.NET 开发经典名著

ASP.NET Core 3 全栈 Web 开发(第 3 版)

使用.NET Core 3.1 和 Angular 9

[意] 瓦莱西奥·德·桑克蒂斯(Valerio De Sanctis) 著
赵利通 崔战友 译

清华大学出版社
北 京

北京市版权局著作权合同登记号　图字：01-2020-6983

图书在版编目(CIP)数据

ASP.NET Core 3全栈Web开发：使用.NET Core 3.1和 Angular 9：第3版 / (意)瓦莱里奥 • 德 • 桑克蒂斯(Valerio De Sanctis)著；赵利通，崔战友译. —北京：清华大学出版社，2021.1

(.NET开发经典名著)

书名原文：ASP.NET Core 3 and Angular 9: Full Stack Web Development with .NET Core 3.1 and Angular 9, Third Edition

ISBN 978-7-302-57218-3

Ⅰ. ①A … Ⅱ. ①瓦… ②赵… ③崔… Ⅲ. 网页制作工具－程序设计 Ⅳ. ①TP393.092.2

中国版本图书馆 CIP 数据核字(2020)第 257426 号

责任编辑：王　军　韩宏志
装帧设计：孔祥峰
责任校对：成凤进
责任印制：沈　露

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
社 总 机：010-62770175　　**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：大厂回族自治县彩虹印刷有限公司
经　　销：全国新华书店
开　　本：170mm×240mm　　**印　　张**：28.25　　**字　　数**：776 千字
版　　次：2021 年 1 月第 1 版　　**印　　次**：2021 年 1 月第 1 次印刷
定　　价：118.00 元

产品编号：088761-01

作 者 简 介

Valerio De Sanctis 是一名掌握丰富技能的 IT 专业人员，在使用 ASP.NET、PHP 和 Java 进行编程、Web 开发和项目管理方面具有超过 15 年的经验。他在多家金融和保险公司担任过高级职务，近来在一家业界领先的售后服务和 IT 服务公司担任首席技术官、首席安全官和首席运营官，这家公司为多个顶尖的人寿和非人寿保险集团提供服务。

在职业生涯中，Valerio 帮助许多私企实现和维护基于.NET 的解决方案，与许多 IT 行业的专家携手工作，并领导过多个前端、后端和 UX 开发团队。他为多个知名的客户和合作伙伴设计了许多企业级 Web 应用程序项目的架构，并监管这些项目的开发。这些客户包括 London Stock Exchange Group、Zurich Insurance Group、Allianz、Generali、Harmonie Mutuelle、Honda Motor、FCA Group、Luxottica、ANSA、Saipem、ENI、Enel、Terna、Banzai Media、Virgilio.it、Repubblica.it 和 Corriere.it。

他是 Stack Exchange 网络上的活跃成员，在 StackOverflow、ServerFault 和 SuperUser 社区中提供关于.NET、JavaScript、HTML5 和 Web 主题的建议和提示。他的大部分项目和代码示例在 GitHub、BitBucket、NPM、CocoaPods、JQuery Plugin Registry 和 WordPress Plugin Repository 中以开源许可提供。他还是 Microsoft 开发技术 MVP，这是一个年度颁发的奖项，用来表彰全球范围内积极与用户和 Microsoft 分享高质量的实用专家技能的卓越技术社区领袖。

自 2014 年以来，他在 www.ryadel.com 上运营一个面向 IT 并关注 Web 的博客，提供业界新闻、评审、代码示例和指导，旨在帮助全球的开发人员和技术爱好者。他撰写了多本关于 Web 开发的图书，许多都在 Amazon 上成为畅销图书，在全球范围内销售了数万本。

审校者简介

Anand Narayanaswamy 是印度特里凡得琅的一名自由撰稿人和审校者，在业界领先的纸质杂志和网上技术门户中发表过文章。他在 2002 年和 2011 年获得了 Microsoft MVP 称号，目前是一名 Windows Insider MVP，也是备受推崇的 ASPInsider 小组的成员。

Anand 为几家出版商担任技术编辑和审校者，并为 Packt Publishing 撰写了 *Community Server Quickly* 一书。他还为 Digit Magazine 和 Manorama Year Book 撰写文章。

Anand 的致谢：我感谢上帝赋予我每天工作的能力。还要感谢 Amrita Venugopal、Kinjal Bari 和 Manthan Patel 给予我的支持和耐心。Packt 的编辑们在项目一开始就认真对待审校者，这很令人敬佩。我也很感谢我的父亲、母亲和哥哥一直以来给我的支持和鼓励。

Santosh Yadav 来自印度浦那，拥有计算机学士学位。他有超过 11 年的开发经验，并使用过多种技术，包括.NET、Node.js 和 Angular。他是 Angular 和 Web 技术的 Google 开发技术专家。他为 Netlify 创建了 ng deploy 库，并且是 *Angular In Depth* 的作者。他还是 Pune Tech Meetup 的演讲人和组织者，并为包括 Angular 和 NgRx 在内的多个项目做出了贡献。

前　言

ASP.NET Core 是一个免费、开源的模块化 Web 框架，由 Microsoft 开发，运行在完整的.NET Framework(Windows)或.NET Core(跨平台)上。它专门为构建高效的 HTTP 服务设计，使这些服务可被各种类型的客户端访问和使用，包括 Web 浏览器、移动设备、智能电视、基于 Web 的智能家居工具等。

Angular 是 AngularJS 的后继产品。AngularJS 是世界知名的开发框架，其设计目标为向开发人员提供一个工具箱，使他们能够构建出响应式、跨平台、基于 Web 的应用，并且能够针对桌面和移动设备优化这些应用。它采用了结构丰富的模板方法，以及自然的、易写易读的语法。

从技术角度看，这两种框架没有太多共同之处：ASP.NET Core 主要关注 Web 开发栈的服务器端部分，而 Angular 则用于处理 Web 应用程序的所有客户端部分，如用户界面(User Interface，UI)和用户体验(User Experience，UX)。但是，这两种框架之所以产生，是因为其创建者具有一个共同的构想：HTTP 协议不限于提供 Web 页面；可以把它用作一个平台来构建基于 Web 的 API，以有效地发送和接收数据。这种设想在万维网的前 20 年间逐渐发展，现在已经成为不可否认的、广泛认同的表述，也是几乎每种现代 Web 开发方法的基础。

对于这种视角的转变，存在许多很好的理由，其中最重要的理由与 HTTP 协议的根本特征有关：使用起来很简单，但足够灵活，能够适应万维网不断变化的环境的大部分开发需求。如今，HTTP 协议的适用范围也变得很广：我们能够想到的几乎每种平台都有一个 HTTP 库，因此 HTTP 服务能够用于各种客户端，包括桌面和移动浏览器、IoT 设备、桌面应用程序、视频游戏等。

本书的主要目的是在一个开发栈中，把最新版本的 ASP.NET Core 和 Angular 结合起来，以演示如何使用它们来创建高性能的、能够被任何客户端无缝使用的 Web 应用程序和服务。

本书面向的读者

本书面向有经验的 ASP.NET 开发人员，你应该已经知道 ASP.NET Core 和 Angular，想学习这两种框架的更多知识，并理解如何结合它们来创建适合生产环境的单页面应用程序(SPA)或渐进式 Web 应用程序(PWA)。但是，本书提供了完整的代码示例，并逐个步骤讲解了实现过程，即使是新手或者刚入门的开发人员，理解本书也不会有太大难度。

本书内容

第 1 章介绍本书将使用的两个框架的一些基本概念，以及可以创建的各种类型的 Web 应用程序，如 SPA、PWA、原生 Web 应用程序等。

第 2 章详细介绍 Visual Studio 2019 提供的.NET Core and Angular 模板的各种后端和前端元素，并且从高层面上解释了如何在典型的 HTTP 请求-响应周期中结合使用它们。

第 3 章全面介绍如何构建一个示例 ASP.NET Core 和 Angular 应用，该应用通过使用基于 Bootstrap 的 Angular 客户端来查询健康检查中间件，为最终用户提供诊断信息。

第 4 章介绍 Entity Framework Core 及其作为对象关系映射(Object-Relational Mapping，ORM)框架的能力，从 SQL 数据库部署(基于云和/或本地实例)一直讲解到数据模型设计，还包括在后端控制器中读写数据的各种技术。

第 5 章介绍如何使用 ASP.NET Core 后端 Web API 来公开 Entity Framework Core 数据，使用 Angular 消费该数据，然后使用前端 UI 向最终用户展示数据。

第 6 章详细介绍如何在后端 Web API 中实现 HTTP PUT 和 POST 方法，以使用 Angular 执行插入和更新操作，还介绍服务器端和客户端数据验证。

第 7 章探讨一些有用的调整和改进，以增强应用程序的源代码，并深入分析 Angular 的数据服务，以理解为什么以及如何使用它们。

第 8 章介绍如何充分利用 Visual Studio 提供的各种调试工具，调试一个典型 Web 应用程序的后端和前端栈。

第 9 章详细介绍测试驱动开发(Test-Driven Development，TDD)和行为驱动开发(Behavior-Driven Development，BDD)，并展示了如何使用 xUnit、Jasmine 和 Karma 来定义、实现和执行后端和前端单元测试。

第 10 章从高层面上介绍身份验证和授权的概念，并展示了如何使用一些技术和方法来恰当地实现专有的或第三方的用户身份系统。本章给出一个基于 ASP.NET Identity 和 IdentityServer4 的、可以工作的 ASP.NET Core 和 Angular 身份验证机制的实际例子。

第 11 章详细介绍如何使用服务工作线程、清单文件和离线缓存功能，将一个现有的 SPA 转换成为一个 PWA。

第 12 章介绍如何部署前面章节中创建的 ASP.NET Core 和 Angular 应用，以及如何使用 Windows Server 2019 或 Linux CentOS 虚拟机把它们发布到云环境中。

最大限度利用本书

下面列出撰写本书及测试源代码时使用的软件包及相关版本号：

- Visual Studio 2019 社区版 16.4.3
- Microsoft .NET Core SDK 3.1.1
- TypeScript 3.7.5
- NuGet Package Manager 5.1.0
- Node.js 13.7.0[强烈建议使用 Node Version Manager(NVM)，来安装 Node.js]
- Angular 9.0.0 最终版

对于在 Windows 上部署：

- ASP.NET Core 3.1 Runtime for Windows
- .NET Core 3.1 CLR for Windows

对于在 Linux 上部署：

- ASP.NET Core 3.1 Runtime for Linux(YUM 包管理器)
- .NET Core 3.1 CLR for Linux(YUM 包管理器)
- Nginx HTTP Server(YUM 包管理器)

提示 如果你使用的是 Windows 平台，强烈建议使用 NVM for Windows 来安装 Node.js，这是适用于 Windows 系统的一个非常出色的 Node.js 版本管理器。从下面的 URL 可以下载 NVM for Windows。

```
https://github.com/coreybutler/nvm-windows/releases
```

