第 1 章

鸿蒙开发环境搭建

1

本章主要介绍如何在个人计算机上搭建鸿蒙（HarmonyOS）App的开发环境，包括鸿蒙系统的发展历程，搭建DevEco Studio的开发环境，创建并编译鸿蒙App项目以及运行和调试。

1.1 鸿蒙系统开发简介

本节介绍鸿蒙系统的历史与现状，包括鸿蒙系统的发展历程、鸿蒙系统的三大特性、鸿蒙系统的总体架构以及鸿蒙应用的技术理念等。

1.1.1 鸿蒙系统的发展历程

在移动互联网时代的初期，除了手机、平板电脑等通信设备，还有耳机、音箱、随身听等电子设备也接入了移动互联网。随着移动互联网的蓬勃发展，正有更多的电气设备加入这个生态，比如传统家电、可穿戴设备，甚至智能家居、电动汽车等。

现在，这种智慧生态已经司空见惯，但在移动互联网兴起的十几年前，几乎无人能预见今天万物互联的时代。2012年，在广东深圳，华为“中央研究院”的一群年轻人敏锐地捕捉到了这一充满潜力的智慧互联生态，并在当时提出了适用于万物互联的分布式操作系统的构想。2015年，分布式操作系统的内核被正式立项研发，并命名为“鸿蒙”，寓意“阴阳初分、天地鸿蒙”，旨在成为万物互联的操作系统基座。

2017年，鸿蒙内核1.0版本成功研发，它采用微内核、分布式架构，迥异于采用宏内核、单体式架构的Linux系统。2019年，具备商用能力的鸿蒙系统1.0版本正式发布。考虑到当时庞大的存量手机和平板设备基础，鸿蒙1.0到鸿蒙4.0版本都保持对开源的AOSP的兼容，即适配Android 10版本。

随着鸿蒙生态的不断壮大，构建完全独立的鸿蒙系统已具备成熟条件。2024年10月22日，不再兼容Android的鸿蒙5.0系统（HarmonyOS 5.0，也称HarmonyOS NEXT）正式发布。该系统不仅是继苹果iOS和谷歌Android之后全球第三大移动操作系统，更是我国首个面向消费市场的从底层系统到应用框架的全面自主研发的操作系统。2025年10月22日，HarmonyOS 6.0新一代分布式操作系统正式发布，其系统的流畅度、性能、安全特性、智能化程度、跨设备协同能力等方面均有显著提升。表1-1展示了鸿蒙系统各版本的发布时间及其功能简介。

表1-1　鸿蒙系统各版本的发布时间及其功能

鸿蒙系统版本	发布时间	功能简介
HarmonyOS 1.0	2019年8月9日	跨终端操作系统尝试
HarmonyOS 2.0	2021年6月2日	分布式能力全面升级，拓展至核心设备
HarmonyOS 3.0	2022年7月27日	提升至万物智联，跨终端性能提升
HarmonyOS 4.0	2023年8月4日	分布式万物互联，元服务，人工智能（AI）能力
HarmonyOS 5.0	2024年10月22日	鸿蒙NEXT，全新自研底层架构，不兼容Android系统
HarmonyOS 6.0	2025年10月22日	引入HMAF智能体框架实现人机交互智能化升级

1.1.2　鸿蒙系统的三大特性

鸿蒙是一款基于微内核的面向全场景的分布式操作系统，既可用于智能手表、智能手机等通信设备，也可用于平板电脑、笔记本电脑等生产力设备，还可用于车载大屏、智慧屏等物联网设备。与其他操作系统相比，鸿蒙系统具有分布式、微内核、全场景3大特性，下面分别进行介绍。

1. 分布式

鸿蒙系统的分布式能力包括分布式软总线、分布式设备虚拟化、分布式数据管理、分布式任务调度和分布式连接。具体说明如下。

1）分布式软总线

分布式软总线是鸿蒙分布式能力中的基础特性，它以手机为中心，将总线分为任务总线（传输指令）和数据总线（同步数据）。分布式软总线的主要特征有下列5点：

（1）优化不稳定的无线环境：相对于传统的传输协议，具有高带宽、低时延、高可靠性、开放性和标准化等特点。

（2）设备自动发现与连接：通过分布式软总线，可以实现设备间的快速自动发现与连接（前提是设备处于同一网络并登录同一华为账号）。

（3）支持多种传输协议：分布式软总线支持整合WiFi、蓝牙、USB等多种传输协议。通过手机等中转设备作为桥梁，能够打通蓝牙设备与WiFi设备之间的隔离，实现互联互通。

（4）极简API和协议：分布式软总线具有极简API和协议，不仅方便开发者，还有效提高了网络传输能力。开发者只需面对一个逻辑协议，无须感知具体的传输协议。

（5）高效通信能力：通过分布式软总线，鸿蒙系统可为处于同一网络内的设备提供高效的通信能力，由此实现万物互联。

2）分布式设备虚拟化

分布式设备虚拟化建立在分布式软总线的基础上，可整合多个鸿蒙设备的性能和资源，从而形成超级虚拟终端。例如，同一个家庭中的手机、路由器和智慧屏等鸿蒙设备可通过单一超级虚拟终端的方式共用硬件资源。

3）分布式数据管理

分布式数据管理建立在分布式软总线的基础上，支持在多个鸿蒙设备之间进行高效的数据同步和管理。

4）分布式任务调度

分布式任务调度建立在分布式软总线和分布式数据管理之上，支持在多个鸿蒙设备之间进行高效的应用流转和应用协同。具体如下：

- 应用流转：指同一个应用程序在不同设备之间的迁移和迁回。例如，用户正在使用手机进行视频通话，若不方便操作手机，可将视频通话界面转移至智慧屏上；待手机操作完成后，用户可将视频通话界面迁回至手机。
- 应用协同：指在不同的鸿蒙设备上显示同一个应用程序的不同功能组件。例如，在手机上显示新闻列表的同时，智慧屏可同步显示新闻内容，用户通过手机选择不同条目，即可实时切换智慧屏所展示的内容，实现高效、流畅的跨设备协作。

5）分布式连接

分布式连接能力实现了智能终端底层硬件与应用层之间的互联互通，支持通过USB接口共享部分硬件资源和软件能力。开发者可基于此开发多样化的生态产品，为用户带来更丰富、灵活的跨设备体验。

2. 微内核

传统的操作系统基于宏内核，宏内核指的是系统内核提供了许多系统功能，包括硬件驱动、设备调度器、地址空间管理、进程间通信、文件系统、虚拟文件系统、系统调用等。然而，在万物互联时代，许多物联网设备属于小型的智能设备，更注重低功耗、低内存和安全性等新要求，显然不适合“大而全”的宏内核，更适合“小而精”的微内核。

微内核通过简化系统功能，将大部分服务封装为模块并转移到用户态空间，使内核只留下基本的系统功能。比如，只有与处理器和输入/输出设备相关的操作代码才放在内核中，而与硬件平台无关的服务程序则放在外核的用户空间，从而灵活地适配各种硬件设备。

采用微内核的系统可以按需启动模块服务，并按需接入外设。由于用户态的进程各自独立，系统的耦合度大大降低。此外，微内核系统的各模块通过进程间通信（IPC）互相联系，因此该系统类型能够很好地支持分布式系统。

3. 全场景

鸿蒙系统的全场景分为两个层次，一个是设备全场景，另一个是应用全场景，分别对应硬件和软件。

（1）设备全场景属于硬件层次，除了传统的手机和平板电脑之外，还有搭载鸿蒙系统的穿戴设备、智慧屏、全屋智能、鸿蒙智行等新兴产品。全场景智慧化战略可概括为“1+8+N”，其中“1”代表手机；“8”代表手表、耳机、计算机、平板电脑、音箱、眼镜、电视、车机等产品；“N”代表其他N种智能硬件，包括智能家居、智联教育、智慧出行、运动健康等智能产品。

（2）应用全场景属于软件层次，除了传统的通信和娱乐服务之外，还有健康监测、家居管理等新兴服务，以及基于鸿蒙智能的各种AI服务，包括智能修图、智能教学、智能办公、智能驾驶等智能服务。

1.1.3　鸿蒙系统的总体架构

鸿蒙系统整体遵从分层设计，从下向上依次为：内核层、系统服务层、应用框架层和应用层。系统功能按照“系统→子系统→功能/模块”逐级展开。在多设备部署场景下，系统支持根据实际需求裁

剪某些非必要的子系统或功能/模块。鸿蒙系统的总体架构如图1-1所示。

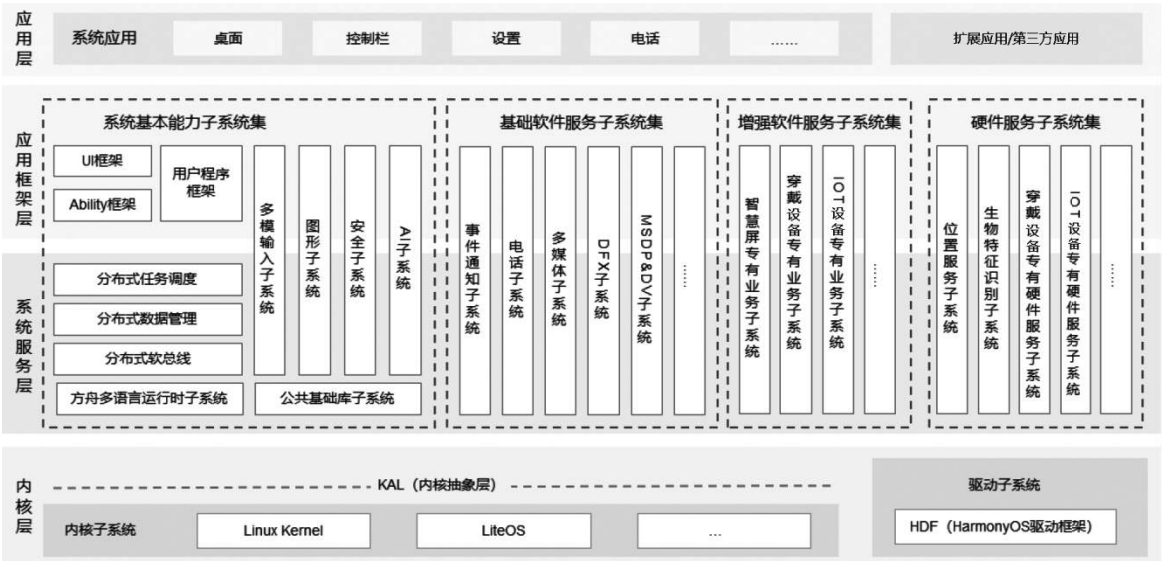

图 1-1　鸿蒙系统的总体架构

下面简要介绍鸿蒙系统总体架构的4个层次。

1. 内核层

内核层位于鸿蒙系统的底层，负责管理底层硬件资源，并提供安全稳定的系统基础功能。它包括进程管理、内存管理、设备驱动、文件系统等核心功能，以及安全和隔离机制，保证系统的可靠性和稳定性。内核层由内核子系统和驱动子系统组成，具体说明如下：

（1）内核子系统：鸿蒙系统采用多内核设计，支持根据不同资源受限设备选用适合的OS内核。内核抽象层（Kernel Abstract Layer，简称KAL）通过屏蔽多内核差异，为上层提供基础的内核能力，包括进程/线程管理、内存管理、文件系统、网络管理和外设管理等。

（2）驱动子系统：硬件驱动框架（HDF）是鸿蒙系统硬件生态开放的基础，提供统一的外设访问能力和驱动开发、管理框架。

2. 系统服务层

系统服务层是鸿蒙系统的核心能力集合，包括通信服务、图形服务、媒体服务、位置服务等。这些服务通过标准接口和协议提供给上层应用，使应用能够更方便地调用系统功能。系统服务层包含以下几个部分：

（1）系统基本能力子系统集：为分布式应用在鸿蒙系统多设备上的运行、调度、迁移等操作提供基础能力。由分布式软总线、分布式数据管理、分布式任务调度、方舟多语言运行时、公共基础库、多模输入、图形、安全和AI等子系统组成。其中，方舟多语言运行时为C/C++/JS等多语言提供运行环境及基础类库支持，也为使用方舟编译器静态化处理的Java程序（如应用程序或框架层中使用Java语言开发的部分）提供运行时支持。

（2）基础软件服务子系统集：为鸿蒙系统提供公共的、通用的软件服务，由事件通知、电话、多媒体、DFX（Design For X）、MSDP&DV等子系统组成。

（3）增强软件服务子系统集：为鸿蒙系统提供针对不同设备的、差异化的能力增强型软件服务，由智慧屏专有业务、穿戴设备专有业务、IoT设备专有业务等子系统组成。

（4）硬件服务子系统集：为鸿蒙系统提供硬件服务，由位置服务、生物特征识别、穿戴设备专有硬件服务、IoT设备专有硬件服务等子系统组成。

3. 应用框架层

应用框架层提供了一系列开发框架和工具，帮助开发者快速构建应用程序。它包括UI框架、Ability框架、数据存储框架、多语言框架等，以及一些常用的应用组件，如通知管理、权限管理、账号管理等。开发者可以通过这些框架实现应用程序的业务逻辑和界面展示。

