Flutter混合工程的自动化

拉维 iOS小生活 2022-03-02 17:56

收录于话题
#Flutter
58个 >

在《Flutter与原生工程的混合开发》中，我介绍了Flutter工程与Native工程的混合开发，今天我们来聊一聊混合工程的自动化。

实际上，这里的混合工程指的就是原生工程混入Flutter Module的形式；Flutter工程里面去调用原生功能是很简单的，今天不做讨论。

在《Flutter与原生工程的混合开发》中，我介绍了如何在一个原生工程中去嵌入Flutter页面。如果你是单人开发的话，没有啥问题，按照我这篇文章去做妥妥的；但是如果是在一个多人团队中，就有问题了。

大家想想看，如果你所在的团队很大，有专门的iOS团队、Android团队、Flutter团队，大家各司其职、互不干扰。现在有一个原生与Flutter的混合工程，Flutter开发工程师开发Flutter相关的内容，iOS开发工程师开发iOS相关的内容，在Flutter开发工程师的电脑上是有Flutter开发环境的（Flutter引擎啥的），这没有任何问题，但是这是一个混合开发的工程，如果iOS开发工程师要想执行该混合工程，是不是也需要配置Flutter开发环境呢？答案是一定的，不然的话这个混合工程就运行不起来。而且**原生开发工程师的电脑上不但要保证拥有Flutter环境，而且还要保证Flutter的版本号与Flutter工程师的电脑上的Flutter版本号一致，而且还要保证FlutterSDK的路径是一致的**。这就很坑了，我一个原生开发工程师又不熟悉Flutter，我还要吭哧吭哧去配一下flutter环境，想想都头疼。按道理我一个iOS开发工程师只需要关心我iOS的相关内容就可以了，其实我不需要在我的电脑上配置Flutter环境，这个时候就需要混合工程自动化相关的内容了。

接下来就介绍下如何去构建混合工程。

一、Framework混合工程

先来介绍下将Flutter-Module工程打包成Framework的方式，需要注意的是，下面介绍的这种方式只有在Flutter1.12版本之后才能使用，在Flutter早期阶段，官方是没有提供这种便捷的一次性将Flutter-Module打包成Framework的方式的。在Flutter1.12版本之前，Flutter工程师需要将Flutter-Module工程中生成的App.framework和Flutter.framework这两个东西手动抽离出来，然后再给到原生工程师，这就会很麻烦。

好，废话少说，直接开干。

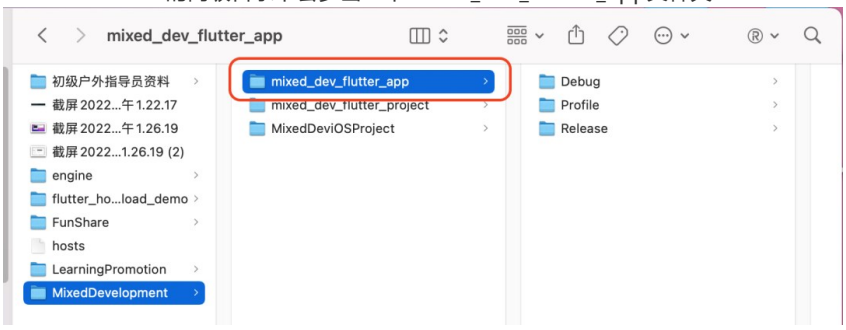
首先来到Flutter-Module所在文件夹路径下：

```
→  mixed_dev_flutter_project pwd
/Users/liwei/Desktop/MixedDevelopment/mixed_dev_flutter_project
→  mixed_dev_flutter_project
```

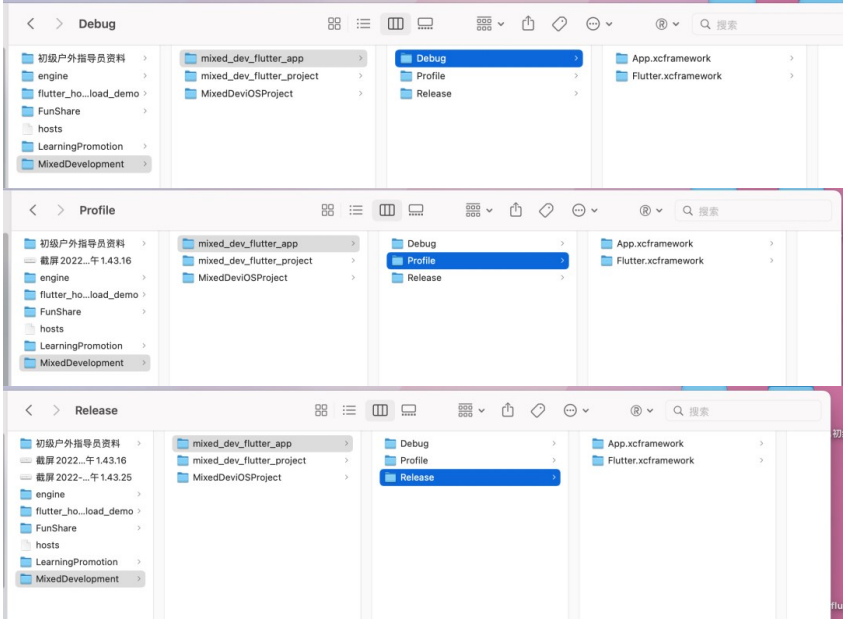
然后执行如下指令，将Flutter-Module打包成Framework：

```
flutter build ios-framework --output=../mixed_dev_flutter_app
```

这里的`../mixed_dev_flutter_app`就是输出的Framework的路径。打包完成之后，在Flutter-Module的同级目录下会多出一个mixed_dev_flutter_app文件夹：



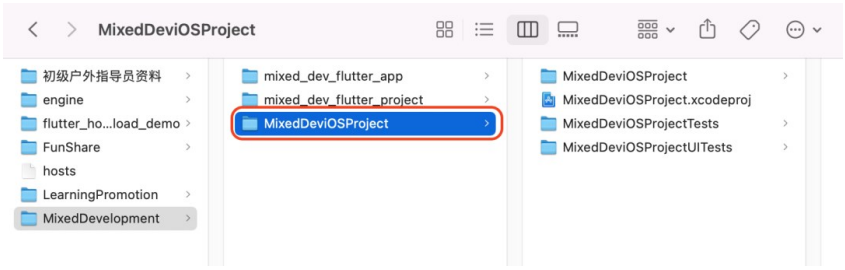
可以看到，最终在`mixed_dev_flutter_app`文件夹下面生成了Debug、Profile和Release三个版本的产物，每个版本的产物里面都有一个App.xcframework和一个Flutter.xcframework，如下：



