代表该类可以附加到 Unity 场景中的 GameObject 上。两个大括号标记了类的边界——括号内部的任何代码都属于该类。

在 C#中，类可以独立存在，这种用法我们将在第 5 章中创建独立的类时遇到。

脚本和类的说法有时在 Unity 的一些资料中被交替使用。为了保持一致性，在本书中，我们将附加到 GameObjects 上的 C#文件称为脚本。然而，如果它们是独立的，我们则称其为类。

2.3.2 类就像蓝图

作为最后一个例子，让我们思考一下本地的邮局。邮局是一个独立且自包含的环境，意味着邮局具有一些属性，比如物理地址(可以视为一个变量)；具有执行业务的能力，比如可以发送信件(可以视为一个方法)。

这便使邮局潜在地成为可用于描述类的一个很好的例子，可以通过下面的伪代码块对其进行概述。

```
public class PostOffice
{
    // Variables
    public string address = "1234 Letter Opener Dr."

    // Methods
    DeliverMail() {}
    SendMail() {}
}
```

这里值得关注的是，当信息和行为遵循某一预定义的蓝图时，复杂的动作和类间的通信便成为可能。例如，如果有另一个类想要通过 PostOffice 类发送一封信，它并不需要思考如何才能触发该操作，可以很简单地从 PostOffice 类中调用 SendMail 方法来实现，如下所示。

```
PostOffice().SendMail()
```

另外，还可以用它查找邮局的地址，以便知道邮件是从哪里寄出的。

```
PostOffice().Address
```

也许你对上面英文单词间使用的英文句点(称为点表示法)存有疑惑，我们将在下一节介绍它，请先跟上。

2.3.3　类之间的通信

到目前为止，我们将类以及作为其拓展的 Unity 组件都描述为分离的、独立的个体。但实际上，它们之间是深深交织在一起的。如果类之间不存在交互或通信，创建任何有价值的应用程序软件都将极其困难。

在之前邮局的例子中，示例代码使用英文句点来引用类、变量和方法。如果将类视为信息的目录，那么点表示法(dot notation)便是用来索引的工具。

```
PostOffice.Address
```

可以通过点表示法访问类中的任何变量、方法或其他数据类型。它同样适用于嵌套的类或子类的信息，我们会在第 5 章涉及所有这些知识点。

点表示法同样也是驱动类与类之间通信的手段。每当一个类需要有关另一个类的信息或想要执行其中的方法时，都会通过点表示法来实现。

```
PostOffice.DeliverMail()
```

在某些文档中，点表示法有时也被称为“.”运算符(点运算符)。所以，如果在某些文档中看到它以这种方式被提及，不要忘记它。

如果尚未熟悉点表示法也不用担心，你一定会慢慢习惯它的。点表示法如同流淌在整个程序躯干中的血液，在被需要的地方承载着信息和上下文。

现在已对类有了稍多的了解，接下来谈谈在编程中使用最频繁的工具——注释！

2.4 使用注释

你可能已经注意到，LearningCurve 脚本中有一行奇特的文本(图 2-6 中的第 10 行)，这一行文本以两条斜线(正斜线/)开头，是创建脚本时默认生成的。

它们是代码注释。在 C#中，有多种方式创建注释，而 Visual Studio(及其他代码编辑应用程序)通常会通过内置的快捷方式使其变得更加容易。

一些专业人士不会将注释视为编程的基本构建块，但我却无法认同这个观点。用有意义的信息将代码进行恰当的注释，是编程新手最应培养的基本习惯之一。

2.4.1 单行注释

下面的单行注释与在 LearningCurve 脚本中的注释相似。

```
// This is a single-line comment
```

Visual Studio 不会将以两条斜线(之间无空格)开头的行视为代码，因此我们可以根据需要尽情使用它们为别人或为自己对代码进行诠释。

2.4.2 多行注释

正如其名，你大概也可以猜出单行注释仅适用于注释代码中的某一行。如果要多行注释，则需要在注释文本前后使用斜线和星号的组合(即用/*和*/分别作为注释的开始和结束字符)。

```
/* this is a
     multi-line comment */
```

还可以通过高亮显示代码块(拖动光标并选中代码块)并使用快捷方式来对该代码块进行注释或取消注释。该快捷方式在 MacOS 上为 Cmd + /，在 Windows 上为 Ctrl + K + C。

Visual Studio 还提供了一个方便的自动生成注释的功能。在任何代码行(变量、方法、类等)之前的行中连续键入三条斜线，将出现一个摘要注释块，如图 2-8 所示。

