

# Flutter 混合开发

公众号：coderwhy

## Flutter 混合开发

原创 coderwhy coderwhy

### 一. 调用原生功能

#### 1.1. Camera

某些应用程序可能需要使用移动设备进行拍照或者选择相册中的照片，Flutter官方提供了插件：image\_picker

##### 1.1.1. 添加依赖

添加对image\_picker的依赖：

- [https://pub.dev/packages/image\\_picker](https://pub.dev/packages/image_picker)

```
dependencies:  
  image_picker: ^0.6.5
```

##### 1.1.2. 平台配置

对iOS平台，想要访问相册或者相机，需要获取用户的允许：

- 依然是修改info.plist文件：/ios/Runner/Info.plist
- 添加对相册的访问权限：Privacy - Photo Library Usage Description
- 添加对相机的访问权限：Privacy - Camera Usage Description

▼ Information Property List		
	Dictionary	(17 items)
Privacy - Photo Library Usage Description	String	访问相册
Privacy - Camera Usage Description	String	访问相机
Localization native development region	String	coderwhy \$(DEVELOPMENT_LANGUAGE)

修改info.plist文件

之后选择相册或者访问相机时，会弹出如下的提示框：



image-20200412171028021

##### 1.1.3. 代码实现

image\_picker的核心代码是pickImage方法：

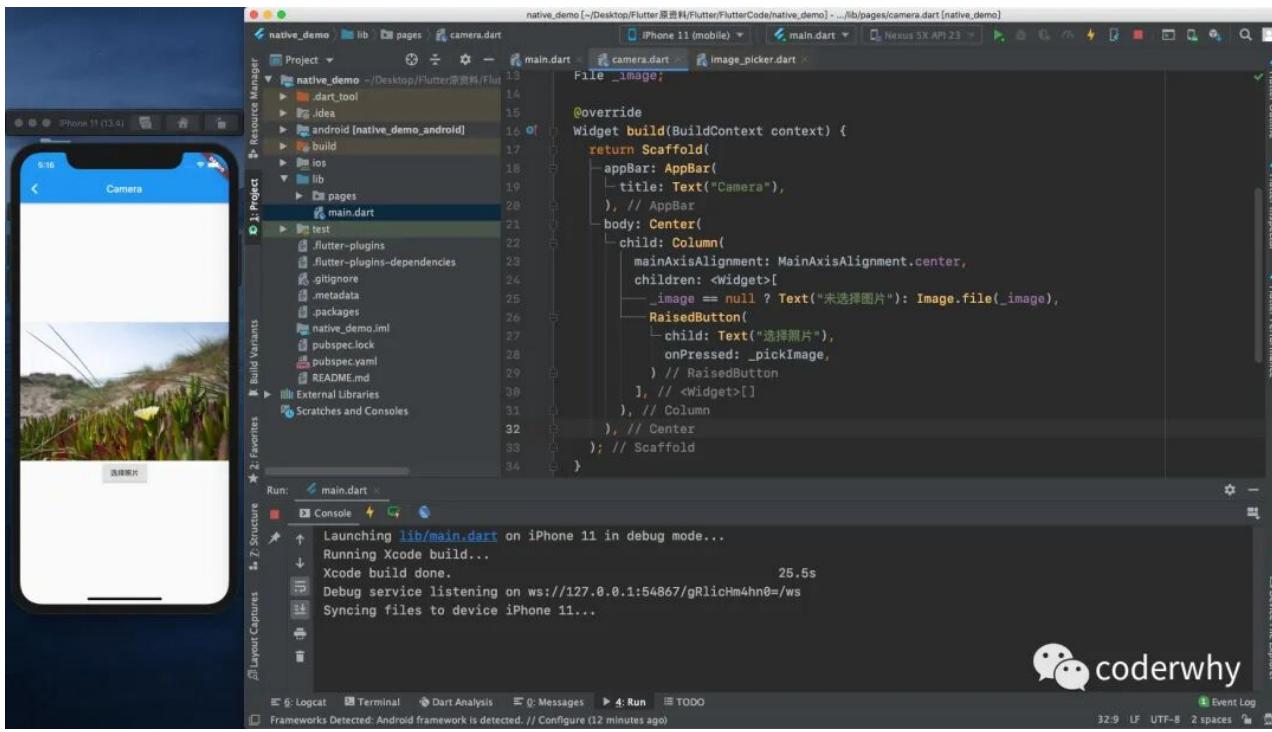
- 可以传入数据源、图片的大小、质量、前置后置摄像头等
- 数据源是必传参数：ImageSource枚举类型

- camera: 相机
- gallery: 相册

```
static Future<File> pickImage(  
    @required ImageSource source,  
    double maxWidth,  
    double maxHeight,  
    int imageQuality,  
    CameraDevice preferredCameraDevice = CameraDevice.rear)) async
```

案例演练：

```
import 'package:flutter/material.dart';  
import 'dart:io';  
import 'package:image_picker/image_picker.dart';  
  
class HYCameraScreen extends StatefulWidget {  
  static const String routeName = "/camera";  
  
  @override  
  _HYCameraScreenState createState() => _HYCameraScreenState();  
}  
  
class _HYCameraScreenState extends State<HYCameraScreen> {  
  File _image;  
  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: Text("Camera"),  
      ),  
      body: Center(  
        child: Column(  
          mainAxisAlignment: MainAxisAlignment.center,  
          children: <Widget>[  
            _image == null ? Text("未选择图片"): Image.file(_image),  
            RaisedButton(  
              child: Text("选择照片"),  
              onPressed: _pickImage,  
            ),  
          ],  
        ),  
      );  
  }  
  
  void _pickImage() async {  
    File image = await ImagePicker.pickImage(source: ImageSource.gallery);  
    setState(() {  
      _image = image;  
    });  
  }  
}
```



