第5章

操作系统软件框架

当前常常在操作系统内核上加一层软件框架，从而形成功能更加丰富并且便于开发的整套操作系统软件框架平台，如华为公司的鸿蒙系统、Google 公司的 Android 系统和机器人操作系统(ROS)都是这种结构，本章主要对这 3 种系统展开介绍。

5.1 Android 系统

Android 系统由 Google 公司开发，从 2008 年发布 Android 1.0 系统，直到 2021 年 Android 12 系统的发布，已为手机、电视、平板计算机、手表等数十亿设备提供平台支持，使得 Android 成为全球主要的移动操作系统。

在嵌入式 Linux 操作系统基础上添加了 Android 软件框架，俗称 Android 操作系统，也是当前比较主流的一款操作系统，主要解决了前几年至今人机交互的应用需求，从而主要用于手机以及平板计算机等人机交互的应用场景。由于需要在这些设备上运行越来越多的智能算法，因此很多公司也提供了 Android 系统的软件库版本。虽然 Android 系统建立于 Linux 系统之上，但结构相对 Linux 系统更加复杂。由于要考虑手机或平板计算机厂商的商业问题，在已有需要开源的 Linux 驱动程序层次上提出了 HAL 层，在此层次上厂商可以对各自设备特有的驱动程序进行保护，所以 Android 系统比 Linux 系统更加复杂，同时开源性也比不上 Linux。Android 系统的设计目标是方便各个手机厂商适配使用此系统，因此整体采用了更加商用友好的 Apache Software License。与 GPL(GNU Public License)相比，Apache Software License 不要求使用并修改源码的使用者重新开放源码，而只是需要在每个修改的文件中保留 License 并说明所修改的内容。

Android 系统架构主要应用于 ARM 平台，但不仅限于 ARM，通过编译控制，在 x86 等体系结构的机器上同样可以运行。Android 系统主要架构如图 5.1 所示。

