鸿蒙OS的跨设备空间计算软件Kit任务技术文件

空间计算
软件总体技术方案

2025年1月3日

目录

1. 简介

1.1 项目背景

近年来，随着计算机视觉（CV）、人工智能（AI）以及光学芯片技术的进步，增强现实（AR）和虚拟现实（VR）技术（以下统称为 XR 技术）和应用迅速发展。由于开发 XR 应用存在对空间定位、环境理解、空间交互等空间计算能力的需求，业界相继推出了提供 XR 能力的 SDK。例如：苹果推出的支持 iOS 和 visionOS 的 ARKit；谷歌推出的支持多种开发环境（Android、iOS、Unity、WebXR、OpenXR 等）的手机 AR 应用 SDK。

目前，XR SDK，以提供 3D 渲染和空间计算等能力为核心，正处于蓬勃发展的阶段，在未来，XR 有潜力成为操作系统（OS）的一部分原生能力，帮助设备实现对周围环境的精确感知和自然交互，提高用户人机交互体验。苹果、谷歌等企业已经在探索 XR 设备的操作系统以及 XR SDK 在专用设备上的扩展。

在 WWDC 2023 上，苹果推出了 Apple Vision Pro 和 visionOS，ARKit in visionOS 针对用户需求和硬件差异，对 iOS 原有能力进行大幅度的更改，为系统提供核心能力，是 visionOS 实现沉浸式场景的重要支撑。visionOS 是第一个为 VR 眼镜设计的操作系统，为它提供基础能力的 ARKit，对于开发跨平台提供 XR 能力的 SDK 具有较大的参考价值。

近期，谷歌发布了 Android XR OS 作为 Android 平台和生态系统在 XR 领域的扩展。Android XR OS 提供的 Android XR SDK 提供了多种将安卓 APP 移植到 Android XR 的方法，如使用与 Android Jetpack 类似的 Jetpack XR SDK 开发 XR 应用等。同时，ARCore 也增加了为 Jetpack XR SDK 设计的扩展，用于在 Android XR 设备上提供空间计算能力。

随着 HarmonyOS 的发展，华为公司也推出了 AREngine 等在 HarmonyOS 上完成空间计算和 3D 渲染等 XR 任务。为了解决未来的跨设备和多设备问题，

有必要将空间计算的部分能力嵌入系统服务，实现在不同形态设备上的流畅运行、多设备协作时数据的稳定传输，为原生应用和第三方应用提供更强的空间计算能力，为开发者提供更丰富和稳定的 XR 服务接口。

本项目旨在开发适用于 HarmonyOS 的支持跨设备的 AREngineKit，具体开发内容包括以下几个方面：

1) **Kit 开发**

实现系统 AR 功能服务化，完善进程管理、资源管理和 Session 生命周期管理、算法特性调度等功能。实现 Kit 内的数据传输通路，以确保传感器与 Kit 间的连接、传感器数据低时延下发给算法。

2) **API 开发**

提供 API 接口，以便上层 XR 应用集成使用。

3) **Sample 开发**

基于 Kit API 构建典型场景 XR Sample，展示空间计算能力。

1.2 竞品分析

1.2.1 ARKit

ARKit 最初设计为 iOS 系统提供 AR 能力，又为了提供能力给 visionOS，将 ARKit 分为了两部分：ARKit in iOS 和 ARKit in visionOS。

1) **ARKit in iOS**

在 2017 年 WWDC 会议上，苹果公司发布了 ARKit in iOS，经过多次改进扩展，为其增加了 3D 模型加载、多设备协同、QuickLook 等能力。ARKit in iOS 是业界较成熟的 AR SDK，附表的 api in ios 部分中总结了 ARKit in iOS 主要能力和 API，对 API 进行整理，可以看到其 API 主要分为四类：Session、

View、Anchor、ARConfiguration。根据其 API 和文档，可以总结出如下图所示的系统架构。

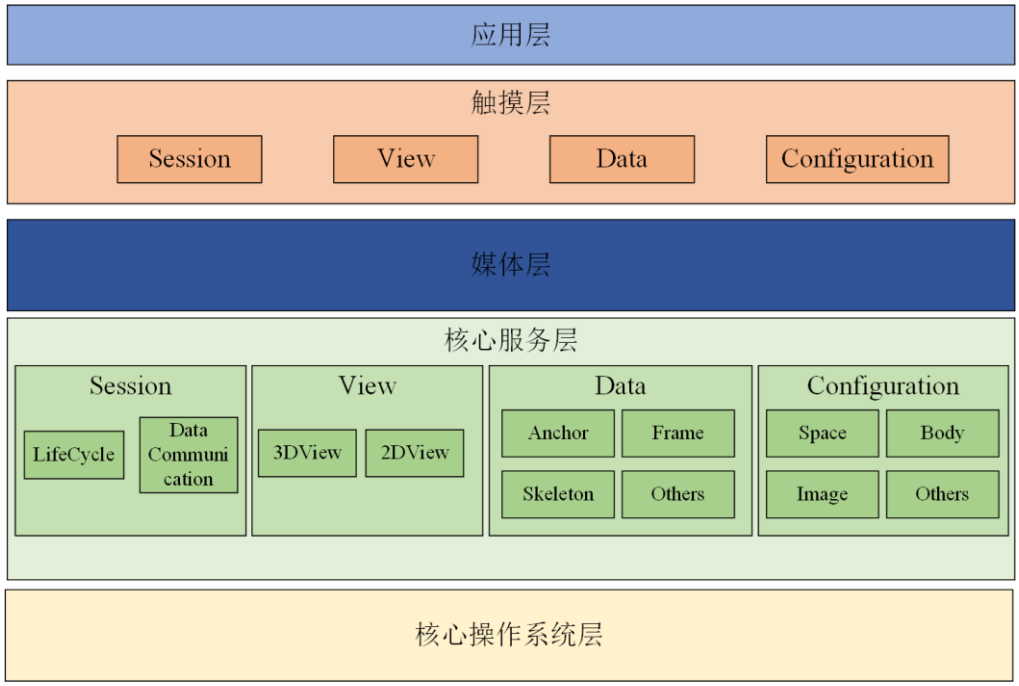

图 1.1 ARKit in iOS 系统架构图

其中，Session 用于管理生命周期，协调 View 和 Configuration 的执行，控制 View 和 Configuration 之间的数据传输；View 负责 2D、3D 视图界面的显示；Configuration 包含世界追踪、肢体检测等能力的配置；Data 是 Configuration 对环境进行分析得到的数据。

2) **ARKit in visionOS**

ARKit in iOS 虽然功能较强，但是由于它最初是为 iOS 专门设计的，并没有考虑在其它设备上的使用；为了更符合 Apple Vision Pro 的开发需求，苹果对 ARKit in visionOS 提供的能力种类进行了较大幅度的调整。而且在 visionOS 中，ARKit 为核心操作系统提供能力中，和 SwiftUI、RealityKit 一起支撑起 visionOS 的运行，推断依据将在后面给出[1]。

附表的 api in visionos 中总结了 ARKit in visionOS 主要能力和 API，对 API 进行整理，可以看到其 API 主要分为三类：Session、Anchor、Dataprovider。根据其 API 和文档，可以总结出如下图所示的系统架构。

