Android 5.0 Camera系统源码分析(4)：Camera预览流程数据流

标签： MTK Android Camera Preview 调用流程 2016-10-17 14:46 919人阅读 评论(0) 收藏 举报

1. 前言

上一篇讲了怎么让Camera进入预览模式，提到了DisplayClient负责显示图像数据，而CamAdapter负责提供图像数据，这里主要记录了CamAdapter怎么获取图像，然后DisplayClient怎么将图像显示在屏幕上。

2. DisplayClient

上一篇提到在setPreviewWindow的时候会构造并初始化DisplayClient，之前没有仔细分析，现在来看看

```
bool
DisplayClient::
init()
{
    bool ret = false;

    ret =   createDisplayThread()
        &&  createImgBufQueue();

    return  ret;
}
```

创建了一个显示线程和一个ImgBuf队列，看下这两个函数的具体实现

```
bool
DisplayClient::
createDisplayThread()
{
    bool    ret = false;
    status_t status = OK;

    mpDisplayThread = IDisplayThread::createInstance(this);
    if  ( mpDisplayThread == 0 ||  OK != (status = mpDisplayThread->run()) )
    {
        ......
    }

    ret = true;
lbExit:
    return  ret;
}
```

```
bool
DisplayClient::
createImgBufQueue()
{
    bool ret = false;

    mpImgBufQueue = new ImgBufQueue(IImgBufProvider::eID_DISPLAY, "CameraDisplay@ImgBufQue");
    if  ( mpImgBufQueue == 0 )
    {
        MY_LOGE("Fail to new ImgBufQueue");
        goto lbExit;
    }
    ......
    ret = true;
lbExit:
    MY_LOGD("-");
```

```
    return  ret;
}
```

ImgBufQueue暂时放一边，createDisplayThread创建了DisplayThread，作为线程关注的重点当然是threadLoop，所以接着看DisplayThread的threadLoop函数

```
bool
DisplayThread::
threadLoop()
{
    Command cmd;
    if  ( getCommand(cmd) )
    {
        switch  (cmd.eId)
        {
        case Command::eID_EXIT:
            MY_LOGD("Command::%s", cmd.name());
            break;

        case Command::eID_WAKEUP:
        default:
            if  ( mpThreadHandler != 0 )
            {
                mpThreadHandler->onThreadLoop(cmd);
            }
            break;
        }
    }

    return  true;
}
```

DisplayThread将接收WAKEUP命令，然后做出响应。那么由谁来发这个WAKEUP命令呢，就在上一篇提到的enableDisplayClient函数里面发送。这里的mpThreadHandler 指的是DisplayClient，也就是在接收到WAKEUP命令后，将回调DisplayClient的onThreadLoop函数

```
bool
DisplayClient::
onThreadLoop(Command const& rCmd)
{
    //  (0) lock Processor.
    sp<IImgBufQueue> pImgBufQueue;
    {
        Mutex::Autolock _l(mModuleMtx);
        pImgBufQueue = mpImgBufQueue;
        if  ( pImgBufQueue == 0 || ! isDisplayEnabled() )
        {
            MY_LOGW("pImgBufQueue.get(%p), isDisplayEnabled(%d)", pImgBufQueue.get(), isDisplayEnabled());
            return  true;
        }
    }

    //  (1) Prepare all TODO buffers.
    if  ( ! prepareAllTodoBuffers(pImgBufQueue) )
    {
        return  true;
    }

    //  (2) Start
    if  ( ! pImgBufQueue->startProcessor() )
    {
        return  true;
    }

    //  (3) Do until disabled.
    while(1)
    {
        //  (.1)
        waitAndHandleReturnBuffers(pImgBufQueue);

        //  (.2) break if disabled.
        if  ( ! isDisplayEnabled() )
        {
            MY_LOGI("Display disabled");

```

```
            break;
        }

        //  (.3) re-prepare all TODO buffers, if possible,
        //  since some DONE/CANCEL buffers return.
        prepareAllTodoBuffers(pImgBufQueue);
    }

    ......

    return  true;
}
```

先分析步骤(1)准备好接收数据的buffers

```
/*******************************************************************************
*   dequePrvOps() -> enqueProcessor() & enque Buf List
*******************************************************************************/
bool
DisplayClient::
prepareAllTodoBuffers(sp<IImgBufQueue>const& rpBufQueue)
{
    bool ret = false;

    while   ( mStreamBufList.size() < (size_t)mi4MaxImgBufCount )
    {
        if  ( ! prepareOneTodoBuffer(rpBufQueue) )
        {
            break;
        }
    }

    return ret;
}
```

```
bool
DisplayClient::
prepareOneTodoBuffer(sp<IImgBufQueue>const& rpBufQueue)
{
    bool ret = false;

    ......

    //  (2) deque it from PrvOps
    sp<StreamImgBuf> pStreamImgBuf;
    if  ( ! dequePrvOps(pStreamImgBuf) )
    {
        goto lbExit;
    }

    //  (3) enque it into Processor
    ret = rpBufQueue->enqueProcessor(
        ImgBufQueNode(pStreamImgBuf, ImgBufQueNode::eSTATUS_TODO)
    );

    //  (4) enque it into List & increment the list size.
    mStreamBufList.push_back(pStreamImgBuf);

    ret = true;
lbExit:
    MY_LOGD_IF((2<=miLogLevel), "- ret(%d)", ret);
    return ret;
}
```

