移动开发经典丛书

Flutter 实战

[荷兰] 弗兰克·扎米蒂(Frank Zammetti) 著
贡国栋 任 强 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：01-2020-2324

Practical Flutter: Improve your Mobile Development with Google's Latest Open-Source SDK

Frank Zammetti

EISBN：978-1-4842-4971-0

Original English language edition published by Apress Media. Copyright © 2019 by Apress Media. Simplified Chinese-Language edition copyright © 2020 by Tsinghua University Press. All rights reserved.

本书中文简体字版由 Apress 出版公司授权清华大学出版社出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目(CIP)数据

Flutter 实战 / (荷)弗兰克 • 扎米蒂(Frank Zammetti) 著；贡国栋，任强 译. －北京：清华大学出版社，2020.7

(移动开发经典丛书)

书名原文：Practical Flutter: Improve your Mobile Development with Google's Latest Open-Source SDK

ISBN 978-7-302-55608-4

Ⅰ. ①F…　Ⅱ. ①弗… ②贡… ③任…　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 089308 号

责任编辑：王　军
装帧设计：孔祥峰
责任校对：牛艳敏
责任印制：杨　艳

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 刷 者：北京富博印刷有限公司
装 订 者：北京市密云县京文制本装订厂
经　　销：全国新华书店
开　　本：170mm×240mm　　**印　　张**：20.25　　**字　　数**：420 千字
版　　次：2020 年 7 月第 1 版　　**印　　次**：2020 年 7 月第 1 次印刷
定　　价：79.80 元

产品编号：086012-01

译 者 序

移动研发由来已久，从早期的 J2ME 开始，到后来居上的 Windows Mobile、Symbian，小众却各领风骚的 BREW、Palm、BlackBerry，以及当前几乎平分市场的 Android 和 iOS。面对如此多的平台和技术，开发者可以择其优者而习之，公司却不得不兼而用之，但随之而来的必然是成本的上升。

因此，跨平台的呼声一直很高，各种方案也不断面世：PhoneGap(即 Cordova)、React Native、WEEX 以及最近的 Flutter。Flutter 无疑是其中的佼佼者，一经问世就已引起广泛关注，无论是其革新的理念，还是其“正统的出身”（毕竟 Flutter 与当今两大移动平台之一 Android 师出同门）。

在有多种选择时总要进行一番对比，孰优孰劣众说纷纭，抱着小马过河的心态，各大公司也不乏实践，做出一些有益的尝试。接到本书的翻译任务时，正值译者将 Flutter 技术尝试运用于公司的 ToB 项目之刻。

本书既然名为实战，内容紧扣主题，不会涉及太多概念介绍、原理深究，一切以实用为目的，是快速入门的好帮手。作者深耕移动研发多年、亲历跨平台技术发展，并且撰写了多部相关技术书籍，是一位当仁不让的资深技术专家。值得一提的是，作者言语诙谐、娓娓道来，令读者在阅读时宛如与真人对话一般，确实是一种不可多得的阅读体验。

感谢清华大学出版社的编辑老师，让我有幸参与此书的翻译工作。译稿虽几经审阅，仍不敢说无一遗漏，望读者在阅读的同时，亲自实践(下载源码并在模拟器或手机上运行)一番以加深理解。

贡国栋

2020 年 1 月 17 日

作 者 简 介

Frank Zammetti 是一位小有名气的技术作家，作为一名开发者，Frank 写过各种各样的代码，在近 40 年的职业生涯中有 25 年从事专职软件开发。最近，你会发现他的名片上印有架构师的头衔，但他内心深处仍然是一名程序员，并且几乎每天都在围着代码转。

技术审校者简介

Herman van Rosmalen是荷兰中央银行De Nederlandsche Bank N.V.的一名开发者/软件架构师。他拥有超过30年使用各种编程语言进行应用软件开发的经验。Herman专注于搭建框架、PC、客户机-服务器架构、网站和移动应用技术。在过去4年中，Herman把主要精力用在使用.NET C#和Angular开发应用软件上，而之前的15年则主要使用Java技术。

Herman与妻子Liesbeth以及孩子们Barbara、Leonie和Ramon生活在荷兰一座名为Pijnacker的小城。除了编写软件外，在业余时间里他还是足球教练，执教一支女子足球队已近10年。当然，他还是Feyenoord队的球迷！

致　　谢

如果从未写过书，让我告诉你个秘密：撰写这样的一本书时，写作本身只是完成并完善整件事的过程中的一小部分。有时，我甚至认为是微不足道的！

因此，我要感谢所有为此努力工作的人，是他们帮助我完成这本书并交到你手上(无论是纸质版还是数字版)，他们是 Nancy Chen、Louise Corrigan、James Markham、Herman van Rosmalen、Welmoed Spahr 和 Dhaneesh Kumar。

我还要感谢 Lars Bak 和 Kasper Lund，是他们创造了 Dart 这一支撑 Flutter 的编程语言。Dart 优雅且易用。作为一名多年前曾创建自己的语言和工具链的人，我非常感谢你们所做的一切！荣耀属于你们！

我要感谢整个 Flutter 开发团队。我使用各种方式从事移动开发将近 20 年(请打开 etherient.com，单击 Products→Eliminator，这是 2001 年我为 Microsoft 的 Pocket PC 平台开发的一款游戏——我确定那是我的第一款移动应用，至少是有记录可查的第一款)，并且我已能够熟练使用诸多移动开发工具、框架和库，数都数不过来。鉴于多年的经验，我可以自信地说，即使只是首个发布版本，Flutter 也能完胜其他方案。Flutter 团队在这么短的时间内完成的工作令人拍案叫绝，没有你们的辛苦工作，我显然无法完成本书！我期待 Flutter 被越来越多的人使用，并且期待你们的更多作品！

前　言

即使经过这么多年开发者们孜孜不倦的努力，创建如原生应用般外观、体验及功能的跨平台移动应用依旧是一个让人棘手的难题。你可以为各个平台分别编写原生代码，并尽可能让它们表现一致，这的确不失为使你的应用获得原生性能和能力的好办法。但实际上，这意味着你的应用要编写多次，而客户往往不太乐意为此买单！

与之相对的是，你可以基于 HTML 技术实现一次编码而到处运行。但那样的话，你将无法使用诸多本地设备能力，更别提差劲的性能表现了(诚然，有一些优化措施，但也只能减少而不是消除这些顾虑)。

由 Google 的天才工程师创建的 Flutter 平台提供了一种只需要编写一遍代码(或多或少)，就能在 Android 和 iOS 两个平台上运行一致且具备原生性能和能力的方法。在移动开发库领域，使用现代工具和开发技术构建的 Flutter 为开发者提供了一种新的编程方式。

在本书中，你将通过构建两个真正的应用来学习 Flutter，而非使用简化、笨拙而又矫揉造作的例子(尽管在早期会因介绍概念而引入一些此类例子)。是的，我们将一起构建可以按自己意愿并直接应用于实践的应用，而非进行简单的技术演示，并且在整个过程中，你会接触到开发过程中的各种问题，包括我曾遇到过的问题以及解决方案。这样，你就会获得在现实环境中使用 Flutter 的扎实而又真实的经验——并借此为将来构建自己的应用做好准备。

你还将学到构建应用的一些关联知识，如使用 Node.js 和 WebSocket 构建服务端。

除此之外，你还将学到类型截然不同的第三个应用：游戏！是的，我们将使用 Flutter 来构建游戏，以介绍 Flutter 的一些附加的、通过前两个应用不一定得到的功能，并且为你提供从不同视角审视 Flutter 的机会，以拓展你的视野。

你终将掌握 Flutter，且具备使用 Flutter 构建自己的 Next Big Thing 应用的能力。

在开始阅读本书之前，我建议你打开 Apress 网站，搜索本书并下载源代码。你将

得到所需的一切代码，而无须亲自输入！读者也可通过手机扫描封底的二维码下载本书的源代码。

不要忘了学习任何知识的最好方式是动手实践，因此一定要深入代码，修改示例代码和应用，然后观察相应的变化。当你读完介绍每个应用的章节时，你应该打开源代码并尝试添加一两个功能(我还会给你一些这么做的建议，为你指明方向)。

我希望你能喜欢本书，并从中学到很多，这是我衷心的愿望！

目　录

第 1 章

初识 Flutter

欢迎来到起跑线！

如果向十个移动开发者问起：他们是如何在 Android 和 iOS 设备上开发移动应用的？你很可能得到十种不同的回复。不过由于新技术 Flutter 的出现，这样的情形可能不会持续太久。

在本章中，我们将聚焦移动开发，讲解 Flutter 在移动开发方面扮演的角色，及其在某些方面是如何彻底改变移动开发的。我们将初步理解 Flutter 的方方面面，为在本书随后的部分构建一些真正的应用做好准备。

