

创建第一个 Flutter APP

在安装完 Flutter 的开发环境之后，本章会分别使用 Android Studio 和 VS Code 创建你的第一个 Flutter APP。

使用 Android Studio 创建第一个 Flutter APP

这里使用 Android Studio 带你创建第一个 Flutter APP。

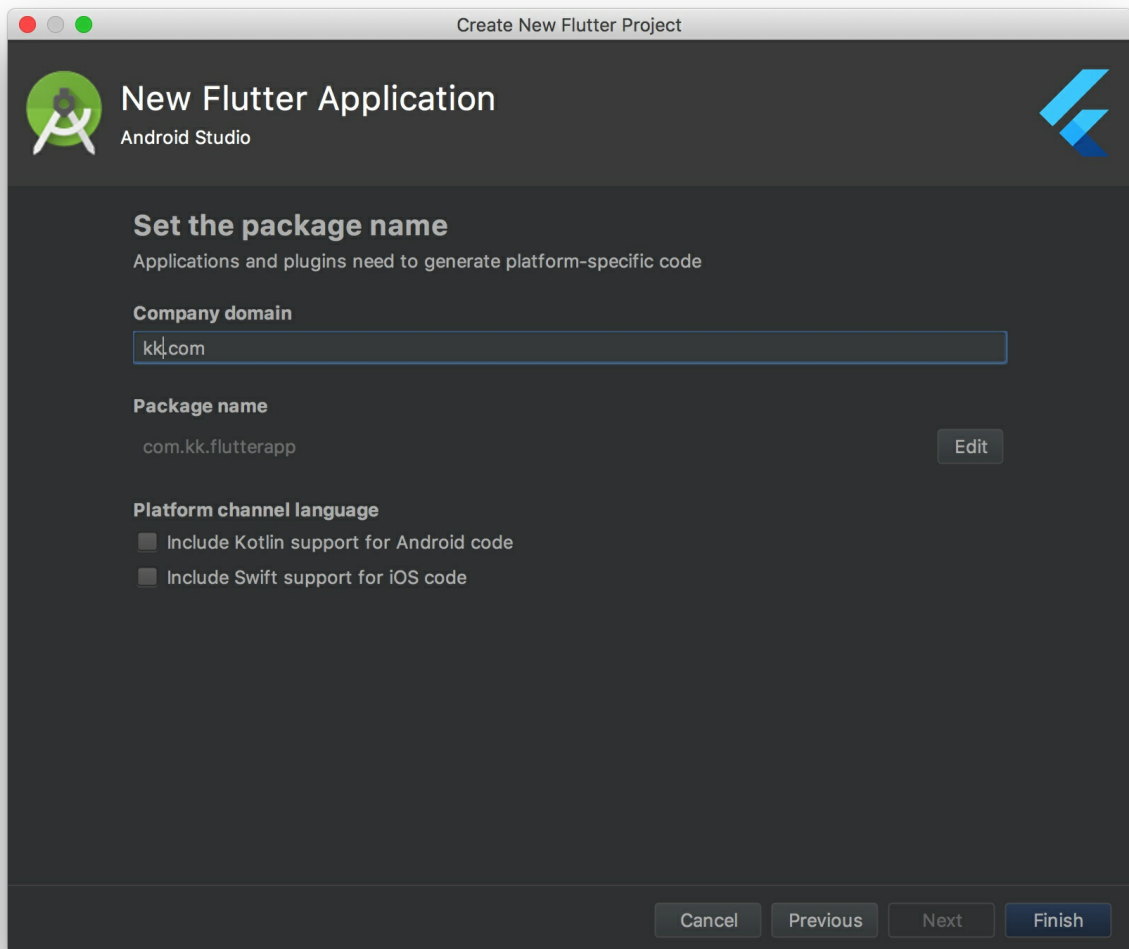
创建 Flutter 项目

在 Android Studio 中，选择 **File > New > New Flutter Project**，会出现下图的框：

选择 **Flutter Application**，然后点击 **Next**，出现如下图的框：

- Project name：工程的名字
- Flutter SDK path：Flutter SDK 的路径（选择前面 Flutter SDK 所在的路径）
- Project location：工程的路径
- Description：工程的描述

