摄像机项目复盘与梳理

——以`chuamgmi_m300`、母婴看护版项目代码为例

计划以插件界面作为突破口，从UI层逐层向下分析，大致以`UI-->入口组件-->嵌套调用组件/组件渲染函数`为主干，梳理代码和实现逻辑。对近期遇到的较为复杂的组件和页面做详细剖析，并揣测采用不同代码处理思路的原因。

写在前面

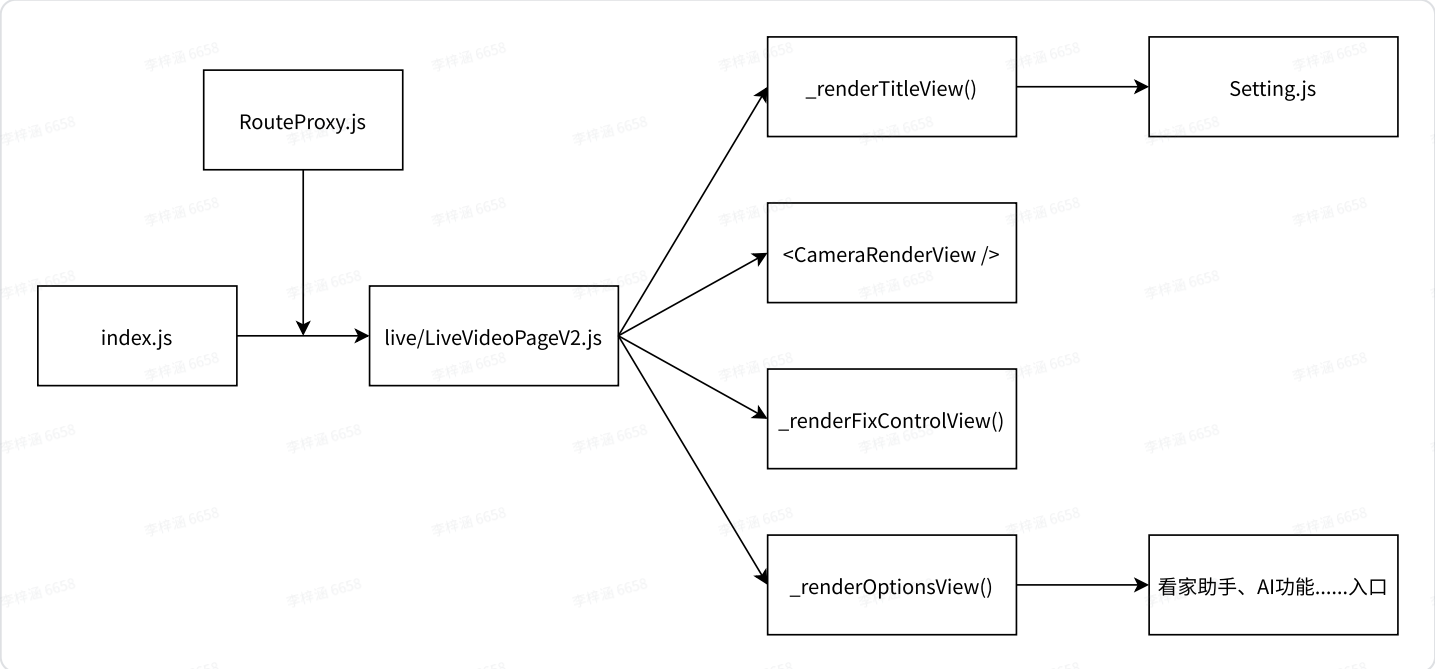
- 摄像机项目使用React Native编写，属于米家拓展程序，需要在米家SDK框架下开发。拓展程序开发可参考小米IoT文档与资源中心
- 米家拓展程序提供常用的UI组件米家插件通用UI组件手册 以及 米家扩展程序开发概述

