《鸿蒙北向应用开发基础》之

ArkTS概述

软通教育教学教研部

CONTENTS

【课程目标】

- 了解ArkTS相关概念；
- 掌握ArkTS基本语法；
- 掌握ArkTS的属性、事件等的配置；
- 了解自定义组件的特点及其生命周期函数。

01 ArkTS概述

UI框架介绍

- HarmonyOS提供了一套UI开发框架，即**方舟开发框架（ArkUI框架）**。方舟开发框架为开发者提供了完整的基础设施，包括简洁的UI语法、丰富的UI功能（组件、布局、动画以及交互事件），以及实时界面预览工具等，可以支持开发者进行可视化界面开发。
- HarmonyOS应用的UI开发框架针对不同目的和技术背景的开发者提供了两种开发范式：**基于ArkTS的声明式开发范式、兼容JS的类Web开发范式。**

声明式开发范式	类Web开发范式
概述：ArkTS是HarmonyOS推荐使用的主力应用开发语言，是TypeScript的超集，继承了TS的所有特性。ArkTS从组件、动画和状态管理三个维度提供UI绘制能力。	**概述：**采用HML（搭建布局）、CSS（样式描述）、JavaScript（逻辑处理）三段式开发方式。更符合Web前端开发者的使用习惯，便于快速将已有的Web应用改造成方舟开发框架应用。
应用场景：复杂度较大、团队合作度较高的程序	**应用场景：**界面较为简单的程序应用和卡片
适用人群：移动系统应用开发人员、系统应用开发人员	**适用人群：**Web前端开发人员

ArkTS声明式UI的主要内容

基于ArkTS的声明式开发范式的方舟开发框架提供了构建HarmonyOS应用UI所必需的能力，主要包括：

ArkTS：ArkTS是UI开发语言，在TS基础上扩展了各种装饰器、自定义组件、UI描述机制。

布局：它定义了组件在界面中的位置。布局方式除了基础的线性布局、层叠布局、弹性布局、相对布局、栅格布局外，也提供了相对复杂的列表、宫格、轮播。

组件：包含系统组件（由框架直接提供）和自定义组件（由开发者定义）。系统组件包括按钮、单选框文本等。开发者可以将系统组件组合为自定义组件，通过这种方式将页面组件化为一个个独立的UI单元，实现页面不同单元的独立创建、开发和复用。

页面路由和组件导航：应用可能包含多个页面，可通过页面路由实现页面间的跳转。一个页面内可能存在组件间的导航如典型的分栏，可通过导航组件实现组件间的导航。

图形：框架提供了多种类型图片的显示能力和多种自定义绘制的能力，并支持绘制形状、填充颜色、绘制文本、变形与裁剪、嵌入图片等。

动画：框架提供了丰富的动画能力，除了组件内置动画效果外，还包括属性动画、显式动画、自定义转场动画以及动画API等，开发者可以通过封装的物理模型或者调用动画能力API来实现自定义动画轨迹。

交互事件：是UI和用户交互的必要元素。框架提供了多种交互事件，如通用事件：触摸事件、鼠标事件、键盘按键事件、焦点事件等，基于通用事件进行进一步识别的手势事件：如点击手势、长按手势、拖动手势、捏合手势、旋转手势、滑动手势，以及通过单一手势事件进行组合的组合手势事件。

