第 5 章 常用组件布局开发

本章内容主要对 HarmonyOS 原子化服务与服务卡片开发过程中可以使用的各项组件与布局进行汇总与练习。通过组件引用和组合的方式，可以快速实现各项样式与功能；通过组件的练习，可以快速地让开发者看到成效。我们创作了部分组件练习案例，读者可以直接引用练习，其他组件的使用思路和流程与此相同。

5.1 JS 通用组件

本节主要对在使用 JS 语言进行原子化服务与服务卡片开发时所用到的相关通用组件进行说明，包括通用属性、样式、事件等。

5.1.1 通用属性

1. 常规属性

常规属性指的是组件普遍支持的用来设置组件基本标识和外观显示特征的属性，该属性的具体相关内容见表 5-1。

表 5-1 常规属性

名称	类型	默认值	必填	描　　述
id	string	—	否	组件的唯一标识
style	string	—	否	组件的样式声明
class	string	—	否	组件的样式类，用于引用样式表
ref	string	—	否	用来指定指向子元素或子组件的引用信息，该引用将注册到父组的 $refs 属性对象上
disabled	boolean	false	否	当前组件是否被禁用，在禁用场景下，组件将无法响应用户交互
dir	string	auto	否	设置元素的布局模式，支持设置 rtl、ltr 和 auto 共 3 种属性值。rtl 使用从右往左的布局模式，ltr 使用从左往右的布局模式，auto 跟随系统语言环境

2. 渲染属性

渲染属性是组件普遍支持的用来设置组件是否渲染的属性，该属性的具体相关内容见表 5-2。

表 5-2 渲染属性

名称	类型	默认值	描　述
for	array	—	根据设置的数据列表，展开当前元素
if	boolean	—	根据设置的 boolean 值，添加或移除当前元素
show	boolean	—	根据设置的 boolean 值，显示或隐藏当前元素

5.1.2 通用样式

通用样式是指普遍支持的可以在 style 或 CSS 中设置组件外观样式的组件，通用样式不是必填项，具体名称、类型属性等见表 5-3。

表 5-3 通用样式

名　称	类　型	默认值	描　述
width	<length>| <percentage>	—	设置组件自身的宽度 缺省时使用元素自身内容所需要的宽度
height	<length>| <percentage>	—	设置组件自身的高度 缺省时使用元素自身内容所需要的高度
min-width	<length>| <percentage>	0	设置元素的最小宽度
min-height	<length>| <percentage>	0	设置元素的最小高度
max-width	<length>| <percentage>	—	设置元素的最大宽度，默认无限制
max-height	<length>| <percentage>	—	设置元素的最大高度，默认无限制
padding	<length>| <percentage>	0	使用简写属性设置所有的内边距属性。该属性可以有 1～4 个值：当指定一个值时，该值指定 4 条边的内边距；当指定两个值时，第 1 个值指定上下两边的内边距，第 2 个值指定左右两边的内边距；当指定 3 个值时，第 1 个值指定上边的内边距，第 2 个值指定左右两边的内边距，第 3 个值指定下边的内边距；当指定 4 个值时，分别指定上、右、下、左边的内边距(顺时针顺序)
padding-[left|top|right|bottom]	<length>| <percentage>	0	设置左、上、右、下内边距属性
padding-[start|end]	<length>| <percentage>	0	设置起始和末端内边距属性

续表

名　称	类　型	默认值	描　述
margin	<length>| <percentage>	0	使用简写属性设置所有的外边距属性，该属性可以有1～4个值。当只有一个值时，这个值会被指定给全部的4条边；当有两个值时，第1个值被匹配给上和下，第2个值被匹配给左和右；当有3个值时，第1个值被匹配给上，第2个值被匹配给左和右，第3个值被匹配给下；当有4个值时，会依次按上、右、下、左的顺序匹配（顺时针顺序）
margin-[left|top|right|bottom]	<length>| <percentage>	0	设置左、上、右、下外边距属性
margin-[start|end]	<length>| <percentage>	0	设置起始和末端外边距属性
border	—	0	使用简写属性设置所有的边框属性，包含边框的宽度、样式、颜色属性，按顺序设置为border-width、border-style、border-color，不设置时，各属性值为默认值
border-style	string	solid	使用简写属性设置所有边框的样式，可选值dotted表示显示为一系列圆点，圆点半径为border-width的一半；可选值dashed表示显示为一系列短的方形虚线；可选值solid表示显示为一条实线
border-[left|top|right|bottom]-style	string	solid	分别设置左、上、右、下4条边框的样式，可选值为dotted、dashed、solid
border-[left|top|right|bottom]	—	—	使用简写属性设置对应位置的边框属性，包含边框的宽度、样式、颜色属性，顺序设置为border-width、border-style、border-color，不设置的值为默认值
border-width	<length>	0	使用简写属性设置元素的所有边框宽度，或者单独为各边边框设置宽度
border-[left|top|right|bottom]-width	<length>	0	分别设置左、上、右、下4条边框的宽度
border-color	<color>	black	使用简写属性设置元素的所有边框颜色，或者单独为各边边框设置颜色
border-[left|top|right|bottom]-color	<color>	black	分别设置左、上、右、下4条边框的颜色
border-radius	<length>	—	border-radius属性用于设置元素的外边框圆角半径。设置border-radius时不能单独设置某一个方向的border-[left|top|right|bottom]-width、border-[left|top|right|bottom]-color、border-[left|top|right|bottom]-style，如果要设置color、width和style，则需要将4个方向一起设置（border-width、border-color、border-style）

续表

名　称	类　型	默认值	描　述
border-[top|bottom]-[left|right]-radius	<length>	—	分别设置左上、右上、右下和左下4个角的圆角半径
background	<linear-gradient>	—	仅支持设置渐变样式，与background-color、background-image不兼容
background-color	<color>	—	设置背景颜色
background-image	string	—	设置背景图片。与background-color、background不兼容，支持本地图片资源地址，示例：background-image：url("/common/background.png")
background-size	string <length> <length> <percentage> <percentage>	auto	设置背景图片的大小。string可选值：contain把图片扩展至最大尺寸，以使其高度和宽度完全适用内容区域；cover把背景图片扩展至足够大，以使背景图片完全覆盖背景区域；背景图片的某些部分也许无法显示在背景定位区域中；auto保持原图的比例不变。length值：设置背景图片的高度和宽度。第1个值设置宽度，第2个值设置高度。如果只设置一个值，则第2个值会被设置为auto。百分比参数以父元素的百分比设置背景图片的宽度和高度。第1个值设置宽度，第2个值设置高度。如果只设置一个值，则第2个值会被设置为auto
background-repeat	string	repeat	针对重复背景图片样式进行设置，背景图片默认在水平和垂直方向上重复。repeat在水平轴和竖直轴上同时重复绘制图片； repeat-x：只在水平轴上重复绘制图片；repeat-y：只在竖直轴上重复绘制图片；no-repeat：不会重复绘制图片
background-position	string string <length> <length> <percentage> <percentage>	0px 0px	关键词方式：如果仅规定了一个关键词，则第2个值为center。两个值分别用于定义水平方向位置和竖直方向位置 left是水平方向上最左侧；right是水平方向上最右侧；top是竖直方向上最顶部；bottom是竖直方向上最底部；center是水平方向或竖直方向上中间位置；length值：第1个值是水平位置，第2个值是垂直位置。左上角是0 0。单位是像素（0px 0px）。如果仅规定了一个值，则另外一个值将是50%。百分比参数：第1个值是水平位置，第2个值是垂直位置。左上角是0% 0%。右下角是100% 100%。如果仅规定了一个值，则另外一个值为50%。可以混合使用<percentage>和<length>

续表

名　称	类　型	默认值	描　述
box-shadow	string	0	语法为 box-shadow：h-shadow v-shadow blur spread color，通过这个样式可以设置当前组件的阴影样式，包括水平位置（必填）、垂直位置（必填）、模糊半径（可选，默认值为 0）、阴影延展距离（可选，默认值为 0）、阴影颜色（可选，默认值为黑色），示例：①box-shadow ：10px 20px 5px 10px ＃888888；②box-shadow ：100px 100px 30px red；③box-shadow ：-100px -100px 0px 40px
filter	string	—	语法为 filter：blur(px)，通过这个样式可以将当前组件布局范围的内容设置为模糊，参数用于指定模糊半径，如果没有设置值，则默认为 0（不模糊），不支持百分比，示例：filter：blur(10px)
backdrop-filter	string	—	语法为 backdrop-filter：blur(px)，通过这个样式可以将当前组件布局范围的背景设置为模糊，参数用于指定模糊半径，如果没有设置值，则默认为 0（不模糊），不支持百分比，示例：backdrop-filter：blur(10px)
opacity	number	1	元素的透明度，取值范围为 0～1，1 表示不透明，0 表示完全透明
display	string	flex	确定一个元素所产生的框的类型，可选值为 flex，表示弹性布局；none 不渲染此元素
visibility	string	visible	是否显示元素所产生的框。不可见的框会占用布局（将 display 属性设置为 none 来完全去除框），可选值为 visible，表示元素正常显示；hidden 表示隐藏元素，但是其他元素的布局不改变，相当于此元素变成透明（说明：visibility 和 display 样式都设置时，仅 display 生效）
flex	number|string	—	规定当前组件如何适应父组件中的可用空间。 flex 可以指定 1 个、2 个或 3 个值。 单值语法：一个无单位数用来设置组件的 flex-grow；一个有效的宽度值用来设置组件的 flex-basis。 双值语法：第 1 个值必须是无单位数，用来设置组件的 flex-grow，第 2 个值可以是一个无单位数，用来设置组件的 flex-shrink，还可以是一个有效的宽度值，用来设置组件的 flex-basis。 三值语法：第 1 个值必须是无单位数，用来设置组件的 flex-grow；第 2 个值必须是无单位数，用来设置组件的 flex-shrink；第 3 个值必须是一个有效的宽度值，用来设置组件的 flex-basis（说明：仅父容器为< div >、< list-item >、< tabs >、< refresh >、< stepper-item >时生效）

续表

名　称	类　型	默认值	描　述
flex-grow	number	0	设置组件的拉伸样式，指定父组件容器主轴方向上的剩余空间（容器本身大小减去所有 flex 子元素占用的大小）的分配权重。0 为不伸展（说明：仅父容器为< div >、< list-item >、< tabs >、< refresh >、< stepper-item >时生效）
flex-shrink	number	1	设置组件的收缩样式，元素仅在默认宽度之和大于容器的时候才会发生收缩，0 为不收缩（说明：仅父容器为< div >、< list-item >、< tabs >、< refresh >、< stepper-item >时生效）
flex-basis	< length >	—	设置组件在主轴方向上的初始大小（说明：仅父容器为< div >、< list-item >、< tabs >、< refresh >、< stepper-item >时生效）
align-self	string	—	设置自身在父元素交叉轴上的对齐方式，该样式会覆盖父元素的 align-items 样式，仅在父容器为 div、list 时生效。可选值 stretch 弹性元素表示在交叉轴方向被拉伸到与容器相同的高度或宽度；flex-start 元素表示向交叉轴起点对齐；flex-end 元素表示向交叉轴终点对齐；center 元素表示在交叉轴居中；baseline 元素表示在交叉轴基线对齐
position	string	relative	设置元素的定位类型，不支持动态变更。 fixed 相对于整个界面进行定位；absolute 相对于父元素进行定位；relative 相对于其正常位置进行定位（说明：absolute 属性仅在父容器为< div >、< stack >时生效）
[left | top | right | bottom]	< length >| < percentage >	—	left|top|right|bottom 需要配合 position 样式使用，来确定元素的偏移位置。 left 属性规定元素的左边缘。该属性定义了定位元素左外边距边界与其包含块左边界之间的偏移。 top 属性规定元素的顶部边缘。该属性定义了一个定位元素的上外边距边界与其包含块上边界之间的偏移。 right 属性规定元素的右边缘。该属性定义了定位元素右外边距边界与其包含块右边界之间的偏移。 bottom 属性规定元素的底部边缘。该属性定义了一个定位元素的下外边距边界与其包含块下边界之间的偏移
[start|end]	< length >| < percentage >	—	start|end 需要配合 position 样式使用，来确定元素的偏移位置。 start 属性规定元素的起始边缘。该属性定义了定位元素起始外边距边界与其包含块起始边界之间的偏移。 end 属性规定元素的结尾边缘。该属性定义了一个定位元素的结尾边距边界与其包含块结尾边界之间的偏移

续表

名　称	类　型	默认值	描　述
z-index	number	—	表示对于同一父节点其子节点的渲染顺序。数值越大，渲染数据越靠后(说明：z-index 不支持 auto，并且 opacity 等其他样式不会影响 z-index 的渲染顺序)
image-fill	<color>	—	为 svg 图片填充颜色，支持组件范围(与设置图片资源的属性)：button(icon 属性)、piece(icon 属性)、search(icon 属性)、input(headericon 属性)、textarea(headericon 属性)、image(src 属性)、toolbar-item(icon 属性)。 svg 图片文件内的 fill 属性颜色值在渲染时将被替换为 image-fill 所配的颜色值，并且仅对 svg 图片内显示声明的 fill 属性生效
clip-path	[<geometry-box>|<basic-shape>]|none	—	设置组件的裁剪区域。区域内的部分显示，区域外的部分不显示。<geometry-box>：表示裁剪区域的作用范围，默认为 border-box。可选值为 margin-box：margin，计算入长和宽尺寸内；border-box：border 计算入长和宽尺寸内；padding-box：padding 计算入长和宽尺寸内；content-box：margin/border/padding 不计算入长和宽尺寸内。 <basic-shape>表示裁剪的形状，包含以下类型：inset，格式为 inset(<percentage>{1,4} [round <'border-radius'>]?)；circle，格式为 circle([<percentage>]? [at <percentage> <percentage>]?)；ellipse，格式为 ellipse([<percentage>{2}]? [at <percentage> <percentage>]?)；polygon，格式为 polygon([<percentage> <percentage>]#)；path，格式为 path(<string>)

