Android 下的摄像头编程

PB20061372 朱云沁　Dec. 8, 2023

1　实验目的

1. 学习 Android 下进行摄像头控制的基本方法.
2. 理解回调函数.

2　实验原理

2.1　摄像头相关的包和类

- android.hardware: 包含了如摄像头、传感器等硬件的支持.
  - Camera: 用于设置图像捕获设置, 开始/停止预览, 拍照以及获取用于视频编<
  - CameraManager: 摄像头管理器, 专门用于检测系统摄像头、打开系统摄像头.
  - CameraCharacteristics: 摄像头特性. 该对象通过 CameraManager 来获取.
  - CameraDevice: 代表系统摄像头, 该类的功能类似于早期的 Camera 类.
  - CameraCaptureSession: 当程序需要预览、拍照时, 都需要先通过该类的实例创建 Sen.
  - CameraRequest.Builder: 负责生成 CameraRequest 对象.
  - ImageReader: 完成摄像头捕获的图像数据到界面上显示控件的显示.

2.2　回调函数

回调函数是通过函数指针调用的函数, 用于在特定事件发生时响应该事件或条件. 在多媒体操作中, 比如摄像头捕获图像, 回调函数通常用于处理采集完图像后的特定操作.

3　实验内容及结果

3.1　环境配置

示例程序在 M1 芯片 macOS 下编译, 使用 Android Studio 2023.1.1.26 自带 JetBrains Runtime 17.0.7 作为 JDK. 使用 Android Gradle Plugin 8.2.0, Gradle 8.2. 目标 SDK 版本为 34 (Android 14.0). 遵循示例代码 build.gradle, 将最小 SDK 版本设置为 21 (Android 5.0).

示例程序在 HUAWEI P40 Pro 下调试运行, 系统为 HarmonyOS 4.0.0, 兼容 Android 12.0.

3.2　调试示例程序

考虑到示例代码过于古早, 我们基于 Kotlin 模版创建新项目, 对其进行移植, 涉及的文件包括 MainActivity.java (MainActivity.kt), activity_main.xml, AndroidManifest.xml 等. 利用 Android Studio 提供的自动转换工具, 将 Java 代码转换为 Kotlin 代码, 并逐一解决错误和警告 (忽略 createCaptureSession 和 getDefaultDisplay 的废弃警告). 项目正常同步、构建, 得到应用名为 VideoCapture, 图标为科大校徽. 运行结果如下:

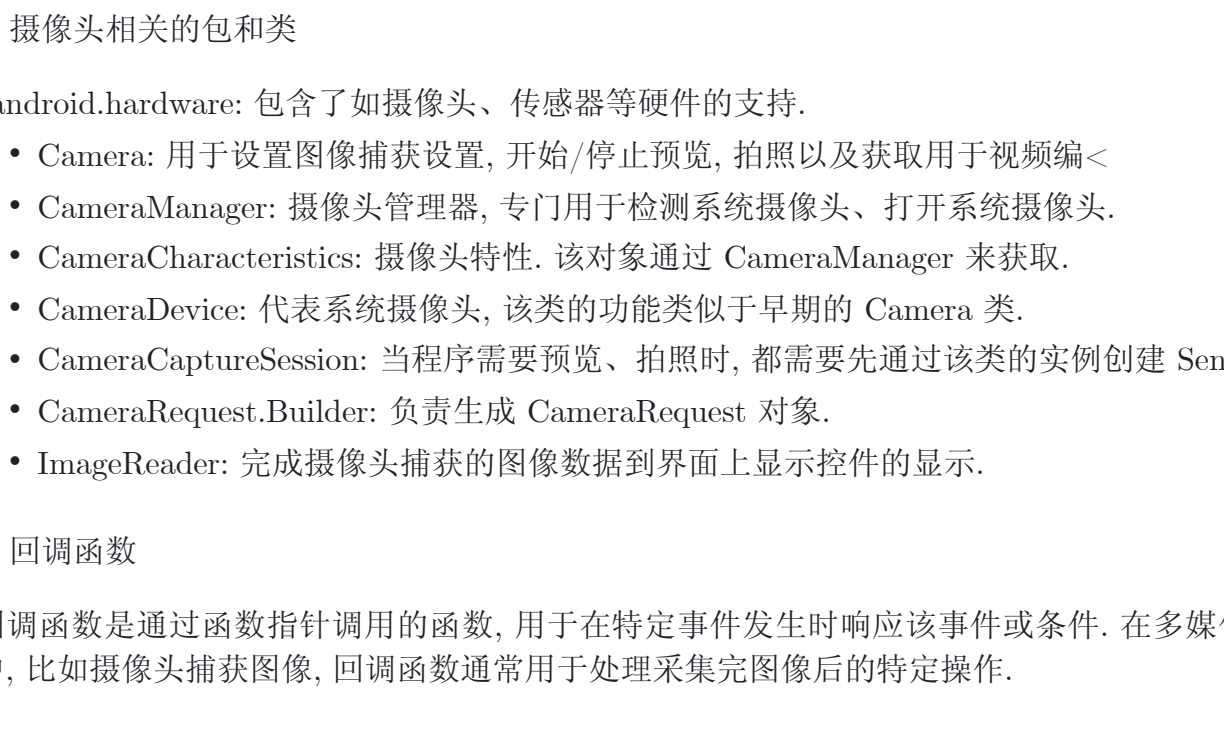

Figure 1. 在 Android Studio 中构建并运行示例程序.

