

第 3 章



Java UI

应用开发中,界面设计和 UI 布局是必不可少的。任何一个应用都需要在屏幕上显示用户界面,包含用户可查看并与之交互的所有内容。在 HarmonyOS 中,可通过 Java 和 JS 两种方式布局 UI。本章先讲述 Java UI 布局,在第 5 章将讲述 JS UI 布局。

在学习使用 Java 对 UI 进行设计之前,首先需要理解一下 HarmonyOS 的 UI 逻辑。依据功能和特点,可将 HarmonyOS 的 UI 组件分为两类:单体组件 Component 和可用于装载其他组件的容器组件 ComponentContainer。

UI 组件根据一定的层级结构进行组合形成整体的 UI。单体组件在未被添加到容器组件中时,既无法显示也无法交互,因此一个用户界面至少包含一个容器组件。此外,容器组件中不但可以添加单体组件,也可以添加其他的容器组件,这样就可以形成丰富多样的 UI 样式。HarmonyOS 的 UI 视图树范例如图 3.1 所示。

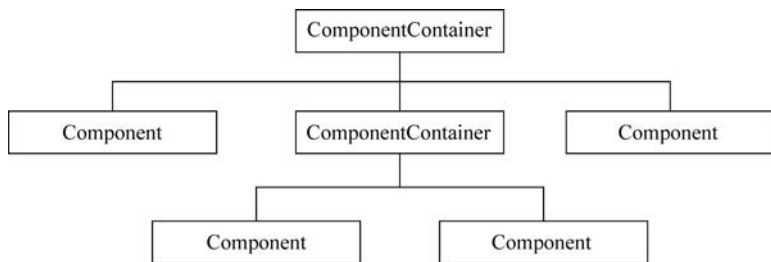


图 3.1 HarmonyOS 的 UI 视图树范例

单体组件和容器组件以树状的层级结构进行组织,这样的布局被称为视图树。视图树的特点是仅有一个根视图,其他视图有且仅有一个父节点,视图之间的关系受到父节点的规则约束。

3.1 Java UI 单体组件

本章主要讲解使用 Java 语言设置单体 UI 组件。

HarmonyOS 常用的单体 UI 组件全部继承自 Component 类,Component 类包含了对

UI 组件的全部常用方法,例如创建、更新、缩放、旋转及设置各类事件监听器。其他的 UI 组件,如 Text、Button 等,都是在 Component 类的基础上添加了对应的功能实现的。以下是常用单体组件的开发流程。

3.1.1 Text 组件

Text 是用来显示字符串的组件,是应用开发中最基础的组件,在界面上显示为一块文本区域。下面学习 Text 的用法。

1. Java 代码创建 Text 组件

首先按照第 1 章的介绍,创建一个 Phone 设备的 Java 模板,用来学习使用代码创建 UI 布局。在新建项目中的界面左侧找到 Project 项目栏,双击打开 entry→src→main→项目 Package 名称(本书中为 com.huawei.myapplication)→Slice 中的 MainAbilitySlice.java 文件,如图 3.2 所示。

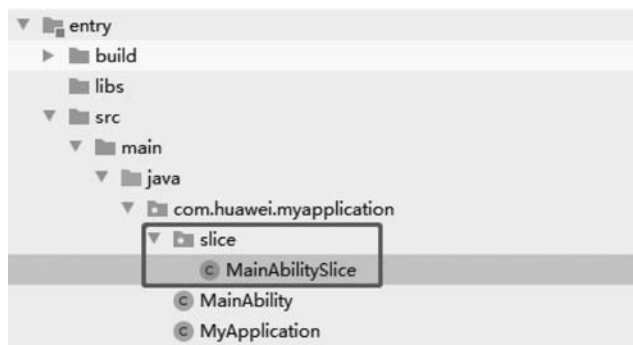


图 3.2 MainAbilitySlice 文件路径

在打开后的 Java 文件中找到 onStart() 方法,修改其中的内容,代码如下:

```
//MainAbilitySlice.java
public void onStart(Intent intent) {
    super.onStart(intent);

    //容器
    DirectionalLayout myLayout = new DirectionalLayout(this);
    myLayout.setWidth(DirectionalLayout.LayoutParams.MATCH_PARENT);
    myLayout.setHeight(DirectionalLayout.LayoutParams.MATCH_PARENT);
    myLayout.setAlignment(Layout.Alignment.HORIZONTAL_CENTER);
    myLayout.setPadding(32, 32, 32, 32);

    //Text 组件
    Text text = new Text(this);
    text.setWidth(DirectionalLayout.LayoutParams.MATCH_CONTENT);
    text.setHeight(DirectionalLayout.LayoutParams.MATCH_CONTENT);
```

```

text.setText("My name is Text.");
text.setTextSize(24, Text.TextSizeType.VP);
myLayout.addComponent(text);

super.setUIContent(myLayout);
}

```

下面对代码的具体内容进行讲解。

第一个代码块对容器组件进行了设置,页面容器组件的相关说明会在下一节中进行。第二个代码块设置并添加了一个 Text 组件,其中前 5 行代码创建并设置了一个 Text 组件,每一句的具体意义如下:

第 1 行:实例化一个 Text 类的实例,实例名为 text。

第 2 和第 3 行:为 text 设置了宽和高属性,这样可以让组件的大小自动适配其内容(文本)的大小。

第 4 行:为 text 添加要显示的文字信息 My name is Text。

第 5 行:为 text 的文本字体设置大小 24vp。

至此,text 的属性就设置完毕了,为了使 text 能够在手机屏幕上显示,还需要执行 super.setUIContent(myLayout),将创建好的 Text 组件实例 text 放入容器组件中,相关知识在后面讲到容器组件时会具体说明。运行上述代码,效果如图 3.3 所示。



图 3.3 Text 组件的运行效果

从图 3.3 可以看出,页面顶部出现了一条黑色的文本信息,显示 My name is Text,说明上述设置均已生效。

还可以使用 Java 代码修改或添加一些属性,例如,使用 setTextColor()方法可以修改 text 的字体颜色,使用 setFont()方法可以修改 text 的字体。在 Text 组件代码块中添加代码如下:

```

//MainAbilitySlice.java 中的 Text 组件
Text text = new Text(this);
text.setWidth(DirectionalLayout.LayoutConfig.MATCH_CONTENT);
text.setHeight(DirectionalLayout.LayoutConfig.MATCH_CONTENT);
text.setText("My name is Text.");
text.setTextSize(24, Text.TextSizeType.VP);
text.setTextColor(Color.BLUE);
text.setFont(Font.DEFAULT_BOLD);
myLayout.addComponent(text);

```

上述代码将 text 的文本设置为蓝色,字体加粗。重新运行程序,运行结果如图 3.4 所示。

2. xml 创建 Text 组件

在实际应用中,为了代码的简洁美观,通常会使用 xml 进行布局,这里学习通过 xml 进行 UI 布局,实现相同的效果。在界面左侧的 Project 项目栏选择 entry→src→main→resources→base→layout,打开 ability_main.xml,如图 3.5 所示。



图 3.4 Text 组件的属性设置效果

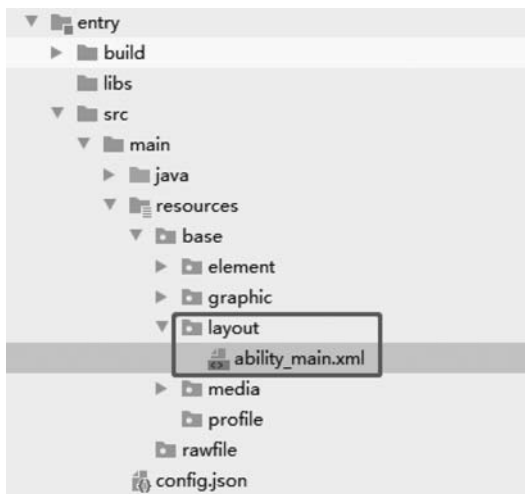


图 3.5 xml 布局目录结构

打开 ability_main.xml 文件,对布局 and 组件进行描述,代码如下:

```
<!-- ability_main.xml -->
<?xml version = "1.0" encoding = "utf - 8"?>
<DirectionalLayout
    xmlns:ohos = "http://schemas.huawei.com/res/ohos"
    ohos:width = "match_parent"
    ohos:height = "match_parent"
    ohos:orientation = "vertical"
    ohos:padding = "32">

    <Text
        ohos:id = " $ + id:text"
        ohos:width = "match_content"
        ohos:height = "match_content"
        ohos:layout_alignment = "horizontal_center"
        ohos:text = "My name is Text."
        ohos:text_size = "24vp"/>

</DirectionalLayout>
```

其中,最外层的 DirectionalLayout 指页面的整体竖向布局。在 Text 中,使用 ohos:id 为当前控件定义一个唯一的标识 id,布局中的组件通常都需要设置独立的 id,以便在程序中

查找该组件。若布局中有不同的组件设置了相同的 id, 则通过 id 查找会返回查找到的第一个组件, 因此尽可能保证 id 的唯一性。使用 `ohos:width` 和 `ohos:height` 为控件指定了宽度和高度, 其属性值 `match_parent` 表示组件大小将扩展为父容器允许的最大值, 占据父组件方向上的剩余大小, 即由父组件决定当前控件的大小。属性值 `match_content` 表示组件大小与它的内容占据的大小范围相适应, 即由控件内容决定当前控件的大小。除此之外也可以通过具体的数值设置宽和高, 例如 24(以像素为单位)或 24vp(以屏幕相对像素为单位)。使用 `ohos:layout_alignment` 指定文字的对齐方式, 其中 `horizontal_center` 指水平方向居中。通过 `ohos:text` 设置 Text 的具体内容。