5.1.3 通用事件

相对于私有事件，通用事件是大部分组件可以绑定的事件，该事件的相关内容具体见表 5-4。

表 5-4 通用事件

名　称	参　数	描　述
click	—	单击动作触发该事件
change	—	监听事件

5.1.4 渐变样式

渐变样式组件普遍支持在 style 或 CSS 中设置，可以平稳过渡两个或多个指定的颜色。

开发框架支持线性渐变（linear-gradient）和重复线性渐变（repeating-linear-gradient）两种渐变效果。

使用渐变样式，需要定义过渡方向和过渡颜色。

1）过渡方向

通过 direction 或者 angle 指定：

（1）direction 表示进行方向渐变。

（2）angle 表示进行角度渐变。

```
background: linear-gradient(direction/angle, color, color, ...);
background: repeating-linear-gradient(direction/angle, color, color, ...);
```

2）过渡颜色

（1）支持以下 4 种方式：

#ff0000、#ffff0000、rgb(255, 0, 0)、rgba(255, 0, 0, 1)，需要至少指定两种颜色。

（2）过渡属性，见表 5-5。

表 5-5 过渡颜色

名　称	类　型	默认值	必填	描　述
direction	to < side-or-corner > < side-or-corner > = [left|right] || [top|bottom]	to bottom（由上到下渐变）	否	指定过渡方向，如：to left（从右向左渐变）；或者 to bottom right（从左上角到右下角）
angle	< deg >	180deg	否	指定过渡方向，以元素几何中心为坐标原点，水平方向为 X 轴，angle 指定了渐变线与 Y 轴的夹角（顺时针方向）
color	< color > [< length >|< percentage >]	—	是	定义使用渐变样式区域内颜色的渐变效果

5.1.5 媒体查询

媒体查询（Media Query）在移动设备上应用十分广泛，开发者经常需要根据设备的大致类型或者特定的特征和设备参数（例如屏幕分辨率）来修改应用的样式。为此媒体查询提供了以下功能：

（1）针对设备和应用的属性信息，可以设计出相匹配的布局样式。

（2）当屏幕发生动态改变时（例如分屏、横竖屏切换），应用页面布局同步更新。

1. CSS 语法规则

使用@media 可引入查询语句，具体规则如下：

```
@media (media-feature) {
CSS-Code;
}
```

示例如下：

```
@media (dark-mode: true) {… }                    //当设备为深色模式时生效
@media screen and (round-screen: true) { … }     //当设备屏幕是圆形时条件成立
@media (max-height: 800) { … }                   //范围查询,CSS level 3 写法
@media (height <= 800) { … }              //范围查询,CSS level 4 写法,与 CSS level 3 写法等价
@media screen and (device-type: tv) or (resolution < 2) { …} //同时包含媒体类型和多个媒体
                                                              //特征的多条件复杂语句查询
```

2. 页面中引用资源

通过@import 方式引入媒体查询，具体使用方法如下：

```
@import url (media-feature);

//例如
@import '../common/style.css'(dark-mode:true);
```

3. 媒体类型

媒体类型见表 5-6，需要按屏幕相关参数进行媒体查询。

表 5-6　媒体类型

类　　型	说　　明
screen	按屏幕相关参数进行媒体查询

4. 媒体逻辑操作

媒体逻辑操作符，and、or、not、only 用于构成复杂媒体查询，也可以通过逗号(,)将其组合起来，详细解释说明见表 5-7。

表 5-7　媒体逻辑操作

类型	说　　明
and	将多个媒体特征(Media Feature)以“与”的方式连接成一个媒体查询，只有当所有媒体特征都为 true 时，查询条件成立。另外，它还可以将媒体类型和媒体功能结合起来，例如：screen and (device-type：wearable) and (max-height：600) 表示当设备类型是智能穿戴同时应用的最大高度小于或等于 600 像素单位时成立

续表

类型	说　明
not	取反媒体查询结果，媒体查询结果不成立时返回值为 true，否则返回值为 false。在媒体查询列表中应用 not，not 仅取反应用它的媒体查询，例如：not screen and (min-height: 50) and (max-height: 600) 表示当应用高度小于 50 像素单位或者大于 600 像素单位时成立（说明：使用 not 运算符时必须指定媒体类型）
only	当整个表达式都匹配时，才会应用选择的样式，可以应用在防止某些较早版本的浏览器上产生歧义的场景。一些较早版本的浏览器对于同时包含了媒体类型和媒体特征的语句会产生歧义，例如：screen and (min-height: 50)，较早版本浏览器会将这句话理解成 screen，从而导致仅仅匹配媒体类型(screen)，只应用了指定样式，使用 only 可以很好地规避这种情况（说明：使用 only 时必须指定媒体类型）
,(逗号)	将多个媒体特征以“或”的方式连接成一个媒体查询，如果存在结果为 true 的媒体特征，则查询条件成立。其效果等同于 or 运算符，例如：screen and (min-height: 1000), (round-screen: true) 表示当应用高度大于或等于 1000 像素单位或者设备屏幕是圆形时，条件成立
or	将多个媒体特征以“或”的方式连接成一个媒体查询，如果存在结果为 true 的媒体特征，则查询条件成立，例如：screen and (max-height: 1000) or(round-screen: true)表示当应用高度小于或等于 1000 像素单位或者设备屏幕是圆形时，条件成立

5. 媒体特征

媒体特征包括高度、宽度、分辨率等内容，具体见表 5-8。

表 5-8　媒体特征

类　型	说　明
height	应用页面显示区域的高度
min-height	应用页面显示区域的最小高度
max-height	应用页面显示区域的最大高度
width	应用页面显示区域的宽度
min-width	应用页面显示区域的最小宽度
max-width	应用页面显示区域的最大宽度
resolution	设备的分辨率，支持 dpi、dppx 和 dpcm 单位。其中，dpi 表示每英寸中物理像素的个数，1dpi≈0.39dpcm；dpcm 表示每厘米上的物理像素个数，1dpcm≈2.54dpi；dppx 表示每个 px 中的物理像素数(此单位按 96px=1 英寸为基准，与页面中的 px 单位计算方式不同)，1dppx=96dpi
min-resolution	设备的最小分辨率
max-resolution	设备的最大分辨率
orientation	屏幕的方向 可选值为 orientation: portrait(设备竖屏)和 orientation: landscape(设备横屏)
aspect-ratio	应用页面显示区域的宽度与高度的比值，例如：aspect-ratio:1/2
min-aspect-ratio	应用页面显示区域的宽度与高度的最小比值

续表

类　型	说　明
max-aspect-ratio	应用页面显示区域的宽度与高度的最大比
device-height	设备的高度
min-device-height	设备的最小高度
max-device-height	设备的最大高度
device-width	设备的宽度
min-device-width	设备的最小宽度
max-device-width	设备的最大宽度
device-type	设备类型，支持 phone(手机)、tablet(平板)、tv(智慧屏)、wearable(智能穿戴)
round-screen	屏幕类型，圆形屏幕为 true，非圆形屏幕为 false
dark-mode	系统在深色模式下为 true，否则为 false

5.2 JS 容器组件

本节主要对 JS 容器组件进行整体介绍与进行部分组件实战练习，包括 JS 容器组件的类别名称、功能说明与实例等。

5.2.1 容器组件

容器组件名称类别及相关功能使用说明，详细见表 5-9。

表 5-9 容器组件

名　称	说　明
badge	如果应用中有需用户关注的新事件提醒，则可以采用新事件标记来标识
div	基础容器，用作页面结构的根节点或将内容进行分组
list	列表包含一系列相同宽度的列表项。适合连续、多行呈现同类数据，例如图片和文本
list-item	< list >的子组件，用来展示列表的具体 item
stack	堆叠容器，子组件按照顺序依次入栈，后一个子组件覆盖前一个子组件
swiper	滑动容器，提供切换子组件显示的能力

5.2.2 容器组件示例

1. badge

如果应用中有需用户关注的新事件提醒，则可以采用新事件标记来标识，需要使用 badge。

1）演示案例主要代码

index. hml 文件的代码如下：

```
/* 第 5 章/badge 案例 index.hml 代码部分 */
<div class = "container">
      //标记提醒
     <badge class = "badge" config = "{{badgeconfig}}" visible = "true" count = "100" maxcount = "99">
         <text class = "text1"> example </text>
     </badge>
     //标记提醒
     <badge class = "badge" visible = "true" count = "0">
         <text class = "text2"> example </text>
     </badge>
</div>
```

index.css 文件的代码如下：

```
//第 5 章/badge 案例 index.css 代码部分
.container {
     flex-direction: column;
     width: 100%;
     align-items: center;
}
//标记样式
.badge {
     width: 50%;
     margin-top: 100px;
}
.text1 {
     background-color: #f9a01e;
     font-size: 30px;
     padding: 10px;
}
.text2 {
     background-color: #46b1e3;
     font-size: 30px;
     padding: 10px;
}
```

index.js 文件的代码如下：

```
//第 5 章/badge 案例 index.js 代码部分
export default {
     data:{
      //提醒样式
        badgeconfig:{
            badgeColor:"#0a59f7",
            textColor:"#ffffff",
```

```
        }
      }
}
```

2）演示案例效果展示

badge 演示案例效果，如图 5-1 所示。

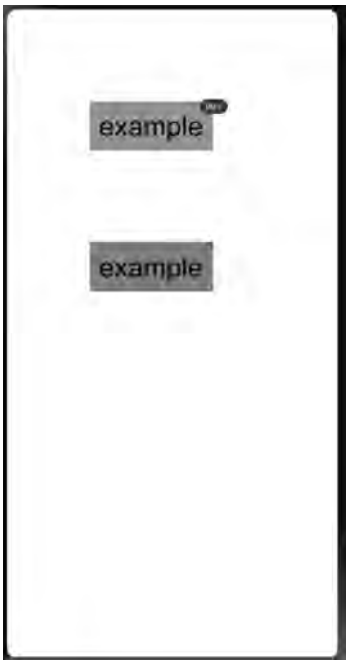


图 5-1　展示效果(1)

2. div

div 是基础容器，用作页面结构的根节点或将内容进行分组。

1）演示案例主要代码

index. hml 文件的代码如下：

```
/ * 第 5 章/div 案例 index. hml 代码部分 * /
< div class = "container">
     < div class = "flex - box">
      //节点
         < div class = "flex - item color - primary"></div >
         < div class = "flex - item color - warning"></div >
         < div class = "flex - item color - success"></div >
     </div >
</div >
```

index.css 文件的代码如下：

```
//第 5 章/div 案例 index.css 代码部分
.container {
    flex-direction: column;
    justify-content: center;
    align-items: center;
    width: 454px;
    height: 454px;
}
.flex-box {
    justify-content: space-around;
    align-items: center;
    width: 400px;
    height: 140px;
    background-color: #ffffff;
}
.flex-item {
    width: 120px;
    height: 120px;
    border-radius: 16px;
}
.color-primary {
    background-color: #007dff;
}
.color-warning {
    background-color: #ff7500;
}
.color-success {
    background-color: #41ba41;
}
```

2）演示案例效果展示

div 基础容器演示案例效果如图 5-2 所示。

3. list 和 list-item

list 和 list-item 是列表类组件，列表包含一系列相同宽度的列表项。适合连续、多行呈现同类数据，例如图片和文本。

1）演示案例主要代码

index.hml 文件的代码如下：

```
/* 第 5 章/list 和 list-item 案例 index.hml 代码部分 */
<div class="container">
      //列表容器
    <list class="todo-wrapper">
```

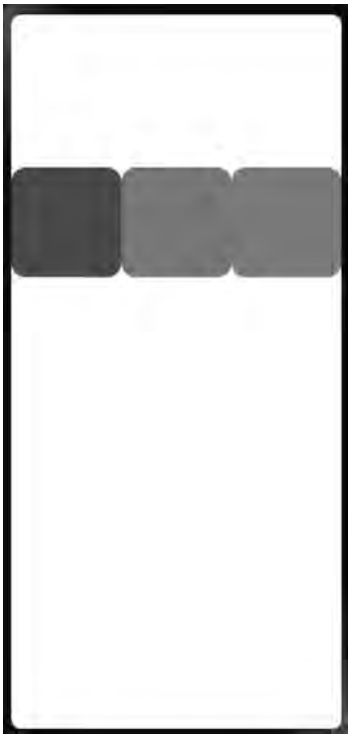


图 5-2　展示效果(2)

```
        <list-item for="{{todolist}}" class="todo-item">
            <text class="todo-title">{{ $item.title}}</text>
            <text class="todo-title">{{ $item.date}}</text>
        </list-item>
    </list>
</div>
```

index.css 文件的代码如下：

```
//第 5 章/list 和 list-item 案例 index.css 代码部分
.container {
    display: flex;
    justify-content: center;
    align-items: center;
    left: 0px;
    top: 0px;
    width: 454px;
    height: 454px;
}
.todo-wrapper {
```

```
    width: 454px;
    height: 300px;
}
.todo-item {
    width: 454px;
    height: 80px;
    flex-direction: column;
}
.todo-title {
    width: 454px;
    height: 40px;
    text-align: center;
}
```

index.js 文件的代码如下：

```
//第 5 章/list 和 list-item 案例 index.js 代码部分
export default {
    data: {
        todolist: [{
                    title: '刷题',
                    date: '2021-12-31 10:00:00',
                }, {
                    title: '看电影',
                    date: '2021-12-31 20:00:00',
                }],
    },
}
```

