

实验十八 Stage 模型

一、实验目的

1. 了解 DevEco Studio 的使用
2. 学习 ArkTS 语言及 Stage 模型
3. 编写代码
4. 编译运行
5. 在模拟器上运行

二、实验原理

1. 鸿蒙开发原理
2. ArkTS, ArkUI 开发原理
3. 鸿蒙应用运行原理

三、实验仪器材料

1. 计算机实训室电脑一台
2. DevEco Studio 开发环境及鸿蒙手机模拟器

四、理论讲解

Ability

了解应用模型之前，需要知道鸿蒙中 Ability 的概念。Ability 是鸿蒙应用的基本组成单元，是系统调度应用的最小单元。你可以把它理解为“一个具备特定功能的页面或服务”。它代表了一个应用所能完成的某一个特定的、完整的功能。一个应用通常由多个 Ability 组成，它们之间可以互相调用，协同工作，共同构成一个完整的应用。用简略文字描述 Stage 模型 中的各种 Ability：

- **UIAbility:** 带界面的能力，是应用和用户交互的入口。每个 UIAbility 实例对应一个独立的任务。
- **ExtensionAbility:** 无界面的扩展能力基类，用于特定场景，包含多种子类：
 - **ServiceExtensionAbility:** 用于在后台执行长时间运行的任务，如音乐播放。
 - **DataShareExtensionAbility:** 用于跨应用共享数据，如相册应用对外提供图片。
 - **FormExtensionAbility:** 用于创建服务卡片，在桌面上显示关键信息或快捷操作。
 - 其他还有如输入法、无障碍服务等特定扩展。

简单来说，UIAbility 管“看得到”的界面任务，ExtensionAbility 家族管“看不到”的后台服务和系统集成。

应用模型

应用模型是系统为开发者提供的应用程序所需能力的抽象提炼，它提供了应用程序必备的组件和运行机制。它相当于应用的“施工图纸”或者“脚手架”，规范化了程序的运行流程，项目结构，文件功能等等。通过了解应用模型，可以知道：

- 实现某个功能应该在哪个文件编写代码
 - 如何感知应用的状态变化，生命周期（启动、关闭、转后台运行，转前台运行等）
 - 如何调整项目配置
- 等等。

鸿蒙开发提供了两种应用模型：

- FA 模型：从 API7 开始支持，目前已经不再主推
- Stage 模型：从 API9 开始新增的模型，是目前主推而且会长期演进的模型

Stage 模型是现代鸿蒙应用的唯一推荐选择。Stage 模型的核心是“分离”：它将 UI 生命周期 (Page) 和 Ability 生命周期 (UIAbility) 分离开，带来了更好的灵活性和性能。

应用程序包概述：

<https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/application-package-overview-V5>

Stage 模型应用程序包结构：

<https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/application-package-structure-stage-V5>

Stage 模型应用配置文件：

<https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/application-configuration-file-overview-stage-V5>

关于编译构建：

<https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/ide-build-V5>

在实验之前，先过一遍这几篇文档的内容。目前为止，我们都是专注于 HAP 包。我们通过实例来理解。

五、实验步骤

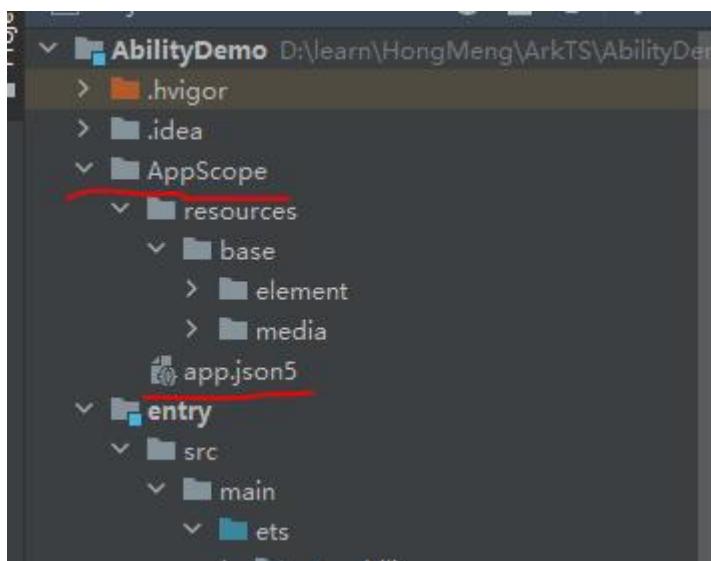
1. 打开 DevEco Studio，点击 Create Project 创建工程

设置项目名称为 AbilityDemo。

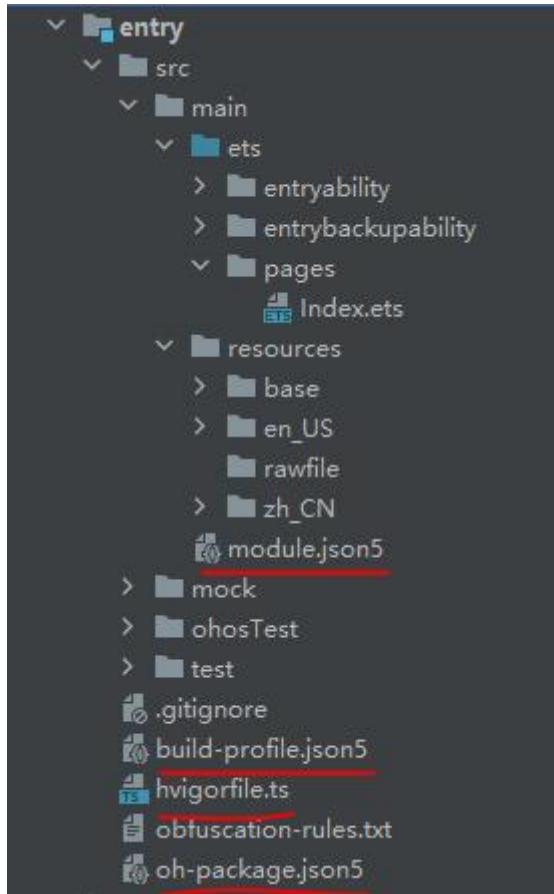
2. 目录和文件说明

我们先来说明一下目录和文件的一些要点。

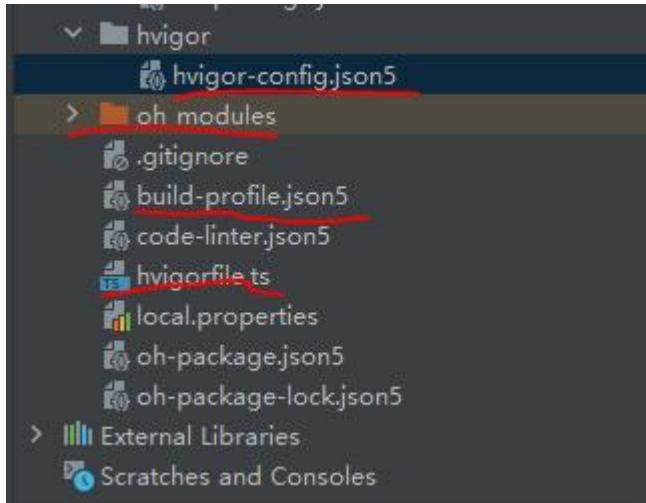
- AppScope 目录里面放置的是整个 APP 全局要使用的资源，以及 app.json5 配置文件，里面是全局配置信息。文件 app.json5 用于声明应用的全局配置信息，比如应用 Bundle 名称、应用名称、应用图标、应用版本号等。



- entry：工程模块，编译构建生成一个 HAP 包



- src > main > ets: 用于存放 ArkTS 源代码
 - src > main > ets > entryability: 应用/服务的入口
 - src > main > ets > pages: 应用/服务包含的页面
 - src > main > resources: 用于存放应用/服务要用到的资源文件
 - src > main > module.json5: 模块应用配置文件用于声明 Module 基本信息、支持的设备类型、所含的组件信息、运行所需申请的权限等
 - build-profile.json5: 工程级或 Module 级的构建配置文件，包括应用签名、产品配置等
 - hvigorfile.ts: 应用级或 Module 级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。混淆开启后，在使用 Release 模式进行编译时，会对代码进行编译、混淆及压缩处理，保护代码资产
 - oh-package.json5: 用于存放依赖库的信息，包括所依赖的三方库和共享包
-
- 工程级的目录及文件



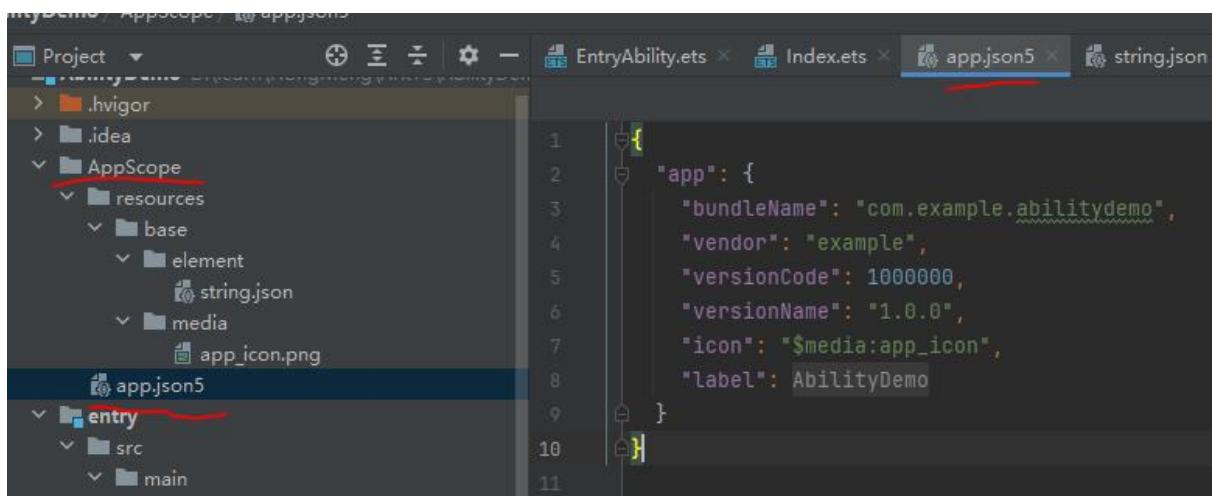
- `hvigorfile.ts`: 此文件在每个 node 下都有一份，是构建的必须文件，在此文件中可以注册插件、任务以及生命周期 hook 等操作
- `hvigorconfig.ts`: 此文件在整个项目中只有根目录下存在一份，不是构建必须的文件并且默认不存在，如有需要可自行创建，此文件被解析执行的时间较早，可用于在 hvigor 生命周期刚开始时操作某些数据
- `hvigor` 目录下提供 `hvigor-config.json5` 文件，以指定 `hvigor` 的版本、构建依赖以及构建行为的配置参数
- `build-profile.json5` 文件分为工程级与模块级，其中 `buildOption` 在工程级文件和模块级文件均可配置，其中相同字段以模块级的字段为准，不同字段模块级的 `buildOption` 配置会继承工程级配置
- `oh_modules` 目录用于存放三方库依赖信息

这些信息一开始比较难记，而且到目前为止，我们用到的，接触到的主要还是红色字体的那几项。后续我们通过一些实例来尝试使用其中的一些文件。

3. 配置应用的图标和标签

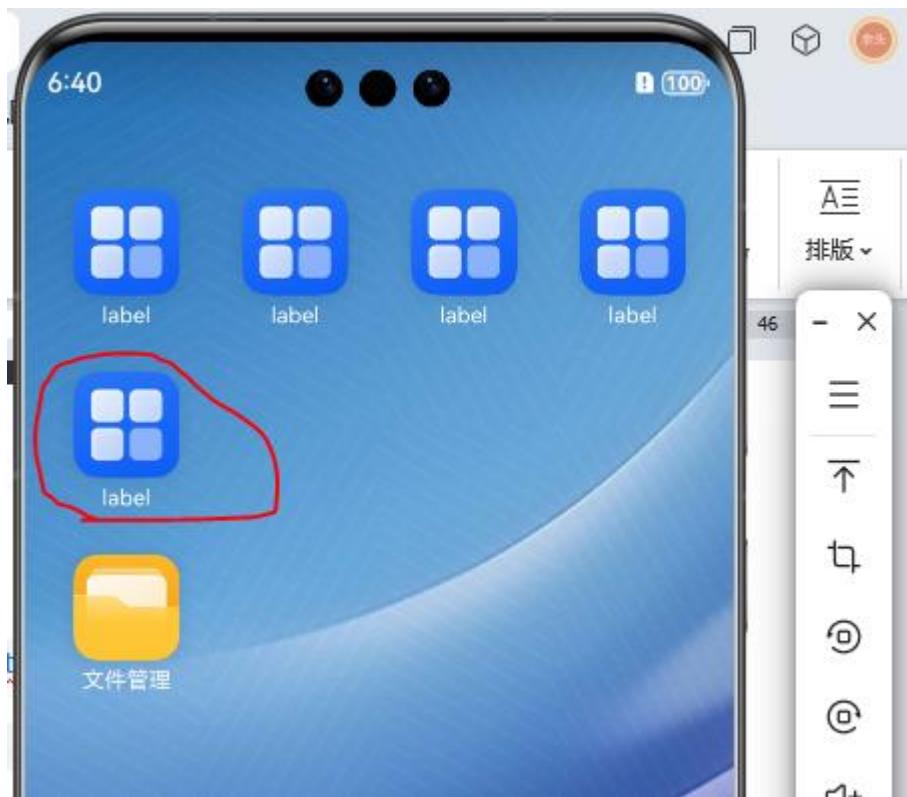
每一个 APP 都需要配置使用自己的图标，比如微信，支付宝这些应用的图标已经深入人心。如何配置自己的 APP 的图标和名称（标签）呢？