好了，让我们这就开始吧，下面首先介绍一点儿有关移动开发的内容。

1.1 在深渊中沉思

软件开发无易事！

对有些人来说，编程就是家常便饭，我也是其中一员。我不是要炫耀我的个人经历而让你觉得无聊，而事实是从 7 岁那年起，我就已经开始使用各种方式进行编程了。也就是说，我从事这个行业已经快 40 年了(其中有 25 年从事专职软件开发)。经历了这么多，我领悟到一个真谛，正如开篇所说：软件开发无易事。不可否认，有些独立的任务和项目可以轻松搞定；但总的来说，我们程序员从事的是一项从根本上就相当棘手的工作。

而且我还没提及移动开发呢，它可能更难！

大约二十年前，我开始从事移动开发工作。那时还是 Microsoft 公司的 Windows CE/PocketPC 系统和 Palm 公司的 PalmPilot 系列设备及其 PalmOS 系统(尽管还有一些其他的设备和操作系统，但可以说，在一定程度上它们是当时的主流)的天下。就这一点来说，移动开发还不是太糟糕：只需要关注有限的几种设备及功能，并且在开发工具方面也并不缺乏。虽然这些工具不像今天我们所使用的那样优秀，但基本上只有一

种用于开发 PocketPC 或 PalmOS 应用的方式，而不是有一大堆的选择。这听上去糟透了，并且在某些方面的确十分糟糕，但选择减少的同时也省去了开发者的困扰，而选择的多寡正是当今软件工程领域最大的争议之一。

另外，当时完全没有人意识到要进行跨平台开发。如果需要在两个平台上运行，只能各自编写一套代码，而这在今天看来是多么不可思议。不过在当时，只需要考虑两个平台的差异性，编写两次代码也就不那么令人难以接受了。你很容易就能找出一个应用在不同平台上的特性，因为开发者不可能或无法将其从一个平台迁移到另一个平台(事实上，最大的理由是，这样做并不值得付出相应的时间和精力)。

从那时起，移动开发领域经历了诸多革新，见证了诸多改变、扩张与收缩。一段时间以来，我们需要支持诸多平台：Android、iOS、webOS、Tizen、Windows Mobile，也许还有其他一些我未记住的平台。在那个年代，在各平台间迁移应用仍然十分普遍，因为并没有一种好用的跨平台解决方案，至少没有一种不做出重大妥协。是的，随着时间的推移，迁移变得更友好，因为至少原生开发工具有了长足改进，尽管你不得不为同一个应用编写多次代码，但是每次编写都较过去容易不少。Apple 公司于 2008 年发布了 iOS SDK，随后 Google 也于一年后发布了 Android SDK，而为这两种平台开发应用意味着(直到今天仍然是)使用两种 SDK，因为 iOS 开发使用的是 Objective-C 语言(现在 Swift 也用得越来越多了)，而 Android 开发主要使用 Java 语言。

最终，平台的数量开始逐渐减少，因为在这一领域的胜败早已尘埃落定。今天的赛道基本上是 Android 和 iOS 双雄争霸(尽管还有一些其他的平台，但此刻它们的影响力几乎可以忽略不计，并且多数开发者，在我看来甚至可以说是绝大多数开发者，都倾向于无视它们，除非有特别的目的必须包含它们)。于是，真正的跨平台开发理念开始变得更加吸引人和可行了。

互联网的兴起提供了一种选择，因为你可以基于 Web 技术开发并打包成一个移动应用，使它在 Android 和 iOS(甚至更多平台)上的表现几乎一致。但我们需要为此做出一些妥协，而这些妥协尽管随时间的推移减少了，但却一直存在。比如性能和真正的原生能力，对 Web 技术来说仍然有些棘手。

除了 Web 技术，在过去几年中，我们还见证了数种跨平台开发技术和工具的诞生，使用这些技术和工具，只需要编写应用一次，就可以运行于所有设备，在保持一致性的同时获得原生般的体验。其中广为人知的有 Corana SDK(主要用于游戏，尽管并非唯一选择)、Xamarin、PhoneGap(本质上仍然是 Web 技术，但巧妙地使用一个原生的 WebView 组件进行了包装)、Titanium、Sencha Touch(虽然还是基于 Web 技术，但提供了较好的抽象)等，其他就不一一列出了。当下并不缺少选择，并且它们也各有拥护者和反对者。

自从 Flutter 以全新竞争者的身份进入该领域后，便远胜其他一切技术，成为编写跨平台移动应用的唯一正确方式。

1.2 透过名字这一表象

众所周知，Flutter 是由 Google 公司发明的，这是一家对互联网发展有着重要影响力的公司。在 2015 年的 Dart 开发者峰会上(请记住 Dart，因为我们很快就会回顾与之相关的内容)，Flutter 以 Sky 为名诞生了。最初，Flutter 只是运行在 Google 自家的 Android 操作系统上，但不久后就被迁移到 Apple 的 iOS 操作系统上，而这两种系统正是引领当今移动操作系统的佼佼者。

自首次亮相后，Flutter 陆续发布了多个预览版本，最终于 2018 年 12 月 4 日发布了第一个稳定版本：Flutter 1.0。这标志着 Flutter 迎来大爆发，对开发者来说是时候转向 Flutter 阵营了——事实上他们已经这么做了。由于一些非常正确的决策，Flutter 的流行可以说是如流星般迅速。其中一个是：Flutter 最初的目标，或者说主要目标之一，就是无论如何都要使应用的 UI 渲染速度达到流畅的 120fps。Google 的工程师们非常明白流畅的 UI 正是用户喜闻乐见并且想要拥有的体验，因此在设计 Flutter 时它被列为所要考虑的首要特性之一。这是明确要实现的最高目标，到目前为止还没有一个跨平台的移动框架可以达成这一点(即使那些并非跨平台的移动框架也曾为此付出过巨大努力)。

Google 在设计 Flutter 时做了一系列不同于其他移动开发方案的关键决策，其中之一是 Flutter 自主渲染其 UI 组件。与其他框架不同，Flutter 并未使用原生平台的组件。换句话说，当你告诉 Flutter 显示一个按钮时，Flutter 将自主完成该按钮的渲染；而非转嫁给底层 OS 来渲染该按钮，这也是其他大多数框架的做法。这一点展示了 Flutter 与其他几乎所有的跨平台方案的重要区别，也正因此而保证了 Flutter 在不同平台上的流畅性。受益于此，新的 UI 组件、微件(也请记住这个概念，它即将像 Dart 一样隆重登场)可以被快速且简单地添加至 Flutter，而无须担心底层平台是否支持。

自主渲染的特性使得 Flutter 可提供特定风格的微件。Flutter 支持两种风格的微件：Material 风格的微件和 Cupertino 风格的微件。前者实现了 Google 自家的 Material 设计语言，也就是 Android 默认的设计语言；后者实现了 Apple 的 iOS 设计语言。

Flutter 内部可从概念上划分为 4 个主要部分。首先是 Dart 平台，因内容庞杂，需要用一整章来介绍，我将暂时跳过它并在下一章中专门讲述。

接下来我们来看第二个组成部分：Flutter 引擎。这是一个(绝大部分)由 C++语言实现的代码库，其核心性能接近原生级别，同时使用 Skia 图形引擎来完成渲染工作。Skia 是一个轻量级的开源图形库，也是由 C++实现的，经改进后在所有兼容平台上都可获得极其优秀的性能。

作为第三个主要组成部分，Flutter 提供了一套所谓的基础类库——用于封装两个平台的原生 SDK 的接口。其目的在于消除原生平台 API 的差异，以便支持 Flutter API

的一致性。也就是说，你无须关注如何在 iOS 及 Android 上启动拍照应用，也不必费心调用两个平台特定的 API。你只需要了解调用哪个 Flutter API 来启动拍照应用，它将自动适配两个平台。

最后一个组成部分是微件，但正如 Dart 一样(内容庞杂)，也须另起一章专门介绍，我们稍后再讨论它们。

简而言之，以上内容就是 Flutter 的全部。老实说，以上所述并未覆盖多少开发 Flutter 应用所需掌握的知识点。尽管如此，我仍然认为了解一点发展历史以及所使用开发工具的原理，总是有用的。希望你也有同感！

