实验十六　ArkUI 页面路由基础

一、实验目的

1. 了解 DevEco Studio 的使用
2. 学习 ArkTS 语言及 ArkUI 路由基础
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、实验步骤

@ohos.router 以及组件导航（Navigation）官方文档参考：
https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/arkts-navigation-navigation-V5

HMRouter 文档参考：
https://developer.huawei.com/consumer/cn/doc/best-practices-V5/bpta-hmrouter-V5

我们尝试三种导航的方案：

- @ohos.router 方案
- Navigation 组件
- HMRouter 方案

重点是后面两种，但是建议@ohos.router 也要掌握，虽然官方不推荐使用，但是很多实例的代码依然是使用这一种的。

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

设置项目名称为 NavigationDemo。

2. @ohos.router 方案

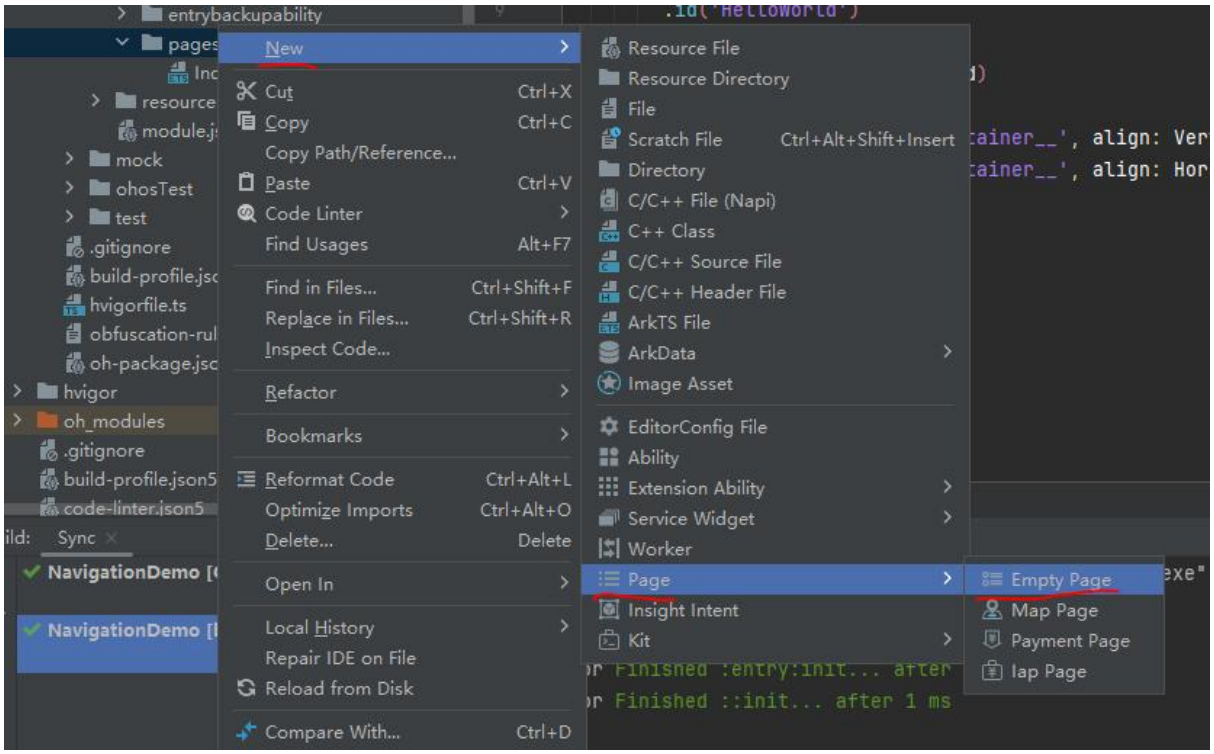
理论讲解部分参考：

https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/arkts-routing-V5

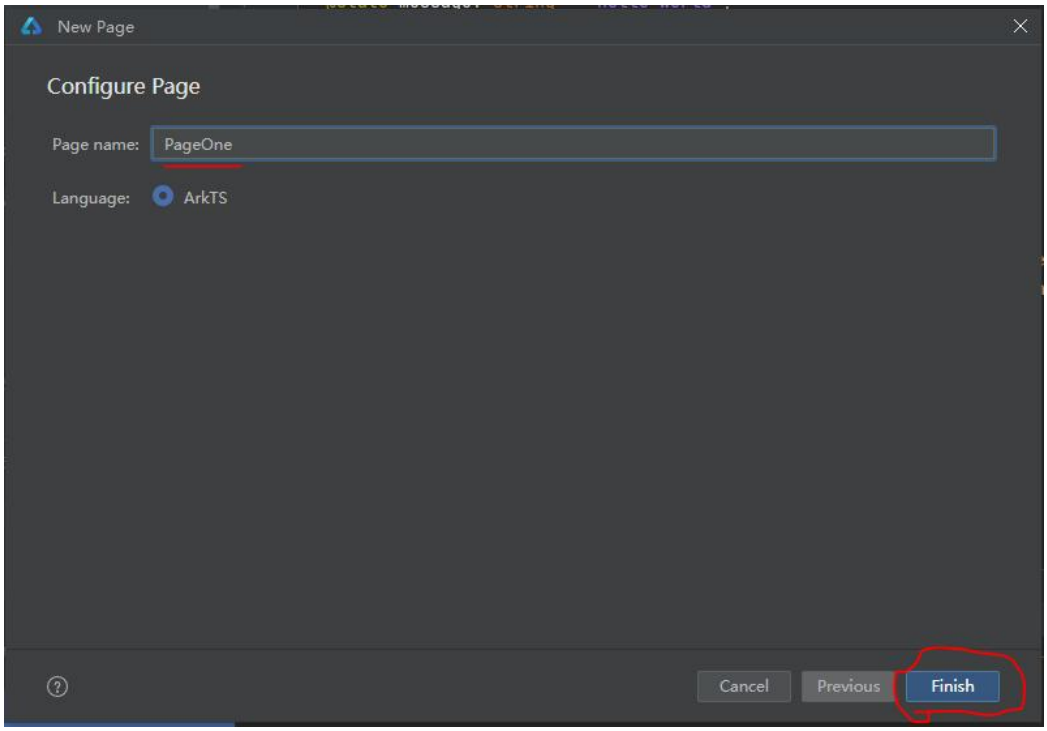
页面路由指的是在应用程序中实现不同页面之间的跳转，以及跳转过程中的数据传递。

- **创建页面**

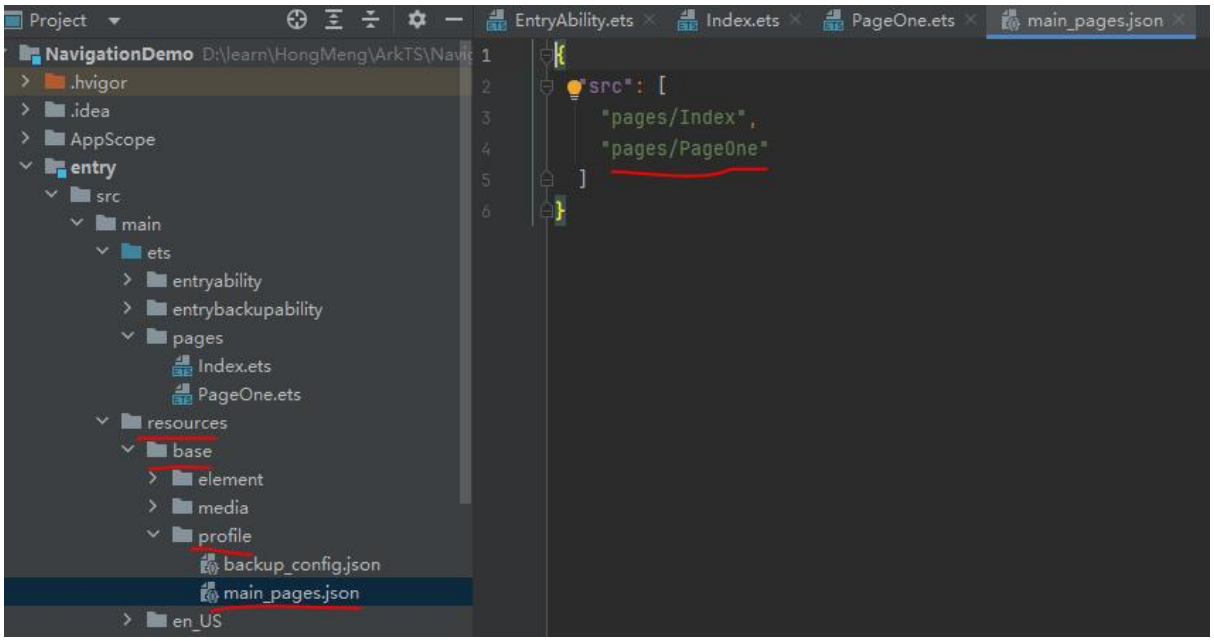
创建一个新的页面有两种方式。第一种是直接鼠标右键选择新建 Page：



在弹出的窗口中，输入页面的名称，比如 PageOne:

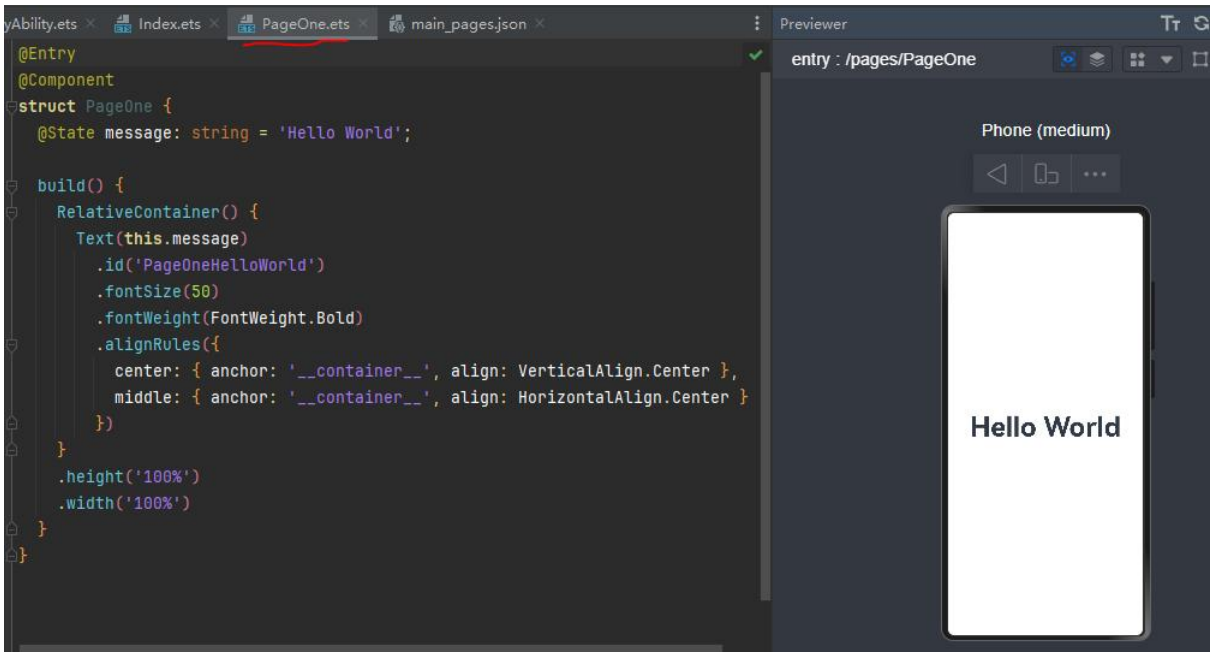


点击 Finish 之后，此时如果我们到 resources > base > profile 目录下的 main_pages.json 文件中查看，可以看到：

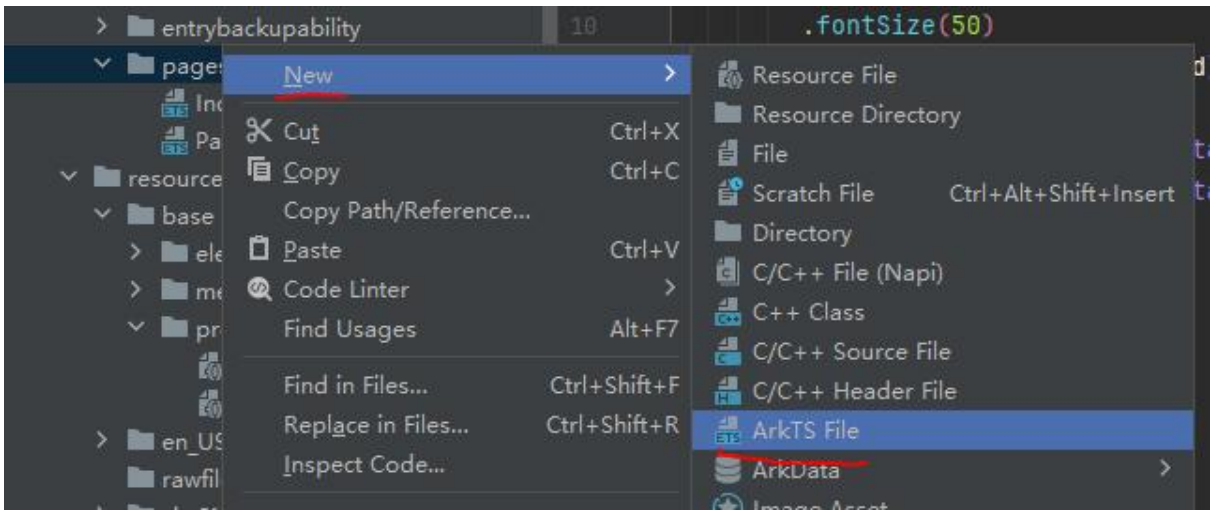


系统自动添加了一行“pages/PageOne”，也就是说通过这种方式，系统会自动在这个文件中把页面的基本信息加入进去。

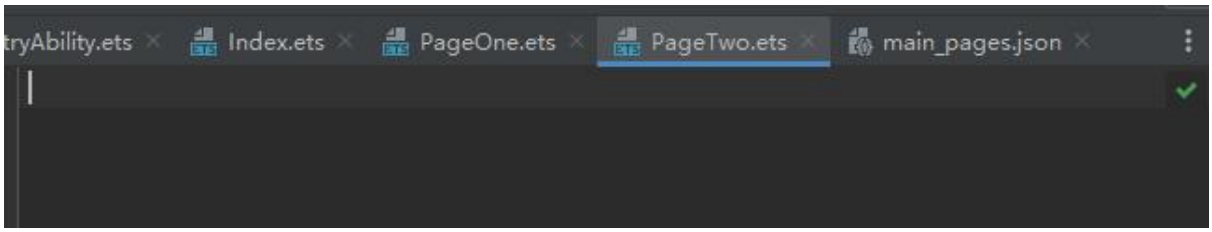
同时，我们看到，PageOne 的基础代码和 Index.ets 的一样：



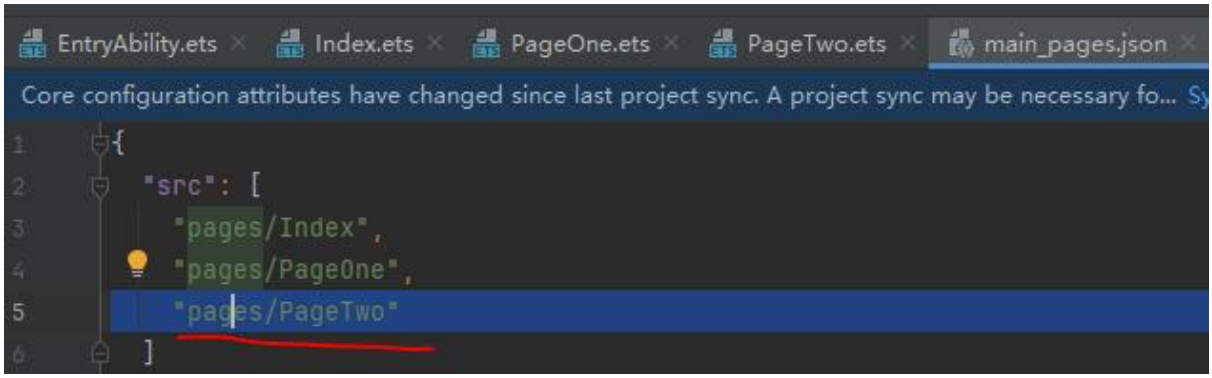
当然，另外一种方法就是我们之前常用的方法，创建一个新的 ArkTS 文件：



比如，我们创建 PageTwo.ets，此时，文件的内容是空的：



如果我们需要加页面导航，需要到 main_pages.json 文件中手动添加，一开始是没有的。我们可以自行加上：



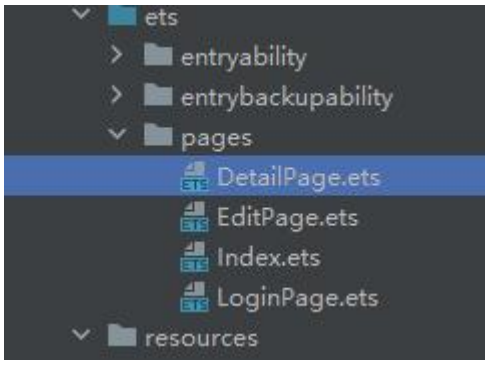
注意，这个“pages/PageTwo”是我们自己手动添加的。

- **页面跳转**

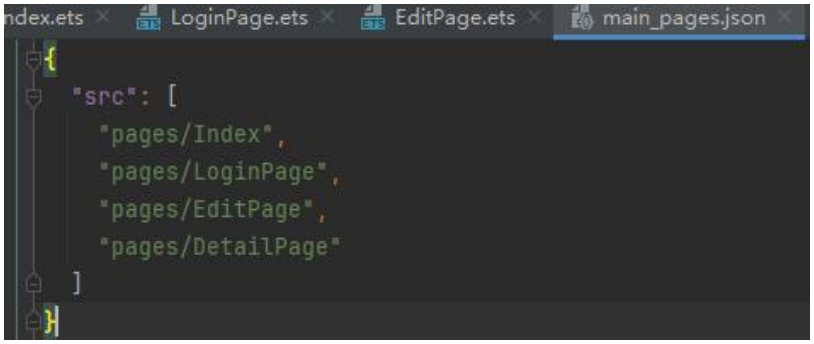
@ohos.router 里面页面跳转的方法主要有三种：

- router.pushUrl：普通跳转，可以返回
- router.replaceUrl：替换跳转，不能返回
- router.back：返回

我们先通过上面的创建页面的方法（选择你喜欢的方式，建议用新建 page 的方式），创建几个页面：



注意，确保在 main_pages.json 中，这几个页面都有：



在 LoginPage 中添加基础代码：

```
import { router } from '@kit.ArkUI'


@Entry
@Component
struct LoginPage {
  build() {
    Column() {
      Text('登录页')
        .fontSize(80)
      Button('登录到首页').onClick(() => {
        router.pushUrl({
          url: 'pages/Index'
        })
      })
    }
  }
}
```

注意，这里

```
import { router } from '@kit.ArkUI'
```

也可以写为：

```
import router from '@ohos.router'
```

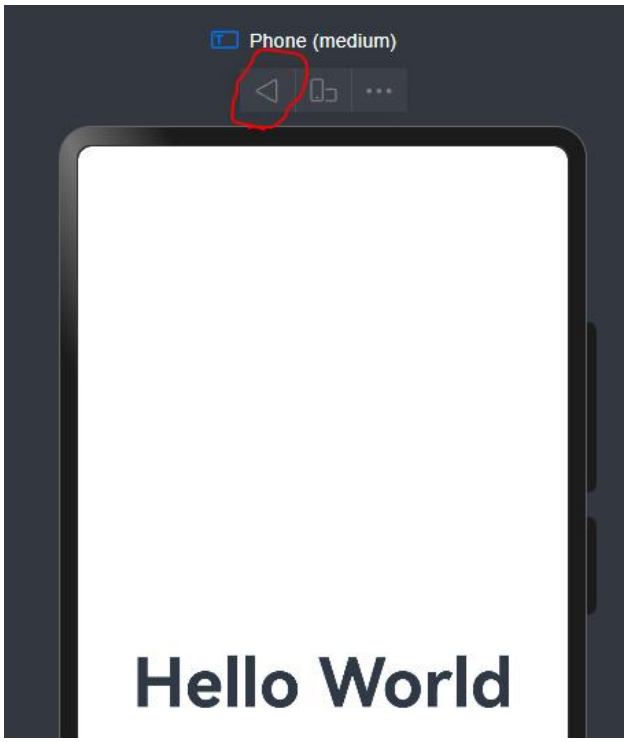
效果是类似的。

两种路由方式的核心区别，为了让你一目了然，我将它们的主要差异整理成了下面的表格：

特性维度	@ohos.router（系统原生路由）	@kit.ArkUI（ArkUI 开发工具包路由）
来源与定位	鸿蒙操作系统底层提供的原生路由模块，API 稳定。	属于 ArkUI 开发工具包的一部分，更侧重于声明式开发范式。
典型应用场景	适用于系统级应用开发，支持跨设备场景。	更适合开发轻量级前端应用，主要在单一设备上运行。
路由方式特点	提供 pushUrl(压栈)和 replaceUrl(替换)等多种跳转模式。	其 API 风格更接近 Web 前端（如 Vue Router）。
技术架构背景	代表鸿蒙早期 API 的风格。	代表鸿蒙向声明式 UI 和更现代开发体验的演进。
简单来说，@ohos.router 是鸿蒙系统的"原生基础路由"，而 @kit.ArkUI 中的 router 则可以理解为在鸿蒙新的声明式 UI 框架下，一个更现代化、对前端开发者更友好的路由工具。		

此时的 Previewer 效果：

点击按钮，会跳转到 Index.ets 页面。在显示 Index.ets 页面的时候，点击预览器上的返回键（红色圈标记出来的按钮），又会回到登录页：



通常情况下，登录之后，跳转到首页，此时无需再回到登录页了。我们可以用 replaceUrl 来替换掉上面的 pushUrl:

```
Text('登录页')
  .fontSize(80)
Button('登录到首页').onClick(() => {
  router.replaceUrl({
    url: 'pages/Index'
  })
})
}
```

刷新 Previewer 之后，我们可以看到，登录页面上点击按钮跳转到首页之后，再点击返回键，无法返回到登录页。请自行尝试。

把首页的代码改为：

```
import { router } from '@kit.ArkUI';


@Entry
```

```
@Component
struct Index {
  build() {
    Column({space: 10}) {
      Text('首页')
        .fontSize(60)
      Button('去编辑页')
        .onClick(() => {
          router.pushUrl({
            url: 'pages/EditPage'
          })
        })
      Button('去详情页')
        .onClick(() => {
          router.pushUrl({
            url: 'pages/DetailPage'
          })
        })
    }
  }
}
```

相应地，在 EditPage 和 DetailPage 里面，把“Hello World”替换为：

```
struct EditPage {
  @State message: string = '编辑页';
```

```
struct DetailPage {
  @State message: string = '详情页';
```

效果：

可以点击按钮跳转进入指定的页面，在具体页面也可以点击预览器的返回按钮返回首页，因为我们用的是 pushUrl 函数。

- **页面栈**

使用@ohos.router 的时候，其实是用到了页面栈的概念。页面栈是用来存储程序运行时页面信息的一种数据结构，遵循先进后出的原则，最大的容量为 32 个页面。

使用 pushUrl 的时候，是把一个页面推入了页面栈，而使用 replaceUrl 则是把页面栈中的当前页面数据给替换掉，所以用 back 无法返回。

可以通过两个函数演示页面栈相关的能力：

- router.getLength(): 获取页面栈的长度

-router.clear(): 清空页面栈

把 EditPage.ets 代码改为：

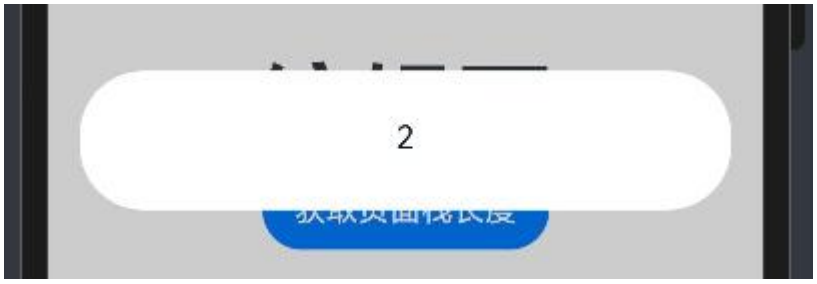
```
import { router } from '@kit.ArkUI';
```

```
@Entry
@Component
struct EditPage {
  @State message: string = '编辑页';

  build() {
    Row() {
      Column() {
        Text(this.message)
          .fontSize(50)
          .fontWeight(FontWeight.Bold)
        Button('获取页面栈长度')
          .onClick(() => {
            AlertDialog.show({
              message: router.getLength()
            })
          })
      }
      .width('100%')
    }
    .height('100%')
```