4. 应用层

应用层是顶层的用户界面，包括系统应用和第三方非系统应用。例如，桌面、控制栏、设置、电话、扩展应用/第三方应用等。应用层通过系统服务层和应用框架层的服务，实现各种业务功能和交互操作。

1.1.4　鸿蒙应用的技术理念

鸿蒙系统结合移动生态发展的趋势，提出了3大技术理念：一次开发，多端部署；可分可合，自由流转；统一生态，鸿蒙智能。下面分别进行说明。

1. 一次开发，多端部署

“一次开发，多端部署”指的是一个项目（本书统称工程），一次开发上架，多端按需部署。目的是帮助开发者高效地开发多种终端设备上的应用。为了实现这一目的，鸿蒙系统提供了几个核心能力，包括多端开发环境、多端开发能力以及多端分发机制。

1）多端开发环境

DevEco Studio是面向全场景、多设备的一站式开发平台，支持多端双向预览、分布式调优、分布式调试、多设备模拟、低代码可视化开发等能力，帮助开发者降低成本、提升效率和质量。

2）多端开发能力

当应用需要在多个设备上运行时，开发者需要适配不同的屏幕尺寸、分辨率、交互方式（如触摸和键盘等）和硬件能力（如内存差异和器件差异等），这增加了开发成本。因此，多端开发能力的核心目标是降低多设备应用的开发成本。为了实现该目标，鸿蒙系统提供了多端UI适配、交互事件归一、设备能力抽象等核心能力，帮助开发者降低开发与维护成本，提高代码复用率。

3）多端分发机制

传统的多设备应用开发通常需要针对不同类型的设备分别进行多次开发、打包和上架，导致高昂的开发和维护成本。为解决这一问题，鸿蒙系统提供了“一次开发，多端部署”的能力，开发者只需要基于一套项目代码，即可一次打包生成多个HAP（HarmonyOS Ability Package），统一上架后，系统可根据设备类型按需进行分发。

此外，除了可以开发传统的应用，开发者还可以开发元服务（Meta-Service）。元服务是一种面向未来的服务形态，具有独立入口、免安装、轻量化等特性，可为用户提供一个或多个便捷的服务体验。鸿蒙系统为元服务提供了更多的分发入口，方便用户获取，同时也增加了元服务展示的机会。

2. 可分可合，自由流转

元服务是鸿蒙系统提供的一种全新的应用形态，具备独立入口，用户可通过单击、碰一碰、扫一扫等方式直接触发，无须显式安装，由程序框架后台静默安装后即可使用，可为用户提供便捷服务。

在传统移动生态中，开发者通常需要开发一个App版本。如果要给用户提供小程序，往往需要开发若干个独立的小程序。在鸿蒙生态中，鸿蒙系统支持元服务开发，开发者无须维护多套代码版本，而是通过业务解耦的方式将应用分解为若干元服务独立开发，并根据具体场景按需组合，构建出功能丰富的复杂应用。

1）可分可合

应用和元服务在生命周期内，同一套项目代码拥有下列几种形态：

- 开发态：开发者通过业务解耦，把不同的业务拆分为多个模块。
- 部署态：开发者可以将一个或多个模块自由组合，打包成不同的App Pack独立上架。
- 分发运行态：单个HAP作为元服务分发，满足用户单一使用场景，多个HAP组合为应用分发，满足用户更加复杂的使用场景。

2）自由流转

传统应用只能在单个设备内运行，当用户有多个设备，且要完成多个任务时，则需要在多个设备间来回切换。因此，应用在设备之间流转，不间断给用户提供服务的能力就变得非常重要。

鸿蒙系统提供了自由流转的能力，使开发者能够便捷地开发支持多设备的应用，方便用户使用这些功能。自由流转可分为跨端迁移和多端协同两种交互形态。跨端迁移用于实现时间上的串行交互；多端协同则用于实现时间上的并行交互。自由流转不仅给用户带来全新的交互体验，也为开发者搭建了一座从单设备时代通往多设备时代的桥梁。

3. 统一生态，鸿蒙智能

1）统一生态

统一生态具有愿景上的意义，旨在打造智能联接，共建智能世界。从设备角度来看，基于鸿蒙可以开发多种全场景终端设备；从应用角度来看，可以为鸿蒙开发多种应用，并在全场景设备上运行，满足智能家居、智慧办公等全场景的使用要求。

为实现这一目标，鸿蒙系统提供了SDK、IDE和开发者服务，以及“一次开发，多端部署”、应用“可分可合，自由流转”、分布式服务等开放能力，使开发者只需维护一个项目、一套代码，即可开发出覆盖多种设备的应用。同时，通过操作系统的内置能力即可实现应用间的互操作与跨设备流转，真正实现“开发即融入生态”。

2）鸿蒙智能

鸿蒙系统内置强大的AI能力，面向鸿蒙生态应用的开发，通过不同层次的AI能力开放，满足开发者在不同开发场景下的诉求，降低应用的开发门槛，以帮助开发者快速实现应用智能化。

此外，还提供了系统级的意图标准体系，通过多维系统感知、大模型等能力构建全局意图范式，实现对用户显性行为与潜在意图的理解。系统可将用户需求及时、准确地传递给生态伙伴，匹配合时宜的服务，为用户提供多模态、场景化的进阶体验。

1.2　搭建 DevEco Studio 开发环境

本节介绍在个人计算机上搭建DevEco Studio开发环境的过程和步骤。首先，我们将介绍作为开发机的计算机应当具备的基本配置，然后详细描述DevEco Studio的安装和配置过程，最后说明修改DevEco Studio的几个常用设置。

1.2.1　计算机配置要求

在使用DevEco Studio进行应用开发时，良好的开发体验离不开合适的硬件支持。目前，大多数开发者都采用笔记本电脑作为主要开发设备。为保障开发效率和工具稳定性，建议开发设备满足以下硬件配置要求：

1. Windows系统

（1）操作系统要求是64位的Windows 10或者Windows 11。
（2）可用内存至少16GB，越大越好。
（3）硬盘剩余空间至少100GB，越大越好。
（4）屏幕分辨率至少为1280×800像素。

2. Mac系统

（1）操作系统要求为macOS(X86) 11/12/13/14或macOS(ARM) 12/13/14。
（2）可用内存至少8GB，越大越好。
（3）硬盘剩余空间至少100GB，越大越好。
（4）屏幕分辨率至少为1280×800像素。

除了以上基本要求之外，还需满足下列几个条件：

（1）计算机能够连接WiFi（方便与服务端的网络通信），且具备USB接口（方便将调试App安装到手机上）。

（2）计算机网络位于国内公网，能够访问华为开发者网站（https://developer.huawei.com/）；且下载速度至少每秒1MB，越快越好。因为DevEco Studio的安装包体积较大，通常为几个吉字节（GB），若带宽不足，将导致安装时间过长。

（3）拥有一部已升级到鸿蒙5.0及更高版本的华为Mate 60/Mate 70/Mate 80/Pura 70/Pura 80/nova 12/nova 13/nova 14系列或者型号更新的手机。

1.2.2　安装 DevEco Studio

DevEco Studio是用于开发鸿蒙App及元服务的集成开发环境（Integrated Development Environment，简称IDE），助力开发者高效开发鸿蒙App及元服务。DevEco Studio的官方下载页面为https://developer.huawei.com/consumer/cn/download/。在下载列表中找到适合自己计算机的安装包（本书安装的是DevEco Studio 6.0.0 Release版本），单击链接即可下载指定版本的DevEco Studio安装包。DevEco Studio的安装步骤如下：

01 双击下载完成的 DevEco Studio 安装程序，弹出安装向导对话框，如图 1-2 所示。直接单击“下一步”按钮，进入如图 1-3 所示的安装路径对话框。

图 1-2　DevEco Studio 的安装向导对话框

图 1-3　DevEco Studio 的安装路径对话框

02 建议将 DevEco Studio 安装在除系统盘外的其他磁盘（如 E 盘），然后单击“下一步”按钮，进入如图 1-4 所示的安装选项对话框。先勾选 DevEco Studio 的桌面快捷方式，再单击“下一步”按钮，进入如图 1-5 所示的开始菜单设置对话框。

图 1-4　DevEco Studio 的安装选项对话框

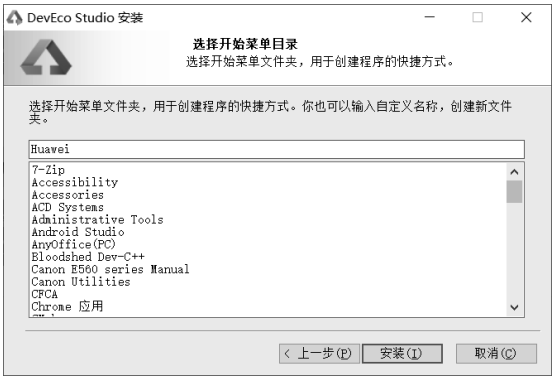

图 1-5　DevEco Studio 的开始菜单设置对话框

03 单击“安装”按钮，进入如图 1-6 所示的安装过程对话框，耐心等待安装完成。安装进度条填满后，单击“下一步”按钮进入安装完成对话框，如图 1-7 所示。

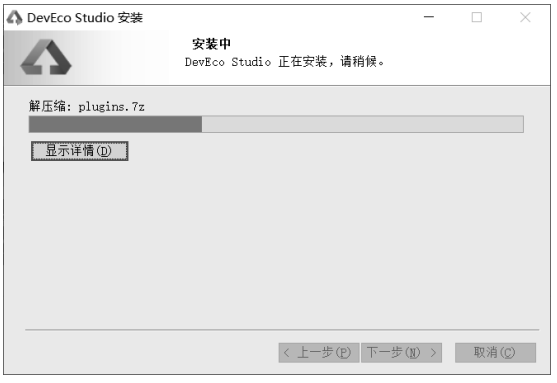

图 1-6　DevEco Studio 的安装过程对话框

图 1-7　DevEco Studio 的安装完成对话框

04 勾选安装完成对话框中的运行 DevEco Studio 选项，再单击“完成”按钮，在结束安装操作的同时启动 DevEco Studio。等待 DevEco Studio 启动后，会弹出如图 1-8 所示的配置导入对话框。先选中 Do not import settings 单选按钮，表示不导入任何配置信息，再单击 OK 按钮，进入如图 1-9 所示的许可条款对话框。

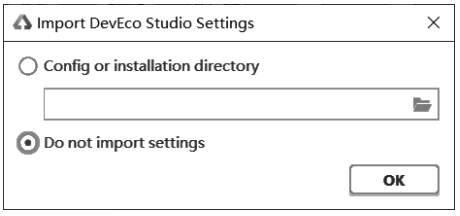

图 1-8　DevEco Studio 的配置导入对话框

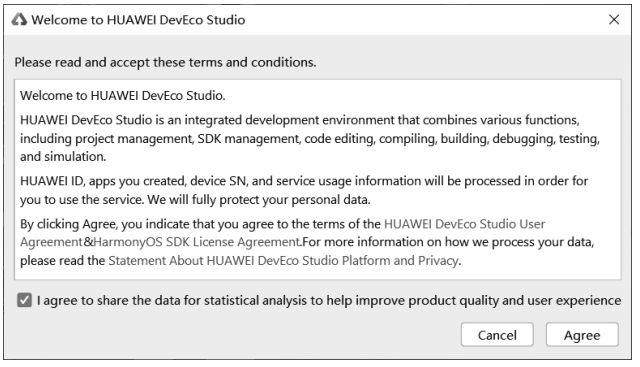

图 1-9　DevEco Studio 的许可条款对话框

05 勾选对话框下方的“I agree to share the data for ……”，表示同意相关条款，然后单击 Agree 按钮，进入 DevEco Studio 的欢迎界面，如图 1-10 所示。选择左侧列表的 Projects 项，再单击 Create Project 按钮，即可开始鸿蒙 App 的开发之旅。

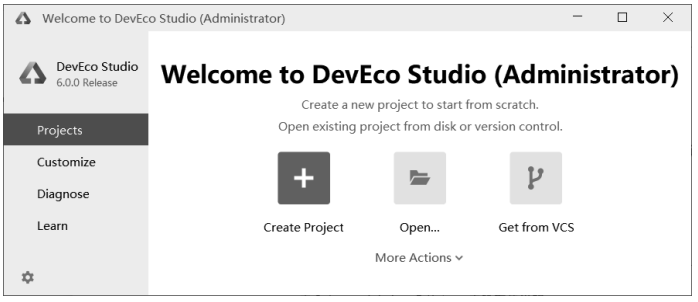

图 1-10　DevEco Studio 的欢迎界面

1.2.3　DevEco Studio 的常用设置

虽然DevEco Studio已成功启动并进入欢迎界面，但为了更贴合国内用户的使用习惯，建议在开始编码之前先修改DevEco Studio的几个常用设置，以便后续的App编码工作更加顺利。

