Android 5.0 Camera系统源码分析(3)：Camera预览流程控制流

标签： mtk Android Camera 源码分析 调用流程 2016-10-11 19:28 1213人阅读 评论(0) 收藏 举报

1. 前言

本文分析的是**Android**系统源码，从frameworks层到hal层，记录了Camera进入预览模式的重点代码，主要为控制流程的代码，有关图像buffer的传递暂不涉及，硬件平台基于mt6735。由于某些函数比较复杂，在贴出代码时会适当对其进行简化。

2. APP层

这里将分析app层令Camera进入预览模式的两个重点api：setPreviewDisplay和startPreview

```
mCamera.setPreviewDisplay(mSurfaceHolder);
mCamera.startPreview();
```

3. setPreviewDisplay函数分析

预览图像最终是要在lcd上显示的，想要在lcd上显示图像就需要用到Surface 。填充Surface有两种方法，一种是注册callback函数，预览数据将在callback函数中返回，得到数据后再把它送到Surface里面；另一种是在开始预览之前就为底层设置好Surface，底层获取数据后直接把数据送到Surface里面，为底层设置好Surface就是setPreviewDisplay的作用，

3.1 frameworks层

先来看frameworks层的实现

```
public final void setPreviewDisplay(SurfaceHolder holder) throws IOException {
    if (holder != null) {
        setPreviewSurface(holder.getSurface());
    } else {
        setPreviewSurface((Surface)null);
    }
}
```

setPreviewSurface是一个jni函数，它的实现在android_hardware_Camera.cpp中

```
static void android_hardware_Camera_setPreviewSurface(JNIEnv *env, jobject thiz, jobject jSurface)
{
    sp<Camera> camera = get_native_camera(env, thiz, NULL);
    if (camera == 0) return;

    sp<IGraphicBufferProducer> gbp;
    sp<Surface> surface;
    if (jSurface) {
        surface = android_view_Surface_getSurface(env, jSurface);
        if (surface != NULL) {
            gbp = surface->getIGraphicBufferProducer();
        }
    }

    if (camera->setPreviewTarget(gbp) != NO_ERROR) {
        jniThrowException(env, "java/io/IOException", "setPreviewTexture failed");
    }
}
```

```
// pass the buffered IGraphicBufferProducer to the camera service
status_t Camera::setPreviewTarget(const sp<IGraphicBufferProducer>& bufferProducer)
{
    sp <ICamera> c = mCamera;
    return c->setPreviewTarget(bufferProducer);
}
```

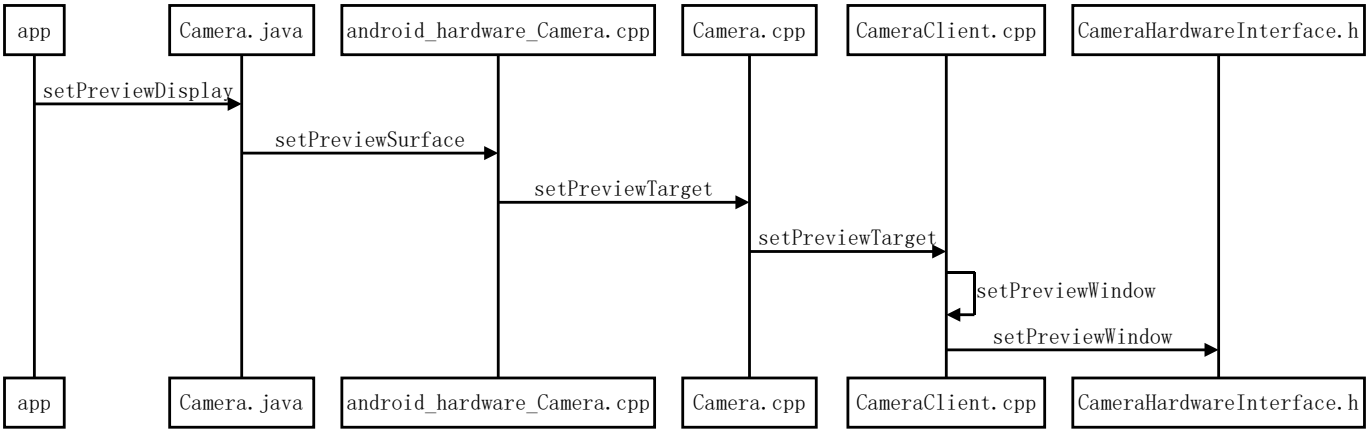
```
// set the buffer consumer that the preview will use
status_t CameraClient::setPreviewTarget(
        const sp<IGraphicBufferProducer>& bufferProducer) {
    sp<IBinder> binder;
    sp<ANativeWindow> window;
    if (bufferProducer != 0) {
        binder = bufferProducer->asBinder();
        window = new Surface(bufferProducer, /*controlledByApp*/ true);
    }
    return setPreviewWindow(binder, window);
}
```

