软通教育
ISOFTSTONE EDUCATION

《鸿蒙北向应用开发基础》之

容器组件介绍（下）

软通教育教学教研部

目录

CONTENTS

1 PART ONE
Scroll组件介绍

2 PART TWO
Tabs组件介绍

3 PART THREE
网格布局Grid

4 PART FOUR
Swiper组件介绍

5 PART FIVE
层叠布局Stack

【课程目标】

- 掌握Scroll、Grid、Swiper、Tabs组件的应用；
- 能够使用Tabs组件制作底部导航栏；

01 Scroll组件介绍

Scroll组件介绍

Scroll为可滚动的容器组件，当子组件内容超出父组件尺寸时，内容可滚动

接口：Scroll(scroller?: Scroller)，该组件的属性及其描述如下：

属性	类型	描述
scrollable	ScrollDirection	设置滚动条方向。默认为Vertical竖直滚动，Horizontal水平滚动，None不滚动
scrollBar	BarState	控制列表滚动条的显示。scrollBar的取值类型为BarState，取值有： • BarState.Auto：按需显示滚动条。此时，当触摸到滚动条区域时显示控件，可上下拖拽滚动条快速浏览内容，拖拽时会变粗。若不进行任何操作，2秒后滚动条自动消失。 • BarState.On：显示滚动条，滚动条一直存在 • BarState.Off;不显示滚动条
scrollBarColor	string | number | Color	滚动条颜色
scrollBarWidth	string | number	滚动条宽度
edgeEffect	EdgeEffect	滚动效果， • EdgeEffect.Spring：弹性物理动效，滑动到边缘后可以根据初始速度或通过触摸事件继续滑动一段距离，松手后回弹。 • EdgeEffect.Fade：阴影效果，滑动到边缘后会有圆弧状的阴影。 • EdgeEffect.None：默认值滑动到边缘后无效果。

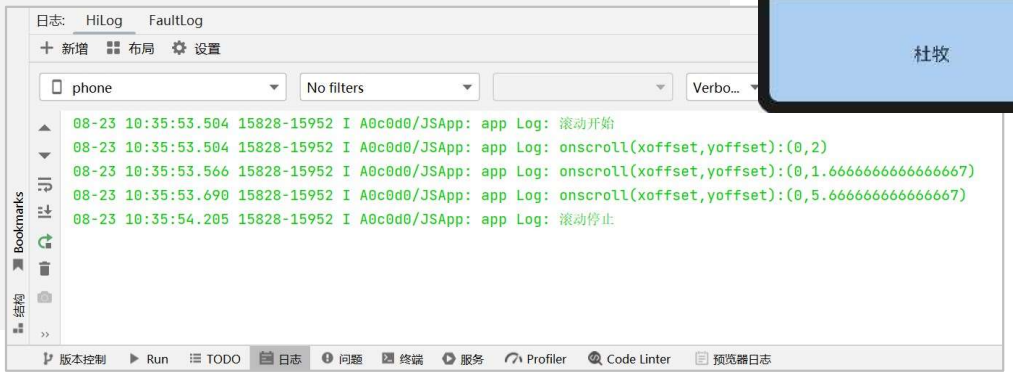
Scroll组件的事件

事件	描述
onScroll(event: (xOffset: number, yOffset: number) => void)	发生滚动时回调，返回滚动时水平和竖直方向的偏移量
onScrollEdge(event: (side: Edge) => void)	滚动到边缘
onScrollEnd (event: () => void)	滚动停止事件回调。
onScrollStart(event: () => void)	滚动开始时触发
onScrollStop(event: () => void)	滚动停止时触发

属性及事件应用

ScrollDemo1.ets

```
@Entry
@Component
struct ScrollDemo1 {
  private students: string[] = ['刘备','关羽','张飞','李白','杜甫','王维','杜牧','秦始皇','康熙','成吉思汗', '武则天']
  build() {
    Scroll() {
      List({space:20}){
        ForEach(this.students,item=>{
          ListItem(){
            Text(item).fontSize(20)
          }.width('100%').height(130).backgroundColor(0xabcdef)
        })
      }
    }
    .scrollable(ScrollDirection.Vertical)   //滚动条方向：垂直方向
    .scrollBar(BarState.On)  //滚动条常驻显示
    .scrollBarColor(Color.Black)  //设置滚动条颜色为黑色
    .scrollBarWidth(10)  //设置滚动条宽度为10，宽度始终为设置的宽度
    //滚动开始时执行
    .onScrollStart(()=>{console.log('滚动开始')})
    // 获取滚动中的水平和竖直方向的偏移量
    .onScroll((xOffset: number,yOffset: number) => {
      console.info('onscroll(xoffset,yoffset):(' + xOffset + ',' + yOffset +')')
    })
    // 当滚动到边界时执行
    .onScrollEdge((side: Edge) => {console.info('滚动到边界')})
    // 当滚动停止时执行
    .onScrollEnd(() => { console.info('滚动停止')})
  }
}
```



Scroller控制器

Scroller滚动控制器，控制某个组件的滚动，可绑定到List、Scroll、ScrollBar、Grid组件上，使用时需要先导入该控制器对象。该对象的方法有：

方法	描述
scrollTo(value: { xOffset: number | string, yOffset: number | string, animation?: { duration: number, curve: Curve } }): void	滑动到指定位置，参数**xOffset**：必填，水平滚动偏移量；**yOffset**：必填，垂直滚动偏移量；animation，非必填，配置动画
scrollPage(value: { next: boolean, direction?: Axis }): void	滚动到下一页或者上一页，参数说明如下 • next：必填，是否向下翻页。true表示向下翻页，false表示向上翻页。 • direction：非必填，设置滚动方向。Axis.Vertical垂直方向，Axis.Horizontal水平方向
scrollToIndex(value: number): void	滑动到指定Index，仅对Grid、List组件有效
currentOffset()	返回当前滚动的偏移量，xOffset表示水平滑动偏移量，yOffset表示竖直滑动偏移量
scrollEdge(value: Edge): void	滚动到容器边缘，不区分滚动轴方向，Edge.Top和Edge.Start表现相同，Edge.Bottom和Edge.End表现相同。
scrollBy(dx: Length, dy: Length): void	滚动指定距离，参数有两个，分别为水平方向滚动距离、垂直方向滚动距离

