# 移动应用开发（三）

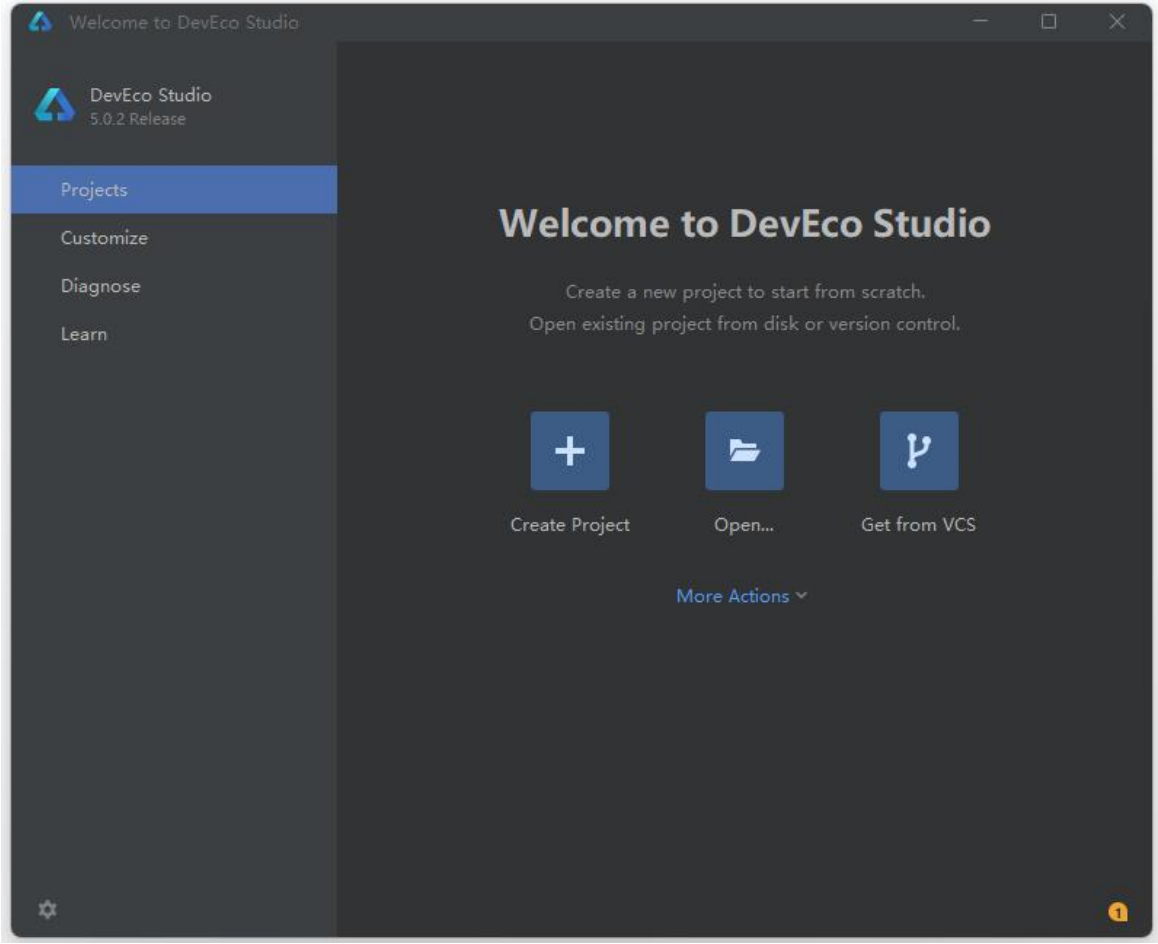
张凯斌

福建农林大学
资源与环境学院

2025.3.5

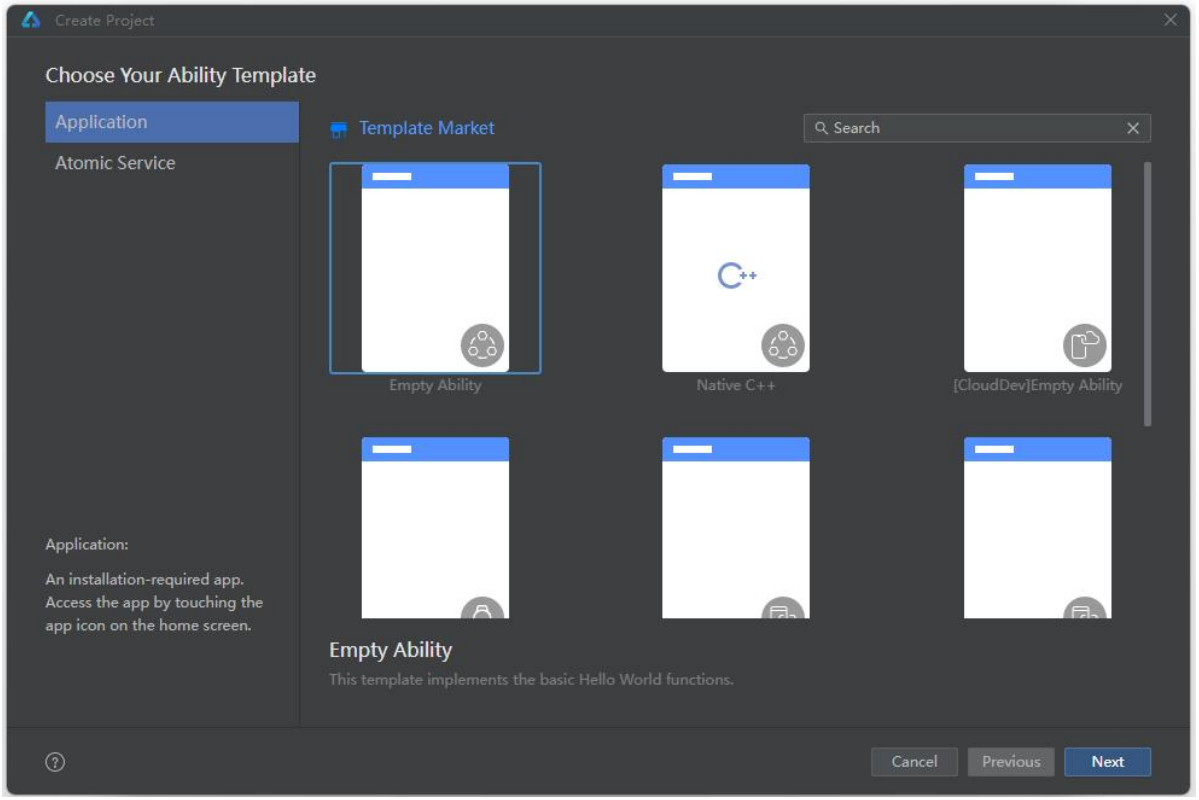
# 启动 DevEco Studio（1/17）

- 新建工程



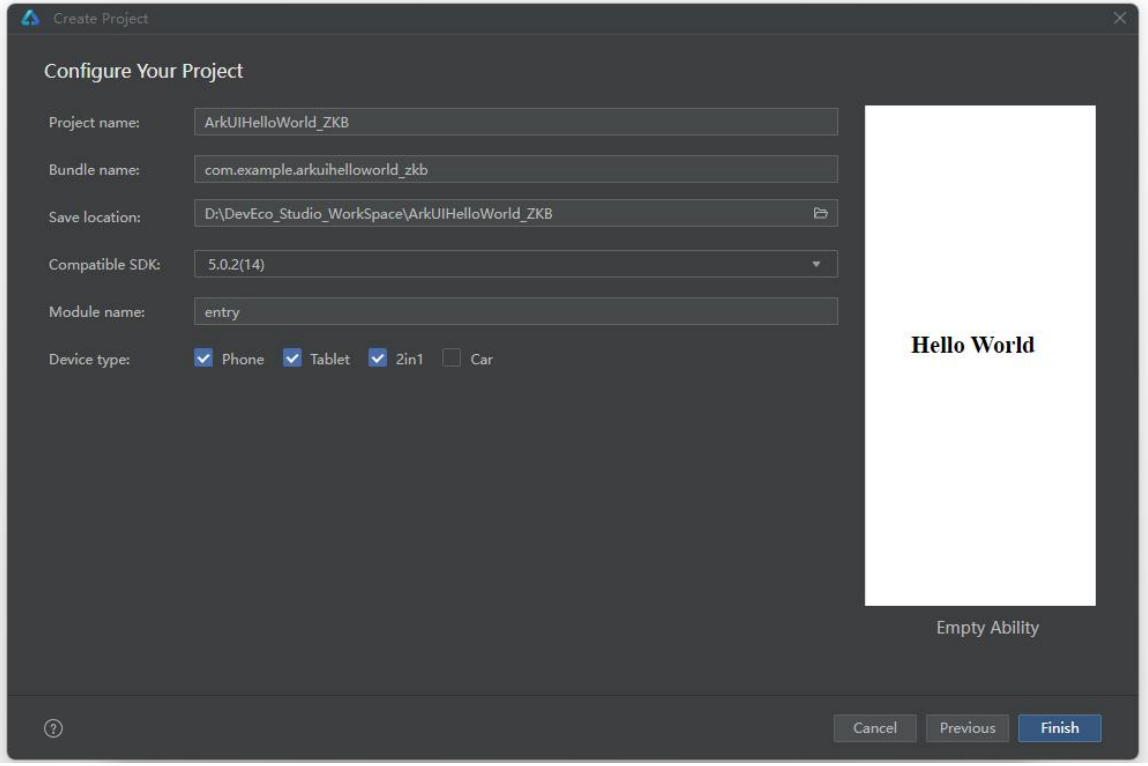
# 启动 DevEco Studio（2/17）

- 新建工程（contd.）
  - Empty Ability



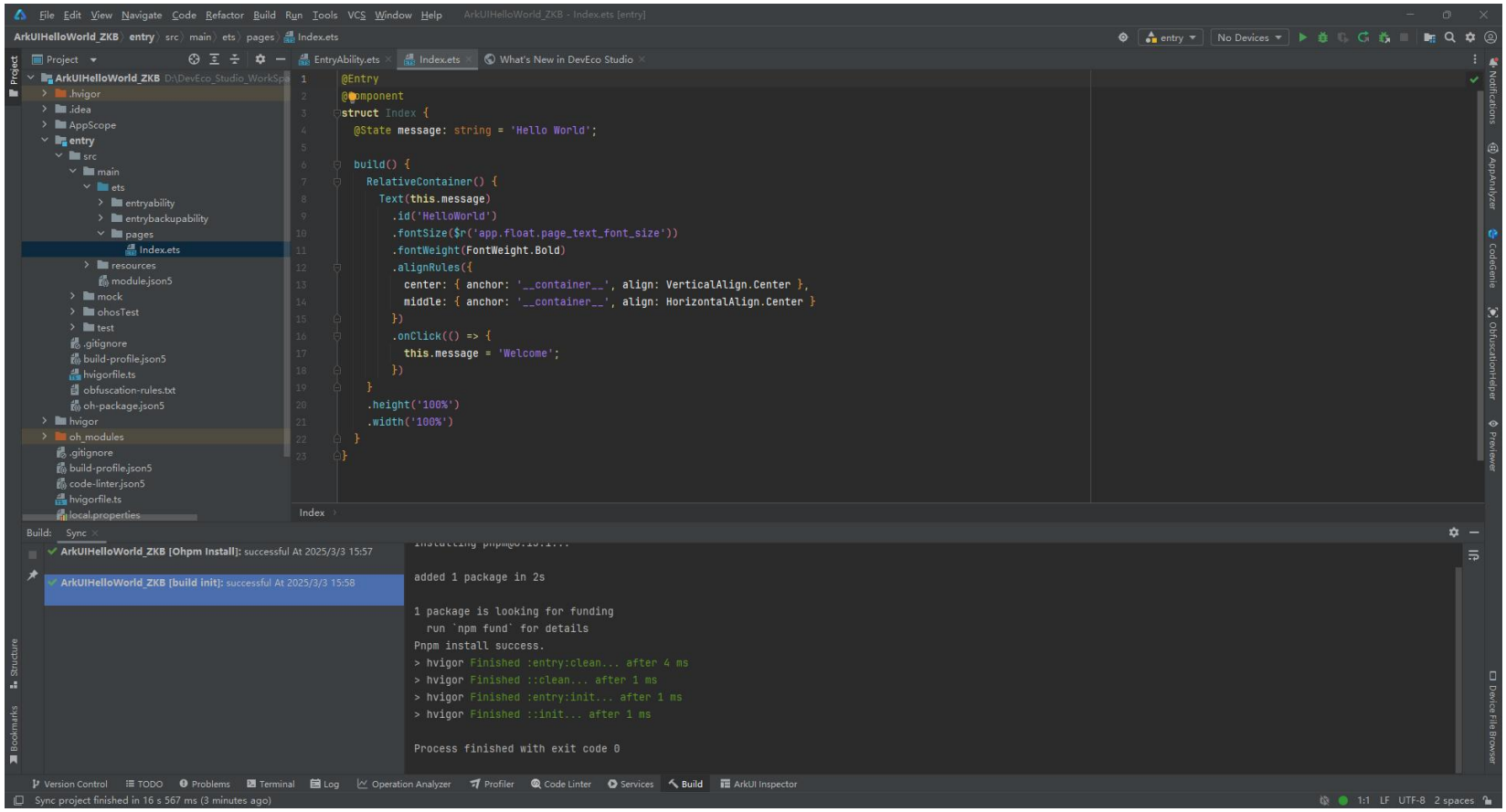
# 启动 DevEco Studio（3/17）

- 新建工程（contd.）
  - 如下设置



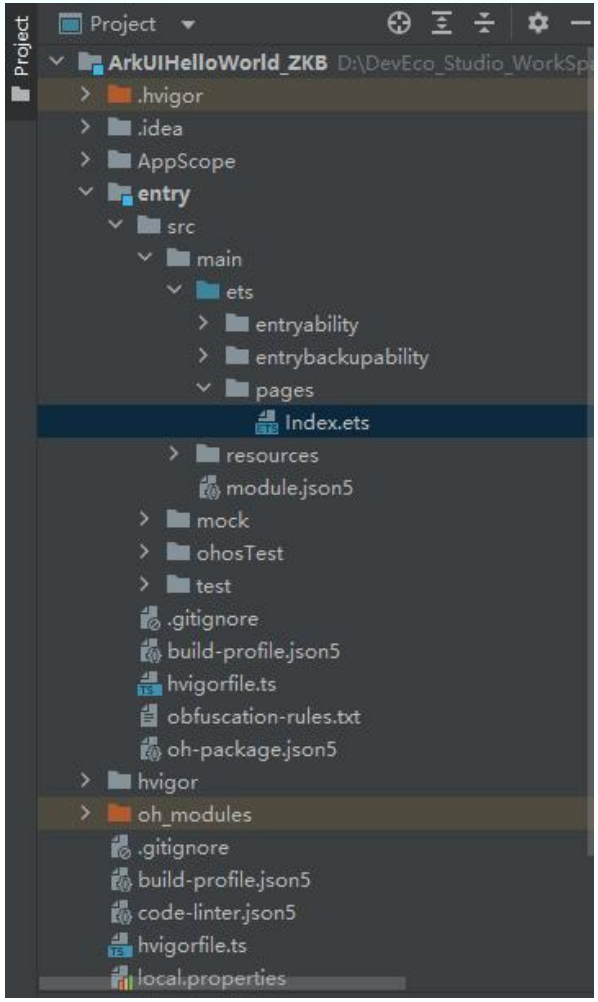
# 启动 DevEco Studio（4/17）

- 默认工程全貌



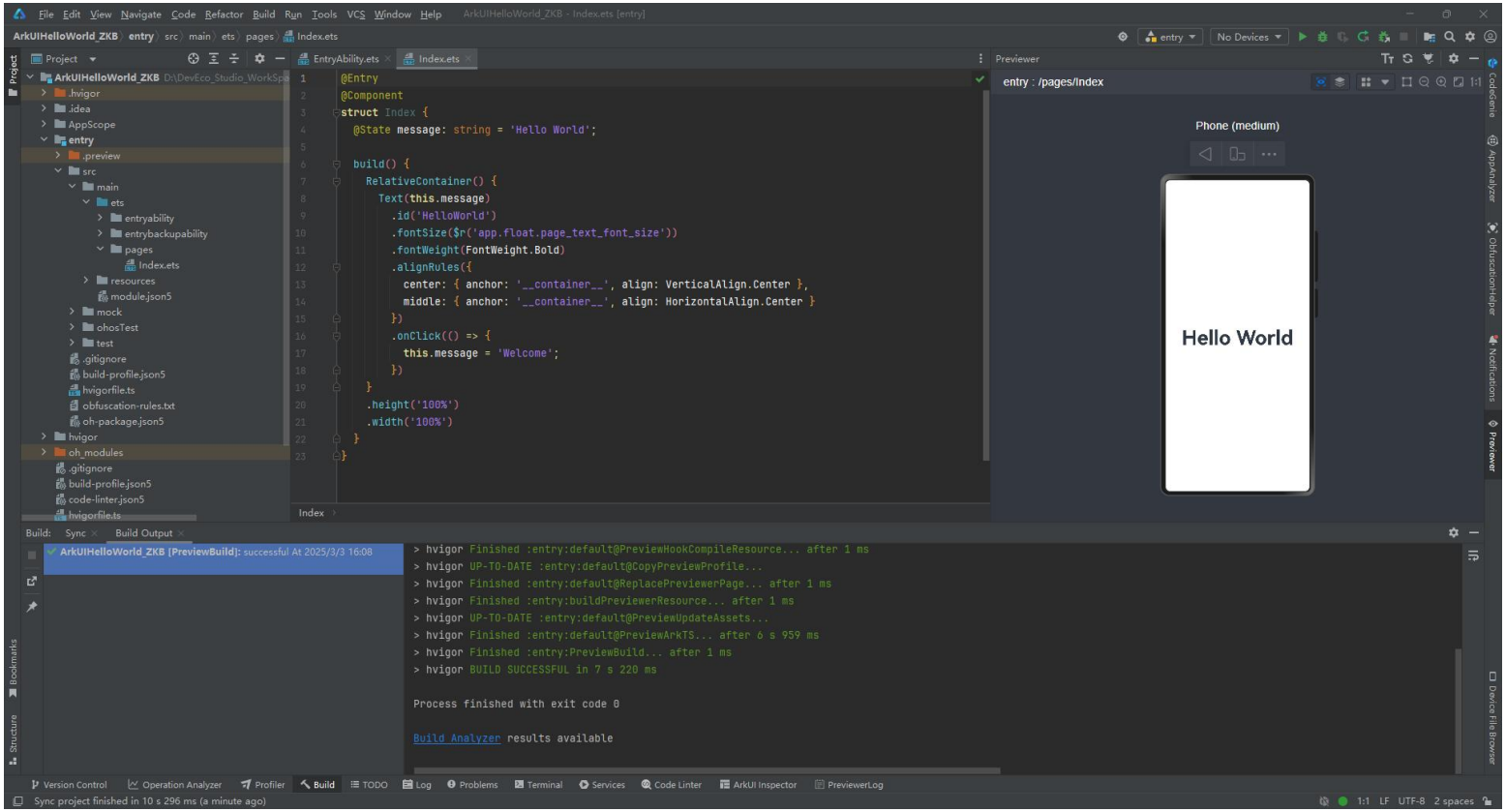
# 启动 DevEco Studio（5/17）

- 工程目录