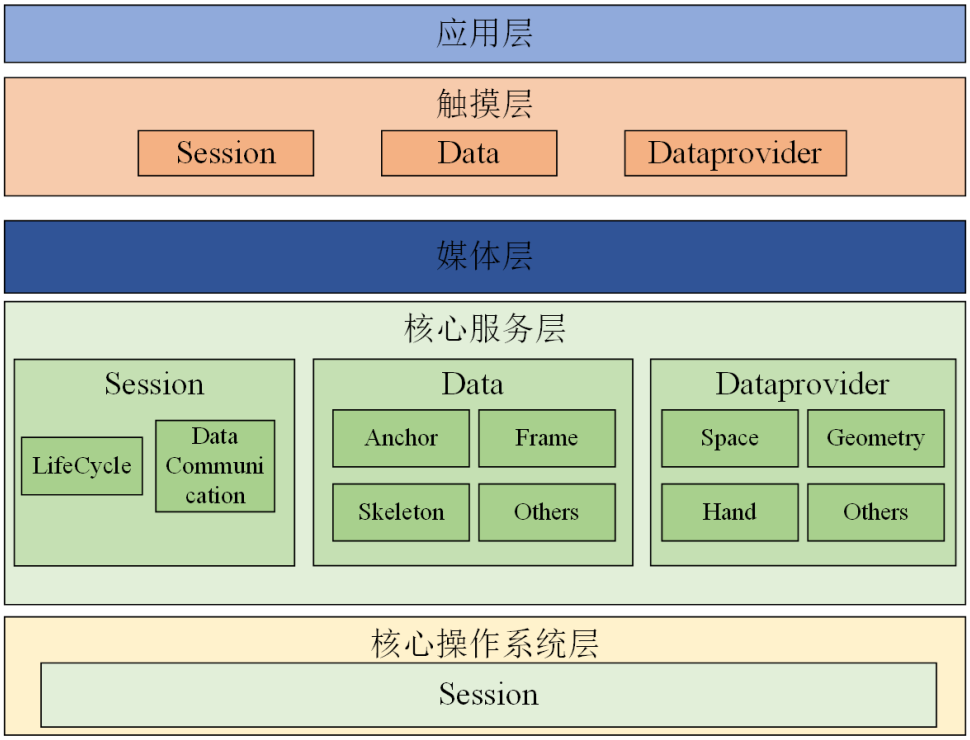

图 1.2 ARKit in visionOS 系统架构图

其中，Session 用于管理生命周期，控制 Dataprovider 的运行，将 Dataprovider 自动的传递到视图进行显示； Data 是 ARKit in visionOS 对环境进行分析得到的数据，包括：锚点、视频帧等；Dataprovider 用于提供世界追踪、平面检测等能力。

苹果官方文档提到大多数软件不需要直接调用 ARKit 即可实现，我们对比了 ARKit in iOS 和 ARKit in visionOS 二者，ARKit in visionOS 在前者基础上经过很大的调整，在架构上，移除了 View 部分，不再使用 ARKit 内部的 ARSCNView 等显示界面，更换为 SwiftUI 和 RealityKit 协同的方式，即使使用了 ARKit，结果也只以自动的方式显示出来，大多数情况下为通过 SwiftUI 和 RealityKit 隐式的使用 ARKit[2]。

另外，还有一些差异体现在二者的 API 上，下表为 Session 的对比，ARKit in visionOS 中对外开放 API 明显少于 iOS，只保留了核心的 run、stop、Event 等。同样的 ARConfiguration、Anchor 等都被简化，保留了核心的接口，如 Dataprovider 的 isSupported、State 等。这一方面是简化了接口的使用，另一方面减少了 ARKit 使用者对系统层的接触。

表 1.1 ARKit Session 对比

ARKit in iOS	ARKit in visionOS
run	init()
identifier:UUID	run
ARSession.RunOptions	stop
configuration	ARKitSession.Error
pause	requestAuthorization
delegate	ARKitSession.AuthorizationType
delegateQueue	queryAuthorization
ARSessionDelegate	ARKitSession.AuthorizationStatus
ARSessionObserver	events:ARKitSession.Events
add	ARKitSession.Events
remove	ARKitSession.Event
getCurrentWorldMap	description:String
createReferenceObject	
setWorldOrigin	
raycas	
trackedRaycas	
getGeoLocation	
currentFrame	
ARFrame	
captureHighResolutionFrame	
update	
ARSession.CollaborationData	
ARSessionProviding	

除了作为企业 API 的 BarcodeDetectionProvider[3]，Dataprovider 相比于 ARConfiguration，看似能力更顶层化了，如 RoomTrackingProvider、SceneReconstructionProvider 等，但这些大致都是构成 visionOS 共享空间（共享空间相关概念见[1]中 A spectrum of immersion 部分）所需的能力。

表 1.2 ARKit AR 能力对比

ARKit in iOS	ARKit in visionOS
ARWorldTrackingConfiguration	CameraFrameProvider
ARGeoTrackingConfiguration	PlaneDetectionProvider
AROrientationTrackingConfiguration	WorldTrackingProvider
ARPositionalTrackingConfiguration	HandTrackingProvider
ARBodyTrackingConfiguration	SceneReconstructionProvider
ARFaceTrackingConfiguration	ImageTrackingProvider
ARImageTrackingConfiguration	EnvironmentLightEstimationProvider
ARObjectScanningConfiguration	ObjectTrackingProvider
	RoomTrackingProvider

另外， RealityKit 以 ARKit 为基础，能够提供图像渲染、相机特效、动画、物理特效等能力，使得 ARKit in visionOS 中一些非核心能力可以剪除，如 Quick Look、HitTest、ObjectScanning 等。

1.2.2 ARCore

1) ARCore for Android

ARCore 是 Google 推出的用于设计 AR 应用、进行空间计算的 SDK，主要在 Android 和 iOS 上提供服务。

附表 api in android 中总结了 ARCore for Android 主要能力和 API，对 API 进行整理后，可以看到其 API 主要分为：Session、Data、EnvironmentEstimation 三种。根据其 API 和文档，可以总结出如下图所示的系统架构。

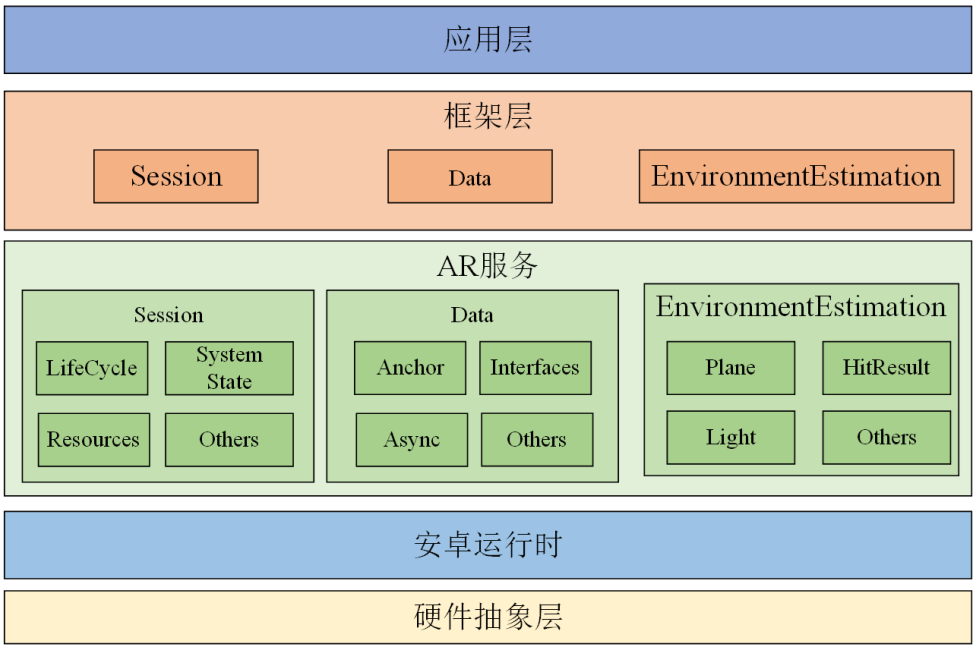

图 1.3 ARCore for Android 系统架构图

其中，Session 用于管理生命周期，控制 EnvironmentEstimation 中 API 的运行； Data 是 ARCore for Android 对环境进行分析得到的数据包括 Anchor、Future、Geometry 等；EnvironmentEstimation 中包含平面检测、光线估计等能力的配置。

