阿里云开发者社区
ALIBABA CLOUD DEVELOPER COMMUNITY | EMAS 移动研发平台

云原生时代下的APP开发

走进阿里云一站式应用研发平台EMAS

主编：泠茗

作为国内移动互联网、云计算领域的行业巨擘，阿里巴巴在大前端、云原生领域有着丰富的实战经验。阿里技术人从2016年开始逐步将阿里集团内部成熟的应用中间件云化输出，并在2018年推出了移动研发平台EMAS，如今，EMAS已经逐渐成长为横跨多端（移动App、H5应用、小程序、Web应用等）场景的云原生应用研发平台，基于广泛的云原生技术（Backend as a Service、Serverless、DevOps、低代码等），为企业、开发者提供一站式的应用研发管理服务，涵盖开发、测试、运维、运营等应用全生命周期。为了让大家更全面地了解EMAS产品背景、产品内容以及相应的应用案例，特此推出该电子书，希望对广大开发者们有所参考和帮助。

欢迎加入 EMAS 开发者钉钉交流群

群号：35248489

阿里云开发者“藏经阁”

海量电子手册免费下载

卷首语

作为国内移动互联网、云计算领域的行业巨擘，阿里巴巴在大前端、云原生领域有着丰富的实战经验。我们从 2016 年开始逐步将阿里集团内部成熟的应用中间件云化输出，并在 2018 年推出了移动研发平台 EMAS（ https://cn.aliyun.com/product/emas ），如今，EMAS 已经逐渐成长为横跨多端（移动 App、H5 应用、小程序、Web 应用等）场景的云原生应用研发平台，基于广泛的云原生技术（Backend as a Service、Serverless、DevOps、低代码等），为企业、开发者提供一站式的应用研发管理服务，涵盖开发、测试、运维、运营等应用全生命周期。为了让大家更全面地了解 EMAS 产品背景、产品内容以及相应的应用案例，特此推出该电子书，希望对广大开发者们有所帮助。

目录

1.EMAS 产品背景

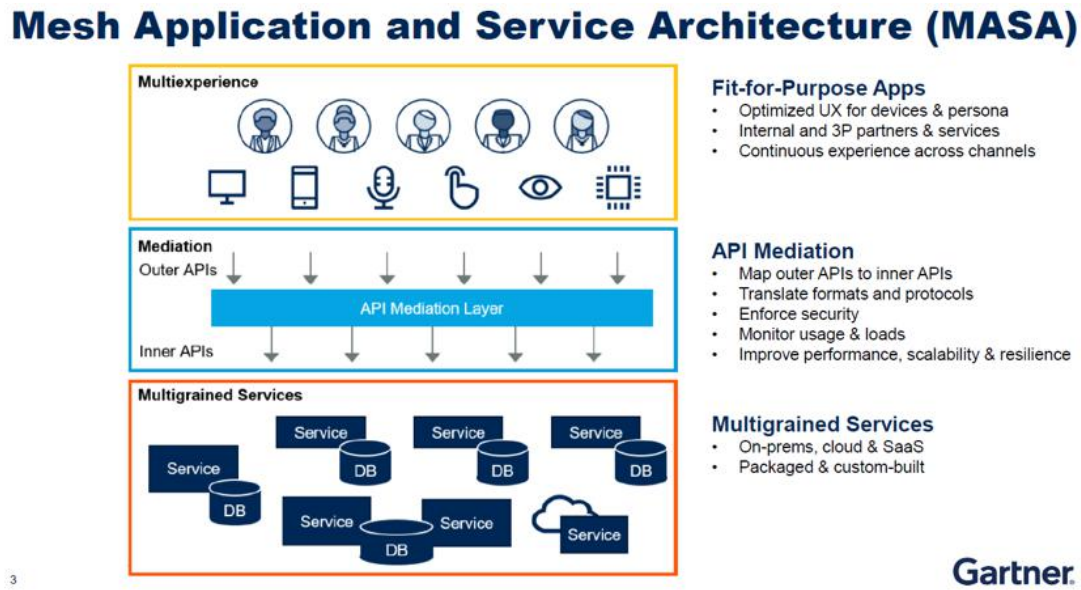
端应用研发进入云原生时代

简介：随着技术的发展和各种用户端场景的涌现，业务前台形式变得更加多样，“面向多样化的端场景提供无缝的、一致的数字用户旅程”已经成为了新时代企业应用架构的关键目标，同时它也是当下大前端技术发展背后的核心业务牵引。基于阿里云在过去几年服务海量用户的经验沉淀，本文总结了新的基于云原生技术的端应用研发范式，期望为广大开发者、企业提供云计算时代面向企业业务前台的应用研发方法论。

作者 | 阿里云 云原生应用研发平台 EMAS 杨锓（泠茗）

多样化用户体验（ Multiexperience ）与大前端

随着云计算、移动化、IoT、AI 等技术概念地落地和持续发展，社会的数字化进程在不断加速。Gartner 近期发布了新的企业应用架构方法论 MASA（Mesh Application and Service Architecture，网格应用和服务架构）[1]，融合近 5 年流行的技术趋势，为广大企业信息化的供应商、开发者和企业用户定义了更广泛的企业数字化应用架构模型。



与阿里所定义的中台不同，MASA 将企业应用拆解为上中下三层，在传统的后端业务能力基础之上，将企业前台，以及用于前后台链接的 API 网关层也涵盖了进来，通过网格化的架构实现企业的业务流程、员工、服务、内容、设备间的动态连接，以构建匹配现有技术形态和未来技术趋势的更敏捷、灵活、可扩展的应用架构。

伴随 MASA，面向企业前台的 Multiexperience（ 多样化用户体验 ）被 Gartner 明确提出并定义为 2020 年的十大技术趋势[2]。Multiexperience 期望利用多元化的前台技术（移动应用、Web、小程序、可穿戴设备等）全面升级企业面向终端客户的数字化触点，以终端客户为中心构建多元（体验多元化）而一体（架构一体化）的用户界面。Multiexperience 与国内所流行的大前端概念不同，但他们背后恰恰有着相通的故事主线。

大前端在国内还没有一个统一的定义，它更偏向一个相对纯粹的技术概念，意指面向客户侧的端技术集合，它的出现始于客户端 Native 与 Web 两大技术栈的不断融合，背后核心是跨平台技术在移动、PC、小程序、Web 等场景下地不断发展和成熟。

大前端技术栈在 Multiexperience 这样的业务需求背景下不断磨砺，同时又反向支撑业务不断扩展其面向终端客户的数字化触点的场景和范围。技术拓展商业边界，商业驱动技术变革。面向全端场景，提供无缝的、一致的数字用户旅程是 Multiexperience 和大前端一脉相通的用户理念。

如今，面向全新的业务架构范式，如何加速新时代下多元化的端应用研发，为业务提供更敏捷而高效地交付呢？云原生技术是最佳选项。

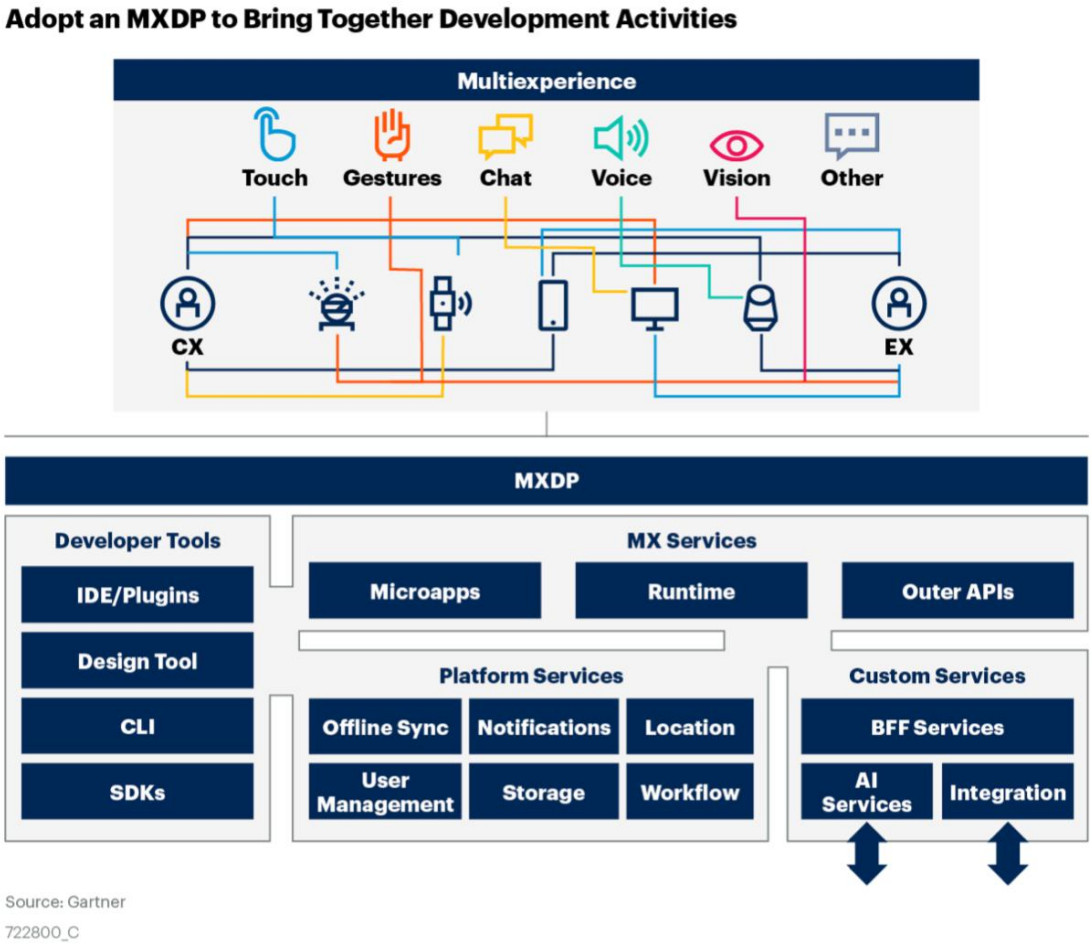
一云多端，云原生技术如何驱动端应用研发

很多人有疑惑，云原生与端测的研发有什么关系，云原生不是一个后端技术域的概念么？其实不然。云原生代表了一种应用构建的方法论：如何最大程度地利用云计算服务模型的优势低成本、敏捷地构建和管理一款弹性的应用。它的关键理念包括：

- 所有的运行环境透明化，弹性伸缩；
- 所有的研发流程流水化，高效交付；
- 所有的基础设施服务化，按量付费；

云原生的研发模型旨在降低业务的技术风险，让开发者可以更单纯地专注于自己的业务。面向端应用场景，云原生技术理念同样适用。

Gartner 在 2019 年定义了一条全新的技术赛道：Multiexperience Development Platform（MXDP）[3]，用以描述那些通过敏捷、现代化的技术能力帮助企业高效地实现 Multiexperience 的研发平台，其能力矩阵示意如下图：



从 MXDP 的关键元素构成我们可以看到，除了传统的研发工具类组件外，云原生技术成为了 MXDP 最核心的技术元素，其中的典型技术包括：

DevOps，驱动端应用的高速迭代

端应用对比传统后端有着典型的差异，以移动 App 为例，应用本身构建在异构的机型、OS 平台之上，运行环境约束较多，依赖大量的后端服务支撑，应用本身的持续集成和交付过程也包含了许多移动场景特有的元素，比如编译环境、兼容测试、内测分发、渠道打包、灰度发布等，这些关键差异决定了端应用必须构建自己特有的 DevOps 体系。

研发域：面向端应用的 DevOps 平台需要解决应用持续集成过程的组织协同和自动化。针对端应用的研发期，成熟的 DevOps 平台需具备：

- 面向多端的编译构建环境并实现自动化的软件更新；
- 可靠，弹性伸缩的构建服务集群；
- 代码与证书托管；
- 静态代码扫描；
- 软件定义的工作流；

测试域：端应用的测试较传统后端应用复杂度大幅提升，除了基础的功能、性能测试外，还需要有面向异构机型、终端、操作系统的兼容性测试，成熟的端应用测试平台应具备：

- 兼容性测试服务，覆盖主流的机型、设备、操作系统；

- 性能测试服务，支持各类应用崩溃、ANR、卡顿、IO、CPU、内存等关键性能指标监测和评估；
- 自动化测试引擎，支持测试用例编写、回放和管理；
- 远程真机能力，支持设备的云端访问；
- 智能 Monkey 与 AITest 等智能化技术驱动的测试能力；

发布域：端应用的目标发布对象是海量的终端设备，生产发布受应用市场审核限制，因此，可靠、精准、定向的应用分发能力是应用生产分发的关键技术要求，包括：

- 企业内分发能力；
- 支持软件定义的灰度能力，支持面向不同的地域、网络、机型、渠道及其他自定义标签进行灰度分发；
- 面向不同渠道的生产发布能力；
- 版本管理与归档；

运维域：面向端应用的运维体系应始终围绕问题的感知，问题的定位，问题的修复展开，在传统的 Metrics，Tracing，Logging 基础之上，我们特别强调面向端的 Hotfix 的能力，这是区别于后端应用形态的特殊之处。成熟的端运维体系应包含：

- 面向端应用的 APM 能力，支持包括启动速度、页面加载、崩溃、网络性能、API 性能等在内的监控报警体系，并能与后端 APM 联动进行基于事务的访问追踪；
- 面向端应用的远程日志能力，支持实时的终端日志记录和管理，加速问题的远程诊断；
- 面向端应用的热修复能力，支持针对不同设备平台的代码、资源热更新；

运营域：端应用作为企业业务入口，是企业面向终端用户的关键界面，必须时刻洞察用户的行为、倾听用户的声音，驱动业务的敏捷迭代。面向 App 的数据分析以及舆情反馈能力是 DevOps 的关键闭环之一。

数据分析应具备：

- 面向全端的用户行为数据采集能力，包括小程序，APP，H5，PC，WEB，IoT 等；
- 易用的终端埋点工具：支持代码埋点，自动埋点，可视化埋点等；

- 开放的数据能力：支持以 API 方式同步数据，支持与云厂商的计算平台无缝对接；

舆情反馈应具备：

- 面向全端的用户反馈通道能力；
- 智能化的应答机器人；
- 舆情数据搜集和分析；

DevOps 为 Multiexperience 的生产实践带来的关键价值包括：

- **更短的业务迭代周期**

  覆盖端应用全生命周期的工作流与自动化能力将带来应用持续交付能力以及跨团队协同效率地大幅提升，进而缩短业务的交付周期。作为企业面向终端用户的入口，更短的业务迭代周期意味着面向市场更快速的反应，这是数字时代商业成功的基石。
- **更合理的人力资源分配**

  云原生的工具链与自动化流水线将帮助企业避免耗费大量工程技术人员来维护本地化的工具和系统，同时大幅削减应用持续交付过程的人工环节。企业能够将宝贵的人力资源专注在自己核心业务的生产和研发上。
- **更稳固的应用交付质量**

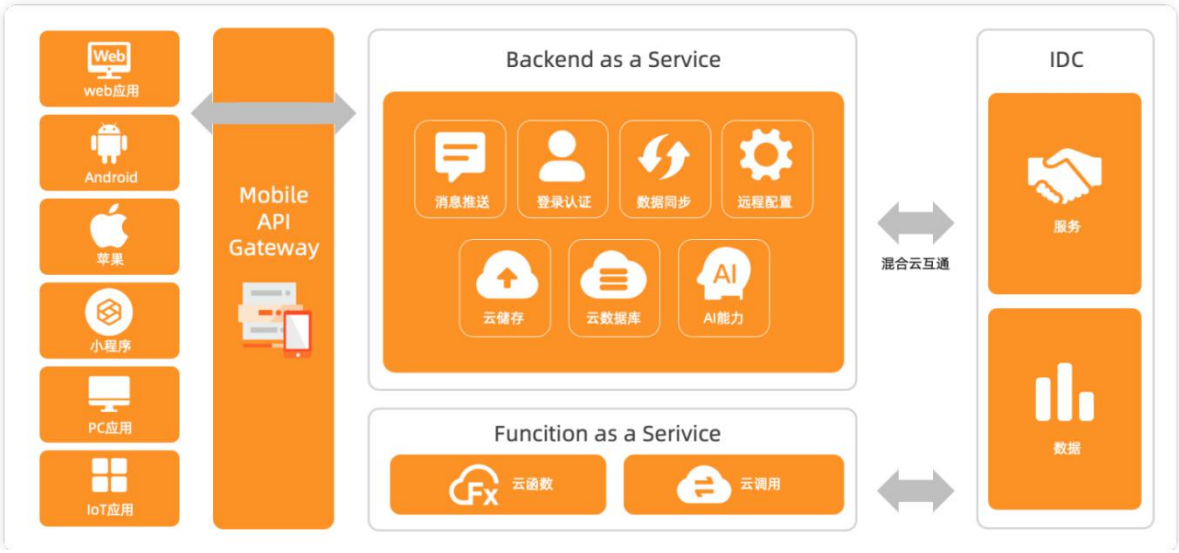
  自动化的终端测试体系以及全方位的监控诊断体系将为端应用提供完整而充分的质量保障，这些云原生服务将为企业节省大量细分领域的专业人员投入，并通过专业且持续的工程技术演进以及智能技术的引进不断优化应用质量保障体系。
- **更优异的即时用户体验**

  围绕端应用全方位的行为数据埋点和分析将帮助企业更好地把握产品功能与市场的匹配度，而即时的舆情反馈能力则帮助企业更好地进行用户管理和关系维护。所有这些直接和间接来自客户的声音将直接驱动业务的快速迭代，通过云原生 DevOps 实现敏捷开发的生产实践。

Serverless & Backend as a Service（BaaS），端应用的运行引擎

Serverless 是当下开发者社区最火爆的话题之一，其核心理念即去服务器化：把底层云计算的基础能力进行高维抽象，以 API/SDK 的方式开放后端能力的访问和获取，无需开发人员配置和部署服务器资源即可获得弹性伸缩、按量付费的后端服务支持。Serverless 的技术理念其实在

数年前就已出现：后端即服务（Backend as a Service，BaaS）[4]是典型的遵循 Serverless 设计理念的服务形态，早在 2012 年 BaaS 便在开发者社区中传播并因其便捷的使用模型而深受开发者喜爱。比较典型的 BaaS 服务包括消息推送、用户认证、云存储、云数据库等。



由于 BaaS 服务大幅削减了企业在后端研发力量上的开销，其在端应用场景得到了大范围地应用。但 BaaS 核心解决的主要是垂直场景化的后端能力抽象，并没法支撑业务本身的后端逻辑部分。Function as a Service（Faas）[5]的出现弥补了这一空缺，并使得 Serverless 的架构范式能够面向端应用场景提供更加完整的闭环。

FaaS 是一种软件构建和部署的新方式，基于事件驱动模型，以函数粒度为开发者提供业务代码的托管环境。这种架构模型在数据处理、Backend for Frontend、移动应用、IoT 应用和 Web 应用等场景有较常见的应用空间。

综上我们可以看到，面向应用的 Serverless 架构包含了 BaaS 和 FaaS 两种服务形态，为了完整地支撑端应用的场景需求，成熟的 Serverless 引擎应包括：

- **消息推送**

  支持在服务器与客户端间建立可靠、省电的长连接，面向 Android、iOS、Web、IoT 等端应用提供下行消息推送能力。

- **登录认证**

  为开发者提供多维度的安全可靠的端到端身份验证能力，从而降低开发者在登录和账号体系上的开发成本和业务风险。身份验证模式包括邮箱认证、短信认证、号码认证以及主流互联网平台（淘宝、支付宝、QQ、微信、Google 等）提供的登录认证能力。

