开源.NET 生态软件开发

Azure、DevOps 和微服务软件架构实战

(第 2 版)

[葡] 加布里埃尔 • 巴普蒂斯特(Gabriel Baptista)
[意] 弗朗西斯科 • 阿布鲁泽塞(Francesco Abbruzzese) 著
叶伟民　张陶栋　王伟　肖宁 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：01-2021-6714

图书在版编目(CIP)数据

Azure、DevOps 和微服务软件架构实战：第 2 版 /(葡) 加布里埃尔 • 巴普蒂斯特(Gabriel Baptista)，(意)弗朗西斯科 • 阿布鲁泽塞(Francesco Abbruzzese)著；叶伟民等译. —北京：清华大学出版社，2023.1
(开源.NET 生态软件开发)
书名原文：Software Architecture with C# 9 and .NET 5: Architecting software solutions using microservices, DevOps, and design patterns for Azure, Second Edition
ISBN 978-7-302-61850-8

Ⅰ.①A… Ⅱ.①加…②弗…③叶… Ⅲ. ①机器学习 ②软件工程—项目管理 Ⅳ. ①TP181 ②TP311.5

中国版本图书馆 CIP 数据核字(2022)第 171206 号

责任编辑：王　军
装帧设计：孔祥峰
责任校对：成凤进
责任印制：

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：
经　　销：全国新华书店
开　　本：170mm×240mm　　印　　张：27　　字　　数：726 千字
版　　次：2023 年 1 月第 1 版　　印　　次：2023 年 1 月第 1 次印刷
定　　价：128.00 元

产品编号：093139-01

中文版推荐序

经过了十余年的发展，云计算已经成为人们日常生活的一部分，对软件架构也产生了重大的影响。作为一名 IT 从业者，在软件架构设计中引入云平台所提供的各种中间件、服务，以及全新的设计理念已经成为基本技能。DevOps 和微服务作为现代软件架构基础实践和框架性解决方案，其覆盖面非常广泛。

2008 年，我第一次使用微软的云计算平台 Azure 的时候，它还是一个 Beta 版，与当时已经相对成熟的 AWS 相比，Azure 那个时候的功能非常简单，仅有 Web App PaaS，甚至连虚拟机都不提供。2022 年，Azure 已经以全球 24%的市场份额稳居云计算市场的第二把交椅，功能上也已经涵盖了 IaaS、PaaS 和 SaaS 三个领域的 600 多项服务，无论在技术方面还是在市场方面都处于领先地位。如何正确地选择适合自己应用的服务并利用这些服务构建最优化的软件架构，是摆在每个软件架构师面前的一道难题。当我开始阅读这本书的时候，我发现自己找到了答案。本书以一个实例为主线，将一系列方法、实践、工具和设计思路串联起来，向读者展现了一个完整的软件架构设计过程。

本书的编写方式与很多技术书籍不同，作者站在架构师的视角，以一个项目的整个生命周期为主线，向读者展示了如何在云时代设计和实现一款软件，其内容涵盖了从软件架构设计的基本原则、需求收集、解决方案设计，可选技术架构的选择与分析，应用软件的数据层、逻辑层和展现层的最佳实践与框架选择，一直到构建团队协作平台、持续交付流水线，以及自动化测试。如果你是一名拥有 3~5 年软件开发经验的软件开发人员，希望能够成为一名架构师，这本书会对你非常有帮助。

徐磊
微软最有价值专家/微软区域技术总监
SmartIDE 开源项目创始人/LEANSOFT 首席架构师
2022 年 10 月 7 日于北京

译 者 序

最近我面试过一些有十几年工作经验的软件架构师，十分惊讶于他们还停留在设计模式、多层架构、TDD 的年代。

作为同样有十几年工作经验的我，十分理解他们。

的确，在我们年轻时，设计模式、多层架构、TDD 曾经是软件架构领域的热门，也是面试时的必考题。那时，掌握这些技术的人可是“抢手货”。

随着时代的发展，云服务、DevOps、微服务慢慢变成了软件架构的热门，软件架构师如果不想被时代淘汰，则必须了解这些技术。这本书就很适合想要了解这些最新技术的软件架构师们。

与其他软件架构方面的书籍相比，本书以一个项目案例贯穿全书，将一颗颗“珍珠”，如云服务、DevOps、微服务等，用项目案例这条“线”串起来，可以帮助你轻松地掌握本书的内容。

更重要的是，对于软件架构师来说，这些新技术可以局部地、慢慢地融入现有项目中。相信本书能够助力你的职业发展。

最后以一个真实场景结尾：

在东莞.NET 俱乐部举办的一场活动中，有人问我，企业出于稳妥起见，不会采用最新一代技术，那么请问：学习新技术的意义是什么？

我是这样回答的：没错，事实的确如此。为了稳妥，企业所采用的技术会落后一代，但是也只会落后一代。如果一直不学习，就会从落后一代变成落后两代、三代，不知不觉就落后了十年，最终会落后于时代。

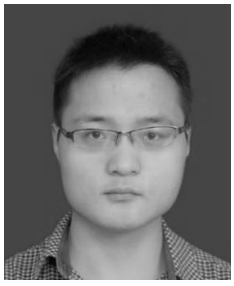
设计模式和多层架构就是如此，都是曾经很热门的新技术，不知不觉间，现在已落后了十年。希望本书能够帮助你追赶上时代的步伐！

译 者 介 绍

叶伟民　拥有 18 年软件开发经验，《.NET 内存管理宝典》等 6 本书的译者（或第一译者），微信公众号“技术翻译实战”创始人，美国硅谷海归学者，广州.NET 俱乐部主席，全国各地.NET 社区名录维护者之一。

张陶栋　翻译爱好者，擅长工程、金融、信息技术等技术领域，对相关交叉学科有技术热情；从事.NET 桌面开发、Web 开发工作 4 年，目前在一家金融企业的人工智能团队从事开发工作；持有系统架构设计师（(软考)证书。

王伟　曾任世界五百强企业研发经理、资深项目经理、高级产品经理；热衷于 CICD、DevOps 文化落地，有丰富的配置管理、DevOps 实施管理经验，致力于帮助团队快速、可靠地交付高质量软件产品；目前主要负责全流程质量及过程优化管理、信息安全部署、审计等相关工作；持有 EXIN DevOps Master 认证。



肖宁　拥有超过 10 年的.NET 开发经验和 Java 开发经验，在企业级 OA 系统和分布式系统的开发方面有丰富的经验，有多年的外企 DevOps 实践经验；目前供职于知名外企。

作 者 简 介

Gabriel Baptista 是一名软件架构师，他领导技术团队跨项目使用 Microsoft 平台完成了多个与零售和工业相关的项目。他是 Azure 解决方案方面的专家，也是一位讲授软件工程、开发和架构等课程的教授，并出版了一些与计算机相关的书籍。他在知名.NET 技术社区网站 Microsoft Channel 9 上演讲，还与他人一起创办了 SMIT 公司，主要开展开发解决方案方面的业务，他将 DevOps 理念视为满足用户需求的关键。

“致我亲爱的家人 Murilo、Heitor 和 Denise，他们经常鼓励我。感谢我的父母 Elisabeth 和 Virgílio，以及我的祖母、外祖母 Maria 和 Lygia，他们一直鼓励我。特别感谢 Packt 团队，全体成员的辛勤劳动保证了这本书的优秀质量。”

Francesco Abbruzzese 是 MVC Controls Toolkit 和 Blazor Controls Toolkit 程序库的作者。他从 ASP.NET MVC 第一个版本就开始为 Microsoft Web 技术栈的传播和推广做贡献。他的公司 Mvcct Team 提供一些与 Web 技术相关的 Web 应用程序、工具和服务。他曾从事人工智能系统相关的工作(例如为金融机构实施了首批决策支持系统)，后来转型去做电视游戏(如当时排名前 10 位的 Puma Street Soccer)。

“感谢亲爱的父母，我的一切都来自他们。特别感谢 Packt 全体员工以及为改进本书整体代码质量做出贡献的审稿人员。”

审校者简介

Mike Goatly 幼年时得到了一台 ZX Spectrum 48K，从此就开始编写程序。他从事过许多行业，包括视频点播、金融科技和零售。他一直专注于 Microsoft 技术栈，从 v1 beta 阶段就开始学习.NET 和 C#，自 2014 年起就一直使用 Azure 构建基于云的软件。Mike 目前拥有 Microsoft Azure 开发者助理认证和 DevOps 工程师专家认证。