ScrollDemo.ets


```
@Entry
@Component
struct ScrollDemo2 {
  // 创建scroller对象
  scroller: Scroller =  new Scroller()
  private students: string[] = ['刘备','关羽','张飞','李白','杜甫','王维','杜牧','秦始皇','康熙','成吉思汗', '武则天']
  build() {
    Stack({ alignContent: Alignment.TopStart }) {
      Scroll(this.scroller) {
        List({space:20}){
          ForEach(this.students,item=>{
            ListItem(){
              Text(item).fontSize(20)
            }.width('100%').height(130).backgroundColor(0xabcdef)
          })
        }
      }


      // 滚动指定距离
      Button('向下滚动:100').margin({ top:10, left: 10 })
        .onClick(() => {
          this.scroller.scrollBy(0,100)
        })
      // 滚动到指定位置
      Button('滚动到:200位置').margin({ top:60, left: 10 })
        .onClick(() => {
          this.scroller.scrollTo({ xOffset: 0, yOffset: 200 })
        })
```

ScrollDemo2.ets

```
      // 滚动到容器边缘，可回到顶部或底部
      Button('回到顶部').margin({ top:110, left: 10 })
        .onClick(() => {
          this.scroller.scrollEdge(Edge.Top)
        })
      // 滚动到下一页或上一页，true表示下一页
      Button('下一页').margin({ top:160, left: 10 })
        .onClick(() => {
          this.scroller.scrollPage({ next: true })
        })

      // 获取当前的滚动偏移量
      Button('当前的滚动偏移量').margin({ top:220, left: 10 })
        .onClick(() => {
          // 定义变量接收返回结果
          var obj=this.scroller.currentOffset()
          console.info('当前的偏移量为：('+obj.xOffset+','+obj.yOffset+')')
        })
    }
  }
}
```

02 Tabs组件介绍

2.1 组件介绍

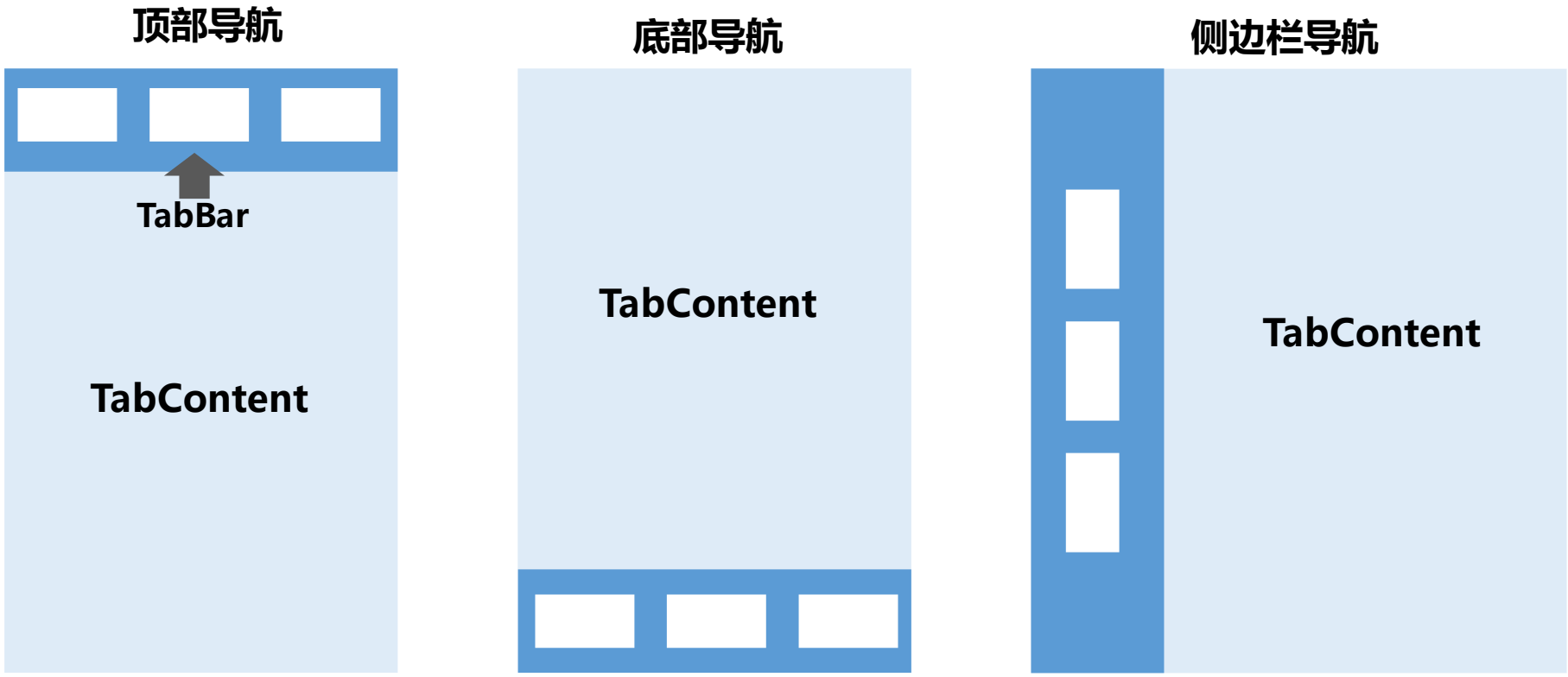
2.2 组件简单应用

2.3 组件属性说明

2.4 Tabs组件实现底部导航栏

2.1 Tabs组件介绍

当页面信息较多时，为了让用户能够聚焦于当前显示的内容，需要对页面内容进行分类，提高页面空间利用率。Tabs组件可以在一个页面内快速实现视图内容的切换，提升查找效率，精简用户单次获取到的信息量。Tabs组件的页面组成包含两个部分，分别是TabContent（内容页）和TabBar（导航页签栏）。其布局分类如下：



