49 | 谈谈App架构的演进

2018-08-18 李运华

49 | 谈谈App架构的演进
李运华
- 00:06 / 09:32

专栏截止到上一期，架构设计相关的理念、技术、实践已经基本讲完，相信你一路学习过来会有一种感觉，这些内容主要都是讲后端系统的架构设计，例如存储高可用、微服务、异地多活等，都是后端系统才会涉及。事实上确实也是如此，通常情况下我们讲架构设计，主要聚焦在后端系统，但这并不意味着App、前端就没有架构设计了，专栏所讲述的整套架构设计理念，虽然是来源于我的后端设计经验，但一旦形成完善的技术理论后，同样适应于App和前端。

首先，先来复习一下我的专栏所讲述的架构设计理念，可以提炼为下面几个关键点：

- 架构是系统的顶层结构。
- 架构设计的主要目的是为了解决软件系统复杂度带来的问题。
- 架构设计需要遵循三个主要原则：合适原则、简单原则、演化原则。
- 架构设计首先要掌握业界已经成熟的各种架构模式，然后再进行优化、调整、创新。

复习完我们就可以进入今天的正题，我来谈谈App架构的演进，以及上面这些架构设计关键点是如何体现的。

Web App

最早的App有很多采用这种架构，大多数尝试性的业务，一开始也是这样的架构。Web App架构又叫包壳架构，简单来说就是在Web的业务上包装一个App的壳，业务逻辑完全还是Web实现，App壳完成安装的功能，让用户看起来像是在使用App，实际上和用浏览器访问PC网站没有太大差别。

为何早期的App或者尝试新的业务采用这种架构比较多呢？简单来说，就是当时业务面临的复杂度决定的。我们以早期的App为例，大约在2010年前后，移动互联网虽然发展很迅速，但受限于用户的设备、移动网络的速度等约束，PC互联网还是主流，移动互联网还是一个新鲜事物，未来的发展前景和发展趋势，其实当年大家也不一定能完全看得清楚。例如淘宝也是在2013年才开始决定“All in 无线”的，在这样的业务背景下，当时的业务重心还是在PC互联网上，移动互联网更多是尝试性的。既然是尝试，那就要求快速和低成本，虽然当时的Android和iOS已经都有了开发App的功能，但原生的开发成本太高，因此自然而然，Web App这种包壳架构就被大家作为首选尝试架构了，其主要解决“快速开发”和“低成本”两个复杂度问题，架构设计遵循“合适原则”和“简单原则”。

原生App

Web App虽然解决了“快速开发”和“低成本”两个复杂度问题，但随着业务的发展，Web App的劣势逐渐成为了主要的复杂度问题，主要体现在：

- 移动设备的发展速度远远超过Web技术的发展速度，因此Web App的体验相比原生App的体验，差距越来越明显。
- 移动互联网飞速发展，趋势越来越明显，App承载的业务逻辑也越来越复杂，进一步加剧了Web App的体验问题。
- 移动设备在用户体验方面有很多优化和改进，而Web App无法利用这些技术优势，只有原生App才能够利用这些技术优势。

因此，随着业务发展和技术演进，移动开发的复杂度从“快速开发”和“低成本”转向了“用户体验”，而要保证用户体验，采用原生App的架构是最合适的，这里的架构设计遵循“演化原则”。

原生App解决了用户体验问题，没记错的话大约在2013年前后开始快速发展，那个时候的Android工程师和iOS工程师就像现在的人工智能工程师一样非常抢手，很多同学也是那时候从后端转行到App开发的。

Hybrid App

原生App很好的解决了用户体验问题，但业务和技术也在发展，移动互联网此时已经成为明确的大趋势，团队需要考虑的不是要不要转移动互联网的问题，而是要考虑如何在移动互联网更具竞争力的问题，因此各种基于移动互联网特点的功能和体验方式不断被创造出来，大家拼的竞争方式就是看谁更快抓住用户需求和痛点。因此，移动开发的复杂度又回到了“快速开发”，这时就发现了原生App开发的痛点：由于Android、iOS、Windows Phone（你没看错，当年确实是这三个主流平台）的原生开发完全不能兼容，同样的功能需要三