2）演示案例效果展示

list 和 list-item 列表类组件演示案例效果，如图 5-3 所示。

图 5-3　展示效果(3)

4. stack

stack 是堆叠容器，子组件按照顺序依次入栈，后一个子组件覆盖前一个子组件。

1）演示案例主要代码

index.hml 文件的代码如下：

```
/* 第 5 章/stack 案例 index.hml 代码部分 */
<stack class = "stack - parent">
      //节点
    <div class = "back - child bd - radius"></div>
    <div class = "positioned - child bd - radius"></div>
    <div class = "front - child bd - radius"></div>
</stack>
```

index.css 文件的代码如下：

```
//第 5 章/stack 案例 index.css 代码部分
.stack - parent {
    width: 400px;
    height: 400px;
    background - color: #ffffff;
    border - width: 1px;
    border - style: solid;
}
.back - child {
    width: 300px;
    height: 300px;
    background - color: #3f56ea;
}
.front - child {
    width: 100px;
    height: 100px;
    background - color: #00bfc9;
}
.positioned - child {
    width: 100px;
    height: 100px;
    left: 50px;
    top: 50px;
    background - color: #47cc47;
}
.bd - radius {
    border - radius: 16px;
}
```

2）演示案例效果展示

stack 堆叠容器组件案例演示效果如图 5-4 所示。

图 5-4　展示效果(4)

5. swiper

swiper 是滑动容器，提供切换子组件显示的能力。

1）演示案例主要代码

index.hml 文件的代码如下：

```
/* 第 5 章/swiper 案例 index.hml 代码部分 */
<div class = "container">
     //滑动容器
    <swiper class = "swiper" id = "swiper" index = "0" indicator = "true" loop = "true" digital =
"false" cachedsize = " - 1"
            scrolleffect = "spring">
        <div class = "swiperContent1" >
            <text class = "text"> 1 </text >
        </div >
        <div class = "swiperContent2">
            <text class = "text"> 2 </text >
        </div >
        <div class = "swiperContent3">
            <text class = "text"> 3 </text >
        </div >
    </swiper >
     //文本输入框和单击事件
    <input class = "button" type = "button" value = "swipeTo" onclick = "swipeTo"></input >
    <input class = "button" type = "button" value = "showNext" onclick = "showNext"></input >
    <input class = "button" type = "button" value = "showPrevious" onclick = "showPrevious">
</input >
</div >
```

index.css 文件的代码如下：

```
//第 5 章/swiper 案例 index.css 代码部分
.container {
    flex-direction: column;
    width: 100%;
    height: 100%;
    align-items: center;
}
.swiper {
    flex-direction: column;
    align-content: center;
    align-items: center;
    width: 70%;
    height: 130px;
    border: 1px solid #000000;
    indicator-color: #cf2411;
    indicator-size: 14px;
    indicator-bottom: 20px;
    indicator-right: 30px;
    margin-top: 100px;
    next-margin:20px;
    previous-margin:20px;
}
.swiperContent1{
    height: 100%;
    justify-content: center;
    background-color: #007dff;
}
.swiperContent2{
    height: 100%;
    justify-content: center;
    background-color: #ff7500;
}
.swiperContent3{
    height: 100%;
    justify-content: center;
    background-color: #41ba41;
}
.button {
    width: 70%;
    margin: 10px;
}
```

```
.text {
    font-size: 40px;
}
```

index.js 文件的代码如下：

```
//第 5 章/swiper 案例 index.js 代码部分
export default {
    swipeTo() {
     //获取下标
        this.$element('swiper').swipeTo({index: 2});
    },
    showNext() {
        this.$element('swiper').showNext();
    },
    showPrevious() {
        this.$element('swiper').showPrevious();
    }
}
```

2）演示案例效果展示

swiper 是滑动容器组件，演示案例效果如图 5-5 中(a)→(b)→(c)所示。

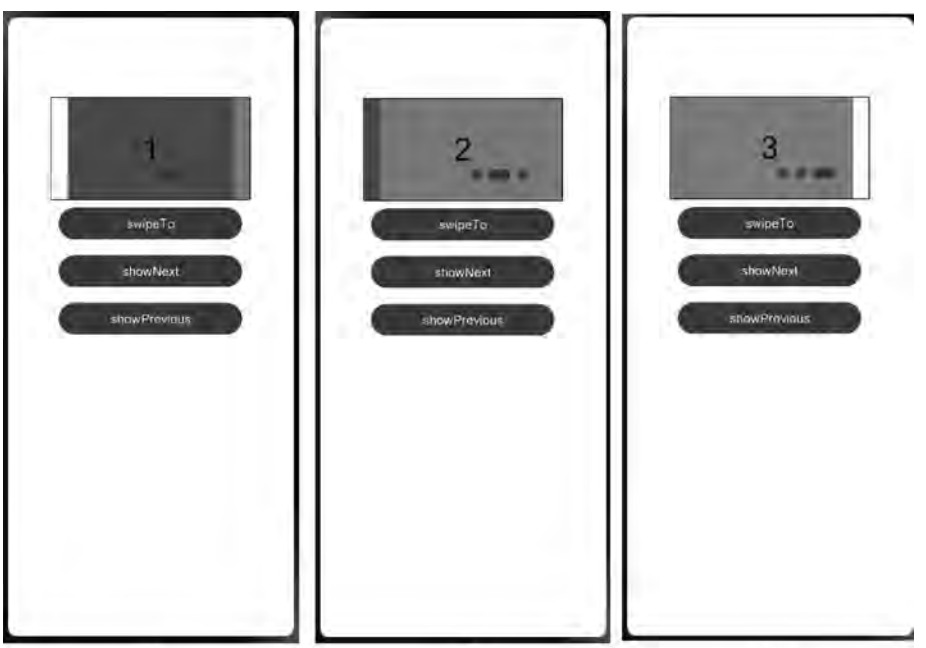


图 5-5　展示效果(5)

5.3 JS 基础组件

本节主要阐述 JS 基础组件的名称类别、功能样式，以及部分 JS 基础组件案例实战练习。

5.3.1 基础组件

JS 基础组件主要的名称类别与相关功能的使用说明见表 5-10。

表 5-10 基础组件

名 称	说 明
button	提供按钮组件，包括胶囊按钮、圆形按钮、文本按钮
calendar	提供日历组件，用于呈现日历界面
chart	图表组件，用于呈现线性图、柱状图、量规图界面
clock	时钟组件，用于提供时钟表盘界面
divider	提供分隔器组件，分隔不同内容块/内容元素。可用于列表或界面布局
image	图片组件，用来渲染并展示图片
progress	进度条，用于显示内容加载或操作处理进度
span	作为< text >子组件提供文本修饰能力
text	文本，用于呈现一段信息

5.3.2 基础组件示例

1. button

button 提供按钮组件，包括胶囊按钮、圆形按钮、文本按钮。

1）演示案例主要代码

index. hml 文件的代码如下：

```
/ * 第 5 章/基础组件 button 案例 index. hml 代码部分 * /
< div class = "div - button">
    < button class = "button text" type = "text"> Text button </button >
      //下载进度
    < button class = "button download" type = "download" id = "download - btn"
            onclick = "progress">{{downloadText}}</button >
      //加载
    < button class = "last" type = "capsule" waiting = "true"> Loading </button >
      //单击跳转
    < button class = "next" value = "Next" onclick = "goIndex"></button >
</div >
```

index. css 文件的代码如下：

```
//第 5 章/基础组件 button 案例 index.css 代码部分
.div-button {
    flex-direction: column;
    align-items: center;
}

.button {
    margin-top: 15px;
}
.last{
    background-color: #F2F2F2;
    text-color: #969696;
    margin-top: 15px;
    width: 280px;
    height:72px;
}
.next{
    width: 180px;
    height: 50px;
    font-size: 26px;
    border-radius: 25px;
}
.button:waiting {
    width: 280px;
}
.text {
    text-color: red;
    font-size: 40px;
    font-weight: 900;
    font-family: sans-serif;
    font-style: normal;
}
.download {
    width: 280px;
    text-color: white;
    background-color: #007dff;
}
```

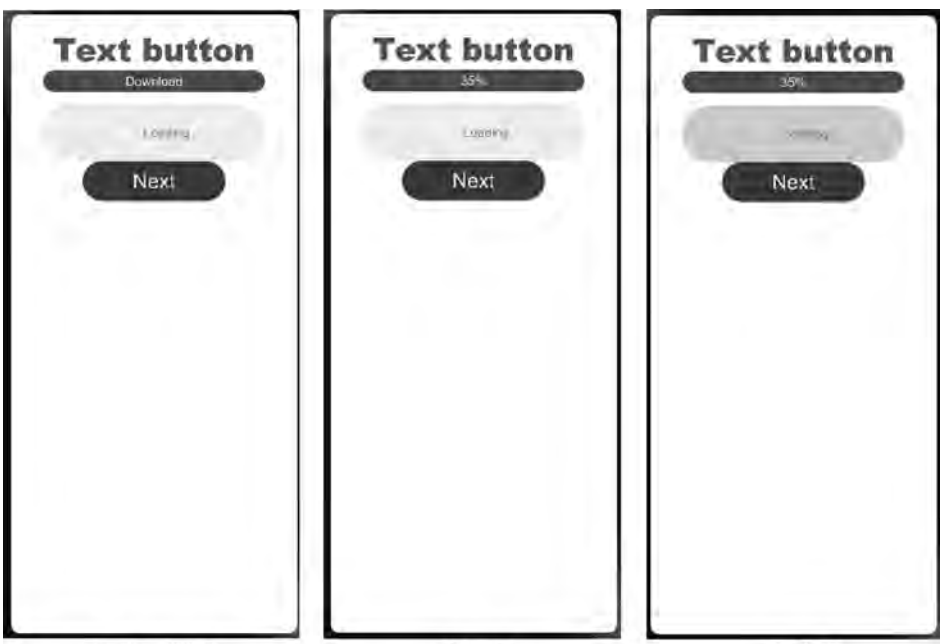
index.js 文件的代码如下：

```
//第 5 章/基础组件 button 案例 index.js 代码部分
import router from '@system.router';        //导入模块
export default {
    data: {
        count: 5,
```

```
        downloadText: "Download"
    },
      //跳转
    goIndex(){
        router.push({
            uri:'pages/index/index'
        })
    },
      //下载
    progress(e) {
        this.count += 10;
        this.downloadText = this.count + " % ";
        this.$element('download-btn').setProgress({ progress: this.count});
        if (this.count >= 100) {
            this.downloadText = "Done";
        }
    }
}
```

2）演示案例效果展示

button 按钮组件，演示案例效果如图 5-6(a)→(b)→(c)所示。



(a) 展示按钮　(b) 下载　(c) 加载

图 5-6　展示效果(6)

2. chart

chart 是图表组件，用于呈现线性图、柱状图、量规图界面。

1）演示案例主要代码

index.hml 文件的代码如下：

```
/* 第 5 章/基础组件 chart 案例 index.hml 代码部分 */
<div class = "container">
    <stack class = "chart - region">
        <div class = "chart - background"></div>
      //图表 - 线状
        <chart class = "chart - data" type = "line" ref = "linechart" options = "{{lineOps}}"
datasets = "{{lineData}}"></chart>
    </stack>
      //按钮
    <button value = "Add data" onclick = "addData"></button>
</div>
```

index.css 文件的代码如下：

```
//第 5 章/基础组件 chart 案例 index.css 代码部分
.container {
    flex - direction: column;
    justify - content: center;
    align - items: center;
}
.chart - region {
    height: 400px;
    width: 700px;
}
.chart - background {
    width: 700px;
    height: 400px;
    background - color: cadetblue;
}
.chart - data {
    width: 700px;
    height: 600px;
}
button {
    width: 100 % ;
    height: 50px;
    background - color: #F4F2F1;
    text - color: #0C81F3;
}
```

index.js 文件的代码如下：

```
//第 5 章/基础组件 chart 案例 index.js 代码部分
export default {
    data: {
        lineData: [
            {
                strokeColor: '#0081ff',
                fillColor: '#cce5ff',
                data: [763, 550, 551, 554, 731, 654, 525, 696, 595, 628, 791, 505, 613, 575,
475, 553, 491, 680, 657, 716],
                gradient: true,
            }
        ],
        lineOps: {
            xAxis: {
                min: 0,
                max: 20,
                display: false,
            },
            yAxis: {
                min: 0,
                max: 1000,
                display: false,
            },
            series: {
                lineStyle: {
                    width: "5px",
                    smooth: true,
                },
                headPoint: {
                    shape: "circle",
                    size: 20,
                    strokeWidth: 5,
                    fillColor: '#ffffff',
                    strokeColor: '#007aff',
                    display: true,
                },
                loop: {
                    margin: 2,
                    gradient: true,
                }
            }
        },
    },
     //单击增加
```

```
    addData() {
        this.$refs.linechart.append({
            serial: 0,
            data: [Math.floor(Math.random() * 400) + 400]
        })
    }
}
```

2）演示案例效果展示

chart 是图表组件，演示案例效果如图 5-7 所示。

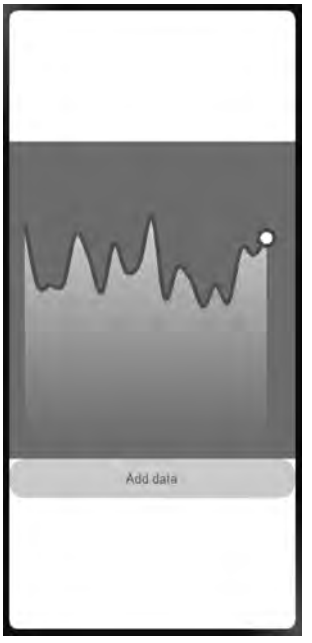


图 5-7　展示效果(7)

