

29 | 混合开发，该用何种方案管理导航栈？

2019-09-03 陈航

Flutter核心技术与实战

[进入课程 >](#)



讲述：陈航

时长 11:11 大小 10.26M



你好，我是陈航。

为了把 Flutter 引入到原生工程，我们需要把 Flutter 工程改造为原生工程的一个组件依赖，并以组件化的方式管理不同平台的 Flutter 构建产物，即 Android 平台使用 aar、iOS 平台使用 pod 进行依赖管理。这样，我们就可以在 Android 工程中通过 FlutterView，iOS 工程中通过 FlutterViewController，为 Flutter 搭建应用入口，实现 Flutter 与原生的混合开发方式。

我在[第 26 篇](#)文章中提到，FlutterView 与 FlutterViewController 是初始化 Flutter 的地方，也是应用的入口。可以看到，以混合开发方式接入 Flutter，与开发一个纯 Flutter 应用在运行机制上并无任何区别，只需要原生工程为它提供一个画板容器（Android 为

FlutterView, iOS 为 FlutterViewController) , Flutter 就可以自己管理页面导航栈, 从而实现多个复杂页面的渲染和切换。

关于纯 Flutter 应用的页面路由与导航, 我已经在[第 21 篇文章](#)中与你介绍过了。今天这篇文章, 我会为你讲述在混合开发中, 应该如何管理混合导航栈。

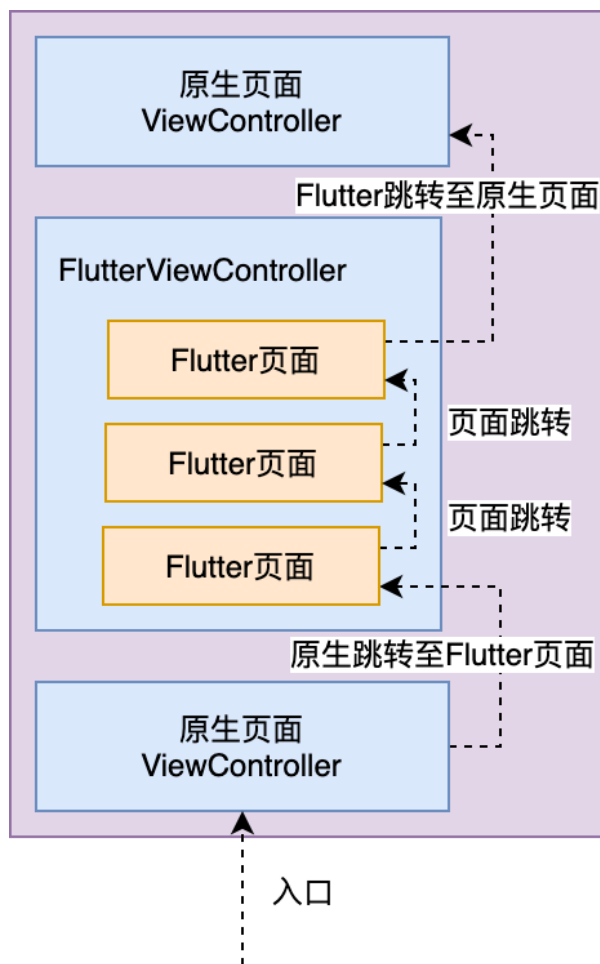
对于混合开发的应用而言, 通常我们只会将应用的部分模块修改成 Flutter 开发, 其他模块继续保留原生开发, 因此应用内除了 Flutter 的页面之外, 还会有原生 Android、iOS 的页面。在这种情况下, Flutter 页面有可能会需要跳转到原生页面, 而原生页面也可能会需要跳转到 Flutter 页面。这就涉及到了一个新的问题: 如何统一管理原生页面和 Flutter 页面跳转交互的混合导航栈。

