第5章 Android用户界面

学习目标

AIMS

01 了解各种界面控件的使用方法

02 掌握各种界面布局的特点和使用方法

03 掌握选项菜单、子菜单和快捷菜单的使用方法

04 掌握操作栏和Fragment的使用方法

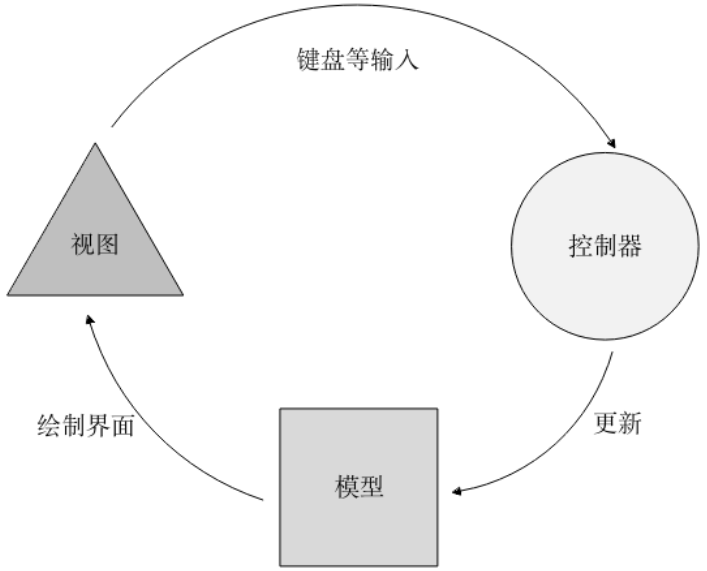
05 掌握按键事件和触摸事件的处理方法

5.1 用户界面基础

- 用户界面（User Interface，UI)是系统和用户之间进行信息交换的媒介，实现信息的内部形式与人类可以接受形式之间的转换
  - 在计算机出现早期，批处理界面（1945-1968）和命令行界面（1969-1983）得到广泛的使用
  - 目前，流行图像用户界面（Graphical User Interface，GUI），采用图形方式与用户进行交互的界面
  - 未来的用户界面将更多的运用虚拟现实技术，使用户能够摆脱键盘与鼠标的交互方式，而通过动作、语言，甚至是脑电波来控制计算机

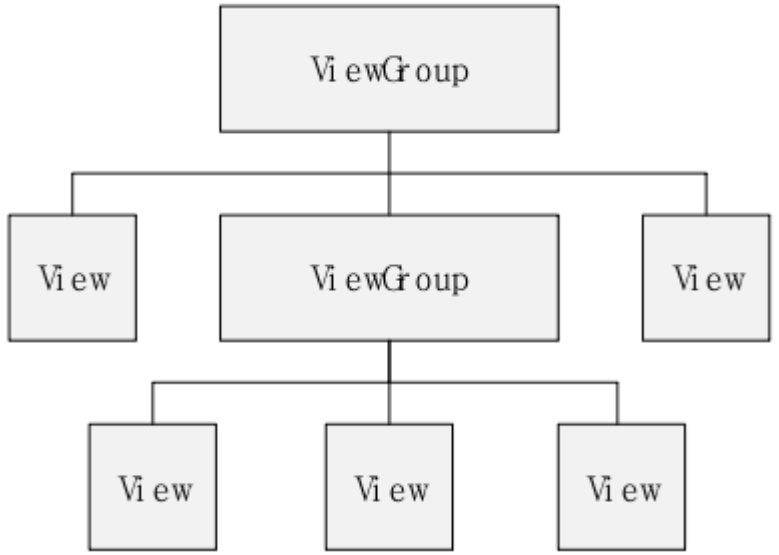
5.1 用户界面基础

- **Android用户界面框架**
  - Android用户界面框架采用MVC(Model-View-Controller)模型
    - 控制器（Controller）处理用户输入
    - 视图（View）显示用户界面和图像
    - 模型（Model）保存数据和代码



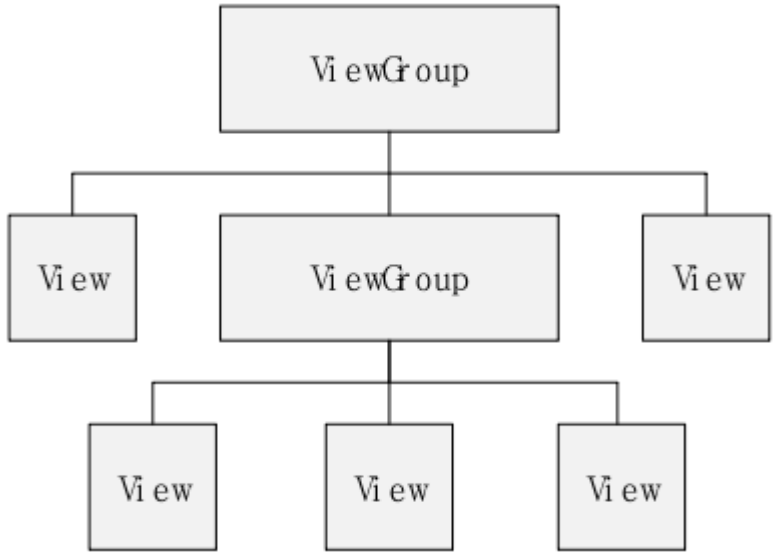
5.1 用户界面基础

- **Android用户界面框架**
  - Android用户界面框架采用视图树（View Tree）模型
    - 由View和ViewGroup构成
    - View是最基本的可视单元
      - 存储了屏幕上特定矩形区域内所显示内容的数据结构
      - 实现所占据区域的界面绘制、焦点变化、用户输入和界面事件处理等
      - 一个重要的基类，所有在界面上的可见元素都是View的子类
  - ViewGroup是一种能够承载含多个View的显示单元
    - 承载界面布局
    - 承载具有原子特性的重构模块



5.1 用户界面基础

- **Android用户界面框架**
  - Android用户界面框架采用视图树（View Tree）模型
    - Android系统会依据视图树的结构从上至下绘制每一个界面元素
    - 每个元素负责对自身的绘制，如果元素包含子元素，该元素会通知其下所有子元素进行绘制



5.1 用户界面基础

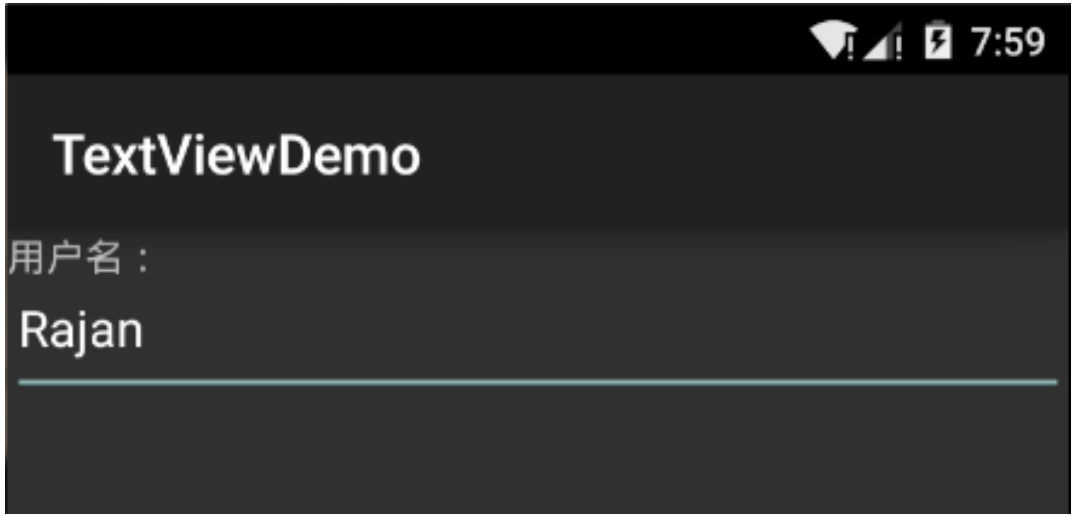
- **Android用户界面框架**
  - 单线程用户界面
    - 控制器从队列中获取事件和视图在屏幕上绘制用户界面，使用的都是同一个线程
    - 特点：处理函数具有顺序性，能够降低应用程序的复杂程度，同时也能降低开发的难度
    - 缺点：如果事件处理函数过于复杂，可能会导致用户界面失去响应

5.2 界面控件

- 常见的系统控件
  - TextView
  - EditText
  - Button
  - ImageButton
  - Checkbox
  - RadioButton
  - Spinner
  - ListView
  - TabHost

5.2 界面控件

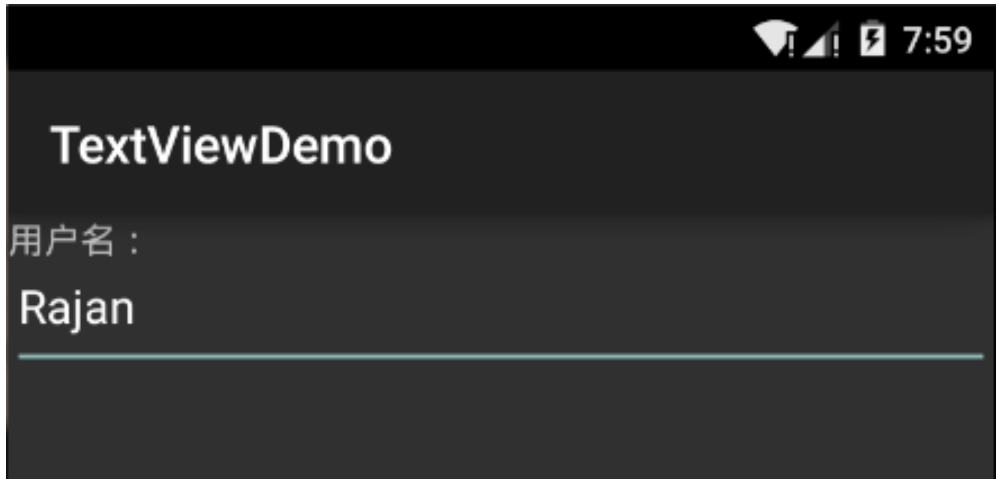
- **5.2.1 TextView和EditText**
  - TextView是一种用于显示字符串的控件
  - EditText则是用来输入和编辑字符串的控件
  - EditText是一个具有编辑功能的TextView



5.2 界面控件

•5.2.1 TextView和EditText

- 建立一个“TextViewDemo”的程序，包含TextView和EditText两个控件
- 上方“用户名”部分使用的是TextView，下方的文字输入框使用的是EditText



5.2 界面控件

- **5.2.1 TextView和EditText**
  - TextViewDemo在XML文件中的代码

```
<TextView android:id="@+id/TextView01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="TextView01" >
</TextView>
<EditText android:id="@+id/EditText01"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="EditText01" >
</EditText>
```

5.2 界面控件

•5.2.1 TextView和EditText

- 第1行android:id属性声明了TextView的ID，这个ID主要用于在代码中引用这个TextView对象
  - “@+id/TextView01”表示所设置的ID值
  - @表示后面的字符串是ID资源
  - 加号（+）表示需要建立新资源名称，并添加到R.java文件中
  - 斜杠后面的字符串（TextView01）表示新资源的名称
  - 如果资源不是新添加的，或属于Android框架的ID资源，则不需要使用加号（+），但必须添加Android包的命名空间，例如android:id="@android:id/empty"

5.2 界面控件

•5.2.1 TextView和EditText

- 第2行的android:layout_width属性用来设置TextView的宽度，wrap_content表示TextView的宽度只要能够包含所显示的字符串即可
- 第3行的android:layout_height属性用来设置TextView的高度
- 第4行表示TextView所显示的字符串，在后面将通过代码更改TextView的显示内容
- 第7行中“fill_parent”表示EditText的宽度将等于父控件的宽度

5.2 界面控件

•5.2.1 TextView和EditText

- TextViewDemo.java文件中代码的修改

```
TextView textView = (TextView)findViewById(R.id.TextView01);
EditText editText = (EditText)findViewById(R.id.EditText01);
textView.setText("用户名： ");
editText.setText("Rajan");
```

- 第1行代码的findViewById()函数能够通过ID引用界面上的任何控件，只要该控件在XML文件中定义过ID即可
- 第3行代码的setText()函数用来设置TextView所显示的内容

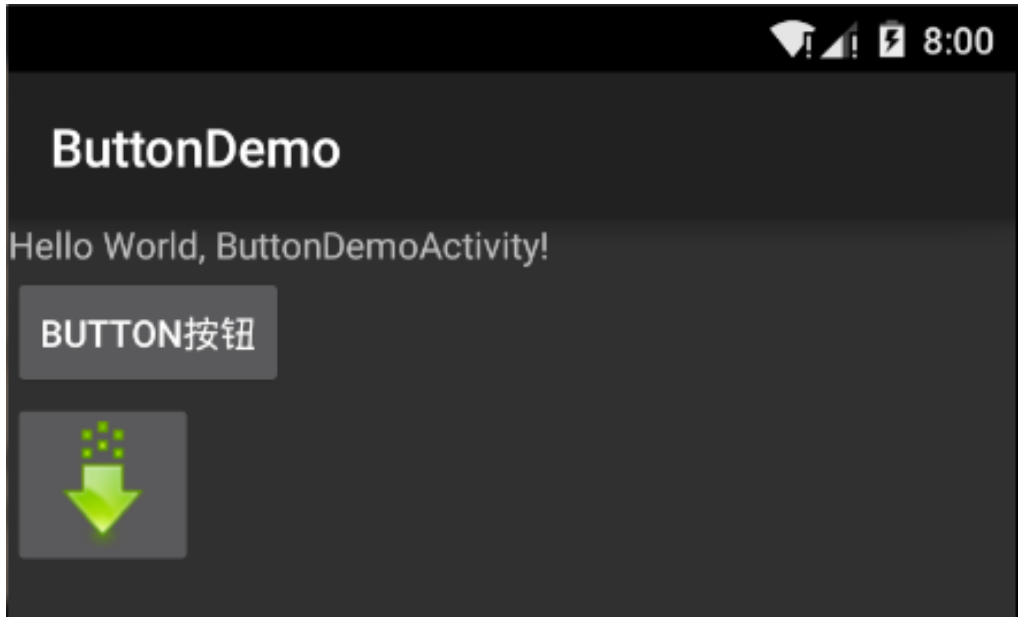
5.2 界面控件

- **5.2.2 Button和ImageButton**
  - Button是一种按钮控件，用户能够在该控件上点击，并后引发相应的事件处理函数
  - ImageButton用以实现能够显示图像功能的控件按钮

5.2 界面控件

•5.2.2 Button和ImageButton

- 建立一个“ButtonDemo”的程序，包含Button和ImageButton两个按钮，上方是“Button按钮”，下方是一个ImageButton控件



5.2 界面控件

•5.2.2 Button和ImageButton

- ButtonDemo在XML文件中的代码

```
1.  <Button android:id="@+id/Button01"
2.  android:layout_width="wrap_content"
3.  android:layout_height="wrap_content"
4.  android:text="Button01" >
5.  </Button>
6.  <ImageButton android:id="@+id/ImageButton01"
7.  android:layout_width="wrap_content"
8.  android:layout_height="wrap_content">
9.  </ImageButton>
```

- 定义Button控件的高度、宽度和内容
- 定义ImageButton控件的高度和宽度，但是没定义显示的图像，在后面的代码中进行定义

5.2 界面控件

- **5.2.2 Button和ImageButton**
  - 引入资源
    - 将download.png文件拷贝到/res/drawable文件夹下

5.2 界面控件

•5.2.2 Button和ImageButton

- 更改Button和ImageButton内容
  - 引入android.widget.Button和android.widget.ImageButton

```
1. Button button = (Button)findViewById(R.id.Button01);
2. ImageButton imageButton =
   (ImageButton)findViewById(R.id.ImageButton01);
3. button.setText("Button按钮");
4. imageButton.setImageResource(R.drawable.download);
```

- 第1行代码用于引用在XML文件中定义的Button控件
- 第2行代码用于引用在XML文件中定义的ImageButton控件
- 第3行代码将Button的显示内容更改为“Button按钮”
- 第4行代码利用setImageResource()函数，将新加入的png文件R.drawable.download传递给ImageButton

5.2 界面控件

•5.2.2 Button和ImageButton

- 按钮响应点击事件：添加点击事件的监听器

```
1.  final TextView textView = (TextView)findViewById(R.id.TextView01);
2.  button.setOnClickListener(new View.OnClickListener() {
3.  public void onClick(View view) {
4.  textView.setText("Button按钮");
5.  }
6.  });
7.  imageButton.setOnClickListener(new View.OnClickListener() {
8.  public void onClick(View view) {
9.  textView.setText("ImageButton按钮");
10. }
11. });
```

- 第2行代码中button对象通过调用setOnClickListener()函数注册一个点击（Click）事件的监听器View.OnClickListener()
- 第3行代码是点击事件的回调函数
- 第4行代码将TextView的显示内容更改为“Button按钮”

5.2 界面控件

•5.2.2 Button和ImageButton

- View.OnClickListener()
  - View.OnClickListener()是View定义的点击事件的监听器接口，并在接口中仅定义了onClick()函数
  - 当Button从Android界面框架中接收到事件后，首先检查这个事件是否是点击事件，如果是点击事件，同时Button又注册了监听器，则会调用该监听器中的onClick()函数
  - 每个View仅可以注册一个点击事件的监听器，如果使用setOnClickListener()函数注册第二个点击事件的监听器，之前注册的监听器将被自动注销

5.2 界面控件

•5.2.2 Button和ImageButton

- 多个按钮注册到同一个点击事件的监听器上，代码如下:

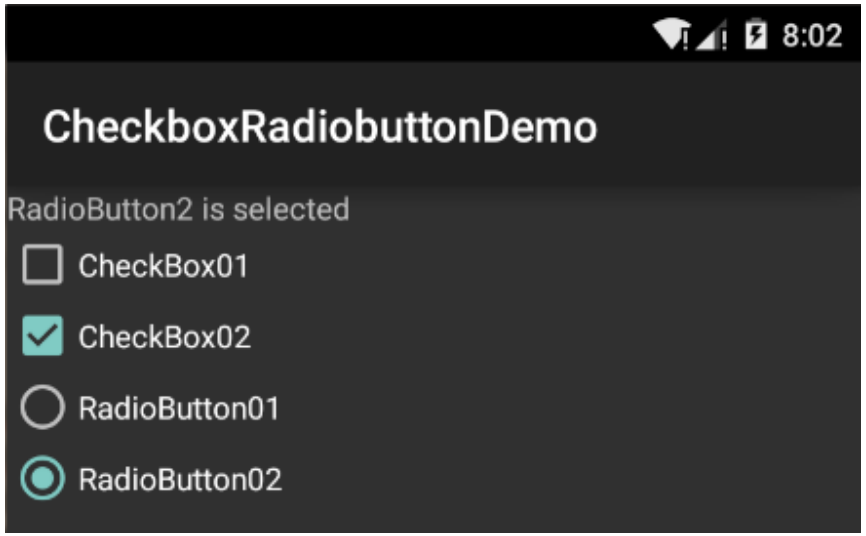
```
1.  Button.OnClickListener buttonListener = new Button.OnClickListener(){
2.      @Override
3.          public void onClick(View v) {
4.              switch(v.getId()){
5.          case R.id.Button01:
6.  textView.setText("Button按钮");
7.   return;
8.                  case R.id.ImageButton01:
9.  textView.setText("ImageButton按钮");
10.  return;
11.             }
12.     }};
13.  button.setOnClickListener(buttonListener);
14.  imageButton.setOnClickListener(buttonListener);
```

- 第1行至第12行代码定义了一个名为buttonListener的点击事件监听器
- 第13行代码将该监听器注册到Button上
- 第14行代码将该监听器注册到ImageButton上

5.2 界面控件

•5.2.3 CheckBox和RadioButton

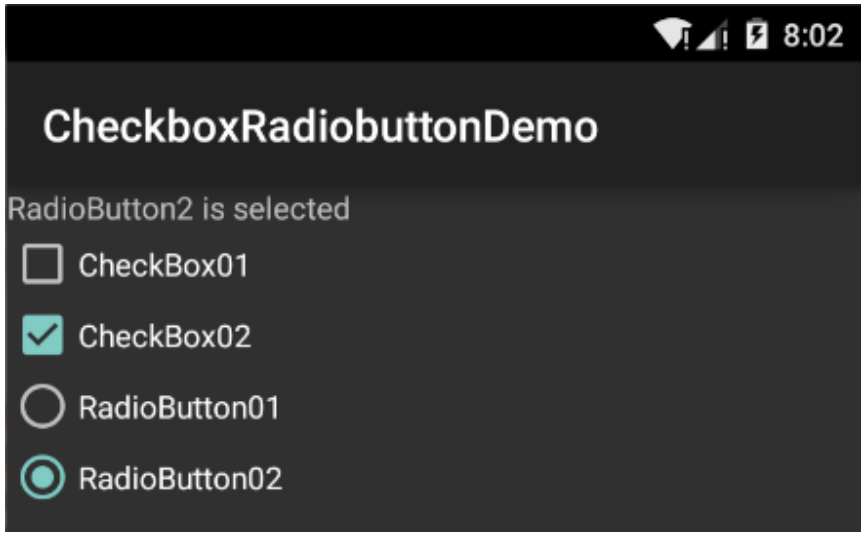
- CheckBox同时可以选择多个选项的控件
- RadioButton则是仅可以选择一个选项的控件
- RadioGroup是RadioButton的承载体，程序运行时不可见。应用程序中可能包含一个或多个RadioGroup，一个RadioGroup包含多个RadioButton，在每个RadioGroup中，用户仅能够选择其中一个RadioButton



5.2 界面控件

•5.2.3 CheckBox和RadioButton

- 建立一个“CheckboxRadiobuttonDemo”工程，包含五个控件，从上至下分别是
  - TextView01
  - CheckBox01
  - CheckBox02
  - RadioButton01
  - RadioButton02



5.2 界面控件

• 5.2.3 CheckBox和RadioButton

- CheckboxRadiobuttonDemo在XML文件中的代码

```
1.  <TextView android:id="@+id/TextView01"
2.  android:layout_width=“match_parent"
3.  android:layout_height="wrap_content"
4.  android:text="@string/hello"/>
5.  <CheckBox android:id="@+id/CheckBox01"
6.  android:layout_width="wrap_content"
7.  android:layout_height="wrap_content"
8.  android:text="CheckBox01" >
9.  </CheckBox>
10. <CheckBox android:id="@+id/CheckBox02"
11. android:layout_width="wrap_content"
12. android:layout_height="wrap_content"
13. android:text="CheckBox02" >
14.  </CheckBox>
15. <RadioGroup android:id="@+id/RadioGroup01"
16. android:layout_width="wrap_content"
17. android:layout_height="wrap_content">
18. <RadioButton android:id="@+id/RadioButton01"
19. android:layout_width="wrap_content"
20. android:layout_height="wrap_content"
21. android:text="RadioButton01" >
22. </RadioButton>
23. <RadioButton android:id="@+id/RadioButton02"
24. android:layout_width="wrap_content"
25. android:layout_height="wrap_content"
26. android:text="RadioButton02" >
27. </RadioButton>
28. </RadioGroup>
```

5.2 界面控件

•5.2.3 CheckBox和RadioButton

- 引用CheckBox和RadioButton的方法参考下面的代码

```
1.  CheckBox checkBox1= (CheckBox)findViewById(R.id.CheckBox01);
2.  RadioButton radioButton1 =(RadioButton)findViewById(R.id.RadioButton01);
```

- CheckBox设置点击事件监听器的简要代码

```
1.  CheckBox.OnClickListener checkboxListener = new CheckBox.OnClickListener(){
2.  @Override
3.  public void onClick(View v) {
4.    //过程代码
5.  }};
6.  checkBox1.setOnClickListener(checkboxListener);
7.  checkBox2.setOnClickListener(checkboxListener);
```

- 与Button设置点击事件监听器中介绍的方法相似，唯一不同在于将Button.OnClickListener换成了CheckBox.OnClickListener

5.2 界面控件

•5.2.3 CheckBox和RadioButton

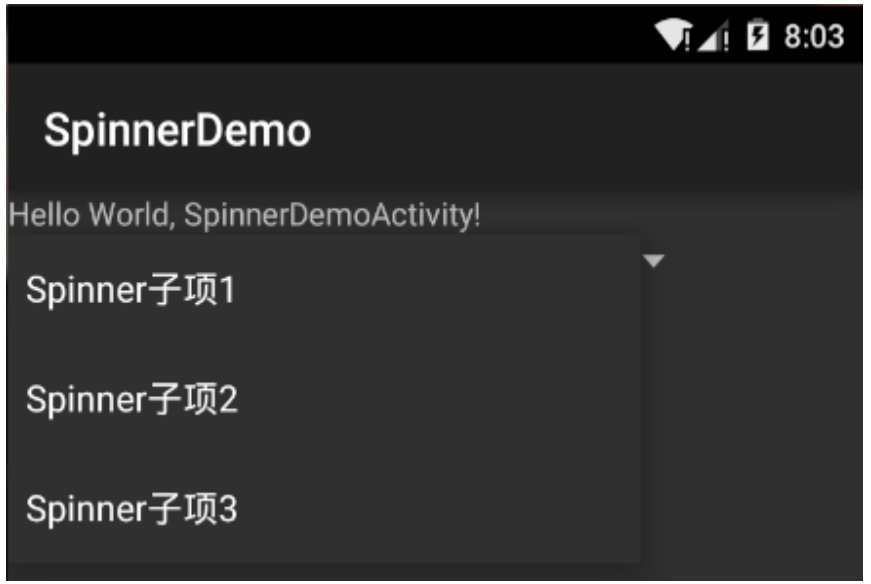
- RadioButton设置点击事件监听器的方法

```
1.  RadioButton.OnClickListener radioButtonListener = new RadioButton.OnClickListener(){
2.  @Override
3.  public void onClick(View v) {
4.    //过程代码
5.  }};
6.  radioButton1.setOnClickListener(radioButtonListener);
7.  radioButton2.setOnClickListener(radioButtonListener);
```

5.2 界面控件

•5.2.4 Spinner

- 一种能够从多个选项中选一选项的控件，使用浮动菜单为用户提供选择
- 类似于桌面程序的组合框（ComboBox）



5.2 界面控件

•5.2.4 Spinner

- SpinnerDemo在XML文件中的代码

```
1.  <TextView  android:id="@+id/TextView01"
2.     android:layout_width=“match_parent"
3.     android:layout_height="wrap_content"
4.     android:text="@string/hello"/>
5.  <Spinner android:id="@+id/Spinner01"
6.  android:layout_width="300dip"
7.  android:layout_height="wrap_content">
8.  </Spinner>
```

- 第5行使用<Spinner>标签声明了一个Spinner控件
- 第6行代码中指定了该控件的宽度为300dip
- dip是设备独立像素，不同设备有不同 的现实效果

5.2 界面控件

- 在SpinnerDemo.java文件中，定义一个ArrayAdapter适配器，在ArrayAdapter中添加需要在Spinner中可以选择的内容
- 适配器绑定界面控件和底层数据，若底层数据更改了，用户界面也相应修改显示内容，就不需要应用程序再监视，从而极大的简化的代码的复杂性

```
1.  Spinner spinner = (Spinner) findViewById(R.id.Spinner01);
2.  List<String> list  = new ArrayList<String>();
3.  list .add("Spinner子项1");
4.  list .add("Spinner子项2");
5.  list .add("Spinner子项3");
6.  ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
    android.R.layout.simple_spinner_item, list );
7.  adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
8.  spinner.setAdapter(adapter);
```

5.2 界面控件

•5.2.4 Spinner

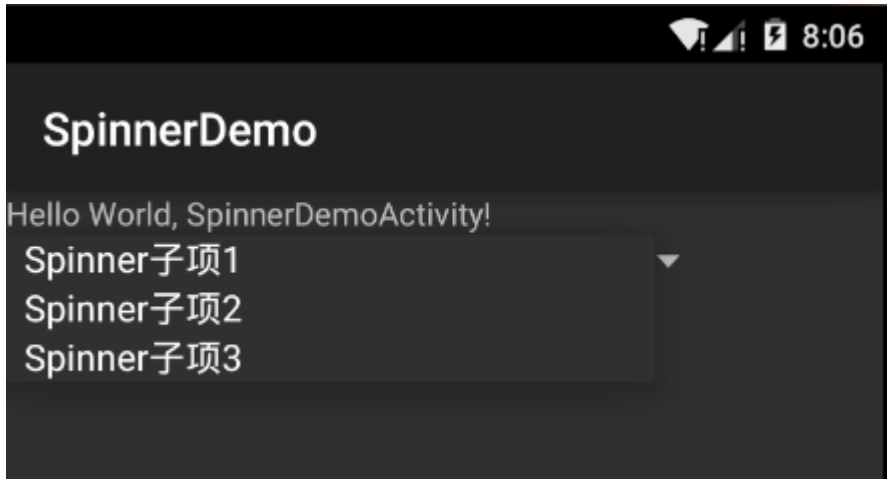
- 第2行代码建立了一个数组列表（ArrayList），这种数组列表可以根据需要进行增减
- <String>表示数组列表中保存的是字符串类型的数据
- 在代码的第3、4、5行中，使用add()函数分别向数组列表中添加3个字符串
- 第6行代码建立了一个ArrayAdapter的数组适配器，数组适配器能够将界面控件和底层数据绑定在一起
- 第8行代码实现绑定过程，所有ArrayList中的数据，将显示在Spinner的浮动菜单中
- 第7行代码设定了Spinner的浮动菜单的显示方式，其中，android.R.layout.simple_spinner_dropdown_item是Android系统内置的一种浮动菜单

5.2 界面控件

•5.2.4 Spinner

- Spinner的浮动菜单的显示方式
  - android.R.layout.simple_spinner_dropdown_item

  - android.R.layout.simple_spinner_item



5.2 界面控件

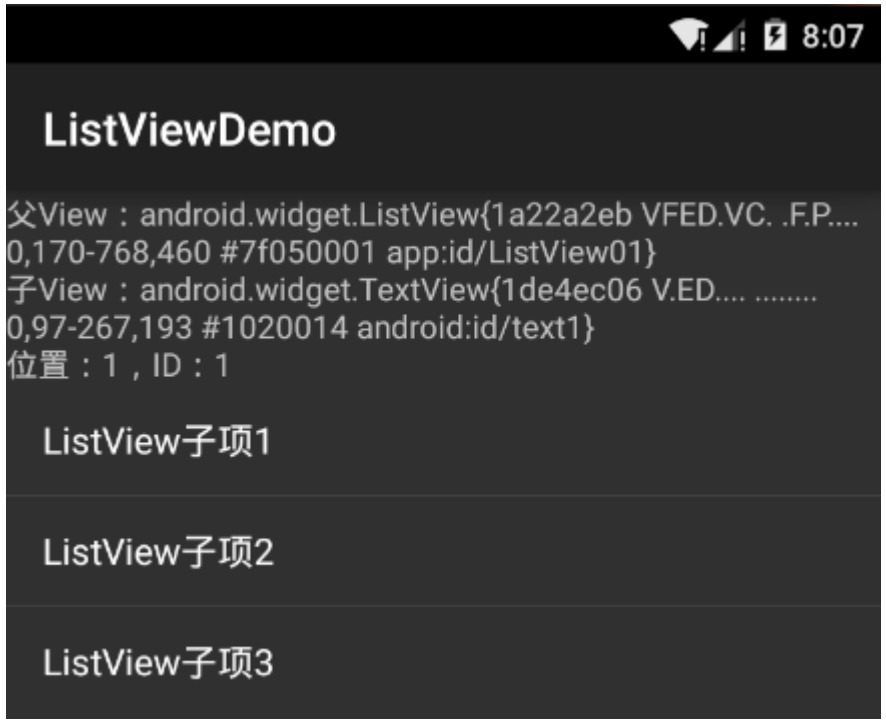
•5.2.5 ListView

- ListView是一种用于垂直显示的列表控件，如果显示内容过多，则会出现垂直滚动条
- ListView能够通过适配器将数据和自身绑定，在有限的屏幕上提供大量内容供用户选择,所以是经常使用的用户界面控件
- ListView支持点击事件处理，用户可以用少量的代码实现复杂的选择功能

5.2 界面控件

•5.2.5 ListView

- 建立一个“ListViewDemo”程序，包含四个控件，从上至下分别为TextView01、ListView01、 ListView02和 ListView03



5.2 界面控件

•5.2.5 ListView

- ListViewDemo在XML文件中的代码

```
1.  <TextView  android:id="@+id/TextView01"
2.  android:layout_width=“match_parent"
3.  android:layout_height="wrap_content"
4.  android:text="@string/hello" />
5.  <ListView android:id="@+id/ListView01"
6.  android:layout_width="wrap_content"
7.  android:layout_height="wrap_content">
8.  </ListView>
```

5.2 界面控件

•5.2.5 ListView

- 在ListViewDemo.java文件中，首先需要为ListView创建适配器，并添加ListView中所显示的内容

```
1.  final TextView textView = (TextView)findViewById(R.id.TextView01);
2.  ListView listView = (ListView)findViewById(R.id.ListView01);
3.  List<String> list  = new ArrayList<String>();
4.  list.add("ListView子项1");
5.  list.add("ListView子项2");
6.  list.add("ListView子项3");
7.  ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_1, list );
8.  listView.setAdapter(adapter);
```

- 第2行代码通过ID引用了XML文件中声明的ListView
- 第7行代码声明了适配器ArrayAdapter，第三个参数list说明适配器的数据源为数组列表
- 第8行代码将ListView和适配器绑定

5.2 界面控件

•5.2.5 ListView

- 下面的代码声明了ListView子项的点击事件监听器，用以确定用户在ListView中，选择的是哪一个子项

```
1. AdapterView.OnItemClickListener listViewListener = new
   AdapterView.OnItemClickListener(){
2. @Override
3. public void onItemClick(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
4. String msg = "父View： "+arg0.toString()+"\n"+"子View： "+arg1.toString()+"\n"+"位置：
   "+String.valueOf(arg2)+"， ID： "+String.valueOf(arg3);
5. textView.setText(msg);
6. }};
7. listView.setOnItemClickListener(listViewListener);
```

5.2 界面控件

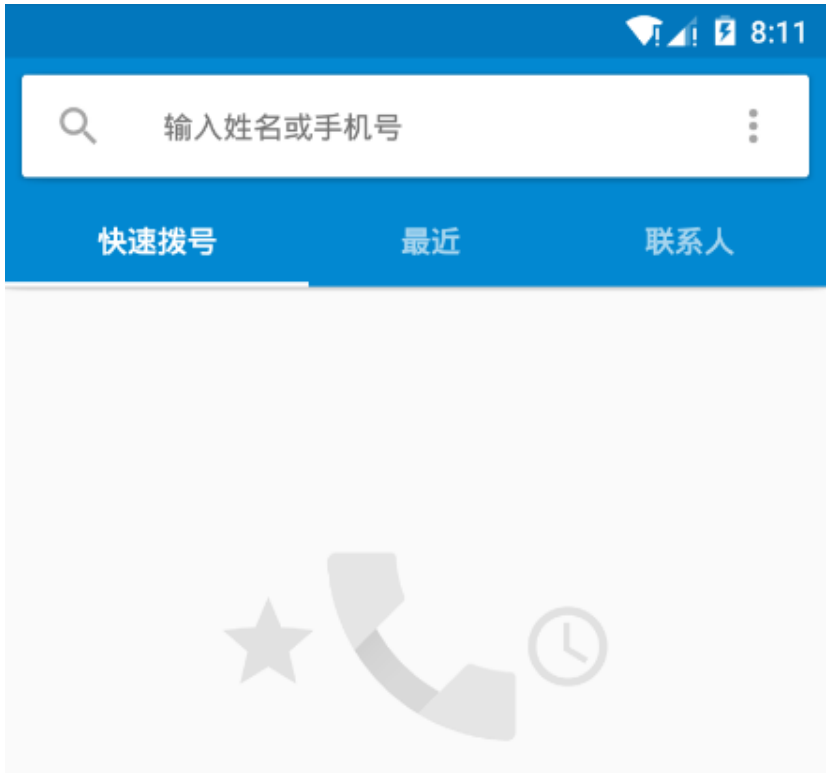
•5.2.5 ListView

- 第1行的AdapterView.OnItemClickListener是ListView子项的点击事件监听器，同样是一个接口，需要实现onItemClick()函数。在ListView子项被选择后，onItemClick()函数将被调用
- 第3行的onItemClick()函数中一共有四个参数，参数0表示适配器控件，就是ListView；参数1表示适配器内部的控件，是ListView中的子项；参数2表示适配器内部的控件，也就是子项的位置；参数3表示子项的行号
- 第4行和第5行代码用于显示信息，选择子项确定后，在TextView中显示子项父控件的信息、子控件信息、位置信息和ID信息
- 第7行代码是ListView指定刚刚声明的监听器

