

第 1 章

样式和主题

在进行程序的界面设计时，界面及界面上的组件经常需要进行显示外观的设置。例如，界面的背景颜色、字体的大小、字体的颜色、组件显示的大小、组件的填充效果、标题栏显示与否等。当然，你也可以为每个组件设置自己的显示属性。但是，为了便于对外观进行统一管理，我们需要将这些外观设置集中起来。Android 是通过样式，即 style，来完成这个工作的。在 Android 中使用样式来定制外观，需要做两个方面的工作，一是定义样式；二是应用样式。先从一个简单的例子谈起。

1.1 样式入门

我们先通过一个简单的例子来看看 Android 是如何定义样式，以及如何使用定义好的样式的。新建一个名称为 Ex01StyleTheme01 的 Android 工程。先将 res/layout/activity_main.xml 文件内容修改为如下代码：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"

    <Button
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:text="@string/text_btn_01" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:text="@string/text_btn_02" />

    <TextView
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="center"
        android:text="@string/text_textview" />
```

```
</LinearLayout>
```

这个布局很简单：一个 `LinearLayout` 容器中包含两个按钮和一个文本框，并且这三个组件平分 `LinearLayout` 的显示空间。然后修改 `res/values/strings.xml` 文件，在其中定义几个字符串常量资源，其代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="App_name">Ex01StyleTheme01</string>

    <string name="text_btn_01">第一个按钮</string>
    <string name="text_btn_02">第二个按钮</string>
    <string name="text_textview">学好 Android 的样式和主题</string>

</resources>
```

所定义三个常量资源，其实就是在布局文件中引用到的三个字符串常量。EX01StyleTheme01 程序运行结果如图 1-1 所示。



图 1-1 Ex01StyleTheme01 程序运行结果

从图 1-1 可以看出，两个按钮和文本框都使用了 Android 自定义的默认样式，现在我们修改按钮及文本框的样式。在 `res/values` 工程目录下，新建一个名为 `mystyles.xml` 的文件，并将其内容修改为如下代码：

```
<resources>

    <style name="MyButtonStyle">
        <item name="android:textColor">#00FF00</item>
        <item name="android:background">#FF0000</item>
        <item name="android:textSize">16sp</item>
    </style>

    <style name="MyTextViewStyle">
        <item name="android:textColor">#0000FF</item>
        <item name="android:typeface">monospace</item>
    </style>

</resources>
```

在 `mystyles.xml` 文件中,我们定义了两个新的 `style` 定义:其中一个 `style` 的名为 `MyButtonStyle`,另一个 `style` 的名为 `MyTextViewStyle`。在 `MyButtonStyle` 中,我们定义了文本颜色、背景和文字大小;在 `MyTextViewStyle` 中,我们定义了字体颜色和字体类型。

现在将这两个已经定义好的 `style` 应用到界面的组件:将 `MyButtonStyle` 应用到界面的第一个 `Button` 组件,将 `MyTextViewStyle` 应用到界面的 `TextView` 组件。将 `res/layout/activity_main.xml` 文件内容修改为如下代码:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <Button
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        style="@style/MyButtonStyle" //将定义好的 style 应用到这个组件
        android:text="@string/text_btn_01" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:text="@string/text_btn_02" />

    <TextView
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="center"
        style="@style/MyTextViewStyle" //将定义好的 style 应用到这个组件
        android:text="@string/text_textview" />

</LinearLayout>
```

注意上述代码第一个按钮中的代码为:

```
style="@style/MyButtonStyle" //将定义好的 style 应用到这个组件
```

这句代码将定义好的 `MyButtonStyle` 应用到这个组件。同时, `TextView` 对应的代码为:

```
style="@style/MyTextViewStyle" //将定义好的 style 应用到这个组件
```

这句代码将定义好的 `MyTextViewStyle` 应用到这个 `TextView` 组件。添加样式后的 `Ex01StyleTheme01` 程序的运行效果如图 1-2 所示。

比较图 1-1 和图 1-2 可知,添加样式前后程序的运行效果不同。通过这个例子,我们对 `Android` 的样式的定义和使用已经有了初步了解,下面将详细介绍如何定义样式及如何使用样式。



图 1-2 添加样式后的 Ex01StyleTheme01 程序的运行效果

1.2 定义样式

1.2.1 定义样式的一般方法

为了定义样式，你需要在 `res/values` 工程目录下新建一个 XML 文件，当然，你也可以在现有的某个文件下，如 `styles.xml`，直接添加要定义的样式。定义样式的一般格式如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <style name="自定义样式名称" parent="父样式名称">
        <item name="样式属性名称">属性值</item>
        .....
    </style>

    <style name="自定义样式名称" parent="父样式名称">
        <item name="样式属性名称">属性值</item>
        .....
    </style>

    .....
</resources>
```