这里的ImgBufQueue就是DisplayClient初始化的时候创建的那个ImgBufQueue，里有两个Buf队列，mTodoImgBufQue和mDoneImgBufQue。prepareOneTodoBuffer函数做的事情就是从dequePrvOps 函数deque出StreamImgBuf，并用它生成ImgBufQueNode，把ImgBufQueNode的标志位设eSTATUS_TODO后调用ImgBufQueue的enqueProcessor函数把所有的ImgBufQueNode都放入到mTodoImgBufQue做接收数据的准备。看下dequePrvOps和enqueProcessor的实现

```
bool
```

```
DisplayClient::
dequePrvOps(sp<StreamImgBuf>& rpImgBuf)
{
    //  [1] dequeue_buffer
    err = mpStreamOps->dequeue_buffer(mpStreamOps, &phBuffer, &stride);
    //  [2] lock buffers
    err = mpStreamOps->lock_buffer(mpStreamOps, phBuffer);
    ......
    //  [5] Setup the output to return.
    rpImgBuf = new StreamImgBuf(mpStreamImgInfo, stride, address, phBuffer, fdIon);

    ret = true;
lbExit:
    return  ret;
}
```

值得一提的是mpStreamOps，它就是上一篇不断提到的mHalPreviewWindow.nw，调用它的dequeue_buffer函数就相当于从Surface中dequeue一个buffer出来，将buffer填满后通过调用enqueue_buffer函数将buffer传给Surface，这样图像就得以显示。

```

bool
ImgBufQueue::
enqueProcessor(ImgBufQueNode const& rNode)
{
    ......
    mTodoImgBufQue.push_back(rNode);
    return  true;
}
```

把所有的ImgBufQueNode都放入到mTodoImgBufQue做接收数据的准备。

回到onThreadLoop函数，步骤(3)进入死循环，不断调用waitAndHandleReturnBuffers函数来接收处理buffer，同时调用 prepareAllTodoBuffers函数来将处理完的buffer重新放回 mTodoImgBufQue，接着看如何接收处理buffer

```

bool
DisplayClient::
waitAndHandleReturnBuffers(sp<IImgBufQueue>const& rpBufQueue)
{
    bool ret = false;
    Vector<ImgBufQueNode> vQueNode;
    ......
    //  (1) deque buffers from processor.
    rpBufQueue->dequeProcessor(vQueNode);

    //  (2) handle buffers dequed from processor.
    ret = handleReturnBuffers(vQueNode);

lbExit:
    return ret;
}
```

在此处调用ImgBufQueue的dequeProcessor()等待通知并接收数据。然后再调用handleReturnBuffers函数将数据发给Surface

```

ImgBufQueue::
dequeProcessor(Vector<ImgBufQueNode>& rvNode)
{
    bool ret = false;

    while   ( mDoneImgBufQue.empty() && mbIsProcessorRunning )
    {
        status_t status = mDoneImgBufQueCond.wait(mDoneImgBufQueMtx);
    }

    if  ( ! mDoneImgBufQue.empty() )
    {
        //  If the queue is not empty, deque all buffers from the queue.
        ret = true;
        rvNode = mDoneImgBufQue;
        mDoneImgBufQue.clear();
    }

```

```
    return ret;
}
```

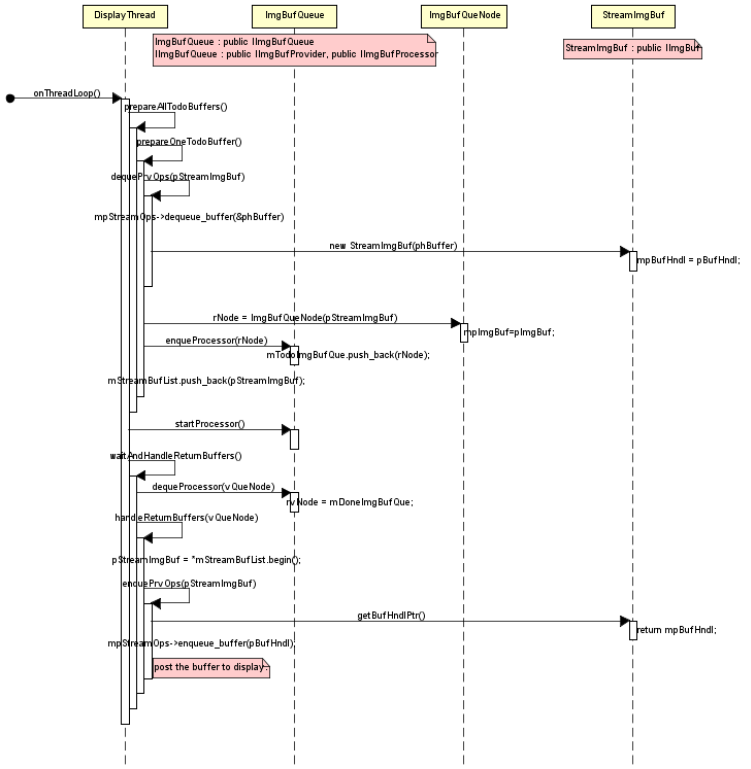
通过mDoneImgBufQueCond.wait(mDoneImgBufQueMtx)等待通知，收到通知后，从mDoneImgBufQue取出所有的ImgBufQueNode，这时候ImgBufQueNode里面已经包含了图像数据。

```
bool
DisplayClient::
handleReturnBuffers(Vector<ImgBufQueNode>const& rvQueNode)
{
    for (int32_t i = 0; i < queSize; i++)
    {
        sp<IImgBuf>const&       rpQueImgBuf = rvQueNode[i].getImgBuf(); //  ImgBuf in Queue.
        sp<StreamImgBuf>const pStreamImgBuf = *mStreamBufList.begin();  //  ImgBuf in List.

        //  (1)  Check valid pointers to image buffers in Queue & List
        if  ( rpQueImgBuf == 0 || pStreamImgBuf == 0 )
        {
            ......
            continue;
        }

        //  (2)  Check the equality of image buffers between Queue & List.
        if  ( rpQueImgBuf->getVirAddr() != pStreamImgBuf->getVirAddr() )
        {
            ......
            continue;
        }

        //  (3)  Every check is ok. Now remove the node from the list.
        mStreamBufList.erase(mStreamBufList.begin());

        //  (4)  enquePrvOps/cancelPrvOps
        if  ( i == idxToDisp ) {
            ......
            enquePrvOps(pStreamImgBuf);
        }
    }

    return  true;
}
```

