移动开发经典丛书

Flutter 入门经典

[美] 马可 • 纳波利(Marco L. Napoli) 著
蒲 成 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：01-2020-2326

Marco L. Napoli
Beginning Flutter: A Hands On Guide To App Development
EISBN：978-1-119-55082-2

图书在版编目(CIP)数据

Flutter入门经典 / (美)马可 • 纳波利(Marco L. Napoli)著；蒲成译. —北京：清华大学出版社，2021.1
(移动开发经典丛书)
书名原文：Beginning Flutter: A Hands On Guide To App Development
ISBN 978-7-302-56954-1

Ⅰ. ①F…　Ⅱ. ①马…　②蒲…　Ⅲ. ①移动终端—应用程序—程序设计　Ⅳ. ①TN929.53

中国版本图书馆 CIP 数据核字(2020)第 223939 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：宋　林

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：涿州京南印刷厂
经　　销：全国新华书店
开　　本：170mm×240mm　　**印　　张**：31.75　　**字　　数**：792 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：085162-01

译 者 序

Flutter是Google公司推出的新一代前端框架，最初目标只是为了满足移动端跨平台的应用开发，开发人员可使用 Flutter 在 iOS 和 Android 上快速构建高质量的原生用户界面。但如今，Flutter 已经开始扩展为同时面向移动端、Web、桌面端以及嵌入式设备开发应用了。Flutter 正在被越来越多的开发人员和组织所使用，也是构建未来的 Google Fuchsia 应用的主要方式，并且它是完全免费、开源的。

Flutter组件采用现代响应式框架构建，这是从React中获得的灵感，其中心思想是用组件(Widget)构建 UI。Widget 描述了在给定其当前配置和状态时 UI 应该显示的样子。当 Widget 状态发生改变时，Widget 就会重构其描述，进而 Flutter 会对比该 Widget 之前的描述，以确定底层渲染树从当前状态转换到下一个状态所需要的最小更改，从而获得极佳的渲染性能。

本书是一本 Flutter 入门级书籍，讲解 Flutter 的主要知识点、核心概念以及基础的 Dart 编程知识，并且介绍各种工具。本书采取循序渐进、示例辅助的内容组织方式，以期让读者能够零基础入门并且随着对本书的深入阅读而稳步提升技能，同时逐渐加深对于 Flutter 和 Dart 的理解。相信在阅读并深刻理解了本书的知识内容之后，读者就能熟练运用 Flutter 和 Dart 进行基本的跨平台应用开发了。本书提供了许多应用示例，并且每一章结尾处都有知识点归纳和小结，读者能通过每一章的知识内容并且结合实践练习来巩固从本书中学到的知识。

当然，任何一种编程语言、开发框架及其开发环境都仅是我们达成工作目标的工具而已。因此，作为使用者，我们所要做的就是学习并掌握其使用方法。就这方面而言，本书称得上是一本非常完备的工具类参考书籍。对于没有任何 Flutter 开发经验的读者，跟随本书的章节内容就可以完全掌握 Flutter 的基本开发技术，并能熟练地对 UI Widget 进行设置和使用，而这也是作者撰写本书的目的。希望会有越来越多的读者投身到 Flutter 的生态之中！

在此要特别感谢清华大学出版社的编辑们，在本书的翻译过程中他们提供了颇有助益的帮助，没有他们的热情付出，本书将难以付梓。

由于译者水平有限，书中难免会存在一些翻译不准确的地方，如果读者能够指出并勘正，译者将不胜感激。

作 者 简 介

Marco L. Napoli 是 Pixolini 有限公司的 CEO，也是一位经验丰富的移动端、Web 和桌面端应用开发者。他在可视化开发优雅美观且易于使用的系统方面已得到了业内的广泛认可。早在 2008 年他就编写了自己的首个原生 iOS 应用。www.pixolini.com 上展示了其工作成果和已发布的应用。

Marco 儿时就迷恋上了计算机。他的父亲注意到了这一点并给他买了一台 PC(个人计算机)，从那时起他就开始开发软件了。他曾就读于迈阿密大学攻读建筑学学位，但当时他就已经开始经营自己的商业业务了，并在四年后他认定建筑学并不适合自己。他为各种各样的行业开发过系统，其中包括银行业、医疗保健行业、房地产行业、教育行业、货运业、娱乐业等。不久之后，一家业内领先的银行业软件公司收购了他的 MLN Enterprises 公司。MLN Enterprises 公司的主要产品是抵押贷款银行业务软件、运算处理业务软件以及市场营销软件。

接下来，他开启了咨询顾问的生涯，并在不久后创建了 IdeaBlocks 有限公司。该公司的主营业务是软件开发咨询，曾经为一个销售酒店服务软件的客户开发了移动端、桌面端和 Web 平台，主要产品包括酒店营销软件、餐饮软件、网络空间软件、客户服务软件以及维护软件；这些产品通过云服务器使用 Microsoft SQL Server 和应用于敏感数据的加密处理进行数据同步。其客户端的用户包括凯悦嘉轩&嘉寓酒店、希尔顿酒店、假日酒店、希尔顿欢朋酒店、万豪酒店、贝斯特韦斯特酒店、丽笙酒店、喜来登酒店、豪生酒店、希尔顿合博套房酒店等。在该公司的合同都完成后，他就关闭了 IdeaBlocks。

如今，他将重心放在 Pixolini 的运营上。Pixolini 开发了用于 iOS、macOS、Android、Windows 以及 Web 的移动端、桌面端和 Web 应用。他同时也在 Udemy 在线教育网站上授课，主要讲解如何使用他开发的一款 Web 应用来分析房地产投资。他已经开发并在各大应用商店中发布了十几款应用。

“离开了意大利的特浓咖啡卡布奇诺，我就无法写代码了，并且我热爱中国武术。”

Marco 和妻子 Carla 共同养育了三个出色的孩子。

技术编辑简介

Zeeshan Chawdhary 是一位狂热的技术专家，已经从业 14 年了。其职业生涯起步于使用 J2ME 开发移动端应用，不久之后他大胆进入 Web 开发领域，开发出了不少健壮且可扩展的 Web 应用。作为首席技术官，他带领团队为许多公司构建了 Web 和移动端应用，这些公司包括诺基亚、摩托罗拉、梅赛德斯、通用汽车、美国航空以及万豪酒店。他目前是一个国际化团队的开发总监，使用各种技术为客户提供服务，这些技术包括 Magento、WordPress、WooCommerce、Laravel、NodeJS、Google Puppeteer、ExpressJS、ReactJS 以及.NET。他还编著了一些有关 iOS、Windows Phone 和 iBooks 的书籍。

致　谢

我想要感谢 Wiley 出版社这个才华横溢的团队，其中包括所有的编辑、经理，以及许多在幕后帮助本书出版的人。感谢 Devon Lewis，他很早就认识到，Flutter 会对整个行业产生巨大影响；同时感谢 Candace Cunningham，她的项目编辑技能和见解对我提供了很多帮助；还要感谢 Zeeshan Chawdhary，他为我提供了技术性的信息输入和建议；此外，Barath Kumar Rajasekaran 及其团队完成了本书的出版准备工作，谢谢你们；我还要感谢 Pete Gaughan 随时随地为我提供的帮助。

要特别感谢 Google 的 Flutter 团队，尤其是 Tim Sneath、Ray Rischpater 和 Filip Hráček，他们给予我盛情且宝贵的反馈。

最后想要感谢的是妻子和孩子，他们耐心地倾听我所表达的内容并对本书所创建的各个项目给予了反馈。

前　言

Flutter 最初是在 2015 年的 Dart 开发者大会上以 Sky 这个名称公开发布的。Eric Seidel(Google 的 Flutter 工程总监)在那次大会的演讲中首先讲到，他参会的目的是介绍 Sky。Sky 是一个实验项目，被称为“移动端上的 Dart”。Eric 已经在 Android Play Store 上构建并且发布了一个演示应用，在开始介绍该演示应用之前，他表示没有使用任何 Java 代码来绘制这个应用，这意味着它是原生应用。Eric 展示的第一个特性是旋转的正方形。Dart 以 60Hz 的频率驱动该设备，这也是该系统的首要目标：快速且灵敏响应(Eric 曾希望能够更快运行，如 120Hz，但他当时所使用的设备的能力限制了这一点)。Eric 继续展示了多点触控、快速滚动以及其他特性。Sky 提供了最佳的移动体验(对于用户和开发人员来说都是这样)；开发人员在 Web 开发方面遇到过不少难处，他们认为应该可以做得更好。用户界面(UI)以及业务逻辑都是以 Dart 编写的，其目标就在于实现平台中立。

很快到了 2019 年，Flutter 强有力地驱动包括 Google Home Hub 在内的 Google 智能显示平台，并且是通往使用 Chrome OS 来支持桌面应用的第一步。目前的结果就是，Flutter 支持运行在 macOS、Windows 以及 Linux 上的桌面应用。Flutter 被描述为一个可移植 UI 框架，它可将一套代码用于所有界面，如移动端、Web、桌面和嵌入式设备。

本书将讲解如何使用 Flutter 框架以及 Dart 编程语言，通过一套代码来开发用于 iOS 和 Android 的移动应用。随着 Flutter 拓展到移动端之外的其他领域，我们能利用本书中讲解的知识来进行针对其他平台的开发。本书读者不需要具备编程经验；本书会从基础知识讲起，并且逐步推进到开发可用于生产环境的应用程序。

本书将以简单朴实的风格讲解每一个概念。读者可以遵循“试一试”的实战练习来实践所学知识并创建出针对特定特性的应用。

每一章内容都是基于之前章节的内容来编写，并会增加一些新概念以便帮助读者了解如何构建美观、具有动画特效且功能丰富的应用。阅读完本书后，读者将能利用所学知识和技术来开发自己的应用。本书的最后四章会创建一个日记应用以便在本地保存数据，还会创建另一个日记应用，其中要使用状态管理、身份验证，以及包括离线同步在内的多设备数据云端同步能力来跟踪用户的心情变化，这些功能对于现今的移动应用而言都是必需的。本书竭尽所能地使用了一种友好且通俗的方法来讲解技术，以便读者能够在阅读本书的过程中学习到工作中所需的基础知识和高级概念。