接下来, 我们就从这个问题入手, 开始今天的学习吧。

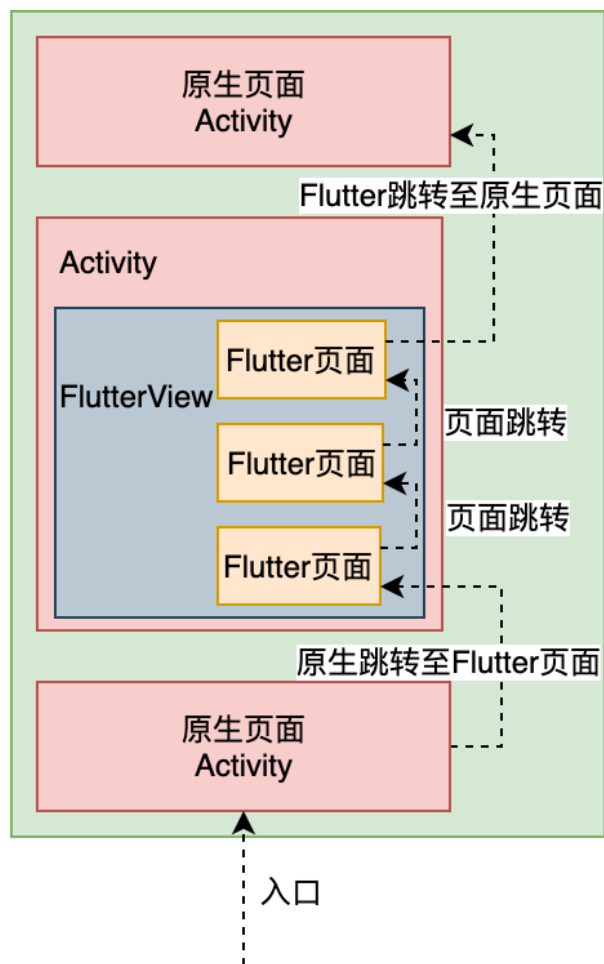
混合导航栈

混合导航栈, 指的是原生页面和 Flutter 页面相互掺杂, 存在于用户视角的页面导航栈视图中。

以下图为例, Flutter 与原生 Android、iOS 各自实现了一套互不相同的页面映射机制, 即原生采用单容器单页面 (一个 ViewController/Activity 对应一个原生页面)、Flutter 采用单容器多页面 (一个 ViewController/Activity 对应多个 Flutter 页面) 的机制。Flutter 在原生的导航栈之上又自建了一套 Flutter 导航栈, 这使得 Flutter 页面与原生页面之间涉及页面切换时, 我们需要处理跨引擎的页面切换。



iOS混合导航栈



Android混合导航栈

图 1 混合导航栈示意图

接下来，我们就分别看看从原生页面跳转至 Flutter 页面，以及从 Flutter 页面跳转至原生页面，应该如何处理吧。


从原生页面跳转至 Flutter 页面

从原生页面跳转至 Flutter 页面，实现起来比较简单。

因为 Flutter 本身依托于原生提供的容器（iOS 为 FlutterViewController，Android 为 Activity 中的 FlutterView），所以我们通过初始化 Flutter 容器，为其设置初始路由页面之后，就可以以原生的方式跳转至 Flutter 页面了。

如下代码所示。对于 iOS，我们初始化一个 FlutterViewController 的实例，为其设置初始化页面路由后，将其加入原生的视图导航栈中完成跳转。

对于 Android 而言，则需要多加一步。因为 Flutter 页面的入口并不是原生视图导航栈的最小单位 Activity，而是一个 View（即 FlutterView），所以我们还需要把这个 View 包装到 Activity 的 contentView 中。在 Activity 内部设置页面初始化路由之后，在外部就可以采用打开一个普通的原生视图的方式，打开 Flutter 页面了。

 复制代码

```
1 //iOS 跳转至 Flutter 页面
2 FlutterViewController *vc = [[FlutterViewController alloc] init];
3 [vc setInitialRoute:@"defaultPage"];// 设置 Flutter 初始化路由页面
4 [self.navigationController pushViewController:vc animated:YES];// 完成页面跳转
5
6
7 //Android 跳转至 Flutter 页面
8
9 // 创建一个作为 Flutter 页面容器的 Activity
10 public class FlutterHomeActivity extends AppCompatActivity {
11     protected void onCreate(Bundle savedInstanceState) {
12         super.onCreate(savedInstanceState);
13         // 设置 Flutter 初始化路由页面
14         View flutterView = Flutter.createView(this, getLifecycle(), "defaultRoute"); // 传入
15         setContentView(flutterView);// 用 FlutterView 替代 Activity 的 ContentView
16     }
17 }
18 // 用 FlutterPageActivity 完成页面跳转
19 Intent intent = new Intent(MainActivity.this, FlutterHomeActivity.class);
20 startActivity(intent);
```

