Android 5.0 Camera系统源码分析(5)：Camera预览3A流程

标签： MTK Android Camera 3A 调用流程

2016-10-18 14:39 2425人阅读 评论(3) 收藏 举报

目录(?) [+]

1. 前言

本文分析的是**Android** Hal层的源码，硬件平台基于mt6735。之前几篇讲的预览流程中3A相关的环节都忽略了，现在重新整理下。

3A指的是Auto Exposure，Auto Focus，Auto White Balance。这三个一起放上来代码实在太多了，这里将重点记录AF的代码。AF的部分工作是由ISP完成的，而ISP的大部分代码mtk都没有开放给我们，比如ISP是如何计算得到对焦位置信息的，但得到对焦位置之后怎么操作对焦马达的代码我们是看得到的，所以涉及到ISP的一些代码将被略过

2. 初始化3A

3A的初始化在DefaultCam1Device的onInit函数里面开始，之前在camera打开流程里面已经提到过

```
bool
DefaultCam1Device::
onInit()
{
    ......
    //  (1) Open 3A
    mpHal3a = NS3A::IHal3A::createInstance(
            NS3A::IHal3A::E_Camera_1,
            getOpenId(),
            LOG_TAG);
    ......
}
```

构造一个Hal3A对象，看下Hal3A::createInstance的实现

```
Hal3A*
Hal3A::
createInstance(MINT32 i4SensorDevId, MINT32 i4SensorOpenIndex)
{
    switch (i4SensorDevId)
    {
        case SENSOR_DEV_MAIN:
            Hal3ADev<SENSOR_DEV_MAIN>::getInstance()->init(i4SensorDevId, i4SensorOpenIndex);
            return Hal3ADev<SENSOR_DEV_MAIN>::getInstance();
        break;
        case SENSOR_DEV_SUB:
```

```
            Hal3ADev<SENSOR_DEV_SUB>::getInstance()->init(i4SensorDevId, i4SensorOpenIndex);
            return Hal3ADev<SENSOR_DEV_SUB>::getInstance();
        break;
        ......
    }
}
```

其实这里的Hal3A并没有直接继承IHal3A，也就是说从IHal3A::createInstance到Hal3A::createInstance的调用过程经历了一番波折，但暂时不用关心它。从Hal3A::createInstance可以看到除了实例化以外还会调用init函数。构造函数没什么好看的-略过，直接看init函数

```
MRESULT
Hal3A::
init(MINT32 i4SensorDevId, MINT32 i4SensorOpenIndex)
{
    ......
    // (1)
    mpStateMgr = new StateMgr(i4SensorDevId);
    // (2)
    bRet = postCommand(ECmd_Init);
    // (3)
    createThread();
    // (4)
    bRet = IspTuningMgr::getInstance().init(m_i4SensorDev, m_i4SensorOpenIdx);
    // (5)
    ret = EnableAFThread(1);
    ......

    return S_3A_OK;
}
```

步骤(1) new StateMgr，构造函数如下

```
StateMgr::StateMgr(MINT32 sensorDevId)
     : ......
{
    #define STATE_INITIALIZE(_state_)\
         mpIState[eState_##_state_] = new State##_state_(sensorDevId, this);

    STATE_INITIALIZE(Init);
    STATE_INITIALIZE(Uninit);
    STATE_INITIALIZE(CameraPreview);
    STATE_INITIALIZE(CamcorderPreview);
    STATE_INITIALIZE(Recording);
    STATE_INITIALIZE(Precapture);
    STATE_INITIALIZE(Capture);
    STATE_INITIALIZE(AF);

    mpCurrentState = mpIState[eState_Uninit];
}
```

初始化3A的状态管理，将各个子状态都保存在mpIState数组里面，并将当前状态设置为Uninit状态

步骤(2) postCommand

```
MBOOL Hal3A::postCommand(ECmd_T const eCmd, MINTPTR const i4Arg)
{
    ......
    ERROR_CHECK(mpStateMgr->sendCmd(eCmd))
    ......
}
```

```
MRESULT StateMgr::sendCmd(ECmd_T eCmd)
{
    Mutex::Autolock lock(m_Lock);

    EIntent_T eNewIntent = static_cast<EIntent_T>(eCmd);

    #define SEND_INTENT(_intent_)\
    case _intent_: return mpCurrentState->sendIntent(intent2type<_intent_>());\

    switch (eNewIntent)
    {
    SEND_INTENT(eIntent_CameraPreviewStart)
    SEND_INTENT(eIntent_CameraPreviewEnd)
    SEND_INTENT(eIntent_CaptureStart)
    SEND_INTENT(eIntent_CaptureEnd)
    SEND_INTENT(eIntent_RecordingStart)
    SEND_INTENT(eIntent_RecordingEnd)
    SEND_INTENT(eIntent_AFUpdate)
    SEND_INTENT(eIntent_AFStart)
    SEND_INTENT(eIntent_AFEnd)
    SEND_INTENT(eIntent_Init)
    SEND_INTENT(eIntent_Uninit)
    }
    return  -1;
}
```

从步骤(1)可以看出这里的mpCurrentState指向的是StateUninit对象，所以接着看StateUninit的sendIntent函数

```
MRESULT
StateUninit::
sendIntent(intent2type<eIntent_Init>)
{
    MY_LOG("[StateUninit::sendIntent]<eIntent_Init>");

    // AAO DMA buffer init
    MINT32 i4SensorIdx = m_pHal3A->getSensorOpenIdx();

if (ENABLE_3A_GENERAL & m_pHal3A->m_3ACtrlEnable) {
if (ENABLE_AAOBUF & m_pHal3A->m_3ACtrlEnable) {
    // AAO DMA buffer init
    if (!IAAOBufMgr::getInstance().init(m_SensorDevId, i4SensorIdx)) {
        MY_ERR("IAAOBufMgr::getInstance().init() fail");
        return E_3A_ERR;
    }

    if (!IAEBufMgr::getInstance().init(m_SensorDevId, i4SensorIdx)) {
        MY_ERR("IAEBufMgr::getInstance().init() fail");
```

```
        return E_3A_ERR;
    }
}
if (ENABLE_AFOBUF & m_pHal3A->m_3ACtrlEnable) {
    // AFO DMA buffer init
    if (!IAFOBufMgr::getInstance().init(m_SensorDevId, i4SensorIdx)) {
        MY_ERR("IAFOBufMgr::getInstance().init() fail");
        return E_3A_ERR;
    }
}
}

    // State transition: eState_Uninit --> eState_Init
    m_pStateMgr->transitState(eState_Uninit, eState_Init);

    return  S_3A_OK;
}
```

做了一堆乱七八糟的初始化之后将3A状态从Uninit状态切换到Init状态

步骤(3) createThread和步骤(5) EnableAFThread

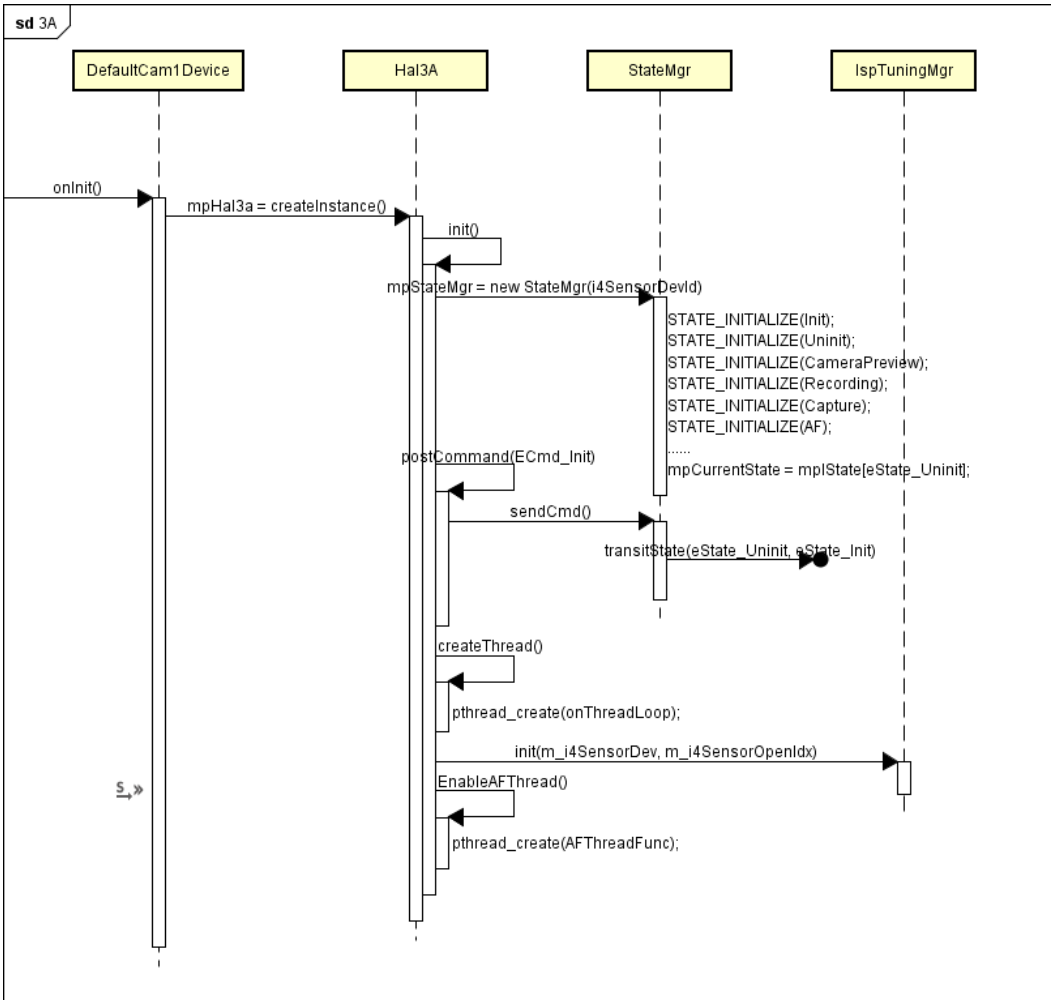
```
MVOID
Hal3A::createThread()
{
    ......
    pthread_create(&mThread, NULL, onThreadLoop, this);
    pthread_create(&mPDThread, NULL, PDThreadLoop, this);
    pthread_create(&mPDVCThread, NULL, PDVCThreadLoop, this);
    ......
}
```

```
MRESULT Hal3A::EnableAFThread(MINT32 a_bEnable)
{
    if (a_bEnable)  {
        if (mbAFThreadLoop== 0)
        {
            ......
            pthread_create(&mAFThread, &attr, AFThreadFunc, this);
        }
    } else {
        ......
    }

    return ret;
}
```

