



# AliFlutter 体系化 建设和实践

阿里经济体内部关于 Flutter 的体系应用和思考

在当今应用生态环境下，跨平台解决方案一直备受关注，业界有越来越多公司尝试 Flutter。阿里巴巴实践 Flutter 的同时，一直在思考如何从经济体技术战略的层面拉通 AliFlutter 的体系建设。这本电子书，分享了阿里巴巴 Flutter 技术及业务应用的实践和思考。

—— 阿里巴巴集团副总裁 汤兴（平畴）



阿里云开发者社区



淘系技术



淘系技术



阿里巴巴淘系技术



bilibili 淘系技术

扫一扫二维码，关注我吧



阿里云开发者“藏经阁”  
海量免费电子书下载

## | 序言

淘系技术部隶属于阿里巴巴新零售技术事业群，旗下包含淘宝技术、天猫技术、阿里农村技术、闲鱼、躺平等团队和业务，是一支是具有商业和技术双重基因的螺旋体。

新零售技术事业群的使命是技术驱动新商业，淘系技术部的愿景是致力于成为全球最懂商业的技术创新团队，打造消费者和商家一体化的新零售智能商业平台，创新商业赛道。

依托淘系丰富的业务形态和海量的用户，我们持续以技术驱动产品和商业创新，不断探索和衍生颠覆型互联网新技术，以更加智能、友好、普惠的科技深度重塑产业和用户体验，打造新商业。

# 目录

阿里集团内如何进行 Flutter 体系化建设?	5
AliFlutter 客户端研发体系概览	20
闲鱼研发框架应用和探索	35
AliFlutter 图片解决方案与优化	50
UC Flutter 技术实践分享	64
基于 Flutter 的 Canvas 探索与应用	82
淘宝特价版开发体系的探索	91
ICBU Flutter 探索之路	106
Flutter 在饿了么的应用与沉淀	123

## 阿里集团内如何进行 Flutter 体系化建设?

2019 年无疑是 Flutter 技术如火如荼发展的一年。每一个移动开发者都在为 Flutter 带来的“快速开发、富有表现力和灵活的 UI、原生性能”的特色和理念而痴狂，从超级 App 到独立应用，从纯 Flutter 到混合栈，开发者们在不同的场景下乐此不疲的探索和应用着 Flutter 技术，也在面临着各种各样不同的挑战。

### 为什么是 Flutter ?

阿里巴巴集团内也有越来越多的业务和团队开始尝试 Flutter 技术栈，从闲鱼的一支独秀引领潮流，到如今淘宝特价版、盒马、优酷、飞猪等 BU 业务相继入局，Flutter 的业务应用在集团内也已经逐渐形成趋势。那么，是什么原因让集团内越来越多的开发者选择拥抱 Flutter 技术栈？Flutter 的哪些优势吸引了集团 Native 开发者们通过 Flutter 开发并交付业务？从技术上看，个人认为 Flutter 最核心的 3 个特点最为吸引开发者：

- 极高的开发与交付效率，良好的开发体验
- 优秀的跨多端多平台能力
- 极强的 UI 表现力

先说一下开发效率。从集团电商业务属性出发，业务响应效率及其背后的研发效率从来都是最为重要的指标。在保证体验的前提下，尽可能的提高研发效率，就意味着更高的生产力。传统的 Native 业务研发 iOS/Android 双端需要分别投入，研发成本高，端差异性大且依赖端侧发版，这也是为什么集团电商业务的活动类技术栈一直较为依赖前端体系，从 H5 到 Weex 到小程序，很大程度上就是在追求研发和交付效率以及灵活性。如今 Flutter 很好的解决了跨端一致性问题，一套代码无差异

的同时跑在 iOS 与 Android 两端；开发体验基本接近前端，支持 on device 的 Hot Reload，去年年底 Flutter 又推出了在 Android Studio 中通过插件实现实时预览并支持交互的 Hot UI 能力，以及 Layout Explorer 可视化布局，让 Flutter 的开发效率和前端效率基本持平。

电商业务发展到当前阶段，已经不再仅仅局限于移动端场景，越来越多的业务需求对跨端跨平台性提出了更高的要求。钉钉 / 干牛桌面端应用场景，甚至天猫精灵、线下门店等业务场景，从长远看都需要一个比 Web 性能一致性更好适配成本更低的多端方案。目前跨多端技术方案主要依赖于浏览器和前端体系，但浏览器本身的沙盒属性、与系统较低的结合度、以及在低端设备上较差的性能都降低了研发效率和用户体验，提高了业务的交付门槛。可以说目前集团内的跨多端多平台方案是实质缺失的。Flutter 从设计上就天然支持多平台开发，它的底层基于 Skia 跨平台图形引擎，向上构建出了一整套平台无关的渲染体系和事件处理体系，并紧贴 Native 研发模式自定义了基于 widgets 的声明 + 响应式编程范式，对系统能力依赖度低，并具备出色的跨平台还原度；支持多平台也是 Flutter 的战略目标之一。目前除了 iOS 和 Android，官方宣布支持的平台有 Mac、Windows 和 Web，Linux 也在开发中，它的技术特性也让将 Flutter 移植到 Linux based IoT 平台上成本很低，同时 Flutter 还是未来 Google 的下一代操作系统 Fuschia 的官方应用研发框架。可以说 Flutter 已经具备了成为下一代跨多端多平台研发模式的一切条件，围绕 Flutter 建立集团的多端多平台研发体系是非常可行的选择。

最后讲一下 UI 表现力。电商业务重体验，重交互，尤其对于流量精细化运营场景，富交互的游戏化表现方式已经成为流量促活的重要手段。在 UI 表现力方面，前端体系一直具备着优势，通过 CSS3 强大的动画能力，开发者可以非常容易的实现复杂的动画效果和交互体验，而基于 Native UI，需要借助各种动画特效三方库，双端开发体验不一致，实现复杂且交付效率低。Flutter 很好的解决了这个问题，从补间 (Tween) 动画、基于物理属性的动画，到相对复杂的页面间 Hero 动画、parallax 交错动画等特效，Flutter 都可以跨平台低成本的高效实现。这里贴一个去年年底

Flutter Interact 大会上 GSkinner 公布的基于 Flutter 实现的交互 Demo 让大家直观感受一下：可以看到 Flutter 的强大 UI 表现力，可以帮助开发者快速高效低成本的开发出极为炫酷的 UI，非常适合电商领域重 UI 视觉交互的各类场景，帮助业务构建出富有表现力的页面。

## Flutter 体系化建设现状

目前集团内有多个业务 BU 均已开始尝试应用 Flutter 技术栈，涵盖了从电商详情业务、导购频道，到 Feeds 流、游戏化交互以及国际化等多个业务场景。目前 Flutter 技术在集团应用的痛点在于，研发基础设施的中台基建不够完善，研发支撑能力与数据运维能力未实现标准化，集团 Flutter 开发者生态还未完全拉通，暂时未能形成合力。这些问题将是我们后面在集团层面建设 Flutter 技术体系的重点。另一方面从行业趋势上看，Flutter 技术已经成为越来越多行业伙伴重点投入的技术建设方向。字节跳动、美团等公司均建设了自己的 Flutter 工程化体系，并服务了各自的业务场景；腾讯也基于 Flutter 在多个 App 上进行了应用尝试，并在 Flutter 渲染能力服务小程序的场景下做了有益探索。行业伙伴们在 Flutter 技术上的投入力度和决心，一方面让我们对 Flutter 技术的应用前景和社区更有信心，另一方面也让我们感到联合集团各方力量共建 Flutter 生态的必要性和紧迫性。

## 手淘的尝试和思考

最后，简单讲一下我们从 18 年到现在在 Flutter 上做的探索和思考。手淘从 18 年 10 月开始探索 Flutter 渲染引擎应用在小程序场景；19 年下半年开始建设 Flutter 基础能力，并服务了淘宝特价版业务，在引擎、图片库、内存优化和加载性能等关键技术上做了沉淀；同时通过对 Flutter 的引擎改造，封装出 Flutter 2D Canvas 能力，向上支持小程序 Canvas 组件及小游戏引擎，服务 2D/2.5D 游戏化业务，并在业务场景中落地。在这个过程中，我们也沉淀了解决内存问题和图片问题等方案，以及 Flutter 技术与 Web 技术的对比与思考，取得了一定的技术及业务价值。通过这些尝试，加深了我们对 Flutter 技术的掌控力和理解。在我们看来，Flutter 的横空出世，

完全可以被看作吹响了 Native 体系复兴的号角。在保持 Native 性能优势的前提下，Flutter 带来了优秀的跨端一致性、贴近前端的研发效率，以及强大的 UI 表现力，为集团业务使用 Native 技术栈带来了新的可能。

**从业务应用上看：**Flutter 目前带来的最大价值是研发效率的提升。在基建和 native 扩展能力完备的前提下，开发基于 Flutter 的纯 Dart 业务的人效比之前各端分别开发的效率提高了接近 2 倍，单位时间内的需求响应能力也相应提高了接近 2 倍，目前已在闲鱼和特价版业务开发中得到了很好的工程化验证。

**从适应场景看：**Flutter 目前比较适合承载富图文内容，如详情、Feeds 流、用户主页等常规业务开发，以及 2D/2.5D 游戏场景以及富动效业务。Flutter 通过单端技术栈可以同时满足以前需要 iOS、Android 以及前端技术栈分别负责的业务场景，甚至可以通过端云一体化的开发模式使用 Dart 负责一部分服务端业务逻辑开发，可以帮助业务团队拓展业务边界的同时，实现前后端研发能力闭环。Flutter 目前的限制在于，动态性能力及前期的投入成本。前期投入成本主要指技术学习与团队研发模式升级的成本，涉及到技术路线选择，是我们和每个业务团队需要一起思考和判断的，这里不展开谈。动态性能力是 Flutter 的相对短板，目前能够通过 Flutter 模板化技术实现基于模板的组件级动态化能力，但基于性能、审核及对原生 Flutter 体系的侵入性等多种因素，目前还不能去直接实现 UI + 逻辑动态化能力。Flutter Web 方案虽然不存在审核限制，但受限于浏览器 DOM API 与 widgets 体系的差异性，目前仍旧存在较严重的性能瓶颈和渲染差异性，仅可作为降级的备用方案，暂时无法作为动态化的主要实现方案。未来在动态化方向的探索也将是个长期的博弈过程。如果后面我们可以解决好 Flutter 动态化的问题，那么 Flutter 完全有机会成为集团业务的核心研发模式之一。综上我们认为，入局 Flutter 的时机已成熟，合力共建 Flutter 集团移动生态，这件事情大有可为。

## AliFlutter – 经济体移动小组 Flutter 共建项目

在这样的背景下，经济体移动技术小组今年也将端侧架构治理的重点方向放在了

Flutter 上。移动技术小组从战略角度提出了 AliFlutter 项目，目标非常明确：

- 从经济体层面拉通 Flutter 体系建设，打造 Flutter 的公共技术基础设施，制定 Flutter 容器、中间件与 API 标准，建设 Flutter 研发支撑与数据运维能力，复用关键技术。
- 联合各 BU 构建经济体 Flutter 技术社区，沉淀共享集团 Flutter 技术及业务组件、能力与解决方案，服务集团 Flutter 业务，共建集团 Flutter 技术生态。
- 在经济体层面构建 Flutter 的对外影响力，联合各 BU 一致对外，打造阿里在行业内的 Flutter 技术制高点。

为经济体的 Flutter 技术体系“建基础、育社区、扛大旗”，我们责无旁贷。未来 AliFlutter 的整体架构如下所示：AliFlutter 建设的第一步，就是要构建集团的 Flutter 基础设施、提供公共容器与组件、研发支撑服务与标准化研发流程，为集团的 Flutter 业务提供一个基础的 Flutter 公共研发服务能力，搭建好技术共建共享的基础和平台；第二步，我们要服务好 Flutter 业务应用，探索业务应用模式与 Flutter 技术特性的结合点，并在过程中打磨技术，形成针对业务特点的解决方案与技术沉淀，真正盘活集团内的 Flutter 社区共建氛围与生态；第三步，面向未来，解决好 Flutter 应用的几个核心关键问题：跨端与交互能力、业务研发效率与业务交付效率，通过技术赋能业务，让 Flutter 真正成为集团移动业务的核心研发模式。接下来，就详细讲一讲每个阶段 AliFlutter 所做的工作和面向未来的思考。

## 基础设施建设

从 19 年 10 月 AliFlutter 项目启动开始到现在，我们基本构建起了一套 Flutter 的公共基础设施，包括 Artifacts 与 pub 库，Flutter 标准容器与 API 标准，并实现了 Flutter 的构建打包自动化，定义了标准的引擎定制工作流与业务研发工作流。目前基础设施已经具备支撑集团 Flutter 业务研发的能力，并支持各 BU 按需定制。

## Artifacts 仓库

产物服务器主要是为了配合引擎定制，加速通过 Flutter 命令获取 Engine 中间产物的后端服务，便于统一 Flutter 开发者的工作环境。开发者可通过设置 FLUTTER\_STORAGE\_BASE\_URL 来将 Flutter 工具链获取 artifacts 的地址指向该服务，同时通过 namespace 便可实现定制化的获取 artifacts 的能力以及内网加速服务。Namespace 设计为区分不同 BU 的引擎产物，同时提供了公共 namespace 来存储公共产物，确保定制性和公共能力的按需配置；若后端存储上不存在需要获取的产物地址，则会触发从 Flutter 官方镜像做一次获取并缓存在服务端。各 BU 可按需定制引擎，并按规定的路径格式上传至自己 namespace 中，即可实现从 namespace 中获取定制版本的引擎中间产物。

## pub 仓库

类似于 Node.js 世界的 npm，Flutter 使用 pub 来管理三方组件与依赖。考虑到易用性、安全性等需求，为了管理集团内的公共二方组件，我们也搭建了内网环境的 Flutter pub 库。该库的目标是成为集团的 pub 发布平台，管理集团内所有二、三方 pub package。用户可通过设置 PUB\_HOSTED\_URL 指向内部地址，来实现通过 Flutter 工具链获取配置以及发布二方 pub 的能力，用户也可以通过 Web portal 的方式访问 pub 库并查询已发布的 pub 组件。

## 容器、中间件与 API

对于业务的接入而言，现阶段核心要解决的问题就是提供一个统一的 Flutter 运行时容器，以及一系列集团标准化移动中间件的 Flutter 封装与 API 能力，并提供集团标准的插件扩展方式供业务方独立开发业务功能。

鉴于集团应用基本上均以混合栈为主，我们将 FlutterBoost 作为 Flutter 容器混合栈的基础，并配合集团标准路由与导航中间件提供了统一的混合栈路由导航能力，业务通过标准路由注册即可快速实现 Flutter 页面和 Native 页面的混合导航能力。

容器通过对接高可用平台，提供了初始化性能埋点与 Crash 数据上报等标准监控能力，为 Flutter 业务的技术性能分析和问题排查提供了基础。集团移动端积累了一整套的标准中间件体系，包括网络库、图片库、push 消息、配置下发、数据采集与监控等一系列基础能力，在 Flutter 体系内无缝使用移动中间件能力对于业务是刚需。

同时，小程序体系建设过程中形成的一系列标准 API，也很大程度上实现了一个完整的小程序运行环境的底层能力抽象，对于 Flutter 体系标准化的访问系统能力，实现平台无关的跨端能力是个非常好的补充。

我们联合淘宝中间件团队与小程序团队，对基础中间件和小程序 API 实现做了 Flutter 侧的封装与标准化，未来也将对 Flutter 中间件和 API 能力进行系统支持。

## 标准化 Flutter 构建

由于 Flutter 研发体系较新，且构建 Pipeline 相较传统的移动构建流程又存在一定特殊性，产物构建配置复杂耗时长易出错，给 Flutter 业务的构建和发版带来了很大阻碍。因此我们也联合研发支撑部的同学，以插件的形式实现了 Flutter 脚本化的构建流程，支持双端自动整包打包和 Flutter Module 打包。目前 AliFlutter 的构建流程默认使用 AliFlutter 的 Flutter 仓库以及集团内部 pub 仓库，引擎产物也统一按配置从 artifacts 仓库获取，较好的实现了支持 Flutter 业务的自动化构建需求。

## 业务应用

在夯实 Flutter 集团共建基础之后，第二步，我们 AliFlutter 在业务应用方面也做了大量工作。一方面通过原生 Flutter 的工程化能力持续服务淘系与集团业务；另一方面通过 Flutter Canvas 项目服务了小程序场景及游戏化场景下的互动业务。

## 淘系与集团业务支撑

目前淘宝特价版已完成详情业务的 Flutter 改造并上线，采用 Flutter 使业务在需求节奏不变的情况下人力投入减少一半，对缓解业务研发压力起到了明显的作用；

同时应用的整体性能和稳定性与 Native 基本持平。后续特价版将基于 Flutter 继续拓展业务改造范围，并沉淀基于 Flutter 的业务域解决方案。

目前盒马、ICBU 、优酷也基于 AliFlutter 进行了容器接入升级与业务适配，盒马依托闲鱼的 Flutter 游戏引擎实现了盒马小镇业务，ICBU 在主链路相关页面使用了 Flutter，优酷则基于 Flutter 实现了会员订单页等场景。

同时我们也在和钉钉及 Google 一起探索 Flutter 桌面端的解决方案。

## Flutter Canvas

在电商活动营销中互动场景日益增多，对性能要求持续提升的前提下，如何提供一个高性能且稳定的 Canvas 基础能力服务好富交互的互动场景就成为了一个重点的课题。

在小程序场景中 Canvas 作为承载互动游戏的主要能力发挥了重要作用。然而受限于小程序架构下 app context 和 page context 的隔离设计，存在从 app worker 到 page renderer 的通信瓶颈，无法充分发挥出 web canvas 的性能，如果有这样一个 native 版的 canvas 实现将可直接在 native 层对接 app worker，降低通信成本，充分发挥 Canvas 的性能。

Flutter 底层基于 Skia，其性能和移动端复杂异构机型的适配性均得到过长期的检验，且 Flutter 基于浏览器的设计实现了一条平台无关的渲染管线，并对浏览器实现做了极大的简化，提供了很好的可靠性和性能。那么如果能够将这条渲染管线直接用于向业务容器提供 Canvas 能力，通过 binding 方式直接向小程序和小游戏容器提供与 Web Canvas 一致的标准 API，一方面可以复用 Flutter 的底层能力，为非 Dart 环境提供渲染支持，另一方面可以借助 Flutter 简化高效的渲染管线实现提供更好的渲染性能。

目前 Flutter Canvas 已落地手机淘宝，并在小程序运动银行业务进行了灰度试点，初步具备了承载小程序 Canvas 业务的能力；其性能在 Android 低端机上的表现有优势，可以作为 Web Canvas 方案的有益补充。

未来 Flutter Canvas 一方面将借助 Flutter 渲染管线的跨平台与高性能特点，以及 Flutter 对 Vulkan 和 Metal 的适配支持，在移动端获得更好的适配性以及性能；同时将继续实现 3D API，支撑未来互动类的业务应用。

## 未来建设

扎根业务之后，接下来的第三步，我们要紧贴 Flutter 体系在阿里集团未来的建设目标，持续回答好 Flutter 面向未来建设路径中的几个关键问题。那么首先，Flutter 体系在阿里集团的建设目标应该是什么？个人以为：Flutter 应成为阿里集团未来跨多端多平台的核心业务研发模式之一。那么，我们目前离这个目标还有多大差距？在我看来，如果要想让 Flutter 成为业务的核心研发模式，那么必须解决好跨端能力、交互能力、业务研发效率以及业务交付效率四个核心问题。

- 从跨端能力看：Flutter 虽然已具备了很好的跨多端能力与高还原度，但涉及到平台能力时，仍然需要通过各端扩展实现，还未形成小程序体系这样的标准化的容器和 API 封装能力。那么如何更好的解决 Flutter 的容器化问题，让业务不感知平台差异性？
- 从交互能力看：Flutter 如何利用好自身交互能力的优势，在提供媲美前端的富交互体验的同时，降低 Native 富交互特性开发的门槛，真正吸引 Native 开发者使用 Flutter 技术开发业务？
- 从业务研发效率看：虽然 Flutter 的 Hot Reload/Hot UI 机制已经让开发 Native 页面的效率追上了前端，但在工程解耦方面仍然有很大提升空间，目前还无法高效的支持多业务团队并行开发；另一方面如何与现今流行的 Serverless 能力结合，实现端云一体研发模式，使业务实现研发闭环，也需要实践的检验。
- 从业务交付效率看：目前 Flutter 仍属于 Native 方案，依赖端侧发版，交付效率低，无法很好的承载电商系灵活性和实效性的需求；那么如何解决 Flutter 的动态化，帮助业务实现快速迭代？

解决好这几个问题，才能真正让 Flutter 成为集团移动业务的核心研发模式，为集团业务研发带来一个飞跃性的提升。下面讲讲我们在这几个方向的思考和探索。

## 提升跨端能力：Flutter 容器化

从工程角度看，虽然 Flutter 通过 Skia 跨平台图形渲染和自建事件体系基本实现了对宿主平台的最小依赖，但对于平台侧能力，目前 Flutter 还未也没有必要从应用框架角度做到一个统一的抽象，这就需要我们根据业务的诉求和特点进行有选择的封装。小程序 API 就做了一个非常好的示范，目前阿里小程序体系提供的 API 达到了 200+，很好的对移动端的 UI、多媒体、文件缓存、网络、设备能力、数据安全以及业务相关能力进行了封装，让业务开发者在小程序侧针对 API 进行系统能力调用，无需关心平台实现。因此 AliFlutter 容器接下来的规划就是从工程体系的角度，提供一套标准化的 API 能力，以规范并抽象移动端的端基础能力，使业务尽量少甚至不关心平台差异性，专注于业务；同时借助标准化 API 的能力，实现跨多端多平台部署。

从移动端架构角度看，各个时期的跨平台方案都对 API 能力有着共同的诉求，从 H5 到 Weex，再到后面的小程序，以及 Flutter 等容器环境，进行了多轮的 API 重复建设，造成了缺少 API 接口的标准化定义，以及缺少实现层统一管控的现状。如果能够在 API 的 native 实现上做到接口统一，再通过各个容器分别提供接口供业务使用，可以更好的做到实现收口，并在统一实现层跨容器实现对系统资源的统一调度、管控和度量。

## 提升交互能力：UI + 游戏引擎

前文提到过，Flutter 目前最大的价值在于研发效率的提升，这是吸引业务团队应用 Flutter 技术的起点；但仅仅依靠研发提效还远远不够，当通过各种工程化手段解决好当前研发痛点，提升研发效率之后，如何说服业务继续使用 Flutter 体系进行业务开发？Flutter 带来的长远价值在哪里？个人认为，这个落脚点应该在通过游戏交互能力的泛化，打破 UI 与游戏引擎的边界，用游戏化的方式创造更有表现力的交

互体验，创造新的业务玩法和价值。大家知道传统的 UI 和游戏引擎是相互独立的两个体系，在 H5 应用中，往往是通过 DOM 或者上层应用框架做 UI，通过建立在 canvas 上的 H5 游戏引擎实现游戏能力。如果在游戏应用中有 UI 的需求，解决方案一般是自建一套简单的 UI 体系与事件体系，通过自绘的方式在游戏中叠加 UI，独立游戏引擎亦是如此。Flutter 从技术原理上看更像是一个建立在 Skia 图形库上的游戏引擎，它通过细粒度的 widgets 设计向上构建 UI 系统；同样得益于这样的细粒度设计，我们也完全可以直接通过 widgets 能力组合出一个完整的游戏引擎，提供 Game, Scene 及 Sprite 动效等 widgets 并扩展对应的 elements 和 render objects，并与 UI 体系共用一套事件处理机制、分层与渲染合成机制。这样做就相当于打破了原来 H5 中 DOM UI 和 Canvas 游戏的边界，让两个体系在 widgets 概念下完美融合起来，通过游戏引擎的能力赋能 UI 实现更多以前 UI 体系难以低成本实现的动效效果（比如一只小盒马一口吃掉了一个订单组件等等）。我们相信，这个方向的探索将会进一步释放 Flutter 的技术潜力，带来更多的业务可玩性与创造性，解放产品和设计的想象力，为业务创造更多价值。

## 提升研发效率：工程解耦与端云一体化

### Flutter 工程解耦

前端体系的研发效率很大程度上来自于基于 URI 的统一路由体系带来的页面间解耦性，与页面内基于 Web API 的标准化带来的内聚性。然而目前的 Flutter 研发模式，仍需要多个业务团队工作在同一个工程下，互相之间存在源码依赖，未来如果跨业务团队大规模应用 Flutter 技术，必将拖慢业务的研发效率。从工程解耦角度看，目前 AliFlutter 容器通过混合栈与标准路由能力基本实现了页面研发的解耦，未来的容器化建设通过提供小程序对等的 API 能力封装，业务对平台无感知，能够让我们有机会解耦业务研发，实现与小程序开发接近的研发体验和效率。理想的方案是：业务可以从业务维度创建一个独立的 Dart 工程，只包含业务相关的页面和逻辑代码，通过 Flutter 的 Hot UI 开发页面，通过 IDE 提供的基于 Flutter Web 的能力本地预览工程并调试 API 与系统调用，完成研发期工作；也可生成预览二维码，使用预装有

AliFlutter SDK 环境的宿主应用扫码预览；研发与构建链路分离，云端主动拉取业务仓库代码参与整包构建。以此达到类似小程序研发方式的前端研发体验，同时实现业务间的研究解耦和并行发布，提高业务的交付效率。

### 端云一体化

如今 Serverless 概念越来越多的受到业务研发的重视和应用，集团在新一代的端云一体化研发模式上的探索这一年多来也做的如火如荼。结合轻量级容器环境、多语言支持能力与统一的 API 服务端编程，端侧同学可以很容易的使用客户端语言如 Java、JS、Swift 甚至 Dart 来开发服务端业务能力，实现服务编排、服务端 FaaS 业务逻辑与 API 自动生成，达到前后端工程体系归一，业务研发闭环的效果。目前闲鱼在 Dart FaaS 云端一体化的探索走在了前面，通过集团的容器规范实现了 Dart function 容器，并联合服务端为部分业务需要的领域服务抽象出来 BaaS 层（存储、消息队列等），并封装了面向前端的 BFF（Backend for Frontend）能力层，使移动端开发者可以很容易的使用 Dart 封装 FaaS 业务逻辑，同时进行移动端和服务端 FaaS 开发，大大提高了业务研发效率。通过将原有的端侧请求接口、组装数据并转换为 ViewModel 的逻辑都后移到了服务端，经过字段映射与页面编排，移动端可直接获取 ViewModel 并刷新页面，通过 BinderAction 双向交互状态数据，有效屏蔽了通信细节，提高了开发效率。云端一体化除了带来了研发与协同效率的提升，同时也重塑了生产关系，使端侧业务从只关注端侧体验和逻辑的开发角色，变成能够向业务整体结果负责；同时也让服务端更专注于领域服务的沉淀。Flutter 良好的跨端特性，能够屏蔽掉端差异化，配合 Flutter 容器化改造，更进一步的简化了业务的全链路研发模式。未来如何在 FaaS 模式下，沉淀出一套可以服务集团业务研发的通用端侧和服务端通信调度框架，让集团 Flutter 开发者和业务都能共享到 Serverless 技术和云端一体化提效的红利，是需要我们共同去探索和定义的新问题。

### 提升交付效率：Flutter 动态化

实现动态化是交付效率提升的重要方式。对于电商强运营强时效性特性来说，动

态化几乎是一个必备的诉求，但从技术上看也是一个非常敏感的需求，是一个在系统厂商平台管控下长期博弈的过程。在我看来，动态化技术需要解决的核心问题，是在保证业务发布确定性的前提下，寻求技术性能、业务迭代效率与灵活性三者之间合理的平衡点。下面讲讲我们在 Flutter 动态化上的两个思路与尝试：Flutter 模板化方案与 Web on Flutter。Flutter 模板化方案动态模板化方案是集团内较为成熟的一套基于 Native 技术的模板化方案，专注于 UI 模板渲染，没有执行逻辑和运行时环境，目前被广泛应用于电商系的一些核心业务场景。同时该方案提供了配套的组件平台，支持在线模板编辑、预览、测试及发布整套流程，结合发布平台形成了一整套完善的服务开发生态闭环。在 Flutter 体系下，目前闲鱼团队依据标准模板协议在 Flutter 侧实现了一套 Dart 版 SDK，通过模板下发实现了 Flutter 端的轻量级动态化组件编排能力；并通过一年多的迭代逐步解决了渲染性能与渲染一致性问题，较好的赋能了 Flutter 业务的组件动态化能力。那么，未来 Flutter 与动态模板化方案有没有更好的结合点？答案是肯定的。从 DSL 角度看，目前模板的写法基本上来自 Android XML，对组件开发者尤其是非 Android 开发者不够友好，具备一定的学习成本；而 Flutter 的结构与属性均可通过 widgets 表达，可以极为灵活且平台中立的用声明式代码构建 UI 并绑定数据，易于开发者编写组件并通过 Flutter 框架独立调试与测试；在移动端运行时，可以按需按场景翻译成 native 组件或 Flutter 组件。未来我们也将持续在这个方向上做探索。

## Web on Flutter

相比贴近 Native 研发模式的模板组件化渲染方案，Web on Flutter 希望通过类 H5 的 DSL + JS，借助 Flutter 的渲染能力，实现贴前端技术栈的动态化能力。目前基于 Web 渲染的小程序方案存在启动耗时高，渲染性能距原生 UI 有较大差距等性能问题，这些问题很大程度上都源自于浏览器引擎的设计历史包袱（渲染管线复杂、CSS multi-pass layout 及 legacy 实现等），以及 JS 到 Native 通信效率低（bridge）。Flutter 的设计思路源自浏览器，一方面直接吸收了前端框架近年来的演进成就，原生支持了声明式 – 响应式编程范式，提高了移动端的研发效率；另一方面，

Flutter 紧贴 native 开发模式有限定义了 UI 结构、布局与渲染的必要元素，在满足 native UI 开发模式的前提下简化了能力定义与布局算法 (single-pass layout 与 repaint boundary 等概念)，很大程度上简化了渲染管线的复杂度，直接为 Flutter 带来了接近原生的性能体验。同时，Flutter 的 widgets 设计巧妙，结构布局属性等基础元素均使用 widgets 表达，且可通过基础 widgets 的组合来构成复杂组件，这种细粒度 + 组合能力设计让 Flutter 有极强的表现力，并具备向上对接各种研发模式的可能性。因此，通过 widgets 组合小程序 DSL，支持小程序 CSS 有限集，实现渲染层替换浏览器引擎，并对接 JS 引擎支持 JS 执行能力，是一个将 Flutter 应用于小程序生态的合理探索方向。相比把开发者惯坏了的浏览器，这种方案在 CSS 的能力支持上必然是受限的，无法满足所有 CSS3 标准的实现，更多通过紧贴 Flutter widgets 的现有能力以及必要的 widgets 扩展，在不破坏 single-pass layout 的前提下组合出 CSS 能力。但从 Flutter 原生开发的角度看，只要 Flutter 现有的原生能力能够满足业务需求，那么受限的 CSS 实现也一样可以提供和 Flutter 对等的能力解决业务问题。同时，通过受限的 CSS 可以换来与 Flutter 相当的高性能，与基于 Web 的实现相比，在性能上带来了质变。

我们在 18 年底通过一个内部项目实现了这个思路的原型，通过使用 C++ 重写 Flutter 的 widgets、rendering、painting 及事件管理等 Dart framework 中的中低层能力，并在 widget 上用 C++ 实现了 CSSOM + DSL  $\rightarrow$  Widgets 的响应式框架，直接从 C++ 层提供 render 实现，将传统由 JS 承担的模板展开、tree-diff 计算与渲染工作交给 C++ 层，通过 Flutter 提供的 Widgets tree 到 RenderObjects tree 的 diff 能力实现，显著提高了性能。

从实现的简单的 demo 看，相对小程序的 web 渲染性能有了大幅的提升。这种方案的问题在于和 Google 代码库分裂后的长期维护性。“[Flutter 和 Web 生态如何对接](#)”这篇文章对集团内在这个方向上尝试的几种思路做了较为全面的对比，未来我们也将继续在这个方向上做深入和持续的探索。