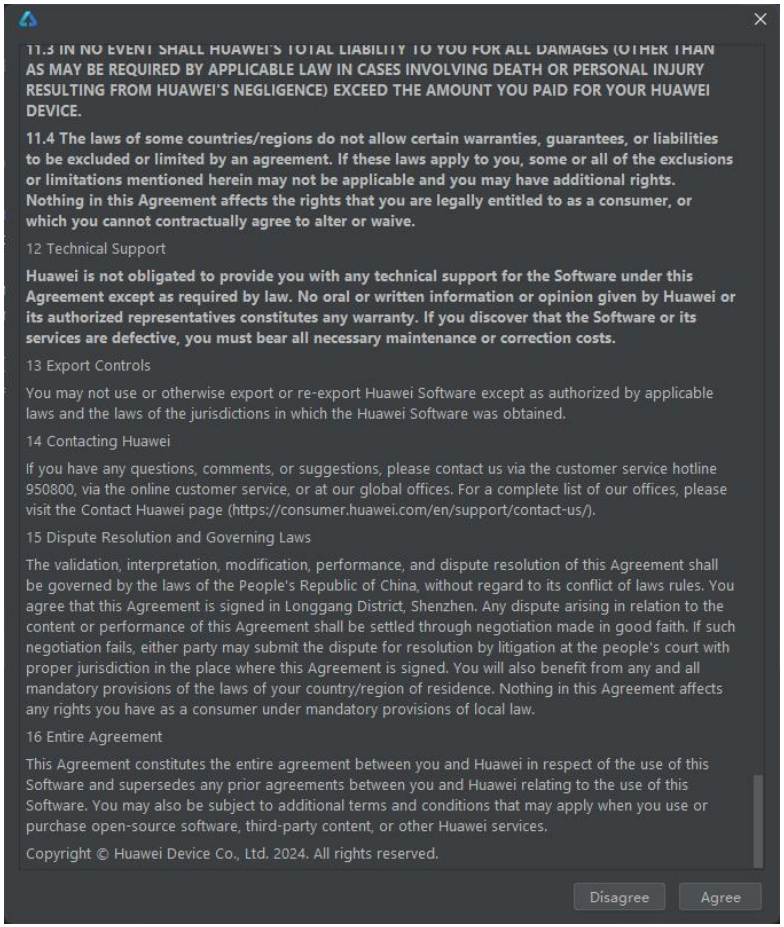
# 启动 DevEco Studio（6/17）

- 页面预览



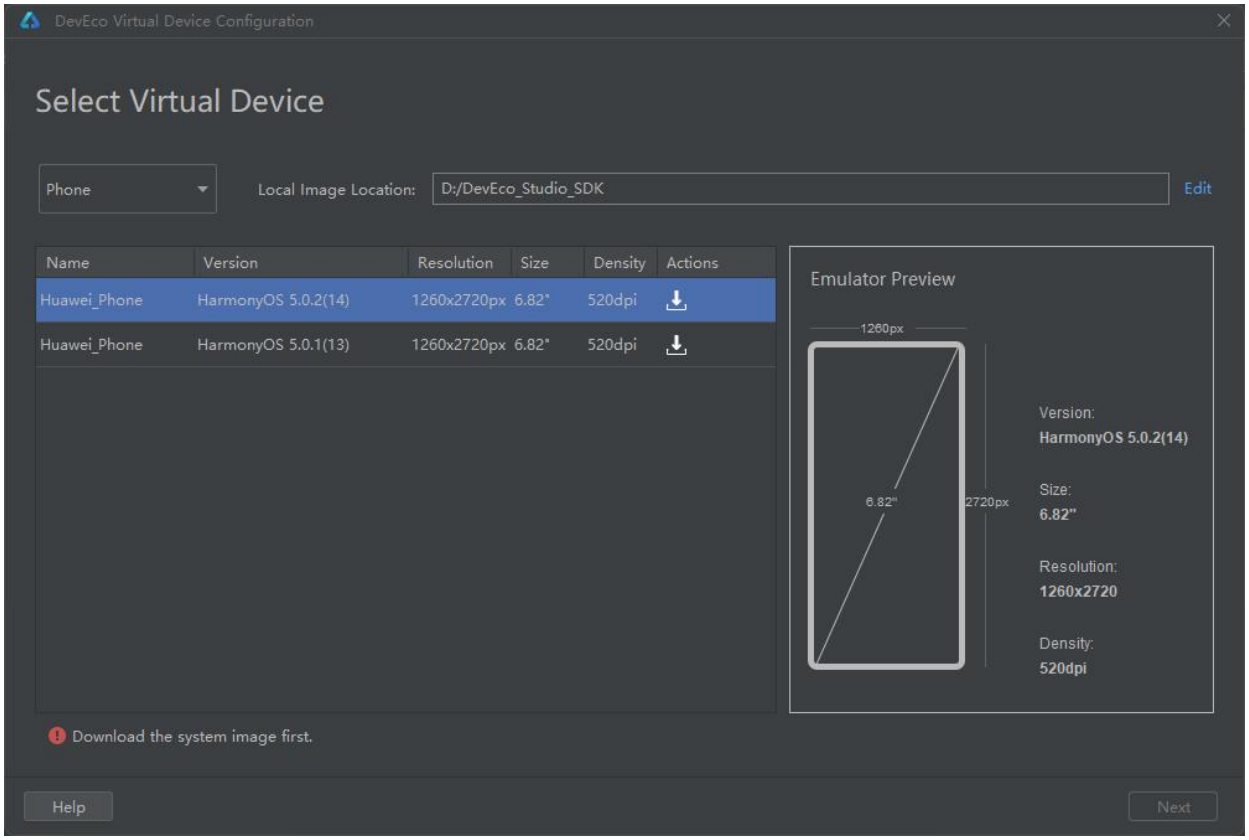
# 启动 DevEco Studio（7/17）

- 安装设备
  - 同意协议



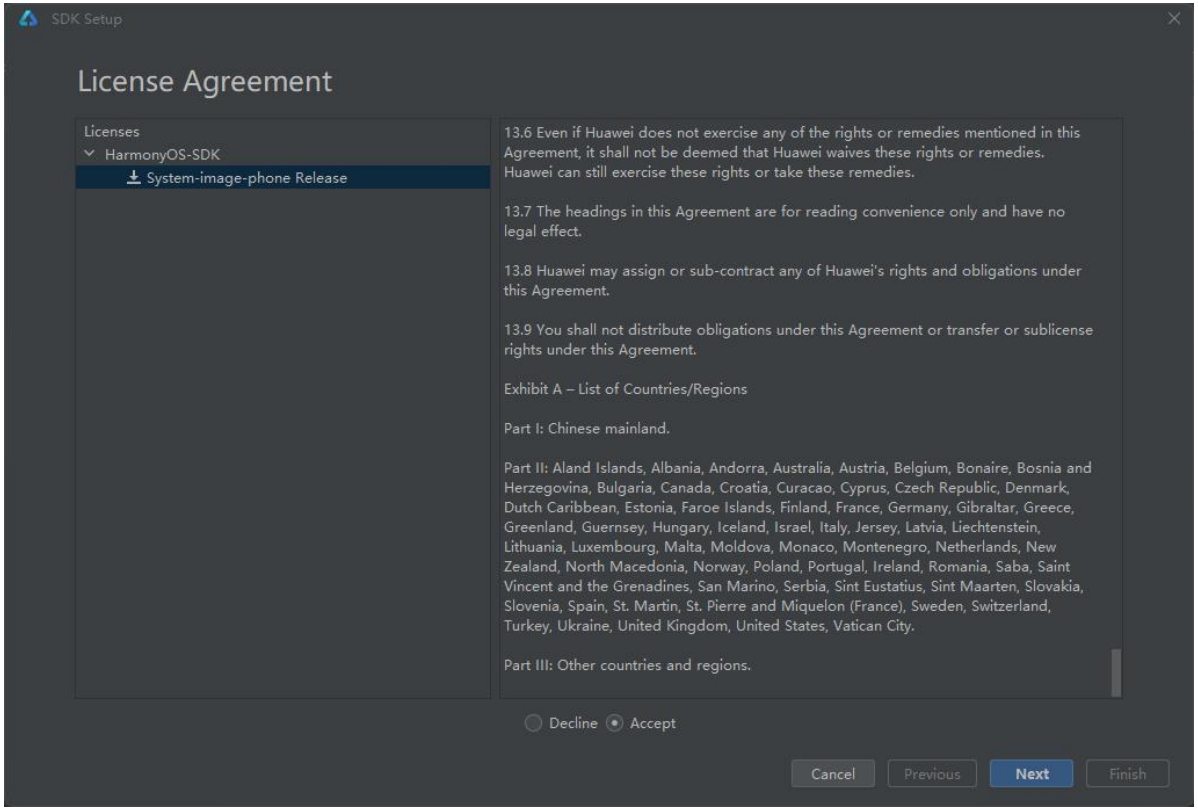
# 启动 DevEco Studio（8/17）

- 安装设备（contd.）
  - 选择设备



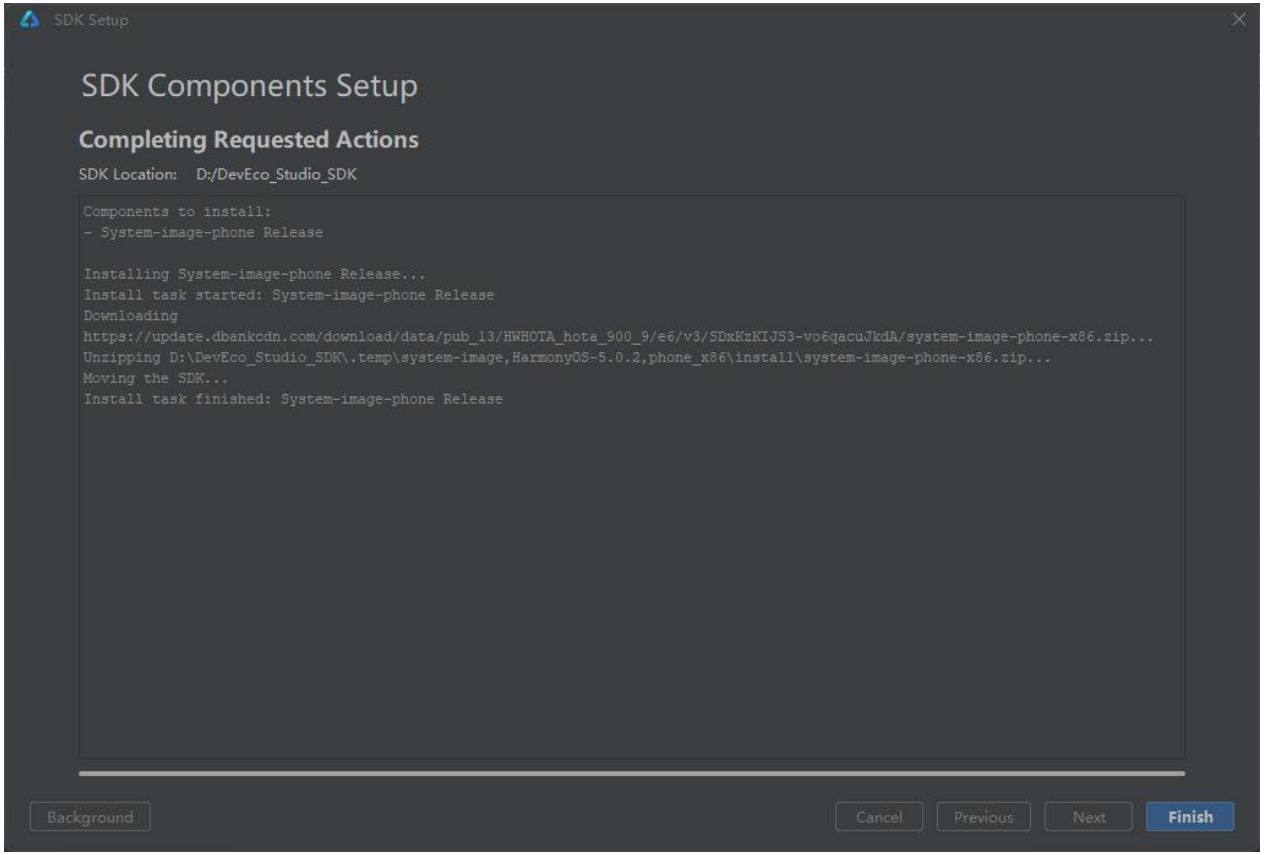
# 启动 DevEco Studio（9/17）

- 安装设备（contd.）
  - 同意并下载



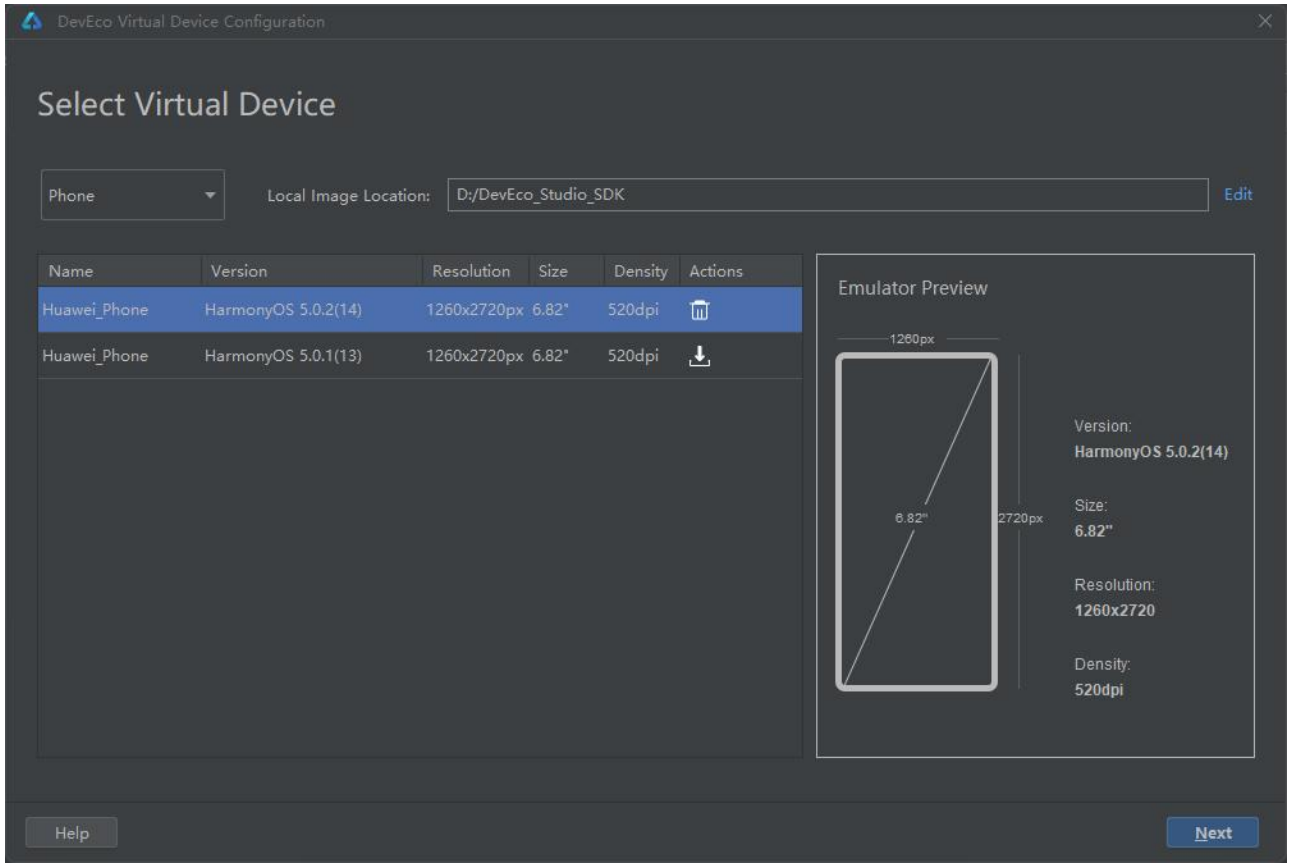
# 启动 DevEco Studio（10/17）

- 安装设备（contd.）
  - 等待下载



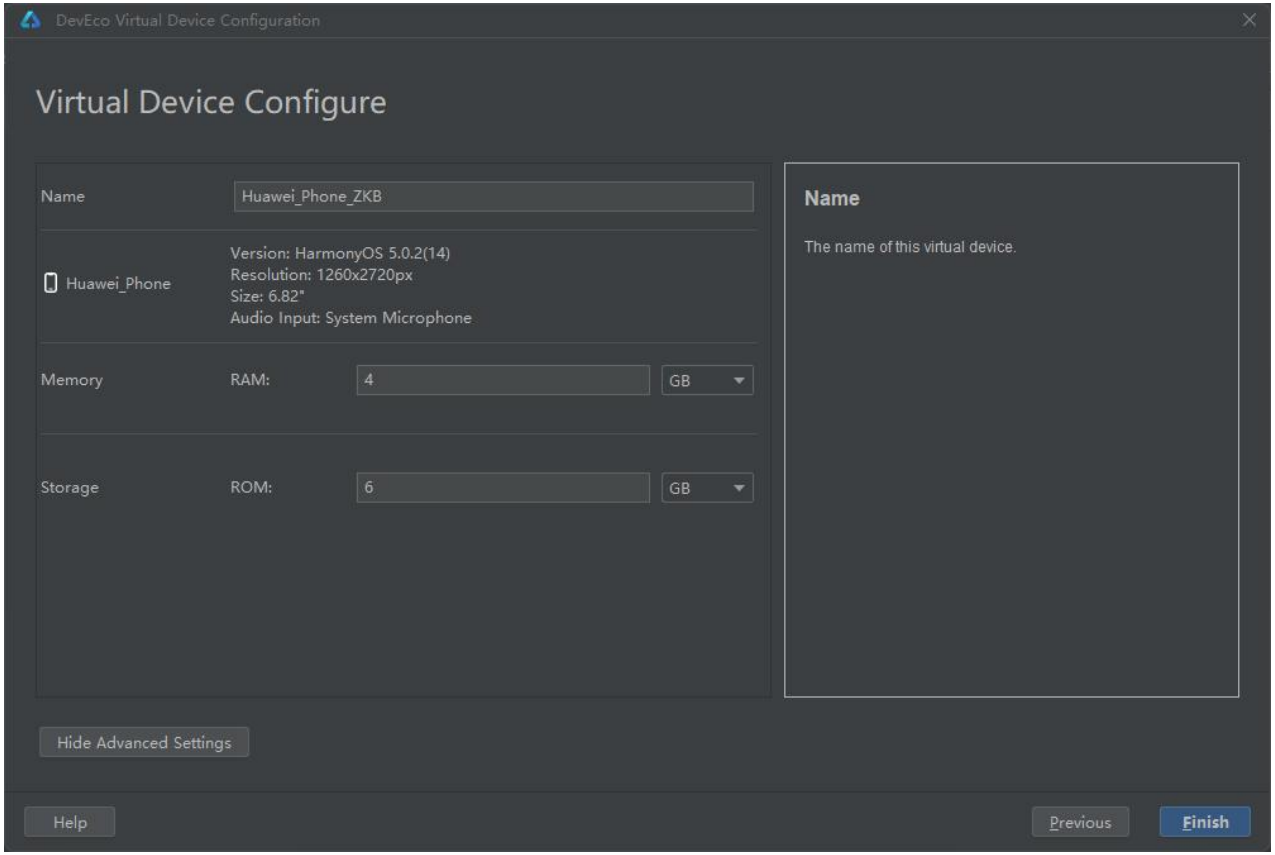