DevEco Studio的设置修改有两种方式：一种是在启动页面左侧列表中单击Customize项（见图1-11），再单击下方的All settings...链接，打开设置页面；另一种是在DevEco Studio的主界面中，依次选择顶部菜单的“File→Settings”菜单选项，也可打开设置页面。常见的设置变更有以下3项。

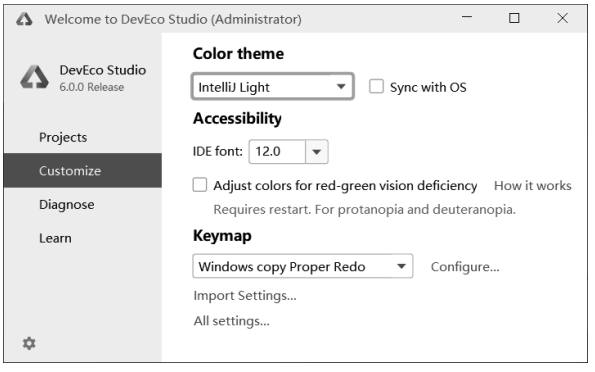

图 1-11　DevEco Studio 欢迎界面的个性化定制

1. 变更DevEco Studio的外观主题

在设置页面中依次选择左侧列表中的“Appearance & Behavior→Appearance”选项，打开外观风格页面，找到页面上方的Theme主题下拉列表，把默认的Darcula（暗黑主题）改为IntelliJ Light（亮白主题），如图1-12所示。设置完成后单击Apply按钮，即可变更DevEco Studio的外观主题。

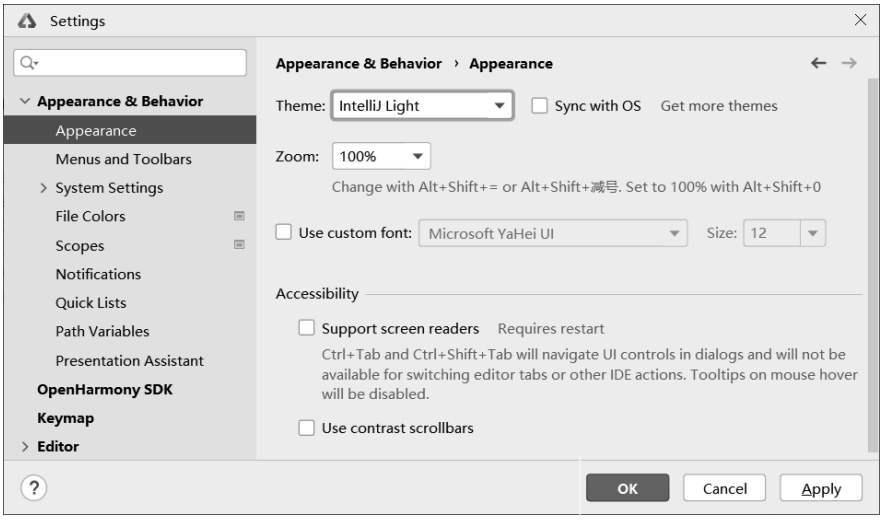

图 1-12　变更 DevEco Studio 的外观主题

2. 变更DevEco Studio的代码字体

在设置页面中依次选择左侧列表中的“Editor→Font”选项，打开编辑器的字体页面。在页面的Font下拉列表中选择代码字体，比如SimSun表示宋体，并在Size编辑框中输入字号大小，如图1-13所示。设置完成后单击Apply按钮，即可变更DevEco Studio的代码字体。

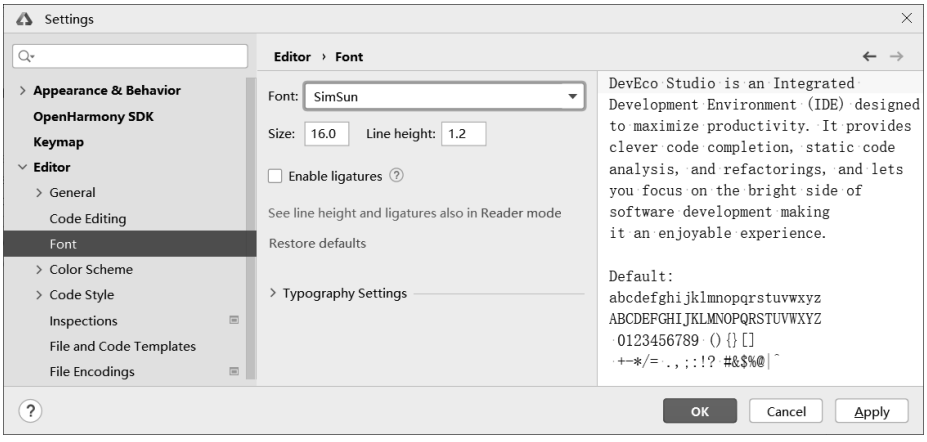

图 1-13　变更 DevEco Studio 的代码字体

3. 变更DevEco Studio的菜单语言

在设置页面中依次选择左侧列表中的“Plugins→Installed”选项，打开已安装的插件列表页面。在插件列表上方的搜索框中输入Chinese，找到Chinese(Simplified)插件，接着勾选该插件或单击插件下方的Enable按钮，如图1-14所示。设置完成后单击OK按钮，重新启动DevEco Studio即可显示中文菜单。

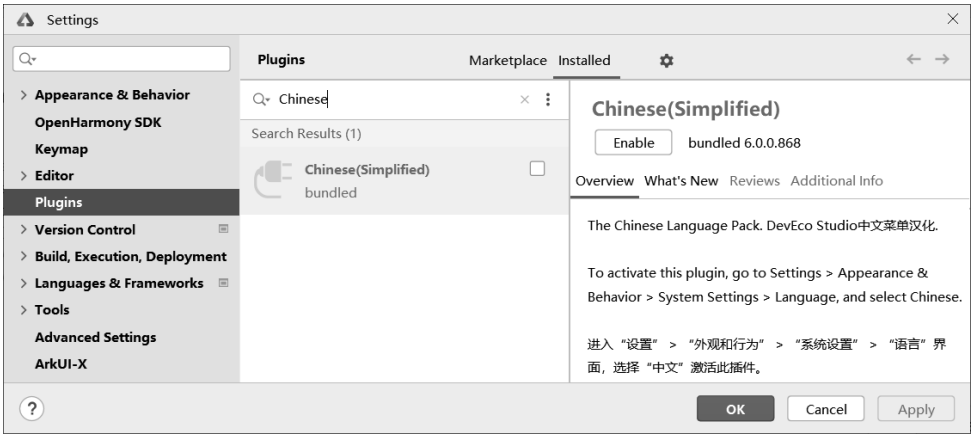

图 1-14　变更 DevEco Studio 的菜单语言

1.3　创建并编译鸿蒙 App 项目

本节介绍了使用DevEco Studio创建并编译鸿蒙App项目的过程和步骤：首先通过DevEco Studio创建一个新的App项目，然后介绍如何导入已有的App项目，最后阐述如何手工编译App项目。

1.3.1　创建鸿蒙 App 新项目

创建鸿蒙App新项目的具体步骤如下：

01 在 1.2.2 节的图 1-10 中，单击 Create Project 按钮会创建初始的新项目。如果要创建其他新项目，也可以在打开 DevEco Studio 之后，在菜单中依次选择“File→New→Create Project”选项。以上两种创建方式都会弹出如图 1-15 所示的项目创建对话框，在该对话框中，保持默认的 Empty Ability 选项，单击 Next 按钮，跳转到项目配置对话框，如图 1-16 所示。

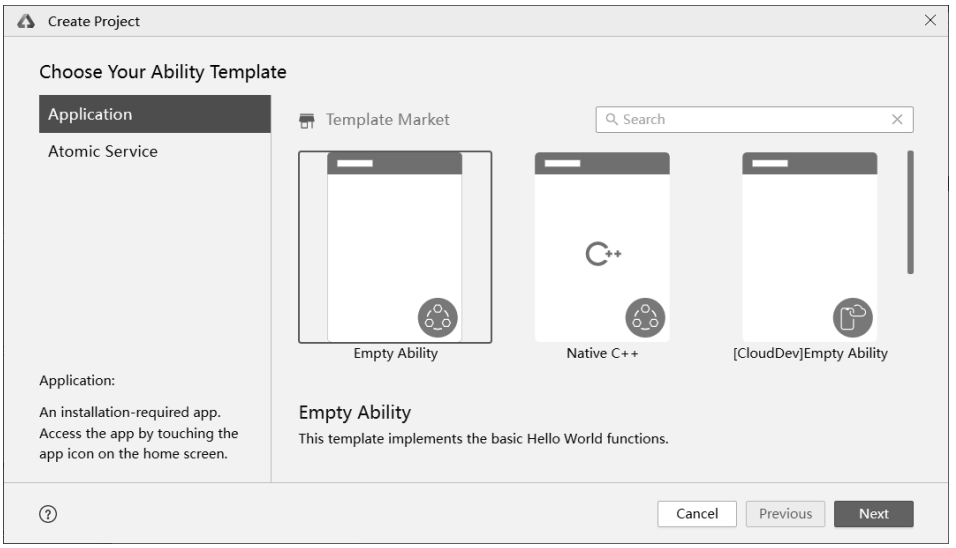

图 1-15　创建新项目

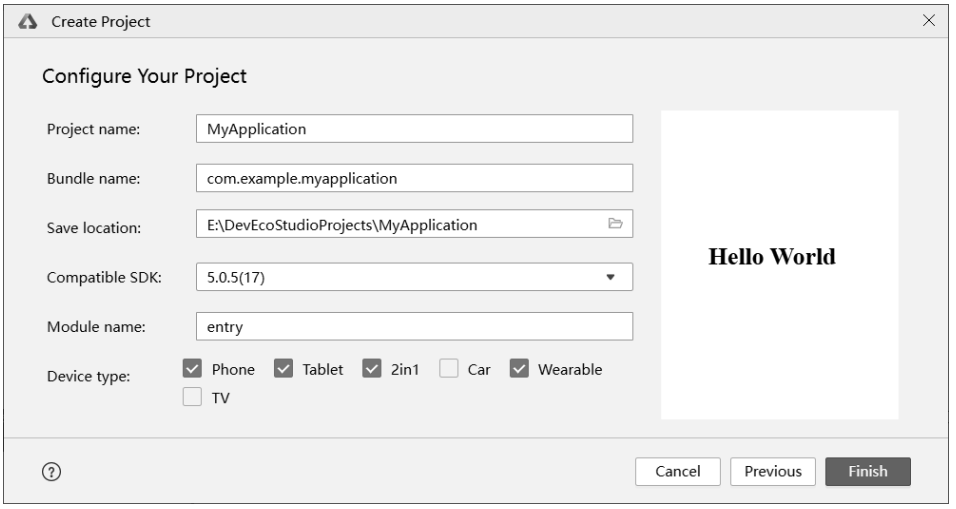

图 1-16　新项目的配置对话框

02 在配置对话框的 Project name 的文本框中输入项目名称；在 Bundle name 的文本框中输入应用的包名；在 Save location 的文本框中输入或选择项目的保存路径；在 Compatible SDK 下拉列表中选择 5.0.5(17)表示采用 OpenHarmony5.0.5 的 SDK17，以便兼容鸿蒙 5 系统；在 Module name 文本框中输入入口模块的名称。

03 单击 Finish 按钮完成配置操作。此时，DevEco Studio 会自动创建一个新项目，并采用先前设置的配置。稍等片刻，DevEco Studio 将呈现刚创建的项目页面，如图 1-17 所示。

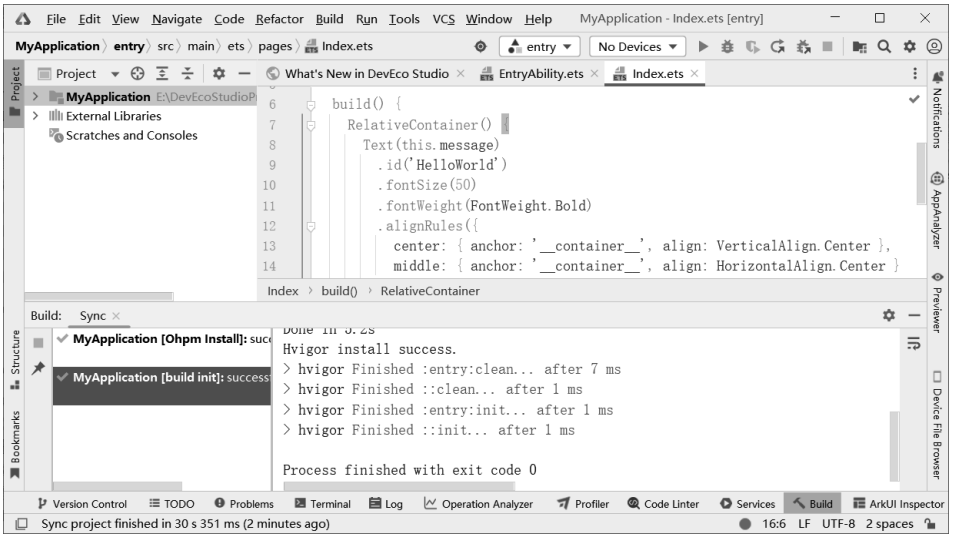

图 1-17　刚创建的项目页面

项目创建完毕后，DevEco Studio会自动打开EntryAbility.ets和Index.ets文件，并默认显示Index.ets的源码。在Index.ets文件的左上方，标签显示该文件的路径结构。注意，主界面左侧有一列竖排标签，从上到下依次是Project、Bookmarks和Structure。单击Project标签，左侧会展开一个小窗口，显示该项目的结构，如图1-18所示。单击Structure标签，左侧将展开一个小窗口，显示该代码的内部方法结构，如图1-19所示。