## 总结与展望

Flutter 体系化建设目前在淘系刚刚起步，仍然有大量工作需要去做，我们在基础设施、工程化以及通过 Flutter 定义和收敛业务域研发模式上做的许多建设性的事情，正朝着把 Flutter 打造为统一移动应用基础研发框架，助力业务回归移动端研发模式这个大目标一点点迈进。在移动技术小组启动 AliFlutter 项目之前，闲鱼技术部已经在 Flutter 技术建设上做了大量探索和投入。一方面通过 Flutter 技术赋能业务提效升级，拿到了很好的业务成绩；另一方面沉淀 Flutter 的技术与业务解决方案，并通过开源反哺社区，在海内外 Flutter 技术社区中建立了显著的技术影响力与领导力，也涌现出一众 Flutter 技术专家。接下来 AliFlutter 的重点任务，就是要和闲鱼、财富等先驱应用者以及盒马、钉钉、飞猪、优酷、饿了么、CBU 等 Flutter 的实践者一起，在集团层面把 Flutter 的场子建起来，把集团 Flutter 生态拉起来，让技术和经验能够共同沉淀和分享，一起来把 Flutter 技术体系在阿里的应用生态内做大做强，真正成为集团业务的核心研发模式，并让每个参与者都能从中受益。我们一直坚信 Flutter 技术的先进性和应用前景，未来我们将继续立足淘系服务集团业务，和集团开发者携手前行，在 Flutter 这个技术方向上坚定的走下去。

## AliFlutter 客户端研发体系概览

**摘要:** Flutter 是开源的 UI 工具包，其能够帮助开发者通过一套代码库高效构建多平台精美应用，支持移动、Web、桌面和嵌入式平台。Flutter 组件采用现代响应式框架构建，中心思想是用组件 (widget) 构建 UI。在 AliFlutter 系列第一场直播中，淘宝终端技术部无线技术专家王康从 Flutter 的原理出发，介绍了 Flutter 的原理、业内现状，以及阿里巴巴在 Flutter 上所作的深度实践和探索。

内容根据演讲视频以及 PPT 整理而成。

观看回放 <http://mudu.tv/watch/5466337>

### 演讲嘉宾简介：

王康(花名：正物)，淘宝终端技术部无线技术专家，Flutter Member(kangwang1988)，AspectD 作者。负责了 Flutter 在闲鱼中的混合开发体系建设与业务落地，做了一系列相关技术方案。在 Flutter 原理与应用、多端一体化编程方面有丰富的实践经验。目前专注于以 Flutter 为核心的多端多平台研发模式的探索与实践。

本次分享主要分为以下四个方面：

一、Flutter 原理

二、Flutter 业内现状

三、AliFlutter 建设与深度实践

四、AliFlutter 研发模式探索

## 一、Flutter 原理

Flutter 主要有四个特点：美观、高效、高性能、开放。

- 美观：**Flutter 提供了丰富的 Widget，比如动画、手势等。Flutter 采用了组合式 API 模式，因此为 UI 创建带来了更强的灵活性。此外，Flutter 使用了游戏引擎的方式来写 APP，使得用户可以具有很强的灵活性，能够在像素级别进行控制。
- 高效：**Flutter 类似于安卓小系统，使得其能够使用一套代码运行在各种各样的平台之上。此外，在 Debug 模式下还支持热重载，使其能够达到高效的研发效率。
- 高性能：**在 Release 模式下，Flutter 是预先编译成机器码，执行期具有高性能。
- 开放：**Flutter 是一个开源的项目，基本所有工作都可以在 GitHub 上看到。



以上四个特点的背后就是 Flutter 的原理。首先，Flutter 架构在 OS 之上，最底下是与平台相关的 Embedder 层，其主要负责的工作是 Surface、Thread 以及 Event Loop。在 Embedder 层之上是 Engine，主要包括三部分，Dart Runtime；

负责将 UI 绘制到 Surface 上的 Skia，负责文本绘制的 Text。在 Engine 之上就是大家所熟知的 Dart 的 Framework。基于上述这些，开发者即可开发应用。

## 阿里巴巴为什么选择 Flutter

在阿里巴巴的电商场景下，往往有一些非常重要的考量，比如用户体验的要求，对于研发效率的要求，以及如何消除多端之间的差异。在阿里经济体里面，应用所需要部署的目标设备是非常多元的，不仅有常见的 iOS 和 Android，还有钉钉等桌面场景、天猫精灵等 IoT 场景以及各种线下大屏的场景。当前，流量增长红利不断减少，需要更多创新玩法为消费者提供更好的用户体验，这就产生了富交互游戏化的需求。Flutter 具有的高性能，高研发效率、跨端一致性，多端多平台支持，以及丰富的表达力使其可以解决这些痛点。

### Ali为什么选择Flutter



## 二、Flutter 业内现状

下图展示了 Flutter 在国内一些主流大厂中的实践。在阿里巴巴内部，目前有十几个 BU 的几十个产品正在使用 Flutter，典型的包括淘宝、闲鱼、UC 以及优酷等。在业内，腾讯的微信、Now 直播、翻译君等，字节跳动的西瓜视频、皮皮虾，快手的快影，美团的美团骑手、美团外卖商家版、美团众包以及百度的贴吧等也都应用了 Flutter。

## Flutter业内现状



- **Alibaba**
  - 淘宝、淘宝特价版、蚂蚁财富、闲鱼、UC、优酷、ICBU、饿了么、考拉、飞猪等
- **Tencent**
  - 微信、腾讯Now直播、腾讯翻译君、腾讯企鹅辅导等
- **字节跳动**
  - 西瓜视频、皮皮虾、懂车帝等
- **快手**
  - 快影等
- **美团**
  - 美团骑手、美团外卖商家版、美团众包等
- **百度**
  - 贴吧等

Flutter 在业内的实践现状主要围绕着体系化、深度、框架以及更多探索这些维度展开。在体系化方面，需要做一些基础设施的建设，这是因为 Flutter 以及 Dart 的很多东西还不成熟。使用 Flutter 解决业务上线问题，需要考虑研发体系的构建。应用上线之后，需要监控各种指标。在深度方面，往往关注引擎大小、包大小、内存优化以及启动时间等。除了上述两部分之外，业内也有很多框架相关的工作，比如混合栈框架、状态管理、动态化 UI、AOP 框架以及生态插件等。此外还有原生 Flutter 以外的一些探索，比如小程序渲染和前后端一体化等实践。

## Flutter业内现状



### 体系化

基础建设  
研发体系  
监控体系  
DevOps  
...



### 深度

引擎大小  
包大小  
内存优化  
启动时间  
...



### 框架

混合栈  
状态管理  
动态化UI  
AOP框架  
生态插件  
...



### 更多探索

小程序渲染  
前后端一体化  
...

## 三、AliFlutter 建设与深度实践

### AliFlutter 业务实践

下图选取了阿里经济体中一些应用了 Flutter 的典型场景。比如宝贝详情是一个业务逻辑非常复杂的页面，属于图文互排的页面，并且具有大量图片，有时还包括一个视频播放器，在这个场景下就全部应用了 Flutter。有团队使用 Flutter 框架用于游戏业务的开发，比如下图所示的是盒马使用 Flutter 构建的一个游戏页面。此外，在 Alibaba 这一应用中，也大量使用 Flutter 用于构建主链路以及订单页面等。



之所以选择 Flutter，有几个典型原因。首先，HotReload 和跨端一致性使得研发非常高效；其次，可用于游戏化的丰富 UI 表达力、动画、图文混排性能等诉求都能被 Flutter 很好地满足。

### AliFlutter 业务研发模型

在业务场景的背后是 AliFlutter 的业务研发模型。其实，Flutter 本身主要解决两个问题，逻辑和 UI。其本身没有各种 Native 能力，需要为其补齐如网络、推送以及接入网关等，使其更加接近于业务开发容器，而不仅仅是 UI 开发框架。再上面就是应用开发框架，比如状态管理框架、游戏化框架、动态化 UI 以及组件库，在此之

上就可以构建一些业务了。除此之外还会涉及到一些研发协同方面的工作，比如打包构建、Linter、Pub 库等。

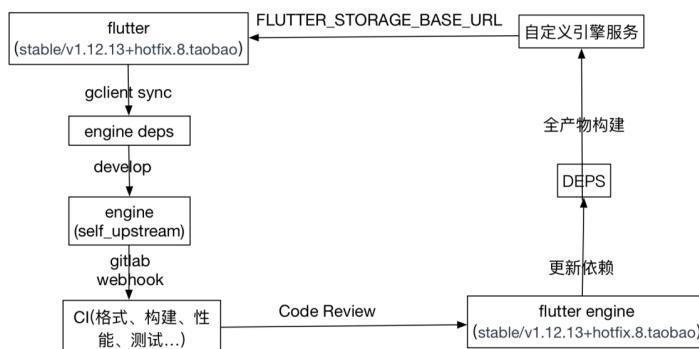
## AliFlutter业务研发模型



## AliFlutter引擎研发模型

在 AliFlutter 之下，存在很多引擎修改的场景。举例而言，在 iOS13 以下可能存在一些后台 GPU 渲染 Crash 的问题，在 Android 上存在特别机型 Flutter 启动闪退的问题。此外，还需要考虑如何让 Flutter 和目前已有生态进行融合，比如图片库、网络库等在阿里内部都有比较好的实践。无论是 Bug 修复还是 Native 能力提升，其实都是对于 Flutter 引擎所做的定制化工作。

## AliFlutter引擎研发模型



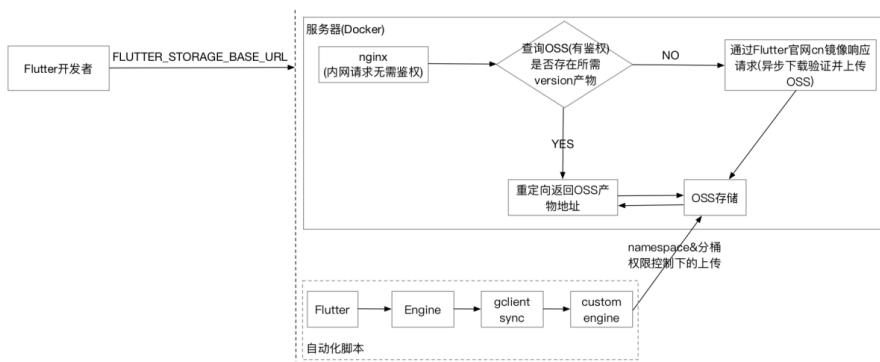
上图展示的就是在阿里内部对于 Flutter 引擎进行定制所做工作的逻辑，首先通过 Flutter 框架获取对应的引擎代码，拉取依赖，进行开发，到 Gitlab 做 CI，代码审核完成之后将产物构建出来上升到服务上面，最终通过简单的方式来提供服务。

## AliFlutter 基础设施建设

### (1) 自定义引擎服务

前面所提到的是自定义 Flutter 引擎的开发流程，而想要将开发完成的东西提供给其他人使用，就需要如图所示的自定义引擎服务了。对于 Flutter 开发者而言，只需设置一个环境变量去服务上查询是否有对应的产物即可，如果有的话，就做一些定制并返回给开发者；如果没有则去官方上游拉取。当然了，对于 Flutter 的基础设施而言，需要有一些多团队的支持，比如 Namespace 等机制。最早的时候，阿里巴巴通过 Git 方式管理自定义引擎，但是 Git 对于二进制很不友好，所以就使用了高效自定义引擎服务来解决问题。

## AliFlutter基础设施建设

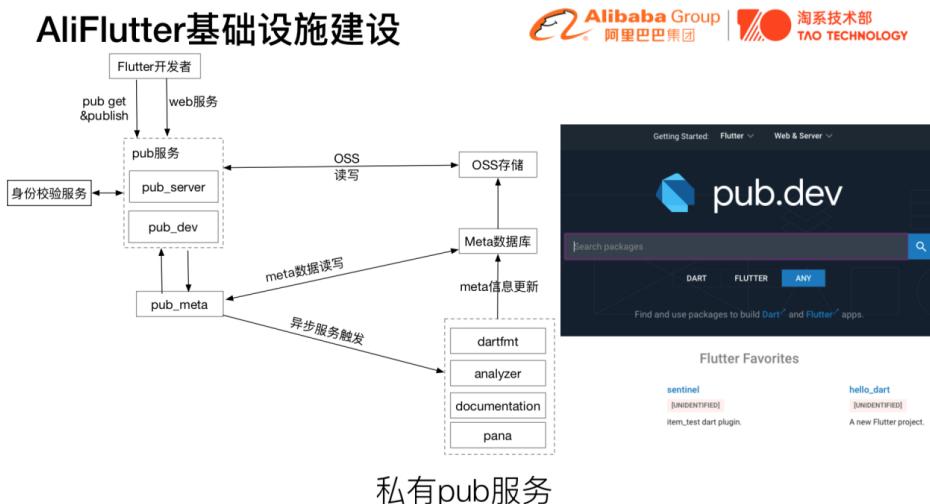


自定义引擎服务

### (2) 私有 Pub 服务

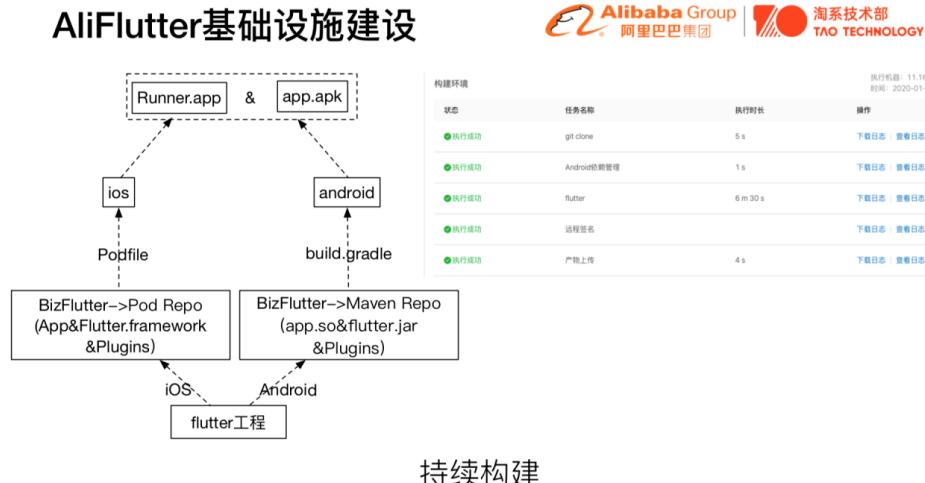
除此之外，AliFlutter 还实现了私有 Pub 服务。最初的想法是将不同人开发的

库等工作归类组织起来，建设更好的内部生态，实现更好的复用。Pub 本身就是 Flutter 所提供的开源框架，但是其深度绑定了谷歌云服务，所以在做这部分的时候需要将对于谷歌云的依赖变成对于阿里内部的依赖。主要工作分为两部分，一部分是对于包的简单管理和存储，这部分可以通过阿里云存储 OSS 实现；还有一部分是监控包的下载量以及健康程度等，这部分还部署了 Meta 数据库服务，在将包上传的时候将数据同步过去，以及面向使用者的前端服务。



### (3) 持续构建

这部分的主要工作就是如何将所写的 Flutter 代码提供给没有 Flutter 环境专门用于打包平台。Flutter 工程可以通过一些脚本构建出一个 Pod 或者 Gradle，进而集成起来变成一个 APP。

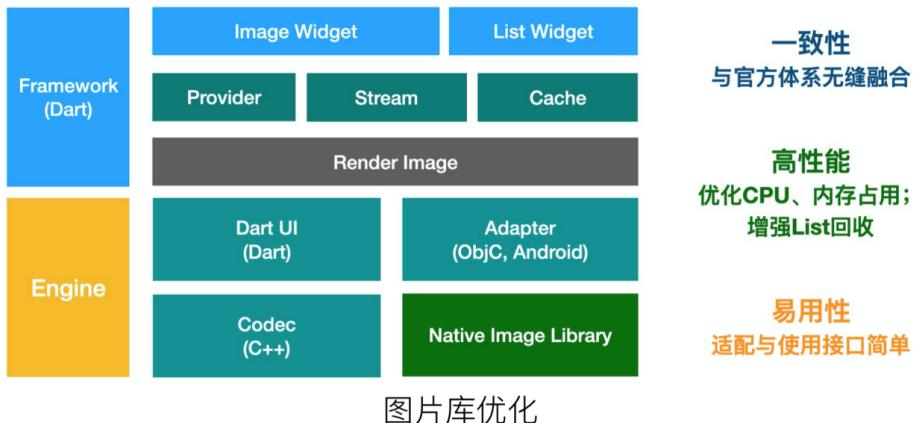


## AliFlutter 深度实践

### (1) 图片库优化

在阿里巴巴内部，除了对于 Flutter 做的一些体系化实践，还有不少深度化实践。比如对于图片库的优化，对于 Flutter 而言，本身的图片库存在一些问题，比如内存占用高，不释放问题、CPU 占用问题。为了尽可能遵循 Flutter 原生的图片库逻辑，做了图片库的优化。主要工作包括对于 Flutter 框架的整体修改，能够较好地实现一致性，与官方体系无缝融合，对接内部图片库，其在性能以及易用性上面也具有较好的表现。

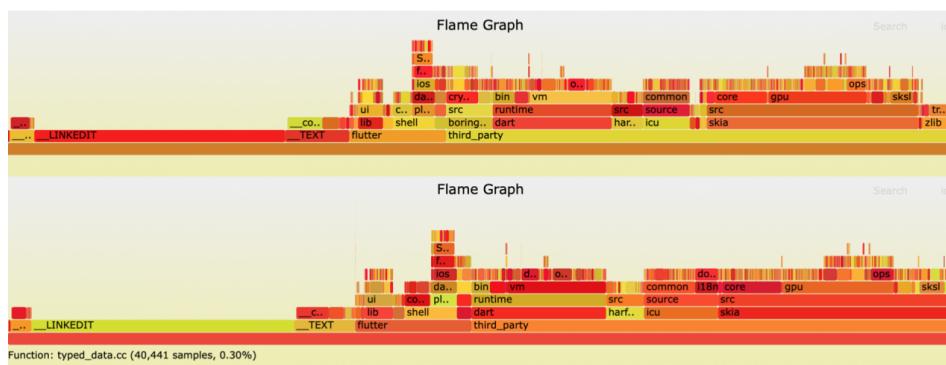
## AliFlutter深度实践



### (2) 引擎大小优化

我们在 Flutter 引擎大小优化方面也做了不少工作，简单介绍对于库的裁剪。如下所示的两张火焰图，其较好地表达了 Flutter 引擎所依赖的各个库裁剪前后的比例对比。裁剪后的内容既保证了功能的完备性，也显著降低了引擎大小。

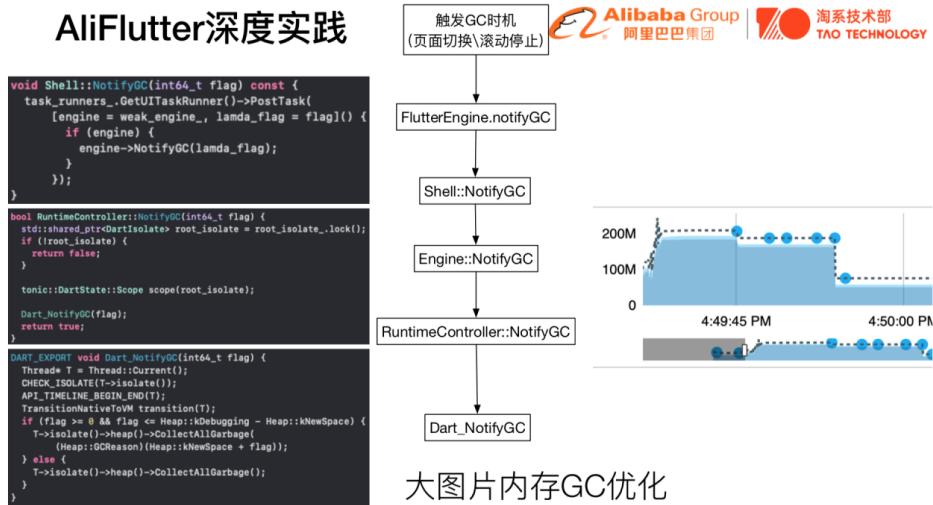
## AliFlutter深度实践



引擎大小优化

### (3) 大图片内存 GC 优化

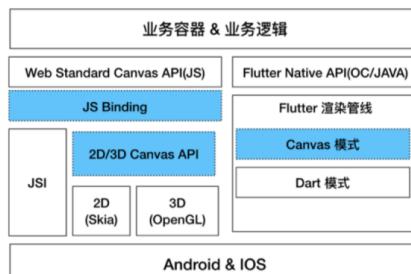
除了前面所提到的类似于音频图片释放等内容之外，阿里在实践的过程中发现 Flutter 在大图片内存 GC 方面存在一些问题，比如在用户退出的时候内存无法得到很好释放。对于社区中使用 Flutter 的同学而言，面对这样的问题建议大家在 Profile 模型下看下点击了 GC 按钮是否能够将内存降低下来。基于此逻辑我们提供了一套供上层业务使用的 GC API。从 FlutterEngine 开始依次调用 Shell、Engine、RuntimeController 以及 Dart 的 NotifyGC。



### (4) Flutter Canvas 实践

Flutter 包含了 Skia，可提供 Canvas 能力。之前的逻辑是通过 Dart 调用 Engine，再调到 Skia，而我们通过实践中对其部分 API 的暴露，将 Skia 的 Canvas 能力加以透出。在 JS 部分有一些 2D 和 3D 的 API，可以将这些暴露成为 Canvas API，整体而言，相比于 Web 的 Pipeline 表现非常不错，这一功能目前已经在部分业务开始灰度测试，数据表现也非常不错。

## AliFlutter深度实践



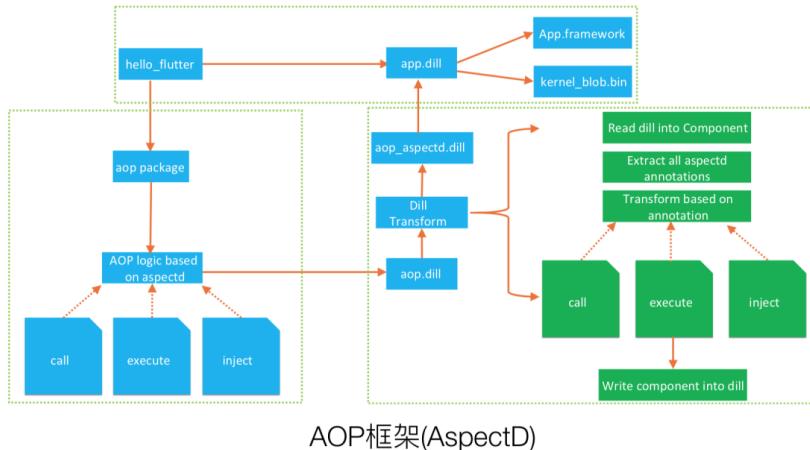
Flutter Canvas实践

### (5) AOP 框架 (AspectD)

Flutter 的 AOP 框架的核心基于 dll 变换，dll 本身是从 Dart 代码到最终代码之间的中间语言表达。其原理简要来说是当我们写了一个 hello\_fultter 的时候，再写一个 AOP 包，AOP 的包会包裹 hello\_fultter 包，使得在 AOP 包的产物 (dll) 里面 hello\_flutter 和 AOP 的逻辑都存在，即其包括两部分内容，hello\_flutter 本身代码的 dll 表达，以及 AOP 框架中写的注解的 dll 表达，将这两者都包含在 app.dll 里面，基于此我们可以通过 dll transform 方式来做变换。这里比较复杂，需要考虑 AST 抽象语法树的各种逻辑。需要将注解提取出来并基于这些注解进行操作，并最终写入到 dll 里面去，这些操作处理完成之后，就变成了 aop\_aspectd.dll，替换掉之前的 app.dll 即可。

## AliFlutter深度实践

Alibaba Group 阿里巴巴集团 | 淘系技术部 TAO TECHNOLOGY

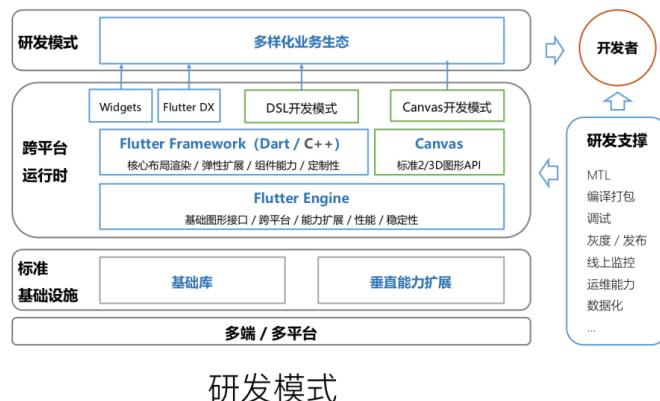


## 四、AliFlutter 研发模式探索

### AliFlutter 研发模式

接下来分享阿里巴巴在 Flutter 研发模式上的一些探索。下图中最重要的就是研发模式和跨平台运行时，目标是提供一种多端多平台的能力。在平台底层是基础库、网络库的基础能力，此外还需要在垂直能力上的扩展，支持各种垂直的基础能力。基础设施之上是 Flutter 的跨平台运行时，运行时基于 Flutter Engine，提供了具有丰富表达力的图形接口、跨平台、能力拓展、性能以及稳定性等。在此之上，Flutter Framework 提供了可以复用的基础能力，比如核心布局渲染、弹性扩展能力、组件能力以及定制性等。除此之外，也有一些研发支撑上面的工作，比如编译打包、调试、灰度发布、线上监控、运维能力以及数据化等。

## AliFlutter研发模式



## 研发模式

## AliFlutter 研发模式展望

在 Flutter 的未来发展方向，阿里巴巴主要的工作将集中于以下四点：

- **跨端能力：**我们考虑对于上层的各种平台提供标准基础能力并 API 化，从而更好在多端多平台进行部署。此外，还希望通过 Flutter 的容器化，使得研发和业务方能够更多地专注在自身业务上面去。
- **交互能力：**我们考虑利用 Flutter 丰富的表达能力在游戏化方向进行更好的扩展，以游戏引擎的方式来开发 APP。基于泛化的交互能力以及更多的可玩性和创新性能够为业务带来更多可能。
- **研发效率：**我们考虑实现工程解耦和云端一体化，目标是业务方只需关注所写的包，通过很简洁的方式集成进来并看到效果，从而提供类似于前端的开发体验。此外通过云端一体从面向端侧负责转变到面向业务整体负责。
- **交付效率：**这部分主要包含两部分，一部分是动态化 UI，另外一部分是 Web On Flutter，期望通过提供更加灵活的动态性，以及前端技术栈下的动态化能力。

## AliFlutter研发模式展望



## 总结

在本文中，首先，为大家分享了 Flutter 的原理，介绍了 Flutter 美观、高效、高性能、开放的特点，以及阿里巴巴为什么选择 Flutter。其次，为大家分享了 Flutter 的业内现状，有大量投入的主流厂商，以及体系化、深度、框架和更多的探索。再次，为大家介绍了 AliFlutter 的建设与实践，包括了业务、研发模型、引擎研发等方面实践。最后，展望了对于 AliFlutter 研发模式的考量和未来发展方向。

由阿里云开发者社区志愿者贾子甲整理。

# 闲鱼研发框架应用和探索

**摘要:** Flutter 是开源的 UI 工具包，其能够帮助开发者通过一套代码库高效构建多平台精美应用，支持移动、Web、桌面和嵌入式平台。在 AliFlutter 系列第二场直播中，阿里巴巴闲鱼无线技术专家梁治峰为大家分享了闲鱼在 Flutter 中研发框架应用和探索，从分别从三个方向介绍 Flutter 一体化研发模式、Flutter 动态化能力、Flutter 互动能。

## 演讲嘉宾简介：

梁治峰（花名：玄川），阿里巴巴无线技术专家，闲鱼买家链路客户端负责人，主导闲鱼 Flutter 化的落地和研发框架演进。

本文内容根据演讲视频以及 PPT 整理而成。

观看回放 <http://mudu.tv/watch/5466337>

本次分享的主题主要包括以下五个方面：

一、闲鱼 Flutter 研发框架使用现状

二、Flutter 研发框架下一代模式

三、Flutter 研发框架下动态能力

四、Flutter 研发框架下互动能

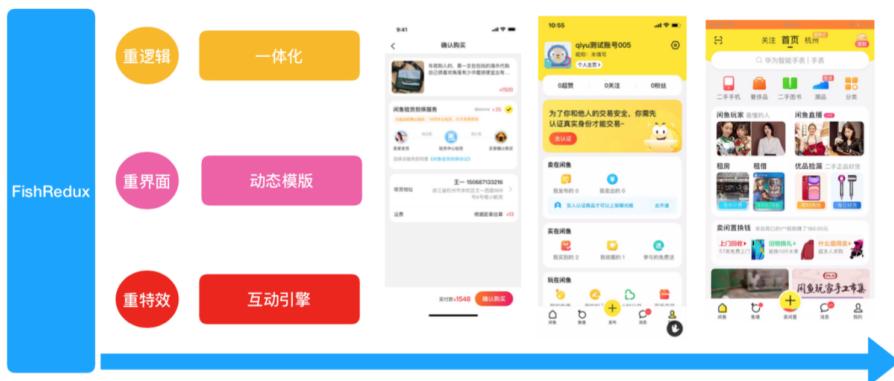
五、后续规划和展望

## 一、闲鱼 Flutter 研发框架使用现状

闲鱼是一个侧重于电商业务的平台，因此随着业务的不断增长，系统的逻辑复杂度也在不断提升。因为属于电商业务，所以对于流量和运营的数据具有较高的需要，因此在闲鱼的体系中也需要具备动态性的能力，并且还需要通过增加特效的能力来增加用户的感知，丰富用户的体验。

### 闲鱼Flutter研发框架使用现状

Alibaba Group | TAO TECHNOLOGY



## 二、Flutter 研发框架下一代模式

### 一体化模式

下图中左侧是传统的客户端 – 服务器架构。在这样的CS架构下，对于客户端开发者而言，往往都会经历相似的痛点。当产品的需求过来，可能客户端的开发同学并不能自己完成，而需要牵扯到服务端的开发，可能需要对于协议进行补充或者添加更多的接口能力。而对于后端开发同学而言，面对一个需求也可能需要领域服务的支持。这样一来，一个貌似很简单的需求，却需要从客户端到后端再到领域服务的相互协调，进而会影响需求的排期问题。而如果客户端也可以写服务端的代码，这样的问题是否就能够被解决掉呢？

## 一体化模式

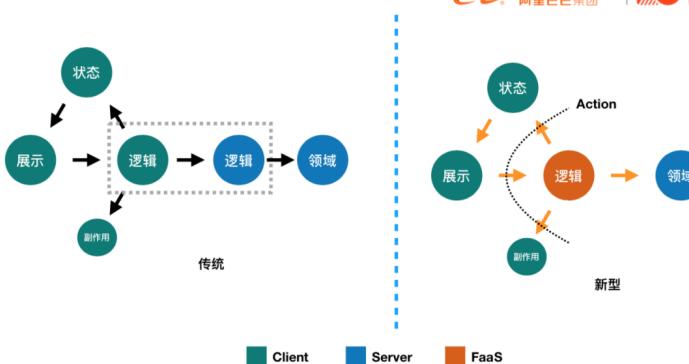
Alibaba Group | 淘系技术部 TAO TECHNOLOGY



在目前闲鱼所给予的 FaaS 框架下的一些场景中也存在上述痛点。如下图所示的是传统基于 FaaS 的模式，可以看出使用 FaaS 能够将逻辑和 UI 彻底进行分离，但是在端上的逻辑部分，无外乎两种，一种是数据的拉去和推送，另外一种是数据的主账号。在后端也会有类似的逻辑，只不过此时不是客户端找服务端要数据，而是服务端找各个领域层要所需要的数据，然后进行数据的加工，再将数据以面向客户端协议的部分进行主账号推送。而上述两个部分存在着一定的逻辑割裂，并且也存在一定的重复工作，因为数据转化被执行了两次。那么，是否能够将上述两种逻辑合二为一，并且让端上的同学进行开发，同时将逻辑后端化呢？结合如今 Serverless 的能力，可以做到在轻量级能力下支持多语言的开发。

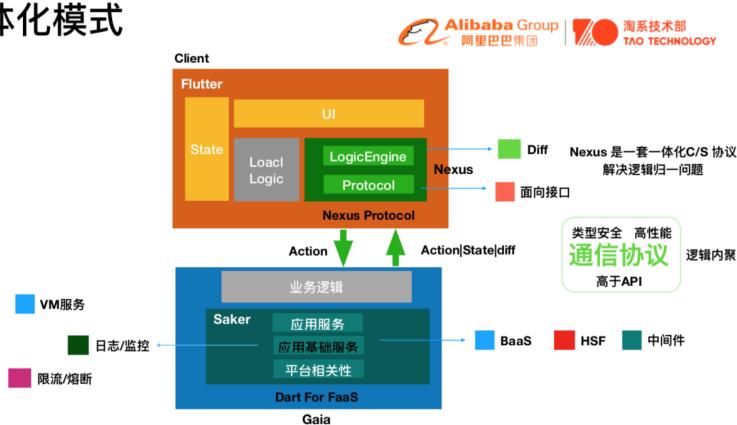
## 一体化模式

Alibaba Group | 淘系技术部 TAO TECHNOLOGY



基于上述的背景，闲鱼在今年实现了一体化的研发解决方案。在云侧兼容了集团通用的 Gaia 容器化能力，用 Dart 语言实现了容器化的部分。之所以使用 Dart 是因为这部分与 Flutter 对应。阿里巴巴希望客户端写后端的情况与 Flutter 使用一体化的语言来完成，屏蔽两者之间的差异。在端侧研发了 Nexus 一体化插件，将现在面向 Action 的部分可以实现端侧与云侧的一体化。这样的好处在于在端侧叫 Action，在云侧也叫 Action，而在端上进行开发的时候并没有感知云侧 Action 的存在，这就是 Nexus 的核心作用。此外，现在面向于通信协议其实就是面向于 API 接口的一部分，这样能够实现端上的高性能和逻辑内聚。

