```
1
2   ————————————————————————————————————
3   #1.SoSoCamLib简介
4
5   1.SoSoCamLib简介
6
7   SoSoCamLib由7个部分组成，整个libSoSoCamSDK的使用思路是：
8   集成SoSoCamLib———》初始化SDK ———》添加摄像机 ———》连接摄像机
    ———》操作摄像机的各种功能。
9   SDK各个部分的功能以及之间的联系,如下：
10
11  IPCamMgr: IPCamMgr是使用SoSoCamLib必须要使用到的一部分，用于批量管理cam。主要包括SDK初
    始化/反初始化，局域网搜索，摄像机批量连接管理功能。在使用其他功能之前，必须先通过IPCa
    mMgr的get_share初始化成功后，才能去使用呢，另外不管你使用哪一种连接方式都要使用IPCamMg
    r的局域网搜索功能，所以IPCamMgr是非常核心的部分。同时IPCamMgr多处使用了Util的方法和IPC
    am的方法。
12
13  IPCam:IPCam也是SoSoCamLib中操作摄像机的核心部分。一般用于摄像机连接后，实现摄像机的各
    种功能以及监控摄像机的各种状态。比如摄像机视频/音频/对讲/录像/拍照/云台操作/tf卡
    功能实现，也可监控摄像机连接状态/报警状态/tf卡状态等等。当然它也用于创建IPCam对象和
    实现cam的连接。
14
15  IPCamVideoView:IPCamVideoView是观看视频必需要用到的一个类。IPCamVideoView其实就是一个U
    IView,需要结合IPCam才能实现观看视频功能，另外IPCamVideoView还实现了一个协议，可以自定
    义实现在视频上单点和上下左右滑动操作功能。
16
17  Storage:Storage用于本地存储的，主要有存储用户/摄像机/收藏摄像机/录像和照片/报警图
    片信息。你可以不适用这个Storage自己实现存储，你也可以直接用我们的存储接口，值得注意的
    是，如果使用我们的接口且你不需要登录功能，你必需创建一个账号，否则存储不了［因为我们的
    存储路径是以账号开头的］。
18
19  WebService:WebService是在需要用到平台的时候用到。一般都是需要登录功能时用到，用来管理
    从服务器获取用户信息/用户中添加的摄像机信息/用户中收藏分享的摄像机/用户添加摄像机的
    报警信息。当然WebService中还有升级系统固件和app版本的功能，这个不需要登录就可以使用。
20
21  Util:Util是用的比较多的一些处理方法，比如字符串的处理。这些功能在SoSoCamLib中多个类中
    有用到，你也可以直接在你的项目里调用。
22
23  WiFiMatching:WiFiMatching是实现SmartLink设置无线功能用的。主要是开始/停止SmartLink设
    置以及SmartLink设置成功的协议代理。
24
25  SoundWaveWifiSetting:SoundWaveWifiSetting是实现声音设置无线功能用的。主要是开始/停止
    声音设置记忆声音设置过程中各个状态的监控。
26
27  #2.集成SoSoCamLib
28
29  1）说明：集成SoSoCamLib是指把SoSoCamLib集成到你的项目中，供你使用
30
31  2）版本支持：
32  Android2.2及以上版本
33
34  3）组成：
35  java的jar文件：sosocamlib.jar
36      工程中存放位置：libs\sosocam.jar
37  库文件：librcipcam3x.so
38          libsinvoice.so
39      工程中存放位置：libs\armeabi-v7a\
40  资源文件：
41          loading_step1.png
42          loading_step2.png
43          loading_step3.png
44          start_play.png
45
46  4）Manifest.xml中权限设置
47  <!-- for storage module -需要使用到storage时添加下列权限->
48      <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
49      <uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
50      <uses-permission android:name="android.permission.WRITE_OWNER_DATA" />
51      <uses-permission android:name="android.permission.READ_OWNER_DATA" />
52
53  <!-- for ipcam and ipcammgr modules -使用到IPCam和IPCamMgr需用到的权限->
54      <uses-permission android:name="android.permission.INTERNET" />
55      <uses-permission android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE" />
```

```
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.WRITE_MEDIA_STORAGE"/>
    <uses-permission android:name="android.permission.GET_TASKS" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
```

5）步骤：

第一步：导入SoDemo工程到eclipse中。

第二步：Manifest.xml中添加必要的权限。

第三步：设置minSdkVersion版本等

第四步：编译。如果能编译通过就算成功了。

#3.SDK初始化

1)说明：SDK初始化是指：使用sosocam.jar之前必须初始化SDK，初始化完成之后，才能使用sosocam.jar。

2)注意：

a)为提高程序性能，一般在程序开启（即OnCreate()）时，就开始初始化SDK;

b)为提高程序性能，一般在程序（即onDestroy()）退出时，反初始化SDK;

3)使用的类：IPCamMgr

4)函数：

```
/*
 *初始化SDK
 */
IPCamMgr.init(this);//创建IPCamMgr 初始化

/*
 *反初始SDK
 */
IPCamMgr.deinit();//释放IPCamMgr，反初始化
```

5)示例：

```
/*使用SDK之前初始化SDK*/
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    IPCamMgr.init(this); //初始化
}

/*程序退出反初始化SDK*/
protected void onDestroy() {
    super.onDestroy();
    IPCamMgr.deinit(); //反初始化
    }
```

#4.局域网搜索

1）说明：局域网搜索是指：搜索和手机在同一个网络里的摄像机，并获取到这些摄像机。只要创建且开始了IPCamMgr，程序就会一直在后台搜索局域网内的摄像机，且不断更新搜索到的摄像机。直到停止IPCamMgr。

2）使用的类：IPCamMgr

3）注意：只有满足以下条件的摄像机才可以搜索到：

a)系统固件版本为："x.x.x.4.x"

b)书写了设备id,且id前缀为"RTEST-","SOSO-","RCAM-","LCAM-","YSTC-"

c)获取到了局域网IP且局域网IP不为"0.0.0.0"或"169.78.32.56"

e)只要创建了IPCamMgr,程序就一直在后台搜素局域网中的cams

g)当程序后台运行再次进入前台时，都要调用public static void restart()手动重新初始化库。和need_restart()配合使用。

```
/*
 *获取局域网摄像机
 *返回的是DISCOVER_CAMERA_INFO类型的对象：
 *String id;    //摄像机id
 *int id_type; //摄像机id_type
 *String alias;     //设备的别名
 *String fw_version;  //系统固件版本
 *String ui_version;  //网页固件版本
 *int model;  //摄像机类型，两个值，
 *String current_ip; //设备当前的IP地址;
 *int port;    //设备当前的端口;
```

```
 *boolean https;  //设备当前是否安全加密传输，true:加密  false:不加密
 *boolean used;   //当前设备是否已被添加到IPCamMgr中，true:已被添加  false:未被添加
*/
public static LinkedHashMap<String, DISCOVERED_CAMERA_INFO>
get_discovered_cameras_list();//获取局域网里的cams
public static void
rediscover_cameras();//重新搜索局域网中的摄像机，结果从get_discovered_cameras_list()方
法中获取
```

5）步骤：

第一步：用IPCamMgr的init函数进行sdk初始化开始IPCamMgr;

第二步：用IPCamMgr的get_discovered_cameras_list获取局域网的摄像机;

6）示例：

```
/*
 *第一步：初始化开始IPCamMgr 开始后台局域网搜索
 */
IPCamMgr.init(this); //初始化

/*
 *第二步：获取局域网摄像机
 */
 LinkedHashMap<String, DISCOVERED_CAMERA_INFO> ipcam_list;
ipcam_list = IPCamMgr.get_discovered_cameras_list();
for(DISCOVERED_CAMERA_INFO ipcam : ipcam_list.values())
{
    if(!ipcam.used)
        Log.e("SoDemo","Current is not used");
}
```

#5.摄像机的添加 / 删除 / 修改更新

说明：摄像机的添加 / 删除 / 修改更新有三个部分，服务器的，本地的和IPCamMgr的，流程如下：

添加：WebService添加 / 删除 / 修改更新———》Storage添加 / 删除 / 修改修改更新———》IPCamMgr添加 / 删除 / 修改更新。

这些过程按照你的需求，择选，详细如下：

如果用到服务器平台管理摄像机，那么需要在WebService添加 / 删除 / 修改更新[可选项];

如果用到本地管理摄像机，那么需要在Storage添加 / 删除 / 修改更新;

如果用到批量管理摄像机或使用IPCamMgr连接摄像机，那么需要在IPCamMgr添加 / 删除 / 修改更新。［推荐使用IPCamMgr管理];

5.1摄像机的添加<服务器 / 本地 / IPCamMgr>

1）说明：摄像机的添加是是指添加摄像机到服务器 / 本地 / IPCamMgr，添加成功后，你才可以对它操作，监控它的状态等。

2）使用的类：WebService[可选]，IPCam，Storage,IPCamMgr

3）函数：

```
/*
 *Storage类
 *Storage的帐号创建和更新
 *详细用法参考<Storage的本地存储用户更新>
 */
public static void update_user(String user_id, String name, String pwd, int
user_type, String alias);
/*
 *Storage类
 *Storage的设置当前帐号
 *详细用法参考<Storage的本地存储用户管理更新>
 */
public static void set_current_user(String user_id); //设置user_id为当前账号
/*
 *WebService添加摄像机到服务器
 *参数:
 *alias:摄像机昵称
 *camera_id: 摄像机id
 *https:安全加密传输，两个值：YES:加密 NO:非加密
 *model: 这个值没有特定的意义，你可以自己定义，但是如果你不去设置的话
 它默认一直是0。我们是定义0云台机  1为卡片机
 *listener:添加摄像机的监听，可以监控到添加摄像机到服务器的结果
 *返回:
 *当ERROR == ERROR.NO_ERROR时表示添加执行成功
 */
public static ERROR add_camera(String alias, String id, boolean https, int model,
```

```
    add_camera_listener listener);//添加摄像机到服务器
190 /*
191  *Storage添加摄像机到本地
192  CAMERA_INFO对象参数:
193  *String id         //摄像机id
194  *String alias     //摄像机昵称
195  *String user     //摄像机登录用户名
196  *String pwd      //摄像机登录密码
197  *boolean https   //加密安全传输，两个值 0:非加密     1:加密
198  *String obj_id     //摄像机sosocam_id
199  *String model    //摄像机模型，两个值，0 云台机  1 卡片机
200  */
201 public static void add_camera(CAMERA_INFO camera_info);
202
203 /*
204  *IPCamMgr添加摄像机到IPCamMgr
205  *参数:
206  *String alias: 摄像机昵称
207  *String id:摄像机id
208  *String user:摄像机登录用户名
209  *String pwd:摄像机登录密码
210  *https:安全加密传输，两个值：YES:加密 NO:非加密
211  *另外IPCamMgr添加有监听，详见IPCamMgr的添加和删除回调，添加后可以监控添加结果
212  */
213 public static IPCam add_camera(String alias, String id, String user, String pwd,
    boolean https);//添加摄像机到IPCamMgr*
214
215 /*
216  *IPCam方法：更新摄像机局域网信息
217  *参数如下:
218
     *in_lan: BOOL值，是否使用局域网连接。true:使用局域网直连；false:使用p2p连接；缺省为fa
     lse
219  *ip: 局域网（in_lan设置为true):当前摄像机ip；p2p（in_lan设置为false）:可以不用设置
220
     *port: 局域网（in_lan设置为true）:当前摄像机port；p2p（in_lan设置为false）:可以不用设
     置
221  *https: BOOL值，当前摄像机是否加密传输.true:加密传输。false:不加密传输。
222  */
223 public void update_lan_status(boolean in_lan, String ip, int port, boolean https)
224
225 /*
226  *IPCam方法：连接摄像机
227  *摄像机的id、pwd、user、https参数都设置好了以后,就可以开始连接了
228  *参数如下:
229  *retryable:是否重连,true:当因为网络错误时会自动重连接；  false:不重连
230  *retry_delay:int类型，单位ms,延迟多长时间重连。
231  */
232 public ERROR start_connect(boolean retryable, int retry_delay)
233
234 4）步骤:
235 第一步：创建更新一个帐号，且设置当前帐号。只要你用到Storage的方法来存储，不管你是否需
    要帐号登陆，都必须进行这一步。
236
237 第二步：使用IPCam中的方法更新局域网信息，连接摄像机，添加摄像机监听。<如果你不需要执行
    下面第四第五步骤，此步骤可以省略，详细参考摄像机连接：通过IPCam摄像机的连接>
238
239 第三步：用WebService的方法添加到服务器上，如果不需用用到服务器平台这一步省略。<这一步
    骤可以添加add_camera_listener接口监控添加结果，详情参考：WebService的服务器上添加摄像
    机>
240
241 第四步：用WebService的方法添加到服务器上,如果监控到摄像机已经被注册，先解绑再继续添加
    。<如果你需要推送功能，且是通过以账号为id来settag实现的，就必须进行这一步。如果不需要
    这一步可不实现>
242
243 第五步：用WebService的方法添加到服务器上,监控添加成功后，需要relogin一下摄像机，使摄像
    机登陆服务器平台。<如果你需要推送功能，且是通过以账号为id来settag实现的，就必须进行这
    一步。如果不需要这一步可不实现>
244
245 第六步：用Storage的方法把摄像机添加到本地。
246
247 第七步：用IPCamMgr的方法添加摄像机到IPCamMgr。<这一步骤可以添加监听接口，监控添加，详
    情参考：IPCamMgr的cam添加和删除>
```

5）注意：

1)用Storage的方法add_camera把摄像机添加到本地来存储，使用之前，不管你是否需要帐号登陆，都必须创建一个帐号且设置一个当前帐号。<因为本地存储路径是以帐号为路径的，如果没有创建路径，存储地址就是无效的>

2)update_user是为了创建路径，创建有效的存储地址。set_current_user是为了指定后面取本地文件的路径，这两个要是同一个账号，且必须实现的两步。

3)如果只要添加到本地，直接使用第一，第六，第七步骤即可。

4)摄像机unregister_from_sosocam和relogin_to_sosocam的操作必须是在摄像机连接成功的状态下进行，所以如果要进行这些操作，必须先连接摄像机。

5)摄像机的一次性连接使用update_lan_status和start_connect可实现，具体可参考摄像机连接之通过IPCam摄像机的连接

6)监控摄像机的连接状态必须添加代理IPCam_Listener，但是使用完必须remove。IPCam_Listener的具体使用参考IPCam的基础接口

7)目前瑞彩的服务器的规则是一个摄像机只能被一个账号添加，所以被添加了必须unregister_from_sosocam。

8)只有relogin_to_sosocam成功了，摄像机才能像服务器上传报警图片以及实现推送功能。

6）示例：

```
/*第一步：创建更新一个帐号，且设置当前帐号*/
Storage.update_user("local@localhost", "", "", Storage.OFFLINE_USER, "");
Storage.set_current_user("local@localhost");

/*第二步：更新局域网信息，连接摄像机*/
public void update_lan_camera(String camera_id){
    LinkedHashMap<String, DISCOVERED_CAMERA_INFO> cameras =
    IPCamMgr.get_discovered_cameras_list();
    DISCOVERED_CAMERA_INFO camera = cameras.get(camera_id);
    if ((camera != null) && (! camera.current_ip.equals("0.0.0.0"))) {
        /*如果id在局域网内就更新局域网信息*/
        m_ipcam.update_lan_status(true, camera.current_ip, camera.port, camera.https);
    }
    /*连接摄像机
     *如果上一步有更新局域网信息，此刻就是局域网连接，否则是p2p连接
     */
    if (ERROR.NO_ERROR == m_ipcam.start_connect(false, 0)) {

        m_ipcam.add_listener(this);//监控摄像机的连接状态，详细参考摄像机的基础代理IPCam_Listener
    }
}

/*第三步：监控摄像机连接成功后，开始添加摄像机到服务器*/
public void on_status_changed(IPCam ipcam) {
    IPCam.CONN_STATUS status;
    IPCam.ERROR error;
    if (m_status == CONN_STATUS.CONNECTED){
        /*连接摄像机成功，开始添加到服务器*/
        add_camera_to_server();
    }
    else if (m_status == CONN_STATUS.IDLE){
        /*连接摄像机失败，删除摄像机代理*/
        m_ipcam.remove_listener();
    }
}
private void add_camera_to_server() {
    //添加到WebService
    if (WebService.ERROR.NO_ERROR != WebService.add_camera(name, m_ipcam.id(),
    m_ipcam.https(), m_ipcam.model, this)) {
        Log.e("SoDemo","failed_add_camera");
    } else {
        Log.e("SoDemo","add_camera_success");

    }
}
```

```
//WebService add_camera结果监听
public void on_add_camera_result(WebService.ERROR error, String obj_id) {
    if (WebService.ERROR.NO_ERROR == error) {
         /*监控到添加添加摄像机到服务器成功*/
        m_ipcam.relogin_to_sosocam(this); //第五步：重新登录到服务器
        add_camera_local(obj_id);  //第六步：添加到本地
    } else if (WebService.ERROR.REGISTERED_BY_OTHERS == error) {
        m_ipcam.unregister_from_sosocam(this);//第四步：摄像机已被其他账号添加，先解绑
    }else{
        /*添加摄像机到服务器失败，删除摄像机代理*/
        m_ipcam.remove_listener(this);
        }
    }

//IPCam unregister_from_sosocam结果监听
public void on_result(IPCam ipcam, ERROR error) {
    ipcam.stop_connect();
    if (m_state == STATE.UNREGISTER_CAMERA) {
        if (error == ERROR.NO_ERROR) {
            add_camera_to_server();//第四步：解绑成功，重新添加摄像机到服务器
        } else {
            m_ipcam.remove_listener(this);//解绑失败，需要删除摄像机代理
        }
    }
}

private void add_camera_local(String obj_id)
    {
        CAMERA_INFO add_cam = new CAMERA_INFO();
        if (obj_id == null) obj_id = "";
        add_cam.setObj_id(obj_id);
        add_cam.setAlias(m_camera_name);
        add_cam.setId(m_ipcam.camera_id());
        add_cam.setUser(m_ipcam.user());
        add_cam.setPwd(m_ipcam.pwd());
        add_cam.setHttps(m_ipcam.https());
        add_cam.setModel(m_ipcam.model);
        Storage.add_camera(add_cam);
        Storage.save_cameras();
        IPCam ipcam = IPCamMgr.add_camera(m_camera_name, m_ipcam.camera_id(),
        m_ipcam.user(), m_ipcam.pwd(), m_ipcam.https());//第七步：添加到IPCamMgr
        if (ipcam != null) {
            ipcam.model = m_ipcam.model;
            ipcam.sosocam_id = obj_id;
        }
    }

5.2摄像机的删除<服务器 / 本地 / IPCamMgr>
1）说明：摄像机的是指从服务器 / 本地 / IPCamMgr删除摄像机。
2）使用的类：WebService[可选]，Storage,IPCamMgr
3）函数：
/*
 *WebService从服务器上删除摄像机
 *参数：
 *obj_id:摄像机sosocam_id
 *delegate:接口delete_camera_listener，添加后可以监控删除结果
 */
public static ERROR delete_camera(String obj_id, delete_camera_listener listener);

/*
 *Storage从本地删除摄像机
 *参数：
 *camera_id:摄像机id
 */
public static void remove_camera(String camera_id);

/*
 *IPCamMgr删除摄像机
 *camera_id:摄像机id
 *另外IPCamMgr删除有个接口IPCamMgr_Listener，添加后可以监控删除结果
 */
public static IPCam delete_camera(String id);
```

