```
//
//  IPCam+IPCamVideo.java
//  SoDemon
//
------------------------------------------------------
```

#1.摄像机的解绑和注册

摄像机的解绑和注册

说明：当需要账号管理摄像机时，才会用到解绑和注册功能。大概流程是，添加时，先判断有没有被其他账号添加，如果被添加了，就先要从服务器解绑。解绑成功后，在用WebService的添加函数添加，添加成功后。再注册的到服务器上。

1）解绑

说明：解绑即从账号里把该摄像机删除。同一个摄像机只能被一个账号添加。

```
/*
 *摄像机解绑 + 实现接口unregister_from_sosocam_listener
 */
```

函数：public ERROR unregister_from_sosocam(unregister_from_sosocam_listener listener);

返回：ERROR:ERROR类型值，当返回是ERROR.NO_ERROR，表示执行成功。

回调函数：

```
public void on_result(IPCam ipcam, ERROR error);
```

ipcam：IPCam对象

error：ERROR类型值;

回调函数返回的error = ERROR.NO_ERROR时，表示解绑成功

2）注册

说明：注册即把摄像机向服务器注册。只有注册成功了才会有推送。

```
/*
 *摄像机注册 + 实现接口relogin_to_sosocam_listener
 */
public ERROR relogin_to_sosocam(relogin_to_sosocam_listener listener);
```

返回：ERROR类型值，当返回是ERROR.NO_ERROR，表示执行成功。

回调函数：

```
public void on_result(IPCam ipcam, ERROR error);
```

ipcam：IPCam对象

error：ERROR类型值;

回调函数返回error = ERROR.NO_ERROR，表示注册成功。

3）示例

```
if (ERROR.NO_ERROR ==
m_ipcam.unregister_from_sosocam(AddCamerabySoundDialog.this)) {//摄像机解绑
    m_state = STATE.UNREGISTER_CAMERA;
}else {
    Log.e("Sodemo","Failed unregister camera");
    }
if(ERROR.NO_ERROR ==
m_ipcam.relogin_to_sosocam(AddCamerabySoundDialog.this)){//摄像机注册
    m_state = STATE.RELOGIN_CAMERA_TO_SOSOCAM;
}else{
    Log.e("Sodemo","Failed relogin to sosocam");
}

public void on_result(IPCam ipcam, ERROR error) {//回调查看解绑和注册结果
    if (m_state == STATE.UNREGISTER_CAMERA) {
        if (error == ERROR.NO_ERROR) {//解绑结果
            Log.e("Sodemo","Unregister camera succeed");
        } else {
            Log.e("Sodemo","Unregister camera failed");
        }
    }else if(m_state == STATE.RELOGIN_CAMERA_TO_SOSOCAM){
        if (error == ERROR.NO_ERROR) {//注册结果
            Log.e("Sodemo","Relogin to sosocam succeed");
        } else {
            Log.e("Sodemo","Relogin to sosocam failed");
        }
    }
}
```

#2. 摄像机属性添加设置和获取

1）摄像机属性添加设置和获取

说明：摄像机属性设置，包括名称，用户名，名称，id等摄像机的相关属性参数。这些属性，只有当你往IPCam添加了，IPCam的创建的对象才具备这些属性（比如：名称，你必须先set_alias了，才能获取到名称）。所以你需要用到哪些属性，当你从stroage里面获取cam里时或添加cam时，就一定要为cam添加哪些属性，否则获取到的都是默认值。当然如果这些属性有改变时，也要同步设置到IPCam里。

函数：主要属性函数，详细如下

```
/*
 *名称，默认为空
 */
public String alias(); //获取昵称
public void set_alias(String alias); //设置昵称

/*
 *用户名，默认为空
 */
public String user();//获取登录名
public void set_user(String user); //设置登录名

/*
 *密码设置，默认为空
 */
public String pwd(); //获取机登录密码
public void set_pwd(String pwd); //设置登录密码

/*
 ＊摄像机id，默认为空
 */
public String camera_id(); //获取id
public void set_id(String id); //设置id

/*
 *摄像机sosocam_id,为int值，未提供函数进行设置和获取，直接对其值进行操作
 *默认为空
 */
public String sosocam_id = "";//sosocam_id

/*
 *摄像机安全传输，默认为False
 */
public boolean https(); //获取https
public void set_https(boolean https); //设置https

/*
 *摄像机模式，int值，未提供函数进行设置和获取，直接对其值进行操作和设置
 *model默认为0，有两个值，0 云台机  1 卡片机
 */
public int model = 0;; //model

/*
 *预览图，Bitmap值，未提供函数进行设置和获取，直接取ipcam.cover即可
 *cover默认为null
 */
public Bitmap cover = null; //预览图

/*
 *摄像机无线Wi-Fi信号强度，返回int值，范围［0，100］
 */
public int wifi_power(); //获取摄像机连接的wifi信号强度

public void set_cache(int cache);//设置缓冲时间
```

示例：

```
/*

 *以下示例，是在程序开始时，从Storage取出cams，同时添加到IPCamMgr，使这些摄像机得到IPCamMgr的批量管理。并且为ipcam添加几个属性
 */
public void load_all_cameras{
   for(CAMERA_INFO cam : Storage.get_cameras()) {
    IPCam ipcam = IPCamMgr.add_camera(cam.getAlias(), cam.getId(),
    cam.getUser(), cam.getPwd(), cam.getHttps());
    ipcam.model = cam.getModel(); //设置模式
```

```
            ipcam.id_4_sosocam = cam.getObj_id(); //设置sosocam_id
            ipcam.cover = cam.getCover(); //设置预览图
            }
        }

#3.视频性能调节
    1）视频性能调节
        函数1：public boolean can_set_video_performance();//判断是否允许设置

        返回：返回一个BOOL类型，如果是true,说明允许操作，如果是false,说明禁止操作，因为视频调节具有优先级，先操作的人具有优先权操作。

        函数2:public void set_video_performance_mode(int mode) //设置性能值
        参数：mode有三个值，详细如下
        mode = 0,  //清晰，质量最好，速度最慢
        mode = 1, //均衡，质量和速度都处于均值
        mode = 2,  //流畅，质量最差，速度最快

        回调函数：public void on_can_set_video_performance(IPCam ipcam);;
        说明：视频性能状态改变，接口是IPCam_Listener

        示例：
        int value = ipcam.video_performance_mode();//获取当前性能值
        case R.id.play_speed:
            if (m_ipcam.can_set_video_performance()){//判断是否允许设置
                m_ipcam.set_video_performance_mode(1);//设置性能值
            }else{
                Log.e("Sodemo","--No--operation-permission---");
            }

#4.缓冲时间设置
    1）缓冲时间设置

        说明：缓冲时间对视频和音频生效，就是推迟多少时间再显示或发声。使用以下函数时，必须保证摄像机是连接成功的。
        函数：public int cache(); //获取当前缓冲值，单位ms
        public void set_cache(int cache); //设置当前缓冲值

        参数：cache缓冲时间，它的值分两种，详细如下：
        局域网：[0-1000],单位:ms
        p2p:[1000-8000],单位:ms
        示例：
        ipcam.cache();  //获取缓冲时间
        ipcam.set_cache(1000); //设置缓冲时间

#5.名称设置
    1）名称设置

        说明：名称设置，是摄像机的一个昵称，设置成功以后，需要保存更新到本地，如果摄像机要保存到服务器，还需要更新保存到服务器。使用以下函数时，必须保证摄像机是连接成功的。
        函数：public String alias(); //获取名称
        public void set_alias(String alias) ; //设置名称


        说明：当摄像机的名称发生改变时，调用此函数，使用说明请参考［2.2摄像机基础接口］
        回调函数：public void on_alias_changed(IPCam ipcam);

        示例：
        m_ipcam.alias(); //获取名称
        public void set_name(){
            m_ipcam.set_alias(m_camera_name);//设置名称
            Storage.update_camera_alias(m_camera_id, m_camera_name);
            Storage.save_cameras();
        }

#6.重置新密码：

    说明：重置密码，是指重新设置一个新的摄像机登录密码。和设置密码不同。使用以下函数时，必须保证摄像机是连接成功的。

    1）重置密码
```

说明：重新给摄像机设置一个新的密码，使用重置密码函数的条件是摄像机必须连接成功。

函数：public ERROR reset_pwd(String pwd, reset_pwd_listener listener)

实现接口:reset_pwd_listener;

参数：pwd:需设置的新密码

2）回调

说明：用reset_pwd函数，实现接口reset_pwd_listener，当有结果时，就会掉用public void on_result(IPCam ipcam, ERROR error)函数。

函数：public void on_result(IPCam ipcam, ERROR error);

error:ERROR类型值;

