云手机

SDK 参考

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2025 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【商标声明】

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

SDK 参考
- SDK 概览
- Web SDK
  - SDK 集成指引
  - 接口说明
  - 事件回调
- Windows SDK
  - SDK 集成指引
  - 接口说明
  - 事件回调
- iOS SDK
  - SDK 集成指引
  - 接口说明
  - 事件回调
- Android SDK
  - SDK 集成指引
  - 接口说明
  - 事件回调

SDK 参考
SDK 概览

最近更新时间：2025-09-03 14:10:22

腾讯云手机为您提供对接云手机服务的客户端 SDK，支持 Web、Windows、iOS 和 Android 终端接入，助力开发者快速开发搭建开始自己的业务。本文档提供各端的 SDK 下载方式、SDK 接口说明文档及快速开始 Demo。

SDK 下载

客户端	SDK 下载地址	开发者文档	快速开始 Demo
Web	JS SDK	Web SDK 说明文档	Web Demo
Windows	Windows SDK	Windows SDK 说明文档	Windows Demo
iOS	iOS SDK	iOS SDK 说明文档	iOS demo
Android	通过 Maven 集成	Android SDK 说明文档	Android Demo

Web SDK
SDK 集成指引

最近更新时间：2025-09-03 14:10:22

此 SDK 提供在 Web 端快速接入云手机 Paas 平台的能力，可以通过 SDK 操作云手机实例。

运行环境

- iOS 11.0 及以上系统版本。
- Android 浏览器：需要支持 WebRTC 的浏览器内核。
- PC Web 浏览器：内核为 Chrome 90+ 的浏览器，以及 Safari 11 及以上版本。

快速接入

获取 SDK

SDK	下载地址	SDK 说明文档	版本发布日志
JavaScript SDK	GitHub下载	接口说明	Release Notes

引入 SDK

> **说明：**
> TCGSDK export UMD module 已默认将 TCGSDK 挂载到 Window 下，可直接使用 TCGSDK 对象，也可以自行初始化实例。SDK 说明可参考 SDK 声明文件 。

方法一： script 标签引入。

```
<script src="path/to/sdk" charset="utf-8"></script>
```

方法二： 模块形式引入，默认 export module 为 UMD。

```
import TCGSDK, {CloudGamingWebSDK} from 'path-to-TCGSDK';

// 直接使用 TCGSDK 或 new CloudGamingWebSDK()
```

启动 SDK

```
// 以下均为伪代码
```

```
// 调用 云API 创建安卓实例访问Token
// https://cloud.tencent.com/document/api/1162/119708

// 参考 https://github.com/tencentyun/cloudgame-js-
sdk/blob/master/samples/android_instance/src/android_instance/network.ts
const {AccessInfo, Token} = await
CreateAndroidInstancesAccessToken({AndroidInstanceIds});

TCGSDK.init({
  appid: 1234567,
  mount: 'mount-point', // 填入需要挂载入云渲染视图的节点
  accessToken: {
    accessInfo: AccessInfo,
    token: Token
  },
  debugSetting: {
    showLog: true,
  },
  // 连接成功回调
  onConnectSuccess: (res) => {
    console.log('onConnectSuccess', res);
  },
  // 网络中断/被踢触发此回调
  onDisconnect: (res) => {
    console.log('onDisconnect', res);
  },
  // 初始化成功回调
  onInitSuccess: async (res) => {
    console.log('%c onInitSuccess', 'color: red', res);

    // 开始渲染
    TCGSDK.access({
      instanceId: AndroidInstanceIds[0]
    });
  }
});
```

完整 Demo

完整代码可参考 Android Instance React Demo 。

接口说明

最近更新时间：2025-09-03 14:10:22

接口概览

生命周期接口

方法	描述
init	初始化云渲染 SDK，初始化成功会回调 onInitSuccess。
access	启动云渲染，启动成功会回调 onConnectSuccess。
destroy	立即停止云渲染。 该接口为前端的停止，即 destroy 当前的连接。云端会在自动检测到心跳消失后，释放资源。
reconnect	重连。 由于网络问题导致的中断，SDK 会自动重连。 重连策略：每5秒尝试一次，最多重连5次。
setAccessToken	设置 accessToken。 通过 创建安卓实例访问Token 接口获取 AccessToken，可通过该接口设置也可以通过 init 参数传入。

安卓实例操作接口

先通过 TCGSDK.getAndroidInstance() 获取安卓实例，以下为 AndroidInstance 对象接口，接口描述文件可参考 androidInstance.d.ts 。

```
const AndroidInstance = TCGSDK.getAndroidInstance();

AndroidInstance.getInstanceImage({instanceId: 'cai-xxx1'});
```

方法	描述
getInstanceImage	获取实例截图信息。
upload	上传文件到实例。
getInstanceDownloadAddress	获取实例下载地址。

switchIME	切换输入法。
setLocation	设置设备 GPS 信息。
setResolution	设置设备分辨率。
paste	粘贴文本。
sendClipboard	发送文本到剪切板。
shake	摇一摇。
setSensor	设置设备传感器信息。
sendTransMessage	发送 App binder 消息。
describeInstanceProperties	查询实例属性。
modifyInstanceProperties	修改实例属性。
listUserApps	查询已安装第三方应用。
modifyKeepFrontAppStatus	修改前台应用保活状态。
describeKeepFrontAppStatus	查询前台应用保活状态。
unInstallByPackageName	卸载应用。
startApp	启动应用。
stopApp	停止应用。
clearAppData	清除应用数据。
enableApp	启用应用。
disableApp	禁用应用。
startCameraMediaPlay	摄像头播放媒体文件。
stopCameraMediaPlay	摄像头停止播放媒体文件。

版权所有：腾讯云计算（北京）有限责任公司

describeCameraMediaPlayStatus	查询当前摄像头媒体播放状态。
displayCameraImage	摄像头显示图片。
addKeepAliveList	增加后台保活应用。
removeKeepAliveList	移除后台保活应用。
setKeepAliveList	覆盖后台保活应用。
describeKeepAliveList	查询后台保活应用。
clearKeepAliveList	清空后台保活应用。
mute	静音开关。
mediaSearch	搜索媒体库文件。
reboot	重启实例。
listAllApps	查询所有应用列表。
moveAppBackground	关闭应用至后台。
addAppInstallBlackList	新增应用安装黑名单。
removeAppInstallBlackList	移除应用安装黑名单。
setAppInstallBlackList	覆盖应用安装黑名单。
describeAppInstallBlackList	查询应用安装黑名单。
clearAppInstallBlackList	清空应用安装黑名单。
getNavVisibleStatus	获取系统导航栏显示状态。
getSystemMusicVolume	获取系统媒体音量大小。

音视频控制相关接口

方法	描述
setVideoVolume	设置 Video 音量。
getVideoVolume	获取 Video 音量。
playVideo	播放视频。
switchMic	开关麦克风。
switchCamera	开关摄像头。
setVideoOrientation	设置 Video 的旋转角度。

鼠标键盘控制相关接口

方法	描述
sendKeyboardEvent	发送键盘事件。
sendMouseEvent	发送鼠标事件。
mobileTouchMove	发送 touch 事件。

其他工具方法

方法	描述
getRequestId	获取当前连接 requestId。
screenShot	推流视频截图。
sendText	聚焦输入框时快速发送内容，会同时粘贴到剪贴板。

Init 方法参数

参数名称	必选	类型	描述

mount	是	String	页面挂载点的 HTML 元素 ID。
mobileGame	否	Boolean	true 为使用接入手游/云手机。
androidInstance	否	Object	{autoRotateOnPC?: boolean;} 在 PC 端自动旋转。
accessToken	否	Object	{token: String; accessInfo: String} 实例访问 token，通过云 API 获取。
mic	否	Boolean	是否开启本地麦克风。
camera	否	Boolean	是否开启本地摄像头。
loadingText	否	String	加载中的文字提示内容，默认为“正在启动云渲染”。
restartText	否	String	重新连接的文字提示内容，默认为“重新连接”。

接口详细说明

初始化

初始化（init）云渲染 SDK，初始化成功会回调 onInitSuccess。

输入参数

参见 Init 方法参数 。

示例

```
TCGSDK.init({
    mount: 'car-mount-point',
    mobileGame: true,
    debugSetting: {
      showLog: true,
    },
    accessToken,
    loadingText: 'Starting...',
    // 连接成功回调
    onConnectSuccess: () => {
```

```
      console.log('onConnectSuccess');
    },
    // 网络中断/被踢触发此回调
    onDisconnect: () => {
      console.log('onDisconnect');
    },
    onInitSuccess: () => {
      console.log('%c onInitSuccess', 'color: red');

      TCGSDK.access({
        instanceId: instanceIds[0]
      });
    }
  });
```

启动云渲染

启动（access）云渲染，需要 init 内传入 accessToken 或调用 setAccessToken。

输入参数

参数名称	必选	类型	描述
instanceId	是	String	实例 ID。

示例

```
TCGSDK.access({instanceId: 'cai-xxxx-xxxx'});
```

停止云渲染

立即停止（destroy）云渲染，该接口为前端的停止，即 clear 当前的连接（peerConnection）。云端会在自动检测到心跳消失后，释放资源。

输入参数

参数名称	必选	类型
code	否	Number
message	否	String

示例

```
TCGSDK.destroy();
```

重连

重连（reconnect）接口，也可以设置 *init reconnect* 参数由 SDK 自动调用，或根据 onDisconnect 回调 code，结合自身场景，主动调用该接口。
重连策略：每5秒尝试一次，最多重连5次。

示例

```
TCGSDK.reconnect();
```

设置 AccessToken

设置 AccessToken（setAccessToken）。

输入参数

参数名称	必选	类型
accessInfo	是	String
token	是	String

示例

```
TCGSDK.setAccessToken({accessInfo: 'xxx', token: 'xxx'});
```

获取实例截图信息

获取实例截图信息（getInstanceImage）。

输入参数

参数名称	必选	类型	描述
instanceId	是	String	实例 ID。
quality	否	Number	截图质量，取值范围 0-100，默认 20。
screenshot_width	否	Number	截图宽度。

screenshot_height	否	Number	截图高度。

返回值

类型：{ url: string }

参数名称	类型	描述
url	String	截图地址。

示例

```
const {url} = AndroidInstance.getInstanceImage({instanceId: 'cai-xxx1'});
```

上传文件到实例

上传文件到实例（upload）。
默认上传到 /sdcard/Download 目录下，可使用 path 指定上传目录（仅支持/sdcard/ 下目录）。

输入参数

参数名称	必选	类型	描述
instanceId	是	String	实例 ID。
files	是	Object[]	-
files.file	是	File	文件。
files.path	否	String	路径。

返回值

类型：Promise<{Code: number; Message: string; FileStatus: { CloudPath: string; FileName: string }[] | null;}>

参数名称	类型	描述
Code	Number	-
Message	String	-

FileStatus	Object[]	-
FileStatus.CloudPath	String	云端路径。
FileStatus.FileName	String	文件名。

示例

```
const {Code, Message, FileStatus} = await AndroidInstance.upload({instanceId: 'cai-xxx1', files: [{file: file1, path: '/sdcard/xxx/'}, {file: file2, path: '/sdcard/xxx/'}]});
```

获取实例下载地址

获取实例下载地址（getInstanceDownloadAddress）。

输入参数

参数名称	必选	类型	描述
instanceId	是	String	实例 ID。
path	否	String	下载路径。

返回值

类型：{ address: string }

参数名称	类型	描述
address	String	下载地址。

示例

```
const {address} = AndroidInstance.getInstanceDownloadAddress({instanceId: 'cai-xxx1', path: '/sdcard/xxx/'});
```

切换输入法

切换输入法（switchIME）。

输入参数

参数名称	必选	类型	描述
ime	是	String	• cloud：云端输入法。 • local： 本地输入法。

示例

```
AndroidInstance.switchIME({ime: 'local'});
```

设置设备 GPS 信息

设置设备 GPS 信息（setLocation）。

输入参数

类型：{ [InstanceId: string]: { Longitude: number; Latitude: number } }

参数名称	必选	类型	描述
Longitude	是	Number	经度。
Latitude	是	Number	纬度。

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.setLocation({'cai-xxx1': {Longitude:
113.32412, Latitude: 23.124124}, 'cai-xxx2': {Longitude: 114.32412,
Latitude: 24.124124}})
console.log(res['cai-xxx1']);
```

设置设备分辨率

设置设备分辨率（setResolution）。

输入参数

类型：{ [InstanceId: string]: { Width: number; Height: number; DPI?: number } };

参数名称	必选	类型	描述
Width	是	Number	宽。
Height	是	Number	高。
DPI	否	Number	-

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.setResolution({'cai-xxx1': {Width: 720, Height: 1080}, {'cai-xxx2': {Width: 720, Height: 1080}});
console.log(res['cai-xxx1']);
```

粘贴文本

粘贴文本（paste）。

输入参数

类型：{ [InstanceId: string]: { Text: string } }

参数名称	必选	类型	描述
Text	是	String	文本。

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.paste({'cai-xxx1': {Text: 'abc'}, {'cai-xxx2': {Text: '123'}});
console.log(res['cai-xxx1']);
```

发送文本到剪切板

发送文本到剪切板（sendClipboard）。

输入参数

类型：{ [InstanceId: string]: { Text: string } }

参数名称	必选	类型	描述
Text	是	String	文本。

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.sendClipboard({'cai-xxx1': {Text:
'abc'}, {'cai-xxx2': {Text: '123'}});
console.log(res['cai-xxx1']);
```

摇一摇

摇一摇（shake）。

输入参数

类型：{ [InstanceId: string]: { } }

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.shake({'cai-xxx1': {}, {'cai-xxx2':
{}});
console.log(res['cai-xxx1']);
```

设置设备传感器信息

设置设备传感器信息（setSensor）。

输入参数

类型：{ [InstanceId: string]: { Type: 'accelerometer' | 'gyroscope'; Values: number[] } }

参数名称	必选	类型	描述
Type	是	String	传感器 accelerometer（加速器），gyroscope（陀螺仪）。
Values	是	String[]	传感器对应的值，数组长度为 3，分别表示 x/y/z 轴的值。

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.setSensor({'cai-xxx1': {Type:
'accelerometer', Values: [10, 10, 10]},  {'cai-xxx2': {Type:
'gyroscope', Values: [10, 10, 10]}});
console.log(res['cai-xxx1']);
```

发送 App binder 消息

发送 App binder 消息（sendTransMessage）。

输入参数

类型：{ [InstanceId: string]: { PackageName: string; Msg: string } }

参数名称	必选	类型	描述
PackageName	是	String	包名。
Msg	是	String	消息。

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.sendTransMessage({'cai-xxx1':
{PackageName: 'com.example.myapplication', Msg: 'abc123'}});
console.log(res['cai-xxx1']);
```

查询实例属性

查询实例属性（describeInstanceProperties）。

输入参数

类型：{ [InstanceId: string]: {} }

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.describeInstanceProperties({'cai-
xxx1': {}, {'cai-xxx2': {}});
console.log(res['cai-xxx1']);
```

修改实例属性

修改实例属性（modifyInstanceProperties）

输入参数

类型：
{ [InstanceId: string]: Partial<InstanceProperties> }

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.ModifyInstanceProperties({'cai-xxx1':
{
    DeviceInfo: {
          Brand: 'Samsung',
          Model: 'Galaxy S24',
        },
      ProxyInfo: {
        Enabled: true,
        Protocol: 'socks5',
        Host: 'proxy.example.com',
        Port: 1080,
        User: 'user123',
        Password: 'pass123',
      },
      GPSInfo: {
```

```
        Longitude: 121.4737,
        Latitude: 31.2304,
      },
      SIMInfo: {
        State: 1,
        PhoneNumber: '+8613812345678',
        IMSI: '460001234567890',
        ICCID: '89860123456789012345',
      },
      LocaleInfo: {
        Timezone: 'Asia/Shanghai',
      },
      LanguageInfo: {
       Language: 'zh',
       Country: 'CN',
     },
     ExtraProperties: [
       {
         Key: 'custom_property1',
         Value: 'value1',
       },
       {
         Key: 'custom_property2',
         Value: 'value2',
       },
     ]
   }})
console.log(res['cai-xxx1']);
```

查询已安装第三方应用

查询已安装第三方应用（listUserApps）。

输入参数

类型：{ [InstanceId: string]: { } }

返回值

类型：Promise<ListUserAppsResponse>

参数名称	类型	描述
AppList	Objec	-

	t[]	
AppList.FirstInstallTimeMs	Number	首次安装时间。
AppList.Label	String	标签。
AppList.LastUpdateTimeMs	Number	最近更新时间。
AppList.PackageName	String	包名。
AppList.VersionName	String	版本名。

示例

```
const res = await AndroidInstance.listUserApps({'cai-xxx1': {}, {'cai-xxx2': {}})
console.log(res['cai-xxx1']);
```

修改前台应用保活状态

修改前台应用保活状态（modifyKeepFrontAppStatus）。

输入参数

类型：{ [InstanceId: string]: { PackageName: string; Enable: boolean; RestartInterValSeconds: number; } }

参数名称	必选	类型	描述
PackageName	是	String	包名。
Enable	是	Boolean	-
RestartInterValSeconds	是	Number	重新拉起最长间隔。

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.modifyKeepFrontAppStatus({'cai-xxx1':
{"PackageName": "com.example.app", "Enable": true,
"RestartInterValSeconds": 5}});
console.log(res['cai-xxx1']);
```

查询前台应用保活状态

查询前台应用保活状态（describeKeepFrontAppStatus）。

输入参数

类型：{ [InstanceId: string]: { } }

返回值

类型：Promise<{ PackageName: string; Enable: boolean; RestartInterValSeconds: number }>;

参数名称	类型	描述
PackageName	String	包名。
Enable	Boolean	-
RestartInterValSeconds	String	重新拉起最长间隔。

示例

```
const res = await AndroidInstance.describeKeepFrontAppStatus({'cai-
xxx1': {}});
console.log(res['cai-xxx1']);
```

卸载应用

卸载应用（unInstallByPackageName）。

输入参数

类型：{ [InstanceId: string]: { PackageName: string; } }

参数名称	必选	类型	描述

PackageName	是	String	包名。

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.unInstallByPackageName({'cai-xxx1':
{"PackageName": "com.unwanted.app"}});
console.log(res['cai-xxx1']);
```

启动应用

启动应用（startApp）。

输入参数

类型：{ [InstanceId: string]: { PackageName: string; ActivityName: string } }

参数名称	必选	类型	描述
PackageName	是	String	包名。
ActivityName	是	String	-

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.startApp({'cai-xxx1': {"PackageName":
"com.launch.app", "ActivityName": "MainActivity"}});
console.log(res['cai-xxx1']);
```

停止应用

停止应用（stopApp）。

输入参数

类型：{ [InstanceId: string]: { PackageName: string; } }

参数名称	必选	类型	描述
PackageName	是	String	包名。

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.stopApp({'cai-xxx1': {"PackageName":
"com.running.app"}});
console.log(res['cai-xxx1']);
```

清除应用数据

清除应用数据（clearAppData）。

输入参数

类型：{ [InstanceId: string]: { PackageName: string; } }

参数名称	必选	类型	描述
PackageName	是	String	包名。

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.clearAppData({'cai-xxx1':
{"PackageName": "com.data.app"}});
console.log(res['cai-xxx1']);
```

启用应用

启用应用（enableApp）。

输入参数

类型：{ [InstanceId: string]: { PackageName: string; } }

参数名称	必选	类型	描述
PackageName	是	String	包名。

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.enableApp({'cai-xxx1': {"PackageName":
"com.data.app"}});
console.log(res['cai-xxx1']);
```

禁用应用

禁用应用（disableApp）。

输入参数

类型：{ [InstanceId: string]: { PackageName: string; } }

参数名称	必选	类型	描述
PackageName	是	String	包名。

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.disableApp({'cai-xxx1':
{"PackageName": "com.data.app"}});
console.log(res['cai-xxx1']);
```

摄像头播放媒体文件

摄像头播放媒体文件（startCameraMediaPlay）

输入参数

类型：{ [InstanceId: string]: { FilePath: string; Loops: number }; }

参数名称	必选	类型	描述
FilePath	是	String	文件路径。
Loops	是	String	循环次数，负数表示无限循环。

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.startCameraMediaPlay({'cai-xxx1':
{"FilePath": "/sdcard/video.mp4", "Loops": 3}});
console.log(res['cai-xxx1']);
```

摄像头停止播放媒体文件

摄像头停止播放媒体文件（stopCameraMediaPlay）。

输入参数

类型：{ [InstanceId: string]: { }; }

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.stopCameraMediaPlay({'cai-xxx1': {}});
console.log(res['cai-xxx1']);
```

查询当前摄像头媒体播放状态

查询当前摄像头媒体播放状态（describeCameraMediaPlayStatus）

输入参数

类型：{ [InstanceId: string]: { }; }

返回值

类型：Promise<{ PackageName: string; Enable: boolean; RestartInterValSeconds: number }>;

参数名称	类型	描述

FilePath	String	包名。
Loops	Number	循环次数，负数表示无限循环。

示例

```
const res = await AndroidInstance.describeCameraMediaPlayStatus({'cai-xxx1': {}});
console.log(res['cai-xxx1']);
```

摄像头显示图片

摄像头显示图片（displayCameraImage）。

输入参数

类型：{ [InstanceId: string]: { FilePath: string; }; }

参数名称	必选	类型	描述
FilePath	是	String	文件路径。

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.displayCameraImage({'cai-xxx1': {"FilePath": "/sdcard/image.jpg"}});
console.log(res['cai-xxx1']);
```

增加后台保活应用

增加后台保活应用（addKeepAliveList）。

输入参数

类型：{ [InstanceId: string]: { AppList: string[] } }

参数名称	必选	类型

AppList	是	String[]

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.addKeepAliveList({'cai-xxx1':
{"AppList": ["com.wechat", "com.alipay", "com.dingtalk"]}});
console.log(res['cai-xxx1']);
```

移除后台保活应用

移除后台保活应用（removeKeepAliveList）。

输入参数

类型：{ [InstanceId: string]: { AppList: string[] } }

参数名称	必选	类型
AppList	是	String[]

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.removeKeepAliveList({'cai-xxx1':
{"AppList": ["com.wechat", "com.alipay", "com.dingtalk"]}});
console.log(res['cai-xxx1']);
```

覆盖后台保活应用

覆盖后台保活应用（setKeepAliveList）。

输入参数

类型：{ [InstanceId: string]: { AppList: string[] } }

参数名称	必选	类型
AppList	是	String[]

版权所有：腾讯云计算（北京）有限责任公司

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.setKeepAliveList({'cai-xxx1':
{"AppList": ["com.wechat", "com.alipay", "com.dingtalk"]}});
console.log(res['cai-xxx1']);
```

查询后台保活应用

查询后台保活应用（describeKeepAliveList）。

输入参数

类型：{ [InstanceId: string]: { }; }

返回值

类型：Promise<{ AppList: string[] }>;

参数名称	类型
AppList	String[]

示例

```
const res = await AndroidInstance.describeKeepAliveList({'cai-xxx1':
{}});
console.log(res['cai-xxx1']);
```

清空后台保活应用

清空后台保活应用（clearKeepAliveList）。

输入参数

类型：{ [InstanceId: string]: { }; }

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.clearKeepAliveList({'cai-xxx1': {}});
console.log(res['cai-xxx1']);
```

静音开关

静音开关（mute）。

输入参数

类型：{ [InstanceId: string]: { Mute: boolean } }

参数名称	必选	类型	描述
Mute	是	Boolean	静音。

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.mute({'cai-xxx1': {Mute: true}});
console.log(res['cai-xxx1']);
```

媒体库文件搜索

媒体库文件搜索（mediaSearch）。

输入参数

类型：{ [InstanceId: string]: { Keyword: string } }

参数名称	必选	类型
Keyword	是	String

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.mediaSearch({'cai-xxx1': {"Keyword":
"abc"}});
```

```
console.log(res['cai-xxx1']);
```

重启实例

重启实例（reboot）。

输入参数

类型：{ [InstanceId: string]: { }; }

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.reboot({'cai-xxx1': {}});
console.log(res['cai-xxx1']);
```

查询所有应用列表

查询所有应用列表（listAllApps）。

输入参数

类型：{ [InstanceId: string]: { }

返回值

类型：Promise<ListUserAppsResponse>

参数名称	类型	描述
AppList	Object[]	应用列表。
AppList.FirstInstallTimeMs	Number	首次安装时间。
AppList.Label	String	标签。
AppList.LastUpdateTimeMs	Number	最近更新时间。
AppList.PackageName	String	包名。

AppList.VersionName	String	版本名。

示例

```
const res = await AndroidInstance.listAllApps({'cai-xxx1': {}});
console.log(res['cai-xxx1']);
```

关闭应用至后台

关闭应用至后台（moveAppBackground）。

输入参数

类型：{ [InstanceId: string]: { }; }

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.moveAppBackground({'cai-xxx1': {}});
console.log(res['cai-xxx1']);
```

新增应用安装黑名单

新增应用安装黑名单（addAppInstallBlackList）。

输入参数

类型：{ [InstanceId: string]: { AppList: string[] } }

参数名称	必选	类型
AppList	是	String[]

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.addAppInstallBlackList({'cai-xxx1':
{"AppList": ["com.wechat", "com.alipay", "com.dingtalk"]}});
```

```
console.log(res['cai-xxx1']);
```

移除应用安装黑名单

移除应用安装黑名单（removeAppInstallBlackList）。

输入参数

类型：{ [InstanceId: string]: { AppList: string[] } }

参数名称	必选	类型
AppList	是	String[]

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.removeAppInstallBlackList({'cai-xxx1':
{"AppList": ["com.wechat", "com.alipay", "com.dingtalk"]}});
console.log(res['cai-xxx1']);
```

覆盖应用安装黑名单

覆盖应用安装黑名单（setAppInstallBlackList）。

输入参数

类型：{ [InstanceId: string]: { AppList: string[] } }

参数名称	必选	类型
AppList	是	String[]

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.setAppInstallBlackList({'cai-xxx1':
{"AppList": ["com.wechat", "com.alipay", "com.dingtalk"]}});
console.log(res['cai-xxx1']);
```

查询应用安装黑名单

查询应用安装黑名单（describeAppInstallBlackList）。

输入参数

类型：{ [InstanceId: string]: { }; }

返回值

类型：Promise<{ AppList: string[] }>;

参数名称	类型
AppList	String[]

示例

```
const res = await AndroidInstance.describeAppInstallBlackList({'cai-
xxx1': {}});
console.log(res['cai-xxx1']);
```

清空应用安装黑名单

清空应用安装黑名单（clearAppInstallBlackList）。

输入参数

类型：{ [InstanceId: string]: { }; }

返回值

类型：Promise<BatchTaskResponse>

示例

```
const res = await AndroidInstance.clearAppInstallBlackList({'cai-xxx1':
{}});
console.log(res['cai-xxx1']);
```

获取系统导航栏显示状态

获取系统导航栏显示状态（getNavVisibleStatus）。

输入参数

类型：{ [InstanceId: string]: { }; }

返回值

类型：Promise<{ Visible: boolean; }>;

参数名称	类型
Visible	Boolean

示例

```
const res = await AndroidInstance.getNavVisibleStatus({'cai-xxx1': {}});
console.log(res['cai-xxx1']);
```

获取系统媒体音量大小

获取系统媒体音量大小（getSystemMusicVolume）。

输入参数

类型：{ [InstanceId: string]: { }; }

返回值

类型：Promise<{ Volume: number; }>;

参数名称	类型	描述
Volume	Number	音量。

示例

```
const res = await AndroidInstance.getSystemMusicVolume({'cai-xxx1':
{}});
console.log(res['cai-xxx1']);
```

批量操作返回值

批量操作返回值（BatchTaskResponse）。

返回值

类型：[InstanceId: string]: {Code: number; Msg: string;};

参数名称	类型	描述
Code	Number	• Code 为 0 表示成功，其他值表示失败。 • Code: 10001, Msg: "invalid param"。 • Code: 10002, Msg: "invalid token"。 • Code: 10003, Msg: "invalid operate"。
Msg	String	-

