



上海建桥学院

SHANGHAI JIAN QIAO UNIVERSITY



布局

万永权

学习目标/Target



了解View与ViewGroup的简介，能够说出View与ViewGroup的作用和关联

掌握界面布局在XML文件中与Java代码中的编写方式，能够独立编写界面布局

掌握编写简单Android程序的步骤，能够编写一个Hello World程序

掌握常见界面布局的特点及使用，能够搭建简单的界面布局

Contents

主要内容



View视图

界面布局编写方式

界面布局的通用属性

线性布局





视图



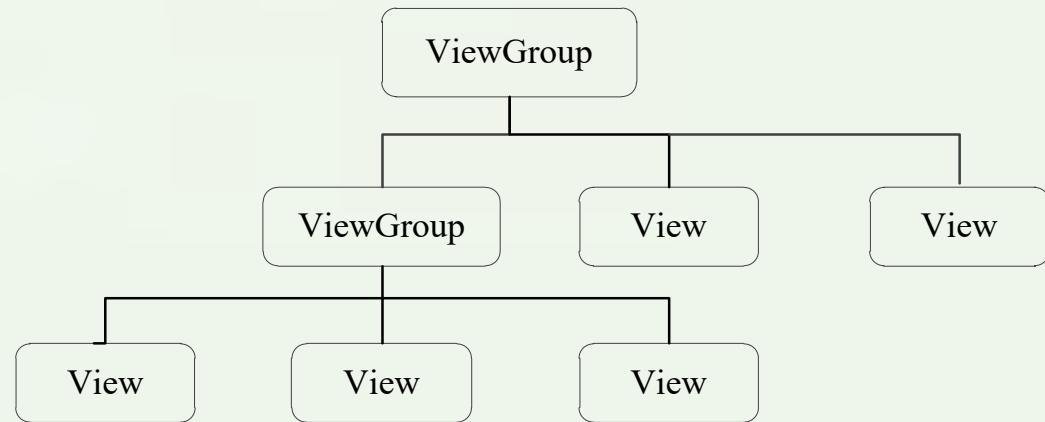


先定一个小
目标！

- 了解 View 与 ViewGroup 的简介，能够说出 View 与 ViewGroup 的作用和关联

// View视图

所有的UI元素都是通过View与ViewGroup构建的，对于一个Android应用的用户界面来说，ViewGroup作为容器盛装界面中的控件，它可以包含普通的View控件，也可以包含ViewGroup。





3.1.1 UI概览



View类

- View是Android中用户界面体现的基础单位。
- Android界面的基本UI元素由android.view.View提供并实现。
- View类架构显示了View类的继承关系。
- View是所有与用户交互的组件（如TextView、Button等）的基类。

View Added in API level 1

[Kotlin](#) | [Java](#)

```
public class View
    extends Object implements Drawable.Callback, KeyEvent.Callback, AccessibilityEventSource
    <java.lang.Object
        &gt; android.view.View

    &lt; Known direct subclasses
        AnalogClock, ImageView, KeyboardView, MediaRouteButton, ProgressBar, Space, SurfaceView, TextView, TextureView, ViewGroup,
        ViewStub

    &lt; Known indirect subclasses
        AbsListView, AbsSeekBar, AbsSpinner, AbsoluteLayout, ActionMenuView, AdapterView<T extends Adapter>, AdapterViewAnimator,
        AdapterViewFlipper, AppWidgetHostView, AutoCompleteTextView, Button, CalendarView, CheckBox, CheckedTextView, Chronometer,
        and 52 others.
```

View类架构



3.1.1 UI概览



ViewGroup子类

- ViewGroup是一个特殊的View，它继承于android.view.View，如图3-3所示。
- 功能：装载和管理下一层的View对象或ViewGroup对象，它是一个容纳其他元素的容器。

ViewGroup Added in API level 1

[Kotlin](#) | [Java](#)

```
public abstract class ViewGroup
    extends View implements ViewParent, ViewManager

    java.lang.Object
        ↴ android.view.View
            ↴ android.view.ViewGroup

    ▾ Known direct subclasses
        AbsoluteLayout, AdapterView<T extends Adapter>, FragmentBreadCrumbs, FrameLayout, GridLayout, LinearLayout, RelativeLayout,
        SlidingDrawer, Toolbar, TvView

    ▾ Known indirect subclasses
        AbsListView, AbsSpinner, ActionMenuView, AdapterViewAnimator, AdapterViewFlipper, AppWidgetHostView, CalendarView,
        DatePicker, DialerFilter, ExpandableListView, Gallery, GestureOverlayView, GridView, HorizontalScrollView, ImageSwitcher, and 20
        others.
```

ViewGroup子类架构

// 界面布局

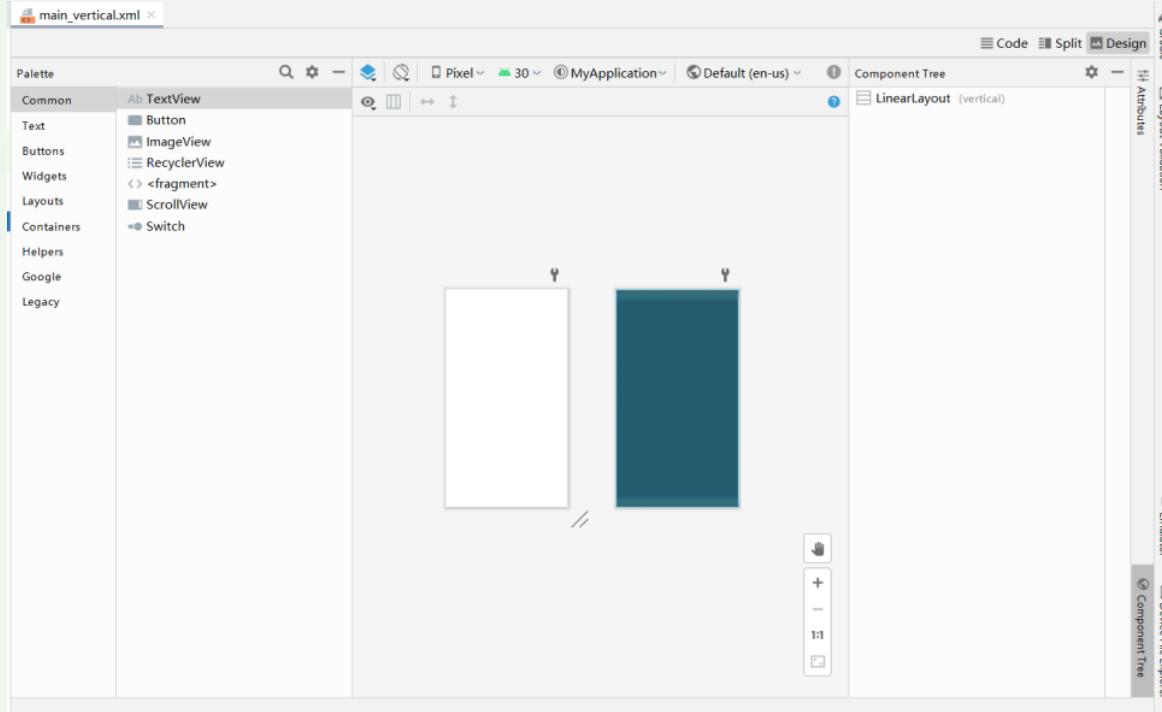
□ 界面布局

- 界面布局（Layout）是用户界面结构的描述，定义了界面中所有的元素、结构和相互关系
- 声明Android程序的界面布局有两种方法
 - ◆ 使用XML文件描述界面布局（推荐使用）
 - ◆ 在程序运行时动态添加或修改界面布局
- 既可以独立使用任何一种声明界面布局的方式，也可以同时使用两种方式

// 界面布局

• 界面布局

- 使用XML文件声明界面布局的优势
 - 将程序的表现层和控制层分离，修改用户界面时，无需更改程序的源代码
 - 可通过Android Studio的“可视化编辑器”直接查看用户界面，有利于加快界面设计的过程





界面布局编写方式

- 掌握**在XML文件中编写布局**，能够搭建简单的布局界面
- 掌握**在Java代码中编写布局**，能够搭建简单的布局界面



// 界面布局编写方式

在实现Android界面效果之前，我们首先需要编写界面布局，界面布局的编写方式有2种，第1种是在[XML文件中编写布局](#)，第2种是在[Java代码中编写布局](#)。

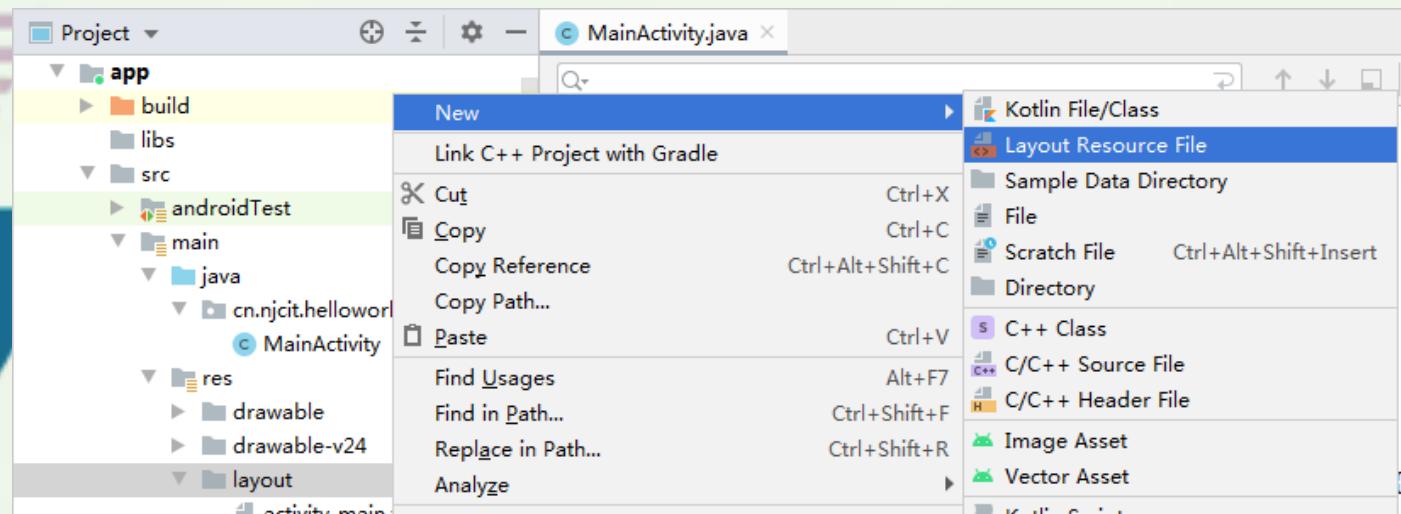
- 在XML文件中编写布局：[推荐此种方式编写布局](#)
 - ◆ 有效的将界面中的布局代码与Java代码隔离，[使程序的结构更加清晰](#)。
- 在Java代码中编写布局
 - ◆ 在Android中所有布局和控件的对象都可以通过new关键字创建出来，[将创建的View控件添加到ViewGroup布局中](#)，从而实现View控件在布局界面中显示。