返回：返回两个参数，详细如下：

ipcam：返回一个IPCam类型，就是一个IPCam对象;

error：ERROR类型值，当返回是ERROR.NO_ERROR时，表示重置密码成功

示例：

```
if (ERROR.NO_ERROR == m_ipcam.reset_pwd(m_pwd, this)){ //重置密码
    m_state = STATE.SET_CAMERA;
}
public void on_result(IPCam ipcam, ERROR error) {
    if (m_state == STATE.SET_CAMERA) {
        if (error == ERROR.NO_ERROR) {
            Storage.update_camera_pwd(m_ipcam.id(), m_ipcam.pwd());
            Storage.save_cameras();
        }else{
            Log.e("Sodemo","----Failed---to----reset--pwd--");
        }
    }
}
```

#7.系统固件升级

说明：系统固件升级主要介绍：新版本检测，当前固件版本号的获取和最新固件版本号的获取，以及实现升级功能。

1）新版本检测

说明:新版本检测函数使用的条件是摄像机必须连接成功。

函数：public boolean need_upgrade()

返回：返回一个BOOL，返回True表示有新版本，需要升级；返回False表示摄像机当前固件版本已经是最新版本，不需要升级。

2）版本号获取

说明，以下两个函数使用的条件都是摄像机必须连接成功。

```
函数：public String current_fw_version(); //获取摄像机当前的固件版本号
public String latest_fw_version(); //获取服务器上最新的固件版本号
```

返回：两个函数都是返回NSString类型，即版本号。

3）升级功能实现

说明：用upgrade_fw函数，实现接口upgrade_fw_listener，当有结果时，就会掉用public void on_result(IPCam ipcam, ERROR error)函数，使用升级函数的条件是摄像机必须连接成功。

```
函数：public ERROR upgrade_fw(upgrade_fw_listener listener) //升级函数
回调函数：public void on_result(IPCam ipcam, ERROR error)
返回：error:ERROR类型值，当返回是ERROR.NO_ERROR，表示升级成功，否则升级失败

示例：
/*检测当前版本是否是最新版本*/
public void checkNewVersion{
    if(m_ipcam.can_upgrade()){
        /*检测到有新版本*/
        Log.e("Sodemo","delect new version,new version is :" +
        ipcam.latest_version()); //获取最新版本号
        if (ERROR.NO_ERROR != m_ipcam.upgrade_fw(CameraUpgradeDialog.this))
        {//升级新版本
            Log.e("Sodemo","upgrading_failed");
        } else {
            Log.e("Sodemo","upgrading now");
            m_state = STATE.UPGRADE;
        }
    }else{
        /*没有发现新版本*/
        Log.e("Sodemo","version:" + ipcam.fw_version() + "is lastest
        !");//获取当前固件版本号
    }
}
```

```
        /*执行升级动作且实现相应接口implements IPCam.upgrade_fw_listener*/
        /*升级完以后的回调*/
        public void on_result(IPCam ipcam, ERROR error) {
            if (error == ERROR.NO_ERROR) {
                Log.e("Sodemo","upgrading ok");
            } else {
                Log.e("Sodemo","upgrading failed");
            }
        }
```

#8.设置安全传输

1）获取摄像机当前https值

说明：使用以下函数的条件摄像机必须连接成功。

函数：public boolean https()

返回：返回一个BOOL类型，当返回true表示当前是安全传输的，返回false表示当前不是安全传输的。

2）安全设置实现

说明：使用reset_https函数，实现接口reset_https_listener，当有结果时，就会回调public void on_reset_https_result(IPCam ipcam, ERROR error);函数，使用安全设置函数的条件是摄像机必须连接成功。

函数：public ERROR reset_https(boolean https, reset_https_listener listener)

listener:reset_https_listener; //安全设置监听

参数：https = true,设置成安全传输；https = false,设置成非安全传输；

回调函数：public void on_reset_https_result(IPCam ipcam, ERROR error);

示例：

```
        /* 获取摄像机当前https*/
        BOOL m_https = m_ipcam.https();

        /* 执行设置https功能*/
        m_button_set.setOnClickListener(new View.OnClickListener() {
            public void onClick(View arg0) {
                if (m_https == m_ipcam.https()) {
                    return;
                }
            if (ERROR.NO_ERROR == m_ipcam.reset_https(m_https,
            SecuritySettingDialog.this)) {
                Log.e("SoDemo","---Succeed--update-security--setting--");
            } else {
                Log.e("SoDemo","---Failed--update-security--setting--");
                }
            }
        });
        /*设置完以后的回调*/
        public void on_reset_https_result(IPCam ipcam, ERROR error) {
            if (error == ERROR.NO_ERROR) {
                Storage.update_camera_https(m_ipcam.id(), m_ipcam.https());
                Storage.save_cameras();
                Log.e("SoDemo","update_security_setting_ok");
            } else {
                Log.e("SoDemo","failed_update_security_setting");
            }
        }
```

#9.恢复出厂参数

说明：恢复出厂参数是指把摄像机里面的参数恢复到出厂时的参数，使用以下函数时，必须保证摄像机是连接成功的。

1）恢复出厂

说明：恢复参数成功后，需要重启才生效［重启是通过cgi实现的］

函数：public ERROR restore_factory(restore_factory_listener listener); //

2）回调

说明：如果执行恢复参数时候，实现了restore_factory_listener代理，那么当恢复出厂有结果后，就会调用此回调函数。

函数：public void on_restore_factory_result(IPCam ipcam, ERROR error);

返回：返回两个参数，详细如下：

ipcam：返回一个IPCam对象

error：ERROR类型值，当返回是ERROR.NO_ERROR，表示恢复出厂成功。

示例：

```
        //恢复出厂参数，且实现接口
```

```
        m_button_restore.setOnClickListener(new View.OnClickListener() {
            public void onClick(View arg0) {
                IPCam ipcam = IPCamMgr.get_camera(m_camera_id);
                if (ipcam == null || ERROR.NO_ERROR !=
                ipcam.restore_factory(RestoreCameraDialog.this)) {
                    Log.e("SoDemo","failed_restore_camera");
                } else {
                    Log.e("SoDemo","succeed_restore_camera");
                }
            }
        });

        /*恢复出厂的回调*/
        public void on_restore_factory_result(IPCam ipcam, ERROR error) {
            if (ERROR.NO_ERROR == error){
                if (ERROR.NO_ERROR != ipcam.set_params("reboot=1", this)) {
                    Log.e("SoDemo","failed_reboot_camera");
                } else {
                    Log.e("SoDemo","succeed_reboot_camera");
                    }
            }else{
                Log.e("SoDemo","failed_restore_camera");
                }
            }
```

#10.IPCam类和IPCamVideoView类

说明：IPCam类是管理单个cam,录像，拍照，对讲等各种功能的实现以及监控cam各种状态，连接，视频，音频等。

使用步骤如下：

1. 初始化:通过IPCamMgr的init函数初始化。
2. 创建对象：new一个IPCam。
3. 实现接口：实现摄像机的IPCam_Listener接口。[这一步不是必须的，但是如果想随时监控连接状态，就必须要进行连接操作前实现此接口]
4. 判断是否摄像机连接：通过status获取摄像机当前状态，如果是CONN_STATUS.IDLE，通过ERROR获取错误原因。
5. 连接成功后相关可行操作：如果status是CONN_STATUS.CONNECTED，表示连接成功，就可以进行名称，密码，布防等各种操作，也可以获取到接状态，报警状态的获取.
6. 视频打开成功后相关可行操作：如果想要进行拍照，录像，看视频，视频性能，缓冲，分辨率切换，轨迹设置，启动位设置等相关操作，必须是视频状态是PLAY_STATUS.PLAYING。所以必须先通过set_video_view设置视频view，然后在播放视频。视频播放成功，才可进行这些操作。

IPCamVideoView类是一个视频类，继承SurfaceView,主要是用的地方就两个：

1） 看视频：new一个IPCamVideoView,添加监听，然后通过IPCam的set_video_view设置视频。

2） 视频上添加手势：如果想要视频上添加手势，就添加IPCamVideoView的set_listener添加代理IPCamVideoView_Listener。在代理on_touch_event中实现功能。

————————

10.1cam状态获取

1）连接状态

说明：一般对摄像机做操作前，都要判断一下摄像机的连接状态

函数：public CONN_STATUS status();

返回：返回CONN_STATUS类型，状态如下：

```
public enum CONN_STATUS {
        IDLE,    //连接失败
        P2P_CONNECTING, //P2P连接中
        CONNECTING,  //局域网连接中
        AUTHING, //摄像机连接认证中
        CONNECTED, //已连接
        WAIT_CONNECTING; //等待连接
    }
```

