


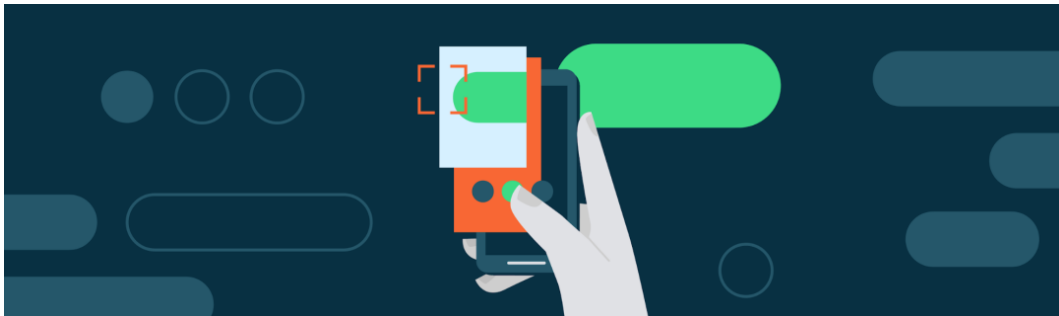
使用 PreviewView 来展示相机预览



谷歌开发者 
已认证的官方帐号

已关注

10 人赞同了该文章



显示相机预览内容是每个相机类应用都会包含的功能，想要完美实现这个却并非易事。原因是，在某些特别极端情况下 camera2 API 的使用会变得很复杂，而且在不同设备上的行为还会有所不同。还好，Jetpack CameraX 库的 PreviewView 可以帮助您解决这一问题。通过在各种 Android 设备上提供开发者友好、一致且稳定的 API，使得展示相机的预览变得不再困难。

PreviewView 的介绍

PreviewView 是一个可以显示相机画面的自定义 View，它被构建的初衷便是降低开发者们在设置和处理相机所使用的预览画面 (preview surface) 的难度。

如果您需要在应用中提供展示相机画面的基本功能，使用 PreviewView 是最推荐的做法，它有以下几个优点：

- **使用简单:** PreviewView 是一个 View，它通过管理 Preview 用例所使用的 Surface 来实现将相机捕捉到的画面展示在界面布局中的全部功能；
- **代码轻量:** PreviewView 只专注于实现相机画面预览功能。它所有内部资源都致力于对相机预览画面的展示，以及在相机使用过程中对预览画面 (preview surface) 进行管理。这样的关注点分离使得 PreviewView 的代码能够保持简洁；
- **支持全面:** PreviewView 解决了在屏幕上展示相机画面过程中最难处理的部分，包括对画面宽高比、缩放和旋转的处理。不同的设备会导致不一致的行为，包括设备、屏幕尺寸、摄像头硬件支持水平，还会需要适配诸如分屏模式、不同锁定方向和可动态调节尺寸的展示窗口等显示模式，为了解决这些问题并在多种设备上提供无缝体验，PreviewView 还做了一些兼容性的处理。

PreviewView 的实现模式

PreviewView 是 FrameLayout 的子类，它会使用 SurfaceView 或者 TextureView 展示来自相机捕捉到的画面。一旦相机准备好，就会创建一个预览画面 (preview surface) 的实例，并在相机使用过程中尽量持有该实例，如果相机还在工作中却提前释放了所持有的预览画面 (preview surface) 实例，就会重新创建一个。

当涉及到诸如功耗和响应时间这些关键指标时，SurfaceView 的表现一般都比 TextureView 要好，这也是为什么 PreviewView 会将 SurfaceView 作为默认实现模式的原因。然而，一些设备 (主要是一些旧版设备) 会在预览画面 (preview surface) 过早释放时出现闪退的情况。可惜的是，使用 SurfaceView 时无法控制何时对画面 (surface) 进行释放，因为这是由 View 层级结构所控制的。因此在这些设备上，PreviewView 只能使用 TextureView 作为实现模式。另外在需要对相机预览界面进行旋转、改变透明度或加入动画的情况下，您也应该强制 PreviewView 使用 TextureView 作为实现模式。



您可以通过调用 `PreviewView.setPreferredImplementationMode(ImplementationMode)` 并设置 `ImplementationMode` 参数为 `SURFACE_VIEW` 或 `TEXTURE_VIEW` 来更改 `PreviewView` 的实现模式。当首选模式设置为 `SURFACE_VIEW` 时，`PreviewView` 会尽可能遵循您的设置 (使用 `SurfaceView`)；而当首选模式设置为 `TEXTURE_VIEW` 时，`PreviewView` 会确保一直使用 `TEXTURE_VIEW` 模式。

△ 在开始使用 `PreviewView` 之前，请务必通过调用 `Preview.setSurfaceProvider(PreviewView.createSurfaceProvider())` 来设置您想要的实现模式。