接下来将辨析几个我刚刚提出的概念，对这些概念稍作扩展并究其细节，下面就从 Dart 开始吧！

1.3 Dart：众神的语言

当 Google 启动 Flutter 项目时，有一个早期决策需要做出：应该使用何种编程语言？或许可以选择基于网页的语言(比如 JavaScript)，或许因 Android 使用 Java 的缘故而自然而然地选择 Java。或许，既然他们打算支持 iOS，Objective-C 或 Swift 也是不错的选择(毕竟 Swift 表面上是开源的，因此我敢打赌，Flutter 团队内部早期一定就此有过争论)。或许，Golang、Ruby 之类的语言更好。传统的 C/C++怎么样？还是响应 Microsoft 的号召加入 C#阵营(因为 C#也是开源的)？

我相信有太多的选择，但最终 Google 决定(由于诸多因素)使用了一种多年前他们自己开发的语言：Dart。

由于接下来的整个第 2 章都用来介绍 Dart，因而在这里我将避免涉及太多细节，但浏览一下下面这个简单的示例还是有必要的：

```
import "dart:math" as math;

class Point {

  final num x, y;

  Point(this.x, this.y);

  Point.origin() : x = 0, y = 0;

  num distanceTo(Point other) {
```

```
    var dx = x - other.x;
    var dy = y - other.y;
    return math.sqrt(dx * dx + dy * dy);
  }

  Point operator +(Point other) => Point(x + other.x, y + other.y);

}

void main() {
  var p1 = Point(10, 10);
  var p2 = Point.origin();
  var distance = p1.distanceTo(p2);
  print(distance);
}
```

是否理解每一行代码并不重要。事实上，此刻你无须理解任意一行代码。如果有过使用 Java 或类 C 语言的经验，我相信你能理解它们且没什么太大障碍，这正是 Dart 的一大优势：其语言特性与绝大多数现代开发者所熟悉的那样如出一辙，可以被快速且容易地掌握。

■ **注意：**

有趣的是，C语言本身以及其他类C语言都是从ALGOL这种相当古老的语言发展而来的。

在此，我没有过多地深入探讨 Dart 的本质细节(第 2 章要介绍的主要内容)，只是想适当地介绍一点 Dart 的背景。如前所述，Google 于 2011 年创建了 Dart，首次公布于在 Aarhus Denmark 举行的 GOTO 会议上。Dart 的首个版本 1.0 发布于 2013 年 11 月，在其发布后的大约两年后 Flutter 发布了。顺便说一句，我们都应感谢 Dart 的创建者 Lars Bak(他同时也是支撑 Chrome 和 Node.js 的 V8 JavaScript 引擎的开发者)和 Kasper Lund。

Dart 语言小而美，由于对 Flutter 的重要性，它很快便获得大量的关注。尽管作为一种通用目的的编程语言，Dart 能够且已被用于构建各类应用，从 Web 应用到服务器端，再到 IoT(物联网)应用以及其他领域。大概在我撰写本章内容时，JAXenter 公开了一项关于 2019 年对开发者最重要的编程语言的调查(详见 https://jaxenter.com/poll-results-dart-word-2019-154779.html)，其结论指出有两种语言脱颖而出且远远超出其他语言，

它们是 Dart 和 Python，并且 Dart 全面胜出。该项调查还显示 Dart 在 2018 年经历了蓬勃发展，且 Flutter 几乎可以说是其唯一强大的推动者，光靠 Dart 自己可能还不足以解释这些结论，但公平地讲，Dart 正迅猛进击所有领域！

因此，Dart 正毋庸置疑地受到越来越多的关注。

Dart 涵盖了哪些内容？简要来讲，有几个关键的要素，并且在前面的代码示例中已演示了大部分：

- Dart 是完全面向对象的。
- Dart 是一种垃圾回收型语言，因此无须关注内存的分配与回收。
- 语法风格基于 C 语言，易于为大部分开发者所掌握(诚然，Dart 确实有一些特性可能会难住你，不过不会比其他有着类似语法的语言更糟)。
- 支持通用语言特性，如接口、混入(mixin)、抽象类、泛型具体化(相对于类型擦除而言)以及静态类型检查。
- 拥有完善的输入辅助系统(但在相同类型的输入上的灵活性，使得 Dart 对开发人员而言起到真正的辅助作用而非负担)。
- 并发隔离，因此你将获得独立的线程，它们以消息传递而非内存共享的方式通信。虽然这会损失一些性能，但能够降低其他形式的并发编程模型带来的风险。
- Dart 可以 AOT(ahead-of-time)方式编译为原生代码以达到汇编级别的最优性能。Dart 代码可被编译为 ARM 或 x86 架构下的二进制代码，但同样也可被编译为 JavaScript，因此 Dart 代码甚至还能在网页上勉强运行。暂且抛开这种转译，既然 Flutter 以移动平台为目标，那么 Dart 代码将以 AOT 为主要编译方式。
- Dart 在其语言基础之上提供了一个相当庞大的包的仓库，其中包括几乎大部分开发者都会用到的各种附加功能，便于他们轻松地导入自己的项目。
- 支持众多流行的开发者工具，包括 Visual Studio Code 和 IntelliJ IDEA。
- 将快照作为 Dart 虚拟机的核心部分，从而提供了一种在运行时存储或序列化对象及数据的方式(Dart 程序可被编译为快照，其中包含了所有的程序源代码和经过预分析的依赖，并且可在启动时立即执行，同时快照还被重度应用于并发编程)。

Dart 现在已成为 ECMA 的标准之一，且由技术委员会 TC52 管理，可以在网站 www.dartlang.org/guides/language/spec 上查阅最新的规范说明(更常见的做法是访问 Dart 语言的主页 www.dartlang.org)。

前面已提到，接下来的第 2 章将为你提供关于 Dart 的最新信息，且足以应对即将到来的 Flutter 代码。但以上介绍的相关内容，仅限于作为一种适当且足够的补充简介。

1.4 拥抱微件

让我们回过头来说一说整场演出中真正的明星——Flutter，以及一个比其他任何概念都更为重要的、用以支撑 Flutter 一切的概念——微件。

在 Flutter 的世界里，任何事物都是微件。当我提到任何事物时，意指几乎所有的事物都是微件(想在 Flutter 中找出一个不是微件的事物，难度要比找出是微件的所有事物高得多)。

那么，什么是微件？它们是应用的 UI 组成部分(尽管并非所有的微件都会显示在屏幕上，只有小部分会显示)。显然，微件也可以是代码行，如下所示：

```
Text("Hello!")
```

以下代码清单同样是微件：

```
RaisedButton(
  onPress : function() {
    // Do something.
  },
  child : Text("Click me!")
)
```

以下代码清单也是微件：

```
ListView.builder(
  itemCount : cars.length,
  itemBuilder : (inContext, inNum) {
    return new CarDescriptionCard(card[inNum]);
  }
)
```

最后，以下代码清单还是微件：

```
Center(
  child : Container(
    child : Row(
      Text("Child 1"),
      Text("Child 2"),
      RaisedButton(
```

```
          onPress : function() {
            // Do something.
          },
          child : Text("Click me")
        )
      )
    )
  )
```

最后这个微件很有趣，它实际上使用了一种由多个微件构成的层级结构：一个 Center 微件，里面嵌入了一个 Container 微件，而在这个 Container 微件的内部又嵌入了一个 Row 微件，在这个 Row 微件的内部又嵌入了两个 Text 微件以及一个 RaisedButton 微件。它们是什么微件并不重要(尽管命名已给出含义)，重要的是你所看到的整个微件的层级结构，正是微件在 Flutter 领域内的存在形式。

没错，在 Flutter 中微件无处不在。微件就在我们身边。Flutter 本身就是一个强大的 UI 框架：你得到的是一个微件！

正如开头提到的，Flutter 中几乎所有事物都是微件。当你在用户界面的上下文环境中提到微件这个词时，人们所能想到的各种显而易见的事物有按钮、列表、图片、文本输入框等事物。它们当然是微件。但是，在 Flutter 中那些一般你不认为是微件的事物，仍然是微件，比如图片的外边距、文本输入框的状态、显示在屏幕上的文本甚至应用所使用的主题，所有这些事物也都是微件。

既然 Flutter 中的一切都是微件，一个显然的结论就是：你的代码从本质上看就是由众多微件堆叠而成的(这在 Flutter 中有一个特别的叫法：微件树)。可见，大多数微件都是容器，意味着它们可以嵌入其他微件。其中一些只能嵌入一个，另外一些可以嵌入多个。同样，嵌入的微件又可以分别嵌入一个或多个其他微件，以此类推。这就是微件的存在形式。

所有的微件都是 Dart 类，所有微件只有一个明确的要求：它们必须实现 build() 方法。这个方法必须返回其他微件！关于这一点少有例外，一些低级别的微件(如 Text 微件)会返回一个原生类型(此处为 String)，但大多数都返回一个或多个其他微件。除此之外，从代码级别看，一个微件就是一个普通的 Dart 类(除了第 2 章将要展示的少量语法之外，这与你在其他面向对象编程语言中认识的类没什么两样)。

