

# 在Flutter中添加资源和图片

- [介绍](#)
- [指定 assets](#)
  - [Asset 变体 \( variant \)](#)
- [加载 assets](#)
  - [加载文本assets](#)
  - [加载 images](#)
    - [声明分辨率相关的图片 assets](#)
    - [加载图片](#)
  - [依赖包中的资源图片](#)
    - [打包 package assets](#)
- [平台 assets](#)
  - [更新app 图标](#)
    - [Android](#)
    - [iOS](#)
  - [更新启动页](#)
    - [Android](#)
    - [iOS](#)

## 介绍

Flutter应用程序可以包含代码和 `assets` ( 有时称为资源 )。asset是打包到程序安装包中的，可在运行时访问。常见类型的asset包括静态数据 ( 例如JSON文件 )，配置文

件，图标和图片（JPEG，WebP，GIF，动画WebP / GIF，PNG，BMP和WBMP）。

## 指定 assets

Flutter使用[pubspec.yaml](#)文件（位于项目根目录），来识别应用程序所需的asset。

这里是一个例子：

```
flutter:
  assets:
    - assets/my_icon.png
    - assets/background.png
```

该assets部分的flutter部分指定应包含在应用程序中的文件。每个asset都通过相对于pubspec.yaml文件所在位置的显式路径进行标识。asset的声明顺序是无关紧要的。

asset的实际目录可以是任意文件夹（在本示例中是assets）。

在构建期间，Flutter将asset放置到称为 *asset bundle* 的特殊存档中，应用程序可以在运行时读取它们。

### Asset 变体（variant）

构建过程支持asset变体的概念：不同版本的asset可能会显示在不同的上下文中。在pubspec.yaml的assets部分中指定asset路径时，构建过程中，会在相邻子目录中查找具有相同名称的任何文件。这些文件随后会与指定的asset一起被包含在asset bundle中。

例如，如果您的应用程序目录中有以下文件：

- .../pubspec.yaml
- .../graphics/my\_icon.png
- .../graphics/background.png
- .../graphics/dark/background.png
- ...etc.

...然后您的pubspec.yaml文件包含：

```
flutter:
  assets:
    - graphics/background.png
```

.....那么这两个graphics/background.png和graphics/dark/background.png 将包含在您的asset bundle中。前者被认为是\_main asset\_，后者被认为是一种变体（variant）。

在选择匹配当前设备分辨率的图片时，Flutter使用asset变体；见下文。将来，这种机制可能会扩展到本地化、阅读提示等方面。

## 加载 assets

您的应用可以通过[AssetBundle](#)对象访问其asset。

有两种主要方法允许从Asset bundle中加载字符串/text ( `loadString` ) 或图片/二进制 ( `load` )。

### 加载文本assets

每个Flutter应用程序都有一个[rootBundle](#)对象，可以轻松访问主资源包。可以直接使用 `package:flutter/services.dart` 中全局静态的 `rootBundle` 对象来加载asset。

但是，建议使用[DefaultAssetBundle](#)来获取当前BuildContext的AssetBundle。这种方法不是使用应用程序构建的默认asset bundle，而是使父级widget在运行时替换的不同的AssetBundle，这对于本地化或测试场景很有用。

通常，可以使用[DefaultAssetBundle.of\(\)](#)从应用运行时间接加载asset ( 例如JSON文件 )。

在Widget上下文之外，或AssetBundle的句柄不可用时，您可以使用[rootBundle](#)直接加载这些asset，例如：

```
import 'dart:async' show Future;
import 'package:flutter/services.dart' show rootBundle;

Future<String> loadAsset() async {
  return await rootBundle.loadString('assets/config.json');
}
```

### 加载 images

Flutter可以为当前设备加载适合其分辨率的图像。

声明分辨率相关的图片 assets

[AssetImage](#) 了解如何将逻辑请求asset映射到最接近当前设备像素比例的asset。为了使这种映射起作用，应该根据特定的目录结构来保存asset

- .../image.png
- .../Mx/image.png
- .../Nx/image.png
- ...etc.