项目目录结构

先对项目目录与功能建立大致的映射关系

```
Main:
├─aicamera                    AI相关功能实现，人脸识别、每日故事等
├─alarm                       看家界面
├─alarmDetail                 看家界面组件、API等
├─allVideo                    展示全部视频界面
├─components                  公共组件
├─config
├─framework
├─live                        首屏界面
├─scene                       智能场景界面
├─sdcard                      拉取SD卡和云存中视频相关的界面
├─setting                     设置相关的界面
├─testPage
├─ui                          ui组件
├─util                        工具类（包括传统打点）
├─util2                       工具类
├─widget                      小部件（日历控件等）
├─API.js
├─BasePage.js                 可复用的基础界面框架
├─BaseSettingPage.js          可复用的设置界面框架
├─Constants.js                一些常量
├─index.js                    入口文件
├─App.js
```

```
├─MHLocalizableString.js          字符串文件（便于国际化）
├─RouteProxy.js                   路由注册文件
├─SceneMain.js                    智能场景的入口文件
├─StackNavigationInstance.js
├─StorageKeys.js                  存储一些状态（弹窗等）
├─Toast.js                        弹窗控件的重写
```



首页 -- LiveVideoPageV2

首页从UI层可分为三大部分（红色框圈出）

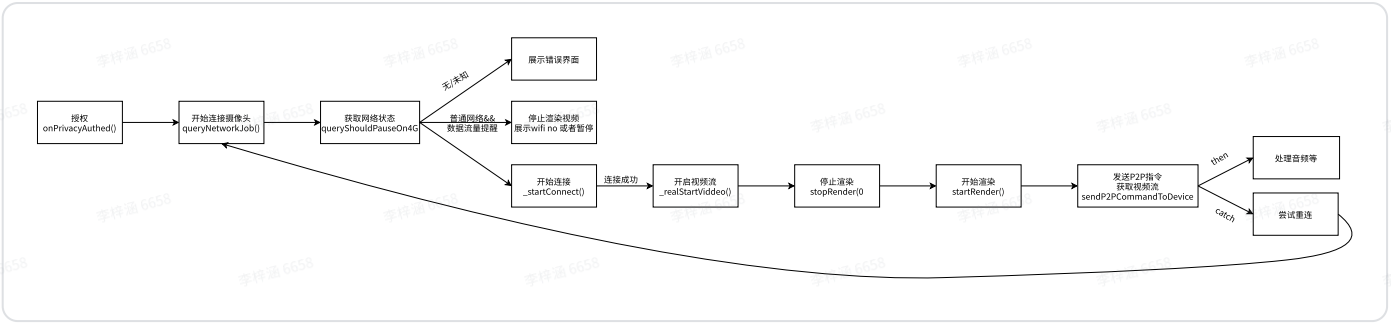
- 标题栏-- `_renderTitleView()`
  - <ImageButton 返回按钮 /> --> 调用 `Package.exit()` 退出插件
  - 标题＋网速
  - <ImageButton 设置入口按钮 />
- 直播画面-- `_renderVideoView()`
  - 依赖于组件 `<CameraRenderView />`
  - 摄像头与米家APP使用P2P点对点传输
  - 缩放功能在Java层实现
- 功能界面
  - 横向固定功能栏
  - 云存运营推广 --> 服务端拉取部分内容

- 竖向滑动功能栏

直播视频流实现细节

连接摄像机，通过P2P推流

- 整体流程：实现视频流展示，并做网络中断重连等错误处理。



a. 绑定相关回调函数（`bindConnectionCallback` 连接状态、`bindNetworkInfoCallback` 网络状态、 `bindPauseAllCallback` 休眠、`bindLocalModeCallback` 模式改变）