Tabs组件介绍

Tabs组件

- 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。
- 实现侧边导航栏需要设置Tabs的属性**vertical为true**。在顶部导航栏中设置vertical为true则导航栏在左侧，在底部导航栏中设置vertical为true则导航栏在右侧，默认在左侧。侧边导航栏多用于平板横屏界面

TabContent组件

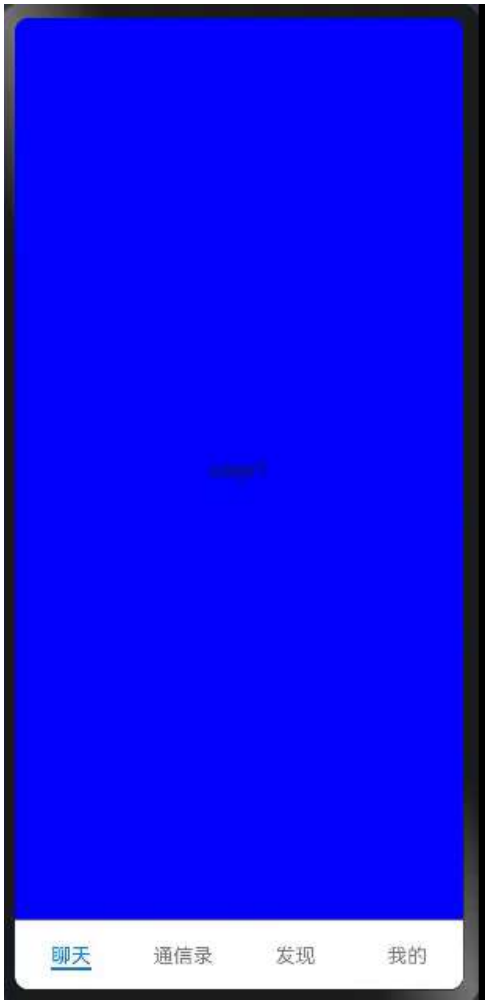
在Tabs组件中使用花括号包裹TabContent，每一个TabContent对应的内容需要有一个**页签，通过**TabContent的**tabBar属性进行配置**。

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。

2.2 Tabs组件的基本应用

底部导航

```
build(){
  Tabs({barPosition:BarPosition.End}){
    TabContent(){
      Text('page1')
    }.backgroundColor(Color.Blue).tabBar('聊天')

    TabContent(){
      Text('page2')
    }.backgroundColor(Color.Yellow).tabBar('通信录')

    TabContent(){
      Text('page3')
    }.backgroundColor(Color.Gray).tabBar('发现')

    TabContent(){
      Text('page4')
    }.backgroundColor(Color.Green).tabBar('我的')
  }
}
```

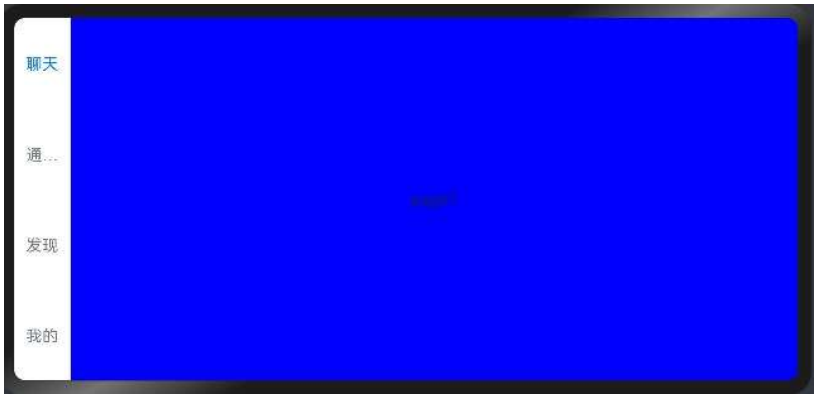


Tabs组件简单应用

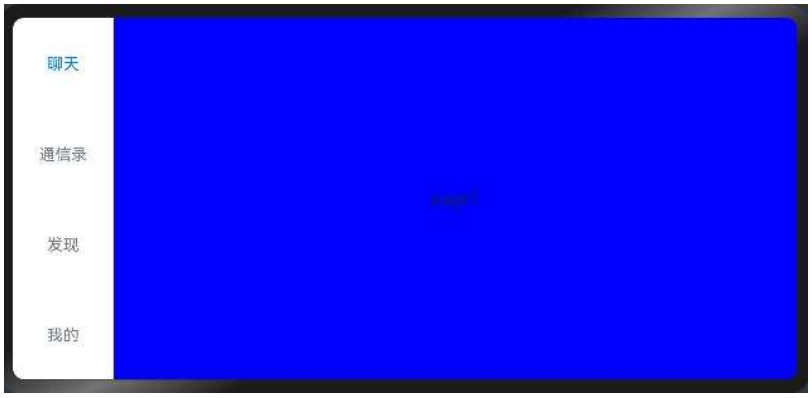
侧边栏导航

```
build(){
  Tabs(){
    TabContent(){
      Text('page1')
    }.backgroundColor(Color.Blue).tabBar('聊天')

    TabContent(){
      Text('page2')
    }.backgroundColor(Color.Yellow).tabBar('通信录')

    TabContent(){
      Text('page3')
    }.backgroundColor(Color.Gray).tabBar('发现')

    TabContent(){
      Text('page4')
    }.backgroundColor(Color.Green).tabBar('我的')
  }.vertical(true).barWidth(100)
}
```