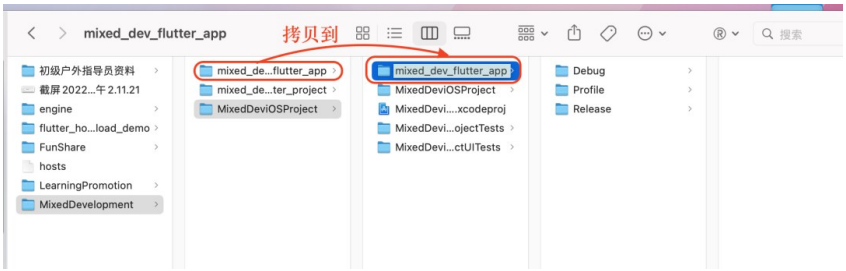
在Debug、Profile和Release这三个版本的产物中，只有Debug版本可以运行在模拟器中，Profile和Release只支持真机不支持模拟器。

在每个版本的产物里面都有一个App.xcframework和一个Flutter.xcframework，这俩东西是由Flutter工程师提供的，Flutter工程师将功能开发、调试、内测完毕后，将Flutter-Module打包成App.xcframework和Flutter.xcframework，交给原生工程师。

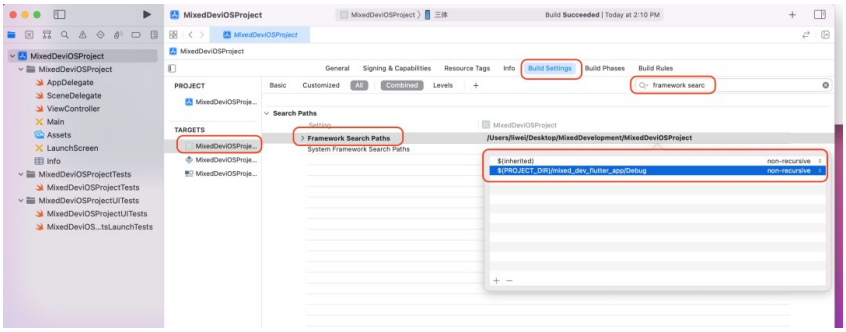
接下来先来创建一个原生工程，如下图所示：



原生工程师在拿到Flutter工程师发送过来的`mixed_dev_flutter_app`文件夹之后，将其拷贝到原生工程根目录下面：



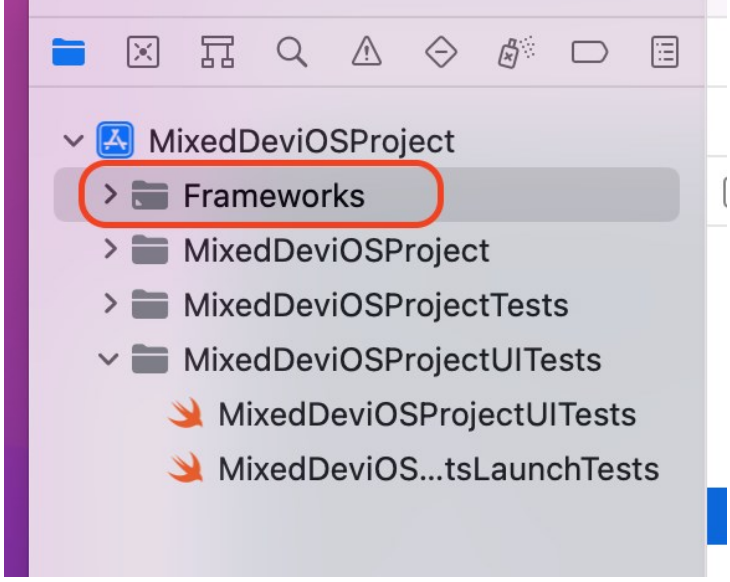
然后来到Xcode工程的TARGETS -> Build Settings，搜索“Framework Search Paths”，然后进行如下配置：



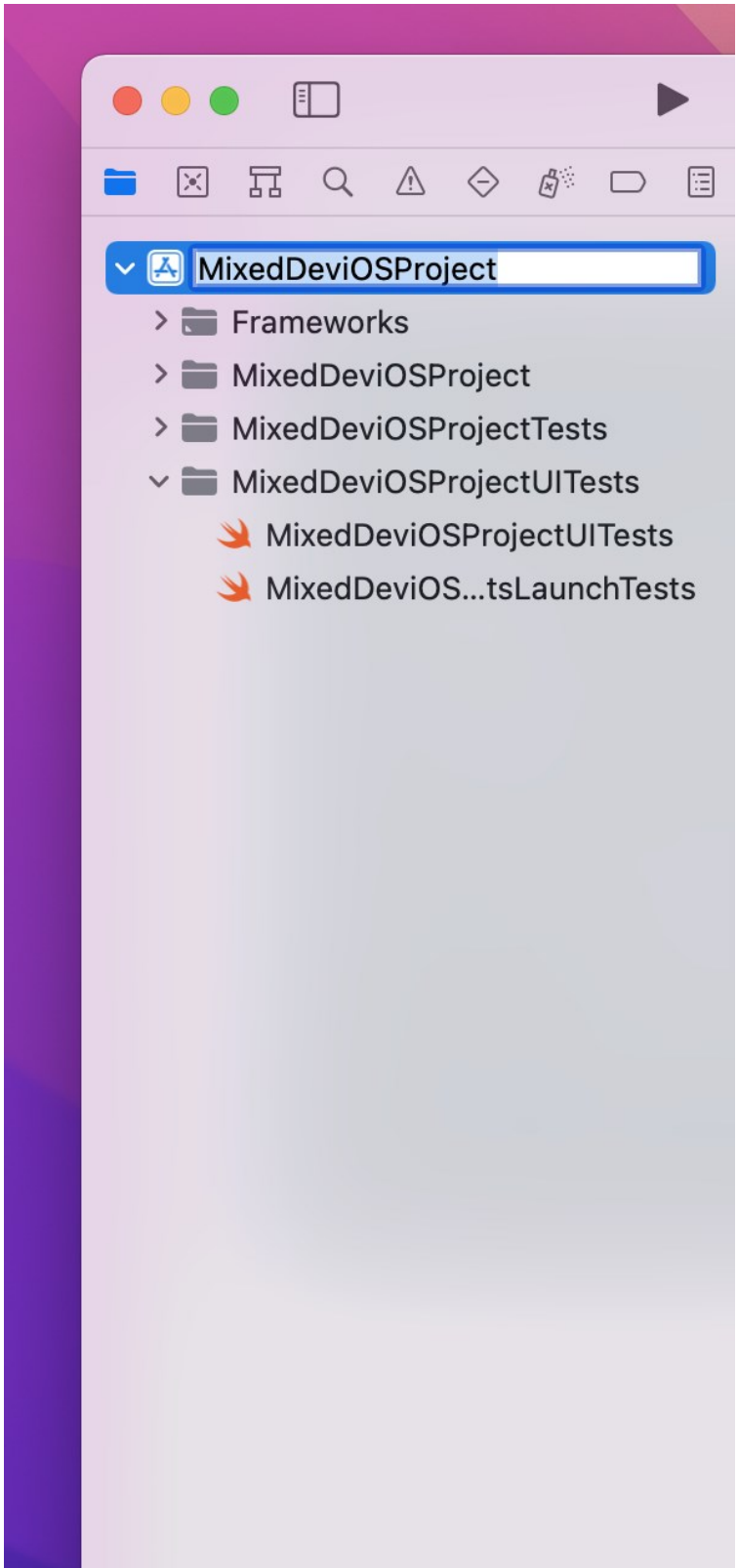
其中，$(PROJECT_DIR)指的是原生工程的路径，所以$(PROJECT_DIR)/mixed_dev_flutter_app/Debug指向的就是Flutter-Module打包后的Debug模式的产物。