3. progress

progress 为进度条组件，用于显示内容加载或操作处理进度。

1）演示案例主要代码

index.hml 文件的代码如下：

```
/* 第 5 章/基础组件 progress 案例 index.hml 代码部分 */
<div class="container">
<!--      顺时针 -->
    <progress class="min-progress" type="scale-ring" percent= "10" secondarypercent=
"50"></progress>
<!--      水平 -->
```

```
    <progress class = "min - progress" type = "horizontal" percent = "10" secondarypercent =
"50"></progress >
<!-- 弧形进度条 -->
    <progress class = "min - progress" type = "arc" percent = "10"></progress >
<!-- 线性渐变 -->
    <progress class = "min - progress" type = "ring" percent = "10" secondarypercent = "50">
</progress >
</div >
```

index.css 文件的代码如下：

```
//第 5 章/基础组件 progress 案例 index.css 代码部分
.container {
    flex - direction: column;
    height: 100 % ;
    width: 100 % ;
    align - items: center;
}
.min - progress {
    width: 200px;
    height: 200px;
}
```

2）演示案例效果展示

progress 进度条组件，演示案例效果如图 5-8 所示。

图 5-8　展示效果(8)

5.4 JS 自定义组件与附录

本节主要阐述 JS 自定义组件的使用方式，以及与相关长度、颜色选择时的一些参考附录及表格等内容。

5.4.1 基本用法

自定义组件是用户根据业务需求，将已有的组件组合，封装成的新组件，可以在工程中多次调用，提高代码的可读性。自定义组件通过 element 引入宿主页面，代码如下：

```
<element name = 'comp' src = '../../common/component/comp.hml'></element>
<div>
      <comp prop1 = 'xxxx' @child1 = "bindParentVmMethod"></comp>
</div>
```

（1）name 属性指自定义组件名称（非必填），组件名称对大小写不敏感，默认使用小写。src 属性指自定义组件 HML 文件的路径（必填），若没有设置 name 属性，则默认使用 HML 文件名作为组件名。

（2）事件绑定：自定义组件中绑定子组件事件使用（on|@）child1 语法，子组件中通过 {action："proxy"，method："eventName"}触发事件并进行传值，父组件执行 bindParentVmMethod 方法并接收子组件传递的参数。

自定义组件配置文件标签的相关内容见表 5-11。

表 5-11 自定义组件配置文件标签

属性	类型	描　述
data	Object	页面的数据模型，类型是对象。属性名不能以 $或_开头，不要使用保留字 for、if、show、tid
props	Array/Object	props 用于组件之间的通信，可以通过< tag xxxx = 'value'>方式传递给组件；props 名称必须用小写，不能以 $或_开头，不要使用保留字 for、if、show、tid。目前 props 的数据类型不支持 Function

5.4.2 自定义事件

自定义组件内支持自定义事件，该事件的标识需要将 action 类型指定为 proxy，事件名则通过 method 指定。使用该自定义组件的卡片页面可以通过该事件名注册相应的事件回调，当自定义组件内该自定义事件触发时，会触发卡片页面上注册的回调事件。

1. 子组件 comp 示例

comp.hml 文件的示例代码如下：

```
<!-- 第 5 章/子组件 comp 示例 comp.hml 代码部分 -->
<div class = "container">
    <div class = "row - 3" if = "true">
        <button onclick = "buttonClicked" value = "click"></button>
    </div>
</div>
```

comp.css 文件的示例代码如下：

```
//第 5 章/子组件 comp 示例 comp.css 代码部分
.container {
    flex - direction:column;
    background - color: green;
    width: 100 % ;
    height: 100 % ;
}

.row - 3 {
    width: 100 % ;
    height: 50px;
    background - color: orange;
    font - size:15px;
}
```

comp.json 文件的示例代码如下：

```
//第 5 章/子组件 comp 示例 comp.json 代码部分
{
     "data": {
     },
    "actions": {
        "buttonClicked": {
           "action": "proxy",
            "method:" "event_1"
         }
      }
}
```

2. 卡片页面示例

index.hml 文件的示例代码如下：

```
<!-- 第 5 章/卡片页面示例 index.hml 代码部分 -->
<element name = 'comp' src = '../../common/customComponent/customComponent.hml'></element>

<div class = "container">
```

```
    <comp @event_1 = "click"></comp>
    <button value = "parentClick" @click = "buttonClick"></button>
</div>
```

index.css 文件的示例代码如下：

```
.container {
    background-color: red;
    height: 500px;
    width: 500px;
}
```

index.json 文件的示例代码如下：

```
//第 5 章/卡片页面示例 index.json 代码部分
{
     "data": {
     },
    "actions": {
       "click": {
           "action": "message",
           "params": {
           "message": "click event"
           }
       },
       "buttonClick": {
       "action": "message",
       "params": {
           "message": "click event 2"
         }
       }
    }
}
```

5.4.3 props

自定义组件可以通过 props 声明属性，父组件通过设置属性向子组件传递参数。具体内容如下。

1. 添加默认值

子组件可以通过固定值 default 设置默认值，当父组件没有设置该属性时，将使用其默认值。此情况下 props 属性必须为对象形式，不能用数组形式，示例如下。

demo.hml 文件的示例代码如下：

```
<!-- 第 5 章/props 自定义组件 demo.hml 代码部分 -->
<div class = "container">
    <div class = "row - 1">
        <div class = "box - 2">
            <text >{{text}}</text >
        </div >
        <div class = "box - 3">
            <text >{{textdata[0]}}</text >
        </div >
    </div >
    <div class = "row - 2" if = "true">
        <button value = "{{progress}}"></button >
    </div >
    <div class = "row - 3" if = "true">
        <button onclick = "buttonClicked" value = "click"></button >
    </div >
</div >
```

demo.json 文件的示例代码如下：

```
//第 5 章/props 自定义组件 demo.json 代码部分
{
    "data": {
       "progress": {
           "default": "80"
       }
    },
     "props": {
      "textdata": {
          "default": ["a","b"]
     },
     "progress": {
       "default": 60
     },
     "text": {
       "default": "ha"
     }
    },
    "actions": {
       "buttonClicked": {
           "action": "proxy",
           "method": "event_1"
       }
    }
}
```

demo.hml 文件的示例代码如下：

```
<!-- 第 5 章/props 自定义组件 demo.hml 代码部分 -->
<element name = 'comp' src = '../../common/customComponent/customComponent.hml'>
</element>

<div class = "container">
    <comp progress = "{{clicknow}}" textdata = "{{texts}}" if = "false" @event_1 = "click">
</comp>
</div>
```

2. 数据单向性

父子组件之间数据的传递是单向的，只能从父组件传递给子组件，子组件不能直接修改父组件传递来的值，可以将 props 传入的值用 data 接收后作为默认值，再对 data 的值进行修改。

5.4.4 附录

附录主要对常用的长度类型和颜色类型的相关内容进行展示，便于查阅与对照。

(1) 长度类型，见表 5-12。

表 5-12 长度类型

名称	类型定义	描　　述
length	string|number	用于描述尺寸单位，输入为 number 类型时，使用 px 单位；输入为 string 类型时，需要显式指定像素单位，当前支持的像素单位包括 px 和 fp。px 是逻辑尺寸单位；fp 是字体大小单位，会随系统字体大小的设置而发生变化，仅支持文本类组件设置相应的字体大小
percentage	string	百分比尺寸单位，如 50%

(2) 颜色类型，见表 5-13。

表 5-13 颜色类型

名称	类型定义	描　　述
color	string|颜色枚举字符串	用于描述颜色信息。字符串格式如下：rgb(255, 255, 255)，rgba(255, 255, 255, 1.0)。HEX 格式：#rrggbb，#aarrggbb。枚举格式：black，white。(说明：JS 脚本中不支持颜色枚举格式)

5.5 Java 组件开发

本节主要阐述常用的 Java 布局、组件相关的整体知识与进行部分布局、组件的实战案例练习。

5.5.1 常用布局

Java 常用布局及相关功能使用说明，见表 5-14。

表 5-14 常用布局

名 称	说 明
DirectionalLayout	DirectionalLayout 是 Java UI 中的一种重要组件布局，用于将一组组件(Component)按照水平或者垂直方向排布，能够方便地对齐布局内的组件。该布局和其他布局的组合，可以实现更加丰富的布局方式
DependentLayout	DependentLayout 是 Java UI 框架里的一种常见布局。与 DirectionalLayout 相比，拥有更多的排布方式，每个组件可以指定相对于其他同级元素的位置，或者指定相对于父组件的位置
StackLayout	StackLayout 直接在屏幕上开辟出一块空白的区域，添加到这个布局中的视图都以层叠的方式显示，而它会把这些视图默认放到这块区域的左上角，第 1 个添加到布局中的视图显示在最底层，最后一个被放在最顶层。上一层的视图会覆盖下一层的视图
TableLayout	TableLayout 使用表格的方式划分子组件
PositionLayout	在 PositionLayout 中，子组件通过指定准确的 x/y 坐标值在屏幕上显示。(0，0)为左上角；当向下或向右移动时，坐标值变大；允许组件之间互相重叠
AdaptiveBoxLayout	①AdaptiveBoxLayout 是自适应盒子布局，该布局提供了在不同屏幕尺寸设备上的自适应布局能力，主要用于相同级别的多个组件需要在不同屏幕尺寸设备上自动调整列数的场景；②该布局中的每个子组件都用一个单独的“盒子”装起来，子组件设置的布局参数都以盒子作为父布局生效，不以整个自适应布局为生效范围；③该布局中每个盒子的宽度固定为布局总宽度除以自适应得到的列数，高度为 match_content，每行中的所有盒子按高度最高的进行对齐；④该布局在水平方向自动分块，因此水平方向不支持 match_content，布局水平宽度仅支持 match_parent 或固定宽度；⑤自适应仅在水平方向进行自动分块，纵向没有限制，因此如果将某个子组件的高设置为 match_parent 类型，则可能导致后续行无法显示

下面我们通过几个案例练习 Java 组件的使用。

1. DirectionalLayout

1) 垂直布局

ability_main.xml 文件的代码如下：

```
/* 第 5 章/DirectionalLayout 案例垂直布局 ability_main.xml 代码 */
<?xml version = "1.0" encoding = "utf - 8"?>
< DirectionalLayout
    xmlns:ohos = "http://schemas.huawei.com/res/ohos"
    ohos:width = "match_parent"
    ohos:height = "match_content"
    ohos:orientation = "vertical">
    //按钮 1
    < Button
```

```
            ohos:width = "330vp"
            ohos:height = "200vp"
            ohos:bottom_margin = "30vp"
            ohos:text_size = "28vp"
            ohos:left_margin = "13vp"
            ohos:background_element = " $ graphic:background_ability_main"
            ohos:text = "Button 1"/>
        //按钮 2
        < Button
            ohos:width = "330vp"
            ohos:height = "200vp"
            ohos:bottom_margin = "30vp"
            ohos:text_size = "28vp"
            ohos:left_margin = "13vp"
            ohos:background_element = " $ graphic:background_ability_main"
            ohos:text = "Button 2"/>
        //按钮 3
        < Button
            ohos:width = "330vp"
            ohos:height = "200vp"
            ohos:text_size = "28vp"
            ohos:bottom_margin = "30vp"
            ohos:left_margin = "13vp"
            ohos:background_element = " $ graphic:background_ability_main"
            ohos:text = "Button 3"/>
</DirectionalLayout >
```

background_ability_main. xml 文件的代码如下：

```
/ * 第 5 章/DirectionalLayout 案例垂直布局 background_ability_main.xml 代码 * /
<?xml version = "1.0" encoding = "utf - 8"?>
< shape xmlns:ohos = "http://schemas.huawei.com/res/ohos"
        ohos:shape = "rectangle">
     < solid
          ohos:color = " # 00FFFD"/>
</shape >
```

2）水平布局

ability_main. xml 文件的代码如下：

```
/ * 第 5 章/DirectionalLayout 案例水平布局 ability_main.xml 代码 * /
<?xml version = "1.0" encoding = "utf - 8"?>
< DirectionalLayout
```

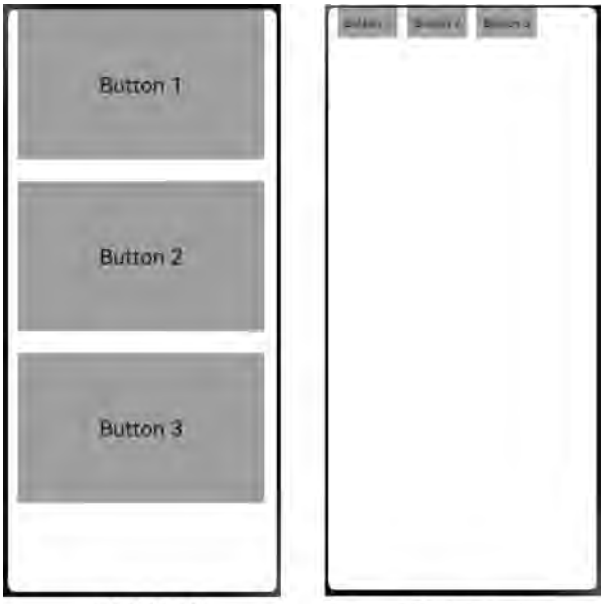
```
    xmlns:ohos = "http://schemas.huawei.com/res/ohos"
    ohos:width = "match_parent"
    ohos:height = "match_content"
    ohos:orientation = "horizontal">          //这里修改为水平布局
    < Button
        ohos:width = "80vp"
        ohos:height = "40vp"
        ohos:bottom_margin = "30vp"
        ohos:text_size = "16vp"
        ohos:left_margin = "13vp"
        ohos:background_element = " $ graphic:background_ability_main"
        ohos:text = "Button 1"/>
    < Button
        ohos:width = "80vp"
        ohos:height = "40vp"
        ohos:bottom_margin = "30vp"
        ohos:text_size = "16vp"
        ohos:left_margin = "13vp"
        ohos:background_element = " $ graphic:background_ability_main"
        ohos:text = "Button 2"/>
    < Button
        ohos:width = "80vp"
        ohos:height = "40vp"
        ohos:text_size = "16vp"
        ohos:bottom_margin = "30vp"
        ohos:left_margin = "13vp"
        ohos:background_element = " $ graphic:background_ability_main"
        ohos:text = "Button 3"/>
</DirectionalLayout >
```

background_ability_main.xml 文件的代码如下：

```
/ * 第 5 章/DirectionalLayout 案例水平布局 background_ability_main.xml 代码 * /
<?xml version = "1.0" encoding = "utf - 8"?>
< shape xmlns:ohos = "http://schemas.huawei.com/res/ohos"
        ohos:shape = "rectangle">
     < solid
         ohos:color = " # 00FFFD"/>
</shape >
```

