极 客 邦 科 技 双 数 研 究 院

InfoQ 研究中心

开发者技术及生态发展 2030

InfoQ 极客传媒

目录 contents

目录 contents

目录 contents

01 移动应用生态与开发者技术发展现状

1.1 移动开发生态概览

2007年，苹果推出iOS操作系统的前身——iPhone OS；紧接着在2008年，Android 1.0正式发布。此后数年间，iOS和Android两大系统快速在全球范围内扩展，逐步确立了对智能手机操作系统市场的主导地位，并推动移动开发生态走向高度集中与标准化。在这一阶段，开发范式围绕App Store和Google Play构建，推动了全球移动应用的爆发式增长，也促成了以iOS和Android为基础的开发者体系的形成和成熟。

2019年，华为发布鸿蒙操作系统（HarmonyOS）1.0，开始打破既有格局，并在“多设备协同”、“全场景覆盖”等方向持续演进。2025年，鸿蒙操作系统已进入发展的第六个年头。根据Counterpoint最新市场数据，截至2025年第二季度，中国智能手机操作系统的市场份额结构已出现明显变化：Android占比逐渐下降至66%，iOS保持在16%，而鸿蒙系统持续上升至17%，并已连续六个季度在中国市场超过iOS。截至2025年12月，HarmonyOS 5宣布其终端设备数量已突破2700万。

这意味着，移动操作系统格局正从过去的两极结构，逐步演化为多元共存的新阶段。

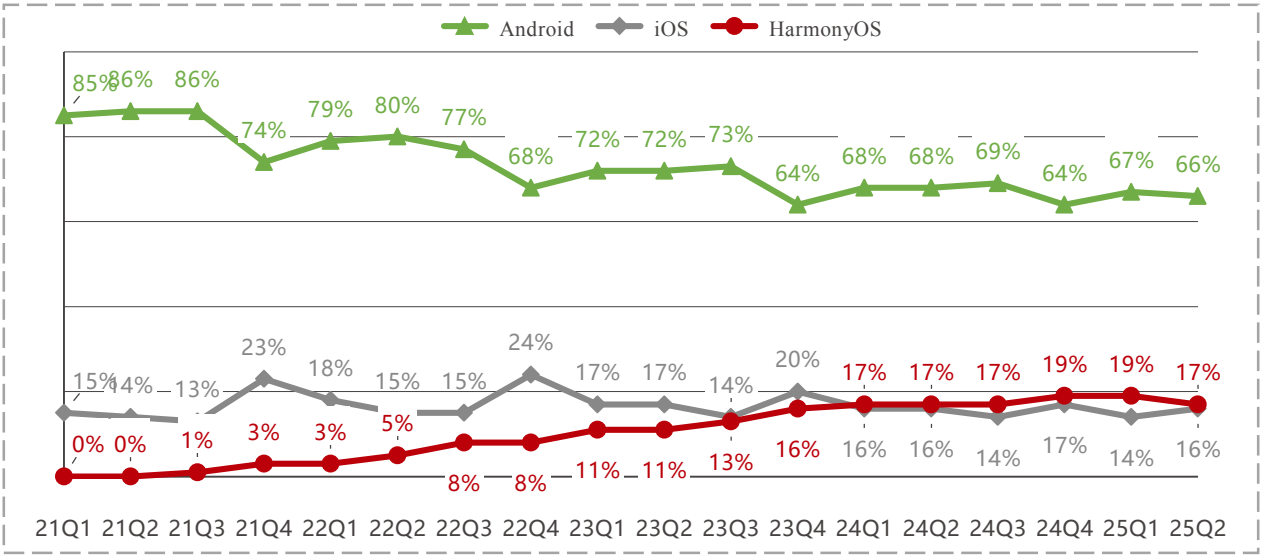


图 2021-2024年中国智能手机操作系统季度市场份额　　数据来源：Counterpoint

关键观点 | **由iOS和Android长期主导的移动操作系统格局，正在逐渐被打破，鸿蒙系统正成为新的关键力量。**

市场格局的变化也显著体现在APP应用生态的载体上。根据工信部数据，2022年以来，其监测的中国市场在架应用数量稳定在260万左右，代表iOS生态的App Store（中国区）与代表Android生态的第三方应用商店的应用数量比例基本持平。总体来看，这两大平台的应用数量保持稳定，进入以应用质量竞争为主的新阶段。

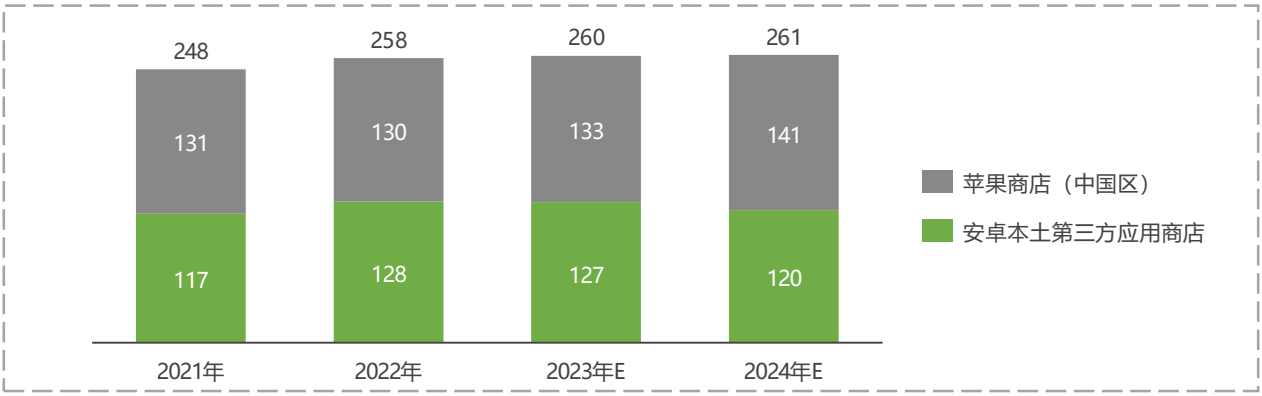


图 2021-2024年在架应用数量（万）　　数据来源：工信部

注：2024年起，工信部不再公布监测的APP数量，故该数据为结合苹果官方公布应用数、监测平台数及其他公开数据的估算数

与之相对应的是，鸿蒙操作系统的应用处于快速爬坡阶段。2024年6月，华为宣布启动能够满足用户使用时长99.9%的TOP 5000应用适配和开发。自2024年初至年底，鸿蒙应用与元服务数量从200个跃升至2万个，现正向其10万应用的阶段性目标冲刺，展现出鸿蒙操作系统持续构建生态的能力与决心。

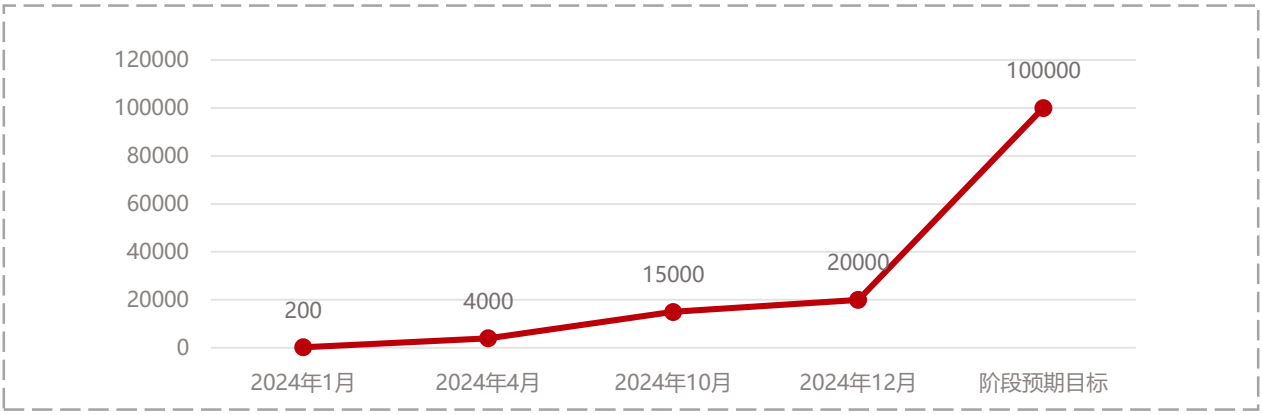


图 鸿蒙应用及元服务上架数量（个）　数据来源：官方

应用生态不仅在数量上演进，其内部结构也日趋多元化。以App Store（中国区）为例，截至2025年7月，工具类（11.5%）、教育类（10.7%）与商务类（10.4%）应用占比最高，反映出用户对效率与专业服务的持续关注。生活类、健康健美类、美食佳饮类等日常场景应用合计占比接近20%；而游戏、娱乐、音乐等内容消费类应用仍是生态重要组成。

相比之下，参考、新闻、导航、图书等传统资讯类应用占比持续收窄，表明用户获取信息的方式正转向多模态交互。以国内某Android应用市场为例，其应用结构呈现出与上述相同的趋势。这意味着主流平台的生态扩展，正从功能工具导向逐步走向以场景和服务入口为核心的融合生态体系。

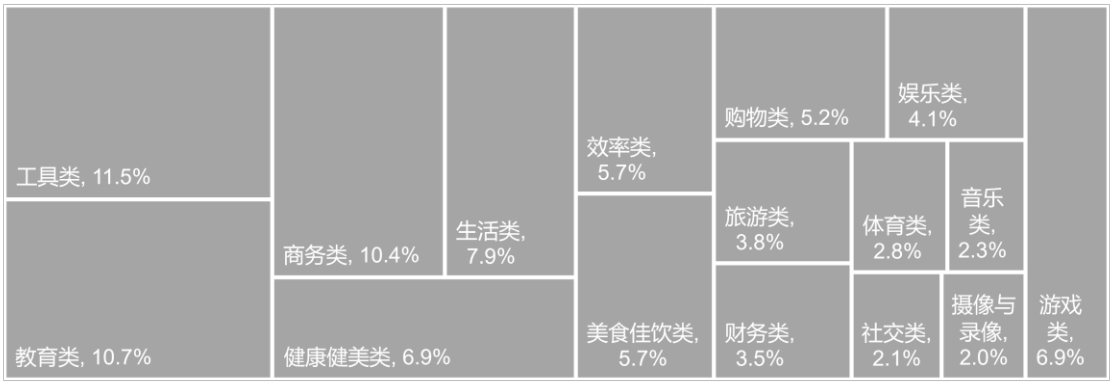


图 App Store应用类型分布　数据来源：第三方监测平台数据2025.7.30

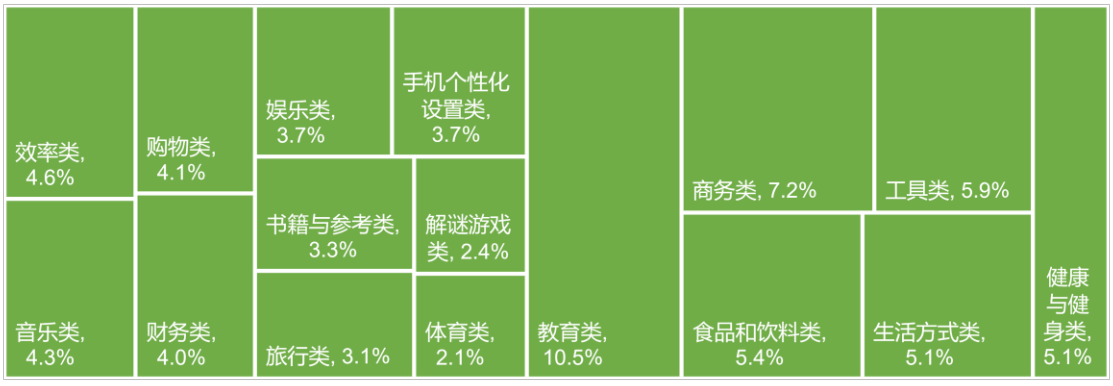


图　安卓第三方应用商店应用类型分布　数据来源：第三方监测平台数据2025.7.30

关键观点 | **移动应用市场正从数量扩张转向结构优化与场景服务化，鸿蒙系统正展现出快速成长的潜力与决心。**

除此之外，操作系统向开发者开放能力的程度，核心体现在其对外暴露的API数量和演进节奏。iOS与Android两大主流移动操作系统在历年系统迭代中，均通过小版本升级持续引入新的API，为开发者解锁更多系统能力。

从历史趋势来看，iOS更强调系统的稳定性与开发体验一致性，API更新与操作系统的年度稳定迭代节奏保持一致，采用渐进式方式引入新功能，从而在创新与兼容之间实现平衡。

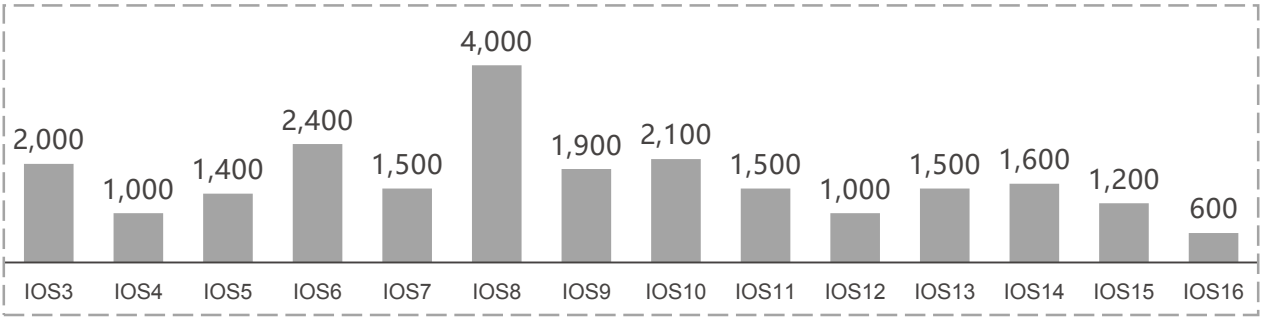


图 iOS各版本间预估新增API数量（个）数据来源：iOS API 官方及开发者对比差异文档

相比之下，Android早期缺乏固定的操作系统版本发布时间表，更倾向于在关键版本中集中式释放大量新API，以快速支持新硬件、新交互和新架构的需求。

此外，在iOS和Android生态的早期，伴随着SDK和API的升级演变，iOS和Android均会发布**API差异报告**。通过“新增/变更/废弃”等标记方式，清晰标注接口的变化与兼容性影响，同时配套详细注释、对照示例、版本适配策略说明等内容。这类实践为早期开发者理解系统演化路径、及时调整适配提供了关键支持。

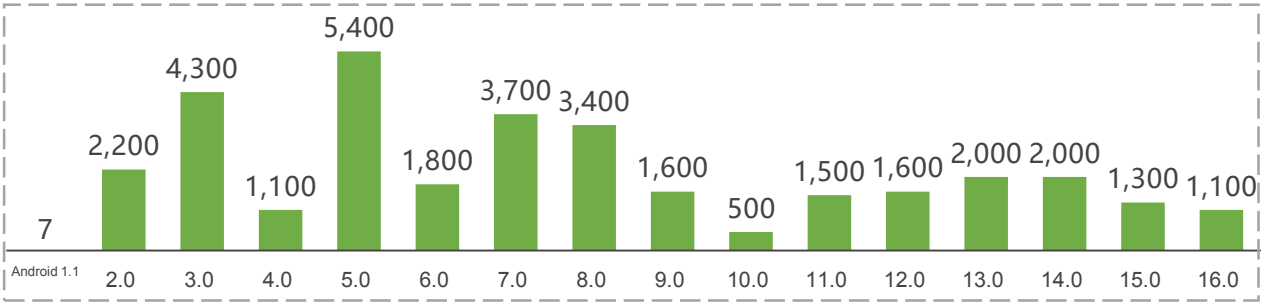


图 Android各版本间预估新增API数量（个） 数据来源：Android官方API各版本差异文档

随着操作系统逐步迈入成熟阶段，iOS与Android的API更新数量均趋于稳定化，新增能力的节奏进入常态化周期，这也反映出平台技术体系的成熟与生态演进的稳态特征。

对于HarmonyOS而言，截至2025年，HarmonyOS已经提供了90多个Kit，覆盖30000多个API接口，为开发者提供了全面、易用的系统服务。

关键观点 | **iOS与Android的API演进已趋于稳定，而鸿蒙系统正通过快速扩展SDK/API，构建自己的开放能力体系。**

与此同时，操作系统的承载形态也正发生显著变化。早期移动操作系统主要围绕智能手机构建生态体系，如今已延伸至多类型设备。iOS系统在iPhone之外，延展至iPad、Apple Watch、Apple TV等多终端设备；Android在平板、电视、可穿戴设备及车载系统中实现广泛部署；鸿蒙自诞生之初即确立“1+8”设备战略，支持包括手机、平板、智慧屏、手表、耳机、车载设备、眼镜等在内的多样化终端形态。

这种设备形态的扩展，也推动了操作系统活跃设备数的持续增长。根据官方披露数据显示，截至2024年，Android全球活跃设备数达33亿台，继续保持主导地位；iOS设备数从2021年的18亿增长至23.5亿，展现稳健扩张态势；鸿蒙版设备数则从2021年的2.2亿跃升至11.9亿台，三年间实现近五倍增长，这一趋势不仅巩固了鸿蒙操作系统在中国本土市场的生态基础，也为其构建“超级终端”生态体系提供了坚实支撑。

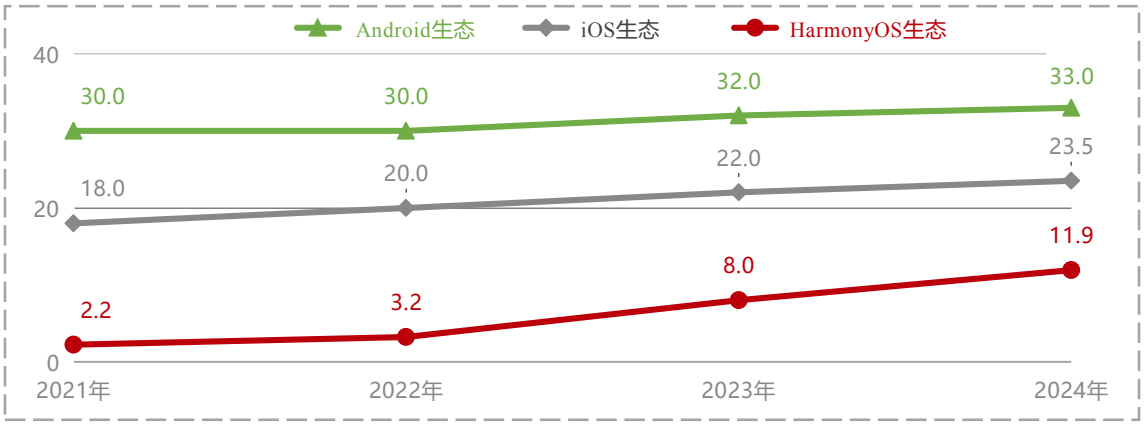


图 三大移动生态活跃生态设备数量（亿） 数据来源：官方

除此之外，开发者作为移动操作系统生态繁荣的核心推动者，其数量变化反映出生态平台的吸引力。

从全球开发者数量来看，iOS平台依托长期稳健的工具体系与高价值用户基础，开发者数量持续增长，从2021年的2800万增长至2024年的5176万，四年间增长近85%，并在2025年预计保持稳定。Android作为全球最广泛部署的操作系统，虽然缺乏官方披露数据，但从各家调研机构数据来看，其全球开发者体量早已突破千万级。

鸿蒙操作系统的开发者生态虽然起步较晚，但近年来增长势头显著。全球开发者数量从2021年的60万增长至2024年的254万。而标志生态成熟的拐点出现在2025年：开发者规模增至1000万人，较前一年实现293%的增速跃升，标志着鸿蒙生态正式进入质变加速期。开发者的持续涌入，为平台创新与服务多样性提供了源源不断的动力，也进一步巩固其生态体系的可持续发展基础。

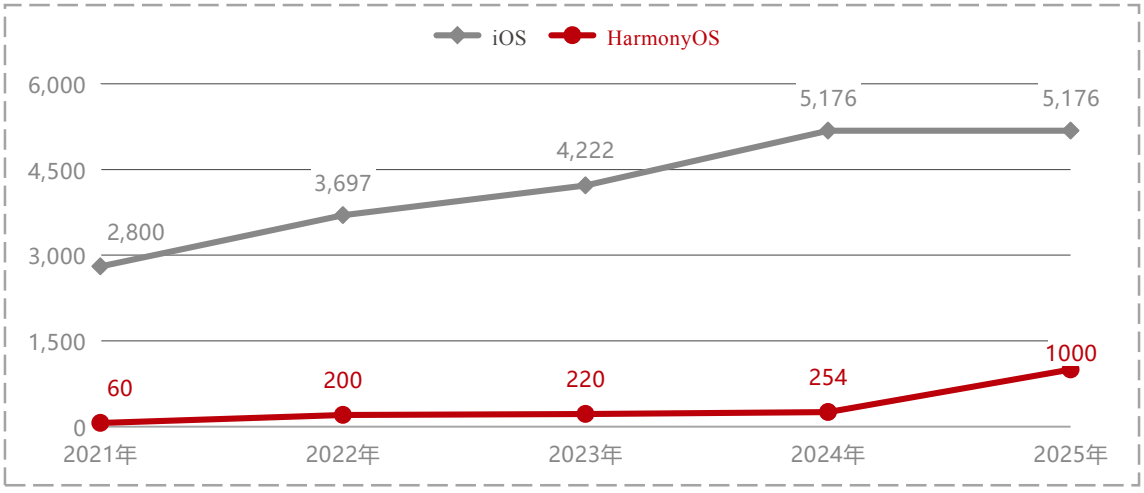


图 iOS与鸿蒙生态注册开发者数量（万） 数据来源：第三方调研报告与官方

关键观点 | **移动操作系统正从单一设备形态走向多终端协同，生态设备数量与开发者规模同步扩张，而鸿蒙正加速迈向质变期。**

总体来看，移动开发生态正在从系统主导—应用驱动—设备联动的线性发展，演变为以多系统共存、多设备协同为特征的复杂生态体系。操作系统格局的重塑、应用结构的多元、设备边界的延展与开发者队伍的壮大，正在共同推动移动生态迈入一个更加动态、多维、开放的竞争阶段。这一变化既对平台能力提出更高要求，也为开发者带来了前所未有的技术机会与生态红利。因此，理解当下生态格局的变迁，是把握未来开发范式转型与平台价值重构的关键起点。

1.1 核心观点总结

移动操作系统格局：由iOS和Android长期主导的移动操作系统格局，正在逐渐被打破，鸿蒙系统正成为新的关键力量。

移动应用格局：移动应用市场正从数量扩张转向结构优化与场景服务化，鸿蒙系统正展现出快速成长的潜力与决心。

API更新节奏：iOS与Android的API演进已趋于稳定，而鸿蒙系统正通过快速扩展SDK/API，构建自己的开放能力体系。

设备与开发者格局：移动操作系统正从单一设备形态走向多终端协同，生态设备数量与开发者规模同步扩张，而鸿蒙正加速迈向质变期。

1.2 移动开发技术体系与生态体系分析

面对正在深化演进的移动开发生态，平台之间的竞争早已不再仅限于市场份额的争夺，更体现在其对开发者的支持能力与生态完整性的持续建设上。无论是操作系统自身提供的原生开发体系，还是围绕平台构建的第三方框架、多设备适配能力，均影响着开发者的选择与开发体验。

因此，理解iOS、Android与HarmonyOS这三大主流移动开发生态，在开发技术体系与生态体系构建上的异同，不仅有助于厘清平台能力的真实边界，也为企业、开发者在技术选型、生态投入与长期发展策略上提供重要参考。

因此，本小节，我们将围绕开发技术体系与生态体系两大视角，系统梳理三大平台的核心能力、工具支持与生态兼容性，勾勒出当前移动开发的能力全景、技术与生态演进路径。

1.2.1 开发技术体系分析

在近20年的发展过程中，iOS与Android分别围绕自身生态战略，构建起相对成熟且具有鲜明特色的开发技术体系。从早期的开发体系探索，到后期的开发体验优化、IDE生态打磨、系统能力持续开放，两大平台不断提升开发体验与系统整合能力，形成了各自独有的技术范式与开发者支持体系。

相比之下，鸿蒙系统虽起步较晚，但其自研开发语言、声明式框架和分布式架构等技术，使其在构建多设备一致体验与工具链闭环方面展现出差异化优势。

本节将围绕编程语言体系、原生框架、开发工具链、SDK/Kit模块和系统级能力开放程度四个维度，系统分析三大平台官方提供的核心开发技术构成，揭示其技术路线背后的生态逻辑与开发者价值主张。

	iOS	Android	HarmonyOS
编程语言	先Objective-C 后Swift	Java和Kotlin并存， C/C++为辅	ArkTS与仓颉并存
原生框架	传统UIkit，新兴SwiftUI	传统Android，新兴 Jetpack Compose	ArkUI
IDE及工具链	Xcode	先Eclipse+ADT插件 后Android Studio	DevEco Studio
SDK/Kit等系统级能力	iOS SDK 26	Android API 36	HarmonyOS API 20

图 三大移动生态开发体系示意图

回顾iOS平台近二十年的技术演进，其开发体系以自研技术栈构建为核心特征，借助稳定的语言演进、深度的系统能力开放，持续引领移动开发范式升级，并构筑了封闭但高度一致化的开发者体验。

阶段一

阶段一（2007-2011年）：早期开发生态探索期

- 2007年iPhoneOS（后改名iOS）随第一代iPhone亮相，采用基于Objective-C的Cocoa Touch框架进行开发。
- 在对应用形态进行一段时间的讨论和研究后，2008年3月，苹果正式推出iPhone SDK、Interface Builder、iPhone模拟器和应用调试工具，以更好地响应开发者开发第三方应用的需求。
- 2008年10月，Xcode 3.0版本发布，首次将SDK、Interface Builder、模拟器与调试工具整合至统一开发环境中，极大提升了开发效率与工具一致性。
- 在这一阶段内，iPhone SDK在早期得到迅速拓展，UIKit、消息推送的自有API和地图、支付等第三方服务API，为开发者提供了较为丰富的系统能力支持。

关键观点 **阶段一：iOS在早期通过Objective-C、Xcode与iPhone SDK奠定了完整开发生态的基础框架。**

阶段二

阶段二（2012-2019年）：现代化自有开发生态建立期

- 2012年，iOS改用自研地图服务，并发布官方地图API（MapKit API），标志着苹果在关键系统服务上强化自有能力布局。
- 2012年，苹果推出Swift编程语言，并于2014年推出Swift 1.0，为开发者提供更安全、现代化的语言选择。
- 2014年，随着iOS 8的发布，苹果一次性开放4000余项API，包括HealthKit、HomeKit、Metal、Touch ID、PassKit、相机接口等，这些系统能力覆盖健康管理、家庭自动化、图形加速、安全支付等多个核心场景。这一轮系统级API的集中开放，显著提升了iOS官方API体系对开发者的吸引力。
- 2019年，苹果发布SwiftUI，正式引入声明式UI编程范式，为iOS原生开发体验带来深层变革。
- 在此阶段，Xcode随系统与语言演进同步升级，持续集成对iOS最新特性的支持，并不断优化对Swift语言与SwiftUI框架的开发体验，成为支撑开发生态稳定演进的关键工具。

关键观点 **阶段二：iOS通过原生语言、框架等核心系统能力的构建，减少了对三方API的依赖，奠定了原生开发生态的基础设施。**

阶段三

阶段三（2020年至今）：多设备适配、智能化探索期

- 2020年后，除了搭载自研手机芯片外，苹果自研芯片开始向电脑和Vision Pro拓展，苹果在软硬件深度整合下，开始在开发体系中探索跨设备开发。除此之外，iOS通过Handoff、Continuity等机制支持数据/任务跨设备（iPhone/iPad/Mac）。
- 随着SwiftUI的持续演进，苹果不断提升声明式UI在多终端平台的适配能力，逐步持续优化iOS、iPadOS、macOS、watchOS、tvOS等系统的特性和设备适配。
- 2020年后，苹果持续推出包括Core ML、Vision、Natural Language、App Intents在内的一系列机器学习与智能服务框架，为开发者提供AI能力的操作系统层面的支持。
- 2024年，苹果正式发布Apple Intelligence，标志其平台能力向AI原生化迈出关键一步。