强烈建议你使用与本书相同的版本。如果你选择使用一个不同的版本，也没有问题，但你可能需要对源代码做一些修改和调整。

下载示例代码文件

可以在 www.tup.com.cn/downpage，输入本书中文书名或 ISBN，下载本书的示例代码文件以及参考资料。

也可扫描封底二维码下载示例代码文件以及参考资料。

本书采用的约定

本书采用了一些排版约定。

代码块的格式如下所示：

```
<mat-form-field [hidden]="!cities">
   <input matInput (keyup)="loadData($event.target.value)"
        placeholder="Filter by name (or part of it)...">
</mat-form-field>
```

当我们希望你注意代码块中的特定部分时，将用粗体显示相关的代码行：

```
import { FormGroup, FormControl } from '@angular/forms';

class ModelFormComponent implements OnInit {
   form: FormGroup;

   ngOnInit() {
     this.form = new FormGroup({
        title: new FormControl()
     });
   }
}
```

命令行输入或输出的格式如下所示：

```
> dotnet new angular -o HealthCheck
```

注意　警告或重要的注意事项将采用这种格式。

提示　提示和技巧将采用这种格式。

目　录

第1章
准 备 工 作

在踏上学习 ASP.NET 和 Angular 的征程之前，本章先做一些准备工作，从理论的高度介绍它们最主要的特性，然后讨论一些实用主题。具体来说，在本章前半部分，我们将简单回顾 ASP.NET Core 和 Angular 框架的近期发展，而在后半部分，则将学习如何配置本地开发环境，从而能够组装、生成和测试一个 Web 应用程序的样板性示例。

到本章结束时，你将了解在过去几年间，ASP.NET Core 和 Angular 为了帮助 Web 开发所做的一些工作，并知道如何设置一个 ASP.NET 和 Angular Web 应用程序。

本章将介绍的主题

- ASP.NET Core 变革：简单回顾 ASP.NET Core 和 Angular 的近期发展。
- 全栈方法：能够学习如何设计、组装和交付完整产品的重要性。
- 单页面应用程序(Single-Page Application，SPA)、原生 Web 应用程序(Native Web Application，NWA)和渐进式 Web 应用程序(Progressive Web Applications，PWA)：各种应用程序的关键特性、它们之间最重要的区别以及 ASP.NET Core 和 Angular 适用各种应用程序的程度。
- 示例 SPA 项目：本书将完成的一个项目。
- 准备工作环境：如何设置工作环境来完成我们的第一个目标，实现一个简单的 Hello World 样板示例，后续章节将进一步扩展这个示例。

1.1 技术需求

撰写本书及测试源代码时使用的软件包(及相关版本号)如下所示：

- Visual Studio 2019 社区版 16.4.3
- Microsoft .NET Core SDK 3.1.1
- TypeScript 3.7.5
- NuGet Package Manager 5.1.0
- Node.js 13.7.0 (强烈建议使用 Node Version Manager，也称为 NVM，来安装 Node.js)
- Angular 9.0.0 最终版

提示 如果你在 Windows 平台上进行开发，那么强烈建议使用 NVM for Windows 来安装 Node.js。NVM for Windows 是适用 Windows 系统的一个整洁的 Node.js 版本管理器。从下面的 URL 可以下载 NVM for Windows:

```
https://github.com/coreybutler/nvm-windows/releases
```

强烈建议使用与本书相同的版本。不过，如果你选择使用另一个版本，也没有问题，只不过可能

需要对源代码做一些修改和调整。

1.2 两个框架，一个目标

从一个功能完善的 Web 应用程序的视角看，可以说 ASP.NET Core 框架提供的 Web API 接口是服务器端处理程序的一个集合，服务器使用它们向定义好的请求-响应消息系统公开众多钩子/端点。这通常是使用结构化标记语言(XML)、语言无关的数据格式(JSON)或 API 查询语言(GraphQL)表达的，并通过在一个公开可用的 Web 服务器(如 IIS、Node.js、Apache、Nginx 等)上使用 HTTP 和/或 HTTPS 公开应用程序编程接口(Application Programming Interface，API)来实现。

与之类似，Angular 是一种现代的、功能丰富的客户端框架，通过将 HTML Web 页面的输入和/或输出部分绑定到一个灵活的、可重用的、易于测试的模型，推动了 HTML 和 ECMAScript 的高级特性以及现代浏览器能力的发展。

我们是否能够把 ASP.NET Core 的后端优势和 Angular 的前端能力结合起来，创建出功能丰富的、非常灵活的现代 Web 应用程序？

答案是肯定的。在后续章节中，我们将通过分析一个代码优美的、设计精良的 Web 产品的各个基础方面，以及如何使用 ASP.NET Core 和/或 Angular 的最新版本来处理这些方面，展示这种 Web 应用程序的开发过程。但是，在那之前，花一点时间回顾过去 3 年间，这两种框架经历了怎样的发展历程，是很有帮助的。尽管面对着越来越多的竞争对手，这两种框架仍然得到人们的青睐，这是有其原因的，理解这些原因对我们很有用。

1.2.1 ASP.NET Core 的变革

要总结过去 4 年间，ASP.NET 发生了什么，并不是一个轻松的任务。简单来说，我们无疑见证了.NET Framework 自诞生以来经历的最重要的一系列变化。这是一场变革，几乎在各个方面改变了 Microsoft 处理软件开发的方法。这家以专有软件、许可和专利闻名的公司变为推动全世界开源开发的一股力量。要理解这些年间发生了什么，从这个缓慢但稳定的发展过程中截取几个关键节点很有帮助。

依我看来，第一个关键节点发生在 2014 年 4 月 3 日的 Microsoft Build Conference 大会上。这是每年举办一次的开发者大会，2014 年度的举办地点为圣弗朗西斯科的 Moscone Center (West)。在这次大会上，Anders Hejlsberg(Delphi 之父，也是 C#的主架构师)发表了令人印象深刻的主题发言，并在此过程中将.NET Compiler Platform 的第一个版本(称为 Roslyn)公开发布为一个开源项目。也是在这次大会上，Microsoft Cloud and AI Group 的执行副总裁 Scott Guthrie 宣布正式启动.NET Foundation，这是一个非营利组织，旨在推进.NET 生态系统中的开源软件开发及协作性工作。

从那个时候起，.NET 开发团队在 GitHub 平台上稳定发布了许多 Microsoft 开源项目，包括 Entity Framework Core(2014 年 5 月)、TypeScript(2014 年 10 月)、.NET Core(2014 年 10 月)、CoreFX(2014 年 11 月)、CoreCLR 和 RyuJIT(2015 年 1 月)、MSBuild(2015 年 3 月)、.NET Core CLI(2015 年 10 月)、Visual Studio Code(2015 年 11 月)和.NET Standard(2016 年 9 月)等。

1. ASP.NET Core 1.x

为开源开发付出的这些努力，最重要的成果是在 2016 年第三季度公开发布了 ASP.NET Core 1.0。ASP.NET Framework 自 2002 年 1 月问世后历经发展，但一直到 4.6.2 版本(2016 年 8 月)，其核心架构并没有重大变化。ASP.NET Core 1.0 彻底重新实现了 ASP.NET Framework。这个全新的框架将之前的

Web 应用程序技术(MVC、Web API 和 Web 页面)合并到一个编程模块中，并称其为 MVC6。新框架引入了一个功能完善的跨平台组件(名为.NET Core)，与前面提到的开源工具一起发布，这些工具包括一个编译器平台(Roslyn)、一个跨平台运行时(CoreCLR)和一个经过改进的 x64 即时编译器(RyuJIT)。

注意 一些读者可能想了解 ASP.NET 5 和 Web API 2 的情况，因为在 2016 年中段之前，它们曾是开发人员耳熟能详的名称。

ASP.NET 5 是 ASP.NET Core 原本的名称，不过开发者们后来决定将其重新命名，以强调这是对 ASP.NET Core 彻底重写这个事实。重命名的原因，以及 Microsoft 对新产品的愿景，在 Scott Hanselman 的一篇博客文章中有详细说明。这篇文章写于 2016 年 1 月 16 日，对新产品的变动做了预期性说明：

```
http://www.hanselman.com/blog/ASPNET5IsDeadIntroducingASPNETCore10AndNETCore10.aspx
```

简单介绍一下 Scott Hanselman。他自 2007 年开始，就担任.NET/ASP.NET/IIS/Azure 和 Visual Studio 的外展和社区经理。关于 ASP.NET 的这种视角转换的更多信息，可以参考 Jeffrey T. Fritz 撰写的下面这篇文章，他是 Microsoft 的项目经理和 NuGet 团队主管：

```
https://blogs.msdn.microsoft.com/webdev/2016/02/01/an-update-on-asp-net-core-and-net-core/
```

注意 Web API 2 是专门用于构建 HTTP 服务的一个框架，这些 HTTP 服务返回纯粹的 JSON 或 XML 数据，而不是 Web 页面。它一开始作为 MVC 平台之外的另一种方案诞生，但现在已经与 MVC 合并起来，形成了一个新的、通用的 Web 应用程序框架，称为 MVC6，作为 ASP.NET Core 的一个单独模块发布。

ASP.NET Core 1.0 发布后不久，ASP.NET Core 1.1 发布了(2016 年第 4 季度)，并带来了一些新功能，性能上也得到了增强，还解决了 1.0 版本的许多 bug 和兼容性问题。这些新功能包括将中间件配置为过滤器(这是通过把它们添加到 MVC 管道而不是 HTTP 请求管道实现的)的能力，一个内置的、与宿主无关的 URL 重写模块(通过专门的 NuGet 包 Microsoft.AspNetCore.Rewrite 提供)，将视图组件作为标签助手的能力，在运行时而不是按需查看编译，以及.NET 原生的压缩和缓存中间件模块等。

注意 关于 ASP.NET Core 1.1 中的全部新功能、改进和 bug 修复的详细清单，可访问下面的链接。

版本说明：https://github.com/aspnet/AspNetCore/releases/1.1.0。

提交列表：https://github.com/dotnet/core/blob/master/release-notes/1.1/1.1-commits.md。

2. ASP.NET Core 2.x

ASP.NET Core 2.0 是又一次跃进。2017 年第二季度发布了该版本的预览版，第三季度发布了最终版。新版本对接口做了大量重要的改进，主要是为了标准化.NET Framework、.NET Core 和.NET Standard 之间共享的 API，使它们能够与.NET Framework 向后兼容。这些工作使得把现有的.NET Framework 项目迁移到.NET Core 和/或.NET Standard 变得比之前容易许多，从而让许多传统开发人员有机会体验和适应新的范式，但同时不必丢弃自己已经掌握的技术。

同样，主版本发布后不久，又发布了一个改进后的版本 ASP.NET Core 2.1。该版本于 2018 年 5 月 30 日正式发布，引入了一系列安全和性能改进，以及许多新功能：SignalR(一个库，简化了在.NET Core 应用中添加实时 Web 功能的工作)；Razor 类库；对 Razor SDK 做了重大改进，允许开发人员把视图和页面构建为可重用的类库，和/或能够作为 NuGet 包发布的库项目；Identity UI 库和基架，用来