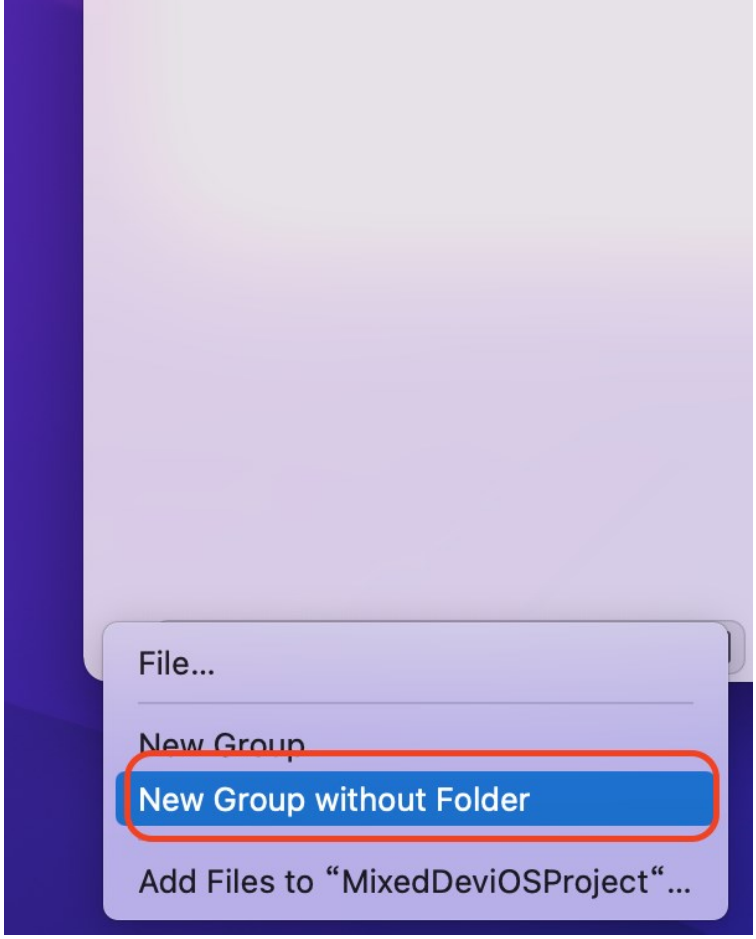
到这里，Framework就配置好了，接下来就将Framework添加到我们的工程中。