ANativeWindow顾名思义“本地窗口”，Surface类继承了ANativeWindow类。按照网上的说法，ANativeWindow类是连接OpenGL和Android窗口系统的桥梁，即OpenGL需要通过ANativeWindow类来间接地操作Android窗口系统。但我们接下来要操作ANativeWindow的不是OpenGL，而是CameraClient

```

status_t CameraClient::setPreviewWindow(const sp<IBinder>& binder,
        const sp<ANativeWindow>& window) {
    if (window != 0) {
        result = native_window_api_connect(window.get(), NATIVE_WINDOW_API_CAMERA);
        if (result != NO_ERROR) {
            ALOGE("native_window_api_connect failed: %s (%d)", strerror(-result),
                    result);
            return result;
        }
    }

    // If preview has been already started, register preview buffers now.
    if (mHardware->previewEnabled()) {
        if (window != 0) {
            native_window_set_scaling_mode(window.get(),
                    NATIVE_WINDOW_SCALING_MODE_SCALE_TO_WINDOW);
            native_window_set_buffers_transform(window.get(), mOrientation);
            result = mHardware->setPreviewWindow(window);
        }
    }
    return result;
}
```

```
/** Set the ANativeWindow to which preview frames are sent */
status_t setPreviewWindow(const sp<ANativeWindow>& buf)
{
    mPreviewWindow = buf;
    mHalPreviewWindow.user = this;
    return mDevice->ops->set_preview_window(mDevice,
                buf.get() ? &mHalPreviewWindow.nw : 0);
}
```

ANativeWindow最终保存在mPreviewWindow变量中，而传到Hal层的则是mHalPreviewWindow.nw 操作集，Hal层将通过它来间接的操作mPreviewWindow。



mDevice就是上篇博文Camera打开流程中最后讲到的从Hal返回的mDevice对象，而它的ops指针指向的是gCameraDevOps结构体，从这里开始进入Hal层

3.2 Hal层

gCameraDevOps就在Cam1Device.cpp中定义

```
static mtk_camera_device_ops const
gCameraDevOps =
{
    #define OPS(name) name: camera_##name

    {
        OPS(set_preview_window),
        OPS(set_callbacks),
        OPS(enable_msg_type),
        OPS(disable_msg_type),
        OPS(msg_type_enabled),
        OPS(start_preview),
        OPS(stop_preview),
        OPS(preview_enabled),
        OPS(store_meta_data_in_buffers),
        OPS(start_recording),
        OPS(stop_recording),
        OPS(recording_enabled),
        OPS(release_recording_frame),
        OPS(auto_focus),
        OPS(cancel_auto_focus),
        OPS(take_picture),
        OPS(cancel_picture),
        OPS(set_parameters),
        OPS(get_parameters),
        OPS(put_parameters),
        OPS(send_command),
        OPS(release),
        OPS(dump)
    },
    OPS(mtk_set_callbacks),

    #undef  OPS
};
```

可以看到有关Camera的所有操作都这这里，接着看函数set_preview_window的实现

```
//  Implementation of camera_device_ops
static int camera_set_preview_window(
    struct camera_device * device,
    struct preview_stream_ops *window
)
{
    int err = -EINVAL;

    Cam1Device*const pDev = Cam1Device::getDevice(device);
```

```
    if  ( pDev )
    {
        err = pDev->setPreviewWindow(window);
    }

    return  err;
}
```

Cam1Device::getDevice函数获取到的将是DefaultCam1Device对象，而setPreviewWindow函数则在它的父类Cam1DeviceBase中实现

```
/*******************************************************************************
 *  Set the preview_stream_ops to which preview frames are sent.
 ******************************************************************************/
status_t
Cam1DeviceBase::
setPreviewWindow(preview_stream_ops* window)
{
    status_t status = initDisplayClient(window);
    if  ( OK == status && previewEnabled() && mpDisplayClient != 0 )
    {
        status = enableDisplayClient();
    }

    return  status;
}
```