## 一体化模式



这里简单介绍一下一体化框架的具体落地场景。对于下图所示的闲鱼下单的页面而言，在原有模式下可能需要 5 个请求接口，这部分请求接口可能部分在端上，部分在云上，并且通过一条信息流进行合并。这种情况下如果需要修改某种状态就会非常复杂。在改造完成之后就将原来的 5 个请求接口全部实现 Action 协议化，这样的好处在于云端的模型统一了，无论是对于云还是客户端都在写同样的逻辑，只不过这样的逻辑部署到了云上。其次，还屏蔽掉了协议的具体部分，只留下了协议名称。第三点好处在于实现了逻辑的归一，所有的逻辑都实现了云端化，大家在书写这样的逻辑时不会存在割裂，最终书写的逻辑都是面向云模型的状态。第四个优点是冗余代码将会大大减少。而最大的好处在于形成了很好的业务的闭环，让客户端开发也可以应用开发的部分工作。

## 一体化模式



## 三、Flutter 研发框架下动态能力

闲鱼本质上主要是一个电商业务平台，其在于流量侧具有强运营时效的特性，很多的运营活动或者决策需要得到及时的响应，如果在这种情况下不具有动态性就会陷入被动。完整的动态性包括了逻辑动态性和 UI 动态性，但是在流量侧部分更加注重 UI 动态性，轻逻辑重 UI。接下来将与大家分享在 Flutter 侧如何使用这样一个微能力的解决方案。



电商业务具有强运营强时效性特性

## 动态模板

动态模板在阿里巴巴整个集团内部都是一套比较成熟的解决方案。首先，通过 DX 平台编辑模板，编辑成二进制文件并生成模板下载链接，之后模板下载解压，进行表达式或者事件的注册，并对于数据进行绑定解析，使得组件得到渲染。借助于集团动态模板的成熟方案，所需要解决的就是在 Flutter 侧如何满足 DSL 的 UI 表达，来实现 UI 布局。

## 动态模版

Alibaba Group | TAO TECHNOLOGY

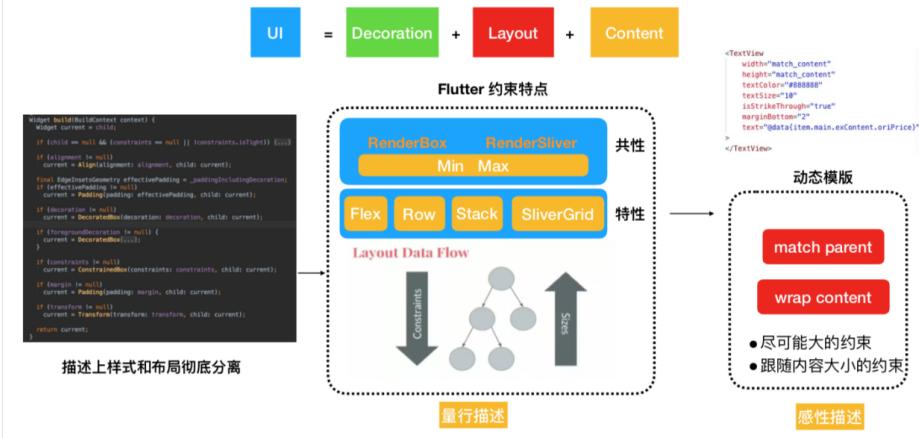


动态模板化是集团内较为成熟的一套基于Native技术的模板化方案，专注于 UI 模板渲染

## 核心问题 -Layout

熟悉安卓或者 Flutter 的人会发现这两部分的 UI 表达其实是格格不入的，那么如何在 Flutter 侧实现一套安卓 UI 布局呢？其实完整的 UI 表达是样式 + 布局 + 内容组合实现的。根据 Flutter 的源码可以看出，在其布局表达里面，样式、布局、内容三个要素表达是彻底分离的。相反而言，在安卓的 DSL 的架构里面，样式和布局是结合的，内容部分是分离的。如果将安卓的部分也进行拆分可以一一映射到 Flutter 中，虽然描述部分可以很容易地做映射，但是核心困难在于布局部分，主要是关于大小的抽象性描述，因此需要了解在 Flutter 侧是如何表达布局的约束的。

# 核心问题-Layout



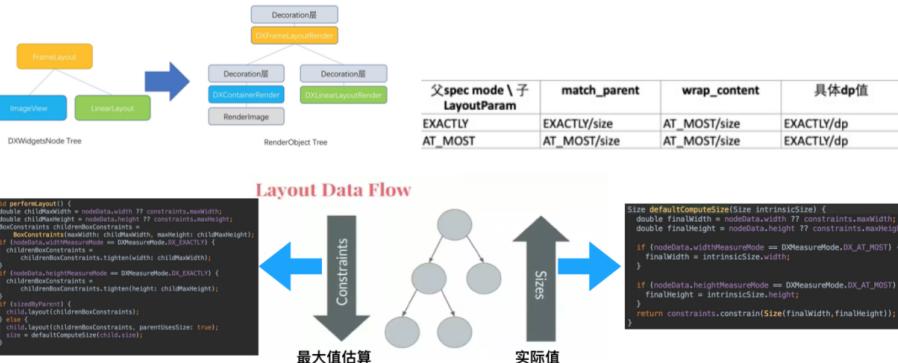
其实在 Flutter 侧主要有两种对于布局的约束设计，分别是盒子模型和条子模型，而以上两种都是感性描述的约束性布局。除此之外，还提供了 30 多个布局的容器部分。这是因为基于上面的感性描述的约束布局情况下，Flutter 可能会存在大量的冗余代码，在约束布局情况下就会显得特别复杂。另外一部分在于性能部分，感性描述远远没有大于量行描述，因此 Flutter 提供的 30 多个量行约束是对于性能的考究。反观安卓的布局部分，相对比较少，大约为 4、5 个，所以这里的问题就是如何将安卓的布局部分使用 Flutter 的布局来表达或者描述。如果想要使用特性来做映射是很困难的，如果退而求其次，使用共性部分的盒子模型和条子模型似乎可以表达。

## 布局表达

如果在端侧已经完成对于动态模板树形结构的解析之后，就能够很容易地将树形结构的节点实现如下图所示的一拆三结构。第一层是装饰层结构，中间层可以基于自低向上和自顶向下的计算规则重新计算出大小，最后一部分则是将内容想要表达的叶子界面进行 Backup。为了实现安卓这样的布局结构阿里巴巴引入了安卓的 Spec Model 模型，其很主要的作用就是表达当需要做依赖于内容适配等情况下，可以使用 Min\_width 的最大估算来预估大小部分，再从自底向上来计算出实际的 Size。动态

模板在 Flutter 侧的实现主要解决的就是这样的侧重点部分。

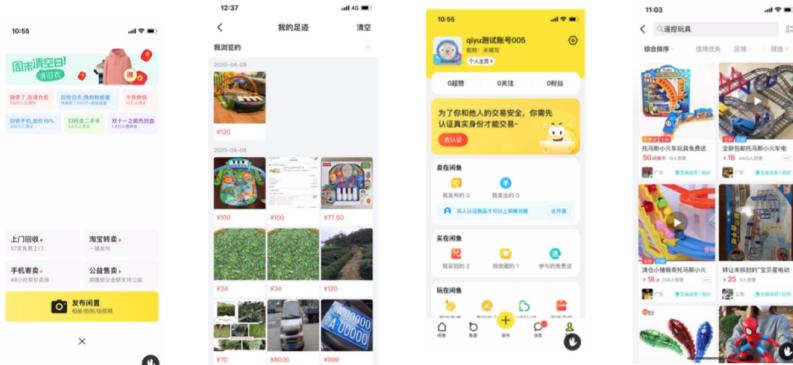
## 布局表达



## 业务效果

整套方案经过闲鱼一整年的打磨之后，已经有大量的业务上线和应用了。无论是卡片还是其他布局部分，都能够使用 Flutter UI 实现。

## 业务效果



## 性能参考

动态性部分往往会和性能存在一定的博弈。在闲鱼的实践中得到的实际结果表明，使用动态模板的 DSL 来表达的性能还可以接受，线上的实际效果大约在 55 帧左右，相比于正常使用 Flutter 原生的 60 帧仅仅存在可被接受的一点差距。

## 性能参考



## 性能的下一步探索

虽然目前的方案和 Flutter 原生仅存在 5 帧的差距，但是如果能够进一步优化，还是有可能达到原生的性能要求的。下图中分别展现了使用 Flutter 原生和 DX 写的卡片布局，可以直观地发现在 Flutter 原生使用了大量的高阶型特性表达，在 DX 中则基本都是常见的容器布局，并且树形结构的深度层次远远大于 Flutter 原生。DX 中使得长度变大的部分在于装饰性的布局部分，因此可以尝试地探索在 DSL 的表达部分将 Padding 在容器层进一步缩短结构，可能会提高 FPS，也就是将现在的简单容器布局进行特性升级。

## 性能下一步探索



## 四、Flutter 研发框架下互动能力

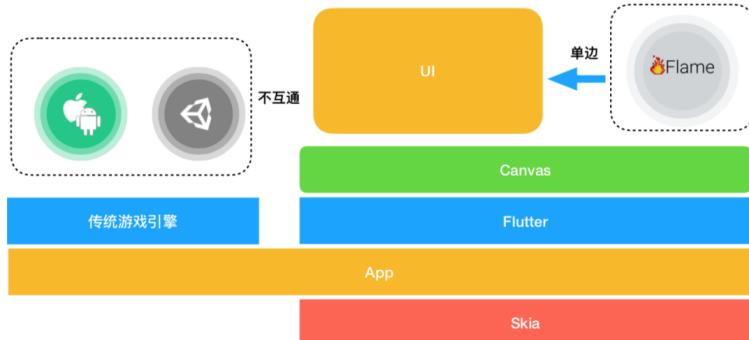
### 背景与现状

在电商领域的业务里面，很多业务想要通过游戏化的方式创造更有表现力的交互体验，创造新的业务玩法和价值。传统的 UI 表达方式，越往后就会越受限，因此需要将 UI 和游戏引擎的边界打破，让 UI 具有游戏的丰富动效能力，也让游戏引擎具有 UI 的丰富控件能力。在传统 APP 的框架下，所能够做的无外乎嫁接游戏引擎，而这样的游戏引擎和原来的 APP 是格格不入，也是不相通的，其能够带来的最大效果就是开辟一个独立的页面来承载游戏，但这样的方式似乎不是所想要达到的设计理念。在 Flutter 侧，今年推出了 Flame 这个游戏框架，其解决了单边引用的过程。Flame 使得在 Flutter 框架里面可以将游戏的控件进行合拢，但是无法实现在游戏里面合拢 UI 界面，因此提供了单边能力。Flame 虽然没有完全解决双边打通的诉求，但是还是提供了很好的思路。

## 背景和现状

Alibaba Group | TAO TECHNOLOGY

### ● UI 与游戏引擎的边界



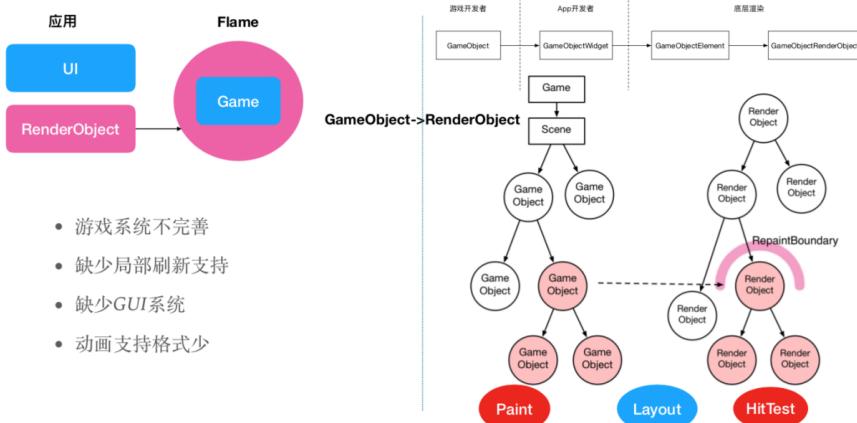
● 用游戏化的方式创造更有表现力的交互体验，创造新的业务玩法和价值

## 核心问题 – 融合

目前而言，所需要解决的核心问题就是将 UI 和游戏引擎融合。Flame 的表现形式其实就是将完整的游戏封装在起来，能够很好地将游戏插入到 UI 中。假如将 Flame 游戏引擎的表达也用类似于 RenderObject 树形结构的表达，就会形成两棵 Flutter 体系下的树结构，能够很好地进行融合比对。闲鱼在这样的思路下进行了新的探索和尝试，重新设计了一套互动游戏引擎，弥补了 Flame 不能满足的需求问题。

## 核心问题-融合

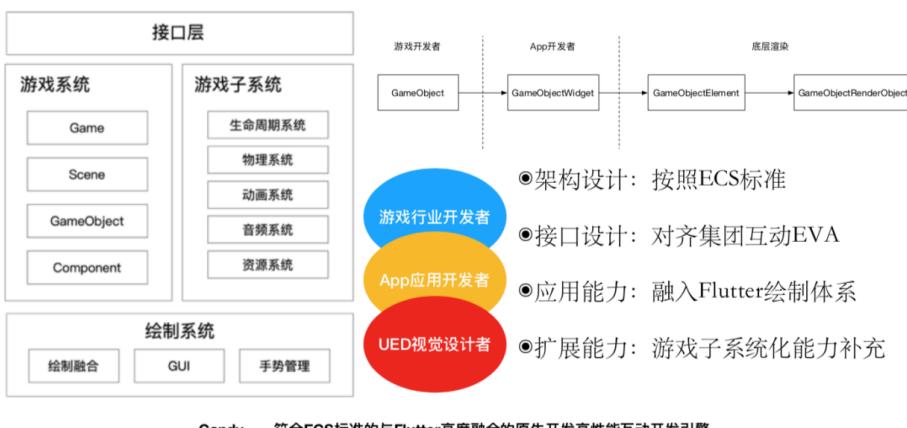
Alibaba Group | TAO TECHNOLOGY



## Candy 整体设计

Candy 是符合 ECS 标准的，与 Flutter 高度融合的原生开发高性能互动开发框架。Candy 在架构设计上完全按照 ECS 的标准；在接口设计上对齐了阿里巴巴集团的互动 EVA，使得集团内部在使用时不会对于接口感到陌生；在应用能力上，Candy 完全融入了 Flutter 的绘制体系；在扩展能力上，Candy 保留了游戏子系统化能力的补充，能够满足 UED 主流能力，使得不同公司或者行业开发者能够更好地使用自己所熟悉的工具体系。

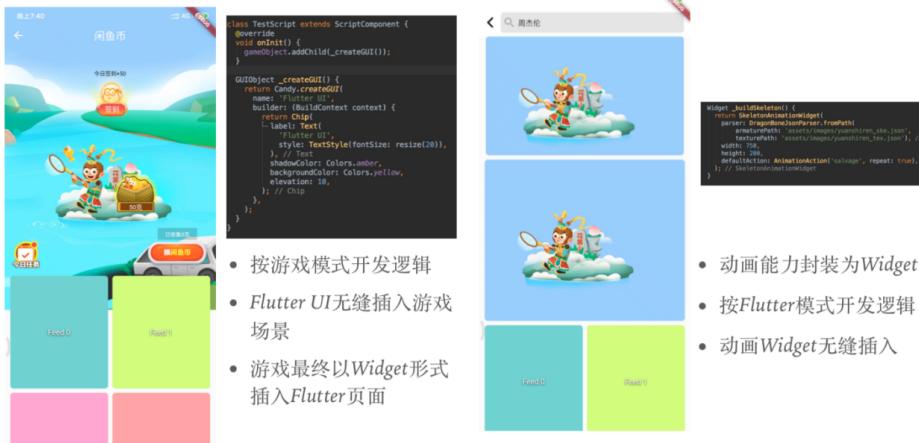
## Candy 整体设计



## 效果

在闲鱼中，签到、换取闲鱼币等常用的按钮在游戏中使用了 Flutter 的游戏控件，能够非常简单地将游戏以 Widget 形式插入到 Flutter 页面中，而这对于使用者而言不会产生任何感知。此外，对于传统应用的开发者，也能够很轻松地将具有动画能力、游戏能力的控件外透，并且与 UI 进行融合。

## 效果



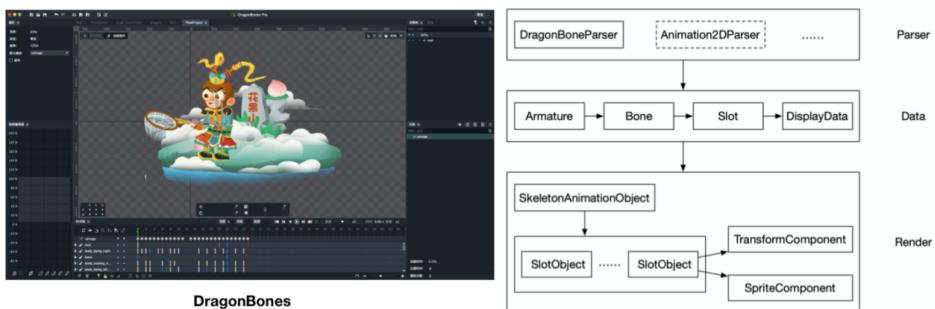
Alibaba Group | TAO TECHNOLOGY

## 效果 - 骨骼

目前，闲鱼的内部方案能够很好地实现龙骨的动画。正是因为在子系统中保留了这样的能力，所以如果未来有其他方案也可以按照 ECS 标准进行主流格式的实现。

## 效果-骨骼

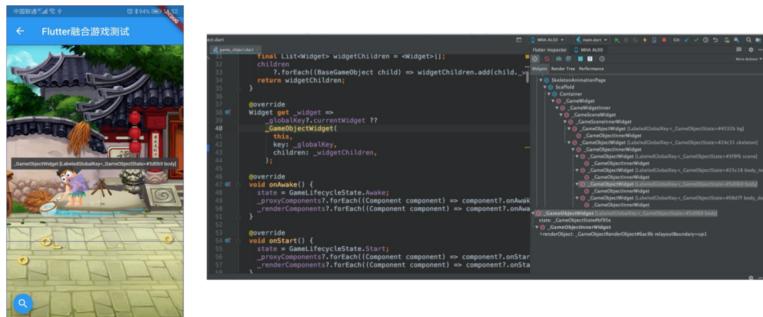
Alibaba Group | TAO TECHNOLOGY



## 效果 – 调试

值得一提的是因为引擎和 UI 不分家，使得在调试工具的使用过程中能够更好地看出游戏或者动画页面的布局情况，降低了调试工作的难度。

## 效果-调试



## 性能参考

基于 Render 层的设计使得我们享受到了 Flutter 引擎侧对于 Canvas 的优化，也享受到了在 Flutter Framework 上局部刷新能力，因此无论是实现粒子还是实现骨骼的动画，性能表现都非常不错。

## 性能参考

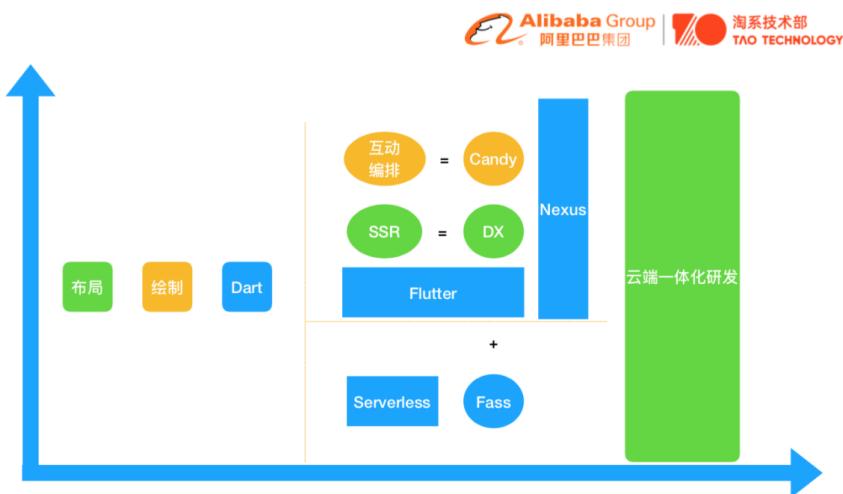


屏幕上粒子个数	页面FPS
6000	29
3000	32
1500	56
1200	58
1000	59
300	60

	CPU	内存	FPS
1个人 36根骨骼	60	127	60
10个人 380根骨骼	63	147	60
30个人 1080根骨骼	145	160 7m增加到190M	60
50个人 1900根骨骼	150	170 2分40s增加到200M	30
100个人 3800根骨骼	160	200 – 250 波动较大	20

## 五、后续规划和展望

回顾本文内容，首先为大家分享了 Flutter 侧在布局方面的突破，Flutter 这样的开源框架具有足够的能力能够让大家在其布局侧进行进一步深挖。此外，还和大家分享了闲鱼在 Flutter 互动领域的突破，能够很好地将 UI 和特效进行融合。第三个突破就是闲鱼在 Dart 侧的突破，将 Dart 语言实现了云端开发和联动，形成了逻辑后移的开发框架。而单点技术难以形成全面的合力，因此在后面与大家分享了将现有能力组合的情况产生不同的体系。假设将 FaaS 远端的动态能力结合 Nexus 一体化能力、DX 基于 UI 的表达能力，似乎就可以通过 SSR 写完整的 UI 部分。同理，FaaS 结合 Candy 也能够实现互动编排的能力，将互动能力在 FaaS 端进行表达。



由阿里云开发者社区志愿者贾子甲整理。

## AliFlutter 图片解决方案与优化

**摘要:** Flutter 与 Native 混合开发将是接下来很长时间的主流开发方式。一套稳定、高效、与官方体系无缝融合的外接图片缓存方案是必不可少的。在 AliFlutter 系列第三场直播中，由阿里巴巴新零售淘系技术部无线开发专家王乾元为大家介绍了 AliFlutter 提供的适合混合应用的外接图片库方案。首先对 Flutter 官方原生方案进行了分析，并提出了 AliFlutter 方案的切入点以及具体优化手段。

### 演讲嘉宾简介：

王乾元，花名神漠，13 年加入阿里，先后负责过天猫、支付宝、手机淘宝 App 的 iOS 架构工作。目前在 AliFlutter 团队负责基础组件、iOS 架构，以及引擎、工具链等方面的研究。

以下内容根据演讲视频以及 PPT 整理而成。

观看回放 <http://mudu.tv/watch/5624777>

本次分享主要围绕以下三个方面：

一、Flutter 如何显示、加载图片

二、AliFlutter 图片解决方案优化

三、如何选择最适图片解决方案

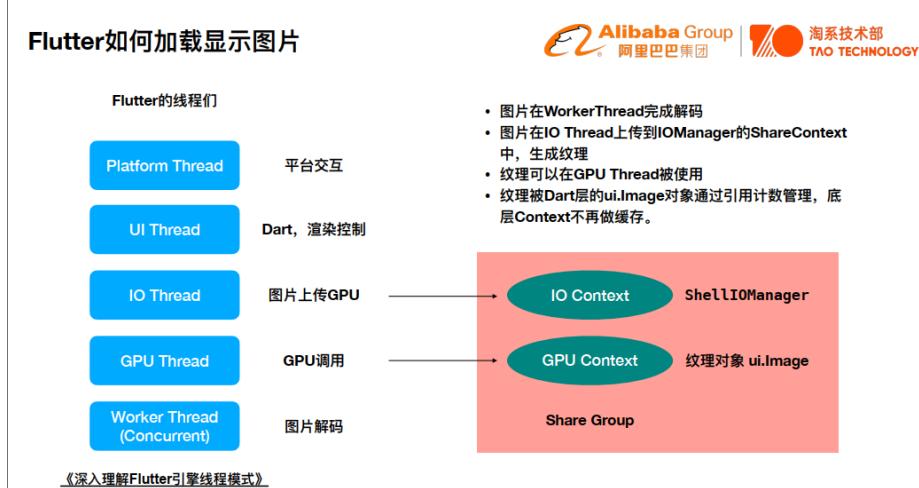
### 一、Flutter 如何显示、加载图片

介绍 Flutter 如何加载、显示图片，以及在线程、缓存设计层面的特点。

## 1. 线程

闲鱼分享的文章《深入理解 Flutter 引擎线程模式》中，详细讲解了 Flutter 引擎线程模型的原理以及作用。

- **Platform Thread:** IOS 与安卓平台层应用的主线程。进行 Flutter Engine 接口的调用，用户手势和输入等也通过 Platform Thread 输入给 Flutter Engine。
- **UI Thread:** Flutter 的主线程，也称为 Dart 线程。同时可运行 C++ 代码。
- **IO Thread:** 进行图片上传。图片在 IO Thread 进行异步上传生成 GPU 纹理。
- **GPU Thread:** 负责 Flutter 最终的 GPU 调用。
- **Worker Thread:** Flutter 中的 fml 会创建若干个并发工作线程。可进行图片解码等工作。



如上图所示，图片在 Worker Thread 完成解码，在 IO Thread 进行异步上传，在引擎启动时创建的 ShellIOManager 会创建 OpenGL Context。同时 GPU Thread 创建 GPU Context。IO Context 与 GPU Context 将存放在 Share Group 中共享纹理。Flutter 中纹理对象是 C++ 的对象，在 Flutter 底层不会对纹理对象进

行任何缓存，而是通过 Dart 层的 ui.Image 对象通过引用计数进行管理。

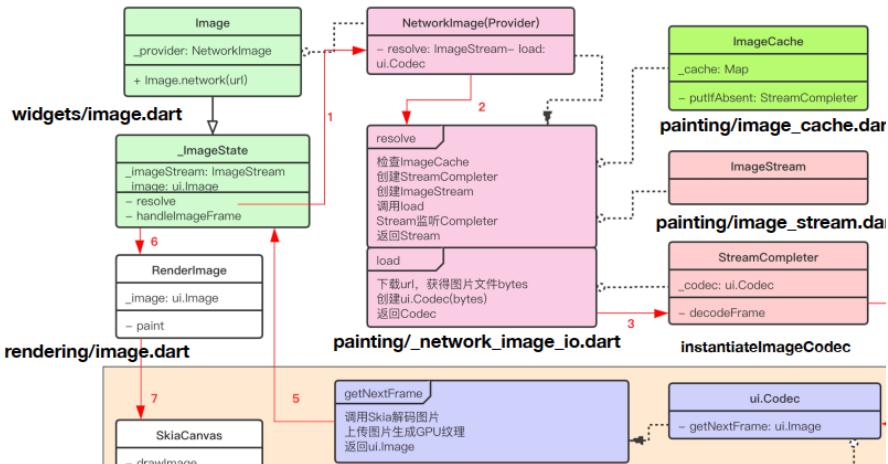
## 2. 图片加载、显示流程

下图为 Flutter 从图片加载到显示的相关类关系图，包括类所在文件。

图片加载用到 Flutter 的 Image Widget，一般是使用其“.network”接口加载网络图片。Image Widget 进行显示绘制时需要 ImageState。ImageState 有两个功能，一是驱动 Provider 下载图片，二是调用 State 管理底层 Render object。Render object 负责图片的渲染上屏。

NetworkImage(Provider) 在自身 resolve 方法中异步调用 http 下载图片。resolve 方法调用 Provider 获取自己的 ImageStream。ImageStream 会添加到 StreamCompleter 作为 Listener。StreamCompleter 可以添加多个 ImageStream 作为 Listeners。图片下载完成后通过 Dart 层和 C++ 层的接口函数 instantiateImageCodec 创建底层 C++ 解码器的 C++ 对象。解码器对象获取图片流后在底层进行异步解码，并生成纹理。ImageState 接收到事件后获取纹理对象绘制图片。上层获取图片纹理后会调用 ImageState 的 setState 方法将纹理对象传给底层 Render object，排版完成后图片就会绘制到屏幕。

底层纹理对象会被上层 Dart 对象引用，具体为以下几个对象。StreamCompleter 负责驱动底层解码器获取纹理对象。因此 StreamCompleter 会持有底层 GPU 纹理，并通过 Listeners 通知所有 ImageState。因此 ImageState 也会持有纹理对象。ImageState 将图片传给底层 Render object，因此 Render object 也会持有纹理对象。当上层 Image Widget 被销毁，Image Cache 清空时，触发底层纹理的释放。



Flutter 加载显示图片的流程包括了图片的组件、下载、解码、上传、绘制等工作，看似复杂，但是其逻辑较为简单。

## 二、AliFlutter 图片解决方案优化

### 1. 问题

首先，利用 Flutter 制作淘宝商品详情页面，图片多，内存、CPU 等占用非常高，性能要求高。Flutter 图片管理能力较弱，缺乏本地缓存能力，图片的重复下载极易造成内存飙升，易发生 OOM (OutOfMemory) 情况。因此 Flutter 原生方案无法满足需求，需要构建适合的 AliFlutter 方案。

第二，电商 APP 需要与 Native 图片库对接，共享缓存、CDN 能力以及监控设施。

第三，在使用简单的基础之上，AliFlutter 需要基于 Flutter 的强大扩展能力，支持小程序、Canvas 等多种场景。

第四，希望 AliFlutter 与官方 Flutter 体系尽可能兼容与融合。



- 电商场景，图片多，性能要求高；
- 需要与Native图片库对接，共享缓存、CDN能力以及监控设施；
- 使用简单
- 扩展能力强，支持小程序、Canvas等多种场景；
- 需要与官方体系尽可能融合。



- 原生图片管理能力比较弱，缺乏本地缓存；
- 高并发，图片多的场景，性能较差；
- 无法支持基于Flutter Canvas的场景。

## 2. AliFlutter 图片解决方案总体架构

下图红色标签为 AliFlutter 方案的重点。在 Dart 层实现了新的 Provider，在 C++ 层实现了新的解码器对象，并基于 Flutter 规范提供了不同平台的 ObjC、安卓的 Java 接口。

AliFlutter 图片解决方案追求以下三个特点。

**一致性：**与官方体系无缝融合。仅在官方基础上添加代码。

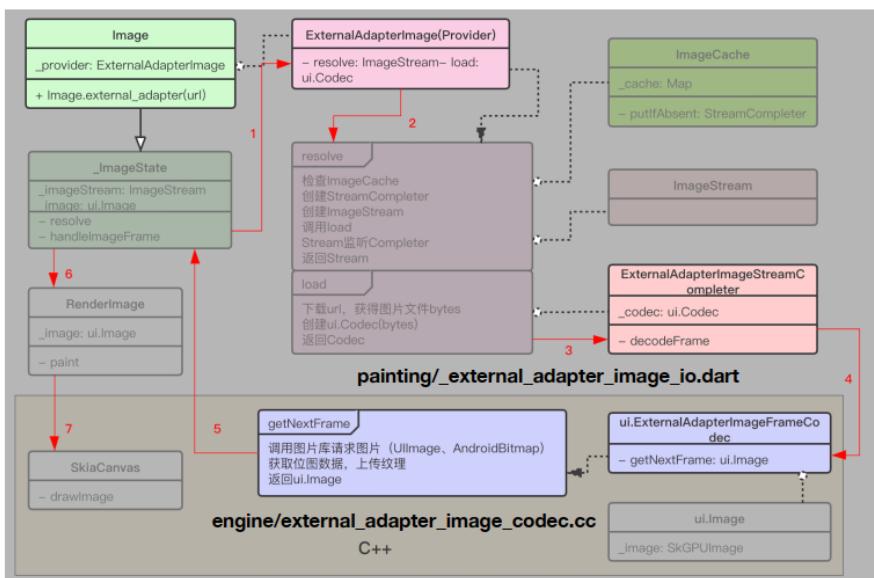
**高性能：**优化 CPU、内存占用，增强 List 回收能力。

**易用性：**适配简单、使用简单、易扩展。



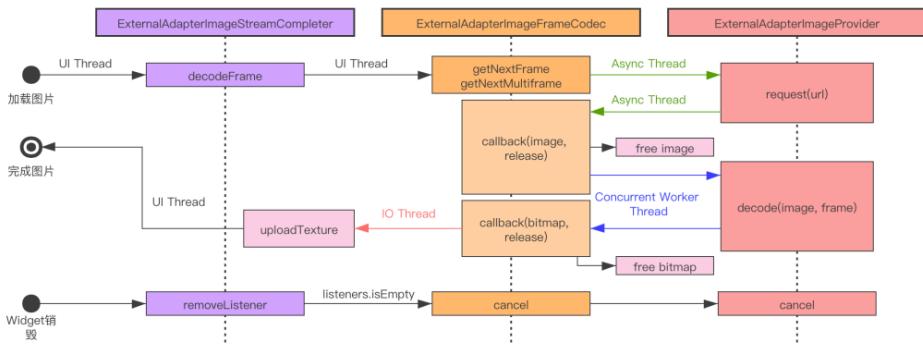
**AliFlutter:** 如下图所示，高亮部分为 AliFlutter 改进部分。