for循环里面通过 enquePrvOps函数将一个个StreamImgBuf发给Surface

```
void
DisplayClient::
enquePrvOps(sp<StreamImgBuf>const& rpImgBuf)
{
    ......
    //  [5] unlocks and post the buffer to display.
    err = mpStreamOps->enqueue_buffer(mpStreamOps, rpImgBuf->getBufHndlPtr());
}
```

就如前面提到的，将buffer填满后需要调用enqueue_buffer函数，这样图像就已经发往Surface。



那么图像数据从哪里来呢，DisplayClient一共维护了两个队列 mTodoImgBufQue和mDoneImgBufQue，也就是说肯定在某个地方有人从mTodoImgBufQue deque了一个ImgBufQueNode，将它填满后enque到mDoneImgBufQue里面并发送通知告诉DisplayClient数据已经准备好

3. Pass1Node

上一篇提到由CamAdapter提供图像数据给DisplayClient。它的大部分工作分别由各个CamNode完成，其中Pass1Node负责和sensor driver打交道，最初的图像数据就是由它来获取，之前已经看过它的onInit和onStart函数，现在来看它的threadLoopUpdate函数

```
MBOOL
Pass1NodeImpl::
threadLoopUpdate()
{
    MBOOL ret = MTRUE;
    //
    if( keepLooping() )
    {
        // deque
        ret = dequeLoop();

        // try to keep ring buffer running
        enqueBuffer();
    }
    else
    {
        ret = stopHw();
        syncWithThread();
    }
    //
    return ret;
}
```

这里值得关注的只有dequeLoop函数

```
MBOOL
Pass1NodeImpl::
dequeLoop()
{
    //FUNC_START;
    MBOOL ret = MTRUE;

```

```
9     //prepare to deque
10    QBufInfo dequeBufInfo;
11    dequeBufInfo.mvOut.reserve(2);
12    if( mpRingImgo ) {
13        BufInfo OutBuf(mpRingImgo->getPortID(), 0);
14        dequeBufInfo.mvOut.push_back(OutBuf);
15    }
16    if( mpRingRrzo ) {
17        BufInfo OutBuf(mpRingRrzo->getPortID(), 0);
18        dequeBufInfo.mvOut.push_back(OutBuf);
19    }
20
21    for(MUINT32 i=0; i<2; i++)
22    {
23        MY_LOGD("frame %d: deque+", muFrameCnt);
24        ret = mpCamIO->deque(dequeBufInfo);
25        MY_LOGD("frame %d: deque-,%d", muFrameCnt, ret);
26        ......
27    }
28
29    ......
30
31    handleNotify(
32            PASS1_EOF,
33            newMagicNum,
34            muSensorDelay == 0 ? dequeMagicNum : MAGIC_NUM_INVALID);
35
36    configFrame(newMagicNum);
37
38    ......
39
40    vector<BufInfo>::const_iterator iter;
41    for( iter = dequeBufInfo.mvOut.begin(); iter != dequeBufInfo.mvOut.end(); iter++ )
42    {
43        mpIspSyncCtrlHw->addPass1Info(
44                iter->mMetaData.mMagicNum_hal,
45                iter->mBuffer,
46                iter->mMetaData,
47                iter->mPortID == PORT_RRZO);
48        ......
49        ret = ret && handlePostBuffer( mapToNodeDataType(iter->mPortID, bIsDynamicPureRaw), (MUINTPTR)iter->mBuffer, iter->mMetaData.mMagi
50    }
51
52    //FUNC_END;
53    return ret;
}
```

