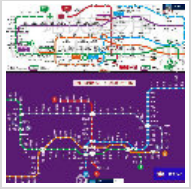
galaren77的ChinaUnix博客

前事不忘后事之师



galaren77

博客访问: 526679
博文数量: 256
博客积分: 1242
博客等级: 少尉
技术积分: 1406
用 户 组: 普通用户
注册时间: 2012-05-03 21:49

Android Camera HAL浅析

2013-09-23 21:04:10

分类: Android平台

原文地址: Android Camera HAL浅析 作者: 守候心田

1、Camera成像原理介绍

Camera工作流程图

Camera的成像原理可以简单概括如下:

景物(SCENE)通过镜头（LENS）生成的光学图像投射到图像传感器(Sensor)表面上，然后转为电信号，经过A/D（模数转换）转换后变为数字 图像信号，再送到数字信号处理芯片（DSP）中加工处理，再通过IO接口传输到CPU中处理，通过DISPLAY就可以看到图像了。

电荷耦合器件(CCD)或**互补金属氧化物半导体（CMOS）**接收光学镜头传递来的影像，经模/数转换器(A/D)转换成数字信号，经过编码后存储。

流程如下:

1）、CCD/CMOS将被摄体的光信号转变为电信号—电子图像（模拟信号）

2）、由模/数转换器（ADC）芯片来将模拟信号转化为数字信号

3）、数字信号形成后，由DSP或编码库对信号进行压缩并转化为特定的图像文件格式储存

数码相机的光学镜头与传统相机相同，将影像聚到感光器件上，即(光)电荷耦合器件(CCD) 。CCD替代了传统相机中的感光胶片的位置，其功能是将光信号转换成电信号，与电视摄像相同。

CCD是半导体器件，是数码相机的核心，其内含器件的单元数量决定了数码相机的成像质量——像素，单元越多，即像素数高，成像质量越好，通常情况下像素的高低代表了数码相机的档次和技术指标。

2、Android Camera框架

Android的Camera子系统提供一个拍照和录制视频的框架。

它将Camera的上层应用与Application Framework、用户库串接起来，而正是这个用户库来与Camera的硬件层通信，从而实现操作camera硬件。

kernel（3）
shell（11）
GDB（13）
c99（19）
LINUX 编程（5）
linux内核（13）
coredump（9）
linux（4）
未分配的博文（13）

文章存档

2016年（5）
2013年（95）
2012年（156）

我的朋友

6449240 蜗牛爬出 wungwei
kaifeng cuteyoun qq10363
wellguar Larpente athzhang

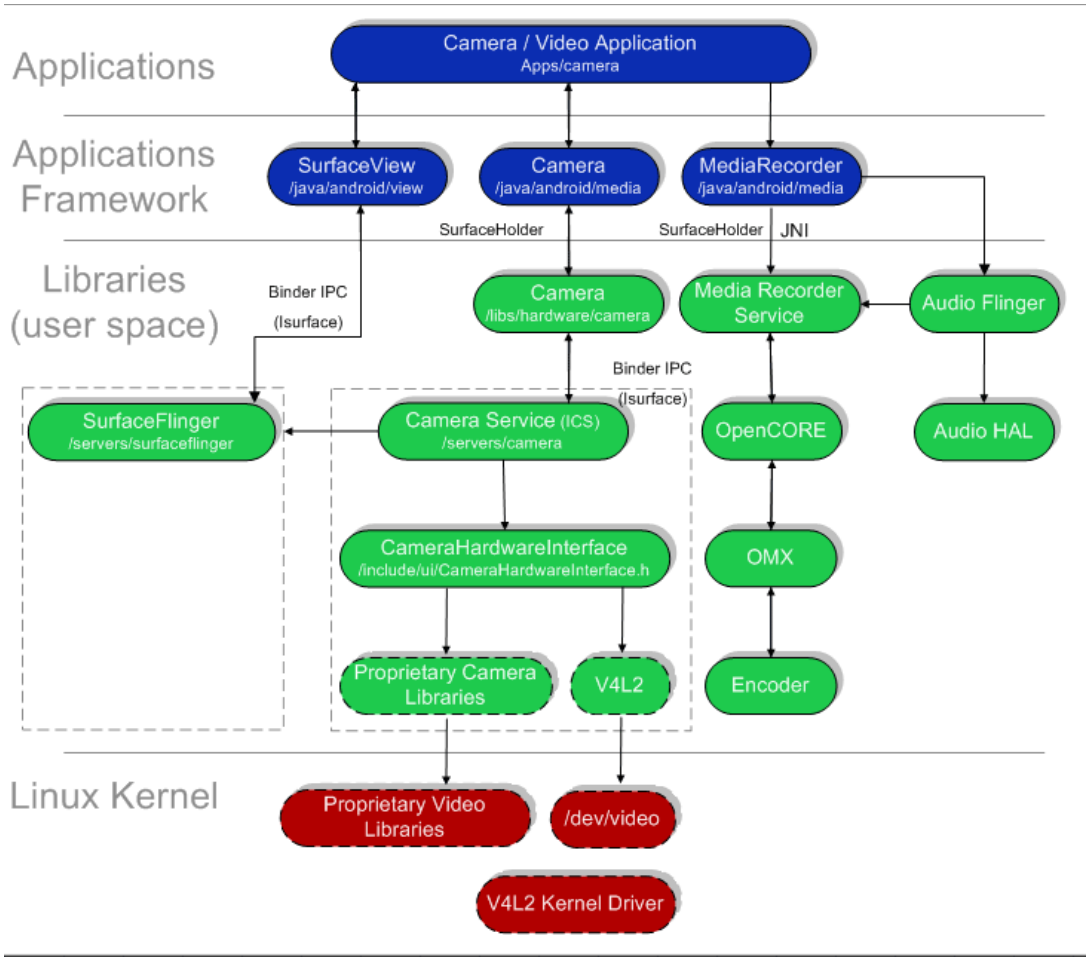
最近访客

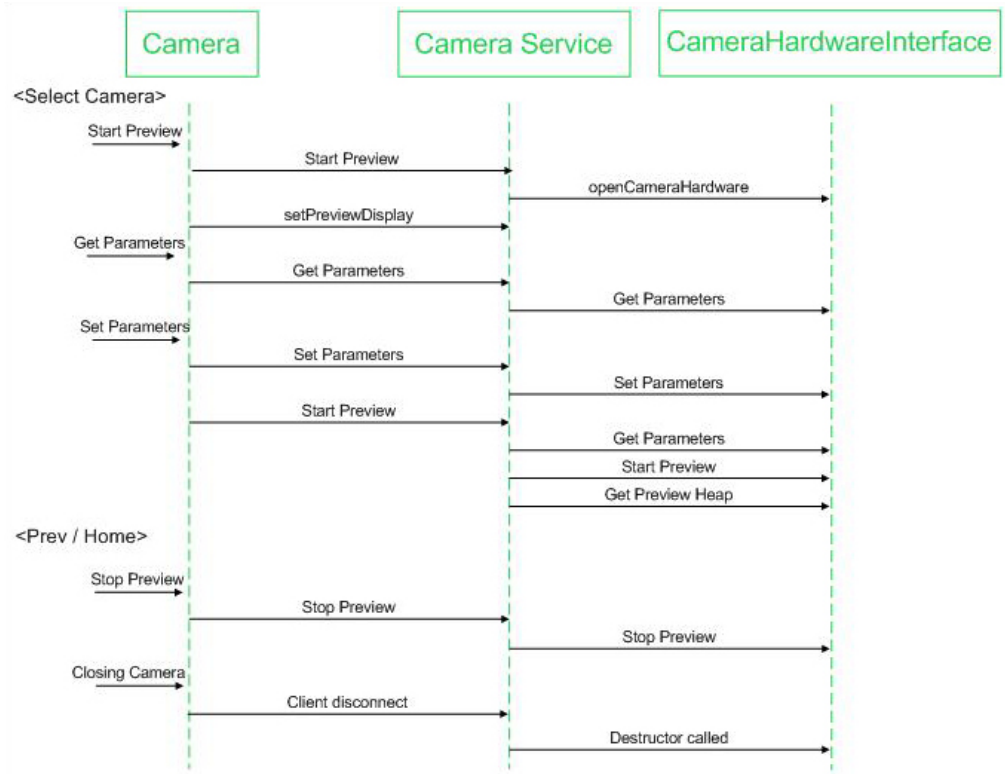
zhihui_y bandiyan xiaoyu_l
不见天高 XiongHai List_lin
lxglxt aggresss jamyy200