下面介绍如何设置 `PreviewView` 的实现模式：

```
// 进行相机画面预览之前，设置想要的实现模式
previewView.preferredImplementationMode = ImplementationMode.SURFACE_VIEW

// 将 previewView 设置到 preview 用例中来开始进行相机画面预览
preview.setSurfaceProvider(previewView.createSurfaceProvider(cameraInfo))
```

PreviewView - Preview

`PreviewView` 通过处理创建 `Preview` 用例所需要的 `SurfaceProvider`，来启动一个预览画面的数据流。`SurfaceProvider` 会准备好需要提供给相机的 `Surface`，用来对预览画面的数据流进行展示，并负责在必要时重新创建 `Surface`。`PreviewView.createSurfaceProvider(CameraInfo)` 接收一个 nullable 的 `CameraInfo` 实例。`PreviewView` 会结合所传入的 `CameraInfo` 参数，以及您所设定的实现模式和当前相机具备功能，来决定内部如何进行功能上的实现。如果您所传入的 `CameraInfo` 是一个 null，那 `PreviewView` 会使用 `TextureView` 作为实现模式，因为它无法确定所选的相机若使用 `SurfaceView` 是否可以正常工作。

一旦您创建好了 `Preview` 用例和一些别的所需要的实例后，将它们绑定至 `LifecycleOwner`，使用所绑定的相机的 `CameraInfo` 来创建 `SurfaceProvider`，再将其绑定至 `Preview` 用例，调用 `Preview.setSurfaceProvider(SurfaceProvider)` 来启动预览画面数据流。

下面的例子展示了如何将 `PreviewView` 绑定至 `Preview` 来开启预览画面数据流：

```
// 创建 preview 用例
val preview = Preview.Builder().build()

// 将 preview 和其他需要的用例绑定到 lifecycle 中
val camera = cameraProvider.bindToLifecycle(lifecycleOwner, cameraSelector, preview)

// 使用所绑定相机的 cameraInfo 创建 surfaceProvider
val surfaceProvider = previewView.createSurfaceProvider(camera.cameraInfo)

// 将 surfaceProvider 绑定至 preview 用例来启动预览
preview.setSurfaceProvider(surfaceProvider)
```

PreviewView - 缩放 (scale) 类型

`PreviewView` 提供了一个 API，通过它可以让您控制预览画面的样式是怎样的 (how) 和在父级视图中的位置 (where)：

- **how** 决定将预览画面放置于 (**FIT**) 父级视图中还是填充于 (**FILL**) 父级视图中；
- **where** 决定预览画面相对于父级视图来说，是左上方对齐 (**START**)，居中对齐 (**CENTER**) 还是右下方对齐 (**END**)。

"how" 和 "where" 所组合出来的结果，代表了 `PreviewView` 支持的缩放 (scale) 类型，包括



的是 FIT_CENTER 和 FILL_CENTER，前者将预览界面在保证宽高比的前提下进行缩放然后居中，后者不会进行缩放，保证居中但是可能会导致画面被裁剪。

有两种方法可以设置缩放 (scale) 类型:

- 通过在 XML 布局文件中设置 PreviewView 的 `scaleType` 属性来实现，如下示例所示:

```
<androidx.camera.view.PreviewView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:scaleType="fitEnd" />
```

- 在代码中通过调用 `PreviewView.setScaleType(ScaleType)` 来实现，如下示例所示:

```
previewView.setScaleType(ScaleType.FIT_CENTER)
```

想要获取到当前 PreviewView 所使用的缩放 (scale) 类型，调用 `PreviewView.getScaleType()` 即可。

PreviewView - 摄像头控制操作

根据相机摄像头传感器的方向、设备的旋转方向、以及显示模式和预览比例，PreviewView 可能会对从相机接收到的预览帧进行相应地缩放、旋转和转换处理，以便在 UI 界面中能正确展示。这也是为什么将 UI 坐标转换成摄像头传感器坐标是很重要的。在 CameraX 中，这种转换是由 `MeteringPointFactory` 完成的，它可以通过 PreviewView 提供的 API 进行创建: `PreviewView.createMeteringPointFactory(cameraSelector)`，其中 `CameraSelector` 参数代表所传入画面流数据的摄像头。

当您需要实现轻点对焦 (tap-to-focus) 功能的时候，PreviewView 的 `MeteringPointFactor` 轻易就可做到。尽管相机预览中默认启用了自动对焦 (需要摄像头支持)，但在 PreviewView 上点击时，您还是可以控制对焦目标。`MeteringPointFactory` 会将对焦目标的坐标转换为摄像头传感器的坐标，然后再使用摄像头对该区域进行对焦。

下面的示例展示了如何使用触摸监听器 (touch listener) 在 PreviewView 上实现轻点对焦功能:

```
fun onTouch(x: Float, y: Float) {
    // 创建 MeteringPoint, 命名为 factory
    val factory = previewView.createMeteringPointFactory(cameraSelector)

    // 将 UI 界面的坐标转换为摄像头传感器的坐标
    val point = factory.createPoint(x, y)

    // 创建对焦需要用的 action
    val action = FocusMeteringAction.Builder(point).build()

    // 执行所创建的对焦 action
    cameraControl.startFocusAndMetering(action)
}
```