Android 系统主要架构包括如下组件。

1. 硬件抽象层

硬件抽象层(Hardware Abstraction Layer，HAL)提供标准界面，向更高级

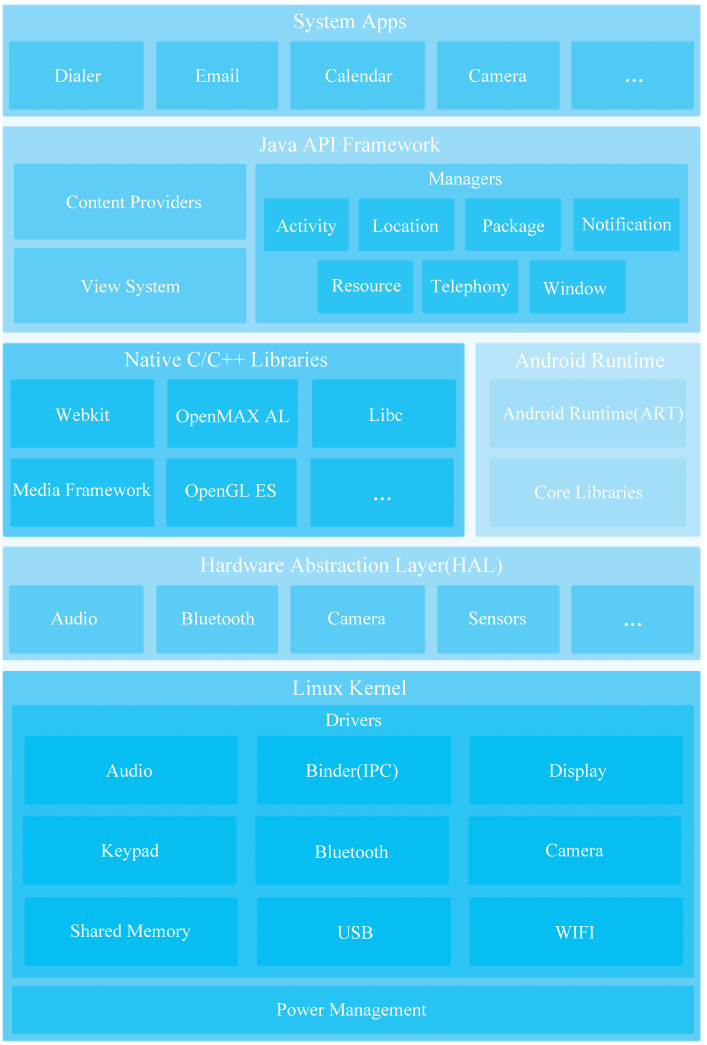

图 5.1　Android 系统主要架构

别的 Java API 框架显示设备硬件功能。HAL 包含多个库模块，其中每个模块都为特定类型的硬件组件实现一个界面，例如相机或蓝牙模块。当框架 API 要求访问设备硬件时，Android 系统将为该硬件组件加载库模块。

2. Android 运行环境

由于 Android 应用的主要开发语言是 Java 语言，所生成的目标文件为中间字节代码，因此需要一个运行时环境将应用从字节码转化为可执行的机器码。其中 Android 版本 5.0(API 级别 21)之前，采用的是 Dalvik 虚拟机的形式，通过解释执行与 JIT(Just-in-Time)编译的方式运行，因此带来性能与功耗的损失。而 Android 5.0(API 级别 21)及以后的版本引入了 Android 运行环境(Android Runtime，ART)和预先编译机制，将 Java 代码预编译为二进制可执行代码，从而避免运行时的编译开销。

3. 原生 C/C++ 库

许多核心 Android 系统组件和服务(例如 ART 和 HAL)构建自原生代码,需要以 C 和 C++ 编写的原生库(Native C/C++ Libraries)。Android 平台提供 Java 框架 API 以向应用显示其中部分原生库的功能。例如,可以通过 Android 框架的 Java OpenGL API 访问 OpenGL ES,以支持在应用中绘制、操作 2D 和 3D 图形。如果开发的是需要 C 或 C++ 代码的应用,可以使用 Android NDK 直接从原生代码访问某些原生平台库。

Android NDK 是能将 C 或 C++(原生代码)嵌入到 Android 应用中的工具,其中,原生共享库是 NDK 从源代码编译构建出动态库(.so 文件);原生静态库是 NDK 编译构建出静态库(.a 文件);Java 原生接口 (JNI)是 Java 和 C++ 组件用于相互通信的接口;应用二进制接口 (ABI),可以非常精确地定义应用的机器代码在运行时应该如何与系统交互,NDK 根据这些定义构建 .so 文件,不同的 ABI 对应不同的架构,NDK 为 32 位 ARM、AArch64、x86 及 x86-64 提供 ABI 支持。

4. Java API 框架

可通过以 Java 语言编写的 API 使用 Android OS 的整个功能集。这些 API 形成创建 Android 应用所需的构建块,它们可简化核心模块化系统组件和服务的重复使用,包括以下组件和服务。

(1) 丰富、可扩展的视图系统,可用以构建应用的 UI,包括列表、网格、文本框、按钮甚至可嵌入的网络浏览器。视图布局中的所有元素均使用 View 和 ViewGroup 对象的层次结构进行构建,View 通常用于绘制用户可看到并与之交互的内容,ViewGroup 则是不可见的容器,用于定义 View 和其他 ViewGroup 对象的布局结构,如图 5.2 所示。