其中 **Flutter SDK path** 必须是 Flutter SDK 所在的路径，填完这些后，点击 **Next**，出现如下图的框：



这个是设置 Android 代码的包名。

下面的两个选项：

- Include Kotlin support for Android code（使用 Kotlin 开发）
- Include Swift support for iOS code（使用 Swift 开发）

如果想用 Kotlin 开发 Android 或者用 Swift 开发 iOS 的话，就选上，如果不选的话，Android 默认使用 Java 开发，iOS 默认使用 Objective-C 开发。可以根据自己的需要来选择。

最后点击 **Finish**，第一个 Flutter APP 就创建完成了，创建完的 Flutter APP 里有默认实现的代码。

运行 Flutter APP

创建完 Flutter APP 的工程后，工程里面有默认的实现，我们可以直接运行 Flutter APP，但在运行 Flutter APP 之前，首先要打开模拟器，或者连接真机。

打开 Android/iOS 模拟器

在 Android Studio 的工具栏里会看到如下图：

选择一个，打开模拟器。

在 MacOS 上，可以选择 iOS 模拟器，也可以选择 Android 模拟器。

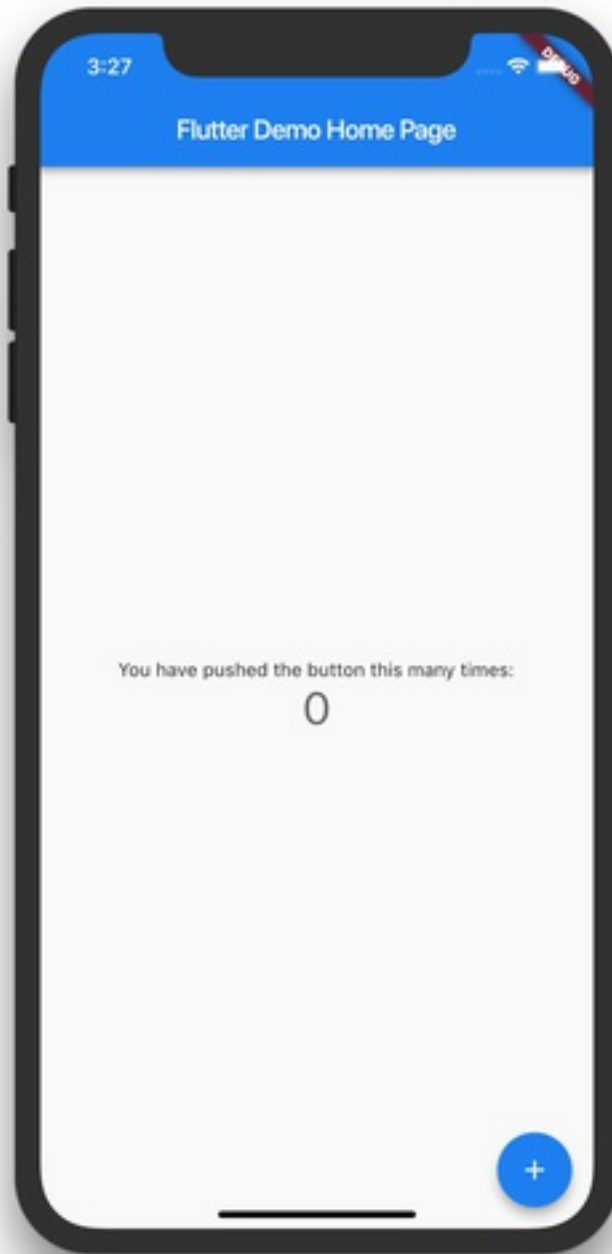
在 Windows 和 Linux 上，只能选择 Android 模拟器。

运行 Flutter APP

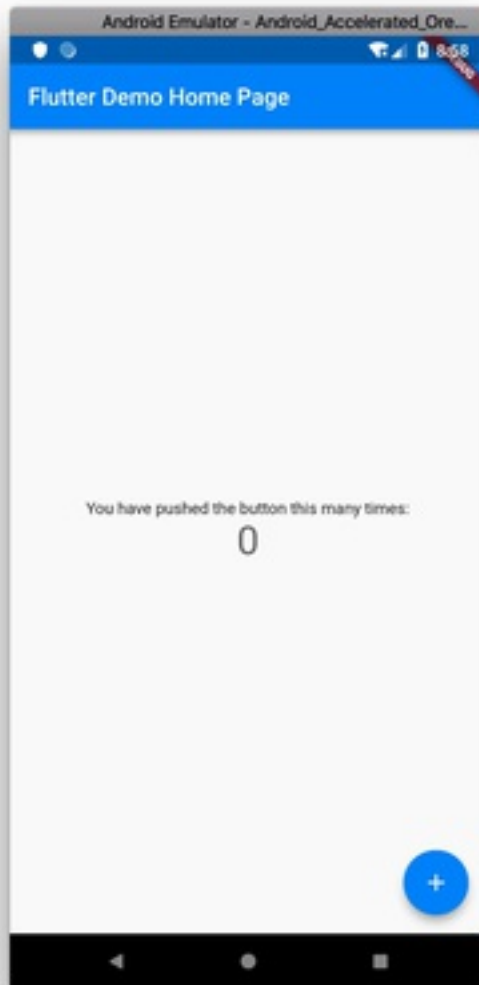
点击如下的按钮运行 Flutter APP：

运行的效果如下图：

iOS 模拟器：



Android 模拟器：



可以看到 iOS 里的 TitleBar 的文字是居中的，Android 里的 TitleBar 的文字是居左的，运行的是同一份代码，Flutter UI 可以根据平台来适配，完全不用我们担心适配的问题。