Image Widget 添加了新类型的 Provider, ExternalAdapterImage。新 Provider 接收的参数是 URL、图片尺寸信息等。可将参数通过 Adapter 传给 Native 图片库, 进行图片下载或从缓存中加载。Completer 会创建新的解码器对象, 通过 Adapter 对接 Native 图片库, 让 Native 图片库提供图片的原始 Buffer, 并进行解码。即不依赖 Flutter 的图片解码能力, 而是依赖平台层例如 IOS 和安卓原生的图片解码能力, 可支持更多图片格式。将平台层解码后的 bitmap 返回给解码器对象, 通过位图数据进行图片纹理的上传。AliFlutter 解码器底层的 C++ 对象支持这两种工作模式。



**一次完整图片加载过程时序图：**首先从 Image Widget 拿到图片请求 URL，调用到底层解码器对象的 `getNextFrame` 方法会将请求异步上传给对接的 Native 图片库。由 Native 图片库做请求，获取平台层的图片对象或 Buffer，将图片对象返回给解码器对象。解码器对象在 Worker Thread 中进行图片解码。图片解码完成后在 IO Thread 进行图片的 GPU 纹理上传。上传完成后在 UI Thread 将图片返回给 Dart。上述流程完成一次图片加载，线程模型与 Flutter 原生保持一致。

**图片取消:** AliFlutter 方案相比 Flutter 原生方案新增了 Cancel 能力。Widget 通过 State 将自己添加到 Completer 的 Listeners 中。因此 Widget 销毁时会将自己从 Listeners 中移除。当 Completer 的 Listeners 全部清空时，表示这次图片请求已经不再需要了，调用底层解码器对象的 cancel 方法。如果图片还未从 Native 图片库返回，可以取消下载；如果已经返回，还有解码或上传 GPU 过程，都可以及时取消操作。Cancel 能力可以避免许多无用的 CPU 和内存的消耗，尤其是电商 App 中常见的快速滑动商品列表的场景。



### 3. 性能优化

AliFlutter 进行了以下层面的优化，除图片取消外，还包括延迟加载、解码并发控制、GIF 逐帧上传纹理、增强 List 回收能力等。

- **延迟加载**
  - 列表快滑时，直接减少无效图片请求；
- **图片取消**
  - 网络慢，纹理上传过程多次切换线程，过程中图片可能不再被需要；
- **解码并发控制**
  - 降低图片到GPU纹理过程临时内存峰值；
- **GIF逐帧上传纹理**
  - 降低GIF播放过程内存峰值；
- **增强List回收能力**
  - 解决大Cell问题，图片离屏纹理即回收。

CPU

内存

**适配与使用：**介绍 AliFlutter 图片方案最终对接到平台层 Native 图片库的接口。

Flutter 的封装是在 IOS 平台公开了 Objective-C 接口，在安卓平台提供了 Java 接口，因此 AliFlutter 遵循 Flutter 规范提供了 OC 接口与 Java 接口。

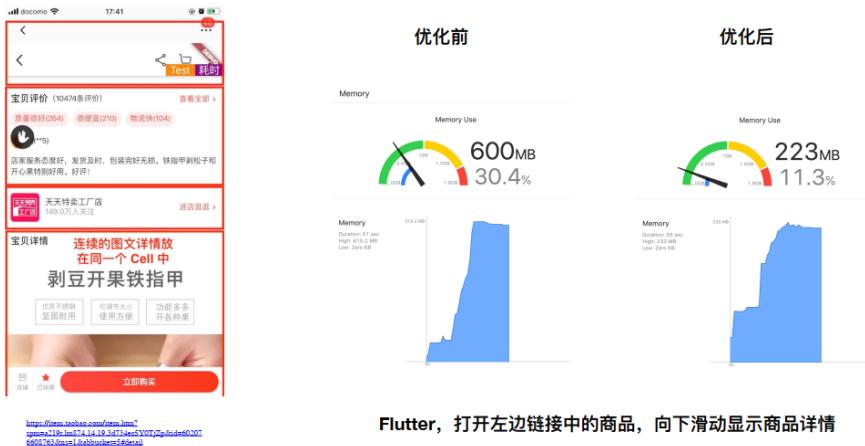
IOS 平台 OC 接口只需要实现一个回调。OC 回调在对接图片库时接收的是 URL 以及一些参数，获取图片后向底层返回 UIImage 即可。使用时可以直接调用 Dart 的 Image.externalAdapter 方法加载一张图片。在此可以指定 placeholderProvider，可以是 AssetImage 或其他网络图片，以此可在主图加载失败时加载一张副图。

Objective-C	Java
<pre> @protocol FlutterExternalAdapterImageRequest &lt;NSObject&gt; - (void)cancel; @end  @protocol FlutterExternalAdapterImageProvider &lt;NSObject&gt; - (id&lt;FlutterExternalAdapterImageRequest&gt;)request:(NSString*)url     targetWidth:(NSInteger)targetWidth     targetHeight:(NSInteger)targetHeight     parameters:(NSDictionary&lt;NSString*, NSString*&gt;*)parameters     extraInfo:(NSDictionary&lt;NSString*, NSString*&gt;*)extraInfo     callback:(void (^)(UIImage*))callback; @end </pre>	<pre> public interface ExternalAdapterImageProvider {     public interface Request {         public abstract void cancel();     }      public interface Response {         public abstract void finish(Image image);     }      public abstract Request request(String url,         int targetWidth,         int targetHeight,         Map&lt;String, String&gt; parameters,         Map&lt;String, String&gt; extraInfo,         Response response     ); } </pre>
Dart	
<pre> Image.externalAdapter(   'https://gw.alicdn.com/tfs/TB1Aa0UcF67gK0jSZPfxXahhFx-750-140.png',   placeholderProvider: AssetImage("assets/placeholder.jpg"),   targetWidth: 375,   targetHeight: 375, ); </pre>	

## 4. 增强 List 回收能力

**优化前后对比：**下图左侧所示为使用 Flutter 制作的淘宝商品详情页面，其中有很多 Cell。其中一个 Cell 为宝贝详情。宝贝详情 Cell 最初的实现方式是解析一段 HTML。商家有时会上传多张高清大图，若此时将连续的图文详情放在一个 Cell 中，用户浏览详情页时会同时加载多张大图。另外 Flutter 默认对所有 Cell 添加 RepaintBoundary 属性，该属性默认将 Cell 中所有内容绘制到一个纹理中，下次浏览时若 Cell 中内容不变，直接使用纹理绘制图片会比较快速。因此易导致内存飙升问题。

如下图所示，优化前内存容易暴增到 600+MB 甚至 1G，几乎 100% 会出现 OOM 问题。在业务代码不进行修改的情况下，优化后的内存增长变得较为平缓。

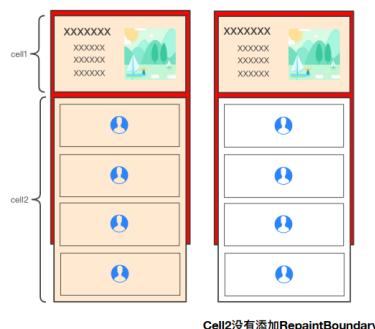


**Flutter List 特点:** Flutter List 回收以 Cell 为单位。下图所示红色框部分为屏幕大小。默认情况下 Flutter 默认对所有 Cell 添加 RepaintBoundary。当列表滚动时，若 Cell 1 绘制过，下次绘制时直接将及纹理上屏即可，无需绘制内部图文元素。而 Cell 2 会占用大量内存，首先其图文多，同时 RepaintBoundary 形成的纹理也会占用大量内存。

因此增强 List 回收能力首先需要解除对部分 Cell 的 RepaintBoundary 设置。

#### 增强List回收能力——Flutter List特点

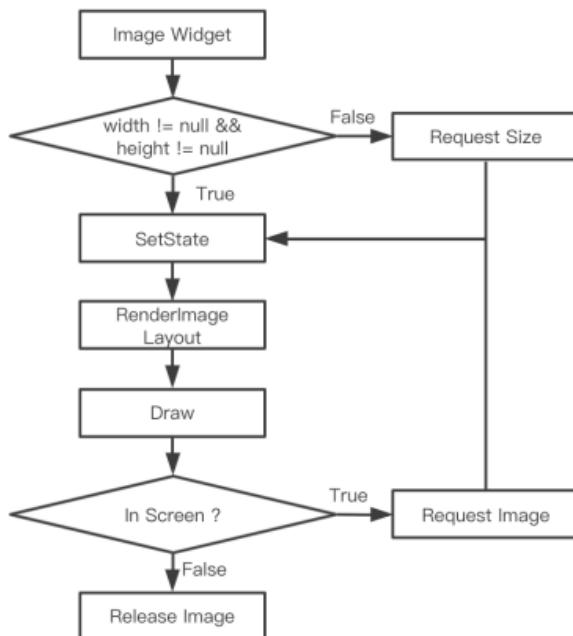
Alibaba Group | TAO TECHNOLOGY



**优化流程：**假设一个 Image Widget 在一个 Cell 中，正常情况下当 Cell 出现，Image Widget 也会被创建并且请求图片。List 回收能力的优化中试图解除此约定，根据图片是否在屏来判断是否需要图片纹理。若不需要，则释放，若需要，进行请求。

Image Widget 的宽、高已知情况下，其排版信息是有效的。SetState 完成后触发底层 Render Object 排版与绘制。在绘制图片过程中添加一段逻辑判断图片是否在屏。若图片不在屏，不作任何处理。若图片在屏幕中，进行图片请求获取真实图片后重复调用 SetState，重新进行图片排版和绘制，并判断是否在屏。若图片随着列表滚动不在屏幕中，则回调通知上层解除纹理引用。

若 Image Widget 的宽、高未知，Flutter 只能在获取图片后根据其真实尺寸进行排版。原本底层解码器对象持有 getNextFrame 接口，该接口导致 GPU 纹理的生成。在优化后可以不依赖图片纹理上传完成再进行排版。在流程中添加了 Request Size 接口，Image Widget 的宽、高未知时调用该接口可以预先通知底层 C++ 解码器获取图片尺寸。得到图片尺寸后再从 SetState 开始流程，避免了无效的纹理上传。



**关键代码：**判断图片是否在屏是通过 Dart 层的 Image Render Object。其 paint 方法中进行图片是否在屏的判断，根据其是否在屏向上层 ImageState 发送回调通知。

实现指定 Cell 不添加 RepaintBoundary 是通过建立虚类 NoRepaintBoundaryHint。若 List 检测到上层某个 Cell 继承自 NoRepaintBoundaryHint，则不给该 Cell 添加 RepaintBoundary。因此可以在每次屏幕滚动时重新进行绘制，了解图片的在屏、离屏信息。

#### 绘制时判断图片是否在屏，回调通知

#### List 对指定 Cell 不添加 RepaintBoundary

```

// image.dart
void paint(PaintingContext context, Offset offset) {
    // Check if Rect(offset & size) intersects with screen bounds.
    final double screenWidth = ui.window.physicalSize.width / ui.window.devicePixelRatio;
    final double screenHeight = ui.window.physicalSize.height / ui.window.devicePixelRatio;

    if (offset.dy >= screenHeight - 1 || offset.dy <= -size.height + 1 ||
        offset.dx >= screenWidth - 1 || offset.dx <= -size.width + 1) {
        // Not in screen
        if (_image != null) {
            // Notify owner
            _needsImageCallback(true);
        }
        return;
    } else if (_image == null) {
        // In screen but image is null, notify owner to download.
        _needsImageCallback(true);
        return;
    }
}

// silver.dart
abstract class NoRepaintBoundaryHint extends RepaintBoundary {
    @override
    Widget build(BuildContext context, int index) {
        assert(index >= 0 && index < children.length);
        Widget child = children[index];
        final Key key = child.key;
        if (key == null || _saltedValueKey(key) == null) {
            throw StateError("The silver's children must not contain null values, but a null value was found at index $index");
        }
        if (addRepaintBoundaries && (child is RepaintBoundary)) {
            child = RepaintBoundary(child: child);
        }
        if (addSemanticIndexes) {
            final int semanticIndex = semanticIndexCallback(child, index);
            if (semanticIndex != null)
                child = Semantics(
                    ...
                );
        }
        return child;
    }
}

```

图片解码时通过 Image Codec 接口实现只获取图片尺寸，不上传纹理。图片尺寸可以直接从图片的头部信息获取，并不需要分配内存。

图片排版时可以仅根据图片的尺寸信息进行排版，无需获取真实图片。

总结起来，大 Cell 优化就是避免图片纹理上传，图片真正在屏时，再获取其纹理，当图片离屏时，立刻清除其纹理。

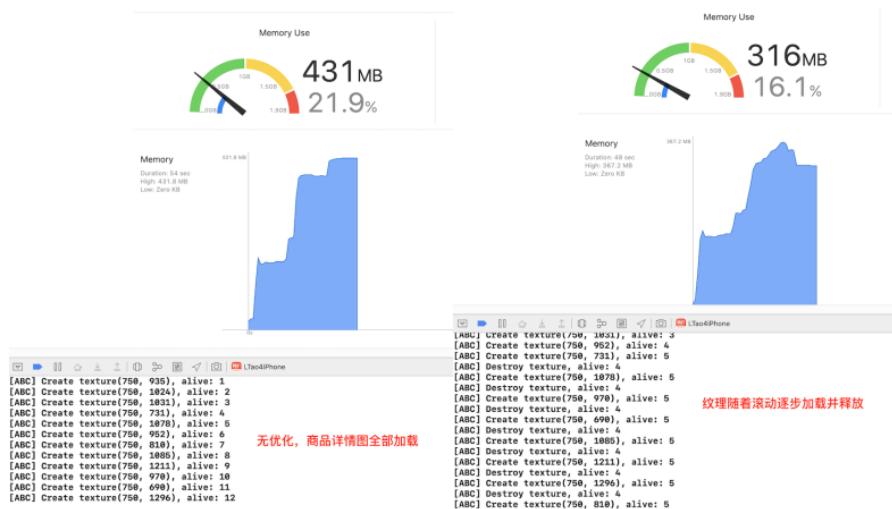
Image Codec获取图片尺寸，但不上传纹理

```
flutter > lib > src > widgets > image.dart > _ImageState > _updateSourceStream
1165 void _handleImageFrame(ImageInfo imageInfo, bool synchronousCall) {
1166   setState(() {
1167     _imageInfo = imageInfo;
1168     _loadingProgress = null;
1169     _frameNumber = _frameNumber == null ? 0 : _frameNumber + 1;
1170     _wasSynchronouslyLoaded |= synchronousCall;
1171   });
1172 }
1173
1174 void _handleImageInfo(int width, int height, int frameCount,
1175   [Duration durationInMs, int repetitionCount]) {
1176   // We get image dimension, notify render object to perform layout.
1177   setState(() {
1178     _imageWidth = width;
1179     _imageHeight = height;
1180   });
1181 }
```

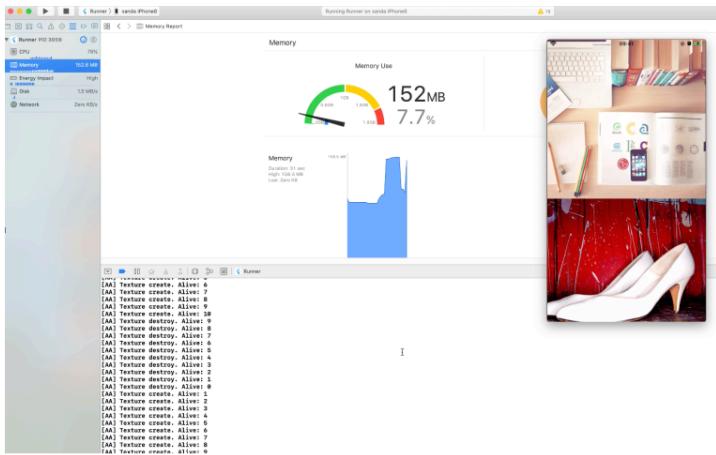
只有图片尺寸信息，也可以进行排版

```
flutter > lib > src > rendering > image.dart > AutoreleaseRenderImage > _sizeForConstraints
460 @override
461 Size _sizeForConstraints(BoxConstraints constraints) {
462   // Fold the given [width] and [height] into [constraints] so they can all
463   // be treated uniformly.
464   constraints = BoxConstraints.tightFor(
465     width: _width,
466     height: _height,
467   ).enforce(constraints);
468
469   // Intrinsic from image itself or image pixel dimension info.
470   if (_image == null && (_imageWidth == null || _imageHeight == null))
471     return constraints.smallest;
472
473   // Use _image if not null
474   if (_image != null) {
475     return constraints.constrainSizeAndAttemptToPreserveAspectRatio(Size(
476       _image.width.toDouble() / _scale,
477       _image.height.toDouble() / _scale,
478     ));
479   }
480
481   // Or else use image dimension info.
482   return constraints.constrainSizeAndAttemptToPreserveAspectRatio(Size(
483     _imageWidth.toDouble(),
484     _imageHeight.toDouble(),
485   ));
486 }
487 }
```

**优化效果：**经过以上优化，List 回收能力的增强取得了较好效果。当商品详情页面有几十张大图在同一列表的同一个 Cell 中出现，可以做到仅加载在屏图片，若图片离屏则释放。

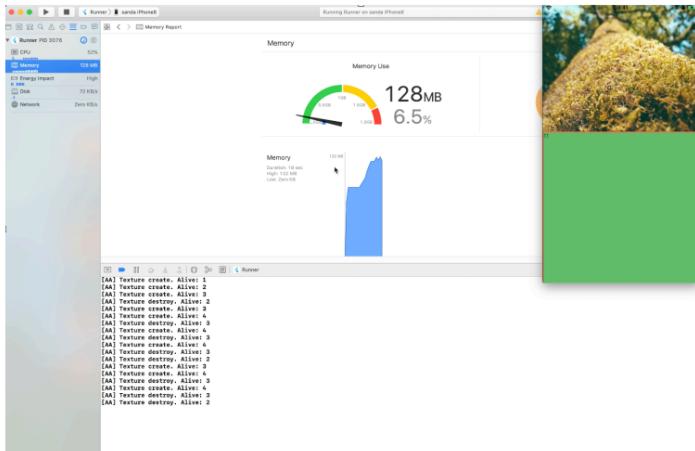


**案例 - 优化前：**十张图片放在一个 Cell 中，内存突增突降。



备注：直播时这里为视频。  
优化前，大Cell中的图片一次性全部加载，统计到存活的纹理为10，当Cell回收后，纹理一次性清空。  
存在内存大起大落的情况。

**案例 - 优化后：**根据图片是否在屏进行加载或释放，内存增降均较为平缓。



备注：直播时这里为视频。  
优化后，大Cell中的图片根据列表滚动时是否在屏来加载与销毁。统计到存活的纹理以及内存的增长、下降比较平缓。

## 5. 后续改进

AliFlutter 图片解决方案还有以下方面可以改进。

**功能改进：**第一，图片在屏、离屏判断优化。第二，支持图片库返回图片原始文件，精简链路。第三，支持业务定制化缓存策略。

**包优化大小：**允许定制化裁剪 Flutter 中的若干图片解码库，同时保证 Flutter 所有功能正常。

**与官方探讨：**如何将 AliFlutter 优化融合到 Flutter 主干。

### 三、如何选择最适图片解决方案

Flutter 图片解决方案诞生以来，开发者也进行了许多尝试，创建图片库方案。难以定论哪些图片方案更加优秀。

如下图所示，开发者可以考虑图片解决方案是否为纯 Flutter 应用，网络图片场景多不多，图片有无必要缓存等方面。根据自己的应用场景选择最适合自己的图片解决方案。

#### 如何选择适合自己App的图片方案



由阿里云开发者社区志愿者吴晨雪整理。

# UC Flutter 技术实践分享

**摘要:** UC 于 19 年开始探索 Flutter 技术，并在同年年底进行规模化落地。规模化落地 Flutter 核心要解决的三类问题分别是工程构建体系的搭建，性能优化和动态性支持。本次分享将由阿里巴巴 UC 事业部无线开发专家刘森森为大家详细介绍 UC 在规模化落地 Flutter 过程中解决的问题，及其思考过程。

**演讲嘉宾简介:** 阿里巴巴 UC 事业部无线开发专家——刘森森（花名：森尼）。14 年加入 UC，长期在 UC 信息流团队负责信息流业务的技术工作，近一年投入到创新产品的研发中，负责 Flutter 技术在创新产品的应用与实践。

以下内容根据演讲视频 <http://mudu.tv/watch/5624777> 以及 PPT 整理而成。

本次分享主要围绕以下五个方面：

一、Flutter 在 UC 落地的情况

二、工程体系构建

三、性能优化

四、动态性探索

五、总结

## 一、Flutter 在 UC 落地的情况

### 1. 业务落地情况

Flutter 具备高性能、高效率两个特性。UC 作为创新事业群的一部分，以创新为

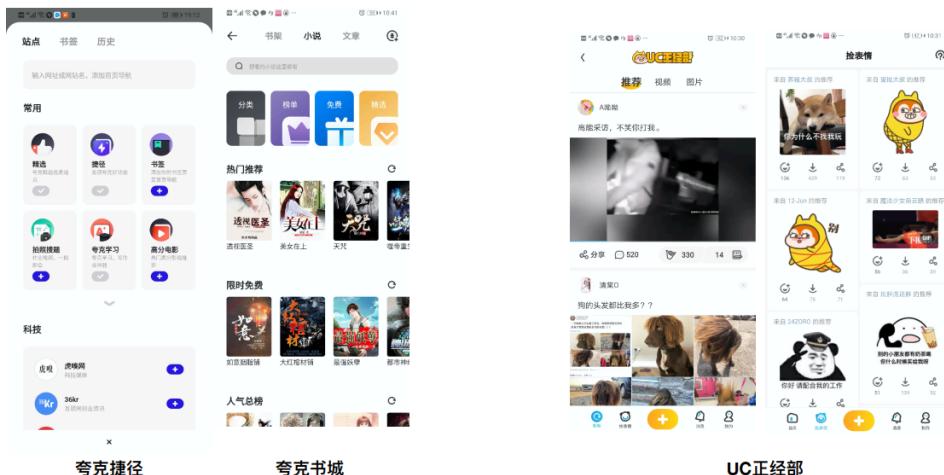
重要使命，希望 Flutter 可以直接加速 UC 的创新。目前 UC 国内有 50% 的研发同学已经使用 Flutter。从前需要两端开发的需求现在一端人力就可以支撑，并且前端同学可以加入做 Native 需求的行列，研发效率得到大幅度提升。

目前 UC 有两类业务。一类是成熟业务，例如 UC 和夸克，其新功能会优先考虑使用 Flutter。一类是创新业务，例如小萌圈和古桃，基本都使用 Flutter 开发 UI。



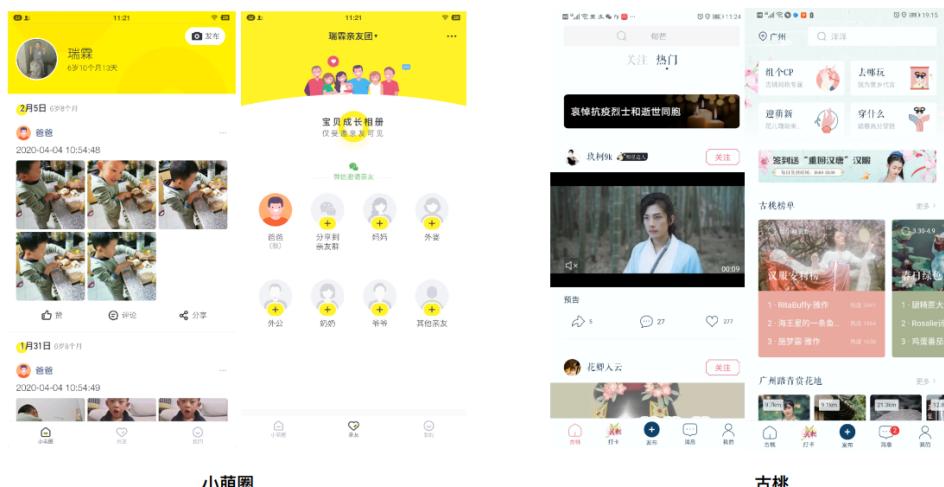
## 2. 夸克 &UC 具体业务形态

夸克 &UC 的业务形态是将 Flutter 嵌入 Native 框架，是混合栈开发模式。业务以 feed 流为主，内容以音视频、图片、GIF 为主。



### 3. 创新产品

90% 页面使用 Flutter 开发。未来发展的方向即 UI 将尽可能使用 Flutter 进行开发。



## 4. 总览

UC 落地 Flutter 有多种场景，目前的阶段是将 Flutter 规模化落地到创新业务中，加速 UC 的创新发展。

当前 UC 的 Flutter 落地面临着以下三类问题。

**工程构建体系：**如何提高 Flutter 落地效率？

**性能优化：**UC 业务痛点在哪里？

**动态性探索：**如何结合业务实现动态性？

## 二、工程构建体系

### 1. 核心目标——高效率

可快速集成到现有 APP。

可快速构建全新的 Flutter 应用。

研发可快速上手。

### 2. 工程架构

将 Flutter 业务落地到 UC 中有两种方案。

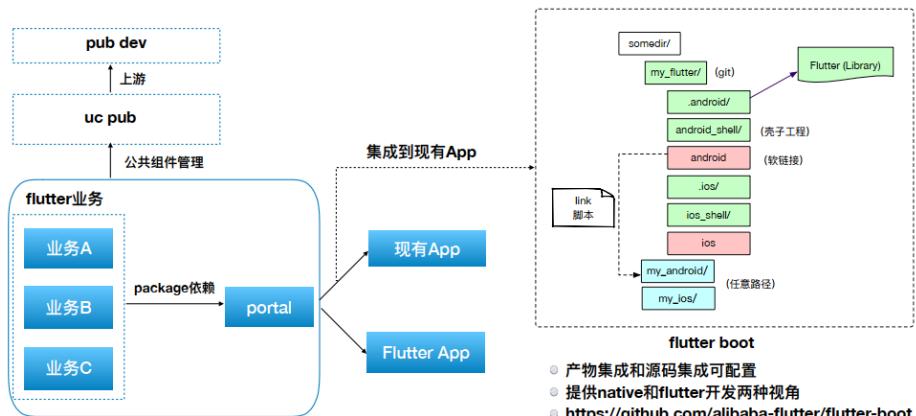
**集成到现有 APP：**Add Flutter to Existing APPs。目前 UC 使用的是 Flutter Boot 解决方案。Flutter Boot 提供了两个核心优势。一，产物集成和源码集成可配置。UC 中有部分同学是没有用到 Flutter 的，在开发中应该避免由于 Flutter 打包带来的效率损耗，所以希望面向该部分同学配置产物集成方式。面向 flutter 开发同学希望配置源码集成方式。以实现效率最大化。

二，Flutter Boot 提供 Native 和 Flutter 开发两种视角。Native 开发视角指在

Native 工程中执行 Native 的构建脚本、命令。官方方案只能使用 Native 开发，无法使用一些 flutter tool 的高效命令，会造成研发效率的下降。另外，前端同学更希望使用 Flutter 原生命令，而不希望理解安卓和 iOS 两种构建体系的差别，因此 Flutter Boot 对前端同学友好。

**集成到 Flutter APP：**全新 APP。推荐使用 Flutter 官方解决方案，更高效、简单。

**工程架构：**Flutter 业务以 Package 依赖的工程结构组织。UC 是多团队、多业务的开发模式，期望业务之间解耦。目前 Flutter 不支持产物分离，都打包在一起，因此前期希望在工程方面尽可能做到解耦。公共组件会放到 UC pub 中，pub 管理比 git 管理更加高效直观。目前 UC pub 中沉淀了许多业务组件和技术组件。UC pub 上游可进行修改。技术组件打磨成熟后可以回归 Ali pub 或 pub dev。



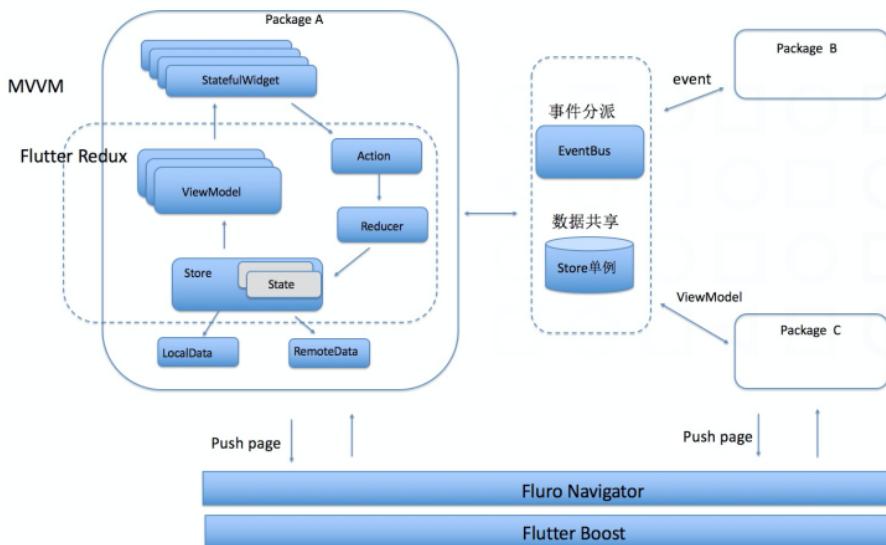
### 3. 业务架构

业务架构核心思路是抽象几个核心模块，构建架构模板，使业务开发同学有一致认知。

最下方使用 Flutter Boost 解决方案，Flutter Boost 是开源比较久的混合栈的解

解决方案。UC 是 Flutter Boost 的重度用户，同时积极反哺社区，例如支持 hero 动画等。即使 UC 使用了 Flutter Boost，UC 希望 Flutter 页面之间的相互跳转是使用原生方式，支持 hero 动画可以使动画做的更漂亮。

每个 Package 使用 Flutter Redux，主要是因为 Flutter Redux 能够解决复杂场景的状态管理。另外许多前端同学拥有 Redux 开发经验，可以带领客户端同学学习。



是 flutter boost 的重度用户，同时也积极反哺社区，例如支持 hero 动画等

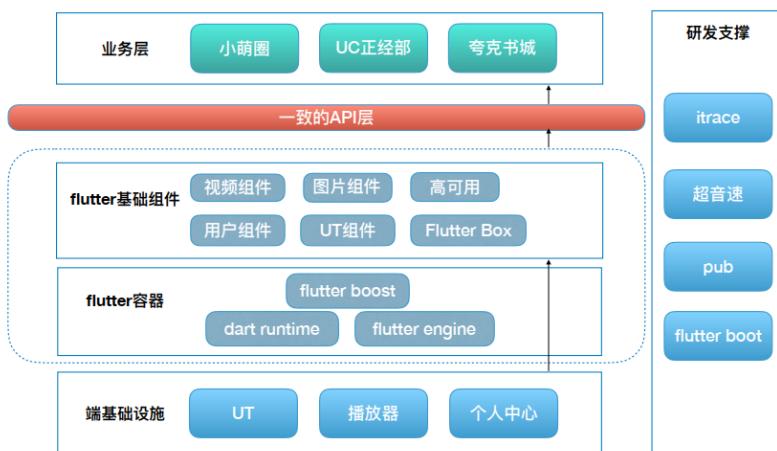
UC 通过上述业务架构约束，使研发认知一致聚焦，降低研发上手的成本。

## 4. 分层架构

UC 在业务层下面构建了容器。容器最底层包含端基础设施，有阿里巴巴集团和 UC 沉淀多年的移动组件。这些组件将被包装为 Flutter 基础组件提供给业务。Flutter 基础组件、Flutter 容器、端基础设施可以打造一致的 API 层提供给业务层去做接入。

好处是开发新 APP 时，一致的 API 层均可以复用。

根据分层架构可以做进一步思考，一致的 API 层能否更方便地进行业务迁移？例如 UC 正经部是在 UC 里做的，是能否将其方便地孵化为一个创新 APP。UC 正经部的功能是使用 Package 依赖，能否方便地将其提取出来放到其他 APP 中。这是一个比较理想化的实现，但其方向是可以研究的。



