Android 样式系统 | 主题背景属性

原创 Android 谷歌开发者 前天

来自专辑

Android 开发技术

在 Android 样式系统系列的前几篇文章中，我们介绍了主题背景与样式的区别，以及为什么说通过主题背景和公共主题背景属性来分解您要实现的内容是一个不错的主意，请点击链接回顾:

- Android 样式系统 | 主题背景和样式
- Android 样式系统 | 常见的主题背景属性

这会让我们通过创建更少的布局或样式，以隔离主题背景中的修改。在实际开发中，您通常希望根据主题背景改变颜色，因此您应该始终通过主题背景属性来引用颜色。

这意味着您可以将如下代码视为有代码异味 (Code smell):

```
<!-- Copyright 2019 Google LLC.
   SPDX-License-Identifier: Apache-2.0 -->
<View …
  android:background="@color/white"/>
```

相反，您应该使用主题背景属性，它允许您按主题更改颜色，例如，在深色主题中提供一个不同的值:

```
<!-- Copyright 2019 Google LLC.
   SPDX-License-Identifier: Apache-2.0 -->
<View …
  android:background="?attr/colorSurface"/>
```

- 深色主题
  https://developer.android.google.cn/guide/topics/ui/look-and-feel/darktheme

即使您当前不支持其他主题 (什么，您的应用还没有支持深色主题？)，我们依然建议您采用这种方法，因为这样会让新主题的采用变得更加简单。

合格的 Colors 文件

您可以通过在不同的配置中添加不同的值来改变颜色 (例如，在 res/values/colors.xml 中和在 res/values-night/colors.xml 中的备选值里均定义 @color/foo)，但我们依然建议您使用主题背景属性来替代它们。对颜色层级的区分，会迫使您给颜色赋予语义化名称，换句话说，您应该不会在给颜色命名为 @color/white 的同时，又为深色模式提供一个深色变体，这会让人感到非常困

惑。所以，您可能会想要使用一个语义化名称，例如 @color/background。这种方法带来的问题是它合并了颜色声明和具体的值，因此，它并没有指出颜色是可以或者能够随主题背景而变化的。

@colors 的变化也会鼓励您创造更多颜色。如果在不同的情境下要使用具有相同值的、新的语义化命名的颜色 (即，不是背景色但应该使用相同颜色)，这时候您仍需要在 colors 文件中创建新的条目。通过使用主题背景属性，我们可以将语义颜色的声明从提供它们的值中区分开来，而且让使用方更清楚地了解到颜色会随主题背景而变化 (因为它们使用 ?attr/ 语法)。将颜色声明保持为字面值，您就可以自定义应用使用的颜色调色板，并在主题背景级别修改它们，这会让 color.xml 较小且易维护。

这种方法的额外好处是，布局/样式引用这些颜色时复用性变得更高。由于主题背景可以被覆盖或者改变，因此这间接表示: 您不需要创建其他布局或样式就可以更改某些颜色——您可以在相同的布局中使用不同的主题背景。

始终使用?

在某些情况下，您或许不想按照主题背景更改颜色。例如，在 Material Design 规范文档中提到，您可能希望在浅色和深色主题中均使用同一类型的颜色。

- Material Design 规范文档
  https://material.io/design/color/dark-theme.html#ui-application

在这种特殊情况下，直接引用颜色资源是再合适不过的:

```
<!-- Copyright 2019 Google LLC.
   SPDX-License-Identifier: Apache-2.0 -->
<FloatingActionButton …
  app:backgroundTint="@color/owl_pink_500"/>
```

当前发展状况

当使用 ColorStateLists 时，您可能也不会在您的布局/样式中直接引用主题背景属性。

```
<!-- Copyright 2019 Google LLC.
   SPDX-License-Identifier: Apache-2.0 -->
<View …
  android:background="@color/primary_20"/>
```

- ColorStateLists
  https://developer.android.google.cn/reference/android/content/res/ColorStateList