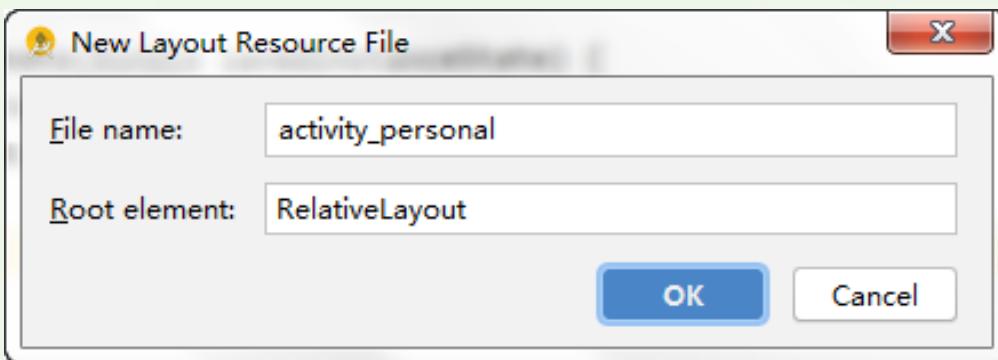
3.2.1 创建XML布局文件



在Android Studio中创建XML布局文件的一般步骤如下。



创建XML布局文件



布局文件名和布局名称

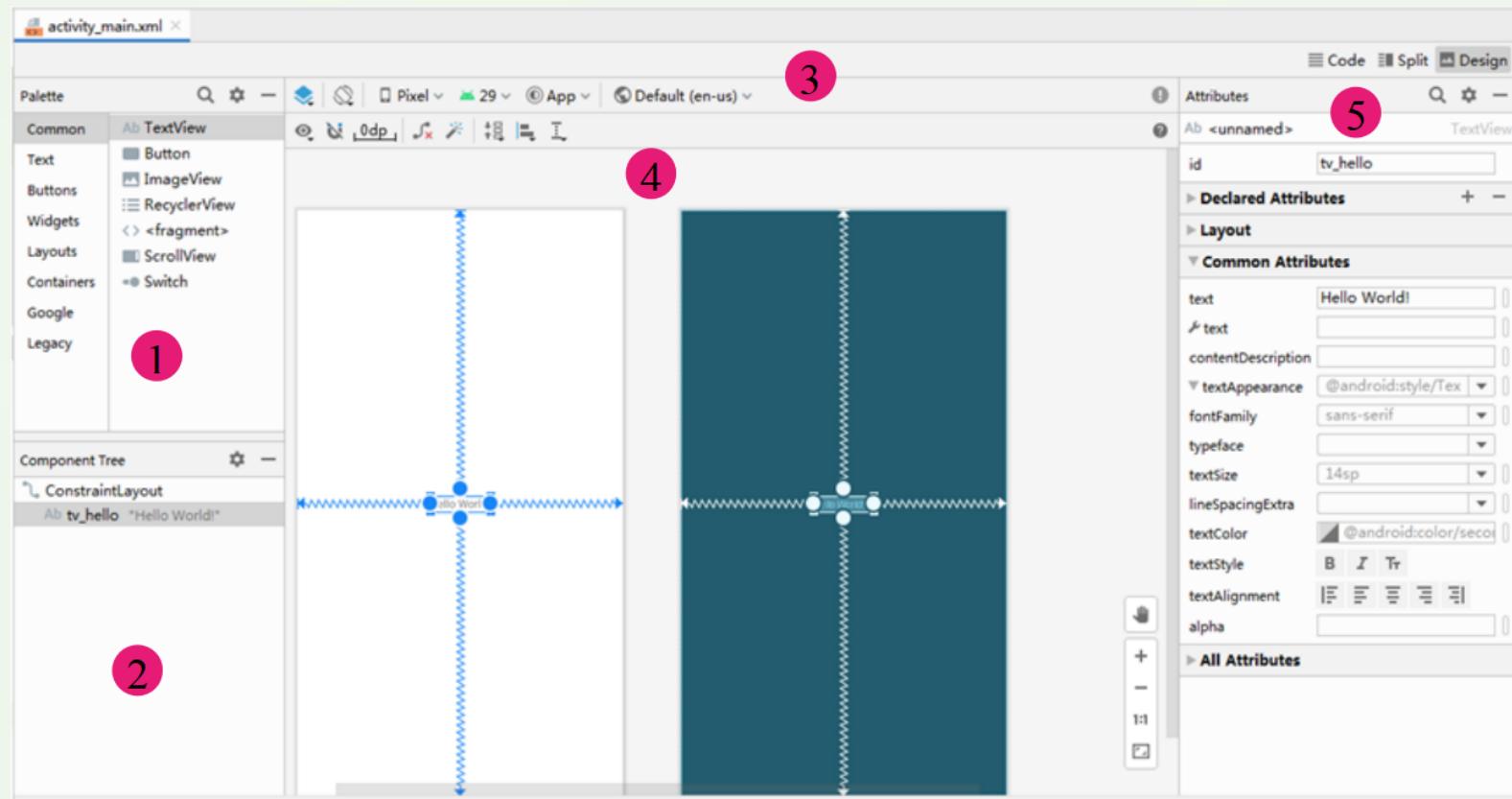


3.2.2 可视化界面编辑器



Design编辑器窗口

当打开 XML 布局文件时，单击下方的“Design”标签，将显示Design编辑器窗口，如图所示。



Design编辑器窗口



3.2.2 可视化界面编辑器



Text编辑器窗口

当打开XML布局文件时，单击下方的“Text”标签，将显示Text编辑器窗口，如图所示。

```
<ImageView  
    android:src="@drawable/ic_star"  
    android:layout_width="36dp"  
    android:layout_height="36dp"  
    app:layout_editor_absoluteX="359dp"  
    app:layout_editor_absoluteY="104dp"  
    android:id="@+id/favorite"  
    android:background="@drawable/info_background"  
    android:padding="5dp"  
    android:contentDescription="dummy"  
    app:layout_constraintTop_creator="1"  
    app:layout_constraintRight_creator="1"  
    app:layout_constraintBottom_creator="0"  
    app:layout_constraintTop_toTopOf="parent"  
    app:layout_constraintRight_toRightOf="parent"  
    android:layout_marginEnd="16dp"  
    app:layout_constraintBottom_toBottomOf="parent"  
    android:layout_marginBottom="16dp"  
    app:layout_constraintVertical_bias="0.19" />  
  
<TextView  
    android:text="Singapore"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    app:layout_editor_absoluteX="16dp"  
    app:layout_editor_absoluteY="140dp"  
    android:id="@+id/title"  
    android:textSize="24sp"  
    app:layout_constraintLeft_creator="1"  
    app:layout_constraintTop_creator="0"  
    app:layout_constraintLeft_toLeftOf="parent"  
    android:layout_marginStart="16dp"  
    app:layout_constraintTop_toBottomOf="@+id/header"  
    android:layout_marginTop="16dp" />  
  
<EditText  
    android:layout_width="0dp"  
    ...>
```

The screenshot shows the Android Studio interface with the XML layout file open. The left side features the Project, Structure, and Captures toolbars. The main area displays the XML code for the layout. On the right, there's a Preview window showing a mobile application interface with a city skyline at night and a camera settings overlay. Below the preview is a detailed description of Singapore. At the bottom, there are buttons for DISCARD and UPLOAD.

Text编辑器窗口



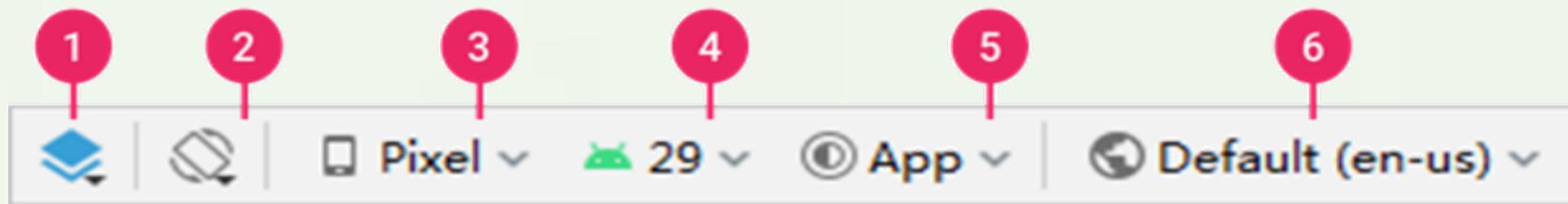


3.2.2 可视化界面编辑器



① 预览外观设计

Design编辑器顶行的工具栏按钮可用于在编辑器中配置布局的外观，如图所示。



Design编辑器工具栏

② 添加界面控件

- 在Android Studio中，通过将小部件从“Palette”窗格拖曳到Design编辑器，并在“Attributes”窗格中优化布局属性，来完成界面的布局工作。



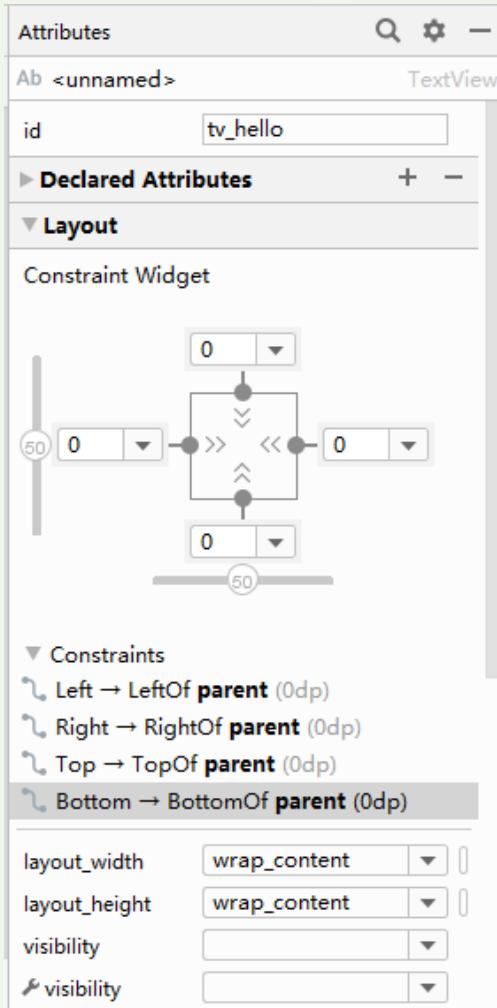
3.2.2 可视化界面编辑器



设置控件属性

在编辑器中选择要查看的控件并编辑该控件的常用属性。

如果选择的视图是ConstraintLayout的子项，则“Attributes”窗格在顶部提供一个带有多个控件的视图检查器，如图所示。



“Attributes”窗格



3.2.3 视图基本属性与事件



XML布局中的View常见XML属性与对应的方法

属性名称	对应方法	描述
android:clickable	setClickable()	设置View是否响应单击事件
android:visibility	setVisibility()	控制View的可见性
android:focusable	setFocusable()	控制View是否可以获取焦点
android:id	setId()	为View设置标识符，可通过findViewById()方法获取
android:longClickable	setLongClickable()	设置View是否响应长单击事件
android:soundEffectsEnabled	setSoundEffectsEnabled()	设置当View触发单击等事件时是否播放音效
android:saveEnabled	setSaveEnabled()	如果未设置，当View被冻结时将不会保存其状态
android:nextFocusLeft	setNextFocusLeftId()	定义当向左搜索时应该获取焦点的View

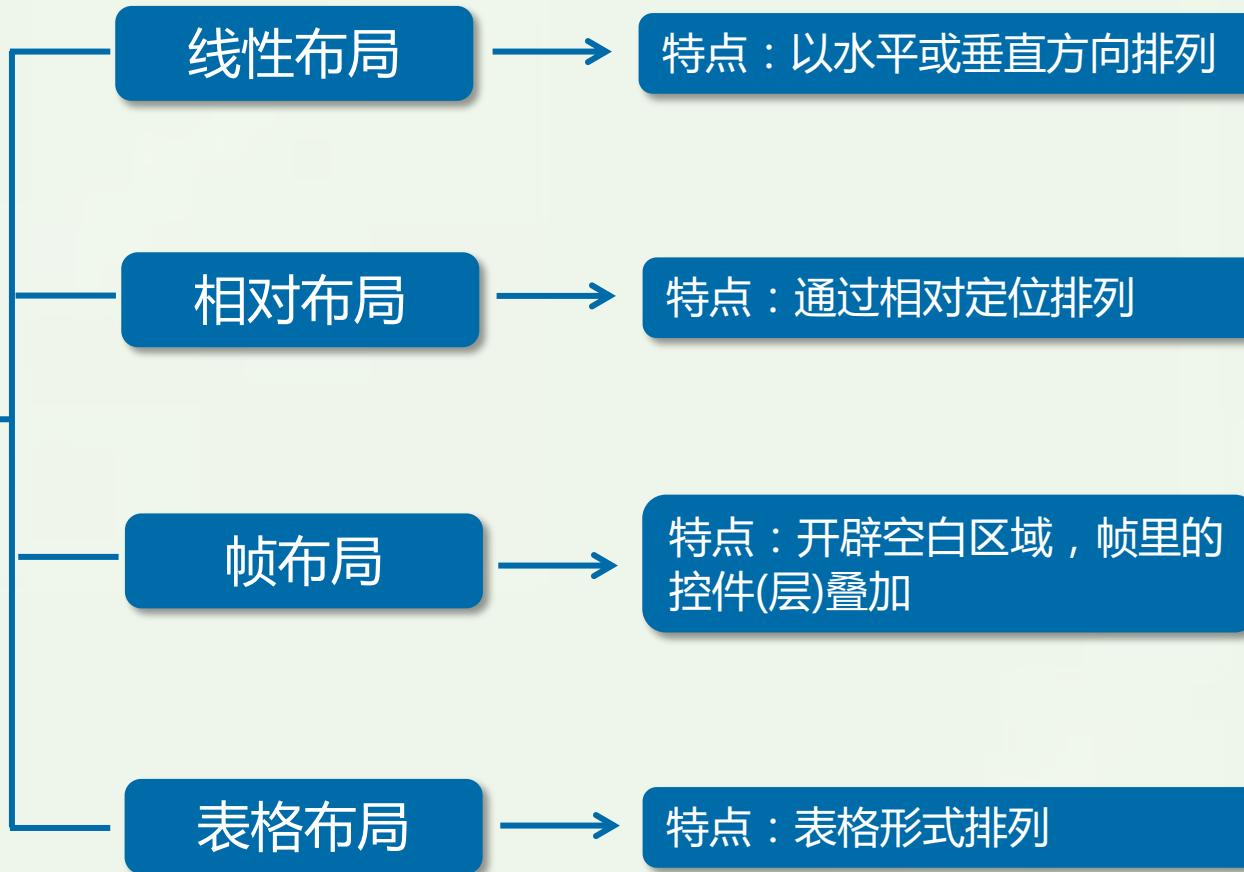


界面布局的通用属性



// 界面布局的通用属性

四种常用布局





// 界面布局的通用属性

Android系统提供的四种常用布局直接或者间接继承自ViewGroup，因此这四种常用布局也支持在ViewGroup中定义属性，这些属性可以看作是布局的通用属性。这些通用属性如下表所示。