首先，打开目录 AppScope 下面的 `app.json5` 文件：



```
{
    "app": {
        "bundleName": "com.example.abilitydemo", //包名 不可省略
        "vendor": "example", // 开发者名称，不可省略
        "versionCode": 1000000, // 内部版本号，通常数字越大版本越高
        "versionName": "1.0.0", // 给用户看的版本号
        "icon": "$media:layered_image", // 应用图标
        "label": "$string:app_name" // 应用名称(标签)
    }
}
```

我们在模拟器中运行，出现 Hello World 页面之后，按住鼠标，从屏幕下方往上推，应用会缩到屏幕上一个图标那里：

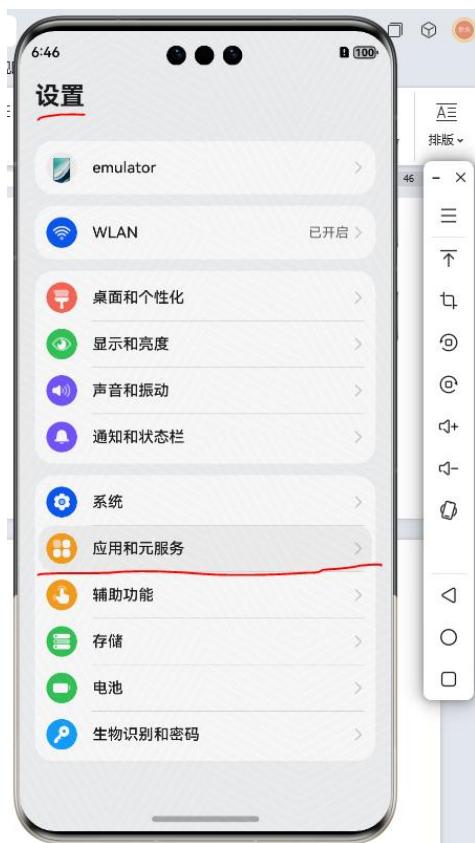


我这个图上有很多个 label，是因为前面运行过好几个实例，我们都没有设置过这些应用的图标和标签，所以都是一样的。当前这个，一般就是排在最后的那个。双击点开，就又可以看到 Hello World 页面。

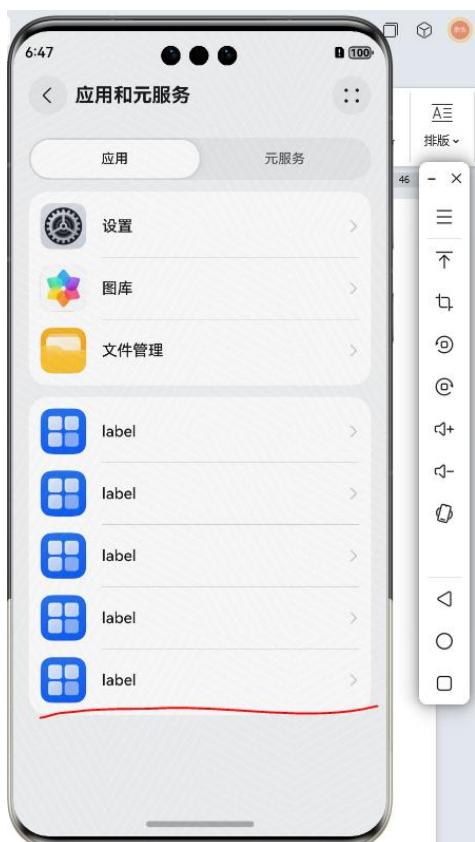
回到应用缩略窗口，滑动到这个页面（鼠标向右滑动）：



点开“设置”：



然后进入“应用和元服务”：



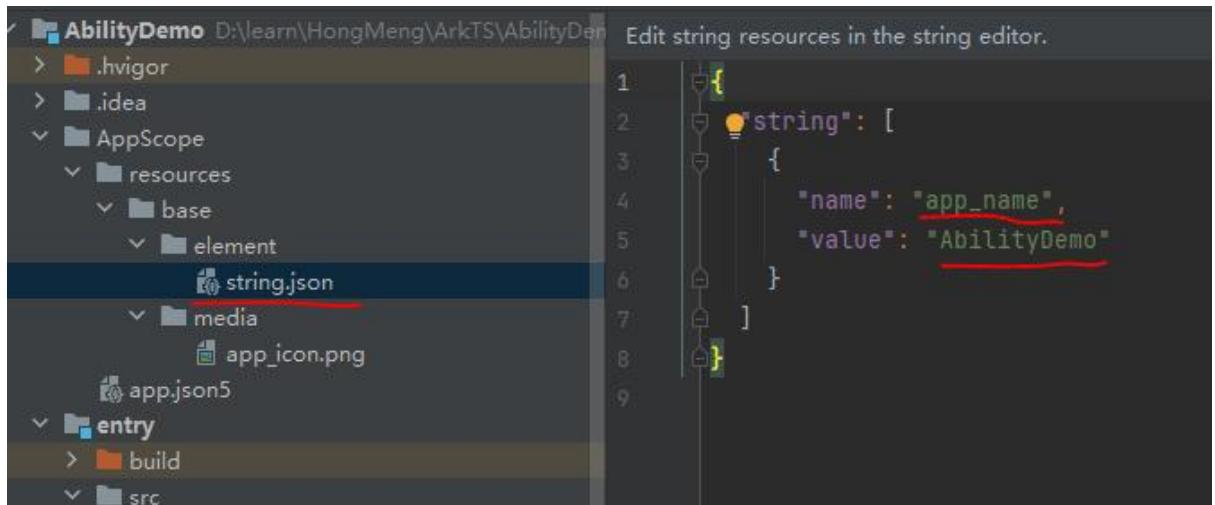
可以看到，屏幕上的应用在这里都能找到对应的。点开最后一个：



这里就有疑问了，在我们的 app.json5 中，“label”对应的是：

```
"label": "$string:app_name" // 应用名称 ( 标签 )
```

而在 AppScope > resources > base > element 下面的 string.json 文件中：



对应的值应该是“AbilityDemo”才对啊？

查看文档：

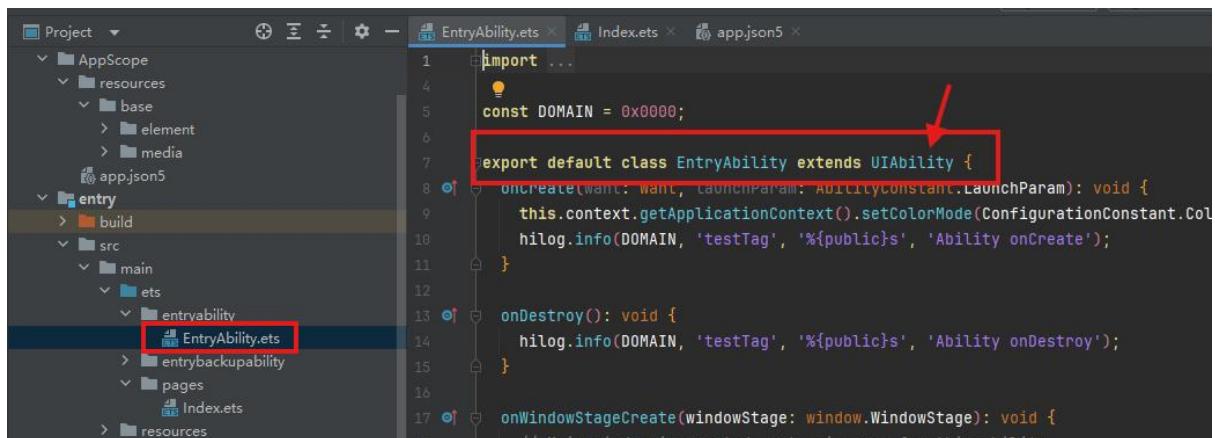
<https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/application-component-configuration-stage-V5>

可以看到，对于 Stage 模型，有两种场景：

生成机制

- HAP中包含UIAbility
 - 如果在module.json5配置文件的abilities标签中配置了icon和label，且该对应的ability中skills标签下面的entities中包含"entity.system.home"、actions中包含"ohos.want.action.home"或者"action.system.home"，则系统将优先返回module.json5中的icon与label。如果存在多个满足条件的ability，优先返回module.json5中mainElement对应的ability配置的icon和label。
 - 如果在module.json5配置文件的abilities标签中未设置icon和label，或者对应的skills标签下面的entities与actions没有做相关配置(entities中包含"entity.system.home"、actions中包含"ohos.want.action.home")，那么系统将使用app.json5中的icon和label。
- HAP中不包含UIAbility，系统将返回app.json5中的icon和label。

我们的 HAP 中是有 UIAbility 的，通过查看 EntryAbility.ets 文件，可以看到 EntryAbility 是继承了 UIAbility 的：



```
import ...
const DOMAIN = 0x0000;
export default class EntryAbility extends UIAbility {
    oncreate(want: Want, LaunchParam: AbilityConstant.LaunchParam): void {
        this.context.getApplicationContext().setColorMode(ConfigurationConstant.Col
        hilog.info(DOMAIN, 'testTag', '{public}s', 'Ability onCreate');
    }
    onDestroy(): void {
        hilog.info(DOMAIN, 'testTag', '{public}s', 'Ability onDestroy');
    }
    onWindowStageCreate(windowStage: window.WindowStage): void {
        // Main window is created, set main stage for this ability
    }
}
```

然后我们查看 module.json5 文件，可以看到，我们满足这个条件：

- HAP中包含UIAbility
 - 如果在module.json5配置文件的abilities标签中配置了icon和label，且该对应的ability中skills标签下面的entities中包含"entity.system.home"、actions中包含"ohos.want.action.home"或者"action.system.home"，则系统将优先返回module.json5中的icon与label。如果存在多个满足条件的ability，优先返回module.json5中mainElement对应的ability配置的icon和label。

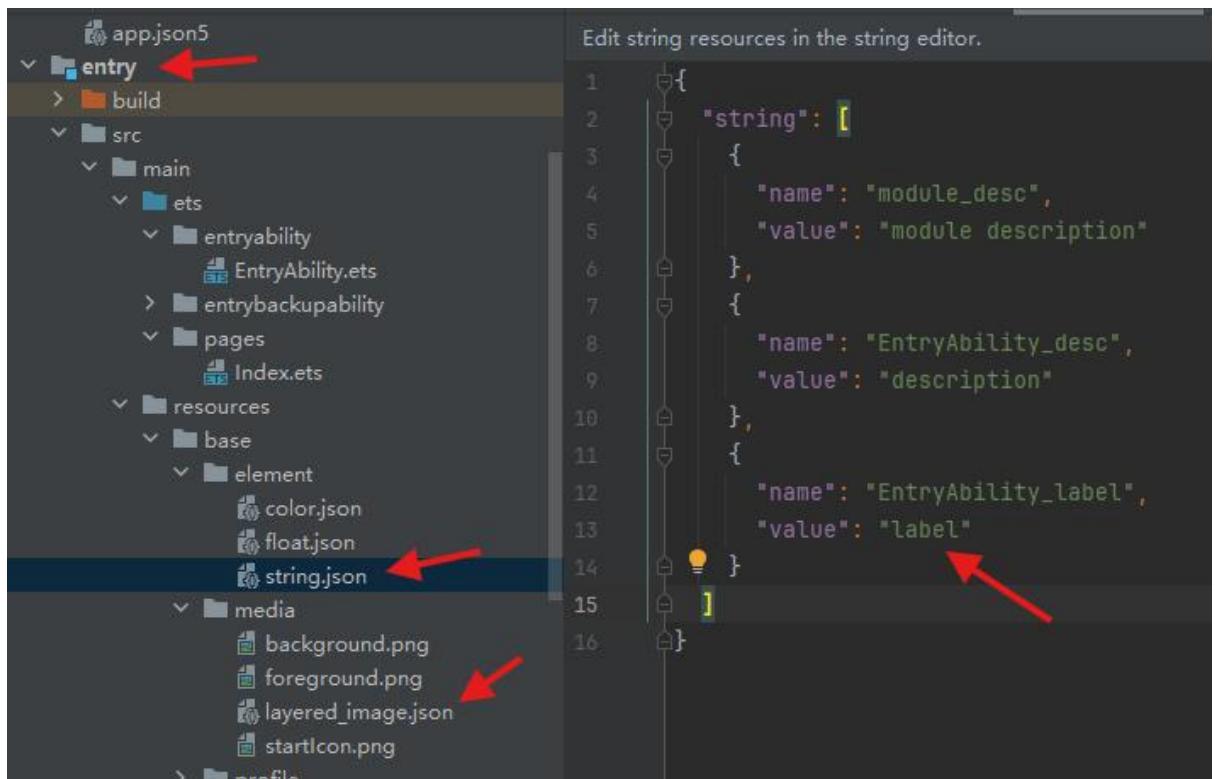
因此，显示的是这个文件中定义的 icon 图标和 label 标签。

```

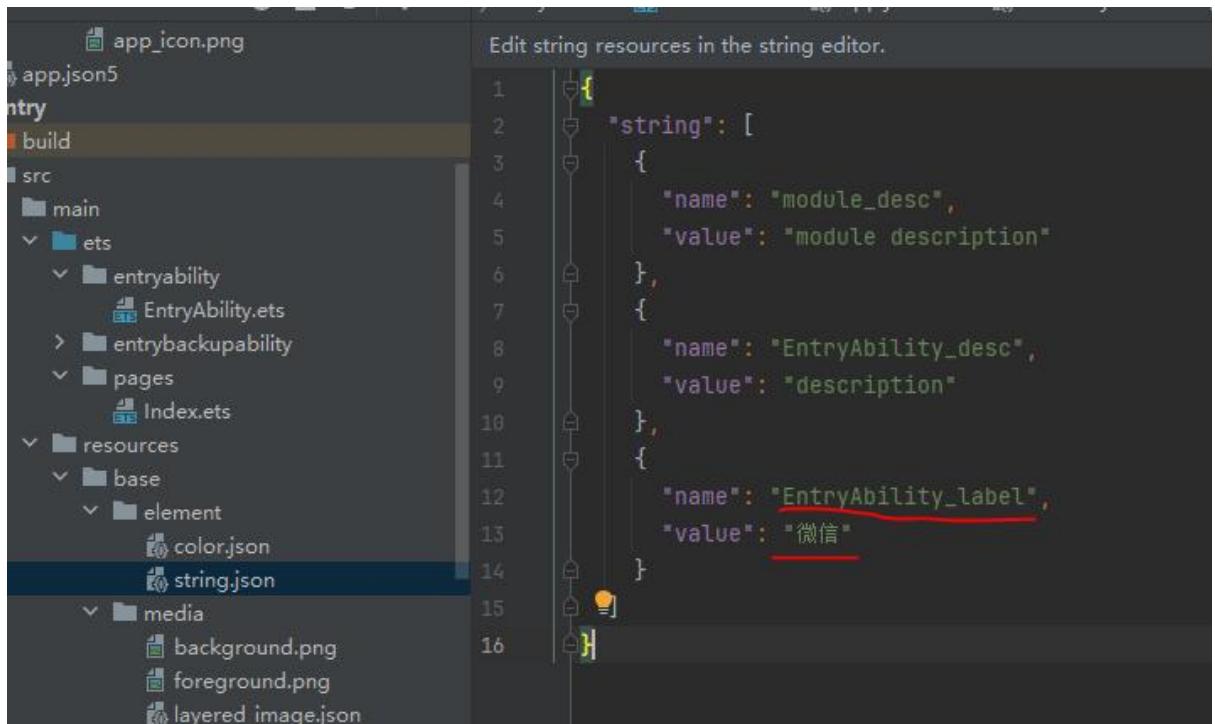
    "name": "ENTRYABILITY",
    "srcEntry": "./ets/entryability/EntryAbility.ets",
    "description": description,
    "icon": "$media:layered_image", ←
    "label": "$string:EntryAbility_label", ←
    "startWindowIcon": "$media:startIcon",
    "startWindowBackground": "#FFFFFF",
    "exported": true,
 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

