易加油混合模式开发架构V1.0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **日期** | **版本** | **编写人** | **审核** | **修订说明** |
| 2021/12/05 | 1.0 |  |  |  |

目录

[易加油易加油混合模式开发架构V1.0V1.0 1](#_Toc1688804047)

[1.1. 引言 1](#_Toc2039267147)

[1.1.1. 需求背景 1](#_Toc123933509)

[1.1.2. 本文读者 1](#_Toc2038831820)

[1.1.3. 参考设计 1](#_Toc1397327208)

[1.2. 总体设计 1](#_Toc2144704911)

[1.2.1. 基础架构 1](#_Toc542424282)

[1.2.2. Flutter Project 基础目录结构 3](#_Toc456826059)

[1.3. 实战开发 5](#_Toc621535588)

[1.3.1. 资源使用 5](#_Toc788168508)

[1.3.2. 序列化与反序列化 5](#_Toc1068979260)

[1.3.3. 状态管理 5](#_Toc486232018)

[1.3.4. 代码规范 5](#_Toc926249691)

[1.3.5. 平台通道 6](#_Toc369599534)

[1.4. GitLab-CI 6](#_Toc1336660814)

[1.4.1. Stage 7](#_Toc431869631)

[1.5. 分支管理 8](#_Toc2085645004)

[1.5.1. Commit 日志规范 8](#_Toc60012247)

