# 第 25 章 XML Drawable与9-Patches



上一章，我们使用高级布局技巧，快速搭建了TV遥控器的用户界面。由于使用的是一成不变的Android式界面风格，当前用户界面虽然看上去还行，但似乎给人一种灰蒙蒙且单调沉闷的感觉。本章，我们将使用两种新的用户界面设计工具，赋予其一种全新的独特视觉体验，如图25-1所示。


图25-1　改进后的用户界面

两种设计工具均属于drawable。Android把任何可绘制在屏幕上的图形图像都称为drawable。drawable可以是一种抽象的图形、一个继承`Drawable`类的子类，或者是一张位图图像。我们已用过的封装图片的`BitmapDrawable`（详见第20章）也是一种drawable。本章，我们将会接触到更

多的drawable：state list drawable、shape drawable、layer list drawable以及nine patch drawable。前三个drawable通常定义在XML布局文件中，因此我们统一将它们归属为XML drawable类别。

## 25.1　XML drawable

在学习使用XML drawable之前，先试着使用`android:background`属性更改按钮的背景颜色，如代码清单25-1所示。

**代码清单25-1**　尝试更改按钮的背景颜色（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>
  <item name="android:background">#ccd7ee</item>
</style>
```

再次运行应用，可看到以下修改后的用户界面，如图25-2所示。


图25-2　哪里出问题了

如图所示，按钮的三维视觉效果消失了。点击任何一个按钮，会发现按钮的状态切换也不起作用了。

只改变了一个属性，为什么会带来如此大的变化？这是因为，和View类不同，Button类没有被赋予默认样式。默认样式来自于所选主题，并会设置一个Drawable作为视图的背景。正是这种背景drawable负责着视图的三维显示效果以及状态的切换。学完本章，我们会知道如何自己定制可用的drawable。

现在开始我们的学习。首先是利用ShapeDrawable创建彩色图形。既然XML drawable与特定的像素密度无关，因此无需考虑特定像素密度的目录，只需将其放入默认的drawable目录即可。

在包浏览器中，创建一个默认的res/drawable目录。然后在该目录下，以shape为根元素创建一个名为button_shape_normal.xml的文件，如代码清单25-2所示。（为何XML文件名包含“normal”字样？这是因为我们马上还会创建一个非normal的XML文件。）

**代码清单25-2 创建一个shape drawable按钮（drawable/button_shape_normal.xml）**

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
  android:shape="rectangle" >

  <corners android:radius="3dp" />

  <gradient
    android:angle="90"
    android:endColor="#cccccc"
    android:startColor="#acacac" />

</shape>
```

该XML文件定义了一个圆角矩形。corner元素指定了圆角矩形的圆角半径，而gradient元素则指定了色彩渐变的方向以及起始颜色。

也可使用shape创建其他各种图形，如椭圆、线条以及环等，并设置不同的视觉风格。请访问开发者文档网页<http://developer.android.com/guide/topics/resources/drawable-resource.html>，查看更多shape的相关信息。

在styles.xml中，更新按钮的样式定义，使用新建的Drawable作为按钮的背景，如代码清单25-3所示。

**代码清单25-3 更新按钮样式（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>
  <item name="android:background">#ccd7ee</item>
  <item name="android:background">@drawable/button_shape_normal</item>
</style>
```

运行RemoteControl应用，比较用户界面升级前后的异同，如图25-3所示。


图25-3　圆角按钮

## 25.2　state list drawable

虽然按钮的显示效果看上去还不错，但这些按钮仍然是静态的。实际上，更新前，Button的背景默认使用了state list drawable。使用state list drawable，可根据关联View的不同状态显示不同的drawable。（第18章中，我们曾用state list drawable切换过列表项的背景。）虽然状态分很多种，但这里只需关心按钮点击前后的状态即可。

首先创建点击状态下的按钮背景。除色彩差别外，其应与正常状态下的按钮背景完全相同。

在包浏览器中，复制一份button_shape_normal.xml文件，并命名为button_shape_pressed.xml。然后打开该XML文件，将其中定义的角度属性值增加180度，以改变渐变方向，如代码清单25-4所示。

**代码清单25-4　创建按钮点击状态下应显示的shape（drawable/button_shape_pressed.xml）**

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
  android:shape="rectangle" >

  <corners android:radius="3dp" />

  <gradient
```

```
    android:angle="90"
    android:angle="270"
    android:endColor="#cccccc"
    android:startColor="#acacac" />

</shape>
```