为任何应用添加身份，以及定制身份来满足自己的需要，默认启用了对 HTTPS 的支持；使用面向隐私的 API 和模板，内置了对通用数据保护条例(General Data Protection Regulation，GDPR)支持，使用户能够控制自己的私人数据和同意是否使用 cookie；针对 Angular 和 ReactJS 客户端框架更新了 SPA 模板等。

注意 关于 ASP.NET Core 2.1 中的全部新功能、改进和 bug 修复的详细清单，可访问下面的链接。

版本说明：https://docs.microsoft.com/en-US/aspnet/core/release-notes/aspnetcore-2.1。

提交列表：https://github.com/dotnet/core/blob/master/release-notes/2.1/2.1.0-commit.md。

你没看错，我们刚才提到了 Angular。事实上，从一开始发布，ASP.NET Core 就被设计为与流行的客户端框架(如 ReactJS 和 Angular)无缝集成。正因为这个原因，本书这样的图书才会出现。ASP.NET Core 2.1 引入的重大区别是更新了默认的 Angular 和 ReactJS 模板，使它们使用相应框架的标准项目结构和生成系统(Angular CLI 和 NPX 的 create-react-app 命令)，而不是依赖任务执行器(如 Grunt 或 Gulp)、模块生成器(如 webpack)或者工具链(如 Babel)，这些工具在过去被广泛使用，不过安装和配置起来十分困难。

注意 能够不再依赖这些工具是一项重大成就，对 2017 年之后改变.NET Core 的用法和提高其在开发社区中的采用率起到了决定性作用。如果看看本书的前两版(2016 年年中出版的 ASP.NET Core and Angular 2 和 2017 年年底出版的 ASP.NET Core 2 and Angular 5)，并比较它们的第 1 章和本书的第 1 章，就能够注意到必须手动使用 Gulp、Grunt 或 webpack 与借助框架集成的工具的区别。复杂程度得以大大降低，使得任何开发人员都从中获益，尤其是不熟悉上述工具的开发人员。

2.1 版本发布 6 个月后，.NET Foundation 又对框架做了一次改进，在 2018 年 12 月 4 日发布了 ASP.NET Core 2.2，在这个版本中修复了一些问题，并添加了一些新功能。例如改进了端点路由系统来更好地分发请求，更新了模板来支持 Bootstrap 4 和 Angular 6，新增了一个健康检查服务来监控部署环境及其底层基础设施的状态，包含容器编排系统(如 Kubernetes)，在 Kestrel 中内置 HTTP/2 支持，使用一个新的 SignalR Java 客户端来方便在 Android 应用程序中使用 SignalR 等。

注意 关于 ASP.NET Core 2.2 中的全部新功能、改进和 bug 修复的详细清单，可访问下面的链接。

版本说明：https://docs.microsoft.com/en-US/aspnet/core/release-notes/aspnetcore-2.2。

提交列表：https://github.com/dotnet/core/blob/master/release-notes/2.2/2.2.0/2.2.0-commits.md。

3. ASP.NET Core 3.x

ASP.NET Core 3 发布于 2019 年 9 月，对性能和安全做了进一步的改进，并提供了更多新功能。例如支持 Windows 桌面应用程序(仅适用于 Windows)，并为导入 Windows Form 和 Windows Presentation Foundation(WPF)提供了高级能力；支持 C# 8；通过一组新的内置 API 来访问.NET Platform-Dependent Intrinsic，从而在某些场景中显著提高性能；通过使用 dotnet publish 命令，并在项目配置文件中使用<PublishSingleFile> XML 元素，或者通过使用/p:PublishSingleFile 命令行参数，支持单文件可执行文件；新增了内置的 JSON 支持，其特点是高性能、低内存，比 JSON.NET 第三方库(在大部分 ASP.NET Web 项目中已成为事实上的标准)可能快两三倍；在 Linux 上支持 TLS 1.3 和

OpenSSL 1.1.1；在 System.Security.Cryptography 命名空间中做了一些安全改进，包括支持 AES-GCM 和 AES-CCM 加密算法等。

对于提高框架在容器环境中的性能和可靠性，也做了大量工作。ASP.NET Core 开发团队投入大量精力，改善了.NET Core 3.0 中的.NET Core Docker 体验。具体来说，这是.NET Core 第一次对运行时做了重大修改，从而使 CoreCLR 更加高效，在默认情况下更好地遵守 Docker 的资源限制(如内存和 CPU 限制)，并提供了更多调整配置的能力。在各种改进中，有必要提到在默认情况下，内存和 GC 堆使用得到了改善。PowerShell Core 也得到了改进，这是著名的自动化和配置工具 PowerShell 的跨平台版本，现在随着.NET Core SDK Docker 容器镜像一起交付。

.NET Core Framework 3 还引入了 Blazor，这是一个免费、开源的 Web 框架，允许开发人员使用 C#和 HTML 创建 Web 应用。

最后需要提到，新的.NET Core SDK 比之前的版本小得多，这主要归功于开发团队在最终版本中删除了多个 NuGet 包中不必要的工件，这些包用于组装以前的 SDK(包括 ASP.NET Core 2.2)，但其中不必要的工件浪费了大量空间。对于 Linux 和 macOS 版本来说，SDK 明显变小了许多，但在 Windows 上，这种改进则没那么明显，因为 SDK 中还包含了新的 WPF 和 Windows Form 库。

注意 关于 ASP.NET Core 3.0 中的全部新功能、改进和 bug 修复的详细清单，可访问下面的链接。

版本说明：https://docs.microsoft.com/en-us/dotnet/core/whats-new/dotnet-core-3-0。

ASP.NET Core 3.0 发布页面：https://github.com/dotnet/core/tree/master/release-notes/3.0。

在撰写本书时，最新的稳定版本是 ASP.NET Core 3.1，发布于 2019 年 12 月 3 日。这个版本中的变化主要集中在 Windows 桌面开发，删除了许多 Windows Form 遗留控件(DataGrid、ToolBar、ContextMenu、Menu、MainMenu 和 MenuItem)，并添加了对创建 C++/CLI 组件的支持(但仅限 Windows 平台)。

大部分 ASP.NET Core 更新修复了 Blazor 的一些问题，如在 Blazor 应用中阻止事件的默认动作或者阻止事件传播，对 Razor 组件提供分部(partial)类支持，更多的标签助手组件功能等。但是，与其他.1 版本一样，NET Core 3.1 的主要目标是优化和改进上一个版本中已经交付的功能，它修复了 150 多个性能和稳定性问题。

注意 关于 ASP.NET Core 3.1 中的全部新功能、改进和 bug 修复的详细清单，可访问下面的链接。

版本说明：https://docs.microsoft.com/en-us/dotnet/core/whats-new/dotnet-core-3-1。

对 ASP.NET Core 近期发展状况的介绍到此结束。下一节将关注 Angular 生态系统，它也经历了相当类似的发展过程。

1.2.2 Angular 有哪些新变化?

如果说跟随 Microsoft 和.NET Foundation 近年来迈出的步伐并不容易，当我们把目光转移到客户端 Web 框架 Angular 时，情况并不会好转。要理解 Angular 的发展过程，需要把目光投向 10 年前。当时，jQuery 和 MooTools 等 JavaScript 库统治着客户端的世界，最早的客户端框架(如 Dojo、Backbone.js 和 Knockout.js)仍在努力赢得认知，以期得到广泛采用，而 React 和 Vue.js 甚至还没有出现)。

提示 实际上，jQuery 在很大程度上仍然占据统治地位，至少 Libscore (http://libscore.com/#libs) 和 w3Techs (https://w3techs.com/technologies/overview/javascript_library/all)认为如此。另一方面，虽然 74.1%的网站仍在使用 jQuery，但相比十年前，选择它的 Web 开发人员已经少了很多。

1. GetAngular

AngularJS 的故事开始于 2009 年。当时，Miško Hevery(现在担任 Google 的高级计算机科学家和敏捷教练)和 Adam Abrons(现在担任 Grand Rounds 的工程主管)在业余时间开发了一个项目，这是一个端到端(end-to-end，E2E)Web 开发工具，提供一个在线 JSON 存储服务，以及一个客户端库，用于构建依赖于该库的 Web 应用程序。为了发布这个项目，他们使用了 GetAngular.com 这个主机名。

在这段时期，Hevery 已经就职于 Google，他与另外两名开发人员一起被分配到 Google Feedback 项目。这几个人在 6 个月的时间中写了 17 000 多行代码，代码在膨胀，测试问题变得严重起来，这让他们非常沮丧。考虑到这种情况，Hevery 向经理请求使用 GetAngular(前面提到的业余项目)来重写应用，并打赌说他自己能在两周内完成重写。经理接受了这个赌约，但不久后，Hevery 输掉了，因为他用了 3 周而不是之前说的两周。但是，新的应用程序只有 1500 行代码，而不是原来的 17 000 行。这足以引起 Google 对新框架的兴趣，不久后，他们把这个新框架命名为 AngularJS。

提示 关于完整的故事，可以观看 Miško Hevery 在 ng-conf 2014 上发表的主题演讲: https://www.youtube.com/watch?v=r1A1VR0ibIQ。

2. AngularJS

AngularJS 的第一个稳定版本(0.9.0 版本，也称为“龙息”)于 2010 年 10 月在 GitHub 上使用 MIT 许可发布；当 AngularJS 1.0.0(也称为“时空统治”)于 2012 年 6 月发布时，这个框架已经在全世界的 Web 开发社区中赢得了很多人的青睐。

这种异乎寻常的成功，很难用几个词总结出来，但我还是会尝试介绍它的一些关键卖点。

- 依赖注入：AngularJS 是第一个实现了依赖注入的客户端框架。这无疑是其相对于竞争对手的一个优势，包括操纵 DOM 的库(如 jQuery)在内。使用 AngularJS 时，开发人员可以编写松散耦合的、易于测试的组件，让框架创建组件、解析依赖并在收到请求时把它们传递给其他组件。
- 指令：可以把指令描述为特定 DOM 项(如元素、特性、样式等)上的标记。这是一种强大的功能，可用来指定自定义的、可重用的、类似 HTML 的元素和特性，它们可以为展示组件定义数据绑定和/或其他具体行为。
- 双向数据绑定：模型和视图组件之间自动同步。当模型中的数据改变时，视图会反映出来；当视图中的数据改变时，模型也会被更新。这是立即、自动发生的，确保了模型和视图始终会被更新。
- 单页面方法：AngularJS 是第一个完全不需要重新加载页面的框架。这使服务器端和客户端都能受益：服务器端收到更少、更小的网络请求，而客户端能够实现更平滑的切换和响应性更好的体验。另外，还为单页面应用程序模式铺平了道路，后来 React、Vue.js 和其他库框架也采用了这种模式。
- 缓存友好：AngularJS 的所有神奇操作都发生在客户端，不需要服务器端生成 UI/UX 部分。因此，所有 AngualrJS 网站都可以缓存到任何地方，并通过 CDN 提供。