“感谢我美丽温柔的妻子和可爱的孩子们给予我的耐心，让我有时间完成这本书。”

Kirk Larkin 是 C#、.NET 和 ASP.NET Core 方面的专家，拥有计算机科学学士学位。他拥有超过 15 年的专业经验，是一名首席软件工程师和 Microsoft MVP。他是 Stack Overflow 网站 ASP.NET Core 话题的最佳回答者、Pluralsight 讲师，以及众多 ASP.NET Core 主题文档的主要作者。他与妻子和两个女儿居住在英国。

前　言

本书涵盖了业界最新的基于云的分布式软件架构中较常见的设计模式和框架。本书基于实际工作中的真实场景来讨论应该何时以及如何使用每种模式。本书还介绍了 DevOps、微服务、Kubernetes、持续集成和云计算等技术与软件开发流程，从而帮助你为客户开发并交付优质的软件解决方案。

本书将帮助你了解客户的需求，指导你解决在开发过程中可能面临的一些大难题，还列出了基于云环境管理应用程序需要了解的注意事项。通过学习本书，你可以了解不同的架构方法，例如分层架构、面向服务架构、微服务、单页应用程序和云架构，并了解如何将这些架构方法应用于具体的业务需求。

本书所有概念都将借助和基于现实工作中的实际用例进行解释。在这些用例中，软件设计原则能够帮助你创建安全和健壮的应用程序。通过学习本书，你可以使用 Azure 在远程环境或云上部署代码，还能够开发和交付高度可扩展且安全的企业级应用程序，以满足最终客户的业务需求。

值得一提的是，本书不仅介绍了软件架构师在开发 C#和.NET Core 解决方案时的最佳实践，还讨论了在最新趋势下开发软件产品需要管理和维护的各种环境。

与第 1 版相比，本书(即第 2 版)在代码及其说明方面进行了改进，并根据 C# 9 和.NET 5 提供的新功能做出了调整。本书添加了最新的框架和技术(如 gRPC 和 Blazor)，并专门新增章节详细介绍了 Kubernetes。

本书读者对象

本书适用于任何希望提高 C#与 Azure 解决方案相关知识水平的软件架构师，也适用于想要成为架构师或希望使用.NET 技术栈构建企业应用程序的工程师和高级开发者。学习本书之前，读者需要拥有 C#和.NET 的使用经验。

本书内容

第 1 章“软件架构的重要性”解释了软件架构的基础知识。该章帮助你以正确的心态来面对客户需求，从而选择正确的工具、模式和框架。

第 2 章“非功能性需求”带你进入应用程序开发的一个重要阶段，帮助你了解收集和说明应用程序必须满足的所有约束和目标，如可扩展性、可用性、可恢复性、性能、多线程、互操作性和安全性。

第 3 章“使用 Azure DevOps 记录需求”介绍一些用于记录应用程序相关需求、bug 和其他信息的技术。该章重点介绍 Azure DevOps 和 GitHub 的使用，其中的大多数概念也通用于其他平台和工具。

第 4 章“确定基于云的最佳解决方案” 对云和 Microsoft Azure 中可用的工具与资源进行了

概述。在该章，你将学习如何正确地搜索工具和资源，以及如何配置它们以满足你的需求。

第 5 章“在企业应用中应用微服务架构”介绍微服务和 Docker 容器。在该章，你将了解如何基于微服务的架构利用云的各种优势，如何在云中使用微服务实现灵活性、高吞吐量和可靠性，以及如何使用容器和 Docker 在架构中混合不同的技术，从而让软件不依赖于特定平台。

第 6 章“Azure Service Fabric”介绍 Microsoft 特有的微服务编排器 Azure Service Fabric。在该章，将实现一个简单的基于微服务的应用程序。

第 7 章“Azure Kubernetes 服务”描述 Kubernetes 在 Azure 上的实现。Kubernetes 是微服务编排器的事实标准。在该章，你将学习如何在 Kubernetes 上打包和部署微服务应用程序。

第 8 章“在 C#中与数据进行交互——Entity Framework Core”详细解释应用程序如何在对象关系映射(ORM)框架，尤其是在 Entity Framework Core 5.0 的帮助下与各种存储引擎进行交互。

第 9 章“在云上选择数据存储”介绍云和 Microsoft Azure 中可用的主要存储引擎。在该章，你将学习如何选择最佳存储引擎来实现所需的读取/写入的并行性，以及如何配置它们。

第 10 章“Azure 函数应用”描述了无服务器计算模型以及如何在 Azure 云中使用它们。在该章，你将学习如何在需要运行某些计算时分配云资源，从而只为实际计算时间付费。

第 11 章“设计模式与.NET 5 实现”通过.NET 5 示例描述了常见的软件模式。在该章，你将了解设计模式的重要性以及使用它们的最佳实践。

第 12 章“不同领域的软件解决方案”描述现代领域驱动设计的软件生产方法，如何使用它来应对需要多领域知识的复杂应用程序，以及如何使用它来为基于云和基于微服务的架构提供帮助。

第 13 章“在 C# 9 中实现代码复用”描述在使用 C# 9 的.NET 5 应用程序中最大化代码可重用性的模式和最佳实践，还讨论代码重构的重要性。

第 14 章“使用.NET Core 实现面向服务的架构”描述面向服务的架构，它使你能够将应用程序的功能公开为 Web 或专用网络上的服务终节点，以便用户通过各种类型的客户端与它们进行交互。在该章，你将学习如何使用 ASP.NET Core 和 gRPC 实现面向服务的架构的服务节点，以及如何使用现有的 OpenAPI 程序包对它们进行文档化。

第 15 章“ASP.NET Core MVC”详细描述 ASP.NET Core 框架。在该章，你将学习如何基于模型-视图-控制器(MVC)模式实现 Web 应用程序，以及如何根据第 12 章所描述的领域驱动设计的方法来组织它们。

第 16 章“Blazor WebAssembly”描述 Blazor 框架，该框架利用 WebAssembly 的强大功能在用户浏览器中运行. NET。在这一章，你将学习如何使用 C#实现单页应用程序。

第 17 章“C# 9 编码最佳实践”描述了使用 C# 9 开发.NET 5 应用程序时的最佳实践。

第 18 章“单元测试用例和 TDD”描述了如何测试应用程序。在该章，你将学习如何使用 xUnit 测试.NET Core 应用程序，并了解如何在测试驱动设计的帮助下轻松开发和维护满足规范的代码。

第 19 章“使用工具编写更好的代码”描述了评估软件质量的指标，以及如何借助 Visual Studio 的所有工具来衡量它们。

第 20 章“DevOps”描述了 DevOps 软件开发和维护方法的基础知识。在该章，你将学习如何组织应用程序的持续集成/持续交付周期。该章还描述了整个部署过程是如何自动化的：首先在源代码存储库中创建新版本，然后通过各种测试和批准步骤，最后在实际生产环境中最终部署应用程序。你还将了解如何使用 Azure Pipelines 和 GitHub Actions 来自动化整个部署过程。

第 21 章“持续集成所带来的挑战”用持续集成场景补充了对 DevOps 的描述。

第 22 章“功能测试自动化”专门介绍功能测试的自动化，即自动验证整个应用程序的版本是否符合约定的功能规范的测试。在该章，你将学习如何使用自动化工具模拟用户操作，以及如何将这些工具与 xUnit 结合使用来编写功能测试。

如何阅读本书

- 本书涉及很多主题，可以将其作为一本解决实际工作问题的指导书。
- 本书需要使用 Visual Studio 2019 社区版或更高版本。
- 读者需要具备 C#和.NET 基础知识。

在线资源

本书的代码存储库托管在 GitHub 上(见参考网站 0.1)。本书免费提供代码、参考网站、各章练习题的答案和各章的扩展阅读等资源，可以通过扫本书封底的二维码下载。

目　录

第1章
软件架构的重要性

软件架构是当今软件行业讨论最多的话题之一，它在未来必将越来越重要。越复杂的解决方案，就越需要优秀的软件架构来支持和维护。由于推出软件新功能的速度不断加快，所以需要优秀的软件架构，同时新的软件架构也不断涌现。

要写好这么重要的主题并不简单，因为这个主题包含了非常多的替代技术和解决方案。本书的主要目的并不是建立一个详尽的、永无止境的可用技术和解决方案列表，而是展示各种技术之间是如何相互关联的，以及这些技术是如何在实践中影响可维护和可持续解决方案构建的。

