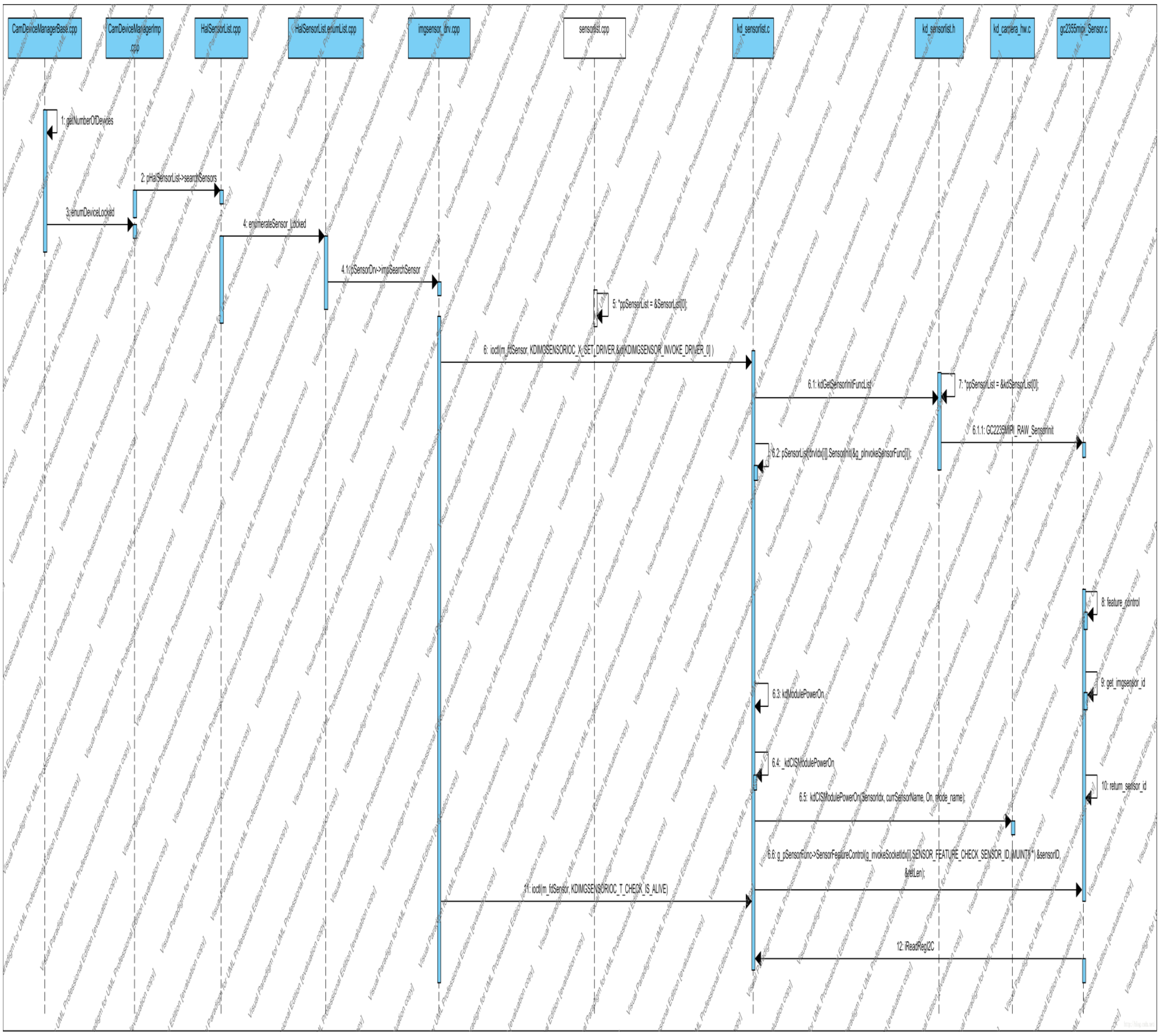
MTK6580（Android6.0)-camera 驱动分析

一、MTK6580 平台 Camera 驱动整体框架

mtk 平台三大件调试中，camera 的调试难度最大，问题也就最多，为此特地分析了一下整个 camera 驱动部分实现过程，以下为 camera 驱动框架序列图：


CamDeviceManagerBase.cpp
CamDeviceManagerImp .cpp
HalSensorList.cpp
HalSensorList.enumList.cpp
imgsensor_drv.cpp
sensorlist.cpp
kd_sensorlist.c
kd_sensorlist.h
kd_camera_hw.c
gc2355mipi_Sensor.c
1: getNumberOfDevices
2: pHalSensorList->searchSensors
3: enumDeviceLocked
4: enumerateSensor_Locked
4.1: pSensorDrv->impSearchSensor
5: *ppSensorList = &SensorList[0];
6: ioctl(m_fdSensor, KDIMGSENSORIOC_X_SET_DRIVER,&id[KDIMGSENSOR_INVOKE_DRIVER_0])
6.1: kdGetSensorInitFuncList
7: *ppSensorList = &kdSensorList[0];
6.1.1: GC2235MIPI_RAW_SensorInit
6.2: pSensorList[drvIdx[i]].SensorInit(&g_pInvokeSensorFunc[i]);
8: feature_control
9: get_imgsensor_id
6.3: kdModulePowerOn
6.4: _kdCISModulePowerOn
10: return_sensor_id
6.5: kdCISModulePowerOn(SensorIdx, currSensorName, On, mode_name);
6.6: g_pSensorFunc->SensorFeatureControl(g_invokeSocketIdx[i],SENSOR_FEATURE_CHECK_SENSOR_ID,(MUINT8*) &sensorID, &retLen);
11: ioctl(m_fdSensor, KDIMGSENSORIOC_T_CHECK_IS_ALIVE)
12: iReadRegI2C

