实验十七　组件和页面生命周期

一、实验目的

1. 了解 DevEco Studio 的使用
2. 学习 ArkTS 语言及 ArkUI 组件和页面的生命周期
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

“生命周期”指的是组件和页面在创建、显示和销毁整个过程中，会自动执行的一系列的函数，也就是所谓的“钩子”，让开发者有机会在这些钩子里面放置自己的代码，从而确保在特定的阶段被运行到。

理论讲解：

https://developer.huawei.com/consumer/cn/doc/harmonyos-guides/arkts-page-custom-components-lifecycle-0000001820879573

几个重点：

- 自定义组件与页面的关系
- 页面生命周期
- 组件生命周期

在开始之前，我们先明确自定义组件和页面的关系：

- 自定义组件：@Component装饰的UI单元，可以组合多个系统组件实现UI的复用，可以调用组件的生命周期。
- 页面：即应用的UI页面。可以由一个或者多个自定义组件组成，@Entry装饰的自定义组件为页面的入口组件，即页面的根节点，一个页面有且仅能有一个@Entry。只有被@Entry装饰的组件才可以调用页面的生命周期。

自定义组件的生命周期

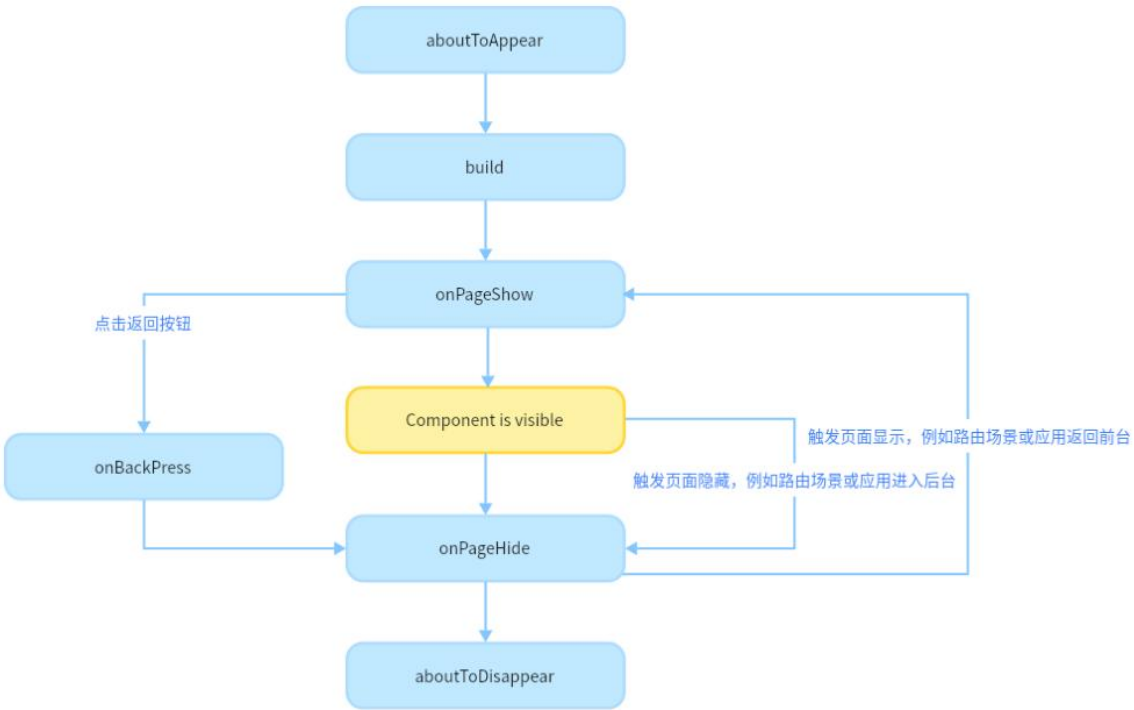
自定义组件生命周期，即用@Component或@ComponentV2装饰的自定义组件的生命周期，提供以下生命周期接口：

- aboutToAppear：组件即将出现时回调该接口，具体时机为在创建自定义组件的新实例后，在执行其build函数之前执行。
- onDidBuild：在组件首次渲染触发的build函数执行完成之后回调该接口，后续组件重新渲染将不回调该接口。开发者可以在这个阶段实现埋点数据上报等不影响实际UI的功能。
- aboutToDisappear：aboutToDisappear函数在自定义组件析构销毁之前执行。不允许在aboutToDisappear函数中改变状态变量，特别是@Link变量的修改可能会导致应用程序行为不稳定。

router 页面生命周期

router页面生命周期，即被@Entry装饰的组件生命周期，提供以下生命周期接口：

- onPageShow：页面每次显示时触发一次，包括路由过程、应用进入前台等场景。
- onPageHide：页面每次隐藏时触发一次，包括路由过程、应用进入后台等场景。
- onBackPress：当用户点击返回按钮时触发。