示例：

```
CONN_STATUS status = ipcam.status();
```

2）摄像机错误

说明：一般当摄像机的连接状态status＝ CONN_STATUS.IDLE，才会去获取错误值

函数：public ERROR error();

返回：返回ERROR类型值，状态如下：

```
public enum ERROR {
//NETWORK_ERROR,    removed in 2.0, using DEVICE_BROKEN
```

```
        //P2P_DISCONNECTED, removed in 2.0, using RUDP_CLOSED
        //P2P_NETWORK_ERROR,    removed in 2.0
        //P2P_OUTOF_CAPACITY,   removed in 2.0

        NO_ERROR(0), //没有错误
        INTERNAL_ERROR(-1),  //内部错误
        CANCELED(-2),   //  new in  2.0，取消
        BAD_PARAM(-3),  //参数错误
        BAD_STATUS(-4),  //状态错误
        BAD_AUTH(-5),   //认证错误
        BAD_ID(-6),     //ID有误
        RESTART_CONN(-7),   //  new in  2.0 ，重新连接
        UNKNOWN(-8),       //未知错误
        CLOSED_BY_DEVICE(-9),   //设备端主动关闭
        DEVICE_BROKEN(-10), //  new in  2.0，设备断开连接
        DEVICE_BAD_PACKET(-11), //  new in  2.0，设备包错误
        DEVICE_NO_DISCOVERED(-12),  //  new in  2.0 未发现设备
        DEVICE_TOO_MANY_SESSIONS(-13), //设备返回会话太多的错误
        DEVICE_INTERNAL_ERROR(-14),  //设备内部错误
        DEVICE_BAD_PARAM(-15),   //设备参数有误
        DEVICE_FORBIDDEN(-16),   //设备禁止连接
        DEVICE_BAD_STATUS(-17),   //设备状态有误
        DEVICE_OPERATION_FAIL(-18), //设备操作失败
        DEVICE_DONE(-19),   //  new in  2.0
        RELAY_TIMEOUT(-20), //  new in  2.0,重连超时
        RELAY_PEER_TIMEOUT(-21),    //  new in  2.0,
        RELAY_CLOSED(-22),  //  new in  2.0，重连关闭
        RELAY_TOO_MANY_SESSIONS(-23),   //  new in  2.0，重连返回会话太多的错误
        P2P_INVALID_ID(-24),  //非法的P2P ID
        P2P_DEVICE_OFFLINE(-25),  //远程机器不在线
        P2P_TIMEOUT(-26), //远程连接超时
        P2P_PEER_TIMEOUT(-27),  //  new in  2.0，同行超时
        P2P_TOO_MANY_SESSIONS(-28), //远程连接会话数已达上限
        P2P_SERVER_ERROR(-29), //P2P服务器错误
        P2P_UNKNOWN(-30),   //  new in  2.0，P2P未知错误
        RUDP_TIMEOUT(-31),  //  new in  2.0,
        RUDP_CLOSED(-32),   //  new in  2.0
        HTTP_GET_ERROR(-33),
        DEVICE_TIMEOUT(-34),
        DEVICE_BAD_REQUEST(-35),
        UPGRADE_BAD_FILE(-36), //升级文件有误
        UPGRADE_BAD_SERVER(-37), //升级服务器有误
        UPGRADE_DOWNLOAD_FAILED(-38), //升级文件下载失败
        SOSOCAM_BAD_ID(-39),  //搜搜平台ID不合法
        SOSOCAM_BAD_ACCESS(-40),
        SOSOCAM_UNREGISTERED(-41),  //sosocam未注册
        RECORD_UNSUPPORT_CODEC(-42),    //  new in  2.0，录像格式不支持
        RECORD_DISK(-43),   //  new in  2.0，录像磁盘已满
        RECORD_PARAM_CHANGED(-44);  //  new in  2.0，录像参数有改变

        public int value;
        private ERROR(int value) {
            this.value = value;
        }
    }
    示例：ERROR camError = m_ipcam.error();
          int errorValue = camError.value;
```

10.2.摄像机基础接口

说明：一般创建了IPCam类，就需实现IPCam_Listener接口，这是IPCam的一个基本接口。add_listener和remove_listener是成对出现的，有add_listener必须有remove_listener，同时回调的那些函数，一旦add_listener了，就必须全部写出来，即使什么功能都不实现，不然程序无法通过编译。

1）添加监听

说明：使用此函数，就添加了IPCam_Listener监听，当相关信息发生改变时，就会反馈到回调函数中。

函数：public void add_listener(IPCam_Listener listener);

2）回调函数

说明：回调函数，主要是相关信息改变时，回调，详细如下：函数：

```
    public void on_alias_changed(IPCam ipcam);//摄像机名称改变
```

```
449
450         public void on_cover_changed(IPCam ipcam); //摄像机封面改变
451
452         public void on_camera_can_upgrade(IPCam ipcam);//摄像机可升级状态改变
453
454         public void on_status_changed(IPCam ipcam);//摄像机连接状态改变
455
456         public void on_video_status_changed(IPCam ipcam);//摄像机视频状态改变
457
458         public void on_audio_status_changed(IPCam ipcam);//摄像机音频状态改变
459
460         public void on_speak_status_changed(IPCam ipcam);//摄像机对讲状态改变
461
462         public void on_local_record_status_changed(IPCam ipcam);//本地录像状态改变
463
464         public void on_tf_record_status_changed(IPCam ipcam);//TF卡录像状态改变
465
466         public void on_tf_record_event(IPCam ipcam, boolean new_record, int
            record_id, boolean error);
467
468         public void on_statistic(IPCam ipcam);
469
470         public void on_camera_tf_changed(IPCam ipcam);
471
472         public void on_camera_tf_capacity(IPCam ipcam);//TF卡
473
474         public void on_camera_wifi_changed(IPCam ipcam);//WiFi信号强弱改变
475
476         public void on_camera_recording_changed(IPCam ipcam);//录像状态改变
477
478         public void on_camera_alarm_changed(IPCam ipcam);   //报警状态改变
479
480         public void on_camera_arm_changed(IPCam ipcam);//布防状态改变
481
482         public void on_camera_temperature_changed(IPCam ipcam);//温度传感器改变
483
484         public void on_camera_sessions_changed(IPCam ipcam);//IPCam会话数改变
485
486         public void on_camera_bell_changed(IPCam ipcam);//门铃状态改变
487
488         public void on_camera_power_down_changed(IPCam ipcam);//电压状态改变
489
490         public void on_camera_dijia_power_changed(IPCam ipcam);//客户定制版本
491
492         public void on_camera_dijia_speed_changed(IPCam ipcam);//客户定制版本
493
494         public void on_camera_dijia_mute_changed(IPCam ipcam);//客户定制版本
495
496         public void on_camera_dijia_status_changed(IPCam ipcam);//客户定制版本
497
498         public void on_camera_meijing_play_changed(IPCam ipcam);//客户定制版本
499
500         public void on_camera_meijing_led_changed(IPCam ipcam);//客户定制版本
501
502         public void on_camera_ewig_status_changed(IPCam ipcam);//客户定制版本
503
504         public void on_camera_juyang_led_changed(IPCam ipcam);//客户定制版本
505
506         public void on_camera_rf_changed_changed(IPCam ipcam);//客户定制版本
507
508         public void on_camera_working_scenes_changed(IPCam ipcam);//客户定制版本
509
510     3）删除代理
511         说明：当不需要监控回调函数里面的这些状态时，就要删除这个代理。
512         函数：public void remove_listener(IPCam_Listener listener);
513
514         示例：
515         /* 添加监听*/
516         m_ipcam.add_listener(this);
517         /*回调实现*/
518         public void on_alias_changed(IPCam ipcam) {
519             Log.e("SoDemo","camera alias change to :" + ipcam.alias);
520         }
```

```
        .
        .
        .
        public void on_camera_recording_changed(IPCam ipcam){
            Logl.e("SoDemo","camera recording changed");
        }
        /*不需要IPCam_Listener，一定要记得删除*/
        m_ipcam.remove_listener(this);;
```

10.3.摄像机视频

说明：摄像机视频，主要是视频设置，视频状态获取，视频开关，以及分辨率切换，使用此函数之前，摄像机必须是连接成功的

1）设置视频页面

说明：通过此函数，可以设置视频页面［之前先要创建添加一个IPCamVideoView，以及设置视频页面］，使用此函数之前，摄像机状态必须是连接成功的。

函数：public void set_video_view(IPCamVideoView view); //设置视频页面

参数：view：这是一个IPCamVideoView，IPCamVideoView是一个继承SurfaceView.