在我首次观看到 Google 所展示的 Flutter 后，就深深地迷恋上它。尤其吸引我的一点是，Flutter 的一切都是以 Widget 为基础的。我们可以利用 Widget 并且将它们嵌套(组合)在一起来创建所需的 UI；而且最好的一点是，我们可以轻易地创建自己的自定义 Widget。另外一点比较吸引我的地方在于，Flutter 能使用一套代码开发出面向 iOS 和 Android 的应用；这是我长期以来一直渴求的特性，但在 Flutter 出现之前我都没有找到一种绝佳的解决方案。Flutter 是声明式的，是一种现代响应式框架，其中 Widget 可根据其当前状态来处理 UI 所应具备的外观。

我对于使用 Flutter 和 Dart 进行开发的热情持续增长，因而我决定撰写这本书来与读者分享我的经验和专业知识。我深信本书对于初学者和有经验的开发人员都是有用的，能让大家掌握相关的工具和知识以便进行应用构建并且发展成为一名多平台开发人员。本书内容中包含许多提示、见解、假设场景、图表、截图、示例代码和练习。可扫描封底二维码下载源代码。

本书读者对象

所有希望学习如何使用 Flutter 和 Dart 来开发移动端、多平台应用的人都可以阅读本书。对于希望学习如何开发现代、具备快速原生性能的响应式 iOS 和 Android 移动应用的无经验初学者而言，本书非常适合阅读。此外，本书也可以让毫无经验的初学者学习到开发可用于生产环境的应用所需的高级概念。本书也适合具备编程经验的、希望学习 Flutter 框架和 Dart 语言的读者阅读。

本书假设读者不具备任何编程、Flutter 或 Dart 经验。如果读者已经具有其他语言的编程经验或者熟悉 Flutter 与 Dart，那么通过阅读本书将更深入地理解每个概念和每种技术。

本书内容要点

前面几章会介绍和讲解 Flutter 框架的架构、Dart 语言，以及创建新项目的步骤。读者将使用这些知识为本书中的每一个练习创建新项目。每一章的内容都会关注新概念，以便让读者学习新知识。同时希望这些章节内容能成为读者巩固掌握每个概念的参考资料。

从第 2 章开始，在读者学习每个概念和每种技术时，还可遵循“试一试”实战练习并且创建新的应用项目，以便学以致用。在继续阅读的过程中，每一章都会帮助读者学习更多高级主题。最后四章专注于创建两个可用于生产环境的应用，其中要应用前面几章所讲解的内容并会实现新的高级概念。

本书内容结构

本书分为三部分，共 16 章。虽然每一章都是基于前面所介绍的概念来组织编写的，但每一章都是独立的，读者可跳到某一感兴趣的章节进行阅读以便学习或进一步理解相关主题。

第 I 部分：Flutter 编程基础

本书的第 I 部分将让读者了解 Flutter 的核心知识，这样就能让读者打下构建 Flutter 应用的坚实基础。

第 1 章：Flutter 入门——将讲述 Flutter 框架工作的背后原理以及使用 Dart 语言的好处。读者将了解 Widget、Element 以及 RenderObject 的相关性，并将理解它们如何组成 Widget 树、Element 树以及渲染树。读者将了解 StatelessWidget 和 StatefulWidget 及其生命周期事件。该章还会介绍 Flutter 的声明特点，这意味着 Flutter 会构建 UI 来反映应用的状态。读者将学习如何在 macOS、Windows 或 Linux 上安装 Flutter 框架、Dart、编辑器和插件。

第 2 章：创建一个 Hello World 应用——将介绍如何创建首个 Flutter 项目，以便读者熟悉项目创建过程。通过编写一个最基础的示例，引导读者掌握应用的基础结构，了解如何在 iOS 和 Android 模拟器上运行应用，以及如何对代码进行变更。这个时候不必担心还不理解代码，后续章节将逐步引导读者掌握代码知识。

第 3 章：学习 Dart 基础知识——Dart 是学习开发 Flutter 应用的基础，该章讲解 Dart 的基础结构。读者将学习如何对代码进行注释、main()函数如何启动应用、如何声明变量以及如何使用 List 来存储值数组。读者还将了解运算符，以及如何使用运算符来执行算术运算、相等性判断、逻辑运算、条件运算及级联标记。该章将介绍如何使用外部包和类，以及如何使用 import 语句。读者将了解如何使用 Future 对象来实现异步编程，还将了解如何创建各个类以便对代码逻辑进行分组并且使用变量来留存数据，以及如何定义执行逻辑的函数。

第 4 章：创建一个初学者项目模板——该章将介绍创建新项目的步骤，将使用和复制这些步骤来创建所有练习，还将介绍如何在项目中组织文件和文件夹。读者将根据所需的操作类型来创建最常用的名称以便分组 Widget、类和文件。读者还将了解如何结构化 Widget 以及导入外部包和库。

第 5 章：理解 Widget 树——Widget 树就是组合(嵌套)使用 Widget 以便创建简单与复杂布局的结果。随着开始嵌套 Widget，代码会变得难以阅读，所以尝试尽可能保持 Widget 树的浅层化。该章将介绍所使用的 Widget。读者将理解深度 Widget 树的影响，并将了解如何将其重构为浅层化 Widget 树，从而让代码更易于管理。该章将介绍创建浅层 Widget 树的三种方式，也就是分别使用常量、方法以及 Widget 类进行重构。读者将了解每种技术实现的优劣。

第 II 部分：充当媒介的 Flutter：具象化一个应用

本书的第 II 部分内容比较棘手一些，需要读者动手实践，这一部分将逐步介绍如何添加创建绝佳用户体验的功能。

第 6 章：使用常用 Widget——将学习如何使用最常用的 Widget，它们都是用于创建美观 UI 以及良好用户体验(UX)的基础构造块。读者将了解如何从应用的资源包以及通过统一资源定位符借助 Web 来加载图片，还将了解如何使用配套的 Material Components 图标以及如何应用装饰器来增强 Widget 的外观体验或将它们用作录入字段的输入指引。该章将介绍如何使用 Form Widget 将文本字段录入 Widget 作为一个分组进行校验。还将介绍检测设备方向的不同方式，以便根据设备所处的纵向或横向模式来相应地布局 Widget。

第 7 章：为应用添加动画效果——将了解如何为应用添加动画效果以便表达操作。在恰当使用动画效果时，将提升 UX，但过多或不必要的动画效果也会造成糟糕的 UX。该章将介绍如何创建 Tween 动画效果，还将介绍如何通过 AnimatedContainer、AnimatedCrossFade 和 AnimatedOpacity Widget 来使用内置的动画效果。读者将了解如何使用 AnimationController 和 AnimatedBuilder 类来创建自定义动画效果。还将了解如何使用多个 Animation 类来创建交错动画。该章将讲解如何使用 CurvedAnimation 类来实现非线性效果。

第 8 章：创建应用的导航——好的导航会创造极佳的 UX，从而让信息访问更加简单。读者将了解到，在导航到另一个页面时添加动画效果也能提升 UX，只要该效果能够表达一项操作，而不是造成困扰。该章将讲解如何使用 Navigator Widget 来管理一系列路由以便在页面之间移动，还将讲解如何使用 Hero Widget 来表达导航动画效果以便将 Widget 从一个页面移动和缩放到另一个页面。该章将介绍使用 BottomNavigationBar、BottomAppBar、TabBar、TabBarView 和 Drawer Widget 来添加导航的不同方法，还将介绍如何使用 ListView Widget 以及 Drawer Widget 来创建可导航菜单项列表。

第 9 章：创建滚动列表和效果——该章将介绍如何使用不同的 Widget 来创建滚动列表，以便帮助用户查看和选择信息，还将介绍如何使用 Card Widget 配合滚动列表 Widget 来分组信息。读者将学习如何使用 ListView Widget 来构建可滚动 Widget 的线性列表，还将学习如何使用 GridView 以网格形式展示可滚动 Widget 的磁贴块。读者将了解如何使用 Stack Widget 配合滚动列表来重叠、定位以及对齐子 Widget。还将了解如何使用像 SliverSafeArea、SliverAppBar、SliverList、SliverGrid 等的 Sliver Widget 来实现 CustomScrollView，以便创建像视差动画这样的自定义滚动效果。

第 10 章：构建布局——将学习如何嵌套 Widget 以便构建专业的布局。嵌套这一概念是创建优美布局的主要组成部分，它也被称为组合。基础和复杂布局都主要基于纵向或横向 Widget，或者基于两者的组合。该章的目标是创建一个日记条目页面用于呈现详细信息，其中包括页眉图片、标题、日记详情、天气、地址、标签以及页脚图片。为了布局该页面，需要使用各种 Widget，如 SingleChildScrollView、SafeArea、Padding、Column、Row、Image、Divider、Text、Icon、SizedBox、Wrap、Chip 以及 CircleAvatar。

第 11 章：应用交互性——将学习如何使用手势为应用增加交互性。在移动应用中，手势是监听用户交互的核心，而充分利用手势则可为应用带来极佳的 UX。不过，不能表达一项操作的手势的滥用也会造成糟糕的 UX。读者将了解如何使用 GestureDetector 手势，如触碰、双击、长按、拖曳、垂直拖动、水平拖动以及缩放；还将了解如何使用 Draggable Widget 来拖动 DragTarget Widget 以便创建拖曳效果从而改变 Widget 的颜色。该章将介绍如何实现 InkWell 与 InkResponse Widget 以便响应触控以及可视化展示波纹动画效果，还会介绍如何通过拖曳来关闭 Dismissible Widget。读者将学习如何使用 Transform Widget 和 Matrix4 类来缩放与移动 Widget。

第 12 章：编写平台原生代码——某些情况下，需要访问特定的 iOS 或 Android API 功能。读者将学习如何使用平台通道在 Flutter 应用与主机平台之间发送和接收消息。该章将介绍如何使用 MethodChannel 从 Flutter 应用(客户端)发送消息，以及如何使用 iOS 的 FlutterMethodChannel 与 Android 的 MethodChannel 来接收调用(主机端)和发送回结果。