属性名称	功能描述
android:id	设置布局的标识
android:layout_width	设置布局的宽度
android: layout_height	设置布局的高度
android:background	设置布局的背景
android:layout_margin	设置当前布局与屏幕边界或与周围控件的距离
android:padding	设置当前布局与该布局中控件的距离

// 界面布局的通用属性

布局的通用属性



android:id

1. 用于设置当前布局的唯一标识。
2. 在XML文件中它的属性值是通过“@+id/属性名称”定义。



android:layout_width

1. 用于设置布局的宽度，其值可以是具体的尺寸，如50dp，也可以是系统定义的值。
2. 系统定义的值有 fill_parent、match_parent 和 wrap_content



android:layout_height

1. 用于设置布局的高度，其值可以是具体的尺寸，如50dp，也可以是系统定义的值。
2. 系统定义的值有 fill_parent、match_parent 和 wrap_content

// 界面布局的通用属性

布局的通用属性



android:background

用于设置布局背景。其值可以引用图片资源，也可以是颜色资源。



android:layout_margin

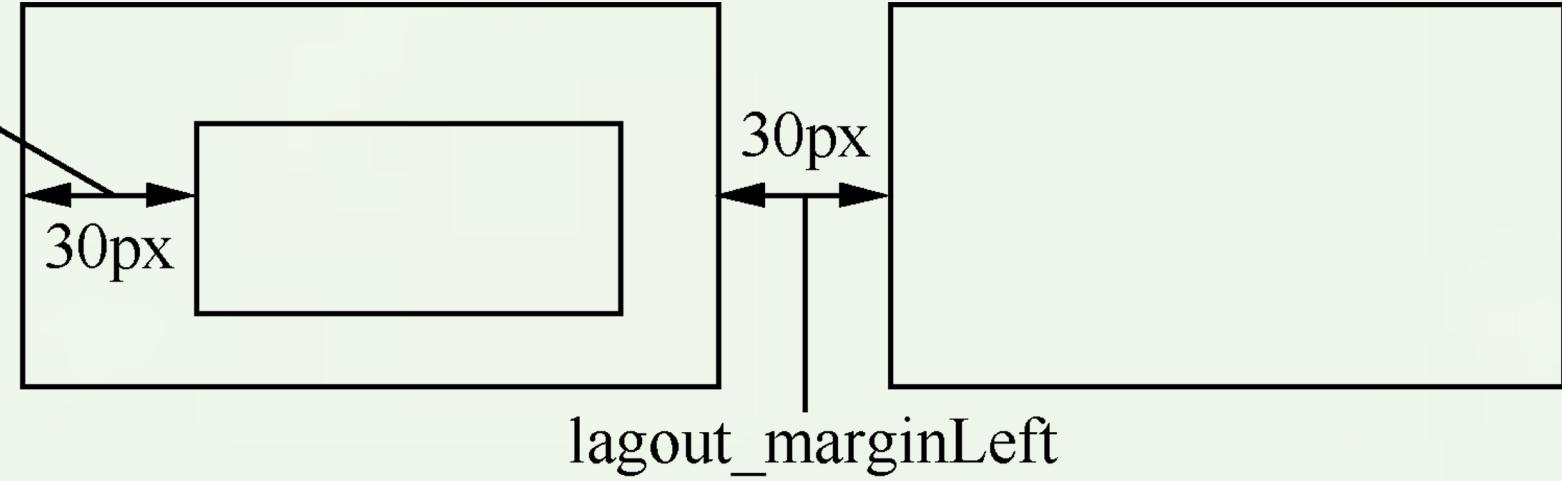
用于设置当前布局与屏幕边界、周围布局或控件的距离。属性值为具体的尺寸，如45dp。



android:padding

用于设置当前布局内控件与该布局的距离，其值可以是具体的尺寸，如45dp。

paddingLeft



padding和layout_margin的区别

// 界面布局的通用属性

布局的通用属性



android:gravity

用于将对象放置在可能更大的容器中的标准. 针对设置该属性的容器的内容生效。



android:layout_gravity

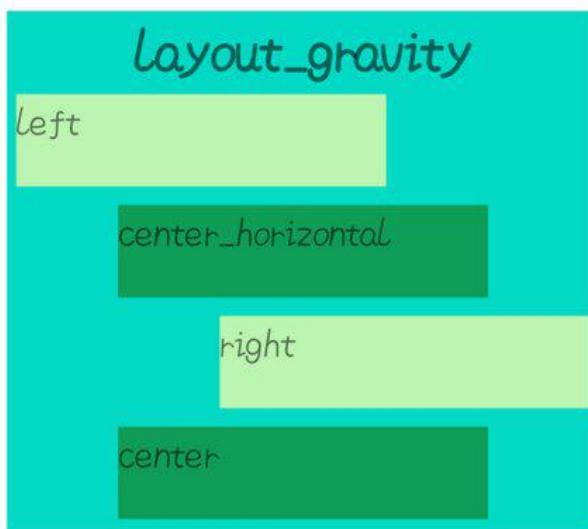
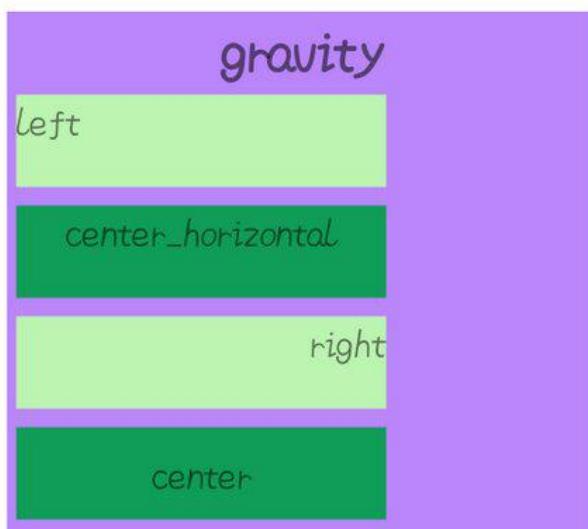
设置View or Layout 相对于其父控件的重力。

```
<LinearLayout  
    android:layout_width="match_parent"  
    android:layout_height="0dp"  
    android:layout_weight="1"  
    android:background="@color/purple_200"  
    android:baselineAligned="false"  
    android:orientation="vertical"  
    android:layout_margin="20dp">  
  
    <TextView  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_gravity="center_horizontal"  
        android:text="gravity"  
        android:textSize="30sp"  
        android:textStyle="bold"/>  
  
    <TextView  
        android:layout_width="200dp"  
        android:layout_height="50dp"  
        android:background="#bcf5b1"  
        android:gravity="left"  
        android:layout_margin="5dp"  
        android:text="left"  
        android:textSize="20sp"/>  
  
    <TextView  
        android:layout_width="200dp"  
        android:layout_height="50dp"  
        android:background="#0F9D5B"  
        android:gravity="center_horizontal"  
        android:layout_margin="5dp"  
        android:text="center_horizontal"  
        android:textSize="20sp"/>  
  
    <TextView  
        android:layout_width="200dp"  
        android:layout_height="50dp"  
        android:background="#bcf5b1"  
        android:gravity="right"  
        android:text="right"  
        android:layout_margin="5dp"  
        android:textSize="20sp"/>  
  
    <TextView  
        android:layout_width="200dp"  
        android:layout_height="50dp"  
        android:background="#0F9D5B"  
        android:gravity="center"  
        android:text="center"  
        android:layout_margin="5dp"  
        android:textSize="20sp"/>  
  
</LinearLayout>
```

```
<LinearLayout  
    android:layout_width="match_parent"  
    android:layout_height="0dp"  
    android:layout_weight="1"  
    android:background="@color/teal_200"  
    android:orientation="vertical"  
    android:layout_margin="20dp">  
  
    <TextView  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_gravity="center_horizontal"  
        android:text="layout_gravity"  
        android:textSize="30sp"  
        android:textStyle="bold"/>  
  
    <TextView  
        android:layout_width="200dp"  
        android:layout_height="50dp"  
        android:layout_gravity="left"  
        android:background="#bcf5b1"  
        android:text="left"  
        android:layout_margin="5dp"  
        android:textSize="20sp"/>  
  
    <TextView  
        android:layout_width="200dp"  
        android:layout_height="50dp"  
        android:layout_gravity="center_horizontal"  
        android:background="#0F9D5B"  
        android:layout_margin="5dp"  
        android:text="center_horizontal"  
        android:textSize="20sp"/>  
  
    <TextView  
        android:layout_width="200dp"  
        android:layout_height="50dp"  
        android:layout_gravity="right"  
        android:background="#bcf5b1"  
        android:text="right"  
        android:layout_margin="5dp"  
        android:textSize="20sp"/>  
  
    <TextView  
        android:layout_width="200dp"  
        android:layout_height="50dp"  
        android:layout_gravity="center"  
        android:background="#0F9D5B"  
        android:text="center"  
        android:layout_margin="5dp"  
        android:textSize="20sp"/>  
  
</LinearLayout>
```

1:46:08
0.13 Kb/s 4G 16%

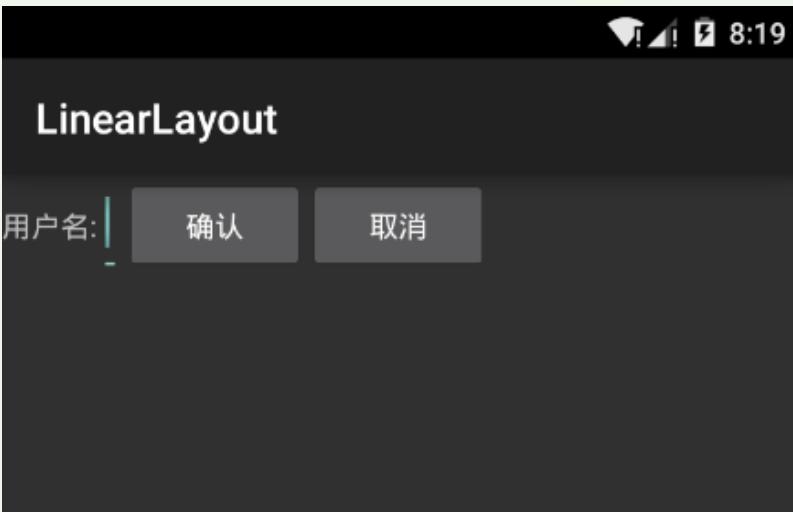
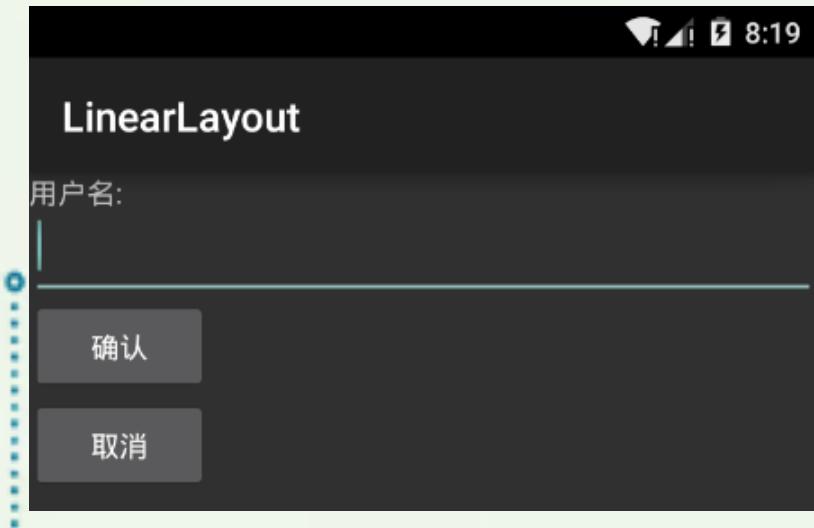
GFG Layout



5.3 界面布局

• 5.3.1 线性布局

- 线性布局（LinearLayout）是一种重要的界面布局中，也是经常使用到的一种界面布局
- 在线性布局中，所有的子元素都按照垂直或水平的顺序在界面上排列
 - 如果垂直排列，则每行仅包含一个界面元素
 - 如果水平排列，则每列仅包含一个界面元素