ArkTS声明式UI的特性

A

基本UI描述：ArkTS定义了各种装饰器、自定义组件、UI描述机制配合UI开发框架中的UI内置组件、事件方法、属性方法等完成UI开发

B

状态管理：ArkTS提供了状态管理机制，UI相关联数据可以在组件内，组件间以及全局范围内传递，甚至可以跨设备传递。数据可以只读单向传递，也可双向传递。

C

UI元素动态构建：ArkTS提供动态构建UI元素的能力，自定义UI结构，复用组件样式，扩展原生组件

D

渲染控制：ArkTS提供了渲染控制能力。支持条件渲染，循环渲染等

ArkTS基本语法

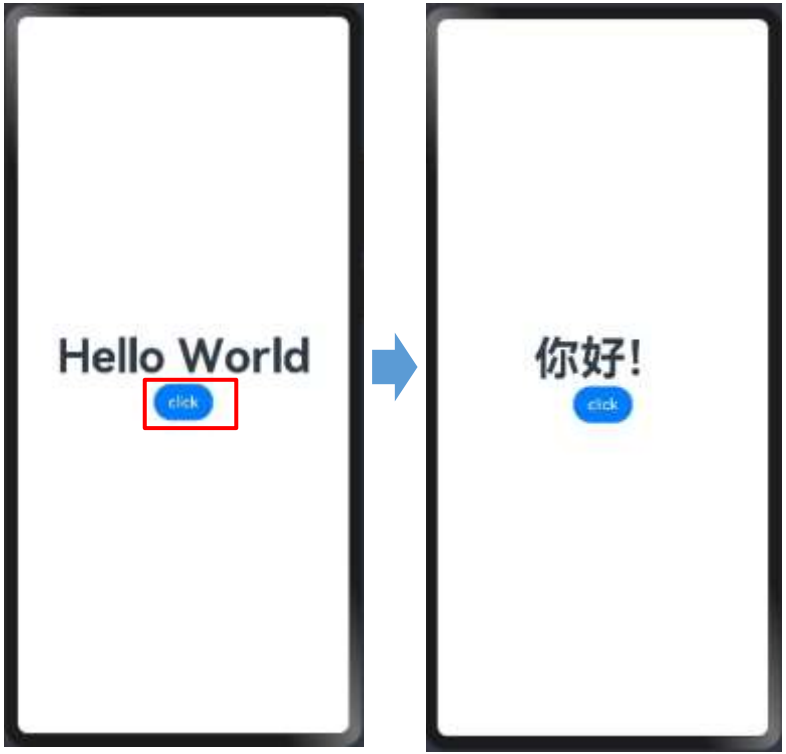
创建一个将HelloWorld项目，并将其pages目录下的Index.ets代码改为下面的内容，实现点击按钮时文本由“Hello World”变成“你好！！”：

```
@Entry
@Component
struct Index{
  @State message: string = 'Hello World'

  build() {
    Row() {
      Column() {
        Text(this.message)
          .fontSize(50)
          .fontWeight(FontWeight.Bold)
        Button("click")
          .onClick(()=>{
            this.message='你好！'
          })
      }
      .width('100%')
    }
    .height('100%')
  }
}
```

1. **装饰器**：使用@符号修饰，用于装饰类、结构、方法以及变量。比如下面三个都是装饰器。
   - @Entry：表示该自定义组件为入口组件；
   - @Component：表示自定义组件；
   - @State表示组件中的状态变量，状态变量变化会触发UI刷新。
2. **自定义组件**：可复用的UI单元，也可组合其他组件。比如被@Component装饰的“struct HelloWorld”表示自定义组件。
3. **UI描述**：以声明式的方式来描述UI的结构，例如build()方法中的代码。
4. **系统组件**：比如：Column、Text、Button，可被开发者直接调用。
5. **属性方法**：用于组件属性配置，支持链式调用，配置多项属性，比如：fontSize()、fontWeight()、height()等
6. **事件方法**：响应组件事件，比如：Button的onClick()