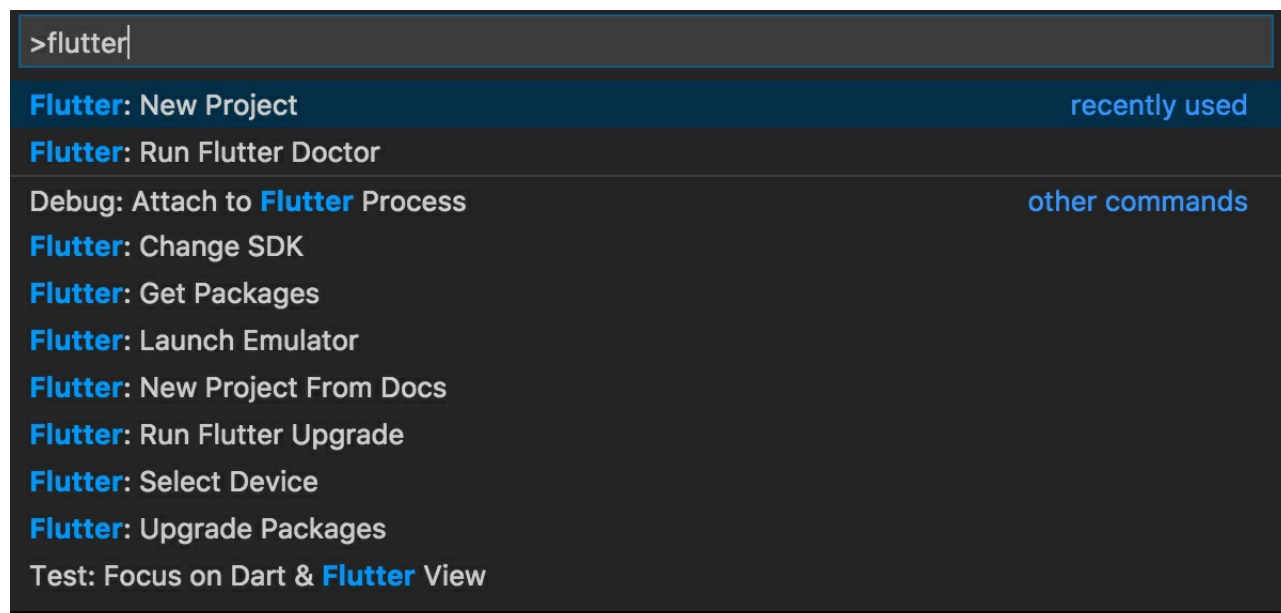
使用 VS Code 创建第一个 Flutter APP

这里使用 VS Code 带你创建第一个 Flutter APP。

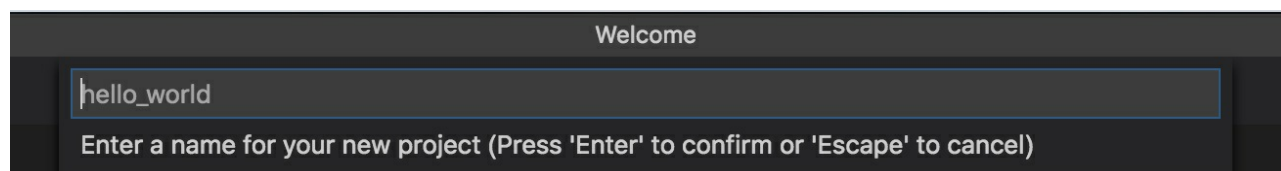
创建 Flutter 项目

在 VS Code 中, 点击 **View > Command Palette...**, 或者快捷键 **Shift+cmd+P**(MacOS) / **Ctrl+Shift+P**(Windows、Linux), 打开命令面板, 输入 **Flutter**

会出现下图:



选择 **Flutter: New Project**, 会弹出如下的框:



然后你输入 Flutter 工程的名字, 你可以自己起一个, 例如: `hello_world`。输入 Flutter 工程的名字后, 回车, 会弹出如下的框:

选择好工程的路径, 点击右下角的 **Select a folder to create the project in**, 第一个 Flutter APP 就创建完成了, 创建完的 Flutter APP 里有默认实现的代码。

这里相比 Android Studio 少了设置 Android 包名的步骤, 所以用 VS Code 创建的 Flutter APP 的 Android 包名用的是默认的。

VS Code 实质是一款代码编辑器，如果你用过 Sublime 的话，应该就很熟悉，最重要的是记住命令面板 的快捷键：Shift+cmd+P(MacOS) /Ctrl+Shift+P(Windows、Linux)，很多功能需要在命令面板中使用。

运行 Flutter APP

创建完 Flutter APP 的工程后，工程里面有默认的实现，我们可以直接运行 Flutter APP，但在运行 Flutter APP 之前，首先要打开模拟器，或者连接真机。