从图中可以看出，整个框架分为三个部分 hal 部分逻辑调用，kernel 层的通用驱动 sensorlist.c 和具体 IC 的驱动 xxxx_mipi_raw.c，kernel 起来后不会直接去访问硬件 sensor,而是会注册相关的驱动，之后 **Android** 系统起来后会启动相关的服务如：camera_service，在 camera 服务中会直接去访问 hal,kernel 驱动，进而操作 camera。

为此本文也穿插了部分 hal 层的调用,至于 camera_service 后面章节会继续补充。

二、 Camera 驱动的具体实现

=========================HAL 层部分初始调用=========================

文件：vendor/mediatek/proprietary/hardware/mtkcam/common/module_hal/devicemgr/CamDeviceManagerBase.cpp

[cpp] view plain copy

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

        ...
}
```

文件：vendor/mediatek/proprietary/hardware/mtkcam/legacy/platform/mt6580/devicemgr/CamDeviceManagerImp.cpp

[cpp] view plain copy

```cpp
int32_t
CamDeviceManagerImp::
enumDeviceLocked()
{
    ...
//------------------------------------------------------------------------------
#if '1'==MTKCAM_HAVE_SENSOR_HAL
    //
    IHalSensorList*const pHalSensorList = IHalSensorList::get();
    size_t const sensorNum = pHalSensorList->searchSensors();
#endif
    ...
    return  i4DeviceNum;
}
```

文件：vendor/mediatek/proprietary/hardware/mtkcam/legacy/platform/mt6580/hal/sensor/HalSensorList.cpp

[cpp] view plain copy

```cpp
MUINT
HalSensorList::
searchSensors()
{
    Mutex::Autolock _l(mEnumSensorMutex);
    //
    MY_LOGD("searchSensors");
    return  enumerateSensor_Locked();
}
```

文件：vendor/mediatek/proprietary/hardware/mtkcam/legacy/platform/mt6580/hal/sensor/HalSensorList.enumList.cpp

[cpp] view plain copy

```cpp
MUINT
HalSensorList::
enumerateSensor_Locked()
{
    ....


    MUINT halSensorDev = SENSOR_DEV_NONE;
    NSFeature::SensorInfoBase* pSensorInfo ;


    SensorDrv *const pSensorDrv = SensorDrv::get();
    SeninfDrv *const pSeninfDrv = SeninfDrv::createInstance();

    int const iSensorsList = pSensorDrv->impSearchSensor(NULL);


    ....
}
```

文件：vendor/mediatek/proprietary/hardware/mtkcam/legacy/platform/mt6580/hal/sensor/imgsensor_drv.cpp

[cpp] view plain copy

```cpp
MINT32
ImgSensorDrv::impSearchSensor(pfExIdChk pExIdChkCbf)
{
    ....


    GetSensorInitFuncList(&m_pstSensorInitFunc);


    LOG_MSG("SENSOR search start \n");


    sprintf(cBuf,"/dev/%s",CAMERA_HW_DEVNAME);
    m_fdSensor = ::open(cBuf, O_RDWR);


    ......

        for (i = 0; i < MAX_NUM_OF_SUPPORT_SENSOR; i++) {
            ....
            err = ioctl(m_fdSensor, KDIMGSENSORIOC_X_SET_DRIVER,&id[KDIMGSENSOR_INVOKE_DRIVER_0] );
            ...
            err = ioctl(m_fdSensor, KDIMGSENSORIOC_T_CHECK_IS_ALIVE);


    ......
}
```

GetSensorInitFuncList 的实现

文件：vendor/mediatek/proprietary/custom/mt6580/hal/imgsensor_src/sensorlist.cpp

[cpp] view plain copy

```
UINT32 GetSensorInitFuncList(MSDK_SENSOR_INIT_FUNCTION_STRUCT **ppSensorList)
{
    if (NULL == ppSensorList) {
        ALOGE("ERROR: NULL pSensorList\n");
        return MHAL_UNKNOWN_ERROR;
    }
    *ppSensorList = &SensorList[0];
    return MHAL_NO_ERROR;
}
```

Sensor 列表的定义如下：

[cpp] view plain copy

```
MSDK_SENSOR_INIT_FUNCTION_STRUCT SensorList[] =
{
//xc add camera start
#if defined(GC2365MIPI_RAW)
    RAW_INFO(GC2365MIPI_SENSOR_ID, SENSOR_DRVNAME_GC2365MIPI_RAW, NULL),
#endif


#if defined(GC2355_MIPI_RAW_BAIKANG_M8112)
    RAW_INFO(GC2355_SENSOR_ID, SENSOR_DRVNAME_GC2355_MIPI_RAW,NULL),
#endif
....
}
```

获取 sensor 列表后，紧接着通过：

err = ioctl(m_fdSensor, KDIMGSENSORIOC_X_SET_DRIVER,&id[KDIMGSENSOR_INVOKE_DRIVER_0]);
err = ioctl(m_fdSensor, KDIMGSENSORIOC_T_CHECK_IS_ALIVE);

访问 kernel 层的数据

======================= Kernel 层驱动的实现 =========================

1. 针对前后摄注册 platform 设备和驱动

文件：kernel-3.18/drivers/misc/mediatek/imgsensor/src/mt6580/kd_sensorlist.c

[cpp] view plain copy

```
static int __init CAMERA_HW_i2C_init(void)
{
    ....



    if (platform_driver_register(&g_stCAMERA_HW_Driver)) //注册主摄 platform 驱动
    if (platform_driver_register(&g_stCAMERA_HW_Driver2)) //注册副摄 platform 驱动