案例效果

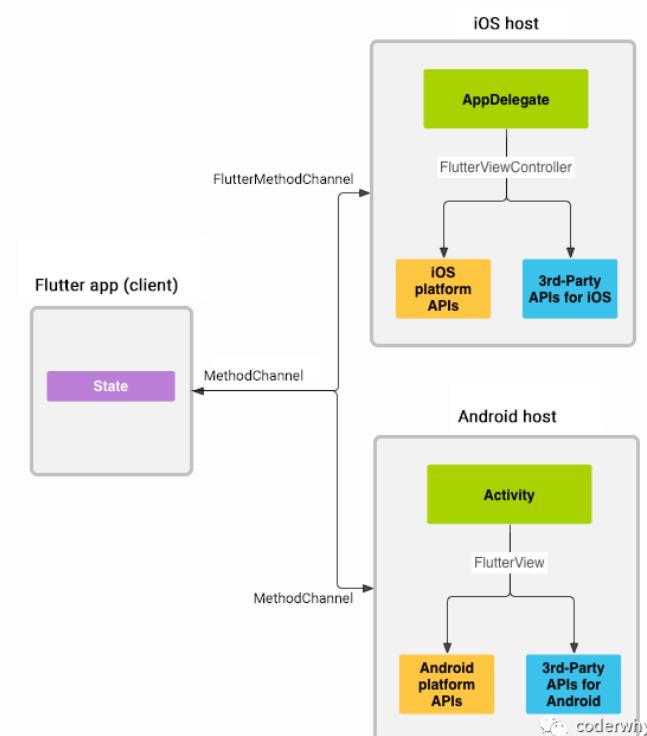
## 1.2. 电池信息

某些原生的信息，如果没有很好的插件，我们可以通过platform channels（平台通道）来获取信息。

### 1.2.1. 平台通过介绍

平台通过是如何工作的呢？

- 消息使用platform channels（平台通道）在客户端（UI）和宿主（平台）之间传递；
- 消息和响应以 异步 的形式进行传递，以确保用户界面能够保持响应；



平台通道架构图

调用过程大致如下：

- 1. 客户端（Flutter端）发送与方法调用相对应的消息
- 2. 平台端（iOS、Android端）接收方法，并返回结果；

- iOS端通过 `FlutterMethodChannel` 做出响应；
- Android端通过 `MethodChannel` 做出响应；

Flutter、iOS、Android端数据类型的对应关系：

Dart	Java	Kotlin	OC	Swift
null	null	null	nil (NSNull when nested)	nil
bool	java.lang.Boolean	Boolean	NSNumber numberWithBool:	NSNumber(value: Bool)
int	java.lang.Integer	Int	NSNumber numberWithInt:	NSNumber(value: Int32)
int, if 32 bits not enough	java.lang.Long	Long	NSNumber numberWithLong:	NSNumber(value: Int)
double	java.lang.Double	Double	NSNumber numberWithDouble:	NSNumber(value: Double)
String	java.lang.String	String	NSString	String
Uint8List	byte[]	ByteArray	FlutterStandardTypedData typedDataWithBytes:	FlutterStandardTypedData(bytes: Data)
Int32List	int[]	IntArray	FlutterStandardTypedData typedDataWithInt32:	FlutterStandardTypedData(int32: Data)
Int64List	long[]	LongArray	FlutterStandardTypedData typedDataWithInt64:	FlutterStandardTypedData(int64: Data)
Float64List	double[]	DoubleArray	FlutterStandardTypedData typedDataWithFloat64:	FlutterStandardTypedData(float64: Data)
List	java.util.ArrayList	List	NSArray	NSArray
Map	java.util.HashMap	HashMap	NSDictionary	NSDictionary

数据类型对应表

### 1.2.2. 创建测试项目

我们这里创建一个获取电池电量信息的项目，分别通过iOS和Android原生代码来获取对应的信息：

创建方式一：默认创建方式

- 目前默认创建的Flutter项目，对应iOS的编程语言是Swift，对应Android的编程语言是kotlin

```
flutter create batterylevel
```

创建方式二：指定编程语言

- 如果我们希望指定编程语言，比如iOS编程语言为Objective-C，Android的编程语言为Java

```
flutter create -i objc -a java batterylevel2
```

### 1.2.3. 编写Dart代码

在Dart代码中，我们需要创建一个MethodChannel对象：

- 创建该对象时，需要传入一个name，该name是区分多个通信的名称
- 可以通过调用该对象的invokeMethod来给对应的平台发送消息进行通信
  - 该调用是异步操作，需要通过await获取then回调来获取结果

```

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

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


class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue, splashColor: Colors.transparent),
      home: HYBatteryScreen(),
    );
  }
}

class HYBatteryScreen extends StatefulWidget {
  static const String routeName = "/battery";

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

class _HYBatteryScreenState extends State<HYBatteryScreen> {
  // 核心代码一:
  static const platform = const MethodChannel("coderwhy.com/battery");
  int _result = 0;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Battery"),
      ),
      body: Center(
        child: Column(
          children: <Widget>[
            Text("当前电池信息: $_result"),
            RaisedButton(
              child: Text("获取电池信息"),
              onPressed: getBatteryInfo,
            )
          ],
        ),
      ),
    );
  }

  void getBatteryInfo() async {
    // 核心代码二
    final int result = await platform.invokeMethod("getBatteryInfo");
    setState(() {
      _result = result;
    });
  }
}

```

当我们通过 `platform.invokeMethod` 调用对应平台方法时，需要在对应的平台实现其操作：

- iOS中可以通过Objective-C或Swift来实现
- Android中可以通过Java或者Kotlin来实现

## 1.2.4. 编写iOS代码

### 1.2.4.1. Swift代码实现

代码相关的操作步骤如下：

- 1.获取FlutterViewController(是应用程序的默认Controller)
- 2.获取MethodChannel(方法通道)
  - 注意：这里需要根据我们创建时的名称来获取
- 3.监听方法调用(会调用传入的回调函数)
  - iOS中获取信息的方式
  - 如果没有获取到,那么返回给Flutter端一个异常