我们通过实例来理解。

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

设置项目名称为 LifeCycleDemo。

2. 组件、页面的生命周期

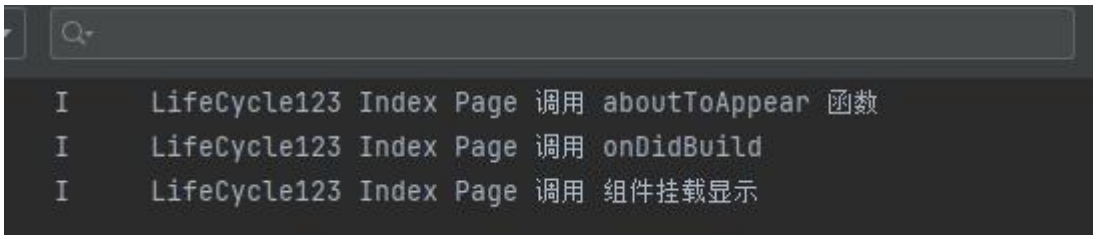
页面本身也是一个组件，我们可以先用页面的组件属性来演示。在 Index.ets 这个页面中，build（）前面，我们加入三个函数：

```
  aboutToAppear(): void {
    console.log('LifeCycle123', 'Index Page 调用 aboutToAppear 函数')
  }

  aboutToDisappear(): void {
    console.log('LifeCycle123', 'Index Page 调用 aboutToDisAppear 函数')
  }

onDidBuild(): void {
  console.log('LifeCycle123', 'Index Page 调用 onDidBuild');
}
```

在 RelativeContainer 上添加函数：

```
    .onAppear(() => {
      console.log('LifeCycle123', 'Index Page 调用 组件挂载显示')
    })
```

刷新 Previewer，在 Log 中可以看到：



这里的容器的 onAppear 函数，是可以表明组件即将显示，也就是上面流程图中的黄色背景部分。

可以看到在 build 之前，aboutToAppear 先调用。所以通常页面的一些初始化的动作可以在这里完成。

在 main > ets > entryability 目录下的 EntryAbility.ets 文件中，我们在 onBackground 函数中，添加一段代码，期望当应用放到后台运行时，直接调用 terminateSelf 关闭应用：

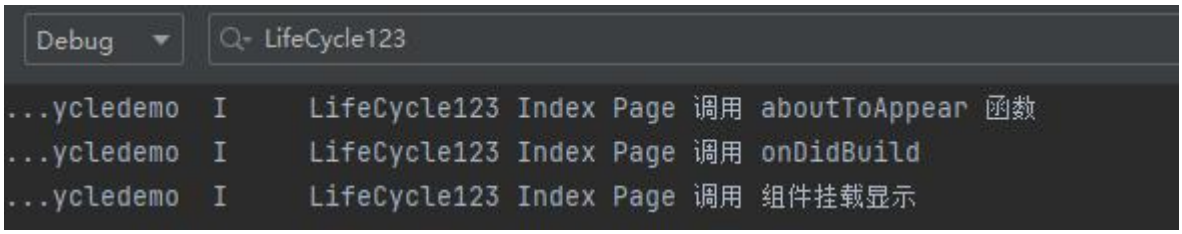
```
  onBackground(): void {
    // Ability has back to background
    hilog.info(0x0000, 'testTag', '%{public}s', 'Ability onBackground');
    try {
      this.context.terminateSelf((err: BusinessError) => {
        if (err.code) {
          // 处理业务逻辑错误
          console.error(`terminateSelf failed, code is ${err.code}, message is ${err.message}`);
          return;
        }
        // 执行正常业务
        console.info('terminateSelf succeed');
      });
```

```
    } catch (err) {
        // 捕获同步的参数错误
        let code = (err as BusinessError).code;
        let message = (err as BusinessError).message;
        console.error(`terminateSelf failed, code is ${code}, message is ${message}`);
    }
  }
```

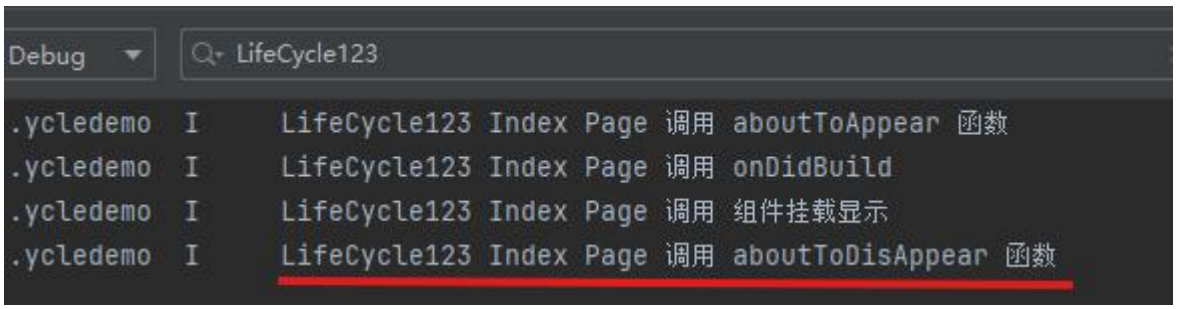
如果有提示错误，需要在文件头部导入 BusinessError 这个类型。

```
import { BusinessError } from '@kit.BasicServicesKit';
```

在模拟器中运行，页面出现之后，查看 Log：



和 Previewer 效果是一样的。然后我们通过上滑页面，将 APP 切到后台运行，此时再看 Log:



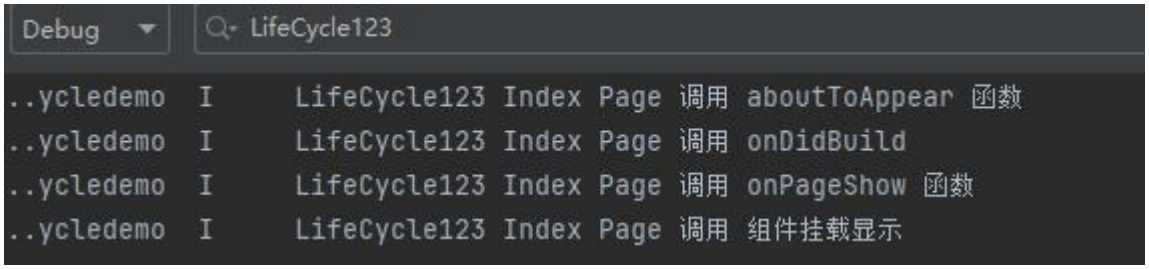
增加了这一行，表示应用被切到后台或关闭的时候，aboutToDisAppear 被调用了。

对于页面来说，页面比普通的组件多了三个函数，我们加上这三个函数：

```
  // 只有被@Entry 装饰的组件才可以调用页面的生命周期
  onPageShow() {
```

```
    console.info('LifeCycle123', 'Index Page 调用 onPageShow 函数');
  }
  // 只有被@Entry 装饰的组件才可以调用页面的生命周期
  onPageHide() {
    console.info('LifeCycle123', 'Index Page 调用 onPageHide 函数');
  }

  // 只有被@Entry 装饰的组件才可以调用页面的生命周期
  onBackPress() {
    console.info('LifeCycle123', 'Index Page 调用 onBackPress 函数');
    return true // 返回 true 表示页面自己处理返回逻辑，不进行页面路由；返回 false 表示使用默认的路由返回逻辑，不设置返回值按照 false 处理
  }
```

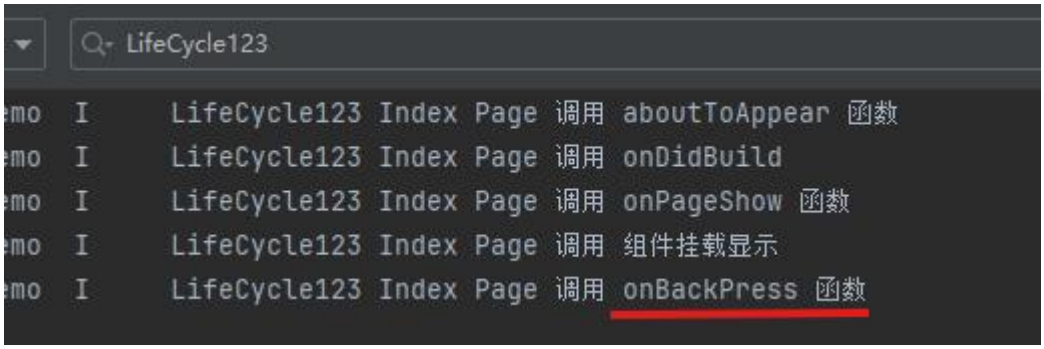
此时，刷新 Previewer，查看 Log：



可以看到，onPageShow 是在组件显示之前，在 build 函数调用之后执行的。

此时，如果点击回退按钮：

查看 Log:



表示 onBackPress 函数被调用了。

再用模拟器来运行，页面出现后，上滑关闭页面：

可以看到，onPageHide 被调用，而且是在 aboutToDisAppear 之前。

在此过程中，也可以点击模拟器上的回退按钮测试 onBackPress 函数的调用情况：

3. 组件的生命周期

页面本身就是组件，上面一节其实已经包含了组件的生命周期，但是页面有其特殊性，所以可以通过加一个单独的子组件来演示。

在 Index.ets 中，加一个子组件 SonCom 并加上生命周期函数 aboutToAppear 以及 aboutToDisAppear：

```
@Component
struct SonCom {
  aboutToAppear(): void {
    console.log('LifeCycle123', 'SonCom 调用 aboutToAppear 函数')
  }
  aboutToDisappear(): void {
    console.log('LifeCycle123', 'SonCom 调用 aboutToDisappear 函数')
  }
  build() {
    Column() {
      Text('我是子组件')
        .fontColor(Color.White)
        .fontSize(30)
```

```
    }
    .width(200)
    .height(200)
    .backgroundColor(Color.Blue)
    .border({ width: 3 })
  }
}
```

修改 Index 的 build()函数，首先给界面添加一个按钮去控制子组件的显示或隐藏，build()函数修改为：

```
  build() {
    RelativeContainer() {
      Text(this.message)
        .id('HelloWorld')
        .fontSize(50)
        .fontWeight(FontWeight.Bold)
        .alignRules({
          center: { anchor: '__container__', align: VerticalAlign.Top },
          middle: { anchor: '__container__', align: HorizontalAlign.Center }
        })

      Button('切换子组件显示')
        .id('ShowHide')
        .alignRules({
```

```
            top: { anchor: 'HelloWorld', align: VerticalAlign.Bottom },
            middle: { anchor: 'HelloWorld', align: HorizontalAlign.Center }
        })

      SonCom()
        .id('SonCom')
        .margin(40)
        .alignRules({
          top: {anchor: 'ShowHide', align: VerticalAlign.Bottom},
          middle: { anchor: 'HelloWorld', align: HorizontalAlign.Center }
        })
    }
    .height('100%')
    .width('100%')
    .onAppear(() => {
      console.log('LifeCycle123', 'Index Page 调用 组件挂载显示')
    })
  }
```

