


## 处理视觉冲突 | 手势导航 (二)



谷歌开发者   
已认证的官方帐号

已关注

2 人赞同了该文章



作者 / Chris Banes, Android 开发者关系团队工程师

我们将在近期为大家带来一个关于 "手势导航" 的系列连载，本文是连载的第二篇，如果您希望了解其他手势导航的话题，请持续关注我们。

在上一篇文章中，我们介绍了如何将应用构建到全面屏设备。然而有些交互可能导致应用的某些视图被系统栏遮盖，导致用户无法看见或操作。本文正是为帮助您解决这个问题而撰写——如何判断安全的交互区域。

更具体一点来说，本文主要处理与系统 UI 出现视觉重叠的问题。系统 UI 包括屏幕上由系统提供的所有 UI，例如导航栏和状态栏，另外它还包括诸如通知面板之类的内容。

### 边衬区 (Insets)

不少 Android 开发者看到边衬区 (insets) 往往会退避三舍，这个可能来自他们在 Android Lollipop 时代试图在状态栏后面绘制 UI 的经历，而这个经历并不那么令人愉悦。我们甚至能看到在 [StackOverflow](#) 上有个一直热门的问题就是关于这个的。

Insets 区域负责描述屏幕的哪些部分会与系统 UI 相交 (intersect)，例如导航或状态栏。如果您的控件出现在了这些区域内，就可能被系统 UI 遮盖。自然，我们可以使用 insets 区域来尝试解决视觉冲突，如把视图从屏幕边缘向内移动到一个合适的位置。

在 Android 上，Insets 区域由 WindowInsets 类表示，在 AndroidX 中则使用 WindowInsetsCompat。在 Android 10 系统中处理应用布局时，开发者需要知晓 5 个获取 insets 区域的方法。需要使用哪种方法取决于具体情况，接下来就让我们逐一说明。

### 系统窗口边衬区

方法: `getSystemWindowInsets()`

系统窗口区域是最常用到的。自 API 1 以来，它们就以各种形式存在着，并且每当系统 UI 重叠显示在您的应用上方时，这个方法就会被调用。常见的例子是下拉状态栏和导航栏，或者弹出屏幕软键盘 (IME)。

我们来看一个使用系统窗口区域的例子。我们有一个悬浮操作按钮 (FAB)，它位于屏幕右下角，距离屏幕边缘 16dp (这符合设计指南中的要求)。

Google I/O 的官方应用中就有这种 FAB，在应用被迭代为全屏应用前它看起来是这个样子：

在迭代为全面屏应用后，为了取得更加沉浸式的体验，我们将日程表控件延展进了导航栏的区域。但这时可以看到 FAB 被导航栏遮住了：

更糟的是，FAB 现在被遮盖了，就意味着用户可能无法点击它。显然我们要解决这种视觉冲突。当系统设置为使用按钮导航模式时（即上图例子所示），视觉冲突会更加明显，因为这时导航栏的高度更大。在系统使用手势导航模式时（即导航栏变成屏幕底部的一条粗线，也就是导航条），由于导航条有动态色彩调整功能，这个冲突可能不会那么明显。但是请记住，系统 UI 可以随时切换为半透明遮盖模式，所以我们有必要彻底解决这个问题。

**再强调一次，您现在最好在所有的导航模式下测试您的应用。**

那么我们如何处理这种视觉冲突呢？系统窗口区域在这就能派上用场。这套 insets 描述了系统栏占据的区域，方便您使用对应的数值将自己的控件从系统栏下面移开。

具体到本例中，FAB 位于底部右侧边缘附近，因此我们可以使用 `systemWindowInsets.bottom` 和 `systemWindowInsets.right` 值来增加 FAB 下方和右方的边距。

增加边距后看到的效果如下：



本文后面会为大家介绍具体做法。

简而言之，系统窗口区域 insets 最适合那些需要点击的控件，可以确保系统栏不遮盖住它们。

## 可点击区域

方法: `getTappableElementInsets()`

接下来是 Android 10 中新增的可点击区域 insets。它们与上面的系统窗口区域 insets 非常相似。可点击区域 insets 用来界定可触发系统点击行为 (tap) 的最小区域。注意，使用可点击区域里的数值进行布局时，依然可能导致自己的控件与系统 UI 在视觉上重叠，这一点与系统窗口区域 insets 不同，使用后者的值对自己的控件进行位移后能确保不会与系统/导航栏发生视觉重叠。

这里让我们仍然使用 FAB 来举例：

注意看上图，在导航栏模式下，FAB 不会进入导航栏占据的高度 (48dp)。在手势操作 (导航条) 模式，且开启了导航条色彩适应后，虽然导航条依然有高度 (即红色区域 16dp)，但它被认为是 "透明" 的，系统在这 16 dp 的高度内依然允许用户点击应用里的控件，所以在可点击区域 insets 中，其 bottom 值为 0dp。如上图所示，FAB 这时会更靠下一些。