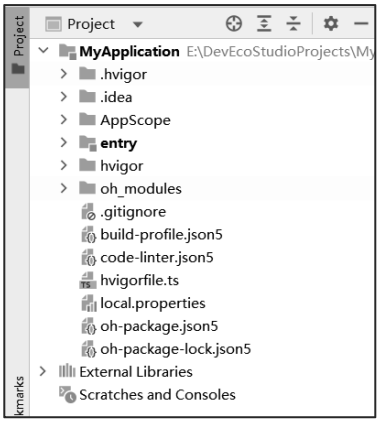

图 1-18　新项目的结构

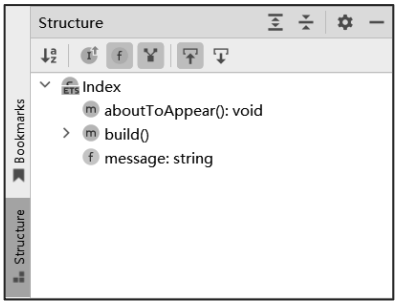

图 1-19　Index.ets 的方法结构

1.3.2　导入已有的项目

本书提供了所有章节的示例源码，为方便学习，读者可将本书源码直接导入DevEco Studio。具体操作步骤如下：

01 打开 DevEco Studio，依次选择菜单栏中的“File→Open”选项，系统将弹出如图 1-20 所示的文件选择对话框。

02 在对话框中选择要导入的项目文件夹路径，确认后单击 OK 按钮。

03 文件对话框关闭后，将弹出一个新的确认对话框（见图 1-21），表示系统正在加载项目配置。在该对话框中有 This Window、New Window 和 Cancel 三个按钮。其中：

- This Window按钮表示在当前窗口打开该项目。
- New Window按钮表示在新窗口打开该项目。
- Cancel按钮表示取消打开操作。

此处我们单击New Window按钮，在新窗口打开App项目，DevEco Studio会自动执行该项目的导入和编译操作。

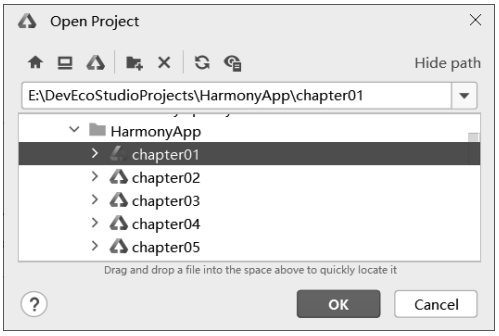

图 1-20　打开 App 项目的文件对话框

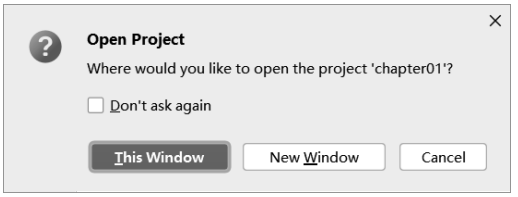

图 1-21　是否开启新窗口的确认对话框

1.3.3　编译 App 项目

与IntelliJ IDEA类似，DevEco Studio支持文件的自动保存功能，开发者在编辑过程中对文件所做

的修改会自动生效，无须手动保存。同时，系统也会对最新代码进行自动编译。如果代码中存在语法错误或逻辑问题，编辑器会在界面中标红提示，帮助开发者及时定位并修复问题。然而，在某些情况下，如DevEco Studio异常关闭，可能导致编译缓存或中间文件损坏，从而影响项目的正常构建。此时，建议开发者手动执行重新编译操作。常见的手动编译方式有以下两种：

（1）在Project区域选中某个模块后，依次选择菜单中的“Build→Make Module ***”选项，该方式会编译指定名称的模块。

（2）先依次选择菜单中的“Build→Clean Project”选项，再依次选择菜单中的“Build→Rebuild Project”选项，表示先清理当前项目，再对整个项目进行重新编译。

无论是编译项目还是编译模块，最终编译结果都展示在DevEco Studio主界面下方的Build窗口中，如图1-22所示。

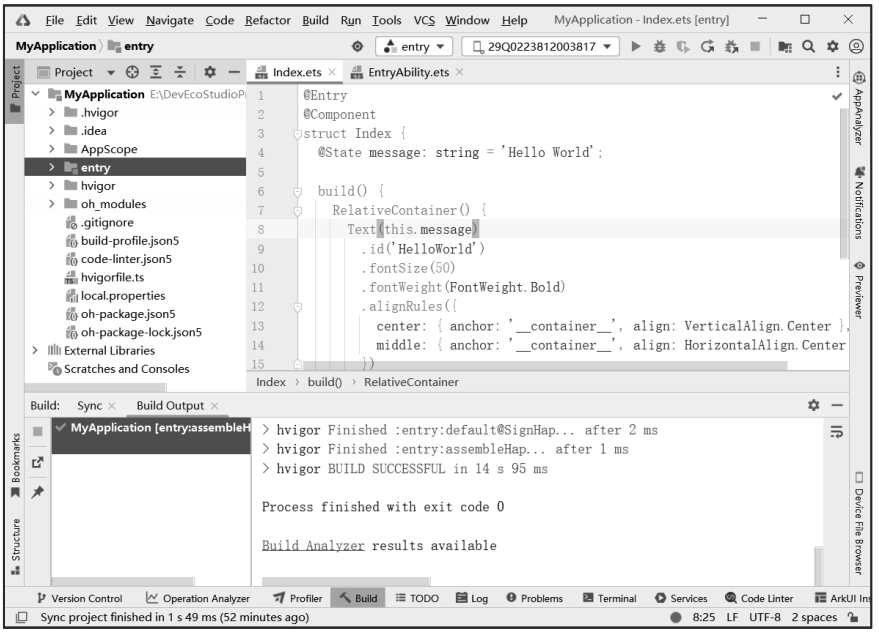

图 1-22　Build 窗口展示

由编译结果可知，当前项目编译耗时14秒95毫秒，未发现错误。

1.4　运行和调试鸿蒙 App

本节介绍使用DevEco Studio运行和调试App的过程：首先叙述如何及时修复App的错误代码，然后描述如何在预览器上运行测试App，最后阐述如何在DevEco Studio中查看App的运行日志。

1.4.1　及时修复错误代码

DevEco Studio具有强大的代码纠错能力，会及时发现错误代码片段，并引导开发者及时处理异常代码。例如，在Index.ets中输入以下代码：

```
    temp: string
```

此时，DevEco Studio会在temp下方显示红色波浪线，表示此处出现了错误代码。将鼠标指针移到temp上方，DevEco Studio会弹出如图1-23所示的提示“Property 'temp' has no initializer and is not definitely assigned in the constructor”，意思是temp属性既没有初始化，也没在构造方法（即构造函数）中赋值。

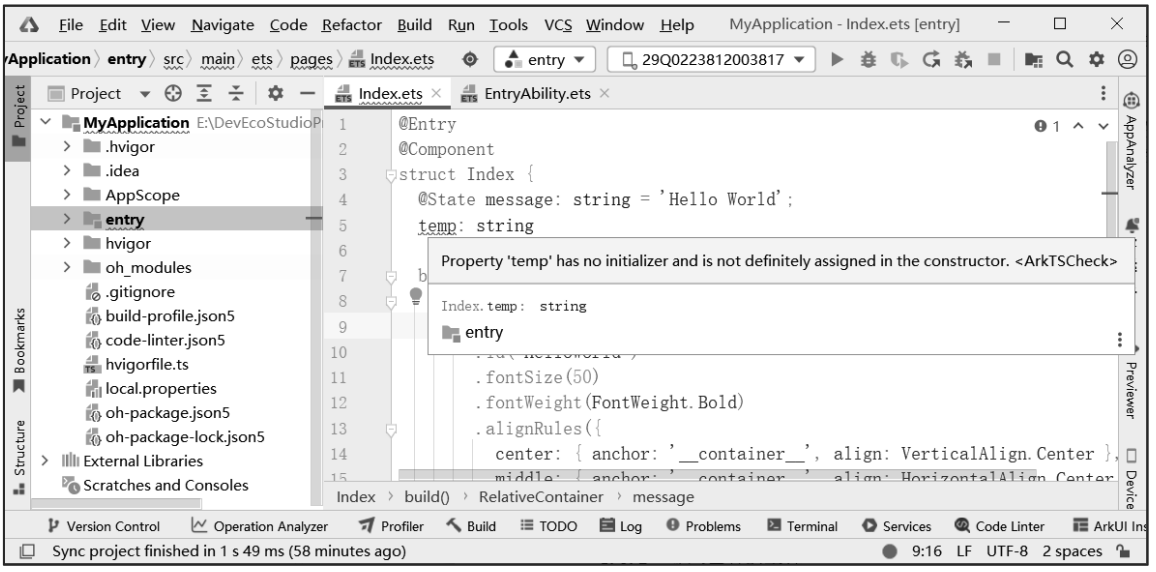

图 1-23　DevEco Studio 的出错提示

根据DevEco Studio的出错提示，需要对temp属性进行初始化并赋值，将代码修改为：

```
temp: string = ''
```

代码修改之后，原来的红色波浪线不见了，说明成功改正了出错代码。

1.4.2　在预览器上运行 App

DevEco Studio主界面右侧有一列竖排标签，从上往下依次为Notifications、AppAnalyzer、Previewer。其中，Previewer（名为预览器）用于预览ETS代码书写的界面效果。

首先，打开Index.ets文件，进入Index.ets编辑窗口，然后单击右侧的Previewer标签。此时，在DevEco Studio主界面的右侧将会弹出预览器窗口，如图1-24所示。可以看到预览器正确显示了Index.ets所要展示的文字内容“Hello World”。

如果修改ETS代码，并且立即预览修改后的界面，可以使用以下3种方式刷新预览效果。

（1）单击Previewer标签，此时预览器窗口会关闭。再次单击Previewer标签，会弹出重新加载的预览器窗口。

（2）在预览器窗口中单击刷新图标（将鼠标指针移过去将提示Reload），预览器也会重新加载当前的ETS界面。

（3）在预览器窗口中单击顺时针弧形箭头图标（将鼠标指针移过去将提示Apply Changes），预览器会把ETS代码改动部分刷新进来。

以上3种预览器刷新方式中，第3种方式（Apply Changes）相当于热刷新，它的刷新速度最快。

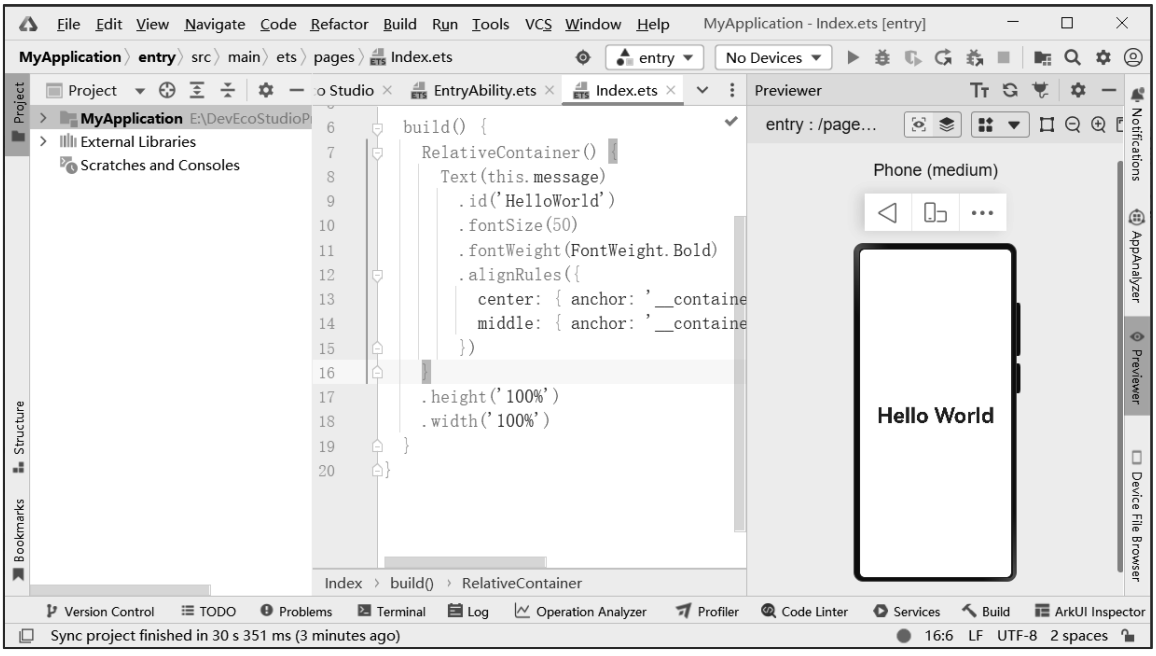

图 1-24　DevEco Studio 的预览器窗口

1.4.3　查看 App 的运行日志

虽然在预览器上能够看到App的运行，但无法看到App的调试信息。如果是编写Java代码，可以通过System.out.println很方便地向IDE的控制台输出日志，DevEco Studio也允许查看App的运行日志，只是鸿蒙App不使用System.out.println，而是采用console工具打印日志。