随后, 在 `MainAbilitySlice.java` 中, 注释掉通过 Java 创建 Text 组件的代码, 并通过 `setUIContent()` 加载该 xml 布局。修改 `MainAbilitySlice.java` 中的代码如下:

```
//MainAbilitySlice.java
public class MainAbilitySlice extends AbilitySlice {

    @Override
    public void onStart(Intent intent) {
        super.onStart(intent);
        ...
        //注释通过 Java 创建 Text 组件代码
        super.setUIContent(ResourceTable.Layout_ability_main); //加载 xml 布局
    }
}
```

现在运行程序, 效果如图 3.6 所示。

在 xml 中也可以对 Text 设置字体颜色及字重。在 `ability_main.xml` 中, 增加属性代码如下:

```
<!-- ability_main.xml -->
<?xml version="1.0" encoding="utf-8"?>
<DirectionalLayout
    xmlns:ohos="http://schemas.huawei.com/res/ohos"
    ohos:width="match_parent"
    ohos:height="match_parent"
    ohos:orientation="vertical"
    ohos:padding="32">

    <Text
        ohos:id="@+id:text"
        ohos:width="match_content"
        ohos:height="match_content"
        ohos:layout_alignment="horizontal_center"
        ohos:text="My name is Text."
        ohos:text_size="24vp"
```

```
ohos:text_color = "blue"  
ohos:text_weight = "700"/>  
  
</DirectionalLayout>
```

通过 `ohos:text_color` 属性可以设置文本的颜色,通过 `ohos:text_weight` 属性可以设置文本的字重。重新运行后效果如图 3.7 所示,可见 `text` 文本变为蓝色,且字体加粗。



图 3.6 xml 创建 Text 组件的运行效果



图 3.7 xml 中设置 Text 颜色及字重运行效果

除此之外,Text 还有很多其他的属性,在开发过程中根据需求查阅相关文档即可。

3.1.2 Button 组件

Button(按钮)是一种常见的与用户进行交互的组件,单击可以触发对应的操作,可以由图标和文本共同组成。

1. Java 代码创建 Button 组件

同样地,还是先尝试使用 Java 代码来添加一个 Button 组件。打开 `MainAbilitySlice.java`,在 `onStart()` 函数中添加如下代码:

```
//MainAbilitySlice.java 中添加 Button 组件  
Button button = new Button(this);  
button.setWidth(DirectionalLayout.LayoutConfig.MATCH_PARENT);  
button.setHeight(DirectionalLayout.LayoutConfig.MATCH_CONTENT);  
button.setText("button");  
button.setTextSize(28, Text.TextSizeType.VP);  
ShapeElement backgroundElement = new ShapeElement();  
backgroundElement.setRgbColor(new RgbColor(0xDC, 0xDC, 0xDC));
```

```
button.setBackground(backgroundElement);
myLayout.addComponent(button);
```

值得注意的是,此时如果直接运行程序会发现 UI 界面没有变化,这是因为在 3.1.1 节最后使用了 `setUIContent()` 加载了 xml 布局 `ability_main.xml`,而修改的代码是创建的布局 `myLayout`,这是两套互不干涉的布局,所以如果想看到刚刚在代码布局中添加的 `button`,需要将 `setUIContent()` 函数中的变量改为 `myLayout`,然后运行程序,这样就可以看到的 `text` 下方多出了 `button`。`Button` 组件也有丰富的自定义属性,大家同样可以根据自己的需求来查阅相关文档。

2. 使用 xml 创建 Button 组件

同样也可以使用 xml 来添加 `Button` 组件,在刚才编写的 `ability_main.xml` 文件中增加 `Button`,代码如下:

```
//ability_main.xml
<?xml version="1.0" encoding="utf-8"?>
<DirectionalLayout
    xmlns:ohos="http://schemas.huawei.com/res/ohos"
    ohos:width="match_parent"
    ohos:height="match_parent"
    ohos:orientation="vertical"
    ohos:padding="32">

    ...

    <Button
        ohos:id="$ + id:button"
        ohos:width="match_parent"
        ohos:height="match_content"
        ohos:text="button"
        ohos:text_size="28vp"
        ohos:background_element="$ graphic:buttonelement"/>

</DirectionalLayout>
```

`Button` 可配置属性与 `Text` 差不多一致。其中, `ohos:background_element` 为对 `Button` 引用背景色,属性值 `$ graphic: buttonelement` 指为 `Button` 创建的背景色文件。在左侧 `Project` 窗口中,打开 `entry`→`src`→`main`→`resources`→`base`→`graphic` 文件夹,右击选择 `New`→`File`,命名为 `buttonelement.xml`,并在 `buttonelement.xml` 中定义 `Button` 的背景,代码如下:

```
<!-- buttonelement.xml -->
<?xml version="1.0" encoding="utf-8"?>
```

```

< shape xmlns:ohos = "http://schemas.huawei.com/res/ohos"
      ohos:shape = "rectangle">
< solid
      ohos:color = "#DCDCDC"/>
</shape>

```

完成上述添加 Button 的操作后,在 MainAbilitySlice.java 中,注释通过 Java 添加布局的代码,并通过 super.setUIContent(ResourceTable.Layout_ability_main) 引用 xml 布局文件,运行后界面如图 3.8 所示。

Button 组件的一个重要功能是当用户单击 Button 组件时,会执行相应的操作。这就需要让系统为 Button 组件设置一个单击事件监听器,监听器会在一个新的线程中不断地检测这个 Button 组件的单击事件。当用户单击按钮时,Button 对象会收到一个单击事件,开发者可以自定义响应单击事件的方法。

例如,通过创建一个 Component.ClickedListener 对象,然后通过调用 setClickedListener 将其分配给 Button 对象,在 onClick() 方法中可以添加单击 Button 后的事件逻辑。通过上述方法实现对 Button 的事件监听,示例代码如下:

```

//Button 单击事件代码示例
public class MainAbilitySlice extends AbilitySlice {

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

        //从定义的 xml 中获取 Button 对象
        Button button = (Button) findComponentById(ResourceTable.Id_button);
        if(button != null) {
            //为按钮设置单击事件监听器
            button.setClickedListener(new Component.ClickedListener() {
                public void onClick(Component v) {
                    //此处添加单击按钮后的事件处理逻辑
                }
            });
        }
    }
}

```



图 3.8 xml 中创建 Button 组件运行效果

下面通过一个实例展示一下按钮的功能。打开 MainAbilitySlice.java,在 onStart() 函

数中添加如下代码：

```
//MainAbilitySlice.java
super.onStart(intent);
super.setUIContent(ResourceTable.Layout_ability_main);
Text text = (Text) findComponentById(ResourceTable.Id_text);
Button button = (Button) findComponentById(ResourceTable.Id_button);
if(button != null && text!= null){
    button.setClickListener(new Component.ClickedListener() {
        @Override
        public void onClick(Component component) {
            text.setTextColor(Color.RED);
            text.invalidate();
        }
    });
}
```

在这段代码中首先从 xml 中取出了 text 和 button 两个实例,然后为 button 设置了单击事件监听器,并将修改 text 的文字颜色作为单击响应事件。再次运行程序后,单击 button,会看到 text 中的文本变为了红色,运行效果如图 3.9 所示。



图 3.9 使用 Button 监听逻辑对 Text 颜色进行更改,左图为修改前,右图为修改后

3.1.3 Image 组件

Image 是用于在屏幕上展示图像资源的组件,是软件开发中非常常用的 UI 组件。接下来讨论 Image 的具体使用方法。

Image 可以通过 setPixelMap(PixelMap)展示 PixelMap 类组件,或者通过 setImageElement(Element)展示 Element 组件,也可以通过 setImageAndDecodeBounds(int)显示资源文件里的图片。

首先将图片资源 picture.jpg 放入 entry→src→main→resources→base→media 文件夹中,

修改 MainAbilitySlice.java 文件,初始化一个 DirectionalLayout 来承载 Image,代码如下:

```
//MainAbilitySlice.java
public class MainAbilitySlice extends AbilitySlice {
    private DirectionalLayout myLayout = new DirectionalLayout(this);
    private DirectionalLayout.LayoutConfig layoutConfig = new DirectionalLayout.LayoutConfig
(ComponentContainer.LayoutConfig.MATCH_PARENT, ComponentContainer.LayoutConfig.MATCH_PARENT);
    @Override
    public void onStart(Intent intent) {
        super.onStart(intent);
    }
}
```

随后在 onStart()方法中,设置 LayoutConfig 的属性,并使用 setImageAndDecodeBounds(int)方法加载 resources 资源文件夹中 media 文件中的图片资源,代码如下:

```
//MainAbilitySlice.java 中加载图片资源
@Override
public void onStart(Intent intent) {
    super.onStart(intent);
    myLayout.setLayoutConfig(layoutConfig);
    layoutConfig.alignment = LayoutAlignment.HORIZONTAL_CENTER;
    layoutConfig.width = ComponentContainer.LayoutConfig.MATCH_CONTENT;
    layoutConfig.height = ComponentContainer.LayoutConfig.MATCH_CONTENT;
    Image image = new Image(this);
    image.setImageAndDecodeBounds(ResourceTable.Media_picture);
    image.setLayoutConfig(layoutConfig);
    myLayout.addComponent(image);
    super.setUIContent(myLayout);
}
```