## 

## 引言

### 需求背景

为了维护开发规范以及后期人员加入，交接能有更详细文档阅读，便于后期人员快速入手开发工作。

采用混合模式开发优点有：目前易能电纯，警惠云flutter采用flutter语言开发，人员互备高，技术栈互通。

### 本文读者

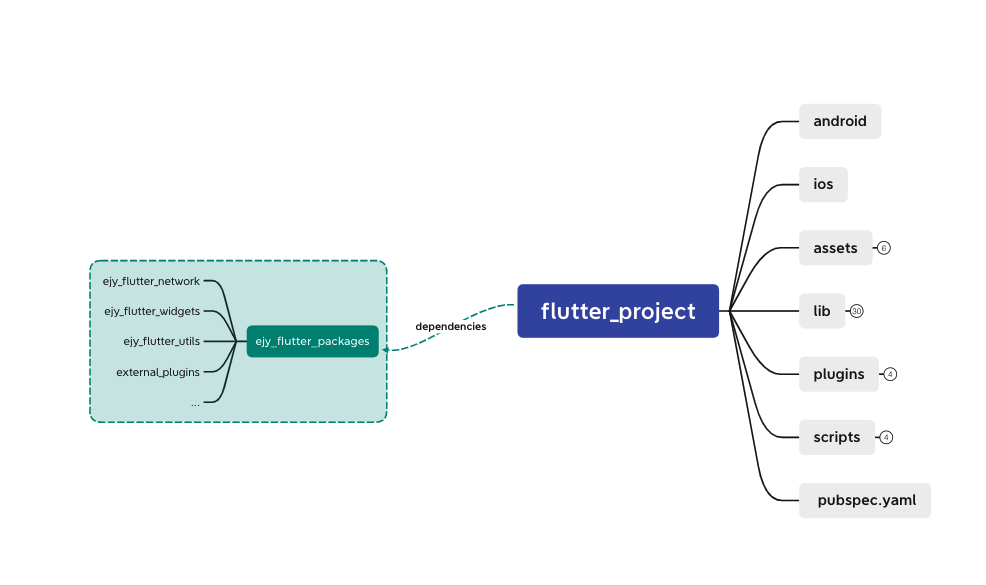
APP开发相关人员

### 参考设计

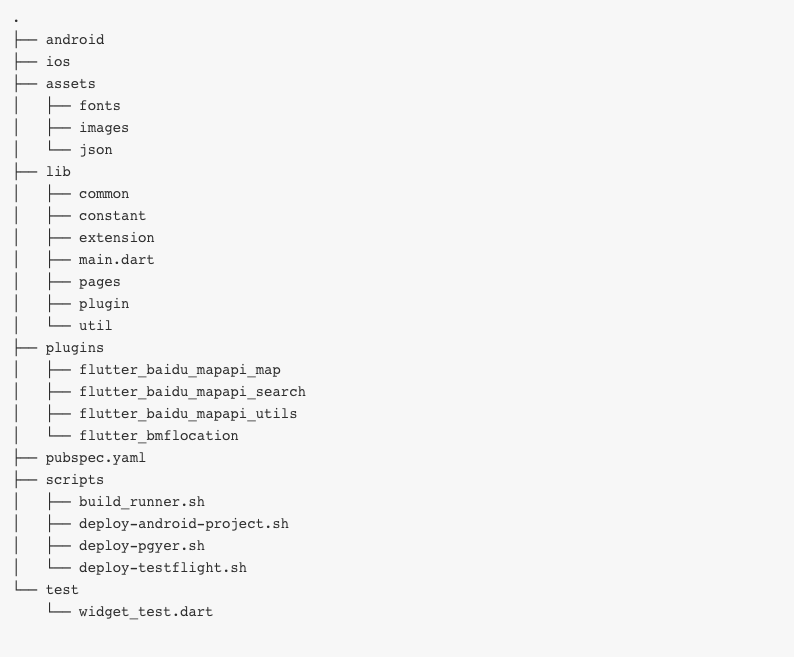
## 总体设计

### **基础架构**

目前易加油Java模块采用MVC模式，Kotlin采用MVP模式，后期加上了flutter插件，目前架构是JAVA+Kotlin+flutter混合模式开发。

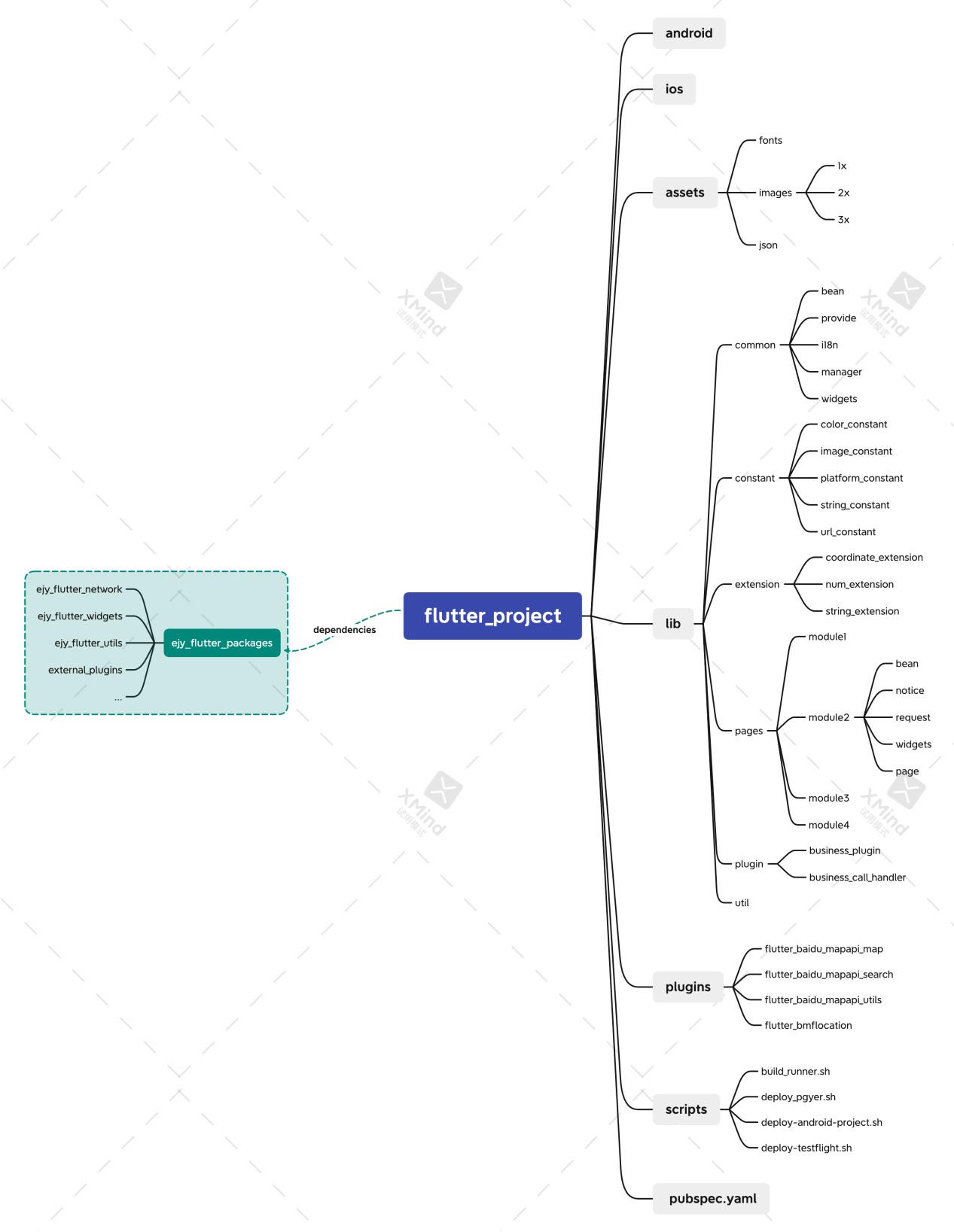


### **Flutter Project 基础目录结构**

**

* android/iOS 为Flutter运行的平台产物，有Flutter自动生成。开发人员无需改动。
* assets：项目所需的资源文件，已存放fonts images json。
* plugins: 外部的plugin，对于一些三方插件需要改动源代码。可以下载到本地以path形式依赖导入。
* scripts：脚本文件，除了开发使用脚本，还有gitlab-ci运行脚本。
* lib：存放的是Dart语言编写的代码，这里是核心代码。以main.dart为入口。

1. common： 通用的类文件。如bean(公用的数据模型)， manager(全局管理者), widgets(统一的ui组件)。
2. constant：常量文件，如color\_constant，image\_constant，url\_constant，string\_constant。
3. extension： 扩展文件
4. pages：根据业务模块区分，每个模块以文件隔离。模块下分别包含page, widgets, bean, request。
5. plugin: 平台通道，用于flutter与原生交互。
6. util：独立的工具类。

****

## **实战开发**

### **资源使用**

对于图片，字体等文件，不仅要将其放置于assets文件中，而且要将文件名称写入统一的constant配置，由constant统一管理。

### **序列化与反序列化**

在项目开发中，必不可少的就是与后端交互，后端返回的response大多是jsonString，如果直接转为Map使用也很不方便。在json转model，model转json中dart有内置的json解码器，但缺点是必须要手动解码，这样在手动编写时很容易逻辑出错从而导致程序抛出异常。所以在大型项目中一般选择自动序列化方式，可以避免人为出错，从而提高开发效率。