首先在工程的根路径下找到Frameworks文件夹，如下：



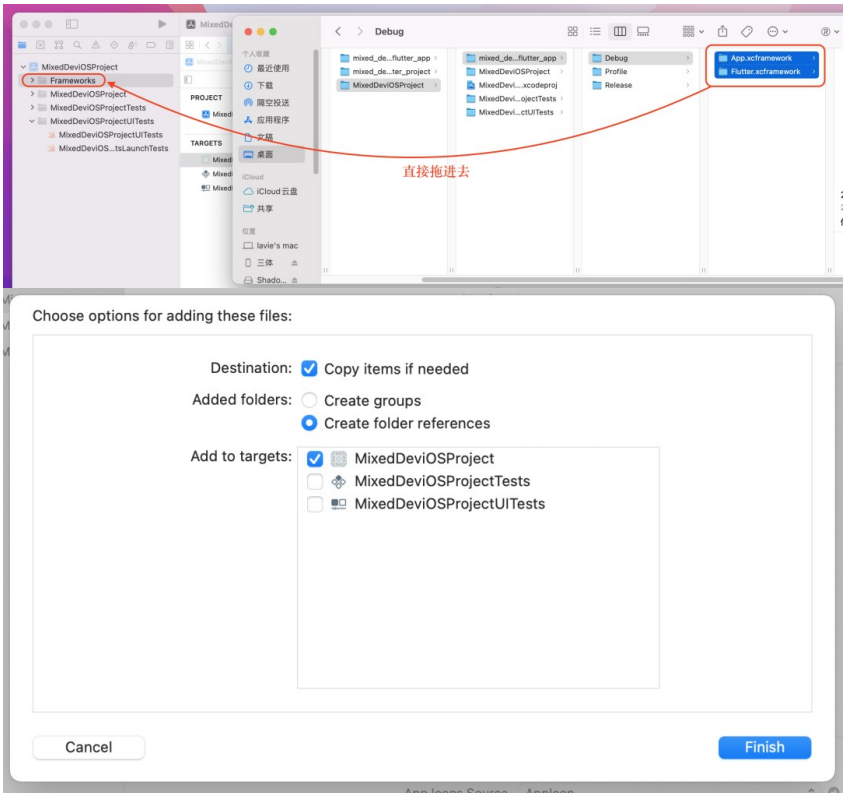
如果你的工程中没有该文件夹，那么就自己手动创建一个，如下：



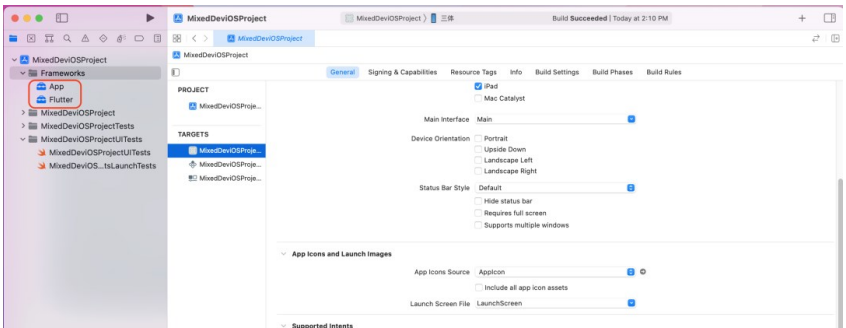


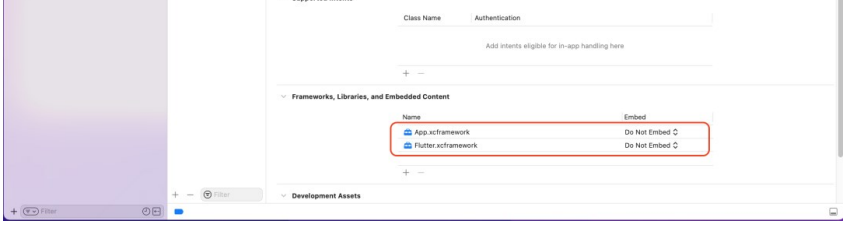
需要注意的是，我们要创建虚拟文件夹，不需要创建物理文件夹。

接下来，我直接将刚才拷贝进工程的Debug模式下的俩产物拖进Frameworks虚拟文件夹，如下：

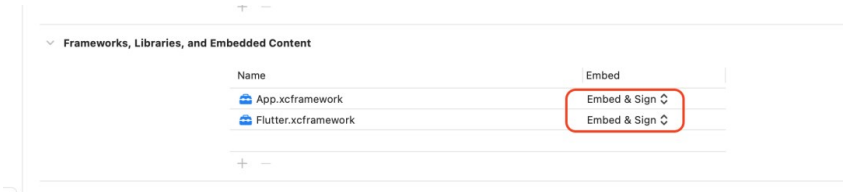


点击Finish之后，这俩Framework就拖进来了，如下：





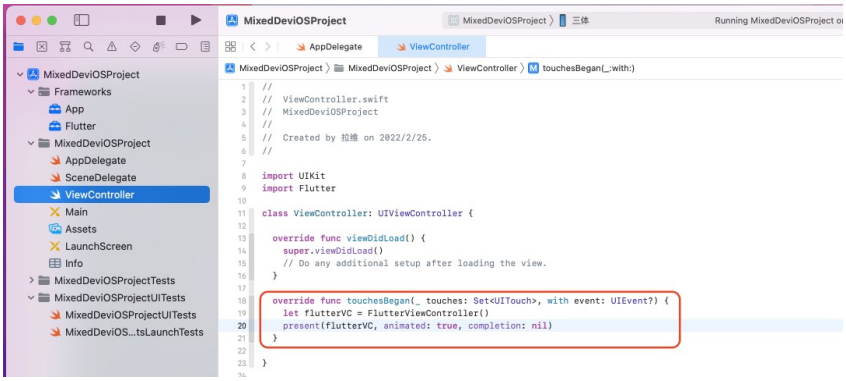
然后我们设置一下Embed：



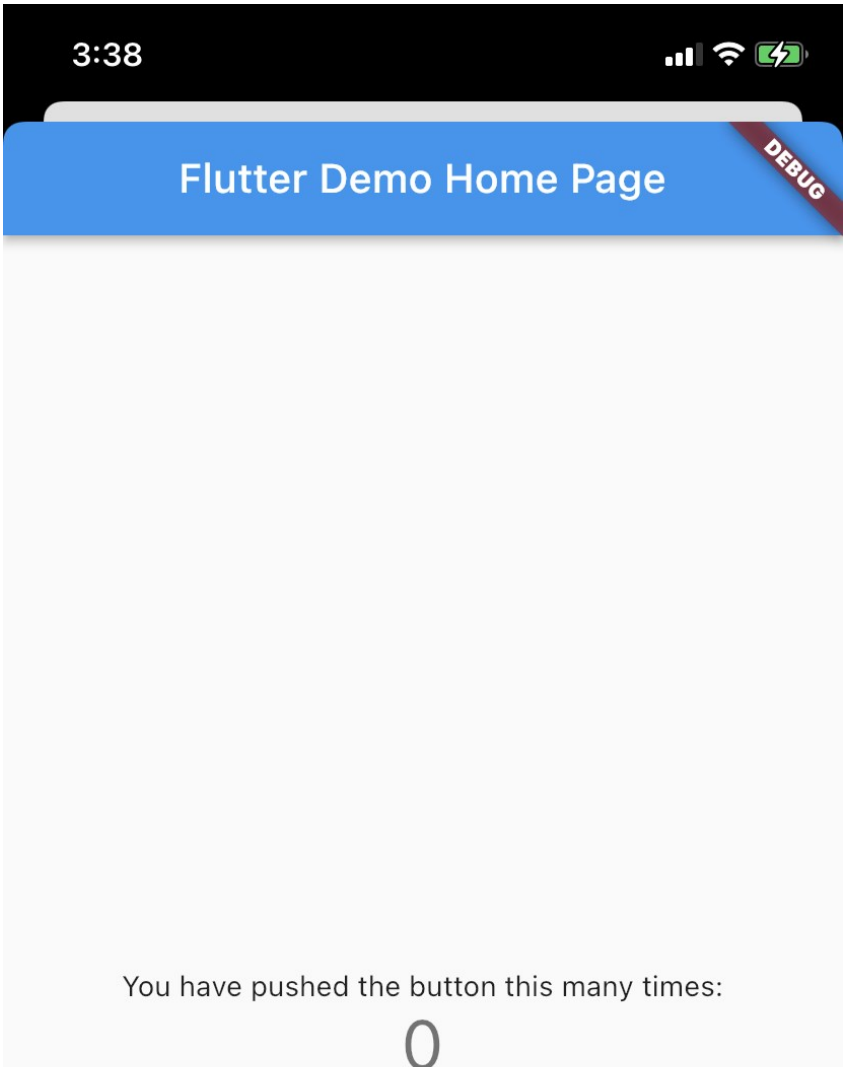
至此，就将App.xcframework和Flutter.xcframework都导入进工程了。

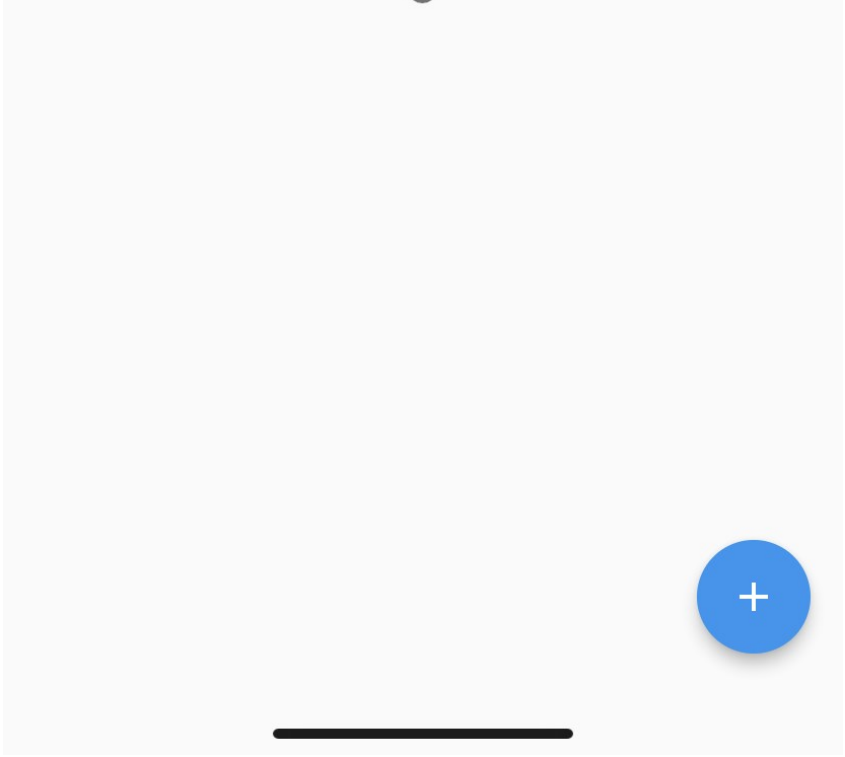
大家现在可以想一下，App.xcframework和Flutter.xcframework到底有啥区别呢？**App.xcframework是Flutter工程师在Flutter-Module工程里面编写的Flutter代码编译之后的产物；Flutter.xcframework是Flutter引擎，它是用来解析App.xcframework的。**