运行结果



02 基本UI描述

2.1 UI描述语言规则

2.2 参数配置

2.3 属性配置

2.4 事件配置

2.5 子组件配置

2.1 UI描述语言规则

在build函数内以链式调用方式进行基本UI描述。UI描述语言需要遵循以下规则：

01

不允许声明本地变量。

02

不允许在UI描述里直接使用console.info，但允许在方法或者函数里使用

03

不允许创建本地的作用域

04

不允许调用除了被@Builder装饰以外的方法

05

不允许switch语法，如果需要使用条件判断，请使用if

UI描述语言规则


```
@Entry
@Component
struct Describe {
  fun1() {....}

  @Builder fun2() {
    Text(`Hello World`)
  }

  build() {
    Row() {
      // 1、不允许声明本地变量
      let a: number = 1;
      // 2、不允许在UI描述里直接使用console.info
      console.info('print debug log');
      Button('click')
        .onClick(()=>{
            console.log('print debug log') // 允许在事件中使用
        })
      // 3、不允许本地作用域
      {
        ...
      }
      // 4、不能调用没有用@Builder装饰的方法
      this.fun1()
      // 可以调用被@Builder装饰的方法
      this.fun2()
    }}}
```

2.2 参数构造配置

组件在创建时，根据组件构造方法的不同，创建组件包含有参数和无参数两种方式

有参数

- 组件接口定义中包含**必选构造参数**，组件后的（）中必须配置相应参数。比如Image组件接口：Image(src: string | PixelMap | Resource)，在创建时必须指定图片的路径。
- **非必选构造参数**，比如Text组件接口：Text(content?: string | Resource)，在创建时参数可选。

无参数

组件接口定义中没有包含必选构造参数，组件后的（）中不需要配置任何内容。比如Divider组件接口：Divider()

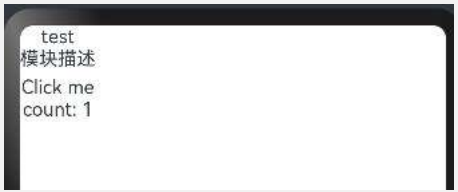
组件参数写法

- 组件中的参数的写法有多种，我们以Text文本组件为例，如下所示：

TextComp.ets

```
@Entry
@Component
struct TextComp {
  // 创建字符串类型的变量
  @State message: string = 'Click me'
  // 创建数字类型的变量
  @State count: number = 0

  build() {
    Column() {
      // string类型的参数
      Text('test')
      // $r形式引入应用资源，可应用于多语言场景
      Text($r('app.string.module_desc'))
      // 无参数形式
      Text()
      //变量用于参数赋值
      Text(this.message)
      //表达式用于参数赋值，其中表达式返回的结果类型必须满足参数类型要求。这里都是数字类型
      Text(`count: ${this.count+1}`)
    }
  }
}
```



使用属性方法配置组件属性

1. 使用“.”运算符连接：

```
Text('hello world')
    .fontSize(12)//设置字体大小
```

2. 链式调用，同时设置多个属性：

```
Image('student1.jpg')
    .alt('张三')
    .width(100)
    .height(100)
```

3. 传递变量或表达式：

```
Image($r('app.media.student1'))
    .alt('张三')
    .width(this.count % 2 === 0 ? 200 : 400)
```

4. 使用枚举类型：

```
Text('hello world')
    .fontSize(20)
    .fontColor(Color.Red)
    .fontWeight(FontWeight.Bold)
```

使用事件方法配置组件支持的事件

1 使用lambda表达式：

```
Button(`点击计数器：${this.counter}`)
        .onClick(() => { this.counter++})
```

2 使用匿名函数，需要使用bind进行绑定，以确保函数体中的this指向当前组件：

```
Button(`点击计数器：${this.counter}`)
        .onClick(function () {
                this.counter++
        }.bind(this))
```

3 使用组件成员函数：

```
buttonClickHander(): void {
        this.counter ++
}
.....
Button(`点击计数器：${this.counter}`)
    .onClick(this.buttonClickHander.bind(this))
```

2.5 子组件配置

- 使用{...}为组件添加子组件描述，通常容器组件会包含多个子组件
- Column、Row、Stack、Grid、List等组件都属于容器组件。

```
Column() {
    Text('Hello')
        .fontSize(20)
    Divider()
    Text(this.myText)
        .fontSize(20)
        .fontColor(Color.Red)
}
```

```
Column() {
  Row() {
    Image('a1.jpg')
      .width(100)
      .height(100)
    Button('click +1')
      .onClick(() => {
        console.info('+1 clicked!')
      })
  }
  Divider()
  Row() {
    Image('a2.jpg')
      .width(100)
      .height(100)
...
  }

  Divider()
  Row() {
...
  }
}
```

