.NET 开发经典名著

ASP.NET Core 应用开发

[美] James Chambers
David Paquette
Simon Timms
著

杜伟　涂曙光　柴晓伟　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：01-2016-9913

图书在版编目(CIP)数据

ASP.NET Core 应用开发 / (美)詹姆斯 • 钱伯斯(James Chambers) 等著；杜伟，涂曙光，柴晓伟 译. —北京：清华大学出版社，2017
(.NET 开发经典名著)
书名原文：ASP.NET Core Application Development: Building an application in four sprints
ISBN 978-7-302-47990-1

Ⅰ. ①A… Ⅱ. ①詹… ②杜… ③涂… ④柴… Ⅲ. ①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(2017)第 209162 号

责任编辑：王 军 于 平
装帧设计：孔祥峰
责任校对：成凤进
责任印制：李红英

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：清华大学印刷厂
经 销：全国新华书店
开 本：185mm×260mm 印 张：23.75 字 数：623 千字
版 次：2017 年 9 月第 1 版 印 次：2017 年 9 月第 1 次印刷
印 数：1～3000
定 价：68.00 元

产品编号：074390-01

译 者 序

2002 年初，微软正式发布了.NET Framework 的第一个版本，将微软开发技术栈带入.NET 时代。伴随.NET Framework 面世的还有 ASP.NET。ASP.NET 虽然继续冠以 ASP 之名，但实际上与 ASP 存在巨大的技术差别，是新一代 Web 开发平台。译者还记得在当时的.NET 和新一代 Visual Studio.NET 发布会上，看到演讲者在 ASP.NET Web Forms 中将一个 DataGrid 控件拖曳到窗体上，然后将控件与数据源绑定，此后数据就出现在页面之上，译者当时不禁由衷地被其快速、简洁的开发模式所吸引。

到了 2009 年，为了适应 Web 开发技术的发展潮流，ASP.NET 家族中又正式添加了一个新的重要成员：ASP.NET MVC。ASP.NET MVC 从第一个版本开始，就开放了它的源代码，这预示着微软已经意识到，在 Web 开发领域，只有走开放之路，才能吸引足够多的 Web 开发工程师。

.NET Core 是微软在一个新的开发时代对.NET 技术栈进行进化的产物。在这个新的开发时代，开源、跨平台、云、移动已经成为举足轻重的几个关键字，不能主动拥抱这些关键字的开发技术无疑是没有未来的。.NET Core 宿主在 GitHub 上，以完全开源、社区协作的方式进行开发，它也不再局限于 Windows 操作系统，而是能够在 Windows/MacOS/Linux 等各个平台上运行，甚至为.NET Core 贡献代码的主力也正在逐渐地从微软员工变成社区的热心成员。

相对应的，ASP.NET Core 也就是 ASP.NET 技术的新一代平台，它同样继承了.NET Core 的几个重要特点：开源、跨平台、社区协作式开发。摆脱了对 Windows Server 的依赖，不但没有降低 ASP.NET Core 的能力，反而让它能够博采众长，具有了更好的发展潜力。在 ASP.NET Core 团队进行的性能测试中，它实现了每秒 115 万次的请求处理能力，相比于 ASP.NET 4.6，ASP.NET Core 取得了 2300%的性能提升！在第三方(如 TechEmpower)所进行的对比测试中，ASP.NET Core 同样展现出非常好的性能，面对 Node.js 这样的 Web 平台也不遑多让。

译者们平时都工作在 Web 开发的第一线，深深地知道最近几年 Web 开发领域最重要的发展几乎都集中在前端领域。面对如火如荼的前端技术，ASP.NET Core 明智地采取了支持、采用、不重复的策略，让 Web 开发人员能够舒心地在 ASP.NET Core 项目中去使用自己喜欢的前端框架。例如，ASP.NET Core 项目既允许开发人员在命令行中使用 npm 指令安装 npm package，也在 Visual Studio 中直接对 npm package 的管理提供一流的支持。ASP.NET Core 将自己当作整个 Web 开发社区中的一员，而不再像过去的 ASP.NET 时代那样去尝试建立一个试图自己向开发人员提供所有所需工具的“独立王国”，那样的策略在现在不但不合时宜，而且也不再可行。

本书以一个虚拟的项目为示例，向读者介绍了如何从头到尾创建一个 ASP.NET Core 应用程序。在翻译本书的过程中，译者感觉到本书中的这个示例不但能够很好地讲述一个

ASP.NET Core 应用程序中需要关注的方方面面，而且在很多地方还引导读者去了解如何设计出一个良好的 Web 应用程序架构。其中很多设计上的思路都非常符合当今云计算与微服务的潮流。另外，本书中的项目示例宿主在 GitHub 上，我们鼓励读者从 GitHub 代码库中将项目源代码 pull 到本地，就像参与一个实际的开源项目那样去阅读和测试这些源代码。

本书全部章节由杜伟、涂曙光、柴晓伟翻译，参与本次翻译的还有梁祝权、钟凤华、毛士之、张杉杉、张文旭，在此一并表示感谢！最后，让我们向读者稍微介绍一下自己。杜伟（微博账号：@Erucy），软件工程师与日本动漫迷，现在一家互联网创业公司担任技术总监，日常使用 JavaScript 和 Cordova 开发跨平台的 Web 和 Mobile 应用。涂曙光（微博账号：@kaneboy），软件工程师，现在一家金融行业公司担任架构师，日常使用 C#/C++ 开发证券交易系统，同时还在使用 JavaScript 开发一些 Web 应用程序。柴晓伟（微博账号：@WindieChai），软件工程师，现在一家大型招聘网站担任技术执行总监，日常使用 JavaScript、Node 和 C#开发各种 Web 前后端以及 Mobile 应用。我们都是微软最有价值专家，并已合作翻译过多本技术书籍。

祝大家阅读愉快！

译　者

前　言

ASP.NET Core MVC 是微软面向 .NET 开发者的最新的 Web 框架，它是如今广为人知的 ASP.NET MVC 框架的下一代，并致力于开启跨平台开发、部署的能力。它广泛利用了各种各样的开源库，当然，它本身也是开源的。ASP.NET Core MVC 帮助开发者独立思考关于业务逻辑、路由、服务以及视图的实现，并提供了一套新的配置和扩展系统。它使用了 C#编程语言，以及 Razor 视图引擎。不管你是一个经验丰富的.NET 开发者，还是一个新手，使用 ASP.NET Core MVC 来构建项目应该都是不错的开端。