打开 Android/iOS 模拟器

在 VS Code 的底部工具栏里会看到如下图：

点击红框内的 **No Devices** ，会看到如下图：

选择一个，打开模拟器。

在 MacOS 上，可以选择 iOS 模拟器，也可以选择 Android 模拟器。

在 Windows 和 Linux 上，只能选择 Android 模拟器。

运行Flutter APP

VS Code 有两种方式运行 Flutter APP：

- Start Debugging
- Start Without Debugging

Start Debugging

点击 **Debug > Start Debugging**。

这个模式下加了调试器，如果有断点，可以跟进调试代码。

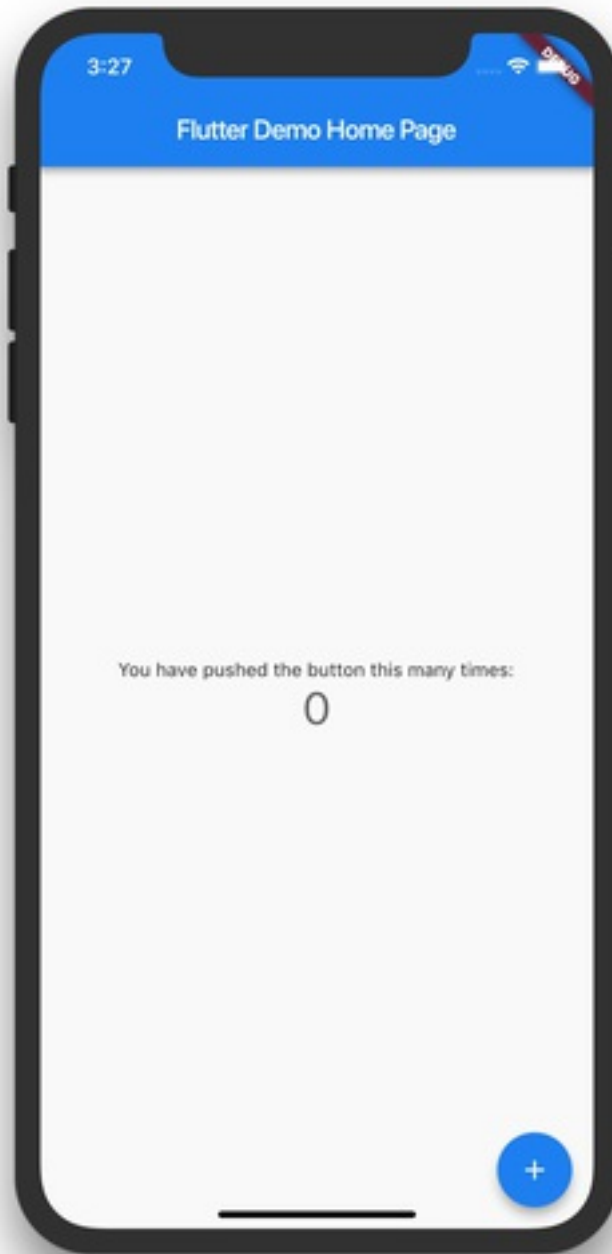
Start Without Debugging

点击 **Debug > Start Without Debugging**。