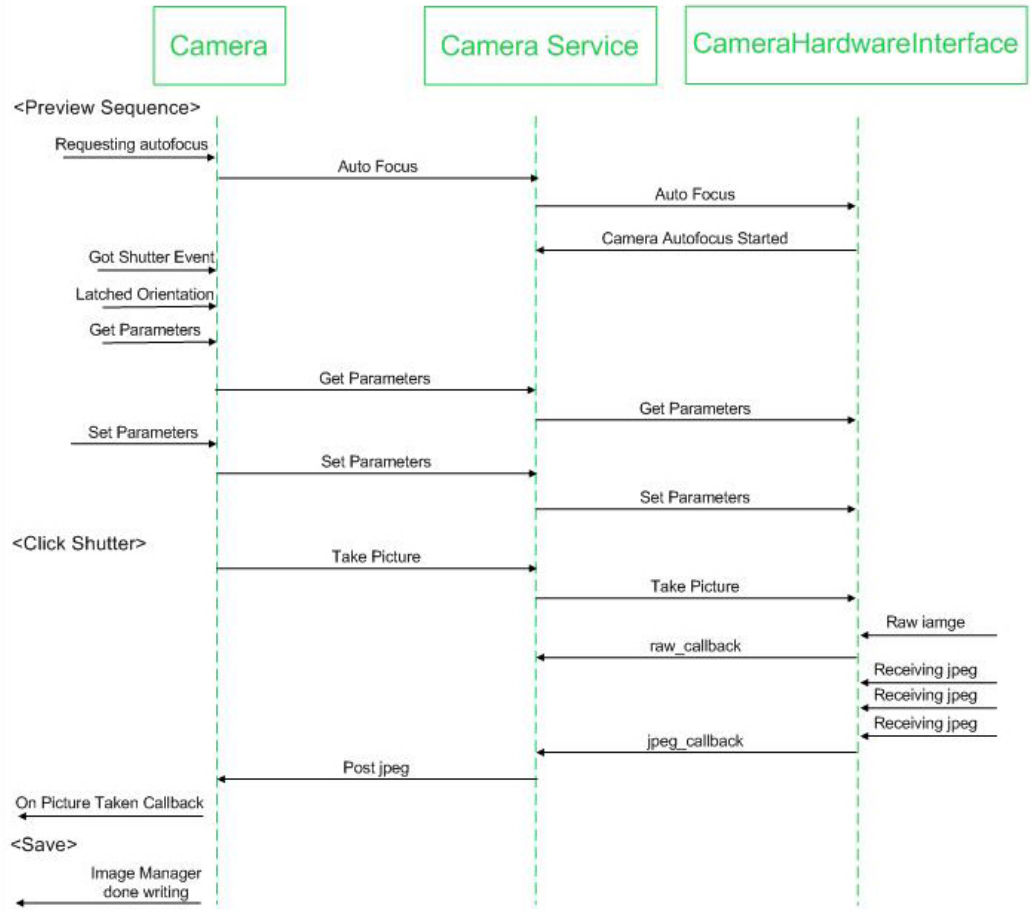
微信关注

IT168企业级官微
微信号：IT168qiye
系统架构师大会
微信号：SACC2013

订阅







3.Camera HAL层部分

代码存放目录：hardware\rk29\camera

编译：

[cpp] view plaincopy

```
1. LOCAL_PATH:= $(call my-dir)
2. include $(CLEAR_VARS)
3. LOCAL_SRC_FILES:=\
4.     CameraHal_Module.cpp\
5.     CameraHal.cpp\
6.     CameraHal_Utils.cpp\
7.     MessageQueue.cpp\
8.     CameraHal_Mem.cpp
9. ...................
10. ifeq ($(strip $(TARGET_BOARD_HARDWARE)),rk30board)
11. LOCAL_MODULE:= camera.rk30board
```

为了实现一个具体功能的Camera，在HAL层需要一个硬件相关的Camera库（例如通过调用video for linux驱动程序和Jpeg编码程序实现或者直接用各个chip厂商实现的私有库来实现，比如Qualcomm实现的libcamera.so和libqcamera.so），此处为camera.rk30board.so实现CameraHardwareInterface规定的接口，来调用相 关的库，驱动相关的driver，实现对camera硬件的操作。这个库将被Camera的服务库libcameraservice.so调用。

3.1CameraHal_Module.cpp主要是Camera HAL对上层提供的接口，和实际设备无关，上层的本地库都直接调用这个文

件里面提供的接口。

[cpp] view plaincopy

```
static int camera_device_open(const hw_module_t* module, const char* name,
            hw_device_t** device);
static int camera_device_close(hw_device_t* device);
static int camera_get_number_of_cameras(void);
static int camera_get_camera_info(int camera_id, struct camera_info *info);


static struct hw_module_methods_t camera_module_methods = {
    open: camera_device_open
};


camera_module_t HAL_MODULE_INFO_SYM = {
  common: {
      tag: HARDWARE_MODULE_TAG,
      version_major: ((CONFIG_CAMERAHAL_VERSION&0xff00)>>8),
      version_minor: CONFIG_CAMERAHAL_VERSION&0xff,
      id: CAMERA_HARDWARE_MODULE_ID,
      name: CAMERA_MODULE_NAME,
      author: "RockChip",
      methods: &camera_module_methods,
      dso: NULL, /* remove compilation warnings */
      reserved: {0}, /* remove compilation warnings */
  },
  get_number_of_cameras: camera_get_number_of_cameras,
  get_camera_info: camera_get_camera_info,
};
```