第 III 部分：创建可用于生产环境的应用

本书最后四章将介绍更高级的领域并且做好将示例应用发布到生产环境的准备。

第 13 章：使用本地持久化保存数据——该章将介绍如何构建一个日记应用。读者将了解如何使用 JSON 文件格式在应用启动运行时就开始持久化保存数据并将文件保存到本地 iOS 和 Android 文件系统。JavaScript 对象表示法(JavaScript Object Notation，JSON)是一种通用开放标准以及独立于语言的文件数据格式，其好处在于可提供人类可读的文本。该章将讲解如何创建数据库类以便写入、读取和序列化 JSON 文件，还将讲解如何格式化列表以及根据日期对其进行排序。

在移动应用中，在处理过程中不阻塞 UI 是非常重要的。该章将介绍如何使用 Future 类以及 FutureBuilder Widget，还将介绍如何呈现一个日期选择日历、校验用户录入数据以及在录入栏之间移动焦点。

读者还将学习如何使用 Dismissible Widget 通过在一个记录上拖动或释放来删除记录。为了根据日期对记录进行排序，该章将讲解如何使用 List().sort 方法以及 Comparator 函数。为了在页面之间进行导航，需要使用 Navigator Widget，该章还将讲解如何使用 CircularProgressIndicator Widget 来展示运行中的操作。

第 14 章：添加 Firebase 和 Firestore 后端——该章和第 15 章及第 16 章将使用之前几章讲解的技术与一些新概念，并将它们结合起来使用以便创建一个可用于生产环境的记录心情的日记应用。在可用于生产环境的应用中，我们应该如何结合使用之前所学到的知识，从而通过仅重绘数据发生变更的 Widget 来提升性能、在页面之间和 Widget 树传递状态、处理用户身份验证凭据、在设备和云之间同步数据、创建用于处理移动应用和 Web 应用之间独立于平台的逻辑类呢？这些正是最后三章的重点所在，读者将学习如何应用之前所了解的技术以及新的重要概念和技术来开发可用于生产环境的移动应用。在这最后三章中，读者将学习如何实现应用范围内以及本地的状态管理，并通过实现业务逻辑组件(BLoC)模式来最大化平台代码共享。

该章将介绍如何使用身份验证并使用 Google 的 Firebase 后端服务器基础设施、Firebase Authentication 和 Cloud Firestore 将数据持久化到云端数据库。读者将了解到，Cloud Firestore 是一个 NoSQL 文档数据库，可使用移动端和 Web 应用的离线支持来存储、查询和同步数据。我们将能在多个设备之间同步数据。读者将学习如何设置和构建无服务应用。

第 15 章：为 Firestore 客户端应用添加状态管理——该章将继续编辑第 14 章中创建的记录心情的日记应用。该章将介绍如何创建应用范围内的以及本地的状态管理，其中要使用 InheritedWidget 类作为提供程序以便在 Widget 和页面之间管理与传递 State。

该章将介绍如何使用 BLoC 模式来创建 BLoC 类，例如管理对于 Firebase 身份验证和 Cloud Firestore 数据库服务类的访问。该章还会介绍如何使用 InheritedWidget 类在 BLoC 和页面之间

传递引用。还将介绍如何使用一种响应式方法，这是通过使用 StreamBuilder、StreamController 以及 Stream 来填充和刷新数据而实现的。

该章将讲解如何创建服务类来管理 Firebase 身份验证 API 以及 Cloud Firestore 数据库 API。还要创建和利用抽象类来管理用户凭据。读者将学习如何创建一个数据模型类来处理 Cloud Firestore QuerySnapshot 到各个记录的映射问题。读者将了解如何创建一个类以便根据所选心情来管理心情图标列表、描述和图标旋转位置。读者还将使用 intl 包并学习如何创建一个日期格式化类。

第 16 章：为 Firestore 客户端应用页面添加 BLoC——该章将继续编辑在第 14 章中创建的记录心情的日记应用以及在第 15 章中创建的附加功能。

该章将介绍如何将 BLoC、服务、提供程序、模型和工具类应用到 UI Widget 页面。使用 BLoC 模式的好处在于，可将 UI Widget 和业务逻辑分开。该章将讲解如何使用依赖注入将服务类注入 BLoC 类中。通过使用依赖注入，BLoC 仍将独立于平台。这一概念极其重要，因为 Flutter 框架正在从移动端扩展到 Web、桌面和嵌入式设备。

读者将学习如何通过实现应用 BLoC 模式的类来执行应用范围内的身份验证状态管理。该章将介绍如何创建 Login 页面，其中要实现 BLoC 模式类以便验证电子邮箱、密码和用户凭据。还将介绍如何通过实现提供程序类(InheritedWidget)以便在页面和 Widget 树之间传递状态。读者将学习如何修改首页以便实现和创建 BLoC 模式类来处理登录凭据校验、创建日记条目列表以及添加和删除各个条目。还将学习如何创建日记编辑页面，它实现了 BLoC 模式类以便添加、修改和保存已有条目。

遵循本书进行练习的前提

读者需要安装 Flutter 框架和 Dart 以便创建示例项目。本书使用 Android Studio 作为主要开发工具，并且所有项目都是面向 iOS 和 Android 编译的。为编译 iOS 应用，读者需要一台安装了 Xcode 的 Mac 计算机。可使用其他编辑器，如 Microsoft Visual Studio Code 或 IntelliJ IDEA。对于最后一个重要项目，读者需要创建一个免费的 Google Firebase 账户，以便利用云端身份验证和数据同步，其中包括离线支持。

内容格式约定

为了帮助读者能够充分理解内容文本，并且持续跟随本书的讲解步骤，本书使用了若干内容格式约定。

试一试

(1) 这些练习由一系列编号步骤构成。

(2) 读者可使用自己的数据库并遵循这些步骤进行处理。

示例说明

在每一个“试一试”的结尾处，都会详细阐释所输入的代码。

代码的呈现有两种不同方式：

(1) 大多数示例代码都显示为等宽字体，不加粗。

(2) 对于在上下文中特别重要的代码，或在以前的代码片段的基础上修改的代码，则显示为粗体。

目　录

第 I 部分　Flutter 编程基础

第 1 章　Flutter 入门 ……3
1.1　Flutter 简介……4
1.2　理解 Widget 生命周期事件……5
1.2.1　StatelessWidget生命周期……5
1.2.2　StatefulWidget生命周期……6
1.3　理解 Widget 树和 Element 树 ……8
1.3.1　StatelessWidget和Element树 ……9
1.3.2　StatefulWidget和Element树 …… 10
1.4　安装 Flutter SDK……13
1.4.1　在macOS上进行安装 …… 13
1.4.2　在Windows上进行安装…… 15
1.4.3　在Linux上进行安装 …… 17
1.5　配置 Android Studio 编辑器……19
1.6　本章小结 ……20
1.7　本章知识点回顾 ……20

第 2 章　创建一个 Hello World 应用 ……23
2.1　设置项目 ……23
2.2　使用热重载 ……27
2.3　使用主题将应用样式化 ……30
2.3.1　使用全局应用主题…… 30
2.3.2　将主题用于应用的局部…… 32
2.4　理解 StatelessWidget 和 StatefulWidget……34
2.5　使用外部包 ……36
2.5.1　搜索包…… 36
2.5.2　使用包…… 37
2.6　本章小结 ……38
2.7　本章知识点回顾 ……38

第 3 章　学习 Dart 基础知识…… 39
3.1　为何使用 Dart？ ……39
3.2　代码注释……40
3.3　运行 main()入口点 ……41
3.4　变量引用……41
3.5　变量声明……42
3.5.1　数字…… 43
3.5.2　String…… 43
3.5.3　Boolean …… 43
3.5.4　List …… 44
3.5.5　Map …… 44
3.5.6　Runes…… 45
3.6　使用运算符……45
3.7　使用流程语句……47
3.7.1　if和else…… 47
3.7.2　三元运算符…… 48
3.7.3　for循环 …… 48
3.7.4　while和do-while…… 49
3.7.5　while和break …… 50
3.7.6　continue…… 50
3.7.7　switch和case…… 51
3.8　使用函数……52
3.9　导入包……53
3.10　使用类……54
3.10.1　类继承…… 57
3.10.2　类混合…… 57
3.11　实现异步编程 ……58
3.12　本章小结……59
3.13　本章知识点回顾……60

第 4 章 创建一个初学者项目模板 ……61
4.1 创建和组织文件夹与文件…… 61
4.2 结构化 Widget…… 64
4.3 本章小结 …… 69
4.4 本章知识点回顾…… 70

第 5 章 理解 Widget 树……71
5.1 Widget 介绍…… 71
5.2 构建完整的 Widget 树 …… 73
5.3 构建浅层 Widget 树 …… 80
5.3.1 使用常量进行重构……80
5.3.2 使用方法进行重构……81
5.3.3 使用Widget类进行重构 ……87
5.4 本章小结 …… 95
5.5 本章知识点回顾…… 95

第 II 部分 充当媒介的 Flutter：具象化一个应用

第 6 章 使用常用 Widget ……99
6.1 使用基础 Widget…… 99
6.1.1 SafeArea…… 103
6.1.2 Container…… 104
6.1.3 Text …… 108
6.1.4 RichText…… 109
6.1.5 Column……111
6.1.6 Row…… 112
6.1.7 Button…… 117
6.2 使用图片和图标…… 129
6.2.1 AssetBundle …… 129
6.2.2 Image …… 129
6.2.3 Icon …… 131
6.3 使用装饰 …… 134
6.4 使用 Form Widget 验证文本框 …… 139
6.5 检查设备方向…… 143
6.6 本章小结 …… 150
6.7 本章知识点回顾…… 150

第 7 章 为应用添加动画效果 ……151
7.1 使用 AnimatedContainer…… 151
7.2 使用 AnimatedCrossFade……155
7.3 使用 AnimatedOpacity……160
7.4 使用 AnimationController……164
7.5 本章小结 ……175
7.6 本章知识点回顾 ……176

