

实验十一 样式和结构重用

一、实验目的

1. 了解 DevEco Studio 的使用
2. 学习 ArkTS 语言，掌握样式和结构重用的方法
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

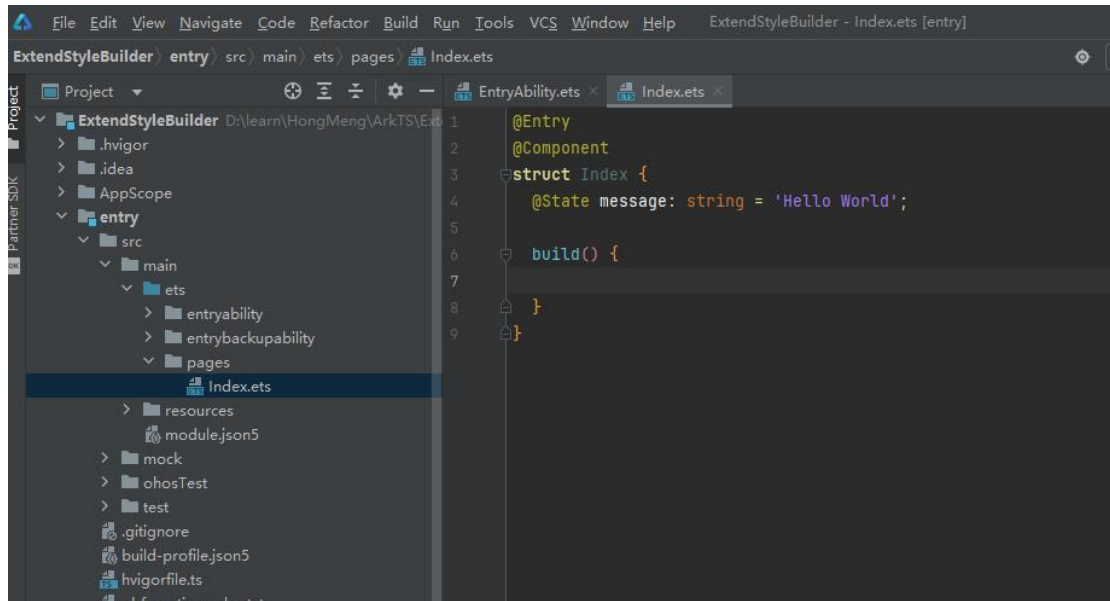
三、实验仪器材料

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

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

配置好项目名称（如 ExtendStyleBuilder），存放位置，设备类型（选择 phone）等，然后点击 Finish 按钮，进入到开发界面，项目创建成功。然后，清理代码，找到 entry > src > main > ets > pages 里面的 Index.ets 文件，将 build() {} 的 {} 里面的代码清空。



在前面的一些实例中，我们会发现，有很多的代码重复，尤其是一些样式和结构。我们可以通过三种重用方式来进行代码的简化：

- @Extend: 扩展组件样式和事件
- @Styles: 抽取通用属性和事件
- @Builder: 自定义构造函数（结构、样式、事件）

下面就通过实例来理解这些样式和结构重用的方式。

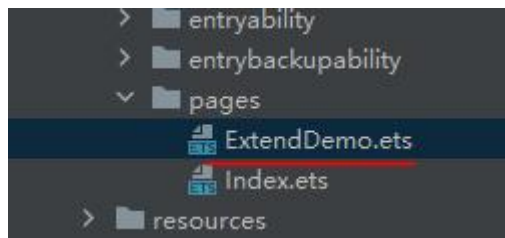
2. @Extend: 扩展组件样式和事件

官方文档:

<https://developer.huawei.com/consumer/cn/doc/harmonyos-guides/arkts-extend>

在一些重复渲染相类似的 UI 组件的场景中，很多组件大部分的样式属性是相同的，只有部分是不同的，比如对于 Swiper 来说，可能每个轮播的内容很多属性是相同的，只有小部分不同。此时可以采用@Extend 来定义扩展样式。我们用实例来演示。