- **数据同步**

  提供一个稳定可靠、加密安全的数据同步系统，支持数据在客户端的离线使用以及在线同步更新，以提供业务在移动应用、Web 应用以及 PC 应用间的一致化用户体验。

- **远程配置**

  远程配置是面向端应用的持久配置管理服务，通过云端管理配置内容，并实时推送更新到客户端，灵活控制应用的功能、配置及 UI 实现。

- **云存储**

  提供基于 API/SDK 的便捷的云端存储能力，支持包括文本、图片、视频以及其他由用户生成的内容。

- **云数据库**

  基于云端的 NoSQL 数据库，提供面向各种前端的便捷访问接口，支持实时的数据操作、跨端数据同步和弹性伸缩。

- **云函数**

  允许开发者直接将程序托管到云函数平台上，以函数作为最小单元完成事件驱动的业务逻辑开发，通过 API 方式进行远程访问和调用。

- **AI 能力**

  应用智能化是端应用的关键技术趋势，基础 AI 能力将成为端应用 Serverless 架构的基础组成，包括但不限于 OCR、人脸识别、语音识别等。

Serverless 架构及服务带来的核心价值体现在三个方面：

- **资源成本**

  传统的应用架构模型需要预先购置一批服务器设备，并按照使用周期内的预估业务峰值来进行财务预算，不确定性因素较多，服务器资源的空置也会带来非常巨大的成本浪费。而 Serverless 的架构模型则实现了按需扩展、按量付费的弹性模型，让企业成本更可控。

- **运维成本**

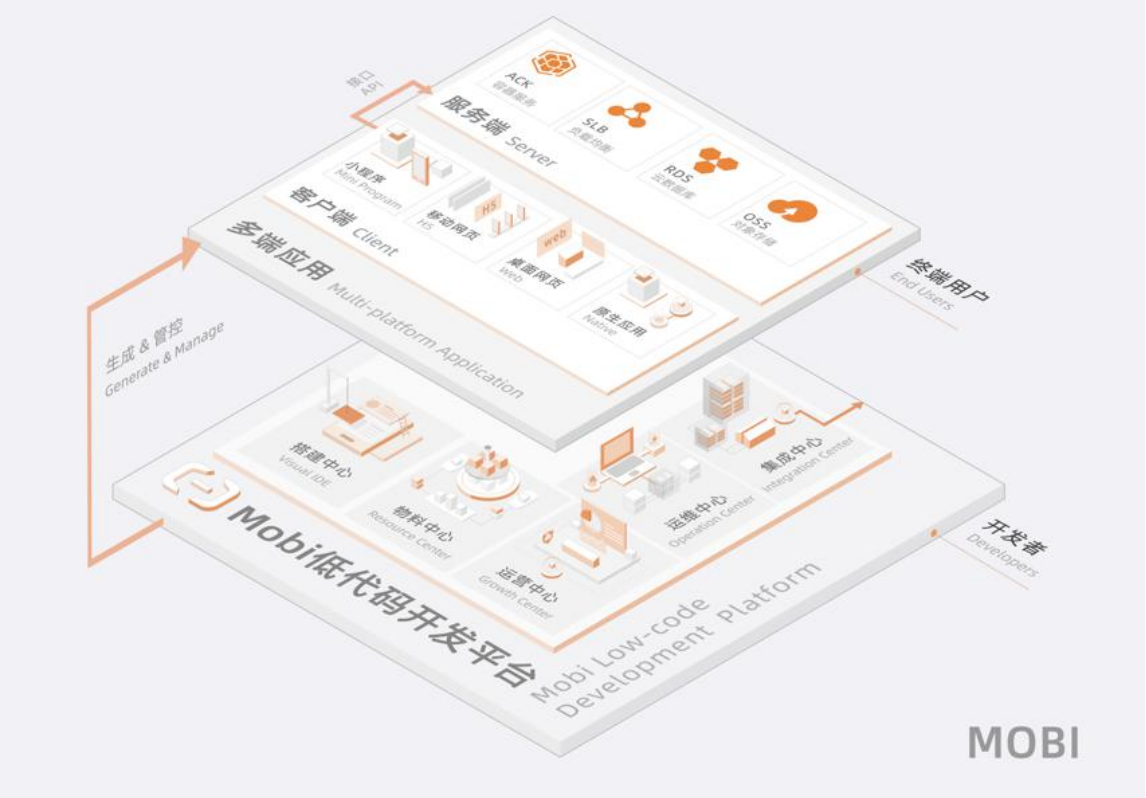
  开发者不必再关心底层计算资源的容量与日常运维问题，所有基础设施维护将会由 Serverless 服务商负责解决并对开发者透明。削减的运维成本，弹性的资源使用和可扩展·能力都将帮助开发者更好地专注于业务本身的增长。

- **研发效率**

  完整的 Serverless 引擎提供了面向端应用的绝大部分场景能力的支撑，使得应用的研发非常便捷并易于维护。而在传统的研发模型下，代码开发、环境搭建、容量压测、集群扩容、应用部署等环节都会带来巨大的时间成本。

低代码，应用研发形态的新变革

云原生技术的出现使得传统业务架构大规模地向云架构转型，软件开发效率在这个阶段也得到了明显地提升。然而数字化时代，各种应用场景地涌现，业务对 IT 面向市场的响应即时性也提出了更高的要求。在跨时代的技术演进浪潮中，Low-code Development Platform（低代码平台）[6]快速浮出水面，并伴随多样化的端应用场景开始加速普及。低代码技术为软件研发效率带来的不仅仅是提升，更是变革。



对比传统的基于手工编码方式构建应用的模型，低代码平台提供给开发者基于 GUI 的软件编辑环境，并结合云原生基础设施帮助开发者快速完成应用的搭建。同时，这样的研发模型使得软件研发对软件开发者的技能要求门槛大幅降低，更多的具备一定 IT 基础概念的人群可以参与到软件开发中，而云原生架构则天然地帮助开发者解决了软件本身的部署、运维等工作。

成熟的低代码平台将广泛应用于企业的生产、营销、BPM、工具应用等场景，其核心能力主要由两部分组成：

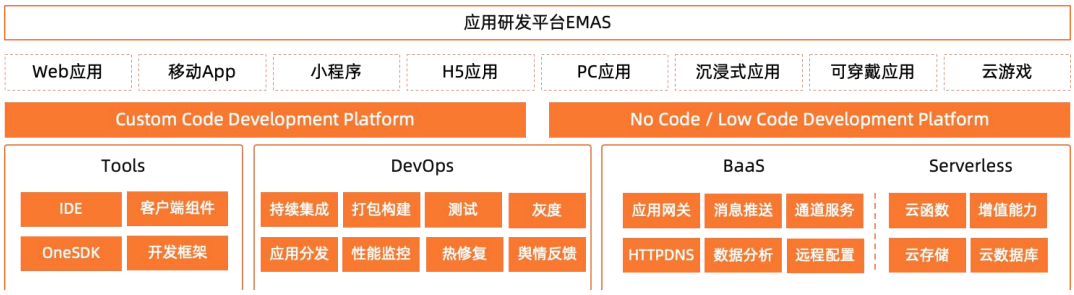
- 可视化应用编排引擎
  - 支持包括 Web、移动 App、小程序在内的多端应用场景；
  - 支持包括 UI 可视化编排、业务流编排、逻辑编排、数据编排等能力；
  - 支持组件、模块、模板等模型，有开放化的物料系统和组件市场；
  - 模型驱动，搭配元数据解析引擎(包括多端转译引擎/渲染引擎等)；
- 云原生应用平台
  - 行业化领域模型与元数据管理
  - 代码生成引擎
  - 云原生应用托管
  - CI/CD
  - 丰富的集成与扩展能力

我们可以从施耐德电气与顶级低代码平台公司 Outsystems 的合作中看到低代码带来的价值：施耐德电气在应用低代码平台后，在短短 20 个月的时间内快速上线了 60 款 App，其中绝大多数 App 在 10 周内完成开发和上线，第一年节省的人工成本达到 650 人天。低代码技术大幅缩短了传统企业数字化转型的路径。

至 2024 年，Gartner 预计所有应用程序开发活动当中的 65%将通过低代码的方式完成，似乎比想象的更快一些，但它确实在持续地发生。在全球市场，我们能够看到 OutSystems、Mendix、PowerApps、App Maker 快速的成长脚步，未来结合 AI 与机器学习，我们可以预见真正的“App 工厂”的诞生。

求变应变，永不止步成就技术革新

作为国内移动互联网、云计算领域的行业巨擘，阿里巴巴在大前端、云原生领域有着丰富的实战经验。我们从 2016 年开始逐步将集团内部成熟的应用中间件云化输出，并在 2018 年推出了移动研发平台 EMAS（https://cn.aliyun.com/product/emas），如今，EMAS 已经逐渐成长为横跨多端（移动 App、H5 应用、小程序、Web 应用等）场景的云原生应用研发平台，基于广泛的云原生技术（Backend as a Service、Serverless、DevOps、低代码等），为企业、开发者提供一站式的应用研发管理服务，涵盖开发、测试、运维、运营等应用全生命周期。



截止到今天，伴随云计算的迅速普及和发展，我们已服务了 15 万以上的企业与开发者。（EMAS 开发者版套餐免费订阅）。

在海量的生产实践中，我们也看到了云原生技术在端应用场景下所面临的的关键挑战：

- **研发心智的改变**

  对于所有开发团队而言，前后端团队的定义根深蒂固，协同界面已成自然。然而随着 Serverless 等云原生技术地广泛应用，在越来越多的端应用场景中，开发团队仅需前端开发人员即可以闭环完成应用的研发和上线工作；在应用架构维度，Serverless FaaS 带来的是基于事件驱动，无状态，函数式逻辑片段的全新范式，与传统的应用模型有着较大的区别。改变即成本，更关键的是改变背后不是纯粹的技术，还有组织的变革，生产关系的变革。

- **技术成熟度**

  无论是 Serverless FaaS 还是低代码开发，都属于商业导入期的技术，产品化完善度还有欠缺，所能覆盖的场景也有一定的局限性，对于主流的复杂应用场景，Serverless FaaS 还需结合传统微服务等架构形成混合式的 Serverless 应用。在系统可观测性，研发调试便捷性，函数启动性能，函数执行时长等技术关键控制点上依然有较长的路要走。

- **架构灵活度**

  云原生能力代表了对云基础设施的高维封装和抽象，抽象即意味着管控粒度变粗，系统的灵活性与定制扩展能力会天然地受到一定的削弱；

虽然挑战巨大，但趋势已现。EMAS BaaS 已经成为国内大量移动 App 的必备基础设施，全球范围内覆盖超过 20 亿的活跃设备终端，每天的 API 调用量超过百亿规模；基于 EMAS Serverless（https://www.aliyun.com/product/miniappdev），疫情期间我们看到大量开发者快速地实现了防疫抗疫工具应用的开发，从诞生想法到产品上线历时仅需一周；在企业内部，越来越多的办公应用和表单应用基于低代码平台快速构建，企业生产力得到了大幅提升。参照 Hype Cycle[7]的模型，云原生的多项新兴技术已经涌现大量生产实践项目，技术成熟度已然进入了稳步爬坡期。

毫无疑问我们站在了又一个技术纪元的前夜，云端一体，一云多端正在成为前台应用研发的事实标准，拥抱云原生将成为开发者享受云计算红利的最短路径。我们欢迎更多的有识之士加入我们（ lingming.yb@alibaba-inc.com ），变革生产力，改变世界。

欢迎加入 EMAS 开发者钉钉交流群

群号：35248489

REFERENCES

[1] Use MASA to Deliver an Agile Multiexperience Enterprise Application Architecture, Gartner, 2019

[2] Top 10 Strategic Technology Trends for 2020: Multiexperience, Gartner, 2020

[3] Technology Insight for Multiexperience Development Platforms, Gartner, 2020

[4] https://en.wikipedia.org/wiki/Mobile_backend_as_a_service, WIKIPEDIA

[5] https://en.wikipedia.org/wiki/Function_as_a_service, WIKIPEDIA

[6] https://en.wikipedia.org/wiki/Low-code_development_platform, WIKIPEDIA

[7] https://en.wikipedia.org/wiki/Hype_cycle，WIKIPEDIA

2.EMAS 产品全景介绍

开发更便捷，阿里云推出一站式应用研发平台 EMAS 2.0

简介：阿里云一站式应用研发平台 EMAS 2.0 正式发布，本次将全新发布包括 Serverless、低代码开发平台 Mobi、AI 工具箱、业务组件等产品服务，同时推出面向中小微企业和开发者的免费套餐扶持计划。开发者可在云时代以更低成本、更便捷地完成应用的开发和维护。

1 月 13 日，阿里云一站式应用研发平台 EMAS 2.0 正式发布，本次将全新发布包括 Serverless、低代码开发平台 Mobi、AI 工具箱、业务组件等产品服务，同时推出面向中小微企业和开发者的免费套餐扶持计划。开发者可在云时代以更低成本、更便捷地完成应用的开发和维护。

低代码开发平台 Mobi

低代码开发平台 Mobi 是一站式可视化的应用研发平台，提供应用前后端的搭建、托管、运维服务，适用于 H5、全平台小程序。帮助应用开发者跨越底层技术壁垒，聚焦于业务场景实现，大幅降低开发门槛和成本，提升开发效率和投入产出比。 以独立应用、独立资源占用的方式标准化输出。

低代码研发平台 Mobi 支持可视化应用搭建 IDE、模型设计、物料管理、集成管理、模型版本管理、调试预览，以及应用运维环境管理、产物版本管理、日志记录、性能监控，同时还有数据监控看板、数据分析等产品功能。

小程序 Serverless

小程序 Serverless 提供包括云函数、数据存储、文件存储等一整套后端服务。开发者通过 API 方式即可获取云函数、数据存储、文件存储、音视频、图像处理等服务，不需要关心服务器或底层运维设施，可以更专注于代码和业务本身。帮助企业快速、低成本地实现一云多端的业务战略。

EMAS 新能力

从服务开发者提高开发效能、应用质量，到更好的帮助客户实现业务增长、变现等核心诉求；近期还将推出视频点播/直播的音视频能力、云码广告的流量变现能力、AI 工具箱（OCR）、号码认证平台服务等，尽请期待。

开发者扶持计划

阿里云此次还升级了开发者扶持计划，EMAS 2.0 公共云版本将全面升级为订阅服务模式，面向广大开发者，提供免费的的开发者版，包括 DevOps、平台服务、用户增长在内的相关板块的一系列免费服务支持。面向一定规模的企业，将提供企业版订阅服务，提供增值产品能力以及专属钉钉技术支持。EMAS 2.0 的订阅模式将会更大规模地让利开发者，帮助大家开启云端技术，智造未来应用。

EMAS 产品技术负责人杨锲表示，EMAS 2.0 将全面利用云计算的服务模型优势，结合阿里巴巴经济体的多元能力沉淀，为开发者提供更完善的应用研发服务。并基于全新的六大核心技术理念（一云多端、云端一体、云原生、低代码、AI 驱动、链接业务）进行此次升级。

- **一云多端，Multiexperience**
  提供多样化用户体验，面向包括移动端、Web、小程序、PC、IoT、AR/VR 等场景实现无缝、一致的数字用户旅程。
- **云端一体，One-Stop**
  一站式提供端开发工具与后端应用构建和托管基础设施，企业仅需具备前端技术栈的专业人才即可低成本完成端到端应用开发。
- **云原生，Cloud Native**
  通过 DevOps，Serverless 等云原生标准技术赋能端应用开发，实现高效率研发、自动化交付、低成本运维。
- **低代码，Low Code**
  通过可视化应用搭建引擎实现低成本的应用开发和维护，为开发者提供 Pro Code 和 Low Code 两种应用研发引擎。

- **AI 驱动，AI Driven**

  全面开放阿里巴巴智能 AI 算法能力，通过端 SDK、API 实现各种用户端场景下的快速集成以及端计算能力。

- **链接业务，Linking Business**

  开放行业插件系统，提供包括零售、金融、本地生活、多媒体在内的多种业务能力。

截止到目前为止，EMAS 已累积了近 20w 的开发者和企业用户，为数十万的 APP 的日常运营保驾护航。

了解更多 EMAS 产品 https://www.aliyun.com/product/emas

加入应用研发平台 EMAS 开发者交流群，钉钉群号：35248489

3.EMAS 客户案例

杭州银行：坚定投入移动化战略，借助 EMAS 研发平台迈入移动开发 3.0 时代

简介：可以想见，未来银行的绝大部分业务、渠道、连接点，都不免与手机银行打通，以实现线上、线下服务的无缝连接，并以此构建起新的银行形态。未来的手机银行将是银行产品创新、业务拓展和战略转型的重要平台，EMAS 移动研发平台已经为杭州银行打下了基础。

一、行业背景

2019 年是中国银行业进入移动化时代的第 20 个年头：1999 年，在移动运营商的支持下，国内第一个手机银行上线，服务范围覆盖全国 26 个经济发达的重点城市。随后，各大银行相继推出基于手机短信和 WAP 网络服务的手机银行服务，开启了国内手机银行发展的第一个黄金十年。

随着手机银行业务的不断丰富以及用户对手机银行体验、速度和功能的要求越来越高，传统的短信银行、WAP 银行已经不能满足用户的需要。与此同时，智能手机、3G 移动通信网络以及安全技术也在这十年间得到了飞速发展，2010 年开始手机银行进入了客户端（即手机银行客户端，以下简称 APP）时代，手机银行由此进入又一个飞速发展的十年。

根据中国银行业协会的数据显示，截止到 2017 年，国内商业银行手机银行个人用户数量已达 15.02 亿户，同比增长 28.28%；2017 年全年手机银行交易达 969.29 亿笔，同比增长 103.24%，交易金额 216.06 亿元，同比增长 53.7%。

与其他国家相比，我国手机银行的发展时间虽然不长，但得益于我国飞速发展的移动互联网和移动支付，手机银行的市场环境和用户习惯基础极好。在我国，手机银行已经不仅仅是银行柜台业务或传统渠道的补充，而是成为了数字银行、智慧银行的重要承载，更促使我国银行业（尤其是中小银行）坚定的朝着移动化的方向发展。

二、杭州银行：稳步有序的移动化发展战略

2012 年，经过两年快速发展，国内各银行纷纷推出支持 Android 和 iOS 的 APP，并逐渐成为主流模式，同时积累了一定的技术经验和开发资源。作为一家始终坚持服务区域经济、中小企业和城乡居民的区域性股份制银行，杭州银行一直致力于为客户提供专业、便捷、亲和的金融服务，至 2013 年，杭州银行在作出国内银行业移动化市场日趋成熟、生态逐渐建立、智能手机成为未来移动办公和银行服务的重要载体的判断后，2013 年确定了移动化战略并投入资源开始进行移动端开发。

据杭州银行信息技术部软件开发三部经理周炼介绍，杭州银行的移动端开发历程在 2018 年前可以分为两个阶段：第一个阶段是 1.0 阶段，属于“零经验的起步阶段”，在这个阶段主要以外部开发合作伙伴为主；第二个阶段是 2.0 阶段，在通过前一阶段积累了一定的经验后，杭州银行开始自主研发移动端，研发了自主化的移动端框架，并将移动端开发从外部开发合作伙伴为主转向了以内部开发为主、合作伙伴为辅的开发模式。

经过两个阶段的发展，杭州银行的移动化战略已经有了比较坚实的基础，内部团队具有了一定的开发能力和开发经验，但在多种因素的影响（包括前期主要采用的“以 H5 为主，原生为辅”的混搭开发模式），杭州银行在移动端开发上存在进一步完善的空间。