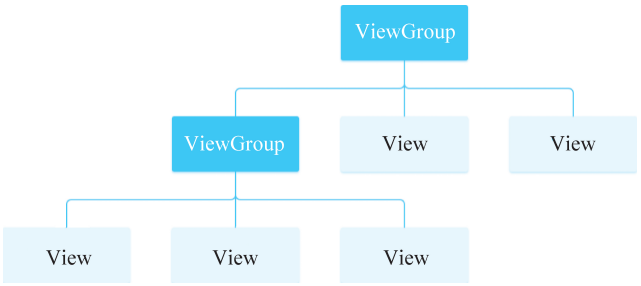

图 5.2 界面布局

Android 系统通过在 XML 中声明界面,从而可以将应用外观代码与控制其行为的代码分开。

(2) 资源管理器,用于访问非代码资源,如本地化的字符串、图形和布局文件。例如,各类资源放入项目 res/目录的特定子目录中,如图 5.3 所示。

此示例中,res/目录包含以下资源(在子目录中):一个图像资源、两个布局资源、启动器图标的 mipmap/目录以及一个字符串资源文件。

(3) 通知管理器,可让所有应用在状态栏中显示自定义提醒。通知是指 Android 在

应用的界面之外显示的消息，旨在向用户提供提醒、来自他人的通信信息或应用中的其他实时信息。用户可以点按通知来打开应用，或直接从通知中执行操作。

```
res/
    drawable/
        graphic.png
    layout/
        main.xml
        info.xml
    mipmap/
        icon.png
    values/
        strings.xml
```

图 5.3　资源管理

(4) Activity 管理器，用于管理应用的生命周期，提供常见的导航返回栈。在 Android 平台上，Activity 充当用户与应用互动的入口点，同时也决定了用户在应用内部或应用之间导航的方式，前者如“返回”按钮，后者如“最近使用的应用”按钮。

Activity 类是 Android 应用的关键组件，而 Activity 的启动和组合方式则是该平台应用模型的基本组成部分。在编程范式中，应用是通过 main() 方法启动的，而 Android 系统与此不同，它会调用与其生命周期特定阶段相对应的特定回调方法来启动 Activity 实例中的代码。

Activity 提供窗口供应用在其中绘制界面。此窗口通常会填满屏幕，但也可能比屏幕小，并浮动在其他窗口上面。大多数应用包含多个屏幕，这意味着它们包含多个 Activity。通常，应用中的一个 Activity 会被指定为主 Activity，这是用户启动应用时出现的第一个屏幕。然后，每个 Activity 可以启动另一个 Activity，以执行不同的操作。在应用中使用 Activity，需要在应用的清单中注册关于 Activity 的信息，并且必须适当地管理 Activity 的生命周期。

5. Linux 内核

Android 平台的基础是 Linux 内核(Linux Kernel)。例如，Android Runtime (ART) 依靠 Linux 内核来执行底层功能，如线程和低层内存管理。开发设备驱动程序与开发典型的 Linux 设备驱动程序类似。Android 使用的 Linux 内核版本包含一些特殊的补充功能，例如低内存终止守护进程(一个内存管理系统，可更主动地保留内存)、唤醒锁定(一种 PowerManager 系统服务)、Binder IPC 驱动程序，以及对移动嵌入式平台来说非常重要的其他功能。这些补充功能主要用于增强系统功能，不会影响驱动程序开发。

6. Binder IPC

Binder 进程间通信(IPC)机制允许应用框架跨越进程边界并调用 Android 系统服务代码，这使得高级框架 API 能与 Android 系统服务进行交互。在应用框架级别，开发者无法看到此类通信的过程，但一切似乎都在“按部就班地运行”。

Android 的内核基于 Linux 内核，但在进程间通信并没有直接利用 Linux 内核间的通信，而是采用了 Binder 方式。只需要复制一次 Binder 数据，而管道、消息队列、Socket 都需要复制两次，但共享内存方式一次内存复制都不需要；从性能角度看，Binder 性能仅次于共享内存。而从稳定性上来说，Binder 是基于 C/S 架构的，架构清晰明朗优越于共享内存。Binder 符合面向对象的思想，将进程间通信转化为通过对某个 Binder 对象的引用调用该对象的方法，而其独特之处在于 Binder 对象是一个可以跨进程引用的对象，它的实体位于一个进程中，而它的引用却遍布于系统的各个进程之中，Android 是基于 Java 语言(面向对象的语句)的特性。

5.2 鸿蒙系统

华为鸿蒙系统是华为公司2019年8月9日在华为开发者大会上正式发布的操作系统。华为鸿蒙系统是一款全新的面向全场景的分布式操作系统，创造一个超级虚拟终端互联的世界，将人、设备、场景有机地联系在一起，将消费者在全场景生活中接触的多种智能终端实现极速发现、极速连接、硬件互助、资源共享，用合适的设备提供场景体验。