关键观点 **阶段三：iOS开发体系通过多终端适配与智能服务框架的发布，迈向智能化原生能力与跨端体验的一体化升级阶段。**

经过三个阶段的发展，iOS开发技术体系已逐步成熟：以Swift为核心语言、SwiftUI为主流开发框架、Xcode为统一开发环境，并辅以随系统迭代稳定演进的SDK，形成高度集成、体验一致的开发闭环。与此同时，Core ML、App Intents等原生智能服务的持续扩展，使iOS在保持开发效率的同时具备多设备协同与智能化能力，巩固其在全球开发生态中的技术领先地位。

自2008年首个Android版本发布以来，Android凭借其开放架构与庞大设备生态，迅速成长为全球最主流的移动操作系统之一。在其发展过程中，Google不断完善开发工具链，迭代语言与框架体系，推动开发者从早期的Eclipse工具迁移至现代化的Android Studio集成环境，并持续拓展至跨平台、智能化、多终端等新方向。

阶段一

阶段一（2008-2013年）：基础架构建立期

- 2008年，Android 1.0发布，并以AOSP进行开源发行。应用主要以Java语言进行开发，官方提供Android SDK与模拟器。同时，支持NDK开发Android应用。该阶段Android使用Eclipse+ADT插件作为官方开发环境，核心UI框架基于View/Activity布局。
- SDK、模拟器与核心API持续迭代，多点触控、GPS、相机、传感器、地图等能力逐步开放，丰富了移动应用场景。

关键观点 **阶段一：Android通过Java语言、SDK/NDK与Eclipse构建早期开发生态，奠定开放式生态基础**

阶段二

阶段二（2014-2021年）：工具链升级与开发体系标准化期

- 2014年，Android Studio 1.0正式发布，集成编译器、调试器、布局编辑器、性能分析工具及APK打包功能。原先的Eclipse+ADT插件于2015年逐步停止官方支持，以推动开发者全面迁移至Android Studio。
- 2014年，Android 5.0引入Material Design设计语言，统一Android视觉与交互风格。
- 2016年，Android Support Library（后发展为AndroidX）日趋完善，构建兼容性与模块化开发基础。2018年，Google将Android Architecture Component架构库整合为 Android Jetpack 组件库，对应用开发提供统一指导，从而加速开发过程并提升应用质量。
- 2017年，为更好适应现代化应用开发，Kotlin编程语言诞生，并于2020年成为Android官方首选开发语言。此外，Android Studio 3.0增加了对Kotlin的全面支持，并提供代码转换工具，帮助开发者快速将Java应用转换为Kotlin。
- 2019年，Jetpack Compose预览发布，开启声明式UI编程转型。

关键观点 **阶段二：Android通过Android Studio、Kotlin与Jetpack构建现代开发栈，为碎片化设备提供统一指导**

阶段三

阶段三（2022年至今）：生态整合与智能化转型期

- 2022–2024年，Android持续拓展多终端能力，并陆续支持Android TV、Wear OS、Android Automotive等；2023年，Kotlin Multiplatform正式发布，并陆续支持跨iOS、Android、桌面、Web的跨平台开发框架。
- 2023年，ML Kit加入Studio工具链，支持AI能力嵌入。
- 2025年3月，Google宣布Android未来的核心功能开发将转入内部闭环，仅会定期向AOSP推送阶段性成果，同时AOSP的更新频率也将放缓，或将阻挡第三方开发者们的持续贡献热情。

关键观点 | **阶段三：Android通过多终端扩展、跨平台框架与AI开发工具迈向智能化，但开源更新放缓或影响第三方生态活力。**

经过十余年的演进，Android已构建起成熟且现代化的开发技术体系：以Kotlin为主流语言、Jetpack Compose为声明式UI框架、Android Studio为官方IDE，辅以Jetpack架构组件、ML Kit等AI工具链，支撑起从手机到平板、穿戴、汽车等多终端的统一开发体验。

与此同时，AndroidX、Kotlin Multiplatform等生态方案不断降低平台切换成本，增强代码复用效率。尽管生态碎片化与多设备适配仍是挑战，但当前Android开发生态已具备强大的扩展能力与工程效率，形成开放与集成并重的技术体系，是全球移动应用开发的重要基石之一。

自2019年首次发布以来，HarmonyOS作为华为自研的操作系统，持续推进“万物互联”战略的技术落地。在多设备协同、分布式体验、原子化服务等方向的不断探索下，其开发体系也经历了从兼容起步到独立成型、再到AI智能化演进的过程。伴随微内核架构逐步成熟、ArkTS和ArkUI开发范式逐步确立，HarmonyOS正在构建一套具备自主可控、分布式特性与智能原生能力的全场景操作系统开发体系。

阶段一

阶段一（2019-2022年）：开发体系建设期

- 2019年，HarmonyOS 1.0发布，方舟编译器等底层工具同时发布。
- 2020年，开发工具DevEco Studio 2.0发布，支持Java/JS开发（兼容原Android代码），但逐步引入Ark编译器和Ability框架（Page/Service Ability）。HMS Core持续完善，提供地图、推送、支付等替代Google API的SDK，生态建设加速。
- 2021年，HarmonyOS 2.0发布，支持原子化服务，分布式调度能力初步落地。eTS语言（ArkTS前身）在这一时期投入应用，并逐步成为后续主推的主要开发语言。
- 2022年，HarmonyOS 3.0发布， DevEco Studio升级到3.0，逐步完善ArkTS调试、UI可视化设计、ArkCompiler AOT支持，增强了开发效率。HarmonyOS 3.0 API 8迭代，该版本引入了声明式UI编程和分布式体验，支持跨设备自适应布局与原子化服务卡片。

关键观点 | **阶段一：HarmonyOS通过ArkTS、ArkUI与DevEco Studio构建基础开发体系，奠定多设备协同的开发能力。**

阶段二

阶段二（2023-至今）：开发体系完善和创新能力探索期

- 2023年，HarmonyOS 4发布，智慧助手小艺接入AI大模型能力，并开始探索操作系统层面的AI开放能力。
- 2024年，HarmonyOS 5发布，基于全新架构，支持全场景互联，鸿蒙智能等功能。新增Cangjie语言与ArkTS形成互补，Cangjie未来将持续深化Agent DSL与AI技术的融合，为鸿蒙应用开发者提供一码三端、AI Agent编程能力。DevEco Studio 5.0发布，支持AI智能辅助编程。
- 在此阶段，官方Kit模块增至92个，系统API总数突破3万，覆盖AI、安全、媒体、终端控制等。

关键观点 | **阶段二：HarmonyOS以多语言体系、原生AI能力与智能工具链，快速迈向全场景融合与智能驱动的开发体系。**

经过多个版本演进与技术体系迭代，HarmonyOS已建立起以ArkTS为核心语言、ArkUI为声明式开发框架、DevEco Studio为官方开发工具的完整原生开发体系。同时，通过Cangjie补充Agent领域开发、AI 能力的深度系统集成、HarmonyOS正在逐步形成区别于Android和iOS的新一代操作系统开发范式。当前，其开发者生态正在围绕多设备协同、AI智能服务融合等核心方向加速扩张，为下一阶段面向全场景智能终端的应用开发奠定坚实基础。

移动生态技术栈转变路径分析

从前文对iOS、Android和HarmonyOS的开发技术体系分析中可以发现，iOS和Android都在2015年前后发布了官方新编程语言（Swift和Kotlin）。两者虽然诞生于不同的技术背景，但背后驱动逻辑高度相似，旧语言体系在语法复杂度、开发效率、安全性及现代特性支持方面未能完全满足移动应用生态的快速演进需求。

新语言的推出，不仅是为了提升开发体验与代码质量，更是出于平台战略层面的考量：通过自研语言绑定操作系统与SDK演进节奏，增强开发者对平台的依赖与忠诚度。下图对比了旧语言体系与新语言体系的核心差异与演进方向。

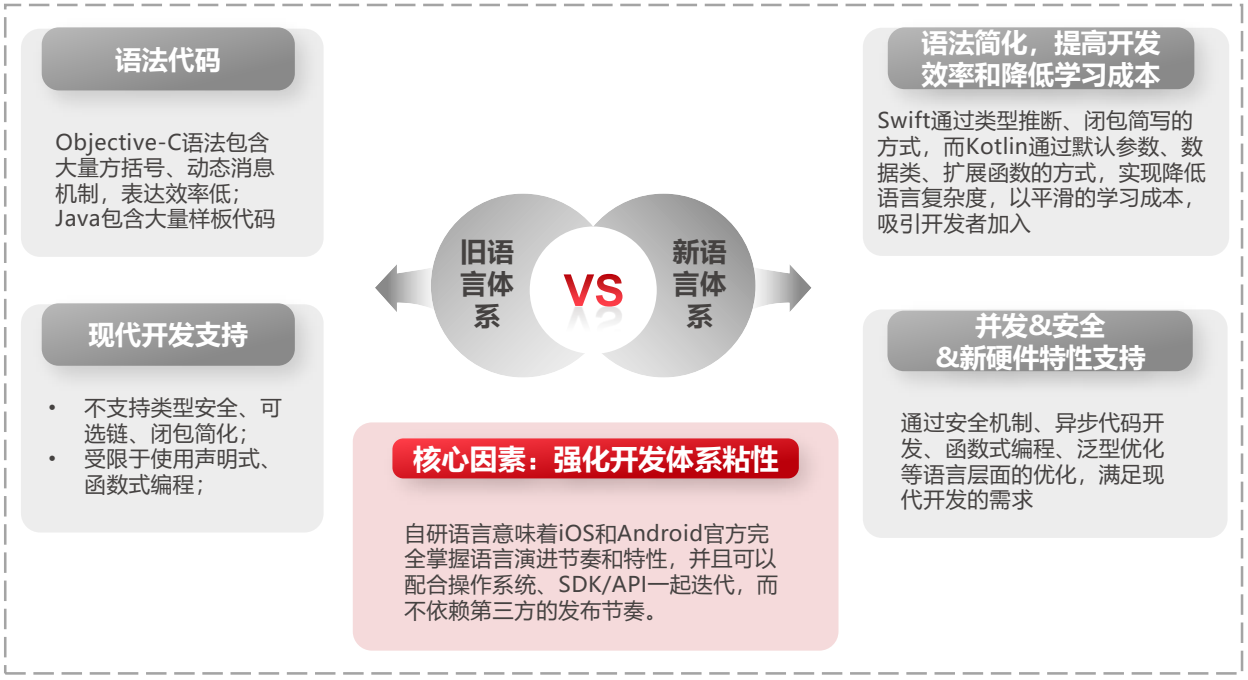


图 iOS和Android推出新语言体系Swift和Kotlin的原因分析

与编程语言的演进类似，iOS和Android也在2019年相继推出了声明式UI开发框架——SwiftUI和Jetpack Compose。这一举措既是为了应对传统UI框架在现代应用开发与多端适配中效率不足、体验欠佳的问题，也是对2015年后Flutter、React Native等跨平台框架冲击的直接回应。

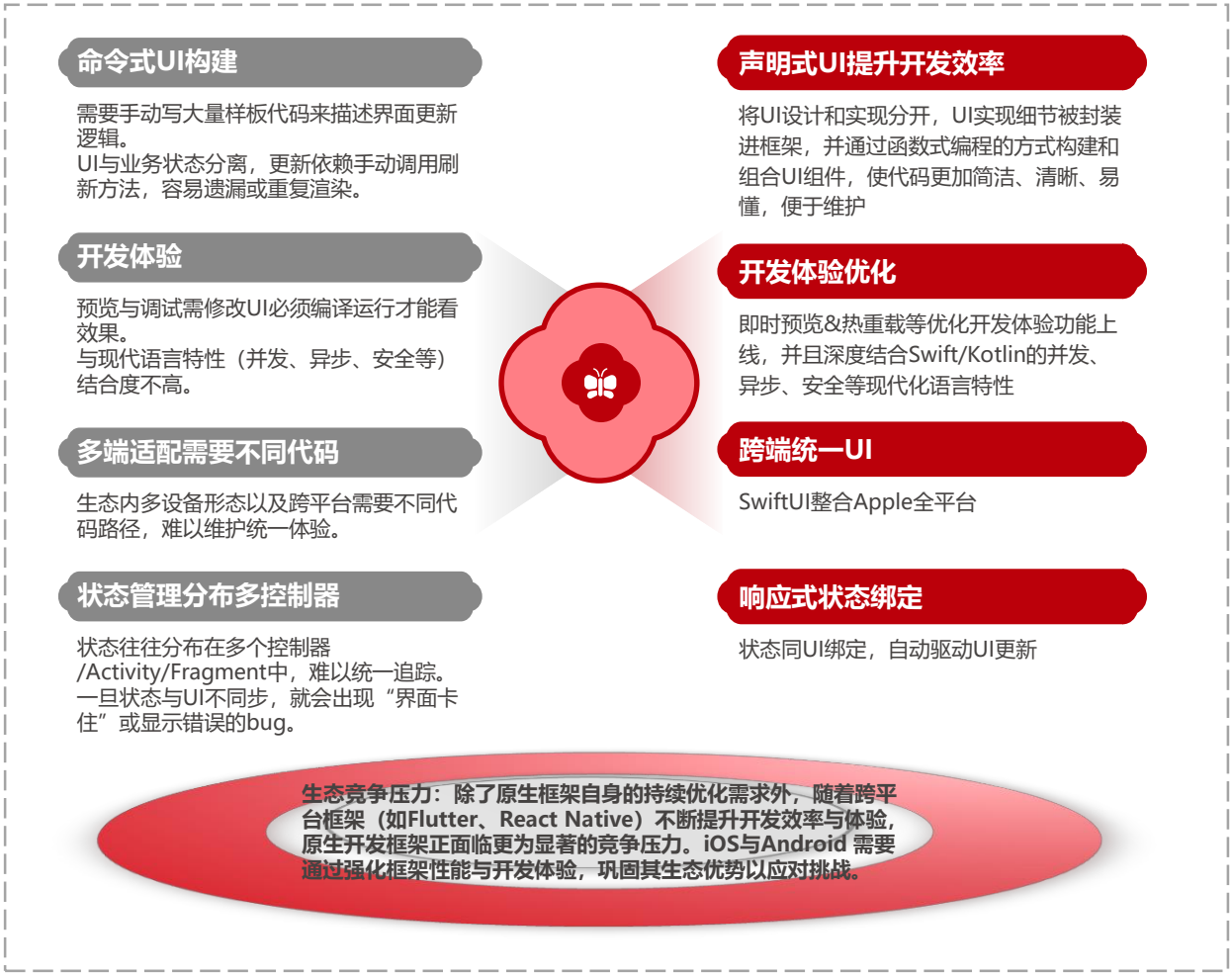


图 iOS和Android推出新框架SwiftUI和Jetpack Compose的原因分析

关键观点 | **官方开发体系的转变，不仅是为了突破旧语言与框架的效率瓶颈，更是出于强化生态掌控力与应对跨平台冲击的战略考量。**

开发者在应用开发的生命周期中，难免会面临技术栈的演进升级，或应用移植至新平台、跨生态运行的需求。此类迁移往往涉及开发模式、语言框架、系统能力等多方面的调整，不仅对开发者的学习曲线和技术成本构成挑战，也对平台方提出了更高的生态承接与引导要求。

迁移过程中，开发者关切的问题集中在迁移成本、性能表现、迁移带来的收益以及工具支持等。新平台应通过系统性宣传与信息透明，积极回应开发者的疑虑，传递迁移的可行性与价值，打消开发者对迁移过程的不确定性顾虑。

在此基础上，平台需构建完善的迁移配套体系，提供高效的自动化迁移工具、详实的技术文档以及多维度的社区支持，以切实降低迁移门槛，帮助开发者顺利推进迁移项目，快速实现落地与应用升级。

随着生态的不断完善，用户规模和商业价值稳步提升，平台应进一步引导开发者充分挖掘并利用新平台独有的技术能力与创新玩法，推动应用实现差异化发展，以激发生态持续活力，促进开发者生态从“迁移驱动”向“创新驱动”的健康演进。

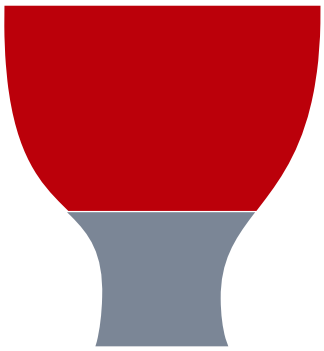
总体来看，该策略路径体现了“先破除顾虑、再降低门槛、后激励创新”的阶段性推进原则，构成了新平台打造可持续、高质量开发者生态的核心保障。

高效便捷的迁移工具——降低改造成本，缩短迁移周期

平台应提供与主流开发环境（如Android Studio、VS Code）无缝集成的一键迁移插件、语言转换器及UI适配脚手架等工具，支持开发者以最小改动迁移现有代码，最大程度保留项目结构与业务逻辑，显著降低初期技术改造成本和开发门槛

多维度社区与技术支持体系——提升问题解决效率，减少试错

建立专属迁移支持通道，包括迁移论坛、技术问答专区、在线交流群等，确保开发者在迁移过程中能够及时获得专业解答与经验分享



系统完善的迁移指南与示例——迁移适配有章法

围绕新平台设计理念和能力边界，针对典型迁移场景，编制详实的迁移路径指南，涵盖API对比、性能优化及最佳实践。同时，提供涵盖不同应用类型的示例工程，帮助开发者快速理解并复用成熟的项目架构和关键组件，实现高效落地。

迁移收益宣传——提供信心与激励，增强行动意愿

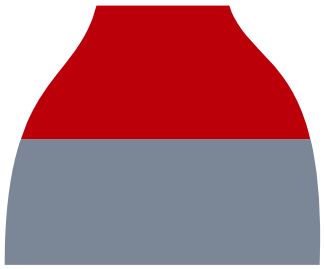
展示迁移后获得的新增用户规模、平台分发红利、系统能力提升带来的产品体验/收入增长等

应用和技术栈迁移要让开发者明白：不是重新开发，而是低成本换更大舞台

新应用特性的探索要让开发者明白：留下来、用起来、玩出彩，带来的是新玩法、新体验、新增长

让创新行为“看得见、赚得到”

针对使用新平台能力的应用，提供流量推荐、功能优先审批、专属分发位等政策激励



平台联合共创计划

设立“先锋开发者计划”或“平台共创营”，邀请头部开发者参与平台能力共建或玩法实验，形成示范引领

图 开发体系迁移两步走战略示意图

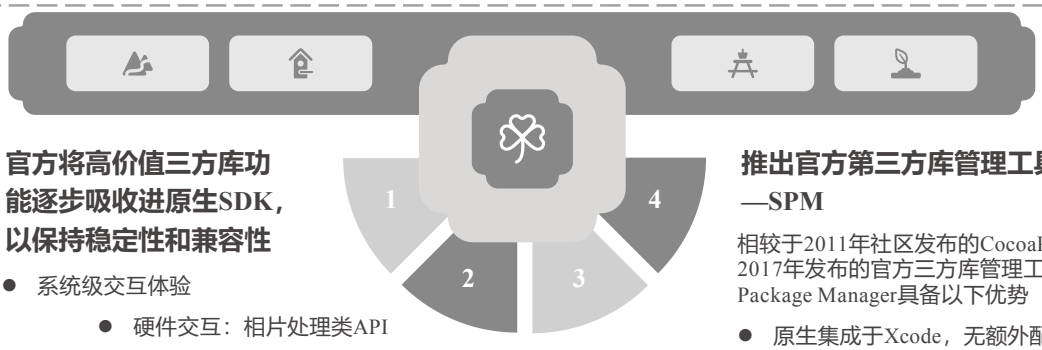
关键观点 **新开发体系应执行两步走战略：先以工具与支持降低迁移成本，再以独有能力激励创新，实现开发者生态由迁移驱动向创新驱动的转变。**

1.2.2 生态体系分析

除了官方提供的开发体系外，围绕操作系统所构建的生态技术资源，如第三方库、跨平台框架、Web开发支持与多设备开发能力，已成为开发者评估平台可持续性与易用性的关键指标。iOS、Android与HarmonyOS在这一维度的策略虽各有侧重，但均在尝试构建一个可扩展、兼容性强、适配多端的开发环境，以降低开发门槛、提升资源复用效率，并激活更广泛的开发者生态。以下将分别分析三大平台在第三方库支持、跨平台适配、Web能力拓展及多设备协同方面的生态构建路径与关键举措。

在开发生态层面，iOS始终坚持以官方标准构建统一技术体系，强化原生体验与系统一致性。苹果通过官方依赖管理工具、声明式开发框架、多设备适配方案等手段，形成高度集成的生态闭环。同时，在跨平台兼容、Web支持等方面，逐步开放部分能力，以适应多样化开发需求，实现性能、安全与多终端体验的平衡。

在**第三方库生态**方面，iOS并没有直接的经济激励或官方认证体系去推动三方库，并且iOS采取了“以官方API为中心”的策略，定期将核心/关键三方库功能吸收进官方SDK中，并通过SPM等官方工具进行管理，以强调系统稳定性、交互一致性与安全可控性。



官方将高价值三方库功能逐步吸收进原生SDK，以保持稳定性和兼容性

- 系统级交互体验
  - 硬件交互：相片处理类API
  - 软件交互：翻译类API
- 隐私和安全保障：生物识别API

推出官方第三方库管理工具——SPM

相较于2011年社区发布的CocoaPods，2017年发布的官方三方库管理工具Swift Package Manager具备以下优势

- 原生集成于Xcode，无额外配置
- 低侵入性，不直接改变修改项目配置
- 自动化依赖解析与构建

围绕官方API，提供一系列官方资源

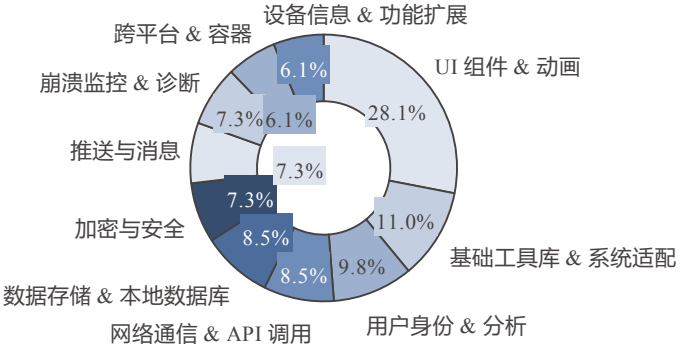
- 官方API文档内嵌代码示例
- Developer Forums官方问答系统
- Apple Developer Academy体系化课程

App Store审核机制间接管理三方库

- iOS官方：通过隐私清单+required reasonable APIs，把 三方库的合规状态变成App提交审核的必要条件。
- 应用开发者：必须主动去确认三方库的合规性，否则App无法上架
- 三方库开发者：被应用开发者倒逼要求提供隐私清单和签名

图 iOS三方库生态体系示意图

对于应用内的三方SDK，在WWDC23之后，iOS将三方库的合规责任部分转嫁给了应用开发者，通过隐私清单和审核规则，迫使开发者去监督 SDK，从而间接实现对生态的管理。



- Abseil
- AFNetworking
- Alamofire
- AppAuth
- BoringSSL / openssl_grpc
- Capacitor
- Charts
- Cordova
- DKPhotoGallery
- FBSDKCoreKit
- file_picker
- FirebaseCore
- Flutter
- GoogleDataTransport
- grpcpp
- GTMAppAuth
- GTMSessionFetcher
- Kingfisher
- leveldDB
- Lottie
- MBProgressHUD
- nanopb
- OneSignal
- OpenSSL

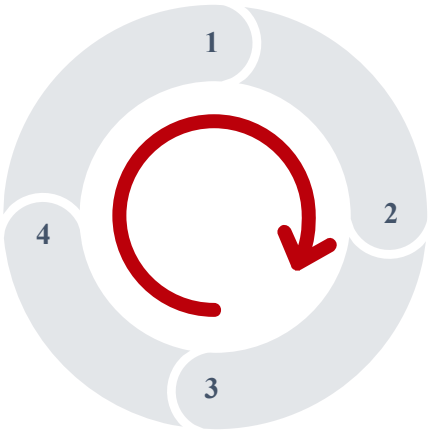
图 iOS官方公布的常用三方SDK分布和三方部分SDK示意图 数据来源：官网

系统克制开放

苹果在系统功能上采取克制策略，很多接口不开放或仅提供基础能力。

稀缺催生繁荣三方库

功能缺口成为刚性需求，由于**iOS平台的独特市场地位和成熟生态**，开发者不会轻易舍弃生态，而是通过第三方库来补位，第三方库被迫承担起补位角色，为开发者提供缺失或更灵活的功能实现。



三方库开发者获得职业或商业回报

在这个循环中，三方开发者往往能获得**技术声誉**（如作者成为社区意见领袖）、**职业机会**（进入大厂或加入苹果官方团队）、以及**商业收益**（库被收购、公司因库而获融资或变现）。这些收益又会吸引更多人继续投入创新，而苹果的克制开放反过来制造新的稀缺，循环由此延续。

官方API吸收成熟三方库方案

当某些第三方方案成熟并形成共识时，苹果往往将其官方化：要么推出**官方API替代**，要么通过**政策框定其合规形态**，既遏制了野生创新的无序扩张，也把真正有价值的成果纳入生态。

图 iOS生态与三方库的循环关系示意图

在**跨平台框架**方面，苹果官方始终未进行主动推广。相反，iOS更加强调原生开发，以最大限度保障性能、安全与系统一致性。但从底层基础设施层面来看，苹果**仍通过标准化的API和协议**，为跨平台兼容性提供了良好的支持基础。

例如，跨平台框架Weex可通过编译生成调用SwiftUI的原生UI组件，从而在iOS端实现原生渲染效果。自动化测试方面，苹果也提供了UIAutomation等官方工具，支持第三方跨平台测试框架的接入。典型移动端自动化测试框架Appium通过WebDriver协议与UIAutomation交互，进而实现对iOS应用的自动化测试支持。

关键观点 **iOS在第三方库与跨平台框架上的核心策略是原生优先：通过官方工具与标准化接口管理并支持第三方，但始终把性能、安全与体验优势牢牢绑定在原生体系中。**

针对**多设备开发**场景，苹果自2019年起推出「Project Catalyst」，提供一系列工具包，便于开发者将iPad应用快速移植至macOS平台。苹果在软硬件协同层面也持续推进统一化架构建设。

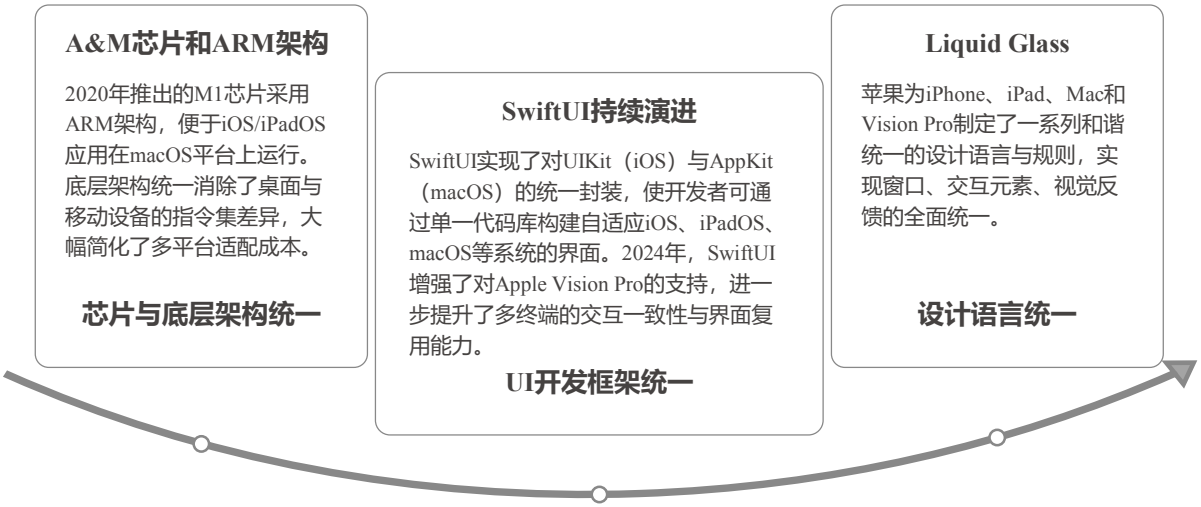


图 iOS多设备开发生态体系示意图

整体来看，iOS官方在多设备开发方面主要在平台适配和应用移植成本上下功夫，在操作系统层面仍然尽可能地保持系统间的独立，但官方在多设备统一体验上的投入正推动iOS开发生态向更加融合与智能化的方向演进。