一共创建了4个线程，暂时只关心onThreadLoop 和AFThreadFunc。onThreadLoop是3A主线程，负责接收处理命令；

AFThreadFunc负责实时更新AF参数



3. 处理PASS1_START_ISP事件

前面的3A初始化做的事情并不多，更多的准备工作是在接收到PASS1_START_ISP事件之后做的，PASS1_START_ISP事件是在之前的Camera预览流程控制流中提到的Pass1Node的startHw函数里面发送

```
MBOOL
Pass1NodeImpl::
startHw(list<HwPortConfig_t> & plPortCfg)
{
    ......
    handleNotify(PASS1_START_ISP, newMagicNum, 0);
    ......
}
```

3.1 DefaultCtrlNode接收处理PASS1_START_ISP事件

Pass1Node发出的event将在DefaultCtrlNode的onNotify函数中接收处理

```
MBOOL
DefaultCtrlNodeImpl::
onNotify(MUINT32 const msg, MUINT32 const ext1, MUINT32 const ext2)
{
    switch(msg)
    {
        case PASS1_START_ISP:
```

```
        {
            if(mpHal3a)
            {
                cmd = ECmd_CameraPreviewStart;
                ......
                mpHal3a->sendCommand(cmd);
            }
        case PASS1_STOP_ISP:
        {
            ......
        }
        case PASS1_EOF:
        {
            ......
        }
        default:
        {
            ret = MTRUE;
        }
    }
    return ret;
}
```

Hal3a的sendCommand函数会把命令加入到命令队列，然后由主线程onThreadLoop获取

```
MVOID*
Hal3A::onThreadLoop(MVOID *arg)
{
    while (_this->getCommand(rCmd, bGetCmd, MFALSE))
    {
        switch (rCmd.eCmd)
        {
            case ECmd_PrecaptureStart:
            {
                ......
            }
            case ECmd_Update:
            {
                ......
            }
            default:
                if ( ! _this->postCommand(rCmd.eCmd, reinterpret_cast<MINTPTR>(&rCmd.rParamIspProfile)))
                {
                    MY_ERR("Cmd(%d) failed(0x%x)", rCmd.eCmd, _this->getErrorCode());
                    AEE_ASSERT_3A_HAL("onThreadLoop postCommand fail(2).");
                }
        }
    }
}
```

onThreadLoop通过getCommand函数获取命令，获取到命令之后调用postCommand函数对命令进行处理

再看一次postCommand

```
MBOOL Hal3A::postCommand(ECmd_T const eCmd, MINTPTR const i4Arg)
{
```

```
    if( eCmd == ECmd_CameraPreviewStart || eCmd == ECmd_CaptureStart)
    {
        mbEnAESenThd = MTRUE;
        createAEThread();
        mEnFlushVSIrq = mFlushVSIrqDone = 0;
        mEnFlushAFIrq = mFlushAFIrqDone = 0;
    }

    ......
    ERROR_CHECK(mpStateMgr->sendCmd(eCmd))
    ......

    return MTRUE;
}
```

接收到的命令是ECmd_CameraPreviewStart，所以这里的createAEThread函数会执行

```
MVOID
Hal3A::createAEThread()
{
    pthread_create(&mAESenThread, NULL, AESensorThreadLoop, this);
}
```

加上这个AESensorThreadLoop，需要关注的线程增加到了3个

3.2 StateInit处理CameraPreviewStart命令

继续看mpStateMgr->sendCmd函数。之前介绍过，它会把命令交给当前状态的sendIntent函数进行处理。在初始化阶段已经把当前状态切换到init状态，所以来看StateInit的sendIntent的实现

```
MRESULT
StateInit::
sendIntent(intent2type<eIntent_CameraPreviewStart>)
{

if (ENABLE_3A_GENERAL & m_pHal3A->m_3ACtrlEnable) {
if (ENABLE_AAOBUF & m_pHal3A->m_3ACtrlEnable) {
    // AAO DMAInit + AAStatEnable
    if (!IAAOBufMgr::getInstance().DMAInit(m_SensorDevId)) {
        MY_ERR("IAAOBufMgr::getInstance().DMAInit() fail");
        return E_3A_ERR;
    }
    if (!IAAOBufMgr::getInstance().AAStatEnable(m_SensorDevId, MTRUE)) {
        MY_ERR("IAAOBufMgr::getInstance().AAStatEnable() fail");
        return E_3A_ERR;
    }
    if (!IAEBufMgr::getInstance().DMAInit(m_SensorDevId)) {
        MY_ERR("IAEBufMgr::getInstance().DMAInit() fail");
        return E_3A_ERR;
    }

    if (!IAEBufMgr::getInstance().AAStatEnable(m_SensorDevId, MTRUE)) {
        MY_ERR("IAEBufMgr::getInstance().AAStatEnable() fail");
        return E_3A_ERR;
```

```
    }
}
if (ENABLE_AFOBUF & m_pHal3A->m_3ACtrlEnable) {
    // AFO DMAInit + AFStatEnable
    if (!IAFOBufMgr::getInstance().DMAInit(m_SensorDevId)) {
        MY_ERR("IAFOBufMgr::getInstance().DMAInit() fail");
        return E_3A_ERR;
    }
    if (!IAFOBufMgr::getInstance().AFStatEnable(m_SensorDevId, MTRUE)) {
        MY_ERR("IAFOBufMgr::getInstance().AFStatEnable() fail");
        return E_3A_ERR;
    }
}

    ......

if (ENABLE_AWB & m_pHal3A->m_3ACtrlEnable) {
    // AWB init
    bRet = (m_pHal3A->get3APreviewMode() == EPv_Normal)
        ? IAwbMgr::getInstance().cameraPreviewInit(m_SensorDevId, i4SensorIdx, rParam)
        : IAwbMgr::getInstance().camcorderPreviewInit(m_SensorDevId, i4SensorIdx, rParam);
    if (!bRet) {
        MY_ERR("IAwbMgr::getInstance().PreviewInit() fail, PvMode = %d\n", m_pHal3A->get3APreviewMode());
        return E_3A_ERR;
    }
}

if (ENABLE_AE & m_pHal3A->m_3ACtrlEnable) {
    // AE init
    err = (m_pHal3A->get3APreviewMode() == EPv_Normal)
        ? IAeMgr::getInstance().cameraPreviewInit(m_SensorDevId, i4SensorIdx, rParam)
        : IAeMgr::getInstance().camcorderPreviewInit(m_SensorDevId, i4SensorIdx, rParam);
    if (FAILED(err)) {
        MY_ERR("IAeMgr::getInstance().PreviewInit() fail, PvMode = %d\n", m_pHal3A->get3APreviewMode());
        return err;
    }
}
if (ENABLE_AF & m_pHal3A->m_3ACtrlEnable) {
    // AF init
    err = IAfMgr::getInstance().init(m_SensorDevId, i4SensorIdx);
    if (FAILED(err)) {
        MY_ERR("AfMgr::getInstance().init() fail\n");
        return err;
    }
}

    IspTuningMgr::getInstance().sendIspTuningIOCtrl(m_SensorDevId, IspTuningMgr::E_ISPTUNING_SET_GMA_SCENA
    IspTuningMgr::getInstance().sendIspTuningIOCtrl(m_SensorDevId, IspTuningMgr::E_ISPTUNING_NOTIFY_START,

    // Reset frame count to -2
    m_pStateMgr->resetFrameCount();

    // State transition: eState_Init --> eState_CameraPreview
    m_pStateMgr->transitState(eState_Init, eState_CameraPreview);

    return  S_3A_OK;
}
```