Json\_annotation和json\_serializable为Google官方提供的优秀pub库，build\_runner则可以自动生成代码。

### **状态管理**

使用官方推荐的Provider来实现MVVM架构的状态管理

setState()这种方法简单直接，而且也可以理解为一种MVVM模式，只不过View和Model仍然耦合在一起，ViewModel并没有承担起它应有的角色。随着我们的工程变得越来越大时，代码里的setState()就会变得越来越多，显得非常混乱，并且有时候会忘记调用setState()，导致浪费很多时间来定位问题。

官方早期也提供的一种状态管理模式叫做BLOC。这种方式依赖于第三方包rxDart，以流(Stream)的方式很好地解决了setState()的问题。但是这种学习难度较大，对Flutter的新手并不友好。后来出现了一种第三方库Provider，这是一种先进的状态管理和依赖注入的工具，并且易于学习和理解，所以目前官方也推荐首选Provider。

### **代码规范**

1. 库、包、目录、源文件名字规则:小写加下划线[\_]

import 'package:dokit/dokit.dart';

import 'file\_system.dart';

1. 类、枚举、typedef和类型参数应该采用大驼峰命名

class MyHomePage extends StatefulWidget {

}

typedef ShowDataFunction = bool Function(

BuildContext context,

);

1. 类成员，顶级定义，变量，参数和命名参数应该采用小驼峰命名

var item ;

HttpRequest httpRequest ;

void align （bool clearItems ）{ // ... }

1. ejy私有package库以统一前缀命名

class ECoordinate {}

class EGpsUtil {}

class ERequestManager {}

1. UI代码分层

Widget build(BuildContext context) {

return Scaffold(

body: SingleChildScrollView(

child: \_buidContent(),

),

);

}

Widget \_buidContent() {

return Column(

children: <Widget>[\_buildTopWidget(), \_buildNewsWidget()],

);