在**Web开发**方面，iOS将WKWebView作为官方Web容器，以提供更高效的渲染性能、更高效的多进程隔离和内存管理以及更安全的沙箱隔离。苹果已经不再接收含原UIWebView API的新App送审，但苹果官方提供了WKWebViewConfiguration统一配置项，支持无缝迁移UIWebView的缓存策略。苹果围绕Web的开发能力建设主要体现在以下几个方面：

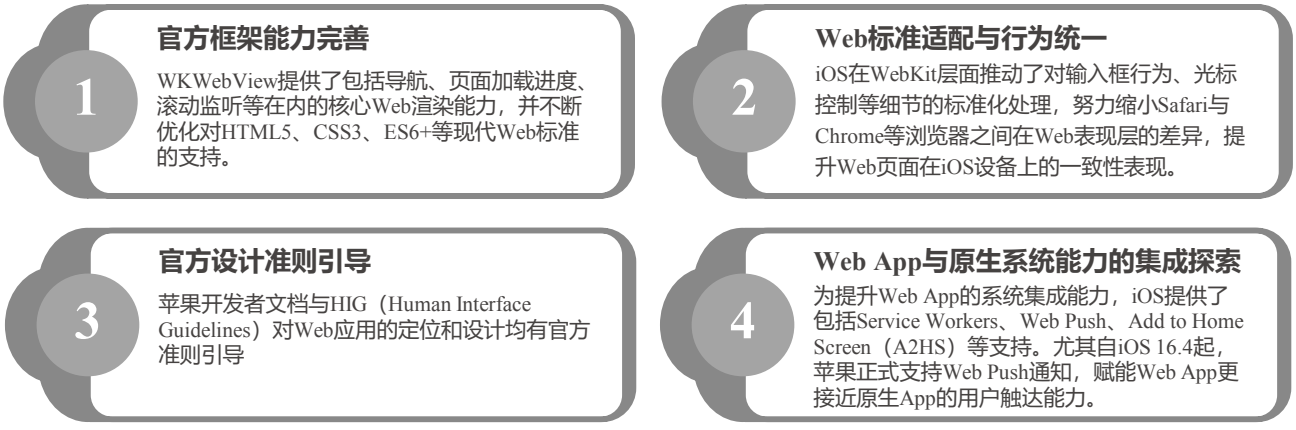


图 iOS Web开发生态体系示意图

Android秉持开放、多元的发展路径，鼓励生态多样性与社区参与。近年来，官方逐步通过标准化工具链、设计体系和平台能力，推动第三方库、跨平台框架和多设备开发生态的融合与现代化。尽管硬件形态和设备差异带来适配挑战，Android正在以渐进式方式构建统一、可扩展的跨终端开发体验。

在**三方库生态**方面，Android通过官方库推动、三方库管理生态和工具链支持，正逐步引导第三方库向现代化结构演进，降低版本冲突和碎片化问题，激发三方库活力。

1 通过AndroidX与Jetpack架构推动生态标准化

AndroidX建立了统一的命名规范与模块结构，第三方库可基于其构建并自动继承系统行为规范，显著降低了适配成本。Jetpack进一步提出架构范式，鼓励第三方库围绕生命周期感知、数据驱动等设计理念构建更高内聚、低耦合的组件，推动生态趋于标准化与可组合化。

2 构建统一的第三方库管理生态

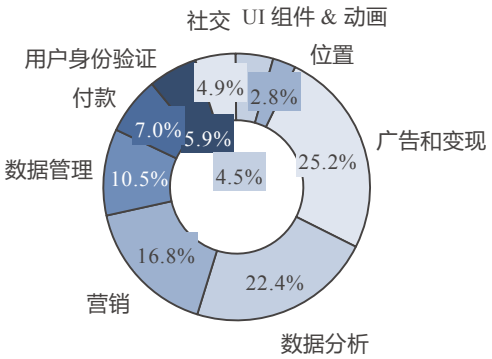
Android官方通过Gradle，配合Maven Central与Google Maven仓库，形成了标准化的第三方库依赖管理机制，解决了库管理的碎片化问题。通过版本控制、依赖自动解析等能力，降低了集成复杂度，提升了项目的可维护性。

3 开发工具链对第三方库的全面支持

Android Studio提供依赖分析器、库冲突检测器等工具，帮助开发者有效识别并解决依赖冲突、重复引用等问题。在测试和打包阶段，官方测试框架（如Espresso、Robolectric）支持第三方库集成测试，打包环节也支持混淆配置等定向优化，保障第三方库的兼容性与性能表现。

图 Android三方库生态体系示意图

除此之外，Google Play的Data Safety表单同样要求开发者申报第三方SDK的数据收集，但Google提供了Google Play SDK Console，部分常用SDK由Google直接维护合规说明，相对来说减轻了开发者负担。



- AdFit Android SDK
- Meta Audience Network
- Unity Ads
- Datadog SDK for Android
- Firebase Crashlytics
- Singular SDK
- Realm Java
- Usercentrics UI
- NAVER Map SDK
- Tapjoy Android SDK
- Paytm All-in-One Payments SDK
- Stripe Android SDK
- Facebook Android SDK
- WeChat SDK for Android
- LINE SDK
- Zendesk Classic Support

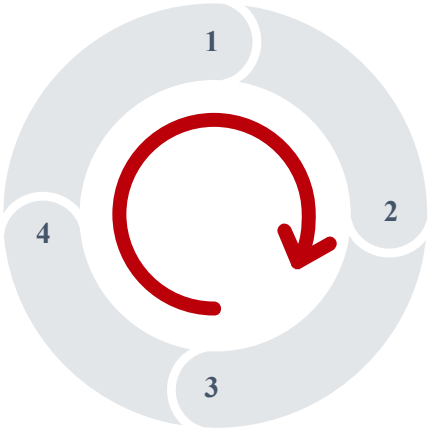
图 Google Play SDK Console官网公布的三方SDK分布和部分三方SDK示意图　数据来源：官网

开放带来繁荣三方库生态

Android基于开源（AOSP），提供全面接口和底层可接触性，功能覆盖范围广，Gradle/Maven降低集成成本，开发者有较高的自由度，让三方库得以快速涌现和传播，为不同层次的开发者提供了解决方案。

治理催生新一轮开放

因为Android提供了合规边界和更安全的技术底座，开发者反而能在更清晰的规则框架下继续探索。
新的功能需求不断涌现，三方库依旧扮演重要角色，推动Android生态进入下一轮开放与繁荣。



高度开放

高自由度带来了广告、支付、UI等三方库的繁荣，同时在隐私、传播等可能存在**隐患**。
治理动因包括维持生态稳定，法律合规和商业利益。

规则治理与技术托底

一方面通过**Data Safety表单、SDK Console和SDK Index等机制**，要求SDK提供方披露数据收集与风险信息，提升生态透明度

另一方面通过**Jetpack官方库和Play Services等基础组件**替代不稳定的三方方案，提供更安全的技术托底，防止生态失控。

图 Android生态与三方库的循环关系示意图

在跨平台框架方面，Android官方通过技术合作或协作的方式探索跨平台开发生态，并通过与官方开发体系，尤其是Android Studio的整合，提升开发者生产效率。

在Android官方体系内，**Flutter与Kotlin Multiplatform互为补充。**Flutter强调多平台一致的UI与高效开发体验，吸引需要iOS/Android/Web/桌面统一UI的团队，而Kotlin Multiplatform则用于跨平台共享逻辑（数据层、网络层、业务逻辑），不同平台的UI仍需分平台实现。

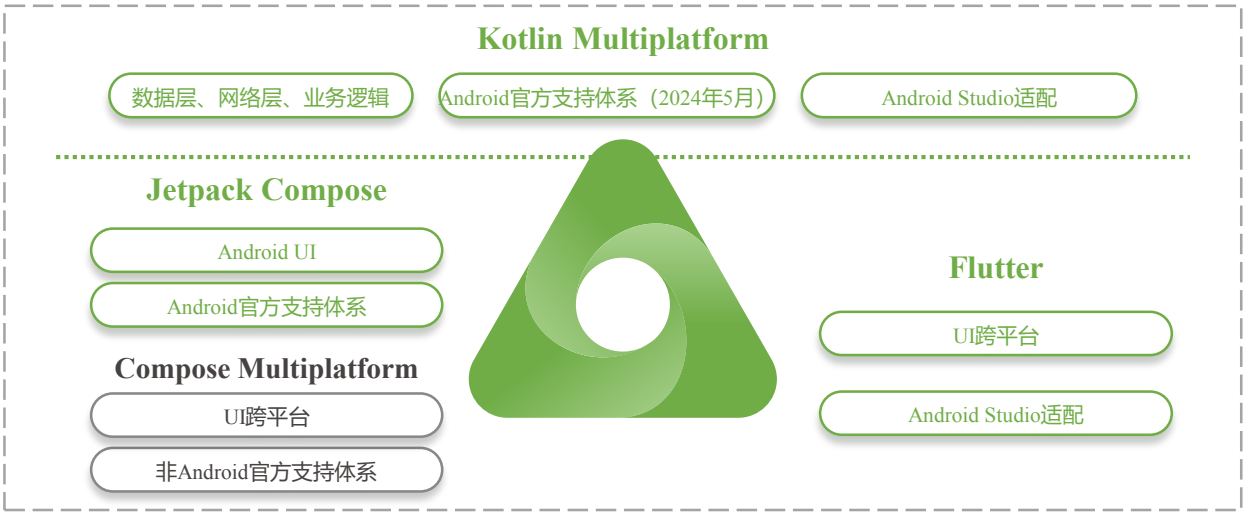


图 Android跨平台开发生态体系示意图

关键观点

Android在三方库与跨平台框架的核心策略是开放兼容：官方通过规则治理和官方库托底保证三方库，Flutter/Kotlin Multiplatform兼顾UI与业务逻辑的跨平台统一。

Android官方已尝试构建多设备协同生态，各项功能规划相对完整，落地效果则仍在逐步推进中，开发者在多终端协同体验构建上可能仍需克服一定的复杂性与兼容性挑战。

图 Android多设备开发生态示意图

官方库推动多形态屏幕适配统一化	提供SDK构建设备间协同能力	构建统一设计规范与UI适配体系	Android Studio提供多设备预览与测试支持
为应对折叠屏、平板等多样化屏幕形态，推出Jetpack WindowManager库，用于统一管理尺寸变化、折叠状态、双屏分布等。	自2022年起，Android官方推出Cross Device SDK，期望支持手机、平板、手表、TV等多设备之间的初步协作。	持续推进Material Design的多设备适配能力，发布了专门针对大屏和折叠屏设备的设计规范（如NavigationRail、Two-Pane Layout等）	Android Studio不断优化多设备开发支持，已提供多种虚拟设备配置、布局验证工具和屏幕适配预览能力。
该库降低了多屏适配的复杂度，是官方在多形态设备适配方面的重要基础设施之一。	设备发现、连接等基础功能已上线	同Jetpack Compose的响应式布局框架一期，提升了在不同屏幕尺寸上的UI一致性与开发效率。	从模拟器角度，提供了多种设备适配和预览工具
但受限于Android设备生态的分散，目前对于多形态屏幕的适配仍处于早期构建阶段	但目前该SDK功能仍较基础，高级功能如远程控制尚未正式开放，且依赖于预装Google Play服务，支持设备范围有限。	从官方指导到实际设备显示存在一定滞后性	但在折叠屏和跨设备协同等复杂场景下，仍存在交互模拟不完整、测试支持不足等问题，有待进一步完善。

Web开发方面，Android从底层引擎、开发工具和应用形态上持续演进。

1. 内置Chromium引擎作为系统WebView渲染核心

Android系统的WebView的底层渲染引擎，统一了各厂商设备的Web页面呈现体验，并在后续版本中实现WebView的独立更新机制，用户无需系统升级即可获取最新Web特性与安全修复。

2. 提供WebView开发、预览和调试接口

官方持续完善WebView API和WebViewCompat库，支持加载本地或远程网页、以及Cookie、缓存、缩放等配置，Android还为WebView提供远程调试能力，Android Studio支持在布局编辑器中嵌入WebView，开发者可便捷集成Web内容，并借助调试工具查看页面加载性能与交互逻辑。

3. 推动渐进式Web应用（PWA）在Android上的标准化支持

通过与Chrome浏览器深度集成，Android支持将符合标准的PWA安装为“类原生”应用，具备离线能力、通知推送、主屏图标等特性。用户可从浏览器直接添加至主屏幕，提升Web App的使用便捷性和系统集成度。

1.2 核心观点总结

iOS开发体系梳理：

阶段一：iOS在早期通过Objective-C、Xcode与iPhone SDK奠定了完整开发生态的基础框架。

阶段二：iOS通过原生语言、框架等核心系统能力的构建，减少了对三方API的依赖，奠定了原生开发生态的基础设施。

阶段三：iOS开发体系通过多终端适配与智能服务框架的发布，迈向智能化原生能力与跨端体验的一体化升级阶段。

Android开发体系梳理：

阶段一：Android通过Java语言、SDK/NDK与Eclipse构建早期开发生态，奠定开放式生态基础。

阶段二：Android通过Android Studio、Kotlin与Jetpack构建现代开发栈，为碎片化设备提供统一指导。

阶段三：Android通过多终端扩展、跨平台框架与AI开发工具迈向智能化，但开源更新放缓或影响第三方生态活力。

HarmonyOS开发体系梳理：

阶段一：HarmonyOS通过ArkTS、ArkUI与DevEco Studio构建基础开发体系，奠定多设备协同的开发能力。

阶段二：HarmonyOS以多语言体系、原生AI能力与智能工具链，快速迈向全场景融合与智能驱动的开发体系。

移动生态技术栈转变路径分析

官方开发体系的转变，不仅是为了突破提升旧语言与框架的效率，更是出于强化生态掌控力与应对跨平台冲击的战略考量。

新开发体系应执行两步走战略：先以工具与支持降低迁移成本，再以独有能力激励创新，实现开发者生态由迁移驱动向创新驱动的转变。

生态体系分析

iOS在第三方库与跨平台框架上的核心策略是原生优先：通过官方工具与标准化接口管理并支持第三方，但始终把性能、安全与体验优势牢牢绑定在原生体系中。

Android在三方库与跨平台框架上的核心策略是开放兼容：官方通过规则治理和官方库托底保证三方库，Compose Multiplatform/Kotlin Multiplatform兼顾UI与业务逻辑的跨平台统一。

iOS官方和Android官方均**未针对第三方库或跨平台框架推出直接资金激励，其激励计划更多直接针对优秀应用进行奖彰和中小开发者进行费用减免和资源帮助。**

在多设备开发和Web开发方面，iOS和Android均通过统一UI框架、官方SDK与工具链支持、多设备预览以及系统集成能力，提升适配效率、界面一致性和Web应用的原生化体验。

1.3 开发者画像分析

除了对开发体系和生态体系的分析外，不同平台开发者的画像同样值得关注。

从iOS开发者对于编程语言的使用选择来看，即使在iOS开发者群体中，将Swift作为主要编程语言的人数占比并不算高，这可能意味着大量开发者的语言技能背景是跨平台或非原生的，这也同JavaScript/TypeScript语言使用率最高的事实相互印证。除此之外，后端或服务端语言使用率也较高，说明iOS开发者群体中有大量全栈或服务端背景的人，他们不仅写客户端，还需要处理后端逻辑、脚本和数据库。

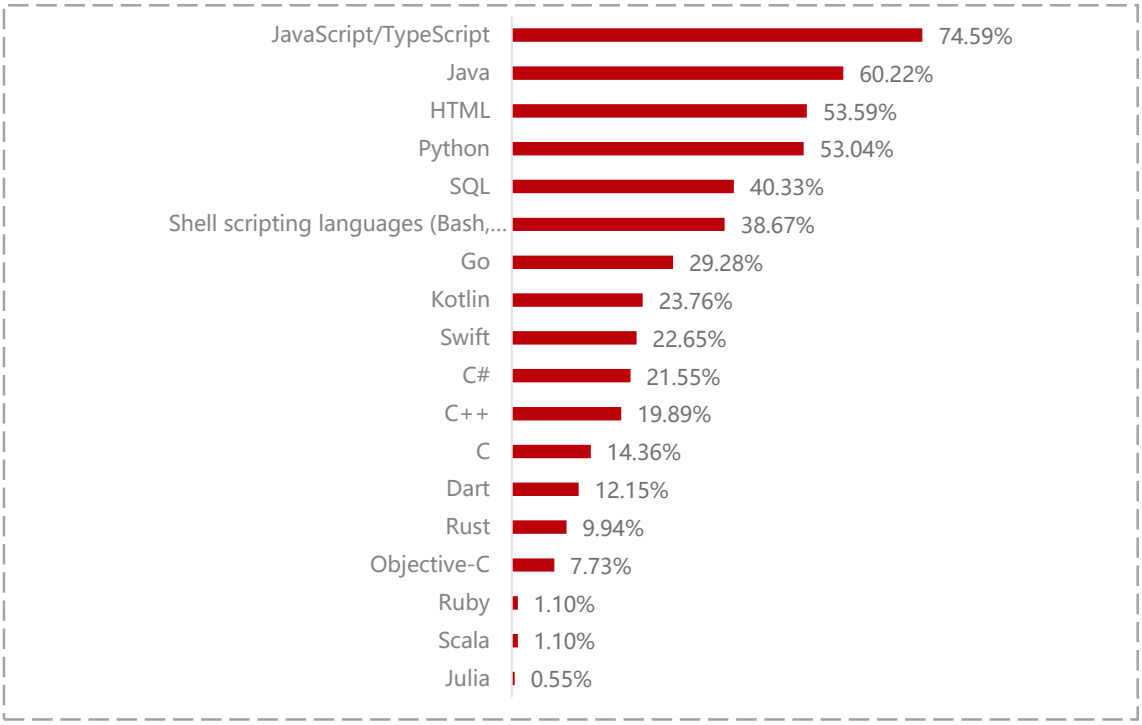
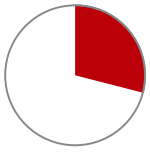


图 受访iOS开发者编程语言的使用情况（N=181）

这一趋势也与工具使用情况相吻合。28.73%的使用率说明大部分iOS应用开发也并不是通过Xcode的原生方式完成。这进一步揭示了iOS开发者群体中非原生化的明显特征。



28.73%的iOS开发者使用Xcode进行移动应用开发

从Android开发者的使用情况来看，可以发现Java依旧是核心语言，使用率高达69.18%，明显高于Kotlin的28.85%，说明尽管Kotlin已成为官方推荐语言，但尚未完全取代Java，整个生态仍处在语言迁移的过渡阶段。

相比之下，Kotlin的普及度高于iOS生态中的Swift（22.65%），这可能反映出Google在语言推广上的成效更显著。

与此同时，同iOS开发者展现的特点一致，Android开发者对JavaScript/TypeScript（72.46%）、HTML（60.66%）和Dart（12.79%）等跨平台技术的使用率同样很高，显示出跨平台开发已经成为常态；而Python（54.10%）、SQL（45.90%）、Go（26.89%）等语言的高占比，则表明Android开发者往往不仅仅聚焦移动端，而是普遍具备服务端与全栈开发的能力。

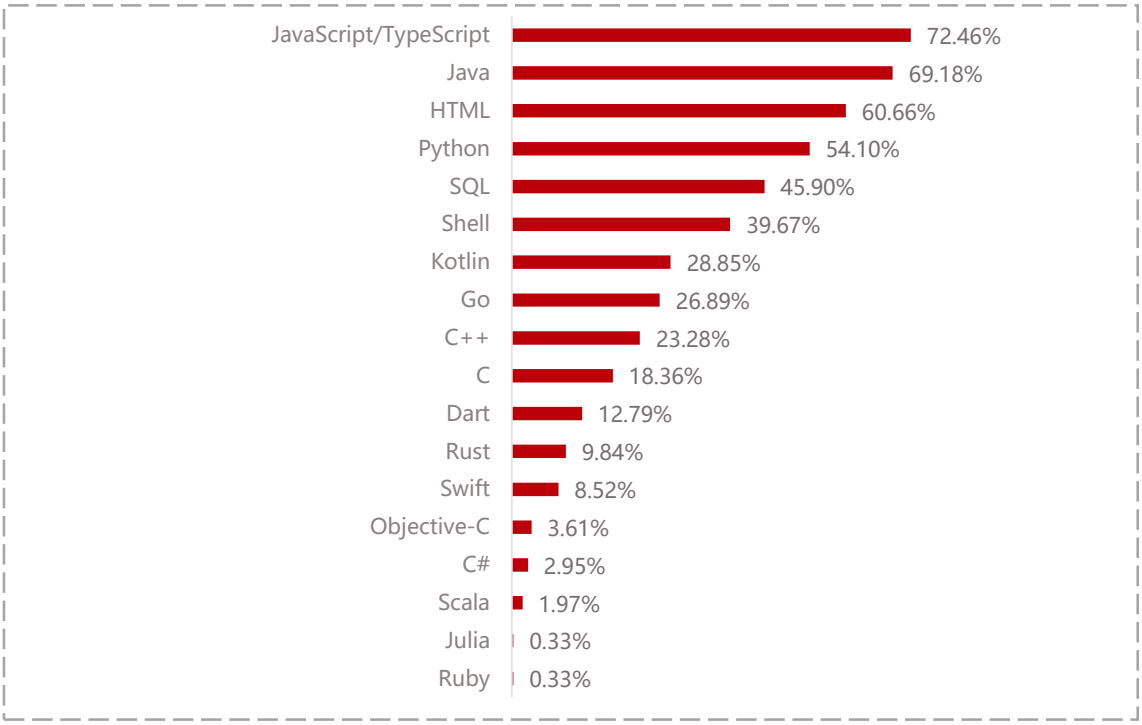


图 受访Android开发者编程语言的使用情况（N=305）

工具的使用情况也进一步印证了这种趋势。数据显示，只有29.84%的Android开发者主要使用Android Studio进行移动应用开发，这意味着相当一部分开发任务依赖跨平台框架或替代性工具完成，强化了非原生化开发的生态特征。

29.84%的Android开发者使用Android Studio进行移动应用开发

从HarmonyOS开发者的使用习惯来看，除了与iOS和Android开发者的一些共性特点外，其语言选择和开发工具呈现出明显的生态策略特征。

官方语言ArkTS的使用率达到32.63%，远高于其他两个生态的原生/非原生编程语言。这表明官方原生语言在短时间内获得了较高的接受度，其基于TypeScript的语法扩展降低了前端和跨平台开发者进入原生开发的门槛。

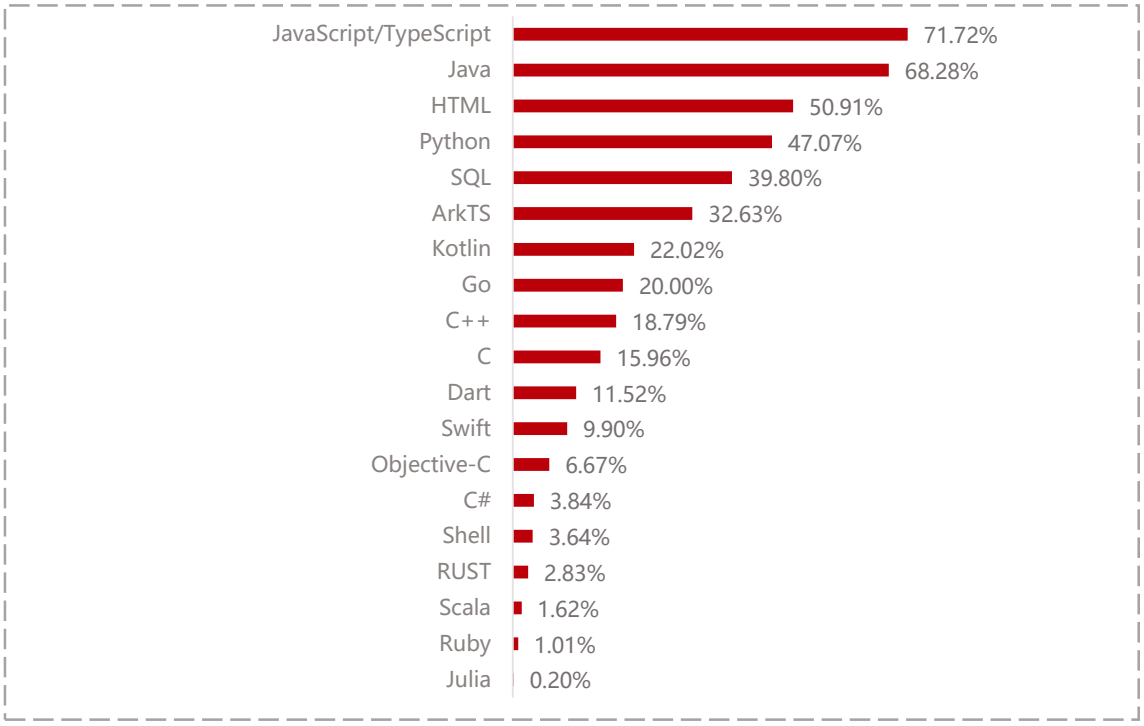


图 受访HarmonyOS开发者编程语言的使用情况（N=495）

与此同时，**DevEco Studio的高使用率反映了官方工具在HarmonyOS生态中对开发者的吸引力**，官方在原生开发者培养和生态建设上的策略非常有效。

62.67%的HarmonyOS开发者使用DevEco Studio进行移动应用开发

在编程语言使用方面，JavaScript/TypeScript在移动应用开发者中使用率位居首位，这一趋势在iOS、Android和HarmonyOS生态中均得到体现，显示出跨平台开发和全栈能力正在成为移动开发的共性需求。

Java的高使用率与我们前文对iOS和Android编程语言体系演变的分析相呼应，这并非单纯由于语言本身的现代化特性，而更多反映了各生态对其开发者社区、工具链和整体生态的掌控能力。

值得关注的是，鸿蒙官方配套语言ArkTS的使用率整体已达到32.63%，显示出官方语言正在形成可观的开发者基础，并处于加速扩散阶段。

究其原因，这些高使用率的语言深刻契合了现代移动开发的核心诉求：高效构建与广泛覆盖。JavaScript/TypeScript凭借其稳固的开源生态系统和良好的跨平台特性，显著提升了移动应用开发的效率与成本效益。这使其成为构建高性能跨平台移动应用的热门选择；Java凭借成熟的JVM跨平台特性及在Android平台长期积累的核心地位与强大生态（如Android SDK、Jetpack库），为开发高性能、功能丰富的原生Android应用提供了坚实、可靠的基础。总体而言，JavaScript/TypeScript和Java的高使用率表明，它们在**生态系统的成熟度、跨平台兼容性以及对主流移动开发实践的深度适配**，满足了开发者的核心需求。

关键观点

移动开发者使用原生开发能力与跨平台开发能力，显示现代移动开发已从单一生态和原生语言依赖，转向以开发效率、应用覆盖度和多端协作为核心驱动的新阶段。

这种语言选择趋势也映射到开放框架上：除了iOS、Android和HarmonyOS的原生开发外，跨平台框架已成为移动开发的重要方式。在开发框架的选择上，移动应用开发者呈现出明显的倾向性，近半数（47.81%）开发者更倾向于直接采用系统级原生框架进行应用开发，以获得对硬件特性的深度适配和极致的性能调优能力。