接下来，我们需要一个state list drawable。state list drawable必须包含一个selector根元素，以及用来描述状态的一个或多个item。右键单击res/drawable/目录，以selector为根元素，创建一个名为button_shape.xml的文件，如代码清单25-5所示。

**代码清单25-5 创建交互式的按钮shape（drawable/button_shape.xml）**

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:drawable="@drawable/button_shape_normal"
    android:state_pressed="false"/>
  <item android:drawable="@drawable/button_shape_pressed"
    android:state_pressed="true"/>
</selector>
```

未点击状态下，按钮显示的是深色文字，其与浅色系背景搭配比较合适。由于背景比较灰暗，因此在未点击状态下，按钮浅色系的背景看上去比较合适。而在点击状态下，使用深色系背景比较合适。与state list shape的创建类似，我们也可以轻松创建并使用state list color。

右键单击res/drawable/目录，创建另一个名为button_text_color.xml的state list drawable，如代码清单25-6所示。

**代码清单25-6 状态敏感的按钮文字颜色（drawable/button_text_color.xml）**

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="false" android:color="#ffffff"/>
    <item android:state_pressed="true" android:color="#556699"/>
</selector>
```

现在，在styles.xml中，调整按钮样式使用新建背景drawable和新建文字颜色，如代码清单25-7所示。

**代码清单25-7 更新按钮样式（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>
  <item name="android:background">@drawable/button_shape_normal</item>
  <item name="android:background">@drawable/button_shape</item>
  <item name="android:textColor">@drawable/button_text_color</item>
</style>
```

运行RemoteControl应用。查看点击状态下的按钮背景，如图25-4所示。


图25-4　点击状态下的按钮

## 25.3　layer list 与 inset drawable

应用初始版本采用的Android老旧样式按钮具有阴影显示效果。不幸的是，shape drawable没有可用的阴影属性。但使用其他两种类型的XML drawable，可自创阴影效果。这两种XML drawable类型分别是：layer list drawable和inset drawable。

下面我们来介绍创建阴影效果的方法。首先，使用与当前按钮drawable同样的shape创建一个阴影。然后，使用`layer-list`将阴影shape与当前按钮组合起来，再使用`inset`对按钮底边进行适当的短距移位，直到能够看到阴影显示。

在res/drawable/目录下，以`layer-list`为根元素，创建一个名为button_shape_shadowed.xml文件，如代码清单25-8所示。

**代码清单25-8　默认状态下的按钮阴影（drawable/button_shape_shadowed.xml）**

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android" >
  <item>
    <shape android:shape="rectangle" >
      <corners android:radius="5dp" />

      <gradient
        android:angle="90"
        android:centerColor="#303339"
```

```
        android:centerY="0.05"
        android:endColor="#000000"
        android:startColor="#00000000" />
    </shape>
  </item>
  <item>
    <inset
      android:drawable="@drawable/button_shape"
      android:insetBottom="5dp" />
  </item>
</layer-list>
```

可以看到，layer-list元素包含了多个Drawable，并以从后至前的绘制顺序进行排序。列表中第二个drawable是一个inset drawable，其任务就是在已创建的drawable底部做5dp单位的移位，并刚好落在移位形成的阴影上。

注意，阴影drawable并未使用单独的文件，而是直接被嵌入了layer list中。该技巧同样适用于其他drawable，如前面讲到的state list drawable。可自行决定究竟是嵌套drawable还是将其放入单独的文件使用。以单独文件的形式使用drawable可减少重复代码，简化各相关文件，但这也同时会使drawable/目录充斥着大量文件。不过要记住，应总是以简单且易于理解的方式编写代码。

在styles.xml中，修改按钮的样式定义，指向可显示阴影的新建drawable，如代码清单25-9所示。

**代码清单25-9 按钮样式的最终修改（values/styles.xml）**

```
<style name="RemoteButton">
  <item name="android:layout_width">0dp</item>
  <item name="android:layout_height">match_parent</item>
  <item name="android:textSize">20dp</item>
  <item name="android:layout_margin">3dp</item>
  <item name="android:background">@drawable/button_shape</item>
  <item name="android:background">@drawable/button_shape_shadowed</item>
  <item name="android:textColor">@drawable/button_text_color</item>
</style>
```