    ....
    return 0;
}
```

主摄平台驱动的定义：

[cpp] view plain copy

```
#ifdef CONFIG_OF
```

```
static const struct of_device_id CAMERA_HW_of_ids[] = {
    {.compatible = "mediatek,camera_hw",},  //主摄匹配规则
    {}
};
#endif


static struct platform_driver g_stCAMERA_HW_Driver = {
    .probe = CAMERA_HW_probe,
    .remove = CAMERA_HW_remove,
    .suspend = CAMERA_HW_suspend,
    .resume = CAMERA_HW_resume,
    .driver = {
           .name = "image_sensor",
           .owner = THIS_MODULE,
#ifdef CONFIG_OF
           .of_match_table = CAMERA_HW_of_ids,
#endif
           }
};
```

副摄平台驱动的定义：

[cpp] view plain copy

```
#ifdef CONFIG_OF
static const struct of_device_id CAMERA_HW2_of_ids[] = {
    {.compatible = "mediatek,camera_hw2",},//副摄匹配规则
    {}
};
#endif


static struct platform_driver g_stCAMERA_HW_Driver2 = {
    .probe = CAMERA_HW_probe2,
    .remove = CAMERA_HW_remove2,
    .suspend = CAMERA_HW_suspend2,
    .resume = CAMERA_HW_resume2,
    .driver = {
           .name = "image_sensor_bus2",
           .owner = THIS_MODULE,
#ifdef CONFIG_OF
           .of_match_table = CAMERA_HW2_of_ids,
#endif


    }
};
```

主副摄 cam 在 dts 中定义设备信息：

[cpp] view plain copy

```
kd_camera_hw1:kd_camera_hw1@15008000 {
    compatible = "mediatek,camera_hw"; //这里必须和主摄一致
    reg = <0x15008000 0x1000>;  /* SENINF_ADDR */
    vcama-supply = <&mt_pmic_vcama_ldo_reg>;
    vcamd-supply = <&mt_pmic_vcamd_ldo_reg>;
    vcamaf-supply = <&mt_pmic_vcamaf_ldo_reg>;
    vcamio-supply = <&mt_pmic_vcamio_ldo_reg>;


};
kd_camera_hw2:kd_camera_hw2@15008000 {
    compatible = "mediatek,camera_hw2"; //这里必须和副摄一致
    reg = <0x15008000 0x1000>;  /* SENINF_ADDR */
};
```

当内核启动后，会解析 dts 编译生成的 dtb 文件，注册里面定义的 device,如果和驱动中定义 id 一致，则挂载启动。

上面注册了两个 platform 驱动 g_stCAMERA_HW_Driver，g_stCAMERA_HW_Driver2，如果匹配成功会调用各自的

probe 函数 CAMERA_HW_probe，CAMERA_HW_probe2

2. 平台 probe 函数的实现

主摄 probe，CAMERA_HW_probe 的实现如下：

[cpp] view plain copy

```cpp
static int CAMERA_HW_probe(struct platform_device *pdev)
{
#if !defined(CONFIG_MTK_LEGACY)
    mtkcam_gpio_init(pdev);
    mtkcam_pin_mux_init(pdev);
#endif
    return i2c_add_driver(&CAMERA_HW_i2c_driver);
}
```

副摄 probe，CAMERA_HW_probe 的实现如下：

[cpp] view plain copy

```cpp
static int CAMERA_HW_probe2(struct platform_device *pdev)
{
    return i2c_add_driver(&CAMERA_HW_i2c_driver2);
}
```

从上可以看出在 main/sub 的平台 probe 中分别注册了各自的 i2c 驱动 CAMERA_HW_i2c_driver，CAMERA_HW_i2c_driver2，main sensor 的 CAMERA_HW_i2c_driver 定义如下：

[cpp] view plain copy

```cpp
#ifdef CONFIG_OF
static const struct of_device_id CAMERA_HW_i2c_of_ids[] = {
    { .compatible = "mediatek,camera_main", },
    {}
};
#endif


struct i2c_driver CAMERA_HW_i2c_driver = {
    .probe = CAMERA_HW_i2c_probe,
    .remove = CAMERA_HW_i2c_remove,
    .driver = {
           .name = CAMERA_HW_DRVNAME1,
           .owner = THIS_MODULE,


#ifdef CONFIG_OF
           .of_match_table = CAMERA_HW_i2c_of_ids,
#endif
           },
    .id_table = CAMERA_HW_i2c_id,
};
sub sensor 的 CAMERA_HW_i2c_driver 定义如下:
#ifdef CONFIG_OF
    static const struct of_device_id CAMERA_HW2_i2c_driver_of_ids[] = {
    { .compatible = "mediatek,camera_sub", },
    {}
    };
#endif


struct i2c_driver CAMERA_HW_i2c_driver2 = {
    .probe = CAMERA_HW_i2c_probe2,
    .remove = CAMERA_HW_i2c_remove2,
    .driver = {
    .name = CAMERA_HW_DRVNAME2,
    .owner = THIS_MODULE,
#ifdef CONFIG_OF
    .of_match_table = CAMERA_HW2_i2c_driver_of_ids,
#endif
           },
    .id_table = CAMERA_HW_i2c_id2,
};
```

对应 main/sub camera i2c 设备 dts 定义如下

文件：kernel-3.18/arch/arm/boot/dts/cust_i2c.dtsi

[cpp] view plain copy

```
&i2c0 {
    camera_main@10 {
        compatible = "mediatek,camera_main"; //和 CAMERA_HW_i2c_driver 定义的一致
        reg = <0x10>;
    };


    camera_main_af@0c {
        compatible = "mediatek,camera_main_af";
        reg = <0x0c>;
    };


    camera_sub@3c {
        compatible = "mediatek,camera_sub"; //和 CAMERA_HW_i2c_driver2 定义的一致
        reg = <0x3c>;
    };