3）效果展示

垂直布局效果如图 5-9(a)所示，水平布局效果如图 5-9(b)所示。



(a) 垂直布局　　(b) 水平布局

图 5-9　展示布局效果

2. StackLayout

1）堆叠布局

ability_main.xml 文件的代码如下：

```
/* 第 5 章/StackLayout 案例堆叠布局 ability_main.xml 代码 */
<?xml version = "1.0" encoding = "utf - 8"?>
< StackLayout
    xmlns:ohos = "http://schemas.huawei.com/res/ohos"
    ohos:id = " $ + id:stack_layout"
    ohos:height = "match_parent"
    ohos:width = "match_parent">

    < Text
        ohos:id = " $ + id:text_blue"
        ohos:text_alignment = "bottom|horizontal_center"
        ohos:text_size = "24fp"
        ohos:text = "Layer 1"
        ohos:height = "400vp"
        ohos:width = "400vp"
        ohos:background_element = " # 3F56EA" />

    < Text
        ohos:id = " $ + id:text_light_purple"
```

```
            ohos:text_alignment = "bottom|horizontal_center"
            ohos:text_size = "24fp"
            ohos:text = "Layer 2"
            ohos:height = "300vp"
            ohos:width = "300vp"
            ohos:background_element = "#00AAEE" />

        <Text
            ohos:id = "$ + id:text_orange"
            ohos:text_alignment = "center"
            ohos:text_size = "24fp"
            ohos:text = "Layer 3"
            ohos:height = "80vp"
            ohos:width = "80vp"
            ohos:background_element = "#00BFC9" />

    </StackLayout>
```

2）效果展示

效果如图 5-10 所示。

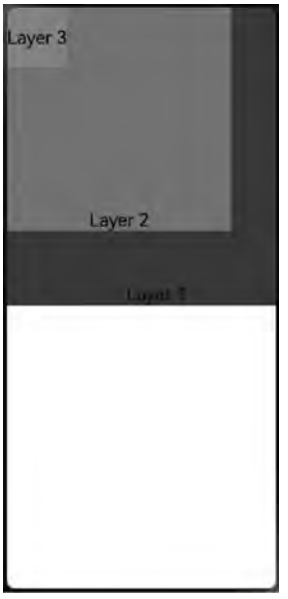


图 5-10　展示效果(1)

5.5.2　常用组件

Java 常用组件及功能使用说明，见表 5-15。

表 5-15 常用组件

名　称	说　明
Text	Text 是用来显示字符串的组件，在界面上显示为一块文本区域。Text 作为一个基本组件，有很多扩展，常见的有按钮组件 Button，文本编辑组件 TextField
Button	Button 是一种常见的组件，单击可以触发对应的操作，通常由文本或图标组成，也可以由图标和文本共同组成
TextField	TextField 提供了一种文本输入框
Image	Image 是用来显示图片的组件
TabList 和 Tab	TabList 可以实现多个页签栏的切换，Tab 为某个页签。子页签通常放在内容区上方，用于展示不同的分类。页签名称应该简洁明了，清晰描述分类的内容
Picker	Picker 提供了滑动选择器，允许用户从预定义范围中进行选择
DatePicker	DatePicker 主要供用户选择日期
TimePicker	TimePicker 主要供用户选择时间
Switch	Switch 是切换单个设置开/关两种状态的组件
RadioButton	RadioButton 用于多选一的操作，需要搭配 RadioContainer 使用，实现单选效果
RadioContainer	RadioContainer 是 RadioButton 的容器，在其包裹下的 RadioButton 保证只有一个被选项
Checkbox	Checkbox 可以实现选中和取消选中的功能
ProgressBar	ProgressBar 用于显示内容或操作的进度
RoundProgressBar	RoundProgressBar 继承自 ProgressBar，拥有 ProgressBar 的属性，在设置同样的属性时用法和 ProgressBar 一致，用于显示环形进度
ToastDialog	ToastDialog 是在窗口上方弹出的对话框，是通知操作的简单反馈，ToastDialog 会在一段时间后消失，在此期间，用户还可以操作当前窗口的其他组件
PopupDialog	气泡对话框是覆盖在当前界面之上的弹出框，可以相对组件或者屏幕显示。显示时会获取焦点，中断用户操作，被覆盖的其他组件无法交互。气泡对话框的内容一般简单明了，并提示用户一些需要确认的信息
CommonDialog	CommonDialog 是一种在弹出框消失之前，用户无法操作其他界面内容的对话框。通常用来展示用户当前需要的或用户必须关注的信息或操作。对话框的内容通常采用不同组件进行组合布局，如：文本、列表、输入框、网格、图标或图片，常用于选择或确认信息
ScrollView	ScrollView 是一种带滚动功能的组件，它采用滑动的方式在有限的区域内显示更多的内容
ListContainer	ListContainer 是用来呈现连续、多行数据的组件，包含一系列相同类型的列表项
PageSlider	PageSlider 是用于页面之间切换的组件，它通过响应滑动事件完成页面间的切换
PageSliderIndicator	PageSliderIndicator 需配合 PageSlider 使用，指示在 PageSlider 中展示哪个界面
WebView	WebView 提供在应用中集成 Web 页面的能力

下面我们通过几个案例练习 Java 组件的使用。

1. 日期选择 DatePicker

通过日期选择组件 DatePicker 选择日期并显示在文本框上。

1）演示案例代码示例

ability_main.xml 文件的代码如下：

```
/* 第 5 章/日期选择 DatePicker 案例 ability_main.xml 代码 */
<?xml version = "1.0" encoding = "utf - 8"?>
< DirectionalLayout
    xmlns:ohos = "http://schemas.huawei.com/res/ohos"
    ohos:height = "match_parent"
    ohos:width = "match_parent"
    ohos:alignment = "center"
    ohos:orientation = "vertical">

    < Text
        ohos:id = " $ + id:text_date"
        ohos:height = "match_content"
        ohos:width = "match_parent"
        ohos:hint = "日期"
        ohos:margin = "8vp"
        ohos:padding = "4vp"
        ohos:text_size = "28fp">
    </Text >

    < DatePicker
        ohos:id = " $ + id:date_pick"
        ohos:height = "300vp"
        ohos:width = "300vp"
        ohos:text_size = "20fp"
        ohos:background_element = " # C89FDEFF">
    </DatePicker >

</DirectionalLayout >
```

在 MainAbilitySlice.java 文件中编写显示逻辑，代码如下：

```
//第 5 章/日期选择 DatePicker 案例 MainAbilitySlice.java 代码
public class MainAbilitySlice extends AbilitySlice {
    @Override
    public void onStart(Intent intent) {
        super.onStart(intent);
        super.setUIContent(ResourceTable.Layout_ability_main);

        //获取 DatePicker 实例
        DatePicker datePicker = (DatePicker) findComponentById(ResourceTable.Id_date_pick);

        Text selectedDate = (Text) findComponentById(ResourceTable.Id_text_date);
        datePicker.setValueChangedListener(
                new DatePicker.ValueChangedListener() {
                    @Override
```

```
                            public void onValueChanged(DatePicker datePicker, int year, int
monthOfYear, int dayOfMonth) {
                                selectedDate.setText(String.format("%02d/%02d/%4d",
dayOfMonth, monthOfYear, year));
                        }
                    }
            );
        }

        @Override
        public void onActive() {
            super.onActive();
        }

        @Override
        public void onForeground(Intent intent) {
            super.onForeground(intent);
        }
}
```

2）演示案例展示效果

DatePicker 日期选择演示案例效果，如图 5-11 所示。

图 5-11　展示效果(2)

2. 弹框组件 CommonDialog

CommonDialog 是一种在弹出框消失之前，用户无法操作其他界面内容的对话框。通常用来展示用户当前需要的或用户必须关注的信息或操作。对话框的内容通常采用不同组件进行组合布局，如：文本、列表、输入框、网格、图标或图片，常用于选择或确认信息。

1）演示案例代码示例

编写页面代码，代码如下：

```
/* 第 5 章/弹框组件 CommonDialog 案例 ability_main.xml 代码 */
<?xml version = "1.0" encoding = "utf - 8"?>
< DependentLayout
    xmlns:ohos = "http://schemas.huawei.com/res/ohos"
    ohos:id = " $ + id:custom_container"
    ohos:height = "match_content"
    ohos:width = "300vp"
    ohos:background_element = " # F5F5F5">

    < Text
        ohos:id = " $ + id:custom_title"
        ohos:height = "match_content"
        ohos:width = "match_content"
        ohos:horizontal_center = "true"
        ohos:margin = "8vp"
        ohos:padding = "8vp"
        ohos:text = "TITLE"
        ohos:text_size = "16fp"/>
    < Image
        ohos:id = " $ + id:custom_img"
        ohos:height = "150vp"
        ohos:width = "280vp"
        ohos:below = " $ id:custom_title"
        ohos:horizontal_center = "true"
        ohos:image_src = " $ media:transparent_bg"
        ohos:clip_alignment = "center"/>
    < Button
        ohos:height = "match_content"
        ohos:width = "match_parent"
        ohos:background_element = " # FF9912"
        ohos:below = " $ id:custom_img"
        ohos:margin = "8vp"
        ohos:padding = "8vp"
        ohos:text = "BUTTON"
        ohos:text_color = " # FFFFFF"
        ohos:text_size = "18vp"/>
</DependentLayout >
```

在 AbilitySlice.java 文件中编写显示逻辑,代码如下:

```
//第 5 章/弹框组件 CommonDialog 案例 AbilitySlice.java 代码
import static ohos.agp.components.ComponentContainer.LayoutConfig.MATCH_CONTENT;  //注意引入

public class MainAbilitySlice extends AbilitySlice {
    @Override
    public void onStart(Intent intent) {
        super.onStart(intent);
        super.setUIContent(ResourceTable.Layout_ability_main);

        findComponentById(ResourceTable.Id_btn).setClickedListener(new Component.
ClickedListener() {
            @Override
            public void onClick(Component component) {
                CommonDialog dialog = new CommonDialog(getContext());
                Component container = LayoutScatter.getInstance(getContext()).parse
(ResourceTable.Layout_layout_custom_dialog, null, false);
                dialog.setContentCustomComponent(container);
                dialog.setSize(MATCH_CONTENT, MATCH_CONTENT);
                dialog.show();
            }
        });

    }

    @Override
    public void onActive() {
        super.onActive();
    }

    @Override
    public void onForeground(Intent intent) {
        super.onForeground(intent);
    }
}
```

2) 演示案例展示效果

CommonDialog 弹框组件演示案例效果,如图 5-12 所示。

3. ScrollView 组件使用

ScrollView 是一种带滚动功能的组件,它采用滑动的方式在有限的区域内显示更多的内容。

1) 演示案例代码示例

编写页面代码,代码如下:

图 5-12　展示效果(3)

```
/* 第 5 章/滚动功能组件 ScrollView 案例 ability_main.xml 代码 */
<?xml version = "1.0" encoding = "utf - 8"?>
< DirectionalLayout
    xmlns:ohos = "http://schemas.huawei.com/res/ohos"
    ohos:height = "match_parent"
    ohos:width = "match_parent"
    ohos:alignment = "center"
    ohos:orientation = "vertical">
    < ScrollView
        ohos:id = " $ + id:scrollview"
        ohos:height = "300vp"
        ohos:width = "300vp"
        ohos:background_element = " # FFDEAD"
        ohos:top_margin = "32vp"
        ohos:bottom_padding = "16vp"
        ohos:layout_alignment = "horizontal_center">
        < DirectionalLayout
            ohos:height = "match_content"
            ohos:width = "match_content">
            <!-- $ media:plant 为在 media 目录引用的图片资源 -->
            < Image
                ohos:width = "300vp"
                ohos:height = "match_content"
                ohos:top_margin = "16vp"
                ohos:image_src = " $ media:show"/>
            < Image
                ohos:width = "300vp"
                ohos:height = "match_content"
```

```
                ohos:top_margin = "16vp"
                ohos:image_src = " $ media:show"/>
            < Image
                ohos:width = "300vp"
                ohos:height = "match_content"
                ohos:top_margin = "16vp"
                ohos:image_src = " $ media:show"/>
        </DirectionalLayout >
    </ScrollView >
</DirectionalLayout >
```

2）演示案例展示效果

ScrollView 滚动功能组件演示案例效果，如图 5-13 所示。

图 5-13　展示效果(4)