第 8 章 创建应用的导航 ……177
8.1 使用 Navigator ……178
8.2 使用 Hero(飞行)动画……189
8.3 使用 BottomNavigationBar……194
8.4 使用 BottomAppBar……201
8.5 使用 TabBar 和 TabBarView ……205
8.6 使用 Drawer 和 ListView……211
8.7 本章小结 ……221
8.8 本章知识点回顾 ……222

第 9 章 创建滚动列表和效果 ……223
9.1 使用 Card ……223
9.2 使用 ListView 和 ListTile ……225
9.3 使用 GridView ……232
9.3.1 使用GridView.count …… 234
9.3.2 使用GridView.extent…… 235
9.3.3 使用GridView.builder…… 236
9.4 使用 Stack ……240
9.5 使用 Sliver(薄片)自定义 CustomScrollView ……247
9.6 本章小结 ……256
9.7 本章知识点回顾 ……256

第 10 章 构建布局 ……257
10.1 布局的概要视图 ……257
10.1.1 天气区域布局…… 259
10.1.2 标签布局…… 259
10.1.3 页脚图片布局…… 260
10.1.4 最终布局…… 260
10.2 创建布局 ……261
10.3 本章小结 ……269
10.4 本章知识点回顾 ……269

第 11 章 应用交互性……271
11.1 设置 GestureDetector：基本处理 ….271

11.2 实现 Draggable 和 DragTarget Widget……278
11.3 使用 GestureDetector 检测移动和缩放……282
11.4 使用 InkWell 和 InkResponse 手势……293
11.5 使用 Dismissible Widget……299
11.6 本章小结……306
11.7 本章知识点回顾……307

第 12 章 编写平台原生代码……309
12.1 理解平台通道……309
12.2 实现客户端平台通道应用……310
12.3 实现 iOS 主机端平台通道……315
12.4 实现 Android 主机端平台通道……319
12.5 本章小结……323
12.6 本章知识点回顾……324

第 III 部分 创建可用于生产环境的应用

第 13 章 使用本地持久化保存数据……327
13.1 理解 JSON 格式……328
13.2 使用数据库类来写入、读取和序列化 JSON……330
13.3 格式化日期……331
13.4 对日期列表进行排序……332
13.5 使用 FutureBuilder 检索数据……333
13.6 构建日记应用……335
13.6.1 添加日记数据库类……339
13.6.2 添加日记条目页……345
13.6.3 完成日记主页面……362
13.7 本章小结……377
13.8 本章知识点回顾……378

第 14 章 添加 Firebase 和 Firestore 后端……381
14.1 Firebase 和 Cloud Firestore 是什么？……382
14.1.1 对Cloud Firestore进行结构化和数据建模……383
14.1.2 查看Firebase身份验证能力……385
14.1.3 查看Cloud Firestore安全规则……387
14.2 配置 Firebase 项目……388
14.3 添加一个 Cloud Firestore 数据库并实现安全规则……395
14.4 构建客户端日记应用……398
14.4.1 将身份验证和Cloud Firestore包添加到客户端应用……399
14.4.2 为客户端应用添加基础布局……405
14.4.3 为客户端应用添加类……409
14.5 本章小结……412
14.6 本章知识点回顾……413

第 15 章 为 Firestore 客户端应用添加状态管理……415
15.1 实现状态管理……416
15.1.1 实现一个抽象类……417
15.1.2 实现InheritedWidget……419
15.1.3 实现模型类……420
15.1.4 实现服务类……421
15.1.5 实现BLoC模式……422
15.1.6 实现StreamController、Stream、Sink和StreamBuilder……423
15.2 构建状态管理……425
15.2.1 添加Journal模型类……427
15.2.2 添加服务类……428
15.2.3 添加Validators类……435
15.2.4 添加BLoC模式……436
15.3 本章小结……455
15.4 本章知识点回顾……455

第 16 章 为 Firestore 客户端应用页面添加 BLoC……457
16.1 添加登录页……458
16.2 修改主页面……464
16.3 修改主页……468
16.4 添加编辑日记页面……476
16.5 本章小结……489
16.6 本章知识点回顾……490

第 I 部分

Flutter 编程基础

- 第 1 章　Flutter 入门
- 第 2 章　创建一个 Hello World 应用
- 第 3 章　学习 Dart 基础知识
- 第 4 章　创建一个初学者项目模板
- 第 5 章　理解 Widget 树

第 1 章

Flutter 入门

本章内容

- Flutter 框架是什么？
- Flutter 能带来哪些好处？
- Flutter 和 Dart 如何共同发挥作用？
- Flutter Widget 是什么？
- Element 是什么？
- RenderObject 是什么？
- 有哪些类型的 Flutter Widget 可用？
- StatelessWidget 和 StatefulWidget 的生命周期是什么？
- Widget 树和 Element 树如何共同发挥作用？
- 如何安装 Flutter SDK？
- 如何在 macOS 上安装 Xcode 以及如何在 macOS、Windows 和 Linux 上安装 Android Studio？
- 如何配置编辑器？
- 如何安装 Flutter 和 Dart 插件？

本章将介绍 Flutter 框架的后台运行机制。Flutter 使用 Widget 来创建用户界面(User Interface，UI)，而 Dart 则是用于开发应用程序的语言。理解 Flutter 处理和实现这些 Widget 的方式，将有助于我们对应用进行架构设计。

本章还会讲解如何在 macOS、Windows 和 Linux 上安装 Flutter。我们要配置 Android Studio 从而安装 Flutter 插件，以便运行、调试和使用热重载。我们要安装 Dart 插件用于代码分析、代码校验和代码补全。

1.1 Flutter 简介

Flutter 是 Google 的移动端 UI 框架，用于构建现代、原生且反应式的 iOS 和 Android 应用。Google 还致力于开发 Flutter 桌面嵌入以及用于 Web 的 Flutter(Hummingbird)和用于嵌入式设备(树莓派、智能家居设备、汽车等)的 Flutter。Flutter 是一个开源项目，它托管在 GitHub 上，Google 及其社区都对其做出了贡献。Flutter 使用了 Dart，这是一种现代的面向对象语言，它可以编译成原生 ARM 代码以及可用于生产环境的 JavaScript 代码。Flutter 使用了 Skia 2D 渲染引擎，该引擎兼容各种类型的硬件和软件平台；Google Chrome、Chrome OS、Android、Mozilla Firefox、Firefox OS 等也都使用了该引擎。Skia 由 Google 赞助和管理，根据 BSD Free Software License(BSD 自由软件许可)，任何人都可以使用它。Skia 使用了基于 CPU 的路径渲染，还支持 OpenGL ES2 加速的后端。

Dart 是用于开发 Flutter 应用的语言，第 3 章中将更详尽地介绍它。Dart 会被预先编译为原生代码，从而让 Flutter 应用得以快速运行。Dart 可以是即时编译，从而让它可以快速呈现代码变更，比如通过 Flutter 的有状态热重载特性来呈现。

Flutter 使用 Dart 来创建用户界面，这样就不必使用像 Markup 标记语言这样的其他语言或可视化设计器了。Flutter 是声明式的；换句话说，Flutter 构建了 UI 来反映应用的状态。当状态(数据)发生变化时，就会重绘 UI，并且 Flutter 会构造一个新的 Widget 实例。本章的 1.3 节将介绍创建 Widget 树和 Element 树时如何配置和挂载 Widget，但就底层而言，渲染树(第三种树)会使用 RenderObject，它会计算和实现基础布局以及绘制协议(将不必与渲染树或 RenderObject 直接交互，因此本书不会进一步讨论它们)。

Flutter 运行速度很快，并且对于性能较高的设备，渲染速度可以是 60fps(帧/秒)和 120fps。渲染速度越高，动画和过渡效果就越平顺。

用 Flutter 开发的应用都是由单个代码库构建的，它们会被编译为原生 ARM 代码，使用图形处理器(GPU)，并可经由平台通道的通信来访问特定 iOS 和 Android API(如 GPS 定位、图片库)。第 12 章将更详尽地介绍平台通道。

Flutter 为开发人员提供了工具以便创建美观且看上去很专业的应用，还提供了自定义应用各个方面的能力。我们能为 UI 添加平滑的动画、手势检测以及水波纹反馈行为。Flutter 应用可以展现出 iOS 和 Android 平台的原生性能。在开发过程中，Flutter 使用了热重载以便在为了添加新功能或调整现有功能而修改源代码时可以毫秒级速度刷新运行中的应用。使用热重载是在保持运行中的应用状态、数据值不变的同时在模拟器或设备中观察代码变更效果的绝佳方式。

定义 Widget 和 Element

Flutter UI 通过使用来自现代反应式框架的 Widget 来实现。Flutter 使用其自有的渲染引擎来绘制 Widget。第 5 章将介绍 Widget，而第 6 章将介绍如何实现 Widget。

读者可能会问，Widget 是什么？可将 Widget 比作乐高积木；通过将各个积木堆叠在一起，我们就能创造一个物体，而通过添加不同种类的积木，我们就能修改该物体的外观和行为。Widget 就是 Flutter 应用的积木，并且每个 Widget 都是用户界面的一个不可变声明。换

句话说，Widget 就是用于 UI 不同部分的配置(指令)。将各个 Widget 放在一起就会形成 Widget 树。例如，假设一名建筑师要绘制一栋房子的设计蓝图；像这栋房子的墙、窗户和门在内的所有物体都是 Widget，并且所有这些共同构成了房子，也就是我们所说的应用。

由于 Widget 是 UI 的各个配置部分，并且它们共同构成了 Widget 树，那么 Flutter 如何使用这些配置呢？Flutter 使用 Widget 作为配置来构建每个 Element，这意味着 Element 就是挂载(渲染)到界面上的 Widget。挂载到界面上的各个 Element 就构成了 Element 树。下一节将介绍与 Widget 树和 Element 树有关的更多信息。第 5 章将详细讲解 Widget 树的操作。