接下来我在原生工程中测试一下：



运行之后，点击屏幕，发现调起了Flutter页面：





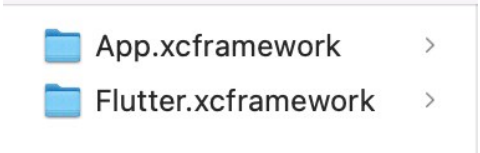
此时，**我一个原生开发工程师，在我的电脑上并没有安装Flutter开发环境，然后我将Flutter工程师打包好发给我的Framework导入到原生工程之中，运行之后就可以在原生工程中直接调用到Flutter相关的内容了**~

在团队开发的时候，这种混合工程开发的方式真的很方便，原生开发工程师完全不需要去配置Flutter开发环境，也不需要关心Flutter的版本以及安装路径，他只需要拿到Flutter工程师发过来的打包好的Framework然后导入到原生工程中就可以直接用了~

二、Cocoapods混合工程

上面介绍了在原生工程中直接拖入Framework的方式来配置混合工程，但是对于一个iOS工程，势必是需要通过CocoaPods来管理一些插件库的，所以我在想，是否可以通过CocoaPods来管理Flutter打包出来的部分Framework呢？答案是可以的，接下来就来介绍一下。

上面说了使用CocoaPods来配置混合工程的一个理由，其实还有另外一个非常必要的理由。我们想想，通过直接导出Frameworks的方式来配置混合工程，最后Flutter工程师导出的Framework产物中每一个模式下都会有两个Framework，如下：



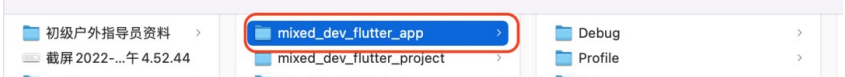
其中，App.xcframework是Flutter工程师在Flutter-Module工程里面编写的Flutter代码编译之后的产物，**Flutter工程师每一次打包出来的App.xcframework都是不一样的，因为Flutter代码变动了**；而Flutter.xcframework是Flutter引擎，它是用来解析App.xcframework的，**只要Flutter工程师使用的Flutter版本没有变化，那么他每一次打包出来的Flutter.xcframework就是一样的**，这样的话，其实完全**没有必要每一次打包Flutter-Module的时候都将Flutter引擎给打包成Framework，而是只需要将改动的Flutter代码打包成App.xcframework并且发给原生端即可**，这样的话省时省事省流量，一举多得。

好，上面说了使用CocoaPods来配置混合工程的两点理由，接下来我们来看一下如何进行配置。

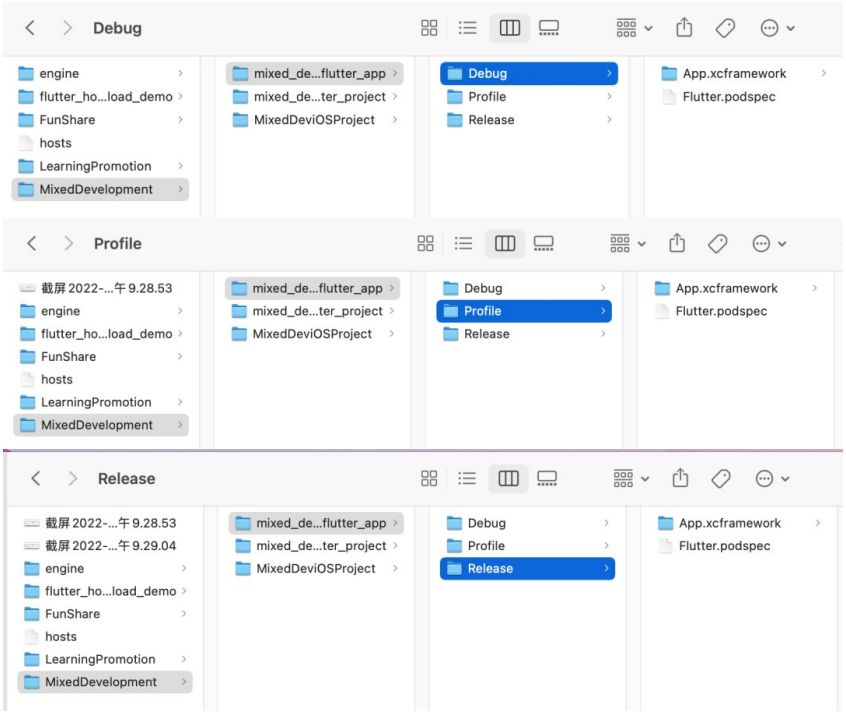
首先创建一个Flutter-Module，然后终端进入该Module文件夹路径，执行如下命令：

```
flutter build ios-framework --cocoapods --output=../mixed_dev_flutter_app
```

然后就可以在Flutter-Module工程的同级路径下找到构建产物了，如下：

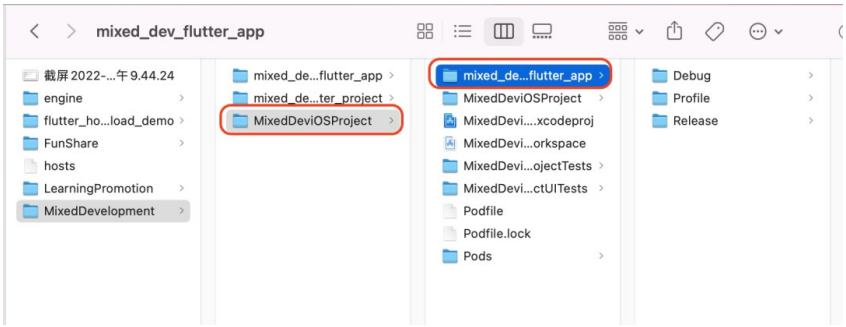


可以看到，跟之前一样，构建产物依然有Debug、Profile和Release三种模式。但是跟之前不同的是，每一种模式下面都只有一个App.xcframework，而Flutter.xcframework没有了，转而替换成了Flutter.podspec，如下：