可以通过在 Java 程序中使用 `R.style` 定义样式名称来访问定义的样式，也可以通过在 XML 文件中使用 `@style/` 自定义样式名称来访问定义的样式。注意，在样式定义中 `parent="父样式名称"`，意味着样式定义是支持继承的，也就是我们常说的级联样式，同时，样式定义中的 `parent` 属性是可选的。我们看一个样式定义的例子：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <style name="GreenText" parent="@android:style/TextAppearance">
        <item name="android:textColor">#00FF00</item>
    </style>

</resources>
```

在上述样式定义中，我们定义了一个名称为 `GreenText` 的样式，它继承了 Android 平台已定义的名称为 `@android:style/TextAppearance` 的样式，同时将 `@android:style/TextAppearance` 中 `android:textColor` 属性的值修改为 `#00FF00`，也就是将文本字体修改为绿色。定义了名为 `GreenText` 的样式后，就可以通过继承来定义新的样式了，如定义名为 `GreenTextLarge` 的样式，其代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

  <style name="GreenText" parent="@android:style/TextAppearance">
    <item name="android:textColor">#00FF00</item>
  </style>

  <style name="GreenTextLarge" parent="@style/GreenText">
    <item name="android:textSize">32sp</item>
  </style>

</resources>
```

上述代码采用继承的方式定义了一个名为 `GreenTextLarge` 的样式。由于 `GreenText` 样式是自定义的样式，所以还可以使用如下方式来定义样式的继承：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

  <style name="GreenText" parent="@android:style/TextAppearance">
    <item name="android:textColor">#00FF00</item>
  </style>

  <style name="GreenTextLarge" parent="@style/GreenText">
    <item name="android:textSize">32sp</item>
  </style>

  <style name="GreenText.Small"> //针对自定义的父样式，可以采用这种方式来继承
    <item name="android:textSize">8sp</item>
  </style>

</resources>
```

上述代码定义了一个名为 `GreenText.Small` 的样式，注意该样式名称比较特殊，这个名称表示 `GreenText.Small` 是一个新样式，但是它的父样式是 `GreenText` 样式。

1.2.2 样式定义中的可用属性

在样式定义中可以使用的属性随着定义样式应用目标的不同而不同。例如，定义一个针对 `TextView` 组件的样式和一个针对 `ImageView` 的样式，可以使用的属性是不相同的。因此，在定义样式时应该针对该样式的应用目标，参考组件的可用 XML 属性来确定可用属性。一种特殊情况是，如果你将某个样式定义应用到某个组件，而在这个样式定义中包含应用到的组件不支持的属性，那么该组件会自动忽略这个不支持的属性，且不影响其他支持属性的作用。

对 Android 支持的完整属性列表感兴趣的读者可以参考 Android 帮助文档中的 `android.R.styleable` 类，这个类给出了针对每个 Android 组件的完整的样式定义可用属性。从

android.R.styleable 类中查看 ImageView 组件的可用属性如图 1-3 所示。

```
public static final int[] ImageView
```

Attributes that can be used with a ImageView.

Includes the following attributes:

Attribute	Description
android:adjustViewBounds	Set this to true if you want the ImageView to adjust its bounds to preserve the ratio of its drawable.
android:baseline	The offset of the baseline within this view.
android:baselineAlignBottom	If true, the image view will be baseline aligned with based on its bottom edge.
android:cropToPadding	If true, the image will be cropped to fit within its padding.
android:maxHeight	An optional argument to supply a maximum height for this view.
android:maxLength	An optional argument to supply a maximum width for this view.
android:scaleType	Controls how the image should be resized or moved to match the size of this Image
android:src	Sets a drawable as the content of this ImageView.

图 1-3 从 android.R.styleable 中查看 ImageView 组件的可用属性

同时，ImageView 是 View 的子类，因此，View 的可用属性也是 ImageView 的可用属性，在 android.R.styleable 类中，可以查看 View 组件的可用属性，如图 1-4 所示。