实例属性

实例属性（InstanceProperties）。

参数名称	类型	描述
DeviceInfo	Object	{ Brand: string; Model: string }
GPSInfo	Object	{ Longitude: number; Latitude: number };
LanguageInfo	Object	{ Language: string; Country: string };
LocaleInfo	Object	{ Timezone: string };
ProxyInfo	Object	{ Enabled: boolean; Protocol: string; Host: string; Port: number; User: string; Password: string };
SIMInfo	Object	{ State: number; PhoneNumber: string; IMSI: string; ICCID: string };
ExtraProperties	Object	{ Key: string; Value: string }[];

设置 video 音量

设置 video 音量（setVideoVolume）。

输入参数

参数名称	必选	类型	描述
value	是	Number	音量 [0-1]。

示例

```
TCGSDK.setVideoVolume(0);
```

获取 video 音量

获取 video 音量（getVideoVolume）。

示例

```
const volume = TCGSDK.getVideoVolume();
```

播放视频

播放视频（playVideo）。

输入参数

参数名称	必选	类型	描述
status	是	String	'play' | 'pause'

示例

```
TCGSDK.playVideo('play');
```

开关麦克风

开关麦克风（switchMic）。

输入参数

参数名称	必选	类型	描述
status	是	String	'open' | 'close'

profile	否	Boolean | MicProfileConstraints	MicProfileConstraints {
			sampleRate?: number; // 默认值 44100
			echoCancellation?: ConstrainBoolean; // 回声消除 默认值 true
			noiseSuppression?: ConstrainBoolean; // 降噪 默认值 true
			autoGainControl?: ConstrainBoolean; // 增益 默认值 true
			deviceId?: string; // 设备id，可以通过 getDevices 接口获取，默认采用系统自选设备

			}

返回值

类型：Promise<{ code: 0 | 1; msg: string; userMedia: MediaStream }>;

参数名称	类型	描述
code	Number	code 为 0 表示成功，其他值表示失败。
msg	String	-
userMedia	MediaStream	获取的媒体信息。

示例

```
TCGSDK.switchMic({status: 'open'});
```

开关摄像头

开关摄像头（switchCamera）。

输入参数

参数名称	必选	类型	描述
status	是	String	'open' | 'close'
profile	否	Boolean | CameraProfileConstraints | CameraProfileType	CameraProfileConstraints { /** * 默认值 1280，传入 null 则采用系统默认选择值 */ width?: number | null; /** * 默认值 720，传入 null 则采用系统默认选择值 */ height?: number | null; frameRate?: number; /** * @deprecated */

			bitrate?: number; /** * input 的设备id，可以通过 getDevices 接口获取，默认采用系统自选设备。 * * 移动端可传入 'user' | 'environment'，来区分前置/后置摄像头 */ deviceId?: string | 'user' | 'environment'; } CameraProfileType = '120p' | '180p' | '240p' | '360p' | '480p' | '720p' | '1080p';

返回值

类型：Promise<{ code: 0 | 1; msg: string; userMedia: MediaStream }>;

参数名称	类型	描述
code	Number	code 为 0 表示成功，其他值表示失败。
msg	String	-
userMedia	MediaStream	获取的媒体信息。

示例

```
TCGSDK.switchCamera({status: 'open'});
TCGSDK.switchCamera({status: 'close'});
// 移动端打开后置摄像头
TCGSDK.switchCamera({ status: 'open', profile: { deviceId: 'environment'
} });
```

设置 video 的旋转角度

设置 video 的旋转角度（setVideoOrientation）。

输入参数

参数名称	必选	类型	描述

deg	是	Number	旋转角度当前只支持 0/90/180/270。
rotateContainer	否	Boolean	是否旋转整个 HTML 视图，默认值 true。
rotateMountPoint	否	Boolean	是否旋转 mountPoint 节点，默认值 false。

示例

```
// 旋转 HTML 文档
TCGSDK.setVideoOrientation({ deg: 90, rotateContainer: true });
// 旋转 MountPoint
TCGSDK.setVideoOrientation({ deg: 90, rotateMountPoint: true });
```

发送键盘事件

发送键盘事件（sendKeyboardEvent），**该方法调用通常是成对出现(像正常打字，通常是 down/up 组合)。**
键盘的键位码可通过 KeyCode 查询，对于云手游常用按键 KeyCode 是：

- KEY_BACK = 158
- KEY_MENU = 139
- KEY_HOME = 172
- KEYCODE_VOLUME_UP = 0x3a
- KEYCODE_VOLUME_Down = 0x3b

输入参数

参数名称	必选	类型	描述
key	是	Number	对应的 Code 码。
down	是	Boolean	是否是按下状态。
location	否	Number	1为左键，2为右键。

示例

```
// 按下时
TCGSDK.sendKeyboardEvent({key: 32, down: true});
```

```
// 抬起时
TCGSDK.sendKeyboardEvent({key: 32, down:  false});
```

发送鼠标事件

发送鼠标事件（sendMouseEvent）。

输入参数

参数名称	必选	类型	描述
type	是	String	鼠标事件类型。 • mouseleft • mouseright • mousemiddle • mouseforward • mousebackward • mousescroll • mousemove • mousedeltamove
down	否	Boolean	是否是按下状态(就像正常鼠标点击，通常是 down/up 组合)。
delta	否	Number	鼠标滚轮，通常传1或-1。

示例

```
// 鼠标左键按下
TCGSDK.sendMouseEvent({type: 'mouseleft', down: true});
// 鼠标左键抬起
TCGSDK.sendMouseEvent({type: 'mouseleft', down: false});
```

发送 touch 事件

发送 touch 事件（mobileTouchMove）。
根据 onVideoStreamConfigChange 回调，可拿到云端屏幕分辨率，x/y 坐标是相对于该分辨率的 width/height 的位置。

输入参数

参数名称	必选	类型	描述

finger_id	是	Number	touch finger id
event_type	是	Number	touch 事件类型。 • touchstart：0 • touchmove：1 • touchend：2 • touchcancel：2
x	是	Number	相对 x 轴坐标。
y	是	Number	相对 y 轴坐标。

示例

```
// start/end 成对发送
TCGSDK.mobileTouchMove({finger_id: 0, event_type: 0, x: 111, y: 1102 });
TCGSDK.mobileTouchMove({finger_id: 0, event_type: 2, x: 111, y: 1102 });
```

获取当前连接 requestId

获取当前连接 requestId（getRequestId），**在调用 TCGSDK.start() / TCGSDK.access() 后生效。**

示例

```
const requestId = TCGSDK.getRequestId();
```

推流视频截图

推流视频截图（screenShot）。

输入参数

参数名称	必选	类型	描述
name	否	String	截图名称，默认为 `tcgsdk-${+new Date()}`。
width	否	Number	截图宽度，默认取视频流宽度。
height	否	Number	截图高度，默认取视频流高度。

示例

```
TCGSDK.screenShot({name: 'abc123', width: 720, height: 1280});
```

快速发送内容

快速发送内容（sendText），会同时粘贴到剪贴板。

输入参数

参数名称	必选	类型	描述
content	是	String	需要发送的内容。

示例

```
TCGSDK.sendText('abc');
```

事件回调

最近更新时间：2025-09-03 14:10:22

Events 概览

方式一：通过 init 参数传入。

```
TCGSDK.init({
  ...,
  onInitSuccess: (res) => {
    console.log(res);
  }
})
```

方式二：事件监听。

```
TCGSDK.on('InitSuccess', (res) => {
  console.log(res);
})
```

方法	描述
InitSuccess	初始化成功回调，触发此回调之后才能调用后面的 API。
ConnectSuccess	连接成功回调。
ConnectFail	连接失败回调，连接断开，重连中会回调该接口。
WebrtcStatusChange	WebRTC 状态回调。
Disconnect	连接断开回调，断开/被踢触发此回调。
TouchEvent	触摸事件回调，移动端适用，移动端触摸事件回调。
VisibilityChange	页面显隐回调，页面显示/隐藏发生变化，status 包含 visible 以及 hidden。
VideoStreamConfigChange	推流分辨率变化回调。
Event	复合类事件回调。

AndroidInstanceEvent	云手机事件回调。
ImageEvent	截图回调，连接成功后会根据 image.interval 间隔时间，截取屏幕画面，并通过 onImageEvent 回调。

初始化成功回调

初始化成功回调（InitSuccess），触发此回调之后才能调用后面的 API。

返回值

参数名称	类型	描述
code	Number	• code = 0，成功。 • code = -1，localOffer 无法获取H264 编码 。 • code = -2，peerConnection 未断开，需要先调用 TCGSDK.destroy() 。
msg	String	-
description	String	相关描述。

示例

```
TCGSDK.on('InitSuccess', (res) => {
  console.log(res);
  // 启动
  TCGSDK.access({instanceId: 'cai-xxxx-xxxx'});
});
```

连接成功回调

连接成功回调（ConnectSuccess）。

示例

```
TCGSDK.on('ConnectSuccess', (res) => {
  console.log(res);
});
```

连接失败回调

连接失败回调（ConnectFail），连接断开，重连中会回调该接口。
通常重连时间超过两分钟（例如连接断开/移动端切后台，两分钟后触发重连），系统会自动回收实例，表现为返回 code > 0，建议该情况下重新 init + createSession。

返回值

参数名称	类型	描述
code	Number	• code = -3，超出重连次数。 • code = -2，自动重连中 。 • code = -1，连接失败，触发了限频操作 5s，可稍后再连接。 • code > 0，Proxy 返回的重连错误，通常连不上，需重新 init + createSession。
msg	String	-

示例

```
TCGSDK.on('ConnectFail', (res) => {
  console.log(res);
});
```

WebRTC 状态回调

WebRTC 状态回调（WebrtcStatusChange）。

返回值

参数名称	类型	描述
code	Number	• code = -2，获取 H264 编码失败。 • code = -1，setRemoteDescription 失败。 • code = 0，成功。 • code = 1，系统繁忙。 • code = 2，票据不合法。 • code = 3，用户带宽不足。 • code = 4，资源不足，没有可用机器。 • code = 5，session 失效，需要重新登录。 • code = 6，媒体描述信息错误。 • code = 7，游戏拉起失败。 • code = 100，Proxy 错误 。

		• code = 255，设备不支持 WebRTC。
msg	String	-

示例

```
TCGSDK.on('WebrtcStatusChange', (res) => {
  console.log(res);
});
```

连接断开回调

连接断开回调（Disconnect），断开/被踢触发此回调，如果持续连接不上，可以看回调 onConnectFail 相关错误码，在里面完善相关逻辑。

返回值

参数名称	类型	描述
code	Number	• code = -2，创建 local offer 失败，需要重新 init + createSession。 • code = -1，需要重连，通常出现在码率掉0，收不到推流，连接超时，ICE 断开，SDK 会自动重连。 • code = 0，主动关闭。 • code = 1，用户重复连接(该消息不可靠)。 • code = 2，用户心跳超时，WebRTC 服务端主动断开，这个消息有可能丢失 init + createSession。
msg	String	-

示例

```
TCGSDK.on('Disconnect', (res) => {
  console.log(res);
});
```

触摸事件回调

移动端适用，移动端触摸事件回调（TouchEvent），返回 OnTouchEventResponse[]，对应当前屏幕触控点数，可根据 array 长度判断多指操作。

返回值

OnTouchEventResponse

参数名称	类型	描述
id	Number	触控事件的 ID。
type	TouchType	事件类型，可选择 'touchstart'，'touchmove'，'touchend'，'touchcancel' 四种之一。
x	Number	触控点在视频区域内的 x 坐标。
y	Number	触控点在视频区域内的 y 坐标。
pageX	Number	触控点在当前网页内的 x 坐标。
pageY	Number	触控点在当前网页内的 y 坐标。
movementX	Number	触控点相对上次坐标的 x 偏移值。
movementY	Number	触控点相对上次坐标的 y 偏移值。

示例

```
TCGSDK.on('TouchEvent', (res) => {
  const [{ id, type, pageX, pageY }] = res;
  TCGSDK.mouseMove(id, type, pageX, pageY);
  if (type === 'touchstart') {
    TCGSDK.sendMouseEvent({ type: 'mouseleft', down: true });
  }
  if (type === 'touchend' || type === 'touchcancel') {
    TCGSDK.sendMouseEvent({ type: 'mouseleft', down: false });
  }
});
```

页面显隐回调

页面显示/隐藏发生变化（VisibilityChange），status 包含 visible 以及 hidden。

返回值

参数名称	类型	描述
status	String	'visible' | 'hidden'

示例

```
TCGSDK.on('VisibilityChange', ({status}) => {
  console.log(status);
});
```

推流分辨率变化回调

推流分辨率变化回调（VideoStreamConfigChange）。

返回值

参数名称	类型
width	Number
height	Number

示例

```
TCGSDK.on('Disconnect', ({width, height}) => {
  console.log(width, height);
});
```

复合类事件回调

复合类事件回调（Event）。

返回值

参数名称	类型	描述
type	String	事件名称 • autoplay：自动播放 • video_state：视频状态 • idle：idle 次数 • noflow • noflowcenter • openurl：云端回传的 URL • pointerlockerror：锁定鼠标失败 • readclipboarderror：获取剪切板失败 • webrtc_stats

		• latency
data	String	不同 type可能 data 不同，参考下表。

type 对应 data 如下表：

type	data
autoplay	Object<{code: number; message: string;}> code： • 0：success • -1：failed
video_state	Object<{code: number; message: string;}> code： • 0：playing • 1：pause • 2：ended
idle	Object<{times: number;}>
noflow	-
noflowcenter	-
openurl	Object<{value: string;}>
pointerlockerror	-
readclipboarderror	Object<{message?: string;}>

webrtc_stats	Object<{
	bit_rate: number; // 客户端接收的码率，单位：Mbps
	cpu: number; // 云端 CPU 占用率，单位：百分比
	gpu: string; // 云端 GPU 占用率，单位：百分比
	delay: number; // 客户端收到图像帧到解码显示的延时，单位：ms，iOS 可能收不到
	fps: number; // 客户端显示帧率
	load_cost_time: number; // 云端加载时长，单位：ms
	nack: number; // 客户端重传次数
	packet_lost: number; // 客户端丢包次数
	packet_received: number; // 客户端收到的包总数
	rtt: number; // 客户端到云端，网络端数据包往返耗时
	timestamp: number; // 此数据回调的时间戳，单位：ms
	}>

latency	Object<{value: number; message: string;}> value： • 0：NETWORK_NORMAL • 1：NETWORK_CONGESTION • 2：NACK_RISING • 3：HIGH_DELAY • 4：NETWORK_JITTER
ice_state	Object<{value: string;}> value： • connected：已连接 • disconnected：连接断开

示例

```
TCGSDK.on('Event', ({type, data}) => {
  // 云端打开浏览器
  if (type === 'openurl') {
    const { value } = data;
    window.open(value);
    console.log('openurl', value);
  }
  if (type === 'autoplay') {
    console.log('autoplay', data);
  }
});
```

云手机事件回调

云手机事件回调（AndroidInstanceEvent）。

返回值

参数名称	类型	描述
type	String	事件名称 • trans_message：云端回传消息 • system_usage：系统用量 • clipboard_event：剪切板事件 • notification_event
data	String	不同 type可能 data 不同，参考下表。

type 对应 data 如下表：

type	data
trans_message	Object<{msg: string; package_name: string;}>
system_usage	Object<{cpu_usage: number; mem_usage: number; gpu_usage: number;}>
clipboard_event	Object<{text: string; writeText?: boolean}> writeText boolean 是否已写入剪切板
notification_event	Object<{package_name: string; title: string; text: string;}>
system_status	Object<{ nav_visible: boolean; music_volume: number;}>

示例

```
TCGSDK.on('AndroidInstanceEvent', ({type, data}) => {
  console.log('AndroidInstanceEvent', type, data);
});
```

截图回调

截图回调（ImageEvent）连接成功后会根据 image.interval 间隔时间，截取屏幕画面，并通过 onImageEvent 回调。

返回值

参数名称	类型	描述
type	String	事件名称 'screenshot'。
data	String	不同 type可能 data 不同，参考下表。

type 对应 data 如下表：

type	data
screenshot	Object<{instanceId: string, url: string;}>[]

示例

```
TCGSDK.on('ImageEvent', (res) => {
```

```
    console.log(res);
});
```

Windows SDK
SDK 集成指引

最近更新时间：2025-09-03 14:10:22

Windows SDK 为开发者提供了完整的云手机集成能力，支持将远程云手机的操作界面与控制功能集成至自研应用。

运行环境说明

本 TcrSdk Windows C++ SDK 适用于 Windows 平台。请确保您的开发和运行环境满足以下要求：

- **操作系统**：Windows 10 64位或更高版本（建议使用 Windows 10/11 64位）。
- **集成开发环境**：Microsoft Visual Studio 2022/2019
- 若您使用 Qt 开发，请确保 Qt 工程配置为 MSVC 2022 64-bit 工具链。
- **运行库**：需确保目标机器已安装 `Visual C++ Redistributable for Visual Studio 2022 (x64)` 。

获取 SDK

SDK	下载地址	SDK 说明文档	版本发布日志
Windows SDK	GitHub	接口说明	Release notes

工程环境准备

SDK 为纯 C 接口，无 Qt 依赖，可在任意 C++ 工程中集成。下文以 Qt Quick 工程为例，介绍典型的集成流程和注意事项。
您也可以下载示例 Qt_Quick_Demo 进行参考。

依赖环境

- **TcrSdk**：包含头文件、lib、dll 文件。
- **下载地址**：TcrSdk Windows C++ SDK 下载页面 。
- **编译器要求**
  - 推荐集成环境使用 Microsoft Visual Studio 2022。
  - 若使用 Qt 或其他项目，请确保编译器为 MSVC 2022 64-bit。
- **运行库**：需确保目标机器已安装 `Visual C++ Redistributable for Visual Studio 2022 (x64)` 。

目录结构建议

```
CloudPhone_QtQuick/
├── src/                    # 业务代码
```

```
├── qml/                    # QML 资源
├── shaders/                # 着色器
├── third_party/
│   └── TcrSdk/             # 拷贝下载的TcrSdk到此处
│       ├── Release/
│       └── Debug/
└── CMakeLists.txt
```

CMake 工程配置

参考如下 CMake 配置片段，确保 SDK 头文件、lib、dll 正确集成：

```
# 源码与头文件
target_include_directories(CloudPhone_QtQuick PRIVATE
    ${CMAKE_CURRENT_SOURCE_DIR}/src
    ${CMAKE_CURRENT_SOURCE_DIR}/third_party/TcrSdk/Release/include
)

# TcrSdk 库路径（以Release为例）
set(TCRSDK_BASE_DIR ${CMAKE_CURRENT_SOURCE_DIR}/third_party/TcrSdk)
if(CMAKE_SIZEOF_VOID_P EQUAL 8)
    set(TCRSDK_PLATFORM x64)
else()
    set(TCRSDK_PLATFORM Win32)
endif()
set(TCRSDK_DLL
"${TCRSDK_BASE_DIR}/Release/${TCRSDK_PLATFORM}/TcrSdk.dll")
set(TCRSDK_LIB
"${TCRSDK_BASE_DIR}/Release/${TCRSDK_PLATFORM}/TcrSdk.lib")

target_link_libraries(CloudPhone_QtQuick PRIVATE ${TCRSDK_LIB})

# 拷贝 DLL 到输出目录
add_custom_command(TARGET CloudPhone_QtQuick POST_BUILD
    COMMAND ${CMAKE_COMMAND} -E copy_if_different
        ${TCRSDK_DLL}
        $<TARGET_FILE_DIR:CloudPhone_QtQuick>
)
```

SDK 典型使用流程

1. 初始化 Client

```
#include "tcr_c_api.h"

TcrConfig config;
config.Token = tokenResult.token.c_str();            // 业务后台获取的 Token
config.AccessInfo = tokenResult.accessInfo.c_str(); // 业务后台获取的
AccessInfo

TcrClientHandle tcrClient = tcr_client_get_instance();
TcrErrorCode err = tcr_client_init(tcrClient, &config);

if (err != TCR_SUCCESS) {
    // 处理初始化失败
}
```

2. 获取实例截图（按需可选）

```
char buffer[2048];
TcrAndroidInstance instance =
tcr_client_get_android_instance(tcrClient);
bool ok = tcr_instance_get_image(instance, buffer, 2048,
instanceId.c_str(), 0, 0, 0);
if (ok) {
    // buffer 返回图片 URL，可用于加载图片
}
```

3. 创建串流会话并连接实例

单实例连接

```
TcrSessionHandle tcrSession = tcr_client_create_session(tcrClient);
const char* instanceIds[] = { instanceId.c_str() };
tcr_session_access(tcrSession, instanceIds, 1, false); // false 表示单实例
```

多实例连接

```
TcrSessionHandle tcrSession = tcr_client_create_session(tcrClient);
std::vector<const char*> idPtrs;
for (const auto& id : instanceIds) idPtrs.push_back(id.c_str());
tcr_session_access(tcrSession, idPtrs.data(), idPtrs.size(), true); //
true 表示多实例
```

> **说明：**
> 如需动态加入实例组，可通过 `tcr_instance_join_group` 。

4. 设置回调

视频帧回调

```
static void VideoFrameCallback(void* user_data, const
TcrVideoFrameBuffer* frame_buffer, int64_t timestamp_us,
TcrVideoRotation rotation) {
    // 处理视频帧
}
static TcrVideoFrameObserver video_observer;
video_observer.user_data = nullptr; // 可传 this 指针
video_observer.on_frame_buffer = VideoFrameCallback;
tcr_session_set_video_frame_observer(tcrSession, &video_observer);
```

事件回调

```
static void SessionEventCallback(void* user_data, TcrSessionEvent event,
const char* eventData) {
    // 处理事件
}
static TcrSessionObserver session_observer;
session_observer.user_data = nullptr; // 可传 this 指针
session_observer.on_event = SessionEventCallback;
tcr_session_set_observer(tcrSession, &session_observer);
```

5. 业务操作

实时控制

```
// 模拟返回键
tcr_session_send_keyboard_event(tcrSession, 158, true);  // 按下
tcr_session_send_keyboard_event(tcrSession, 158, false); // 抬起

// 触摸事件（按下）
tcr_session_touchscreen_touch(tcrSession, 100, 200, 0, 720, 1280, 0); //
eventType: 0=DOWN, 1=MOVE, 2=UP

// 透传消息
```

```
tcr_session_send_trans_message(tcrSession, "com.example.app",
"your_message");

// 粘贴文本
tcr_session_paste_text(tcrSession, "hello world");

// 切换输入法
tcr_session_switch_ime(tcrSession, "cloud"); // "cloud" 或 "local"
```

批量任务

对云手机进行批量任务操作（如修改分辨率、应用管理等）：

```
const char* input_json = R"({
    "TaskType": "ModifyResolution",
    "Params": {
        "cai-xxxx-xxx": { "Width": 1080, "Height": 1920 }
    }
})";
char* output = nullptr;
size_t output_size = 0;
TcrAndroidInstance instance =
tcr_client_get_android_instance(tcrClient);
TcrErrorCode code = tcr_instance_request(instance, input_json, &output,
&output_size);

if (code == TCR_SUCCESS && output) {
    // output 为 JSON 字符串，具体内容需解析
    // ... 业务处理 ...
    tcr_instance_free_result(instance, output);
}
```

6. 资源释放

每一个创建的会话实例在使用完成后必须释放：

```
tcr_client_destroy_session(tcrClient, tcrSession);
```

批量任务接口说明

批量任务调用

```
const char* input_json = R"({
    "TaskType": "ModifyResolution",
    "Params": {
        "cai-xxxx-xxx": { "Width": 1080, "Height": 1920 }
    }
})";
char* output = nullptr;
size_t output_size = 0;
TcrErrorCode code = tcr_instance_request(instance, input_json, &output,
&output_size);

if (code == TCR_SUCCESS && output) {
    // output 为 JSON 字符串，具体内容需要根据不同任务进行解析（不关注结果则可忽略）
    // ... 业务处理 ...
    tcr_instance_free_result(tcrOperator, output);
}
```

支持的任务类型

操作类型	任务
修改分辨率	ModifyResolution
修改 GPS	ModifyGPS
粘贴文本	Paste
发送剪贴板	SendClipboard
修改传感器	ModifySensor
摇动/吹气	• Shake • Blow
透传消息	SendTransMessage
修改实例属性	ModifyInstanceProperties

应用管理	• StartApp • StopApp • UnInstallByPackageName • ClearAppData • EnableApp • DisableApp
保活管理	• AddKeepAliveList • RemoveKeepAliveList • SetKeepAliveList • ClearKeepAliveList • DescribeKeepAliveList • ModifyKeepFrontAppStatus
摄像头	• StartCameraMediaPlay • StopCameraMediaPlay • DescribeCameraMediaPlayStatus • DisplayCameraImage
应用黑名单	• AddAppInstallBlackList • SetAppInstallBlackList • DescribeAppInstallBlackList • ClearAppInstallBlackList
查询应用列表	• ListUserApps • ListAllApps
静音	Mute
媒体库搜索	MediaSearch
重启实例	Reboot
应用后台	MoveAppBackground

详情参见头文件，每个任务类型的参数和返回格式均有详细说明。

常见问题与建议

- 内存释放：批量任务返回的 `output` 需调用 `tcr_operator_free_result` 释放。
- 会话管理：每一个 `TcrClientHandle` 只能调用一次 `tcr_session_access` ，如需多个会话请创建多个实例并释放使用过的实例。

- **日志调试：您需要通过 `tcr_set_log_callback` 、 `tcr_set_log_level` 配置日志，以便在出现问题之后将日志反馈给开发团队定位问题。**

接口说明

最近更新时间：2025-09-04 15:11:42

接口概览

生命周期接口

函数名	说明
tcr_client_get_instance	获取 TcrClientHandle 实例。
tcr_client_init	初始化 TcrClientHandle。
tcr_client_create_session	创建会话对象。
tcr_session_init	初始化会话。
tcr_session_access	启动会话。
tcr_session_start	开始会话。
tcr_client_destroy_session	销毁会话对象。
tcr_session_set_observer	设置会话事件观察者。
tcr_set_log_callback	设置日志回调函数。
tcr_set_log_level	设置日志输出级别。
tcr_get_log_level	获取当前日志输出级别。
tcr_session_create_data_channel	创建自定义数据通道。
tcr_data_channel_send	通过自定义数据通道发送数据。
tcr_data_channel_close	关闭自定义数据通道。

安卓实例操作接口

函数名	说明

tcr_client_get_android_instance	获取 TcrAndroidInstance 操作对象。
tcr_instance_join_group	加入分组
tcr_instance_request_stream	控制实例视频流开关及质量。
tcr_instance_set_master	设置/取消主控实例。
tcr_instance_set_sync_list	设置同步列表。
tcr_instance_get_image	获取实例截图图片 URL。
tcr_instance_get_download_address	获取文件下载 URL。
tcr_instance_get_download_logcat_address	获取 logcat 日志下载地址。
tcr_instance_upload_media	上传文件到云实例相册。
tcr_instance_upload_files	上传本地文件到云手机。
tcr_instance_request	批量请求云端任务。
tcr_instance_free_result	释放结果内存。

音视频控制接口

函数名	说明
tcr_session_set_video_frame_observer	设置视频帧回调。
tcr_video_frame_buffer_to_i420	转换视频帧到 I420 格式。
tcr_video_frame_buffer_is_d3d11	检查是否为 D3D11 格式。
tcr_video_frame_buffer_get_d3d11_texture	获取 D3D11 纹理指针。
tcr_session_pause_strea	暂停媒体流。

ming	
tcr_session_resume_streaming	恢复媒体流。
tcr_session_set_remote_video_profile	设置远端视频流参数。
tcr_session_send_trans_message	发送透传消息。
tcr_session_paste_text	粘贴文本到输入框。
tcr_session_input_text	直接输入文本。
tcr_session_switch_ime	切换输入法。
tcr_session_enable_local_camera	启用本地摄像头。
tcr_session_disable_local_camera	禁用本地摄像头。
tcr_session_enable_local_camera_with_config	启用本地摄像头（带视频配置参数）。
tcr_session_is_local_camera_enabled	检查本地摄像头是否已启用。
tcr_session_local_camera_config	配置本地摄像头参数。
tcr_session_get_camera_device_count	获取可用摄像头设备数量。
tcr_session_get_camera_device	获取摄像头设备信息。
tcr_session_enable_local_microphone	启用本地麦克风。
tcr_session_disable_local_microphone	禁用本地麦克风。
tcr_session_is_local_microphone_enabled	检查本地麦克风是否已启用。