b. 入口函数： `_startConnect()` 连接摄像机

c. `startVideo()` 开启视频流，发送P2P指令；失败时进入检查网络参数的流程 `queryNetWorkJob()`

d. `queryNetWorkJob()` 监测网络状况，失败进入 `_stopAll()` 销毁，网络允许进入 `_startConnect()`

- 监听连接过程的状态---几个回调函数

```
1 CameraPlayer.getInstance().bindConnectionCallback(this._connectionHandler);//
2 CameraPlayer.getInstance().bindP2pCommandCallback(this._p2pCommandHandler);//
3 CameraPlayer.getInstance().bindNetworkInfoCallback(this._networkChangeHandler
4 CameraPlayer.getInstance().bindPauseAllCallback(() => { this._stopAll(false,
5 CameraPlayer.getInstance().bindPowerOffCallback(this._powerOffHandler);
6 CameraPlayer.getInstance().bindLocalModeCallback(this._localModeChange);// 本
```

- 获取视频具体过程：

a. 首先使用 `connectToDeviceWithStateChangeCallBack` 建立插件与摄像机固件之间的P2P连接；

b. 其次使用 `Service.miotcamera.bindP2PCommandReceiveCallback` 通知米家App命令回复值通过这个callback回调给插件，相当于摄像机发送命令时直接接受方是米家APP 需要告诉米家App使用该回调函数将命令返回给插件；

c. 可以使用 `CameraRenderView.startRender()` 替换前面两步；

d. 最后使用 `Service.miotcamera.sendP2PCommandToDevice` 向设备发送start指令；

e. 可以使用 `CameraRenderView.stopRender()` 停止视频流的渲染；

f. 使用 `Service.miotcamera.disconnectToDevice(did)` 断开米家APP建立的P2P连接

g. 发送P2P指令等是封装在原生模块中的，可以使用 `NativeModules` 直接调用

看家界面

- 利用 `createMaterialTopTabNavigator` 建立一个顶部Tab切换界面

注：利用Tab切换时不会触发组件销毁过程，涉及到生命周期的功能需全面考虑。可以利用 `didFocus` 监控navigation的跳转。

视频文件的获取

在 `util` 文件夹下已经有了获取视频文件相关的工具类，

SD卡

`SdcardEventLoader` 与 `SdFileManager`

云存

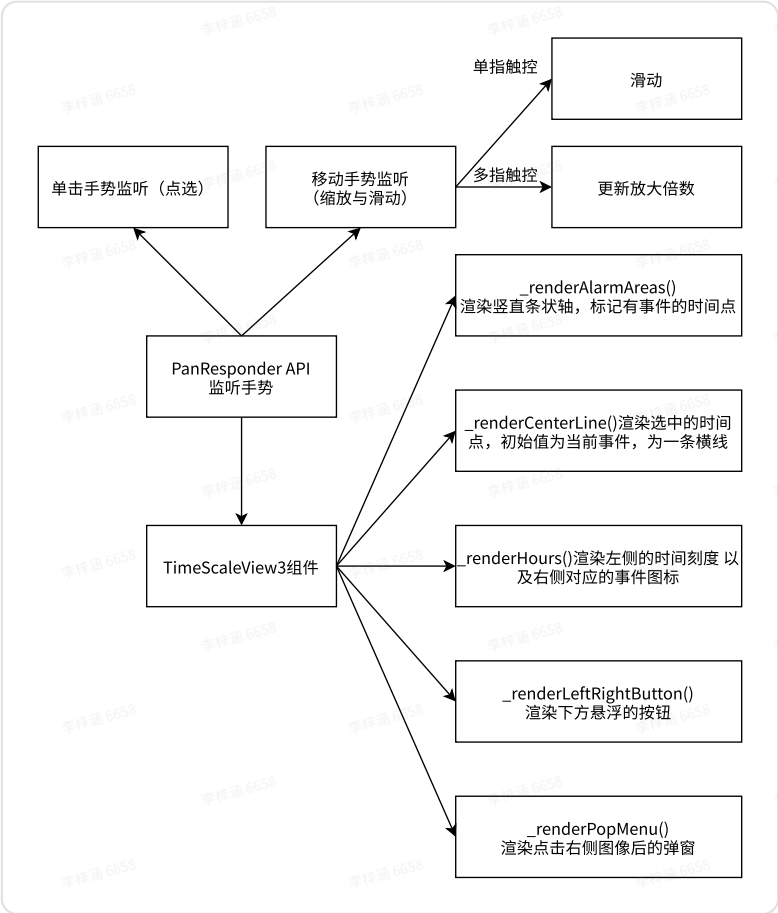
`CloudVideoUtil`

```
setCloudFilesReceivedCallback // 设置收到云存文件时的回调

// 调用接口：/common/app/v1/cloud/file/exist
fetchCloudVideoDataSerials(did, model, isVip){} // 拉取连续视频数据

getLastestVideo(){} // 找到最新视频

searchNeareastVideoItem(selectedTime){} // 找到特定时间附近的事件
```