其中M和N是数字标识符，对应于其中包含的图像的分辨率，也就是说，它们指定不同设备像素比例的图片

主资源默认对应于1.0倍的分辨率图片。看一个例子：

- .../my\_icon.png

- .../2.0x/my\_icon.png
- .../3.0x/my\_icon.png

在设备像素比率为1.8的设备上，.../2.0x/my\_icon.png 将被选择。对于2.7的设备像素比率，.../3.0x/my\_icon.png将被选择。

如果未在Image控件上指定渲染图像的宽度和高度，以便它将占用与主资源相同的屏幕空间量（并不是相同的物理像素），只是分辨率更高。也就是说，如果.../my\_icon.png是72px乘72px，那么.../3.0x/my\_icon.png应该是216px乘216px; 但如果未指定宽度和高度，它们都将渲染为72像素×72像素（以逻辑像素为单位）。

pubspec.yaml中asset部分中的每一项都应与实际文件相对应，但主资源项除外。当主资源缺少某个资源时，会按分辨率从低到的顺序去选择（译者语：也就是说1x中没有的话会在2x中找，2x中还没有的话就在3x中找）。

加载图片

要加载图片，请在widget的build方法中使用 [AssetImage](#)类。

例如，您的应用可以从上面的asset声明中加载背景图片：

```
Widget build(BuildContext context) {
  // ...
  return new DecoratedBox(
    decoration: new BoxDecoration(
      image: new DecorationImage(
        image: new AssetImage('graphics/background.png'),
        // ...
      ),
      // ...
    ),
  );
  // ...
}
```

使用默认的 asset bundle 加载资源时，内部会自动处理分辨率等，这些处理对开发者来说是无感知的。（如果您使用一些更低级别的类，如 [ImageStream](#)或 [ImageCache](#), 您会注意到有与缩放相关的参数）

依赖包中的资源图片

要加载依赖包中的图像，必须给AssetImage提供package参数。

例如，假设您的应用程序依赖于一个名为“my\_icons”的包，它具有如下目录结构：

- .../pubspec.yaml
- .../icons/heart.png

- .../icons/1.5x/heart.png
- .../icons/2.0x/heart.png
- ...etc.

然后加载图像，使用：

```
new AssetImage('icons/heart.png', package: 'my_icons')
```

包使用的本身的资源也应该加上`package`参数来获取。

打包 `package assets`

如果在`pubspec.yaml`文件中声明了期望的资源，它将会打包到响应的`package`中。特别是，包本身使用的资源必须在`pubspec.yaml`中指定。

包也可以选择在其`lib/`文件夹中包含未在其`pubspec.yaml`文件中声明的资源。在这种情况下，对于要打包的图片，应用程序必须在`pubspec.yaml`中指定包含哪些图像。例如，一个名为“fancy\_backgrounds”的包，可能包含以下文件：

- .../lib/backgrounds/background1.png
- .../lib/backgrounds/background2.png
- .../lib/backgrounds/background3.png

要包含第一张图像，必须在`pubspec.yaml`的`assets`部分中声明它：

```
flutter:
```

```
  assets:
```

```
    - packages/fancy_backgrounds/backgrounds/background1.png
```

The `lib/` is implied, so it should not be included in the asset path.

`lib/`是隐含的，所以它不应该包含在资产路径中。

## 平台 assets

也有时候可以直接在平台项目中使用`asset`。以下是在Flutter框架加载并运行之前使用资源的两种常见情况。

更新app 图标

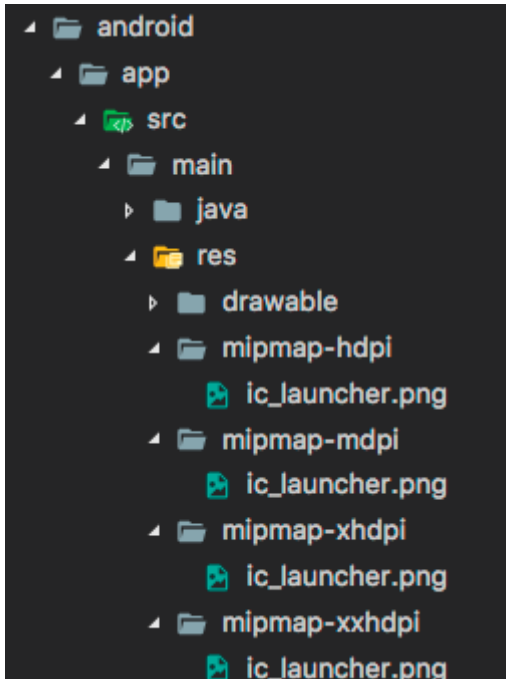
更新您的Flutter应用程序的启动图标的方式与在本机Android或iOS应用程序中更新启动图标的方式相同



Launch icon

Android

在Flutter项目的根目录中，导航到`.../android/app/src/main/res`。各种位图资源文件夹（如`mipmap-hdpi`已包含占位符图像“`ic_launcher.png`”）。只需按照[Android开发人员指南](#)中的说明，将其替换为所需的资源，并遵守每种屏幕密度的建议图标大小标准。