2）视频状态

说明：通过此函数，可以获取到视频状态，使用此函数之前，摄像机必须是连接成功的。

函数：public PLAY_STATUS video_status();

返回：返回PLAY_STATUS类型，类型状态如下：

```
public enum PLAY_STATUS {
    STOPPED, //停止
    REQUESTING, //请求中
    PLAYING; //播放中
}
```

3）视频质量参数

说明：通过以下函数，可以获取到视频当前的质量参数，使用以下函数之前，摄像机必须是连接的。

函数：

```
public int video_render_fps(); //视频发送fps
public int video_recv_fps(); //视频接收的fbs
public int video_byterate(); //视频的byte
```

4）播放视频或切换播放视频流

说明：通过此函数，可以实现播放视频或切换播放视频流功能，使用此函数之前，摄像机必须是连接成功的。

函数：public ERROR play_video(int stream) ;

参数：stream 是视频流参数（0或1），3518e只有两个分辨率，stream = 0 640*360

标清；stream = 1, 1280*720 高清。

返回：返回ERROR类型值，当返回类型是ERROR.NO_ERROR时，表示开启播放视频或者切换视频流成功。

5）停止播放视频

说明：通过此函数，可以实现停止播放视频功能，使用此函数之前，摄像机必须是连接成功的。

函数：public void stop_video();

6）视频监听

说明：视频接口主要有两个，一个是IPCam_Listener，请参考［摄像机基础监听］,另外一个是IPCamVideoView里的IPCamVideoView_Listener，如果你想要在视频上[注意是IPCamVideoView上不是IPCam]添加手势监听，就需实现此接口。

函数：public void set_listener(IPCamVideoView_Listener listener);

回调：public void on_touch_event(TOUCH_EVENT event);

回调返回参数：返回TOUCH_EVENT参数，详细如下：

```
public enum TOUCH_EVENT {
            CLICK,   //单点
            MOVE_UP, //向上滑动
            MOVE_DOWN, //向下滑动
            MOVE_LEFT, //向左滑动
            MOVE_RIGHT;//向右滑动
        }
```

示例：

```
/*创建一个IPCamVideoView设置大小，再设置视频页面*/
IPCamVideoView m_video_view =
(IPCamVideoView)findViewById(R.id.LiveVideoView);
m_ipcam.set_video_view(m_video_view);//设置视频页面
```

```
578
579         /*在视频上实现接口IPCamVideoView_Listener*/
580         m_video_view.set_listener(this);
            //m_video_view添加IPCamVideoView_Listener监听，而不是m_ipcam
581         /* 视频播放和停止*/
582         case R.id.play_start:
583             {
584                 if (m_ipcam.video_status() == PLAY_STATUS.PLAYING) {
                    //判断当前视频状态
585                         m_ipcam.stop_video();// 停止播放视频
586                 }else{
587
                        m_ipcam.play_video(m_playing_video_stream);//播放m_playing_video_s
                        tream路视频流
588                 }
589             }
590
591         /*设置播放视频流，即高清和标清设置或切换 */
592         case R.id.play_change_screen:
593             {
594                 if (m_playing_video_stream == 0){
595                     m_playing_video_stream = 1;
596                     m_ipcam.stop_video();
597                     m_ipcam.play_video(m_playing_video_stream);
598                 }else{
599                     m_playing_video_stream = 0;
600                     m_ipcam.stop_video();
601                     m_ipcam.play_video(m_playing_video_stream);
602                     }
603             }
604         /*IPCamVideoView_Listener回调*/
605         public void on_touch_event(IPCamVideoView_Listener.TOUCH_EVENT event) {
606                 switch (event) {
607                 case CLICK :
608                     Log.e("SoDemo","单点了一下");
609                     break;
610                 case MOVE_UP:
611                     Log.e("SoDemo","向上移动了一下");
612                     break;
613                 case MOVE_DOWN:
614                     Log.e("SoDemo","向下移动了一下");
615                     break;
616                 case MOVE_LEFT:
617                     Log.e("SoDemo","向左移动了一下");
618                     break;
619                 case MOVE_RIGHT:
620                     Log.e("SoDemo","向右移动了一下");
621                     break;
622                 default:
623                     break;
624                 }
625             }
626
627 10.4，摄像机音频
628 说明：摄像机音频，主要包括音频状态获取，音频开启和关闭操作。操作以下函数前，摄像机必须是连接成功的。
629     1）音频状态
630         说明：通过此函数，可以获取到音频状态，使用此函数之前，摄像机必须是连接成功的。
631         函数：public PLAY_STATUS audio_status();
632         返回：返回PLAY_STATUS类型，类型状态如下：
633         public enum PLAY_STATUS {
634                 STOPPED, //停止
635                 REQUESTING, //请求中
636                 PLAYING; //播放中
637             }
638
639     2）音频质量
640
            说明：使用以下两个函数可以获取到音频质量的相关数据，使用以下两个函数之前，摄像机必须是连接成功的。
641         函数：
642         public int audio_sps(); //音频的sps
643         public int audio_byterate(); //音频的byte
```

3）开启音频

说明：通过此函数，可以实现开启音频功能，使用此函数之前，摄像机必须是连接成功的。

函数：public ERROR play_audio();

返回：返回ERROR类型，当返回类型是ERROR.NO_ERROR时，表示开启音频成功。

4）关闭音频

说明：通过此函数，可以实现关闭音频功能，使用此函数之前，摄像机必须是连接成功的。

函数：public void stop_audio();

5）监听

说明：使用方法请参考［2.2摄像机基础监听］

回调函数：public void on_audio_status_changed(IPCam ipcam);

示例：

```
case R.id.play_voice:
    {
        if (m_ipcam.audio_status() == PLAY_STATUS.PLAYING){//判断获取音频状态
            m_ipcam.stop_audio();    //关闭音频
        }else{
            m_ipcam.play_audio();    //开启音频
        }
    }
```

10.5.摄像机对讲

说明：摄像机对讲，主要包括对讲状态获取，对讲开启和关闭操作。操作以下函数前，摄像机必须是连接成功的。

1）对讲状态

说明：通过此函数，可以获取到摄像机当前的对讲状态，使用此函数之前，摄像机必须是连接成功的。

函数：public PLAY_STATUS speak_status();

返回：返回PLAY_STATUS类型，类型状态如下：

```
public enum PLAY_STATUS {
        STOPPED, //停止
        REQUESTING, //请求中
        PLAYING; //播放中
    }
```

2）对讲质量

说明：通过以下函数可以获取到对讲数量相关数据，使用以下函数前，摄像机必须是连接的。

函数：

```
public int speak_sps(); //对讲的fps
public int speak_byterate() //对讲的byte
```

3）开启对讲

说明：通过此函数，可以实现开启对讲功能，使用此函数之前，摄像机必须是连接成功的。

函数：public ERROR start_speak();

返回：返回ERROR类型，当返回类型是ERROR.NO_ERROR时，表示开启对讲成功。

4）关闭对讲

说明：通过此函数，可以实现关闭对讲功能，使用此函数之前，摄像机必须是连接成功的。

函数：public void stop_speak();

5）监听

说明：使用方法请参考［2.2摄像机基础监听］

回调函数：public void on_speak_status_changed(IPCam ipcam);

示例：

```
case R.id.play_talk:
    if (m_ipcam.speak_status() == PLAY_STATUS.PLAYING) //判断对讲状态
        m_ipcam.stop_speak();
    else
        m_ipcam.start_speak();
```

10.6，摄像机本地录像

说明：摄像机本地录像，主要包括本地录像状态获取，开启本地录像和停止本地录像。操作以下函数前，摄像机必须是连接成功的。

1）本地录像状态

说明：通过此函数，可以获取到摄像机本地录像状态，使用此函数之前，摄像机必须是连接成功的。

函数：public PLAY_STATUS local_record_status();

返回：返回PLAY_STATUS类型，类型状态如下：

```
public enum PLAY_STATUS {
        STOPPED, //停止
        REQUESTING, //请求中
        PLAYING; //播放中
    }
```

2）开启本地录像

说明：使用下面任何一个函数可以实现录像功能，函数1不带存储地址的函数，使用前，必须创建一个登录账号[否则会因为存储地址错误，导致存储失败]，获取录像文件直接通过Storage的一个函数直接获取。函数2带存储地址，自己决定存储地址，自己从地址中获取录像文件。另外不管是用那一个函数实现录像，使用前必须保证摄像机视频是PLAYING状态的。

函数1:public String start_local_record();

返回：返回String类型，当返回路径时，表示开启录像成功。

函数2:public String start_local_record(String record_file_path, RECORD_FORMAT format)

参数：record_file_path：录像本地存储地址

format:录像格式，RECORD_FORMAT类型对象

```
  public enum RECORD_FORMAT{
      RECORD_3GP,
      RECORD_MOV,
  }
```

返回：返回String类型，当返回路径时，表示录像成功。

函数3：public String start_local_record(RECORD_FORMAT format)