创建一个新的文件 ExtendDemo.ets:



添加代码:

```
@Entry
@Component
struct ExtendDemo {

    @State message: string = '@Extend-扩展组件(样式,事件)';

    build() {

        Column() {

            Text(this.message)

            .fontSize(20)

            .fontWeight(FontWeight.Bold)

            .margin({top: 20, bottom: 20})

            Swiper() {
```

```
Text('1')
```

```
.textAlign(TextAlign.Center)
```

```
.backgroundColor(Color.Orange)
```

```
.fontColor(Color.White)
```

```
.fontSize(30)
```

```
.onClick() => {
```

```
  AlertDialog.show({
```

```
    message: '轮播图 1'
```

```
  })
```

```
})
```

```
Text('2')
```

```
.textAlign(TextAlign.Center)
```

```
.backgroundColor(Color.Green)
```

```
.fontColor(Color.White)
```

```
.fontSize(30)
```

```
.onClick() => {
```

```
  AlertDialog.show({
```

```
    message: '轮播图 2'
```

```
  })
```

```
})
```

```
Text('3')
```

```
.textAlign(TextAlign.Center)
```

```
        .backgroundColor(Color.Gray)

        .fontColor(Color.White)

        .fontSize(30)

        .onClick() => {

            AlertDialog.show({

                message: '轮播图 3'

            })

        })

    }

    .width('100%')

    .height(160)

    }

    .width('100%')

    .height('100%')

    }

}
```

可以看到，对于这个轮播图来说，每个 Text 的属性大部分是相同的，不同的地方在于 Text 文本不同，背景颜色不同，点击后弹窗的内容不同。

效果：



现在，我们在@Entry 前面，定义一个@Extend (Text)：

```
@Extend(Text)
```

```
function bannerItem (bgColor: ResourceColor, msg: string) {
```

```
    .textAlign(TextAlign.Center)
```

```
    .backgroundColor(bgColor)
```

```
    .fontColor(Color.White)
```

```
    .fontSize(30)
```

```
    .onClick() => {
```

```
        AlertDialog.show({
```

```
            message: msg
```

```
        })
```

```
    })
```

```
}
```

注意，`@Extend(Text)`，这里的 `Text` 是表示我们扩展的是 `Text` 相关的属性。

跟着需要定义一个函数，用 `function` 开头，参数是背景颜色和消息，相应的参数在属性或事件中被使用到了，请留意上面代码中红色字体部分。

有了这个扩展之后，在 `Swiper` 中，我们就可以简化了。把 `Swiper` 部分的代码改为：

```
Swiper() {  
  
  Text('1')  
  
    .bannerItem(Color.Orange, '轮播图 1 号')  
  
  Text('2')  
  
    .bannerItem(Color.Brown, '轮播图 2 号')  
  
  Text('3')  
  
    .bannerItem(Color.Green, '轮播图 3 号')  
  
}
```