5.2 界面控件

•5.2.6 TabHost

- Tab标签页是界面设计时经常使用的界面控件，可实现多个分页之间的快速切换，每个分页可以显示不同的内容
- 下图是Android系统内置的Tab标签页，点击“呼出/接听键”后出现，用于电话呼出和查看拨号记录、联系人



5.2 界面控件

•5.2.6 TabHost

- 在Android SDK 3.0中，随着新的UI设计思想的引入，android.app.Fragment成为一种新的界面设计模式
- Android SDK 4.0继承了3.0版本的设计思路，因此不建议开发者使用android.app.TabActivity，而用使用新出现的Fragment实现Tab标签页
- 但因旧版本Android系统还有一定的生存周期，且使用TabActivity实现的Tab标签页的方法在Android SDK 4.0中仍可以正常运行，所以仍对这种方法进行介绍

5.2 界面控件

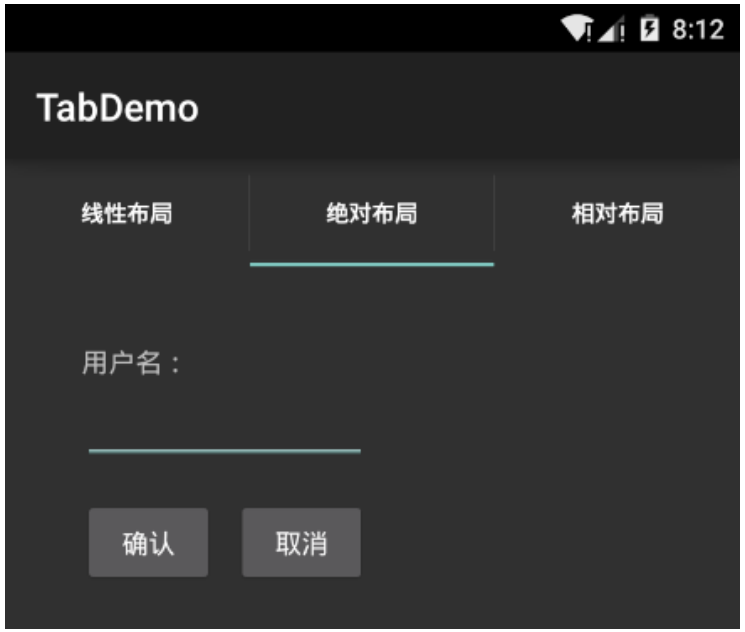
•5.2.6 TabHost

- Tab标签页的使用
  - 首先要设计所有的分页的界面布局
  - 在分页设计完成后，使用代码建立Tab标签页（ TabActivity ），并给每个分页添加标识和标题
- 每个分页建立一个XML文件，用以编辑和保存分页的界面布局，使用的方法与设计普通用户界面没有什么区别

5.2 界面控件

•5.2.6 TabHost

- 建立一个“TabDemo”程序，包含三个XML文件，分别为tab1.xml、tab2.xml和tab3.xml，这3个文件分别使用线性布局、相对布局和绝对布局示例中的main.xml的代码，并将布局的ID分别定义为layout01、layout02和layout03



5.2 界面控件

•5.2.6 TabHost

- tab1.xml文件代码

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:id = "@+id/layout01"
  ……
  ……
</LinearLayout>
```

- tab2.xml文件代码

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout android:id="@+id/layout02"
  ……
  ……
</AbsoluteLayout>
```

- tab3.xml文件代码

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout android:id="@+id/layout03"
 ……
 ……
</RelativeLayout>
```

5.2 界面控件

• 5.2.6 TabHost

- 在TabDemo.java文件中键入下面的代码，创建Tab标签页，并建立子页与界面布局直接的关联关系

```
1.  package edu.hrbeu.TabDemo;
2.  import android.app.TabActivity;
3.  import android.os.Bundle;
4.  import android.widget.TabHost;
5.  import android.view.LayoutInflater;
6.  @SuppressWarnings("deprecation")
7.  public class TabDemoActivity extends TabActivity {
8.  @Override
9.  public void onCreate(Bundle savedInstanceState) {
10. super.onCreate(savedInstanceState);
11. TabHost tabHost = getTabHost();
12. LayoutInflater.from(this).inflate(R.layout.tab1,
    tabHost.getTabContentView(),true);
13. LayoutInflater.from(this).inflate(R.layout.tab2,
    tabHost.getTabContentView(),true);
14. LayoutInflater.from(this).inflate(R.layout.tab3,
    tabHost.getTabContentView(),true);
15. tabHost.addTab(tabHost.newTabSpec("TAB1")
16. .setIndicator("线性布局").setContent(R.id.layout01));
17. tabHost.addTab(tabHost.newTabSpec("TAB2")
18. .setIndicator("绝对布局").setContent(R.id.layout02));
19. tabHost.addTab(tabHost.newTabSpec("TAB3")
20. .setIndicator("相对布局").setContent(R.id.layout03));
21.    }
22. }
```

5.2 界面控件

•5.2.6 TabHost

- 第8行代码的声明TabDemo类继承与TabActivity，与以往继承Activity不同，TabActivity支持内嵌多个Activity或View
- 第12行代码通过getTabHost()函数获得了Tab标签页的容器，用以承载Tab标签和分页的界面布局
- 第13行代码通过LayoutInflater将tab1.xml文件中的布局转换为Tab标签页可以使用的View对象
- 第16行代码使用addTab()函数添加了第1个分页，tabHost.newTabSpec("TAB1")表明在第12行代码中建立的tabHost上，添加一个标识为TAB1的Tab分页
- 第17行代码使用setIndicator()函数设定分页显示标题，使用setContent()函数设定分页所关联的界面布局

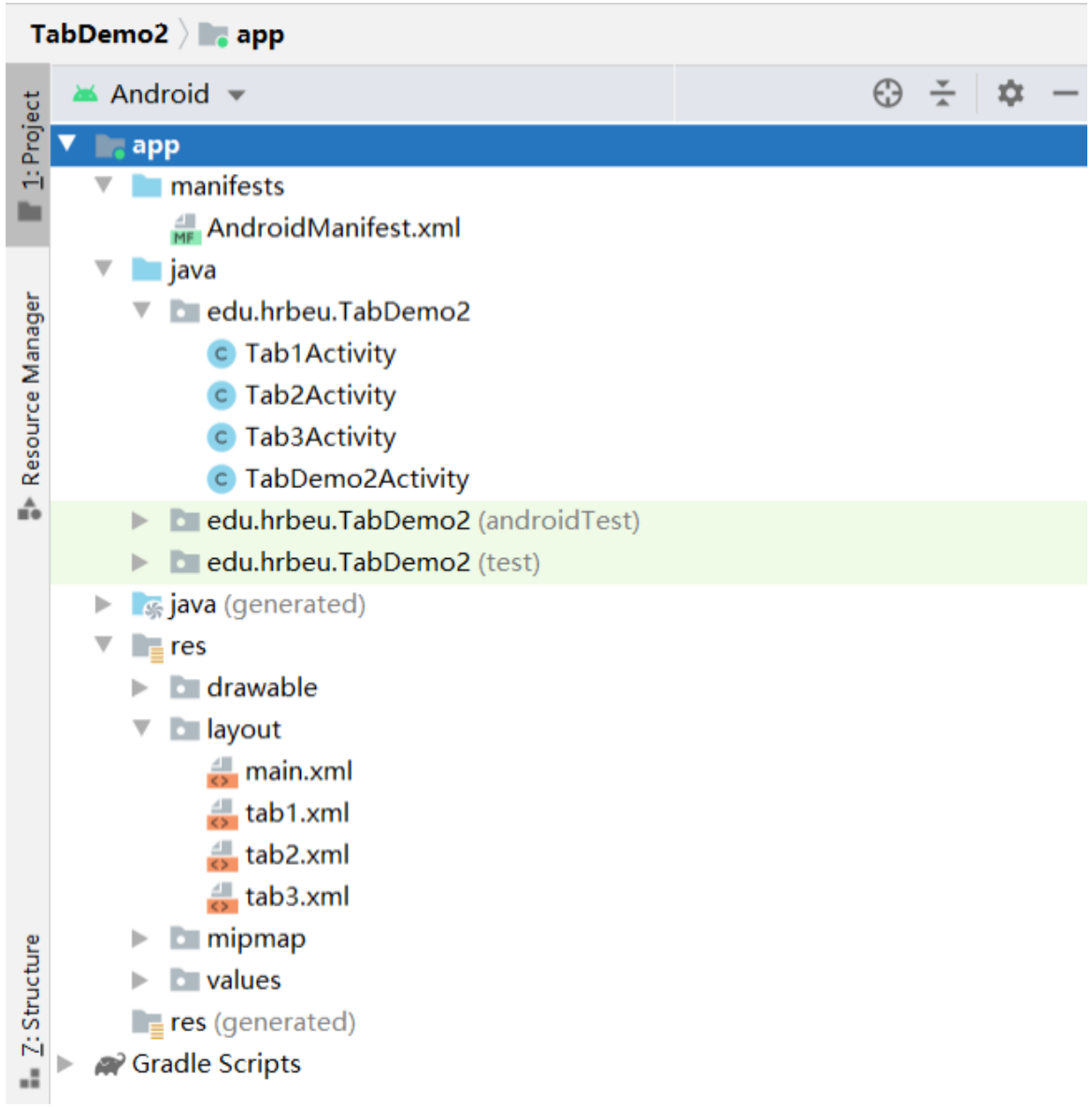
5.2 界面控件

•5.2.6 TabHost

- 在实现Tab标签页时，除了可以将多个Tab分页放置在同一个Activity中，还可以将不同Tab分页加载到不同的Activity上
  - 两种方式在界面显示上是没有区别的
  - 建议使用后一种方式处理Tab分页和Activity之间的关系，每个Tab分页对应一个Activity，有利于用户对界面控件的管理和控制

5.2 界面控件

- TabDemo2示例说明如何将不同的Activity显示在不同的Tab分页上
  - TabDemo2示例与TabDemo示例的用户界面是完全相同的



5.2 界面控件

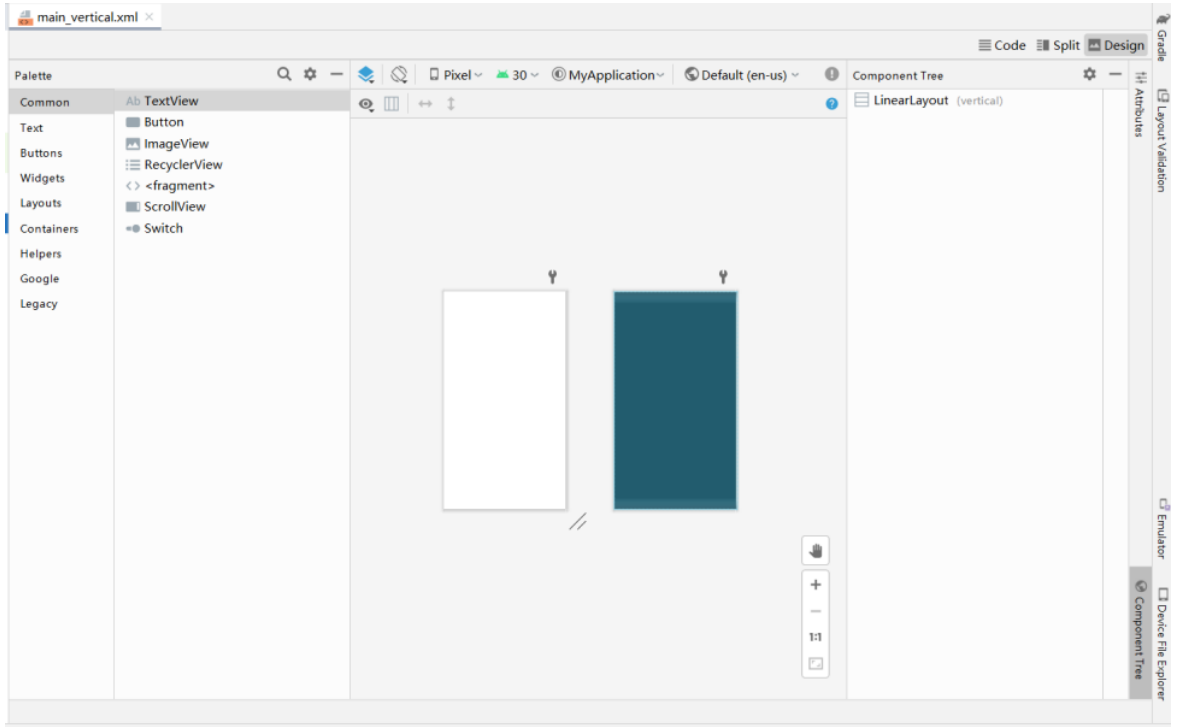
- TabDemo1与 TabDemo2对比
  - 与TabDemo1示例相比，TabDemo2示例的布局目录中（/res/layout）中多了一个main.xml文件
  - 代码目录中增加了Tab1Activity.java、Tab2Activity.java和Tab3Activity.java三个文件

5.3 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

5.3 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



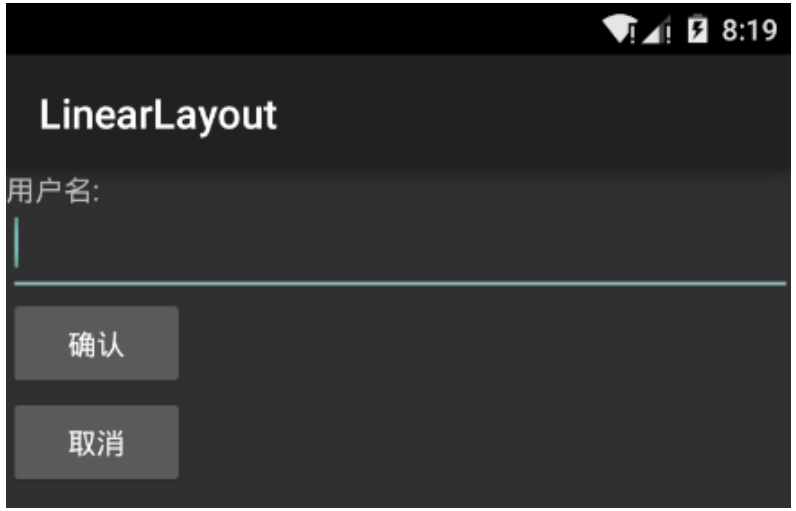
5.3 界面布局

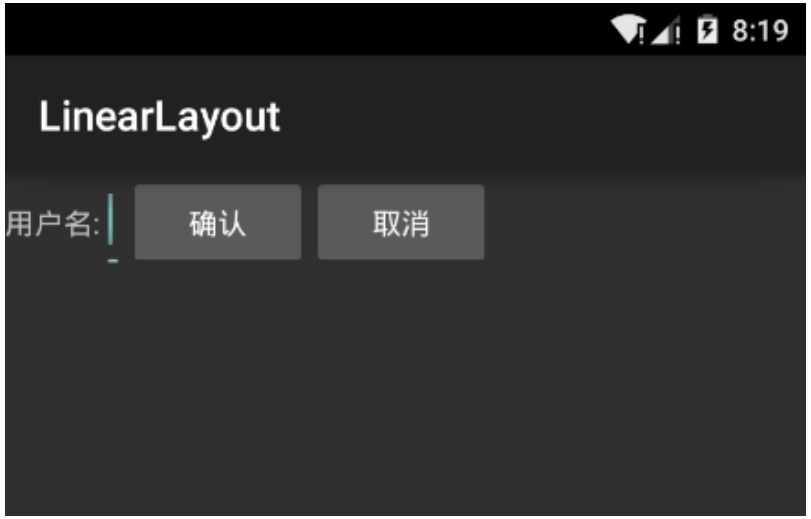
- **常用的6种界面布局**
  - 线性布局
  - 框架布局
  - 表格布局
  - 相对布局
  - 绝对布局
  - 网格布局

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

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

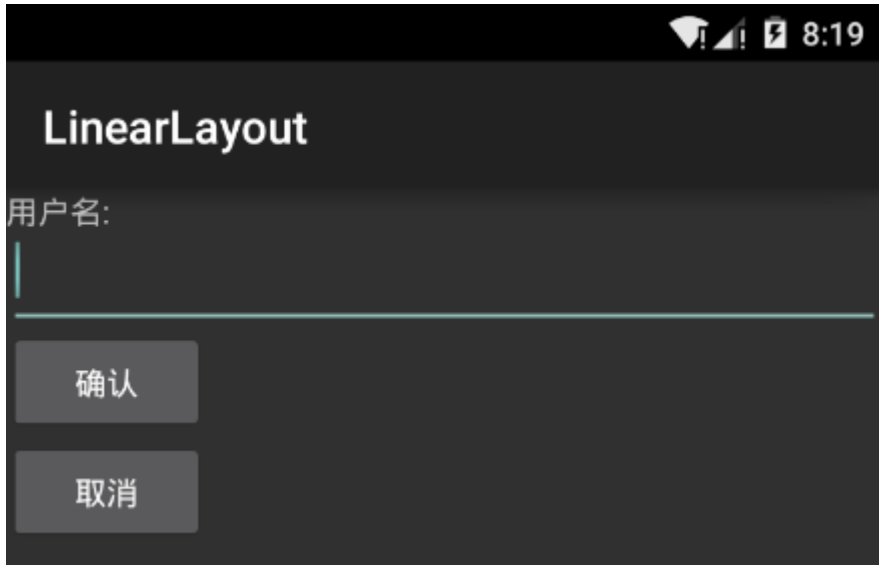
•5.3.1 线性布局

- 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>
```