从 Flutter 页面跳转至原生页面

从 Flutter 页面跳转至原生页面，则会相对麻烦些，我们需要考虑以下两种场景：

从 Flutter 页面打开新的原生页面；

从 Flutter 页面回退到旧的原生页面。

首先，我们来看看 Flutter 如何打开原生页面。

Flutter 并没有提供对原生页面操作的方法，所以不可以直接调用。我们需要通过方法通道（你可以再回顾下[第 26 篇](#)文章的相关内容），在 Flutter 和原生两端各自初始化时，提供 Flutter 操作原生页面的方法，并注册方法通道，在原生端收到 Flutter 的方法调用时，打开新的原生页面。

接下来，我们再看看如何从 Flutter 页面回退到原生页面。

因为 Flutter 容器本身属于原生导航栈的一部分，所以当 Flutter 容器内的根页面（即初始化路由页面）需要返回时，我们需要关闭 Flutter 容器，从而实现 Flutter 根页面的关闭。同样，Flutter 并没有提供操作 Flutter 容器的方法，因此我们依然需要通过方法通道，在原生代码宿主为 Flutter 提供操作 Flutter 容器的方法，在页面返回时，关闭 Flutter 页面。

Flutter 跳转至原生页面的两种场景，如下图所示：

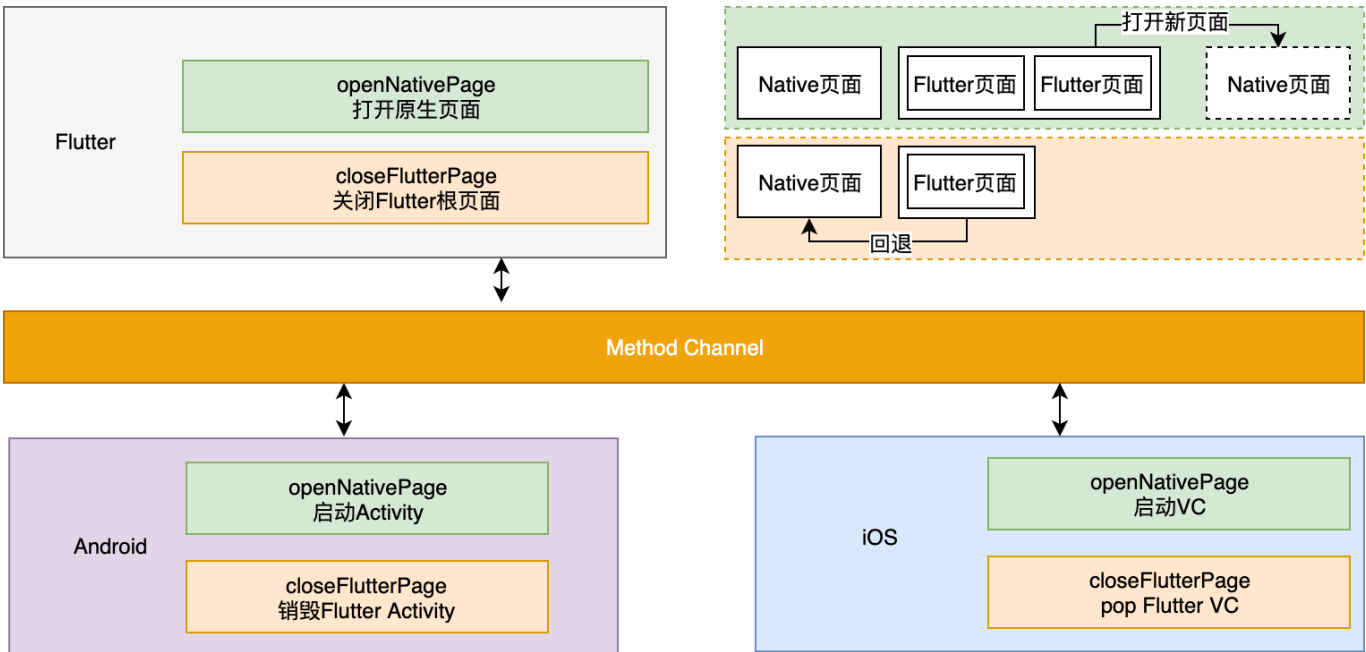


图 2 Flutter 页面跳转至原生页面示意图

接下来，我们一起看看这两个需要通过方法通道实现的方法，即打开原生页面 **openNativePage**，与关闭 Flutter 页面 **closeFlutterPage**，在 Android 和 iOS 平台上分别如何实现。