对于持续专注于创建实际、高效的企业解决方案，这一需求不断增加：用户总是想给应用程序添加更多的新功能。此外，由于市场的快速变化，因此需要频繁地交付应用程序，对于这个需求，必须使用复杂的软件架构和开发技术才能满足。

本章涵盖以下主题：

- 了解什么是软件架构。
- 一些可以帮助你成为软件架构师的软件开发过程模型。
- 如何收集正确信息以设计高质量软件。
- 开发过程中的辅助设计技术。
- 分析系统性需求。
- 贯穿本书的案例简介。

本书将通过一个为 World Wild Travel Club(WWTravelClub)旅行社创建软件架构的案例来讲解整本书的知识点。

使用该案例的目的是帮助你了解每章所讲的理论，以及通过一个实践示例来讲解如何使用 Azure、Azure DevOps、C# 9、.NET 5、ASP.NET 及其他技术来开发企业应用程序。

完成本章的学习，你将能够确切地理解什么是软件架构，了解什么是 Azure，以及如何创建 Azure 账户。你还将了解软件过程、模型和其他技术，这些知识将帮助你管理团队。

1.1 什么是软件架构

在阅读本书时，我们应该感谢那些决定把软件开发当作一个工程领域的计算机科学家。20 世纪 60 年代末，这些科学家提出开发软件的方式与建造建筑物的方式是非常相似的，这就是命名为软件架构的原因。就像建筑架构师设计一个建筑物并根据这个设计监督施工一样，软件架构师的

主要目标是确保得到一个优秀的软件应用程序，而要实现这个目标则需要设计出一个优秀的解决方案。

在专业的软件开发项目中，必须做到以下几点：

- 准确定义客户需求。
- 设计一个很好的解决方案来满足这些需求。
- 实现这个解决方案。
- 与客户一起验证这个解决方案。
- 在工作环境中部署这个解决方案。

软件工程将这一系列活动定义为软件开发生命周期。所有软件开发过程模型(瀑布、螺旋、增量、敏捷等)都是以某种方式管理这个软件开发生命周期。无论使用哪种模型，如果不能处理好前面介绍的这些基本任务，都无法交付可被客户接受的软件。

本书旨在介绍如何设计优秀的解决方案。现实工作中，优秀的解决方案需要满足以下几点：

- 解决方案需要满足用户需求。
- 解决方案需要按时交付。
- 解决方案需要符合项目预算。
- 解决方案需要提供高质量的产品。
- 解决方案需要保证能够在未来进行安全、高效的演进。

优秀的解决方案应该是可持续的，没有优秀的软件架构就没有可持续的软件。优秀的软件架构需要依靠现代化的工具和环境才能完美地满足用户的需求。

因此，本书将使用 Microsoft 提供的一些优秀的工具。.NET 5 是一个统一的软件开发平台，因此.NET 可以帮助我们创建一个很好的解决方案。

.NET 5 是和 C# 9 一起发布的。因为.NET 支持图 1.1 所示的平台和设备，因此 C#是目前世界上最常用的编程语言之一，使用 C#既可以编写在小型设备上运行的程序，也可以编写在不同操作系统和环境中的大型服务器上运行的程序。

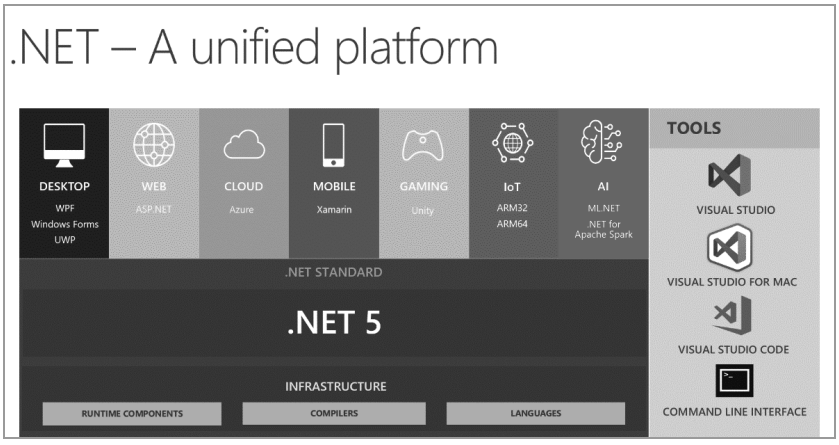


图 1.1 .NET 5 支持的平台和设备

本书还将使用 Azure。Azure 是 Microsoft 的云平台，通过 Azure 可以获得 Microsoft 为构建高级软件架构解决方案提供的所有组件，其中之一就是 Azure DevOps—— 一个可以使用最新的软件开发方法来构建解决方案、管理应用程序生命周期的管理环境。

要想成为一名软件架构师，必须了解上述技术以及其他许多技术。本书将引导你成为一名可以带领团队工作的软件架构师。你将使用本书列出的工具提供最佳的解决方案。下面从创建 Azure 账户开始。

创建 Azure 账户

Microsoft Azure 是目前市面上最好的云解决方案之一。重要的是，Azure 包含了可以帮助我们定义 21 世纪现代化解决方案的架构。

可以通过 Alexey Polkovnikov 开发的优秀网站(见参考网站 1.1)查看 Microsoft Azure 的各种组件。

本节将指导创建 Azure 账户。如果已经创建了该账户，可以跳过这部分。

打开创建 Azure 账户的网站(见参考网站 1.2)，可以找到注册 Azure 所需的信息。该网站通常会自动跳转到你的母语对应的版本。

如果之前没有注册过，则单击“免费使用 Azure”按钮，这样就可以免费使用 Azure 的一些功能。详情请查看参考网站 1.3 的相关内容。

(1) 创建免费账户的过程非常简单，按照网页引导的要求去做即可。首先会要求你注册一个 Microsoft 账户或 GitHub 账户。

(2) 在这个过程中，你还会被要求提供信用卡号码，以确保你是人类而不是垃圾邮件或机器人程序。但是不会向你收费，除非你升级了账户。

(3) 要完成整个过程，需要接受订阅协议、报价详细信息和隐私声明。

账户创建完成后，你将能够访问 Azure 门户。如图 1.2 所示，Azure 门户显示了一个可自定义的仪表板，左侧显示了一个菜单，可以在该菜单中设置将在解决方案中使用的 Azure 组件。在本书的后面，会回到图 1.2 所示界面来设置组件，以帮助构建现代化的软件架构。单击“所有服务”菜单，即可找到更多组件和服务。

创建完 Azure 账户之后，就可以使用 Azure 来了解软件架构师是如何充分利用 Azure 提供的所有组件和服务来带领团队开发软件的。但重要的是，要记住软件架构师的视野不能局限于技术，因为这个角色是由那些需要定义如何交付软件的人来担当的。

如今，软件架构师不仅是一个软件的基础架构师，还决定了整个软件的开发和部署是如何进行的。下一节将介绍世界上广泛使用的软件开发过程模型，首先介绍传统的软件开发过程模型，然后讨论改变了软件构建方式的敏捷模型。

图 1.2　Azure 门户

1.2　软件开发过程模型

作为一名软件架构师，了解目前大多数企业使用的一些常见开发过程非常重要。软件开发过程定义了团队中的人员如何生产和交付软件。一般来说，这个过程与软件工程理论有关，被称为软件开发过程模型。从软件开发被定义为一个工程过程开始，人们就提出了许多用于软件开发的过程模型。下面先回顾一下传统的模型，然后介绍目前常见的敏捷模型。

1.2.1　传统的软件开发过程模型

软件工程理论中引入的一些模型被认为是传统的模型，已经过时。本书的目的并不是涵盖所有这些传统模型，只是对一些公司仍然在使用的瀑布模型和增量模型进行简要介绍。

1. 了解瀑布模型

瀑布模型是一项过时的技术，但事实上，的确有一些公司仍然在使用瀑布模型，我们也刚好借助瀑布模型来讲解软件开发的所有基本步骤。

任何软件开发项目都包括以下步骤：

- 需求，创建产品需求文档，它是软件开发的基础。
- 设计，根据需求设计软件架构。
- 实现，编写程序以实现设计。
- 验证，验证应用程序能否正常工作和是否符合设计的测试。
- 维护，交付后，再次循环执行上面几个步骤。