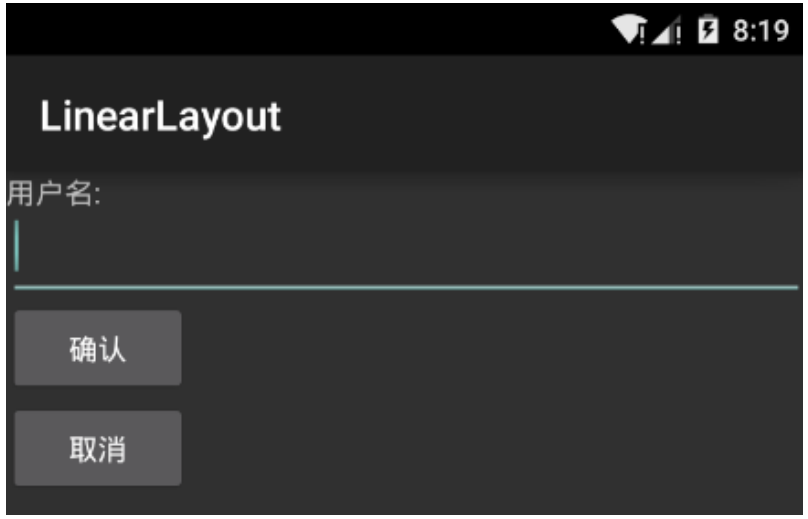
5.3 界面布局

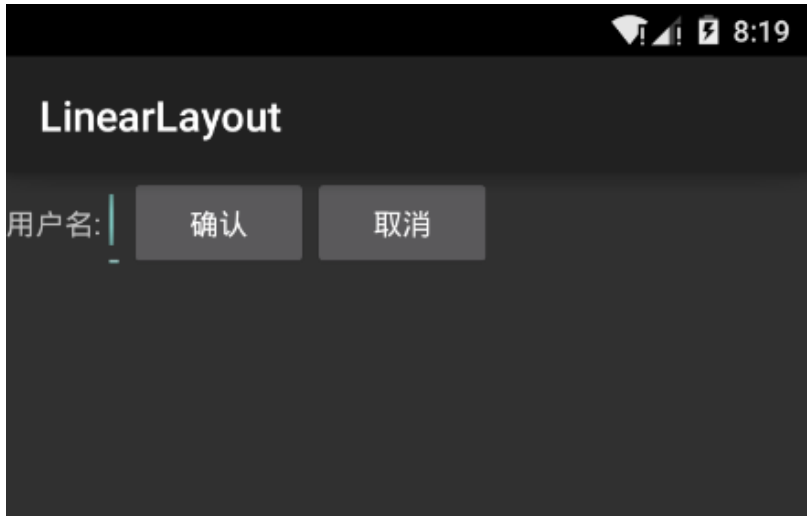
```
12.  <Button android:id="@+id/ok"
13.  android:layout_width="wrap_content"
14.  android:layout_height="wrap_content"
15.  android:text="确认">
16.  </Button>
17.  <Button android:id="@+id/cancel"
18.  android:layout_width="wrap_content"
19.  android:layout_height="wrap_content"
20.  android:text="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



5.3 界面布局

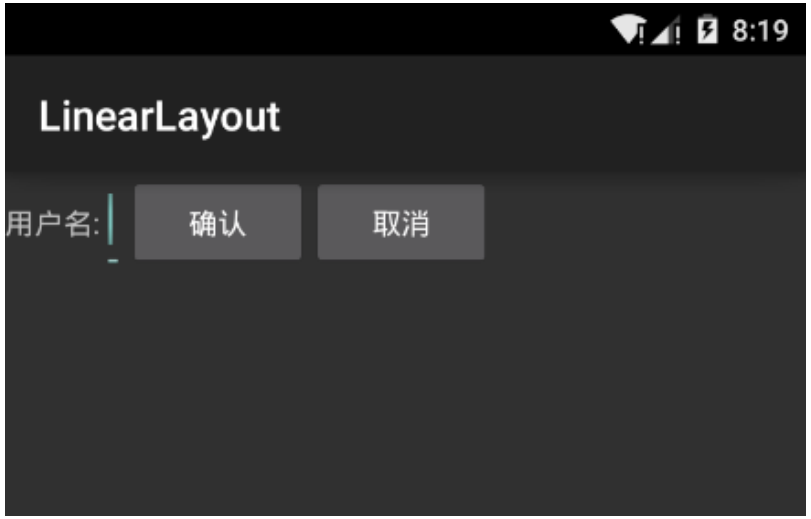
- **5.3.1 线性布局**
  - 将LinearLayout.java文件中的setContentView(R.layout.main)，更改为setContentView(R.layout.main_vertical)。运行后的结果如图





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)



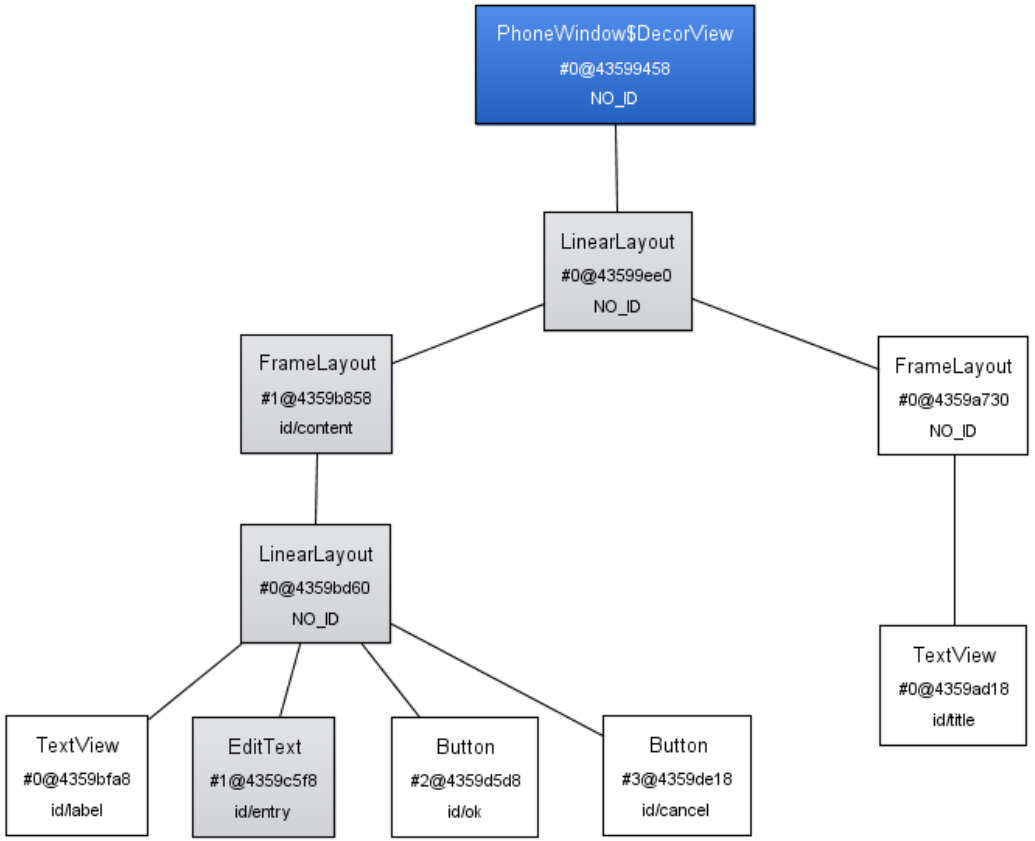
5.3 界面布局

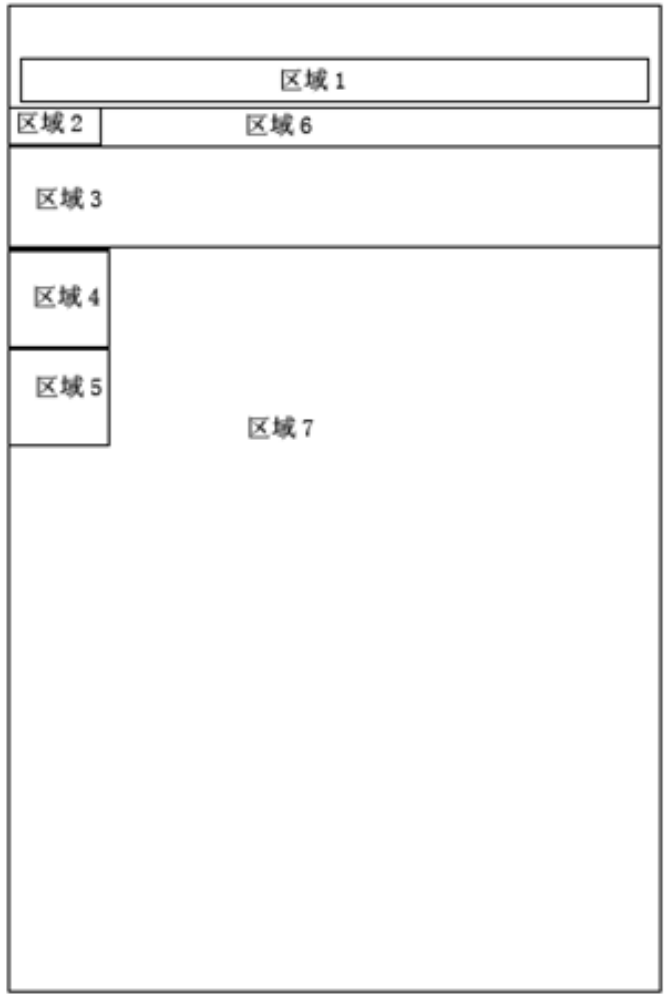
- **5.3.2 框架布局**
  - 框架布局（FrameLayout）是最简单的界面布局，是用来存放一个元素的空白空间，且子元素的位置是不能够指定的，只能够放置在空白空间的左上角
  - 如果有多个子元素，后放置的子元素将遮挡先放置的子元素
  - 使用Android SDK中提供的层级观察器（Hierarchy Viewer）进一步分析界面布局

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

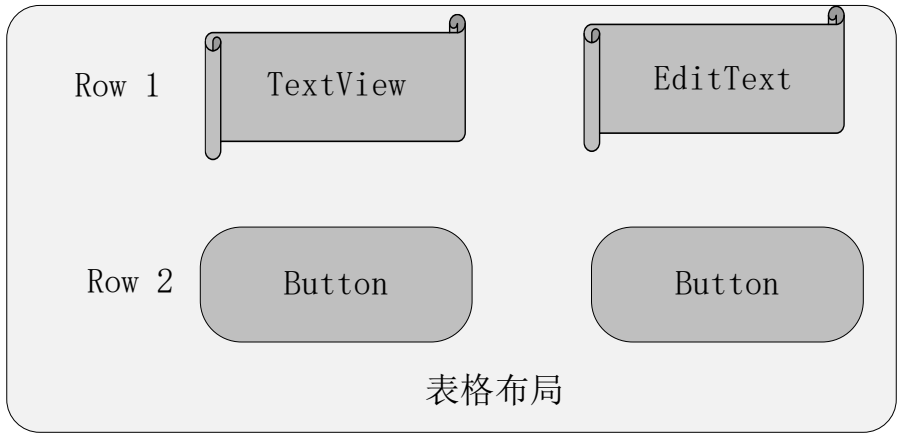
5.3 界面布局

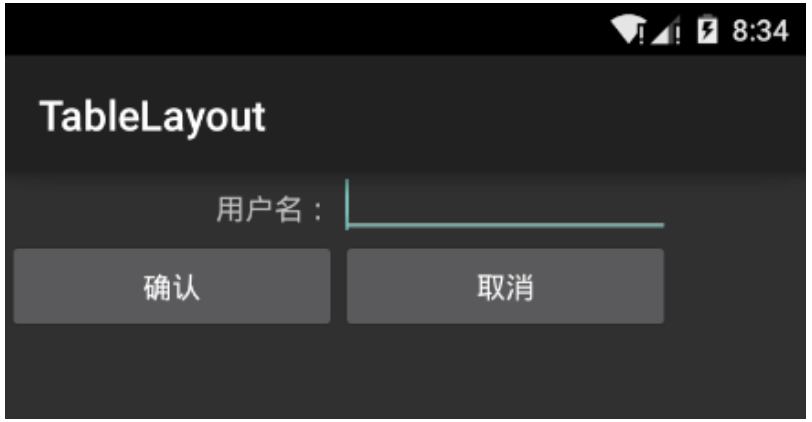
- **5.3.3 表格布局**
  - 表格布局（TableLayout）是一种常用的界面布局，通过指定行和列将界面元素添加到表格中
    - 网格的边界对用户是不可见的
    - 表格布局支持嵌套
      - 可以将表格布局放置在表格布局的表格中
      - 可以在表格布局中添加其他界面布局，例如线性布局、相对布局等

5.3 界面布局

•5.3.3 表格布局

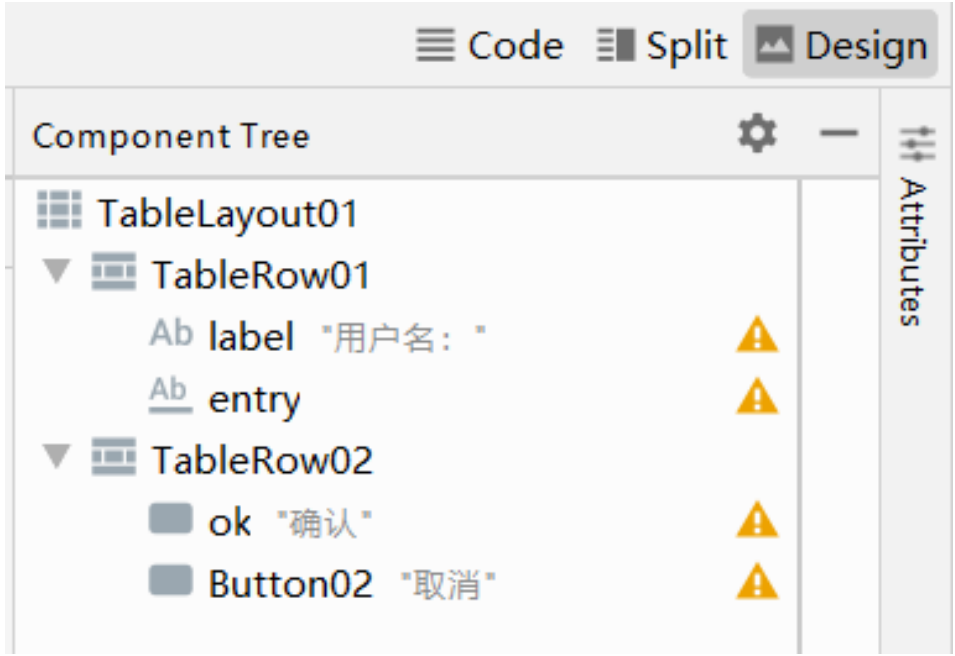
- 表格布局示意图及效果图





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>
```