第9行，初始化DisplayClient

第11行，通知DisplayClient开始工作

重点关注下函数initDisplayClient 的实现

```
status_t
Cam1DeviceBase::
initDisplayClient(preview_stream_ops* window)
{
    status_t status = OK;
    Size previewSize;

    //  [1] Check to see whether the passed window is NULL or not.
    if  ( ! window )
    {
        if  ( mpDisplayClient != 0 )
        {
            mpDisplayClient->uninit();
            mpDisplayClient.clear();
        }
        status = OK;
        goto lbExit;
    }

    //  [2] Get preview size.
    if  ( ! queryPreviewSize(previewSize.width, previewSize.height) )
    {
        status = DEAD_OBJECT;
        goto lbExit;
    }
    //  [3] Initialize Display Client.
    if  ( mpDisplayClient != 0 )
    {
        ......
    }
    //  [3.1] create a Display Client.
    mpDisplayClient = IDisplayClient::createInstance();
    if  ( mpDisplayClient == 0 )
    {
        MY_LOGE("Cannot create mpDisplayClient");
        status = NO_MEMORY;
        goto lbExit;
    }
    //  [3.2] initialize the newly-created Display Client.

```

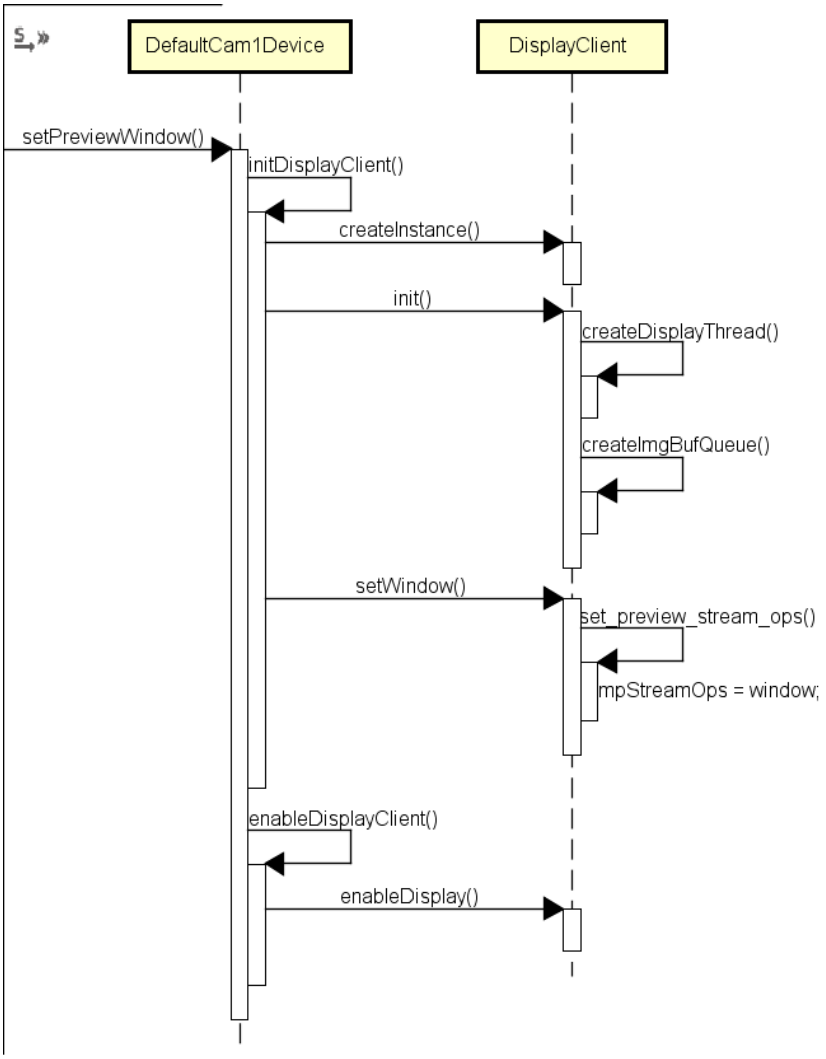
```
41      if  ( ! mpDisplayClient->init() )
42      {
43          MY_LOGE("mpDisplayClient init() failed");
44          mpDisplayClient->uninit();
45          mpDisplayClient.clear();
46          status = NO_MEMORY;
47          goto lbExit;
48      }
49      //  [3.3] set preview_stream_ops & related window info.
50      if  ( ! mpDisplayClient->setWindow(window, previewSize.width, previewSize.height, queryDisplayBufCount()) )
51      {
52          status = INVALID_OPERATION;
53          goto lbExit;
54      }
55      //  [3.4] set Image Buffer Provider Client if it exist.
56      if  ( mpCamAdapter != 0 && ! mpDisplayClient->setImgBufProviderClient(mpCamAdapter) )
57      {
58          status = INVALID_OPERATION;
59          goto lbExit;
60      }
61
62      status = OK;
63
64  lbExit:
65      if  ( OK != status )
66      {
67          MY_LOGD("Cleanup...");
68          ......
69      }
70
71      return  status;
    }
```