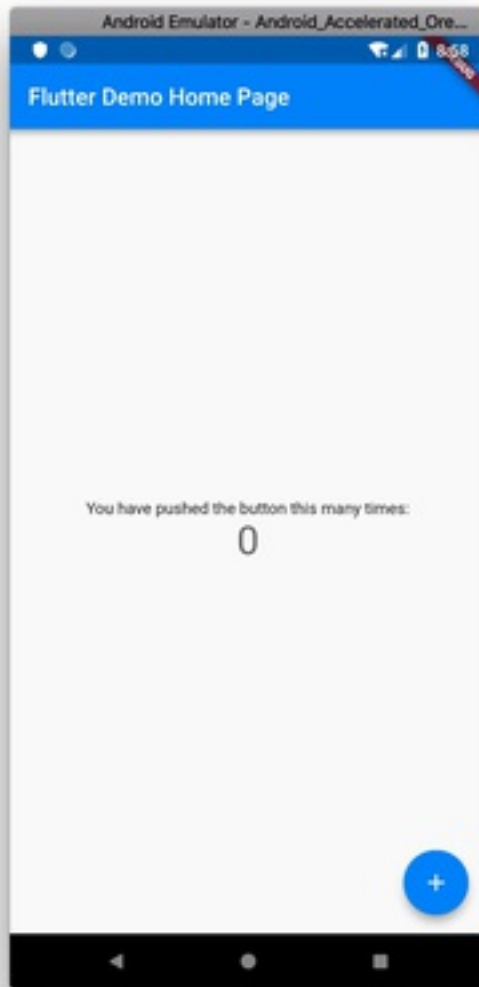
这个模式下没有调试器，就是启动已经编译好的程序。

运行的效果如下图：

iOS模拟器：



Android 模拟器：



可以看到 iOS 里的 TitleBar 的文字是居中的，Android 里的 TitleBar 的文字是居左的，运行的是同一份代码，Flutter UI 可以根据平台来适配，完全不用我们担心适配的问题。

Flutter 的调试信息

Flutter 运行之后，可以在 VS Code 的下方看到调试信息：

VS Code 简单使用说明

VS Code 有一个简单直观的布局，它的用户界面分为五个方面：

- (A) 视图工具栏：位于最左边的视图栏，包括文件目录、搜索等功能。
- (B) 侧边框：包含不同的管理器，可以打开不同的功能。
- (C) 编辑区域：来编辑你的文件，可以同时打开三个文件编辑。
- (D) 面板：在编辑区下面有不同的面板，面板有输入或调试信息，错误和警告或一个集成终端。
- (E) 状态栏：显示打开的项目和编辑的文件的相关信息。

