第 24 章

样式与include标签的使用

应用开发时，虽然首要任务是保证应用能够正常运行，但应用的外在观感和使用体验如何，也应引起重视。具有优雅漂亮UI的应用往往能够从海量应用的市场中脱颖而出。

即使有图形设计师设计好的UI，我们也必须有效使用它们。因此，本章我们来会熟悉一些设计工具，并利用它们独立快速地进行应用原型设计。这样，在与图形设计师合作时，可以更清楚自身对UI的需求，并有效利用设计师们创造的资源。

接下来的两章，我们将创建一个电视遥控应用。不过，这只是一个虚拟应用，主要用于练习使用设计工具。实际上，它什么也控制不了。本章，我们将使用样式（style）和include来创建如图24-1所示的遥控应用。

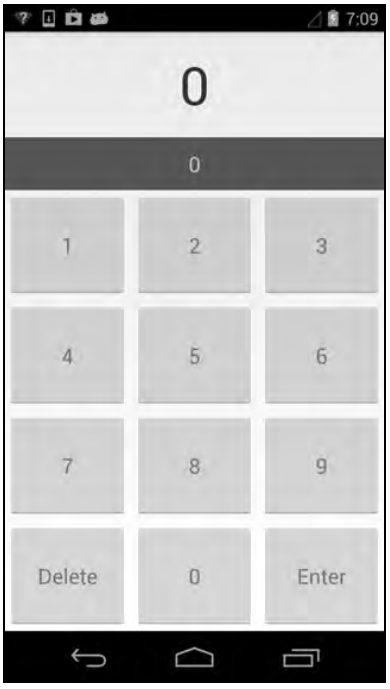

图24-1　RemoteControl应用

RemoteControl应用只有一个activity。应用界面的最顶端区域可显示当前频道。下面紧接着的区域可显示用户正在输入的新频道。点击Delete按钮可清除输入区域的频道。点击Enter按钮可变换频道，即更新显示当前频道并清除输入区域的频道。

24.1　创建 RemoteControl 项目

新建一个Android应用项目，并参照图24-2对其进行配置。

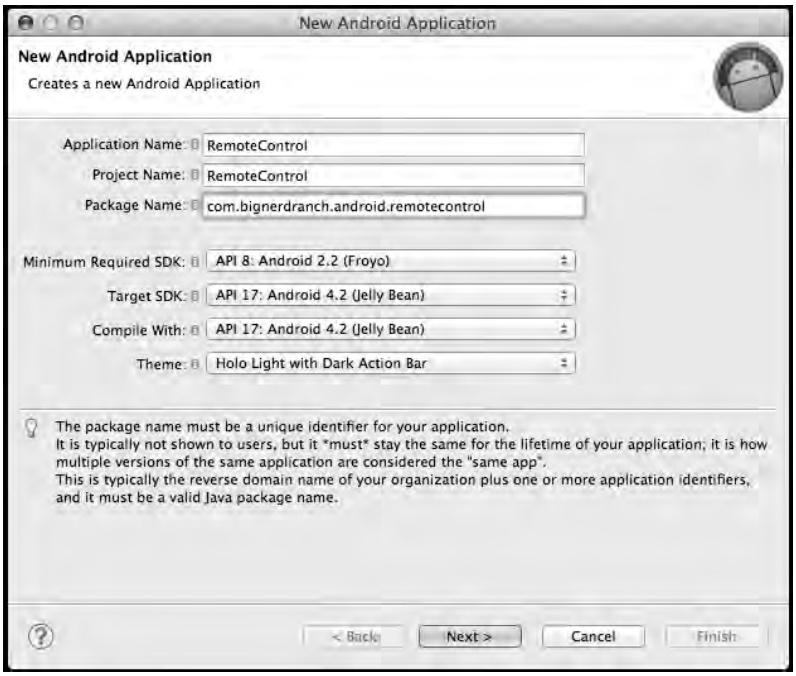

图24-2　新建RemoteControl项目

然后通过新建应用向导，创建一个名为RemoteControlActivity的空白activity。

24.1.1　编码实现RemoteControlActivity

RemoteControlActivity将会继承SingleFragmentActivity抽象类。因此，首先需将CriminalIntent项目中的SingleFragmentActivity.java复制到com.bignerdranch.android.remotecontrol包中。然后再将activity_fragment.xml文件复制到RemoteControl项目的res/layout目录中。