运行上述代码,展示效果如图 3.10 所示。



图 3.10 Image 组件的效果展示

Image 还可以通过 `setPixelMap(PixelMap)` 展示 `PixelMap` 类组件,修改 `onStart()` 中的代码如下:

```
//MainAbilitySlice.java
@Override
public void onStart(Intent intent) {
    super.onStart(intent);
    myLayout.setLayoutConfig(layoutConfig);
    layoutConfig.alignment = LayoutAlignment.HORIZONTAL_CENTER;
    layoutConfig.width = ComponentContainer.LayoutConfig.MATCH_CONTENT;
    layoutConfig.height = ComponentContainer.LayoutConfig.MATCH_CONTENT;
    ResourceManager resourceManager = this.getResourceManager();
    Resource resource = null;
    try {
        resource = resourceManager.getResource(ResourceTable.Media_picture);
    } catch (IOException e) {
        e.printStackTrace();
    } catch (NotExistException e) {
        e.printStackTrace();
    }
    ImageSource imageSource = ImageSource.create(resource, new ImageSource.SourceOptions());
    PixelMap pixelMap = imageSource.createPixelmap(new ImageSource.DecodingOptions());
    Image image = new Image(this);
    image.setPixelMap(pixelMap);
    image.setLayoutConfig(layoutConfig);
    myLayout.addComponent(image);
    super.setUIContent(myLayout);
}
```

运行上述代码,可以得到和图 3.10 相同的运行效果。

还可以通过 `setImageElement(Element)` 令 `Image` 组件展示 `Element` 组件。构造一个蓝色矩形的 `element1`,然后把这个 `element1` 传给 `Image` 组件并显示。修改 `onStart()` 中代码如下:

```
//MainAbilitySlice.java
@Override
public void onStart(Intent intent) {
    super.onStart(intent);
    myLayout.setLayoutConfig(layoutConfig);
    layoutConfig.alignment = LayoutAlignment.HORIZONTAL_CENTER;
    layoutConfig.width = 1000;
    layoutConfig.height = 1000;

    ShapeElement element1 = new ShapeElement();
    element1.setShape(ShapeElement.RECTANGLE);
```

```
element1.setRgbColor(new RgbColor(0, 0, 255));

Image image = new Image(this);
image.setImageElement(element1);
image.setLayoutConfig(layoutConfig);
myLayout.addComponent(image);
super.setUIContent(myLayout);
}
```

运行上述代码,展示效果如图 3.11 所示。

还可以根据需求,通过 `setBounds()` 方法设置 `element` 的位置和大小,如新增 `element2`,设置一定的位置,将颜色设置为红色,并通过 `setBackground()` 方法将 `element1` 设置为背景,在 `onStart()` 方法中调整代码如下:

```
ShapeElement element2 = new ShapeElement();
element2.setShape(ShapeElement.RECTANGLE);
element2.setRgbColor(new RgbColor(255, 0, 0));
element2.setBounds(300,100,800,1000);
Image image = new Image(this);
image.setBackground(element1);

image.setImageElement(element2);
```

运行上述代码,效果如图 3.12 所示。

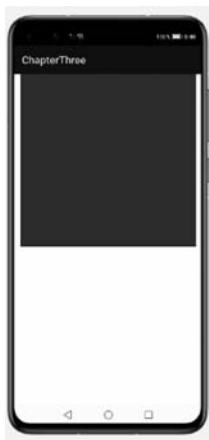


图 3.11 设置 element 效果图

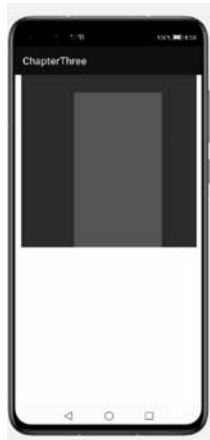


图 3.12 调整 element 大小

`image` 和其中放的图片内容都是有大小的,如果 `image` 的大小和内部图片的大小不一致,则需要使用 `setScalemode()` 方式来调整图片大小。接下来,详细介绍一下 `scalemode` 的使用方法。

scalemode 有 6 种参数,分别演示一下效果。修改 onStart()方法中 Image 部分代码如下:

```
Image image1 = new Image(this);
image1.setImageAndDecodeBounds(ResourceTable.Media_picture);
Image image2 = new Image(this);
image2.setImageAndDecodeBounds(ResourceTable.Media_picture);
image2.setHeight(300);
image2.setWidth(300);
image2.setBackground(element1);

myLayout.addComponent(image1);
myLayout.addComponent(image2);
```

运行上述代码,会得到一张显示效果不全的图片,运行效果如图 3.13 所示。其中, image1 是原图,组件大小和图片大小一致。image2 把图片组件缩小到 300×300 ,这样就导致图片显示不全。

创建 Image 实例 image3,同样将大小设置为 300×300 ,并将 image3 的 scalemode 改变为 INSIDE,代码如下:

```
Image image3 = new Image(this);
image3.setImageAndDecodeBounds(ResourceTable.Media_picture);
image3.setBackground(element1);
image3.setHeight(300);
image3.setWidth(300);
image3.setScaleMode(Image.ScaleMode.INSIDE);
myLayout.addComponent(image3);
```

运行代码效果如图 3.14 所示,其中第 3 张是 INSIDE 的效果,为了直观地看到运行效果,可以将组件的背景用蓝色充满,可以看出 image3 的 scalemode 设置为 INSIDE 后,会根据比例对图像进行缩小,使得图像与组件大小相同或更小,并在中心显示图像内容。image3 大小为 300×300 ,但图片被按比例缩小在了 element1 中。



图 3.13 Image 显示不全效果图



图 3.14 image3 按比例缩放效果图

若将 image3 的 scalemode 改变为 STRETCH, STRETCH 不使用任何比例对图像进行缩放。修改代码如下:

```
image3.setScaleMode(Image.ScaleMode.STRETCH);
```

运行后效果如图 3.15 所示,可以清楚地看到,图片原本的比例被破坏,被拉长后充满了 300×300 的组件。



图 3.15 image3 设置为 STRETCH 效果图

ZOOM_CENTER 可以实现放大效果,将 Image 组件放大到 900×900,使得组件大于图片大小,原图片与组件 image2 间的空白位置用蓝色填充, scalemode 设置为 ZOOM_CENTER 后的图片与组件 image3 间的空白位置用青色填充,代码如下:

```
ShapeElement element1 = new ShapeElement();
element1.setShape(ShapeElement.RECTANGLE);
element1.setRgbColor(new RgbColor(0, 0, 255));

ShapeElement element2 = new ShapeElement();
element2.setShape(ShapeElement.RECTANGLE);
element2.setRgbColor(new RgbColor(0, 255, 255));

Image image1 = new Image(this);
image1.setImageAndDecodeBounds(ResourceTable.Media_picture);
image1.setLayoutConfig(layoutConfig);
image1.setHeight(900);
image1.setWidth(900);
image1.setBackground(element1);

Image image2 = new Image(this);
image2.setImageAndDecodeBounds(ResourceTable.Media_picture);
```

```
image2.setLayoutConfig(layoutConfig);
image2.setHeight(900);
image2.setWidth(900);
image2.setBackground(element2);
image2.setScaleMode(Image.ScaleMode.ZOOM_CENTER);
```

运行效果如图 3.16 所示。可以清楚地看到第一张为原图片,第二张图片根据比例将图片进行了放大,使得图片宽度与组件宽度相同,并在中心显示图片内容。

随后分别设置 image2 的 scalemode 为 ZOOM_START 和 ZOOM_END,可以看到效果如图 3.17 所示。ZOOM_START 可使图片放大并在左上角显示图片,ZOOM_END 可使图片放大并在右下角显示图片。



图 3.16 图片中心放大效果



图 3.17 Image 放大并分别显示在左上角、右下角

3.2 Java UI 容器组件

Java UI 框架提供了一些具有标准布局功能的容器,均继承自 ComponentContainer 类,一般以 Layout 结尾,如 DirectionalLayout、DependentLayout 等(也有例外,例如 ListContainer 类也是一个布局类)。

由图 3.1 视图树可知,完整的用户界面是一个容器组件,用户界面中的一部分也可以是一个容器组件。容器组件中可以容纳单个组件与其他容器组件。

HarmonyOS 为开发者分别提供了在 Java 代码和 xml 格式的文件中声明布局的方法。下面将分别使用代码和 XML 文件来开发 HarmonyOS 的常用布局。



13min

3.2.1 线性布局 DirectionalLayout

DirectionalLayout 是 Java UI 中的一种重要组件布局,用于将一组组件(Component 或 ComponentContainer)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。3.1.1 和 3.1.2 节就是将 Text 和 Button 等组件放在了 DirectionalLayout 中,实现了 Text 与 Button 自上而下有序排列的效果。

DirectionalLayout 的排列方向(orientation)分为水平(horizontal)和竖直(vertical)方向。使用 orientation 设置布局内组件的排列方式,默认为垂直排列。示意图如图 3.18 所示。

1. 代码创建 DirectionalLayout 组件

这里设置了 3 个单体组件 Button 和一个线性容器组件,并实现 Button 组件在线性容器内自上而下竖直排列。在 MainAbilitySlice.java 中修改 onStart() 函数,代码如下:

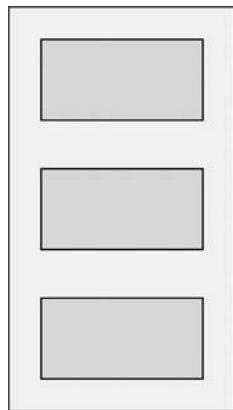


图 3.18 DirectionalLayout 竖直排列示意图

```
//MainAbilitySlice.java
public void onStart(Intent intent) {
    super.onStart(intent);
    this.setDisplayOrientation(AbilityInfo.DisplayOrientation.PORTRAIT);

    //创建容器组件 DirectionalLayout 实例
    DirectionalLayout myLayout = new DirectionalLayout(this);
    //设置 DirectionalLayout 对内部组件的排列方式并设置为竖直排列
    myLayout.setOrientation(Component.VERTICAL);
    //设置 DirectionalLayout 内部的组件都排列在屏幕水平的中央位置
    myLayout.setAlignment(LayoutAlignment.HORIZONTAL_CENTER);
    //容器组件 DirectionalLayout 的 LayoutConfig
    DirectionalLayout.LayoutConfig layoutConfig = new DirectionalLayout.LayoutConfig
        (DirectionalLayout.LayoutConfig.MATCH_PARENT, DirectionalLayout.LayoutConfig.MATCH_PARENT);
    myLayout.setLayoutConfig(layoutConfig);

    //创建 3 个 Button 实例
    Button button1 = new Button(this);
    Button button2 = new Button(this);
    Button button3 = new Button(this);
    //单体组件 Button 的 LayoutConfig

    LayoutConfig buttonConfig = new LayoutConfig(
        DirectionalLayout.LayoutConfig.MATCH_CONTENT,
        DirectionalLayout.LayoutConfig.MATCH_CONTENT);
    buttonConfig.setMargins(0,100,0,100);
    //将 LayoutConfig 应用到 3 个 Button 上
    button1.setLayoutConfig(buttonConfig);
    button2.setLayoutConfig(buttonConfig);
```

```

        button3.setLayoutConfig(buttonConfig);
        //创建 Button 的背景
        ShapeElement bottomElement = new ShapeElement();
        bottomElement.setRgbColor(new RgbColor(200,200,200));
        //将背景应用到 3 个 Button 上
        button1.setBackground(bottomElement);
        button2.setBackground(bottomElement);
        button3.setBackground(bottomElement);

        //设定 3 个 Button 内显示的文本
        button1.setText("Button 1");
        button2.setText("Button 2");
        button3.setText("Button 3");

        //设定 3 个 Button 内显示的文字大小
        button1.setTextSize(80, Text.TextSizeType.VP);
        button2.setTextSize(80, Text.TextSizeType.VP);
        button3.setTextSize(80, Text.TextSizeType.VP);

        //将 3 个 Button 加入 DirectionalLayout 中
        myLayout.addComponent(button1);
        myLayout.addComponent(button2);
        myLayout.addComponent(button3);

        super.setUIContent(myLayout);
    }

```

在这段代码中首先创建了一个 DirectionalLayout 容器组件实例,然后对它的属性进行了设置,其中比较重要的属性有如下 3 种。

Orientation: 代表了 DirectionalLayout 对其内部组件(子组件)的排列规则。VERTICAL 代表从上到下竖直排列, HORIZONTAL 则代表从左至右水平排列。

Alignment: 代表了 DirectionalLayout 内部组件(子组件)的排列位置。HORIZONTAL_CENTER 代表水平位置的正中,常用的还有左对齐 LEFT、右对齐 RIGHT 和竖直位置的正中 VERTICAL_CENTER。

LayoutConfig: 代表了对 DirectionalLayout 本身的一些设置,不过在这里只设置了宽和高为 MATCH_PARENT,这意味着这个 DirectionalLayout 的大小将充满它的上一层容器(父容器组件),在这里这个 DirectionalLayout 没有父容器组件,所以将充满整个屏幕。

尝试运行程序,可以看到 3 个 Button 组件在屏幕水平方向的正中央自上而下竖直排列,如图 3.19 所示。

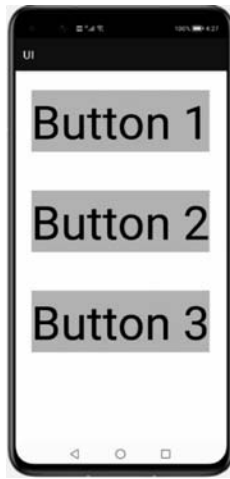


图 3.19 代码创建的 DirectionalLayout 效果展示

2. xml 创建 DirectionalLayout

现在尝试使用 xml 来创建 DirectionalLayout,以便了解更多功能。这里设置 3 个 Button,并将它们设置为垂直排列。首先在 layout 文件夹中新建布局文件 directional_layout.xml,在其中加入 3 个 Button,并将其最外层标签设置为线性布局 DirectionalLayout,代码如下:

```
<!-- directional_layout.xml -->
<?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">
    <Button
        ohos:width="100vp"
        ohos:height="50vp"
        ohos:bottom_margin="13vp"
        ohos:left_margin="13vp"
        ohos:background_element="$ graphic:buttonelement"
        ohos:text="Button 1"
        ohos:text_size="24vp"/>
    <Button
        ohos:width="100vp"
        ohos:height="50vp"
        ohos:bottom_margin="13vp"
        ohos:left_margin="13vp"
        ohos:background_element="$ graphic:buttonelement"
        ohos:text="Button 2"
        ohos:text_size="24vp"/>
    <Button
        ohos:width="100vp"
        ohos:height="50vp"
        ohos:bottom_margin="13vp"
        ohos:left_margin="13vp"
        ohos:background_element="$ graphic:buttonelement"
        ohos:text="Button 3"
        ohos:text_size="24vp"/>
</DirectionalLayout>
```

其中,ohos:orientation 的属性值设置为 vertical,即垂直排列。在 MainAbilitySlice 中修改代码,引入 directional_layout.xml 布局文件,代码如下:

```
//MainAbilitySlice.java
public class MainAbility extends Ability {

    @Override
```

```

public void onStart(Intent intent) {
    super.onStart(intent);
    super.setUIContent(ResourceTable.Layout_directional_layout);
}
}

```

运行效果如图 3.20 所示,可见 3 个 Button 为垂直排列。



图 3.20 3 个 Button 垂直排列

垂直排列默认为左对齐,组件可以通过 `ohos:layout_alignment` 控制自身在布局中的对齐方式。在垂直排列中,`layout_alignment` 的属性值 `left` 表示左对齐,`right` 表示右对齐,`horizontal_center` 表示水平方向居中,`center` 表示水平方向和垂直方向均居中。当属性值的对齐方式与排列方式方向一致时,对齐方式不会生效,如设置了水平方向的排列方式,则左对齐、右对齐将不会生效。如修改 `directional_layout.xml` 布局文件,代码如下:

```

<!-- directional_layout.xml -->
<?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">
    <Button
        ohos:width="100vp"
        ohos:height="50vp"
        ohos:layout_alignment="left"
        ... />
    <Button
        ohos:width="100vp"

```

```
        ohos:height = "50vp"
        ohos:layout_alignment = "horizontal_center"
        ... />
    <Button
        ohos:width = "100vp"
        ohos:height = "50vp"
        ohos:layout_alignment = "right"
        ... />
</DirectionalLayout>
```

运行后效果如图 3.21 所示。可见,3 个 Button 分别实现了左对齐、水平居中和右对齐。

在 DirectionalLayout 布局中,可以将 ohos:orientation 属性值改为 horizontal,这样 Button 则变为水平排列,修改代码如下:

```
<!-- directional_layout.xml -->
<?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">

    ...

</DirectionalLayout>
```

运行后效果如图 3.22 所示,可以看到 3 个 Button 变为水平排列。

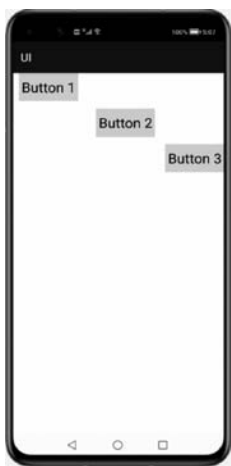


图 3.21 对齐方式效果示例



图 3.22 3 个 Button 水平排列

注意,DirectionalLayout 布局不能自动换行,子视图会按照设定的方向依次排列,若超过布局本身的大小,则超出布局大小的部分将不会被显示。修改 3 个 button 的宽度,代码如下:

```
<!-- directional_layout.xml -->
<?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">
    <Button
        ohos:width = "150vp"
        ohos:height = "50vp"
        ... />
    <Button
        ohos:width = "150vp"
        ohos:height = "50vp"
        ... />
    <Button
        ohos:width = "150vp"
        ohos:height = "50vp"
        ... />
</DirectionalLayout>
```

运行后界面效果如图 3.23 所示。

此布局包含了 3 个 Button,但是因为宽度超出了布局范围,所以界面上只完整显示了两个 Button,超出布局大小的视图部分无法正常显示,因此在开发布局中要注意避免这类问题的出现。

在 HarmonyOS 的 UI 组件中,布局之间也可以按层级关系互相组合,DirectionalLayout 布局和其他布局的组合,可以实现更加丰富的布局方式。具体逻辑可以参照后面章节的视图树。

3.2.2 相对布局 DependentLayout

DependentLayout 是 Java UI 里另一种常见布局,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进行布局。与 DirectionalLayout 相比,它拥有更多的排布方式。布局的示意图如图 3.24 所示。