瀑布模型软件开发周期如图 1.3 所示。

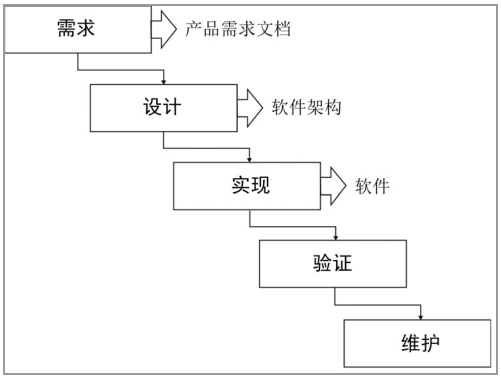


图 1.3　瀑布模型软件开发周期(见参考网站 1.4)

通常，使用瀑布模型会带来软件交付延迟，以及由于交付的最终产品和原本期望的产品之间的差异而导致用户不满等相关问题。此外，根据我的经验，开发完成之后才开始测试会让人感觉压力很大。

2. 增量模型分析

增量模型试图克服瀑布模型最大的问题：用户只能在项目开发工作都完成之后才可以测试解决方案。

增量模型的思想是让用户有机会尽早地与解决方案进行交互，以便用户能够提供有用的反馈，从而有助于软件的开发。

图 1.4 所示的增量模型是作为瀑布模型的替代方法引入的。增量模型的思想是多次增量执行一组软件开发步骤(沟通、计划、建模、构建和部署)。

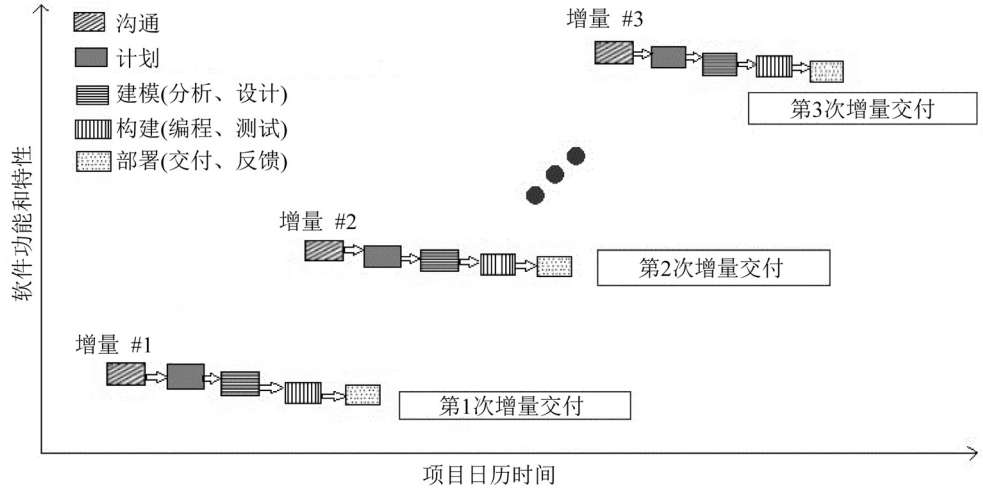


图 1.4　增量模型软件开发周期(见参考网站 1.5)

虽然增量模型减少了因缺乏与客户沟通而产生的问题，但对于大型项目，因为每次增量需要的时间太长而导致增量数量太少仍然是一个问题。

当增量模型在 20 世纪末被大规模应用时，由于需要撰写大量的文件，因此产生了许多与项目官僚作风有关的问题。这些笨拙的场景导致了软件开发行业中一个非常重要的运动——“敏捷开发”的兴起。

1.2.2 敏捷软件开发过程模型

21 世纪初，软件开发被认为是工程领域中最混乱的子领域之一。软件项目失败的比例非常高，这一事实证明需要一种与传统方法不一样的方法来实现软件开发项目所要求的灵活性。

2001 年，敏捷软件开发宣言(简称敏捷宣言)诞生了，之后涌现出了各种敏捷过程模型。其中一些至今依然很常见。

敏捷宣言已被翻译成 60 多种语言。详情请查看参考网站 1.6。

敏捷模型和传统模型的最大区别之一就是开发者与用户交互的方式。所有敏捷模型传递的信息都是越快向用户交付软件越好。对于软件开发者来说，他们可能会困惑：自己只想写代码，为什么需要跟用户沟通？

虽然敏捷软件开发宣言(见图 1.5)的内容如此，但是软件架构师需要记住一点：敏捷过程并不意味着缺乏纪律。当你使用敏捷过程时，将很快明白，没有纪律就无法开发出好的软件。另外，作为一个软件架构师，需要理解软件项目是需要灵活性的。一个拒绝灵活性的软件项目往往会随着时间的推移毁了自己。

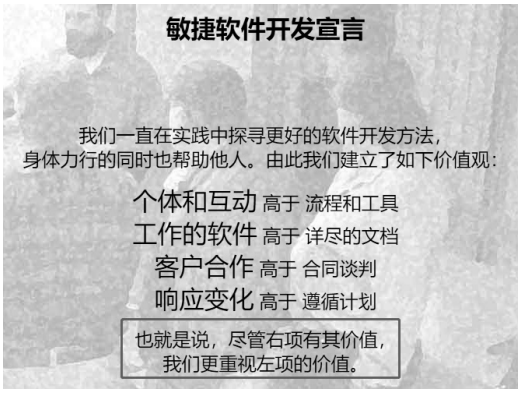


图 1.5 敏捷软件开发宣言

敏捷宣言的 12 条原则[1]就是这种灵活性的基础：

(1) 我们最重要的目标，是通过持续不断地及早交付有价值的软件使客户满意。

(2) 欣然面对需求变化，即使在开发后期也一样。为了客户的竞争优势，敏捷过程掌控变化。

(3) 经常地交付可工作的软件，相隔几星期或一两个月，倾向于采取较短的周期。

(4) 业务人员和开发者必须相互合作，项目中的每一天都不例外。

(5) 激发个体的斗志，以他们为核心搭建项目。提供所需的环境和支援，辅以信任，从而达

1 译者注：此处内容摘自“敏捷宣言”官网，是官网中的中文版内容，并非译者翻译。

成目标。

(6) 不论团队内外，传递信息效果最好，效率也最高的方式是面对面的交谈。

(7) 可工作的软件是进度的首要度量标准。

(8) 敏捷过程倡导可持续开发。责任人、开发者和用户要能够共同维持其步调稳定延续。

(9) 坚持不懈地追求技术卓越和良好设计，敏捷能力由此增强。

(10) 以简洁为本，它是极力减少不必要工作量的艺术。

(11) 最好的架构、需求和设计出自自组织团队。

(12) 团队定期地反思如何能提高成效，并依此调整自身的举止表现。

即使在敏捷宣言发布 20 年后，它的重要性以及与软件开发团队当前需求的联系丝毫未改变。当然，有很多公司都不太接受这种方法，但是作为一个软件架构师，应该了解它是一个可以促进团队发展的机会。

软件开发社区的许多技术和模型都是基于敏捷方法的。下面讨论精益软件开发、极限编程和 Scrum 这几个基于敏捷方法的模型，以供软件架构师决定使用哪些方法来改进软件交付时参考。

1. 精益软件开发

敏捷宣言诞生之后，敏捷社区引入了基于丰田汽车生产方式的精益软件开发方法。已经在丰田汽车实验成功的精益制造方法给敏捷社区提供了一个即使投入很少资源，依然能够获得高质量产品的选项。

“精益软件开发”一词源于 Mary Poppendieck 和 Tom Poppendieck 的同名书籍。两人列出了软件开发的 7 条精益原则，它们与敏捷开发以及 21 世纪许多公司的做法有着真正的联系，具体内容如下。

(1) 消除浪费：按照精益思维，任何不能为客户增加价值的行为即为浪费，包括不必要的功能和代码、软件开发过程的延迟、不明确的需求、繁文缛节、低效的内部沟通。

(2) 内建质量：一个想要保证质量的组织需要在一开始编写代码时就注重提升质量，而不是等到测试阶段才考虑提升质量。

(3) 创造知识：取得卓越成就的公司都有一个共同的模式，即通过严格的实验来获得并记录新知识，并且确保这些知识在整个组织中传播。

(4) 尽量延迟决定：因为软件开发通常具有一定的不确定性，尽可能地延迟决定，直到能够基于事实而不是基于不确定的假设和预测来做出决定。系统越复杂，那么对这个系统能够应对变化的要求就越高，所以需要具备推迟做出重要及关键决定的能力。

(5) 快速交付：交付软件的速度越快，就越能消除浪费，你的客户就比竞争对手具有更显著的优势。

(6) 下放权利：传统的团队里都是由团队的领导者来决定和分配每个人所要完成的任务，但是精益开发主张将这种权利下放到团队的每个人手里，从而使开发者有权利来阐述自己的观点并提出建议。