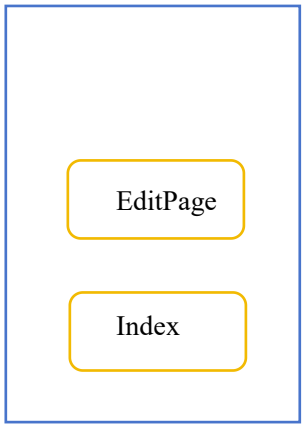
```
  }
}
```

效果：

回到 Index.ets，刷新 Previewer，然后点击“去编辑页”，到了编辑页之后，点击这个“获取页面栈长度”按钮：



也就是说，此时页面栈内有两个页面的信息：



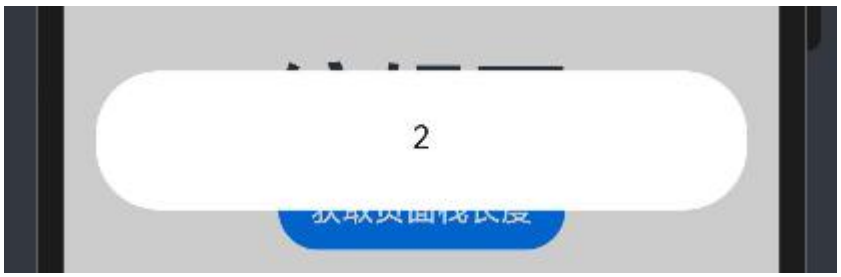
此时，如果我们把 LoginPage 中的 replaceUrl 改为 pushUrl:

```
Button('登录到首页').onClick(() => {
  router.pushUrl({
    url: 'pages/Index'
  })
```

再刷新 LoginPage，然后点击按钮，一直跳转到 EditPage，然后再获取长度，此时：

可以设想一下，在页面栈里面，最上面是 EditPage，然后是 Index，最下面是 LoginPage。

把 LoginPage 里面的 pushUrl 再改为 replaceUrl，刷新后跳转到 EditPage，再获取长度：



请思考为什么是 2。

我们再给 EditPage 中加一个按钮“清空页面栈”：

```
Button('获取页面栈长度')
  .onClick(() => {
    AlertDialog.show({
      message: router.getLength()
    })
  })
Button('清空页面栈')
  .onClick(() => {
    router.clear()
  })
}
.width('100%')
```

效果：

我们再从 Index 页面跳转过来后，点击长度：

再点击“清空页面栈”，然后再获取长度：

此时，页面栈中只有当前页面这一项了。此时再点击回退按钮，无法返回到 Index。请自行尝试。

我们给每一个页面都加上这个“获取页面栈长度的按钮”。请自己添加代码。

在 EditPage 中加入一个新的按钮“跳转到详情页”：

```
import { router } from '@kit.ArkUI';


@Entry
@Component
struct EditPage {
```

```
  @State message: string = '编辑页';


  build() {
    Row() {
      Column({space: 10}) {
        Text(this.message)
          .fontSize(50)
          .fontWeight(FontWeight.Bold)
        Button('跳转到详情页')
          .onClick(() => {
            router.pushUrl({
              url: 'pages/DetailPage'
            })
          })
        Button('获取页面栈长度')
          .onClick(() => {
            AlertDialog.show({
              message: router.getLength()
            })
          })
        Button('清空页面栈')
          .onClick(() => {
```

```
                router.clear()
            })
        }
        .width('100%')
    }
    .height('100%')

  }
}
```

在 DetailPage 详情页中添加一个按钮“跳转到编辑页”：

```
import { router } from '@kit.ArkUI';

@Entry
@Component
struct DetailPage {
  @State message: string = '详情页';

  build() {
    Row() {
      Column({space: 10}) {
        Text(this.message)
          .fontSize(50)
```

```
            .fontWeight(FontWeight.Bold)
        Button('跳转到编辑页')
            .onClick(() => {
              router.pushUrl({
                url: 'pages/EditPage'
              })
            })
        Button('获取页面栈长度')
            .onClick(() => {
              AlertDialog.show({
                message: router.getLength()
              })
            })
      }
      .width('100%')
    }
    .height('100%')
  }
}
```

然后，我们从 Index 首页出发，跳转到编辑页，再跳转到详情页，再跳转到编辑页，再跳转到详情页，此时，点击详情页上的获取长度按钮：

和我一样无聊的同学可以一直这样循环下去，最后发现跳转不动了，此时点击获取长度：

因为此时已经到达了页面栈长度的极限了：32。

如果我们把 EditPage 和 DetailPage 中跳转到对方页面的 pushUrl 改为 replaceUrl:

```
Button('跳转到详情页')
  .onClick(() => {
    router.replaceUrl({
      url: 'pages/DetailPage'
    })
  })
Button('获取页面栈长度')
```

```
Button('跳转到编辑页')
  .onClick(() => {
    router.replaceUrl({
      url: 'pages/EditPage'
    })
  })
```

再从 Index 首页出发，来回在编辑页和详情页之间跳转，获取长度：

结果始终是 2。请自行尝试并思考为什么。

- **路由模式**

使用@ohos.router 的 pushUrl 时候，提供了两种不同的跳转模式：

- Standard: 默认缺省的模式，无论页面之前是否在页面栈中添加过，一直添加到页面栈，除非页面栈满了（回忆一下上面的 32 的场景）

- Single: 如果目标页面在页面栈中已经存在，会将已有的最近的相同的 Url 页面移动到栈的顶部

调用的方式：

router.pushUrl(options, mode)

通过设置 mode 为 Standard（其实不用设置，因为是默认的）或 Single，可以控制一下栈内重复的页面 Url。

回到上面的例子，我们把 EditPage 和 DetailPage 里面的跳转加上 Single 模式的设置，比如在 EditPage 中：

```
Button('跳转到详情页')
  .onClick(() => {
    router.pushUrl({
      url: 'pages/DetailPage'
    }, router.RouterMode.Single)
  })
```

红色字体为新加的部分。另一个文件请自行修改。

再从 Index 首页开始，不断循环在编辑页和详情页之间跳转，点击获取页面栈长度：

始终是 3。请自行尝试并思考为什么。

- **路由传参**

页面跳转的过程中，参数传递非常重要。比如大家都是从登录页跳转到首页上，但是不同的人登录的信息不同，从而看到的首页也是不同的。在一个页面上，点击不同的商品，需要跳转到不同的对应详情页面，这些过程都需要在页面跳转中传递参数。

基本的语法：

跳转触发页面：

```
router.pushUrl({
    url: 'pages/DetailPage',
    Params: {
        // 以对象的形式传递参数
    }
})
```

而在跳转到的目标页面，需要在页面刚启动的函数（后面会详细讲生命周期函数）中获取参数：

```
aboutToAppear(): void {
    // 通过 as 类型断言转为具体的类型
    const params = router.getParams() as 类型
    // 后续通过点语法即可取值
    params.xxx
}
```

假设我们从登录页传参到首页。我们在登录页上加一个 TextInput 来输入用户名，完整代码：

```
import router from '@ohos.router'


@Entry
```

```
@Component
struct LoginPage {
  @State username: string = ''

  build() {
    Column({space: 10}) {
      Text('登录页')
        .fontSize(80)
      TextInput({
        text: $$this.username,
        placeholder: "请输入用户名"
      })
      Button('登录到首页').onClick(() => {
        router.pushUrl({
          url: 'pages/Index',
          params: {
            username: this.username,
            msg: '登录用户名'
          }
        })
      })
      Button('获取页面栈长度')
```

```
            .onClick(() => {
              AlertDialog.show({
                message: router.getLength()
              })
            })
        }
    }
}
```

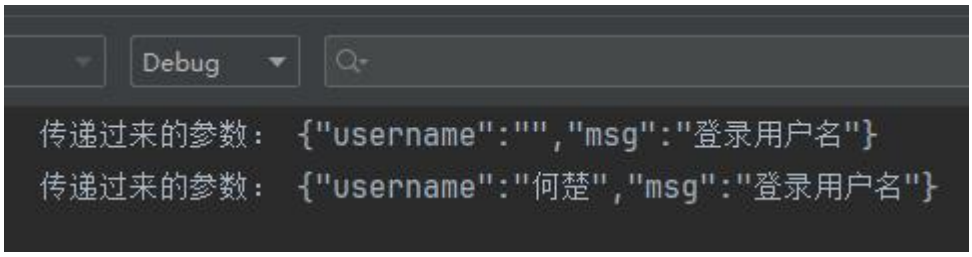
红色字体为新加的代码。效果：

点击“登录到首页”，可以正常跳转。但是此时首页上看不到任何效果。

修改 Index.ets 首页代码，在 build()前面加上代码：

```
  // aboutToAppear 是一进入页面就会执行的函数
  aboutToAppear(): void {
    console.log('传递过来的参数：', JSON.stringify(router.getParams()))
  }
```

此时回到 LoginPage，刷新 Previewer，在用户名区域输入自己的名字，点击跳转到首页，在 Log 里面可以看到：



参数被正确传递了。

为了获取参数并使用，我们需要定义一下参数的类型，在 Index.ets 文件的@Entry 上面定义一个 interface:

```
interface paramsGot {
  username: string,
  msg: string
}
```

然后在 aboutToAppear 里面用 as 类型断言去获取参数，就是加上红色字体那句：

```
  aboutToAppear(): void {
    console.log('传递过来的参数：', JSON.stringify(router.getParams()))
    const params = router.getParams() as paramsGot
  }
```

然后，我们定义一个状态变量，通过赋值的方式获取参数中的值：

```
interface paramsGot {
  username: string,
  msg: string
}

@Entry
@Component
struct Index {
  @State username: string = ''
  @State message: string = ''

  // aboutToAppear是一进入页面就会执行的函数
  aboutToAppear(): void {
    console.log('传递过来的参数: ', JSON.stringify(router.getParams()))
    const params = router.getParams() as paramsGot
    this.username = params.username
    this.message = params.msg
  }
```

请自己加上代码。

在 Index 的页面上用上接收到的参数：

```
build() {
  Column({space: 10}) {
    Text(this.username + '首页')
      .fontSize(60)
    Text(this.message)
      .fontSize(20)
    Button('去编辑页')
      .onClick(() => {
```

请自己加上代码。

效果，如果用户名输入为 LiSi：

可以看到，参数接收到之后，赋值给状态变量，然后被正确地渲染到页面上了。

3. 组件导航（Navigation）方案

理论讲解部分参考：

https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/arkts-navigation-navigation-V5

Navigation 是路由容器组件，一般作为首页的根容器，包括单栏(Stack)、分栏(Split)和自适应(Auto)三种显示模式。Navigation 组件适用于模块内和跨模块的路由切换，一次开发，多端部署场景。通过组件级路由能力实现更加自然流畅的转场体验，并提供多种标题栏样式来呈现更好的标题和内容联动效果。在不同尺寸的设备上，Navigation 组件能够自适应显示大小，自动切换分栏展示效果。

Navigation 组件主要包含　导航页(NavBar)和子页(NavDestination)。导航页由标题栏(Titlebar，包含菜单栏 menu)、内容区(Navigation 子组件)和工具栏(Toolbar)组成，其中导航页可以通过 hideNavBar 属性进行隐藏，导航页不存在页面栈中，导航页和子页，以及子页之间可以通过路由操作进行切换。

在 API Version 9 上，需要配合 NavRouter 组件实现页面路由，从 API Version 10 开始，推荐使用 NavPathStack 实现页面路由。

对于页面显示模式，请直接参考上面链接中的示意图。

我们先通过一个实例来看它的导航功能。

创建一个新的项目 NavigationDemo0，在 Index.ets 中加入基础代码：

```
@Entry
@Component
struct Index {
  pathInfos: NavPathStack = new NavPathStack()
  @State message: string = 'First Page, Click Me';

  build() {
    Column() {
      Navigation(this.pathInfos) {
        Column({space: 10}) {
          Text(this.message)
            .fontSize(16)
            .fontWeight(FontWeight.Bold)
          Button('Go to page 2')
            .onClick(() => {
              this.pathInfos.pushPathByName('second page', 'params from 1st Page')
            })
        }
      }
      .width('100%')
```

```
        .mode(NavigationMode.Auto)
        .title('First Page')
    }
    .height('100%')
    .width('100%')
  }
}
```