4）步骤：

第一步：用WebService的方法从服务器上删除摄像机，如果不需用用到服务器平台这一步省略。<这一步骤可以添加WebServiceRemoveCameraDelegate代理监控删除结果，详情参考：IPCamMgr的服务器上删除摄像机>
第二步：用Storage的方法把从本地删除摄像机。
第三步：用IPCamMgr的方法从IPCamMgr中删除摄像机。<这一步骤可以添加IPCamMgrDelegate代理，监控添加，详情草考：IPCamMgr的cam添加和删除>

5）示例：

```
/*第一步：从服务器上删除摄像机+添加监听on_delete_camera_result*/
public void delete_camera_from_server(){
    if (WebService.ERROR.NO_ERROR != WebService.delete_camera(obj_id,
    DeleteCameraDialog.this)) {
         Log.e("SoDemo","remove camera from service fail!");
    }
}
/*第二步：从本地和IPCamMgr删除摄像机*/
public void on_delete_camera_result(WebService.ERROR error) {
    if (WebService.ERROR.NO_ERROR == error){
        /*监控到从服务器删除摄像机成功*/
        Storage.remove_camera(m_camera_id);//从本地删除摄像机
        Storage.save_cameras();
        IPCamMgr.delete_camera(m_camera_id); //从IPCamMgr删除摄像机
        Log.e("Sodemo","remove camera from service success!");
    }else{
        Log.e("Sodemo","remove camera from service fail!");
    }
}
```

5.3摄像机的修改更新
1）说明：摄像机的修改更新是指摄像机的做了修改要实时同步更新到服务器 / 本地 / IPCamMgr。但是更新内容有限制，比如只有名称更新了才能同步更新到服务器，其他的更新就不同步到服务器。详细如下：
服务器：更新名称
IPCamMgr：更新密码
Storage：更新名称 / 登录名 / 登录密码 / https / id /sosocam_id/cover

2）使用的类：WebService[可选]，Storage,IPCamMgr,IPCam

3）注意：
a）更新涉及到很多内容，这里着重讲密码和名称的更新。
b）密码修改，是指摄像机的登录密码修改。为了隐私保护，不会同步到服务器。所以只需要更新到本地和IPCamMgr。
c)
名称修改，不是针对摄像机修改的，只是当前帐号当前摄像机的一个昵称，故不涉及到操作摄像机，只要更新本地和服务器端就好。
d) 其他一些修改更新比如https等，<详情参考:Storage本地存储收藏摄像机管理和IPCam的操作>

4）函数

```
/*
 *IPCam修改摄像机登录密码
 *参数：
 *pwd：摄像机新密码
 *listener:接口reset_pwd_listener可以监控摄像机设置密码的结果
 *返回：
 *返回ERROR == ERROR.NO_ERROR表示设置新密码成功
 */
public ERROR reset_pwd(String pwd, reset_pwd_listener listener)

/*
 *IPCam设置IPCam的cam名称

 */
public void set_alias(String alias);

/*
 *WebService更新参数
 *参数：
 *obj_id:摄像机sosocam_id
 *params:新参数
 *listener:接口update_camera_listener可以监控更新结果
 *返回：
```

```
 *当ERROR == ERROR.NO_ERROR时表示更新执行成功
 */
 public static ERROR update_camera(String obj_id, String params,
 update_camera_listener listener);

/*
 *IPCamMgr更新密码到IPCamMgr
 *参数:
 *id:摄像机id
 *pwd:新密码
 */
public static void update_camera_pwd(String id, String pwd);

/*
 *Storage更新摄像机的名称
 *参数
 *camera_id:要被更新的摄像机id
 *camera_alias:摄像机昵称更新为camera_alias
 */
public static void update_camera_alias(String camera_id, String camera_alias);

/*
 *Storage更新摄像机的登录密码
 *参数
 *camera_id:要被更新的摄像机id
 *camera_pwd:摄像机登录密码更新为camera_pwd
 */
public static void update_camera_pwd(String camera_id, String camera_pwd);
```

5）步骤

摄像机登录密码和更新

第一步：用IPCam的reset_pwd修改摄像机登录密码。<详情参考：IPCam的重置新密码>

第二步：用IPCamMgr的密码更新方法和Storage的密码更新方法更新密码

名称修改和更新

第一步［可选］：用WebService的方法更新名称，可以添加接口update_camera_listener监控更新结果。<详情参考：WebService服务器上更新摄像机参数>

第二步：用Storage的方法更新名称。

第三步：更新新名称到IPCam，可以添加接口IPCam_Listener。<详情参考：摄像机基础接口>

6）示例：

摄像机登录密码和更新

```
/*第一步：设置摄像机新密码＋添加监听*/
if (ERROR.NO_ERROR == m_ipcam.reset_pwd(m_pwd, this)){ //重置密码
    m_state = STATE.SET_CAMERA;
}
/*第二步：更新到IPCamMgr和Storage*/
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());//更新保存到本地
            Storage.save_cameras();
            IPCamMgr.update_camera_pwd(m_ipcam.id(), m_ipcam.pwd());//更新到IPCamMgr
        }else{
            Log.e("Sodemo","----Failed---to----reset--pwd--");
        }
    }
}
```

名称修改和更新

```
/*第一步：更新名称到服务器＋添加接口update_camera_listener*/
if (WebService.ERROR.NO_ERROR !=
WebService.update_camera(Storage.get_camera_obj_id(m_camera_id),
                         "alias=" + name, this)) {
    Log.e("SoDemo","update name to service fail!");
} else {
    Log.e("SoDemo","update name to service succeed!");
}

/*第二步：更新名称到IPCam和本地*/
public void on_update_camera_result(WebService.ERROR error) {
    if (WebService.ERROR.NO_ERROR == error) {
        m_ipcam.set_alias(m_camera_name);//更新名称到IPCam
```

```
            Storage.update_camera_alias(m_camera_id, m_camera_name);//更新名称到本地
            Storage.save_cameras();
            Log.e("SoDemo","update_camera_name_ok");
        } else {
            Log.e("SoDemo","failed_update_camera_name");
        }
    }

    #6.摄像机连接操作
    说明：摄像机的连接有两种：第一种是是通过
    IPCamMgr来进行摄像机的连接，可根据网络状态和前后台的状态来自动管理多个摄像机的连接，Re
    ecam推荐此种方式。第二种是通过IPCam来进行摄像机的连接，此种方式由应用程序完全自我掌控
    摄像机的连接。

    6.1通过IPCamMgr摄像机的连接 [Reecam推荐的连接方式]

    1）说明:IPCamMgr摄像机的连接是指：通过IPCamMgr里的方法来连接摄像机，方法是把摄像机添加
    到IPCamMgr，这样IPCamMgr里的所有摄像机就自动进行连接。添加的cams会根据网络状态自动选择
    连接方式(局域网连接还是p2p连接),自动重连，还可设置moblie联网下摄像机的连接模式（禁止连
    接还是允许连接），还可操作让摄像机立即连接。

    2）使用的类：IPCamMgr，IPCam[可选]，Storage[可选]

    3)注意点：

        a)摄像机连接，至少要设置好id、user、pwd、https四个参数。其中https默认是false,如果
        此时摄像机https是true,这时是连接不上的。
        b)Android当程序从后台切到前台时，都要调用public static void
        restart()手动重新初始化库。和need_restart()配合使用。

        c)p2p连接摄像机时，摄像机的p2p_svr和id_type要书写正确[详细参考：wiki.reecam.cn/P2p
        /Config]

    4）函数：
    /*
     *SDK初始化／反初始化＋开始／停止IPCamMgr
     详情参考：IPCamMgr的"SDK初始化和反初始化"
     */
    IPCamMgr.init(this);//创建IPCamMgr 初始化

    IPCamMgr.deinit();//释放IPCamMgr，反初始化
    /*
     *局域网搜索[可选]
     * 详情参考：IPCamMgr的"局域网搜索"
     */
    public static LinkedHashMap<String, DISCOVERED_CAMERA_INFO>
    get_discovered_cameras_list();

    /*
     *IPCamMgr摄像机添加+添加的监听接口
     *详细参考：IPCamMgr的"添加和删除监听"
     */
    public static IPCam add_camera(String alias, String id, String user, String pwd,
    boolean https);

    /*
     *摄像机连接状态[可选]
     *详细参考：IPCam的"摄像机基础监听"
     */
    public void on_status_changed(IPCam ipcam);

    /*
     *立即连接[可选]
     *详细参考：IPCamMgr的"立即连接"
     */
    public static void reset_camera(String id);

    /*
     *moblie连接控制[可选]
     *详细参考：IPCamMgr的"数据(moblie)联网下的连接设置"
     */
    public static void set_forbidden_in_mobile(boolean forbidden);
```

5）步骤：

第一步：用IPCamMgr的init函数进行sdk初始化开始IPCamMgr;

第二步：使用IPCamMgr的add_camera函数添加摄像机到IPCamMgr类，摄像机添加后，就会自动连接摄像机。[所以如果使用这种方式来连接摄像机的话，添加摄像机成功后，必须要在添加摄像机到IPCamMgr，同时删除摄像机后，也必须从IPCamMgr删除摄像机]

第三步[可选步]：监控摄像机的连接状态。[摄像机的连接状态监控详细参考：<IPCam类的摄像机基础监听>]

第四步[可选项]：立即连接。IPCamMgr是多个摄像机的管理。你可以通过IPCamMgr的reset_camera优先设置摄像机立即连接。

第五步[可选项]：数据流量下的连接模式设置。可通过IPCamMgr的set_forbidden_in_mobile来设置（设置mobile联网下是否允许对摄像机的连接）。

6）例子：

```
/*
 *第一步：使用IPCamMgr的init函数初始化SDK
 */
IPCamMgr.init(this);

 /*
 *第二步：添加摄像机到PCamMgr
 */
LinkedHashMap<String, DISCOVERED_CAMERA_INFO> ipcam_list;
ipcam_list = IPCamMgr.get_discovered_cameras_list();

IPCam ipcam = IPCamMgr.add_camera(m_ipcam.alias, m_ipcam.id, "admin", "ipcam_pwd",
m_ipcam.https);//添加摄像机到IPCamMgr
/*
 *第三步[可选]：监控摄像机的连接状态 [注意IPCam_Listener的接口都要加进来]
 */
m_ipcam.add_listener(this);//添加IPCam_Listener代理

public void on_status_changed(IPCam ipcam){
    Log.e("SoDemo","cam connect status is::" + ipcam.status());
}
.
.
.

/*
 *第四步[可选]：立即连接
 */
 IPCamMgr.reset_camera(ipcam.id());

/*
 *第五步[可选]：摄像机数据流量下连接连接设置
 */
IPCamMgr.set_forbidden_in_mobile(true);//设置数据流量下禁止连接
```

6.2通过IPCam摄像机的连接，

1）说明：使用IPCam来进行摄像机的连接是指：通过IPCam的方法连接摄像机，方法是创建一个IPCam,然后调用连接函数连接。

2）使用的类：IPCamMgr，IPCam

3)注意点：

a)摄像机连接，至少要设置好id、user、pwd、https四个参数。其中https默认是false,如果此时摄像机https是true,这时是连接不上的。

b)Android当程序从后台切到前台时，都要调用public static void restart()手动重新初始化库。和need_restart()配合使用。

c)p2p连接摄像机时，摄像机的p2p_svr和id_type要书写正确[详细参考：wiki.reecam.cn/P2p/Config]

4）函数：

```
/*
 *更新摄像机局域网信息[可选]，注意：
```

```
 *1)如需采用局域网ip直连的方式连接摄像机，调用IPCam的start_connect函数来设置摄像机的局域网地址，否则缺省使用p2p的方式进行连接

 *2)start_connect可以在start_connect前后使用。如果在start_connect之前设置，摄像机就直接使用设置的模式in_lan去连接。如果在start_connect之后设置，摄像机就会先断开连接，再使用设置的模式in_lan去连接。

 *3)in_lan未改变，摄像机不会根据当前网络是局域网还是p2p自动连接。只会使用in_lan设置的模式去连接，直到in_lan值改变
*/
/*
 *参数如下：
 *ip：当前摄像机ip；
 *port：当前摄像机port
 *retryable:是否重连
 *retry_delay:重连间隔
 */
 public ERROR start_connect(String ip, int port, boolean retryable, int retry_delay);

/*
 *连接函数
 *参数如下：
 *retryable:是否重连,true:当因为网络错误时会自动重连接； false:不重连
 *retry_delay:int类型，单位ms,间隔多长时间进行重连。
 */
public ERROR start_connect(boolean retryable, int retry_delay);

/*断开摄像机
 说明：当不需要在连接摄像机时，可以使用stop_connect来主动断开摄像机的连接
 */
public void stop_connect();

5）步骤：
第一步：用IPCamMgr的init函数进行sdk初始化开始IPCamMgr;
第二步：new一个IPCam,用IPCam的start_connect连接摄像机。

6）示例
/*
 *第一步：使用PCamMgr的init函数初始化SDK
 */
IPCamMgr.init(this);

/*
 *第二步：new一个IPCam＋设置IPCam的参数
 */
public void set_cameras()
{
    IPCam m_ipcam = new IPCam();
    m_ipcam.set_id("SOSO-000019-VLTVM");
    m_ipcam.set_user("admin");
    m_ipcam.set_pwd("88888888");
    m_ipcam.set_https(false);
}

/*
 *第二步：监控摄像机的连接状态 [注意IPCam_Listener的接口都要加进来]
 */
m_ipcam.add_listener(this);//添加IPCam_Listener接口
-(void)on_status_changed:(id)ipcam{

    CONN_STATUS conn_status = m_ipcam.status();
}
.
.
.
/*
 *第三步：连接摄像机
 */
m_ipcam.start_connect(false, 0);

/*
```

```
     第六步：不需要连接时，断开摄像机连接;不需要监控摄像机状态时，删除监听;不需要使用sdk时，反初始化sdk
695   */
696  m_ipcam.remove_listener(this);
697  m_ipcam.stop_connect();
698  IPCamMgr.deinit();
699
700  #7.本地摄像机和服务器摄像机获取
701  1)说明：本地摄像机和服务器摄像机获取是指：获取被你添加到本地或服务器的摄像机，包括本地和服务器的获取以及对比同步。流程如下：
702  WebService里面获取添加到服务器的摄像机 ———> WebService对比同步 ———> Storage获取本地的摄像机 ———> 添加到IPCamMgr。
703  这些过程按照你的需求，择选，详细如下：
704  如果用到服务器平台管理摄像机，那么需要在WebService获取[可选项];
705  如果用到服务器平台管理摄像机，那么需要将添加到本地的摄像机和添加在服务器的摄像机进行对比同步[可选项];
706  不管用不用到平台，都需要在Storage获取[必须];
707  如果用到批量管理摄像机或使用IPCamMgr连接摄像机，那么需要添加到IPCamMgr。［推荐使用IPCamMgr管理];
708
709  2）使用类：storage，IPCamMgr，WebService
710  3）注意点：
711
         a)这里介绍怎么获取摄像机，是基于使用Storage存储的，而不是自己定义的存储。基于Storage存储的方法请参考上面的<摄像机的添加／删除／修改更新>。
712      b)存储的时候是使用Storage存储的，才能用以下介绍的方法获取。
713
         c)使用IPCamMgr的方式连接摄像机批量管理，从本地获取到摄像机后或和服务器同步后必须把从本地获取到的摄像机添加到IPCamMgr，这样子摄像机才会自动连接等，而后面需要获取摄像机都从IPCamMgr获取即可。[推荐方式]
714
         d)使用平台管理要注意，到登录状态改变时，每次重新获取到账号登陆成功，如果比较出本地和服务器的摄像机不一样，需要先把IPCamMgr的摄像机全部清除掉，然后在获取到本地的摄像机，逐个添加到IPCamMgr。
715  4）函数：
716
717  /*
718   *storag的方法
719   *storage获取本地存储的摄像机
720   *返回：返回一个CAMERA_INFO对象的列表，列表的每一个元素是一个CAMERA_INFO对象，如下：
721   *String id   //摄像机id
722   *获取：public String getId()
723
724   *String alias    //摄像机昵称
725   *获取：public String getAlias()
726
727   *String user    //摄像机登录用户名
728   *获取：public String getUser()
729
730   *String pwd   //摄像机登录密码
731   *获取：public String getPwd()
732
733   *boolean https  //加密安全传输，两个值 0:非加密    1:加密
734   *获取：public boolean getHttps()
735
736   *String obj_id   //摄像机sosocam_id
737   *获取：public String getObj_id()
738
739   *byte [] cover   //摄像机预览图data数据
740   *获取：public Bitmap getCover()
741
742   *String model //摄像机模型，两个值，0 云台机  1 卡片机
743   *获取：public int getModel()
744   */
745  public static ArrayList<CAMERA_INFO> get_cameras(); //storage获取本地存储的摄像机
746
747  /*
748   *IPCamMgr方法
749   *添加摄像机到IPCamMgr
750   *参数：
751   *alias:摄像机昵称
752   *id: 摄像机id
753   *user:摄像机登录用户名
```

```
 *pwd:摄像机登录密码
 *https:安全加密传输，两个值：true:加密 false:非加密
 *另外IPCamMgr添加有个监听IPCamMgr_Listener，添加后可以监控添加结果
 */
public static IPCam add_camera(String alias, String id, String user, String pwd,
boolean https);

/*
 *WebService的方法
 *WebService获取服务器当前帐号的摄像机
 *返回：返回一个列表，数组的每一个元素是一个SERVICE_CAMERA_INFO对象，如下：
 * String obj_id; 摄像机sosocam_id
 * String alias; 摄像机昵称
 * String id;  摄像机id
 * boolean https; 摄像机传输是否加密
 * String user;  摄像机登录用户名
 * String pwd;  摄像机登录密码
 */
public static ArrayList<SERVICE_CAMERA_INFO> svr_cameras = new
ArrayList<SERVICE_CAMERA_INFO>();
svr_cameras = WebService.cameras;//当前账号里添加的摄像机

/*
 *storage的方法
 *storage的比较方法
 *参数：
 *server_cameras_list:服务器上获取到的摄像机
 *返回：返回一个返回BOOL值
 *如果返回true表示:服务器上的cams和本地的cams一样
 *如果返回false表示:服务器上的cams和本地的cams不一样，且自动做同步
 */
Storage.merge_cameras(WebService.cameras);
```