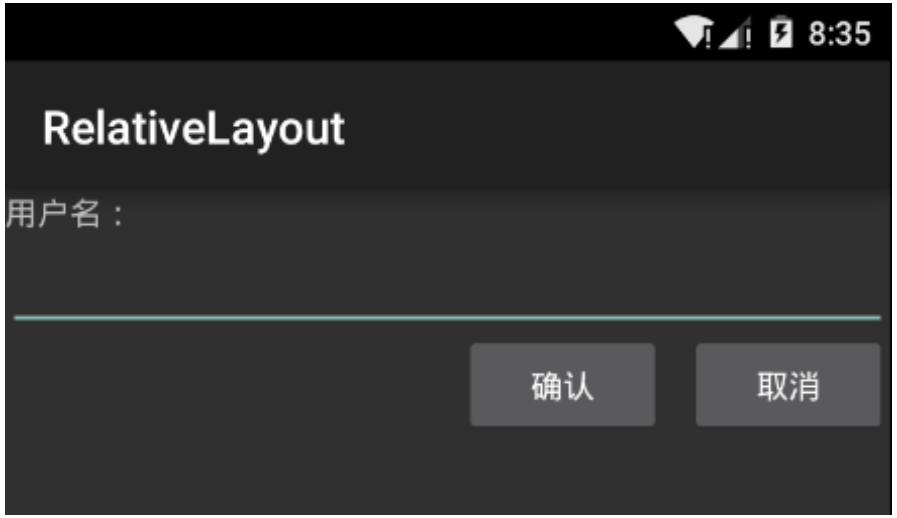
5.3 界面布局

- 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

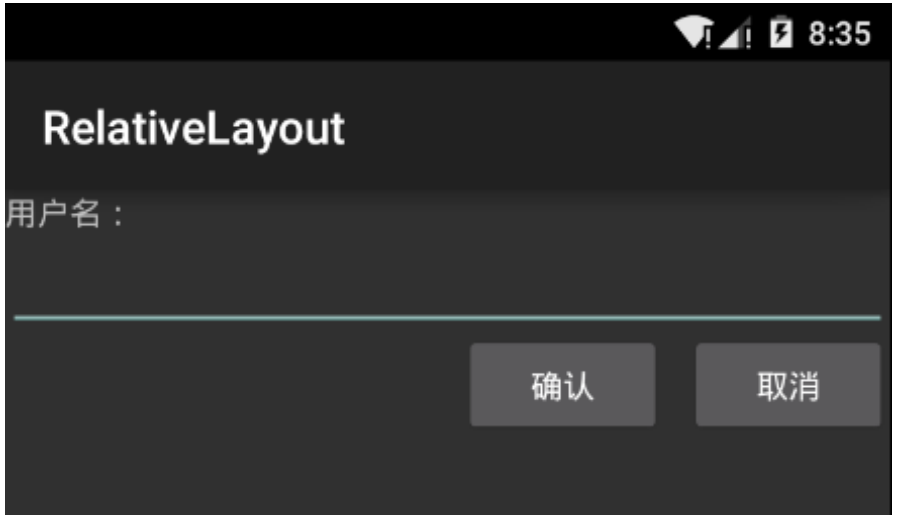
5.3 界面布局

•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



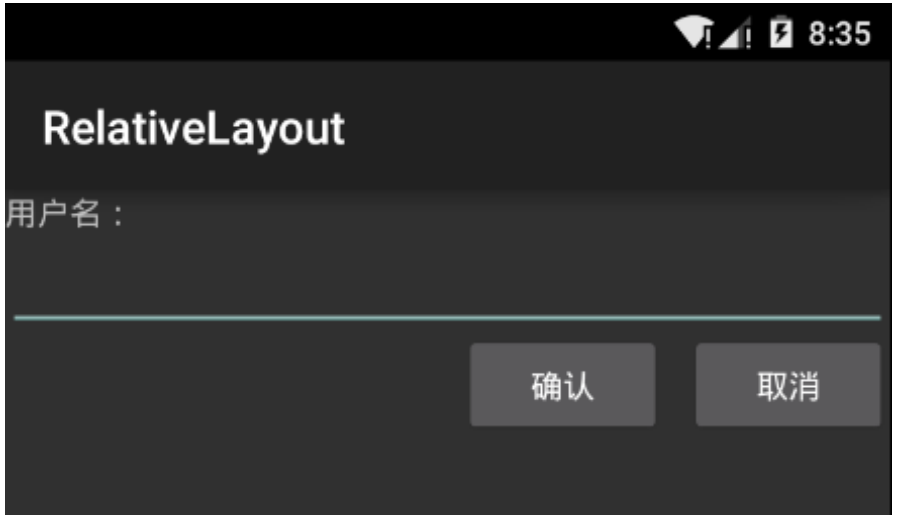
5.3 界面布局



- *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




```
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

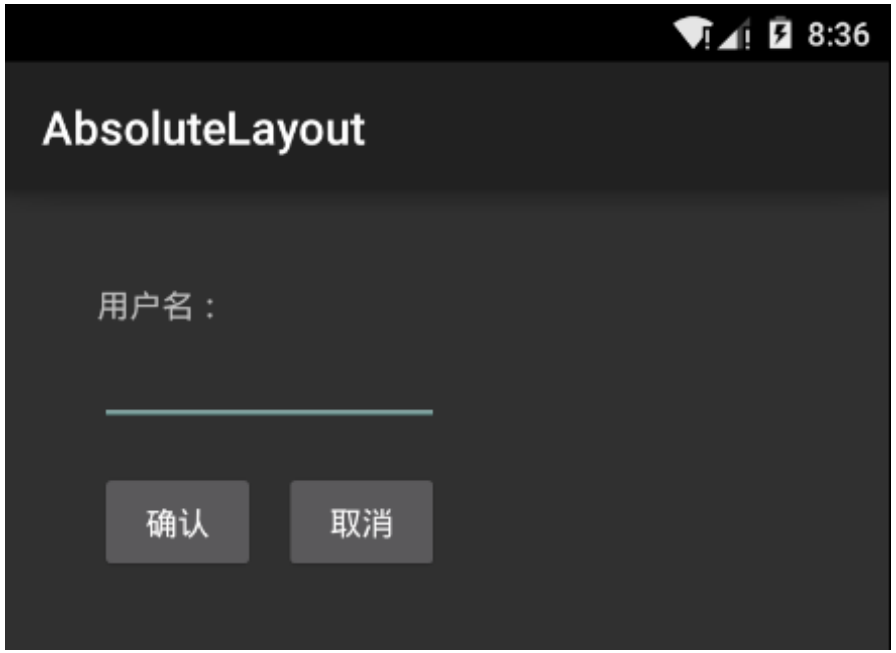
- **5.3.4 相对布局**
  - 第3行使用了<RelativeLayout>标签声明一个相对布局
  - 第15行使用位置属性android:layout_below，确定EditText控件在ID为label的元素下方
  - 第20行使用属性android:layout_alignParentRight，声明该元素在其父元素的右边边界对齐
  - 第21行设定属性android:layout_marginLeft，左移10dip
  - 第22行声明该元素在ID为entry的元素下方
  - 第28行声明使用属性android:layout_toLeftOf，声明该元素在ID为cancel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

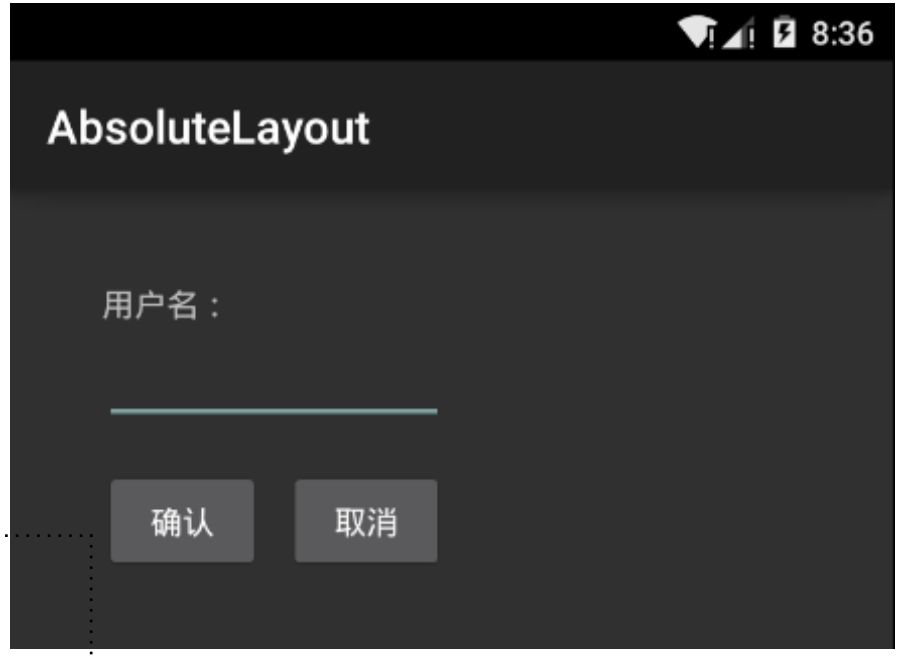
- **5.3.5 绝对布局**
  - 绝对布局（AbsoluteLayout）能通过指定界面元素的坐标位置，来确定用户界面的整体布局
  - 绝对布局是一种不推荐使用的界面布局，因为通过X轴和Y轴确定界面元素位置后，Android系统不能够根据不同屏幕对界面元素的位置进行调整，降低了界面布局对不同类型和尺寸屏幕的适应能力

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：

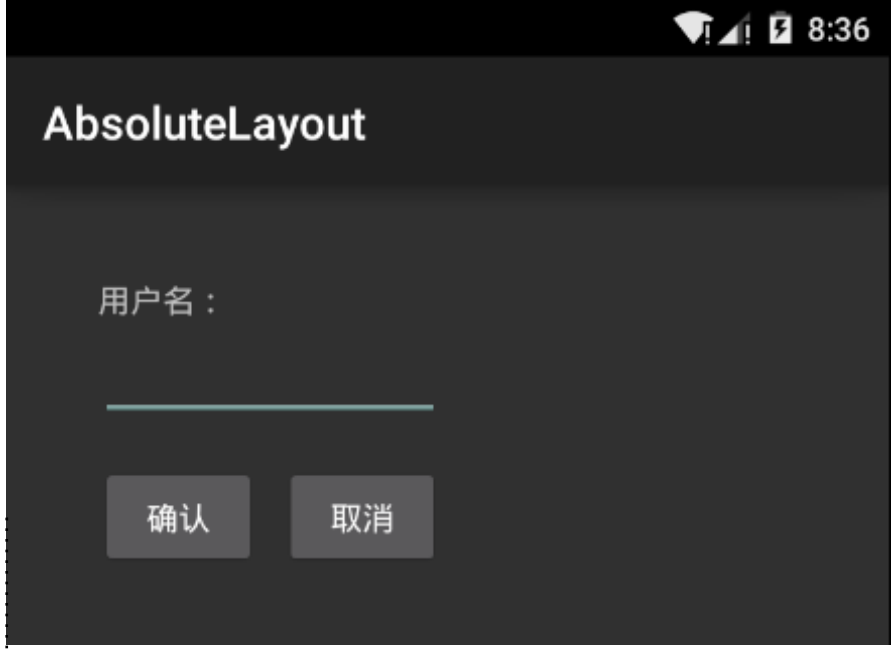
```
1.  <?xml version="1.0" encoding="utf-8"?>
2.
3.  <AbsoluteLayout android:id="@+id/AbsoluteLayout01"
4.  android:layout_width=“match_parent"
5.  android:layout_height=“match_parent"
6.  xmlns:android="http://schemas.android.com/apk/res/android">
7.  <TextView android:id="@+id/label"
8.  android:layout_x="40dip"
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>
```

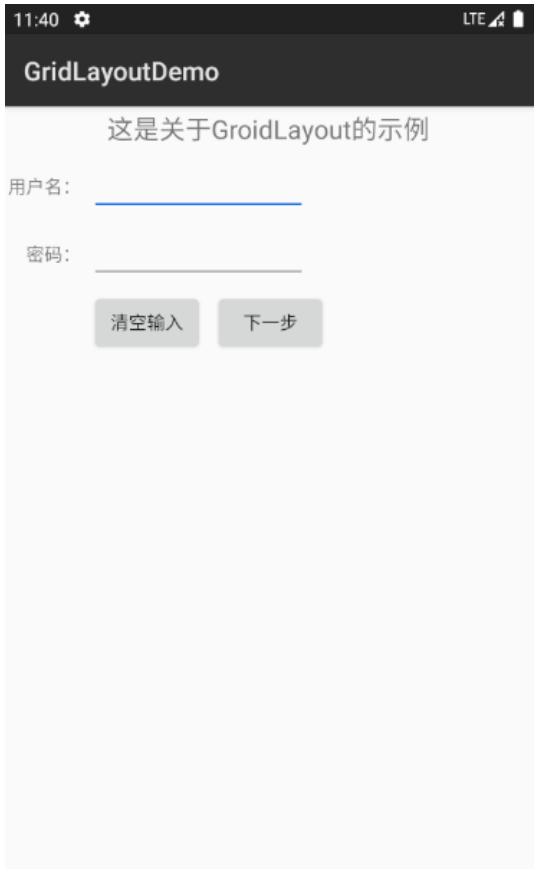


5.3 界面布局

- **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。
  - GroidLayoutDemo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

- **GroidLayoutDemo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

- GroidLayoutDemo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，GroidLayoutDemo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GroidLayout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

5.4 菜单

- **菜单**
  - 应用程序中非常重要的组成部分
  - 在不占用界面空间的前提下，为应用程序提供了统一的功能和设置界面
  - 为程序开发人员提供了易于使用的编程接口
- **Android系统支持三种菜单**
  - 选项菜单（Option Menu）
  - 子菜单（Submenu）
  - 快捷菜单（Context Menu）

5.4菜单

- **5.4.1菜单资源**
  - Android程序的菜单
    - 可以在代码中动态生成
    - 使用XML文件制作菜单资源
      - 使用XML描述菜单是较好的选择
      - 可以将菜单的内容与代码分离
      - 有利于分析和调整菜单结构

5.4 菜单

•5.4.1菜单资源

- 代码(Java)生成的菜单

```
1.  final static int MENU_00 = Menu.FIRST;
2.  final static int MENU_01 = Menu.FIRST+1;
3.  final static int MENU_02 = Menu.FIRST+2;
4.  final static int MENU_03 = Menu.FIRST+3;
5.  final static int MENU_04 = Menu.FIRST+4;
6.
7.  @Override
8.  public boolean onCreateOptionsMenu(Menu menu) {
9.  menu.add(0,MENU_00,0,"打印").setIcon(R.drawable.pic0);
10. menu.add(0,MENU_01,1,"新建").setIcon(R.drawable.pic1);
11. menu.add(0,MENU_02,2,"邮件").setIcon(R.drawable.pic2);
12. menu.add(0,MENU_03,3,"设置").setIcon(R.drawable.pic3);
13. menu.add(0,MENU_04,4,"订阅").setIcon(R.drawable.pic4);
14. return true;15}
```

5.4菜单

•5.4.1菜单资源

- XML菜单

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <menu xmlns:android="http://schemas.android.com/apk/res/android">
3.      <item android:id="@+id/main_menu_0"
4.          android:icon="@drawable/pic0"
5.          android:title="打印" />
6.      <item android:id="@+id/main_menu_1"
7.          android:icon="@drawable/pic1"
8.          android:title="新建" />
9.      <item android:id="@+id/main_menu_2"
10.         android:icon="@drawable/pic2"
11.         android:title="邮件" />
12.     <item android:id="@+id/main_menu_3"
13.         android:icon="@drawable/pic3"
14.         android:title="设置" />
15.     <item android:id="@+id/main_menu_4"
16.         android:icon="@drawable/pic4"
17.         android:title="订阅" />
18. </menu>
```

5.4 菜单

- **5.4.1菜单资源**
  - 代码生成具有5个子项的菜单
    - 代码第2行的menu是菜单的容器，菜单资源必须以menu作为根元素
    - 代码第3行item是菜单项，其属性值id、icon和title分别是菜单项的ID值、图标和标题
    - 代码第4行以及后续的代码中，定义了菜单项的图标，但菜单资源选择的菜单模式不同，菜单项图标可能会不显示
      - MenuResource示例使用的是选项菜单，因此菜单项的图标没有显示，而仅显示了菜单项的标题

5.4菜单

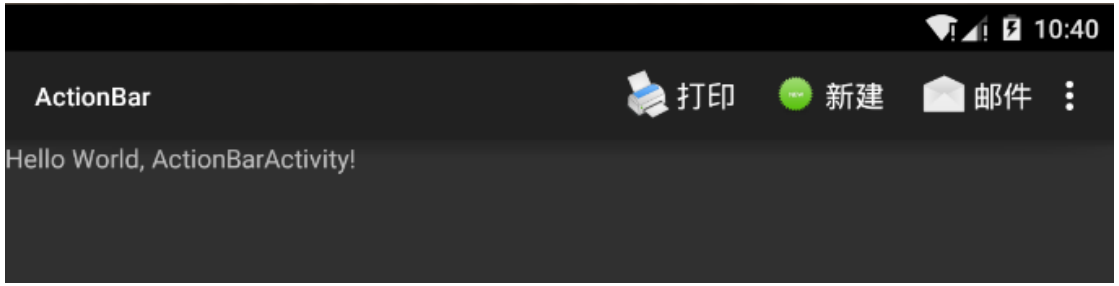
•5.4.2 选项菜单

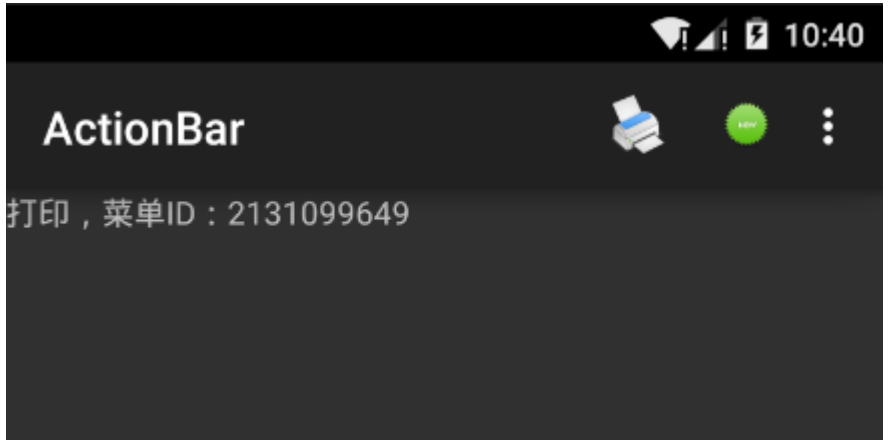
- 经常被使用的Android系统菜单，通过“菜单键”（MENU key）打开
- 在Android 2.3之前的系统中，选项菜单分为
  - 图标菜单（Icon Menu）
  - 浮动菜单（Overflow Menu）

5.4 菜单

•5.4.2 选项菜单

- 在Android 4.0系统中，选项菜单只出现浮动菜单，而没有图标菜单，图标菜单的功能由操作栏（ActionBar）代替实现





5.4 菜单

- **5.4.2 选项菜单**
  - 选项菜单的两个重要函数：
    - onCreateOptionsMenu()
      - Activity在创建时调用
      - 初始化自身的菜单系统
    - onOptionsItemSelected()
      - 在选择菜单项后，处理菜单选择事件

5.4 菜单

- **5.4.2 选项菜单**
  - 使用XML的菜单
  - 使用代码生成菜单

5.4 菜单

•5.4.2 选项菜单

- OptionMenu示例，使用XML文件生产选项菜单

```
1.  @Override
2.  public boolean onCreateOptionsMenu(Menu menu) {
3.  MenuInflater inflater = getMenuInflater();
4.  inflater.inflate(R.menu.main_menu, menu);
5.  return true;
6.  }
```

5.4 菜单

• 5.4.2 选项菜单

- OptionMenu2示例，使用Java代码生产选项菜单

```
1.    final static int MENU_00 = Menu.FIRST;
2.    final static int MENU_01 = Menu.FIRST+1;
3.    final static int MENU_02 = Menu.FIRST+2;
4.    final static int MENU_03 = Menu.FIRST+3;
5.    final static int MENU_04 = Menu.FIRST+4;
6.
7.    @Override
8.    public boolean onCreateOptionsMenu(Menu menu) {
9.                menu.add(0,MENU_00,0,"打印").setIcon(R.drawable.pic0);
10.               menu.add(0,MENU_01,1,"新建").setIcon(R.drawable.pic1);
11.               menu.add(0,MENU_02,2,"邮件").setIcon(R.drawable.pic2);
12.               menu.add(0,MENU_03,3,"设置").setIcon(R.drawable.pic3);
13.               menu.add(0,MENU_04,4,"订阅").setIcon(R.drawable.pic4);
14.               return true;15
15.           }
```