注意这里的红色字体部分，我们需要先创建一个 NavPathStack 的实例 pathInfos，然后对于页面的内容，我们使用 Navigation 容器组件把所有的内容都包起来 Navigation(this.pathInfos)。

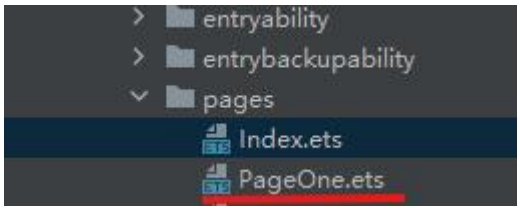
然后，在其中需要导航的地方，比如按钮，在 onClick 事件中，将路由 push 到 NavPathStack 之中：

```
this.pathInfos.pushPathByName('second page', 'params from 1st Page')
```

其中，路由名称为“second page”，参数为 params from 1st Page。

Navigation 容器组件的 mode 我们这里设为 Auto，也给了一个 title：First Page。

再创建第二个文件 PageOne.ets：（注意，这个不是一个页面，而是一个组件文件）



加入代码：

```
@Builder
export function PageOneBuilder(name: string, param: string) {
  PageOne({name: name, value: param})
}
```

```
@Component
export struct PageOne {
  pathInfos: NavPathStack = new NavPathStack()
  name: string = '';
  @State value: string = ''

  build() {
    NavDestination() {
      Column({space: 10}) {
        Text('Second Page')
          .width('100%')
          .fontSize(20)
        Text(`The param from previous page: ${this.value}`)
          .width('100%')
          .fontSize(20)
        Button('return')
          .width('40%')
          .height(40)
          .margin({top: 30})
          .onClick(() => {
            this.pathInfos.pop();
```

```
            })
        }
        .size({width: '100%', height: '100%'})
    }.title(`${this.name}`)
    .onReady((ctx: NavDestinationContext) => {
        this.pathInfos = ctx.pathStack
    })
  }
}
```

这里，首先必须要使用@Builder 创建一个 PageOneBuilder 函数，里面就是调用 PageOne 的结构体的构造函数。这个在后续的配置中需要用到。

然后，对于 PageOne 结构体，整体就用 NavNavigation 包起来，然后在它的 onReady 生命周期函数中，将 NavDestinationContext 中的 pathInfos 赋值给本地的 pathInfos，这样第一个页面的 pathInfos 和这个组件的 pathInfos 就同步了。

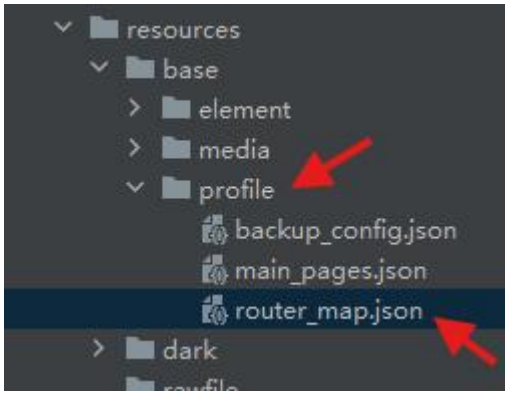
对于返回按钮，调用 pop 函数：

```
this.pathInfos.pop();
```

然后，关键的一步，我们需要配置路由表。首先，在 module.json5 文件中加入：

```
"routerMap": "$profile:router_map",
```

然后，在 resources > base > profile 目录下创建一个新的文件 router_map.json:



在 router_map.json 中加入：

```
{
  "routerMap" : [
    {
      "name" : "second page",
      "pageSourceFile"   : "src/main/ets/pages/PageOne.ets",
      "buildFunction" : "PageOneBuilder"
    }
  ]
}
```

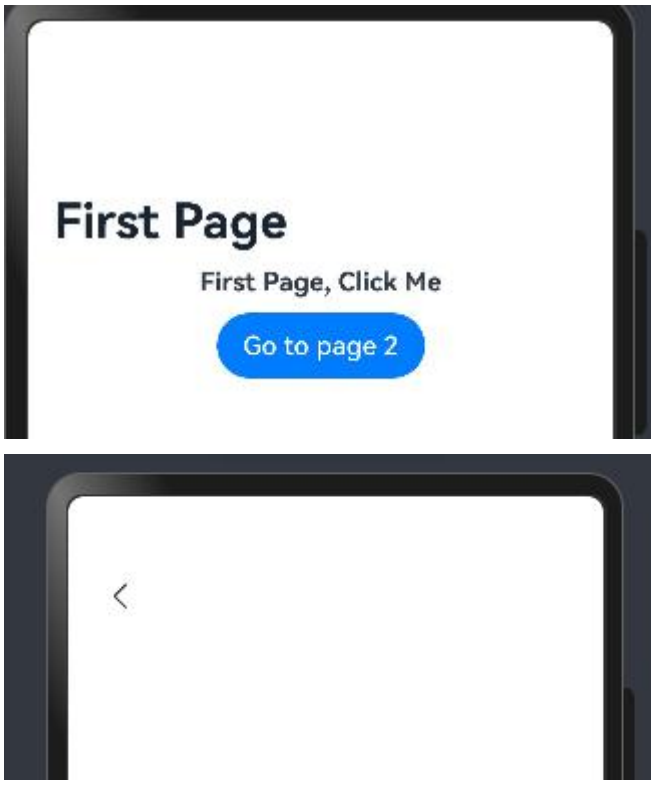
这里 name 对应的是下面的函数中的第一个参数：

this.pathInfos.pushPathByName('second page', 'params from 1st Page')

而 pageSourceFile 对应的是我们创建的 PageOne.ets。

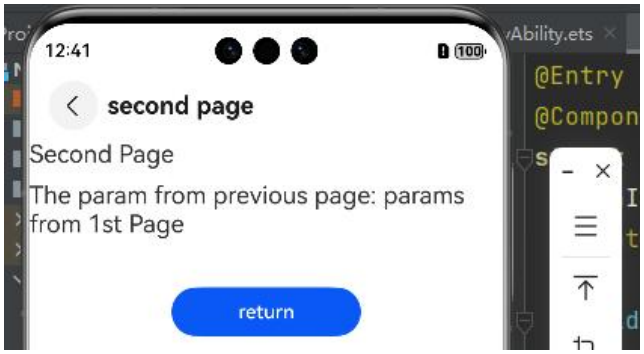
然后 buildFunction 对应的就是 PageOne.ets 中的那个@Builder 函数。

在 Previewer 中看效果：



我们发现在预览器中第二页不显示内容。

改到模拟器中运行：



点击 return 或左上角的 < 返回按钮，都可以返回到第一页。

我们再加一个新的文件 PageTwo.ets，加入代码：

```
@Builder
export function PageTwoBuilder(name: string, param: string) {
  PageTwo({name: name, value: param})
}

@Component
export struct PageTwo {
  pathInfos: NavPathStack = new NavPathStack()
  name: string = '';
  @State value: string = ''

  build() {
    NavDestination() {
      Column({space: 10}) {
        Text('Second Page')
          .width('100%')
          .fontSize(20)
        Text(`The param from previous page: ${this.value}`)
          .width('100%')
          .fontSize(20)
```

```
            Button('return')
                .width('40%')
                .height(40)
                .margin({top: 30})
                .onClick(() => {
                    this.pathInfos.pop();
                })
        }
        .size({width: '100%', height: '100%'})
    }.title(`${this.name}`)
    .onReady((ctx: NavDestinationContext) => {
        this.pathInfos = ctx.pathStack
    })
  }
}
```

在 Index.ets 中加入一个按钮，点击导航到第三个页面：

```
            Button('Go to page 3')
                .onClick(()=>{
                    this.pathInfos.pushPathByName('third page', 'params from 1st page')
                })
```

加在哪里请同学们自己找一下。

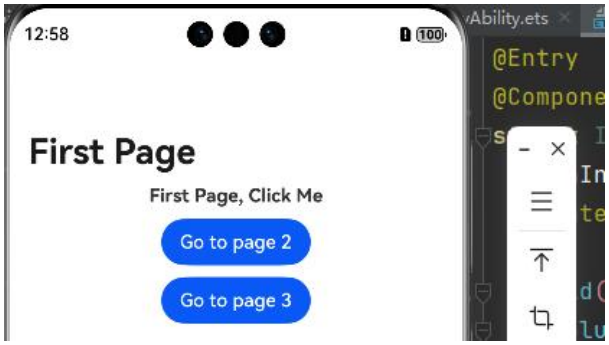
在 PageOne.ets 中，加入一个按钮，点击从第二页导航到第三页：

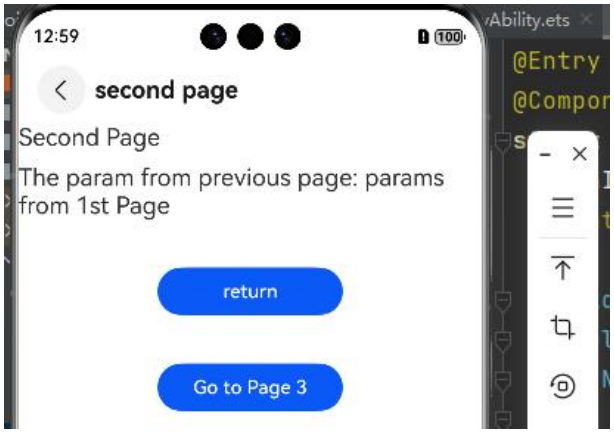
```
        Button('Go to Page 3')
          .width('40%')
          .height(40)
          .margin({top: 30})
          .onClick(() => {
            this.pathInfos.pushPathByName('third page', 'params from 2nd page');
          })
```

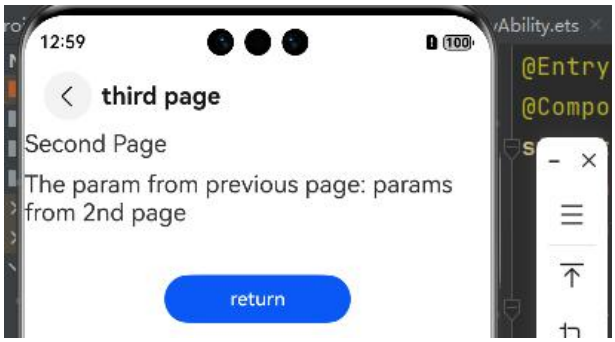
然后，很关键的步骤，需要在 router_map.json 中加入新的路由：

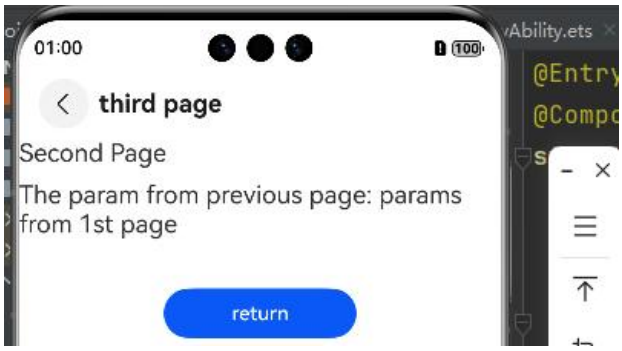
```
{
  "routerMap" : [
    {
      "name" : "second page",
      "pageSourceFile"  : "src/main/ets/pages/PageOne.ets",
      "buildFunction" : "PageOneBuilder"
    },
    {
      "name" : "third page",
      "pageSourceFile"  : "src/main/ets/pages/PageTwo.ets",
      "buildFunction" : "PageTwoBuilder"
    }
  ]
}
```