initDisplayClient函数都做了些什么事情注释已经写得很清楚

第31-47行，创建并初始化DisplayClient，其中DisplayClient是图像消费者，由它负责将图像数据送往Surface

第48-53行，DisplayClient想要操作Surface只能通过preview_stream_ops，也就是从上层传下来mHalPreviewWindow.nw操作集，setWindow函数会通过preview_stream_ops对Surface设置一些参数，并把preview_stream_ops保存在DisplayClient的mpStreamOps变量中，以后用到的时候才找得到。

第54-59行，DisplayClient作为消费者，那么就会有生产者，也就是CamAdapter。由CamAdapter提供图像数据，再由DisplayClient将数据送往Surface。但由于这个时候的 mpCamAdapter 为空，所以这里的setImgBufProviderClient函数暂时不会被调用。



4. startPreview函数分析

app层通过调用startPreview函数来进入预览模式，与setPreviewWindow的流程一样，最终会调到Cam1DeviceBase的startPreview函数

4.1 Cam1DeviceBase::startPreview函数分析

```
/*******************************************************************************
 *  Start preview mode.
 *******************************************************************************/
status_t
Cam1DeviceBase::
startPreview()
{
    status_t status = OK;

    if  ( ! onStartPreview() )
    {
        MY_LOGE("onStartPreviewLocked() fail");
        status = INVALID_OPERATION;
        goto lbExit;
    }

    if  ( mpDisplayClient == 0 )
    {
        MY_LOGD("DisplayClient is not ready.");
    }
    else if ( OK != (status = enableDisplayClient()) )
    {
        goto lbExit;
    }

    ......
```

```
    //  startPreview in Camera Adapter.
    {
        status = mpCamAdapter->startPreview();
        if  ( OK != status )
        {
            MY_LOGE("startPreview() in CameraAdapter returns: [%s(%d)]", ::strerror(-status), -status);
            goto lbExit;
        }
    }

    ......

    status = OK;
lbExit:
    if  ( OK != status )
    {
        ......
    }

    MY_LOGI("- status(%d)", status);
    return  status;
}
```

第10行， onStartPreview函数主要就是创建并初始化 CameraAdapter

第21行， 通知DisplayClient开始工作

第30行， mpCamAdapter->startPreview函数工作量巨大，包含了初始化buffer、3A，设置ISP和sensor驱动进入预览模式等工作。

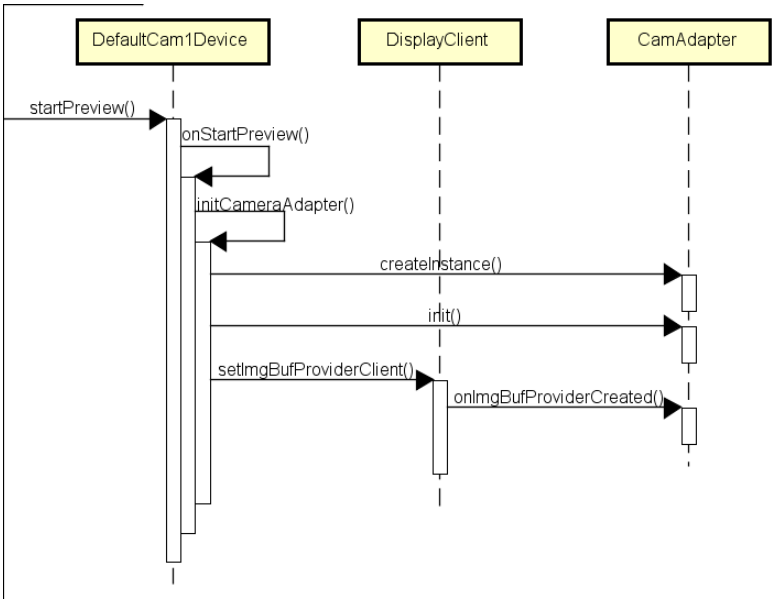
先看CameraAdapter的初始化

```
DefaultCam1Device::
onStartPreview()
{
    bool ret = false;

    ......

    //  (2) Initialize Camera Adapter.
    if  ( ! initCameraAdapter() )
    {
        MY_LOGE("NULL Camera Adapter");
        goto lbExit;
    }
    //
    ret = true;
lbExit:
    return ret;
}
```

```
bool
Cam1DeviceBase::
initCameraAdapter()
{
    bool ret = false;

    //  Create & init a new CamAdapter.
    mpCamAdapter = ICamAdapter::createInstance(mDevName, mi40penId, mpParamsMgr);
    if  ( mpCamAdapter != 0 && mpCamAdapter->init() )
    {
        //  (.1) init.
        mpCamAdapter->setCallbacks(mpCamMsgCbInfo);
        mpCamAdapter->enableMsgType(mpCamMsgCbInfo->mMsgEnabled);

        //  (.2) Invoke its setParameters
        if  ( OK != mpCamAdapter->setParameters() )
        {
            //  If fail, it should destroy instance before return.
            MY_LOGE("mpCamAdapter->setParameters() fail");
            goto lbExit;
        }