5）步骤：

第一步：使用WebService的WebService.cameras获取当前帐号服务器的所有摄像机<使用平台管理，才用到此步骤，否则跳过>

第二步：使用storage的merge_cameras的方法比较同步<使用平台管理，才用到此步骤，否则跳过>

第三步：使用storage的get_cameras()获取本地当前帐号的所有摄像机，也是该账号所有添加的摄像机<如果没有用到服务器存储，直接使用此步骤>

第四步：使用IPCamMgr的add_camera添加摄像机到IPCamMgr<如果需要使用IPCamMgr管理>

第五步：如果使用平台管理摄像机，每次重新获取到账号登陆成功，且同步返回false,需先清掉IPCamMgr的摄像机，然后逐个添加到IPCamMgr。

6）示例：

```
/*第一步：获取服务器上当前存储的"添加摄像机"*/
public static ArrayList<SERVICE_CAMERA_INFO> svr_cameras = new
ArrayList<SERVICE_CAMERA_INFO>();
svr_cameras = WebService.cameras;

/*第二步：对比且同步本地和服务器上该帐号的摄像机*/
Storage.merge_cameras(WebService.cameras);

/*第三步：获取该帐号本地的所有摄像机*/
ArrayList<CAMERA_INFO> m_ipcams_list = Storage.get_cameras();//获取本地cams

/*第四步：添加摄像机到IPCamMgr，用于批量管理摄像机*/
public void add_cameras_to_ipcammgr(){
    IPCamMgr.init(this);
    CAMERA_INFO cam;
    int i;
    for(CAMERA_INFO cam : Storage.get_cameras()) {
        IPCam ipcam = IPCamMgr.add_camera(cam.getAlias(), cam.getId(),
        cam.getUser(), cam.getPwd(), cam.getHttps());//逐一添加cam到IPCamMgr
        /*如果还需要为ipcam设置什么属性，在此处添加，比如设置预览图/model/sosocam_id
         *这样子IPCamMgr的cam就具备这些属性了
         */
        ipcam.model = cam.getModel();
        ipcam.sosocam_id = cam.getObj_id();
        ipcam.cover = cam.getCover();

    }
}
/*第五步：如果使用平台管理摄像机，每次重新获取到账号登陆成功，且对比后发现有不同做同步
```

```
    操作后返回true,此时先清掉IPCamMgr的摄像机，然后逐个添加到IPCamMgr。*/
820 public void on_login_status_changed() {
821     switch (WebService.login_status) {
822     case LOGGED_ON:
823         if (Storage.merge_cameras(WebService.cameras)) {
824             IPCamMgr.delete_all_cameras();
825             for(CAMERA_INFO cam : Storage.get_cameras()) {
826                 IPCam ipcam = IPCamMgr.add_camera(cam.getAlias(), cam.getId(),
                        cam.getUser(), cam.getPwd(), cam.getHttps());
827                 ipcam.cover = cam.getCover();
828                 ipcam.model = cam.getModel();
829                 ipcam.sosocam_id = cam.getObj_id();
830             }
831         }
832         Storage.merge_collections(WebService.collections);
833         break;
834     }
835 }
836
837 #8.摄像机基本功能实现
838 8.1.摄像机视频
839
840 1）说明：摄像机视频主要是指：视频页面的设置，视频播放停止，视频高清标清切换，以及视频
    状态获取和视频的相关监听函数。
841 2）使用的类：IPCam,IPCamVideoView
842 3）函数
843 a)设置视频页面
844 /*
845  *IPCamVideoView类：设置视频页面
846  *参数：
847  *video_view：这是一个IPCamVideoView，IPCamVideoView是一个继承SurfaceView.
848  */
849 public void set_video_view(IPCamVideoView view);  //设置视频页面
850
851 b)视频状态
852 /*
853  *IPCam类：获取视频状态
854  *返回：返回PLAY_STATUS类型，详细如下PLAY_STATUS
855  */
856 public PLAY_STATUS video_status();
857 public enum PLAY_STATUS {
858         STOPPED, //停止
859         REQUESTING, //请求中
860         PLAYING; //播放中
861     }
862
863 c)视频质量参数
864 /*
865  *IPCam类：视频质量参数
866  *返回：都是返回int类型
867  */
868 public int video_render_fps(); //视频发送fps
869 public int video_recv_fps(); //视频接收的fbs
870 public int video_byterate(); //视频的byte
871
872 d)播放视频或切换播放视频流
873 /*
874  *IPCam类：播放视频或切换播放视频流
875  *参数：
876  *stream:视频流参数（0或1）,3518e只有两个分辨率:0:640*480(标清);1:1280*720(高清)
877  *返回：
878  *返回ERROR类型，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
879  */
880 public ERROR play_video(int stream);
881
882 e)停止播放视频
883 /*
884  *IPCam类：停止播放视频
885  */
886 public void stop_video();
887
888 f)视频监听
889 /*
```

```
 *IPCamVideo类

 *IPCamVideoView_Listener:需在视频上[注意是IPCamVideoView上不是IPCam]添加手势操作，就添加此监听
 *返回参数：
 *event:返回TOUCH_EVENT参数，详细如下：
 */
public void set_listener(IPCamVideoView_Listener listener);//设置IPCamVideoView_Listener接口
public void on_touch_event(TOUCH_EVENT event); //IPCamVideoView_Listener回调函数
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
        }

/*
 *IPCam类
 *IPCam_Listener:需在监控视频状态，就要在IPCam上添加此接口
 *IPCam_Listener的详细用法请参考IPCam的基础接口
 */
public void add_listener(IPCam_Listener listener);//设置IPCam_Listener接口
public void on_video_status_changed(IPCam ipcam);
//IPCam_Listener回调函数，视频状态改变时回调
public void remove_listener(IPCam_Listener listener); //删除IPCam_Listener代理
4）注意点：
    a)播放视频或获取视频状态或设置视频相关，必须是在摄像机连接成功为前提

    b)添加IPCam_Listener代理后，代理回调的所有函数都要添加，即使不用，而不是只添加视频状态回调代理。如果没有添加齐全，会编译不过

    c)因很多状态都是IPCam_Listener接口，所以要注意IPCam_Listener接口不允许多次添加，不需要使用时也要记得remove。

5）步骤：
第一步：创建并设置视频页面。
第二步:实现接口IPCamVideoView_Listener和IPCam_Listener，这一步根据需要添加。<可选>
第三步：进行视频操作：播放停止/视频流切换。
第四步：处理回调函数：添加了监听才有回调函数。<可选>

6）示例：
/*第一步：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
int m_playing_video_stream = 0;   //定义默认播放视频流

/*第二步：在视频上添加代理IPCamVideoView_Listener和IPCam_Listener*/
m_video_view.set_listener(this); //m_videoview添加IPCamVideoView_Listener不是m_ipcam
m_ipcam.add_listener(this); //m_ipcam添加IPCam_Listener

/* 第三步:视频播放和停止*/
case R.id.play_start:
    {
        if (m_ipcam.video_status() == PLAY_STATUS.PLAYING) { //判断当前视频状态
                m_ipcam.stop_video();// 停止播放视频
        }else{

            m_ipcam.play_video(m_playing_video_stream);//播放m_playing_video_stream路视频流
            /*视频质量如下*/
        }
        Log.e("视频发送：",m_ipcam.video_render_fps() + "fps"); //发送fps
        Log.e("视频接收：",m_ipcam.video_recv_fps() + "fps"); //接收fps
        Log.e("视频当前：",m_ipcam.video_byterate() + "byte"); //当前视频byte
    }

/*第三步:设置播放视频流，即高清和标清设置或切换 */
case R.id.play_change_screen:
    {
        if (m_playing_video_stream == 0){
            m_playing_video_stream = 1;
```

```
            m_ipcam.stop_video();
            m_ipcam.play_video(m_playing_video_stream);
        }else{
            m_playing_video_stream = 0;
            m_ipcam.stop_video();
            m_ipcam.play_video(m_playing_video_stream);
            }
    }

/*第四步：IPCamVideoView_Listener回调实现*/
public void on_touch_event(IPCamVideoView_Listener.TOUCH_EVENT event) {
        switch (event) {
        case CLICK :
            Log.e("SoDemo","单点了一下");
            break;
        case MOVE_UP:
            Log.e("SoDemo","向上移动了一下");
            break;
        case MOVE_DOWN:
            Log.e("SoDemo","向下移动了一下");
            break;
        case MOVE_LEFT:
            Log.e("SoDemo","向左移动了一下");
            break;
        case MOVE_RIGHT:
            Log.e("SoDemo","向右移动了一下");
            break;
        default:
            break;
        }
    }
/*第四步：IPCamD_Listener摄像机视频状态改变*/
public void on_video_status_changed(IPCam ipcam) {
    PLAY_STATUS status = ipcam.video_status();
    }

#8.2 摄像机音频
1)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。
2)使用类：IPCam
3)函数：
a）音频状态
/*
 *IPCam类方法
 *返回：
 *返回PLAY_STATUS，详细如下：
 */
public PLAY_STATUS audio_status();
public enum PLAY_STATUS {
        STOPPED, //停止
        REQUESTING, //请求中
        PLAYING; //播放中
    }

b)音频质量
/*
 *IPCam类方法
 *返回：
 *返回int类型，真实大小，详细如下：
 */
public int audio_sps(); //音频的sps
public int audio_byterate(); //音频的byte
c）开启音频
/*
 *IPCam类方法:开启音频
 *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
 */
public ERROR play_audio();

d）关闭音频
/*
 *IPCam类方法:关闭音频
```

```
 */
public void stop_audio();

e）音频监听
/*
 *IPCam类:IPCam_Listener接口
 *IPCam_Listener的详细用法请参考IPCam的基础监听
 */
public void add_listener(IPCam_Listener listener);//设置IPCam_Listener接口
public void on_audio_status_changed(IPCam ipcam);
//IPCam_Listener回调函数，音频状态改变时回调
public void remove_listener(IPCam_Listener listener); //删除IPCam_Listener代理

4）注意点：
    a)开启停止音频或获取音频状态等相关操作，必须是在摄像机连接成功为前提

    b)添加IPCam_Listener接口后，所有的回调函数都要添加，即使不用，而不是只添加音频状态回调函数。如果没有添加齐全，编译会报错

    c)因很多状态都是IPCam_Listener接口，所以要注意IPCam_Listener接口不允许多次添加，不需要使用时也要记得remove。

5）步骤：
第一步:添加接口IPCam_Listener，这一步根据需要添加。<可选>
第二步：摄像机连接成功即可直接开启停止音频等操作。
第三步：处理回调函数，添加了接口才有回调函数。<可选>

6）示例：
/*第一步：添加IPCam_Listener接口*/
m_ipcam.add_listener(this); //m_ipcam添加监听
/*第二步：音频操作 */
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
        }
        /*音频质量如下*/
        Log.e("音频质量：",m_ipcam.audio_sps() + "fps"); //音频的fps
        Log.e("音频质量：", m_ipcam.audio_byterate() + "byte"); //音频的byte
    }

/*第二步：回调函数 */
public void on_audio_status_changed(IPCam ipcam) {
    PLAY_STATUS status = m_ipcam.audio_status(); //获取音频当前状态
}

#8.3 摄像机对讲
1)说明：摄像机音频，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。
2)使用类：IPCam
3)函数：
a）对讲状态
/*
 *IPCam类：对讲状态获取
 *返回：
 *返回PLAY_STATUS，详细状态如下：
 */
public ERROR start_speak();
public enum PLAY_STATUS {
        STOPPED, //停止
        REQUESTING, //请求中
        PLAYING; //播放中
    }
b)对讲质量
/*
 *IPCam类：对讲状态获取
 *返回：
 *返回int类型，真实大小，详细如下：
 */
public int speak_sps(); //对讲的fps
public int speak_byterate() //对讲的byte
```

c）开启对讲

```
/*
 *IPCam类：开启对讲
 *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
 */
public ERROR start_speak();
```

d）关闭对讲

```
/*
 *IPCam类：关闭对讲
 */
public void stop_speak();
```

e）对讲接口

```
/*
 *IPCam类:IPCamDelegate代理
 *IPCamDelegate的详细用法请参考IPCam的基础代理
 */
public void add_listener(IPCam_Listener listener);//设置IPCam_Listener接口
public void on_speak_status_changed(IPCam ipcam);
//IPCam_Listener回调函数，对讲状态改变时回调
public void remove_listener(IPCam_Listener listener); //删除IPCam_Listener监听
```

4）注意点：

a)开启停止对讲或获取对讲状态等相关操作，必须是在摄像机连接成功为前提

b)添加IPCamDelegate代理后，代理回调的所有函数都要添加，即使不用，而不是只添加音频状态回调代理。如果没有添加齐全，会引起程序奔溃

c)因很多状态都是IPCamDelegate代理，所以要注意IPCamDelegate代理不允许多次添加，是需要使用时也要记得remove。

5）步骤：

第一步:添加接口IPCam_Listener，这一步根据需要添加。<可选>

第二步：摄像机连接成功即可直接开启停止对讲等操作。

第三步：处理回调函数：添加了接口才有回调函数。<可选>

6）示例

```
/*第一步：添加IPCam_Listener接口*/
m_ipcam.add_listener(this); //m_ipcam添加监听
/*第二步：对讲操作 */
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();
    /*对讲质量如下*/
    Log.e("对讲质量：",m_ipcam.speak_sps() + "fbs"); //对讲的fps
    Log.e("对讲质量：",m_ipcam.speak_byterate() + "byte"); //对讲的byte
}
/*第二步：回调函数 */
public void on_speak_status_changed(IPCam ipcam){
    PLAY_STATUS status = m_ipcam.speak_status(); //获取对讲当前状态
}
```

#8.4 摄像机拍照

1）说明：摄像机拍照是指：把当前画面拍照保留到本地。有两个函数可以实现拍照，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<推荐方法>。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。

2)使用类：IPCam

3)函数：

```
/*
 *IPCam方法：拍照
 *返回：
 *返回ERROR类型，当返回类型是ERROR.NO_ERROR时，表示拍照成功。
 */
```

函数1：public String snapshot();

```
/*
 *IPCam方法：拍照
 *参数：
 *photo_file_path：照片存储的完整路径
 *返回：
```

```
 *返回ERROR类型，当返回类型是ERROR.NO_ERROR时，表示拍照成功。
 */
public String snapshot(String snapshot_file_path);
```

4）注意点：

a)使用拍照功能必须以视频开启成功为前提，也就是能看到视频画面后才能拍照。

b)使用snapshot保存的路径是在文件管理的sosocam目录下，而不是系统相册，需要放到系统相册，需要再导出

5）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");
    }
```

#8.5 摄像机本地录像

1）说明:摄像机本地录像是指：把当前视频录制到本地。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。有两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。

2）使用类:IPCam

3）函数：

a）本地录像状态