```
public static final int[] View
```

Attributes that can be used with View or any of its subclasses. Also see ViewGroup_Layout for attributes that processed by the view's parent.

Includes the following attributes:

Attribute	Description
android:accessibilityLiveRegion	Indicates to accessibility services whether the user should be notified when this view changes.
android:alpha	alpha property of the view, as a value between 0 (completely transparent) and 1 (completely opaque).
android:background	A drawable to use as the background.
android:backgroundTint	Tint to apply to the background.
android:backgroundTintMode	Blending mode used to apply the background tint.
android:clickable	Defines whether this view reacts to click events.
android:contentDescription	Defines text that briefly describes content of the view.

图 1-4 View 组件的可用属性

1.3 应用样式

完成样式定义后，就可以将定义好的样式应用到需要的地方了，即将样式应用到某个组件，或将样式应用到某个 Activity 或整个应用程序。

1.3.1 将样式应用到某个组件

将定义好的样式应用到某个组件是一件非常简单的工作：在组件的配置中，添加“style”XML 配置属性。例如，将上文定义的 GreenText.Small 样式应用到 TextView 的定义中，只需要加上 style 属性即可，其代码如下所示：

```
<TextView
    style="@style/GreenText.Small"
    .....
    android:text="@string/hello" />
```

你可以将样式应用到具体组件，也可以将样式应用到容器组件，注意，应用到容器组件的样式只对这个容器组件有效，对放置于这个容器中的子组件无效。

1.3.2 将样式应用到某个 Activity 或整个应用程序

本章的标题是“样式和主题”，可是到目前为止我们一直没有介绍什么是主题。那么，到底什么是主题呢？用一句话来说就是，当我们把样式应用到某个 Activity 或整个应用程序时，这个样式就成了主题。为了将样式应用到某个 Activity 或整个应用程序，需要在 `AndroidManifest.xml` 文件中针对该 Activity 或整个应用程序添加 `android:theme` 属性。我们可以先定义一个如下样式：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <color name="custom_theme_color">#b0b0ff</color>
  <style name="CustomTheme" parent="@style/MyTheme.Light">
    <item name="android:windowBackground">@color/custom_theme_color</item>
    <item name="android:colorBackground">@color/custom_theme_color</item>
  </style>
</resources>
```

然后，将这个样式应用某个 Activity，其代码如下：

```
<activity android:theme="@style/CustomTheme">
```

或将这个样式应用到整个应用程序，其代码如下：

```
<Application android:theme="@style/CustomTheme">
```

那么这个样式就是主题。

主题是一种特殊的样式，由于主题样式是应用于某个 Activity 或整个应用程序的，Android 为主题样式的定义引入了一些特殊的属性。例如，`android:windowNoTitle` 属性就只能用于对主题样式的定义，该属性表示在显示某个 Activity 或整个应用程序时不显示 Activity 的标题；`android:textSelectHandle` 表示在进行文本选择时显示的文本定位图片资源等。完整的可用于主题样式定义的属性可参考 Android 帮助文档 `android.R.styleable` 类定义中的 Theme 小节。部分可用于主体样式的属性如图 1-5 所示。

```
public static final int[] Theme
```

These are the standard attributes that make up a complete theme.

Includes the following attributes:

Attribute	Description
<code>android:absListViewStyle</code>	Default AbsListView style.
<code>android:actionBarDivider</code>	Custom divider drawable to use for elements in the action bar.
<code>android:actionBarItemBackground</code>	Custom item state list drawable background for action bar items.
<code>android:actionBarPopupTheme</code>	Reference to a theme that should be used to inflate popups shown by widgets in the action bar.
<code>android:actionBarSize</code>	Size of the Action Bar, including the contextual bar used to present Action Modes.
<code>android:actionBarSplitStyle</code>	Reference to a style for the split Action Bar.
<code>android:actionBarStyle</code>	Reference to a style for the Action Bar.
<code>android:actionBarTabBarStyle</code>	