不要在代码中硬编码上面提到的值 (48dp / 16 dp)，因为导航栏的尺寸是会变动的，请使用 insets 获取需要的数值。



Insets 其实并没有规定 "您应在何处放置自己的控件"，所以从理论上讲可以这么做：

但这个做法显然不好，因为 FAB 这时非常靠近导航条，虽然依然可以点击，但会让用户感觉迷惑。

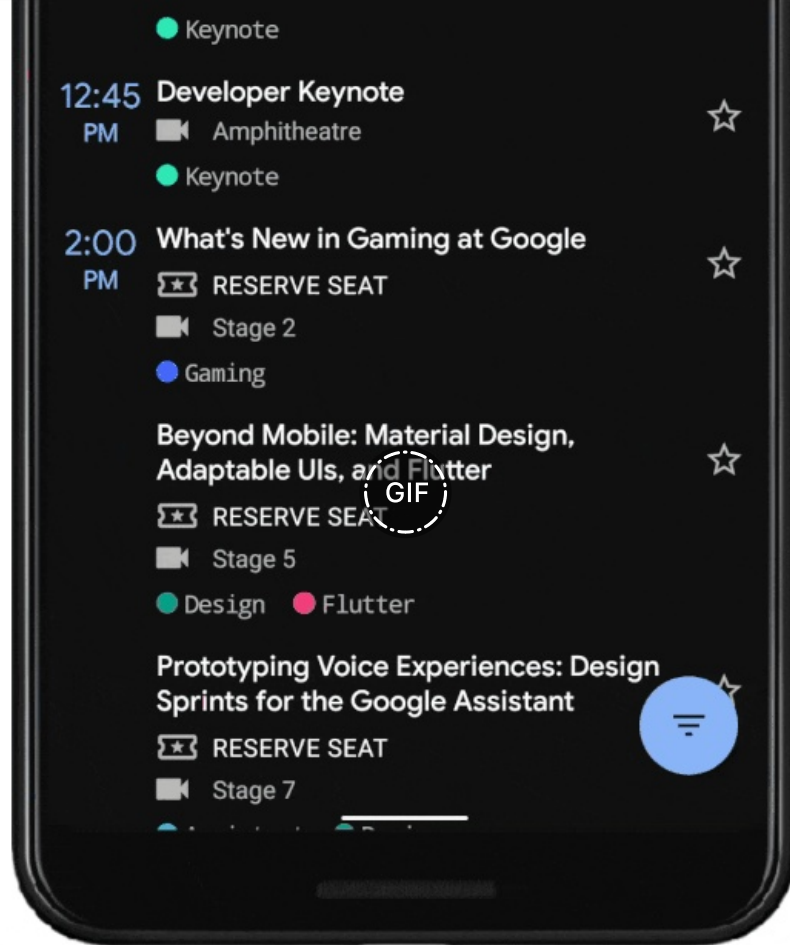
从实用的角度出发，在日常开发中我建议使用系统窗口区域 insets，它可以更好地满足几乎所有需要使用可点击区域 insets 的用例。

## 系统手势边衬区

方法: `getSystemGestureInsets()` & `getMandatorySystemGestureInsets()`

这是在 Android 10 中新增的: 系统手势区域边衬区 (insets)。Android 10 带来了新的手势导航模式，允许用户通过手势动作，而不是导航按钮来进行导航：

1. 从屏幕左/右边缘向中间滑动，相当于后退按钮 (Back)。
2. 从屏幕底部开始向上滑动，可以让用户切换最近使用的应用 (Recent)。



在系统手势区域中，系统手势操作优先于应用自己的手势操作。您可能已经注意到系统手势区域有两个获取方法。这是因为 `getMandatorySystemGestureInsets()` 只包含强制性系统手势区域，是系统手势区域的子集。这里我们分开来说：

### 系统手势边衬区

首先是系统手势边衬区。在这些区域内，系统手势优先于应用手势。在 Android 10 上，系统手势区域如下：

△ 左/右侧的后退操作区域宽 40dp，下方的主屏操作区域高 60dp

如果您有需要滑动操作的控件出现在了系统手势区域内，就可以使用对应的数值来将这些控件挪开。常见的例子包括底部导航菜单 ([Bottom Sheets](#))、游戏里的滑动交互、多图展示 ([ViewPager](#)) 等。

### 强制系统手势边衬区

强制系统手势边衬区是系统手势边衬区的子集，之所以称之为“强制区域”，是因为应用无法修改这些区域。关于如何修改系统手势区域，请参考我们接下来的文章《[如何处理手势冲突！手势导航连](#)