以下是对各种可用的 Widget 的简要介绍：

- 具有结构化 Element 的 Widget，比如列表、网格、文本和按钮。
- 具有输入 Element 的 Widget，比如表单、表单字段和键盘监听器。
- 具有样式 Element 的 Widget，比如字体类型、字号、粗细、颜色、边框和阴影。
- 对 UI 进行布局的 Widget，比如行、列、层叠、定位居中和内边距。
- 具有交互式 Element 的 Widget，这些交互式 Element 对应着触摸、手势、拖动和滑动删除。
- 具有动画和动作 Element 的 Widget，如 Hero 动画、动画容器、动画淡入淡出、透明渐变、旋转、缩放、大小、平移和不透明度。
- 具有像资源、图片和图标这样的 Element 的 Widget。
- 可以嵌套在一起创建所需 UI 的 Widget。
- 可以自行创建的自定义 Widget。

1.2　理解 Widget 生命周期事件

编程领域中，不同的生命周期事件通常是以线性模式发生的，每个阶段完成时逐个执行。这一节将介绍 Widget 生命周期事件及其用途。

为了构建 UI，需要使用两种主要类型的 Widget，即 StatelessWidget(无状态 Widget)和 StatefulWidget(有状态 Widget)。当值(状态)不会发生变化时，就要使用 StatelessWidget，而当值(状态)发生变化时，则要使用 StatefulWidget。第 2 章将详细介绍何时使用 StatelessWidget 或 StatefulWidget。每个无状态或有状态的 Widget 都有一个带有 BuildContext 的 build 方法，它会处理 Widget 在 Widget 树中的位置。BuildContext 对象实际上是 Element 对象，它们是 Widget 在树中位置的实例。

1.2.1　StatelessWidget 生命周期

StatelessWidget 是基于其自身配置来构建的，并且不会动态变化。例如，显示带有描述的图片的界面并且该界面不会发生变化。StatelessWidget 是用一个类来声明的，第 3 章将介绍这些类。可从三个不同场景中调用 StatelessWidget 的 build(UI 部分)方法。在一开始创建 Widget 时就会调用该方法，当该 Widget 的父 Widget 发生变化以及 InheritedWidget 发生变化时，也会调用该方法。第 15 章将介绍如何实现 InheritedWidget。

以下示例代码显示了 StatelessWidget 的基础结构；图 1.1 显示了 Widget 的生命周期。

```
class JournalList extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return Container();
    }
}
```

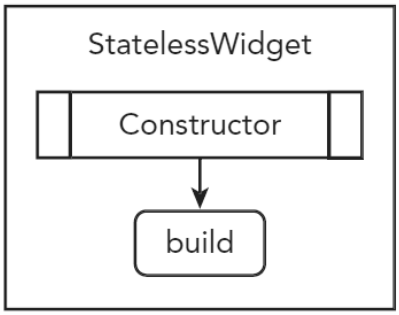

图 1.1 StatelessWidget 生命周期

1.2.2 StatefulWidget 生命周期

StatefulWidget 是基于其自有配置来构建的，但可动态变更。例如，界面显示了一个图标和一段描述，但值会根据用户的交互而变化，比如选择另一个图标或描述。这类 Widget 具有可以随时间而变化的可变状态。StatefulWidget 是用两个类声明的，即 StatefulWidget 类和 State 类。StatefulWidget 类会在 Widget 配置发生变化时被重新构建，而 State 类则可以保持不变(被保留)，从而提高性能。例如，当状态发生变化时，就会重建 Widget。如果从树中移除 StatefulWidget，然后在未来某个时候将其插入回去，就会创建一个新的 State 对象。注意，在某些特定情况和限制下，我们可使用 GlobalKey(跨整个应用的唯一键)来重用(而非重建)State 对象；不过，全局键成本很高，除非需要它们，否则大家可能不希望考虑使用它们。我们要调用 setState()方法来通知框架，这个对象发生了变化，并且调用(排定)了该 Widget 的 build 方法。我们要在 setState()方法内设置新的状态值。第 2 章将介绍如何调用 setState()方法。

下面这个示例显示了 StatefulWidget 基础结构，而图 1.2 显示了该 Widget 的生命周期。其中有两个类，JournalEdit StatefulWidget 类和_JournalEditState 类。

```
class JournalEdit extends StatefulWidget {
    @override
    _ JournalEditState createState() => _JournalEditState();
    }
    class _JournalEditState extends State<JournalEdit> {
      @override
      Widget build(BuildContext context) {
          return Container();
      }
}
```

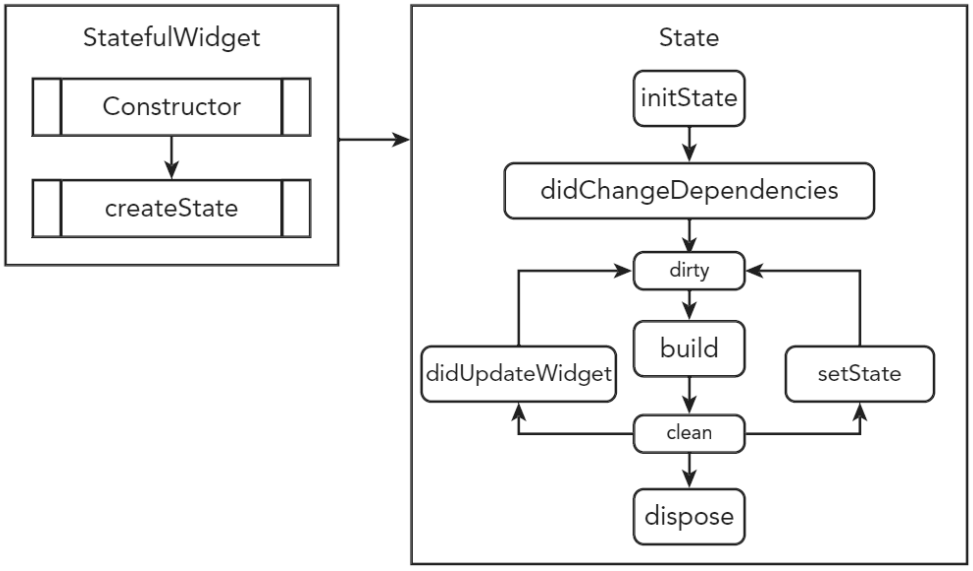

图 1.2　StatefulWidget 生命周期

我们可以重写 StatefulWidget 的不同部分，以便在该 Widget 生命周期的不同时间点自定义和操作数据。表 1.1 显示了 StatefulWidget 的一些主要重写，大部分时候我们都会用到 initState()、dispose()和 didChangeDependencies()方法。我们从始至终都要使用 build()方法来构建 UI。

表 1.1　StatefulWidget 生命周期

方法	描述	示例代码

initState()	当这个对象被插入树中时被调用一次	@override
		void initState () {
		super. initState ();
		print (' initState ');
		}

dispose()	当这个对象从树中被永久移除时调用	@override
		void dispose () {
		print (' dispose ');
		super. dispose ();
		}

didChangeDependencies()	当这个 State 对象发生变化时调用	@override
		void didChangeDependencies () {
		super. didChangeDependencies ();
		print (' didChangeDependencies ');
		}

(续表)

方法	描述	示例代码

didUpdateWidget(Contacts oldWidget)	当 Widget 配置发生变化时调用	@override
		void didUpdateWidget (Contacts oldWidget) {
		super. didUpdateWidget (oldWidget);
		print (' didUpdateWidget: $oldWidget ');
		}

deactivate()	当这个对象从树中被移除时调用	@override
		void deactivate () {
		print('deactivate');
		super.deactivate();
		}

build(BuildContext context)	可以被多次调用以构建 UI，BuildContext 会处理这个 Widget 在树中的位置	@override
		Widget build (BuildContext context) {
		print (' build ');
		return Container ();
		}

setState()	告知框架这个对象的状态已经发生变化，以便安排调用这个 State 对象的 build	setState(() {
		name = _newValue;
		});

1.3 理解 Widget 树和 Element 树

上一节已经介绍了，Widget 包含创建 UI 的指令，当我们将各个 Widget 组合(嵌套)在一起时，它们就会构成 Widget 树。Flutter 框架使用 Widget 作为被挂载(渲染)在界面上的每个 Element 的配置。所挂载的显示在界面上的各个 Element 构成 Element 树。目前有两种树，即具有 Widget 配置的 Widget 树，以及代表界面上所渲染 Widget 的 Element 树(见图 1.3)。

当应用启动时，main()函数会调用 runApp()方法，通常会采用 StatelessWidget 作为参数，并且被挂载作为该应用的根 Element。Flutter 框架会处理所有 Widget，并且每个对应的 Element 都会被挂载。

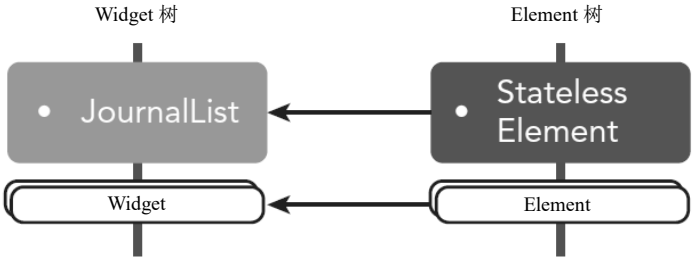

图 1.3　Widget 树和 Element 树

下面是一段示例代码，这段代码会启动一个 Flutter 应用，而 runApp()方法充当了 MyApp StatelessWidget，这意味着主应用本身就是一个 Widget。正如我们所见，Flutter 中的一切都是 Widget。

```
void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
   @override
   Widget build(BuildContext context) {
     return MaterialApp(
        title: 'Flutter App',
        theme: ThemeData(
           primarySwatch: Colors.blue,
        ),
        home: MyHomePage(title: 'Home'),
     );
   }
}
```

Element 具有指向 Widget 的引用，并且它们要负责比较 Widget 的差异。如果一个 Widget 负责构建各个子 Widget，就要为每个子 Widget 创建 Element。当看到 BuildContext 对象的使用时，它们就是 Element 对象。为了避免对于 Element 对象的直接操作，我们要转而使用 BuildContext 接口。Flutter 框架使用 BuildContext 对象来防止我们操作 Element 对象。也就是说，我们要使用 Widget 来创建 UI 布局，但最好弄清楚 Flutter 框架是如何构建的，以及其背后的运行机制是怎样的。