UC 通过以上三种架构模板解决 Flutter 落地到业务的效率问题。后续会随着社区和业务的发展，进一步优化架构体系。

### 三、性能优化

#### 1. 核心目标——解决业务痛点

做业务高可用建设。

引擎启动速度优化。

视频外接纹理方案优化。

图片组件优化。

## 2. 业务高可用

UC 主要做了两件事。

**指标体系：**一是构建了一套指标体系，与 Web 和 Native 的指标模型一致。有以下指标项。

页面打开性能：页面首帧绘制、内容首帧、首屏可交互。

页面流畅度：平均帧率，卡顿帧率。

页面稳定性：白屏检测，异常收集，网络请求异常。

UC 的高可用组件在闲鱼高可用基础上进行了升级，增加了一些指标，并支持原生 Navigator。

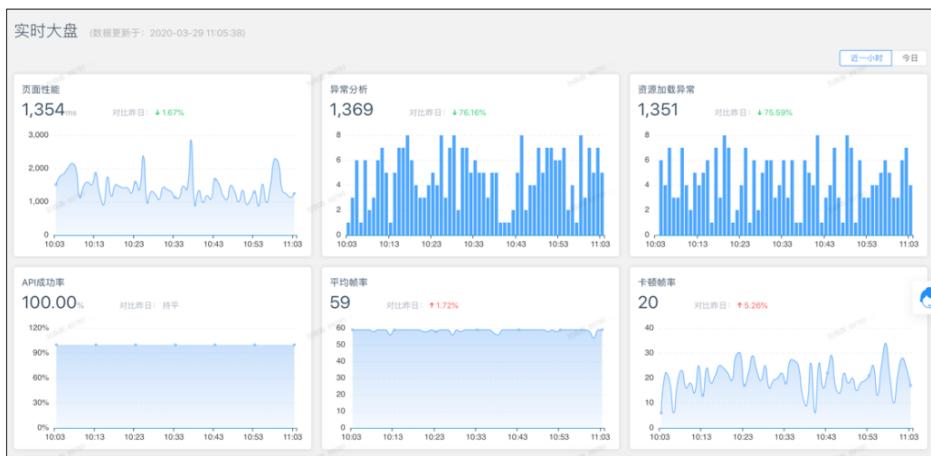
**itrace：**二是将收集到的数据上传到了实时监控平台 itrace 上。itrace 是 UC 使用了多年的实时监控平台服务。目前 itrace 对内、对外可接入。

对外：<https://yueying.effirst.com>

对内：<https://yuque.antfin-inc.com/wpk/help/ppxlrq>

itrace 主要有以下优势。一，分钟级别的实时性。二，提供多维度的分析方式。三，支持聚合，快速定位 top 问题。四，可配置预警规则。

**Flutter 实时大盘：**如下图所示（样例），以实时图表形式展现数据。



高可用展示使用样例，非线上真实数据

**高可用——页面性能：**页面打开流程如下。首先监听路由跳转，在路由起始时打一个起始点，然后监听每一帧的回调。第一帧回调视为首帧，打一个点，继续检测，直到图片、文字、视频等 Render Object 出现，打一个内容首帧。内容首帧贴近用户视角。继续检测直到组件覆盖度大于 60%，打一个首屏可交互。

该逻辑中还添加了白屏检测，可从 dart 层检查是否有文字、视频、图片。

页面性能在 itrace 上的展示形态如下。将内容首帧、可交互时间全部展示出来。下方是其实时曲线。可以选择分钟级别、小时级别、天级别。每一个业务下可能有很多个页面，可以具体查看每一个页面性能。

页面帧率使用 handleBeginFrame 和 handleDrawFrame 计算每一帧的耗时。通过算法可以计算出平均帧率与卡顿帧率。



目前页面性能仅支持 UI Thread 的监控。handleFrame 方案是实时的，可在发生卡顿时还原一些现场。

**高可用——维度分析：**可以根据网络类型、运营商、平台、客户端、页面版本、地区等设置维度来查看。例如将 WiFi 设为筛选，就可以展示出 WiFi 下的性能。



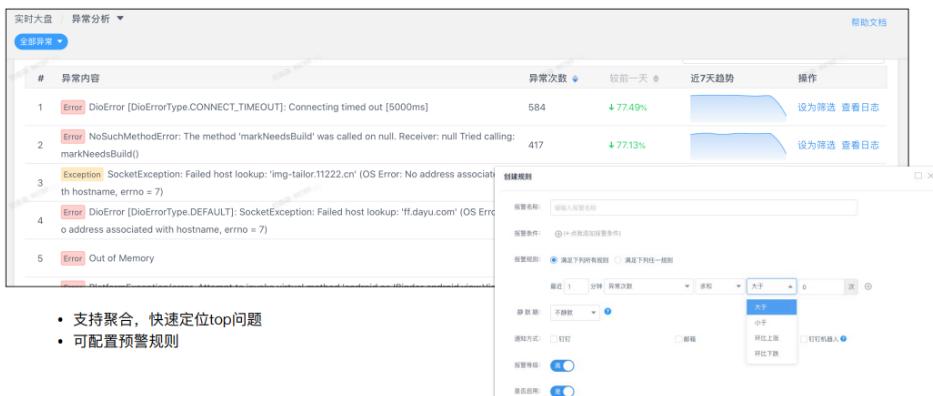
**高可用——异常收集：**此处指 Dart 异常收集，不包括引擎的异常收集，引擎异

常崩溃走 Native 高可用。Dart 异常重可导致白屏，轻可导致局部白屏。

使用 Dart 提供的 runZoned 进行异常收集。使用 HttpOverrides 做网络接口的异常采集。

异常展现在 itrace 上，可以按照一定的次数进行聚合，根据优先级逐个解决。另外，异常收集可配置预警规则。

- 使用 dart 提供的 runZoned 进行异常收集
- 使用 HttpOverrides 做网络接口的异常采集



### 3. 引擎启动优化

**问题：**通过高可用监控发现页面打开的最大问题是耗时，根本原因是引擎初始化时间长。发现如下两类问题。

一，Flutter 初始化时函数耗时长。例如 iOS，在主线程执行创建 GL Context 占用了 30% 的时间。在子线程创建 RootIsolate 占用 40% 时间。页面渲染之前的 Font Init 模块耗时也较长。测试不同机型耗时 100ms~200ms。

二，Flutter 的主线程等待子线程执行完毕后才释放，浪费 CPU。

**一般优化策略：**一是裁剪，裁剪对引擎的侵入性较大。二是异步化，最大程度利用 CPU。若异步化做得详细，侵入性也较大，因此最合理方案是针对一两个较为耗

时的问题进行优化。三是将一些耗时流程延后或提前。例如在 Flutter 创建引擎之前进行预热。只要不是首屏就用到 Flutter 的场景，预热是有效的优化方式。

**UC 优化方案：**一，在子线程预创建 GL Context，引擎初始化时将复用 GL Context。二，在子线程预热 RootIsolate。预热之后为了不占用内存将其关闭。三，异步化。四，优化 Font Init。Font Init 耗时的核心原因是做了多次 dlopen，可通过添加变量控制 dlopen 做一次即可，这个改动后续会进行更加优雅的调优，目标是回到社区。

**效果：**预热的内存占用仅 1.4M，相比预热整个引擎需要 16M 有很大优势，适合混合式开发，例如 UC 和手淘。预热的耗时性能提升了 62%，异步方案在部分机型上有约 100ms 的优化。Font Init 优化后在安卓部分机型耗时优化 50ms~90ms。



**内存占用：**预热只消耗 1.4M，相比预热整个引擎需要 16M 有很大的优势，适合混合式开发，特别是大型成熟 app 例如 UC 和手淘

#### 效果

**耗时优化：**

预热 363ms->137ms 提升 62% (iPhone release 版本测试数据)

异步方案在部分机型上有 100ms 左右的优化

Font init 优化 Android 部分机型优化 50ms~90ms

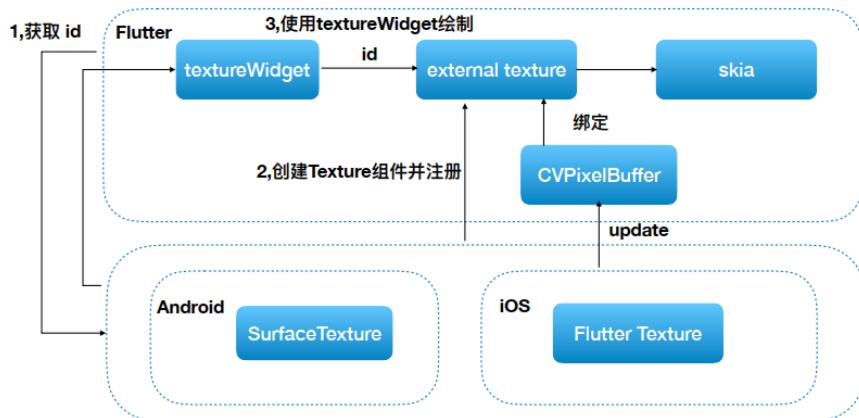
## 4. 视频外接纹理

UC 在视频、图片组件使用的是外接纹理方案。

**外接纹理原理：**通过三个步骤完成外接纹理流程。第一，Flutter 通知 Native 获取 Texture id。第二，Native 同步创建并注册 Texture 组件给 Flutter。第三，

Flutter 创建 textureWidget 通过 Texture id 找到 Texture 组件，向 skia 绘制。

目前 iOS 和安卓的外接纹理流程存在差异。安卓使用到 Surface Texture，iOS 使用到 Flutter Texture，需要 Native 回写 CVPixelBuffer，Flutter 内部会将 CVPixelBuffer 绑定到 Texture，当 CVPixelBuffer 数据发生变化，Texture 会向 skia 进行绘制。

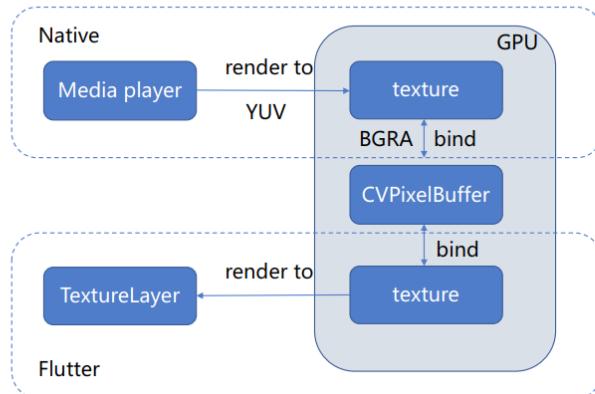


**问题：**大部分解码出的视频格式是 YUV，但 Flutter 内部 CVPixelBuffer 需要 BGRA 格式，需进行格式转换。如何使视频格式转换不影响性能？

**优化：**优先考虑在 iOS 端不修改引擎，通过高速纹理在 GPU 上把 YUV 转换为 BGRA 格式。Flutter 内部 CVPixelBuffer 绑定 Texture 时使用到 Binding API，相反，通过 Binding API 也可以将 Texture 绑定到 CVPixelBuffer 上。那么向 Texture 绘制 YUV 数据时设置 BGRA 的转换，这样两个 Texture 就通过 CVPixelBuffer 实现共享。该格式转换不耗时，并且不会使视频出现卡顿等问题。

**问题：**解码出来的视频格式是YUV，但Flutter需要BGRA格式，需要转换。如何做到不影响性能呢？

**解法：**iOS端不改引擎，通过高速纹理在GPU上把YUV转换成BGRA



**问题：**Flutter 在外接纹理时使用默认的 NEAREST 算法，会将纹理附近的几个颜色进行加权平均使用，导致弧形斜线等锯齿明显。

**优化：**已经给官方提 PR: <https://github.com/flutter/flutter/issues/53080>

Flutter在外接纹理时使用了默认的NEAREST算法，导致锯齿明显  
已给官方提PR: <https://github.com/flutter/flutter/issues/53080>



## 5. 图片组件优化 – 外接纹理方案

**问题：**Flutter 和 Native 内存无法共享。

原生方式多图滑块的场景内存回收不及时，低端机会崩溃。

侧面反映出 Flutter 目前图片场景做的并不够优秀。

**收益：**图片组件场景可以享受 Native 组件库成熟的优化手段，例如 LRU 回收，多级缓存，线程池等。以及更多的图片格式支持，如 iOS 的 HEIC 格式。

**效果：**内存占用较 Flutter 原生实现减少 30% 左右。内存回收表现更加及时。



## 四、动态性探索

**业务需要：**UC 等以 feed 流业务为主的，社区类的产品，产品框架比较固定，内容的动态性最为关键，即需要提供给用户好看、热点的内容。在端侧内容入口的展现形式要与内容相匹配，例如运营位置、卡片样式，这部分是有动态性诉求的。而内容打开后的动态性更希望以 Web 方式解决。

**业务诉求：**需要轻量级的动态卡片解决方案。

**解决方案：**Flutter box 项目，UC 希望技术方案更加面向 Flutter 社区，选取 Dart 作为 DSL 描述。Dart 有三点优势。

第一，Dart 对 UI 的描述是结构化的。例如下方左侧代码，是 Dart 写法，通过组装式 API 编写对 UI 的描述。该模式的好处一方面是在 AST 转化为需要的协议时非常贴近、自然，令一方面对于模板开发者而言是自然的写法，无需做太多约束。

第二，Dart 可一体式开发，从模板编写、预览到解析执行。

第三，支持 Dart 语法的表达式。例如下图右侧表达式，定义了一个 create 函数，可在 build 时调用。create 函数声明 List 的对象，通过一次 for 循环创建 Widget 将其返回给 build 的 container。实现了动态创建 Widget 树的方案。

**业务诉求：**轻量级的动态卡片解决方案

**解决方案：**Flutter box，使用Dart作为DSL描述

- 1.Dart对UI的描述是结构化的
- 2.可一体式开发，从模板编写，预览到解析执行
- 3.可支持Dart语法的表达式

```
@override
Widget build(BuildContext context) {
  return new Container(
    width: 170.0,
    height: 100.0,
    decoration: new BoxDecoration(
      borderRadius: new BorderRadius.circular(8.0),
      color: new Color(data['color']),
      image: new DecorationImage(
        image: new NetworkImage(data['url']),
      ), // DecorationImage // BoxDecoration
      padding: new EdgeInsets.fromLTRB(10, 8, 0, 0),
    ),
    child: new Text(
      data['title'] + i,
      style:
        new TextStyle(color: new Color(data['textColor']))
    ), // Text
  ); // Container
}
```

UI描述

```
List<Widget> create() {
  List list = List();
  for (int i = 0; i < 10; i++) {
    list.add(createWidget());
  }
  return list;
}

Widget createWidget() {
  return Text('hello world');
}
```

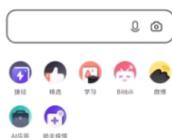
表达式

**效果：**将 Flutter box 方案落地到夸克书城，一方面提升了运营能力，一方面提升了页面加载、滑动速度，给业务带来的正向反馈。

根据测试所得性能数据，转换耗时 100ms，滑动帧率 50fps。转换过程指文件首先在端侧解析为中间树，为其绑定数据，再将中间树转换为 Dart 需要的 Widget 树。业务数据表现为书城点击率提升 2%。



## Quark



- 性能数据：转换耗时100ms，滑动帧率50fps

- 业务数据表现：书城点击率提升2%



## 五、总结

### 1. 发展阶段

现阶段 UC Flutter 处于规模化落地阶段。

### 2. 工程构建体系

搭建了标准的工程架构。为创新业务提供了可复用的运行时容器。抽象业务核心模块，使研发具备一致认知。

### 3. 性能优化

构建了性能监控体系，针对核心场景的视频、图片性能和体感进行了优化。并且性能优化方案对 Flutter 原生引擎修改尽可能少。

### 4. 动态性探索

从业务需要出发，落地了轻量级动态卡片，满足了运营能力。

## 5. 未来

UC 将更加深入地挖掘 Flutter，包括对引擎的优化，充分体现出 Flutter 在效率、性能方面的优势，为业务赋能。

### 总结



现阶段的UC Flutter：规模化落地



由阿里云开发者社区志愿者吴晨雪整理。

## | 基于 Flutter 的 Canvas 探索与应用

**摘要：**目前在小程序互动场景下遇到的业务痛点，并且给出了基于 Flutter 引擎的解法。基于 Flutter 引擎，对外提供标准的 Web Canvas API 和并利用 flutter 渲染管线，让业务代码在小程序 worker 线程中直接渲染，缩短了渲染链路，提高了渲染性能。本次分享将由淘宝技术部无线开发专家万红波为大家分享目前在小程序互动场景下遇到的业务痛点，以及基于 Flutter 引擎的解法。

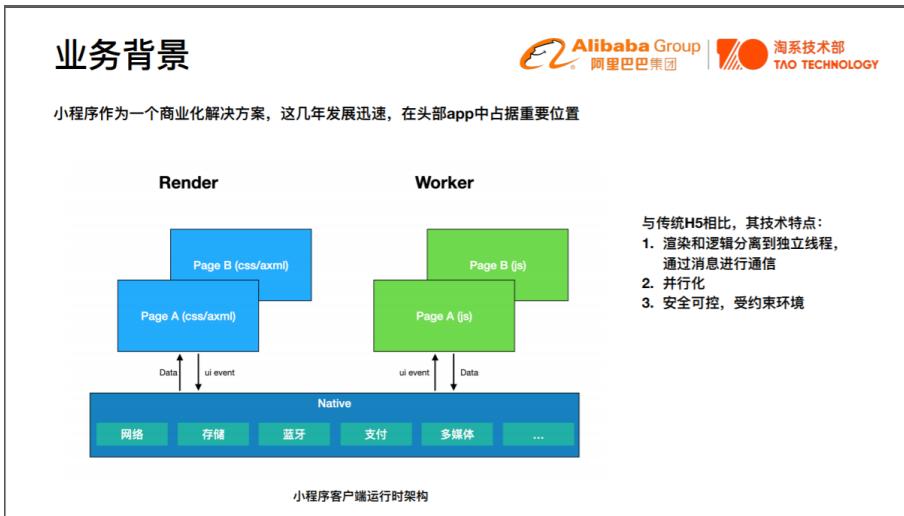
**演讲嘉宾简介：**万红波，花名远湖，淘宝技术部无线开发专家。主要从事浏览器内核以及渲染引擎方向的研究。目前在 AliFlutter 团队主要负责底层引擎以及 Canvaa 相关研究于应用。

以下内容根据演讲视频以及 PPT 整理而成。

本次分享主要围绕以下六个方面：

- 业务背景
- 业务痛点
- 解决方案
- 技术改造
- 业务效果
- 未来规划

## 一、业务背景



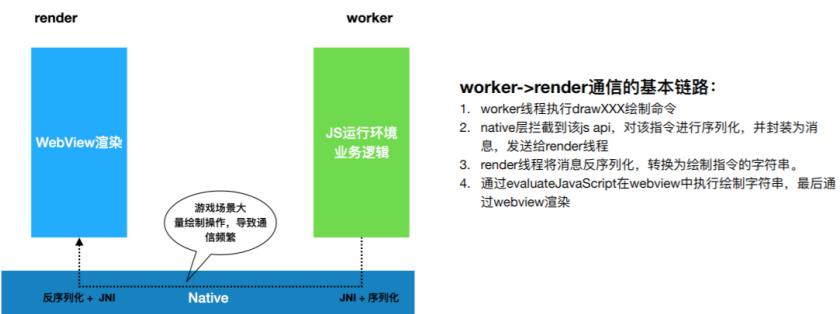
小程序作为一个商业化解决方案，这几年发展迅速，在头部 APP 中占据重要位置，与传统 H5 同，渲染和逻辑分离到各自独立线程，可以做到并行化。Worker 和 Render 之间通过 PostMessage 或者 SetData 来通信。

## 二、业务痛点

### 业务痛点

#### 小程序的架构给互动游戏业务带来的痛点：

业务通常使用Canvas API进行每帧的绘制，每一帧通常包含大量的绘制操作，worker和render通信频繁，渲染性能大幅降低，用户体验较差



小程序的架构可以满足大部分场景，但在互动场景下，业务通常使用 Canvas API 进行每帧的绘制，每一帧通常包含大量的绘制操作，worker 和 render 之间通信频繁，渲染性能大幅度降低，用户体验较差。其中耗时较大的点在数据的多次序列化和反序列化且多次通过 JNI 来实现 Java 和 C++ 的数据传递。在微信小程序和支付宝小程序文档中，提供了一些最佳实践来减低影响，比如 PostMessage 或者 SetData 的接口，一秒内调用不可超过 20 次，且对 SetData 的数据大小有所限制。但对于互动业务来说，要求每秒 FPS 的帧率，且每帧都需要大量的绘制指令需要传递，则很难通过这种方式解决。

### 三、解决方案

#### 解决方案（1/4）

在小程序 worker 环境中直接对接渲染引擎，缩短渲染链路，减少通信消耗，提高渲染性能

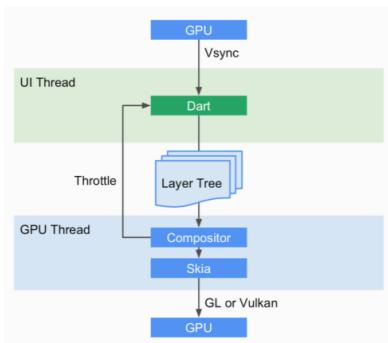


我们的解决方案是在小程序 worker 环境中直接对接渲染引擎，缩短渲染链路，减少通信损耗，提高渲染性能。Worker 中对接独立的 2D 渲染引擎，不再通过 render 和 worker 传递消息。而其他的页面，由于通信并不频繁，仍使用原来的链路渲染。

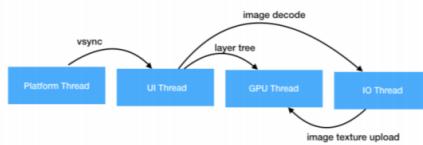
## 解决方案 (2/4)

独立2D渲染引擎方案：flutter引擎

Flutter渲染流水线：



Flutter线程模型：



选择Flutter的原因：

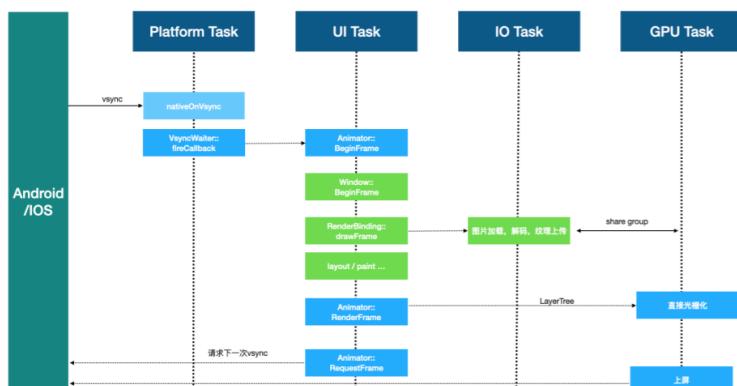
1. 跨平台
2. 高性能
3. Skia

我们选择了 Flutter 引擎作为独立的渲染引擎。我们选择 Flutter 的原因：

1. 跨平台的特性，客户端包含安卓和 IOS，忽略客户端细节，仅完成渲染，而 Flutter 引擎可以很好的屏蔽客户端细节；
2. 高性能，和传统的 H5 渲染引擎相比，是一个轻量型的多线程 2D 引擎；
3. Flutter 内部使用 Skia 作为渲染引擎，其兼容性、性能和适配性已在生产环境中被多次检验，安全稳定。

## 解决方案 (3/4)

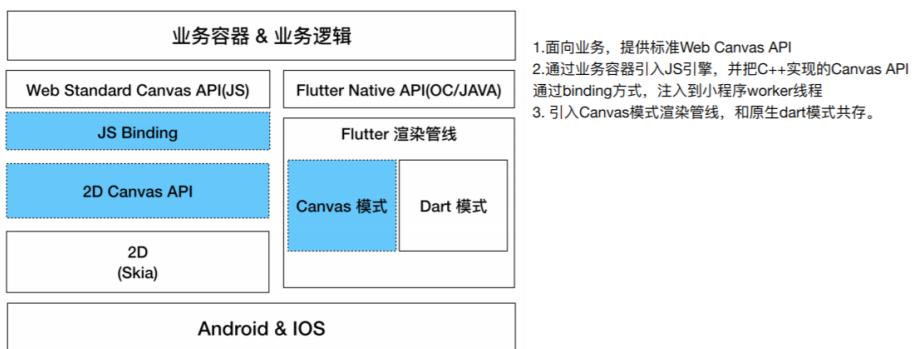
Flutter原生渲染管线



Flutter 渲染管线较短，平台层发出 vsync 信号，丢到 Platform task，调用 UI 线程已注册的 callback，即 Animator 的 BeginFrame 函数，其将调用 window 的 BeginFrame 函数，window 对象在 Flutter 中较为特殊，其连接上层 Framework 和下层 Flutter engine。图中绿色的区域即为 Dart 需要完成的事情，当 BeginFrame 进入 Dart 层之后，dart 层调用 Render 的 DrawFrame 函数，做 layout 和 paint，paint 过程中图片相关的部分会传递给 IO 线程抑或请求图片解码，同时在 GPU 线程上传纹理。上述完成后，调用 RenderFrame 函数，实现上屏。其中 Dart 层的绘制指令被组织为 Layer Tree 这种数据结构，它后续被传递给 GPU Task，GPU Task 读取 Layer Tree 中的渲染指令，直接光栅化，之后上屏。RenderFrame 做下一次的 vsync 请求。

## 解决方案 (4/4)

我们的方案：

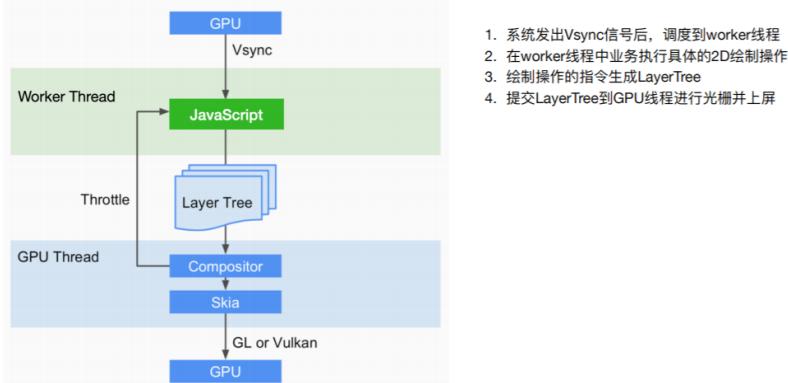


最终方案：面向业务，提供标准的 Web Canvas API，业务感知不到下层引擎变化；通过业务容器引入 JS 引擎，并把 C++ 实现的 Canvas API 通过 binging 方式，注入到小程序 worker 线程；引入 Canvas 渲染管线，和原生 dart 模式共存，提高性能和内存使用率，业务可灵活切换。

## 四、技术改造

### 技术改造 (1/3)

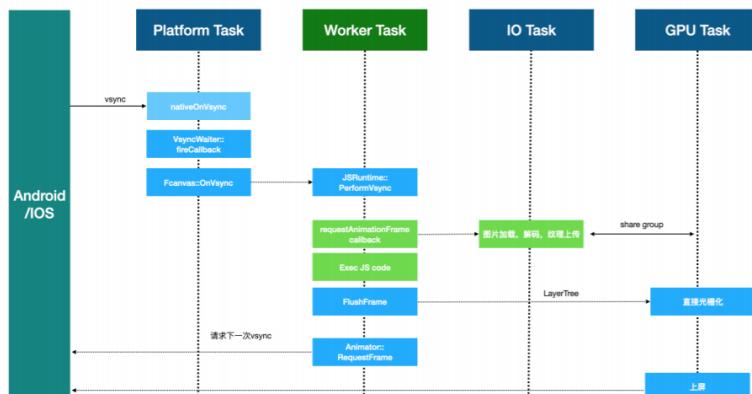
渲染流水线改造



方案对渲染流水线有较大的改造，原先 platform 线程接收到 vsync 之后，将其传递给 UI 线程，UI 线程在 Dart 层做绘制操作。而改造后在小程序环境中，用 JS 逻辑代替 Dart 层，业务逻辑在 worker 线程中完成，vsync 被传递给 JS 线程，JS 线程收到 vsync 后，驱动业务执行 JS 代码，进行绘制并生成渲染指令生成后依然生成 layer tree，传递给 GPU 线程，光栅化完成后上屏。

### 技术改造 (2/3)

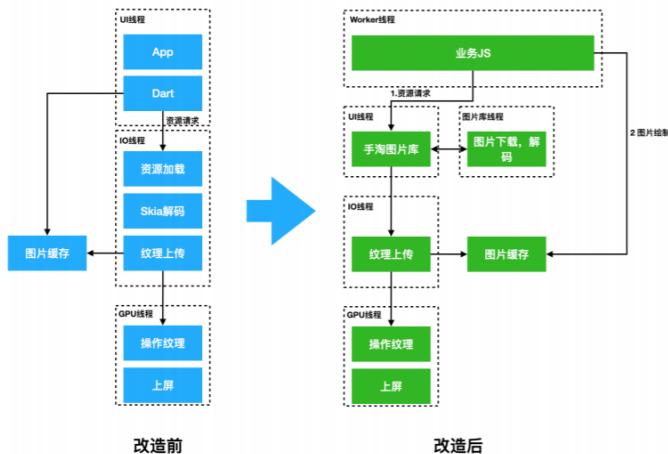
Canvas渲染管线



改造后的渲染管线，与改造前不同的是：platform 线程收到 vsync 信号后，不再调用原先的 callback，而是调用我们自己注册的自定义 callback，即 OnVsync，将 Vsync 传递给 JS 线程。JS 线程主要执行 requestAnimationFrame 的 callback，对屏幕进行绘制更新，JS 线程执行完成后，调用 FlushFrame，将 layer tree 传递给 GPU 线程，直接光栅化后上屏。绘制结束后，通过 Animator 的 RequestFrame 函数请求下一次 vsync。

## 技术改造（3/3）

图片加载流程改造



图片加载流程改造，原先的图片加载流程并不适用小程序环境下，JS 线程中的资源请求被丢给平台的手淘图片库，完成图片的下载和解码，解码后的数据传递给 IO 线程。IO 线程生成 GPU 纹理，并将其缓存在 Flutter engine 中。业务直接使用引擎中已经生成纹理的图片，可直接上屏。

## 五、业务效果

### 业务效果 (1/2)

在Android低端机上，业务改造前后  
小程序进程数据对比：

	CPU	内存	FPS
改造前	60%	435~549M	15
改造后	55%	407~498M	42

注：

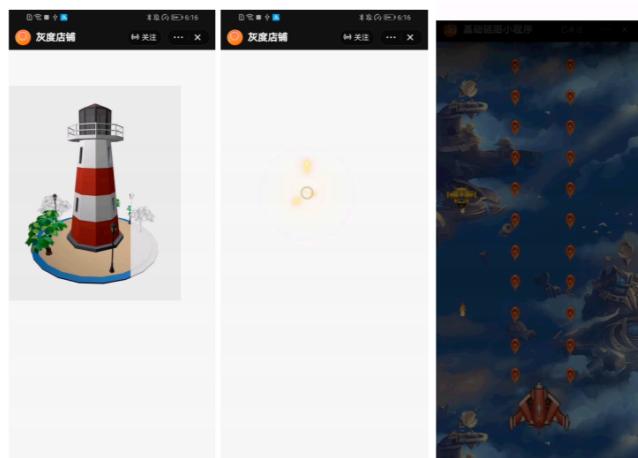
1. FPS数据由于运动银行业务没有通过Vsync 来驱动绘制，而是直接通过timer来驱动，使得CPU占用率较高，且改造后FPS提高很大。
2. 如果采用VSync来驱动绘制，通过线下测试，改造后FPS提高30%左右



安卓低端机上，业务改造前 FPS 为 15，改造后 42，效果明显。FPS 数据由于测试业务没有通过 Vsync 来驱动绘制，而是直接使用 timer 来驱动，使得 CPU 占用率较高，且改造后 FPS 提高很大。线下测试中，采用 Vsync 驱动绘制，改造后的 FPS 提高 30% 左右。

### 业务效果 (2/2)

1. 支持Lottie动画
2. 支持白鹭游戏引擎



目前我们还支持 Lottie 动画，设计师可以通过 Lottie 动画设计画面效果。并且支持白鹭游戏引擎。

## 六、未来规划

未来规划有三个方向，首先接入更多三方游戏引擎，和三方游戏引擎深度合作，其次探索高阶渲染 API 的实现。支持 WebGL API。第三探索独立小游戏容器，能够基于已完成的方案做更多的尝试。

由阿里云开发者社区志愿者张甜甜整理。

## 淘宝特价版开发体系的探索

**摘要：**淘宝特价版为了解决 app 自身的研发效率，用户体验问题，引入 Flutter 框架，并结合 FaaS 进行云端一体化融合。一个开发者可以在框架内顺畅的完成前端 + 后端的开发，相对于传统 Native 开发交付流程，节省一半以上的开发成本。本次分享将由淘宝特价版开发团队 iOS 高级开发工程师李彬为大家详细介绍淘宝特价版开发体系在探索过程中遇到的问题，以及基于 Flutter 和 FaaS 的页面构建方案。

