CloudDevice 服务

开发指南

文档版本 01

发布日期 2025-08-20

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 云车机开发指南

1.1 开发概述

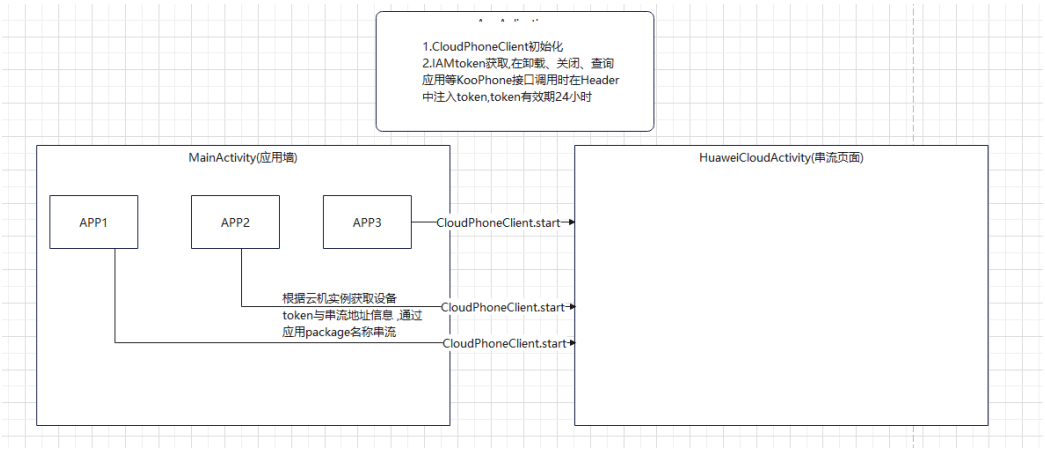
云车机融合了ARM服务器虚拟化、音视频编解码、实时传输能力等核心技术，为更多带屏联网设备提供云算力和云应用的能力。充分发挥端云协同的优势，提供端云交互、麦克风和摄像头等各类传感器的模拟仿真，以及应用和数据权限云端统一管控等多种功能特性，并基于华为全场景智慧生态满足客户不同应用诉求。

云车机开发指南将指导您如何开发云车机APP。

1.2 总体开发思路

云车机整体结构如图1-1所示。

图 1-1 云车机结构图



云车机开发一般分为两个页面。

1. 一个车机应用墙页面，加载预制或从云OS中查询到的应用列表。
2. 一个专门串流的页面，打开app后，画面在此页面上显示。

1.3 准备工作

开发技能要求

熟悉Android开发。

收集信息

用户在本地安装“云车机”APP，需要先下载最新的云车机端侧SDK开发包，可通过**下载云车机SDK软件包**获取最新版本的云车机端侧SDK开发包。

环境要求

- JDK11+
- 只支持arm64设备

搭建开发环境

下载并安装Android studio，创建app工程目录。

1.4 快速开始

步骤1 在工程libs下导入aar。

将cloudphone-merge-cloudapp-xxx-kp.hmtp-cabin-xxx.aar放到工程libs下引入。

步骤2 初始化SDK。

1. 在AppAplication中初始化串流SDK。

```
CloudPhoneClient.init(Application application, Callback callback, boolean openLog);
CloudPhoneClient.playerForceTextureView(false);
CloudPhoneClient.setLogFolder(this.getExternalFilesDir("log").getPath() + File.separator);
```

初始化说明：

- callback：非必填，云机串流的结果及中途异常消息回调，初始化可空。
- openLog：是否输出log，是否加密本地日志文件。

传入日志文件位置：由于安全合规且自定义，需要传入app私有路径，方便排查。

2. 获取IAM token，可参考文档**获取IAM用户Token**。

接口地址：POST https://iam.myhuaweicloud.com/v3/auth/tokens?nocatalog=true。

请求示例：

```
{
    "auth": {
        "identity": {
            "methods": [
                "password"
            ],
            "password": {
                "user": {
                    "domain": {
                        "name": "IAMDomain"        //IAM用户所属账号名
                    },
                    "name": "IAMUser",             //IAM用户名
```

```
                "password": "IAMPassword"      //IAM用户密码
            }
        }
    },
    "scope": {
        "project": {
            "name": "cn-north-1"              //项目名称
        }
    }
  }
}
```

获取token:

```
响应Header参数（获取到的Token）：
X-Subject-Token:MIIatAYJKoZIhvcNAQcCoIIapTCCGqECAQExDTALB...
```