2) ARCore for Jetpack XR

近期，谷歌发布了 Android XR OS 作为 Android 平台和生态系统在 XR 领域的扩展。Android XR OS 提供的 Android XR SDK 提供了多种将安卓 APP 移植到 Android XR 的方法，如使用与 Android Jetpack 类似的 Jetpack XR SDK 开发 XR 应用等。同时，ARCore 也增加了为 Jetpack XR SDK 设计的扩展，用于在 Android XR 设备上提供空间计算能力，其提供的空间计算能力较少：包括平面检测和 HitTest 两种。

附表 api in AndroidXR 中总结了 ARCore for Jetpack XR 主要能力和 API，对 API 进行整理后，可以看到其 API 主要分为三类：Session、Data、Perception、

EnvironmentEstimation。根据其 API 和文档，可以总结出如下图所示的系统架构。

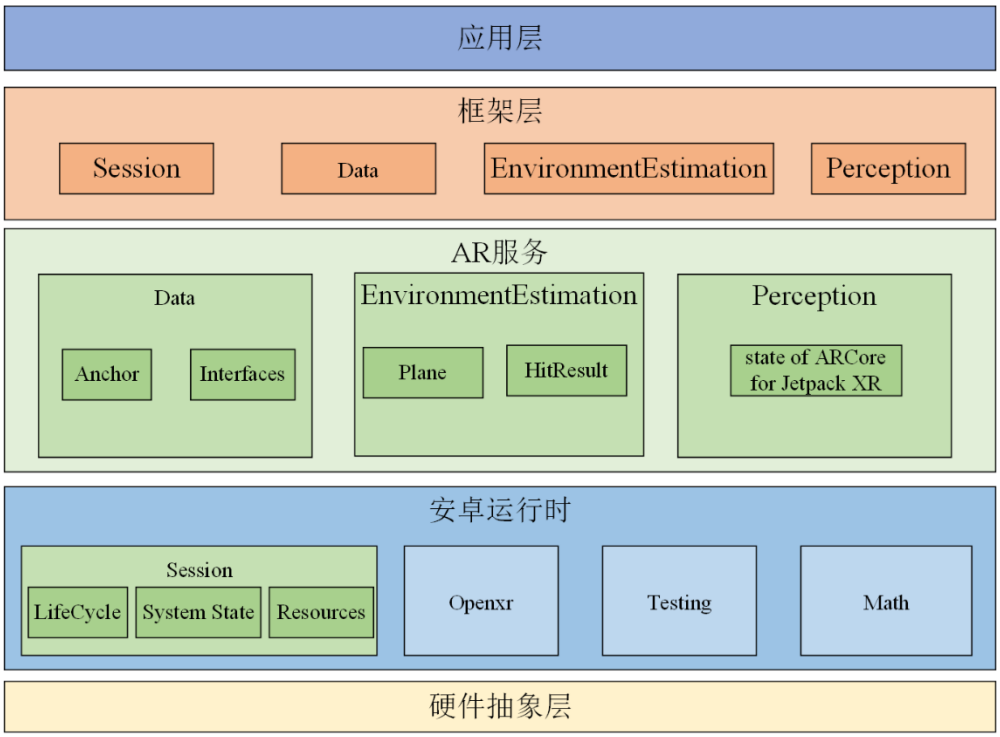

图 1.4 ARCore for Jetpack XR 系统架构图

其中，Session 用于管理生命周期，控制 EnvironmentEstimation 中 API 的运行； Data 是 ARCore for Jetpack XR 对环境进行分析得到的数据包括 Anchor、Future、Geometry 等；EnvironmentEstimation 中包含平面检测、命中测试等能力的配置、Perception 代表 ARCore for Jetpack XR 在特定时间点的状态。

与 ARCore for Android 的 api 相比，ARCore for Jetpack XR，大幅度减少了提供的 API 接口和能力数量，详见附表 api in AndroidXR 和 api in android，目前仅保留了 Plane 和 HitTest 两种空间计算能力，并且将 Session 转移到 AndroidXR 的 runtime 中运行[4]；另外由 SceneCore 针对空间计算能力的 API 进行了扩展，并且 ARCore 和 SceneCore 共用了 runtime 中的 Session[5]；而 Jetpack XR 的其他部分 Compose、View 等承担显示界面视图的任务，其中 Compose 是基于 SceneCore 构建而成的[6]。

1.2.3 总结

ARKit 和 ARCore 是业界内较为完善的提供空间计算能力的 SDK。随着 XR 技术和 XR 设备的发展，都推出自己的 XR 操作系统、设备和 SDK 扩展，二者都对原有 SDK 进行了修改，裁剪封装了原来用于手机应用开发的 SDK，并且将能力加入到系统服务，以提供适配眼镜设备的 SDK。根据 1.2.1 和 1.2.2 两节可以总结二者以下异同点：

表 1.3 ARKit ARCore 对比

差异	ARKit in visionOS	ARCore for Jetpack XR
	删改较轻，保留了较多 XR 能力	删改较重，只保留了 Plane、HitTest
	Session 仅 ARKit 使用	Session 放入 runtime 与 SceneCore 共用
	独占式，共享空间下，App 不能使用 ARKit	可共享的
相同	基于 ARKit/ARCore 使用其它 SDK 提供额外的空间计算能力	
	基于 ARKit/ARCore 为 UI 相关 SDK 扩展 XR 系统显示能力	
	手机和眼镜上两套 SDK 相差较大	

根据二者的异同点，我们想要设计一套不同设备共用的 AREngineKit，其主要特点为：不同设备通用，AR 能力可共享的。为满足以上特点，我们预期对已有的 AREngine 进行改进，鉴于与 ARKit in visionOS 的 Session API 有较高的相似性，可以 Session API 不变，保留已有的 AR 能力，并下沉到系统服务层，提供可共享使用的 Session 及 AR 能力，同时保证了 AR 能力 API 以及 UI 相关 SDK 的可拓展性。在改进过程中，针对 XR 场景下需要，为应对启动速度和资源占用问题，可能需要对 Session 和 AR 能力进行优化。

1.3 文档结构说明

本文档共分为 3 节，第 1 节为项目简介，主要介绍了项目的背景，对 XR SDK 的主要竞品进行了分析，提出我们要设计支持跨设备开发、将空间计算能力嵌入到系统能力的 ARKit; 第 2 节为 HarmonyOS 操作系统的介绍，主要介绍了 HarmonyOS 分布式架构和生态统一性的技术特点，还介绍了 HarmonyOS 下

AR 系统的独特优势以及 HarmonyOS SA 开发介绍；第 3 节为系统框架设计，主要介绍了本项目 ARKit 预期中的系统总体架构、模块划分与职责、AREngine 执行流程、代码目录结构以及模块间接口定义。

