

#1. SoSoCamLib简介

1. SoSoCamLib简介

SoSoCamLib由7个部分组成，整个libSoSoCamSDK的使用思路是：
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。
SDK各个部分的功能以及之间的联系，如下：

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IPCamMgr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IPCamMgr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IPCam的方法。

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。

IPCamVideoView: IPCamVideoView是观看视频必需要用到的一类。IPCamVideoView其实就是一个UIView，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。

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] 。

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。

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。

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。

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。

#2. 集成SoSoCamLib

1) 说明：集成SoSoCamLib是指把SoSoCamLib集成到你的项目中，供你使用

2) 版本支持：

Android 2.2及以上版本

3) 组成：

java的jar文件：sosocamlib.jar
工程中存放位置：libs\sosocam.jar
库文件：librcipcam3x.so
libsinvoice.so
工程中存放位置：libs\armeabi-v7a\
资源文件：

loading_step1.png
loading_step2.png
loading_step3.png
start_play.png

4) Manifest.xml中权限设置

<!-- for storage module -需要使用到storage时添加下列权限-->

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />  
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />  
<uses-permission android:name="android.permission.WRITE_OWNER_DATA" />  
<uses-permission android:name="android.permission.READ_OWNER_DATA" />
```

<!-- for ipcam and ipcammgr modules -使用到IPCam和IPCamMgr需用到的权限-->

```
<uses-permission android:name="android.permission.INTERNET" />  
<uses-permission android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE" />
```

```

56 <uses-permission android:name="android.permission.WAKE_LOCK" />
57 <uses-permission android:name="android.permission.RECORD_AUDIO" />
58 <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
59 <uses-permission android:name="android.permission.WRITE_MEDIA_STORAGE"/>
60 <uses-permission android:name="android.permission.GET_TASKS" />
61 <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
62 <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
63

```

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:

```

82 /*
83  *初始化SDK
84  */
85 IPCamMgr.init (this); //创建IPCamMgr 初始化
86
87 /*
88  *反初始SDK
89  */
90 IPCamMgr.deinit (); //释放IPCamMgr, 反初始化
91

```

5) 示例:

```

93 /*使用SDK之前初始化SDK*/
94 protected void onCreate(Bundle savedInstanceState) {
95     super.onCreate(savedInstanceState);
96     IPCamMgr.init (this); //初始化
97 }
98
99 /*程序退出反初始化SDK*/
100 protected void onDestroy() {
101     super.onDestroy();
102     IPCamMgr.deinit (); //反初始化
103 }

```

#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索局域网中的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。

```

114 /*
115  *获取局域网摄像机
116  *返回的是DISCOVER_CAMERA_INFO类型的对象:
117  *String id; //摄像机id
118  *int id_type; //摄像机id type
119  *String alias; //设备的别名
120  *String fw_version; //系统固件版本
121  *String ui_version; //网页固件版本
122  *int model; //摄像机类型, 两个值,
123  *String current_ip; //设备当前的IP地址;
124  *int port; //设备当前的端口;

```

```

125     *boolean https; //设备当前是否安全加密传输, true:加密 false:不加密
126     *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
127     */
128     public static LinkedHashMap<String, DISCOVERED_CAMERA_INFO>
get_discovered_cameras_list(); //获取局域网里的cams
129     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

```

130

131 **5) 步骤:**

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

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

134

135 **6) 示例:**

136 /*

137 *第一步: 初始化开始IPCamMgr 开始后台局域网搜索

138 */

139 IPCamMgr.init(this); //初始化

140

141 /*

142 *第二步: 获取局域网摄像机

143 */

144 LinkedHashMap<String, DISCOVERED_CAMERA_INFO> ipcam_list;

145 ipcam_list = IPCamMgr.get_discovered_cameras_list();

146 for(DISCOVERED_CAMERA_INFO ipcam : ipcam_list.values())

147 {

148 if(!ipcam.used)

149 Log.e("SoDemo", "Current is not used");

150 }

151

152 **#5.摄像机的添加 / 删除 / 修改更新**

153 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:

154 添加: WebService添加 / 删除 / 修改更新——》Storage添加 / 删除 / 修改更新——》IPC

155 amMgr添加 / 删除 / 修改更新。

156 这些过程按照你的需求, 择选, 详细如下:

157 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];

158 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;

159 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

160 。 [推荐使用IPCamMgr管理];

161

162 **5.1摄像机的添加<服务器 / 本地 / IPCamMgr>**

163 **1) 说明:** 摄像机的添加是指添加摄像机到服务器 / 本地 / IPCamMgr, 添加成功后, 你才可以对

164 它操作, 监控它的状态等。

165

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

167

168 **3) 函数:**

169 /*

170 *Storage类

171 *Storage的帐号创建和更新

172 *详细用法参考<Storage的本地存储用户更新>

173 */

174 public static void update_user(String user_id, String name, String pwd, int

175 user_type, String alias);

176 /*

177 *Storage类

178 *Storage的设置当前帐号

179 *详细用法参考<Storage的本地存储用户管理更新>

180 */

181 public static void set_current_user(String user_id); //设置user_id为当前账号

182 /*

183 *WebService添加摄像机到服务器

184 *参数:

185 *alias:摄像机昵称

186 *camera id: 摄像机id

187 *https:安全加密传输, 两个值: YES:加密 NO:非加密

188 *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

189 它默认一直是0。我们是定义0云台机 1为卡片机

190 *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

191 *返回:

192 *当ERROR == ERROR.NO_ERROR时表示添加执行成功

193 */

194 public static ERROR add_camera(String alias, String id, boolean https, int model,

```

190 add_camera_listener listener); //添加摄像机到服务器
191 /*
192 *Storage添加摄像机到本地
193 CAMERA_INFO对象参数:
194 *String id //摄像机id
195 *String alias //摄像机昵称
196 *String user //摄像机登录用户名
197 *String pwd //摄像机登录密码
198 *boolean https //加密安全传输, 两个值 0:非加密 1:加密
199 *String obj_id //摄像机sosocam_id
200 *String model //摄像机模型, 两个值, 0 云台机 1 卡片机
201 */
202 public static void add_camera(CAMERA_INFO camera_info);
203
204 /*
205 *IPCamMgr添加摄像机到IPCamMgr
206 *参数:
207 *String alias: 摄像机昵称
208 *String id:摄像机id
209 *String user:摄像机登录用户名
210 *String pwd:摄像机登录密码
211 *https:安全加密传输, 两个值: YES:加密 NO:非加密
212 *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
213 */
214 public static IPCam add_camera(String alias, String id, String user, String pwd,
215 boolean https); //添加摄像机到IPCamMgr*
216
217 /*
218 *IPCam方法: 更新摄像机局域网信息
219 *参数如下:
220
221 *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false
222 *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
223 *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
224 *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。
225 */
226 public void update_lan_status(boolean in_lan, String ip, int port, boolean https)
227
228 /*
229 *IPCam方法: 连接摄像机
230 *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
231 *参数如下:
232 *retryable:是否重连,true:当因为网络错误时会自动重连接; false:不重连
233 *retry_delay:int类型, 单位ms,延迟多长时间重连。
234 */
235 public ERROR start_connect(boolean retryable, int retry_delay)
236

```

4) 步骤:

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。

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>

第三步: 用WebService的方法添加到服务器上, 如果不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>

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>

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>

第六步: 用Storage的方法把摄像机添加到本地。

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>

248

249 5) 注意:

250

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>

251

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。

252

3) 如果只要添加到本地, 直接使用第一, 第六, 第七步骤即可。

253

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。

254

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

255

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

256

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。

257

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。

258

259 6) 示例:

260 /*第一步: 创建更新一个帐号, 且设置当前帐号*/

261 Storage.update_user("local@localhost", "", "", Storage.OFFLINE_USER, "");

262 Storage.set_current_user("local@localhost");

263

264 /*第二步: 更新局域网信息, 连接摄像机*/

265 public void update_lan_camera(String camera_id){

266 LinkedHashMap<String, DISCOVERED_CAMERA_INFO> cameras =

IPCamMgr.get_discovered_cameras_list();

267 DISCOVERED_CAMERA_INFO camera = cameras.get(camera_id);

268 if ((camera != null) && (! camera.current_ip.equals("0.0.0.0"))) {

269 /*如果id在局域网内就更新局域网信息*/

270 m_ipcam.update_lan_status(true, camera.current_ip, camera.port, camera.https);

271 }

272 /*连接摄像机

273 *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

274 */

275 if (ERROR.NO_ERROR == m_ipcam.start_connect(false, 0)) {

276

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

277 }

278 }

279

280 /*第三步: 监控摄像机连接成功后, 开始添加摄像机到服务器*/

281 public void on_status_changed(IPCam ipcam) {

282 IPCam.CONN_STATUS status;

283 IPCam.ERROR error;

284 if (m_status == CONN_STATUS.CONNECTED){

285 /*连接摄像机成功, 开始添加到服务器*/

286 add_camera_to_server();

287 }

288 else if (m_status == CONN_STATUS.IDLE){

289 /*连接摄像机失败, 删除摄像机代理*/

290 m_ipcam.remove_listener();

291 }

292 }

293 private void add_camera_to_server() {

294 //添加到WebService

295 if (WebService.ERROR.NO_ERROR != WebService.add_camera(name, m_ipcam.id(), m_ipcam.https(), m_ipcam.model, this)) {

296 Log.e("SoDemo", "failed_add_camera");

297 } else {

298 Log.e("SoDemo", "add_camera_success");

299

300 }

301 }

```

302 //WebService add_camera结果监听
303 public void on_add_camera_result(WebService.ERROR error, String obj_id) {
304     if (WebService.ERROR.NO_ERROR == error) {
305         /*监控到添加添加摄像机到服务器成功*/
306         m_ipcam.relogin_to_sosocam(this); //第五步: 重新登录到服务器
307         add_camera_local(obj_id); //第六步: 添加到本地
308     } else if (WebService.ERROR.REGISTERED_BY_OTHERS == error) {
309         m_ipcam.unregister_from_sosocam(this); //第四步: 摄像机已被其他账号添加, 先解绑
310     } else {
311         /*添加摄像机到服务器失败, 删除摄像机代理*/
312         m_ipcam.remove_listener(this);
313     }
314 }
315
316 //IPCam unregister_from_sosocam结果监听
317 public void on_result(IPCam ipcam, ERROR error) {
318     ipcam.stop_connect();
319     if (m_state == STATE.UNREGISTER_CAMERA) {
320         if (error == ERROR.NO_ERROR) {
321             add_camera_to_server(); //第四步: 解绑成功, 重新添加摄像机到服务器
322         } else {
323             m_ipcam.remove_listener(this); //解绑失败, 需要删除摄像机代理
324         }
325     }
326 }
327
328 private void add_camera_local(String obj_id)
329 {
330     CAMERA_INFO add_cam = new CAMERA_INFO();
331     if (obj_id == null) obj_id = "";
332     add_cam.setObj_id(obj_id);
333     add_cam.setAlias(m_camera_name);
334     add_cam.setId(m_ipcam.camera_id());
335     add_cam.setUser(m_ipcam.user());
336     add_cam.setPwd(m_ipcam.pwd());
337     add_cam.setHttps(m_ipcam.https());
338     add_cam.setModel(m_ipcam.model);
339     Storage.add_camera(add_cam);
340     Storage.save_cameras();
341     IPCam ipcam = IPCamMgr.add_camera(m_camera_name, m_ipcam.camera_id(),
342     m_ipcam.user(), m_ipcam.pwd(), m_ipcam.https()); //第七步: 添加到IPCamMgr
343     if (ipcam != null) {
344         ipcam.model = m_ipcam.model;
345         ipcam.sosocam_id = obj_id;
346     }
347 }

```

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:

```

352 /*
353  *WebService从服务器上删除摄像机
354  *参数:
355  *obj_id:摄像机sosocam_id
356  *delegate:接口delete_camera_listener, 添加后可以监控删除结果
357  */
358 public static ERROR delete_camera(String obj_id, delete_camera_listener listener);
359
360 /*
361  *Storage从本地删除摄像机
362  *参数:
363  *camera_id:摄像机id
364  */
365 public static void remove_camera(String camera_id);
366
367 /*
368  *IPCamMgr删除摄像机
369  *camera_id:摄像机id
370  *另外IPCamMgr删除有个接口IPCamMgr_Listener, 添加后可以监控删除结果
371  */
372 public static IPCam delete_camera(String id);
373

```

374 4) 步骤:

375

376 第一步: 用WebService的方法从服务器上删除摄像机, 如果不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>

377 第二步: 用Storage的方法把从本地删除摄像机。

378 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>

379

380 5) 示例:

381 /*第一步: 从服务器上删除摄像机+添加监听on_delete_camera_result*/

382 public void delete_camera_from_server(){

383 if (WebService.ERROR.NO_ERROR != WebService.delete_camera(obj_id, DeleteCameraDialog.this)) {

384 Log.e("SoDemo", "remove camera from service fail!");

385 }

386 }

387 /*第二步: 从本地和IPCamMgr删除摄像机*/

388 public void on_delete_camera_result(WebService.ERROR error) {

389 if (WebService.ERROR.NO_ERROR == error) {

390 /*监控到从服务器删除摄像机成功*/

391 Storage.remove_camera(m_camera_id); //从本地删除摄像机

392 Storage.save_cameras();

393 IPCamMgr.delete_camera(m_camera_id); //从IPCamMgr删除摄像机

394 Log.e("Sodemo", "remove camera from service success!");

395 }else{

396 Log.e("Sodemo", "remove camera from service fail!");

397 }

398 }

399

400 5.3摄像机的修改更新

401 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:

402 服务器: 更新名称

403 IPCamMgr: 更新密码

404 Storage: 更新名称 / 登录名 / 登录密码 / https / id / sosocam_id / cover

405

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

407

408 3) 注意:

409 a) 更新涉及到很多内容, 这里着重讲密码和名称的更新。

410 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。

411 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。

412 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>

413

414 4) 函数

415 /*

416 *IPCam修改摄像机登录密码

417 *参数:

418 *pwd: 摄像机新密码

419 *listener: 接口reset_pwd_listener可以监控摄像机设置密码的结果

420 *返回:

421 *返回ERROR = ERROR.NO_ERROR表示设置新密码成功

422 */

423 public ERROR reset_pwd(String pwd, reset_pwd_listener listener)

424

425 /*

426 *IPCam设置IPCam的cam名称

427

428 */

429 public void set_alias(String alias);

430

431 /*

432 *WebService更新参数

433 *参数:

434 *obj_id: 摄像机sosocam_id

435 *params: 新参数

436 *listener: 接口update_camera_listener可以监控更新结果

437 *返回:

```

438 *当ERROR == ERROR.NO_ERROR时表示更新执行成功
439 */
440 public static ERROR update_camera(String obj_id, String params,
update_camera_listener listener);
441
442 /*
443 *IPCamMgr更新密码到IPCamMgr
444 *参数:
445 *id:摄像机id
446 *pwd:新密码
447 */
448 public static void update_camera_pwd(String id, String pwd);
449
450 /*
451 *Storage更新摄像机的名称
452 *参数
453 *camera_id:要被更新的摄像机id
454 *camera_alias:摄像机昵称更新为camera_alias
455 */
456 public static void update_camera_alias(String camera_id, String camera_alias);
457
458 /*
459 *Storage更新摄像机的登录密码
460 *参数
461 *camera_id:要被更新的摄像机id
462 *camera_pwd:摄像机登录密码更新为camera_pwd
463 */
464 public static void update_camera_pwd(String camera_id, String camera_pwd);
465

```

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*/