DependentLayout 的布局方式,有相对同级其他组件和相对父组件两种布局方式。



图 3.23 DirectionalLayout 不可自动换行示例

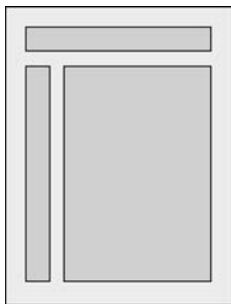


图 3.24 DependentLayout 布局示意图

1. 代码创建 DependentLayout

和 DirectionalLayout 举的例子类似, 在这里设置 3 个单体组件 Button 和一个 DependentLayout 容器组件, 并实现 Button 组件在 DependentLayout 容器内自上而下竖直排列。为了方便展示, 还是在 MainAbilitySlice.java 中修改 onStart() 函数, 代码如下:

```
//MainAbilitySlice.java
public void onStart(Intent intent) {
    super.onStart(intent);
    this.setDisplayOrientation(AbilityInfo.DisplayOrientation.PORTRAIT);

    //创建容器组件 DependentLayout 实例, 并为之设置 Id
    DependentLayout myLayout = new DependentLayout(this);
    int L1 = 0;
    myLayout.setId(L1);
    //容器组件 DependentLayout 的 LayoutConfig
    DependentLayout.LayoutConfig layoutConfig = new
    DependentLayout.LayoutConfig(
    DependentLayout.LayoutConfig.MATCH_PARENT,
    DependentLayout.LayoutConfig.MATCH_PARENT);
    myLayout.setLayoutConfig(layoutConfig);

    //创建 3 个 Button 实例, 并为之设置 Id
    Button button1 = new Button(this);
    int B1 = 1;
    button1.setId(B1);
    Button button2 = new Button(this);
    int B2 = 2;
    button2.setId(B2);
    Button button3 = new Button(this);
    int B3 = 3;
    button3.setId(B3);

    //单体组件 Button 的 LayoutConfig
    DependentLayout.LayoutConfig buttonConfig1 = new DependentLayout.LayoutConfig(
```

```

DependentLayout.LayoutConfig.MATCH_CONTENT,
DependentLayout.LayoutConfig.MATCH_CONTENT);
    buttonConfig1.addRule(DependentLayout.LayoutConfig.CENTER_IN_PARENT);
    buttonConfig1.setMargins(0,100,0,100);
    DependentLayout.LayoutConfig buttonConfig2 = new DependentLayout.LayoutConfig(
DependentLayout.LayoutConfig.MATCH_CONTENT,
DependentLayout.LayoutConfig.MATCH_CONTENT);
    buttonConfig2.addRule(DependentLayout.LayoutConfig.BELOW, B1);
    buttonConfig2.setMargins(0,100,0,100);
    DependentLayout.LayoutConfig buttonConfig3 = new DependentLayout.LayoutConfig(
DependentLayout.LayoutConfig.MATCH_CONTENT,
DependentLayout.LayoutConfig.MATCH_CONTENT);
    buttonConfig3.addRule(DependentLayout.LayoutConfig.ALIGN_PARENT_BOTTOM);
    buttonConfig3.addRule(DependentLayout.LayoutConfig.ALIGN_PARENT_RIGHT);
    buttonConfig3.setMargins(0,100,0,100);
    //将 LayoutConfig 应用到 3 个 Button 上
    button1.setLayoutConfig(buttonConfig1);
    button2.setLayoutConfig(buttonConfig2);
    button3.setLayoutConfig(buttonConfig3);
    //创建 Button 的背景
    ShapeElement bottomElement = new ShapeElement();
    bottomElement.setRgbColor(new RgbColor(200,200,200));
    //将背景应用到 3 个 Button 上
    button1.setBackground(bottomElement);
    button2.setBackground(bottomElement);
    button3.setBackground(bottomElement);

    //设定 3 个 Button 内显示的文本
    button1.setText("Button 1");
    button2.setText("Button 2");
    button3.setText("Button 3");

    //设定 3 个 Button 内显示的文字大小
    button1.setTextSize(50, Text.TextSizeType.VP);
    button2.setTextSize(50, Text.TextSizeType.VP);
    button3.setTextSize(50, Text.TextSizeType.VP);

    //将 3 个 Button 加入 DependentLayout 中
    myLayout.addComponent(button1);
    myLayout.addComponent(button2);
    myLayout.addComponent(button3);

    super.setUIContent(myLayout);

}

```

运行效果如图 3.25 所示。

可以发现,与之前 `DirectionalLayout` 例子不同的是为每个组件(包括容器组件和单体组件)都设置了一个 `Id`,这是因为在 `DependentLayout` 中命令需要利用 `Id` 来识别组件对象,进而实现组件之间的相对布局。

为了实现相对布局,此处为每个子组件(范例中的 `Button` 实例)添加了一个 `DependentLayout`. `LayoutConfig` 类实例,在对应的 `DependentLayout`. `LayoutConfig` 实例中,使用 `addRule()` 函数为这些组件添加想要的排列规则,具体的排列规则如下:

对于 `button1`,为其添加的规则为 `LayoutConfig.CENTER_IN_PARENT`,这个规则规定了 `button1` 将位于其父组件(范例中的 `DependentLayout` 实例)的正中间。

对于 `button2`,为其添加的规则为 `LayoutConfig.BELOW` 和 `B1`,这个规则规定了 `button2` 将位于 `Id` 为 `B1`(范例中的 `button1` 实例)的下方。值得注意的是,在效果图中可以看出 `button2` 并没有位于 `button1` 的正下方,而是下方的左侧。这是因为这个规则只规定了 `button2` 垂直方向的位置,而并没有规定水平方向的位置,所以水平方向的位置仍为系统的默认值,即左侧排列。这一类问题在使用时需要注意。

对于 `button3`,为其添加的规则为 `LayoutConfig.ALIGN_PARENT_BOTTOM` 和 `LayoutConfig.ALIGN_PARENT_RIGHT`,这两个规则分别规定了 `button3` 要位于其父组件的下方和右侧,所以从效果图中可以看出 `button3` 位于界面的右下方。这说明一个 `DependentLayout`. `LayoutConfig` 实例可以添加多个规则。如果规则之间相互“冲突”,可以尝试将 `button3` 添加的 2 个规则修改为 `LayoutConfig.ALIGN_PARENT_RIGHT` 和 `LayoutConfig.ALIGN_PARENT_LEFT`,运行后观察效果如何。

2. xml 创建 `DependentLayout`

相对于同级其他组件的位置,每个组件有 `above`(位于同级组件的上侧)、`below`(位于同级组件的下侧)、`start_of`(位于同级组件的起始侧)、`end_of`(位于同级组件的结束侧)、`left_of`(位于同级组件的左侧)、`right_of`(位于同级组件的右侧)6 种相对位置。这里通过实例直观体会一下。首先在 `layout` 文件夹中新建布局文件 `dependent_layout.xml`,代码如下:

```
<!-- dependent_layout.xml -->
<?xml version="1.0" encoding="utf-8"?>
<DependentLayout
    xmlns:ohos="http://schemas.huawei.com/res/ohos"
    ohos:width="match_parent"
    ohos:height="match_content">
```

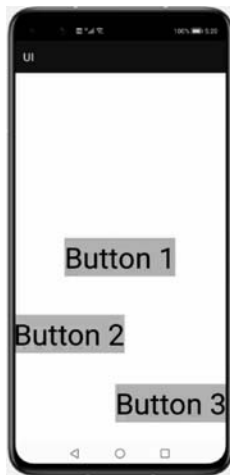


图 3.25 代码创建的 `DependentLayout` 效果展示

```

< Text
    ohos:id = " $ + id:text1"
    ohos:width = "match_content"
    ohos:height = "match_content"
    ohos:left_margin = "15vp"
    ohos:top_margin = "15vp"
    ohos:bottom_margin = "15vp"
    ohos:text = "text1"
    ohos:text_size = "20fp"
    ohos:background_element = " $ graphic:text_element"/>

< Text
    ohos:id = " $ + id:text2"
    ohos:width = "match_content"
    ohos:height = "match_content"
    ohos:left_margin = "15vp"
    ohos:top_margin = "15vp"
    ohos:bottom_margin = "15vp"
    ohos:text = "end_of text1"
    ohos:text_size = "20fp"
    ohos:background_element = " $ graphic:text_element"
    ohos:end_of = " $ id:text1"/>

< Text
    ohos:id = " $ + id:text3"
    ohos:width = "match_content"
    ohos:height = "match_content"
    ohos:left_margin = "15vp"
    ohos:text = "below text1"
    ohos:text_size = "20fp"
    ohos:background_element = " $ graphic:text_element"
    ohos:below = " $ id:text1"/>

</DependentLayout >

```

其中,ohos:end_of 属性可以让一个组件位于另一个组件的结束侧,属性值中需要指定相对控件的 id 引用,如 text2 中设置 ohos:end_of=" \$ id:text1",表示让该组件位于 text1 的结束侧。text3 中通过设置 ohos:below=" \$ id:text1",让 text3 位于 text1 的下方。其他相对位置同理。

在 graphic 文件夹中新建 text_element.xml 文件,代码如下:

```

<!-- text_element.xml -->
<?xml version = "1.0" encoding = "utf - 8"?>
< shape xmlns:ohos = "http://schemas.huawei.com/res/ohos"
    ohos:shape = "rectangle">
    < solid
        ohos:color = " # D5CF72"/>
    </shape>

```

在 MainAbility 中更改布局文件引用,代码如下:

```
super.setUIContent(ResourceTable.Layout_dependent_layout);
```

运行代码,效果如图 3.26 所示。

除了可以相对于同级其他组件位置进行定位,也可以相对父组件进行布局。相对于父组件的位置,每个组件有 align_parent_left(位于父组件的左侧)、align_parent_right(位于父组件的右侧)、align_parent_start(位于父组件的起始侧)、align_parent_end(位于父组件的结束侧)、align_parent_top(位于父组件的上侧)、align_parent_bottom(位于父组件的下侧)、center_in_parent(位于父组件的中间)7 种相对位置。基于以上布局,也可以形成左上角、左下角、右上角和右下角的布局。修改 dependent_layout.xml 中的代码如下:



图 3.26 相对同级组件位置布局

```
<!-- dependent_layout.xml -->
<?xml version="1.0" encoding="utf-8"?>
<DependentLayout
    xmlns:ohos="http://schemas.huawei.com/res/ohos"
    ohos:width="match_parent"
    ohos:height="match_parent">

    <Text
        ohos:id="$ + id:text1"
        ohos:width="match_content"
        ohos:height="match_content"
        ohos:text="left_top"
        ohos:text_size="20fp"
        ohos:background_element="$ graphic:text_element"
        ohos:align_parent_left="true"
        ohos:align_parent_top="true"/>

    <Text
        ohos:id="$ + id:text2"
        ohos:width="match_content"
        ohos:height="match_content"
        ohos:text="right_top"
        ohos:text_size="20fp"
        ohos:background_element="$ graphic:text_element"
        ohos:align_parent_right="true"
        ohos:align_parent_top="true"/>

    <Text
        ohos:id="$ + id:text3"
        ohos:width="match_content"
        ohos:height="match_content"
```

```

        ohos:text = "center"
        ohos:text_size = "20fp"
        ohos:background_element = " $ graphic:text_element"
        ohos:center_in_parent = "true"/>
    < Text
        ohos:id = " $ + id:text4"
        ohos:width = "match_content"
        ohos:height = "match_content"
        ohos:text = "bottom_center"
        ohos:text_size = "20fp"
        ohos:background_element = " $ graphic:text_element"
        ohos:align_parent_bottom = "true"
        ohos:center_in_parent = "true"/>

</DependentLayout>