5.5.3　自定义组件与布局

1. 自定义组件与布局概述

（1）HarmonyOS 提供了一套复杂且强大的 Java UI 框架，其中 Component 提供内容显示，是界面中所有组件的基类。ComponentContainer 作为容器容纳 Component 或 ComponentContainer 对象，并对它们进行布局。

（2）Java UI 框架也提供了一部分 Component 和 ComponentContainer 的具体子类，即常用的组件（例如：Text、Button、Image 等）和常用的布局（例如：DirectionalLayout、DependentLayout 等）。如果现有的组件和布局无法满足设计需求，例如仿遥控器的圆盘按钮、可滑动的环形控制器等，可以通过自定义组件和自定义布局实现。

(3) 自定义组件是由开发者定义的具有一定特性的组件,通过扩展 Component 或其子类实现,可以精确控制屏幕元素的外观,也可响应用户的单击、触摸、长按等操作。

(4) 自定义布局是由开发者定义的具有特定布局规则的容器类组件,通过扩展 ComponentContainer 或其子类实现,可以将各子组件摆放到指定的位置,也可响应用户的滑动、拖曳等事件。

2. 自定义组件

当 Java UI 框架提供的组件无法满足设计需求时,可以创建自定义组件,根据设计需求添加绘制任务,并定义组件的属性及事件响应,完成组件的自定义。

Component 类相关接口名称及作用,见表 5-16。

表 5-16 Component 类相关接口

接 口 名	作 用
setEstimateSizeListener	设置测量组件的侦听器
setEstimatedSize	设置测量的宽度和高度
onEstimateSize	测量组件的大小以确定宽度和高度
EstimateSpec. getChildSizeWithMode	基于指定的大小和模式为子组件创建度量规范
EstimateSpec. getSize	从提供的度量规范中提取大小
EstimateSpec. getMode	获取该组件的显示模式
addDrawTask	添加绘制任务
onDraw	通过绘制任务更新组件时调用

3. 自定义布局

Component 类相关接口名称及作用,见表 5-17。

表 5-17 Component 类相关接口

接 口 名 称	作 用
setEstimateSizeListener	设置测量组件的侦听器
setEstimatedSize	设置测量的宽度和高度
onEstimateSize	测量组件的大小以确定宽度和高度
EstimateSpec. getChildSizeWithMode	基于指定的大小和模式为子组件创建度量规范
EstimateSpec. getSize	从提供的度量规范中提取大小
EstimateSpec. getMode	获取该组件的显示模式
arrange	相对于容器组件设置组件的位置和大小

ComponentContainer 类相关接口名称及作用,见表 5-18。

表 5-18 ComponentContainer 类相关接口

接 口 名 称	作 用
setArrangeListener	设置容器组件布局子组件的侦听器
onArrange	通知容器组件在布局时设置子组件的位置和大小

5.6 eTS 组件开发

本节主要对 eTS 组件进行整体介绍并进行相关案例实战练习，eTS 组件从 API Version 7 才开始支持使用。

5.6.1 通用事件

1. 单击事件

单击事件相关内容，见表 5-19。

表 5-19 单击事件

名　　称	是否冒泡	功 能 描 述
onClick(callback：(event?：ClickEvent) => void)	否	单击动作触发该方法调用

2. 触摸事件

触摸事件相关内容，见表 5-20。

表 5-20 触摸事件

名　　称	是否冒泡	功 能 描 述
onTouch(callback：(event?：TouchEvent) => void)	是	触摸动作触发该方法调用

3. 挂载卸载事件

挂载卸载事件相关内容，见表 5-21。

表 5-21 挂载卸载事件

名　　称	是否冒泡	功 能 描 述
onAppear(callback：() => void)	否	组件挂载显示时触发此回调
onDisappear(callback：() => void)	否	组件卸载消失时触发此回调

4. 按键事件

按键事件相关内容，见表 5-22。

表 5-22 按键事件

名　　称	是否冒泡	功 能 描 述
onKeyEvent(event：(event?：KeyEvent) => void)	是	按键动作触发该方法调用

5.6.2 通用属性

1. 尺寸设置

尺寸设置相关内容，见表 5-23。

表 5-23 尺寸设置

名 称	参数说明	默认值	描 述
width	Length	—	设置组件自身的宽度，缺省时使用元素自身内容需要的宽度
height	Length	—	设置组件自身的高度，缺省时使用元素自身内容需要的高度

size	{
	width?: Length,
	height?: Length
	}
		—	设置高宽尺寸

padding	{ top?: Length, right?: Length, bottom?: Length, left?: Length }|Length	0	设置内边距属性，参数为 Length 类型时，4 个方向内边距同时生效

margin	{
	top?: Length,
	right?: Length,
	bottom?: Length,
	left?: Length
	}
	|Length
		0	设置外边距属性，参数为 Length 类型时，4 个方向外边距同时生效

constraintSize	{
	minWidth?: Length,
	maxWidth?: Length,
	minHeight?: Length,
	maxHeight?: Lenght
	}
		{
		minWidth: 0,
		maxWidth: Infinity,
		minHeight: 0,
		maxHeight: Infinity
		}
			设置约束尺寸，组件布局时，进行尺寸范围限制

layoutWeight	number	0	容器尺寸确定时，元素与兄弟节点主轴布局尺寸按照权重进行分配，忽略本身尺寸设置（说明：仅在 Row/Column/Flex 布局中生效）

2. 位置设置

位置设置相关内容，见表 5-24。

表 5-24 位置设置

名 称	参数类型	默认值	描 述
align	Alignment	Center	设置元素内容的对齐方式，只有当设置的 width 和 height 大小超过元素本身内容大小时生效
direction	Direction	Auto	设置元素水平方向的布局

position	{
	x：Length，
	y：Length
	}
		—	使用绝对定位，设置元素锚点相对于父容器顶部起点偏移位置。在布局容器中，设置该属性不影响父容器布局，仅在绘制时进行位置调整

markAnchor	{
	x：Length，
	y：Length
	}
		{
		x：0，
		y：0
		}
			设置元素在位置定位时的锚点，以元素顶部起点作为基准点进行偏移

offset	{
	x：Length，
	y：Length
	}
		{
		x：0，
		y：0
		}
			相对布局完成位置坐标偏移量，设置该属性，不影响父容器布局，仅在绘制时进行位置调整

3. 布局约束

布局约束相关内容，见表 5-25。

表 5-25 布局设置

名 称	参数说明	默认值	描 述
aspectRatio	number	—	指定当前组件的宽高比
displayPriority	number	—	设置当前组件在布局容器中显示的优先级，当父容器空间不足时，低优先级的组件会被隐藏（说明：仅在 Row/Column/Flex(单行)容器组件中生效）

4. Flex 布局

Flex 布局相关内容，见表 5-26。

表 5-26 Flex 布局

名 称	参数说明	默认值	描 述
flexBasis	'auto'|Length	'auto'	此属性所在的组件在 Flex 容器中主轴方向上的基准尺寸
flexGrow	number	0	Flex 容器的剩余空间分配给此属性所在的组件的比例
flexShrink	number	1	Flex 容器压缩尺寸分配给此属性所在的组件的比例
alignSelf	ItemAlign	Auto	覆盖 Flex 布局容器中 alignItems 默认配置

5. 边框设置

边框设置相关内容，见表 5-27。

表 5-27 边框设置

名 称	参数类型	默认值	描 述

border	{
	width?: Length,
	color?: Color,
	radius?: Length,
	style?:
	BorderStyle
	}
		—	统一边框样式设置接口

borderStyle	BorderStyle	Solid	设置元素的边框样式
borderWidth	Length	0	设置元素的边框宽度
borderColor	Color	-	设置元素的边框颜色
borderRadius	Length	0	设置元素的边框圆角半径

6. 背景设置

背景设置相关内容，见表 5-28。

表 5-28 背景设置

名 称	参数类型	默认值	描 述
backgroundColor	Color	—	设置组件的背景色
backgroundImage	src: string, repeat?: ImageRepeat	—	src 参数：图片地址，支持网络图片资源和本地图片资源地址(不支持 svg 类型的图片)。repeat 参数：设置背景图片的重复样式，默认不重复
backgroundImageSize	{ width?: Length, height?: Length }|ImageSize	Auto	设置背景图像的高度和宽度。当输入为{width: Length, height: Length}对象时，如果只设置一个属性，则第 2 个属性保持图片原始宽高比进行调整。默认保持原图的比例不变

backgroundImagePosition	{
	x?: Length,
	y?: Length
	}|Alignment
		{
		x: 0,
		y: 0
		}
			设置背景图在组件中的显示位置

7. 透明度设置

透明度设置相关内容，见表 5-29。

表 5-29 透明度设置

名称	参数类型	默认值	描 述
opacity	number	1	元素的不透明度，取值范围为 0～1，1 表示不透明，0 表示完全透明

8. 显隐控制

显隐设置相关内容，见表 5-30。

表 5-30 显隐设置

名称	参数类型	默认值	描 述
visibility	Visibility	Visible	控制当前组件显示或隐藏

9. 禁用控制

禁用设置相关内容，见表 5-31。

表 5-31 禁用设置

名称	参数类型	默认值	描 述
enabled	boolean	true	值为 true 表示组件可用，可响应单击等操作；当值为 false 时，不响应单击等操作

10. 浮层

浮层相关内容，见表 5-32。

表 5-32 浮层

名称	参数类型	默认值	描 述

overlay	title：string，
	options：{
	align?：Alignment，
	offset?：{x：number，y：number}
	}
		{
		align：Alignment.Center，
		offset：{0，0}
		}
			在当前组件上，增加遮罩文本，布局与当前组件相同

11. Z 序控制

Z 序控制相关内容，见表 5-33。

表 5-33 Z 序控制

名称	参数类型	默认值	描 述
zIndex	number	0	同一容器中兄弟组件显示层级关系，z 值越大，显示层级越高

12. 图形变换

图形变换相关内容，见表 5-34。

13. 图像效果

图像效果相关内容，见表 5-35。

表 5-34 图形变换

名 称	参数类型	默认值	描 述

rotate	{
	x?: number,
	y?: number,
	z?: number,
	angle?: Angle,
	centerX?: Length,
	centerY?: Length
	}
		{
		x: 0,
		y: 0,
		z: 0,
		angle: 0,
		centerX: '50%',
		centerY: '50%'
		}
			(x, y, z)指定一个向量,表示旋转轴,正角度为顺时针转动,负角度为逆时针转动,默认值为 0,同时可以通过 centerX 和 centerY 设置旋转的中心点

translate	{
	x?: Length,
	y?: Length,
	z? : Length
	}
		{
		x: 0,
		y: 0,
		z: 0
		}
			可以分别设置 X 轴、Y 轴、Z 轴的平移距离,距离的正负可控制平移的方向,默认值为 0

scale	{
	x?: number,
	y?: number,
	z?: number,
	centerX?: Length,
	centerY?: Length
	}
		{
		x: 1,
		y: 1,
		z: 1,
		centerX:'50%',
		centerY:'50%'
		}
			可以分别设置 X 轴、Y 轴、Z 轴的缩放比例,默认值为 1,同时可以通过 centerX 和 centerY 设置缩放的中心点

transform	matrix: Matrix4	-	设置当前组件的变换矩阵

表 5-35 图像效果

名 称	参数类型	默认值	描 述
blur	number	—	为当前组件添加内容模糊效果,入参为模糊半径,模糊半径越大越模糊,为 0 时不模糊
backdropBlur	number	—	为当前组件添加背景模糊效果,入参为模糊半径,模糊半径越大越模糊,为 0 时不模糊

shadow	{
	radius: number,
	color?: Color,
	offsetX?:
	number,
	offsetY?:
	number
	}
		—	为当前组件添加阴影效果,入参为模糊半径(必填)、阴影的颜色(可选,默认为灰色)、X 轴的偏移量(可选,默认为 0),Y 轴的偏移量(可选,默认为 0),偏移量单位为 px

续表

名　称	参数类型	默认值	描　述
grayscale	number	0.0	为当前组件添加灰度效果。将值定义为灰度转换的比例，入参为 1.0 时完全转换为灰度图像，入参为 0.0 时图像无变化，入参在 0.0 和 1.0 之间时，效果呈线性变化(百分比)
brightness	number	1.0	为当前组件添加高光效果，入参为高光比例，值为 1 时没有效果，值小于 1 时亮度变暗，值为 0 时全黑；值大于 1 时亮度增加，数值越大亮度越大
saturate	number	1.0	为当前组件添加饱和度效果，饱和度为颜色中的含色成分和消色成分(灰)的比例，入参为 1 时，显示原图像，值大于 1 时含色成分越大，饱和度越大；值小于 1 时消色成分越大，饱和度越小(百分比)
contrast	number	1.0	为当前组件添加对比度效果，入参为对比度的值，值为 1 时，显示原图；值大于 1 时，值越大对比度越高，图像越清晰醒目；值小于 1 时，值越小对比度越低；当对比度为 0 时，图像变为全灰(百分比)
invert	number	0	反转输入的图像。入参为图像反转的比例。值为 1 时完全反转。值为 0 时图像无变化(百分比)
sepia	number	0	将图像转换为深褐色。入参为图像反转的比例。值为 1 时完全是深褐色的，值为 0 时图像无变化(百分比)
hueRotate	Angle	0deg	为当前组件添加色相旋转效果，入参为旋转的角度值。当入参为 0deg 时图像无变化(默认值是 0deg)，入参没有最大值，超过 360deg 的值相当于又绕一圈

14. 形状剪裁

形状剪裁相关内容，见表 5-36。

表 5-36　形状剪裁

名称	参数类型	默认值	描　述
clip	Shape|boolean	false	当参数为 Shape 类型时，按指定的形状对当前组件进行裁剪；当参数为 boolean 类型时，设置是否按照边缘轮廓进行裁剪
mask	Shape	—	在当前组件上加上指定形状的遮罩

