Android 5.0 Camera 系统源码分析(1)：CameraService 启动流程

标签： MTKandroidcamera 源码分析调用流程
2016-08-01 15:11 1594 人阅读 评论(2) 收藏 举报

1. 前言

本文将分析 **Android** 系统源码，从 frameworks 层到 hal 层，暂不涉及 app 层和 kernel 层。由于某些函数比较复杂，在贴出代码时会适当对其进行简化。

本文属于自己对源码的总结，仅仅是贯穿代码流程，不会深入分析各个细节。

分析 android 系统源码，需要对 android 系统的某些知识点有所了解

涉及的知识点有：

（1）Android 系统的**智能**指针 - 参考老罗的 Android 系统的智能指针（轻量级指针、强指针和弱指针）的实现原理分析

（2）Android 进程间通信 Binder - 参考老罗的 Android 进程间通信（IPC）机制 Binder 简要介绍和学习计划

（3）Android 硬件抽象层(HAL) - 参考老罗的 Android 硬件抽象层（HAL）概要介绍和学习计划

2. frameworks 层

Android 的各个子模块的启动都是从它们的 Service 的启动开始的，所以我们将从 CameraService 的启动开始分析。CameraService 的启动就在 MediaServer 的 main 函数中，代码路径在：frameworks/av/media/mediaserver/main_mediaserver.cpp

[cpp] view plain copy

```cpp
int main(int argc __unused, char** argv)
{
    ......
    CameraService::instantiate();
    ......
}
```

CameraService 类定义如下：

[cpp] view plain copy

```cpp
class CameraService :
    public BinderService<CameraService>,
    public BnCameraService,
    public IBinder::DeathRecipient,
    public camera_module_callbacks_t
{
    static char const* getServiceName() { return "media.camera"; }
    ......
}
```

mediaserver 的 main 函数中调用了 CameraService 的 instantiate 函数来创建实例，该函数的实现在其父类 BinderService 中实现

[cpp] view plain copy

```cpp
template<typename SERVICE>
class BinderService
{
    static status_t publish(bool allowIsolated = false) {
        sp<IServiceManager> sm(defaultServiceManager());
        return sm->addService(
                String16(SERVICE::getServiceName()),
                new SERVICE(), allowIsolated);
    }

    static void instantiate() { publish(); }

}
```

1. instantiate 函数只是简单的调用了 publish 函数

2. publish 函数先构造 CameraService，再通过 addService 函数将它注册到 ServiceManager 当中，而 getServiceName 函数获取到的值为“media camera”。这一切都是为了 binder 通信做准备

3. 这里使用了 c++模版，从上面的 CameraService 类定义中可以看出，这里的 SERVICE 等于 CameraService，也就是说 publish 函数中的 new SERVICE 等于 new CameraService

4. 同时还使用了智能指针，也就是说除了调用 CameraService 的构造函数外，还会调用 onFirstRef 函数

[cpp] view plain copy

```cpp
1.  CameraService::CameraService()
2.      :mSoundRef(0), mModule(0)
3.  {
4.      ALOGI("CameraService started (pid=%d)", getpid());
5.      gCameraService = this;
6.
7.      for (size_t i = 0; i < MAX_CAMERAS; ++i) {
8.          mStatusList[i] = ICameraServiceListener::STATUS_PRESENT;
9.      }
10.
11.     this->camera_device_status_change = android::camera_device_status_change;
12. }
13.
14. void CameraService::onFirstRef()
15. {
16.     LOG1("CameraService::onFirstRef");
17.
18.     BnCameraService::onFirstRef();
19.
20.     if (hw_get_module(CAMERA_HARDWARE_MODULE_ID,
21.                 (const hw_module_t **)&mModule) < 0) {
22.         ALOGE("Could not load camera HAL module");
23.         mNumberOfCameras = 0;
24.     }
25.     else {
26.         ALOGI("Loaded \"%s\" camera module", mModule->common.name);
27.         mNumberOfCameras = mModule->get_number_of_cameras();
28.         if (mNumberOfCameras > MAX_CAMERAS) {
29.             ALOGE("Number of cameras(%d) > MAX_CAMERAS(%d).",
30.                     mNumberOfCameras, MAX_CAMERAS);
31.             mNumberOfCameras = MAX_CAMERAS;
32.         }
33.         for (int i = 0; i < mNumberOfCameras; i++) {
34.             LOG1("setCameraFree(%d)", i);
35.             setCameraFree(i);
36.         }
37.
38.         if (mModule->common.module_api_version >=
39.                 CAMERA_MODULE_API_VERSION_2_1) {
40.             mModule->set_callbacks(this);
41.         }
42.
43.         VendorTagDescriptor::clearGlobalVendorTagDescriptor();
44.
45.         if (mModule->common.module_api_version >= CAMERA_MODULE_API_VERSION_2_2) {
46.             setUpVendorTags();
47.         }
48.
49.         CameraDeviceFactory::registerService(this);
50.     }
51. }
```