包含了AWB、AE、AF在内的ISP相关的初始化，相关的初始化完成之后会调用m_pStateMgr->transitState函数将当前状态切换到CameraPreview状态。

代码太多，这里只关注AF的初始化

```
MRESULT AfMgr::init(MINT32 i4SensorIdx, MINT32 isInitMCU)
{

    ......

    // --- init MCU ---
    SensorStaticInfo rSensorStaticInfo;
    if (m_i4EnableAF == -1)
    {
        IHalSensorList* const pIHalSensorList = IHalSensorList::get();
        IHalSensor* pIHalSensor = pIHalSensorList->createSensor("af_mgr", m_i4SensorIdx);
        SensorDynamicInfo rSensorDynamicInfo;

        switch(m_i4CurrSensorDev)
        {
            case ESensorDev_Main:
                pIHalSensorList->querySensorStaticInfo(NSCam::SENSOR_DEV_MAIN, &rSensorStaticInfo);
                pIHalSensor->querySensorDynamicInfo(NSCam::SENSOR_DEV_MAIN, &rSensorDynamicInfo);
                break;
            case ESensorDev_Sub:
                ......
            default:
                MY_ERR("Invalid sensor device: %d", m_i4CurrSensorDev);
        }
        if(pIHalSensor) pIHalSensor->destroyInstance("af_mgr");

        ......

        m_i4CurrSensorId=rSensorStaticInfo.sensorDevID;
        MCUDrv::lensSearch(m_i4CurrSensorDev, m_i4CurrSensorId);
        m_i4CurrLensId = MCUDrv::getCurrLensID(m_i4CurrSensorDev);
        ......
    }

    if(isInitMCU)
    {
        m_pMcuDrv = MCUDrv::createInstance(m_i4CurrLensId);

        if (m_pMcuDrv->init(m_i4CurrSensorDev) < 0)
        {
            MY_ERR("m_pMcuDrv->init() fail");
            m_i4EnableAF = 0;
        }
        else
        {
            m_pMcuDrv->moveMCU( 0, m_i4CurrSensorDev);
            m_MoveLensTimeStamp = getTimeStamp();
        }
    }

    // --- init ISP Drv/Reg ---
    ......
```

```
54
55      // --- checking PDAF is supported or not ---
56      ......
57
58      // --- init af algo ---
59      ......
60
61      // --- NVRAM ---
62      int err;
63      err = NvBufUtil::getInstance().getBufAndRead(CAMERA_NVRAM_DATA_LENS, m_i4CurrSensorDev, (void*&)g_pNVI
64      if(err!=0)
65          MY_ERR("AfAlgo NvBufUtil get buf fail! \n");
66      m_NVRAM_LENS.rFocusRange = g_pNVRAM_LENS->rFocusRange;
67      m_NVRAM_LENS.rAFNVRAM= g_pNVRAM_LENS->rAFNVRAM;
68      m_NVRAM_LENS.rPDNVRAM    = g_pNVRAM_LENS->rPDNVRAM;
69
70      // --- Param ---
71      m_sAFParam = getAFParam();
72      m_sAFConfig = getAFConfig();
73      m_pIAfAlgo->setAFParam(m_sAFParam, m_sAFConfig, m_NVRAM_LENS.rAFNVRAM);
74      m_pIAfAlgo->initAF(m_sAFInput, m_sAFOutput);
75      ......
76      m_pIAfAlgo->setAFMode(m_eLIB3A_AFMode);
77
78      //init pd mgr
79      ......
80
81      return S_AF_OK;
    }
```

第30行，调用lensSearch函数匹配镜头驱动

第35-49行，将镜头移动到起始位置

第60-67行，获取af tuning参数

第69-73行，设置af tuning参数

3.3 匹配镜头驱动

MTK为多个镜头做了兼容，所以AF初始化的第一步就是找到当前对应镜头的型号。lensSearch函数实现了lens的匹配过程

```
1
2   int
3   MCUDrv::lensSearch( unsigned int a_u4CurrSensorDev, unsigned int a_u4CurrSensorId)
4   {
5       INT32 i;
6
7       LensCustomInit(a_u4CurrSensorDev);
8       if (a_u4CurrSensorDev == MCU_DEV_MAIN )
9       {
10          LensCustomGetInitFunc(&MCUDrv::m_LensInitFunc_main[0]);
11          MCUDrv::m_u4CurrLensIdx_main = 0;
12
13          for (i=0; i<MAX_NUM_OF_SUPPORT_LENS; i++)
14          {
15              if ((MCUDrv::m_LensInitFunc_main[i].LensId == DUMMY_LENS_ID) ||
                    (MCUDrv::m_LensInitFunc_main[i].LensId == SENSOR_DRIVE_LENS_ID) /*||
```

```
                (MCUDrv::m_LensInitFunc_main[i].LensId == FM50AF_LENS_ID)*/
                )
            {
                MCUDrv::m_u4CurrLensIdx_main = i;
            }
        }

        // force assign LensIdx if SensorId != DUMMY_SENSOR_ID (to support backup lens/new lens driver)
        for (i=0; i<MAX_NUM_OF_SUPPORT_LENS; i++)
        {
            if ((MCUDrv::m_LensInitFunc_main[i].SensorId == a_u4CurrSensorId) && (a_u4CurrSensorId!=0xFFFF
            {
                MCUDrv::m_u4CurrLensIdx_main = i;
                MCU_DRV_DBG("[idx]%d [CurrSensorId]0x%04x,[CurrLensIdx]0x%04x\n", i, a_u4CurrSensorId, MCU
                break;
            }
        }
        LensCustomSetIndex(MCUDrv::m_u4CurrLensIdx_main);
        MCU_DRV_DBG("[CurrLensIdx]%d", MCUDrv::m_u4CurrLensIdx_main);

    }
    else if( a_u4CurrSensorDev == MCU_DEV_SUB)
    {
        ......
    }
    else
        return MCU_INVALID_DRIVER;

    return MCU_NO_ERROR;
}
```

先看LensCustomInit函数，由它来获取拷贝整个lens列表

```
MUINT32 LensCustomInit(unsigned int a_u4CurrSensorDev)
{
    GetLensInitFuncList(&LensInitFunc[0], a_u4CurrSensorDev);
    return 0;
}
```

```
UINT32 GetLensInitFuncList(PMSDK_LENS_INIT_FUNCTION_STRUCT pLensList, unsigned int a_u4CurrSensorDev)
{
    if(a_u4CurrSensorDev==2) //sub
        memcpy(pLensList, &LensList_sub[0], sizeof(MSDK_LENS_INIT_FUNCTION_STRUCT)* MAX_NUM_OF_SUPPORT_LEN
    else if(a_u4CurrSensorDev==4) //main 2
        memcpy(pLensList, &LensList_main2[0], sizeof(MSDK_LENS_INIT_FUNCTION_STRUCT)* MAX_NUM_OF_SUPPORT_L
    else  // main or others
        memcpy(pLensList, &LensList_main[0], sizeof(MSDK_LENS_INIT_FUNCTION_STRUCT)* MAX_NUM_OF_SUPPORT_LE

    return MHAL_NO_ERROR;
}
```

根据前后摄像头拷贝不同的LensList，这里只看其中的LensList_main

```
MSDK_LENS_INIT_FUNCTION_STRUCT LensList_main[MAX_NUM_OF_SUPPORT_LENS] =
```

```
{
    {DUMMY_SENSOR_ID, DUMMY_LENS_ID, "Dummy", pDummy_getDefaultData},
    #if defined(SENSORDRIVE)
        {OV3640_SENSOR_ID, SENSOR_DRIVE_LENS_ID, "kd_camera_hw", pSensorDrive_getDefaultData},
    #endif
    #if defined(FM50AF)
        {DUMMY_SENSOR_ID, FM50AF_LENS_ID, "FM50AF", pFM50AF_getDefaultData},
    #endif
    #if defined(DW9714AF)
        {IMX135_SENSOR_ID, DW9714AF_LENS_ID, "DW9714AF", pDW9714AF_getDefaultData},
    #endif
    ......
};
```

LensList_main包含了后摄所有可用的lens，如果要新增一个lens驱动，就需要往这个数组添加相关的信息，看下MSDK_LENS_INIT_FUNCTION_STRUCT结构体的定义

```
typedef struct
{
    UINT32 SensorId;
    UINT32 LensId;
    UINT8  LensDrvName[32];
    UINT32 (*getLensDefault)(VOID *pDataBuf, UINT32 size);

} MSDK_LENS_INIT_FUNCTION_STRUCT, *PMSDK_LENS_INIT_FUNCTION_STRUCT;
```

SensorId：表示这个lens driver配置给对应的sensor使用，如果配置成DUMMY_SENSOR_ID则表示这个lens driver适合所有sensor使用