步骤3 串流打开数据通道。

1. 获取云机设备token与地址信息，可参考文档**租户实例串流前获取设备的device_token**。

   接口地址：POST https://koophone.myhuaweicloud.com/v1/instances/{instance_id}/auth。

   请求示例

   post接口，请求url中携带租户自己购买的实例和租户的token直接调用。

   ```
   /v1/instances/xxxxxx/auth
   ```

   响应示例

   状态码：200

   实例鉴权接口返回设备信息。

   ```
   {
     "data" : {
       "resource" : {
         "sdk" : {
           "internal" : {
             "address" : null,
             "aport" : null,
             "atype" : null,
             "address_ipv6" : null
           },
           "external" : {
             "address" : "10.83.71.187",
             "aport" : 10030,
             "atype" : 1,
             "address_ipv6" : null
           }
         },
         "rtc" : {
           "ice_signaling" : {
             "signaling_url" : "http://100.93.2.248:18082",
             "expired_time" : null,
             "ice_servers" : [ ]
           }
         },
         "device_id" : "7b0cd026df8d495b8a65d628d7bec433",
         "kp_id" : "Q39YyZvl"
       },
       "device_token" : "dee5081f40c83ddea3ded91c387351e9"
     },
     "error_code" : "0",
     "error_msg" : "ok"
   }
   ```

2. 启动串流。

   关于CloudPhoneClient其他公开方法说明可参考**表1-1**。

表 1-1 CloudPhoneClient 方法名及参数说明

方法名及参数	说明
start(PlayerFragment mPlayerFragment, Bundle extras)	开始串流
closeApp(String packageName)	通过应用包名强制关闭云手机中运行的目标应用
startApp(String packageName)	通过包名/类名或Scheme协议启动目标应用，支持传递自定义参数
uninstallApp(String packageName)	通过应用包名卸载云手机中的指定应用（系统级静默卸载，无需用户确认）
getRunningApps(GetAppsReq req)	查询当前云手机环境中正在运行的应用列表及其状态信息
switchStreamType(String type)	改变串流类型 streamType枚举： – DATA：仅数据传输通道 – VIDEO：仅拉取视频流，不拉取音频流，包含DATA – AUDIO：仅拉取音频流，不拉取视频流，包含DATA – BOTH：启动后音视频正常拉流，默认，包含DATA
void setPlayerCallback(Callback callback)	串流回调注册
void setUnPlayerCallback(Callback callback)	串流回调注销，断开串流需要移除
void stop()	主动断开串流调用该方法
String version()	获取当前版本号
boolean setProfile(int level)	设置清晰度档位。0：自动，1：流畅，2：高清，3：超清
void mute(boolean mute)	设置静音
boolean isPlaying()	判断当前是否串流中
sendExtMessageToCloud(String message)	发送消息到云端App。message消息体（json格式的字符串）
setCloudScreenAspect(int screenWidth, int screenHeight)	动态设置云机画面比例。 screenWidth：真机屏幕宽， screenHeight：真机屏幕高
void enableAudioKeeping(boolean isKeep)	保持后台保活

方法名及参数	说明
void setSensorEvent(boolean isLandscape);	设置串流横屏或者竖屏显示。true：横屏，false：竖屏
sendKeyEvent（ int keycode ）	KeyEvent.KEYCODE_BACK，KeyEvent.KEYCODE_HOME，KeyEvent.KEYCODE_APP_SWITCH

a. 在MainActivity中根据接口获取到的token与设备信息组装串流参数打开数据通道串流。

```
 public void jumpTargetApp(ApiResponse authResponseResponseBean) {
        Bundle bundle = new Bundle();
        if (authResponseResponseBean != null && null != authResponseResponseBean.getData()) {
            ApiResponse.Data data = authResponseResponseBean.getData();
            //校验token，后台返回 如果云机校验失败会返回100002异常码
            bundle.putString(CLOUD_APP_LAUNCH_KEY_TOKEN,
TextUtils.isEmpty(data.getDevice_token()) ? "" : data.getDevice_token());
            ApiResponse.Data.Resource resource = data.getResource();
            if (null != resource) {
                ApiResponse.Data.Resource.Sdk sdk = resource.getSdk();
                if (null != sdk) {
                    ApiResponse.Data.Resource.Sdk.External external = sdk.getExternal();
                    if (null != external) {
                        //address 云机ip，后台返回
                        bundle.putString(CLOUD_APP_LAUNCH_KEY_ADDRESS, external.getAddress());
                        //aport:云机端口，后台返回
                        bundle.putInt(CLOUD_APP_LAUNCH_KEY_PORT, external.getAport());
                        //atype:后台返回
//                          bundle.putInt(CLOUD_APP_LAUNCH_KEY_TYPE, external.getAtype());
                        bundle.putInt(CLOUD_APP_LAUNCH_KEY_TYPE, 8);
                        bundle.putInt(CLOUD_APP_LAUNCH_KEY_V_TYPE, 8);
                    }
                }
            }
        }
        //后台返回，有就填
//        bundle.putString(CLOUD_APP_LAUNCH_KEY_BUILD_DEVICE,"");
        // 强制开启H264
//        bundle.putInt(CLOUD_APP_LAUNCH_KEY_VENC_TYPE, 0);
        // 强制开启H265
        bundle.putInt(CLOUD_APP_LAUNCH_KEY_VENC_TYPE, 1);
        bundle.putInt(CLOUD_APP_LAUNCH_KEY_PROFILE_LEVEL, PROFILE_SPEED);
        //多云机共用用户id，可用用户登录id，也可自定义，保证不与其他用户相同
        bundle.putString("userId", "43740902807");
        bundle.putBoolean(CLOUD_APP_HIDE_STREAM_AT_START_UP, true);
//        if (!TextUtils.isEmpty(pkg)) {
//            bundle.putString("pkgs", pkg);
//        }
        //开启
        bundle.putBoolean("free_aspect", true);
        bundle.putInt("screen_width", 1920);//任意分辨率宽
        bundle.putInt("screen_height", 1080);//任意分辨率高
        CloudPhoneClient.setPlayerType(HMTP_PLAYER);
        CloudPhoneCallBack callBack = CloudPhoneCallBack.getInstance();
        callBack.setStartHandler(v->onStreamStarted());
        callBack.setAppHandler(this::updateAppList);
        callBack.setStartAppHandler(v->startApp());
        CloudPhoneClient.enableAudioKeeping(true);
        CloudPhoneClient.setPlayerCallback(callBack);
        CloudPhoneClient.setAppCallback(callBack);
        //执行串流的，串流结果成功会从onSuccsss返回通知，失败则是onFailure带code
        CloudPhoneClient.start(null, bundle);
    }
```