- 通过result将结果回调给Flutter端
- 3.1. 判断是否是getBatteryInfo的调用, 告知Flutter端没有实现对应的方法
- 3.2. 如果调用的是getBatteryInfo的方法, 那么通过封装的另外一个方法实现回调

```

import UIKit
import Flutter

@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate {
  override func application(
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?
  ) -> Bool {
    // 1. 获取FlutterViewController(是应用程序的默认Controller)
    let controller : FlutterViewController = window?.rootViewController as! FlutterViewController

    // 2. 获取MethodChannel(方法通道)
    let batteryChannel = FlutterMethodChannel(name: "coderwhy.com/battery",
                                                binaryMessenger: controller.binaryMessenger)

    // 3. 监听方法调用(会调用传入的回调函数)
    batteryChannel.setMethodCallHandler({
      [weak self] (call: FlutterMethodCall, result: FlutterResult) -> Void in
      // 3.1. 判断是否是getBatteryInfo的调用, 告知Flutter端没有实现对应的方法
      guard call.method == "getBatteryInfo" else {
        result(FlutterMethodNotImplemented)
        return
      }
      // 3.2. 如果调用的是getBatteryInfo的方法, 那么通过封装的另外一个方法实现回调
      self?.receiveBatteryLevel(result: result)
    })

    GeneratedPluginRegistrant.register(with: self)
    return super.application(application, didFinishLaunchingWithOptions: launchOptions)
  }

  private func receiveBatteryLevel(result: FlutterResult) {
    // 1. iOS中获取信息的方式
    let device = UIDevice.current
    device.isBatteryMonitoringEnabled = true

    // 2. 如果没有获取到, 那么返回给Flutter端一个异常
    if device.batteryState == UIDevice.BatteryState.unknown {
      result(FlutterError(code: "UNAVAILABLE",
                           message: "Battery info unavailable",
                           details: nil))
    } else {
      // 3. 通过result将结果回调给Flutter端
      result(Int(device.batteryLevel * 100))
    }
  }
}

```

#### 1.2.4.2. Objective-C代码实现

实现思路和上面是一致的, 只是使用了Objective-C来实现:

- 可以参考注释内容

```

#import <Flutter/Flutter.h>
#import "AppDelegate.h"
#import "GeneratedPluginRegistrant.h"

@implementation AppDelegate
- (BOOL)application:(UIApplication*)application didFinishLaunchingWithOptions:(NSDictionary*)launchOptions {
    // 1. 获取FlutterViewController(是应用程序的默认Controller)
    FlutterViewController* controller = (FlutterViewController*)self.window.rootViewController;

    // 2. 获得MethodChannel(方法通道)
    FlutterMethodChannel* batteryChannel = [FlutterMethodChannel
        methodChannelWithName:@"coderwhy.com/battery"
        binaryMessenger:controller.binaryMessenger];

    // 3. 监听方法调用(会调用传入的回调函数)
    __weak typeof(self) weakSelf = self;
    [batteryChannel setMethodCallHandler:^(FlutterMethodCall* call, FlutterResult result) {
        // 3.1. 判断是否是getBatteryInfo的调用
        if ([@"getBatteryInfo" isEqualToString:call.method]) {
            // 1.iOS中获取信息的方式
            int batteryLevel = [weakSelf getBatteryLevel];
            // 2. 如果没有获取到,那么返回给Flutter端一个异常
            if (batteryLevel == -1) {
                result([FlutterError errorWithCode:@"UNAVAILABLE"
                    message:@"Battery info unavailable"
                    details:nil]);
            } else {
                // 3. 通过result将结果回调给Flutter端
                result(@(batteryLevel));
            }
        } else {
            // 3.2. 如果调用的是getBatteryInfo的方法, 那么通过封装的另外一个方法实现回调
            result(FlutterMethodNotImplemented);
        }
    }];
}

[GeneratedPluginRegistrant registerWithRegistry:self];
return [super application:application didFinishLaunchingWithOptions:launchOptions];
}

- (int)getBatteryLevel {
    // 获取信息的方法
    UIDevice* device = UIDevice.currentDevice;
    device.batteryMonitoringEnabled = YES;
    if (device.batteryState == UIDeviceBatteryStateUnknown) {
        return -1;
    } else {
        return (int)(device.batteryLevel * 100);
    }
}

@end