LensId：Lens driver的唯一标识

LensDrvName：是实现AF功能的驱动对应的设备驱动节点名。例如FM20AF，实现AF功能的是内核驱动里面的fm20af.c，这个驱动会生成一个驱动节点”/dev/fm20af”

getLensDefault：这个函数指针指向了获取AF tuning参数表的一个函数

回到lensSearch函数，现在lens列表已经保存在LensInitFunc数组中了，接下来调用LensCustomGetInitFunc函数把它拷贝到m_LensInitFunc_main数组中

```
MUINT32 LensCustomGetInitFunc(MSDK_LENS_INIT_FUNCTION_STRUCT *a_pLensInitFunc)
{
    if (a_pLensInitFunc != NULL) {
        memcpy(a_pLensInitFunc, &LensInitFunc[0], sizeof(MSDK_LENS_INIT_FUNCTION_STRUCT) * MAX_NUM_OF_SUPI
        return 0;
    }
    return -1;
}
```

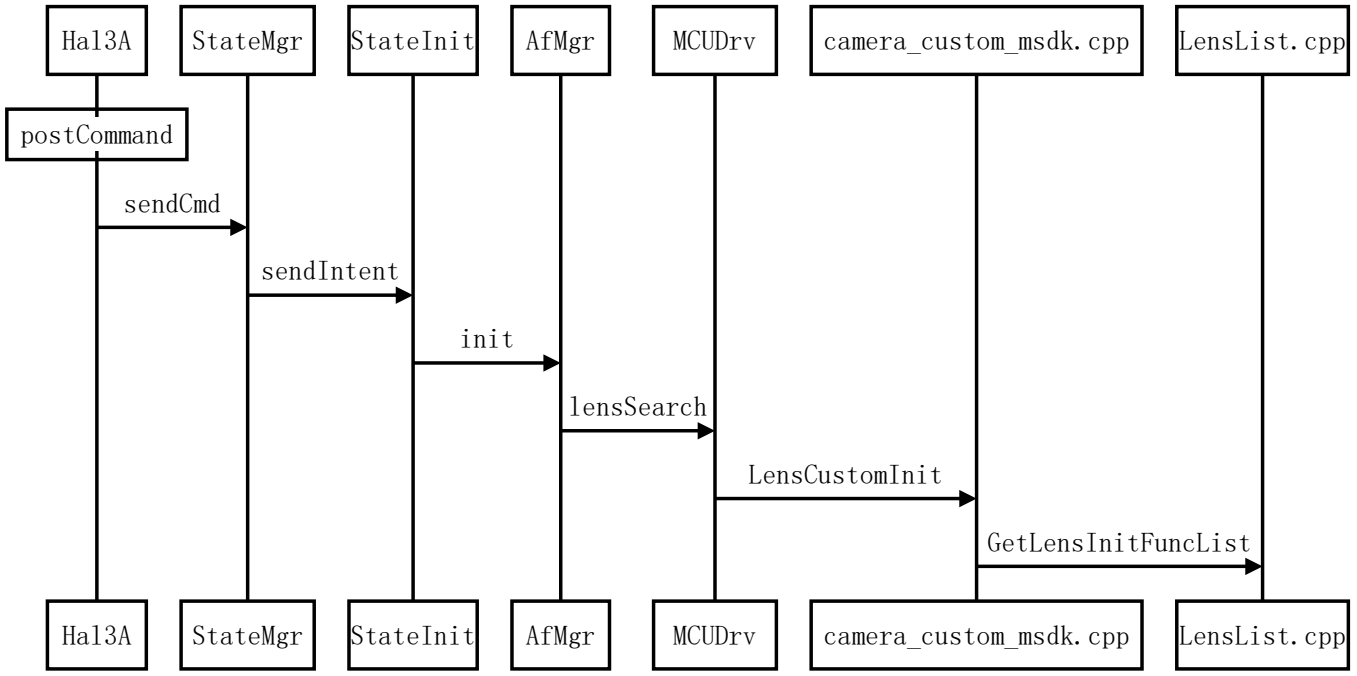
lensSearch函数再接下来就是两个for循环，从m_LensInitFunc_main数组中搜索符合要求的lens驱动

第一个for循环：m_u4CurrLensIdx_main指向m_LensInitFunc_main数组里lensId配置为DUMMY_LENS_ID或

SENSOR_DRIVE_LENS_ID的最后一个元素

第二个for循环：查找m_LensInitFunc_main数组中是否有SensorId和当前使用的sensor的ID相匹配的lens driver。如果有则为符合条件的第一个元素，没有的话则为第一个for循环搜索到的结果

最后调用LensCustomSetIndex把匹配到的lens driver的index保存下来



3.4 初始化镜头驱动

找到镜头驱动之后还需要对镜头进行初始化，也就是将镜头移动到起始位置，所以回到AfMgr::init函数

主要是下面这3行代码

```
m_pMcuDrv = MCUDrv::createInstance(m_i4CurrLensId);
m_pMcuDrv->init(m_i4CurrSensorDev);
m_pMcuDrv->moveMCU( 0, m_i4CurrSensorDev);
```

MCUDrv::createInstance创建的是LensDrv对象，它继承了MCUDrv类

接下来调用m_pMcuDrv->init函数来打开设备驱动节点，然后调用m_pMcuDrv->moveMCU函数来控制焦马达将镜头移动到起始位置

```
int
LensDrv::init(unsigned int a_u4CurrSensorDev )
{

    char  cBuf[64];
    unsigned int a_u4CurrLensIdx;

    if(a_u4CurrSensorDev==MCU_DEV_MAIN)
    {
        a_u4CurrLensIdx=MCUDrv::m_u4CurrLensIdx_main;

```

```
12          sprintf(cBuf, "/dev/%s", MCUDrv::m_LensInitFunc_main[a_u4CurrLensIdx].LensDrvName);
13          DRV_DBG("main lens init() [m_userCnt]%d  +\n", m_userCnt_main);
14          DRV_DBG("[main Lens Driver]%s\n", cBuf);
15
16          Mutex::Autolock lock(mLock);
17
18          if (m_userCnt_main == 0) {
19              if (m_fdMCU_main == -1) {
20                  m_fdMCU_main = open(cBuf, O_RDWR);
21                  if (m_fdMCU_main < 0) {
22                      ......
23                  }
24              }
25          }
26          m_userCnt_main++;
27          DRV_DBG("main lens init() [m_userCnt]%d [fdMCU_main]%d - \n", m_userCnt_main,m_fdMCU_main);
28      }
29      else if(a_u4CurrSensorDev==MCU_DEV_SUB)
30      {
31          ......
32      }
33      else
34          return MCUDrv::MCU_INVALID_DRIVER;
35
36      return MCUDrv::MCU_NO_ERROR;
37  }
```

第20行，根据之前匹配到的lens信息打开设备驱动节点，例如/dev/fm20af

```
1
2   int
3   LensDrv::moveMCU(int a_i4FocusPos,unsigned int a_u4CurrSensorDev )
4   {
5       //DRV_DBG("moveMCU() - pos = %d \n", a_i4FocusPos);
6       int err, a_fdMCU,a_u4CurrLensIdx;
7
8       if(a_u4CurrSensorDev==MCU_DEV_MAIN)
9       {
10          a_fdMCU=m_fdMCU_main;
11          a_u4CurrLensIdx=MCUDrv::m_u4CurrLensIdx_main;
12      }
13      else if(a_u4CurrSensorDev==MCU_DEV_SUB)
14      {
15          ......
16      }
17
18      ......
19
20      err = ioctl(a_fdMCU,mcuIOC_T_MOVETO,(unsigned long)a_i4FocusPos);
21      if (err < 0) {
22          DRV_ERR("[moveMCU] ioctl - mcuIOC_T_MOVETO, error %s",  strerror(errno));
23          return err;
24      }
25
26      return MCUDrv::MCU_NO_ERROR;
27  }
```

第19行，通过ioctl函数来移动对焦马达，kernel层对应的lens驱动会通过i2c设置lens的寄存器。ioctl的最后一个参数

a_i4FocusPos代表将镜头移动到什么位置

3.5 设置AF参数

接下来还需要把之前获取到的lens tuning参数表设置到ISP里面，在AfMgr::init函数的第60-73行代码。

首先通过NvBufUtil的getBufAndRead函数读取lens tuning参数表，这个函数最终将调用到GetLensDefaultPara函数

```
1  void GetLensDefaultPara(PNVRAM_LENS_PARA_STRUCT pLensParaDefault)
2  {
3      MUINT32 i;
4
5      MUINT32 LensId = LensInitFunc[gMainLensIdx].LensId;
6
7      if (LensInitFunc[0].getLensDefault == NULL)
8      {
9          CAM_MSDK_LOG("[GetLensDefaultPara]: uninit yet\n\n");
10         return;
11     }
12
13     for (i=0; i<MAX_NUM_OF_SUPPORT_LENS; i++)
14     {
15         if (LensId == LensInitFunc[i].LensId)
16         {
17             break;
18         }
19     }
20
21     if (pLensParaDefault != NULL)
22     {
23         LensInitFunc[i].getLensDefault((VOID*)pLensParaDefault, sizeof(NVRAM_LENS_PARA_STRUCT));
24     }
25
26 }
```