注意 关于 AngularJS 从 0.9.0 到 1.7.8 之间提供的功能、改进和 bug 修复的详细清单，请访问下面的链接。

AngularJS 1.x 更新日志：https://github.com/angular/angular.js/blob/master/CHANGELOG.md。

3. Angular 2

AngularJS 的新版本发布于 2016 年 9 月 14 日，也称为 Angular JS 2，它基于新的 ECMAScript 6(官方名称为 ECMAScript 2015)规范完全重写了上一个版本。与 ASP.NET Core 重写一样，这种变革在架构级别、HTTP 管道处理、应用程序生命周期和状态管理方面引入了许多突破性改变，使得把老代码移植到新版本成为几乎不可能完成的任务。尽管保留了原来的名称，但新版本是一个全新的框架，与上一个版本没有太大共同点。

Angular 2 没有与 AngularJS 向后兼容，明显表明开发团队决定采用一种全新的方法：不只在代码语法上做出改变，在思考和设计客户端应用方面也做出了改变。新的 Angular 版本更加模块化，基于组件的程度更高，提供了一个新的、改进的依赖注入模型，以及老版本中不具备的许多编程模式。

下面简单介绍 Angular 2 中引入的最重要改进。

- 语义化版本：Angular 2 是第一个使用语义化版本(semantic versioning，也称为 SemVer)的版本。SemVer 是版本化软件的一种通用方式，使开发人员能够跟踪版本，而不必翻阅更新日志的细节。SemVer 基于 3 个数字(X.Y.Z)，其中 X 代表主版本，Y 代表小版本，Z 代表修订版本。具体来说，向稳定的 API 引入不兼容的 API 修改时，需要递增 X 代表的主版本号；当添加向后兼容的功能时，需要递增 Y 代表的小版本号；当修复向后兼容的 bug 时，需要递增 Z 代表的修订版本号。这种改进很容易被人低估，但是在大部分现代软件开发场景中，持续交付(Continuous Delivery，CDE)极为重要，新版本以极快频率发布，此时语义化版本是必须具备的一项功能。
- TypeScript：如果你是一名有经验的 Web 开发人员，可能已经知道 TypeScript 是什么。不知道也没有关系，后面将进行详细介绍，因为我们在介绍 Angular 的章节中将大量使用 TypeScript。现在只是简单说明一下，TypeScript 是 JavaScript 的一个超集，由 Microsoft 开发出来，允许使用 ES2015 的全部功能(如 Default-Rest-Spread 参数、模板字面量、箭头函数、promise 等)，并在开发过程中添加了强大的类型检查和面向对象功能(如类和类型声明)。TypeScript 源代码能够转译为所有浏览器都理解的标准 JavaScript 代码。
- 服务器端渲染(Server-Side Rendering，SSR)：Angular 2 提供 Angular Universal，这是一种开源技术，允许后端服务器运行 Angular 应用程序，只把得到的静态 HTML 文件提供给客户端。简言之，服务器将对页面进行第一遍渲染，以便更快交付给客户端，然后立即使用客户端代码刷新页面。SSR 有一些值得注意的地方，例如主机上必须安装 Node.js 才能执行必要的预渲染步骤，以及获得 node_modules 文件夹，但是可以显著缩短应用程序在典型浏览器中的响应时间，从而缓解 AngularJS 的性能问题。
- Angular Mobile Toolkit(AMT)：针对创建高性能的移动应用设计的一套工具。
- 命令行接口(Command-Line Interface，CLI)：开发人员可以在 Angular 2 中引入的 CLI 中使用控制台/终端命令和简单的测试 shell 生成组件、路由、服务和管道。
- 组件：组件是 Angular 2 的主要组成模块，完全取代了 AngularJS 的控制器和作用域，并且接管了原来的指令负责的大部分任务。Angular 2 应用程序的应用数据、业务逻辑、模板和样式都可以用组件创建。

注意　在 ASP.NET Core 和 Angular 2 的最终版发布后不久，我在 2016 年 10 月出版了自己的第一本著作：*ASP.ENT Core and Angular 2*。在这本书中，我尽力探讨了这些功能。详细信息见：

https://www.packtpub.com/application-development/aspnetcore-and-angular-2。

4. Angular 4

2017 年 3 月 23 日，Google 发布了 Angular 4。在这个日期之前，有许多 Angular 组件被独立开发，例如 Angular Router 已经进入了版本 3.x。为了把所有这些组件的主版本统一起来，Angular 的第 3 版这个版本号被跳过了。从 Angular 4 开始，整个 Angular 框架被统一为使用相同的“主版本号.小版本号.修订版本号”SemVer 模式。

这个新的主版本引入了一定数量的突破性修改，如新的、改进后的路由系统，对 TypeScript 2.1+的支持(并且将其定为必要条件)，并弃用了一些接口和标签。另外还做了如下的大量改进。

- 预先(ahead-of-time)编译：Angular 4 在生成阶段编译模板，并相应地生成 JavaScript 代码。相比 AngularJS 和 Angular 2 使用的 JIT 模式(在运行时编译应用)，这是架构上的巨大改进。例如，不只应用程序在启动时变得更快(因为客户端不需要进行编译)，而且大部分组件错误将在生成时(而不是运行时)抛出/中断，从而能够实现更加安全的、稳定的部署。
- 动画 npm 包：所有现有的和新增的 UI 动画和效果被移动到@angular/animations 包中，而不是作为@angular/core 的一部分。这是一个很聪明的决定，使得不需要动画的应用能够丢开这部分代码，从而变得更小、甚至更快。

其他值得注意的改进包括：提供了一个新的表单验证器，可检查有效的电子邮件地址；为 HTTP 路由模块中的 URL 参数提供了一个新的 paramMap 接口；对国际化提供了更好的支持等。

5. Angular 5

Angular 5 发布于 2017 年 11 月 1 日，提供了对 TypeScript 2.3 的支持、少部分的突破性修改、许多性能和稳定性方面的改进，以及一些新功能，其中一些如下。

- 新的 HTTP 客户端 API：从 Angular 4.3 开始，@angular/http 模块失宠，新的@angular/common/http 包成为宠儿，它提供了更好的 JSON 支持、拦截器和不可变的请求/响应对象等。Angular 5 中完成了这种切换，老模块已被弃用，推荐在所有应用中使用新的模块。
- 状态转移 API：这是一种新功能，使开发人员能够在服务器和客户端之间转移应用程序的状态。
- 提供一组新的路由事件，从而在更细的粒度上控制 HTTP 生命周期：ActivationStart、ActivationEnd、ChildActivationStart、ChildActivationEnd、GuardsCheckStart、GuardsCheckEnd、ResolveStart 和 ResolveEnd。

注意 2017 年 11 月，我的著作 *ASP.NET Core 2 and Angular 5* 也出版了，该书讨论了前面提到的大部分改进：
https://www.packtpub.com/application-development/aspnetcore-2-and-angular-5。

2018 年 6 月，该书的内容作为视频课程提供：
https://www.packtpub.com/web-development/asp-net-core-2-and-angular-5-video。

6. Angular 6

Angular 6 发布于 2018 年 4 月，主要是一个维护性版本，更加关注提高框架及其工具链在整体上的一致性，而不是添加新功能。因此，Angular 6 中没有重大的突破性改变。RxJS 6 支持一种新的注册提供者的方式，即新增的 providedIn 可注入装饰器，改进了对 Angular Material 的支持(Angular Material 是专门用于在 Angular 客户端 UI 中实现材料设计的一个组件)，提供了更多 CLI 命令/更新等。

另外一个有必要提到的改进是新增的 ng add CLI 命令，它使用包管理器下载新的依赖，并调用一个安装脚本把配置变化更新到项目中，添加额外的依赖，添加包特定的初始化代码。

最后，Angular 团队引入了 Ivy，这是下一代 Angular 渲染引擎，旨在提高应用程序的速度，降低应用程序的大小。

7. Angular 7

Angular 7 发布于 2018 年 10 月，是一次重大更新。Google 的开发人员关系维护主管，也是一位著名的 Angular 发言人，Stephen Fluin 在 Angular 的官方开发博客上就这次正式发布写了下面的一段话，可以证明我们的猜测：

“这是一次跨越整个平台的重大发布，包括核心框架、Angular Material 和与主版本同步的 CLI。这个版本包含了工具链的新功能，并促成了几个合作伙伴发布新版本。”

下面列出了新功能。

- 易于升级：由于版本 6 中做的基础工作，Angular 团队能减少把现有 Angular 应用从老版本升级到最新版本所需的步骤。https://update.angular.io 提供了详细的步骤，这是一个极为有用的交互式 Angular 升级向导，可用来快速了解必要的步骤，例如 CLI 命令、包更新等。完成这些步骤后，才能把现有的 Angular 应用从老版本升级到最新的版本。
- CLI 更新：通过遵守前面提到的步骤，有一个新的命令会试图自动升级 Angular 应用程序及其依赖。
- CLI 提示：Angular 命令行接口经过修改，能够在运行常用命令(如 ng new 或 ng add @angular/material)时给出提示，帮助开发人员发现内置的功能，如路由、SCSS 支持等。
- Angular Material 和 CDK：提供了更多 UI 元素，如虚拟滚动，这个组件能够根据列表的可见部分，加载和卸载 DOM 中的元素，从而能够为使用非常大的可滚动列表的用户打造极快的体验；还包括 CDK 原生的拖放支持以及改进的下拉列表元素等。
- 合作伙伴发布：改进了与许多第三方社区项目的兼容性；如 Angular Console 是一个可下载的控制台，用于在本地机器上启动和运行 Angular 项目；AngularFire 用于 Firebase 集成的官方 Angular 包；Angular for NativeScript 用于 Angular 和 NativeScript 的集成，NativeScript 是一个使用 JavaScript 和/或基于 JS 的客户端框架构建原生 iOS 和 Android 应用的框架。针对 StackBlitz 的一些有趣的、Angular 特有的功能，StackBlitz 是一个在线 IDE，可用于创建 Angular 和 React 项目，如标签式编辑器。还与 Angular Language Service 集成等。
- 更新了依赖：添加了对 TypeScript 3.1、RxJS 6.3 和 Node 10 的支持，不过为了向后兼容，仍然允许使用之前的版本。

提示　Angular Language Service 是在 Angular 模板中获取自动完成、错误、提示和导航的一种方式，可以把它想象成为语法高亮器、智能感知和实时语法错误检查器的混合。Angular 7 添加了对 StackBlitz 的支持，但在这个版本之前，只有 Visual Studio Code 和 WebStorm 才提供了这种功能。

关于 Angular Language Service 的更多信息，请访问下面的 URL:
https://angular.io/guide/language-service。

8. Angular 8

Angular 7 发布后不久，在 2018 年 5 月 29 日发布了 Angular 8。新版本的主要变化是人们期待已久的 Ivy，这是一个新的 Angular 编译器/运行时。虽然从 Angular 5 就开始开发 Ivy 项目，但 Angular 8