核心就是，对于 `Text` 组件，我们可以直接调用上面定义的 `Extend` 属性函数，不同的部分用参数传入进去即可。

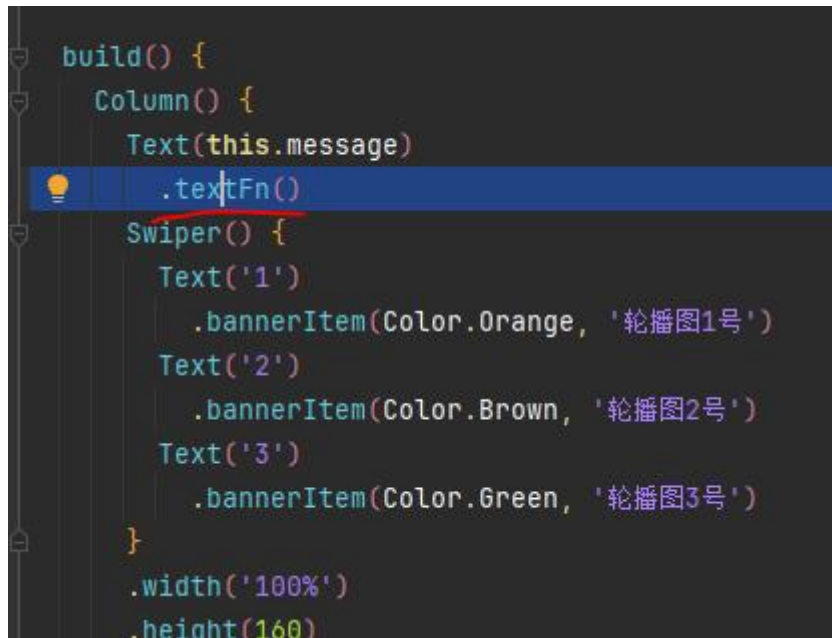
现在刷新 `Previewer`，效果是一样的。

同样，对于标题部分，我们也可以简化，虽然价值没有那么高。我们再定义一个 `@Extend(Text)` 函数：

```
@Extend(Text)  
  
function textFn () {  
  
  .fontSize(20)  
  
  .fontWeight(FontWeight.Bold)  
  
}
```

```
.margin({top: 20, bottom: 20})  
  
}
```

没有传入的参数。然后在页面 build() 里面调用：



```
build() {  
  Column() {  
    Text(this.message)  
    .textFn()  
    Swiper() {  
      Text('1')  
        .bannerItem(Color.Orange, '轮播图1号')  
      Text('2')  
        .bannerItem(Color.Brown, '轮播图2号')  
      Text('3')  
        .bannerItem(Color.Green, '轮播图3号')  
    }  
  }  
  .width('100%')  
  .height(160)  
}
```

刷新 Previewer，效果是一样的。也就是说，这种虽然没有多处用，但是为了简化主程序，也是可以采用的。

总之，可以采用@Extend 去扩展某个组件的属性和事件，从而简化代码。语法：

```
@Extend(组件名)  
  
function 函数名 (参数 1, 参数 2) {  
  
  }  
}
```

官网定义语法：

```
@Extend(UiComponentName) function functionName { ... }
```

然后，在使用组件的时候：

```
组件  
  
  .函数名 (参数 1, 参数 2)
```


当然，像前面的 `textFn()` 那样，也可以没有参数。

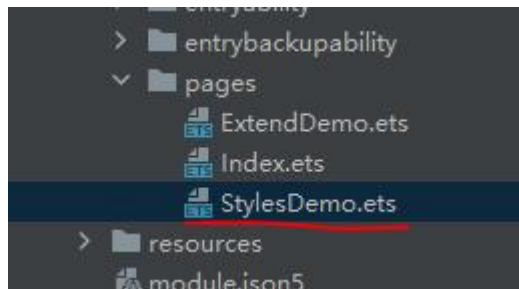
3. @Styles: 抽取通用属性和事件

官方文档:

<https://developer.huawei.com/consumer/cn/doc/harmonyos-guides/arkts-style>

在有些场景中，不同的组件有一些相同的属性，比如 width, height, 背景颜色等等。可以通过@Styles 来抽取通用的属性和事件，从而简化代码。我们用实例来演示。

创建一个新的文件 StylesDemo.ets:



添加代码:

```
@Entry
```

```
@Component
```

```
struct StylesDemo {
```

```
  @State message: string = '@styles';
```

```
  @State bgColor: ResourceColor = Color.Blue
```

```
  build() {
```

```
    Column({ space: 10 }) {
```

```
      Text(this.message)
```

```
        .fontColor(Color.White)
```

```
        .width(100)
```

```
        .height(100)
```

```
.backgroundColor(this.bgColor)
```

```
.onClick() => {
```

```
  this.bgColor = Color.Orange
```

```
})
```

```
Column() {}
```

```
.width(100)
```

```
.height(100)
```

```
.backgroundColor(this.bgColor)
```

```
.onClick() => {
```

```
  this.bgColor = Color.Orange
```

```
})
```

```
Button('按钮')
```

```
.width(100)
```

```
.height(100)
```

```
.backgroundColor(this.bgColor)
```

```
.onClick() => {
```

```
  this.bgColor = Color.Orange
```

```
})
```

```
}
```

```
.width('100%')
```

```

        .height('100%')
    }
}