可通过barWidth和barHeight属性设置侧边栏的宽度和高度



无barWidth属性



有barWidth属性

2.3 Tabs组件的属性介绍

属性	说明
vertical()	值为true实现侧边导航栏。默认值为false，表明内容页和导航栏垂直方向排列。
barWidth()	设置导航栏的宽度
barHeight()	设置导航栏的高度
scrollable()	限制导航栏的滑动切换。 • 默认值为true，表示可以滑动； • false表示限制滑动，此时只能通过单击进行界面切换
barMode()	控制导航栏是否可以滚动。用法barMode(BarMode.Scrollable) • 默认值为Fixed，固定导航栏不可滚动，无法被拖拽滚动，内容均分tabBar的宽度； • 设置为Scrollable即可设置为可滚动导航栏，应用于内容分类较多，屏幕宽度无法容纳所有分类页签的情况下。

2.4 自定义导航栏

介绍

Tabs组件系统默认情况下是采用下划线标志当前活跃的页签，而自定义导航栏需要自行实现相应的样式，用于区分当前活跃页签和未活跃页签。比如在聊天软件的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。

应用

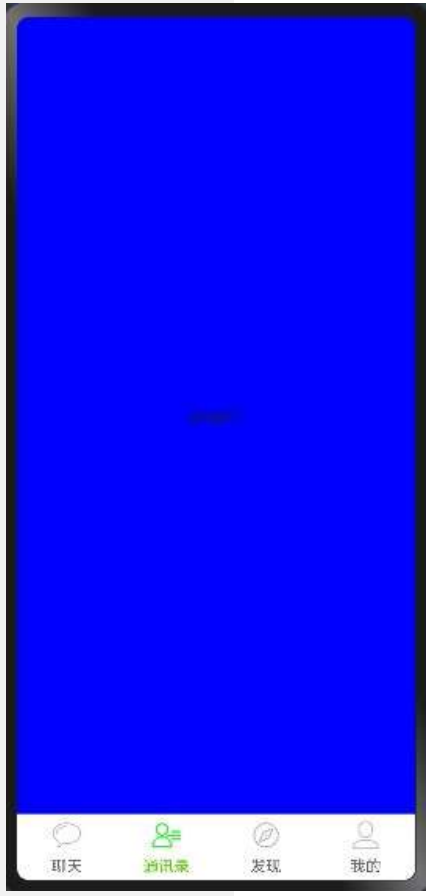
设置自定义导航栏需要**使用tabBar的参数**。

- 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。**

自定义导航栏应用

下面通过Tabs组件自定义导航栏实现与聊天软件相同的底部效果。在pages目录下创建一个TabsBar.ets

```
@Entry
@Component
struct TabsBar{
  @State menuIndex: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,currentIndex:number,imgActive:Resource,img:Resource){
   Column({space:5}){
     Image(this.menuIndex===currentIndex ? imgActive : img).width(30).height(30)
     Text(title).fontColor(this.menuIndex === currentIndex ? '#58d400' : '#6B6B6B').fontSize(14)
   }.onClick(()=>{
     this.menuIndex=currentIndex
   })
 }
 build(){
     Tabs({barPosition:BarPosition.End}){
     TabContent(){
       Text('page1')
     }.backgroundColor(Color.Blue)
     // 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('聊天', 0, $r('app.media.msg_active'), $r('app.media.msg')))

     TabContent(){
       Text('page2')
     }.backgroundColor(Color.Yellow).tabBar(this.TabBuilder('通讯录', 1, $r('app.media.addressbook_active'), $r('app.media.addressbook')))

     TabContent(){
       Text('page3')
     }.backgroundColor(Color.Gray).tabBar(this.TabBuilder('发现', 2, $r('app.media.discover_active'), $r('app.media.discover')))

     TabContent(){
       Text('page4')
     }.backgroundColor(Color.Green).tabBar(this.TabBuilder('我的', 3, $r('app.media.my_active'), $r('app.media.my')))
   }.barHeight(60)
 }
}
```



问题及解决方法（1）

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。**

```
@Entry
@Component
struct TabsBar{
  @State menuIndex:number=0
  private tabsController : 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,currentIndex:number,imgActive:Resource,img:Resource){
    Column({space:5}){
      Image(this.menuIndex===currentIndex ? imgActive : img).width(30).height(30)
      Text(title).fontColor(this.menuIndex === currentIndex ? '#58d400' : '#6B6B6B').fontSize(14)
    }
    .onClick(()=>{
      this.menuIndex=currentIndex
      this.tabsController.changeIndex(this.menuIndex)
    })
  }
  build(){
    Tabs({barPosition:BarPosition.End,controller:this.tabsController}){
      TabContent(){
        Text('page1')
      }.backgroundColor(Color.Blue)
      // 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('聊天', 0, $r('app.media.msg_active'), $r('app.media.msg')))
      ......
    } .barHeight(60)
  }
}
```

问题及解决方法（2）

问题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。

```
@Entry
@Component
struct TabsBar{
  @State menuIndex:number=0
  private tabsController : 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,currentIndex:number,imgActive:Resource,img:Resource){..... }
  build(){
      Tabs({barPosition:BarPosition.End,controller:this.tabsController}){
     TabContent(){
       Text('page1')
     }.backgroundColor(Color.Blue)
     // 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('聊天', 0, $r('app.media.msg_active'), $r('app.media.msg')))
     .....
   }.barHeight(60)
   .onChange((index)=>{
     this.menuIndex=index
   })
  }
}
```