**演讲嘉宾简介：**李彬，花名潇侠，特价版开发团队 iOS 高级开发工师。曾做过 Tangram 开源动态化框架，现负责特价版前后端一体化的开发模式升级，落地 Flutter。

以下内容根据演讲视频以及 PPT 整理而成。

观看回放 <http://mudu.tv/watch/5708706>

本次分享主要围绕以下三个方面：

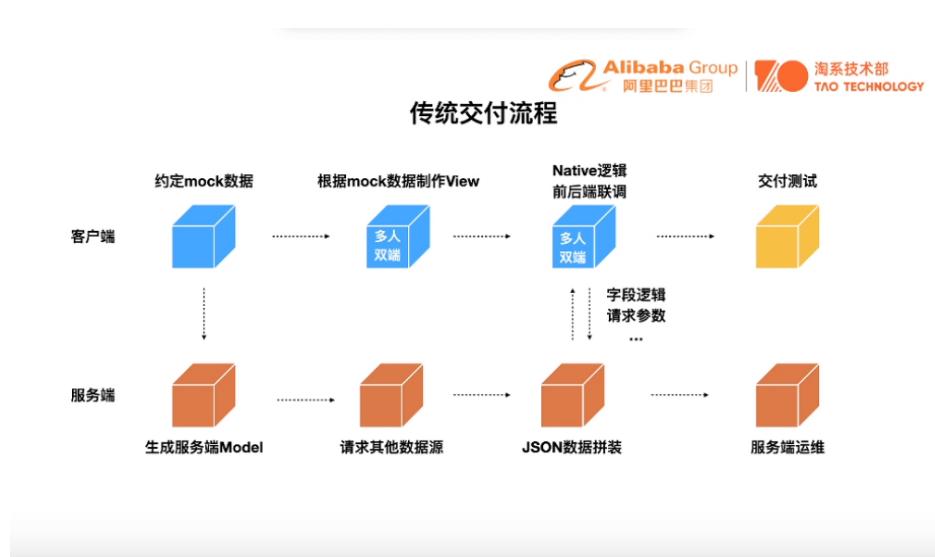
- 一、淘宝特价版遇到的问题和解决构想
- 二、基于 Flutter 和 FaaS 的页面构建方案
- 三、前后端一体化模式的进一步探索

## 一、淘宝特价版遇到的问题和解决构想



特价版业务开发遇到的问题

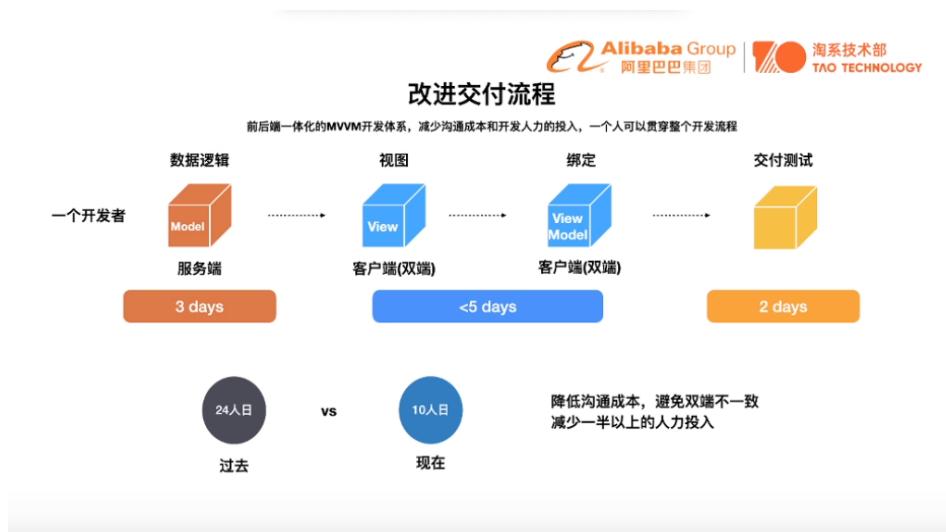
首当其冲的是研发效率，首先，传统的开发方式分为安卓、iOS、服务端，至少投入三人，人员投入多，交付效率低，当 iOS 和安卓人员分配不对等时，将引发资源分配的问题。其次调试慢，Native 的开发使用 xcode 或者 Android Studio 调试时，debug 一次需 30s 甚至更长，阻碍开发进程。然后是双端逻辑不一致，安卓和 iOS 由两个开发者负责，代码逻辑可能会有所不同，进入测试阶段会出现两端不一致的情况，导致重新开发。第二个是双端体验，安卓和 iOS 均投入人员优化，才能达到双端体验。需要解决帧率低、加载时间长、内存占用大的问题。最后是规模化，存在技术方案基础不统一、没有持久支持的社区、无法推广到全链路业务的问题，一旦更改其他方案，则之前所有的沉淀将推倒重来。



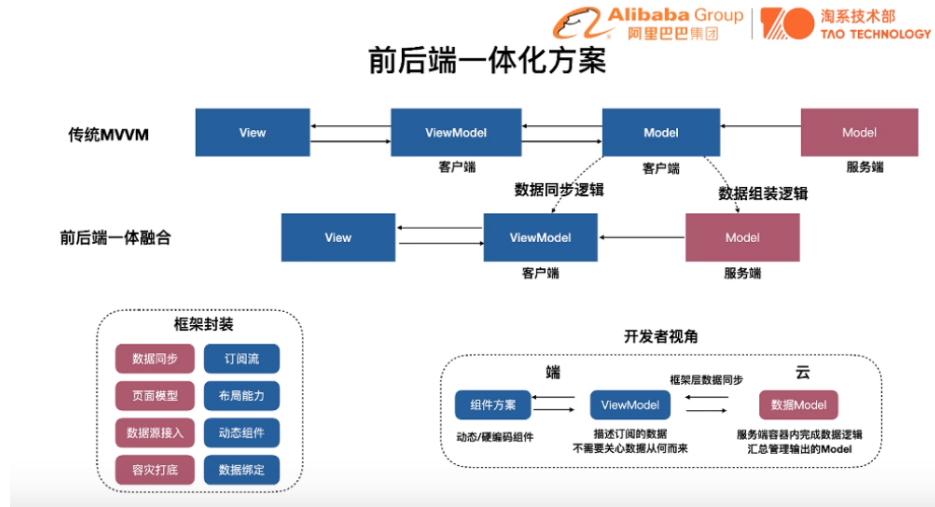
传统的交付流程，客户端和服务端约定 mock 数据，客户端根据 mock 数据制作 View，之后需要多人协作，安卓和 iOS 共同开发，完成 Native 逻辑，前后端联调。服务端在约定 mock 数据后，生成服务端 Model，然后请求其他数据源，然后完成 JSON 数据拼装。此时两端联调，结束后，客户端交付测试，服务端进入运维阶段（评估一个服务请求多少 QPS、RT 如何优化、需要多少台机器）。



传统的交付流程拉长需求时间线，例如开发阶段，iOS 需要五天，安卓需要五天，服务端需要三天，后面两天在等待，导致服务端时间线凌乱。联调阶段，服务端和客户端约定字段，当字段逻辑不一致时，两端需要互相沟通，确认字段逻辑，这部分会占据开发流程 30% 的时间。测试阶段，发现两端逻辑不一致时，又需要回溯开发阶段确认解决。整个流程存在 Native 调试慢，逻辑必须双端开发、前后端沟通成本高、双端逻辑不一致的问题，拉长了交付时间。

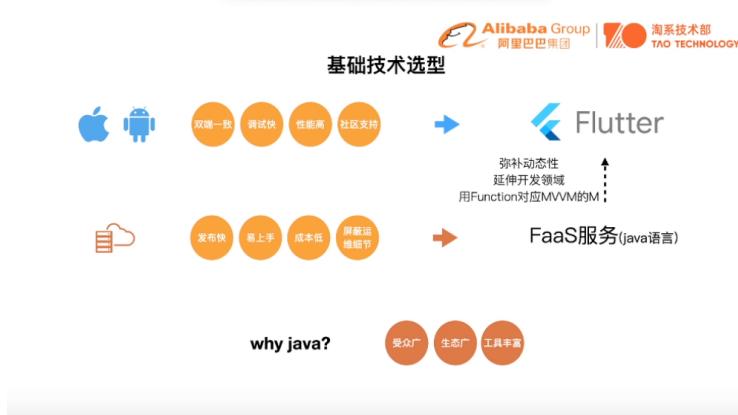


针对以上问题，改善交付流程，实现前后端一体化的 MVVM 开发体系，减少沟通成本和开发人力的投入，一个人可以贯穿整个开发流程。首先一个开发者，在服务端创建一个 Model (请求数据源并汇总)，接下来是视图，应用跨端技术，实现一个开发者开发双端，然后是客户端逻辑和视图绑定，即 View Model，将 Model 的数据处理后绑定到 View 上，实现解析流程，最后交付测试。整体过程有一个开发者负责，不需要安卓、iOS、服务端特定开发人员，降低沟通成本，避免双端不一致，减少一半以上的人力投入，从过去的 24 人日到现在的 10 人日。



为构建以上交付流程，制定了前后端一体化方案。抽象来看，传统的 MVVM，客户端和服务端割裂，客户端是一个单独的 MVVM 的模型，即 View 到 ViewModel 到 Model。服务端是 MVVM 或者其他框架，只需要输出 JSON 格式数据，和客户端的流程完全割裂。本方案期待将 MVVM 贯穿前端开发的流程。拆分客户端的 Model，传统 MVVM 中客户端的 Model 主要请求服务端或者 A\B\C 接口，并将三个接口合并为 model 类，传递给 ViewModel。数据的组装逻辑在服务端 Model 完成，数据的同步逻辑在客户端的 ViewModel 完成。而新的体系中，前后端整体是一个 MVVM 模型。为了完成一体融合，需要框架封装，服务端包括：数据同步、页面模型、数据源接入和容灾打底，客户端包括订阅流、布局能力、动态组件和数据绑定。于开发者视角，首先看到的是组建方案，即用怎样的组件完成编码（动态组件 / 硬编码组件），其次是 ViewModel，仅需要知道订阅的数据是什么，不需要关心数据从何而来（请求哪个接口，从哪里获取数据），最后是云端的数据 Model，服务器完成数据逻辑，汇总输出数据 Model。云和端通过框架层同步数据，数据协议对开发者完全屏蔽。

## 二、基于 Flutter 和 FaaS 的页面构建方案



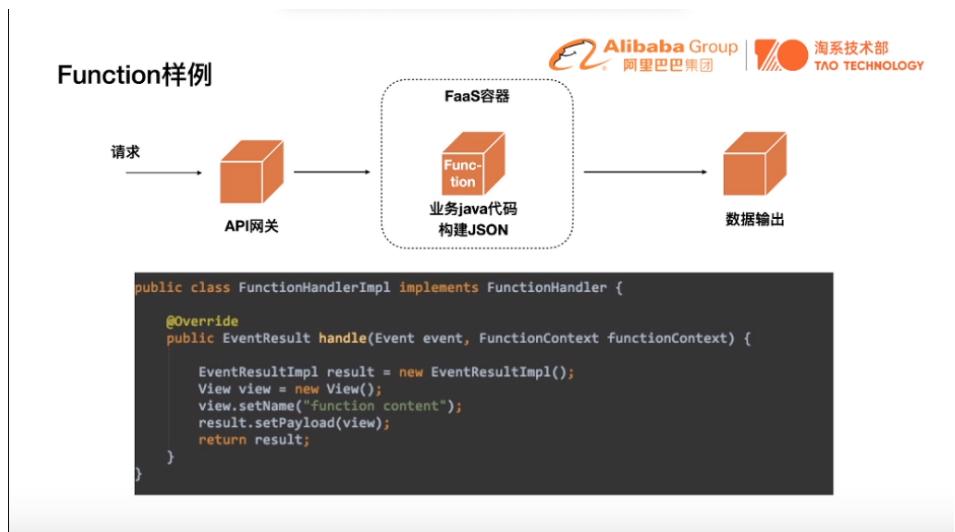
为了构建框架，客户端和服务端均需要基础支持。客户端要求双端一致、调试快、性能高、社区支持，选择 Flutter，云端需要发布快、易上手、成本低和屏蔽运维细节，最终选择 FaaS 服务 (Function as a service，简单来说是一个服务端容器)。FaaS 服务弥补 Flutter 无法动态的特性；延伸开发领域，客户端开发人员可以开发服务端需求；Function 对应 MVVM 中的 M，位于前后端一体化 Model 的位置。FaaS 容器支持多种语言，包括 Java、Python、Go，最终因受众广（开发人员一般均接触过 Java）、生态广（基本框架丰富）、工具丰富（IDE 工具，降低开发门槛）的特性，选择了 Java 语言。



在我们的体系中，Function是可以作为MVVM的M，这是我们选择FaaS最重要的理由

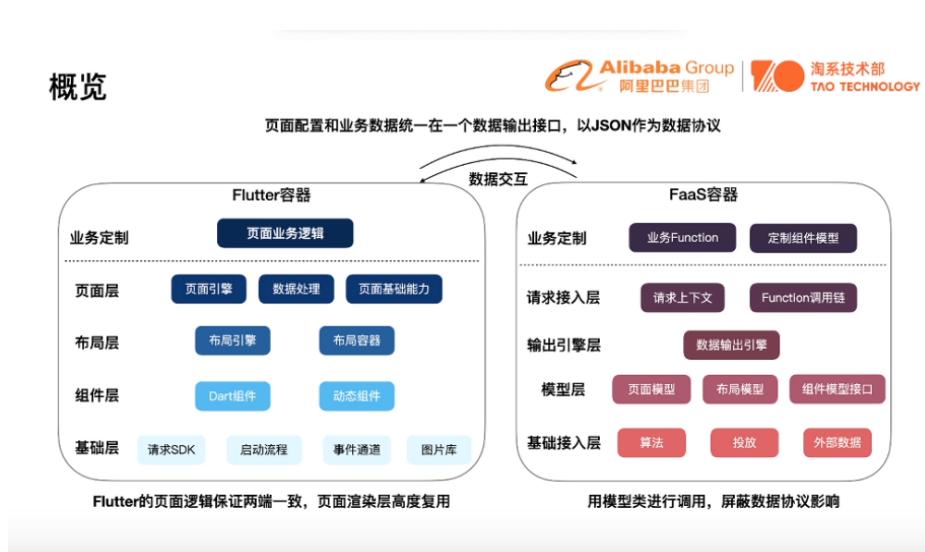
FaaS 相比于传统服务端开发存在这几个优势：首先服务端无需管理服务器，传统交付流程中，存在服务端运维的环节，需考虑 QPS、机器等，而使用 FaaS 后，减少开发人员需考虑的事情。其次是资源弹性伸缩、事件触发执行和不执行不付费。在淘宝特价版的开发体系中，Function 可以作为 MVVM 的 M，这是选择 FaaS 而不选择 PaaS 的最重要理由。

FaaS 功能强大，服务端所需要的功能在 FaaS 里面有所体现，包括函数计算、对象存储、API 网关、表格存储、日志服务和批量计算。函数计算的客户只需要编写代码并设置运行的条件，即可以弹性、安全的运行。



FaaS 的示例，一个请求到达 API 网关，FaaS 容器里面包含 Function 代码，实现数据输出。Function 即一个函数，包含输入参数和输出对象，不论是对于 iOS 还是安卓开发者，简单易上手。

以特价版首页介绍基于 Flutter 和 FaaS 的方案，第一阶段基于一个集中数据接口的方案，具有普适性，第二阶段持续演进的一体化方案。

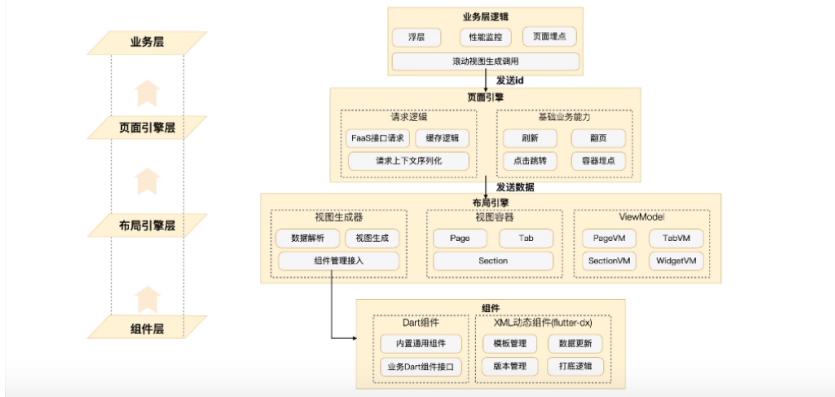


基于一个集中数据接口的方案，前端使用 Flutter 容器，后端使用 FaaS 服务，页面配置和业务数据统一在一个数据输出接口，以 JSON 作为数据协议。Flutter 容器分为四层，最底层是基础层，用于和基础技术对接，包括请求 SDK、启动流程、事件通信、图片库。往上是组件层，包含 Dart 组件和动态组件。再往上是布局层，包含布局引擎和布局容器。最上面一层是页面层，有页面引擎、数据处理、页面基础能力（例如埋点和点击跳转等）。基础层、组件层、布局层和页面层都是可以被其他页面复用，最终的业务定制层即页面业务逻辑，四层复用可以有效提高开发效率。

FaaS 容器底层是基础接入层，有算法、投放和外部数据。之后是模型层，和客户端对应，例如客户端的布局层、组件层、页面层，在服务端模型层都有对应，即页面模型、布局模型、组件模型接口。模型层之上是输出引擎层、请求接入层，最上面是业务定制的部分。服务端的一个特点是，业务方不需要操作底层 object 或 JSON，而是操作模型层的对象，可以屏蔽数据协议的影响。

## 客户端页面分层架构

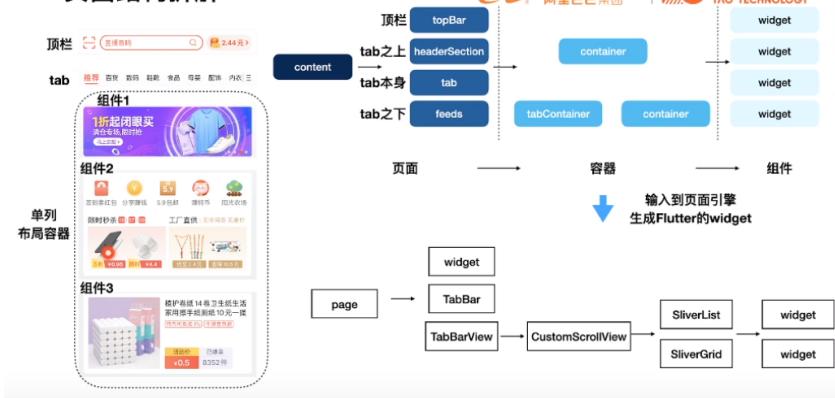
Alibaba Group | TAO TECHNOLOGY



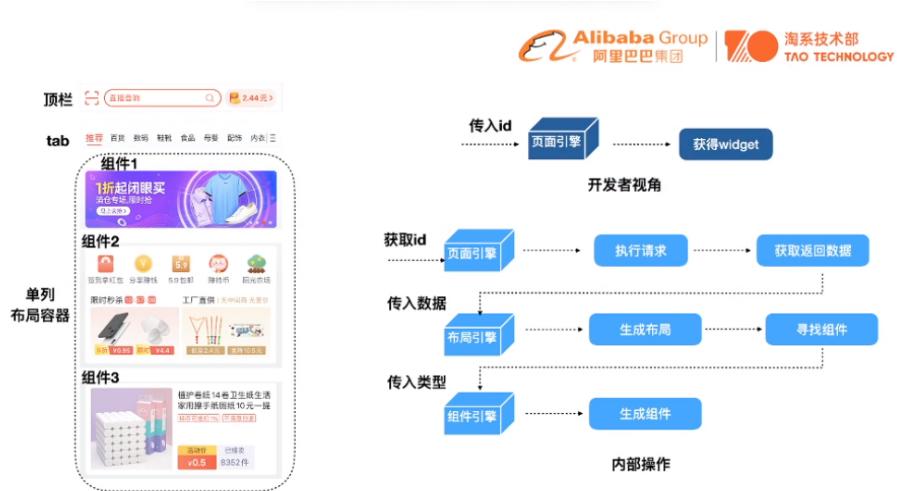
客户端页面分层架构，由下至上为组件层、布局引擎层、页面引擎层和业务层。在实际的调用流程中，业务层逻辑包含滚动视图以外的部分，例如浮层、性能监控和页面埋点，滚动视图以内的部分由页面引擎负责。页面引擎的入参为 id，通过 id 做请求，此时对接 FaaS 的接口、缓存、请求上下文序列化，在 FaaS 的 Function 入参获取请求上下文的内容。页面引擎同时负责基础的业务能力，即刷新、翻页、点击跳转和容器埋点。请求数据后获取 JSON 格式的数据，传输到布局引擎。布局引擎解析视图，生成 Page 对象、Tab 对象、Section 对象。生成视图后对接到组件，组件包含 Dart 组件和 XML 动态组件。

## 页面结构拆解

Alibaba Group | TAO TECHNOLOGY



淘宝特别版的页面结构可拆解为如上图所示。根节点 content，体现在数据中，之后页面被拆机为四个部分：顶栏 (topbar)、tab 之上 (headerSection，可以理解为后续在 tab 上插入图片或者组件)、tab 本身 (tab) 和 tab 之下 (feeds，feeds 中可能包含多个 feeds，tab 中有个类目，每个类目下存在多个容器，feeds 首先进入 tabContainer，之后进入 container，存在单列和瀑布流的形式)。页面到容器到组件，业务方拿到的是 page，page 即 Flutter 中的 view。page 包含 widget、TabBar、TabBarView 子节点，接下来是 CustomScrollView、Sliver 和 widget。Flutter 中连接 List 或者 Grid 极为方便，例如混合一行一列和一行两列，用 CustomScrollView 可以很好解决。



开发者视角，传入 id 给页面引擎，获得 widget。引擎层面有页面引擎、布局引擎、组件引擎，页面引擎执行请求，获取数据给布局引擎，布局引擎生成布局，寻找组件给组件引擎，组件引擎生成组件。业务方获取到 page 对象，可直接贴在页面。

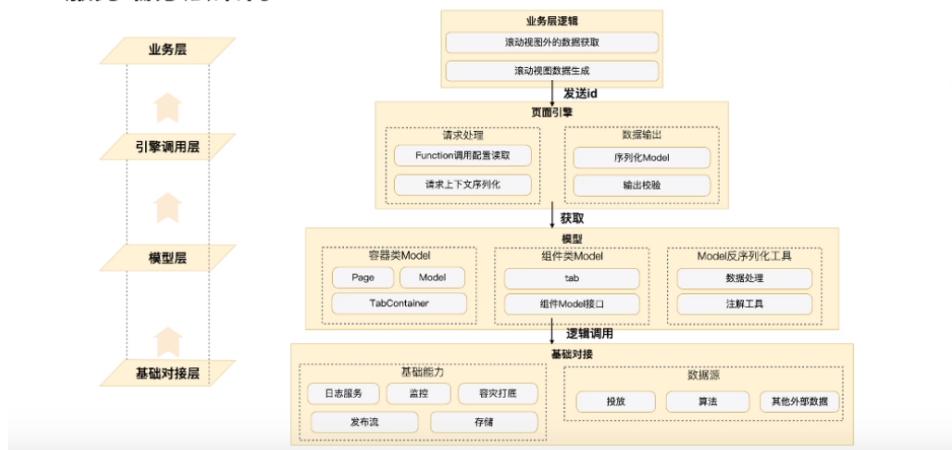
## 服务端方案



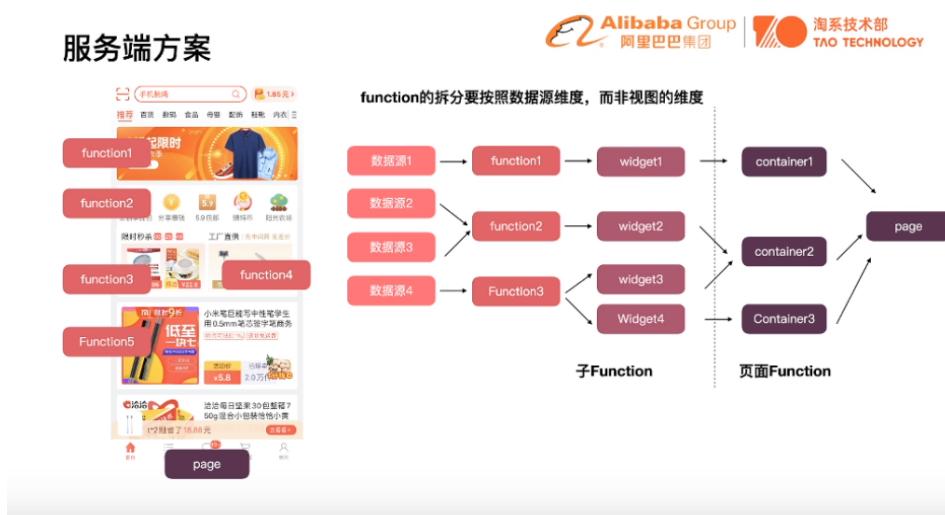
- 函数计算，使用Serverless平台，无需管理服务器资源，降低操作成本
- 选用Java语言写FaaS，降低上手门槛，可以使用Java的生态
- Function是可以作为MVVM的M，更好地实现前后端一体化
- 让客户端来写服务端，需要再服务端构建一套和客户端结合的框架

服务端方案使用 FaaS，有以下特性：函数计算，使用 serverless 平台，无需管理服务器资源，降低操作成本；选用 Java 语言写 FaaS，降低上手门槛，可以使用 Java 的生态；Function 是可以作为 MVVM 的 M，可以更好实现前后端一体化；让客户端来写服务端，需要在服务端构建一套和客户端结合的框架。

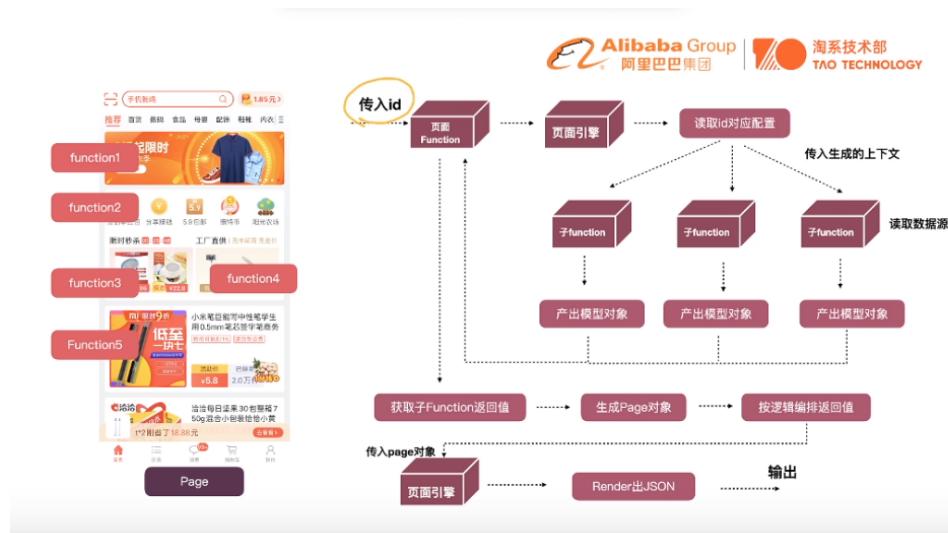
## 服务端分层架构



服务端分层架构由下至上为基础对接层、模型层、引擎调用层和业务层。和客户端类似，滚动视图之内的部分由业务层逻辑负责，页面引擎接收客户端发送的 id，通过页面 Function 调用链读取配置，做请求上下文序列化，之后 Function 获取具体组件的 Model、容器类 Model、Model 反序列化工具。底层是基础对接，包括日志服务、监控、容灾打底、发布流和存储等。



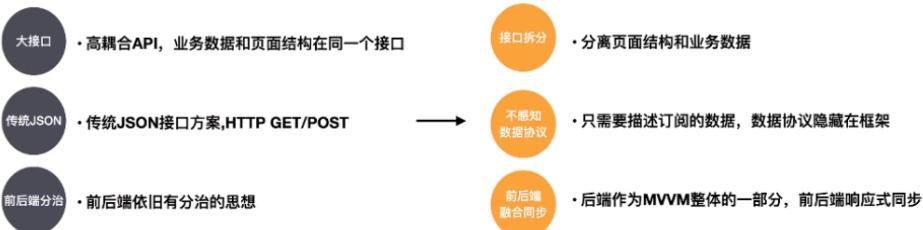
客户端所见即所得，即按照看到的页面（视图）做拆分，而服务端按照数据源做拆分。数源可能是一对一，多对一或者一对多的关系，需要具体情况具体分析。Function 对接的数据源数量不固定，接下来输出的 Function 模型对象即 widget（组件），页面 Function 组织组件，将其放入 container 中，最后 page 对象将其序列化输出到客户端。



首先页面 Function 接收 id，进入页面引擎后，读取 id 对应配置即需要读取哪些子 Function，子 Function 输出模型对象（组件对象），模型对象会被汇总到页面 Function 中，然后可以获取子 Function 的返回值，生成 page 对象，由业务方负责按照逻辑编排返回值，之后呈现出 JSON 给业务方。

### 三、前后端一体化模式的进一步探索

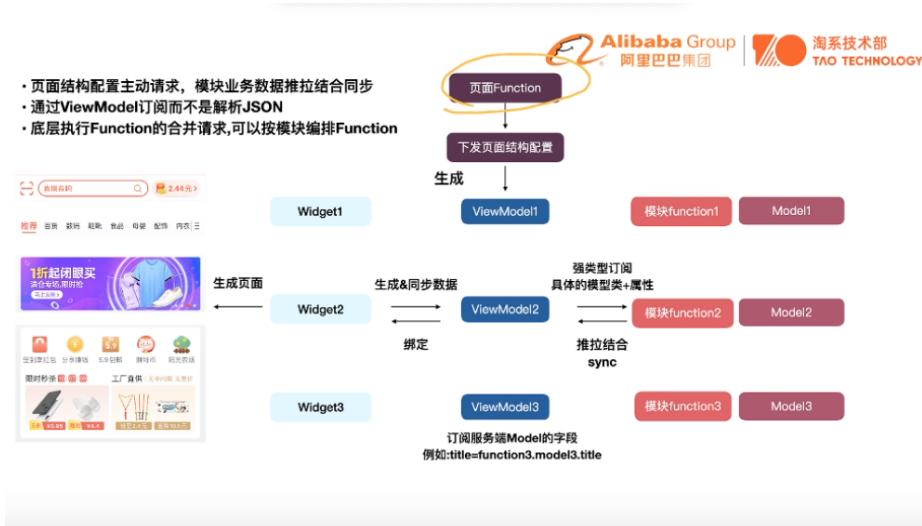
#### 前后端一体化的演进



上述方案基于统一的 API，存在以下问题：高耦合 API，业务数据和页面结构在同一个接口；传统 JSON 接口方案；前端依旧有分治的思想（客户端所见即所得，服务端根据数据源组织 Function）。接下来的演进方向，首先是接口拆分，分离页面结构和业务数据，第二是不感知数据协议，数据协议隐藏在框架，第三是后端作为 MVVM 整体的一部分，前端响应式同步。



Function1、Function2 和 Function3 对应到页面 Function，页面 Function 转化为 Page 对象，之后生成 widget。页面 Function 到 Page 承担着中枢的作用，一次请求必须更新所有的模块 Function，重复拉取占用流量并且带来更新不及时的问题，例如限时秒杀模块中，希望倒计时结束时刷新数据，期待结束时的数据已到达客户端。整体的耦合逻辑偏高，页面 Function 包含业务数据和布局描述。



首先拆分页面 Function 和模块 Function，页面 Function 下发页面结构配置，模块数据集中在框架内完成逻辑。ViewModel 订阅字段强类型模式，例如 title=function3.model3.title，仅需要关心订阅的数据是什么，无需知道从何而来，即具体的模型类和属性。ViewModel 到 Function 通过同步 sync 的推拉结合方式，需要的时候请求，某些时候服务端推送数据。模块 Function 存在不同的依赖，底层会执行 Function 的合并请求，可以按照模块编排 Function，实现更彻底前后端一体化的融合。最后是 widget(view) 的生成，同步数据到 ViewModel，ViewModel 绑定 widget，生成页面。以上是目前探索的方向，对 FaaS 的能力要求较高，不仅需要统一的 API，还需要推拉结合的能力，目前仍在探索，实现真正的前后端一体化。