与此同时，跨平台框架的使用也在不断扩大，它允许开发者以统一代码基础覆盖多平台和多设备，加快开发进度、降低维护成本，并支持前端、后端及多端协作能力的整合。

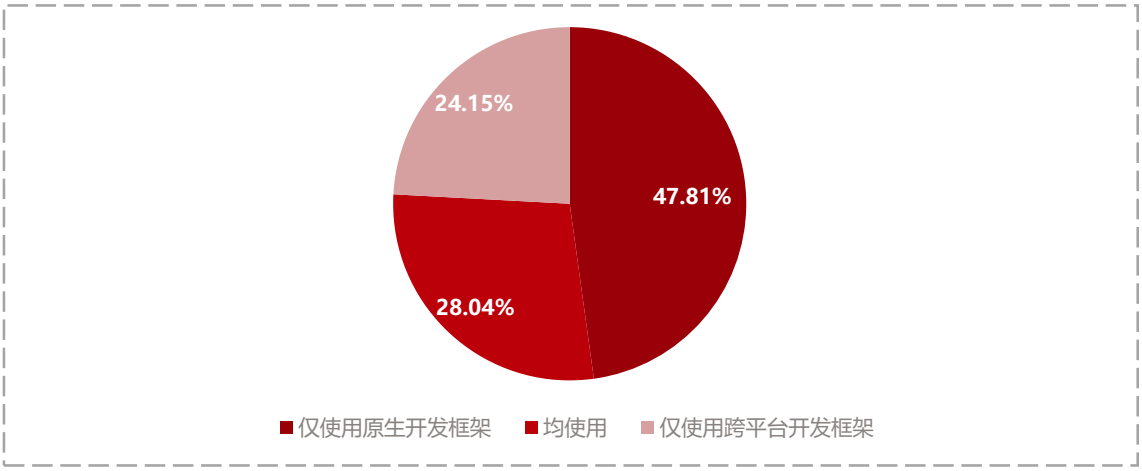


图 受访开发者进行移动开发时的框架选择（N=617）

在跨平台框架的选择中，开发者最常使用Uni-app和Flutter，同时两个框架的使用者呈现出一定的差异特性：5000人以上团队规模的开发者对Flutter有着最高的使用率，50人以下团队规模的受访开发者则更倾向于使用Uni-app进行跨端开发。

这种分化源于核心特性契合不同规模团队的需求：Uni-app的核心竞争力在于对小程序生态的深度整合，前端开发者通过Vue.js可无缝生成iOS、Android、H5及多平台小程序，极大降低了小团队的技术迁移与多端适配成本；Flutter的关键优势则体现在高性能渲染引擎（如Skia自绘架构）与像素级一致性控制，精准匹配大型企业应对复杂业务场景时所需的跨端治理能力与性能稳定性。

这也意味着，对于跨平台框架而言，跨平台框架必须将**持续优化跨端一致性**作为发展的核心任务，这不仅包括跨设备的兼容性，也包括跨端调试以及用户体验的一致性。同时也要保持一定的**易用性**，避免给开发者带来额外的学习门槛。

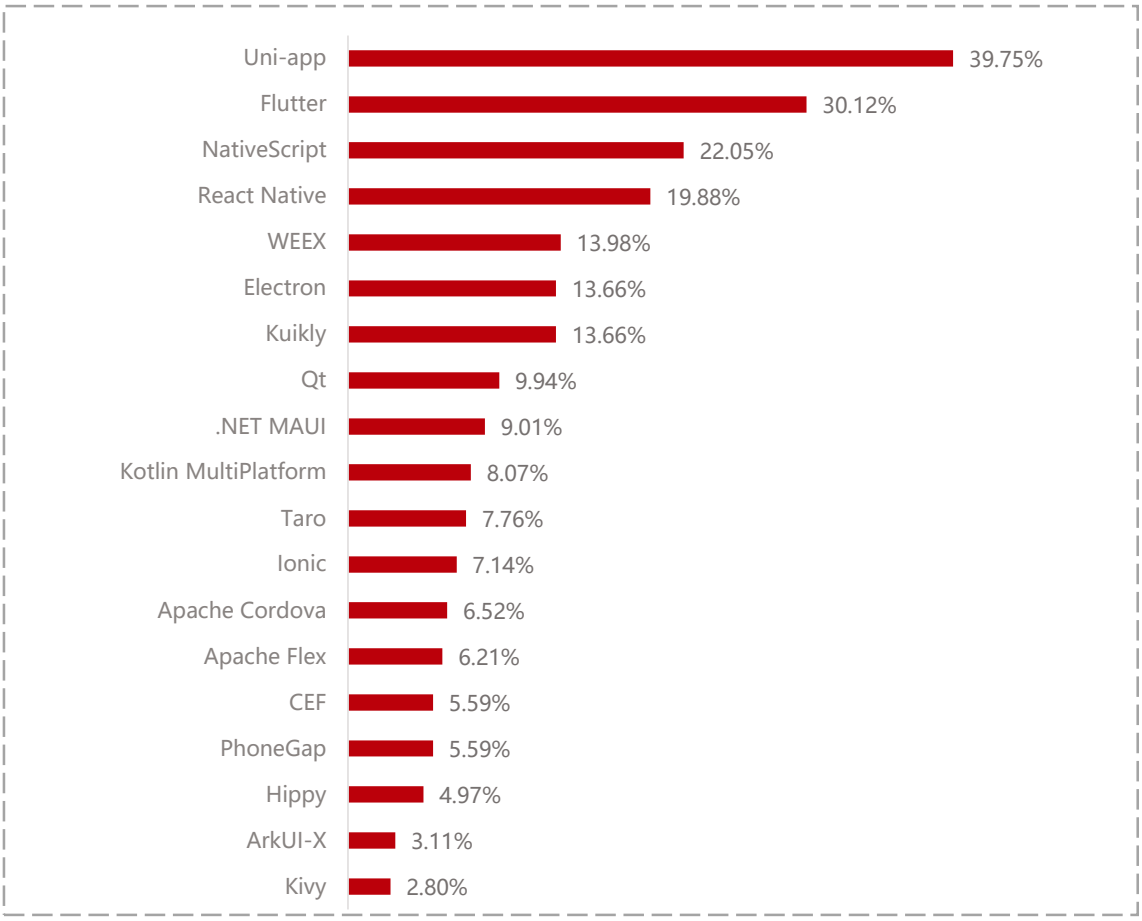


图 受访开发者跨平台开发框架的使用情况（N=322）

关键观点

移动开发正在呈现双轨发展趋势：近半数开发者偏向原生框架以追求极致性能，而跨平台框架则因团队规模和业务需求差异分化使用，推动跨端一致性和易用性成为未来技术竞争核心。

开发者对开发体验的核心关注点集中在技术基建的完备性层面。根据调研结果，关键因素前三项分别是开发工具链完善程度、第三方资源丰富性和语言及应用框架成熟度，这反映出开发者普遍期望获得体系化的技术支撑，以降低基础环境搭建和软件开发的门槛。工具链的成熟度直接影响开发效率，也是技术生态健康度的直观体现；丰富的第三方资源以及成熟的语言和框架，则决定了快速实现业务逻辑的可行性。

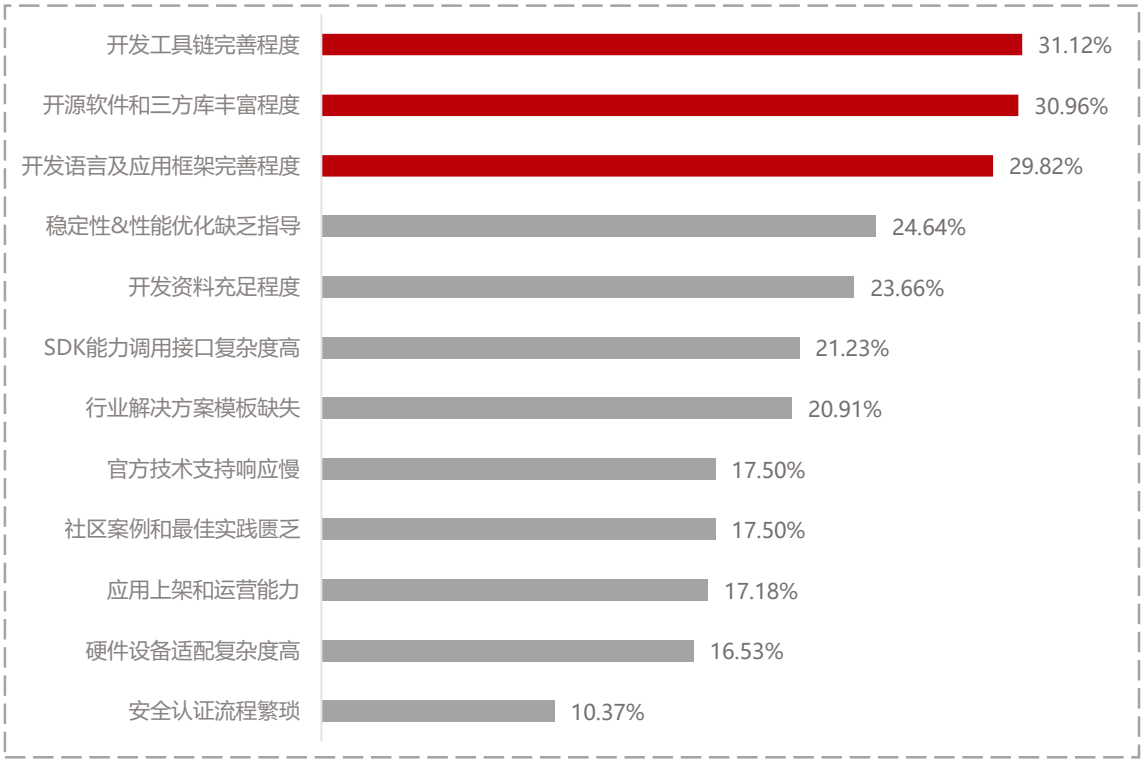


图 受访开发者选择影响开发体验的Top3因素（N=617）

这也意味着，开发生态需要把“工具链—资源—框架”视为同等重要的底座，持续加大投入：一方面由官方打造开箱即用的开发环境和稳定易用的开发工具链；另一方面通过资金激励、流量扶持及严格的质量评审机制，引入并沉淀高质量的第三方组件与服务，并保持核心语言和框架的迭代周期与兼容性，才能形成正向循环，巩固并放大平台对开发者的长期吸引力。

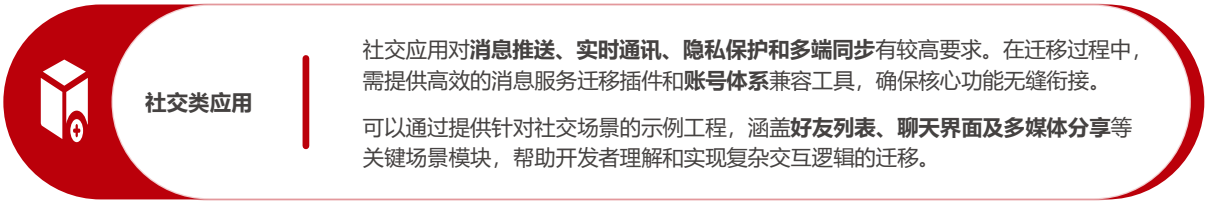
关键观点 | **无论建立原生开发体系还是激励三方扩展体系，开发者体验的核心在于“工具链—资源—框架”三位一体的完备支撑。**

此外，不同类型的应用在开发与迁移过程中呈现出差异化的技术诉求和适配偏好。平台在制定迁移工具、提供能力支持与后续激励策略时，亦需因类施策、有的放矢。

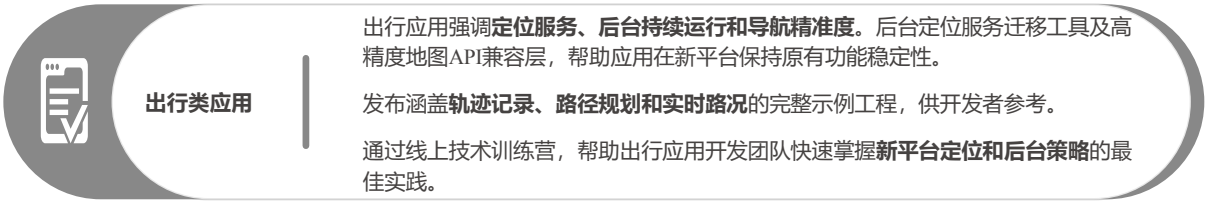
以社交类应用为例，前期开发选择的核心因素是**用户策略**和**盈利模式**：

- **用户策略**：定制化→首选iOS；追求快速覆盖→首选Android/跨平台开发。
- **盈利模式**：以内购/订阅为主→首选iOS；以广告变现为主→首选Android。

除此之外，社交的独特属性也需要考虑在内，双端覆盖有利于保持社交链完整，否则可能影响活跃和留存，然后才需要考虑良好用户体验的**技术实现便利性。**



对于出行类应用而言，除了像社交类应用一样需要重点考虑**用户策略和盈利模式**（例如定制化服务考虑iOS原生开发，大众出行/快速扩张市场则优先考虑Android开发）外，还高度依赖**定位、地图渲染、路径规划和调度算法**，并可能需要**访问传感器、蓝牙、支付系统、摄像头扫码**等硬件能力。因此，**原生开发**更容易充分利用系统级能力，在这些场景下通常优先考虑原生开发。

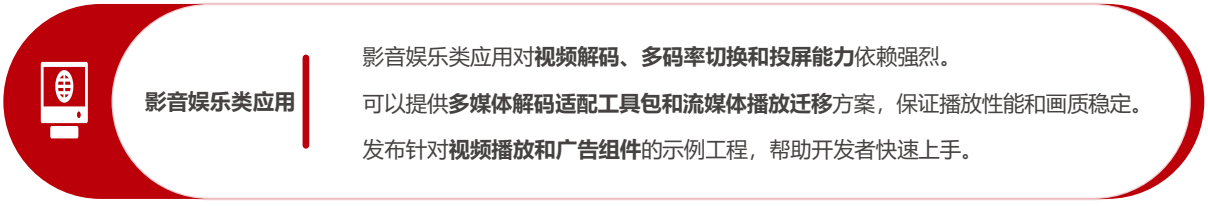


在影音娱乐类应用中，平台选择的首要影响因素是**是否依赖高性能影音处理。**

重度影音处理（如长视频流媒体、4K/8K播放、实时直播、复杂音效处理）→原生开发优先，以保障播放流畅度、性能优化和版权安全。

轻量内容消费（如播客、博客、音乐播放器或简单短视频工具）→跨平台开发可行，以降低成本、加快迭代。

在此基础上，再结合用户策略（定制化 vs 大众覆盖）和盈利模式（订阅/内购vs广告）来决定优先平台（iOS或Android）。



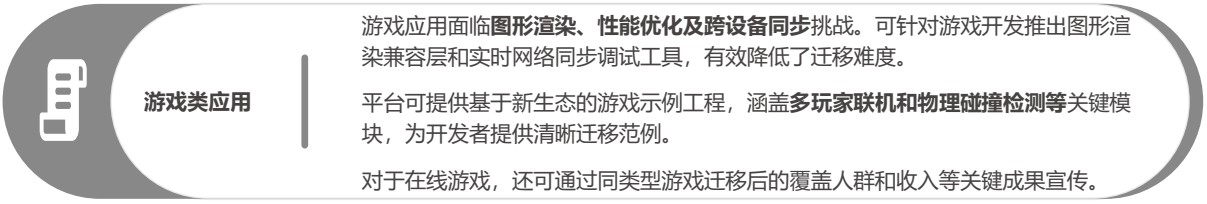
同影音娱乐类应用相似，游戏类应用的首要影响因素也是性能与体验，相较于平台的选择，游戏引擎的选择更为关键。

中大型3D游戏/高帧率动作游戏→Unity/Unreal，包括部分厂商会选择自研引擎，因为对GPU/CPU、渲染优化、内存管理要求极高。

轻量休闲游戏/2D游戏→可考虑 Cocos Creator、Cocos 2d-x、LayaAir等游戏引擎，快速上线、多端覆盖，甚至可以通过小程序完成轻量级开发。

目前大多数游戏引擎均已实现了对iOS和Android双平台的支持，同时iOS和Android也对游戏的移植和适配提供了工具或指南，对于使用自研引擎的游戏，也提供了参考模板和指南。

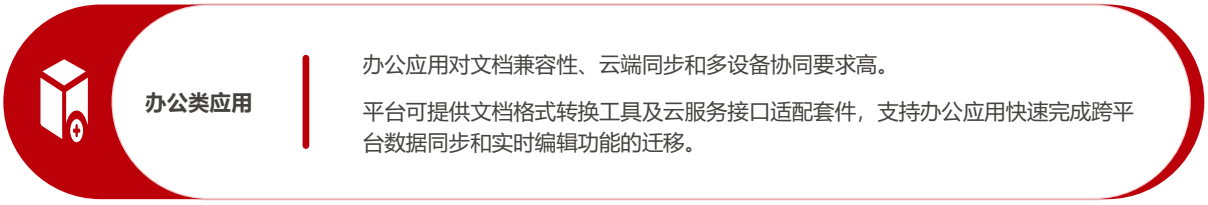
其次才是对于**用户策略和盈利模式**的考量。



至于办公类应用，首要考虑的是在iOS/Android/Web/桌面之间无缝协作（如文档、表格、会议、IM协作），因此会优先考虑**跨平台开发**。

在**功能层面**，若涉及复杂表格运算、实时音视频会议或文件加解密，则需要依赖原生开发以保证性能，而轻量级功能如任务清单、日历或审批流程则可通过跨平台方案快速实现。

整体来看，办公类应用天然要求双端+Web全面覆盖，常采用**跨平台为主、关键功能原生优化**的混合策略。



不同类型的应用在平台选择与迁移时，核心考量点有所差异：**社交类首要评估用户策略与商业模式，出行类首要考量对系统级硬件与定位能力的依赖，影音娱乐与游戏类看重性能与体验要求，办公类则以跨平台一致性与协作为首要诉求。**因此平台在制定迁移与支持策略时，应因类施策、有的放矢。

1.3 核心观点总结

移动开发者画像呈现新变化，显示现代移动开发从单一生态和原生语言依赖，转向以开发效率、应用覆盖度和多端协作为核心驱动的新特性。

iOS：iOS开发者技能多样，原生Swift使用率不高，跨平台和全栈能力普遍，非原生化开发明显。

Android：Android开发者以Java为主，Kotlin语言普及过渡中，同时广泛掌握跨平台与服务端技术，非原生开发已成为常态。

HarmonyOS：HarmonyOS开发者倾向官方原生ArkTS和DevEco Studio，原生开发采纳率高。

移动开发正在呈现双轨发展趋势：近半数开发者偏向原生工具以追求极致性能，而跨平台框架则因团队规模和业务需求差异分化使用，推动跨端一致性和易用性成为未来技术竞争核心。

无论建立原生开发体系还是激励三方扩展体系，开发者体验的核心在于“工具链—资源—框架”三位一体的完备支撑。

开发平台选择：不同类型的应用在平台选择与迁移时，核心考量点有所差异：社交类首要评估用户策略与商业模式，出行类首要考量对系统级硬件与定位能力的依赖，影音娱乐与游戏类看重性能与体验要求，办公类则以跨平台一致性与协作为首要诉求。因此平台在制定迁移与支持策略时，应因类施策、有的放矢。

1.4 移动开发技术及生态演进趋势

编程语言：现代化与生态扩张两大主线， Kotlin与Swift通过互操作性加速跨生态支持

从原因与历程来看，尽管Swift与Kotlin在具体演进路径上有所差异，但它们近年来的核心发展主线高度一致，都是持续推进语言现代化改进，并不断扩展语言生态。

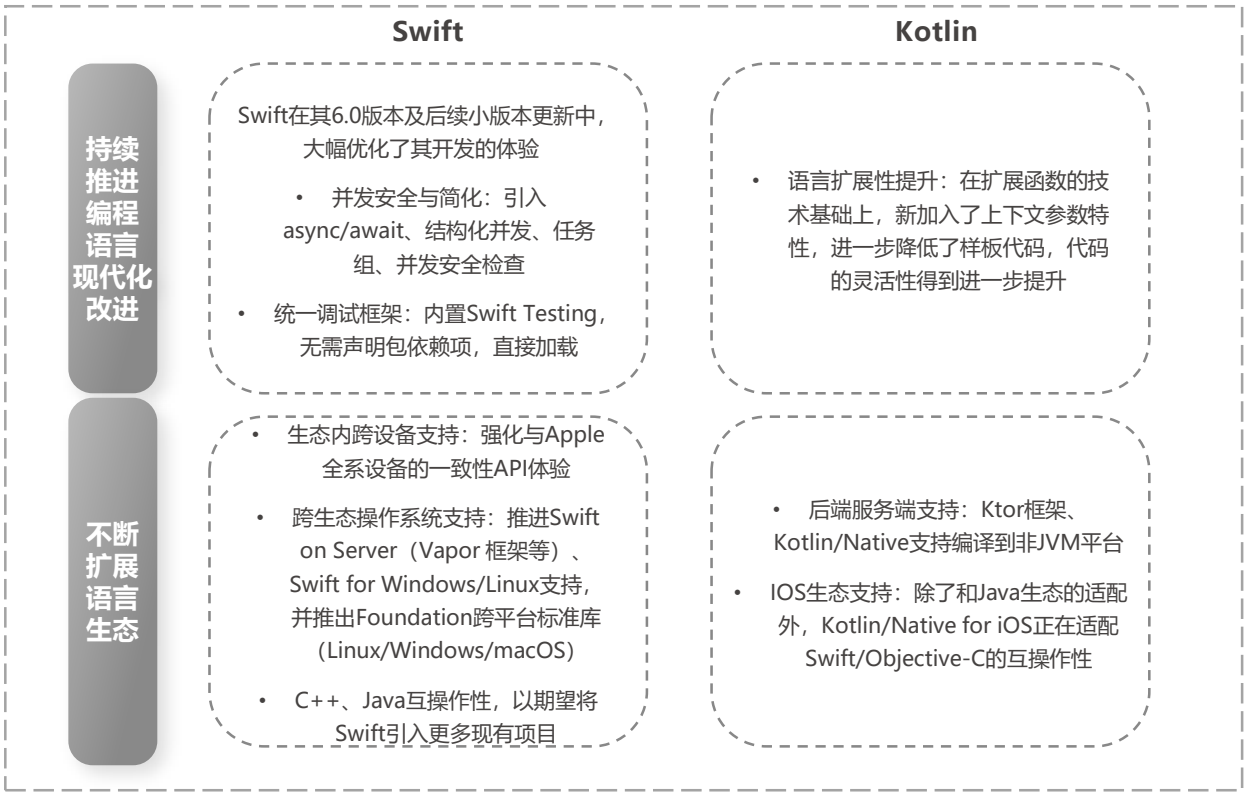


图 Swift和Kotlin近期演变

我们推测这两条主线仍然会是Swift和Kotlin的重要发展主线。

值得注意的是，2025年6月，Swift官方宣布成立Android工作组，标志着Swift将全面进入Android生态。目前Swift stdlib已能在Android上编译，Foundation等核心库的适配也在推进中。与Kotlin/Native for iOS对iOS的渗透相呼应，Swift与Kotlin正在跨越原有生态边界，形成正面交锋的趋势。

	当前重点方向	关键更新（NEW 表示新增）
语言演进	提升数据处理效率、增加抽象能力、以清晰代码提升性能	查看完整语言功能与提案（YouTrack）
编译器	Kotlin/Wasm 推进、弃用旧编译器	NEW 完成JSpecify支持 NEW 弃用K1编译器 NEW Kotlin/Wasm（wasm-js 目标）升至BetaKotlin/Wasm 切换 wasm-wasi到WASI Preview 2 Kotlin/Wasm支持Component Model
多平台	Swift Export、多平台库分发格式升级	首次发布Swift Export 默认启用并发标记-清除 GC稳定 klib跨平台编译 下一代多平台库分发格式 项目级多平台依赖声明 统一内联语义 默认开启klib增量编译
工具链	Kotlin/Wasm 开发体验优化、Gradle集成增强	NEW Kotlin/Wasm项目在IDEA中体验提升 NEW 导入性能优化 NEW XCFrameworks资源支持 NEW Kotlin Notebook体验改进IDEA K2模式全面发布 Build Tools API设计 Kotlin生态插件支持 Declarative GradleGradle项目隔离支持 Kotlin/Native与Gradle集成优化构建报告 改进Gradle DSL中公开稳定编译器参数Kotlin脚本与.gradle.kts体验优化
库生态	多平台标准库扩展、库稳定化	Dokka HTML UI优化返回非Unit值但未使用函数默认警告/错误多平台API支持 Unicode/codepointskotlinx-io稳定化Kotlin 发行版体验改进（覆盖率 & 兼容性验证） kotlinx-datetime升至Beta
Ktor	服务端框架能力拓展	NEW gRPC支持+生成器插件&教程 NEW 后端项目结构简化 NEW CLI生成器发布到SNAP NEW Kubernetes 生成器插件 NEW 简化依赖注入 NEW HTTP/3支持
Exposed	ORM框架稳定化与新特性	NEW 发布1.0.0 NEW 增加R2DBC支持

图 Kotlin技术路线图

开发框架：持续提升框架性能和体验，尝试引入AI开发

从2019年发布至今，SwiftUI与Jetpack Compose虽然在技术背景与实现细节上存在差异，但其发展路径高度相似，均可归纳为下图中的两大主线：框架能力现代化建设、生态与跨平台扩展。前者聚焦于性能与编译优化、布局与动画能力升级，以及更丰富的交互与视觉体验；后者则通过跨设备适配、跨框架互操作性与平台级功能整合，不断拓展框架的应用边界与生态覆盖面。

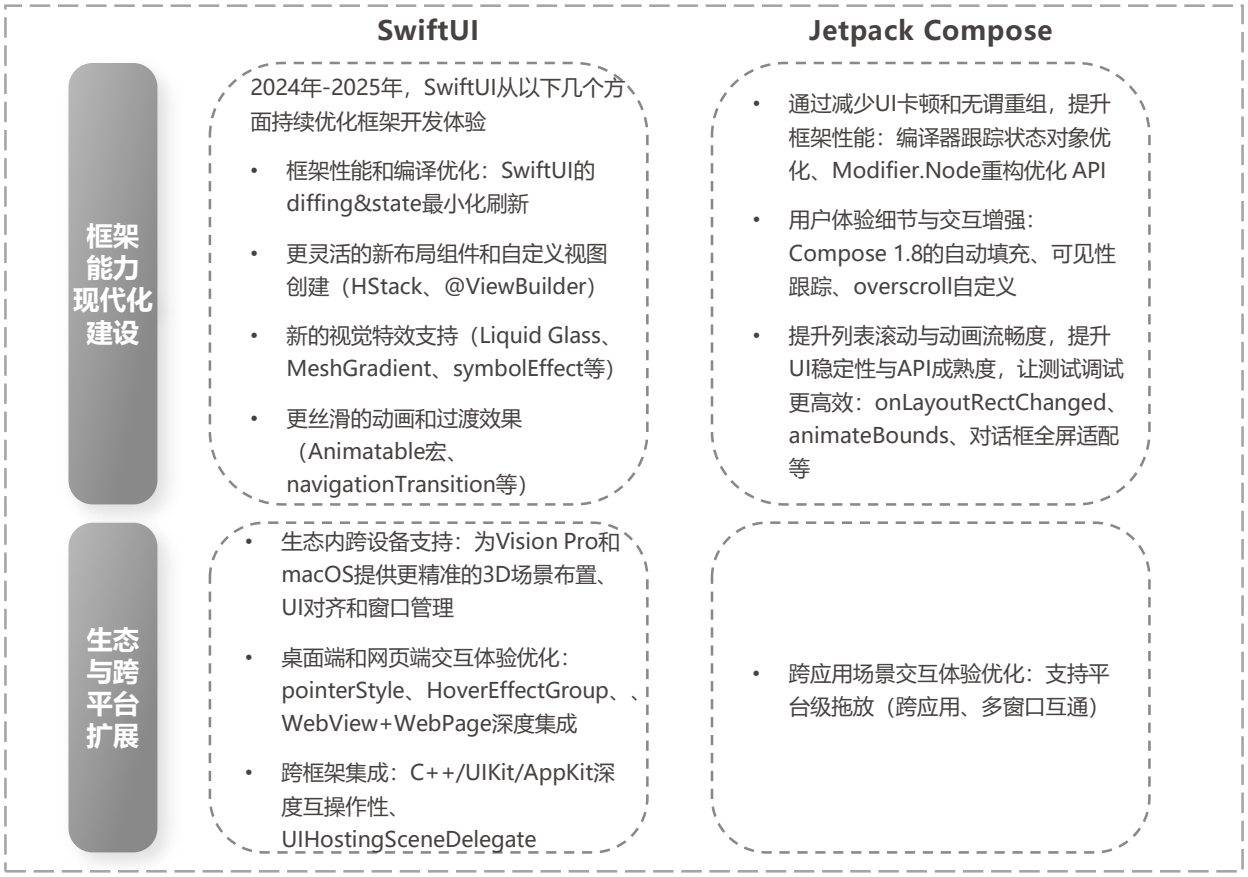


图 SwiftUI和JetPack Compose近期演变

基于此，SwiftUI和JetPack Compose在短期预计会继续延续两大主线思路进行演进。SwiftUI和Jetpack Compose将继续从以下两个方面推进框架能力现代化建设：