在 2018 杭州云栖大会上，杭州银行信息技术部技术总监徐建芳曾就杭州银行的移动端开发痛点做了详细的描述，她表示，杭州银行在移动端开发的痛点分为两个层面：

- 架构层面的痛点：包括消息推送到达率低，没有跟大部分手机做适配；数据分析、移动测试等模块较为分散，没有深度结合；缺少移动端性能监控和评估，对于端上 APP 运行情况没有整体把控；iOS 版本缺少应急修复机制，出现问题需要较长的修复周期；缺少多维度的灰度发布机制。
- 开发模式的痛点：包括机型适配工作量大；出现闪退等情况，没有详细日志信息，无法及时定位问题；不能充分做到跨终端的效果，开发周期长；多人协作开发的效率和版本管理方面不足。

与此同时，近几年互联网服务和互联网金融的发展对杭州银行的员工和客户都产生了非常明显的影响，移动办公、移动银行的概念和使用习惯深入人心，无论是行内人员还是外部客户，都对移动端的运行速度、使用便捷性、服务体验、应用稳定性有了更高、更严的要求，周炼表示，在以上这些因素的影响下，杭州银行的移动端开发开始进入 3.0 阶段。

“在 3.0 阶段，移动端的关键是体验，我们定义的 3.0 就是移动端服务体验的升级，与此同时，还要提升开发团队的知识和能力水平，让大约 50 人的移动端开发团队的开发经验能够互相支持、互助提高。” 周炼表示，在 3.0 阶段，杭州银行选择与阿里云合作，以阿里云 EMAS 移动研发平台为基础，结合阿里巴巴在移动端开发领域的经验，提升杭州银行移动端的用户体验。

三、开发平台慎重选型：EMAS 融入杭州银行长期移动化战略

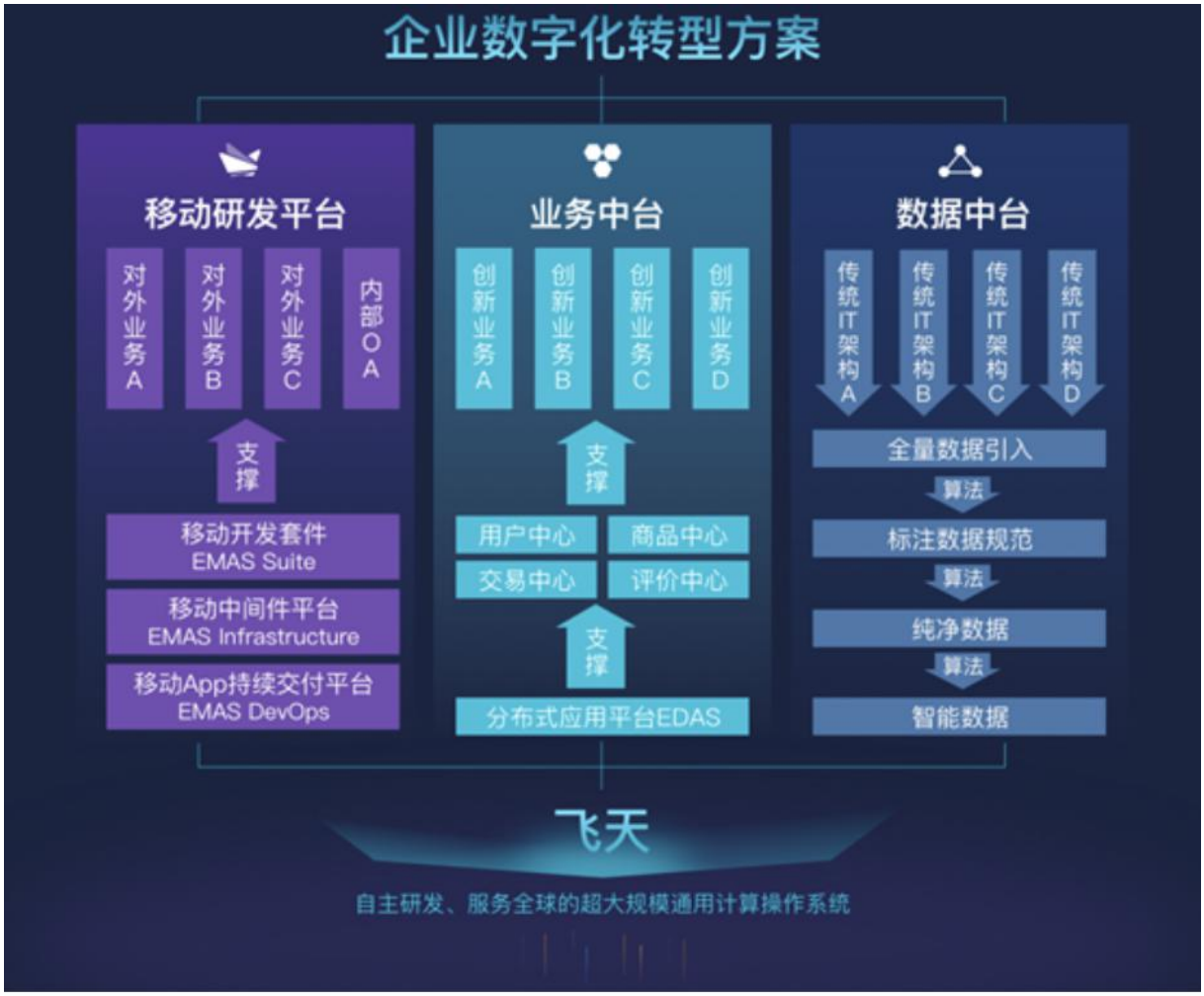
移动研发平台是企业移动化战略的基础，它不仅决定了企业移动端开发的效率、质量、成本、可靠性；更决定了企业是否能够及时、有效、符合需求的为最终客户提供移动化服务，支持传统线下业务向线上转化和传统 PC 服务向移动端转化。

因此，企业移动研发平台的选型维度一般会有如下四个切入点：

- 统一的研发管控平台：可以统一的管理开发、测试、构建、发布、性能监控、热修复等工作，在 APP 完整的生命周期中形成闭环；
- 具备“一次开发，多平台运行”的能力：同时支持 Android 和 iOS 平台，无需重复开发，同时支持 H5、WEEX、原生等开发模式；
- 完善、稳定的功能组件库支持，提高开发速度，加速开发进度，避免重复性开发底层、基础功能；
- 移动接入网关，提供 API 管理、限流、MOCK 等功能，以及和后台服务的无缝对接；

以上四个切入点同样得到了杭州银行的关注，杭州银行最初的移动研发平台诉求与此类似，也同样可以归纳为四点：1、具备打造跨平台、兼容性高的 APP 的能力，防止出现闪退等情况；2、具备快速更新的能力，可以在线更新（热修复），对于小代码量的补丁包可以做到无感更新；3、具备信息采集的能力，能够支持杭州银行的对客户反馈、客户使用情况的收集；4、要比原有 H5 有更好的效果，但也要避免大量的原生开发，既要保证客户（的原生 APP）体验，也要满足（类似 H5 的）开发上线速度的要求。

经过详细的评估对比，杭州银行最终选择了 EMAS 作为其移动研发平台，同时首选在内部移动办公 APP 上开始落地：“杭州银行的内部移动办公 APP 并不统一，内部有多个 APP，这对行内人员的使用体验、使用效率和使用安全上都有影响，所以在 EMAS 移动研发平台上的第一阶段工作，我们希望把内部 APP 整合到一起服务行内人员，为内部移动办公提供更好的体验和更及时的问题处理。”



阿里巴巴面向企业数字化转型的中台能力输出

杭州银行移动办公系统（内部移动办公 APP）的服务对象为所有行内人员，是行内办公应用在移动端的延伸，正如周炼所说“内部移动办公 APP 非常重要，它是杭州银行提高团队内部资源和知识共享水平；充分利用时间空间，提升工作效率；促进无纸化办公模式的主要手段。”

内部移动办公 APP 的研发非常需要 EMAS 移动研发平台的支持，一方面，内部移动办公 APP 经常需要变更，版本迭代快，特殊化需求较多，对基础组件要求高；另一方面，内部移动办公 APP 涉及到的业务部门较多，业务种类繁多，需要多人协作开发；与此同时，由于是内部移动办公 APP，因此不允许发布应用市场，需要平台来解决发布部署，这些问题促使杭州银行首选内部移动办公 APP 作为 EMAS 移动研发平台的第一个落地实践应用。

“在阿里云团队的支持下，杭州银行的开发团队先是内部熟悉了平台，组建和调整了开发团队，继而基于 EMAS 移动研发平台实现了内部移动办公超级 APP 的上线，我们看到 EMAS 在开发效率、APP 启动速度、无感更新、信息收集、问题定位、跨平台开发、设备兼容性等多个方面，

产生了很大的帮助。”

随着杭州银行内部移动办公 APP 升级为移动办公超级 APP，这一内部应用不仅很好的融合了过去 H5、WEEX 和原生应用的统一入口、统一接入，融合了包括公告，会议通知、工作动态、待办提醒、审批，CRM 等完整功能（比如说，行内人员在 APP 上统一认证登录后，即可访问行务公告、公文、会议通知、移动审批、你问我答、通讯录等应用），为行内人员提供了集中化的服务体验，同时对于开发团队来说，将原来已有的 APP 经过很简单的转换，就可以作为组件 Bundle 接入超级 APP，极大的减少了内部移动办公 APP 融合的工作量。

与此同时，由于 EMAS 提供了许多公共化的功能，杭州银行的开发团队不必再在用户登录等通用模块上投入太大精力，“我们不需要太多关注通用模块、开发框架，而是可以专注于做自己的业务开发、创新功能特点，团队的开发效率和开发专注度有很大的提高。”

四、EMAS：从支持移动端开发到赋能移动开发团队

EMAS 移动研发平台为杭州银行的移动办公系统转型升级移动办公超级 APP 提供了有效的支撑，但其所起到的作用却不仅仅局限于技术赋能，对于杭州银行来说，EMAS 移动研发平台的引入为移动开发团队自身的成长同样带来了新的价值。

传统移动开发团队为了同时满足 Android 和 iOS 跨平台的开发需求，往往分为两个专项团队，这意味着即使是相同的功能，由于针对两个平台、分属两个团队，仍然需要进行重复开发，这不仅导致了极大的人力、物力、时间上的浪费，而且两个团队之间在开发技能、开发经验和安全体系上，也由此存在着差异和差距。

随着采用 EMAS 移动研发平台，杭州银行可以通过 WEEX 技术实现跨平台开发，而且由于 WEEX 在底层采用原生渲染，在性能和功能实现度上很接近原生开发，不仅客户应用体验几乎与原生 APP 毫无差别，而且将 Android 和 iOS 两个研发团队集中在一个平台上，实现基于 EMAS、面向 APP 功能（而不是跨平台或硬件兼容性）的合作开发，聚焦 APP 的具体功能、业务创新，特别是促进了开发团队成员之间的经验分享，避免了原有两个专项团队并立导致的“重复踩坑”的问题，也让开发人员在知识体系掌握方面更为全面。

EMAS 移动研发平台的引入，让杭州银行的移动开发团队建立了一个“直通车式”的产品、技术人员对接体系，业务需求可以对接到人、精准反馈，客户问题可以精准定位，开发团队不再被 APP 开发本身的技术问题占用大量精力，从而能够在提高业务开发能力、前瞻性技术开发上投

入更多的时间。

随着基于 EMAS 移动研发平台的移动办公超级 APP 上线，杭州银行在下一阶段计划将手机银行、托管银行、直销银行等移动客户端切换到 EMAS 上，从而为外部银行客户提供更好的移动服务体验，以此支持杭州银行在当前银行业移动化服务爆发大背景下的移动化战略拓展。

事实上，是否能够为客户提供更好的移动化服务，已经成为决定银行未来发展的决定性因素。据 2018 年手机银行市场调研结果显示，手机银行认知度达到 97.8%，渗透率（使用手机银行人数占比）达到 91.1%，手机银行已经成为客户（尤其是零售客户）与银行之间建立连接的核心触点。

可以想见，未来银行的绝大部分业务、渠道、连接点，都不免与手机银行打通，以实现线上、线下服务的无缝连接，并以此构建起新的银行形态。未来的手机银行将是银行产品创新、业务拓展和战略转型的重要平台，EMAS 移动研发平台已经为杭州银行打下了基础。

4.隐私保护政策下 EMAS 的产品升级

EMAS 发布最新隐私协议，为客户信息安全保驾护航

简介：公民个人信息不容侵犯，数据显示，近年来工信部持续开展 APP 侵权整治活动，开展了六批次集中抽检，检查了 76 万款 APP，通报 748 款违规 APP，下架了 245 款拒不整改的 APP。移动研发平台 EMAS 高度重视个人信息的保护，在客户使用 EMAS 服务时，将根据法律法规要求并参照行业最佳实践为客户个人信息安全提供充分保障。现 EMAS 最新隐私协议已发布，将全方面为客户的信息安全提供保障。

公民个人信息不容侵犯，确保 APP 不“越界”，国家一直在行动。数据显示，近年来，工信部持续开展 APP 侵害用户权益的整治活动，开展了六批次集中抽检，检查了 76 万款 APP，通报 748 款违规 APP，下架了 245 款拒不整改的 APP。

友商案例

以极光举例，是通过提供消息推送，即时通讯、统计分析、社会化组件和短信等开发者服务累计数据和标签，继而通过精准营销、金融风控、市场洞察和商业地理服务等数据业务进行变现。在南方都市报发表于 2020 年 11 月 27 日的文章中被指出存在使用违规收集个人信息的第三方组件的问题，此外腾讯、个推、小米 SDK（软件开发工具包），以及旧版本的 360 加固工具同样被揭露存在此类问题。

阿里云数据隐私保护

阿里云作为业界首家发起发起《数据保护倡议》的企业，数据隐私保护是阿里云的第一原则。

隐私保护采取严苛的标准：全球首张云安全国际认证金牌（CSA-STAR） / 亚洲合规资质最全的云服务商 / 全球第一张 ISO / 22301 国际认证证书 / 第一家同时完成德国 C5 云安全基础要求和附加要求评审的云服务提供商 / 欧盟数据保护法规 GDPR / 等保 2.0 四级。

EMAS 发布最新隐私协议

移动研发平台 EMAS 高度重视个人信息的保护，在客户使用 EMAS 服务时，将根据法律法规要求并参照行业最佳实践为客户的个人信息安全提供充分保障。为此，EMAS 制定本《隐私权政策》以帮助客户充分了解 EMAS 平台对客户和最终用户的个人信息的处理方式。

隐私政策内容：

一、我们如何收集和使用您以及您最终用户（以下简称客户）的个人信息
二、我们如何共享、转让、公开披露客户的个人信息
三、我们如何保护客户的个人信息
四、客户如何行使个人信息的权利
五、我们如何处理未成年人的信息
六、客户的个人信息储存与全球范围转移
七、本隐私政策如何更新
八、如何联系我们

该隐私政策适用于 EMAS 移动推送/HTTPDNS/移动热修复/远程日志/崩溃分析/性能分析/移动用户反馈等产品。

现 EMAS 最新隐私协议已发布，为客户信息安全保驾护航。

5.EMAS 旗下移动性能测试

云上的移动性能测试平台

简介：功能决定现在，性能决定未来。欢迎大家围观《云上的移动性能测试平台》，了解 EMAS 性能测试平台的能力与规划。

1. 功能决定现在，性能决定未来

性能测试在移动测试领域一直是一个大难题，它最直观的表现是用户在前台使用 App 时的主观体验，然而决定体验优劣的背后，涉及到了许许多多的技术变迁。

- 当我们习惯于诺基亚时，智能机出现了；当我们学会 native 开发时，hybrid 来了；当各种 hybrid 框架下的巨型应用倾向成熟时，小程序出现在我们眼前；紧接着直播、iot、ar、vr、人工智能，新的技术与应用场景正在以无法想象的速度向前发展。性能测试技术在快速变化的场景与开发技术面前，面临着巨大的挑战，当我们还在纠结如何测试 a 时，b 就已经出来了。
- 性能测试本身，有发展日渐成熟的解决方案，如线上性能监控 APM、线下性能采集工具；有基于各个应用场景衍生的测试技术，如压力测试、稳定性测试、功耗测试等；也有基于各项性能指标（内存、cpu、电量、流量）而来的各种专项测试能力。

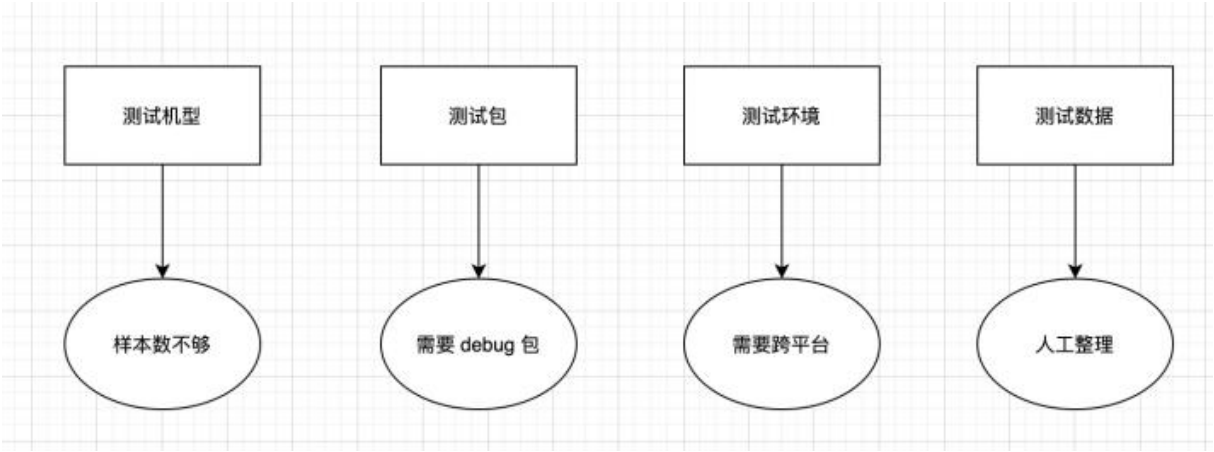
我们致力于打造线上线下一体的性能解决方案，希望能够帮助开发者发现、定位与解决一系列移动端性能问题。本文将着重介绍 EMAS 性能测试平台的能力与规划，还是那句话，功能决定现在，性能决定未来。

2. 云上的性能测试工具

通常我们在进行专项测试（内存、cpu、电量、流量等）时，需要准备测试机型、测试包、测试环境、测试数据，会遇到以下问题：

- 机型样本数不够多。
- debug 包不一定真实反应生产包的性能，但 Android Studio 需要 debug 包才能测试。
- Android/iOS 测试环境的搭建与跨平台困难。
- 大量测试数据整理分析。