```

## 1.2.5. 编写Android代码

### 1.2.5.1. Kotlin代码实现

实现思路和上面是一致的，只是使用了Kotlin来实现：

- 可以参考注释内容

```
import androidx.annotation.NonNull
import io.flutter.embedding.android.FlutterActivity
import io.flutter.embedding.engine.FlutterEngine
import io.flutter.plugin.common.MethodChannel

import android.content.Context
import android.content.ContextWrapper
import android.content.Intent
import android.content.IntentFilter
import android.os.BatteryManager
import android.os.Build.VERSION
import android.os.Build.VERSION_CODES

class MainActivity: FlutterActivity() {

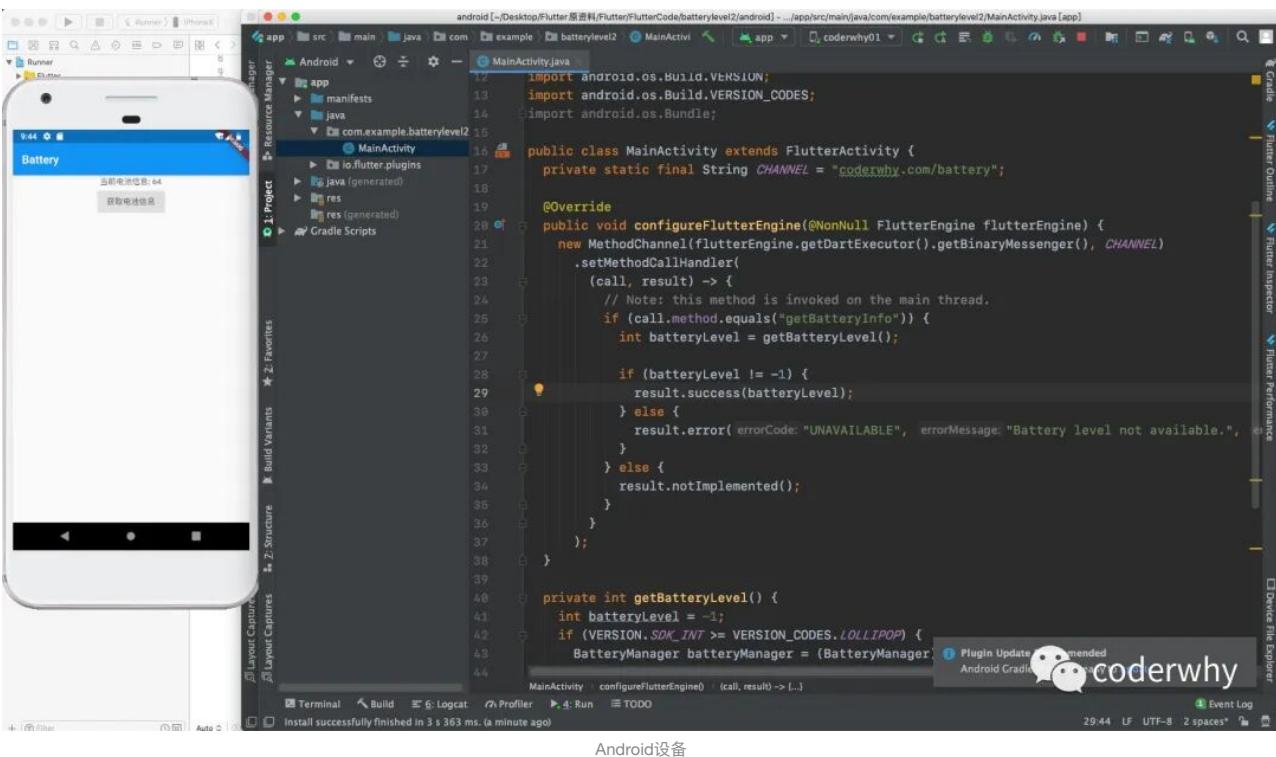
    private val CHANNEL = "coderwhy.com/battery"

    override fun configureFlutterEngine(@NonNull flutterEngine: FlutterEngine) {
        // 1. 创建MethodChannel对象
        val methodChannel = MethodChannel(flutterEngine.dartExecutor.binaryMessenger, CHANNEL)

        // 2. 添加调用方法的回调
        methodChannel.setMethodCallHandler {
            call, result ->
            // Note: this method is invoked on the main thread.
            // 2.1. 如果调用的方法是getBatteryInfo,那么正常执行
            if (call.method == "getBatteryInfo") {
                // 2.1.1. 调用另外一个自定义方法回去电量信息
                val batteryLevel = getBatteryLevel()

                // 2.1.2. 判断是否正常获取到
                if (batteryLevel != -1) {
                    // 获取到返回结果
                    result.success(batteryLevel)
                } else {
                    // 获取不到抛出异常
                    result.error("UNAVAILABLE", "Battery level not available.", null)
                }
            } else {
                // 2.2. 如果调用的方法是getBatteryInfo,那么正常执行
                result.notImplemented()
            }
        }
    }

    private fun getBatteryLevel(): Int {
        val batteryLevel: Int
        if (VERSION.SDK_INT >= VERSION_CODES.LOLLIPOP) {
            val batteryManager = getSystemService(Context.BATTERY_SERVICE) as BatteryManager
            batteryLevel = batteryManager.getIntProperty(BatteryManager.BATTERY_PROPERTY_CAPACITY)
        } else {
            val intent = ContextWrapper(applicationContext).registerReceiver(null, IntentFilter(Intent.ACTION_BATTERY_CHANGED))
            batteryLevel = intent!!.getIntExtra(BatteryManager.EXTRA_LEVEL, -1) * 100 / intent.getIntExtra(BatteryManager.EXTRA_SCALE, -1)
        }
        return batteryLevel
    }
}
```



Android设备

#### 1.2.5.1. Java代码实现

实现思路和上面是一致的，只是使用了Java来实现：

- 可以参考注释内容

```

package com.example.batterylevel2;

import androidx.annotation.NonNull;
import io.flutter.embedding.android.FlutterActivity;
import io.flutter.embedding.engine.FlutterEngine;
import io.flutter.plugins.GeneratedPluginRegistrant;
import io.flutter.plugin.common.MethodChannel;
import android.content.ContextWrapper;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.BatteryManager;
import android.os.Build.VERSION;
import android.os.Build.VERSION_CODES;
import android.os.Bundle;

public class MainActivity extends FlutterActivity {
    private static final String CHANNEL = "coderwhy.com/battery";

    @Override
    public void configureFlutterEngine(@NonNull FlutterEngine flutterEngine) {
        // 1. 创建MethodChannel对象
        MethodChannel methodChannel = new MethodChannel(flutterEngine.getDartExecutor().getBinaryMessenger(), CHANNEL);

        // 2. 添加调用方法的回调
        methodChannel.setMethodCallHandler(
            (call, result) -> {
                // 2.1. 如果调用的方法是getBatteryInfo,那么正常执行
                if (call.method.equals("getBatteryInfo")) {

                    // 2.1.1. 调用另外一个自定义方法回去电量信息
                    int batteryLevel = getBatteryLevel();

                    // 2.1.2. 判断是否正常获取到
                    if (batteryLevel != -1) {
                        // 获取到返回结果
                        result.success(batteryLevel);
                    } else {
                        // 获取不到抛出异常
                        result.error("UNAVAILABLE", "Battery level not available.", null);
                    }
                } else {
                    // 2.2. 如果调用的方法是getBatteryInfo,那么正常执行
                    result.notImplemented();
                }
            });
    }

    private int getBatteryLevel() {
        int batteryLevel = -1;
        if (VERSION.SDK_INT >= VERSION_CODES.LOLLIPOP) {
            BatteryManager batteryManager = (BatteryManager) getSystemService(BATTERY_SERVICE);
            batteryLevel = batteryManager.getIntProperty(BatteryManager.BATTERY_PROPERTY_CAPACITY);
        } else {
            Intent intent = new ContextWrapper(getApplicationContext()).
                registerReceiver(null, new IntentFilter(Intent.ACTION_BATTERY_CHANGED));
            batteryLevel = (intent.getIntExtra(BatteryManager.EXTRA_LEVEL, -1) * 100) /
                intent.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
        }

        return batteryLevel;
    }
}