红色字体为修改或新增的部分，此时刷新 Previewer，效果：

接着给按钮添加 onClick 事件，去控制子组件的显示。先添加一个状态变量：

```
@State show: boolean = true
```

然后添加 onClick 事件：

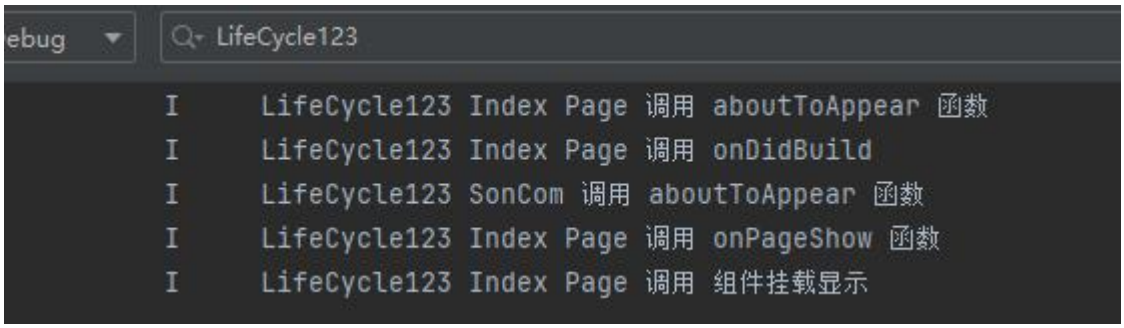
```
.onClick(() => {
  this.show = !this.show
})
```

再把 SonCom 子组件的调用用 if(this.show)条件判断包起来：

```
if(this.show) {
  SonCom()
    .id('SonCom')
    .margin(40)
    .alignRules({
      top: {anchor: 'ShowHide', align: VerticalAlign.Bottom},
      middle: { anchor: 'HelloWorld', align: HorizontalAlign.Center }
```

```
        })
    }
```

相信同学们可以自己找到对应的代码位置。刷新 Previewer，查看 Log:



可以看到，子组件的 aboutToAppear 是在页面的 aboutToAppear 之后就调用了的。

此时，点击按钮，子组件消失：

查看 Log:

可以看到，子组件的 aboutToDisappear 被调用了。

再点击按钮，查看 Log:

可以看到，子组件的 aboutToAppear 被调用。

这样，可以说明子组件在显示和消失的时候，会调用相应的函数。而随着页面显示的时候，是在页面的 aboutToAppear 函数之后就调用了，应该是在页面的 build 之前，当然也在子组件的 build 之前。也就是说页面和子组件的这些生命周期函数基本上是一个嵌套顺序调用的关系。

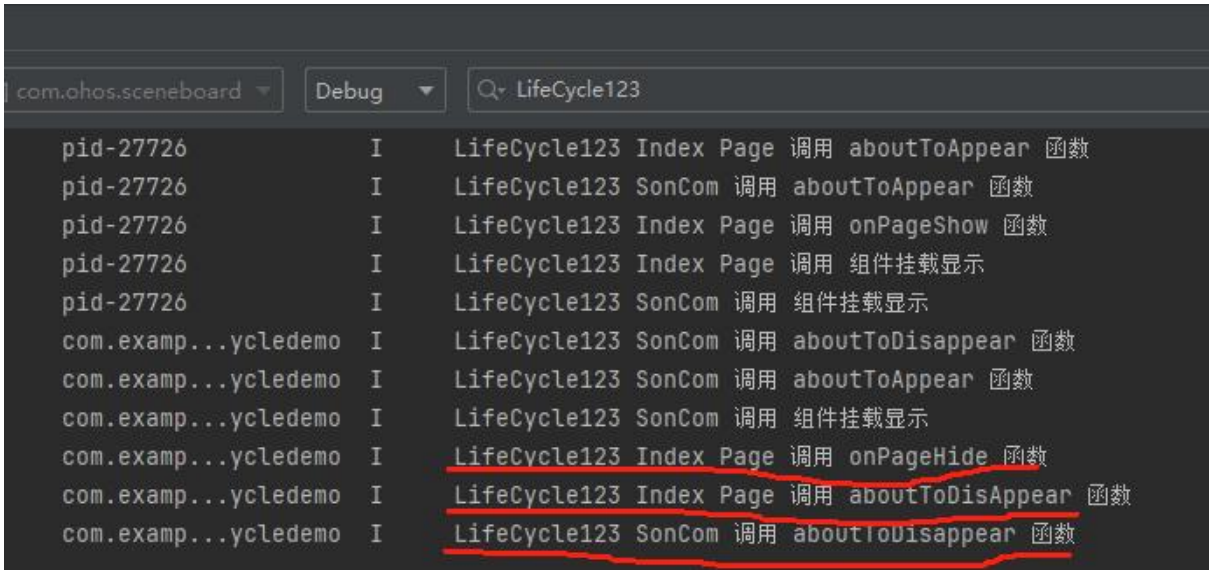
我们给子组件也加上 onAppear 组件挂载显示函数：

```
    .onAppear(() => {
      console.log('LifeCycle123', 'SonCom 调用 组件挂载显示')
    })
```

刷新 Previewer，查看 Log:

可以更好地理解嵌套顺序。

也可以在模拟器上运行，做一个完整的过程，从显示，点击按钮让子组件消失，再点击恢复，再到最后关闭应用，查看 Log:



请自行尝试，理解这个顺序。

再看一个例子，说明父组件和子组件的生命周期函数的调用顺序，创建一个新的页面ParentChildDemo.ets，加入代码：

```
@Entry
@Component
struct Parent {
  @State showChild: boolean = true;
  @State btnColor: string = '#FF007DFF';

  // 组件生命周期
  aboutToAppear() {
    console.info('Parent aboutToAppear');
  }
```

```
  // 组件生命周期
  onDidBuild() {
    console.info('Parent onDidBuild');
  }

  // 组件生命周期
  aboutToDisappear() {
    console.info('Parent aboutToDisappear');
  }

  build() {
    Column() {
      // this.showChild 为 true，创建 Child 子组件，执行 Child aboutToAppear
      if (this.showChild) {
        Child()
      }
      Button('delete Child')
        .margin(20)
        .backgroundColor(this.btnColor)
        .onClick(() => {
          // 更改 this.showChild 为 false，删除 Child 子组件，执行 Child
aboutToDisappear
```

```
            // 更改 this.showChild 为 true ，添加 Child 子组件 ，执行 Child aboutToAppear
            this.showChild = !this.showChild;
        })
    }
  }
}

@Component
struct Child {
  @State title: string = 'Hello World';

  // 组件生命周期
  aboutToDisappear() {
    console.info('Child aboutToDisappear');
  }

  // 组件生命周期
  onDidBuild() {
    console.info('Child onDidBuild');
  }

  // 组件生命周期
```

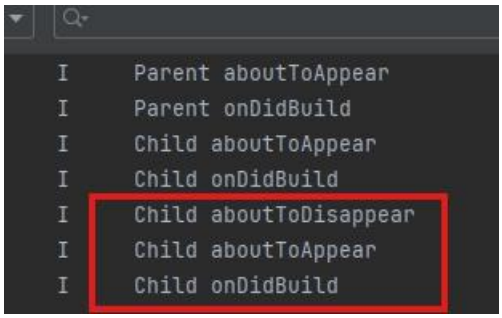
```
  aboutToAppear() {
    console.info('Child aboutToAppear');
  }

  build() {
    Text(this.title)
      .fontSize(50)
      .margin(20)
      .onClick(() => {
        this.title = 'Hello ArkUI';
      })
  }
}
```

刷新 Previewer，查看 Log:

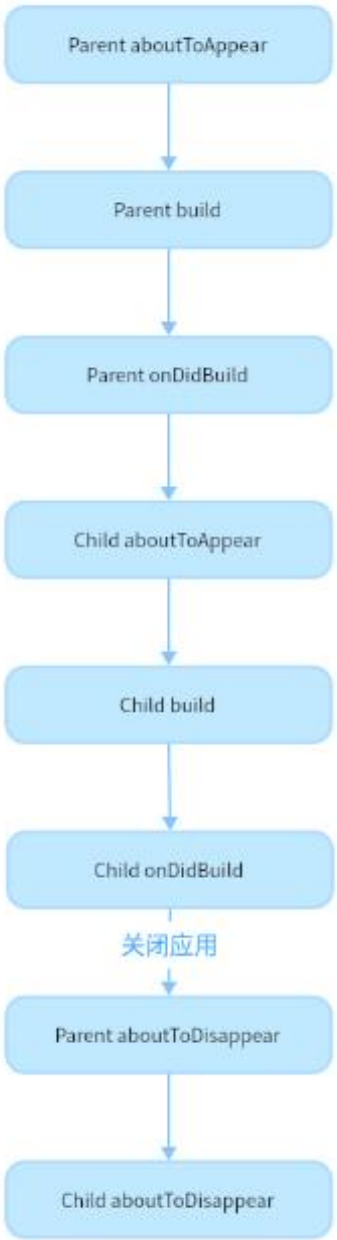
```
I     Parent aboutToAppear
I     Parent onDidBuild
I     Child aboutToAppear
I     Child onDidBuild
```

说明应用冷启动的初始化流程为：Parent aboutToAppear --> Parent build --> Parent onDidBuild --> Child aboutToAppear --> Child build --> Child onDidBuild。此处体现了自定义组件懒展开特性，即 Parent 执行完 onDidBuild 之后才会执行 Child 组件的 aboutToAppear。

点击 delete Child 按钮，之后再点击一次，可以看到 Log:



在模拟器上运行（为了更好演示，需要在 console.log 的前面加上过滤字符串，如 LifeCycle123），可以尝试推出程序或切换到后台等操作。基本上，顺序是这样的：



4. **通过导航来演示页面的生命周期**

单独一个页面其实是比较难演示页面的 Hide，DisAppear，回退这些动作的效果的。可以加一个新的页面，然后通过导航跳转到这个页面，再回退到之前的页面，这样可以更好的理解页面和组件的生命周期。