以上这些问题很容易导致整个测试效率的低下，甚至无法实施落地。

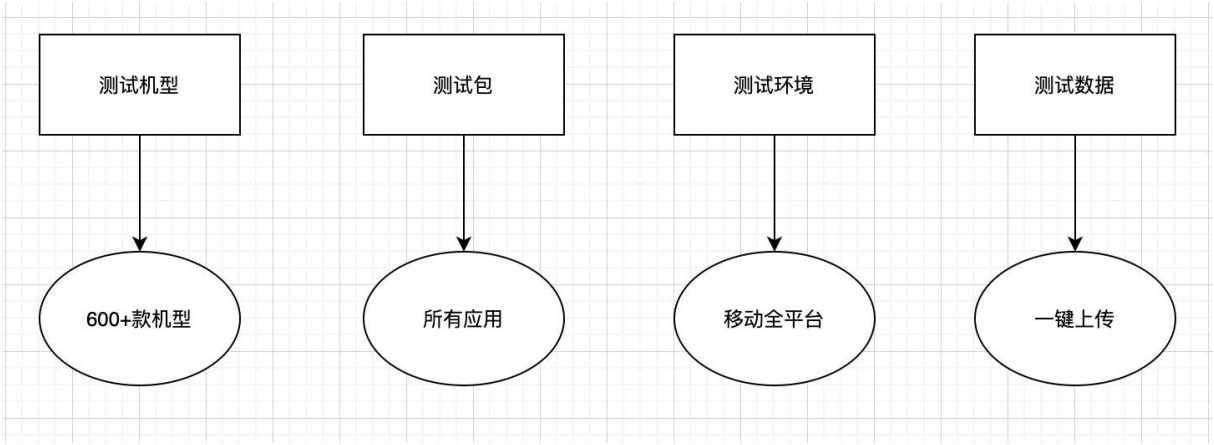


基于 EMAS 云真机的调试能力，MQC 在云上提供了更加完美便捷的性能测试工具。

云真机天然提供多达 600+款测试机型，支持调试测试所有已安装应用，不依赖任何本地环境配置，测试数据一键上传统计。

同时，EMAS 性能测试工具有以下特点：

- 基于 app_process 与 instruments 协议实现的移动双端、跨平台性能采集；
- 无侵入、短间隔（采集间隔稳定 1s），低延迟（性能数据延迟小于 100ms）、低功耗（对设备性能影响低于 1%）；
- 应用+进程的测试方案，满足 hybrid、小程序的测试需求。



3. 云上数据看板

性能数据的意义在于它将我们常见的各种问题通过技术的手段进行度量与量化，可以帮助我们在产品功能上线前，尽可能的发现潜在的性能问题和风险。MQC 性能测试平台将存储于云上的数据，以尽可能多的维度可视化给用户，把好版本发布前的关口。

3.1 任务

用户使用云真机进行的每一次测试任务与性能数据，都会直接保存成测试任务，方便再次查看与确认历史数据。

3.2 用例

在实际的测试过程中，我们很容易发现，不同应用场景的性能数据是完全没有可比性的，在统计方法上只看性能数据的平均值也很难直接给出定性和定量的判断，没办法影响开发、产品的决策。

即便看似相同的场景，不同的产品决策也可能带来很大的性能数据差距：比如大多数云盘的相册基于流量与性能的考虑，显示的都是压缩后的图片；而我们的本地的一些相册软件，显示的基本都是原图，这样产品上的选择便导致了内存开销上巨大的差异。

数据看板在最初设计的时候就吸取了功能自动化用例平台时的建设经验，将每一次性能测试任务分用例存储，并且按照不同的用例维度对性能数据进行统计。在 EMAS 移动测试 控制台，不同子账号查看和管理相同的 app 与用例，满足多用户云端协作的需求。

3.3 多维度聚合

在用例维度的基础上，MQC 性能测试平台提供了多个维度的数据统计、聚合与分析能力。

- **设备分级**

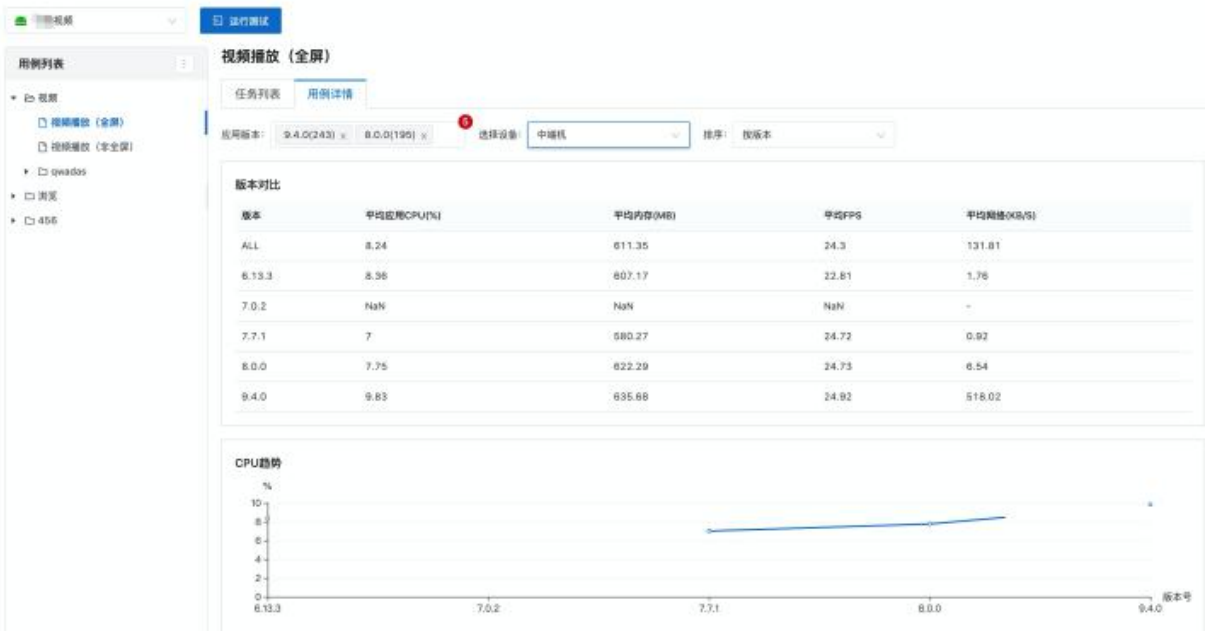
根据设备硬件性能打分，划分为高、中、低三个级别。

由于不同等级机型对 APP 实际性能指标的影响较大，这个因子可以较大程度减少硬件性能对指标置信度的影响。

- **应用版本**

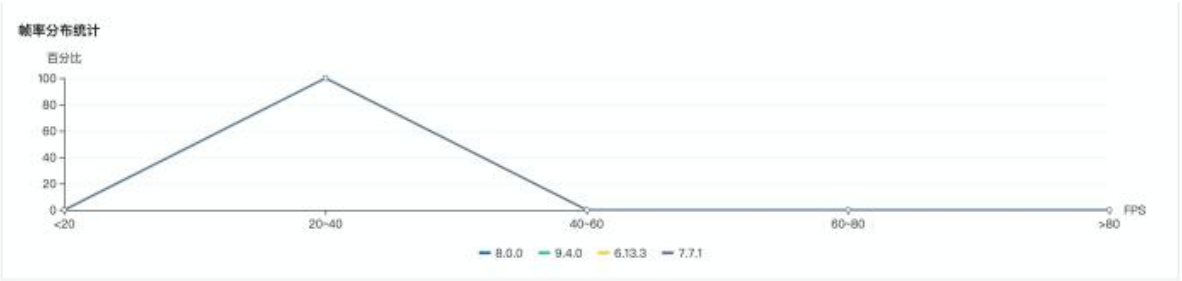
对于性能指标来说，通常有三种问题判断标准：

  - 基于行业技术经验定义的基线指标，这通常是技术决策者基于用户体验、性能要求、大数据分析给开发提出的底线标准；
  - 同行业 APP 性能指标横向对比，学习行业内优秀的技术实现一直是互联网快速发展的重要原因之一；
  - 同 APP 不同版本间的纵向对比，快速的发现新版本的优化效果，新功能对 APP 的影响等。



- 指标分布

  指标分布可以帮助开发者快速判断指标区间，定位可能的异常任务与异常指标区间，更加有针对性的去查阅任务报告。



4. 未来规划

- 丰富指标：我们将继续完善更多性能指标的采集方案，如电量、GPU、温度等；
- 行业指标：MQC 将基于云上开发者数据、专家测试数据，整理并统计各个行业性能指标作为参考分享出来；
- 性能基线：上文提到，通常性能指标的观察标准有三，同行业 APP 性能指标；同 APP 不同版本性能指标；基于技术方案与行业数据的性能标准。性能基线的定义可以更好的约束开发者对极致性能体验的追求，最大可能降低性能问题出现的概率，如 OOM、ANR 问题。

最后附上演示视频，请点击下方链接查看

https://developer.aliyun.com/live/245866

6.EMAS 旗下低代码 mobi 产品背景

什么是低代码（Low-Code）？

简介：什么是低代码？我们为什么需要低代码？低代码会让程序员失业吗？本文总结了低代码领域的基本概念、核心价值与行业现状，带你全面了解低代码。

作者 | 阿里云 云原生应用研发平台 EMAS 彭群（楚衡）

一、前言

如果选择用一个关键词来代表即将过去的 2020 年，我相信所有人都会认同是“新冠”。疫情来得太快就像龙卷风，短短数月就阻断了全世界范围内无数人与人之间的物理连接。但好在，我们已经全面迈入互联网时代：N95 口罩再厚，也阻挡不了信息比特流的顺畅流通（宅男：B 站依然香）；居家隔离再久，也妨碍不了钉钉消息的准时送达（社畜：工作依然苦）。逍遥子在 9 月份的云栖大会上说：“新技术代表的新生产力，一定是我们全速战胜疫情、开创未来最好的原动力。” 那么在后疫情时代，究竟需要什么样的新技术，才能真正解放 IT 生产力，加速社会数字化转型，Make The World Great Again？我认为是低代码（Low-Code）。

基于经典的可视化和模型驱动理念，结合最新的云原生与多端体验技术，低代码能够在合适的业务场景下实现大幅度的提效降本，为专业开发者提供了一种全新的高生产力开发范式（Paradigm Shift）。另一方面，低代码还能让不懂代码的业务人员成为所谓的平民开发者（Citizen Developer），弥补日益扩大的专业人才缺口，同时促成业务与技术深度协作的终极敏捷形态（BizDevOps）。本文将重点介绍低代码相关背景知识，包括低代码的定义与意义、相关概念、行业发展等，期望能帮助大家更好地认识与理解低代码这个新兴领域。

二、什么是低代码

“Low-Code”是什么？如果你是第一次听说，没准也会跟我当年从老板口中听到这个词后的内

心戏一样：啥？“Low-Code”？“Code”是指代码我知道，但这个“Low”字是啥意思？不会是老板发现我最近赶工写的代码很丑很“Low”吧... 想多了，老板怎么可能亲自 review 代码呢。那难道是指，“Low-level programming”里的“Low”？老板终于发现让我等编程奇才整天堆 Java 业务代码太浪费，要派我去闭关写一个高性能 C 语言网络库... 显然也不是，老板哪能有这技术情怀呢。那到底是什么意思？作为一名搜商比情商还高的程序员，能问 Google 的绝不会问老板。于是我一顿操作后，不假思索地点开了第一条搜索结果：Low-code development platform。

Wikipedia 定义

从 Wiki 的这段定义中，我们可以提炼出几个关键信息：

- 低代码开发平台（LCDP）本身也是一种软件，它为开发者提供了一个创建应用软件的开发环境。看到“开发环境”几个字是不是很亲切？对于程序员而言，低代码开发平台的性质与 IDEA、VS 等代码 IDE（集成开发环境）几乎一样，都是服务于开发者的生产力工具。
- 与传统代码 IDE 不同的是，低代码开发平台提供的是更高维和易用的可视化 IDE。大多数情况下，开发者并不需要使用传统的手写代码方式进行编程，而是可以通过图形化拖拽、参数配置等更高效的方式完成开发工作。

Forrester 定义

顺着 Wiki 的描述还能发现，原来“Low-Code”一词早在 2014 年就由 Forrester 提出了，它对低代码开发平台的始祖级定义是这样的：

相比 Wiki 的版本，这个定义更偏向于阐明低代码所带来的核心价值：

- 低代码开发平台能够实现业务应用的快速交付。也就是说，不只是像传统开发平台一样“能”开发应用而已，低代码开发平台的重点是开发应用更“快”。更重要的是，这个快的程度是颠覆性的：根据 Forrester 在 2016 年的调研，大部分公司反馈低代码平台帮助他们把开发效率提升了 5-10 倍。而且我们有理由相信，随着低代码技术、产品和行业的不断成熟，这个提升倍数还能继续上涨。
- 低代码开发平台能够降低业务应用的开发成本。一方面，低代码开发在软件全生命周期流程上的投入都要更低（代码编写更少、环境设置和部署成本也更简单）；另一方面，低代码开发还显著降低了开发人员的使用门槛，非专业开发者经过简单的 IT 基础培训就能快速上岗，既能充分调动和利用企业现有的各方面人力资源，也能大幅降低对昂贵专业开发者资源的依赖。

低代码核心能力

基于上述的定义和分析，不难总结出如下这 3 条低代码开发平台的核心能力：

- **全栈可视化编程：**可视化包含两层含义，一个是编辑时支持的点选、拖拽和配置操作，另一个是编辑完成后所及即所得（WYSIWYG）的预览效果。传统代码 IDE 也支持部分可视化能力（如早年 Visual Studio 的 MFC/WPF），但低代码更强调的是全栈、端到端的可视化编程，覆盖一个完整应用开发所涉及的各个技术层面（界面/数据/逻辑）。
- **全生命周期管理：**作为一站式的应用开发平台，低代码支持应用的完整生命周期管理，即从设计阶段开始（有些平台还支持更前置的项目与需求管理），历经开发、构建、测试和部署，一直到上线后的各种运维（e.g. 监控报警、应用上下线）和运营（e.g. 数据报表、用户反馈）。

- **低代码扩展能力：**使用低代码开发时，大部分情况下仍离不开代码，因此平台必须能支持在必要时通过少量的代码对应用各层次进行灵活扩展，比如添加自定义组件、修改主题 CSS 样式、定制逻辑流动作等。一些可能的需求场景包括：UI 样式定制、遗留代码复用、专用的加密算法、非标系统集成。

不只是少写代码

回到最初那个直击心灵的小白问题：Low-Code 中的“Low”，到底是啥意思？答案已经显而易见：既不是指抽象程度很低（相反，低代码开发方式的抽象程度要比传统编程语言高一个 level），也不是指代码很 low（也相反，低代码所生成的代码一般都经过精心维护和反复测试，整体质量强于大部分手写代码），而是单纯的“少写代码” —— 只在少数需要的情况下才手写代码，其他大部分时候都能用可视化等非代码方式解决。

再往深一点儿看，低代码不只是少写代码而已：代码写得少，bug 也就越少（正所谓“少做少错”），因此开发环节的两大支柱性工作“赶需求”和“修 bug”就都少了；要测的代码少了，那么测试用例也可以少写不少；除了开发阶段以外，平台还覆盖了后续的应用构建、部署和管理，因此运维操作也更少了（Low-Code → Low-Ops）。

然而，少并不是最终目的：如果单纯只是想达到少的效果，砍需求减人力、降低质量要求也是一样的。低代码背后的哲学，是少即是多（Less is More），或者更准确说是多快好省（Do More with Less） —— 能力更多、上线更快、质量更好，成本还更省，深刻践行了阿里“既要，又要，还要”的价值观精髓。

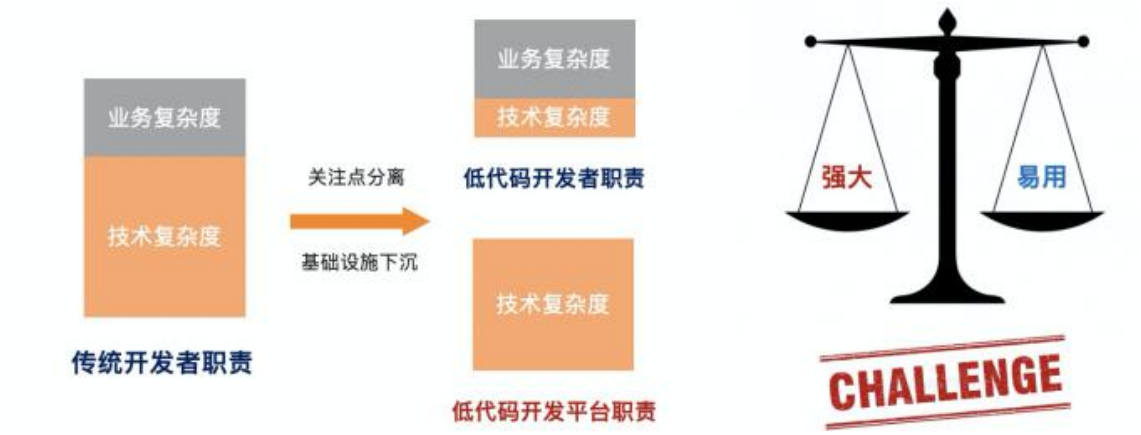
平台的职责与挑战

上面说的是低代码给开发者提供的能力与吸引力，那么作为服务的提供方与应用的承载者，低代码开发平台自身应该承担怎样的职责，其中又会遇到多大的挑战？是否就一定要如阿里云所主张的那样，“把复杂留给自己，把简单留给别人”？虽然这句话听起来很深明大义，但不知道大家有没有想过，为什么我们一定要抱着复杂不放，平白无故给自己找事？就不能直接干掉复杂，也给咱阿里云自己的员工留点简单吗？是工作太容易就体现不出来 KPI 价值了，还是家里的饭菜不如公司的夜宵香？

冥思苦想许久后，我从热力学第一定律中找到了答案：开发一个应用的总复杂度是恒定的，只能转移而不可能凭空消失。要想让开发者做的更少，安心享受简单的快乐，那么平台方就得做的更多，默默承担尽可能多的复杂度。就像一个满身腱子肉的杂技男演员，四平八稳地托举着在高处旋转与跳跃的女搭档；上面的人显得越轻盈越毫不费力，下面的人就得越稳重越用尽全力。当然，不是说上面的女演员就很轻松没压力，只是他们各自的分工不同，所承担的复杂度也不一样。

根据《人月神话》作者 Fred Brooks 的划分，软件开发的复杂度可以划分为本质复杂度（Essential complexity ）和偶然复杂度（Accidental complexity）。前者是解决问题时固有的最小复杂度，跟你用什么样的工具、经验是否丰富、架构好不好等都无关，而后者就是除此之外在实际开发过程中引入的复杂度。通常来说，本质复杂度与业务要解决的特定问题域强相关，因此这里我把它称为更好理解的“业务复杂度”；这部分复杂度不是任何开发方法或工具能解决的，包括低代码。而偶然复杂度一般与开发阶段的技术细节强相关，因此我也相应把它称为“技术复杂度”；而这一部分复杂度，恰好就是低代码所擅长且适合解决的。

为开发者尽可能屏蔽底层技术细节、减少不必要的技术复杂度，并支撑其更好地应对业务复杂度（满足灵活通用的业务场景需求），这是身为一个低代码开发平台所应该尽到的核心职责。



在尽到上述职责的同时，低代码开发平台作为一个面向开发者的产品，还需要致力于为开发者提供简单直观的极致开发体验。这背后除了巨大的工作量，还得能在“强大”和“易用”这两个很难两全其美的矛盾点之间，努力找到一个符合自己产品定位与目标客户需求的平衡点 —— 这也许是设计一个通用低代码开发平台所面临的最大挑战。

三、低代码相关概念对比

纯代码（Pro-Code / Custom-Code）

“纯代码”可能算是我杜撰的一个词，更常见的说法是专业代码（Pro-Code）或定制代码（Custom-Code）；但意思都一样，就是指传统的以代码为中心（Code-Centric）的开发模式。之所以我选择用“纯代码”，是因为如果用“专业代码”会显得似乎低代码就不专业了一样，而用“定制代码”又容易让人误解成低代码无法支持定制的自定义代码。

