Android Media Provider

软件设计文档

# 修订历史

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author | Reviewer | Abstract |
| 2017/12/20 | Liu, Mingyang | None | Initial creation. |
| 2018/01/31 | Liu, Mingyang | Deng Jerry | Added data flow and module descriptions. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 目录

目录

[Android Media Provider 1](#_Toc510018450)

[软件设计文档 1](#_Toc510018451)

[1 修订历史 2](#_Toc510018452)

[2 目录 3](#_Toc510018453)

[3 简介 4](#_Toc510018454)

[3.1 目的 4](#_Toc510018455)

[3.2 参考资料 4](#_Toc510018456)

[4 系统概览 5](#_Toc510018457)

[5 系统架构 6](#_Toc510018458)

[6 组件设计 7](#_Toc510018459)

[6.1 MIO (Media I/O) 7](#_Toc510018460)

[6.1.1 操作 V4L2 7](#_Toc510018461)

[6.1.2 管理 MMZ 7](#_Toc510018462)

[6.2 (mediad) 媒体守护进程 8](#_Toc510018463)

[6.3 (mediac) 媒体客户端 8](#_Toc510018464)

[6.4 消费者 8](#_Toc510018465)

[7 附录 9](#_Toc510018466)

# 简介

## 目的

把媒体功能分散到不同的模块中，目的是：

* 便于为计算机视觉算法提供视频数据；
* 提高 Android 视频供应的稳定性和性能；
* 降低对 Android 的依赖。

## 参考资料

Linux V4L2 API 参考文档。

Android camera framework 文档。

# 系统概览

把 Android Camera HAL 的视频供应功能拆分为：

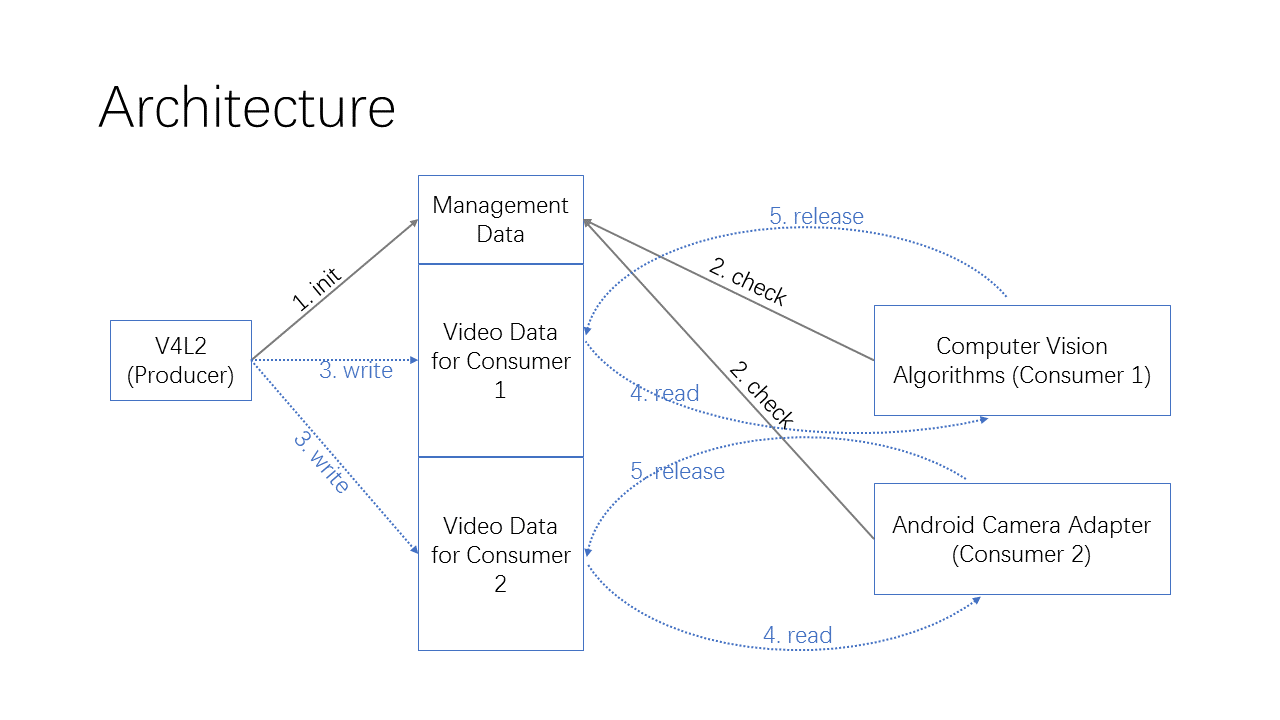
* MMZ (Media Memory Zone) Kernel Driver: 一个标准的 Linux Kernel 模块，在 Android 上执行 lsmod 命令可看到 mmz 1544 2 - Live 0x00000000 (0). 其功能只有一个，就是申请一块足够大的物理内存，目前为 64MB。用户空间的多个进程通过 memory map 的方式获得这块内存，实现高效的进程间通信。
* Media Daemon: 一个独立的进程，其角色是视频的生产者，调用 V4L2 接口从相机获得视频帧，将其保存到 MMZ. 如果视频帧不是 YUV 格式，它还负责调用解码器将其转换为 YUV 格式。
* Media client: 媒体客户端，通过 UNIX socket 与media daemon 通信。它只是简单地把媒体控制命令发送给 media daemon.
* Camera HAL Camera Adapter: 位于标准的 Android 框架内，其角色是视频的消费者，从 MMZ 读取视频帧，提供给 Android 上层框架。