```

479 if (ERROR.NO_ERROR == m_ipcam.reset_pwd(m_pwd, this)){ //重置密码
480     m_state = STATE.SET_CAMERA;
481 }

```

/*第二步: 更新到IPCamMgr和Storage*/

```

483 public void on_result(IPCam ipcam, ERROR error) {
484     if (m_state == STATE.SET_CAMERA) {
485         if (error == ERROR.NO_ERROR) {
486             Storage.update_camera_pwd(m_ipcam.id(), m_ipcam.pwd()); //更新保存到本地
487             Storage.save_cameras();
488             IPCamMgr.update_camera_pwd(m_ipcam.id(), m_ipcam.pwd()); //更新到IPCamMgr
489         }else{
490             Log.e("Sodemo", "----Failed---to----reset--pwd--");
491         }
492     }
493 }

```

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*/

```

497 if (WebService.ERROR.NO_ERROR !=
WebService.update_camera(Storage.get_camera_obj_id(m_camera_id),
498     "alias=" + name, this)) {
499     Log.e("SoDemo", "update name to service fail!");
500 } else {
501     Log.e("SoDemo", "update name to service succeed!");
502 }

```

/*第二步: 更新名称到IPCam和本地*/

```

505 public void on_update_camera_result(WebService.ERROR error) {
506     if (WebService.ERROR.NO_ERROR == error) {
507         m_ipcam.set_alias(m_camera_name); //更新名称到IPCam

```



```

508         Storage.update_camera_alias(m_camera_id, m_camera_name); //更新名称到本地
509         Storage.save_cameras();
510         Log.e("SoDemo", "update_camera_name_ok");
511     } else {
512         Log.e("SoDemo", "failed_update_camera_name");
513     }
514 }
515

```

516 #6. 摄像机连接操作

517 说明：摄像机的连接有两种：第一种是通过 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，Reecam 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。

518 6.1 通过 IPCamMgr 摄像机的连接 [Reecam 推荐的连接方式]

519 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。

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

523 3) 注意点：

524 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。

525 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。

526 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]

527 4) 函数：

```

528 /*
529  *SDK初始化 / 反初始化+开始 / 停止 IPCamMgr
530  *详情参考：IPCamMgr 的“SDK初始化和反初始化”
531  */
532 IPCamMgr.init(this); //创建 IPCamMgr 初始化
533
534 IPCamMgr.deinit(); //释放 IPCamMgr，反初始化
535 /*
536  *局域网搜索 [可选]
537  * 详情参考：IPCamMgr 的“局域网搜索”
538  */
539 public static LinkedHashMap<String, DISCOVERED_CAMERA_INFO>
540 get_discovered_cameras_list();
541
542 /*
543  *IPCamMgr 摄像机添加+添加的监听接口
544  *详细参考：IPCamMgr 的“添加和删除监听”
545  */
546 public static IPCam add_camera(String alias, String id, String user, String pwd,
547 boolean https);
548
549 /*
550  *摄像机连接状态 [可选]
551  *详细参考：IPCam 的“摄像机基础监听”
552  */
553 public void on_status_changed(IPCam ipcam);
554
555 /*
556  *立即连接 [可选]
557  *详细参考：IPCamMgr 的“立即连接”
558  */
559 public static void reset_camera(String id);
560
561 /*
562  *moblie 连接控制 [可选]
563  *详细参考：IPCamMgr 的“数据 (moblie) 联网下的连接设置”
564  */
565 public static void set_forbidden_in_mobile(boolean forbidden);
566

```

568 5) 步骤:
569 第一步: 用IPCamMgr的init函数进行sdk初始化开始IPCamMgr;
570
571 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I
PCamMgr, 同时删除摄像机后, 也必须从IPCamMgr删除摄像机]
572
573 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>]
574
575 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。
576
577 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)。

578 6) 例子:

```
579 /*  
580  *第一步: 使用IPCamMgr的init函数初始化SDK  
581  */  
582 IPCamMgr.init(this);  
583  
584 /*  
585  *第二步: 添加摄像机到PCamMgr  
586  */  
587 LinkedHashMap<String, DISCOVERED_CAMERA_INFO> ipcam_list;  
588 ipcam_list = IPCamMgr.get_discovered_cameras_list();  
589  
590  
591 IPCam ipcam = IPCamMgr.add_camera(m_ipcam.alias, m_ipcam.id, "admin", "ipcam_pwd",  
m_ipcam.https); //添加摄像机到IPCamMgr  
592 /*  
593  *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]  
594  */  
595 m_ipcam.add_listener(this); //添加IPCam_Listener代理  
596  
597 public void on_status_changed(IPCam ipcam) {  
598     Log.e("SoDemo", "cam connect status is::" + ipcam.status());  
599 }  
600 .  
601 .  
602 .  
603  
604 /*  
605  *第四步[可选]: 立即连接  
606  */  
607 IPCamMgr.reset_camera(ipcam.id());  
608  
609 /*  
610  *第五步[可选]: 摄像机数据流量下连接连接设置  
611  */  
612 IPCamMgr.set_forbidden_in_mobile(true); //设置数据流量下禁止连接
```

614 6.2通过IPCam摄像机的连接,

615 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IPC
616 am, 然后调用连接函数连接。

617 2) 使用的类: IPCamMgr, IPCam

618 3) 注意点:

619 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
620 此时摄像机https是true, 这时是连接不上的。

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

623 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]

624 4) 函数:

```
625 /*  
626  *更新摄像机局域网信息[可选], 注意:  
627  
628
```

```

629 *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
630
        *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。
631
        *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
632 */
633 /*
634 *参数如下：
635 *ip: 当前摄像机ip;
636 *port: 当前摄像机port
637 *retryable:是否重连
638 *retry_delay:重连间隔
639 */
640 public ERROR start_connect(String ip, int port, boolean retryable, int retry_delay);
641
642 /*
643 *连接函数
644 *参数如下：
645 *retryable:是否重连,true:当因为网络错误时会自动重连接; false:不重连
646 *retry_delay:int类型，单位ms,间隔多长时间进行重连。
647 */
648 public ERROR start_connect(boolean retryable, int retry_delay);
649
650 /*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
651 */
652 public void stop_connect();
653
654
655 5) 步骤：
656 第一步：用IPCamMgr的init函数进行sdk初始化开始IPCamMgr;
657 第二步：new一个IPCam,用IPCam的start_connect连接摄像机。
658
659 6) 示例
660 /*
661 *第一步：使用PCamMgr的init函数初始化SDK
662 */
663 IPCamMgr.init(this);
664
665 /*
666 *第二步：new一个IPCam+设置IPCam的参数
667 */
668 public void set_cameras()
669 {
670     IPCam m_ipcam = new IPCam();
671     m_ipcam.set_id("SOSO-000019-VLTVM");
672     m_ipcam.set_user("admin");
673     m_ipcam.set_pwd("88888888");
674     m_ipcam.set_https(false);
675 }
676
677 /*
678 *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]
679 */
680 m_ipcam.add_listener(this);//添加IPCam_Listener接口
681 -(void)on_status_changed:(id)ipcam{
682
683     CONN_STATUS conn_status = m_ipcam.status();
684 }
685 .
686 .
687 .
688 /*
689 *第三步：连接摄像机
690 */
691 m_ipcam.start_connect(false, 0);
692
693 /*
694

```

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
702  和服务器的获取以及对比同步。流程如下：
703  WebService里面获取添加到服务器的摄像机 ——> WebService对比同步 ——>
704  Storage获取本地的摄像机 ——> 添加到IPCamMgr。
705  这些过程按照你的需求，择选，详细如下：
706  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]；
707  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
708  比同步[可选项]；
709  不管用不用到平台，都需要在Storage获取[必须]；
710  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IPCa
711  mMgr管理]；
712
713  2) 使用类：storage, IPCamMgr, WebService
714  3) 注意点：
715
716  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Stora
717  ge存储的方法请参考上面的<摄像机的添加 / 删除 / 修改更新>。
718  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。
719
720  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
721  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
722  机都从IPCamMgr获取即可。[推荐方式]
723
724  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
725  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
726  机，逐个添加到IPCamMgr。
727
728  4) 函数：
729
730  /*
731  *storag的方法
732  *storage获取本地存储的摄像机
733  *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：
734  *String id //摄像机id
735  *获取：public String getId()
736
737  *String alias //摄像机昵称
738  *获取：public String getAlias()
739
740  *String user //摄像机登录用户名
741  *获取：public String getUser()
742
743  *String pwd //摄像机登录密码
744  *获取：public String getPwd()
745
746  *boolean https //加密安全传输，两个值 0:非加密 1:加密
747  *获取：public boolean getHttps()
748
749  *String obj_id //摄像机sosocam_id
750  *获取：public String getObj_id()
751
752  *byte [] cover //摄像机预览图data数据
753  *获取：public Bitmap getCover()
754
755  *String model //摄像机模型，两个值，0 云台机 1 卡片机
756  *获取：public int getModel()
757  */
758  public static ArrayList<CAMERA_INFO> get_cameras(); //storage获取本地存储的摄像机
759
760  /*
761  *IPCamMgr方法
762  *添加摄像机到IPCamMgr
763  *参数：
764  *alias:摄像机昵称
765  *id: 摄像机id
766  *user:摄像机登录用户名
```

```

754 *pwd:摄像机登录密码
755 *https:安全加密传输, 两个值: true:加密 false:非加密
756 *另外IPCamMgr添加有个监听IPCamMgr_Listener, 添加后可以监控添加结果
757 */
758 public static IPCam add_camera(String alias, String id, String user, String pwd,
boolean https);
759
760 /*
761 *WebService的方法
762 *WebService获取服务器当前帐号的摄像机
763 *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:
764 * String obj_id; 摄像机sosocam_id
765 * String alias; 摄像机昵称
766 * String id; 摄像机id
767 * boolean https; 摄像机传输是否加密
768 * String user; 摄像机登录用户名
769 * String pwd; 摄像机登录密码
770 */
771 public static ArrayList<SERVICE_CAMERA_INFO> svr_cameras = new
ArrayList<SERVICE_CAMERA_INFO>();
772 svr_cameras = Webservice.cameras;//当前账号里添加的摄像机
773
774 /*
775 *storage的方法
776 *storage的比较方法
777 *参数:
778 *server_cameras_list:服务器上获取到的摄像机
779 *返回: 返回一个返回BOOL值
780 *如果返回true表示:服务器上的cams和本地的cams一样
781 *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
782 */
783 Storage.merge_cameras(WebService.cameras);
784

```

5) 步骤:

```

786 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>
787 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>
788 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>
789 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>
790 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IP
CamMgr的摄像机, 然后逐个添加到IPCamMgr。

```

6) 示例:

```

792 /*第一步: 获取服务器上当前存储的“添加摄像机”*/
793 public static ArrayList<SERVICE_CAMERA_INFO> svr_cameras = new
ArrayList<SERVICE_CAMERA_INFO>();
794 svr_cameras = Webservice.cameras;
795
796 /*第二步: 对比且同步本地和服务上该帐号的摄像机*/
797 Storage.merge_cameras(WebService.cameras);
798
799 /*第三步: 获取该帐号本地的所有摄像机*/
800 ArrayList<CAMERA_INFO> m_ipcams_list = Storage.get_cameras();//获取本地cams
801
802 /*第四步: 添加摄像机到IPCamMgr, 用于批量管理摄像机*/
803 public void add_cameras_to_ipcammgr(){
804     IPCamMgr.init(this);
805     CAMERA_INFO cam;
806     int i;
807     for(CAMERA_INFO cam : Storage.get_cameras()){
808         IPCam ipcam = IPCamMgr.add_camera(cam.getAlias(), cam.getId(),
cam.getUser(), cam.getPwd(), cam.getHttps());//逐一添加cam到IPCamMgr
809         /*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
810         *这样子IPCamMgr的cam就具备这些属性了
811         */
812         ipcam.model = cam.getModel();
813         ipcam.sosocam_id = cam.getObj_id();
814         ipcam.cover = cam.getCover();
815     }
816 }
817
818 /*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(),
827                                                         cam.getUser(), cam.getPwd(), cam.getHttps());
828                     ipcam.cover = cam.getCover();
829                     ipcam.model = cam.getModel();
830                     ipcam.sosocam_id = cam.getObj_id();
831                 }
832             }
833             Storage.merge_collections(WebService.collections);
834             break;
835         }
836     }
```

#8.摄像机基本功能实现

8.1.摄像机视频

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。

2) 使用的类: IPCam, IPCamVideoView

3) 函数

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
```

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 }
```

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fps
870 public int video_byterate(); //视频的byte
```

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);
```

e) 停止播放视频

```
883 /*
884  *IPCam类: 停止播放视频
885  */
886 public void stop_video();
```

f) 视频监听

```
889 /*
```

```

890 *IPCamVideo类
891
892 *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
893 *返回参数:
894 *event:返回TOUCH_EVENT参数，具体如下:
895 */
896 public void set_listener(IPCamVideoView_Listener
listener);//设置IPCamVideoView_Listener接口
897 public void on_touch_event(TOUCH_EVENT event); //IPCamVideoView_Listener回调函数
898 public enum TOUCH_EVENT {
899     CLICK, //单点
900     MOVE_UP, //向上滑动
901     MOVE_DOWN, //向下滑动
902     MOVE_LEFT, //向左滑动
903     MOVE_RIGHT;//向右滑动
904 }
905
906 /*
907 *IPCam类
908 *IPCam_Listener:需在监控视频状态，就要在IPCam上添加此接口
909 *IPCam_Listener的详细用法请参考IPCam的基础接口
910 */
911 public void add_listener(IPCam_Listener listener);//设置IPCam_Listener接口
912 public void on_video_status_changed(IPCam ipcam);
//IPCam_Listener回调函数，视频状态改变时回调
913 public void remove_listener(IPCam_Listener listener); //删除IPCam_Listener代理
914 4) 注意点:
915     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
916     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
917     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。
918
919 5) 步骤:
920 第一步: 创建并设置视频页面。
921 第二步:实现接口IPCamVideoView_Listener和IPCam_Listener，这一步根据需要添加。<可选>
922 第三步: 进行视频操作: 播放停止 / 视频流切换。
923 第四步: 处理回调函数: 添加了监听才有回调函数。<可选>
924
925 6) 示例:
926 /*第一步: 设置视频页面*/
927 IPCamVideoView m_video_view = (IPCamVideoView)findViewById(R.id.LiveVideoView);
928 m_ipcam.set_video_view(m_video_view);//设置视频页面
929 int m_playing_video_stream = 0; //定义默认播放视频流
930
931 /*第二步: 在视频上添加代理IPCamVideoView_Listener和IPCam_Listener*/
932 m_video_view.set_listener(this); //m_videoview添加IPCamVideoView_Listener不是m_ipcam
933 m_ipcam.add_listener(this); //m_ipcam添加IPCam_Listener
934
935 /* 第三步:视频播放和停止*/
936 case R.id.play_start:
937     {
938         if (m_ipcam.video_status() == PLAY_STATUS.PLAYING) { //判断当前视频状态
939             m_ipcam.stop_video();// 停止播放视频
940         }else{
941             m_ipcam.play_video(m_playing_video_stream);//播放m_playing_video_stream路
942             视频流
943             /*视频质量如下*/
944         }
945         Log.e("视频发送: ",m_ipcam.video_render_fps() + "fps"); //发送fps
946         Log.e("视频接收: ",m_ipcam.video_recv_fps() + "fps"); //接收fps
947         Log.e("视频当前: ",m_ipcam.video_byterate() + "byte"); //当前视频byte
948     }
949
950 /*第三步:设置播放视频流，即高清和标清设置或切换 */
951 case R.id.play_change_screen:
952     {
953         if (m_playing_video_stream == 0){
954             m_playing_video_stream = 1;