5.2.1 鸿蒙系统架构介绍

HarmonyOS具备分布式软总线、分布式数据管理和分布式安全三大核心能力。其中，分布式软总线让多设备融合为一个设备，带来设备内和设备间高吞吐、低时延、高可靠的流畅连接体验。分布式数据管理让跨设备数据访问如同访问本地，大大提升跨设备数据远程读写和检索性能等。分布式安全能够把手机的内核级安全能力扩展到其他终端，进而提升全场景设备的安全性，通过设备能力互助，共同抵御攻击，保障智能家居网络安全；HarmonyOS通过定义数据和设备的安全级别，对数据和设备都进行分类分级保护，确保数据流通安全可信。

HarmonyOS是华为公司基于开源项目OpenHarmony开发的面向多种全场景智能设备的商用版本。华为已于2020年、2021年分两次把鸿蒙操作系统的基础能力全部捐献给开放原子开源基金会。OpenHarmony是由开放原子开源基金会孵化及运营的开源项目，由基金会的OpenHarmony项目群工作委员会负责运作，遵循Apache 2.0等开源协议，目标是面向全场景、全连接、全智能时代，基于开源的方式，搭建一个智能终端设备操作系统的框架和平台。

OpenHarmony整体遵从分层设计，从下向上依次为内核层、系统服务层、框架层和应用层。整个硬件架构如图5.4所示。

图5.4 OpenHarmony架构

1. 内核层

内核层主要包括内核子系统和驱动子系统。

1）内核子系统

采用多内核设计，包括 Linux、LiteOS-M 和 LiteOS-A 3 种内核，支持在 128KB 到 xGB RAM 资源的设备上运行系统组件。OpenHarmony 针对不同量级的系统，分别使用了不同形态的内核，分别为 LiteOS 和 Linux。在轻量系统、小型系统上可以选用 LiteOS；在小型系统和标准系统上可以选用 Linux。

（1）LiteOS。

OpenHarmony LiteOS 内核主要包括进程和线程调度、内存管理、IPC 机制、timer 管理等内核基本功能。

OpenHarmony LiteOS 内核的源代码分为 kernel_liteos_a 和 kernel_liteos_m 这两个代码仓库。其中，kernel_liteos_a 主要针对小型系统（Small System）和标准系统（Standard System），而 kernel_liteos_m 则主要针对轻量系统（Mini System）。

LiteOS-M 用于轻量系统，面向 MCU 类处理器，例如 ARM Cortex-M、RISC-V 32 位的设备，硬件资源极其有限，支持的设备最小内存为 128KB，可以提供多种轻量级网络协议，轻量级的图形框架，以及丰富的 IoT 总线读写部件等。可支撑的产品如智能家居领域的连接类模组、传感器设备、穿戴类设备等。

LiteOS-A 用于小型系统，面向应用处理器，例如 ARM Cortex-A 的设备，支持的设备最小内存为 1MB，可以提供更高的安全能力、标准的图形框架、视频编解码的多媒体能力。可支撑的产品如智能家居领域的 IP Camera、电子猫眼、路由器以及智慧出行领域的行车记录仪等。

（2）Linux。

Linux 用于标准系统，面向应用处理器，例如 ARM Cortex-A 的设备，支持的设备最小内存为 128MB，可以提供增强的交互能力、3D GPU 以及硬件合成能力、更多控件以及动效更丰富的图形能力、完整的应用框架。可支撑的产品如智能手机等。OpenHarmony 中 Linux 内核从 LTS 版本中选择合适的版本作为内核的基础版本，目前已完成对 Linux-4.19 及 Linux-5.10 的适配及支持。内核的 Patch 组成模块，在编译构建流程中，针对具体芯片平台，合入对应的架构驱动代码，进行编译对应的内核镜像。所有补丁来源均遵守 GPL-2.0 协议。

2）驱动子系统

驱动框架（Hardware Driver Foundation，HDF）为驱动开发者提供驱动框架能力，包括驱动加载、驱动服务管理和驱动消息机制。HDF 类似于 Android 系统中的 HAL 层，支持以下关键特性。