在模拟器中运行：









请注意最后两个页面的区别。

我们再来看一个经典的实例。创建一个新的项目 NavigationDemo1，在 Index.ets 中加入代码：

```
@Entry
@Component
struct NavigationDemo1 {
  pathInfos: NavPathStack = new NavPathStack()
  private listArray: Array<string> = ['WLAN', 'Bluetooth', 'Personal Hotspot', 'Connect
& Share']

  build() {
    Column() {
      Navigation(this.pathInfos) {
        TextInput({placeholder: '输入关键字搜索'})
          .width('90%')
          .height(40)
          .margin({bottom: 10})

        List({space: 12, initialIndex: 0}) {
          ForEach(this.listArray, (item: string) => {
            ListItem() {
              Row() {
                Row() {
                  Text(`${item.slice(0, 1)}`)
                    .fontColor(Color.White)
```

```
                .fontSize(14)
                .fontWeight(FontWeight.Bold)
            }
            .width(30)
            .height(30)
            .backgroundColor('#a8a8a8')
            .margin({right: 20})
            .borderRadius(20)
            .justifyContent(FlexAlign.Center)

            Column() {
              Text(item)
                .fontSize(16)
                .margin({bottom: 5})
            }
            .alignItems(HorizontalAlign.Start)

            Blank()

            Row()
              .width(12)
              .height(12)
```

```
                .margin({right: 15})
                .border({
                  width: {top: 2, right: 2},
                  color: 0xcccccc
                })
                .rotate({angle: 45})
            }
            .borderRadius(15)
            .shadow({radius:100, color: '#ededed'})
            .width('90%')
            .alignItems(VerticalAlign.Center)
            .padding({left:15, top: 15, bottom: 15})
            .backgroundColor(Color.White)
        }
        .width('100%')
        .onClick(()=> {
          this.pathInfos.pushPathByName(`${item}`, '详情页面参数')
        })
      }, (item: string): string => item)
    }
    .listDirection(Axis.Vertical)
    .edgeEffect(EdgeEffect.Spring)
```

```
            .sticky(StickyStyle.Header)
            .chainAnimation(false)
            .width('100%')
        }
        .width('100%')
        .mode(NavigationMode.Auto)
        .title('设置')
    }
    .size({width: '100%', height: '100%'})
    .backgroundColor(0xf4f4f5)
  }
}
```

注意上面的红色字体的代码。

创建 PageOne.ets，加入代码：

```
@Builder
export function PageOneBuilder(name: string, param: string) {
  PageOne({name: name, value: param});
}

@Component
export struct PageOne {
```

```
  pathInfos: NavPathStack = new NavPathStack();
  name:string = '';
  @State value: string = '';

  build() {
    NavDestination() {
      Column() {
        Text(`${this.name}设置页面`)
          .width('100%')
          .fontSize(20)
          .fontColor(0x333333)
          .textAlign(TextAlign.Center)
          .textShadow({
            radius: 2,
            offsetX: 4,
            offsetY: 4,
            color: 0x909399
          })
          .padding({ top: 30 })
        Text(`${JSON.stringify(this.value)}`)
          .width('100%')
          .fontSize(18)
```

```
                .fontColor(0x666666)
                .textAlign(TextAlign.Center)
                .padding({top: 45})
            Button('返回')
                .width('50%')
                .height(40)
                .margin({top:50})
                .onClick(()=>{
                  this.pathInfos.pop();
                })
        }
        .size({width: '100%', height: '100%'})
    }.title(`${this.name}`)
    .onReady((ctx: NavDestinationContext) => {
      this.pathInfos = ctx.pathStack;
    })
  }
}
```

创建新文件 PageTwo.ets，加入代码：

```
@Builder
export function PageTwoBuilder(name: string) {
  PageTwo({ name: name });
```

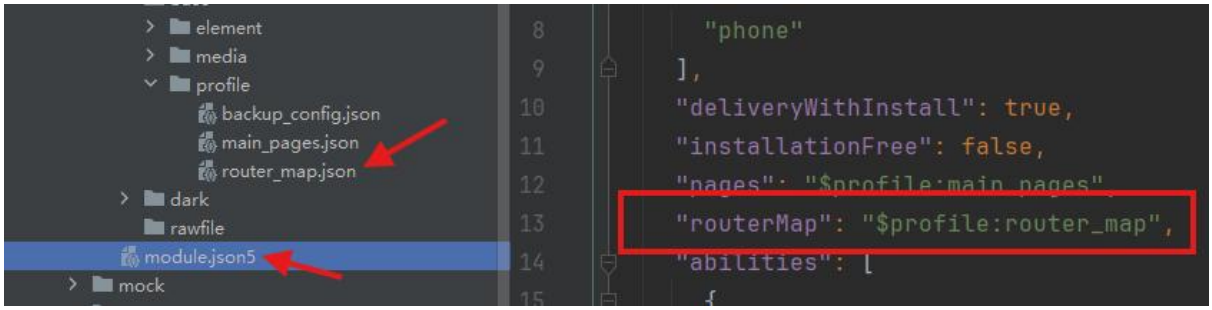
```
}

@Component
export struct PageTwo {
  pathInfos: NavPathStack = new NavPathStack();
  name: string = '';
  private listArray: Array<string> = ['Projection', 'Print', 'VPN', 'Private DNS', 'NFC'];
  build() {
    NavDestination() {
      Column() {
        List({ space: 12, initialIndex: 0 }) {
          ForEach(this.listArray, (item: string) => {
            ListItem() {
              Row() {
                Row() {
                  Text(`${item.slice(0, 1)}`)
                    .fontColor(Color.White)
                    .fontSize(14)
                    .fontWeight(FontWeight.Bold)
                }
                .width(30)
                .height(30)
```

```
                .backgroundColor('#a8a8a8')
                .margin({ right: 20 })
                .borderRadius(20)
                .justifyContent(FlexAlign.Center)
                Column() {
                  Text(item)
                    .fontSize(16)
                    .margin({ bottom: 5 })
                }
                .alignItems(HorizontalAlign.Start)
                Blank()
                Row()
                  .width(12)
                  .height(12)
                  .margin({ right: 15 })
                  .border({
                    width: { top: 2, right: 2 },
                    color: 0xcccccc
                  })
                  .rotate({ angle: 45 })
              }
              .borderRadius(15)
```

```
                    .shadow({ radius: 100, color: '#ededed' })
                    .width('90%')
                    .alignItems(VerticalAlign.Center)
                    .padding({ left: 15, top: 15, bottom: 15 })
                    .backgroundColor(Color.White)
                }
                .width('100%')
                .onClick(() => {
                    this.pathInfos.pushPathByName(`${item}`, '页面设置参数');
                })
            }, (item: string): string => item)
        }
        .listDirection(Axis.Vertical)
        .edgeEffect(EdgeEffect.Spring)
        .sticky(StickyStyle.Header)
        .width('100%')
    }
    .size({ width: '100%', height: '100%' })
}.title(`${this.name}`)
.onReady((ctx: NavDestinationContext) => {
    // NavDestinationContext 获取当前所在的导航控制器
    this.pathInfos = ctx.pathStack;
```

```
    })
  }
}
```

然后，在 module.json5 中加入：



代码：

```
    "routerMap": "$profile:router_map",
```

然后，在 resources > base > profile 目录下创建文件 router_map.json，加入代码：

```
{
  "routerMap" : [
    {
      "name" : "WLAN",
      "pageSourceFile"   : "src/main/ets/pages/PageOne.ets",
      "buildFunction" : "PageOneBuilder"
    },
    {
      "name" : "Bluetooth",
      "pageSourceFile"   : "src/main/ets/pages/PageOne.ets",
```

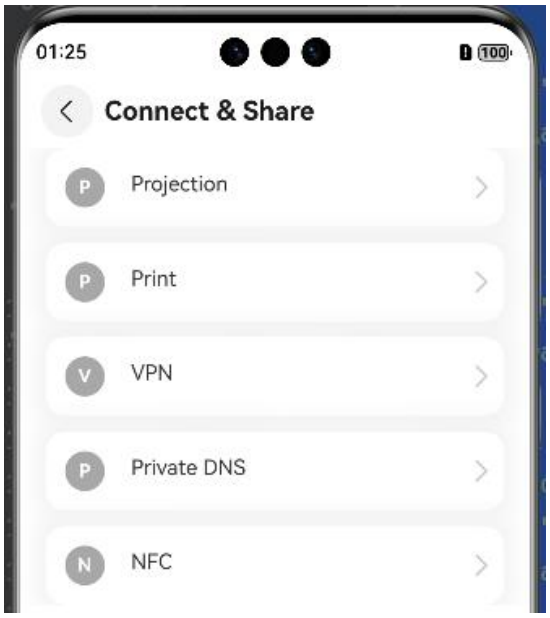
```
        "buildFunction" : "PageOneBuilder"
    },
    {
        "name" : "Personal Hotspot",
        "pageSourceFile"   : "src/main/ets/pages/PageOne.ets",
        "buildFunction" : "PageOneBuilder"
    },
    {
        "name" : "Connect & Share",
        "pageSourceFile"   : "src/main/ets/pages/PageTwo.ets",
        "buildFunction" : "PageTwoBuilder"
    },
    {
        "name" : "Projection",
        "pageSourceFile"   : "src/main/ets/pages/PageOne.ets",
        "buildFunction" : "PageOneBuilder"
    },
    {
        "name" : "Print",
        "pageSourceFile"   : "src/main/ets/pages/PageOne.ets",
        "buildFunction" : "PageOneBuilder"
    },
```

```
    {
      "name" : "VPN",
      "pageSourceFile"   : "src/main/ets/pages/PageOne.ets",
      "buildFunction" : "PageOneBuilder"
    },
    {
      "name" : "Private DNS",
      "pageSourceFile"   : "src/main/ets/pages/PageOne.ets",
      "buildFunction" : "PageOneBuilder"
    },
    {
      "name" : "NFC",
      "pageSourceFile"   : "src/main/ets/pages/PageOne.ets",
      "buildFunction" : "PageOneBuilder"
    }
  ]
}
```

注意，只有“Connect & Share”这里是 PageTwo。那么这样会是什么效果呢？在模拟器中运行，效果：


设置
输入关键字搜索
WLAN
Bluetooth
Personal Hotspot
Connect & Share


01:25
Bluetooth
Bluetooth设置页面
"详情页面参数"
返回


01:25
Connect & Share
Projection
Print
VPN
Private DNS
NFC

这样我们基本上理解了如何使用 Navigation 组件来做导航。下面我们来看一下这个组件的配置。

4. HMRouter 方案

Navigation 路由容器组件用起来比较复杂，而且和 UI 页面的耦合度比较大，容易出差错。华为鸿蒙又推出了 HMRouter 的方案：

https://gitee.com/hadss/hmrouter/blob/master/HMRouterLibrary/README.md#hmrouter

在“最佳实践”中也有一些介绍：

https://developer.huawei.com/consumer/cn/doc/best-practices/bpta-hmrouter

该框架底层对 Navigation 相关能力进行了封装，帮助开发者减少对 Navigation 相关细节内容的关注、提高开发效率，同时该框架对页面跳转能力进行了增强，例如其中的路由拦截、单例页面等。

我们创建一个新的项目 HMRouterDemo 来演示。

要使用 HMRouter，首先需要安装依赖。可以有两种方法（选一种即可）：

方法一：通过 ohpm 安装：

在 Terminal 里面，运行命令：

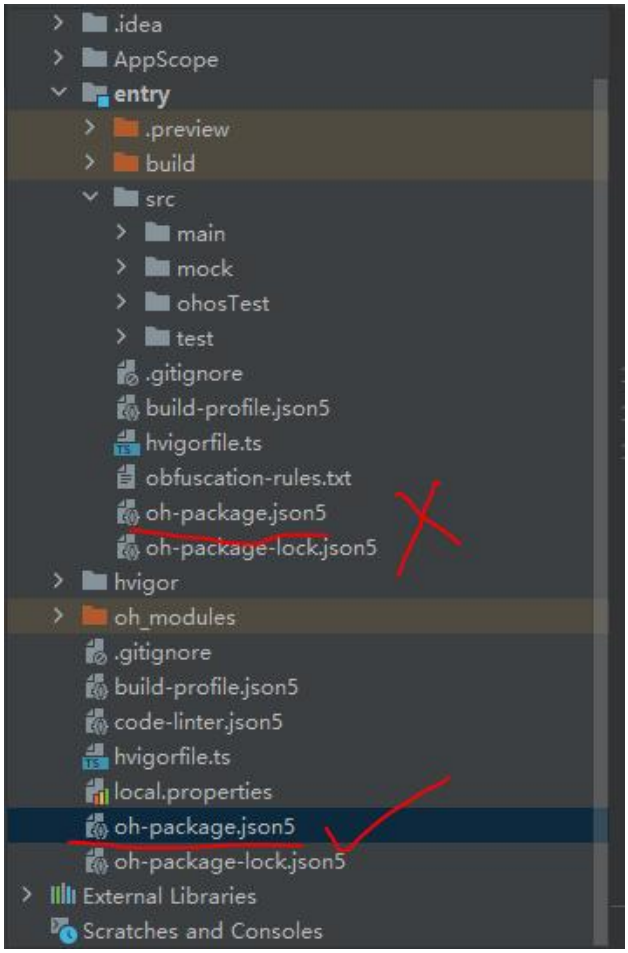
```
ohpm install @hadss/hmrouter
```

如下图所示：

```
Terminal:  Local  +
Windows PowerShell
版权所有（C） Microsoft Corporation。保留所有权利。

安装最新的 PowerShell，了解新功能和改进！ https://aka.ms/PSWindows

PS D:\learn\HongMeng\HMRouterDemo> ohpm install @hadss/hmrouter
ohpm INFO: fetch meta info of package '@hadss/hmrouter' success https://ohpm.openharmony.cn/ohpm/@hadss/hmrouter
ohpm INFO: fetch meta info of package '@ohos/hypium' success https://ohpm.openharmony.cn/ohpm/@ohos/hypium
ohpm INFO: fetch meta info of package '@ohos/hamock' success https://ohpm.openharmony.cn/ohpm/@ohos/hamock
install completed in 1s 30ms
PS D:\learn\HongMeng\HMRouterDemo>
```