```

```
23
24          //  (.3) Send to-do commands.
25          {
26              Mutex::Autolock _lock(mTodoCmdMapLock);
27              for (size_t i = 0; i < mTodoCmdMap.size(); i++)
28              {
29                  CommandInfo const& rCmdInfo = mTodoCmdMap.valueAt(i);
30                  MY_LOGD("send queued cmd(%#x),args(%d,%d)", rCmdInfo.cmd, rCmdInfo.arg1, rCmdInfo.arg2);
31                  mpCamAdapter->sendCommand(rCmdInfo.cmd, rCmdInfo.arg1, rCmdInfo.arg2);
32              }
33              mTodoCmdMap.clear();
34          }
35
36          //  (.4) [DisplayClient] set Image Buffer Provider Client if needed.
37          if  ( mpDisplayClient != 0 && ! mpDisplayClient->setImgBufProviderClient(mpCamAdapter) )
38          {
39              MY_LOGE("mpDisplayClient->setImgBufProviderClient() fail");
40              goto lbExit;
41          }
42      }
43
44      ret = true;
45  lbExit:
46      return ret;
  }
```

创建CamAdapter实例并对它进行初始化。其中第35-40行，之前在setPreviewWindow里没机会调用的mpDisplayClient->setImgBufProviderClient函数将在这里调用。DisplayClient和CamAdapter将会通过setImgBufProviderClient函数关联起来，也就是告诉DisplayClient图像数据将由CamAdapter提供。至于CamAdpter如何获取图像数据和DisplayClient如何将数据送往Surface将在以后解析。



4.2 mpCamAdapter->startPreview函数分析

既然数据由CamAdapter提供，那么怎么告诉它开始向DisplayClient提供数据呢，还的继续分析mpCamAdapter->startPreview函数

```
1
2   status_t
3   CamAdapter::
4   startPreview()
5   {
6       return  mpStateManager->getCurrentState()->onStartPreview(this);
    }
```

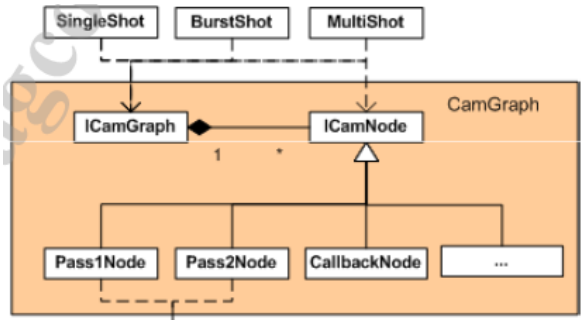
```
1
2   status_t
3   StateIdle::
4   onStartPreview(IStateHandler* pHandler)
5   {
        ......
```

```
    status = pHandler->onHandleStartPreview();
    ......
    return  status;
}
```