- **性能与多线程优化：**SwiftUI近期强化diffing/state刷新机制，Compose路线图中列出了多线程测量、多线程调度、SlotTable重写等，这预示UI渲染和编译器优化会更深入多核并行与内存复用。
- **更细腻的交互与动画能力：**SwiftUI已推出Liquid Glass、MeshGradient、Animatable宏等视觉与动画特性，Compose则计划引入共用元素过渡、LazyList动画和更高级动画导航，未来两者都会继续强化沉浸感与过渡体验。

多终端适配仍然会是SwiftUI和Jetpack Compose的重点发展目标，SwiftUI在手机、平板、电脑、Vision Pro等设备的布局优化、3D UI对齐等；Compose在路线图中有TV Compose、跨屏拖放、双窗格布局等计划，表明多终端UI形态适配是长期方向。

除此之外，Jetpack Compose已在技术路线图中将**生成式AI与界面开发**列为实验项目，预计SwiftUI也将在Xcode预览、代码生成、动态UI调整等方面引入Apple Intelligence大模型框架，推动智能化布局生成、代码补全与性能调优。

IDE：AI赋能是现阶段开发工具升级的探索重点，iOS强调可控辅助，Android积极探索全流程自动化

对于Xcode和Android Studio而言，AI是近两年推动IDE升级的关键动力。无论是在代码编写、开发体验提升、测试与调试环节，都能看到AI的身影。

在此过程中，Xcode更强调**多模型支持与用户对修改的可控性**，致力于为苹果生态开发者提供安全可靠的智能辅助；而Android Studio则依托Google Gemini模型，推动**AI从辅助向全流程自动化**演进，积极融合云服务和多设备测试，提升跨平台开发的协同效率。

	AI深度集成	开发体验提升	调试与分析	跨平台与生态支持
共性	IDE内嵌生成式AI提升编码效率、错误定位、代码生成 支持结合上下文进行修改与解释	启动、加载速度优化代码补全，编辑器功能增强 测试与自动化改进	调试工具与性能分析增强 崩溃分析中引入AI辅助修复建议	强化与自家生态深度整合
Xcode	Swift Assist支持多模型选则（第三方/苹果） 用户可手动确认修改，并且提供代码快照功能，可恢复之前版本的代码，更强调安全与可控	编辑器新增多词搜索、Swift语音输入 内联Playground实时代码运行 下载包体积缩小	调试和分析工具：Instruments提供硬件级分析（CPU分支追踪、功耗分析）、Organizer提供历史趋势分析 测试框架：通过Swift Testing，提供更简洁的测试定义和参数化测试，并发调试增强	聚焦Apple生态内设备、iOS/macOS/watchOS/tvOS
Android Studio	深度绑定 Google Gemini 2025 推出Agent Mode+Journeys，全流程自动化执行任务，相较苹果更加强调自动化操作	简化登录流程：简化Firebase/Gemini服务接入流程，统一Google登录	- Profiler+Firebase实时看板 - 云端崩溃分析与多设备实时对比	多设备适配支持：布局编辑器和Android Device Streaming，支持多设备同步预览，优化屏幕适配和云端物理设备测试应用 跨平台开发支持

图 Xcode和Android Studio近期演变

系统级AI能力：iOS App Intents关注系统无感融入，Android GenAI API更聚焦实际场景和功能

除了IDE层面的AI外，iOS和Android也通过提供相应的SDK/API，帮助开发者将AI能力引入应用。

在WWDC2024上，apple宣布开发者将可以通过**App Intents API**在自己的应用程序中引入Apple Intelligence，完成产品的智能化升级，接入到苹果的AI生态内。截至目前，App Intents API已经能够实现以下几项功能:

- **系统与Siri集成**：开发者可以通过App Intent Domains将应用与Siri和Apple Intelligence连接，使应用的功能可以被语音调用和智能建议触发。
- **内容发现与搜索优化**：通过IndexedEntity协议和相关Swift宏，应用实体能够被收录到Spotlight中，用户可快速搜索到应用内容。
- **视觉智能集成**：借助IntentValueQuery接口，应用可以将实体信息提供给系统，实现与视觉智能功能的互动。
- **多媒体与交互控制**：支持CameraCaptureIntent捕捉照片和视频、AudioRecordingIntent录制音频，以及ControlConfigurationIntent和WidgetKit在锁屏或控制中心放置控件。
- **内容共享与传输**：通过Transferable协议，应用可以共享或接收其他应用提供的数据，并通过IntentFile和FileEntity定义文件和内容存储方式。
- **高级意图管理**：支持URLRepresentableIntent/Entity/Enum实现深度链接，使用UnionValue宏定义多类型参数，以及通过UniqueAppEntity创建单例实体，为系统设置或应用内关键配置提供统一管理。
- **错误与确认处理**：通过AppIntentError和requestConfirmation API，可以向用户提供明确的操作指引和条件化确认，保证操作安全与体验一致性。

2025 Google I/O宣布ML Kit推出一组端上的GenAI API，以帮助开发者将Gemini Nano（端侧模型）集成到安卓应用中。新发布的4个API包括：

- **总结**：用于总结文章和对话。
- **校对**：用于润色短文本。
- **改写**：以不同风格改写文本。
- **图像描述**：为图像提供简短描述。

根据技术文档，目前GenAI API适用于谷歌Pixel 9、荣耀Magic 7、iQOO 13等22款机型。

1.4 核心观点总结

编程语言：现代化与生态扩张两大主线， Kotlin与Swift通过互操作性加速跨生态支持

开发框架：持续提升框架性能和体验，尝试引入AI开发

IDE：AI赋能是现阶段开发工具升级的探索重点，iOS强调可控辅助，Android积极探索全流程自动化

系统级AI能力：iOS App Intents关注系统无感融入，Android GenAI API更聚焦实际场景和功能

02

智能化时代
开发者机遇

2.1 智能化浪潮下的新趋势

随着本轮大模型技术浪潮的快速发展，智能应用正在逐渐探索从单一任务辅助向具备自主决策、情境理解和多模态交互能力的AI Agent的演变。这一趋势同时对终端硬件、端云协同和操作系统提出了新的要求。在此背景下，端云协同、端侧模型推理以及软硬件协同优化成为普遍关注的重点，为智能能力在手机、可穿戴设备、车机等多终端落地提供了可能性。

同时，各大操作系统通过原生AI能力的布局——包括模型集成、多模态支持及系统级API暴露——正在推动AI从附加功能向系统原生能力转变，从而构建智能操作系统的新生态格局。本节将从大模型技术突破、软硬协同以及操作系统智能化三个维度，系统分析智能化时代的技术演进趋势与核心动力。

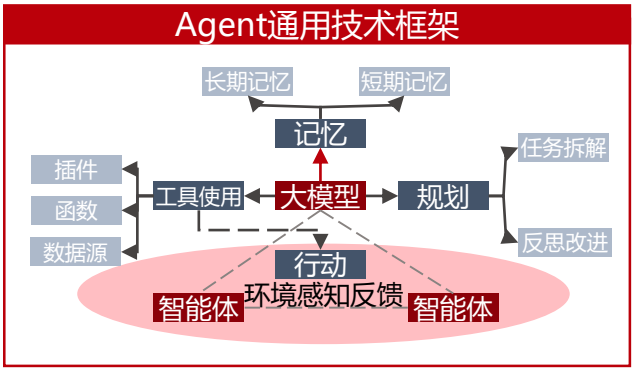
2.1.1 大模型突破推动AI Agent演化

2022年，图像生成模型和ChatGPT的发布，为本轮大模型浪潮拉开了序幕。此后，文本模型沿着长文本处理、对话智能化的方向持续演进，开源模型登上战场，推动了模型多样化和生态繁荣。图像、音频、视频生成模型等生成模型相继取得突破，形成了多模态内容快速累积的格局。

从2024年下半年开始，o1、DeepSeek-R1、k1.5等一系列**推理模型**的发布，则标志着文本模型正式掀开推理模型的序幕。而推理模型所展现出来的逻辑推理、规划与自我验证能力的提升，也成为AI Agent发展的关键拐点。

图 2017-2025年大模型技术演进重点趋势

从下图中的Agent通用技术框架可知，大模型是智能体的「大脑」，从一定程度上，模型的性能和表现直接决定了智能体的表现。这是因为，智能体在运行中必须依托模型来完成信息理解、任务规划与执行交互等关键环节。只有当大模型具备足够的理解力，才能正确解析并把握用户意图；只有当模型具备足够的推理与规划能力，智能体才能将复杂目标转化为可执行的行动路径；而在生成与交互环节，模型的表达准确性与连贯性又直接决定了智能体的执行效果。底层模型的迭代不仅是单点能力的增强，更在根本上决定了智能体的上限。



另智能体并不仅仅依赖于模型本身，还需要通过感知、记忆、规划、执行、反馈等模块与外部环境交互，形成一个闭环的工作体系。因此，理解Agent的演化，既要关注大模型本身的迭代，也要考察其所依托的框架与系统设计。下图中总结了2023年1月至2025年8月以来，关键Agent架构的发布时间、智能体类型以及多/单智能体结构的变化：

（1）整体来看，智能体框架发布的主力类型经历了从**通用型到软件开发类，再到终端交互类**的转变。

（2）**通用型智能体框架**在大模型爆发初期（2023年上半年）经历了大量发布，但受限于底层模型的能力表现，探索逐渐转向以软件开发为代表的垂类任务领域，但在推理模型浪潮开始后，对于通用型框架的探索热情又逐渐被激活。

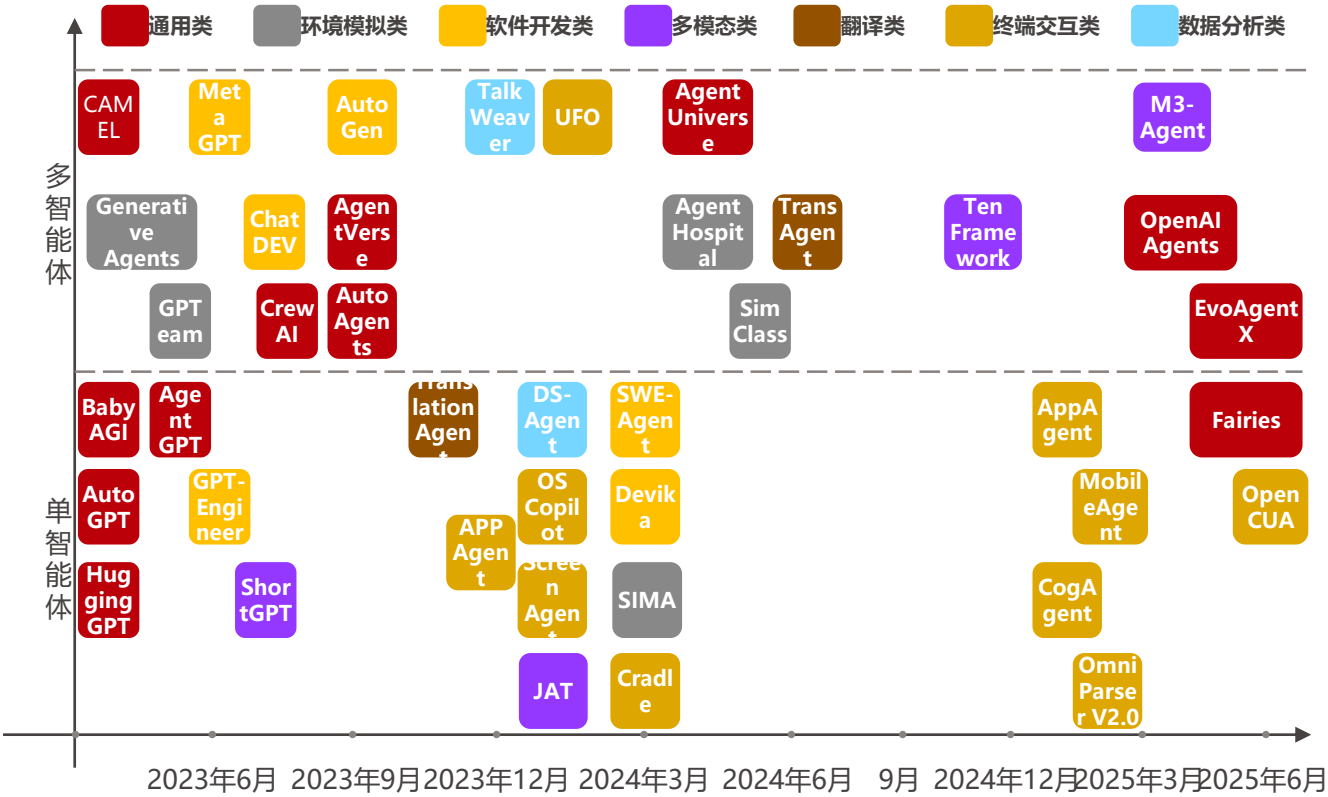


图 2023年1月至2025年7月关键Agent架构发布情况

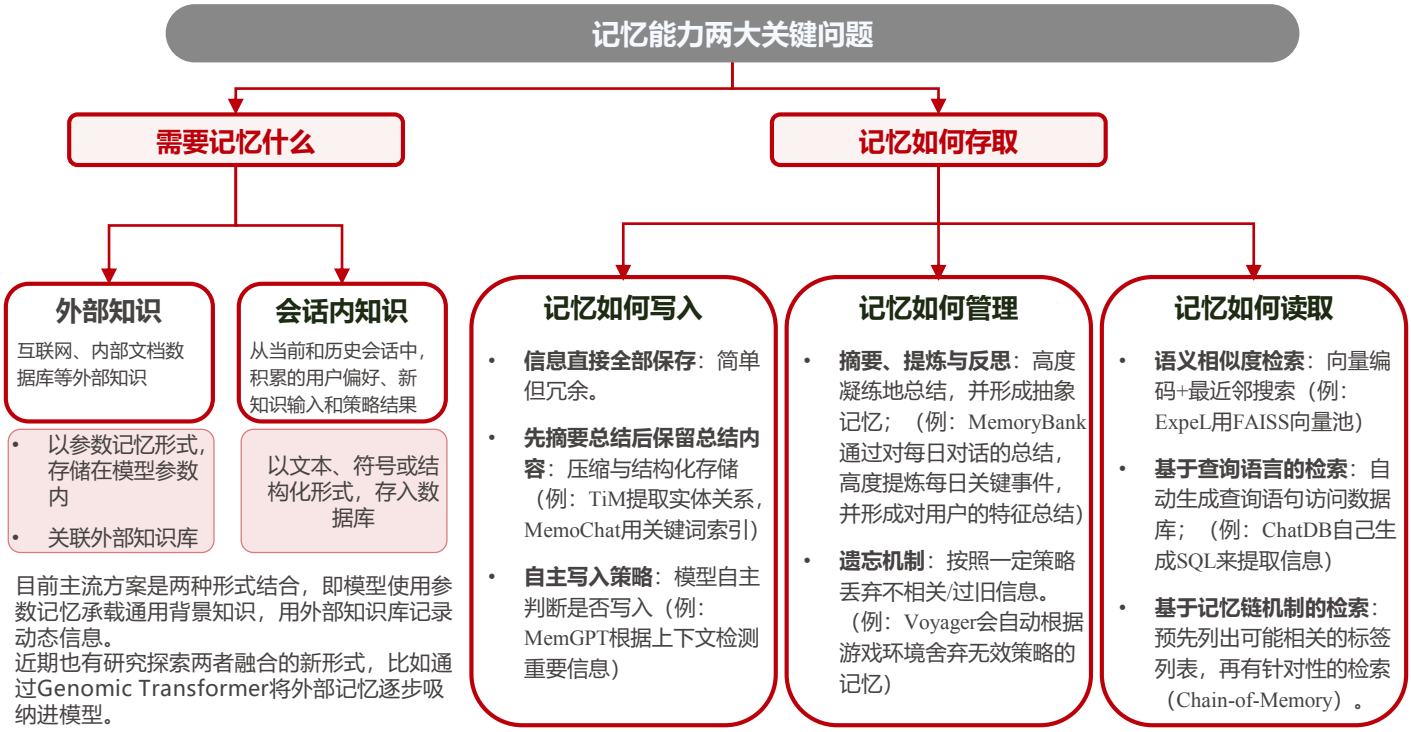
（3）软件开发类智能体框架是探索最为集中的垂类任务领域，其探索已从传统代码补全扩展到整个开发工具链的深度集成。

（4）终端交互类智能体框架在2024年以来，成果不断累积。探索终端涵盖电脑、手机等，通过模拟鼠标点击、输入、跨应用操作，使大模型具备直接操纵计算机环境的能力。

关键观点 | **智能体框架的演进，实质上是大模型能力不断突破后，在通用验证、垂直应用与真实环境交互三个层次上的递进探索。**

在模型和框架的支撑之外，Agent想要向理想状态前进，其关键能力的进展同样不可或缺，其中最为核心的环节包括记忆、规划、工具使用与执行。

首先是**记忆模块**。随着模型在长文本处理上的突破，以及外部记忆库的引入和对写入、管理、读取机制的探索，智能体逐渐具备了跨周期保留交互经验的能力。短时记忆保证了对当前上下文的敏锐响应，长期记忆则通过数据库或知识库记录用户偏好、任务进度与环境状态。二者结合，使智能体在多轮交互中能够展现出连续的人格与稳定的对话风格，从而提供更具连贯性与沉浸感的使用体验。



关键观点 | **在这一框架支撑下，记忆模块通过结合底层模型的长文本能力与外部记忆机制，保障交互的连续性与沉浸感。**

第二是**规划模块。**

在早期Agent设计中，规划能力主要依赖外部规划器和人类反馈增强。也就是说，系统本身缺乏独立生成和优化行动策略的能力，更多依赖于人为的指令、规则或外部工具来完成复杂任务。因此，Agentic WorkFlow在此阶段诞生，可以通过工程化手段，预设和提升Agent的执行稳定性。

随着推理模型的出现，这一模式正在发生根本性的变化。推理模型能够结合强化学习、自我反思以及记忆检索等机制，使Agent在执行任务时向独立自主规划更近一步。这不仅增强了系统的灵活性和适应性，也使其规划能力逐步内生化——即规划不再完全依赖外部输入，而是可以在模型内部进行优化和迭代，从而在面对动态或复杂场景时表现得更加稳健和高效。

然而，这种提升仍存在一定局限。首先，智能体在面对完全未知或高度动态的环境时，规划仍可能出现错误或不稳定；其次，多步推理和跨任务长期规划能力仍受限，可能影响决策的连贯性；自我反思和记忆检索可能引入噪声或过时信息，影响规划结果的可靠性；强化学习和检索机制增加了计算开销，对于实时性要求高的场景存在挑战；最后，规划能力内生化使得决策过程越来越黑箱，可解释性和可控性受到一定影响，在安全关键型场景中仍需谨慎。总体来看，内生化规划带来了自主性和稳健性提升，但仍需要结合外部机制和人类监督以应对复杂环境和高风险应用。

关键观点 **随着交互的加深，规划能力从依赖外部规划器和人类反馈增强，逐步过渡到推理模型驱动下的自主化，通过强化学习、自我反思与记忆检索增强，形成更稳健的内生规划机制**

除了记忆和规划外，智能体需要通过调用外部工具来具体执行指定任务。经历了最初API调用，到插件生态，再到MCP协议的诞生发展后，智能体的外部工具生态开始迈向标准化与统一化，为能力外延提供了极大支持。

以电脑操作为代表的设备使用框架的出现，使智能体能够模拟鼠标点击、键盘输入等操作，跨应用完成文件管理、表格处理、信息检索等复杂任务。执行力的提升赋予了智能体真正的行动属性，使其能够在真实数字环境中自主完成任务。

关键观点 **外部工具生态的统一化与设备操作框架的演进显著提升了智能体的执行力，使其真正具备在数字环境中自主完成任务的能力。**

2.1.2 软硬协同加速终端智能普及

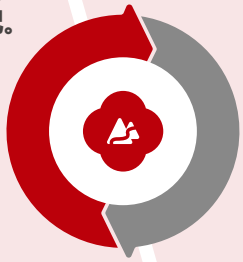
终端智能的普及，本质上是一场软硬件协同演化的结果。仅依靠云端大模型，难以支撑终端智能的实时交互和大规模应用的普及。一方面，远程推理带来延迟高、带宽占用大、算力成本高的问题；另一方面，云端集中处理也存在数据安全与隐私保护的隐忧。反之，如果完全依赖终端侧算力，现有芯片与推理框架又难以承载复杂的大模型任务。因此，端云协同成为现阶段的必然选择：云端作为能力底座，提供大规模推理与知识更新，端侧则作为交互入口，承担实时响应与个性化处理。只有在软硬件共同演进的前提下，终端智能才具备普及的可能。

软件层面：模型压缩与推理优化是端侧可用的前提。

轻量化方法在保持模型关键能力的前提下，实现参数规模的端侧适配。

通过剪枝、蒸馏、量化等轻量化手段，原本数百亿参数的大模型可以缩减到数亿/数十亿规模，同时保留核心性能。

例：DeepSeek使用R1模型作为教师模型，使用Qwen-7B作为学生模型蒸馏的DeepSeek-R1-Distill-Qwen-7B在AIME 2024pass@1和MATH-500 pass@1上得分超越QwQ-32B-Preview，一定程度上证明了蒸馏方法下，小模型能够获得接近大模型的性能表现。



端侧推理框架保障端侧小模型在多终端下的性能一致性。

在推理框架上，ONNX Runtime、TensorRT、CoreML、MindSpore Lite等提供了**显存、延迟和兼容性**的优化，保证小模型在不同终端的高效运行。

例：Apple发布的Core ML针对多类型模型进行端侧优化，能够充分利用Apple芯片并尽可能降低内存与能耗，同时提供便捷的模型转换工具，并与Xcode生态应用深度整合。

硬件层面：AI端侧芯片是端侧智能的性能底座。

AI手机芯片和PC级芯片纷纷强化NPU/TPU等专用算力单元，显著提升了端侧AI推理的速度与能效，这使得终端设备逐渐具备运行本地智能体的条件。

芯片类型	厂商	芯片名称	发布时间	算力参数
手机芯片	苹果	A18 Pro	2024年09月	35TOPS
	苹果	A18	2024年09月	35TOPS
	高通	骁龙8 Elite	2024年10月	80TOPS
	联发科	天玑9400	2024年10月	80TOPS
	华为	麒麟9020	2024年11月	-
	小米	玄戒O1	2025年05月	44TOPS
PC级芯片	高通	骁龙X Elite	2023年10月	45TOPS
	高通	骁龙X Plus	2024年04月	45TOPS
	苹果	M4	2024年05月	38TOPS
	AMD	锐龙9000	2024年06月	50TOPS
	AMD	锐龙AI 300	2024年06月	50TOPS
	英特尔	酷睿 Ultra 200V	2024年09月	48TOPS

图 各家芯片厂商最新一代AI芯片规格情况统计

软硬协同：算子优化和端云协同是终端智能大规模落地的核心保障。

软硬协同构成了终端智能普及的关键枢纽。一方面，算子优化成为软件与硬件间的桥梁，通过统一的算子库和加速指令实现模型在不同芯片上的高效调度。另一方面，端云协同正在成为主流架构：云端提供大模型的能力底座，端侧承担实时交互与个性化处理，两者实现动态分工。这种协同模式既降低了延迟和隐私风险，又能灵活调用云端算力，确保终端设备在有限算力下也能获得接近云端的体验。

图 端云协同模式示意图

关键观点

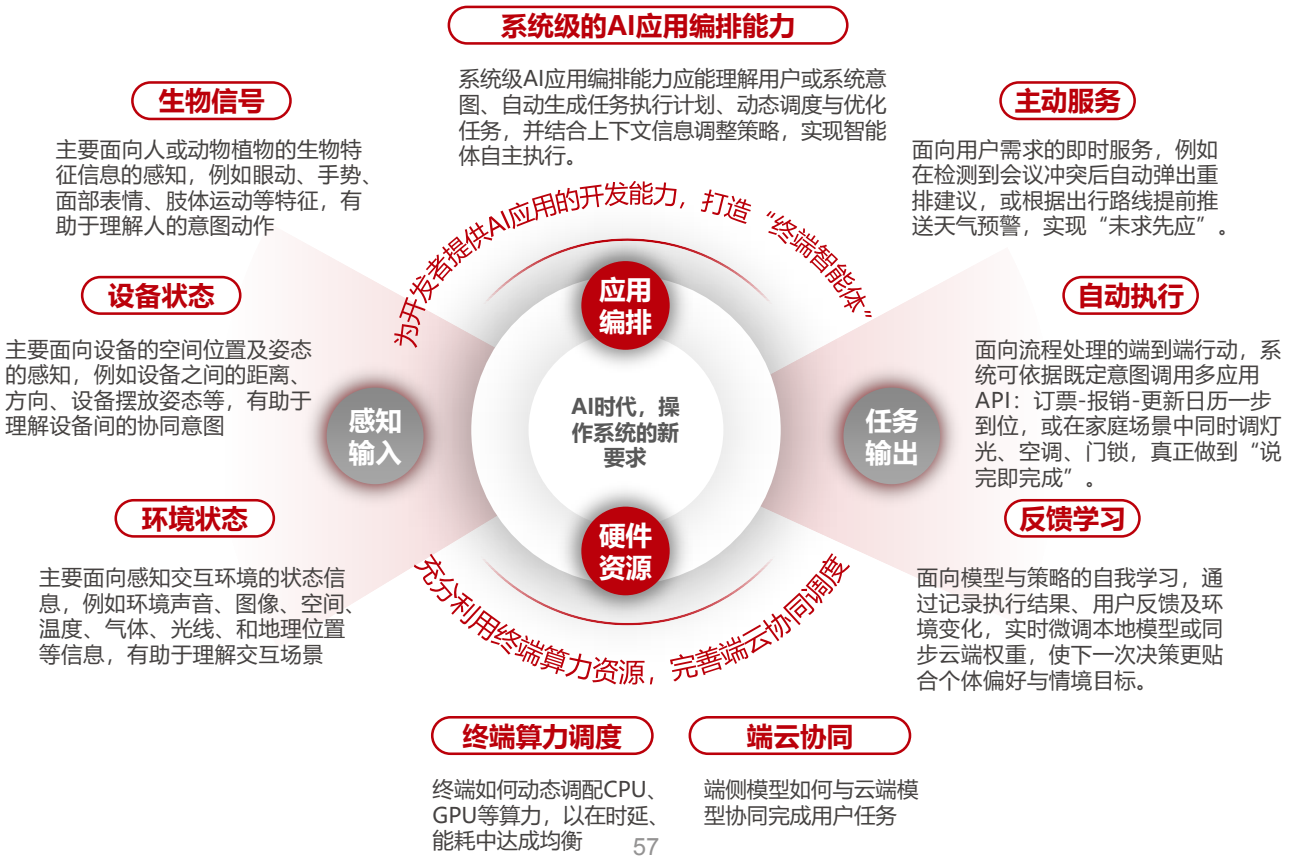
终端智能的普及依赖软硬件协同：模型压缩与推理优化确保端侧模型可用性，AI端侧芯片提供性能支撑，而算子优化与端云协同则是智能体大规模落地的关键保障。

2.1.3 操作系统加速智能原生化

在本轮AI引领的技术变革中，向下管理硬件资源、向上支撑软件应用的操作系统也面临新的挑战和革新动力。传统操作系统的架构并不是专门面向AI任务设计的，这意味着一方面现有内核的调度与资源管理机制对异构算力与云端算力弹性协同的场景，无法满足大模型推理对时延和能耗的要求；另一方面，缺乏对多模态感知、意图解析与跨应用自动编排的原生支持，也使操作系统难以把生成式AI的决策即时落地为安全且稳定的智能行动。