外设控制接口

函数名	说明
tcr_session_switch_camera	开关摄像头。
tcr_session_switch_mic	开关麦克风。
tcr_session_send_keyboard_event	发送云端键盘事件。
tcr_session_touchscreen_touch	发送触摸事件。

详细说明

获取 TcrClientHandle 实例

获取 TcrClientHandle 实例，用于后续所有客户端相关操作。

```
TcrClientHandle tcr_client_get_instance();
```

初始化 TcrClientHandle

初始化 TcrClientHandle。

```
TcrErrorCode tcr_client_init(TcrClientHandle client, const TcrConfig*
config);
```

参数	类型	说明
client	TcrClientHandle	客户端句柄。
config	const TcrConfig*	配置信息指针。

创建会话对象

创建会话对象，用于串流和控制云手机实例。

```
TcrSessionHandle tcr_client_create_session(TcrClientHandle client);
```

参数	类型	说明
client	TcrClientHandle	客户端句柄。

初始化会话

初始化会话，初始化成功后会通过回调发送 TCR_SESSION_EVENT_STATE_INITED 事件通知。

```
TCRSDK_API void tcr_session_init(TcrSessionHandle session);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄。

启动会话并连接云手机

启动会话并连接云手机。

```
void tcr_session_access(TcrSessionHandle session, const char**
instanceIds, int32_t length);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄。
instanceIds	const char**	云端实例 ID 数组。
length	int32_t	实例 ID 数组长度。

开始会话

启动会话。注意，每个会话只能调用一次此函数。

```
TCRSDK_API void tcr_session_start(TcrSessionHandle session, const char*
serverSession);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄。
serverSession	const char*	服务端会话标识。

销毁会话对象

销毁会话对象，释放相关资源。

```
void tcr_client_destroy_session(TcrClientHandle client, TcrSessionHandle
session);
```

参数	类型	说明
client	TcrClientHandle	客户端句柄。
session	TcrSessionHandle	会话句柄。

设置会话事件观察者

设置会话事件观察者，用于接收会话生命周期及业务事件。

```
void tcr_session_set_observer(TcrSessionHandle session, const
TcrSessionObserver* observer);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄。
observer	const TcrSessionObserver*	观察者结构体指针。

设置日志回调函数

设置日志回调函数，用于接收 SDK 日志。

```
void tcr_set_log_callback(const TcrLogCallback* callback);
```

参数	类型	说明
callback	const TcrLogCallback*	日志回调结构体指针。

设置日志输出级别

```
void tcr_set_log_level(TcrLogLevel level);
```

参数	类型	说明
level	TcrLogLevel	日志级别。

获取当前日志输出级别

```
TcrLogLevel tcr_get_log_level();
```

创建自定义数据通道

创建并返回一个自定义数据通道句柄，用于数据传输。

```
TCRSDK_API TcrDataChannelHandle tcr_session_create_data_channel(
    TcrSessionHandle session,
    int32_t port,
    const TcrDataChannelObserver* observer
);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄。
port	int32_t	云端唯一标识数据通道的端口号。
observer	const TcrDataChannelObserver*	观察者结构体指针，回调时会传入 user_data。生命周期需调用方保证有效。
返回值	TcrDataChannelHandle	数据通道句柄，失败返回 NULL。

发送数据

通过已创建的数据通道句柄向云端发送自定义数据。

```
TCRSDK_API TcrErrorCode tcr_data_channel_send(
    TcrDataChannelHandle channel,
    const uint8_t* data,
    size_t size
);
```

参数	类型	说明
channel	TcrDataChannelHandle	数据通道句柄。
data	const uint8_t*	数据指针。

size	size_t	数据长度。
返回值	TcrErrorCode	错误码，TCR_SUCCESS 表示成功。

关闭数据通道

关闭指定的数据通道。

```
TCRSDK_API void tcr_data_channel_close(TcrDataChannelHandle channel);
```

参数	类型	说明
channel	TcrDataChannelHandle	数据通道句柄。

获取 TcrAndroidInstance 操作对象

```
TcrAndroidInstance tcr_client_get_android_instance(TcrClientHandle
client);
```

参数	类型	说明
client	TcrClientHandle	客户端句柄。

将多个实例加入当前群组会话

将多个实例加入当前群组会话，实现多实例同步控制。

```
void tcr_instance_join_group(TcrAndroidInstance op, const char**
instanceIds, int32_t length);
```

参数	类型	说明
op	TcrAndroidInstance	实例操作句柄。
instanceIds	const char**	实例 ID 字符串数组。
length	int32_t	实例 ID 数组长度。

请求打开或关闭指定实例的视频流

请求打开或关闭指定实例的视频流，并设置流质量。

```
void tcr_instance_request_stream(TcrAndroidInstance op, const char*
instanceID, bool status, const char* level);
```

参数	类型	说明
op	TcrAndroidInstance	实例操作句柄。
instanceID	const char*	实例 ID。
status	bool	• true：打开流 • false：关闭流。
level	const char*	流质量级别（"low"、"normal"、"high"）。

设置或取消主控实例

设置或取消主控实例，实现主从同步。

```
void tcr_instance_set_master(TcrAndroidInstance op, const char*
instanceId, bool status);
```

参数	类型	说明
op	TcrAndroidInstance	实例操作句柄。
instanceId	const char*	主控实例 ID。
status	bool	true 设置为主控，false 取消主控。

设置群组会话同步实例列表

设置群组会话中需要被操作的实例列表。

```
void tcr_instance_set_sync_list(TcrAndroidInstance op, const char**
instanceIds, int32_t length);
```

参数	类型	说明
op	TcrAndroidInstance	实例操作句柄。
instanceIds	const char**	需要同步控制的实例 ID 数组。

length	int32_t	实例 ID 数组长度。

获取指定实例的截图图片 URL

```
bool tcr_instance_get_image(TcrAndroidInstance op, char* buffer, int
bufferLen, const char* instanceId, int width, int height, int quality);
```

参数	类型	说明
op	TcrAndroidInstance	实例操作句柄。
buffer	char*	输出缓冲区。
bufferLen	int	输出缓冲区长度。
instanceId	const char*	实例 ID。
width	int	图片宽度（≤0 为默认）。
height	int	图片高度（≤0 为默认）。
quality	int	图片质量（1–100）。

获取文件下载 URL

```
bool tcr_instance_get_download_address(TcrAndroidInstance op, char*
buffer, int bufferLen, const char* instanceId, const char* path);
```

参数	类型	说明
op	TcrAndroidInstance	实例操作句柄。
buffer	char*	输出缓冲区。
bufferLen	int	缓冲区长度。
instanceId	const char*	实例 ID。
path	const char*	文件路径。

获取 logcat 日志下载地址

获取云手机实例 logcat 日志压缩包的下载地址。

```
bool tcr_instance_get_download_logcat_address(TcrAndroidInstance op,
char* buffer, int bufferLen, const char* instanceId, int recentDays);
```

参数	类型	说明
op	TcrAndroidInstance	实例操作句柄。
buffer	char*	输出缓冲区。
bufferLen	int	缓冲区长度。
instanceId	const char*	实例 ID。
recentDays	int	下载几天内的日志。

上传文件到云实例相册

上传文件到云实例，固定上传至 /data/media/0/DCIM，并自动添加到相册。

```
bool tcr_instance_upload_media(
    TcrAndroidInstance op,
    const char* instanceId,
    const char** local_paths,
    int file_count,
    char** output,
    size_t* output_size,
    const TcrUploadCallback* callback
);
```

参数	类型	说明
op	TcrAndroidInstance	实例操作句柄。
instanceId	const char*	实例 ID。
local_paths	const char**	本地文件路径数组。
file_count	int	文件数量。
output	char**	输出 JSON 字符串（需释放）。
output_size	size_t*	输出字符串长度。
callback	const TcrUploadCallback*	上传进度回调（可为 NULL）。

上传本地文件到云手机实例

上传本地文件到云手机实例指定目录。

```
bool tcr_instance_upload_files(
    TcrAndroidInstance op,
    const char* instanceId,
    const char** local_paths,
    const char** cloud_paths,
    int file_count,
    char** output,
    size_t* output_size,
    const TcrUploadCallback* callback
);
```

参数	类型	说明
op	TcrAndroidInstance	实例操作句柄。
instanceId	const char*	实例 ID。
local_paths	const char**	本地文件路径数组。
cloud_paths	const char**	云端目标路径数组（可为 NULL）。
file_count	int	文件数量。
output	char**	输出 JSON 字符串（需释放）。
output_size	size_t*	输出字符串长度。
callback	const TcrUploadCallback*	上传进度回调（可为 NULL）。

批量请求云端任务

`tcr_instance_request` 用于批量请求云端安卓实例的各类任务操作，包括但不限于分辨率修改、文本粘贴、GPS 修改、应用管理、设备属性变更、保活策略、媒体操作等。该接口支持一次性对多个实例下发同类型任务。

```
TcrErrorCode tcr_instance_request(TcrAndroidInstance op, const char*
input_json, char** output, size_t* output_size);
```

参数	类型	说明

op	TcrAndroidInstance	实例操作句柄。
input_json	const char*	输入 JSON 字符串，指定任务类型、目标实例及参数，详见下方 输入 JSON 格式说明 。
output	char**	输出 JSON 字符串（需调用 tcr_instance_free_result 释放）。
output_size	size_t*	输出字符串长度。

输入 JSON 格式

```
{
  "TaskType": "任务类型字符串",
  "Params": {
    "cai-xxxx-xxx": { ... }, // 实例ID: 任务参数
    "cai-yyyy-yyy": { ... }
  }
}
```

- **TaskType**：必填，字符串，指定要执行的任务类型。支持的任务类型详见 下表 。
- **Params**：必填，字典。key 为实例 ID，value 为该实例的任务参数对象。参数结构依赖于 TaskType。

支持的任务类型及参数说明

任务类型	说明	参数结构示例
ModifyResolution	修改分辨率。	`{ "Width": 1080, "Height": 1920, "DPI": 320 }`
ModifyGPS	修改 GPS 位置。	`{ "Longitude": 121.47, "Latitude": 31.23 }`
Paste	粘贴文本。	`{ "Text": "hello" }`
SendClipboard	发送剪贴板内容。	`{ "Text": "hello" }`
ModifySensor	修改传感器数据。	`{ "Type": "accelerometer", "Values": [0.1, 0.2, 0.3], "Accuracy": 3 }`
Shake	摇动设备。	`{}`
Blow	吹气。	`{}`

SendTransMessage	发送透传消息。	`{ "PackageName": "com.example", "Msg": "hello" }`
ModifyInstanceProperties	修改实例属性。	详见下方说明
ModifyKeepFrontAppStatus	修改保活应用状态。	`{ "PackageName": "com.example", "Enable": true, "RestartInterValSeconds": 3 }`
UnInstallByPackageName	卸载应用。	`{ "PackageName": "com.example" }`
StartApp	启动应用。	`{ "PackageName": "com.example", "ActivityName": "xxx" }`
StopApp	停止应用。	`{ "PackageName": "com.example" }`
ClearAppData	清除应用数据。	`{ "PackageName": "com.example" }`
EnableApp	启用应用。	`{ "PackageName": "com.example" }`
DisableApp	禁用应用。	`{ "PackageName": "com.example" }`
StartCameraMediaPlay	开始摄像头媒体播放。	`{ "FilePath": "/sdcard/video.mp4", "Loops": 3 }`
DisplayCameraImage	显示摄像头图像。	`{ "FilePath": "/sdcard/image.jpg" }`
AddKeepAliveList	添加保活列表。	`{ "AppList": ["com.wechat", "com.alipay"] }`
RemoveKeepAliveList	移除保活列表。	`{ "AppList": ["com.wechat"] }`
SetKeepAliveList	设置保活列表。	`{ "AppList": ["com.wechat", "com.alipay"] }`
DescribeKeepFrontAppStatus	获取保活应用状态。	`{}`
StopCameraMediaPlay	停止摄像头播放。	`{}`

DescribeCameraMediaPlayStatus	获取摄像头播放状态。	{}
DescribeKeepAliveList	获取保活列表。	{}
ClearKeepAliveList	清除保活列表。	{}
ListUserApps	列出用户应用。	{}
Mute	静音开关。	{ "Mute": true }
MediaSearch	媒体库文件搜索。	{ "Keyword": "abc" }
Reboot	重启实例。	{}
ListAllApps	查询所有应用列表。	{}
MoveAppBackground	关闭应用至后台。	{}
AddAppInstallBlackList	新增应用安装黑名单。	{ "AppList": ["com.xxx"] }
RemoveAppInstallBlackList	移除应用安装黑名单。	{ "AppList": ["com.xxx"] }
SetAppInstallBlackList	覆盖应用安装黑名单。	{ "AppList": ["com.xxx"] }
DescribeAppInstallBlackList	查询应用安装黑名单。	{}
ClearAppInstallBlackList	清空应用安装黑名单。	{}

说明：
详细参数结构和返回示例可参考头文件中的说明。

典型输入示例

```
{
  "TaskType": "ModifyResolution",
  "Params": {
```

```
    "cai-xxxx-xxx": { "Width": 1080, "Height": 1920 },
    "cai-yyyy-yyy": { "Width": 720, "Height": 1280 }
  }
}
```

输出结果格式

返回值为一个 JSON 对象，key 为实例 ID，value 为该实例的任务执行结果对象。例如：

```
{
  "cai-xxxx-xxx": { "Code": 0 },
  "cai-yyyy-yyy": { "Code": 10100, "Msg": "operation denied" },
  "cai-zzzz-zzz": { "Code": -101, "Msg": "HTTP request failed" }
}
```

- `Code` 为0表示成功，非0表示失败， `Msg` 为失败原因。
- 某些查询类任务（如 DescribeInstanceProperties、ListUserApps 等）返回结构会包含详细数据，详见各任务类型在头文件中的说明。

其他说明

- `output` 由 SDK 内部分配内存，使用完毕后需调用 `tcr_instance_free_result` 释放。
- 支持批量对多个实例下发同类型任务，便于大规模自动化管理。
- 返回的 JSON 字符串长度通过 `output_size` 返回。
- 任务类型及参数如有疑问，请参考接口注释或联系技术支持。

返回值

- `TCR_SUCCESS` ：任务请求成功。
- `TCR_ERR_INVALID_PARAMS` ：输入参数无效。

释放由 tcr_instance_request 分配的结果内存

```
void tcr_instance_free_result(TcrAndroidInstance op, char* output);
```

参数	类型	说明
op	TcrAndroidInstance	实例操作句柄。
output	char*	需要释放的内存指针。

设置远端视频帧观察者

设置远端视频帧观察者，用于接收解码后的视频帧数据。

```
void tcr_session_set_video_frame_observer(TcrSessionHandle session,
const TcrVideoFrameObserver* observer);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄。
observer	const TcrVideoFrameObserver*	视频帧观察者结构体指针。

视频帧缓冲区转 I420 格式

将输入的视频帧缓冲区转换为 I420 格式缓冲区。

```
TCRSDK_API bool tcr_video_frame_buffer_to_i420(const
TcrVideoFrameBuffer* buffer, TcrI420Buffer* output_i420);
```

参数	类型	说明
buffer	const TcrVideoFrameBuffer*	输入的视频帧缓冲区。
output_i420	TcrI420Buffer*	输出的 I420 缓冲区指针。
返回值	bool	转换是否成功。

检查视频帧格式

检查视频帧缓冲区是否为D3D11格式。

```
TCRSDK_API bool tcr_video_frame_buffer_is_d3d11(const
TcrVideoFrameBuffer* buffer);
```

参数	类型	说明
buffer	const TcrVideoFrameBuffer*	视频帧缓冲区。
返回值	bool	是否为D3D11格式。

获取 D3D11 纹理指针

获取视频帧缓冲区中的 D3D11 纹理指针（需保证缓冲区为 D3D11 格式）。

版权所有：腾讯云计算（北京）有限责任公司

```
TCRSDK_API void* tcr_video_frame_buffer_get_d3d11_texture(const
TcrVideoFrameBuffer* buffer);
```

参数	类型	说明
buffer	const TcrVideoFrameBuffer*	视频帧缓冲区（必须为 D3D11 格式）。
返回值	void*	ID3D11Texture2D 指针，失败时返回 NULL。

暂停媒体流

暂停媒体流（如视频流），通常用于临时挂起。

```
void tcr_session_pause_streaming(TcrSessionHandle session);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄。

恢复媒体流

恢复媒体流（如视频流），与暂停配合使用。

```
void tcr_session_resume_streaming(TcrSessionHandle session);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄。

设置远端视频流参数

设置远端视频流参数（帧率、码率等）。

```
void tcr_session_set_remote_video_profile(TcrSessionHandle session,
int32_t fps, int32_t minBitrate, int32_t maxBitrate);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄。

fps	int32_t	视频帧率。
minBitrate	int32_t	最小码率（kbps）。
maxBitrate	int32_t	最大码率（kbps）。

向云端应用发送透传消息

```
void tcr_session_send_trans_message(TcrSessionHandle session, const char* package_name, const char* msg);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄。
package_name	const char*	应用包名。
msg	const char*	消息内容。

将文本写入剪贴板并粘贴到输入框

```
void tcr_session_paste_text(TcrSessionHandle session, const char* msg);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄。
msg	const char*	文本内容。

将文本直接输入到输入框

该方法直接输入文本，不使用剪贴板。

> **注意：**
> 调用此方法前，客户端必须已经收到 ime_type 设置为 "local" 的 IME_STATUS_CHANGE 事件。

```
void tcr_session_input_text(TcrSessionHandle session, const char* content, const char* mode);
```

参数	类型	说明

session	TcrSessionHandle	会话句柄。
content	const char*	输入文本内容。
mode	const char*	输入模式："append" 或 "override"。

切换输入法

```
void tcr_session_switch_ime(TcrSessionHandle session, const char* ime);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄。
ime	const char*	输入法类型："cloud" 或 "local"。

启用本地摄像头

启用或禁用本地摄像头。

```
bool tcr_session_enable_local_camera(TcrSessionHandle session, bool
enable);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄
enable	bool	true 启用，false 禁用摄像头

返回值：操作是否成功，true 表示成功。

禁用本地摄像头

禁用本地摄像头。

```
void tcr_session_disable_local_camera(TcrSessionHandle session);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄。

启用本地摄像头（带视频配置参数）

启用本地摄像头，支持指定视频配置参数。内部会自动查找并选择最匹配的摄像头。
建议客户端可通过 `tcr_session_get_camera_device_count` 和 `tcr_session_get_camera_device` 获取设备支持的摄像头能力进行参数选择。

```
bool tcr_session_enable_local_camera_with_config(TcrSessionHandle
session, const TcrVideoConfig* config);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄。
config	const TcrVideoConfig*	视频配置参数。

返回值：是否成功启用，true 表示成功。

检查本地摄像头是否已启用

判断当前会话的本地摄像头是否已启用。

```
bool tcr_session_is_local_camera_enabled(TcrSessionHandle session);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄。

返回值：true 已启用，false 未启用。

配置本地摄像头参数

设置摄像头的最大码率和最大帧率（需要先启用摄像头，否则无效）。

```
bool tcr_session_local_camera_config(TcrSessionHandle session, int
max_bitrate_bps, int max_fps);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄。
max_bitrate_bps	int	最大码率（单位：bps）。
max_fps	int	最大帧率（单位：fps）。

返回值：配置结果，true 表示成功。

获取可用摄像头设备数量

用于查询设备支持的摄像头数量，负值表示出错。

```
int tcr_session_get_camera_device_count(TcrSessionHandle session);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄。

返回值：可用摄像头设备数量（≥0），负值表示出错。

获取摄像头设备

获取指定摄像头设备的详细信息。

```
TCRSDK_API bool tcr_session_get_camera_device(TcrSessionHandle session,
int device_index, TcrCameraDeviceInfo* info);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄。
device_index	int	设备索引，范围：(0, 设备数量-1)。
info	TcrCameraDeviceInfo	输出参数，指向 TcrCameraDeviceInfo 结构体，由调用方分配。

返回值：成功返回 true，失败返回 false。

启用本地麦克风

启用本地麦克风。

```
TCRSDK_API bool tcr_session_enable_local_microphone(TcrSessionHandle
session);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄。

返回值：是否成功启用，true 表示成功。

禁用本地麦克风

禁用本地麦克风。

```
TCRSDK_API void tcr_session_disable_local_microphone(TcrSessionHandle
session);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄。

检查本地麦克风

检查本地麦克风是否已启用。

```
TCRSDK_API bool tcr_session_is_local_microphone_enabled(TcrSessionHandle
session);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄。

返回值：是否已启用，true 表示已启用。

开关云端摄像头

```
void tcr_session_switch_camera(TcrSessionHandle session, const char*
status);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄。
status	const char*	状态："open" 或 "close"。

开关云端麦克风

```
void tcr_session_switch_mic(TcrSessionHandle session, const char*
status);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄。
status	const char*	状态："open" 或 "close"。

发送键盘事件

向云端实例发送单个键盘事件（按键按下/抬起），用于模拟物理键盘操作。

```
void tcr_session_send_keyboard_event(TcrSessionHandle session, int32_t keycode, bool down);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄。
keycode	int32_t	按键码（建议使用十进制，可参考 keycode 列表 或 Android KeyEvent 定义）。
down	bool	true 表示按下（KeyDown），false 表示抬起（KeyUp）。

功能说明

该接口用于向云端实例发送单个按键事件，通常需要成对调用（即先发送按下事件，再发送抬起事件），以模拟真实物理键盘的操作。例如，模拟输入字母“A”时，应先调用一次 `down=true`，再调用一次 `down=false`。

常用云手机按键 keycode 对照表（十进制）

按键名称	keycode
返回键	158
菜单键	139
Home 键	172
音量加	58
音量减	59

注意事项

- keycode 建议使用十进制，部分特殊按键可参考上表或 Android KeyEvent 定义。

- 若需模拟连续输入，请依次发送对应的按下/抬起事件。
- 该接口仅发送单个按键事件，复杂输入需自行组合多次调用。

使用示例

模拟点击返回键：

```
tcr_session_send_keyboard_event(session, 158, true);   // 按下
tcr_session_send_keyboard_event(session, 158, false);  // 抬起
```

发送触摸屏触摸事件

```
void tcr_session_touchscreen_touch(TcrSessionHandle session, float x,
float y, int eventType, int width, int height, long timestamp);
```

参数	类型	说明
session	TcrSessionHandle	会话句柄。
x	float	触摸点横坐标（像素）。
y	float	触摸点纵坐标（像素）。
eventType	int	事件类型。 • 0：按下 • 1：移动 • 2：抬起
width	int	屏幕宽度（像素）。
height	int	屏幕高度（像素）。
timestamp	long	事件时间戳（毫秒）。

事件回调

最近更新时间：2025-09-03 14:10:22

SDK 类型(TcrSdkType)

SDK 的工作模式类型。

枚举值	名称	说明
0	CloudPhonePaas	云手机 PaaS 类型。
1	CloudStream	云串流类型。

错误码枚举(TcrErrorCode)

用于标识 SDK 操作的结果状态，便于上层应用根据错误码进行相应处理。

枚举值	数值	说明
TCR_SUCCESS	0	操作成功。
TCR_ERR_TIMEOUT	100010	操作超时，可能由于网络延迟或服务端无响应。
TCR_ERR_INVALID_PARAMS	100011	参数无效，需检查接口参数设置。
TCR_ERR_UNKNOWN	100012	未知错误，建议收集日志排查。
TCR_ERR_INTERNAL_ERROR	100013	SDK 内部错误，通常为系统级异常。
TCR_ERR_STATE_ILLEGAL	100014	状态非法，当前上下文不允许该操作。
TCR_ERR_DATA_CHANNEL_BASE_CODE	102000	数据通道模块错误码起始值。
TCR_ERR_CREATE_FAILURE	102001	数据通道创建失败。
TCR_ERR_CLOSED	102002	数据通道已关闭或失效。

会话配置信息结构体(TcrConfig)

通过云 API 获取的会话配置信息。

字段	类型	说明
token	const char*	访问令牌。
accessInfo	const char*	访问信息。
sdkType	TcrSdkType	SDK 的工作模式类型。
streamOpt	bool	是否开启串流优化，默认为 false。

视频帧定义

视频帧类型(TcrVideoBufferType)

视频缓冲区类型枚举。

枚举值	名称	说明
0	TCR_VIDEO_BUFFER_TYPE_I420	I420格式 CPU 缓冲区。
1	TCR_VIDEO_BUFFER_TYPE_D3D11	D3D11格式 GPU 纹理缓冲区。

视频帧缓冲区(TcrVideoFrameBuffer)

通用视频帧缓冲区，联合体，支持多种视频缓冲区类型。

```
/**
 * @brief 通用视频帧缓冲区联合体
 * 支持多种视频缓冲区类型
 */
typedef struct {
    TcrVideoBufferType type;   ///< 缓冲区类型
    union {
        TcrI420Buffer i420;    ///< I420格式缓冲区
        TcrD3D11Buffer d3d11; ///< D3D11格式缓冲区
    } buffer;                  ///< 具体缓冲区数据
} TcrVideoFrameBuffer;
```

I420缓冲区(TcrI420Buffer)

I420(YUV420P) 格式的视频帧缓冲区。

```
/**
```

```
 * @brief I420(YUV420P)格式的视频帧缓冲区
 * 常用的YUV格式，包含亮度(Y)和色度(U,V)分量
 */
typedef struct {
    const uint8_t* data_y;     ///< 亮度(Y)分量数据指针
    const uint8_t* data_u;     ///< 色度(U)分量数据指针
    const uint8_t* data_v;     ///< 色度(V)分量数据指针
    int32_t stride_y;          ///< Y分量的行跨度(字节数)
    int32_t stride_u;          ///< U分量的行跨度(字节数)
    int32_t stride_v;          ///< V分量的行跨度(字节数)
    int32_t width;             ///< 视频帧的宽度(像素)
    int32_t height;            ///< 视频帧的高度(像素)
} TcrI420Buffer;
```

D3D11缓冲区(TcrD3D11Buffer)

Direct3D 11 视频帧缓冲区。

```
/**
 * @brief D3D11视频帧缓冲区
 * 使用Direct3D 11的GPU纹理格式
 */
typedef struct {
    void* texture;             ///< ID3D11Texture2D指针
    int32_t width;             ///< 视频帧的宽度(像素)
    int32_t height;            ///< 视频帧的高度(像素)
} TcrD3D11Buffer;
```

旋转角度(TcrVideoRotation)

视频旋转角度枚举。

枚举值	名称	说明
0	TCR_VIDEO_ROTATION_0	不旋转(0度)。
90	TCR_VIDEO_ROTATION_90	顺时针90度。
180	TCR_VIDEO_ROTATION_180	顺时针180度。
270	TCR_VIDEO_ROTATION_270	顺时针270度。

摄像头设备信息(TcrCameraDeviceInfo)

摄像头设备详细信息。

```
/**
 * 摄像头设备详细信息
 */
typedef struct TcrCameraDeviceInfo {
    char device_name[256];  // 设备名称
    char device_id[256];    // 设备ID
    TcrVideoCapability capabilities[16]; // 支持的分辨率能力（最多16个）
    int capability_count;   // 能力数量
} TcrCameraDeviceInfo;
```

摄像头能力结构体(TcrVideoCapability)

摄像头视频上行。

```
/**
 * 摄像头能力结构体
 */
typedef struct TcrVideoCapability {
    int width;      // 视频宽度
    int height;     // 视频高度
} TcrVideoCapability;
```

上行视频配置结构体(TcrVideoConfig)

摄像头视频上行。

```
/**
 * @brief 上行视频配置结构体
 */
typedef struct TcrVideoConfig {
    int width;              // 视频宽度  这里只是期望的宽高，实际取决于摄像头支
持的情况
    int height;             // 视频高度
    int max_fps;            // 最大帧率
    int max_bitrate_bps;    // 最大码率
    const char* device_id;// 摄像头设备ID，为空则使用默认摄像头
} TcrVideoConfig;
```

文件上传进度回调

回调函数类型(TcrUploadProgressCallback)