mpStateManager->getCurrentState函数获取到的是idle状态，在上文提到 mpCamAdapter->init函数中设置。而 StateIdle::onStartPreview函数将会回调CamAdapter的onHandleStartPreview函数，这个函数很长，非常长，相当长。

```
/*******************************************************************************
*   CamAdapter::startPreview() -> IState::onStartPreview() ->
*   IStateHandler::onHandleStartPreview() -> CamAdapter::onHandleStartPreview()
********************************************************************************/
status_t
CamAdapter::
onHandleStartPreview()
{
    ......

    mpPass2Node = Pass2Node::createInstance(PASS2_FEATURE);
    mpCamGraph          = ICamGraph::createInstance(
                                    getOpenId(),
                                    mUserName.string());
    mpPass1Node         = Pass1Node::createInstance(p1NodeInitCfg);
    mpCamGraph->setBufferHandler(   PASS1_RESIZEDRAW,   mpAllocBufHdl);
    mpCamGraph->setBufferHandler(   PASS1_FULLRAW,      mpAllocBufHdl);
    mpCamGraph->connectData(    PASS1_RESIZEDRAW,   CONTROL_RESIZEDRAW, mpPass1Node,        mpDefaultCtrlNode);
    mpCamGraph->connectData(    CONTROL_PRV_SRC,    PASS2_PRV_SRC,      mpDefaultCtrlNode,  mpPass2Node);
    mpCamGraph->connectNotify(  PASS1_START_ISP,    mpPass1Node,        mpDefaultCtrlNode);
    mpCamGraph->connectNotify(  PASS1_STOP_ISP,     mpPass1Node,        mpDefaultCtrlNode);
    mpCamGraph->connectNotify(  PASS1_EOF,          mpPass1Node,        mpDefaultCtrlNode);

    if ( !mpCamGraph->init() ) {
        ......
    }
    if ( !mpCamGraph->start() ) {
        ......
    }
lbExit:
    ......
    return ret;
}
```

暂时先把那些乱七八糟的参数设置的代码忽略掉，重点关注下 Pass1Node、 Pass2Node和DefaultCtlNode，以及作为各个Node通讯的桥梁的CamGraph。



CamGraph代表了整个系统，而使用不同的Node来描述不同的buffer处理， 所有的Node都需要连接到CamGraph。各个Node之间的通讯就需要用到connectData和 connectNotify函数， connectData为两个node之间buffer传输的连接，而 connectNotify为两个node之间消息传输的连接。

例如第18行调用了connectData(PASS1_RESIZEDRAW, CONTROL_RESIZEDRAW, mpPass1Node,mpDefaultCtrlNode)之后Pass1Node和DefaultCtrlNode就连接在一起，事件是 PASS1_RESIZEDRAW，也就是说当Pass1Node调用handlePostBuffer(PASS1_RESIZEDRAW, buffer)的时候，DefaultCtrlNode里面的onPostBuffer函数将会接受到Pass1Node的buffer。

同理第20行调用了connectNotify(PASS1_START_ISP, mpPass1Node, mpDefaultCtrlNode)，事件是 PASS1_START_ISP，当Pass1Node调用handleNotify(PASS1_START_ISP)的时候，DefaultCtrlNode里面的onNotify函数将会接收到 PASS1_START_ISP消息。

connectData和connectNotify的不同之处在于，一个可以传输整个buffer，但只能一对一连接，一个只能传输消息，但可以一对多连接，这两个函数的实现这里就不解析了，里面各种子类、父类的关系比较复杂，整理起来比较麻烦。需要关注的是 mpCamGraph->init和 mpCamGraph->start这两个函数，先来看看init

```
MBOOL
ICamGraph::
init()
{
    return mpImpl->init();
}
```

这里的 mpImpl指的是ICamGraphImpl

```
MBOOL
ICamGraphImpl::
init()
{
    Mutex::Autolock _l(mLock);
    MY_LOGD("init +");
    MY_ASSERT_STATE( mState == State_Connected, mState );

    MBOOL ret = MTRUE;
    vector< ICamNodeImpl* >::const_iterator iter;
    for( iter = mvNodeImpls.begin(); iter != mvNodeImpls.end(); iter++ )
    {
        MY_ASSERT_NODE_OP( ret, (*iter), init );
    }

lbExit:
    if( !ret )
    {
        ......
    }
    else
    {
        mState = State_Initiated;
    }
    MY_LOGD("init -");
    return ret;
}
```

mvNodeImpls里保存的是ICamThreadImpl对象， 每一个ICamThreadImpl代表一个CamNode，例如Pass1Node。这个函数所做的事情就是循环遍历所有的ICamThreadImpl，并且调用它们的init函数

```
MBOOL
ICamThreadImpl::
init()
{
    Mutex::Autolock _l(mLock);
    MY_ASSERT_STATE( mState == State_Connected, mState );

    MY_LOGV("init");
    MY_ASSERT( mpSelf->onInit() );
    MY_ASSERT( mpThread->createThread()
            && mpThread->sendThreadCmd(TCmd_Sync)
            && mpThread->sendThreadCmd(TCmd_Init)
            && mpThread->sendThreadCmd(TCmd_Sync));

    mState = State_Initiated;
    return MTRUE;
}
```

