《嵌入式应用开发》

青蛙影院项目—底部导航栏搭建实验指导手册

版本：V 1.0

目录

（一）实验目的

1. 掌握青蛙影院基于 ArkTS 语言的开发。
2. 掌握 Tabs、TabContent 和 TabBar 组件的使用。
3. 理解页签页和对应内容页切换的实现逻辑。
4. 能够运用所学制作底部导航栏效果。

（二）实验涉及知识点

1. 基础组件和布局。
2. 装饰器的使用。
3. 使用 Tab 搭建项目框架。
4. MVVM 开发模式。

（三）实验准备

参考开发环境：

操作系统：Window 10

开发工具：DevEco Studio 3.1.1

HarmonyOS SDK 版本：API version 9 及以上版本

开发语言：ArkTS

内存：8G 及以上

（四）详细实验过程

1 Tabs 组件制作简单的底部导航栏效果

青蛙影院底部导航栏的效果如下图所示：

在搭建页面框架的时候，我们采用 Tabs 组件及其子组件 TabContent 来实现。

导航栏位置使用 Tabs 组件的参数 barPosition 进行设置，其值有 2 个：默认值 Start，导航栏位于顶部。End，导航栏位于底部。

在 Tabs 组件中使用花括号包裹 TabContent，每一个 TabContent 对应的内容需要有一个页签，通过 TabContent 的 tabBar 属性进行配置。TabContent 组件不

支持设置宽高属性，其宽度默认撑满 Tabs 父组件，高度由 Tabs 父组件高度与 TabBar 组件高度决定。

1.1 Tabs 组件的应用

接下来我们来制作一个简单的底部导航栏效果，在 MainPage.ets 中编写如下代码：

```
@Preview
@Entry
@Component
struct MainPage {

  build(){
    Tabs({barPosition:BarPosition.End}){
      TabContent(){
        Text('page1')
      }.backgroundColor("#f1f3f5").tabBar('首页')

      TabContent(){
        Text('page2')
      }.backgroundColor("#f1f3f5").tabBar('影院')

      TabContent(){
        Text('page3')
      }.backgroundColor("#f1f3f5").tabBar('抢先看')

      TabContent(){
        Text('page4')
      }.backgroundColor("#f1f3f5").tabBar('个人中心')
    }
  }
}
```

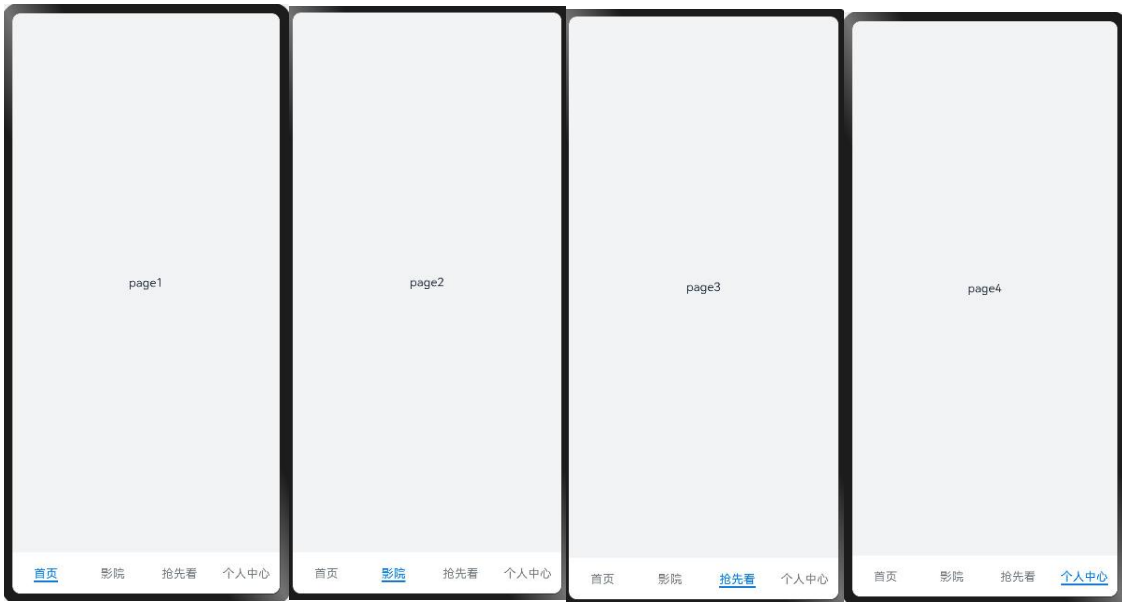
代码解读：关于@Preview 装饰器

因为我们的页面路由的逻辑是从登录页输入正确的用户名和密码后，先进过渡页，然后才到主界面。所以在预览器中预览时，为了避免每次预览主页面时都要先从登录页进来，所以可以在该 MainPage 组件最上方添加@Preview 装饰器，这样预览器预览的就只是 MainPage 页面。