b. 串流回调结果Callback。

需要在串流界面内注册Callback。

i. 注册用回调CloudPhoneClient.setPlayerCallback(new Callback())。

ii. 注册应用操作回调。

```
CloudPhoneClient.setAppCallback(callBack);
```

iii. 退出云机注销CloudPhoneClient.setUnPlayerCallback(this)。

c. 接受到串流回调onStreamStarted加载应用墙。

步骤4 应用墙MainActivity显示应用列表。

查询云机中已安装的应用列表，与本地缓存或数据库保存的应用白名单匹配，取并集后把应用列表显示到应用墙上。

1. 调用SDK接口可以获取云机已经安装的应用列表。

```
CloudPhoneClient.getInstalledApps(GetAppsReq req)
```

请求参数GetAppsReq的参数说明参考**表1-2**。

表 1-2 GetAppsReq 参数说明

参数名	类型	描述	约束条件
appType	AppTypeEnum	应用类型筛选（必填）	枚举值：SYSTEM_APP（系统应用）、THIRD_APP（第三方应用）、BOTH（全部）
quality	Integer	图标质量参数（选填，默认100）	取值范围0-100，建议取值为70~85以平衡清晰度与大小
pageNum	Integer	分页页码（必填）	从1开始，需配合pageSize使用
pageSize	Integer	分页大小（必填）	最大支持20条/页
needIcon	Boolean	是否返回图标（必填）	true返回图标数据/URL，false不返回图标相关字段

getInstalledApps的回调参数说明参考**表1-3**。

表 1-3 getInstalledApps 回调参数说明

字段	类型	业务说明
error_code	String	状态码（0=成功）

字段	类型	业务说明
error_msg	String	错误详情
data	**GetAppsRsp**	磁盘使用数据对象

回调接口通过CloudPhoneClient.setAppCallback(CloudPhoneClient.Callback.AppCallback callback) 注册，回调参数为AppOperateResponse<GetAppsRsp>，其中GetAppsRsp结构参考**表1-4**。

表 1-4 GetAppsRsp 整体结构

参数名	类型	描述
appList	List	应用列表（分页数据）
totalCount	Integer	符合筛选条件的应用总数
pageNum	Integer	当前分页页码（与请求参数一致）
pageSize	Integer	当前分页大小（与请求参数一致）

CloudAppInfo应用详情结构参考**表1-5**。

表 1-5 CloudAppInfo 整体结构

参数名	类型	描述	说明
appName	String	应用名称	如“微信”、“设置”
packageName	String	应用包名（唯一标识）	如“com.tencent.mm”
icon	String	WebP格式图标原始数据Base64编码	仅当needIcon=true时可能返回，null表示无数据
iconUrl	String	WebP格式图标URL	端侧优先使用此字段，无值时allback到icon字节数组