```
/*
 *IPCam方法：本地录像状态
 *返回：
 *返回PLAY_STATUS，详细如下：
 */
public PLAY_STATUS local_record_status();
public enum PLAY_STATUS {
        STOPPED, //停止
        REQUESTING, //请求中
        PLAYING; //播放中
    }
```

b）开启本地录像

```
/*
 *IPCam方法：开启本地录像
 *返回：
 *返回ERROR类型，当返回类型是ERROR.NO_ERROR时，表示本地录像成功。
 */
函数1:public String start_local_record();

/*
 *IPCam方法：开启本地录像
 *参数：
 *record_file_path：录像存储的完整路径
 *返回：
 *返回ERROR类型，当返回类型是ERROR.NO_ERROR时，表示本地录像成功。
 */

函数2:public String start_local_record(String record_file_path);
```

c）停止本地录像

```
/*
 *IPCam方法：停止本地录像
*/
public void stop_local_record();
```

d）本地录像监听

```
/*
 *IPCam类:IPCam_Listener监听
 *IPCam_Listener的详细用法请参考IPCam的基础监听
 */
public void add_listener(IPCam_Listener listener);//设置IPCam_Listener接口
public void on_camera_recording_changed(IPCam ipcam);
//IPCam_Listener回调函数，录像状态改变时回调
public void remove_listener(IPCam_Listener listener); //删除IPCam_Listener监听
```

4)注意点：

a)使用本地录像功能必须以视频开启成功为前提，也就是能看到视频画面后才能录制。

b)添加IPCam_Listener接口后，所有的回调函数都要添加，即使不用，而不是只添加本地录像状态回调函数。如果没有添加齐全，编译会报错

c)因很多状态都是IPCam_Listener接口，所以要注意IPCam_Listener接口不允许多次添加，不需要使用时也要记得remove。

d)使用start_local_record保存的路径是在文件管理sosocam目录下，而不是系统相册，需要放到系统相册，需要再导出

5)步骤

第一步:添加代理IPCam_Listener，这一步根据需要添加。<可选>

第二步：摄像机视频出来后即可直接开启停止等操作。

第三步：处理回调函数：添加了接口才有回调函数。<可选>

6）示例：

```
/*第一步：添加IPCam_Listener接口*/
m_ipcam.add_listener(this); //m_ipcam添加IPCam_Listener

/*第二步：录像操作 */
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;
            }
        }

/*第二步：回调函数 */
public void on_camera_recording_changed(IPCam ipcam){
    PLAY_STATUS status = m_ipcam.local_record_status(); //获取本地录像当前状态
}
```

#9.摄像机PTZ操作<包括云台操作>

1）说明：摄像机ptz操作是指：通过ptz_control命令来实现摄像机的一些控制命令。主要是云台操作功能的实现。

2）使用类：IPCam

3）函数：

```
/*
 *IPCam方法：ptz控制命令
 *参数：
 *cmd:功能命令，详细查看一下的IPCAM_PTZ_CMD
 *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>
 */
public ERROR ptz_control(PTZ_CMD cmd, int param)

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
    }
```

4）示例：

```
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;
```

#10.摄像机CGI命令设置参数<如报警布防和TF卡录像>

1）说明：摄像机CGI命令设置参数是指通过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。
2）使用类：IPCam
3）函数：

```
/*
 *IPCam方法：CGI设置参数
 *参数：
 *params:cgi参数，由reecam提供，下面列举一些
 *params:set_params_listener监听，可以监控cgi设置的结果
 */
public ERROR set_params(String params, set_params_listener listener)
/*其中arm_schedule有三个值，0:布防，1:计划布防，2:撤防*/
报警params："save=1&reinit_alarm=1&arm_schedule=0"
/*
 *record_schedule_list=[]    停止tf卡录像
 *record_schedule_list=[{\"start\":0,\"end\":96,\"day\":127}]    开启录像
 */
录像params："save=1&reinit_record=1&record_schedule_list=[]"
重启params:"reboot=1"
/*
 *set_params_listener回调函数
 *返回参数：
 *ipcam：返回一个IPCam对象
 *ERROR类型，当返回是ERROR.NO_ERROR，表示设置成功
 */

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

4）注意：

a)这些CGI的参数，不会主动反馈参数的值，如果想要知道此时参数是什么值什么状态，请参考<IPCam的CGI命令设置参数>，有详细介绍cgi
命令设置参数以及参数值的获取。
b)CGI命令设置参数是以摄像机连接成功为前提的。

5)示例

```
/*第一步：cgi设置参数+添加监听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");
        }
    }
}
```

#11.无线网络设置
说明：无线网络设置是指：把摄像机设置成连接某一个无线路由器。无线设置分两大类：手机已连接的摄像机的无线网络设置和手机未连接的摄像机无线网络设置。

11.1已连接的摄像机的无线网络设置
11.1web设置无线

1）说明：web设置无线是指通过html实现的，你只需要加载html，其他的都由html实现。它可以实现wifi的选择，保存多个Wi-Fi，增加和删除Wi-Fi。

2）注意：

1）这个html由reecam提供，需要的可以联系reecam

3)示例：

```
 public void set_wifi_from_web()
{
    private WebView mwebView;
    String m_url = "file:///android_asset/app/wifi.html?" + "host="+ m_ipcam.ip() + "&port=" +
    m_ipcam.port() + "&user=" + user + "&pwd=" + pwd + "&ssl=" + (m_ipcam.https()?1:0) +"&sys=0"+ "&lng=" + Tools.check_language());
    mwebView.loadUrl(m_url);
}
```

11.2APP设置无线

说明：无线网络设置包括，扫描周围wifi和设置摄像机wifi功能。

使用类：ipcam

函数：

a）扫描wifi

注意：

1）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前必须先要连接摄像机成功。

2）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)。

```
/*
 *扫描函数
 *返回：
 *返回ERROR类型，当返回是ERROR.NO_ERROR，表示wifi_scan执行成功
 */
public ERROR wifi_scan(wifi_scan_listener listener);

/*
 *扫描结果函数回调
 *参数：
 *ipcam：返回一个IPCam对象
 *error:ERROR类型，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
*/
public void on_result(IPCam ipcam, ERROR error, ArrayList<ap_info> ap_list);
/*
 *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;
    }
```

b）摄像机无线设置

注意点：

1）函数1和函数2都可以实现设置无线功能。

2）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。
3）函数2没有无线测试功能，直接设置。
4）不管摄像机当前是有线连接还是无线连接都可以使用。

5）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。

```
/*
 *参数：
 *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_key_index：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_type:，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
 *listener:监听set_wifi_listener
 */
```

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);
```

```
 public enum WIFI_WEP_KEY_TYPE {
        HEX, //hex类型
        ASCII; //ASCII类型
    }

/*
 *set_wifi_listener回调函数 Wi-Fi设置进程
 *参数返回：
 *ipcam：返回一个IPCam对象
 *state：设置Wi-Fi的进程，是一个SET_WIFI_STATE类型，详细参考下面的SET_WIFI_STATE
 */
public void on_progress(IPCam ipcam, SET_WIFI_STATE state);

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

/*
 *set_wifi_listener回调函数 Wi-Fi设置结果
 *参数返回：
 *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。
 */
public void on_result(IPCam ipcam, ERROR error);
```

步骤：
第一步：扫描wifi
第二步：选择wifi设置无线
第三步：监控无线设置过程和结果

3)示例

```
/*第一步：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");
    }
}
```

```
1488    /*第二步：扫描结果，IPCam.wifi_scan_listener回调 */
1489    public void on_result(IPCam ipcam, ERROR error, ArrayList<ap_info> ap_list){
1490        if (ERROR.NO_ERROR == error) {
1491            m_aps_list = ap_list;
1492            if (m_ap_list.size() == 0) {
1493                    m_listview_wifi.setAdapter(null);
1494                    m_state = STATE.SCANNING_NOTHING;
1495                    Log.e("SoDemo","SCAN_WIFI_NOTHING");
1496                } else {
1497                    m_listview_wifi.setAdapter(m_wifiListAdapter);
1498                    m_state = STATE.SCANNING_OK;
1499                    Log.e("SoDemo","SCAN_WIFI_OK");
1500                }
1501
1502        } else {
1503            m_listview_wifi.setAdapter(null);
1504            Log.e("SoDemo","SCAN_WIFI_FAILED");
1505        }
1506    }
1507
1508    /*第三步：设置wifi*/
1509    public void set_wifi() {
1510        m_ap.auth = ap.auth;
1511        m_ap.encrypt = ap.encrypt;
1512        m_ap.ssid = ap.ssid;
1513        if (m_ap.auth == WIFI_AUTH.OPEN) {
1514            set_camera();
1515        }else{
1516            key = m_edittext_key.getText().toString();
1517        }
1518    }
1519
1520    private void set_camera() {
1521        WIFI_WEP_KEY_TYPE type = WIFI_WEP_KEY_TYPE.ASCII;
1522        if (m_ap.auth == WIFI_AUTH.WEP) {
1523            if ((m_ap.key.length() == 5) || (m_ap.key.length() == 13))
1524                type = WIFI_WEP_KEY_TYPE.ASCII;
                    //有加密方式，且key的长度是5或13时type为ASCII
1525            else
1526                type = WIFI_WEP_KEY_TYPE.HEX;//其他加密，type为hex
1527        }
1528        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*/
1529            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)) {
1530                m_state = STATE.SET_CAMERA;
1531                Log.e("SoDemo","succeed_set_wifi");
1532            } else {
1533                Log.e("SoDemo","failed_set_wifi");
1534                }
1535        }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*/
1536            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)) {
1537                m_state = STATE.SET_CAMERA;
1538                Log.e("SoDemo","succeed_set_wifi");
1539            } else {
1540                Log.e("SoDemo","failed_set_wifi");
1541            }
1542        }
1543    }
1544    /*第四步：设置wifi进度和结果监控，IPCam.wifi_scan_listener回调*/
1545    public void on_progress(IPCam ipcam, SET_WIFI_STATE state) {
1546        switch (state) {
1547            case SET_WIFI_STATE.SETTING: //Wifi设置中
1548                m_state = SETTING_WIFI;
1549                break;
1550            case SET_WIFI_STATE.TESTING: //wifi测试中
1551                m_state = TESTING_WIFI;
1552                break;
1553            case SET_WIFI_STATE.SAVING: //wifi保存中
1554                m_state = SAVING_WIFI;
1555                break;
1556            default:
1557                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");
    }
}
```

11.2未连接的摄像机的无线网络设置

说明：未连接的摄像机的无线网络设置：是指摄像机未添加到app里，或摄像机添加到app但是未连接成功，在这两种情况下进行的摄像机无线设置。未连接的摄像机的无线网络设置有两种：声音（sound）设置无线网络，smartlink设置无线。这两种方式可以单独使用，也可以同时使用。但同时使用时，一定要注意。当一种方式成功时，一定要记得把另外一种设置方式停止。即smartlinK设置先成功时，要记得把声音（sound）止。当声音（sound）设置先成功时，要记得把smartlink停止。

#11.2.1声音（sound）设置无线

1）说明：使用声音设置无线就是利用声音传输来设置无线网络。可以实现开始/停止无线网络设置，也可监控无线设置过程和结果。
2）使用类：SoundWaveWifiSetting,Tools，IPCamMgr
3）注意点：
    a)SDK初始化成功
    a)系统固件版本为："^\\d+.\\d+.\\d+.4.\\d+$"
    b)书写了设备id,且id前缀为"RTEST-","SOSO-","RCAM-","LSCAM-","YSTC-"
    c)手机喇叭是好的

    d)设置成功后返回的id是分两部分编码返回，需查询id是否在局域网，需要先用Tools的方法对id进行编码再比较
    e)获取局域网摄像机之前，必须要先初始化IPCamMgr.init()。

4）函数：
a）开始声音设置无线

```
/*
 *SoundWaveWifiSetting开始无线设置
 * 参数如下：
 * ssid:wifi的ssid
 * psk:wifi的密码
 * listener:接口，当前对象需实现sound_wave_wifi_setting_listener接口
 * 返回：
 * 返回一个BOOL值，当返回true时，表示函数执行成功，否则执行失败。
 */
```

说明：使用以下函数可以通过声音设置无线网络也可以监控设置的过程。
函数：public boolean start(String ssid, String psk, sound_wave_wifi_setting_listener listener)

```
/*
 *这是接口sound_wave_wifi_setting_listener的回调函数
 *开始声音设时，添加了此接口
 *只要声音设置无线状态有改变时候就会调用此函数
 */
```

回调函数：public void on_sound_wave_wifi_setting_state_changed();

b）停止声音设置无线

```
/*
 *SoundWaveWifiSetting停止声音设置无线
 *SoundWaveWifiSetting设置成功自动就停止了声音设置，不需要主动去设置停止
 *当退出设置或者smartlink先设置成功就需要主动停止sound设置
 */
 public void stop();
```

c）声音设置无线信息获取：

```
/*
```

```
 *声音设置无线结果，共有两种结果，详细如下：
 *声音设置结果通过public SOUND_WAVE_WIFI_SETTING_RESULT result()获取
 */
public static enum SOUND_WAVE_WIFI_SETTING_RESULT {
        SUCCEED, //设置成功
        TIMEOUT  //设置超时，需重新设置
    }

/*
 *声音设置无线状态，共有三种状态，详细如下：
 *状态通过public SOUND_WAVE_WIFI_SETTING_STATE state()
 */
public static enum SOUND_WAVE_WIFI_SETTING_STATE {
        IDLE,           //状态为空时，可查询当前设置结果，进行不同操作
        SEND_SETTING,   //正在发送Wifi设置信息
        WAIT_CONFIRM    //等待验证，即等待设备响应
    };
```

d) 查询ID是否在局域网里

```
/*
 *因为id太长，摄像机传不过来，所以把id编码分成name_of_camera_id和serial_of_camera_id来传
 *特征码1和特征码2说明：
 *
 1.设置wifi成功的机器，可在局域网中通过IPCamMgr.get_discovered_cameras_list()方法搜索到；
 *
 2.得到该camera的id后，可通过Tools.get_name_of_camera_id(id)的方法得到一个特征码m1，通过Tools.get_serial_of_camera_id(id)的方法得到一个特征码m2；
 *
 3.第二步中得到的特征码m1和该编码生成的特征码1进行比对，并且m2和该编码生成的特征码2相等则此ID即为当前声波设置wifi成功的机器；
 */

/*
 *SoundWaveWifiSetting&Tools编码id
 *id太长，分成两部分编码，比较的时候
 *SoundWaveWifiSetting的name_of_camera_id和Tools的get_name_of_camera_id比较
 *SoundWaveWifiSetting的serial_of_camera_id和Tools的get_serial_of_camera_id比较
 */
public int name_of_camera_id(); //SoundWaveWifiSetting设置成功，返回编码的id

public int serial_of_camera_id(); //SoundWaveWifiSetting设置成功，返回编码的id

public static int get_name_of_camera_id(String id);  //Tools编码id

public static int get_serial_of_camera_id(String id); //Tools编码id
```

5）步骤：

第一步：创建SoundWaveWifiSetting对象。
第二步：开始sound无线设置
第三步：实时监控sound的无线设置状态和结果
第四步：设置成功后，获取摄像机的id和局域网内的摄像机一一比较，确认是否成功设置，摄像机是否再局域网了

6）示例：

```
/*
 第一步：创建一个SoundWaveWifiSetting对象
 */
private SoundWaveWifiSetting m_sound_wave_wifi_setting = new SoundWaveWifiSetting();
m_sound_wave_wifi_setting.init(m_parent);
/*
 *第二步：发送声音 实现接口sound_wave_wifi_setting_listener
 */
 boolean  m_sound_wifi = m_sound_wave_wifi_setting.start(m_ap.ssid, m_ap.key,
 AddCamerabySoundDialog.this);
   if (! m_sound_wifi){
        Log.e("SoDemo","SEND_WIFI_INFO_FAIL");
    }

/*
 *监测 声音设置无线的过程
```

```
 */
public void on_sound_wave_wifi_setting_state_changed() {
    SOUND_WAVE_WIFI_SETTING_STATE state = m_sound_wave_wifi_setting.state();
    SOUND_WAVE_WIFI_SETTING_RESULT result = m_sound_wave_wifi_setting.result();
    Log.e("SoDemo","SOUND_WAVE_WIFI_SETTING_STATE = " + state
    +"SOUND_WAVE_WIFI_SETTING_RESULT = " + result);
    if (state == SOUND_WAVE_WIFI_SETTING_STATE.SEND_SETTING) {
            Log.e("SoDemo","Transmitting wireless setting now");//正在传送无线设置
            ...;
    } else if (state == SOUND_WAVE_WIFI_SETTING_STATE.WAIT_CONFIRM) {
            Log.e("SoDemo","Wait device response");//正在等待设备响应 ...;
    } else if (state == SOUND_WAVE_WIFI_SETTING_STATE.IDLE) {
        if (result == SOUND_WAVE_WIFI_SETTING_RESULT_SUCCEED){
            m_name_of_camera_id = m_sound_wave_wifi_setting.name_of_camera_id();
            m_serial_of_camera_id = m_sound_wave_wifi_setting.serial_of_camera_id();
            Log.e("SoDemo","Search camera in LAN");//设置成功可在局域网开始搜索此机器
            discover_camera();
        }
    }
}