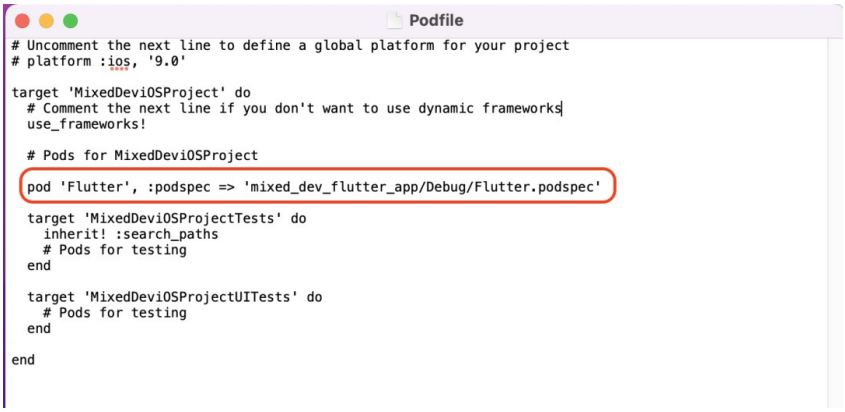
这里的**Flutter.podspec是CocoaPods的脚本，可以用来将Flutter引擎接入工程**。

接下来我们新建一个原生Xcode工程，然后将上面生成的mixed_dev_flutter_app拖入到根目录，如下：

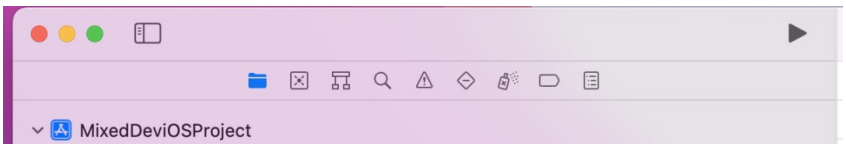


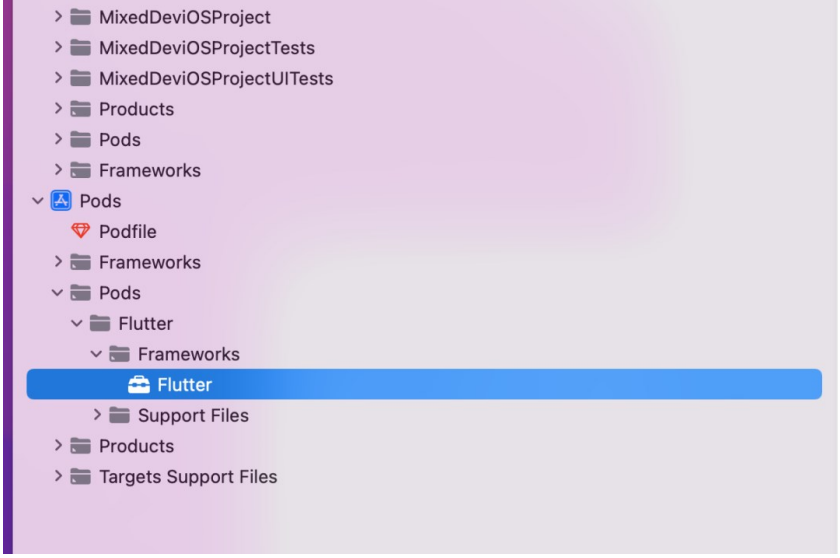
然后进入终端pod init，之后打开Podfile，加入下面一行代码：

```
pod 'Flutter', :podspec => 'mixed_dev_flutter_app/Debug/Flutter.podspec'
```



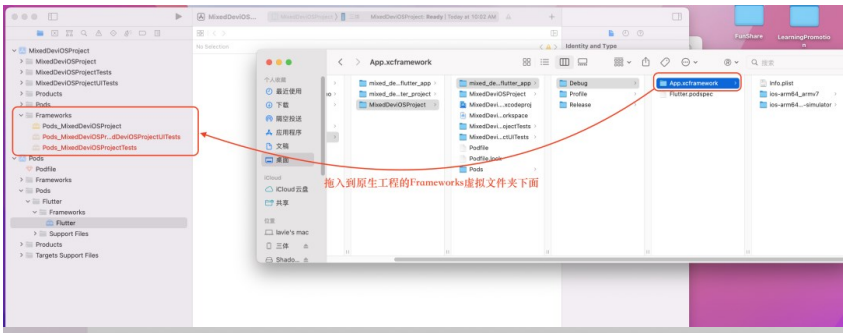
保存之后，pod install，然后我们在工程中就可以看到，**Flutter引擎已经通过CocoaPods的方式引入进来了**：

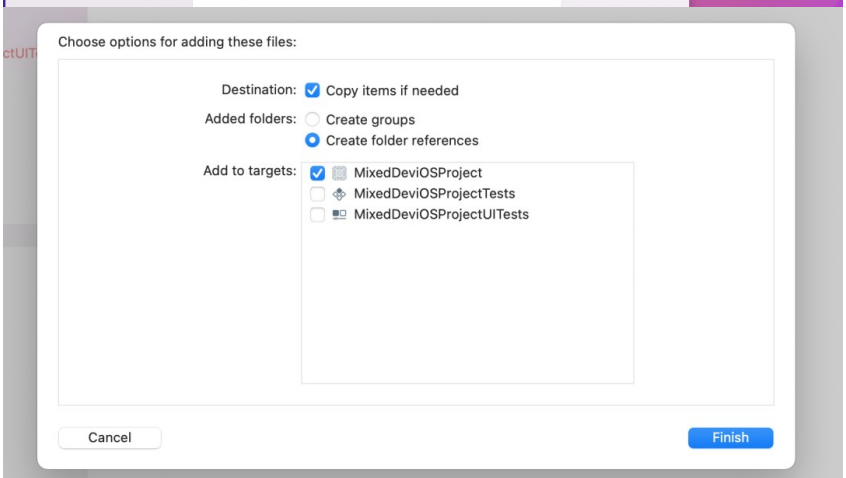


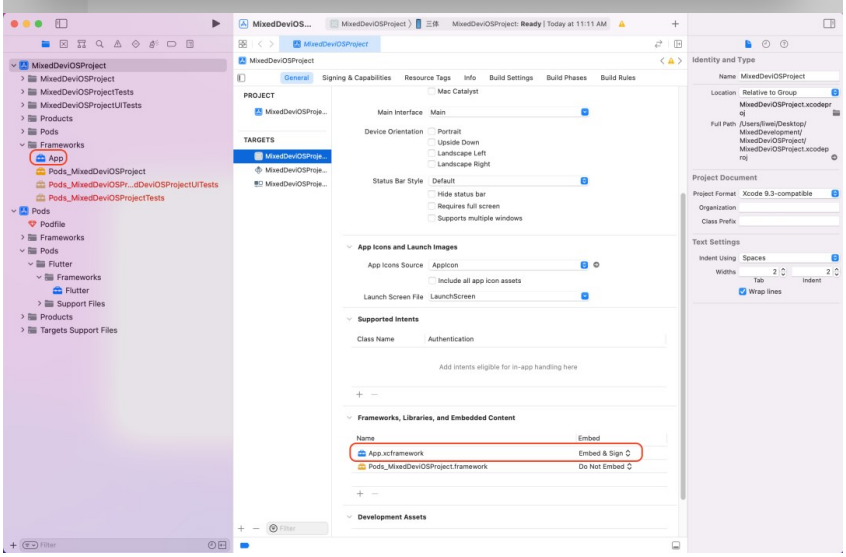


也就是说，通过这种方式，**Flutter工程师就不需要在每一次更新的时候都对FlutterEngine进行打包，而只需要打包更新的Flutter工程代码即可，Flutter引擎信息会存放进`Flutter.podspec`文件中，然后在iOS原生工程中会通过CocoaPods将FlutterFramework给导入进来。**此时，**原生开发工程师就只写原生代码即可，他不需要写任何Flutter代码，也不需要在自己电脑上进行任何Flutter相关的配置，更不需要关心当前使用的是哪一个Flutter版本。**