第一次正式提供了运行时切换选项，允许开发人员选择使用 Ivy。从 Angular 9 开始，Ivy 将成为默认的运行时。

提示 要在 Angular 8 中启用 Ivy，开发人员必须在应用程序的 tsconfig.json 文件的 angularCompilerOptions 节中添加一个"enableIvy": true 属性。
如果想了解 Ivy 的更多信息，建议详细阅读 Cédric Exbrayat 撰写的下面这篇文章，他是 Ninja-Squad 网站的联合创始人和培训师，现在是 Angular 开发团队的成员。
https://blog.ninja-squad.com/2019/05/07/what-is-angularivy/。

下面列出其他值得注意的改进和新功能。

- Bazel 支持：Angular 8 是第一个支持 Bazel 的版本，Bazel 是 Google 开发和使用的一个免费的软件工具，用于自动完成软件的生成和测试工作。对于想要自动化交付管道的开发人员，它是一个很有用的工具，允许增量生成和测试，甚至允许在生成农场(build farm)上配置远程生成(和缓存)。
- 路由：引入了新语法，能够使用 TypeScript 2.4+的 import()语法来声明延迟加载路由，而不是依赖一个字符串字面量。为了向后兼容，保留了旧语法，但是可能很快就会弃用旧语法。
- 服务工作线程：引入了一种新的注册策略，以允许开发人员选择在什么时候注册工作线程，而不是在应用程序启动生命周期结束时自动注册。通过使用新的 ngsw-bypass 头部，还能够为特定 HTTP 请求绕过服务工作线程。
- 工作空间 API：提供了一种新的、更方便的方式来读取和修改 Angular 工作空间配置，而不需要手动修改 angular.json 文件。

注意 在客户端开发中，服务工作线程是浏览器在后端运行的一个脚本，用来执行不需要用户界面或用户交互的工作。

Angular 8 还引入了一些突破性改变(主要是 Ivy 带来的改变)，并删除了一些弃用已久的包，例如@angular/http，它在 Angular 4.3 中被@angular/common/http 取代，并在 Angular 5.0 中被正式弃用。

提示 Angular 官方弃用指导中提供了所有弃用 API 的完整列表，其 URL 如下所示:
https://angular.io/guide/deprecations。

9. Angular 9

经历了 2019 年第四季度的一系列候选版本后，Angular 9 在 2020 年 2 月正式发布，是目前最新的版本。

新版本带来了下面的新功能。

- JavaScript bundle 和性能：庞大的 bundle 文件是以前的 Angular 版本中最累赘的问题之一。Angular 9 解决了这个问题，从而显著缩短了下载时间并提高了总体性能。
- Ivy 编译器：这个新的 Angular 生成和渲染管道在 Angular 8 中还是选择性预览，但在 Angular 9 中成为默认的渲染引擎。
- 无选择器的绑定：这是以前的渲染引擎中可用、但 Angular 8 Ivy 预览没有提供的功能。现在，Ivy 中也可使用这种功能。
- 国际化：这是另一种 Ivy 增强，通过在 Angular CLI 中使用新增的 i18n 特性，能够生成为翻译器创建文件所需的大部分标准代码，以及使用多种语言发布 Angular 应用。

提示　新增的 i18n 特性是一种代称，用作国际化(internationalization)的别名。数字 18 表示单词 internationalization 的第一个字母 i 和最后一个字母 n 之间的字母数。这个术语可能是数字设备公司(Digital Equipment Corporation，DEC)在 20 世纪 70 年代或 80 年代创造的，另外有一个术语 l10n，代表本地化(localization)。创造这两个术语，是因为涉及的两个单词太长了。

Ivy 编译器在 Angular 的未来发展中将扮演一个重要的角色，所以有必要多说几句。

你可能已经知道，对于任何前端框架的整体性能，渲染引擎扮演着重要角色，因为这个工具负责将展示逻辑(在 Angular 中就是组件和模板)执行的操作和意图转换为更新 DOM 的指令。如果渲染器比较高效，就需要更少的指令，从而提高总体性能，同时减少必要的 JavaScript 代码量。因为 Ivy 生成的 JavaScript bundle 比原来的渲染引擎小得多，所以 Angular 9 在性能和大小上的整体改进很明显。

我们对 ASP.NET Core 和 Angular 生态系统的近期发展所做的简要介绍到此结束。在接下来的小节中，将总结一下我们在 2020 年仍然选择使用 NET Core 和 Angular 的最重要原因。

1.2.3　选择.NET Core 和 Angular 的理由

我们已经看到，在过去 3 年中，这两个框架都经历了密集的变化。这导致它们重建内核，之后就一直面临着重返巅峰的压力，或者至少不失去自己的地盘。面临的竞争对手有两种：在它们的黄金时代过去后问世的大部分现代框架，如用于服务器端的 Python、Go 和 Rust，用于客户端的 React、Vue.js 和 Ember.js，还有急于统治开发者世界的 Node.js 和 Express 生态系统；另一种就是 20 世纪 90 年代和 21 世纪初的大部分老竞争对手，例如 Java、Ruby 和 PHP，它们依然生机勃勃。

虽然如此，在 2019 年选择使用 ASP.NET Core 有如下许多有说服力的理由。

- 性能：新的.NET Core Web 栈非常快，版本 3.x 尤其明显。
- 集成：支持大部分现代客户端框架，包括 Angular、React 和 Vue.js。
- 跨平台方法：.NET Core Web 应用程序几乎能以无缝方式运行在 Windows、macOS 和 Linux 上。
- 托管：.NET Core Web 应用程序几乎可以托管在任何地方，从装有 IIS 的 Windows 机器，到装有 Apache 或 Nginx 的 Linux 设备；从 Docker 容器，到使用 Kestrel 和 WebListener HTTP 服务器的自托管场景(不过这种场景很罕见)。
- 依赖注入：框架支持内置的依赖注入设计模式，在开发过程中提供了众多优势，如降低依赖、改进代码可重用性、可读性和便于测试。
- 模块化的 HTTP 管道：ASP.NET Core 中间件使开发人员能够在细粒度上控制 HTTP 管道，既可以把管道缩减到其核心部分(针对极轻量级的任务)，也可以使用强大的、高度可配置的功能(如国际化、第三方身份验证/授权、缓存、路由等)来增强管道。
- 开源：整个.NET Core 栈作为开源项目发布，在强大的社区支持上，因此每天会被数千名开发人员进行评审和改进。
- 并排执行：它支持在同一台机器上同时运行应用程序或组件的多个版本。这意味着在同一台机器上，同时有公共语言运行时的多个版本，并且有多个版本的应用程序和组件使用该运行时的不同版本。对于大部分现实开发场景来说，这一点很有帮助，因为开发团队能够控制某个应用程序绑定到组件的哪些版本，以及应用程序使用运行时的哪个版本。

至于 Angular 框架，我们选择它而不是其他优秀的 JS 库(如 React、Vue.js 和 Ember.js)的最重要原因是，它本身已经提供了大量功能，因此成为一个最合适的选项，不过这可能也使它不如其他框架/

库那么简单。再加上 TypeScript 语言提供的一致性优势，可以说 Angular 从 2016 年重生直到现在，都比其他框架更加积极地采用了框架方法。过去 3 年间，这一点被一再证实，因为这个框架在这 3 年间经历了 6 个主版本，并且在稳定性、性能和功能方面有了很大的提升，同时并没有失去很多向后兼容性、最佳实践和总体方法。这些理由足以说服我们投入 Angular 框架上，希望它能够继续保持并发展这些优势。

了解使用这两个框架的理由后，我们来问自己一个问题：要了解关于这两个框架的更多信息，最佳方式是什么？接下来的小节应该提供我们需要的答案。

1.3 全栈方法

学习使用 ASP.NET Core 和 Angular，意味着能使用 Web 应用程序的前端(客户端)和后端(服务器端)。换言之，意味着能够设计、组装和交付一个完整的产品。

要做到这一点，我们需要研究下面的主题：

- 后端编程
- 前端编程
- UI 样式和 UX 设计
- 数据库设计、建模、配置和管理
- Web 服务器配置和管理
- Web 应用程序部署

初看起来，这种方法可能不符合常识，因为一名开发人员不应该独自完成所有工作。每个开发人员都知道，后端和前端需要的技能和经验是完全不同的。那么，为什么我们应该采用这种方法呢？

在回答这个问题之前，应该理解前面说“能够”的含义。我们不需要成为技术栈中每个层次的专家，也没有人期望我们成为这样的专家。当我们选择拥抱全栈方法时，真正需要做的是提高对使用的整个栈的认识，这意味着需要知道后端如何工作，以及后端如何与前端通信。我们需要知道如何存储、检索数据以及将数据提供给客户端。需要知道有哪些交互，以便把构成 Web 应用程序的各个组件划分到不同的层中，还需要知道安全问题、身份验证机制、优化策略、负载均衡技术等。

这并不意味着我们在这些领域都拥有强大的技能；事实上，我们也很难做到这一点。尽管如此，如果想采用全栈方法，就需要理解所有这些领域的含义、角色和作用范围。另外，在需要的时候，我们应该有能力在这些领域熟练工作。

1.4 SPA、NWA 和 PWA

为了演示如何结合 ASP.NET Core 和 Angular，使它们发挥最大能力，最佳方式是构建一些小型 SPA 项目，并使其具有大部分原生 Web 应用程序(Native Web Application)的功能。做出这种选择的原因很明显：没有更好的方式来展示它们如今提供的一些最好的功能。我们在构建项目的过程中，将能够使用现代接口和模式，如 HTML5 pushState API、webhook、基于数据传输的请求、动态 Web 组件、UI 数据绑定，以及无状态的、AJAX 驱动的架构。我们还将使用一些 NWA 功能，如服务工作线程、Web 清单文件等。

如果你不知道这些定义和缩写的含义，不必担心，在接下来的几节中，我们将探讨这些概念。这几节将专门介绍下面 3 种类型的 Web 应用程序的重要功能：SPA、NWA 和 PWA。在讨论过程中，我们还将试图理解产品负责人对典型 Web 项目最常见的期望。

1.4.1 单页面应用程序

简单来说，单页面应用程序(Single-Page Application，SPA)是一个基于 Web 的应用程序，但尽力提供与桌面应用程序相同的用户体验。如果考虑到所有 SPA 仍然是通过 Web 服务器提供，因而是通过 Web 浏览器访问的(就像其他任何标准网站一样)，就能够理解到，要实现它们期望的行为，只能修改 Web 开发中常用的一些默认模式，如资源加载、DOM 管理和 UI 导航。在好的 SPA 中，内容和资源(HTML、JavaScript、CSS 等)是在一次页面加载中获取的，或者是在需要时动态获取的。这也意味着页面不会重新加载或刷新，而只是发生改变来响应用户操作，在后端执行必要的服务器端调用。

