Android Camera流程

Camera2的操作流程

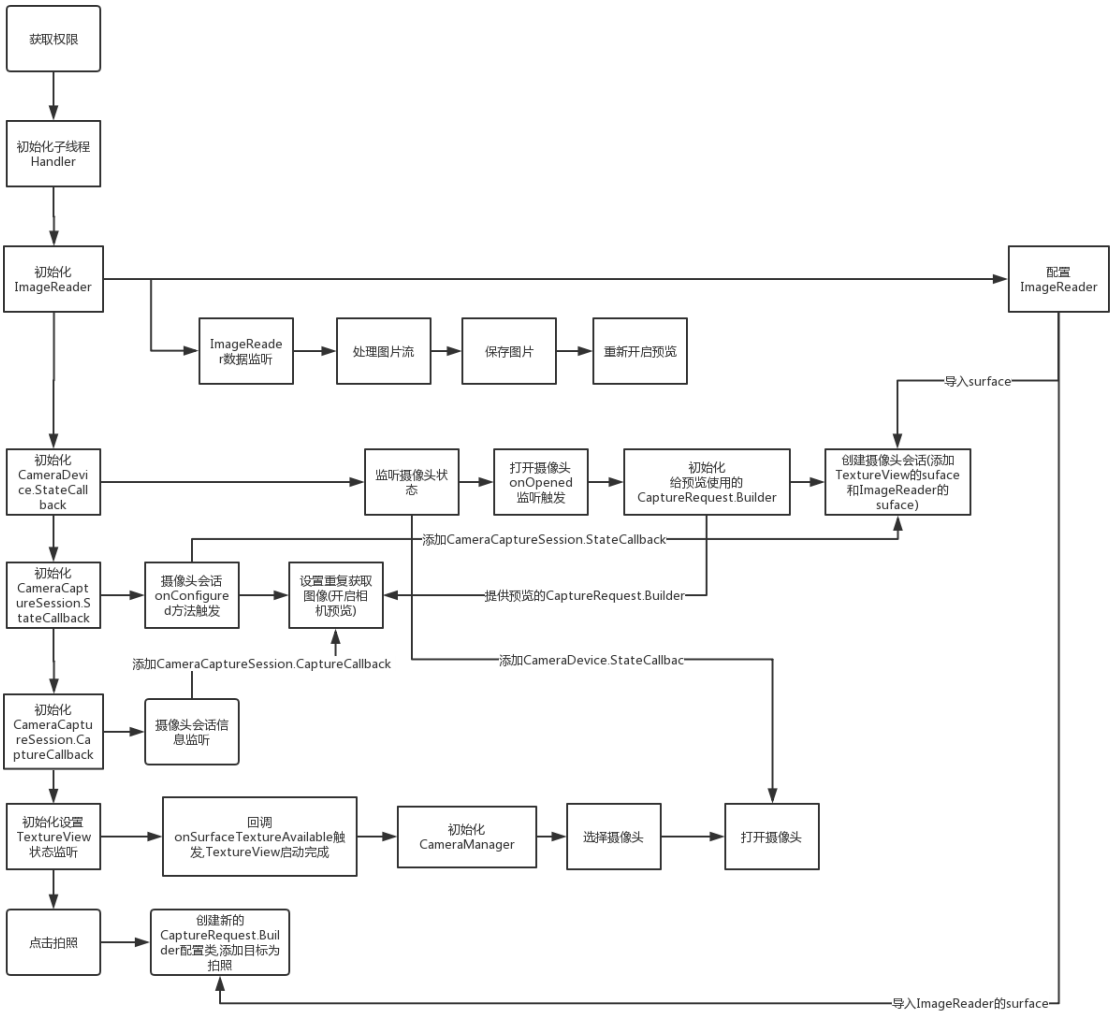
在上面的API介绍里,你是不是对这么多的配置类/会话类/接口回调类感到眼花缭乱?是的,Camera2的使用是相当眼花缭乱的,但是我们抓住一条线慢慢从上面跟到下面就应该能明白是怎么一回事了.下面我们来简单介绍一些Camera2的操作流程:

初始化流程:

1. 初始化动态授权,这是基本操作
2. 初始化一个子线程的Handler,Camera2的操作可以放在主线程也可以放在子线程.按例一般都是子线程里,但是Camera2只需要我们提供一个子线程的Handler就行了.
3. 初始化ImageReader,这个没有初始化顺序要求,并且它有数据回调接口,接口回调的图片数据我们直接保存到内部存储空间,所以提前初始化提供给后续使用.
4. 初始化TextureView,添加TextureView的接口回调.
5. 在TextureView的接口回调里回调启用成功方法后,我们开始初始化相机管理类initCameraManager
6. 然后继续初始化CameraDevice.StateCallback 摄像头设备状态接口回调类,先初始化提供给后续使用.(在这个接口类的开启相机的回调方法里,我们需要实现创建预览图像请求配置和创建获取数据会话)
7. 继续初始化CameraCaptureSession.StateCallback 摄像头获取数据会话类的状态接口回调类,先初始化提供给后续使用.(在这个接口类的配置成功回调方法里,我们需要实现预览图像或者实现拍照)
8. 继续初始化CameraCaptureSession.CaptureCallback 摄像头获取数据会话类的获取接口回调类,先初始化提供给后续使用.(啥都不干)
9. 判断摄像头前后,选择对应id
10. 打开指定id的摄像头
11. 实现拍照