```
typedef void (*TcrUploadProgressCallback)(
    const char* instance_id,
    double total_bytes,
    double transferred_bytes,
    void* user_data);
```

参数	类型	说明
instance_id	const char*	当前操作的实例 ID。
total_bytes	double	文件总大小（字节）。
transferred_bytes	double	已传输字节数。
user_data	void*	用户自定义数据指针。

上传回调结构体(TcrUploadCallback)

字段	类型	说明
on_progress	TcrUploadProgressCallback	进度回调函数。
user_data	void*	用户自定义数据。

示例代码

```
void my_upload_progress(const char* instance_id, double total, double
transferred, void* user_data) {
    printf("上传进度：%.2f%%\n", transferred / total * 100);
}

TcrUploadCallback callback = {
    .on_progress = my_upload_progress,
    .user_data = NULL
};
```

会话事件类型枚举(TcrSessionEvent)

定义了客户端与云端会话过程中可能产生的所有事件类型。

枚举值	数值	说明	事件数据类型
TCR_SESSION_EVENT_STATE_INITED	0	会话已初始化。	-
TCR_SESSION_EVENT_STATE_CONNECTED	1	会话已连接。	-
TCR_SESSION_EVENT_STATE_CLOSED	2	会话已关闭。	字符串（关闭原因）
TCR_SESSION_EVENT_CLIENT_STATS	3	性能数据更新。	JSON 字符串，如： `{"bitrate":123,"fps":30,"packet_lost":0,"rtt":50}`
TCR_SESSION_EVENT_SCREEN_CONFIG_CHANGE	4	云端屏幕配置变更。	JSON 字符串，如： `{"degree":"90_degree","width":1080,"height":1920}`
TCR_SESSION_EVENT_REQUEST_STREAMING_STATUS	5	请求云端流媒体推流结果。	JSON 字符串，如： `{"user":"xxx","code":0,"status":"success"}`
TCR_SESSION_EVENT_HIT_INPUT	6	远端输入框状态变更。	JSON 字符串，如： `{"field_type":"normal_input","text":"abc"}`
TCR_SESSION_EVENT_CAMERA_STATUS	7	摄像头状态变更。	JSON 字符串，如： `{"status":"open_back","height":720,"width":1280}`
TCR_SESSION_EVENT_TRANSMISSION_MESSAGE	8	云端应用透传消息。	JSON 字符串，如： `{"package_name":"com.xxx","msg":"hello"}`
TCR_SESSION_EVENT_SYSTEM_USAGE	9	云端系统资源使用率。	JSON 字符串，如： `{"cpu_usage":10,"mem_usage":20,"gpu_usage":30}`
TCR_SESSION_EVENT_CLIPBOARD_EVENT	10	云端剪贴板内容变更。	JSON 字符串，如：`{"text":"内容"}`

TCR_SESSION_EVENT_NOTIFICATION_EVENT	11	云端设备通知栏新通知。	JSON 字符串，如：`{"package_name":"com.xxx","title":"标题","text":"内容"}`
TCR_SESSION_EVENT_IME_STATUS_CHANGE	12	云端输入法状态变更。	JSON 字符串，如：`{"ime_type":"cloud"}`

数据通道观察者

TcrDataChannelObserver

自定义数据通道观察者结构体。

```
/**
 * @brief 自定义数据通道观察者结构体
 */
typedef struct TcrDataChannelObserver {
    void* user_data;
    void (*on_connected)(void* user_data, const int32_t port);
    void (*on_error)(void* user_data, const int32_t port, const TcrErrorCode& code, const char* msg);
    void (*on_message)(void* user_data, const int32_t port, const uint8_t* data, size_t size);
} TcrDataChannelObserver;
```

- `on_connected` ：数据通道已连接回调。
- `on_error` ：异常回调。
- `on_message` ：消息接收回调。

日志相关

日志级别枚举(TcrLogLevel)

枚举值	数值	说明
TCR_LOG_LEVEL_TRACE	0	TRACE 级别。
TCR_LOG_LEVEL_DEBUG	1	DEBUG 级别。
TCR_LOG_LEVEL_I	2	INFO 级别。

NFO		
TCR_LOG_LEVEL_WARN	3	WARN 级别。
TCR_LOG_LEVEL_ERROR	4	ERROR 级别。

日志回调结构体(TcrLogCallback)

字段	类型	说明
user_data	void*	用户自定义数据。
on_log	void (*)(void*, TcrLogLevel, const char*, const char*)	日志回调函数。

日志回调函数原型

```
void (*on_log)(void* user_data, TcrLogLevel level, const char* tag,
const char* log);
```

参数	类型	说明
user_data	void*	用户自定义数据。
level	TcrLogLevel	日志级别。
tag	const char*	日志标签。
log	const char*	日志内容。

示例代码

```
void my_log_callback(void* user_data, TcrLogLevel level, const char*
tag, const char* log) {
    printf("[%d][%s] %s\n", level, tag, log);
}

TcrLogCallback log_cb = {
    .user_data = NULL,
    .on_log = my_log_callback
};
```

版权所有：腾讯云计算（北京）有限责任公司

iOS SDK
SDK 集成指引

最近更新时间：2025-09-03 14:10:22

iOS SDK 为开发者提供了完整的云手机集成能力，支持将远程云手机的操作界面与控制功能集成至自研应用。

运行环境

iOS 12及以上系统版本。

快速接入

获取 SDK

SDK	下载地址	SDK 说明文档	版本发布日志
iOS SDK	GitHub下载	接口说明	Release notes

SDK 集成

在 Podfile 中添加：

```
# 集成指定版本的SDK，x.x.x需替换为真实版本号
pod 'TCRSDK', :git => "https://github.com/tencentyun/cloudgame-ios-
sdk.git", :tag => 'x.x.x'

# 虚拟按键可选库
# pod 'TCRVKey', :git => "https://github.com/tencentyun/cloudgame-ios-
sdk.git", :tag => 'x.x.x'
```

SDK 使用

```
// 以下是伪代码，
// 完整代码参考 https://github.com/tencentyun/cloudgame-ios-
sdk/blob/master/Demo/TCAIDemo/TCAIDemo/CAIDemoLoginVC.m#L222
#import <TCRSDK/TCRSDK.h>

- (void) quickStart {
    // 调用 云API 创建安卓实例访问Token
    // https://cloud.tencent.com/document/api/1162/119708
```

```
    NSDictionary *androidInstancesAccessToken = [self
CreateAndroidInstancesAccessToken];
    NSString *token = androidInstancesAccessToken[@"Token"];
    NSString *accessInfo = androidInstancesAccessToken[@"AccessInfo"];

    // 使用Token和AceessInfo构建TcrConfig
    TcrConfig* tcrConfig = [[TcrConfig alloc] initWithToken:token
accessInfo:accessInfo];

    // 给TCRSDK设置Token和AccessInfo
    [[TcrSdkInstance sharedInstance] setTcrConfig:tcrConfig error:nil];

    // 创建TcrSession
    NSMutableDictionary *tcrSessionConfig = [NSMutableDictionary
dictionary];
    TcrSession *session = [[TcrSdkInstance sharedInstance]
createSessionWithParams:tcrSessionConfig];
    [session setTcrSessionObserver:self];

    // 发起会话连接控制云端实例
    [session accessWithInstanceId:instanceIds[0]];

    // ........进行云手机操作..........//

    // 销毁TcrSession
    [[TcrSdkInstance sharedInstance] destroySession:session];
}
```

完整 Demo

完整接入代码可 参考 Demo 。

接口说明

最近更新时间：2025-09-03 14:10:22

接口概览

初始化与配置接口

方法	描述
TcrSdkInstance setTcrConfig:error	给 TcrSdk 设置 AccessInfo 和 Token，AccessInfo 和 Token 可以请求 云 API 获取。
TcrSdkInstance setLogger:withMinLevel	设置日志记录器和最小日志级别。
TcrSdkInstance createSessionWithParams	创建一个新的 TcrSession 实例。
TcrSdkInstance destroySession	销毁一个 TcrSession 对象。
TcrSdkInstance getAndroidInstance	获取 Android 实例操作对象。
TcrSdkInstance updateToken	更新所有实例的访问令牌。

云手机操作接口

方法	描述
AndroidInstance setImageEventWithInterval:quality:screenshotWidth:screenshotHeight	设置截图事件。
AndroidInstance stopImageEvent	停止截图事件。
AndroidInstance transMessageWithPackageName:package:	通过安卓 Messenger 发送消息。
AndroidInstance getInstanceImageWithInstanceId	获取实例截图信息。
AndroidInstance uploadWithInstanceId:files:completion	向云端实例上传文件。

AndroidInstance uploadMediaWithInstanceId:files:completion	向云端实例上传媒体文件。
AndroidInstance getInstanceDownloadAddressWithInstanceId:path	获取实例下载地址。
AndroidInstance getInstanceDownloadLogcatAddressWithInstanceId:recentDay	获取 logcat 日志下载地址。
AndroidInstance setLocationWithParams:completion	批量设置设备 GPS 位置信息。
AndroidInstance setResolutionWithParams:completion	批量设置设备分辨率。
AndroidInstance pasteWithParams:completion	批量粘贴文本到设备。
AndroidInstance sendClipboardWithParams:completion	批量发送文本到设备剪贴板。
AndroidInstance shakeWithParams:completion	批量执行摇一摇操作。
AndroidInstance blowWithParams:completion	批量执行吹一吹操作。
AndroidInstance setSensorWithParams:completion	批量设置设备传感器信息。
AndroidInstance sendTransMessageWithParams:completion	批量发送 Messenger 消息。
AndroidInstance describeInstancePropertiesWithParams:completion	批量查询实例属性。
AndroidInstance listUserAppsWithParams:completion	查询已安装第三方应用。
AndroidInstance unInstallByPackageNameWithParams:completion	批量卸载应用。
AndroidInstance startAppWithParams:completion	批量启动应用。
AndroidInstance stopAppWithParams:completion	批量停止应用。
AndroidInstance clearAppDataWithParams:completion	批量清除应用数据。
AndroidInstance enableAppWithParams:completion	批量启用应用。

版权所有：腾讯云计算（北京）有限责任公司

AndroidInstance disableAppWithParams:completion	批量禁用应用。
AndroidInstance startCameraMediaPlayWithParams:completion	批量在摄像头播放媒体文件。
AndroidInstance stopCameraMediaPlayWithParams:completion	批量停止摄像头媒体播放。
AndroidInstance describeCameraMediaPlayStatusWithParams:completion	批量查询摄像头播放状态。
AndroidInstance displayCameraImageWithParams:completion	批量在摄像头显示图片。
AndroidInstance modifyKeepFrontAppStatusWithParams:completion	修改前台应用保活状态。
AndroidInstance describeKeepFrontAppStatusWithParams:completion	查询前台应用保活状态。
AndroidInstance addKeepAliveListWithParams:completion	批量添加后台保活应用。
AndroidInstance removeKeepAliveListWithParams:completion	批量移除后台保活应用。
AndroidInstance setKeepAliveListWithParams:completion	批量设置后台保活列表。
AndroidInstance describeKeepAliveListWithParams:completion	批量查询后台保活列表。
AndroidInstance clearKeepAliveListWithParams:completion	批量清空后台保活列表。
AndroidInstance muteWithParams:completion	批量设置设备静音。
AndroidInstance mediaSearchWithParams:completion	批量搜索媒体库文件。
AndroidInstance rebootWithParams:completion	批量重启设备。
AndroidInstance listAllAppsWithParams:completion	批量查询所有应用列表。
AndroidInstance moveAppBackgroundWithParams:completion	批量关闭应用到后台。

版权所有：腾讯云计算（北京）有限责任公司

AndroidInstance addAppInstallBlackListWithParams:completion	批量新增应用安装黑名单。
AndroidInstance removeAppInstallBlackListWithParams:completion	批量移除应用安装黑名单。
AndroidInstance setAppInstallBlackListWithParams:completion	批量覆盖应用安装黑名单。
AndroidInstance describeAppInstallBlackListWithParams:completion	批量查询应用安装黑名单。
AndroidInstance clearAppInstallBlackListWithParams:completion	批量清空应用安装黑名单。
AndroidInstance getNavVisibleStatusWithParams:completion	批量获取导航栏可见状态。
AndroidInstance getSystemMusicVolumeWithParams:completion	获取系统媒体音量。

会话连接接口

会话事件回调接口

方法	描述
TcrSessionObserver onEvent:eventData	会话事件回调接口。

会话初始化与配置接口

方法	描述
TcrSession initWithParams:andDelegate	初始化会话，设置参数和代理。
TcrSession setTcrSessionObserver	设置会话观察者。

云实例连接接口

方法	描述

TcrSession accessWithInstanceId	启动会话，在本地会话对象初始化成功后，调用此方法以建立和指定 ID 的云手机的连接。在成功建立连接后，此会话对象才能与指定的云手机进行交互。
TcrSession releaseSession	释放会话，客户端释放本地会话资源。云端会在自动检测到心跳消失后释放云端资源。

会话操作接口

方法	描述
TcrSession getRequestId	获取当前连接的请求 ID。
TcrSession setRenderView	设置渲染视图。
TcrSession setVideoSink	设置视频接收器。
TcrSession setAudioSink	设置音频接收器。
TcrSession setEnableAudioPlaying	启用音频播放。
TcrSession setEnableLocalAudio	启用本地音频。
TcrSession setEnableLocalVideo	启用本地视频。

上下行流控制接口

方法	描述
TcrSession pauseStreaming	暂停媒体流。
TcrSession resumeStreaming	恢复媒体流。
TcrSession setLocalVideoProfile	设置本地视频配置。
TcrSession	设置远程音频配置。

setRemoteAudioProfile	
TcrSession pasteText	向云应用粘贴文本。
TcrSession inputText:mode	向云设备输入文本。
TcrSession setDisableCloudInput	禁用云输入。
TcrSession sendCustomAudioData	发送自定义音频数据。

云端外设交互接口

方法	描述
TcrSession getKeyboard	获取云键盘接口。
TcrSession getMouse	获取云鼠标接口。
TcrSession getGamepad	获取云游戏手柄接口。
TcrSession getTouchScreen	获取云触摸屏接口。
TcrSession getMotionSensor	获取云运动传感器接口。
TcrSession sendKeycodeMessage	发送按键消息。

触摸屏接口

方法	描述
TouchScreen touchWithX:y:eventType:fingerID:width:height:timestamp	触发云触屏触摸事件。

运动传感器接口

方法	描述

MotionSensor onSensorData:x:y:z	传感器数据变化事件。
MotionSensor onLocationChanged:latitude	位置变化事件。
MotionSensor onSimulateSensorEvent	模拟传感器事件。

鼠标接口

方法	描述
Mouse onMouseKey:isDown	云鼠标点击事件。
Mouse onMouseScroll	云鼠标滚轮滚动事件。
Mouse onMouseDeltaMove:deltaY	云鼠标相对移动事件。
Mouse onMouseMoveTo:y	云鼠标移动事件。

键盘接口

方法	描述
Keyboard onKeyboard:down	云键盘按键事件。
Keyboard onKeyboard:down:isLeft	区分左右键的云键盘按键事件。
Keyboard checkKeyboardCapsLock	查询云虚拟键盘的大小写锁定状态。
Keyboard resetKeyboardCapsLock	重置云虚拟键盘的大小写锁定状态。
Keyboard resetKeyboard	重置云键盘按键状态。

版权所有：腾讯云计算（北京）有限责任公司

游戏手柄接口

方法	描述
Gamepad connectGamepad	连接虚拟游戏手柄。
Gamepad disconnectGamepad	断开虚拟游戏手柄。
Gamepad onGamepadKey:down	云游戏手柄按键事件。
Gamepad onGamepadStick:x:y	云游戏手柄摇杆事件。
Gamepad onGamepadTrigger:type:value:down	云游戏手柄触发器事件。

音视频帧回调接口

方法	描述
VideoSink onRenderVideoFrame	解码后的视频帧数据回调。
AudioSink onAudioData	音频数据回调。

音频会话代理接口

方法	描述
TCRAudioSessionDelegate onSetCategory:withOptions:error	设置音频会话类别。
TCRAudioSessionDelegate onSetMode:error	设置音频会话模式。
TCRAudioSessionDelegate onSetActive:error	设置音频会话活动状态。

接口详细说明

配置 Token 和 AccessInfo

接口原型

```
- (void)setTcrConfig:(TcrConfig*) tcrConfig error:(NSError**)error;
```

输入参数

参数名称	必选	类型	描述
tcrConfig	是	TcrConfig*	会话配置信息，通过业务后端调用云 API CreateAndroidInstancesAccessToken 获得。
error	否	NSError**	输出错误信息。

返回值

无

调用示例

```
TcrConfig *config = [[TcrConfig alloc] initWithToken:@"your_token"
accessInfo:@"your_access_info"];
NSError *error = nil;
[[TcrSdkInstance sharedInstance] initWithTcrConfig:config error:&error];
if (error) {
    NSLog(@"初始化失败: %@", error);
}
```

设置日志

接口原型

```
+ (void)setLogger:(id<TCRLogDelegate> _Nonnull)logger withMinLevel:
(TCRLogLevel)minLevel;
```

输入参数

参数名称	必选	类型	描述
logger	是	id<TCRLogDelegate>	日志代理。
minLevel	是	TCRLogLevel	最小日志级别。

返回值

无

调用示例

```
id<TCRLogDelegate> logger = ^(TCRLogLevel logLevel, NSString *log) {
    NSLog(@"[SDK Log] Level: %@, Message: %@",
NSStringFromTCRLogLevel(logLevel), log);
};
[TCrSdkInstance setLogger:logger withMinLevel:TCRLogLevelInfo];
```

创建会话

接口原型

```
- (TcrSession*)createSessionWithParams:(NSDictionary*)params;
```

输入参数

参数名称	必选	类型	描述
params	否	NSDictionary*	包含 TcrSession 配置参数的字典。

参数选择如下：

- `"preferredCodec"`：可选值为 `"H264"`，`"H265"`，`"VP8"`，和 `"VP9"`。用于设置首选的编解码器。如果设置了此字段，会尝试使用首选编解码器进行通信。如果首选编解码器不可用，将使用其他可用的编解码器。如果未设置此字段或设置无效，将使用默认编解码器。
- `"preferredCodecList"`：值为 NSArray，包含可选值 `"H264"`，`"H265"`，`"VP8"`，和 `"VP9"`。表示优先选择的编解码器列表，列表中元素的顺序表示优先级。例如，索引0为最高优先级的编解码器。如果无法选择设置的编解码器，则会选择其他可用的编解码器。
- `"idleThreshold"`：空闲检测阈值，即用户无活动的时间。单位为毫秒。当达到此阈值时，将触发事件 `TcrEvent.CLIENT_IDLE`。
- `"local_video"`：可选值为布尔类型。用于启用本地摄像头。
- `"local_audio"`：可选值为布尔类型。用于启用本地麦克风。
- `"enableCustomAudioCapture"`：@{@"sampleRate": NSInteger, @"useStereoInput": BOOL}，启用自定义音频捕获并设置自定义捕获音频的采样率和通道数（两个参数都是必需的）。例如，`@"enableCustomAudioCapture":@{@"sampleRate":@(48000), @"useStereoInput":@(false)}` 表示采样率为48000的单声道数据。此外，要启用自定义音频捕获，还需要将 `"local_audio"` 设置为启用本地音频上行。

返回值

类型	描述
TcrSession*	新创建的 TcrSession 实例，如果创建失败则返回 nil。

调用示例

```
NSDictionary *sessionParams = @{
    @"preferredCodec": @"H264",
    @"idleThreshold": @(30000),
    @"local_video": @YES,
    @"local_audio": @YES
};
TcrSession *session = [[TcrSdkInstance sharedInstance]
createSessionWithParams:sessionParams];
if (!session) {
    NSLog(@"创建会话失败");
}
```

销毁会话

接口原型

```
-(void)destroySession:(TcrSession*) session;
```

输入参数

参数名称	必选	类型	描述
session	是	TcrSession*	要销毁的会话对象。

返回值

无

调用示例

```
[[TcrSdkInstance sharedInstance] destroySession:session];
```

获取 Android 实例

接口原型

```
- (AndroidInstance*)getAndroidInstance;
```

返回值

类型	描述
AndroidInstance*	返回 Android 实例操作对象，如果未初始化则返回 nil。

调用示例

```
AndroidInstance *androidInstance = [[TcrSdkInstance sharedInstance]
getAndroidInstance];
if (!androidInstance) {
    NSLog(@"获取Android实例失败");
}
```

更新令牌

接口原型

```
- (void)updateTokens:(NSString *)token;
```

输入参数

参数名称	必选	类型	描述
token	是	NSString*	新的访问令牌字符串。

返回值

无

调用示例

```
[[TcrSdkInstance sharedInstance] updateTokens:@"new_token"];
```

设置截图事件

接口原型

```
- (void)setImageEventWithInterval:(NSInteger)interval quality:(nullable
NSNumber *)quality screenshotWidth:(int)screenshotWidth
screenshotHeight:(int)screenshotHeight;
```

输入参数

参数名称	必选	类型	描述
interval	是	NSInteger	截图事件的间隔（毫秒）。
quality	否	NSNumber*	截图质量（0-100，可选）。
screenshotWidth	是	int	截图宽度。
screenshotHeight	是	int	截图高度。

调用示例

```
[androidInstance setImageEventWithInterval:1000 quality:@(50)
screenshotWidth:1080 screenshotHeight:1920];
```

停止截图事件

接口原型

```
- (void)stopImageEvent;
```

输入参数

无

调用示例

```
[androidInstance stopImageEvent];
```

通过安卓 Messenger 发送消息

接口原型

```
- (void)transMessageWithPackageName:(NSString *_Nonnull)packageName
message:(NSString *_Nonnull)message;
```

输入参数

参数名称	必选	类型	描述
packageName	是	NSString*	应用包名。
message	是	NSString*	消息内容。

调用示例

```
[androidInstance transMessageWithPackageName:@"com.example.app"
message:@"Hello World"];
```

获取实例截图信息

接口原型

```
- (NSString * _Nullable)getInstanceImageWithInstanceId:(NSString
*_Nonnull)instanceId;
```

输入参数

参数名称	必选	类型	描述
instanceId	是	NSString*	实例 ID。

调用示例

```
NSString *screenshotURL = [androidInstance
getInstanceImageWithInstanceId:@"cai-device-001"];
```

向云端实例上传文件

接口原型

```
- (void)uploadWithInstanceId:(NSString *_Nonnull)instanceId files:
(nullable NSArray<CaiUploadFileItem *> *)files completion: (nullable
CaiUploadFilesCompletion)completion;
```

> ⚠ **注意：**

只允许将文件上传至/sdcard/目录下。

输入参数

参数名称	必选	类型	描述
instanceId	是	NSString*	实例 ID。
files	是	NSArray<CaiUploadFileItem >	上传的文件。
completion	否	CaiUploadFilesCompletion	任务完成的回调。

调用示例

```
NSMutableArray<CaiUploadFileItem*>* files = [NSMutableArray new];
CaiUploadFileItem* itemMp4 = [CaiUploadFileItem new];
itemMp4.fileName = @"movie.mp4";
itemMp4.filePath = @"/sdcard/media";
itemMp4.fileData = [self dataForAssetNamed:@"movie" ofType:@"mp4"];
[files addObject:itemMp4];
[androidInstance uploadWithInstanceId:@"cai-device-001" files:files
completion:^(CaiUploadResponse *response, NSError *error) {
        if (error) {
            NSLog(@"系统错误: %@", error);
            return;
        }

        if (response.code != 0) {
            NSLog(@"上传文件失败: [%ld] %@", (long)response.code,
response.msg);
            return;
        }

        NSLog(@"上传文件成功: %@", response.fileStatus);
}];
```

向云端实例上传媒体文件

接口原型

```
- (void)uploadMediaWithInstanceId:(NSString *_Nonnull)instanceId files:
(nullable NSArray<CaiUploadMediaFileItem *> *)files completion:
```

```
(nullable CaiUploadMediaFilesCompletion)completion;
```

> **说明：**
> 此接口会将文件固定上传至/data/media/0/DCIM，上传完毕会通知系统添加媒体文件到相册中。

输入参数

参数名称	必选	类型	描述
instanceId	是	NSString*	实例 ID。
files	是	NSArray<CaiUploadMediaFileItem >	上传的媒体文件。
completion	否	CaiUploadFilesCompletion	任务完成的回调。

调用示例

```
NSMutableArray<CaiUploadMediaFileItem*>* files = [NSMutableArray new];
CaiUploadMediaFileItem* itemMp4 = [CaiUploadMediaFileItem new];
itemMp4.fileName = @"movie.mp4";
itemMp4.fileData = [self dataForAssetNamed:@"movie" ofType:@"mp4"]; // 加
载文件至NSData*
[files addObject:itemMp4];
[androidInstance uploadWithInstanceId:@"cai-device-001" files:files
completion:^(CaiUploadResponse *response, NSError *error) {
        if (error) {
            NSLog(@"系统错误: %@", error);
            return;
        }

        if (response.code != 0) {
            NSLog(@"上传文件失败: [%ld] %@", (long)response.code,
response.msg);
            return;
        }

        NSLog(@"上传文件成功");
}];
```

获取实例下载地址

接口原型

```
- (NSString *_Nonnull)getInstanceDownloadAddressWithInstanceId:(NSString
*_Nonnull)instanceId path:(NSString *_Nonnull)path;
```

说明：

- 只允许下载/sdcard/目录下文件。
- 下载文件夹时会自动压缩为 zip 进行下载。
- 单实例下载限速10MB/s，同实例多个下载请求共享该限速。

输入参数

参数名称	必选	类型	描述
instanceId	是	NSString*	实例 ID。
path	是	NSString*	下载路径。

调用示例

```
NSString *downloadURL = [androidInstance
getInstanceDownloadAddressWithInstanceId:@"cai-device-001"
path:@"/sdcard/downloads/example.txt"];
```

获取logcat日志下载地址

接口原型

```
- (NSString *_Nonnull)getInstanceDownloadLogcatAddressWithInstanceId:
(NSString *_Nonnull)instanceId recentDay:(int)recentDay;
```

输入参数

参数名称	必选	类型	描述
instanceId	是	NSString*	实例 ID。
recentDay	是	int	最近几天的日志。

调用示例

```
NSString *logcatURL = [androidInstance
getInstanceDownloadLogcatAddressWithInstanceId:@"cai-device-001"
recentDay:7];
```

批量设置设备 GPS 位置信息

接口原型

```
- (void)setLocationWithParams:(NSDictionary<NSString *, NSDictionary *>
*_Nonnull)params completion:(CaiBatchTaskCompletion
_Nullable)completion;
```

输入参数

参数名称	必选	类型	描述

params	是	NSDictionary<NSString , *NSDictionary *>*	位置信息参数字典，结构为：
			{
			@"Longitude": @(经度值), // 经度 (double)
			@"Latitude": @(纬度值) // 纬度 (double)
			}

completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *locations = @{
    @"cai-device-001": @{@"Longitude": @113.32412, @"Latitude":
@23.124124},
    @"cai-device-002": @{@"Longitude": @114.32412, @"Latitude":
@24.124124}
};
[androidInstance setLocationWithParams:locations
completion:^(CaiBatchTaskResponse * _Nonnull response, NSError *
_Nullable error) {
    if (error) {
        NSLog(@"系统错误: %@", error);
        return;
```

```
    }

    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }

    NSLog(@"位置设置成功: %@", response.deviceResponses);
}];
```

批量设置设备分辨率

接口原型

```
- (void)setResolutionWithParams:(NSDictionary<NSString *, NSDictionary
*> *_Nonnull)params completion:(CaiBatchTaskCompletion
_Nullable)completion;
```

输入参数

参数名称	必选	类型	描述

params	是	NSDictionary<NSString , *NSDictionary *>*	分辨率参数字典，结构为：
			{
			"Width": "宽度像素(NSInteger)",
			"Height": "高度像素(NSInteger)",
			"DPI": "像素密度(NSInteger，可选)"
			}

completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *resolutions = @{
    @"cai-device-001": @{@"Width": @720, @"Height": @1280, @"DPI":
@240},
    @"cai-device-002": @{@"Width": @1080, @"Height": @1920}
};
```

