操作指南

**背景介绍**：

此项目通过修改Flutter 编译器以及Flutter引擎，打破Flutter产物生成链路，剥离数据段，并将数据段内置压缩或远程下发；同时篡改引擎加载逻辑，加载分离的数据段。通过这种方式可以大大减小Flutter产物体积，达到瘦身的效果，并为公司提供一套通用的、无损的，低成本接入的Flutter瘦身方案。

由于Flutte engine 代码量较大，以及环境配置耗时费力，因此我们提供源码的同时，也提供了各个版本编译后的产物。

**若需要自己编译，可将source中的源码替换engine中对应的文件编译即可。**

文件具体目录如下：

engine/src/third\_party/dart/runtime/vm/dart\_api\_impl.cc

engine/src/third\_party/dart/runtime/include/dart\_api.h

engine/src/flutter/shell/platform/darwin/ios/framework/Source/FlutterDartProject.mm

engine/src/flutter/shell/platform/darwin/ios/framework/Source/ FlutterViewController.mm

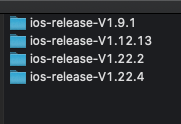
engine/src/third\_party/dart/runtime/bin/gen\_snapshot.cc

engine/src/third\_party/dart/runtime/vm/image\_snapshot.cc

engine/src/third\_party/dart/runtime/bin/snapshot\_utils.cc

**若直接使用：**

目前提供Flutter sdk v1.9.1, v1.12.13, v1.22.2,v1.22.4等版本



**使用方式：**

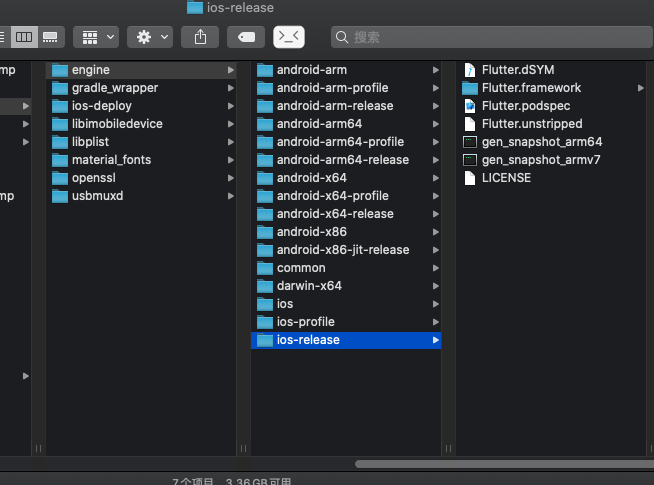
Thin-Flutter技术方案最终落地有两种方式

1. 远程下发（将剥离出的Flutter数据段以及资源通过远程下发的形式，下载到App沙盒内，瘦身效果最好）
2. 内置压缩（将剥离出的Flutter数据段以及资源压缩后内置在ipa内，通过版本管理App启动时解压）

最终启动改造后的Flutter engine，加载对应的产物。

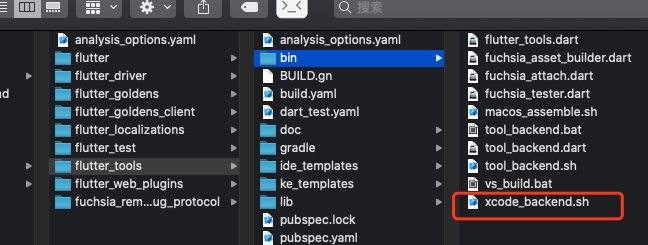
**使用步骤：以V1.22.2为例**

1. **复制对应产物到Flutter sdk**

选择对应版本，将对应ios-release-V1.22.2下的所有产物复制到Flutter SDK路径/bin/cache/artifacts/engine/ios-release下，替换ios-release文件夹下原有的产物。