打开RemoteControlActivity.java文件，调整RemoteControlActivity的父类为SingleFragmentActivity类并实现父类的createFragment()方法以创建RemoteControlFragment（稍后将创建该fragment类）。最后，覆盖RemoteControlActivity.onCreate(...)方法，隐藏activity的操作或标题栏，如代码清单24-1所示栏。

代码清单24-1 RemoteControlActivity的编码实现（RemoteControlActivity.java）

```
public class RemoteControlActivity extends Activity SingleFragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_remote_control);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_remote_control, menu);
        return true;
    }

    @Override
    protected Fragment createFragment() {
        return new RemoteControlFragment();
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        super.onCreate(savedInstanceState);
    }
}
```

接下来，打开AndroidManifest.xml文件，限制activity的视图以竖直方向展现，如代码清单24-2所示。

代码清单24-2 锁定activity的视图以竖直方向展现（AndroidManifest.xml）

```
<activity
  android:name="com.bignerdranch.android.remotecontrol.RemoteControlActivity"
  android:label="@string/app_name"
  android:screenOrientation="portrait">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />

    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

24.1.2 创建RemoteControlFragment

在包浏览器中，重命名activity_remote_control.xml为fragment_remote_control.xml。简单起见，先创建一个只有三个按钮的remote control应用。以代码清单24-3的内容替换fragment_remote_control.xml布局的默认内容。

代码清单24-3 只有三个按钮的初始布局（layout/fragment_remote_control.xml）

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
```

```
    tools:context=".RemoteControlActivity" >

    <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_centerHorizontal="true"
      android:layout_centerVertical="true"
      android:text="@string/hello_world" />

</RelativeLayout>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/fragment_remote_control_tableLayout"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:stretchColumns="*" >
  <TextView
    android:id="@+id/fragment_remote_control_selectedTextView"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="2"
    android:gravity="center"
    android:text="0"
    android:textSize="50dp" />
  <TextView
    android:id="@+id/fragment_remote_control_workingTextView"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"
    android:layout_margin="15dp"
    android:background="#555555"
    android:gravity="center"
    android:text="0"
    android:textColor="#cccccc"
    android:textSize="20dp" />
  <TableRow android:layout_weight="1" >
    <Button
      android:id="@+id/fragment_remote_control_zeroButton"
      android:layout_width="0dp"
      android:layout_height="match_parent"
      android:text="0" />
    <Button
      android:id="@+id/fragment_remote_control_oneButton"
      android:layout_width="0dp"
      android:layout_height="match_parent"
      android:text="1" />
    <Button
      android:id="@+id/fragment_remote_control_enterButton"
      android:layout_width="0dp"
      android:layout_height="match_parent"
      android:text="Enter" />
  </TableRow>
</TableLayout>
```

注意布局中的android:stretchColumns="*"属性定义,该属性可确保表格布局的列都具有同样的宽度。另外，文字尺寸大小使用dp单位而非sp单位。这意味着无论用户如何设置，设备屏幕上的文字大小都是一样的。

最后，新建RemoteControlFragment类，并设置其超类为android.support.v4.app.

Fragment支持库类。在新建的RemoteControlFragment.java中，覆盖onCreateView(...)方法，配置并启用按钮，如代码清单24-4所示。

代码清单24-4 RemoteControlFragment的编码实现（RemoteControlFragment.java）

```
public class RemoteControlFragment extends Fragment {
    private TextView mSelectedTextView;
    private TextView mWorkingTextView;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_remote_control, parent, false);

        mSelectedTextView = (TextView)v
                .findViewById(R.id.fragment_remote_control_selectedTextView);
        mWorkingTextView = (TextView)v
                .findViewById(R.id.fragment_remote_control_workingTextView);

        View.OnClickListener numberButtonListener = new View.OnClickListener() {
            public void onClick(View v) {
                TextView textView = (TextView)v;
                String working = mWorkingTextView.getText().toString();
                String text = textView.getText().toString();
                if (working.equals("0")) {
                    mWorkingTextView.setText(text);
                } else {
                    mWorkingTextView.setText(working + text);
                }
            }
        };

        Button zeroButton = (Button)v
            .findViewById(R.id.fragment_remote_control_zeroButton);
        zeroButton.setOnClickListener(numberButtonListener);

        Button oneButton = (Button)v
            .findViewById(R.id.fragment_remote_control_oneButton);
        oneButton.setOnClickListener(numberButtonListener);

        Button enterButton = (Button) v
            .findViewById(R.id.fragment_remote_control_enterButton);
        enterButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                CharSequence working = mWorkingTextView.getText();
                if (working.length() > 0)
                    mSelectedTextView.setText(working);
                mWorkingTextView.setText("0");
            }
        });

        return v;
    }
}
```