本书展示了一个重构项目的最初几个迭代版本，该项目是由一个虚构公司 Alpine Ski House 的团队重新开发的。每一章都介绍了一些在开发过程中团队所遇到的挑战，以及他们是如何克服这些难关的。除了每章前面的一个小故事之外，本书深入介绍了从 ASP.NET Core MVC 的特性，到构建、维护和部署应用程序所使用的相关工具。

除故事片段和关于 ASP.NET Core MVC 的技术内容之外，本书还讨论了新版的 Entity Framework、包管理系统，以及其他在 Web 开发领域中流行的周边工具。除相关技术内容的介绍外，本书也附带了一个项目——正是 Alpine Ski House 的开发者们构建的那个项目。

本书读者对象

本书以一个程序员的视角，贯穿了使用 ASP.NET Core 开发一个全新应用程序，并将其发布到互联网上的所有必经步骤。不过仍然有相当多的程序员还没有接触过 Web 开发，或者还停留在 Webforms 时代，很少接触到如今全新的各种工具。本书将帮助他们掌握这些技巧、树立信心来跟上脚步，使用新兴的框架来构建现代应用程序。本书将帮助读者探索应用程序的架构，部署并构建适用于云端的应用程序。

阅读本书的前提条件

本书的读者需要拥有中高级的程序开发能力、熟练掌握 C#、拥有 Web 开发的背景知识，并了解 Visual Studio 的基本功能。如果了解上一个版本的 MVC 会更有帮助，不过它不是必需的。熟悉使用命令行界面进行工作也是个加分项。在阅读本书后，你将有能力构建一个真实的、由关系型数据库驱动的应用程序，并可以将其部署在基于云端的基础架构。

本书可能不适合……

如果你是经验丰富的高级 ASP.NET MVC 开发者，始终在密切关注甚至参与了 ASP.NET Core MVC 开发的话，那么本书可能不太适合你。

本书结构安排

本书创新性地从开发者的视角出发，贯穿了一个完整应用程序开发的各个迭代环节。书中不仅包含了技术内容，也涵盖了如何从错误中吸取教训、根据用户的反馈进行调整，从零开始，逐步构建出一个完整的产品。

本书分为如下四个部分：

- 第Ⅰ部分：“Alpine Ski House”。介绍了一些背景知识，构建了示例应用程序，并引入了贯穿本书的所有虚构角色。
- 第Ⅱ部分：“迭代回顾：千里之行”。关注能够让应用程序运行起来的必要特性，对构建流水线进行配置，使配置实时生效，从而使整个团队都关注到项目进度。
- 第Ⅲ部分：“迭代回顾：激流勇进”。关注一些核心的特性能够在示例程序的基础上加上所需的业务逻辑。该部分中，我们使用了 Entity Framework Core 来进行数据访问、使用 Razor 创建视图，还介绍了配置、日志、安全、用户管理，以及依赖注入。
- 第Ⅳ部分：“迭代回顾：最后冲刺”。介绍了 JavaScript、依赖管理，以及在前文介绍的基础上进行构架的内容。

在附录中包含了一些重要主题，比如测试、重构和扩展能力。

寻找本书最佳切入点

本书中的不同章节涵盖了 ASP.NET Core 框架中相关的各种技术。根据你的需要，以及你对微软技术栈的掌握程度，你可能会需要重点关注本书中某些特定的领域。可以通过如下表格来决定你阅读本书的最佳切入点：

如果你……	阅读建议
是 ASP.NET Core 开发的新手，或者已经是 ASP.NET Core 的开发者	关注第Ⅰ、第Ⅱ和第Ⅲ部分，或者按照顺序阅读整本书
熟悉之前版本的 ASP.NET	如果你只需要关注核心内容，可以略过前两章，并通读本书中的其他章节，以了解新的技术
对客户端开发感兴趣	阅读第Ⅳ部分的第 15、16、17 章，略读第 20 章中关于 JavaScript 服务的介绍
对跨平台开发感兴趣	整本书的内容都可应用在跨平台开发中，不过第 8、9 章的主题特别涉及了该内容

本书中的大部分章节都包括了动手示例，通过它们你可以练习刚刚学到的内容。不论你关注的是哪部分内容，请在你的系统中下载并安装示例应用程序。

本书的约定和特色

本书在介绍内容时使用了一些约定，了解它们可以让阅读变得更易理解。

- 本书中的代码是面向 C# 程序员的，使用的语法涵盖了 HTML、CSS、SCSS 和 Razor。

- 在两个按键之间使用加号(+)表示同时按下两个键。比如“按下 Alt + Tab”的意思是你需要在按住 Alt 键的同时，按下 Tab 键。
- 在两个或多个菜单项之间的竖线符号(比如文件 | 关闭)，意思是你需要先选择第一个菜单或菜单项，然后再选择下一个，以此类推。

系统要求

为了运行本书的示例应用程序，你需要如下的软、硬件配置：

- .NET Core 1.0 及以上版本，可以跨平台安装，来自 https://dot.net。
- 选择你的代码编辑器。我们使用的是 Windows 上的 Visual Studio 2015(任何一个版本都可以)及以上版本，或者也可以使用 Windows / Mac / Ubuntu Linux 上的 Visual Studio Code。
- SQL Server LocalDB(包含在 Windows 中的 Visual Studio 2015 及以上版本中)。对于 Linux 或者 Mac 的用户，你需要访问一个位于其他 Windows 机器或者 MicrosoftAzure 上的 SQLServer 数据库。
- 电脑的处理器至少是 1.6GHz。
- 至少 1GB 内存。
- 4GB 剩余磁盘空间。
- 互联网连接(用于下载软件和示例项目)。

根据你的 Windows 配置，可能需要本地管理员权限来安装或配置 Visual Studio 2015。

下载：示例项目

本书中大部分章节都包含了来自这个示例项目中的代码片段，该项目可以在 GitHub 上找到：

```
https://github.com/AspNetMonsters/AlpineSkiHouse
```

按照 GitHub 上的指示下载并运行该示例代码。

注意：
除了示例项目之外，你的系统还需要安装.NET Core 1.0 及以上的版本。

勘误表、更新及内容支持

我们尽了最大的努力来确保本书内容的正确性。你可以访问本书的更新(以勘误表的形式记录了相关的错误和纠正内容)：

```
https://aka.ms/ASPCoreAppDev/errata
```

如果你发现了一个尚未列出的错误，请使用该页面提交错误。

可以在网站 https://aka.ms/ASPCoreAppDev/downloads 中下载所有的示例代码以及完整的应用程序。

请注意上述地址并不提供对微软的软、硬件产品的支持。如果需要这些支持，请访问 http://support.microsoft.com。

期待您的反馈

在 MS Press，您的满意是我们首要的目标，您的反馈是我们最有价值的资产。请通过如下地址告诉我们你对本书的看法：

```
http://aka.ms/tellpress
```