或者是当你的自定义组件没有@Entry 装饰器时（即该组件不再作为入口组件），也可以通过添加@Preview 装饰器来预览。

需要注意的是@Preview 在单个文件中只能加一次，哪个步骤需要预览时就添加给它即可。

此时代码的预览效果如下所示，通过下划线来区分当前页，单击菜单可以切换到对应的内容。

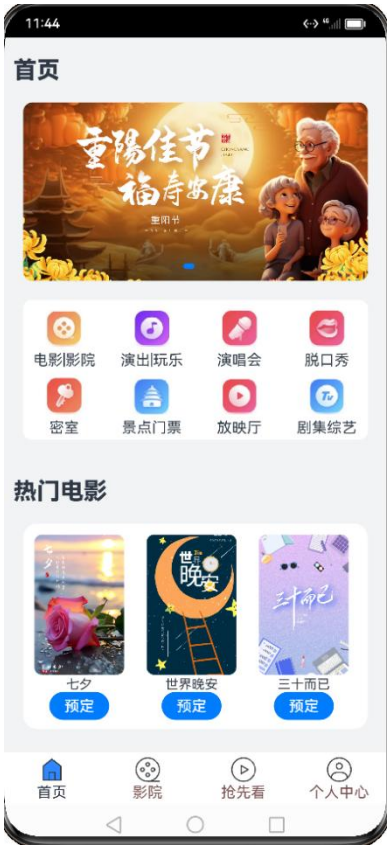


1.2 问题分析

上面的底部导航栏效果与青蛙影院实际的底部导航栏效果还存在差异。

①青蛙影院的底部导航栏是自定义导航栏，导航栏中会组合文字以及对应图片表示页签内容，这种情况下就需要自定义导航页签的样式，用于区分当前活跃页签和未活跃页签。

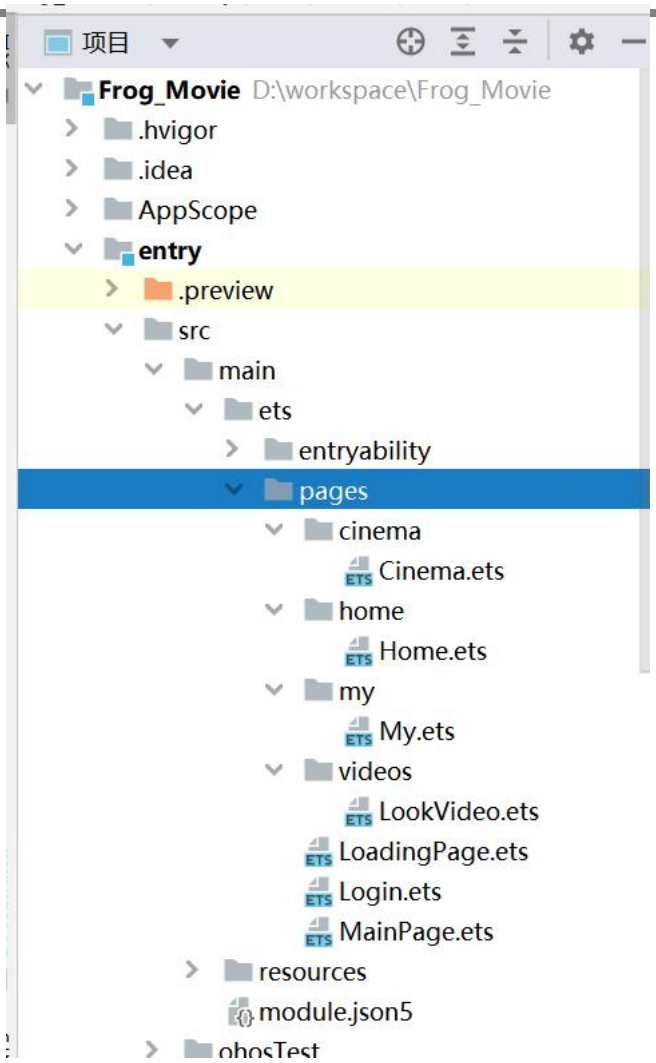
②每个页签对应的页面，如首页对应的 page1，我们是在 TabContent()中通过添加了 Text('page1')文本组件来呈现的。而在实际的开发中首页页签对应的应该是一个单独的首页页面（如下图所示），所以我们需要创建“首页、影院、抢先看、个人中心”对应的 4 个子组件。