与System.out.println不同，console工具将各类日志划分为5个等级，每个等级的重要性不同，这些日志等级对应的调用方法按从高到低的顺序依次说明如下：

- console.error：表示错误类型的日志，比如可能导致程序崩溃或发生异常。
- console.warning：表示警告类型的日志。
- console.info：表示信息类型的日志。
- console.log：表示一般日志。
- console.debug：表示调试信息，可把程序运行时的变量值打印出来，方便跟踪调试。

一般而言，日常开发调用console.info方法即可。下面是在Index.ets的Index结构中添加日志信息的代码示例：

```
（完整代码见chapter01/entry/src/main/ets/pages/Index.ets）
  aboutToAppear(): void {
    console.log('我看到你啦');
  }
```

单击顺时针弧形箭头图标 刷新预览器，等待预览器刷新App界面后，单击DevEco Studio底部的Log标签，此时主界面下方弹出日志窗口，如图1-25所示。

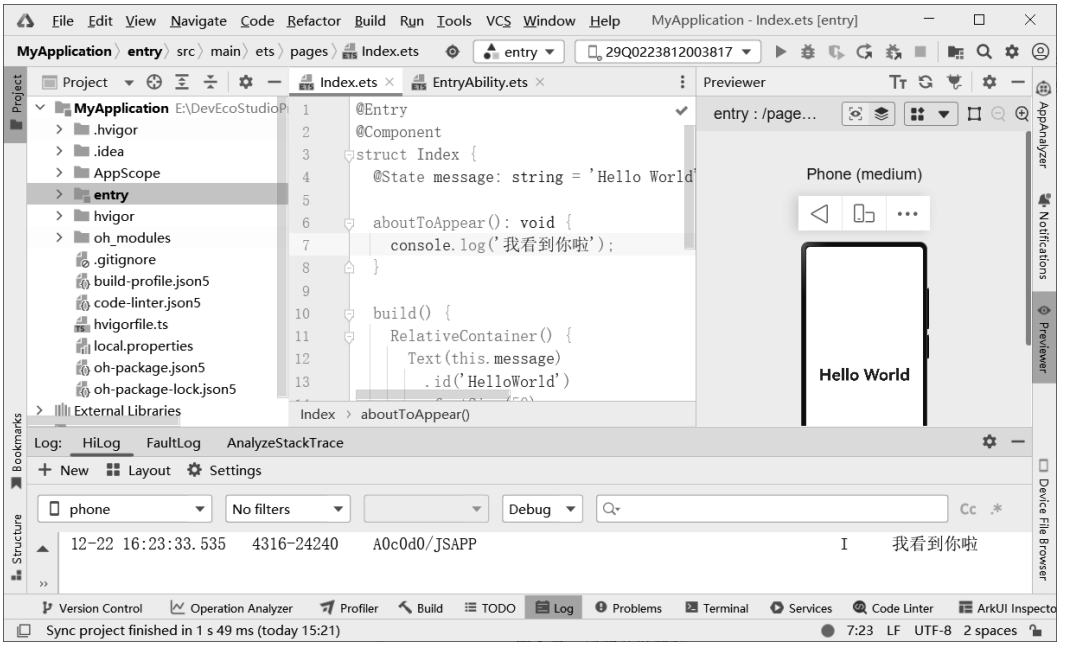

图 1-25　DevEco Studio 的日志查看窗口

日志窗口的顶部是条件筛选框，从左到右依次为：测试设备的类型或者名称（例如phone表示手机）、日志的应用过滤器（例如No filter表示显示所有应用日志）、查看日志的级别（例如只显示级别不低于Debug，即console.debug的日志）、日志内容包含的字符串。经过条件筛选之后，Logcat窗口只显示一行文字“I　　我看到你啦”，说明前面代码输出了级别为info的日志内容“我看到你啦”。

1.5　小结

本章主要介绍了鸿蒙开发环境的搭建过程，包括：鸿蒙开发简介（鸿蒙系统的发展历程、鸿蒙系统的3大特性和总体架构、鸿蒙应用的技术理念）、搭建DevEco Studio开发环境（计算机配置要求、安装DevEco Studio、修改DevEco Studio的常用设置）、创建并编译鸿蒙App项目（创建新项目、导入已有的项目、编译App项目）、运行和调试鸿蒙App（及时修复错误代码、在预览器上运行App、观察App的运行日志）。

通过本章的学习，读者可以掌握DevEco Studio的基本操作技能，能够使用自己搭建的DevEco Studio环境创建简单的App项目，并在预览器上观察测试页面的预览效果。

1.6　动手练习

请上机实验搭建App的开发环境，主要步骤如下：

（1）下载并安装DevEco Studio的最新版本或者本书使用的版本（DevEco Studio 6.0.0 Release）。
（2）创建一个鸿蒙App项目“Hello World”。
（3）在预览器上刷新第（2）步创建App的Index.ets，观察能否看到“Hello World”字样。

第 2 章

鸿蒙App开发基础

2

本章介绍基于鸿蒙系统的App开发常识，主要内容包括：鸿蒙App开发的特点，鸿蒙App的项目结构以及鸿蒙App的调试和打包的过程。

2.1 鸿蒙 App 的开发特点

本节介绍鸿蒙App的开发有别于其他软件开发的特点。例如，App能在哪些操作系统上运行、App开发时使用的编程语言以及App能操作哪些数据库等。了解鸿蒙App的开发和运行环境后，才能有的放矢，不走弯路。

2.1.1 App 的运行环境

App是运行在手机上的一种应用软件，而应用软件依附于操作系统。在开机时，无论是计算机还是手机，都将显示桌面，这个桌面便是操作系统的工作台。个人计算机的操作系统主要有微软的Windows和苹果的macOS，而智能手机的流行操作系统有3种，分别是：安卓手机的Android、苹果手机的iOS以及华为手机的HarmonyOS（即鸿蒙系统）。

本书讲述的鸿蒙App开发是基于HarmonyOS上的应用开发。HarmonyOS基于开源的OpenHarmony内核，而非Linux内核，因此鸿蒙App无法在Linux系统上运行。为了兼容已有的应用生态，早期的HarmonyOS在底层兼容了Android系统的AOSP内核，因此HarmonyOS 4及更早版本都兼容安卓App。但从HarmonyOS 5.0（也称HarmonyOS Next）开始，鸿蒙不再兼容AOSP，也就是说，无法安装安卓App，只能安装鸿蒙App才能运行。

DevEco Studio是华为官方推出的App开发环境，提供了支持Windows和macOS操作系统的安装包。这就带来了一个问题：虽然开发者可以在计算机上安装DevEco Studio并使用它来开发鸿蒙App，但编译出来的鸿蒙App却无法直接在计算机上运行。这种情况令人困惑，因为通常在学习C语言、Java或Python等语言时，在计算机的开发环境中都能直接查看程序运行的过程。即便是J2EE开发，开发者也能通过浏览器在网页中观察Web程序的运行情况。然而，鸿蒙的App竟然无法在计算机上直接运行，那么该如何验证App的界面展示及业务逻辑是否正确呢？

为了验证鸿蒙App的业务功能是否正常，可采取下列3种测试方式：

（1）利用DevEco Studio提供的预览器：通过预览器，可以调试简单的界面组件交互，但无法验证数据库、网络通信等复杂操作。

（2）使用DevEco Studio创建模拟器：开发者可以启动模拟器，在模拟器上运行鸿蒙App应用，但模拟器会占用大量内存，且无法覆盖所有的测试场景。

（3）使用真机测试鸿蒙App：该方式在实际开发中更为常见。由于模拟器运行在计算机上，会占用计算机的CPU和内存，会拖慢计算机的运行速度；此外，由于模拟器仅仅是模拟，无法完全验证App的所有功能，因此需要通过真机测试来确保App的全部功能和可靠性。

2.1.2　App 的开发语言

鸿蒙App的开发语言有3种：ArkTS、仓颉和C/C++，分别说明如下。

1. ArkTS

ArkTS是鸿蒙生态的应用开发语言。它在保持TypeScript（简称TS）基本语法风格的基础上，进一步强化静态检查和分析，使程序在开发阶段检测出更多的错误，提高代码的健壮性和运行性能。同时，ArkTS提供了声明式UI范式、状态管理支持等特性，使得开发者能够以更简洁、自然的方式开发高性能应用。

许多有网页开发经验的读者可能对ArkTS较为熟悉，因为ArkTS语言源自TypeScript语言，TypeScript语言又源自JavaScript（简称JS）语言，而JavaScript是网页开发的主流脚本语言。三者的代码文件与使用场景的区别如下：

（1）JavaScript：代码文件扩展名为js，常用于HTML5网页开发。

（2）TypeScript：代码文件扩展名为ts，微信小程序和抖音小程序等应用采用TypeScript作为开发语言。

（3）ArkTS：代码文件扩展名为ets，是鸿蒙App的开发语言。

2. 仓颉

仓颉编程语言是一门面向全场景智能的新一代编程语言，专注于鸿蒙智能化、原生全场景适配、高性能、强安全性。未来，仓颉编程语言将融入鸿蒙生态，为开发者提供良好的编程体验。本书以ArkTS作为鸿蒙App的主要开发语言，暂不介绍仓颉在App开发中的应用。

3. C/C++

无论是ArkTS还是仓颉，它们都是将系统功能封装后，通过API接口提供给开发者调用。由于中间多了一层翻译操作，因此在高速计算的场合其性能有所下降。为了应对性能要求高的场景，鸿蒙系统支持通过C/C++编程进行NDK开发。

NDK（全称Native Development Kit）是HarmonyOS SDK提供的Native API、配备了相应的编译脚本和编译工具链的集合，方便开发者使用C/C++语言实现应用的关键功能。需要注意的是，NDK只覆盖了HarmonyOS的一些基础底层能力，如C运行时基础库libc、图形库、窗口系统、多媒体、压缩库、面向ArkTS/JS与C跨语言的Node-API等，并没有提供ArkTS/JS API的完整能力。

采用C/C++编程的NDK开发适用于下列场景：

（1）对性能要求高的场景，如游戏、物理模拟等计算密集型应用。

（2）复用已有C/C++库的场景。

（3）针对CPU特性进行专项定制库的场景，如Neon加速。

2.1.3　App 连接的数据库

在学习Java编程时，我们通常会学如何通过JDBC连接数据库进行记录的增删改查操作，要连接的数据库可能是MySQL、Oracle，或者是SQL Server。然而，手机应用不能直接操作上述这几种数据库，因为数据库软件也需要像应用软件那样安装到操作系统中。例如，MySQL提供了Windows系统的安装包和Linux系统的安装包，但并没有为鸿蒙系统提供安装包，因此MySQL无法在鸿蒙系统上安装，这也意味着手机中的鸿蒙系统中的App不能直接连接MySQL数据库。

既然像MySQL这样的企业级数据库无法在手机上安装，鸿蒙App是如何管理业务数据记录呢？实际上，鸿蒙系统内置了专门的数据库（基于SQLite组件），SQLite遵循关系数据库的设计理念，它的SQL语法类似于MySQL。不同之处在于，SQLite无须单独安装，它内嵌到应用进程中，因此App无须配置连接信息即可直接执行增删改查操作。由于SQLite嵌入到了应用程序中，省去了配置数据库服务器的开销，因此它被归类为嵌入式数据库。

SQLite数据库文件通常存储在用户的手机本地，而开发者无法直接访问用户的设备，这使得直接获取App中存储的业务数据（如用户的注册信息、购物记录等）变得困难。相比之下，Java Web应用通常将业务数据集中存储在后端数据库服务器中，开发者只需登录数据库服务器即可方便地查询和导出所需的用户数据。

手机端的App，包括其程序代码和内置的嵌入式数据库（如SQLite），是一个独立且完整的程序实体，主要负责手机上的用户交互与信息处理，这种实体被称为客户端。而后端的Java Web服务，包括Web代码和数据库服务器，构成另一个独立运行的程序实体，它只负责后台的业务逻辑与数据库操作，这一实体被称为服务端。客户端与服务端之间通过HTTP接口通信，当客户端需要向服务端发送信息或从服务端获取信息时，会主动发起HTTP请求，服务端接收到请求后，根据业务规则处理数据，并将处理结果返回给客户端。通过这种机制，客户端可以通过服务端间接实现对后端数据库（如MySQL）的读写操作。具体的信息交互过程如图2-1所示。