```

而这里的\$media 和\$string，对应的是 entry > src > main > resources > base 下面的 string.json 和 media 目录中的资源：



我们改一下这里的资源：



```
app.json5
entry
build
src
  main
    ets
      entryability
        EntryAbility.ets
      entrybackupability
      pages
        Index.ets
    resources
      base
        element
          color.json
          string.json
      media
        background.png
        foreground.png
        layered image.json
```

Edit string resources in the string editor.

```
1 "string": [
2   {
3     "name": "module_desc",
4     "value": "module description"
5   },
6   {
7     "name": "EntryAbility_desc",
8     "value": "description"
9   },
10  {
11    "name": "EntryAbility_label",
12    "value": "微信"
13  }
14]
```

把 EntryAbility_label 对应的值改为“微信”。

此时，先卸载掉这个应用再安装才能生效：（按住鼠标左键，出现卸载按钮后点击）



然后点击“卸载”：（可以顺便把其他的实例 APP 都卸载掉）

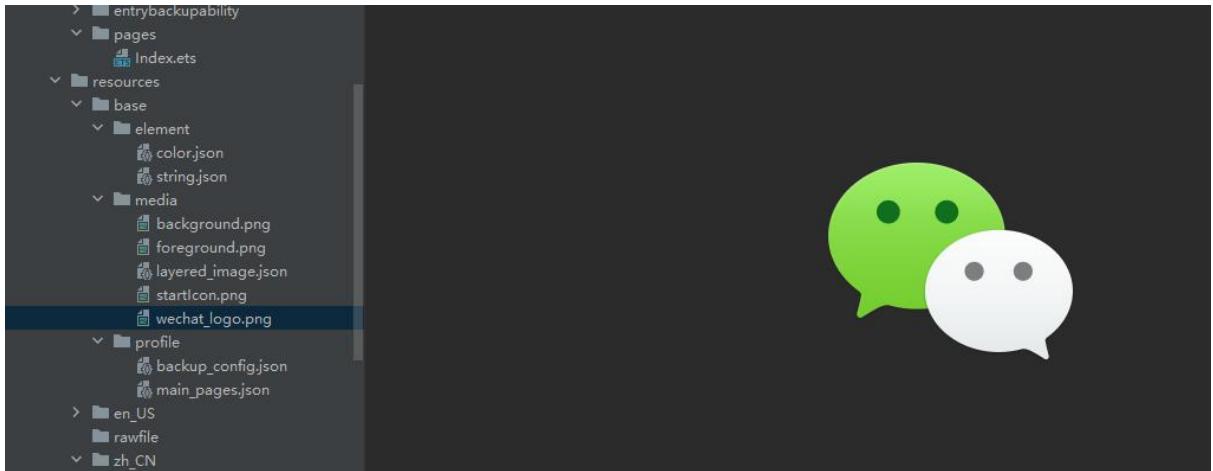


卸载之后，再次重新在模拟器中运行，结果：



可以看到，标签成功改为微信。

我们再去网上下载一个微信的 LOGO，放到 resources > base > media 下面：



然后修改 module.json5 中的 icon 对应的值：

```
"abilities": [  
    {  
        "name": "EntryAbility",  
        "srcEntry": "./ets/entryability/EntryAbility.ets",  
        "description": "$string:EntryAbility_desc",  
        "icon": "$media:wechat_logo",  
        "label": "$string:EntryAbility_label",
```

```
    "pages": "$profile:main_pages",
    "abilities": [
        {
            "name": "EntryAbility",
            "srcEntry": "./ets/entryability/EntryAbility.ets",
            "description": description,
            "icon": "$media:",
            "label": "$media:app_icon"
            "startWi
            "startWi
            "exporte
            "skills"
            {
                "entities": [

```

再来一次卸载，重新运行，效果：



当然，这里的所谓“微信”APP是我们的Hello World应用。

现在，我们再试试这里说的另外一个方法：

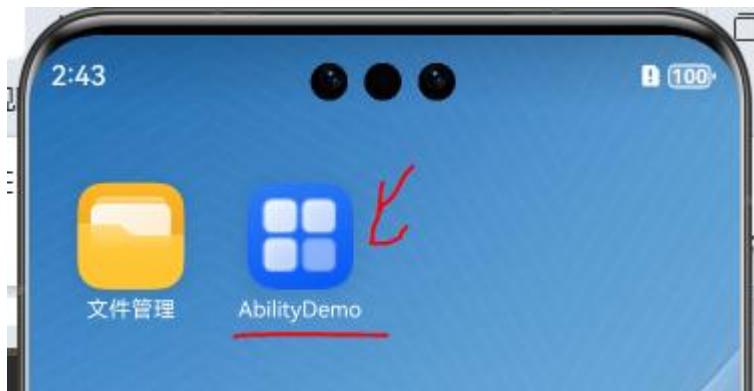
- 如果在module.json5配置文件的abilities标签中未设置icon和label，或者对应的skills标签下面的entities与actions没有做相关配置(entities中包含"entity.system.home"、actions中包含"ohos.want.action.home")，那么系统将使用app.json5中的icon和label。

首先，去到 module.json5 这里，注释掉 skills 里面的配置：

```
module.json5  AppScope\...\...\string.json  resources\zh_CN\...\string.json

18     "srcEntry": "./ets/entryability/EntryAbility.ets",
19     "description": description,
20     "icon": "$media:wechat_logo",
21     "label": "微信",
22     "startWindowIcon": "$media:startIcon",
23     "startWindowBackground": "#FFFFFF",
24     "exported": true,
25     "skills": [
26         {
27             "entities": [
28                 "entity.system.home"
29             ],
30             "actions": [
31                 "action.system.home"
32             ]
33         }
34     ]
35 }
```

此时再次卸载，卸载掉我们的假微信，重新运行：



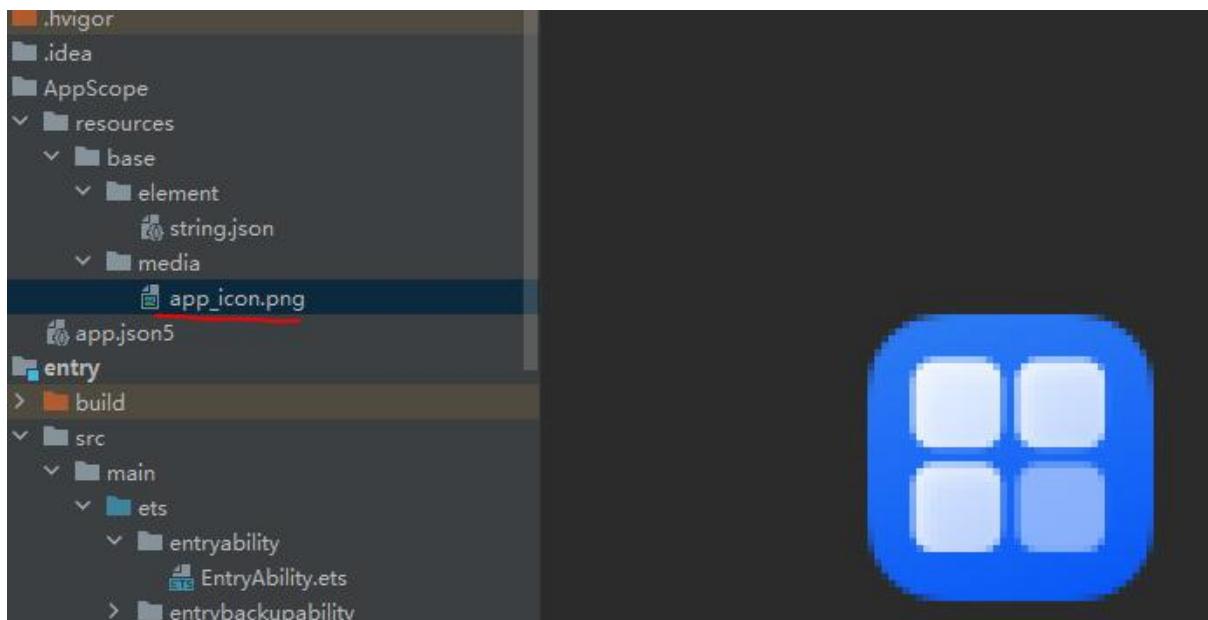
可以看到，label 标签成为 AbilityDemo，而图标变为缺省图标。这个和 app.json5 文件中的配置是一致的：

```

1 app": {
2     "bundleName": "com.example.abilitydemo",
3     "vendor": "example",
4     "versionCode": 1000000,
5     "versionName": "1.0.0",
6     "icon": "$media:app_icon",
7     "label": "AbilityDemo"
8 }
9
10
11

```

图标:



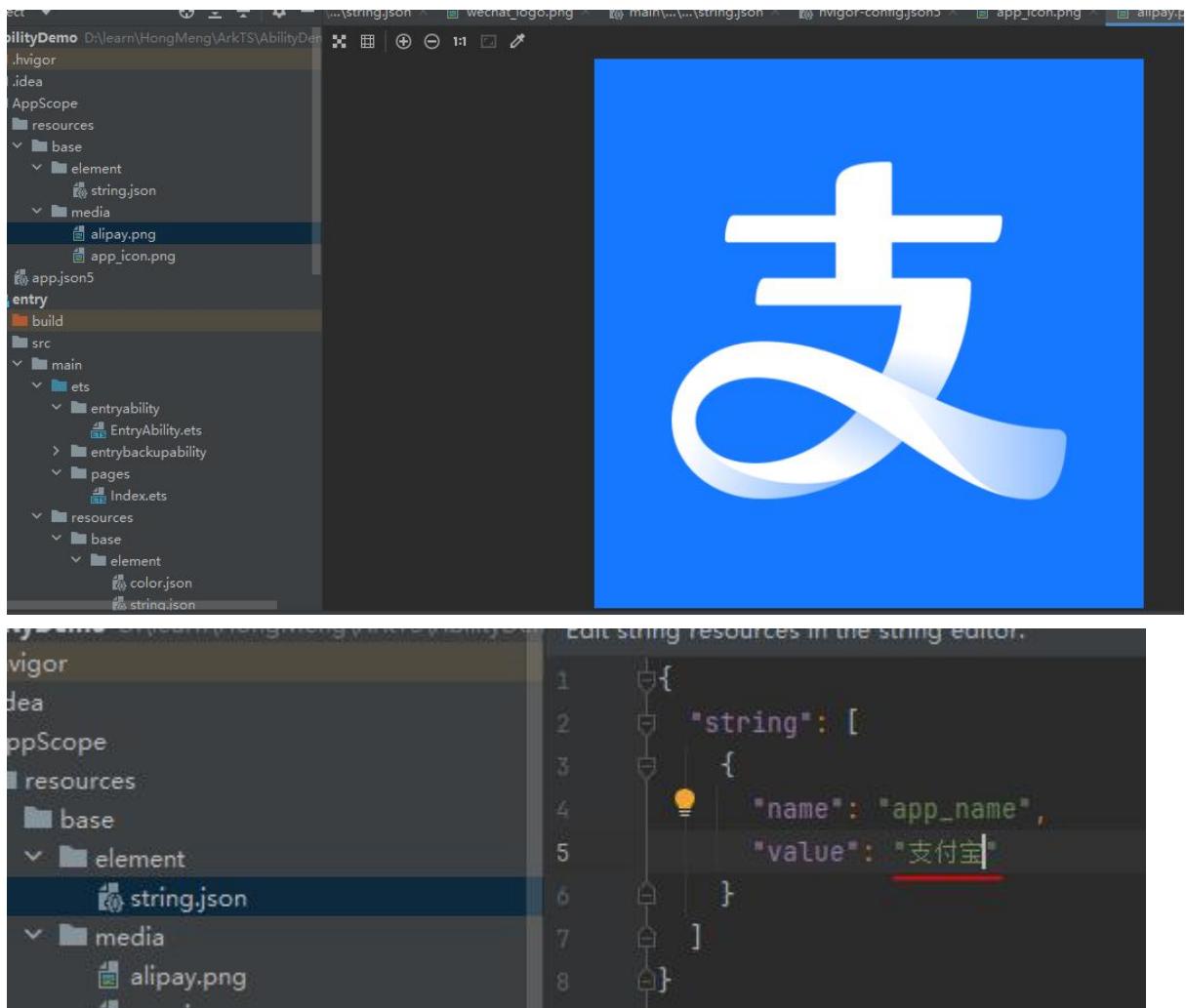
标签:

```

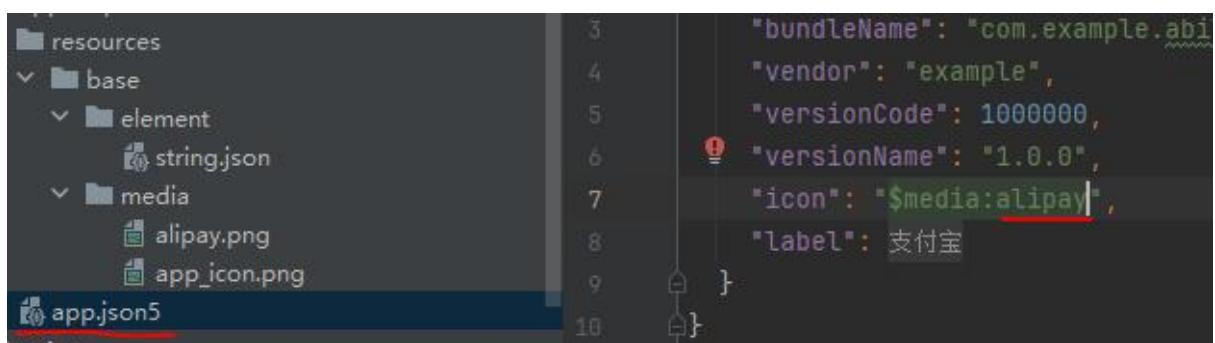
1 {
2     "string": [
3         {
4             "name": "app_name",
5             "value": "AbilityDemo"
6         }
7     ]
8 }