我们知道你公务繁忙，所以只保留了非常简短的几个问题。你的答案将会直接发送给 MS Press 的编辑(不需要提供你的个人信息)。非常感谢您的反馈！

保持关注

让我们持续保持交流，我们的 Twitter 是 http://twitter.com/MicrosoftPress。

目　录

第 I 部分

Alpine Ski House

- **第 1 章** 一路走来
- **第 2 章** 影响者
- **第 3 章** 模型、视图和控制器
- **第 4 章** 定义项目范围
- **第 5 章** 生成
- **第 6 章** 部署

以下内容介绍了本书中涵盖的虚构情节的背景知识，其中包含创建 Alpine Ski House(“滑雪之家”)应用程序的虚构角色。

即使最狂热的滑雪者也不得不承认：滑雪季已经渐入尾声。虽说比不上记忆中最好的滑雪季，但刚过去的这个倒也算不上是最差的。从各个角度看，这个滑雪季都有点儿平平淡淡。二月下旬的一次停电故障，终于让曾经排练过许久的应急计划有了登场的机会。当地的新闻也报道过几次儿童被困缆车的事故，但都因为并不恶劣的气候而最终有惊无险。搞搞赠票促销，就可以让滑雪者们和雪地摩托车手们络绎不绝了。

每年开春，滑雪场的长期雇员们将会重聚，而临时雇员们则回到他们要去度夏的地方。在长期雇员中间一直有传闻说，有一半的临时雇员从工作的滑雪场一下山，就会被移民局抓起来并遣送回澳大利亚。丹妮简直没法想象，为什么他们不愿被送回澳大利亚呢？不管怎么说，澳大利亚都比这个除了冬天之外都死气沉沉的山间小城要好得多呀。

现在就开始为明年做计划未免有些太早，丹妮决定在滑雪场重新变得忙碌之前，先好好利用一下这一两个月的空余时间。过去十年来，她一直都是滑雪之家唯一的一个程序员。大部分时间里，她的工作都是保证这个古老的电脑系统能正常运行，并为第二年的滑雪活动做一些细微的调整。这算不上是世界上最让人开心的工作，但在冬天，雇员们可以在好天气的日子里，开小差去滑雪场上痛快地滑上几个小时，这种福利想想还是挺让人开心的。

打开被雇员们亲切地称为“老巢”的 Alpine Ski House 底层办公室的门，丹妮吃惊地发现大家都在低声聊着什么。一些平时在这个时间很少会出现在办公室的人，都三三两两地聚集在一起交头接耳。一头雾水的丹妮扔下包，端起一杯咖啡，开始左寻右找可以加入的人群。她看到 IT 部的其他雇员都围着有些发福的提姆，他是 IT 部的头儿，也是丹妮的老板。于是

丹妮一头扎进了这堆人里面。

“丹妮！你怎么看？要我说，这下可热闹了。”提姆连珠炮似地说道。

“什么怎么看？”丹妮问道。

“你还不知道呢？”说话的是阿尔让，“公司刚刚收购了发达谷和柏丽山庄。管理层正在整合运营部门的员工，我们就要失业了！”

阿尔让提到的那两家滑雪场离 Alpine Ski House 并不远。其中的发达谷规模比较小，一共才有三架缆车，却拥有一群忠诚的滑雪粉丝。对于想要在冬季躲开一大群吵吵嚷嚷的游客的本地人来说，发达谷是最佳之选。与发达谷相比，柏丽山庄则完全是另外一番天地。它是一个横跨了三座山头的巨型滑雪胜地，拥有数不清的缆车以及足够容纳两倍小城人口的山顶宾馆。每个周末，都会有乐队去到柏丽山庄演出，你不时会在人群中看到诸如 Scott Gu 和 John Skeet 这样的著名人物，和其他人混在一起寻欢作乐。

“得了，阿尔让，”提姆说道，“没人说过下岗裁员之类的话。遇到这种情况，管理层就会想要将这几家的系统尽快整合在一起，所以 IT 部门的工作压力一般都会不减反增呢。我们只不过必须等等看，具体的计划是什么。”

丹妮感觉自己有点站不稳。她还有几年就可以退休了，现在可不是再去找另一份工作的好时候。再说，在这样一个山间小城，又有多少程序员职位呢？“别傻了，”她对自己说道，“现状不明，现在就去想要怎么搬回到一个大城市工作对事情毫无帮助。过几个星期，情况就会明朗起来。”

结果，她根本不需要等几个星期。

就在午餐时间，丹妮刚刚消灭完一份蒲公英拌核桃沙拉和一份甜香薯片，提姆就走过来敲了敲她的小隔间。

“我们正准备在大会议室开会。看起来发达谷和柏丽山庄的程序员们也都在。”

咽下剩下的羊奶，丹妮抓起一支笔和一个黄色大本子，匆忙赶到了会议室。随身带着的本子和笔只是用来展现姿态用的；她好久都没有在开会时真的用它们来记点什么了。相比用一个本子做做提前规划，直接和一个活生生的人面对面地在一个小房间里面讨论事情要容易得多。不过考虑到裁员的传闻，现在还是需要用它们来给大家留下一个好印象。

除了用来举行员工聚餐，平时大会议室很少被使用，原因也很简单，它太大了，根本没有那么多的人能够装满它。但是今天是例外，虽然说不上拥挤，但它确实已经满满当当了。五个看起来像嬉皮士的年轻人坐在会议桌的一端，正在吸着里面有着各式各样奇怪东西的沙冰。丹妮想知道怎么会有人为了这样的目的而想要去进口红毛果，这未免太不环保了。不过不管怎么说，这次总比之前那次她想要破开一个榴莲的体验要好得多，那次体验最终以她将那个带刺的水果扔出窗外而悲剧收场。

会议室里面和那几个嬉皮士一看就不是一拨的另外一群人，瞧着就像是大城市里面的那种“西装革履男”。他们的样子就像是来参加一门高尔夫课程一般，一个个显得皮肤黝黑且神色轻松。

提姆等到每个人都就坐后，开始了自己的发言，“诸位，好消息，你们都将是这个未来之队中的一员。只要你现在在这个房间里面，那么你可以尽管放轻松，因为你的工作已经保住了。我相信你们都还有一些问题想问，如果有的话，散会后可以马上来找我面谈。”

“管理层要求我在合并后保留尽可能多的程序员，因为他们有一些希望我们能够马上开始着手执行的新想法。在未来几年之内，我们将要更新现存的所有客户系统以服务整个滑雪

场。高层意识到这是一件不折不扣的大事，他们中的一些人还读了几本 CIO 杂志，并从中学到了敏捷和微服务。我可以向你保证，以后这些见鬼的杂志在被送到高层手里之前就都会被烧掉，但现在的问题是，我们已经掉到坑里了。”