参数：format:录像格式，RECORD_FORMAT类型对象；

返回：返回String类型，当返回路径时，表示录像成功。

3）停止本地录像

说明：通过此函数实现停止录像功能

函数：public void stop_local_record();

4）本地录像监听

说明：使用方法请参考［2.2摄像机基础监听］

回调函数：public void on_camera_recording_changed(IPCam ipcam);

示例：

```
case R.id.play_record:
    if (m_recording){
        m_ipcam.stop_local_record();//停止录像
        m_recording = false;
    }else{
        m_local_record_filepath = m_ipcam.start_local_record(); //开启录像
        Log.e("sosocam record","--------m_local_record_filepath----------" +
        m_local_record_filepath);
        if (m_local_record_filepath == null) {
            Log.e("SoDemo","-camera--local_record--failed-");
        } else {
            m_recording = true;
            }
        }
```

10.7.摄像机拍照

说明：使用下面任何一个函数可以实现拍照功能，函数1不带存储地址的函数，使用前，必须创建一个登录账号[否则会因为存储地址错误，导致存储失败]，获取照片直接通过Storage的一个函数直接获取。函数2带存储地址，自己决定存储地址，自己从地址中获取照片。另外不管是用那一个函数实现拍照，使用前必须保证摄像机视频是打开的。

函数1：public String snapshot();

返回：返回String类型，当返回拍照路径时，表示拍照成功。

函数2:public String snapshot(String snapshot_file_path);

参数：snapshot_file_path：照片本地存储地址

返回：返回String类型，当返回拍照路径时，表示拍照成功。

示例：

```
case R.id.play_photograph:
    String filepath = m_ipcam.snapshot(); //执行拍照
    if (filepath == null) {
        Log.e("SoDemo","failed_snapshot");
```

```
        } else {
            Log.e("SoDemo","snapshot succeed");
        }
```

10.8.ptz命令控制巡航

说明：摄像机ptz巡航操作，主要水平巡航开关，垂直巡航开关，轨迹巡航开关，启动预置位的设置和清除，轨迹巡航点设置，以及上下左右操作

1）水平/垂直/轨迹巡航设置

函数：public ERROR ptz_control(PTZ_CMD cmd, int param)

参数：主要有两个参数，详细如下：

cmd：PTZ_CMD命令，巡航操作命令主要如下：

```
public enum PTZ_CMD {
        UP, //向上
        DOWN, //向下
        LEFT, //向左
        RIGHT, //向右
        T_PATROL, //水平巡航
        P_PATROL, //垂直巡航
        TRACK_PATROL,//轨迹巡航
        MEIJING_VOLUME_UP,//客户定制功能
        MEIJING_VOLUME_DOWN,//客户定制功能
        MEIJING_NEXT_TRACK,//客户定制功能
        MEIJING_PREVIOUS_TRACK,//客户定制功能
        MEIJING_PLAY_PAUSE,//客户定制功能
        MEIJING_LED_POWER,//客户定制功能
        MEIJING_LED_STYLE;//客户定制功能
    }
```

param：参数值，0:关 20:开 [上下左右控制没有停止命令，param固定位20，发一下，动一次]

示例：

```
/*水平巡航*/
case R.id.play_cache_v:
    if (conn_status == CONN_STATUS.CONNECTED)
        m_ipcam.ptz_control(PTZ_CMD.T_PATROL, 20);//开启水平巡航
    break;
case R.id.play_cache_v_stop:
    if (conn_status == CONN_STATUS.CONNECTED)
        m_ipcam.ptz_control(PTZ_CMD.T_PATROL, 0);//关闭水平巡航
    break;

/*上下左右控制*/
public void on_touch_event(IPCamVideoView_Listener.TOUCH_EVENT event) {
        switch (event) {
        case MOVE_UP:
            m_ipcam.ptz_control(PTZ_CMD.UP, 20);//上
            break;
        case MOVE_DOWN:
            m_ipcam.ptz_control(PTZ_CMD.DOWN, 20);//下
            break;
        case MOVE_LEFT:
            m_ipcam.ptz_control(PTZ_CMD.LEFT, 20);//左
            break;
        case MOVE_RIGHT:
            m_ipcam.ptz_control(PTZ_CMD.RIGHT, 20);//右
            break;
        default:
            break;
        }
    }
```

2）启动预置位设置

1）设置预置位

函数：public ERROR set_boot_preset(set_boot_preset_listener listener);//设置预置位以及添加监听

回调函数：public void on_set_boot_preset_result(IPCam ipcam, ERROR error);

2）清除预置位置

函数：public ERROR clear_boot_preset(clear_boot_preset_listener listener);//清除预置位以及添加监听

回调函数：public void on_clear_boot_preset_result(IPCam ipcam, ERROR error);

示例：

```
case R.id.preset_complete:
    if (ERROR.NO_ERROR == m_ipcam.set_boot_preset(LiveActivity.this)) {
        Log.e("SoDemo","set_boot_preset_OK");
```

```
            } else {
                Log.e("SoDemo","set_boot_preset_failed");
            }
            break;
        case R.id.preset_cancel:
            if (ERROR.NO_ERROR == m_ipcam.clear_boot_preset(LiveActivity.this)) {
                Log.e("SoDemo","clear_boot_preset_OK");
            } else {
                Log.e("SoDemo","clear_boot_preset_failed");
            }
            break;

        public void on_set_boot_preset_result(IPCam ipcam, ERROR error) {
            if (ERROR.NO_ERROR == error) {
                Log.e("SoDemo","set_boot_preset_succeed");
            } else {
                Log.e("SoDemo","set_boot_preset_failed");
            }
        }
        public void on_clear_boot_preset_result(IPCam ipcam, ERROR error) {
            if (ERROR.NO_ERROR == error) {
                Log.e("SoDemo","clear_boot_preset_succeed");
            } else {
                Log.e("SoDemo","clear_boot_preset_failed");
            }
        }
```

3）轨迹设置

1）获取当前轨迹位置

函数：public ERROR get_pt_pos(get_pt_pos_listener listener);//获取当前轨迹位置

回调函数：public void on_get_pt_pos_result(IPCam ipcam, ERROR error, int pos);

回调返回参数：回调返回三个参数，如下：

ipcam：一个IPCam对象

error：错误值

pos：轨迹位置 [最多设置十六个轨迹位]

2）设置轨迹位置

函数：public ERROR set_track(ArrayList<TRACK_NODE> nodes_list, set_track_listener listener)

参数：nodes_list是一个列表，每一个元素都是一个TRACK_NODE对象，具体如下：

```
        public int pos = 0; //轨迹位置
        public int ms = 0;  //当前轨迹位置停留时间
```

回调函数：public void on_set_track_result(IPCam ipcam, ERROR error);

回调返回参数：回调返回两个参数，如下：

ipcam：一个IPCam对象

error：错误值

示例：

```
        case R.id.track_next_node:
            if (track_current_node_index == 16) {
                Log.e("SoDemo","最多只能设置16个位置");
                break;
            }
            m_ipcam.get_pt_pos(this));//获取当前轨迹位置

        /*获取当前轨迹的回调函数*/
        public void on_get_pt_pos_result(IPCam ipcam, ERROR error, int pos) {
            if (ERROR.NO_ERROR == error) {
                TRACK_NODE node = new TRACK_NODE();
                node.pos = pos;
                node.ms = 5 * 1000;
                m_nodes_list.add(node);
            }
        }

        m_ipcam.set_track(m_nodes_list, this));//设置轨迹
        /*设置轨迹的回调*/
        public void on_set_track_result(IPCam ipcam, ERROR error) {
            if (ERROR.NO_ERROR == error) {
                Log.e("SoDemo","设置轨迹成功");
            } else {
                Log.e("SoDemo","设置轨迹失败");
            }
        }
```

10.9sensor命令设置参数

说明：senor命令可以设置亮度（brightness）/对比度（contrast）/锐度（sharpness）/饱和度（saturation），频率(power_frequency),图像翻转（flip），场景设置（white_balance）这些参数，使用这个命令的条件也是要摄像机连接成功。

1）用sensor命令获取参数

说明：使用get_sensor_params时实现监听IPCam.get_sensor_params_listener，一旦有结果，就会回调public void on_result(IPCam ipcam, ERROR error, SENSOR_PARAMS params);。

函数：public ERROR get_sensor_params(get_sensor_params_listener listener);

回调函数：public void on_result(IPCam ipcam, ERROR error, SENSOR_PARAMS params);