第 20 行. 通过 hw_get_module 函数加载了一个 hw_module_t 模块，这个模块是与 hal 层对接的接口，ID 为 CAMERA_HARDWARE_MODULE_ID，并将它保存在 mModule 成员变量中。

第 27 行. 通过 mModule->get_number_of_cameras 函数进入到 hal 层，获取到了 camera 的个数。这个函数很重要，对于 frameworks 层来说只是拿到了 camera 的个数，但对于 hal 层和 drivers 层来说 Camera 的上电和初始化流程都是从这里开始的

3. hal 层-基于 MTK 平台

先来看看 mtk camera module 的定义，代码路径在：vendor/mediatek/proprietary/hardware/mtkcam/module_hal/module/module.h

[cpp] view plain copy

```cpp
1.  static
2.  camera_module
3.  get_camera_module()
```

```
{
    camera_module module = {
        common:{
            tag                     : HARDWARE_MODULE_TAG,
            #if (PLATFORM_SDK_VERSION >= 21)
            module_api_version      : CAMERA_MODULE_API_VERSION_2_3,
            #else
            module_api_version      : CAMERA_DEVICE_API_VERSION_1_0,
            #endif
            hal_api_version         : HARDWARE_HAL_API_VERSION,
            id                      : CAMERA_HARDWARE_MODULE_ID,
            name                    : "MediaTek Camera Module",
            author                  : "MediaTek",
            methods                 : get_module_methods(),
            dso                     : NULL,
            reserved                : {0},
        },
        get_number_of_cameras       : get_number_of_cameras,
        get_camera_info             : get_camera_info,
        set_callbacks               : set_callbacks,
        get_vendor_tag_ops          : get_vendor_tag_ops,
        #if (PLATFORM_SDK_VERSION >= 21)
        open_legacy                 : open_legacy,
        #endif
        reserved                    : {0},
    };
    return  module;
};
```

1. 保存在 frameworks 层 CameraService 的成员变量 mModule 里面的就是上面这个 module 结构体

2. 当 frameworks 层调用 mModule->get_number_of_cameras 函数时，实际就是调用上面结构体的 get_number_of_cameras 函数

[cpp] view plain copy

```cpp
CamDeviceManagerImp gCamDeviceManager;

ICamDeviceManager*
getCamDeviceManager()
{
    return &gCamDeviceManager;
}

static
int
get_number_of_cameras(void)
{
    return  NSCam::getCamDeviceManager()->getNumberOfDevices();
}
```

1. 这里先通过 getCamDeviceManager 函数获取了 CamDeviceManagerImp 对象

2. CamDeviceManagerImp 继承了 CamDeviceManagerBase，这里的 getNumberOfDevices 方法将由父类 CamDeviceManagerBase 实现

[cpp] view plain copy

```cpp
int32_t
CamDeviceManagerBase::
getNumberOfDevices()
{
    mi4DeviceNum = enumDeviceLocked();
    return  mi4DeviceNum;
}
```

这里只是调用了 enumDeviceLocked 函数，并将它的返回值（代表了 camera 的个数）返回到 frameworks 层。接着看 enumDeviceLocked 的实现

[cpp] view plain copy

```cpp
int32_t
CamDeviceManagerImp::
enumDeviceLocked()
{
    IHalSensorList*const pHalSensorList = IHalSensorList::get();
    size_t const sensorNum = pHalSensorList->searchSensors();

    for (size_t i = 0; i < sensorNum; i++)
    {
        int32_t const deviceId = i;

```

```
12.         sp<EnumInfo> pInfo = new EnumInfo;
13.         mEnumMap.add(deviceId, pInfo);
14.
15.         IMetadataProvider> pMetadataProvider = IMetadataProvider::create(deviceId);
16.         pInfo->pMetadata            = pMetadataProvider->getStaticCharacteristics();
17.         pInfo->iFacing              = (pMetadataProvider->getDeviceFacing() == MTK_LENS_FACING_FRONT)
18.                                         ? CAMERA_FACING_FRONT
19.                                         : CAMERA_FACING_BACK
20.                                         ;
21.         pInfo->iWantedOrientation   = pMetadataProvider->getDeviceWantedOrientation();
22.         pInfo->iSetupOrientation    = pMetadataProvider->getDeviceSetupOrientation();
23.         i4DeviceNum++;
24.     }
25.
26.     return i4DeviceNum;
27. }
```