提姆从来都对管理层的伟大新构想不太感兴趣。他就像是一个应急刹车装置，不断地“刹住”管理层的疯狂点子。他继续说道：“管理层想要实现的第一个功能，是让用户可以在线购买缆车票。他们说现在已经是 21 世纪的第 16 个年头了，我们早就应该提供这个功能，这个世界上除了我们之外的其他任何一个滑雪场都已经有了这个功能。”提姆看起来有点对管理层的说辞感到有一些恼火；当他和管理层之间讨论这些问题时，场面一定“愉快”极了。

“管理层想要在一个月之后就看到一个原型。如果我们能够展现出我们正在有所进展，我还能再拖上一个星期。”

一个月之后！丹妮感到一阵恍惚。一个月也就刚刚够丹妮在脑子里把问题好好理清。她望着那几个嬉皮士程序员，指望看到他们和自己一样脸色苍白的样子。但是那些吸着小麦草汁的家伙，正在快乐地点着头。

提姆看起来正要做一个总结陈词，他的样子像是要准备从他的肥皂箱上走下来。“瞧，伙计们，我们需要通过这个项目来让管理层认可我们。我会为你们清除前进道路上的每一块绊脚石。你们可以使用任何你们觉得最好的技术，购买任何你们需要的工具。我相信你们一定会成功的。”

第 1 章

一 路 走 来

提姆结束了他的发言后，和那些西装革履的人一起走出了会议室，将丹妮和五个嬉皮士程序员独自留在了房间里面。显然，房间里的人中间，没有人被任命为新开发团队的负责人。很可能提姆也读过一点他所唾弃的那些 CIO 杂志，于是觉得一个最好的团队应该是一个自组织的团队。丹妮想着如果偷偷塞给邮递员一张二十块的钞票，让他以后都把那本杂志假装“遗失”掉，要比整年都去买止疼药划算得多。

嬉皮士们依次介绍了他们自己，他们是阿德里安(Adrian)、切斯特(Chester)、坎迪斯(Candice)和“双马”：马可(Marc)和马克(Mark)。他们都来自柏丽山庄，山庄几周前刚刚决定停止使用外包开发团队，转而招募了一个内部的全职团队。嬉皮士们以前都在硅谷的一家创业公司工作，柏丽山庄将他们团队整个招聘过来，对这一点，丹妮也觉得确实是一个好主意。他们团队被山庄买下来后，还没来得及开始做一个项目。他们看起来都是既友好又兴奋。

“管理层让我们自己选择自己喜欢的技术，这让我很喜欢，”马克说道。“这表示他们足够信任我们，不会对我们指手画脚。”

“挑选合适的技术一直都是我在一个项目中最喜欢的部分。”坎迪斯说道。

“现在有着大把的好技术可供选择：Phoenix、Revel、Express 甚至 Meteor。你最喜欢的是哪个，丹妮？”阿德里安问道。

Phoenix？Express？这些 Web 框架的名字感觉就像是他们的作者在去过一家中餐馆之后，就顺便用中餐馆的名字为他们的作品命名(注: Phoenix 和 Express 都是常见的美国中餐馆名称中所使用的单词，比如美国著名中餐馆连锁 Panda Express)。丹妮从来没有听说过它们中间的任何一个。她一直都是一名 ASP.NET 程序员。来 Alpine Ski House 以前，她一直都是在写 WinForms 应用程序，从去年开始，她才开始使用 ASP.NET MVC。她决定豁出去了，“我一般都使用 ASP.NET。”她回答道。

“哦耶!”马可喊道。貌似他和马克的不同，就是他总是会用 90 分贝的声调发表意见。“微软的团队正在用 ASP.NET Core 做一些激动人心的事情。你们应该看看他们使用 ASP.NET Core 跑出来的性能吞吐量！简直碉堡了!”

丹妮不太清楚马可说的“碉堡”到底是什么意思。不过听起来这不是一个贬义词，所以她点头表示赞同。

“呃，”坎迪斯说道，“我没有料到 ASP.NET 会出现在我们的候选技术名单里面。”

“它又新又酷，”马可说道。“丹妮，你使用 ASP.NET 已经有一阵子了。你能给我们讲讲它是如何发展到这一步的吗？”

Web 开发不是一个新鲜事儿。在最开始，Web 上还只有包含导航超链接的静态内容。这种实现方式的限制很快就变得明显了起来。要让每个页面都保持统一的风格很麻烦，每个人都因为看到一模一样的内容而感到很乏味。

Web 服务器于是开始支持根据用户输入信息的不同，而在页面上显示不同的内容。许多技术都昙花一现，比如 Server Side Includes、Common Gateway Interface(CGI)脚本之类的。一般被简称为 ASP 的 Active Server Pages 技术则持续使用了将近二十年时间，只不过这二十年间，它经历了翻天覆地的变化。我们现在又再次处在了使用 ASP 这个名字的技术进行大幅更新的当口。在仔细了解这个技术的最新发展趋势之前，让我们先看看它是如何一路走来的。

1.1　Active Server Pages(ASP)

在 1996 年，Internet Information Services 发布了第三版，这个版本支持 ASP 的第一个版本。ASP 构建于活动脚本(Active Scripting)技术之上。活动脚本不仅限于在 ASP 中被使用，还被应用于 Internet Explorer 浏览器和 Windows Script Host。

ASP 允许页面上包含使用另一种语言编写的脚本，当页面被浏览时，这些脚本就会被执行。理论上，通过活动脚本中的 Component Object Model(COM)集成技术，ASP 可以支持大多数编程语言。JScript 和 VBScript 是那时在 ASP 领域最被微软支持的两种互相竞争的语言。在 20 世纪 90 年代中期，其他类似 Perl 那样的语言也很流行。由于 Perl 支持 Linux，并且还可以通过 Apache CGI 网关来运行，因此它允许一定程度上的跨平台移植。Perl 是当时在早期版本 Apache 上创建可交互式 Web 应用程序的主流编程语言之一，并导致了 Apache 的 NCSA HTTPd 服务器的诞生。

下面查看一个使用 VBScript 作为脚本语言的示例代码文件。我们会在输出的 HTML 中使用最新的 HTML5 标签。没有理由不可以使用 ASP 创建现代的应用程序。

```
<%@ Language= "VBScript" %>

<html>
  <head>
  <title>Example 1</title>
</head>
<body>
  <header>
    <h1>Welcome to my Home Page</h1>
  </header>
    <section id="main">
      <%
        dim strDate
        dim strTime

        'Get the date and time.
```

```
        strDate = Date()
        strTime = Time()

        'Display a different greeting depending on the time of day
        If "AM" = Right(strTime, 2) Then
        Response.Write "<span>Good Morning!</span>"
      Else
        Response.Write "<span>Good Afternoon!</span>"
      End If
    %>
    Today's date is <%=strDate %> and the time <%=strTime%>

    </section>
  </body>
</html>
```