强制系统手势边衬区只包含那些系统保留的区域，在这些区域内系统手势操作永远优先。在 Android 10 上，当前唯一的强制区域是屏幕底部的主屏手势区域，系统保留这个区域就可以让用户在任何时候都可以退出当前应用：

△ 底部 60dp 即为强制系统手势边衬区

## 稳定显示边衬区

方法: `getStableInsets()`

这也是我们今天提到的 5 个 inset 方法的最后一个。严格来说，这个方法与手势导航关系不大，但是为了知识的完整性，我们这里快速介绍一下这个方法。

和系统窗口边衬区类似，稳定显示区域是系统 UI 可能在您的应用上显示的位置。在有些显示模式下 (比如放松模式和沉浸模式)，系统 UI 可能会根据情况在可见与不可见之间切换 (如游戏、照片浏览、视频播放器等)。这时使用稳定显示区域就可以确保自己的控件不会被 "突然出现" 的系统 UI 挡住。

## 处理边衬区冲突

希望您现在对不同类型的 insets 区域有了更深的了解，下面我们来看看您需要如何在应用中实际使用它们。

访问 `WindowInsets` 主要是通过 `setOnApplyWindowInsetsListener` 这个方法。我们来看一下例子，我们想给某个控件增加一些边距，让它不被导航栏遮挡：

```
ViewCompat.setOnApplyWindowInsetsListener(view) { v, insets ->
    v.updatePadding(bottom = insets.systemWindowInsets.bottom)
    // Return the insets so that they keep going down the view hierarchy
    insets
}
```

在这里，我们仅将系统窗口区域的底部边距值赋给了控件的底边距。

*注意: 如果您要在 `ViewGroup` 上执行此操作，则可能要对其进行设置 `android:clipToPadding="false"`。这是因为默认情况下，所有视图都会在填充区域内裁剪图形。该属性通常与 `RecyclerView` 一起使用，我们将在以后的文章中对其进行详细介绍。*

但是，请确保 Listener 里的计算操作有幂等性，即多次进行该计算所得到的结果应该相同。以下是一个错误的例子：

```
ViewCompat.setOnApplyWindowInsetsListener(view) { v, insets ->
    v.updatePadding(bottom = v.paddingBottom + insets.systemWindowInsets.bottor
```

请不要在计算边距时使用自加运算 (+=)。因为这个计算可能会重复多次，自加运算会导致结果不符合预期。



## 使用 Jetpack

使用 insets 时，我建议始终用 Jetpack 中的 `WindowInsetsCompat` 类，无论您需要的最低 SDK 版本是什么。多年来，`WindowInsets` API 已得到改进和扩展，而 `compat` 版本在所有的 API 级别上都提供了一致的 API 和行为。

在 Android 10 中新增的 insets 方面，`compat` 版本的方法在所有 API 级别的设备上都能得到正确的结果。要访问 AndroidX 中的新 API，请确保更新到 `androidx.core:core:1.2.0-xxx` (目前为 Alpha 版) 或更高版本。

## 更进一步

本文提到的是使用 `WindowInsetsCompat` API 的最简单方法，但它们可能会让您的代码非常冗长和重复。我在今年早些时候写了一篇博文，详细介绍了一些使用绑定转换操作显著提高效率的做法。

在本次连载的下一篇文章《如何处理手势冲突 | 手势导航连载 (三)》中，我们将为大家介绍如何处理应用与系统的手势导航冲突，敬请持续关注。

[点击这里](#) 进一步了解 Android 手势导航

发布于 2019-11-28

[Android](#)

[Android 开发](#)

### 推荐阅读



#### android binder资料总结

1 Android binder 是学习 Android 系统一定要啃得硬骨头，可能你刚开始的时候并不理解其中的精髓，但是在 android 系统的很多地方你都会遇到它。不过要我自己写明白

#### 微信抢红包插件与Android辅助功能

逢年过节大家都少不了发微信红包，通过微信红包来表达祝福。同时，微信还有拼手气群红包。各种群红包群 亲戚群 工作群逢年过



▲ 赞同 2 ▼

● 1 条评论

➦ 分享

♥ 喜欢

★ 收藏

📄 申请转载

...

1 条评论

切换为时间排序

写下你的评论...



aslant

2019-11-28

打开手势导航，将第三方应用分屏置于分屏下层，返回分屏上层应用，重新进入第三方应用的时候，屏幕中间必现黑色条纹。类似操作可以用Pixel的Q版本打开QQ和腾讯视频操复现。原因可能是DecorView的drawLegacyNavigationBarBackground方法没有针对手势导航特殊处理，你们优化下吧。



赞