MenuItem android.view.Menu.add(int groupId, int itemId, int order, CharSequence title)

5.4 菜单

• 5.4.2 选项菜单

- onOptionsItemSelected()每次用户点击菜单子项时都会被调用
- 将菜单选择事件的响应代码放置在onOptionsItemSelected()中

```
1.  @Override
2.  public boolean onOptionsItemSelected(MenuItem item) {
3.  TextView label = (TextView)findViewById(R.id.label);
4.
5.  switch (item.getItemId()) {
6.  case R.id.main_menu_0:
7.  label.setText("打印，菜单ID：" + item.getItemId());
8.  return true;
9.  case R.id.main_menu_1:
10. label.setText("新建，菜单ID：" + item.getItemId());
11. return true;
12. case R.id.main_menu_2:
13. label.setText("邮件，菜单ID：" + item.getItemId());
14. return true;
15. case R.id.main_menu_3:
16. label.setText("设置，菜单ID：" + item.getItemId());
17. return true;
18. case R.id.main_menu_4:
19. label.setText("订阅，菜单ID：" + item.getItemId());
20. return true;
21. default:
22. return false;
23. }
24. }
```

5.4 菜单

- **5.4.3 子菜单**
  - 子菜单就是二级菜单，点击选项菜单或快捷菜单中的菜单项，就可以打开子菜单
  - Android系统使用浮动窗体的形式显示菜单子项，可以更好适应小屏幕的显示方式

5.4 菜单

(+)设置
  (-)打印
(+)新建
  (-)邮件
  (-)订阅

•5.4.3 子菜单

- SubMenu示例的菜单结构：
- XML菜单的代码

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <menu xmlns:android="http://schemas.android.com/apk/res/android">
3      <item android:id="@+id/main_menu_0"
4         android:icon="@drawable/pic0"
5         android:title="设置" >
6         <menu>
7            <item android:id="@+id/sub_menu_0_0"
8               android:icon="@drawable/pic4"
9                  android:title="打印" />
10        </menu>
11     </item>
12     <item android:id="@+id/main_menu_1"
13        android:icon="@drawable/pic1"
14        android:title="新建" >
15        <menu>
16           <item android:id="@+id/sub_menu_1_0"
```

5.4 菜单

•5.4.3 子菜单

```
17              android:icon="@drawable/pic2"
18              android:title="邮件" />
19          <item android:id="@+id/sub_menu_1_1"
20              android:icon="@drawable/pic3"
21              android:title="订阅" />
22      </menu>
23    </item>
24  </menu>
```

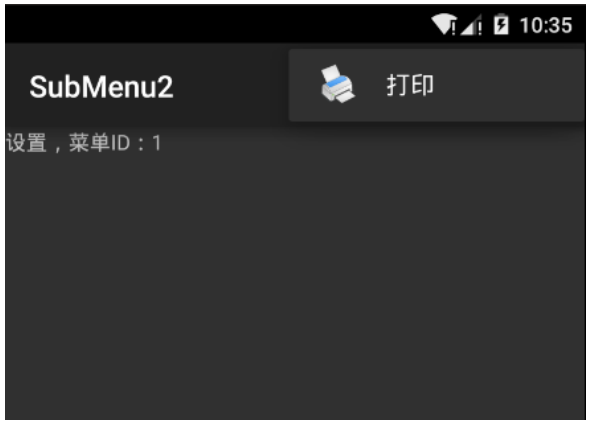
- 代码第6行至代码第10行是一个子菜单的描述。子菜单也使用<menu>标签进行声明，内部使用<item>标签描述菜单项

5.4 菜单

• 5.4.3 子菜单

- SubMenu2示例使用代码实现子菜单

```
1   final static int MENU_00 = Menu.FIRST;
2   final static int MENU_01 = Menu.FIRST+1;
3   final static int SUB_MENU_00_01 = Menu.FIRST+2;
4   final static int SUB_MENU_01_00 = Menu.FIRST+3;
5   final static int SUB_MENU_01_01 = Menu.FIRST+4;
6
7    SubMenu sub1 = (SubMenu) menu.addSubMenu(0,MENU_00,0,"设置")
8                .setHeaderIcon(R.drawable.pic3);
9    sub1.add(0,SUB_MENU_00_01 ,0,"打印").setIcon(R.drawable.pic0);
10
11   SubMenu sub2 = (SubMenu) menu.addSubMenu(0,MENU_01,1,"新建")
12               .setHeaderIcon(R.drawable.pic1);
13   sub2.add(0,SUB_MENU_01_00 ,0,"邮件").setIcon(R.drawable.pic2);
14   sub2.add(0,SUB_MENU_01_01 ,0,"订阅").setIcon(R.drawable.pic4);
```



5.4菜单

5.4.4快捷菜单

- 当用户点击界面元素超过2秒后，将启动注册到该界面元素的快捷菜单
- 类似于计算机程序中的“右键菜单”
- 与“选项菜单”的方法非常相似，需要重载：
  - onCreateContextMenu()函数
  - onContextItemSelected()函数
- 快捷菜单注册到界面中的某个控件上
  - registerForContextMenu()函数

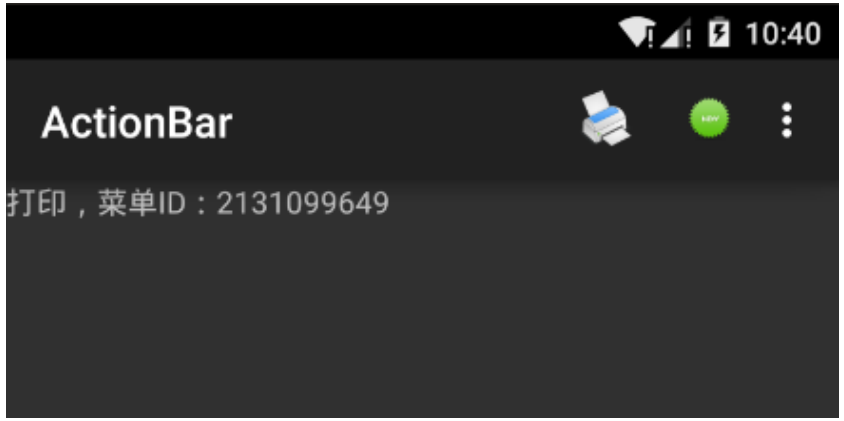
5.4菜单

- **5.4.4快捷菜单**
  - 使用registerForContextMenu()函数，将快捷菜单注册到TextView控件上

```
TextView LabelView = null;
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
LabelView = (TextView)findViewById(R.id.label);
registerForContextMenu(LabelView);
}
```

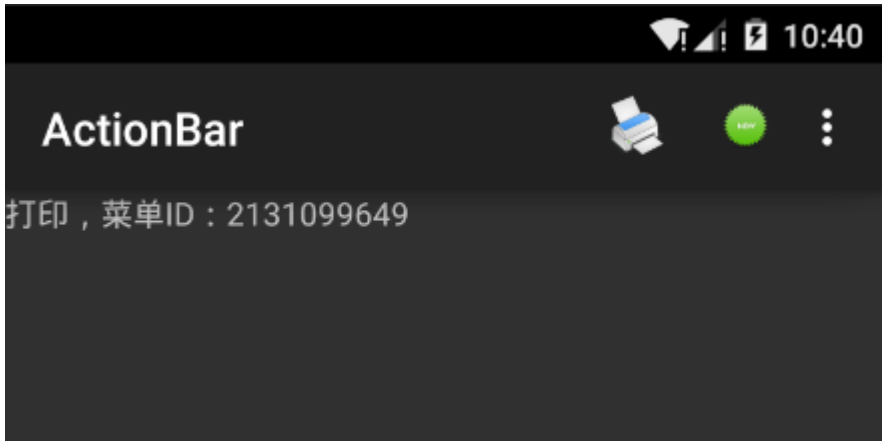
5.5 操作栏与Fragment

- 操作栏（ActionBar）和Fragment是Android 3.0新引入的界面控件，一定程度上是为了适应Android平板电脑等大屏幕设备界面设计需要而产生的
- 在Android 4.0系统中得到了进一步的发展，可以良好的支持不同屏幕尺寸的设备，并可以根据屏幕大小的不同改变显示内容



5.5 操作栏与Fragment

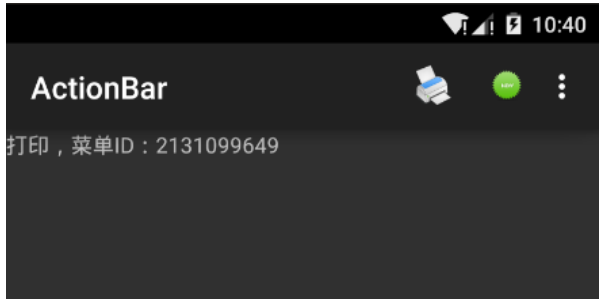
- **5.5.1操作栏**
  - 操作栏（Action Bar）代替传统的“标题栏”和“选项菜单”功能
  - 操作栏左侧的图标是应用程序的图标（Logo），图标旁边是应用程序当前Activity的标题，右侧的多个图标则是“选项菜单”中的菜单项

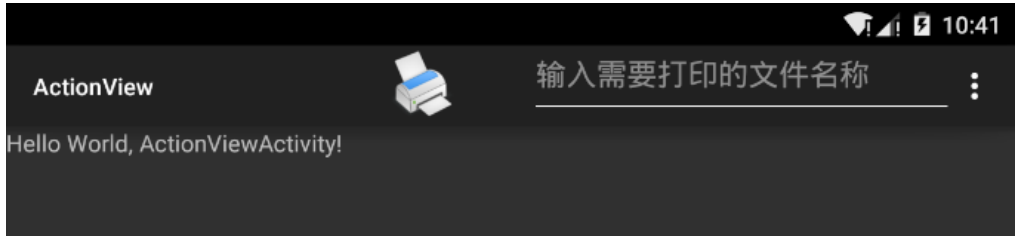


5.5 操作栏与Fragment

•5.5.1操作栏

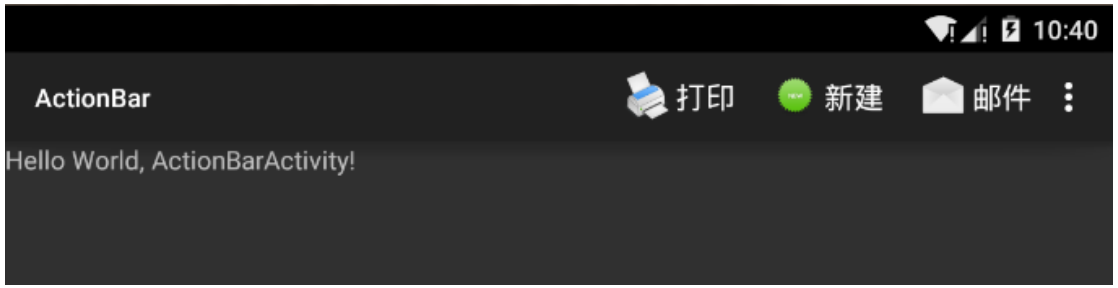
- 可以提供多个实用的功能：
  - （1）将“选项菜单”的菜单项显示在操作栏的右侧；
  - （2）基于Fragment实现类似于Tab页的导航切换功能；
  - （3）为导航提供可“拖拽—放置”的下拉列表；
  - （4）可在操作栏上实现类似于“搜索框”的功能。

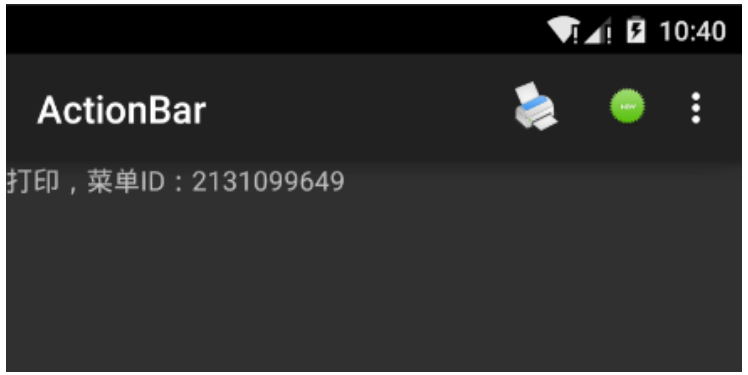




5.5 操作栏与Fragment

- **5.5.1操作栏与Fragment**
  - ActionBar示例
    - 操作栏的实际显示效果，取决于屏幕分辨率和屏幕方向





5.5 操作栏与Fragment

•5.5.1操作栏与Fragment

- ActionBar示例的main_menu.xml文件部分代码：

```
1.  <menu xmlns:android="http://schemas.android.com/apk/res/android"
2.      xmlns:tools="http://schemas.android.com/tools">
3.      <item android:id="@+id/main_menu_0"
4.          android:icon="@drawable/pic0"
5.          android:title="打印"
6.          android:showAsAction="ifRoom|withText"
7.          tools:ignore="AppCompatResource" />
8.  </menu>
```

- 第6行代码中的ifRoom表示如果操作栏有剩余空间，则显示该菜单项的图标；withText表示显示图标的同时显示文字标题

5.5 操作栏与Fragment

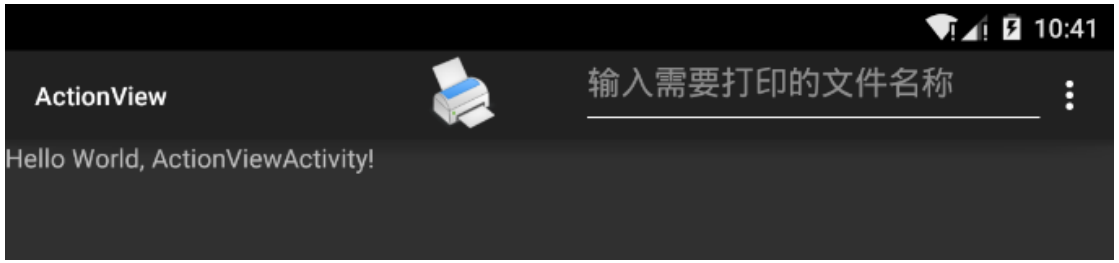
- **5.5.1操作栏与Fragment**
  - ActionBar示例的

main_menu.xml文件的完整代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <item android:id="@+id/main_menu_0"
        android:icon="@drawable/pic0"
        android:title="打印"
        android:showAsAction="ifRoom|withText"
        tools:ignore="AppCompatResource" />
    <item android:id="@+id/main_menu_1"
        android:icon="@drawable/pic1"
        android:title="新建"
        android:showAsAction="ifRoom|withText"
        tools:ignore="AppCompatResource" />
    <item android:id="@+id/main_menu_2"
        android:icon="@drawable/pic2"
        android:title="邮件"
        android:showAsAction="ifRoom|withText"
        tools:ignore="AppCompatResource" />
    <item android:id="@+id/main_menu_3"
        android:icon="@drawable/pic3"
        android:title="设置"
        android:showAsAction="ifRoom|withText"
        tools:ignore="AppCompatResource" />
    <item android:id="@+id/main_menu_4"
        android:icon="@drawable/pic4"
        android:title="订阅"
        android:showAsAction="ifRoom|withText"
        tools:ignore="AppCompatResource" />
</menu>
```

5.5 操作栏与Fragment

- **5.5.1操作栏**
  - ActionView示例
  - 在ActionBar示例基础上做的修改
  - 在操作栏上增加了文字输入功能



WXGA720 (1280x720）分辨率下的显示效果

5.5 操作栏与Fragment

•5.5.1操作栏与Fragment

- 在item标签中添加android:actionLayout属性，并将属性值定义为需要显示的布局文件

```
1.  <item android:id="@+id/main_menu_0"
2.          android:icon="@drawable/pic0"
3.          android:title="打印"
4.          android:showAsAction="ifRoom|withText"
5.          tools:ignore="AppCompatResource"
6.          android:actionLayout="@layout/printview" />
```