（1）弹性化的框架能力。在传统的驱动框架能力的基础上，OpenHarmony 驱动子系统通过构建弹性化的框架能力，可支持千字节级别到兆字节级容量的终端产品形态部署。

（2）规范化的驱动接口。定义了常见驱动接口，为驱动开发者和使用者提供丰富、稳定接口，并和未来开放的面向手机、平板、智慧屏等设备驱动接口保持 API 兼容性。

(3) 组件化的驱动模型。支持组件化的驱动模型,为开发者提供更精细化的驱动管理,开发者可以对驱动进行组件化拆分,使得驱动开发者可以更多关注驱动与硬件交互部分。同时系统也预置了部分模板化的驱动模型组件,如网络设备模型等。

(4) 归一化的配置界面。提供统一的配置界面,构建跨平台的配置转换和生成工具,实现跨平台的无缝切换。为了方便驱动开发者更易于开发 OpenHarmony 驱动程序,OpenHarmony 驱动子系统在 DevEco 集成了驱动开发套件工具,支持驱动工程管理、驱动模板生成、配置管理等界面化的操作。

开发者基于 HDF 驱动框架开发的驱动主要包含三部分。

(1) 驱动程序部分——完成驱动的功能逻辑。

(2) 驱动配置信息——指示驱动的加载信息内容。

(3) 驱动资源配置——配置驱动的硬件配置信息。

OpenHarmony 驱动主要部署在内核态,当前主要采用静态连接方式,随内核子系统编译和系统镜像打包。

2. 系统服务层

系统服务层是 OpenHarmony 的核心能力集合,通过框架层对应用程序提供服务。该层包含以下 5 部分。

1) 分布式软总线

分布式软总线是多设备终端的统一基座,为设备间的无缝互连提供了统一的分布式通信能力,能够快速发现并连接设备,高效地传输任务和数据。

分布式软总线子系统旨在为 OpenHarmony 系统提供通信相关的能力,包括 WLAN 服务能力、蓝牙服务能力、软总线、进程间通信 RPC(Remote Procedure Call)等通信能力。

(1) WLAN 服务。为用户提供 WLAN 基础功能、P2P(Peer-to-Peer)功能和 WLAN 消息通知的相应服务,让应用可以通过 WLAN 和其他设备互联互通。

(2) 蓝牙服务。为应用提供传统蓝牙以及低功耗蓝牙相关功能和服务。

(3) 软总线。为应用和系统提供近场设备间分布式通信的能力,提供不区分通信方式的设备发现、连接、组网和传输功能。

(4) 进程间通信。提供不区分设备内或设备间的进程间通信能力。

2) 分布式数据管理

分布式数据管理子系统支持单设备的各种结构化数据的持久化,以及跨设备之间数据的同步、共享功能。开发者通过分布式数据管理子系统,能够方便地完成应用程序数据在不同终端设备间的无缝衔接,满足用户跨设备使用数据的一致性体验。

(1) 本地数据管理。提供单设备上结构化数据的存储和访问能力。使用 SQLite 作为持久化存储引擎,提供了多种类型的本地数据库,分别是关系数据库(Relational Database,RDB)和轻量级偏好数据库(Preferences),用以满足使用不同数据模型对应用数据进行持久化和访问的需求。

(2) 分布式数据服务。分布式数据服务支持数据跨设备相互同步,为用户提供在多种终端设备上一致的数据访问体验。通过结合账号、应用和数据库三元组,分布式数据服

务对数据进行隔离。在通过可信认证的设备间，分布式数据服务支持数据相互同步，提供跨设备的数据访问。

3）分布式任务调度

分布式任务调度基于分布式软总线、分布式数据管理、分布式 Profile 等技术特性，构建统一的分布式服务管理（发现、同步、注册、调用）机制，支持对跨设备的应用进行远程启动、远程调用、绑定/解绑，以及迁移等操作，能够根据不同设备的能力、位置、业务运行状态、资源使用情况并结合用户的习惯和意图，选择最合适的设备运行分布式任务。主要功能如下。

