App架构设计经验谈:技术选型

2016-03-17 安卓应用频道

(点击上方公众号，可快速关注)

> 作者：Keegan小钢
> 网址：http://keeganlee.me/post/architecture/20160114

系列文章：

App架构设计经验谈:接口的设计

当你做架构设计时，必然会面临技术选型的抉择，不同的技术方案，架构也可能完全不同。有哪些技术选型需要做决策呢？比如，App是纯原生开发，还是Web App，抑或Hybrid App？iOS开发，语言上是选择Objective-C还是Swift？架构模式用MVC，还是MVP，或者MVVM？下面根据我的一些经验对某些方面做点总结分享。

原生/H5

关于用原生好，还是用H5好的争论从没间断过。但我觉得，脱离了实际场景来讨论孰好孰坏意义不大。就说我们目前正在做的项目，先说明下背景：

1. 不止要做Android和iOS App，也要做微信公众号；
2. H5人员缺乏，只有一两个兼职的可用，而且不可控因素很高；
3. 我们对原生比较熟；
4. 开发时间只有半个月。

首先，需求上来说，大部分页面用H5实现，可以减少很多工作量。但因为不可控因素太高，而时间又短，风险太大。而我们对原生比较熟，开发效率比较高，很多东西我也控制得了，风险相对比较低。而且，我们的主推产品是App，微信属于辅助性产品，所以，微信要求也没那么高。因此，我决定以原生为主，H5为辅，App大部分页面用原生完成，小部分用WebView加载H5。

另外，WebView加载H5也有两种模式，一种是加载服务器的H5页面，一种是加载本地的H5页面。加载服务器的H5页面比较简单，WebView只要load一下URL就可以了。加载本地的H5页面，则需要将H5文件存放在本地，包括关联的CSS和JS文件。这种方式相对比较复杂，不过，加载速度会比第一种快很多。我们当前项目基于上面考虑，只能选择第一种方案。

如果人员和时间资源充足的话，那又如何选型呢？毫无疑问，我会以H5为主，微信和App都有的页面统一用H5，App专有的部分，比如导航栏、标题栏、登录等，才用原生实现。另外，WebView里的H5有点击事件时，也许是URL链接，也许是调用JS的，都不会让它直接在该WebView里做跳转，需要拦截下来做些原生处理后跳转到一个新的原生页面，原生页面也许嵌入另一个WebView，用来展示新的H5页面。这是简单的例子，关于Hybrid App详细的设计，以后再讲。另外，关于H5，绝对是大趋势，强烈建议所有App开发人员都去学习。

Objective-C/Swift

我在项目中选择了Swift，主要基于三个原因：

1. Swift真的很简洁，生产效率很高；
2. Swift取代Objective-C是必然的趋势；
3. 目前iOS只有我一个人开发，不需要顾虑到团队里没人懂Swift。

如果你的团队里没人懂Swift，那还是乖乖用Objective-C吧；如果有一两个懂Swift的，那可以混合开发，并让不懂的人尽快学会Swift；如果都懂了，不用想了，直接上Swift吧。

当语言上选择了Swift，相应的一些第三方库也面临着选型。比如，依赖库管理，Objective-C时代大部分用CocoaPods，Swift时代，我更喜欢Carthage。Carhage是用Swift写的，和CocoaPods相比，轻耦合，也更灵活。我个人也不太喜欢CocoaPods，使用起来比较麻烦，耦合性也较高，我使用过程中也经常出问题，而且还总是不知道该怎么解决，要移除时也是非常麻烦。

再推荐几个关于Swift的第三方库：

1. Alamofire：Swift版本的网络基础库，和AFNetworking是同一个作者
2. AlamofireImage：基于Alamofire的图片加载库
3. ObjectMapper：Swift版本的Json和Model转换库
4. AlamofireObjectMapper：Alamofire的扩展库，结合了ObjectMapper，自动将JSON的Response数据转换为了Swift对象

MVC/MVP/MVVM

先分别简单介绍下这三个架构模式吧：

- MVC：Model-View-Controller，经典模式，很容易理解，主要缺点有两个：

1. View对Model的依赖，会导致View也包含了业务逻辑；
2. Controller会变得很厚很复杂。

- MVP：Model-View-Presenter，MVC的一个演变模式，将Controller换成了Presenter，主要为了解决上述第一个缺点，将View和Model解耦，不过第二个缺点依然没有解决。

- MVVM：Model-View-ViewModel，是对MVP的一个优化模式，采用了双向绑定：View的变动，自动反映在ViewModel，反之亦然。

架构模式上，我不会推崇说哪种模式好，每种模式都各有优点，也各有极限性。越高级的模式复杂性越高，实现起来也越难。最近火热的微服务架构，比起MVC，复杂度不知增加了多少倍。

我在实际项目中思考架构时，也不会想着要用哪种模式，我只思考现阶段，以现有的人力资源和时间资源，如何才能更快更好地完成需求，适当考虑下如何为后期扩展或重构做准备。就说我前段时间分享的Android项目重构之路系列中讲的那个架构，确切地说，都不属于上面三种架构模式之一。

写在最后

技术选型，决策关键不在于每种技术方案的优劣如何，而在于你团队的水平、资源的多寡，要根据实际情况选择最适合你们当前阶段的架构方案。当团队拓展了，资源也充足了，肯定也是需要再重构的，到时再思考其他更合适更优秀的方案。