另一个在相机预览界面中常用的功能是捏拉缩放 (pinch-to-zoom)，它可以让您通过在预览界面进行捏拉来实现画面的缩放操作。想要在 PreviewView 上实现它，在其之上添加一个触摸监听器，并将其绑定到缩放手势监听器 (scale gesture listener) 上。这样就可以做到拦截捏拉手势，然后相应地更新摄像头的缩放比例。

下方的示例展示了如何在 PreviewView 上实现捏拉缩放 (pinch-to-zoom) 操作:



```
// 创建一个名为 listener 的回调函数，当手势事件发生时调用这个回调函数
val listener = object : ScaleGestureDetector.SimpleOnScaleGestureListener() {
    override fun onScale(detector: ScaleGestureDetector): Boolean {
        // 获取当前的摄像头的缩放比例
        val currentZoomRatio: Float = cameraInfo.zoomRatio.value ?: 1F

        // 获取用户捏拉手势所更改的缩放比例
        val delta = detector.scaleFactor

        // 更新摄像头的缩放比例
        cameraControl.setZoomRatio(currentZoomRatio * delta)
        return true
    }
}

// 将 PreviewView 的触摸监听器绑定到缩放手势监听器上
val scaleGestureDetector = ScaleGestureDetector(context, listener)

// 将 PreviewView 的触摸事件传递给缩放手势监听器上
previewView.setOnTouchListener { _, event ->
    scaleGestureDetector.onTouchEvent(event)
    return@setOnTouchListener true
}
```

PreviewView - 如何进行测试

PreviewView 可在各种不同的 Android 设备上提供一致的相机处理行为，这要归功于 CameraX 在 [自动化测试实验室](#) 中对 PreviewView 及其他 API 上进行的投资。这些测试主要分为两个主要类别：

- **单元测试**可以结合当前的实现模式，缩放类型和 MeteringPointFactor 来验证 PreviewView 的行为。当出现父级视图的大小更改，或是展示的布局发生了变化，亦或是被绑定到 [Window](#) 上的情况时，单元测试还可以确保 PreviewView 在适当的时候能够正确地去调整预览画面；
- **集成测试**可以确保 PreviewView 集成到应用中，可以正常去显示或者停止显示来自相机的画面数据流。这些测试会验证 preview 在各种情况时的状态，包括在应用运行时进行多次关闭然后重新打开，切换前置后置摄像头，以及应用的生命周期销毁后重新创建的情况。当前这些测试覆盖的主要范围是使用 TextureView 作为 PreviewView 的实现模式，因为使用 SurfaceView 之后想要捕获相机预览开始和结束时的信号会非常困难。

总结

综上所述：

- PreviewView 是一个自定义的 View，它可以方便地展示相机的预览画面；
- PreviewView 默认使用 SurfaceView 作为它预览画面 (preview surface) 的实现，但是在需要的时候会转而使用 TextureView；
- 将诸如 ImageCapture 和 ImageAnalysis 这样的用例绑定到 LifecycleOwner 上，创建一个 surfaceProvider，将其绑定到 Preview 用例来启动相机预览；
- 通过定义 PreviewView 的缩放类型来控制预览画面的展示方式；
- 通过给 PreviewView 创建 MeteringPointFactory 来实现对焦功能；
- 通过给 PreviewView 设置手势监听来实现捏拉缩放功能。

想了解更多关于 CameraX 的优秀功能吗？请查阅以下资料及推荐阅读：

- [Android 开发文档 | CameraX 概览](#)
- [developer.android.google.cn... Codelab | CameraX](#)
- [Codelab | CameraX 使用指南](#)

groups.google.com/a/android/

- 示例代码 | 使用 CameraX 构建相机应用
[github.com/android/came...](https://github.com/android/camera)

如果您有 PreviewView 或 Preview 相关的问题，欢迎在下方评论区留言。感谢您的阅读！

[点击这里](#)了解更多 CameraX 相关内容

发布于 06-17

[Android](#) [Android 开发](#)

推荐阅读



原来这样就可以开发出一个百万量级的Android相机

腾讯云技术社区



安卓学习<一>：布局与控件

Arendelle

Android屏幕适配-重点盘点

享学课堂诚邀作者：周周 转载请声明出处！引子屏幕适配是 android 开发/面试 绕不开的一个问题。本文将屏幕适配的知识要点完整展现给

享学课堂 Java架构

全新定义 View 的尺寸

HenCoder Android UI 部分 2-2：全新定义 View 的尺寸

扔物线

发表于HenCo...

还没有评论

写下你的评论...



▲ 赞同 10 ▼

● 添加评论

🔗 分享

♥ 喜欢

★ 收藏

📄 申请转载

...