```

可以看到，对于三个组件 Text, Column, Button 来说，都有相同的 width, height, backgroundColor 属性，然后都有相同的点击事件，点击之后都是改背景颜色。

效果（点击之前是蓝色，点击之后是橙色）：



我们在@Entry 前面添加@Styles 定义：

```

// 1. 全局定义

@Styles function commonStyles () {

    .width(100)

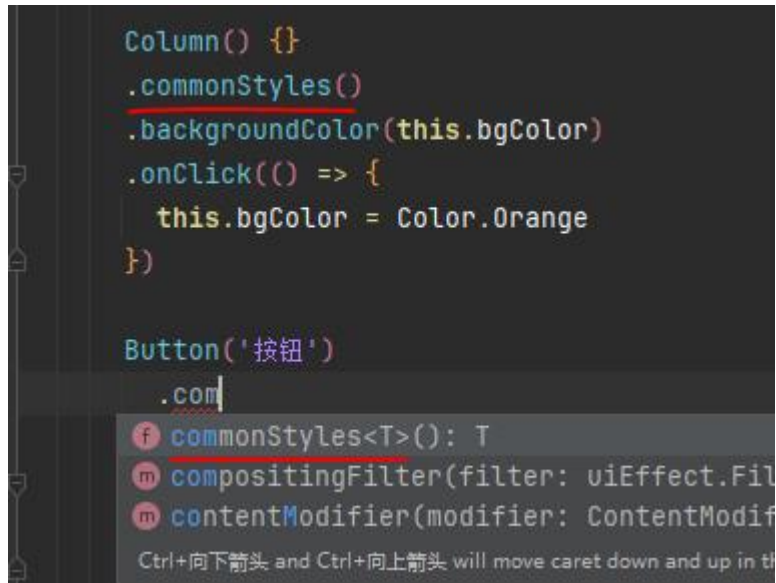
    .height(100)

}

```

定义了一个 commonStyles 函数，里面包含了宽度和高度。注意，@Styles 的函数是不支持传参的。

然后，把 Text, Column, Button 对应的.width(100).height(100)都用.commonStyles()替换掉。注意，在输入代码的时候，系统已经有相应的提示了：



改完之后，刷新 Previewer，效果不变。请思考一下，为什么我们不把背景颜色也放在这个 `commonStyles` 里面呢？

对于背景颜色和点击事件修改背景颜色，我们需要在 `struct` 的内部去定义一个 `@Styles` 函数，因为这样才能够访问到 `this` 后面的数据，在 `build()` 前面添加代码：

// 2. 组件内定义(才能通过 this 访问到自己的状态)

```
@Styles setBg() {  
  
    .backgroundColor(this.bgColor)  
  
    .onClick() => {  
  
        this.bgColor = Color.Green  
  
    })  
  
}
```

然后，修改 `build()` 函数成为：

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

```
.fontColor(Color.White)
```

```
.commonStyles()
```

```
.setBg()
```

```
Column() {}
```

```
.commonStyles()
```

```
.setBg()
```

```
Button('按钮')
```

```
.commonStyles()
```

```
.setBg()
```

```
}
```

```
.width('100%')
```

```
.height('100%')
```

```
}
```

背景相关的部分都改为了 `.setBg()`，也就是我们自己通过 `@Styles` 定义的属性及事件函数。

请思考，为什么 `.fontColor` 不能定义到 `@Styles` 中去？

刷新 Previewer，效果有点变化，因为在 `@Styles` 里面，点击事件中，新的背景颜色是绿色了。总体上，页面的 `build()` 代码更加简洁清晰了。