5.3 界面布局

• 5.3.1 线性布局

- 最小化的线性布局XML文件：

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <LinearLayout
3.     xmlns:android="http://schemas.android.com/apk/res/android"
4.     android:layout_width="match_parent"
5.     android:layout_height="wrap_content"
6.     android:orientation="vertical">
7. </LinearLayout>
```

- 第2行代码是声明XML文件的根元素为线性布局
- 第4、5、6行代码是在属性编辑器中修改过的宽度、高度和排列方式的属性



// 线性布局LinearLayout

除了布局的通用属性外，LinearLayout布局还有两个比较常用的属性，分别是 `android:orientation` 和 `android:layout_weight`，具体介绍如下所示。

属性名称	功能描述
<code>android:orientation</code>	设置布局内控件的排列顺序
<code>android:layout_weight</code>	在布局内设置控件权重，属性值可直接写int值

属性`android:orientation`的值为可选值，可选值为`vertical`和`horizontal`。

(1) `vertical`：表示LinearLayout布局内控件依次从上到下**竖直排列**。

(2) `horizontal`：表示LinearLayout布局内控件依次从左到右**水平排列**。

属性`android:layout_weight`：

(1) 该属性被称为**权重**，通过设置该属性值，可使**布局内的控件按照权重比显示大小**。

(2) 在进行屏幕适配时起到**关键作用**。



5.3 界面布局

• 5.3.1 线性布局

- 修改界面控件的属性

编号	类型	属性	值
1	TextView	Id	@+id/label
		Text	用户名:
2	EditText	Id	@+id/entry
		Layout width	match_parent
		Text	[null]
3	Button	Id	@+id/ok
		Text	确认
4	Button	Id	@+id/cancel
		Text	取消

- ID是一个字符串，编译时被转换为整数，可以用来在代码中引用界面元素
- 一般仅在代码中需要动态修改的界面元素，才界面元素设置ID，反之则不需要设置ID



5.3 界面布局

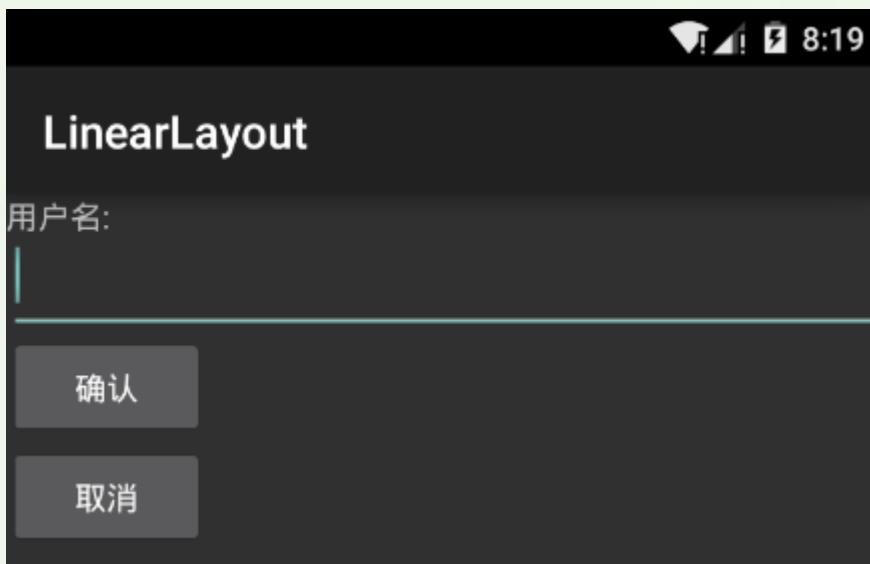
• 5.3.1 线性布局

- 打开XML文件编辑器查看main_vertical.xml文件代码：

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <LinearLayout
3.     xmlns:android="http://schemas.android.com/apk/res/android"
4.     android:layout_width="match_parent"
5.     android:layout_height="wrap_content"
6.     android:orientation="vertical">
7.
8.     <TextView android:id="@+id/label"
9.             android:layout_width="wrap_content"
10.            android:layout_height="wrap_content"
11.            android:text="用户名:" >
12.        </TextView>
13.        <EditText android:id="@+id/entry"
14.                 android:layout_height="wrap_content"
15.                 android:layout_width="match_parent">
16.            </EditText>
```

5.3 界面布局

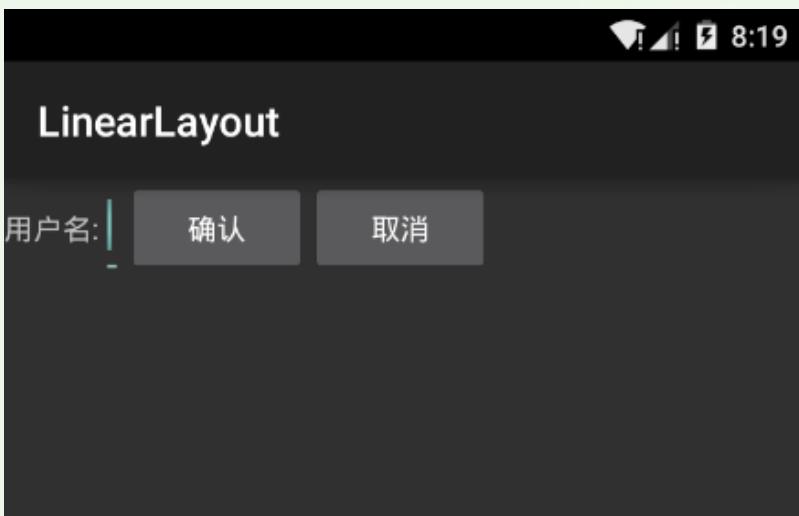
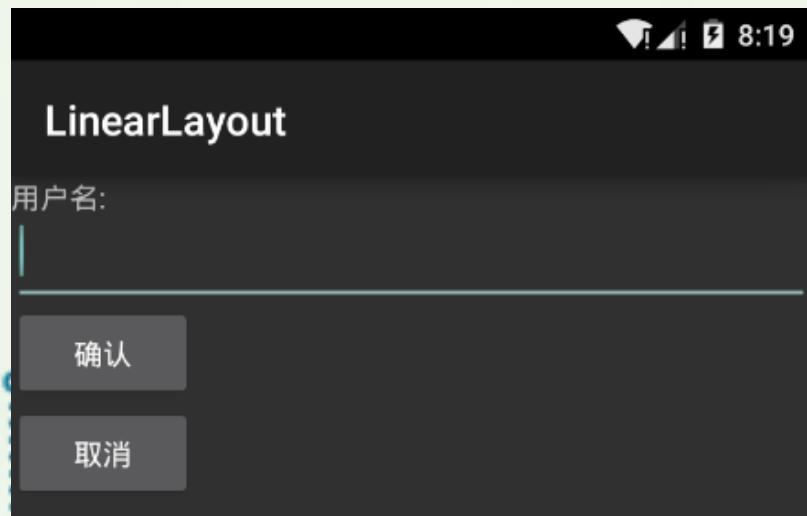
```
12. <Button android:id="@+id/ok"  
13.     android:layout_width="wrap_content"  
14.     android:layout_height="wrap_content"  
15.     android:text="确认">  
16. </Button>  
17. <Button android:id="@+id/cancel"  
18.     android:layout_width="wrap_content"  
19.     android:layout_height="wrap_content"  
20.     android:text="取消" >  
21. </Button>  
22. </LinearLayout>
```



5.3 界面布局

• 5.3.1 线性布局

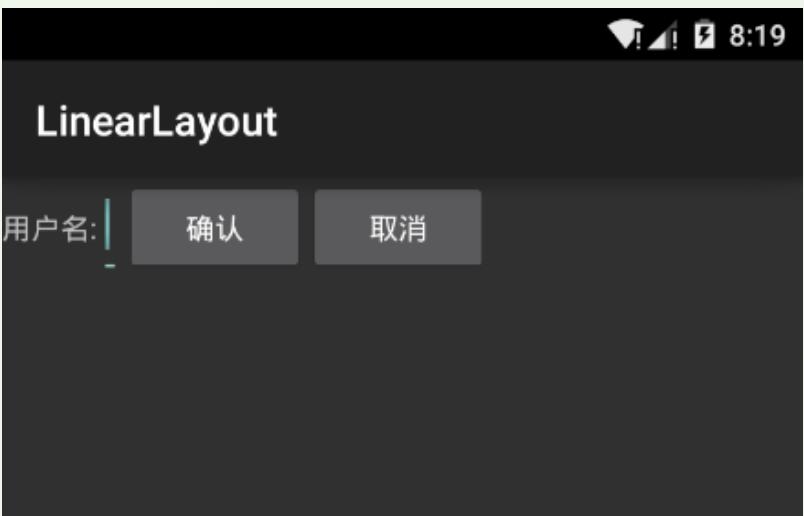
- 将LinearLayout.java文件中的setContentView(R.layout.main), 更改为setContentView(R.layout.main_vertical)。运行后的结果如图



5.3 界面布局

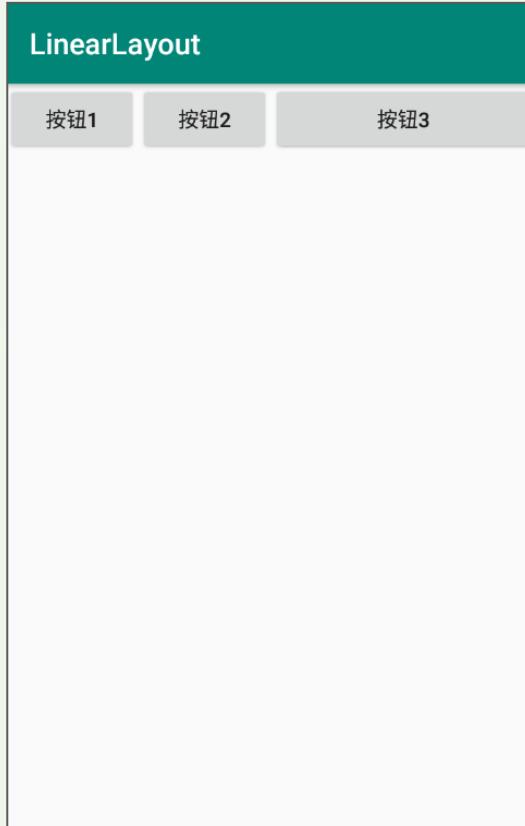
• 5.3.1 线性布局

- 横向线性布局
 - 建立main_horizontal.xml文件
 - 线性布局的Orientation属性的值设置为horizontal
 - 将EditText的Layout width 属性的值设置为 wrap_content
 - 将LinearLayout.java文件中的 setContentView(R.layout.main_vertical) 修改为 setContentView(R.layout.main_horizontal)



// 线性布局LinearLayout

接下来，我们通过一个案例来演示如何使用`android:layout_weight`属性为`LinearLayout`中的控件分配权重。本案例中使用了线性布局`LinearLayout`，在线性布局中放置了3个按钮，这3个按钮的宽度在水平方向的比重是1:1:2，线性布局界面的效果如下图所示。



// 案例步骤

STEP 01

创建程序

创建一个名为LinearLayout的应用程序，指定包名为
com.<yourname>.linearlayout。





// 案例步骤

STEP 02

放置界面控件

在activity_main.xml文件的LinearLayout布局中放置3个Button控件，分别用于显示按钮1、按钮2和按钮3。

```
<LinearLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    .....  
    android: orientation = "vertical">  
        <Button  
            android:layout_width="0dp"  
            android:layout_height="wrap_content"  
            android:layout_weight="1"  
            android:text="按钮1"/>  
        .....  
    </LinearLayout>
```

// 线性布局LinearLayout



注意

LinearLayout布局中的`android:layout_width`属性值不可设为`wrap_content`。这是因为LinearLayout的优先级比Button高，如果设置为`wrap_content`，则Button控件的`android:layout_weight`属性会失去作用。

```
<Button  
    android:layout_width="0dp"  
    android:layout_height="wrap_content"  
    android: layout_weight ="2"/>
```

注意：当控件使用权重属性时，布局宽度属性值通常设置为0dp。

// 实战

□ 1. 添加以下布局结构

LinearLayout (horizontal)

button_save "保存"

button_cancel "取消"

□ 2. 尝试不同的设置，观察UI的效果

case	控件	Layout_width	Layout_weight
1	button_save	wrap_content	1
	button_cancel	wrap_content	2
2	button_save	match_parent	1
	button_cancel	match_parent	2

	layout_width	layout_height	gravity	layout_gravity
layout_1	wrap_content	wrap_content		
layout_2	wrap_content	wrap_content	center	
button_save				center

// 请回答

1. 添加以下 布局结构

Component Tree	
└ LinearLayout (vertical)	
└ layout_1 (horizontal)	
├ textView5 "TextView"	
└ editTextTextPersonName "Name"	
└ layout_2 (horizontal)	
└ button_save "保存"	
└ button_cancel "取消"	

	layout_width	layout_height	gravity	layout_gravity
layout_1	wrap_content	wrap_content		
layout_2	match_parent	match_parent	center	
button_save				center

	layout_width	layout_height	gravity	layout_gravity
layout_1	wrap_content	wrap_content		
layout_2	match_parent	match_parent	start	
button_save				center

// 实战演练—仿动物连连看游戏界面

为了让大家更好地理解线性布局在实际开发中的应用，接下来通过一个**仿动物连连看游戏界面**的案例来演示如何使用线性布局来排列界面上的动物和空格子，界面效果如下图所示。