当然，更准确的称谓我认为是“高代码”（与低代码恰好对应，只是名字太难听，被我嫌弃了...），因为即便是使用传统的代码 IDE，有些开发工作也支持（甚至更适合）以非代码方式完成，比如：iOS 端开发时使用的 SwiftUI 界面设计器、服务端开发数据库应用时使用的 PowerDesigner 建模工具。不过这部分可视化工作在传统开发模式下只是起辅助作用，最后通常也是生成开发者可直接修改的代码；开发者仍然是以代码为中心来开展主要工作。

低代码与纯代码之间的关系，其实跟视频和文章之间很像：

- 低代码就像是现代的“视频”，大部分内容都由直观易理解、表达能力强的图片组成，因此更容易被大众所接受。但与此同时，视频也不是死板得只能有图片，完全可以添加少量文字（如字幕、标注）来弥补图片表达不够精确的问题。BTW，关于“图”和“文字”之间的辩证关系，可以进一步参考《架构制图：工具与方法论》[1]这篇文章中的相关描述。
- 纯代码则更像是传统的“文章”，虽然很久以来都一直是信息传播的唯一媒介，但自从视频技术诞生以及相应软硬件基础设施的普及以来，便逐渐开始被抢走了风头。如今，视频已成为大部分人获取信息的主要渠道（从电视电影到 B 站抖音），而经常读书读文章的人却越来越少。但不可否认的是，文章依然有它存在的意义和受众（不然我也不会费这劲敲这么多字了），即使“市场份额”一直在被挤压，但永远会有它立足的空间。

如果按上面这种类比关系推导，低代码未来也会遵循与视频类似的发展轨迹，超越纯代码成为主流开发模式。Gartner 的预测也表达了相同的观点：到 2024 年，所有应用程序开发活动当中的 65%将通过低代码的方式完成，同时 75%的大型企业将使用至少四种低代码开发工具进行应用开发。

但同样地，就像是视频永远无法取代文章一样，低代码也永远无法彻底取代纯代码开发方式。未来低代码和纯代码方式将以互补的形态长期共存，各自在其所适合的业务场景中发光发热。在后面的“低代码业务场景”章节，会详细列出哪些场景在现阶段更适合用低代码模式开发。

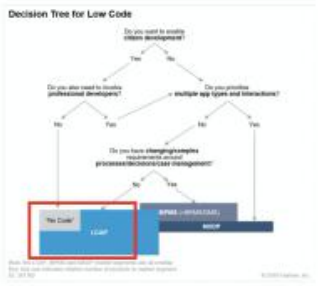
零代码（Zero-Code / No-Code）

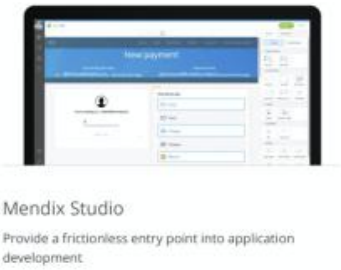
从分类的完备性角度来看，有“纯代码”自然也应该有完全相反的“零代码”（也称为“无代码”）。零代码就是完全不需要写代码的应用开发平台，但这并不代表零代码就比低代码更高级和先进，它只是做了一个更极端的选择而已：彻底拥抱简单的图形可视化，完全消灭复杂的文本代码。选择背后的原因是，零代码开发平台期望能尽可能降低应用开发门槛，让人人都能成为开发者（注意：开发 ≠ 写代码），包括完全不懂代码的业务分析师、用户运营，甚至是产品经理（不懂装懂可不算懂）。

即便是专业开发者，在技术分工越来越精细的趋势下（前端/后端/算法/SRE/数据分析..），也很难招到一个能独立开发和维护整套复杂应用的全栈工程师。但零代码可以改变这一切：无论是 Java 和 JavaScript 傻傻分不清楚的技术小白，还是精通深度学习但没时间学习 Web 开发的算法大牛，都可以通过零代码实现自己的技术梦或全栈梦。“改变世界的 idea 已有，就差一个程序员了”，这句玩笑话或许真的可以成真；哦不，甚至都用不着程序员，有 idea 的人自己就能上。

当然，所有选择都要付出代价，零代码也不例外。完全抛弃代码的代价，就是平台能力与灵活性受限：

- 一方面，可视化编辑器的表达能力远不及图灵完备的通用编程语言，不引入代码根本没法实现灵活的定制与扩展（当然，理论上也可以做成 Scrach/Blockly 那样的图形编程语言，但那样不过是换一种形式在手写代码而已）。
- 另一方面，由于目标受众是非专业开发人员，平台能支持的操作会更趋于“傻瓜化”（e.g. 页面只支持大块业务组件的简单堆叠，不支持细粒度原子组件和灵活的 CSS 布局定义），同时也只会透出相对“亲民化”的模型和概念（e.g. 使用“表格”表示数据，而不是用“数据库”），无法支撑强大专业的底层开发原语和编程理念。



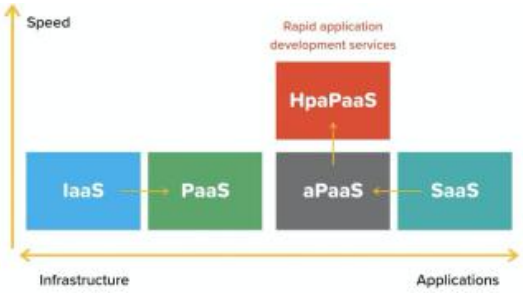


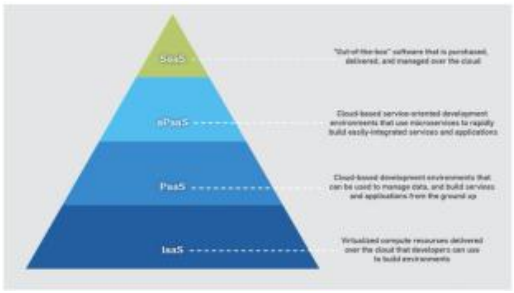
虽然零代码与狭义上的低代码有着上述明显差异，但从广义上来说，零代码可以当作低代码的一个子集。Gartner 在其相关调研报告中，就是将“No Code”划在了范围更广的低代码应用平台“LCAP”（Low-Code Application Platform）中。而当前市面上很多通用的低代码开发平台，也都兼具一定程度的零代码能力；比如低代码领域领头羊 Mendix，既提供了简单易用的零代码 Web IDE - Mendix Studio，也包括一个功能更强大的低代码桌面 IDE - Mendix Studio Pro。

HpaPaaS（高生产力应用 PaaS）

上文提到，“Low-Code”一词是拜 Forrester 所赐。作为同样是国际知名调研机构（a.k.a 造词小能手）的 Gartner，显然不会轻易在这场可能决定低代码领域江湖地位的新概念作词大赛中认输，于是也于 2017 年发明了“HpaPaaS”（High-productivity application Platform as a Service）这个听上去更高大上的缩写词。

按照 Gartner 的定义，HpaPaaS 是一种支持声明式、模型驱动设计和一键部署的平台，提供了云上的快速应用开发（RAD）、部署和运行特性；这显然与低代码的定义如出一辙。但事实证明，名字起得太专业并不见得是好事，“HpaPaas”最终还是败给了起源更早、更接地气也更顺口的“Low-Code”：从 2019 年开始，Gartner 在其相关调研报告中也开始全面采用“Low-Code”一词（如 LCAP），亲手为“HpaPaaS”打上了 @deprecated 印记。





图源：https://blog.kintone.com/business-with-heart/difference-saas-iaas-paas-apaas-hpapaas

值得补充的是，“HpaPaaS“这个词也并非横空出世，而是传承自更早之前 Gartner 提出的“aPaaS”，它俩之间的关系是：HpaPaaS 只是 aPaaS 的一个子类；除了 HpaPaaS 这种通过低代码实现的高生产力应用开发平台以外，aPaaS 还包括面向纯代码的传统应用开发平台（High-control aPaaS，即可控度更高的纯代码开发方式）。

不值得但就想八卦一下的是，“aPaaS”这个词也非凭空捏造，而是与云计算的兴起渊源颇深。相信各位云道中人都已猜到，aPaaS 与 IaaS/PaaS/SaaS 这些云计算远古概念是一脉相承的：aPaaS 介于 PaaS 和 SaaS 之间，相比 PaaS 提供的服务更偏应用，但又不像 SaaS 一样提供现成的软件服务（更详细的说明可参考配图来源文章）。

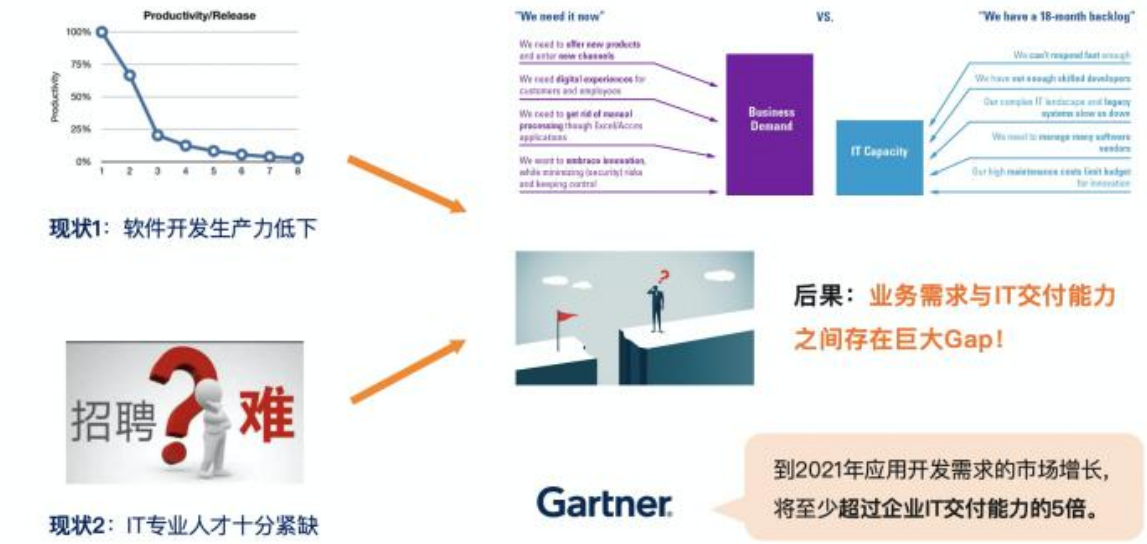
四、为什么需要低代码

低代码是什么可能并没那么重要，毕竟在这个信息爆炸的世界，永远不缺少新奇而又短命的事物。大部分所谓的新技术都只是昙花一现：出现了，被看到了；大部分人“哦”了一声，已阅但表示不感兴趣；小部分人惊叹于它的奇思妙想，激动地点了个赞后，回过头来该用什么还是什么。真正决定新技术是否能转化为新生产力的，永远不是技术本身有多么优秀和华丽，而是它是否真的被需要，即：为什么需要低代码？如果用不同的主语填充上面这个问句（冷知识：这叫做“延迟主语初始化”），可以更全面地看待这个问题：

为什么「市场」需要低代码？

在这个大爷大妈都满嘴“互联网+”和“数字化转型”的时代，企业越来越需要通过应用（App）来改善企业内部的信息流转、强化与客户之间的触点连接。然而，诞生还不太久的 IT 信息时代，

也正面临着与我国社会主义初级阶段类似的供需关系矛盾：落后的软件开发生产力跟不上人民日益增长的业务需求。



Gartner 预测，到 2021 年应用开发需求的市场增长将至少超过企业 IT 交付能力的 5 倍。面对如此巨大的 IT 缺口，如果没有一种革命性的“新生产力”体系，很难想象仅凭现有传统技术体系的发展延续就能彻底解决问题。而低代码技术正是带着这样的使命而降临，期望通过以下几个方面彻底革新应用开发生产力，拯救差一点就要迈入水深火热的 IT 世界：

提效降本 & 质量保障

虽然软件行业一直在高速发展，新的语言、框架和工具层出不穷，但作为从业者我们不得不承认：软件开发仍处于手工作坊阶段，效率低、人力成本高、质量不可控。项目延期交付已成为行业常态，而瓶颈几乎总是开发人员（对机器能解决的问题都不是问题）；优秀的开发人才永远是稀缺资源，还贼贵；软件质量缺陷始终无法收敛，线上故障频发资损不断。

相比而言，传统制造业经过几百年工业革命的发展，大部分早已摆脱了对“人”的强依赖：从原料输入到制品输出，中间是各种精密仪器和自动化流水线的稳定支撑，真正实现生产的标准化和规模化。虽然信息化号称是人类的第三次工业革命，但以软件行业目前的状况，远远还没到达成熟的“工业化”阶段。

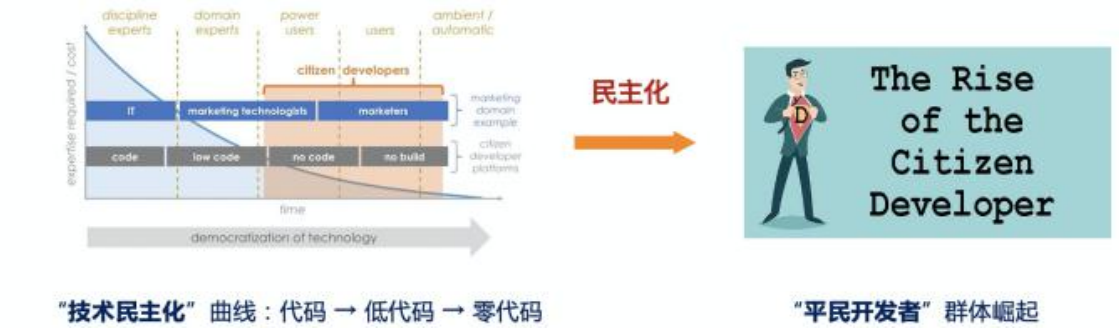
所以，亲爱的程序员朋友，当你与前端联调了一上午接口，又与产品撕逼了一下午需求，再与自己的 bug 抗争了一整晚，好不容易遁入梦乡又被一连串报警短信吵醒时，是否有抬头对着星空憧憬过：“I have a dream... that one day，软件开发也能像工业制品一样，批量流水化生产，稳定高效没烦恼。” 事到如今，不管你有没有意识到，这个憧憬正在慢慢变成现实。

“手工作坊”式饼干制作　　“工厂流水线”式饼干制作（白色恋人）

是的，低代码正在将应用软件开发过程工业化：每个低代码开发平台都是一个技术密集型的应用工厂，所有项目相关人员都在同一条产线内紧密协作。开发主力不再是熟知 for 循环一百种写法的技术 Geek，而是一群心怀想法业务 sense 十足的应用 Maker。借助应用工厂中各种成熟的基础设施、现成的标准零件、自动化的装配流水线，开发者只需要专注于最核心的业务价值即可。即便是碰到非标需求，也可以随时自己动手，用最灵活的手工定制（代码）方式来解决各种边角问题。

扩大应用开发劳动力

通过让大部分开发工作可以仅通过简单的拖拽与配置完成，低代码（包括零代码）显著降低了使用者门槛，让企业能够充分利用前面所提到的平民开发者资源。部分纯零代码需求场景下，低代码还能让业务人员实现自助式（self-service）应用交付，既解决了传统 IT 交付模式下的任务堆积（backlog）问题，避免稀缺的专业开发资源被大量简单、重复性的应用开发需求所侵占，也能让业务人员真正按自己的想法去实现应用，摆脱交由他人开发时不可避免的桎梏。



"技术民主化"曲线：代码 → 低代码 → 零代码　　"平民开发者"群体崛起

至此，应用开发能力不再是少数专业开发者的专利和特权，且今后所需要的技能门槛与拥有成本也会越来越低，真正实现所谓的“技术民主化”（democratization of technology）。

加强开发过程的沟通协作

多方调查结果显示，软件项目失败的最主要原因之一就是缺乏沟通（poor communication）。传统开发模式下，业务、产品、设计、开发、测试与运维人员各司其职，且各有一套领域内的工具和语言，长久以来很容易形成一个个“竖井”（silos），让跨职能的沟通变得困难而低效。这也是为什么当前热门的敏捷开发和 DevOps 都在强调沟通（前者是协同 Biz 与 Dev，而后者是协同 Dev 和 Ops），而经典的 DDD 领域驱动设计也主张通过“统一语言”来减少业务与技术人员之间的沟通不一致。

“竖井”效应：Biz | Dev | Ops　　Biz-Dev-Ops：终极敏捷协作

有了低代码后，这一状况将得到根本改善：上述各角色都可以在同一个低代码开发平台上紧密协作（甚至可以是同一个人），这种全新的协作模式不仅打破了职能竖井，还能通过统一的可视化语言和单一的应用表示（页面/数据/逻辑），轻松对齐项目各方对应用形态和项目进度的理解，实现更终极的敏捷开发模式，以及在传统 DevOps 基础之上更进一步的 BizDevOps[2]。

统一开发平台下的聚合效应

低代码尝试将所有与应用开发相关活动都收敛到同一个平台（one platform）上后，将会产生更多方面的聚合效应与规模收益：

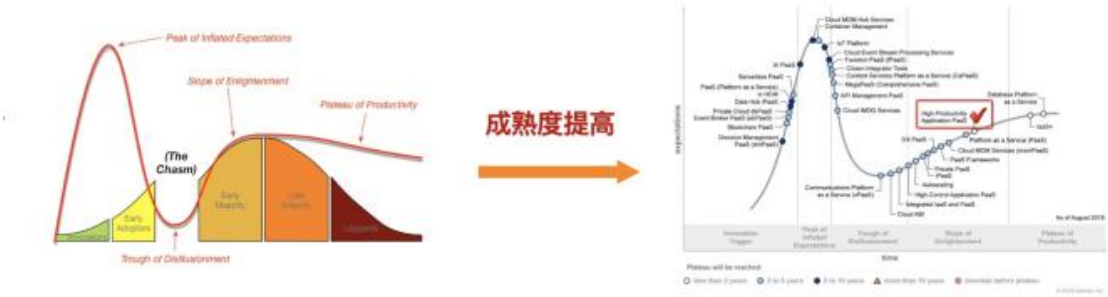
- **人员聚合：**除了上一点所提到的各职能角色紧密协作以外，人员聚合到统一的低代码开发平台进行作业后，还能促进整个项目流程的标准化、规范化和统一化。
- **应用聚合：**一方面，新应用的架构设计、资产复用、相互调用变得更容易；另一方面，各应用的数据都天然互通，同时平台外数据也能通过集成能力进行打通，彻底消除企业的数据孤岛问题。
- **生态聚合：**当低代码开发平台聚合了足够多的开发者和应用后，将形成一个巨大的、连接一切、有无限想象力的生态体系，彻底放飞低代码的价值。

为什么「这个时代」才需要低代码？

如果你了解过市面上各种低代码产品，不难发现其实这个领域的许多玩家在低代码概念诞生之前就已经存在了，比如：低代码领域的另一个巨头 OutSystems，早在 2001 年就已经创立；而去年也被 Forrester 评为低代码行业 leader 之一的 FileMaker，更是诞生于遥远的 1985 年（正好 35 岁，似乎在疯狂暗示什么）。那么，如果低代码像前面说的那么好，为什么以前没有火起来呢？从技术和业务两个角度看，可以归纳为以下原因：

技术成熟度不足

低代码底层的各项核心技术（可视化、模型驱动、RAD、BPMS...）都已经有漫长的发展历史，看上去似乎只是新瓶装旧酒。然而理智的人都知道，任何技术都会遵循所谓的“技术成熟度曲线”（The Hype Cycle），不可能刚一诞生就跳过发育直接秀翻全场，被大规模采纳和投入生产。以模型驱动技术为例，虽然十几年前就已经有体系化的理论研究（e.g. MDA）和配套工具（e.g. EMF），但在当时的技术背景下，由于能力不完备、过于理想化、技术门槛高等原因，一直没能在工业界走向主流。