第23行，通过mpCamIO->deque取出一帧数据

第30-33行，发送 PASS1_EOF消息，其它的CamNode接收到消息后做相应的处理，例如更新3A

第35行， configFrame不知道它在做什么，有待研究

第42-46行，将Pass1的deque信息加入到IspSyncCtrl

第48行，将deque到的数据post到Pass2Node(其实是post到DefaultCtrlNode，再由它post给Pass2Node)

这里CamIO指的是NormalPipe，在Pass1Node的onInit函数里创建，看下它如何deque数据

```
1
2   MBOOL
3   NormalPipe::deque(QBufInfo& rQBuf, MUINT32 u4TimeoutMs)
4   {
5       for (MUINT32 ii=0 ; ii<port_cnt ; ii++ ) {
6           if (MFALSE == mpCamIOPipe->dequeOutBuf(portID, rQTSBufInfo) ) {
7               ......
8           }
9
10          if ( rQTSBufInfo.vBufInfo.size() >= 1 )  {
11              ......
12              buff.mPortID = rQBuf.mvOut.at(ii).mPortID;
13              buff.mBuffer = pframe;
14              buff.mMetaData = result;
15              buff.mSize = rQTSBufInfo.vBufInfo.at(idx).u4BufSize[0];
16              buff.mVa = rQTSBufInfo.vBufInfo.at(idx).u4BufVA[0];
```

```
            buff.mPa = rQTSBufInfo.vBufInfo.at(idx).u4BufPA[0];
            rQBuf.mvOut.at(ii) = buff;
        }
    }
    return ret;
}
```

```
MBOOL
CamIOPipe::
dequeOutBuf(PortID const portID, QTimeStampBufInfo& rQBufInfo, MUINT32 const u4TimeoutMs /*= 0xFFFFFFFF*/)
{
    MUINT32 dmaChannel = 0;
    stISP_FILLED_BUF_LIST bufInfo;
    ISP_BUF_INFO_L  bufList;

    ......

    bufInfo.pBufList = &bufList;
    if ( 0 != this->m_CamPathPass1.dequeueBuf( dmaChannel,bufInfo) ) {
        ......
    }

    rQBufInfo.vBufInfo.resize(bufList.size());
    for ( MINT32 i = 0; i < (MINT32)rQBufInfo.vBufInfo.size() ; i++) {
        rQBufInfo.vBufInfo[i].memID[0]          = bufList.front().memID;
        rQBufInfo.vBufInfo[i].u4BufSize[0]      = bufList.front().size;
        rQBufInfo.vBufInfo[i].u4BufVA[0]        = bufList.front().base_vAddr;
        rQBufInfo.vBufInfo[i].u4BufPA[0]        = bufList.front().base_pAddr;
        rQBufInfo.vBufInfo[i].i4TimeStamp_sec   = bufList.front().timeStampS;
        rQBufInfo.vBufInfo[i].i4TimeStamp_us    = bufList.front().timeStampUs;
        rQBufInfo.vBufInfo[i].img_w             = bufList.front().img_w;
        rQBufInfo.vBufInfo[i].img_h             = bufList.front().img_h;
        rQBufInfo.vBufInfo[i].img_stride        = bufList.front().img_stride;
        rQBufInfo.vBufInfo[i].img_fmt           = bufList.front().img_fmt;
        ......
        bufList.pop_front();
    }
    return  MTRUE;
}
```

```
int CamPathPass1::dequeueBuf( MUINT32 dmaChannel ,stISP_FILLED_BUF_LIST& bufInfo )
{
    int ret = 0;
    Mutex   *_localVar;

    //check if there is already filled buffer
    if ( MFALSE == this->ispBufCtrl.waitBufReady(dmaChannel) ) {
        ......
    }
    //move FILLED buffer from hw to sw list
    if ( eIspRetStatus_Success != this->ispBufCtrl.dequeueHwBuf( dmaChannel, bufInfo ) ) {
        ......
    }

    return ret;
}
```