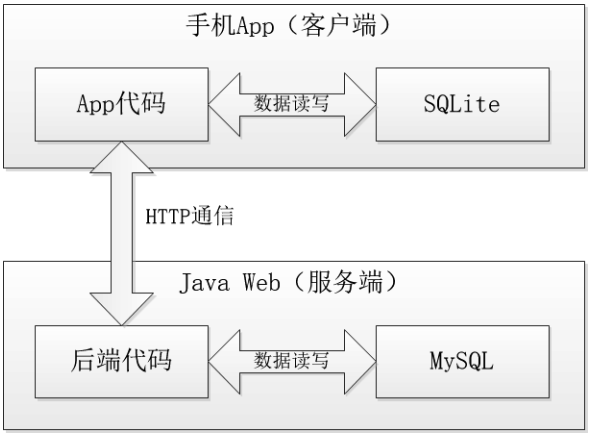

图 2-1　客户端与服务端分别操作的数据库

从上述描述可以看出，具备用户管理功能的App软件，不仅仅是手机上的应用，还包括与之对应的Java Web服务。手机中的客户端App面向的是手机用户，客户端App与用户通过手机屏幕交互；而后

端的服务程序则面向的是手机客户端App，客户端与服务端之间通过HTTP接口交互。客户端和服务端这种多对一的架构关系如图2-2所示。

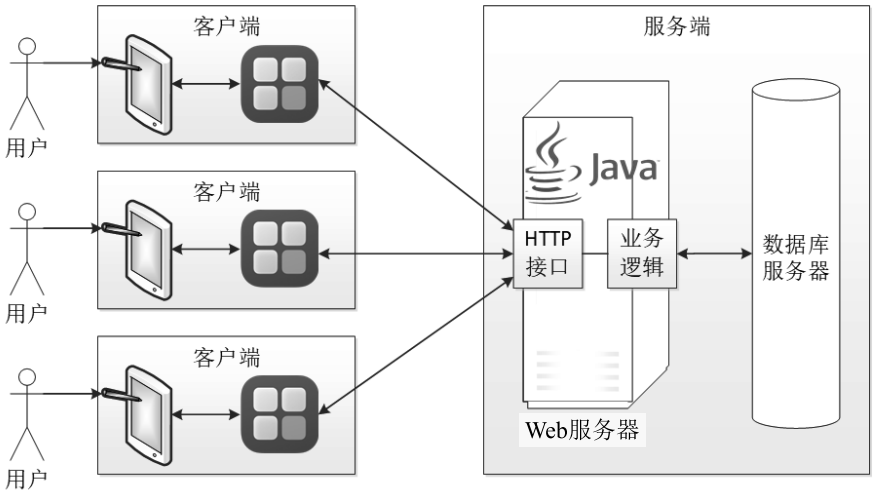

图 2-2　客户端与服务端的多对一架构关系图

总结一下，鸿蒙App能够直接操作内置的SQLite数据库，但无法直接操作像MySQL这种企业级数据库。开发者需要事先搭建好服务端程序（如Java Web），然后客户端与服务端通过HTTP接口进行通信，再由服务端去操作MySQL这样的数据库服务器。

2.2　鸿蒙 App 的项目结构

本节介绍鸿蒙App项目的基本结构及其常用配置。首先，说明项目和模块的区别以及项目内部各目录与配置文件的用途；然后，详细讲解编译配置文件build-profile.json5的配置信息；最后，介绍运行配置文件module.json5的节点信息及其属性。

2.2.1　App 项目目录结构

鸿蒙App项目分为两个层次：第一个层次是项目，开发者依次选择菜单中的“File→New→Create Project”选项，即可创建新项目；第二个层次是模块，开发者依次选择菜单中的“File→New→New Module”选项，即可在当前项目创建新模块。

模块依附于项目，常见的模块类型有entry和feature。其中，entry类型为入口模块，每个App项目只能有一个入口模块；feature类型为动态特性模块，App项目可以有多个动态特性，也可以没有动态特性。通常所讲的“编译运行App”指的是运行App项目的入口模块，而不能直接运行动态特性模块。

单击DevEco Studio左上角竖排的Project标签，可以看到App工程的项目结构，如图2-2所示。项目结构默认以Project方式呈现，看起来有些杂乱；单击Project下拉框，将其切换到Ohos方式后，呈现出来的项目结构如图2-3所示。

从图2-3中可以看到，该项目下有3个主要分类：AppScope、entry和configuration，接下来我们逐一说明。

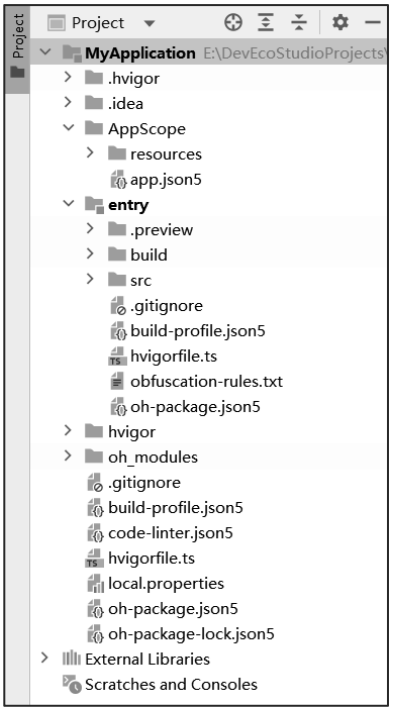

图 2-2　Project 形式的项目结构

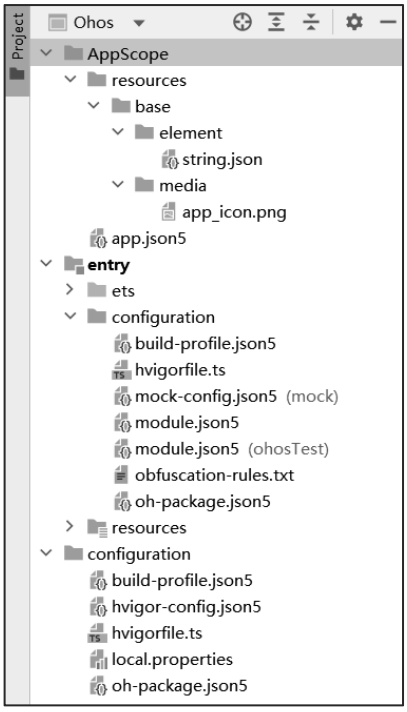

图 2-3　Ohos 形式的项目结构

1. AppScope

该类别用于存放App的安装信息，主要存放在AppScope目录下的app.json5中。下面列出的是app.json5的初始内容，每行后面的注释文字都说明了各字段的含义。

```
{
  "app": {
    "bundleName": "com.example.myapplication",      // App包名，不可重复
    "vendor": "example",                 // 公司名称
    "versionCode": 1000000,              // 版本编号
    "versionName": "1.0.0",              // 版本名称
    "icon": "$media:app_icon",           // App图标。图标文件为media目录下的app_icon
    "label": "$string:app_name"          // App名称。名称文本的键名为string.json的app_name
  }
}
```

除了app.json5之外，还有App的名称配置文件AppScope/resources/base/element/string.json，以及App的图标文件AppScope/resources/base/media/app_icon.png。其中，string.json的配置内容如下：

```
{
  "string": [
    {
      "name": "app_name",
      "value": "MyApplication"
    }
  ]
}
```

由上述代码可以看到，string.json配置的键名name的字段值为app_name，对应的键值value为MyApplication。因此，当App安装到手机后，桌面上的App名称即为MyApplication。

2. entry

该类别用于存放App的入口模块信息，包括App的代码文件和图片等资源文件，以及App模块的运行配置文件module.json5等。以下是该类别的子目录说明：

- ets：存放App的所有ETS代码文件，实际的目录路径为entry/src/main/ets。
- configuration：存放App项目的模块级别配置文件，常用的文件为运行配置文件module.json5，实际的文件路径为entry/src/main/module.json5。
- resources：存放App所需的图片、音频、视频等资源文件，还有App的页面配置、颜色配置、字符串配置等json文件，实际的目录路径为entry/src/main/resources。

3. configuration

该类别存放App项目用于编译构建的项目级别配置文件，包括构建配置文件、编译构建任务脚本、混淆规则文件、依赖的共享包信息等。各配置文件简要说明如下：

- build-profile.json5：App项目的构建配置文件，包括应用签名、产品配置等，具体说明参见2.2.2节。
- hvigor-config.json5：构建工具Hvigor的配置文件，用于指定hvigor的版本、构建依赖以及构建行为的配置参数。
- hvigorfile.ts：App项目的编译构建任务脚本，开发者可以自定义编译构建工具版本、控制构建行为的配置参数。
- obfuscation-rules.txt：混淆规则文件。开启混淆后，在使用Release模式进行编译时，将对代码进行编译、混淆及压缩处理，以保护代码资产。
- oh-package.json5：存放依赖库的信息，包括App依赖的第三方库和共享包。

2.2.2　编译配置文件 build-profile.json5

build-profile.json5是App项目的编译配置文件，分为项目级与模块级。项目级的build-profile.json5位于App项目的根目录，模块级的build-profile.json5位于对应的模块目录中。其中的buildOption节点在项目级文件和模块级文件均可进行配置。若有相同名称的字段，则以模块级的字段为准；若是不同名称的字段，模块级的buildOption配置将继承项目级的配置。

项目级的build-profile.json5包含两大类编译配置：应用编译节点app和模块列表节点modules。下面分别对其进行说明。

1. 应用编译节点app

app节点包含应用的编译配置信息，内部包含签名配置节点（signingConfigs）、产品类型节点（products）和构建模式集合节点（buildModeSet）等，分别说明如下。

1）signingConfigs

signingConfigs节点保存App的签名配置，安装到手机上的所有App都需要进行签名。该节点的下级字段说明如下：

- name：签名配置的名称。
- material：签名配置相关的材料，如密码、证书等。
- type：签名类型。对于HarmonyOS Next来说，固定填写HarmonyOS。

2）products

products节点保存App的产品信息，该节点的下级字段说明如下：

- name：产品的名称，必须存在名为default的产品（product）。
- signingConfig：产品的签名配置名称，该名称应与signingConfigs节点中定义的某个签名配置保持一致。
- bundleName：产品的包名。
- buildOption：产品的编译构建配置。
- runtimeOS：产品的运行环境。对于HarmonyOS 6来说，固定填写HarmonyOS。
- targetSdkVersion：运行所需的目标SDK版本。推荐设置为最新的SDK版本，以便使用最新的特性和优化。
- compatibleSdkVersion：兼容的最低SDK版本。对于HarmonyOS 6来说，版本号不能低于“5.0.0(12)”。

3）buildModeSet

buildModeSet节点保存App的构建模式，该节点的下级字段说明如下：

- name：构建模式名称。默认生成debug和release两个名称，其中debug表示开发阶段，release表示发布阶段。
- buildOption：构建模式使用的具体配置信息。

2. 模块列表节点（modules）

modules节点包含项目中所有的模块配置，至少需要有一个入口模块。该节点的下级字段说明如下：

- name：模块名称，必须与module.json5文件中的module.name保持一致。
- srcPath：模块的源码路径，采用模块根目录相对子项目根目录的相对路径。
- targets：模块的target信息，用于定制多目标构建产物。

2.2.3　运行配置文件 module.json5

module.json5是App模块的运行配置文件，每个模块都有单独的module.json5。该文件内部的各节点说明如下：

- name：标识当前模块的名称，确保该名称在整个应用中唯一。
- type：标识当前模块的类型。它的取值说明见表2-1。

表2-1　module.json5的type字段的取值说明

类　型　值	类型说明
entry	应用的主模块，也是应用的入口模块
feature	应用的动态特性模块
har	静态共享包模块
shared	动态共享包模块

- description：标识当前模块的描述信息。
- mainElement：标识当前模块的入口UIAbility名称。

- deviceTypes：标识当前模块可以运行在哪类设备上。设备数组中的phone表示手机，tablet表示平板电脑，2in1表示PC，即个人计算机（在传统PC外接键盘和鼠标的基础上增加屏幕触摸的输入形式，所以两个并在一起称为2in1）。
- deliveryWithInstall：标识当前模块是否在用户主动安装时一起安装，即该Module对应的HAP是否跟随应用一起安装。为true表示主动安装时安装，为false表示主动安装时不安装。
- installationFree：标识当前模块是否支持免安装特性。为true表示支持免安装特性，且符合免安装约束；为false表示不支持免安装特性。
- pages：标识当前模块的profile资源，用于列举每个页面信息。
- abilities：标识当前模块中UIAbility的配置信息，只对当前UIAbility生效。abilities内部的子节点说明如下：
  - name：标识当前UIAbility组件的名称，确保该名称在整个应用中唯一。
  - srcEntry：标识入口UIAbility的ETS代码路径。
  - description: 标识当前UIAbility组件的描述信息，采用描述信息的资源索引，以支持多语言。
  - icon：标识当前UIAbility组件的图标，取值为图标资源文件的索引。
  - label：标识当前UIAbility组件显示给用户显示的名称，采用该名称的资源索引，以支持多语言。
  - startWindowIcon：标识当前UIAbility组件启动页面图标资源文件的索引。
  - startWindowBackground：标识当前UIAbility组件启动页面背景颜色资源文件的索引。
- exported: 标识当前UIAbility组件是否可以被其他应用调用。为true表示可以被其他应用调用，为false表示不能被其他应用调用。