增加一个页面 SecondPage.ets，页面布局和 Index.ets 基本一样，仅仅修改一下一些标题性的内容或背景颜色：

```
@Entry
@Component
struct SecondPage {
  @State message: string = '第二个页面';
  @State show: boolean = true

  // 只有被@Entry 装饰的组件才可以调用页面的生命周期
  onPageShow() {
    console.info('LifeCycle123', 'Second Page 调用 onPageShow 函数');
  }
  // 只有被@Entry 装饰的组件才可以调用页面的生命周期
  onPageHide() {
    console.info('LifeCycle123', 'Second Page 调用 onPageHide 函数');
  }

  // 只有被@Entry 装饰的组件才可以调用页面的生命周期
  onBackPress() {
```

```
        console.info('LifeCycle123', 'Second Page 调用 onBackPress 函数');
        return true // 返回 true 表示页面自己处理返回逻辑，不进行页面路由；返回 false 表示使用默认的路由返回逻辑，不设置返回值按照 false 处理
    }

    aboutToAppear(): void {
        console.log('LifeCycle123', 'Second Page 调用 aboutToAppear 函数')
    }

    aboutToDisappear(): void {
        console.log('LifeCycle123', 'Second Page 调用 aboutToDisAppear 函数')
    }

onDidBuild(): void {
    console.log('LifeCycle123', 'Second Page 调用 onDidBuild 函数')
}

    build() {
        RelativeContainer() {
            Text(this.message)
                .id('HelloWorld')
                .fontSize(50)
```

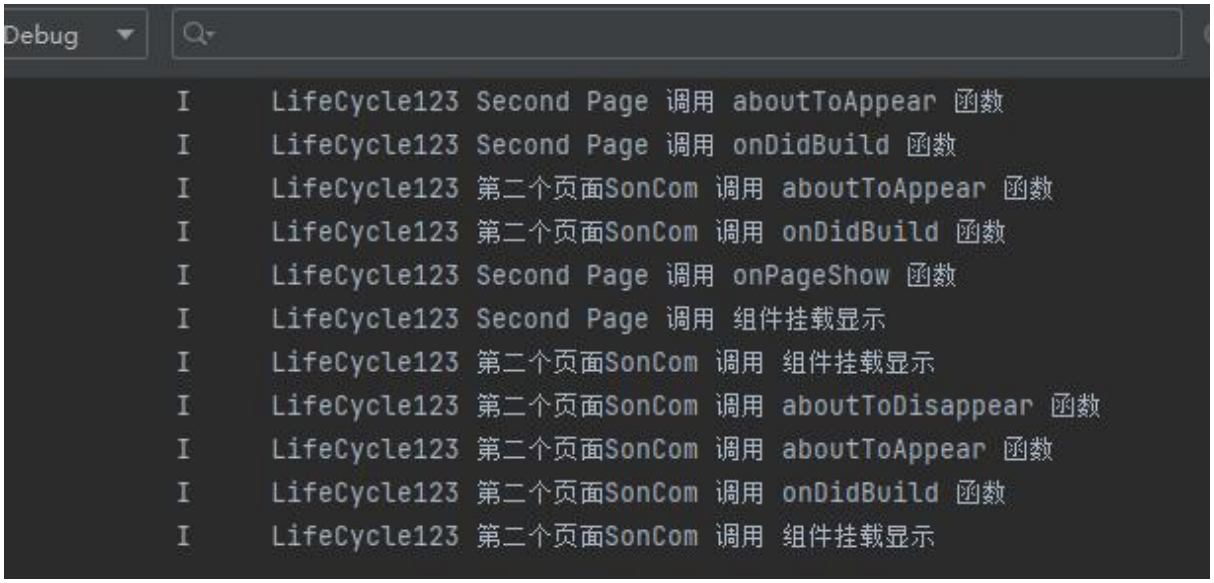
```
            .fontWeight(FontWeight.Bold)
            .alignRules({
              center: { anchor: '__container__', align: VerticalAlign.Top },
              middle: { anchor: '__container__', align: HorizontalAlign.Center }
            })

        Button('切换子组件显示')
            .id('ShowHide')
            .alignRules({
              top: { anchor: 'HelloWorld', align: VerticalAlign.Bottom },
              middle: { anchor: 'HelloWorld', align: HorizontalAlign.Center }
            })
            .onClick(() => {
              this.show = !this.show
            })

        if(this.show) {
            SonCom2()
              .id('SonCom')
              .margin(40)
              .alignRules({
                top: {anchor: 'ShowHide', align: VerticalAlign.Bottom},
```

```
                middle: { anchor: 'HelloWorld', align: HorizontalAlign.Center }
            })
        }
    }
    .height('100%')
    .width('100%')
    .onAppear(() => {
        console.log('LifeCycle123', 'Second Page 调用 组件挂载显示')
    })
  }
}

@Component
struct SonCom2 {
  aboutToAppear(): void {
    console.log('LifeCycle123', '第二个页面 SonCom 调用 aboutToAppear 函数')
  }
  aboutToDisappear(): void {
    console.log('LifeCycle123', '第二个页面 SonCom 调用 aboutToDisappear 函数')
  }
```

```
onDidBuild(): void {
    console.log('LifeCycle123', '第二个页面 SonCom 调用 onDidBuild 函数')
}
  build() {
    Column() {
      Text('我是第二个页面子组件')
        .fontColor(Color.White)
        .fontSize(30)
    }
    .width(200)
    .height(200)
    .backgroundColor(Color.Pink)
    .border({ width: 3 })
    .onAppear(() => {
      console.log('LifeCycle123', '第二个页面 SonCom 调用 组件挂载显示')
    })
  }
}
```

效果：

在这个页面上，单独刷新 Previewer 并对按钮操作两次，也可以看到 Log:



我们回到 Index.ets，添加一个按钮，跳转到第二个页面：

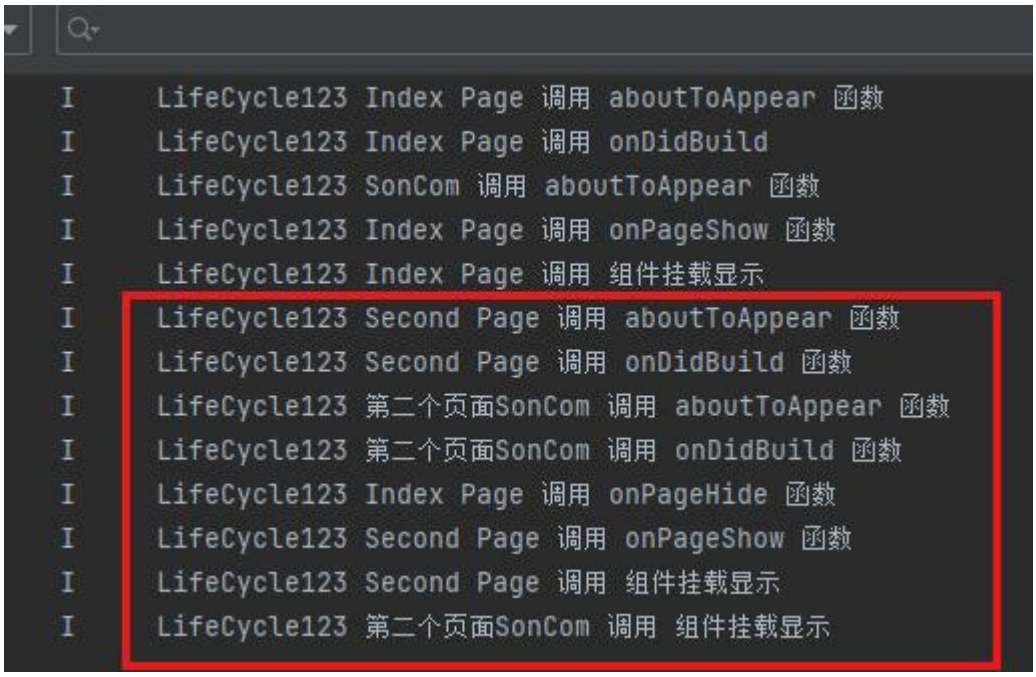
```
Button('跳转到页面二')
  .id('JumpToPage2')
  .margin({top: 20})
  .alignRules({
    top: { anchor: 'SonCom', align: VerticalAlign.Bottom },
    middle: { anchor: 'SonCom', align: HorizontalAlign.Center }
```

```
        })
        .onClick(() => {
          router.pushUrl({url: 'pages/SecondPage'})
        })
```

注意，到记得在 main_pages.json 确保 SecondPage 有加入，而且这里也到导入 router。

刷新 Previewer，Index 的效果：

点击跳转按钮，调到页面二，此时查看 Log:



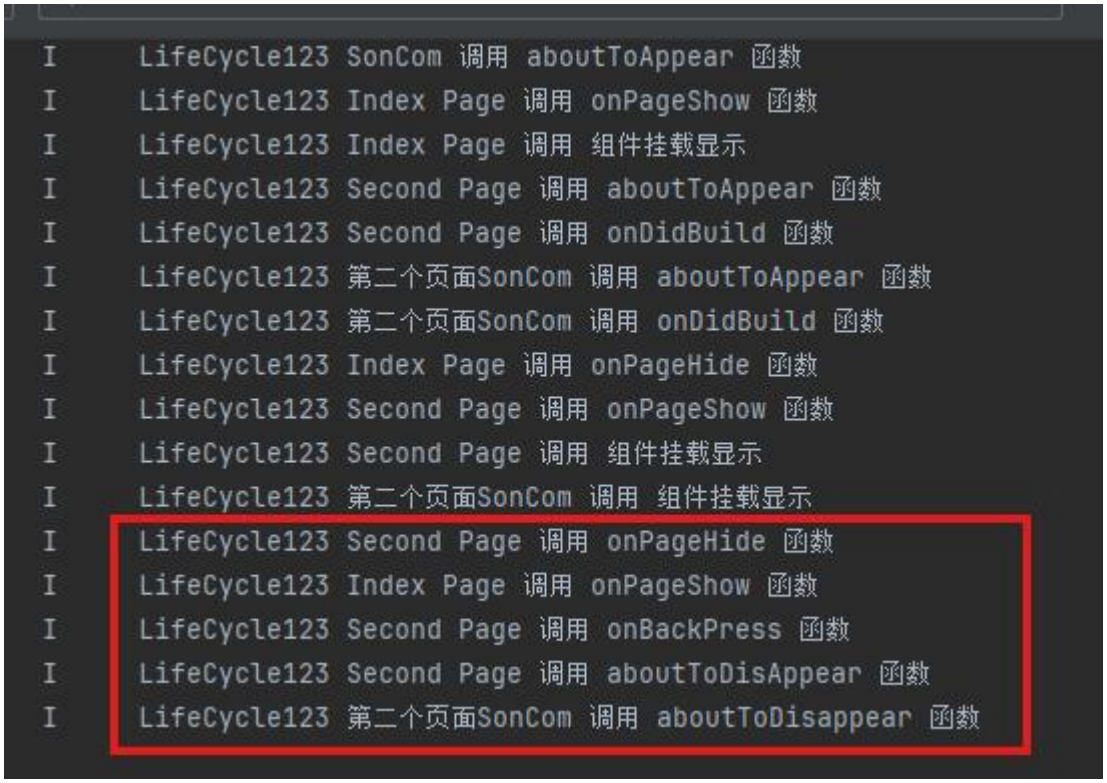
可以看到，页面 1 及其子组件的生命周期，页面 2 及其子组件的生命周期函数显示阶段的都被调用了。