参数名	类型	描述	说明
isReady	Boolean	应用是否准备就绪（可启动状态）	true表示应用可正常使用，false可能为未安装完成或异常状态

2. 打开串流应用。

```
CloudPhoneClient.startApp(pkgName);
```

收到打开应用成功回调onStartApp后打开串流页面。

```
private void startApp(){
    //点击事件：
    Intent intent = new Intent(context, HuaweiCloudActivity.class);
    intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TOP);
    startActivity(intent);
}
private void initPlayer(Bundle savedInstanceState) {
    mPlayerFragment = (PlayerFragment)
getFragmentManager().findFragmentById(R.id.fragment_player);
    callBack = CloudPhoneCallBack.getInstance();
    callBack.setProfileHandler(this::profileReceived);
    CloudPhoneClient.setPlayerCallback(callBack);
    CloudPhoneClient.setPlayerType(HMTP_PLAYER);
    CloudPhoneClient.bindFragment(mPlayerFragment);
    CloudPhoneClient.switchStreamType(CloudPhoneConst.CLOUD_APP_SWITCH_STREAM_TYPE.BOTH);
}
```

Callback回调列表以及onFailure回调异常码。

```
public interface Callback {
/**
* 串流成功返回，调用start()开始串流方法后 成功就会返回
*/
void onSuccess();

/**
* 串流断开,主动或被动 可认为是串流最后一次回调 可以收到回调后处理释放资源
*/
void onTerminated();

/**
* 数据通道串流成功回调
**/
@Override
public void onStreamStarted() {
    if(startHandler!=null){
        startHandler.accept(null);
    }
}
/**
* 开始串流start()失败以及串流中失败异常回调
*
* @param code    错误码
* @param message 错误消息
*/
void onFailure(int code, String message);

/**
* 云机屏幕方向改变
*
* @param orientation 参考Android横竖屏 {@link
android.content.pm.ActivityInfo#SCREEN_ORIENTATION_LANDSCAPE} or {@link
android.content.pm.ActivityInfo#SCREEN_ORIENTATION_PORTRAIT}
*/
void onOrientationChange(int orientation);
```

```
/**
* 按固定周期的回调，展示实时串流的信息，包括分辨率，码率，帧率
*
* @param profile {"fps":"30","br":"2.24Mb","rtt":"14ms","resolution":"720x1280"}
*/
void onProfileReceived(String profile);

/**
* 客户端发起分辨率修改接口setProfile之后，产生的回调，展示修改结果
*
* @param level
* @param isForceDSD 是否被强制降挡 当setProfile的level不支持 需要降低档位level
*/
void onProfileChanged(int level,boolean isForceDSD);


/**
* 云侧扩展消息通道
*
* @param act     上下文，自定义UI时为空
* @param message 消息体
*/
void onExtMessageReceived(Activity act, String message);

/**
* "服务端主动断流回调（典型场景：多端登录时，云侧回调给端侧的踢出串流信息，收到回调时，客户端需要提示有其他端登录）
* @param message 提示信息
*/
void onKicked(String message);
}
```

onFailure异常码列表。

```
/**
* 内部错误
*/
public static final int CLOUD_APP_RET_CODE_INTERNAL_ERROR = 100000;//内部错误
/**
* 无效的启动参数
*/
public static final int CLOUD_APP_RET_CODE_INVALID_PARAMS = 100001;//无效的启动参数
/**
* token无效，检查启动参数accessToken或secretKey是否正确
*/
public static final int CLOUD_APP_RET_CODE_INVALID_TOKEN = 100002;//token无效，检查启动参数accessToken或secretKey是否正确
/**
* sdk版本不匹配，更新最新的sdk版本
*/
public static final int CLOUD_APP_RET_CODE_INVALID_VERSION = 100003;//sdk版本不匹配，更新最新的sdk版本
/**
* 应用未找到，检查云机是否安装应用
*/
public static final int CLOUD_APP_RET_CODE_GAME_NOT_FOUND = 100004;//应用未找到，检查云机是否安装应用
/**
* 应用启动失败，检查云机安装应用是否正常运行
*/
public static final int CLOUD_APP_RET_CODE_GAME_LAUNCH_FAIL = 100005;//应用启动失败，检查云机安装应用是否正常运行
/**
* 用户主动退出
*/
public static final int CLOUD_APP_RET_CODE_USER_EXIT = 100008;//用户主动退出
/**
* 用户重复登录
*/
public static final int CLOUD_APP_RET_CODE_DUPLICATE_USER = 100009;//用户重复登录
/**
```

```
    * 后端主动断开streaming
    */
    public static final int CLOUD_APP_RET_STOP_STREAMING = 100010;//后端主动断开streaming

    /**
    * 连接服务器失败
    */
    public static final int CLOUD_APP_RET_CODE_CONNECT_FAIL = 110000;//连接服务器失败
    /**
    * 请求异常，服务器未能正常响应(如404、500...)
    */
    public static final int CLOUD_APP_RET_CODE_CONNECT_ERROR = 110001;//请求异常，服务器未能正常
    响应(如404、500...)
    /**
    * player与云机断开连接、断线
    */
    public static final int CLOUD_APP_RET_CODE_DROP_LINE = 110003;//player与云机断开连接、断线
    /**
    * 心跳超时
    */
    public static final int CLOUD_APP_RET_CODE_HEARTBEAT_TIMEOUT = 110004;//心跳超时
    /**
    * 云机服务异常
    */
    public static final int CLOUD_APP_RET_CODE_STRM_SERVER_ERROR = 110006;//云机服务异常

    /**
    * 云机服务忙
    */
    public static final int CLOUD_APP_RET_CODE_STRM_SERVICE_BUSY = 5021107;//云机服务忙
```