};
```

3. I2c probe 的实现

从上可以看出 main/sub sensor 在各自的平台 probe 中，注册了 i2c_driver,当各自的 i2c_driver 和设备匹配（如何匹配本章不作分析）成功后，会调用各自的 i2c_probe 函数。main sensor 的 probe 函数 CAMERA_HW_i2c_probe：

[cpp] view plain copy

```
static int CAMERA_HW_i2c_probe(struct i2c_client *client, const struct i2c_device_id *id)
{
    .....
    /* Register char driver */
    i4RetValue = RegisterCAMERA_HWCharDrv();


    .....
    return 0;
}
```

sub sensor 的 probe 函数 CAMERA_HW_i2c_probe2：

[cpp] view plain copy

```
static int CAMERA_HW_i2c_probe2(struct i2c_client *client, const struct i2c_device_id *id)
{
    .....


    /* Register char driver */
    i4RetValue = RegisterCAMERA_HWCharDrv2();


    .....
}
```

从上可以看出 main/sub 在各自的 i2cprobe 中，通过该调用 RegisterCAMERA_HWCharDrv，RegisterCAMERA_HWCharDrv2 注册了字符设备。各自注册 cdev 函数实现如下：

[cpp] view plain copy

```
static inline int RegisterCAMERA_HWCharDrv(void)//main sensor 注册 cdev
{


    .....
    /* Attatch file operation. */
    cdev_init(g_pCAMERA_HW_CharDrv, &g_stCAMERA_HW_fops); //初始化字符设备

    /* Add to system */
    cdev_add(g_pCAMERA_HW_CharDrv, g_CAMERA_HWdevno, 1) //注册到内核


    //创建目录 /sys/class/sensordrv/
    sensor_class = class_create(THIS_MODULE, "sensordrv");
    //创建目录/sys/class/sensordrv/kd_camera_hw
    sensor_device = device_create(sensor_class, NULL, g_CAMERA_HWdevno, NULL, CAMERA_HW_DRVNAME1);


    ....
    return 0;
}
static inline int RegisterCAMERA_HWCharDrv2(void)//sub sensor 注册 cdev
{
    ....


    /* Attatch file operation. */
    cdev_init(g_pCAMERA_HW_CharDrv2, &g_stCAMERA_HW_fops0);//初始化字符设备
    /* Add to system */
    cdev_add(g_pCAMERA_HW_CharDrv2, g_CAMERA_HWdevno2, 1));//注册到内核
    //创建目录 /sys/class/sensordrv2/
    sensor2_class = class_create(THIS_MODULE, "sensordrv2");
    //创建目录/sys/class/sensordrv2/kd_camera_hw_bus2
    sensor_device2 = device_create(sensor2_class, NULL, g_CAMERA_HWdevno2, NULL, CAMERA_HW_DRVNAME2);
    ....
    return 0;
}
main/sub 创建各自的字符设备过程中绑定各自的 fops,g_stCAMERA_HW_fops 和 g_stCAMERA_HW_fops0
他们各自定义如下
static const struct file_operations g_stCAMERA_HW_fops = { //main sensor fops
    .owner = THIS_MODULE,
    .open = CAMERA_HW_Open,
    .release = CAMERA_HW_Release,
    .unlocked_ioctl = CAMERA_HW_Ioctl,
#ifdef CONFIG_COMPAT
    .compat_ioctl = CAMERA_HW_Ioctl_Compat,
#endif


};


static const struct file_operations g_stCAMERA_HW_fops0 = { //sub sensor fops
    .owner = THIS_MODULE,
    .open = CAMERA_HW_Open2,
    .release = CAMERA_HW_Release2,
    .unlocked_ioctl = CAMERA_HW_Ioctl,
#ifdef CONFIG_COMPAT
    .compat_ioctl = CAMERA_HW_Ioctl_Compat,
#endif


};
```

从上可以看出各自的 fops 指定了相同的 Iioctl 函数，意味着上层操作 main/sub sensor 只需要对应一个底层的 ioctl 即可，至于 sensor 的区分可以借助 idx,后面会讲到

[cpp] view plain copy

```
/*******************************************************************************
* CAMERA_HW_Ioctl
********************************************************************************/


static long CAMERA_HW_Ioctl(struct file *a_pstFile,
                unsigned int a_u4Command, unsigned long a_u4Param)
{


    ...
    pIdx = (u32 *) pBuff;
    switch (a_u4Command) {
    ...

```

```cpp

    case KDIMGSENSORIOC_X_SET_DRIVER:
        i4RetValue = kdSetDriver((unsigned int *)pBuff);
        break;


    case KDIMGSENSORIOC_X_FEATURECONCTROL:
        i4RetValue = adopt_CAMERA_HW_FeatureControl(pBuff);
        break;


    case KDIMGSENSORIOC_T_CHECK_IS_ALIVE:
        i4RetValue = adopt_CAMERA_HW_CheckIsAlive();
        break;

    ....
    default:
        PK_DBG("No such command\n");
        i4RetValue = -EPERM;
        break;


    }


    .....
}
```

这里 ioctl 和上层一一对应，上层要控制 caemra 只需要传入相应的 cmd 和 data 而已

================================ HAL 调用 Kernel 层驱动的逻辑 =========================

前面介绍了 HAL 层调用 ioctl 和 kernel 层注册驱动，接下来继续分析，HAL 层调用后驱动具体的实现流程。

4. ioctl 底层的实现

4.1 先来看 ioctl(m_fdSensor, KDIMGSENSORIOC_X_SET_DRIVER,&id[KDIMGSENSOR_INVOKE_DRIVER_0]);

当 KDIMGSENSORIOC_X_SET_DRIVER 被传下时，会调用 kernel 层的 kdSetDriver 接口

[cpp] view plain copy