尽管现在有三个按钮，但我们只需两个点击事件监听器。这是因为其中两个数字按钮可共用一个事件监听器。点击某个数字按钮，要么添加另一个数字到数字输入区域，要么以点击的数字

替换输入区域原有数字，这取决于0是否是当前已输入数字。最后，在点击Enter按钮时，首先将输入区域的数字更新显示在已选频道区域，然后将输入区域清零。

运行RemoteControl应用。应该能看到一个带有0和1数字按钮的遥控应用，如图24-3所示。假设两个数字按钮就能搞定电视频道，谁还会需要更多呢？

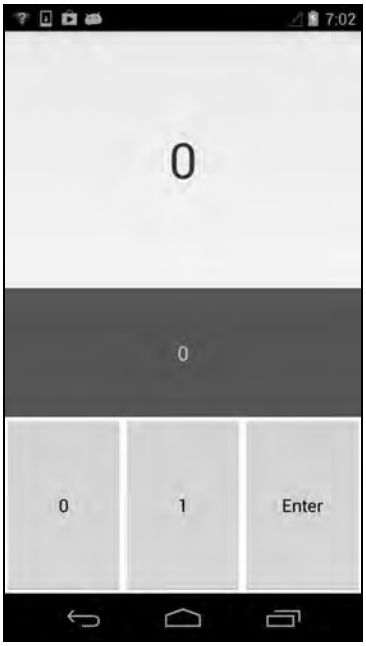

图24-3　如果电视发明者费罗・T. 法恩斯沃斯认为两个数字按钮就够了，这将意味着什么呢

24.2　使用样式消除重复代码

既然遥控应用已基本可用，我们再来看看应用的XML布局文件。看出来了吗？各按钮完全一样，现在这不是什么问题。但如需添加某个属性给按钮，我们就得重复添加三次。想象一下，有十二个或更多按钮呢？

令人欣慰的是，Android提供了各种UI样式，可用于避免重复性劳动。样式资源类似于CSS样式。每个样式定义了一套XML属性和值的组合。样式也可以具有层级结构：子样式拥有与父样式同样的属性及属性值，但也可覆盖它们或添加另外的属性值。

类似字符串资源，样式定义在XML文件的<resources>标签内，并存放在res/values目录中。另外还有一点相同的是，资源文件取什么名并不重要，但根据约定，样式通常定义在styles.xml文件中。

实际上，Android项目向导已创建了默认的styles.xml文件。（注意，该文件为RemoteControl应用定义了一套Android自带的陈旧主题，该主题配置决定了按钮、应用背景以及其他常见组件

的外观显示。）

现在需要从各按钮的属性定义中，提取出公共部分，并放入新的RemoteButton样式定义中。参照代码清单24-5，将新的样式添加到styles.xml样式文件中。

代码清单24-5 初始的RemoteControl样式（values/styles.xml）

```
<resources>

  <style name="AppTheme" parent="android:Theme.Light" />

  <style name="RemoteButton">
    <item name="android:layout_width">0dp</item>
    <item name="android:layout_height">match_parent</item>
  </style>

</resources>
```

每种样式都以<style>元素节点和一个或多个<item>元素节点进行定义。每个样式item都是以XML属性进行命名的，元素内的文字即为属性值。

我们现在只有一个名为RemoteButton的样式，该样式比较简单。不要着急，稍后，我们会逐步完善它。

要使用某个样式，首先要为布局中的视图添加样式属性，然后设置样式属性值以引用样式，如代码清单24-6所示。修改fragment_remote_control.xml布局文件，删除旧的属性定义，转而使用新的样式。

代码清单24-6 在布局中使用样式（layout/fragment_remote_control.xml）

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
  ...
  >

  ...