现在已经将Flutter引擎引入到工程中了，接下来还需要将Flutter工程代码导入进来：







这样配置好之后，原生工程就可以正常运行了，书写测试代码之后，发现也可以调起Flutter页面。

三、混合工程自动化

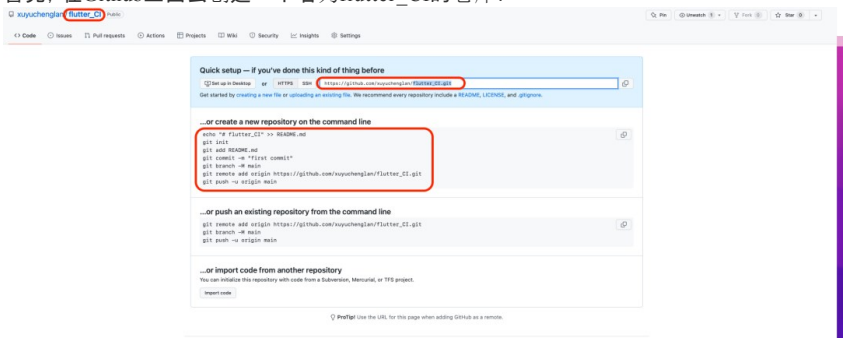
上面介绍了Flutter混合工程配置的两种官方方案，**这两种方案就是当前混合工程配置的最好的方案**；在官方推出这些方案之前，都是自己去抽取对应的Framework，然后进行对应的配置，是非常不方便的。

接下来我们就在上面介绍的这两种混合工程配置的基础之上，介绍一下如何通过Github搭建一个CI。

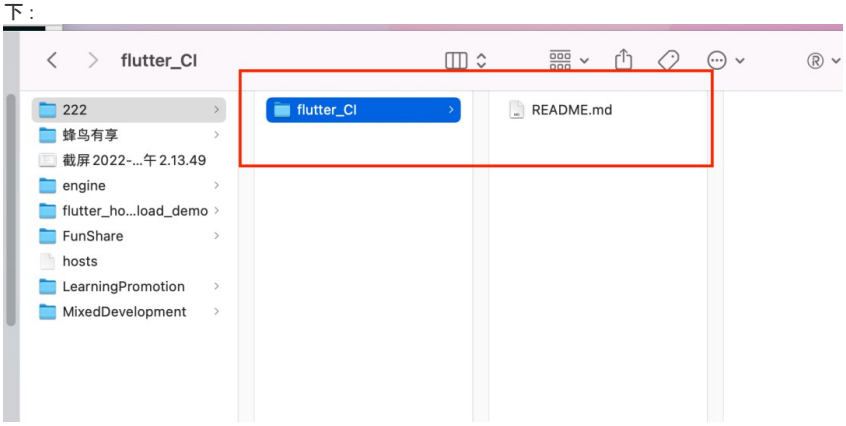
其实，所谓的**自动化，就是写脚本，通过脚本来自动执行相关操作**。比如说，编译Flutter引擎的过程很耗时间，而现在多人在做引擎调试，那么在多个人的电脑上就需要编译多次这个引擎，那就会很麻烦；此时，其实是**可以将编译的操作放在服务端的，而客户端只拉取编译之后的产物来进行运行，这就是典型的CI**。

接下来，我们就**将这个编译的过程放在Github上面**。

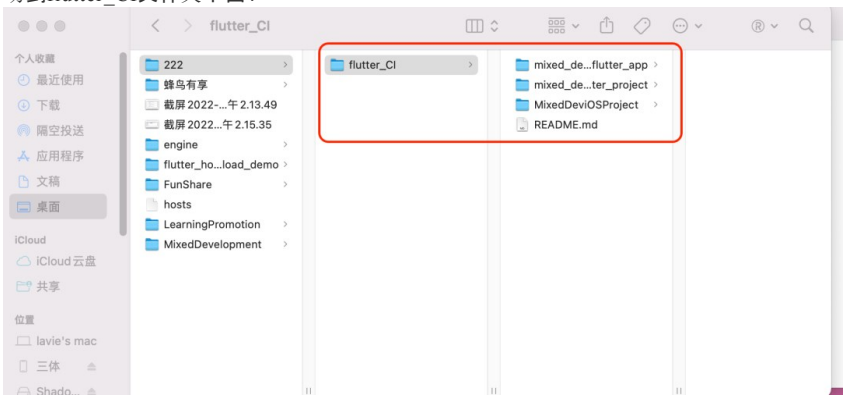
首先，在Github上面去创建一个名为flutter_CI的仓库：



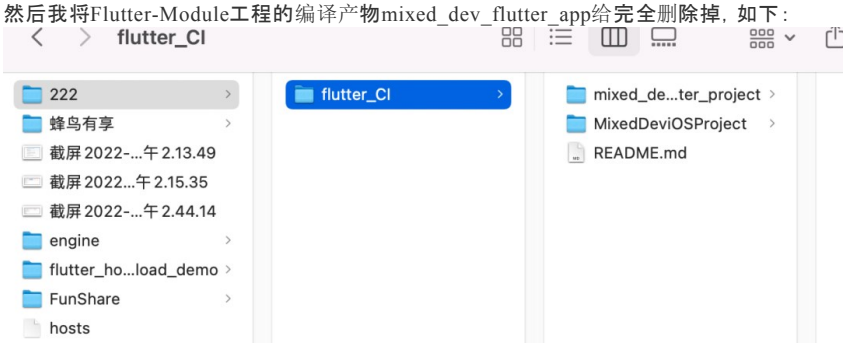
然后我们在合适的目录下按照上图标红指令创建一个工程，然后克隆到合适的目的目录下：



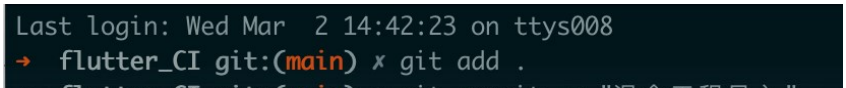
然后将上面第二个章节《CocoaPods混合工程》中创建并配置好的三个工程一口气儿都移动到flutter_CI文件夹下面：