03 网格布局Grid

3.1 网格布局介绍

3.2 行列数量、占比、行列间距

3.3 子组件占行列数

3.4 构建可滚动的网格

3.1 网格布局介绍

网格布局（GridLayout）实现页面均分以及子组件占比控制，网格布局通过Grid容器组件和GridItem子组件实现。**Grid容器设置网格布局参数，GridItem定义子组件特征**

1. 容器组件尺寸发生变化时，所有子组件及其间距会按比例调整，实现布局自适应能力

2. 可以定义行数和列数，以及每行和每列尺寸占比

4. 可以设置子组件跨行和跨列

3. 可以设置子组件行列间距

3.2 行列数量、占比、行列间距

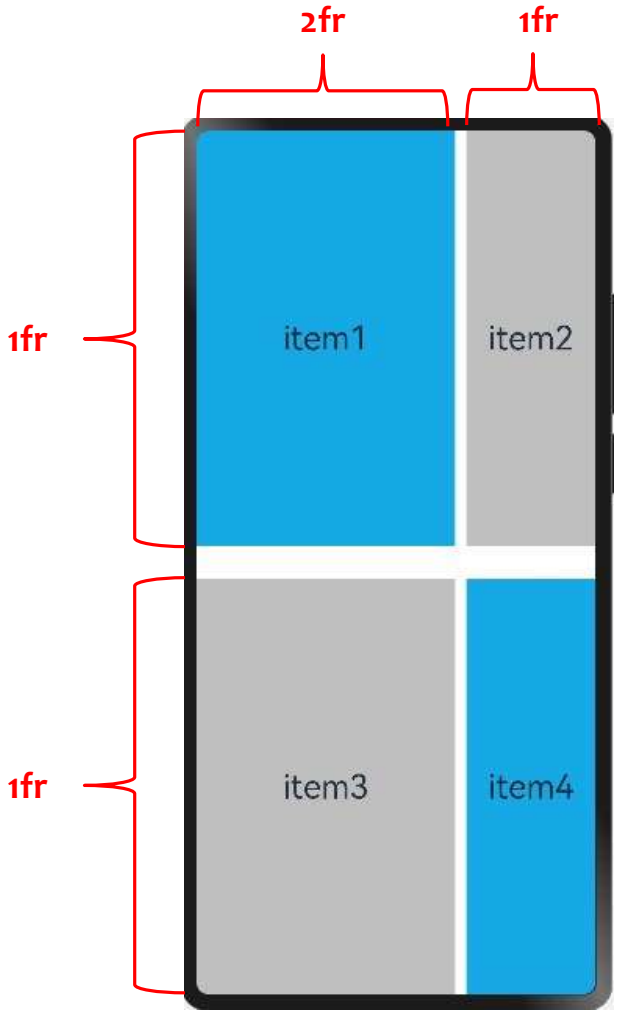
通过设置行列数量与尺寸占比可以确定网格布局的整体排列方式，Grid容器组件的属性如下：

属性	描述
columnsTemplate	列数量及占比 columnsTemplate('1fr 1fr 1fr 1fr')：数字+fr并使用空格分隔，数字表示列宽占比，fr为比例单位
rowTemplate	行数量及占比 rowsTemplate('1fr 1fr')：数字+fr并使用空格分隔，数字表示行宽占比，fr为比例单位
columnsGap	列与列之间的间隔
rowsGap	行与行之间的间隔

行列数量、占比、行列间距

GridLayoutDemo1.ets

```
build() {
  Grid(){
    GridItem(){
      Text('item1').fontSize(30)
    }.backgroundColor('#17A8E6')
    GridItem(){
      Text('item2').fontSize(30)
    }.backgroundColor('#BFBFBF')
    GridItem(){
      Text('item3').fontSize(30)
    }.backgroundColor('#BFBFBF')
    GridItem(){
      Text('item4').fontSize(30)
    }.backgroundColor('#17A8E6')
  }
  // 设置两行两列的网格，行高都相等，第一列的宽度是第二列的2倍
  .rowsTemplate('1fr 1fr')
  .columnsTemplate('2fr 1fr')
  // 列与列之间的间隔是10
  .columnsGap(10)
  // 行与行之间的间隔是30
  .rowsGap(30)
}
```



3.3 子组件占行列数

除了设置等比例网格外，有时还会有“合并单元格”的情况，可以通过设置GridItem的rowStart、rowEnd、columnStart和columnEnd属性可以实现单个网格横跨多行或多列的场景。GridItem组件的属性如右表：

属性	描述	
rowStart	设置起始行编号	行列编号都从1开始
rowEnd	终止行编号	
columnStart	起始列编号	
columnEnd	终止列编号	