```

上述代码中,通过 `ohos:align_parent_left` 和 `ohos:align_parent_top` 进行属性设置,使 `text1` 位于父组件的左上角,同理令 `text2` 位于父组件的右上角,`text3` 位于父组件居中,`text4` 位于父容器底部居中。运行后结果如图 3.27 所示。



图 3.27 相对父组件位置布局

3.2.3 绝对坐标布局 PositionLayout

绝对坐标布局容器 `PositionLayout` 也是比较常用的基础布局组件之一,它可以依据绝对坐标对其内部的组件进行布局,示意图如图 3.28 所示。具体来讲,当 `PositionLayout` 作为父组件时,其在屏幕上所占有的空间可视为一个二维直角坐标系,其子组件可以依据坐标值进行布局。

作为最直观的容器组件, `PositionLayout` 的使用方法与前两种 `Layout` 不同,它不需要使用 `LayoutConfig` 实现组件之

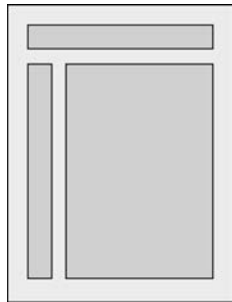


图 3.28 绝对坐标位置布局

间的布局,而仅需要为每个子组件直接设置坐标即可,实现 PositionLayout 的代码如下:

```
//MainAbilitySlice.java
public void onStart(Intent intent) {
    super.onStart(intent);
    this.setDisplayOrientation(AbilityInfo.DisplayOrientation.PORTRAIT);

    //创建容器组件 PositionLayout 实例,并为之设置 Id
    PositionLayout myLayout = new PositionLayout(this);
    //容器组件 PositionLayout 的 LayoutConfig
    PositionLayout.LayoutConfig layoutConfig = new
    PositionLayout.LayoutConfig(
    PositionLayout.LayoutConfig.MATCH_PARENT,
    PositionLayout.LayoutConfig.MATCH_PARENT);
    myLayout.setLayoutConfig(layoutConfig);

    //创建 3 个 Button 实例,并为之设置 Id
    Button button1 = new Button(this);
    Button button2 = new Button(this);
    Button button3 = new Button(this);

    //将 LayoutConfig 应用到 3 个 Button 上
    button1.setTop(100);
    button1.setLeft(100);
    button2.setTop(1000);
    button2.setLeft(500);
    button3.setTop(1500);
    button3.setRight(500);
    //创建 Button 的背景
    ShapeElement bottomElement = new ShapeElement();
    bottomElement.setRgbColor(new RgbColor(200,200,200));
    //将背景应用到 3 个 Button 上
    button1.setBackground(bottomElement);
    button2.setBackground(bottomElement);
    button3.setBackground(bottomElement);

    //设定 3 个 Button 内显示的文本
    button1.setText("Button 1");
    button2.setText("Button 2");
    button3.setText("Button 3");

    //设定 3 个 Button 内显示的文字大小
    button1.setTextSize(50, Text.TextSizeType.VP);
    button2.setTextSize(50, Text.TextSizeType.VP);
    button3.setTextSize(50, Text.TextSizeType.VP);

    //将 3 个 Button 加入 PositionLayout 中
```

```

myLayout.addComponent(button1);
myLayout.addComponent(button2);
myLayout.addComponent(button3);

super.setUIContent(myLayout);

}

```

在这里我们为 button1 和 button2 设置了 Left 和 Top 坐标,为 button3 设置了 Top 和 Right 坐标。运行的效果如图 3.29 所示。

可以看出,button1 和 button2 的左上角分别位于其父布局(范例中的 PositionLayout 实例)的(100,100)和(100,500)位置上,这是符合预期的,而 button3 却位于(0,1500)位置上,说明其 Right 坐标的设置是不能够改变组件位置的。经过一些测试发现对于 Bottom 坐标的设置也是不能够改变组件位置的。

由于 PositionLayout 是依据绝对坐标进行的布局,所以代码的灵活性和扩展性都相对较差。例如,PositionLayout 在适配各种分辨率的屏幕上是比较困难的,需要对每个组件的坐标值进行修改。在切换横屏竖屏时,若要保持布局的整齐性,每个坐标值都需要重新计算,这是比较麻烦的。



图 3.29 代码创建的 PositionLayout 效果展示

3.2.4 滚动菜单 ListContainer

手机屏幕空间有限,能显示的内容不多。可以借助 ListContainer 来显示更多的内容。ListContainer 允许用户通过上下滑动来将屏幕外的数据滚动到屏幕内,同时屏幕内原有的数据滚动出屏幕,从而显示更多的数据内容。但是 ListContainer 的使用和一般的组件使用不同,ListContainer 一定要和 Provider 适配器搭配使用,HarmonyOS 里 Provider 有 BaseItemProvider 和它的子类 RecyclerViewProvider。

在前面的章节中都是分别使用 Java 代码或者 xml 来创建布局容器(或单体组件),事实上还可以采用两者结合的手段进行创建,在这里将采用 xml 来创建 ListContainer,然后使用 Java 代码为其添加各种属性。首先在 resource 里定义一个带有 ListContainer 的布局,代码如下:

```

<!-- listcontainer.xml -->
<?xml version="1.0" encoding="utf-8"?>
<DirectionalLayout
    xmlns:ohos="http://schemas.huawei.com/res/ohos"

```

```

        ohos:width = "match_parent"
        ohos:height = "match_parent"
        ohos:orientation = "vertical">
<ListContainer
        ohos:id = "$ + id:list"
        ohos:width = "match_content"
        ohos:height = "match_content"
        ohos:margin = "50px"
    />

</DirectionalLayout>

```

然后需要设置一个适配器 provider 来为 ListContainer 提供内容,代码如下:

```

//MainAbilitySlice.java
public class SamplePagerAdapter extends BaseItemProvider {
    private List mList;
    public SamplePagerAdapter(List list) {
        mList = list;
    }
    @Override
    public int getCount() {
        return mList.size();
    }

    @Override
    public Object getItem(int i) {
        return mList.get(i);
    }

    @Override
    public long getItemId(int i) {
        return i;
    }

    @Override
    public Component getComponent(int i, Component component, ComponentContainer componentContainer) {
        Text title = new Text(getContext());
        title.setText((String)mList.get(i));
        title.setTextSize(200);
        title.setTextAlignment(TextAlignment.CENTER);
        title.setLayoutConfig(new StackLayout.LayoutConfig(
            StackLayout.LayoutConfig.MATCH_CONTENT,
            StackLayout.LayoutConfig.MATCH_CONTENT));
    }
}

```

```

        title.setTextColor(Color.RED);
        ShapeElement shapeElement = new ShapeElement();
        shapeElement.setShape(ShapeElement.RECTANGLE);
        shapeElement.setRgbColor(new RgbColor(0,0,255));
        title.setBackground(shapeElement);
        return title;
    }
}

```

Provider 必须继承自 `BaseItemProvider` 和它的子类 `RecycleItemProvider`, 然后需要覆写 `getCount()`、`getItem()`、`getItemId()` 和 `getComponent()` 这 4 种方法。

在 `ListContainer` 绘制之前, 会首先调用 `public int getCount()` 方法, 这种方法的返回值代表 `listContainer` 的长度, 然后根据这个值来确定 `getComponent()` 的执行次数。

在绘制每一行之前都会执行一次 `getComponent()`, 用于获取当前行需要显示的内容, 其中, 第一个回调参数 `i` 代表当前的行数, 第二个回调参数 `component` 代表上一次绘制时本行显示的内容, 所以易知首次绘制时此参数为空 (因为还没有旧的 `component`)。

另外两个函数 `public Object getItem(int i)` 的作用是返回一个子 `Component`, 即 `ListContainer` 中的一个子条目。 `public long getItemId(int i)` 的作用是返回一个 `item` 的 `id`, 由参数 `i` 决定是哪个 `id`。

覆写完上述 4 个函数后, 就完成了 `Provider` 的简易构建。上述 `Provider` 的业务逻辑是使 `getComponent()` 每次运行时得到列表 `mList` 中的一个字符串, 然后实例化一个 `Text` 组件来展示这个字符串, 所以还需要定义一个名为 `mList` 的 `ArrayList` 用于承载这些字符串, 代码如下:

```

//MainAbility.java
DirectionalLayout myLayout = new DirectionalLayout(this);
DirectionalLayout.LayoutParams layoutConfig = new
DirectionalLayout.LayoutParams(DirectionalLayout.LayoutParams.MATCH_PARENT, DirectionalLayout.
LayoutParams.MATCH_PARENT);
layoutConfig.alignment = Layout.Alignment.HORIZONTAL_CENTER;
myLayout.setLayoutParams(layoutConfig);
layoutConfig.width = ComponentContainer.LayoutParams.MATCH_CONTENT;
layoutConfig.height = ComponentContainer.LayoutParams.MATCH_CONTENT;
List aList = new ArrayList<>();
ListContainer listContainer = new ListContainer(this);
listContainer.setLayoutParams(layoutConfig);
aList.add("测试 1");
aList.add("测试 2");
aList.add("测试 3");
aList.add("测试 4");
aList.add("测试 5");
aList.add("测试 6");
aList.add("测试 7");
listContainer.setItemProvider(new SamplePagerAdapter(aList));
myLayout.addComponent(listContainer);

```

上述代码创建了一个 list,用于承载 7 个字符串,把这个 list 传递给 Provider,最后对 xml 中的布局和 ListContainer 进行加载,将创建的 Provider 和布局中的 ListContainer 进行绑定,这样就能展示整个 ListContainer 了,展示效果如图 3.30 所示。