```

```

953         m_ipcam.stop_video();
954         m_ipcam.play_video(m_playing_video_stream);
955     }else{
956         m_playing_video_stream = 0;
957         m_ipcam.stop_video();
958         m_ipcam.play_video(m_playing_video_stream);
959     }
960 }
961
962 /*第四步: IPCamVideoView_Listener回调实现*/
963 public void on_touch_event(IPCamVideoView_Listener.TOUCH_EVENT event) {
964     switch (event) {
965         case CLICK :
966             Log.e("SoDemo", "单点了一下");
967             break;
968         case MOVE_UP:
969             Log.e("SoDemo", "向上移动了一下");
970             break;
971         case MOVE_DOWN:
972             Log.e("SoDemo", "向下移动了一下");
973             break;
974         case MOVE_LEFT:
975             Log.e("SoDemo", "向左移动了一下");
976             break;
977         case MOVE_RIGHT:
978             Log.e("SoDemo", "向右移动了一下");
979             break;
980         default:
981             break;
982     }
983 }
984 /*第四步: IPCamD_Listener摄像机视频状态改变*/
985 public void on_video_status_changed(IPCam ipcam) {
986     PLAY_STATUS status = ipcam.video_status();
987 }
988

```

#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

```

994 /*
995  *IPCam类方法
996  *返回:
997  *返回PLAY_STATUS, 详细如下:
998  */
999 public PLAY_STATUS audio_status();
1000 public enum PLAY_STATUS {
1001     STOPPED, //停止
1002     REQUESTING, //请求中
1003     PLAYING; //播放中
1004 }

```

b) 音频质量

```

1007 /*
1008  *IPCam类方法
1009  *返回:
1010  *返回int类型, 真实大小, 详细如下:
1011  */
1012 public int audio_sps(); //音频的sps
1013 public int audio_byterate(); //音频的byte

```

c) 开启音频

```

1015 /*
1016  *IPCam类方法:开启音频
1017  *返回:
1018  *返回ERROR类型, 当返回类型是ERROR.NO_ERROR时, 表示开启音频成功
1019  */
1020 public ERROR play_audio();

```

d) 关闭音频

```

1023 /*
1024  *IPCam类方法:关闭音频

```



```

1025     */
1026 public void stop_audio();
1027
1028 e) 音频监听
1029 /*
1030  *IPCam类:IPCam_Listener接口
1031  *IPCam_Listener的详细用法请参考IPCam的基础监听
1032  */
1033 public void add_listener(IPCam_Listener listener); //设置IPCam_Listener接口
1034 public void on_audio_status_changed(IPCam ipcam);
1035 //IPCam_Listener回调函数, 音频状态改变时回调
1036 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:

```

1048 /*第一步: 添加IPCam_Listener接口*/
1049 m_ipcam.add_listener(this); //m_ipcam添加监听
1050 /*第二步: 音频操作 */
1051 case R.id.play_voice:
1052     {
1053         if (m_ipcam.audio_status() == PLAY_STATUS.PLAYING) { //判断获取音频状态
1054             m_ipcam.stop_audio(); //关闭音频
1055         } else {
1056             m_ipcam.play_audio(); //开启音频
1057         }
1058         /*音频质量如下*/
1059         Log.e("音频质量: ", m_ipcam.audio_sps() + "fps"); //音频的fps
1060         Log.e("音频质量: ", m_ipcam.audio_byterate() + "byte"); //音频的byte
1061     }
1062
1063 /*第二步: 回调函数 */
1064 public void on_audio_status_changed(IPCam ipcam) {
1065     PLAY_STATUS status = m_ipcam.audio_status(); //获取音频当前状态
1066 }

```

#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

```

1073 /*
1074  *IPCam类: 对讲状态获取
1075  *返回:
1076  *返回PLAY_STATUS, 详细状态如下:
1077  */
1078 public ERROR start_speak();
1079 public enum PLAY_STATUS {
1080     STOPPED, //停止
1081     REQUESTING, //请求中
1082     PLAYING; //播放中
1083 }

```

b) 对讲质量

```

1085 /*
1086  *IPCam类: 对讲状态获取
1087  *返回:
1088  *返回int类型, 真实大小, 详细如下:
1089  */
1090 public int speak_sps(); //对讲的fps
1091 public int speak_byterate() //对讲的byte

```

```

1092
1093 c) 开启对讲
1094 /*
1095  *IPCam类: 开启对讲
1096  *返回:
1097  *返回ERROR类型, 当返回类型是ERROR.NO_ERROR时, 表示开启对讲成功
1098  */
1099 public ERROR start_speak();
1100
1101 d) 关闭对讲
1102 /*
1103  *IPCam类: 关闭对讲
1104  */
1105 public void stop_speak();
1106
1107 e) 对讲接口
1108 /*
1109  *IPCam类:IPCamDelegate代理
1110  *IPCamDelegate的详细用法请参考IPCam的基础代理
1111  */
1112 public void add_listener(IPCam_Listener listener); //设置IPCam_Listener接口
1113 public void on_speak_status_changed(IPCam ipcam);
1114 //IPCam_Listener回调函数, 对讲状态改变时回调
1115 public void remove_listener(IPCam_Listener listener); //删除IPCam_Listener监听
1116
1117 4) 注意点:
1118 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
1119 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
1120 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。
1121
1122 5) 步骤:
1123 第一步: 添加接口IPCam_Listener, 这一步根据需要添加。<可选>
1124 第二步: 摄像机连接成功即可直接开启停止对讲等操作。
1125 第三步: 处理回调函数: 添加了接口才有回调函数。<可选>
1126
1127 6) 示例
1128 /*第一步: 添加IPCam_Listener接口*/
1129 m_ipcam.add_listener(this); //m_ipcam添加监听
1130 /*第二步: 对讲操作 */
1131 case R.id.play_talk:
1132     if (m_ipcam.speak_status() == PLAY_STATUS.PLAYING) //判断对讲状态
1133         m_ipcam.stop_speak();
1134     else
1135         m_ipcam.start_speak();
1136     /*对讲质量如下*/
1137     Log.e("对讲质量: ", m_ipcam.speak_sps() + "fbs"); //对讲的fps
1138     Log.e("对讲质量: ", m_ipcam.speak_byterate() + "byte"); //对讲的byte
1139 }
1140 /*第二步: 回调函数 */
1141 public void on_speak_status_changed(IPCam ipcam){
1142     PLAY_STATUS status = m_ipcam.speak_status(); //获取对讲当前状态
1143 }
1144
1145 #8.4 摄像机拍照
1146 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。
1147 2) 使用类: IPCam
1148 3) 函数:
1149 /*
1150  *IPCam方法: 拍照
1151  *返回:
1152  *返回ERROR类型, 当返回类型是ERROR.NO_ERROR时, 表示拍照成功。
1153  */
1154 函数1: public String snapshot();
1155 /*
1156  *IPCam方法: 拍照
1157  *参数:
1158  *photo_file_path: 照片存储的完整路径
1159  *返回:

```

```
1159 *返回ERROR类型，当返回类型是ERROR.NO_ERROR时，表示拍照成功。
1160 */
1161 public String snapshot(String snapshot_file_path);
1162
```

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

```
1166
1167 case R.id.play_photograph:
1168     String filepath = m_ipcam.snapshot(); //执行拍照
1169     if (filepath == null) {
1170         Log.e("SoDemo", "failed_snapshot");
1171     } else {
1172         Log.e("SoDemo", "snapshot succeed");
1173     }
1174 }
1175
```

#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

```
1181 /*
1182  *IPCam方法:本地录像状态
1183  *返回:
1184  *返回PLAY_STATUS, 具体如下:
1185  */
1186 public PLAY_STATUS local_record_status();
1187 public enum PLAY_STATUS {
1188     STOPPED, //停止
1189     REQUESTING, //请求中
1190     PLAYING; //播放中
1191 }
```

b) 开启本地录像

```
1194 /*
1195  *IPCam方法:开启本地录像
1196  *返回:
1197  *返回ERROR类型,当返回类型是ERROR.NO_ERROR时,表示本地录像成功。
1198  */
1199 函数1:public String start_local_record();
```

```
1200
1201 /*
1202  *IPCam方法:开启本地录像
1203  *参数:
1204  *record_file_path:录像存储的完整路径
1205  *返回:
1206  *返回ERROR类型,当返回类型是ERROR.NO_ERROR时,表示本地录像成功。
1207  */
```

```
1209 函数2:public String start_local_record(String record_file_path);
```

c) 停止本地录像

```
1212 /*
1213  *IPCam方法:停止本地录像
1214  */
1215 public void stop_local_record();
```

d) 本地录像监听

```
1218 /*
1219  *IPCam类:IPCam_Listener监听
1220  *IPCam_Listener的详细用法请参考IPCam的基础监听
1221  */
1222 public void add_listener(IPCam_Listener listener); //设置IPCam_Listener接口
1223 public void on_camera_recording_changed(IPCam ipcam);
1224 //IPCam_Listener回调函数,录像状态改变时回调
1225 public void remove_listener(IPCam_Listener listener); //删除IPCam_Listener监听
```

1226 4) 注意点:
1227 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。
1228
1229 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
1230 状态回调函数。如果没有添加齐全, 编译会报错
1231
1232 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
1233 需要使用时也要记得remove。
1234
1235 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
1236 放到系统相册, 需要再导出

1231 5) 步骤
1232 第一步: 添加代理IPCam Listener, 这一步根据需要添加。<可选>
1233 第二步: 摄像机视频出来后即可直接开启停止等操作。
1234 第三步: 处理回调函数: 添加了接口才有回调函数。<可选>

1236 6) 示例:
1237 /*第一步: 添加IPCam Listener接口*/
1238 m_ipcam.add_listener(this); //m_ipcam添加IPCam Listener
1239
1240 /*第二步: 录像操作 */
1241 case R.id.play_record:
1242 if (m_recording){
1243 m_ipcam.stop_local_record(); //停止录像
1244 m_recording = false;
1245 }else{
1246 m_local_record_filepath = m_ipcam.start_local_record(); //开启录像
1247 Log.e("sosocam record", "-----m_local_record_filepath-----" +
1248 m_local_record_filepath);
1249 if (m_local_record_filepath == null) {
1250 Log.e("SoDemo", "-camera--local_record--failed-");
1251 } else {
1252 m_recording = true;
1253 }
1254 }
1255
1256 /*第二步: 回调函数 */
1257 public void on_camera_recording_changed(IPCam ipcam) {
1258 PLAY_STATUS status = m_ipcam.local_record_status(); //获取本地录像当前状态
1259 }
1260

1261 #9. 摄像机PTZ操作<包括云台操作>

1262 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
1263 操作功能的实现。

1264 2) 使用类: IPCam

1265 3) 函数:

```
1266 /*  
1267 *IPCam方法: ptz控制命令  
1268 *参数:  
1269 *cmd:功能命令, 详细查看一下的IPCAM PTZ_CMD  
1270 *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>  
1271 */  
1272 public ERROR ptz_control(PTZ_CMD cmd, int param)  
1273  
1274 public enum PTZ_CMD {  
1275 UP, //向上  
1276 DOWN, //向下  
1277 LEFT, //向左  
1278 RIGHT, //向右  
1279 T_PATROL, //水平巡航  
1280 P_PATROL, //垂直巡航  
1281 TRACK_PATROL, //轨迹巡航  
1282 MEIJING_VOLUME_UP, //客户定制功能  
1283 MEIJING_VOLUME_DOWN, //客户定制功能  
1284 MEIJING_NEXT_TRACK, //客户定制功能  
1285 MEIJING_PREVIOUS_TRACK, //客户定制功能  
1286 MEIJING_PLAY_PAUSE, //客户定制功能  
1287 MEIJING_LED_POWER, //客户定制功能  
1288 MEIJING_LED_STYLE; //客户定制功能  
1289 }  
1290
```

1290 4) 示例:

```

1291 case R.id.play_cache_v:
1292     if (conn_status == CONN_STATUS.CONNECTED)
1293         m_ipcam.ptz_control(PTZ_CMD.T_PATROL, 20); //开启水平巡航
1294     break;
1295 case R.id.play_cache_v_stop:
1296     if (conn_status == CONN_STATUS.CONNECTED)
1297         m_ipcam.ptz_control(PTZ_CMD.T_PATROL, 0); //关闭水平巡航
1298     break;
1299

```

#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:

```

1306 /*
1307  *IPCam方法: CGI设置参数
1308  *参数:
1309  *params:cgi参数, 由reecam提供, 下面列举一些
1310  *params:set_params_listener监听, 可以监控cgi设置的结果
1311  */
1312 public ERROR set_params(String params, set_params_listener listener)
1313 /*其中arm_schedule有三个值, 0:布防, 1:计划布防, 2:撤防*/
1314 报警params: "save=1&reinit_alarm=1&arm_schedule=0"
1315 /*
1316  *record_schedule_list=[] 停止tf卡录像
1317  *record_schedule_list=[{"start":0,"end":96,"day":127}] 开启录像
1318  */
1319 录像params: "save=1&reinit_record=1&record_schedule_list=[]"
1320 重启params:"reboot=1"
1321 /*
1322  *set_params_listener回调函数
1323  *返回参数:
1324  *ipcam: 返回一个IPCam对象
1325  *ERROR类型, 当返回是ERROR.NO_ERROR, 表示设置成功
1326  */
1327
1328 public void on_result(IPCam ipcam, ERROR error);
1329

```

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

```

1332 /*第一步: cgi设置参数+添加监听set_params_listener
1333  */
1334 String params = "save=1&reinit_alarm=1&arm_schedule=1"; //计划布防cgi
1335 m_ipcam.set_params(params, this);
1336 m_state = STATE.SET_CAMERA;
1337 /*
1338  *IPCam.set_params_listener回调函数
1339  */
1340 public void on_result(IPCam ipcam, ERROR error) {
1341     ipcam.stop_connect();
1342     if (m_state == STATE.SET_CAMERA) {
1343         if (error == ERROR.NO_ERROR) {
1344             Log.e("SoDemo", "设置成功");
1345         } else {
1346             Log.e("SoDemo", "设置失败");
1347         }
1348     }
1349 }
1350

```

#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

1360 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。

1361 2) 注意:

1362 1) 这个html由reecam提供,需要的可以联系reecam

1363 3) 示例:

```
1364 public void set_wifi_from_web()
1365 {
1366     private WebView mwebView;
1367     String m_url = "file:///android_asset/app/wifi.html?" + "host="+ m_ipcam.ip() +
1368     "&port=" +
1369     m_ipcam.port() + "&user=" + user + "&pwd=" + pwd + "&ssl=" +
1370     (m_ipcam.https()?1:0) + "&sys=0" + "&lng=" + Tools.check_language();
1371     mwebView.loadUrl(m_url);
1372 }
```

11.2 APP设置无线

1373 说明: 无线网络设置包括,扫描周围wifi和设置摄像机wifi功能。

1374 使用类: ipcam

1375 函数:

1376 a) 扫描wifi

1377 注意:

1378 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

1380 要连接摄像机成功。

1381 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)。

```
1382 /*
1383  *扫描函数
1384  *返回:
1385  *返回ERROR类型,当返回是ERROR.NO_ERROR,表示wifi_scan执行成功
1386  */
1387 public ERROR wifi_scan(wifi_scan_listener listener);
1388
1389 /*
1390  *扫描结果函数回调
1391  *参数:
1392  *ipcam: 返回一个IPCam对象
1393  *error:ERROR类型,当返回是ERROR.NO_ERROR,表示scan成功
1394  *ap_list:一个列表,列表的每一个元素是ap_info类型,具体如下:
1395     public String bssid; // bssid wifi的唯一标示码,相当MAC地址
1396     public String ssid; // ssid wifi名称
1397     public WIFI_AUTH auth; //认证方式
1398     public WIFI_ENCRYPT encrypt; //加密方式
1399     public int rssi; //信号强度
1400 */
1401 public void on_result(IPCam ipcam, ERROR error, ArrayList<ap_info> ap_list);
1402 /*
1403  *认证方式有五种,如IPCAM_WIFI_AUTH所示
1404  */
1405 public enum WIFI_AUTH {
1406     OPEN, //无密码
1407     WEP, //WEP
1408     WPAPSK, //WPAPSK
1409     WPA2PSK, //WPA2PSK
1410     UNKNOWN; //unknown
1411 }
1412 /*
1413  *加密方式有五种,如IPCAM_WIFI_ENCRYPT所示
1414  */
1415 public enum WIFI_ENCRYPT {
1416     NONE,
1417     WEP,
1418     TKIP,
1419     AES,
1420     UNKNOWN;
1421 }
```

1422 b) 摄像机无线设置

1423 注意点:

1424 1) 函数1和函数2都可以实现设置无线功能。

1425

- 1427 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。
- 1428 3) 函数2没有无线测试功能,直接设置。
- 1429 4) 不管摄像机当前是有线连接还是无线连接都可以使用。
- 1430
- 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。

```
1431
1432 /*
1433 *参数:
1434 *ssid:wifi名称,通过扫描返回
1435 *key: wifi密码
1436 *auth: 认证方式,通过扫描返回
1437 *encrypt: 加密方式,通过扫描返回
1438
1439 *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
1440
1441 *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
1442 *listener:监听set_wifi_listener
1443 */
1444 函数1:
1445 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);
1446 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);
1447
1448 public enum WIFI_WEP_KEY_TYPE {
1449     HEX, //hex类型
1450     ASCII; //ASCII类型
1451 }
1452
1453 /*
1454 *set_wifi_listener回调函数 Wi-Fi设置进程
1455 *参数返回:
1456 *ipcam: 返回一个IPCam对象
1457 *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
1458 */
1459 public void on_progress(IPCam ipcam, SET_WIFI_STATE state);
1460
1461 public enum SET_WIFI_STATE {
1462     SETTING, //wifi设置中
1463     TESTING, //wifi测试中
1464     SAVING; //wifi保存中
1465 };
1466
1467 /*
1468 *set_wifi_listener回调函数 Wi-Fi设置结果
1469 *参数返回:
1470 *ipcam: 返回一个IPCam对象
1471
1472 *error: ERROR类型,当返回是ERROR.NO_ERROR,表示设置wifi成功,返回ERROR.DEVICE_OPERATI
    ON_FAIL表示密码错误,其他都是设置失败。
1473 */
1474 public void on_result(IPCam ipcam, ERROR error);
```

- 1475 步骤:
- 1476 第一步: 扫描wifi
- 1477 第二步: 选择wifi设置无线
- 1478 第三步: 监控无线设置过程和结果

```
1479
1480 3) 示例
1481 /*第一步: 扫描Wi-Fi*/
1482 public void scan_wifi() {
1483     /*扫描Wi-Fi+添加监听IPCam.wifi_scan_listener*/
1484     if (ERROR.NO_ERROR != m_ipcam.wifi_scan(this)) {
1485         Log.e("SoDemo", "SCAN_WIFI_FAIL");
1486     }
1487 }
```

```

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;
1525              //有加密方式，且key的长度是5或13时type为ASCII
1526          else
1527              type = WIFI_WEP_KEY_TYPE.HEX;//其他加密，type为hex
1528      }
1529      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*/
1530          if (ERROR.NO_ERROR == m_ipcam.set_wifi(m_ap.ssid, m_ap.auth, m_ap.encrypt,
1531              m_ap.wep_key_index, type, m_ap.key, this)) {
1532              m_state = STATE.SET_CAMERA;
1533              Log.e("SoDemo", "succeed_set_wifi");
1534          } else {
1535              Log.e("SoDemo", "failed_set_wifi");
1536          }
1537      }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*/
1538          if (ERROR.NO_ERROR == m_ipcam.set_wifi_without_testing(m_ap.ssid, m_ap.auth,
1539              m_ap.encrypt, m_ap.wep_key_index, type, m_ap.key, this)) {
1540              m_state = STATE.SET_CAMERA;
1541              Log.e("SoDemo", "succeed_set_wifi");
1542          } else {
1543              Log.e("SoDemo", "failed_set_wifi");
1544          }
1545      }
1546  }
1547
1548  /*第四步：设置wifi进度和结果监控，IPCam.wifi_scan_listener回调*/
1549  public void on_progress(IPCam ipcam, SET_WIFI_STATE state) {
1550      switch (state) {
1551          case SET_WIFI_STATE.SETTING: //wifi设置中
1552              m_state = SETTING_WIFI;
1553              break;
1554          case SET_WIFI_STATE.TESTING: //wifi测试中
1555              m_state = TESTING_WIFI;
1556              break;
1557          case SET_WIFI_STATE.SAVING: //wifi保存中
1558              m_state = SAVING_WIFI;
1559              break;
1560          default:
1561              break;
1562      }
1563  }

```



```

1558     }
1559 }
1560
1561 public void on_result(IPCam ipcam, ERROR error) {
1562     if (error == ERROR.NO_ERROR) {
1563         Log.e("SoDemo","wifi_setting_ok");
1564     } else if ((error == ERROR.DEVICE_OPERATION_FAIL) && (m_ap.auth !=
WIFI_AUTH.OPEN)) {
1565         Log.e("SoDemo","failed_join_wifi,wifi key error");
1566         m_state = STATE.ENTER_AP_KEY;
1567     } else {
1568         Log.e("SoDemo","failed_set_wifi");
1569     }
1570 }
1571

```

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

```

1591 /*
1592 *SoundWaveWifiSetting开始无线设置
1593 * 参数如下：
1594 * ssid:wifi的ssid
1595 * psk:wifi的密码
1596 * listener:接口，当前对象需实现sound_wave_wifi_setting_listener接口
1597 * 返回：
1598 * 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。
1599 */

```

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)

```

1604 /*
1605 *这是接口sound_wave_wifi_setting_listener的回调函数
1606 *开始声音设时，添加了此接口
1607 *只要声音设置无线状态有改变时候就会调用此函数
1608 */
1609 回调函数：public void on_sound_wave_wifi_setting_state_changed();

```

b) 停止声音设置无线

```

1612 /*
1613 *SoundWaveWifiSetting停止声音设置无线
1614 *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
1615 *当退出设置或者smartlink先设置成功就需要主动停止sound设置
1616 */
1617 public void stop();

```

c) 声音设置无线信息获取：

```

1620 /*

```

```

1621 *声音设置无线结果，共有两种结果，详细如下：
1622 *声音设置结果通过public SOUND_WAVE_WIFI_SETTING_RESULT result()获取
1623 */
1624 public static enum SOUND_WAVE_WIFI_SETTING_RESULT {
1625     SUCCEED, //设置成功
1626     TIMEOUT //设置超时，需重新设置
1627 }
1628
1629 /*
1630 *声音设置无线状态，共有三种状态，详细如下：
1631 *状态通过public SOUND_WAVE_WIFI_SETTING_STATE state()
1632 */
1633 public static enum SOUND_WAVE_WIFI_SETTING_STATE {
1634     IDLE, //状态为空时，可查询当前设置结果，进行不同操作
1635     SEND_SETTING, //正在发送wifi设置信息
1636     WAIT_CONFIRM //等待验证，即等待设备响应
1637 };
1638
1639 d) 查询ID是否在局域网里
1640 /*
1641
1642 *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
1643 *特征码1和特征码2说明：
1644 *
1645 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
1646 到；
1647 *
1648 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
1649 过Tools.get_serial_of_camera_id(id)的方法得到一个特征码m2；
1650 *
1651 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
1652 等则此ID即为当前声波设置wifi成功的机器；
1653 */
1654 /*
1655 *SoundWaveWifiSetting&Tools编码id
1656 *id太长，分成两部分编码，比较的时候
1657 *SoundWaveWifiSetting的name_of_camera_id和Tools的get_name_of_camera_id比较
1658 *SoundWaveWifiSetting的serial_of_camera_id和Tools的get_serial_of_camera_id比较
1659 */
1660 public int name_of_camera_id(); //SoundWaveWifiSetting设置成功，返回编码的id
1661
1662 public int serial_of_camera_id(); //SoundWaveWifiSetting设置成功，返回编码的id
1663
1664 public static int get_name_of_camera_id(String id); //Tools编码id
1665
1666 public static int get_serial_of_camera_id(String id); //Tools编码id
1667
1668 5) 步骤：
1669 第一步：创建SoundWaveWifiSetting对象。
1670 第二步：开始sound无线设置
1671 第三步：实时监控sound的无线设置状态和结果
1672 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
1673 是否再局域网了
1674
1675 6) 示例：
1676 /*
1677 第一步：创建一个SoundWaveWifiSetting对象
1678 */
1679 private SoundWaveWifiSetting m_sound_wave_wifi_setting = new SoundWaveWifiSetting();
1680 m_sound_wave_wifi_setting.init(m_parent);
1681 /*
1682 *第二步：发送声音 实现接口sound_wave_wifi_setting_listener
1683 */
1684 boolean m_sound_wifi = m_sound_wave_wifi_setting.start(m_ap.ssid, m_ap.key,
1685 AddCamerabySoundDialog.this);
1686 if (!m_sound_wifi){
1687     Log.e("SoDemo", "SEND_WIFI_INFO_FAIL");
1688 }
1689 /*
1690 *监测 声音设置无线的过程

```

```

1684  */
1685  public void on_sound_wave_wifi_setting_state_changed() {
1686      SOUND_WAVE_WIFI_SETTING_STATE state = m_sound_wave_wifi_setting.state();
1687      SOUND_WAVE_WIFI_SETTING_RESULT result = m_sound_wave_wifi_setting.result();
1688      Log.e("SoDemo", "SOUND_WAVE_WIFI_SETTING_STATE = " + state
1689          + "SOUND_WAVE_WIFI_SETTING_RESULT = " + result);
1689      if (state == SOUND_WAVE_WIFI_SETTING_STATE.SEND_SETTING) {
1690          Log.e("SoDemo", "Transmitting wireless setting now");//正在传送无线设置
1691          ...;
1692      } else if (state == SOUND_WAVE_WIFI_SETTING_STATE.WAIT_CONFIRM) {
1693          Log.e("SoDemo", "Wait device response");//正在等待设备响应 ...;
1694      } else if (state == SOUND_WAVE_WIFI_SETTING_STATE.IDLE) {
1695          if (result == SOUND_WAVE_WIFI_SETTING_RESULT.SUCCEED) {
1696              m_name_of_camera_id = m_sound_wave_wifi_setting.name_of_camera_id();
1697              m_serial_of_camera_id = m_sound_wave_wifi_setting.serial_of_camera_id();
1698              Log.e("SoDemo", "Search camera in LAN");//设置成功可在局域网开始搜索此机器
1699              discover_camera();
1700          }
1701      }
1702  }
1703  /*
1704  *第三步：设置成功后，查询id是否在局域网内,来获取id
1705  */
1706  public void discover_camera(){
1707      LinkedHashMap<String, DISCOVERED_CAMERA_INFO> cameras =
1708      IPCamMgr.get_discovered_cameras_list();
1709      String camera_id = null;
1710      int name_of_camera_id;
1711      int serial_of_camera_id;
1712
1713      for (String id : cameras.keySet()) {
1714          /*对局域网的摄像机id进行编码*/
1715          name_of_camera_id = Tools.get_name_of_camera_id(id);
1716          serial_of_camera_id = Tools.get_serial_of_camera_id(id);
1717          /*查id 是否在局域网内*/
1718          if (name_of_camera_id == m_name_of_camera_id &&
1719              serial_of_camera_id == m_serial_of_camera_id) {
1720              camera_id = id;
1721              Log.e("SoDemo", "Camera id is" + id);
1722              break;
1723          }
1724      }
1725      /*
1726      *不需要了，强制取消SoundWaveWifisetting设置。设置成功了，不需要做停止的操作
1727      */
1728      m_sound_wave_wifi_setting.stop();
1729      m_sound_wave_wifi_setting.deinit();
1730
1731  #11.2.2.smartlink设置无线网络
1732
1733  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
1734  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。
1735
1736  使用的类：smartlink
1737  注意点：
1738  a) SDK初始化成功
1739  b) 系统固件版本为： "^\\d+\\.\\d+\\.\\d+\\.4\\.\\d+$"
1740  c) 书写了设备id,且id前缀为"RTEST-", "SOSO-", "RCAM-", "LSCAM-", "YSTC-"
1741
1742  函数：
1743  /*
1744  *开始smartlink设置
1745  *参数如下：
1746  *ssid:wifi的ssid
1747  *psk:wifi的密码
1748  *listener:wifi_matching_listener接口
1749  * 返回：
1750  * 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。
1751  */
1752  public static boolean start(String ssid, String psk, wifi_matching_listener listener)

```

```

1752  /*
1753  *这是接口wifi_matching_listener的回调函数
1754  *开始smartlink设置时时，实现此接口
1755  *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
1756  */
1757  public void on_wifi_matching_succeed(String camera_id);
1758
1759  /*
1760  *smartlink停止设置无线
1761  *smartlink设置成功自动就停止了设置，不需要主动去设置停止
1762  *当退出设置或者SoundWaveWifiSetting先设置成功就需要主动停止sound设置
1763  */
1764  public static void stop();
1765

```

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:

```

1771  /*
1772  *第一步: 创建一个WiFiMatching对象
1773  */
1774  private WiFiMatching m_smart_link_wifi_setting = new WiFiMatching();
1775
1776  /*
1777  *第二步: 发送smartlink 实现接口wifi_matching_listener
1778  */
1779  boolean m_smart_link = m_smart_link_wifi_setting.start(m_ap.ssid, m_ap.key,
1780  AddCamerabySmartDialog.this);;
1781  if (! m_smart_link){
1782  Log.e("SoDemo", "Smart Link start OK!");
1783  }
1784
1785  /*
1786  *第三步: wifi_matching_listener回调函数
1787  */
1788  public void on_wifi_matching_succeed(String camera_id){
1789  if (camera_id != "") {
1790  /*设置成功获取到id,可以进行下一步操作，比如添加摄像机*/
1791  Log.e("SeDemo", "smartlink设置成功, camera_id= " + camera_id);
1792  }
1793  }
1794
1795  /*
1796  *第四步: 不需要了，强制取消smartlink设置，设置成功了，不需要做停止的操作
1797  */
1798  m_smart_link_wifi_setting.stop();
1799

```

#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

```

1807  /*
1808  *storage方法:获取本地录像和图片
1809  *参数:
1810  *camera id:摄像机id
1811  *返回:返回一个list，列表的每一个元素是一个ALBUM_ITEM对象，具体如下:
1812  *Date t      //时间Date
1813  *long size   // 文件大小
1814  *boolean image //文件类型，两个值，true:图片 false:录像
1815  *String path  //文件路径
1816  */
1817  public static ArrayList<ALBUM_ITEM> get_local_records_list(String camera_id)
1818

```

示例1:

```

1822 Storage.get_local_records_list(cam.id());
1823
1824 b) 删除文件
1825 /*
1826  *storage方法：删除指定摄像机的所有的录像和拍照文件
1827  *参数：camera_id:摄像机id
1828  */
1829 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
1830
1831 /*
1832  *storage方法：删除指定摄像机的指定的录像或拍照文件
1833  *参数：path:文件路径
1834  */
1835 public static void delete_local_records_list_file(String path);
//删除某一个文件（图片或录像文件）
1836
1837 示例2:
1838 Storage.delete_local_records_list("RTEST-001015-DUGMR");//删除所有文件
1839 Storage.delete_local_records_list_file(path);//删除当前路径对应的一个文件
1840
1841 c) 导出文件
1842 /*
1843  *Tools方法：把图片导出到系统相册
1844  *参数:
1845  *path:图片/录像路径
1846  */
1847 public static void add_media(Context context, String path)//导出图片/录像到系统相册
1848
1849 示例3:
1849 Tools.add_media(LiveActivity.this, filepath);
1850
1851 #13 推送的实现
1852 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。
1853 #pragma mark --13.1以账号为id来settag(使用了平台功能)
1854 13.1以账号为id来settag
1855
1856 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。
1857 2) 使用的类：JPushInterface [jpush的类]
1858 3) 函数：
1859 /*
1860  *JPushInterface方法
1861  *此接口为jpush提供，不是瑞彩提供
1862  *每一个参数说明请查阅jpush官方文档
1863  *此方法的tags为账号登陆成功后，服务器返回的账号的“user_id”
1864  */
1865 public static void setAliasAndTags(Context context,
1866                                     String alias,
1867                                     Set<String> tags,
1868                                     TagAliasCallback callback)
1869
1870 4) 步骤:
1871 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。
1872 第二步：客户端实现推送。
1873
1874 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”
1875
1876 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]
1877
1878 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]
1879
1880 6) 示例:
1881 /*这里只介绍setTags部分，其他部分请参考jpush官方文档*/

```

```

1881 this.setTag(getApplicationContext(),"user_id");//注册账号，需要推送
1882
1883 public static void setTag(Context context,String tag){
1884     Log.e("sosocam", "set tag: " + tag);
1885     m_context = context;
1886     m_tagSet.clear();
1887     if (! tag.equals(""))
1888         m_tagSet.add(tag);
1889     JPushInterface.setAliasAndTags(context, null, m_tagSet, mTagsCallback);
1890 }
1891 private final static TagAliasCallback mTagsCallback = new TagAliasCallback() {
1892
1893     @Override
1894     public void getResult(int code, String alias, Set<String> tags) {
1895         switch (code) {
1896             case 0:
1897                 Log.e("sosocam", "set tag result: 0");
1898                 break;
1899             case 6002:
1900                 Log.e("sosocam", "set tag result: 6002");
1901                 JPushInterface.setAliasAndTags(m_context, null, m_tagSet,
1902                     mTagsCallback);
1903                 break;
1904             default:
1905                 Log.e("sosocam", "set tag result: " + code);
1906                 break;
1907         }
1908     }
1909 };

```

1910 #13.2以摄像机为id来settag(未使用平台功能)

1911 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。

1912 2) 使用类：Tools和JPushInterface [jpush的类]

1913 3) 函数：

```

1914 /*
1915  *JPushInterface方法
1916  *此接口为jpush提供，不是瑞彩提供
1917  *每一个参数说明请查阅jpush官方文档
1918  *此方法的tags为摄像机id，id需要编码
1919  */
1920 public static void setAliasAndTags(Context context,
1921     String alias,
1922     Set<String> tags,
1923     TagAliasCallback callback)
1924 /*
1925  *Tools方法
1926  *对摄像机id进行编码
1927  */
1928 public static String replaceExceptonChar(String s)
1929

```

1930 4) 步骤：

1931 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。
1932 第二步：客户端实现推送。

1933 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。

1934 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正确的，具体的请联系瑞彩

1935 5) 示例：

1936 /*这里只介绍setTags部分，其他部分请参考jpush官方文档*/

```

1937 LinkedHashMap<String, IPCam> ipcam_list = null;
1938 ipcam_list = IPCamMgr.get_cameras_list();
1939 public static void setTag(Context context,LinkedHashMap<String, IPCam> ipcam_list){
1940     m_context = context;
1941     Log.e("sosocam", "context is " + context);
1942     m_tagSet.clear();
1943     for (IPCam ipcam : ipcam_list.values())
1944     {

```

```

1947         Log.e("sosocam", "set tag:id----->>>>:: " + ipcam.id());
1948         m_tagSet.add(Tools.replaceExceptonChar(ipcam.id()));
1949     }
1950     JPushInterface.setAliasAndTags(context, null, m_tagSet,
        mTagsCallback);//注册所有摄像机，需要推送
1951 }
1952 JPushInterface.stopPush(context);//不需要推送,停止推送
1953
1954 private final static TagAliasCallback mTagsCallback = new TagAliasCallback() {
1955
1956     @Override
1957     public void getResult(int code, String alias, Set<String> tags) {
1958         switch (code) {
1959             case 0:
1960                 Log.e("sosocam", "set tag result: 0");
1961                 break;
1962             case 6002:
1963                 Log.e("sosocam", "" + JPushInterface.isPushStopped(m_context) + ","
                    + JPushInterface.getConnectionState(m_context));
1964
1965                 Log.e("sosocam", "set tag result: 6002");
1966                 JPushInterface.setAliasAndTags(m_context, null, m_tagSet,
                    mTagsCallback);
1967                 break;
1968             default:
1969                 Log.e("sosocam", "set tag result: " + code);
1970                 break;
1971         }
1972     }
1973
1974 };

```

#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：

```

1988 /*
1989  *WebService方法:获取当前账号所有摄像机的报警信息
1990  *返回：ERROR类型error值，error == ERROR.NO_ERROR表示获取成功；
1991  */
1992 public static void get_cameras_alarm_list(get_cameras_alarm_list_listener listener);
1993
1994 /*
1995  *WebService方法:get_cameras_alarm_list_listener回调
1996  *回调参数：
1997  *error:ERROR类型，error == ERROR.NO_ERROR表示获取成功
1998  *另一个是一个list,每一个元素都是CAMERA_ALARM_INFO对象，具体如下：
1999  *public String obj_id = "";//摄像机sosocam_id
2000  *public String id = ""; //摄像机id
2001  *public int unread = 0; //未读报警数目
2002  *public int total = 0; //报警总数目
2003  */
2004 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：

```

2013  /*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*/
2014  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

2015
2016  /*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*/
2017  public void on_get_cameras_alarm_list_result(ERROR error,
      ArrayList<CAMERA_ALARM_INFO> svr_alarm_list) {
2018      if (error == ERROR.NO_ERROR) {
2019          for (CAMERA_ALARM_INFO local_alarm : alarm_list) {
2020              for (CAMERA_ALARM_INFO svr_alarm : svr_alarm_list) {
2021                  if (local_alarm.id.equals(svr_alarm.id)) {
2022                      local_alarm.obj_id = svr_alarm.obj_id;
2023                      local_alarm.total = svr_alarm.total;
2024                      local_alarm.unread = svr_alarm.unread;
2025                  }
2026              }
2027          }
2028      }
2029  }
2030

```

#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

```

2036  /*
2037   *WebService方法:获取摄像机详细报警信息
2038   *参数:
2039   *obj_id: 摄像机sosocam平台id
2040   *listener:get_alarm_detail_list_listener回调
2041   */
2042  public static void get_alarm_detail_list(String obj_id,
      get_alarm_detail_list_listener listener);

2043
2044  /*
2045   *WebService方法:get_alarm_detail_list_listener回调
2046   *回调参数:
2047   *error:ERROR类型，error == ERROR.NO_ERROR表示获取成功
2048   *另一个是一个list,每一个元素都是ALARM_DETAIL对象，具体如下:
2049   *public String aid;//摄像机报警id
2050   *public int alarm_type; //摄像机报警类型
2051   *public int image_numbers; //报警图片数量
2052   *public boolean read;//是否读过
2053   *public Date date;//报警时间
2054   *public String thumb_url;//预览的url地址
2055   */
2056  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:

```

2066  /*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*/
2067  WebService.get_alarm_detail_list(activity.m_obj_id, activity);
2068  /*第二步:get_alarm_detail_list_listener回调*/
2069  public void on_get_alarm_detail_list_result(ERROR error, ArrayList<ALARM_DETAIL>
      alarm_detail_list){
2070      if (error == ERROR.NO_ERROR){
2071          /*在此处把改摄像机的每一条包报警信息取出*/
2072      }
2073  }
2074

```

#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


```

2078 3) 函数:
2079 /*
2080 *WebService方法:获取某一条报警的图片信息
2081 *参数:
2082 *aid: 报警id
2083 *listener:get_alarm_image_info_list_listener监听
2084 */
2085 public static ERROR get_alarm_image_info_list(String aid,
get_alarm_image_info_list_listener listener);
2086
2087 /*
2088 *WebService方法:get_alarm_image_info_list_listener回调
2089 *回调参数:
2090 *error:ERROR类型, error == ERROR.NO_ERROR表示获取成功
2091 *另一个是list, 每一个元素都是ALARM_IMAGE_INFO对象, 具体如下:
2092 *public Date date;//报警时间
2093 *public String url;//每张图片所代表的url地址
2094 */
2095 public void on_get_alarm_image_info_list_result(ERROR error,
ArrayList<ALARM_IMAGE_INFO> alarm_image_info_list);
2096
2097 /*
2098 *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
2099 */
2100 public static void cancel_get_alarm_image_info_list_tasks();
2101
2102 /*
2103 *WebService方法:获取单个摄像机的其中一条报警信息的图片
2104 *参数:
2105 *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
2106 *url:报警信息的图片URL
2107 *listener:get_alarm_image_listener
2108 */
2109 public static void get_alarm_image(String image_id, String url,
get_alarm_image_listener listener)
2110
2111 /*
2112 *WebService方法:get_alarm_image_listener回调
2113 *回调参数:
2114 *image id:当前照片的index值
2115 *data:获取到的图片data数据
2116 */
2117 public void on_get_alarm_image_result(String image_id, byte [] data);
2118
2119 /*
2120 *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
2121 */
2122 public static void cancel_get_alarm_image_tasks()
2123
2124 4) 步骤:
2125 第一步: 通过get_alarm_image_info_list获取图片的时间和url
2126 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
2127
2128 5) 注意:
2129 a) get_alarm_image_info_list需要的参数alarm_id是通过get_alarm_detail_list获取到的。
2130
2131 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_n
umbers范围内
2132
2133 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ca
ncel_get_alarm_image_info_list_tasks()和cancel_get_alarm_image_tasks()
2134
2135 6) 示例:
2136 /*第一步: 通过get_alarm_images_list获取图片的时间和url*/
2137 if (ERROR.NO_ERROR == Webservice.get_alarm_image_info_list(m_aid, this))
2138     Log.e("SoDemo", "download_alarm_image_list_now");
2139 else
2140     Log.e("SoDemo", "download_alarm_image_list_failed");
2141
2142 /*第二步: get_alarm_image_info_list_listener回调+get_alarm_image*/
2143 public void on_get_alarm_image_info_list_result(ERROR error,
ArrayList<ALARM_IMAGE_INFO> alarm_image_info_list) {

```

```

2142     if (ERROR.NO_ERROR == error) {
2143         for (ALARM_IMAGE_INFO info : alarm_image_info_list) {
2144             Log.e("SoDemo", "处理每张图片")
2145             WebService.get_alarm_image("'" + alarm_image_info_list.indexOf(info) +
                ".jpg", info.url, this);
2146         }
2147         Log.e("SoDemo", "download_alarm_image_list_succeed")
2148     } else {
2149         Log.e("SoDemo", "download_alarm_image_list_failed");
2150     }
2151 }
2152
2153 /*第三步: WebServiceGetImageDelegate回调*/
2154 public void on_get_alarm_image_result(String image_id, byte [] data) {
2155     if (data != null) {
2156         Bitmap bitmap = data;
2157         Storage.write_jpg_file(data, Storage.get_alarm_folder(m_id, m_aid) + "/" +
            filename);
2158     }
2159 }
2160
2161 /*第四步:不需要的时候要手动取消*/
2162 WebService.cancel_get_alarm_image_info_list_tasks();
2163 WebService.cancel_get_alarm_image_tasks();
2164
2165
2166 #16.分享和收藏
2167 #16.1开启分享
2168 16.1开启分享
2169 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。
2170 2) 使用的类: WebService, IPCam
2171 3) 函数:
2172 /*
2173 *WebService方法: 开启分享
2174 *参数:
2175 *obj_id:摄像机sosocam平台的id
2176 *listener:share_camera_listener监听
2177 */
2178 public static ERROR share_camera(String obj_id, share_camera_listener listener)
2179
2180 /*
2181 *WebService方法: WebServiceShareCameraDelegate回调
2182 *参数:
2183 *result;//sosocam平台返回的分享结果ERROR值
2184 *share_id;//分享码的id
2185 *share_url;//分享码的url地址
2186 *share_qrcode;//分享码的二维码地址
2187 */
2188 public void on_share_camera_result(ERROR result, String share_id, String share_url,
String share_qrcode);
2189
2190 /*
2191 *IPCam方法: CGI设置参数
2192 *参数:
2193 *params:cgi参数, 由reecam提供, 开启分享是"save=1&reinit_guest=1&group2=1"
2194 *listener:IPCam.set_params_listener监听, 可以监控cgi设置的结果
2195 */
2196 public ERROR set_params(String params, set_params_listener listener)
2197
2198 /*
2199 *IPCam方法: IPCam.set_params_listener回调函数
2200 *返回参数:
2201 *ipcam: 返回一个IPCam对象
2202 *ERROR类型, 当返回是ERROR.NO_ERROR, 表示设置成功
2203 */
2204 public void on_result(IPCam ipcam, ERROR error);
2205
2206 4) 步骤:
2207 第一步: 开启分享, 添加接口share_camera_listener。
2208 第二步: 开启分享成果后, 用cgi设置权限, 且添加监听set_params_listener。
2209 第三步: set_params_listener回调
2210

```

2211 5) 注意: a) 开启分享音频也是用set_params, params为"save=1&reinit_guest=1&group2=9"
2212 b) 关闭分享音频也是用set_params, params为"save=1&reinit_guest=1&group2=1"
2213 c) 设置分享的密码也是用set_params, params为"save=1&reinit_guest=1&pwd2=%@"
2214 d) 这些设置必须是以摄像机连接成功为前提

2215
2216 6) 示例:

```
2217 /*第一步:开启分享, 添加监听share_camera_listener*/  
2218     if(ERROR.NO_ERROR != Webservice.share_camera(ipcam_sosocam_id, this)){  
2219         /*开启分享失败*/  
2220     }  
2221  
2222 /*第二步: 开启分享成功后, 用cgi设置权限, 且添加监听set_params_listener. */  
2223 public void on_share_camera_result(ERROR result, String share_id,String share_url,  
2224 String share_qrcode) {  
2225     if(result == ERROR.NO_ERROR){  
2226         String params = "save=1&reinit_guest=1&group2=1";  
2227         if(ERROR.NO_ERROR != m_ipcam.set_params(params, this)){  
2228             /*开启分享失败*/  
2229         }  
2230     }else{  
2231         /*开启分享失败*/  
2232     }  
2233 }  
2234 /*第三步: set_params_listener回调函数*/  
2235 public void on_result(IPCam ipcam, ERROR error) {  
2236     ipcam.stop_connect();  
2237     if (m_state == STATE.SET_CAMERA) {  
2238         if (error == ERROR.NO_ERROR) {  
2239             Log.e("SoDemo", "设置成功");  
2240         } else {  
2241             Log.e("SoDemo", "设置失败");  
2242         }  
2243     }  
2244 }
```

2245 #16.2关闭分享

2247 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。

2248 2) 使用的类: Webservice

2249
2250 3) 函数:

```
2251 /*  
2252 *WebService方法: 关闭分享  
2253 *参数:  
2254 *obj_id:摄像机sosocam平台的id  
2255 *listener: 监听disshare_camera_listener  
2256 */  
2257 public static ERROR disshare_camera(String obj_id, disshare_camera_listener listener)  
2258  
2259 /*  
2260 *WebService方法: disshare_camera_listener回调  
2261 *参数:  
2262 *error:错误值, 当错误值为ERROR.NO_ERROR时, 说明开启分享成功  
2263 */  
2264 public void on_disshare_camera_result(ERROR result);  
2265  
2266 /*  
2267 *IPCam方法: CGI设置参数  
2268 *参数:  
2269 *params:cgi参数, 由reecam提供, 开启分享是"save=1&reinit_guest=1&group2=1"  
2270 *listener:IPCam.set_params_listener监听, 可以监控cgi设置的结果  
2271 */  
2272 public ERROR set_params(String params, set_params_listener listener)  
2273  
2274 /*  
2275 *IPCam方法: IPCam.set_params_listener回调函数  
2276 *返回参数:  
2277 *ipcam: 返回一个IPCam对象  
2278 *ERROR类型, 当返回是ERROR.NO_ERROR, 表示设置成功  
2279 */  
2280 public void on_result(IPCam ipcam, ERROR error);  
2281
```

2282 4) 步骤
2283 第一步: 关闭分享, 添加监听disshare_camera_listener。
2284 第二步: 关闭分享成果后, 用cgi设置权限, 且添加监听set_params_listener。
2285 第三步: set_params_listener回调

2286
2287 5) 注意: a) 这些设置必须是以摄像机连接成功为前提

2288
2289 6) 示例:

```
2290  
2291 public void disshare_camera(){  
2292     if (WebService.ERROR.NO_ERROR != WebService.disshare_camera(m_ipcam.sosocam_id,  
2293         this)){  
2294         /*关闭分享失败*/  
2295     }  
2296 }  
2297 public void on_disshare_camera_result(com.sosocam.webservice.WebService.ERROR  
2298 result) {  
2299     if(result == com.sosocam.webservice.WebService.ERROR.NO_ERROR){  
2300         if(ERROR.NO_ERROR != m_ipcam.set_params("save=1&reinit_guest=1&group2=" +  
2301             m_group, this)){  
2302             /*关闭分享失败*/  
2303         }  
2304     }else{  
2305         /*关闭分享失败*/  
2306     }  
2307 }  
2308 public void on_result(IPCam ipcam, ERROR error) {  
2309     ipcam.stop_connect();  
2310     if (m_state == STATE.SET_CAMERA) {  
2311         if (error == ERROR.NO_ERROR) {  
2312             Log.e("SoDemo", "设置成功");  
2313         } else {  
2314             Log.e("SoDemo", "设置失败");  
2315         }  
2316     }  
2317 }
```

2318 #16.3分享状态获取

2319 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

2320 2) 使用类: IPCam

2321 3) 函数:

```
2322 /*  
2323 *IPCam方法: CGI获取参数  
2324 *参数:  
2325 *params:cgi参数, 由reecam提供  
2326 *listener:get_params_listener监听, 可以监控cgi获取的结果  
2327 *返回:  
2328 *返回ERROR类型, 当返回ERROR.NO_ERROR, 则表示函数执行成功  
2329 */  
2330 public ERROR get_params(String params, get_params_listener listener);  
2331  
2332 /*  
2333 *IPCam方法: get_params_listener回调  
2334 *参数:  
2335 *ipcam: 返回一个IPCam对象  
2336 *error:ERROR类型, 当返回是ERROR.NO_ERROR, 表示获取参数成功  
2337 *params: 一个JSONObject;  
2338 */  
2339 public void on_result(IPCam ipcam, ERROR error, JSONObject json);
```

2340 4) 步骤:

2341 第一步: 获取参数, 且添加监听get_params_listener

2342 第二步: 根据get_params_listener结果判断状态

2343 5) 注意: a) 这些设置是以摄像机连接成功为前提

2344 b) 摄像机分享信息状态的获取, params为"group2=&pwd2="

2345
2346 6) 示例:

```
2347 /*第一步: 获取参数+添加监听get_params_listener*/  
2348 -(void)get_share_info{  
2349     if(ERROR.NO_ERROR != m_ipcam.get_params("group2=&pwd2=", this)){  
2350         /*获取分享信息失败*/
```

```

2351     }
2352 }
2353 /*第二步: 根据get_params_listener结果判断状态*/
2354 public void on_result(IPCam ipcam, ERROR error, JSONObject json) {
2355     String pwd = null;
2356     if(error == ERROR.NO_ERROR){
2357         /*获取分享信息成功*/
2358         try {
2359             cur_auth = json.getInt("group2");
2360             pwd = json.getString("pwd2");
2361         } catch (JSONException e) {
2362             /*摄像机未连接*/
2363         }
2364         if(pwd != null && ! pwd.equals(""))
2365             /*分享密码: 有密码*/
2366         if(0 >= cur_auth){
2367             cur_auth = 0;
2368             /*分享状态为: 关闭*/
2369         }else{
2370             /*分享状态为: 开启*/
2371             if(((cur_auth >> 3) & 0x01) == 1){
2372                 /*分享音频状态为: 开*/
2373             }else{
2374                 /*分享音频状态为: 关闭*/
2375             }
2376         }
2377     }else{
2378         /*获取分享信息失败*/
2379     }
2380 }

```

#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:

```

2386 /*
2387  *Storage方法: 添加收藏到本地
2388  *参数:
2389  *列表的每一个元素是一个COLLECTION_INFO类型的对象, 具体如下:
2390  *String collection_id 分享id
2391  *boolean valid 当前camera的收藏是否有效,
2392  * true: 有效可继续观看
2393  * 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
2394  *String alias 昵称
2395  *String user 登录名
2396  *String pwd 登录密码
2397  *String cover_url 预览图url
2398  */
2399 public static void add_collection(COLLECTION_INFO collection_info);
2400
2401 /*
2402  *WebService方法:获取收藏摄像机信息
2403  *参数:
2404  *share_id: 分享id
2405  *listener:get_collection_listener监听
2406  */
2407 public static void get_collection(String share_id, get_collection_listener listener)
2408
2409 /*
2410  *WebService方法:get_collection_listener回调
2411  *参数:
2412  *error:错误值, 当错误值为ERROR.NO_ERROR时, 说明获取收藏成功
2413  *alias:摄像机别名
2414  *cover_url: 摄像机预览图
2415  */
2416 public void on_get_collection_result(ERROR result, String alias, String cover_url);
2417
2418 /*
2419  *WebService方法:账号添加分享的摄像机: 即添加到服务器和本地
2420  *参数:
2421  *share_id:摄像机分享id号

```

```

2422 *listener: 监听add_collection_listener
2423 */
2424 public static ERROR add_collection(String share_id, add_collection_listener listener)
2425
2426 /*
2427 *WebService方法:add_collection_listener回调
2428 *参数:
2429 *error:错误值, 当错误值为ERROR.NO_ERROR时, 说明开启添加收藏成功
2430 *alias:摄像机别名
2431 *cover_url: 摄像机预览图
2432 */
2433 public void on_add_collection_result(ERROR result, String alias, String cover_url);
2434

```

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()

```

2450 {
2451     if (ERROR.NO_ERROR != WebService.add_collection(m_Share_id, this)) {
2452         /*添加失败*/
2453     }
2454 }
2455 /*第二步: add_collection_listener监听回调*/
2456 public void on_add_collection_result(ERROR result, String alias,String cover_url) {
2457     if(result == ERROR.NO_ERROR){
2458         /*添加到服务器成功, 然后添加到本地*/
2459         COLLECTION_INFO collection = new COLLECTION_INFO();
2460         collection.collection_id = m_Share_id;
2461         collection.alias = alias;
2462         collection.cover_url = cover_url;
2463         Storage.add_collection(collection);
2464     } else if(result == ERROR.INVALID_SHARE_ID){
2465         /*分享id无效*/
2466     }else{
2467         /*添加到服务器失败*/
2468     }
2469 }

```

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()

```

2474 {
2475     if (ERROR.NO_ERROR != WebService.get_collection(m_Share_id, this)) {
2476         /*获取失败*/
2477     }
2478 }
2479
2480 /*第二步: get_collection_listener回调*/
2481 public void on_get_collection_result(ERROR result, String alias,String cover_url) {
2482     if(result == ERROR.NO_ERROR){
2483         /*获取收藏摄像机信息成功, 然后添加到本地*/
2484         COLLECTION_INFO collection = new COLLECTION_INFO();
2485         collection.collection_id = m_Share_id;
2486         collection.alias = alias;
2487         collection.cover_url = cover_url;
2488         /*添加到本地*/
2489         Storage.add_collection(collection);
2490     } else if(result == ERROR.INVALID_SHARE_ID){
2491         /*分享id无效*/
2492     } else{
2493         /*获取收藏摄像机信息失败*/
2494     }

```

```

2495     }
2496
2497
2498 #16.5连接收藏摄像机（一次性连接）
2499 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。
2500 2) 使用的类： IPCamMgr
2501 3) 函数：
2502 /*
2503  *IPCamMgr方法：获取一次性摄像机
2504  *返回：
2505  *返回一个IPCam对象
2506  */
2507 public static IPCam get_disposable_camera()//获取一次性摄像机
2508
2509 /*
2510  * IPCamMgr方法：设定一个一次性摄像机且自动连接
2511  *参数：
2512  *id:摄像机id
2513  *user: 摄像机登录名
2514  *pwd:摄像机登陆密码
2515  *https:摄像机安全传输值
2516  *返回：
2517  *返回一个BOOL, true表示设置成功, false表示设置失败
2518  */
2519 public static boolean set_disposable_camera(String id, String user, String pwd,
boolean https);//设定一次性摄像机
2520
2521 /*IPCamMgr方法：清除一次性摄像机*/
2522 public static void clear_disposable_camera();//清除一次性摄像机
2523
2524 /*IPCamMgr方法：更新一次性摄像机密码到IPCamMgr
2525  *返回一个BOOL, true表示更新成功, false表示更新失败
2526  */
2527 public static boolean update_disposable_camera_pwd(String pwd);
2528
2529 /*IPCamMgr方法：重连一次性摄像机
2530  *返回一个BOOL, true表示执行函数成功, false表示执行函数失败
2531  */
2532 public static boolean reset_disposable_camera();//重连一次性摄像机
2533
2534 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
2535
2536                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
2537                c) 不连接了一定要clear_disposable_camera
2538
2539 6) 示例：
2540     IPCamMgr.set_disposable_camera(id, user, pwd, https);
2541
2542 #16.6删除收藏摄像机
2543 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。
2544 2) 使用的类： Storage, Webservice
2545 3) 函数：
2546 /*
2547  *Storage方法：删除当前账号本地收藏的摄像机
2548  *参数：
2549  *share_id:分享id
2550  */
2551 public static void remove_collection(String collection_id);
2552
2553 /*
2554  *WebService方法：删除当前账号服务器上收藏的摄像机
2555  *参数：
2556  *share_id:分享id
2557  *delegate:监听remove_collection_listener
2558  *返回：
2559  *ERROR: 当返回值为ERROR.NO_ERROR说明函数执行成功
2560  */
2561 public static ERROR remove_collection(String share_id, remove_collection_listener
listener)
2562

```

```

2563  /*
2564  *WebService方法: remove_collection_listener回调函数
2565  *参数:
2566  *error:错误值, 当错误值为ERROR.NO_ERROR时, 说明删除成功
2567  */
2568  public void on_remove_collection_result(ERROR result, String share_id);
2569
2570  4) 步骤:
2571  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
2572  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
2573
2574  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。
2575           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。
2576
2577  6) 示例:
2578  /*第一步: 删除服务器上的收藏摄像机+添加监听remove_collection_listener*/
2579  public void delete_server()
2580  {
2581      if (ERROR.NO_ERROR != Webservice.remove_collection(m_Share_id, this)) {
2582          /*删除失败*/
2583      }
2584  }
2585  /*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*/
2586  public void on_remove_collection_result(ERROR result, String share_id) {
2587      if(result == ERROR.NO_ERROR){
2588          /*服务器删除成功, 本地进行删除*/
2589          Storage.remove_collection(m_Share_id);
2590      } else {
2591          /*删除失败*/
2592      }
2593  }
2594
2595  #16.7更新收藏摄像机本地信息
2596  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。
2597  2) 使用的类: Storage
2598  3) 函数:
2599  /*Storage方法
2600  *更新本地收藏的摄像机是否依旧合法, 可观看视频
2601  *collection id:要被更新的收藏摄像机分享id
2602  *valid:收藏摄像机的valid本地更新为valid
2603  */
2604  public static void update_collection_valid(String collection_id, boolean valid);
2605
2606  /*Storage方法
2607  *更新本地收藏摄像机的访问密码
2608  *collection id:要被更新的收藏摄像机分享id
2609  *pwd:收藏摄像机的pwd本地更新为pwd
2610  */
2611
2612  public static void update_collection_pwd(String collection_id, String pwd);
2613
2614  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
2615           b) 这些方法都是基于瑞彩提供的存储方式存储的。
2616
2617  5) 示例:
2618      Storage.init(this);
2619      Storage.update_collection_pwd(m_collection.collection_id, pwd);
2620
2621  #16.8获取服务器和本地收藏的摄像机
2622  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。
2623  2) 使用的类: Webservice, Storage
2624  3) 函数:
2625  /*
2626  *WebService方法: 从服务器获取账号收藏的所有摄像机
2627  *返回:返回一个COLLECTION_INFO对象, 具体如下:
2628  *share_id: 分享id
2629  *alias: 别名
2630  *cover_url:预览图的url
2631  */
2632  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的摄像机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");

```