/*
 *第三步：设置成功后，查询id是否在局域网内,来获取id
 */
public void discover_camera(){
    LinkedHashMap<String, DISCOVERED_CAMERA_INFO> cameras =
    IPCamMgr.get_discovered_cameras_list();
    String camera_id = null;
    int name_of_camera_id;
    int serial_of_camera_id;

    for (String id : cameras.keySet()) {
        /*对局域网的摄像机id进行编码*/
        name_of_camera_id = Tools.get_name_of_camera_id(id);
        serial_of_camera_id = Tools.get_serial_of_camera_id(id);
        /*查id 是否在局域网内*/
        if (name_of_camera_id == m_name_of_camera_id &&
            serial_of_camera_id == m_serial_of_camera_id) {
            camera_id = id;
            Log.e("SoDemo","Camera id is" + id);
            break;
        }
    }
/*
 *不需要了，强制取消SoundWaveWifisetting设置。设置成功了，不需要做停止的的操作
 */
    m_sound_wave_wifi_setting.stop();
    m_sound_wave_wifi_setting.deinit();

#11.2.2.smartlink设置无线网络

说明：smartlink设置无线网络是指通过手机广播一个UDP数据包,包中放置SSID跟密码等信息。摄像机接收到该包，解析后，获得
SSID和密码等数据，就可以配置自己，连接路由器了。smartlink设置无线包括开始/停止smartlink设置和smartlink设置成功返回摄像机id。

使用的类：smartlink
注意点：
    a)SDK初始化成功
    b)系统固件版本为："^\\d+.\\d+.\\d+.4.\\d+$"
    c)书写了设备id,且id前缀为"RTEST-","SOSO-","RCAM-","LSCAM-","YSTC-"
函数：
/*
 *开始smartlink设置
 *参数如下：
 *ssid:wifi的ssid
 *psk:wifi的密码
 *listener:wifi_matching_listener接口
 * 返回：
 * 返回一个boolean值，当返回true时，表示函数执行成功，否则执行失败。
 */
public static boolean start(String ssid, String psk, wifi_matching_listener listener)
```

```
/*
 *这是接口wifi_matching_listener的回调函数
 *开始smartlink设置时时，实现此接口
 *当smartlink设置无线成功后就会调用此函数且返回成功摄像机的id,其他时候不会掉用此函数
 */
public void on_wifi_matching_succeed(String camera_id);

/*
 *smartlink停止设置无线
 *smartlink设置成功自动就停止了设置，不需要主动去设置停止
 *当退出设置或者SoundWaveWifiSetting先设置成功就需要主动停止sound设置
 */
public static void stop();
```

步骤：
第一步：创建WiFiMatching对象
第二步：开始smartlink设置且实现接口wifi_matching_listener
第三步：如果smartlink设置成功了且前面添加了实现接口wifi_matching_listener接口，传递id

3）示例：

```
/*
 *第一步：创建一个WiFiMatching对象
 */
private WiFiMatching m_smart_link_wifi_setting = new WiFiMatching();

/*
 *第二步：发送smartlink 实现接口wifi_matching_listener
 */
 boolean  m_smart_link = m_smart_link_wifi_setting.start(m_ap.ssid, m_ap.key,
 AddCamerabySmartDialog.this);;
    if (! m_smart_link){
       Log.e("SoDemo","Smart Link start OK!");
    }

/*
 *第三步：wifi_matching_listener回调函数
 */
public void on_wifi_matching_succeed(String camera_id){
    if (camera_id != "") {
        /*设置成功获取到id,可以进行下一步操作，比如添加摄像机*/
        Log.e("SeDemo","smartlink设置成功, camera_id= " + camera_id);
        }
}

/*
*第四步：不需要了，强制取消smartlink设置，设置成功了，不需要做停止的的操作
*/
m_smart_link_wifi_setting.stop();
```

#12 摄像机本地录像和拍照文件管理<本地录像和图片获取 / 删除 / 导出到系统相册>

1）说明：摄像机本地录像和拍照文件管理是指把存储到本地的录像和拍照文件获取出来，导出到系统相册以及删除等操作。
2）使用类：storage,Tools
3）注意：以下介绍的文件的获取和删除都是基于保存方式是使用瑞彩的方式保存的，即保存路径不是自己定义的。
4）函数：
a)获取本地录像和图片

```
/*
 *storage方法:获取本地录像和图片
 *参数:
 *camera_id:摄像机id
 *返回:返回一个list，列表的每一个元素是一个ALBUM_ITEM对象，详细如下:
 *Date t       //时间Date
 *long size    // 文件大小
 *boolean image  //文件类型，两个值，true:图片 false:录像
 *String path    //文件路径
 */
public static ArrayList<ALBUM_ITEM> get_local_records_list(String camera_id)
```

示例1：

```
Storage.get_local_records_list(cam.id());

b）删除文件
/*
 *storage方法：删除指定摄像机的所有的录像和拍照文件
 *参数：camera_id:摄像机id
 */
public static void delete_local_records_list(String
camera_id);//删除该id的摄像机下所有图片和录像文件

/*
 *storage方法：删除指定摄像机的指定的录像或拍照文件
 *参数：path:文件路径
 */
public static void delete_local_records_list_file(String path);
//删除某一个文件（图片或录像文件）

示例2：
Storage.delete_local_records_list("RTEST-001015-DUGMR");//删除所有文件
Storage.delete_local_records_list_file(path);//删除当前路径对应的一个文件

c)导出文件
/*
 *Tools方法：把图片导出到系统相册
 *参数：
 *path:图片/录像路径
 */
public static void add_media(Context context, String path)//导出图片/录像到系统相册
示例3：
Tools.add_media(LiveActivity.this, filepath);

#13 推送的实现
说明：推送的实现是基于Jpush上实现的，相关lib和使用请参考jpush官方文档。这里主要介绍set
tag瑞彩服务器部分。瑞彩实现推送两种方式，一种是以账号为id来settag,另外是以摄像机为id来
settag。
#pragma mark --13.1以账号为id来settag(使用了平台功能)
13.1以账号为id来settag

1）说明：以账号为id来settag是指：使用账号登陆，以账号为标记码，对账号里的摄像机进行推
送总管理，只要账号settag成功，账号里所有摄像机就可以收到推送消息。这种方式，优点是：实
现简单，只要使用的时候，以账号为id来settag一次就可以，不需要推送时，以""为id来settag。
账号里删除添加了摄像机瑞彩服务器会处理，客户端这边不需要任何处理。缺点是：必须使用服务
器平台管理摄像机，即用账号登陆服务器。
2）使用的类：JPushInterface [jpush的类]
3）函数：
/*
 *JPushInterface方法
 *此接口为jpush提供，不是瑞彩提供
 *每一个参数说明请查阅jpush官方文档
 *此方法的tags为账号登陆成功后，服务器返回的账号的"user_id"
 */
 public static void setAliasAndTags(Context context,
                                    String alias,
                                    Set<String> tags,
                                    TagAliasCallback callback)

4）步骤：
第一步：在jpush上创建应用后，把应用的AppKey和Master
Secret提供给瑞彩，瑞彩需要添加到服务器，才可测试使用。
第二步：客户端实现推送。

5）注意：1）setTags的tags不是账号，而是账号登陆成功后，服务器返回的账号的"user_id"

        2)通过这种方式实现推送，一个摄像机只能添加到一个账号，所以添加摄像机的时候需要
        注意，如果摄像机被注册
了也就是被其他账号添加了，需要先解绑unregister,再添加。［详细参考：摄像机的添加］

        3）通过这种方式实现推送的，摄像机必须要登陆SOSO服务器平台成功，所以添加摄像机
        的时候需要注意，添加摄像机成功后，需要relogin_to_sosocam，登陆服务器。［详细参
        考：摄像机的添加］
6）示例：
/*这里只介绍setTags部分，其他部分请参考jpush官方文档*/
```

```
this.setTag(getApplicationContext(),"user_id");//注册账号，需要推送

public static void setTag(Context context,String tag){
        Log.e("sosocam", "set tag: " + tag);
        m_context = context;
        m_tagSet.clear();
        if (! tag.equals(""))
            m_tagSet.add(tag);
        JPushInterface.setAliasAndTags(context, null, m_tagSet, mTagsCallback);
    }
private final static TagAliasCallback mTagsCallback = new TagAliasCallback() {

        @Override
        public void gotResult(int code, String alias, Set<String> tags) {
            switch (code) {
            case 0:
                Log.e("sosocam", "set tag result: 0");
                break;
            case 6002:
                Log.e("sosocam", "set tag result: 6002");
                JPushInterface.setAliasAndTags(m_context, null, m_tagSet,
                mTagsCallback);
                break;
            default:
                Log.e("sosocam", "set tag result: " + code);
                break;
            }
        }
    };

#13.2以摄像机为id来settag(未使用平台功能)
1）说明：以摄像机为id来settag是指：使摄像机为标记码，对摄像机进行单个推送管理，那一台摄像机settag成功，哪一台摄像机就可以收到推送消息。这种方式，优点是：只要本地管理摄像机就好，不需要用到服务器平台管理摄像机。缺点是：实现比较复杂，只要使用的时候，以摄像机为id来一个一个settag，需要推送时，以@""为id来settag。账号里删除添加了摄像机，客户端这边都需要重新把本地的摄像机一个一个settag。
2）使用类：Tools和JPushInterface [jpush的类]
3）函数：
/*
 *JPushInterface方法
 *此接口为jpush提供，不是瑞彩提供
 *每一个参数说明请查阅jpush官方文档
 *此方法的tags为摄像机id，id需要编码
 */
 public static void setAliasAndTags(Context context,
                                    String alias,
                                    Set<String> tags,
                                    TagAliasCallback callback)
/*
 *Tools方法
 *对摄像机id进行编码
 */
public static String  replaceExceptonChar(String s)

4）步骤：
第一步：在jpush上创建应用后，把应用的AppKey和Master Secret提供给瑞彩，瑞彩需要添加到服务器，才可测试使用。
第二步：客户端实现推送。

注意：1）setTags的tags是摄像机id，但是必须先把id通过replaceExceptonChar编码，才去settags。
     2）通过这种方式实现推送的推送，摄像机的推送地址必须要是正确的，具体的请联系瑞彩

5）示例：
/*这里只介绍setTags部分，其他部分请参考jpush官方文档*/
LinkedHashMap<String, IPCam> ipcam_list = null;
ipcam_list = IPCamMgr.get_cameras_list();
public static void setTag(Context context,LinkedHashMap<String, IPCam> ipcam_list){
    m_context = context;
    Log.e("sosocam", "context is " + context);
    m_tagSet.clear();
    for (IPCam ipcam : ipcam_list.values())
    {
```

```
            Log.e("sosocam", "set tag:id---------- --->>>>>:: " + ipcam.id());
            m_tagSet.add(Tools.replaceExceptonChar(ipcam.id()));
        }
        JPushInterface.setAliasAndTags(context, null, m_tagSet,
        mTagsCallback);//注册所有摄像机，需要推送
    }
    JPushInterface.stopPush(context);//不需要推送,停止推送

     private final static TagAliasCallback mTagsCallback = new TagAliasCallback() {

            @Override
            public void gotResult(int code, String alias, Set<String> tags) {
                switch (code) {
                case 0:
                    Log.e("sosocam", "set tag result: 0");
                    break;
                case 6002:
                    Log.e("sosocam", "" + JPushInterface.isPushStopped(m_context) + ","
                    + JPushInterface.getConnectionState(m_context));

                    Log.e("sosocam", "set tag result: 6002");
                    JPushInterface.setAliasAndTags(m_context, null, m_tagSet,
                    mTagsCallback);
                    break;
                default:
                    Log.e("sosocam", "set tag result: " + code);
                    break;
                }
            }

        };

    #14.登陆实现
    参考WebService的用户管理 <注册 / 忘记密码 / 登录 / 激活 / 账号信息>

    #15.报警信息
    说明：报警信息是指摄像机或连接到摄像机的外部设备发生报警，像服务器推送的报警信息（包括图片，报警类型）。这些信息保存在服务器，保留7天的报警信息，最多300张图片，100条报警信息（一条报警信息最多最多有5张图片）。

    #15.1 获取当前账号内所有摄像机的报警信息
    15.1 获取当前账号内所有摄像机的报警信息

    1）说明：获取当前账号内所有摄像机的报警信息是指：获取当前账号所有摄像机（不管有没有报警）的id,sosocam_id以及各个报警摄像机报警总数目和已查看的报警信息数目。
    2）使用类：WebService
    3）函数：
    /*
     *WebService方法:获取当前账号所有摄像机的报警信息
     *返回：ERROR类型error值，error == ERROR.NO_ERROR表示获取成功；
     */
    public static void get_cameras_alarm_list(get_cameras_alarm_list_listener listener);

    /*
     *WebService方法:get_cameras_alarm_list_listener回调
     *回调参数：
     *error:ERROR类型，error == ERROR.NO_ERROR表示获取成功
     *另一个是一个list,每一个元素都是CAMERA_ALARM_INFO对象，具体如下：
     *public String obj_id = "";//摄像机sosocam_id
     *public String id = ""; //摄像机id
     *public int unread = 0; //未读报警数目
     *public int total = 0; //报警总数目
     */
    public void on_get_cameras_alarm_list_result(ERROR error,
    ArrayList<CAMERA_ALARM_INFO> cameras_alarm_list);

    4）步骤：
    第一步：获取当前账号内所有摄像机的报警信息，且添加get_cameras_alarm_list_listener监听
    第二步：get_cameras_alarm_list_listener回调，取得信息

    4）注意：1）此操作必须以成功登陆账号（服务器）为前提

    5）示例：
```

```
/*第一步：获取当前账号内所有摄像机的报警信息 +
添加get_cameras_alarm_list_listener监听*/
WebService.get_cameras_alarm_list(CamAlarmListActivity.this);;//获取cams报警信息且添加
监听 如果不是返回ERROR.NO_ERROR 说明获取失败

/*获取到结果后触发回调函数，当回调函数中返回error == ERROR.NO_ERROR表示获取成功*/
public void on_get_cameras_alarm_list_result(ERROR error,
ArrayList<CAMERA_ALARM_INFO> svr_alarm_list) {
    if (error == ERROR.NO_ERROR) {
        for (CAMERA_ALARM_INFO local_alarm : alarm_list) {
            for (CAMERA_ALARM_INFO svr_alarm : svr_alarm_list) {
                if (local_alarm.id.equals(svr_alarm.id)) {
                    local_alarm.obj_id = svr_alarm.obj_id;
                    local_alarm.total = svr_alarm.total;
                    local_alarm.unread = svr_alarm.unread;
                }
            }
        }
    }
}

#15.2 获取单个cam报警信息

1）说明：获取单个cam报警信息是指：获取当前账号某一个摄像机的报警信息，包括每一条报警信
息的类型，id,图片数目，报警时间，预览图和查看状态。
2）使用类：WebService
3）函数
/*
 *WebService方法:获取摄像机详细报警信息
 *参数:
 *obj_id: 摄像机sosocam平台id
 *listener:get_alarm_detail_list_listener回调
 */
 public static void get_alarm_detail_list(String obj_id,
 get_alarm_detail_list_listener listener);

/*
 *WebService方法:get_alarm_detail_list_listener回调
 *回调参数:
 *error:ERROR类型，error == ERROR.NO_ERROR表示获取成功
 *另一个是一个list,每一个元素都是ALARM_DETAIL对象，具体如下:
 *public String aid;//摄像机报警id
 *public int alarm_type; //摄像机报警类型
 *public int image_numbers; //报警图片数量
 *public boolean read;//是否读过
 *public Date date;//报警时间
 *public String thumb_url;//预览的url地址
 */
public void on_get_alarm_detail_list_result(ERROR error, ArrayList<ALARM_DETAIL>
alarm_detail_list);

4）步骤:
第一步：获取当前账号内当前摄像机的报警信息，且添加get_alarm_detail_list_listener监听
第二步：get_alarm_detail_list_listener回调，取得信息

5）注意：a)以摄像机登陆成功为前提

6）示例：

/*第一步：获取当前账号内当前摄像机的报警信息+添加get_alarm_detail_list_listener监听*/
WebService.get_alarm_detail_list(activity.m_obj_id, activity);
/*第二步:get_alarm_detail_list_listener回调*/
public void on_get_alarm_detail_list_result(ERROR error, ArrayList<ALARM_DETAIL>
alarm_detail_list){
    if (error == ERROR.NO_ERROR){
        /*在此处把改摄像机的每一条包报警信息取出*/
    }
}

#15.3 获取某一条报警的图片信息
1）说明：获取某一条报警的图片信息是指，获取某摄像机的某一条报警信息的图片信息。包括报
警信息的每一张图片的url和时间，index和data。
2）使用类：WebService
```

3）函数：

```
/*
 *WebService方法:获取某一条报警的图片信息
 *参数：
 *aid：报警id
 *listener:get_alarm_image_info_list_listener监听
 */
public static ERROR get_alarm_image_info_list(String aid,
get_alarm_image_info_list_listener listener);

/*
 *WebService方法:get_alarm_image_info_list_listener回调
 *回调参数：
 *error:ERROR类型，error == ERROR.NO_ERROR表示获取成功
 *另一个是list，每一个元素都是ALARM_IMAGE_INFO对象，具体如下：
 *public Date date;//报警时间
 *public String url;//每张图片所代表的url地址
 */
public void on_get_alarm_image_info_list_result(ERROR error,
ArrayList<ALARM_IMAGE_INFO> alarm_image_info_list);

/*
 *WebService方法:取消获取单个摄像机所有报警图片，即停止get_alarm_images_list
 */
public static void cancel_get_alarm_image_info_list_tasks();