个平台重复开发，每个平台还有一些差异，因此自然快不起来。

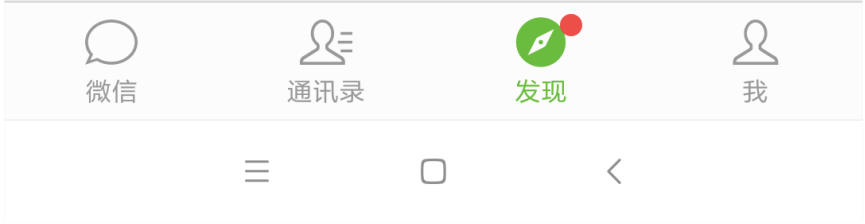
为了解决“快速开发”的复杂度问题，大家自然又想到了Web的方式，但Web的体验还是远不如原生，怎么解决这个问题呢？其实没有办法完美解决，但可以根据不同的业务要求选取不同的方案，例如对体验要求高的业务采用原生App实现，对体验要求不高的可以采用Web的方式实现，这就是Hybrid App架构的核心设计思想，主要遵循架构设计的“合适原则”。

组件化 & 容器化

Hybrid App能够较好的平衡“用户体验”和“快速开发”两个复杂度问题（注意是“平衡”，不是“同时解决”），但对于一些超级App来说，随着业务规模越来越大、业务越来越复杂，虽然在用户看来可能是一个App，但事实上承载了几十上百个业务。

以手机淘宝为例，阿里确认“All in无线”战略后，手机淘宝定位为阿里集团移动端的“航空母舰”，上面承载了非常多的子业务，下图是淘宝的首页第一屏，相关的子业务初步估计就有10个以上。

再以微信为例，作为腾讯在移动互联网的“航空母舰”，其业务也是非常的多，如下图，“发现”tab页就有7个子业务。



这么多业务集中在一个App上，每个业务又在不断地扩展，后续又可能会扩展新的业务，并且每个业务就是一个独立的团队负责开发，因此整个App的可扩展性引入了新的复杂度问题。

我在专栏第32期提到，可扩展的基本思想就是“拆”，但是这个思想应用到App和后端系统时，具体的做法就明显不同了。简单来说，App和后端系统存在一个本质的区别，App是面向用户的，后端系统是不面向用户的，因此App再怎么拆，对用户还是只能呈现同一个App，不可能将一个App拆分为几十个独立App；而后端系统就不一样了，采用微服务架构后，后端系统可以拆分为几百上千个子服务都没有问题。同时，App的业务再怎么拆分，技术栈是一样的，不然没法集成在一个App里面；而后端就不同了，不同的微服务可以用不同的技术栈开发。

在这种业务背景下，组件化和容器化架构应运而生，其基本思想都是将超级App拆分为众多组件，这些组件遵循预先制定好的规范，独立开发、独立测试、独立上线。如果某个组件依赖其他组件，组件之间通过消息系统进行通信，通过这种方式来实现组件隔离，从而避免各个团队之间的互相依赖和影响，以提升团队开发效率和整个系统的可扩展性。组件化和容器化的架构出现遵循架构设计的“演化原则”，只有当业务复杂度发展到一定规模后才适应，因此我们会看到大厂应用这个架构的比较多，而中小公司的App，业务没那么复杂，其实并不一定需要采用组件化和容器化架构。

对于组件化和容器化并没有非常严格的定义，我理解两者在规范、拆分、团队协作方面都是一样的，区别在于发布方式，组件化采用的是静态发布，即所有的组件各自独自开发测试，然后跟随App的某个版本统一上线；容器化采用的是动态发布，即容器可以动态加载组件，组件准备好了直接发布，容器会动态更新组件，无需等待某个版本才能上线。

关于手机淘宝App更详细的架构演进可以参考《Atlas：手淘Native容器化框架和思考》，微信App的架构演进可以参考《微信Android客户端架构演进之路》。

跨平台App

前面我介绍的各种App架构，除了Web App外，其他都面临着同一个问题：跨平台需要重复开发。同一个功能和业务，Android开发一遍，iOS也要开发一遍，这里其实存在人力投入的问题，违背了架构设计中的“简单原则”。站在企业的角度来讲，当然希望能够减少人力投入成本（虽然我站在程序员的角度来讲是不希望程序员被减少的），因此最近几年各种跨平台方案不断涌现，比较知名的有Facebook的React Native、阿里的Weex、Google的Flutter。虽然也有很多公司在尝试使用，但目前这几个方案都不算很成熟，且在用户体验方面与原生App还是有一定差距，例如Airbnb就宣布放弃使用 React Native，回归使用原生技术（https://www.oschina.net/news/97276/airbnb-sunsetting-react-native）。