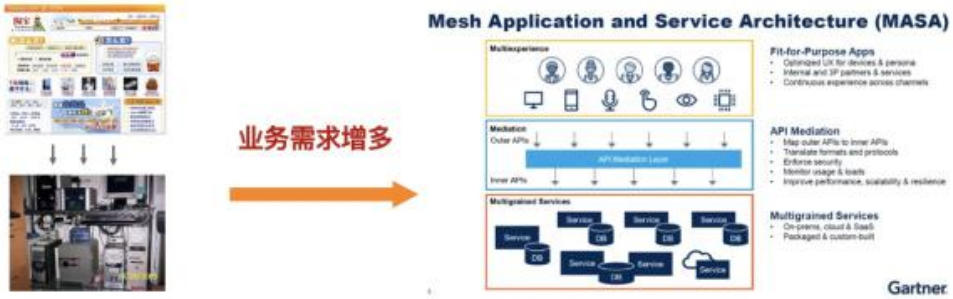
技术采纳周期：早期技术难以跨越鸿沟　　技术成熟度曲线：低代码已经快爬到生产力高地

而如今这个时代，支撑低代码的那些“老”技术都已经过长时间的发展酝酿与市场检验，而另一些完美互补的“新”技术（e.g. 云原生、响应式 Web）也在飞速发展和走向成熟，是时候通过“低代码”这个新酒瓶重新包装上市，为亟需新生产力的传统 IT 市场带来一场真香之旅了。

业务收益不明显

即使十几年前的低代码技术已经足够成熟，也一定不会在当年的应用开发市场上产生现在这样的影响力。为什么？因为技术都是为业务服务的，而当时的应用开发业务需求可比现在简单多了：没有如今的多渠道（Multi-channel）、多样化体验（Multi-experience）和各种集成与定制需求，也不会奢求如今已成为企业级应用标配的弹性、分布式和高可用，更是缺乏快速变化的 IT 业务场景来推动持续集成与快速交付。

虽然低代码可以完美解决上述所有问题（e.g. 多端应用生成、云原生架构、API 集成能力），但放在当年的市场和业务背景下，加上前面所说的技术不成熟度，整体的投入产出比会很低，不足以让企业大面积采纳低代码解决方案。



过去：应用架构简单粗暴，但够用　　现在：Gartner MASA 应用架构（多样化体验/多粒度服务）

而如今这个时代，企业都快被新技术带来的能力和收益“惯坏了”，动不动就是：我想做一个送菜应用。用户端？安卓、iOS、H5、小程序都来一套。运营端？一般都在电脑上看，但记得手机上也得适配啊。服务端？上云，必须的。哦，我听技术合伙人说现在流行多云架构，也给我整一套哈。运维还要钱？啥是运维？应用有了不就能用了嘛，运维还要花我钱？你当投资者给我的钱是大风刮来的啊！

如果用传统的开发模式，这么全套下来的工时与报价，可能早就吓跑了这群跟产品经理一样天真可爱的人；但现代化的低代码技术，可以圆了上面这位创业者的卖菜梦，用白菜一般的价格，实现白粉一样的价值。当年的程维如果能用上现在的低代码，第一版的滴滴 App 也就不至于被外包做得乌烟瘴气直接报废了（至少能多扛一阵子...）。

为什么「专业开发者」也需要低代码？

虽然零代码确实是设计给非专业开发者用的，但其所能支撑的业务场景确实有限，无法真正革新传统开发模式，替代那些仍需专业开发者参与的复杂业务场景。而狭义上的低代码却有潜力做到这一点，因为它天生就是为专业开发者而量身定制的。Gartner 最近的一项调研报告显示，“66%的低代码开发平台用户都是企业 IT 部门的专业开发者”。这充分说明了，专业开发者比平民开发者更需要低代码。

屏幕前一批穿格子衬衫的同学要发问了：“低代码都不怎么写代码了，怎么能算是为我们程序员服务呢？”。虽然程序员讨厌重复自己，但重要的事情还是得多说一遍：开发 ≠ 写代码。1 万年前蹲在洞穴里的原始人，在用小石子画远古图腾；100 年前坐在书桌前的徐志摩，在用钢笔给林徽因写情书；而今天趴在屏幕前的很多人，相信都已经开始用上手写板或 iPad 涂涂写写了。千百年来，人类使用的工具一直在演进，但所从事活动的本质并没有多大改变。无论是用小石子还是小鼠标，写作绘画的本质都是创造与表达，最终作品的好坏并不取决于当时你手中拿着什么；同样地，应用开发的本质是想法和逻辑，最终价值的高低也不取决你实现时是用的纯代码还是低代码。

而相比纯代码而言，低代码极有可能成为更好的下一代生产力工具：

减少不必要的工作量

可视化拖拽与参数配置的极简开发模式，结合模型驱动的代码自动生成机制，可以消灭绝大部分繁琐和重复的 boilerplate 代码；一站式的部署和运维管理平台，无需自己搭建 CI/CD 流水线、申请环境资源、配置监控报警；一次搭建同时生成、构建和发布多端应用，免去人工同步维护多个功能重复的端应用；开箱即用的组件库、模板库、主题库、连接器等，让最大化软件复用成为可能。总而言之，低代码能够让专业开发者更专注于创新性、有价值、有区分度的工作，而不是把宝贵开发时间都耗费在上面那些不必要的非业务核心工作上。

强大的平台能力支撑

虽然上面列的技术支撑性工作并不直接产生业务价值，但却会直接影响业务的性能、成本、稳定性、安全性、可持续发展能力等。有远见的企业，绝不允许牺牲这些重要指标，来换取短暂的业务加速。低代码开发平台深知这一点，因此在简化和屏蔽底层技术细节的同时，也会尽可能把自己所 cover 的部分做到最好（至少能和纯代码开发方式一样好），包括但不限于：

- 现代化的技术架构和实现：现代化的低代码开发平台，在支撑用户应用时所选择的技术架构与实现方案，也会是现代化且符合业界最佳实践的，例如，前端基于主流的 HTML5/CSS3 标准和 React 框架，后端基于成熟的 Java 语言、SpringBoot 框架和 MySQL 数据库，部署环境基于云原生的 Docker 镜像、CI/CD 流水线、K8s 集群和 Service Mesh 技术（相关知识可参考《正确入门 Service Mesh：起源、发展和现状》）。
- 零成本的技术升级和维护：低代码的高维抽象开发方式，让应用的核心业务逻辑与底层技术细节彻底解耦。开发者在大部分情况下都不需要关心底层技术选型，同时也无需亲自跟进这些技术的版本升级与漏洞修复，免费享受与时俱进的技术红利和应用安全性提升。即便遇到某些底层技术或工具需要进行彻底更换（比如不再维护的开源项目），开发者也完全不必感知；技术迁移再费劲再难搞，平台自己努力就行，对开发者来说只要服务一直在线，岁月就依然静好；事后可能还会惊喜地发现，应用访问突然就变得更快了，仿佛冥冥中自有天助，感激上苍和低代码。

一体化生态能力复用

复用（Reuse）是提升软件开发效率和工程质量的最有效途径。传统的代码开发模式下，开发者可以通过提取公共类/函数、引用共享库、调用外部 API 服务、沉淀代码片段和模板等方式实现复用。在低代码的世界里，平台也可以提供对应的多层次多粒度复用手段，比如页面组件库、逻辑函数库、应用模板库等。

但更重要的是，低代码平台还可以充分发挥其一体化的生态优势，提供强大易用的可复用能力（资产）的发现、集成与共享体系：以页面组件为例，你可以直接用系统组件，也可以在平台自带的组件市场上搜索和引用更合适的组件，还可以自己用代码开发一个自定义组件并发布到市场中。平台的生态体系越大，积累的可复用能力就越多，应用的开发成本也会越低。

相比而言，虽然传统代码世界整体生态更庞大和深厚，但由于各类技术不互通、缺乏统一平台与市场、代码集成成本高等原因，一直以来都没有形成有类似规模潜力的生态能力复用体系，导致重复造轮子和低水平重复建设的现象司空见惯，还美名为“新基建”。

说到这里，另一批裹着冲锋衣头顶锃亮的同学也忍不住了：“万一低代码真的发展起来了，是不是就不需要那么多程序员了啊？上有老下有小的，同是码农身，相煎何太急！”。低代码虽然是一场应用开发生产力革命，但并不会革掉程序员的饭碗。它去掉的只是难懂的编程语法、繁琐的技术细节和一切可自动化的重复性工作，并没有也无法去掉应用开发最核心的东西：严谨的业务逻辑、巧妙的算法设计、良好的工程风格等。对于真正的程序员，即使剥去他一层又一层的编程语言和工具熟练度技能外壳，最终剩下的仍然是一个有价值的硬核开发者。

当然，如果你坚持要用纯粹的写代码方式来改变世界，也不至于失业。要么，你可以选择那些低代码暂时不太适用的领域，比如底层系统驱动、3D 游戏引擎、火箭发射程序；或者，你也可以选择去写低代码中那一部分不可或缺的自定义代码扩展，为平民开发者提供高质量的积木。最后，你也完全可以选择为低代码平台本身的底层代码添砖加瓦，比如加入阿里云云原生应用研发平台 EMAS 团队（〃'▽'〃），与作者一起共建下一代云原生低代码开发平台“Mobi”，内推直达邮箱：pengqun.pq@alibaba-inc.com。

为什么「我不」需要低代码

即使所有人都认同上述“为什么要用低代码”的理由，但仍不时会有试水者跳出来，给大家细数“为什么我不需要低代码”。实践出真知没错，而且大部分质疑背后也都有一定道理；但在我看来，更多的可能是主观或无意识的偏见。这里我列了一些对低代码的常见质疑和我个人的看法，期望能帮助大家看到一个更全面和客观的低代码。

质疑 1：低代码平台不好使

> “试用过一些所谓的低代码开发平台，要么能力很弱，要么体验太差，只能开发点玩具应用。”

作为调研过国内外多款低代码产品的深度体验用户，我的观点是：不能以偏概全。低代码市场在国内正处于爆发初期，所以许多与低代码只沾一点边的产品也都在蹭热点；但它们并不能代表低代码目前的业界水平和发展方向。市面上真正成熟的企业级低代码开发平台，完全有能力以高效的开发方式满足大部分复杂场景的功能需求，以及企业级应用所需要的安全、性能、可伸缩等非功能需求，这一点在国外市场已得到充分验证（不然也不会这么被寄予厚望）。

当然，国内市场尚处于鱼龙混杂的混战阶段，遇到真龙的概率很低，但碰上金鱼鲤鱼甚至木头假鱼都在所难免。相信随着时间推移，真正有实力和口碑的产品都能脱颖而出，为大家展现低代码该有的样子。

质疑 2：低代低开发不可控

> “平台上的各种可视化组件、逻辑动作和部署环境都是黑盒，如果内部出问题无法排查和解决。”

作为同样不搞清楚底层原理不舒服斯基的程序员，我更愿意相信：问题只是暂时的。虽然这确实是目前使用低代码平台时绕不开的一个痛点，但并不属于低代码技术本身的固有缺陷。计算机领域有一句至理名言：任何问题都可以通过增加一个间接的中间层来解决。低代码的思路亦是如此：与当年的操作系统和现在的云平台一样，都是想通过建立一个黑盒化的中间层抽象来降低开发者的工作量与心智负担。

当然，所有额外增加的中间层都不是完全免费的，低代码也不例外。作为一个尚未成熟稳定的新的中间层，低代码必然会出现各种让使用者束手无措的问题，就跟当年的操作系统内核 bug、如今的云主机 I/O hang 一样。但历史规律也告诉我们，所有伟大的技术最终都会走向成熟；只要低代码领域一直健康发展，问题总会越来越少，最终降到一个绝大部分人感知不到的范围内。过去萦绕在 Windows 用户心中挥之不去的“蓝屏”问题，对如今的新用户来说早已不知为何物；今天低代码开发者所遇到的种种“蓝瘦”问题，未来也终将成为被遗忘的历史（谁还没段黑历史呢）。

质疑 3：低代码应用难维护

> “应用一旦复杂起来，各种复杂逻辑流穿插着自定义代码，看不懂也改不动，还不如全用代码呢。”

作为对软件可维护性深有感触的无脑级布道者（见《救火必备！问题排查与系统优化手册》），我不得不说：用低代码开发，也要讲基本法。一般来说，无论是使用低代码开发还是纯代码开发，造成应用可维护性低的根本原因往往不在于开发工具，而是开发者自身没有去遵循一些软件开发的普适原则，比如工程规范性、命名可读性、DRY/KISS/SOLID 原则等。

好的低代码平台绝不会阻碍开发者去改善应用的可维护性；恰恰相反，还会尽可能提供引导和帮助。以 Mendix 为例，除了支持基本的模型分析与重构（e.g. 无用模型、对象重命名、子逻辑流提取）以外，甚至还提供了基于 ISO/IEC 25010 标准的应用质量监控（AQM）能力。另一方面，让应用变得难以维护的一个客观原因也是应用本身过于复杂，而低代码作为高度抽象和自动化的开发模式，在降低应用复杂度方面是专业的。

综合来看，低代码虽然不是能解决一切问题的银弹，但更不是会带来更多问题的炸弹：在提高应用可维护性方面的上限，一定比传统开发模式更高；但决定应用可维护性下限的，依然还是开发者自己。

五、低代码行业发展

回应质疑的最好方式，就是做好你自己，用实际的表现说话。对于一个行业而言，判断它当前

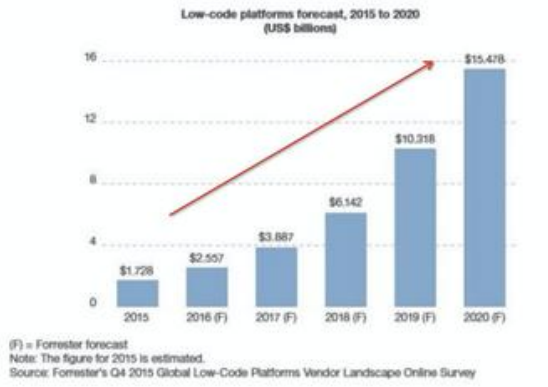
的表现是否够好，或者未来是否有潜力做到更好，可以从以下这三个方面进行衡量：市场规模（蛋糕够不够大）、适用场景（是否可落地）、竞品状况（有没有被验证过）。

市场规模

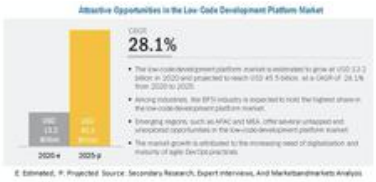
> "Talk is cheap，show me the code money."
> —— Linus Starcraft

文章可以忽悠，但市场不会说谎：

- Forrester 在 2015 年曾预测过，低代码的市场将从 2015 年的 17 亿美元增长至 2020 年的 150 亿美元。
- Marketsandmarkets 在今年四月份的分析报告中预测，低代码的市场将从 2020 年的 130 亿美元（估算值，可以看出来与 Forrester 当年的预测是接近的）增长到 2025 年的 450 亿美元（年复合增长率：28.1%）。
- PS Inteligence 在 2018 年的分析报告中预测，全球的低代码开发平台市场中，亚太地区将在今后五年（2019-2024 年）中保持最高的增长速度。



Forrester预测：市场规模持续增长，2020年达到150亿美元



Marketsandmarkets预测：2025年将达到455亿美元

PS Intelligence预测：亚太地区的市场增长将会最快

总结一下就是两点：

- 低代码的市场规模足够大，且一直都在高速增长。

- 作为亚太地区的经济大国与 IT 强国，中国的低代码市场将会引来一个爆发期，未来几年内的增速都会超过全球平均水平。

适用场景

理论上来说，低代码是完全对标传统纯代码的通用开发模式，应该有能力支撑所有可能的业务场景。但理论也只是理论，低代码一统江湖的梦想尚未照进现实，也不可能完全取代现实。前文中提到过，低代码与纯代码方式是互补关系，未来也将长期共存，各自在其所适合的业务场景中发光发热。同时还需要指出的是，当前阶段的低代码技术、产品和市场都尚未完全成熟，因此部分本来可能很适合用低代码来开发的场景，目前也只能先用纯代码来替代。

Gartner 在 2019 年的低代码调研报告中，曾经绘制过一张用来阐述低代码适用场景的“应用金字塔”：

不同类型应用与低代码开发模式的契合度

- **应用级别划分：**从下往上，分别为工作组级（Workgroup Class）、部门级（Departmental Class）、企业级（Enterprise Class）、可扩展需求极强的企业级（Extreme-Scale Enterprise Class）。容易看出来，它主要的划分维度就是应用所面向的用户基数（基数越大，可扩展需求也越高）。
- **任务关键性：**从下往上，各级别应用的任务关键性（Mission Criticality）逐级递增。例如一个只在工作组内使用的后台管理应用，一般都不会涉及到影响整个企业的关键任务。脱离企业这个视角来看，整个软件产业中也有很多通用的任务关键型应用，比如：实时操作系统、航空调度系统、银行对账系统。

- **实现复杂度：**从下往上，各级别应用的复杂度（Complexity）也逐级递增。例如最上层的企业级应用，除了功能覆盖面大导致业务复杂以外，往往还需要满足更多苛刻的非功能需求，包括但不限于：用户体验、性能、可靠性、安全性、可伸缩性、可维护性、兼容性。其他一些复杂软件的案例包括：3D 游戏界面（交互复杂）极其底层的游戏引擎（逻辑复杂）、超大型 CRM 系统（一方面是实现很复杂，另一方面，这种成熟软件的标准化程度较高，大部分情况下可以直接用现成的 SaaS 软件）。
- **应用需求量：**从上往下，各级别应用的需求体量（Volume）逐级递增，呈现一个金字塔形状。这个特征可以用万能的 2/8 原则来理解：20%的“全民”应用，由于需求的通用性和普适性，可以覆盖至少 80%的用户群体（例如企业大部分人都要用的考勤系统）；而剩下那 80%的“小众”应用，由于需求的定制化和特殊性（例如蚂蚁的期权系统...），就只能覆盖各自小圈子里那 20%的用户了。
- **与低代码的契合关系：**从上往下，各级别应用与低代码越来越契合（Relevant）。也就是说：越简单的应用，越契合低代码；越不太关键的任务，也越契合低代码。同时，由于契合低代码的应用更偏金字塔底层，而这些应用的需求量都更大，所以可以得出如下判断：低代码能够适用于大部分业务场景（而且这个比例会一直上升，逐步往金字塔的更上层应用逼近），例如：B2E 类应用（表单、审批流、ERP 系统）、B2B 类应用（企业商城、工业控制台）、B2C 类应用（企业展示、营销页、店铺装修）。

竞品概况

低代码虽然是一个新兴概念，但这个行业本身并不算很新（前文也有提到），这些年以来早就积累了不少资深的荣耀王者。同时，低代码作为一个朝阳产业和资本热点，近几年也不断有更多的新玩家在加入这个刺激战场。