```

2703         } else {
2704             Log.e("SoDemo","ejecting_tf_failed");
2705         }
2706     }
2707 }
2708
2709 #17.1.2格式化tf卡
2710 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
2711 2) 使用类: IPCam
2712 3) 函数:
2713
2714 /*
2715  *IPCam方法: 格式化tf卡
2716  *参数:
2717  *listener:format_tf_listener
2718  *返回: 返回ERROR类型, 当返回ERROR.NO_ERROR, 表示没有错误, 调用成功
2719  */
2720 public ERROR format_tf(format_tf_listener listener);
2721
2722 /*
2723  *IPCam方法: format_tf_listener回调
2724  *回调参数:
2725  *ipcam: 返回一个IPCam对象
2726  *error:ERROR类型, 当返回是ERROR.NO_ERROR, 表示弹出成功
2727  */
2728 public void on_result(IPCam ipcam, ERROR error);
2729
2730 4) 示例:
2731 /*第一步: 格式化tf卡+添加监听format_tf_listener
2732  */
2733 m_ipcam.format_tf(this);
2734 m_state = STATE.FORMAT;
2735 /*第二步: 监听format_tf_listener回调 */
2736
2737 public void on_result(IPCam ipcam, ERROR error) {
2738     if(m_state == STATE.FORMAT){
2739         if (error == ERROR.NO_ERROR) {
2740             Log.e("SoDemo","format_tf_succeed");
2741         } else {
2742             Log.e("SoDemo","format_tf_failed");
2743         }
2744     }
2745 }
2746
2747 #17.1.3tf卡状态和容量
2748 1) 说明: tf卡状态和容量是指: 查看tf状态和容量状态
2749 2) 使用类: IPCam
2750 3) 函数:
2751 /*
2752  *IPCam方法: 获取tf卡当前状态
2753  *返回: 返回TF_STATUS类型, 详细如下:
2754  *TF_STATUS.NONE:没有tf卡
2755  *TF_STATUS.READY:tf卡录像中
2756  *TF_STATUS.ERROR:tf卡错误
2757  *TF_STATUS.FULL:tf卡已满
2758  *TF_STATUS.BUSY:tf卡检测中
2759  */
2760 public TF_STATUS tf_status()
2761 /*
2762  *IPCam方法: 获取tf当前剩余容量百分比
2763  *返回:返回int,返回0-100
2764  */
2765 public int tf_free();
2766
2767 /*
2768  *IPCam方法: 获取tf总容量
2769  *返回:返回float, 单位MB
2770  */
2771 public int get_disk_size();
2772
2773 4) 示例:
2774     TF_STATUS tf = ipcam.tf_status();//状态
2775     int sd_disk = ipcam.tf_free();//容量

```

```
2776 float sd_capacity_total = ipc.get_disk_size()/1024; //总容量
2777 float sd_capacity_used = sd_capacity_totly*sd_disk/100; //剩余容量
2778
```

2779 #17.2tf卡文件列表 <下载所有文件 / 一天 / 一小时 / 一刻钟（预览图）>

2780 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。

2781 #17.2.1 下载tf卡文件

2782 17.2.1 下载tf卡文件（到刻钟）

2783 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。

2784 2)使用类：IPCam

2785 3)函数：

```
2786 /*
2787 *IPCam方法：下载所有的录像文件（到刻钟）
2788 *参数：
2789 *listener:load_tf_records_listener
2790 *返回：
2791 *返回ERROR类型，当返回的ERROR为! ERROR.NO_ERROR表示下载失败
2792 */
2793
2794 public ERROR load_tf_records(load_tf_records_listener listener)
2795 /*
2796 *IPCam方法：load_tf_records_listener回调
2797 *参数函数：
2798 *ipcam:返回当前摄像机
2799 *error:错误值，返回ERROR类型，当返回的ERROR.NO_ERROR表示获取成功。
2800 */
```

```
2801 public void on_result(IPCam ipcam, ERROR error);
```

```
2802
2803 /*
2804 *IPCam方法：停止load_tf_records
2805 */
```

```
2806 public void clear_tf_records()
```

2807 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。

2809 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

2810 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。

2811 #17.2.2 获取一天文件

2812 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。

2813 2)使用类：IPCam

2814 3)函数：

```
2815 /*
2816 *IPCam方法：获取每天的录像文件
2817 *参数：
2818 *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
2819 *
2820 *返回：返回一个TF_RECORD_DAY_INFO对象，具体如下：
2821 public class TF_RECORD_DAY_INFO {
2822     public boolean valid; 当天是否有录像文件，true:有录像文件，false:无录像文件
2823     public boolean alarm; 当天是否有报警，true:有报警，false:无报警
2824     public int week; 星期几，1至7分别代表星期天到星期五
2825     public boolean today; 今天，true:是今天，false:不是今天
2826     public boolean yesterday; 昨天，true:是昨天，false:不是昨天
2827     public boolean dby; 前天，true:是前天，false:不是前天
2828     public int valid_hours; 当前，此小时有没有录像文件
2829 }
2830 *
2831 */
2832
2833 public TF_RECORD_DAY_INFO get_tf_record_day_info(int day)
```

2834 4)注意：必须要load_tf_records成功后才能get_tf_record_day_info

2836

```

2837 #17.2.3 获取一小时文件
2838 1)说明: 获取一小时文件是指获取某一天某一个小时的录像文件, 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。
2839 2)使用类: IPCam
2840 3)函数:
2841 /*
2842 *IPCam方法: 获取第几天第几个小时的录像文件
2843 *参数:
2844 *day:第几天, 共7天, 范围 [0, 7)
2845 *hour: 第几个小时, 共24个小时, 返回 [0, 24)
2846 *返回:
2847 *返回一个BOOL, true表示执行成功, false表示执行失败
2848 */
2849
2850 public boolean get_tf_record_hour_valid(int day, int hour)
2851
2852 4) 注意: a)必须要load_tf_records成功后才能get_tf_record_day_info
2853
2854         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
2855
2856 #17.2.4 获取一刻钟录像信息
2857 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。
2858 2)使用的类: IPCam
2859 3)函数:
2860 /*
2861 *
2862 *IPCam方法: 获取第几天第几个小时第几刻钟的录像文件
2863 *参数:
2864 *day:第几天, 共7天, 范围 [0, 7)
2865 *hour: 第几个小时, 共24个小时, 范围[0, 24)
2866 *quarter:第几刻钟, 共4刻钟, 范围[0,4)
2867 *
2868 *返回: 返回一个TF_RECORD_QUARTER_INFO对象, 具体如下:
2869 public class TF_RECORD_QUARTER_INFO {
2870     public boolean valid; 当前一刻钟是否有录像文件, true:有文件, false:没有文件
2871     public boolean alarm; 当前一刻钟是否有报警, true:有报警, false:没有报警
2872 }
2873 */
2874 public TF_RECORD_QUARTER_INFO get_tf_record_quarter_info(int day, int hour, int
quarter)
2875
2876 4) 注意: 必须是get_tf_record_hour_valid成功后, 再去获取
2877
2878 #17.2.5 获取一刻钟的录像预览图
2879 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
2880 2) 使用类: IPCam
2881 3) 函数:
2882 /*
2883 *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
2884 *参数:
2885 *day:第几天, 共7天, 范围 [0, 7)
2886 *hour: 第几个小时, 共24个小时, 范围[0, 24)
2887 *valid_hour_index:有录像文件小时索引
2888 *quarter:第几刻钟, 共4刻钟, 范围[0,4)
2889 *listener:get_tf_record_quarter_thumb_listener
2890 */
2891
2892 public ERROR get_tf_record_quarter_thumb(int day, int hour, int valid_hour_index,
int quarter, get_tf_record_quarter_thumb_listener listener)
2893
2894 /*
2895 *IPCam方法: get_tf_record_quarter_thumb_listener接口
2896 *返回参数:
2897 *ipcam:返回的IPCam对象
2898 *day:第几天, 共7天, 范围 [0, 7)
2899 *hour: 第几个小时, 共24个小时, 范围[0, 24)
2900 *valid_hour_index:有录像文件小时索引
2901 *quarter:第几刻钟, 共4刻钟, 范围[0,4)

```

```

2902     *thumb: 预览图数据
2903     */
2904     public void on_result(IPCam ipcam, int day, int hour, int valid_hour_index, int
quarter, byte[] thumb);

```

2905
2906 4)注意: a)必须是get_tf_record_quarter_info成功后再去获取。

2908 17.2.6 示例

```

2909     /*第一步: 下载tf卡文件*/
2910     public void load_record(){
2911         if (ERROR.NO_ERROR != m_ipcam.load_tf_records(this)) {
2912             //下载tf卡文件失败
2913         }
2914     }
2915 }
2916
2917 /*第二步: load_record回调成功后, 先后获取天/时/刻录像文件信息*/
2918 public void on_result(IPCam ipcam, ERROR error) {
2919     {
2920         if (error != ERROR.NO_ERROR) {
2921             //下载tf卡文件失败
2922             return;
2923         }
2924         //load_record成功后开始获取7天的录像文件
2925         TF_RECORD_DAY_INFO day_info;
2926         TF_RECORD_QUARTER_INFO quarter_info;
2927         m_record_info = new RECORD_DAY_INFO[7];
2928         int d, h, q, vh;
2929         m_current_day = 0;
2930         for (d = 0; d < 7; d++) {
2931             /*
2932              *2.1 分别获取7天的录像文件
2933              */
2934             day_info = m_ipcam.get_tf_record_day_info(d);
2935             if (day_info.valid) {
2936                 //当天有录像文件
2937             } else {
2938                 //当天无录像文件
2939             }
2940             if (day_info.today) {
2941                 //第d天是今天
2942             } else if (day_info.yesterday) {
2943                 //第d天是昨天
2944             } else if (day_info.dby) {
2945                 //第d天是前天
2946             } else {
2947                 if (day_info.week == 1) {
2948                     //第d天是星期天
2949                 } else if (day_info.week == 2) {
2950                     //第d天是周一
2951                 } if (day_info.week == 3) {
2952                     //第d天是周二
2953                 } if (day_info.week == 4) {
2954                     //第d天是周三
2955                 } if (day_info.week == 5) {
2956                     //第d天是周四
2957                 } if (day_info.week == 6) {
2958                     //第d天是周五
2959                 } if (day_info.week == 7) {
2960                     //第d天是周六
2961                 }
2962             }
2963             //获取当天有录像文件开始获取当天24小时录像文件
2964             if (day_info.valid) {
2965                 m_current_day = d;
2966                 m_record_info[d].hours = new
RECORD_HOUR_INFO[m_record_info[d].valid_hours];
2967                 for (h = 0; h < 24; h++) {
2968                     /*
2969                      *2.2 分别获取每一天24小时的录像文件
2970                      */
2971                     if (m_ipcam.get_tf_record_hour_valid(d, h)) {
2972                         m_record_info[d].hours[vh] = new RECORD_HOUR_INFO();

```

```

2973         m_record_info[d].hours[vh].hour = h;
2974         m_record_info[d].hours[vh].quarters = new RECORD_QUARTER_INFO[4];
2975         for (q = 0;q < 4; q ++) {
2976             /*
2977              *2.3 分别获取一刻钟录像文件信息
2978              */
2979             m_record_info[d].hours[vh].quarters[q] = new
RECORD_QUARTER_INFO();
2980             quarter_info = m_ipcam.get_tf_record_quarter_info(d, h, q);
2981             if (quarter_info.valid){
2982                 //当前刻钟有录像文件
2983             }
2984             else{
2985                 //当前刻钟无录像文件
2986             }
2987             if (quarter_info.alarm){
2988                 //当前刻钟有报警
2989             }
2990             else{
2991                 //当前刻钟无报警
2992             }
2993         }
2994     }
2995     vh ++;
2996 }
2997 }
2998 }
2999 }
3000 /*第三步：获取刻成功后，获取每小时每刻的预览图*/
3001
3002 public View getView(int position, View convertView, ViewGroup parent) {
3003     if (convertView == null) {
3004         convertView =
m_inflater.inflate(getResources().getIdentifier("gridhour_item", "layout",
m_package_name) , null);
3005     }
3006     ImageView image;
3007     ImageView alarm;
3008     ProgressBar progress;
3009     int hour;
3010     RECORD_QUARTER_INFO info;
3011     int q;
3012     TF_RECORD_QUARTER_TIME t;
3013     for (q = 0;q < 4;q ++) {
3014         info = m_record_info[m_current_day].hours[position].quarters[q];
3015         if (info.load_state == LOAD_STATE.UNLOAD) {
3016             /*分别获取每一刻钟的预览图*/
3017             if (ERROR.NO_ERROR != m_ipcam.get_tf_record_quarter_thumb(m_current_day,
hour, position, q, this)) {
3018                 progress.setVisibility(INVISIBLE);
3019                 image.setVisibility(VISIBLE);
3020             } else {
3021                 info.load_state = LOAD_STATE.LOADING;
3022             }
3023         }
3024     }
3025     return convertView;
3026 }
3027
3028 /*第四步：获取预览图回调 */
3029 public void on_result(IPCam ipcam, int day, int hour, int valid_hour_index, int
quarter, byte [] thumb) {
3030     if (m_loadrecord_state != LOAD_STATE.LOADED) {
3031         return;
3032     }
3033     info.load_state = LOAD_STATE.LOADED;
3034     info.thumb = thumb;
3035     if (info.thumb != null) {
3036         image.setImageBitmap(BitmapFactory.decodeByteArray(info.thumb, 0,
info.thumb.length));
3037     }
3038 }
3039 }

```