示例中的注释看着再好，也不如在自己的代码中做好注释。开始习惯加注释永远不会太早！

2.4.3　添加注释

打开 LearningCurve，并在 ComputeAge()方法上方添加三条斜线，见图 2-8。

```
31          /// <summary>
32          /// Time for action - adding comments
33          /// Computes a modified age integer
34          /// </summary>
35          void ComputeAge()
36          {
37              Debug.Log(CurrentAge + AddedAge);
38          }
39      }
40
```

图 2-8　为方法自动生成的三行注释

此时应该看到三行注释，其中包含由 Visual Studio 根据方法名称生成的方法的描述，夹在两个<summary>标签之间。当然，我们可以像在文本文档中一样更改注释文本或按 Enter 键添加新行，只要确保没有触及<summary>标签，否则，Visual Studio 将无法正确识别注释。

当想了解自己编写的某个方法时，这种详细注释就起到了作用。当某方法使用了三条斜线进行注释，在任何调用该方法的位置，若将鼠标悬停在该方法名称上，Visual Studio 会弹出显示该方法的注释摘要。见图 2-9。

```
11          /// Start is called before the first frame update
12          void Start()
13          {
14              ComputeAge();
15          }
16
17          // Up
18          void update()
19          {
```

void LearningCurve.ComputeAge()
Computes a modified age integer

图 2-9　Visual Studio 弹出信息框以显示注释摘要

至此，我们的基础编程工具箱已基本完备(至少是理论部分)。然而，我们仍需要了解本章所学内容如何应用于 Unity 游戏引擎中，这将在下一节中重点介绍！

2.5　整合基本构建块

在本章收尾之前，随着各基本构建块的揭示，是时候做一些针对 Unity 的整理工作了。具体来说，我们需要更多地了解 Unity 是如何处理附加到 GameObjects 上的 C#脚本的。

在本节的示例中，我们将继续使用 LearningCurve 脚本和 Main Camera 这个 GameObject。

2.5.1 将脚本变成组件

所有 GameObject 的组件都是脚本，无论是由我们还是由 Unity 编写的。唯一的区别是，对于由 Unity 提供的原生组件(例如 Transform)及其对应的脚本，用户通常不应该编辑它。

一旦将创建的脚本拖放到某个 GameObject 上，它便成为该游戏对象的一个新组件，这便是它会出现在 Inspector 面板中的原因。对于 Unity 而言，它同其他任何组件一样能“走”、会“说”，并可随时在该组件下更改其公共变量。尽管我们不应该编辑 Unity 提供的原生组件，但仍然可以访问它们的属性和方法，从而使它们成为更强大的开发工具。

当脚本成为组件时，Unity 还会自动地进行一些可读性调整。你可能已经注意到，当我们向 Main Camera 添加 LearningCurve 脚本时，如图 2-3 和图 2-5 所示，在 Inspector 中，Unity 将其显示为“Learning Curve”，同时 CurrentAge 将其显示为“Current Age”。

在 2.1.2 节“变量充当占位符”中，我们已经尝试在 Inspector 面板中更新变量值，现在有必要更详细地讲一下它的工作原理。调整某一属性的值可以分为以下三种情况：

- 在 Unity Editor 窗口的 Play 模式下(运行时状态)
- 在 Unity Editor 窗口的开发模式下(编辑器状态)
- 在 Visual Studio 代码编辑器中

在 Play 模式下所做的更改会立即实时生效，这非常适合对游戏进行测试和微调。但是，请务必注意，在停止游戏并返回到开发模式后，在 Play 模式下所做的任何更改都将丢失。若丢失在 Play 模式下的更改会令人非常沮丧，所以请务必留意游戏试玩测试时所处的模式。