2. HarmonyOS 操作系统介绍

2.1 HarmonyOS 技术特点

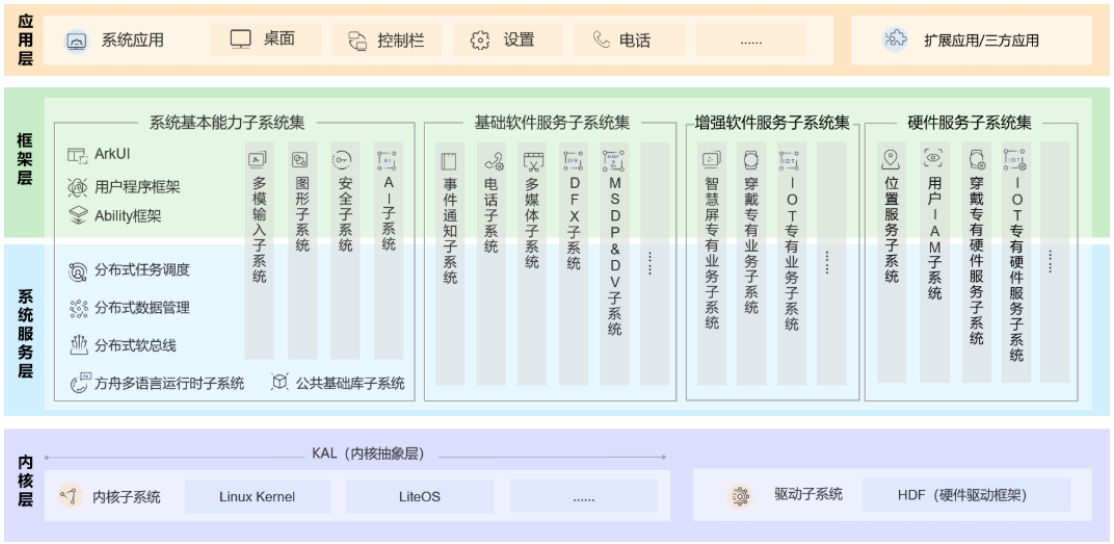

图 2.1 OpenHarmony 技术架构

HarmonyOS 是华为推出的一款面向全场景的分布式操作系统，基于 OpenHarmony 开发，OpenHarmony 技术架构如图 2.1。HarmonyOS 操作系统旨在为智能设备提供统一的操作系统平台。它支持多设备协同、跨设备交互以及一次开发多端适配的能力，能够覆盖手机、平板、智能穿戴设备、智能家居、车载设备等多种终端设备，满足全场景智慧生活的需求。

2.1.1 分布式架构的设计优势

HarmonyOS 的分布式架构是其核心技术特点之一，它通过软总线技术实现了跨设备的无缝连接和协同。以下是分布式架构的设计优势：

1）跨设备协同运行

HarmonyOS 支持多设备之间的资源共享和任务协同。例如，手机可以与平板协作，手机上的任务可以无缝切换到平板继续执行。

HarmonyOS 通过分布式虚拟总线（Distributed Virtual Bus）技术，将多设备的硬件能力进行整合与协同，从用户的角度来看，多个设备之间可以协同工

作，就像一个“超级虚拟设备”一样，这种方式使设备间的资源调度更加统一和高效。

2）分布式数据管理

数据在设备之间共享时无需手动同步，系统会自动完成数据的一致性管理。无论是文件、图片还是应用数据，都可以通过分布式架构实现实时同步。

3）分布式安全

HarmonyOS 提供了分布式身份认证和设备可信管理，确保数据在不同设备间传输时的安全性。

用户在一个设备上完成认证后，其他设备无需重复认证即可访问相关服务。

4）低延迟高性能

分布式架构通过软硬件协同优化，减少了设备间的通信延迟，提升了系统整体的性能和响应速度。

2.1.2 生态统一性

HarmonyOS 提供了一套统一的开发框架，使开发者可以通过一次开发实现多端适配。这种生态统一性极大地降低了开发成本，提高了开发效率。

1）统一的开发语言和框架：

HarmonyOS 使用 ArkUI 和 JS/eTS（增强型 TypeScript）作为开发语言，支持多端应用的统一开发。

开发者可以通过一套代码适配手机、平板、智能手表等多种设备。

2）多端适配能力：

HarmonyOS 提供了分布式 UI 框架，支持不同设备屏幕尺寸和交互方式的适配。

开发者可以通过自适应布局和多端组件，快速构建适配于多种设备的应用。

3）组件化设计：

HarmonyOS 的组件化设计使应用可以根据设备能力进行动态加载，避免不同设备间的重复开发。

开发者可以通过定义不同的能力包（Ability）来实现功能的灵活扩展。

4）生态协同效应：

HarmonyOS 提供了统一的分布式服务平台，开发者可以将应用能力开放给其他设备，实现生态协同。

例如，用户可以通过智能手表控制家中的智能家电，这些能力都可以通过一次开发实现。

2.2 HarmonyOS 下 AR 系统的独特优势

HarmonyOS 在 AR 系统的设计中，结合了其分布式架构和生态统一性的特点，提供了独特的优势，使其在 AR 应用场景中具有强大的竞争力。

1）多设备协同的分布式 AR

通过分布式架构，HarmonyOS 可以实现多设备间的 AR 数据同步。例如，用户在手机上创建的 AR 场景可以实时同步到平板或智能眼镜上，多个设备共享同一个 AR 空间。

此外，HarmonyOS 支持分布式计算，可以将 AR 场景的渲染任务分发到多台设备运行。例如，手机负责摄像头数据采集，平板负责场景渲染，提升了复杂 AR 场景的运行效率。

2）优化的硬件协同能力

硬件资源的整合使用：

HarmonyOS 可以将多设备的硬件资源虚拟化为一个超级设备，使 AR 应用可以整合多个设备的硬件能力。

例如，手机的摄像头与智能眼镜的显示屏协同工作，提供更沉浸的 AR 体验。

性能优化：

通过软硬件协同优化，HarmonyOS 提供了低延迟、高性能的 AR 数据处理能力。

例如，摄像头数据采集、场景识别、渲染等任务可以通过多线程和分布式计算加速完成。

3）一次开发多端适配的 AR 应用

统一的开发框架：

开发者可以通过 HarmonyOS 提供的开发工具，一次开发适配手机、平板、智能眼镜等多种 AR 设备。

例如，开发者可以通过 ArkUI 构建适配多设备的 AR 应用界面。

多设备的 AR 应用场景：

HarmonyOS 支持多种 AR 应用场景的统一开发，例如 AR 导航、AR 游戏、AR 教育等。

开发者可以通过分布式服务，将 AR 应用能力扩展到更多设备。

4）分布式安全保障 AR 数据隐私

分布式身份认证：

HarmonyOS 的分布式安全框架确保 AR 数据在多设备间传输时的安全性。

用户在一个设备上完成身份认证后，其他设备可以无缝访问 AR 应用。

数据隐私保护：

HarmonyOS 提供了数据隔离和权限管理机制，确保用户的 AR 数据不会被未经授权的设备访问。

2.3 HarmonyOS SA 开发介绍

OpenHarmony 是基于 HarmonyOS 的开源分布式操作系统，旨在为智能终端设备提供统一的软件平台。而 SA（System Ability） 是 OpenHarmony 中非常重要的组件，负责提供具体的业务能力。它是一种面向服务的能力开发方式，允许开发者基于分布式架构实现多设备协同的业务功能。

2.3.1 SA 的典型特点：

- 后台运行方式：主要用于处理后台逻辑，支持长期驻留内存。
- 按需启动方式：按照用户 APP 的调用请求，动态响应调用需求，启动不同的 SA 服务。
- 分布式特性：基于软总线支持通过分布式能力调用，服务可以在本地或其他设备上运行。
- 统一管理：由 SystemAbilityManager 对 SA 进行统一的管理
- 轻量化：专注于服务的能力提供，与界面无关。

- 灵活性强：专注于功能的实现，可以配合前台模块（FA）或者独立工作。

2.3.2 IPC 通信和软总线通信

2.3.2.1 IPC 介绍

IPC（进程间通信） 是操作系统中不同进程之间的一种通信机制，用于在相互隔离的进程间传递信息。由于 OpenHarmony 的分布式特性，其 IPC 通信不仅限于单个设备，还支持分布式 IPC，允许设备之间的进程进行通信。

在 OpenHarmony 中，IPC 是通过分布式软总线实现的，具体机制分为“本地 IPC”和“分布式 IPC”，本地 IPC 在同一设备上运行的两个进程之间进行通信，分布式 IPC 则支持跨设备的进程通信，底层使用 OpenHarmony 的分布式架构支持。

2.3.2.2 IPC 的通信机制

在 OpenHarmony 中，IPC 通信通常是通过 Ability Framework 框架实现的。其提供了用于进程间数据交互的能力。

其核心机制为：

1. **Proxy 和 Stub**：
    - **Proxy** 是客户端的接口，它提供访问服务的方式。
    - **Stub** 是服务端的实现，定义了服务的具体行为。
    - 通过代理和存根的结合，客户端可以像调用本地方法一样，调用远程服务的方法。
2. **中间商通信机制（类似 Binder）：**