回调参数：回调参数一共有3个，详情如下：

ipcam：一个IPCam对象;

error:ERROR类型值，当返回是ERROR.NO_ERROR，表示获取参数成功。

SENSOR_PARAMS介绍如下：

```
public class SENSOR_PARAMS {
    public int brightness; //亮度
    public int contrast; //对比度
    public int sharpness; //锐度
    public int saturation; //饱和度
    public int power_frequency;
    //频率设置，共有三个值。0：60Hz；1:50Hz；2：室外
    public int white_balance;
    //场景设置，共有6个值。0:自动;1：白织灯;2:冷光源;3:阳光;4:多云;5:阴天
    public int flip;
    //图像翻转设置，共有4个值。0:正常;1:镜像;2:上下翻转;3:镜像＋上下翻转;
}
```

2）用sensor命令设置参数

函数：public ERROR sensor_control(SENSOR_CMD cmd, int param)

参数：一共有两个参数，详细如下：

cmd：命令参数，如下：

```
public enum SENSOR_CMD {
        BRIGHTNESS, //亮度设置
        CONTRAST, //对比度设置
        SHARPNESS, //锐度设置
        SATURATION,  //饱和度设置
        POWER_FREQUENCY, //频率设置
        WHITE_BALANCE, //场景设置
        FLIP;   //图像翻转设置
    }
```

param：命令参数值

3）示例：

```
/*
 *用sensor命令设置参数
 */
m_ipcam.sensor_control(SENSOR_CMD.POWER_FREQUENCY,1);;//设置频率为50Hz
/*
 *用sensor命令获取参数＋添加IPCam.get_sensor_params_listener监听
 */
m_ipcam.get_sensor_params(listener);
/*
 *IPCam.get_sensor_params_listener回调
 */
 class GET_SENSOR_PARAMS_LISTENER implements IPCam.get_sensor_params_listener {
    @Override
    public void on_result(IPCam ipcam, ERROR error, SENSOR_PARAMS params) {
        if(ERROR.NO_ERROR = error){
        Log.e("SoDemo","-当前flip为--" + param.flip);
        Log.e("SoDemo","-当前频率为--" + param.power_frequency);
        Log.e("SoDemo","-当前亮度为--" + param.brightness);
        Log.e("SoDemo","-当前对比度为--" + param.contrast);
        Log.e("SoDemo","-当前锐度为--" + param.sharpness);
        Log.e("SoDemo","-当前饱和度为--" + param.saturation);
        Log.e("SoDemo","-当前场景设置为--" + param.white_balance);
        }
    }
}
```

10.10.CGI命令设置参数

说明:CGI命令操作是通过cgi命令设置的一些参数，比如重启，录像，tf的设置，等其它功能。用cgi命令操作前必须确认摄像机是连接成功的。

函数：public ERROR set_params(String params, set_params_listener listener)

参数：params为设置摄像机cgi参数。由瑞彩提供，下面提供几种常见的cgi,如下：

1）报警布防："save=1&reinit_alarm=1&arm_schedule=0"

//其中arm_schedule有三个值，0:布防，1:计划布防，2:撤防

2）录像："save=1&reinit_record=1&record_schedule_list=[]"

//其中record_schedule_list=[]:停止录像；"save=1&reinit_record=1&record_schedule_list=[{\"start\":0,\"end\":96,\"day\":127}]:开启全时录像

3）重启："reboot=1"

回调函数：

```
public void on_result(IPCam ipcam, ERROR error);
```

回调参数：回调返回两个参数，详细如下：

ipcam：返回一个IPCam类型的对象

error:ERROR类型，当返回是ERROR.NO_ERROR，表示重置成功

示例：

```
/*
 *cgi设置参数+添加监听IPCam.set_params_listener
 */
String params = "save=1&reinit_alarm=1&arm_schedule=1"; //计划布防cgi
m_ipcam.set_params(params, this);
m_state = STATE.SET_CAMERA;
/*
 *IPCam.set_params_listener回调函数
 */
public void on_result(IPCam ipcam, ERROR error) {
    ipcam.stop_connect();
    if (m_state == STATE.SET_CAMERA) {
        if (error == ERROR.NO_ERROR) {
            Log.e("SoDemo","设置成功");
        } else {
            Log.e("SoDemo","设置失败");
        }
    }
}
```

10.11获取参数

说明：参数类的值都可以通过此函数获取。

函数：public ERROR get_params(String params, get_params_listener listener);

参数：需要去获取的那个参数。比如:图像翻转 "flip="

返回：返回ERROR类型，当返回ERROR.NO_ERROR，则表示函数执行成功。

回调函数：当get_params时，添加了监听IPCam.get_params_listener，当有结果时，就会回调此函数。

```
public void on_result(IPCam ipcam, ERROR error, JSONObject json);
```

回调参数说明：返回三个参数值，详细如下：

ipcam：返回一个IPCam对象

error:ERROR类型，当返回是ERROR.NO_ERROR，表示获取参数成功

params：一个JSONObject；

示例：

```
/*
 *获取参数＋添加监听IPCam.get_params_listener
 */
m_ipcam.get_params("arm_schedule=",this);
/*
 *IPCam.get_params_listener回调函数
 */
    @Override
    public void on_result(IPCam ipcam, ERROR error, JSONObject json) {
        if (error == ERROR.NO_ERROR) {
            try {
                arm_schedule = json.getInt("arm_schedule");
                if(arm_schedule == 2){
                    m_button_disarm.setTextColor(Color.RED);
                }else if(arm_schedule == 0){
                    m_button_arm.setTextColor(Color.RED);
                }else if(arm_schedule == 1){
                    m_button_schedule_arm.setTextColor(Color.RED);
```

```
                    }
                } catch (JSONException e) {
                    e.printStackTrace();
                }
            } else {
                Log.e("SoDemo","---获取参数失败----");
            }
        }
```

10.11串口通信

说明：串口通信，主要包括串口开关功能和串口的读写功能实现。用串口命令操作前必须确认摄像机是连接成功的。

函数：public ERROR write_comm(byte[] data);

10.12.RF开关设备操作

说明：RF开关设备操作，主要包括如何获取开关设备，以及对开关设备进行开关操作

1）获取开关设备

说明：通过以下函数可以获取到rf开关设备，以及设备的相关数据

函数：public ERROR get_rf_switch_devices(get_rf_switch_devices_listener listener);//获取外接设备列表函数

返回：返回ERROR类型，返回值为ERROR.NO_ERROR则调用成功

回调：

监听：public void on_get_rf_switch_devices_result(IPCam ipcam, ERROR error, ArrayList<rf_switch_device_info> rf_switch_devices_list);;//IPCam.get_rf_switch_devices_listener回调

返回：回调会返回三个参数

ipcam：Ipcam对象

error：ERROR类型值

rf_switch_devices_list：列表，列表的每一个元素是一个rf_switch_device_info对象，详细如下：

```
public class rf_switch_device_info {
    public int addr;  //开关地址
    public int switch_status;  //设备开关状态 1:开  0:关
    public int link;
    public String name;  //设备名称
}
```

示例：

```
m_ipcam.get_rf_switch_devices(this);;//获取开关设备，且添加监听get_rf_switch_devices_listener
/*
 *获取开关设备有结果后，回调函数
 */
 public void on_get_rf_switch_devices_result(IPCam ipcam, ERROR error, ArrayList<rf_switch_device_info> rf_switch_devices_list) {
        if (error == ERROR.NO_ERROR ) {
            m_rf_switch_devices_list = rf_switch_devices_list;
            if(m_rf_switch_devices_list.size() > 0){
                Log.e("SoDemo","当前外设数量为" + m_rf_switch_devices_list.size());
                rf_switch_device_info device = m_rf_switch_devices_list.get(0);
                Log.e("SoDemo","当前外设名字为" + device.name);
                Log.e("SoDemo","当前外设状态为" + device.switch_status);
            }else{
                Log.e("SoDemo","没有找到外设设备");
            }
        }
    }
```

2）对开关设备进行开关操作

说明：通过一下函数，可以对获取到的开关设备进行开关操作

函数：public ERROR switch_rf_device(int addr, int status, switch_rf_device_listener listener);//设置开关设备和添加IPCam.switch_rf_device_listener监听

参数：两个参数，详细如下：

addr：设置的这个开关设备的地址

status：设置的值，1：开 0:关

回调函数

函数：public void on_switch_rf_device_result(IPCam ipcam, ERROR error);
返回：回调会返回两个参数，如下：
ipcam：Ipcam对象
error：错误值

示例：

```
int status = (switch_device.switch_status!=0)?0:1;
if (ERROR.NO_ERROR == m_ipcam.switch_rf_device(switch_device.addr, status,
this)) { //设置开关
    Log.e("SoDemo","设置开关函数调用成功");
}

/* 设置开关有结果后的回调*/
public void on_switch_rf_device_result(IPCam ipcam, ERROR error) {
        if (ERROR.NO_ERROR == error)
            Log.e("SoDemo","设置开关成功");
        else
            Log.e("SoDemo","设置开关失败");
    }
```