//CAMERA_DEVICE_NAME "/dev/video" 以下都是通过读取节点信息来获取摄像头的数目及摄像头设备信息

[cpp] view plaincopy

```
int camera_device_close(hw_device_t* device)
{
  int ret = 0;
  rk_camera_device_t* rk_dev = NULL;

  LOGD("%s", __FUNCTION__);

  android::Mutex::Autolock lock(gCameraHalDeviceLock);

  if (!device) {
      ret = -EINVAL;
      goto done;
  }

  rk_dev = (rk_camera_device_t*) device;

  if (rk_dev) {
      if (gCameraHals[rk_dev->cameraid]) {
```

```
            delete gCameraHals[rk_dev->cameraid];
            gCameraHals[rk_dev->cameraid] = NULL;
            gCamerasOpen--;
        }

        if (rk_dev->base.ops) {
            free(rk_dev->base.ops);
        }
        free(rk_dev);
    }
done:

    return ret;
}

/*******************************************************************
 * implementation of camera_module functions
 *******************************************************************/

/* open device handle to one of the cameras
 *
 * assume camera service will keep singleton of each camera
 * so this function will always only be called once per camera instance
 */

int camera_device_open(const hw_module_t* module, const char* name,
                hw_device_t** device)
{
    int rv = 0;
    int cameraid;
    rk_camera_device_t* camera_device = NULL;
    camera_device_ops_t* camera_ops = NULL;
    android::CameraHal* camera = NULL;

    android::Mutex::Autolock lock(gCameraHalDeviceLock);

    LOGI("camera_device open");

    if (name != NULL) {
        cameraid = atoi(name);

        if(cameraid > gCamerasNumber) {
            LOGE("camera service provided cameraid out of bounds, "
                    "cameraid = %d, num supported = %d",
                    cameraid, gCamerasNumber);
            rv = -EINVAL;
            goto fail;
        }

        if(gCamerasOpen >= CAMERAS_SUPPORTED_SIMUL_MAX) {
```

```
        LOGE("maximum number(%d) of cameras already open",gCamerasOpen);
        rv = -ENOMEM;
        goto fail;
    }

    camera_device = (rk_camera_device_t*)malloc(sizeof(*camera_device));
    if(!camera_device) {
        LOGE("camera_device allocation fail");
        rv = -ENOMEM;
        goto fail;
    }

    camera_ops = (camera_device_ops_t*)malloc(sizeof(*camera_ops));
    if(!camera_ops) {
        LOGE("camera_ops allocation fail");
        rv = -ENOMEM;
        goto fail;
    }

    memset(camera_device, 0, sizeof(*camera_device));
    memset(camera_ops, 0, sizeof(*camera_ops));

    camera_device->base.common.tag = HARDWARE_DEVICE_TAG;
    camera_device->base.common.version = 0;
    camera_device->base.common.module = (hw_module_t *)(module);
    camera_device->base.common.close = camera_device_close;
    camera_device->base.ops = camera_ops;

    camera_ops->set_preview_window = camera_set_preview_window;
    camera_ops->set_callbacks = camera_set_callbacks;
    camera_ops->enable_msg_type = camera_enable_msg_type;
    camera_ops->disable_msg_type = camera_disable_msg_type;
    camera_ops->msg_type_enabled = camera_msg_type_enabled;
    camera_ops->start_preview = camera_start_preview;
    camera_ops->stop_preview = camera_stop_preview;
    camera_ops->preview_enabled = camera_preview_enabled;
    camera_ops->store_meta_data_in_buffers = camera_store_meta_data_in_buffers;
    camera_ops->start_recording = camera_start_recording;
    camera_ops->stop_recording = camera_stop_recording;
    camera_ops->recording_enabled = camera_recording_enabled;
    camera_ops->release_recording_frame = camera_release_recording_frame;
    camera_ops->auto_focus = camera_auto_focus;
    camera_ops->cancel_auto_focus = camera_cancel_auto_focus;
    camera_ops->take_picture = camera_take_picture;
    camera_ops->cancel_picture = camera_cancel_picture;
    camera_ops->set_parameters = camera_set_parameters;
    camera_ops->get_parameters = camera_get_parameters;
    camera_ops->put_parameters = camera_put_parameters;
    camera_ops->send_command = camera_send_command;
    camera_ops->release = camera_release;
```

```
        camera_ops->dump = camera_dump;

        *device = &camera_device->base.common;

        // -------- RockChip specific stuff --------

        camera_device->cameraid = cameraid;

        camera = new android::CameraHal(cameraid);

        if(!camera) {
            LOGE("Couldn't create instance of CameraHal class");
            rv = -ENOMEM;
            goto fail;
        }

        gCameraHals[cameraid] = camera;
        gCamerasOpen++;
    }

    return rv;

fail:
    if(camera_device) {
        free(camera_device);
        camera_device = NULL;
    }
    if(camera_ops) {
        free(camera_ops);
        camera_ops = NULL;
    }
    if(camera) {
        delete camera;
        camera = NULL;
    }
    *device = NULL;
    return rv;
}

int camera_get_number_of_cameras(void)
{
    char cam_path[20];
    char cam_num[3],i;
    int cam_cnt=0,fd=-1,rk29_cam[CAMERAS_SUPPORT_MAX];
    struct v4l2_capability capability;
    rk_cam_info_t camInfoTmp[CAMERAS_SUPPORT_MAX];
    char *ptr,**ptrr;
    char version[PROPERTY_VALUE_MAX];

    if (gCamerasNumber > 0)
```

```
        goto camera_get_number_of_cameras_end;

    memset(version,0x00,sizeof(version));
    sprintf(version,"%d.%d.%d",((CONFIG_CAMERAHAL_VERSION&0xff0000)>>16),
        ((CONFIG_CAMERAHAL_VERSION&0xff00)>>8),CONFIG_CAMERAHAL_VERSION&0xff);
    property_set(CAMERAHAL_VERSION_PROPERTY_KEY,version);

    memset(&camInfoTmp[0],0x00,sizeof(rk_cam_info_t));
    memset(&camInfoTmp[1],0x00,sizeof(rk_cam_info_t));

    for (i=0; i<10; i++) {
        cam_path[0] = 0x00;
        strcat(cam_path, CAMERA_DEVICE_NAME);
        sprintf(cam_num, "%d", i);
        strcat(cam_path,cam_num);
        fd = open(cam_path, O_RDONLY);
        if (fd < 0)
            break;

        memset(&capability, 0, sizeof(struct v4l2_capability));
        if (ioctl(fd, VIDIOC_QUERYCAP, &capability) < 0) {
            LOGE("Video device(%s): query capability not supported.\n",cam_path);
            goto loop_continue;
        }

        if ((capability.capabilities & (V4L2_CAP_VIDEO_CAPTURE | V4L2_CAP_STREAMING)) != (V4L2_CAP_VIDEO_CAPTURE | V4L2_CAP_STREAMING)) {
            LOGD("Video device(%s): video capture not supported.\n",cam_path);
        } else {
            memset(camInfoTmp[cam_cnt&0x01].device_path,0x00, sizeof(camInfoTmp[cam_cnt&0x01].device_path));
            strcat(camInfoTmp[cam_cnt&0x01].device_path,cam_path);
            memset(camInfoTmp[cam_cnt&0x01].fival_list,0x00, sizeof(camInfoTmp[cam_cnt&0x01].fival_list));
            memcpy(camInfoTmp[cam_cnt&0x01].driver,capability.driver, sizeof(camInfoTmp[cam_cnt&0x01].driver));
            camInfoTmp[cam_cnt&0x01].version = capability.version;
            if (strstr((char*)&capability.card[0], "front") != NULL) {
                camInfoTmp[cam_cnt&0x01].facing_info.facing = CAMERA_FACING_FRONT;
            } else {
                camInfoTmp[cam_cnt&0x01].facing_info.facing = CAMERA_FACING_BACK;
            }
            ptr = strstr((char*)&capability.card[0],"-");
            if (ptr != NULL) {
                ptr++;
                camInfoTmp[cam_cnt&0x01].facing_info.orientation = atoi(ptr);
            } else {
                camInfoTmp[cam_cnt&0x01].facing_info.orientation = 0;
            }
            cam_cnt++;

            memset(version,0x00,sizeof(version));
            sprintf(version,"%d.%d.%d",((capability.version&0xff0000)>>16),
```

```
            ((capability.version&0xff00)>>8),capability.version&0xff);
        property_set(CAMERADRIVER_VERSION_PROPERTY_KEY,version);

        LOGD("%s(%d): %s:%s",__FUNCTION__,__LINE__,CAMERADRIVER_VERSION_PROPERTY_KEY,version);

        if (cam_cnt >= CAMERAS_SUPPORT_MAX)
            i = 10;
    }
loop_continue:
    if (fd > 0) {
        close(fd);
        fd = -1;
    }
    continue;
  }
  //zyc , change the camera infomation if there is a usb camera
  if((strcmp(camInfoTmp[0].driver,"uvcvideo") == 0)) {
    camInfoTmp[0].facing_info.facing = (camInfoTmp[1].facing_info.facing == CAMERA_FACING_FRONT) ? CAMERA_FACING_BACK:CAMERA_FACING_FRONT;
    camInfoTmp[0].facing_info.orientation = (camInfoTmp[0].facing_info.facing == CAMERA_FACING_FRONT)?270:90;
  } else if((strcmp(camInfoTmp[1].driver,"uvcvideo") == 0)) {
    camInfoTmp[1].facing_info.facing = (camInfoTmp[0].facing_info.facing == CAMERA_FACING_FRONT) ? CAMERA_FACING_BACK:CAMERA_FACING_FRONT;
    camInfoTmp[1].facing_info.orientation = (camInfoTmp[1].facing_info.facing == CAMERA_FACING_FRONT)?270:90;
  }
  gCamerasNumber = cam_cnt;

#if CONFIG_AUTO_DETECT_FRAMERATE
  rk29_cam[0] = 0xff;
  rk29_cam[1] = 0xff;
  for (i=0; i
    if (strcmp((char*)&camInfoTmp[i].driver[0],"rk29xx-camera") == 0) {
        if (strcmp((char*)&camInfoTmp[i].driver[0],(char*)&gCamInfos[i].driver[0]) != 0) {
            rk29_cam[i] = i;
        }
    } else {
        rk29_cam[i] = 0xff;
    }
  }

  if ((rk29_cam[0] != 0xff) || (rk29_cam[1] != 0xff)) {
    if (gCameraFpsDetectThread == NULL) {
        gCameraFpsDetectThread = new CameraFpsDetectThread();
        LOGD("%s create CameraFpsDetectThread for enum camera framerate!!",__FUNCTION__);
        gCameraFpsDetectThread->run("CameraFpsDetectThread", ANDROID_PRIORITY_AUDIO);
    }
  }
```

```
#endif
    #if CONFIG_CAMERA_SINGLE_SENSOR_FORCE_BACK_FOR_CTS
    if ((gCamerasNumber==1) && (camInfoTmp[0].facing_info.facing==CAMERA_FACING_FRONT)) {
        gCamerasNumber = 2;
        memcpy(&camInfoTmp[1],&camInfoTmp[0], sizeof(rk_cam_info_t));
        camInfoTmp[1].facing_info.facing = CAMERA_FACING_BACK;
    }
    #endif

    memcpy(&gCamInfos[0], &camInfoTmp[0], sizeof(rk_cam_info_t));
    memcpy(&gCamInfos[1], &camInfoTmp[1], sizeof(rk_cam_info_t));

camera_get_number_of_cameras_end:
    LOGD("%s(%d): Current board have %d cameras attached.",__FUNCTION__, __LINE__, gCamerasNumber);
    return gCamerasNumber;
}

int camera_get_camera_info(int camera_id, struct camera_info *info)
{
    int rv = 0,fp;
    int face_value = CAMERA_FACING_BACK;
    int orientation = 0;
    char process_name[30];

    if(camera_id > gCamerasNumber) {
        LOGE("%s camera_id out of bounds, camera_id = %d, num supported = %d",__FUNCTION__,
                camera_id, gCamerasNumber);
        rv = -EINVAL;
        goto end;
    }

    info->facing = gCamInfos[camera_id].facing_info.facing;
    info->orientation = gCamInfos[camera_id].facing_info.orientation;
end:
    LOGD("%s(%d): camera_%d facing(%d), orientation(%d)",__FUNCTION__,__LINE__,camera_id,info->facing,info->orientation);
    return rv;
}
```