15. 文本样式设置

文本样式设置相关内容，设置见表 5-37。

表 5-37 文本样式设置

名 称	参数类型	默认值	描 述
fontColor	Color	—	设置文本颜色
fontSize	Length	—	设置文本尺寸，Length 为 number 类型时，使用 fp 单位
fontStyle	FontStyle	Normal	设置文本的字体样式
fontWeight	number|FontWeight	Normal	设置文本的字体粗细，number 类型的取值为[100，900]，取值间隔为 100，默认为 400，取值越大，字体越粗。提供常用枚举值，参考：FontWeight
fontFamily	string	—	设置文本的字体列表。使用多种字体，使用','进行分割，优先级按顺序生效（例如：Arial，sans-serif）

16. 栅格设置

栅格设置相关内容，见表 5-38。

表 5-38 栅格设置

名 称	参数类型	默认值	描 述

useSizeType	{
	xs?: number|{ span: number, offset: number },
	sm?: number|{ span: number, offset: number },
	md?: number|{ span: number, offset: number },
	lg?: number|{ span: number, offset: number }
	}
		—	设置在特定设备宽度类型下的占用列数和偏移列数，span 指占用列数。
			offset：指偏移列数；当值为 number 类型时，仅设置列数；当格式如{"span": 1, "offset": 0}时，指同时设置占用列数与偏移列数；xs 指设备宽度类型为 SizeType.XS 时的占用列数和偏移列数；sm：指设备宽度类型为 SizeType.SM 时的占用列数和偏移列数；md：指设备宽度类型为 SizeType.MD 时的占用列数和偏移列数；lg 指设备宽度类型为 SizeType.LG 时的占用列数和偏移列数

gridSpan	number	1	默认占用列数，指 useSizeType 属性没有设置对应尺寸的列数（span）时，占用的栅格列数（说明：设置了栅格 span 属性，组件的宽度由栅格布局决定）
gridOffset	number	0	默认偏移列数，指 useSizeType 属性没有设置对应尺寸的偏移（offset）时，当前组件沿着父组件 Start 方向，偏移的列数，也是当前组件位于第 n 列（说明：①配置该属性后，当前组件在父组件水平方向的布局不再跟随父组件原有的布局方式，而是沿着父组件的 Start 方向偏移一定位移；②偏移位移=（列宽+间距）×列数；③设置了偏移（gridOffset）的组件之后的兄弟组件会根据该组件进行相对布局，类似于相对布局）

17. 颜色渐变

颜色渐变相关内容，见表 5-39。

表 5-39 颜色渐变

名 称	参数类型	默认值	描 述

linearGradient	{
	angle?: Angle,
	direction?:
	GradientDirection,
	colors: Array<ColorStop>
	repeating?: boolean
	}
		—	线性渐变的参数如下。
			angle：线性渐变的角度；
			direction：线性渐变的方向；
			colors：为渐变的颜色描述；
			repeating：为渐变的颜色重复着色

sweepGradient	{
	center: Point,
	start?: angle,
	end?: angle,
	colors: Array<ColorStop>
	repeating?: boolean
	}
		—	角度渐变的参数如下。
			center：为角度渐变的中心点；
			start：角度渐变的起点；
			end：角度渐变的终点；
			colors：为渐变的颜色描述；
			repeating：为渐变的颜色重复着色

radialGradient	{
	center: Point,
	radius: Length,
	colors: Array<ColorStop>
	repeating: boolean
	}
		—	径向渐变的参数如下。
			center：径向渐变的中心点；
			radius：径向渐变的半径；
			colors：为渐变的颜色描述；
			repeating：为渐变的颜色重复着色

18. Popup 控制

Popup 控制相关内容，见表 5-40。

表 5-40 Popup 控制

名 称	参数类型	默认值	参数描述

bindPopup	{
	show: boolean,
	popup: {
	message: string,
	placementOnTop: boolean,
	primaryButton?: {
	value: string,
	action: () => void
	},
	secondaryButton?:{
	value: string,
	action: () => void
	},
	onStateChange?: (isVisible:
	boolean) => void
	}
	}
		—	show：当前弹窗提示是否显示，默认值为 false。
			message：弹窗信息内容。
			placementOnTop：是否在组件上方显示，默认值为 false。
			primaryButton：第 1 个按钮。
			secondaryButton：第 2 个按钮。
			onStateChange：弹窗状态变化事件回调

19. Menu 控制

Menu 控制相关内容，见表 5-41。

表 5-41 Menu 控制

名 称	参数类型	默认值	描 述
bindMenu	Array < MenuItem >	—	给组件绑定菜单，单击后弹出菜单

5.6.3 手势处理

1. 绑定手势方法

绑定手势方法相关内容，见表 5-42。

表 5-42 绑定手势方法

名 称	参数类型	默认值	描 述
gesture	gesture: GestureType, mask?: GestureMask	gesture: -, mask: GestureMask.Normal	绑定手势识别。gesture：为绑定的手势类型，mask 为事件响应设置
priorityGesture	gesture: GestureType, mask?: GestureMask	gesture: -, mask: GestureMask.Normal	绑定优先识别手势。gesture 为绑定的手势类型，mask 为事件响应设置。说明：默认情况下，子组件优先于父组件识别手势，当父组件配置 priorityGesture 时，父组件优先于子组件进行识别
parallelGesture	gesture: GestureType, mask?: GestureMask	gesture: -, mask: GestureMask.Normal	绑定可与子组件手势同时触发的手势。gesture 为绑定的手势类型，mask 为事件响应设置。说明：手势事件为非冒泡事件。当父组件设置 parallelGesture 时，父子组件相同的手势事件都可以触发，实现类似冒泡效果

2. 基础手势

基础手势相关内容，见表 5-43。

表 5-43 基础手势

接 口	事件名称	功能描述
TapGesture(options?: { count?: number, fingers?: number })	onAction((event?: GestureEvent) => void)	Tap 手势识别成功回调

续表

接　口	事件名称	功能描述
LongPressGesture(options?: { fingers?: number, repeat?: boolean, duration?: number })	onAction((event?: GestureEvent) => void)	LongPress 手势识别成功回调
	onActionEnd((event?: GestureEvent) => void)	LongPress 手势识别成功,手指抬起后触发回调
	onActionCancel(event: () => void)	LongPress 手势识别成功,接收到触摸取消事件时触发回调
PanGesture(options?: { fingers?: number, direction?: PanDirection, distance?: number }| PanGestureOption)	onActionStart((event?: GestureEvent) => void)	Pan 手势识别成功回调
	onActionUpdate((event?: GestureEvent) => void)	Pan 手势移动过程中回调
	onActionEnd((event?: GestureEvent) => void)	Pan 手势识别成功,手指抬起后触发回调
	onActionCancel(event: () => void)	Pan 手势识别成功,接收到触摸取消事件时触发回调
PinchGesture(options?: { fingers?: number, distance?: number })	onActionStart((event?: GestureEvent) => void)	Pinch 手势识别成功回调
	onActionUpdate((event?: GestureEvent) => void)	Pinch 手势移动过程中回调
	onActionEnd((event?: GestureEvent) => void)	Pinch 手势识别成功,手指抬起后触发回调
	onActionCancel(event: () => void)	Pinch 手势识别成功,接收到触摸取消事件时触发回调
RotationGesture(options?: { fingers?: number, angle?: number })	onActionStart((event?: GestureEvent) => void)	Rotation 手势识别成功回调
	onActionUpdate((event?: GestureEvent) => void)	Rotation 手势移动过程中回调
	onActionEnd((event?: GestureEvent) => void)	Rotation 手势识别成功,手指抬起后触发回调
	onActionCancel(event: () => void)	Rotation 手势识别成功,接收到触摸取消事件时触发回调

3. 组合手势

组合手势相关内容,见表 5-44。

表 5-44　组合手势

接　口	事件名称	功能描述
GestureGroup(mode: GestureMode, ...gesture: GestureType[])	onCancel(event: () => void)	顺序组合手势(GestureMode.Sequence)取消后触发回调

5.6.4　基础组件

基础组件的主要类别名称与相关说明,见表 5-45。

表 5-45 基础组件

名 称	说 明
Blank	空白填充组件,在容器主轴方向上,空白填充组件具有自动填充容器空余部分的能力
Button	提供按钮组件
DataPanel	数据面板组件,用于将多个数据的占比情况使用环形占比图进行展示
Divider	提供分隔器组件,分隔不同内容块/内容元素
Image	图片组件,用来渲染展示图片
ImageAnimator	提供帧动画组件,实现逐帧播放图片的能力,可以配置需要播放的图片列表,每张图片可以配置时长
Progress	进度条,用于显示内容加载或操作处理进度
QRCode	显示二维码信息
Rating	评分条组件
Span	文本段落,只能作为 Text 子组件,呈现一段文本信息
Slider	滑动条组件,用来快速调节设置值,如音量、亮度等
Text	文本,用于呈现一段信息

下面我们通过几个案例练习 eTS 基础组件的使用。

1. Blank

Blank 是空白填充组件,在容器主轴方向上,空白填充组件具有自动填充容器空余部分的能力。

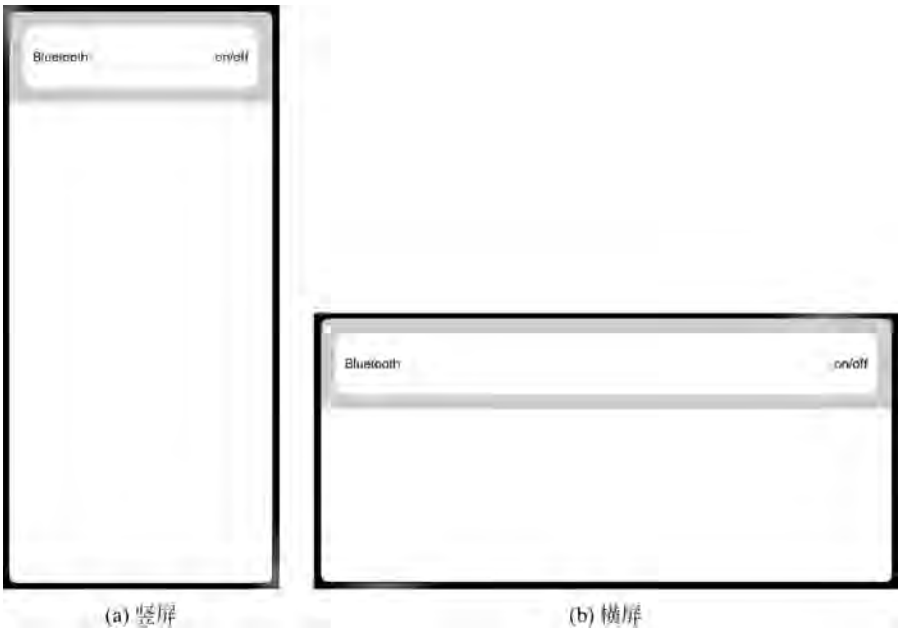
1) 演示案例示例代码

index.ets 文件的代码如下:

```
//第 5 章/基础组件 Blank 示例 index.ets 代码
@Entry
@Component
struct BlankExample {
  build() {
     //纵轴布局
    Column() {
     //水平布局
      Row() {
        Text('Bluetooth').fontSize(18)
     //填充容器
        Blank()
        Text('on/off').fontSize(18).height(60)
      }.width('100 % ').backgroundColor(0xFFFFFF).borderRadius(15).padding(12)
    }.backgroundColor(0xcccccc).padding(20)
  }
}
```

2）演示案例效果展示

Blank 空白填充组件演示案例的竖屏效果如图 5-14(a)所示，横屏效果如图 5-14(b)所示。



(a) 竖屏　　(b) 横屏

图 5-14　展示效果(5)

2. DataPanel

DataPanel 是数据面板组件，用于将多个数据的占比情况使用环形占比图进行展示。

1）演示案例示例代码

index.ets 文件的代码如下：

```
//第 5 章/基础组件 DataPanel 示例 index.ets 代码
@Entry
@Component
struct DataPanelExample {
  //数据占比
  public values1: number[] = [30, 20, 20, 10, 10]
  build() {
    Column({ space: 10 }) {
      //数据面板
      DataPanel({ values: this.values1, max: 100 })
        .width(150)
        .height(150)
    }.width('100%').margin({ top: 5 })
  }
}
```

2）演示案例效果展示

DataPanel 数据面板组件演示案例效果如图 5-15 所示。

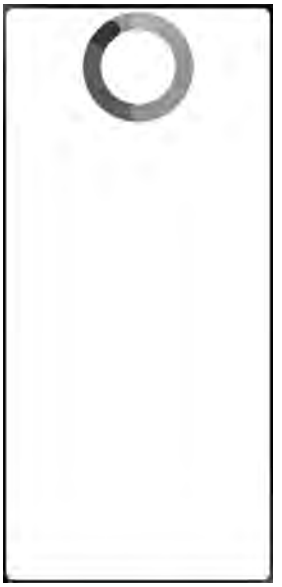


图 5-15　展示效果(6)

3. Rating

Rating 是评分条组件，用于用户各场景下进行各项评分时使用。

1）演示案例示例代码

index.ets 文件的代码如下：

```
//第 5 章/基础组件 Rating 示例 index.ets 代码
@Entry
@Component
struct RatingExample {
  @State rating: number = 1
  @State indicator: boolean = false

  build() {
// Flex 布局
    Flex({ direction: FlexDirection.Column, alignItems: ItemAlign.Center, justifyContent:
FlexAlign.SpaceBetween }) {
      Text('current score is ' + this.rating).fontSize(20)
//评分条组件
      Rating({ rating: this.rating, indicator: this.indicator })
        .stars(5)
        .stepSize(0.5)
```

```
// 监听事件
        .onChange((value: number) => {
          this.rating = value
        })
    }.width(350).height(200).padding(35)
  }
}
```