第 5-6 行. 这里需要重点关注 pHalSensorList->searchSensors 函数，它的返回值就是 camera 的个数

第 8-24 行. 循环构造并初始化一个 EnumInfo 对象，并把它保存在 mEnumMap 中

[cpp] view plain copy

```
1.  MUINT
2.  HalSensorList::
3.  enumerateSensor_Locked()
4.  {
5.      int ret_count = 0;
6.      SensorDrv *const pSensorDrv = SensorDrv::get();
7.      int const iSensorsList = pSensorDrv->impSearchSensor(NULL);
8.
9.      if((iSensorsList & SENSOR_DEV_MAIN) == SENSOR_DEV_MAIN)
10.     {
11.         halSensorDev = SENSOR_DEV_MAIN;
12.         pSensorInfo = pSensorDrv->getMainSensorInfo();
13.         addAndInitSensorEnumInfo_Locked(halSensorDev, ret_count, mapToSensorType(pSensorInfo->GetType()), pSensorInfo->getDrvMacroName());
14.         ret_count++;
15.     }
16.
17.     if((iSensorsList & SENSOR_DEV_SUB) == SENSOR_DEV_SUB)
18.     {
19.         halSensorDev = SENSOR_DEV_SUB;
20.         pSensorInfo = pSensorDrv->getSubSensorInfo();
21.         addAndInitSensorEnumInfo_Locked(halSensorDev, ret_count, mapToSensorType(pSensorInfo->GetType()), pSensorInfo->getDrvMacroName());
22.         ret_count++;
23.     }
24.
25.     mEnumSensorCount = ret_count;
26.     return  ret_count;
27. }
28.
29. MUINT
30. HalSensorList::
31. searchSensors()
32. {
33.     return  enumerateSensor_Locked();
34. }
```

第 33 行. searchSensors 函数只是调用了 enumerateSensor_Locked 函数，这里并没有贴出 enumerateSensor_Locked 函数的所有代码，删减了一些我们暂时不关注的东西

第 7 行. 重点函数 pSensorDrv->impSearchSensor，它的返回值决定了 enumerateSensor_Locked 的返回值，也就是 camera 的个数

[cpp] view plain copy

```
1.  MINT32
2.  ImgSensorDrv::impSearchSensor(pfExIdChk pExIdChkCbf)
3.  {
4.      MUINT32 SensorEnum = (MUINT32) DUAL_CAMERA_MAIN_SENSOR;
5.      MUINT32 i,id[KDIMGSENSOR_MAX_INVOKE_DRIVERS] = {0,0};
6.      MINT32 sensorDevs = SENSOR_NONE;
7.
8.      GetSensorInitFuncList(&m_pstSensorInitFunc);
9.      m_fdSensor = ::open("/dev/kd_camera_hw", O_RDWR);
10.
11.     for (SensorEnum = DUAL_CAMERA_MAIN_SENSOR; SensorEnum <= DUAL_CAMERA_SUB_SENSOR; SensorEnum <<= 1) {
```

```
12.         for (i = 0; i < MAX_NUM_OF_SUPPORT_SENSOR; i++) {
13.             //end of driver list
14.             if (m_pstSensorInitFunc[i].getCameraDefault == NULL) {
15.                 LOG_MSG("m_pstSensorInitFunc[i].getCameraDefault is NULL: %d \n", i);
16.                 break;
17.             }
18.
19.             id[KDIMGSENSOR_INVOKE_DRIVER_0] = (SensorEnum << KDIMGSENSOR_DUAL_SHIFT) | i;
20.             err = ioctl(m_fdSensor, KDIMGSENSORIOC_X_SET_DRIVER,&id[KDIMGSENSOR_INVOKE_DRIVER_0] );
21.             err = ioctl(m_fdSensor, KDIMGSENSORIOC_T_CHECK_IS_ALIVE);
22.
23.             if (err < 0 || err2 < 0) {
24.                 LOG_MSG("sensor ID mismatch\n");
25.                 continue;
26.             }
27.
28.             if (SensorEnum == DUAL_CAMERA_MAIN_SENSOR) {
29.                 m_mainSensorDrv.index[m_mainSensorDrv.number] = i;
30.                 m_mainSensorDrv.type[m_mainSensorDrv.number] = sensorType;
31.                 m_mainSensorDrv.position = socketPos;
32.                 m_mainSensorDrv.sensorID = m_pstSensorInitFunc[m_mainSensorDrv.index[m_mainSensorDrv.number]].SensorId;
33.                 m_mainSensorDrv.number++;
34.             } else if (SensorEnum == DUAL_CAMERA_SUB_SENSOR) {
35.                 m_subSensorDrv.index[m_subSensorDrv.number] = i;
36.                 m_subSensorDrv.type[m_subSensorDrv.number] = sensorType;
37.                 m_subSensorDrv.position = socketPos;
38.                 m_subSensorDrv.sensorID = m_pstSensorInitFunc[m_subSensorDrv.index[m_subSensorDrv.number]].SensorId;
39.                 m_subSensorDrv.number++;
40.             }
41.         }
42.     }
43.
44.     if (BAD_SENSOR_INDEX != m_mainSensorDrv.index[0]) {
45.         m_mainSensorId = m_mainSensorDrv.sensorID;
46.         m_mainSensorIdx = m_mainSensorDrv.index[0];
47.         sensorDevs |= SENSOR_MAIN;
48.     }
49.     if (BAD_SENSOR_INDEX != m_subSensorDrv.index[0]) {
50.         m_subSensorId = m_subSensorDrv.sensorID;
51.         m_subSensorIdx = m_subSensorDrv.index[0];
52.         sensorDevs |= SENSOR_SUB;
53.     }
54.
55.     return sensorDevs;
56. }
```