第23行，LensInitFunc[i].getLensDefault之前已经提到过，getLensDefault指向对应镜头的getDefaultData函数指针，例如fm50af的pFM50AF_getDefaultData，而这个函数指针则指向FM50AF_getDefaultData函数，调用这个函数会将lens tuning参数表拷贝到buff里

```
const NVRAM_LENS_PARA_STRUCT FM50AF_LENS_PARA_DEFAULT_VALUE =
{
    //Version
    NVRAM_CAMERA_LENS_FILE_VERSION,

    // Focus Range NVRAM
    {0, 1023},

    // AF NVRAM
    {
        // ------------------------------ sAF_Coef ---------------------------------------
        {
            {
                200, // i4Offset
                13,  // i4NormalNum
                13,  // i4MacroNum
                0,   // i4InfIdxOffset
                0,   // i4MacroIdxOffset
                {
                    0,  20,  45,  70,  95, 120, 150, 180, 220, 260,
                    305, 355, 405,   0,   0,   0,   0,   0,   0,   0,
                    0,   0,   0,   0,   0,   0,   0,   0,   0,   0
                }
            },
            15, // i4THRES_MAIN;
            10, // i4THRES_SUB;
            1,  // i4AFC_FAIL_CNT;
            0,  // i4FAIL_POS;

            4,  // i4INIT_WAIT;
            {500, 500, 500, 500, 500}, // i4FRAME_WAIT
            0,  // i4DONE_WAIT;
        },
```

获取到lens tuning参数表之后调用m_pIAfAlgo->setAFParam函数将参数表设置到ISP里面，很遗憾IAfAlgo相关函数的实现我们是看不到的

AfMgr::init函数执行完之后，AF相关的准备工作就已经完成，接下来就是根据不同的场景实时更新3A参数了

4. 实时更新AF

之前提到过有3个需要重点关注的线程，onThreadLoop、AFThreadFunc和AESensorThreadLoop。

其中AFThreadFunc负责实时更新AF参数

```
MVOID * Hal3A::AFThreadFunc(void *arg)
{
    ......
    while (_this->mbAFThreadLoop) {
        if ( _this->mpIspDrv_forAF->waitIrq(&waitIrq) > 0) // success
        {
            MY_LOG_IF(fgLogEn, "[Hal3A::AFThreadFunc] AF waitIrq done\n");
            _this->mpScheduler->jobAssignAndTimerStart(E_Job_Af);
            _this->mpStateMgr->sendCmd(ECmd_AFUpdate);
            _this->mpScheduler->jobTimerEnd(E_Job_Af);
            MY_LOG_IF(fgLogEn, "[Hal3A::AFThreadFunc] StateMgr::sendCmd(ECmd_AFUpdate) done\n");
        }

        ......
    }
    return NULL;
}
```

AFThreadFunc函数还挺长的，但除了上面贴出来的这些代码，其它的我都不知道它在做什么。当需要更新AF的参数时ISP会产生一个中断，而这里则通过一个死循环不断去捕获中断。捕获到中断之后通过mpStateMgr->sendCmd函数将命令交给当前状态的sendIntent函数进行处理。当前状态已经在处理PASS1_START_ISP事件时切换到CameraPreview状态了，所以AFUpdate命令将

在StateCameraPreview的sendIntent函数中处理

```
MRESULT
StateCameraPreview::
sendIntent(intent2type<eIntent_AFUpdate>)
{
    ......
    // (0) Dequeue AFO DMA buffer
    IAFOBufMgr::getInstance().dequeueHwBuf(m_SensorDevId, rBufInfo);

    // (1) get AF window from AF, and set to AE meter, then get Y value.
    IAfMgr::getInstance().getAFRefWin(m_SensorDevId, rWinSize);
    rAeWinSize.i4Left  =rWinSize.i4Left;
    rAeWinSize.i4Right =rWinSize.i4Right;
    rAeWinSize.i4Top   =rWinSize.i4Top;
    rAeWinSize.i4Bottom=rWinSize.i4Bottom;
    rAeWinSize.i4Weight=rWinSize.i4Weight;
    IAeMgr::getInstance().getAEMeteringYvalue(m_SensorDevId, rAeWinSize, &iYvalue);

    // (2) get current AE info, and write to AF for reference.
    IAeMgr::getInstance().getAEBlockYvalues(m_SensorDevId, rAEInfo.aeBlockV, 25);
    IAeMgr::getInstance().getPreviewParams(m_SensorDevId, rPreviewInfo);
    IAeMgr::getInstance().getRTParams(m_SensorDevId, AEFrameParam);
    rAEInfo.i4IsAEStable= IAeMgr::getInstance().IsAEStable(m_SensorDevId);
    rAEInfo.i4ISO=rPreviewInfo.u4RealISO;
    rAEInfo.i4SceneLV=IAeMgr::getInstance().getLVvalue(m_SensorDevId,MTRUE);
    rAEInfo.iYvalue=(MINT64)iYvalue;
    rAEInfo.ishutterValue=AEFrameParam.u4PreviewShutterSpeed_us;
    ......
    IAfMgr::getInstance().setAE2AFInfo(m_SensorDevId, rAEInfo);

    // (3) doAF
    IAfMgr::getInstance().doAF(m_SensorDevId, reinterpret_cast<MVOID *>(rBufInfo.virtAddr));

    // (4) Enqueue AFO DMA buffer
    IAFOBufMgr::getInstance().enqueueHwBuf(m_SensorDevId, rBufInfo);
    ......
    return  S_3A_OK;
}
```

直接看步骤(3)doAF函数吧，其它的步骤大部分是在为Algo设置参数，而Algo的代码不开放，所以也不知道设置的那些信息是做什么用的

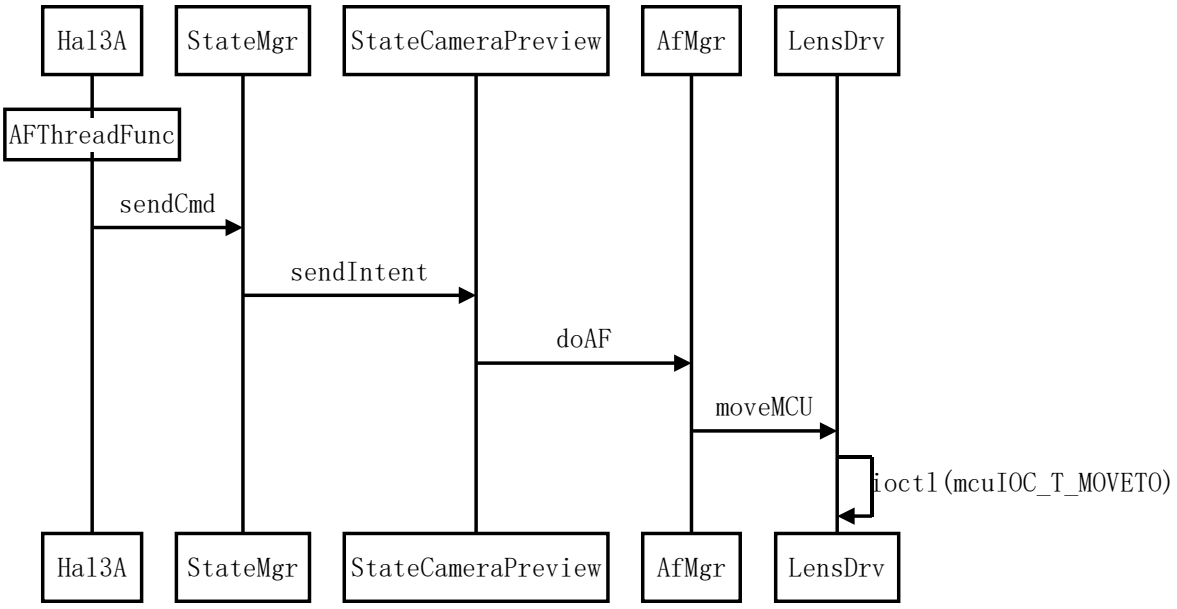
```
MRESULT AfMgr::doAF(MVOID *pAFStatBuf)
{
    if (m_i4EnableAF == 0)
    {
        m_sAFOutput.i4IsAFDone = 1;
        m_sAFOutput.i4IsFocused = 0;
        m_i4LastFocusModeTAF= FALSE;
        m_sAFOutput.i4AFPos = 0;
        mAFMgrInited = MTRUE;
        MY_LOG("disableAF");
        return S_AF_OK;
    }