最后要做的是，为整个主视图创建一个色彩渐变的drawable，以获得细微的光影效果。以shape为根元素，新建一个名为remote_background.xml的drawable文件。然后添加代码清单25-10所示代码。（注意，如未指定shape，系统会默认使用矩形。）

**代码清单25-10 用于根视图的新背景drawable（drawable/remote_background.xml）**

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android" >
  <gradient
    android:centerY="0.05"
    android:endColor="#dbdbdb"
    android:gradientRadius="500"
    android:startColor="#f4f4e9"
    android:type="radial" />
</shape>
```

编辑fragment_remote_control.xml文件，引入新建的背景drawable，如代码清单25-11所示。

**代码清单25-11　将背景drawable应用于布局（layout/fragment_remote_control.xml）**

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/fragment_remote_control_tableLayout"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:background="@drawable/remote_background"
  android:stretchColumns="*" >
```

可以看到，在整个用户界面的调整过程中，无需对fragment布局文件做出调整。（最后一次代码修改除外。当然，也可为TableLayout创建新样式。）这确实给应用开发带来了方便。添加新按钮或重新布置视图需要更新布局文件，但更改用户界面的显示效果，使用样式文件即可。

## 25.4　使用 9-patch 图像

如聘请了专业设计师进行UI设计，他们完成的设计成果往往无法直接通过XML drawable使用。然而，我们仍然可能需要在多个地方复用可渲染资源。对于诸如按钮背景这样的可拉伸UI元素，Android提供了一种叫做9-Patch的图形处理工具。

接下来，我们对应用界面顶部的两个TextView进行修复。保持按钮布局不变，但使用可拉伸drawable作为背景图。布局与背景图比TextView更窄，这样可以节约一定的占用空间。第一张window.png为可拉伸背景图，可为TextView提供雅致的渐变色，并赋予边缘一定的边界空间，如图25-5所示。

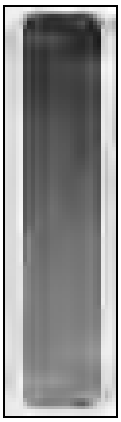

图25-5　用于频道显示窗口的背景图

另一张bar.png背景图提供了底部轻微阴影以及顶部细边。由于原图太小，这里显示的是它的放大版本，如图25-6所示。

在随书代码文件的25_XMLDrawables/RemoteControl/res/drawable-hdpi目录下，找到这些图像。然后，将其复制到RemoteControl应用的/res/drawable-hdpi目录中。

修改fragment_remote_control.xml文件，引入这些图像作为对应视图的背景图。同时调整文字

显示颜色以匹配新的背景图。将顶部视图的文字颜色调整为白色，将底部视图的文字颜色调整为默认的黑色。最后，为使界面看上去更整洁，设置底部TextView上的显示文字为斜体样式，如代码清单25-12所示。