Gartner魔力象限：企业级低代码应用平台（2020年）

Forrester技术波谱：低代码开发平台（2019年）

上图分别是 Gartner 给出的低代码平台魔力象限和 Forrester 给出的低代码平台技术波谱。从图中可以看到：

- OutSystems 和 Mendix 一马当先，是公认的低代码领域头牌。这两家都是很纯粹的通用低代码开发平台，且都经过了长时间的发展和积累：OutSystems 成立于 2001 年，员工人数 1000+，年营收超过 1 亿美元；2018 年 6 月获得了 KKR 和高盛的 3.6 亿美元融资，目前估值超过 10 亿美元；Mendix 成立于 2005 年，员工人数 500+，年营收超过 2300 万美元（18 年数据），2018 年 8 月被西门子以 7.3 亿美元收购。
- Salesforce 和 Microsoft 紧随其后，都处于行业领先者地位。但这两家的公司性质和发展路径都很不一样：Salesforce 是以 SaaS 起家，公司规模就不用多说了，反正就是 SaaS 届的巨无霸。这类 SaaS 厂商做低代码的动力，是为了解决客户对成品 SaaS 软件的定制诉求。M$更不用多介绍，只说下他们做低代码的天然优势：一方面，作为办公软件航空母舰，低代码可以帮助他们的客户实现从 Excel 表单到定制 App 的能力与体验升级；另一方面，作为云计算三巨头之一，低代码可以帮助他们连接内部的云计算生态体系，为开发者提供一个统一和易用的上云界面。
- 国外市场已经得到充分验证，但国内市场还刚刚兴起，还没有一家能够赢得上述调研机构的芳心，挤进上面这两张方图。国内目前的一些竞品和融资情况包括：2018 年 5 月，搭搭云完成 A 轮的千万级融资；2018 年 9 月，宜创科技得到清源创投的战略融资；2018 年 12 月，轻流完成千万级 Pre-A 融资；2019 年 8 月，数式科技得到盈动资本的数千万人民币天使轮融资；2019 年 8 月，ClickPaas 获得晨兴资本数百万美元的 A 轮融资；2019 年，奥哲分别获得阿里

5 千万的 A+轮融和高榕资本上亿元的 B 轮融资。（注：竞品数据来源于我们组 PD 的辛勤整理；为此我决定这篇文章剩下内容再也不黑 PD 了；下篇再说。）

六、结语

本文总结了低代码领域的基本概念、核心价值与行业现状。虽然这些内容都比较基础和偏理论，但我始终认为，深刻理解一个系统的前提，正是这些务虚的东西 —— 技术架构只会告诉你这个系统是怎么实现的（How），无法准确表述它到底能用来做什么（What），以及为什么要做这样一个东西（Why）；而后面这两个问题的答案，才是后续系统所有设计与演进的根因和驱动力。

虽然程序员真的不喜欢重复自己，但冗余也是一种必要的容错手段，好东西真的不容错过：欢迎各位技术同路人加入阿里云云原生应用研发平台 EMAS 团队，，我们专注于广泛的云原生技术（Backend as a Service、Serverless、DevOps、低代码平台等），致力于为企业、开发者提供一站式的应用研发管理服务，内推直达邮箱：pengqun.pq@alibaba-inc.com。

7.EMAS 旗下 serverless 小程序开发

基于小程序云 Serverless 开发微信小程序

简介：本文主要以使用小程序云 Serverless 服务开发一个记事本微信小程序为例介绍如何使用小程序云 Serverless 开发微信小程序。记事本小程序的开发涉及到云函数调用、云数据库存储、图片存储等功能，较好地展示了小程序云 Serverless 服务在实际研发场景中如何帮助开发者提升研发效率。

本文主要以使用小程序云 Serverless 服务开发一个记事本微信小程序为例介绍如何使用小程序云 Serverless 开发微信小程序。记事本小程序的开发涉及到云函数调用、云数据库存储、图片存储等功能，较好地展示了小程序云 Serverless 服务在实际研发场景中如何帮助开发者提升研发效率。

一、准备工作

在开始前，确保已经完成以下准备工作：

1、注册微信开放平台账号，并创建微信小程序，获得微信小程序 AppID；
2、下载并安装微信开发者工具；
3、已安装 nodejs 开发环境。

二、操作步骤

2.1 步骤一 开通小程序云服务

阿里云账号首次使用小程序云服务时，需要开通小程序云服务：

1、登录小程序云控制台。
2、在产品开通页面，勾选服务协议并单击立即开通。

2.2 步骤二 创建服务空间

开发小程序用到的小程序云 Serverless 相关资源，如云函数、数据库、文件存储，都以服务空间的维度进行管理。每个服务空间都有一个全局唯一的 Space ID，小程序在使用云资源时，通过这个 Space ID 关联到具体的云资源。

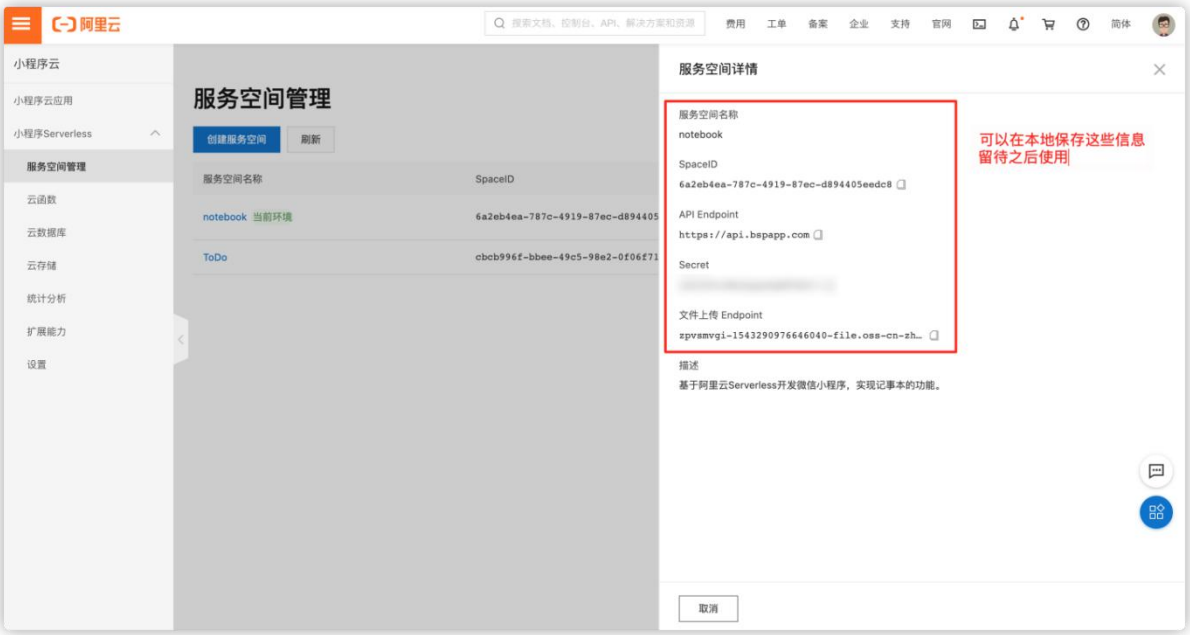
我们通过以下步骤创建服务空间并获得相关配置：

1、登录小程序云控制台。

2、在左侧导航栏选择**小程序 Serverless > 服务空间管理**。

3、单击**创建服务空间**，填写**服务空间名称**和**描述**，单击**确定**。

4、创建成功后，单击服务空间右侧的**详情**可查看 Space ID、Secret 和 API Endpoint、文件上传 Endpoint 等信息，并将这些信息保存下来待用。



2.3 步骤三 创建微信小程序并配置域名白名单

微信会对小程序需要通过网络访问的服务提供方进行白名单限制，我们还需要配置小程序服务器域名白名单：

1、登录微信小程序公众号平台。

2、在左侧导航栏选择**开发**，单击**开发设置**页签。

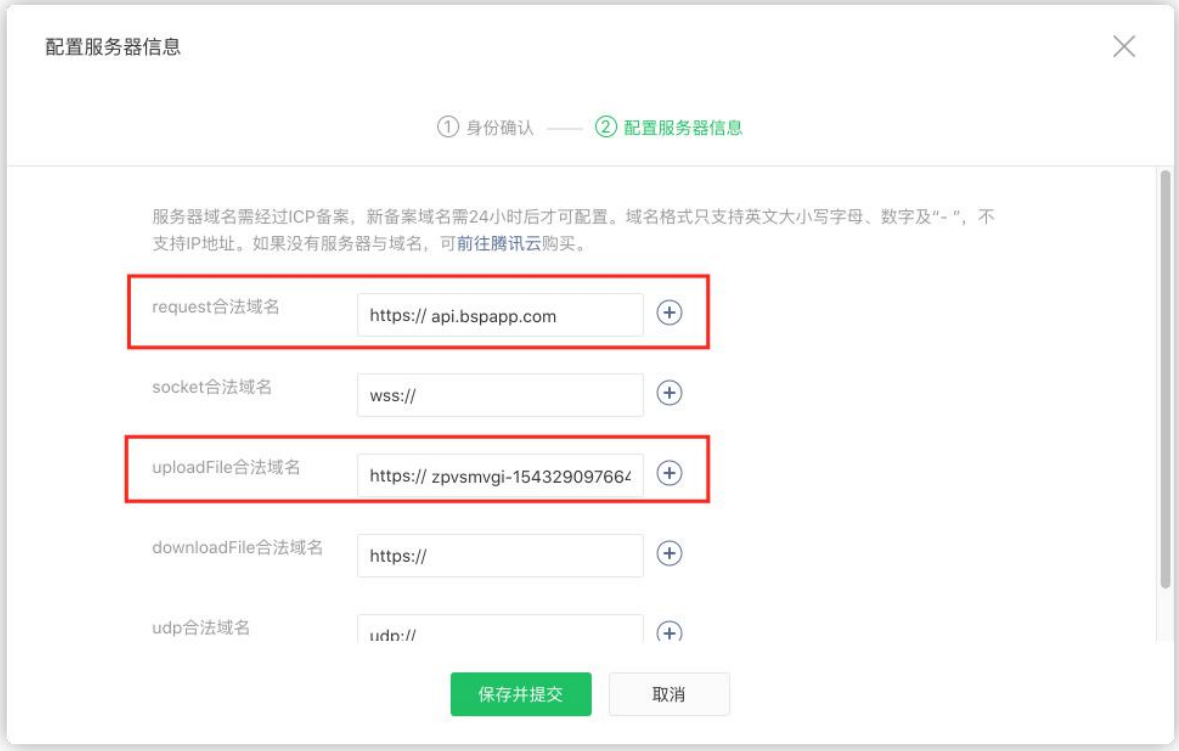
3、在**服务器域名**区域，单击**修改**，根据提示重新扫码进行身份认证。

4、根据**步骤二**中保存的服务空间信息配置 **request 合法域名**（api.bspapp.com）和 **uploadFile 合法域名**，确认无误后保存并保存。

配置说明：

request 合法域名：API Endpoint。

uploadFile 合法域名：https://文件上传 Endpoint。

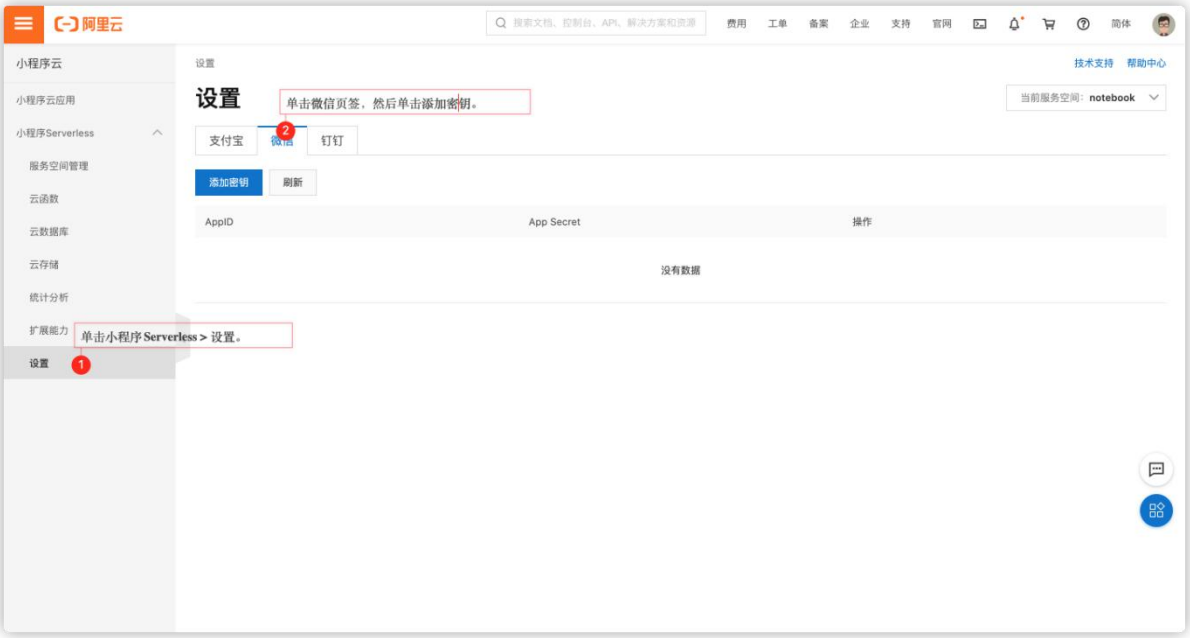


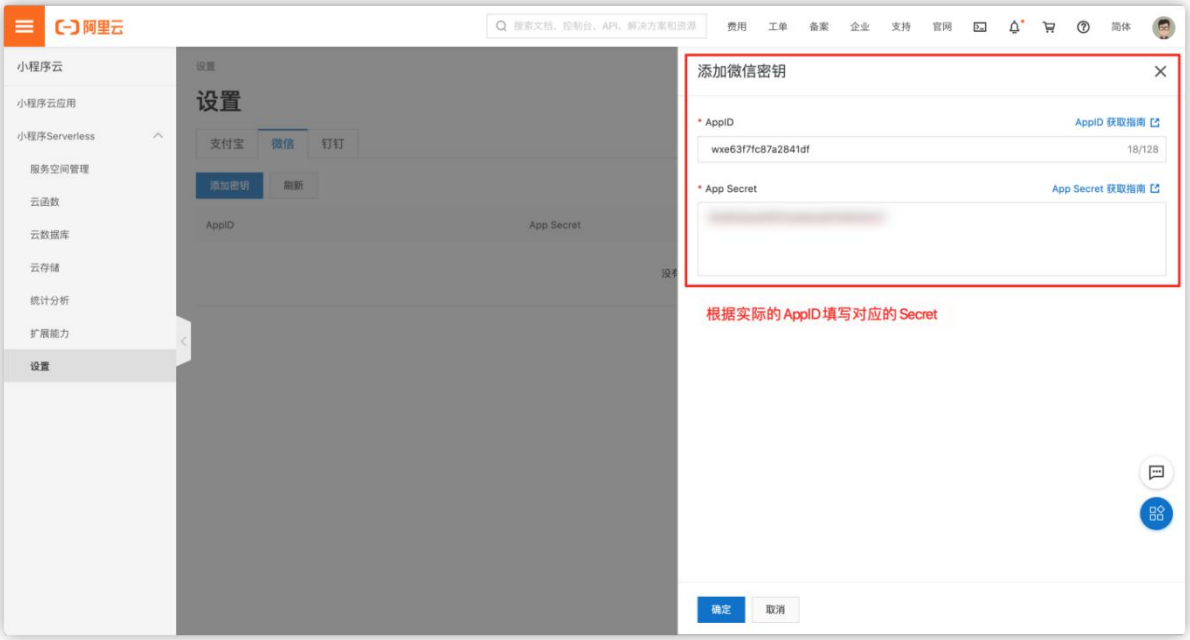
2.4 步骤四 在小程序云控制台添加微信小程序凭证

我们还需要在小程序云控制台添加微信小程序凭证：

1、在小程序云控制台的左侧导航栏，选择**小程序 Serverless > 设置**。

2、选择微信**页签**，单击**添加密钥**，输入 AppID 和 App Secret，单击**确定**。（微信小程序 AppID 和 App Secret 获取方式请参考链接，请妥善保存小程序的 App Secret）





2.5 步骤五 初始化小程序工程

1、 创建小程序工程

在微信开发者工具中创建新项目，填写 AppID，并勾选“**不使用云服务**”。

2、IDE 配置

在微信小程序 IDE 的右上角，单击**详情**，勾选**增强编译**。

在 1.02.1904282 以及之后版本的微信开发者工具中，增加了**增强编译**的选项来增强 ES6 转 ES5 的能力，启用后会使用新的编译逻辑以及提供额外的选项供开发者使用。

3、引入小程序云 Serverless 客户端 SDK

在使用小程序云 Serverless 服务前，我们需要在小程序中安装小程序云 Serverless 客户端 SDK 并初始化。小程序 Serverless 客户端 SDK 的更多信息请参见 安装客户端 SDK2.3 版本。

对于微信小程序端，我们需要直接引入 SDK 源文件。将下载后得到的 mpserverless.js 保存在此项目文件的文件夹中，建议单独保存。如本项目中存储路径: /sdk/mpserverless.js 。

4、进行 SDK 初始化

打开工程根目录的 app.js 文件，在构造 App 对象之前，添加如下代码：

```
const MPServerless = require('/sdk/mpserverless.js');    // 此路径即为上述引入 mpserverless.js 文件的保存路径 const mpServerless = new MPServerless({
  uploadFile: wx.uploadFile,
  request: wx.request,
  getAuthCode: wx.login,
  getFileInfo: wx.getFileInfo,
  getImageInfo: wx.getImageInfo,
}, {
  appId: '', // 小程序应用标识
  spaceId: '', // 服务空间标识
  clientSecret: '', // 服务空间 secret key
  endpoint: '', // 服务空间地址，从小程序 serverless 控制台处获得
});
```

并根据我们之前得到的 appId、spaceId、clientSecret、endpoint 参数填充单引号里的内容。然后，我们通过如下方式进行登录授权并获得用户身份信息：

```
const { mpServerless } = getApp();await mpServerless.user.authorize({
  authProvider: 'wechat_openapi',
});
const getUserInfoRes = await mpServerless.user.getInfo();if (getUserInfoRes.success) {
  // 获得用户
  const userInfo = getUserInfoRes.result.user;
}
```

获取到的 userInfo 中有 userId 字段，可以唯一标识用户身份。

2.6 步骤六 服务空间配置

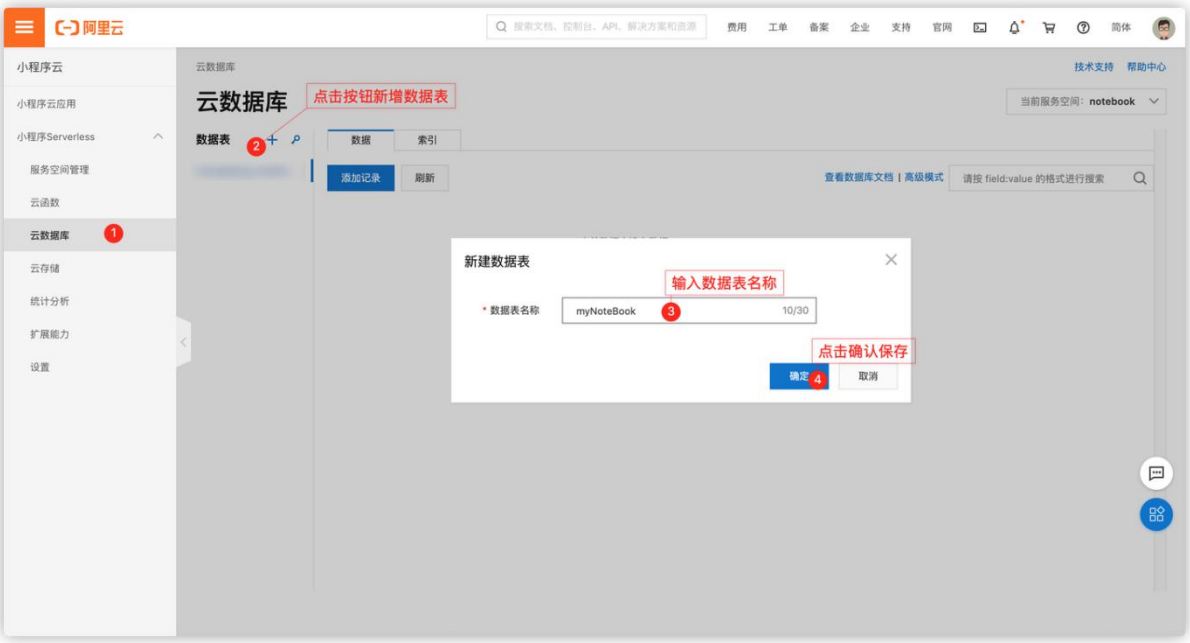
在初始化小程序云 Serverless 完成后，我们就可以在小程序中使用 Serverless 相关服务了。我们先登录小程序云 Serverless 控制台，进行数据库、云存储、云函数的配置。