因此，操作系统必须在纵向和横向同步进化。纵向上，AI时代的操作系统应更高效地调度终端算力资源，这既包括异构算力单元的调度，也包括端云协同的实现。在调度之上，操作系统也需要具备AI应用编排的系统级能力，提供原生的意图驱动编排能力，更好的支撑终端智能体的各项应用。

横向上，操作系统也需要持续拓展感知输入的范围和种类，操作系统能够采集生物信号、设备状态和环境变化，并将这些上下文输入模型进行推断。推断结果随即化为主动服务或自动执行，执行效果又被反馈给模型，驱动下一轮调优。通过闭合“感知与记忆—自主规划—工具—行动”循环，操作系统才能真正完成从传统“人机交互”到“机器自主”的能力提升。



iOS：AI不是产品，而是更智能的iOS系统的底层支撑，本质是增强本身设备产品的竞争力。关注隐私安全，端侧探索优先。

苹果自研的Apple Foundation Models（AFM）构成了Apple Intelligence的核心基础，其中包括一个拥有30亿参数、能够在设备端运行的AFM-on-device模型，以及一个规模推测约为700亿参数的云端模型AFM-Server。在云端层面，苹果同时开放了第三方大模型的API接入能力，允许它们通过系统接口集成进Apple Intelligence的工作流中。需要注意的是，这些第三方模型在系统中并不会获得额外的权限，用户必须显式同意数据传输才能使用。一旦用户授权，其对话数据将会被发送至例如OpenAI的服务器，并受到相关服务商的隐私政策保护，而不再适用苹果的隐私保障体系。

为了帮助开发者更好地开发和使用系统级AI，苹果同时开放了两类关键API：其一是Foundation Models framework，开发者可以通过该框架直接调用端侧模型，对模型输出进行约束并结合iOS系统框架，实现对话管理、工具调用和多模态处理等能力。更值得一提的是，因为开发为端侧模型，算力由苹果本地设备提供，这意味着开发者不需要为AI功能进行额外付费，同时也保障了用户的隐私安全。

其二是App Intents API，开发者可以借助它让应用的功能和数据向Siri、Spotlight及Apple Intelligence这类系统级入口暴露，从而实现应用功能的智能化调度与系统级调用。

让APP内部获得端侧AI能力

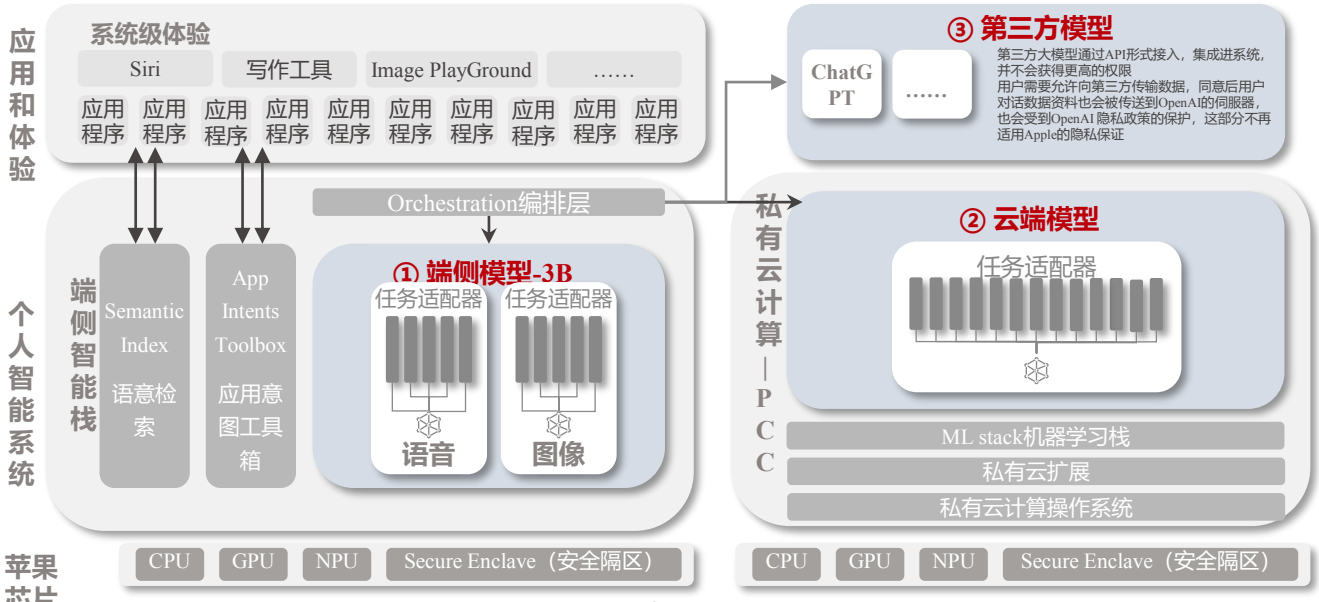
API暴露①：Foundation Models framework

开发者可通过该API调用苹果端侧模型，在约束输出结构的同时，支持调用开发者预先定义好的工具（应用数据库、系统框架API等）、整合上下文与对话管理，并在此过程中确保结果的可控性与安全性。

让APP被系统智能发现&调用

API暴露②：App Intents API

开发者可通过该API，让应用功能可被Siri/Apple Intelligence调用，暴露应用数据给系统搜索/智能体以及定义Siri/Apple Intelligence可以执行的交互动作



整体而言，**苹果的路径强调AI不是独立产品，而是作为智能操作系统的底层支撑。**其核心逻辑是通过端云协同架构保障隐私与性能的平衡，并通过系统级API规范生态，确保应用与系统的协同始终处于苹果的控制框架之下。

Android：把AI视为操作系统的核心驱动力，通过Gemini打造统一入口，云端模型使用更为激进，既服务多模态与跨设备场景，也承载着推动Google云服务商业化的战略目标。

Gemini模型体系构成了Android系统级AI的核心基础，包括端侧模型Gemini Nano和云端Gemini Flash/Pro模型。在云端层面，Android积极开放Gemini Live API，使开发者能够将实时语音、图像、视频和屏幕共享功能集成到应用中，同时支持App调用，实现AI与应用功能的深度联动。部分场景下，Gemini Live结合端侧Gemini Nano做延迟优化，以提升用户实时交互体验。

图 Android AI相关的API

在Android系统中，AI能力通过多层API对开发者开放，形成端云协同的开发体系。AICore SDK提供端侧模型（Gemini Nano）的托管与更新能力，使App能够在本地完成推理任务，保障低延迟和离线能力；在此基础上，ML Kit GenAI API封装了高阶功能，为中小开发者提供开箱即用的AI能力，如文本生成、总结、校对及图像描述等，实际调用仍依赖AICore管理的Nano模型运行时支持；AppFunctions API允许开发者将应用功能注册至系统，以便系统发现、索引并智能调度App的能力，实现应用与系统的协同调用；而Gemini Live API则专注于实时对话与多模态交互，主要由云端Gemini Flash/Pro模型提供生成能力，部分场景结合端侧Gemini Nano做延迟优化，支持实时语音转写与生成、多模态输入（语音、图像、视频、屏幕共享）及函数调用功能，使应用能够直接与系统级AI交互，完成复杂任务处理和跨设备智能体验。

整体来看，Android的路径强调AI作为操作系统的核心主体，通过端云协同架构释放计算能力与模型潜力，推动跨设备、多模态场景的智能化落地，同时为开发者提供开放API生态，以便应用深度嵌入系统级AI服务。**相较于苹果，Android在云端模型使用上更为激进，探索更丰富的实时交互与跨设备体验，但短期内可能面临延迟、算力与用户隐私感知等挑战。**

Apple和Android在AI方面的布局情况总结：

	Apple	Android
核心定位	更智能的操作系统是增强生态设备产品竞争力的手段	通过降低开发者接入Gemini的门槛，把应用流量和实时AI调用导向Google Cloud，从而推动云服务的使用与商业化。
端侧模型	AFM-on-device（3B）	Gemini-Nano（1.8B/3.25B）
云端模型	AFM-Server（推测70B）	Gemini Flash/Pro
第三方模型集成	√，ChatGPT（需要用户明确授权）	×
优先探索终端	手机、iPad、Macbook、PC	手机、耳机、眼镜等可穿戴设备
功能升级方向	文本和语音优先	多模态优先
API暴露	Foundation Models framework和App Intents API	AICore SDK、ML Kit GenAI API、AppFunctions API、Gemini Live API
未来路线	建设siri、Spotlight等系统级入口，整合应用服务，强化用户感知	AI入口集中在Gemini Live，短期用户感知可能更明显，但隐私与延迟优化仍需权衡
安全	硬件安全+PCC安全	硬件安全+网络隔离

2.2 多设备融合下的新体验

在智能化技术不断演进的背景下，硬件能力提升、端云协同加速以及操作系统原生AI能力的不断增强，为跨设备协同和多终端智能化应用奠定了基础。

然而，技术进步不仅改变了单一终端的智能能力，也正在打破设备边界的限制，使用户的任务、数据和注意力不再局限于某一设备。手机、平板、可穿戴设备和车机等多种终端逐渐形成一个互联的智能网络，用户期望在不同设备间实现任务的无缝延续。

这种趋势不仅提出了新的智能体验需求，也对应用开发方式提出了挑战。这意味着开发者必须考虑多终端环境下的任务流、数据同步和交互一致性，从而推动以设备为中心向以跨设备服务为中心的开发范式演化。

也意味着开发生态需要在工具链、操作系统能力和跨端服务支持上进行系统升级，以保障开发者能够高效构建可在多设备间流转的服务和应用，并通过统一接口和分布式框架实现任务迁移、状态同步与硬件共享，从而支撑新一代智能体验的落地。

2.2.1 多设备时代的用户体验诉求与生态分工

根据工信部统计数据，截至2024年12月，中国智能手机人均保有量已达到1.11部，家用电视机保有量为1.087台，每百户家庭拥有计算机44.8台、汽车51.2台。这一系列数据表明，大众每天所使用的设备类型和数量都在持续增长。随着设备保有量不断提升，用户的注意力、任务流和数据也被分散在不同的终端之中。**单一设备已经难以完全满足人们在通信、娱乐、出行、健康和办公等多方面的需求，多终端之间的任务融合与体验统一因此成为不可避免的发展方向。**这种趋势不仅体现为硬件形态的丰富，更体现在操作系统与生态体系中需要提供跨设备的连接能力、任务迁移机制和硬件共享方案，从而让用户在复杂的多设备环境中仍能获得流畅、一致的使用体验。

从技术体系的角度看，多终端融合下任务执行的技术体系分为三层。最底层是连接与传输，这一层通过蓝牙、Wi-Fi、USB、UWB、NFC等设备连接协议建立起跨设备的通信链路，并依托TCP/IP、UDP、HTTP、MQTT等传输协议完成数据传输，为任务的迁移与共享奠定基础。

在此之上，是操作系统提供的开放能力，它们将底层的异构连接进行抽象与统一，形成开发者和应用可调用的接口，**这也意味着开发者并不需要直接处理复杂的协议栈和硬件类型**，就能够通过系统提供的API实现设备发现、任务迁移和硬件共享等功能。

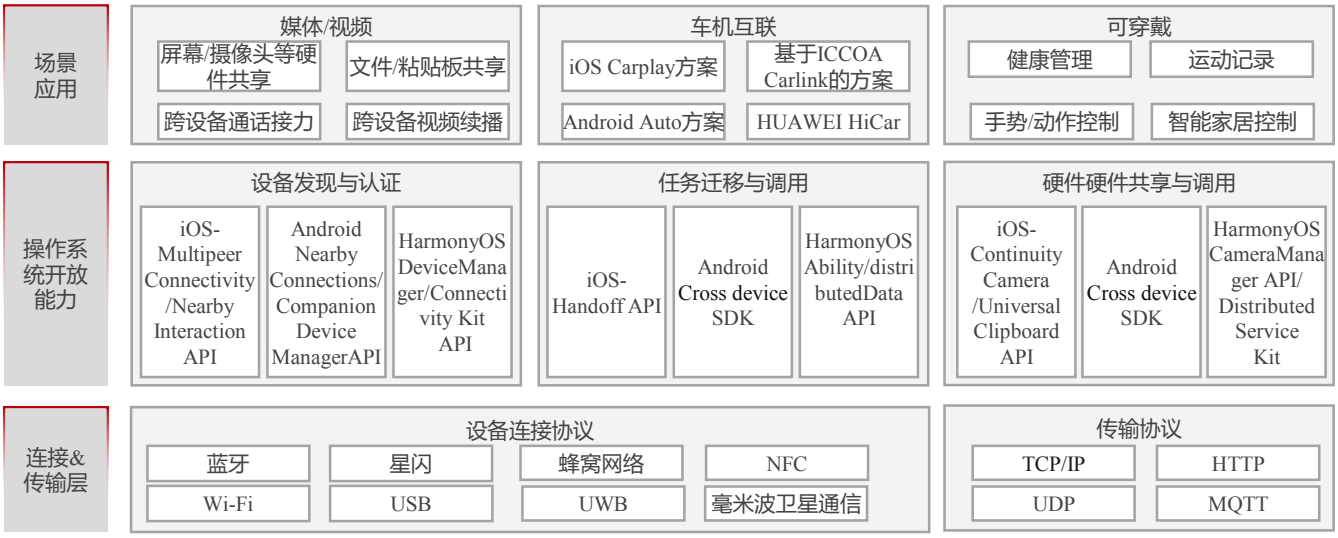


图 跨设备协同技术体系示意图

在操作系统开放能力的支持下，最上层便是具体的场景应用。这一层直接面向用户，将底层的通信与系统抽象转化为可感知的体验。例如，在媒体和通话场景中，用户能够在手机与平板之间实现通话的接力和视频的续播；在车机互联场景中，手机上的导航与媒体播放可以无缝

缝切换至车载屏幕；在可穿戴场景中，手表采集的健康数据可以自动同步至手机或云端，耳机也能够承担通话与语音交互的延伸。

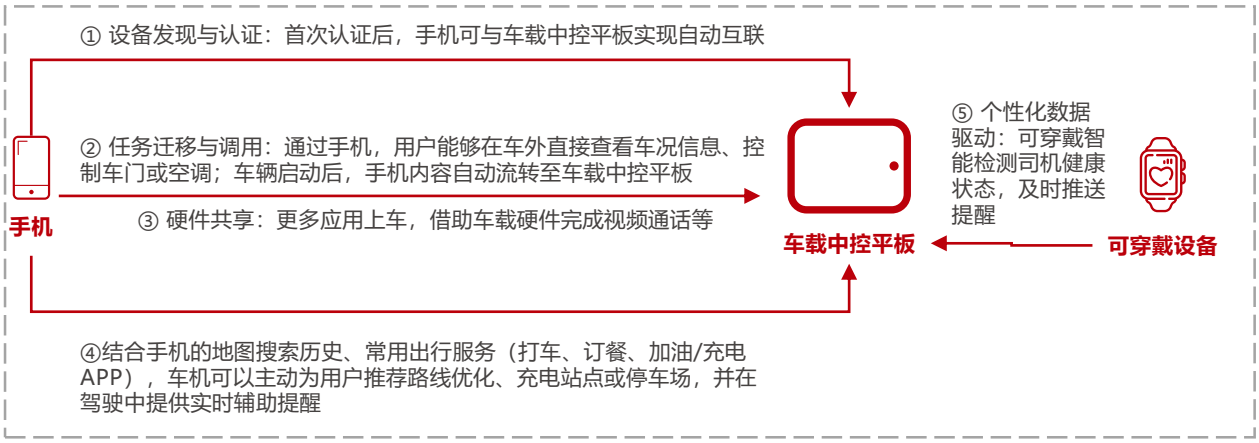


图 车载场景跨设备协同应用示意图

与这三层技术架构相对应，操作系统需要提供一系列开发工具，以帮助生态伙伴更高效地构建跨设备应用。需要强调的是，多终端协同的落地，并非完全依赖操作系统单方面完成。**操作系统在其中的角色，是提供统一的底层连接、分布式运行和开放接口等基础设施，以确保不同设备能够在安全、稳定的环境中互联互通。**例如，设备发现与认证机制、跨协议通信抽象、分布式运行时和状态同步框架，都是必须由操作系统打底实现的核心能力。开发者无需直接处理复杂的协议和硬件差异，只需调用操作系统提供的标准接口，即可构建跨设备的应用逻辑。

在此基础之上，更上层的丰富应用与场景创新，则主要依靠生态来推动。无论是即时通信应用实现的跨端消息接力，视频平台提供的投屏与多屏互动，还是车企和出行服务商打造的手机—车机联动体验，均属于生态利用操作系统底座能力进行差异化创新的成果。与此同时，医疗、教育、办公等垂直行业应用，也可以在操作系统的分布式数据与多终端适配框架上，构建契合自身需求的跨端解决方案。此外，**还有一些能力需要操作系统与生态的协同，**例如多模态交互、跨端界面适配、AI驱动的个性化推荐，往往需要操作系统提供统一的交互接口与隐私保护机制，而生态开发者则在此基础上实现具体的业务逻辑与差异化体验。

由此可见，多终端任务融合的健康发展路径应当是“操作系统打底、生态百花齐放”：操作系统通过提供连接抽象、分布式能力与开发工具奠定技术统一性与安全性，而生态则在这一底座之上释放创新活力，共同推动多终端融合体验的演进。

关键观点

在设备数量和形态不断扩张的背景下，多终端协同的实现需要操作系统提供统一的连接和技术底座，同时依托生态伙伴在应用层面的创新，共同推动用户跨设备体验的流畅与一致。

2.2.1 核心观点总结

多终端普及与用户需求分散：智能手机、电视、计算机和汽车等设备保有量持续增长，用户的注意力和任务被分散在多种终端上，单一设备已难以满足全方位需求。

多终端融合成为发展趋势：跨设备的任务迁移、体验统一和硬件共享成为必然方向，操作系统与生态体系需提供相应能力以保证流畅一致的用户体验。

技术体系分三层支撑多终端任务：底层是连接与传输，通过各种协议实现跨设备通信；中层是操作系统开放能力，提供统一API让开发者无需处理复杂协议和硬件差异；上层是场景应用，将底层能力转化为用户可感知的具体体验。

操作系统提供技术底座：通过设备发现、认证机制、跨协议通信抽象、分布式运行时和状态同步框架等基础设施，保障跨设备应用的安全性和稳定性。

生态推动场景创新：具体应用和差异化体验主要由生态开发者在操作系统提供的底座能力上实现，包括跨端消息接力、投屏、多屏互动及垂直行业解决方案。

操作系统与生态协同能力：多模态交互、跨端界面适配和个性化AI推荐等功能需要操作系统提供统一接口和隐私保护，生态开发者则实现具体业务逻辑与创新体验。

健康发展路径：操作系统打底、生态百花齐放：操作系统提供统一性、安全性和分布式能力，生态释放创新活力，共同推动多终端融合体验演进。

2.2.2 多设备协同下的应用开发

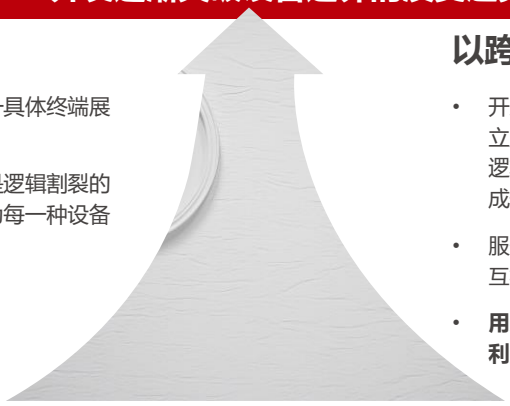
在多终端融合的趋势下，应用开发正在经历着一场深刻的范式转变，其核心就是从以单设备为中心逐步演化为以跨设备服务为中心。

在过去的应用生态中，开发者通常以**单一设备**作为核心入口，围绕手机、平板、手表或车机分别开发独立的应用程序，并针对不同的屏幕尺寸、硬件性能和交互方式进行专门适配。这种模式虽然满足了早期多设备并行发展的需求，但也带来了明显的弊端：其一，开发者需要重复投入资源来维护多个版本，增加了开发与运维的复杂度；其二，用户在不同设备之间切换时，往往面临体验断裂的问题，任务无法自然延续，导致用户必须在多个设备上分别完成同一流程；其三，服务逻辑与硬件高度耦合，制约了应用的扩展性与场景化创新。

开发逐渐突破设备边界的演变趋势

以单设备为中心

- 应用的开发与分发通常围绕某一具体终端展开，例如手机、平板、电脑等。
- 用户在不同设备上使用的往往是逻辑割裂的多个应用版本，开发者也必须为每一种设备单独维护代码与功能适配。
- 功能边界与硬件强绑定。



以跨设备服务为中心

- 开发者的目标不再是为某类单一设备构建独立应用，而是围绕用户需求抽象出核心服务逻辑，并通过操作系统提供的分布式框架完成一次开发、多端运行。
- 服务逻辑一次开发即可跨设备运行，UI与交互根据设备特性自动适配。
- **用户关注的不是在哪个设备上用，而是能否利用跨设备来无缝完成任务。**

随着用户需求的演进和设备形态的持续丰富，这种以单设备为边界的开发模式已经难以满足跨终端、一致化的体验要求。用户的注意力和数据被分散在手机、平板、穿戴设备、车机、电视等不同终端之中，他们真正关注的不是应用在哪个设备运行，而是服务是否能够无缝衔接地完成任务。例如，在出行场景中，用户希望手机上的导航能够自动流转到车机屏幕；在影音娱乐场景中，用户期待能在电视上接续播放手机上尚未看完的视频；在健康管理场景中，用户希望手表实时采集的生理数据能够自动同步至手机应用或云端，供医生远程分析。所有这些需求，都指向一个共同的方向：应用开发必须脱离单一设备的约束，以跨设备服务为核心来组织逻辑与体验。

以跨设备服务为中心的开发模式，强调的是服务逻辑与单一界面表现的解耦。开发者不再需要为每一类设备重复构建应用，而是以服务作为基本单元，将核心功能与任务抽象出来，交由操作系统的分布式框架和运行环境去完成跨设备的分发、调度和渲染。换言之，服务只需一次开发，便可以在不同终端间自由流动和扩展。

这一转变对操作系统提出了更高要求。操作系统需要构建坚实的分布式底座，提供设备发现与认证机制、跨协议通信抽象、状态同步框架和跨端界面适配工具，帮助开发者屏蔽不同设备间的硬件差异和协议复杂性。同时，操作系统还需要在隐私保护和安全机制上提供保障，确保服务在跨设备流转过程中不泄露数据，维持用户对多终端体验的信任感。开发者借助操作系统的这些能力，可以将精力集中在服务逻辑与场景创新上，而不是被底层的兼容性问题消耗。

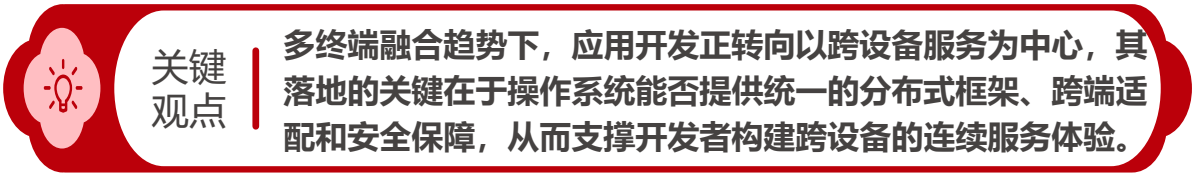
与此同时，生态的作用同样不可忽视。操作系统负责提供统一的分布式运行与连接框架，而生态开发者则是这一能力的直接受益者和价值实现者。即时通信应用、视频平台、出行服务、医疗健康系统等领域的开发者，可以基于这一底座构建跨设备的服务体验。

也正是如此，应用和服务的分发逻辑也正在产生变化。

在**以单一设备为中心**的模式下，分发逻辑是应用商店→设备→应用：每个设备都有自己的应用商店或分发渠道（如手机应用市场、车机应用市场、智能手表应用市场），用户需要在每个设备上单独下载和安装应用。这样导致同一服务在不同设备上有多个版本，更新和维护也都割裂。

而在**以跨设备服务为中心**的模式下，分发逻辑逐渐转向服务→用户→多设备：用户只需一次获取服务（例如绑定账号或订阅），应用的核心逻辑就可以在操作系统的分布式框架下被调用，并根据设备特性自动分发到合适的终端。

所以，分发逻辑的变化本质上是从面向设备的多版本应用分发演进为面向用户的统一服务分发，而操作系统需要在其中提供跨端调度、界面适配和状态同步的能力，确保服务能够随用户而动，而不是被设备所限。



2.2.2 核心观点总结

开发范式转变：应用开发正在从以单设备为中心向以跨设备服务为中心演进，核心目标是实现跨终端、一致化的用户体验。

传统模式弊端：以单设备为边界的开发导致开发与运维成本高、用户体验断裂，以及服务逻辑与硬件高度耦合，制约扩展性与场景创新。

用户关注点变化：用户真正关心的是服务能否无缝完成任务，而非应用运行在哪个设备上，多终端间的任务流和数据流必须自然衔接。

以跨设备服务为核心的开发模式：服务逻辑与界面表现解耦，开发者只需一次开发核心功能，操作系统负责跨设备分发、调度和渲染，实现功能在不同终端自由流动。

操作系统的新要求：操作系统需要提供分布式底座、设备发现与认证、跨协议通信、状态同步及跨端界面适配，同时保障隐私与安全，使开发者专注于服务逻辑和场景创新。

生态作用不可忽视：操作系统提供底座能力，而生态开发者利用这些能力实现跨设备服务体验，如即时通信、视频播放、出行和医疗健康服务等。

分发逻辑变化：分发模式由应用商店→设备→应用的多版本模式，演进为服务→用户→多设备的统一服务分发，操作系统需提供跨端调度、界面适配和状态同步能力。

2.3 面向智能时代的新开发范式

正如2.2所言，多终端融合推动了以跨设备服务为中心的开发模式，使应用逻辑与界面表现解耦，分发方式由以设备为中心转向以跨设备服务为中心，同时操作系统与生态共同支撑了跨端体验的实现。

这一转变不仅改变了开发者的角色和分工，也对开发工具链和流程提出了更高要求。在智能化时代，随着大模型、AI辅助工具以及自然语言交互的成熟，开发范式正在迎来新的重构：工具链、开发形态都在嵌入智能能力，这也将重新定义应用构建的方式和效率。

接下来我们会从AI辅助开发工具和Agent类产品出发，聚焦面向智能时代的新开发范式，分别探讨大模型如何赋能开发工具链重构、意图驱动开发形态的初步形成，以及智能能力在开发流程中嵌入所面临的挑战。

2.3.1 大模型赋能开发工具链重构

从Github Copilot的诞生，到Cursor、Windsurf和Devin的发布，大模型在软件开发领域内的应用，在产品层面体现得淋漓尽致。在这过程中，AI辅助开发工具的产品形态也经历了几轮的变化，逐渐走过了简易工具辅助编码—对话式代码助手—AI辅助开发助手—编码智能体的演进路径。

图 编码工具的四个发展阶段

目前，AI辅助开发工具的产品形态已逐渐稳定。一种是以Github Copilot、DevEco CodeGenie为代表的**扩展插件形态**。这类产品依附于现有的IDE或编辑器生态，以扩展的方式提供代码补全、函数生成、文档生成、语义搜索等功能。它们的优势在于对开发者的工作流干扰最小，能够快速融入既有工具链，因此适合大规模推广。但其局限也较为明显，交互方式仍然局限在局部的代码片段层面，难以完成跨文件、跨模块的复杂任务。

第二种是以Cursor、Windsurf为代表的**AI IDE**形态。通过打造全新的AI原生IDE，产品团队能够在交互逻辑、界面设计、数据闭环等方面拥有完整自由度，并在持续使用中沉淀宝贵的用户行为数据，从而不断优化模型与工具的协同。这种形态更适合深度探索新一代智能开发环境。

两类形态各有优势：**IDE路线强调完整的产品控制与长期数据价值，VSCode Extension则依托现有生态快速渗透用户群体。**值得注意的是，也存在同时探索这两种产品形态的团队，例如**字节跳动**，既拥有插件形态的**MarsCode**，又发布了自己的AI IDE产品**TRAE**，这种“双线并行”策略既保证了快速覆盖，又能积累深度数据和优化长期体验。