----结束

1.5 代码示例

串流页面：

```
public class HuaweiCloudActivity extends AppCompatActivity {
    private static final String TAG = "HuaweiCloudActivity";
    private  PlayerFragment mPlayerFragment;
    private LinearLayout float_opt;
    private CountDownTimer inactivityTimer;
    private static final long INACTIVITY_TIMEOUT = 3000; // 3秒
    private int failedNum;
    private Bundle extras;
    private LoadingLayout mLoadingLayout;
    private static Integer timer = 0;
    private  CloudPhoneCallBack callBack ;
    private Handler mHandler = new Handler();
    @SuppressLint("MissingInflatedId")
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        initView(savedInstanceState);
        ActivityManager.addActivity(this);
    }
    private void initView(Bundle savedInstanceState){
        View decorView = getWindow().getDecorView();
        Log.i(TAG, "======onCreate(Bundle savedInstanceState)");
        int uiOptions =
                View.SYSTEM_UI_FLAG_FULLSCREEN          // 隐藏状态栏
                        | View.SYSTEM_UI_FLAG_HIDE_NAVIGATION   // 隐藏导航栏
                        | View.SYSTEM_UI_FLAG_IMMERSIVE_STICKY; // 沉浸式模式（触摸后自动恢复）
        decorView.setSystemUiVisibility(uiOptions);
        // 监听变化，防止退出沉浸模式
        decorView.setOnSystemUiVisibilityChangeListener(visibility -> {
            if ((visibility & View.SYSTEM_UI_FLAG_FULLSCREEN) == 0) {
```

```
                decorView.setSystemUiVisibility(uiOptions);
            }
        });
        setContentView(R.layout.activity_huawei_cloud);
        initPlayer(savedInstanceState);
    }
    @Override
    public boolean dispatchGenericMotionEvent(MotionEvent event) {
        // 手柄事件
        if (event.isFromSource(InputDevice.SOURCE_JOYSTICK) && event.getAction() ==
MotionEvent.ACTION_MOVE) {
            CloudPhoneClient.sendJoypadEvent(event);
            // 轴MotionEvent事件会额外触发对应的keyEvent
            return true;
        }
        return super.onGenericMotionEvent(event);
    }
    private static final Set<Integer> JOYSTICK_KEYCODE_SET = new HashSet<>(
            Arrays.asList(
                    // 方向键
                    KeyEvent.KEYCODE_DPAD_UP,
                    KeyEvent.KEYCODE_DPAD_DOWN,
                    KeyEvent.KEYCODE_DPAD_LEFT,
                    KeyEvent.KEYCODE_DPAD_RIGHT,
                    // ABXY
                    KeyEvent.KEYCODE_BUTTON_A,
                    KeyEvent.KEYCODE_BUTTON_B,
                    KeyEvent.KEYCODE_BUTTON_X,
                    KeyEvent.KEYCODE_BUTTON_Y,
                    // 左右肩键
                    KeyEvent.KEYCODE_BUTTON_L1,
                    KeyEvent.KEYCODE_BUTTON_L2,
                    KeyEvent.KEYCODE_BUTTON_R1,
                    KeyEvent.KEYCODE_BUTTON_R2,
                    // 左右摇杆按下
                    KeyEvent.KEYCODE_BUTTON_THUMBL,
                    KeyEvent.KEYCODE_BUTTON_THUMBR,
                    // 通用功能键
                    KeyEvent.KEYCODE_BUTTON_SELECT,
                    KeyEvent.KEYCODE_BUTTON_START
            ));
    @Override
    public boolean dispatchKeyEvent(KeyEvent event) {
        // 转发手柄对应的按键
        int keyCode = event.getKeyCode();
        if (JOYSTICK_KEYCODE_SET.contains(keyCode)) {
            CloudPhoneClient.sendKeyEvent(keyCode, event.getAction());
            return true;
        }
        return super.dispatchKeyEvent(event);
    }
    @Override
    protected void onResume() {
        Log.i(TAG, "======onResume()");
        super.onResume();
        // Activity恢复时重置计时器
    }
    @Override
    protected void onNewIntent(Intent intent) {
        super.onNewIntent(intent);
        CloudPhoneClient.bindFragment(mPlayerFragment);
        CloudPhoneClient.switchStreamType(CloudPhoneConst.CLOUD_APP_SWITCH_STREAM_TYPE.BOTH);
    }
    @Override
    protected void onPause() {
        Log.i(TAG, "======onPause()");
        super.onPause();
        // Activity暂停时取消计时器
//        inactivityTimer.cancel();
```

```
    }
    @Override
    protected void onStop() {
        Log.i(TAG, "======onStop()");