如果开发者拿到一个App项目的源码，如何通过module.json5找到App首页的ETS源码呢？下面逐步介绍首页ETS代码的寻找过程：

01 检查各模块的 module.json5，找到 type 字段值为 entry 的模块，这便是应用的主模块，也是应用的入口模块。

02 在主模块 module.json5 中找到 mainElement 字段，这里存放的是当前模块的入口 UIAbility 名称，通常为 EntryAbility。

03 module.json5 的 abilities 字段配置了当前模块所有的 UIAbility 数组，在此可以找到 EntryAbility 的代码路径，通常为“./ets/entryability/EntryAbility.ets”。

04 打开 EntryAbility.ets 的源码，找到 onWindowStageCreate 方法，该方法内部调用了 windowStage.loadContent 接口，其作用是加载页面内容，并且 loadContent 方法加载的页面路径为 pages/Index，到这步就可以找到 App 首页的源码路径 pages/Index.ets。

2.3　鸿蒙 App 的调试打包

本节介绍鸿蒙App的调试和打包过程：首先描述如何通过计算机把App安装到真机上测试；然后阐述如何为测试App添加开发阶段的默认签名；最后讲解应用程序包在不同阶段的3种形态，以及HAP、HAR、HSP等文件的区别。

2.3.1　连接真机测试

真机指的是真实的手机设备，而非预览器或模拟器。鸿蒙App要求真机的操作系统必须为HarmonyOS 5.0或者更高的版本，如HarmonyOS 6.0。

在使用真机调试鸿蒙App时，还需满足以下3个条件。

1. 使用数据线把手机连接到计算机

手机的电源线拔掉插头即可作为数据线使用。将数据线长方形的一端插入计算机的USB插槽，即可完成手机与计算机的连接。

2. 打开手机的开发者选项并启用USB调试

手机出厂时默认关闭了开发者选项，需要手动开启开发者选项才能调试App。打开鸿蒙手机自带的“设置”程序，依次单击“系统”→第二项的“***的Mate60”选项，打开“关于本机”页面。接着，连续单击该页面的“软件版本”选项，会弹出如图2-4所示的“开启‘开发者选项’”窗口。

单击窗口右下角的“确认重启并开启”按钮，手机会自动重启，重启后就能在“设置”程序的“系统”菜单下找到“开发者选项”。进入“开发者选项”页面，启用“开发者选项”和“USB调试”两处设置，随后弹出如图2-5所示的“允许USB调试”窗口。

单击右下角的“允许”按钮，即表示允许手机通过USB接口安装调试应用，设置完成的“开发者选项”页面如图2-6所示。

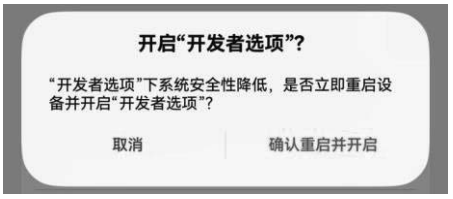

图 2-4　“开启‘开发者选项’”窗口

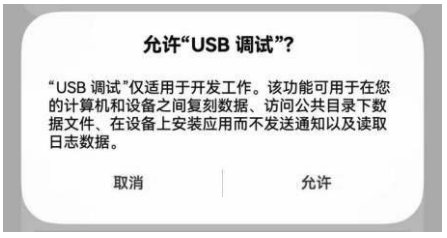

图 2-5　“允许‘USB 调试’”窗口

图 2-6　“开发者选项”的设置页面

3. 确保手机处于解锁状态

在锁屏状态下，DevEco Studio会拦截App的安装行为，因此要保证手机处于解锁状态，才能顺利将鸿蒙App安装到手机上。

完成以上操作后，即可通过计算机将鸿蒙App安装到真机上。启动DevEco Studio，在顶部中央的执行区域找到已连接的手机信息，如图2-7所示。此时，设备信息显示为“HUAWEI Mate 60 Pro[29Q0223812003817]”。单击手机编号右边的三角运行按钮，接下来就是等待DevEco Studio往手机上安装App。

图 2-7　找到已连接的真机设备

2.3.2　给 App 添加开发签名

要在真机上运行和调试App，除了单击上一小节末尾提到的三角运行按钮外，还可以依次选择顶部菜单中的“Run→Run 'entry'”选项，这两种方式都能把App直接安装到手机上。

首次在真机上安装和调试某个App时，DevEco Studio下方的Run区域会报错“error: no signature file.”，意思是缺少签名文件，如图2-8所示。

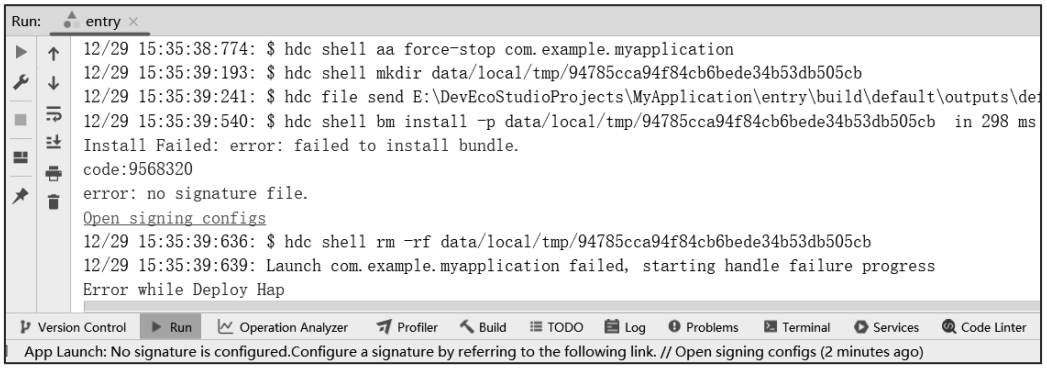

图 2-8　首次安装时报出缺少签名文件的错误

为了给App添加默认的签名文件，可单击图2-8中的“Open signing configs”链接，弹出如图2-9所示的签名配置窗口。

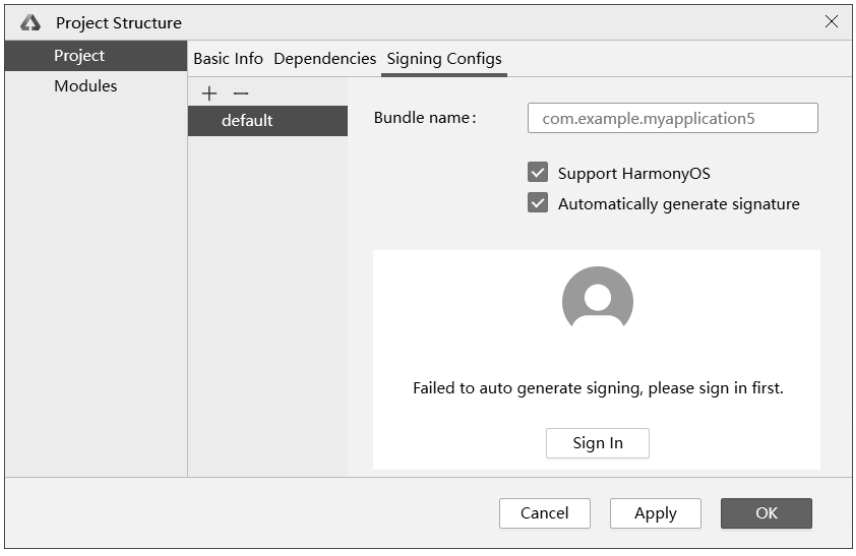

图 2-9　初始的签名配置窗口

在签名配置窗口右下区域，单击“Sign In”按钮，会启动浏览器并跳转到华为开发者网站的登录页面，此时会提示输入账号和密码，如图2-10所示。

如果还没有注册华为账号，就先单击注册链接完成注册流程。如果已有华为账号，可以直接输入华为账号及密码进行登录，登录成功后会出现如图2-11所示的“HUAWEI DevEco Studio想要访问您的华为账号”窗口。

图 2-10　华为开发者网站的登录页面

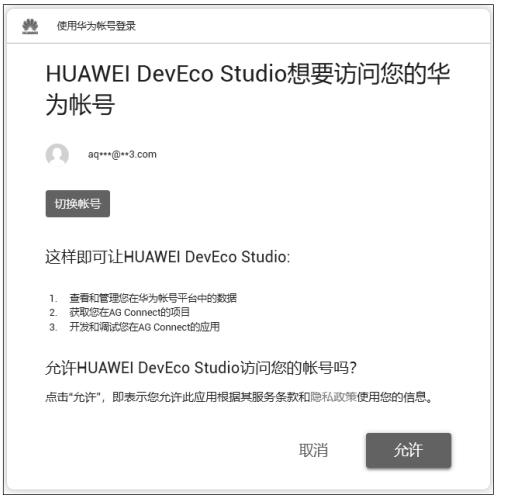

图 2-11　访问华为账号的确认窗口

单击“允许”按钮，浏览器会提示“您已成功登录HUAWEI DevEco Studio客户端，请返回DevEco Studio进行下一步操作”。此时回到DevEco Studio界面，弹出许可条款对话框，如图2-12所示。

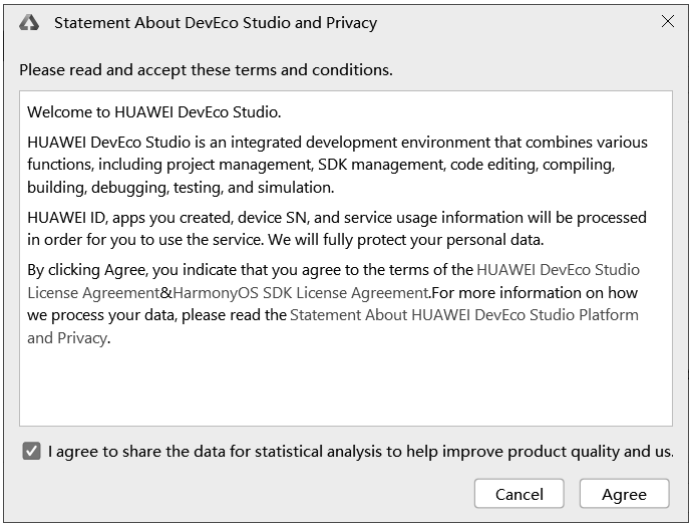

图 2-12　许可条款对话框

勾选“I agree to share the data for statistical analysis to help improve product quality and us.”复选框，表示同意上述条款，然后单击Agree按钮，打开签名配置对话框，在该对话框中可以看到已经自动填上了默认的签名信息，如图2-13所示。

可见，签名配置窗口上方的“Automatically generate signature”栏默认为勾选状态，可以先单击该复选框取消勾选，再单击该复选框重新勾选，接着单击窗口右下角的Apply按钮，DevEco Studio才会自动把签名信息添加到App项目级别build-profile.json5的signingConfigs节点中。

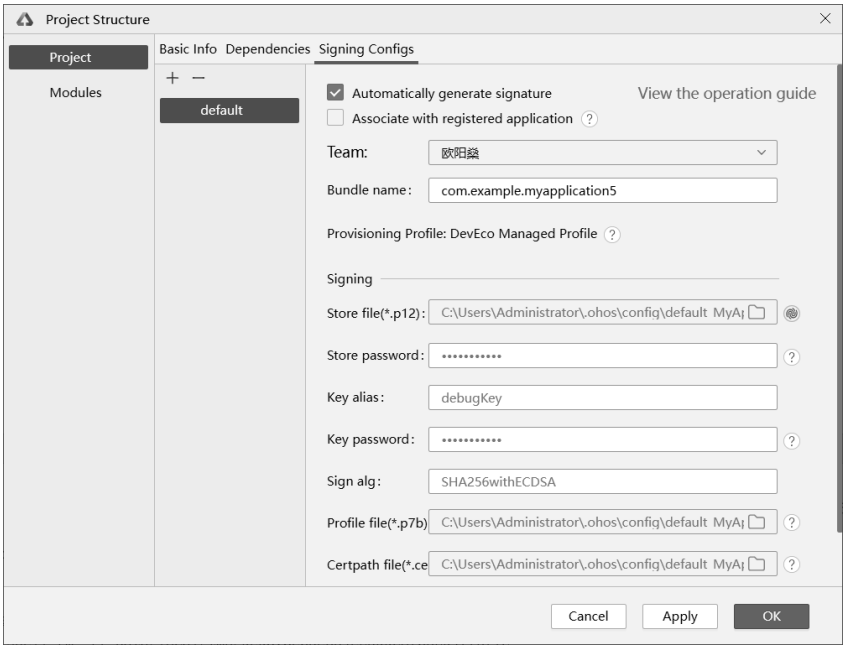

图 2-13　填上了默认签名信息的签名配置窗口

单击OK按钮，DevEco Studio会自动配置默认签名。此时，再次往真机上安装调试App，DevEco Studio下方的Run区域会提示“Launch com.example.myapplication success”，如图2-14所示，说明测试App安装成功，并且可以在手机上看到App的测试页面。