1

创建程序：

① 创建一个名为AnimalConnection的应用程序。

② 指定包名为com.<yourname>.animalconnection。

2

导入界面图片：导入界面需要的图片到drawable-hdpi文件夹中

3

创建图片控件样式：创建名为btnStyle的样式。

① 放置3个LinearLayout布局文件。

② 每个布局中放置4个Button控件。

4

放置界面控件：③ 每个布局中的控件都是水平排列





5.3 界面布局

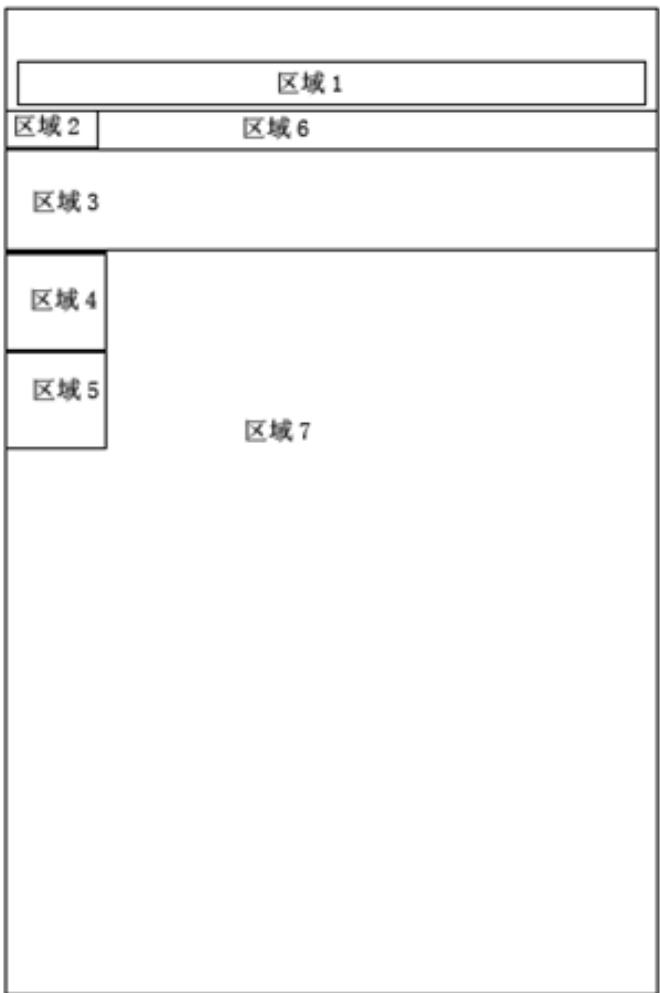
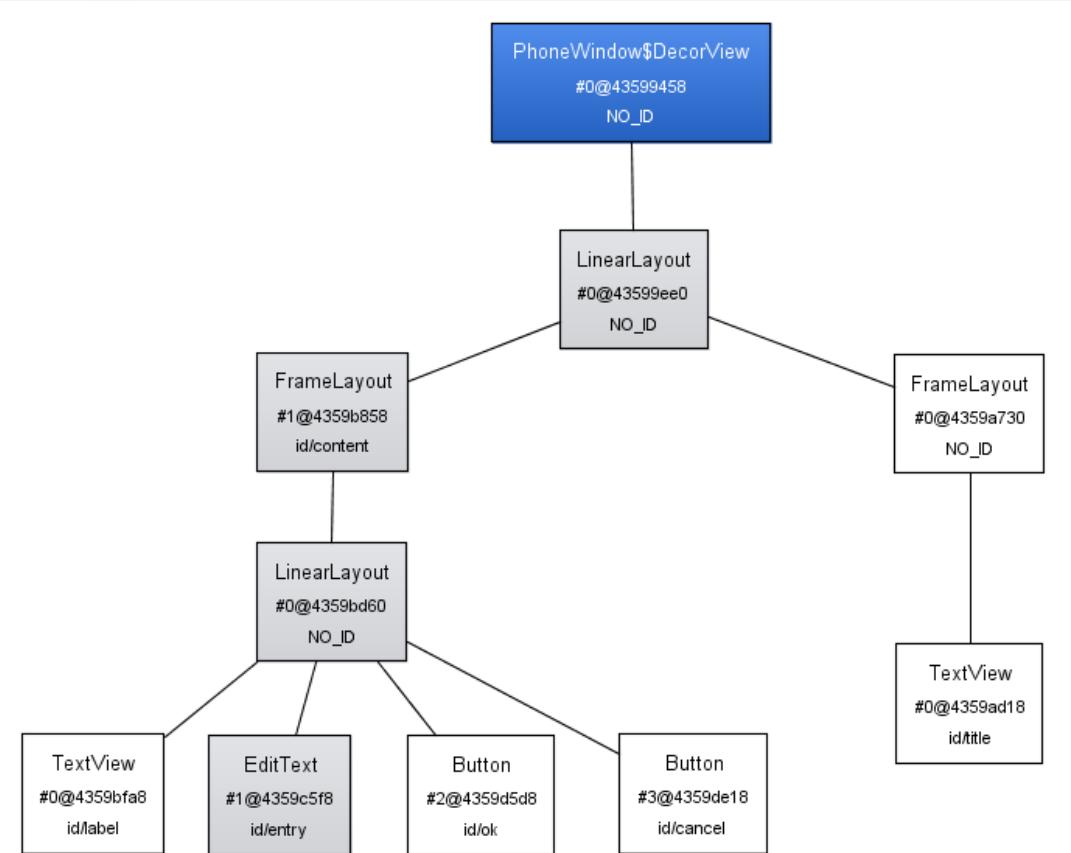
• 5.3.2 框架布局

- 框架布局（FrameLayout）是最简单的界面布局，是用来存放一个元素的空白空间，且子元素的位置是不能够指定的，只能够放置在空白空间的左上角
- 如果有多个子元素，后放置的子元素将遮挡先放置的子元素
- 使用Android SDK中提供的层级观察器（Hierarchy Viewer）进一步分析界面布局

5.3 界面布局

• 5.3.2 框架布局

- 树形结构图和界面示意图





5.3 界面布局

• 5.3.2 框架布局

- 结合界面布局的树形结构图和示意图，分析不同界面布局和界面控件的区域边界
 - 用户界面的根节点 (#0@43599ee0) 是线性布局，其边界是整个界面，也就是示意图的最外层的实心线
 - 根节点右侧的子节点 (#0@43599a730) 是框架布局，仅有一个节点元素 (#0@4359ad18)，这个子元素是 TextView 控件，用来显示 Android 应用程序名称，其边界是示意图中的区域1。因此框架布局元素 #0@43599a730 的边界是同区域1的高度相同，宽带充满整个根节点的区域。这两个界面元素是系统自动生成的，一般情况下用户不能够修改和编辑
 - 根节点左侧的子节点 (#1@4359b858) 也是框架布局，边界是区域2到区域7的全部空间



5.3 界面布局

• 5.3.2 框架布局

- 子节点(#1@4359b858)下仅有一个子节点(#0@4359bd60)元素是线性布局，因为线性布局的Layout width属性设置为match_parent, Layout height属性设置为wrap_content, 因此该线性布局的宽度就是其父节点 #1@4359b858的宽带，高度等于所有子节点元素的高度之和
- 线性布局#0@4359bd60四个子节点元素#0@4359bfa8、#1@4359c5f8、#2@4359d5d8和#3@4359de18的边界，分别是界面布局示意图中的区域2、区域3、区域4和区域5



5.3 界面布局

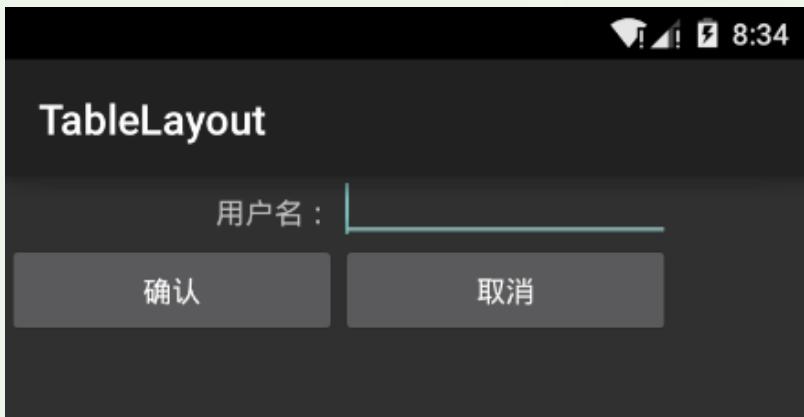
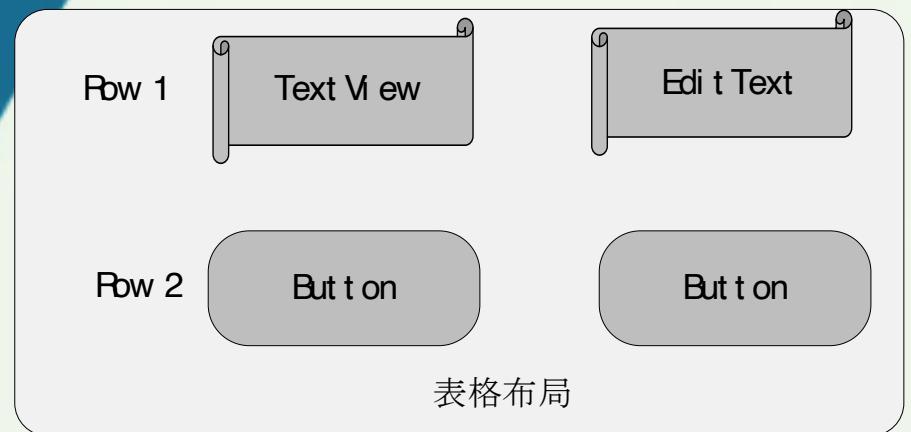
• 5.3.3 表格布局

- 表格布局 (TableLayout) 是一种常用的界面布局，通过指定行和列将界面元素添加到表格中
 - 网格的边界对用户是不可见的
 - 表格布局支持嵌套
 - 可以将表格布局放置在表格布局的表格中
 - 可以在表格布局中添加其他界面布局，例如线性布局、相对布局等

5.3 界面布局

• 5.3.3 表格布局

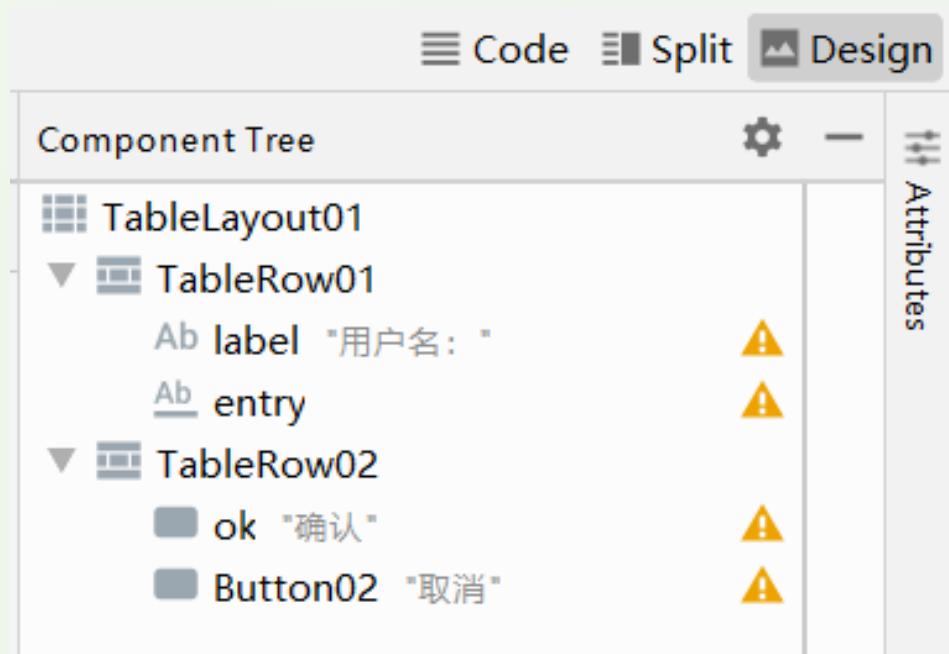
- 表格布局示意图及效果图



5.3 界面布局

• 5.3.3 表格布局

- 建立表格布局要注意以下几点
 - 在界面可视化编辑器上，向TableRow01中拖拽TextView和EditText





5.3 界面布局

• 5.3.3 表格布局

- 建立表格布局要注意以下几点
 - 在界面可视化编辑器上，再向TableRow02中拖拽两个Button
 - 参考表5.2设置TableRow中四个界面控件的属性值

编号	类型	属性	值
1	TextView	Id	@+id/label
		Text	用户名:
		Gravity	right
		Padding	3dip
		Layout width	160dip
2	EditText	Id	@+id/entry
		Text	[null]
		Padding	3dip
		Layout width	160dip
3	Button	Id	@+id/ok
		Text	确认
		Padding	3dip
4	Button	Id	@+id/cancel
		Text	取消
		Padding	3dip