```
[androidInstance setResolutionWithParams:resolutions
completion:^(CaiBatchTaskResponse * _Nonnull response, NSError *
_Nullable error) {
    if (error) {
        NSLog(@"系统错误: %@", error);
        return;
    }

    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }

    NSLog(@"设置成功: %@", response.deviceResponses);
}];
```

批量粘贴文本到设备

接口原型

```
- (void)pasteWithParams:(NSDictionary<NSString *, NSDictionary *> *
_Nonnull)params completion:(CaiBatchTaskCompletion _Nullable)completion;
```

输入参数

参数名称	必选	类型	描述

params	是	NSDictionary<NSString , *NSDictionary *>*	粘贴文本参数字典，结构为：
			{
			@"Text": @"粘贴文本" // 要粘贴的文本内容 (NSString)
			}

completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *texts = @{
    @"cai-device-001": @{@"Text": @"Hello World"},
    @"cai-device-002": @{@"Text": @"粘贴测试文本"}
```

```
};
[androidInstance pasteWithParams:texts completion:^(CaiBatchTaskResponse
* _Nonnull response, NSError * _Nullable error) {
    if (error) {
        NSLog(@"系统错误: %@", error);
        return;
    }

    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }

    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量发送文本到设备剪切板

接口原型

```
- (void)sendClipboardWithParams:(NSDictionary<NSString *, NSDictionary
*> * _Nonnull)params completion:(CaiBatchTaskCompletion
_Nullable)completion;
```

输入参数

参数名称	必选	类型	描述

params	是	NSDictionary<NSString , *NSDictionary *>*	剪切板文本参数字典，结构为：
			`{`
			`@"Text": @"剪切板内容" // 要复制到剪切板的文本 (NSString)`
			`}`

completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *clipboardTexts = @{
    @"cai-device-001": @{@"Text": @"剪切板测试内容"},
```

```
    @"cai-device-002": @{@"Text": @"这是第二台设备的文本"}
};
[androidInstance sendClipboardWithParams:clipboardTexts
completion:^(CaiBatchTaskResponse *response, NSError *error) {
    if (error) {
        NSLog(@"系统错误: %@", error);
        return;
    }

    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }

    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量执行摇一摇操作

接口原型

```
- (void)shakeWithParams:(NSDictionary<NSString *, NSDictionary *> *
_Nonnull)params completion:(CaiBatchTaskCompletion _Nullable)completion;
```

输入参数

参数名称	必选	类型	描述
params	是	NSDictionary<NSString , *NSDictionary *>*	设备参数字典，结构为：`@{}`
completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *devices = @{
    @"cai-device-001": @{},
    @"cai-device-002": @{}
};
[androidInstance shakeWithParams:devices
completion:^(CaiBatchTaskResponse *response, NSError *error) {
```

```
    if (error) {
        NSLog(@"系统错误: %@", error);
        return;
    }

    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }

    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量执行吹一吹操作

接口原型

```
- (void)blowWithParams:(NSDictionary<NSString *, NSDictionary *> *
_Nonnull)params completion:(CaiBatchTaskCompletion _Nullable)completion;
```

输入参数

参数名称	必选	类型	描述
params	是	NSDictionary<NSString , *NSDictionary *>*	设备参数字典，结构为：`@{}`
completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *devices = @{
    @"cai-device-001": @{},
    @"cai-device-002": @{}
};
[androidInstance blowWithParams:devices
completion:^(CaiBatchTaskResponse *response, NSError *error) {
    if (error) {
        NSLog(@"系统错误: %@", error);
        return;
    }
```

```
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }

    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量设置设备传感器信息

接口原型

```
- (void)setSensorWithParams:(NSDictionary<NSString *, NSDictionary *>
_Nonnull)params completion:(CaiBatchTaskCompletion _Nullable)completion;
```

输入参数

参数名称	必选	类型	描述

params	是	NSDictionary<NSString，*NSDictionary *>*	操作参数字典，结构为：
			{
			@"Type": sensorType, // 传感器类型字符串（@"accelerometer"/@"gyroscope"）
			@"Values": valuesArray // 传感器数值数组（@[@(xValue), @(yValue), @(zValue)]）
			}

completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *sensorData = @{
    @"cai-device-001": @{@"Type": @"accelerometer", @"Values": @[ @1.5,
@0.2, @9.8 ]},
    @"cai-device-002": @{@"Type": @"gyroscope", @"Values": @[ @0.5,
@0.0, @1.2 ]}
```

```
};
[androidInstance setSensorWithParams:sensorData
completion:^(CaiBatchTaskResponse *response, NSError *error) {
    if (error) {
        NSLog(@"系统错误: %@", error);
        return;
    }

    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }

    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量发送 Messenger 消息

接口原型

```
- (void)sendTransMessageWithParams:(NSDictionary<NSString *,
NSDictionary *> * _Nonnull)params completion:(CaiBatchTaskCompletion
_Nullable)completion;
```

输入参数

参数名称	必选	类型	描述

params	是	NSDictionary<NSString，*NSDictionary* *>	消息参数字典，结构为：
			{
			@"PackageName": @"应用包名", // 接收消息的应用包名 (NSString)
			@"Msg": @"消息内容" // 要发送的自定义消息 (NSString)
			}

completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *messages = @{
    @"cai-device-001": @{@"PackageName": @"com.example.app", @"Msg": @"命令更新"},
    @"cai-device-002": @{@"PackageName": @"com.demo.app", @"Msg": @"数据同步"}
};
[androidInstance sendTransMessageWithParams:messages
completion:^(CaiBatchTaskResponse *response, NSError *error) {
    if (error) {
        NSLog(@"系统错误: %@", error);
        return;
    }

    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }

    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量查询实例属性

接口原型

```
- (void)describeInstancePropertiesWithParams:(NSDictionary<NSString *,
NSDictionary *> * _Nonnull)params
                                  completion:(nullable
CaiDescribeInstancePropertiesCompletion)completion;
```

输入参数

参数名称	必选	类型	描述
params	是	NSDictionary<NSString , *NSDictionary *>*	设备参数字典，结构为：`@{}`
completion	否	CaiDescribeInstancePropertiesCompletion	完成回调。

调用示例

```
NSDictionary *devices = @{@"cai-device-001": @{}, @"cai-device-002":
@{}};
[androidInstance describeInstancePropertiesWithParams:devices
completion:^(CaiDescribeInstancePropertiesResponse *response, NSError
*error) {
    if (error) {
        NSLog(@"系统错误: %@", error);
        return;
    }

    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }

    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

查询已安装第三方应用

接口原型

```
- (void)listUserAppsWithParams:(NSDictionary<NSString *, NSDictionary *>
* _Nonnull)params
                    completion:(CaiListUserAppsCompletion
_Nullable)completion;
```

输入参数

参数名称	必选	类型	描述
params	是	NSDictionary<NSString , *NSDictionary* *>	设备参数字典，结构为 `@{}`
completion	否	CaiListUserAppsCompletion	完成回调。

调用示例

```
NSDictionary *devices = @{@"cai-device-001": @{}, @"cai-device-002":
@{}};
```

```
[androidInstance listUserAppsWithParams:devices
completion:^(CaiListUserAppsResponse *response, NSError *error) {
    if (error) {
        NSLog(@"查询失败: %@", error);
        return;
    }
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    ,NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量卸载应用

接口原型

```
- (void)unInstallByPackageNameWithParams:(NSDictionary<NSString *,
NSDictionary *> * _Nonnull)params
                              completion:(CaiBatchTaskCompletion
_Nullable)completion;
```

输入参数

参数名称	必选	类型	描述

params	是	NSDictionary<NSString , *NSDictionary *>*	卸载参数字典，结构为：
			{
			@"PackageName": @"应用包名" // 要卸载的应用包名 (NSString)
			}

completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *uninstallParams = @{@"cai-device-001": @{@"PackageName":
@"com.unwanted.app"}};
```

```
[androidInstance unInstallByPackageNameWithParams:uninstallParams
completion:^(CaiBatchTaskResponse *response, NSError *error) {
    if (error) {
        NSLog(@"卸载失败: %@", error);
        return;
    }
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量启动应用

接口原型

```
- (void)startAppWithParams:(NSDictionary<NSString *, NSDictionary *> *
_Nonnull)params
                completion:(CaiBatchTaskCompletion
_Nullable)completion;
```

输入参数

参数名称	必选	类型	描述

params	是	NSDictionary<NSString , *NSDictionary *>*	启动参数字典，结构为：
			{
			@"PackageName": @"应用包名", // 应用包名 (NSString)
			@"ActivityName": @"Activity" // Activity名称 (NSString)
			}

completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *startParams = @{@"cai-device-001": @{@"PackageName":
@"com.launch.app", @"ActivityName": @"MainActivity"}};
[androidInstance startAppWithParams:startParams
completion:^(CaiBatchTaskResponse *response, NSError *error) {
    if (error) {
        NSLog(@"启动失败: %@", error);
        return;
    }
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量停止应用

接口原型

```
- (void)stopAppWithParams:(NSDictionary<NSString *, NSDictionary *> *
_Nonnull)params
               completion:(CaiBatchTaskCompletion _Nullable)completion;
```

输入参数

参数名称	必选	类型	描述

params	是	NSDictionary<NSString，*NSDictionary* *>	停止参数字典，结构为：
			{
			@"PackageName": @"应用包名" // 要停止的应用包名 (NSString)
			}

completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *stopParams = @{@"cai-device-001": @{@"PackageName":
@"com.running.app"}};
```

```
[androidInstance stopAppWithParams:stopParams
completion:^(CaiBatchTaskResponse *response, NSError *error) {
    if (error) {
        NSLog(@"停止失败: %@", error);
        return;
    }
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量清除应用数据

接口原型

```
- (void)clearAppDataWithParams:(NSDictionary<NSString *, NSDictionary *>
* _Nonnull)params
                    completion:(CaiBatchTaskCompletion
_Nullable)completion;
```

输入参数

参数名称	必选	类型	描述

params	是	NSDictionary<NSString , *NSDictionary *>*	清除参数字典，结构为：
			`{`
			`@"PackageName": @"应用包名" // 要清除数据的应用包名 (NSString)`
			`}`

completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *clearParams = @{@"cai-device-001": @{@"PackageName":
@"com.cache.heavy"}};
```

```
[androidInstance clearAppDataWithParams:clearParams
completion:^(CaiBatchTaskResponse *response, NSError *error) {
    if (error) {
        NSLog(@"清除失败: %@", error);
        return;
    }
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量启用应用

接口原型

```
- (void)enableAppWithParams:(NSDictionary<NSString *, NSDictionary *> *
_Nonnull)params
                 completion:(CaiBatchTaskCompletion
_Nullable)completion;
```

输入参数

参数名称	必选	类型	描述

params	是	NSDictionary<NSString , *NSDictionary* *>	启用参数字典，结构为：
			{
			@"PackageName": @"应用包名" // 要启用的应用包名 (NSString)
			}

completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *enableParams = @{@"cai-device-001": @{@"PackageName":
@"com.disabled.app"}};
```

```
[androidInstance enableAppWithParams:enableParams
completion:^(CaiBatchTaskResponse *response, NSError *error) {
    if (error) {
        NSLog(@"启用失败: %@", error);
        return;
    }
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量禁用应用

接口原型

```
- (void)disableAppWithParams:(NSDictionary<NSString *, NSDictionary *> *
_Nonnull)params
                  completion:(CaiBatchTaskCompletion
_Nullable)completion;
```

输入参数

参数名称	必选	类型	描述

params	是	NSDictionary<NSString , *NSDictionary *>*	禁用参数字典，结构为：
			{
			@"PackageName": @"应用包名" // 要禁用的应用包名 (NSString)
			}

completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *disableParams = @{@"cai-device-001": @{@"PackageName":
@"com.disabled.app"}};
```

```
[androidInstance disableAppWithParams:disableParams
completion:^(CaiBatchTaskResponse *response, NSError *error) {
    if (error) {
        NSLog(@"禁用失败: %@", error);
        return;
    }
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量在摄像头播放媒体文件

接口原型

```
- (void)startCameraMediaPlayWithParams:(NSDictionary<NSString *,
NSDictionary *> * _Nonnull)params
                            completion:(CaiBatchTaskCompletion
_Nullable)completion;
```

输入参数

参数名称	必选	类型	描述

params	是	NSDictionary<NSString , *NSDictionary* *>	播放参数字典，结构为：
			{
			@"FilePath": @"视频文件路径", // 视频文件路径 (NSString)
			@"Loops": @(循环次数) // 循环次数 (NSInteger，负数表示无限循环)
			}

completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *playParams = @{@"cai-device-001": @{@"FilePath":
@"/sdcard/video.mp4", @"Loops": @3}};
[androidInstance startCameraMediaPlayWithParams:playParams
completion:^(CaiBatchTaskResponse *response, NSError *error) {
    if (error) {
        NSLog(@"播放媒体文件失败: %@", error);
        return;
    }
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量停止摄像头媒体播放

接口原型

```
- (void)stopCameraMediaPlayWithParams:(NSDictionary<NSString *,
NSDictionary *> * _Nonnull)params
                           completion:(CaiBatchTaskCompletion
_Nullable)completion;
```

输入参数

参数名称	必选	类型	描述
params	是	NSDictionary<NSString , *NSDictionary* *>	设备参数字典，结构是：`@{}`
completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *stopParams = @{@"cai-device-001": @{}};
[androidInstance stopCameraMediaPlayWithParams:stopParams
completion:^(CaiBatchTaskResponse *response, NSError *error) {
    if (error) {
        NSLog(@"停止播放媒体文件失败: %@", error);
```

```
        return;
    }
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量查询摄像头播放状态

接口原型

```
- (void)describeCameraMediaPlayStatusWithParams:(NSDictionary<NSString
*, NSDictionary *> * _Nonnull)params
                                     completion:
(CaiDescribeCameraMediaPlayStatusCompletion _Nullable)completion;
```

输入参数

参数名称	必选	类型	描述
params	是	NSDictionary<NSString , *NSDictionary* *>	设备参数字典，结构是：`@{}`
completion	否	CaiDescribeCameraMediaPlayStatusCompletion	任务完成的回调。

调用示例

```
NSDictionary *params = @{@"cai-device-001": @{}};
[androidInstance describeCameraMediaPlayStatusWithParams:params
completion:^(CaiDescribeCameraMediaPlayStatusResponse *response, NSError
*error) {
    if (error) {
        NSLog(@"查询摄像头播放状态失败: %@", error);
        return;
    }
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
```

```
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量在摄像头显示图片

接口原型

```
- (void)displayCameraImageWithParams:(NSDictionary<NSString *,
NSDictionary *> * _Nonnull)params
                          completion:(CaiBatchTaskCompletion
_Nullable)completion;
```

输入参数

参数名称	必选	类型	描述

params	是	NSDictionary<NSString , *NSDictionary *>*	图片参数字典，结构是：
			{
			@"FilePath": @"图片文件路径"
			}

completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *imageParams = @{@"cai-device-001": @{@"FilePath":
@"/sdcard/image.jpg"}};
[androidInstance displayCameraImageWithParams:imageParams
completion:^(CaiBatchTaskResponse *response, NSError *error) {
    if (error) {
        NSLog(@"显示图片失败: %@", error);
        return;
    }
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
```

```
}];
```

修改前台应用保活状态

接口原型

```
- (void)modifyKeepFrontAppStatusWithParams:(NSDictionary<NSString *,
NSDictionary *> * _Nonnull)params
                                completion:(CaiBatchTaskCompletion
_Nullable)completion;
```

输入参数

参数名称	必选	类型	描述

params	是	NSDictionary<NSString , *NSDictionary* *>	保活参数字典，结构为：
			{
			@"PackageName": @"应用包名", // 应用包名 (NSString)
			@"Enable": @(YES/NO), // 是否启用保活 (BOOL)
			@"RestartInterValSeconds": @(秒数) // 重新拉起最长间隔 (NSInteger)
			}

completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *keepFrontAppParams = @{@"cai-device-001":
@{@"PackageName": @"com.example.app", @"Enable": @YES,
@"RestartInterValSeconds": @5}};
[androidInstance modifyKeepFrontAppStatusWithParams:keepFrontAppParams
completion:^(CaiBatchTaskResponse *response, NSError *error) {
    if (error) {
        NSLog(@"修改保活状态失败: %@", error);
        return;
     }
    if (response.code != 0) {
```

```
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

查询前台应用保活状态

接口原型

```
- (void)describeKeepFrontAppStatusWithParams:(NSDictionary<NSString *,
NSDictionary *> * _Nonnull)params
                                  completion:
(CaiDescribeKeepFrontAppStatusCompletion _Nullable)completion;
```

输入参数

参数名称	必选	类型	描述
params	是	NSDictionary<NSString , *NSDictionary* *>	设备参数字典，结构为：`@{}`
completion	否	CaiDescribeKeepFrontAppStatusCompletion	任务完成的回调。

调用示例

```
NSDictionary *params = @{@"cai-device-001": @{}};
[androidInstance describeKeepFrontAppStatusWithParams:params
completion:^(CaiDescribeKeepFrontAppStatusResponse *response, NSError
*error) {
    if (error) {
        NSLog(@"查询保活状态失败: %@", error);
        return;
    }
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
```

```
}];
```

批量添加后台保活应用

接口原型

```
- (void)addKeepAliveListWithParams:(NSDictionary<NSString *,
NSDictionary *> * _Nonnull)params
                        completion:(CaiBatchTaskCompletion
_Nullable)completion;
```

输入参数

参数名称	必选	类型	描述

params	是	NSDictionary<NSString , *NSDictionary *>*	保活参数字典，结构为：
			{
			@"AppList": @[@"应用包名1", @"应用包名2"] // 应用包名数组
			(NSArray<NSString*>)
			}

completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *keepAliveParams = @{@"cai-device-001": @{@"AppList":
@[@"com.wechat", @"com.alipay"]}};
[androidInstance addKeepAliveListWithParams:keepAliveParams
completion:^(CaiBatchTaskResponse *response, NSError *error) {
    if (error) {
        NSLog(@"添加后台保活应用失败: %@", error);
        return;
    }
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
```

```
}];
```

批量移除后台保活应用

接口原型

```
- (void)removeKeepAliveListWithParams:(NSDictionary<NSString *,
NSDictionary *> * _Nonnull)params
                           completion:(CaiBatchTaskCompletion
_Nullable)completion;
```

输入参数

参数名称	必选	类型	描述

params	是	NSDictionary<NSString , *NSDictionary *>*	移除参数字典，结构为：
			{
			@"AppList": @[@"应用包名1", @"应用包名2"] // 应用包名数组 (NSArray<NSString*>)
			}

completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *removeParams = @{@"cai-device-001": @{@"AppList":
@[@"com.wechat", @"com.alipay"]}};
[androidInstance removeKeepAliveListWithParams:removeParams
completion:^(CaiBatchTaskResponse *response, NSError *error) {
    if (error) {
        NSLog(@"移除后台保活应用失败: %@", error);
        return;
    }
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量设置后台保活列表

接口原型

```
- (void)setKeepAliveListWithParams:(NSDictionary<NSString *,
NSDictionary *> * _Nonnull)params
                        completion:(CaiBatchTaskCompletion
_Nullable)completion;
```

输入参数

参数名称	必选	类型	描述

params	是	NSDictionary<NSString , *NSDictionary *>*	保活参数字典，结构为：
			{
			@"AppList": @[@"应用包名1", @"应用包名2"] // 应用包名数组
			(NSArray<NSString*>)
			}

completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *keepAliveListParams = @{@"cai-device-001": @{@"AppList":
@[@"com.wechat", @"com.alipay"]}};
[androidInstance setKeepAliveListWithParams:keepAliveListParams
completion:^(CaiBatchTaskResponse *response, NSError *error) {
    if (error) {
        NSLog(@"设置后台保活列表失败: %@", error);
        return;
    }
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量查询后台保活列表

接口原型

```
- (void)describeKeepAliveListWithParams:(NSDictionary<NSString *,
NSDictionary *> * _Nonnull)params
                             completion:
(CaiDescribeKeepAliveListCompletion _Nullable)completion;
```

输入参数

参数名称	必选	类型	描述
params	是	NSDictionary<NSString , *NSDictionary *>*	设备参数字典，结构为 `@{}`
completion	否	CaiDescribeKeepAliveListCompletion	任务完成的回调。

调用示例

```
NSDictionary *params = @{@"cai-device-001": @{}};
[androidInstance describeKeepAliveListWithParams:params
completion:^(CaiDescribeKeepAliveListResponse *response, NSError *error)
{
    if (error) {
        NSLog(@"查询失败: %@", error);
        return;
    }
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量清空后台保活列表

接口原型

```
- (void)clearKeepAliveListWithParams:(NSDictionary<NSString *,
NSDictionary *> * _Nonnull)params
                          completion:(CaiBatchTaskCompletion
_Nullable)completion;
```

输入参数

参数名称	必选	类型	描述
params	是	NSDictionary<NSString , *NSDictionary *>*	设备参数字典，结构为 `@{}`
completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *params = @{@"cai-device-001": @{}};
[androidInstance clearKeepAliveListWithParams: params
completion:^(CaiBatchTaskResponse *response, NSError *error) {
    if (error) {
        NSLog(@"清空失败: %@", error);
        return;
    }
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量设置设备静音

接口原型

```
- (void)muteWithParams:(NSDictionary<NSString *, NSDictionary *> *
_Nonnull)params
            completion:(CaiBatchTaskCompletion _Nullable)completion;
```

输入参数

参数名称	必选	类型	描述

params	是	NSDictionary<NSString , *NSDictionary *>*	静音参数字典，结构为：
			{
			@"Mute": @(YES/NO) // 静音状态(BOOL，YES:静音， NO:取消静音)
			}，

completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *muteParams = @{@"cai-device-001": @{@"Mute": @YES}};
[androidInstance muteWithParams:muteParams
completion:^(CaiBatchTaskResponse *response, NSError *error) {
    if (error) {
        NSLog(@"设置静音失败: %@", error);
        return;
    }
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量搜索媒体库文件

接口原型

```
- (void)mediaSearchWithParams:(NSDictionary<NSString *, NSDictionary *>
* _Nonnull)params
                   completion:(CaiMediaSearchCompletion
_Nullable)completion;
```

输入参数

参数名称	必选	类型	描述

params	是	NSDictionary<NSString , *NSDictionary *>	搜索参数字典，结构为：
			{
			@"Keyword": @"搜索关键词" // 搜索关键词 (NSString)
			}

completion	否	CaiMediaSearchCompletion	任务完成的回调。

调用示例

```
NSDictionary *searchParams = @{@"cai-device-001": @{@"Keyword": @"搜索关键词"}};
[androidInstance mediaSearchWithParams:searchParams
completion:^(CaiMediaSearchResponse *response, NSError *error) {
    if (error) {
        NSLog(@"搜索失败: %@", error);
        return;
    }
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量重启设备

接口原型

```
- (void)rebootWithParams:(NSDictionary<NSString *, NSDictionary *> *
_Nonnull)params
                 completion:(CaiBatchTaskCompletion _Nullable)completion;
```

输入参数

参数名称	必选	类型	描述
params	是	NSDictionary<NSString , *NSDictionary *>	设备参数字典，结构为 @{}

completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *rebootParams = @{@"cai-device-001": @{}};
[androidInstance rebootWithParams:rebootParams
completion:^(CaiBatchTaskResponse *response, NSError *error) {
    if (error) {
        NSLog(@"重启失败: %@", error);
        return;
    }
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量查询所有应用列表

接口原型

```
- (void)listAllAppsWithParams:(NSDictionary<NSString *, NSDictionary *>
* _Nonnull)params
                   completion:(CaiListAllAppsCompletion
_Nullable)completion;
```

输入参数

参数名称	必选	类型	描述
params	是	NSDictionary<NSString，*NSDictionary *>*	设备参数字典，结构为 `@{}`
completion	否	CaiListAllAppsCompletion	任务完成的回调。

调用示例

```
NSDictionary *params = @{@"cai-device-001": @{}};
```

```
[androidInstance listAllAppsWithParams:params
completion:^(CaiListAllAppsResponse *response, NSError *error) {
    if (error) {
        NSLog(@"查询失败: %@", error);
        return;
    }
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量关闭应用到后台

接口原型

```
- (void)moveAppBackgroundWithParams:(NSDictionary<NSString *,
NSDictionary *> * _Nonnull)params
                         completion:(CaiBatchTaskCompletion
_Nullable)completion;
```

输入参数

参数名称	必选	类型	描述
params	是	NSDictionary<NSString , *NSDictionary **>	设备参数字典，结构为 `@{}`
completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *params = @{@"cai-device-001": @{}};
[androidInstance moveAppBackgroundWithParams: params
completion:^(CaiBatchTaskResponse *response, NSError *error) {
    if (error) {
        NSLog(@"移动应用到后台失败: %@", error);
        return;
    }
    if (response.code != 0) {
```

```
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量新增应用安装黑名单

接口原型

```
- (void)addAppInstallBlackListWithParams:(NSDictionary<NSString *,
NSDictionary *> * _Nonnull)params
                                completion:(CaiBatchTaskCompletion
_Nullable)completion;
```

输入参数

参数名称	必选	类型	描述

params	是	NSDictionary<NSString , *NSDictionary* *>	黑名单参数字典，结构为：
			`{`
			`@"AppList": @[@"应用包名1", @"应用包名2"]  // 应用包名数组`
			`}`

completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *blacklistParams = @{@"cai-device-001": @{@"AppList":
@[@"com.example.app"]}};
[androidInstance addAppInstallBlackListWithParams: blacklistParams
completion:^(CaiBatchTaskResponse *response, NSError *error) {
    if (error) {
        NSLog(@"新增应用安装黑名单失败: %@", error);
        return;
    }
    if (response.code != 0) {
```

```
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量移除应用安装黑名单

接口原型

```
- (void)removeAppInstallBlackListWithParams:(NSDictionary<NSString *,
NSDictionary *> * _Nonnull)params
                                 completion:(CaiBatchTaskCompletion
_Nullable)completion;
```

输入参数

参数名称	必选	类型	描述

params	是	NSDictionary<NSString , *NSDictionary *>*	黑名单参数字典，结构为：
			`{`
			`@"AppList": @[@"应用包名1", @"应用包名2"] // 应用包名数组`
			`}`

completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *blacklistParams = @{@"cai-device-001": @{@"AppList":
@[@"com.example.app"]}};
[androidInstance removeAppInstallBlackListWithParams: blacklistParams
completion:^(CaiBatchTaskResponse *response, NSError *error) {
    if (error) {
        NSLog(@"移除应用安装黑名单失败: %@", error);
        return;
    }
    if (response.code != 0) {
```

```
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量覆盖应用安装黑名单

接口原型

```
- (void)setAppInstallBlackListWithParams:(NSDictionary<NSString *,
NSDictionary *> * _Nonnull)params
                                 completion:(CaiBatchTaskCompletion
_Nullable)completion;
```

输入参数

参数名称	必选	类型	描述

params	是	NSDictionary<NSString , *NSDictionary *>*	黑名单参数字典，结构为：
			`{`
			`@"AppList": @[@"应用包名1", @"应用包名2"] // 应用包名数组`
			`}`

completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *blacklistParams = @{@"cai-device-001": @{@"AppList":
@[@"com.example.app"]}};
[androidInstance setAppInstallBlackListWithParams: blacklistParams
completion:^(CaiBatchTaskResponse *response, NSError *error) {
    if (error) {
        NSLog(@"覆盖应用安装黑名单失败: %@", error);
        return;
    }
    if (response.code != 0) {
```

```
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量查询应用安装黑名单

接口原型

```
- (void)describeAppInstallBlackListWithParams:(NSDictionary<NSString *,
NSDictionary *> * _Nonnull)params
                                   completion:
(CaiDescribeAppInstallBlackListCompletion _Nullable)completion;
```

输入参数

参数名称	必选	类型	描述
params	是	NSDictionary<NSString , *NSDictionary* *>	设备参数字典，结构为 `@{}`
completion	否	CaiDescribeAppInstallBlackListCompletion	任务完成的回调。

调用示例