而对于为上层提供的HAL层接口函数，并不直接操作节点，而是间接的去调用CameraHal.cpp去操作节点。

[cpp] view plaincopy

```
int camera_start_preview(struct camera_device * device)
{
    int rv = -EINVAL;
    rk_camera_device_t* rk_dev = NULL;

    LOGV("%s", __FUNCTION__);

    if(!device)
        return rv;

```

```
    rk_dev = (rk_camera_device_t*) device;

    rv = gCameraHals[rk_dev->cameraid]->startPreview();

    return rv;
}

void camera_stop_preview(struct camera_device * device)
{
    rk_camera_device_t* rk_dev = NULL;

    LOGV("%s", __FUNCTION__);

    if(!device)
        return;

    rk_dev = (rk_camera_device_t*) device;

    gCameraHals[rk_dev->cameraid]->stopPreview();
}
```

3.2CameraHal.cpp去操作节点来进行实际的操作。

//这个几个线程很关键，分别对应着各种不同的情况，但是一直在运行

[cpp] view plaincopy

```
CameraHal::CameraHal(int cameraId)
            :mParameters(),
            mSnapshotRunning(-1),
            mCommandRunning(-1),
            mPreviewRunning(STA_PREVIEW_PAUSE),
            mPreviewLock(),
            mPreviewCond(),
            mDisplayRuning(STA_DISPLAY_PAUSE),
            mDisplayLock(),
            mDisplayCond(),
            mANativeWindowLock(),
            mANativeWindowCond(),
            mANativeWindow(NULL),
            mPreviewErrorFrameCount(0),
            mPreviewFrameSize(0),
            mCamDriverFrmHeightMax(0),
            mCamDriverFrmWidthMax(0),
            mPreviewBufferCount(0),
            mCamDriverPreviewFmt(0),
            mCamDriverPictureFmt(0),
            mCamDriverV4l2BufferLen(0),
            mPreviewMemory(NULL),
            mRawBufferSize(0),
            mJpegBufferSize(0),
            mMsgEnabled(0),
            mEffect_number(0),
```

```
27.         mScene_number(0),
28.         mWhiteBalance_number(0),
29.         mFlashMode_number(0),
30.         mGps_latitude(-1),
31.         mGps_longitude(-1),
32.         mGps_altitude(-1),
33.         mGps_timestamp(-1),
34.         displayThreadCommandQ("displayCmdQ"),
35.         displayThreadAckQ("displayAckQ"),
36.         previewThreadCommandQ("previewCmdQ"),
37.         previewThreadAckQ("previewAckQ"),
38.         commandThreadCommandQ("commandCmdQ"),
39.         commandThreadAckQ("commandAckQ"),
40.         snapshotThreadCommandQ("snapshotCmdQ"),
41.         snapshotThreadAckQ("snapshotAckQ"),
42.         mCamBuffer(NULL)
43. {
44.     int fp,i;
45.
46.     cameraCallProcess[0] = 0x00;
47.     sprintf(cameraCallProcess,"/proc/%d/cmdline",getCallingPid());
48.     fp = open(cameraCallProcess, O_RDONLY);
49.     if (fp < 0) {
50.         memset(cameraCallProcess,0x00,sizeof(cameraCallProcess));
51.         LOGE("Obtain calling process info failed");
52.     } else {
53.         memset(cameraCallProcess,0x00,sizeof(cameraCallProcess));
54.         read(fp, cameraCallProcess, 30);
55.         close(fp);
56.         fp = -1;
57.         LOGD("Calling process is: %s",cameraCallProcess);
58.     }
59.
60.     iCamFd = -1;
61.     memset(&mCamDriverSupportFmt[0],0, sizeof(mCamDriverSupportFmt));
62.     mRecordRunning = false;
63.     mPictureRunning = STA_PICTURE_STOP;
64.     mExitAutoFocusThread = false;
65.     mDriverMirrorSupport = false;
66.     mDriverFlipSupport = false;
67.     mPreviewCmdReceived = false;
68.     mPreviewStartTimes = 0x00;
69.     memset(mCamDriverV4l2Buffer, 0x00, sizeof(mCamDriverV4l2Buffer));
70.     memset(mDisplayFormat,0x00,sizeof(mDisplayFormat));
71.     for (i=0; i
72.         mPreviewBufferMap[i] = NULL;
73.         mDisplayBufferMap[i] = NULL;
74.         memset(&mGrallocBufferMap[i],0x00,sizeof(rk_previewbuf_info_t));
75.         mPreviewBufs[i] = NULL;
76.         mVideoBufs[i] = NULL;
```

```
77.
78.         mPreviewBuffer[i] = NULL;
79.     }
80.
81.     //open the rga device,zyc
82.     mRGAFd = -1;
83.
84.     if (cameraCreate(cameraId) == 0) {
85.         initDefaultParameters();
86.
87.         cameraRawJpegBufferCreate(mRawBufferSize,mJpegBufferSize);
88.
89.         mDisplayThread = new DisplayThread(this);
90.         mPreviewThread = new PreviewThread(this);
91.         mCommandThread = new CommandThread(this);
92.         mPictureThread = new PictureThread(this);
93.     mSnapshotThread = new SnapshotThread(this);
94.         mAutoFocusThread = new AutoFocusThread(this);
95.         mDisplayThread->run("CameraDispThread",ANDROID_PRIORITY_URGENT_DISPLAY);
96.         mPreviewThread->run("CameraPreviewThread",ANDROID_PRIORITY_DISPLAY);
97.         mCommandThread->run("CameraCmdThread", ANDROID_PRIORITY_URGENT_DISPLAY);
98.         mAutoFocusThread->run("CameraAutoFocusThread", ANDROID_PRIORITY_DISPLAY);
99.         mSnapshotThread->run("CameraSnapshotThread", ANDROID_PRIORITY_NORMAL);
100.
101.        LOGD("CameraHal create success!");
102.    } else {
103.        mPreviewThread = NULL;
104.        mDisplayThread = NULL;
105.        mCommandThread = NULL;
106.        mPictureThread = NULL;
107.        mSnapshotThread = NULL;
108.        mAutoFocusThread = NULL;
109.    }
110.
111. }
```