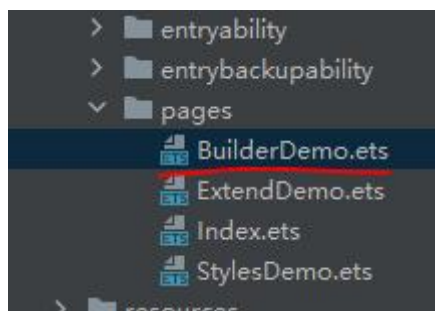
4. @Builder: 自定义构造函数

官方文档:

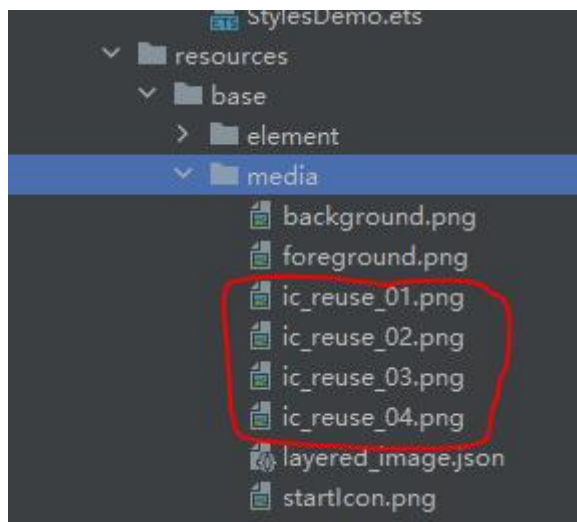
<https://developer.huawei.com/consumer/cn/doc/harmonyos-guides/arkts-builder>

在有些场景中,有些结构被重复使用,比如某些组件的组合被重复使用。这种情况下,可以通过@Builder 来定义这种结构,从而简化代码。当然,对于复杂的结构,最好是通过后面要学习到的自定义组件来处理。

我们用实例来演示。创建一个新的文件 BuilderDemo.ets:



为了更好的效果,我们导入一些图标资源到 resources > base > media 下面:



这四张图片:



在 BuilderDemo.ets 中加入代码:

```
@Entry
```

```
@Component
```

```
struct BuilderDemo {
```

```
    @State message: string = '@Builder';
```

```
    build() {
```

```
        Column({ space: 20 }) {
```

```
            Text(this.message)
```

```
                .fontSize(30)
```

```
        Row() {
```

```
            Row() {
```

```
                Column({ space: 10 }) {
```

```
                    Image($r('app.media.ic_reuse_01'))
```

```
                        .width('80%')
```

```
                    Text('阿里拍卖')
```



```
}
```

```
.width('25%')
```

```
.onClick() => {
```

```
  AlertDialog.show({
```

```
    message: '点了 阿里拍卖'
```

```
  })
```

```
})
```

```
Column({ space: 10 }) {
```

```
  Image($r('app.media.ic_reuse_02'))
```

```
    .width('80%')
```

```
  Text('菜鸟')
```

```
}
```

```
.width('25%')
```

```
.onClick() => {
```

```
  AlertDialog.show({
```

```
    message: '点了 菜鸟'
```

```
  })
```

```
})
```

```
Column({ space: 10 }) {
```

```
Image($r('app.media.ic_reuse_03'))
```

```
.width('80%')
```

```
Text('芭芭农场')
```

```
}
```

```
.width('25%')
```

```
.onClick() => {
```

```
AlertDialog.show({
```

```
message: '点了 芭芭农场'
```

```
})
```

```
})
```

```
Column({ space: 10 }) {
```

```
Image($r('app.media.ic_reuse_04'))
```

```
.width('80%')
```

```
Text('阿里药房')
```

```
}
```

```
.width('25%')
```

```
.onClick() => {
```

```
AlertDialog.show({
```

```
message: '点了 阿里药房'
```

```
})
```

```

    })
  }
}
}

.width('100%')

.height('100%')

}

}