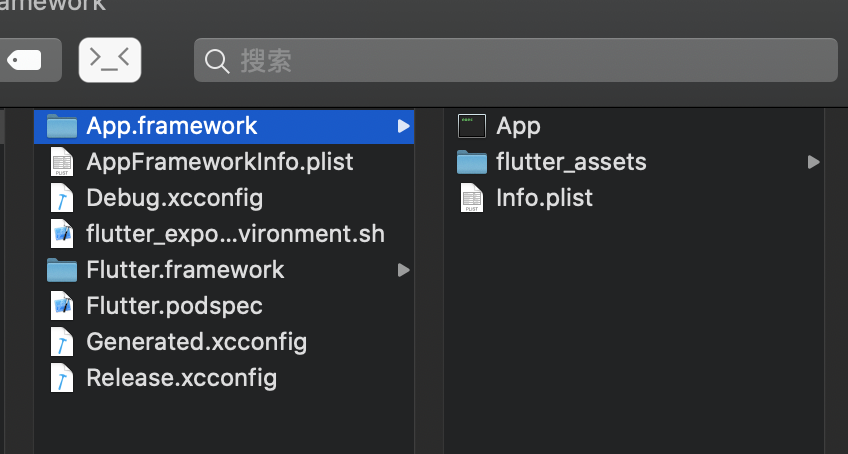
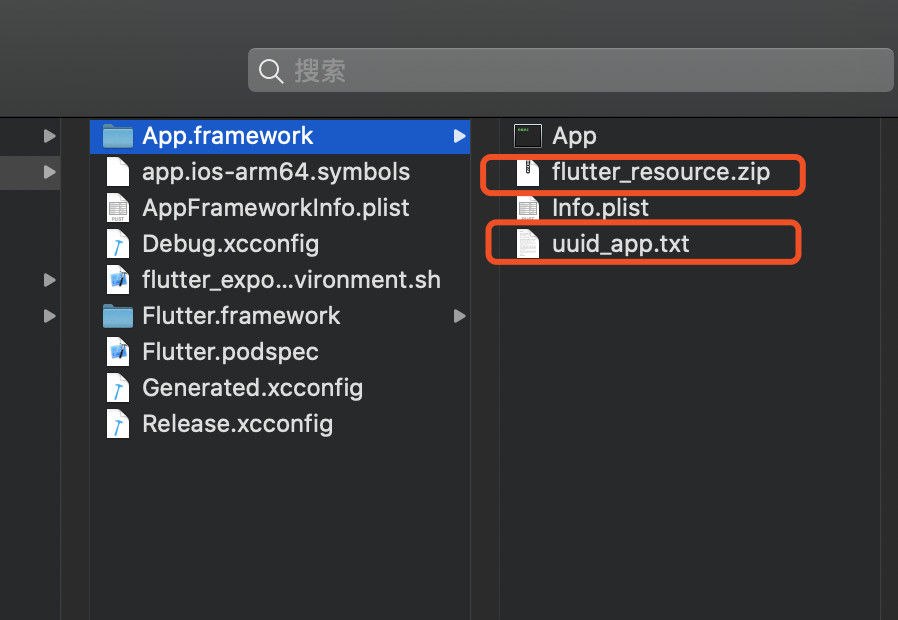
1. **复制压缩脚本，替换官方脚本**

将source/xcode\_backend.sh脚本替换官方Flutter SDK路径/bin/ flutter\_tools/bin/xcode\_backend.sh下的xcode\_backend.sh脚本，如下图