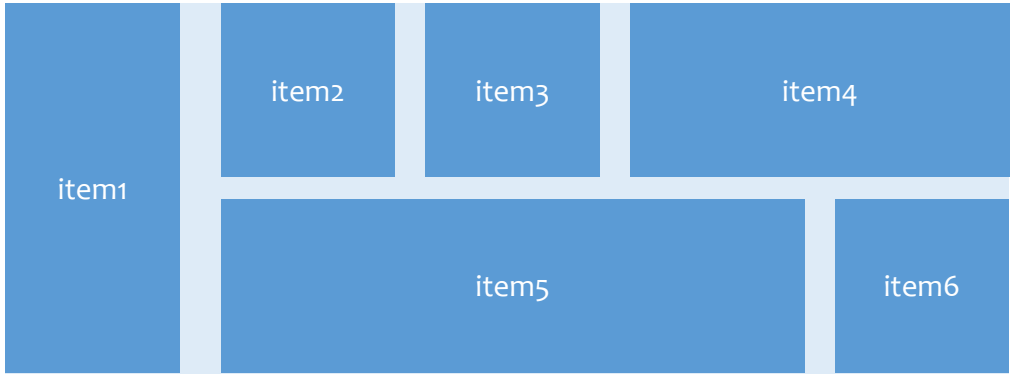
子组件占行列数

比如要将一个两行五列的网格合并为如下图所示的不均匀网格布局。实现思路如下

定义一个2行5列的网格

定义6个GridItem子组件

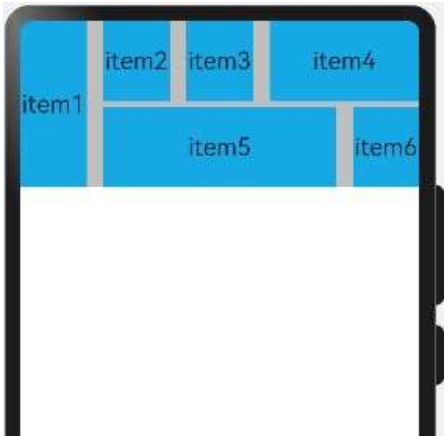
设置需要合并的子组件的开始和结束行列编号



子组件占行列数

GridLayoutDemo2.ets

```
build() {
  Grid() {
    GridItem() {
      Text('item1').fontSize(22)
    }.backgroundColor('#17A8E6').rowStart(1).rowEnd(2) // 子组件从第1行到第2行
    GridItem() {
      Text('item2').fontSize(22)
    }.backgroundColor('#17A8E6')
    GridItem() {
      Text('item3').fontSize(22)
    }.backgroundColor('#17A8E6')
    GridItem() {
      Text('item4').fontSize(22)
    }.backgroundColor('#17A8E6').columnStart(4).columnEnd(5) // 从第4列到第5列
    GridItem() {
      Text('item5').fontSize(22)
    }.backgroundColor('#17A8E6').columnStart(2).columnEnd(4) // 从第2列到第4列
    GridItem() {
      Text('item6').fontSize(22)
    }.backgroundColor('#17A8E6')
  }
  // 定义两行五列的网格
  .columnsTemplate('1fr 1fr 1fr 1fr 1fr')
  .rowsTemplate('1fr 1fr')
  .columnsGap(15)
  .rowsGap(5)
  .width('100%')
  .backgroundColor('#BFBFBF')
  .height(150)
}
```

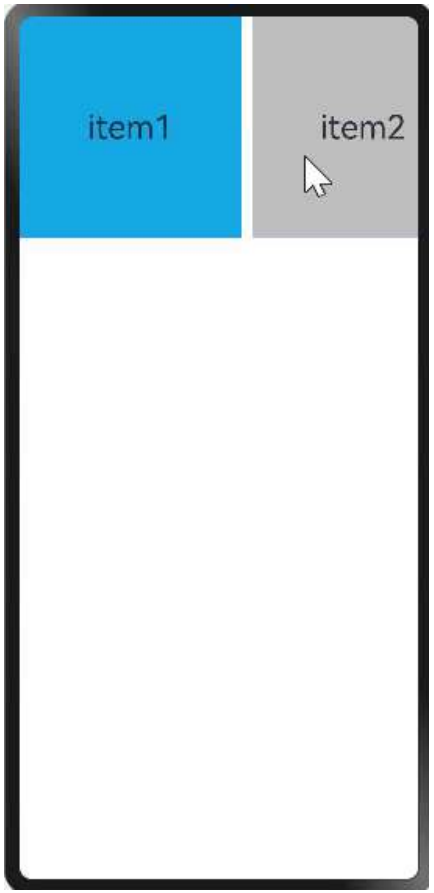


- 在设置Grid的行列数量与占比时，如果仅设置rowsTemplate或columnsTemplate属性，网格单元按照设置的方向排列，超出Grid显示区域后，Grid拥有可滚动能力。
- 如果设置的是columnsTemplate，Grid的滚动方向为垂直方向；如果设置的是rowsTemplate，Grid的滚动方向为水平方向。

构建可滚动的网格

GridLayoutDemo3.ets

```
build() {
    Grid(){
      GridItem(){
        Text('item1').fontSize(30)
      }.backgroundColor('#17A8E6').width(200).height(200)
      GridItem(){
        Text('item2').fontSize(30)
      }.backgroundColor('#BFBFBF').width(200).height(200)
      GridItem(){
        Text('item3').fontSize(30)
      }.backgroundColor('#17A8E6').width(200).height(200)
      GridItem(){
        Text('item4').fontSize(30)
      }.backgroundColor('#BFBFBF').width(200).height(200)
    }
    // 滚动方向为水平方向，1行
    .rowsTemplate('1fr')
    .columnsGap(10)
    .height(200)
```



04 Swiper组件介绍

Swiper组件介绍

Swiper组件提供滑动轮播能力，本身作为容器组件来使用，在自身尺寸属性未被设置时，会自动根据子组件的大小设置自身的尺寸。

接口：Swiper(controller?: SwiperController)

参数

controller：SwiperController，给组件绑定一个控制器，用来控制组件翻页，该控制器的方法有

- showNext(): void，下一页
- showPrevious(): void，上一页
- finishAnimation(callback?: () => void): void，停止播放动画，动画结束回调callback函数