03 自定义组件

自定义组件

在ArkUI中，由开发者定义的组件称为自定义组件。在进行 UI 界面开发时，通常不是简单的将系统组件进行组合使用，而是需要考虑代码可复用性、业务逻辑与UI分离等因素。因此，将UI和部分业务逻辑封装成自定义组件是不可或缺的能力。自定义组件的特点如下：

01

可组合

允许开发者组合使用系统组件、及其属性和方法。

02

可重用

自定义组件可以被其他组件重用，并作为不同的实例在不同的父组件或容器中使用。

03

数据驱动UI更新

通过状态变量的改变，来驱动UI的刷新。

自定义组件的结构

自定义组件的结构如下所示：

```
@Entry
@Component
struct MyComponent {
  build() {
      ......
  }
}
```

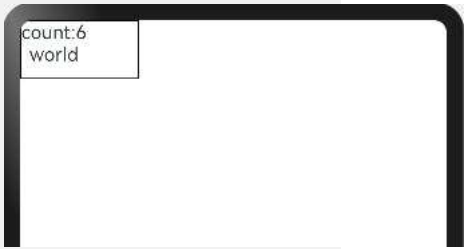
- struct：**struct + 自定义组件名 + {...}**的组合构成自定义组件，不能有继承关系。注意：自定义组件名、类名、函数名不能和系统组件名相同。
- @Component：@Component装饰器仅能装饰struct关键字声明的数据结构。一个struct只能被一个@Component装饰。
- build()函数：build()函数用于定义自定义组件的声明式UI描述，自定义组件必须定义build()函数。
- @Entry：@Entry装饰的自定义组件将作为UI页面的入口。在单个UI页面中，最多可以使用@Entry装饰一个自定义组件。

自定义组件的应用

MyComp.ets

```
@Component
struct MyComp{
  private count:number=3
  private message1:string='hello'
  build(){
    Column(){
      Text(`count:${this.count}`)
      Text(this.message1)
    }
  }
}

@Entry
@Component
struct ChildComp{
  @State message2:string='world'
  build(){
    Column(){
      // 创建MyComp实例，并将MyComp的成员变量count初始化为6，将成员变量message1初始化为this.message2
      // 并为自定义组件添加通用样式
      MyComp({count:6,message1:this.message2})
        .width(100)
        .height(50)
        .borderWidth(1)
    }
  }
}
```



自定义组件生命周期

- 自定义组件生命周期是自定义组件从诞生到销毁的过程
- 自定义组件生命周期回调函数用于通知用户该自定义组件的生命周期
- 回调函数是私有的，在运行时由开发框架在特定时间进行调用，不可从应用程序中手动调用

aboutToAppear函数

- 在创建自定义组件的实例后，在执行其build函数前执行
- 可以在该函数中改变状态变量，更改会在build函数执行时生效
- 可以在该函数中申请资源

aboutToDisappear函数

- 在自定义组件析构销毁之前执行
- 可以在该函数中释放资源，避免资源泄露

【课程小结】

- ArkTS声明式UI的特性：基本UI描述、状态管理、UI元素动态构建能力、渲染控制
- UI描述语言遵循的规则：不允许声明本地变量、不允许在UI描述中直接使用console语句、不允许创建本地作用域、不允许调用除了被@Builder装饰以外的方法、不允许switch语法
- 属性方法的配置通过“.”运算符连接，链式调用，可同时配置多个属性。
- 自定义组件的特点：可组合、可重用、数据驱动UI更新
- 自定义组件的生命周期：**aboutToAppear函数**在创建自定义组件的实例后，在执行其build函数前执行；**aboutToDisappear函数**在自定义组件析构销毁之前执行

匠心育人，学以致用