# 启动 DevEco Studio（11/17）

- 安装设备（contd.）
  - 完成下载



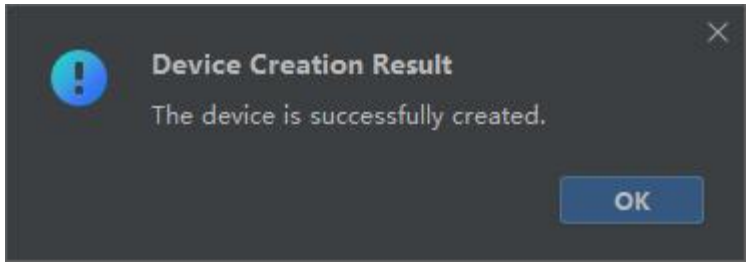
# 启动 DevEco Studio（12/17）

- 安装设备（contd.）
  - 完成安装



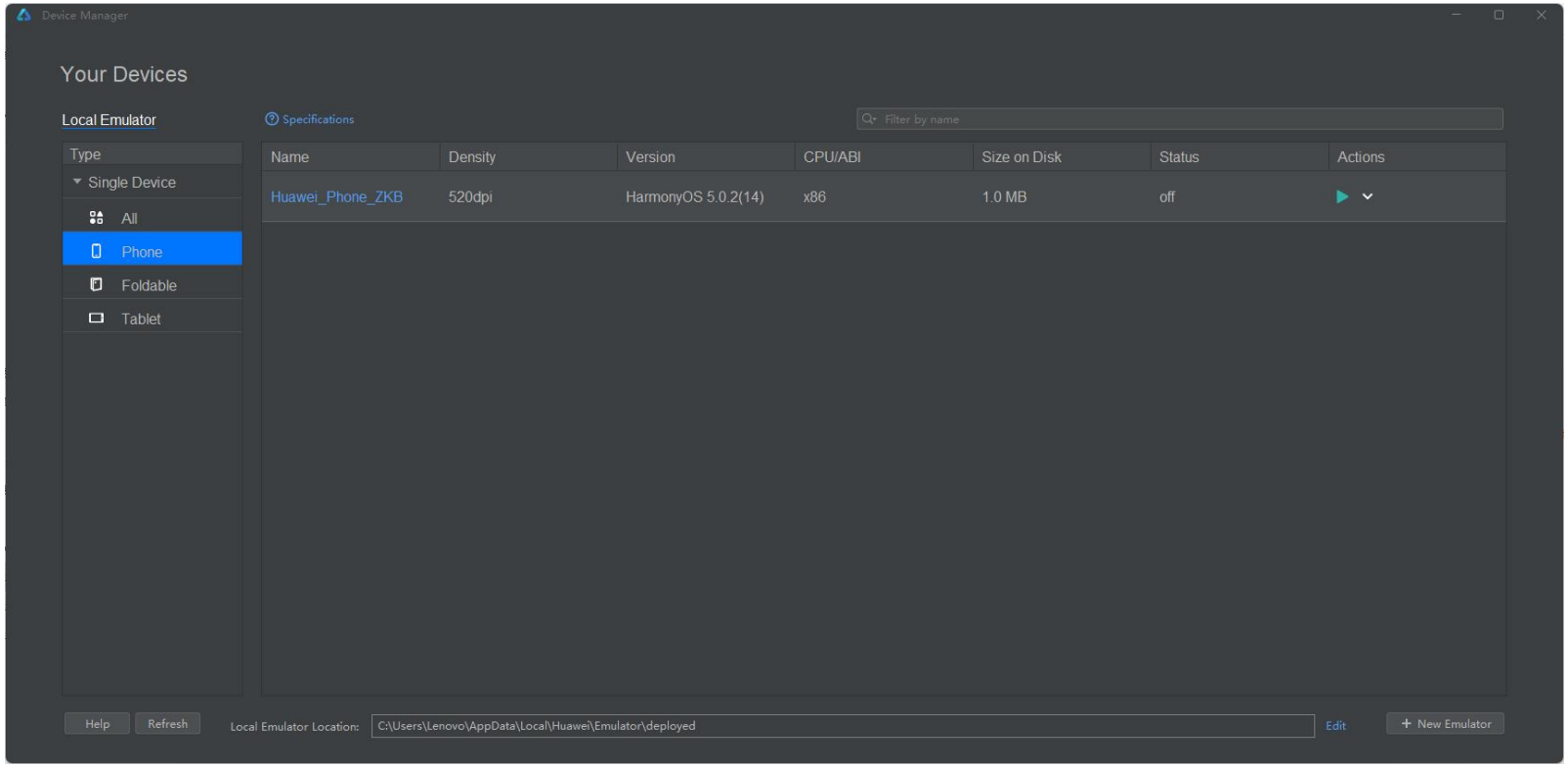
# 启动 DevEco Studio（13/17）

- 安装设备（contd.）
  - 最终提示



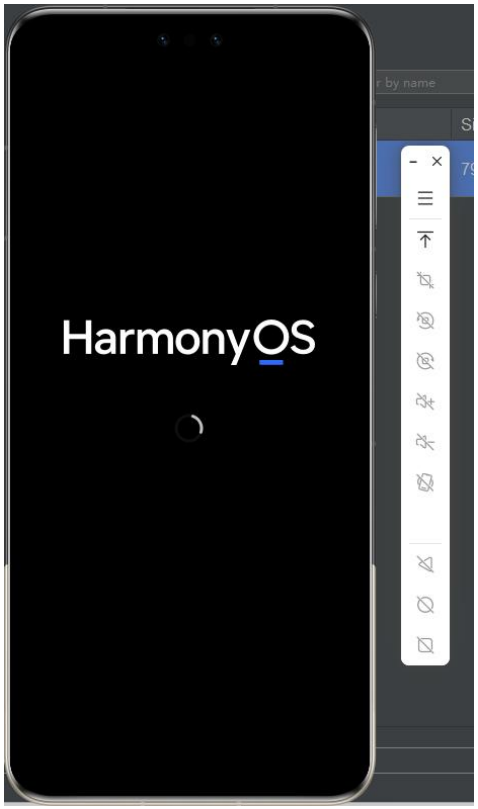
# 启动 DevEco Studio（14/17）

- 运行设备
  - 点击绿色三角形按钮运行



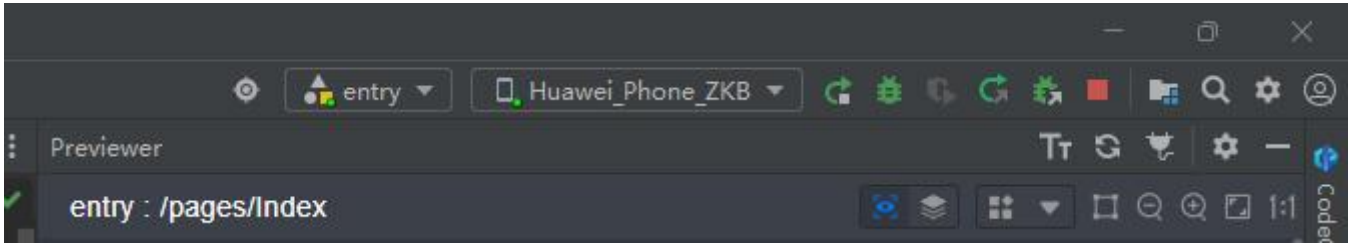
# 启动 DevEco Studio（15/17）

- 运行设备（contd.）
  - 开机界面




# 启动 DevEco Studio（16/17）

- 运行设备（contd.）
  - 右上角点击运行



# 启动 DevEco Studio（17/17）

- 运行设备（contd.）