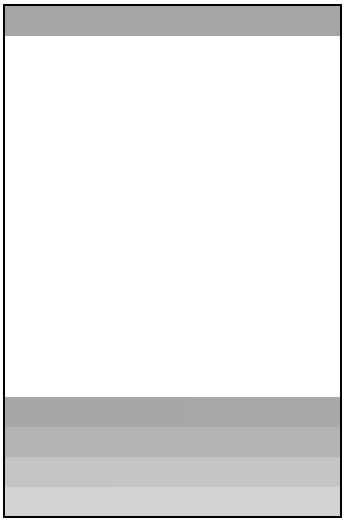

图25-6 用于频道输入区的背景图

**代码清单25-12 将drawable添加给样式（layout/fragment_remote_control.xml）**

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
  ... >
  <TextView
    android:id="@+id/fragment_remote_control_selectedTextView"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="2"
    android:background="@drawable/window"
    android:gravity="center"
    android:text="0"
    android:textColor="#ffffff"
    android:textSize="50dp" />
  <TextView
    android:id="@+id/fragment_remote_control_workingTextView"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_margin="15dp"
    android:layout_weight="1"
    android:background="#555555"
    android:background="@drawable/bar"
    android:gravity="center"
    android:text="0"
    android:textColor="#cccccc"
    android:textStyle="italic"
    android:textSize="20dp" />
  ...
</TableLayout>
```

运行应用，查看用户界面显示效果，如图25-7所示。

可以看到，各图像经均匀的四面拉伸，铺满了整个视图。有时可能会需要这样的效果，但这里并无必要。显然，模糊不清的图像以及底部`TextView`无法居中显示数字并不是我们想要的效果。

使用9-patch图像可解决该问题。9-patch图像是一种特殊格式的文件，因此Android知道图像的哪些部分可以拉伸缩放，哪些部分不可以。经适当处理后，可保证背景图的边角与工具创建的图像保持一致性。


图25-7　破碎的梦

为什么要叫做9-patch呢？9-patch可将图像分成3×3的网格，即由9部分或9 patch组成的网格。网格角落的patch不会被缩放，边缘部分的4个patch只按一个维度缩放，而中间部分则同时按两个维度缩放，如图25-8所示。

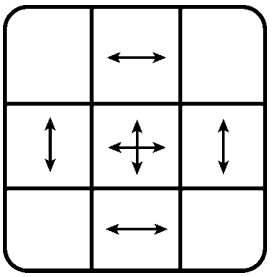

图25-8　9-patch的作用

9-patch图像和普通的png图像基本相同，但以下两点除外：9-patch图像文件名是以.9.png结尾的，图像边缘具有一个像素宽度的边框，用以指定9-patch图像的中间位置。边框像素绘制为黑线，以表明中间位置，边缘部分则用透明色表示。

可使用任何图形编辑工具来创建一张9-patch图像，但使用Android SDK中自带的draw9patch工具要更方便些。该工具位于SDK安装目录下的tools目录内。工具运行后，可直接拖曳文件到编辑区或从File菜单打开待编辑文件。