注册方法通道最合适的地方，是 Flutter 应用的入口，即在 FlutterViewController (iOS 端) 和 Activity 中的 FlutterView (Android 端) 这两个容器内部初始化 Flutter 页面前。为了将 Flutter 相关的行为封装到容器内部，我们需要分别继承 FlutterViewController 和 Activity，在其 **viewDidLoad** 和 **onCreate** 初始化容器时，注册 **openNativePage** 和 **closeFlutterPage** 这两个方法。

iOS 端的实现代码如下所示：

```

1 @interface FlutterHomeViewController : FlutterViewController
2 @end
3
4 @implementation FlutterHomeViewController
5 - (void)viewDidLoad {
6     [super viewDidLoad];
7     // 声明方法通道
8     FlutterMethodChannel* channel = [FlutterMethodChannel methodChannelWithName:@"sample"
9     // 注册方法回调
10    [channel setMethodCallHandler:^(FlutterMethodCall* call, FlutterResult result) {
11        // 如果方法名为打开新页面
12        if([call.method isEqualToString:@"openNativePage"]) {
13            // 初始化原生页面并打开
14            SomeOtherNativeViewController *vc = [[SomeOtherNativeViewController alloc] :
15            [self.navigationController pushViewController:vc animated:YES];
16            result(@0);
17        }
18        // 如果方法名为关闭 Flutter 页面
19        else if([call.method isEqualToString:@"closeFlutterPage"]) {
20            // 关闭自身 (FlutterHomeViewController)
21            [self.navigationController popViewControllerAnimated:YES];
22            result(@0);
23        }
24        else {
25            result(FlutterMethodNotImplemented); // 其他方法未实现
26        }
27    }];
28 }
29 @end

```

Android 端的实现代码如下所示:

```

1 // 继承 AppCompatActivity 来作为 Flutter 的容器
2 public class FlutterHomeActivity extends AppCompatActivity {
3     @Override
4     protected void onCreate(Bundle savedInstanceState) {
5         super.onCreate(savedInstanceState);
6         // 初始化 Flutter 容器
7         FlutterView flutterView = Flutter.createView(this, getLifecycle(), "defaultPage"
8         // 注册方法通道
9         new MethodChannel(flutterView, "samples.chenhang/navigation").setMethodCallHandl
10        new MethodCallHandler() {
11            @Override
12            public void onMethodCall(MethodCall call, Result result) {
13                // 如果方法名为打开新页面

```



```

14         if(call.method.equals("openNativePage")) {
15             // 新建 Intent, 打开原生页面
16             Intent intent = new Intent(FlutterHomeActivity.this, SomeNative
17                 startActivity(intent);
18                 result.success(0);
19         }
20         // 如果方法名为关闭 Flutter 页面
21         else if(call.method.equals("closeFlutterPage")) {
22             // 销毁自身 (Flutter 容器)
23             finish();
24             result.success(0);
25         }
26         else {
27             // 方法未实现
28             result.notImplemented();
29         }
30     }
31 });
32 // 将 flutterView 替换成 Activity 的 contentView
33 setContentView(flutterView);
34 }
35 }


```

经过上面的方法注册，我们就可以在 Flutter 层分别通过 `openNativePage` 和 `closeFlutterPage` 方法，来实现 Flutter 页面与原生页面之间的切换了。

在下面的例子中，Flutter 容器的根视图 `DefaultPage` 包含有两个按钮：

点击左上角的按钮后，可以通过 `closeFlutterPage` 返回原生页面；

点击中间的按钮后，会打开一个新的 Flutter 页面 `PageA`。`PageA` 中也有一个按钮，点击这个按钮之后会调用 `openNativePage` 来打开一个新的原生页面。

 复制代码

```

1 void main() => runApp(_widgetForRoute(window.defaultRouteName));
2 // 获取方法通道
3 const platform = MethodChannel('samples.chenhang/navigation');
4
5 // 根据路由标识符返回应用入口视图
6 Widget _widgetForRoute(String route) {
7     switch (route) {
8         default:// 返回默认视图
9         return MaterialApp(home:DefaultPage());
10    }
11 }