```cpp
int kdSetDriver(unsigned int *pDrvIndex)
{
    ...


    kdGetSensorInitFuncList(&pSensorList))   //获得 sensor 初始化列表


    for (i = KDIMGSENSOR_INVOKE_DRIVER_0; i < KDIMGSENSOR_MAX_INVOKE_DRIVERS; i++) {
    ....
        pSensorList[drvIdx[i]].SensorInit(&g_pInvokeSensorFunc[i]); //获取各个 cam 驱动中 Init 函数入口


    ....
    }
    return 0;
}
```

kdGetSensorInitFuncList 的实现：

[cpp] view plain copy

```cpp
UINT32 kdGetSensorInitFuncList(ACDK_KD_SENSOR_INIT_FUNCTION_STRUCT **ppSensorList)
{
    if (NULL == ppSensorList) {
        PK_ERR("[kdGetSensorInitFuncList]ERROR: NULL ppSensorList\n");
        return 1;
    }
    *ppSensorList = &kdSensorList[0];  //获取 sensorlist 数组首地址
    return 0;
}               /* kdGetSensorInitFuncList() */
```

kdSensorList 定义如下：

文件：kernel-3.18/drivers/misc/mediatek/imgsensor/src/mt6580/kd_sensorlist.h

[cpp] view plain copy

```cpp
ACDK_KD_SENSOR_INIT_FUNCTION_STRUCT kdSensorList[MAX_NUM_OF_SUPPORT_SENSOR+1] =
{
    ....


#if defined(SUB_GC2355_MIPI_RAW)
    {GC2355S_SENSOR_ID, SENSOR_DRVNAME_GC2355S_MIPI_RAW,Sub_GC2355_MIPI_RAW_SensorInit},
#endif


    ....
}
```

获取列表之后紧接着调用各自的 Init 函数,这里以 GC2355 为例

[cpp] view plain copy

```cpp
UINT32 GC2235MIPI_RAW_SensorInit(PSENSOR_FUNCTION_STRUCT *pfFunc)
{
    /* To Do : Check Sensor status here */
    if (pfFunc!=NULL)
        *pfFunc=&sensor_func;
    return ERROR_NONE;
}   /*  GC2235MIPI_RAW_SensorInit   */
```

从中可以看出，gc2355 的 Init 函数地址传给了 pfFunc，也就是时候，后面在通用驱动可以直接凭借 pfun 指针调用 sensorlist 中的驱动

4.2 再来看 ioctl(m_fdSensor, KDIMGSENSORIOC_T_CHECK_IS_ALIVE);

当 KDIMGSENSORIOC_T_CHECK_IS_ALIVE 被传下时，会调用 kernel 层的 adopt_CAMERA_HW_Feature Control 接口

[cpp] view plain copy

```cpp
static inline int adopt_CAMERA_HW_CheckIsAlive(void)
{
    ....
    /* power on sensor */
    kdModulePowerOn((CAMERA_DUAL_CAMERA_SENSOR_ENUM *) g_invokeSocketIdx, g_invokeSensorNameStr,
            true, CAMERA_HW_DRVNAME1);


    ....

    if (g_pSensorFunc) {
        for (i = KDIMGSENSOR_INVOKE_DRIVER_0; i < KDIMGSENSOR_MAX_INVOKE_DRIVERS; i++) {
            if (DUAL_CAMERA_NONE_SENSOR != g_invokeSocketIdx[i]) {
                err =
                    g_pSensorFunc->SensorFeatureControl(g_invokeSocketIdx[i],
                                    SENSOR_FEATURE_CHECK_SENSOR_ID,
                                    (MUINT8 *) &sensorID,
                                    &retLen);
                if (sensorID == 0) {    /* not implement this feature ID */
                    PK_DBG
                        (" Not implement!!, use old open function to check\n");
                    err = ERROR_SENSOR_CONNECT_FAIL;
                } else if (sensorID == 0xFFFFFFFF) {    /* fail to open the sensor */
                    PK_DBG(" No Sensor Found");
                    err = ERROR_SENSOR_CONNECT_FAIL;
                } else {


                    PK_INF(" Sensor found ID = 0x%x\n", sensorID);
                    snprintf(mtk_ccm_name, sizeof(mtk_ccm_name),
                         "%s CAM[%d]:%s;", mtk_ccm_name,
                         g_invokeSocketIdx[i], g_invokeSensorNameStr[i]);
                    psensorResolution[0] = &sensorResolution[0];
                    psensorResolution[1] = &sensorResolution[1];
                    // don't care of the result
                    g_pSensorFunc->SensorGetResolution(psensorResolution);
                    if(g_invokeSocketIdx[i] == DUAL_CAMERA_MAIN_SENSOR)
                        curr_sensor_id = 0;
                    else if(g_invokeSocketIdx[i] == DUAL_CAMERA_SUB_SENSOR)
                        curr_sensor_id = 1;
                    /* fill the cam infos with name/width/height */
                    snprintf(g_cam_infos, sizeof(g_cam_infos),"%s CAM[%d]:%s,Width:%d, Height:%d;",
                                g_cam_infos, g_invokeSocketIdx[i], g_invokeSensorNameStr[i],
                                sensorResolution[curr_sensor_id].SensorFullWidth, sensorResolution[curr_sensor_id].SensorFullHeight);


                    err = ERROR_NONE;
                }
                if (ERROR_NONE != err) {
                    PK_DBG
                        ("ERROR:adopt_CAMERA_HW_CheckIsAlive(), No imgsensor alive\n");
                }
            }
        }
    } else {
        PK_DBG("ERROR:NULL g_pSensorFunc\n");
    }
}               /* adopt_CAMERA_HW_Open() */
```

这个函数非常重要，它主要进行了以下几个动作，

1）通过 kdModulePowerOn 给 Sensor 上电

2）通过 SensorFeatureControl 读取 SensorID

先看 kdModulePowerOn 的实现

[cpp] view plain copy