面板

点击侧边栏的

红色部分，可以打开、隐藏面板。

面板里有四个 tab：

我们主要使用如下的三个：

1. PROBLEMS

查看代码错误的地方：

2. DEBUG CONSOLE

查看调试信息和日志输出的地方：

3. TERMINAL

终端：

Flutter 入口文件

在创建完 Flutter APP 之后，就可以开始 Flutter 的开发了，那么代码应该写在哪里，APP 是从哪里开始运行的呢？

所以我们有必要先看一下 Flutter 代码的入口文件及入口函数。

Flutter 入口文件是 `main.dart`，入口文件的名字必须是 `main.dart`，不能更改，在 Flutter 工程的 `lib` 目录下：

Flutter 入口函数

打开 `main.dart`：

这里的 `main()` 函数就是 Flutter 的入口函数。

在 `main()` 函数里要运行 `runApp()` 函数，`runApp()` 函数的参数类型是 `Widget` 类型。使用的方法如下：

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

这个使用方法是固定的。

Flutter 工程类型介绍

前面用 Android Studio 创建 Flutter 工程的时候，让选 Flutter 工程类型：

可以见到 Flutter 有四种工程类型，这四种工程类型有不同的应用场景。

1. Flutter Application

如果能让 Flutter 直接运行在 Android 或 iOS 上给用户使用，就用 Application 类型。

本小册里的所有 Flutter 示例都是 Application 类型的。

2. Flutter Plugin

如果你想让 Flutter 调用 Android 或 iOS 的 API，并将这个功能封装起来供第三方使用，那么就用 Plugin 类型。

Plugin 是 Flutter 的一个插件包，例如，在 Android 上有播放音乐的功能，为了在 Flutter 中使用，就可以使用 Plugin 的模式：播放音乐的功能在 Android 上用 Java 实现，然后 Flutter 的代码通过 PlatformChannel 使用 Android 的播放音乐的功能，这就是一个完整的插件包，提供了播放音乐的功能。其他 Flutter APP 想要使用播放音乐的功能，就可以依赖这个插件包。

Flutter 想要使用 Android 或 iOS 的功能，都可以使用 Plugin 的模式。

3. Flutter Package

Flutter Package 是纯 Dart 模块，只能实现 Flutter 的相关功能，例如实现一个 Widget，然后给第三方使用。

4. Flutter Module

如果要将 Flutter 添加到 Android APP 或 iOS APP 中，实现 Flutter 和 Native 的混合开发，就使用 Module 类型。

Flutter 的混合开发模式就采用的这种类型，在 Android 上打包成 aar，在 iOS 上打包成 Framework，可以很方便的集成进原有的 Native 工程里，实现 Flutter 和 Native 的混合开发。

Plugin 和 Package

Flutter 的 Plugin 和 Package 可以用于组件化开发。

可以在 <https://pub.dartlang.org/> (<https://pub.dartlang.org/>) 网站上看到所有的 Flutter 库。

或者在国内镜像 <https://pub.flutter-io.cn/> (<https://pub.flutter-io.cn/>) 上查看 Flutter 库。

Flutter 不同工程的代码目录结构介绍

1. Flutter Application 目录结构

如下图是前面创建的 Flutter APP 的目录截图：

- android 目录

这个目录下是一个完整的 Android APP 工程的代码。可以理解成 Flutter 在 Android 上的壳子。这个目录里的代码都会被打包进 Flutter 的 Android 安装包里。

- ios 目录

这个目录下是一个完整的 iOS APP 工程的代码。可以理解成 Flutter 在 iOS 上的壳子。这个目录里的代码都会被打包进 Flutter 的 iOS 安装包里。

- lib 目录

这里是 Flutter 的代码，使用 Dart 语言编写。main.dart 是 Flutter 的入口文件。

- test 目录

这里是 Flutter 的测试代码，使用 Dart 语言编写。

- pubspec.yaml 文件

这个是 Flutter 的配置文件，声明了 Flutter APP 的名称、版本、作者等的元数据文件，还有声明的依赖库，和指定的本地资源（图片、字体、音频、视频等）。

pubspec.yaml 是 Flutter 的配置，是 Flutter 里的重要部分。

Flutter Application 包含 Dart 代码、Android 代码和 iOS 代码，可以直接生成 Android 安装包和 iOS 安装包。

2. Flutter Plugin 目录结构

如下图是 Flutter Plugin 的目录截图：

- android 目录

这个目录下是一个 Android Module 工程的代码。这个目录里的代码都会被打包进 Flutter Plugin。

- ios 目录

这个目录下是一个 iOS 静态库的代码。这个目录里的代码都会被打包进 Flutter Plugin。

- example 目录

这个目录下是一个完整的 Flutter APP 的应用代码，是为了方便 Plugin 代码的开发，在开发时 Plugin 的代码是运行在这个 Flutter APP 上的，但当打包的时候，example 下的代码并不会打包进 Plugin。