如今，一个有竞争力的 SPA 应该提供下面的关键功能。

- 没有服务器端往返：一个有竞争力的 SPA 在重绘客户端 UI 的任何部分时，不需要完整的服务器端往返过程来获取一个完整的 HTML 页面。这主要是通过实现关注点分离(Separation of Concerns，SoC)设计原则来实现的，该原则意味着数据源、业务逻辑和展示层将被分离。
- 高效路由：一个有竞争力的 SPA 能够在用户的导航过程中，使用有组织的、基于 JavaScript 的路由器来跟踪用户当前的状态和位置。后续章节在介绍服务器端和客户端路由的概念时，将深入讨论这一点。
- 性能和灵活性：一个有竞争力的 SPA 通常使用某种 JavaScript SDK(Angular、jQuery、Bootstrap 等)，将所有 UI 转移到客户端完成。这通常有助于提高网络性能，因为增加客户端渲染和离线处理能够降低 UI 对网络的影响。但是，这种方法的真正优点在于使 UI 变得更加灵活，因为开发人员能够完全重写应用程序的前端，除了一些静态资源文件之外，对服务器只有很小的影响，甚至没有影响。

这只是一个合理设计的、有竞争力的 SPA 的一些主要优势。如今，这些方面起着重要的作用，因为许多商业网站和服务正在从传统的多页面应用程序(Multi-Page Application，MPA)思想转变到完全的或者混合的、基于 SPA 的方法。

1.4.2 原生 Web 应用程序

多页面应用程序从 2015 年以来变得越来越受欢迎，它们常被称为原生 Web 应用程序(Native Web Application，NWA)，因为它们通常实现了许多小规模的单页面模块，并在一个多页面结构上把它们结合起来，而不是构建单独一个庞大的 SPA。

更不必说，有大量企业级 SPA 和 NWA 每天完美地服务着众多用户，例如 WhatsApp Web、Teleport Web 和 Flickr，以及多种 Google Web 服务，包括 Gmail、Contacts、Spreadsheet、Maps 等。这些服务，加上它们庞大的用户群，证明了这不是一个短期趋势，过一段时间就会消亡。相反，我们在见证一种新模式的完成，它肯定会长期存在下去。

1.4.3 渐进式 Web 应用程序

在 2015 年，另一种 Web 开发模式进入人们的视野。当时，Frances Berriman(一位英国的自由职业设计师)和 Alex Russel(一位 Google Chrome 工程师)第一次使用术语 PWA(Progressive Web Application)表示这样的 Web 应用程序，它们利用了现代浏览器支持的两种重要的新功能：服务工作线程和 Web 清单文件。通过使用标准的、基于 Web 的开发工具(如 HTML、CSS 和 JavaScript)，这两种重要的改进可被成功地用来交付通常只在移动应用中可用的一些功能，如推送通知、离线模式、基于权限的硬件访问等。

渐进式 Web 应用程序的崛起发生在 2018 年 3 月 19 日，当时 Apple 在 Safari 11.1 中实现了对服务工作线程的支持。从这个时候起，PWA 在行业中被广泛采用，这主要归功于它们相对 MPA、SPA 和 NWA 提供的不可否认的优势：更快的加载速度、更小的应用大小、更高的用户参与度等。

根据 Google 的表述，渐进式 Web 应用的主要技术特征如下所示。

- 渐进式：使用渐进式增强原则，可被使用各种浏览器的每个用户使用。
- 响应性：适合任何设备类型，包括桌面计算机、移动设备、平板电脑或将来出现的设备。
- 与连接无关：服务工作线程允许离线或在低质量的网络上使用 Web 应用。
- 与移动应用类似：提供应用风格的交互和导航，让用户使用起来感觉与移动应用类似。
- 最新：服务工作线程的更新进程决定总是将 Web 应用更新到最新。
- 安全：通过 HTTPS 提供，可以防止窃听，并确保内容未被篡改。
- 可被发现：通过 Web 清单(manifest.json)文件和一个注册的服务工作线程可被识别为一个应用程序，被搜索引擎发现。
- 再次吸引用户：能够使用推送通知保持用户参与度。
- 可安装：提供主页图标，并不需要使用 App Store。
- 可链接：可通过 URL 轻松分享，并不需要复杂的安装过程。

不过，它们的技术基础可被限制到下面的子集。

- HTTPS：必须从一个安全的来源提供，这意味着通过 TLS 传输，显示绿色挂锁图标(没有活动的混合内容)。
- 最小离线模式：即使设备没有连接到网络，也必须能够启动，并且提供受限的功能，至少显示一个自定义离线页面。
- 服务工作线程：必须注册一个服务工作线程，并使其具备一个 fetch 事件处理程序(用于提供前面提到的最低离线支持)。
- Web 清单文件：需要引用一个有效的 manifest.json 文件，其中至少有 4 个关键属性(name、short_name、start_url 和 display)和必要图标的最小集合。

注意 如果想直接阅读这些信息的来源，可从下面的链接访问 Google Developers 网站的相关文章:
https://developers.google.com/web/progressive-web-apps/。
另外，Alex Russell 的 Infrequently Noted 博客上还写了两篇后续文章:
https://infrequently.org/2015/06/progressive-apps-escapingtabs-without-losing-our-soul/，
https://infrequently.org/2016/09/what-exactly-makessomething-a-progressive-web-app/。
Alex Russell 从 2008 年 12 月以来，就担任 Google 的高级软件工程师。

虽然存在一些相似之处，但 PWA 和 SPA 是两个不同的概念，有不同的需求，并在许多重要方面存在区别。我们可以看到，前面提到的所有 PWA 需求都没有提及单页面应用程序或者服务器端往返。渐进式 Web 应用程序能够在一个 HTML 页面和基于 AJAX 的请求中工作(从而也是一个 SPA)，但它也可以请求其他服务器渲染的页面或者静态页面，并/或执行标准的 HTTP GET 或 POST 请求，就如同 MPA 一样。反过来也一样：任何 SPA 也可以实现任何一个 PWA 技术条件，这要取决于产品负责人的需求(稍后将详细介绍)、使用的服务器端和客户端框架以及开发人员的最终目标。

因为我们将使用 Angular，而 Angular 关注单页面应用程序开发，并且自 Angular 5 以来提供了强大而稳定的服务工作线程实现，所以我们能够兼得两种技术的优势。因此，我们将在有需要的时候，使用服务工作线程，从而获得它们带来的可靠性和性能优势，同时仍然采用 SPA 方法。不仅如此，每当能够使用微服务给应用程序减轻一些工作负担的时候，就像好的原生 Web 应用程序那样，我们

将实现一些战略性 HTTP 往返(和/或其他基于重定向的技术)。

这些功能能否满足现代市场的需求？我们来找出答案。

1.4.4 产品负责人的期望

许多现代敏捷软件开发框架(如 Scrum)产生了许多代码，其中最值得注意、但也被低估的一点是角色的含义和定义的重要性。在这些角色中，最重要的是产品负责人，在极限编程方法学中也称为客户，在其他领域则称为客户代表。产品负责人提出我们需要满足的期望，告诉我们要交付的最重要功能是什么，并且他们将根据业务价值而非底层的架构价值来调整开发工作的优先级。管理层将授权他们做出决策，以及做出一些艰难的决定，这有时候很有帮助，有时候则不然；无论如何，这常常会对开发日程产生重要影响。简言之，产品负责人对项目负责，所以为了交付满足他们期望的 Web 应用程序，我们需要理解他们的构想。

这一点始终成立，即使项目的产品负责人是你父亲、另一半或者好朋友也一样。

明确了这一点，我们来看看在典型的基于 Web 的 SPA 项目中，产品负责人最常见的一些需求。我们应该会看到，选择使用 ASP.NET Core 和 Angular 能否满足每个期望，如下所示。

- 早发布：无论我们在销售什么，客户总是想看到他们买到的东西。例如，如果我们计划使用一个敏捷开发框架(如 Scrum)，那么在每个冲刺结束时，需要发布一个可以交付的产品。如果计划使用基于瀑布的方法，就会有里程碑等。为了高效组织开发工作，我们最好采用一种迭代式和/或面向模块的方法。ASP.NET Core 和 Angular 在底层采用了基于 MVC 或 MVVM 的模式，因此能够实现关注点分离，使我们逐渐形成这种思维方式。
- GUI 优先于后端：我们常被要求创建 GUI 和前端功能，因为对客户来说，这是他们真正能够看到和衡量的东西。这意味着我们需要模拟数据模型，然后尽早处理前端，而推迟依赖后端的功能，即使这意味着暂时不实现该功能。注意，这种方法并不一定是坏事；我们这么做，并不只是为了满足项目负责人的期望。相反，选择使用 ASP.NET Core 和 Angular，使我们能够轻松地将展示层和数据层解耦，实现前者而模拟后者，这是很有帮助的。我们能够抢先看到工作方向，而不至于浪费宝贵的时间，或者被迫做出可能错误的决定。ASP.NET Core 的 Web API 接口提供了合适的工具来实现这种行为。通过使用 Visual Studio 提供的控制器模板，以及 Entity Framework Core 支持的内存数据上下文(我们能够使用 Entity 模型和代码优先来访问)，能够在几秒内创建一个 Web 应用程序的骨架。之后，就可以使用 Angular 的展示层工具箱转向 GUI 设计，直到得到期望的结果。对结果感到满意后，只需要正确地实现 Web API 控制器接口并绑定到实际的数据。
- 快速完成：除非能够在合理的时间段内完成，否则前面提到的工作都没有意义。这是选择结合使用服务器端框架和客户端框架的主要原因之一。我们选择 ASP.NET Core 和 Angular，不只是因为它们都有牢固、一致的基础，也因为它们能够满足我们的需要，可在各自一端完成工作，然后向另一端提供一个可用的接口。
- 适应性：如敏捷宣言所说，响应变化比遵循计划更重要。在软件开发中，这一点尤为适用，我们甚至可以说，不能处理变化的项目是一个失败的项目。这是拥抱我们选择的两个框架带来的关注点分离的另一个重要原因，因为这使开发人员能够管理、甚至在一定程度上欢迎开发阶段可能出现的大部分布局和结构上的变化。

提示 前面提到了 Scrum，这是最流行的敏捷软件开发框架之一。不知道这个框架的开发人员应该花些时间，了解一下它为结果驱动的团队领导和/或项目经理提供的功能。下面是一个不错的起点。

https://en.wikipedia.org/wiki/Scrum_(software_development)。

如果你对瀑布模型感兴趣，可以通过下面的链接了解更多信息。

https://en.wikipedia.org/wiki/Waterfall_model。

注意，这里的介绍可能并不完整，因为不知道具体的项目，是无法了解全部期望的。我们只是试图为下面这个一般性问题提供一个宽泛的答案：如果要构建一个 SPA 和/或 PWA，ASP.NET Core 和 Angular 是合适的选择吗？答案无疑是肯定的，尤其把二者结合起来更加合适。

这意味着我们已经完成介绍了吗？并不是，因为我们不打算只是简单地告诉你这个结论。相反，接下来，我们不再使用一般性术语来进行说明，而开始通过实际操作演示结论。在接下来的小节中，我们将准备、构建并测试一个单页面应用程序示例。