    在 OpenHarmony 中，这一类都可以归为软总线。

    - OpenHarmony 的 IPC 底层基于 **Binder** 实现。Binder 是一种轻量级、高效的通信方式，广泛应用于 Android 和 Linux。
    - OpenHarmony 使用分布式软总线（Distributed Soft Bus）支持跨设备的 IPC 通信。允许开发者在不同设备上调用服务。

2.3.2.3 IPC 通信框图：

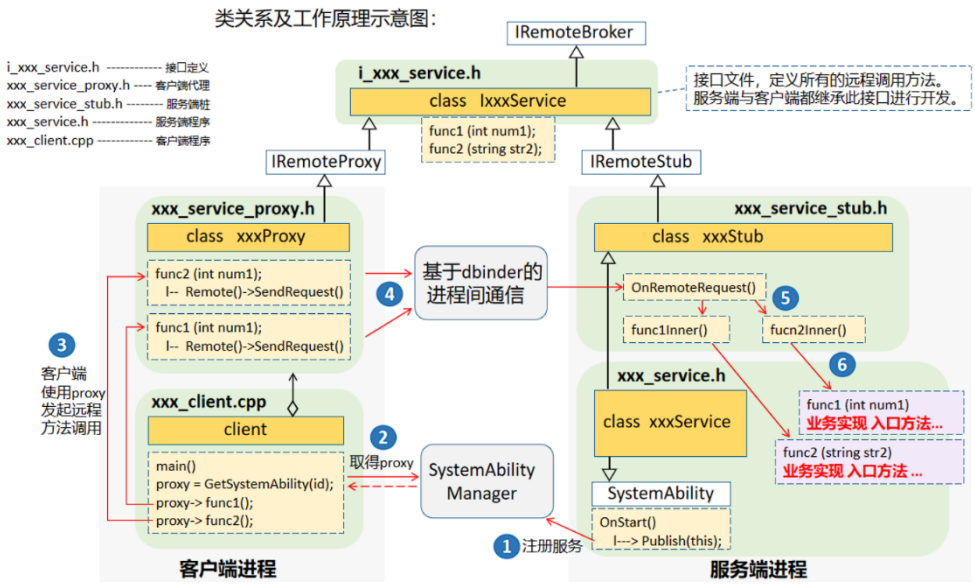

图 2.2 IPC 通信框图

2.3.3 SA 开发流程

- 注册 SA ID

  在 foundation/systemabilitymgr/samgr/interfaces/innerkits/samgr_proxy/include/system_ability_definition.h

  路径中配置 SA 服务的 ID

- 配置子系统部件编译

  在 build/subsystem_config.json 文件中定义 SA 的描述信息

- 配置设备子系统信息

  在 vendor/hihope/rk3568/config.json 中定义 SA 的描述信息

- 配置 SELinux 权限

  base/security/selinux_adapter/sepolicy/base/public/service_contexts

  base/security/selinux_adapter/sepolicy/base/public/service.t

  base/security/selinux_adapter/sepolicy/base/public/init.te

  base/security/selinux_adapter/sepolicy/base/public/type.te

  base/security/selinux_adapter/sepolicy/base/te/<your sa name>.te

在上述文件中进行 SeLinux 的 SA 相关安全配置

- 配置 SA 系统特权

  vendor/hihope/rk3568/security_config/high_privilege_process_list.json

  在上面的路径文件中配置 SA 的权限配置。

2.3.4 SA 启动过程

每次开机，init 阶段会读取配置<your sa profile>.cfg，找到<sa profile>.json 文件，通过 json 文件的指示，启动<your sa>以及配置中的相关服务。之后拉起服务动态库<your service name>.z.so，最后便进入运行阶段。

3. 系统框架设计

3.1 系统总体架构

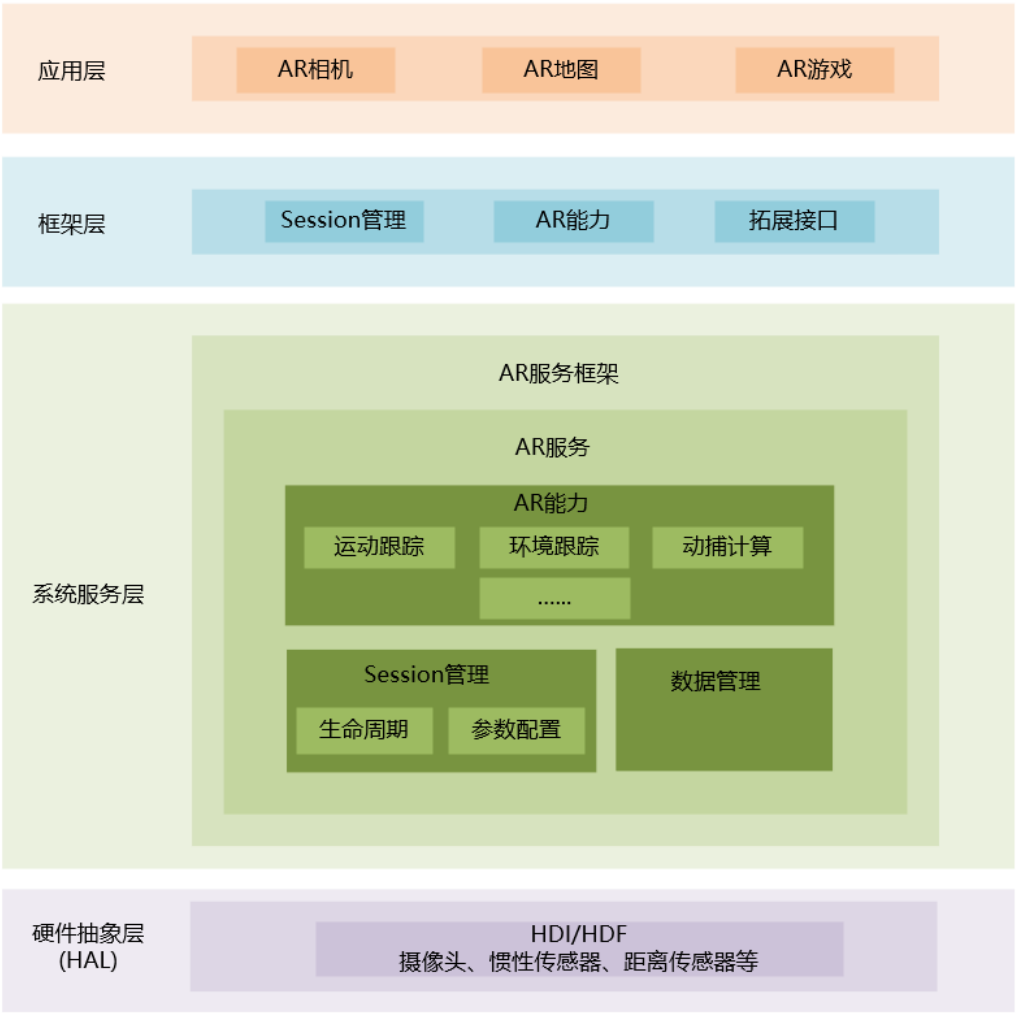

图 3.1 系统总体架构图

该 AREngineKit 整体架构设计主要分为四个层次，从下到上分别是硬件抽象层 (HAL)、系统服务层、框架层和应用层。其设计目的在于通过利用 HarmonyOS 层次化的架构，实现对 AR 功能的高效支持。

层次功能和关系：

1） 应用层

应用层直接面对开发者和最终用户，提供如“AR 相机”、“AR 地图”、“AR 游戏”等应用功能。这些功能主要通过调用框架层（如 Session 和 AR 能力接口）的高层 API 来实现。