初始化时参数的配置，默认参数图片大小，分辨率，帧等：

[cpp] view plaincopy

```
1.  void CameraHal::initDefaultParameters()
2.  {
3.      CameraParameters params;
4.      String8 parameterString;
5.      int i,j,previewFrameSizeMax;
6.      char cur_param[32],cam_size[10];
7.      char str_picturesize[100];//We support at most 4 resolutions: 2592x1944,2048x1536,1600x1200,1024x768
8.      int ret,picture_size_bit;
9.      struct v4l2_format fmt;
10.
11.     LOG_FUNCTION_NAME
```

```
12.     memset(str_picturesize,0x00,sizeof(str_picturesize));
13.     if (CAMERA_IS_UVC_CAMERA()) {
14.         /*preview size setting*/
15.         struct v4l2_frmsizeenum fsize;
16.
17.         memset(&fsize, 0, sizeof(fsize));
18.         picture_size_bit = 0;
19.         fsize.index = 0;
20.         fsize.pixel_format = mCamDriverPreviewFmt;
21.         while ((ret = ioctl(iCamFd, VIDIOC_ENUM_FRAMESIZES, &fsize)) == 0) {
22.             if (fsize.type == V4L2_FRMSIZE_TYPE_DISCRETE) {
23.                 if ((fsize.discrete.width == 320) && (fsize.discrete.height == 240)) {
24.                     if (strcmp(cameraCallProcess,"com.tencent.android.pad") == 0) {
25.                         fsize.index++;
26.                         continue;
27.                     }
28.                 }
29.                 memset(cam_size,0x00,sizeof(cam_size));
30.                 if (parameterString.size() != 0)
31.                     cam_size[0]=',';
32.                 sprintf((char*)(&cam_size[strlen(cam_size)]),"%d",fsize.discrete.width);
33.                 strcat(cam_size, "x");
34.                 sprintf((char*)(&cam_size[strlen(cam_size)]),"%d",fsize.discrete.height);
35.                 parameterString.append((const char*)cam_size);
36.
37.                 if ((strlen(str_picturesize)+strlen(cam_size))<sizeof(str_picturesize)) {
38.                     if (fsize.discrete.width <= 2592) {
39.                         strcat(str_picturesize, cam_size);
40.                         if (fsize.discrete.width > mCamDriverFrmWidthMax) {
41.                             mCamDriverFrmWidthMax = fsize.discrete.width;
42.                             mCamDriverFrmHeightMax = fsize.discrete.height;
43.                         }
44.                     }
45.                 } else {
46.                     break;
47.                 }
48.             } else if (fsize.type == V4L2_FRMSIZE_TYPE_CONTINUOUS) {
49.
50.                 break;
51.             } else if (fsize.type == V4L2_FRMSIZE_TYPE_STEPWISE) {
52.
53.                 break;
54.             }
55.             fsize.index++;
56.         }
57.         if (ret != 0 && errno != EINVAL) {
58.             LOGE("ERROR enumerating frame sizes: %d\n", errno);
59.         }
60.
61.         params.set(CameraParameters::KEY_SUPPORTED_PREVIEW_SIZES, parameterString.string());
```

```
    params.setPreviewSize(640,480);
    /*picture size setting*/
    params.set(CameraParameters::KEY_SUPPORTED_PICTURE_SIZES, str_picturesize);
    params.setPictureSize(mCamDriverFrmWidthMax,  mCamDriverFrmHeightMax);

    if (mCamDriverFrmWidthMax <= 1024) {
        mRawBufferSize = RAW_BUFFER_SIZE_1M;
        mJpegBufferSize = JPEG_BUFFER_SIZE_1M;
    } else if (mCamDriverFrmWidthMax <= 1600) {
        mRawBufferSize = RAW_BUFFER_SIZE_2M;
        mJpegBufferSize = JPEG_BUFFER_SIZE_2M;
    } else if (mCamDriverFrmWidthMax <= 2048) {
        mRawBufferSize = RAW_BUFFER_SIZE_3M;
        mJpegBufferSize = JPEG_BUFFER_SIZE_3M;
    } else if (mCamDriverFrmWidthMax <= 2592) {
        mRawBufferSize = RAW_BUFFER_SIZE_5M;
        mJpegBufferSize = JPEG_BUFFER_SIZE_5M;
    } else {
        LOGE("%s(%d):Camera Hal is only support 5Mega camera, but the uvc camera is %dx%d",
            __FUNCTION__,__LINE__,mCamDriverFrmWidthMax, mCamDriverFrmHeightMax);
        mRawBufferSize = RAW_BUFFER_SIZE_5M;
        mJpegBufferSize = JPEG_BUFFER_SIZE_5M;
    }

    /* set framerate */
    struct v4l2_streamparm setfps;

    memset(&setfps, 0, sizeof(struct v4l2_streamparm));
    setfps.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    setfps.parm.capture.timeperframe.numerator=1;
    setfps.parm.capture.timeperframe.denominator=15;
    ret = ioctl(iCamFd, VIDIOC_S_PARM, &setfps);

    /*frame rate setting*/
    params.set(CameraParameters::KEY_SUPPORTED_PREVIEW_FRAME_RATES, "15");
    params.setPreviewFrameRate(15);
    /*frame per second setting*/
    parameterString = "15000,15000";
    params.set(CameraParameters::KEY_PREVIEW_FPS_RANGE, parameterString.string());
    parameterString = "(15000,15000)";
    params.set(CameraParameters::KEY_SUPPORTED_PREVIEW_FPS_RANGE, parameterString.string());
    /*not support zoom */
    params.set(CameraParameters::KEY_ZOOM_SUPPORTED, "false");

} else if (CAMERA_IS_RKSOC_CAMERA()) {

    fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    fmt.fmt.pix.pixelformat= mCamDriverPreviewFmt;
    fmt.fmt.pix.field = V4L2_FIELD_NONE;