看家视频的展示有两种形式：

- 利用 `react-native-video` 中的 `<Video/>` 组件通过视频文件的uri获取视频资源并展示；
- 利用 `<CameraRenderView>` 组件通过规定开始时间和播放时长展示特定时间视频。

时间轴



didFocus和didBlur

React Native之didFocus和didBlur-CSDN博客

设置

Spec属性

插件通过Spec协议获取、更新摄像机相关设置，需要小米IoT开发者平台中定义了相关属性或事件。

1. 根据平台上定义的Spec name在文件中定义对应的piid与siid变量
2. 获取spec属性状态

```
/**
const params = [
    { sname: SIID_THEME_SETTINGS, pname: PIID_THEME },
    { sname: SIID_THEME_SETTINGS, pname: PIID_TIME_SHOW },
    { sname: SIID_THEME_SETTINGS, pname: PIID_CALENDER_SETTINGS },
    { sname: SIID_THEME_SETTINGS, pname: PIID_DEVICE_LOCATION }
```

```
];
**/
AlarmUtilV2.getSpecPValue(params)
  .then((res) => {
    if (res[0].code == 0) {
      console.log("get ThemeSettings OK=========", res, res[3].value);
      /*拿到属性后对应处理*/
    } else {
      Toast.fail('c_get_fail');
    }
  })
  .catch((err) => {
    console.log("=========error", err);
    Toast.fail('c_get_fail', err);
  });
```

3. 设置spec属性值

```
/**
const params = [
    { sname: SIID_THEME_SETTINGS, pname: PIID_TIME_SHOW, value: result },
    { sname: SIID_THEME_SETTINGS, pname: PIID_TIME_SHOW, value: result }
]
**/
AlarmUtilV2.setSpecPValue(setParamsValue)
  .then((res) => {
    if (res[0].code === 0) {
      console.log('set OK ===========', res)
      Toast.success("c_set_success");
  }).catch((err) => {
    Toast.fail('c_get_fail', err);
  });
```

4. 具体的Spec属性设置与获取过程
   - 构造params参数。我们只需要传入对应的名称，Spec属性值是通过Spec协议获取的

```
static async getSpecPiidParams(skey, pkey) {
    let res = await Service.spec.getSpecByKey(Device.deviceID, { mkey: null,
    return res;
}
static async buildGetSpecPiidParams(params) {
    let getParams = [];
    for (let element of params) {
```

```
        let res = await this.getSpecPiidParams(element.sname, element.pname);
        getParams.push({ did: Device.deviceID, siid: res[0].siid, piid: res[0].
    }
    return getParams;
}
```

- getValue的实现

```
static async getSpecPValue(params, datasource = 2, tag = "tag") {
    let requestTime = Date.now();
    LogUtil.logOnAll(tag, "getSpecPValue params=", params, " taskID=", reques
    let requestParams = await this.buildGetSpecPiidParams(params);
    LogUtil.logOnAll(tag, "getSpecPValue buildGetSpecPiidParams=", requestPar
    return new Promise((resolve, reject) => {
      Service.spec.getPropertiesValue(requestParams, datasource).then((res) =
        LogUtil.logOnAll(tag, "getSpecPValue res==", JSON.stringify(res), " t
        resolve(res);
      }).catch((err) => {
        LogUtil.logOnAll(tag, "getSpecPValue err==", JSON.stringify(err), " t
        reject(err);
      });
    });
}
```