若要复制你在 Play 模式中所做的任何更改，请执行以下步骤。

(1) 右击修改后的组件，选择 Copy Component，如图 2-10 所示。

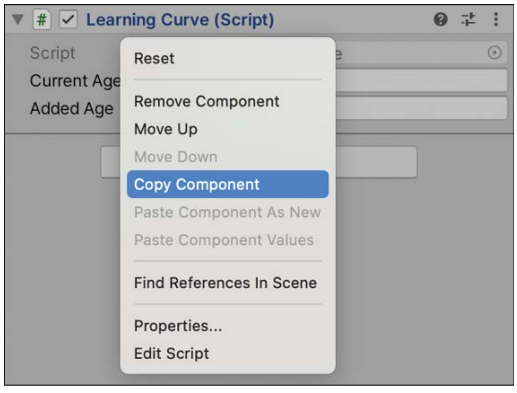

图 2-10　选择 Copy Component

(2) 退出 Play 模式，再次右击组件，这次选择 Paste Component Values，如图 2-11 所示。

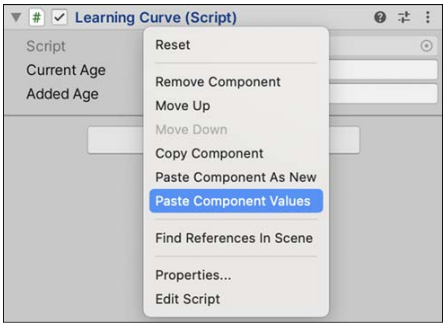

图 2-11　选择 Paste Component Values

在“开发”模式下，对变量所做的任何更改都将由 Unity 保存。这意味着，假设你退出 Unity 再重新启动它，更改仍将被保留。

在 Inspector 面板中对值所做的任何更改都不会影响脚本中的内容，但是在“运行”模式下，这些更改将覆盖在脚本中分配的值。

在 Play 模式下所做的任何更改都将在停止 Play 模式时自动重置。如果需要撤销在 Inspector 面板中所做的任何更改，可以将脚本重置为其默认值(有时称为初始值)。单击任何组件右侧的三个垂直点的图标，然后选择 Reset(重置)，如图 2-12 所示。

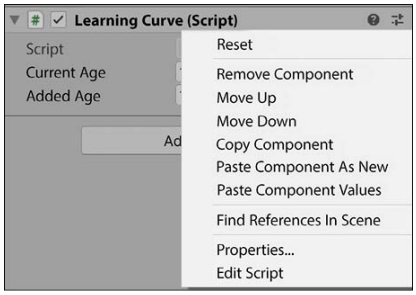

图 2-12　Inspector 中的脚本重置选项

如果变量失控了，我们总可以硬重置(hard reset)它们，这会让我们很安心。

2.5.2　来自 MonoBehaviour 的助力

由于 Unity 的 C#脚本都是类，那么 Unity 又是如何知道该把哪些脚本视为组件，而哪些不视为组件呢？简单来说，LearningCurve(以及在 Unity 中创建的任何脚本)继承自名称为 MonoBehaviour 的类(一个由 Unity 提供的默认类)。这便告诉了 Unity，可以将这个 C#类转换为组件。然而，并非所有脚本都必须继承自 MonoBehaviour 类——只有那些想要添加到 Unity 场景中的游戏对象上的脚本才有必要继承。

在我们编程旅程的当前阶段，介绍有关类的继承的知识点还略显有些超前。我们暂可以认为 MonoBehaviour 类将其一部分变量和方法借给 LearningCurve 类使用。在第 5 章中，将通过实践详细介绍类的继承，届时也将介绍如何编写一个不继承自 MonoBehaviour 的类。

常使用的 Start()和 Update()方法也都属于 MonoBehaviour 类，任何附加到 GameObject 的脚本中只要存在以上两个方法或其中任一个，Unity 都会自动运行它们。在场景开始运行时，Start()方法会执行一次；而 Update()方法会每帧都执行一次(因此执行频率取决于计算机的帧率)。

既然我们对 Unity 文档的熟悉程度有了很大的提高，那么我们来尝试一个简短的可选挑战！

英雄的试炼——脚本 API 中的 MonoBehaviour

现在是时候尝试独自使用 Unity 文档了，还有什么比查找一些常见的 MonoBehaviour 方法更好的方式呢。

- 尝试在脚本 API 中搜索 Start()和 Update()方法，来更好地了解它们在 Unity 中的作用。
- 想挑战的话，还可以更进一步，查看手册中的 MonoBehaviour 类，了解更详细的解释和说明。

2.6　本章小结

了解基本概念(例如变量、方法和类)的理论将为我们打下坚实的基础。请记住，这些基本构建块在现实世界中都可以找到非常真实的参照物。变量保管值，如同信箱保管信件；方法如同配方，存储指令说明，以便我们遵循这些指令以获得预期的结果；类就像蓝图，与真实的蓝图作用一样。

如果你希望建造的房屋结实坚固，那么就不能没有一个经过深思熟虑的设计方案来遵循。

本书接下来将带你从零开始深入地学习 C#语法，在第 3 章中将详细介绍如何创建变量、管理数值类型以及使用一些或简单或复杂的方法。

2.7　小测验——C#的基本构建块

(1) 变量的主要目的是什么？
(2) 方法在脚本中起什么作用？
(3) 脚本是如何变成组件的？
(4) 点表示法的目的是什么？

不要忘记将你的回答与附录中提供的小测验答案做对照，以检验你的学习成果。