（1）远程启动元能力。跨设备拉起远端设备上的指定元能力。

（2）远程迁移元能力。将元能力跨设备迁移到远端设备。

（3）远程绑定元能力。跨设备绑定远端设备上的指定元能力。

（4）系统服务管理。提供系统服务的本地启动、注册、查询等功能；提供系统服务的跨设备查询功能。

4）公共基础库子系统

公共基础库子系统提供了一些常用的 C、C ++ 开发增强 API。C ++ 部分：文件、路径、字符串相关操作的能力增强接口；读写锁、信号量、定时器、线程增强及线程池等接口；安全数据容器、数据序列化等接口；各子系统的错误码相关定义；C 语言安全函数接口。C 部分：简易的数据存取接口 kv_store；标准文件相关操作 HAL 接口；其他一些内部功能，如定时器等。

5）方舟多语言运行时子系统

方舟多语言运行时子系统提供了 JavaScript、C/C ++ 语言程序的编译、执行环境，提供支撑运行时的基础库，以及关联的 API 接口、编译器和配套工具。当前支持的编程语言包括 JavaScript、C/C ++ 。子系统中的组件划分也是基于编程语言维度，每个组件支持单独编译，可以基于场景进行组合和分离。

3. 框架层

框架层为应用开发提供了 C/C ++ /JavaScript 等多语言的用户程序框架和 Ability 框架，适用于 JavaScript 语言的 JavaScript UI 框架，以及各种软硬件服务对外开放的多语言框架 API。根据系统的组件化裁减程度，设备支持的 API 也会有所不同。

1）UI 框架

使用基于 JavaScript 扩展的类 Web 开发范式的方舟开发框架，包括应用层（Application）、前端框架层（Framework）、引擎层（Engine）和平台适配层（Porting Layer）。

（1）应用层表示开发者开发的 FA 应用，这里的 FA 应用特指 JavaScript FA 应用。

（2）前端框架层主要完成前端页面解析，以及提供 MVVM（Model-View-View-Model）开发模式、页面路由机制和自定义组件等能力。

（3）引擎层主要提供动画解析、DOM（Document Object Model）树构建、布局计算、渲染命令构建与绘制、事件管理等能力。

(4) 适配层主要完成对平台层进行抽象,提供抽象接口,可以对接到系统平台,比如事件对接、渲染管线对接和系统生命周期对接等。

2) 用户程序框架

用户程序框架子系统是 OpenHarmony 为开发者提供的一套开发 OpenHarmony 应用程序的框架,包含以下模块。

(1) AppKit。是用户程序框架提供给开发者的开发包,开发者基于该开发包可以开发出基于 Ability 组件的应用。

(2) AppManagerService。应用管理服务,用于管理应用运行关系、调度应用进程生命周期及状态的系统服务。

(3) BundleManagerService。是负责管理安装包的系统服务,常见的如包安装、更新、卸载和包信息查询等,运行在 Foundation 进程。

3) Ability 框架

(1) Page Ability 介绍。

Page Ability(见图 5.5)是主要负责页面交互的,那么就可以理解为 Android 的 Activity。类似于 Activity 有生命周期,同样 Page Ability 也是有生命周期的,生命周期分别是 onStart()、onActive()、onInactive()、onBackground()、onForeground()、onStop()。

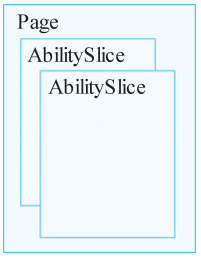

图 5.5 Page Ability 多页面示意图

Ability 是应用所具备能力的抽象,也是应用程序的重要组成部分。一个应用可以具备多种能力(即可以包含多个 Ability),HarmonyOS 支持应用以 Ability 为单位进行部署。Ability 可以分为 FA(Feature Ability)和 PA(Particle Ability)两种类型,每种类型为开发者提供了不同的模板,以便实现不同的业务功能。FA 支持 Page Ability(页面能力),用于提供与用户交互的能力。一个 Page 可以由一个或多个 AbilitySlice 构成,AbilitySlice 是指应用的单个页面及其控制逻辑的总和。一个 Page 可以包含多个 AbilitySlice,但是 Page 进入前台时界面默认只展示一个 AbilitySlice。