```
int
kdModulePowerOn(CAMERA_DUAL_CAMERA_SENSOR_ENUM socketIdx[KDIMGSENSOR_MAX_INVOKE_DRIVERS],
        char sensorNameStr[KDIMGSENSOR_MAX_INVOKE_DRIVERS][32], BOOL On, char *mode_name)
{
    MINT32 ret = ERROR_NONE;
    u32 i = 0;


    for (i = KDIMGSENSOR_INVOKE_DRIVER_0; i < KDIMGSENSOR_MAX_INVOKE_DRIVERS; i++) {
        if (g_bEnableDriver[i]) {
            /* PK_XLOG_INFO("[%s][%d][%d][%s][%s]\r\n",__FUNCTION__,g_bEnableDriver[i],socketIdx[i],sensorNameStr[i],mode_name); */
#ifndef CONFIG_FPGA_EARLY_PORTING
            ret = _kdCISModulePowerOn(socketIdx[i], sensorNameStr[i], On, mode_name);
#endif
            if (ERROR_NONE != ret) {
                PK_ERR("[%s]", __func__);
                return ret;
            }
        }
    }
    return ERROR_NONE;
}
```

在 kdModulePowerOn 中又调用_kdCISModulePowerOn

[cpp] view plain copy

```
int _kdCISModulePowerOn(CAMERA_DUAL_CAMERA_SENSOR_ENUM SensorIdx, char *currSensorName, BOOL On,
            char *mode_name)
{
    ....


    ret = kdCISModulePowerOn(SensorIdx, currSensorName, On, mode_name);
    ....
    return ret;
}
```

在_kdCISModulePowerOn 又调用 kdCISModulePowerOn 函数

文件：kernel-3.18/drivers/misc/mediatek/imgsensor/src/mt6580/camera_hw/kd_camera_hw.c

//改函数为上下电函数，通过传入 BOOL 值来判断上/下电

[cpp] view plain copy

```cpp
int kdCISModulePowerOn(CAMERA_DUAL_CAMERA_SENSOR_ENUM SensorIdx, char *currSensorName, BOOL On,
               char *mode_name)
{


    u32 pinSetIdx = 0;  /* default main sensor */


#define IDX_PS_CMRST 0
#define IDX_PS_CMPDN 4
#define IDX_PS_MODE 1
#define IDX_PS_ON   2
#define IDX_PS_OFF  3
#define VOL_2800 2800000
#define VOL_1800 1800000
#define VOL_1500 1500000
#define VOL_1200 1200000
#define VOL_1000 1000000




    u32 pinSet[3][8] = {
        /* for main sensor */
        {       /* The reset pin of main sensor uses GPIO10 of mt6306, please call mt6306 API to set */
         CAMERA_CMRST_PIN,
         CAMERA_CMRST_PIN_M_GPIO,   /* mode */
         GPIO_OUT_ONE,  /* ON state */
         GPIO_OUT_ZERO, /* OFF state */
         CAMERA_CMPDN_PIN,
         CAMERA_CMPDN_PIN_M_GPIO,
         GPIO_OUT_ONE,
         GPIO_OUT_ZERO,
         },
        /* for sub sensor */
        {
         CAMERA_CMRST1_PIN,
         CAMERA_CMRST1_PIN_M_GPIO,
         GPIO_OUT_ONE,
         GPIO_OUT_ZERO,
         CAMERA_CMPDN1_PIN,
         CAMERA_CMPDN1_PIN_M_GPIO,
         GPIO_OUT_ONE,
         GPIO_OUT_ZERO,
         },
        /* for main_2 sensor */
        {
         GPIO_CAMERA_INVALID,
         GPIO_CAMERA_INVALID,   /* mode */
         GPIO_OUT_ONE,  /* ON state */
         GPIO_OUT_ZERO, /* OFF state */
         GPIO_CAMERA_INVALID,
         GPIO_CAMERA_INVALID,
         GPIO_OUT_ONE,
         GPIO_OUT_ZERO,
         }
    };


    if (DUAL_CAMERA_MAIN_SENSOR == SensorIdx)
        pinSetIdx = 0;
    else if (DUAL_CAMERA_SUB_SENSOR == SensorIdx)
        pinSetIdx = 1;
    else if (DUAL_CAMERA_MAIN_2_SENSOR == SensorIdx)
```

```
        pinSetIdx = 2;


    /* power ON */
    if (On) {


#if 0
        ISP_MCLK1_EN(1);
        ISP_MCLK2_EN(1);
        ISP_MCLK3_EN(1);
#else
        if (pinSetIdx == 0)
            ISP_MCLK1_EN(1);
        else if (pinSetIdx == 1)
            ISP_MCLK2_EN(1);
#endif


    printk("fangkuiccm %d ,currSensorName = %s pinSetIdx = %d ",__LINE__,currSensorName,pinSetIdx );

    //通过 DriverName 来区分 SensorIC
    if (currSensorName && (0 == strcmp(SENSOR_DRVNAME_GC2355_MIPI_RAW, currSensorName))) {