```

```
    /*picture size setting*/
    fmt.fmt.pix.width = 10000;
    fmt.fmt.pix.height = 10000;
    ret = ioctl(iCamFd, VIDIOC_TRY_FMT, &fmt);

    mCamDriverFrmWidthMax = fmt.fmt.pix.width;
    mCamDriverFrmHeightMax = fmt.fmt.pix.height;

    if (mCamDriverFrmWidthMax > 2592) {
        LOGE("Camera driver support maximum resolution(%dx%d) is overflow 5Mega!",mCamDriverFrmWidthMax,mCamDriverFrmHeightMax);
        mCamDriverFrmWidthMax = 2592;
        mCamDriverFrmHeightMax = 1944;
    }

    /*preview size setting*/
    if (mCamDriverFrmWidthMax >= 176) {
        fmt.fmt.pix.width = 176;
        fmt.fmt.pix.height = 144;
        if (ioctl(iCamFd, VIDIOC_TRY_FMT, &fmt) == 0) {
            if ((fmt.fmt.pix.width == 176) && (fmt.fmt.pix.height == 144)) {
                parameterString.append("176x144");
                params.setPreviewSize(176, 144);
                previewFrameSizeMax =  PAGE_ALIGN(176*144*2)*2;        // 176*144*2     rgb565
                //params.set(CameraParameters::KEY_PREFERRED_PREVIEW_SIZE_FOR_VIDEO,"176x144");
            }
        }
    }

    if ((mCamDriverCapability.version & 0xff) >= 0x07) {
        int tmp0,tmp1;
        if (cameraFramerateQuery(mCamDriverPreviewFmt, 240,160,&tmp1,&tmp0) == 0) {
            if (mCamDriverFrmWidthMax >= 240) {
                fmt.fmt.pix.width = 240;
                fmt.fmt.pix.height = 160;
                if (ioctl(iCamFd, VIDIOC_TRY_FMT, &fmt) == 0) {
                    if ((fmt.fmt.pix.width == 240) && (fmt.fmt.pix.height == 160)) {
                        parameterString.append(",240x160");
                        params.setPreviewSize(240, 160);
                        previewFrameSizeMax =  PAGE_ALIGN(240*160*2)*2;        // 240*160*2     rgb565

                    }
                }
            }
        }
    }

    if (strcmp(cameraCallProcess,"com.tencent.android.pad")) {
        if (mCamDriverFrmWidthMax >= 320) {
            fmt.fmt.pix.width = 320;
```

```
            fmt.fmt.pix.height = 240;
            if (ioctl(iCamFd, VIDIOC_TRY_FMT, &fmt) == 0) {
                if ((fmt.fmt.pix.width == 320) && (fmt.fmt.pix.height == 240)) {
                    parameterString.append(",320x240");
                    params.setPreviewSize(320, 240);
                    previewFrameSizeMax =  PAGE_ALIGN(320*240*2)*2;         // 320*240*2

                }
            }
        }
    }
    if (mCamDriverFrmWidthMax >= 352) {
        fmt.fmt.pix.width = 352;
        fmt.fmt.pix.height = 288;
        if (ioctl(iCamFd, VIDIOC_TRY_FMT, &fmt) == 0) {
            if ((fmt.fmt.pix.width == 352) && (fmt.fmt.pix.height == 288)) {
                parameterString.append(",352x288");
                params.setPreviewSize(352, 288);
                previewFrameSizeMax =  PAGE_ALIGN(352*288*2)*2;         // 352*288*1.5*2

            }
        }
    }

    if (mCamDriverFrmWidthMax >= 640) {
        fmt.fmt.pix.width = 640;
        fmt.fmt.pix.height = 480;
        if (ioctl(iCamFd, VIDIOC_TRY_FMT, &fmt) == 0) {
            if ((fmt.fmt.pix.width == 640) && (fmt.fmt.pix.height == 480)) {
                parameterString.append(",640x480");
                params.setPreviewSize(640, 480);
                previewFrameSizeMax =  PAGE_ALIGN(640*480*2)*2;         // 640*480*1.5*2

            }
        }
    }

    if (mCamDriverFrmWidthMax >= 720) {
        fmt.fmt.pix.width = 720;
        fmt.fmt.pix.height = 480;
        if (ioctl(iCamFd, VIDIOC_TRY_FMT, &fmt) == 0) {
            if ((fmt.fmt.pix.width == 720) && (fmt.fmt.pix.height == 480)) {
                parameterString.append(",720x480");
                previewFrameSizeMax =  PAGE_ALIGN(720*480*2)*2;         // 720*480*1.5*2

            }
        }
    }

    if (mCamDriverFrmWidthMax >= 1280) {
```

```
        fmt.fmt.pix.width = 1280;
        fmt.fmt.pix.height = 720;
        if (ioctl(iCamFd, VIDIOC_TRY_FMT, &fmt) == 0) {
            if ((fmt.fmt.pix.width == 1280) && (fmt.fmt.pix.height == 720)) {
                parameterString.append(",1280x720");
                previewFrameSizeMax =  PAGE_ALIGN(1280*720*2)*2;        // 1280*720*1.5*2

            }
        }
    }
    mSupportPreviewSizeReally = parameterString;
    /* ddl@rock-chips.com: Facelock speed is low, so scale down preview data to facelock for speed up */
    if ((strcmp(cameraCallProcess,"com.android.facelock")==0)) {
        if (strstr(mSupportPreviewSizeReally.string(),"640x480")||
            strstr(mSupportPreviewSizeReally.string(),"320x240")) {
            parameterString = "160x120";
            params.setPreviewSize(160, 120);
        }
    }
    params.set(CameraParameters::KEY_SUPPORTED_PREVIEW_SIZES, parameterString.string());

    strcat(str_picturesize,parameterString.string());
    strcat(str_picturesize,",");
    if(mCamDriverFrmWidthMax <= 640){
        strcat( str_picturesize,"640x480,320x240");
        mRawBufferSize = RAW_BUFFER_SIZE_0M3;
        mJpegBufferSize = JPEG_BUFFER_SIZE_0M3;
        params.setPictureSize(640,480);
    }else if (mCamDriverFrmWidthMax <= 1280) {
        strcat( str_picturesize,"1024x768,640x480,320x240");
        mRawBufferSize = RAW_BUFFER_SIZE_1M;
        mJpegBufferSize = JPEG_BUFFER_SIZE_1M;
        params.setPictureSize(1024,768);
    } else if (mCamDriverFrmWidthMax <= 1600) {
        strcat( str_picturesize,"1600x1200,1024x768,640x480");
        mRawBufferSize = RAW_BUFFER_SIZE_2M;
        mJpegBufferSize = JPEG_BUFFER_SIZE_2M;
        params.setPictureSize(1600,1200);
    } else if (mCamDriverFrmWidthMax <= 2048) {
        strcat( str_picturesize,"2048x1536,1600x1200,1024x768");
        mRawBufferSize = RAW_BUFFER_SIZE_3M;
        mJpegBufferSize = JPEG_BUFFER_SIZE_3M;
        params.setPictureSize(2048,1536);
    } else if (mCamDriverFrmWidthMax <= 2592) {
        strcat( str_picturesize,"2592x1944,2048x1536,1600x1200,1024x768");
        params.setPictureSize(2592,1944);
        mRawBufferSize = RAW_BUFFER_SIZE_5M;
        mJpegBufferSize = JPEG_BUFFER_SIZE_5M;
    } else {
        sprintf(str_picturesize, "%dx%d", mCamDriverFrmWidthMax,mCamDriverFrmHeightMax);
```

```
            mRawBufferSize = RAW_BUFFER_SIZE_5M;
            mJpegBufferSize = JPEG_BUFFER_SIZE_5M;
            params.setPictureSize(mCamDriverFrmWidthMax,mCamDriverFrmHeightMax);
        }

        params.set(CameraParameters::KEY_SUPPORTED_PICTURE_SIZES, str_picturesize);

        /*frame rate setting*/
        cameraFpsInfoSet(params);

        /*zoom setting*/
        struct v4l2_queryctrl zoom;
        char str_zoom_max[3],str_zoom_element[5];
        char str_zoom[200];
        strcpy(str_zoom, "");//default zoom
        int max;

        zoom.id = V4L2_CID_ZOOM_ABSOLUTE;
        if (!ioctl(iCamFd, VIDIOC_QUERYCTRL, &zoom)) {
            mZoomMax = zoom.maximum;
            mZoomMin= zoom.minimum;
            mZoomStep = zoom.step;

            max = (mZoomMax - mZoomMin)/mZoomStep;
            sprintf(str_zoom_max,"%d",max);
            params.set(CameraParameters::KEY_ZOOM_SUPPORTED, "true");
            params.set(CameraParameters::KEY_MAX_ZOOM, str_zoom_max);
            params.set(CameraParameters::KEY_ZOOM, "0");
            for (i=mZoomMin; i<=mZoomMax; i+=mZoomStep) {
                sprintf(str_zoom_element,"%d,", i);
                strcat(str_zoom,str_zoom_element);
            }
            params.set(CameraParameters::KEY_ZOOM_RATIOS, str_zoom);
        }
    }
    /*preview format setting*/
    params.set(CameraParameters::KEY_SUPPORTED_PREVIEW_FORMATS, "yuv420sp,rgb565,yuv420p");
    params.set(CameraParameters::KEY_VIDEO_FRAME_FORMAT,CameraParameters::PIXEL_FORMAT_YUV420SP);
    if (strcmp(cameraCallProcess,"com.android.camera")==0) {    //for PanoramaActivity
        params.setPreviewFormat(CameraParameters::PIXEL_FORMAT_RGB565);
    } else {
        params.setPreviewFormat(CameraParameters::PIXEL_FORMAT_YUV420SP);
    }
    /* zyc@rock-chips.com: preset the displayformat for cts */
    strcpy(mDisplayFormat,CAMERA_DISPLAY_FORMAT_NV12);


    params.set(CameraParameters::KEY_VIDEO_FRAME_FORMAT,CameraParameters::PIXEL_FORMAT_YUV420SP);
```

```
    /*picture format setting*/
    params.set(CameraParameters::KEY_SUPPORTED_PICTURE_FORMATS, CameraParameters::PIXEL_FORMAT_JPEG);
    params.setPictureFormat(CameraParameters::PIXEL_FORMAT_JPEG);

    /*jpeg quality setting*/
    params.set(CameraParameters::KEY_JPEG_QUALITY, "70");

    /*white balance setting*/
    struct v4l2_queryctrl whiteBalance;
    struct v4l2_querymenu *whiteBalance_menu = mWhiteBalance_menu;
    char str_whitebalance[200];
    strcpy(str_whitebalance, "");//default whitebalance
    whiteBalance.id = V4L2_CID_DO_WHITE_BALANCE;
    if (!ioctl(iCamFd, VIDIOC_QUERYCTRL, &whiteBalance)) {
        for (i = whiteBalance.minimum; i <= whiteBalance.maximum; i += whiteBalance.step) {
            whiteBalance_menu->id = V4L2_CID_DO_WHITE_BALANCE;
            whiteBalance_menu->index = i;
            if (!ioctl(iCamFd, VIDIOC_QUERYMENU, whiteBalance_menu)) {
                if (i != whiteBalance.minimum)
                    strcat(str_whitebalance, ",");
                strcat(str_whitebalance, (char *)whiteBalance_menu->name);
                if (whiteBalance.default_value == i) {
                    strcpy(cur_param, (char *)whiteBalance_menu->name);
                }
                mWhiteBalance_number++;
            }
            whiteBalance_menu++;
        }
        params.set(CameraParameters::KEY_SUPPORTED_WHITE_BALANCE, str_whitebalance);
        params.set(CameraParameters::KEY_WHITE_BALANCE, cur_param);
    }

    /*color effect setting*/
    struct v4l2_queryctrl effect;
    struct v4l2_querymenu *effect_menu = mEffect_menu;
    char str_effect[200];