事件

onChange(event: (index: number) => void)：当前显示的组件索引变化时触发，返回值为当前显示组件的索引值

Swiper组件属性

属性	类型	描述
index	number	当前在容器中显示的子组件的索引值，默认为0，即第一个子组件
autoPlay	boolean	是否自动播放，默认为false，不自动播放；true自动播放
interval	number	子组件之间播放的时间间隔，单位毫秒，默认为3000
loop	boolean	循环播放，默认true，值为false时则在第一页或最后一页时，无法继续向前或者向后切换页面
duration	number	子组件切换动画时长，单位毫秒，默认：400
vertical	boolean	是否纵向滑动，默认为false即横向滑动，true表示纵向滑动
disableSwipe	boolean	是否滑动切换，默认为false，可以滑动切换；true表示禁止滑动切换
indicatorstyle	{left?:Length, top?:Length, right?:Length bottom?: Length,size?:Length,mask?: boolean, color?: Resourcecolor, selectedColor?: ResourceColor}	导航点样式： • top、bottom、left、right：导航点据父组件上下左右方向上的距离。 • size：导航点大小， • mask：是否显不导航点蒙层样式 • color：导航点颜色 • selectedColor：导航点选中时颜色
displayCount	number	在一个页面内同时显示多个子组件

Swiper组件应用

SwiperDemo.ets

```
@Entry
@Component
struct SwiperDemo {
  private controller: SwiperController = new SwiperController()
  @State imgList:Resource[]=[$r('app.media.swiper0'), $r('app.media.swiper1'), $r('app.media.swiper2'), $r('app.media.swiper3'), $r('app.media.swiper4')]

  build() {
    Column({ space: 10 }) {
      Swiper(this.controller) {
        ForEach(this.imgList, (item: Resource) => {
          Image(item).width('95%').height('70%')
        })
      }
      .index(2)                    // 页面加载成功后，显示第3张图片
      .autoPlay(true)              // true表示自动播放
      .interval(4000)              // 每隔4秒切换下一张图片
      .loop(true)                  // 设置循环播放
      .duration(20)                // 子组件切换的速度为20毫秒
      .disableSwipe(true)  // 禁止滑动切换
      // 设置导航点的样式，位置居左10，大小为40，导航点的颜色为白色，选中时的颜色为红色
      .indicatorStyle({left:10,size:40,color:Color.White,selectedColor:Color.Red})
      // .vertical(true)纵向滑动

      Row({ space: 10 }) {
        Button('Next>>')
          .onClick(() => {
            this.controller.showNext()// 控制组件翻页，上一页
          })
        Button('<<Previous').onClick(() => {
          this.controller.showPrevious() // 下一页
        })
      }.margin(8)
    }.width('100%')
    .margin({ top: 8 })}}
```

横向滑动

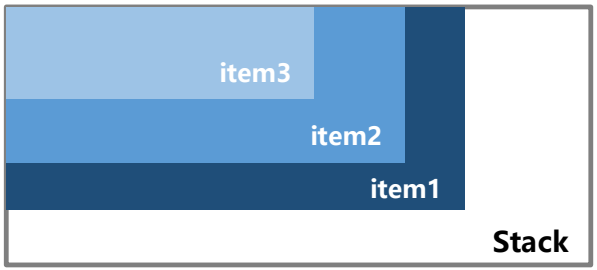
纵向滑动

05 层叠布局Stack

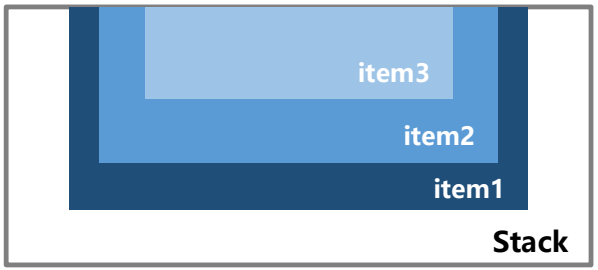
层叠布局介绍

- 层叠布局（StackLayout）在屏幕的某块区域提供组件的重叠展示，层叠布局通过Stack容器实现，容器内可包含各种子组件，子组件依次入栈，后面的元素覆盖前面的元素，子元素可以叠加，也可以设置位置。
- 在Stack组件中子组件根据自己的大小默认进行居中堆叠。子元素被约束在Stack下，进行自己的样式定义以及排列。
- Stack组件通过alignContent参数实现位置的相对移动，类型为Alignment。子元素在容器内的对齐方式有9种方式：