  - 结果界面

# 应用工程结构（1/6）

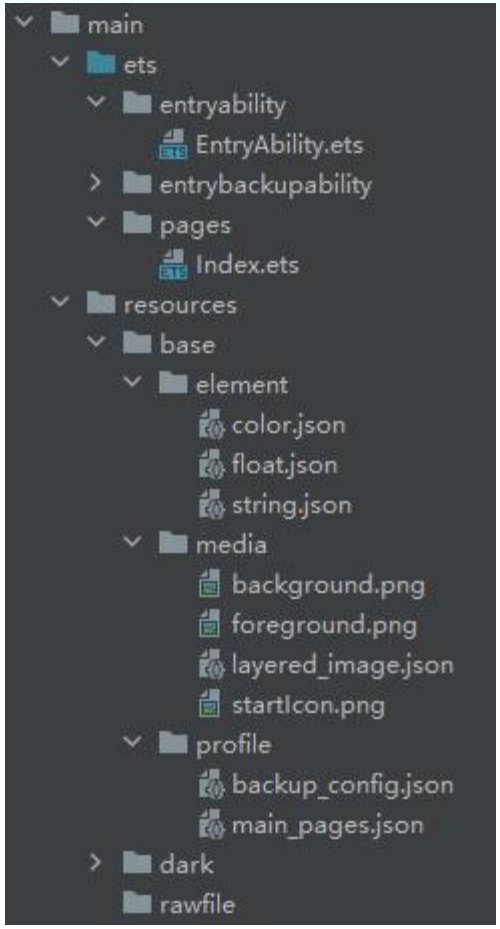
- 工程级目录

# 应用工程结构（2/6）

- 工程级目录（contd.）

# 应用工程结构（3/6）

- 工程级目录（contd.）



# 应用工程结构（4/6）

- 工程级目录（contd.）

```
{
  "app": {
    "bundleName": "com.example.arkuihelloworld_zkb",
    "vendor": "example",
    "versionCode": 1000000,
    "versionName": "1.0.0",
    "icon": "$media:app_icon",
    "label": ArkUIHelloWorld_ZKB
  }
}
```

# 应用工程结构（5/6）

- 工程级目录（contd.）

```
"module": {
  "name": "entry",
  "type": "entry",