  <TableRow android:layout_weight="1" >
    <Button
      android:id="@+id/fragment_remote_control_zeroButton"
      android:layout_width="0dp"
      android:layout_height="match_parent"
      style="@style/RemoteButton"
      android:text="0" />
    <Button
      android:id="@+id/fragment_remote_control_oneButton"
      android:layout_width="0dp"
      android:layout_height="match_parent"
      style="@style/RemoteButton"
      android:text="1" />
    <Button
      android:id="@+id/fragment_remote_control_enterButton"
      android:layout_width="0dp"
      android:layout_height="match_parent"
      style="@style/RemoteButton"
      android:text="Enter" />
  </TableRow>
</TableLayout>
```

运行RemoteControl应用。可以看到，应用整体看起来和以前没什么不同。但是，布局定义看上去简洁多了，重复代码也更少了。

24.3　完善布局定义

有了样式文件，接下来的布局完善要省时省力多了。下面，我们来更新遥控应用的布局，实现一个漂亮且完整的用户界面（包含0～9数字键）。

首先，我们需要创建一个3×4的按钮阵列。因为阵列中的按钮几乎完全相同，所以无需逐个对按钮进行编码，我们可创建三个一排的按钮定义，然后使用四次。现在，创建一个名为res/layout/button_row.xml的文件，并在其中添加一排按钮的定义，如代码清单24-7所示。

代码清单24-7　三个一排的按钮定义（layout/button_row.xml）

```
<?xml version="1.0" encoding="utf-8"?>
<TableRow xmlns:android="http://schemas.android.com/apk/res/android" >
  <Button style="@style/RemoteButton" />
  <Button style="@style/RemoteButton" />
  <Button style="@style/RemoteButton" />
</TableRow>
```

然后，我们只需在布局中引入按钮组四次。如何引入？具体来说，就是使用include标签，如代码清单24-8所示。

代码清单24-8　在主布局定义中引入按钮组（layout/fragment_remote_control.xml）

```
<TableRow android:layout_weight="1" >
  <Button
    android:id="@+id/fragment_remote_control_zeroButton"
    style="@style/RemoteButton"
    android:text="0" />
  <Button
    android:id="@+id/fragment_remote_control_oneButton"
    style="@style/RemoteButton"
    android:text="1" />
  <Button
    android:id="@+id/fragment_remote_control_enterButton"
    style="@style/RemoteButton"
    android:text="Enter" />
</TableRow>
<include
  android:layout_weight="1"
  layout="@layout/button_row" />
<include
  android:layout_weight="1"
  layout="@layout/button_row" />
<include
  android:layout_weight="1"
  layout="@layout/button_row" />
<include
  android:layout_weight="1"
  layout="@layout/button_row" />
```

注意，定义在layout/button_row.xml中的三个按钮是没有资源id的。因为需引入按钮组四次，

如果为按钮设置id的话，则无法保证按钮的唯一性。同样地，按钮定义也没包含任何字符串资源。没关系，我们可以在代码中处理。现在，重新引用按钮并设置其上要显示的文字，如代码清单24-9所示。

代码清单24-9 重新引用数字键按钮（RemoteControlFragment.java）

```
View.OnClickListener numberButtonListener = new View.OnClickListener() {
  ...
};

Button zeroButton = (Button)v.findViewById(R.id.fragment_remote_control_zeroButton);
zeroButton.setOnClickListener(numberButtonListener);

Button oneButton = (Button)v.findViewById(R.id.fragment_remote_control_oneButton);
oneButton.setOnClickListener(numberButtonListener);
TableLayout tableLayout = (TableLayout)v
        .findViewById(R.id.fragment_remote_control_tableLayout);
int number = 1;
for (int i = 2; i < tableLayout.getChildCount() - 1; i++) {
    TableRow row = (TableRow)tableLayout.getChildAt(i);
    for (int j = 0; j < row.getChildCount(); j++) {
        Button button = (Button)row.getChildAt(j);
        button.setText("" + number);
        button.setOnClickListener(numberButtonListener);
        number++;
    }
}
```

代码中，for循环以TableLayout的子元素索引下标2为起点，跳过2个文本视图，然后遍历第一排的每个按钮。为各按钮统一设置前面创建的numberButtonListener监听器方法，并以字符形式对应设置按钮上要显示的数字。

以同样的方式完成其余三排按钮的设置。不过，最后一排按钮的处理有点棘手：Delete和Enter按钮需要特殊的处理，如代码清单24-10所示。

代码清单24-10 最后一排按钮的特殊处理（RemoteControlFragment.java）

```
for (int i = 2; i < tableLayout.getChildCount() - 1; i++) {
    ...
}

TableRow bottomRow = (TableRow)tableLayout
        .getChildAt(tableLayout.getChildCount() - 1);

Button deleteButton = (Button)bottomRow.getChildAt(0);
deleteButton.setText("Delete");
deleteButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        mWorkingTextView.setText("0");
    }
});

Button zeroButton = (Button)bottomRow.getChildAt(1);
zeroButton.setText("0");
zeroButton.setOnClickListener(numberButtonListener);
```