由阿里云开发者社区志愿者张甜甜整理。

## I CBU Flutter 探索之路

**摘要:** ICBU 建设 Flutter 的核心目标是保障 Flutter 的性能和质量，进一步提升提效效果，进一步扩大提效范围。本次分享将由 ICBU Flutter 架构师路少德为大家详细介绍 ICBU 在 Flutter 实践中的思考和沉淀。整体分为两部分，第一部分通过业务背景和技术原理推导出接入 Flutter 的必要性和待解决的问题。第二部分以接入工作中的混合工程和混合栈为重点进行技术上的阐述。

### 演讲嘉宾简介：

路少德，花名白及，ICBU Flutter 架构师，将 Flutter 接入 ICBU，设计实现 ICBU 的 Flutter 架构和基础组件，实现和输出 ICBU 的混合栈、多语言等能力，并推动 ICBU 无线技术部持续朝 Flutter 化演进。

以下内容根据演讲视频以及 PPT 整理而成。

观看回放 <http://mudu.tv/watch/5817421>

本次分享主要围绕以下四个方面：

一、背景介绍

二、Flutter 技术接入

三、Flutter 能力补齐

四、结果 & 规划

## 一、背景介绍

### ICBU Flutter 现状

首先在场景覆盖层面，ICBU 目前有 30+Flutter 页面，交易业务覆盖 90%，目前所有新页面都选择 Flutter 进行开发。在人员层面，所有无线开发掌握 Flutter 开发知识，Android&iOS 间的壁垒正在逐渐消失。在提效效果层面，目前已累计提效大量人日，同时获得了不错的技术成果，如参与 AliFlutter 的建设，并提出了基于官方的工程体系和混合栈方案，及多语言、本地化等多种国际化方案。

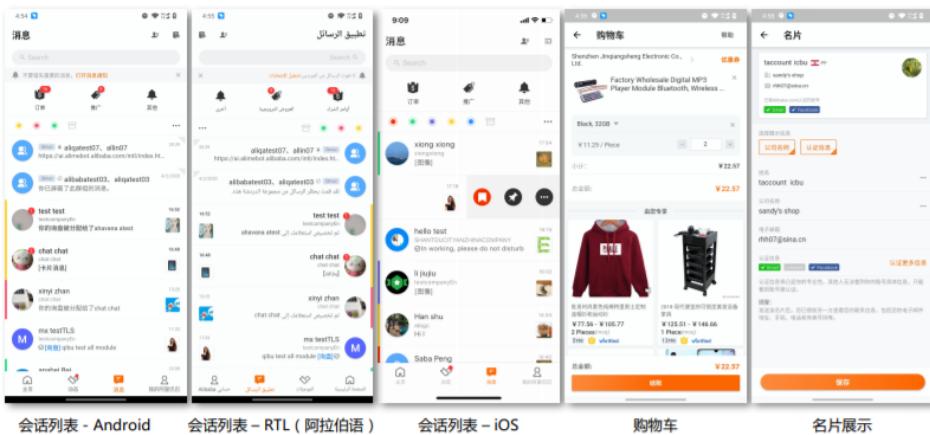
#### 背景 ICBU Flutter现状



下图是 APP 使用 Flutter 后的页面，包括 Android 版的会话列表、iOS 的会话列表、RTL（阿拉伯语）的会话列表，以及购物车等交互非常复杂的页面。

## 背景

ICBU Flutter 页面展示



## 为什么选择 Flutter

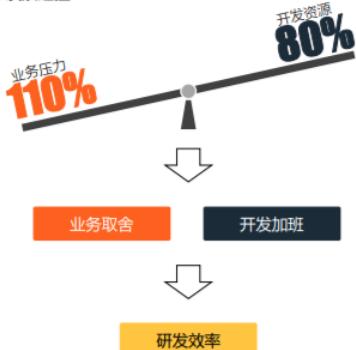
Alibaba.com 属于 B 类国际电商业务，B 类国际电商业务在当下的特点是发展非常迅速，业务压力很大，每次的业务需求都会超出人员需求，并且开发资源会被其他事情占用（如技术架构演化），此时不得不进行非关键业务的取舍或者加班加点赶工期。但这两种策略都不能从根本上解决问题，ICBU 希望的是通过提高研发效率，用更少的开发资源承担更多的业务需求，甚至可以解放生产力做其他事情。第二个特点是业务非常复杂，下图右侧展示了如果需要进行跨境 B 类贸易，则需要经过的 28 步流程。在如此复杂的业务场景下，由于 Android 和 iOS 开发对业务的理解的不同，导致业务实现出现差异，从而不得不耗费更多的开发资源进行对齐，所以 ICBU 希望有跨端一致性的框架解决这个问题。

**背景**

业务背景

**B类国际电商业务特点：**

发展迅猛

**业务复杂**

再进一步而言，ICBU 无线技术诉求是有一个取代 Native 开发的跨端框架。对于 Native 大家已经非常熟悉，它的特点包括极致体验，优秀性能，同时具备高稳定性。在此之前，ICBU 已经尝试过很多跨端框架，包括 H5、小程序、uni-app、Weex、以及阿里集团内部的 DynamicX 等动态组件，上述框架都有各自的优点和使用场景，但是各自都存在一些问题，如性能差、一致性差、存在能力缺陷、开发效率低等，都无法取代 Native 开发。

在 18 年左右，Flutter 进入了大家的视线中，下图右侧展示了 Weex、Native 与 Flutter 技术原理的对比，以 Android 开发为例，Native 布局的渲染就是将 Layout XML 变成了 Native View Tree，Weex 也是借助了类似的原理，即上层使用 JS，在渲染时经过几次 DOM 的转换，最终以 Native View Tree 进行页面的渲染。因为 Weex 相比与 Native 多一层 JS，所以性能上无法和 Native 一样，并且随着页面复杂度越来越高，与 Native 的性能差距也会越来越大。而 Flutter 运行在 Dart 虚拟机上，直接进行绘制，没有借助 Native View Tree。通过技术原理，我们可以推断 Flutter 具备完美一致性，另外就是性能上限比较高。

## 背景

技术决策

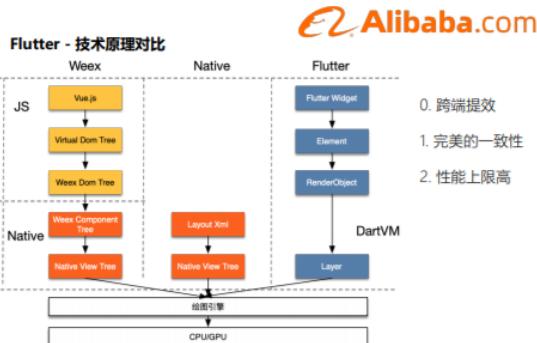
ICBU无线技术诉求：取代Native开发的跨端框架

Native的优点		
极致体验	高性能	高稳定性

不同的跨端框架		
Web渲染	H5、小程序、uni-app .....	
Native渲染	Weex、ReactNative .....	
动态组件	VV、DynamicX .....	

其他框架的问题			
性能差	一致性低	能力缺陷	开发效率低

结论：都有各自的优势和使用场景  
但都无法取代Native开发



### App Store政策风险的考量

Apple的核心利益	有风险的技术项	Flutter的应对
应用市场付费渠道	三方支付	无支付相关技术
应用市场存在的意义	动态化能力	无法动态化
iOS用户的体验	H5等页面滥用	Cupertino组件库，高性能

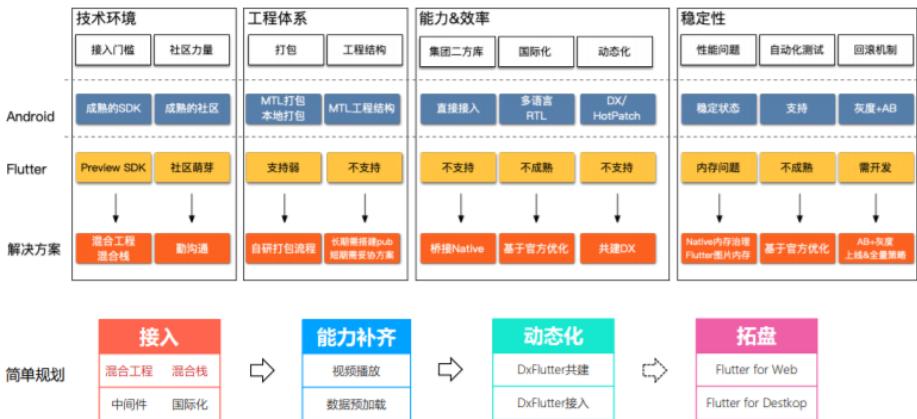
## 二、Flutter 技术接入

既然决定采用 Flutter，就需要弄清楚我们要解决什么问题，以及解决的方案。

下图展示了 Flutter 与 Android 的比对，并且划分了四个技术体系，分别包括技术环境，工程体系，能力 & 效率，以及稳定性。而且分别列出来 Android 和 Flutter 当时的状态，通过一一对比，就可以发现每一项问题及对应的解决方案，再将解决方案重新规划。如首先解决接入问题，即混合工程和混合栈问题；再针对 Flutter 薄弱点进行能力补齐，即视频播放和数据预加载能力；接下来是解决 Flutter 动态化能力；最后在前面的解决方案都顺利的情况下，利用 Flutter 跨端提效的特点，在更多应用上进行拓盘，如应用于卖家 App 或者 PC 上。

## Flutter - 架构设计

发现问题 – 解决问题



## Flutter- 混合工程

Flutter 开发大体分为两种，一种是原生开发，另一种是混合开发。原生开发适合开发新 App，混合开发更适合存量 App，有基建成本，但官方支持不够成熟。混合开发核心需求解决两个问题，即混合工程和混合栈。混合技术首先需要解决前三个问题：

- Native 工程如何源码依赖 Flutter 工程；
- Flutter 工程如何生成中间产物（如 Android 上是 aar，iOS 上是 Framework 等）；
- Native 工程如何依赖中间产物。

右侧是以 AliFlutter 为代表的工程结构，下层是 AliFlutter 基建，原则是将 AliFlutter 的基建都以产物的形式交付给业务团队。业务团队在上层只需要关心三个工程：Native 主工程、Flutter 主工程、Flutter 业务工程。

## Flutter- 混合工程

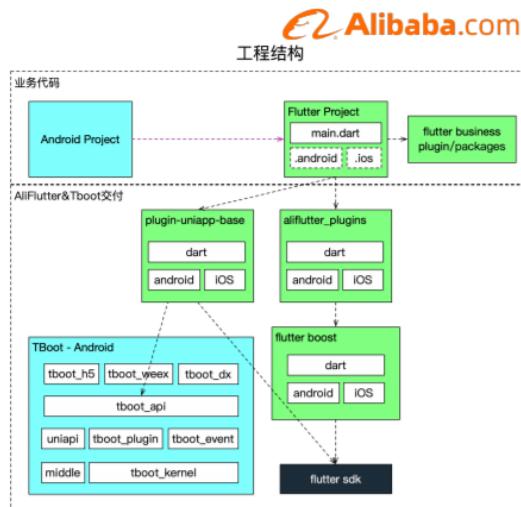
背景&工程结构

	原生开发	混合开发
工程体系	官方支持	官方不成熟
页面栈管理	官方支持	官方不成熟
总结	适合新App	适合存量App，有基建成本

从效率和集团技术现状来看，我们必须Native和Flutter混合开发。

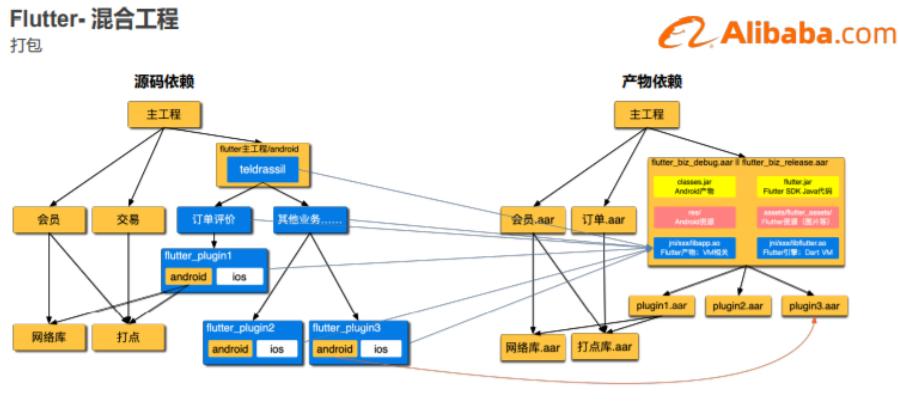
混合开发要解决的问题：

1. Native工程如何源码依赖Flutter工程
2. Flutter工程如何生成中间产物(aar等)
3. Native工程如何依赖中间产物
4. Native页面和Flutter页面的栈管理



## 打包

混合工程第二个问题是打包。下图左侧是在源码依赖情况下，工程结构的示意  
图。右侧是在产物依赖情况下，工程结构示意图。黄色部分是 Native 代码实现，蓝  
色部分是 Flutter 代码实现。从左至右，Native 部分代码没有很大变化，而 Flutter  
代码发生了翻天覆地的变化。所有 Flutter 代码都被打包一个产物，而 Flutter 插件  
所使用的 Android 代码或 iOS 代码被分门别类的打成独立的包。如果再细分，可以  
变成三个问题，首先是如何区分 release 和 debug 产物；其次是如何递归的打出插  
件的 Native 产物（图中黄色部分）；最后是如何一键依赖如此多的产物。通过 Shell  
脚本打出 release 和 debug 产物，再通过 Gradle 脚本打包 Flutter 产物，再通过  
Gradle 脚本注入的方式将每个插件的信息注入到每个插件的打包脚本中，如此便可  
以打出每个产物，且每个产物都配有 POM 文件，通过 POM 文件可以实现一键递归  
依赖。



打包的主要问题：

1. 如何产出和依赖release和debug产物
2. 如何递归的打出插件的Native产物
3. 如何优雅的一键依赖如此多的产物



**Shell脚本 + Gradle脚本 + Gradle脚本注入 + POM文件**

## Flutter- 混合栈

混合栈分为两个问题，首先是引擎复用方案，其次是 Native 页面及 Flutter 页面的约束方案。下图左侧是不复用引擎和复用引擎的区别。复用引擎无需在每个页面生成独立的引擎，从而减少引擎内存的占用，减少引擎创建时间，提高内存稳定性，并且减少页面启动时间。

复用引擎需要解决的核心问题是生命周期的处理，也就是 Native 生命周期、Flutter 引擎的绑定和解绑、及 Flutter 自己的生命周期这三部分之间的关系和处理非常关键。

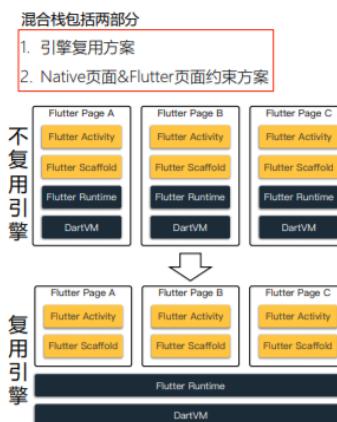
最简单的例子，当页面从打开到关闭，是一个线性的流程，故可以与 Flutter 的生命周期进行绑定。第二点，从 A 页面打开半透明 B 页面时，Android 不会调用 A 的 `onStop` 方法，如果在 `onStop` 方法上做一些特殊处理，则不会被调用，出现问题。第三点，如果从 B 页面返回 A 页面时，会先调 B 的 `onPause` 方法、再调 A 的 `onStart`、`onResume`、再调 B 的 `onStop`、`onDetach`，此时如果在 `onResume` 中做一些正向操作，同时在 `onDetach` 做负向操作，就会出现问题。最后 `ViewPager` 和 `Transaction` 的 Fragment 生命周期管理问题非常令人头疼，因为 `ViewPager` 和

Transaction 对生命周期的处理都不同，而且会调用自己独特的方法，对生命周期的处理进一步增加难度。

综上，Android 的生命周期十分复杂，牵一发动全身，同时 Flutter 引擎处理也十分复杂，如果绑定和解绑时序错乱会带来非常严重的后果。

### Flutter- 混合栈

引擎复用 – 核心问题



复用引擎的核心问题：生命周期的处理

1. 正常生命周期处理 ( A打开关闭 )

`onAttach` `onStart` `onResume` `onPause` `onStop` `onDetach`  
`getEngine` `attachSurface` `attachEngine` `detachEngine` `detachSurface`

2. 半透明界面生命周期处理 ( A打开半透明B )

`onStart` `onResume` `onPause` → `onAttach` `onStart` `onResume`

3. 返回时生命周期处理 ( B返回A )

`onPause` `onStart` `onResume` `onStop` `onDetach`

4. ViewPager/Transaction的Fragment生命周期处理

( 入场 ) `onAttach` ( 切入 ) `onStart` `onResume` `onHiddenChanged` `setUserVisibleHint`  
( 切出 ) `onPause` `onStop` ( 退出 ) `onDetach`

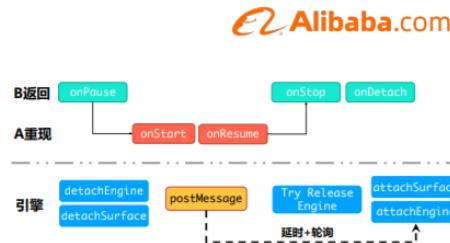
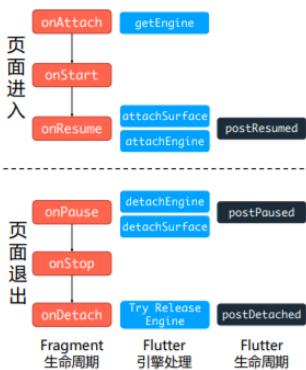
Android生命周期十分复杂，牵一发动全身  
Flutter引擎处理十分复杂，时序错乱会带来严重后果

### 引擎复用 – 技术方案

下图梳理了将 Flutter 引擎与 Android 生命周期间的绑定和解绑的原理，如引擎绑定和解绑分别放在 onResume 和 onPause 上，从而避免半透明界面问题，同时又将 Flutter 生命周期也放在其中。针对返回页面问题，加上了延时和轮询队列保证 onResume 中的方法晚于上一个页面销毁的方法，突破生命周期的限制。

## Flutter- 混合栈

引擎复用 – 技术方案



1. 保证Flutter生命周期的完整性
2. onResume + onPause 解决半透明界面的问题
3. 通过延时+轮询突破生命周期的限制

下图通过代码对比了三种不同的方案，第一个是官方的 Flutter SDK，第二个是闲鱼的 Flutter Boost，最后是 ICBU 混合栈。可以发现官方的 Flutter SDK 直接调用了 onStart 方法，并没有解决半透明页面问题。同时在 B 页面返回 A 页面时，使用了 Handler().post() 的消息队列，造成一定程度上的时延。Flutter Boost 对生命周期做了很多梳理，但是也做了一些简化，如 onPause 时没有调用 Flutter 的 postPaused 方法，这样可以极大的简化生命周期的复杂度，梳理出所需要的生命周期。而 ICBU 的混合栈则是兼顾了 Native 和 Flutter 的生命周期。

## Flutter- 混合栈

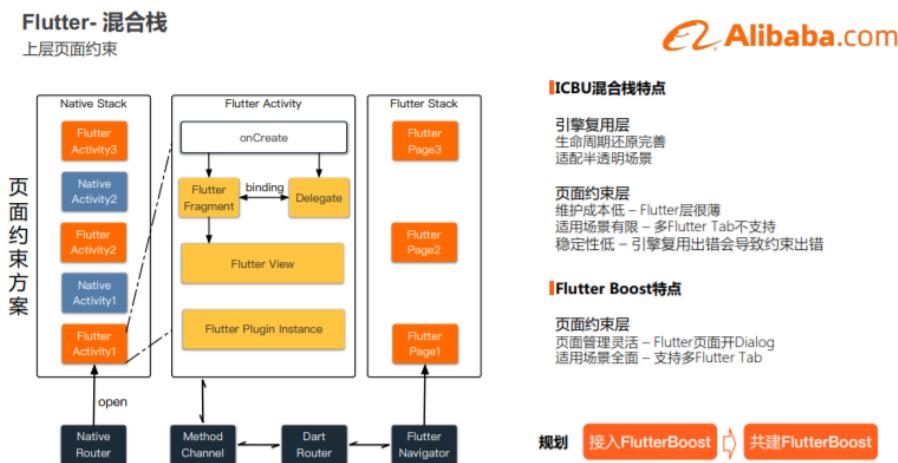
引擎复用 – 方案对比



<pre> void onStart() {     Log.v(TAG, "message: \"onStart()\"");     ensureAlive();      // We post() the code that attaches the host to the flutter engine     // some kind of blocking logic on the host causes launching Activities to wait     // behavior we are able to move this back to the flutter engine     // TODO(metcarron): figure out how to do this     new Handler().post(new Runnable() {         @Override         public void run() {             Log.v(TAG, "message: \"Attaching Flutter View\"");             flutterView.attachToFlutterEngine();         }     }); } </pre>	<pre> public void onResume() {     mSyncer.onAppear();      Log.v(TAG, "onResume()");     ensureAlive();     flutterEngine.getLifecycleChannel().appIsResumed(); }  flutterEngine.getActivityControlSurface().attachToHost(     host.getActivity(),     host.getLifecycle() ); </pre>	<pre> void onResume() {     Log.v(TAG, "message: \"onResume()\"");     ensureAlive();     // 为了确保onResume一定晚于上个页面onDestroy的方法() }  void 为了确保onResume一定晚于上个页面onDestroy的方法() {     new Handler().postDelayed(runnable, delayMin: 100); }  Runnable runnable = () -&gt; {     if (LifeAllFlutterFragment.isFinishing) {         // 为了确保onResume一定晚于上个页面onDestroy的方法()         return;     }      if (flutterEngine == null &amp;&amp; flutterView == null) {         boolean isVisible = false;         if (host instanceof LifeAllFlutterFragment) {             isVisible = ((Fragment) host).getUserVisibleHint();         }         if (isVisible) {             flutterEngine.getLifecycleChannel().appIsResumed();             flutterView.attachToFlutterEngine(flutterEngine);             doInitialFlutterViewRun();             host.configureFlutterEngine(flutterEngine);         }     } } </pre>
<b>Flutter SDK</b> 不支持半透明界面	<b>Flutter Boost</b> 生命周期不完整	<b>ICBU 混合栈</b> 兼顾Native和Flutter 生命周期

## 上层页面约束

上层页面约束最核心的是要保证一个 Native 页面要一比一的对应 Flutter 页面。ICBU 的页面约束是利用了 Native 的页面栈及 Flutter 的页面栈，通过两个栈一起出一起进的行为保证一比一的页面约束。但其实页面约束层还是存在几点问题，首先是稳定性低，引擎复用一旦出错会导致约束出错，其实是适用场景有限，不支持多 Flutter Tab。与之对应，Flutter Boost 的页面约束层非常优秀，它通过对每一个 Flutter 容器单开一个 Navigator，来保证页面约束是一比一的。因此 ICBU 计划接入 Flutter Boost，共建 Flutter Boost。



## 阶段性总结

ICBU 混合工程实现了一键切换源码和产物依赖，实现了一键本地或 MTL 打包，搭建了 AliFlutter 初版工程框架。在混合栈方面，主要是在内存和页面启动时长方面的提升，复用引擎相比于不复用引擎在内存上有所减少，下图中只是对比了两个页面，随着打开页面的增加，内存差距也会越来越大。同时复用引擎节省了大概 300 毫秒左右的启动时长。在经历了半年的初步建设之后，Flutter 在提效，UI 丰富力及

极致体验方面都表现优秀，当然在视频播放、动态化能力、架构优化等方面还有待优化，进一步提升研发效率。

**Flutter**

阶段结果&新的问题

**混合工程**

- 一键切换源码或产物依赖
- 一键本地或MTL打包
- 搭建AliFlutter初版工程框架

**混合栈**

		基础内存	订单列表	订单详情
内存	复用	300.2M	394.6M	440.5M
	不复用	300.1M	409.8M	460.0M

	版本	准备时间	请求时间	总耗时
页面启动时长	7.4.1	747ms	810ms	1557ms
	7.5.1	454ms	437ms	891ms

对比新旧版本，页面启动时长减少了42%



**初步总结**


  
提效  
基建50人日，提效100人日
 


  
UI富有表现力  
扩展能力完备，组件库丰富
 


  
极致体验  
运行效率、UI还原

**待优化项**

1. 视频播放
2. 动态化能力
3. 架构优化

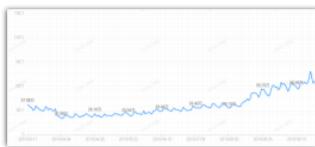
## 三、Flutter 能力补齐

### 视频播放解决方案

一般的视频方案是直接提供 View，然后直接播放。但在 Flutter 中很难复用 View，即便可以，代价也非常昂贵。ICBU 借鉴 Flutter 官方和闲鱼的代码，整个 Flutter 视频渲染流程如下图右侧，我们通过纹理传递视频实现跨技术栈的渲染，Native 视频播放器将自己的视频流通过纹理渲染到 Flutter 的控件上。对视频的播放暂停等控制行为通过 Flutter 自带的消息通道进行传递。如此便可以封装成一个完整的 Flutter 的播放组件，同时底层可以复用 App 中已有的视频播放。

## Flutter – 视频播放

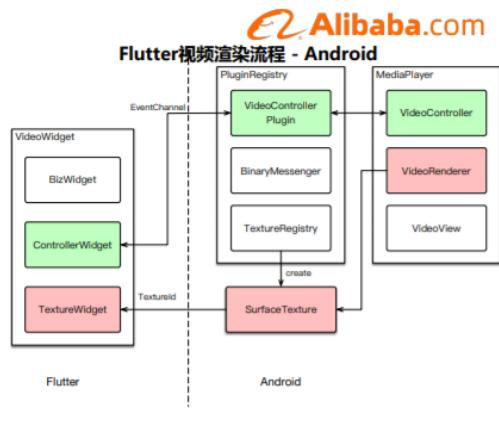
背景 & 方案



视频开始在ICBU落地  
视频的播放量飞速上升

如何借助Flutter为视频业务提效？

**难点** 复用View十分昂贵 ( PlatformView )  
如何借用Native播放器能力



**解法**

放弃View级别的复用  
通过纹理传递视频流实现跨技术栈渲染  
纹理传递 + 修改手淘播放器SDK

视频技术方案如下图，底层是播放器，通过纹理的传递向 Flutter 提供视频播放的能力，播放控制和优化策略是通过统一视频开放接口向上支持。Flutter 自己可以在上层可以封装各种组件，进行业务提效。Flutter 视频页启动时长相较于 Weex 降低了 400ms，对应视频方案已输出给考拉，健康等用户。

## Flutter – 视频播放

结果

### 视频技术方案



**优点**

1. 无需修改FlutterSDK和引擎
2. 下层实现对上层透明
3. 封装视频组件赋能业务

**Alibaba.com**

### 页面示例

仅限受邀买家 11/13 商家动态 (13)

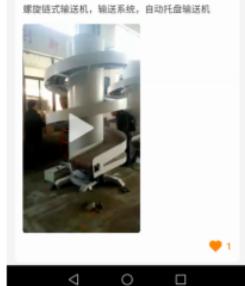
我的收藏夹

产品(603) 供应商(54) 商家动态(13)

**结果**

视频页启动时长  
对比Weex降低400ms

方案输出给考拉&健康



## Flutter 动态化

大家都知道 Flutter 的动态化是它的薄弱点，下图展示了业界所有的动态化技术方案在各个方面的比对情况。首先一类是编译产物下发，如 Hot patch；第二类是 JS 类框架，如 RN, Weex, H5；第三类是以淘系为代表的动态组件框架，如 DinamicX。经过反复的考量，编译产物下发的政策风险较高，Google Play 和 AppStore 都不允许此方案的上线，而 JS 类框架性能损失很大，而以 DinamicX 为代码的动态组件方案在各个方面都取得了优秀的平衡。DinamicX 是组件级接入，使用非常灵活，而且它使用了 Android Layout Xml，学习成本更低。其配合 Flutter 可以达到  $1+1>2$  的效果，借助 Flutter 的一致性，不需要再维护 Android 和 iOS 两套 SDK，维护成本有所减少，针对已使用 DinamicX 的业务，其迁移成本更低。

### Flutter - 动态化

技术选型



技术方案	技术示例	下发内容	两端一致	性能损失	维护成本	动态范围	政策风险
编译产物下发	Hot Patch	小	否	无	中	高（全代码）	<b>非常高</b>
JS类框架	RN、Weex	中	是	<b>很大</b>	高	高（全代码）	中
动态组件	DinamicX	小	是	较小	中	中（组件）	无

以DinamicX为代表的动态组件方案，在各个方面取得了优秀的平衡。

#### DinamicX的优点

- 组件级接入，使用灵活
- 使用Android Layout Xml，学习成本低



#### **1+1 > 2**

- 完美的一致性
- 维护成本减少
- 迁移成本低

#### 电商业务的特点

- 需要运营场馆多 -> 重视UI动态性

## 四、结果 & 规划

ICBU 的 Flutter 建设工作在前期参与了同闲鱼的技术共建，实现了多种控件，具备了国际化 RTL 能力，适配了高版本的 SDK。在未来，我们期望所有 Native 页面将被 Flutter 页面取代，针对动态化场景使用 DinamixX，若还是有欠缺，则使用 H5, Weex, 小程序等进行补全。

## Flutter - 动态化

节奏



参与同闲鱼的技术共建

- 实现多种控件 (Achieve various controls)
- 国际化 : RTL (Internationalization: RTL)
- 高版本SDK适配 (High version SDK compatibility)



未来场景

## 架构图

通过左侧工程体系的建设，保证研发效率，同时针对不同的中间件进行建设，以支持不同的业务场景，在未来，除了 Android 和 iOS，我们还希望 Flutter 可以带来更多平台上的优势。

## 结果

架构图



## 结果

从结果层面，我们沉淀了丰富的技术，包括混合工程体系，中间件等。同时通过与阿里巴巴集团的紧密合作，向 AliFlutter 提供了工程体系及多语言能力，同闲鱼一起共建了 Dx-Flutter。使用 Flutter 之后，开发效率相比于使用 Native 提升了 60% 以上。通过 Flutter，使得资源弹性增大，避免出现 Android 和 iOS 资源不平衡导致的业务不一致的问题。最后，通过 Flutter 节省了大量人日，让更多的人力投入到端智能和数据化及更多其他业务中。技术影响力也在逐渐扩大，目前很多的技术方案已提供阿里集团内部很多 BU。



## 未来规划

ICBU 建设 Flutter 的核心目标是保障 Flutter 的性能和质量，进一步提升提效效果，进一步扩大提效范围。拆分开来分别对应性能 & 质量，接入及拓盘。在早期，Flutter 本身不够成熟，各个问题的最优解都不是很清楚。在现在，Flutter 逐渐成熟，大家对各部分的最优解都达成了一致，阿里希望有个统一的 Flutter 中台，承担大部分的基建工作。因此，ICBU 也计划进一步接入 AliFlutter，减少维护成本，同时对 AliFlutter 形成反馈，增加 Flutter 基建的价值。

## 规划

FY2021



**核心目标：**保障Flutter的性能和质量，进一步提升提效效果、进一步扩大提效范围。

### 性能&质量

页面启动时长	质量标准的建立
内存&帧率	长效机制的建立

### 接入AliFlutter

工程体系	打包体系	混合栈
------	------	-----

### 拓盘

卖家无线	接入
PC	Flutter for Desktop接入
H5	Flutter for Web接入



### 核心价值

用户体验

### 核心价值

极大减少维护成本

增加基建的价值

### 核心价值

卖家无线 跨端提效/卖家共用页面

PC 新技术栈的提效

H5 和App的共用页面

由阿里云开发者社区志愿者董黎明整理。

# Flutter 在饿了么的应用与沉淀

**摘要:** Flutter 作为当前最火的跨平台研发方案，它到底好在哪里？饿了么从 2018 年下半年开始接触 Flutter，并在多个 App 大量落地 Flutter 业务。饿了么对 Flutter 的期待是保质提效，赋能业务。阿里巴巴新零售淘系技术 AliFlutter 系列第八场直播中邀请了蜂鸟大前端资深 iOS 工程师李永光为大家介绍“饿了么为了”保质提效，赋能业务”，选择 Flutter 作为跨平台研发方案的缘由，Flutter 在饿了么应用与落地情况，饿了么在 Flutter 应用过程中的基础建设和沉淀。相信能给大家带来更多尝试使用 Flutter、以及把 Flutter 实际用于业务开发的信心和决心。

## 演讲嘉宾简介：

李永光，花名雍光，蜂鸟大前端资深 iOS 工程师。4 年深耕移动端，饿了么最早的一批 Flutter 玩家，重点参与了 Flutter 在蜂鸟团队的业务开发落地、工程架构演进。