在这个示例中，你可以看到将数据放到输出中的不同方式。第一种方式是通过 Response.Write 来直接访问响应流，这种方式用来输出“上午好”或者“下午好”的问候信息。第二种方式是使用<%= %>指令，这是一种输出内容到响应流的简写方式。这种方式和其他脚本语言，比如 PHP 中所使用的方式非常类似。除非 HTML 标签中包含特别的指示符，否则编译器就会忽略它们。包含特别指示符的标签都以<%开头。<%@ 指示符是一个处理指示符，用来提供 ASP 编译器处理页面所需知道的信息。上面的示例中，使用了一个指示符来设置页面上主要使用的编程语言。普通的<% 指示符用来标识一个简单的代码块。

和 PHP 不同，ASP 可以通过 COM 实现库的共享。这个功能使得 ASP 代码可以访问原生组件，以实现高性能或是访问原生操作系统的功能。这还允许你编译组件库，来提高应用程序的结构化和鼓励代码复用。

在 20 世纪 90 年代后期，微软开始开发它的下一代开发平台。当时这个平台被称为 Next Generation Windows Services，或 NGWS。2000 年的年底，这个平台改名为.NET Framework。.NET Framework 重新打造了流行的 Visual Basic 语言。这次重新打造的结果是将面向对象编程的能力加入到这个语言中，这无疑是一个巨大的变化。从很多角度看，VB.NET 成为一门完全不同的语言。.NET 包含了一个提供许多通用功能的、构架良好的基础类库。这个基础类库(Base Class Library，BCL)受到了 Java 语言基础类库的很大影响。

.NET 还引入了一门全新的名为 C#的语言。深受 Java 影响，C#为具有 C 或 Java 语言背景的工程师提供了更类似 C 语言的语法风格。微软非常强力地推广 C#语言，这导致它成为.NET 平台上最流行的语言。

与新的 C#和 Visual Basic .NET 语言一起推出的，是一个全新版本的 ASP。这个版本被称为 ASP.NET。

1.2　ASP.NET

在 20 世纪 90 年代末期，World Wide Web 就已经开始显示出它的巨大潜力。遗憾的是，微

软有一个麻烦。他们已经花费了数年来打造通过拖拉编辑器来创建桌面应用程序的开发工具。开发人员已经习惯了使用这种方式开发应用程序，大公司们也不想花钱去重新培训他们的工程师。控件的交互都是通过一个事件处理程序模型来进行。窗体上的一个按钮控件会使用一个事件处理程序来执行某些后台操作，然后更新用户界面(User Interface，UI)。开发者社区已经很适应这种开发模型，同时也觉得这种模型的开发效率很高。直到今天，也没有哪个工具的开发效率比得上 Visual Basic 和 WinForms 中的“所见即所得”(What You See Is What You Get，WYSIWYG)代码生成器。这个工具所生成的界面虽然称不上最赏心悦目，但是如果你在自己的产品中只是需要在灰色背景上创建一些灰色框，则 Visual Basic 6 或 WinForms 通常是很好的选择。

ASP.NET Web Forms 是将这种高效开发工具带入 Web 开发领域的一次尝试。它采用一种大家熟知的方式，通过拖拉功能向一个窗格上添加控件，然后使用编译后的服务器端代码来与控件进行交互。

遗憾的是，与控件的这种交互方式需要将数据发送回服务器，而这种回发模型忽略了 Web 应用程序与桌面应用程序之间的一个基础性区别，那就是每次将数据发回到服务器上并让服务器执行一个操作，都需要通过互联网进行一次 HTTP 请求和响应。尽管有这个缺陷，ASP.NET Web Forms 仍然成为一个非常成功的产品，使用它创建了企业内部和互联网上的无数个 Web 网站。

Web Forms 中的页面通常包含两部分：一个视图和其背后的代码。视图(或显示)部分使用一种混合了 HTML 和 Visual Basic 或 C#代码的特别语法编写而成。这些文件以.aspx 作为扩展名，并被传递给一个 ASP 编译器进行处理。关于这些文件中应该包含哪些内容有着一些争论。有些人觉得将所有逻辑都放在文件中是可以接受的，而另外一些人则坚持只在其中包含与显示相关的逻辑。这两种风格都有一些成功的项目作为各自的论据。后台代码的部分都被编译进一个 DLL，这也意味着在线即时修改页面是不可能的。.aspx 文件是在被用户浏览时进行编译，所以虽说非常不鼓励这么做，但确实是可以在服务器上直接使用一个文本编辑器，对它进行在线即时修改。

每个页面都会实现为一个 HTML 表单。这让 Web Forms 很容易就能通过表单的回发，捕获到状态的变化并对这些变化进行响应。如果你曾经使用过一个只要将鼠标从一个数据输入框中移走页面就会整个刷新一次的网站，它很可能就是一个 Web Forms 应用程序正在将页面的状态发送回服务器，这样服务器才能够根据用户输入的信息来决定应该如何修改页面上的内容。这不是一种很好的交互模型，通过使用只会将页面的部分数据发送回服务器的 AJAX 和 Update Panel 控件，可以对这种模型进行一些优化。

为了在整个网站中维护一致的外观，Web Forms 使用了母版页技术。这些母版页包含外观一致的组件，开发者再将每个被执行页面中的不同内容插入母版页里面。这种模式减少了维护网站外观界面的工作量，避免了在不同的页面上单独地修改外观。一个应用程序可以有多个不同的母版页，这些母版页甚至可以互相嵌套。你可以将一个母版页应用于未登录用户，再将另一个母版页应用于已登录用户。母版页可以相互嵌套的特性也会相应地出现一些有趣的应用场景，比如就像图 1-1 所示，页面可以像套娃玩具那样层层嵌套，每一层母版页嵌套都向页面上增加一些内容。