第7行，当buffer准备好时ISP会产生一个中断，而这里将通过ioctl去等待获取这个中断

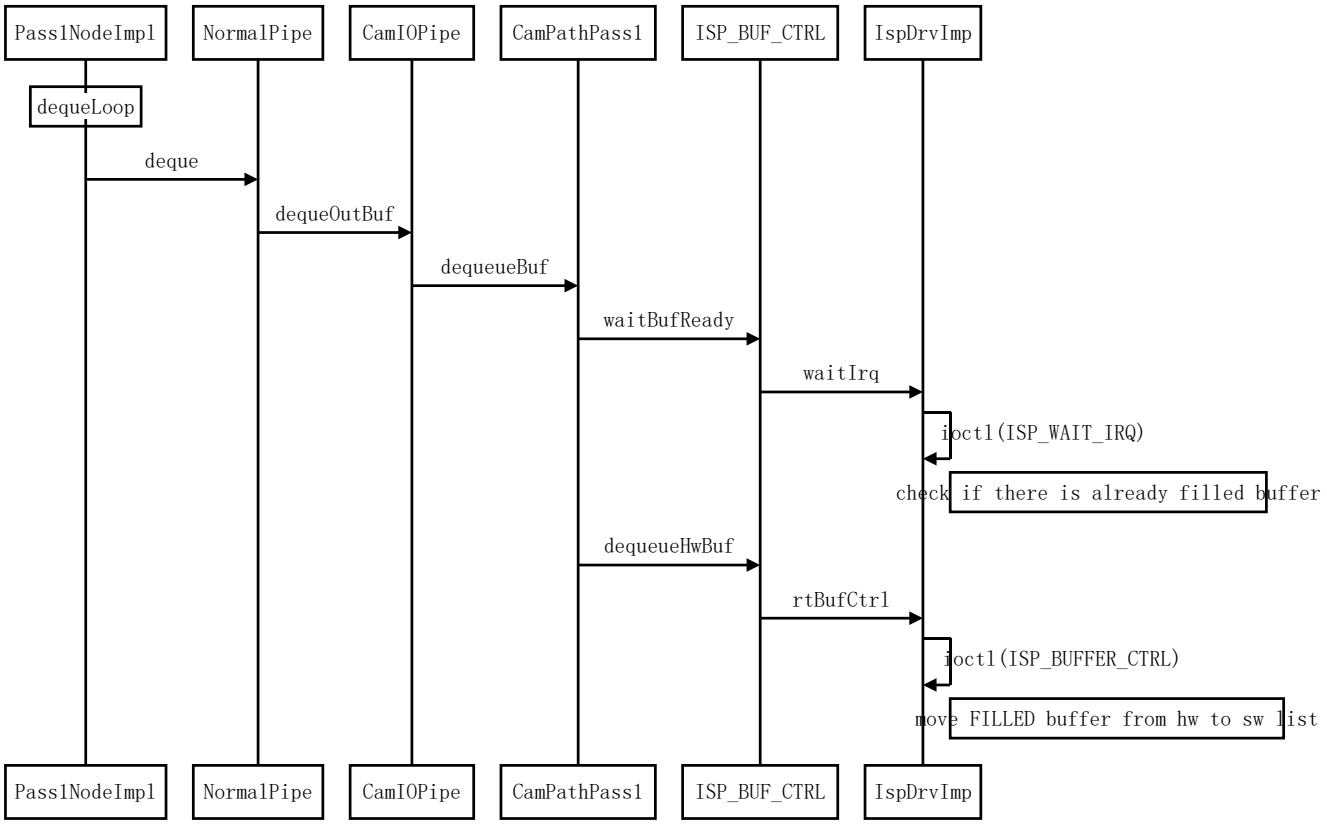
第11行，从底层获取已经填满的buffer

```
EIspRetStatus
ISP_BUF_CTRL::
dequeueHwBuf( MUINT32 dmaChannel, stISP_FILLED_BUF_LIST& bufList )
{
    if ( ISP_PASS1   == this->path || \
         ISP_PASS1_D == this->path_D || \
         ISP_PASS1_CAMSV   == this->path || \
         ISP_PASS1_CAMSV_D == this->path_D
        ) {
            //deque filled buffer
            buf_ctrl.ctrl = ISP_RT_BUF_CTRL_DEQUE;
            buf_ctrl.buf_id = (_isp_dma_enum_)rt_dma;
```

```
            buf_ctrl.data_ptr = 0;
            buf_ctrl.pExtend = (unsigned char*)&deque_buf;

            if ( MTRUE != this->m_pIspDrvShell->m_pPhyIspDrv_bak->rtBufCtrl((void*)&buf_ctrl) ) {
                ISP_FUNC_ERR("ERROR:rtBufCtrl");
                ret = eIspRetStatus_Failed;
                goto EXIT;
            }
        ......
    } else { // Pass2
        ......
    }

EXIT:
    return ret;
}
```

```
MBOOL IspDrvImp::rtBufCtrl(void *pBuf_ctrl)
{
    MINT32 Ret;

    Ret = ioctl(mFd,ISP_BUFFER_CTRL,pBuf_ctrl);

    return MTRUE;
}
```