```

```

12
13 class PageA extends StatelessWidget {
14     ...
15     @override
16     Widget build(BuildContext context) {
17         return Scaffold(
18             body: RaisedButton(
19                 child: Text("Go PageB"),
20                 onPressed: ()=>platform.invokeMethod('openNativePage')// 打开原生页面
21             ));
22     }
23 }
24
25 class DefaultPage extends StatelessWidget {
26     ...
27     @override
28     Widget build(BuildContext context) {
29         return Scaffold(
30             appBar: AppBar(
31                 title: Text("DefaultPage Page"),
32                 leading: IconButton(icon:Icon(Icons.arrow_back), onPressed:() => platform.i
33             )),
34             body: RaisedButton(
35                 child: Text("Go PageA"),
36                 onPressed: ()=>Navigator.push(context, MaterialPageRoute(builder: (coi
37             ));
38     }
39 }

```



整个混合导航栈示例的代码流程，如下图所示。通过这张图，你就可以把这个示例的整个代码流程串起来了。

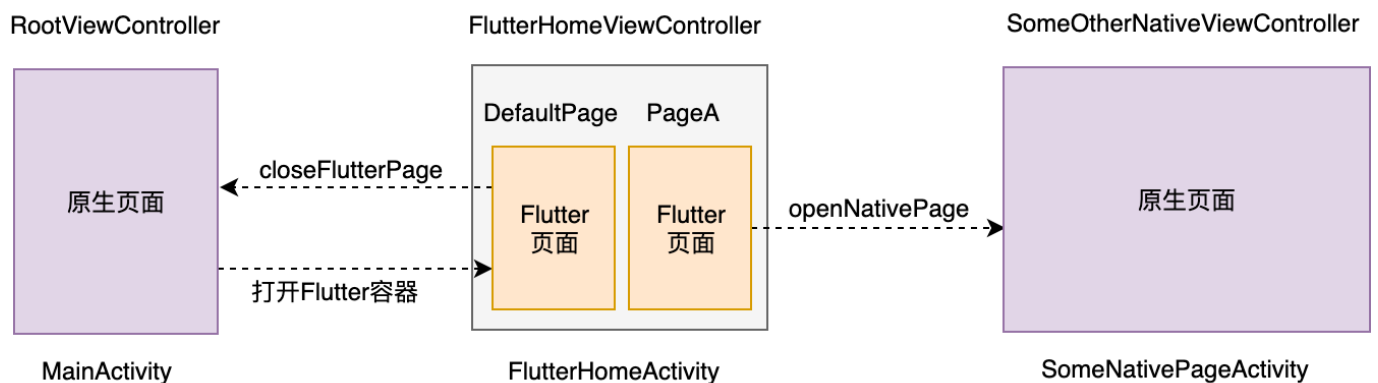


图 3 混合导航栈示例

在我们的混合应用中，RootViewController 与 MainActivity 分别是 iOS 和 Android 应用的原生页面入口，可以初始化为 Flutter 容器的 FlutterHomeViewController (iOS 端) 与 FlutterHomeActivity (Android 端)。

在为其设置初始路由页面 DefaultPage 之后，就可以以原生的方式跳转至 Flutter 页面。但是，Flutter 并未提供接口，来支持从 Flutter 的 DefaultPage 页面返回到原生页面，因此我们需要利用方法通道来注册关闭 Flutter 容器的方法，即 closeFlutterPage，让 Flutter 容器接收到这个方法调用时关闭自身。

在 Flutter 容器内部，我们可以使用 Flutter 内部的页面路由机制，通过 Navigator.push 方法，完成从 DefaultPage 到 PageA 的页面跳转；而当我们想从 Flutter 的 PageA 页面跳转到原生页面时，因为涉及到跨引擎的页面路由，所以我们仍然需要利用方法通道来注册打开原生页面的方法，即 openNativePage，让 Flutter 容器接收到这个方法调用时，在原生代码宿主完成原生页面 SomeOtherNativeViewController (iOS 端) 与 SomeNativePageActivity (Android 端) 的初始化，并最终完成页面跳转。

总结

好了，今天的分享就到这里。我们一起总结下今天的主要内容吧。

对于原生 Android、iOS 工程混编 Flutter 开发，由于应用中会同时存在 Android、iOS 和 Flutter 页面，所以我们需要妥善处理跨渲染引擎的页面跳转，解决原生页面如何切换 Flutter 页面，以及 Flutter 页面如何切换到原生页面的问题。