2 Tabs 组件自定义底部导航栏效果

2.1 工程目录结构

基于上一步中的问题，我们先来创建“首页、影院、抢先看、个人中心”对应的 4 个子组件。创建后的工程目录结构如下图所示：



其中：Cinema.ets 为影院模块，Home.ets 为首页，My.ets 为个人中心模块，LookVideo.ets 为抢先看模块。

2.2 编写 4 个子页面

接下来简单修改这 4 个 ets 文件的内容，我们以 Home.ets 为例，其代码如下：

```
@Component
export struct Home {
  @State message: string = 'page1'

  build() {
```

```
    Row() {
      Column() {
        Text(this.message)
          .fontSize(50)
          .fontWeight(FontWeight.Bold)
      }
      .width('100%')
    }
    .height('100%')
  }
}
```

另外 3 个组件的代码同 Home.ets，这里只需先简单修改变量 message 的值为对应的 page 页做简单区分即可。

由于我们需要在 MainPage.ets 组件相应的 TabContent(){}位置中引入自定义的 4 个组件，所以在这 4 个子组件中我们需要删除@Entry 装饰器，使其不再作为入口组件，组件独立封装成自定义组件，需要被外部组件引用，那么必须加上 export 关键字。

2.3 Tabs 自定义底部导航栏

2.3.1 在 MainPage.ets 中引入定义好的 4 个自定义子组件

4 个页面准备好后，我们先将这 4 个子组件引入到 MainPage.ets 中，此时 MainPage.ets 中的代码如下：

```
import { Home } from './home/Home'
import {Cinema} from './cinema/Cinema'
import {LookVideo} from './videos/LookVideo'
import {My} from './my/My'

@Preview
@Entry
@Component
```

```
struct MainPage {

  build(){
    Tabs({barPosition:BarPosition.End}){
      TabContent(){
        Home()

       // Text('page1')
      }.backgroundColor("#f1f3f5").tabBar('首页')

      TabContent(){
        Cinema()

       // Text('page2')
      }.backgroundColor("#f1f3f5").tabBar('影院')

      TabContent(){
        LookVideo()

       // Text('page3')
      }.backgroundColor("#f1f3f5").tabBar('抢先看')

      TabContent(){
        My()

       // Text('page4')
      }.backgroundColor("#f1f3f5").tabBar('个人中心')
    }
  }
}
```

2.3.2 自定义导航栏效果

接下来我们自定义底部导航栏效果，并实现在切换导航栏菜单时切换到对应的页面。

在导航栏中会组合文字以及对应图片表示页签内容，如下图所示。

设置自定义导航栏需要使用 tabBar 的参数。首先定义自定义函数组件，并在自定义函数组件中传入参数：包括页签文字，对应位置索引，以及选中状态和未选中状态的图片资源。通过当前活跃的索引和页签对应的索引匹配与否，决定 UI 显示的样式。