正如之前所述，还有第三种树，被称为渲染树，它是继承自 RenderObject 的一种低级别布局和绘制系统。RenderObject 会计算和实现基础布局与绘制协议。不过，我们不需要直接与渲染树交互，而要使用 Widget 与之交互。

1.3.1　StatelessWidget 和 Element 树

StatelessWidget 具有创建无状态 Element 的配置。每个 StatelessWidget 都具有一个对应的

无状态 Element。Flutter 框架会调用 createElement 方法(创建一个实例)，进而创建该无状态 Element 并将其挂载到 Element 树。也就是说，Flutter 框架会对 Widget 进行请求以便创建一个 Element，然后将该 Element 挂载(添加)到 Element 树。每个 Element 都包含指回 Widget 的引用。Element 会调用 Widget 的 build 方法以检查各个子 Widget，而每个子 Widget(如 Icon 或 Text)都会创建其自己的 Element，并且这些 Element 也会被挂载到 Element 树。这一过程会生成两棵树：Widget 树和 Element 树。

图 1.4 显示了 JournalList StatelessWidget，它具有表示 Widget 树的 Row、Icon 和 Text Widget。Flutter 框架会要求每个 Widget 创建 Element，并且每个 Element 都具有指回 Widget 的引用。Widget 树上的每个 Widget 都会经历这一过程，并会创建 Element 树。Widget 包含构建挂载到界面上的 Element 的指令。注意，开发人员创建的是 Widget，而 Flutter 框架会处理 Element 挂载以及 Element 树的创建。

```
// Simplified sample code
class JournalList extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return Row(
          children: <Widget>[
              Icon(),
              Text(),
          ],
        );
    }
}
```

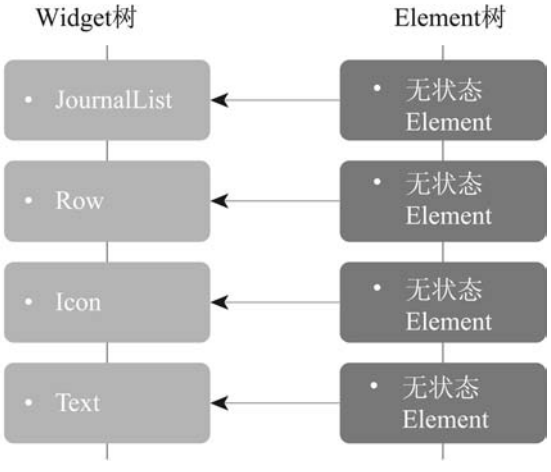

图 1.4 StatelessWidget Widget 树和 Element 树

1.3.2 StatefulWidget 和 Element 树

StatefulWidget 具有创建有状态 Element 的配置。每个 StatefulWidget 都有一个对应的有

状态 Element。Flutter 框架会调用 createElement 方法来创建有状态 Element，并且该有状态 Element 会被挂载到 Element 树。由于这是一个 StatefulWidget，因此该有状态 Element 表示，Widget 会通过调用 StatefulWidget 类的 createState 方法来创建一个 State 对象。

这样，该有状态 Element 就具有了指向 State 对象和位于 Element 树指定位置的 Widget 的引用。有状态 Element 会调用 State 对象 Widget 的 build 方法来检查各个子 Widget，并且每个子 Widget 都会创建自己的 Element 并被挂载到 Element 树。这一过程会生成两棵树：Widget 树和 Element 树。注意，如果一个显示状态的子 Widget(journal note，日记记录)是一个像 Text Widget 这样的 StatelessWidget，那么为这个 Widget 创建的 Element 就是一个无状态的 Element。State 对象维护着指向 Widget(StatefulWidget 类)的引用，还会用最新值来处理 Text Widget 的构造。图 1.5 显示了 Widget 树、Element 树和 State 对象。注意，有状态 Element 具有指向 StatefulWidget 和 State 对象的引用。

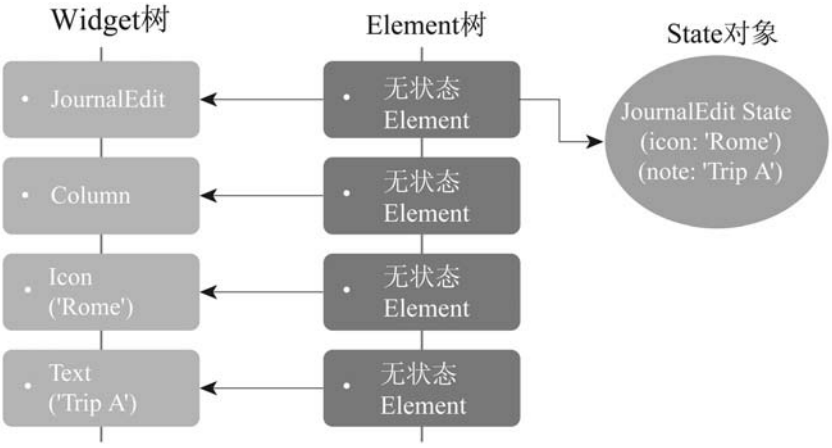

图 1.5　StatefulWidget Widget 树、Element 树以及 State 对象

要使用新数据更新 UI，可以调用 1.2.2 节中介绍过的 setState()方法。要设置新的数据(属性/变量)值，可以调用 setState()方法来更新 State 对象，而 State 对象会将 Element 标记为脏(已变更)并且造成 UI 被更新(编排)。有状态 Element 会调用 State 对象的 build 方法来重建各个子 Widget。根据新的状态值，会创建一个新的 Widget，同时会移除旧 Widget。

例如，有一个 StatefulWidget JournalEntry 类，并且在 State 对象类中调用了 setState()方法，通过将 note 变量的值设置为'Trip B'来将 Text Widget 描述从'Trip A'变更为'Trip B'。State 对象 note 变量被更新为'Trip B'值，然后 State 对象将 Element 标记为脏，并且 build 方法会重建 UI 子 Widget。使用新的'Trip B'值创建了新的 Text Widget，而值为'Trip A'的旧 Text Widget 则会被移除(见图 1.6)。

```
// Simplified sample code
class JournalEdit extends StatefulWidget {
   @override
   _JournalEditState createState() => _JournalEditState();
}
```

```
class _JournalEditState extends State<JournalEdit> {
  String note = 'Trip A';

  void _onPressed() {
     setState(() {
        note = 'Trip B';
     });
  }
@override
Widget build(BuildContext context) {
    return Column(
      children: <Widget>[
         Icon(),
         Text('$note'),
         FlatButton(
            onPressed: _onPressed,
         ),
      ],
    );
  }
}
```

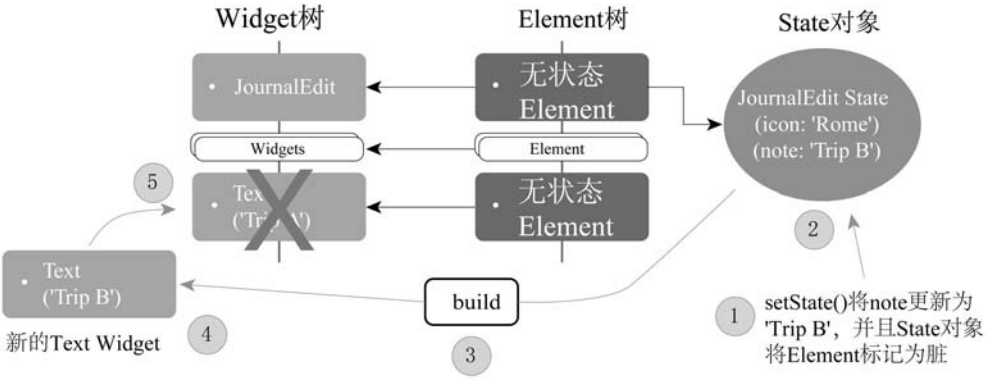

图 1.6　更新状态处理过程

由于旧的和新的 Widget 都是 Text Widget，因此现有 Element 会更新其引用以指向新的 Widget，而 Element 会保留在 Element 树中。Text Widget 是一个无状态的 Widget，并且其对应的 Element 也是一个无状态的 Element；尽管 Text Widget 已经被替换了，但其状态会保持不变(被保留)。State 对象具有较长的生命周期跨度，并且只要新 Widget 的类型与旧 Widget 的类型相同，则其仍旧会附加到 Element 树。

我们继续处理前面的示例；值为'Trip A'的旧 Text Widget 已被移除并被值为'Trip B'的新 Text Widget 替换掉了。由于旧的和新的 Widget 都是同一类型的 Text Widget，因此 Element 会留存在 Element 树上，并且具有更新后的指向新的 Text 'Trip B' Widget 的引用(见图 1.7)。

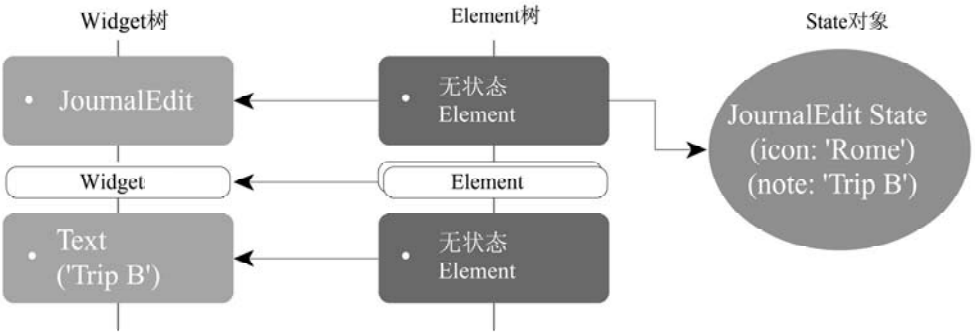

图 1.7　Widget 树和 Element 树更新后的状态

1.4　安装 Flutter SDK

安装 Flutter SDK 需要下载 Flutter 网站处 SDK 的最新版本，撰写本书时的最新版本是 1.5.4(读者使用的版本可能更高)。这一章包含在 macOS、Windows 和 Linux 上进行安装的内容(注意，针对 iOS 平台的编译需要一台 Mac 电脑和 Xcode，也就是 Apple 的开发环境)。不要被后面介绍的一系列步骤所困扰；这些步骤仅在一开始安装时执行。