```
NSDictionary *params = @{@"cai-device-001": @{}};
[androidInstance describeAppInstallBlackListWithParams: params
completion:^(CaiDescribeAppInstallBlackListResponse *response, NSError
*error) {
    if (error) {
        NSLog(@"查询应用安装黑名单失败: %@", error);
        return;
    }
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
```

```
}];
```

批量清空应用安装黑名单

接口原型

```
- (void)clearAppInstallBlackListWithParams:(NSDictionary<NSString *,
NSDictionary *> * _Nonnull)params
                                completion:(CaiBatchTaskCompletion
_Nullable)completion;
```

输入参数

参数名称	必选	类型	描述
params	是	NSDictionary<NSString , *NSDictionary *>*	设备参数字典，结构为 `@{}`
completion	否	CaiBatchTaskCompletion	任务完成的回调。

调用示例

```
NSDictionary *params = @{@"cai-device-001": @{}};
[androidInstance clearAppInstallBlackListWithParams: params
completion:^(CaiBatchTaskResponse *response, NSError *error) {
    if (error) {
        NSLog(@"清空应用安装黑名单失败: %@", error);
        return;
    }
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

批量获取导航栏可见状态

接口原型

```
- (void)getNavVisibleStatusWithParams:(NSDictionary<NSString *,
NSDictionary *> * _Nonnull)params
                                completion:
(CaiGetNavVisibleStatusCompletion _Nullable)completion;
```

输入参数

参数名称	必选	类型	描述
params	是	NSDictionary<NSString , *NSDictionary *>*	设备参数字典，结构为 `@{}`
completion	否	CaiGetNavVisibleStatusCompletion	任务完成的回调。

调用示例

```
NSDictionary *params = @{@"cai-device-001": @{}};
[androidInstance getNavVisibleStatusWithParams: params
completion:^(CaiGetNavVisibleStatusResponse *response, NSError *error) {
    if (error) {
        NSLog(@"获取导航栏可见状态失败: %@", error);
        return;
    }
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

获取系统媒体音量

接口原型

```
- (void)getSystemMusicVolumeWithParams:(NSDictionary<NSString *,
NSDictionary *> * _Nonnull)params
                                completion:
(CaiGetSystemMusicVolumeCompletion _Nullable)completion;
```

输入参数

参数名称	必选	类型	描述
params	是	NSDictionary<NSString，*NSDictionary *>*	设备参数字典，结构为 `@{}`
completion	否	CaiGetSystemMusicVolumeCompletion	任务完成的回调。

调用示例

```
NSDictionary *params = @{@"cai-device-001": @{}};
[androidInstance getSystemMusicVolumeWithParams: params
completion:^(CaiGetSystemMusicVolumeResponse *response, NSError *error)
{
    if (error) {
        NSLog(@"获取系统媒体音量失败: %@", error);
        return;
    }
    if (response.code != 0) {
        NSLog(@"整体操作失败: [%ld] %@", (long)response.code,
response.message);
        return;
    }
    NSLog(@"操作成功: %@", response.deviceResponses);
}];
```

会话事件回调

用于监听 TcrSession 内部事件。

接口原型

```
- (void)onEvent:(TcrEvent)event eventData:(id _Nullable)eventData;
```

输入参数

参数名称	必选	类型	描述
event	是	TcrEvent	事件类型。
eventData	否	id	事件数据。

调用示例

```
id<TcrSessionObserver> observer = ^(TcrEvent event, id eventData) {
    switch (event) {
        case STATE_CONNECTED:
            NSLog(@"会话已连接");
            break;
        case STATE_RECONNECTING:
            NSLog(@"会话重连中");
            break;
        case STATE_CLOSED:
            NSLog(@"会话关闭");
            break;
        case CLIENT_STATS:
            info = (NSDictionary *)eventData;
            NSLog(@"SDK性能数据: %@", info);
            break;

        // 其他事件处理...
        default:
            break;
    }
};
TcrSession *session = [[TcrSession alloc] initWithParams:nil
andDelegate:observer];
```

会话初始化

接口原型

```
- (instancetype _Nonnull)initWithParams:(NSDictionary *_Nullable)params
andDelegate:(id<TcrSessionObserver> _Nonnull)Observer;
```

输入参数

参数名称	必选	类型	描述
params	否	NSDictionary*	会话参数。
Observer	是	id<TcrSessionObserver>	会话观察者。

参数字典选择如下：

- `"preferredCodec"`：可选值为 `"H264"`，`"H265"`，`"VP8"`，和 `"VP9"`。用于设置首选的编解码器。如果设置了此字段，会尝试使用首选编解码器进行通信。如果首选编解码器不可用，将使用其他可用的编解码器。如果未设置此字段或设置无效，将使用默认编解码器。
- `"preferredCodecList"`：值为 NSArray，包含可选值 `"H264"`，`"H265"`，`"VP8"`，和 `"VP9"`。表示优先选择的编解码器列表，列表中元素的顺序表示优先级，例如索引0为最高优先级的编解码器。如果无法选择设置的编解码器，则会选择其他可用的编解码器。
- `"idleThreshold"`：空闲检测阈值，即用户无活动的时间。单位为毫秒。当达到此阈值时，将触发事件 `TcrEvent.CLIENT_IDLE`。
- `"local_video"`：可选值为布尔类型。用于启用本地摄像头。
- `"local_audio"`：可选值为布尔类型。用于启用本地麦克风。
- `"enableCustomAudioCapture"`：@{@"sampleRate": NSInteger, @"useStereoInput": BOOL}，启用自定义音频捕获并设置自定义捕获音频的采样率和通道数（两个参数都是必需的）。例如，`@"enableCustomAudioCapture":@{@"sampleRate":@(48000), @"useStereoInput":@(false)}` 表示采样率为48000的单声道数据。此外，要启用自定义音频捕获，还需要将 `"local_audio"` 设置为启用本地音频上行。

返回值

类型	描述
TcrSession*	返回初始化后的会话实例。

调用示例

```
NSDictionary *params = @{@"idleThreshold": @(6000)};
id<TcrSessionObserver> observer = ^(TcrEvent event, id eventData) {
    // 处理会话事件
};
TcrSession *session = [[TcrSession alloc] initWithParams: params
andDelegate: observer];
```

设置会话观察者

用于监听 TcrSession 内部事件。

接口原型

```
- (void)setTcrSessionObserver:(id<TcrSessionObserver> _Nonnull)Observer;
```

输入参数

参数名称	必选	类型	描述
Observer	是	id<TcrSessionObserver>	会话观察者。

返回值

无

调用示例

```
id<TcrSessionObserver> observer = ^(TcrEvent event, id eventData) {
    // 处理会话事件
};
[session setTcrSessionObserver:observer];
```

连接实例

接口原型

```
- (void)accessWithInstanceId:(NSString *_Nonnull)instanceId;
```

输入参数

参数名称	必选	类型	描述
instanceId	是	NSString*	云实例 ID。

返回值

无

调用示例

```
[session accessWithInstanceId:@"cai-device-001"];
```

释放会话

接口原型

```
- (void)releaseSession;
```

输入参数

无

返回值

无

调用示例

```
[session releaseSession];
```

获取请求 ID

接口原型

```
- (NSString*_Nonnull)getRequestId;
```

返回值

类型	描述
NSString*	当前会话的请求 ID。

调用示例

```
NSString *requestId = [session getRequestId];
NSLog(@"当前会话的请求ID: %@", requestId);
```

设置渲染视图

设置本次会话的渲染视图，SDK 会将流内容渲染到该视图上。

接口原型

```
- (void)setRenderView:(TcrRenderView *_Nullable)renderView;
```

输入参数

参数名称	必选	类型	描述
renderView	否	TcrRenderView*	渲染视图。

返回值

无

调用示例

```
TcrRenderView *renderView = [[TcrRenderView alloc]
initWithFrame:self.view.frame] // 实例化TcrRenderView对象
[session setRenderView:renderView];
```

设置视频接收器

为本次会话设置一个视频接收器。设置完成后，SDK 会将解码后的视频帧数据回调到该视频接收器。

接口原型

```
- (void)setVideoSink:(id<VideoSink> _Nullable)videoSink;
```

输入参数

参数名称	必选	类型	描述
videoSink	否	id<VideoSink>	视频接收器。

返回值

无

调用示例

```
id<VideoSink> videoSink = ...; // 实现VideoSink协议的对象
[session setVideoSink:videoSink];
```

设置音频接收器

为本次会话设置一个音频接收器。设置完成后，SDK 会将音频数据回调到该音频接收器。

接口原型

```
- (void)setAudioSink:(id<AudioSink> _Nullable)audioSink;
```

输入参数

参数名称	必选	类型	描述

audioSink	否	id<AudioSink>	音频接收器。

返回值

无

调用示例

```
id<AudioSink> audioSink = ...; // 实现AudioSink协议的对象
[session setAudioSink:audioSink];
```

启用音频播放

接口原型

```
- (void)setEnableAudioPlaying:(BOOL)enable;
```

输入参数

参数名称	必选	类型	描述
enable	是	BOOL	是否启用音频播放。

返回值

无

调用示例

```
[session setEnableAudioPlaying:YES]; // 启用音频播放
```

启用本地音频

启用或禁用从麦克风捕获的本地音轨。启用后，本地捕获的音频会上行至云端。

接口原型

```
- (void)setEnableLocalAudio:(BOOL)enable;
```

输入参数

参数名称	必选	类型	描述

enable	是	BOOL	是否启用本地音频。

返回值

无

调用示例

```
[session setEnableLocalAudio:YES]; // 启用本地音频
```

启用本地视频

启用或禁用从摄像头捕获的本地视频。启用后，本地捕获的视频画面会上行至云端。

接口原型

```
- (void)setEnableLocalVideo:(BOOL)enable;
```

输入参数

参数名称	必选	类型	描述
enable	是	BOOL	是否启用本地视频。

返回值

无

调用示例

```
[session setEnableLocalVideo:YES]; // 启用本地视频
```

暂停音视频流

接口原型

```
- (void)pauseStreaming;
```

调用示例

```
[session pauseStreaming];
```

恢复音视频流

接口原型

```
- (void)resumeStreaming;
```

调用示例

```
[session resumeStreaming];
```

设置上行视频配置

接口原型

```
- (void)setLocalVideoProfile:(int)width height:(int)height fps:(int)fps
minBitrate:(int)minBitrate maxBitrate:(int)maxBitrate isFrontCamera:
(BOOL)isFrontCamera;
```

输入参数

参数名称	必选	类型	描述
width	是	int	视频宽度。
height	是	int	视频高度。
fps	是	int	帧率。
minBitrate	是	int	最小比特率。
maxBitrate	是	int	最大比特率。
isFrontCamera	是	BOOL	是否为前置摄像头。

返回值

无

调用示例

```
[session setLocalVideoProfile:1280 height:720 fps:30 minBitrate:1000
maxBitrate:15000 isFrontCamera:NO];
```

版权所有：腾讯云计算（北京）有限责任公司

设置云端音频配置

接口原型

```
- (void)setRemoteAudioProfile:(float)volume;
```

输入参数

参数名称	必选	类型	描述
volume	是	float	云端音频音量。

返回值

无

调用示例

```
[session setRemoteAudioProfile:5.0]; // 设置远程音频流的音量为5.0
```

向云应用粘贴文本

接口原型

```
- (void)pasteText:(NSString *_Nonnull)text;
```

输入参数

参数名称	必选	类型	描述
text	是	NSString*	要粘贴的文本内容。

返回值

无

调用示例

```
[session pasteText:@"Hello, Cloud!"];
```

向云设备输入文本

此方法仅在客户端切换到**本地输入法模式**时可用。它会将文本直接发送到云端设备的输入框，并且不会更改本地剪贴板的内容。

接口原型

```
- (void)inputText:(NSString *_Nonnull)text mode:(BOOL)append;
```

输入参数

参数名称	必选	类型	描述
text	是	NSString*	要输入的文本内容。
append	是	BOOL	写入模式控制，是否追加写入。

返回值

无

调用示例

```
[session inputText:@"Hello" mode:YES]; // 追加模式（默认）
[session inputText:@"New Text" mode:NO]; // 覆盖模式
```

禁用云端输入

当云输入被禁用时，云应用程序的软键盘将不会出现。

接口原型

```
- (void)setDisableCloudInput:(BOOL)disableCloudInput;
```

输入参数

参数名称	必选	类型	描述
disableCloudInput	是	BOOL	是否禁用云端输入。

返回值

无

调用示例

```
[session setDisableCloudInput:YES]; // 禁用云输入
```

发送自定义音频数据

此方法仅在启用自定义音频捕获时有效。

接口原型

```
- (BOOL)sendCustomAudioData:(NSData *_Nonnull)audioData captureTimeNs:
(uint64_t)captureTimeNs;
```

输入参数

参数名称	必选	类型	描述
audioData	是	NSData*	要发送的 PCM 音频数据。
captureTimeNs	是	uint64_t	音频捕获时间（纳秒）。

返回值

类型	描述
BOOL	是否成功发送自定义音频数据。

调用示例

```
NSData *audioData = ...; // 获取音频数据
uint64_t captureTimeNs = ...; // 音频捕获时间
BOOL result = [session sendCustomAudioData:audioData
captureTimeNs:captureTimeNs];
if (result) {
    NSLog(@"成功发送自定义音频数据");
} else {
    NSLog(@"发送自定义音频数据失败");
}
```

获取键盘接口

接口原型

```
- (id<Keyboard> _Nonnull)getKeyboard;
```

返回值

类型	描述
id<Keyboard>	云键盘接口。

调用示例

```
id<Keyboard> keyboard = [session getKeyboard];
[keyboard onKeyboard:65 down:YES]; // 模拟按下键盘的A键
[keyboard onKeyboard:65 down:NO]; // 模拟松开键盘的A键
```

获取鼠标接口

接口原型

```
- (id<Mouse> _Nonnull)getMouse;
```

返回值

类型	描述
id<Mouse>	云鼠标接口。

调用示例

```
id<Mouse> mouse = [session getMouse];
[mouse onMouseKey:LEFT isDown:YES]; // 模拟按下鼠标左键
[mouse onMouseKey:LEFT isDown:NO]; // 模拟松开鼠标左键
```

获取游戏手柄接口

接口原型

```
- (id<Gamepad> _Nonnull)getGamepad;
```

返回值

类型	描述
id<Gamepad>	云游戏手柄接口。

调用示例

```
id<Gamepad> gamepad = [session getGamepad];
[gamepad onGamepadKey:4096 down:YES]; // 模拟按下游戏手柄的A键
[gamepad onGamepadKey:4096 down:NO]; // 模拟松开游戏手柄的A键
```

获取触摸屏接口

接口原型

```
- (id<TouchScreen> _Nonnull)getTouchScreen;
```

返回值

类型	描述
id<TouchScreen>	云触摸屏接口。

调用示例

```
id<TouchScreen> touchscreen = [session getTouchScreen];
[touchscreen touchWithX:@(100) y:@(200) eventType:0 fingerID:1
width:1920 height:1080 timestamp:1669253200];
```

获取运动传感器接口

接口原型

```
- (id<MotionSensor> _Nonnull)getMotionSensor;
```

返回值

类型	描述
id<MotionSensor>	云运动传感器接口。

调用示例

```
id<MotionSensor> motionSensor = [session getMotionSensor];
[motionSensor onSensorData:ACCELEROMETER x:0.0 y:0.0 z:0.0];
```

发送按键消息

接口原型

```
- (void)sendKeycodeMessage:(NSDictionary *_Nonnull)msg;
```

输入参数

参数名称	必选	类型	描述
msg	是	NSDictionary*	要发送的按键消息。

返回值

无

调用示例

```
NSDictionary *msg = @{@"keycode": @(65), @"down": @YES}; // 模拟按下键盘的A
键
[session sendKeycodeMessage:msg];
```

设置音频会话代理

接口原型

```
+ (void)setAudioSessionDelegate:
(id<TCRAudioSessionDelegate>_Nonnull)delegate;
```

输入参数

参数名称	必选	类型	描述
delegate	是	id<TCRAudioSessionDelegate>	音频会话代理。

返回值

无

调用示例

```
[session setAudioSessionDelegate:delegate];
```

触发云触屏触摸事件

接口原型

```
- (void)touchWithX:(NSNumber *)x
                 y:(NSNumber *)y
         eventType:(int)eventType
          fingerID:(int)fingerID
             width:(int)width
            height:(int)height
         timestamp:(long)timestamp;
```

输入参数

参数名称	必选	类型	描述
x	是	NSNumber*	触摸点在视频流坐标系中的 X 坐标值。
y	是	NSNumber*	触摸点在视频流坐标系中的 Y 坐标值。
eventType	是	int	事件类型，0为按下，1为移动，2为抬起。
fingerID	是	int	与特定指针数据索引关联的指针标识符。
width	是	int	视频流的宽度。
height	是	int	视频流的高度。
timestamp	是	long	事件发生的时间戳。

调用示例

```
[myTouchScreen touchWithX:@(100) y:@(200) eventType:0 fingerID:1
width:1920 height:1080 timestamp:1669253200];
```

传感器数据变化事件

接口原型

```
- (void)onSensorData:(SensorType)key x:(double)x y:(double)y z:
(double)z;
```

输入参数

参数名称	必选	类型	描述
key	是	SensorType	传感器类型。
x	是	double	X 轴坐标值。
y	是	double	Y 轴坐标值。
z	是	double	Z 轴坐标值。

调用示例

```
[myMotionSensor onSensorData:ACCELEROMETER x:0.0 y:0.0 z:0.0];
```

位置变化事件

接口原型

```
- (void)onLocationChanged:(double)longitude latitude:(double)latitude;
```

输入参数

参数名称	必选	类型	描述
longitude	是	double	经度值。
latitude	是	double	纬度值。

调用示例

```
[myMotionSensor onLocationChanged:116.39153 latitude:39.90742];
```

模拟传感器事件

接口原型

```
- (void)onSimulateSensorEvent:(SensorType)type;
```

输入参数

参数名称	必选	类型	描述
type	是	SensorType	事件类型。

调用示例

```
[myMotionSensor onSimulateSensorEvent:GYRO];
```

云鼠标点击事件

接口原型

```
- (void)onMouseKey:(TCRMouseKeyType)key isDown:(BOOL)down;
```

输入参数

参数名称	必选	类型	描述
key	是	TCRMouseKeyType	鼠标按键类型。
down	是	BOOL	是否按下。

调用示例

```
[myMouse onMouseKey:LEFT isDown:YES];
```

云鼠标滚轮滚动事件

接口原型

```
- (void)onMouseScroll:(float)delta;
```

输入参数

参数名称	必选	类型	描述

delta	是	float	滚轮滚动的偏移量。

调用示例

```
[myMouse onMouseScroll:-1.0];
```

云鼠标相对移动事件

接口原型

```
- (void)onMouseDeltaMove:(int)deltaX deltaY:(int)deltaY;
```

输入参数

参数名称	必选	类型	描述
deltaX	是	int	水平轴的偏移量。
deltaY	是	int	垂直轴的偏移量。

调用示例

```
[myMouse onMouseDeltaMove:10 deltaY:-5];
```

云鼠标移动事件

接口原型

```
- (void)onMouseMoveTo:(int)x y:(int)y;
```

输入参数

参数名称	必选	类型	描述
x	是	int	远程设备上的 X 坐标。
y	是	int	远程设备上的 Y 坐标。

调用示例

```
[myMouse onMouseMoveTo:100 y:200];
```

云键盘按键事件

接口原型

```
- (void)onKeyboard:(NSInteger)keycode down:(BOOL)down;
```

输入参数

参数名称	必选	类型	描述
keycode	是	NSInteger	键码。
down	是	BOOL	是否按下。

调用示例

```
[myKeyboard onKeyboard:65 down:YES]; // A键按下
```

区分左右键的云键盘按键事件

接口原型

```
- (void)onKeyboard:(NSInteger)keycode down:(BOOL)down isLeft:
(BOOL)isLeft;
```

输入参数

参数名称	必选	类型	描述
keycode	是	NSInteger	键码。
down	是	BOOL	是否按下。
isLeft	是	BOOL	是否为左键。

调用示例

```
[myKeyboard onKeyboard:16 down:YES isLeft:YES]; // 左侧shift键按下
[myKeyboard onKeyboard:16 down:NO isLeft:YES]; // 左侧shift键抬起
```

查询云虚拟键盘的大小写锁定状态

接口原型

```
- (void)checkKeyboardCapsLock:(void (^)(int retCode))finishBlk;
```

输入参数

参数名称	必选	类型	描述
finishBlk	是	void (^)(int retCode)	回调块，用于返回大小写锁定状态。

调用示例

```
[myKeyboard checkKeyboardCapsLock:^(int retCode) {
    NSLog(@"Caps Lock status: %d", retCode);
}];
```

重置云虚拟键盘的大小写锁定状态

接口原型

```
- (void)resetKeyboardCapsLock;
```

调用示例

```
[myKeyboard resetKeyboardCapsLock];
```

重置云键盘按键状态

接口原型

```
- (void)resetKeyboard;
```

调用示例

```
[myKeyboard resetKeyboard];
```

连接虚拟游戏手柄

版权所有：腾讯云计算（北京）有限责任公司

接口原型

```
- (void)connectGamepad;
```

调用示例

```
[myGamepad connectGamepad];
```

断开虚拟游戏手柄

接口原型

```
- (void)disconnectGamepad;
```

调用示例

```
[myGamepad disconnectGamepad];
```

云游戏手柄按键事件

接口原型

```
- (void)onGamepadKey:(NSInteger)keycode down:(BOOL)down;
```

输入参数

参数名称	必选	类型	描述
keycode	是	NSInteger	键码。
down	是	BOOL	是否按下。

调用示例

```
[myGamepad onGamepadKey:0x41 down:YES]; // A键按下
```

云游戏手柄摇杆事件

接口原型

```
- (void)onGamepadStick:(KeyType)type x:(NSInteger)x y:(NSInteger)y;
```

输入参数

参数名称	必选	类型	描述
type	是	KeyType	摇杆类型。
x	是	NSInteger	摇杆 X 坐标。
y	是	NSInteger	摇杆 Y 坐标。

调用示例

```
[myGamepad onGamepadStick:LS x:10 y:-20]; // 左摇杆向左下移动
```

云游戏手柄触发器事件

接口原型

```
- (void)onGamepadTrigger:(KeyType)type value:(NSInteger)value down:
(BOOL)down;
```

输入参数

参数名称	必选	类型	描述
type	是	KeyType	触发器类型。
value	是	NSInteger	触发器的绝对位置。
down	是	BOOL	是否按下。

调用示例

```
[myGamepad onGamepadTrigger:LT value:128 down:YES]; // LT触发器半压
```

音频会话代理

接口原型

```
- (BOOL)onSetCategory:(NSString *_Nonnull)category withOptions:
(AVAudioSessionCategoryOptions)options error:(NSError *_Nullable
*_Nullable)outError;
- (BOOL)onSetMode:(NSString *_Nonnull)mode error:(NSError *_Nullable
*_Nullable)outError;
- (BOOL)onSetActive:(BOOL)active error:(NSError *_Nullable
*_Nullable)outError;
```

输入参数

参数名称	必选	类型	描述
category	是	NSString*	音频会话类别。
mode	是	NSString*	音频会话模式。
active	是	BOOL	音频会话活动状态。
outError	否	NSError**	输出错误信息。

返回值

类型	描述
BOOL	是否成功设置。

调用示例

```
[TCRSession setAudioSessionDelegate:delegate];
```

解码后的视频帧数据回调

接口原型

```
- (void)onRenderVideoFrame:(TCRVideoFrame *)frame;
```

输入参数

参数名称	必选	类型	描述
frame	是	TCRVideoFrame*	视频帧对象。

调用示例

```
- (void)onRenderVideoFrame:(TCRVideoFrame *)frame {
    // 在这里处理解码后的视频帧数据
    NSLog(@"Rendering video frame with dimensions %dx%d", frame.width,
frame.height);
}
```

音频数据回调

接口原型

```
- (void)onAudioData:(TCRAudioFrame *)data;
```

输入参数

参数名称	必选	类型	描述
data	是	TCRAudioFrame*	音频帧对象。

调用示例

```
- (void)onAudioData:(TCRAudioFrame *)data {
    // 在这里处理音频数据
    NSLog(@"Received audio data with %d samples, %d channels, and %d
bits per sample", data.frames, data.channels, data.bitsPerSample);
}
```

数据结构概览

数据结构	描述
TCRVideoFrame	表示视频帧的数据结构。
TCRAudioFrame	表示音频帧的数据结构。
CaiDeviceResponse	表示单个设备操作的响应结果。
CaiBatchTaskResponse	表示批量操作的响应结果。
CaiDeviceInfo	包含设备的品牌和型号信息。
CaiGPSInfo	表示设备的地理位置。

CaiLanguageInfo	表示设备的语言设置。
CaiLocaleInfo	表示设备的地区设置。
CaiProxyInfo	表示设备的网络代理配置。
CaiSIMInfo	表示设备的 SIM 卡状态和识别信息。
CaiExtraProperty	表示设备自定义属性。
CaiInstancePropertiesResponseItem	单台设备的属性查询结果。
CaiApp	表示设备的应用程序信息。
CaiListUserAppsResponseItem	单台设备的第三方应用查询结果。
CaiListAllAppsResponseItem	单台设备的所有应用程序查询结果。
CaiDescribeKeepFrontAppStatusResponseItem	单台设备的前台保持状态。
CaiDescribeCameraMediaPlayStatusResponseItem	单台设备的摄像头媒体播放状态。
CaiDescribeKeepAliveListResponseItem	单台设备的后台保活应用列表。
CaiMediaSearchResponseItem	单台设备的媒体搜索结果。
CaiDescribeAppInstallBlackListResponseItem	单台设备的应用安装黑名单。
CaiGetNavVisibleStatusResponseItem	单台设备的导航栏显示状态。
CaiGetSystemMusicVolumeResponseItem	单台设备的系统媒体音量大小。
CaiUploadFileItem	表示上传文件的数据结构。
CaiUploadMediaFileItem	表示上传媒体文件的数据结构。
CaiDescribeInstancePropertiesResponse	多台设备的属性查询结果。
CaiListUserAppsResponse	多台设备的第三方应用查询结果。

版权所有：腾讯云计算（北京）有限责任公司

CaiDescribeCameraMediaPlayStatusResponse	多台设备的摄像头媒体播放状态。
CaiDescribeKeepFrontAppStatusResponse	多台设备的前台保持状态。
CaiDescribeKeepAliveListResponse	多台设备的后台保活应用列表。
CaiListAllAppsResponse	多台设备的所有应用程序查询结果。
CaiDescribeAppInstallBlackListResponse	多台设备的应用安装黑名单。
CaiGetNavVisibleStatusResponse	多台设备的导航栏显示状态。
CaiGetSystemMusicVolumeResponse	多台设备的系统媒体音量大小。
CaiBatchTaskCompletion	批量任务基础回调类型。
CaiUploadFilesCompletion	上传文件回调类型。
CaiUploadMediaFilesCompletion	上传媒体文件回调类型。
CaiDescribeInstancePropertiesCompletion	属性查询回调类型。
CaiListUserAppsCompletion	用户应用查询回调类型。
CaiDescribeCameraMediaPlayStatusCompletion	摄像头媒体状态回调类型。
CaiDescribeKeepFrontAppStatusCompletion	应用前台状态回调类型。
CaiDescribeKeepAliveListCompletion	保活列表查询回调类型。
CaiMediaSearchCompletion	媒体搜索回调类型。
CaiListAllAppsCompletion	所有应用查询回调类型。
CaiDescribeAppInstallBlackListCompletion	应用黑名单查询回调类型。
CaiGetNavVisibleStatusCompletion	获取当前系统导航栏显示状态回调类型。
CaiGetSystemMusicVolumeComplet	获取当前系统媒体音量大小回调类型。

版权所有：腾讯云计算（北京）有限责任公司

ion	

视频帧数据结构

TCRVideoFrame