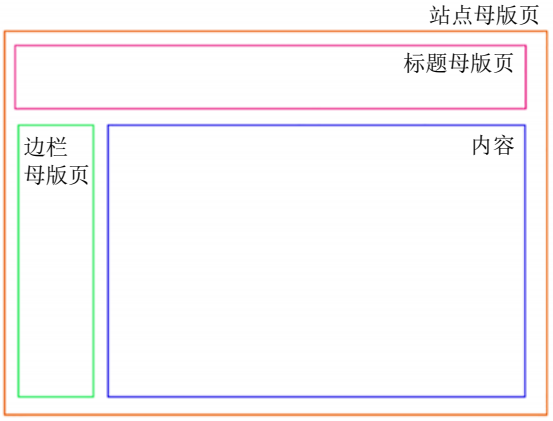

图 1-1　嵌套的母版页

用户控件是另一种组件重用机制。用户控件通常被定义在一个.ascx 文件和一个与之配套的 VB 或 C#后台代码文件中，一个用户控件就像是一个可以被放置到其他页面里面的微型.aspx 页面。例如，如果你在网站中的各个地方都需要一个选择汽车品牌的下拉框，你要做的就是创建一个封装了这个功能的用户控件。这个控件可以被嵌入需要到它的各个地方。一个用户控件的逻辑和显示代码都被集中封装在用户控件里面，这样就避免了在不同的地方再去重复编写同样的功能。

创建受欢迎的用户控件成为许多创业公司的商业模式。程序员自己写一个 Table 控件通常是一件乏味的事情，于是一大帮控件提供商就开始比拼着争相创建功能众多的 Table 控件。不管你需要什么样的控件，总会有一家提供商愿意卖给你一大堆满足你需要的控件。在你学会了使用这些复杂的控件之后，你的生产力将有显著的提升。

虽然 Web Forms 的开发效率较高，但是它也许无法带给用户所期望的现代 Web 体验。为 Web 编程创建一个类似桌面开发的抽象层带来一个问题，那就是 Web 并不是桌面。为了实现这个抽象层所引入的许多技术手段，使得 Web 应用程序的交互显得很笨重。这种交互模式对于 Intranet 网站来说可能是可以接受的，但是很多基于 Web Forms 的网站却并不仅仅是 Intranet 网站。从很多方面来看，Web Forms 都像是在一个方孔里面塞入一根圆柱(如图 1-2 所示)，虽然勉强能塞进去，但周围却留下了很多空洞，总体显得格格不入。

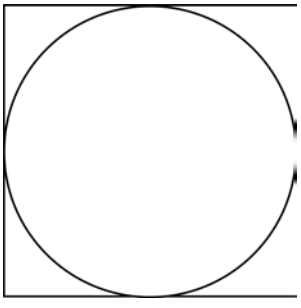

图 1-2　Web Forms 技术就像是在方孔中塞入圆柱，显得格格不入

单页应用程序(single page application，SPA)是位于 Web 客户端的页面，当用户进行导航浏览时，并不会进行整页刷新。单页应用程序的典型例子包括 Gmail(也许它是第一个流行的 SPA

应用程序)、Trello 和 Twitter。当浏览这些网站时，你也许会注意到页面只会重新加载有变化的内容，而不会重新刷新和下载整个页面。这种网站的用户体验要比传统的网站好很多，因为用户在浏览时不会觉得被页面刷新所打断。虽说只用 Web Forms 也能创建一个单页应用程序，但是会困难重重且非常耗时。

一个 Web Form 页面的状态被存储在一个名为 ViewState 的容器内。取决于网站如何实现 ViewState，编码后的 ViewState 数据可以被放置到每个页面上，然后当每次客户端向服务器端发起请求时，就跟随请求在客户端和服务器端之间传递。可以想象，ViewState 会显著地增加传输的页面的大小，有时其中甚至会包含数 MB 的内容。对于互联网网站，页面传输的速度会对从用户满意度到网站搜索排名等方方面面造成影响。网站的页面传输速度在移动平台上是非常关键的，通过移动网络加载一个很大的页面，足够赶走潜在的用户了。

为了解决 Web Forms 的各种缺点，微软在 2007 年发布了 ASP.NET MVC。

1.3　ASP.NET MVC

微软团队总是会在保持一个产品的向后兼容性上付出极大的努力。虽然 ASP.NET MVC 在很多方面都要好于 Web Forms，但是它从来没有尝试过完全取代 Web Forms，而是作为创建 Web 应用程序的第二个选择。为了避免引发争论，应当说，Web Forms 在某些方面仍有强过 ASP.NET MVC 之处。ASP.NET MVC 比 Web Forms 更接近底层，它移除了那些让 Web Forms 变得庞大且易于使用的抽象层。你不需要从这两者中选一个，因为你可以在一个项目里面同时使用 MVC 和 Web Forms。如果要将一个现有的应用程序从 Web Forms 迁移到 MVC，这种混合使用两者的方式就非常有用，开发者可以逐个页面地进行迁移。

MVC 相比 Web Forms 修改掉的第一个概念，就是每个页面都有一个后台代码文件这种模型。使用 Web Forms 来创建拥有良好概念分离的应用程序并非不可能。但是，在 Web Forms 应用程序中，很容易就会不正确地实现每个层级的职责隔离，而将本应分隔在不同层级中的代码混在一块。ASP.NET MVC 构建于 Model-View-Controller 模式之上，它鼓励我们将 UI 的功能隔离在一个视图里面，将操作行为隔离在一个控制器里面，将存储和通信的功能隔离在一个模型里面。这种内置的隔离性，使得代码的可测试性和可重用性都更佳。而这还只是 ASP.NET MVC 为了确保开发者成功实施项目所提供的诸多工具之一。

当一个客户端请求到达一个 ASP.NET MVC 网站时，这个请求首先会被路由表所处理，路由表会将请求转发给一个文件，或是转发给一个控制器中的一段代码。这个机制消除了对磁盘上的.aspx 文件的依赖。路由表开启了一系列有趣的可能性。它可以创建自定义的 URL，让用户根据 URL 就能了解当前页面上会有什么样的内容，而这也很有利于搜索引擎优化。如果你有一个网站是同时给多个客户使用的，那么可以很容易让 URL 看起来像这样：

```
https://site.com/{ComapnyName}/User/Add
```

这可比下面这种形式的 URL 友好多了：

```
https://site.com/User/Add?companyId=15
```

你可以为一个终结点添加多个路由，以此来消除许多重复代码。

路由通常都将一个客户端请求映射到一个控制器之上。控制器是继承自一个 controller 基类的简单对象。在控制器里面，返回一个 ActionResult 对象的 public 方法可以用来处理路由。这些操作方法可以像一个普通.NET方法那样接收普通的参数类型。MVC 使用模型绑定来将HTTP请求里面所能找到的参数转换成.NET 类型。模型绑定异常强大，大大简化了开发的工作量。对于复杂的对象，还可以创建自定义的模型绑定器来进行映射；然而，在大部分情况下，默认的映射器就已经可以完成所有所需的工作了。