```

## 二. 嵌入原有项目

首先，我们先明确一点：Flutter设计初衷并不是为了和其它平台进行混合开发，它的目的是为了打造一个完整的跨平台应用程序。

但是，实际开发中，原有项目完全使用Flutter进行重构并不现实，对于原有项目我们更多可能采用混合开发的方式。

### 2.1. 创建Flutter模块

对于需要进行混合开发的原有项目，Flutter可以作为一个库或者模块，继承进现有项目中。

- 模块引入到你的Android或iOS应用中，以使用Flutter渲染一部分的UI，或者共享的Dart代码。
- 在Flutter v1.12中，添加到现有应用的基本场景已经被支持，每个应用在同一时间可以集成一个全屏幕的Flutter实例。

但是，目前一些场景依然是有限制的：

- 运行多个Flutter实例，或在屏幕局部上运行Flutter可能会导致不可以预测的行为；
- 在后台模式使用Flutter的能力还在开发中（目前不支持）；
- 将Flutter库打包到另一个可共享的库或将多个Flutter库打包到同一个应用中，都不支持；
- 添加到应用在Android平台的实现基于 FlutterPlugin 的 API，一些不支持 `FlutterPlugin` 的插件可能会有不可预知的行为。

创建Flutter Module

```
flutter create --template module my_flutter
```

创建完成后，该模块和普通的Flutter项目一致，可以通过Android Studio或VSCode打开、开发、运行；

目录结构如下：

- 和之前项目不同的iOS和Android项目是一个隐藏文件，并且我们通常不会单独打开它们再来运行；
- 它们的作用是将Flutter Module进行编译，之后继承到现有的项目中；

```
my_flutter/
├── .ios/
├── .android/
├── lib/
│   └── main.dart
└── test/
    └── pubspec.yaml
```

## 2.2. 嵌入iOS项目

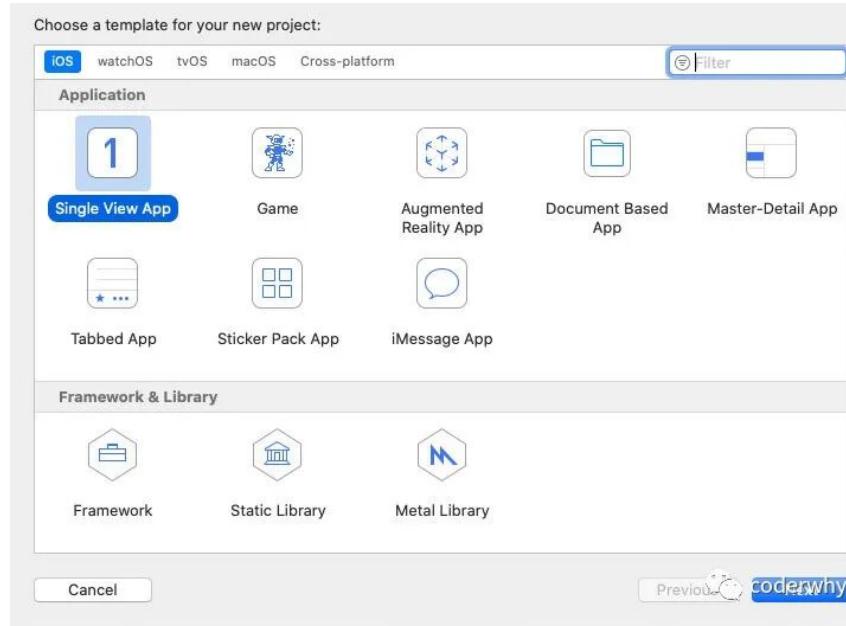
嵌入到现有iOS项目有多种方式：

- 可以使用 CocoaPods 依赖管理和已安装的 Flutter SDK；
- 也可以通过手动编译 Flutter engine、你的 dart 代码和所有 Flutter plugin 成 framework，用 Xcode 手动集成到你的应用中，并更新编译设置；

目前iOS项目几乎都已经使用Cocoapods进行管理，所以推荐使用第一种CocoaPods方式；

我们按照如下的方式，搭建一个需要继承的iOS项目：

1.为了进行测试，我们这里创建一个默认的iOS项目：使用Xcode创建即可



创建一个iOS项目

2.将项目加入CocoaPods进行管理

- 电脑上需要已经安装了CocoaPods

初始化CocoaPods：

```
pod init
```

安装CocoaPods的依赖:

```
pod install
```

编译Podfile文件:

```
# Uncomment the next line to define a global platform for your project
# platform :ios, '9.0'

# 添加模块所在路径
flutter_application_path = '../my_flutter'
load File.join(flutter_application_path, '.ios', 'Flutter', 'podhelper.rb')

target 'ios_my_test' do
  # Comment the next line if you don't want to use dynamic frameworks
  use_frameworks!

  # 安装Flutter模块
  install_all_flutter_pods(flutter_application_path)

  # Pods for ios_my_test

end
```

重新执行安装CocoaPods的依赖:

```
pod install
```

## 2.2.1. Swift代码

为了在既有的iOS应用中展示Flutter页面，需要启动 `Flutter Engine` 和 `FlutterViewController`。

通常建议为我们的应用预热一个 `长时间存活` 的FlutterEngine:

- 我们将在应用启动的 `app delegate` 中创建一个 `FlutterEngine`，并作为属性暴露给外界。

```
import UIKit
import FlutterPluginRegistrant

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {
    // 1. 创建一个FlutterEngine对象
    lazy var flutterEngine = FlutterEngine(name: "my flutter engine")

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool
    {
        // 2. 启动flutterEngine
        flutterEngine.run()
        return true
    }
}
```

在启动的ViewController中，创建一个UIButton，并且点击这个Button时，弹出FlutterViewController

```

import UIKit
import Flutter

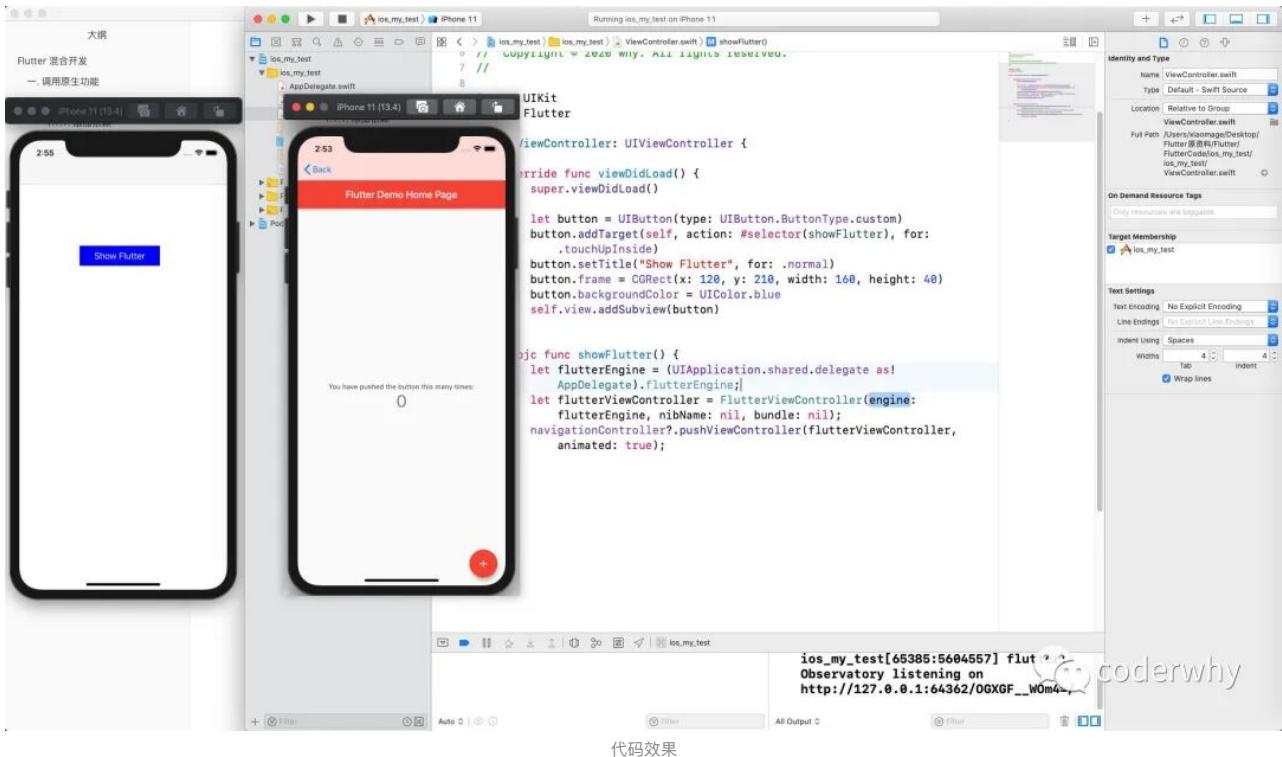
class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        // 1. 创建一个按钮
        let button = UIButton(type: UIButton.ButtonType.custom)
        button.addTarget(self, action: #selector(showFlutter), for: .touchUpInside)
        button.setTitle("Show Flutter", for: .normal)
        button.frame = CGRect(x: 80, y: 210, width: 160, height: 40)
        button.backgroundColor = UIColor.blue
        self.view.addSubview(button)
    }

    @objc func showFlutter() {
        // 2. 创建FlutterViewController对象 (需要先获取flutterEngine)
        let flutterEngine = (UIApplication.shared.delegate as! AppDelegate).flutterEngine
        let flutterViewController = FlutterViewController(engine: flutterEngine, nibName: nil, bundle: nil)
        navigationController?.pushViewController(flutterViewController, animated: true);
    }
}

```



我们也可以省略预先创建的 `FlutterEngine` :

- 不推荐这样做，因为在第一针图像渲染完成之前，可能会出现明显的延迟。

```

func showFlutter() {
    let flutterViewController = FlutterViewController(project: nil, nibName: nil, bundle: nil)
    present(flutterViewController, animated: true, completion: nil)
}

```

## 2.2.2. Objective-C代码

如果上面的代码希望使用Objective-C也是可以实现的：

- 代码的逻辑是完全一致的

AppDelegate.h代码：

```

#import UIKit;
#import Flutter;

@interface AppDelegate : FlutterAppDelegate
@property (nonatomic,strong) FlutterEngine *flutterEngine;
@end

```

AppDelegate.m代码：

```

#import <FlutterPluginRegistrant/GeneratedPluginRegistrant.h> // Used to connect plugins.

#import "AppDelegate.h"

@implementation AppDelegate

- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary<UIApplicationLaunchOptionsKey, id> *)launchOptions {
    self.flutterEngine = [[FlutterEngine alloc] initWithName:@"my flutter engine"];
    [self.flutterEngine run];
    [GeneratedPluginRegistrant registerWithRegistry:self.flutterEngine];
    return [super application:application didFinishLaunchingWithOptions:launchOptions];
}

@end