然后我将Flutter-Module工程的编译产物mixed_dev_flutter_app给完全删除掉，如下：

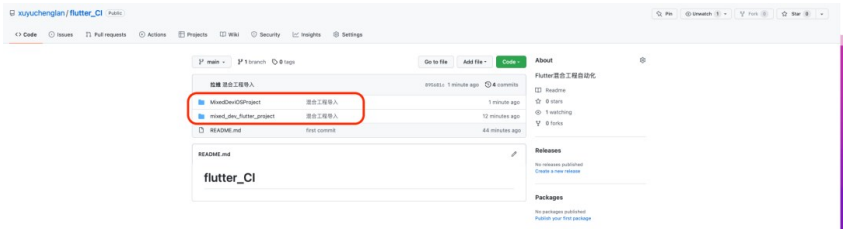


然后提交到远程Github：



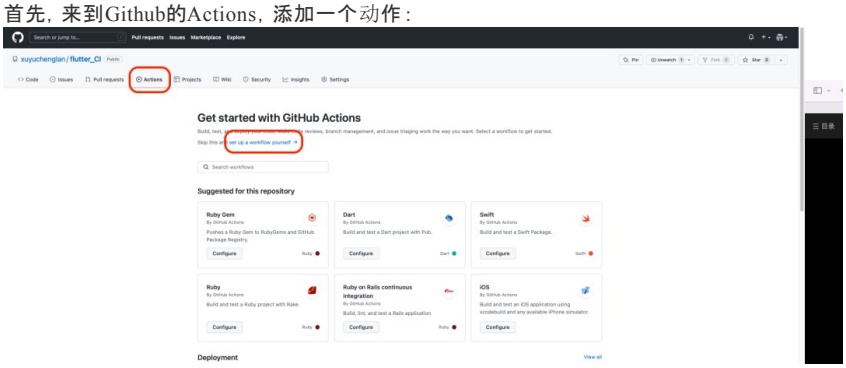
```
→  flutter_CI git:(main) ✗ git commit -m "混合工程导入"
→  flutter_CI git:(main) git push
```

然后来到Github，就可以看到我刚才提交的原生工程和Flutter工程了：



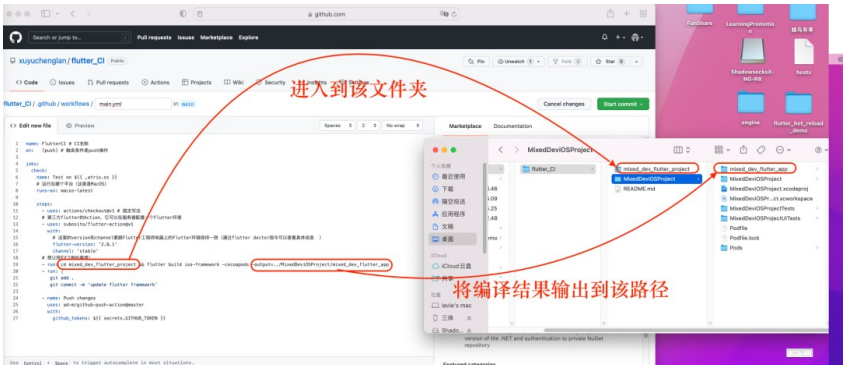
好，现在在Github的远程仓库flutter_CI中，有两个工程，一个是纯iOS原生工程，另外一个是Flutter工程。而Flutter-Module需要打包成Framework才能被iOS原生工程所使用，接下来就来介绍一下**如何通过Github的CI来远程编译Flutter-Module**。

首先，来到Github的Actions，添加一个动作：

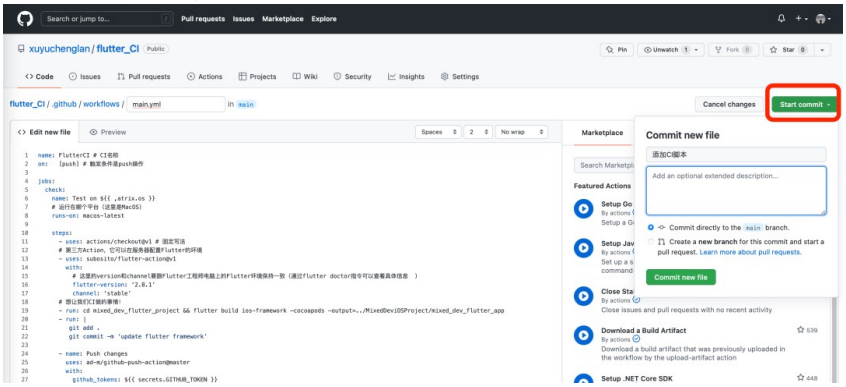


动作的脚本如下：

```yaml
name: FlutterCI # CI名称
on:   [push] # 触发条件是push操作

jobs:
  check:
    name: Test on ${{ matrix.os }}
    # 运行在哪个平台(这里是MacOS)
    runs-on: macos-latest

    steps:
      - uses: actions/checkout@v2 # 固定写法
      # 第三方flutter的Action，它可以在服务器配置一个Flutter环境
      - uses: subosito/flutter-action@v2
        with:
          # 这里的version和channel要跟Flutter工程师电脑上的Flutter环境保持一致
(通过flutter doctor指令可以查看具体信息  )
          flutter-version: '2.8.1'
          channel: 'stable'
      # 想让我们CI做的事情！
      - run: cd mixed_dev_flutter_project && flutter build ios-framework
  --cocoapods --output=../MixedDeviOSProject/mixed_dev_flutter_app
      - run: |
          git add .
          git commit -m 'update flutter framework'

      - name: Push changes
        uses: ad-m/github-push-action@master
        with:
          github_tokens: ${{ secrets.GITHUB_TOKEN }}
```



脚本添加完毕，并且检查无误之后，就点击“Start commit”，然后“Commit new file”：



之后我们就可以在Actions中看到该脚本正在执行，当**脚本执行完毕之后，我们的iOS原生项目的mixed_dev_flutter_app目录下就会多出来三个环境的编译打包产物，这个时候，我原生开发工程师只需要在自己的电脑上执行git pull操作，就可以将Github打包编译的Flutter-Module产物给拉取下来，这样的话就可以直接跑工程了。**
实际上，我在**脚本中监听的是Flutter工程师的Push操作**，他**每一次push之后，Github都会打包编译一次，然后就会将产物放到原生工程的mixed_dev_flutter_app目录下，原生工程师拉取一下远程代码就可以直接将产物拉取下来，然后就可以直接运行原生工程了**。

通过上面的介绍我们可以看到，**CI可以将混合开发的过程变得简单，节约时间，原生开发者和Flutter开发者互不干扰但又相互配合**，这在大企业里面是经常用到的。我们这里只是做了简单的介绍，后面我们真正在开发项目的时候，肯定不会这样简单地使用，到时候我们再根据自己团队的具体情况而对CI脚本进行完善。

关于自动化脚本，我后面会专门聊聊各种自动化脚本的搭建，因为里面的东西实在太多，这里就不做过多的讲述了。

以上。

收录于话题 #Flutter 58

下一篇 · Flutter的热重载原理 >