```

效果：



可以看到，对于每一个图标加文字的组合，其实本质上是一样的，都是一个 Column 容器，然后上面是一个图标，跟着是一个文本。

我们可以通过@Builder 来定义一个函数来简化，在@Entry 前面，加上代码：

```
// 全局 Builder
```

@Builder

```
function navItem(icon: ResourceStr, txt: string) {
```

```
    Column({ space: 10 }) {
```

```
        Image(icon)
```

```
        .width('80%')
```

```
        Text(txt)
```

```
    }
```

```
    .width('25%')
```

```
    .onClick() => {
```

```
        AlertDialog.show({
```

```
            message: '点了' + txt
```

```
        })
```

```
    })
```

```
}
```

注意，如果是定义在@Entry 外面的，可以称之为全局 Builder，而此时要注意的是，在这个 Builder 里面，不能访问 struct 里面的成员变量和函数。

另外，可以看到，定义了两个参数，一个是图标，对应的是资源字符串类型，一个是文本，在函数中也用到了这两个参数。

然后，我们修改 build() 函数：

```
build() {
```

```
    Column({ space: 20 }) {
```

```
        Text(this.message)
```

```
.fontSize(30)
```

```
Row() {
```

```
Row() {
```

```
navItem($r('app.media.ic_reuse_01'), '阿里拍卖')
```

```
navItem($r('app.media.ic_reuse_02'), '菜鸟')
```

```
navItem($r('app.media.ic_reuse_03'), '巴巴农场')
```

```
navItem($r('app.media.ic_reuse_04'), '阿里药房')
```

```
}
```

```
}
```

```
}
```

```
.width('100%')
```

```
.height('100%')
```

```
}
```

刷新 Previewer，效果一样。但是可以看到，build() 代码得到了很大的简化。

我们也可以把@Builder 定义在@Entry 或者说@Component 内部。加上我们在 onClick 事件中，需要弹窗里也显示 this.message 里面的内容，比如我们在 onClick 里加上 this.message:

```

// 全局 Builder
@Builder
function navItem(icon: ResourceStr, txt: string) {
  Column({ space: 10 }) {
    Image(icon)
      .width('80%')
    Text(txt)
  }
  .width('25%')
  .onClick(() => {
    AlertDialog.show({
      message: '点了' + txt + this.message
    })
  })
}

```

此时会报错：

```

lems:  File 1  Project Errors 1  Preview Checker
BuilderDemo.ets  D:\learn\HongMeng\ArkTS\ExtendStyleBuilder\entry\src\main\ets\pages 1 problem
Using "this" inside stand-alone functions is not supported (arkts-no-standalone-this) <ArkTSCheck> :12

```

我们需要把这个@Builder 移动到@Component 内部来，放到 build() 前面（红色字体部分）：

```
@State message: string = '@Builder';
```

```
// 局部 Builder
```

```
@Builder
```

```
navItem(icon: ResourceStr, txt: string) {
```

```
  Column({ space: 10 }) {
```

```
    Image(icon)
```

```
      .width('80%')
```

```
    Text(txt)
```

```
}
```

```
.width('25%')
```

```
.onClick() => {
```

```
  AlertDialog.show({
```

```
    message: '点了' + txt + this.message
```

```
  })
```

```
})
```

```
}
```

```
build() {
```

```
  Column({ space: 20 }) {
```

注意，在 `navItem` 前面，我们需要把 `function` 给删除掉。

此时，对于局部的 `@Builder` 函数，调用的时候需要在前面加上 `this.`：

```

build() {
  Column({ space: 20 }) {
    Text(this.message)
      .fontSize(30)

    Row() {
      Row() {
        this.navItem($r('app.media.ic_reuse_01'), '阿里拍卖')
        this.navItem($r('app.media.ic_reuse_02'), '菜鸟')
        this.navItem($r('app.media.ic_reuse_03'), '巴巴农场')
        this.navItem($r('app.media.ic_reuse_04'), '阿里药房')
      }
    }
  }
  .width('100%')
  .height('100%')
}

```

刷新 Previewer，效果一样。

@Builder 可以认为是一个轻量级的自定义组件。

同学们需要重点掌握@Extend 和@Builder，因为这两种对应的场景很常见。@Styles 的场景相对来说在实际项目中比较少。

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？