mFd是通过open(“/dev/camera-isp”，O_RDWR)得到的，而这里通过ioctl获取到已经填满的buffer的地址。

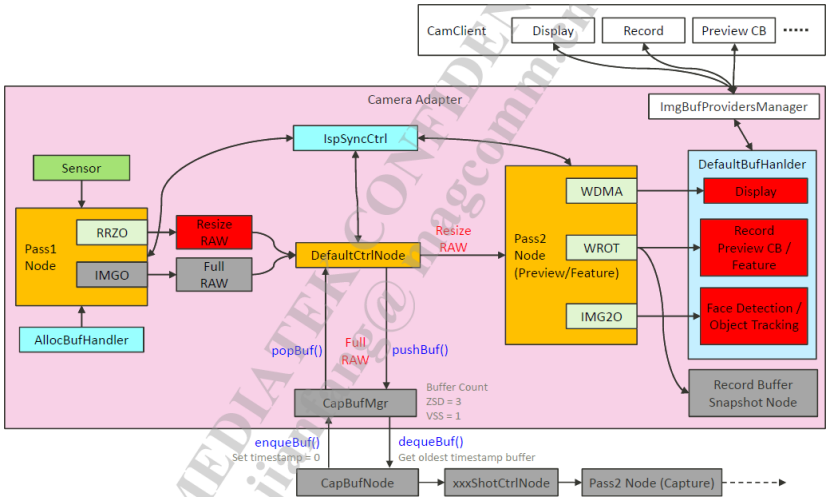


到这里我们已经获取到了一帧图像，但还是不知道是谁把buffer放到DisplayClient的mDoneImgBufQue里面去

4. Pass2Node

回到Pass1Node的dequeLoop函数，最后一个步骤handlePostBuffer函数。上一篇提到在CamAdapter的onHandleStartPreview函数里面，通过connectData把Pass1Node和DefaultCtrlNode连接起来，把Pass2Node和DefaultCtrlNode连接起来

```
mpCamGraph->connectData(PASS1_RESIZEDRAW,CONTROL_RESIZEDRAW,mpPass1Node,mpDefaultCtrlNode);
mpCamGraph->connectData(CONTROL_PRV_SRC,PASS2_PRV_SRC,mpDefaultCtrlNode,mpPass2Node);
```



所以Pass1Node的handlePostBuffer函数会先把buffer post到DefaultCtrlNode，DefaultCtrlNode接收到之后再将它post给Pass2Node，这里直接看Pass2Node的onPostBuffer函数

4.1 onPostBuffer函数分析

```
MBOOL
Pass2NodeImpl::
onPostBuffer(MUINT32 const data, MUINTPTR const buf, MUINT32 const ext)
{
    if( pushBuf(data, (IImageBuffer*)buf, ext) )
    {
        // no thing
    }
}
```

```
MBOOL
Pass2NodeImpl::
pushBuf(MUINT32 const data, IImageBuffer* const buf, MUINT32 const ext)
{
    PostBufInfo postBufData = {data, buf, ext};
    mlPostBufData.push_back(postBufData);

    muPostFrameCnt++;

    if( isReadyToEnque() )
    {
        triggerLoop();
    }

    return MTRUE;
}
```

保存好buffer之后调用triggerLoop函数，triggerLoop会给自身的线程发送update命令，然后Pass2Node的threadLoopUpdate函数就会被调用

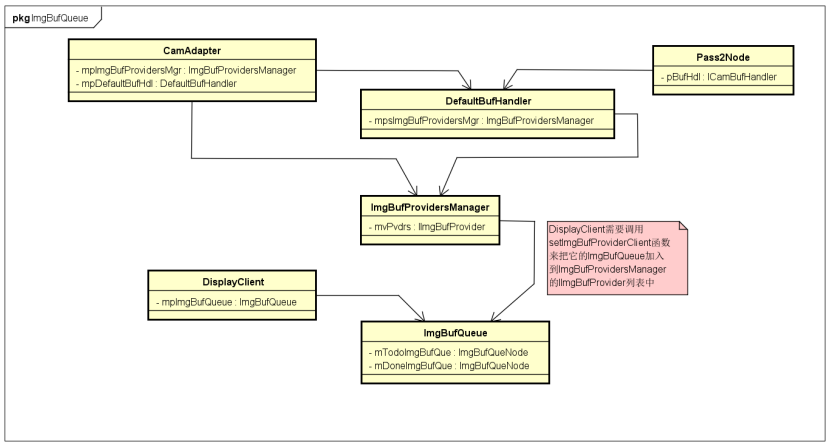
```
MBOOL
Pass2NodeImpl::
threadLoopUpdate()
{
    MBOOL ret = MTRUE;

    ret = enquePass2(MTRUE);

    return ret;
}
```

```
MBOOL
Pass2NodeImpl::
enquePass2(MBOOL const doCallback)
{
    QParams enqueParams;

```

```
7       vector<p2data> vP2data;
8
9       if( !getPass2Buffer(vP2data) )
10      {
11          // no dst buffers
12          return MTRUE;
13      }
14      ......
15      configFeature();
16
17      if( !mpIspSyncCtrlHw->lockHw(IspSyncControlHw::HW_PASS2) )
18      {
19          ......
20      }
21
22      enqueParams.mpfnCallback = pass2CbFunc;
23      enqueParams.mpCookie = this;
24      if( !mpPostProcPipe->enque(enqueParams) )
25      {
26          ......
27      }
28
29      return MTRUE;
    }
```

第8行，通过DefaultBufHandler从mTodoImgBufQue取出buffer

第23行，enque目标buffer到IHalPostProcPipe，至于IHalPostProcPipe做了些什么事情我也不知道，可能是图像缩放之类的工作，有待研究。IHalPostProcPipe处理完之后会回调pass2CbFunc函数。Pass2CbFunc会把处理过的buffer通过DefaultBufHandler放回mDoneImgBufQue里面。



DefaultBufHandler的作用是管理所有的CamClient的buffer队列，例如DisplayClient。DisplayClient需要调用setImgBufProviderClient函数来把它的ImgBufQueue加入到ImgBufProvidersManager的IImgBufProvider列表中。从类图可以看到，Pass2Node只要获取到DefaultBufHandler，就能拿到DisplayClient的ImgBufQueue