        super.onStop();
    }
    @Override
    protected void onStart() {
        Log.i(TAG, "======onStart()");
        super.onStart();
    }
    private void initPlayer(Bundle savedInstanceState) {
        mPlayerFragment = (PlayerFragment)
getFragmentManager().findFragmentById(R.id.fragment_player);
        callBack = CloudPhoneCallBack.getInstance();
        callBack.setProfileHandler(this::profileReceived);
        CloudPhoneClient.setPlayerCallback(callBack);
        CloudPhoneClient.setPlayerType(HMTP_PLAYER);
//        CloudPhoneClient.enableAudioKeeping(true);
        CloudPhoneClient.bindFragment(mPlayerFragment);
        CloudPhoneClient.switchStreamType(CloudPhoneConst.CLOUD_APP_SWITCH_STREAM_TYPE.BOTH);
//        CloudPhoneClient.switchStreamType(CloudPhoneConst.CLOUD_APP_SWITCH_STREAM_TYPE.BOTH);
        //执行串流的，串流结果成功会从onSuccsss返回通知，失败则是onFailure带code
//        CloudPhoneClient.start(mPlayerFragment, extras);
    }
    private void logBundleContents(Bundle bundle) {
        if (bundle == null) return;
        Log.i(TAG, "Bundle contains " + bundle.size() + " items:");
        for (String key : bundle.keySet()) {
            Log.i(TAG, key + " = " + bundle.get(key));
        }
    }
    @Override
    protected void onSaveInstanceState(Bundle outState) {
        Log.i(TAG, "======onSaveInstanceState()");
        NetworkInfoComp.netWorkMap.remove("帧率");
        NetworkInfoComp.netWorkMap.remove("下行流量");
        NetworkInfoComp.netWorkMap.remove("延迟");
        super.onSaveInstanceState(outState);
        outState.putBundle("EXTRA_BUNDLE", getIntent().getExtras());
    }
    @Override
    protected void onDestroy() {
        Log.i(TAG, "======onDestroy()");
        super.onDestroy();
        CloudPhoneClient.setUnPlayerCallback(callBack);
        mHandler.removeCallbacksAndMessages(null);
        ActivityManager.removeActivity(this);
    }
    private boolean isFmActive() {
        final AudioManager am = (AudioManager) getSystemService(Context.AUDIO_SERVICE);
        if (am == null) {
            Log.w(TAG, "isFmActive: couldn't get AudioManager reference");
            return false;
        }
        return am.isMusicActive();
    }
    /**
     * 再次连接
     */
    private void linkAgain() {
        //执行串流的，串流结果成功会从onSuccsss返回通知，失败则是onFailure带code
        CloudPhoneClient.stop();
        CloudPhoneClient.start(mPlayerFragment, extras);
        onResume();
    }
    public void profileReceived(String s) {
        NetInfo netInfo = JSONUtil.fromJson(s, NetInfo.class);
        int rrt = Integer.parseInt(netInfo.getRtt().replaceAll("[^0-9]", ""));
```



```
        if(rrt>100){
//            CloudPhoneClient.setProfile(1);
            timer = 3;
        } else if (rrt<80) {
            timer = Math.max(0, timer - 1);
            if(timer==0){
//                CloudPhoneClient.setProfile(0);
            }
        }
        if(AppApplication.getInstance().getDebugMode()) {
            String numberStr = netInfo.getBr().replaceAll("[^0-9.]", "");
            String letters = netInfo.getBr().replaceAll("[^a-zA-Z]", "").toUpperCase();
            double v = Double.parseDouble(numberStr) / 8;
            DecimalFormat df = new DecimalFormat("#0.00");
            String formattedResult = df.format(v);
            List<NetworkConfig> networkConfigList = AppApplication.getInstance().getDebugConfig();
            if(networkConfigList.stream().anyMatch(item->item.getLabel().equals("发送帧"))){
                NetworkInfoComp.netWorkMap.put("发送帧", netInfo.getFps());
            }
            if(networkConfigList.stream().anyMatch(item->item.getLabel().equals("下行流量"))){
                NetworkInfoComp.netWorkMap.put("下行流量",formattedResult+letters);
            }
            if(networkConfigList.stream().anyMatch(item->item.getLabel().equals("网络延迟"))){
                NetworkInfoComp.netWorkMap.put("网络延迟", netInfo.getRtt());
            }
            NetworkInfoComp.reloadGirdView();
        }
        Log.i(TAG, "======onProfileReceived(String s).s="+s);
    }
}
```