5.3 界面布局

■ 5.3.3 表格布局

□ 建立表格布局main.xml文件的完整代码如下：

```
1. <?xml version="1.0" encoding="utf-8"?>
2.
3. <TableLayout android:id="@+id/TableLayout01"
4.     android:layout_width="match_parent"
5.     android:layout_height="match_parent"
6.     xmlns:android="http://schemas.android.com/apk/res/android">
7.     <TableRow android:id="@+id/TableRow01"
8.         android:layout_width="wrap_content"
9.         android:layout_height="wrap_content">
10.        <TextView android:id="@+id/label"
11.            android:layout_height="wrap_content"
12.            android:layout_width="160dip"
13.            android:gravity="right"
14.            android:text="用户名："
15.            android:padding="3dip" >
16.        </TextView>
```



5.3 界面布局

```
17. <EditText android:id="@+id/entry"
18.     android:layout_height="wrap_content"
19.     android:layout_width="160dip"
20.     android:padding="3dip" >
21. </EditText>
22. </TableRow>
23. <TableRow android:id="@+id/TableRow02"
24.     android:layout_width="wrap_content"
25.     android:layout_height="wrap_content">
26.     <Button android:id="@+id/ok"
27.         android:layout_height="wrap_content"
28.         android:padding="3dip"
29.         android:text="确认">
30.     </Button>
31.     <Button android:id="@+id/Button02"
32.         android:layout_width="wrap_content"
33.         android:layout_height="wrap_content"
34.         android:padding="3dip"
35.         android:text="取消">
36.     </Button>
37. </TableRow>
38. </TableLayout>
```



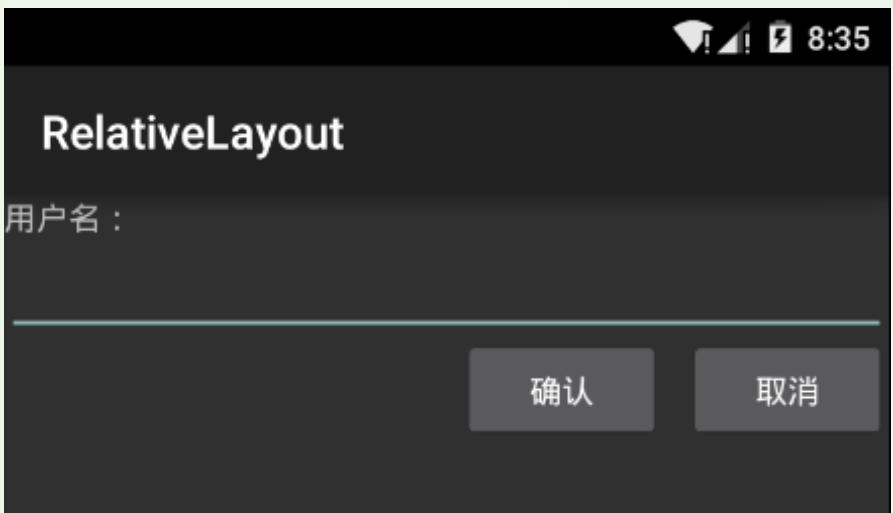
5.3 界面布局

- 第3行代码使用了<TableLayout>标签声明表格布局
- 第7行和第23行代码声明了两个TableRow元素
- 第12行设定宽度属性android:layout_width : 160dip
- 第13行设定属性android:gravity, 指定文字为右对齐
- 第15行使用属性android:padding, 声明TextView元素与其他元素的间隔距离为3dip

5.3 界面布局

• 5.3.4 相对布局

- 相对布局(RelativeLayout)是一种非常灵活的布局方式，能够通过指定界面元素与其他元素的相对位置关系，确定界面中所有元素的布局位置
- 特点：能够最大程度保证在各种屏幕尺寸的手机上正确显示界面布局

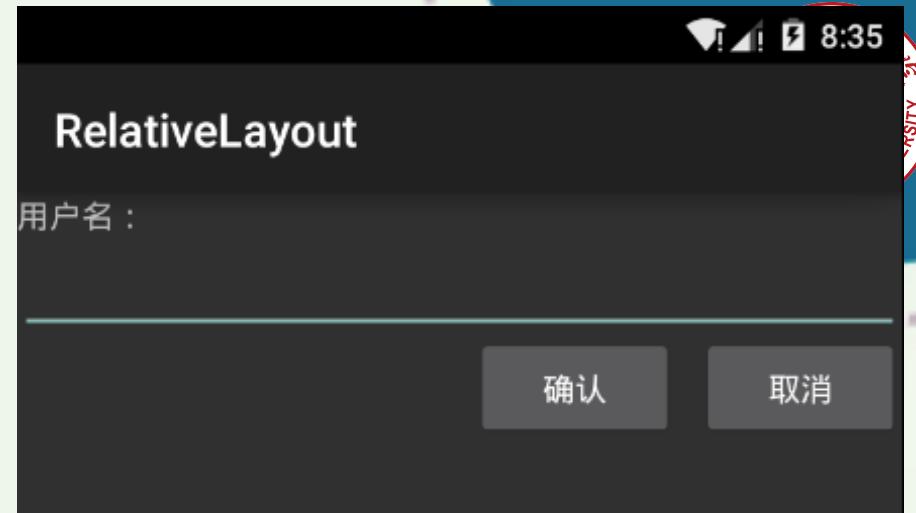


5.3 界面布局

• 5.3.4 相对布局

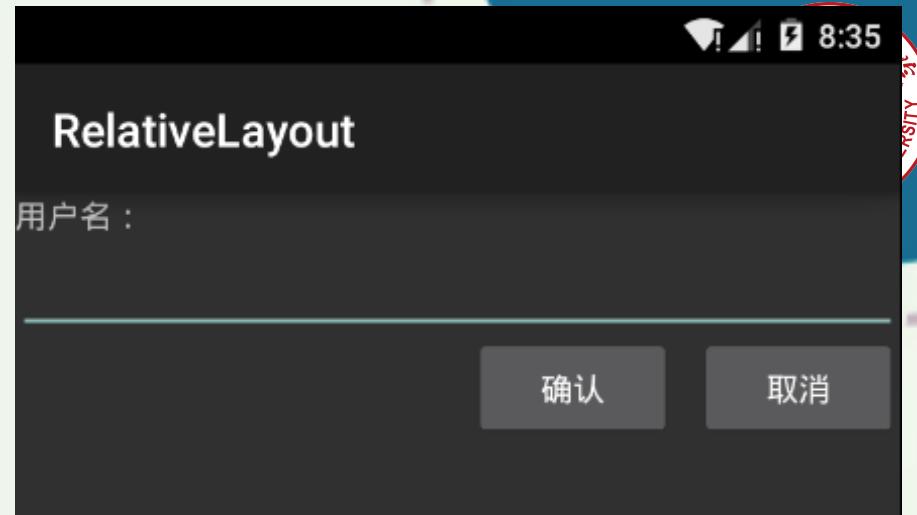
- 相对布局在main.xml文件的完整代码

```
1. <?xml version="1.0" encoding="utf-8"?>
2.
3. <RelativeLayout android:id="@+id/RelativeLayout01"
4.     android:layout_width="match_parent"
5.     android:layout_height="match_parent"
6.     xmlns:android="http://schemas.android.com/apk/res/android">
7.     <TextView android:id="@+id/label"
8.         android:layout_height="wrap_content"
9.         android:layout_width="match_parent"
10.        android:text="用户名: ">
11.    </TextView>
12.    <EditText android:id="@+id/entry"
13.        android:layout_height="wrap_content"
14.        android:layout_width="match_parent"
15.        android:layout_below="@+id/label">
16.    </EditText>
```



5.3 界面布局

8:35



```
17. <Button android:id="@+id/cancel"  
18.         android:layout_height="wrap_content"  
19.         android:layout_width="wrap_content"  
20.         android:layout_alignParentRight="true"  
21.         android:layout_marginLeft="10dip"  
22.         android:layout_below="@+id/entry"  
23.         android:text="取消" >  
24.     </Button>  
25.     <Button android:id="@+id/ok"  
26.         android:layout_height="wrap_content"  
27.         android:layout_width="wrap_content"  
28.         android:layout_toLeftOf="@+id/cancel"  
29.         android:layout_alignTop="@+id/cancel"  
30.         android:text="确认" >  
31.     </Button>  
32. </RelativeLayout>
```



5.3 界面布局

• 5.3.4 相对布局

- 第3行使用了<RelativeLayout>标签声明一个相对布局
- 第15行使用位置属性android:layout_below, 确定EditText控件在ID为label的元素下方
- 第20行使用属性android:layout_alignParentRight, 声明该元素在其父元素的右边边界对齐
- 第21行设定属性android:layout_marginLeft, 左移10dip
- 第22行声明该元素在ID为entry的元素下方
- 第28行声明使用属性android:layout_toLeftOf, 声明该元素在ID为cancel元素的左边
- 第29行使用属性android:layout_alignTop, 声明该元素与ID为cancel的元素在相同的水平位置



5.3 界面布局

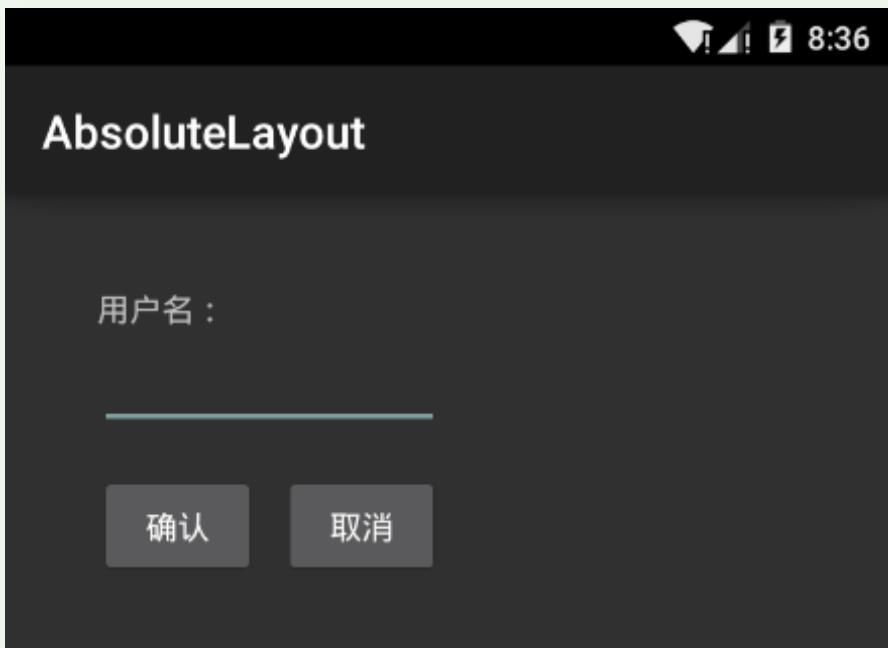
• 5.3.5 绝对布局

- 绝对布局 (AbsoluteLayout) 能通过指定界面元素的坐标位置，来确定用户界面的整体布局
- 绝对布局是一种不推荐使用的界面布局，因为通过X轴和Y轴确定界面元素位置后，Android系统不能够根据不同屏幕对界面元素的位置进行调整，降低了界面布局对不同类型和尺寸屏幕的适应能力

5.3 界面布局

• 5.3.5 绝对布局

- 每一个界面控件都必须指定坐标 (X, Y) , 例如 “确认” 按钮坐标是 (40, 120) , “取消” 按钮的坐标是 (120, 120) 。坐标原点 (0, 0) 在屏幕的左上角

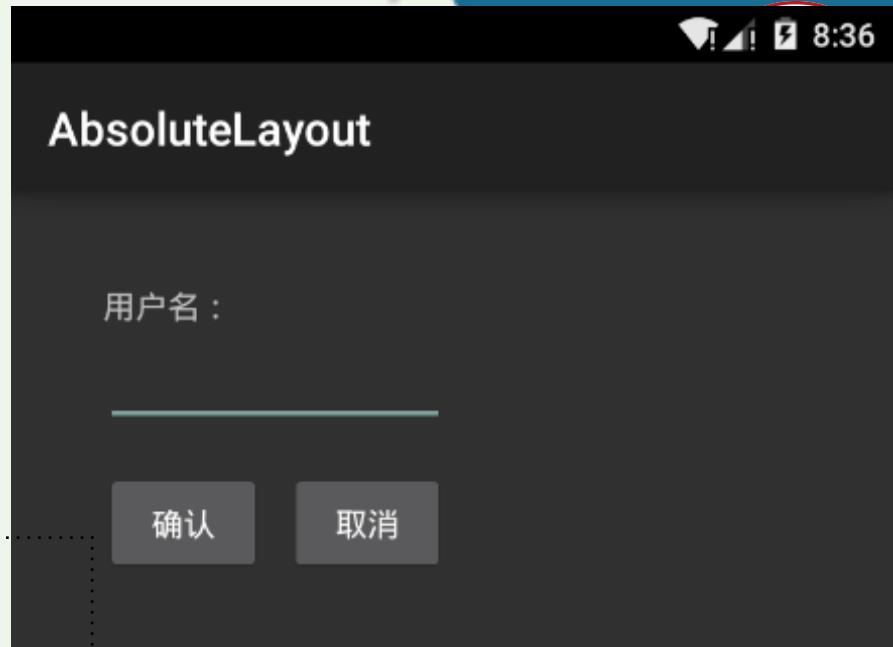


5.3 界面布局

• 5.3.5 绝对布局

- 绝对布局示例的main.xml文件完整代码：

```
1. <?xml version="1.0" encoding="utf-8"?>
2.
3. <AbsoluteLayout android:id="@+id/AbsoluteLayout01"
4.     android:layout_width="match_parent"
5.     android:layout_height="match_parent"
6.     xmlns:android="http://schemas.android.com/apk/res/android">
7.     <TextView android:id="@+id/label"
8.         android:layout_x="40dip"
9.         android:layout_y="40dip"
10.        android:layout_height="wrap_content"
11.        android:layout_width="wrap_content"
12.        android:text="用户名: ">
13.     </TextView>
14.     <EditText android:id="@+id/entry"
15.         android:layout_x="40dip"
16.         android:layout_y="60dip"
```