逻辑流程:

动态相机权限获取 >> 设置TextureView回调 >> TextureView启用成功回调方法触发 >> 选择摄像头 >> 打开相机 >> 相机开启回调方法触发 >> 创建CaptureRequest.Builder配置类 >> 设置配置类图像预览模式 >> 配置类导入需要显示预览的TextureView的surface >> 创建数据会话 >> 数据会话的配置成功回调方法触发 >> 创建预览图像 >> 预览图像显示成功 >> 按键点击拍照 >> 创建新的CaptureRequest.Builder配置类,添加目标为拍照 >> 配置类导入ImageReader的surface >> 数据会话使用这个配置类创建拍照 >> ImageReader的接口类图片可用方法触发 >> 保存图片



CSDN @沅霖

拍照成功后,你下一个要面临的麻烦可能就是**内存泄露,注意**要释放的资源,否则一直持有会导致内存泄露.

一般情况下会有以下几个操作导致内存泄露:

1.CaptureRequest 如果没有释放Surface, 一定操作释放 mCaptureRequest.removeTarget(mSurface);

2.SurfaceTexture 需要被释放，除了 mSurfaceTexture.release();

也可以用这种mTextureView.getSurfaceTextureListener().onSurfaceTextureDestroyed(mTextureView.getSurfaceTexture());方式释放SurfaceTextur,

但是在上面的public boolean onSurfaceTextureDestroyed(SurfaceTexture surface) 回调中你需要返回true或者自己执行surface.release(); 这样才会运行onSurfaceTextureDestroyed回调

3.Surface 需要被释放

4.释放是需要按**顺序一个一个释放**的

5.如果你想实现一个工具类来以建造者模式来创建相机拍照,请注意TextureView不能传入工具类里,因为你需要实现setSurfaceTextureListener监听,而这个监听你是无法在工具类里释放它的,它是需要在activity里才能被释放,请切记!

你最好只传入SurfaceTexture给工具类

内存泄漏可以通过systrace中GC的情况来确认，一般有内存泄漏的情况下，内存会持续上升，并且执行GC后内存回落很少，销毁后内存下降也不明显

特征	正常 GC	内存泄漏导致的 GC
内存曲线	锯齿状波动，整体稳定	持续上升，锯齿底部越来越高
GC 后内存回落	明显回落至合理水平	回落很少，甚至不回落
组件销毁后内存变化	显著下降	下降不明显，或几乎不变
堆中对象状态	无用对象被完全回收	存在大量 “本应销毁” 的对象实例

是的，Systrace 中可以查看内存相关信息，主要通过内存使用趋势图表和关键事件标记，帮助分析应用或系统的内存变化情况。具体能看到的内存信息及查看方式如下：

1. 可查看的内存相关数据

- **内存使用量趋势**：应用或系统的总内存占用（如 Java 堆、Native 堆、栈内存等）随时间的变化曲线。
- **内存分配事件**：如频繁的对象创建（可能导致 GC 频繁）。
- **GC 事件**：垃圾回收的时间点、类型（如 Partial GC、Full GC）及持续时长。
- **内存回收相关操作**：如系统内存紧张时的 `memreclaim`（内存回收）事件。
- **进程级内存变化**：单个进程（如你的拍照进程）的内存占用变化，可与其他进程对比。

2. 在 Systrace 中查看内存的方法

（1）录制时勾选内存相关模块

确保录制 Systrace 时包含内存相关的追踪模块，否则可能看不到内存数据：

- 命令行录制时，添加 `memreclaim`、`dalvik`（包含 GC 信息）等模块，例如：

  bash

  ```
  python -m systrace -t 10 sched gfx view memreclaim dalvik
  ```

- Android Studio 中录制时，在配置界面勾选 **Memory** 相关选项（如 “Memory Allocation”“GC”）。

（2）在报告中找到内存面板

打开生成的 Systrace HTML 报告后：

- **全局内存趋势**：在顶部的汇总面板中，可能会有 **“Memory”** 图表，显示系统整体内存使用趋势。
- **进程级内存**：在左侧 “Processes” 面板中，展开你关注的进程（如拍照进程），其时间轴中会有 **“mem”** 或 **“memory”** 相关的子项，显示该进程的内存占用变化（通常以折线图形式呈现）。
- **GC 事件**：在进程时间轴中，会以 `GC` 或 `dalvik-gc` 标记显示垃圾回收事件，点击可查看具体信息（如持续时间、回收内存量）。

（3）分析内存变化与操作的关联

- 结合你的操作（如打开相机、拍照、关闭页面），观察内存曲线是否有异常变化：
  - 正常操作后，内存应在 GC 后回落；
  - 若操作后内存持续上升且 GC 无法有效回收，可能存在内存泄漏嫌疑。

3. 注意事项

- Systrace 的内存数据是**宏观趋势性的**，主要用于观察 “内存变化规律” 和 “与其他事件（如 GC、用户操作）的关联”，但无法显示具体对象的内存占用（如某个 Bitmap 占用多少内存）。
- 若需精准分析内存对象，需结合 **Android Profiler 的 Memory 面板** 或 **MAT（Memory Analyzer Tool）** 等工具，通过堆转储（Heap Dump）查看具体对象的分配和引用情况。

总之，Systrace 是分析内存 “趋势和异常模式” 的有效工具，可作为内存问题（如泄漏、过度分配）的初步排查手段，再结合其他工具深入定位。