回调处理对象：

```
package com.KooCabin.service;
import android.app.Activity;
import android.util.Log;
import androidx.core.util.Consumer;
import com.KooCabin.activity.HuaweiCloudActivity;
import com.KooCabin.base.BasePresenter;
import com.KooCabin.ui.playerfloat.FloatWindowMainView;
import com.cloudphone.client.api.CloudPhoneClient;
import com.cloudphone.client.api.CloudPhoneConst;
import com.cloudphone.client.bean.AppInfo;
import com.cloudphone.client.bean.AppLifeCycle;
import com.cloudphone.client.bean.AppOperateResponse;
import com.cloudphone.client.bean.DiskUsageInfo;
import com.cloudphone.client.bean.GetAppsRsp;
import com.huawei.kp.KPClientStat;
import com.huawei.kp.KPCloudStat;
import lombok.Setter;
@Setter
public class CloudPhoneCallBack implements CloudPhoneClient.Callback,
CloudPhoneClient.Callback.AppCallback {
    private static volatile CloudPhoneCallBack instance;
    // 私有构造方法
    private CloudPhoneCallBack() {
        // 防止外部实例化
    }
    // 公共静态获取实例方法
    public static CloudPhoneCallBack getInstance() {
        if (instance == null) {
            synchronized (CloudPhoneCallBack.class) {
                if (instance == null) {
                    instance = new CloudPhoneCallBack();
                }
            }
        }
        return instance;
    }
    private static final String TAG = "CloudPhoneCallback";
```

```
private Consumer<AppOperateResponse<GetAppsRsp>> appHandler;
private Consumer<Void> startHandler;
private Consumer<AppOperateResponse<String>> unInstallHandler;
private Consumer<AppOperateResponse<String>> unLoadAppHandler;
private Consumer<AppOperateResponse<String>> startAppHandler;
private Consumer<AppOperateResponse<String>> closeAppHandler;
private Consumer<String> profileHandler;
private Consumer<Void> streamSuccessHandler;
private Consumer<AppOperateResponse<DiskUsageInfo>> usageHandler;
@Override
public void onGetInstalledApps(AppOperateResponse<GetAppsRsp> appOperateResponse) {
    if(appHandler != null) {
        appHandler.accept(appOperateResponse);
    }
    Log.i(TAG, "onGetInstalledApps success");
}
@Override
public void onSuccess() {
    if(streamSuccessHandler!=null){
        streamSuccessHandler.accept(null);
    }
}
@Override
public void onTerminated() {
}
@Override
public void onFailure(int i, String s) {
}
@Override
public void onMenuOnClick(Activity activity, int i) {
}
@Override
public String getUrl() {
    return "";
}
@Override
public void lifecycle(Activity activity, int i) {
}
@Override
public void onOrientationChange(int i) {
}
@Override
public void onSlotsInfo(String s) {
}
@Override
public void onRoomInfoUpdate(String s) {
}
@Override
public void onPermissionRequest(String s) {
}
@Override
public void onPermissionResult(String s) {
}
@Override
public void onProfileReceived(String s) {
    if(profileHandler!=null){
        profileHandler.accept(s);
    }
}
@Override
public void onProfileChanged(int i, boolean b) {
}
@Override
public void onExitConfirm(int i) {
}
@Override
public void onUserExit() {
}
@Override
```

```
    public void onExtMessageReceived(Activity activity, String s) {
    }
    @Override
    public void onScreenshotUpdate(String s, String s1) {
    }
    @Override
    public void onScreenshotAuthFailed(String s) {
    }
    @Override
    public void onKicked(String s) {
    }
    @Override
    public void onPermissionGranted(String[] strings) {
    }
    @Override
    public void onPermissionDenied(String[] strings) {
    }
    @Override
    public void onHttpResponse(String s) {
    }
    @Override
    public void onActionMessage(String s) {
    }
    @Override
    public void onPerfLog(int i, String s) {
    }
    @Override
    public void onSensorInjectEvent(String s) {
    }
    @Override
    public void onCloudStatistics(KPCloudStat kpCloudStat) {
    }
    @Override
    public void onClientStatistics(KPClientStat kpClientStat) {
    }
    @Override
    public void onStreamStarted() {
        Log.i("11111", "onStreamStarted======onSuccess()");
//        CloudPhoneClient.switchStreamType(CloudPhoneConst.CLOUD_APP_SWITCH_STREAM_TYPE.BOTH);
        if(startHandler!=null){
            startHandler.accept(null);
        }
    }
    @Override
    public void onCloseApp(AppOperateResponse<String> appOperateResponse) {
        Log.i("11111", "======onCloseApp()");
        if(closeAppHandler!=null){
            closeAppHandler.accept(appOperateResponse);
        }
    }
    @Override
    public void onStartApp(AppOperateResponse<String> appOperateResponse) {
        Log.i("CALLBACK:","=====onStartApp=======");
        if(startAppHandler!=null){
            startAppHandler.accept(appOperateResponse);
        }
    }
    @Override
    public void onGetRunningApps(AppOperateResponse<GetAppsRsp> appOperateResponse) {
        if(appHandler!=null){
            appHandler.accept(appOperateResponse);
        }
    }
    @Override
    public void onRestartApp(AppOperateResponse<String> appOperateResponse) {
        Log.i("11111", "======onRestartApp()");
    }
    @Override
    public void onUninstallApp(AppOperateResponse<String> appOperateResponse) {
```

```
        if(unInstallHandler!=null){
            unInstallHandler.accept(appOperateResponse);
        }
    }
    @Override
    public void onLoadApp(AppOperateResponse<String> appOperateResponse) {
        if(unLoadAppHandler!=null){
            unLoadAppHandler.accept(appOperateResponse);
        }
    }
    @Override
    public void onGetDiskUsage(AppOperateResponse<DiskUsageInfo> appOperateResponse) {
        if(usageHandler!=null){
            usageHandler.accept(appOperateResponse);
        }
    }
    @Override
    public void onAPKInstallState(AppOperateResponse<AppInfo> appOperateResponse) {
        Log.i("11111", "======onAPKInstallState()");
    }
    @Override
    public void onAppLifeCycle(AppOperateResponse<AppLifeCycle> appOperateResponse) {
        Log.i("11111", "======onAppLifeCycles()");
    }
}
```