图 1-5 部分可用于主题样式的属性

1.4 使用 Android 平台已定义的样式和主题

Android 平台定义了一系列样式和主题以供应用程序使用,在所有定义的样式中,以 Theme 开头的样式是主题样式,其他样式则是普通样式。Android 完整的样式定义可参考 android.R.styleable 类。要使用 Android 已定义的样式或主题,需要将样式或主题名中的下划线(_)替换为小数点(.)。例如,为了在程序的某个 Activity 中使用 Theme_NoTitleBar 主题样式,则需按如下方式使用:

```
<activity android:theme="@android:style/Theme.NoTitleBar">
```

1.4.1 Android 已定义的典型的样式

根据不同的 Android SDK 版本,Android 自定义了一系列的主题,以下是典型的 Android 不同版本的主题。

(1) API 1。

android:Theme 表示根主题。

android:Theme.Black 表示背景为黑色。

android:Theme.Light 表示背景为白色。

android:Theme.Wallpaper 表示以桌面墙纸为背景。

android:Theme.Translucent 表示透明背景。

android:Theme.Panel 表示平板风格。

android:Theme.Dialog 表示对话框风格。

(2) API 11。

android:Theme.Holo 表示 Holo 根主题。

android:Theme.Holo.Black 表示 Holo 黑色主题。

android:Theme.Holo.Light 表示 Holo 白色主题。

(3) API 14。

Theme.DeviceDefault 表示设备默认根主题。

Theme.DeviceDefault.Black 表示设备默认主题为黑色主题。

Theme.DeviceDefault.Light 表示设备默认主题为白色主题。

(4) API 21: 常说的 Android Material Design 就要用这种主题。

Theme.Material 表示 Material 根主题。

Theme.Material.Light 表示 Material 白主题。

(5) 兼容包 v7 中的主题。

Theme.AppCompat 表示兼容主题的根本主题。

Theme.AppCompat.Black 表示兼容主题的黑色主题。

Theme.AppCompat.Light 表示兼容主题的白色主题。

1.4.2 使用主题的注意事项

所有能应用于应用程序主题的名称都是以“Theme.”开头的,主题名称不是以“Theme.”开头的就不是用于应用程序的主题,而是用于某些局部控件的主题,如“ThemeOverlay”主题可用于 Toolbar,又如“TextAppearance”主题可用于设置文字外观,这里不做深入分析了。

很多主题在使用时会报错，其原因有很多，如窗体必须继承 `AppCompatActivity`、`ActionBarActivity` 或者 `FragmentActivity`，需要手动指定宽高，需要提升最低 API 版本，需要使用更高版本的 SDK，或者兼容包版本不对等。所以，在使用主题时要特别小心。

1.5 Android 应用程序的主题样式结构分析

介绍完样式与主题的相关知识，现在回到 Android 程序，看看 Android 程序的样式主题相关内容。

当你在 Android Studio 中新建一个 Android 工程时，Android 已经为你制定了默认的主题。`AndroidManifest.xml` 文件的内容如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.ttt.ex01styletheme01">

    <Application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/App_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" > > //指定该应用程序的主题
    <activity
        android:name=".MainActivity"
        android:label="@string/App_name"
        android:theme="@style/AppTheme.NoActionBar" > //指定该应用程序的主题
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

其中，如下所示语句：

```
android:theme="@style/AppTheme" > //指定该应用的主题
```

指定了该应用程序的主题为 `AppTheme`。在 `res/values` 目录下的 `styles.xml` 文件中，可以找到对主题样式 `AppTheme` 的定义。`res/values/styles.xml` 文件下的内容如下所示：

```
<resources>

    <!-- Base Application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

    <style name="AppTheme.NoActionBar">
```

```
<item name="windowActionBar">false</item>
<item name="windowNoTitle">true</item>
</style>

<style name="AppTheme.AppBarOverlay"
        parent="ThemeOverlay.AppCompat.Dark.ActionBar" />

<style name="AppTheme.PopupOverlay" parent="ThemeOverlay.AppCompat.Light" />

</resources>
```

1.6 本章同步练习

Android 平台预定义了很多主题样式, 请将 1.4.1 节中 Android 已定义的典型的样式应用到你的一个例子程序中, 并观察每个主题样式的外观。