1.5 SPA 项目示例

现在，我们需要构想一种合适的测试场景，这个场景需要与我们最终要处理的场景类似。这个场景将是一个 SPA 示例项目，具有我们期望一个可交付产品具备的所有核心功能。

为此，我们将首先把自己的角色转变为客户，提出一个将与自己的另一个角色分享的想法。然后，把自己的角色转变为开发人员，并把抽象的计划拆分成需要实现的需求点。它们将成为项目的核心需求。最后，设置工作环境，获取必要的包，添加资源文件，并在 Visual Studio IDE 中配置 ASP.NET Core 和 Angular 框架。

并不是常见的 Hello World

本书将编写的代码不会是一个粗浅的项目，仅用其来演示全栈开发的概念；我们不是时不时给出一点代码，期望你能够把它们结合起来。我们的目标是使用框架创建一个还不错的、有实际用途的 Web 应用程序，使其具备服务器端 Web API 和客户端 UI，而且在这个过程中，将遵守最新的开发最佳实践。

每一章将专门介绍一个核心主题。如果你觉得自己已经了解该主题，可以跳到下一章。反过来，如果你愿意跟着我们一章一章地学习，则将了解到 ASP.NET Core 和 Angular 最有用的方面，并看到如何结合使用它们，完成最常见、最有用的各种 Web 开发任务，包括简单的任务和复杂的任务。这种投入将会收到回报，因为你将得到一个可维护、可扩展、结构合理的项目，并知道如何构建自己的项目。接下来的章节将带领你走完整个过程。在此期间，你还将学习如何处理一些重要的、高层次的问题，例如 SEO、安全、性能问题、最佳编码实践和部署，因为当你的应用程序最终发布到一个生产环境时，它们将变得十分重要。

为避免学习过程太枯燥，我们将选择一些有趣的主题和场景，但它们在现实世界中都有其用途。要明白我们在说什么，请继续阅读。

1.6 准备工作空间

我们首先要做的是设置工作空间。这并不困难，因为我们必须使用的工具很少，包括 Visual Studio 2019、新版本的 Node.js 运行时、一个开发 Web 服务器(如内置的 IIS Express)和一个不错的源代码控制系统，如 Git、Mercurial 或 Team Foundation。我们将认为你已经安装了一个源代码控制系统。

提示 如果还没有安装，则首先应该安装一个源代码控制系统，然后继续阅读后面的内容。可以访问 www.github.com、www.bitbucket.com 或你喜欢的其他在线源代码管理服务，创建一个免费账户，然后花一些时间来学习如何有效使用这些工具。可以肯定的是，你不会后悔。

在接下来的小节中，我们将创建一个 Web 应用程序项目，安装或升级包和库，生成并最终测试工作成果。但是，在那之前，我们来花几分钟的时间，理解一个非常重要的概念。这对于正确使用本书很重要，如果不能理解，可能会让你在阅读过程中遇到挫折。

1.6.1 免责声明

在继续介绍后面的内容之前，有一个重要问题需要先解释清楚。如果你是一名经验丰富的开发人员，很可能已经知道我们要说明的问题；但是，因为本书针对(几乎)每个人，所以我认为有必要尽早说明这个问题。

本书将大量使用许多不同的编程工具、外部组件、第三方库等，其中大部分(如 TypeScript、NPM、NuGet、大部分.NET Core 框架/包/运行时等)都随着 Visual Studio 2019 一起提供，但其他一部分(如 Angular、必要的 JS 依赖和其他第三方服务器端和客户端包)需要从它们的官方存储库获取。这些工具、组件或库等应该以完全兼容的方式工作，但是，随着时间的流逝，它们都会发生改变或被更新。这些更新有可能影响它们彼此交互的方式，项目的健康程度可能会降低。

1. 代码有错的说法

为了尽可能避免出现这种情况，本书在用到第三方组件的时候，总是使用固定的版本，这是通过配置文件来处理的。但是，一些更新，如 Visual Studio 和/或.NET Framework 的更新，可能不在这个范围内，导致项目出现问题。源代码可能无法再工作，或者 Visual Studio 可能突然无法正确编译项目。

发生这种情况时，经验不足的开发人员倾向于把责任推给图书。一些人甚至会开始这么想：

这么多编译错误，源代码一定有问题。

或者，他们可能这么想：

代码示例不能工作。作者一定急着把书写出来，而忘了测试自己写的代码。

不必说，这些推测基本都不符合真实情况，毕竟图书的作者、编辑和技术审校者花了大量的时间来编写、测试和优化源代码，最后生成项目，放到 GitHub 上，甚至常常把最终应用程序的工作实例发布到全世界可以公开访问的网站上。

提示 本书的 GitHub 存储库的地址如下所示。

https://github.com/PacktPublishing/ASP.NET-Core-3-andAngular-9-Third-Edition。

这里为每一章包含一个 Visual Studio 解决方案文件(Chapter_01.sln、Chapter_02.sln 等)，还有一个解决方案文件(All_Chapters.sln)包含所有章节的源代码。

有经验的开发人员会理解，如果存在有问题的代码，这些工作将无法完成；如果没有百分百可以工作的源代码，本书不可能会出版。当然，代码中可能存在少量拼写错误，但这种错误很快会被报告给出版社，所以短时间内就会在 GitHub 存储库中得到修正。在不太可能出现的情况中，问题似乎没有得到修正，例如代码报出意外的编译错误，此时新手开发人员应该花一些时间尝试理解导致错误的根本原因。

新手开发人员应该首先试着回答下面的问题：

- 我使用的开发框架、第三方库和版本是否与图书的代码相同？

- 如果我感觉有必要更新代码，那么是否意识到所做的修改可能影响源代码？我是否阅读了相关的更新日志？我是否花时间寻找可能影响源代码的突破性修改和/或已知的问题？
- 图书的 GitHub 存储库是否也受到这个问题的影响？我是否对比了存储库中的代码与我自己的代码，甚至替换我自己的代码？

如果有任何一个问题的答案是否定的，那么很可能问题不在本书上。

2. 保持上进心，但要承担责任

不要误会：如果你想使用新版本的 Visual Studio，更新 TypeScript 编译器或者升级任何第三方库，我会鼓励你这么做。这与本书想传递的理念是相同的：跳出本书代码示例的局限，知道你自己在做什么，能够做什么。

但是，如果你感觉自己已经准备好这么做，则需要相应地调整代码。大部分时候，需要做的工作并不困难，尤其在如今这个时代，你可以使用 Google 搜索问题的答案，并且/或者在 StackOverflow 上寻求答案。或许你需要加载新的类型，或许需要找到新的命名空间，并相应地做出修改。

基本上就是这样。代码会反映时间的变化。开发人员需要跟上这种变化，在必要时对代码做最少量的更改。如果更新环境后，没有意识到需要修改一些代码来使项目能够继续工作，那么能够责备的人只有你自己。

我是不是在说，作者不对本书的源代码承担责任？恰恰相反，作者始终是有责任的。作者应该尽力修复所有报出来的不兼容问题，同时保持 GitHub 存储库中的代码最新。但是，你自己也应该承担一定程度的责任，具体来说，你应该理解讲解开发的图书的基本情况，并知道时间不可避免会影响任何给定源代码。无论作者多么努力地维护源代码，打补丁总是不够快、不够全面，不能做到使这些代码在任何给定场景中总是能够工作。因此，你应该理解的最重要一点，也是现代软件开发中最有价值的概念是：变化总会发生，因此你应该有能力高效处理不可避免的变化。

拒绝接受这一点，就注定无法适应现代软件开发。

1.6.2 创建项目

假设我们已经安装了 Visual Studio 2019 和 Node.js。需要做的工作如下：

(1) 下载并安装.NET Core SDK。
(2) 检查确认.NET CLI 会使用该 SDK 版本。
(3) 创建一个新的.NET Core 和 Angular 项目。
(4) 在 Visual Studio 中检查新创建的项目。
(5) 将所有包和库更新到我们选择的版本。

下面就开始工作。

1. 安装.NET Core SDK

我们可以从 Microsoft 官方页面(https://dotnet.microsoft.com/download/dotnet-core)或 GitHub 官方发布页面(https://github.com/dotnet/core/blob/master/release-notes/README.md)下载最新版本。

安装过程十分直观，只需要按照向导操作，如图 1-1 所示。

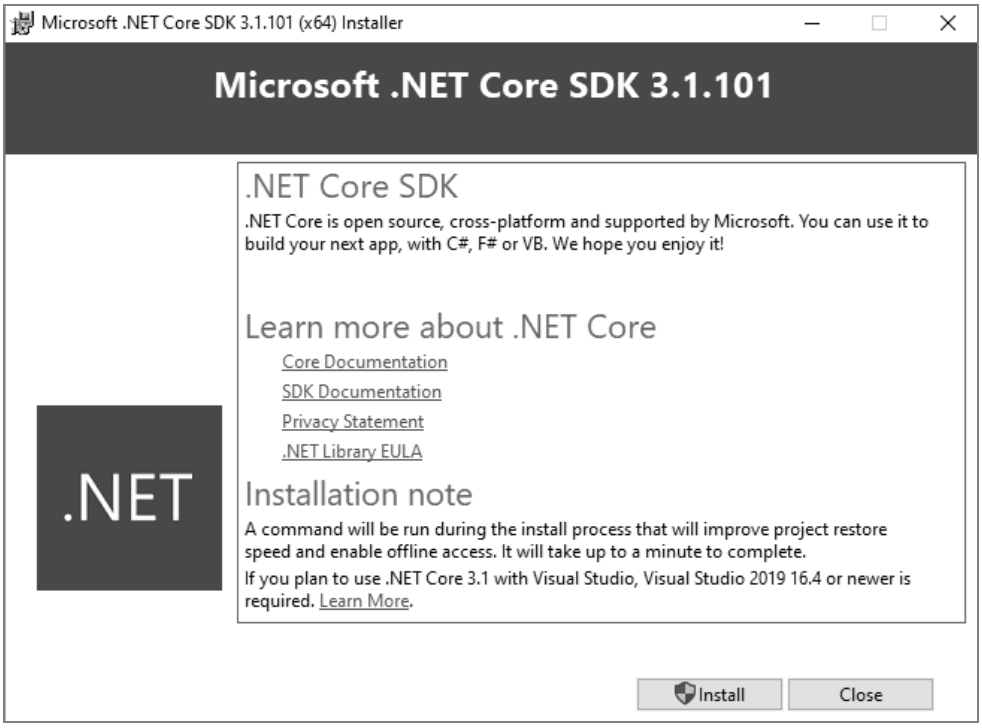

图 1-1　安装过程

整个安装过程应该在几分钟内完成。

2. 检查 SDK 版本

安装了.NET Core SDK 后，我们需要确认已经正确地设置了新的 SDK PATH，并且/或者.NET CLI 会使用该 SDK PATH。要执行这种检查，最快捷的方式是打开命令提示，键入下面的命令。

```
>dotnet --help
```

确保.NET CLI 能够成功执行，并且版本号与我们刚才安装的版本相同。

提示　如果命令提示无法执行该命令，则进入 Control Panel | System | Advanced System Settings | Environment Variables，检查 PATH 环境变量中是否有 C:\Program Files\dotnet\文件夹；如果没有，就手动添加。