方法二： 在模块中配置

在工程根目录下的 oh-package.json5 文件中添加依赖，注意是根目录下的，而不是 module 里面的：



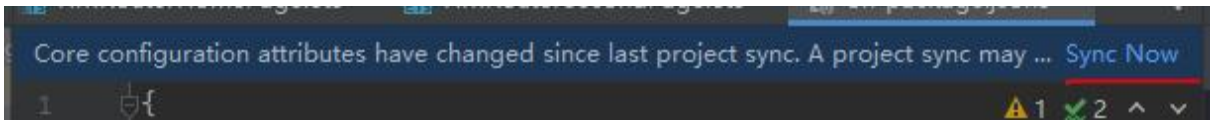
加入：

```
"dependencies": {
  "@hadss/hmrouter": "^1.2.2"
},
```

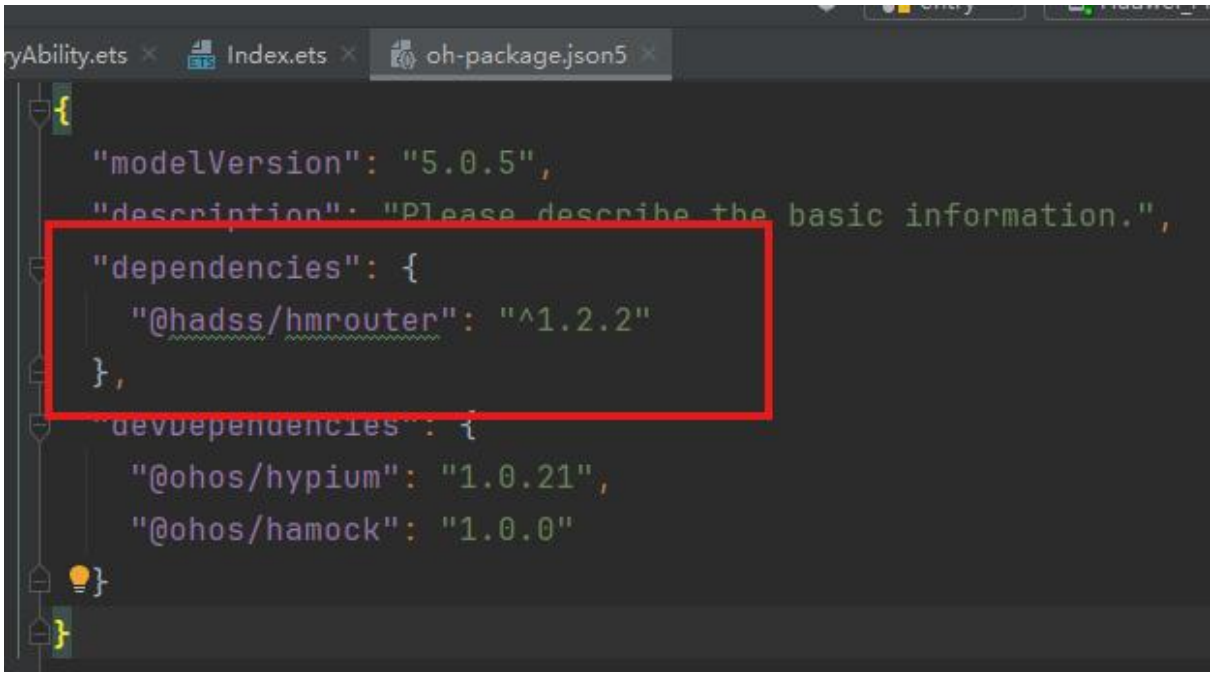
也可以写 latest：

```
"dependencies": {
  "@hadss/hmrouter": "latest"
}
```

可以直接用最新的，也就是 latest。当系统提示 sync 的时候，就点击“sync now”同步一下。



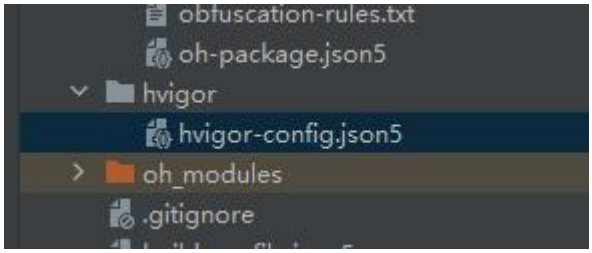
我用的是第一种方法，可以看到，安装成功之后，系统在根目录下的 oh-package.json5 中自动添加了依赖：

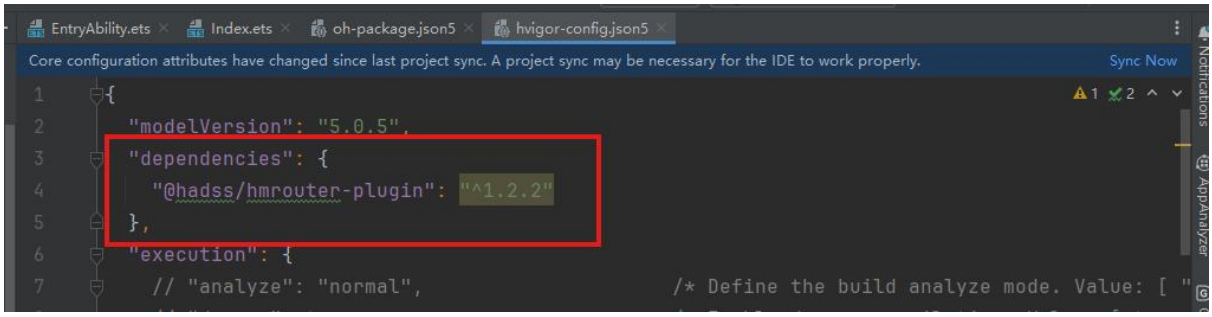


看起来到我写这个文档这天，最新的是 1.2.2 版本。

使用配置：

1. 修改工程的 hvigor/hvigor-config.json 文件，加入路由编译插件：

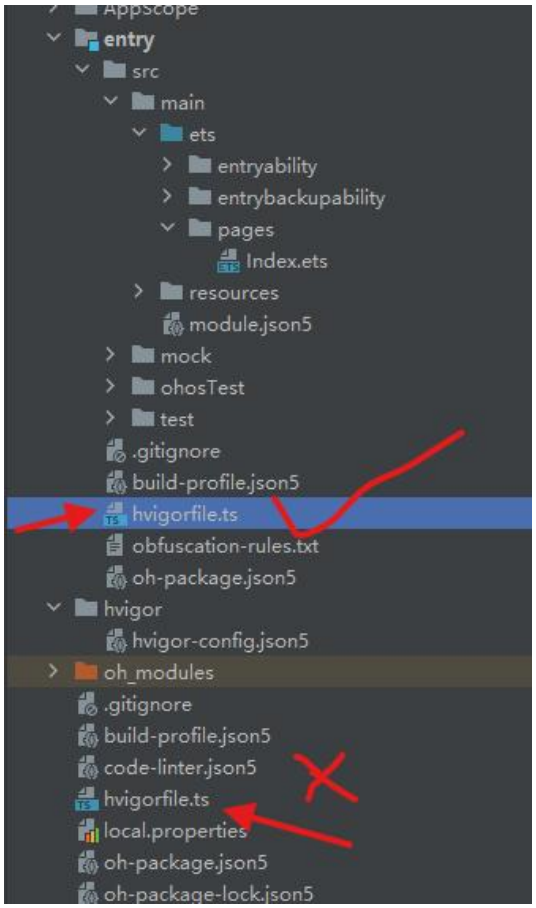





```
"dependencies": {
  "@hadss/hmrouter-plugin": "^1.2.2"
},
```

同样，加完之后，点击”Sync Now”。

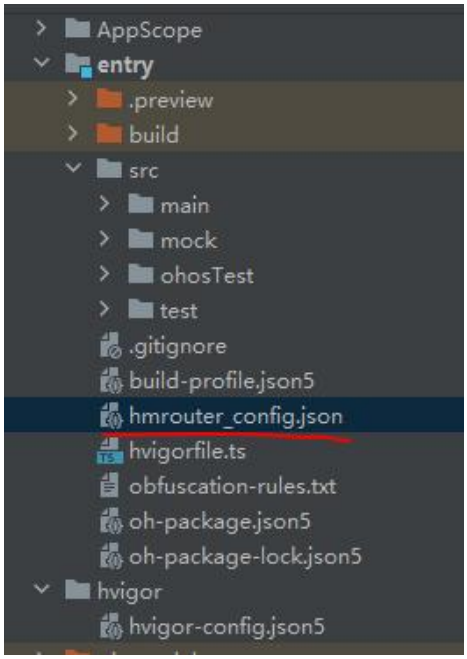
2. 在使用到 HMRouter 的模块中引入路由编译插件，修改模块目录中的 hvigorfile.ts，注意是模块目录下的：




```
import { hapTasks } from '@ohos/hvigor-ohos-plugin';
import { hapPlugin } from '@hadss/hmrouter-plugin';

export default {
  system: hapTasks,
  plugins: [hapPlugin()] // 使用 HMRouter 标签的模块均需要配置，与模块类型保持一致
}
```

3. 在模块目录或项目根目录创建路由编译插件配置文件 hmrouter_config.json（可选）

可以在模块目录或工程根目录下创建 hmrouter_config.json 文件:（我们就在模块目录下，即 entry 目录下面）

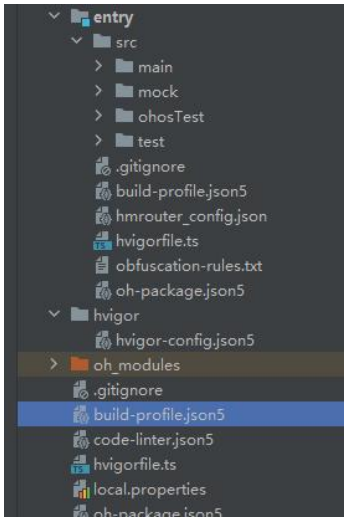


加入代码：

```
{
  "scanDir": ["src/main/ets/pages"],
  "saveGeneratedFile": false
}
```

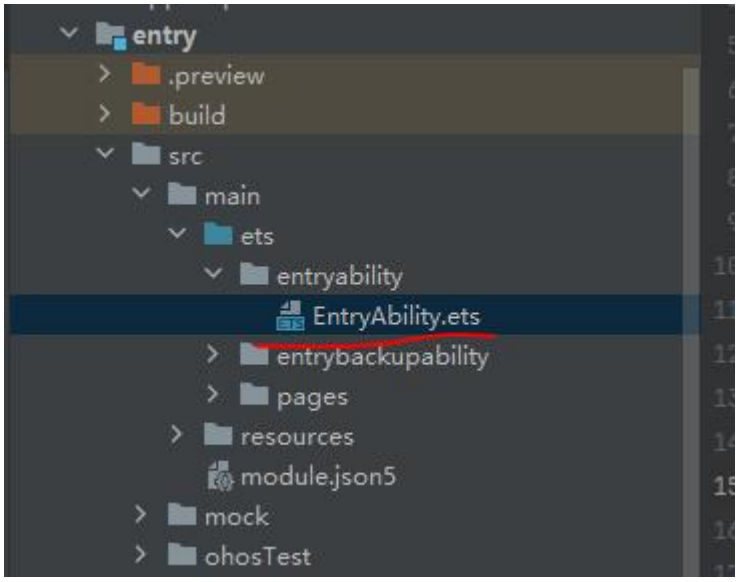
这样的好处是，HMRouter 会只扫描 scanDir 定义的目录，也不会保存中间文件，编译的速度会快一些。这个配置文件的逻辑是，系统会按照模块目录到工程根目录的顺序进行递归查找，如果本模块目录有这个 hmrouter_config.json 文件，就使用当前模块目录中的，没有的话再去上一层找。