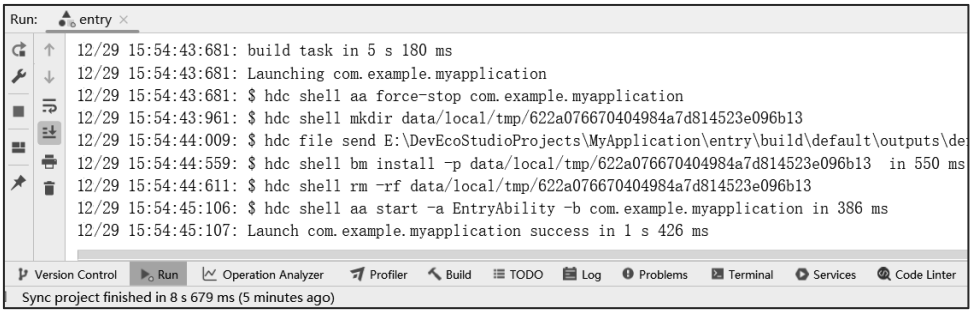

图 2-14　成功安装默认签名的测试 App

2.3.3　App 的编译态和发布态

在2.3.2节中，我们成功将测试App安装到手机上，接下来在App项目的entry/build/default/outputs/default目录中，可以找到两个HAP文件：entry-default-signed.hap和entry-default-unsigned.hap。其中，entry-default-signed.hap是已签名的App安装包，可直接安装到鸿蒙手机上；而entry-default-unsigned.hap为未签名的App安装包，不能直接安装到鸿蒙手机，必须经过签名后才能安装。

那么，为什么DevEco Studio会把App项目编译为.hap文件，而每个App项目在编译后只有一个HAP文件呢？这是因为鸿蒙App的应用程序包在不同阶段拥有着不同的组织形态。从使用DevEco Studio创建App项目开始，到App安装到手机为止，应用程序包依次经历了开发态、编译态和发布态三种形态，具体说明如下。

1. 开发态

使用DevEco Studio创建App项目后，DevEco Studio自动生成的App源码目录就处于开发态。关于开发态的项目结构，可以参考2.2节的介绍。

2. 编译态

不同类型的Module（App模块）编译后会生成对应的HAP、HAR、HSP等文件，开发态视图与编译态视图的对照关系如图2-14所示。

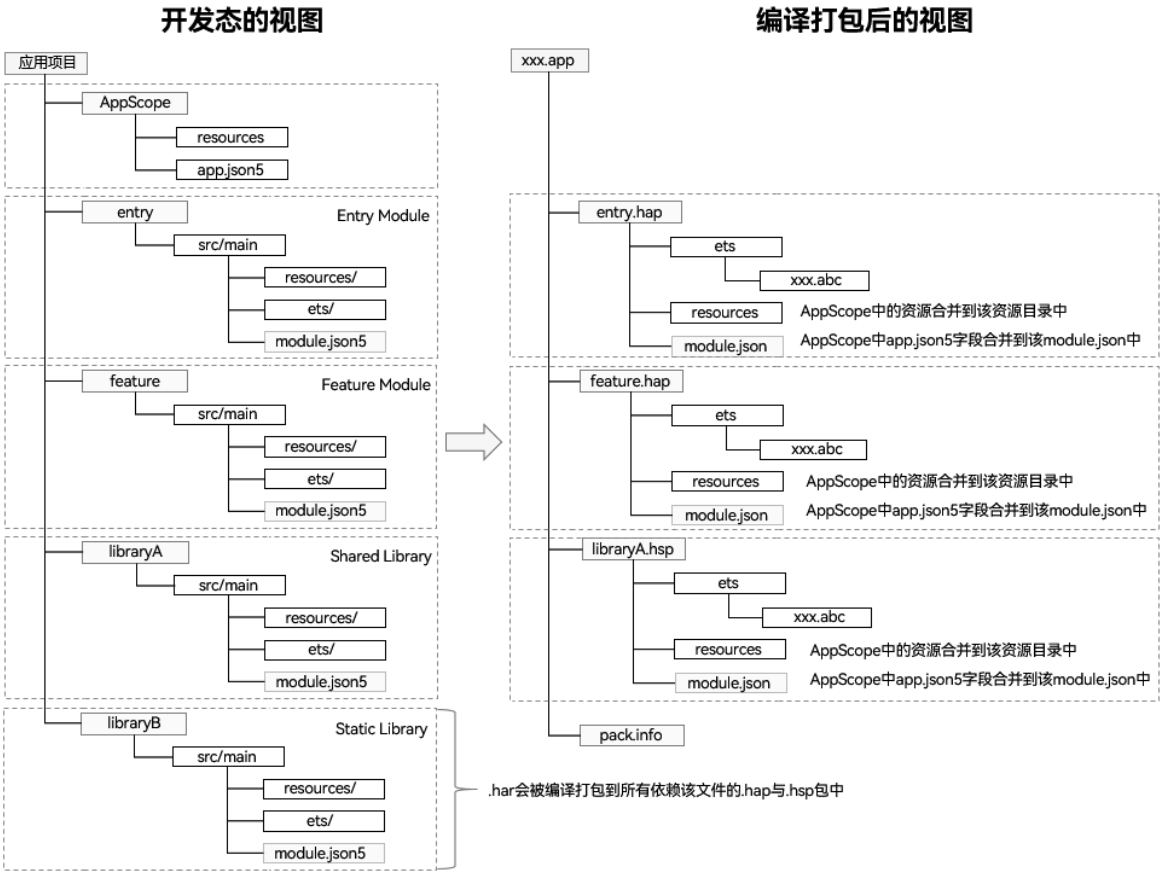

图 2-14　开发态与编译态的 App 项目结构视图

从开发态到编译态，Module中的文件会发生如下改变：

- ets目录：ArkTS源码编译生成.abc文件。
- resources目录：AppScope目录下的资源文件会合并到Module下的资源目录中，合并后的目录下如果存在重名文件，编译打包后只会保留AppScope目录下的资源文件。
- 模块配置文件：AppScope目录下的app.json5文件字段会合并到Module下的module.json5文件中，编译后生成HAP或HSP最终的module.json文件。

注意　在编译HAP和HSP时，会把它们所依赖的HAR直接编译进HAP和HSP中，但它们所依赖的HSP不会被集成到HAP和HSP中。

3. 发布态

每个应用至少包含一个.hap文件，可能包含若干个.hsp文件，也可能不含。一个应用中的所有.hap与.hsp文件合在一起构成Bundle，并且有唯一的bundleName标识（bundleName标签来自App项目的配置文件app.json5，详见2.2.1节的介绍）。

当应用发布到应用市场时，需要将Bundle打包为一个以.app为后缀的文件用于上架，这个.app文件称为App Pack（Application Package）。同时，DevEco Studio工具会自动会生成一个pack.info文件。pack.info文件描述了App Pack中每个HAP和HSP的属性，包含App中的包名bundleName和版本号versionCode等信息，以及模块中的name、type和abilities等信息。

App Pack与HAP/HSP的区别如下：

（1）App Pack：是发布上架到应用市场的基本单元，但不能直接在设备上安装和运行。

（2）HAP/HSP：在应用签名、云端分发、端侧安装时，都是以HAP/HSP为单位进行签名、分发和安装的。

App项目从编译发布到上架部署的流程如图2-15所示。

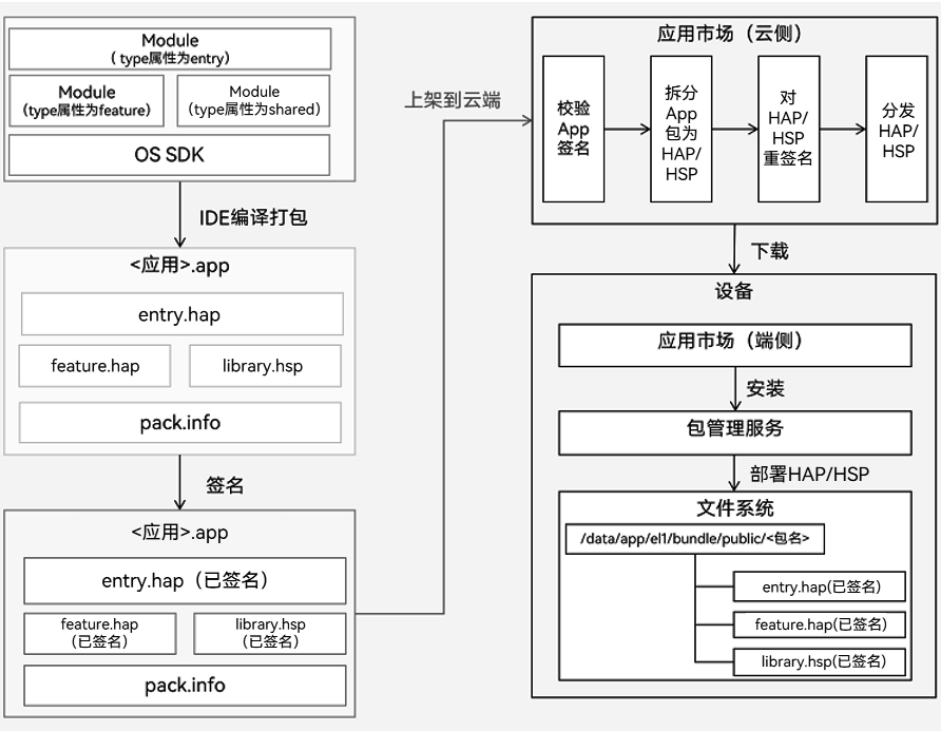

图 2-15　App 项目的编译发布与上架部署流程图

上述内容多次提到HAP、HAR、HSP等文件，接下来分别解释这3种文件的功能和使用场景。

（1）HAP文件：对应模块类型为Ability（module.json5中的type值为entry或feature）。HAP是应用的功能模块，可以独立安装和运行。每个应用必须包含一个entry类型的HAP，可以不包含feature类型的HAP，也可以包含一个或多个feature类型的HAP。

（2）HAR文件：对应模块类型为Static Library（module.json5中的type值为har）。HAR是静态共享包，可在编译态供HAP文件复用。HAR既支持在应用内共享，也支持在发布后供其他应用使用。

（3）HSP文件：对应的模块类型为Shared Library（module.json5中的type值为shared）。HSP是动态共享包，可在运行时被其他HAP或HSP文件复用。当多包（HAP/HSP）同时引用同一个共享包时，采用HSP替代HAR，可以避免HAR造成的多包间代码和资源的重复拷贝，从而减小应用包大小。

HAP、HAR、HSP三者支持的规格配置见表2-2，其中“√”表示支持，“×”表示不支持。

表2-2　HAP、HAR、HSP支持的规格对比

规　　格	HAP	HAR	HSP
支持在配置文件中声明UIAbility组件与ExtensionAbility组件	√	×	×
支持在配置文件中声明pages页面	√	×	√
支持包含资源文件与.so文件	√	√	√
支持依赖其他HAR文件	√	√	√
支持依赖其他HSP文件	√	√	√
支持在设备上独立安装运行	√	×	×

另外，对HAR和HSP补充说明如下：

（1）HAR虽然不支持在配置文件中声明pages页面，但可以包含pages页面，并通过命名路由的方式进行跳转。

（2）由于HSP仅支持应用内共享，如果HAR依赖了HSP，则该HAR文件仅支持应用内共享，不支持发布到二方仓或三方仓供其他应用使用，否则会导致编译失败。

（3）HAR和HSP均不支持循环依赖，也不支持依赖传递。

2.4　小结

本章主要介绍了鸿蒙App开发必须掌握的基础知识，包括鸿蒙App的开发特点（App的运行环境、开发语言、对数据库的访问）、鸿蒙App的项目结构（目录结构、编译配置文件build-profile.json5、运行配置文件module.json5）、鸿蒙App的调试和打包（连接真机调试、给App添加开发签名、App的编译态和发布态）。

通过本章的学习，读者应该了解鸿蒙App开发的基本概念，熟悉鸿蒙App项目的组织形式，熟练使用DevEco Studio完成开发调试，并将测试App安装到真机上。

2.5　动手练习

请在实验环境中创建一个测试用的鸿蒙App项目，并进行真机调试，主要步骤如下：

（1）创建一个新的App项目。

（2）开启真机的开发者模式。

（3）使用手机的数据线将真机连接到计算机的USB插槽。

（4）使用DevEco Studio将测试App安装到真机上。