  "description": module description,
  "mainElement": "EntryAbility",
  "deviceTypes": [
    "phone",
    "tablet",
    "2in1"
  ],
  "deliveryWithInstall": true,
  "installationFree": false,
  "pages": "$profile:main_pages",
```

```
"abilities": [
  {
    "name": "EntryAbility",
    "srcEntry": "./ets/entryability/EntryAbility.ets",
    "description": description,
    "icon": "$media:layered_image",
    "label": label,
    "startWindowIcon": "$media:startIcon",
    "startWindowBackground": #FFFFFF,
    "exported": true,
    "skills": [
      {
        "entities": [
          "entity.system.home"
        ],
        "actions": [
          "action.system.home"
        ]
      }
    ]
  }
],
```

```
"extensionAbilities": [
  {
    "name": "EntryBackupAbility",
    "srcEntry": "./ets/entrybackupability/EntryBackupAbility.ets",
    "type": "backup",
    "exported": false,
    "metadata": [
      {
        "name": "ohos.extension.backup",
        "resource": "$profile:backup_config"
      }
    ],
  }
]
```

# 应用工程结构（6/6）

- 工程级目录（contd.）

```
"src": [
  "pages/Index"
]
```

# Ability开发（1/8）

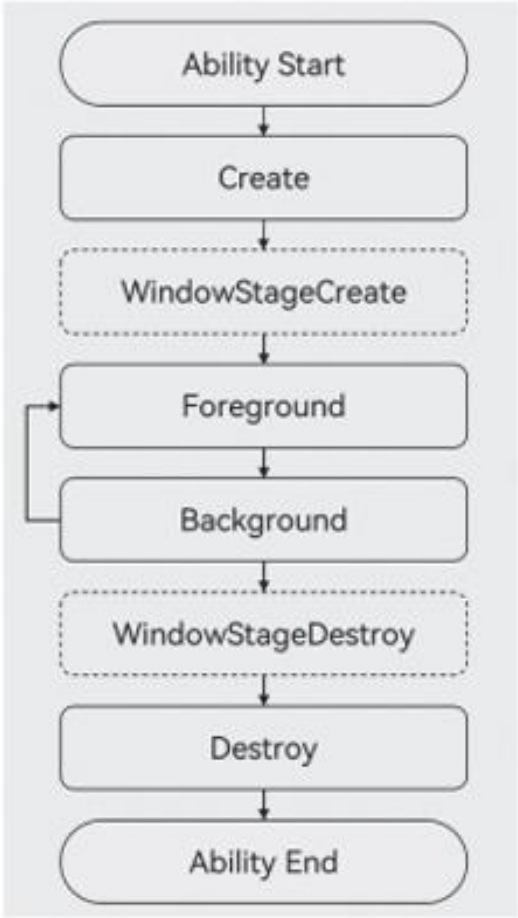
- Ability是应用/元服务所具备的能力的抽象
  - 一个应用可以具备多种能力，也就是说可以包含多个Ability
    - HarmonyOS支持应用以Ability为单位进行部署

# Ability开发（2/8）

- 应用/元服务先后提供了两种应用模型
  - FA（Feature Ability）模型
    - API 7开始支持的模型，已经不再主推
  - Stage模型
    - HarmonyOS 3.1 Developer Preview版本开始新增
    - 目前主推且会长期演进的模型
    - 只支持使用ArkTS语言进行开发

# Ability开发（3/8）

- Stage模型的Ability生命周期
  - 四个状态
    - Create
    - Foreground
    - Background
    - Destroy