4. 工程目录下的 build-profile.json5 中，确保配置 useNormalizedOHMUrl 属性为 true



现在，配置基本完成，我们可以开始使用 HMRouter 了。

1）到 ets > entryability > EntryAbility.ets 文件中：

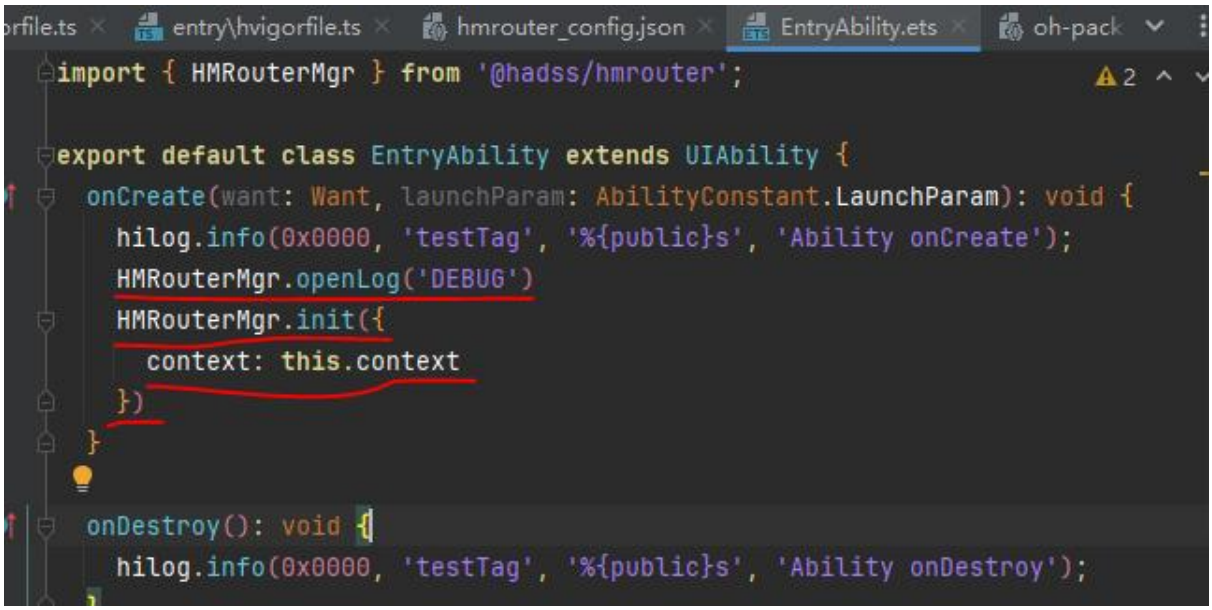


在 onCreate 函数中，添加一下 HMRouterMgr 的初始化代码：

```
export default class EntryAbility extends UIAbility {
  onCreate(want: Want, launchParam: AbilityConstant.LaunchParam): void {
    hilog.info(0x0000, 'testTag', '%{public}s', 'Ability onCreate');
    HMRouterMgr.openLog('DEBUG')
    HMRouterMgr.init({
      context: this.context
    })
  }
```

当然，前面需要导入：

```
import { HMRouterMgr } from '@hadss/hmrouter';
```



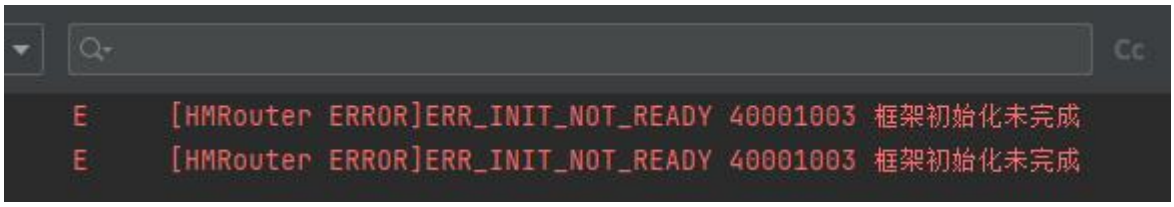
红色字体为新增的代码。

回到 Index.ets 文件，代码改为：

```
import { HMDefaultGlobalAnimator, HMNavigation } from '@hadss/hmrouter';
import { AttributeUpdater } from '@kit.ArkUI';


@Entry
@Component
export struct Index {
  modifier: NavModifier = new NavModifier();


  build() {
    Column() {
      // 使用 HMNavigation 容器
      HMNavigation({
```

```
            navigationId: 'mainNavigation',
            options: {
              standardAnimator: HMDefaultGlobalAnimator.STANDARD_ANIMATOR,
              dialogAnimator: HMDefaultGlobalAnimator.DIALOG_ANIMATOR,
              modifier: this.modifier
            }
          })
      }.height('100%')
      .width('100%')
    }
}

class NavModifier extends AttributeUpdater<NavigationAttribute> {
  initializeModifier(instance: NavigationAttribute): void {
    instance.mode(NavigationMode.Stack);
    instance.navBarWidth('100%');
    instance.hideTitleBar(true);
    instance.hideToolBar(true);
  }
}
```

此时如果刷新 Preivewer，在 Log 里面会有提示：



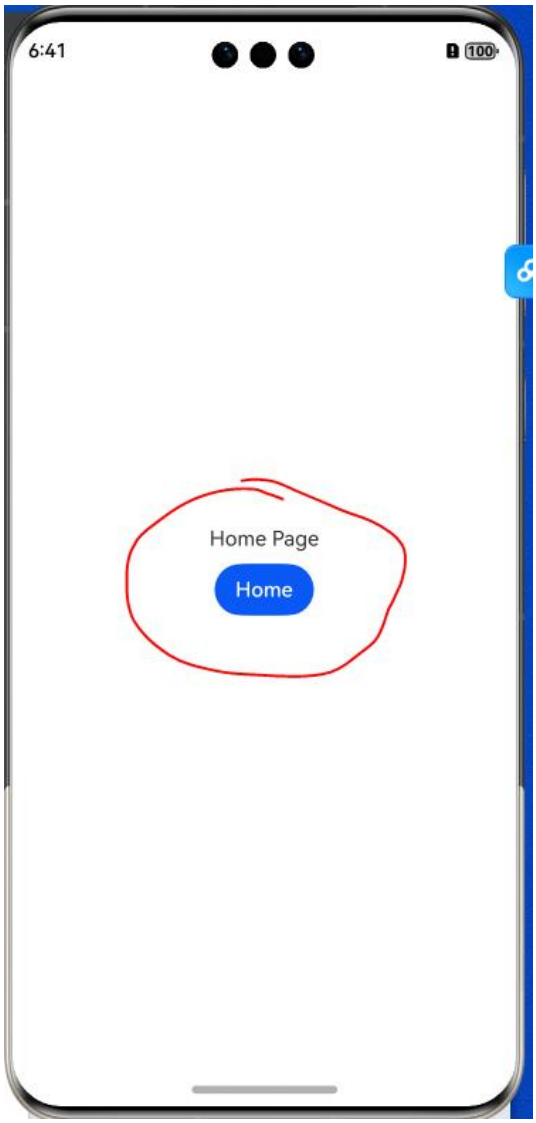
原因是当前预览器不支持 HMRouter。（不确定未来会不会支持）

我们用模拟器来操作，此时，看到模拟器是空白的，因为我们在容器里面没有放任何东西。我们放一些内容在里面：（红色字体部分）

```
  build() {
    Column() {
      // 使用 HMNavigation 容器
      HMNavigation({
        navigationId: 'mainNavigation',
        options: {
          standardAnimator: HMDefaultGlobalAnimator.STANDARD_ANIMATOR,
          dialogAnimator: HMDefaultGlobalAnimator.DIALOG_ANIMATOR,
          modifier: this.modifier
        }
      }){
        Column({space: 10}) {
          Text('Home Page')
          Button('Home').onClick(() => {
          })
        }
        .width('100%')
```

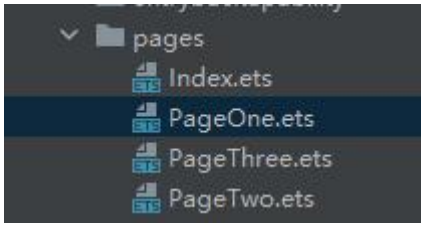
```
            .height('100%')
            .justifyContent(FlexAlign.Center)
        }
    }.height('100%')
    .width('100%')
  }
```

此时再次在模拟器中运行：



4.1 页面跳转与返回

通过添加 ArkTS File 的方式，增加三个页面：PageOne.ets，PageTwo.ets，PageThree.ets。



PageOne.ets 中加入基础代码：

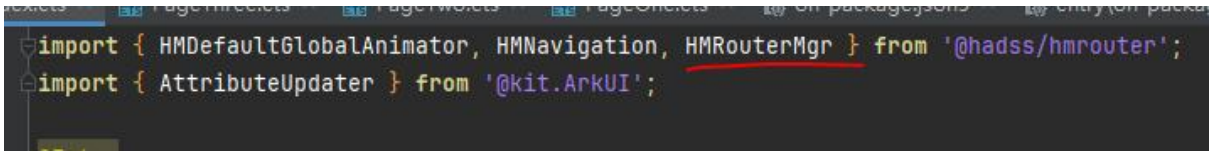
```
import { HMRouter} from '@hadss/hmrouter'


@HMRouter({pageUrl: 'PageOne'})
@Component
export struct PageOne {
  build() {
    Column({space: 10}) {
      Text('Page One')
      Button('One').onClick(() => {
      })
    }
    .width('100%')
    .height('100%')
    .justifyContent(FlexAlign.Center)
  }
}
```

注意，这里我们没有加@Entry，而是加了红色字体的那一行。当然相应地要导入 HMRouter。

这一行里面，我们定义了当前的这个页面的 Url 为“PageOne”（名字不必一定要一样），这样我们

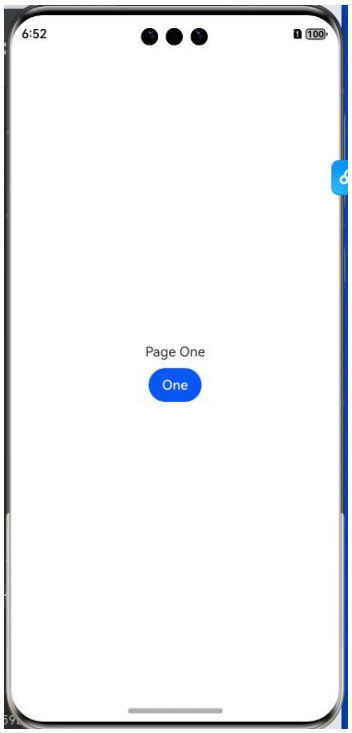
回到 Index.ets，给 home 按钮加上点击事件，让其跳转到 PageOne:

相应地，前面要导入这个 HMRouterMgr:（可通过选项自动更新导入）



请自行尝试添加代码。

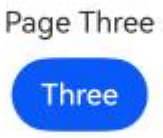
此时，在模拟器中再次运行，点击 home 按钮，可以看到页面跳转到 PageOne:



而且可以看到一个动画的效果。

请自行添加代码，实现从 home 跳转到 PageOne，然后跳转到 PageTwo，再到 PageThree。

现在我们想在 PageThree 中加入返回，如果点击 Three 按钮，就返回到上一页：



PageThree.ets 中修改代码，onClick 事件中加入这句话：

```
HMRouterMgr.pop()
```

```
import { HMRouter, HMRouterMgr } from '@hadss/hmrouter'

@HMRouter({pageUrl: 'PageThree'})
@Component
export struct PageThree {
  build() {
    Column({space: 10}) {
      Text('Page Three')
      Button('Three')
        .onClick(() => {
          HMRouterMgr.pop()
        })
    }
    .width('100%')
    .height('100%')
    .justifyContent(FlexAlign.Center)
  }
}
```