4.2 getPass2Buffer函数分析

```
1   MBOOL
2   PrvPass2::
3   getPass2Buffer(vector<p2data>& vP2data)
4   {
5       MBOOL haveDst = MFALSE;
6       // src
7       {
8           Mutex::Autolock lock(mLock);
9           p2data one;
10          MUINT32 count = 0;
11          //
12          if( mlPostBufData.size() < muMultiFrameNum )
13          {
14              ......
15          }
16          //
17          list<PostBufInfo>::iterator iter = mlPostBufData.begin();
18          while( iter != mlPostBufData.end() )
```

```
19          {
20              one.src = *iter;
21              iter = mlPostBufData.erase(iter);
22              //
23              vP2data.push_back(one);
24              count++;
25              //
26              if(count == muMultiFrameNum)
27              {
28                  break;
29              }
30          }
31      }
32      // dst
33      {
34          MBOOL bDequeDisplay = MTRUE;
35
36          vector<p2data>::iterator pData = vP2data.begin();
37          while( pData != vP2data.end() )
38          {
39              for(MUINT32 i = 0; i < MAX_DST_PORT_NUM; i++)
40              {
41                  MBOOL ret;
42                  ImgRequest outRequest;
43                  //
44                  ......
45                  //
46                  ret = getDstBuffer(
47                          muDequeOrder[i],
48                          &outRequest);
49                  //
50                  if(ret)
51                  {
52                      haveDst = MTRUE;
53
54                      if(muDequeOrder[i] == PASS2_PRV_DST_0)
55                      {
56                          bDequeDisplay = MFALSE;
57                      }
58                      //
59                      pData->vDstReq.push_back(outRequest);
60                      pData->vDstData.push_back(muDequeOrder[i]);
61                  }
62              }
63              ......
64          }
65      }
66  }
```

第6-31行，把之前保存在mlPostBufData里的图像数据取出来并保存在p2data中

第46-48行，从DefaultBufHandler取出所有CamClient的buffer

第50-61行，把从DefaultBufHandler获取到的目标buffer一起放到p2data中

```
1
2   MBOOL
3   Pass2NodeImpl::
4   getDstBuffer(
5       MUINT32         nodeData,
6       ImgRequest*     pImgReq)
7   {
8       MBOOL ret = MFALSE;
9       ICamBufHandler* pBufHdl = getBufferHandler(nodeData);
10      if(pBufHdl && pBufHdl->dequeBuffer(nodeData, pImgReq))
11      {
12          ret = MTRUE;
13      }
14      return ret;
    }
```

```
1
2   MBOOL
3   DefaultBufHandlerImpl::
4   dequeBuffer(MUINT32 const data, ImgRequest * pImgReq)
    {
```

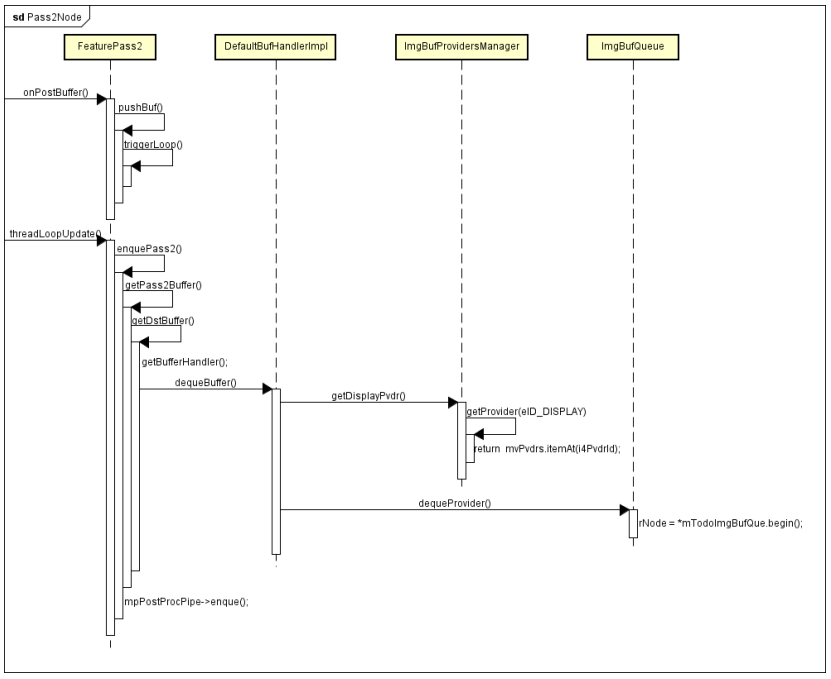
```
5       ......
6       sp<IImgBufProvider> bufProvider = NULL;
7       switch((*iterMapPort).bufType)
8       {
9           case eBuf_Disp:
10          {
11              bufProvider =  mspImgBufProvidersMgr->getDisplayPvdr();
12              pImgReq->mUsage = NSIoPipe::EPortCapbility_Disp;
13              break;
14          }
15          ......
16      }
17      ......
18      if(bufProvider->dequeProvider(node))
19      {
20          node.setCookieDE((*iterMapPort).bufType);
21          mvBufQueNode[bufQueIdx].push_back(node);
22          isDequeProvider = MTRUE;
23          break;
24      }
25      ......
26  }
```