```

我们把标签改为“支付宝”，再去下载一个支付宝的图标放到 AppScope > base > media下面:



然后，修改 app.json5 文件：



再次卸载，重新运行：



此时，我们到模拟器的“设置”->“应用和元服务”中查看：



点进去：



你也可以自己尝试修改一下版本号“versionName”，查看效果，比如我改为 2.0.2，注意版本号只能是在这里配置：

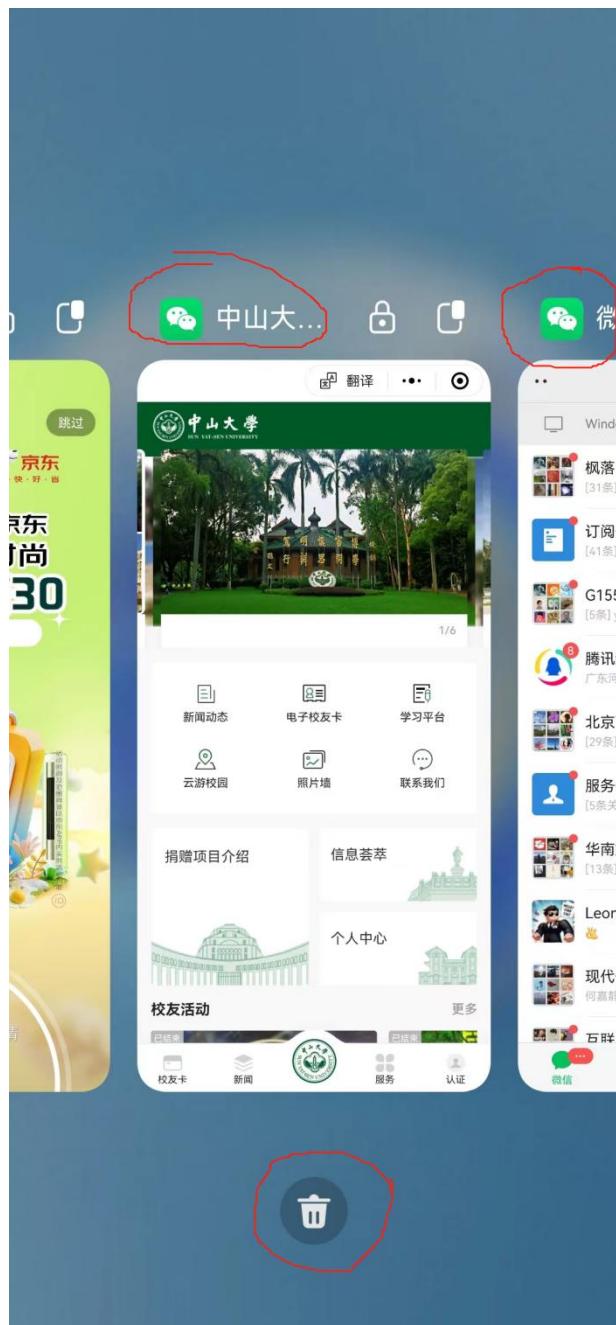


OK，到这里，相信你知道如何去配置自己的 APP 的图标和名字了。

4. Stage 模型中的 UIAbility 组件

UIAbility 是一种包含了用户界面的应用组件，主要用于和用户进行交互。每一个 UIAbility 实例，都对应于一个最近任务列表中的任务。

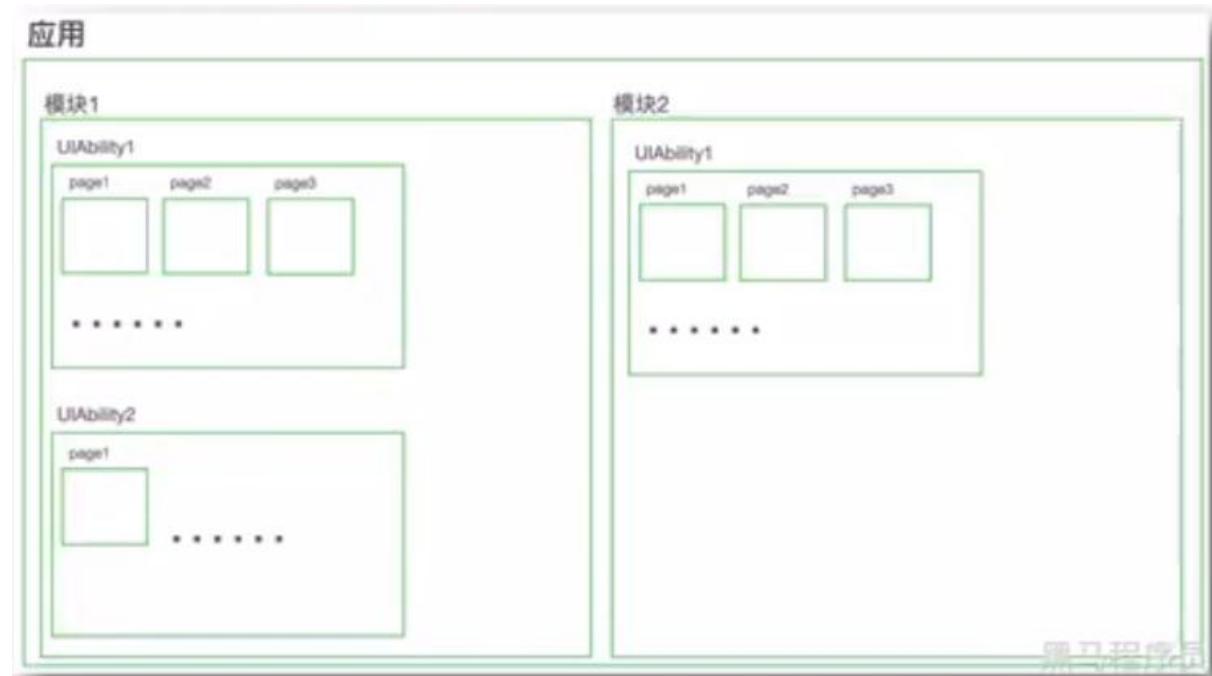
一个应用可以有一个，也可以有多个 UIAbility 组件或实例，对于只有单个的，任务列表中只有一个任务，而对于多个的，任务列表中有多个任务。比如对于微信，我在微信里面启动了一个小程序，那么滑动屏幕可以看到有两个微信界面的应用实例：



对于 UIAbility，我们学习一下几个方面的知识：

- 如何配置启动页面
- UIAbility 的生命周期
- UIAbility 组件之间的交互

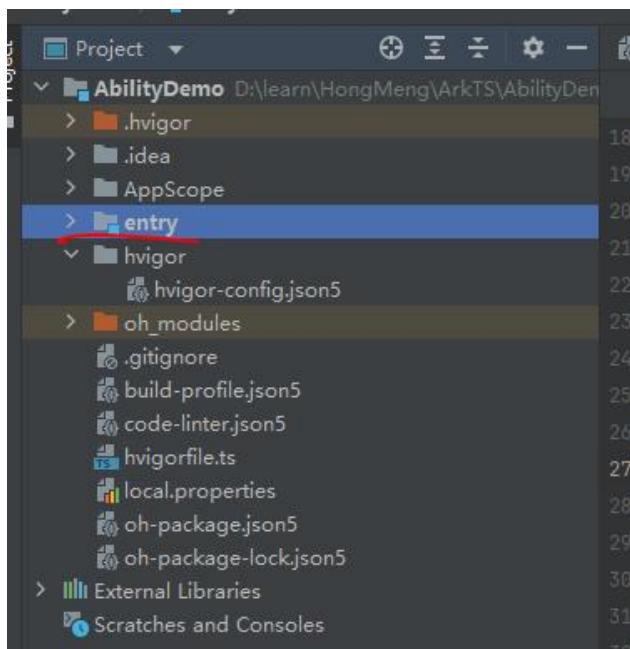
当然，这里还牵扯到模块的概念，我们知道，一个应用是可以有多个模块的，而每个模块可以有单个或多个 UIAbility：



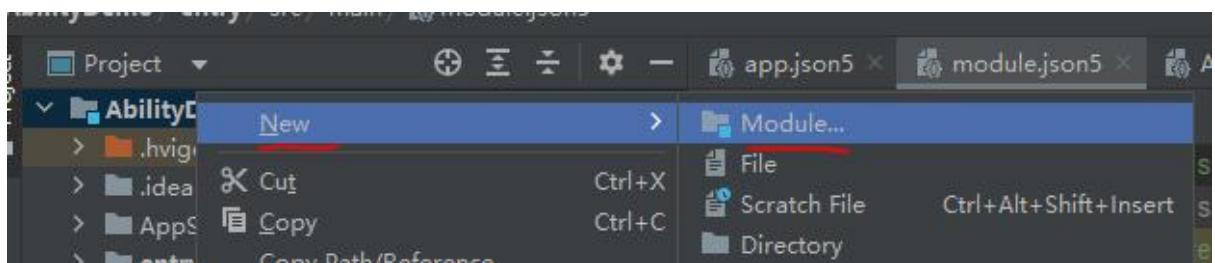
我们在前面了解到，模块有 HAP，HAR，HSP 等类型。我们一直在做的都是 HAP。