图 3.30 ListContainer 效果展示

3.2.5 滑动布局管理器 PageSlider

PageSlider 和 ListContainer 类似,是一种容器布局,需要 PageSliderProvider 作为 Provider 才能显示内容。PageSlider 可以允许用户左右或者上下滑动来翻页,其中每页都可以添加其他 Component 进行显示。

与上一节的 ListContainer 类似,首先需要定义一个 Provider,代码如下:

```
//MainAbilitySlice.java
private class PagerAdapter extends PageSliderProvider {
    private List list;
    public PagerAdapter(List<Image> list) {
        this.list = list;
    }

    @Override
    public int getCount() {
        return list.size();
    }

    @Override
    public Object createPageInContainer( ComponentContainer componentContainer, int i) {
        componentContainer.addComponent((Component)list.get(i));
        return list.get(i);
    }
}
```

```

        @Override
        public void destroyPageFromContainer ( ComponentContainer componentContainer, int i,
        Object o) {
            componentContainer.removeComponent(((Component)list.get(i));

        }
        @Override
        public boolean isPageMatchToObject(Component component, Object o) {
            return component == o;
        }
    }
}

```

可以看到,所有的 PageProvider 都需要继承自 PageSliderProvider 类,并且需要覆写 isPageMatchToObject (Componentcomponent, Object o)、destroyPageFromContainer(ComponentContainer componentContainer,int i,Object o)、createPageInContainer(ComponentContainer componentContainer,int i)、getCount()这 4 个函数。

其中,getCount 返回要滑动的 Component 的个数,createPageInContainer 是从当前 componentContainer 的指定位置 i 中添加 Component,然后返回这个 Component。

destroyPageFromContainer 是把 component 从 componentContainer 的当前位置 i 销毁。

在示例中首先构筑布局,然后初始化一个用来存储 Component 的列表,代码如下:

```

//MainAbilitySlice.java
DirectionalLayout myLayout = new DirectionalLayout(this);
DirectionalLayout. LayoutConfig layoutConfig = new DirectionalLayout. LayoutConfig
(ComponentContainer. LayoutConfig. MATCH_PARENT, ComponentContainer. LayoutConfig. MATCH_PARENT);
private List<Image> list = new ArrayList<>();

```

然后给 LayoutConfig 设置一些参数,这样可以保证 Component 居中展示,代码如下:

```

layoutConfig.alignment = LayoutAlignment.HORIZONTAL_CENTER;
myLayout.setLayoutConfig(layoutConfig);
layoutConfig.width = ComponentContainer. LayoutConfig. MATCH_CONTENT;
layoutConfig.height = ComponentContainer. LayoutConfig. MATCH_CONTENT;

```

接下来新建一个 PageSlider,代码如下:

```

PageSlider pageSlider = new PageSlider(this);

```

其中,Provider 中放入了包含 4 个 Image 的 list,代码如下:

```

//MainAbilitySlice.java
Image image = new Image(this);
image.setImageAndDecodeBounds(ResourceTable.Media_picture);
Image image1 = new Image(this);

```

```

image1.setImageAndDecodeBounds(ResourceTable.Media_pic);
image1.setWidth(1125);
image1.setHeight(760);
image1.setScaleMode(Image.ScaleMode.ZOOM_CENTER);

Image image2 = new Image(this);
image2.setImageAndDecodeBounds(ResourceTable.Media_pic2);
image2.setScaleMode(Image.ScaleMode.ZOOM_CENTER);
image2.setWidth(1125);
image2.setHeight(760);

Image image3 = new Image(this);
image3.setImageAndDecodeBounds(ResourceTable.Media_pic3);
image3.setScaleMode(Image.ScaleMode.ZOOM_CENTER);
image3.setWidth(1125);
image3.setHeight(760);
list.add(image);
list.add(image1);
list.add(image2);
list.add(image3);

```

然后可以把 PageSlider 和 Provider 关联起来,代码如下:

```

pageSlider.setProvider(new PagerAdapter(list));
myLayout.addComponent(pageSlider);

```

这样就可以看到翻页效果,运行上述代码,效果如图 3.31 所示,分别为第一张图片到最后一张图片的切换过程。

完成 PageSlider 后还可以初始化一个 PageSliderIndicator,然后把 Indicator 和 PageSlider 关联起来,代码如下:

```

PageSliderIndicator pageSliderIndicator = new PageSliderIndicator(this);
pageSliderIndicator.setViewPager(pageSlider);
pageSliderIndicator.setLayoutConfig(layoutConfig);
myLayout.addComponent(pageSliderIndicator);

```

这样就完成了指示器的构造,运行代码可以看到翻页效果,如图 3.32 所示。

可以看到灰色的长条表示当前所在的页面,Indicator 明确地指示了 PageSlider 的页面。

3.2.6 其他布局容器

除以上布局容器之外,HarmonyOS 还提供了多种布局容器,这里只对它们做一个简单的介绍,具体的实现可以自行尝试。



图 3.31 PageSlide 翻页展示效果



图 3.32 指示器效果运行效果图

窗口布局容器 `StackLayout` 提供一个窗口,提供一个框架布局,其中的元素可以重叠。`StackLayout` 用于在屏幕上保留一个区域来显示视图中的元素。通常,框架布局中只应该放置一个子组件。如果存在多个子组件,则显示最新添加的组件,之前添加的组件会被遮盖掉。

桌面布局容器 `TableLayout` 是一种像 Windows 桌面一样的排列布局容器。该布局容器用于在带有表的组件中安排组件。`TableLayout` 提供了对齐和安排组件的接口,以在带有表的组件中显示组件。可以配置排列方式、行数和列数及组件的位置。

还有自适应框容器 `AdaptiveBoxLayout`。自适应框将自动分为具有相同宽度和可能不同高度的框的行和列。框的宽度取决于布局宽度和每行中框的数量,这由它的 `LayoutConfig` 指定。新行仅在上一行填充后才开始。每个框都包含一个子组件。每个框的高度取决于其包含的子组件的高度。每行的高度由该行中的最高框确定。自适应框布局容器的宽度只能设置为 `MATCH_PARENT` 或固定值,但是开发者可以为容器中的组件自由设置长度、宽度和对齐方式。

3.3 Java UI 动画

3.3.1 动画类介绍

动画是组件的基础特性之一,精心设计的动画使 UI 变化更直观,有助于改进应用程序的外观并改善用户体验。HarmonyOS 在 Java UI 框架下提供了 `Animator` 类对各种组件添加动画效果,它的子类有数值动画 (`AnimatorValue`) 和属性动画 (`AnimatorProperty`),还提供了将多个动画同时操作的动画集合 (`AnimatorGroup`),开发者可以自由组合这些动画元素,从而构建丰富多样的动画效果。`Animator` 作为动画的基类,提供了与动画的启动、停止、暂停和恢复相关的 API,同时还支持为动画设定持续时间、启动延迟、重复次数和指定的曲线类型。`AnimatorValue` 提供随时间变化的变动数值,开发者可以自定义动画的样式,代入这个数值即可实现动画效果。`AnimatorProperty` 则是对 `AnimatorValue` 的自动化封装,原生组件提供了缩放、平移、旋转等动画效果。`AnimatorGroup` 则可以通过创建一个动画组,实现多个组建的序列或并行播放动画。

3.3.2 数值动画 `AnimatorValue`

前文说到 `AnimatorValue` 提供随时间变化的数值,这个数值是从 0 到 1 变化的浮点数,本身与 `Component` 对象或种类无关。由于 `AnimatorValue` 是 `Animator` 的子类,所以开发者可以在 `AnimatorValue` 中调用 `Animator` 中 API 并自定义数值从 0 到 1 的变化过程,例如更长的动画持续时间意味着数值的变化会更慢、不同的变化曲线则可以让数值实现各种各样的非匀速变化、设定重复次数可以让数值从 0 到 1 变化循环数次……

开发者可以利用数值随时间变化的特性,实现动画效果。例如通过值的变化改变控件

的属性,从而实现控件的运动。

在本示例中主要演示一个 Text 组件的动画,并通过 Button 启动动画播放。先定义布局设置 LayoutConfig 和用于设定背景颜色的 ShapeElement 等参数,然后定义一个 Button 用于启动动画,代码如下:

```
//动画效果布局定义
layoutConfig.alignment = LayoutAlignment.CENTER;
layoutConfig.setMargins(100,600,100,100);
myLayout.setLayoutConfig(layoutConfig);
layoutConfig.width = ComponentContainer.LayoutConfig.MATCH_CONTENT;
layoutConfig.height = ComponentContainer.LayoutConfig.MATCH_CONTENT;
ShapeElement shapeElement = new ShapeElement();
shapeElement.setShape(ShapeElement.RECTANGLE);
shapeElement.setCornerRadius(80);
shapeElement.setRgbColor(new RgbColor(0,255,255));

Button button = new Button(this);
button.setLayoutConfig(layoutConfig);
button.setBackground(shapeElement);
button.setText("启动动画效果");
button.setTextSize(130);
```

接下来声明一个用于展示的 Text,此时这个 Text 可以视为动画的初始状态,代码如下:

```
Text t = new Text(this);
t.setLayoutConfig(layoutConfig);
t.setText("动画测试");
t.setTextColor(Color.RED);
t.setTextAlignment(Component.HORIZONTAL);
t.setTextSize(60);
```

然后实例化 AnimatorValue,并设置变化属性,在这里设置动画的持续时间为 2000ms、启动延时为 1000ms、循环播放 2 次,并设置变化曲线为 Bounce 型,代码如下:

```
AnimatorValue animator = new AnimatorValue();
animator.setDuration(2000);
animator.setDelay(1000);
animator.setLoopedCount(2);
animator.setCurveType(Animator.CurveType.BOUNCE);
```