- setValue的实现

```
static async setSpecPValue(params, tag = "tag") {
    let requestTime = Date.now();
    LogUtil.logOnAll(tag, "setSpecPValue params=", params, " taskID=", reques
    let requestParams = await this.buildSetSpecPiidParams(params);
    LogUtil.logOnAll(tag, "setSpecPValue buildSetSpecPiidParams=", requestPar
    return new Promise((resolve, reject) => {
      Service.spec.setPropertiesValue(requestParams).then((res) => {
        LogUtil.logOnAll(tag, "setSpecPValue res==", JSON.stringify(res), " t
        resolve(res);
      }).catch((err) => {
        LogUtil.logOnAll(tag, "setSpecPValue err==", JSON.stringify(err), " t
        reject(err);
      });
    });
}
```

设置界面

无论是一级设置界面还是某个设置界面，都可以继承 `BaseSettingPage` 来实现标准化的快速开发，该类提供了统一的标题样式、返回动作等，可以专注于具体的设置内容。

通用设置

对于通用设置选项，使用 `Setting` 函数进行渲染，通过设置 `firstOptions` 和 `secondOptions` 控制设置项的显隐。

```
// 显示部分一级菜单项
let firstOptions = [ firstAllOptions.NAME, firstAllOptions.LEGAL_INFO ];
// 显示部分二级菜单项
let secondOptions = [ secondAllOptions.SECURITY, secondAllOptions.ADD_TO_DESKTOP

/*下面放在render中的return中进行渲染*/
{Settings({
  navigation: this.props.navigation,
  firstOptions,
  secondOptions,
  showDot: this.state.showDot,
  extraOptions: this.state.extraOptions,
  children: customOptions })}
```

特有设置

常用米家SDK中的 `ListItem` 和 `ListItemWithSwitch` 进行某个具体的设置项开发

米家插件通用UI组件手册

埋点使用说明

什么是埋点

埋点是记录某一事件的信息集合，包括事件名&事件属性。这些信息以键值对的形式记录在日志中埋点--词典

需要我们做什么

1. 在文件中（一般为 `Main\util\TrackPoints.js` ）加入点位的 `唯一标识` 与 `key_name` 的映射关系（ `"key_name:["唯一标识", ""]"` ）。类似于一个点位的数据库；
2. 找到点位的触发代码，在UI层面全面考虑触发点位的情况；

3. 利用 `console.log()` 或 `Toast` 等方式进行验证，是否全面触发点位、是否会重复触发点位；
4. 对于传统打点，选择打点方法
    a. `TrackUtil.reportClickEvent("key_name")`：点击事件、触发等；
    b. `TrackUtil.reportResultEvent('key_name', 'type', 'value')`：其中value支持 `(int,strig,float,boolean)` 类型；开关状态、筛选状态、事件上报等涉及到数值、状态上传的。

常见埋点类型

1. 点击事件：直接找到对应的点击触发函数即可；
2. 触发事件：同点击事件，但要注意尽量避免在渲染函数 `render` 中埋点，这可能会导致多次重复上报；
3. 曝光时间：全面考虑能切换到该页面的入口，先记录进入时间，在退出页面时进行时间差的上报

    如：母婴看护版项目的SD卡页和云存页，存在Tab切换，这样组件不会销毁，再进入时不执行 `componentWillMount`、`componendDidMount` 方法，可以直接利用页面已经实现的 `_onResume()` 和 `_onPause()` 方法。

『React Navigation 3x系列教程』之createStackNavigator开发指南

react-native系列(14)导航篇：页面导航StackNavigator参数及使用详解_stack.navigator-CSDN博客