我们将使用 Terminal 窗口来运行安装和配置命令。

1.4.1　在 macOS 上进行安装

在开始安装前，我们需要确保所用的 Mac 至少满足以下硬件和软件要求。

1. 系统要求

- macOS(64 位)
- 700MB 硬盘空间(不包括用于集成开发环境和其他工具的硬盘空间)
- 以下命令行工具：
  - ➤ Bash
  - ➤ mkdir
  - ➤ rm
  - ➤ git
  - ➤ curl
  - ➤ unzip
  - ➤ which

2. 获取 Flutter SDK

Flutter 网站在线提供了最新的安装详情，地址是 https://flutter.dev/docs/get-started/install/macos/。在所用 Mac 的 Terminal 窗口中执行步骤(2)及后续步骤。

(1) 下载以下安装文件以获取 Flutter SDK 的最新发布版本，即 v1.7.8 或更高版本：

```
https://storage.googleapis.com/flutter_infra/releases/stable/macos/flutter
```

```
_macos_v1.7.8+hotfix.4-stable.zip.
```

(2) 使用 Terminal 窗口在期望位置提取文件。

```
cd ~/development
unzip ~/Downloads/flutter_macos_v1.7.8+hotfix.4-stable.zip
```

(3) 将 Flutter 工具添加到路径(pwd 表示当前的工作目录；对于我们而言，这个目录就是 development 文件夹)。

```
export PATH="$PATH:`pwd`/flutter/bin"
```

(4) 永久更新路径。

a. 获取步骤(3)中所用的路径。例如，用我们自己的开发文件夹路径替换 MacUserName。

```
/Users/MacUserName/development/flutter/bin
```

b. 打开或创建$HOME/.bash_profile。读者电脑上的文件路径或文件名可能与此不同。

c. 编辑.bash_profile(将打开一个命令行编辑器)。

d. 输入 nano .bash_profile。

e. 输入 export PATH=/Users/MacUserName/development/flutter/bin:$PATH。

f. 按下^X(Control+X)进行保存，按下 Y(Yes)进行确认，按下 Enter 接受文件名。

g. 关闭 Terminal 窗口。

h. 重新打开 Terminal 窗口并且输入 echo $PATH 来验证路径已经被添加。然后输入 flutter 以确保该路径已生效。如果无法识别该命令，则表示 PATH 出了问题。应该执行检查以确保路径中包含电脑的正确用户名。

3. 检查依赖项

在 Terminal 窗口中运行以下命令，以检查完成设置所需安装的依赖项。

```
flutter doctor
```

浏览生成的报告并且检查可能需要的其他软件。

4. iOS 设置：安装 Xcode

需要具有 Xcode 10.0 或更新版本的 Mac。

(1) 打开 App Store 并且安装 Xcode。

(2) 在 Terminal 窗口中运行 sudo xcode-select --switch/Applications/Xcode.app/Contents/Developer 以便配置 Xcode 命令行工具。

(3) 在终端中运行 sudo xcodebuild –license 以确认签署了 Xcode 许可协议。

5. Android 设置：安装 Android Studio

可访问 https://flutter.dev/docs/get-started/install/macos#android-setup 了解完整的安装详情。Flutter

需要 Android Studio 的完整安装以满足 Android 平台依赖。要牢记的是，可以在不同的编辑器(Visual Code 或 IntelliJ IDEA)中编写 Flutter 应用。

(1) 从 https://developer.android.com/studio/处下载和安装 Android Studio。

(2) 启动 Android Studio 并且遵循 Setup Wizard(安装向导)，该向导会安装 Flutter 需要的所有 Android SDK。如果该向导询问是否要导入之前的设置，则可以单击 Yes 按钮以使用当前设置，或单击 No 按钮以启用默认设置。

6. 设置 Android 模拟器

可在 https://developer.android.com/studio/run/managing-avds 处查看关于如何创建和管理虚拟设备的详细介绍。

(1) 在电脑上启用 VM 加速。https://developer.android.com/studio/run/emulator-acceleration 处提供了操作说明。

(2) 如果是首次安装 Android Studio，那么要访问 AVD Manager，就需要创建一个新项目。启动 Android Studio，单击 Start A New Android Studio Project，可以随意取一个名称并且接受默认设置。项目创建后，继续执行这些步骤。

现在，在可以访问 Android 子菜单之前，Android Studio 需要打开一个 Android 项目。

(1) 在 Android Studio 中，选择 Tools | Android | AVD Manager | Create Virtual Device。

(2) 选择所选用设备并且单击 Next。

(3) 为要模拟的 Android 版本选择 x86 或 x86_64 映像。

(4) 如可能，请确保选择 Hardware – GLES 2.0 来启用硬件加速，以便让 Android 模拟器更快运行。

(5) 单击 Finish 按钮。

(6) 为了检查该模拟器映像已经被正确安装，可选择 Tools | AVD Manager，然后单击 Run 按钮(播放图标)。

1.4.2　在 Windows 上进行安装

在开始安装前，我们需要确保所用的 Windows 至少满足以下硬件和软件要求。

1. 系统要求

- Windows 7 SP1 或更新版本(64 位)
- 400MB 硬盘空间(不包括用于集成开发环境和其他工具的硬盘空间)
- 以下命令行工具：
  - ➤ PowerShell 5.0 或更新版本
  - ➤ Git for Windows(可以在 Windows 命令提示符中使用的 Git)

2. 获取 Flutter SDK

https://flutter.dev/docs/get-started/install/windows/处提供了最新的安装详细说明。

(1) 下载以下安装文件以获取 Flutter SDK 的最新发布版本，在撰写本书时是 v1.7.8(读者

使用的版本可能更高)：

```
https://storage.googleapis.com/flutter_infra/releases/stable/windows/flutter_windows_v1.7.8+hotfix.4-stable.zip。
```

(2) 在期望的位置提取文件。

将 WindowsUserName 替换成我们自己的 Windows 用户名。不要将 Flutter 安装在需要提升权限的目录中，如 C:\Program Files\。我使用的是以下文件夹位置：

```
C:\Users\WindowsUserName\flutter
```

(3) 查看 C:\Users\WindowsUserName\flutter 并双击 flutter_console.bat 文件。

(4) 永久更新路径(以下是 Windows 10 的更新过程)。

a. 打开控制面板并且向下访问到 Desktop App | User Accounts | User Accounts | Change My Environment Variables。

b. 在 WindowsUserName 的 User Variables 下，选择 Path 变量并单击 Edit 按钮，如果 Path 变量缺失，则跳到子步骤 c。

c. 在 Edit 环境变量中，单击 New 按钮。

- 输入路径 C:\Users\WindowsUserName\flutter\bin。
- 单击 OK 按钮，然后关闭 Environment Variables 界面。

d. 在 WindowsUserName 的 User Variables 下，如果 Path 变量缺失，则要转而单击 New 按钮，并输入 Path 作为 Variable 名称，输入 C:\Users\WindowsUserName\flutter\bin 作为 Variable 值。

(5) 重启 Windows 以应用这些变更。

3. 检查依赖项

在 Windows 命令提示符中运行以下命令，以便检查是否具备完成设置需要安装的依赖项。

```
flutter doctor
```

浏览所生成的报告并检查可能需要安装的其他软件。

4. 安装 Android Studio

https://flutter.dev/docs/get-started/install/windows#android-setup 处提供了完整的安装详细说明。Flutter 需要 Android Studio 的完整安装以满足 Android 平台依赖。要牢记的是，可在不同的编辑器(如 Visual Code 或 IntelliJ IDEA)中编写 Flutter 应用。

(1) 从 https://developer.android.com/studio/处下载和安装 Android Studio。

(2) 启动 Android Studio 并且遵循 Setup Wizard(安装向导)，该向导会安装 Flutter 需要的所有 Android SDK。如果该向导询问是否要导入之前的设置，则可单击 Yes 按钮以使用当前设置，或单击 No 按钮以启用默认设置。

5. 设置 Android 模拟器

可在 https://developer.android.com/studio/run/managing-avds 处查看关于如何创建和管理虚拟设备的详细介绍。

(1) 在电脑上启用 VM 加速。https://developer.android.com/studio/run/emulator-acceleration 处提供了操作说明。

(2) 如果是首次安装 Android Studio，那么要访问 AVD Manager，就需要创建一个新项目。启动 Android Studio，单击 Start A New Android Studio 项目，可以随意取一个名称并且接受默认设置。项目创建后，继续执行这些步骤。

现在，在可以访问 Android 子菜单之前，Android Studio 需要打开一个 Android 项目。

(3) 在 Android Studio 中，选择 Tools | Android | AVD Manager | Create Virtual Device。

(4) 选择所选用设备并单击 Next 按钮。

(5) 为要模拟的 Android 版本选择 x86 或 x86_64 映像。

(6) 如有可能，请确保选择 Hardware – GLES 2.0 来启用硬件加速，以便让 Android 模拟器更快运行。

(7) 单击 Finish 按钮。

(8) 为了检查该模拟器映像已经被正确安装，可选择 Tools | AVD Manager，然后单击 Run 按钮(播放图标)。

1.4.3　在 Linux 上进行安装

在开始安装前，我们需要确保所用的 Linux 至少满足以下硬件和软件要求。

1. 系统要求

- Linux(64 位)
- 600MB 硬盘空间(不包括用于集成开发环境和其他工具的硬盘空间)
- 以下命令行工具：
  - ➢ Bash
  - ➢ curl
  - ➢ git 2.x
  - ➢ mkdir
  - ➢ rm
  - ➢ unzip
  - ➢ which
  - ➢ xz-utils
- mesa 包所提供的 libGLU.so.1 共享库(如 Ubuntu/Debian 上的 libglu1-mesa)

2. 获取 Flutter SDK

https://flutter.dev/docs/get-started/install/linux/处提供了最新的安装详细说明。

(1) 下载以下安装文件以获取 Flutter SDK 的最新发布版本，在撰写本书时是 v1.7.8(读者

使用的版本可能更高)：

```
https://storage.googleapis.com/flutter_infra/releases/stable/linux/flutter
_linux_v1.7.8+hotfix.4-stable.tar.xz。
```