然后在 TabContent 对应 tabBar 属性中传入自定义函数组件，并传递相应的参数。

继续修改 MainPage.ets 中的代码如下：

```
import { Home } from './home/Home'
import {Cinema} from './cinema/Cinema'
import {LookVideo} from './videos/LookVideo'
import {My} from './my/My'

@Preview
@Entry
@Component
struct MainPage {
  // 当前的索引值
  @State currentIndex: number = 0
  // 自定义函数组件,传递 4 个参数，分别为：页签文字、当前的索引，选中时的图片路径、未选中时的图片路径
  @Builder
  TabBuilder(title: string, index: number, selectImg: Resource, normalImg: Resource) {
    Column() {
      Image(this.currentIndex == index ? selectImg : normalImg)
        .width(30)
        .height(30)
      Text(title)
```

```
        .fontColor(this.currentIndex == index ? "#ff0a1a6c" : "#ff7c5454")
    }.onClick(() => {
      this.currentIndex = index
      console.log(`----->index:${index}`)
    })
  }


  build(){
    Tabs({barPosition:BarPosition.End}){
      TabContent(){
        Home()
        // Text('page1')
      }.backgroundColor("#f1f3f5")
      // 在tabBar属性中调用TabBuilder自定义函数组件，并传入相应的实际参数
      .tabBar(this.TabBuilder("首页", 0, $r("app.media.home_1"),
$r("app.media.home_0")))

      TabContent(){
        Cinema()
        // Text('page2')
      }.backgroundColor("#f1f3f5")
      .tabBar(this.TabBuilder("影院", 1, $r("app.media.movie_1"),
$r("app.media.movie_0")))

      TabContent(){
        LookVideo()
        // Text('page3')
      }.backgroundColor("#f1f3f5")
      .tabBar(this.TabBuilder("抢先看", 2, $r("app.media.play_1"),
$r("app.media.play_0")))

      TabContent(){
        My()
        // Text('page4')
```

```
        }.backgroundColor("#f1f3f5")
        .tabBar(this.TabBuilder("个人中心", 3, $r("app.media.me_1"),
$r("app.media.me_0")))
    }
  }
}
```

此时的页面效果如下图所示：

2.3.3 问题分析与解决

此时的代码存在两个问题：

①在鼠标单击切换时，菜单栏能够正常的切换到当前状态，但是内容页无法随着切换。

解决方法：使用 TabsController，TabsController 是 Tabs 组件的控制器，用于控制 Tabs 组件进行页签切换。通过 TabsController 的 changeIndex 方法来实现跳转至指定索引值对应的 TabContent 内容。

②使用 TabsController 可以实现点击页签与页面内容的联动，但不能实现滑动页面时，页面内容与对应页签的联动。

解决方法：使用 Tabs 提供的 onChange 事件方法，监听索引 index 的变化，并将其当前活跃的 index 值传递给 menuIndex，实现页签内容的切换。

此时 MainPage.ets 的完整代码如下：

```
import { Home } from './home/Home'
import {Cinema} from './cinema/Cinema'
import {LookVideo} from './videos/LookVideo'
import {My} from './my/My'

@Preview
@Entry
@Component
struct MainPage {
  // 当前的索引值
  @State currentIndex: number = 0
  // 定义 TabsController 控制器
  tabController: TabsController = new TabsController()
  // 自定义函数组件,传递 4 个参数，分别为：页签文字、当前的索引，选中时的图片路径、未选中时的图片路径
  @Builder
  TabBuilder(title: string, index: number, selectImg: Resource, normalImg: Resource) {
    Column() {
      Image(this.currentIndex == index ? selectImg : normalImg)
        .width(30)
        .height(30)
```

```
      Text(title)
        .fontColor(this.currentIndex == index ? "#ff0ala6c" : "#ff7c5454")
    }.onClick(() => {
      this.currentIndex = index
      console.log(`----->index:${index}`)
      // 通过 TabsController 的 changeIndex 方法来实现跳转至指定索引值对应的
TabContent 内容
      this.tabController.changeIndex(index)
    })
  }


  build(){
    Tabs({barPosition:BarPosition.End,controller:this.tabController}){
      TabContent(){
        Home()
        // Text('page1')
      }.backgroundColor("#f1f3f5")
      // 在 tabBar 属性中调用 TabBuilder 自定义函数组件，并传入相应的实际参数
      .tabBar(this.TabBuilder("首页", 0, $r("app.media.home_1"),
$r("app.media.home_0")))

      TabContent(){
        Cinema()
        // Text('page2')
      }.backgroundColor("#f1f3f5")
      .tabBar(this.TabBuilder("影院", 1, $r("app.media.movie_1"),
$r("app.media.movie_0")))

      TabContent(){
        LookVideo()
        // Text('page3')
      }.backgroundColor("#f1f3f5")
      .tabBar(this.TabBuilder("抢先看", 2, $r("app.media.play_1"),
$r("app.media.play_0")))
```

```
      TabContent(){
        My()
        // Text('page4')
      }.backgroundColor("#f1f3f5")
      .tabBar(this.TabBuilder("个人中心", 3, $r("app.media.me_1"),
$r("app.media.me_0")))
    }
    // onChange 事件方法，监听索引 index 的变化，并将其当前活跃的 index 值传递给
currentIndex，实现页签内容的切换
    .onChange((index)=>{
      this.currentIndex=index
    })

  }
}
```