可以看到，点击 Three 按钮之后，会返回到 PageTwo。

通常推荐的做法是，在 Index.ets 中，我们在 HMNavigation 容器中放空，然后在参数里面加上一个 homePageUrl 的配置：

```
    // 使用HMNavigation容器
    HMNavigation({
      navigationId: 'mainNavigation', homePageUrl: 'PageOne', options: {
        standardAnimator: HMDefaultGlobalAnimator.STANDARD_ANIMATOR,
        dialogAnimator: HMDefaultGlobalAnimator.DIALOG_ANIMATOR,
        modifier: this.modifier
      }
    }){
      // Column({space: 10}) {
      //   Text('Home Page')
      //   Button('Home').onClick(() => {
      //     HMRouterMgr.push({pageUrl: 'PageOne'})
      //   })
      // }
      // .width('100%')
      // .height('100%')
      // .justifyContent(FlexAlign.Center)
    }
  }.height('100%')
  .width('100%')
}
```

这里相当于我们是把 PageOne 定义为了主页。

此时再次运行，在模拟器中第一个出现的页面就是 PageOne 了：

回到 PageThree，在 pop 里面可以带上参数：

```
    Button('Three')
      .onClick(() => {
        HMRouterMgr.pop({ pageUrl: 'PageOne' })
      })
  }
```

这样点击 Three 按钮，就直接返回到 PageOne 了，也就是现在的首页。

也可以将 PageThree 中的 pop 改为 replace：

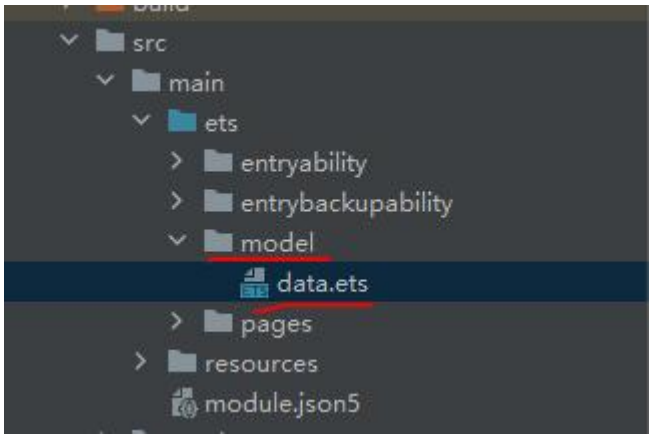
```
    Button('Three')
      .onClick(() => {
        HMRouterMgr.replace({ pageUrl: 'PageOne' })
      })
  }
```

点击 Three 按钮，页面替换为 PageOne。

4.2 路由传参

通过尝试从 PageOne 传参数到 PageTwo 来演示如何在路由跳转的同时传递参数。

先在 src > main > ets 下面创建一个目录 model，在 model 里面创建一个文件 data.ets：

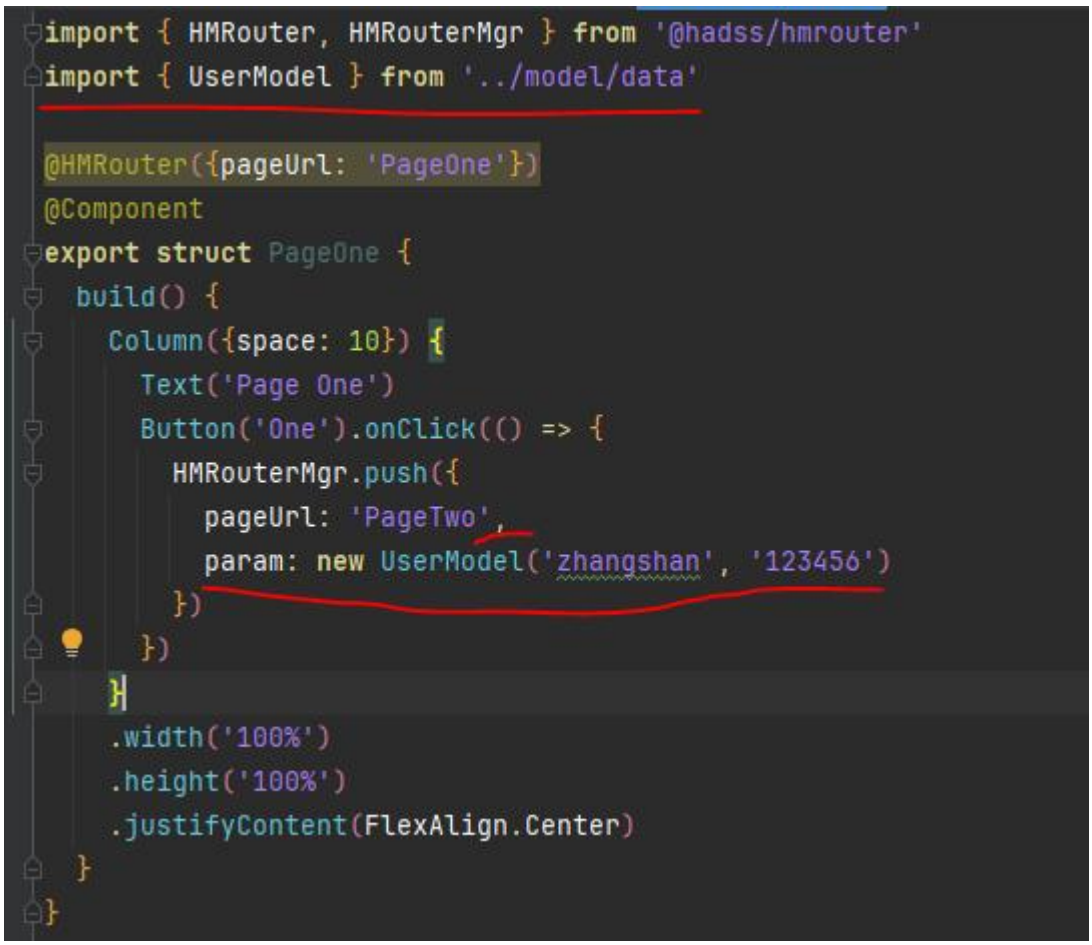


然后创建一个类 UserModel：

```
export class UserModel {
  username: string
  password?: string

  constructor(username: string, password: string) {
```

```
    this.username = username
    this.password = password
  }
}
```

在 PageOne.ets 中，调用 push 的时候，带上参数：



```
import { HMRouter, HMRouterMgr } from '@hadss/hmrouter'
import { UserModel } from '../model/data'

@HMRouter({pageUrl: 'PageOne'})
@Component
export struct PageOne {
  build() {
    Column({space: 10}) {
      Text('Page One')
      Button('One').onClick(() => {
        HMRouterMgr.push({
          pageUrl: 'PageTwo',
          param: new UserModel('zhangshan', '123456')
        })
      })
    }
    .width('100%')
    .height('100%')
    .justifyContent(FlexAlign.Center)
  }
}
```

请自己加上代码。

到 PageTwo.ets 中，先在 aboutToAppear()函数中接收数据：（这个函数加在哪里，请自行确定）

```
  aboutToAppear(): void {
    const model = HMRouterMgr.getCurrentParam() as UserModel
    console.log('PageTwo 收到的参数：', JSON.stringify(model))
```

```
}
```

此时重新在模拟器上运行，点击 One 按钮跳转到 PageTwo，查看 Log：

```
PageTwo收到的参数:  {"username":"zhangshan","password":"123456"}
```

可以看到，参数被正确传递过来了。我们也可以加一些 UI 去显示接收到的数据：

```
export struct PageTwo {
  @State username: string = ''

  aboutToAppear(): void {
    const model = HMRouterMgr.getCurrentParam() as UserModel
    console.log('PageTwo收到的参数: ', JSON.stringify(model))
    this.username = model.username
  }

  build() {
    Column({space: 10}) {
      Text('Page Two')
      Text(this.username)
      Button('Two').onClick(() => {
        HMRouterMgr.push({pageUrl: 'PageThree'})
      })
    }
    .width('100%')
```

效果：

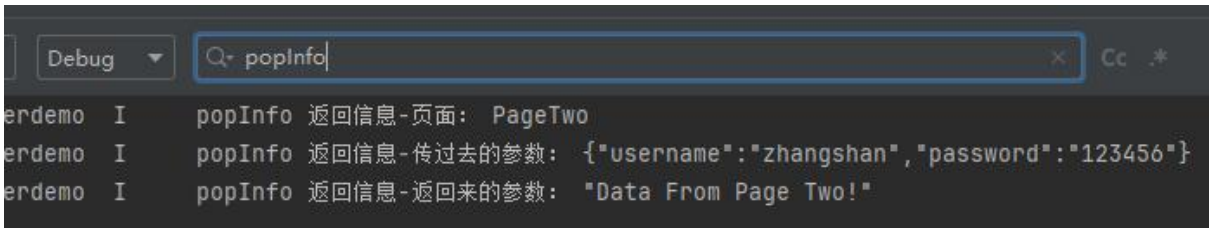
类似的，pop 返回也可以带参数，我们把 PageTwo 中按钮 Two 的事件改为点击后返回到 PageOne，也带一个参数，简单起见，就带一个字符串：

回到 PageOne，如何接收页面返回的参数信息呢？修改 push 里面的内容，加上 onResult:

```
build() {
  Column({ space: 10 }) {
    Text('Page One')
    Button('One').onClick(() => {
      HMRouterMgr.push({
        pageUrl: 'PageTwo',
        param: new UserModel('zhangshan', '123456')
      }, {
        onResult(popInfo: HMPopInfo) {
          console.log('popInfo', '返回信息-页面：', popInfo.srcPageInfo.name)
          console.log('popInfo', '返回信息-传过去的参数：', JSON.stringify(popInfo.srcPageInfo.param))
          console.log('popInfo', '返回信息-返回来的参数：', JSON.stringify(popInfo.result))
        }
      })
```

```
      })
    }
    .width('100%')
    .height('100%')
    .justifyContent(FlexAlign.Center)
  }
```

红色字体为新加的部分。模拟器中再次运行，One 跳转到 Two，再返回 One，查看 Log:

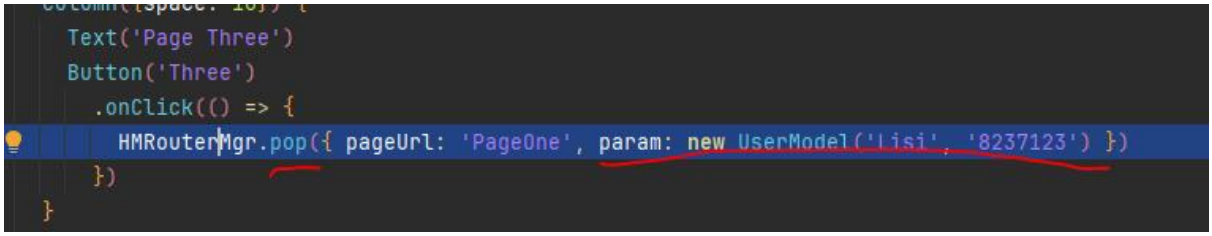


可以看到，返回的参数’Data From Page Two!’被正确接收了。

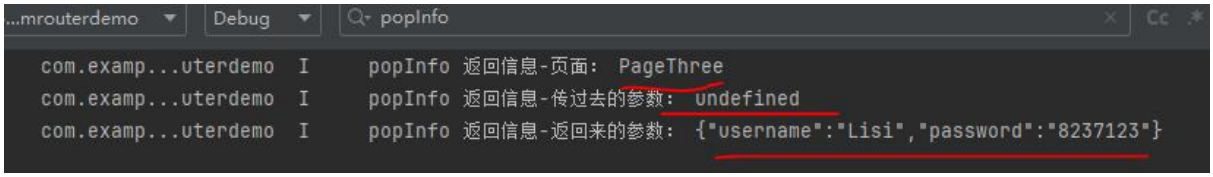
把 PageTwo 的按钮点击事件改回为跳转到第三个页面，然后第三个页面按钮返回第一个页面，即 PageOne -> PageTwo->PageThree->PageOne。

PageTwo：

在 PageThree 的 pop 中带上参数：



再次运行，返回到 PageOne 之后，查看 Log：



可以看到，参数也被正确传递了。

到这里为止，基本上讲解了@ohos.router, Navigation 组件，HMRouter 的最基本的用法。

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？