设置摄像头权限后, 应用正常启动, 在 TextureView 中实时预览摄像头捕获的图像. 显示, ImageView 隐藏.

3.3　示例程序流程示意图

示例程序中, TextureView 是用于显示摄像头预览的视图, ImageView 是用于显示拍摄到的图像的视图, Button 是用于触发拍照或返回操作的按钮. TextureView 通过其 SurfaceTexture 对象新建 Surface 获取图像数据, ImageView 通过 ImageReader 的 Surface 获取图像数据然后设置位图. 部分函数的功能如下:

- onCreate:, 初始化 TextureView, ImageView 和 Button, 设置监听器.
- openCamera：配置 ImageReader, CamaraManager, 打开摄像头.
- takePreview：创建请求, 配置自动对焦、闪光灯等, 创建会话, 开始预览.
- takePicture：创建请求, 配置自动对焦、闪光灯等, 捕获静态图像.
- onImageAvailable: 获取拍摄到的图像数据, 转换为 Bitmap 格式, 在 ImageView 中显示.
- closeCamera：释放资源, 包括摄像头会话、设备、ImageReader 等, 并停止后台线程.

与访问摄像头相关的步骤的流程图如下:

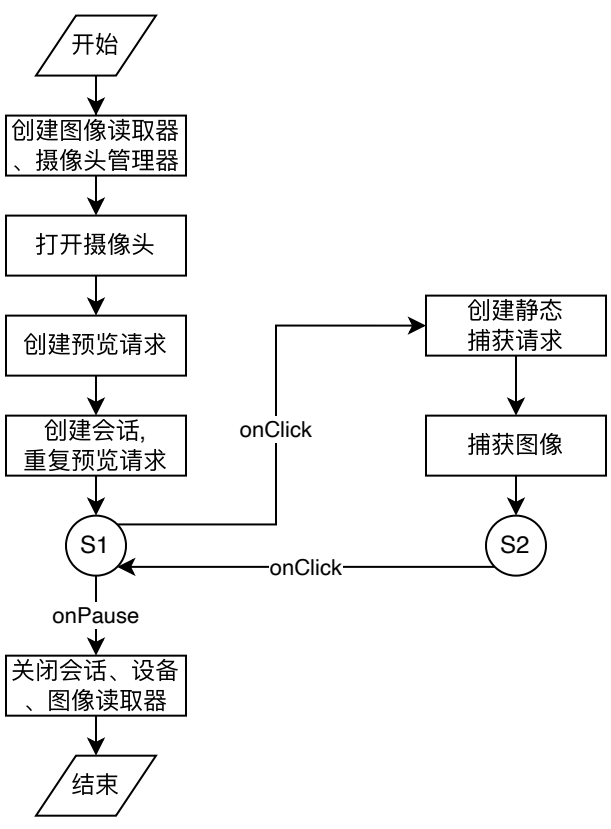

Figure 2. 示例程序访问摄像头的流程示意图.

上图中状态 S1, TextureView 可见, ImageView 不可见, Button 显示为拍照按钮. 状态 S2, TextureView 不可见, ImageView 可见, Button 显示为返回按钮.

4　思考题

1. 示例代码中用到的 TextureView 与一般的 View 有什么不同?

   TextureView 基于 SurfaceTexture, 通过 GPU 直接将纹理 (Texture) 渲染到屏幕上, 在后台进行图像缓冲区的处理, 不会阻塞 UI 线程, 因而相较一般的 View 可以更加高效地显示图像, 特别适用于展示实时视频流或者摄像头预览等需要连续渲染的场景.

2. 阐述一下个人关于 Callback 函数用途的理解.

   - 回调函数用于处理异步操作和监听事件.
   - 回调函数可以实现代码的解耦, 将特定功能的实现分离出去, 提高代码的模块化和可维护性. 通过传递不同的回调函数, 可以实现不同的行为.

   在使用 Camera2 API 这样的异步操作频繁的情况下, 回调函数使得程序灵活、可控. 例如, CameraDevice.StateCallback 用于监听摄像头设备状态的变化, ImageReader.OnImageAvailableListener 用于监听图像可用事件的回调接口, CameraCaptureSession.StateCallback 用于监听相机捕获会话的状态变化.

   另一方面, 在 UI 线程中, 回调函数不仅实现了 UI 事件异步处理, 还使得 UI 控件更加通用、灵活, 可以根据需求做出不同的响应. 例如 Button 的 OnClickListener, 用于监听按钮点击; TextureView.SurfaceTextureListener, 用于监听 TextureView 的 SurfaceTexture 变化.