5.3 界面布局

• 5.3.5 绝对布局

- 绝对布局示例的main.xml文件完整代码：

```
17.     android:layout_height="wrap_content"
18.     android:layout_width="150dip">
19. </EditText>
20. <Button android:id="@+id/ok"
21.         android:layout_width="70dip"
22.         android:layout_height="wrap_content"
23.         android:layout_x="40dip"
24.         android:layout_y="120dip"
25.         android:text="确认">
26. </Button>
27. <Button android:id="@+id/cancel"
28.         android:layout_width="70dip"
29.         android:layout_height="wrap_content"
30.         android:layout_x="120dip"
31.         android:layout_y="120dip"
32.         android:text="取消">
33. </Button>
34. </AbsoluteLayout>
```

AbsoluteLayout

用户名：

确认

取消



5.3 界面布局

• 5.3.5 网格布局

- 网格布局 (GridLayout)
 - Android SDK 4.0新支持的布局方式
 - 将用户界面划分为网格，界面元素可随意摆放在网格中
 - 网格布局比表格布局 (TableLayout) 在界面设计上更加灵活，在网格布局中界面元素可以占用多个网格的，而在表格布局却无法实现，只能将界面元素指定在一个表格行 (TableRow) 中，不能跨越多个表格行。
- GridLayoutDemo示例说明网格布局的使用方法
 - 在Android Studio界面设计器中的界面图示和在Android模拟器运行后的用户界面

5.3 界面布局

• GridLayoutDemo示例

- 界面设计器图示及模拟器运行结果





5.3 界面布局

• 5.3.5 网格布局

- GridLayoutDemo示例
 - 界面设计器中可以看到虚线网格，但在模拟器的运行结果中是看不到的
 - 网格布局将界面划分成多个块，这些块是根据界面元素动态划分的
 - 具体的讲，GridLayoutDemo示例的左边第一列的宽度，是综合分析在第一列中的两个界面元素“用户名”和“密码”TextView的宽度而进行设定的，选择两个元素中最宽元素的宽度，作为第一列的宽度
 - 最上方第一行的高度，也是分析“这是关于GridLayout的示例”这个TextView元素的高度进行设定的。因此，网格布局中行的高度和列的宽度，完全取决于本行或本列中，高度最高或宽对最宽的界面元素