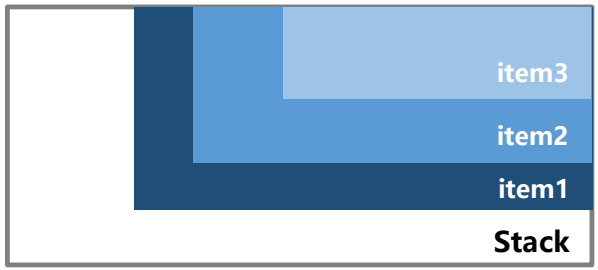
对齐方式



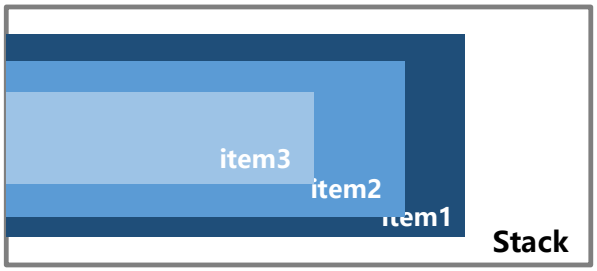
TopStart



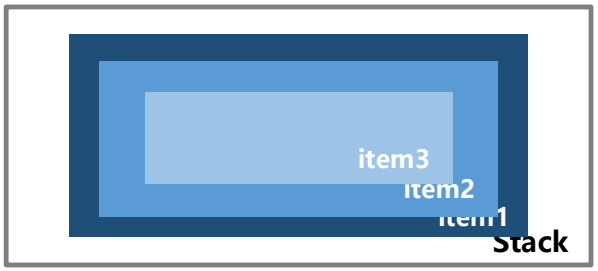
Top



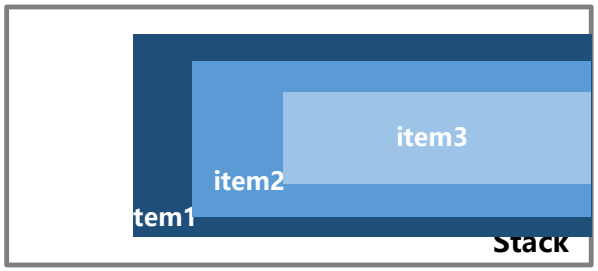
TopEnd



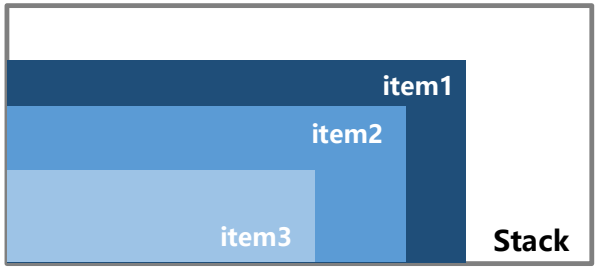
Start



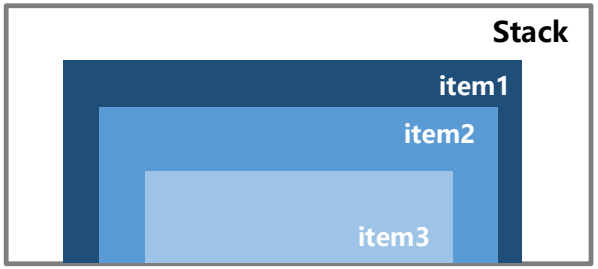
Center



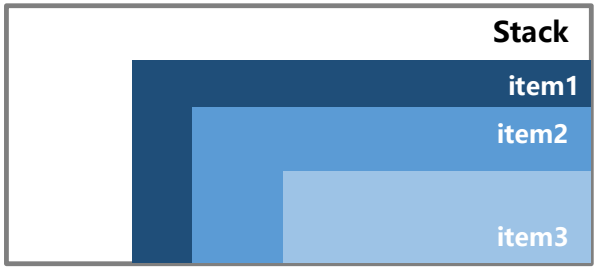
End



BottomStart



Bottom

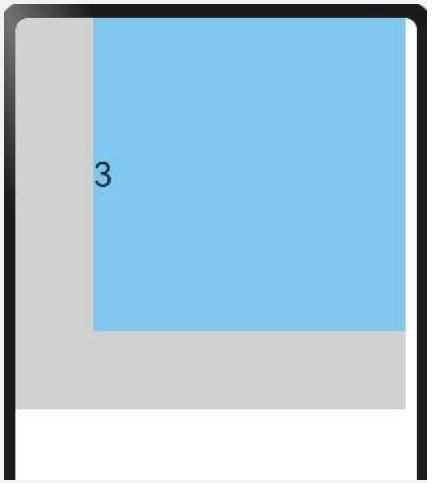


BottomEnd

Stack层叠布局应用

StackLayoutDemo1.ets

```
build() {
  Stack({ alignContent: Alignment.TopEnd }) {
    Text('1')
      .fontSize(30)
      .width(80)
      .height(80)
      .backgroundColor("#1F4E79")
    Text('2')
      .fontSize(30)
      .width(120)
      .height(120)
      .backgroundColor("#1D90CB")
    Text('3')
      .fontSize(30)
      .width(280)
      .height(280)
      .backgroundColor("#80C8ED")
  }.width(350).height(350).backgroundColor("#D2D2D2")
}
```

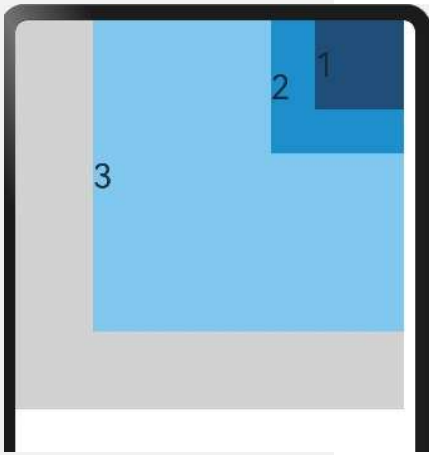


默认情况下在文档中越靠后的元素在页面中越靠前，所以Text3覆盖了Text1和Text2。可通过zIndex属性改变层叠顺序

zIndex属性控制Stack容器中的子组件的叠放顺序，默认值0，值越大，显示层级越高。可以为负值

分别为三个文本组件添加zIndex属性后如下所示：

```
build() {
  Stack({ alignContent: Alignment.TopEnd }) {
    Text('1')
      ......
      .zIndex(10)
    Text('2')
    ......
      .zIndex(4)
    Text('3')
      ......
      .zIndex(1)
  }.width(350).height(350).backgroundColor("#D2D2D2")
```



【课程小结】

- Tabs组件的参数**barPosition**设置导航栏的位置，默认值**Start**（顶部导航栏）。**End**（底部导航栏）。实现侧边导航栏需要设置Tabs的属性**vertical为true**。
- 层叠布局中通过zIndex属性控制Stack容器中的子组件的叠放顺序，默认值0，值越大，显示层级越高。可以为负值。
- **Grid容器设置网格布局参数**，属性有columnsTemplate（列数量及占比）、rowTemplate（行数量及占比）、columnsGap（列间隔）、rowsGap（行间隔）。**GridItem定义子组件特征**，属性有rowStart（起始行编号）、rowEnd（终止行编号）、columnStart（起始列编号）、columnEnd（终止列编号）