2）演示案例效果展示

Rating 评分条组件演示案例效果，如图 5-16 所示。

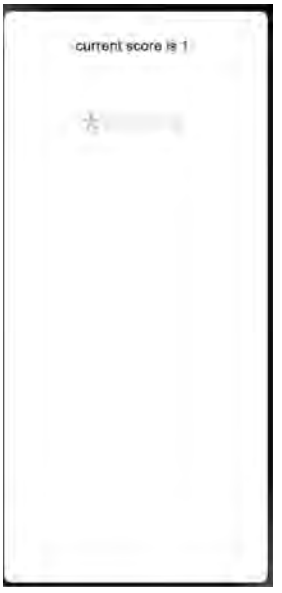


图 5-16　展示效果(7)

5.6.5　容器组件

容器组件的类别名称和相关说明，见表 5-46。

表 5-46　容器组件

名　称	说　明
AlphabetIndexer	字母索引条
Badge	新事件标记组件，在组件上提供事件信息展示能力
Column	沿垂直方向布局的容器
ColumnSplit	将子组件纵向布局，并在每个子组件之间插入一根横向的分割线
Counter	计数器组件，提供相应的增加或者减少的计数操作

续表

名　　称	说　　明
Flex	弹性布局组件
GridContainer	纵向排布栅格布局容器，仅在栅格布局场景中使用
Grid	网格容器，二维布局，将容器划分成“行”和“列”，产生单元格，然后指定“项目所在”的单元格，可以任意组合不同的网格，做出各种各样的布局
GridItem	网格容器中单项内容容器
Hyperlink	超链接组件，给子组件范围
List	列表包含一系列相同宽度的列表项。适合连续、多行呈现同类数据，例如图片和文本
ListItem	用来展示列表的具体 item，宽度默认充满 List 组件，必须配合 List 来使用
Navigator	路由容器组件，提供路由跳转能力
Panel	可滑动面板。提供一种轻量的内容展示的窗口，可方便地在不同尺寸中切换，属于弹出式组件
Row	沿水平方向布局容器
RowSplit	将子组件横向布局，并在每个子组件之间插入一根纵向的分割线
Scroll	可滚动的容器组件，当子组件的布局尺寸超过父组件的视口时，内容可以滚动
Stack	堆叠容器，子组件按照顺序依次入栈，后一个子组件覆盖前一个子组件
Swiper	滑动容器，提供切换子组件显示的能力
Tabs	一种可以通过页签进行内容视图切换的容器组件，每个页签对应一个内容视图
TabContent	仅在 Tabs 中使用，对应一个切换页签的内容视图

下面我们通过几个案例练习 eTS 容器组件的使用。

1. Column

Column 是沿垂直方向布局的容器。

1）演示案例示例代码

index.ets 文件的代码如下：

```
//第 5 章/容器组件 Column 示例 index.ets 代码
@Entry
@Component
struct ColumnExample {
  build() {
    //纵轴垂直
    Column({ space: 5 }) {
      Text('space').fontSize(24).fontColor(0x333333).width('90 % ')
      Column({ space: 5 }) {
        Column().width('100 % ').height(50).backgroundColor(0xAFEEEE)
        Column().width('100 % ').height(50).backgroundColor(0x00FFFF)
      }.width('90 % ').height(107).border({ width: 1 })
    //开头垂直

Text('alignItems(Start)').fontSize(24).fontColor(0x333333).width('90 % ')
      Column() {
```

```
        Column().width('50%').height(50).backgroundColor(0xAFEEEE)
        Column().width('50%').height(50).backgroundColor(0x00FFFF)
      }.alignItems(HorizontalAlign.Start).width('90%').border({ width: 1 })
    //结尾垂直

Text('alignItems(End)').fontSize(24).fontColor(0x333333).width('90%')
      Column() {
        Column().width('50%').height(50).backgroundColor(0xAFEEEE)
        Column().width('50%').height(50).backgroundColor(0x00FFFF)
      }.alignItems(HorizontalAlign.End).width('90%').border({ width: 1 })
    }.width('100%').padding({ top: 5 })
  }
}
```

2）演示案例效果展示

Column 沿垂直方向布局的容器演示案例效果，如图 5-17 所示。

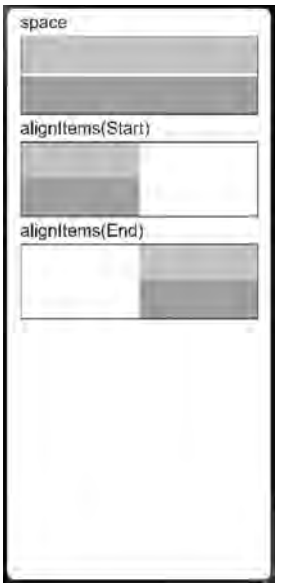


图 5-17　展示效果(8)

2. Counter

Counter 是计数器组件，提供相应的增加或者减少的计数操作。

1）演示案例示例代码

index.ets 文件的代码如下：

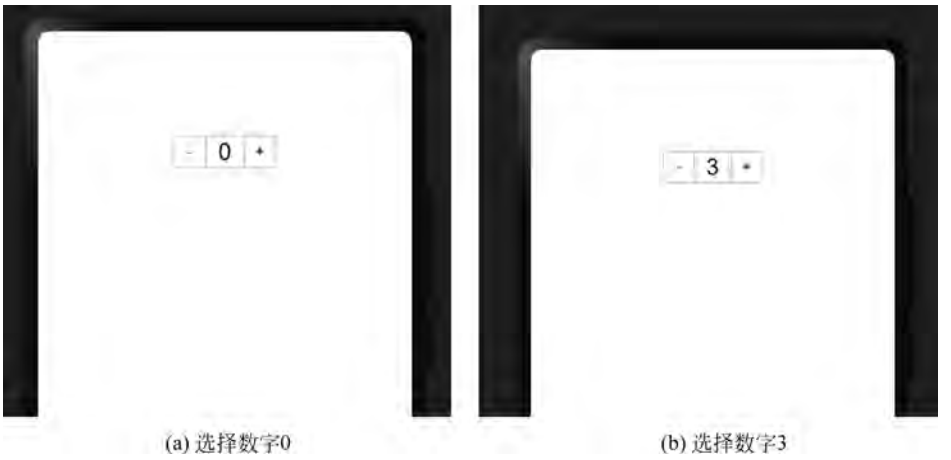
```
//第 5 章/容器组件 Counter 示例 index.ets 代码
@Entry
@Component
```

```
struct CounterExample {
  @State value: number = 0

  build() {
    Column() {
      //计数器
      Counter() {
        Text(this.value.toString())
        .fontSize(22)
      }
      .margin(100)
      //加
      .onInc(() => {
        this.value++
      })
      //减
      .onDec(() => {
        this.value--
      })
    }.width("100%")
  }
}
```

2）演示案例效果展示

选择数字 0 的效果如图 5-18(a)所示，选择数字 3 的效果如图 5-18(b)所示。



(a) 选择数字0　　(b) 选择数字3

图 5-18　展示效果(9)

3. Stack

Stack 是堆叠容器，子组件按照顺序依次入栈，后一个子组件覆盖前一个子组件。

1）演示案例示例代码

index.ets 文件的代码如下：

```
//第 5 章/容器组件 Stack 示例 index.ets 代码
@Entry
@Component
struct StackExample {
  build() {
    //堆叠容器
    Stack({ alignContent: Alignment.Bottom }) {
      Text('First child, show in bottom')
        .width('90 % ').height('100 % ')
        .fontSize(28)
        .backgroundColor(0xd2cab3)
        .align(Alignment.Top)
      Text('Second child, show in top')
        .width('70 % ').height('60 % ')
        .fontSize(22)
        .fontColor(0xfefefe)
        .backgroundColor("red")
        .align(Alignment.Top)
    }.width('100 % ').height(150).margin({ top: 50 })
  }
}
```

2）演示案例效果展示

Stack 堆叠容器演示案例效果，如图 5-19 所示。

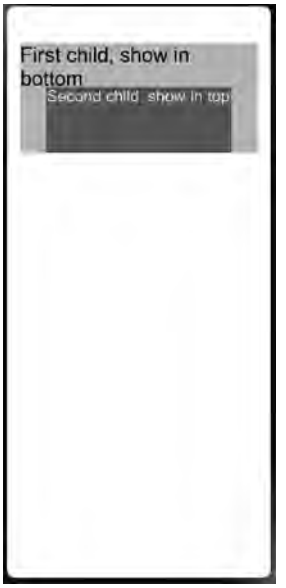


图 5-19　展示效果(10)

5.6.6 媒体组件

媒体组件名称、相关说明与接口，见表 5-47。

表 5-47 媒体组件

名称	说 明	接 口
Video	视频播放组件	Video(value：{src?：string，currentProgressRate?：number|string，previewUri?：string，controller?：VideoController})

5.6.7 绘制组件

绘制组件名称类别与相关说明，见表 5-48。

表 5-48 绘制组件

名称	说 明
Circle	圆形绘制组件
Ellipse	椭圆绘制组件
Line	直线绘制组件
Polyline	折线绘制组件
Polygon	多边形绘制组件
Path	路径绘制组件
Rect	矩形绘制组件
Shape	绘制组件的父组件，父组件中会描述所有绘制组件均支持的通用属性。①绘制组件使用 Shape 作为父组件，实现类似 SVG 的效果；②绘制组件单独使用，用于在页面上绘制指定的图形

下面我们通过几个案例练习 eTS 绘制组件的使用。

1. Circle

Circle 是圆形绘制组件，用于应用开发中圆形的实现，具体开发案例如下。

1）演示案例示例代码

index.ets 文件的代码如下：

```
//第 5 章/绘制组件 Circle 示例 index.ets 代码
@Entry
@Component
struct CircleExample {
  build() {
    Flex({ justifyContent: FlexAlign.SpaceAround }) {
      //绘制一个直径为 150 的圆
      Circle({ width: 150, height: 150 ,})
      //绘制一个直径为 150 的圆
      Circle()
```

```
        .width(150)
        .height(150)
    }.width('100%').margin({ top: 5 })
  }
}
```

2）演示案例效果展示

Circle 圆形绘制组件演示案例效果，如图 5-20 所示。

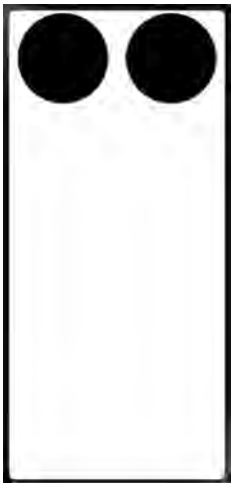


图 5-20　展示效果(11)

2. Polyline

Polyline 是多边形绘制组件，用于应用中多边图形的实现，具体开发案例如下。

1）演示案例示例代码

index.ets 文件的代码如下：

```
//第 5 章/绘制组件 Polyline 示例 index.ets 代码
@Entry
@Component
struct PolylineExample {
  build() {
    Column({ space: 5 }) {
      Flex({ justifyContent: FlexAlign.SpaceAround ,alignItems:ItemAlign.Center,}) {
        //在 100 * 100 的矩形框中绘制一段折线,起点(0, 0),经过(20,60),到达终点(100, 100)
        //绘制折线
        Polyline({ width: 100, height: 100 }).points([[0, 0], [20, 60], [100, 100]])
        //在 100 * 100 的矩形框中绘制一段折线,起点(0, 0),经过(0,100),到达终点(100, 100)
        Polyline().width(100).height(100).points([[0, 0], [0, 100], [100, 100]])
      }.width('100%')
```

```
        .height(500)
    }.margin({ top: 5 })
  }
}
```

2）演示案例效果展示

Polyline 多边形绘制组件演示案例效果，如图 5-21 所示。

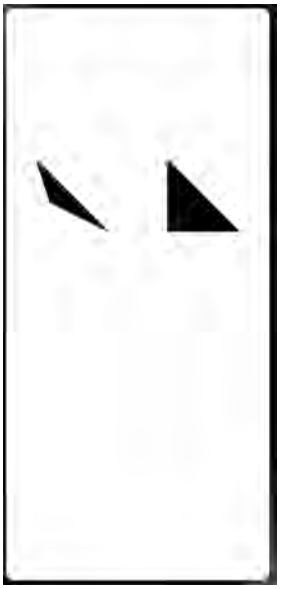


图 5-21　展示效果(12)

3. Rect

Rect 矩形绘制组件，用于应用开发中，矩形的实现，具体开发案例如下。

1）演示案例示例代码

index.ets 文件的代码如下：

```
//第 5 章/绘制组件 Rect 示例 index.ets 代码
@Entry
@Component
struct RectExample {
  build() {
    Column({ space: 5 }) {
      Text('normal').fontSize(24).fontColor(0x333333).width('90 % ')
      //绘制 90 %  * 50 矩形
      Rect({ width: '90 % ', height: 50 })
      //绘制 90 %  * 50 矩形
```

```
        Rect().width('90%').height(50)

        Text('with rounded corners').fontSize(24).fontColor(0x333333).width('90%')
        //绘制 90% * 50 矩形, 圆角宽和高均为 20
        Rect({ width: '90%', height: 50 }).radiusHeight(20).radiusWidth(20)
        //绘制 90% * 50 矩形, 圆角宽和高均为 20
        Rect({ width: '90%', height: 50 }).radius(20)
      }.width('100%').margin({ top: 5 })
    }
  }
```

2) 演示案例效果展示

Rect 矩形绘制组件演示案例效果,如图 5-22 所示。

图 5-22 展示效果(13)