5.3 界面布局

• 5.3.5 网格布局

- main.xml文件的全部代码





5.3 界面布局

• 5.3.5 网格布局

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <GridLayout
3.     xmlns:android="http://schemas.android.com/apk/res/android"
4.
5.     android:layout_width="match_parent"
6.     android:layout_height="match_parent"
7.     android:useDefaultMargins="true"
8.     android:columnCount="4" >
9.     <TextView
10.        android:layout_columnSpan="4"
11.        android:layout_gravity="center_horizontal"
12.        android:text="这是关于GridLayout的示例"
13.        android:textSize="20dip" />
14.
15.     <TextView
16.        android:text="用户名: "
17.        android:layout_gravity="right" />
18.     <EditText
19.        android:ems="8"
20.        android:layout_columnSpan="2"/>
21.     <TextView
22.        android:text="密码: "
23.        android:layout_column="0"
24.        android:layout_gravity="right"/>
25.     <EditText
26.        android:ems="8"
27.        android:layout_columnSpan="2" />
28.     <Button
29.        android:text="清空输入"
30.        android:layout_column="1"
31.        android:layout_gravity="fill_horizontal"/>
32.     <Button
33.        android:text="下一步"
34.        android:layout_column="2"
35.        android:layout_gravity="fill_horizontal"/>
36. </GridLayout>
```



5.3 界面布局

• 5.3.5 网格布局

- 代码第7行的useDefaultMargins表示网格布局中的所有元素都遵循缺省的边缘规则，就是说所有元素之间都会留有一定的边界空间。代码第8行的columnCount表示纵向分为4列，从第0列到第3列，程序开发人员也可以在这里定义横向的行数，使用rowCount属性。
- 代码第10行的layout_columnSpan属性表示TeixtView控件所占据的列的数量。代码第12行的layout_gravity = center_horizontal表示文字内容在所占据的块中居中显示
- 代码第9行到第13行定义了第一个界面控件，虽然定义了纵向所占据的块的数量，但却没有定义元素起始位置所在的块，原因是网格布局中第1个元素默认在第0行第0列
- 代码第16行到第18行定义了第2个界面控件，仍然没有定义元素起始位置所在的块。根据网格布局界面元素的排布规则，如果没有明确说明元素所在的块，那么当前元素会放置在前一个元素的同一行右侧的块上；如果前一个元素已经是这一行的末尾块，则当前元素放置在下一行的第一个块上；如果当前元素在纵向上占据多个块，而前一个元素右侧没有足够数量的块，则当前元素的起始位置也会放置在下一行的第一个块上



5.3 界面布局

• 5.3.5 网格布局

- 代码第7行的useDefaultMargins表示网格布局中的所有元素都遵循缺省的边缘规则，就是说所有元素之间都会留有一定的边界空间。代码第8行的columnCount表示纵向分为4列，从第0列到第3列，程序开发人员也可以在这里定义横向的行数，使用rowCount属性。
- 代码第10行的layout_columnSpan属性表示TeixtView控件所占据的列的数量。代码第12行的layout_gravity = center_horizontal表示文字内容在所占据的块中居中显示
- 代码第9行到第13行定义了第一个界面控件，虽然定义了纵向所占据的块的数量，但却没有定义元素起始位置所在的块，原因是网格布局中第1个元素默认在第0行第0列
- 代码第16行到第18行定义了第2个界面控件，仍然没有定义元素起始位置所在的块。根据网格布局界面元素的排布规则，如果没有明确说明元素所在的块，那么当前元素会放置在前一个元素的同一行右侧的块上；如果前一个元素已经是这一行的末尾块，则当前元素放置在下一行的第一个块上；如果当前元素在纵向上占据多个块，而前一个元素右侧没有足够数量的块，则当前元素的起始位置也会放置在下一行的第一个块上

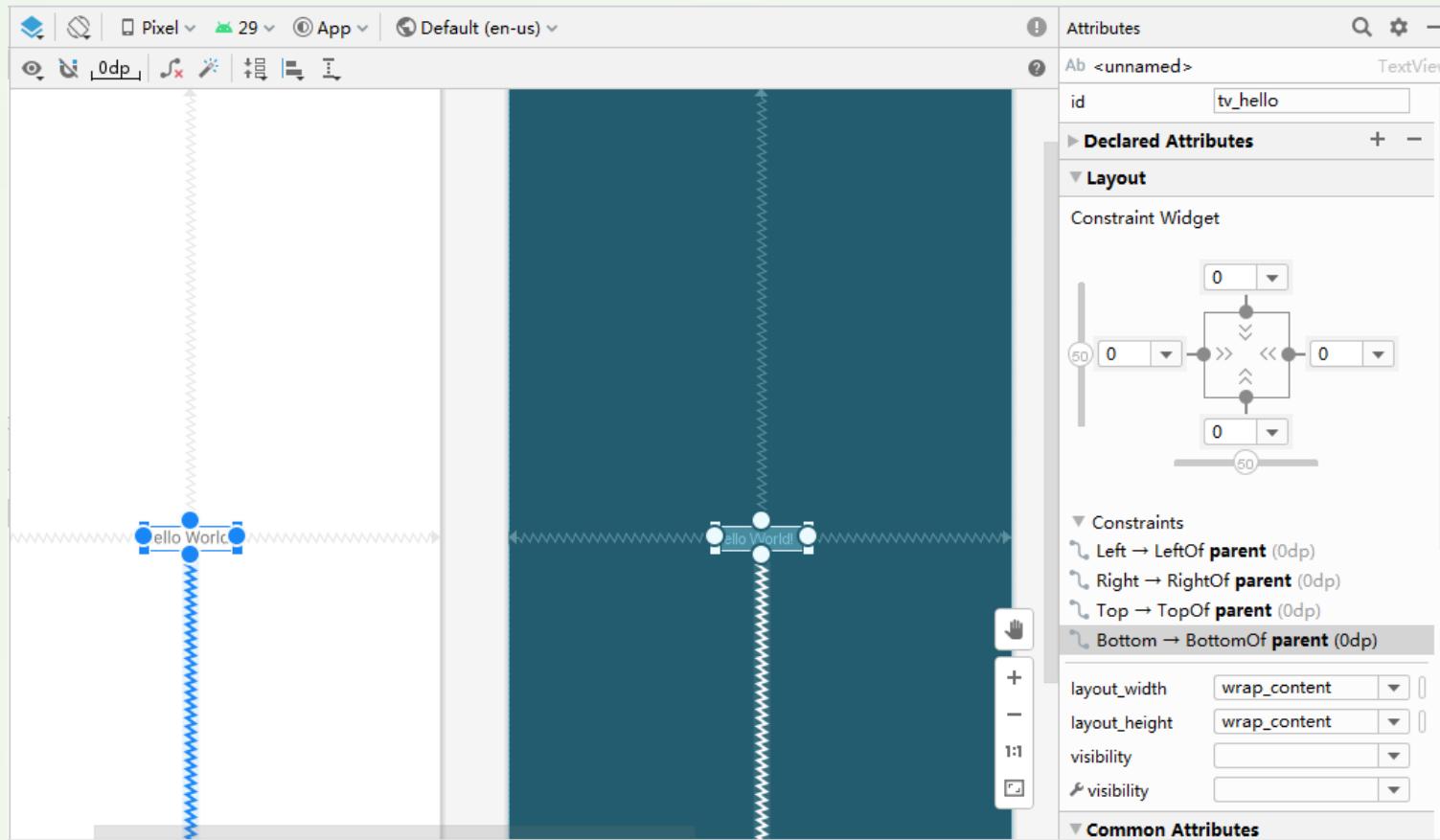


3.3.2 约束布局



ConstraintLayout编辑器

ConstraintLayout编辑器提供设计预览和蓝图功能，如图所示。



ConstraintLayout编辑器



3.3.2 约束布局



手动创建约束

开发人员可以通过拖曳约束手柄来为布局手动创建约束。首先认识约束手柄。
ConstraintLayout编辑器为布局中的控件添加了一些约束手柄，如图所示。



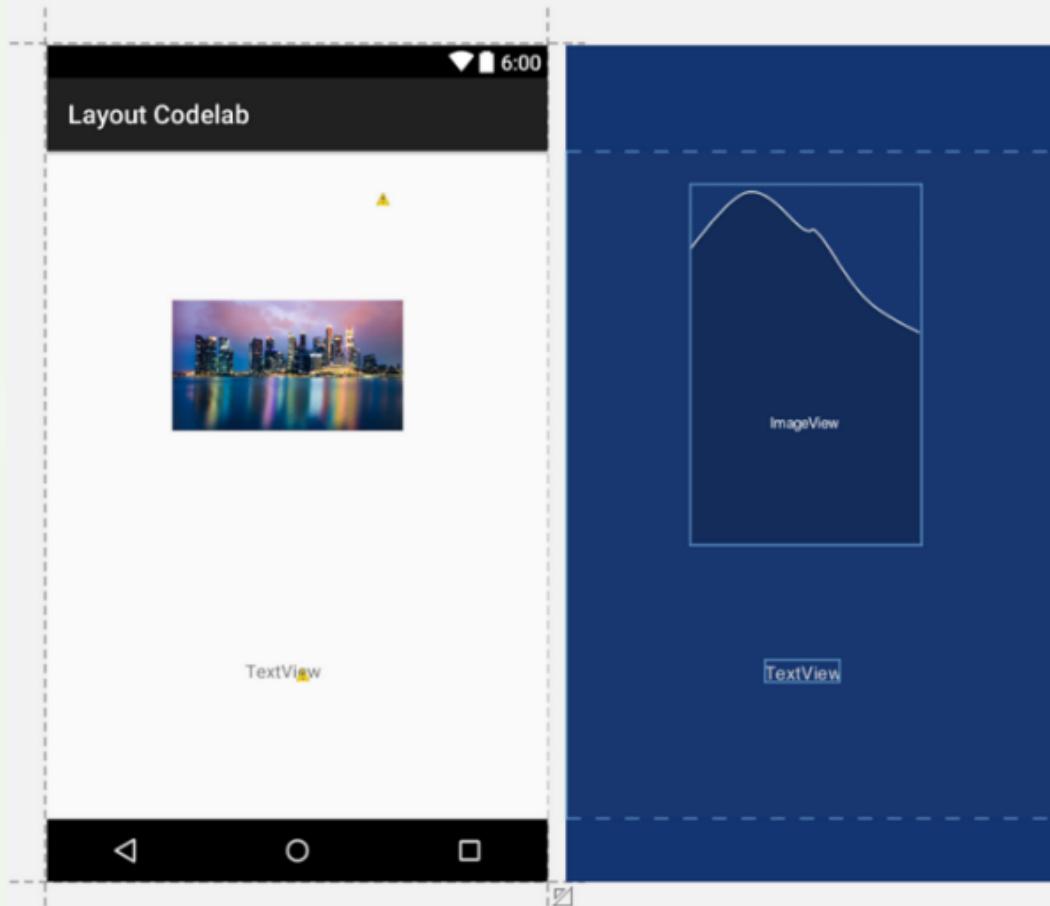
控件的约束手柄



3.3.2 约束布局



下面以一个示例演示在ConstraintLayout编辑器中创建约束的方法。在容器内添加一个 ImageView 和一个 TextView，设计效果如图所示。

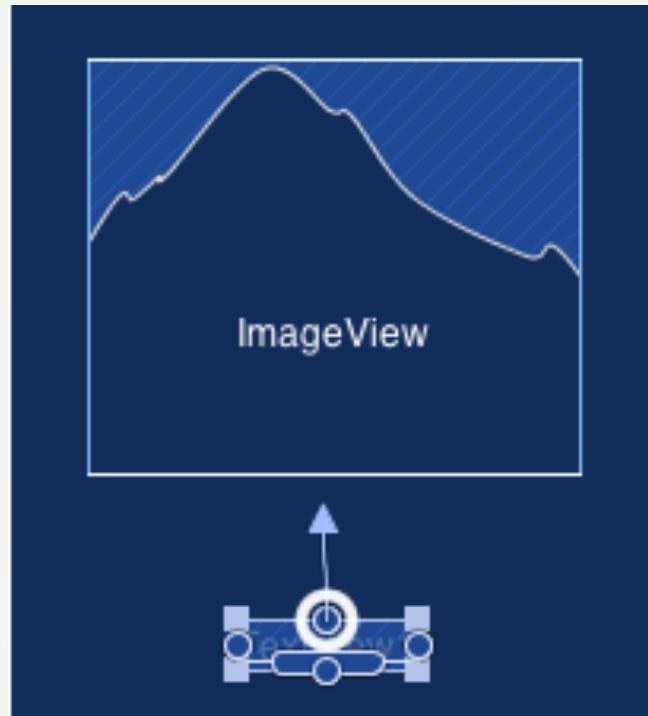


在容器内添加ImageView和TextView的设计效果



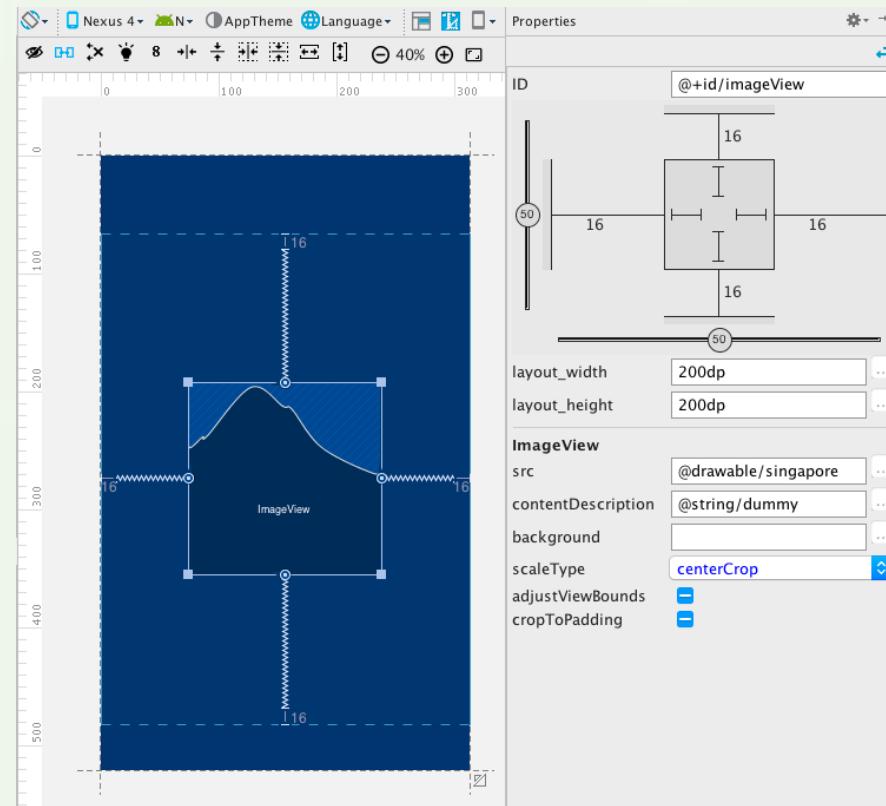
3.3.2 约束布局

- 第一步，创建ImageView和TextView的约束，使TextView显示在ImageView的下方。



创建TextView和ImageView的约束

- 第二步，创建ImageView与容器的约束。



Inspector编辑区域



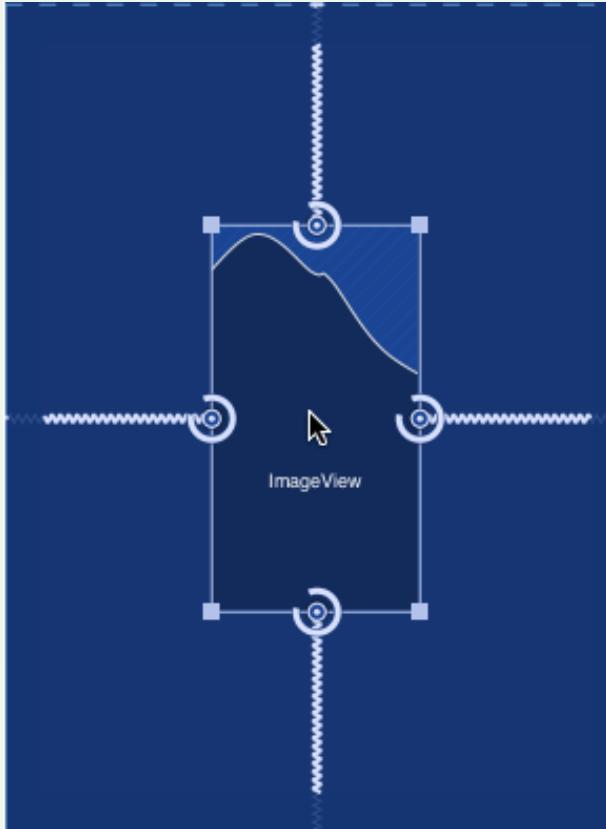


3.3.2 约束布局

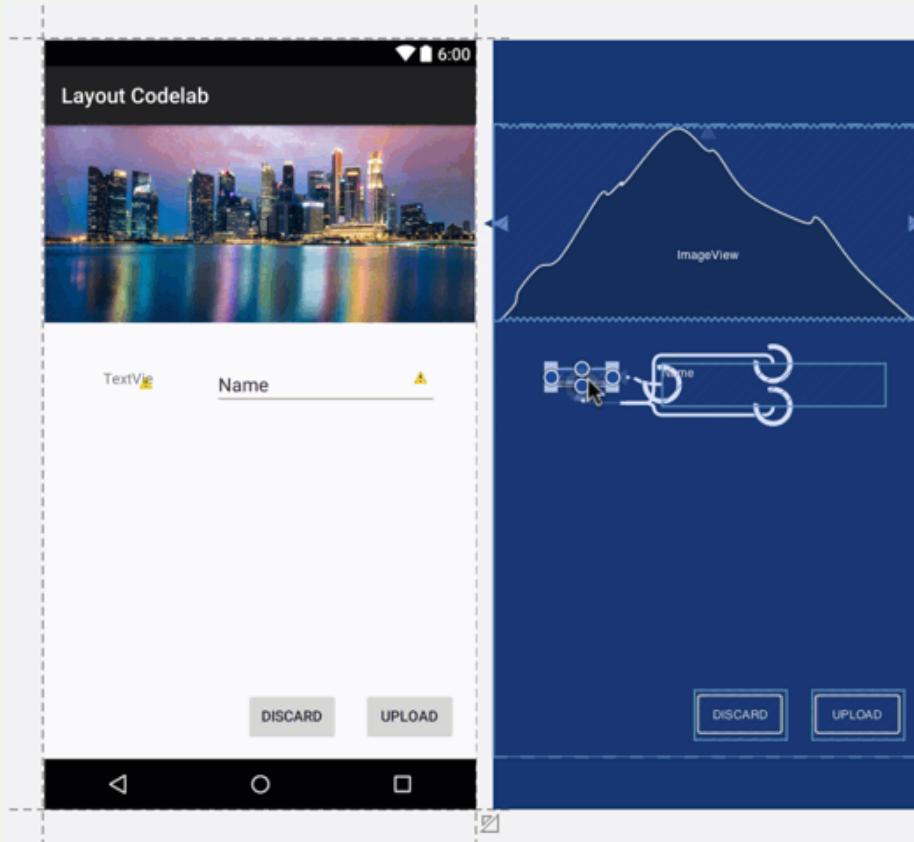


自动创建约束

下面的示例演示了自动创建约束的方法。



自动为ImageView创建与容器的约束



自动创建约束效果



任务3.2 设计App “我” 界面



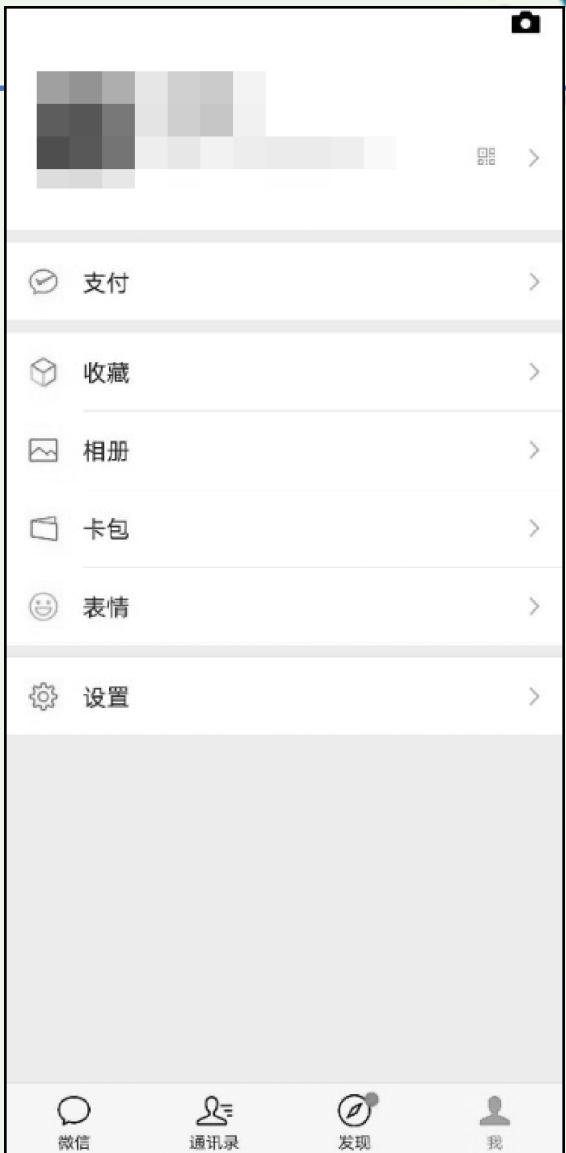
自动创建约束

1. 任务描述

参照微信“我”界面，使用常用控件、约束布局、线性布局，实现App“我”界面。

2. 运行结果

本任务运行结果如图所示。

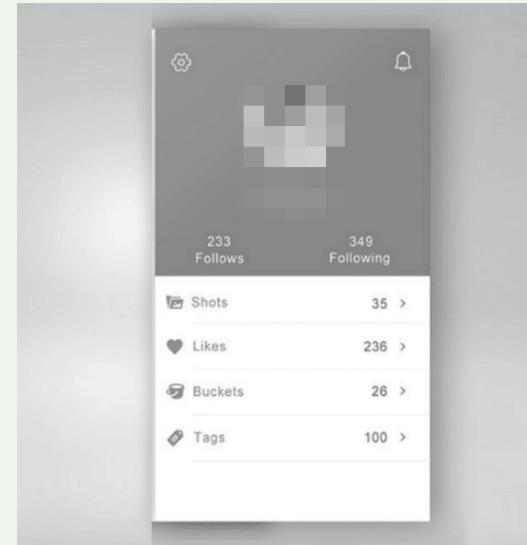


运行结果



动手实践

使用约束布局设计一个如图所示的个人信息展示界面（不要求实现界面的交互功能）。



个人信息展示界面