(7) 全局优化：全局优化使得每个部门之间的联系更紧密。与努力降低每个部门内部的成本相比，消除部门之间的隔阂和浪费会产生更显著的效果。在 DevOps 成为一大趋势的今天，开发部门、质量管理部门和运维部门之间的协同变得越来越重要了。

以上精益原则促使团队或公司采取措施来提高客户真正需要的功能的质量。它还减少了在某些客户根本不会使用的功能上所浪费的时间。

精益方法中先确定哪些功能特性对客户重要然后指导团队交付，这正是敏捷宣言在软件开发团队中推广的内容。

2. 极限编程

早在敏捷宣言发布之前，敏捷宣言一些设计文档的参与者，特别是 Kent Beck，就已经向世界展示了用于开发软件的极限编程(Extreme Programming，XP)方法。

XP 的价值观是简单、沟通、反馈、尊重和勇气。Beck 在关于 XP 的第二本书中说过，XP 将来会被认为是编程领域中的一种社会变革。后来的发展证明，XP 的确给编程领域带来了巨大的变化。

XP 指出，每个团队都应该简单地只做要求做的事情，每天面对面地交流，尽早把软件演示给用户以获得反馈，尊重团队每个成员的专业知识，并有勇气说出真实进度和评估真相，以及将团队的工作作为一个整体来考虑。

XP 还提供了一组规则。如果团队发现某些地方工作不正常，团队成员可能会更改这些规则，但始终坚持采用这种方法是很重要的。

这些规则分为计划、管理、设计、编码和测试等方面。Don Wells 绘制了 XP 相关内容(见参考网站 1.7)。尽管 XP 的一些想法受到许多公司和专家的强烈批评，但目前还是有很多做法是很好的，举例如下。

- 使用用户故事(User Story)编写软件需求：用户故事是一种描述用户需求，以及用于保证用户需求正确实现的验收测试的敏捷方法。
- 将整个软件开发分为迭代版本并按小版本进行交付：迭代在软件开发中的实践是瀑布模型之后的所有方法所捍卫的原则。事实上，更快更小的版本交付的确降低了软件无法达到客户预期的风险。
- 持续保证速度和避免加班：尽管这点肯定是软件架构师可能要处理的最困难的任务之一，但超时工作就已经表明了在整个软件开发过程中有些地方是有问题的。
- 保持简单：在开发解决方案时，通常会尝试预测客户希望拥有的功能。这种方法增加了开发的复杂性和解决方案上线的时间。复杂多样的功能会导致开发的成本较高，而有些功能实际使用的可能性较低。
- 重构：持续重构代码这个方法很好，因为它支撑起软件的开发，并保证了设计的改进。
- 让客户接近你的团队：按照 XP 的规则，你的团队成员中应该有一位来自客户的专家。这当然是一件很难办到和处理的事情，但是这么做能够保证客户参与软件开发的所有部分。另一个好处是，让客户接近你的团队意味着他们了解开发团队所遇到的困难和所具有的专业知识，从而增加双方之间的信任。
- 持续集成：这种做法是当前 DevOps 方法的基础之一。开发者的个人存储库和主存储库之间的差异越小越好。
- 先写单元测试再写实际程序代码：单元测试是一种为实际程序代码的单个单元(类/方法)编写测试代码的方法。这点将会在测试驱动开发(TDD)方法中讨论。这里的主要目标是保证每个业务规则都有自己的单元测试用例。

- 代码必须按照约定的标准编写：需要基于这样一种想法来确定编码的标准，即无论哪个开发者来处理项目的特定部分，都必须能够理解这部分代码。
- 结对编程：结对编程是一种很难在软件开发项目里实现的技术，但是这种技术本身——一个程序员编码，另一个程序员在旁边积极观察并提供意见、批评和建议——在关键场景中是很有用的。
- 验收测试：采用验收测试来满足用户需求是一个很好的方法，可以保证新发布的软件版本不会对现有需求造成损害。一个更好的选择是将这些验收测试自动化。

值得一提的是，目前这些规则中有许多被认为是不同软件开发方法中的重要实践，包括 DevOps 和 Scrum。我们将在本书第 20 章讨论 DevOps。下面介绍 Scrum 模型。

3. Scrum 模型

Scrum 是一种敏捷的软件开发项目管理模型。该模型来源于精益原则，是目前广泛应用的软件开发方法之一。

有关 Scrum 框架的更多信息，请查看参考网站 1.8。

如图 1.6 所示，Scrum 的基础是有一个灵活的 Product Backlog(用户需求)列表，这个列表需要在每个敏捷周期(即 Sprint)中讨论。

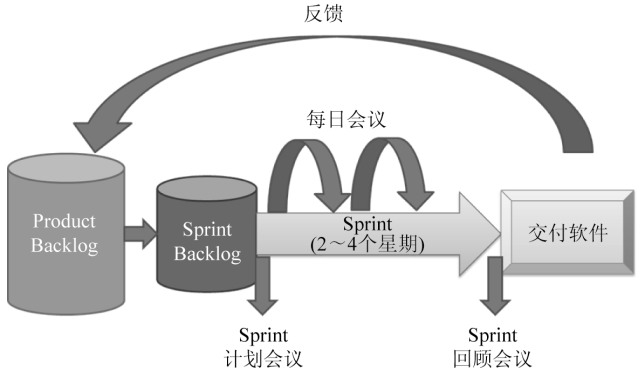


图 1.6　Scrum 过程

每个 Sprint 的目标(即 Sprint Backlog)由 Scrum 团队确定，该团队由 Product Owner、Scrum Master 和开发团队组成。Product Owner 负责确定在这个 Sprint 中交付的 Product Backlog 及其优先级。在整个 Sprint 期间，Product Owner 将帮助团队开发客户所需的功能特性。在 Scrum 过程中，领导团队的人称为 Scrum Master，所有的会议和过程都将由这个人主持。

值得注意的是，Scrum 过程并没有讨论软件是如何实现的，也没有讨论哪些软件开发流程中的步骤是需要完成的，因此你必须将 Scrum 和本章开头讨论的软件开发基础以及流程模型结合在一起。DevOps 是可以帮助你将软件开发过程模型与 Scrum 结合使用的方法之一。详情请参阅第 20 章。

1.3 收集正确信息以设计高质量软件

假设你刚刚接手了一个软件开发项目，现在，是时候用你所有的知识来提供最好的软件给客户了。你的第一个问题可能是：“我如何开始？”作为软件架构师，你将是回答这个问题的人，而且你的答案会随着你领导的每个软件开发项目的不同而变化。

软件开发项目的首要任务是定义软件开发过程。这通常是在项目计划阶段完成的，或者在项目开始之前就要完成。

另一个非常重要的任务是收集软件需求。无论使用哪种软件开发过程，收集真实的用户需求都是一项艰巨而持续的工作。当然，有一些技术可以帮助你很好地完成这项工作。另外，收集需求将有助于定义软件架构的重要方面。

这两项任务被大多数软件开发专家认为是软件开发项目成功的关键。作为软件架构师，你需要很好地完成这两项任务，以便在指导你的团队时能够避免尽可能多的问题。

1.3.1 了解需求收集过程

可以用许多不同的方法表示需求，最传统的方法是在开始分析之前编写一个完美的规范。敏捷方法建议你在准备开始一个开发周期时就编写用户故事。

记住，你不只是为用户编写需求，也是在为你和你的团队编写需求。

事实是，无论在项目中采用何种方法，都必须遵循一些步骤来收集需求，这就是需求工程，其过程如图 1.7 所示。

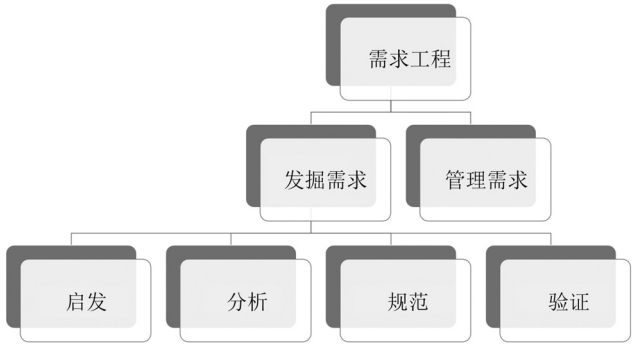


图 1.7 需求工程过程

在需求工程过程中，需要确保解决方案是可行的。某些情况下，可行性分析也是项目计划过程的一部分，应该在开始获取需求时，就已经完成了可行性分析报告。

接下来检查一下这个过程的其他部分，这些内容将提供许多与软件架构相关的重要信息。

1.3.2 收集准确的用户需求

有很多方法可以收集特定场景中用户的真实需求，这个过程称为启发。一般来说，这个过程

可以通过使用那些能够帮助你理解用户需求的技术来完成。常见技术列表如下。

- 想象力：如果你是解决方案所属领域的专家，可以利用自己的想象力来发现新的用户需求，也可以与一组专家一起进行头脑风暴来定义用户的需求。
- 问卷调查：这个工具可用于检测常见的和重要的需求，如用户数量和类型、系统使用峰值，以及常用的操作系统(OS)和 Web 浏览器。
- 访谈：作为软件架构师，访问用户有助于发现问卷调查和想象力无法涵盖的用户需求。
- 观察：没有什么能比与用户在一起相处一天更好的方法来了解用户的日常生活了。