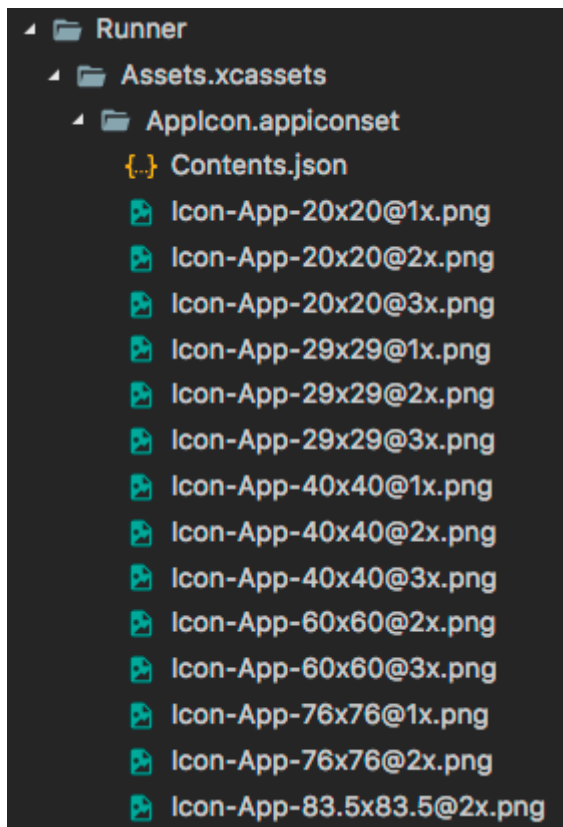


Android icon location

**注意:** 如果您重命名.png文件，则还必须在您的`AndroidManifest.xml`的`<application>`标签的`android:icon`属性中更新名称。

iOS

在你的Flutter项目的根目录中，导航到`.../ios/Runner`。该目录中`Assets.xcassets/AppIcon.appiconset`已经包含占位符图片。只需将它们替换为适当大小的图片。保留原始文件名称。



iOS icon location

更新启动页



Launch screen

在Flutter框架加载时，Flutter会使用本地平台机制绘制启动页。此启动页将持续到Flutter渲染应用程序的第一帧时。

**注意:** 这意味着如果您不在应用程序的`main()`方法中调用`runApp`函数（或者更具体地说，如果您不调用`window.render`去响应`window.onDrawFrame`）的话，启动屏幕将永远持续显示。

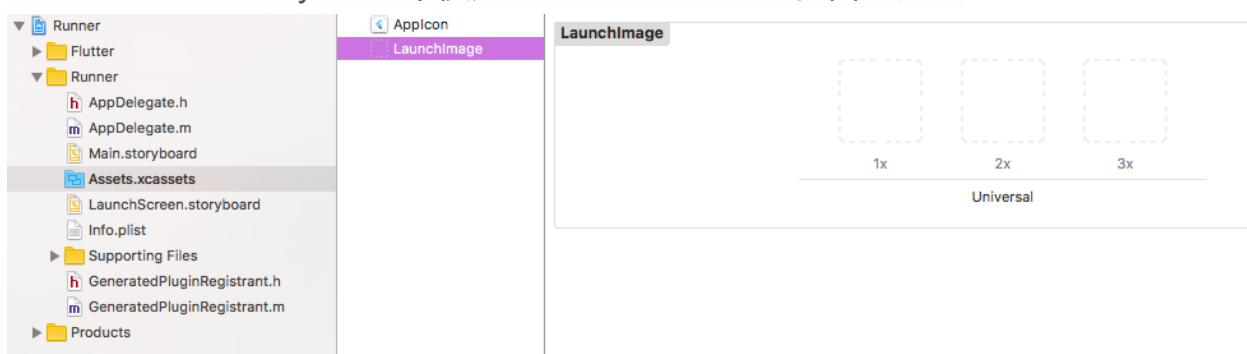
Android

要将启动屏幕（splash screen）添加到您的Flutter应用程序，请导航至`.../android/app/src/main`。在`res/drawable/launch_background.xml`，通过自定义drawable来实现自定义启动界面。

iOS

要将图片添加到启动屏幕（splash screen）的中心，请导航至`.../ios/Runner`。在`Assets.xcassets/LaunchImage.imageset`，拖入图片，并命名为`images LaunchImage.png`、`LaunchImage@2x.png`、`LaunchImage@3x.png`。如果您使用不同的文件名，那您还必须更新同一目录中的`Contents.json`文件。

您也可以通过打开Xcode完全自定义storyboard。在Project Navigator中导航到`Runner/Runner`然后通过打开`Assets.xcassets`拖入图片，或者通过在`LaunchScreen.storyboard`中使用Interface Builder进行自定义。



Adding launch icons in Xcode