Flutter 微件扩展自 Flutter 所提供的几个少数标准类之一，符合典型的面向对象编程范式。扩展类决定了我们将使用何种基本微件，而在 99%的时间里你将围绕两种微件进行开发：StatelessWidget 和 StatefulWidget。

继承自 StatelessWidget 的微件不会发生改变，它们因不含状态而被称为无状态微

件。比如显示小图的 Icon 微件，显示字符串文本的 Text 微件，就属于此类微件。无状态微件的示例如下：

```
class MyTextWidget extends StatelessWidget {

  Widget build(inContext) {
    return new Text("Hello!");
  }
}
```

是的，就是这么简单！

相对而言，StatefulWidget 基类在内部带有一个状态属性，当用户与之交互时将以某种形式发生改变。如 CheckBox、Slider、TextField(当你遇到此类将首字母大写的单词写法时，意即真实的 Flutter 微件类名，而普通的表达将正常译出，如文本表单域)，都是大家熟知的有状态微件。当你编写这样一个微件时，实际上须创建两个类：有状态微件类和相应的状态类。下面展示了一个有状态微件及其关联的状态类：

```
class LikesWidget extends StatefulWidget {

  @override
  LikesWidgetState createState() => LikesWidgetState();

}

class LikesWidgetState extends State<LikesWidget> {

  int likeCount = 0;

  void like() {
    setState(() {
      likeCount += 1;
    });
  }

  @override
  Widget build(BuildContext inContext) {
```

```
      return Row(
        children : [
          RaisedButton(
            onPressed : like,
            child : Text('$likeCount')
          )
        ]
      );
    }
  }
```

像之前一样，无须纠结太多细节，目前并不期望你能读懂这些代码，我们将在第2章中深入讨论，在那之前你只需要掌握Dart的一些基础概念即可。但同时，我相信你多少对我们所讨论的内容有一个基本的认知，因为它们还是比较浅显直白的。除了微件代码及其状态对象是如何交互和发生联系的部分有些难懂之外，其他大部分内容都还算简单，不过不必担心，这样的状况不会持续太久！

暂时回到无状态微件的概念上来，我们应注意到术语“无状态”其实有些不够准确，因为Dart中的类封装了属性和数据，无状态微件也不例外，因此从某种意义上说，它也拥有状态。有状态微件与无状态微件的关键不同在于，无状态微件在“状态”发生改变时并不会被Flutter核心框架触发而自动重新渲染，而有状态微件则会自动触发重新渲染。当有状态微件的状态改变时，无论因何缘故导致，都将触发某些特定的生命周期事件。这些触发生命周期事件回调的函数调用，最终引起Flutter重新渲染微件在屏幕上占据的部分(如果这次改变是必需的，Flutter将替你做出决策，因为它清楚这个微件在事件发生前后的状态)。

我们可以这样理解：两种类型的微件都有状态，不过Flutter只关注有状态微件的状态，甚至在某种程度上会管理其状态。因此，只有有状态微件才会在适当的时候自动重新渲染，且受控于Flutter框架本身而不是你的代码。

你可能倾向于认为你会一直使用有状态微件，这样就可以省很多事儿，但从后续我们构建的应用中你将看到，事情并非如此。事实是：你会发现你将不可思议地大量使用无状态微件。我们改天再来讨论这个话题。

你应该已经注意到有两件事的重要性非同一般。其一，微件构成了Flutter的UI。并形成了早些时候我曾提到的微件树。同时，微件的代码是面向对象的，UI的构建方式就是一种组合范例。这一点非常重要，大部分Flutter微件本身都相当简单，并且也只能通过组合方式来构建强大的UI。有些框架使用可以称为“全能”的组件，它们自身就代表整个应用。Flutter则完全不同，即使相对较小的UI都很可能由多个微件组合

而成。

其二，Flutter 通过代码构建 UI。我知道这是显而易见的，但请想一想：Flutter 不像网页开发那样，并没有一种专门用于构建 UI 的标记语言。由此带来的好处是只需要学会一种语言，掌握一种编程范式。乍一看这没什么，但却是 Flutter 与相对其他竞争者的巨大优势。

暂时这就是所有你需要掌握的微件知识点了。我们将从第 3 章开始将 Flutter 自带的微件列表整理为目录并深入介绍它们，然后在第 4 章基于这些微件来构建三个应用，以讲解每个微件的用法。最终你将对进行 Flutter 开发时用到的大多数常见微件形成良好的理解，并很好地掌握如何使用和构建微件。

1.5 言归正传：Flutter 的优劣对比

与其他任何框架一样，合格的开发者需要评价我们可能考虑的任意选择的收益和风险，Flutter 也不例外。我个人认为 Flutter 可以做很多事，但它并非灵丹妙药，它也有自身的问题。Flutter 有其不足之处，它并不适用于所有项目，尽管如此，我仍然坚信，即使它不是最好的，但也绝不是最差的。

既然如此，我们就来说一说 Flutter 的优劣，为了容易理解，我们将它与其他可选方案进行对比。

- 优势：热重载——当我们搭建好 Flutter 开发环境并快速预览一个模板应用之后，我们将回顾这一点，你将看到，这是 Flutter 的一个巨大优势。在集成了第三方的 Expo 组件后，React Native 也可以拥有这一能力。但 Flutter 的表现更为优异，以我的经验来看，更为流畅。其他框架则很少能做到这一点。
- 劣势：专注移动应用——在撰写本书时，你只能使用 Flutter 开发 iOS 和 Android 移动应用。有了 Flutter 后，你会失望于它不能满足你的所有开发需求。不过，你可以选择使用 Flutter 进行 Web 开发乃至 Windows、Mac 和 Linux 原生开发。不久之后，这种劣势可能变成优势，因为整个计算机世界都可以使用 Flutter 构建(对某些人来说，将成为一种意义重大的优势：Google 将比现在拥有更多话语权——你必须自己做出决定)。
- 优势：完全跨平台——你的 Flutter 应用将以最小代价运行于 iOS 及 Android(乃至 Android 的继任者 Fuscia)平台上。与众不同的是，Flutter 提供了两组微件，其中一组适用于 iOS，而另一组则适用于 Android。因此，你的应用不仅可以表现一致，还可以与平台完美融合(这取决于你，而并非强制要求)。其他平台则很少能像 Flutter 这么轻松做到这一点(你通常需要编写大量分支代码以适配两个平台，使用 Flutter 则相对轻松很多)。

- 劣势：代码混合——一些开发者，特别是那些拥有 Web 开发背景的，认为 UI 与逻辑的分离是再普通不过的做法(相对于使用 HTML/CSS 来定义 UI，使用 JavaScript 及运行于后台的语言定义逻辑)，他们很可能对 Flutter 代码将 UI 与逻辑混合在一起的做法感到惊讶。当然并非只有 Flutter 这么做，React Native 也曾因此饱受诟病。
- 优势：Dart——Dart 简洁而强大、面向对象、强类型，并赋予开发者快速且可靠的代码高产能力。一旦掌握，大多数开发者对 Dart 的热爱都会高过 JavaScript、Objective-C 和 Java。
- 劣势：Google——我把这点列为劣势，更多是个人主观判断，你当然可以同意或反对(老实说，我自己也一直踌躇不定)。有些人对 Google 对互联网的掌控力之大而有所不满，即使那些掌控并非 Google 有意而为之。当你作为游戏的主宰者时，你会倾向于拥有绝对的控制力。尽管如此，有人认为使用 Google 独自研发的技术开发(所有平台的)移动应用有些不自量力。当然也会有人为有如此强大的后盾支持而拍手称快。所以，这是一场你必须自己去寻找答案的辩论。
- 优势：微件——Flutter 自带一组丰富的微件，并且实际上将满足你构建任何应用之需。尽管如此，你仍然可以创建自己的微件(实际上，你必须创建自己的微件，但它们的复杂度将大有不同)，你甚至可以引入其他第三方微件以扩展 Flutter 的能力。新创建或引入的微件与 Flutter 自带的微件一样容易使用。以 React Native 为例，其主要依赖添加诸多第三方微件来弥补自有微件的不足，而 Flutter 则提供了太多开箱即用的微件。
- 劣势：微件树——这点被归为劣势是由于有时你最终会因嵌套太深(如果没有其他原因的话)，导致代码可读性太差，而难以理解其结构。在过去的二十年中，随着万维网的兴起，我们已经变得对此有些习以为常了，因为与 HTML 遇到的问题如出一辙。但因为事实上 Flutter 中的任何事物都是微件，所以嵌套有时比 HTML 还要深，并且 Dart 代码风格导致阅读此类代码有些棘手。当然，你可以运用一些技术手段来减少嵌套，我将在随后带领大家阅读真实的应用代码时就此再作讨论，但嵌套过多的劣势仍需要我们时刻注意，并自行处理。关于这一点，Dart 和 Flutter 都无能为力。
- 优势：开发工具——在 1.6 节中，你将看到完成 Flutter 基础开发环境的搭建是如此简单。尽管如此，你可以抛开基础开发环境，而是使用那些你在进行其他开发工作时所习惯使用的工具。这一点再次减少了开发者的阻力。
- 劣势：响应式编程及状态管理——Flutter 通常被认为是一种响应式编程范式，这意味着可通过给定微件的当前状态来管理应用的 UI。前面介绍过的 build() 方法以当前状态作为入参，并返回包含当前状态的微件的可视化描述。当状