在原生页面切换到 Flutter 页面时，我们通常会将 Flutter 容器封装成一个独立的 ViewController (iOS 端) 或 Activity (Android 端)，在为其设置好 Flutter 容器的页面初始化路由（即根视图）后，原生的代码就可以按照打开一个普通的原生页面的方式，来打开 Flutter 页面了。

而如果我们想在 Flutter 页面跳转到原生页面，则需要同时处理好打开新的原生页面，以及关闭自身回退到老的原生页面两种场景。在这两种场景下，我们都需要利用方法通道来注册相应的处理方法，从而在原生代码宿主实现新页面的打开和 Flutter 容器的关闭。

需要注意的是，与纯 Flutter 应用不同，原生应用混编 Flutter 由于涉及到原生页面与 Flutter 页面之间切换，因此导航栈内可能会出现多个 Flutter 容器的情况，即多个 Flutter 实例。

Flutter 实例的初始化成本非常高昂，每启动一个 Flutter 实例，就会创建一套新的渲染机制，即 Flutter Engine，以及底层的 Isolate。而这些实例之间的内存是不互相共享的，会带来较大的系统资源消耗。

因此我们在实际业务开发中，应该尽量用 Flutter 去开发闭环的业务模块，原生只需要能够跳转到 Flutter 模块，剩下的业务都应该在 Flutter 内部完成，而**尽量避免 Flutter 页面又跳回到原生页面，原生页面又启动新的 Flutter 实例的情况。**

为了解决混编工程中 Flutter 多实例的问题，业界有两种解决方案：

以今日头条为代表的[修改 Flutter Engine 源码](#)，使多 FlutterView 实例对应的多 Flutter Engine 能够在底层共享 Isolate；

以闲鱼为代表的[共享 FlutterView](#)，即由原生层驱动 Flutter 层渲染内容的方案。

坦白说，这两种方案各有不足：

前者涉及到修改 Flutter 源码，不仅开发维护成本高，而且增加了线程模型和内存回收出现异常的概率，稳定性不可控。

后者涉及到跨渲染引擎的 hack，包括 Flutter 页面的新建、缓存和内存回收等机制，因此在一些低端机或是处理页面切换动画时，容易出现渲染 Bug。

除此之外，这两种方式均与 Flutter 的内部实现绑定较紧，因此在处理 Flutter SDK 版本升级时往往需要耗费较大的适配成本。

综合来说，目前这两种解决方案都不够完美。所以，在 Flutter 官方支持多实例单引擎之前，我们还是尽量在产品模块层面，保证应用内不要出现多个 Flutter 容器实例吧。

我把今天分享所涉及到的知识点打包到了 GitHub ([flutter module page](#)、[android demo](#)、[iOS demo](#)) 中，你可以下载下来，反复运行几次，加深理解与记忆。

思考题

最后，我给你留两道思考题吧。

1. 请在 openNativePage 方法的基础上，增加页面 id 的功能，可以支持在 Flutter 页面打开任意的原生页面。

2. 混编工程中会出现两种页面过渡动画：原生页面之间的切换动画、Flutter 页面之间的切换动画。请你思考下，如何能够确保这两种页面过渡动画在应用整体的效果是一致的。

欢迎你在评论区给我留言分享你的观点，我会在下一篇文章中等待你！感谢你的收听，也欢迎你把这篇文章分享给更多的朋友一起阅读。



Flutter 核心技术与实战

来自 Google 的高性能跨平台开发框架

陈航

美团点评高级技术专家



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 28 | 如何在原生应用中混编Flutter工程？

下一篇 30 | 为什么需要做状态管理，怎么做？

精选留言 (2)

写留言

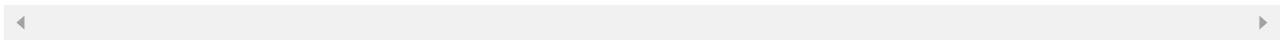


矮个子先生

2019-09-04

老师,不知您是否有出现,push FlutterViewController之后,然后closeFlutterPage,但是内存依然居高不下.

作者回复: FlutterEngine内部实现存在循环引用的情况, 所以会有内存泄漏的问题。不过一般的混合应用只会创建一个FlutterViewController, 或者纯FlutterApp全部都是FlutterViewController, 不释放其实也没什么问题。如果真的需要关闭flutter容器, 可以把FlutterViewController缓存起来(作为单例使用)。



许童童

2019-09-03

跟着老师一起精进, 感谢老师的分享。

展开 ▾