/*
 *WebService方法:获取单个摄像机的其中一条报警信息的图片
 *参数：
 *image_id:取intValue,取当前报警信息中的第几张照片，最多是5张，通过image_numbers可知
 *url:报警信息的图片URL
 *listener:get_alarm_image_listener
 */
public static void get_alarm_image(String image_id, String url,
get_alarm_image_listener listener)

/*
 *WebService方法:get_alarm_image_listener回调
 *回调参数：
 *image_id:当前照片的index值
 *data:获取到的图片data数据
 */
public void on_get_alarm_image_result(String image_id, byte [] data);

/*
 *WebService方法:取消获取单个摄像机某一条报警信息,即停止get_alarm_image
 */
public static void cancel_get_alarm_image_tasks()
```

4）步骤：

第一步：通过get_alarm_image_info_list获取图片的时间和url

第二步：使用get_alarm_image_info_list_listener回调来的url，通过get_alarm_image获取图片data数据

5）注意：

a)get_alarm_image_info_list需要的参数alarm_id是通过get_alarm_detail_list获取到的。

b)get_alarm_image需要的参数image_id,一定在通过get_alarm_detail_list获取到的image_numbers范围内

c)get_alarm_image_info_list和get_alarm_image都是异步的，所以不需要的时候，一定要cancel_get_alarm_image_info_list_tasks()和cancel_get_alarm_image_tasks()

6）示例：

```
/*第一步：通过get_alarm_images_list获取图片的时间和url*/
if (ERROR.NO_ERROR == WebService.get_alarm_image_info_list(m_aid, this))
    Log.e("SoDemo","download_alarm_image_list_now");
else
    Log.e("SoDemo","download_alarm_image_list_failed");

/*第二步：get_alarm_image_info_list_listener回调＋get_alarm_image*/
public void on_get_alarm_image_info_list_result(ERROR error,
ArrayList<ALARM_IMAGE_INFO> alarm_image_info_list) {
```

```
        if (ERROR.NO_ERROR == error) {
            for (ALARM_IMAGE_INFO info : alarm_image_info_list) {
                Log.e("SoDemo","处理每张图片")
                WebService.get_alarm_image("" + alarm_image_info_list.indexOf(info) +
                ".jpg", info.url, this);
                }
            Log.e("SoDemo","download_alarm_image_list_succeed")
        } else {
            Log.e("SoDemo","download_alarm_image_list_failed");
            }
        }

    /*第三步：WebServiceGetImageDelegate回调*/
    public void on_get_alarm_image_result(String image_id, byte [] data) {
        if (data != null) {
            Bitmap bitmap = data;
            Storage.write_jpg_file(data, Storage.get_alarm_folder(m_id, m_aid) + "/" +
            filename);
            }
    }

    /*第四步:不需要的时候要手动取消*/
    WebService.cancel_get_alarm_image_info_list_tasks();
    WebService.cancel_get_alarm_image_tasks();


    #16.分享和收藏
    #16.1开启分享
    16.1开启分享
    1）说明：开启分享是指：把摄像机的分享功能打开，摄像机的分享功能打开后，就会生成分享id,
    url QRcode，你可以通过这些收藏访问这台摄像机。
    2）使用的类：WebService，IPCam
    3）函数：
    /*
     *WebService方法：开启分享
     *参数：
     *obj_id:摄像机sosocam平台的id
     *listenter:share_camera_listener监听
     */
    public static ERROR share_camera(String obj_id, share_camera_listener listenter)

    /*
     *WebService方法：WebServiceShareCameraDelegate回调
     *参数：
     *result;//sosocam平台返回的分享结果ERROR值
     *share_id;//分享码的id
     *share_url;//分享码的url地址
     *share_qrcode;//分享码的二维码地址
     */
    public void on_share_camera_result(ERROR result, String share_id, String share_url,
    String share_qrcode);

    /*
     *IPCam方法：CGI设置参数
     *参数：
     *params:cgi参数，由reecam提供，开启分享是"save=1&reinit_guest=1&group2=1"
     *listener:IPCam.set_params_listener监听，可以监控cgi设置的结果
     */
    public ERROR set_params(String params, set_params_listener listener)

    /*
     *IPCam方法：IPCam.set_params_listener回调函数
     *返回参数：
     *ipcam：返回一个IPCam对象
     *ERROR类型，当返回是ERROR.NO_ERROR，表示设置成功
     */
    public void on_result(IPCam ipcam, ERROR error);

    4）步骤：
    第一步：开启分享，添加接口share_camera_listener。
    第二步：开启分享成果后，用cgi设置权限，且添加监听set_params_listener。
    第三步：set_params_listener回调
```

5）注意：a）开启分享音频也是用set_params，params为"save=1&reinit_guest=1&group2=9"
b）关闭分享音频也是用set_params，params为"save=1&reinit_guest=1&group2=1"
c)设置分享的密码也是用set_params，params为"save=1&reinit_guest=1&pwd2=%@"
d)这些设置必须是以摄像机连接成功为前提

6）示例：

```
/*第一步:开启分享，添加监听share_camera_listener*/
   if(ERROR.NO_ERROR != WebService.share_camera(ipcam_sosocam_id, this)){
        /*开启分享失败*/
    }

/*第二步：开启分享成功后，用cgi设置权限，且添加监听set_params_listener。*/
public void on_share_camera_result(ERROR result, String share_id,String share_url,
String share_qrcode) {
    if(result == ERROR.NO_ERROR){
        String params = "save=1&reinit_guest=1&group2=1";
        if(ERROR.NO_ERROR != m_ipcam.set_params(params, this)){
            /*开启分享失败*/
        }
    }else{
        /*开启分享失败*/
    }
}

/*第三步：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");
        }
    }
}
```

#16.2关闭分享

1)说明：关闭分享是指：关闭摄像机的分享功能，摄像机的分享功能一旦关闭，id,url QRcode就会变成无效，你不可以通过这些id,url QRcode来访问摄像机了。

2)使用的类：WebService

3)函数：

```
/*
 *WebService方法：关闭分享
 *参数：
 *obj_id:摄像机sosocam平台的id
 *listenter：监听disshare_camera_listener
 */
public static ERROR disshare_camera(String obj_id, disshare_camera_listener listenter)

/*
 *WebService方法：disshare_camera_listener回调
 *参数：
 *error:错误值，当错误值为ERROR.NO_ERROR时，说明开启分享成功
 */
public void on_disshare_camera_result(ERROR result);

/*
 *IPCam方法：CGI设置参数
 *参数：
 *params:cgi参数，由reecam提供，开启分享是"save=1&reinit_guest=1&group2=1"
 *listener:IPCam.set_params_listener监听，可以监控cgi设置的结果
 */
public ERROR set_params(String params, set_params_listener listener)

/*
 *IPCam方法：IPCam.set_params_listener回调函数
 *返回参数：
 *ipcam：返回一个IPCam对象
 *ERROR类型，当返回是ERROR.NO_ERROR，表示设置成功
 */
public void on_result(IPCam ipcam, ERROR error);
```

4）步骤
第一步：关闭分享，添加监听disshare_camera_listener。
第二步：关闭分享成果后，用cgi设置权限，且添加监听set_params_listener。
第三步：set_params_listener回调

5）注意：a)这些设置必须是以摄像机连接成功为前提

6）示例：

```
public void disshare_camera(){
    if (WebService.ERROR.NO_ERROR != WebService.disshare_camera(m_ipcam.sosocam_id,
    this){
        /*关闭分享失败*/
    }
}
public void on_disshare_camera_result(com.sosocam.webservice.WebService.ERROR
result) {
    if(result == com.sosocam.webservice.WebService.ERROR.NO_ERROR){
        if(ERROR.NO_ERROR != m_ipcam.set_params("save=1&reinit_guest=1&group2=" +
        m_group, this)){
            /*关闭分享失败*/
        }
    }else{
        /*关闭分享失败*/
    }

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");
        }
    }
}
```

#16.3分享状态获取
1）说明：分享状态获取，是指摄像机有没有开启分享，有没有开启音频分享，有没有设置分享密码。这些状态的获取是用过get_params来实现的
2）使用类：IPCam
3）函数：

```
/*
 *IPCam方法：CGI获取参数
 *参数：
 *params:cgi参数，由reecam提供
 *listener:get_params_listener监听，可以监控cgi获取的结果
 *返回：
 *返回ERROR类型，当返回ERROR.NO_ERROR，则表示函数执行成功
 */
public ERROR get_params(String params, get_params_listener listener);

/*
 *IPCam方法：get_params_listener回调
 *参数：
 *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；
 */
public void on_result(IPCam ipcam, ERROR error, JSONObject json);
```

4）步骤：
第一步：获取参数，且添加监听get_params_listener
第二步：根据get_params_listener结果判断状态

5）注意：a)这些设置是以摄像机连接成功为前提
b)摄像机分享信息状态的获取，params为"group2=&pwd2="

6）示例：

```
/*第一步：获取参数＋添加监听get_params_listener*/
-(void)get_share_info{
    if(ERROR.NO_ERROR != m_ipcam.get_params("group2=&pwd2=", this)){
        /*获取分享信息失败*/
```

```
    }
}
/*第二步：根据get_params_listener结果判断状态*/
public void on_result(IPCam ipcam, ERROR error, JSONObject json) {
        String pwd = null;
        if(error == ERROR.NO_ERROR){
            /*获取分享信息成功*/
            try {
                cur_auth = json.getInt("group2");
                pwd = json.getString("pwd2");
            } catch (JSONException e) {
                /*摄像机未连接*/
            }
            if(pwd != null && ! pwd.equals(""))
                /*分享密码：有密码*/
            if(0 >= cur_auth){
                cur_auth = 0;
                /*分享状态为：关闭*/
            }else{
                /*分享状态为：开启*/
                if(((cur_auth >> 3) & 0x01) == 1){
                    /*分享音频状态为：开*/
                }else{
                    /*分享音频状态为：关闭*/
                }
            }
        }else{
            /*获取分享信息失败*/
        }
}

#16.4添加收藏摄像机
1)说明：添加收藏摄像机是指添加别人分享的摄像机到你的账号（服务器）或者本地（离线使用）。只要摄像机被分享了，就可以被多个人（账号）添加。但是一旦摄像机的分享功能被关闭了，你就无法获取到分享摄像机信息。
2）使用的类：Storage，WebService
3）函数：
/*
 *Storage方法：添加收藏到本地
 /*参数：
 *列表的每一个元素是一个COLLECTION_INFO类型的对象，具体如下：
 *String collection_id   分享id
 *boolean valid    当前camera的收藏是否有效，
 *                 true：有效可继续观看
 *                 false：当前收藏无效，即向你分享此camera的用户，已设置为不再分享
 *String alias     昵称
 *String user      登录名
 *String pwd       登录密码
 *String cover_url 预览图url
 */
public static void add_collection(COLLECTION_INFO collection_info);

/*
 *WebService方法:获取收藏摄像机信息
 *参数：
 *share_id：分享id
 *listener:get_collection_listener监听
 */
public static void get_collection(String share_id, get_collection_listener listenter)

/*
 *WebService方法:get_collection_listener回调
 *参数：
 *error:错误值，当错误值为ERROR.NO_ERROR时，说明获取收藏成功
 *alias:摄像机别名
 *cover_url：摄像机预览图
 */
public void on_get_collection_result(ERROR result, String alias, String cover_url);

/*
 *WebService方法:账号添加分享的摄像机：即添加到服务器和本地
 *参数：
 *share_id:摄像机分享id号
```

```
 *listener: 监听add_collection_listener
 */
public static ERROR add_collection(String share_id, add_collection_listener listenter)

/*
 *WebService方法:add_collection_listener回调
 *参数:
 *error:错误值，当错误值为ERROR.NO_ERROR时，说明开启添加收藏成功
 *alias:摄像机别名
 *cover_url: 摄像机预览图
 */
public void on_add_collection_result(ERROR result, String alias, String cover_url);
```

4)步骤:
账号添加（需要添加到服务器）:
第一步：添加分享的摄像机到服务器上，且添加监听add_collection_listener，监控添加结果
第二步：add_collection_listener监听回调，如服务器添加成功就直接添加本地

离线使用（只需要添加都本地）:
第一步：获取收藏摄像机的信息，且添加监听get_collection_listener。
第二步：get_collection_listener监听回调，如果获取成功就直接添加到本地。

5）注意：a)添加之前，必须确保了登陆了账号（或者update_user）且set_current_user了。
b)分享的摄像机可以被多个账号添加，没有限制
6）示例:
账号添加（需要添加到服务器）

```
/*第一步：添加分享的摄像机到本地＋添加监听add_collection_listener*/
-(void)add_collection_to_service()
{
    if (ERROR.NO_ERROR != WebService.add_collection(m_Share_id, this)) {
        /*添加失败*/
    }
}
/*第二步：add_collection_listener监听回调*/
public void on_add_collection_result(ERROR result, String alias,String cover_url) {
        if(result == ERROR.NO_ERROR){
            /*添加到服务器成功，然后添加到本地*/
            COLLECTION_INFO collection = new COLLECTION_INFO();
            collection.collection_id = m_Share_id;
            collection.alias = alias;
            collection.cover_url = cover_url;
            Storage.add_collection(collection);
        } else if(result == ERROR.INVALID_SHARE_ID){
             /*分享id无效*/
        }else{
            /*添加到服务器失败*/
        }
}
```

离线使用（只需要添加都本地）

```
/*第一步：获取收藏摄像机的信息＋添加监听get_collection_listener*/
public void add_collection_to_service()
{
    if (ERROR.NO_ERROR != WebService.get_collection(m_Share_id, this)) {
        /*获取失败*/
    }
}

/*第二步：get_collection_listener回调*/
public void on_get_collection_result(ERROR result, String alias,String cover_url) {
        if(result == ERROR.NO_ERROR){
            /*获取收藏摄像机信息成功，然后添加到本地*/
            COLLECTION_INFO collection = new COLLECTION_INFO();
            collection.collection_id = m_Share_id;
            collection.alias = alias;
            collection.cover_url = cover_url;
            /*添加到本地*/
            Storage.add_collection(collection);
        } else if(result == ERROR.INVALID_SHARE_ID){
            /*分享id无效*/
        } else{
            /*获取收藏摄像机信息失败*/
        }
```

```
    }


#16.5连接收藏摄像机（一次性连接）
1）说明：连接收藏摄像机是指：把收藏的摄像机连接，观看视频。在此采用一次性连接。
2）使用的类：IPCamMgr
3）函数：
/*
 *IPCamMgr方法：获取一次性摄像机
 *返回：
 *返回一个IPCam对象
 */
public static IPCam get_disposable_camera()//获取一次性摄像机

/*
 * IPCamMgr方法：设定一个一次性摄像机且自动连接
 *参数：
 *id:摄像机id
 *user：摄像机登录名
 *pwd:摄像机登陆密码
 *https:摄像机安全传输值
 *返回：
 *返回一个BOOL，true表示设置成功，false表示设置失败
 */
public static boolean set_disposable_camera(String id, String user, String pwd,
boolean https);//设定一次性摄像机

/*IPCamMgr方法：清除一次性摄像机*/
public static void clear_disposable_camera();//清除一次性摄像机

/*IPCamMgr方法：更新一次性摄像机密码到IPCamMgr
 *返回一个BOOL，true表示更新成功，false表示更新失败
 */
public static boolean update_disposable_camera_pwd(String pwd);

/*IPCamMgr方法：重连一次性摄像机
 *返回一个BOOL，true表示执行函数成功，false表示执行函数失败
 */
public static boolean reset_disposable_camera();//重连一次性摄像机

5）注意点：a)连接就一个步骤，set_disposable_camera，摄像机的其它操作，比如看视频就和其它摄像机一样

        b)如果分享无效连接是不成功的。所以连接之前最好get_collection根据回调判断摄像机是否是有效分享
        c)不连接了一定要clear_disposable_camera

6）示例：
    IPCamMgr.set_disposable_camera(id, user, pwd, https);

#16.6删除收藏摄像机
1)说明：删除收藏摄像机是只删除添加的收藏摄像机从服务器和本地。
2）使用的类：Storage，WebService
3）函数：
/*
 *Storage方法：删除当前账号本地收藏的摄像机
 *参数：
 *share_id:分享id
 */
public static void remove_collection(String collection_id);

/*
 *WebService方法：删除当前账号服务器上收藏的摄像机
 *参数：
 *share_id:分享id
 *delegate:监听remove_collection_listener
 *返回：
 *ERROR：当返回值为ERROR.NO_ERROR说明函数执行成功
 */
 public static ERROR remove_collection(String share_id, remove_collection_listener
 listenter)
```

```
/*
 *WebService方法：remove_collection_listener回调函数
 *参数：
 *error:错误值，当错误值为ERROR.NO_ERROR时，说明删除成功
 */
public void on_remove_collection_result(ERROR result, String share_id);
```

4）步骤：
第一步：删除服务器上的收藏摄像机，且添加监听remove_collection_listener，监控删除结果
第二步：remove_collection_listener回调函数，如服务器删除成功就直接删除本地

5）注意：a)未添加到服务器，即离线使用，直接删除本地remove_collection即可。
b)这些方法的执行，前提是添加收藏的时候按照reecam的方式添加到本地添加到服务器。

6）示例：

```
/*第一步：删除服务器上的收藏摄像机＋添加监听remove_collection_listener*/
public void delete_server()
{
    if (ERROR.NO_ERROR != WebService.remove_collection(m_Share_id, this)) {
        /*删除失败*/
    }
}
/*第二步：remove_collection_listener回调函数，如服务器删除成功就直接删除本地*/
public void on_remove_collection_result(ERROR result, String share_id) {
        if(result == ERROR.NO_ERROR){
            /*服务器删除成功，本地进行删除*/
            Storage.remove_collection(m_Share_id);
        } else {
             /*删除失败*/
        }
    }
```

#16.7更新收藏摄像机本地信息
1）说明：更新收藏摄像机本地信息是指：更新收藏摄像机的本地存储的信息，这些信息只保存到本地不会同步到服务器。
2）使用的类：Storage
3）函数：