```
@interface TCRVideoFrame : NSObject
@property (nonatomic, assign) int width;
@property (nonatomic, assign) int height;
@property (nonatomic, assign) int rotation;
@property (nonatomic, assign) int64_t timeStampNs;
@property (nonatomic, assign) CVPixelBufferRef pixelBuffer;
@end
```

变量

变量名称	类型	描述
width	int	视频帧的宽度。
height	int	视频帧的高度。
rotation	int	视频帧的旋转角度。
timeStampNs	int64_t	视频帧的时间戳，单位为纳秒。
pixelBuffer	CVPixelBufferRef	视频帧的像素缓冲区。

音频帧数据结构

TCRAudioFrame

```
@interface TCRAudioFrame : NSObject
@property (nonatomic, copy) NSData *data;
@property (nonatomic, assign) int bitsPerSample;
@property (nonatomic, assign) int sampleRate;
@property (nonatomic, assign) int channels;
@property (nonatomic, assign) int frames;
@end
```

变量

变量名称	类型	描述

data	NSData *	音频帧的数据。
bitsPerSample	int	每个音频样本的位数。
sampleRate	int	音频的采样率。
channels	int	音频数据中的通道数量。
frames	int	音频数据中的样本帧数。

单设备操作响应结果

CaiDeviceResponse

```
@interface CaiDeviceResponse : NSObject
@property (nonatomic, assign) NSInteger code;
@property (nonatomic, copy, nullable) NSString *msg;
@end
```

变量

变量名称	类型	描述
code	NSInteger	单个设备操作状态码。
msg	NSString *	单个设备状态消息。

批量操作响应结果

CaiBatchTaskResponse

```
@interface CaiBatchTaskResponse<__covariant DeviceResponseType:
CaiDeviceResponse *> : NSObject
@property (nonatomic, assign) NSInteger code;
@property (nonatomic, copy, nullable) NSString *message;
@property (nonatomic, strong, nullable) NSDictionary<NSString *,
DeviceResponseType> *deviceResponses;
@end
```

变量

变量名称	类型	描述
code	NSInteger	整体操作状态码。

版权所有：腾讯云计算（北京）有限责任公司

message	NSString *	整体状态消息。
deviceResponses	NSDictionary<NSString *, DeviceResponseType> *	各设备响应字典。

设备品牌和型号信息

CaiDeviceInfo

```
@interface CaiDeviceInfo : NSObject
@property (nonatomic, copy, nullable) NSString *brand;
@property (nonatomic, copy, nullable) NSString *model;
@end
```

变量

变量名称	类型	描述
brand	NSString *	设备品牌。
model	NSString *	设备型号。

设备地理位置

CaiGPSInfo

```
@interface CaiGPSInfo : NSObject
@property (nonatomic, assign) double longitude;
@property (nonatomic, assign) double latitude;
@end
```

变量

变量名称	类型	描述
longitude	double	经度值。
latitude	double	纬度值。

设备语言设置

CaiLanguageInfo

```
@interface CaiLanguageInfo : NSObject
@property (nonatomic, copy, nullable) NSString *language;
@property (nonatomic, copy, nullable) NSString *country;
@end
```

变量

变量名称	类型	描述
language	NSString *	语言代码。
country	NSString *	国家/地区代码。

设备地区设置

CaiLocaleInfo

```
@interface CaiLocaleInfo : NSObject
@property (nonatomic, copy, nullable) NSString *timezone;
@end
```

变量

变量名称	类型	描述
timezone	NSString *	时区设置。

设备网络代理配置

CaiProxyInfo

```
@interface CaiProxyInfo : NSObject
@property (nonatomic, assign) BOOL enabled;
@property (nonatomic, copy, nullable) NSString *protocol;
@property (nonatomic, copy, nullable) NSString *host;
@property (nonatomic, assign) NSInteger port;
@property (nonatomic, copy, nullable) NSString *user;
@property (nonatomic, copy, nullable) NSString *password;
@end
```

变量

变量名称	类型	描述
enabled	BOOL	是否启用代理。
protocol	NSString *	代理协议。
host	NSString *	代理服务器主机地址。
port	NSInteger	代理服务器端口。
user	NSString *	代理认证用户名。
password	NSString *	代理认证密码。

设备 SIM 卡状态和识别信息

CaiSIMInfo

```
@interface CaiSIMInfo : NSObject
@property (nonatomic, assign) NSInteger state;
@property (nonatomic, copy, nullable) NSString *phoneNumber;
@property (nonatomic, copy, nullable) NSString *IMSI;
@property (nonatomic, copy, nullable) NSString *ICCID;
@end
```

变量

变量名称	类型	描述
state	NSInteger	SIM 卡状态，0为未激活，1为已激活。
phoneNumber	NSString *	手机号码。
IMSI	NSString *	国际移动用户识别码。
ICCID	NSString *	集成电路卡识别码。

设备自定义属性

CaiExtraProperty

```
@interface CaiExtraProperty : NSObject
@property (nonatomic, copy, nullable) NSString *key;
@property (nonatomic, copy, nullable) NSString *value;
```

版权所有：腾讯云计算（北京）有限责任公司

```
@end
```

变量

变量名称	类型	描述
key	NSString *	属性键名。
value	NSString *	属性值。

单台设备的属性查询结果

CaiInstancePropertiesResponseItem

```
@interface CaiInstancePropertiesResponseItem : CaiDeviceResponse
@property (nonatomic, copy, nullable) NSString *requestId;
@property (nonatomic, strong, nullable) CaiDeviceInfo *deviceInfo;
@property (nonatomic, strong, nullable) CaiGPSInfo *GPSInfo;
@property (nonatomic, strong, nullable) CaiLanguageInfo *languageInfo;
@property (nonatomic, strong, nullable) CaiLocaleInfo *localeInfo;
@property (nonatomic, strong, nullable) CaiProxyInfo *proxyInfo;
@property (nonatomic, strong, nullable) CaiSIMInfo *SIMInfo;
@end
```

变量

变量名称	类型	描述
requestId	NSString *	请求 ID。
deviceInfo	CaiDeviceInfo *	设备信息。
GPSInfo	CaiGPSInfo *	GPS 信息。
languageInfo	CaiLanguageInfo *	语言信息。
localeInfo	CaiLocaleInfo *	地区信息。
proxyInfo	CaiProxyInfo *	代理信息。
SIMInfo	CaiSIMInfo *	SIM 卡信息。

设备的应用程序信息

CaiApp

```
@interface CaiApp : NSObject
@property (nonatomic, assign) long long firstInstallTimeMs;
@property (nonatomic, copy, nullable) NSString *label;
@property (nonatomic, assign) long long lastUpdateTimeMs;
@property (nonatomic, copy, nullable) NSString *packageName;
@property (nonatomic, copy, nullable) NSString *versionName;
@end
```

变量

变量名称	类型	描述
firstInstallTimeMs	long long	首次安装时间，Unix 时间戳 (毫秒)。
label	NSString *	应用显示名称。
lastUpdateTimeMs	long long	最后更新时间，Unix 时间戳 (毫秒)。
packageName	NSString *	应用包名。
versionName	NSString *	应用版本名称。

单台设备的第三方应用查询结果

CaiListUserAppsResponseItem

```
@interface CaiListUserAppsResponseItem : CaiDeviceResponse
@property (nonatomic, strong, nullable) NSArray<CaiApp *> *appList;
@end
```

变量

变量名称	类型	描述
appList	NSArray<CaiApp *> *	应用列表，设备上安装的第三方应用程序列表。

单台设备的所有应用程序查询结果

CaiListAllAppsResponseItem

```
@interface CaiListAllAppsResponseItem : CaiDeviceResponse
@property (nonatomic, strong, nullable) NSArray<CaiApp *> *appList;
@end
```

变量

变量名称	类型	描述
appList	NSArray<CaiApp *> *	所有应用列表，包括系统应用和第三方应用。

单台设备的前台保持状态

CaiDescribeKeepFrontAppStatusResponseItem

```
@interface CaiDescribeKeepFrontAppStatusResponseItem : CaiDeviceResponse
@property (nonatomic, strong, nullable) NSString *packageName;
@property (nonatomic, assign) BOOL enable;
@property (nonatomic, assign) int restartIntervalSeconds;
@end
```

变量

变量名称	类型	描述
packageName	NSString *	应用包名。
enable	BOOL	应用前台保持。
restartIntervalSeconds	int	重启间隔时长。

单台设备的摄像头媒体播放状态

CaiDescribeCameraMediaPlayStatusResponseItem

```
@interface CaiDescribeCameraMediaPlayStatusResponseItem :
CaiDeviceResponse
@property (nonatomic, copy, nullable) NSString *filePath;
@property (nonatomic, assign) NSInteger loops;
@end
```

变量

变量名称	类型	描述
filePath	NSString *	媒体文件路径。
loops	NSInteger	循环次数，负数表示无限循环。

单台设备的后台保活应用列表

CaiDescribeKeepAliveListResponseItem

```
@interface CaiDescribeKeepAliveListResponseItem : CaiDeviceResponse
@property (nonatomic, strong, nullable) NSArray<NSString *> *appList;
@end
```

变量

变量名称	类型	描述
appList	NSArray<NSString *> *	保活应用列表，应用包名的数组。

设备的媒体搜索项

CaiMediaItem

```
@interface CaiMediaItem : NSObject
@property (nonatomic, copy, nullable) NSString *fileName;
@property (nonatomic, copy, nullable) NSString *filePath;
@property (nonatomic, assign) long long fileBytes;
@property (nonatomic, assign) long long fileModifiedTime;
@end
```

变量

变量名称	类型	描述
fileName	NSString *	文件名，包含扩展名。
filePath	NSString *	完整文件路径。
fileBytes	long long	文件大小，以字节为单位。
fileModifiedTime	long long	文件修改时间，Unix 时间戳 (毫秒)。

单台设备的媒体搜索结果

CaiMediaSearchResponseItem

```
@interface CaiMediaSearchResponseItem : CaiDeviceResponse
@property (nonatomic, strong, nullable) NSArray<CaiMediaItem *>
*mediaList;
```

```
@end
```

变量

变量名称	类型	描述
mediaList	NSArray<CaiMediaItem *> *	媒体文件列表。

批量媒体搜索响应结果

CaiMediaSearchResponse

```
@interface CaiMediaSearchResponse :
CaiBatchTaskResponse<CaiMediaSearchResponseItem *>
@end
```

变量

CaiMediaSearchResponse 是一个批量任务响应基类，具体变量在父类 CaiBatchTaskResponse 中定义。

单台设备的应用安装黑名单

CaiDescribeAppInstallBlackListResponseItem

```
@interface CaiDescribeAppInstallBlackListResponseItem :
CaiDeviceResponse
@property (nonatomic, strong, nullable) NSArray<NSString *> *appList;
@end
```

变量

变量名称	类型	描述
appList	NSArray<NSString *> *	黑名单应用列表，禁止安装的应用包名数组。

单台设备的导航栏显示状态

CaiGetNavVisibleStatusResponseItem

```
@interface CaiGetNavVisibleStatusResponseItem : CaiDeviceResponse
@property (nonatomic, assign) BOOL visible;
@end
```

变量

变量名称	类型	描述
visible	BOOL	导航栏是否显示。

单台设备的系统媒体音量大小

CaiGetSystemMusicVolumeResponseItem

```
@interface CaiGetSystemMusicVolumeResponseItem : CaiDeviceResponse
@property (nonatomic, assign) NSInteger volume;
@end
```

变量

变量名称	类型	描述
volume	NSInteger	当前系统媒体音量大小（0-100）。

上传文件的数据结构

CaiUploadFileItem

```
@interface CaiUploadFileItem : NSObject
@property (nonatomic, strong, nullable) NSData *fileData;
@property (nonatomic, copy, nullable) NSString *fileName;
@property (nonatomic, copy, nullable) NSString *filePath; // 可选路径
@end
```

变量

变量名称	类型	描述
fileData	NSData *	要上传的文件数据。
fileName	NSString *	文件名。
filePath	NSString *	可选的文件路径。

上传媒体文件的数据结构

CaiUploadMediaFileItem

```
@interface CaiUploadMediaFileItem : NSObject
@property (nonatomic, strong, nullable) NSData *fileData;
@property (nonatomic, copy, nullable) NSString *fileName;
@end
```

变量

变量名称	类型	描述
fileData	NSData *	要上传的媒体文件数据。
fileName	NSString *	媒体文件名。

批量属性查询响应

CaiDescribeInstancePropertiesResponse

```
@interface CaiDescribeInstancePropertiesResponse :
CaiBatchTaskResponse<CaiInstancePropertiesResponseItem *>@end
```

批量用户应用查询响应

CaiListUserAppsResponse

```
@interface CaiListUserAppsResponse :
CaiBatchTaskResponse<CaiListUserAppsResponseItem *>@end
```

批量摄像头媒体状态响应

CaiDescribeCameraMediaPlayStatusResponse

```
@interface CaiDescribeCameraMediaPlayStatusResponse :
CaiBatchTaskResponse<CaiDescribeCameraMediaPlayStatusResponseItem *>@end
```

批量前台保持响应

CaiDescribeKeepFrontAppStatusResponse

```
@interface CaiDescribeKeepFrontAppStatusResponse :
CaiBatchTaskResponse<CaiDescribeKeepFrontAppStatusResponseItem *>@end
```

版权所有：腾讯云计算（北京）有限责任公司

批量保活列表响应

CaiDescribeKeepAliveListResponse

```
@interface CaiDescribeKeepAliveListResponse :
CaiBatchTaskResponse<CaiDescribeKeepAliveListResponseItem *>@end
```

批量所有应用查询响应

CaiListAllAppsResponse

```
@interface CaiListAllAppsResponse :
CaiBatchTaskResponse<CaiListAllAppsResponseItem *>@end
```

批量应用黑名单响应

CaiDescribeAppInstallBlackListResponse

```
@interface CaiDescribeAppInstallBlackListResponse :
CaiBatchTaskResponse<CaiDescribeAppInstallBlackListResponseItem *>@end
```

导航栏显示状态响应

CaiGetNavVisibleStatusResponse

```
@interface CaiGetNavVisibleStatusResponse :
CaiBatchTaskResponse<CaiGetNavVisibleStatusResponseItem *>@end
```

系统媒体音量响应

CaiGetSystemMusicVolumeResponse

```
@interface CaiGetSystemMusicVolumeResponse :
CaiBatchTaskResponse<CaiGetSystemMusicVolumeResponseItem *>@end
```

上传文件响应

CaiUploadResponse

```
@interface CaiUploadResponse : NSObject
@property (nonatomic, assign) NSInteger code;
```

```
@property (nonatomic, copy, nullable) NSString *msg;
@property (nonatomic, strong, nullable) NSArray<NSDictionary *>
*fileStatus;
@end
```

变量

变量名称	类型	描述
code	NSInteger	响应码。
msg	NSString *	响应消息。
fileStatus	NSArray<NSDictionary *> *	文件状态。

上传媒体文件响应

CaiUploadMediaFilesResponse

```
@interface CaiUploadMediaFilesResponse : NSObject
@property (nonatomic, assign) NSInteger code;
@property (nonatomic, copy, nullable) NSString *msg;
@end
```

变量

变量名称	类型	描述
code	NSInteger	响应码。
msg	NSString *	响应消息。

批量任务基础回调类型

CaiBatchTaskCompletion

```
typedef void (^CaiBatchTaskCompletion)(CaiBatchTaskResponse *_Nonnull
response, NSError * _Nullable error);
```

变量

变量名称	类型	描述

response	CaiBatchTaskResponse *	批量任务响应结构。
error	NSError *	错误对象（系统级错误）。

上传文件回调类型

CaiUploadFilesCompletion

```
typedef void (^CaiUploadFilesCompletion)(CaiUploadResponse *_Nonnull
response, NSError *_Nullable error);
```

变量

变量名称	类型	描述
response	CaiUploadResponse *	上传文件响应。
error	NSError *	错误对象（系统级错误）。

上传媒体文件回调类型

CaiUploadMediaFilesCompletion

```
typedef void (^CaiUploadMediaFilesCompletion)
(CaiUploadMediaFilesResponse *_Nonnull response, NSError *_Nullable
error);
```

变量

变量名称	类型	描述
response	CaiUploadMediaFilesResponse *	上传媒体文件响应。
error	NSError *	错误对象（系统级错误）。

属性查询回调类型

CaiDescribeInstancePropertiesCompletion

```
typedef void (^CaiDescribeInstancePropertiesCompletion)
(CaiDescribeInstancePropertiesResponse *_Nonnull response, NSError
*_Nullable error);
```

变量

变量名称	类型	描述
response	CaiDescribeInstancePropertiesResponse *	属性查询响应。
error	NSError *	错误对象（系统级错误）。

用户应用查询回调类型

CaiListUserAppsCompletion

```
typedef void (^CaiListUserAppsCompletion)(CaiListUserAppsResponse
*_Nonnull response, NSError *_Nullable error);
```

变量

变量名称	类型	描述
response	CaiListUserAppsResponse *	用户应用查询响应。
error	NSError *	错误对象（系统级错误）。

摄像头媒体状态回调类型

CaiDescribeCameraMediaPlayStatusCompletion

```
typedef void (^CaiDescribeCameraMediaPlayStatusCompletion)
(CaiDescribeCameraMediaPlayStatusResponse *_Nonnull response, NSError
*_Nullable error);
```

变量

变量名称	类型	描述
response	CaiDescribeCameraMediaPlayStatusResponse *	摄像头媒体状态响应。
error	NSError *	错误对象（系统级错误）。

应用前台状态回调类型

CaiDescribeKeepFrontAppStatusCompletion

```
typedef void (^CaiDescribeKeepFrontAppStatusCompletion)
(CaiDescribeKeepFrontAppStatusResponse *_Nonnull response, NSError
*_Nullable error);
```

变量

变量名称	类型	描述
response	CaiDescribeKeepFrontAppStatusResponse *	应用前台状态响应。
error	NSError *	错误对象（系统级错误）。

保活列表查询回调类型

CaiDescribeKeepAliveListCompletion

```
typedef void (^CaiDescribeKeepAliveListCompletion)
(CaiDescribeKeepAliveListResponse *_Nonnull response, NSError *_Nullable
error);
```

变量

变量名称	类型	描述
response	CaiDescribeKeepAliveListResponse *	保活列表响应。
error	NSError *	错误对象（系统级错误）。

媒体搜索回调类型

CaiMediaSearchCompletion

```
typedef void (^CaiMediaSearchCompletion)(CaiMediaSearchResponse
*_Nonnull response, NSError *_Nullable error);
```

变量

变量名称	类型	描述
response	CaiMediaSearchResponse *	媒体搜索响应。
error	NSError *	错误对象（系统级错误）。

所有应用查询回调类型

CaiListAllAppsCompletion

```
typedef void (^CaiListAllAppsCompletion)(CaiListAllAppsResponse
*_Nonnull response, NSError *_Nullable error);
```

变量

变量名称	类型	描述
response	CaiListAllAppsResponse *	所有应用查询响应。
error	NSError *	错误对象（系统级错误）。

应用黑名单查询回调类型

CaiDescribeAppInstallBlackListCompletion

```
typedef void (^CaiDescribeAppInstallBlackListCompletion)
(CaiDescribeAppInstallBlackListResponse *_Nonnull response, NSError
*_Nullable error);
```

变量

变量名称	类型	描述
response	CaiDescribeAppInstallBlackListResponse *	应用黑名单响应。
error	NSError *	错误对象（系统级错误）。

当前系统导航栏显示状态回调类型

CaiGetNavVisibleStatusCompletion

```
typedef void (^CaiGetNavVisibleStatusCompletion)
(CaiGetNavVisibleStatusResponse *_Nonnull response, NSError *_Nullable
error);
```

变量

变量名称	类型	描述

response	CaiGetNavVisibleStatusResponse *	获取当前系统导航栏显示状态响应。
error	NSError *	错误对象（系统级错误）。

当前系统媒体音量大小回调类型

CaiGetSystemMusicVolumeCompletion

```
typedef void (^CaiGetSystemMusicVolumeCompletion)
(CaiGetSystemMusicVolumeResponse *_Nonnull response, NSError *_Nullable
error);
```

变量

变量名称	类型	描述
response	CaiGetSystemMusicVolumeResponse *	获取当前系统媒体音量大小响应。
error	NSError *	错误对象（系统级错误）

事件回调

最近更新时间：2025-09-03 14:10:22

事件回调概览

TcrEvent 事件

下列事件通过 TcrSessionObserver 回调：

事件名称	描述
STATE_INITED	表示会话已初始化。关联数据为 NSString (client Session)。
STATE_CONNECTED	表示会话已连接。无关联数据。
STATE_RECONNECTING	表示会话正在重连。无关联数据。
STATE_CLOSED	表示会话已关闭，无法再使用。关联数据为 NSInteger 类型。
CLIENT_STATS	表示性能数据已更新。关联数据为 NSDictionary 类型。
MIC_STATUS_CHANGED	表示服务器的麦克风状态已改变。关联数据为 NSDictionary 类型，表示是否需要开启麦克风。
CAMERA_STATUS_CHANGED	表示服务器的摄像头状态已改变。关联数据为 NSDictionary 类型，包含分辨率宽度、高度和摄像头状态。
IME_TYPE	表示连接到云设备时分配的输入法类型。关联数据为 NSDictionary 类型，包含服务器端的 IME 状态。
INPUT_STATUS_CHANGED	表示服务器的输入状态已改变。关联数据为 NSDictionary 类型，包含状态信息。
SCREEN_CONFIG_CHANGE	表示云手机屏幕配置已改变。关联数据为 NSDictionary 类型，包含屏幕方向、角度等信息。
VIDEO_STREAM_CONFIG_CHANGED	表示云手机视频流配置已更新。关联数据为 NSDictionary 类型，包含屏幕宽度、高度等信息。
CLIENT_IDLE	表示用户处于空闲状态。无关联数据。
CLIENT_LOW_FPS	表示帧率连续10秒保持低水平。无关联数据。
INPUT_STATE_CHANGE	表示远程输入框状态已改变。关联数据为 NSDictionary 类型，包含输入框类型、文本内容等信息。

CAI_TRANS_MESSAGE	云端应用向客户端发送的透传消息事件。关联数据为 NSDictionary 类型，包含包名和消息内容。
CAI_SYSTEM_USAGE	云端应用向客户端发送的负载信息。关联数据为 NSDictionary 类型，包含 CPU、内存和 GPU 使用率。
CAI_CLIPBOARD	云端剪贴板内容同步事件。关联数据为 NSDictionary 类型，包含剪贴板文本内容。
CAI_NOTIFICATION	云端通知消息下发事件。关联数据为 NSDictionary 类型，包含应用包名、通知标题和正文。
CAI_IMAGE_EVENT	定期回调云端实例的画面截图。关联数据为 NSDictionary 类型，包含实例 ID 和截图 URL。
CAI_SYSTEM_STATUS	每1秒云端应用通过 WebRTC 向客户端发送的系统状态。关联数据为 NSDictionary 类型，包含导航栏可见性和媒体音量。

事件详细说明

初始化成功事件

当 SDK 初始化成功时触发 STATE_INITED 事件。

返回值

`NSString*`：客户端会话标识符。

监听示例

```
- (void)onEvent:(TcrEvent)eventType eventData:(id _Nullable)eventData {
    if (eventType == STATE_INITED) {
        NSString *clientSession = (NSString *)eventData;
        // 处理初始化成功事件，使用clientSession初始化TcrSession
    }
}
```

会话连接事件

当会话成功连接时触发 STATE_CONNECTED 事件。

返回值

无。

监听示例

```
- (void)onEvent:(TcrEvent)eventType eventData:(id _Nullable)eventData {
    if (eventType == STATE_CONNECTED) {
        // 处理会话连接事件，例如更新UI状态
    }
}
```

会话重连中事件

当会话正在重连时触发 STATE_RECONNECTING 事件，可用于提示用户。

返回值

无。

监听示例

```
- (void)onEvent:(TcrEvent)eventType eventData:(id _Nullable)eventData {
    if (eventType == STATE_RECONNECTING) {
        // 处理重连中事件，例如显示加载提示
    }
}
```

会话关闭事件

当会话关闭且无法再使用时触发 STATE_CLOSED 事件。

返回值

`NSInteger`：关闭代码。

监听示例

```
- (void)onEvent:(TcrEvent)eventType eventData:(id _Nullable)eventData {
    if (eventType == STATE_CLOSED) {
        NSInteger closeCode = [(NSNumber *)eventData integerValue];
        // 处理会话关闭事件，例如记录错误码
    }
}
```

性能数据更新事件

当客户端性能数据更新时触发 CLIENT_STATS 事件。

返回值

`NSDictionary`：性能数据字典。

监听示例

```
- (void)onEvent:(TcrEvent)eventType eventData:(id _Nullable)eventData {
    if (eventType == CLIENT_STATS) {
        NSDictionary *stats = eventData;
        // 处理性能数据事件，例如分析性能指标
    }
}
```

服务器麦克风状态改变事件

当服务器麦克风状态（如打开或关闭）改变时触发 MIC_STATUS_CHANGED 事件。

返回值

`NSDictionary`：

键	值类型	描述
status	NSString	麦克风状态：`open`（打开）、`close`（关闭）。

监听示例

```
- (void)onEvent:(TcrEvent)eventType eventData:(id _Nullable)eventData {
    if (eventType == MIC_STATUS_CHANGED) {
        NSDictionary *micStatus = eventData;
        // 处理麦克风状态事件，例如打开/关闭音频上行
    }
}
```

服务器摄像头状态改变事件

当服务器摄像头状态（如打开或关闭）改变时触发 CAMERA_STATUS_CHANGED 事件。

返回值

`NSDictionary`：

键	值类型	描述

width	NSNumber	摄像头分辨率宽度。
height	NSNumber	摄像头分辨率高度。
status	NSString	摄像头状态：`open_back`（打开后置摄像头）、`open_front`（打开前置摄像头）、`close`（关闭）。

监听示例

```
- (void)onEvent:(TcrEvent)eventType eventData:(id _Nullable)eventData {
    if (eventType == CAMERA_STATUS_CHANGED) {
        NSDictionary *cameraStatus = eventData;
        // 处理摄像头状态事件，例如打开/关闭视频上行
    }
}
```

输入法类型事件

当连接到云设备时，输入法类型（云端或客户端本地）确定时触发 IME_TYPE 事件。

返回值

`NSDictionary`：

键	值类型	描述
ime_type	NSString	输入法类型：`cloud`（使用云端输入法）、`local`（使用客户端本地输入法）。

监听示例

```
- (void)onEvent:(TcrEvent)eventType eventData:(id _Nullable)eventData {
    if (eventType == IME_TYPE) {
        NSDictionary *imeInfo = eventData;
        // 处理输入法类型事件，例如切换输入法UI
    }
}
```

输入状态改变事件

当服务器输入状态改变时触发 INPUT_STATUS_CHANGED 事件。

返回值

`NSDictionary`：

键	值类型	描述
status	NSString	输入状态：`disabled`（输入被禁用）。

监听示例

```
- (void)onEvent:(TcrEvent)eventType eventData:(id _Nullable)eventData {
    if (eventType == INPUT_STATUS_CHANGED) {
        NSDictionary *inputStatus = eventData;
        // 处理输入状态事件，例如禁用用户输入控件
    }
}
```

云手机屏幕配置改变事件

当云手机屏幕配置（如方向或尺寸）改变时触发 SCREEN_CONFIG_CHANGE 事件。

返回值

`NSDictionary`：

键	值类型	描述
orientation	NSString	屏幕方向：`landscape`（横屏）、`portrait`（竖屏）。
degree	NSString	旋转角度：`0_degree`、`90_degree`、`180_degree`、`270_degree`。
width	NSNumber	屏幕宽度。
height	NSNumber	屏幕高度。

监听示例

```
- (void)onEvent:(TcrEvent)eventType eventData:(id _Nullable)eventData {
    if (eventType == SCREEN_CONFIG_CHANGE) {
```

```
        NSDictionary *screenConfig = eventData;
        // 处理屏幕配置事件，例如调整渲染尺寸
    }
}
```

云手机视频流配置改变事件

当云手机视频流配置（如分辨率）更新时触发 VIDEO_STREAM_CONFIG_CHANGED 事件。

返回值

`NSDictionary`：

键	值类型	描述
width	NSNumber	视频流宽度。
height	NSNumber	视频流高度。

监听示例