除此之外，Devin的发布也拉开了**Coding Agent**探索的序幕，让大众看到了**异步Agent**的工作方式魅力，即Devin可以在后台独立运行任务，用户在此期间可以专注于其他工作，而无需持续监督Agent的每一步操作。相比传统的一问一答式工具，Devin不仅能够自主规划任务流程、编写代码、调试和创建文件，还能通过虚拟机访问互联网获取所需信息，实现较高程度的自主性。

更重要的是，用户可以随时打断、调整其执行进程，从而保持对开发方向和结果的掌控。这种协作式工作模式突破了早期AI辅助工具对用户注意力的依赖，使Agent从被动响应型助手演化为主动推进任务的智能开发伙伴。同时，通过与Slack等协作工具的深度集成，Devin能够获取任务背景和上下文信息，进一步提升执行的准确性和效率。

	AI辅助开发工具	Coding Agent类产品
核心目标	解决的是如何更快地写/改代码，增强专业开发者的编码体验，减少查文档/写样板代码时间	解决的是如何把事情做完，如何将一个功能/项目完整的交付
任务颗粒度	函数/片段级别的补全、文档生成、代码搜索。现在正在经历片段生成→上下文理解→文件操作→环境配置的转变，对代码的理解逐渐深入	现阶段可以参与多文件/模块级任务，执行跨文件修改、复杂需求拆解、全流程迭代，目标最终为实现端到端开发
交互模式	以人类驱动为核心，AI对于人类指令被动响应，不会根据结果调整下一步行动，流程相对固定，人类输入明确提示词，模型给出回应，人类选择是否接受修改	用户派任务，Agent自行规划行动，并根据用户反馈持续调整直至完成
典型产品	GitHub Copilot、Cursor、Windsurf、Momentic（UI自动化测试）、Gru AI（单元测试）、MarsCode、Trea	Devin、Replit Agent、AMP、Cursor Background Agent、Windsruf Agent、Trea Builder
特点	强调快和开发工具内置、意图预测，帮助开发者保持心流	需要长上下文+长推理链，具备执行与迭代能力，能与sandbox/runtime交互
对开发效率的影响	提高局部环节的代码编写速度，并且与现有IDE或编辑器无缝集成，开发者迁移学习成本低	借助自主规划和执行能力，Agent可以异步执行与并行处理：开发者可在执行其他工作时，让Agent在后台运行，提高整体工作产出效率。
对开发流程的影响	• **流程仍高度依赖开发者**：任务执行依赖用户持续监督和输入，无法独立完成跨模块或复杂任务。 • **优化重点在代码编写的局部环节**：主要改善代码书写效率和错误检测，而对项目整体流程或跨任务协调作用有限。	• **流程自动化与协作优化**：Agent可在关键节点向用户汇报进展，允许用户随时调整，实现高度透明的协作模式。 • **人机协作角色转变**：开发者从手动执行者转为策略规划者与监督者，任务可规模化扩展，部分重复性或复杂技术环节完全交给Agent。

图 AI辅助开发工具和Coding Agent类产品的区别

然而，过度的自主性也可能带来风险和管理成本，过度自由的Agent可能导致任务执行的不可控或结果偏离预期。例如，Devin的使用者反馈，Devin在某些情况下会尝试推进实际上不可能完成的任务，这时Devin会耗费大量时间且生成无效方案，甚至可能编造方案。因此，一些Coding Agent产品正在探索**更合理的人机交互方式**，以在自主性与可控性之间找到平衡。例如，Replit推出的AI Agent在执行任务时，会持续与用户保持交互，定期汇报进展，并在关键决策节点请求确认，从而确保开发方向符合用户预期。

这种设计体现了协作优先的理念：Agent既能独立完成复杂任务，又能够在用户需要时进行即时调整和干预，实现人机协作的最优效率。通过这种交互模式，用户可以利用Agent处理重复性或复杂的技术细节，同时保留对核心逻辑、架构决策的掌控，从而提高整体开发效率并降低错误风险。这也标志着Coding Agent产品进一步向合作型智能体迈进，为软件开发带来新的生产力范式。

关键观点

辅助工具定位于局部编码效率的提升，仍依赖开发者的主动操作与监督；而Coding Agent则通过任务自主执行与异步协作，定位在功能或产品的完整交付，代表着软件开发从工具辅助向智能协作的跃迁。

2.3.1 核心观点总结

AI辅助开发工具形态演进：从最初的简易编码辅助，到对话式代码助手，再到智能开发助手和Coding Agent，产品形态经历了从局部任务支持到跨模块自主执行的演进。

扩展插件与AI原生IDE优势与局限：以GitHub Copilot为代表的IDE插件依托现有工具链快速渗透用户，干扰低，但交互粒度局限于单文件或局部代码片段；而AI原生IDE如Cursor和Windsurf提供完整的交互逻辑和界面自由度，同时沉淀用户行为数据，支持模型持续优化，更适合探索深度智能开发环境。

双线策略的实践：部分团队同时发展插件与AI IDE（如字节跳动的MarsCode与TRAE），兼顾快速用户覆盖和长期数据积累，为优化智能开发体验提供策略参考。

Coding Agent的自主性与协作性：Devin等Agent可独立规划、执行、调试任务，并访问互联网获取信息，实现异步自主执行，同时用户可随时打断和调整，实现协作式工作模式。

自主性带来的风险与管控需求：过度自由的Agent可能执行不可行任务或生成无效方案，因此设计需在自主性与可控性间平衡，如Replit Agent通过阶段性汇报与用户确认保障方向一致。

协作优先理念：现代Coding Agent通过人机协作处理重复性或复杂任务，同时保留用户对核心逻辑和架构决策的控制，实现开发效率提升与错误风险降低，为软件开发引入新的生产力范式。

AI辅助开发工具和Coding Agent类产品的定位差别：辅助工具定位于局部编码效率的提升，仍依赖开发者的主动操作与监督；而Coding Agent则通过任务自主执行与异步协作，定位在功能或产品的完整交付，代表着软件开发从工具辅助向智能协作的跃迁。

2.3.2 意图驱动开发形态初步形成

如果说大模型赋能开发工具链的重构主要体现为工具形态的演进与功能边界的拓展，那么更深层次的变化则正在开发交互方式中发生。随着模型在理解自然语言和复杂上下文方面能力的提升，**开发者与AI的关系正从指令—执行逐步走向意图—协作**。这意味着开发活动不再仅仅依赖明确的编码操作，而是越来越多地通过表达目标和意图来驱动。在这一趋势下，意图驱动的开发形态初步形成，其雏形主要体现在三个方面：意图表达、任务拆解与流程编排。

在这些初步实践的基础上，意图驱动开发的雏形不仅显现出当前特征，也为未来的发展方向提供了参考，主要体现在以下几个方面：

1）多模态与自然交互深化

除了文本指令，未来开发者还可以通过图形界面、设计稿、语音、操作演示等多模态方式表达意图，降低与AI交互的门槛。AI能够综合这些输入生成准确的执行结果，使意图表达更自然、开发体验更顺畅。这将进一步推动意图驱动从实验性探索走向广泛应用。

2）更强的上下文感知能力

未来AI工具将不仅理解单个代码片段或模块，而是能够掌握整个代码库、业务逻辑、架构约束和项目规范，实现更精准的任务拆解与代码生成。这将提升开发效率，减少错误和返工，同时增强Agent处理复杂项目的能力。

3）自主性与可控性的平衡

未来Agent将探索更灵活的自主执行策略，能够在后台异步执行任务，同时保持用户对核心逻辑、任务优先级和最终结果的掌控。通过可控的自主性，开发者可以专注于设计和决策，AI负责执行重复性或复杂性高的操作，从而实现更高效、可靠的人机协作。

4）流程闭环与智能优化

未来的意图驱动开发将不仅限于代码生成，还可能覆盖整个开发流程，包括环境配置、依赖管理、自动测试和部署。AI Agent将能够智能优化流程，自动选择最佳执行策略，实现从需求到可交付产品的端到端闭环开发。

关键观点

意图驱动开发正逐步从实验性探索向系统化演进迈进，通过多模态交互、任务拆解与流程闭环，将开发者与AI的关系从指令—执行转变为意图—协作，为更高效、可控和智能化的软件开发奠定基础。

2.3.2 核心观点总结

开发交互方式变革：大模型使开发者与AI关系从指令—执行转向意图—协作，开发不再局限于编码操作，而是通过表达目标和意图驱动。

意图表达多样化：自然语言和多模态输入（如设计稿、界面原型）使开发者可以以文字或视觉方式传达需求，降低交互门槛。

任务拆解智能化：规划模块、分层规则和人机协作模式支持将宏观目标分解为可控任务，确保执行规范化与结果符合预期。

流程编排自动化：AI可与CI/CD及协作平台集成，实现多环境、多阶段任务的自动化执行，并支持优先级和成本管理，保持用户对流程的掌控。

未来发展方向一—多模态与自然交互：未来AI将整合文本、图形、语音、操作演示等多模态输入，使意图表达更自然、开发体验更顺畅。

未来发展方向二—上下文感知能力增强：AI将理解整个代码库和业务逻辑，实现精准任务拆解与生成，提升复杂项目开发效率并降低错误率。

未来发展方向三—自主性与可控性平衡：Agent将实现可控自主执行，后台处理复杂任务同时保持开发者对核心逻辑和最终结果的掌控。

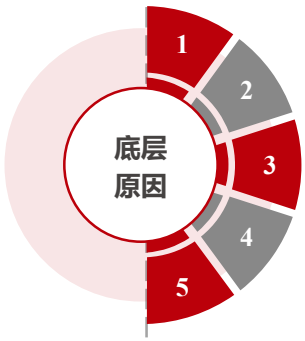
未来发展方向四—流程闭环与智能优化：意图驱动开发将覆盖从需求到交付的全流程，AI智能优化执行策略，实现端到端闭环开发。

2.3.3 智能能力嵌入开发流程的挑战

在前一节中，我们看到意图驱动开发的雏形已经初步形成，开发者通过自然语言、多模态输入以及Agent的任务拆解和流程编排，能够更高效地完成开发活动。然而，尽管这些实践展示了AI在提升效率和降低表达成本方面的潜力，将智能能力真正嵌入完整的开发流程仍面临诸多挑战。这些挑战不仅关乎技术实现，还涉及权限、信息获取、模型规划能力以及人机协作的边界。

图 软件开发核心环节的典型应用和难点分析

为深入理解这些挑战的本质，我们需要追溯到底层原因。虽然意图驱动开发已经在实践中初现雏形，开发者可以通过自然语言、多模态输入以及Agent的任务拆解和流程编排提升开发效率，但将智能能力真正嵌入完整的开发流程仍面临多重制约。分析这些底层因素，有助于厘清AI在权限控制、信息获取、模型规划能力以及人机协作边界等方面的局限，为后续优化和落地提供参考。



不确定性与模糊需求处理能力不足

AI难以处理探索性设计、方案权衡及模糊或冲突的需求，无法像人类开发者一样综合判断优先级和整体影响。这限制了其在设计规划和需求分析环节的应用。

复杂推理与泛化能力有限

在现阶段，AI在面对多层次决策任务、执行异常以及全新的用户需求时，缺乏足够的推理能力和灵活性，难以完全自主回溯错误或重新规划方案，从而影响开发流程的连续性和效率。

上下文与全局认知受限

当前AI模型在理解单个文件或模块时表现良好，但对整个代码库、业务逻辑、架构约束的掌握仍有限。这导致生成方案可能在局部最优，但全局欠佳，也影响任务拆解和跨模块协作的效果。

信息获取与环境感知不足

AI获取信息的能力远低于人类全局感知。人类可以随时查阅文档、网络资料或其他工具获取支持，而AI在权限、接口或工具链限制下难以达到同等水平，限制了其在复杂任务中的独立执行能力。

操作权限与系统桥梁不完善

AI要真正执行开发任务，需要访问代码库、依赖管理、编译环境、测试工具以及部署系统等。如果权限不足或AI与开发环境、操作系统之间缺乏完善桥梁，即便模型具备强大理解和规划能力，也无法实现端到端执行。这直接限制了自主性、流程闭环能力，以及在复杂工程中的实用性。

图 对开发流程中智能能力嵌入难点的底层原因分析

针对现阶段AI在软件开发中的应用挑战，业界已在尝试多种解决方案以弥补能力不足。以最底层的操作权限与系统桥梁不完善为例，ComputerUse、BrowserUse、MCP等进一步拓展Agent在底层系统的操作范围和环境感知能力，使其能够安全、高效地访问代码库、依赖管理、测试工具和部署环境。

除此之外，对于不确定性与模糊需求能力不足的挑战，以Cursor的Agent mode和Replit Agent为代表的产品，也在探索更积极主动的人机协作机制，通过逐步审批、关键决策询问和高层授权策略，让AI在处理模糊或冲突需求时仍保持受控并优化决策过程。

关键观点

尽管意图驱动开发与Coding Agent在实践中初现雏形并提升了开发效率，但AI在权限控制、信息获取、复杂规划和人机协作等底层能力上的局限仍限制了其在完整开发流程中广泛应用。

2.3.3 核心观点总结

意图驱动开发潜力：开发者可通过自然语言、多模态输入及Agent的任务拆解和流程编排提升开发效率，但在完整开发流程中嵌入智能能力仍存在挑战。

设计与规划局限：AI在需求分析、架构设计和方案评估中，受限于不确定性管理能力和全局架构认知，难以在模糊需求或多方案权衡中提供最优决策。

代码实现与维护限制：AI可能不了解项目全局上下文，跨模块交互处理能力不足，生成代码的可维护性和可读性较差，长期可持续性受限。

测试与交付瓶颈：AI生成的测试用例覆盖不足，异常处理能力有限，自动化流程集成不完全，仍需人工干预以确保可靠交付。

运行与优化挑战：AI在系统权限、跨环境执行和资源调度中能力受限，自主性与可控性平衡难以把握，过度自主可能带来风险。

底层制约因素：权限控制、信息获取、模型规划能力和人机协作边界是制约智能能力全流程嵌入的根本因素。

业界应对策略：通过扩展底层系统访问权限（如ComputerUse、BrowserUse、MCP）和主动协作机制（逐步审批、关键决策询问、高层授权策略），提升Agent在模糊需求和复杂任务中的可控性和效率。

2.4 全民开发时代的到来

在前两节中，我们系统梳理了大模型对开发工具链的重构、意图驱动开发形态的初步形成，以及智能能力在完整开发流程中嵌入所面临的技术与协作挑战。可以看到，AI正在逐步改变开发者的工作方式，使开发活动从传统的手动编码，转向以意图表达、任务规划和协作执行为核心的新范式。然而，这种技术驱动的变革不仅影响开发效率与流程，也在根本上扩展了开发者的概念边界。

随着工具能力的提升和使用门槛的降低，越来越多非专业群体开始能够参与应用构建和创新实践，这意味着软件开发正在迎来一个全新的群体——**全民开发者**。面对这一趋势，我们需要从全民开发者是如何加入软件开发说起，并进一步探讨面向全民开发者的产品功能和服务设计原则，为全民开发者提供可触达、高效和低门槛的支持环境。

2.4.1 开发者内涵持续外延拓展

从近期Lovable、v0、Bolt等产品的快速走红，以及氛围编码（Vibe Coding）概念的兴起，我们在软件开发的道路上，看到了越来越多的新兴开发者的出现。这也意味着软件开发正在经历一次重要的转折：**开发者的身份边界正被重新定义。**过去，开发者主要指具备专业编程技能、能够掌握复杂技术栈的工程师；而现在，随着AI辅助工具与自然语言交互的不断成熟，更多非专业群体开始被纳入开发者范畴。

设计师可以借助AI自动生成界面与交互原型，运营人员能够通过自然语言快速搭建活动页面或小工具，学生也可以在学习过程中以AI为辅助，直接将创意转化为可运行的应用。换句话说，**应用构建正从少数专业人士的专属技能，逐渐演变为一种大众化的创造方式。**与此同时，**在工具的支持下，全民开发者与专业开发者之间也开始形成新的分工关系：**前者更侧重于创意表达与业务需求转化，后者则在复杂逻辑、性能优化与工程化落地方面继续发挥关键作用。

推动这一转变的，不仅是模型本身的能力提升，更是工具在交互与体验设计上的创新。具体而言，当前涌现的面向全民开发者产品普遍具备以下几个关键特征。

自然语言与多模态交互

让全民开发者不必必须学习语法和框架，仅基于自然语言的描述或者原型图的展示，就可以将模糊的意图转化为可执行的逻辑和应用

低沟通门槛　快速转译

渐进式协作

借助对话式迭代，AI可以不断追问、澄清、补充上下文，这对于专业开发需要明确需求文档的开发方式是一种重要的补充，使非专业用户即使在需求尚不完整时，也能逐步推动创意落地。

交互式需求澄清　递进式构建

即时部署与可视化

用户能快速看到代码结果并即时上线；与传统的编译、打包和部署流程相比，这种方式让创意的实现更加直观和高效。

即时反馈　快速迭代

活跃社区与丰富模板

平台沉淀了丰富的模板和案例，帮助用户少走弯路；不同于从零开始的传统开发，这让入门与复用变得更加轻松。

资源复用　学习与互助

除此之外，我们也看到了**面向不同细分领域的工具**，例如擅长原型和设计体验的Lovable、面向设计师和前端开发者的v0和bolt.new，专注Web应用、网页游戏和教育的Div-idy。这也意味着，无论是设计师、学生、运营，还是创业者与教育者，都即将能在更合适、更便捷的场景中表达创意并找到合适的工具将其落地。

2.4.1 核心观点总结

开发者边界正在外延化：AI辅助工具和自然语言交互使非专业群体也能参与应用构建，开发者身份从专业技能持有者扩展为更广泛的大众群体。

全民开发者与专业开发者形成新分工：前者侧重创意表达和业务需求转化，后者专注复杂逻辑、性能优化及工程化落地。

工具创新推动开发门槛下降：自然语言与多模态交互、渐进式协作、即时部署与可视化，使非专业用户能够快速将模糊意图转化为可执行应用。

社区与模板助力学习与复用：活跃的社区和丰富的模板降低了入门成本，支持全民开发者高效获取参考和复用现有方案。

细分领域工具满足多样化场景：不同工具针对设计、前端、Web应用、教育等领域提供专业化支持，使全民开发者能够在最适合的场景中实现创意。

2.4.2 全民开发者需要的产品功能和服务

在谈论全民开发者所需要的产品功能和服务之前，有必要先对当前AI Coding产品的整体格局进行梳理。**从专业开发者—全民开发者以及Copilot—Agent两个维度切入**，我们能够更清晰地看到不同类型产品的定位、功能特点以及适配人群，从而为后续分析全民开发者所需的产品与服务奠定基础。

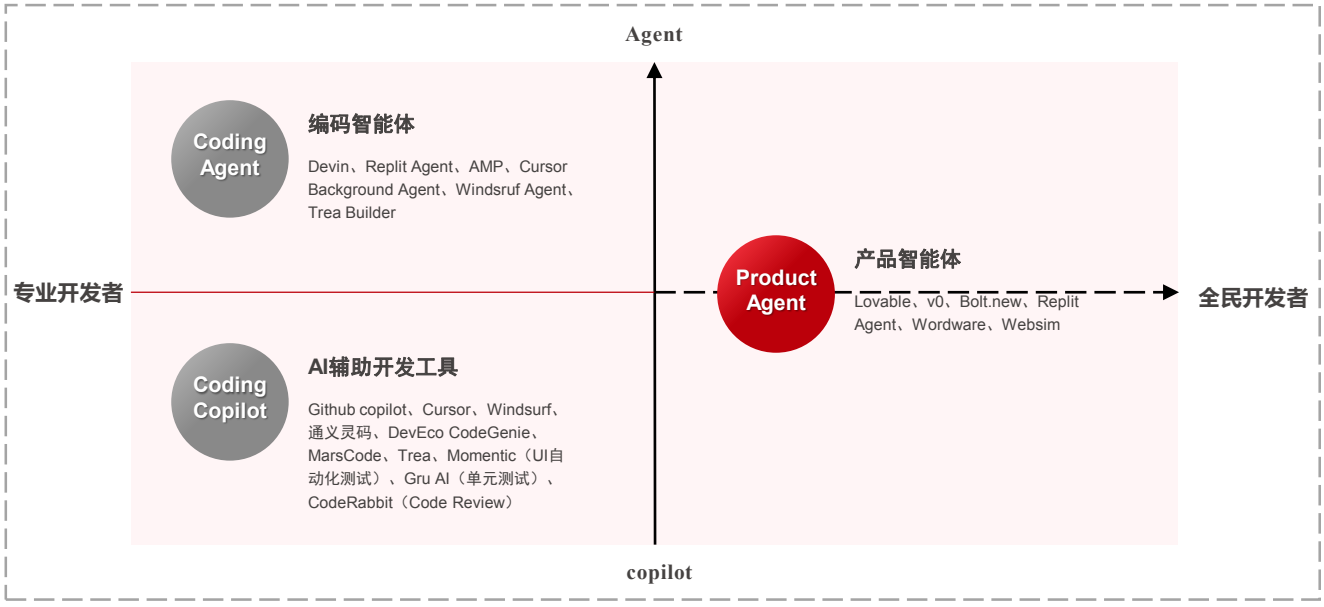


图 AI Coding产品的格局分析

从现阶段对产品智能体类型的市场反馈来看，尽管这类产品目前在交互体验和代码生成能力上取得了显著突破，但行业普遍认为，这类工具目前更适合用于**UI原型制作和创意验证**，而不适合承担复杂逻辑的生产环境开发。同时，在安全性、Bug修复、性能优化、资源消耗及成本管控等后续运维环节，这类平台仍面临显著挑战，这意味着在这些环节，产品仍然需要专业开发者介入保障稳定性与可靠性。

在以上内容的基础上，我们来看全民开发者需要的产品功能和服务。与前文相对应的是在产品细节上，工具需要支持**自然语言与多模态交互**，降低非专业用户的学习门槛，通过**渐进式协作与模块化架构**，让用户从模糊想法逐步落地，并保留专业开发者可介入优化的空间。除此之外，对于上线应用的后续**安全和质量保障**也需要有相应的功能支撑。

社区和生态方面是该类产品需要重点考虑的因素。就像人人都是创作者时代下的剪辑工具中内置的剪辑模板一样，全民开发者工具也需要提供**可复用的项目模板、组件库和示例项目**，让用户能够快速上手，无需从零构建。活跃的社区能够提供**实时交流、答疑和协作机会**，帮助用户在创意落地过程中获得反馈和支持，降低学习曲线，增强平台粘性。这种社区驱动的生态不仅加速了用户成长，也促进了模板、扩展和插件的持续积累，从而形成自我循环的长期价值体系。

在体验与服务设计层面，工具应提供**即用即走的操作体验、一键部署与商业化接口**，让全民开发者可以快速将创意转化为可交付成果。同时，通过**学习路径与教育嵌入**，在使用过程中潜移默化培养用户的编程思维，并提供清晰、可预测的收费模式，让用户感受到直接价值。

综上所述，全民开发者时代的到来，不仅依赖于AI模型本身能力的提升，更依赖于工具在**交互体验、分工协作、社区支持和可落地服务**方面的持续迭代。未来，随着技术成熟和生态完善，设计师、运营人员、学生甚至创业者，都能够在合适的工具和场景中，将创意高效落地，真正实现人人可开发软件创作愿景。

全民开发者需要的产品功能和服务

交互与产品细节

自然语言与多模态交互：自然语言与多模态交互：支持文字、语音、手绘或拖拽等多种表达方式，降低零学习门槛，让非专业用户也能快速描述需求并生成应用内容。
渐进式协作与模块化架构：通过对话式迭代和模块化设计，允许用户从模糊想法逐步走向成型应用，同时保留专业开发者介入优化核心逻辑的可能性。
安全与质量内置：提供默认的安全护栏、性能优化和合规检测，降低“黑盒”生成带来的风险，确保应用可在可控环境下运行。

社区与模板生态

模板与资源库：Lovable、v0和Bolt在官网都将社区放在首要位置，通过模板、示例项目和开源资源，新手用户可以快速上手，无需从零构建，从而降低从想法到原型的心理和技术门槛。
活跃的支持氛围：论坛、问答、教程和社区活动能够维系用户黏性，使平台生态自我循环增长，形成长期的用户增长和内容积累优势。

体验和服务设计

即用即走的开发体验：工具界面直观，用户无需理解底层技术即可快速产出可交互原型。
一键部署与商业化接口：覆盖应用分发、支付、数据收集等环节，让非专业开发者也能实现价值转化。
收费模式探索：Lovable、v0和Bolt均使用了免费增值机制，通过免费额度/水印或操作次数的限制先吸引用户进行产品试用，再根据额外积分/消耗Token/任务次数等进行增值收费。但现阶段，全民开发者对积分、Token消耗相对不敏感，同时不同产品仍在探索收费模式，这意味着该类产品仍然需要强化价值感知，让用户在体验和成果产出上直接感知付费收益。

2.4.2 核心观点总结

AI Coding产品格局分析：通过专业开发者—全民开发者和Copilot—Agent两个维度，可以清晰识别不同产品类型、功能特点及适配人群，为全民开发者工具设计提供基础。

现阶段面向全民开发者的产品智能体工具适用性有限：AI产品在交互体验和代码生成上表现突出，但更适合UI原型和创意验证，复杂逻辑开发及后续运维可能仍需专业开发者介入保障稳定性与可靠性。

产品智能体的核心功能需降低学习门槛：全民开发者工具应支持自然语言和多模态交互，通过渐进式协作与模块化架构，使用户能将模糊创意逐步落地，同时保留专业优化空间。

产品智能体的社区与生态加速用户成长：提供模板、组件库和示例项目，以及活跃的交流与协作社区，可帮助用户快速上手、获得反馈，形成长期自循环价值体系。

产品智能体的体验与服务需面向快速落地：即用即走操作体验、一键部署、商业化接口、学习路径嵌入和可预测收费模式，让全民开发者能高效将创意转化为可交付成果。

工具与生态共同支撑全民开发者时代：AI能力提升只是前提，交互体验优化、分工协作设计、社区支持和可落地服务的完善，才是真正实现人人可开发软件创作的关键。

03

开发者技术及生态愿景2030

3.1 愿景总览

到2030年，移动开发将进入一个智能化、跨终端、生态协作的新阶段。多终端和多设备环境日益丰富，用户的注意力、数据和任务被分散在手机、平板、手表、车机等多种终端之中。单一设备的开发模式已经无法满足业务需求和用户体验的高度统一要求。

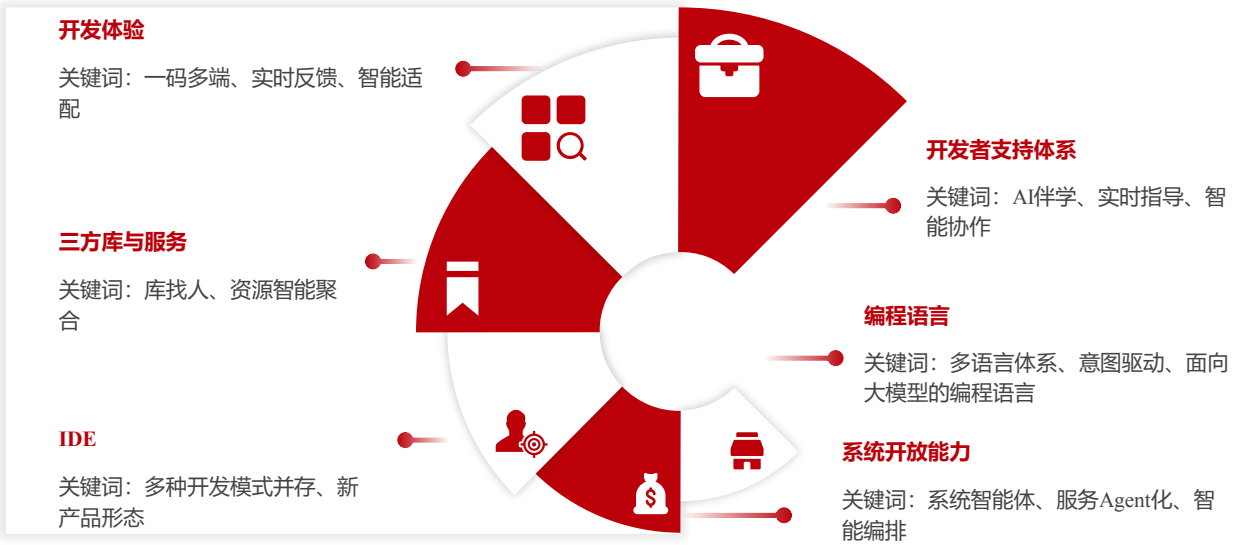
同时，在智能化的浪潮下，开发流程正经历重大转变：从以手工编码和单设备调试为核心，向意图驱动、AI辅助、多端协同的智能开发模式演进，使开发者能够在更高层次定义业务逻辑、界面和交互，并由系统和AI自动完成适配与优化。