以下内容根据演讲视频以及 PPT 整理而成。

观看回放 <http://mudu.tv/watch/5817421>

本次分享主要围绕以下四个方面：

一、背景介绍

二、Flutter 在饿了么的应用

三、基础建设与沉淀

四、展望与规划

## 一、为什么选择使用 Flutter

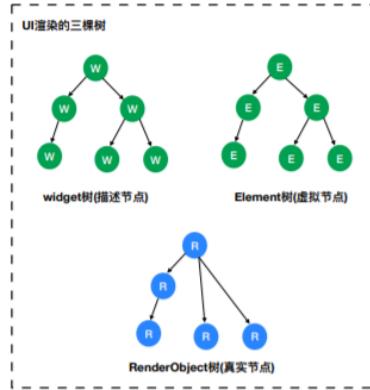
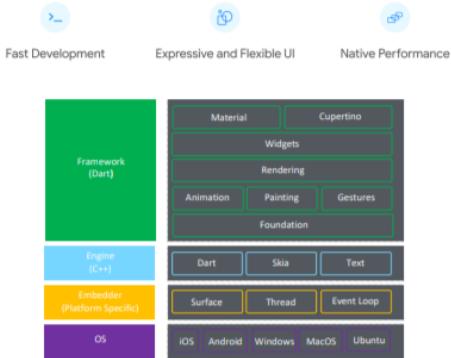
### Flutter 的三大特点及框架

打开 Flutter 官网，映入眼帘的是 Flutter 的三大特点。首先是快速开发，毫秒级的绘制，亚秒级的热加载，丰富的 Widget，可以快速开发出用户界面。二是富有表现力的 UI，丰富的动效接口，流畅的滑动，支持 iOS 和 Android 两种风格的 Widget，可以组合出非常漂亮且灵活的界面。三是可媲美 Native 的运行性能，AOT 模式下，Flutter 代码可以被编译成 ARM 机器码。下图是 Flutter 架构图，最底层是 Flutter 应用的各个平台，包括 iOS、Android、Windows、MacOS、Ubuntu 等。各个平台分别实现 Embedder 平台相关，如提供绘制的画布，在 Android 和 iOS 上就是 OpenGL Context，还包含了线程设置和事件循环。再上一层是 Flutter Engine 层，使用了 C++，包含 DartVM，Skia 2D 渲染等通用能力，还包含文字绘制等关键能力。最上一层是 Flutter Framework 层，使用了 Dart，包含了 Material 和 Cupertino 两种风格的 Widget，以及一部分渲染逻辑，当然还包含框架与引擎之间的通信接口。Flutter 的渲染基本与平台无关，平台只是提供了画布，这为 Flutter 的 UI 一致性和运行一致性提供了坚实的基础。

### Flutter 原理

下图右侧是 Flutter UI 渲染的三棵树，分别包含描述节点，虚拟节点和真实节点。Widget 树在运行期会生成 Element 树和 RenderObject 树，Widget 树在更新时会触发 Element 树的比对和更新，以触发 RenderObject 树的最小更新。Flutter 与 RN 等方案最大的不同就在于 Flutter 的 RenderObject 是自己实现的，而 RN 是使用 Native 控件。

## Flutter 原理



## 客户端研发方案对比

下图给出了五种不同研发方案在四个维度上的对比，包括 Native、Flutter、RN、Weex 和 H5。

1) 在性能方面，H5 最差，Native 最优。Flutter 相比于 RN，在 AOT 编译模式下，由于没有 JS 虚拟机，相对运行性能更高。而且，Flutter 的渲染是直接对接底层的，RN 还需要操作 Native 控件，有着非常高的通信成本。Flutter 的页面加载速度和流畅度都比 RN 更优，在测试中也是符合实际预期的。

2) 在动态性方面，H5 可以实时发布，所以动态性最好。RN 有 JS 虚拟机，可以一定程度上进行代码的动态下方和执行。在 AOT 编译下，Flutter 的动态性可以认为是与 Native 差不多的，都非常弱。如果有此类需求，需要借助其它方式，如 DSL 等。

3) 跨端一致性可以分为 UI 一致性和运行一致性。H5 借助于浏览器规范，其 UI 一致性是非常好的，运行一致性也不错。Native 由于开发栈的不同，其 RenderObject，API 和实现都有很大的不同，因此跨端一致性非常差。Flutter 实现了渲染层，在 Android 和 iOS 上 RenderObject 的实现是一致的。正是因为有了渲染层，平台

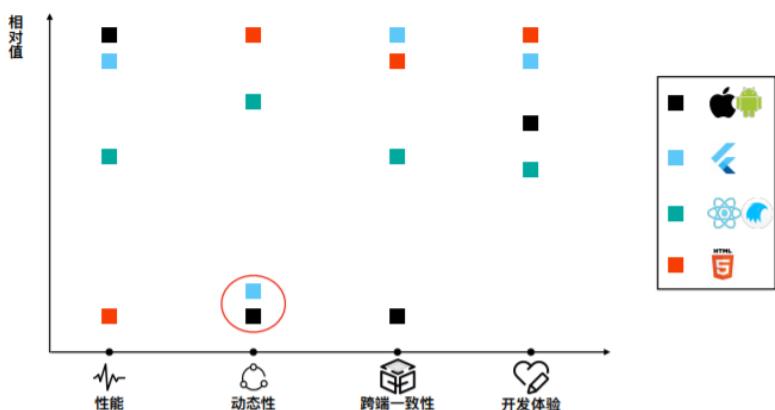
相关性非常低，所以可以像 H5 一样做到一处编写，处处运行。

4) 开发体验方面，H5 开发预览调试都非常方便，Flutter 的各个插件使用起来也比较方便，HotReload 使得开发，调试和运行都可以节省不少时间。

总结起来，Flutter 在动态性上有所诟病以外，性能可以与 Native 媲美，极强的跨端一致性，还可以提供非常好的开发体验。

## 客户端研发方案对比

Alibaba Group 阿里巴巴集团 | TAO TECHNOLOGY 淘系技术部



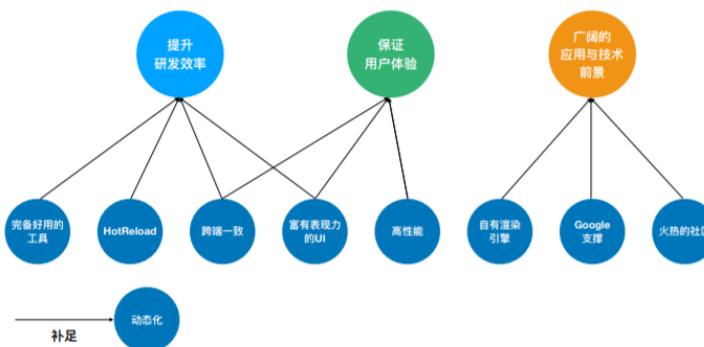
## 饿了么的选择

饿了么对研发方案的期望是用最少的人，搬最快的砖，用户无感知！简而言之是研发效率高，用户体验好。最后，饿了么选择了 Flutter 作为跨平台研发方案。Flutter 具备完备好用的工具，如 Pub，IDE 插件，HotReload，优秀的跨端一致性，富有表现力的 UI，这些特点可以为开发调试和验证工作节省很多时间，从而大大提升研发效率。跨端一致性，丰富的 UI，以及高性能使得 Flutter 开发的页面与 Native 开发的页面体验无差别，而且 Android 与 iOS 可以保持高度的一致性。另外，Flutter 采用的自有渲染引擎在未来可扩展的余地很多，Flutter 是 Google 出品的，Flutter 现在的社区也是非常火热的，阿里巴巴，腾讯，头条等企业的很多 App

都使用了 Flutter，可以说，Flutter 有着非常广阔的应用与技术前景。在动态性方面，目前大家已经有了很多解决方案，如 DSL 等。这点上还是以自己的需求出发，稍作补足即可。

## 我们的选择

Alibaba Group | TAO TECHNOLOGY



## 二、Flutter 在饿了么的应用

从 2018 年下半年到现在，饿了么很多 App 上都已经使用了 Flutter。其中至冠配送大概 80% 的页面都使用了 Flutter。

### 使用Flutter的APP

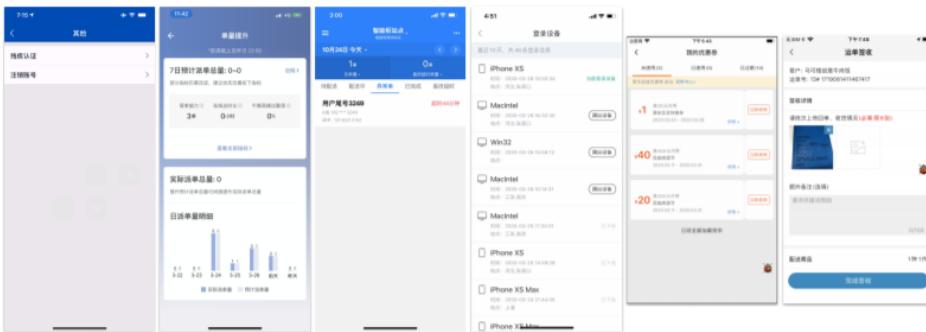
Alibaba Group | TAO TECHNOLOGY



下图展示了饿了么已经上线的 Flutter 页面，涵盖了大部分常见的场景，包括残疾骑士验证页面、单量提升页面、优惠券页面、运单签收页面等。他们的页面加载速度和滑动流畅度都基本与 Native 页面相当。

## Flutter 页面

 Alibaba Group |  淘系技术部  
TAO TECHNOLOGY

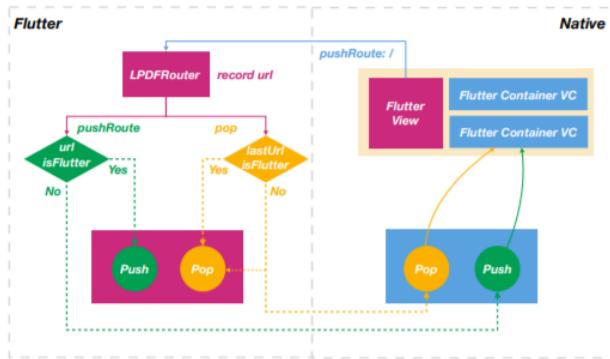


## 混合开发(旧)

Flutter 代码主要以 Module 的形式关联到主工程，自然涉及到混合栈管理问题。Flutter 页面与 Native 页面的切换主要有以下三种情况，Native 切换 Flutter、Flutter 切换 Flutter、Flutter 切换 Native。从 2018 年到 2019 年间，饿了么也实现了自己的混合栈管理方案，Android 和 iOS 的理念相似，这里以 iOS 为例，关键在于 Native 侧公用多个 FlutterVC 的 FlutterView，解决 FlutterVC 不释放问题。饿了么在 Flutter 侧做了一个 LPDFRouter，记录所有与 Flutter 页面有关的 URL。当从 Native 切换 Flutter 时，先打开 Flutter Container VC，Flutter 侧的 Navigator 会 push URL。当连续打开 Flutter 页面时，Native 容器不动，Flutter 的 Navigator 直接 push。当 Flutter 切换 Native 时，Native 容器返回按键绑定 Dart 方法，当上一个页面是 Flutter 时，使用 Native 直接 pop 即可，如果上一个页面是 Native 页面，除了把当前 URL pop 掉，还需要将容器 pop 掉。下图的混合栈管理方案基本

上满足了饿了么的需求，但是方案的一半在 Native，另一半在 Flutter 中，不太适合 Tab 非常复杂的情况。

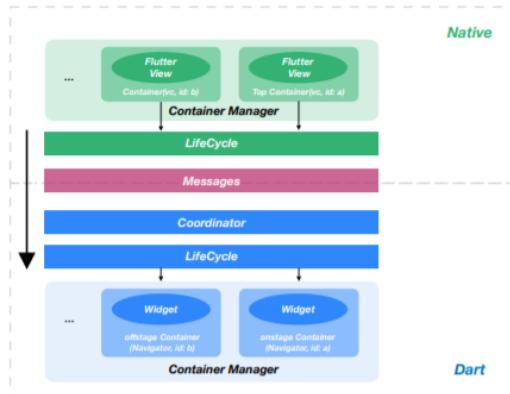
## 混合开发(旧)



## 混合开发(新 – boost)

后来饿了么使用了闲鱼开发的 Flutter boost 作为混合栈管理方案，无论从 Native 还是 Flutter 打开一个页面，都会先打开一个原生的容器，容器的生命周期 ID 会通过 Channel 传到 Flutter 侧，Flutter 容器管理器会打开一个 Flutter 容器。Flutter 容器管理器管理了多个 Navigator，每个 Navigator 只有一个 Flutter 页面，一个 Widget。Flutter 容器与 Native 容器 ID 是一致的。最新的 Flutter boost 使用了多个 Flutter View 的组合，不再复用单个 Flutter View+ 截图。可以发现 Flutter boost 对混合栈的管理是非常纯粹的，都在 Native 侧。在使用 Flutter 时，只需要关注 Native 容器的生命周期即可，适合多 Tab 的 Flutter 页面情况。

## 混合开发(新-boost)

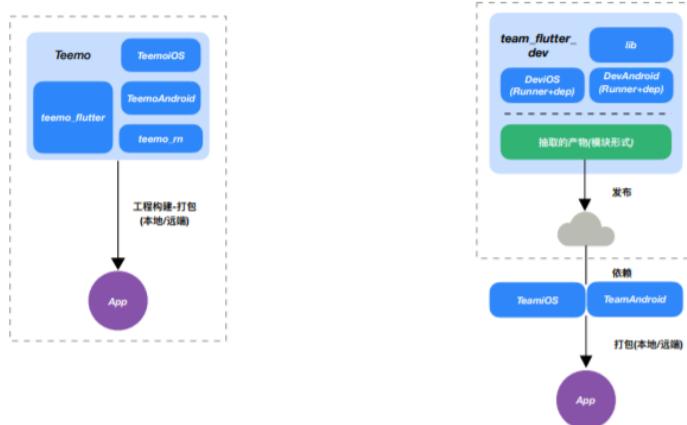


## 研发 / 集成模式

Flutter 工程与 Native 工程是如何组织的，下图中的虚线框代表需要有 Flutter 环境。第一种是至冠模式 Teemo，饿了么最初就希望将至冠配送 100% Flutter 化。因此将 Flutter 工程与 Native 工程放在了同一个仓库里，Native 工程通过 ruby 脚本关联到 Flutter 工程。Native 工程编译时先编译 Flutter 工程，将产物引进来，因此工程的开发、调试、测试、打包等都需要 Flutter 环境。下图右侧是蜂鸟模式，蜂鸟业务非常复杂，饿了么考虑可能只有部分的同学会开发 Flutter 业务，因此将 Flutter 环境与其它业务环境隔离开来。饿了么做了一个 Runner 工程，同步了蜂鸟主工程的很多依赖，如登录、用户管理、安全等。在 Runner 工程中把 Flutter 业务开发完之后，通过本地和远端把 Flutter 产物抽取出来进行发布，如 App Framework、Flutter Framework、以及对应的原生代码。发布之后，蜂鸟工程会版本 tag 的形式依赖 Flutter 的原生产物，那么只开发 Native 业务的同学不需要 Flutter 环境。不同的团队有不要的研发和集成模式，并不存在最佳实践，适合自己的才是最关键的。

## 研发/集成模式

Alibaba Group | 淘系技术部 TAO TECHNOLOGY



## 质量 & 效率结果

从 2018 年到现在，饿了么已经上线了很多 Flutter 页面，页面占比在不同团队都是不同的。Crash 方面，稳定下来后在 0.01% 级别。在流畅度方面，使用 Flutter 工具或使用线上回调函数计算，都可以达到 50 帧以上。而最大的惊喜是在提高研发效率上，在过去一年多的时间里节省了 100 多开发人日。前期不是很熟练，1 个人可以顶 1.5 个人，后期便熟练后，1 个人可以顶 1.8 个人。

## 三、基础建设与沉淀

### 控件库

随着 Flutter 页面的开发越来越多，饿了么联合 UED 做了基础控件的规范和封装，包括按钮、TextFeild 等。当基础的控件创建完后，后续的 Flutter 页面可以复用控件，进一步提高开发效率，同时使得不同页面间的一致性更好。

## 控件库



## 基础插件

Flutter 是一个 UI 的开发框架，因此也不能免于使用 Native 能力，如定位、持久化。饿了么在开发过程中也积累了很多基础的插件，如 Crash 上报、推送处理、社交分享，还桥接了 Native 网络库发出 Flutter 请求，使得性能与体验都保持一致。

## 基础插件



Crash上报



网络请求



推送处理



定位



持久化



声音播放



社交分享

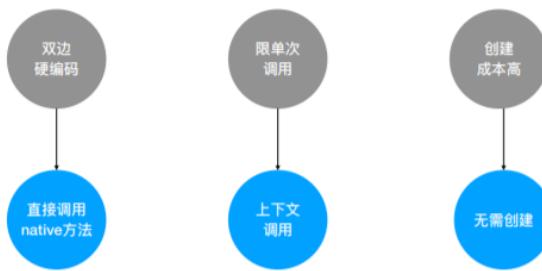


日志记录

## dna – 背景与目标

随着 Channel 越来越多，小到获取 Native 参数，大到获取 Native 执行功能，使得 Channel 使用越来越不便，其痛点主要有以下三点，一是双边硬编码。在 Dart 和 Native 两边针对 Channel 的名字，如方法名等参数做硬编码，一旦出错就会调用错误。二是只能单次调用，Flutter 调 Native 时只能通过方法名匹配到一个代码块。三是创建成本高，不单要编写 Channel 代码，有时还需要创建 Plugin。为了解决以上问题，饿了么也做了一些探索和实践，即 dna Plugin。在设计和实现 dna Plugin 时也有几个对应的目标，首先是直接调用 Native 方法，不再使用 Channel 名或方法名等，同时 Native 侧不再使用硬编码。二是支持上下文调用，Dart 可以直接调用 Native 代码。三是无需创建 Channel 和 Plugin。

## dna — 背景与目标



©雍光 菜叽 执御 泽卦

## dna – 使用

下图展示了 dna 快捷方法的使用，获取了 Android, iOS 对应的版本号，以及系统平台的字符串。可以发现，使用了 NativeObject 作为 Native 变量，Native-Object 的生命周期与 Native 的生命周期是一致的。上一个方法的返回值可以作为下一个方法的参数，即上下文调用。原生调用上与 Native 很像，支持链式语法。返

回值默认是最后一次 context 的的返回值。下图下层框里是假想的代码， dna 执行 Native 方法时相当于执行了框中的方法。



```

dna — 使用

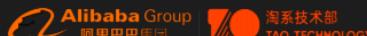
String platformVersion = await Dna.traversingNative(ObjCContext context) {
    NativeObject version = context.classFromString('UDDevice').invoke(method: 'currentDevice').invoke(method: 'systemVersion');
    NativeObject platform = context.classFromString("NSString").invoke(method: 'stringWithString:', args: ['iOS-']);
    NativeObject platformVersionText = platform.invoke(method: 'stringByAppendingString:', args: [version]);
    context.returnVar = platformVersionText; // 该句可省略
}, (JavaContext context) {
    NativeObject version = context.classFromString('me.ele.dna_example.DnaVersion').invoke(method: 'getVersion');
    NativeObject platform = context.newJavaObjectFromConstructor('java.lang.String', ["Android-"]);
    NativeObject platformVersionText = platform.invoke(method: "concat", args: [version]);
    context.returnVar = platformVersionText; // 该句可省略
});

id version = [UDevice currentDevice] systemVersion];
id platform = [NSString stringWithFormat:@"iOS-%"];
id platformVersionText = [platform stringByAppendingString:version];
return platformVersionText;

String version = DnaVersion.getVersion();
String platform = new String("Android-");
String platformVersionText = platform.concat(version);
return platformVersionText;

```

实际使用的 dna 代码相对更简单，下图展示的是拆箱解码的操作。iOS 侧调用了一个类来获取实例，Android 是直接调用了实例。iOS 包含一个类一个方法，Android 还需要一个 dna method 注解。



```

dna — 使用

Dna.traversingNative(ObjCContext context) {
    context.classFromString("LPDEquipmentModuleService").invoke(method: 'sharedInstance').invoke(method: "rebindStarbucksMonitor");
}, (JavaContext context) {
    context.classFromString("me.ele.sensor.SensorDetectorManager").invoke(method: 'unbind');

}

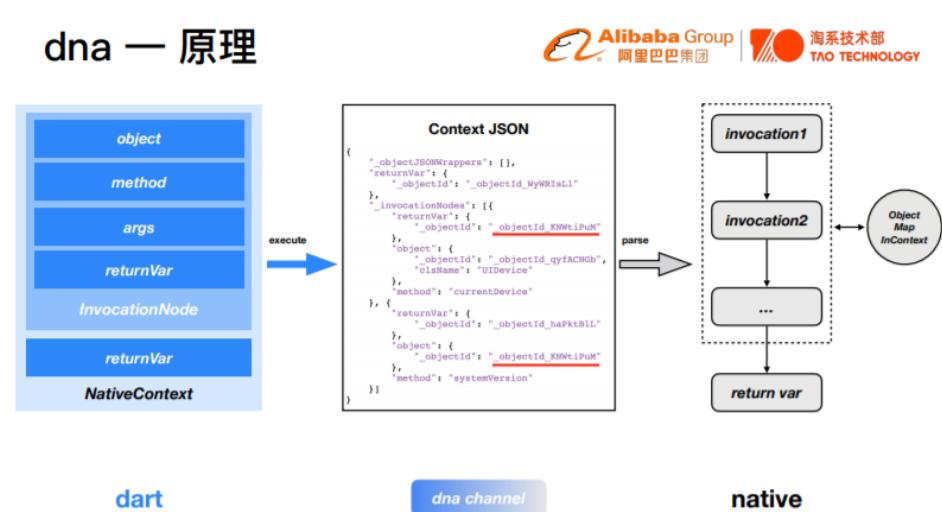
@implementation LPDEquipmentModuleService
+ (instancetype)sharedInstance {
    //...
}
- (void)rebindStarbucksMonitor {
    //...
}
@end

public class SensorDetectorManager {
    @DnaMethod
    public static void unbind() {
        //...
    }
}

```

## dna - 原理

当上面的 context 中的 Native Object 不断的 Invoke 方法时，会形成一个个的 InvocationNode，InvocationNode 定义了方法名，参数以及返回值。当 context 调用时，会转换成一个 context JSON，传给 Native。Native 解析 JSON，转换成对应的一个个 Invocation，被陆续调用。每个 Native Object 都对应一个 ID。一个 Invocation 里不但有调用值的 ID，还调用了返回值的 ID，他们都会放在一个 map 里，实现上下文关联。



dna 在调用到一个 Java 方法时，需要一个 dna method 注解，主要是因为 Android 的 release 中做了代码的混淆，无法通过类名和方法名定位。当一个方法加了 Method 注解时，可以扫描这个注解，生成代理类和代理方法。APT 生成代理文件中包含扫描注解、生成的代理方法、存储的方法名和参数信息。dna 调用时先通过运行时注解匹配到代理方法，然后通过 owner 调用到真正的对象方法。



## Channel VS. dna

dna 不需要新建 Plugin 和 Channel，而且由于 dna 是直接调用 Native，所以所有硬编码都没有了，dna 和 Channel 都不支持 C 函数，也不支持 Native 对象内存管理。目前 dna 传递 JSON 时还是使用 Channel 传递。很多情况下，只需要调用小部分 Native 代码，dna 也无需创建 Plugin 及 Channel，而且 Native 也不需要硬编码。在蜂鸟团队中，非常基础的功能是使用 Channel 做的，稍微涉及业务的功能都使用 dna。

## Channel vs dna



对比项 工具	新建Plugin及channel	dart调用包含硬编码	native处理包含硬编码	native处理统一	调用已有类方法	dart上下文调用native	调用C函数	native对象内存管理
Channel	Y	Y	Y	N	N	N	N	N
dna	N	Y	N	Y	Y	Y	N	N

最小收益：1.无需创建Plugin及channel  
2. native 无需额外处理

channel 调用 native的一个代码块

dart			native		
dart channel	method	args	native channel	method	args

dna调用一个native类的一个方法  
约定顺序参数列表

dart			native		
class name	method name	args	class	method	

## 其他实践

在饿了么自身的实践中，也做了其他的探索，如热修复，虽然还没有上线，但为 Flutter boost 提供了一些支持。

## 其他



@eleme WP

## 参与共建 – 背景

AliFlutter 的目标是共同打造基础设施，制定标准，复用技术。通过培育集团各

个 BU Flutter 生态，沉淀业务与技术。顺便可以联合对外产生凝聚的影响力。

## 参与共建 — 背景



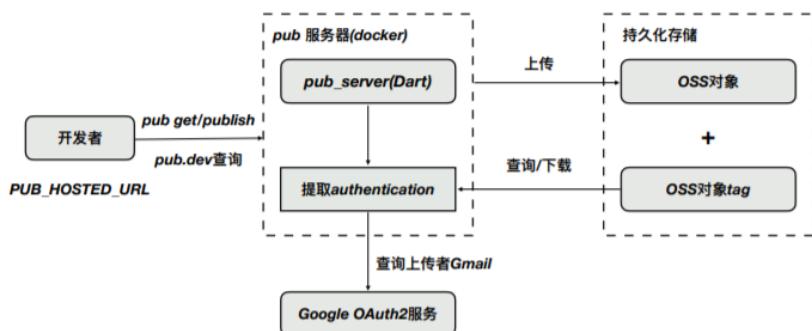
### AliFlutter

拉通	共同打造基础设施/制定标准/复用技术
生态	培育各BU Flutter生态/沉淀业务与技术
影响力	联合对外产生凝聚的影响力

## 共建 – Pub Server

在 Pub Server 中，通过 get 和 publish 可以下载一些库。AliFlutter 为了 Pub 库的保密性，以及外部库的下载速度，也做了自己的 Pub Server。与官方方案最大的不同是将 Google Cloud 存储换成了阿里云 OSS 存储，当查询外部的库时先看内部是否有缓存，没有才去下载外部的库。

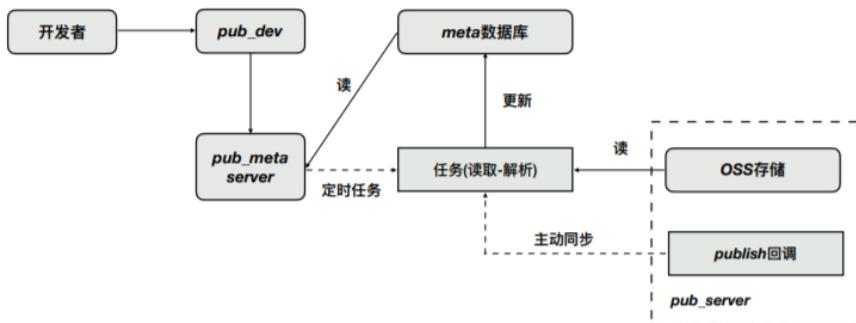
## 共建 – Pub Server



## 共建 – Pub Dev (前端检索)

Pub Dev 是大家所使用的前端检索页面，可以查到已发布的 Plugin 或 Dart 库。既然有了内部的 Pub Server，还需要做 Pub Dev。Pub Server 提供的检索功能非常有限。饿了么做了自己的元信息数据库，有一个定时任务可以定时读取和解析 Pub Server 的产物信息。如此 Pub Dev 才可以满足各类检索需求。

## 共建 – Pub Dev(前端检索)

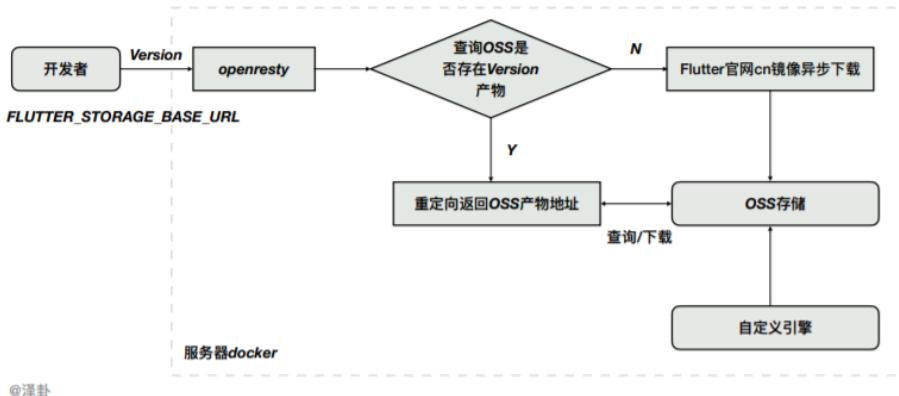


©硬明 ©泽卦

## 共建 – 产物服务器

AliFlutter 也做了自己的产物服务器，最初主要也是为了提高产物查询速度。与 Pub Server 类似，依然是先检查是否有内部的缓存，无需再去查询外部产物，从而后面同学查询的速度也更快了。

## 共建 – 产物服务器

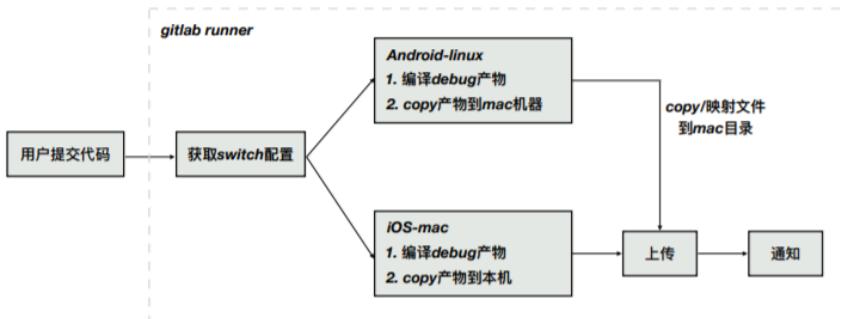


@泽卦

## 共建 – 引擎工作流

同时，产物服务器也可以作为引擎工作流。AliFlutter 也对引擎做了一些修改，如图片优化和机器优化，再将 Android 和 iOS 产物，如编译 debug 产物，上传到产物服务器中。整个引擎工作流是通过 CI 实现的。

## 共建 – 自定义引擎工作流

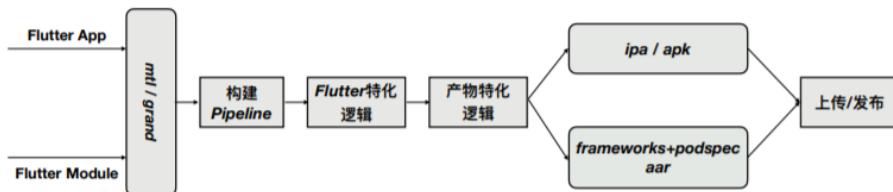


@泽卦 @昭宗

## 共建 – CI/CD

AliFlutter CI/CD 的主要目标是打出包含 Flutter 代码的 App，以及 Flutter Module 下的产物。在标准的 App 构建流程 pipeline 之上，饿了么加了很多 Flutter 编译逻辑和产物的特化逻辑。先把 App Framework 和 Flutter Framework 都打出来，还需要把 Flutter Plugin 的代码单独打成 Framework 或 aar，生成 postsec，最后上传并发布。

## 共建 – CI/CD



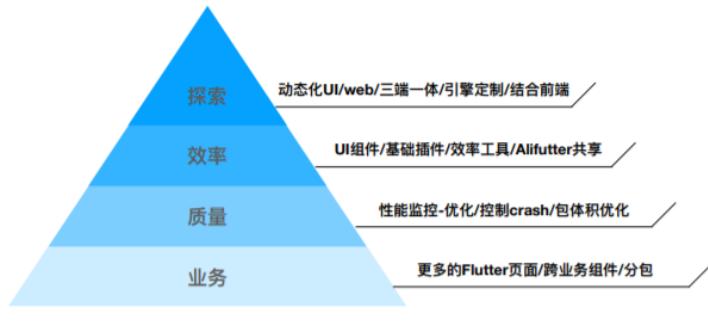
@eleme WP

## 四、展望与规划

Flutter 给饿了么带来的最大的价值是业务落地的提速。在未来，饿了么也会推进更多的业务使用 Flutter 进行开发，包括跨业务组件的开发，以及 Flutter 代码的分包。围绕业务，还需保证质量，目前对 Flutter 性能监控和优化都是不够的，所以计划在性能监控和优化上再进一步扩展。另外，在业务的狂奔过程中，控制 Crash 数量，做好包体积优化。此外，饿了么也希望做进一步提高研发效率的工作，如扩大 UI 组件范围，抽取基础插件，通过与 AliFlutter 共建，做好基础设施及效率工具，如扫码开发等。最后，有剩余时间的话还可以做一些探索性的工作，如动态化 UI，

Flutter Web，三端一体化开发，引擎定制，同时可以结合前端做一些思考和遐想。总结起来，饿了么对 Flutter 的期待是保质提效，赋能业务。

## 规划与展望



## 保质提效 赋能业务

由阿里云开发者社区志愿者董黎明整理。



淘系技术

淘系技术

阿里巴巴淘系技术

淘系技术

扫一扫二维码，关注我吧



阿里云开发者“藏经阁”  
海量免费电子书下载