态发生改变时，微件通过再次调用 build()方法，同时传入其最新的状态进行重建以“响应”该变化。这一切的发生都表现为 Flutter 的函数及其生命周期；你(通常)不必考虑太多，而只需要提供 build()方法。相对于其他框架，你需要先构造一个微件，再调用一系列赋值方法以设置其正确的状态。在 Flutter 或其他框架中，这一范式被广为接受，但它也可能被当作 Flutter 的一个劣势，因为有时候，在完成一些微小的功能时无可辩驳地更为复杂(我们将在后续章节中见识到这种复杂性以及一些相应的处理方法)。与此相关的还有状态管理话题，并没有一种标准可用来判断对错，至少在撰写本书时，这仍是 Flutter 的一大不足。你会有多种方式用来管理状态，或优或劣，但你必须决定哪种最适合自己(当然，我也会给出我认为不错的方式)。Google 正在拟订一种标准，但目前还未完成，所以直到它完成之前，我将把(对状态管理)缺乏有力的指导视为 Flutter 的劣势之一(尽管有些观点认为这种灵活性可被当作优势来看，但无论如何我都不会卷入任何无意义的争论之中)。

- 优势：平台特定的微件——由于 Flutter UI 通过代码构建，维护一套代码的同时支持 iOS 和 Android 就变得简单直接了，即使在需要处理一些平台差异时也是如此。举例来说，你总是可以在运行时通过查询 Platform.isAndroid 或 Platform.isIOS 的值来判断当前运行环境，并借此判断应该构造何种平台特定的微件。也许你希望在 Android 平台上使用 RaisedButton 而在 iOS 平台上使用扁平的 Button。没有必要维护两套不同的代码，多数情况下一个简单的分支判断即可完成此类工作。
- 劣势：应用大小——因为需要嵌入核心 Flutter 引擎、Flutter 框架、支持库以及其他资源，Flutter 应用一般要比纯原生方式开发的应用大一些。一个简单的“Hello, World!”Flutter 应用，大小可达 7MB。这当然需要权衡，如果有一个用例对应用大小有严格要求，那么 Flutter 也许不是最佳选择。

好了，至此我敢说你已经掌握了，也可以说积累了有关 Flutter 的一些基本概念，你对 Flutter 有了一个大致的了解。让我们开始动手写一些真正的代码吧！

1.6 无须多言，向 Flutter 进发吧

在真正动手写代码之前，我们应该先安装 Flutter 以及一些需要用到的开发工具，对吧？“工欲善其事，必先利其器！”

幸运的是，Flutter 开发环境的搭建相当简单。

1.7 Flutter SDK

毫无疑问，首先要完成的就是下载、安装以及配置 Flutter SDK。这是一切的基础。接下来，虽然不是必需的，但为配合本书使用还是建议大家下载、安装和配置 Android Studio(包括 Android SDK 和模拟器的设置)。

打开网址 https://flutter.io，这里是 Flutter 安装过程和文档的“一站式购物平台”。单击顶部的 Get Started 按钮，在 Install 页面可以选择你所使用的操作系统(Windows、macOS 或 Linux)。

■ 注意：

我是一名忠实的Windows用户。Windows是我所知道的最好的操作系统。因此，本书将围绕Windows来讲解。说到这儿，我将努力指出不同系统间的重大区别。总体来说，无论你是否使用Windows，都不会有太大区别。虽然安装软件的过程的确有所不同，但是Flutter网站会在你需要协助的地方给予正确的指引。

选择适合的链接，Flutter 网站将为你提供下载及安装 SDK 的信息。与其他 SDK 或软件不同，你可能会遇到一点困难。必须指出的一点是如何将 SDK 添加至系统环境变量的那部分说明。尽管这样做有利于编译构建，但你仍然可以跳过这一步骤，如果愿意的话。不过要注意，如果坚持跳过，所有命令的执行都必须位于 SDK 路径之下，否则必须指定 SDK 的完整路径。一旦我们完成 Android Studio 的安装，你就会发现无须添加 SDK 至系统环境变量，只有当确实需要在命令行环境下执行 SDK 命令而非通过 Android Studio 执行编译构建时，设置系统环境变量才有用。

说到命令，事实上你可能需要执行的一条命令，一条按照 Flutter 网站上的安装说明需要在 SDK 安装完毕时执行的命令，就是 flutter doctor。大多数命令将被转发给 SDK，除此之外就是 flutter，这也是实际上执行的程序，doctor 是可以发送给它的命令之一。这可能是最重要的一条命令，可用来执行一系列的检查和配置以确保 Flutter 可以运行。

刚开始执行这条命令时，你可能会看到一些可以预见的失败条目，下一步将解决这个问题：安装 Android Studio。

1.8 Android Studio

Flutter 网站上的说明将指导你安装 Android Studio，虽然在不同系统上略有不同，但是当你下载完成后，你将启动并执行 Android Studio 的安装向导。你将下载 Android SDK 和模拟器镜像，以及其他一切必需的工具。同时，你需要安装一些与 Dart 和 Flutter

相关的特定插件。

现在，完成上述步骤后，如果继续遵照安装说明，接下来你将需要将 Android 手机或平板模拟器连接到计算机，并确保 flutter doctor 指令运行无误。尽管如此，你仍然可以跳过该步骤！当然，如果确实拥有一台 Android 设备，你完全可以跳过它。

不过，如果你是一名 iOS 爱好者，并且你不想在开发 Flutter 代码时使用真实设备——老实说，我也不喜欢一直将我的手机连接至计算机——那么我的建议是打开 Android Studio，找到 AVD(Android 虚拟机)管理器，你将在启动界面上找到 Configure 菜单，并创建自己的 Android 虚拟设备。建议创建一台 Pixel 2 虚拟设备，使用 API 级别 28(请确认已安装对应的 SDK，可以通过 Configure 菜单找到 SDK 管理器来确认)并设定分辨率为 1080×1920(420dpi)、目标为 Android 9。选择一种 x86(ABI x86_64)镜像。长久以来，Android 虚拟机在性能方面臭名远扬，但此类虚拟设备表现异常优异，几乎可以媲美原生性能。尽管这么做可能没什么必要，但还是请为其配置 512MB 的 SD 卡。大部分设置使用默认大小即可满足需求，但 API 级别和 CPU 类型比较关键(需要进行特别设定)。

完成上述设定后，你已准备好随时通过 Android Studio 来运行 Flutter 代码了。也可以通过命令行基于 SDK 来运行，不过我不会过多描述如何通过命令行来操作 SDK，除了 flutter doctor 之外。在本书的剩余部分，我们都将在 Android Studio 中完成一切。

注意，当你再次运行 flutter doctor 时，你可能仍旧得到错误提示，说找不到可用的 Android 设备，这可能是因为你刚刚创建的虚拟设备还没有运行起来。尽管如此，flutter doctor 应该能检测出原因，并向你展示一个干净又健康的状态清单。最终，如果虚拟设备确实没有运行，并且你也没有将一台真实的 Android 设备连接至计算机，但只要这是 flutter doctor 报告的唯一错误，你就可以继续前进。

想了解 iOS 是什么状况，先别急！因为我们使用的是 Android Studio，通过它所能做的事，无论操作方式还是文件格式，都不可能运行在 iOS 上。只有以下场景才需要考虑这一点：当你希望在一台真实的 iOS 设备上进行测试，抑或部署的目标设备是 iOS，抑或构建用于分发的应用时，你才需要一台 Mac 机器并安装好 Apple 的 Xcode IDE。本书不涉及 iOS 或 Android 的应用分发主题，因此模拟器就可以满足我们的需求。

1.9 (不那么)经典的“Hello, World!”应用

如果继续遵从 Flutter 网站上的安装指导，你将在最后一步创建一个小型的 Flutter 应用。尽管文档中的讲解非常棒，但我还是建议你先跳过这个步骤，让我来带你实际体验一遍。