ICamThreadImpl里的mySelf成员就指向了它所代表的CamNode，例如Pass1Node。也就是说接下来所有保存在 mvNodeImpls里面的CamNode的onInit函数都会被调用。保存在mvNodeImpls里面的CamNode有很多，例如Pass1Node、Pass2Node、DefaultCtrlNode等。Pass1Node负责和Sensor Driver、ISP Driver打交道，进入预览模式的重点工作都由它来完成，所以这里只分析Pass1Node，来看看Pass1Node的onInit函数

```
MBOOL
Pass1NodeImpl::
onInit()
{
    ......
    mpIspSyncCtrlHw = IspSyncControlHw::createInstance(getSensorIdx());
    mpIspSyncCtrlHw->setIspEnquePeriod(mIspEnquePeriod);
    mpIspSyncCtrlHw->setSensorInfo(
            mInitCfg.muScenario,
            sensorSize.w,
            sensorSize.h,
            mSensorInfo.sensorType);
    ......
    mpCamIO = (IHalCamIO*)INormalPipe::createInstance(getSensorIdx(), getName(), mIspEnquePeriod);
    if( !mpCamIO )
    {
        MY_LOGE("create NormalPipe failed");
        goto lbExit;
    }
    if( !mpCamIO->init() )
    {
        MY_LOGE("camio init failed");
        goto lbExit;
    }
    ret = MTRUE;
lbExit:
    return ret;
}
```

主要就是对IspSyncCtrl和CamIO进行初始化，一个用来和ISP打交道，另一个用来和驱动打交道

回到onHandleStartPreview函数，在执行完mpCamGraph->init函数之后就到 mpCamGraph->start函数了。和mpCamGraph->init的流程一样，mpCamGraph->start所做的事情就是循环遍历所有的CamNode，并且回调它们的onStart函数，直接看Pass1Node的onStart函数

```
MBOOL
Pass1NodeImpl::
onStart()
{
    list<HwPortConfig_t> lHwPortCfg;
    if( !getHwPortConfig(&lHwPortCfg) )
    {
        MY_LOGE("getHwPortConfig failed");
        goto lbExit;
    }

    if( !startHw(lHwPortCfg) )
    {
        MY_LOGE("startHw failed");
        goto lbExit;
    }
    ret = MTRUE;
lbExit:
    FUNC_END;
    return ret;
}
```

接着看startHw函数的实现

```
MBOOL
Pass1NodeImpl::
startHw(list<HwPortConfig_t> & plPortCfg)
{
    // 1. Allocated ring buffers.
    if( pthread_create(&mThreadHandle, NULL, doThreadAllocBuf, &th_data) != 0 )
    {
        MY_LOGE("pthread create failed");
        goto lbExit;
    }

```

```
12      // 2. Lock Pass1 HW
13      if( !mpIspSyncCtrlHw->lockHw(IspSyncControlHw::HW_PASS1) )
14      {
15          MY_LOGE("isp sync lock pass1 failed");
16          goto lbExit;
17      }
18
19      ......
20      // 3. Configure RRZO and IMGO
21      if( !mpCamIO->configPipe(halCamIOinitParam) ) {
22          MY_LOGE("configPipe failed");
23          goto lbExit;
24      }
25
26      newMagicNum = mpIspSyncCtrlHw->getMagicNum(MTRUE);
27      if( !configFrame(newMagicNum) ) {
28          MY_LOGE("configFrame failed");
29          goto lbExit;
30      }
31
32      // 4. Send PASS1_START_ISP event
33      handleNotify(PASS1_START_ISP, newMagicNum, 0);
34
35      ......
36      // 5. Enque buffer
37      if( !mpCamIO->enque(halCamIOQBuf) ) {
38          MY_LOGE("enque failed");
39          goto lbExit;
40      }
41
42      // 6. Start ISP
43      if( !mpCamIO->start() ) {
44          MY_LOGE("start failed");
45          goto lbExit;
46      }
47
48      ret = MTRUE;
49  lbExit:
50      if( !ret ) {
51          ......
52      }
53      return ret;
54  }
```