```

```
15      //depth AF for algo data
16      m_DAF_TBL.curr_p1_frm_num=i4curFrmNum;
17      if(m_DAF_TBL.is_daf_run==1) m_sAFInput.i4HybridAFMode = 1;
18      else                        m_sAFInput.i4HybridAFMode = 0;
19
20      if(     m_PDAF_Sensor_Support_Mode==1 && m_PDBuf_Type==EPDBuf_Raw)     m_sAFInput.i4HybridAFMode = 2;
21      else if(m_PDAF_Sensor_Support_Mode==2 && m_PDBuf_Type==EPDBuf_VC)      m_sAFInput.i4HybridAFMode = 2;
22      else if(m_PDAF_Sensor_Support_Mode==2 && m_PDBuf_Type==EPDBuf_VC_Open) m_sAFInput.i4HybridAFMode = 10
23
24      if(m_DAF_TBL.is_daf_run==1)
25      {
26          m_sAFInput.i4CurrP1FrmNum = i4curFrmNum;
27          if(m_next_query_FrmNum == 0xFFFFFFFF)
28          {
29              m_sAFInput.i4DafDacIndex = 0;
30              m_sAFInput.i4DafConfidence = 0;
31          }
32          else
33          {
34          m_sAFInput.i4DafDacIndex  = m_DAF_TBL.daf_vec[m_next_query_FrmNum % DAF_TBL_QLEN].daf_dac_index;
35          m_sAFInput.i4DafConfidence= m_DAF_TBL.daf_vec[m_next_query_FrmNum % DAF_TBL_QLEN].daf_confidence;
36          }
37          MY_LOG("DAF--[Mode]%d [cp1#]%d [cp2#]%d [nextF#]%d [DafDac]%d [DafConf]%d [daf_dist]%d\n",
38          (MINT32)m_sAFInput.i4HybridAFMode,
39          (MINT32)m_sAFInput.i4CurrP1FrmNum,
40          (MINT32)m_DAF_TBL.curr_p2_frm_num,
41          (MINT32)m_next_query_FrmNum,
42          (MINT32)m_sAFInput.i4DafDacIndex,
43          (MINT32)m_sAFInput.i4DafConfidence,
44          m_DAF_TBL.daf_vec[m_next_query_FrmNum % DAF_TBL_QLEN].daf_distance);
45
46          if(m_DAF_TBL.daf_vec[m_next_query_FrmNum % DAF_TBL_QLEN].daf_confidence)
47          {
48              MY_LOG("DAFAA-%d %d\n", m_daf_distance, m_DAF_TBL.daf_vec[m_next_query_FrmNum % DAF_TBL_QLEN].
49              m_daf_distance = (MINT32)m_DAF_TBL.daf_vec[m_next_query_FrmNum % DAF_TBL_QLEN].daf_distance;
50          }
51      }
52
53      //PDAF
54      ......
55
56      //handle AF
57      if (m_pIAfAlgo)
58          m_pIAfAlgo->handleAF(m_sAFInput, m_sAFOutput);
59
60      //move AF
61      m_pMcuDrv->moveMCU(m_sAFOutput.i4AFPos,m_i4CurrSensorDev);
62
63      //set AF info to IspTuning
64      AF_INFO_T sAFInfo;
65      sAFInfo.i4AFPos = m_sAFOutput.i4AFPos;
66      IspTuningMgr::getInstance().setAFInfo(m_i4CurrSensorDev, sAFInfo);
67
68      return S_AF_OK;
    }
```

这个函数的代码还挺多的，看不懂的代码就不贴上来了。其中大部分代码都是为了第57行m_pIAfAlgo->handleAF函数做准备，

也就是设置m_sAFInput参数。m_pIAfAlgo->handleAF会计算得到对焦位置信息并保存在m_sAFOutput参数里面，当然algo的代码我们看不到。得到对焦位置信息之后会调用m_pMcuDrv->moveMCU函数来移动对焦马达，也就是第60行，这个函数之前已经分析过了。最后第65行需要把对焦信息设置到Isp Tuning里面。



就这样AFThreadFunc通过一个死循环等待ISP中断，然后计算出对焦位置并通知镜头驱动移动对焦马达

5. 实时更新AE

在上一篇Camera预览流程数据流里提到过，Pass1Node每deque一帧数据就会发出PASS1_EOF事件来更新3A，这个事件同样由DefaultCtrlNode的onNotify函数接收处理

5.1 处理PASS1_EOF事件

```
1  MBOOL
2  DefaultCtrlNodeImpl::
3  onNotify(MUINT32 const msg, MUINT32 const ext1, MUINT32 const ext2)
4  {
5
6      ......
7
8      switch(msg)
9      {
10         ......
11         case PASS1_EOF:
12         {
13             {
14                 Mutex::Autolock _l(mLock);
15                 //
16                 if( mpHal3a &&
17                     getFlag(muStateFlag, FLAG_DO_3A_UPDATE) && // to avoid send update after precaptrue-er
18                     ext1 != MAGIC_NUM_INVALID )
19                 {
20                     MUINT32 zoomRatio = 0, cropX = 0, cropY = 0, cropW = 0, cropH = 0;
21                     if(mpIspSyncCtrl->getCurPass2ZoomInfo(zoomRatio,cropX,cropY,cropW,cropH))
```

```
                    {
                        mpHal3a->setZoom(
                                    zoomRatio,
                                    cropX,
                                    cropY,
                                    cropW,
                                    cropH);
                    }
                    //do 3A update
                    mCurMagicNum = ext1;
                    ParamIspProfile_T _3A_profile(
                            mIspProfile,
                            mCurMagicNum,
                            MTRUE,
                            ParamIspProfile_T::EParamValidate_All);
                    mpHal3a->sendCommand(ECmd_Update, reinterpret_cast<MINTPTR>(&_3A_profile));
                }
                else
                {
                    MY_LOGD("skip update");
                }
            }
            ......
            break;
        }
        default:
        {
            ret = MTRUE;
        }
    }

    return ret;
}
```

第37行，向Hal3A发送update命令。和之前的AFUpdate命令一样，由Hal3A的postCommand函数接收命令，它再将命令转发给3A的当前状态，也就是CameraPreview状态处理

```
MRESULT
StateCameraPreview::
sendIntent(intent2type<eIntent_VsyncUpdate>)
{
    ......

    // update AE
    MBOOL isNeedUpdateI2C;
    IAeMgr::getInstance().doPvAEmonitor(m_SensorDevId, m_pStateMgr->getFrameCount(), reinterpret_cast<MVO
    m_pScheduler->jobAssignAndTimerStart(E_Job_AeFlare), 1,
    m_pScheduler->isSlowMotion(), isNeedUpdateI2C);

    MBOOL isSlowMotionUpdateI2C;
    isSlowMotionUpdateI2C = m_pScheduler->jobAssignAndTimerStart(E_Job_AeFlare) && m_pScheduler->isSlowMo

    if(isNeedUpdateI2C || isSlowMotionUpdateI2C)
    {
        MY_LOG_IF(fgLogEn, "[%s] postToAESenThread : wait to update I2C (%d, %d)", __FUNCTION__, isNeedUp
        m_pHal3A->postToAESenThread(MFALSE);
    }
```

```
21
22      IAeMgr::getInstance().doPvAE(m_SensorDevId, m_pStateMgr->getFrameCount(), reinterpret_cast<MVOID *>(rI
23      m_pScheduler->jobAssignAndTimerStart(E_Job_AeFlare), 1,
24      m_pScheduler->isSlowMotion());
25
26      m_pScheduler->jobTimerEnd(E_Job_AeFlare);
27      if (isNeedUpdateI2C || isSlowMotionUpdateI2C)
28      {
29          MY_LOG_IF(fgLogEn, "[%s] postToAESenThread : ready to update I2C", __FUNCTION__);
30          m_pHal3A->postToAESenThread(MTRUE);
31      }
32
33      IspTuningMgr::GMA_AE_DYNAMIC_INFO dynamicInfo;
34      dynamicInfo.bStable = IAeMgr::getInstance().IsAEStable(m_SensorDevId);
35      IspTuningMgr::getInstance().sendIspTuningIOCtrl(m_SensorDevId, IspTuningMgr::E_ISPTUNING_SET_GMA_AE_DY
36
37      // workaround for iVHDR
38      MUINT32 u4AFSGG1Gain;
39      IAeMgr::getInstance().getAESGG1Gain(m_SensorDevId, &u4AFSGG1Gain);
40      IAfMgr::getInstance().setSGGPGN(m_SensorDevId, (MINT32) u4AFSGG1Gain);
41
42      MY_LOG_IF(fgLogEn, "[StateCameraPreview::sendIntent<eIntent_VsyncUpdate>] doPvAE done\n");
43
44      // update AWB
45      if (m_pScheduler->jobAssignAndTimerStart(E_Job_Awb))
46          IAwbMgr::getInstance().doPvAWB(m_SensorDevId, m_pStateMgr->getFrameCount(), bAEStable, i4AoeCompLv
47      m_pScheduler->jobTimerEnd(E_Job_Awb);
48      MY_LOG_IF(fgLogEn, "[StateCameraPreview::sendIntent<eIntent_VsyncUpdate>] doPvAWB done\n");
49
50      return  S_3A_OK;
51  }
```