应用层通过框架层提供的接口（通常为 SDK 或 API）与 AR 服务交互。调用的内容包括初始化 AR 会话、获取帧数据、查询跟踪结果等。

2） 框架层

框架层作为开发者接口层，封装了对底层服务的调用逻辑。Session 模块需要负责请求数据的封装和结果的解析，负责 AR 会话的生命周期管理，管理会话所需的配置（如跟踪模式、场景类型）。

AR 能力负责封装具体的 AR 功能，提供基于场景和算法的计算接口。AR 能力提供的接口由 Session 管理统一暴露给应用层。

3） 系统服务层

系统服务层是整个框架的核心，负责 AR 服务功能的实现，直接管理与硬件和算法相关的交互。主要包含：

- Session 管理：会话的生命周期、参数配置、输入输出管理。
- AR 能力：实现运动跟踪、环境跟踪、命中检测等功能。
- 数据管理：管理层次或模块间传输的一些数据，提供缓存等能力。

其中 AR 能力是实现具体 AR 功能的模块，如运动跟踪、环境跟踪、命中检测等，功能描述如下：

- 运动跟踪能力：通过获取终端设备摄像头数据，结合图像特征和惯性传感器（IMU），计算设备位置（沿 x、y、z 轴方向位移）和姿态（绕 x、y、z 轴旋转），实现 6 自由度（6DoF）运动跟踪能力。
- 环境跟踪能力：通过检测和跟踪设备周围的平面及语义，实现环境跟踪能力。环境跟踪能力包括：平面检测、平面语义、目标语义。

- 命中检测能力：通过命中检测（Hit Testing）技术能将终端设备屏幕上的兴趣点映射为现实环境中的兴趣点。命中检测以现实环境中的兴趣点为源，发出一条射线连接到摄像头所在位置，返回射线与平面（或特征点）的交点。通过命中检测能力，用户可以通过点击终端设备屏幕，选中现实环境中的兴趣点，与虚拟物体进行交互。

4） 硬件抽象层（HAL）

封装底层硬件（如相机、IMU 传感器、深度传感器）的驱动和接口，为上层服务提供统一的数据访问接口，隐藏硬件实现细节。

3.2 AREngineKit 执行流程

3.2.1 AREngineKit 启动流程

AREngineKit 可以按需启动，也可以随系统启动，对于不同设备，AREngineKit 的启动流程不同，具体启动流程如图 3.3 和图 3.4。

对于手机、平板等设备， AREngineKit 通常按需启动，当用户首次打开 AR 应用，AREngineKit 被拉起，运行在后台进程。而对于眼镜等 XR 设备，AREngineKit 属于核心功能，随系统启动而被拉起，始终运行在后台。用户程序通过 IPC 通信跨进程调用 AREngineKit 接口，多个应用可以共用同一个计算程序，而不需为每个进程在内存中创建一套 AR 能力算法。

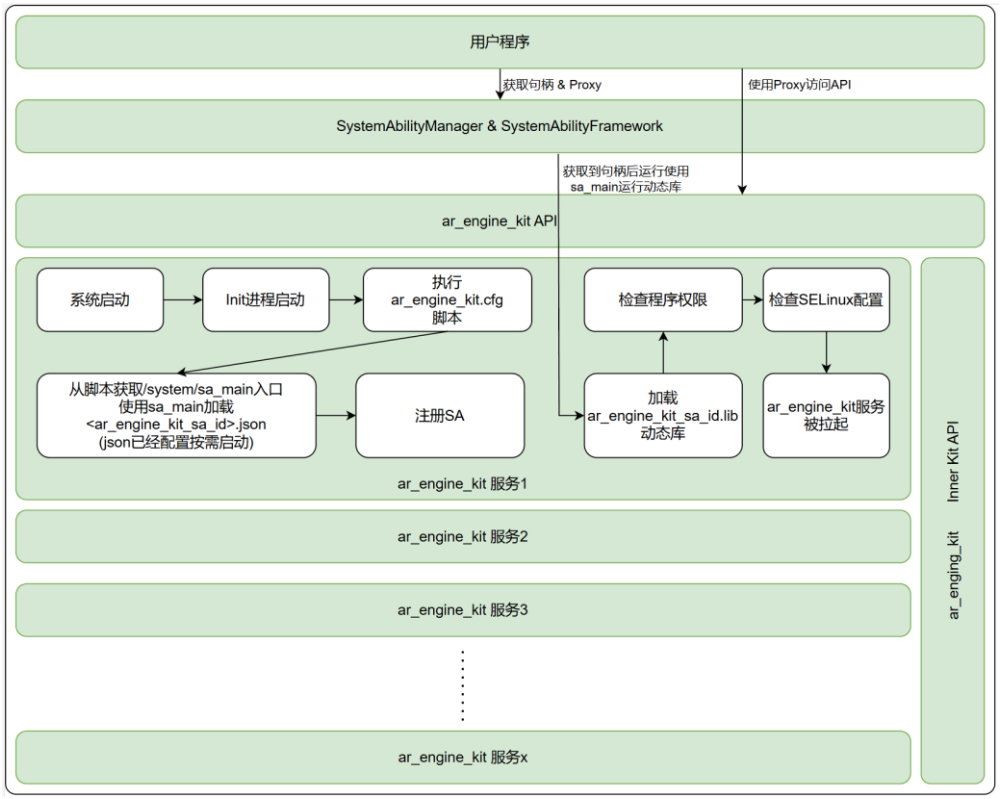

图 3.2 按需启动流程

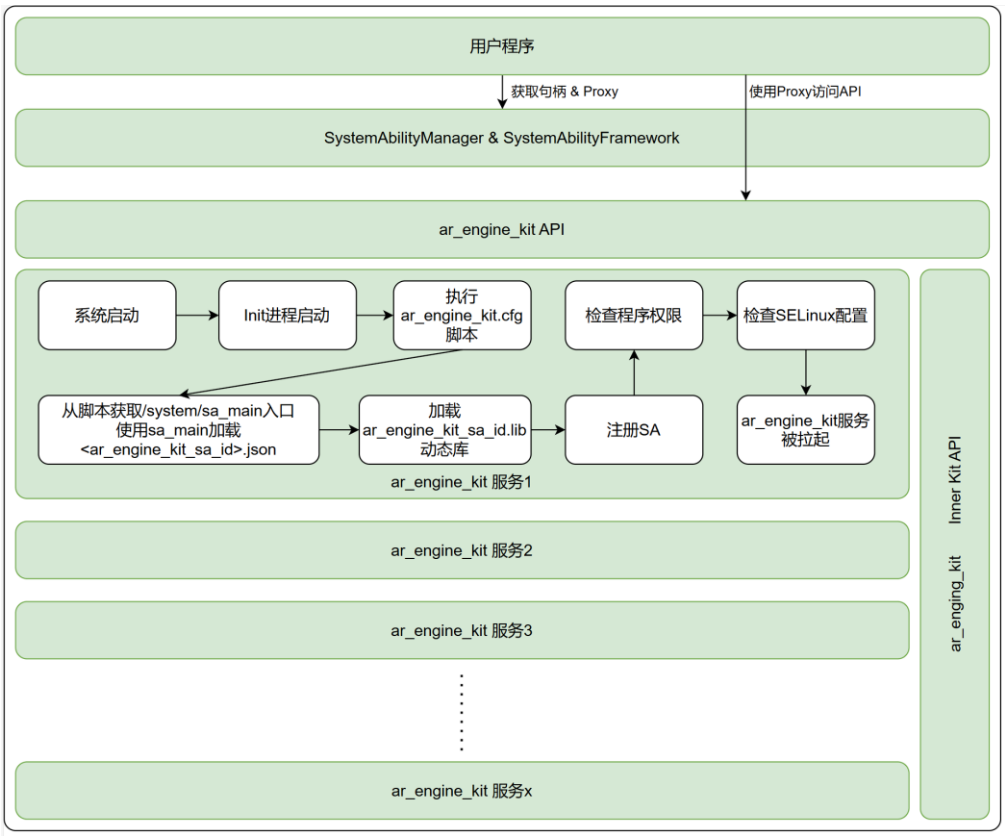

图 3.3 随系统启动流程

对于按需启动来说，在系统启动阶段，只执行对应的配置文件，比如 Init 阶段的 ar_engine_kit.cfg 文件，以及之后通过 SAMGR 入口 sa_main 执行的 ar_engine_kit.json 文件。之后在应用程序通过 SAMRG 请求时，再通过 ar_engine_kit.json 中注册的信息拉起 lib 库，之后检查程序权限、SELinux 配置，通过后标志 SA 已经被拉起。