一旦你应用了以上技术中的一个或多个，你就能够得到与用户需求相关的重要信息。

现在你已经能够收集用户需求了。下一节讲述如何分析需求。

在任何时候，你都可以使用这些技术来收集需求，无论是针对整个系统还是针对单个用户故事。

1.3.3　分析需求

收集了用户需求之后，就可以开始分析需求了。我们可以使用以下技术分析需求。

- 原型设计：原型对于阐明和实现系统需求来说是非常有用的。现在已经有很多工具可以帮助你设计原型，Pencil Project 是一个不错的开源原型设计工具。你可以在参考网站 1.9 上找到更多信息。
- 用例：如果你需要详细的文档，统一建模语言(UML)用例模型是一个选项。UML 用例模型由一个详细的说明和一个图表组成。ArgoUML 是一个可以帮助你解决这个问题的开源工具。图 1.8 所示是一个 UML 用例模型的示例。

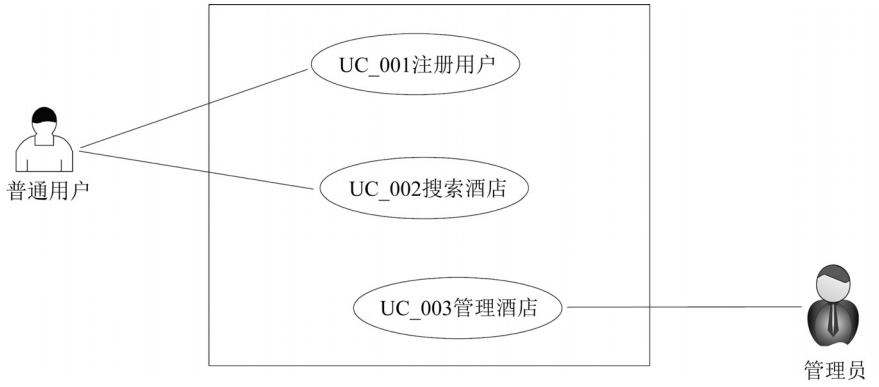


图 1.8　UML 用例模型示例

通过分析需求，你将能够明确用户的真实需求是什么。当你不确定需要解决的真正问题时，这点是很有帮助的，这比那些没有认真分析需求就马上开始编写程序然后期待最好结果的做法要靠谱得多。在需求分析上投入时间就是为更高的软件程序质量投入时间。

1.3.4　将需求整理成规范的文档

完成了需求分析之后，将需求整理成规范的文档是非常重要的。我们可以使用敏捷项目中常

用的用户故事来将需求整理成规范的文档。

规范的需求文档反映了用户和软件开发团队之间的约定。这种文档的编写需要遵循以下基本规则。

- 要确保所有利益相关者都能够准确理解这份约定的内容，即使他们不是技术人员。
- 文档内容需要清晰、明了。
- 需要对每个需求进行分类。
- 使用恰当的表达方式来表示每个需求。
  - 糟糕的例子：普通用户自行注册。
  - 很好的例子：普通用户应自行注册。
- 需要避免模棱两可和有争议的表达。

在文档中添加一些附加信息可以帮助软件开发团队了解他们将要处理的项目上下文。以下是一些添加有用附加信息的提示。

- 添加介绍性的章节以给出解决方案的来龙去脉和完整全貌。
- 创建词汇表，从而让大家更容易理解。
- 描述解决方案将覆盖的用户类型。
- 编写功能性和非功能性需求：功能性需求很容易理解，因为它们准确地描述了软件的功能；而非功能性需求描述了与软件相关的限制，包括可扩展性、健壮性、安全性和性能。我们将在下一节介绍这些方面的内容。
- 附加可以帮助用户理解规则的文档。

在编写用户故事方面，一个很好的建议是通过以下简短的句子来描述每个用户在系统中的每一刻：

```
作为<什么用户>，我想要<什么功能>，所以<需要怎么做>
```

通过这种方法能够准确解释实现该功能的来龙去脉。这种方法也是一个很好的、可以帮助你分析出最关键用户故事并确定任务优先级的工具。这些用户故事还可以很好地告知相关人员应该构建什么样的自动化验收测试。

了解可扩展性、健壮性、安全性和性能的概念

收集用户需求是一项可以帮助你了解要开发的软件的任务。但是，作为软件架构师，除了必须关注的功能性需求，了解非功能性需求也是很重要的，并且是软件架构师在项目开发过程中需要最早完成的工作之一。

我们将在第 2 章更详细地讨论非功能性需求。在这里，我们只需要知道，收集用户需求阶段还需要收集可扩展性、健壮性、安全性和性能方面的非功能性需求。接下来逐一介绍这些概念。

- 可扩展性：全球化提供了在世界各地运行软件程序的机会。这点太棒了，但是作为软件架构师，你需要设计一个能够应对不同时间段、不同请求量的解决方案。可扩展性是指应用程序能够在需要时，通过增加和减少资源数量来应对不同的请求量。
- 健壮性：无论应用程序的可扩展性有多高，如果它不能保证稳定且始终在线，那么你将无法安心。健壮性对于关键应用程序很重要，因为它所要解决的问题类型决定了你在任何时候都没有机会进行维护。在许多行业，应用程序是不可能停止的，很多进程是在无

人监管的情况下运行(如夜间、节假日等)。因此，设计一个健壮的解决方案可以保证你在夜间或节假日不被软件故障所打扰。

- 安全性：这是收集用户需求阶段需要讨论的另一个非常重要的问题。每个人都担心安全问题，世界各地都有不同的法律来处理安全问题。作为软件架构师，必须在设计层面就保证安全性。这是满足安全领域目前正在讨论的所有需求的唯一办法。
- 性能：作为软件架构师，需要了解将要开发的系统才能明确需要做些什么才能获得系统所需的性能。这点需要与用户讨论，才能确定在开发阶段可能面临的大多数性能瓶颈。

值得一提的是，所有这些概念都是当今社会所需要的新一代解决方案的要求。在收集用户需求阶段，为了满足这些要求所做的工作决定了一个软件的质量。

1.3.5　复核用户需求文档

将需求整理成规范的文档后，就需要与利益相关者确认他们是否同意这些文档的内容。这点可以在需求复核会议上完成，也可以使用协作工具在线完成。

展示你所收集的所有原型、文档和信息，每个人都同意这些文档的内容之后，就可以开始研究如何实现项目了。

值得一提的是，你既可以对整个软件也可以对其中的一小部分使用下面描述的这些过程。

1.4　设计技术

定义解决方案并不是一件容易的事，在现有的软件程序上定义解决方案更是难上加难。在软件架构师的职业生涯中，有许多项目并非从零开始开发一套全新的解决方案，而是在客户现有的解决方案基础上继续开发。如果你所设计的解决方案只是在概念上正确，但是没有直观地展现给客户并得到客户的确认，那么可能会很危险。大多数情况下，这个阶段的失误会给将来的工作带来很多问题。

那么，如何将你所设计的解决方案直观地展现给客户呢？一些设计技术可以帮助我们做到这一点，下面介绍其中的两种：设计思维与设计冲刺[1]。

你必须了解的是，这些技术是发现客户真正需求的极好工具。作为软件架构师，你要致力于帮助你的团队在正确的时间使用正确的工具，这些工具可能是确保项目成功的正确选择。

1.4.1　设计思维

设计思维(design thinking)能够帮助你直接从用户那里收集数据，专注于获得解决问题的最佳方案。通过设计思维这种设计技术，软件开发团队将有机会找出所有与系统交互的角色。这将给解决方案带来极好的影响，因为你可从用户角度来设计软件。