```
Button enterButton = (Button)
        v.findViewById(R.id.fragment_remote_control_enterButton);
Button enterButton = (Button)bottomRow.getChildAt(2);
enterButton.setText("Enter");
enterButton.setOnClickListener(new View.OnClickListener() {
  ...
});
```

至此，遥控应用就完成了。运行应用并使用它，虽然能够运行，但应用却无法遥控实际设备。作为练习，读者可尝试将应用与真实电视进行无线连接。

图24-4　对应真实设备的电视频道值

为按钮添加样式，我们可批量改变它们的显示外观。将代码清单24-11中加粗的三行内容添加到RemoteButton样式定义中。

代码清单24-11　微调样式（values/styles.xml）

```
<style name="RemoteButton">
  <item name="android:layout_width">0dp</item>
  <item name="android:layout_height">match_parent</item>
  <item name="android:textColor">#556699</item>
  <item name="android:textSize">20dp</item>
  <item name="android:layout_margin">3dp</item>
</style>
```

再次运行应用，查看样式微调后的效果，如图24-5所示。

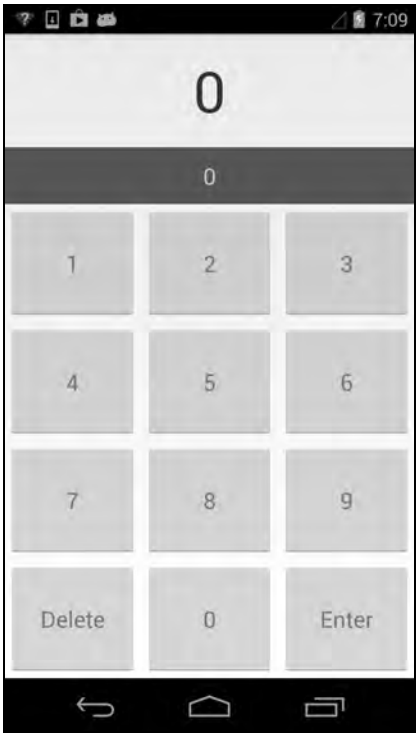

图24-5 整齐一致的按钮

24.4 深入学习：使用 include 与 merge 标签

在本章的学习中，使用include标签，我们在同一布局中多次引入了一排RemoteButton。可以看到，资源引入的实现方式简单直白：

```
<include layout="@layout/some_partial_layout"/>
```

以上代码表明，include将引入资源ID为@layout/some_partial_layout的布局文件内容。

如本章前几节看到的那样，使用include标签即可减少单个布局的重复代码，也可减少多个布局的重复代码。如有一大段布局定义需要出现在两个或多个布局定义中，可将它们提取出来单独定义为一个公共布局，然后在需要的地方引入它。这样，如果需要更新其他布局引入的布局内容，只需在一处更新即可。

有关include标签，还应掌握另外两个知识点。首先，基于当前设备配置，引入的布局同样会经历筛选过程。因此，如同其他布局的使用一样，引入布局也可以使用配置修饰符。其次，通过在include标签上指定android:id以及任何android:layout_*属性，可以覆盖引入布局根元素的对应属性。这样，可实现在多处引入同一布局，而在每次引入时使用不同的属性。

merge标签可与include标签一起协同工作。代替实际组件，merge可用作引入布局的根元素。

布局引入另一个以merge作为根元素的布局时，merge的子元素也会直接被引入。结果它们成了include元素父元素的子元素，而merge标签则会被丢弃。

merge的实际使用没有听起来那么难。如果还是搞不太清楚，只需记住一点：merge标签天生是要丢弃的。它只是用来满足XML文件的格式规范要求的：XML布局文件必须具有一个根元素。

24.5　挑战练习：样式的继承

对称分布在屏幕底部的Delete和Enter按钮以及数字按钮都使用了同样的样式。为同其他按钮区分开来，此类“操作”按钮需使用某种粗体样式。

本章的挑战练习是：让操作按钮看起来更醒目特别一些。完成这个练习，只需新建一个继承RemoteButton样式的按钮样式，并设置一个粗体文字样式属性，然后将新样式配置给底排的第一及第三个按键。

创建继承其他样式的新样式非常简单，具体有两种方式可供选择。一种是设置样式的parent属性为要继承样式的名称。另外一种是将父样式名称加上“.”符号后，作为前缀直接附加给样式名称，如ParentStyleName.MyStyleName。显然，第二种方法要更简单些。开发时，可根据实际情况自行选择。