首先，让 Android Studio 自动构建一个应用(在 SDK 的协同下)。这个过程相当简单，当我们在虚拟设备上把这个应用运行起来时，我们将稍作修改并一睹热重载如何

发挥作用。

但是，我们得先创建一个项目！当首次启动 Android Studio 时，你会看到如图 1-1 所示的界面。

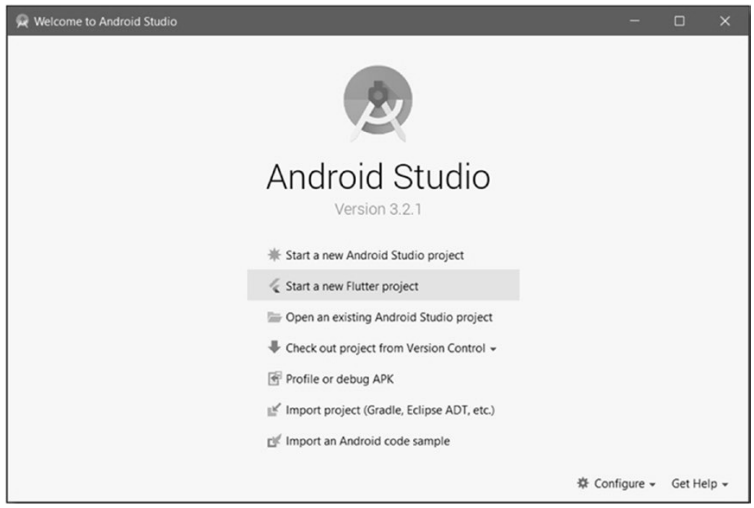

图 1-1　启动 Android Studio

看到 Start a new Flutter project 这行文字了吗？这正是我们需要的，单击它！你将看到如图 1-2 所示的应用创建向导。

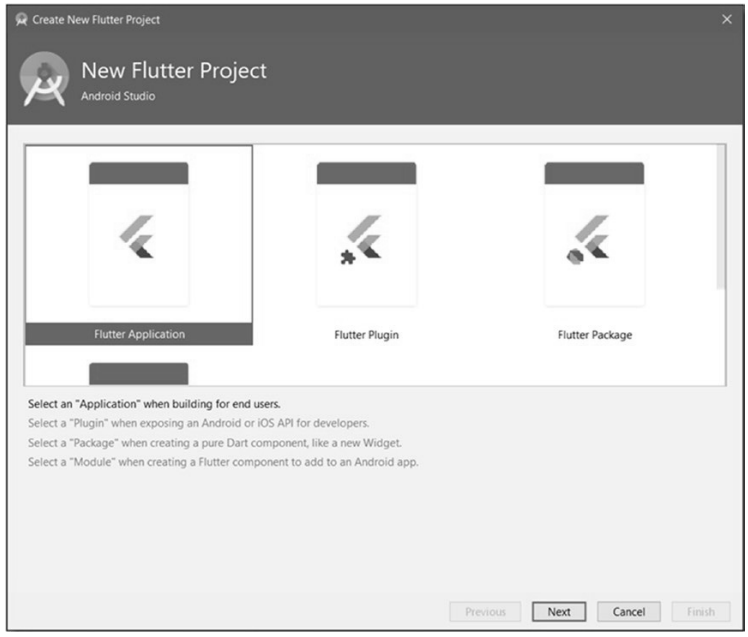

图 1-2　选择需要创建的 Flutter 项目类型

你可以创建四种类型的 Flutter 项目：

- Flutter Application(贯穿本书所使用的类型)
- Flutter Plugin(一种可以导出原生 Android 或 iOS 功能到使用 Dart 语言开发的 Flutter 应用的插件)
- Flutter Package(只有在你想发布一个独立于应用的自定义微件时才需要选择这种类型)
- Flutter Module(可以将 Flutter 应用嵌入原生应用)

选择 Flutter Application 并单击 Next 按钮，你将看到如图 1-3 所示的界面。

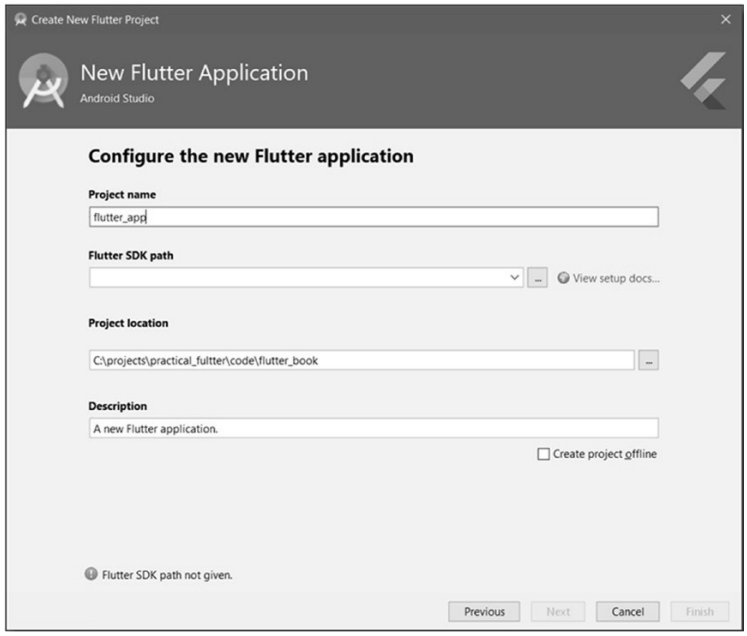

图 1-3　输入应用所需的必要信息

在这里，你需要输入刚刚创建的应用的一些必要信息。默认值也可以，或者按你喜欢的方式对项目进行命名，以及修改相关描述。同时，你可以修改项目的保存路径(保留默认值也行)。看到底部的错误提示了吗？如果没有错误提示，那更好，说明你已经正确配置了系统环境变量。但是，如果遇到错误提示，那是因为 Android Studio 还不知道你的 Flutter SDK 安装在哪里，你必须告诉它。单击 SDK 路径输入框右侧的 ... 按钮，浏览已经安装好的 SDK 路径并选中，确认 Android Studio 加载成功(你会发现错误提示消失了)，再次单击 Next 按钮，我们将看到如图 1-4 所示的界面。

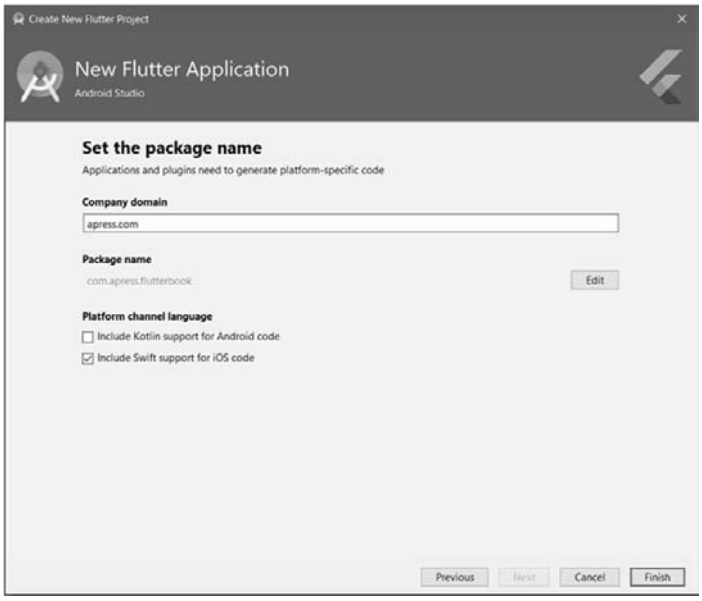

图 1-4　新建项目的最后细节

在最后一个界面上，你需要填写一些额外的信息，主要是公司的域名。当然不一定非得是公司，关键是需要填写规范的点分格式的内容，也就是互联网域名。如果还没有一个域名，你可以输入任何你喜欢的内容。关键是：输入任何对你来说有意义的内容，同时注意包名会随之发生改变——把项目名和你在最后一个界面上填写的域名拼接在了一起。如果希望在应用商店中分发这个应用，那么包名必须保证唯一性，不过既然我们在此以测试为目的，所以叫什么都不要紧。

■ 注意：

你也许还能看到一种名为Sample Application的项目类型，这取决于所安装的Android Studio和Flutter插件的版本。能否看到都无所谓，如果需要的话，它可以帮你创建一些模板代码，但本书用不到，因此无论有没有这个选项都无关紧要。

最后，你需要在平台所支持的编程语言中做出选择。一般来说，比较常见的是不选 Kotlin 而选中 Swift。这里指的是在 Flutter 框架下使用的平台特定的语言，除非在应用中需要与原生代码交互，否则对你来说没什么不同。