我们给页面 2 的 onBackPress 函数添加回退回 Index 的功能：

```
  onBackPress() {
    router.back()
    console.info('LifeCycle123', 'Second Page 调用 onBackPress 函数');
    return true // 返回 true 表示页面自己处理返回逻辑，不进行页面路由；返回 false 表示使用默认的路由返回逻辑，不设置返回值按照 false 处理
  }
```

就是加了红色字体的这一行。记得这里也要导入 router。

再从 Index.ets 刷新 Previewer，然后点击跳转按钮到页面 2，显示页面 2 之后，点击回退按钮返回到页面 Index，此时查看 Log:



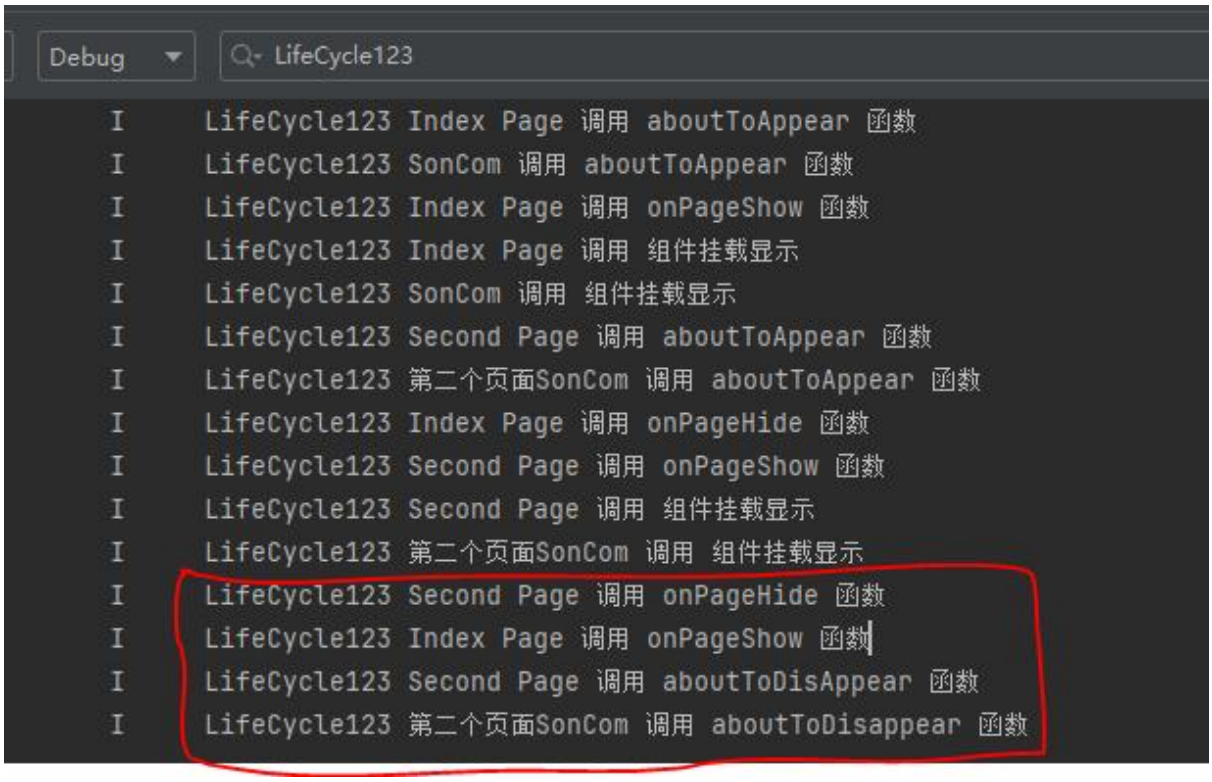
可以看到，首先是调用了 SecondPage 的 Hide，然后 Index 页面的 Show 被调用，然后才是页面 2 的 onBackPress 函数，之后，页面 2 消失（DisAppear），相应的子组件也消失。

如果不用这个回退按钮，而是在页面 2 加一个返回按钮呢？我们去掉 onBackPress 函数中的 router.back()的调用，然后在页面 2 中加入代码：

```
        Button('返回到上一页')
          .id('BackToPage1')
          .margin({top: 20})
          .alignRules({
            top: { anchor: 'SonCom', align: VerticalAlign.Bottom },
            middle: { anchor: 'SonCom', align: HorizontalAlign.Center }
          })
          .onClick(() => {
```

```
            router.back()
        })
```

再从 Index.ets 刷新 Previewer，然后点击跳转按钮到页面 2，显示页面 2 之后，点击“返回到上一页”按钮返回到页面 Index，此时查看 Log:



可以看到，onBackPress 没有被调用，也就是说这个纯粹是和那个回退按钮相关的，和 router 的返回不是一回事。

至此，基本上页面和组件的生命周期函数大家应该清楚了。

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？