10.13.TF卡相关操作

说明：TF卡相关操作，主要包括弹出tf卡，格式化tf卡操作，等相关功能的实现。

1）弹出tf卡

说明：弹出tf卡后，tf讲不在能使用，需要重新插拔才能读取到tf卡。
函数：public ERROR unplug_tf(unplug_tf_listener listener);
返回：返回ERROR类型，当返回ERROR.NO_ERROR，表示当前没有读卡，调用成功。

回调函数：public void on_result(IPCam ipcam, ERROR error);
回调参数：
ipcam：一个IPCam对象
error：ERROR类型，当返回是ERROR.NO_ERROR，表示弹出成功

示例：

```
/*
 * 弹出tf卡＋添加回调IPCam.unplug_tf_listener
 */
m_ipcam.unplug_tf(this);
m_state = STATE.EJECT;
/*
 * IPCam.unplug_tf_listener回调
 */
public void on_result(IPCam ipcam, ERROR error) {
    if(m_state == STATE.EJECT){
        if (error == ERROR.NO_ERROR) {
            Log.e("SoDemo","ejecting_tf_succeed");
        } else {
            Log.e("SoDemo","ejecting_tf_failed");
        }
    }
}
```

2）格式化tf卡

说明：格式化tf卡
函数：public ERROR format_tf(format_tf_listener listener);
返回：返回ERROR类型，当返回ERROR.NO_ERROR，表示没有测试，调用成功。

回调函数：public void on_result(IPCam ipcam, ERROR error);
回调参数：
ipcam：一个IPCam对象
error：ERROR类型，当返回ERROR.NO_ERROR，表示弹出成功
示例：

```
/*
 * 格式化tf卡＋添加回调IPCam.format_tf_listener
 */
m_ipcam.format_tf(this);
m_state = STATE.FORMAT;
/*
 * IPCam.format_tf_listener回调
 */
public void on_result(IPCam ipcam, ERROR error) {
    if(m_state == STATE.FORMAT){
        if (error == ERROR.NO_ERROR) {
```

```
                    Log.e("SoDemo","format_tf_succeed");
                } else {
                    Log.e("SoDemo","format_tf_failed");
                }
            }
        }
```

3）tf卡状态

说明：此函数用来获取当前摄像机的tf状态信息

函数：public TF_STATUS tf_status()

返回：TF_STATUS类型值，具体如下：

```
public enum TF_STATUS {
    NONE,  //未检测到TF卡
    READY, //tf卡状态正常
    ERROR, //tf卡出错
    FULL,  //tf卡存储空间已满
    CHECK;  //检测tf卡，摄像机正在读取tf中文件信息，此时不能对tf卡进行操作
}
```

4）tf卡当前空余容量

函数：public int tf_free()

返回值：int类型值，单位为MB

5）tf卡总容量

函数：public int tf_capacity()

返回值：int类型值，单位为MB;

示例：

```
TF_STATUS tf_status = ipcam.tf_status();
if(tf_status == READY){
    Log.e("SoDemo","current TF card capacity is " + ipcam.get_disk_size())
    Log.e("SoDemo","current TF card spare capacity is " + ipcam.tf_free())
}
```

#11.无线网络设置

说明：无线网络设置包括，扫描周围wifi和设置摄像机wifi功能。

1）扫描wifi

说明：扫描Wi-Fi，是通过摄像机去扫描周围的wifi,不是通过手机扫描。所以使用wifi_scan前必须先要链接摄像机成功。使用wifi_scan时，如添加了wifi_scan_listener，一旦扫描有结果了就会调用public void on_result(IPCam ipcam, ERROR error, ArrayList<ap_info> ap_list)。

函数：public ERROR wifi_scan(wifi_scan_listener listener);

返回：返回ERROR类型，当返回是ERROR.NO_ERROR，表示wifi_scan执行成功。

回调函数：public void on_result(IPCam ipcam, ERROR error, ArrayList<ap_info> ap_list);

回调参数：回调函数返回三个参数，详细如下：

ipcam：返回一个IPCam对象

error:ERROR类型值，当返回是ERROR.NO_ERROR，表示scan成功

ap_list:一个列表，列表的每一个元素是ap_info类型，具体如下：

```
public String bssid;  // bssid wifi的唯一标示码，相当MAC地址
public String ssid;  // ssid  wifi名称
public WIFI_AUTH auth; //认证方式
public WIFI_ENCRYPT encrypt;  //加密方式
public int rssi;  //信号强度
/*
 *认证方式有五种，如IPCAM_WIFI_AUTH所示
 */
 public enum WIFI_AUTH {
        OPEN,       //无密码
        WEP,        //WEP
        WPAPSK,     //WPAPSK
        WPA2PSK,    //WPA2PSK
        UNKNOWN;    //unknown
    }
/*
 *加密方式有五种，如IPCAM_WIFI_ENCRYPT所示
 */
public enum WIFI_ENCRYPT {
        NONE,
        WEP,
        TKIP,
```

```
        AES,
        UNKNOWN;
    }
```

2）摄像机无线设置

说明：函数1和函数2都可以实现设置无线功能。函数1增加了无线测试功能，但是只有当当前摄像机的是有线连接时（即[m_ipcam wifi_power] == 0,wifi信号是0）时，才可以使用。函数2没有无线测试功能，直接设置。不管摄像机当前是有线连接还是无线连接都可以使用。推荐方式，摄像机当前是有线连接时，使用函数1设置。摄像机当前是无线连接时，使用函数2设置。

函数1：

```
public ERROR set_wifi(String ssid, WIFI_AUTH auth, WIFI_ENCRYPT encrypt, int wep_key_index, WIFI_WEP_KEY_TYPE key_type, String key, set_wifi_listener listener);
```

函数2:

```
public ERROR set_wifi_without_testing(String ssid, WIFI_AUTH auth, WIFI_ENCRYPT encrypt, int wep_key_index, WIFI_WEP_KEY_TYPE key_type, String key, set_wifi_listener listener);
```

参数：当认证方式auth是wep时，才要使用wep_key_index，wep_key_type参数，其他加密方式，这两个参数实际上用不着。

ssid:wifi名称，通过扫描返回
key：wifi密码
auth：认证方式，通过扫描返回
encrypt：加密方式，通过扫描返回

```
/*
 *wep索引，共有四种，［0，3］。瑞彩处理：强制设置为0索引，其他三个索引就不考虑了
 */
```

wep_key_index：wep索引

```
/*
 *wep类型，共有两种，HEX和ASCII。瑞彩处理：当auth无密码或auth是wep+key的长度是5或13时，把wep_key_type设置为HEX，其他都是ASCII
 */
 public enum WIFI_WEP_KEY_TYPE {
        HEX, //hex类型
        ASCII; //ASCII类型
    }
```

回调函数1：

```
public void on_progress(IPCam ipcam, SET_WIFI_STATE state);
```

回调返回：返回两个参数，详细如下：
ipcam：一个IPCam对象
state：设置Wi-Fi的进程，一个SET_WIFI_STATE类型，详细如下：

```
public enum SET_WIFI_STATE {
    SETTING, //wifi设置中
    TESTING, //wifi测试中
    SAVING; //wifi保存中
        };
```

回调函数2:

```
public void on_result(IPCam ipcam, ERROR error);
```

回调返回：
ipcam：返回一个IPCam对象

error：ERROR类型，当返回是ERROR.NO_ERROR，表示设置wifi成功，返回ERROR.DEVICE_OPERATION_FAIL表示密码错误，其他都是设置失败。