设定完成后，单击 Finish 按钮，Android Studio 会快速生成一个简单的 Flutter 应用。这花不了几分钟，你可以观察底部的状态栏以确保所有任务已完成。完成后，查看 Android Studio 工具栏的顶部，找到用于列出已连接设备列表的下拉菜单，如图 1-5 所示。

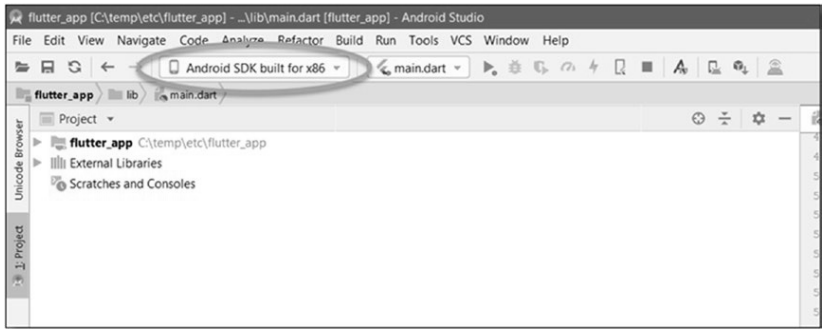

图 1-5　Android Studio 中的虚拟机下拉列表

你会看到前面创建的模拟器。选中它，如果还未运行，那么很快就会启动它。运行起来后，单击写着 main.dart 的下拉列表旁的绿色箭头(这是应用的入口)。然后，稍等一会儿，等待应用完成构建、部署并运行在模拟器上(这取决于你的机器配置，可能需要一分钟左右，不要急——不过首次构建之后就会快很多)。之后你会看到如图 1-6 所示的界面。

图 1-6　这是我的第一个 Flutter 应用

这个应用虽然简单，但却展示了大量内容。单击底部带有加号的圆形按钮(即悬浮按钮或 FAB)，你会发现每单击一次计数就会加 1。

Android Studio 将自动打开已生成的代码，形如以下代码清单(不过请注意，我已经移除注释并重新格式化了部分代码，希望便于阅读)：

```
import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}

class MyHomePage extends StatefulWidget {

  MyHomePage({Key key, this.title}) : super(key: key);

  final String title;

  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {

    int _counter = 0;

    void _incrementCounter() {
    setState(() {
```

```
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
              'You have pushed the button this many times:',
            ),
            Text(
              '$_counter',
              style: Theme.of(context).textTheme.display1,
            ),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: Icon(Icons.add),
      ),
    );
  }
}
```

代码量虽然不大，但还是涉及很多内容。此刻，你还没有掌握足够的知识点，也就是说还不能深入理解，毕竟我们还没开始讲解 Dart 的细节。不过我也没打算完全不

做任何解读而让你自行领悟，因此我会指出其中比较关键的几点。

首先，请注意每个 Flutter 应用的入口都是 main()方法，它将简单地调用由 Flutter 自身提供的 runApp()方法，并传入顶级的根微件。总会有一个微件位于层级的顶层，并包含其他所有微件，这里是一个 MyApp 类的实例。这个类碰巧是前面讨论过的无状态微件类，因此其唯一需要做的就是提供 build()方法，正如你所看到的那样。build()方法返回的微件(请牢记，build()方法返回的永远是一个子微件或有或无的微件)是一个 MaterialApp 类的实例，该微件也是由 Flutter 提供的(你可以在代码的顶部看到它是从 flutter/material 导入的)。我们将在第 3 章深入讨论这个微件，但关键是它提供了 Material 风格(也就是 Google UI 风格)应用的基础框架。你可以看到它设置了标题(MaterialApp 构造方法的入参之一)，用于在应用运行时显示在状态栏上。你还可以看到，你能为 Flutter 应用设置主题和细节，比如主题的主色调，此处设置为蓝色。

该 MaterialApp 微件有一个子微件——一个 MyHomePage 类的实例。Dart 有点儿奇怪的地方在于，当你实例化一个类时，你不需要像大多数面向对象语言那样提供一个 new 关键字，就像这里所展示的那样)。

MyHomePage 类定义了一个有状态微件，在这个例子中我们需要两个类：继承自 StatefulWidget 的核心类，以及与之关联的继承自 State 的状态类。_ MyHomePageState 就是那个状态类，虽然看上去有点儿怪，但它也是一个微件，你可以从它拥有 build()方法看出这一点。你的第一个直觉可能是：build()方法应该在 MyHomePage 内，同时 _MyHomePageState 应该仅包含描述 MyHomePage 微件的数据，但事实恰恰相反。

无论如何，build()方法再次返回了一个微件，这次是 Scaffold 微件。你仍无须考虑它是一个什么微件，我们将在第 3 章深入讨论。但简要来说，它为应用提供了基本的可视化布局，包括显示标题的状态栏(实际上是一个 AppBar 微件)，等等。Scaffold 同时还提供了绑定 FAB 的能力，也可以说，将 FloatingActionButton 微件的实例，作为入参 floatingActionButton 的值传入 Scaffold 的构造方法。

另外一个传入 Scaffold 的构造方法的参数是 body，我们将其他微件作为子节点添加至此。在这里，你将开始见证“一切皆微件”：我们有一个 Center 微件，它是一个容器(你也许已经猜到了)并居中设置了唯一的子微件。此例中，child 是一个 Column 微件，它是 Flutter 提供的众多布局相关的微件之一，该微件将其所有子微件排成一列。这里的 Column 微件的两个子微件都是 Text 微件，一个用于展示文本“You have pushed the button this many times: ”，另一个用于展示单击按钮的次数。

在接下来的两章中，我将不断深入探索 Dart 和 Flutter，这一切将逐渐明朗。尽管我省略了大量细节，但我还是觉得以上解读足以让你产生适度的理解(另外，我移除的那些注释实际上很有用，它们提供了很多有用的信息，当你自行生成这个应用时请记得仔细阅读)。

1.10 热重载：你会喜欢上它的

从现在起，一切将变得难以置信。先确保应用已在模拟器上运行起来，再打开 Android Studio 并找到如下代码：

```
Text(
  'You have pushed the button this many times:',
),
```

随便修改下，比如将 button 改为 FAB，并按 Ctrl+S 快捷键或选择 File 菜单下的 Save All 选项以进行保存。现在，观察模拟器，几乎在同一时间你看到屏幕也发生了变化(也许要花几秒时间，但绝对比首次运行快多了)。

棒极了，是吧？

热重载仅工作于调试模式，从应用的右上角的调试条幅中我们可以看出。在调试模式下，应用运行在 Dart VM (虚拟机)上，而不是编译好的本地 ARM 代码，只有用于发布的应用才会编译为本地代码(因此，在调试模式下，应用的运行会有一点点慢)。热重载通过将修改过的源代码注入正在运行的应用宿主 Dart VM 而生效。注入后，VM 将更新所有因改变成员变量或方法而引起变更的类。然后，Flutter 框架触发一次微件树的重建，于是你的变更被自动响应。你不需要重新构建应用、重新部署或重启任何东西，所有的事情将按需自动完成，以使变更以最快的速度反映在屏幕上。

尽管很少发生，但你可能偶尔会碰到热重载并未如期响应某些代码变更的情形。如果碰到这种情况，先尝试单击工具栏上的 Hot Reload 图标，它看起来像闪电，如图 1-7 所示(也可以在 Run 菜单中找到 Hot reload 选项，快捷键是 Ctrl+/)：

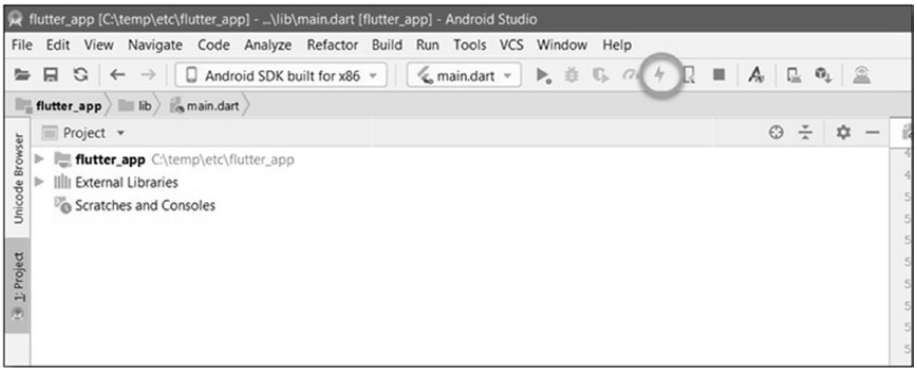

图 1-7　Android Studio 中的 Hot reload 图标

这应该能解决前面的问题。同时请注意控制台窗格，通常位于 Android Studio 下方并自动显示，你会看到一些输出内容，如下所示：

```
Performing hot reload...
Reloaded 1 of 448 libraries in 2,777ms.
```