用户空间有自己的共享内存 API shmget(2), 但 Android 并未在 C 库里导出该系统调用，但可以自己封装。由 Linux 内核模块申请的内存作为共享内存的优点是内存块连续，缺点是可移植性差。本方案权衡后决定由内核模块申请。

使用场景中，除了 Camera Adapter 是一个消费者外，还有一个消费者——图像算法。因此，本方案中一个生产者对应多个消费者。

# 系统架构

下图所示为简化的数据走向图：



MMZ 是一块连续的物理内存空间。在逻辑，它被分为 3 部分：一个用于管理的数据结构区域 (management data 区域)、以及两个存放视频数据的区域。生产者和消费者都通过 memory map 的方式把 MMZ 映射到自己的进程虚拟地址空间。

对 MMZ 的使用流程为：

1. 生产者初始化 MMZ 的 management data 区域；
2. 消费者检查 management data 区域是否已被初始化。若未初始化，则隔一段时间再次检查；若已初始化，则继续后续操作。
3. 生产者把从 V4L2 获得的视频数据写入 MMZ 的视频数据区域。此处会产生一个视频帧副本，故复制完就立刻把 buffer 归还给 V4L2 驱动。若所有的数据都被消费者使用中，即没有空闲的空间，则不覆盖任何数据，本次写入失败。这是因为消费者（尤其是计算机视觉算法）可能较长时间地占用视频帧，若强制正在使用中的视频帧，可能导致算法异常。
4. 消费者从视频数据区域读取一帧图像。若没有新产生的帧，则返回失败。
5. 消费者使用完一帧图像，将其释放。

# 组件设计

## MIO (Media I/O)

MIO 模块封装了对 V4L2 的操作和 MMZ 的映射及管理。

### 操作 V4L2

主要有以下操作：

* 打开/关闭 /dev/video\* 设备。
* 查询设备能力。
* 设置视频帧的宽高及像素格式。
* 设置帧率。
* 开始/停止产生视频帧。
* 获取/释放视频帧。

释放 V4L2 视频帧 buffer 的线程和使用该 buffer 的线程，不是同一个，故需要同步二者。本模块内部会记录正在使用中的 buffer 个数，仅当所有的 buffer 都不在使用中时，才调用 VIDIOC\_REQBUFS 接口把申请的所有 V4L2 buffer 都释放。

### 管理 MMZ

规划 MMZ, 将其分为管理数据区域和视频数据区域：

* Management data 区域：包含整个 MMZ 的管理数据。
* 2 个视频数据区域，其内容为：
  + 该 视频数据区域的控制信息，如图像格式、宽高、占用的内存大小。
  + 2 个 distributions. 由于有多个消费者，每个消费者应保持独立、互不影响，故生产者的数据需要被分发到不同的地方，每个地方称为一个 distribution, 每个 distribution 包含生产者的一份数据副本。

MMZ 中保存的视频帧为 YUV 格式，每一帧所占用的内存空间大小相等。这样有助于高效地利用 MMZ.

初始化过程中，根据视频帧的宽高和像素格式，计算每个视频帧所占用的内存大小。进一步计算每个视频数据区域所能容纳的视频帧个数。

初始化完成后，置相应的标志。

使用加锁的读写索引管理视频数据区域，解决读写竞争。

视频区域的每个帧都有一个逻辑上的头部，描述其状态（无数据、未使用、已使用）及其他基本信息。

为生产者提供向 MMZ 写视频帧的接口。

为消费者提供从 MMZ 读视频帧，及向 MMZ 归还视频帧的接口。

## (mediad) 媒体守护进程

该进程作为 Linux daemon 进程运行，其主要工作为：

* 以生产者的身份初始化 MIO.
* 为每个生产者（有两个相机）创建一个线程，用于从 V4L2 中获取视频数据，并保存到 MMZ 中。
* 创建通信线程，用于监听并响应来自 media client 的请求。通信方式为 UNIX socket.

## (mediac) 媒体客户端

Media client 只是简单地封装了一些媒体控制操作，它把这些操作命令通过 UNIX socket 发送给 media daemon, 由后者负责具体的执行。

Media client 主要提供给两个使用者：

Android camera HAL, 目的是使之可以灵活地控制底层相机设备。此时 media client 被编译为动态库。

开发者，目的是使开发者可以方便地调试相机。此时media client 被编译为 Linux 可执行程序。

## 消费者

目前的使用场景中有两类消费者：

* 计算机视觉算法；
* Android camera HAL.

本方案为了节约内存，消费者获得图像时，只获得 MMZ 中的数据地址，而不产生副本。即 MMZ 中的一份数据会被生产者和消费者共享，由此产生的读写竞争问题由 MIO 模块负责解决。

# 附录