            /* First Power Pin low and Reset Pin Low */
            if (GPIO_CAMERA_INVALID != pinSet[pinSetIdx][IDX_PS_CMPDN])
                mtkcam_gpio_set(pinSetIdx, CAMPDN,
                        pinSet[pinSetIdx][IDX_PS_CMPDN + IDX_PS_OFF]);


            if (GPIO_CAMERA_INVALID != pinSet[pinSetIdx][IDX_PS_CMRST])
                mtkcam_gpio_set(pinSetIdx, CAMRST,
                        pinSet[pinSetIdx][IDX_PS_CMRST + IDX_PS_OFF]);


            mdelay(50);


            /* VCAM_A */
            if (TRUE != _hwPowerOn(VCAMA, VOL_2800)) {
                PK_DBG
                    ("[CAMERA SENSOR] Fail to enable analog power (VCAM_A),power id = %d\n",
                     VCAMA);
                goto _kdCISModulePowerOn_exit_;
            }


            mdelay(10);


            /* VCAM_IO */
            if (TRUE != _hwPowerOn(VCAMIO, VOL_1800)) {
                PK_DBG
                    ("[CAMERA SENSOR] Fail to enable IO power (VCAM_IO),power id = %d\n",
                     VCAMIO);
                goto _kdCISModulePowerOn_exit_;
            }


            mdelay(10);


            if (TRUE != _hwPowerOn(VCAMD, VOL_1500)) {
                PK_DBG
                    ("[CAMERA SENSOR] Fail to enable digital power (VCAM_D),power id = %d\n",
                     VCAMD);
                goto _kdCISModulePowerOn_exit_;
            }


            mdelay(10);


            /* AF_VCC */
            if (TRUE != _hwPowerOn(VCAMAF, VOL_2800)) {
                PK_DBG
                    ("[CAMERA SENSOR] Fail to enable analog power (VCAM_AF),power id = %d\n",
                     VCAMAF);
                goto _kdCISModulePowerOn_exit_;
            }




            mdelay(50);


            if (GPIO_CAMERA_INVALID != pinSet[pinSetIdx][IDX_PS_CMRST]) {
                mtkcam_gpio_set(pinSetIdx, CAMRST,
                        pinSet[pinSetIdx][IDX_PS_CMRST + IDX_PS_OFF]);
                mdelay(5);
                mtkcam_gpio_set(pinSetIdx, CAMRST,
                        pinSet[pinSetIdx][IDX_PS_CMRST + IDX_PS_ON]);
            }
            mdelay(5);
            /* enable active sensor */
            if (GPIO_CAMERA_INVALID != pinSet[pinSetIdx][IDX_PS_CMPDN]) {
                mtkcam_gpio_set(pinSetIdx, CAMPDN,
                        pinSet[pinSetIdx][IDX_PS_CMPDN + IDX_PS_ON]);
                mdelay(5);
                mtkcam_gpio_set(pinSetIdx, CAMPDN,
                        pinSet[pinSetIdx][IDX_PS_CMPDN + IDX_PS_OFF]);
            }


            mdelay(5);
        }
    }else{ //poweroff
     if (currSensorName   //上完电就要下电不然会造成漏电，最终会影响手机功耗
               && (0 == strcmp(SENSOR_DRVNAME_GC2355_MIPI_RAW, currSensorName))) {
#if 0
            mt_set_gpio_mode(GPIO_SPI_MOSI_PIN, GPIO_MODE_00);
            mt_set_gpio_dir(GPIO_SPI_MOSI_PIN, GPIO_DIR_OUT);
            mt_set_gpio_out(GPIO_SPI_MOSI_PIN, GPIO_OUT_ONE);
#endif
            /* First Power Pin low and Reset Pin Low */
            if (GPIO_CAMERA_INVALID != pinSet[pinSetIdx][IDX_PS_CMPDN]) {
                if (mt_set_gpio_mode
                    (pinSet[pinSetIdx][IDX_PS_CMPDN],
                     pinSet[pinSetIdx][IDX_PS_CMPDN + IDX_PS_MODE])) {
                    PK_DBG("[CAMERA LENS] set gpio mode failed!! (CMPDN)\n");
                }
                if (mt_set_gpio_dir(pinSet[pinSetIdx][IDX_PS_CMPDN], GPIO_DIR_OUT)) {
                    PK_DBG("[CAMERA LENS] set gpio dir failed!! (CMPDN)\n");
                }
                if (mt_set_gpio_out
                    (pinSet[pinSetIdx][IDX_PS_CMPDN],
                     pinSet[pinSetIdx][IDX_PS_CMPDN + IDX_PS_OFF])) {
                    PK_DBG("[CAMERA LENS] set gpio failed!! (CMPDN)\n");
                }
            }
    }
}
```

上电操作完成后，紧接着读取 SensorID，通用驱动使用 SensorFeatureControl 来读取 ID 如：

```
 g_pSensorFunc->SensorFeatureControl(g_invokeSocketIdx[i],
SENSOR_FEATURE_CHECK_SENSOR_ID,
(MUINT8 *) &sensorID,
&retLen);
```

这步操作会调用 GC2355 中的 feature_control 函数如下：

文件：kernel-3.18/drivers/misc/mediatek/imgsensor/src/mt6580/gc2355_mipi_raw/gc2355mipi_Sensor.c

[cpp] view plain copy

```cpp
static kal_uint32 feature_control(MSDK_SENSOR_FEATURE_ENUM feature_id,
                             UINT8 *feature_para,UINT32 *feature_para_len)
{
    ....
    LOG_INF("feature_id = %d\n", feature_id);
    switch (feature_id) {
        ....
        case SENSOR_FEATURE_CHECK_SENSOR_ID:
            get_imgsensor_id(feature_return_para_32);
            break;
        ....
        default:
            break;
    }


    return ERROR_NONE;
}
```

优化传入的 cmd 为 SENSOR_FEATURE_CHECK_SENSOR_ID，则会调用 feature_control 中的 get_imgsensor_id 再看 get_imgsensor_id 的实现

[cpp] view plain copy