除此之外，开发者角色日益泛化，设计师、运营、学生、创业者纷纷加入应用构建行列，未来的移动开发生态也需要为这部分用户提供相应的工具和服务。

在这样的背景下，未来移动开发的愿景不再仅仅是工具和语言的迭代，而是构建一个智能化、生态化、跨终端的开发环境。这个环境将从底层到应用全链路，全面提升开发效率、体验一体化和生态协作能力，使开发者能够专注于业务价值和创新，而非繁琐的适配与集成工作。

为达成这一目标，第三章节提出面向2030年的移动开发愿景，分别从编程语言体系、操作系统能力、IDE与开发工具链、三方库与服务生态、开发体验一体化，以及开发者支持体系六个维度进行阐述。

2030年，移动开发生态将在智能化、一体化与生态化三条主线的交织中演进，形成以意图为入口的人机共创、跨端一致、可自演进的智能生态体系。



3.2 编程语言体系的多样与融合

在移动开发的历史演进中，编程语言一直是一条最清晰的主线：**如何在开发高效、运行高性能与安全性之间找到平衡。**高效，代表开发者的生产力与工程可维护性；高性能，代表终端体验与算力利用的极限；安全，则是保障应用可信与生态稳健的基石。展望2030年，这三者将构成移动编程语言体系的铁三角，任何单一维度的偏重都无法满足生态的长期发展。

因此，在这种平衡逻辑下，**移动应用编程语言体系逐步形成了一个四层协作结构。**底层的C++、Rust等高性能语言，继续承担系统内核、AI推理和图形渲染等极限任务，是性能与稳定性的保障；中层的Swift、Kotlin、ArkTS等现代高效语言，成为大多数应用逻辑的主力军，以跨端一致性、并发安全与工程高效为主要特征；上层的自然语言接口结合向大模型的语义编程语言，则借助AI将开发门槛大幅降低，把更多非专业群体引入应用构建流程，扩展了开发者的边界。

除了效率与性能的平衡，未来更重要的变化在于**高效编程语言与自然语言的协作。**自然语言正在成为编程的新入口层。在这一演进过程中，语义编程语言将成为连接自然语言与高效编程语言的关键中间层——它以结构化语义为核心，将人类意图转化为可执行逻辑，使AI能够理解、编排并生成系统级代码。未来，通过自然语言进行的意图应用开发，也将借助语义编程语言的支撑，成为应用开发的重要途径之一。

这种格局将直接改变开发者的角色分工。**专业开发者**不再是逐行代码的编写者，而是与AI协作的架构师和质量守护者。他们的重点在于需求建模、框架设计、性能优化和安全审核。**全民开发者**则依赖自然语言接口，能够以低门槛方式构建应用，快速满足个性化的创意实现。到2030年，移动开发不再只是专业工程师的专属，而是成为全民都能参与的创作过程，语言体系的多样化正是这种趋势的技术基础。

关键观点

到2030年，移动编程语言体系将形成四层协作格局：底层以高性能语言保障极限算力与系统稳定，辅以语义编程语言，中层以现代高效语言承担开发主体，上层以自然语言扩展全民开发边界。整个体系将在高效—高性能—高安全的铁三角中不断寻求动态平衡

除此之外，移动应用生态的**安全风险**在未来五年可能更为复杂：一方面，AI自动生成的代码可能会引入潜在漏洞；另一方面，多设备、多模态的交互场景让攻击面进一步扩大。语言体系本身必须承担起更强的安全责任。现在我们已经能看到Swift、Kotlin、ArkTS、Rust等编程语言在并发安全、内存安全等方面加大力度。到2030年，语言本身会成为安全治理的第一道防线，而对于AI生成的代码，可能会诞生新的审查流程或专门的审查工具，也可能会诞生新的职业角色。

最终，到2030年，移动编程语言体系的演化不在于某一种语言的胜出，而在于能否在高效开发、极限性能与内生安全之间找到动态平衡。这种四层分工，将构建出一个既能快速迭代、又能支撑极限性能并保障安全可信的语言体系，推动移动开发范式的系统性重塑。

关键观点

编程语言将成为安全治理的第一道防线，以并发安全等安全特性为核心。围绕AI生成代码与跨端交互的安全风险，新的审查流程与工具将加速出现，为未来的移动开发生态提供全新的安全保障机制。

3.3 操作系统对外开放能力的升级

在移动开发生态的演进过程中，操作系统的对外开放程度始终决定了开发者能否充分利用设备的潜能。过去五年，iOS、Android与HarmonyOS的开放逻辑各有差异：iOS在稳定性与隐私优先的框架下谨慎开放API，如SiriKit、HealthKit、App Intents；Android更强调灵活与开放，通过ML Kit、Google Play Services为开发者提供广泛的接口能力；HarmonyOS则以分布式特性为差异化优势，提供AbilityKit等支撑多设备协同。虽然策略不同，但趋势一致：**系统能力正从单一设备硬件接口，逐步扩展到跨设备与智能化的系统级能力平台。**

第一，智能原生能力的嵌入将成为未来五年的核心特征

到2030年，主流操作系统不再仅仅提供传感器、摄像头、定位等硬件调用接口，而是直接开放AI原生能力：语音识别、图像理解、手势识别、情境感知等以系统级API形式提供。开发者可以直接调用系统自带的模型接口，享受本地推理带来的低延迟和高隐私保障；也可以利用端云协同，探索更多应用创新。这意味着应用创新的门槛大幅降低，AI最终将不再是差异化选项，而是系统能力的基础配置。

第二，跨端智能调度：让多设备无缝协同成为常态

未来的操作系统不再只负责单一设备资源的调度，而是具备多终端之间任务和服务的智能迁移能力。用户在手机上进行的会议，可以一键切换到平板继续，或者在车机上延续；一个游戏场景可以在电视和手机之间自由切换，数据与进度完全一致。对开发者而言，这类能力通过统一的系统接口开放，不再需要单独适配每一类设备，而是通过调用分布式调度API，让应用天然具备多端协同能力。

第三，操作系统将成为调度各应用/中枢完成智能体协作的调度中枢

操作系统不仅会提供AI模型接口，还会逐步演化为Agent管理平台。应用不再只是被动地提供功能，而是能够以Agent的形式被系统调用。例如，系统级的个人助理可以调用打车应用的Agent来完成出行任务，也可以调用文档应用的Agent来生成或修改文件。我们能看到iOS的App Intents、Android的AppFunctions API，以及HarmonyOS的意图框架与AI控件。到2030年，开发者只需按照平台规范封装应用Agent，就能与系统级智能体无缝对接，融入更广泛的任务编排与多模态交互流程中。操作系统成为**智能体的协调枢纽**，开发者的工作则是为应用定义可被智能体调用的接口。

第四，安全与隐私将成为操作系统开放能力的前置条件

当系统开始直接暴露AI原生能力和跨设备调度接口时，所涉及的数据范围和敏感度都会急剧增加。语音、视觉、位置信息以及多设备的实时状态，如果缺乏精细的权限管理，将带来巨大的隐私风险。因此，到2030年，操作系统在开放能力的同时，必须构建更细粒度的权限与安全模型，如基于场景的动态授权、AI生成结果的可信溯源等，才能让开放生态在安全边界内运行。

2030年，操作系统将升级为原生智能与多端调度平台

操作系统不再只是应用运行的环境，而是一个拥有AI原生能力、统一多端调度机制和Agent平台化支持的智能操作系统。对于开发者来说，这意味着开发体验从面向设备转变为面向服务，应用能力从单点功能转变为智能生态的一环。操作系统的对外开放能力，将成为塑造移动开发生态未来格局的关键变量。

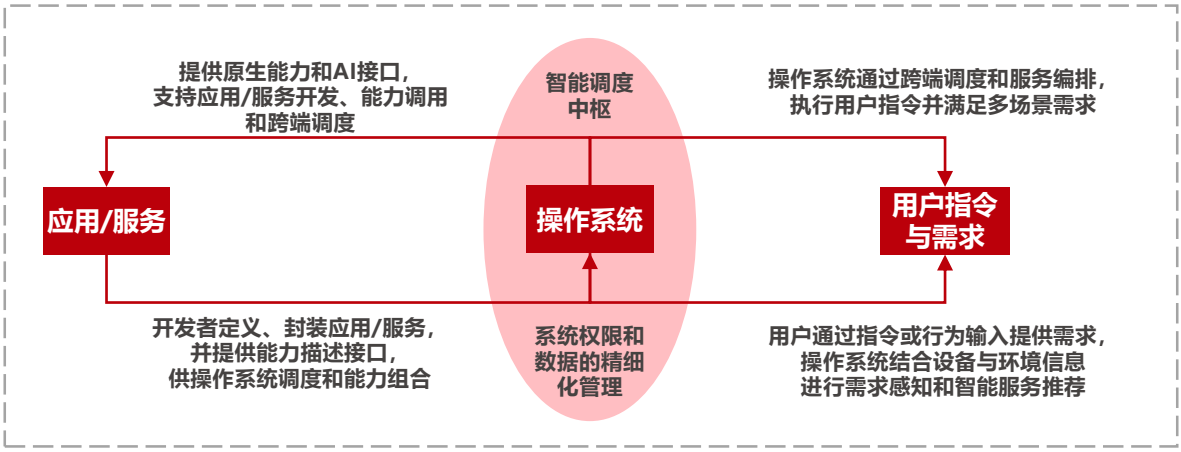


图 操作系统成为智能调度中枢，通过能力开放与智能理解，连接用户意图、应用服务与系统资源

3.4 IDE与开发工具链的AI化重构

在过去的移动开发中，IDE主要扮演编写与调试界面的角色。Xcode、Android Studio与DevEco Studio已经建立了较为成熟的工具体系，为开发者提供代码补全、调试、模拟器和包管理等功能。然而，它们依旧是工具化的存在，开发者必须亲自操作与决策。随着AI的加速嵌入，IDE的定位正在发生根本转变：它们正从单一的工作界面，进化为能够理解意图、主动协作、动态适应的**智能伙伴**。

一、意图表达开发形式崛起，多种开发模式并存

到2030年，开发者在IDE中的主要工作不再是仅依赖于逐行输入代码，而是可以通过自然语言或结构化意图描述需求。除此之外，IDE还可以支持多模态输入，比如开发者用草图描述界面、用语音表达业务逻辑，IDE自动生成界面代码。嵌入式大模型会自动生成框架、优化跨端适配、提示性能瓶颈，甚至主动提出需求澄清。IDE的角色从被动执行工具，转变为能理解上下文、参与决策的合作者，开发体验由此演进为意图驱动与对话驱动的新范式。

同时，不同的开发模式将在IDE内并行共生，形成长期共存格局。开发者可在任务执行过程中自由切换。除此之外，面向全民开发者，产品智能体可能会诞生新的产品形态。

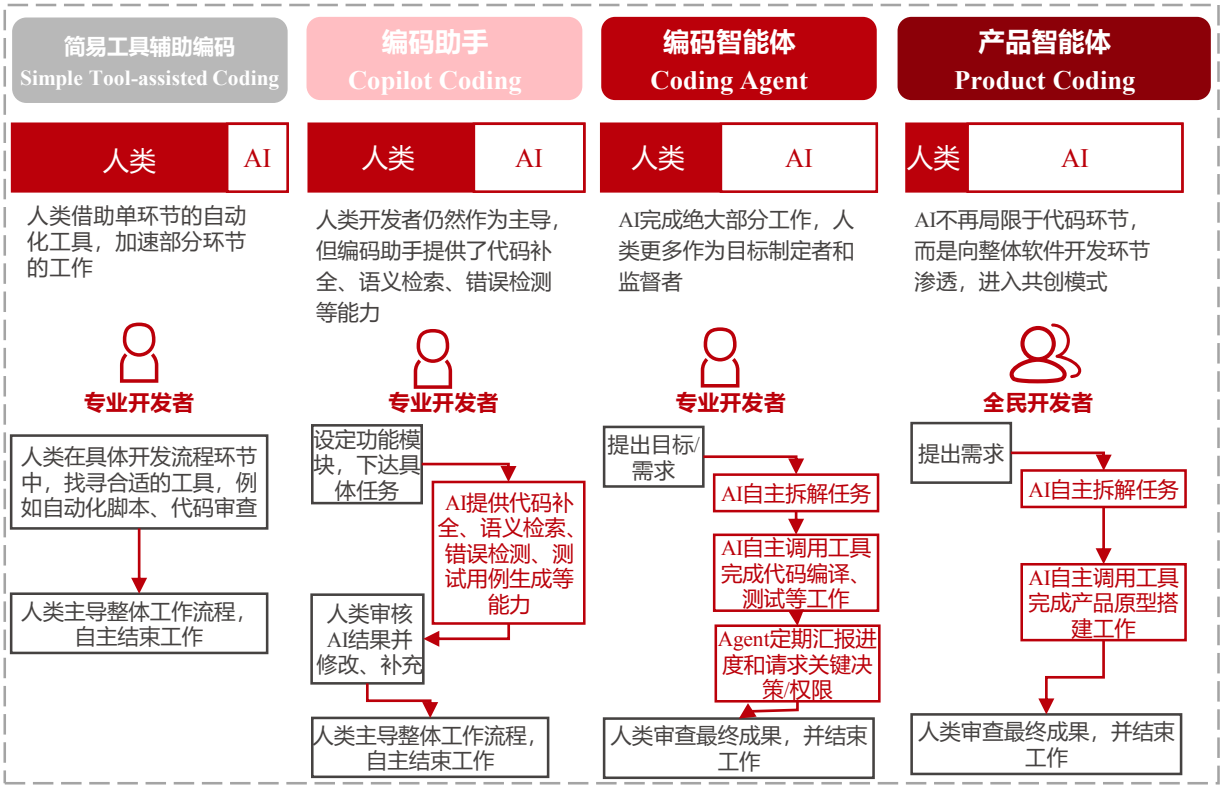


图 多种开发模式并存

二、AI将深度嵌入整个工具链，推动开发流程自动化

未来的IDE不仅在编码环节提供智能化支持，更会接管构建、测试、部署和运维等端到端环节。一个应用从需求定义到上线迭代，可能主要依赖AI的全链路自动化：它会为生成的代码自动创建测试用例，运行安全扫描与兼容性检查，上线后实时监控运行数据并触发优化。开发者从执行者转向监督者，在AI驱动的自我闭环DevOps体系中扮演质量与架构的把关人。

三、安全与可信将成为IDE AI化的必经考验

当AI深度参与开发，代码的正确性、合规性与可信度面临前所未有的挑战。AI可能引入潜在漏洞，生成代码可能涉及版权或敏感数据泄露。到2030年，IDE必须内置AI审计与溯源机制，对自动生成的代码实时检测，保证其符合安全与合规标准。与此同时，开发者也会承接安全监督员角色，他们不仅要建模与设计，还要保障开发成果的可靠与可信。

2030年，IDE将完成从工具到伙伴的跃迁

IDE及开发工具链不再是冷冰冰的界面，而是能够理解需求、对话交流、主动协作的虚拟同事；它通过AI化工具链实现全流程自动化，通过智能适配与协作重塑开发体验，通过安全机制确保可信运行。到2030年，IDE的智能化演化不仅会极大提升开发效率与创新速度，更会重塑开发者与工具、团队与生态之间的关系。

3.5 三方库与服务生态的聚合演进

在移动开发的生态体系中，三方库和服务扮演着加速创新的关键角色。从网络请求、图像处理、UI组件，还是支付、地图、消息推送等服务，开发者都可使用外部资源来避免重复造轮子。但过去五年，三方库生态也伴随着碎片化、版本冲突、兼容性与安全风险等问题，让开发者不得不在效率与稳定之间反复权衡。

第一，生态聚合：从分散代码到智能资源池

到2030年，三方库和服务不再以松散、独立的形式存在，而是通过**平台化聚合**形成统一的智能开发资源池。开发者能够在单一环境中快速检索、调用和组合能力，无需依赖繁杂的依赖管理工具，也不必担心版本兼容性。

实现这一愿景的路径可能是分层聚合：核心能力由官方库和认证三方提供标准化接口并内置于操作系统或IDE；其他第三方能力则通过插件或标准化API接入，使开发者能够像调用系统API一样便捷地使用可靠组件。AI将在其中扮演调度和组合的角色，自动完成跨库适配和拼装，但开发者仍需监督和验证结果，保障系统健壮性。

第二，开发者角色将从代码消费者转变为编排者

过去，开发者需要亲自去GitHub或Maven等平台寻找合适的库，手动解决依赖冲突，写大量glue code来拼装功能模块；未来，AI可以根据需求自动选择最合适的组件并完成集成。开发者的核心竞争力将转向**高效编排能力**：定义业务逻辑、监督AI拼装、保障系统架构健壮性。

第三，三方能力将向可调用、可组合的方向进化

库与服务不再只是孤立的功能模块，而是可被AI理解并自动组合的能力单元。开发者提出需求后，AI可检索、拼装并集成组件，同时在必要时进行跨库适配。开发过程从手工选择和拼装，逐步转向能力编排与监督，开发者重点在于评估与监管，而非低效整合。

第四，安全与可信将成为聚合生态的生命线

随着资源池集中化，潜在风险也更容易集中爆发。平台将内置严格的审计与溯源机制，确保所有组件经过安全检测，AI自动识别漏洞、后门和不合规代码，并在运行时通过可信执行沙箱控制行为边界。开发者调用能力时，可获得风险等级、合规报告和可信标签，从而避免黑箱依赖。

第五，聚合生态将推动商业模式升级

三方库和服务的聚合不仅是技术趋势，也将催生新的收益模式。部分高价值能力可能以API即服务提供，按调用量、并发数或功能等级计费；开发者通过订阅或即用即付方式快速接入复杂能力，而不必承担沉重的研发和维护成本。这将形成以能力为商品的开发经济，为开发者提供新的价值实现路径。

2030年，三方库与服务生态将从碎片化、松散的代码仓库，演进为安全、可信的智能开发资源池。

它将与操作系统开放能力、AI化IDE共同构成移动开发基础设施。开发者面对的不是分散的依赖，而是一座高度组织化、智能化的能力超市，能够即取即用、灵活组合，为跨端、多场景的智能化应用开发提供坚实支撑。尽管完全统一和零风险仍有技术挑战，但通过官方和认证三方的标准化接口、AI自动化和安全机制的协同，这一愿景在2030年是可达到的生态演进方向。

图 三方库与服务从人找库向库找人转变

3.6 多端开发一体化的持续追求

在多终端和跨平台开发趋势下，开发体验成为提升效率和保障用户体验的关键。不同设备（手机、平板、手表、车机等）具有各自的屏幕尺寸、交互方式和硬件特性，传统开发模式下，开发者需要针对每类设备单独实现UI、业务逻辑和性能优化，这不仅增加了开发成本，也容易造成体验差异。

到2030年，移动开发环境将实现多端开发的一体化，跨平台开发不再是多端手动适配的负担，而是一种自然能力。开发者可以在统一框架和IDE中定义应用逻辑、UI布局和交互流程，系统和AI将自动将这些定义适配到各类终端，保证多设备间功能和体验的统一。

第一，开发框架将实现多端适配，向一码多端演变

未来的开发框架将提供统一的编程模型和标准化组件库，自动处理多端差异。开发者定义一次逻辑，框架即可生成适配不同屏幕、分辨率、输入方式和硬件性能的实现版本。框架将内置性能优化策略，自动调整图像分辨率、动画帧率和GPU调度，保证低性能设备也能流畅运行。多端交互一致性通过统一事件模型实现，无论触摸、手势还是语音输入，都能保持相同响应逻辑。同时，对于多屏协作、实时数据同步或跨设备通知等复杂功能，框架将自动封装适配逻辑，让开发者无需重复实现，从而降低开发成本并提升体验一致性。

第二，AI将成为体验一致性的重要保障之一

AI将深度嵌入IDE和框架，实时监控和优化多端表现。它能够识别各终端的UI差异并自动调整布局和样式，确保视觉一致；通过模拟操作流程和性能环境，AI优化响应速度、动画流畅度和资源占用。AI还会自动生成跨端测试用例，发现潜在问题并提供优化建议或直接调整实现。更重要的是，AI通过分析用户行为和运行数据不断优化多端表现，使体验随时间持续提升，实现智能化、闭环的体验一致性保障。

总体来看，到2030年，多端开发一体化将成为开发常态。开发者无需为多终端调试和重复适配耗费精力，而是依托智能框架和AI，实现一次定义、多端应用、高度一致的用户体验，并保持高效开发和持续创新能力。

3.7 面向智能时代的开发者支持体系升级

到2030年，开发者支持体系将实现全面智能化升级，不再依赖传统文档、社区或培训课程，而是通过AI伴学、实时指导与协同共创，形成全流程、个性化、闭环的开发者支持生态。开发者在学习与开发过程中，将获得前所未有的效率提升和体验一致性。

第一，AI伴学成为开发者学习新途径

到2030年，开发者将拥有智能伴学助手，它不仅理解开发者的操作和意图，还能在开发的每一个环节提供实时、个性化的指导。AI能够根据开发者的技能水平、项目上下文和历史行为，主动推荐最佳实践、示范代码片段和设计模式，甚至可以直接生成UI布局、逻辑代码或接口调用示例。遇到问题时，AI可主动提示潜在解决方案、优化策略，并引导开发者完成调整和改进，使开发过程中的学习与实践无缝融合，实现边做边学、随时优化的智能伴学体验。

第二，实时指导贯穿开发全流程

AI将深度嵌入IDE和开发框架，实现对项目编码、组件组合、接口接入、多端调试、性能验证和上线部署的全流程实时指导。它能够监测潜在问题、性能瓶颈和安全隐患，并提供可执行的优化方案。AI会模拟不同终端和运行环境，自动校准开发方案，确保多端表现一致，降低反复试错成本，提升项目质量和用户体验。

第三，协同共创重塑团队协作

未来的开发者支持体系将成为团队协作枢纽。AI可以分析团队成员角色和任务，智能分配工作、协调版本合并和多端适配，实现高效协作。团队成员可在IDE或在线平台中实时共创，AI辅助处理冲突、推荐最佳方案，并记录协作经验形成动态知识库，使团队整体能力不断提升。

第四，智能知识库与生态服务融合

传统文档、教程和社区经验将演化为动态、智能化的知识库。AI能自动从官方文档、开源项目、社区经验和企业实践中提取、整理和更新知识，开发者可通过自然语言查询或案例示范快速获取解决方案。知识库与生态服务平台深度融合，使开发者能够即时调用技术方案和方法，实现开发、学习和创新的一体化闭环。

图 面向智能时代的开发者知识体系

总体来看，到2030年，开发者支持体系将实现从被动信息提供到主动智能服务的跃迁。AI伴学、实时指导和协同共创的模式，将使开发者快速掌握新技术以应对复杂场景，同时提升团队协作效率和创新能力，成为智能开发生态的重要基石。

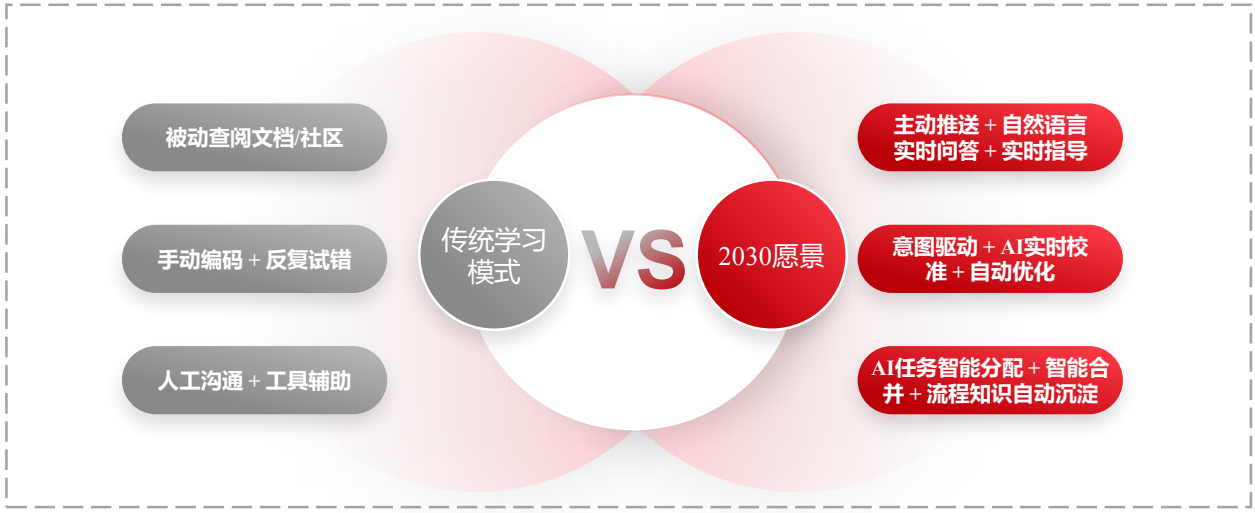


图 传统学习模式与2030愿景对比

极 客 邦 科 技 双 数 研 究 院

InfoQ 研究中心

InfoQ 研究中心隶属于极客邦科技双数研究院，秉承客观、深度的内容原则，追求研究扎实、观点鲜明、生态互动的目标，聚焦创新技术与科技行业，围绕数字经济观察、数字人才发展进行研究。

InfoQ 研究中心主要聚焦在前沿科技领域、数字化产业应用和数字人才三方面，旨在加速创新技术的孵化、落地与传播，服务相关产业与更广阔的市场、投资机构， C-level 人士、架构师/高阶工程师等行业观察者，为全行业架设沟通与理解的桥梁，跨越从认知到决策的信息鸿沟。

技术市场趋势洞察

- 市场份额追踪
- 细分市场分析
- 市场规模预测
- 市场分析模型输出

技术市场用户分析

- 用户规模评估
- 用户认知分析
- 用户决策分析
- 用户行为分析

数字化实践趋势分析

- 技术需求洞察
- 技术实践分析
- 应用规划建议
- 发展趋势研判

内容咨询：researchcenter@geekbang.com

商务合作：hezuo@geekbang.com

Geekbang. 极客邦科技

数字人才 KaaS 模式学习平台企业

- 极客邦科技，以“推动数字人才全面发展”为己任，致力于为技术从业者提供全面的、高质量的资讯、课程、会议、培训等服务。极客邦科技的核心是独特的专家网络和优质内容生产体系，为企业、个人提供其成功所必需的技能和思想。
- 极客邦科技自 2007 年开展业务至今，已建设线上全球软件开发知识与创新社区 InfoQ，发起并成立技术领导者社区 TGO 鲲鹏会，连续多年举办业界知名技术峰会（如 QCon、ArchSummit 等），自主研发数字人才在线学习产品极客时间 App，以及企业级一站式数字技术学习 SaaS 平台，在技术人群、科技驱动型企业、数字化产业当中具有广泛的影响力。
- 2022年成立双数研究院，专注于数字经济观察与数字人才发展研究，原创发布了数字人才粮仓模型，以此核心整合极客邦科技专业的优质资源，通过 KaaS模式助力数字人才系统化学习进阶，以及企业数字人才体系搭建。
- 公司业务遍布中国大陆主要城市、港澳台地区，以及美国硅谷等。十余年间已经为全球千万技术人，数万家企业提供服务。

InfoQ 促进数字技术领域知识与创新的传播

TGO 鲲鹏會 科技领导者同侪学习社区

极客时间 数字人才的移动知识库

极客时间 | 企业版 一站式数字技术学习 SaaS 平台