```

3040 #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>
3041 17.3.1 三种播放tf卡文件方法
3042 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。
3043 2) 使用类: IPCam
3044 3) 函数:
3045 /*
3046 *IPCam方法: 播放录像文件, 从某一日日期开始
3047 *播放距离此日期最近的录像文件
3048 *SOSOCAM在播放报警录像的时候有用到
3049 *返回:
3050 *返回一个int32_t类型, 就是播放的录像文件id
3051 */
3052 public static int get_tf_record_play_id(Date date)
3053
3054 /*
3055 *IPCam方法: 播放录像文件, 从某一刻钟开始
3056 *SOSOCAM在点击某一刻钟直接播放时用到
3057 *参数:
3058 *day:播放哪一天, 范围 [0, 7)
3059 *hour:播放哪一小时, 范围 [0, 24)
3060 *quarter:播放哪一个刻钟, 范围 [0, 4)
3061 *返回:
3062 *返回一个int32_t类型, 就是播放的录像文件id
3063 */
3064
3065 public int get_tf_record_play_id(int day, int hour, int quarter)
3066
3067 /*
3068 *IPCam方法: 播放录像文件, 从某一刻钟某10s的开始
3069 *SOSOCAM在播放的时候拖动进度条的时候用到
3070 *参数:
3071 *day:播放哪一天, 范围 [0, 7)
3072 *hour:播放哪一小时, 范围 [0, 24)
3073 *quarter:播放哪一个刻钟, 范围 [0, 4)
3074 *no:播放刻钟的哪一个片段, 范围 [0, 90)
3075 *返回:
3076 *返回一个int32_t类型, 就是播放的录像文件id
3077 */
3078 public int get_tf_record_play_id(int day, int hour, int quarter, int no)
3079
3080 /*
3081 *IPCam方法: 播放tf卡录像文件
3082 *
3083 */
3084 public ERROR play_tf_record(int record_id)
3085
3086 /*
3087 *IPCam方法: IPCam Listener接口之tf录像事件监控
3088 *当有新录像文件生成或录像文件出错时会掉用此方法
3089 *SOSOCAM在播放tf卡录像文件是使用了此代理
3090 *使用方法: 参考IPCam基础监听
3091 *回调参数:
3092 *ipcam: IPCam对象
3093 *new_record: 是否是新生成的录像文件, true:是, false:否
3094 *record id: 录像文件开始id
3095 *error: 是否有错误, true:是, false:否
3096 *
3097 */
3098 public void on_tf_record_event(IPCam ipcam, boolean new_record, int record_id,
    boolean error);
3099
3100 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
3101
3102
3103
3104 5) 示例
3105
3106     IPCam.get_tf_record_play_id(date); //从某个时间最近的时间开始播放
3107     m_ipcam.play_tf_record(get_tf_record_play_id(0, 2, 2)); //从某一刻开始播放
3108     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
3109

```

3110 17.3.2 tf卡文件播放停止 / 暂停 / 续播 / TF卡录像播放状态

3111

3112 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。

3113 2) 使用类: IPCam

3114 3) 函数

3115

3116 /*

3117 *IPCam方法: tf卡录像播放状态

3118 *返回: 返回TF_RECORD_STATUS, 详细如下

3119 */

3120 public TF_RECORD_STATUS tf_record_status() //tf播放状态

3121 public enum TF_RECORD_STATUS {

3122 STOPPED, //停止

3123 REQUESTING, //请求

3124 PLAYING, //播放

3125 BUFFING, //缓冲

3126 PAUSING; //暂停

3127 }

3128 /*

3129 *IPCam方法: IPCam_Listener监听之tf播放状态监控

3130 *使用方法: 参考IPCam

3131 */

3132

3133 public void on_tf_record_status_changed(IPCam ipcam);

3134

3135 /*

3136 *IPCam方法: 停止播放tf卡录像文件

3137 */

3138 public void stop_tf_record();//停止播放

3139

3140 /*

3141 *IPCam方法: 暂停播放tf卡录像文件

3142 *返回:

3143 *返回IPCAM_ERROR类型, 当返回IPCAM_ERROR_NO_ERROR时, 表示暂停成功

3144 */

3145 public ERROR pause_tf_record();//暂停

3146

3147 /*

3148 *IPCam方法: 继续播放tf卡录像文件

3149 *返回:

3150 *返回ERROR类型, 当返回ERROR.NO_ERROR时, 表示继续播放成功

3151 */

3152 public ERROR continue_tf_record();//继续播放

3153

3154 4) 注意: a) 这些操作必须以摄像机连接成功为前提

3155 b) 只支持SOSO固件

3156

3157 5) 示例

3158

3159 TF_RECORD_STATUS status = m_ipcam.tf_record_status(); //状态

3160 m_ipcam.continue_tf_record();//继续播放

3161 m_ipcam.pause_tf_record();//暂停

3162 m_ipcam.stop_tf_record();//停止

3163

3164 17.3.3 上 / 下一个tf卡录像文件播放

3165

3166 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

3167 2) 使用类: IPCam

3168 3) 函数

3169 /*

3170 *IPCam方法: 播放下一个文件

3171 *参数:

3172 *day: 当前天, 范围 [0, 7)

3173 *hour: 当前小时, 范围 [0, 24)

3174 *quarter: 当前刻钟, 范围 [0, 4)

3175 *

3176 *返回: 返回下一个TF_RECORD_QUARTER_TIME对象, 具体如下:

3177 public class TF_RECORD_QUARTER_TIME {

3178 public int day; 下一个播放录像天, 范围 [0, 7)

3179 public int hour; 下一个播放录像小时, 范围 [0, 24)

3180 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 5) 示例

```

3203 /*播放上一个tf卡录像文件*/
3204 tf_play_prev.setOnClickListener(new View.OnClickListener() {
3205     @Override
3206     public void onClick(View arg0) {
3207         TF_RECORD_QUARTER_TIME t =
3208             m_ipcam.get_previous_tf_record_quarter_time(m_day, m_hour, m_quarter);
3209         //根据当前时间获取上一个文件的时间
3210         if (t == null)
3211             return;
3212         m_day = t.day;
3213         m_hour = t.hour;
3214         m_quarter = t.quarter;
3215         m_ipcam.stop_tf_record(); //先停止播放当前文件
3216         m_ipcam.play_tf_record(get_tf_record_play_id(m_day, m_hour, m_quarter));
3217         //播放上一个文件
3218     }
3219 });

```

```

3219 /*播放下一个tf卡录像文件*/
3220 tf_play_next.setOnClickListener(new View.OnClickListener() {
3221     @Override
3222     public void onClick(View arg0) {
3223         TF_RECORD_QUARTER_TIME t = m_ipcam.get_next_tf_record_quarter_time(m_day,
m_hour, m_quarter);
3224         if (t == null)
3225             return;
3226         m_day = t.day;
3227         m_hour = t.hour;
3228         m_quarter = t.quarter;
3229         m_ipcam.stop_tf_record(); //先停止播放当前文件
3230         m_ipcam.play_tf_record(get_tf_record_play_id(m_day, m_hour, m_quarter));
3231         //播放上一个文件
3232     }
3233 });

```

3234 17.3.4 下载tf卡录像文件

3235 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

3236 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 /*

```

3241 *IPCam方法:
3242 */
3243 public ERROR set_play_tf_record_cache(int cache)
3244
3245 /*
3246 *IPCam方法: 设置播放(下载)速度
3247 *参数
3248 *speed: 播放(下载速度), 有四个值
3249 *0:对应sosocam的1x, 正常播放(下载)
3250 *3:对应sosocam的8x, 跳跃播放(下载)
3251 *4:对应sosocam的16x, 跳跃播放(下载)
3252 *5:对应sosocam的32x, 跳跃播放(下载)
3253 */
3254 public void set_record_performance_speed (int speed); //设置播放(下载)速度
3255 /*
3256 *IPCam方法: 播放(下载)速度
3257 *返回
3258 *speed: 返回一个int数值
3259 *0:对应sosocam的1x, 正常播放(下载)
3260 *3:对应sosocam的8x, 跳跃播放(下载)
3261 *4:对应sosocam的16x, 跳跃播放(下载)
3262 *5:对应sosocam的32x, 跳跃播放(下载)
3263 */
3264 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

```

3271 /*示例一: 设置(播放)下载tf卡录像文件速度*/
3272 speed_slower = (ImageView) findViewById(getResources().getIdentifier("speed_slower",
3273 "id", m_package_name));
3274 speed_slower.setOnClickListener(new View.OnClickListener() {
3275     /*这里示范的是减速设置*/
3276     @Override
3277     public void onClick(View arg0) {
3278         String speed_v = "";
3279         speed_v = (String) record_play_speed.getText(); //获取当前播放(下载)速度
3280
3281         if (m_local_recording)
3282             return;
3283         if (speed_v.equals("1X"))
3284             m_ipcam.set_record_performance_speed(5); //设置播放(下载)速度
3285         else if (speed_v.equals("8X"))
3286             m_ipcam.set_record_performance_speed(0);
3287         else if (speed_v.equals("16X"))
3288             m_ipcam.set_record_performance_speed(3);
3289         else if (speed_v.equals("32X"))
3290             m_ipcam.set_record_performance_speed(4);
3291         m_ipcam.stop_tf_record(); //停止播放
3292
3293         record_play_speed.setText(get_performance_speed_string(m_ipcam.record_performa
3294 nce_speed()));
3295         m_ipcam.play_tf_record(m_ipcam.get_tf_record_play_id(m_day, m_hour,
3296 m_quarter, m_no));
3297     }
3298 });
3299
3300 /*示例二: 下载tf卡录像文件*/
3301 play_record =
3302 (ImageView) findViewById(getResources().getIdentifier("download_record", "id",
3303 m_package_name));
3304 play_record.setOnClickListener(new View.OnClickListener() {
3305     @Override
3306     public void onClick(View arg0) {
3307         if (m_local_recording){
3308             stop_local_record(true);
3309         }else{
3310             m_local_record_filepath = m_ipcam.start_local_record();
3311             if (m_local_record_filepath == null) {
3312                 Log.e("SODemo", "record fail");
3313             } else {

```

```

3308         m_local_recording = true;
3309         if (m_ipcam.record_performance_speed() != 0) {
3310             m_ipcam.set_record_performance_speed(0);
3311             record_play_speed.setText(get_performance_speed_string(0));
3312         }
3313         m_ipcam.set_play_tf_record_cache(0xffffffff);
3314     }
3315 }
3316 }
3317 });
3318

```

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

```

3321 /*
3322 *IPCam方法：获取一刻钟的某一个10s录像文件时间
3323 *
3324 *参数：
3325 *subrecord_id
3326 *返回：返回一个TF_RECORD_CLIP_TIME对象，具体如下：
3327 public class TF_RECORD_CLIP_TIME {
3328     public int day;    天
3329     public int hour;  时
3330     public int quarter; 刻钟
3331     public int no;    当前刻钟的第几个10S
3332 }
3333 */
3334 public TF_RECORD_CLIP_TIME get_tf_record_clip_time(int subrecord_id)
3335
3336 /*
3337 *IPCam方法：获取一刻钟的某一个10s录像预览图图
3338 *参数：
3339 *day:第几天，共7天，范围 [0, 7)
3340 *hour: 第几个小时，共24个小时，范围[0, 24)
3341 *quarter:第几刻钟，共4刻钟，范围[0,4)
3342 *no:第几个10s,15分钟，10s中一个文件，共90个文件，范围[0,90)
3343 *listener:get_tf_record_clip_thumb_listener
3344 *返回：
3345 */
3346 public ERROR get_tf_record_clip_thumb(int day, int hour, int quarter, int no,
3347 get_tf_record_clip_thumb_listener listener)
3348
3349 /*
3350 *IPCam方法：get_tf_record_clip_thumb_listener回调
3351 *返回参数：
3352 *ipcam:摄像机
3353 *day:第几天，共7天，范围 [0, 7)
3354 *hour: 第几个小时，共24个小时，范围[0, 24)
3355 *quarter:第几刻钟，共4刻钟，范围[0,4)
3356 *no:第几个10s,15分钟，10s中一个文件，共90个文件，范围[0,90)
3357 *thumb: 预览图数据
3358 */
3359 public void on_result(IPCam ipcam, int day, int hour, int quarter, int no, byte[]
3360 thumb);
3361
3362 /*
3363 *IPCam方法：获取某一刻钟90个10s录像文件报警信息
3364 *
3365 *参数：
3366 *day:天，范围 [0, 7) ,类型 NSNumber (int)
3367 *hour:小时，范围 [0, 24) ,类型 NSNumber (int)
3368 *quarter:刻钟，范围 [0, 7) ,类型 NSNumber (int)
3369 *listener:get_tf_record_quarter_detail_listener
3370 */
3371 public ERROR get_tf_record_quarter_detail(int day, int hour, int quarter,
3372 get_tf_record_quarter_detail_listener listener)
3373
3374 /*
3375 *IPCam方法：get_tf_record_quarter_detail_listener回调
3376 *

```

```

3377 *返回:
3378 *ipcam:摄像机
3379 *day:天
3380 *hour:时
3381 *clips:一个TF_RECORD_CLIP_INFO对象, 90个10s的录像文件信息, 具体如下:
3382 public class TF_RECORD_CLIP_INFO {
3383     public boolean valid;    当前刻钟是否有录像文件
3384     public boolean thumb;   当前刻钟是否有预览图
3385     public int alarm;       当前刻钟是否有报警
3386 }
3387 */
3388 public void on_result(IPCam ipcam, int day, int hour, int quarter,
TF_RECORD_CLIP_INFO[] clips);
3389
3390 4) 示例
3391
3392 /* 示例一: 获取某一时刻的90个文件的信息*/
3393 m_ipcam.get_tf_record_quarter_detail(day, hour, quarter, this);
3394
3395 //get_tf_record_quarter_detail_listener回调
3396 public void on_result(IPCam ipcam, int day, int hour, int quarter,
TF_RECORD_CLIP_INFO [] clips) {
3397     if ((m_day == day) && (m_hour == hour) && (m_quarter == quarter)) {
3398         int i;
3399         if (clips == null) {
3400             } else {
3401                 m_clips_info = clips;
3402                 for (i = 0; i < 90; i++) {
3403                     if (! clips[i].valid)
3404                         //此10s文件有录像文件
3405                     else if (clips[i].alarm == 0)
3406                         //此10s文件有录像文件+无报警
3407                     else
3408                         //此10s文件有录像文件+且有报警
3409                 }
3410             }
3411         }
3412     }
3413
3414 /* 示例二: 通过播放id获取当前文件时间*/
3415
3416 TF_RECORD_CLIP_TIME t = m_ipcam.get_tf_record_clip_time(record_id);
3417 m_day = t.day;//天
3418 m_hour = t.hour;//时间
3419 m_quarter = t.quarter;//刻钟
3420 m_no = t.no;//第几个10秒
3421
3422 int minutes = quarter * 15 + no / 6;
3423 int seconds = (m_no * 10) % 60;
3424
3425
3426 /*示例三: 通过获取10s文件预览图*/
3427 m_ipcam.get_tf_record_clip_thumb(m_day, m_hour, m_quarter, no, this);
3428
3429 //get_tf_record_clip_thumb_listener回调
3430 public void on_result(IPCam ipcam, int day, int hour, int quarter, int no, byte []
thumb) {
3431     if (thumb == null) {
3432         m_clip_thumb.setImageBitmap(null);
3433     } else {
3434         m_clip_thumb.setImageBitmap(BitmapFactory.decodeByteArray(thumb, 0,
thumb.length));
3435     }
3436 }
3437
3438 17.4.6取消tf卡录像文件的所有操作
3439 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。
如果继续操作tf卡录像文件, 直接操作就不需要重新load_tf_records。
3440 2) 使用类: IPCam
3441 3) 函数:
3442 /*
3443 *IPCam方法: 取消tf卡录像文件的所有操作
3444 */

```

```
3445 public void cancel_tf_record_tasks()  
3446  
3447 4) 示例  
3448 m_ipcam.cancel_tf_record_tasks();  
3449  
3450  
3451  
3452  
3453
```