```cpp
static kal_uint32 get_imgsensor_id(UINT32 *sensor_id)
{
    kal_uint8 i = 0;
    kal_uint8 retry = 2;
    //sensor have two i2c address 0x6c 0x6d & 0x21 0x20, we should detect the module used i2c address
    while (imgsensor_info.i2c_addr_table[i] != 0xff) {
        spin_lock(&imgsensor_drv_lock);
        imgsensor.i2c_write_id = imgsensor_info.i2c_addr_table[i];
        spin_unlock(&imgsensor_drv_lock);
        do {
            *sensor_id = return_sensor_id(); //return_sensor_id 读取 IC 的 ID
            if (*sensor_id == imgsensor_info.sensor_id) {
                LOG_INF("i2c write id: 0x%x, sensor id: 0x%x\n", imgsensor.i2c_write_id,*sensor_id);
                return ERROR_NONE;
            }
            LOG_INF("Read sensor id fail, write id: 0x%x, id: 0x%x\n", imgsensor.i2c_write_id,*sensor_id);
            retry--;
        } while(retry > 0);
        i++;
        retry = 2;
    }
    ....
    return ERROR_NONE;
}
```

再看 return_sensor_id 的实现

[cpp] view plain copy

```cpp
static kal_uint32 return_sensor_id(void)
{
    return ((read_cmos_sensor(0xf0) << 8) | read_cmos_sensor(0xf1));
}
static kal_uint16 read_cmos_sensor(kal_uint32 addr)
{
    kal_uint16 get_byte=0;


    char pu_send_cmd[1] = {(char)(addr & 0xFF) };
    iReadRegI2C(pu_send_cmd, 1, (u8*)&get_byte, 1, imgsensor.i2c_write_id);


    return get_byte;


}
```

文件：kernel-3.18/drivers/misc/mediatek/imgsensor/src/mt6580/kd_sensorlist.c

[cpp] view plain copy

```cpp
int iReadRegI2C(u8 *a_pSendData, u16 a_sizeSendData, u8 *a_pRecvData, u16 a_sizeRecvData,
        u16 i2cId)
{
    int i4RetValue = 0;


    if (gI2CBusNum == SUPPORT_I2C_BUS_NUM1) {
        spin_lock(&kdsensor_drv_lock);
        g_pstI2Cclient->addr = (i2cId >> 1);
        g_pstI2Cclient->ext_flag = (g_pstI2Cclient->ext_flag) & (~I2C_DMA_FLAG);


        /* Remove i2c ack error log during search sensor */
        /* PK_ERR("g_pstI2Cclient->ext_flag: %d", g_IsSearchSensor); */
        if (g_IsSearchSensor == 1) {
            g_pstI2Cclient->ext_flag = (g_pstI2Cclient->ext_flag) | I2C_A_FILTER_MSG;
        } else {
            g_pstI2Cclient->ext_flag = (g_pstI2Cclient->ext_flag) & (~I2C_A_FILTER_MSG);
        }


        spin_unlock(&kdsensor_drv_lock);
        /*  */
        i4RetValue = i2c_master_send(g_pstI2Cclient, a_pSendData, a_sizeSendData);
        if (i4RetValue != a_sizeSendData) {
            PK_ERR("[CAMERA SENSOR] I2C send failed!!, Addr = 0x%x\n", a_pSendData[0]);
            return -1;
        }


        i4RetValue = i2c_master_recv(g_pstI2Cclient, (char *)a_pRecvData, a_sizeRecvData);
        if (i4RetValue != a_sizeRecvData) {
            PK_ERR("[CAMERA SENSOR] I2C read failed!!\n");
            return -1;
        }
    } else {
        spin_lock(&kdsensor_drv_lock);
        g_pstI2Cclient2->addr = (i2cId >> 1);


        /* Remove i2c ack error log during search sensor */
        /* PK_ERR("g_pstI2Cclient2->ext_flag: %d", g_IsSearchSensor); */
        if (g_IsSearchSensor == 1) {
            g_pstI2Cclient2->ext_flag = (g_pstI2Cclient2->ext_flag) | I2C_A_FILTER_MSG;
        } else {
            g_pstI2Cclient2->ext_flag = (g_pstI2Cclient2->ext_flag) & (~I2C_A_FILTER_MSG);
        }
        spin_unlock(&kdsensor_drv_lock);
        i4RetValue = i2c_master_send(g_pstI2Cclient2, a_pSendData, a_sizeSendData);
        if (i4RetValue != a_sizeSendData) {
            PK_ERR("[CAMERA SENSOR] I2C send failed!!, Addr = 0x%x\n", a_pSendData[0]);
            return -1;
        }


        i4RetValue = i2c_master_recv(g_pstI2Cclient2, (char *)a_pRecvData, a_sizeRecvData);
        if (i4RetValue != a_sizeRecvData) {
            PK_ERR("[CAMERA SENSOR] I2C read failed!!\n");
            return -1;
        }
    }
    return 0;
}
```

这一步完成 I2c 的读取，也就是说如果 I2c 配置正确，并且上电正确，到这一步就可以正确的读取 ID，整个 camera 也就基本就调通了。

三、总结

通过上述分析，我们可以看出，camera 驱动先是注册平台驱动，再注册 I2c 驱动，然后又为前后摄注册字符设备，封装底层方法，上层访问底层驱动时候显示使用 setdriver 将具体 IC 的驱动入口获取，然后使用 checkisalive 对 sensorlist 中的 IC 进行上电，上电完成就读取设备 ID，到此为止，上层应用与底层驱动挂接完成，紧接着就是预览和拍照，不过这都是具体 IC 驱动的实现了。