设计思维包括以下步骤。

- 共情：在这一步中，你必须进行实地调查以发现用户的关注点。这是你了解软件使用者的阶段。这一步有助于你了解要为谁以及为什么开发这个软件。
- 定义：了解了用户的问题之后，就可以定义用户的需求来解决这些问题了。

1　译者注：设计思维和设计冲刺并非从属于敏捷，但是可以与敏捷结合使用。

- 构思：围绕这些需求举行会议，一起集思广益讨论出一些可能的解决方案。
- 原型设计：原型设计可以直观地呈现这些可能的解决方案，以便大家确认这些可能的解决方案是否真的有效。
- 测试：测试原型将帮助你得出与用户实际需求最吻合的原型。

设计思维专注于加速得出正确产品需求的过程，然后在这个基础上得出最小可行产品(MVP)。其中的原型设计步骤将帮助利益相关者了解产品的最终形态，同时还能帮助软件开发团队交付最佳的解决方案。

1.4.2 设计冲刺

设计冲刺(design sprint)是指在一个为期 5 天的冲刺中通过设计来解决关键业务问题。这项技术是由谷歌提出的，它可以帮助你快速地领会客户的想法，然后快速地构建出一个解决方案并推向市场，从而快速地验证这个想法是否受市场欢迎。

设计冲刺将会专门安排一周的时间组织专家在一个作战室里集中解决上述问题。这一周的工作分配如下。

- 周一：重点是确定冲刺的目标，并列出完成这个目标所面临的困难和问题。
- 周二：确定了冲刺的目标之后，参与者开始制订问题的可能解决方案，同时联系客户参与测试即将得出的这些解决方案。
- 周三：确定了最有把握的解决方案之后，需要将这个解决方案绘制成用户故事地图，从而为原型设计提供基础。
- 周四：基于用户故事地图设计出原型。
- 周五：设计出原型之后，将其展现给客户，并根据客户的反馈做调整。

从以上内容可以看出，加速得出正确产品需求过程的关键之处是原型设计，原型设计可以把团队的想法具体化，使其成为对最终用户来说更具体、更直观的东西。

1.5 收集需求阶段就要考虑的常见问题

如果你想设计出一套优秀的软件，那么本章所讨论的所有信息都很有用，本节所讨论的问题也很重要，无论是使用传统模型还是使用敏捷模型进行开发，是否有考虑这些问题决定了你的软件是专业水平还是业余水平。

如果没有考虑过这些问题，那么将来会给你的软件项目带来一些麻烦。这些问题旨在描述会影响项目质量的地方，以及如何使用前面提到的技术来帮助开发团队解决这些问题。

大多数情况下，如果我们对这些常见问题有概念，那么只需要简单的沟通就能够从客户那里获得足够的信息，从而得出一个优秀的解决方案，处理好这些问题。接下来我们研究一下收集需求阶段需要考虑的三个常见问题：性能、用户的需求未得到正确实现、系统会在什么环境使用。

1.5.1 问题 1：网站太慢，无法打开网页

性能是软件架构师在职业生涯中要解决的最大问题之一。任何软件都可能存在性能问题，因为我们没有无限的计算资源。此外，计算资源的成本很高，特别是对于那些同时拥有大量用户的软件。

仅仅编写需求文档是不能解决性能问题的。但是，如果在编写需求文档阶段把性能需求描述正确，那么以后的工作就会顺利得多。我们必须有与系统期望的性能相关的非功能性需求文档。只需要用如下简单的一句话来描述，就可以帮助到整个项目团队。

非功能性需求: 性能——该软件的任何网页都至少应在 2 秒内做出响应，即使有 1000 个用户同时访问它。

以上这句话只是让每个人(用户、测试人员、开发者、架构师、经理等)得知任何网页都有这么一个目标要实现。这是一个好的开始，但还不够。开发和部署应用程序阶段的工作对解决性能问题也很重要。这就是.NET 5 可以给你很大帮助的地方，尤其是对于 Web 应用程序，在这方面，ASP.NET Core 是现有的、最快的解决方案之一。

作为软件架构师，如果谈到性能，可以考虑使用后面章节中列出的技术和具体的测试用例来满足这一非功能性需求。还有一点很重要，即 ASP.NET Core 及 Microsoft Azure 提供的一些 PaaS(平台即服务)解决方案也能够帮助你轻松地满足这一需求。

1. 缓存

缓存是一种很好的技术，可以提升那些经常会产生相同结果的查询的性能。例如从数据库获取可用的汽车模型，虽然数据库中的汽车数量会增加，但并不会改变汽车模型数据。因为对于一个需要经常访问汽车模型数据的应用程序，一个好的做法就是将这些数据缓存起来。

我们可以使用内存缓存。制订可扩展解决方案时，可以使用 Azure 平台配置分布式缓存。事实上，这两种方式 ASP.NET 都有提供，因此可以选择最适合自己的方式。我们将在第 2 章讲解 Azure 平台的可扩展性方面的内容。

2. 异步编程

开发 ASP.NET Core 应用程序时，应用程序需要设计成可供多个用户同时访问。异步编程可以让你简单地做到这一点，ASP.NET Core 通过关键字 async 和 await 提供了这方面的支持。

这些关键字背后的基本原理是 async 允许任何方法异步运行，await 允许同步异步方法的调用而不阻塞调用它的线程。这种易于开发的模式将使应用程序运行时不会出现性能瓶颈并提供了更好的响应能力。本书将在第 2 章介绍这方面的更多内容。

3. 对象分配

避免性能不足的一个很好的技巧是了解垃圾回收器(GC)是如何工作的。使用完内存之后，GC 会自动回收内存。由于 GC 的复杂性，这里必须介绍几个非常重要的内容。

GC 不会对有些类型的对象自动回收内存，包括与 I/O 交互的任何对象，例如文件和流。如果没有正确使用 C#语法来创建和销毁此类对象，则会出现内存泄漏，从而降低应用程序的性能。

以下是使用 I/O 对象的错误做法：

```
System.IO.StreamWriter file = new System.IO.StreamWriter(@"C:\sample. txt");
file.WriteLine("Just writing a simple line");
```

以下是使用 I/O 对象的正确做法：

```
using (System.IO.StreamWriter file = new System.IO.StreamWriter(@"C:\
sample.txt"))
```

```
{
    file.WriteLine("Just writing a simple line");
}
```

值得注意的是，以上正确做法还可以确保内容肯定写入文件中(它调用了 Flush)。而在错误做法的示例中，可能会导致内容没有写入文件中。尽管以上实践只提到了 I/O 对象，但强烈建议对所有非托管资源对象都这么做。可以在 Visual Studio 解决方案中使用代码分析器的错误警告功能来提醒自己避免犯这些错误，这将有助于 GC 回收内存，从而保证应用程序以适当的内存量运行。否则，这样的错误可能会根据对象的类型像滚雪球一样越滚越大，最终导致其他糟糕情况发生，例如端口/连接耗尽。

你还需要了解的另一个重要内容是 GC 回收对象所花费的时间也会影响应用程序的性能。因此，我们要避免分配大对象；否则，需要等待 GC 完成任务，这会给你带来麻烦。

4. 数据库访问

数据库访问是最常见的性能弱点之一。在编写数据库查询或 lambda 表达式从数据库获取数据时，如果不注意，可能会带来性能问题。本书将在第 8 章介绍相关内容，包括如何正确地从数据库获取数据，如何正确地筛选行与列以避免数据库访问方面的性能问题。

一个好消息是，前面所介绍的与缓存、异步编程和对象分配相关的最佳实践完全适用于数据库环境。所以，我们只需要选择正确的模式就能获得更好的软件性能。

1.5.2 问题 2：用户的需求未得到正确实现

越通用的技术就越难以准确地满足用户的需求。也许这句话听起来很奇怪，但你必须明白，一般来说，开发者主要研究如何使用通用的技术来开发软件，而很少研究如何使用这些通用的技术来满足特定领域的需求。当然，学习如何开发软件并不容易，要理解特定领域的特定需求就更难了。本节的问题是，一个开发者，无论是否软件架构师，如何做到针对他们负责的特定领域交付软件而不仅仅停留在通用领域。

用户需求收集阶段如果做得好将帮助你完成这项艰巨的任务，需求文档编写得好将使你真正了解用户的需求并设计好相应的软件架构。以下几种方法可以最大限度地降低生产出来的软件不吻合用户真实需求的风险：