```

ViewController.m代码：

```

#import Flutter;
#import "AppDelegate.h"
#import "ViewController.h"

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    // Make a button to call the showFlutter function when pressed.
    UIButton *button = [UIButton buttonWithType:UIButtonTypeCustom];
    [button addTarget:self
               action:@selector(showFlutter)
     forControlEvents:UIControlEventTouchUpInside];
    [button setTitle:@"Show Flutter!" forState:UIControlStateNormal];
    button.backgroundColor = UIColor.blueColor;
    button.frame = CGRectMake(80.0, 210.0, 160.0, 40.0);
    [self.view addSubview:button];
}

- (void)showFlutter {
    FlutterEngine *flutterEngine =
    ((AppDelegate *)UIApplication.sharedApplication.delegate).flutterEngine;
    FlutterViewController *flutterViewController =
    [[FlutterViewController alloc] initWithEngine:flutterEngine nibName:nil bundle:nil];
    [self presentViewController:flutterViewController animated:YES completion:nil];
}
@end

```

## 2.3. 嵌入Android项目

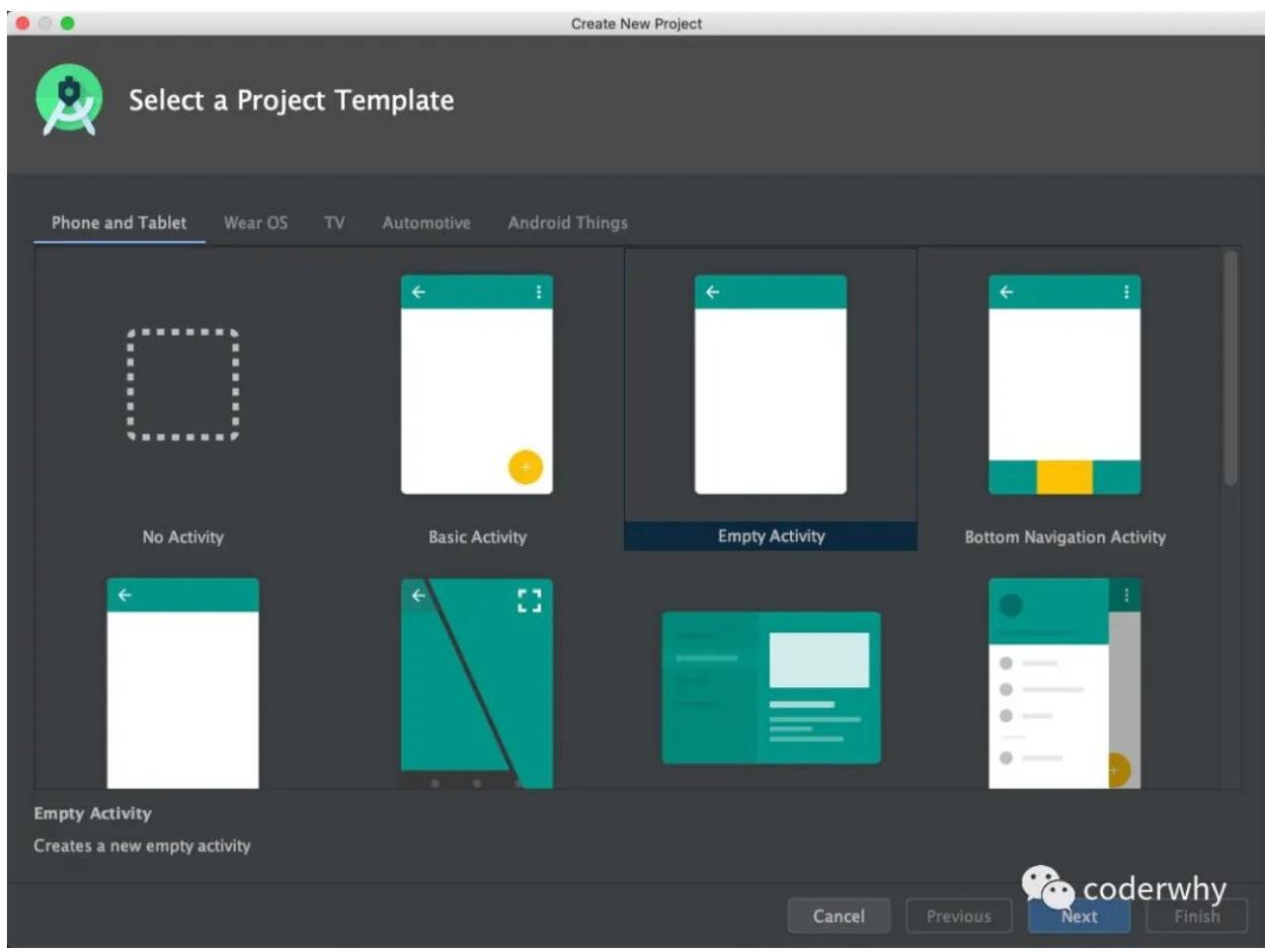
嵌入到现有Android项目有多种方式：

- 编译为AAR文件（Android Archive）
  - 通过Flutter编译为aar，添加相关的依赖
- 依赖模块的源码方式，在gradle进行配置

这里我们采用第二种方式

1. 创建一个Android的测试项目

- 使用Android Studio创建



创建Android项目

## 2.添加相关的依赖

修改Android项目中的settings.gradle文件:

```
// Include the host app project.
include ':app' // assumed existing content
setBinding(new Binding([gradle: this])) // new
evaluate(new File( // new
    settingsDir.parentFile, // new
    'my_flutter/.android/include_flutter.groovy' // new
))
```

另外，我们需要在Android项目工程的build.gradle中添加依赖:

```
dependencies {
    implementation project(':flutter')
}
```

编译代码，可能会出现如下错误：

- 这是因为从Java8开始才支持接口方法
- Flutter Android引擎使用了该Java8的新特性

**Default interface methods are only supported starting with Android N (--minSdkVersion 24)  
 androidx.lifecycle.DefaultLifecycleObserver.onCreate(androidx.lifecycle.LifecycleOwner)**

image-20200413161942859

解决办法：通过设置Android项目工程的build.gradle配置使用Java8编译：

```
compileOptions {
    sourceCompatibility 1.8
    targetCompatibility 1.8
}
```