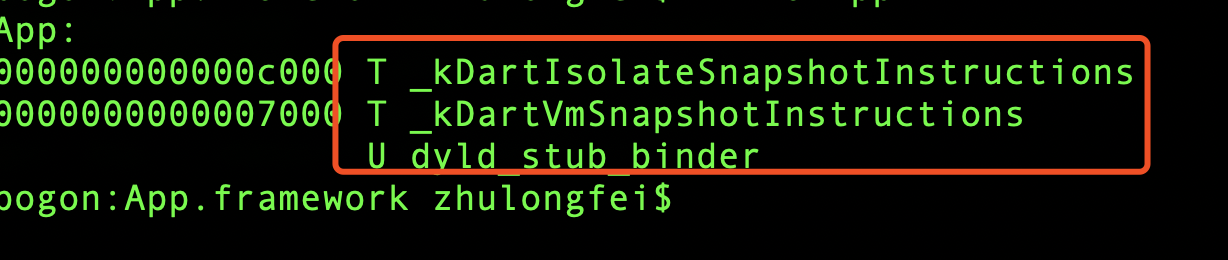


1. **编译Flutter工程**

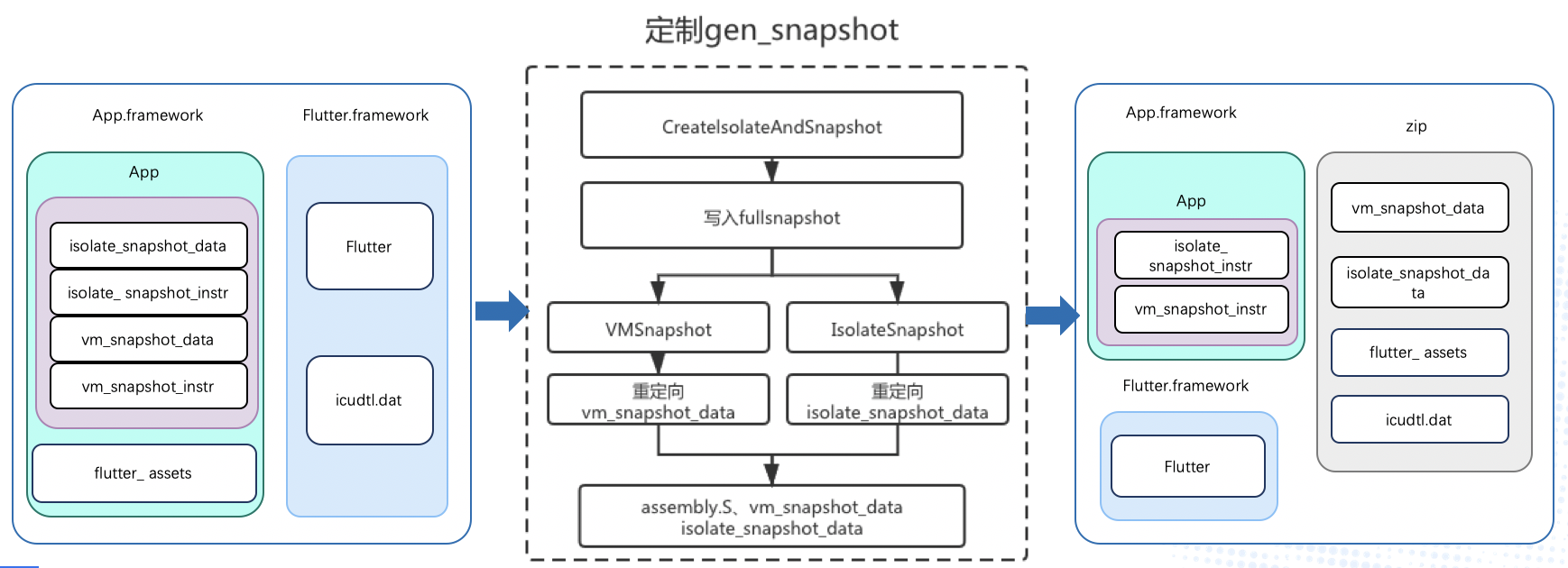
此时，编译flutter 工程会发现，App.framework的体积变小了，同时结构也发生了变化。

App可执行文件里只剩下了两个指定段。



这也是Thin-Flutter项目的核心，通过定制gen\_snapshot编译器将一部分产物剥离出来，压缩成zip，也就是flutter\_resource.zip，同时为了便于版本管理，会提供一份产物id,写在uuid\_app.txt内。



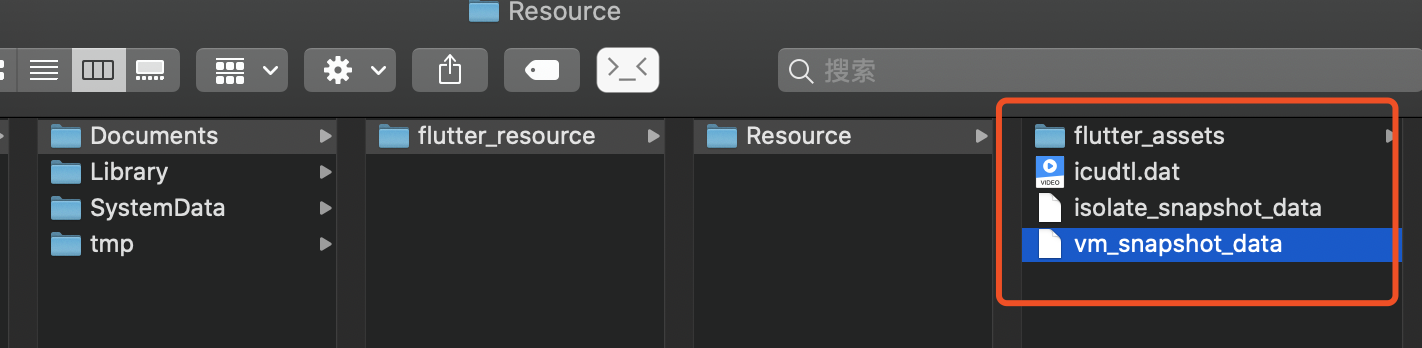
那么flutter\_resource.zip与uuid\_app.txt, 默认放在App.framework中，可以直接使用，也就是内置压缩方式。也可以将其剥离出来，放在远端服务器，在App初始化之前下载即可。看情况和自己公司的发布平台相结合。

另外，远程下发的方式，由于网络原因，会存在下载失败的情况，因此大家可以适当增加重试环节以及进度提示，给用户明确的加载感知，提高用户体验

1. **解压产物到指定目录**

无论是远程下发还是内置压缩，都要确保zip中的所有产物放在指定位置，

Document/flutter\_resource/Resource/



Done