接下来需要对 AnimatorValue 实例设置监听器 ValueUpdateListener,其中的回调参数 v 就是前述中从 0 到 1 变化的数值,这个数值会在设置的 Duration 时间内从 0 变化到 1,接下来将动画样式与 v 实现连接,如本例中使用了设定文本大小的函数 setTextSize((int)

($200 * v$)), 那么文本大小会在 2s 内从 $200 * 0 = 0$ 变化到 $200 * 1 = 200$, 即完成文本字体大小从 0 到 200 的变化, 每一时刻的变化速度和之前设置的变化曲线有关。添加监听事件的代码如下:

```
animator.setValueUpdateListener((animatorValue, v) -> t.setTextSize((int) (200 * v)));
```

为 Button 设置监听事件, 如果检测到按钮单击, 则启动动画, 代码如下:

```
button.setClickListener(component -> animator.start());
```

AnimatorValue 动画效果如图 3.33 所示。



图 3.33 在 Text 组件上添加 AnimatorValue 动画效果图

可以清楚地看到,用于测试的 Text 组件已经被添加了动画效果,由于 Duration 设置值为 2000,因此这个动画效果会在 2s 内播放完毕。

3.3.3 属性动画 AnimatorProperty

因为 AnimatorProperty 是对 AnimatorValue 的自动化封装,内置实现了 Component 的平移、旋转、缩放的动画效果,使用方法较 AnimatorValue 更为简单。

首先直接对一个 Component(此处以 3.3.2 节中的 Button 为例)实例化 AnimatorProperty 对象,代码如下:

```
AnimatorProperty animator = button.createAnimator();
```

为 AnimatorProperty 实例设置变化属性,可链式调用,代码如下:

```
animator.moveFromX(50).moveToX(1000).alpha(0).setDuration(2500).setDelay(500).setLoopedCount(5);
```

如上述代码设置了动画的起始 X 轴位置为 50、终止 X 轴位置为 1000、透明度为 0、持续时长为 2500 ms、启动延时为 500 ms 和循环次数 5 次。下面通过 start() 方法启动动画,代码如下:

```
animator.start();
```

其中,AnimatorProperty 实例可以重复使用,例如可以使用 setTarget() 方法改变关联的 Component 对象,代码如下:

```
animator.setTarget(t);
```

下面来看一个示例,为了便于展示,新建一个 Button 组件,并设置 LayoutConfig 和背景 ShapeElement,代码如下:

```
//动画效果布局定义
layoutConfig.setMargins(100,600,100,100);
myLayout.setLayoutConfig(layoutConfig);
layoutConfig.width = ComponentContainer.LayoutConfig.MATCH_CONTENT;
layoutConfig.height = ComponentContainer.LayoutConfig.MATCH_CONTENT;
ShapeElement shapeElement = new ShapeElement();
shapeElement.setShape(ShapeElement.RECTANGLE);
shapeElement.setCornerRadius(80);
shapeElement.setRgbColor(new RgbColor(0,255,255));
Button button = new Button(this);
button.setLayoutConfig(layoutConfig);
```

```
button.setBackground(shapeElement);  
button.setText("启动动画效果");  
button.setTextSize(130);
```

接下来创建一个属性动画,将这个动画和前面的 Button 实例关联起来。

设置动画的效果为 2.5s 内从 X 轴的 50 位置,移动到屏幕水平方向上 500 的位置,然后旋转 90°,代码如下:

```
AnimatorProperty animator = button.createAnimatorProperty();  
animator.moveFromX(50). moveToX(500). rotate(90). alpha((float) 0.5). setDuration(2500).  
setDelay(500);  
animator.setCurveType(Animator.CurveType.BOUNCE);
```

运行上述代码,效果如图 3.34 所示。



图 3.34 属性动画效果

3.3.4 动画集合 AnimatorGroup

如果需要使用一个组合动画,可以把多个动画对象添加到 AnimatorGroup 中。AnimatorGroup 提供了两种方法:runSerially()和 runParallel(),分别表示动画按序列启动和动画并发启动。下面是一个简单的示例。

首先需要声明一个动画集合 AnimatorGroup,代码如下:

```
AnimatorGroup animatorGroup = new AnimatorGroup();
```

其次,实例化多个动画,并将它们添加到动画集合中。这里实例化了 4 个动画 animator1~animator4,分别配置在 4 个按钮 button1~button4 上。动画的配置方法均参考 3.3.3 节的属性动画,animator1 只从左向右移动,animator2 从右向左移动并旋转 90°,animator3 从左向右移动并旋转 90°,animator4 与 animator2 相同,代码如下:

```
//动画的实例化与配置
AnimatorProperty animator1 = button1.createAnimatorProperty();
animator1.moveFromX(50).moveToX(500).alpha((float) 0.5).setDuration(2500).setDelay(500);
AnimatorProperty animator2 = button2.createAnimatorProperty();
animator2.moveFromX(50).moveToX(500).rotate(90).alpha((float) 0.5).setDuration(2500).
setDelay(500);
AnimatorProperty animator3 = button3.createAnimatorProperty();
animator3.moveFromX(500).moveToX(50).rotate(90).alpha((float) 0.5).setDuration(2500).
setDelay(500);
AnimatorProperty animator4 = button4.createAnimatorProperty();
animator4.moveFromX(50).moveToX(500).rotate(90).alpha((float) 0.5).setDuration(2500).
setDelay(500);
AnimatorGroup animatorGroup = new AnimatorGroup();
//4 个动画同时播放
animatorGroup.runParallel(animator1, animator2, animator3, animator4);
```

设置一个按钮并配置监听事件,用于启动动画集合,代码如下:

```
button5.setClickListener(new Component.ClickedListener() {
    @Override
    public void onClick(Component component) {
        animatorGroup.start();
    }
});
```

运行上述代码,可以看到动画 1 至动画 4 同时播放的效果,如图 3.35 所示。

若将上文中的 animatorGroup.runParallel(animator1, animator2, animator3, animator4)更



图 3.35 同时播放动画组效果

改为 `animatorGroup.runSerially(animator1, animator2, animator3, animator4)`, 则动画将从并行播放变为序列播放, 效果如图 3.36 所示。

为了更加灵活地处理多个动画的播放, 例如一些动画序列播放, 而另一些动画并行播放, Java UI 框架提供了更方便的动画 Builder 接口。首先声明 `AnimatorGroup Builder`, 使用 `addAnimators()` 方法为 Builder 添加多个动画, 同一个 `addAnimators()` 内的动画并行播放, 不同的 `addAnimators()` 中的动画按添加顺序播放。本例中动画的播放效果为首先播放完 `animator1`, 随后并行启动 `animator2` 和 `animator3` 至两者全部播放完毕, 最后启动 `animator4`, 代码如下:

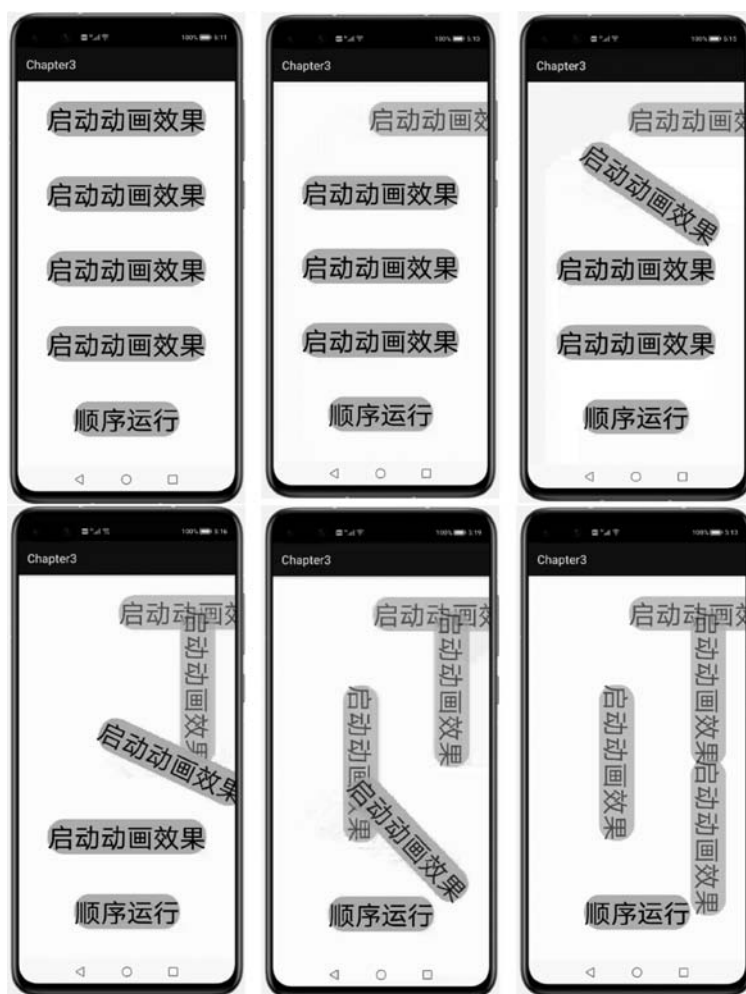


图 3.36 顺序播放动画组效果

```
//动画播放
AnimatorGroup animatorGroup = new AnimatorGroup();
AnimatorGroup.Builder animatorGroupBuilder = animatorGroup.build();
animatorGroupBuilder.addAnimators ( animator1 ). addAnimators ( animator2, animator3 ).
addAnimators(animator4);
//4 个动画的顺序为: animator1 -> animator2/animator3 -> animator4
button5.setOnClickListener(new Component.ClickedListener() {
    @Override
    public void onClick(Component component) {
        animatorGroup.start();
    }
});
```

运行代码得到效果如图 3.37 所示。



图 3.37 动画组 Builder 接口效果