49 | 谈谈App架构的演进

2018-08-18 李运华 来自北京

《从0开始学架构》

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。

14:45 ... 移动 4G 65%

扫一扫 | 旅行包 双肩包 | 分红包 | 会员码

天猫 | 聚划算 | 天猫国际 | 饿了么 | 天猫超市

充值中心 | 飞猪旅行 | 领金币 | 拍卖 | 分类

淘宝头条

超赞 天气热得蛙仔都不想出门啦，

视频 由内致外全面升级：惠普战

淘抢购

00:14:09

有好货

高颜值美物

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。

看一看

搜一搜

购物

游戏

小程序

微信

通讯录

发现

我

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！）

精选留言 (36)

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: 学不动了😂😂😂

共 5 条评论>

73

江龙

2018-09-03

有个api的设计原则问题最近困扰好久，请教下，就是图中的首页其实有多个资源聚合，那是应该app端去请求多个资源服务，然后聚合出来展示；还是有个后台服务去聚合后端的各个基础服务，然后只提供一个接口给app访问？这其中的粒度如何把握？

作者回复: 访问量大的，核心业务用服务器聚合，性能好；
访问量小的，非核心业务用app聚合，可扩展性好；

59

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: 我也有点反感😊

共 3 条评论>

18

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: 英雄所见略同😄

共 3 条评论 >

14

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: 不是先预估请求量，再设计架构么？

12

木得感情的编码机器

2020-07-19

曾经做过一个移动端社交项目，开始就是用react native 开发的，但是工程里还是用了些原生的库和代码，而且是IM和拍照美化这两个核心功能。所以开发时既要边学边写原生代码，还要写RN，而且一旦RN在真机上报错，debug就是一头包。项目内测时就发现了RN光启动什么也不干就用掉了100多M内存，而且拍照美化是原生加RN，性能非常差。
内测之后就抛弃了RN，两个程序员很快就把安卓和苹果两个原生版本做出来了，因为很多原生代码可以复用，而且原生也有很好的开发框架和模式，调试非常方便和清晰。
我总结一下，是否用跨平台的方案取决于业务。如果做资讯类，信息聚合类，电商类对性能不苛求的项目，用RN这类东西完全可以。如果是游戏类，协同工具类，拍照视频类这些对性能有极高要求的，还是得用原生去开发，必要时也可以配合H5混合实现活动、任务、促销等业务。
未来前端的发展肯定是越来越复杂的，但要出现真正大一统跨平台开发框架还是需要很长一段时间，但是我这个东西一定是出至谷歌或苹果，而不会是FB或者其他大厂。现在有flutter ，但是说不准哪天swift 也能写安卓。

作者回复: 很好的案例

8

Niuniu

2019-01-16

我的观点可能有点相反，但凡有人上来要做native app的时候，我一般都劝退不劝进。先仔细想想你需要的那些功能WebApp能不能实现，用户体验有没有差很多，没有的话，没必要做na

tive app。能上webapp，先上webapp，人手不够时间紧，可以考虑hybrid，实在是非native app不可，才考虑写native的。

作者回复: 赞同，新APP首要是快速验证业务，不是体验

6

亚林
2019-05-07

前端技术茫茫多，所以我转后端了

作者回复: 后端更多啊，😂😂

共 3 条评论>

4

H
2020-05-08

有一个疑问，虽然在这节中提问不太合适哈哈哈。
疑问：如果一次mysql查询中，涉及到7-8个表的联合查询。就是join查询。除了暴力查询方法外，有没有其他办法？比如合表，把多个字段数据合在一起，放在一个表中。这样子可以吗？如果不对，望作者指正

作者回复: 可以的，这个叫做反范式设计，可以用在一些特殊场合，但不推荐普遍使用，因为数据库针对join做了非常多的优化，只要你的表设计没问题，join的性能是很高的

3

天外来客
2018-09-29

未来app必然朝着更好的用户体验，更快的开发速度方向发展，考虑到Android和iOS都是基于Linux底层，可能会出现一个平台层隔离两者的差异，提供统一的移动端系统API供开发者使用，希望这一天早点到来

作者回复: google Flutter已经在努力

共 2 条评论>

3

Monday

2020-10-20

分久必合，安卓和IOS终将大一统为跨平台 App

作者回复: 我看好H5，因为性能劣势被硬件的飞速发展弥补了，现在的App开发基本都是Native + H5，Native做底层，H5做业务

2

Eric

2018-11-20

老师，我想问下 分布式 这种架构也算是面向服务做切分的一种架构模式嘛？我个人理解 分布式服务 有按功能分割的 ，也有按任务计算资源分割的，还是说 分布式 本身是一种扩展方案？我概念上有点乱，不知道老师怎么定义 分布式 这个概念，谢谢

作者回复: 分布式是统称，泛指多台服务器联合起来完成业务功能，高性能，高可用，可扩展都可以用分布式架构

2

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: 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

共 2 条评论>

2

prader26

2021-04-19

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。

app 的演进：
Web App
原生 App
Hybrid App
组件化 & 容器化
跨平台 App

1

大鹏
2021-01-19

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。——这句话点拨了我，我自己在工作当中，往往忽略了这一点。事实大家都知道，但架构的原则会避免我们钻牛角尖

作者回复: 架构原则除了避免钻牛角尖，还可以帮助你做架构决策

1

小豹哥
2020-06-24

2016年刚毕业哪会，好多培训机构都力推安卓和IOS的课程，原来在2013年就是大势所趋了，一个技术的火爆程度有滞后效应，对技术要有前瞻性，还要有快速学习的能力来享受先发红利。

1

Kliyes
2019-03-11

微信小程序一统江湖！

作者回复: 不可能的

共 3 条评论>

1

killer
2018-09-28

各个平台的目标用户习惯本来就有差异。是个挑战

1

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。

1

龙程飞
2022-08-08 来自上海

这个应该主要看操作系统发展吧?