这个函数做的事情比较多，上面标记的每个步骤都很复杂

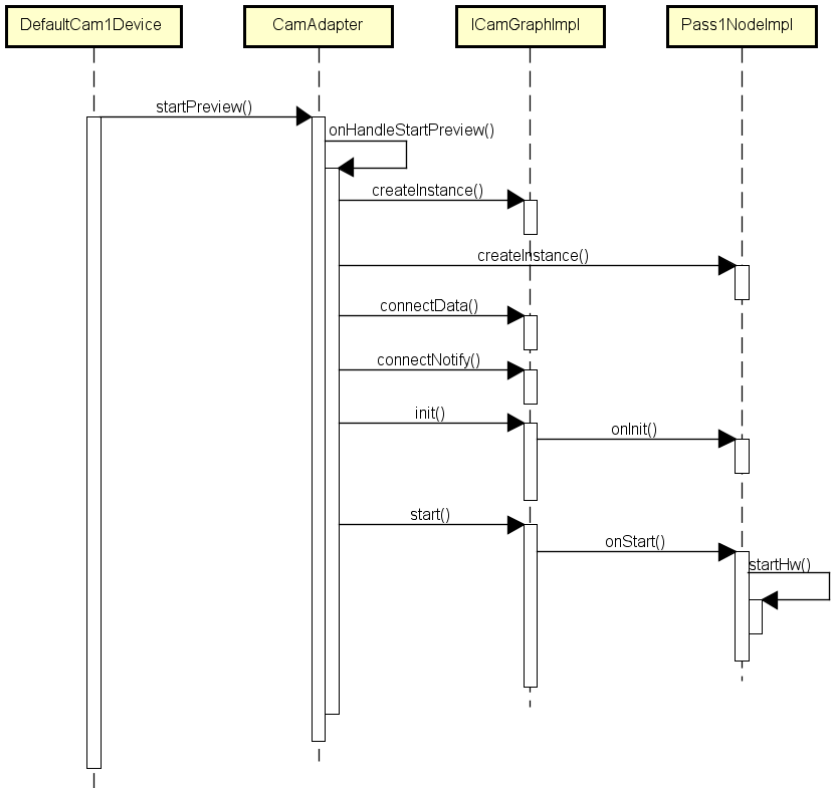
第5-10行：创建一个线程来分配ring buffers，用于存放从驱动获取到的图像数据

第20-24行：配置ISP和Sensor驱动预览相关的参数，记得sensor驱动中(例如imx214mipiraw_Sensor.c)的preview_setting函数吗，就是在这个时候被调用的

```
static void preview_setting(void)
{
  //Preview 2104*1560 30fps 24M MCLK 4lane 608Mbps/lane
  //preview 30.01fps
  write_cmos_sensor(0x0100,0x00);
  write_cmos_sensor(0x0114,0x03);
  write_cmos_sensor(0x0220,0x00);
  write_cmos_sensor(0x0221,0x11);
  write_cmos_sensor(0x0222,0x01);
  write_cmos_sensor(0x0340,0x08);
  write_cmos_sensor(0x0341,0x3E);
  write_cmos_sensor(0x0342,0x13);
  write_cmos_sensor(0x0343,0x90);
  write_cmos_sensor(0x0344,0x00);
  write_cmos_sensor(0x0345,0x00);
  write_cmos_sensor(0x0346,0x00);
  write_cmos_sensor(0x0347,0x00);
```

第33行：发送PASS1_START_ISP事件，其它的CamNode接收到该事件后会做相应的处理，例如DefaultCtlNode，会通知Hal3A进入CameraPreview状态

第42-46行：让ISP开始工作，到这里准备工作都已经完成，Camera已经进入了预览模式，接下来就是不断获取图像数据，并将它送到显示器了。



5. 总结

setPreviewWindow函数就是为hal层准备好Surface，hal层只能通过上层传下来的mHalPreviewWindow.nw来间接的操作Surface，而mHalPreviewWindow.nw保存在DisplayClient里面，也就是说DisplayClient是lcd显示图像的关键

startPreview函数的工作重点在CamAdapter，它代表Camera硬件，由它提供图像数据给DisplayClient。CamAdapter包含了多个CamNode，不同的CamNode用来描述不同的buffer处理，例如Pass1Node，它负责和驱动打交道，进入预览模式的重点工作都在它的startHw函数里面完成。