在编辑区打开文件后，在图像顶部填充黑色像素，并在左边框标记图像的可伸缩区域。参照图25-9所示，为window.png添加像素。

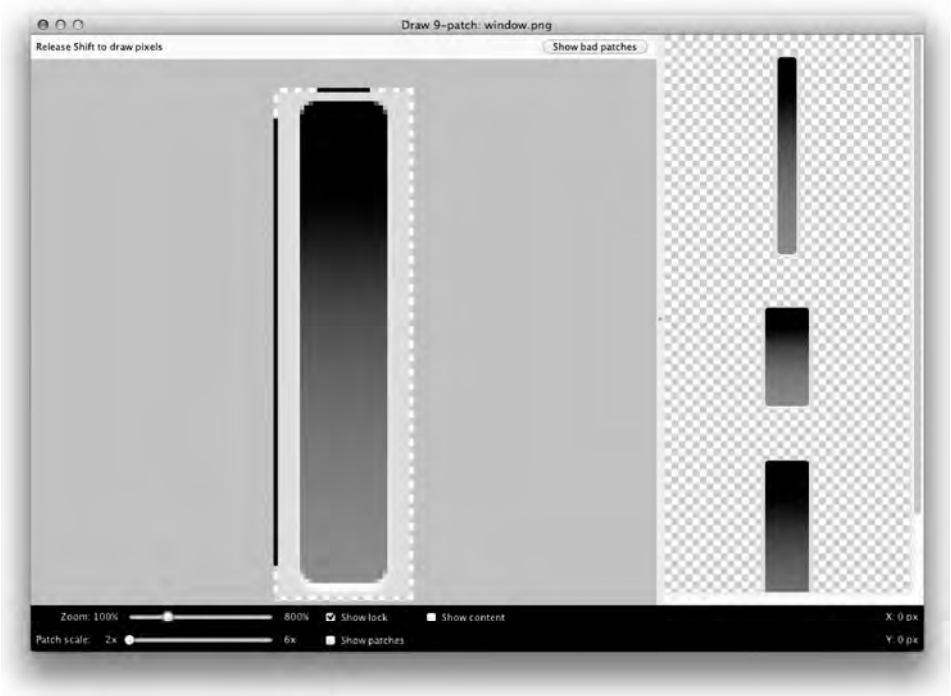


图25-9 频道显示窗口的 9-patch

顶部以及左边框标记了图像的可伸缩区域。那么底部以及右边框又要如何处理呢？它们定义了用于9-patch图像的可选drawable区域。drawable区域是内容（通常是文字）绘制的地方。如不引用drawable区域，则默认与可拉伸区域保持一致。这就是所需的效果，因此，可不引用drawable区域。

完成后，将结果保存到window_patch.9.png中。注意，不要让9-patch图像与其他图像发生冲突。例如，如同时存在名为window.9.png和window.png的两张图像，应用编译会失败。

紧接着，为bar.png图像编辑一个9-patch。使用drawable区域以水平垂直方向居中显示文字，并在边缘提供4个像素宽度的边框，如图25-10所示。

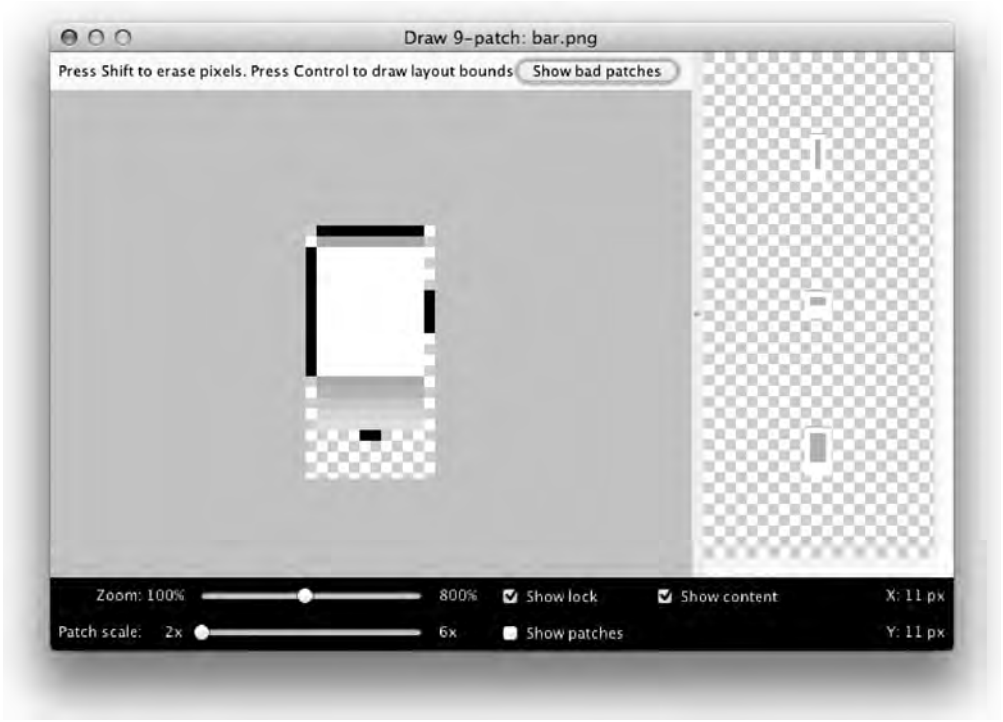


图25-10　频道输入区域的9-patch

完成后，将文件保存为bar_patch.9.png。

然后，右键单击res/目录，并点击Refresh按钮进行刷新。Eclipse自顾自的运行着，Refresh按钮可提醒其再次查看文件系统。

最后，使用9-patch drawable调整两个`TextView`，而非使用普通的老旧图像，如代码清单25-13所示。

**代码清单25-13　改用9-patch（layout/fragment_remote_control.xml）**

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
  ... >
  <TextView
    android:id="@+id/fragment_remote_control_selectedTextView"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="2"
    ~~android:background="@drawable/window"~~
    android:background="@drawable/window_patch"
    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_margin="15dp"
        android:layout_weight="1"
        android:background="@drawable/bar"
        android:background="@drawable/bar_patch"
        android:gravity="center"
        android:text="0"
        android:textColor="#cccccc"
        android:textSize="20dp" />
    ...
</TableLayout>
```

运行RemoteControl应用。如图25-11所示，背景图很漂亮！还记得应用最初的简陋界面吗？有了9-patch，设计用户界面可以说是省时又省力。而且，纵观各种流行应用，有着精美UI的应用更容易吸引用户。




图25-11 优化后的RemoteControl应用界面