```
/*Storage方法
 *更新本地收藏的摄像机是否依旧合法，可观看视频
 *collection_id:要被更新的收藏摄像机分享id
 *valid:收藏摄像机的valid本地更新为valid
 */
public static void update_collection_valid(String collection_id, boolean valid);

/*Storage方法
 *更新本地收藏摄像机的访问密码
 *collection_id:要被更新的收藏摄像机分享id
 *pwd:收藏摄像机的pwd本地更新为pwd
 */

public static void update_collection_pwd(String collection_id, String pwd);
```

4）注意：a）分享摄像机是否还有效，是根据获取分享摄像机的回调结果中的error来判断的，详细参考on_get_collection_result的用法
b)这些方法都是基于瑞彩提供的存储方式存储的。

5）示例：

```
    Storage.init(this);
    Storage.update_collection_pwd(m_collection.collection_id, pwd);
```

#16.8获取服务器和本地收藏的摄像机
1)说明：获取服务器和本地收藏的摄像机是指：把添加到账号（服务器）或本地的分享出的摄像机获取出来显示操作。
2)使用的类：WebService，Storage
3)函数：

```
/*
 *WebService方法：从服务器获取账号收藏的所有摄像机
 *返回:返回一个COLLECTION_INFO对象，具体如下：
 *share_id：分享id
 *alias：别名
 *cover_url:预览图的url
 */
public static ArrayList<SERVICE_COLLECTION_INFO> current_collections = new
```

```
    ArrayList<SERVICE_COLLECTION_INFO>();
2633 current_collections = WebService.collections;//WebService服务器收藏列表
2634
2635 /*
2636  *Storage方法：从本地获取账号收藏的所有摄像机
2637  *返回一个列表，列表的每一个元素是一个COLLECTION_INFO类型的对象，具体如下：
2638  *
2639  *String collection_id   分享id
2640  *boolean valid    当前camera的收藏是否有效，
2641  *                 true：有效可继续观看
2642  *                 false：当前收藏无效，即向你分享此camera的用户，已设置为不再分享
2643  *String alias     昵称
2644  *String user      登录名
2645  *String pwd       登录密码
2646  *String cover_url 预览图url
2647  */
2648 public static ArrayList<COLLECTION_INFO> get_collections();//获取Storage本地收藏列表
2649
2650 /*
2651  *Storage方法：比较本地和服务器收藏的摄像机
2652  *参数：
2653  *SERVICE_COLLECTION_INFO:服务器收藏的摄像机列表
2654  *返回：返回一个BOOL参数
2655  *当BOOL值为true时，表示本地和服务器的收藏摄像机已进行同步
2656  *当BOOL值为false时，表示本地和服务器的收藏摄像机未做同步操作
2657  */
2658 public static boolean merge_collections(ArrayList<SERVICE_COLLECTION_INFO>
    server_collections_list);
2659
2660 4）注意：a）为提高程序性能，一般程序初始化就同步本地和服务器摄像机merge_collections_li
    st。这样子，后面需要获取收藏摄像机，直接从本地Storage中获取。
2661         b）当然不需要使用服务器，只收藏在本地，用到的时候直接从本地Storage中获取。
2662         c)这些方法都是基于瑞彩提供的存储方式存储的。
2663
2664 5）示例
2665         Storage.init(this);
2666         Storage.merge_collections(WebService.collections);
2667
2668 #17.TF卡相关操作
2669 说明：TF卡相关操作包括：tf卡操作和tf卡文件操作
2670
2671 #17.1 tf卡操作
2672 说明：tf卡操作是指:弹出tf卡，格式化tf卡，获取tf卡的容量和状态
2673 #17.1.1 弹出tf卡
2674 1）说明：弹出tf卡是指：把tf卡弹出磁盘，弹出之后，摄像机检测不到tf卡。
2675 2）使用类：IPCam
2676 3)函数：
2677 /*
2678  *IPCam方法：弹出tf卡
2679  *参数：
2680  *listener:IPCam.unplug_tf_listener
2681  *返回：返回ERROR类型，当返回ERROR.NO_ERROR，表示没有错误，调用成功
2682  */
2683 public ERROR unplug_tf(unplug_tf_listener listener);
2684
2685 /*
2686  *IPCam方法：unplug_tf_listener回调
2687  *回调参数：
2688  *ipcam：返回一个IPCam对象
2689  *error:ERROR类型，当返回是ERROR.NO_ERROR，表示弹出成功
2690  */
2691 public void on_result(IPCam ipcam, ERROR error);
2692
2693 4)示例：
2694 /*第一步：弹出tf卡＋添加监听IPCam.unplug_tf_listener */
2695 m_ipcam.unplug_tf(this);
2696 m_state = STATE.EJECT;
2697
2698 /*第二步：监听IPCam.unplug_tf_listener回调*/
2699 public void on_result(IPCam ipcam, ERROR error) {
2700     if(m_state == STATE.EJECT){
2701         if (error == ERROR.NO_ERROR) {
2702             Log.e("SoDemo","ejecting_tf_succeed");
```

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

#17.1.2格式化tf卡
1）说明：格式化tf卡是指把tf卡本摄像机的所有文件都删除
2）使用类：IPCam
3)函数：

/*
 *IPCam方法：格式化tf卡
 *参数：
 *listener:format_tf_listener
 *返回：返回ERROR类型，当返回ERROR.NO_ERROR，表示没有错误，调用成功
 */
public ERROR format_tf(format_tf_listener listener);

/*
 *IPCam方法：format_tf_listener回调
 *回调参数：
 *ipcam：返回一个IPCam对象
 *error:ERROR类型，当返回是ERROR.NO_ERROR，表示弹出成功
 */
public void on_result(IPCam ipcam, ERROR error);

4）示例：
/*第一步：格式化tf卡＋添加监听format_tf_listener
 */
m_ipcam.format_tf(this);
m_state = STATE.FORMAT;
/*第二步：监听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");
        }
    }
}

#17.1.3tf卡状态和容量
1）说明：tf卡状态和容量是指：查看tf状态和容量状态
2）使用类：IPCam
3)函数：
/*
 *IPCam方法：获取tf卡当前状态
 *返回：返回TF_STATUS类型，详细如下：
 *TF_STATUS.NONE:没有tf卡
 *TF_STATUS.READY:tf卡录像中
 *TF_STATUS.ERROR:tf卡错误
 *TF_STATUS.FULL:tf卡已满
 *TF_STATUS.BUSY:tf卡检测中
 */
public TF_STATUS tf_status()
/*
 *IPCam方法：获取tf当前剩余容量百分比
 *返回:返回int,返回0-100
 */
public int tf_free();

/*
 *IPCam方法：获取tf总容量
 *返回:返回float，单位MB
 */
public int get_disk_size();

4）示例：
    TF_STATUS tf = ipcam.tf_status();//状态
    int sd_disk = ipcam.tf_free(); //容量
```

```
        float sd_capacity_total = ipcam.get_disk_size()/1024; //总容量
        float sd_capacity_usered = sd_capacity_totly*sd_disk/100; //剩余容量
```

#17.2tf卡文件列表 <下载所有文件 / 一天 / 一小时 / 一刻钟（预览图）>

说明：获取tf卡文件是指获取tf卡内当前摄像机的，获取到刻钟。获取到每一刻钟录像文件的简单信息，包括时间，是否有文件，是否有报警，预览图等。录像文件是10s一个片段，我们把它组成15min一个文件来显示播放，即一个文件有90个小片段组成。

#17.2.1 下载tf卡文件

17.2.1 下载tf卡文件（到刻钟）

1)说明：下载tf卡所有文件是指：从摄像机里面下载tf卡内当前摄像机所属的所有录像文件，到刻钟，获取简单的文件信息，需要网络和时间。对摄像机文件操作必须先load_tf_records。

2)使用类：IPCam

3)函数：

```
/*
 *IPCam方法：下载所有的录像文件（到刻钟）
 *参数：
 *listener:load_tf_records_listener
 *返回：
 *返回ERROR类型，当返回的ERROR为！ERROR.NO_ERROR表示下载失败
 */

 public ERROR load_tf_records(load_tf_records_listener listener)
/*
 *IPCam方法：load_tf_records_listener回调
 *参数函数：
 *ipcam:返回当前摄像机
 *error:错误值，返回ERROR类型，当返回的ERROR.NO_ERROR表示获取成功。
 */
public void on_result(IPCam ipcam, ERROR error);

/*
 *IPCam方法：停止load_tf_records
 */
 public void clear_tf_records()
```

4)注意：a)进行tf卡文件操作，必须先成功load_tf_records,就像使用使用libSoSoCamSDK之前必须成功初始化SDK。

b)clear_tf_records与cancel_tf_record_tasks不同，它除了取消tf卡所有的操作，同时把存储录像文件信息的数组释放了。所以一旦clear_tf_records，必须要先load_tf_records才做tf文件其他操作

c)当完全不需要tf卡文件操作时才clear_tf_records，而不是load_tf_records完成后就可clear_tf_records。所以clear_tf_records前肯定需要cancel_tf_record_tasks，但是cancel_tf_record_tasks不一定会去clear_tf_records。

#17.2.2 获取一天文件

1)说明：获取一天文件是指：获取某一天的录像文件，只是简单的获取某一个信息即可，信息在load_tf_records的时候已经下载下来，所以不需要占用时间和网络什么的。主要包含当天是否有录像文件，当天是否有报警等。

2)使用类：IPCam

3)函数：

```
/*
 *IPCam方法：获取每天的录像文件
 *参数：
 *day：获取哪一天的录像文件，范围[0，7)，总共7天，只保留最近7天的录像文件
 *
 *返回：返回一个TF_RECORD_DAY_INFO对象，具体如下：
 public class TF_RECORD_DAY_INFO {
        public boolean valid; 当天是否有录像文件，true:有录像文件，false:无录像文件
        public boolean alarm; 当天是否有报警，true:有报警，false:无报警
        public int week;  星期几，1至7分别代表星期天到星期五
        public boolean today;   今天，true:是今天，false:不是今天
        public boolean yesterday;   昨天，true:是昨天，false:不是昨天
        public boolean dby;    前天，true:是前天，false:不是前天
        public int valid_hours;   当前，此小时有没有录像文件
    }
 *
 */
public TF_RECORD_DAY_INFO get_tf_record_day_info(int day)
```

4)注意：必须要load_tf_records成功后才能get_tf_record_day_info

#17.2.3 获取一小时文件

1)说明：获取一小时文件是指获取某一天某一个小时的的录像文件，只是简单的获取某一个信息即可，信息在load_tf_records的时候已经下载下来，所以不需要占用时间和网络什么的。

2)使用类：IPCam

3)函数：

```
/*
 *IPCam方法：获取第几天第几个小时的录像文件
 *参数：
 *day:第几天，共7天，范围［0，7）
 *hour：第几个小时，共24个小时，返回［0，24）
 *返回：
 *返回一个BOOL，true表示执行成功，false表示执行失败
 */

public boolean get_tf_record_hour_valid(int day, int hour)
```

4）注意：a)必须要load_tf_records成功后才能get_tf_record_day_info

b)一般是get_tf_record_day_info成功后，判断valid有文件才去get_tf_record_hour_valid

#17.2.4 获取一刻钟录像信息

1)说明：获取一刻钟录像信息是指:获取某一天某一小时某一刻钟的录像文件，只是简单的获取某一个信息即可，信息在load_tf_records的时候已经下载下来，所以不需要占用时间和网络什么的。此处只是简单的获取当前刻钟是否有录像文件，是否有报警。

2)使用的类：IPCam

3)函数：

```
/*
 *
 *IPCam方法：获取第几天第几个小时第几刻钟的录像文件
 *参数：
 *day:第几天，共7天，范围［0，7）
 *hour：第几个小时，共24个小时，范围[0，24)
 *quarter:第几刻钟，共4刻钟，范围[0,4)
 *
 *返回：返回一个TF_RECORD_QUARTER_INFO对象，具体如下：
 public class TF_RECORD_QUARTER_INFO {
        public boolean valid;  当前一刻钟是否有录像文件，true:有文件，false:没有文件
        public boolean alarm;  当前一刻钟是否有报警，true:有报警，false:没有报警
    }
 */
public TF_RECORD_QUARTER_INFO get_tf_record_quarter_info(int day, int hour, int quarter)
```

4）注意：必须是get_tf_record_hour_valid成功后，再去获取

#17.2.5 获取一刻钟的录像预览图

1）说明：获取一刻钟的录像预览图是指获取某一小时内某一刻钟录像文件的预览图，原则是取最靠近这一刻钟的预览图

2）使用类：IPCam

3）函数：

```
/*
 *IPCam方法：获取第几天第几个小时第几刻钟的录像录像预览图
 *参数：
 *day:第几天，共7天，范围［0，7）
 *hour：第几个小时，共24个小时，范围[0，24)
 *valid_hour_index:有录像文件小时索引
 *quarter:第几刻钟，共4刻钟，范围[0,4)
 *listener:get_tf_record_quarter_thumb_listener
 */

public ERROR get_tf_record_quarter_thumb(int day, int hour, int valid_hour_index, int quarter, get_tf_record_quarter_thumb_listener listener)

/*
 *IPCam方法：get_tf_record_quarter_thumb_listener接口
 *返回参数：
 *ipcam:返回的IPCam对象
 *day:第几天，共7天，范围［0，7）
 *hour：第几个小时，共24个小时，范围[0，24)
 *valid_hour_index:有录像文件小时索引
 *quarter:第几刻钟，共4刻钟，范围[0,4)
```

```
 *thumb: 预览图数据
 */
public void on_result(IPCam ipcam, int day, int hour, int valid_hour_index, int quarter, byte[] thumb);
```

4)注意：a)必须是get_tf_record_quarter_info成功后再去获取。

17.2.6 示例

```
/*第一步：下载tf卡文件*/
public void load_record(){
    if (ERROR.NO_ERROR != m_ipcam.load_tf_records(this)) {
        //下载tf卡文件失败
    }
}

/*第二步：load_record回调成功后，先后获取天/时/刻录像文件信息*/
public void on_result(IPCam ipcam, ERROR error) {
{
   if (error != ERROR.NO_ERROR) {
        //下载tf卡文件失败
        return;
    }
    //load_record成功后开始获取7天的录像文件
    TF_RECORD_DAY_INFO day_info;
    TF_RECORD_QUARTER_INFO quarter_info;
    m_record_info = new RECORD_DAY_INFO[7];
    int d, h, q, vh;
    m_current_day = 0;
    for (d = 0;d < 7; d++) {
        /*
         *2.1 分别获取7天的录像文件
         */
        day_info = m_ipcam.get_tf_record_day_info(d);
        if (day_info.valid) {
            //当天有录像文件
        } else {
            //当天无录像文件
        }
        if (day_info.today) {
            //第d天是今天
        } else if (day_info.yesterday) {
            //第d天是昨天
        }else if(day_info.dby) {
            //第d天是前天
        }else {
            if (day_info.week == 1) {
                //第d天是星期天
            } else if (day_info.week ==  2) {
                //第d天是周一
            } if (day_info.week ==  3) {
                //第d天是周二
            } if (day_info.week ==  4) {
                //第d天是周三
            } if (day_info.week ==  5) {
                //第d天是周四
            } if (day_info.week ==  6) {
                //第d天是周五
            } if (day_info.week ==  7) {
                //第d天是周六
            }
        }
        //获取当天有录像文件开始获取当天24小时录像文件
        if (day_info.valid) {
            m_current_day = d;
            m_record_info[d].hours = new
            RECORD_HOUR_INFO[m_record_info[d].valid_hours];
            for (h = 0;h < 24;h ++) {
                /*
                 *2.2 分别获取每一天24小时的录像文件
                 */
                if (m_ipcam.get_tf_record_hour_valid(d, h)) {
                    m_record_info[d].hours[vh] = new RECORD_HOUR_INFO();
```

```
                    m_record_info[d].hours[vh].hour = h;
                    m_record_info[d].hours[vh].quarters = new RECORD_QUARTER_INFO[4];
                    for (q = 0;q < 4; q ++) {
                        /*
                         *2.3 分别获去一刻钟录像文件信息
                         */
                        m_record_info[d].hours[vh].quarters[q] = new
                        RECORD_QUARTER_INFO();
                        quarter_info = m_ipcam.get_tf_record_quarter_info(d, h, q);
                        if (quarter_info.valid){
                            //当前刻钟有录像文件
                        }
                        else{
                            //当前刻钟无录像文件
                        }
                        if (quarter_info.alarm){
                            //当前刻钟有报警
                        }
                        else{
                            //当前刻钟无报警
                        }
                    }
                }
                vh ++;
            }
        }
    }
}
/*第三步：获取刻成功后，获取每小时每刻的预览图*/

public View getView(int position, View convertView, ViewGroup parent) {
    if (convertView == null) {
        convertView =
        m_inflater.inflate(getResources().getIdentifier("gridhour_item", "layout",
        m_package_name) , null);
    }
    ImageView image;
    ImageView alarm;
    ProgressBar progress;
    int hour;
    RECORD_QUARTER_INFO info;
    int q;
    TF_RECORD_QUARTER_TIME t;
    for (q = 0;q < 4;q ++) {
        info = m_record_info[m_current_day].hours[position].quarters[q];
        if (info.load_state == LOAD_STATE.UNLOAD) {
            /*分别获去每一刻钟的预览图*/
            if (ERROR.NO_ERROR != m_ipcam.get_tf_record_quarter_thumb(m_current_day,
            hour, position, q, this)) {
                progress.setVisibility(INVISIBLE);
                image.setVisibility(VISIBLE);
            } else {
                info.load_state = LOAD_STATE.LOADING;
            }
        }
    }
    return convertView;
}

/*第四步：获取预览图回调 */
public void on_result(IPCam ipcam, int day, int hour, int valid_hour_index, int
quarter, byte [] thumb) {
    if (m_loadrecord_state != LOAD_STATE.LOADED) {
        return;
    }
    info.load_state = LOAD_STATE.LOADED;
    info.thumb = thumb;
    if (info.thumb != null) {
        image.setImageBitmap(BitmapFactory.decodeByteArray(info.thumb, 0,
        info.thumb.length));
    }
}
```