这个函数也是长得不要不要的，它除了更新了AE、AWB参数外，还更新了其他图像参数，但这里只关注AE。乱七八糟的代码略过，看第22行doPvAE函数的实现

```
1   MRESULT AeMgr::doPvAE(MINT32 i4FrameCount, MVOID *pAEStatBuf, MINT32 i4ActiveAEItem, MUINT32 u4AAOUpdate,
2   {
3       strAEInput rAEInput;
4       strAEOutput rAEOutput;
5
6       ......
7
8       rAEInput.pAESatisticBuffer = pAEStatBuf;
9       rAEInput.eAeTargetMode = m_eAETargetMode;
10      if(m_pIAeAlgo != NULL) {
11          if(m_bRestoreAE == MFALSE) {
12              AaaTimer localTimer("handleAE", m_eSensorDev, (m_3ALogEnable & EN_3A_SCHEDULE_LOG));
13              m_pIAeAlgo->handleAE(&rAEInput, &rAEOutput);
14              localTimer.End();
15
16              copyAEInfo2mgr(&m_rAEOutput.rPreviewMode, &rAEOutput);
17              m_rAEOutput.rCaptureMode[0] = m_rAEOutput.rPreviewMode;
18
19              mPreviewMode = m_rAEOutput.rPreviewMode;
20              m_i4WaitVDNum = 0; // reset the delay frame
21              if((rAEInput.eAeState == AE_STATE_NORMAL_PREVIEW) || (rAEInput.eAeState == AE_STATE_ONE_SHOT))
22                  m_bAEStable = rAEOutput.bAEStable;
```

```
23                  m_bAEMonitorStable = m_bAEStable;
24              }
25          } else {
26              bRestore=1;
27              m_bRestoreAE = MFALSE;
28              MY_LOG("Restore AE, skip AE one time\n");
29          }
30      } else {
31          MY_LOG("[%s()] The AE algo class is NULL  i4SensorDev = %d line:%d", __FUNCTION__, m_eSensorDev, _
32      }
33
34      ......
35      if ((i4ActiveItem & E_AE_AE_APPLY) || (bApplyAE == MTRUE)){   // apply AE
36          UpdateSensorISPParams(AE_AUTO_FRAMERATE_STATE);
37      }
38      ......
39
40      return S_AE_OK;
41  }
```

第13行，调用m_pIAeAlgo->handleAE函数计算AE相关的参数，包括曝光时间和亮度的Gain值

第19行，将得到的AE参数保存到mPreviewMode变量中

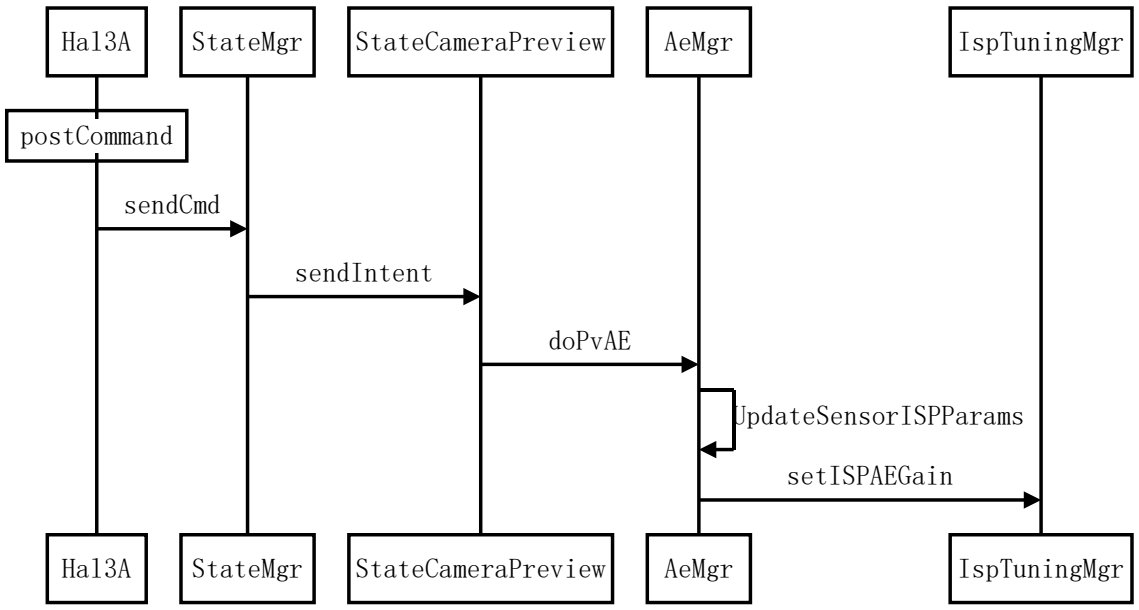
第36行，获取到AE参数之后调用UpdateSensorISPParams函数更新sensor和ISP的参数

```
1
2   MRESULT AeMgr::UpdateSensorISPParams(AE_STATE_T eNewAEState)
3   {
4       MRESULT err;
5       AE_INFO_T rAEInfo2ISP;
6       MUINT32 u4IndexRatio;
7
8       m_AEState = eNewAEState;
9
10      switch(eNewAEState)
11      {
12          case AE_INIT_STATE:
13          case AE_REINIT_STATE:
14              ......
15          case AE_AUTO_FRAMERATE_STATE:
16          case AE_MANUAL_FRAMERATE_STATE:
17              if(m_pIAeAlgo != NULL) {
18                  m_pIAeAlgo->getAEInfoForISP(rAEInfo2ISP, LIB3A_SENSOR_MODE_PRVIEW);
19                  rAEInfo2ISP.i4GammaIdx = m_i4GammaIdx;
20                  rAEInfo2ISP.i4LESE_Ratio = m_i4LESE_Ratio;
21                  rAEInfo2ISP.u4SWHDR_SE = m_u4SWHDR_SE;
22                  rAEInfo2ISP.u4MaxISO = m_u4MaxISO*m_rAEPLineLimitation.u4IncreaseISO_x100/100;
23
24                  rAEInfo2ISP.u4AEStableCnt = m_u4StableCnt;
25              }
26
27              rAEInfo2ISP.u4Eposuretime = mPreviewMode.u4Eposuretime;
28              rAEInfo2ISP.u4AfeGain = mPreviewMode.u4AfeGain;
29              rAEInfo2ISP.u4IspGain = mPreviewMode.u4IspGain;
30              rAEInfo2ISP.u4EVRatio = m_rAEInitInput.rAEPARAM.pEVValueArray[m_eAEEVcomp];
31
32              if(m_i4WaitVDNum <= m_i4IspGainDelayFrames) {
                    if(m_i4WaitVDNum == m_i4ShutterDelayFrames) {
```

```
33                      m_bSetShutterValue = MTRUE;
34                      m_u4UpdateShutterValue = mPreviewMode.u4Eposuretime;
35                      m_u4PrevExposureTime = mPreviewMode.u4Eposuretime;
36                  }
37                  if(m_i4WaitVDNum == m_i4SensorGainDelayFrames) {
38                      m_bSetGainValue = MTRUE;
39                      m_u4UpdateGainValue = mPreviewMode.u4AfeGain;
40                      m_u4PrevSensorGain = mPreviewMode.u4AfeGain;
41                  }
42
43                  IspTuningMgr::getInstance().setIspFlareGainOffset((ESensorDev_T)m_eSensorDev, mPreviewMode
44                  IspTuningMgr::getInstance().setAEInfo((ESensorDev_T)m_eSensorDev, rAEInfo2ISP);
45
46                  if(m_i4WaitVDNum == m_i4IspGainDelayFrames) {
47                      IspTuningMgr::getInstance().setISPAEGain((ESensorDev_T)m_eSensorDev, MFALSE, mPreviewM
48                      m_AEState = eNewAEState;
49                  }
50              }
51              break;
52          case AE_AF_STATE:
53          ......
54          default:
55              break;
56      }
57      return S_AE_OK;
58  }
```

第32-41行，把曝光时间和Gain值保存下来，后面会用到

第47行，将保存在mPreviewMode里的Gain值设置到Isp Tuning里面去，Isp Tuning的重点代码不开放，再往下跟已经没有意义了



还没结束，这里只更新了ISP的参数，并没有更新Sensor的参数

5.2 AESensorThreadLoop函数分析

之前提到有3个重点关注的线程，剩下最后一个了

```
MVOID*
Hal3A::AESensorThreadLoop(MVOID *arg)
{

    ......
    // (2) thread-in-loop
    while(1)
    {
        MY_LOG_IF(fgLogEn, "waitVsync start.");
        _this->waitVSirq();
        MY_LOG_IF(fgLogEn, "waitVsync done.");

        ......

        MY_LOG_IF(fgLogEn, "[AESensorThreadLoop] updateSensorbyI2C start\n");
        IAeMgr::getInstance().updateSensorbyI2C(_this->m_i4SensorDev);
        MY_LOG_IF(fgLogEn, "[AESensorThreadLoop] updateSensorbyI2C end\n");

    }

    return NULL;
}
```