(2) 使用 Terminal 窗口在期望的位置提取文件：

```
cd ~/development
tar xf ~/Downloads/flutter_linux_v1.7.8+hotfix.4-stable.tar.xz
```

(3) 将 Flutter 工具添加到路径(pwd 表示当前的工作目录；对于我们而言，这个目录就是 development 文件夹)。

```
export PATH="$PATH:`pwd`/flutter/bin"
```

(4) 永久更新路径。

a. 获取步骤(3)中所用的路径。例如，用我们自己的开发文件夹路径替换 PathToDev。

```
/PathToDev/development/flutter/bin
```

b. 打开或创建$HOME/.bash_profile。读者电脑上的文件路径或文件名可能与此不同。

c. 编辑.bash_profile(将打开一个命令行编辑器)。

e. 添加以下命令行并且确保用自己的路径替换 PathToDev。

```
export PATH="$PATH :/PathToDev/development/flutter/bin"
```

d. 运行 source $HOME/.bash_profile 以刷新当前窗口。

e. 在 Terminal 窗口中，输入 echo $PATH 以验证路径是否已被添加。然后输入 flutter 以确保该路径已生效。如果未能识别该命令，那么 PATH 一定出了问题。应该检查以确保使用正确的路径。

3. 检查依赖项

在 Terminal 窗口中运行以下命令以检查完成设置需要安装的依赖项。

```
flutter doctor
```

浏览生成的报告并检查可能需要的其他软件。

4. 安装 Android Studio

https://flutter.dev/docs/get-started/install/linux#android-setup 处提供了完整的安装详情。Flutter 需要 Android Studio 的完整安装以满足 Android 平台依赖。要牢记的是，可在不同的编辑器(如 Visual Code 或 IntelliJ IDEA)中编写 Flutter 应用。

(1) 从 https://developer.android.com/studio/处下载和安装 Android Studio。

(2) 启动 Android Studio 并且遵循 Setup Wizard(安装向导)，该向导会安装 Flutter 需要的所有 Android SDK。如果该向导询问是否要导入之前的设置，可以单击 Yes 按钮以使用当前

设置，或者单击 No 按钮以启用默认设置。

5. 设置 Android 模拟器

可以在 https://developer.android.com/studio/run/managing-avds 处查看关于如何创建和管理虚拟设备的详细介绍。

(1) 在电脑上启用 VM 加速。https://developer.android.com/studio/run/emulator-acceleration 处提供了操作说明。

(2) 如果是首次安装 Android Studio，那么要访问 AVD Manager，就需要创建一个新项目。启动 Android Studio，单击 Start A New Android Studio Project，可以随意取一个名称并且接受默认设置。项目创建后，继续执行这些步骤。

现在，在可以访问 Android 子菜单之前，Android Studio 需要打开一个 Android 项目。

(3) 在 Android Studio 中，选择 Tools | Android | AVD Manager | Create Virtual Device。

(4) 选择所选用设备并且单击 Next 按钮。

(5) 为要模拟的 Android 版本选择 x86 或 x86_64 映像。

(6) 如有可能，请确保选择 Hardware – GLES 2.0 来启用硬件加速，以便让 Android 模拟器更快运行。

(7) 单击 Finish 按钮。

(8) 为了检查该模拟器映像已经被正确安装，可以选择 Tools | AVD Manager，然后单击 Run 按钮(播放图标)。

1.5　配置 Android Studio 编辑器

我们要使用的编辑器就是 Android Studio。Android Studio 是 Google Android 操作系统的官方集成开发环境，是专门为 Android 开发而设计的，也是使用 Flutter 开发应用的一个绝佳开发环境。在开始构建一个应用之前，需要为编辑器安装 Flutter 和 Dart 插件，以便让其更易于编写代码(支持这些插件的其他编辑器是 IntelliJ 或 Visual Studio Code)。这两个编辑器插件提供了自动补全、语法高亮、运行和调试支持等。使用一个不带任何插件的普通文本编辑器来编写代码也是可行的，但建议最好使用这些插件特性。

https://flutter.dev/docs/get-started/editor/处提供了安装不同代码编辑器的说明。为了支持 Flutter 开发，请安装以下插件。

- 用于开发人员工作流的 Flutter 插件，如支持运行、调试和热重载。
- 用于代码分析的 Dart 插件，如支持即时代码验证以及代码自动补全。

遵循以下步骤安装 Flutter 和 Dart 插件。

(1) 启动 Android Studio。

(2) 单击 Preferences | Plugins(macOS 系统)或 File | Settings | Plugins(Windows 和 Linux 系统)。

(3) 单击 Browse Repositories，选择 Flutter plug-in，然后单击 Install 按钮。

(4) 当提示安装 Dart 插件时单击 Yes 按钮。

(5) 当出现提示时单击 Restart 按钮。

1.6 本章小结

本章介绍了 Flutter 框架的后台运行机制。我们认识到，Flutter 是用于构建 iOS 和 Android 移动应用的绝佳移动端 UI 框架。Flutter 还计划支持桌面端、Web 端以及嵌入式设备的开发。可从本章了解到，Flutter 应用是从单一代码库构建的，会使用 Widget 创建 UI 并使用 Dart 语言进行开发。Flutter 使用了 Skia 2D 渲染引擎来兼容各种不同类型的硬件和软件。

Dart 语言预先编译为原生代码，从而让应用得到较好的性能。Dart 是 JIT 编译的，这样就能借助 Flutter 的有状态热重载来快速显示代码变更。Widget 就是构成 UI 的构造块，并且每个 Widget 都是 UI 的不可变声明。Widget 就是创建 Element 的配置。Element 是挂载和绘制在界面上的具化的、有意义的 Widget。RenderObject 实现了基础布局和绘制协议。

本章介绍了无状态和有状态 Widget 的生命周期事件。无状态的 Widget 是通过扩展(继承) StatelessWidget 类的单个类来声明的。有状态的 Widget 是使用两个类来声明的，即 StatefulWidget 类和 State 类。

Flutter 是声明式的，并且当状态发生变化时其 UI 会自行重建。Widget 是 Flutter 应用的构造块，并且 Widget 是用于 UI 的配置。

嵌套(复合)Widget 会引发 Widget 树的创建。Flutter 框架使用 Widget 作为构建每个 Element 的配置，从而创建 Element 树。Element 就是挂载(渲染)到界面上的 Widget。前面的过程会创建渲染树，这是一种低级别布局和绘制系统。我们要使用 Widget 并且不需要直接与渲染树交互。无状态的 Widget 具有创建无状态 Element 的配置。有状态的 Widget 具有创建有状态 Element 的配置，并且有状态 Element 会请求 Widget 创建一个状态对象。

本章讲解了如何安装 Flutter SDK、用于 iOS 编译的 Xcode，以及用于 Android 设备编译的 Android Studio。当 Mac 上安装了 Android Studio 时，它就可为 iOS(通过 Xcode)和 Android 设备处理编译工作。本章还介绍了如何安装 Flutter 和 Dart 插件以帮助开发人员的工作流程顺利进行，如代码补全、语法高亮、运行、调试、热重载以及代码分析。

下一章将介绍如何创建首个应用并使用热重载来实时查看变更，如何使用主题将应用样式化，何时使用无状态或有状态的 Widget，以及如何使用外部包来快速添加功能(如 GPS 和图表)。

1.7 本章知识点回顾

主题	关键概念
Flutter	Flutter 是一种移动端 UI 框架，用于从单个代码库中构建用于 iOS 和 Android 的现代、原生和反应式移动应用。Flutter 还扩展了桌面端、Web 端和嵌入式设备的开发体验
Skia	Flutter 使用 Skia 2D 渲染引擎，因此可兼容不同的硬件和软件平台
Dart	Dart 是用于开发 Flutter 应用的语言。Dart 是预先(AOT)编译为原生代码的，以便获得较高性能。Dart 是即时(JIT)编译的，以便通过 Flutter 的有状态热重载来快速显示代码变更

(续表)

主题	关键概念
声明式用户界面(UI)	Flutter 是声明式的，并且会构建 UI 来反映应用的状态。当状态发生变化时，UI 就会被重绘。Flutter 使用 Dart 来创建 UI
Widget	Widget 是 Flutter 应用的构造块，并且每个 Widget 都是用户界面(UI)的一个不可变声明。Widget 是创建 Element 的配置
Element	Element 是挂载(渲染)到界面上的 Widget。Element 是由 Widget 的配置创建的
RenderObject	RenderObject 是渲染树中的一个对象，它会计算和实现基础布局和绘制协议
Widget 生命周期事件	每个无状态或有状态的 Widget 都具有一个带有 BuildContext 的 build 方法，BuildContext 处理了该 Widget 在 Widget 树中的位置。BuildContext 对象都是 Element 对象，这表明 Element 对象是 Widget 在树中某个位置的实例
StatelessWidget	StatelessWidget 是基于其自有配置来构建的，并且不能动态变更。无状态的 Widget 是用一个类来声明的
StatefulWidget	StatefulWidget 是基于其自有配置来构建的，但是可以动态变更。有状态的 Widget 是用两个类来声明的，即 StatefulWidget 类和 State 类
Widget 树	在编排(嵌套)Widget 时，就会创建 Widget 树；这被称为复合。将会创建三棵树：Widget 树、Element 树以及渲染树
Element 树	Element 树代表着挂载(渲染)到界面上的每一个 Element
渲染树	渲染树是一种低级别布局和绘制系统，它继承自 RenderObject。RenderObject 实现了基础布局和绘制协议。我们要使用 Widget，却不需要与渲染树直接交互
Flutter SDK 和 Dart	该移动端软件开发工具集已扩展到桌面端、Web 端以及嵌入式设备
Xcode 和 Android Studio	构建 iOS 和 Android Mobile 应用的开发工具
Flutter 插件	这个插件有助于开发人员工作流程的处理，如运行、调试和热重载
Dart 插件	这个插件有助于代码分析，如即时代码验证和代码补全