3. 创建.NET Core 和 Angular 项目

接下来，需要创建第一个.NET Core 和 Angular 项目。这是我们的第一个应用程序。我们将使用.NET Core SDK 提供的 Angular 项目模板，因为它提供了一个方便的起点，会添加所有必要的文件和一个通用的配置文件，我们可在以后定制该配置文件来满足自己的需要。

在命令行，创建一个根文件夹，我们将在该文件夹中包含全部项目。然后，进入该文件夹。

提示 本书将使用\Projects\作为根文件夹。强烈建议经验不足的开发人员也使用这个文件夹，以避免与路径名太长有关的路径错误和/或问题(Windows 10 将路径名限制为不超过 260 个字符，对于一些深层嵌套的 NPM 包，这可能造成一些问题)。使用 C:盘以外的盘符是明智的做法，可以避免权限问题。

进入该文件夹后，键入下面的命令来创建 Angular 应用。

```
>dotnet new angular -o HealthCheck
```

这个命令将在 C:\Projects\HealthCheck\文件夹中创建我们的第一个 Angular 应用。可以猜到，它的名称是 HealthCheck。选择这个名称是有理由的，后面就会明白。

4. 在 Visual Studio 中打开新项目

现在是时候启动 Visual Studio 2019，快速检查新创建的项目了。双击 HealthCheck.csproj 文件，或者使用 VS2019 的主菜单(File | Open | Project/Solution)，可在 Visual Studio 中打开该项目。

打开后，应该能够看到项目的源文件树，如图 1-2 所示。

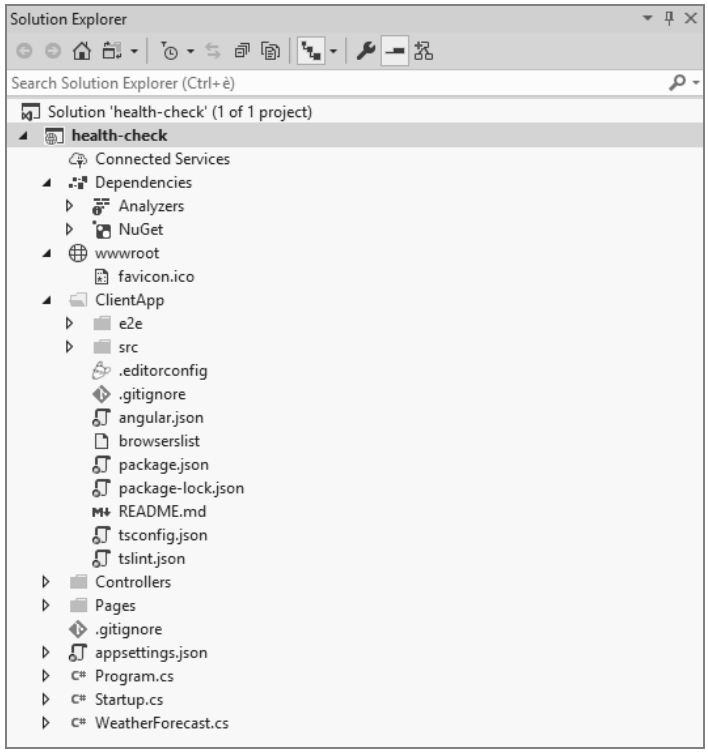

图 1-2 项目的源文件树

从图 1-2 可以看到，这是一个相当简洁的样板，只包含必要的.NET Core 和 Angular 配置文件、资源和依赖项，都是我们开始编码所需要的东西。

但是，在开始编码之前，我们来简单介绍一下。查看各个文件夹可知，工作环境包含下面的文件夹：

- 默认的 ASP.NET MVC /Controllers 和/Pages 文件夹，它们都包含一些可工作的示例。
- /ClientApp/src/文件夹包含一些 TypeScript 文件，其中包含一个 Angular 示例应用的源代码。

- /ClientApp/e2e/文件夹包含一些 E2E 示例测试，它们是用 Protractor 测试框架生成的。
- /wwwroot/文件夹。每当我们需要本地执行客户端代码，或者将其发布到其他某个位置时，Visual Studio 将使用此文件夹生成客户端代码的优化版本。该文件夹一开始为空，但项目第一次运行后将在其中填充内容。

花些时间查看这些文件夹中的内容会发现，.NET Core 的开发人员做了出色的工作，方便了创建.NET 和 Angular 项目的过程。如果将这个样板与 Visual Studio 2015/2017 中内置的 Angular 2.x/5.x 模板进行比较，会注意到可读性和代码整洁性方面的巨大改进，文件和文件夹结构也变得更好。在过去艰难地使用任务执行器(如 Grunt 或 Gulp)和/或客户端生成工具(如 webpack)的开发人员很可能会欣喜地发现，新模板完全不同于那种情况：Visual Studio 将通过底层的.NET Core 和 Angular CLI 处理所有打包、生成和编译工作，并针对开发和生产提供具体的加载策略。

注意　坦白说，选择使用像这样的预制模板有其缺陷。在一个项目中同时托管后端(.NET Core API)和前端(Angular 应用)很有用，对于学习和开发阶段都是很大的帮助，但是不推荐在生产中采用这种方法。

理想情况下，最好将服务器端和客户端代码拆分到两个项目中，以实现解耦合，这在构建基于微服务的架构时极为重要。虽然如此，能够在同一个项目中使用后端和前端，对于学习来说是一种好方法，所以这些模板对于一本介绍编程的图书来说是理想的选择，我们也将会使用这种方法。

在介绍后面的内容之前，应该快速地执行一次测试，确保项目能够正确工作。

5. 执行测试

好消息是，在目前这个阶段，执行测试很简单，只需要单击 Run 按钮或者按 F5 键，如图 1-3 所示。

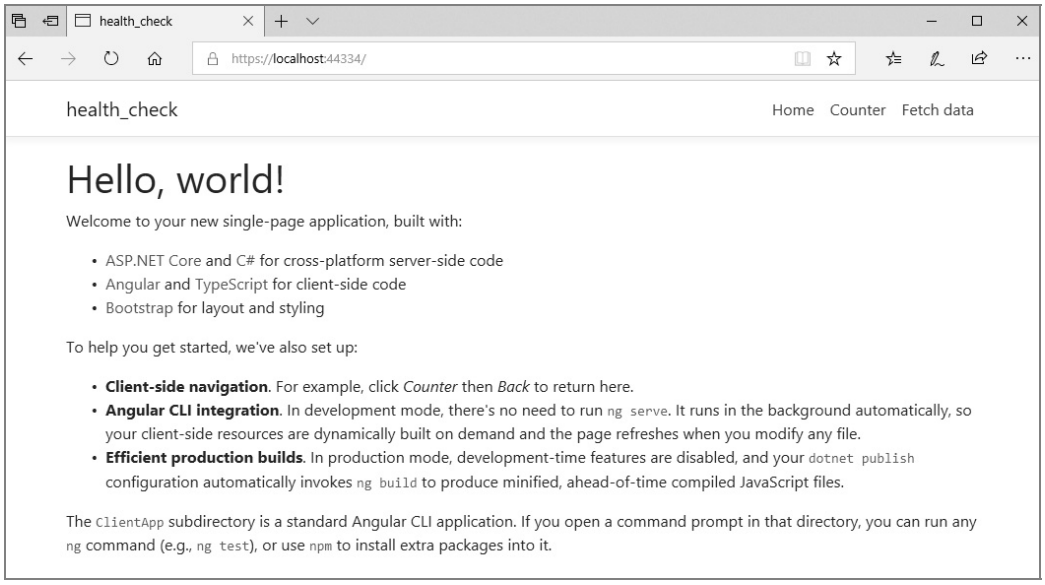

图 1-3　执行测试

这是一种很好的一致性检查，可确保开发系统已经被正确配置。如果看到示例 Angular SPA 正确运行，如图 1-3 所示，则开发系统没有问题；否则，可能缺少了某些东西，或者有冲突软件阻止 Visual

Studio 和/或底层的.NET Core 和 Angular CLI 正确编译项目。

为了修复这种问题，可以尝试下面的方法。

- 卸载/重新安装 Node.js，因为我们可能安装了过时的版本。
- 卸载/重新安装 Visual Studio 2019，因为当前的安装可能已经损坏。.NET Core SDK 应该已经包含在 Visual Studio 2019 中，不过，我们也可以尝试重新安装.NET Core SDK。

如果仍然失败，则可以尝试在干净环境(可以是物理系统或 VM)中安装 VS2019 和前面提到的包，以避免当前的操作系统配置导致的问题。

提示 如果这些方法都没有作用，那么我们能做的就是在.NET Core 社区论坛上寻求帮助，地址为 https://forums.asp.net/1255.aspx/1?ASP+NET+Core。

如果测试运行成功，则意味着示例应用程序能够工作，我们就可以准备接下来的工作了。

1.7 小结

到目前为止，一切顺利。我们创建了一个应用程序骨架。在继续介绍后面的内容之前，先来快速回顾在本章做了哪些工作，学到了哪些知识。

我们简单介绍了将会使用的平台(ASP.NET Core 和 Angular)，并说明了它们两个结合起来，在创建现代 Web 应用程序时所具有的潜力。我们回顾了这两个平台在过去 3 年间的发展变化，并总结了开发团队在重塑和改进框架时付出的努力。尽管总是有新的竞争对手出现，但我们仍然选择使用这两个框架，这是有原因的，这里的回顾对于理解我们的选择非常有帮助。

之后，我们讨论了 SPA、MPA 和 PWA 的区别，这些方法经过调整后，可用来创建现代的 Web 应用程序。还解释到，因为我们将使用.NET Core 和 Angular，所以将采用 SPA 方法，但也会实现大部分 PWA 功能，如服务工作线程和 Web 清单文件。为了创建符合现实的生产场景，我们还介绍了大部分常用的 SPA 特征，首先从技术角度进行介绍，然后转到典型的产品负责人的角度，解释了他们的期望。

最后，我们学习了如何设置开发环境。我们选择使用.NET Core SDK 提供的最新 Angular SPA 模板，因而采用了标准的 ASP.NET Core 方法。我们使用了.NET Core CLI 创建应用程序，然后在 Visual Studio 中进行测试，确保它能正确工作。

下一章将详细解释刚刚创建的示例应用程序，以理解.NET Core 后端和 Angular 前端如何执行各自的任务，以及它们结合起来能够实现什么样的功能。

1.8 推荐主题

敏捷开发、Scrum、极限编程、MVC 和 MVVM 架构模式、ASP.NET Core、.NET Core、Roslyn、CoreCLR、RyuJIT、单页面应用程序(SPA)、渐进式 Web 应用程序(PWA)、原生 Web 应用程序(NWA)、多页面应用程序(MPA)、NuGet、NPM、ECMAScript 6、JavaScript、TypeScript、webpack、SystemJS、RxJS、缓存控制、HTTP 头部、.NET 中间件、Angular Universal、服务器端渲染(SSR)、预先编译(AOT)、编译器、服务工作线程、Web 清单文件。