第11行，这里获取到的就是DisplayClient的ImgBufQueue，它继承了IImgBufProvider类

第18行，获取ImgBufQueNode

```
1   bool
2   ImgBufQueue::
3   dequeProvider(ImgBufQueNode& rNode)
4   {
5       bool ret = false;
6       //
7       Mutex::Autolock _lock(mTodoImgBufQueMtx);
8       //
9       if  ( ! mTodoImgBufQue.empty() )
10      {
11          //  If the queue is not empty, take the first buffer from the queue.
12          ret = true;
13          rNode = *mTodoImgBufQue.begin();
14          mTodoImgBufQue.erase(mTodoImgBufQue.begin());
15      }
16      //
17      return ret;
18  }
```

第13行，可以看到deque最后是从mTodoImgBufQue里面取出buffer



4.3 pass2CbFunc函数分析

之前提到过IHalPostProcPipe处理完之后会回调pass2CbFunc函数。

```
MVOID
Pass2NodeImpl::
pass2CbFunc(QParams& rParams)
{
     Pass2NodeImpl* pPass2NodeImpl = (Pass2NodeImpl*)(rParams.mpCookie);
     pPass2NodeImpl->handleP2Done(rParams);
}
```

```
MBOOL
Pass2NodeImpl::
handleP2Done(QParams& rParams)
{
    ......
    //
    if( !mpIspSyncCtrlHw->unlockHw(IspSyncControlHw::HW_PASS2) )
    {
        MY_LOGE("isp sync unlock pass2 failed");
        goto lbExit;
    }

    for( iterIn = rParams.mvIn.begin() ; iterIn != rParams.mvIn.end() ; iterIn++ )
    {
        MUINT32 nodeDataType = mapToNodeDataType( iterIn->mPortID );
        handleReturnBuffer( nodeDataType, (MUINTPTR)iterIn->mBuffer, 0 );
    }

    vpDstBufAddr.clear();
    for( iterOut = rParams.mvOut.begin() ; iterOut != rParams.mvOut.end() ; iterOut++ )
    {
        MBOOL bFind = MFALSE;

        ......

        if(!bFind)
        {
            MUINT32 nodeDataType = mapToNodeDataType( iterOut->mPortID );
            handlePostBuffer( nodeDataType, (MUINTPTR)iterOut->mBuffer, 0 );
            vpDstBufAddr.push_back(iterOut->mBuffer);
        }
    }

    return ret;
}
```

第13-17行，之前Pass1Node调用handlePostBuffer把buffer传到Pass2Node，而现在调用handleReturnBuffer则会把buffer返回给Pass1Node，由Pass1Node的onReturnBuffer函数接收处理

第29行，再次调用handlePostBuffer函数，这里由于没有连接其它的CamNode，所以会回调Pass2Node的onReturnBuffer函数

Pass1Node的onReturnBuffer函数就是把处理完的buffer放回ring buffer里面，这里不再分析，来看看Pass2Node的onReturnBuffer函数

```
MBOOL
Pass2NodeImpl::
onReturnBuffer(MUINT32 const data, MUINTPTR const buf, MUINT32 const ext)
{

    ICamBufHandler* pBufHdl = getBufferHandler(data);
    ......
    MBOOL ret = pBufHdl->enqueBuffer(data, (IImageBuffer*)buf);
    ......
    return MTRUE;
}
```

```
MBOOL
DefaultBufHandlerImpl::
enqueBuffer(MUINT32 const data, IImageBuffer const * pImageBuffer)
```

```
{
    ......
    switch(keepImgBufQueNode.getCookieDE())
    {
        case eBuf_Disp:
        {
            bufProvider = mspImgBufProvidersMgr->getDisplayPvdr();
            break;
        }
        ......
    }

    ......
    bufProvider->enqueProvider(keepImgBufQueNode);
    ......
}
```

```
bool
ImgBufQueue::
enqueProvider(ImgBufQueNode const& rNode)
{
    ......
    Mutex::Autolock _lock(mDoneImgBufQueMtx);

    mDoneImgBufQue.push_back(rNode);
    mDoneImgBufQueCond.broadcast();

    return  true;
}
```

第8行，流程上和之前的deque差不多，可以看到enque最终将buffer放到mDoneImgBufQue里面

第9行，准备好之后发送广播通知DisplayClient

5. 总结

DisplayClient准备好buffer放到mTodoImgBufQue里面。

Pass1Node从底层deque一帧数据，然后将数据post给DefaultCtrlNode，DefaultCtrlNode又将数据post给Pass2Node。

Pass2Node保存好buffer之后会触发threadLoopUpdate，threadLoopUpdate通过DefaultBufHandler从mTodoImgBufQue取出buffer，再将buffer交给IHalPostProcPipe处理，当IHalPostProcPipe处理完之后会回调Pass2CbFunc函数，Pass2CbFunc通过DefaultBufHandler把buffer放回mDoneImgBufQue里面。

最后DisplayClient不断从mDoneImgBufQue里面取出已经处理好的buffer送到Surface里面