- 代码第6行表示显示该菜单项时，采用/layout/printview.xml文件作为自定义布局

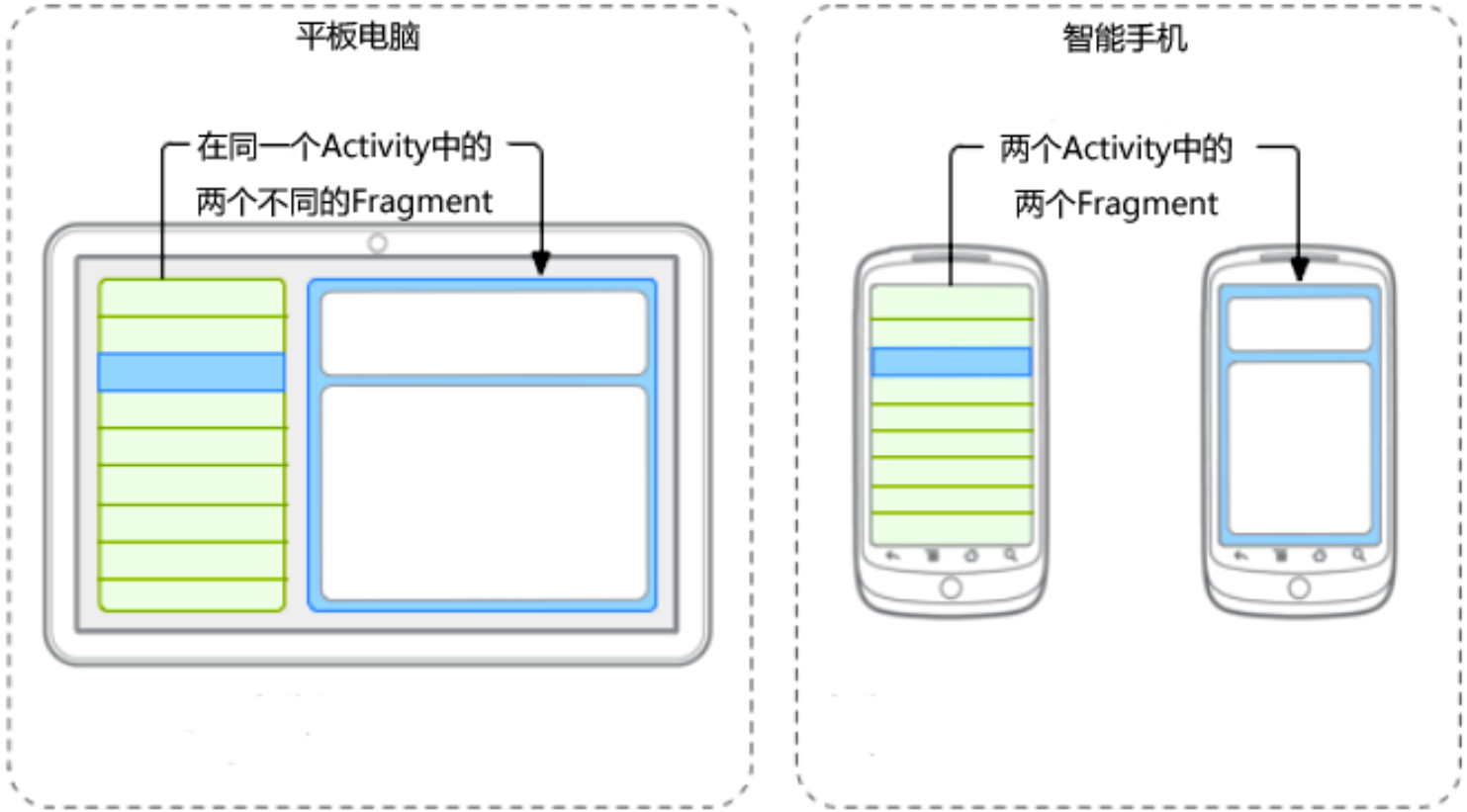
5.5 操作栏与Fragment

- **5.5.1操作栏与Fragment**
  - printview.xml文件的完整代码如下：

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      android:layout_width="wrap_content"
4.      android:layout_height="wrap_content"
5.      android:orientation="horizontal" >
6.      <ImageView
7.          android:layout_width="wrap_content"
8.          android:layout_height="wrap_content"
9.          android:src="@drawable/pic0" />
10.     <EditText
11.         android:layout_width="wrap_content"
12.         android:layout_height="wrap_content"
13.         android:hint="输入需要打印的文件名称"
14.         android:ems="12" />
15. </LinearLayout>
```

5.5 操作栏与Fragment

- **5.5.2 Fragment**
  - 用途是在大屏幕设备上实现灵活、动态的界面设计

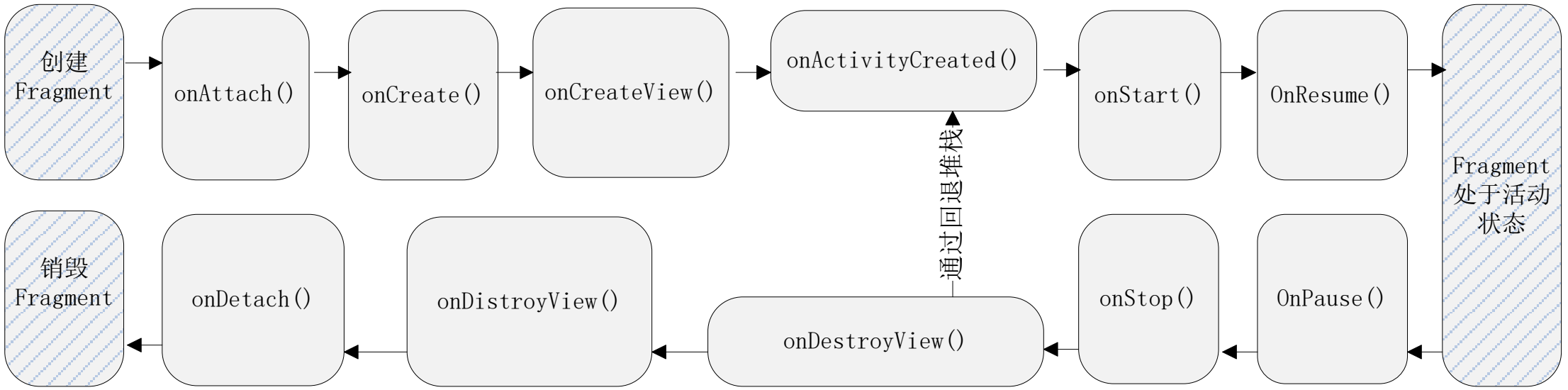


5.5 操作栏与Fragment

- **5.5.2 Fragment**
  - 可以被设计成为可重用模块的，因为每个Fragment都有自己的布局和生命周期回调函数，可以将同一个Fragment放置到多个不同的Activity中
  - 为了重复使用Fragment，应该避免直接从一个Fragment去操纵另一个Fragment，这样会增加两个Fragment之间的耦合度，不利于模块的重用

5.5 操作栏与Fragment

- **5.5.2 Fragment**
  - Fragment具有与Activity类似的生命周期，但比Activity支持更多的事件回调函数

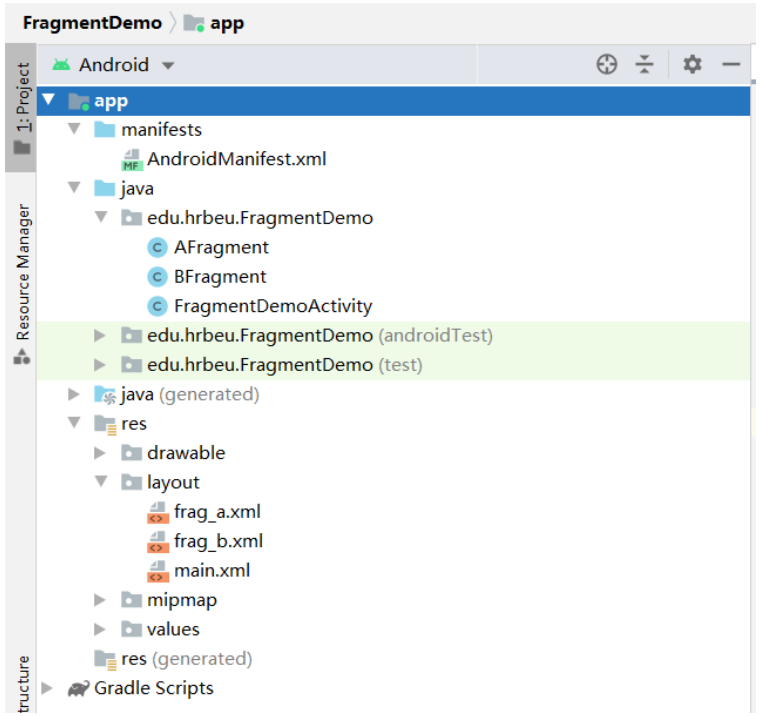


5.5 操作栏与Fragment

- **5.5.2 Fragment**
  - 通常情况下，创建Fragment需要继承Fragment的基类，并至少应实现onCreate()、onCreateView()和onPause()三个生命周期的回调函数
    - onCreate()函数是在Fragment创建时被调用，用来初始化Fragment中的必要组件
    - onCreateView()函数是Fragment在用户界面上第一次绘制时被调用，并返回Fragment的根布局视图
    - onPause()函数是在用户离开Fragment时被调用，用来保存Fragment中用户输入或修改的内容
  - 如果仅通过Fragment显示元素，而不进行任何的数据保存和界面事件处理，则可仅实现onCreateView()函数

5.5 操作栏与Fragment

- **5.5.2 Fragment**
  - FragmentDemo示例
    - 说明如何在一个Activity中同时加载两个Fragment



5.5 操作栏与Fragment

- **5.5.2 Fragment**
  - FragmentDemo示例
    - main.xml文件是Activity的布局文件，两个Fragment在界面上的位置关系就在这个文件中进行的定义

```
1.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.     android:orientation="horizontal"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent">
5.     <fragment android:name ="edu.hrbeu.FragmentDemo.AFragment"
6.           android:id="@+id/fragment_a"
7.           android:layout_weight="1"
8.           android:layout_width="0px"
9.           android:layout_height="match_parent" />
10.     <fragment android:name ="edu.hrbeu.FragmentDemo.BFragment"
11.          android:id="@+id/fragment_b"
12.          android:layout_weight="1"
13.          android:layout_width="0px"
14.          android:layout_height="match_parent" />
15. </LinearLayout>
```

5.5 操作栏与Fragment

• 5.5.2 Fragment

- FragmentDemo示例
  - FragmentDemoActivity是该示例主界面的Activity，加载了main.xml文件声明的界面布局
  - FragmentDemoActivity.java文件的完整代码如下:

```
1.  public class FragmentDemoActivity extends Activity {
2.      @Override
3.      public void onCreate(Bundle savedInstanceState) {
4.          super.onCreate(savedInstanceState);
5.          setContentView(R.layout.main);
6.      }
7.  }
```

  - Android系统会根据代码第5行的内容加载界面布局文件main.xml，然后通过main.xml文件中对Fragment所在的“包+类”的描述，找到Fragment的实现类，并调用类中的onCreateView()函数绘制界面元素

5.5 操作栏与Fragment

- **5.5.2 Fragment**
  - FragmentDemo示例
    - AFragment.java文件的核心代码如下:

```
1.  public class AFragment extends Fragment{
2.  @Override
3.      public View onCreateView(LayoutInflater inflater, ViewGroup container,
4.                               Bundle savedInstanceState) {
5.          return inflater.inflate(R.layout.frag_a, container, false);
6.      }
7.  }
```

    - AFragment中只实现了onCreateView()函数(代码第3行)，返回值是AFragment的视图
    - 代码第5行使用inflate()函数，通过指定资源文件R.layout.frag_a，获取到AFragment的视图

5.5 操作栏与Fragment

- **5.5.2 Fragment**
  - FragmentDemo示例
    - 最后给出frag_a.xml文件的全部代码如下:

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      android:layout_width="wrap_content"
4.      android:layout_height="wrap_content"
5.      android:orientation="vertical" >
6.      <TextView
7.          android:layout_width="wrap_content"
8.          android:layout_height="wrap_content"
9.          android:text="AFragment" />
10.     <TextView
11.         android:layout_width="wrap_content"
12.         android:layout_height="wrap_content"
13.         android:text="这是AFragment的显示区域，通过这行文字可以看到与BFragment的边界" />
14.     <CheckBox
15.         android:layout_width="wrap_content"
16.         android:layout_height="wrap_content"
17.         android:text="AF选项" />
18.     <Button
19.         android:layout_width="wrap_content"
20.         android:layout_height="wrap_content"
21.         android:text="AF按钮" />
22. </LinearLayout>
```

5.5 操作栏与Fragment

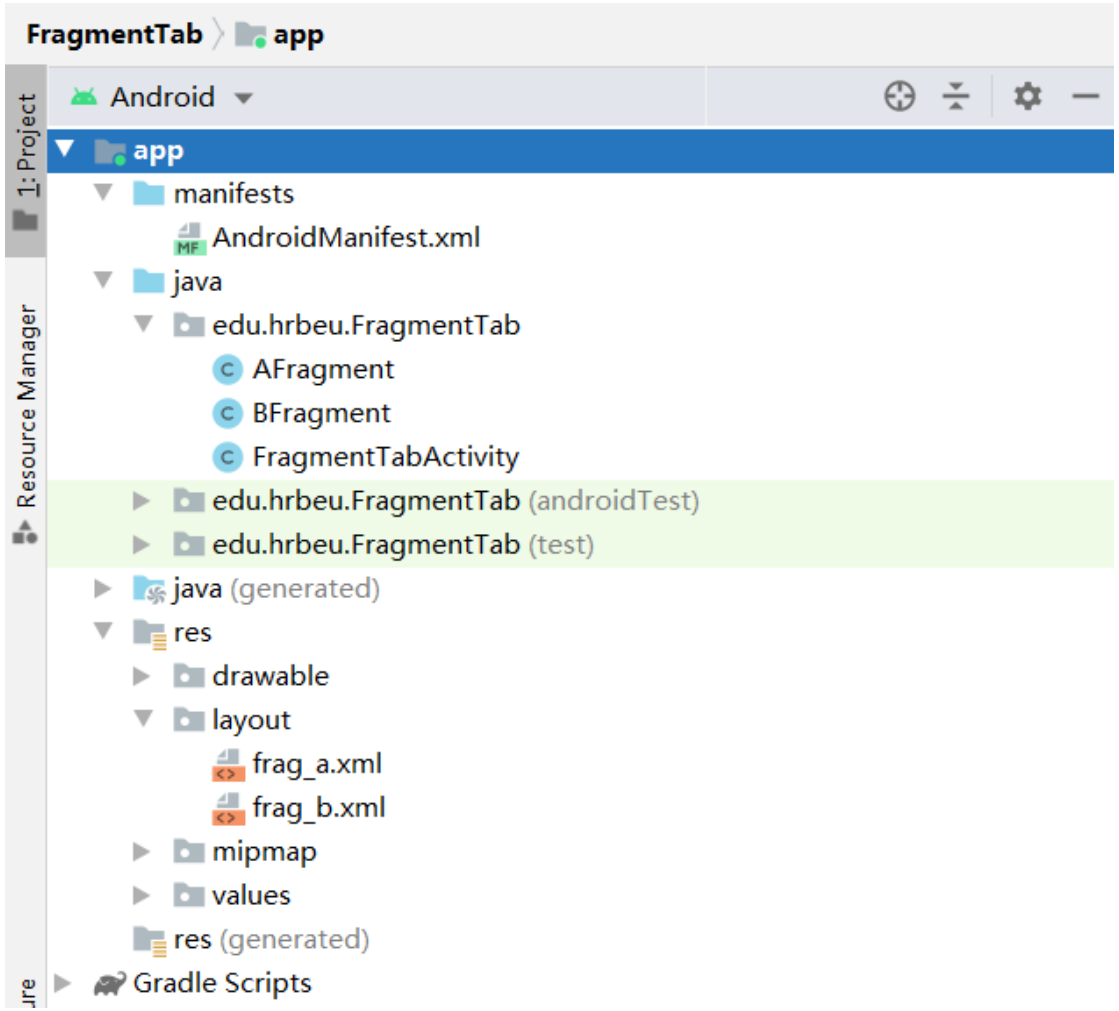
• 5.5.2 Tab导航栏

- 在界面控件的章节中介绍过使用TabHost和TabActivity实现Tab导航栏的功能，但因为TabActivity已经过期，所以这里介绍一种新方法，使用操作栏和Fragment实现Tab导航栏。
- 下面用FragmentTab示例说明如何使用操作栏和Fragment实现Tab导航栏，FragmentTab示例的用户界面如图:

5.5 操作栏与Fragment

• 5.5.2 Tab导航栏

- Tab导航栏介绍
  - 第一个Tab页的标题为“FRAGMENT A”，第二个Tab页的标题为“FRAGMENT B”，两个Tab页分别加载了不同Fragment，两个Fragment所显示的界面元素略有不同。从图5.35的文件结构可以看得出来，FragmentTab示例和FragmentDemo示例中的一部分文件的文件名称是完全相同的，这些文件中的代码也是完全相同的。这些文件包括AFragment.java、BFragment.java、frag_a.xml和frag_b.xml。这里就不再给出上述文件的源代码，读者可以参考FragmentDemo示例。



5.5 操作栏与Fragment

• 5.5.2 Tab导航栏

- Tab导航栏介绍
  - 建立Tab导航栏代码，以及将导航栏和Fragment关联起来的代码都在FragmentTabActivity.java文件中。下面分别介绍FragmentTabActivity.java文件中的核心函数。
  - 先给出onCreate()函数的代码：

```
@Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        final ActionBar bar = getActionBar();
        bar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
        bar.setDisplayOptions(0, ActionBar.DISPLAY_SHOW_TITLE);
        bar.addTab(bar.newTab()
                .setText("Fragment A")
                .setTabListener(new TabListener<AFragment>(
                        this, "fa",AFragment.class)));
        bar.addTab(bar.newTab()
                .setText("Fragment B")
                .setTabListener(new TabListener<BFragment>(
                    this, "fb", BFragment.class)));
        if (savedInstanceState != null) {
            bar.setSelectedNavigationItem(savedInstanceState.getInt("tab", 0));
        }
    }
```

5.5 操作栏与Fragment

- **5.5.2 Tab导航栏**
  - Tab导航栏介绍
    - 代码第5行调用getActionBar()获取操作栏实例。
    - 代码第6行将操作栏的导航模式设置为Tab导航栏模式，NAVIGATION_MODE_TABS常量的值为2。还支持的常量包括NAVIGATION_MODE_LIST（值为1）和NAVIGATION_MODE_STANDARD（值为0），分别表示列表导航栏和标准导航栏。
    - 代码第7行用来设置操作栏的显示选项。setDisplayOptions(int options, int mask)函数的options参数表示显示的内容，而mask参数则表示不显示的内容。第7行代码的意思是关闭“显示标题文字(DISPLAY_SHOW_TITLE)”。setDisplayOptions()函数支持的常量如表5.4所示。
    - 代码第9行使用add()函数添加Tab页，代码第10行设置Tab页的标题，代码第11行定义这是Tab页点击事件的监听函数。
    - 代码第16行和第17行，表明如果Activity不是首次启动，则在savedInstanceState变量中获取当前Tab页的索引号。

5.5 操作栏与Fragment

5.5.2 Tab导航栏

Tab导航栏介绍

setDisplayOptions()函数支持的常量

常量	值	说明
DISPLAY_HOME_AS_UP	4	在Home元素左侧显示回退按钮
DISPLAY_SHOW_CUSTOM	16	显示自定义视图
DISPLAY_SHOW_HOME	2	在操作栏中显示Home元素
DISPLAY_SHOW_TITLE	8	显示Activity的标题
DISPLAY_USE_LOGO	1	使用Logo代替程序图标