这个函数比较长，所以只贴出关键代码

第 8 行，调用 GetSensorInitFuncList 函数来获取 hal 层的 sersors 列表，并把它保存在 m_pstSensorInitFunc 变量中

第 9 行，通过系统调用 open 函数打开 camera 的设备节点，后面会通过这个节点来进入到 kernel 层

第 11-12 行，通过两个 for 循环来遍历 sensorlist 中所有可能存在的 camera

第 20 行, 通过 ioctl 下达 setDriver 指令，并下传正在遍历的 sensorlist 中的 ID。Driver 层根据这个 ID，挂载 Driver 层 sensorlist 中对应的操作接口

第 21 行, 通过 ioctl 下达 check ID 指令，Driver 层为对应 sensor 上电，通过 I2C 读取预存在寄存器中的 sensor id。然后比较读取结果，如果不匹配 return error 后继续遍历

第 29-41 行，将 sensor 相关的信息保存在 m_mainSensorDrv 和 m_subSensorDrv 中

第 45-56 行，给 sensroDevs 变量赋值，并将它返回给上一级

这里暂不分析 kernel 层的代码，先来看看 GetSensorInitFuncList 函数，代码在 sensorlist.cpp 中

[cpp] view plain copy

```
1.  MSDK_SENSOR_INIT_FUNCTION_STRUCT SensorList[] =
2.  {
3.  #if defined(IMX175_MIPI_RAW)
4.      RAW_INFO(IMX175_SENSOR_ID, SENSOR_DRVNAME_IMX175_MIPI_RAW,NULL),
5.  #endif
6.  #if defined(IMX179_MIPI_RAW)
7.      RAW_INFO(IMX179_SENSOR_ID, SENSOR_DRVNAME_IMX179_MIPI_RAW,NULL),
8.  #endif
9.  #if defined(IMX219_MIPI_RAW)
```

```
10.     RAW_INFO(IMX219_SENSOR_ID, SENSOR_DRVNAME_IMX219_MIPI_RAW, NULL),
11. #endif
12. #if defined(IMX214_MIPI_RAW)
13.     RAW_INFO(IMX214_SENSOR_ID, SENSOR_DRVNAME_IMX214_MIPI_RAW,NULL),
14. #endif
15. #if defined(GC2235_RAW)
16.     RAW_INFO(GC2235_SENSOR_ID, SENSOR_DRVNAME_GC2235_RAW, NULL),
17. #endif
18. #if defined(GC2035_YUV)
19.     YUV_INFO(GC2035_SENSOR_ID, SENSOR_DRVNAME_GC2035_YUV, NULL),
20. #endif
21.     ......
22. }
23.
24. UINT32 GetSensorInitFuncList(MSDK_SENSOR_INIT_FUNCTION_STRUCT **ppSensorList)
25. {
26.     *ppSensorList = &SensorList[0];
27.     return MHAL_NO_ERROR;
28. }
```

hal 层的 sensorList，再熟悉不过的代码，需要注意的是 hal 层 sensorList 和 kernel 层的 sensorList 顺序必须保持一致

4. 总结

至此，除 kernel 层外，简述了 CameraService 的启动流程，大概过程如下图所示