MVC 框架使用如下方式来选择控制器里面所定义的方法：一个使用控制器名称的函数、控制器中的方法以及在方法上定义的特性修饰。例如，在一个控制器里面，可以定义两个通过方法签名进行区分的 Add 方法。两个方法分别定义了[HttpGet]和[HttpPost]特性。

```
[HttpGet]
public ActionResult Add(){
//perform some operation
return View();
}
[HttpPost]
public ActionResult Add(AddModel model){
   //validate model
   //save
   //redirect
}
```

一种常用的范式，就是使用 GET 操作返回一个表单页面，使用 POST 操作接收表单的内容并存储这些内容，再重定向到其他操作。操作可以返回各种不同的 ActionResult。你可以返回一个重定向结果来将用户重定向到另外一个页面，或是返回一个文件流结果来将一个文件的内容返回给用户。JSON 结果会将 mimetype 设置为 application/json，并将操作的返回值序列化成 JSON 字符串后返回给客户端。视图结果则会将操作的返回值传递给视图引擎，然后视图引擎负责将返回值转换成 HTML 内容，最后显示给最终用户。

视图(也就是 MVC 里面的 V)是框架中的展现层，它执行的操作类似于以前一个.aspx 文件所能完成的。MVC 中的视图引擎是可替换的，以允许开发者使用不同的视图绑定语法。早期版本的 MVC 使用了和 Web Forms 相同的视图引擎，但是最新发布的 Razor 已经成为 MVC 的默认视图引擎。Razor 是一个更具有现代感的视图引擎，它能够和 HTML 更无缝地整合在一起。Razor 的设计目标是使之更易于学习、开发效率更高。这些目标无疑已经成功地实现，Razor 的使用体验要比之前的 WebView 语法好得多。

Razor 语法比之前的视图引擎更接近纯粹的 HTML。如果在一个公司里面，是由 Web 设计人员负责创建实际显示的 HTML，那么使用 Razor 要比使用 Web Forms 引擎更方便。

控制器有多种方式将数据传递给视图。第一种方式是将数据值添加到 ViewBag 或 ViewData 中，这是两个可以在视图和控制器里面都能访问到的集合。ViewData 是一个 string-object 的键值字典，ViewBag 是一个可以很方便地添加属性的动态对象。显然，这两个集合都是弱类型的，视图要使用里面的值就需要进行转型，而转型则有可能导致运行时错误。

第二种方式，也是通常来说更好的方式，是在视图和控制器之间使用强类型模型进行通信。模型可以对其包含的数据的类型和名称进行强约束。ViewBag 和 ViewData 可能引发的运行时错

误会在第二种方式的编译时就暴露出来，这绝对是一件好事。

一个模型就是一个普通的包含了一组字段的 CLR 对象(plain old CLR Object，POCO)。MVC 为模型提供了一些进行数据校验的功能。使用 ComponentModel 命名空间中的修饰特性(attribute)，模型对象的字段可以将自己标记为必需、不允许为 null，或是限制在某个范围内等。如果需要进行更复杂的校验，比如使用数据库中的数据进行检查，或是需要调用一个第三方的校验服务，那么还可以编写自定义的校验器。在控制器中的操作被执行时，通过简单地检查 ModelState 是否合法，就会触发数据校验。

对 ASP.NET MVC 早期版本的一个抱怨，是觉得它在开发效率上比不上 Web Forms。Web Forms 中强大的用户控件，就不存在于 MVC 中。MVC 后来通过引入编辑器和视图模板弥补了这个缺陷。这个强大的功能让你可以将一个模板赋予一个模型中的字段。当 Razor 遇到一个指示符指示说要为字段显示一个编辑器(就像下面所展示的那样)，它就会检查模型，看是不是通过一个 HTML 助手或者局部视图为表单中的这个字段定义特别的编辑器或视图：

```
@Html.DisplayFor(model =>model.StartDate)
@Html.EditorFor(model =>model.StartDate)
```

这些指示符的定义方式，可以是基于约定，或者是在模型中使用 UI Hint 特性来修饰一个字段。助手模板是一种特别的 Razor 文件，里面包含了在显示和编辑字段时需要显示的特定内容或者编辑逻辑。在上面那个显示 StartDate 字段的例子中，编辑模板可以包含一个用来编辑日期的控件。通过在各个模型中使用修饰特性，你可以将日期类型字段的显示和编辑都定义在一个集中的地方。如果要修改日期类型字段显示的方式，只需要修改一个文件就行了。这是一个典型的“横切”(cross cutting)概念的例子，也就是将同一个逻辑应用到许多不同的类中。

在 MVC 框架中，并不是只有这一个特性使用了横切概念，过滤器也使用了相同的这种概念。一个过滤器就是一个应用到整个控制器或者控制器中某个操作方法的修饰特性。过滤器可以截获或者修改客户端请求，向其中添加数据或者修改已有数据，甚至更改客户端请求的流向。安全控制也通常通过过滤器修饰特性来实现。Authorization 特性提供了一种非常简单的方法，在整个应用程序中应用复杂的权限规则。

在一个 MVC 应用程序中使用横切概念的另一个特性是中间件。中间件构建于.NET 开放 Web 接口(Open Web Interface for .NET, OWIN)标准之上，它很大程度上取代了自定义 IIS 模块的作用，中间件和 IIS 模块一样，可以用来截获所有的客户端请求。在中间件中可以执行常规操作。例如，如果你想要在日志中记录下每一个客户端请求，中间件就可以用来在客户端请求开始和结束时进行记录。

横切概念并非一定只能用在应用程序的逻辑中。它们可以被用在用户界面中。类似 Web Forms 可以使用母版页，MVC 也可以使用和母版页作用相同的布局。单个视图可以将自己想要显示给用户的内容，例如页面内容和面包屑导航，插入到布局中的不同地方。

MVC 允许对实际的 HTML 内容进行更强的控制，从 Web Forms 中移除许多与现代 HTML 技术不兼容的抽象层。使用 MVC 创建的应用程序具有更好的可测试性、更好的架构、更快的用户体验，对于新上手的 Web 开发者来说也更容易学习。

1.4　Web API

ASP.NET MVC 的迅速流行启发了 ASP.NET 团队创建出一个新的 Web 技术框架：Web API。ASP.NET MVC 的目标是入侵 WebForms 的传统地盘，Web API 的目标则是入侵.asmx 文件 SOAP(Simple Object Access Protocol)服务的地盘。.asmx 文件是微软创建的第一代通过 HTTP 协议进行远程方法调用的技术。SOAP Web 服务曾是一种功能强大的技术，但它很快就变得过度复杂。像 WS-Security 和 WS-BPEL 这些标准都异常复杂且难以理解。为实现这些标准，微软创建了 WCF(Windows Communication Foundation)，一个实现了 WS-*复杂标准的运行时和一组 API，它是如此复杂，以至于一个人几乎无法完全了解它的方方面面。