前端的情况也是类似的，有兴趣的同学可以看看玉伯的文章《Web研发模式演变》，专栏里我就不在赘述了。

小结

今天我为你讲了App架构演进背后的原因和架构分析，希望对你有所帮助。

这就是今天的全部内容，留一道思考题给你吧，你认为App架构接下来会如何演进？谈谈你的思考和分析。

欢迎你把答案写到留言区，和我一起讨论。相信经过深度思考的回答，也会让你对知识的理解更加深刻。（编辑乱入：精彩的留言有机会获得丰厚福利哦！）

李奋斗

2018-08-18

端上的技术，山上的天儿，都变得太快。端上的架构怎么演进？我觉得要把答案交给想象力，把时间尺度拉大看，想象力才是端上复杂度的主要来源，交互革命和场景升级是端技术栈发展的重要推动力。鼠标的发明，颠覆了命令行的交互思维，Iphone的问世，分分钟让习惯了上屏下键的人们大开眼界，手机和网络的突进，解锁了一堆令人兴奋的场景。VR，混合现实，AI，5G等技术都可能极大推进端技术的变革，未来端架构怎么演进？不清楚，但有一点是清晰的，一大波复杂度，就在路上。

作者回复

2018-08-20

学不动了❀❀❀❀❀❀

borefo

2018-08-18

一个系统架构设计出来后，如何预估这个系统能够支撑多大的请求量呢？

作者回复

2018-08-20

不是先预估请求量，再设计架构么？

feifei

2018-08-20

我认为这个演讲也是朝着 all in one，即平台统一化，在app与原生接口间会出现统一化一个技术，类似java与jvm
原因：
1，原生开发成本高，每个平台都需要专门的开发人员
2，手机性能的提高，能够为平台统一化提供条件
3，用户体验，苹果的生态封闭，体验相对较好，但安卓平台很多公司都封装一套 导致体验差异很大

作者回复

2018-08-20

英雄所见略同❀❀

张玮(大圣)

2018-08-19

我的想法是：分久必合，合久必分

就像移动端技术随业务的发展一样，不断变换，但最终还是因为某一个很痛的点回归原生，鸡汤下，也就是走在路上时间久了，注意看看来时的路，记得当初为什么出发？❀❀

前几天在看graalvm，提供大一统平台，各种支持，如果单从技术角度来看，还是用最合适的技术解决合适的业务场景，合适的同时也成就了简单。

短时间看，架构遵循华兄说的合适，简单
长时间看，遵循演进原则。

kyll

2018-08-18

其实，对于现在很多业务应用强制将用户绑到移动端很是反感。举个例子，第一次用丰巢寄件，竟然花了半小时，注册很麻烦。有些餐厅强制手机点餐，在pc端登录强制扫二维码，也很无语。多设备多渠道本质应该是为了方便快捷，随时随地享用服务。任何设备都有局限性，做好自己的本职即可。

作者回复

2018-08-20

我也有点反感❀❀

文竹

2018-08-26

跨平台App仍是主要发展方向，此外一些跨平台的App快速设计产品也是主流（比如用Js编写，工具自动转换成原生APP）

作者回复

2018-08-26

React Native，weex，flutter

陶邦仁

2018-08-22

未来提供平台化，屏蔽底层原生，正如pc端cs到bs的演进

作者回复

2018-08-23

期待端出现一个JAVA，或者web一统天下，这样就不用两边甚至多平台重复开发

商伯阳

2018-08-20

在app整体做统一，未来一两年内还是比较困难吧感觉，我们目前就是遵守iOS和Android的规则。

就像浏览器厂商对接EcmaScript规范一样，前端开发接入要跟手机厂商，软件版本走，就导致了要打很多热补丁才保证了app view层的适配统一。

作者回复

2018-08-21

目前有很多尝试了，我看好Flutter，坐等打脸����

问题究竟系边度

2018-08-20

我觉得会，定义一个前端展示的实现接口标准，然后不同平台去实现内部展示逻辑。业务逻辑根据接口进行展示调用。
组件间能够进行部分升级而非整个app升级

我是做后端的，纯属瞎想。。

作者回复

2018-08-20

大一统的规范，目前就H5算是大家都支持，其它的很难落地，平台总是要一些差异化来展现自己的竞争力

kevenxi

2018-08-20

很有收获，是不具体分析几个APP的架构?

作者回复

2018-08-20

网上很多案例

朱月俊

2018-08-20

后面是否可以结合一个实际开源的项目，再结合前面各种方法论，好好解剖一个流行的分布式开源库？至少你讲的东西，一般人做起来因为各种原因都不一定做的下去。

作者回复

2018-08-20

这类事情很多人已经做了，我在如何学习开源项目中也给出了方法论和技巧

更多一手资源请添加QQ/微信1182316662

更多一手资源请添加QQ/微信1182316662

更多一手资源请添加QQ/微信1182316662