# Ability开发（4/8）

- Stage模型的Ability生命周期（contd.）
  - WindowStageCreate和WindowStageDestroy为窗口管理器（WindowStage）
    - 在Ability中管理UI界面功能的两个生命周期回调，从而实现Ability与窗口之间的弱耦合

# Ability开发（5/8）

- Ability启动模式
  - 实例在启动时的不同呈现状态
  - 针对不同的业务场景，系统提供了3种启动模式
    - singleton（单实例模式）
    - standard（标准实例模式）
    - specified（指定实例模式）

# Ability开发（6/8）

- Ability启动模式（contd.）
  - singleton
    - 系统中只存在唯一一个该Ability实例
    - 当再次调用*startAbility()*方法启动该Ability实例时
      - 只会进入该Ability的*onNewWant()*回调，不会进入其*onCreate()*和*onWindowStageCreate()*生命周期回调
  - 如果需要使用singleton启动模式
    - 将module.json5配置文件中的"abilities"字段中的"launchType"字段配置为"singleton"即可

# Ability开发（7/8）

- Ability启动模式（contd.）
  - standard
    - 每次调用*startAbility()*方法时，都会在应用进程中创建一个新的该类型的Ability实例
      - 即在最近任务列表中可以看到有多个该类型的Ability实例
  - 如果需要使用standard启动模式
    - 将module.json5配置文件中的"abilities"字段中的"launchType"字段配置为"standard"即可

# Ability开发（8/8）

- Ability启动模式（contd.）
  - specified
    - 在Ability实例创建之前，允许开发者为该实例创建一个唯一的字符串Key
    - 创建的Ability实例绑定Key之后
      - 后续每次调用*startAbility()*方法时，都会询问应用使用哪个Key对应的Ability实例来响应startAbility请求
  - 如果需要使用specified启动模式
    - 将module.json5配置文件中的"abilities"字段中的"launchType"字段配置为"specified"即可

# 实例（1/10）

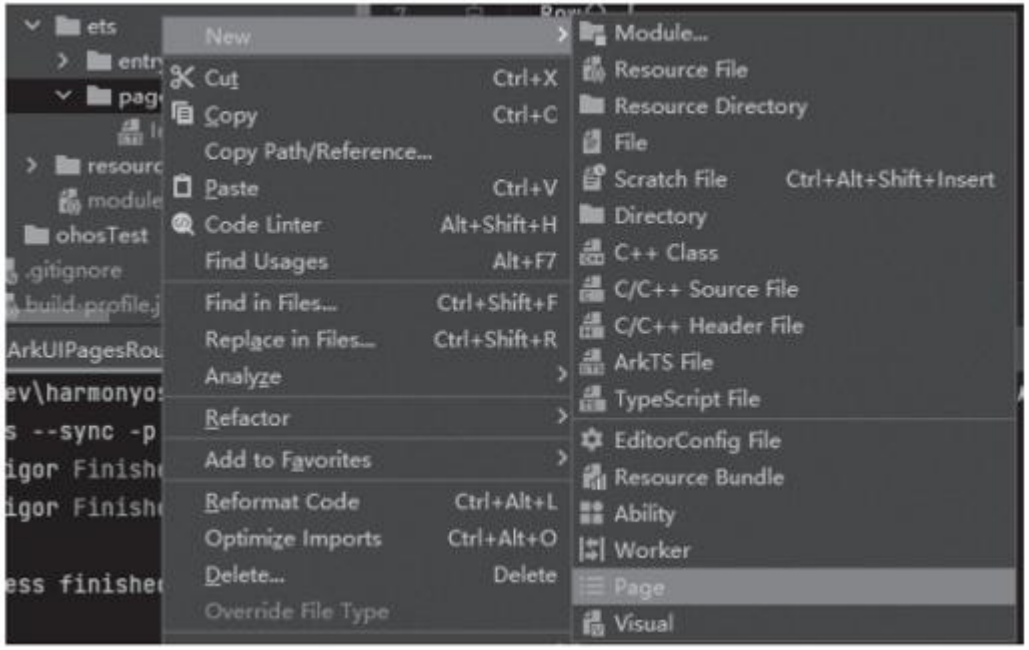
- Ability内页面的跳转和数据传递
  - Ability的数据传递包括
    - Ability内页面的跳转和数据传递
    - Ability间的数据跳转和数据传递
  - 本次展示Ability内页面的跳转和数据传递

# 实例（2/10）

- 新建Ability内页面
  - 初始化工程之后，会生成以下内容：
    - 在src/main/ets/entryability目录下，初始会生成一个Ability文件EntryAbility.ets
    - 可以在EntryAbility.ets文件中根据业务需要实现Ability的生命周期回调内容
  - 在src/main/ets/pages目录下，会生成一个Index页面
    - 这也是基于Ability实现的应用的入口页面
    - 可以在Index页面中根据业务需要实现入口页面的功能

# 实例（3/10）

- 新建Ability内页面（contd.）
  - 为了实现页面的跳转和数据传递，需要新建一个页面
    - 在src/main/ets/pages目录下，可以通过右击New→Page来新建页面



# 实例（4/10）

- 新建Ability内页面（contd.）
  - 在原有Index页面的基础上，新建一个名为Second的页面
  - Second页面创建完成之后，会自动做两个动作
    - 在src/main/ets/pages目录下创建一个Second.ets文件
    - 将Second页面信息配置到
      - src/main/resources/base/profile/main_pages.json文件

# 实例（5/10）

- 页面跳转及传参
  - 在使用页面路由之前，需要先导入router模块
  - 页面跳转有以下几种方式，根据需要选择一种方式跳转即可
    - *router.pushUrl()*
    - *router.replaceUrl()*

# 实例（6/10）

- 页面跳转及传参（contd.）
  - *router.pushUrl()*
    - 跳转到Ability内的指定页面
      - 每调用一次*router.pushUrl()*方法，均会在栈中新建一个页面
    - 当页面栈数量较大或者超过限制时，可以通过调用*router.clear()*方法清除页面栈中的所有历史页面

# 实例（7/10）

- 页面跳转及传参（contd.）
  - *router.pushUrl()*加mode参数
    - 在单实例模式下，如果目标页面在页面栈中已经存在同URL的页面，离栈顶最近的同URL的页面会被移动到栈顶
    - 如果目标页面在页面栈中不存在同URL的页面，那么按照标准模式跳转

# 实例（8/10）

- 页面跳转及传参（contd.）
  - *router.replaceUrl()*
    - 跳转到Ability内的指定页面
    - 使用新的页面替换当前页面，并销毁被替换的当前页面
    - 加mode参数的语义如同*router.pushUrl()*

# 实例（9/10）

- Want概述
  - 在Stage模型中，Want是对象间信息传递的载体，可以用于应用组件间的信息传递
  - Want的使用场景之一是作为startAbility的参数，其包含指定的启动目标，以及启动时需携带的相关数据
    - 如bundleName和AbilityName字段分别指明目标Ability所在应用的包名以及对应包内的Ability名称
    - 当AbilityA启动AbilityB并需要传入一些数据给AbilityB时，Want可以作为一个数据载体将数据传给AbilityB

# 实例（10/10）

- Want的类型
  - 显式
    - 在启动Ability时指定了**abilityName**和**bundleName**的Want
      - 当有明确处理请求的对象时，通过提供目标Ability所在应用的包名信息（bundleName）
      - 并在Want内指定abilityName便可启动目标Ability
      - 显式Want通常在启动当前应用开发中某个已知Ability时被用到
  - 隐式
    - 在启动Ability时未指定abilityName的Want称为隐式Want

# Q&A

# Good Luck!