对于随系统启动来说，在系统启动阶段，即 Init 阶段执行 ar_enging_kit.cfg 文件，以及之后在 sa_main 中通过 json 直接拉起 lib 库，之后检查程序权限、SELinux 配置，通过后标志 SA 已经被拉起。

这两种方式中，多个 SA 服务都通过 inner_kit 暴露接口，使用 IPC 进行进程间通信。对外则使用 ar_engine_kit 提供统一接口。

3.2.2 执行整体流程

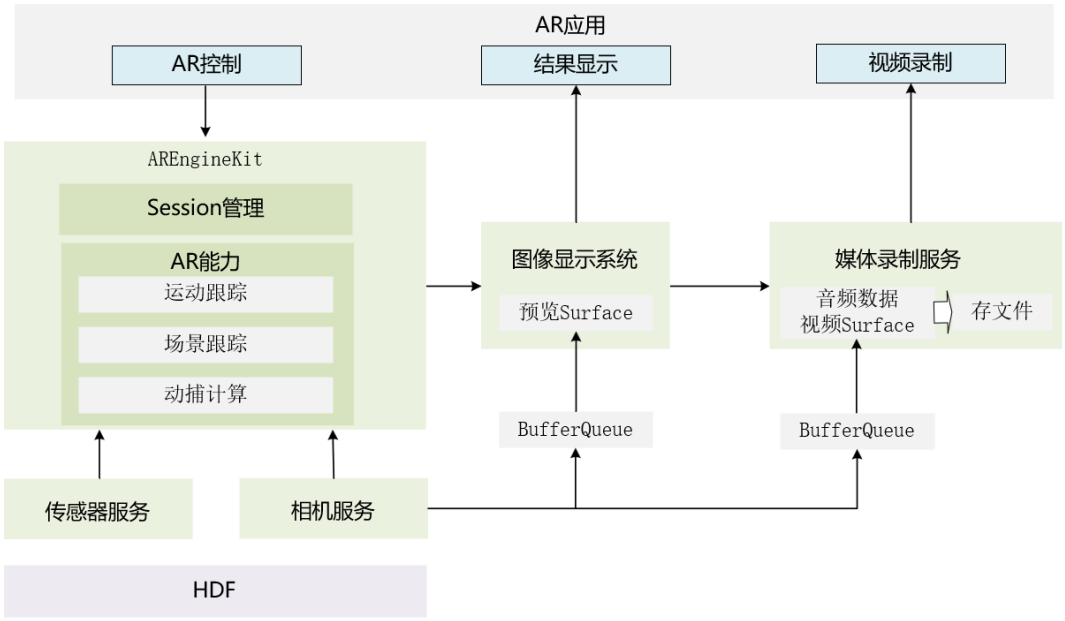

图 3.4 执行流程

AREngineKit 启动后，应用可以通过 AREngineKit 实现 AR 功能。数据流从底层硬件（相机和传感器）到 AREngine Kit 核心模块，再到图像显示系统，控制流由应用层通过 Session 管理调用 AR 能力。执行流程为：

- Session 管理模块启动会话并调度 AR 能力。
- AR 能力模块（运动跟踪、场景跟踪、动态捕算）完成核心功能处理。
- 图像显示系统负责渲染与实时预览。

- 媒体录制服务记录增强现实内容（可选）。
- AR 应用层负责用户交互与结果展示。

3.2.3 AREngineKit 执行阶段

在不同设备上，AREngineKit 生命周期不同。手机上 AREngineKit 手动开启然后运行在后台，随着所有 AR 应用的关闭而销毁进程。眼镜等 XR 设备 AREngineKit 随系统启动后一直运行，随系统关闭而销毁。

应用使用 AREngineKit 时，分为不同阶段，每个阶段程序执行不同操作，具体如下：

1）初始化阶段

在应用启动或需要使用 AR 功能时，初始化操作包括以下步骤：

- 应用层调用框架层接口：

应用层通过调用初始化接口（如 HMS_AREngineKit_ARConfig_Create）启动初始化流程。

- 框架层向系统服务层发送初始化请求：

框架层通过 IPC 机制向系统服务层（Session 管理模块）发送初始化指令。

- 系统服务层（SA）初始化 AREngine：

系统服务层加载场景渲染、锚点注册和场景理解模块，配置跟踪模式、环境检测模式（如平面检测、光照估计），加载相机、传感器配置。

数据传输：

从应用层到框架层：AREngine 配置数据（如 ARConfig）。

从框架层到系统服务层：传递会话的具体配置（如跟踪模式、渲染参数、锚点检测能力）。

2）启动阶段

在初始化完成后，进入 AR 会话启动阶段：

- 应用层调用启动接口：

应用层调用框架层的启动接口（如 HMS_AREngineKit_ARConfig_Start），请求开始 AR 会话。

- 框架层发送启动请求：

框架层通过 IPC 向系统服务层发送启动会话请求。

- 系统服务层创建并启动 AR 会话：

启动设备位姿跟踪、点云生成和其他子模块。

配置场景渲染、锚点管理等任务。

数据传输：

从应用层到框架层：启动指令。

从框架层到服务层：会话配置（如跟踪模式、场景理解参数）。

3）运行阶段

AR 会话的运行阶段是系统的核心，包含以下功能：

- 帧数据采集：

系统服务层：接收并缓存帧数据，确保处理实时性。

- 帧数据处理：

系统服务层（SA）：运行运动跟踪算法，计算设备的位姿（位置和姿态），运行环境跟踪算法（如平面检测、点云生成），处理锚点管理和命中检测，将处理后的数据（如位姿、点云、平面信息）传递给框架层。

- 数据分发：

框架层：将服务层处理结果（如场景帧、命中检测结果）转发至应用层。

应用层：使用处理结果进行逻辑控制（如对象放置、交互），并完成场景渲染。

数据传输：

从服务层到框架层：设备位姿、环境特征（平面、点云）、实时生成的锚点数据。

从框架层到应用层：处理结果（如场景帧、命中检测结果）。

4）停止阶段

当应用退出或需要暂停 AR 会话时：

- 应用层调用停止接口：应用层调用框架层的停止接口（如 HMS_AREngineKit_ARConfig_Stop）。
- 框架层发送停止请求：通过 IPC 向系统服务层发送停止会话请求。
- 系统服务层停止会话：停止 AREngine 的所有子模块（如场景渲染、锚点管理、环境理解），关闭数据采集流（如相机和传感器）。

数据传输：

从应用层到框架层：停止指令。

从框架层到服务层：停止会话请求。

5）暂停阶段

当应用进入后台时，触发以下流程：

- 应用层调用暂停接口：调用框架层的接口（如 HMS_AREngineKit_ARConfig_Pause），请求暂停 AR 会话。
- 框架层请求系统服务层暂停会话：通过 IPC 向系统服务层发送暂停请求。
- 系统服务层处理暂停任务：停止相机流和传感器数据的采集，暂停实时处理任务（如位姿跟踪、环境理解）。

数据传输过程：

从应用层到框架层：暂停指令。

从框架层到服务层：暂停会话请求。

6）销毁阶段

在应用彻底关闭或系统不再需要 AREngine 时：

- 应用层调用销毁接口：应用层调用框架层的销毁接口（如 HMS_AREngineKit_ARConfig_Destroy）。
- 框架层发送销毁请求：通过 IPC 请求系统服务层销毁会话。
- 系统服务层销毁会话：释放会话对象、数据缓存。停止并卸载所有子模块（如运动跟踪、锚点管理）。