(2) Service Ability 介绍。

Service 也是一种 Ability,Ability 为 Service 提供了以下生命周期方法,开发者可以重写这些方法,来添加其他 Ability 请求与 Service Ability 交互时的处理方法。

如果 Service 需要与 Page Ability 或其他应用的 Service Ability 进行交互,则须创建用于连接的 Connection。Service 支持其他 Ability 通过 connectAbility()方法与其进行连接。

如果 Service 需要与 Page Ability 或其他应用的 Service Ability 进行跨设备交互,则须创建用于连接的 Connection。Service 支持其他 Ability 通过 connectAbility()方法与其进行跨设备连接。

(3) Data Ability 介绍。

通过 Ability 派生出的 DataAbility 类,有助于应用管理其自身和其他应用存储数据的访问,并提供与其他应用共享数据的方法。Data 既可用于同设备不同应用的数据共

享，也支持跨设备不同应用的数据共享。

4. 应用层

应用层包括系统应用和第三方非系统应用。应用由一个或多个 FA（Feature Ability）或 PA（Particle Ability）组成。其中，FA 有 UI 界面，提供与用户交互的能力；而 PA 无 UI 界面，提供后台运行任务的能力以及统一的数据访问抽象。基于 FA/PA 开发的应用，能够实现特定的业务功能，支持跨设备调度与分发，为用户提供一致、高效的应用体验。

根据图 5.4，OpenHarmony 框架在应用框架层和系统服务层之间提供了系统基本能力子系统集、基础软件服务子系统集、增强软件服务子系统集和硬件服务子系统集。除了框架层和系统服务层本身已有的模块外，还有如下模块。

在系统基本能力子系统集中有如下模块。

（1）多模输入子系统。OpenHarmony 旨在为开发者提供 NUI（Natural User Interface）的交互方式，有别于传统操作系统的输入，在 OpenHarmony 上，将多种维度的输入整合在一起，开发者可以借助应用程序框架、系统自带的 UI 组件或 API 接口轻松地实现具有多维、自然交互特点的应用程序。多模输入子系统目前支持传统的输入交互方式，例如按键、触控等。

（2）图形子系统。图形子系统主要包括 UI 组件、布局、动画、字体、输入事件、窗口管理、渲染绘制等模块，构建基于轻量 OS 应用框架满足硬件资源较小的物联网设备或者构建基于标准 OS 的应用框架满足富设备（如平板计算机和轻智能机等）的 OpenHarmony 系统应用开发。

（3）安全子系统。安全子系统包括系统安全、数据安全、应用安全等功能，为 OpenHarmony 提供有效保护应用和用户数据的能力。安全子系统当前开源的功能，包括应用完整性保护、应用权限管理、设备认证、密钥管理服务、数据分级保护。

（4）AI 子系统。AI 业务子系统是 OpenHarmony 提供原生的分布式 AI 能力的子系统。本次开源范围是提供了统一的 AI 引擎框架，实现算法能力快速插件化集成。框架中主要包含插件管理、模块管理和通信管理等模块，对 AI 算法能力进行生命周期管理和按需部署。后续，会逐步定义统一的 AI 能力接口，便于 AI 能力的分布式调用。同时，提供适配不同推理框架层级的统一推理接口。

在基础软件服务子系统集有如下模块。

（1）事件通知子系统。OpenHarmony 通过公共事件服务（Common Event Service，CES）为应用程序提供订阅、发布、退订公共事件的能力。

公共事件可分为系统公共事件和自定义公共事件。

系统公共事件：系统将收集到的事件信息，根据系统策略发送给订阅该事件的用户程序。例如，系统关键服务发布的系统事件（例如 hap 安装、更新、卸载等）。

自定义公共事件：应用自定义一些公共事件用来实现跨应用的事件通信能力。

每个应用都可以按需订阅公共事件，订阅成功且公共事件发布，系统会把其发送给应用。这些公共事件可能来自系统、其他应用和应用自身。