如何添加新的模块

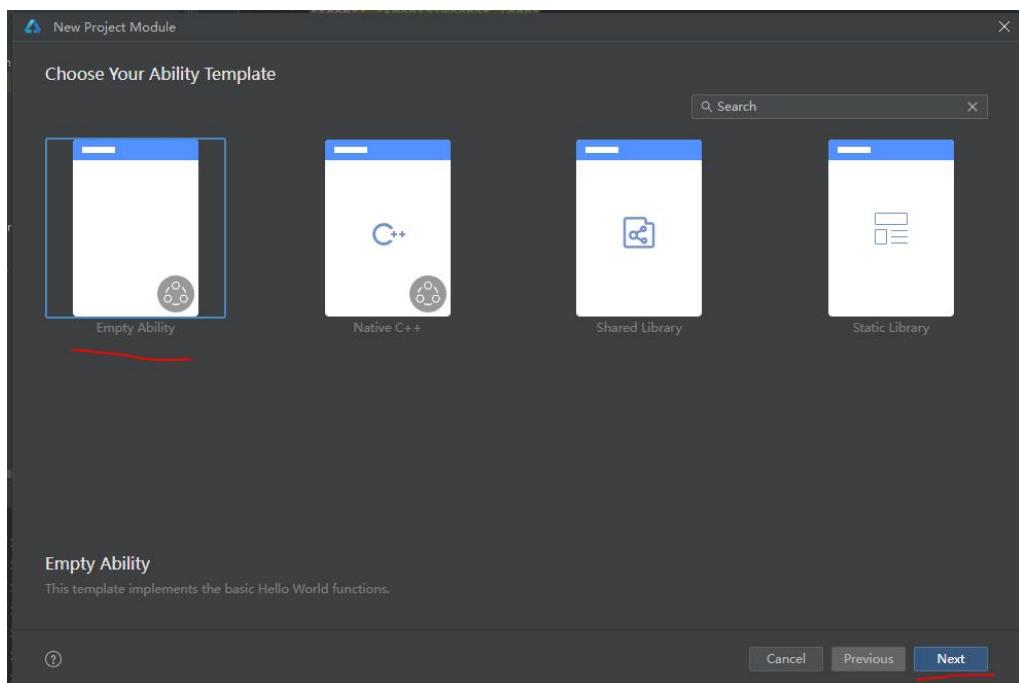
首先，我们先增加一个新的模块。目前我们只有一个模块 entry：



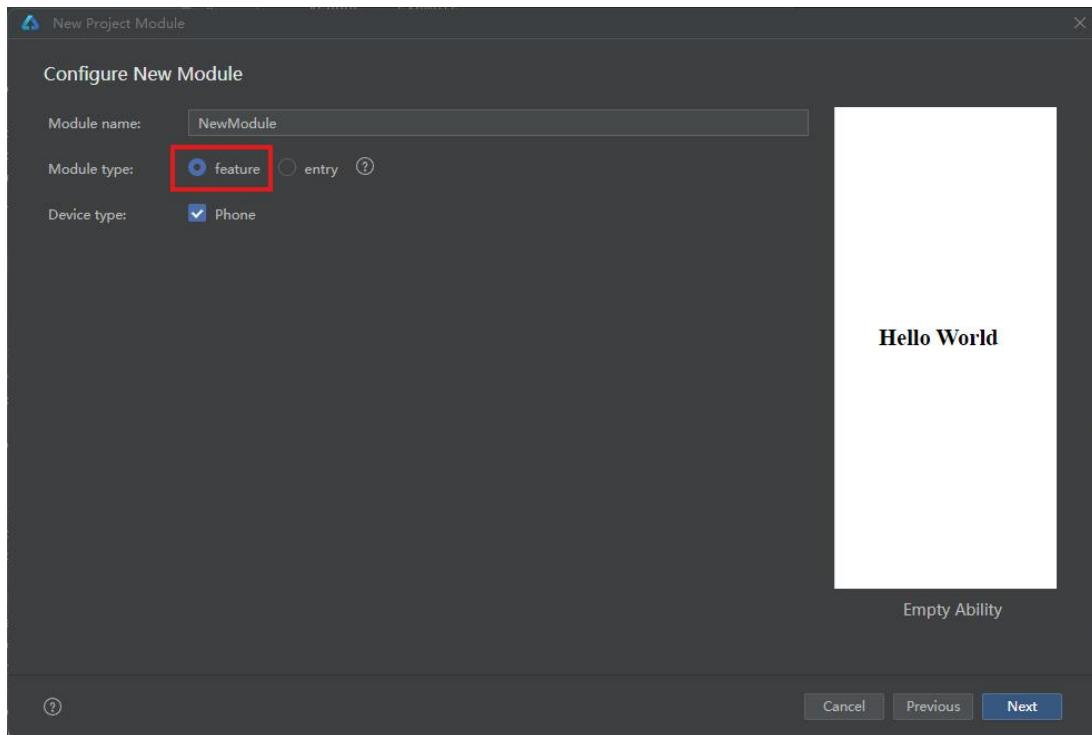
在项目应用名 AbilityDemo 上面，点击右键，选择模块 Module...，添加一个新的模块：



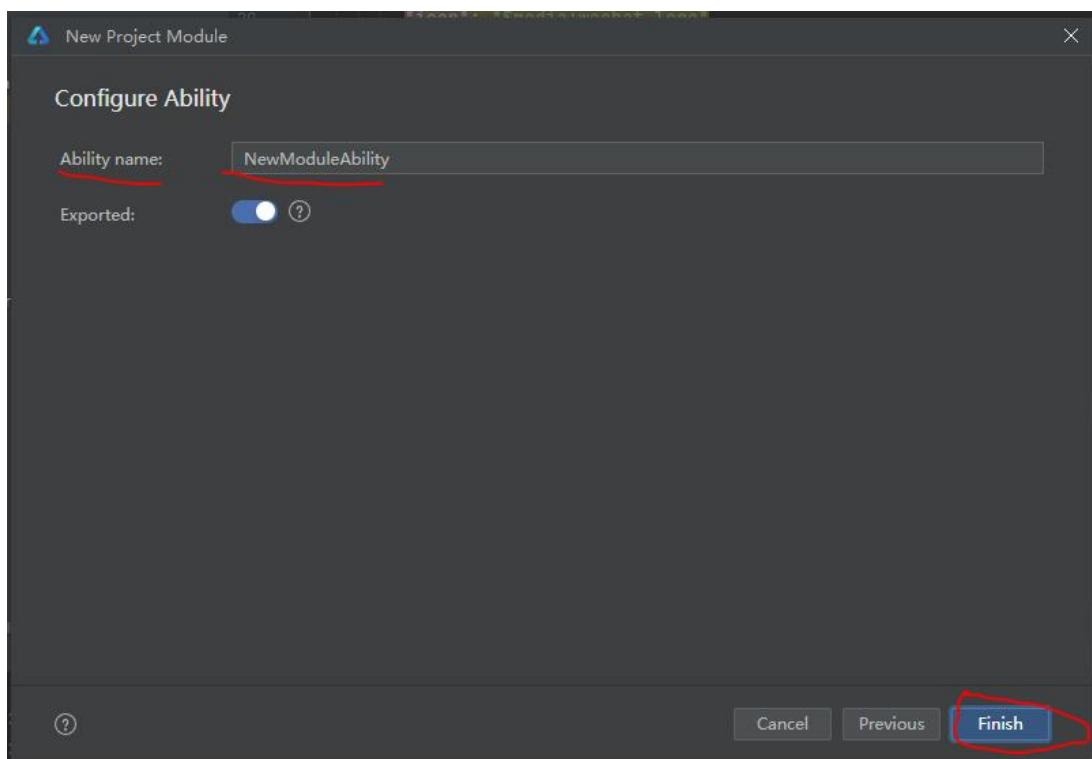
选择 Empty Ability，然后点击 Next：



取一个名字，我这里用的是“NewModule”，选择 feature（由于我们已经有一个 entry 类型的模块了，所以这里其实我们只能选 feature），然后点击 Next：