#17.3播放tf卡文件<三种播放tf卡文件方式 / 上 / 下一个文件播放>

17.3.1 三种播放tf卡文件方法

1）说明：播放tf卡录像文件是指播放当前摄像机存储在tf卡内的录像文件。有三种方式可以获取到录像文件。

2）使用类：IPCam

3）函数：

```
/*
 *IPCam方法：播放录像文件，从某一日日期开始
 *播放距离此日期最近的录像文件
 *SOSOCAM在播放报警录像的时候有用到
 *返回：
 *返回一个int32_t类型，就是播放的录像文件id
 */
public static int get_tf_record_play_id(Date date)

/*
 *IPCam方法：播放录像文件，从某一刻钟开始
 *SOSOCAM在点击某一刻钟直接播放时用到
 *参数：
 *day:播放哪一天，范围［0，7）
 *hour:播放哪一小时，范围［0，24）
 *quarter:播放哪一个刻钟，范围［0，4）
 *返回：
 *返回一个int32_t类型，就是播放的录像文件id
 */

public int get_tf_record_play_id(int day, int hour, int quarter)

/*
 *IPCam方法：播放录像文件，从某一刻钟某10s的开始
 *SOSOCAM在播放的时候拖动进度条的时候用到
 *参数：
 *day:播放哪一天，范围［0，7）
 *hour:播放哪一小时，范围［0，24）
 *quarter:播放哪一个刻钟，范围［0，4）
 *no:播放刻钟的哪一个片段，范围［0，90）
 *返回：
 *返回一个int32_t类型，就是播放的录像文件id
 */
public int get_tf_record_play_id(int day, int hour, int quarter, int no)

/*
 *IPCam方法：播放tf卡录像文件
 *
 */
public ERROR play_tf_record(int record_id)

/*
 *IPCam方法：IPCam_Listener接口之tf录像事件监控
 *当有新录像文件生成或录像文件出错时会掉用此方法
 *SOSOCAM在播放tf卡录像文件是使用了此代理
 *使用方法：参考IPCam基础监听
 *回调参数：
 *ipcam：IPCam对象
 *new_record:是否是新生成的录像文件,true:是，false:否
 *record_id：录像文件开始id
 *error:是否有错误，true:是，false:否
 *
 */
 public void on_tf_record_event(IPCam ipcam, boolean new_record, int record_id,
 boolean error);
```

4）注意：a）这些操作必须以摄像机连接成功为前提

b)播放完之后会自动切换播放下一个刻钟的的录像文件

c)只支持soso固件

5）示例

```
    IPCam.get_tf_record_play_id(date);//从某个时间最近的时间开始播放
    m_ipcam.play_tf_record(get_tf_record_play_id(0, 2, 2));//从某一刻开始播放
    play_tf_record(m_ipcam.get_tf_record_play_id(1, 6, 3,
    65));//从某一刻钟某10s的开始播放
```

17.3.2 tf卡文件播放停止 / 暂停 / 续播 / TF卡录像播放状态

1）说明：tf卡文件播放停止 / 暂停 / 续播 / TF卡录像播放状态分别指：停止tf卡录像播放，暂停tf卡录像播放，暂停后继续播放，以及随时获取TF卡录像播放状态。
2）使用类：IPCam
3）函数

```
/*
 *IPCam方法：tf卡录像播放状态
 *返回：返回TF_RECORD_STATUS，详细如下
 */
public TF_RECORD_STATUS tf_record_status() //tf播放状态
public enum TF_RECORD_STATUS {
        STOPPED,   //停止
        REQUESTING, //请求
        PLAYING, //播放
        BUFFING,  //缓冲
        PAUSING;  //暂停
    }
/*
 *IPCam方法：IPCam_Listener监听之tf播放状态监控
 *使用方法：参考IPCam
 */

public void on_tf_record_status_changed(IPCam ipcam);

/*
 *IPCam方法：停止播放tf卡录像文件
 */
public void stop_tf_record();//停止播放

/*
 *IPCam方法：暂停播放tf卡录像文件
 *返回：
 *返回IPCAM_ERROR类型，当返回IPCAM_ERROR_NO_ERROR时，表示暂停成功
 */
public ERROR pause_tf_record();//暂停

/*
 *IPCam方法：继续播放tf卡录像文件
 *返回：
 *返回ERROR类型，当返回ERROR.NO_ERROR时，表示继续播放成功
 */
public ERROR continue_tf_record(); //继续播放
```

4）注意：a）这些操作必须以摄像机连接成功为前提
        b)只支持SOSO固件

5）示例

```
    TF_RECORD_STATUS status = m_ipcam.tf_record_status(); //状态
    m_ipcam.continue_tf_record();//继续播放
    m_ipcam.pause_tf_record(); //暂停
    m_ipcam.stop_tf_record();//停止
```

17.3.3 上 / 下一个tf卡录像文件播放

1)说明：上 / 下一个tf卡录像文件播放是指：播放当前录像文件的上一个tf卡录像文件和下一个tf卡录像文件
2）使用类：IPCam
3）函数

```
/*
 *IPCam方法：播放下一个文件
 *参数：
 *day:当前天，范围［0，7）
 *hour:当前小时，范围［0，24）
 *quarter:当前刻钟，范围［0，4）
 *
 *返回：返回下一个TF_RECORD_QUARTER_TIME对象，具体如下：
 public class TF_RECORD_QUARTER_TIME {
        public int day;  下一个播放录像天，范围［0，7）
        public int hour;  下一个播放录像小时，范围［0，24）
        public int quarter; 下一个播放录像刻钟，范围［0，7）
```

```
3181        }
3182     */
3183    public TF_RECORD_QUARTER_TIME get_next_tf_record_quarter_time(int day, int hour, int
        quarter)
3184
3185    /*
3186     *IPCam方法：播放上一个文件
3187     *参数：
3188     *day:当前天，范围［0，7）
3189     *hour:当前小时，范围［0，24）
3190     *quarter:当前刻钟，范围［0，4）
3191     *
3192     *返回：返回上一个TF_RECORD_QUARTER_TIME对象，具体如下：
3193     *day:上一个播放录像天，范围［0，7）
3194     *hour:上一个播放录像小时，范围［0，24）
3195     *quarter:上一个播放录像刻钟，范围［0，7）
3196     */
3197    public TF_RECORD_QUARTER_TIME get_previous_tf_record_quarter_time(int day, int hour,
        int quarter)
3198
3199    4）注意：a）这些操作必须以摄像机连接成功为前提
3200            b)只支持SOSO固件
3201            c)播放之前，如果当前录像文件正在播放，需要先stop_tf_record
3202
3203    5）示例
3204    /*播放上一个tf卡录像文件*/
3205    tf_play_prev.setOnClickListener(new View.OnClickListener() {
3206        @Override
3207        public void onClick(View arg0) {
3208            TF_RECORD_QUARTER_TIME t =
            m_ipcam.get_previous_tf_record_quarter_time(m_day, m_hour, m_quarter);
            //根据当前时间获取上一个文件的时间
3209            if (t == null)
3210                return;
3211            m_day = t.day;
3212            m_hour = t.hour;
3213            m_quarter = t.quarter;
3214            m_ipcam.stop_tf_record();//先停止播放当前文件
3215            m_ipcam.play_tf_record(get_tf_record_play_id(m_day, m_hour, m_quarter));
            //播放上一个文件
3216        }
3217    });
3218
3219    /*播放下一个tf卡录像文件*/
3220    tf_play_next.setOnClickListener(new View.OnClickListener() {
3221
3222        @Override
3223        public void onClick(View arg0) {
3224            TF_RECORD_QUARTER_TIME t = m_ipcam.get_next_tf_record_quarter_time(m_day,
            m_hour, m_quarter);
3225            if (t == null)
3226                return;
3227            m_day = t.day;
3228            m_hour = t.hour;
3229            m_quarter = t.quarter;
3230            m_ipcam.stop_tf_record();//先停止播放当前文件
3231            m_ipcam.play_tf_record(get_tf_record_play_id(m_day, m_hour, m_quarter));
            //播放上一个文件
3232        }
3233    )};
3234
3235    17.3.4下载tf卡录像文件
3236    1)说明：下载tf卡录像文件是指把当前摄像机存储在tf的录像文件下载到app相册，tf录像文件是
        以每10s一个文件.bin
        格式存储到tf卡的，且以刻钟为节点，每个刻钟90个文件。下载tf卡录像文件，实际上是一遍播放
        .bin
        文件，一遍录制成.mov格式文件,根据设置的record_performance_speed数值，播放速度不一样，
        如果record_performance_speed不为0，录像是跳跃式播放的，录下来的文件也是取关键帧录制的
        。
3237    2）使用类：IPCam
3238    3）函数
3239
3240    /*
```

```
 *IPCam方法：
 */
public ERROR set_play_tf_record_cache(int cache)

/*
 *IPCam方法：设置播放（下载）速度
 *参数
 *speed：播放（下载速度），有四个值
 *0:对应sosocam的1x,正常播放（下载）
 *3:对应sosocam的8x,跳跃播放（下载）
 *4:对应sosocam的16x,跳跃播放（下载）
 *5:对应sosocam的32x,跳跃播放（下载）
 */
public void set_record_performance_speed (int speed); //设置播放（下载）速度
/*
 *IPCam方法：播放（下载）速度
 *返回
 *speed：返回一个int数值
 *0:对应sosocam的1x,正常播放（下载）
 *3:对应sosocam的8x,跳跃播放（下载）
 *4:对应sosocam的16x,跳跃播放（下载）
 *5:对应sosocam的32x,跳跃播放（下载）
 */
public int record_performance_speed () //播放（下载）速度
```

4）注意：a）这些操作必须以摄像机连接成功为前提

b)只支持SOSO固件

5）示例

```
/*示例一：设置（播放）下载tf卡录像文件速度*/
speed_slower = (ImageView)findViewById(getResources().getIdentifier("speed_slower",
"id", m_package_name));
speed_slower.setOnClickListener(new View.OnClickListener() {
    /*这里示范的是减速设置*/
    @Override
    public void onClick(View arg0) {
        String speed_v = "";
        speed_v = (String) record_play_speed.getText();//获取当前播放（下载）速度

        if (m_local_recording)
            return;
        if (speed_v.equals("1X"))
            m_ipcam.set_record_performance_speed(5);//设置播放（下载）速度
        else if (speed_v.equals("8X"))
            m_ipcam.set_record_performance_speed(0);
        else if (speed_v.equals("16X"))
            m_ipcam.set_record_performance_speed(3);
        else if (speed_v.equals("32X"))
            m_ipcam.set_record_performance_speed(4);
        m_ipcam.stop_tf_record();//停止播放

        record_play_speed.setText(get_performance_speed_string(m_ipcam.record_performa
        nce_speed()));
        m_ipcam.play_tf_record(m_ipcam.get_tf_record_play_id(m_day, m_hour,
        m_quarter, m_no));
    }
});

/*示例二：下载tf卡录像文件*/
play_record =
(ImageView)findViewById(getResources().getIdentifier("download_record", "id",
m_package_name));
play_record.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View arg0) {
        if (m_local_recording){
            stop_local_record(true);
        }else{
            m_local_record_filepath = m_ipcam.start_local_record();
            if (m_local_record_filepath == null) {
                Log.e("SODemo","record fail");
            } else {
```

```
                    m_local_recording = true;
                    if (m_ipcam.record_performance_speed() != 0) {
                        m_ipcam.set_record_performance_speed(0);
                        record_play_speed.setText(get_performance_speed_string(0));
                    }
                    m_ipcam.set_play_tf_record_cache(0xffffffff);
                }
            }
        }
    });
```

17.3.5 获取10s录像信息

1)说明：获取10s录像信息是指：获取每个10s文件的录像时间，获取每个10s录像文件的预览图，获取每个10s录像文件的报警信息等

2）使用类：IPCam

3）函数

```
/*
 *IPCam方法：获取一刻钟的某一个10s录像文件时间
 *
 *参数：
 *subrecord_id
 *返回：返回一个TF_RECORD_CLIP_TIME对象，具体如下：
 public class TF_RECORD_CLIP_TIME {
        public int day;   天
        public int hour;  时
        public int quarter;   刻钟
        public int no;   当前刻钟的第几个10S
    }
 */
public TF_RECORD_CLIP_TIME get_tf_record_clip_time(int subrecord_id)

/*
 *IPCam方法：获取一刻钟的某一个10s录像预览图图
 *参数：
 *day:第几天，共7天，范围［0，7）
 *hour：第几个小时，共24个小时，范围[0，24)
 *quarter:第几刻钟，共4刻钟，范围[0,4)
 *no:第几个10s,15分钟，10s中一个文件，共90个文件，范围[0,90)
 *listener:get_tf_record_clip_thumb_listener
 *返回：
 */
public ERROR get_tf_record_clip_thumb(int day, int hour, int quarter, int no,
get_tf_record_clip_thumb_listener listener)

/*
 *IPCam方法：get_tf_record_clip_thumb_listener回调
 *返回参数：
 *ipcam:摄像机
 *day:第几天，共7天，范围［0，7）
 *hour：第几个小时，共24个小时，范围[0，24)
 *quarter:第几刻钟，共4刻钟，范围[0,4)
 *no:第几个10s,15分钟，10s中一个文件，共90个文件，范围[0,90)
 *thumb：预览图数据
 */
public void on_result(IPCam ipcam, int day, int hour, int quarter, int no, byte[]
thumb);

/*
 *IPCam方法：获取某一刻钟90个10s录像文件报警信息
 *
 *参数：
 *day:天，范围［0，7）,类型 NSNumber（int）
 *hour:小时，范围［0，24）,类型 NSNumber（int）
 *quarter:刻钟，范围［0，7）,类型 NSNumber（int）
 *listener:get_tf_record_quarter_detail_listener
 */

public ERROR get_tf_record_quarter_detail(int day, int hour, int quarter,
get_tf_record_quarter_detail_listener listener)

/*
 *IPCam方法：get_tf_record_quarter_detail_listener回调
 *
```

```
 *返回:
 *ipcam:摄像机
 *day:天
 *hour:时
 *clips:一个TF_RECORD_CLIP_INFO对象，90个10s的录像文件信息，具体如下：
 public class TF_RECORD_CLIP_INFO {
        public boolean valid;  当前刻钟是否有录像文件
        public boolean thumb;  当前刻钟是否有预览图
        public int alarm;  当前刻钟是否有报警
    }
 */
public void on_result(IPCam ipcam, int day, int hour, int quarter,
TF_RECORD_CLIP_INFO[] clips);

4)示例

/* 示例一：获取某一刻的90个文件的信息*/
m_ipcam.get_tf_record_quarter_detail(day, hour, quarter, this);

//get_tf_record_quarter_detail_listener回调
public void on_result(IPCam ipcam, int day, int hour, int quarter,
TF_RECORD_CLIP_INFO [] clips) {
        if ((m_day == day) && (m_hour == hour) && (m_quarter == quarter)) {
            int i;
            if (clips == null) {
            } else {
                m_clips_info = clips;
                for (i = 0;i < 90;i ++) {
                    if (! clips[i].valid)
                        //此10s文件有录像文件
                    else if (clips[i].alarm == 0)
                        //此10s文件有录像文件+无报警
                    else
                        //此10s文件有录像文件+且有报警
                }
            }
        }
    }

/* 示例二：通过播放id获取当前文件时间*/

TF_RECORD_CLIP_TIME t = m_ipcam.get_tf_record_clip_time(record_id);
m_day = t.day;//天
m_hour = t.hour;//时间
m_quarter = t.quarter;//刻钟
m_no = t.no;//第几个10秒

int minutes = quarter * 15 + no / 6;
int seconds = (m_no * 10) % 60;


/*示例三：通过获取10s文件预览图*/
m_ipcam.get_tf_record_clip_thumb(m_day, m_hour, m_quarter, no, this);

//get_tf_record_clip_thumb_listener回调
public void on_result(IPCam ipcam, int day, int hour, int quarter, int no, byte []
thumb) {
        if (thumb == null) {
            m_clip_thumb.setImageBitmap(null);
        } else {
            m_clip_thumb.setImageBitmap(BitmapFactory.decodeByteArray(thumb, 0,
            thumb.length));
        }
    }

17.4.6取消tf卡录像文件的所有操作
1）说明：取消tf卡录像文件的所有操作是指把tf录像相关操作全部取消，除了load_tf_records。
如果想要继续操作tf卡录像文件，直接操作就好不需要重新load_tf_records。
2）使用类：IPCam
3）函数：
/*
 *IPCam方法：取消tf卡录像文件的所有操作
 */
```

```
public void cancel_tf_record_tasks()

4)示例
m_ipcam.cancel_tf_record_tasks();
```