1、创建数据表并配置权限

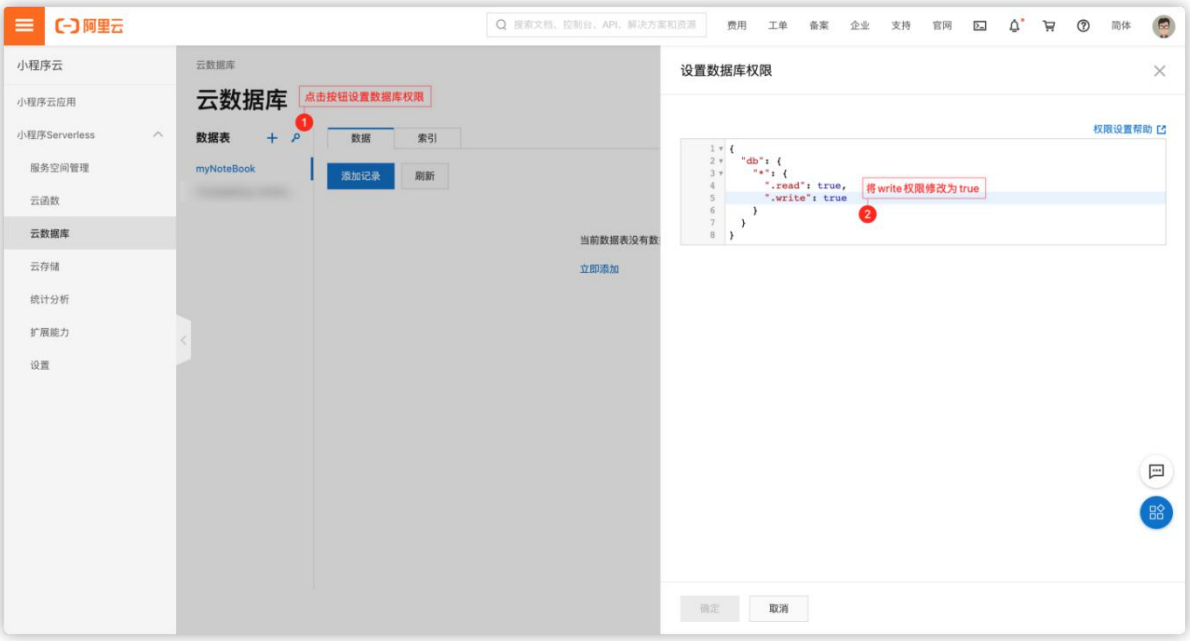
小程序 Serverless 服务使用的是分布式文件存储数据库 MongoDB，以 JSON 格式存储数据。一个数据库中可以包含多个数据表，我们存储数据之前需要创建相应数据表。

1.在小程序云控制台的左侧导航栏，选择**小程序 Serverless > 云数据库**。

2.单击新建数据表按钮 + ，输入数据表名称（以 myNoteBook 为例）后单击**确认**。



3.新建数据库“myNoteBook”后，单击设置数据库权限按钮进入修改数据库权限界面，将 write 权限修改为 true。

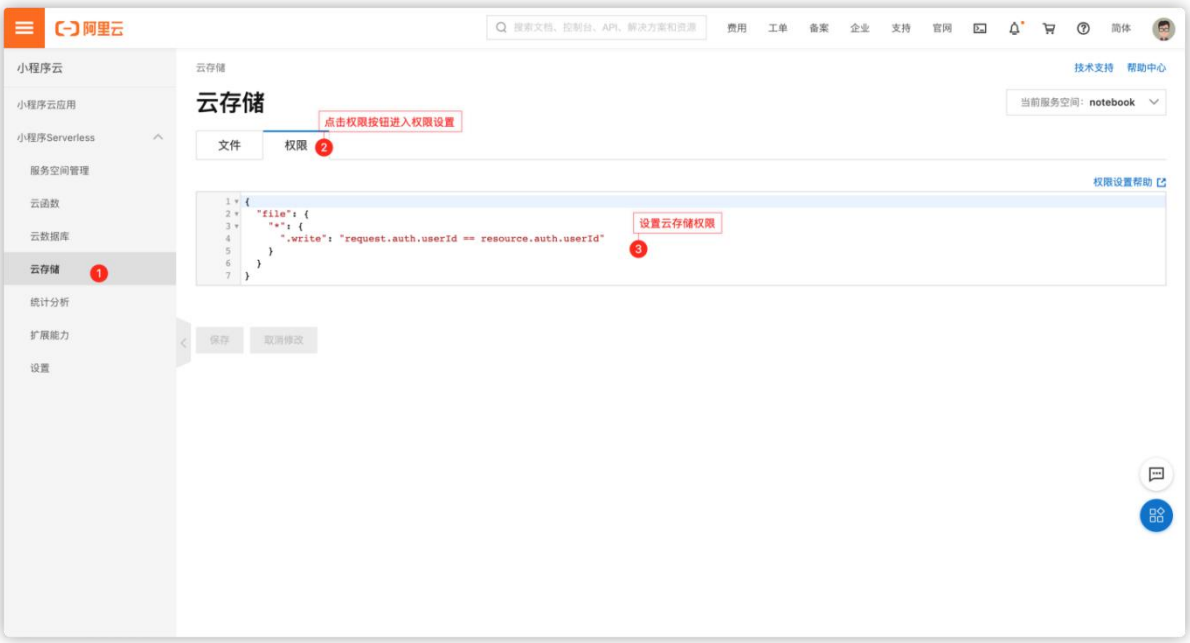


2、配置云存储访问权限

本例中记事本功能涉及图片的上传，所以我们需要配置云存储的访问权限：

1.在小程序云控制台的左侧导航栏，选择**小程序 Serverless > 云存储**。

2.单击选择**权限**页签进入权限设置，并设置云存储的读写权限。此时默认权限为数据创建者可写。

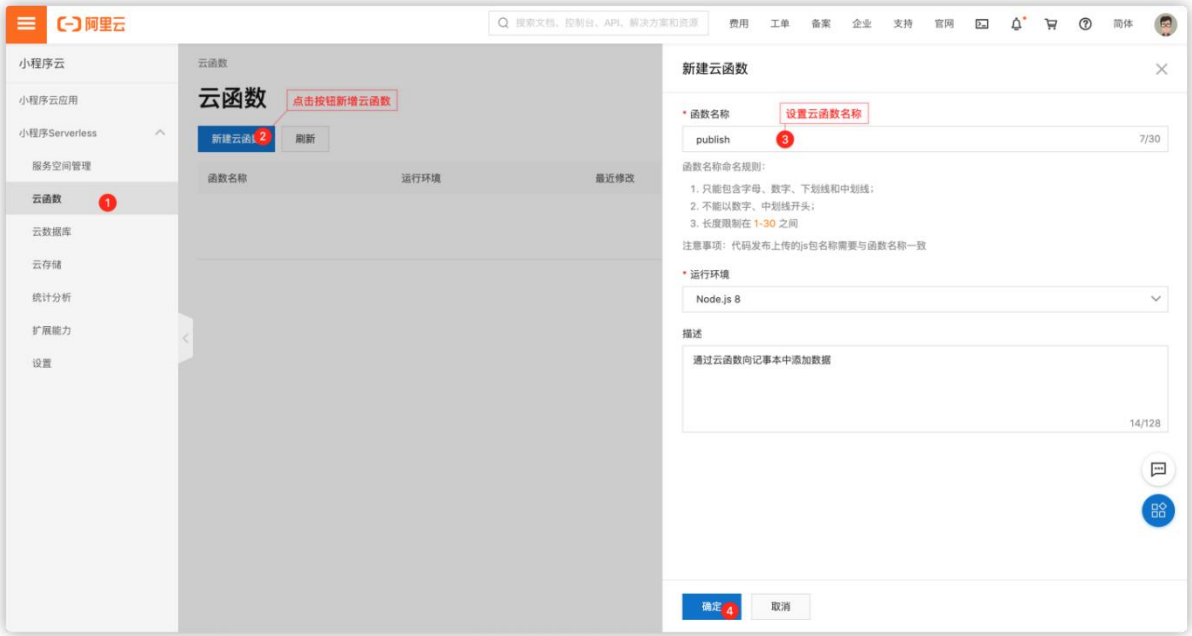


3、创建云函数并上传

云函数（FaaS）是一段运行在云端的、轻量的、无关联的、并且可重用的代码。无需管理服务器，只需编写和上传代码，即可获得对应的数据结果。使用云函数可以使企业和开发者不需要担心服务器或底层运维设施，也可以使代码进一步解耦，增加其重用性。在小程序端只需引入小程序云 Serverless 的 SDK，在应用的上下文中进行简单配置，即可调用云函数。

在调用之前，我们需要编写并上传云函数。
在本例中，我们会创建一个名为 publish 的云函数，它的逻辑是传入一个记事条目，把这个记事条目存储到云数据库中。

1.登录小程序云控制台，在对应的服务空间下，新建云函数。添加成功后可以在控制台查看云函数的名称、备注等信息。



2.我们在本地开发 js 代码，实现上述逻辑，保存为一个简单的 index.js 文件，如：

```
'use strict';
module.exports = async function (ctx) {
  return await ctx.mpserverless.db.collection('myNoteBook').insertOne({
    "title": ctx.args.title,
```

```
    "txt": ctx.args.txt,
    "pic": ctx.args.pic,
    "date": ctx.args.date,
    "userId": ctx.args.userId,
  });
};
```

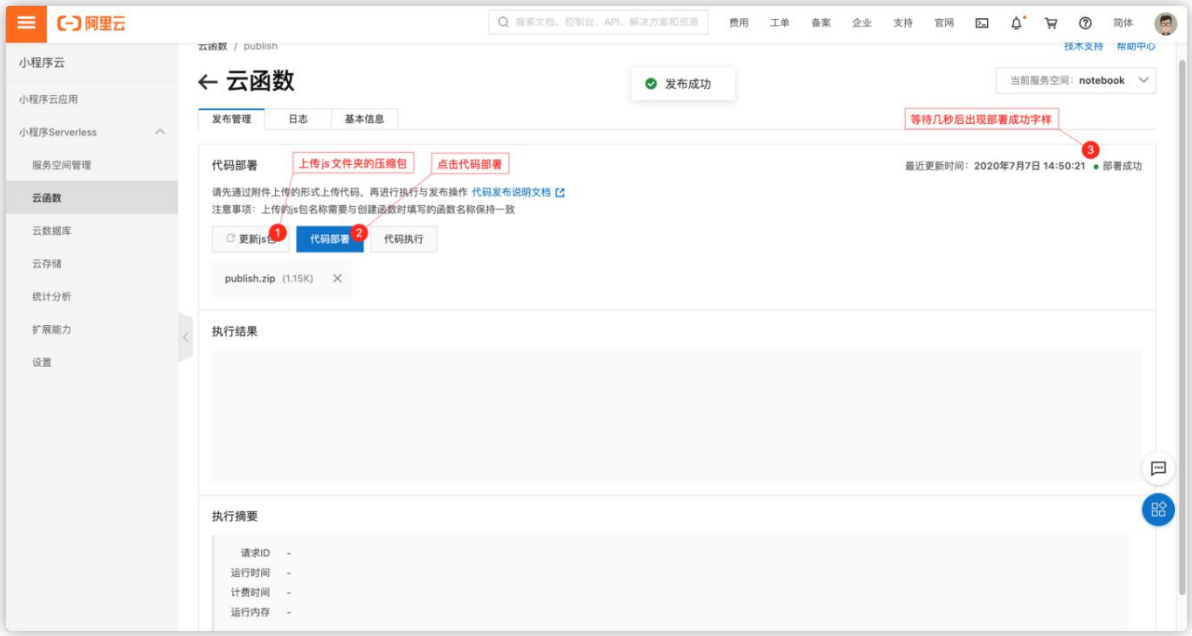
如上述，这个 js 文件实现了一个简单的逻辑：传入记事条目，然后存储到云数据库中。

1）把 index.js 文件放在文件夹 publish 中，并将此文件夹压缩成为 publish.zip 。
2）在小程序云控制台的云函数页面，单击云函数 publish 名称或左右管理进入当前隐函数的**详情页面**，单击上传 js 包上传 publish.zip ，显示上传成功后**单击代码部署**。

代码包上传说明：

- 代码包的名称必须和在控制台上创建的函数名称一致。
- 代码包必须是.zip 文件。
- 上传的代码包必须包含 index.js 文件。

出现如下界面说明部署成功（注意右上角“部署成功”的提示）：



如果您需要更新云函数，只需要点击“更新 js 包”按钮并上传新的 publish.zip 文件即可。

说明：只修改本地的文件是不能修改云函数内容的。每一次试图修改云函数，都需要重新制作压缩包并在后台云函数点击“更新 js 包”。

2.7 步骤七 小程序开发

1、首页获取记事列表数据

我们可以调用小程序云 Serverless SDK 的数据库接口获取数据：

```
const app = getApp();const { mpServerless } = app;const noteBookCollection = mpServerless.db.collection('myNoteBook');const PAGE_SIZE = 5;

Page({
  ...
  fetchPageData: async function () {
    noteBookCollection.find({
      "userId": app.globalData.userId,  // 假设我们已经取得 userId 并保存在 globalData 中
    }, {
      sort: {
        date: -1,
      },
      skip: (this.data.pageNum < 0) ? 0 : PAGE_SIZE * this.data.pageNum,
      limit: PAGE_SIZE,
    })
    .then((res) => {
      if (res.success) {
        // 更新列表
        this.setData({
          datalist: this.data.datalist.concat(res.result),
        });
      } else {
```

```
          // 失败提示
        }
      });
  },
  ...
})
```

2、新增记事条目

新增一个记事条目时，只需调用我们已经在上述中创建的云函数：

```
var util = require('../../utils/util.js');const app = getApp();const { mpServerless } = app;

Page({
  publishNote: async function() {
    mpServerless.function.invoke('publish', {
      "date": util.formatTime(new Date()), // 获取时间信息
      "userId": app.globalData.userId, // 小程序用于 userId
      "title": // 标题
      "txt": // 内容
      "pic": // 上传获得的图片地址
    })
    .then((res) => {
      if (res.success) {
        // 上传成功
      } else {
        // 失败提示
      }
    });
  }
})
```

在本小程序中，还可以为一次记事附加一张图片。我们先上传图片，然后把上传后获得的链接附带在记事条目中即可。上传逻辑参考：

```
var util = require('../../utils/util.js');const app = getApp();const { mpServerless } = app;

Page({
  ...
  uploadImg: function () {
    wx.chooseImage({
      success: ((res) => {
        const path = res.tempFilePaths[0];
        const options = {
          filePath: path,
          headers: {
            contentDisposition: 'inline',
          },
        };

        mpServerless.file.uploadFile(options).then((image) => {
              // 获得图片路径
        });
      })
    });
  },
  ...
})
```

3、获取记事详情

在首页点击对应记事本条目时，将跳转到记事详情页。我们需要在页面跳转时传入对应记事的_id。

```
// pages/index/index.wxml

……
  <navigator url="/pages/detail/detail?dataId={{datalist[index]._id}}" class="txt">
    ……
  </navigator>
……
```

利用传入的 id 信息，在 detail 页面就可以进行对应的信息拉取：

```
// pages/detail/detail.js
var util = require('../../utils/util.js');
const app = getApp();
const noteBookCollection = app.mpServerless.db.collection('myNoteBook');

Page({
    onLoad: function (options) {
    noteBookCollection.find({
      _id: options.dataId,
    })
    .then((res) => {
      if (res.success) {
        this.setData({
          id: options.dataId,
          txt: res.result[0].txt,
          pic: res.result[0].pic,
          date: res.result[0].date,
          title: res.result[0].title,
        });
      } else {
        // 失败提示
      }
    });
  },
})
```

4、删除记事条目

在首页浏览时，可以点击删除按钮直接删除记事条目。删除时调用代码如下：

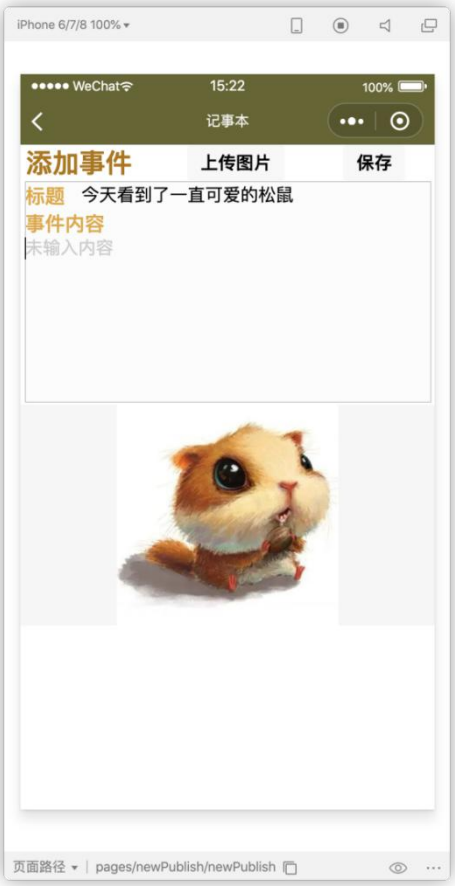
```
// pages/index/index.jsconst app = getApp();const noteBookCollection = app.mpServerless.db.collection('myNoteBook');

Page({
  ...
    delete(event) {
    const deleteAction = () => {
      const id = this.data.datalist[event.currentTarget.id]._id;
      noteBookCollection.deleteOne({
        _id: id,
      })
      .then((res) => {
        if (res.success) {
          wx.showToast({
            title: '删除成功',
          });
          // 刷新页面
        }
      });
    }

    wx.showModal({
      title: '提示',
      content: '请问确认要删除吗？',
      success: ((res) => {
        if (res.confirm) {
          deleteAction();
        }
      })
    })
  },
})
```

三、成果展示

小程序页面展示：



四、总结

至此，我们已经基于小程序云 Serverless 提供的云函数、云数据库、云存储等 BaaS 能力，快速实现了一个记事本微信小程序的开发。在这个过程中我们体会到，使用小程序云 Serverless 之后，我们通过 API 方式即可方便地获取云函数、数据存储、文件存储、音视频、图像处理等服务，不需要关心服务器或底层运维设施，可以更专注于代码和业务本身，大幅提升研发效率。

钉钉搜索 35248489，加入阿里云云原生应用研发平台 EMAS 技术交流群，探讨最新最热门的应用研发技术和实践。（或钉钉扫码加入）