# 

# iOS技术文档

编写日期：2018.1.20

编写人：刘南增

[iOS技术文档 0](#_Toc504227957)

[1. 简介 2](#_Toc504227958)

[2. 语言 3](#_Toc504227959)

[3. 架构 4](#_Toc504227960)

[3.1目的 4](#_Toc504227961)

[3.2方案 5](#_Toc504227962)

[4.框架 6](#_Toc504227963)

[5.版本管理 8](#_Toc504227964)

[6.持续构建 9](#_Toc504227965)

## 简介￼



目前 iOS 的市场需求量是不如前几年移动互联浪潮刚来的时候大的，但是iOS 并没有没落，而是对 APP 产出的质量要求更高。蛋壳 iOS 也在大量的项目实战中不断拓展我们的技术栈、针对不同项目选择最优架构和开发模式等等，研究怎么写出更易于维护、更优雅的代码。

## 语言



语言目前 iOS 开发就是 OC 和 Swift，蛋壳目前仍以 OC 为主， Swift 方面若要在团队里使用，毕竟它的 ABI 还没完全稳定。目前还是风险大收益小的，但最好保持对它的关注，因为苹果到后面肯定是要主推 Swift 的。

## 架构

### 3.1目的



客户端的架构可以说都是为了提高开发协作效率，因为功能可以用多种方法实现，可以不需要什么架构，无论大型还是小型 APP 都可以按一套来实现，只不过差的架构在中大型 APP上代码会比较混乱，导致在开发/协作/debug上效率会越来越低，好的架构则会提升这里的效率。业界有各种各种的架构模式，像 MVC/MVVM/MVP/VIPER 等。

### 3.2方案

我们目前常用的架构有两种MVCS 和 MVVM，如果是非常大型的项目，我们还会把不同的功能的业务进行完成解耦，利用 Cocoapods 封装成组件，再引入项目中，便于维护和管理。关于具体项目中怎么选择架构，其实一个项目中并非一定只使用一种架构，而是可以根据不同业务进行选择，不论使用了哪种架构，其目的是减少Controller 的代码量已达到便于维护的目的，经过大量的验证后我们总结出一种最优方案：

1. 先分析一个界面的结构，哪些是可能会增加内容的，哪些可能会插入内容动态变化的，将界面进行拆分。
2. 根据第一点拆分好的界面分别构造专门的 UIView 的子类，这些 view 可以专注于管理好自己的子控件和处理一些业务，这些 view 可以称之为视图层面的组件，可不可复用看具体需求。
3. 然后 Controller 将这些 view 进行拼装和自动布局。完成这些工作后有没有发现Controller 变得很清爽，最后一个问题是view 中界面跳转问题，其实很简单，只需要拿到根控制器的最上层控制器就可以完成跳转功能了。

这个方案一套流程下来我们会发现控制器瘦身了，而 View 层分工也明确不臃肿，不但提高了开发效率，也易于维护。

## 4.框架



在框架的选择上，我们不推崇重复造轮子，第三方框架无法满足需求的我们会选择自己封装。目前常用的框架如下（注：Dankal 表示内部封装，Three 表示第三方）：

1. 网络层：DKNetworking（Dankal）
2. 加载动画：DKProgressHUD（Dankal）
3. 滑动管理视图：DKViewPage（Dankal）
4. 扫描二维码/条形码：DKScanner（Dankal）
5. 常用分类：DKExtension（Dankal）
6. 图片选择：DKPhotoPicker（Dankal）
7. 下拉菜单：DKPullDownMenu（Dankal）
8. 函数响应式编程：ReactiveCocoa（Three）
9. 网络底层：AFNetworking（Three）
10. 网路图片加载：SDWebImage（Three）
11. 字典模型转换：MJExtension（Three）
12. 上下拉刷新：MJRefresh（Three）
13. 本地缓存：YYCache（Three）
14. 图表 : ZFChart（Three）
15. 文本处理：YYText（Three）

各种框架比较多，只列出了一部分，目前现有的框架基本可以完成所有需求了。我们会对框架进行分析，选择适合的。以高效，高质量完成产品需求。

## 5.版本管理



蛋壳现在使用的版本管理工具是 git，我们有自己的一套 gitflow 工作流，可以在团队开发时快速合并代码，并基本杜绝冲突，举个例子，假设现在两人合作开发：

1. 分支的创建：master（发布分支），dev（开发分支），me（ 自己分支），other（另一个人）
2. 提交代码流程：在自己分支先提交到远处仓库，然后切到开发分支合并代码，合并完成后提交到远程仓库，最后切回自己分支 rebase 开发分支完成两条分支同步。
3. master 分支是发布版本时使用的，操作和以上步骤基本一致，但是流程长一些，大概顺序是: me->dev->master->dev->me

这样做的好处是可以清晰的看出每次提交代码的来龙去脉，不会污染 master 分支，自己分支每次提交代码后保持和dev 分支同步。

## 6.持续构建



Jenkins + Fastlane + Fir：

1. 这个组合的核心是 Fastlane，Fastlane 简化了 iOS 的打包流程。简化和自动化 App 发布或部署的过程，将之变成一条平直的工作流。
2. 通过 Fastlane + Fir 系列脚本配合 Jenkins 的触发构建脚本即可实现一键打包。