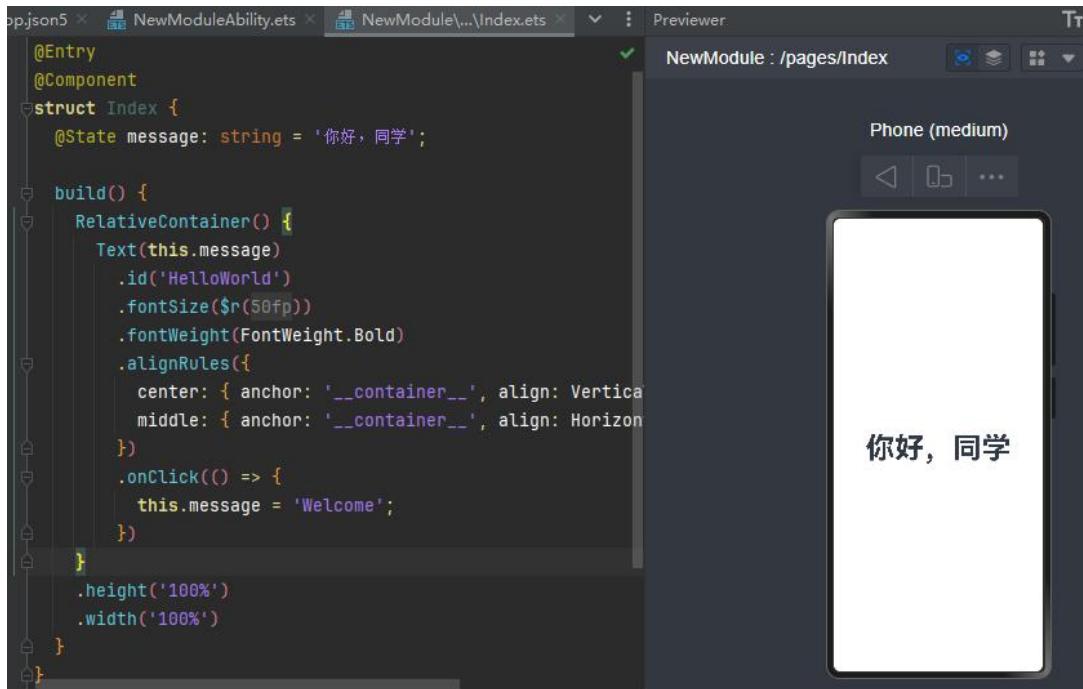


Ability 的名字可以直接用默认的命名，然后点击 Finish：

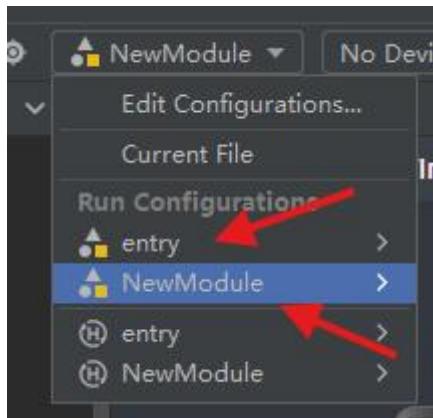


此时，可以看到，新的模块被创建了，和 entry 的目录结构一样，pages 里面的 Index.ets 的初始代码

也是一样的。我们把 message 的值改为“你好，同学”，然后用预览器查看：



而对于模拟器，我们可以选择运行 entry，也运行 NewModule：



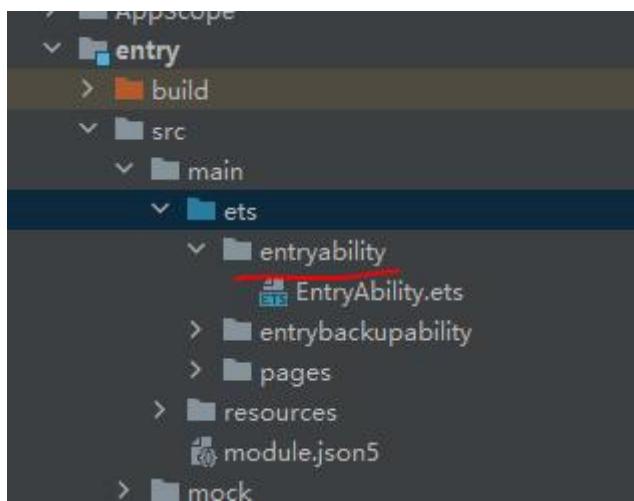
可以尝试选择 NewModule，运行：



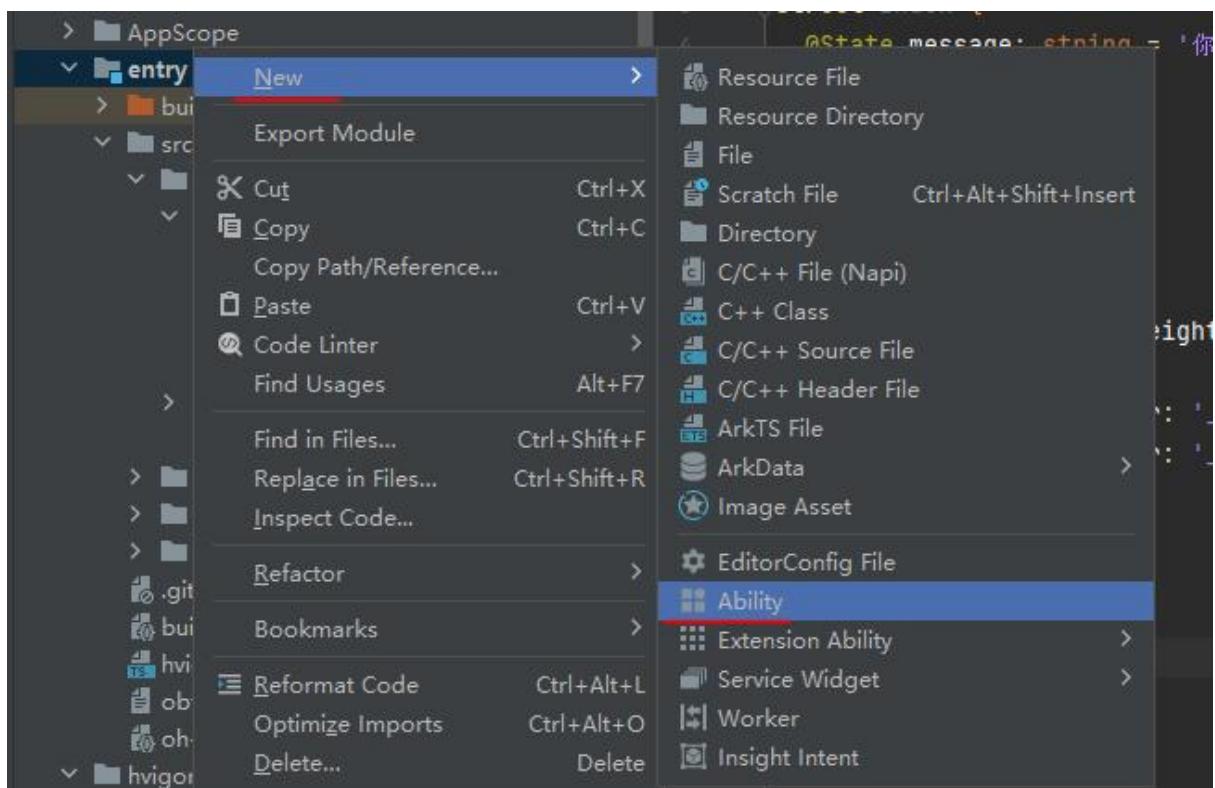
好的，现在我们知道了如何添加一个模块。

如何添加新的 UIAbility

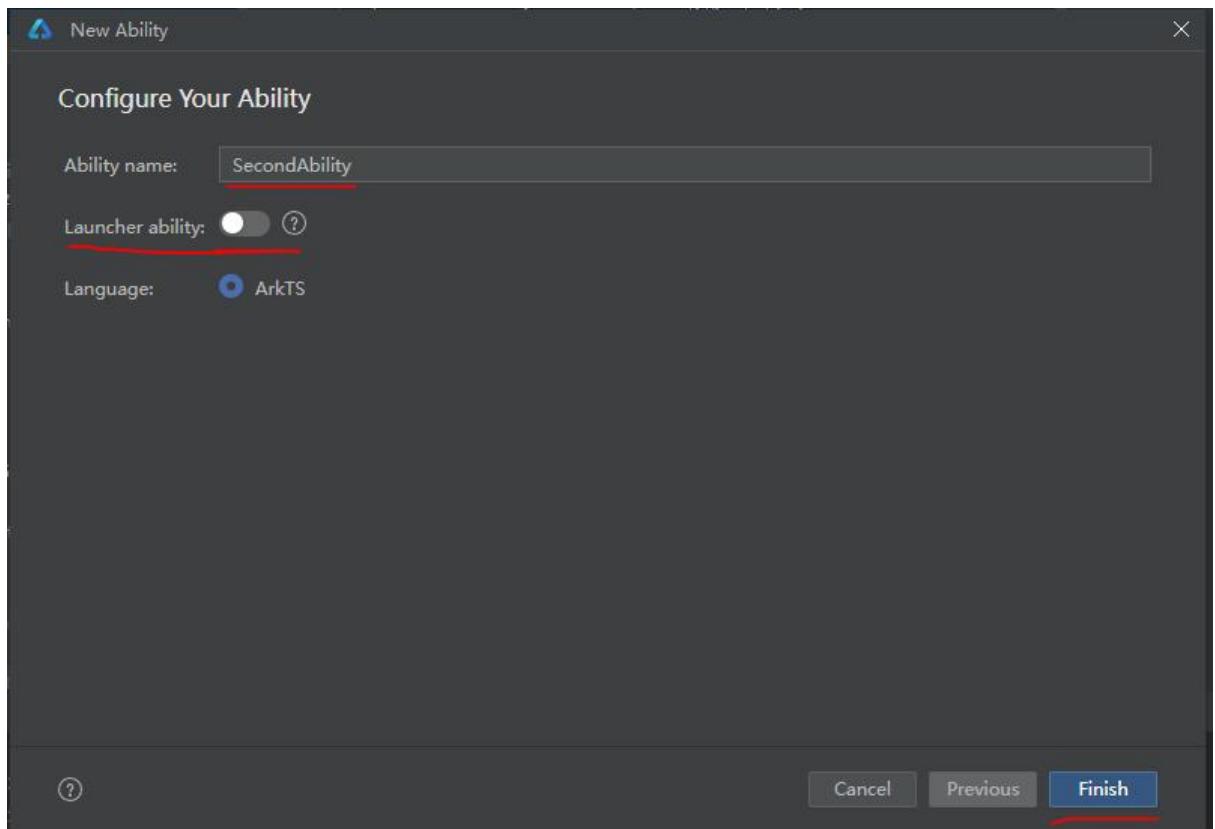
回到 entry 这个模块，如何在这个模块中添加一个新的 UIAbility 呢？目前，在 entry 模块里面，只有一个 entryability：



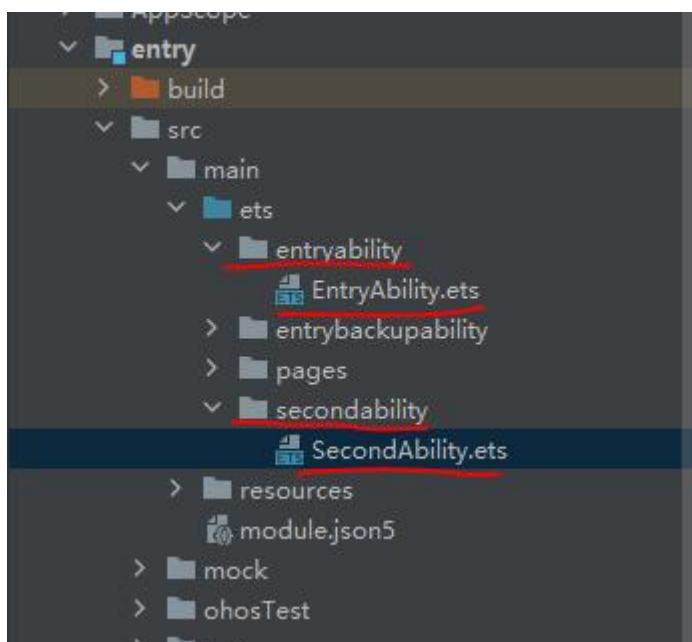
在 entry 上点击右键，新增一个 Ability:



设一个名字，我这里取名为“SecondAbility”，对于“Launcher ability”，它的意思是问你是否需要在屏幕上为这个 Ability 单独设置图标和标签。我们先不选。点击 Finish:



此时，我们看到，entry 模块下面多了一个 secondability:



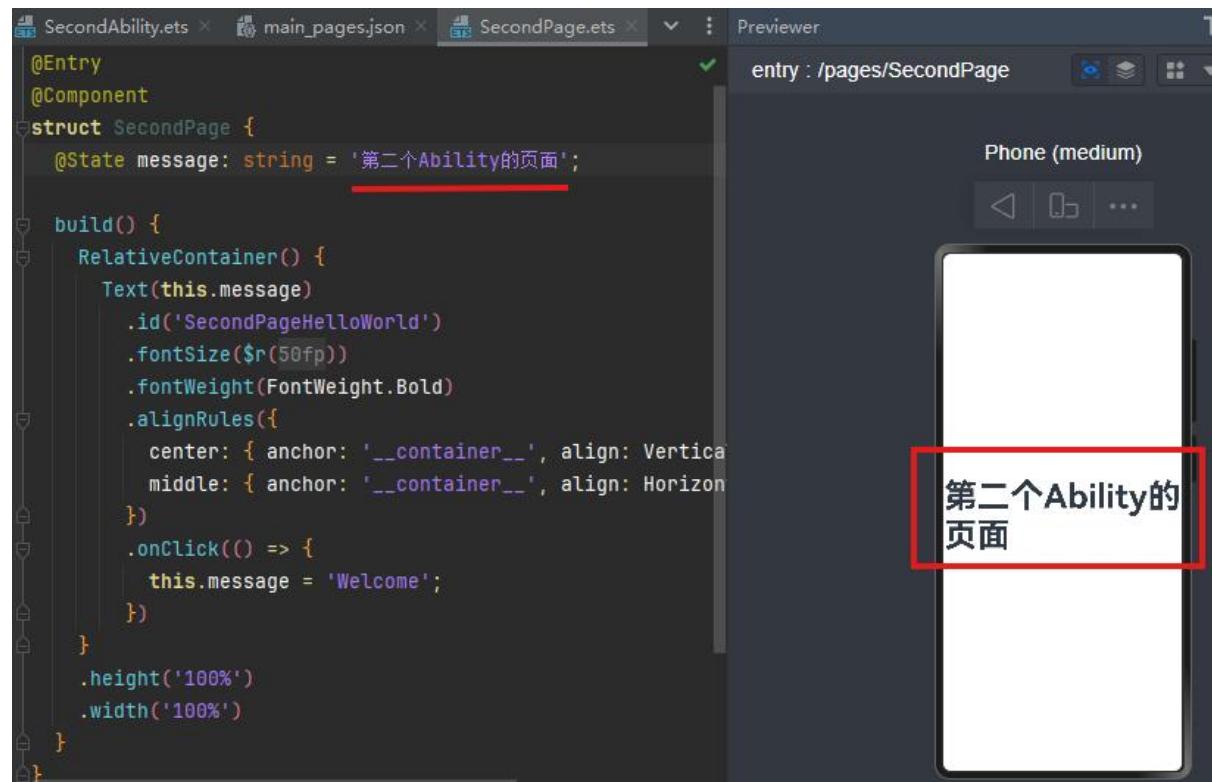
查看 SecondAbility.ets 文件中的代码，看到 loadContent 用的页面还是 pages/Index，也就是说，两个 Ability 其实是可以用到同一个 pages 目录下的页面。

```

    onWindowStageCreate(windowStage: window.WindowStage): void {
        // Main window is created, set main page for this ability
        hilog.info(DOMAIN, 'testTag', '%{public}s', 'Ability onWindowStageCreate');
        windowStage.loadContent('pages/Index', (err) => {
            if (err.code) {
                hilog.error(DOMAIN, 'testTag', 'Failed to load the content. Code: %{public}d', err.code);
            }
        });
    }
}

```

为了有所区分，我们在 pages 目录下增加一个新的页面 SecondPage.ets（注意，请使用创建新的 Page 的方式，而不是 ArkTS File 的方式），改一下 message，显示不同的信息：

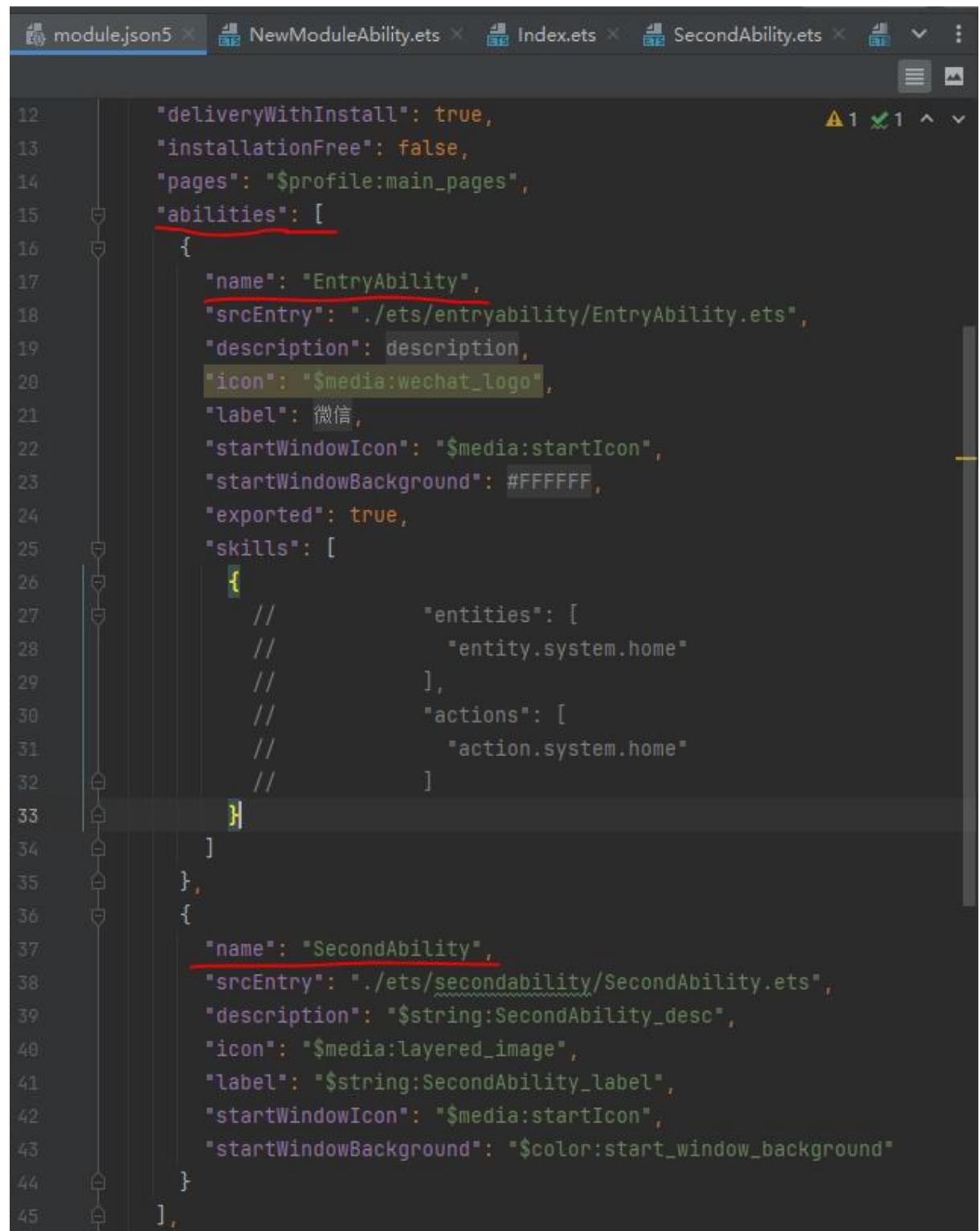


然后，我们把 `SecondAbility.ets` 中的初始 `loadContent` 的内容改为使用 `SecondPage`:



相信学到这里，同学们应该不需要我再给详细的代码了。

在新增了 Ability 之后，查看模块的配置文件 module.json5，可以看到，在“abilities”数组里面，增加了一个新的 Ability，就是 SecondAbility：



The screenshot shows a code editor with multiple tabs at the top: module.json5 (selected), NewModuleAbility.ets, Index.ets, SecondAbility.ets, and others. The module.json5 tab contains the following JSON code:

```
12     "deliveryWithInstall": true,
13     "installationFree": false,
14     "pages": "$profile:main_pages",
15     "abilities": [
16         {
17             "name": "EntryAbility",
18             "srcEntry": "./ets/entryability/EntryAbility.ets",
19             "description": description,
20             "icon": "$media:wechat_logo",
21             "label": 微信,
22             "startWindowIcon": "$media:startIcon",
23             "startWindowBackground": "#FFFFFF",
24             "exported": true,
25             "skills": [
26                 {
27                     //           "entities": [
28                     //             "entity.system.home"
29                     //           ],
30                     //           "actions": [
31                     //             "action.system.home"
32                     //           ]
33                 }
34             ]
35         },
36         {
37             "name": "SecondAbility",
38             "srcEntry": "./ets/secondability/SecondAbility.ets",
39             "description": "$string:SecondAbility_desc",
40             "icon": "$media:layered_image",
41             "label": "$string:SecondAbility_label",
42             "startWindowIcon": "$media:startIcon",
43             "startWindowBackground": "$color:start_window_background"
44         }
45     ]
```