和之前的AF线程一样，通过一个死循环不断去捕获中断。捕获到中断之后调用updateSensorbyI2C函数进行处理

```
MRESULT AeMgr::updateSensorbyI2C()
{
    MINT32 err = S_AE_OK;

    if(m_bSetFrameRateValue) {     // update frame rate
        m_bSetFrameRateValue = MFALSE;
        AaaTimer localTimer("SetFrameRater", m_eSensorDev, (m_3ALogEnable & EN_3A_SCHEDULE_LOG));
        err = AAASensorMgr::getInstance().setPreviewMaxFrameRate((ESensorDev_T)m_eSensorDev, m_u4UpdateFra
        localTimer.End();
        if (FAILED(err)) {
            MY_ERR("AAASensorMgr::getInstance().setPreviewMaxFrameRate fail\n");
        }
    }

    if((m_eAETargetMode == AE_MODE_AOE_TARGET) || (m_eAETargetMode == AE_MODE_MVHDR_TARGET)) {   // mVHDR/
        ......
    } else {   // normal control
        if(m_bSetShutterValue) {    // update shutter value
            AaaTimer localTimer("SetSensorShutter", m_eSensorDev, (m_3ALogEnable & EN_3A_SCHEDULE_LOG));
            err = AAASensorMgr::getInstance().setSensorExpTime((ESensorDev_T)m_eSensorDev, m_u4UpdateShut
            localTimer.End();
            m_bSetShutterValue = MFALSE;
            if (FAILED(err)) {
                MY_ERR("AAASensorMgr::getInstance().setSensorExpTime fail\n");
            }
        }

        if(m_bSetGainValue) {    // update sensor gain value
            AaaTimer localTimer("SetSensorGain", m_eSensorDev, (m_3ALogEnable & EN_3A_SCHEDULE_LOG));
            err = AAASensorMgr::getInstance().setSensorGain((ESensorDev_T)m_eSensorDev, m_u4UpdateGainValu
```

```
31              err = AAASensorMgr::getInstance().setSensorIso((ESensorDev_T)m_eSensorDev, m_eSensorMode, m_r
32              localTimer.End();
33              m_bSetGainValue = MFALSE;
34              if (FAILED(err)) {
35                  MY_ERR("AAASensorMgr::getInstance().setSensorGain fail\n");
36              }
37          }
38      }
39      return S_AE_OK;
40  }
```

第18-26行，更新设置sensor的快门打开时间，也就是曝光时间

第28-37行，更新设置sensor的亮度的Gain值

其中m_u4UpdateShutterValue 和m_u4UpdateGainValue的值都是在前面的UpdateSensorISPParams函数中设置

看下setSensorGain函数的实现

```
1   MRESULT
2   AAASensorMgr::
3   setSensorGain(MINT32 i4SensorDev, MUINT32 a_u4SensorGain)
4   {
5       MINT32 ret = S_AAA_SENSOR_MGR_OK;
6
7       ......
8
9       // Set sensor gain
10      if(i4SensorDev == ESensorDev_Main) {
11          ret = m_pHalSensorObj->sendCommand(NSCam::SENSOR_DEV_MAIN, SENSOR_CMD_SET_SENSOR_GAIN, (MUINTPTR)&
12      } else if(i4SensorDev == ESensorDev_Sub) {
13          ......
14      }
15
16      ......
17
18      return (ret);
19  }
```

```
1   MINT HalSensor::sendCommand(
2       MUINT sensorDevIdx,
3       MUINTPTR cmd,
4       MUINTPTR arg1,
5       MUINTPTR arg2,
6       MUINTPTR arg3)
7   {
8       switch (cmd) {
9       case SENSOR_CMD_SET_SENSOR_GAIN:
10          cmdId = CMD_SENSOR_SET_SENSOR_GAIN;
11          pSensorDrv->sendCommand((SENSOR_DEV_ENUM)sensorDevId,cmdId, arg1);
12          break;
13      ......
14
15      return ret;
16  }
```

```
MINT32
ImgSensorDrv::sendCommand(
        SENSOR_DEV_ENUM sensorDevId,
        MUINT32 cmd,
        MUINTPTR arg1,
        MUINTPTR arg2,
        MUINTPTR arg3
)
{
    switch (cmd) {
    case CMD_SENSOR_SET_SENSOR_GAIN:
        FeatureId = SENSOR_FEATURE_SET_GAIN;
        FeaturePara[0] = *parg1; //from 10b to 6b base
        FeaturePara[0] >>= 4;
        FeatureParaLen = sizeof(MUINT64);
        pFeaturePara =  (MUINT8*)FeaturePara;
        break;

    ......

    err= featureControl((CAMERA_DUAL_CAMERA_SENSOR_ENUM)sensorDevId, FeatureId,  (MUINT8*)pFeaturePara,(MU
    if (err < 0) {
        LOG_ERR("[sendCommand] Err-ctrlCode (%s) \n", strerror(errno));
        return -errno;
    }

    ......

    return err;
}
```

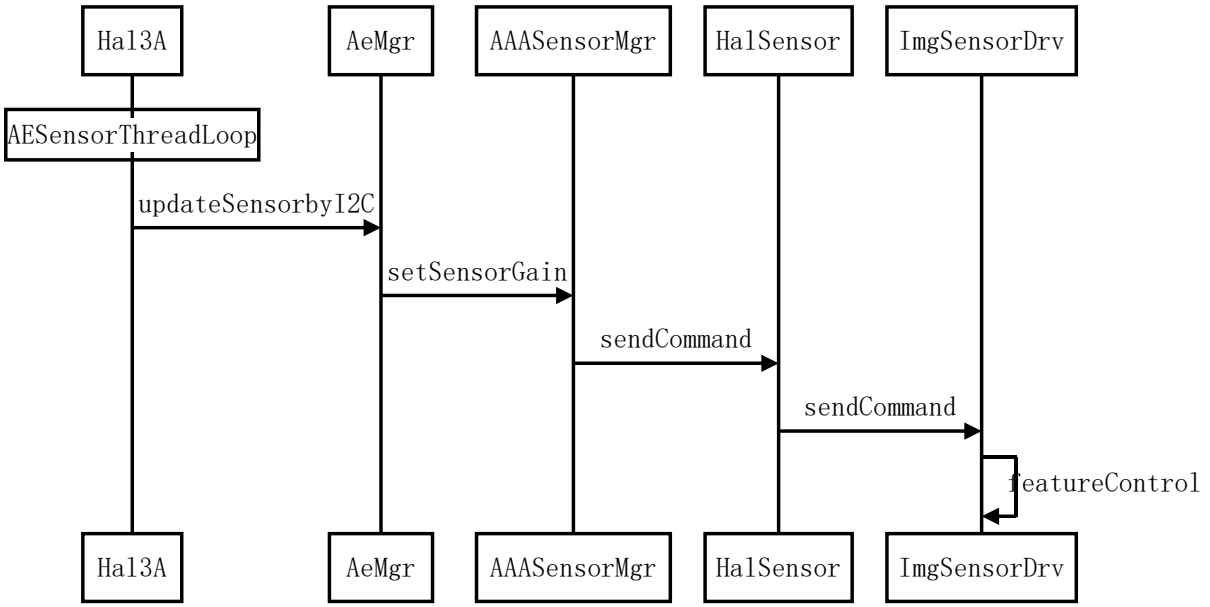
```
MINT32
ImgSensorDrv::featureControl(
    CAMERA_DUAL_CAMERA_SENSOR_ENUM InvokeCamera,
    ACDK_SENSOR_FEATURE_ENUM FeatureId,
    MUINT8 *pFeaturePara,
    MUINT32 *pFeatureParaLen
)
{
    ACDK_SENSOR_FEATURECONTROL_STRUCT featureCtrl;
    MINT32 err = SENSOR_NO_ERROR;

    ......
    featureCtrl.InvokeCamera = InvokeCamera;
    featureCtrl.FeatureId = FeatureId;
    featureCtrl.pFeaturePara = pFeaturePara;
    featureCtrl.pFeatureParaLen = pFeatureParaLen;

    err = ioctl(m_fdSensor, KDIMGSENSORIOC_X_FEATURECONCTROL , &featureCtrl);
    if (err < 0) {
        LOG_ERR("[featureControl] Err-ctrlCode (%s) \n", strerror(errno));
        return -errno;
    }

    return err;
}
```

最后调用到imgsensor_drv.cpp的featureControl函数，通过ioctl进入到kernel层，kernel层对应的sensor驱动会通过i2c设置sensor的寄存器



6. 总结

3A的初始化在DefaultCam1Device的onInit函数里面开始，主要就是初始化3A的状态管理并切换到init状态，创建了onThreadLoop 和AFThreadFunc两个线程。onThreadLoop是3A主线程，负责接收处理命令；AFThreadFunc负责实时更新AF参数

接收到PASS1_START_ISP事件之后，Hal3A会再创建一个AESensorThreadLoop线程负责实时更新sensor的AE参数，同时还会对AWB、AE、AF进行初始化，最后将3A状态切换到CameraPreview状态。

Pass1Node每deque一帧数据就会发出PASS1_EOF事件来更新3A，Hal3A接收到消息之后会计算ISP相关的参数并将得到的参数设置到Isp Tuning里面

当需要更新Sensor的参数时ISP会产生一个中断，而AFThreadFunc和AESensorThreadLoop则通过一个死循环不断去捕获中断，捕获到中断之后会让kernel层对应的驱动通过i2c设置相关的寄存器