同样，当热重载执行时你会发现一个小的气泡提示，显示在控制台窗格的旁边。

现在，一件关于热重载机制的令人振奋的事情是：应用的状态被保留了下来！你可以通过单击几次 FAB 按钮，并且观察到文本确实改变了来确认这一点。换句话说，单击按钮的计数值在重载后保留了下来。这使得改变 UI 的同时保留其当前状态变得非常容易，因此你可以在两种设计中轻易地进行 A/B 测试。但如果不想保留状态呢？在这种场景下，需要热重启。你可以通过 Run 菜单中的 Hot restart 选项或使用快捷键(Ctrl+Shift+/)来激活它，但必须手动执行(相对而言，热重载会在你改变代码并保存时自动执行)。

有趣的是，工具栏上并没有提供热重启的图标，不过无论如何，这都将重启你的应用并清除状态，只是没有重新构建。

你当然可以在任何需要的时候执行一次构建(执行 Run 命令)，但这将引起一次完整的编译过程，因此会比较慢。热重启几乎像热重载一样快，因为它所做的工作少得多，却带来几乎相同的效果(当然，除去代码变更的因素——你需要为此执行构建，或者默认情况下通过热重载来完成)。

希望你能明白热重载的好处，开发者可以使用它进行高效开发。我想随着对 Flutter 的理解不断加深，你一定会感激它的。

1.11 Flutter 应用的基本结构

本章的最后一个话题是自动生成的应用结构，基本的文件夹结构如图 1-8 所示。

图 1-8　基本的文件夹结构

如你所见，有 5 个顶级文件夹，它们分别是：

- android——这里包含了 Android 平台特定的代码及资源，比如应用图标、Java 代码、Gradle 配置及临时资源等(Android 使用 Gradle 作为构建系统)。其实，这就是一个标准的 Android 项目，你可以使用 Android 工具链来完成构建。大部分情况下，你只需要修改图标(位于 android/app/src/main/res 目录下，其中每个子目录都代表一种不同的分辨率)。通过 android/app/src/main 目录下的 AndroidManifest.xml 文件，可以按应用所需设置相应的 Android 平台特定的应用属性。
- ios——同 android 目录一样，该目录包含了 iOS 特定的代码。关键的不同在于 ios/Runner/Assets.xcassets 目录，iOS 应用所需的平台特定图标都放在这里；至于 ios/Runner 目录下的 Info.plist 文件，其作用等同于 Android 应用的 AndroidManifest.xml 文件。
- lib——尽管第一次接触时会觉得有点奇怪，但它就是应用代码所在的目录。在这里，你可以用你希望的方式自由组织代码，创建你喜欢的任意目录结构。但你需要一个文件作为入口，大多数情况下，自动生成的 main.dart 文件承担了这个职责。
- res——这个目录包含了一些资源，比如应用需要的国际化字符串资源。本书不会涉及此项内容。
- test——在这个目录中你会找到一些 Dart 文件，用于执行针对应用的测试。Flutter 提供了 Widget Tester 功能，它可以使用这些测试来确保微件功能的正确性。与 res 目录一样，本书也不会涉及测试方面的内容，因为在 Flutter 开发中这是可选的，而且单就测试本身都足以写一本书了。当然，测试相当重要，但在你学会开发 Flutter 应用之前，你无须测试任何东西，本书将聚焦于掌握开发技能这一目标。

在 Android Studio 中默认被隐藏显示的.idea 目录中存储了 Android Studio 的配置信息，你可以忽略它(从 Android Studio 源自 IntelliJ IDEA IDE 可知，.idea 目录因此得名)。此外还有 build 目录，用于存储在使用 Android Studio 和 Flutter SDK 构建应用时用到的一些信息。通常可以忽略它们。

除了以上目录外，你还会在项目的根目录中看到一些文件。有一些位于 lib 目录之外的文件需要关注，它们是：

- .gitignore——Git 版本控制用来管理不需要纳入版本控制的那些文件。开发 Flutter 应用时是否使用 Git 是完全可选的，但该文件无论如何都会自动生成。源代码控制本身是一个完整的话题，本书同样不会涉及此项内容，因此也可以忽略。
- .metadata——Android Studio 用于记录项目的数据。你可以忽略它，因为你永远不会手动编辑它。

- .packages——随 Flutter 提供的包管理器用于管理项目的依赖。这个包管理器叫作 Pub，该文件就是 Pub 用来管理项目的依赖的。你不会与它发生直接的交互，甚至也不会直接用到 Pub，所以也可以忽略它(直接通过命令行使用 Pub 并非绝无可能，但在 Android Studio 中确实少见，随 Flutter SDK 提供的命令行接口已抽象并隔离其中的大部分操作)。
- *.iml——这是 Android Studio 的项目配置文件，它将以你的项目命名。你永远不会直接编辑它，因此可以忽略它。
- pubspec.lock 和 pubspec.yaml——你用过 NPM 吗？你熟悉 NPM 使用的 package.json 和 package-lock.json 文件吗？是的，这就是 Pub 用于同样目的的文件。如果不熟悉 NPM，那么可以使用 pubspec.yaml 向 Pub 描述你的项目及其依赖。pubspec.lock 文件是在 Pub 内部使用的。你一定会直接修改 pubspec.yaml，而绝不能修改 pubspec.lock，稍后将介绍 pubspec.yaml 的细节。
- README.md——你可以按自己喜欢的方式自由使用的自述文件。通常，GitHub 之类的网站使用这个 Markdown 文件来展示项目信息，当你浏览一个仓库时，就会发现它位于根目录中。

到现在为止，pubspec.yaml 是最重要的文件，它也是少有的几个你需要编辑的文件之一，所以你可以忘记其他事，但一定要记得它！稍后当我们需要添加一些依赖到项目中时，我们会用到它，但此刻，自动生成的文件足以满足我们的需求了。

1.12 其他一些“隐藏的”细节

打开 ios 目录，看一下其中的文件，你会发现有一个文件名含有 Runner 这个词。这提示我们构建为发布版本的 Flutter 应用是怎么运行的。正如前面提到的，热重载可以运行是因为你的代码是以调试模式运行在 VM 上的；而在编译为发布版本时，热重载就不能工作了。这时，你的代码将被编译为本地 ARM 代码。事实上，你的代码将被编译为一个 ARM 库，这也解释了为什么你的代码位于 lib 目录中。

Flutter 引擎的代码和你的应用代码，在 iOS 上是使用 LLVM(低级别虚拟机，一种编译器架构，使用 C++编写，是为了对编译期、链接期、运行期以及“空闲期”使用任意编程语言编写的程序进行优化而设计的)提前编译(AOT)的，在 Android 上则使用原生开发包(NDK)编译为一个本地 ARM 库。这个库能被一个叫作 runner 的本地应用加载，也正是 runner 在运行你的应用。可以把 runner 理解为一个轻量的运行器，它知道怎么运行你的应用，并为之提供一些服务。在某些方面，这个运行器表现得像一个 VM，尽管相当轻量级(几乎等同于一个 Docker 容器，如果熟悉的话)。

最后，这个运行器和预编译库都被打包进 iOS 平台的.ipa 文件，或被打包进 Android 平台的.apk 文件，于是你便获得一个完整的、准备好发布的安装包！当应用被启动后，

运行器加载 Flutter 库以及你的应用代码，从这一刻开始，所有的渲染、输入输出事件处理将被委托给编译好的 Flutter 应用。

■ 注意：

这与大多数移动游戏引擎的运行方式相似。我以前写过一本关于Corona SDK的书，Corona SDK的运行方式与Flutter非常相似，尽管它使用Lua语言而非Dart语言(我打赌Flutter团队曾考虑将Lua列为备选语言)。我觉得这很有趣，Google最终从游戏引擎获得灵感来设计Flutter，这也证实了我一直所说的：要想成为一名合格的开发者，就写游戏吧！这是用来磨炼技能的最合适的项目类型。继续往后看，你会发现本书的最后两章将聚焦于使用Flutter构建一个游戏。

1.13 小结

本章开启了你的 Flutter 之旅！你了解了 Flutter 是什么，它能做什么，以及选择它的理由(甚至一些不选择它的理由)。你掌握了一些关键的概念，比如 Dart 和微件。你学会了如何设置开发环境以开发 Flutter 代码，还创建了第一个简单的 Flutter 应用并在模拟器上运行它。

在下一章中，你将掌握更多 Dart 知识点，建立良好的基础体系，以便在不久后可以使用 Flutter 开发一些真正的应用！