3)示例

```
/*扫描Wi-Fi*/
public void scan_wifi() {
    /*扫描Wi-Fi+添加监听IPCam.wifi_scan_listener*/
    if (ERROR.NO_ERROR != m_ipcam.wifi_scan(this)) {
        Log.e("SoDemo","SCAN_WIFI_FAIL");
    }
}
/*IPCam.wifi_scan_listener回调 */
public void on_result(IPCam ipcam, ERROR error, ArrayList<ap_info> ap_list){
```

```
1293            if (ERROR.NO_ERROR == error) {
1294                m_aps_list = ap_list;
1295                if (m_ap_list.size() == 0) {
1296                        m_listview_wifi.setAdapter(null);
1297                        m_state = STATE.SCANNING_NOTHING;
1298                        Log.e("SoDemo","SCAN_WIFI_NOTHING");
1299                    } else {
1300                        m_listview_wifi.setAdapter(m_wifiListAdapter);
1301                        m_state = STATE.SCANNING_OK;
1302                        Log.e("SoDemo","SCAN_WIFI_OK");
1303                    }
1304
1305                } else {
1306                    m_listview_wifi.setAdapter(null);
1307                    Log.e("SoDemo","SCAN_WIFI_FAILED");
1308                }
1309            }
1310
1311            /*设置wifi*/
1312            public void set_wifi() {
1313                m_ap.auth = ap.auth;
1314                m_ap.encrypt = ap.encrypt;
1315                m_ap.ssid = ap.ssid;
1316                if (m_ap.auth == WIFI_AUTH.OPEN) {
1317                    set_camera();
1318                }else{
1319                    key = m_edittext_key.getText().toString();
1320                }
1321            }
1322
1323            private void set_camera() {
1324                WIFI_WEP_KEY_TYPE type = WIFI_WEP_KEY_TYPE.ASCII;
1325                if (m_ap.auth == WIFI_AUTH.WEP) {
1326                    if ((m_ap.key.length() == 5) || (m_ap.key.length() == 13))
1327                        type = WIFI_WEP_KEY_TYPE.ASCII;
                            //有加密方式，且key的长度是5或13时type为ASCII
1328                    else
1329                        type = WIFI_WEP_KEY_TYPE.HEX;//其他加密，type为hex
1330                }
1331                if (m_wifi_power == 0){
                    /*当摄像机当前是有线连接，使用set_wifi,且强制设置索引为0*/
1332                    if (ERROR.NO_ERROR == m_ipcam.set_wifi(m_ap.ssid, m_ap.auth,
                        m_ap.encrypt, m_ap.wep_key_index, type, m_ap.key, this)) {
1333                        m_state = STATE.SET_CAMERA;
1334                        Log.e("SoDemo","succeed_set_wifi");
1335                    } else {
1336                        Log.e("SoDemo","failed_set_wifi");
1337                        }
1338
                    }else{/*当摄像机当前是无线连接，使用set_wifi_without_testing,且强制设置索
                    引为0*/
1339                    if (ERROR.NO_ERROR == m_ipcam.set_wifi_without_testing(m_ap.ssid,
                        m_ap.auth, m_ap.encrypt, m_ap.wep_key_index, type, m_ap.key, this)) {
1340                        m_state = STATE.SET_CAMERA;
1341                        Log.e("SoDemo","succeed_set_wifi");
1342                    } else {
1343                        Log.e("SoDemo","failed_set_wifi");
1344                    }
1345                }
1346            }
1347
1348            /*IPCam.wifi_scan_listener回调*/
1349
1350            public void on_progress(IPCam ipcam, SET_WIFI_STATE state) {
1351                switch (state) {
1352                    case SET_WIFI_STATE.SETTING: //Wifi设置中
1353                        m_state = SETTING_WIFI;
1354                        break;
1355                    case SET_WIFI_STATE.TESTING: //wifi测试中
1356                        m_state = TESTING_WIFI;
1357                        break;
1358                    case SET_WIFI_STATE.SAVING: //wifi保存中
1359                        m_state = SAVING_WIFI;
```

```
                    break;
                default:
                    break;
            }
        }

        public void on_result(IPCam ipcam, ERROR error) {
            if (error == ERROR.NO_ERROR) {
                Log.e("SoDemo","wifi_setting_ok");
            } else if ((error == ERROR.DEVICE_OPERATION_FAIL) && (m_ap.auth !=
            WIFI_AUTH.OPEN)) {
                Log.e("SoDemo","failed_join_wifi,wifi key error");
                m_state = STATE.ENTER_AP_KEY;
            } else {
                Log.e("SoDemo","failed_set_wifi");
            }
        }
```

#12.报警类型和报警设备名字获取

1）报警状态监控

说明：报警状态监控，情参考［2.2.摄像机基础监听］

函数：public void on_camera_alarm_changed(IPCam ipcam);

2）获取报警名字

说明：这是外连设备的名字，只有外连设备才使用此函数获取，摄像机本身检测到的报警类型没有此选项。

函数：public ERROR get_alarm_name(get_alarm_name_listener listener);

回调函数：public void on_get_alarm_name_result(IPCam ipcam, ERROR error, String alarm_name);

回调函数说明：当get_alarm_name时添加了监听IPCam.get_alarm_name_listener，当有结果时，就会掉用回调函数on_get_alarm_name_result。回调函数，共返回以下参数，详细如下：

ipcam：返回一个IPCam对象

error:ERROR类型，当返回是ERROR.NO_ERROR，表示获取名字成功。

alarm_name：报警名字

2）报警类型获取

说明：通过此函数，可以获取当前报警类型

函数：public ALARM alarm();

返回：返回ALARM类型，RF是外连设备的报警，类型如下：

```
        public enum ALARM {
                NONE,     //无报警
                MOTION_DETECT, //移动侦测报警
                TRIGGER_DETECT,//红外报警
                SOUND_DETECT, //声音侦测报警
                TEMPERATURE,//温度报警
                HUMIDITY,//湿度报警
                RF_EMERGENCY, //RF紧急报警
                RF_MAGNETIC,//RF门磁报警
                RF_PIR,//PIR报警
                RF_SMOKE,//RF烟雾报警
                RF_GAS, //RF燃气报警
                RF_LOW_VOLTAGE, //RF低压报警
                RF_SHAKE, //RF震动报警
                RF_GLASS_BROKEN, //RF玻璃破碎报警
                RF_CUSTOM,//RF用户自定义报警
                UNKNOWN; //未知报警
            }
```

4）示例

```
        /*
         *报警状态的监控，添加了IPCam_Listener监听，每当有报警就会回调此函数
         */
         public void on_camera_alarm_changed(IPCam ipcam) {
                ALARM alarm = ipcam.alarm(); //获取报警类型
                if (alarm == ALARM.NONE)
                    ;
                else if (alarm == ALARM.MOTION_DETECT)
                    ;
```

```
1426                else if (alarm == ALARM.SOUND_DETECT)
1427                    ;
1428                else
1429
                        ipcam.get_alarm_name(this);//RF设备，可以获取报警的rf设备名字＋添
                        加IPCam.get_alarm_name_listener
1430            }
1431        /*
1432
             *添加了IPCam.get_alarm_name_listener监听，当get_alarm_name有结果时，就会掉用
             此函数。
1433         */
1434        public void on_get_alarm_name_result(IPCam ipcam, ERROR error, String
            alarm_name)
1435            {
1436                camListAdapter.notifyDataSetChanged();
1437            }
1438
1439        /*
1440         * 显示报警详细信息，报警类型＋报警名字
1441         */
1442        ALARM alarm = cam.alarm();
1443        if (alarm == ALARM.NONE) {
1444            Log.e("SoDemo","--no alarm!-");
1445        } else{
1446            if (alarm == ALARM.MOTION_DETECT)
1447                Log.e("SoDemo","--alarm_dected");//"移动侦测报警！ ");
1448            else if (alarm == ALARM.SOUND_DETECT)
1449                Log.e("SoDemo","-alarm_sound");//"声音侦测报警！ ");
1450            else if (alarm == ALARM.TRIGGER_DETECT)
1451                Log.e("SoDemo","-alarm_trigger" + cam.alarm_name());//紧急报警！ ");
1452            else if (alarm == ALARM.RF_EMERGENCY)
1453                Log.e("SoDemo","-emergency_alarm" +
                    cam.alarm_name());//")紧急报警！ ");
1454            else if (alarm == ALARM.RF_MAGNETIC)
1455                Log.e("SoDemo","-magentic_contacts" +
                    cam.alarm_name());//")门磁报警！ ");
1456            else if (alarm == ALARM.RF_PIR)
1457                Log.e("SoDemo","-infrared_alarm" + cam.alarm_name());//红外报警！ ");
1458            else if (alarm == ALARM.RF_SMOKE)
1459                Log.e("SoDemo","-smog_alarm" + cam.alarm_name()) ;//")烟雾报警！ ");
1460            else if (alarm == ALARM.RF_GAS)
1461                Log.e("SoDemo","-gas_alarm" + cam.alarm_name()) ;//")煤气报警！ ");
1462            else if (alarm == ALARM.RF_LOW_VOLTAGE)
1463                Log.e("SoDemo","-low_voltage_alarm" + cam.alarm_name())
                    ;//")低电压报警！ ");
1464            else if (alarm == ALARM.RF_SHAKE)
1465                Log.e("SoDemo","-shake_alarm" + cam.alarm_name()) ;//")煤气报警！ ");
1466            else if (alarm == ALARM.RF_GLASS_BROKEN)
1467                Log.e("SoDemo","-glass_broken_alarm" + cam.alarm_name())
                    ;//")低电压报警！ ");
1468            else if (alarm == ALARM.UNKNOWN)
1469                Log.e("SoDemo","-unknown_alarm");//"未知报警！ ");
1470        }
1471
1472
1473
1474
1475
1476
1477
1478
```