}

\_buildTopWidget() {}

\_buildNewsWidget() {}

### **平台通道**

由于Flutter很多事件需要依赖原生去完成，所有需要搭建平台通道用于flutter和原生之间的交互。

MethodChannel： Flutter向原生发送事件

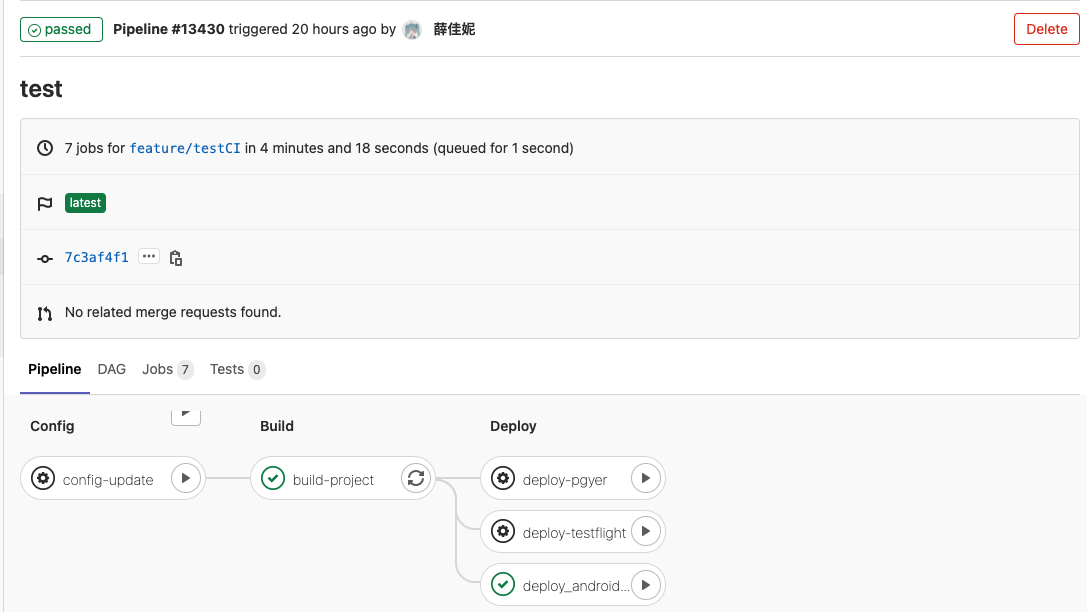
EventChannel： 原生向Flutter发送事件

## **GitLab-CI**

目前，易能电flutter项目已配置CI流水线自动打包。无需手动打包修改build号，通过deploy Stage即可打包发布。

原则上不允许手动打包，自动打包可以避免：

1. flutter跨平台，不同平台互相打包的问题
2. 无需切换分支，线上自动处理
3. 保证代码唯一性，避免因每个人本地缓存不一致打出的包差异性问题



### Stage

config为预留stage, 目前暂无配置, 可以忽略。

build为check stage, 代码编译错误会无法进行到下一阶段。

deploy为打包stage, 可配置不同平台

#### build

自动运行job: build-project

#### deploy

全部为手动运行job, 需要打什么包自己选择即可。

1. deploy-pgyer

​ 发布iOS和安卓包到蒲公英

2. deploy-testflight

​ 发布iOS包到App Connect, 自动配置版本号无需修改

3. deploy\_android\_project

​ 打出安卓渠道包，自动配置版本号无需修改

## 分支管理

master: 主分支, 最新线上版本分支

develop: 开发分支，始终保持最新完成以及bug修复后的代码

feature: 功能分支，开发新功能时，以develop为基础创建feature分支，只与develop交互

release: 预发布分支，由develop合入

hotfix: 热修复分支，由release分支创建，修复完成后再合入release, develop, master

### Commit 日志规范

建议参考规范：<type>(scope)：<subject>

比如：fix(首页模块)：修复弹窗 Bug。

type 表示 动作类型，可分为：

fix：修复 xxx Bug

feat：新增 xxx 功能

test：调试 xxx 功能

style：变更 xxx 代码格式或注释

docs：变更 xxx 文档

refactor：重构 xxx 功能或方法

scope 表示 影响范围，可分为：模块、类库、方法等。

subject 表示 简短描述，最好不要超过 60 个字，如果禅道有相关 Bug，建议在描述中加上。