- lib 目录

这里是 Flutter 的代码，使用 Dart 语言编写。`main.dart` 是 Flutter 的入口文件。

- pubspec.yaml 文件

这个是 Flutter 的配置文件，声明了 Flutter APP 的名称、版本、作者等的元数据文件，还有声明的依赖库，和指定的本地资源（图片、字体、音频、视频等）。

Flutter Plugin 里既有 Dart 代码，也有 Android 的代码，和 iOS 的代码。

3. Flutter Package 目录结构

如下图是 Flutter Package 的目录截图：

- lib 目录

这里是 Flutter 的代码，使用 Dart 语言编写。`main.dart` 是 Flutter 的入口文件。

- test 目录

这里是 Flutter 的测试代码，使用 Dart 语言编写。

- pubspec.yaml 文件

这个是 Flutter 的配置文件，声明了 Flutter APP 的名称、版本、作者等的元数据文件，还有声明的依赖库，和指定的本地资源（图片、字体、音频、视频等）。

4. Flutter Module 目录结构

如下图是 Flutter Module 的目录截图：

- .android 目录

这个目录下是一个完整的 Android APP 工程的代码，具体如下图：

这个目录下的 Android APP 工程是为了方便开发和调试，因为 Flutter Module 打包生成的是 aar，没办法运行，所以加了 .android 目录，使 Flutter Module 在开发的时候可以运行在 APP 上，但打包的时候 只有Flutter目录下的Android代码才会被打包进 aar。

- .ios 目录

这个目录下是一个完整的iOS APP工程的代码。具体如下图：

.ios 目录和 .android 目录的功能是一样的，最终打包的时候只有 Flutter 目录下的 iOS 代码才会被打包进 Framework。

- lib 目录

这里是 Flutter 的代码，使用 Dart 语言编写。main.dart 是 Flutter 的入口文件。

- test 目录

这里是 Flutter 的测试代码，使用 Dart 语言编写。

- pubspec.yaml 文件

这个是 Flutter 的配置文件，声明了 Flutter APP 的名称、版本、作者等的元数据文件，还有声明的依赖库，和指定的本地资源（图片、字体、音频、视频等）。

Flutter Module 中既包含 Dart 代码，也有 Android 的代码和 iOS 的代码。

pubspec.yaml 介绍

pubspec.yaml 是 Flutter 工程的配置文件，使用 YAML 语言来写，下面是一个 Flutter 工程的 pubspec.yaml：

```

name: flutter_doubanmovie
description: A new Flutter project.

version: 1.0.0+1

authors:
- Natalie Weizenbaum <nweiz@google.com>
- Bob Nystrom <rnystrom@google.com>

homepage: https://flutter.dev/

environment:
  sdk: ">=2.1.0 <3.0.0"

dependencies:
  flutter:
    sdk: flutter

  cupertino_icons: ^0.1.2
  http: ^0.12.0+2
  shared_preferences: ^0.5.2

dev_dependencies:
  flutter_test:
    sdk: flutter

flutter:

  uses-material-design: true

```

下表是 pubspec.yaml 支持的字段：

字段名	含义	可选/必选
name	工程的名字	必选

name	工程的名字	必填 想要发布到 Pub (https://pub.dev/)
description	工程的描述	上，就是必须的 想要发布到 Pub (https://pub.dev/)
version	工程的版本号	上，就是必须的
author or authors	作者名字	可选
homepage	主页	可选
environment	指定 Dart 的版本，因为 随着时间的推移，Dart 不断发展，一个软件包可能只适用于某些版本的平台。	必填
repository	指向工程的源代码的地址	可选
issue_tracker	指向跟踪工程issue的地址	可选
documentation	指向工程文档的地址	可选
dependencies	依赖的开发库	如果你的工程没有依赖的话，可以省略
dev_dependencies	依赖的测试库	如果你的工程没有依赖的话，可以省略
dependency_overrides	在开发过程中，您可能需要暂时覆盖依赖项。	如果你的工程不需要要覆盖依赖的话，可以省略
executables	用于将包的可执行文件放在PATH上：可以将其一个或多个脚本公开为可以直接从命令行运行的可执行文件。	可选

publish_to	指定发布包的位置，默认是 Pub (https://pub.dev/)	可选
flutter	flutter 资源相关的配置，包括图片、字体等，后面会有具体场景	必选