人们需要一种在服务之间以及 Web 客户端和服务器之间进行通信的简单方法。幸运的是，Roy Fielding 基于“具象状态传输”(Representation State Transfer，REST)的论文获得了大量的关注。REST 是一种在一组对象上使用 HTTP 方法进行沟通操作的简单方法。在一个 SOAP Web 服务中，你需要创建类似名为 AddUser 的终结点，它会从一个 SOAP 数据包中获取所需要的参数来创建一个用户。SOAP 数据包中包含消息正文和其他一些可能存在的头信息。服务器在处理完成之后会返回一些回复，以确认服务器端已经完成了添加用户的操作。一个请求的格式如下所示：

```
POST /User HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

<?xml version="1.0"?>
<soap:Envelopexmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:AddUserxmlns:m="http://www.example.org/user">
      <m:FirstName>Glenn</m:FirstName>
      <m:LastName>Block</m:LastName>
    </m: AddUser>
  </soap:Body>
</soap:Envelope>
```

可以看到，这个请求的消息格式非常复杂。如果服务器能够正确地处理这个请求，它也许会返回一个 HTTP 200 状态码。一个实现相同功能的 RESTful 消息的格式是这样的：

```
POST /User HTTP/1.1
Host: www.example.org
Content-Type: application/json; charset=utf-8
Content-Length: 75

{ "FirstName": "Glenn", "LastName":"Block"}
```

除了消息数据包的尺寸大为减小，更有趣的是，服务器可能会回复一个 201 状态码。如果你对这个状态码不熟悉，它代表的是“已创建”状态。通过使用 HTTP 规范中的各种约定，REST 大大降低了 SOAP 的复杂度。例如，要删除一项数据，可以使用 DELETE 方法。要获取一项数据的信息，就使用 GET 方法。

WCF 支持通过创建 WCF Web API 项目来创建 RESTful Web 服务。这是一个简化版的 WCF，它将 WCF 中许多可用选项的默认值都针对 RESTful 的场景进行了特别设置。WCF Web API 项目最终变成了普通的 Web API 项目，它的复杂度也得到了进一步降低。

当第一次看到一个 Web API 项目的代码时，可能会将其误认为一个 MVC 应用程序。一个 Web API 项目和一个 MVC 应用程序项目一样，包含一个 Controllers 文件夹和一个 Models 文件夹。而其他 Views、Content、Scripts 文件夹，因为它们存在的目的是为了包含用来渲染 UI 界面的资源，所以在 Web API 项目中并不存在。

当查看一个 Web API 项目中的一个控制器时，你会注意到里面的操作都定义成它们所负责处理的 HTTP 方法的名字。例如，控制器中可能会有一个 Post 方法，它会在接收到一个 HTTP POST 请求时被调用执行。项目的默认模板会像下面这样生成一个 Post 方法：

```
public async Task<IHttpActionResult> Post(User user)
      {
         if (!ModelState.IsValid)
         {
            return BadRequest(ModelState);
         }

         // TODO: Add create logic here.

         // return Created(user);
         return StatusCode(HttpStatusCode.NotImplemented);
      }
```

如你所见，方法的名字确实就是它所负责处理的 HTTP 方法的名字。

WCF 支持在 IIS 之外单独运行，也就是自托管模式运行，这个特性被 Web API 所继承。这意味着可以将一个 Web API 应用程序复制到某个地方，然后直接运行它。

也许 Web API 最有意思的地方就是它是微软首批开源的项目之一。它的源代码以 Apache 2.0 授权许可完全公开。Web API 项目属于非营利性的.NET 基金会，这是一个在.NET 生态圈致力于促进开源发展的独立组织。

Web API 中的大部分功能都和 ASP.NET MVC 一致。因为 MVC 控制器在简单场景中可以完成和 Web API 控制器几乎一样的功能，所以对于何时应该使用一个 Web API 控制器，何时应该使用一个 MVC 控制器，会让开发者感到一些困惑。在一个项目中，可以同时使用 Web API 和 MVC。

1.5　ASP.NET Core

ASP.NET 的进化之路仍在继续，下一步就是 ASP.NET Core。你也许曾经听到过 ASP.NET

vNext、ASP.NET 5、ASP.NET MVC 6 这些名字，这些都是 ASP.NET Core 以前用过的名字。下一代 ASP.NET 和 ASP.NET MVC 的开发已经以一种非常开放的方式完成。虽说之前版本的 ASP.NET MVC 是开源的，但是开发团队仍然保持了独立封闭的开发模式。外部开发者以前也可以向项目提交代码，但是审核的流程会相当复杂。甚至可以说，以前开发团队并不太鼓励外部开发者提交自己的代码。换句话说，如果你希望在 ASP.NET MVC 中添加一个对你而言非常重要的功能，你只能指望微软内部的开发团队来添加。

ASP.NET Core 使用 GitHub 来进行开发过程的管理，任何人都可以提交对于功能提议和 bug 修复的 pull 请求。虽然微软仍然对项目进行着整体的管理，但是社区对项目已经做出了非常大的贡献，产品的质量相比微软内部团队自己开发也得到了大幅提高。

如果你对参与 ASP.NET 项目感兴趣，可以直接访问 https://github.com/aspnet/home，阅读有关如何参与的说明。

ASP.NET Core 是一个与之前版本 ASP.NET 大大不同的产品。它被设计得非常现代，具备和市场上最强大的 Web 框架(例如 Node、Elixr 和 Go)竞争的实力。

ASP.NET Core 与名为.NET Core 的.NET 运行库有紧密的联系。.NET Core 具备诸多改进之处，其中最突出的一项改进就是它支持 Windows 以外的操作系统。这意味着你可以在 macOS 上开发你的应用程序，然后将它以一个 Docker 镜像的方式部署到一组位于 Azure 云中的 Linux 机器上。

ASP.NET Core 是一项工程杰作，值得用一本书的篇幅来介绍。

1.6　本章小结

微软的 Web 技术栈支持多种不同的服务器端技术。如何正确地选择合适的技术，取决于项目的具体需求。在有些场景中，Web Forms 仍然是正确的选择，而在另外一些需要对输出的 HTML 内容进行进一步优化的场景中，MVC 和 Web API 则会是更好的选择。

虽然微软的技术栈历经变化，但市场上也存在着其他的 Web 开发技术。无论微软的 Web 开发技术会变得如何强大，以及如何具有影响力，它都无法独立垄断 Web 技术的未来。因此，了解 ASP.NET 在更大范围的 Web 开发领域中的位置以及它受到了哪些技术的影响是非常重要的。我们将在第 2 章对此进行介绍。