串流页面布局：

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/player_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment
        android:id="@+id/fragment_player"
        android:name="com.nbc.acsdk.widget.PlayerFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
    <!--串流页面其他自定义UI自行添加-->
    <!--    悬浮操作-->
    <com.KooCabin.ui.NetworkInfoComp
        android:id="@+id/network_info"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="top|end"
         />
</FrameLayout>
```

1.6 常用调试方法

调试工具

- 安卓手机、平板、车机
- Windows系统电脑
- USB数据线

Android Studio 调试器

1. 断点调试。

   在Java/Kotlin/C++代码行号旁单击设置断点，运行Debug模式启动应用。支持条件断点、日志断点、异常捕获断点。

2. 变量监控。
   Debug窗口查看实时变量值，支持表达式求值。
3. 多进程调试。
   通过Attach debugger to Android process动态附加到已运行进程。

1.7 常见问题

- 如何获取监控参数：如网络延迟，下行流量，fps信息。
  在HuaweiCloudActivity的回调方法中，云端会每秒回传以下示例信息：

```
@Override
public void onProfileReceived(String s) {
    NetInfo netInfo = JSONUtil.fromJson(s, NetInfo.class);
    int rrt = Integer.parseInt(netInfo.getRtt().replaceAll("[^0-9]", ""));
    if(rrt>100){
        CloudPhoneClient.setProfile(1);
        timer = 3;
    } else if (rrt<80) {
        timer = Math.max(0, timer - 1);
        if(timer==0){
            CloudPhoneClient.setProfile(0);
        }
    }
    if(AppApplication.getInstance().getDebugMode()) {
        String numberStr = netInfo.getBr().replaceAll("[^0-9.]", "");
        String letters = netInfo.getBr().replaceAll("[^a-zA-Z]", "").toUpperCase();
        double v = Double.parseDouble(numberStr) / 8;
        DecimalFormat df = new DecimalFormat("#0.00");
        String formattedResult = df.format(v);
        List<NetworkConfig> networkConfigList = AppApplication.getInstance().getDebugConfig();
        if(networkConfigList.stream().anyMatch(item->item.getLabel().equals("发送帧"))){
            NetworkInfoComp.netWorkMap.put("发送帧", netInfo.getFps());
        }
        if(networkConfigList.stream().anyMatch(item->item.getLabel().equals("下行流量"))){
            NetworkInfoComp.netWorkMap.put("下行流量",formattedResult+letters);
        }
        if(networkConfigList.stream().anyMatch(item->item.getLabel().equals("网络延迟"))){
            NetworkInfoComp.netWorkMap.put("网络延迟", netInfo.getRtt());
        }
        NetworkInfoComp.reloadGirdView();
    }
    Log.i(TAG, "======onProfileReceived(String s).s="+s);
}
```

- 如何在返回桌面后保持声音。
  串流开始前调用：

```
CloudPhoneClient.enableAudioKeeping(true)
```

- 重复串流黑屏的问题。
  如果不需要保持声音在后台播放，在关闭串流页面时调用stop()方法；或者如音乐播放类需要在后台播放，在重新串流前先调用stop方法：

```
CloudPhoneClient.stop()
```

- device_token失效导致黑屏的问题。
  device_token重复申请会导致上一次申请的device_token失效，正在串流的页面就会中断或者黑屏，建议每次串流前获取device_token。