- 对界面进行原型化以更快地确保用户界面是正确的。
- 设计数据流以检测系统和用户操作之间的差距。
- 定期召开相关会议，根据用户对需求的反馈进行更新，并在这个基础上进行增量交付。

同样，作为软件架构师，你必须定义如何实现软件。大多数时候，你不会是动手编程的那个人，但你将永远是对此负责的那个人。因此，可以使用以下技术避免错误的实现：

- 与开发者一起复核需求，以确保开发者真正了解需要开发的内容。
- 用于检验代码的标准。本书第 19 章将介绍这一点。
- 召开会议协调各种资源以消除开发者遇到的各种障碍。

请记住，实现用户的真正需求是软件架构师的责任，所以请使用所能使用的所有工具来做到这一点。

1.5.3　问题 3：系统会在什么环境使用

了解系统会在什么环境使用是软件项目成功的关键。软件的呈现方式和解决问题的方式决定了用户是否会真正使用它。作为软件架构师，必须了解系统会在什么环境中使用。

了解用户的真实需求将帮助你确定软件是在网页上运行的，还是在手机上运行的，或者是在后台运行的。了解这方面的信息对于软件架构师来说是非常重要的，因为只有正确地了解系统的使用环境，才能够正确地呈现系统的界面元素。

另外，如果你不在乎这一点，那么你只能提供一个勉强可以工作的软件。这在短时间内也许是可行的，但这样的软件终究不能完全满足用户的真正需求。必须记住一点：一个优秀的软件应该能在多种平台和设备上运行。

.NET 5 是一个十分优秀的跨平台软件。通过.NET 5 既可以开发出一套能够运行在 Linux、Windows、Android 和 iOS 上的解决方案，也可以在大屏幕、平板电脑、手机甚至无人机上运行应用程序；既可以将应用程序嵌入各种物联网设备实现自动化，也可以嵌入 HoloLens 实现混合现实。软件架构师必须以开放的心态来设计用户需要的东西。

1.6　World Wild Travel Club 案例简介

正如本章开头提到的，本书将通过一个为 World Wild Travel Club(WWTravelClub)旅行社创建软件架构的案例来讲解整本书的知识点。

WWTravelClub 是一家旅行社，旨在改变人们在世界各地度假和旅行的决策方式。为此，该旅行社开发了一项在线服务。在该服务中，旅客可以得到目的地专家的协助从而带来极佳的旅行体验。

该平台的理念是，你可以同时成为旅客和目的地专家。作为专家帮助别人的次数越多，得到的积分就越高。这些积分可以兑换成该平台原本需要付费购买的服务和套餐。

需要知道的是，一般来说，客户不会按照软件开发者方便理解的格式来准备好需求，这就是收集需求过程如此重要的原因。客户对平台提出了以下需求。

- 普通用户视图
  - 首页上的促销套餐
  - 搜索套餐
  - 每个套餐的详细信息
    - ► 购买套餐
    - ► 购买包含专家的套餐
      - ♦ 评论你的经历
      - ♦ 询问专家
      - ♦ 评价专家
    - ► 注册为普通用户
- 目的地专家视图
  - 与普通用户视图相同的视图
  - 回答关于你的目的地专业知识的问题

  - 管理回答问题所获得的积分
  - 积分兑换功能
- 管理员视图
  - 管理套餐
  - 管理普通用户
  - 管理目的地专家

同时需要注意的是，WWTravelClub 计划为每个套餐配备 100 多名目的地专家，并针对世界各地提供大约 1000 种不同的套餐。

了解用户需求和系统需求

你可以通过以下用户故事总结出 WWTravelClub 的用户需求[1]：

- US_001: 作为一名未注册用户，我想在首页查看有哪些促销套餐，从而轻松地得出下一次度假的时间段。
- US_002: 作为一名未注册用户，我想搜索出那些在首页上找不到的套餐，从而浏览其他旅行机会。
- US_003: 作为一名未注册用户，我想查看套餐的详细信息，从而决定购买哪个套餐。
- US_004: 作为一名未注册用户，我想自行注册成注册用户，从而开始购买套餐。
- US_005: 作为一名已注册用户，我想进行付款，从而购买套餐。
- US_006: 作为一名已注册用户，我想购买一个包含专家的套餐，从而有一次绝佳的旅行经验。
- US_007: 作为一名已注册用户，我想咨询一名专家，从而得出这次旅行我应该怎样做才能过得最愉快。
- US_008: 作为一名已注册用户，我想评论一下我的体验，从而对我的旅行给出反馈。
- US_009: 作为一名已注册用户，我想评价一名帮助过我的专家，从而与其他人分享他有多棒。
- US_010: 作为一名已注册用户，我想注册为目的地专家，从而帮助那些来我市旅游的人。
- US_011: 作为一名专家用户，我想回答我所在城市的相关问题，从而获取积分以在将来使用。
- US_012: 作为一名专家用户，我想用积分兑换机票，从而有更多的旅行。
- US_013: 作为一名管理员用户，我想管理套餐，从而让用户有绝佳的旅行机会。
- US_014: 作为一名管理员用户，我想管理注册用户，从而保证 WWTravelClub 能够提供良好的服务。
- US_015: 作为一名管理员用户，我想管理专家用户，从而保证与目的地相关的所有问题都能得到回答。
- US_016: 作为一名管理员用户，我想针对世界各地提供 1000 多个套餐，从而让注册用户在不同的国家都能够体验到 WWTravelClub 的服务。
- US_017: 作为 CEO，我希望即使有 1000 多名用户同时访问网站，网站也能够正常服务，从而让我们的业务能够有效扩展。

1 译者注：注意，在编写用户故事时，将用户描述的普通用户拆分成未注册用户和已注册用户两种。

- US_018: 作为一名任何种类的用户，我想用我的母语浏览 WWTravelClub，从而轻松理解网站提供的套餐内容。
- US_019: 作为一名任何种类的用户，我希望通过 Chrome、Firefox 和 Edge Web 等浏览器都能够正常访问 WWTravelClub，从而可以使用我喜欢的 Web 浏览器。
- US_020: 作为一个任何种类的用户，我希望我的信用卡信息是被安全存储的，从而放心地购买套餐。

注意，在编写用户故事时，可以加上那些与非功能性需求(如安全性、环境、性能和可扩展性)相关的信息。

但是，在编写用户故事时，你很可能会忽略某些系统需求，而这些需求是必须包含在软件规范文档中的。这些需求可能与法律、硬件和软件先决条件有关，甚至与系统正确交付的注意事项有关，因此必须列出这些需求并整理成用户故事。WWTravelClub 的系统需求如下(注意，这些系统需求是针对将来编写的，因为系统目前还不存在)。

- SR_001: 系统应使用 Microsoft Azure 组件来提供所需的可扩展性。
- SR_002: 系统应遵守欧盟《一般数据保护法案》[1](GDPR)的要求。
- SR_003: 系统应在 Windows、Linux、iOS 和 Android 平台上运行。
- SR_004: 系统的任何网页都应在 2 秒内做出响应，即使有 1000 个用户同时访问它。

如果从软件架构的角度来思考 WWTravelClub 的开发，那么以上这些用户需求和系统需求用户故事列表将帮助你理解 WWTravelClub 的开发可能有多复杂。

1.7　本章小结

本章介绍了在软件开发团队中担任软件架构师有哪些要求和需要完成哪些工作，软件开发过程模型和收集需求过程的基础知识，以及如何创建 Azure 账户，本书的案例需要这一步。另外，本章还讲解了功能性需求和非功能性需求，以及如何将它们编写成用户故事。这些技术和工具将帮助你交付更好的软件产品。

在第 2 章，你将有机会了解功能性需求和非功能性需求对于软件架构的重要性。

1.8　练习题

1. 软件架构师需要具备哪些专业技能？
2. Azure 是如何帮助软件架构师的？
3. 软件架构师如何选择在项目中使用的最佳软件开发过程模型？
4. 软件架构师如何帮助开发人员收集需求？
5. 软件架构师需要编写什么样的需求规范？
6. 设计思维和设计冲刺如何帮助软件架构师收集需求？

1 译者注：又名《一般数据保护条例》《通用数据保护法案》《通用数据保护条例》。

7. 用户故事如何帮助软件架构师整理需求?
8. 开发高性能软件的技术有哪些?
9. 软件架构师如何检查用户需求是否得到正确实现?