数据传输：

从应用层到框架层：销毁指令。

从框架层到服务层：销毁会话请求。

3.3 目录结构说明

AR 子系统目录结构如下：

```
/foundation/multimedia/ar_engine                # AR 子系统业务代码
├── frameworks                                  # 框架代码
│   ├── native                                  # 内部接口实现
│   │   ├── ar_session                          # 会话管理实现
│   │   ├── data_manager                        # 数据输出实现（渲染、锚点）
│   │   └── utils                               # 工具类实现（数据处理、数学计算等）
│   └── js                                      # 外部接口实现
│       └── ar_napi                             # AR NAPI（JS 接口）实现
├── interfaces                                  # 接口代码
│   ├── inner_api                               # 内部接口
│   │   └── utils.h                             # 工具类接口
│   └── kits                                    # 外部接口
├── LICENSE                                     # 许可证文件
├── ohos.build                                  # 构建文件
├── sa_profile                                  # 服务配置文件
│   └── ar_service_profile.json                 # 配置 AR 服务的元数据，比如权限等
└── services                                    # 服务代码
    ├── ar_service                              # AR 服务实现
    └── etc                                     # 服务配置
        ├── ar_service_config.json              # 服务配置文件
        └── permissions.json                    # 服务权限配置文件
```

1） /frameworks

职责 ：实现 AR 框架的核心功能模块。

子目录 ：

native ：C++ 实现的内部接口，用于为下层服务提供核心功能支持。

ar_session ：负责管理会话的生命周期和参数配置。

data_manager ：负责管理数据（如图片或传感器数据的缓存等）。

utils ：工具类模块，提供数学计算、坐标转换、空间标定等功能。

js ：实现对外提供的 JS 接口，主要面向开发者使用的 API，通过 NAPI 实现与 AR 框架的交互。

2） /interfaces

职责 ：定义内部和外部接口，分为面向系统服务层的内部 API 和面向应用层的外部 API。

子目录 ：

inner_api ：提供内部使用的接口，供底层模块调用。

kits ：提供给开发者调用的外部接口，封装了框架层的核心功能。对于不同设备（如手机和眼镜）

3） /services

职责 ：实现 AR 服务的核心逻辑，负责管理会话、处理传感器数据、场景理解等功能。

子目录 ：

ar_service ：管理 ar_engine_kit 的生命周期，包括启动、暂停、恢复和销毁，管理参数配置，支持开发者调整 AR 会话的核心参数。

etc ：

用于存放服务的配置文件，例如权限配置、设备适配的元数据等。

4） /sa_profile

职责 ：定义服务属性和元数据，包括服务描述、权限声明等。

文件 ： ar_service_profile.json 包含服务的基本配置（如名称、权限、依赖关系）。

5） 根目录

LICENSE ：项目的许可证文件，说明使用和分发方式。

ohos.build ：构建系统的配置文件，定义模块的编译规则等。

3.4 模块间主要接口定义

AREngineKit 的关键接口设计（Kit API）,兼容 AREngine：

1. Session 管理接口

HMS_AREngine_ARSession_Create：创建 AR 会话，初始化 Session 管理模块。

HMS_AREngine_ARSession_Update：实时更新 Session 中的计算结果。

HMS_AREngine_ARSession_SetDisplayGeometry：配置显示几何，用于适配不同分辨率的屏幕，避免显示问题。

HMS_AREngine_ARSession_SetCameraGLTexture：设置 OpenGL 纹理，用于存储相机预览流的数据。

2. AR 能力接口

HMS_AREngine_ARFrame_Create：生成一个新的 ARFrame 对象，用于每帧的虚拟内容更新和渲染。

HMS_AREngine_ARSession_GetAllTrackables：获取当前所有可追踪的物体信息，支持锚点、平面等类型。

HMS_AREngine_ARFrame_HitTest：执行命中检测，根据屏幕上的兴趣点，确定虚拟物体的位置。

HMS_AREngine_ARHitResult_AcquireNewAnchor：在命中点创建锚点，用于绑定虚拟内容。

HMS_AREngine_ARTrackableList_AcquireItem：在可追踪对象列表中获取指定的对象。

HMS_AREngine_ARPlane_GetCenterPose：获取平面的中心位置和姿态。

3. 相机管理接口

HMS_AREngine_ARFrame_AcquireCamera：获取当前帧的相机参数，用于同步虚拟和现实。

HMS_AREngine_ARCamera_GetPose：获取当前相机的位姿信息，用于定位。

4. 命中检测接口

HMS_AREngine_ARHitResultList_Create：创建命中检测结果列表。

HMS_AREngine_ARHitResultList_GetSize：获取命中检测结果列表的大小。

HMS_AREngine_ARHitResultList_GetItem：根据索引获取命中检测结果对象。

HMS_AREngine_ARHitResult_AcquireTrackable：获取命中检测中被命中的可追踪对象。

5. 位姿与坐标转换接口

HMS_AREngine_ARPose_Create：创建并初始化一个位姿对象。

HMS_AREngine_ARCamera_GetPose：在 AR 世界空间中获取相机的当前位姿。

手机和 XR 设备（AR 眼镜）设计的差异

1） 硬件适配

手机：屏幕尺寸和分辨率多样，需要灵活适配 HMS_AREngine_ARSession_SetDisplayGeometry 接口。

通常单摄像头或多摄像头组合，用于深度和 RGB 数据采集。

交互方式依赖触摸屏，重点使用 HMS_AREngine_ARFrame_HitTest 处理屏幕点击事件。

眼镜：显示设备多为微型屏幕（如 OLED 显示），对低延迟要求更高。

配备双目摄像头和 IMU 传感器，增加了对深度感知和手势捕捉的支持。

交互以手势或语音为主，需扩展 XR 交互功能模块。

2） 会话管理

手机：会话管理较为通用，需处理各种分辨率和应用场景的适配。

眼镜：需优化会话启动速度和资源占用，适应实时性要求。

3） 命中检测

手机：屏幕点击为主，使用 HMS_AREngine_ARFrame_HitTest 进行屏幕坐标转换。

眼镜：使用空间手势或目光焦点（gaze）交互，需结合 HMS_AREngine_ARCamera_GetPose 和命中检测功能。

4） 显示优化

手机：注重屏幕渲染效果，支持高分辨率和高帧率显示。

眼镜：专注于双目立体显示，需优化 HMS_AREngine_ARSession_SetDisplayGeometry 以支持不同光学设备。

5） 扩展功能

手机：适配更多应用场景（如 AR 导航、AR 购物），提供标准接口。

眼镜：支持 XR 功能扩展，如虚拟全息投影和多任务处理。

设计中的接口以 **Session 管理**、**AR 能力**和**命中检测**为核心，兼容手机与眼镜设备的差异化需求。眼镜需要更多对实时性、空间交互的优化，而手机更注重通用性和灵活适配能力。

通过上述接口设计，可以明确各模块的职责和数据交互方式。模块间通过接口解耦，具有良好的扩展性和维护性，可以适应复杂的 AR 应用场景需求。

4. 附录

[1] https://developer.apple.com/visionos/ Apple frameworks — extended for spatial computing 部分

[2] https://developer.apple.com/documentation/visionos/bringing-your-arkit-app-to-visionos Replace your ARKit code

[3] https://developer.apple.com/documentation/visionos/building-spatial-experiences-for-business-apps-with-enterprise-apis

[4] https://developer.android.com/reference/androidx/xr/runtime/package-summary

[5] https://developer.android.com/develop/xr/jetpack-xr-sdk/check-spatial-capabilities?hl=zh-cn

[6] https://developer.android.com/develop/xr/jetpack-xr-sdk/develop-ui-views?hl=zh-cn