通过对“mainElement”和“exported”的改动，我们可以把SecondAbility配置为APP刚启动时候的UI界面，修改这几个地方：

The screenshot shows a code editor with a dark theme displaying a JSON configuration file. The file defines a module named 'entry' with a type of 'entry'. It includes a 'mainElement' field set to 'SecondAbility', which is underlined in red. The 'abilities' array contains two entries. The first entry is for 'EntryAbility', which has its 'exported' field commented out with a double slash. The second entry is for 'SecondAbility', which has its 'exported' field explicitly set to true. Both entries have their 'srcEntry' fields pointing to ETS files: 'entryability/EntryAbility.ets' and 'secondability/SecondAbility.ets' respectively. The 'SecondAbility' entry also includes a 'skills' array.

```
{  
  "module": {  
    "name": "entry",  
    "type": "entry",  
    "description": "module description",  
    "mainElement": "SecondAbility",  
    "deviceTypes": [...],  
    "deliveryWithInstall": true,  
    "installationFree": false,  
    "pages": "$profile:main_pages",  
    "abilities": [  
      {  
        "name": "EntryAbility",  
        "srcEntry": "./ets/entryability/EntryAbility.ets",  
        "description": "description",  
        "icon": "$media:wechat_logo",  
        "label": "微信",  
        "startWindowIcon": "$media:startIcon",  
        "startWindowBackground": "#FFFFFF",  
        // "exported": true,  
        "skills": [...]  
      },  
      {  
        "name": "SecondAbility",  
        "srcEntry": "./ets/secondability/SecondAbility.ets",  
        "description": "$string:SecondAbility_desc",  
        "icon": "$media:layered_image",  
        "label": "$string:SecondAbility_label",  
        "startWindowIcon": "$media:startIcon",  
        "startWindowBackground": "$color:start_window_background",  
        "exported": true  
      }  
    ],  
    "extensionAbilities": [  
    ]  
  }  
}
```

也就是把 mainElement 这里配置为“SecondAbility”，把 EntryAbility 的“exported”注释掉，或者改为 false，然后在 SecondAbility 这里增加一行“exported”，值设为 true。

此时，运行模拟器（注意是选择 entry 模块）：



先启动的是 SecondAbility 的首页，也就是 loadContent 加载的 SecondPage。

当然，此时，对于 SecondAbility，如果我们加上 skills 的配置，然后把 entities 和 actions 配置上：

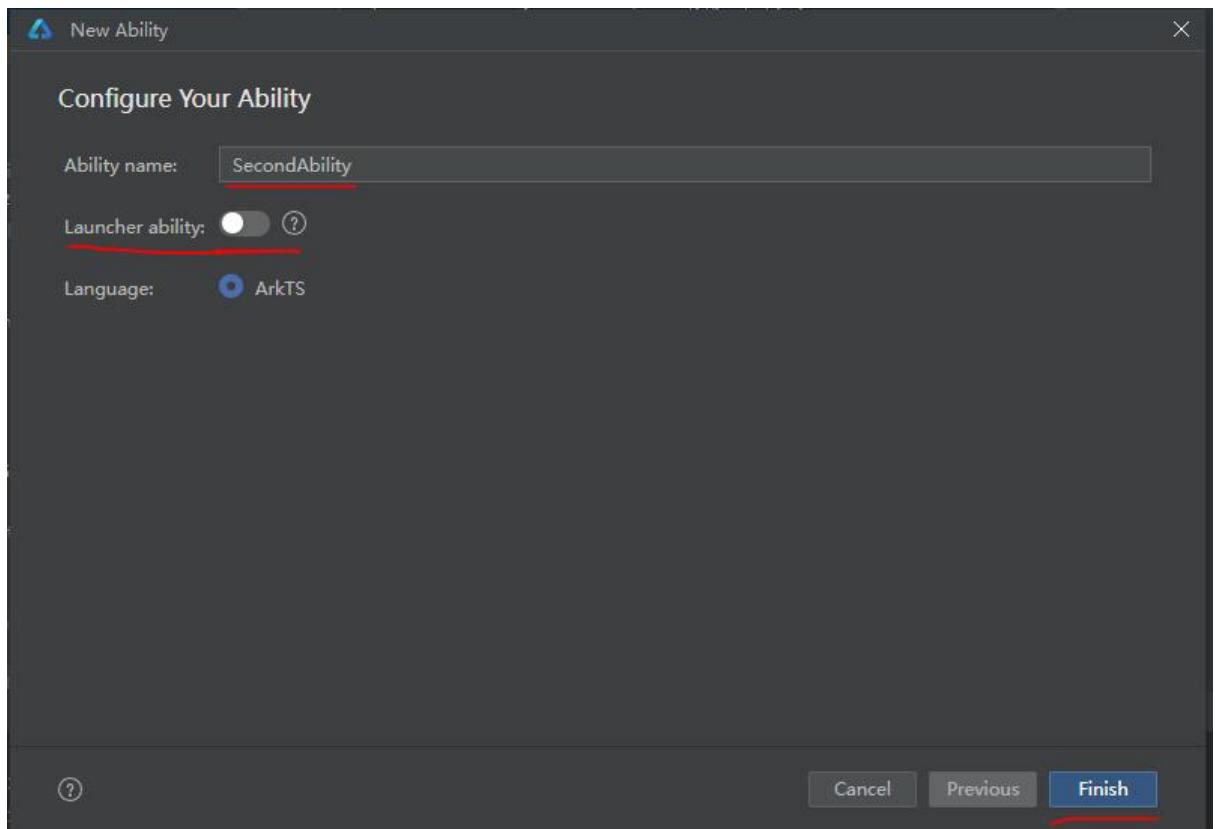
```
{  
    "name": "SecondAbility",  
    "srcEntry": "./ets/secondability/SecondAbility.ets"  
    "description": description,  
    "icon": "$media:layered_image",  
    "label": label,  
    "startWindowIcon": "$media:startIcon",  
    "startWindowBackground": "#FFFFFF",  
    "exported": true, 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

卸载 APP，再次在模拟器上运行，此时应用的图标和便签变为：

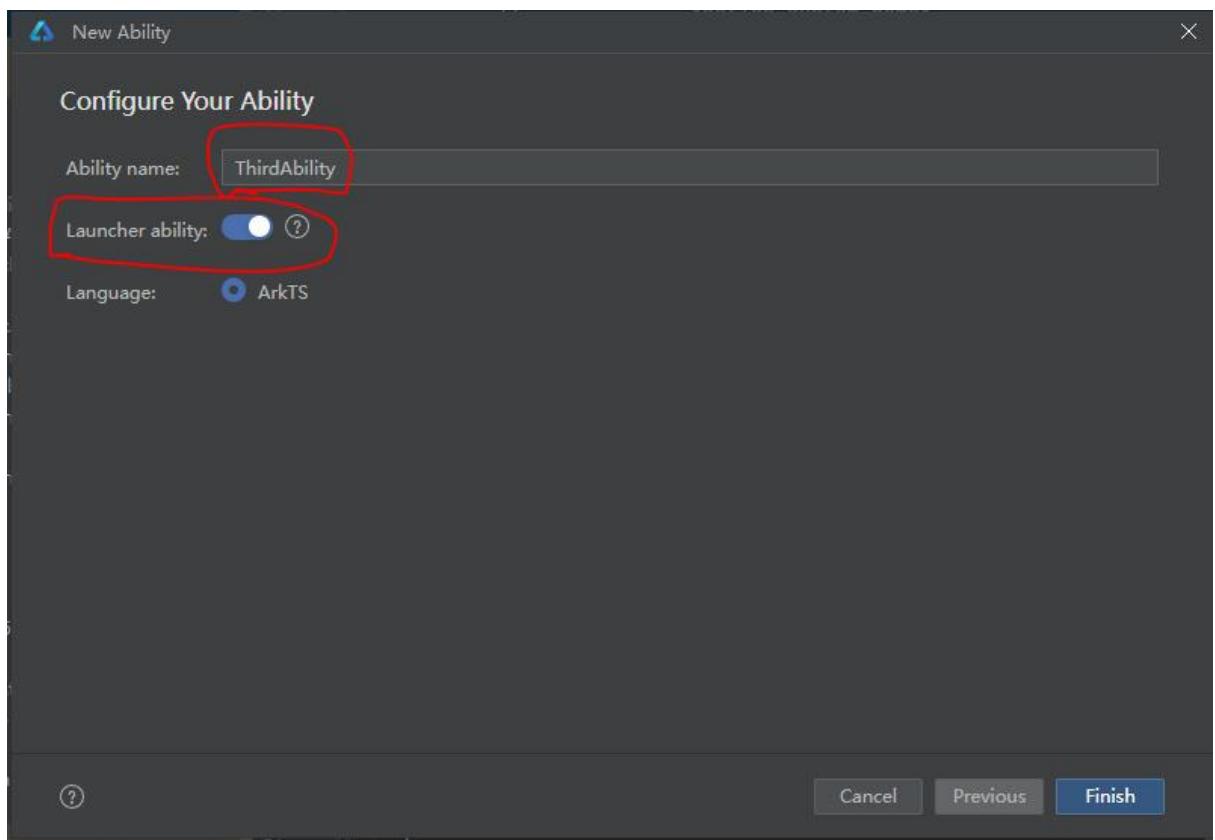


至于为什么会是这样？相信大家可以自己找到答案。

前面在添加 SecondAbility 的时候，在 Launcher ability 这里选择了 false:



我们尝试再添加一个 ThirdAbility，在这一步选择 true：



同样，添加一个新的页面 ThirdPage，改一下 message，也在 loadContent 加载初始页面那里加载 ThirdPage：

```
15
16  onWindowStageCreate(windowStage: window.WindowStage): void
17      // Main window is created, set main page for this ability
18      hilog.info(DOMAIN, 'testTag', '%{public}s', 'Ability on')
19
20      windowStage.loadContent('pages/ThirdPage', (err) => {
21          if (err.code) {
22              hilog.error(DOMAIN, 'testTag', 'Failed to load the page');
23              return;
24          }
25          hilog.info(DOMAIN, 'testTag', 'Succeeded in loading the page');
26      });
27  }
28
29  onWindowStageDestroy(): void {
30      // Main window is destroyed, release UI related resources
31      hilog.info(DOMAIN, 'testTag', '%{public}s', 'Ability on');
32 }
```

加完之后，在 module.json5 中可以看到，对于第三个 Ability，自动把 exported 设为 true，也添加了 skills：

```
{
    "name": "ThirdAbility",
    "srcEntry": "./ets/thirdability/ThirdAbility.ets",
    "description": "$string:ThirdAbility_desc",
    "icon": "$media:layered_image",
    "label": "$string:ThirdAbility_label",
    "startWindowIcon": "$media:startIcon",
    "startWindowBackground": "$color:start_window_background",
    "exported": true,
    "skills": [
        {
            "entities": [
                "entity.system.home"
            ],
            "actions": [
                "action.system.home"
            ]
        }
    ]
},
"extensionAbilities": [
]
```

我们现在把 module.json5 文件改回到这样的状态：

- EntryAbility 是 mainElement
- 图标和标签使用 EntryAbility 的图标及标签，也就是“微信”

- SecondAbility 恢复原样

完整代码：

```
{  
    "module": {  
        "name": "entry",  
        "type": "entry",  
        "description": "$string:module_desc",  
        "mainElement": "EntryAbility",  
        "deviceTypes": [  
            "phone"  
        ],  
        "deliveryWithInstall": true,  
        "installationFree": false,  
        "pages": "$profile:main_pages",  
        "abilities": [  
            {  
                "name": "EntryAbility",  
                "srcEntry": "./ets/entryability/EntryAbility.ets",  
                "description": "$string:EntryAbility_desc",  
                "icon": "$media:wechat_logo",  
                "label": "$string:EntryAbility_label",  
                "startWindowIcon": "$media:startIcon",  
            }  
        ]  
    }  
}
```

```
        "startWindowBackground": "$color:start_window_background",

        "exported": true,

        "skills": [

        {

            "entities": [

                "entity.system.home"

            ],


            "actions": [

                "action.system.home"

            ]


        }

    ]


},


{

    "name": "SecondAbility",

    "srcEntry": "./ets/secondability/SecondAbility.ets",

    "description": "$string:SecondAbility_desc",

    "icon": "$media:layered_image",

    "label": "$string:SecondAbility_label",

    "startWindowIcon": "$media:startIcon",

    "startWindowBackground": "$color:start_window_background",

//        "exported": true,
```

```
//      "skills": [  
  
//      {  
  
//          "entities": [  
  
//              "entity.system.home"  
  
//          ],  
  
//          "actions": [  
  
//              "action.system.home"  
  
//          ]  
  
//      }  
  
//  ]  
  
},  
  
{  
  
    "name": "ThirdAbility",  
  
    "srcEntry": "./ets/thirdability/ThirdAbility.ets",  
  
    "description": "$string:ThirdAbility_desc",  
  
    "icon": "$media:layered_image",  
  
    "label": "$string:ThirdAbility_label",  
  
    "startWindowIcon": "$media:startIcon",  
  
    "startWindowBackground": "$color:start_window_background",  
  
    "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"  
            }  
        ],  
    },  
]
```