    strcpy(str_effect, "");//default effect
    effect.id = V4L2_CID_EFFECT;
    if (!ioctl(iCamFd, VIDIOC_QUERYCTRL, &effect)) {
        for (i = effect.minimum; i <= effect.maximum; i += effect.step) {
            effect_menu->id = V4L2_CID_EFFECT;
            effect_menu->index = i;
            if (!ioctl(iCamFd, VIDIOC_QUERYMENU, effect_menu)) {
                if (i != effect.minimum)
                    strcat(str_effect, ",");
                strcat(str_effect, (char *)effect_menu->name);
                if (effect.default_value == i) {
                    strcpy(cur_param, (char *)effect_menu->name);
```

```
358.            }
359.            mEffect_number++;
360.        }
361.        effect_menu++;
362.    }
363.    params.set(CameraParameters::KEY_SUPPORTED_EFFECTS, str_effect);
364.    params.set(CameraParameters::KEY_EFFECT, cur_param);
365. }
366.
367. /*scene setting*/
368. struct v4l2_queryctrl scene;
369. struct v4l2_querymenu *scene_menu = mScene_menu;
370. char str_scene[200];
371. strcpy(str_scene, "");//default scene
372. scene.id = V4L2_CID_SCENE;
373. if (!ioctl(iCamFd, VIDIOC_QUERYCTRL, &scene)) {
374.    for (i=scene.minimum; i<=scene.maximum; i+=scene.step) {
375.        scene_menu->id = V4L2_CID_SCENE;
376.        scene_menu->index = i;
377.        if (!ioctl(iCamFd, VIDIOC_QUERYMENU, scene_menu)) {
378.            if (i != scene.minimum)
379.                strcat(str_scene, ",");
380.            strcat(str_scene, (char *)scene_menu->name);
381.            if (scene.default_value == i) {
382.                strcpy(cur_param, (char *)scene_menu->name);
383.            }
384.            mScene_number++;
385.        }
386.        scene_menu++;
387.    }
388.    params.set(CameraParameters::KEY_SUPPORTED_SCENE_MODES, str_scene);
389.    params.set(CameraParameters::KEY_SCENE_MODE, cur_param);
390.
391. }
392.
393. /*flash mode setting*/
394. struct v4l2_queryctrl flashMode;
395. struct v4l2_querymenu *flashMode_menu = mFlashMode_menu;
396. char str_flash[200];
397. strcpy(str_flash, "");//default flash
398. flashMode.id = V4L2_CID_FLASH;
399. if (!ioctl(iCamFd, VIDIOC_QUERYCTRL, &flashMode)) {
400.    for (i = flashMode.minimum; i <= flashMode.maximum; i += flashMode.step) {
401.        flashMode_menu->id = V4L2_CID_FLASH;
402.        flashMode_menu->index = i;
403.        if (!ioctl(iCamFd, VIDIOC_QUERYMENU, flashMode_menu)) {
404.            if (i != flashMode.minimum)
405.                strcat(str_flash, ",");
406.            strcat(str_flash, (char *)flashMode_menu->name);
407.            if (flashMode.default_value == i) {
```

```
                strcpy(cur_param, (char *)flashMode_menu->name);
            }
            mFlashMode_number++;
        }
        flashMode_menu++;
    }
    params.set(CameraParameters::KEY_SUPPORTED_FLASH_MODES, str_flash);
    params.set(CameraParameters::KEY_FLASH_MODE, cur_param);
}

/*focus mode setting*/
struct v4l2_queryctrl focus;

parameterString = CameraParameters::FOCUS_MODE_FIXED;
params.set(CameraParameters::KEY_FOCUS_MODE, CameraParameters::FOCUS_MODE_FIXED);
focus.id = V4L2_CID_FOCUS_AUTO;
if (!ioctl(iCamFd, VIDIOC_QUERYCTRL, &focus)) {
    parameterString.append(",");
    parameterString.append(CameraParameters::FOCUS_MODE_AUTO);
    params.set(CameraParameters::KEY_FOCUS_MODE, CameraParameters::FOCUS_MODE_AUTO);
}

focus.id = V4L2_CID_FOCUS_CONTINUOUS;
if (!ioctl(iCamFd, VIDIOC_QUERYCTRL, &focus)) {
    parameterString.append(",");
    parameterString.append(CameraParameters::FOCUS_MODE_EDOF);
}

focus.id = V4L2_CID_FOCUS_ABSOLUTE;
if (!ioctl(iCamFd, VIDIOC_QUERYCTRL, &focus)) {
    parameterString.append(",");
    parameterString.append(CameraParameters::FOCUS_MODE_INFINITY);
    parameterString.append(",");
    parameterString.append(CameraParameters::FOCUS_MODE_MACRO);
}

params.set(CameraParameters::KEY_SUPPORTED_FOCUS_MODES, parameterString.string());

/*mirror and flip query*/
struct v4l2_queryctrl mirror,flip;

mirror.id = V4L2_CID_HFLIP;
if (!ioctl(iCamFd, VIDIOC_QUERYCTRL, &mirror)) {
    mDriverMirrorSupport = true;
} else {
    mDriverMirrorSupport = false;
}

flip.id = V4L2_CID_VFLIP;
if (!ioctl(iCamFd, VIDIOC_QUERYCTRL, &flip)) {
```

```
        mDriverFlipSupport = true;
    } else {
        mDriverFlipSupport = false;
    }


    /*Exposure setting*/
    struct v4l2_queryctrl exposure;
    char str_exposure[16];
    exposure.id = V4L2_CID_EXPOSURE;
    if (!ioctl(iCamFd, VIDIOC_QUERYCTRL, &exposure)) {
        sprintf(str_exposure,"%d",exposure.default_value);
        params.set(CameraParameters::KEY_EXPOSURE_COMPENSATION, str_exposure);
        sprintf(str_exposure,"%d",exposure.maximum);
        params.set(CameraParameters::KEY_MAX_EXPOSURE_COMPENSATION, str_exposure);
        sprintf(str_exposure,"%d",exposure.minimum);
        params.set(CameraParameters::KEY_MIN_EXPOSURE_COMPENSATION, str_exposure);
        sprintf(str_exposure,"%d",exposure.step);
        params.set(CameraParameters::KEY_EXPOSURE_COMPENSATION_STEP, str_exposure);
    } else {
        params.set(CameraParameters::KEY_EXPOSURE_COMPENSATION, "0");
        params.set(CameraParameters::KEY_MAX_EXPOSURE_COMPENSATION, "0");
        params.set(CameraParameters::KEY_MIN_EXPOSURE_COMPENSATION, "0");
        params.set(CameraParameters::KEY_EXPOSURE_COMPENSATION_STEP, "0.000001f");
    }
    /*rotation setting*/
    params.set(CameraParameters::KEY_ROTATION, "0");

    /*lzg@rockchip.com :add some settings to pass cts*/
    /*focus distance setting ,no much meaning ,only for passing cts */
    parameterString = "0.3,50,Infinity";
    params.set(CameraParameters::KEY_FOCUS_DISTANCES, parameterString.string());
    /*focus length setting ,no much meaning ,only for passing cts */
    parameterString = "35";
    params.set(CameraParameters::KEY_FOCAL_LENGTH, parameterString.string());
   /*horizontal angle of view setting ,no much meaning ,only for passing cts */
    parameterString = "100";
    params.set(CameraParameters::KEY_HORIZONTAL_VIEW_ANGLE, parameterString.string());
    /*vertical angle of view setting ,no much meaning ,only for passing cts */
    parameterString = "100";
    params.set(CameraParameters::KEY_VERTICAL_VIEW_ANGLE, parameterString.string());

   /*quality of the EXIF thumbnail in Jpeg picture setting */
    parameterString = "50";
    params.set(CameraParameters::KEY_JPEG_THUMBNAIL_QUALITY, parameterString.string());
   /*supported size of the EXIF thumbnail in Jpeg picture setting */
    parameterString = "0x0,160x128";
    params.set(CameraParameters::KEY_SUPPORTED_JPEG_THUMBNAIL_SIZES, parameterString.string());
    parameterString = "160";
    params.set(CameraParameters::KEY_JPEG_THUMBNAIL_WIDTH, parameterString.string());
```

```
    parameterString = "128";
    params.set(CameraParameters::KEY_JPEG_THUMBNAIL_HEIGHT, parameterString.string());
    /* zyc@rock-chips.com: for cts ,KEY_MAX_NUM_DETECTED_FACES_HW should not be 0 */
    params.set(CameraParameters::KEY_MAX_NUM_DETECTED_FACES_HW, "0");
    params.set(CameraParameters::KEY_MAX_NUM_DETECTED_FACES_SW, "0");
    params.set(CameraParameters::KEY_RECORDING_HINT,"false");
    params.set(CameraParameters::KEY_VIDEO_STABILIZATION_SUPPORTED,"false");
    params.set(CameraParameters::KEY_VIDEO_SNAPSHOT_SUPPORTED,"true");
    params.set(CameraParameters::KEY_MAX_NUM_METERING_AREAS,"0");

    LOGD ("Support Preview format: %s ",params.get(CameraParameters::KEY_SUPPORTED_PREVIEW_FORMATS));
    LOGD ("Support Preview sizes: %s ",params.get(CameraParameters::KEY_SUPPORTED_PREVIEW_SIZES));
    LOGD ("Support Preview FPS range: %s",params.get(CameraParameters::KEY_SUPPORTED_PREVIEW_FPS_RANGE));
    LOGD ("Support Preview framerate: %s",params.get(CameraParameters::KEY_SUPPORTED_PREVIEW_FRAME_RATES));
    LOGD ("Support Picture sizes: %s ",params.get(CameraParameters::KEY_SUPPORTED_PICTURE_SIZES));
    if (params.get(CameraParameters::KEY_SUPPORTED_WHITE_BALANCE))
        LOGD ("Support white balance: %s",params.get(CameraParameters::KEY_SUPPORTED_WHITE_BALANCE));
    if (params.get(CameraParameters::KEY_SUPPORTED_EFFECTS))
        LOGD ("Support color effect: %s",params.get(CameraParameters::KEY_SUPPORTED_EFFECTS));
    if (params.get(CameraParameters::KEY_SUPPORTED_SCENE_MODES))
        LOGD ("Support scene: %s",params.get(CameraParameters::KEY_SUPPORTED_SCENE_MODES));
    if (params.get(CameraParameters::KEY_SUPPORTED_FLASH_MODES))
        LOGD ("Support flash: %s",params.get(CameraParameters::KEY_SUPPORTED_FLASH_MODES));
    LOGD ("Support focus: %s",params.get(CameraParameters::KEY_SUPPORTED_FOCUS_MODES));
    LOGD ("Support zoom: %s(ratios: %s)",params.get(CameraParameters::KEY_ZOOM_SUPPORTED),
        params.get(CameraParameters::KEY_ZOOM_RATIOS));
    if (strcmp("0", params.get(CameraParameters::KEY_MAX_EXPOSURE_COMPENSATION))
        || strcmp("0", params.get(CameraParameters::KEY_MIN_EXPOSURE_COMPENSATION))) {
        LOGD ("Support exposure: (%s -> %s)",params.get(CameraParameters::KEY_MIN_EXPOSURE_COMPENSATION),
            params.get(CameraParameters::KEY_MAX_EXPOSURE_COMPENSATION));
    }
    LOGD ("Support hardware faces detecte: %s",params.get(CameraParameters::KEY_MAX_NUM_DETECTED_FACES_HW));
    LOGD ("Support software faces detecte: %s",params.get(CameraParameters::KEY_MAX_NUM_DETECTED_FACES_SW));
    LOGD ("Support video stabilization: %s",params.get(CameraParameters::KEY_VIDEO_STABILIZATION_SUPPORTED));
    LOGD ("Support recording hint: %s",params.get(CameraParameters::KEY_RECORDING_HINT));
    LOGD ("Support video snapshot: %s",params.get(CameraParameters::KEY_VIDEO_SNAPSHOT_SUPPORTED));
    LOGD ("Support Mirror and Filp: %s",(mDriverMirrorSupport && mDriverFlipSupport)? "true":"false");

    cameraConfig(params);
    LOG_FUNCTION_NAME_EXIT

}
```