5.5 操作栏与Fragment

•5.5.2 Tab导航栏

- Tab导航栏介绍
  - onSaveInstanceState()函数在Activity临时推出时，将当前Tab页的索引号保存在Bundle中，代码如下：

```
1.      @Override
2.       protected void onSaveInstanceState(Bundle outState) {
3.          super.onSaveInstanceState(outState);
4.          outState.putInt("tab", getActionBar().getSelectedNavigationIndex());
5.       }
```

  - 构造Tab导航栏的事件监听函数，必须实现ActionBar.TabListener接口，主要是实现接口中3个函数，分别是onTabSelected()、onTabUnselected()和onTabReselected()。onTabSelected()在当前Tab页被选中时调用，onTabUnselected()在其它Tab页被选中时调用，onTabReselected()在当前Tab页被再次选中时调用。

5.5 操作栏与Fragment

- **5.5.2 Tab导航栏**
  - Tab导航栏介绍
    - 静态类TabListener的代码如下：

```
1    public static class TabListener<T extends Fragment> implements ActionBar.TabListener {
2        private final Activity mActivity;
3        private final String mTag;
4        private final Class<T> mClass;
5        private final Bundle mArgs;
6        private Fragment mFragment;
7
8        public TabListener(Activity activity, String tag, Class<T> clz) {
9            this(activity, tag, clz, null);
10       }
11
12       public TabListener(Activity activity, String tag, Class<T> clz, Bundle args) {
13           mActivity = activity;
14           mTag = tag;
15           mClass = clz;
16           mArgs = args;
17
18           mFragment = mActivity.getFragmentManager().findFragmentByTag(mTag);
19           if (mFragment != null && !mFragment.isDetached()) {
```

5.5 操作栏与Fragment

5.5.2 Tab导航栏

Tab导航栏介绍

静态类TabListener的代码如下：

```
20              FragmentTransaction ft = mActivity.getFragmentManager().beginTransaction();
21              ft.detach(mFragment);
22              ft.commit();
23          }
24      }
25
26      public void onTabSelected(Tab tab, FragmentTransaction ft) {
27          if (mFragment == null) {
28              mFragment = Fragment.instantiate(mActivity, mClass.getName(), mArgs);
29              ft.add(android.R.id.content, mFragment, mTag);
30          } else {
31              ft.attach(mFragment);
32          }
33      }
34
35      public void onTabUnselected(Tab tab, FragmentTransaction ft) {
36          if (mFragment != null) {
37              ft.detach(mFragment);
38          }
39      }
40
41      public void onTabReselected(Tab tab, FragmentTransaction ft) {
42          Toast.makeText(mActivity, "Reselected!", Toast.LENGTH_SHORT).show();
43      }
44  }
```

5.5 操作栏与Fragment

- **5.5.2 Tab导航栏**
  - Tab导航栏介绍
    - 静态类TabListener的代码
      - FragmentTransaction是封装了Fragment变换所要用的函数，包括将Fragment加入到Activity的add()函数，将Fragment从当前界面分离的Detach()函数，将被Detach()函数分离的Fragment重新连接到界面的attach()函数
      - 上面的代码具有一定的难度，部分内容涉及到Java泛型编程的内容，例如代码第1行和第12行，读者可以参考Java语言的相关资料

5.6 界面事件

- **5.6.1 按键事件**
  - 在MVC模型中，控制器根据界面事件（UI Event）类型不同，将事件传递给界面控件不同的事件处理函数
    - 按键事件（KeyEvent）将传递给onKey()函数进行处理
    - 触摸事件（TouchEvent）将传递给onTouch()函数进行处理

5.6 界面事件

- **5.6.1 按键事件**
  - Android系统界面事件的传递和处理遵循一的规则
    - 如果界面控件设置了事件监听器，则事件将先传递给事件监听器
    - 如果界面控件没有设置事件监听器，界面事件则会直接传递给界面控件的其他事件处理函数
    - 即使界面控件设置了事件监听器，界面事件也可以再次传递给其他事件处理函数

5.6 界面事件

•5.6.1 按键事件

- Android系统界面事件的传递和处理遵循一的规则
  - 是否继续传递事件给其他处理函数是由事件监听器处理函数的返回值决定的
  - 如果监听器处理函数的返回值为true，表示该事件已经完成处理过程，不需要其他处理函数参与处理过程，这样事件就不会再继续进行传递
  - 如果监听器处理函数的返回值为false，则表示该事件没有完成处理过程，或需要其他处理函数捕获到该事件，事件会被传递给其他的事件处理函数

5.6 界面事件

- **5.6.1 按键事件**
  - 以EditText控件中的按键事件为例，说明Android系统界面事件传递和处理过程，假设EditText控件已经设置了按键事件监听器
    - 当用户按下键盘上的某个按键时，控制器将产生KeyEvent按键事件
    - Android系统会首先判断EditText控件是否设置了按键事件监听器，因为EditText控件已经设置按键事件监听器OnKeyListener，所以按键事件先传递到监听器的事件处理函数onKey()中

5.6 界面事件

- **5.6.1 按键事件**
  - 事件能够继续传递给EditText控件的其他事件处理函数，完全根据onKey()函数的返回值来确定
  - 如果onKey()函数返回false，事件将继续传递，这样EditText控件就可以捕获到该事件，将按键的内容显示在EditText控件中
  - 如果onKey()函数返回true，将阻止按键事件的继续传递，这样EditText控件就不能够捕获到按键事件，也就不能够将按键内容显示在EditText控件中

5.6 界面事件

•5.6.1 按键事件

- Android界面框架支持对按键事件的监听，并能够将按键事件的详细信息传递给处理函数
- 为了处理控件的按键事件，先需要设置按键事件的监听器，并重载onKey()函数
- 示例代码如下：

```
1. entryText.setOnKeyListener(new OnKeyListener(){
2. @Override
3. public boolean onKey(View view, int keyCode, KeyEvent keyEvent) {
4. //过程代码……
5. return true/false;
6. }
```

5.6 界面事件

- **5.6.1 按键事件**
  - 第1行代码是设置控件的按键事件监听器
  - 第3行代码的onKey ()函数中的参数
    - 第1个参数view表示产生按键事件的界面控件
    - 第2个参数keyCode表示按键代码
    - 第3个参数keyEvent则包含了事件的详细信息，如按键的重复次数、硬件编码和按键标志等
  - 第5行代码是onKey ()函数的返回值
    - 返回true，阻止事件传递
    - 返回false，允许继续传递按键事件

5.6 界面事件

5.6.1 按键事件

- KeyEventDemo是说明如何处理按键事件的示例
- KeyEventDemo用户界面
  - 最上方的EditText控件是输入字符的区域
  - 中间的CheckBox控件用来控制onKey()函数的返回值
  - 最下方的TextView控件用来显示按键事件的详细信息
    - 按键动作
    - 按键代码
    - 按键字符
    - Unicode编码
    - 重复次数
    - 功能键状态
    - 硬件编码
    - 按键标志

5.6 界面事件

• 5.6.1 按键事件

- 在EditText中，每当任何一个键子按下或抬起时，都会引发按键事件
- 为使EditText处理按键事件，需要使用setOnKeyListener ()函数在代码中设置按键事件监听器，并在onKey()函数添加按键事件的处理过程

```
1. entryText.setOnKeyListener(new OnKeyListener(){
2. @Override
3. public boolean onKey(View view, int keyCode, KeyEvent keyEvent) {
4. int metaState = keyEvent.getMetaState();
5. int unicodeChar = keyEvent.getUnicodeChar();
6. String msg = "";
```

5.6 界面事件

•5.6.1 按键事件

```
7.  msg +="按键动作:" + String.valueOf(keyEvent.getAction())+"\n";
8.  msg +="按键代码:" + String.valueOf(keyCode)+"\n";
9.  msg +="按键字符:" + (char)unicodeChar+"\n";
10. msg +="UNICODE:" + String.valueOf(unicodeChar)+"\n";
11. msg +="重复次数:" + String.valueOf(keyEvent.getRepeatCount())+"\n";
12. msg +="功能键状态:" + String.valueOf(metaState)+"\n";
13. msg +="硬件编码:" + String.valueOf(keyEvent.getScanCode())+"\n";
14. msg +="按键标志:" + String.valueOf(keyEvent.getFlags())+"\n";
15. labelView.setText(msg);
16. if (checkBox.isChecked())
17. return true;
18. else
19. return false;
20. }
```

5.6 界面事件

- **5.6.1 按键事件**
  - 第4行代码用来获取功能键状态。功能键包括左Alt键、右Alt键和Shift键，当这三个功能键被按下时，功能键代码metaState值分别为18、34和65；但没有功能键被按下时，功能键代码metaState值分别为0
  - 第5行代码获取了按键的Unicode值，在第9行中，将Unicode转换为字符，显示在TextView中
  - 第7行代码获取了按键动作，0表示按下按键，1表示抬起按键。第7行代码获取按键的重复次数，但按键被长时间按下时，则会产生这个属性值
  - 第13行代码获取了按键的硬件编码，不同硬件设备的按键硬件编码都不相同，因此该值一般用于调试
  - 第14行获取了按键事件的标志符

5.6 界面事件

•5.6.2触摸事件

- Android界面框架支持对触摸事件的监听，并能够将触摸事件的详细信息传递给处理函数
- 需要设置触摸事件的监听器，并重载onTouch ()函数

```
1.  touchView.setOnTouchListener(new View.OnTouchListener(){
2.  @Override
3.  public boolean onTouch(View v, MotionEvent event) {
4.  //过程代码……
5.  return true/false;
6.  }
```

  - 第1行代码是设置控件的触摸事件监听器
  - 在代码第3行的onTouch()函数中，第1个参数View表示产生触摸事件的界面控件；第2个参数MontionEvent表示触摸事件的详细信息，如产生时间、坐标和触点压力等
  - 第5行是onTouch()函数的返回值

5.6 界面事件

•5.6.2触摸事件

- TouchEventDemo是一个说明如何处理触摸事件的示例
- TouchEventDemo用户界面
  - 浅蓝色区域是可以接受触摸事件的区域，用户可以在Android模拟器中使用鼠标点击屏幕，用以模拟触摸手机屏幕
  - 下方黑色区域是显示区域，用来显示触摸事件的类型、相对坐标、绝对坐标、触点压力、触点尺寸和历史数据量等信息

5.6 界面事件

- **5.6.2触摸事件**
  - 在用户界面中使用了线性布局，并加入了3个TextView控件
    - 第1个TextView（ID为touch_area）用来标识触摸事件的测试区域
    - 第2个TextView（ID为history_label）用来显示触摸事件的历史数据量
    - 第3个TextView（ID为event_label）用来触摸事件的详细信息，包括类型、相对坐标、绝对坐标、触点压力和触点尺寸

5.6 界面事件

- **5.6.2触摸事件**
  - XML文件的代码如下:

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.  android:orientation="vertical"
4.  android:layout_width=“match_parent"
5.  android:layout_height=“match_parent">
6.  <TextView  android:id="@+id/touch_area"
7.  android:layout_width=“match_parent"
8.  android:layout_height="300dip"
9.  android:background="#0FF"
10. android:textColor="#FFFFFF"
11. android:text="触摸事件测试区域">
12. </TextView>
```

5.6 界面事件

•5.6.2触摸事件

```
13.  <TextView android:id="@+id/history_label"
14.                         android:layout_width="wrap_content"
15.  android:layout_height="wrap_content"
16.  android:text="历史数据量：" >
17.  </TextView>
18.  <TextView android:id="@+id/event_label"
19.  android:layout_width="wrap_content"
20.  android:layout_height="wrap_content"
21.  android:text="触摸事件：" >
22.  </TextView>
23.   </LinearLayout>
```

- 第9行代码定义了TextView的背景颜色，#80A0FF是颜色代码
- 第10行代码定义了TextView的字体颜色

5.6 界面事件

- **5.6.2触摸事件**
  - 在代码中为了能够引用XML文件中声明的界面元素，使用了下面的代码

```
1. TextView labelView = null;
2. labelView = (TextView)findViewById(R.id.event_label);
3. TextView touchView = (TextView)findViewById(R.id.touch_area);
4. final TextView historyView = (TextView)findViewById(R.id.history_label);
```

5.6 界面事件

- **5.6.2触摸事件**
  - 当手指接触到触摸屏并且在触摸屏上移动或离开触摸屏时，分别会引发ACTION_DOWN、ACTION_UP和ACTION_MOVE触摸事件，而无论是哪种触摸事件，都会调用onTouch()函数进行处理
  - 事件类型包含在onTouch()函数的MotionEvent参数中，可以通过getAction()函数获取到触摸事件的类型，然后根据触摸事件的不同类型进行不同的处理
  - 为了能够使屏幕最上方的TextView处理触摸事件，需要使用setOnTouchListener()函数在代码中设置触摸事件监听器，并在onTouch()函数添加触摸事件的处理过程

5.6 界面事件

- **5.6.2触摸事件**

```
touchView.setOnTouchListener(new View.OnTouchListener(){
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            int action = event.getAction();
            switch (action) {
              case (MotionEvent.ACTION_DOWN):
              Display("ACTION_DOWN",event);
              break;
              case (MotionEvent.ACTION_UP):
              int historySize = ProcessHistory(event);
              historyView.setText("历史数据量："+historySize);
              Display("ACTION_UP",event);
              break;
              case (MotionEvent.ACTION_MOVE):
              Display("ACTION_MOVE",event);
              break;
            }
```

5.6 界面事件

•5.6.2触摸事件

```
18.                 return true;
19.             }
20.         });
```

- 第7行代码的Display()是一个自定义函数，主要用来显示触摸事件的详细信息，函数的代码和含义将在后面进行介绍
- 第10行代码的ProcessHistory()也是一个自定义函数，用来处理触摸事件的历史数据，后面进行介绍
- 第11行代码是使用TextView显示历史数据的数量

5.6 界面事件

•5.6.2触摸事件

- MotionEvent参数中不仅有触摸事件的类型信息，还触点的坐标信息，获取方法是使用getX()和getY()函数，这两个函数获取到的是触点相对于父界面元素的坐标信息。如果需要获取绝对坐标信息，则可使用getRawX()和getRawY()函数
- 触点压力是一个介于0和1之间的浮点数，用来表示用户对触摸屏施加压力的大小，接近0表示压力较小，接近1表示压力较大，获取触摸事件触点压力的方式是调用getPressure()函数
- 触点尺寸指用户接触触摸屏的接触点大小，也是一个介于0和1之间的浮点数，接近0表示尺寸较小，接近1表示尺寸较大，可以使用getSize()函数获取

5.6 界面事件

•5.6.2触摸事件

- Display()将MotionEvent参数中的事件信息提取出来，并显示在用户界面上

```
1.  private void Display(String eventType, MotionEvent event){
2.      int x = (int)event.getX();
3.      int y = (int)event.getY();
4.      float pressure = event.getPressure();
5.      float size = event.getSize();
6.      int RawX = (int)event.getRawX();
7.      int RawY = (int)event.getRawY();
8.
9.      String msg = "";
10.     msg += "事件类型：" + eventType + "\n";
11.     msg += "相对坐标："+String.valueOf(x)+","+String.valueOf(y)+"\n";
12.     msg += "绝对坐标："+String.valueOf(RawX)+","+String.valueOf(RawY)+"\n";
13.     msg += "触点压力："+String.valueOf(pressure)+"，  ";
14.     msg += "触点尺寸："+String.valueOf(size)+"\n";
15.     labelView.setText(msg);
16.     }
```

5.6 界面事件

- **5.6.2触摸事件**
  - 一般情况下，若用户将手指放在触摸屏上，但不移动，然后抬起手指，应先后产生ACTION_DOWN和ACTION_UP两个触摸事件
  - 但如果用户在屏幕上移动手指，然后再抬起手指，则会产生这样的事件序列：ACTION_DOWN → ACTION_MOVE → ACTION_MOVE → ACTION_MOVE → ……→ ACTION_UP

5.6 界面事件

•5.6.2触摸事件

- 在手机上运行的应用程序，效率是非常重要的。如果Android界面框架不能产生足够多的触摸事件，则应用程序就不能够很精确的描绘触摸屏上的触摸轨迹
- 如果Android界面框架产生了过多的触摸事件，虽然能够满足精度的要求，但却降低了应用程序效率
- Android界面框架使用了“打包”的解决方法。在触点移动速度较快时会产生大量的数据，每经过一定的时间间隔便会产生一个ACTION_MOVE事件，在这个事件中，除了有当前触点的相关信息外，还包含这段时间间隔内触点轨迹的历史数据信息，这样既能够保持精度，又不至于产生过多的触摸事件

5.6 界面事件

•5.6.2触摸事件

- 通常情况下，在ACTION_MOVE的事件处理函数中，都先处理历史数据，然后再处理当前数据

```
1.  private int ProcessHistory(MotionEvent event)
2.      {
3.      int historySize = event.getHistorySize();
4.      for (int i = 0; i < historySize; i++) {
5.      long time = event.getHistoricalEventTime(i);
6.      float pressure = event.getHistoricalPressure(i);
7.      float x = event.getHistoricalX(i);
8.      float y = event.getHistoricalY(i);
9.      float size = event.getHistoricalSize(i);
10.
11.     // 处理过程.....
12.     }
13.     return historySize;
14.     }
```

5.6 界面事件

- **5.6.2触摸事件**
  - 第3行代码获取了历史数据的数量
  - 然后在第4行至12行中循环处理这些历史数据
  - 第5行代码获取了历史事件的发生时间
  - 第6行代码获取历史事件的触点压力
  - 第7行和第8行代码获取历史事件的相对坐标
  - 第9行获取历史事件的触点尺寸
  - 在第14行返回历史数据的数量，主要是用于界面显示
- Android模拟器并不支持触点压力和触点尺寸的模拟，所有触点压力恒为0.50390625
- 同时Android模拟器上也无法产生历史数据，因此历史数据量一直显示为0

习题：

- 1.简述6种界面布局的特点。
- 2.参考下图中界面控件的摆放位置，使用多种布局方法实现用户界面，并对比各种布局实现的复杂程度和对不同屏幕尺寸的适应能力。

- 3.简述Android系统三种菜单的特点及其使用方式。
- 4.说明使用操作栏为程序开发所带来的便利。