```
    }
```

```
]
```

```
}
```

```
}
```

此时，图标和标签是微信：



5. UIAbility 组件的生命周期

在继续讲解如何在模块内部的 UIAbility 之间跳转，以及不同模块之间的 UIAbility 之间跳转之前，先需要学习一下 UIAbility 的生命周期。

理论讲解：

<https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/uiability-lifecycle-V5>

当用户打开、切换和返回到对应应用时，应用中的 UIAbility 实例会在其生命周期的不同状态之间转换。UIAbility 类提供了一系列回调，通过这些回调可以知道当前 UIAbility 实例的某个状态发生改变，会经过 UIAbility 实例的创建和销毁，或者 UIAbility 实例发生了前后台的状态切换。

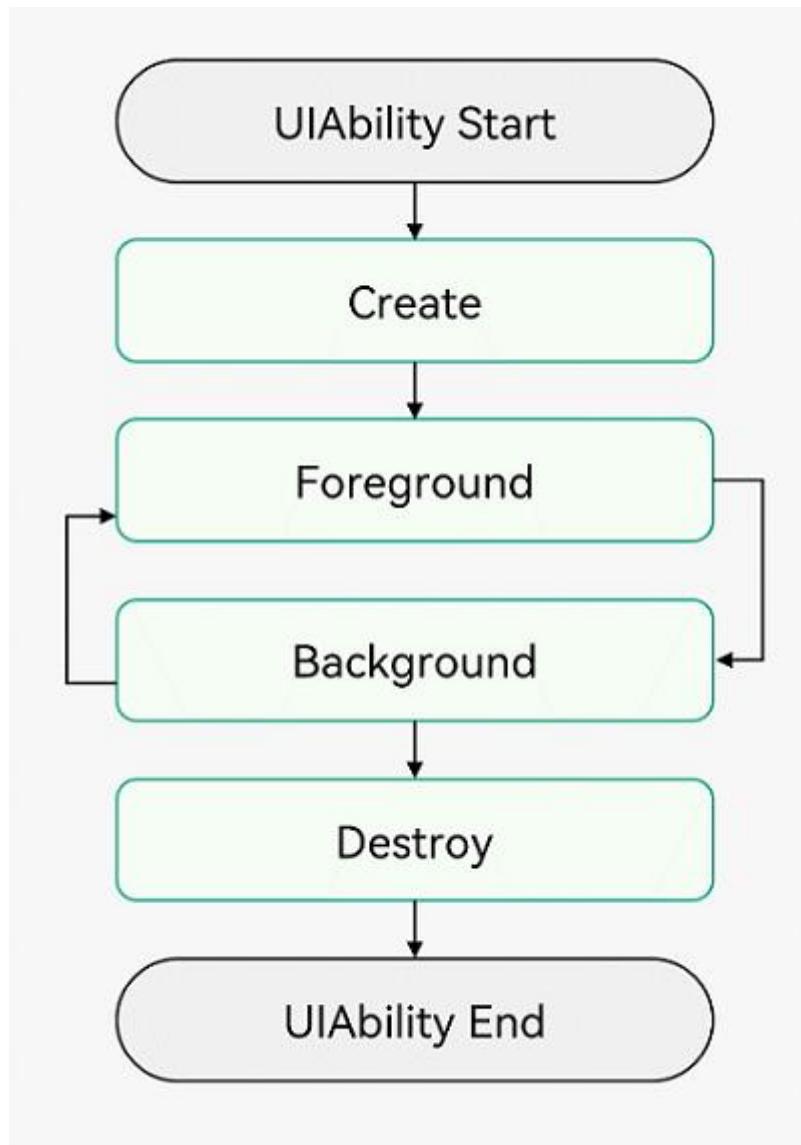
UIAbility 的生命周期包括 Create、Foreground、Background、Destroy 四个状态，如下图所示。

Create 状态

Create 状态为在应用加载过程中，UIAbility 实例创建完成时触发，系统会调用 onCreate() 回调。可以在该回调中进行页面初始化操作，例如变量定义资源加载等，用于后续的 UI 展示。

WindowStageCreate 和 WindowStageDestroy 状态（窗口舞台搭建/销毁）

UIAbility 实例创建完成之后，在进入 Foreground 之前，系统会创建一个 WindowStage。WindowStage 创建完成后会进入 onWindowStageCreate() 回调，可以在该回调中设置 UI 加载、设置 WindowStage 的事件订阅。在 UIAbility 实例销毁之前，则会先进入 onWindowStageDestroy() 回调，可以在该回调中释放 UI 资源。



WindowStageWillDestroy 状态

对应 onWindowStageWillDestroy() 回调，在 WindowStage 销毁前执行，此时 WindowStage 可以使用。

Foreground 和 Background 状态

Foreground 和 Background 状态分别在 UIAbility 实例切换至前台和切换至后台时触发，对应于 onForeground() 回调和 onBackground() 回调。

onForeground() 回调，在 UIAbility 的 UI 可见之前，如 UIAbility 切换至前台时触发。可以在 onForeground() 回调中申请系统需要的资源，或者重新申请在 onBackground() 中释放的资源。

onBackground() 回调，在 UIAbility 的 UI 完全不可见之后，如 UIAbility 切换至后台时候触发。可以在 onBackground() 回调中释放 UI 不可见时无用的资源，或者在此回调中执行较为耗时的操作，例如状态保存等。

Destroy 状态

Destroy 状态在 UIAbility 实例销毁时触发。可以在 onDestroy() 回调中进行系统资源的释放、数据的保存等操作。

其实，在每一个 UIAbility 创建的时候，系统已经帮我们创建了这些钩子函数，比如在 EntryAbility.ets 文件中，在 SecondAbility.ets 以及 ThirdAbility.ets 文件中。以 EntryAbility 为例，可以看到这些函数：

- onCreate
- onDestroy
- onWindowStageCreate
- onWindowStageDestroy
- onForeground
- onBackground

从名字就很容易看出来这些函数分别对应的状态。在这些函数里面，基本上都调用了 hilog.info 在控制台打印信息出来，加的标识都是 testTag。

```

export default class EntryAbility extends UIAbility {
    onCreate(want: Want, launchParam: AbilityConstant.LaunchParam): void {
        this.context.getApplicationContext().setColorMode(ConfigurationConstant.ColorMode.COLOR_MODE_NOT_SET);
        hilog.info(DOMAIN, 'testTag', '%{public}s', 'Ability onCreate');
    }

    onDestroy(): void {
        hilog.info(DOMAIN, 'testTag', '%{public}s', 'Ability onDestroy');
    }

    onWindowStageCreate(windowStage: window.WindowStage): void {
        // Main window is created, set main page for this ability
        hilog.info(DOMAIN, 'testTag', '%{public}s', 'Ability onWindowStageCreate');

        windowStage.loadContent('pages/Index', (err) => {
            if (err.code) {
                hilog.error(DOMAIN, 'testTag', 'Failed to load the content. Cause: %{public}s', JSON.stringify(err));
                return;
            }
            hilog.info(DOMAIN, 'testTag', 'Succeeded in loading the content.');
        });
    }

    onWindowStageDestroy(): void {
        // Main window is destroyed, release UI related resources
        hilog.info(DOMAIN, 'testTag', '%{public}s', 'Ability onWindowStageDestroy');
    }

    onForeground(): void {
        // Ability has brought to foreground
        hilog.info(DOMAIN, 'testTag', '%{public}s', 'Ability onForeground');
    }
}

```

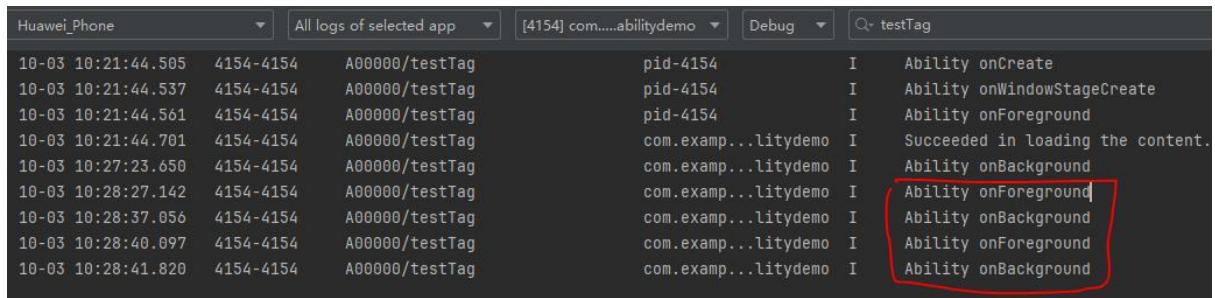
通过运行程序，查看 Log，可以看到这些生命周期函数调用的顺序：（注意用 testTag 来过滤一下）

Huawei_Phone	All logs of selected app	[9681] com....abilitydemo	Debug	testTag
11-18 21:47:29.210	9681-9681	A00000/testTag	com.examp...litydemo I	Ability onCreate
11-18 21:47:29.224	9681-9681	A00000/testTag	com.examp...litydemo I	Ability onWindowStageCreate
11-18 21:47:29.234	9681-9681	A00000/testTag	com.examp...litydemo I	Ability onForeground
11-18 21:47:29.264	9681-9681	A00000/testTag	com.examp...litydemo I	Succeeded in loading the content.
11-18 21:47:33.462	9681-9681	A00000/testTag	com.examp...litydemo I	Ability onBackground

可以看到，先调用 onCreate，然后开始搭建窗口舞台，然后显示到前台。注意，最后一行的 Succeeded in loading the content，是在 onWindowStageCreate 函数中调用的。我的理解是，这个函数里面调用了 loadContent，而这个是需要一些时间的，相当于是一个异步的过程，所以在加载成功之后，再输出这个 Succeeded in loading the content，而同时系统已经调用了 onForeground。

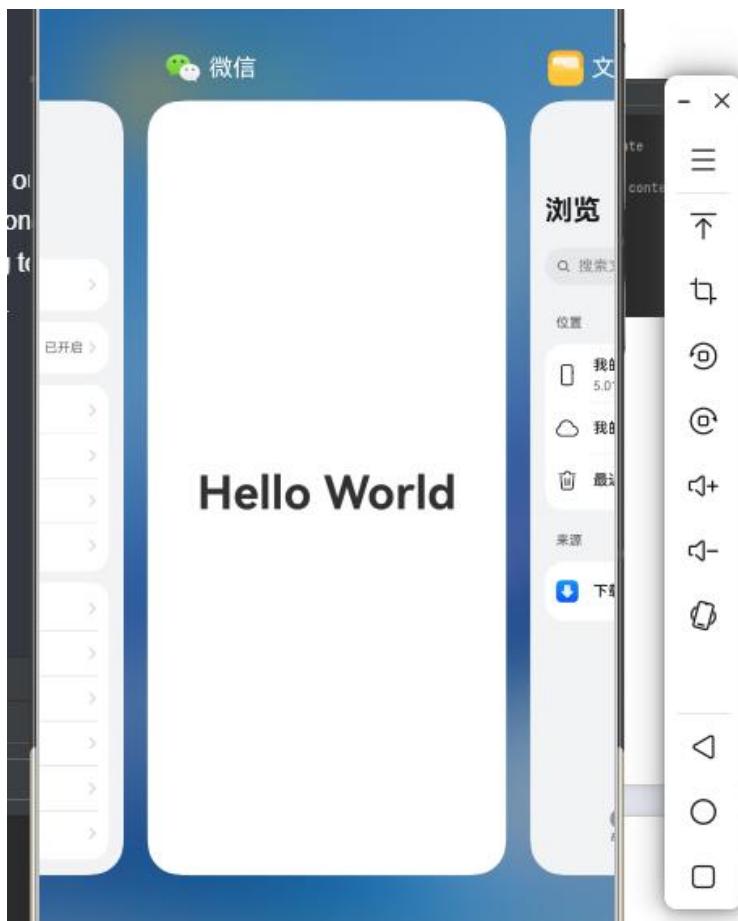
上面的 log 也说明我们已经把这个应用切到后台，可以看到，onBackground 函数被调用了。

打开回到前台再放回后台，来回操作两次：



Date	Time	Process ID	Log Tag	Log Content
10-03	10:21:44.505	4154-4154	A00000/testTag	pid-4154 I Ability onCreate
10-03	10:21:44.537	4154-4154	A00000/testTag	pid-4154 I Ability onWindowStageCreate
10-03	10:21:44.561	4154-4154	A00000/testTag	pid-4154 I Ability onForeground
10-03	10:21:44.701	4154-4154	A00000/testTag	com.examp...litydemo I Succeeded in loading the content.
10-03	10:27:23.650	4154-4154	A00000/testTag	com.examp...litydemo I Ability onBackground
10-03	10:28:27.142	4154-4154	A00000/testTag	com.examp...litydemo I Ability onForeground
10-03	10:28:37.056	4154-4154	A00000/testTag	com.examp...litydemo I Ability onBackground
10-03	10:28:40.097	4154-4154	A00000/testTag	com.examp...litydemo I Ability onForeground
10-03	10:28:41.820	4154-4154	A00000/testTag	com.examp...litydemo I Ability onBackground

让窗口处于这样的状态，然后选择另外一个应用放到前台：



可以看到，此时 onBackground 也会被调用。请自行尝试。

在上图的状态下，向上快速滑动我们的应用，也就是杀掉我们的应用的进程，此时得到：

Date	Time	Process ID	Log Tag	Log Level	Log Message
10-03	10:42:17.232	8543-8543	A00000/testTag	com.examp...litydemo	I Ability onCreate
10-03	10:42:17.272	8543-8543	A00000/testTag	com.examp...litydemo	I Ability onWindowStageCreate
10-03	10:42:17.301	8543-8543	A00000/testTag	com.examp...litydemo	I Ability onForeground
10-03	10:42:17.376	8543-8543	A00000/testTag	com.examp...litydemo	I Succeeded in loading the content.
10-03	10:42:30.811	8543-8543	A00000/testTag	com.examp...litydemo	I Ability onBackground
10-03	10:42:30.851	8543-8543	A00000/testTag	com.examp...litydemo	I Ability onWindowStageDestroy
10-03	10:42:30.851	8543-8543	A00000/testTag	com.examp...litydemo	I Ability onDestroy

可以看到，先放到后台，然后调用 `onWindowStageDestroy`，然后 `onDestroy` 被调用。

通过查看官方的实例，可以看到还有很多的不同的场景。这里我们只介绍了最基本的使用方法。后续还需要同学们自己去探索。

五、实验注意事项

1. 注意教师的操作演示。
2. 学生机与教师机内网连通，能接收和提交实验结果。
3. 按实验要求输入测试数据，并查看输出结果是否符合实验结果。

六、思考题

1. 通过这个实验，你学到了什么？