然后剩下的大部分都是针对这个线程的运行实现以及对于CameraHal_Module.cpp中实现的为上层提供的接口的具体实

现，比如：

[cpp] view plaincopy

```
1. int CameraHal::startPreview()
2. {
3.     LOG_FUNCTION_NAME
4.     Message msg;
5.     Mutex::Autolock lock(mLock);
6.
7.     if ((mPreviewThread != NULL) && (mCommandThread != NULL)) {
8.         msg.command = CMD_PREVIEW_START;
9.         msg.arg1 = (void*)CMDARG_NACK;
10.        commandThreadCommandQ.put(&msg);
11.    }
12.    mPreviewCmdReceived = true;
13.    LOG_FUNCTION_NAME_EXIT
14.    return NO_ERROR ;
15. }
16.
17. void CameraHal::stopPreview()
18. {
19.    LOG_FUNCTION_NAME
20.    Message msg;
21.    int ret = 0;
22.    Mutex::Autolock lock(mLock);
23.
24.    if ((mPreviewThread != NULL) && (mCommandThread != NULL)) {
25.        msg.command = CMD_PREVIEW_STOP;
26.        msg.arg1 = (void*)CMDARG_ACK;
27.        commandThreadCommandQ.put(&msg);
28.
29.        if (mANativeWindow == NULL) {
30.            mANativeWindowCond.signal();
31.            LOGD("%s(%d): wake up command thread for stop preview",__FUNCTION__,__LINE__);
32.        }
33.
34.        while (ret == 0) {
35.            ret = commandThreadAckQ.get(&msg);
36.            if (ret == 0) {
37.                if (msg.command == CMD_PREVIEW_STOP) {
38.                    ret = 1;
39.                }
40.            }
41.        }
42.    } else {
43.        LOGE("%s(%d): cancel, because thread (%s %s) is NULL", __FUNCTION__,__LINE__,
(mPreviewThread == NULL)?"mPreviewThread":" ",
44.            (mCommandThread == NULL)?"mCommandThread":" ");
45.    }
46.    mPreviewCmdReceived = false;
47.    LOG_FUNCTION_NAME_EXIT
```

```
48. }
49.
50. int CameraHal::autoFocus()
51. {
52.     LOG_FUNCTION_NAME
53.     int ret = 0;
54.     Message msg;
55.     Mutex::Autolock lock(mLock);
56.
57.     if ((mPreviewThread != NULL) && (mCommandThread != NULL)) {
58.         msg.command = CMD_AF_START;
59.         msg.arg1 = (void*)CMDARG_ACK;
60.         commandThreadCommandQ.put(&msg);
61.         while (ret == 0) {
62.             ret = commandThreadAckQ.get(&msg,5000);
63.             if (ret == 0) {
64.                 if (msg.command == CMD_AF_START) {
65.                     ret = 1;
66.                 }
67.             } else {
68.                 LOGE("%s(%d): AutoFocus is time out!!!\n",__FUNCTION__,__LINE__);
69.             }
70.         }
71.     } else {
72.         LOGE("%s(%d):  cancel, because thread (%s %s) is NULL", __FUNCTION__,__LINE__,
(mPreviewThread == NULL)?"mPreviewThread":" ",
73.             (mCommandThread == NULL)?"mCommandThread":" ");
74.     }
75.     LOG_FUNCTION_NAME_EXIT
76.     return NO_ERROR;
77. }
```

[cpp] view plaincopy

```
1. "code" class="cpp">"code" class="cpp">"code" class="cpp">

2.
3.
4.
5.
6.
7.
8.
9.
10.
11.
12.
13.
14.
15.
16.
```

17.

阅读(9021) | 评论(0) | 转发(0) |

上一篇：andorid之摄像头驱动流程--MTK平台
下一篇：android系统开发(六)-HAL层开发基础

0

相关热门文章

Android之开发环境搭建	linux dhcp peizhi roc
Android自定义View的实现...	关于Unix文件的软链接
AndroidManifest.xml配置文件...	求教这个命令什么意思，我是新...
Android相对布局+圆角按钮+Sha...	sed -e "/grep/d" 是什么意思...
查看Android应用包名package和...	谁能够帮我解决LINUX 2.6 10...

给主人留下些什么吧！~~

评论热议

请登录后评论。
登录 注册

1 android3	4 控制算法	7 win7纯净版	10 数据流
2 短信接口api	5 上海徐家汇房价	8 html5开发app	11 mac
3 awb	6 android下载	9 网页文件夹	12 上海市房