如果 primary_20 是一个 ColorStateList，它本身引用主题背景属性来获取色值也可能是合理的(请参见下文)。ColorStateLists 通常为不同的状态 (按下，禁用等) 提供不同的颜色，但它还有另外一种可用于主题化功能您可在选取的颜色上指定透明度值:

```
<!-- Copyright 2019 Google LLC.
    SPDX-License-Identifier: Apache-2.0 -->
<selector …
  <item android:alpha="0.20" android:color="?attr/colorPrimary" />
</selector>
```

这种单项 ColorStateList (即只提供单个默认颜色，而非每种状态的不同颜色) 有助于减少您需要维护的颜色资源数量。它并没有定义一个新的颜色资源的方式来手动为您 (每一个配置文件) 的 primary 颜色设置 alpha 值，而是通过改变当前主题背景中的 colorPrimary 的方式。如果您的原始颜色发生了变化，则只需要在一个地方进行更新，无需调整所有已更新的地方。

虽然此技术很有用，但仍有一些注意事项:

1. 如果指定的颜色也具有 alpha 值，则 alpha 会被合并。例如，将 50% 的 alpha 应用于 50% 的不透明白色中，将产生 25% 的白色:

```
<!-- Copyright 2019 Google LLC.
    SPDX-License-Identifier: Apache-2.0 -->
<selector …
  <item android:alpha="0.50" android:color="#80ffffff" />
</selector>
```

因此，最好将主题背景颜色指定为完全不透明，然后使用 ColorStateLists 修改它们的 alpha。

2. 仅在 API 23 中添加了 alpha 组件，因此，如果您的最小 sdk 低于这个版本，请确保使用支持此行为的 AppCompatResources.getColorStateList (并始终使用 android:alpha 命名空间，而绝不使用 app:alpha 命名空间)。

- AppCompatResources.getColorStateList
  https://developer.android.google.cn/reference/androidx/appcompat/content/res/AppCompatResources.html#getColorStateList(android.content.Context,%20int)

3. 通常，我们使用简写法，将颜色设置为 Drawable，例如:

```
<!-- Copyright 2019 Google LLC.
    SPDX-License-Identifier: Apache-2.0 -->
<View …
  android:background="@color/foo"/>
```

View 的背景是一个 Drawable，此简写把给定的颜色强转成了一个 ColorDrawable。但是没有办法把 ColorStateList 转换成 Drawable (API 29 之前使用 ColorStateListDrawable 解决这个问题)。

- ColorDrawable
  https://developer.android.google.cn/reference/android/graphics/drawable/ColorDrawable
- ColorStateListDrawable
  https://developer.android.google.cn/reference/android/graphics/drawable/ColorStateListDrawable

但是，我们可以通过迂回的方式绕过此限制:

```
<!-- Copyright 2019 Google LLC.
   SPDX-License-Identifier: Apache-2.0 -->
<View …
  android:background="@drawable/a_solid_white_rectangle_shape_drawable"
  app:backgroundTint="@color/some_color_state_list"/>
```

请确保您的 backgroundTint 支持您的 View 所需的状态，例如，如果被禁用时需要更改。

强制执行

即使您已经说服自己使用主题背景属性和 ColorStateList，但如何在代码库或者团队中使用呢？您可以在 Code review 期间尝试保持警惕，但它的扩展性不是很好。更好的方法是依靠工具来解决此问题。

《Making Android Lint Theme Aware》这篇文章简述了如何通过添加 Lint 检查来寻找直接引用颜色的用法，并涵盖了文中提及到的所有建议。

- 《Making Android Lint Theme Aware》
  https://proandroiddev.com/making-android-lint-theme-aware-6285737b13bc

间接使用

使用主题背景属性和 ColorStateList 将颜色分解为主题背景的方法，可使您的布局和样式更加灵活，提高代码复用性并保持代码库的精简和易维护性。

我们将在后续文章中介绍更多主题背景的用法以及它们之间的相互影响，感兴趣的读者请继续关注。