接下来，我们这里尝试添加一个Flutter的screen到Android应用程序中

Flutter提供了一个FlutterActivity来展示Flutter界面在Android应用程序中，我们需要先对FlutterActivity进行注册：

- 在AndroidManifest.xml中进行注册

```
<activity
    android:name="io.flutter.embedding.android.FlutterActivity"
    android:theme="@style/AppTheme"
    android:configChanges="orientation|keyboardHidden|keyboard|screenSize|locale|layoutDirection|fontScale|screenLayout|density|uiMode"
    android:hardwareAccelerated="true"
    android:windowSoftInputMode="adjustResize"
/>
```

### 2.3.1. Java代码

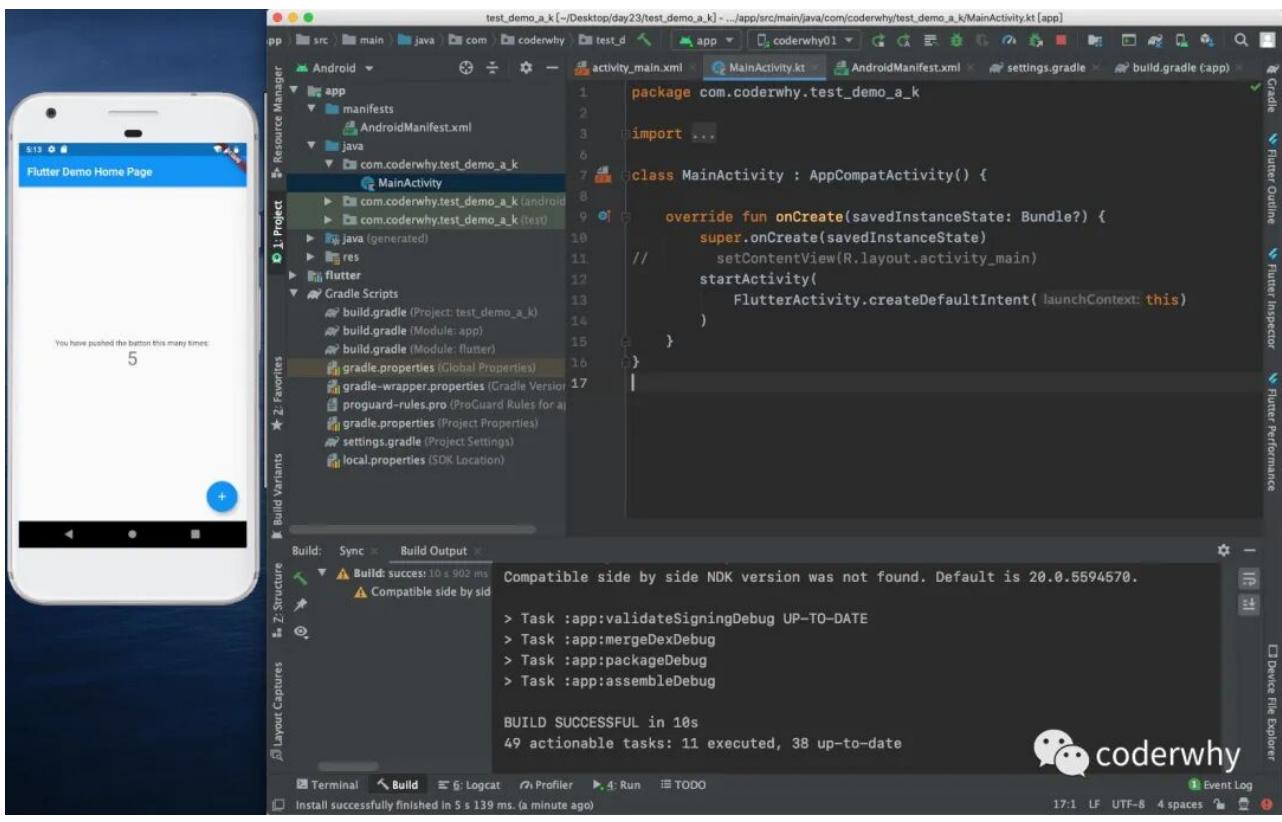
```
package com.coderwhy.testandroid;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import io.flutter.embedding.android.FlutterActivity;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // setContentView(R.layout.activity_main);
        startActivity(
            FlutterActivity.createDefaultIntent(this)
        );
    }
}
```

也可以在创建时，传入默认的路由：

```
package com.coderwhy.testandroid;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import io.flutter.embedding.android.FlutterActivity;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // setContentView(R.layout.activity_main);
        startActivity(
            FlutterActivity
                .withNewEngine()
                .initialRoute("/my_route")
                .build(currentActivity)
        );
    }
}
```



将Flutter页面嵌入到Android项目

### 2.3.2. Kotlin代码

```
package com.coderwhy.test_demo_a_k

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import io.flutter.embedding.android.FlutterActivity

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // setContentView(R.layout.activity_main)
        startActivityForResult(
            FlutterActivity.createDefaultIntent(this)
        )
    }
}
```

也可以在创建时指定路由：

```
package com.coderwhy.test_demo_a_k

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import io.flutter.embedding.android.FlutterActivity

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // setContentView(R.layout.activity_main)
        startActivityForResult(
            FlutterActivity
                .withNewEngine()
                .initialRoute("/my_route")
                .build(this)
        );
    }
}
```

一旦将Flutter模块继承到你的项目中，并且使用Flutter平台的API运行Flutter引擎或UI，那么就可以先普通的Android或者iOS一样来构建自己的Android或者iOS项目了。

但是Flutter的一个非常大的优势是其快速开发，也就是hot reload。

那么对应Flutter模块，我们如何使用hot reload加速我们的调试速度呢？

- 可以使用flutter attach

```
# --app-id是指定哪一个应用程序  
# -d是指定连接哪一个设备  
flutter attach --app-id com.coderwhy.ios-my-test -d 3D7A877C-B0DD-4871-8D6E-0C5263B986CD
```