```
- (void)onEvent:(TcrEvent)eventType eventData:(id _Nullable)eventData {
    if (eventType == VIDEO_STREAM_CONFIG_CHANGED) {
        NSDictionary *videoConfig = eventData;
        // 处理视频流配置事件，例如更新解码器
    }
}
```

客户端空闲事件

当用户长时间未操作鼠标、键盘或屏幕时触发 CLIENT_IDLE 事件。

返回值

无关联数据。

监听示例

```
- (void)onEvent:(TcrEvent)eventType eventData:(id _Nullable)eventData {
    if (eventType == CLIENT_IDLE) {
        // 处理空闲事件，例如显示节能提示
    }
}
```

客户端低帧率事件

当帧率连续10秒保持低位时触发 CLIENT_LOW_FPS 事件。

返回值

无关联数据。

监听示例

```
- (void)onEvent:(TcrEvent)eventType eventData:(id _Nullable)eventData {
    if (eventType == CLIENT_LOW_FPS) {
        // 处理低帧率事件，例如优化渲染性能
    }
}
```

云端输入框状态改变事件

当云端输入框状态（如焦点）改变时触发 INPUT_STATE_CHANGE 事件。

返回值

`NSDictionary`：

键	值类型	描述
field_type	NSString	输入框类型：`normal_input`（常规窗口焦点）、`autologin_input`（已弃用）、`unfocused`（焦点移出窗口）。
text	NSString	输入框文本内容（仅 Android 应用支持）。
status	NSString	输入状态（仅 Windows 系统）：`disable`（禁用输入）、`enable`（启用输入）。
input_type	NSNumber	文本内容类型位定义（整数）。

监听示例

```
- (void)onEvent:(TcrEvent)eventType eventData:(id _Nullable)eventData {
    if (eventType == INPUT_STATE_CHANGE) {
        NSDictionary *inputState = eventData;
        // 处理输入框状态事件，例如同步文本内容
    }
```

```
}
```

云端透传消息事件

当云端应用向客户端发送透传消息时触发 CAI_TRANS_MESSAGE 事件。

返回值

`NSDictionary`：

键	值类型	描述
package_name	NSString	目标应用的包名（Android 应用标识）。
msg	NSString	透传的原始消息内容（字符串或 JSON 对象）。

监听示例

```
- (void)onEvent:(TcrEvent)eventType eventData:(id _Nullable)eventData {
    if (eventType == CAI_TRANS_MESSAGE) {
        NSDictionary *transMessage = eventData;
        // 处理透传消息事件
    }
}
```

云端系统负载信息事件

当云端应用向客户端发送系统负载信息（如 CPU、内存、GPU 使用率）时触发 CAI_SYSTEM_USAGE 事件。

返回值

`NSDictionary`：

键	值类型	描述
cpu_usage	NSNumber	CPU 使用率（百分比，浮点数）。
mem_usage	NSNumber	内存使用率（百分比，浮点数）。
gpu_usage	NSNumber	GPU 使用率（百分比，浮点数）。

监听示例

```
- (void)onEvent:(TcrEvent)eventType eventData:(id _Nullable)eventData {
```

```
    if (eventType == CAI_SYSTEM_USAGE) {
        NSDictionary *systemUsage = eventData;
        // 处理系统负载信息事件
    }
}
```

云端剪贴板同步事件

当云端设备的剪贴板内容发生变化时触发 CAI_CLIPBOARD 事件，将最新文本同步至客户端。

返回值

`NSDictionary`：

键	值类型	描述
text	NSString	剪贴板的最新文本内容。

监听示例

```
- (void)onEvent:(TcrEvent)eventType eventData:(id _Nullable)eventData {
    if (eventType == CAI_CLIPBOARD) {
        NSDictionary *clipboardData = eventData;
        NSString *text = clipboardData[@"text"];
        // 处理剪贴板同步事件
    }
}
```

云端通知消息事件

当云机上有新通知时，触发 CAI_NOTIFICATION 事件将通知消息下发至客户端。

返回值

`NSDictionary`：

键	值类型	描述
package_name	NSString	发送通知的应用包名。
title	NSString	通知标题。
text	NSString	通知正文。

监听示例

```
- (void)onEvent:(TcrEvent)eventType eventData:(id _Nullable)eventData {
    if (eventType == CAI_NOTIFICATION) {
        NSDictionary *notification = eventData;
        // 处理通知消息事件
    }
}
```

云端画面截图事件

CAI_IMAGE_EVENT 事件定期回调云端实例的画面截图 URL。

返回值

`NSDictionary`：键为实例 ID（NSString），值为截图 URL（NSString）。

键（实例 ID）	值类型	描述
instanceId1	NSString	实例1的截图 URL。
instanceId2	NSString	实例2的截图 URL。
...	...	...

监听示例

```
- (void)onEvent:(TcrEvent)eventType eventData:(id _Nullable)eventData {
    if (eventType == CAI_IMAGE_EVENT) {
        NSDictionary *imageEvents = eventData;
        // 遍历处理每个实例的截图URL
        [imageEvents enumerateKeysAndObjectsUsingBlock:^(NSString
*instanceId, NSString *screenshotUrl, BOOL *stop) {
            // 处理截图URL
        }];
    }
}
```

云端系统状态事件

CAI_SYSTEM_STATUS 事件回调云端应用每1秒通过 WebRTC 向客户端发送的系统状态信息。

返回值

`NSDictionary`：

键	值类型	描述
nav_visible	NSNumber (BOOL)	导航栏是否可见。
music_volume	NSNumber	系统媒体音量（范围0-100）。

监听示例

```
- (void)onEvent:(TcrEvent)eventType eventData:(id _Nullable)eventData {
    if (eventType == CAI_SYSTEM_STATUS) {
        NSDictionary *systemStatus = eventData;
        BOOL navVisible = [systemStatus[@"nav_visible"] boolValue];
        NSInteger musicVolume = [systemStatus[@"music_volume"]
integerValue];
        // 处理系统状态事件
    }
}
```

Android SDK
SDK 集成指引

最近更新时间：2025-09-03 14:10:22

此 SDK 提供在 Android 端快速接入云手机 Paas 平台的能力，可以通过 Android 端 SDK 方便地操作云手机实例。

运行环境

Android 4.4及以上系统版本。

快速接入

获取 SDK

SDK	下载地址	SDK 说明文档	版本发布日志
Android SDK	通过 Maven 集成	接口说明	Release notes

SDK 集成

在应用模块的 'build.gradle' 中引用：

```
implementation 'com.tencent.tcr:tcrsdk-full:3.26.1'
```

SDK 使用

```
// 请求云手机实例的访问信息AceessInfo和鉴权Token
private void requestAccessToken() {

AsyncCallback<ExpServerResponse<CreateAndroidInstancesAccessTokenRespons
e>> createAndroidInstancesAccessTokenCallback = new
AsyncCallback<ExpServerResponse<CreateAndroidInstancesAccessTokenRespons
e>>() {
        @Override
        public void
onSuccess(ExpServerResponse<CreateAndroidInstancesAccessTokenResponse>
expServerResponse) {
            if (expServerResponse.Error != null) {
                Log.e(TAG, "CreateAndroidInstancesAccessToken
expServerResponse.Error: " + expServerResponse.Error);
```

```
                return;
            }
            if (expServerResponse.Response == null) {
                Log.e(TAG, "CreateAndroidInstancesAccessToken Response:
null");
                return;
            }

            // 初始化 TcrSdk
            initTcrSdk(expServerResponse);
        }

        @Override
        public void onFailure(int code, String errorMsg) {
            Log.e(TAG, "CreateAndroidInstancesAccessToken failed: " +
code + ", " + errorMsg);
            Toast.makeText(FunctionActivity.this, "获取实例访问信息失败: " +
code + ", " + errorMsg, Toast.LENGTH_LONG).show();
            finish();
        }
    };
    ......
}

// 初始化TcrSdk
private void
initTcrSdk(ExpServerResponse<CreateAndroidInstancesAccessTokenResponse>
expServerResponse) {
    TcrSdk.TcrConfig config = new TcrSdk.TcrConfig();
    config.type = SdkType.CloudPhone;
    config.ctx = this;
    config.logger = new TcrLogger() {
        // 记录日志
    };
    config.callback = new AsyncCallback<Void>() {
        @Override
        public void onSuccess(Void result) {
            Log.i(TAG, "init TcrSdk success");
        }

        @Override
        public void onFailure(int code, String msg) {
```

```
            String errorMsg = "init TcrSdk failed:" + code + " msg:" +
msg;
            Log.e(TAG, "init TcrSdk fail. errorMsg: " + errorMsg);
        }
    };
    config.accessInfo = expServerResponse.Response.AccessInfo;
    config.token = expServerResponse.Response.Token;
    TcrSdk.getInstance().init(config);
}

// 创建Tcr会话对象
private void initTcrSession() {
    TcrSessionConfig tcrSessionConfig =
TcrSessionConfig.builder().observer(mSessionObserver).build();
    mTcrSession =
TcrSdk.getInstance().createTcrSession(tcrSessionConfig);
    if (mTcrSession == null) {
        Log.e(TAG, "mTcrSession = null");
        showToast("创建会话失败，请检查TcrSdk是否初始化成功",
Toast.LENGTH_SHORT);
    }
}

// 创建渲染视图
private void initTcrRenderView() {
    if (mTcrSession == null) {
        showToast("创建渲染视图必须有关联的会话", Toast.LENGTH_SHORT);
        return;
    }
    // 创建渲染视图
    mRenderView = TcrSdk.getInstance().createTcrRenderView(this,
mTcrSession, TcrRenderViewType.SURFACE);
    if (mRenderView == null) {
        Log.e(TAG, "mRenderView = null");
        showToast("创建渲染失败，请检查TcrSdk是否初始化成功",
Toast.LENGTH_SHORT);
        return;
    }
    // 将渲染视图添加到界面上
    ((FrameLayout)
findViewById(R.id.render_view_parent)).addView(mRenderView);
}
```

```
// Tcr会话的观察者，处理各类事件通知的消息和数据
private final Observer mSessionObserver = new Observer() {
    @Override
    public void onEvent(TcrSession.Event event, Object eventData) {
        switch (event) {
            case STATE_INITED:// 本地会话对象初始化完成
                // 可以开始连接指定的云手机。
                boolean ret = mTcrSession.access(mGroupInstanceIds,
mIsGroup);
                if (!ret) {
                    showToast("连接云手机失败，请重试", Toast.LENGTH_SHORT);
                    finish();
                }
                break;
            case STATE_CONNECTED:// 成功和指定的云手机建立连接
                // 设置云手机的主控、同步列表、请求主控视频流
                if (mIsGroup) {
                    String masterId = mGroupInstanceIds.get(0);

TcrSdk.getInstance().getAndroidInstance().setMaster(masterId);

TcrSdk.getInstance().getAndroidInstance().setSyncList(mGroupInstanceIds)
;

TcrSdk.getInstance().getAndroidInstance().requestStream(masterId,
"open", "low");
                }
                break;
            case STATE_RECONNECTING:
                showToast("重连中...", Toast.LENGTH_LONG);
                break;
            case STATE_CLOSED:
                showToast("会话关闭", Toast.LENGTH_SHORT);
                finish();
                break;
            default:
                break;
        }
    }
};
```

完整 Demo

完整接入代码可参考 TcrCloudPhoneDemo 。

接口说明

最近更新时间：2025-09-03 14:10:22

接口概览

顶层接口

通过 `TcrSdk` 对象调用的接口。

方法	描述
init	初始化 SDK，初始化成功后才能使用其他接口。
createTcrSession	创建会话对象 TcrSession，用于和云手机进行交互。
createTcrRenderView	创建渲染视图 TcrRenderView，用于显示云手机画面和视图交互。
setAccessToken	设置 accessToken，通过 云 API 获取 accessToken，可通过该接口设置也可以通过 init 参数传入。
getAndroidInstance	获取安卓实例对象 AndroidInstance，用于操作云手机安卓实例。

会话接口

通过 `TcrSession` 对象调用的接口。

生命周期相关接口

方法	描述
access	启动会话，在本地会话对象初始化成功后，调用此方法，以建立和指定 ID 的云手机的连接。在成功建立连接后，此会话对象才能与指定的云手机进行交互。
release	释放会话，客户端释放本地会话资源。云端会在自动检测到心跳消失后释放云端资源。
getRequestId	获取会话请求 ID。

音视频流控制接口

方法	描述

setEnableAudioPlaying	开关音频播放。
setEnableLocalAudio	开关音频上行。
setEnableLocalVideo	开关视频上行。
pauseStreaming	暂停音视频流下行。
resumeStreaming	恢复音视频流下行。
setRemoteAudioPlayProfile	设置下行的音频的参数。
setLocalVideoProfile	设置上行的视频的参数。
setVideoSink	设置下行视频的接收器。
setAudioSink	设置下行音频的接收器。

云端应用交互接口

方法	描述
restartCloudApp	重启云端应用进程。
pasteText	向云端应用输入框粘贴文本（需输入框已聚焦）。
setDisableCloudInput	禁用云端软键盘（仅支持移动端应用）。

云端外设交互接口

方法	描述
getTouchScreen	获取云端触摸屏交互对象 TouchScreen。

触摸屏交互接口

通过 `TouchScreen` 对象调用的接口。

方法	描述

touch	触发云端触屏触摸事件。

安卓实例操作接口

通过 `AndroidInstance` 对象调用的接口。

单实例操作接口

方法	描述
getInstanceImageAddress	获取实例截图地址。
getInstanceImage	获取实例截图。
uploadFile	上传文件到实例。
uploadMedia	上传媒体文件到实例。
getInstanceDownloadAddress	获取文件下载地址。
getInstanceDownloadLogcatAddress	获取日志文件下载地址。
downloadFile	下载文件。
switchIME	切换输入法。

多实例批量操作接口

方法	描述
setLocation	设置设备 GPS 信息，支持批量实例操作。
setResolution	设置设备分辨率，支持批量实例操作。
paste	粘贴文本，支持批量实例操作。
sendClipboard	发送文本到剪切板，支持批量实例操作。
shake	摇一摇，支持批量实例操作。
setSensor	设置设备传感器信息，支持批量实例操作。
sendTransMessage	发送 App binder 消息，支持批量实例操作。

版权所有：腾讯云计算（北京）有限责任公司

describeInstanceProperties	查询实例属性，支持批量实例操作。
modifyInstanceProperties	修改实例属性，支持批量实例操作。
listUserApps	查询已安装第三方应用，支持批量实例操作。
modifyKeepFrontAppStatus	修改前台应用保活状态，支持批量实例操作。
describeKeepFrontAppStatus	查询前台应用保活状态，支持批量实例操作。
unInstallByPackageName	卸载应用，支持批量实例操作。
startApp	启动应用，支持批量实例操作。
stopApp	停止应用，支持批量实例操作。
clearAppData	清除应用数据，支持批量实例操作。
enableApp	启用应用，支持批量实例操作。
disableApp	禁用应用，支持批量实例操作。
startCameraMediaPlay	摄像头播放媒体文件，支持批量实例操作。
stopCameraMediaPlay	摄像头停止播放媒体文件，支持批量实例操作。
describeCameraMediaPlayStatus	查询当前摄像头媒体播放状态，支持批量实例操作。
displayCameraImage	摄像头显示图片，支持批量实例操作。
addKeepAliveList	增加后台保活应用，支持批量实例操作。
removeKeepAliveList	移除后台保活应用，支持批量实例操作。
setKeepAliveList	覆盖后台保活应用，支持批量实例操作。
describeKeepAliveList	查询后台保活应用，支持批量实例操作。
clearKeepAliveList	清空后台保活应用，支持批量实例操作。
mute	静音开关，支持批量实例操作。
mediaSearch	搜索媒体库文件，支持批量实例操作。

版权所有：腾讯云计算（北京）有限责任公司

reboot	重启实例，支持批量实例操作。
listAllApps	查询所有应用列表，支持批量实例操作。
moveAppBackground	关闭应用至后台，支持批量实例操作。
addAppInstallBlackList	新增应用安装黑名单，支持批量实例操作。
removeAppInstallBlackList	移除应用安装黑名单，支持批量实例操作。
setAppInstallBlackList	覆盖应用安装黑名单，支持批量实例操作。
describeAppInstallBlackList	查询应用安装黑名单，支持批量实例操作。
clearAppInstallBlackList	清空应用安装黑名单，支持批量实例操作。
getNavVisibleStatus	获取系统导航栏显示状态，支持批量实例操作。
getSystemMusicVolume	获取系统媒体音量大小，支持批量实例操作。

接口详细说明

初始化 SDK

初始化（init）云手机 SDK，初始化成功后才能使用 SDK 的其他接口。

输入参数

TcrConfig

参数名称	必选	类型	描述
type	是	SdkType	SDK 类型，云手机请设置为 CloudPhone。
ctx	是	Context	当前应用的 Android Context。
logger	是	TcrLogger	用于接收 SDK 日志的对象，App 需要将收到的 SDK 日志存储到文件和上报，以便分析定位 SDK 运行时的问题。 如果 App 没有显示设置 logger，SDK 会将日志打印到系统 logcat 以及写入日志文件（/storage/emulated/0/Android/data/app package name/files/tcrlogs 目录下）。

			注意，当您反馈 SDK 问题时，您需要提供相应的日志。
callback	是	AsyncCallback	初始化的异步回调对象。必须在回调成功后才能使用 SDK 的其他接口。
accessInfo	否	Boolean	实例访问的 accessInfo，通过 云 API 获取。
token	否	Boolean	实例访问的 token，通过 云 API 获取。

示例

```
TcrSdk.TcrConfig config = new TcrSdk.TcrConfig();
config.type = SdkType.CloudPhone;
config.ctx = this;
config.logger = new TcrLogger() {
// 实现日志记录逻辑
};
config.callback = new AsyncCallback<Void>() {
……
};
config.accessInfo = ……;
config.token = ……;
TcrSdk.getInstance().init(config);
```

创建会话

创建会话（createTcrSession）对象，用于和云手机进行交互。
会话对象创建后，SDK 内部会异步执行会话的本地初始化逻辑，初始化结果会通知给 builder 传入的参数 TcrSession#Observer 对象的 onEvent() 方法 。成功的通知事件为 TcrSession.Event#STATE_INITED，失败的通知事件为 TcrSession.Event#STATE_CLOSED。使用者只有在收到会话初始化成功的事件之后，才能进一步调用 access() 方法与云手机进行建连。

输入参数

参见 TcrSessionConfig 。

返回值

创建的会话对象。如果为 null 则代表对象创建失败（通常是因 SDK 状态异常，例如未初始化成功）。

> ⚠ **注意：**

> 只有成功创建了会话对象，内部才会执行会话的本地初始化逻辑，才可能进一步收到 observer 的回调事件。

示例

```
// Tcr会话的观察者，处理各类事件通知的消息和数据
private final Observer mSessionObserver = new Observer() {
    @Override
    public void onEvent(TcrSession.Event event, Object eventData) {
        switch (event) {
            case STATE_INITED:// 本地会话对象初始化完成
                // 可以开始连接指定的云手机。
                boolean ret = mTcrSession.access(mGroupInstanceIds,
mIsGroup);
                if (!ret) {
                    showToast("连接云手机失败，请重试", Toast.LENGTH_SHORT);
                    finish();
                }
                break;
            case STATE_CONNECTED:// 成功和指定的云手机建立连接
                // 设置云手机的主控、同步列表、请求主控视频流
                if (mIsGroup) {
                    String masterId = mGroupInstanceIds.get(0);

TcrSdk.getInstance().getAndroidInstance().setMaster(masterId);

TcrSdk.getInstance().getAndroidInstance().setSyncList(mGroupInstanceIds)
;

TcrSdk.getInstance().getAndroidInstance().requestStream(masterId,
"open", "low");
                }
                break;
            case STATE_CLOSED:
                showToast("会话关闭", Toast.LENGTH_SHORT);
                finish();
                break;
        }
    }
};

// 创建Tcr会话对象
private void initTcrSession() {
```

```
    TcrSessionConfig tcrSessionConfig =
TcrSessionConfig.builder().observer(mSessionObserver).build();
    mTcrSession =
TcrSdk.getInstance().createTcrSession(tcrSessionConfig);
    if (mTcrSession == null) {
        Log.e(TAG, "mTcrSession = null");
        showToast("创建会话失败，请检查TcrSdk是否初始化成功",
Toast.LENGTH_SHORT);
    }
}
```

创建渲染视图

创建渲染视图（createTcrRenderView）对象，用于 UI 上显示云手机画面和视图交互。

输入参数

参数名称	必选	类型	描述
context	是	Context	视图的 Android Context。
session	是	TcrSession	视图关联的会话对象。
type	是	TcrRenderViewType	视图类型。

示例

```
// 创建渲染视图
mRenderView =
TcrSdk.getInstance().createTcrRenderView(PlayActivity.this, mTcrSession,
TcrRenderViewType.SURFACE);
if (mRenderView == null) {
    Log.e(TAG, "createTcrRenderView 失败");
    return;
}
// 将渲染视图添加到界面上
((FrameLayout)
findViewById(R.id.render_view_parent)).addView(mRenderView);
```

设置 AccessToken

设置 accessToken（setAccessToken）。

输入参数

参数名称	必选	类型	描述
accessInfo	是	String	通过 云 API 获取的访问信息。
token	是	String	通过 云 API 获取的令牌。

获取安卓实例

获取安卓实例对象（getAndroidInstance），用于操作云手机安卓实例。

```
AndroidInstance androidInstance =
TcrSdk.getInstance().getAndroidInstance();
```

启动会话

启动会话（access），建立与云手机的连接。在本地会话对象初始化成功后（参见 创建会话 ），调用此方法以建立和指定 ID 的云手机的连接。在调用此方法之前，需要在 init() 方法的 TcrConfig 内传入正确的 accessToken 或者调用了 setAccessToken()。
建连的结果会异步回调给 TcrSession#Observer#onEvent() 方法，成功事件为 TcrSession.Event#STATE_CONNECTED，失败事件为 TcrSession.Event#STATE_CLOSED。使用者只有在收到成功建立连接的事件后，才能使用此会话对象与指定的云手机进行交互。

输入参数

参数名称	必选	类型	描述
instanceIds	是	ArrayList<String>	此会话要控制的实例 ID 列表。如果 isGroupControl 为 false，则只会取列表里的第一个元素。

返回值

连接请求是否成功发起（注意：实际连接结果异步回调返回）。

- 返回 true：已成功发起连接请求。
- 返回 false：请求发送失败（通常因参数无效或会话状态异常）。

示例

参见前面 创建会话 的示例。

释放会话

释放会话（release），客户端释放本地会话资源。云端会在自动检测到心跳消失后释放云端资源。

示例

```
mTcrSession.release();
```

获取会话请求 ID

获取当前会话的请求 ID（getRequestId），用于和 SDK 后台排查问题。

示例

```
mTcrSession.getRequestId();
```

开关音频播放

setEnableAudioPlaying

输入参数

enable：true 为启用音频播放，false 为禁用。

返回值

无

调用示例

参考 TcrDemo 。

开关音频上行

启用或禁用从麦克风捕获的本地音轨。启用后，本地捕获的音频会上行至云端。

输入参数

enableLocalAudio：true 为启用，false 为禁用。

返回值

无

调用示例

参考 TcrDemo 。

开关视频上行

setEnableLocalAudio

启用或禁用从摄像头捕获的本地视频。启用后，本地捕获的视频画面会上行至云端。

输入参数

enableLocalVideo：true 为启用，false 为禁用。

返回值

无

调用示例

参考 TcrDemo 。

暂停音视频流下行

pauseStreaming

调用示例

参考 TcrDemo 。

恢复音视频流下行

resumeStreaming

调用示例

参考 TcrDemo 。

设置上行的视频的参数

setLocalVideoProfile
配置本地上行的视频的参数（分辨率、帧率、码率、摄像头方向）。

输入参数

参数名称	必选	类型	描述
width	是	int	视频宽度。
height	是	int	视频高度。
fps	是	int	帧率。
minBitrate	是	int	最小比特率。
maxBitrate	是	int	最大比特率。

isFrontCamera	是	BOOL	是否为前置摄像头。

返回值

无

调用示例

参考 TcrDemo 。

设置下行的音频的参数

设置远端下行的音频的音量参数（setRemoteAudioPlayProfile）。

输入参数

参数名称	必选	类型	描述
volume	是	float	音量系数，取值 [0,10]。

返回值

无

调用示例

参考 TcrDemo 。

设置下行视频的接收器

setVideoSink
设置视频渲染目标。SDK 将解码后的视频帧回调到 videoSink，用于后处理或渲染。

输入参数

videoSink：视频接收器，传 null 可移除现有目标。

返回值

无

调用示例

参考 TcrDemo 。

设置下行音频的接收器

setAudioSink

为本次会话设置一个音频接收器。设置完成后，SDK 会将音频数据回调到该音频接收器。

输入参数

audioSink：音频接收器，传 null 可移除现有目标。

返回值

无

调用示例

参考 TcrDemo 。

重启云端应用

restartCloudApp
重启云端应用进程。

输入参数

无

返回值

无

调用示例

参考 TcrDemo 。

向云应用粘贴文本

pasteText
向云端应用输入框粘贴文本（需输入框已聚焦）。

输入参数

参数名称	必选	类型	描述
text	是	String	要粘贴的文本内容。

返回值

无

调用示例

参考 TcrDemo 。

禁用云端软键盘

setDisableCloudInput
禁用云端软键盘（仅支持移动端应用）。

输入参数

参数名称	必选	类型	描述
disableCloudInput	是	boolean	是否禁用云端输入。

返回值

无

调用示例

参考 TcrDemo 。

获取云端触摸屏交互对象

getTouchScreen
获取当前会话的云端触摸屏交互对象，用于和云端触摸屏进行交互。该方法将本地触摸视图的事件转换为云端触摸的事件，实现远程触控交互。

输入参数

无

返回值

可以和云端触摸屏进行交互的 TouchScreen 对象。

调用示例

参考 TcrDemo 。

触发云端触屏触摸事件

touch
向云端触摸屏发送触摸事件，模拟用户对云端设备的触控操作（如点击、滑动等）。

输入参数

参数名称	必选	类型	描述
x	是	float	触摸点在渲染视图中的水平坐标。

y	是	float	触摸点在渲染视图中的垂直坐标。
eventType	是	int	触摸事件类型，0代表按下，1代表移动，2代表抬起。
fingerID	是	int	触摸点唯一标识符。同一手势中，同一手指的多次事件（如按下→移动→抬起）共享相同 fingerID，用于跟踪多点触控轨迹。 来源：通常从 android.view.MotionEvent.getPointerId(int) 获取。
width	是	int	渲染视图的宽度（像素单位）。用于坐标归一化计算，确保 x 值在 [0, width] 范围内有效。
height	是	int	渲染视图的高度（像素单位）。用于坐标归一化计算，确保 y 值在 [0, height] 范围内有效。
timestamp	是	long	• 含义：事件发生的时间戳（单位：毫秒），基于系统启动时间（SystemClock.uptimeMillis()）。 • 用途：保证事件顺序一致性，云端可据此处理时序逻辑（如手势识别）。 • 来源：从 MotionEvent.getEventTime() 获取。

返回值

无

调用示例

参考 TcrDemo 。

获取实例截图地址

获取实例截图地址（getInstanceImageAddress）

输入参数

Map<String, Object> params，内容为：

参数名称	必选	类型	描述
instanceId	是	String	实例 ID。
quality	否	int	截图质量，取值范围 0-100，默认 20。
screenshot_width	否	int	截图宽度。

screenshot_height	否	int	截图高度。

返回值

参数名称	类型	描述
url	String	截图地址，失败返回 null。

示例

```
Map<String, Object> params = new HashMap<>();
params.put("instance_id", mInstanceIds.get(0));
params.put("screenshot_quality", 80);
params.put("screenshot_width", 1080);
params.put("screenshot_height", 1920);
String url = getAndroidInstance().getInstanceImageAddress(params);
```

获取实例截图

获取实例截图信息（getInstanceImage）。

输入参数

Map<String, Object> params，内容为：

参数名称	必选	类型	描述
instanceId	是	String	实例 ID。
quality	否	int	截图质量，取值范围 0-100，默认 20。
screenshot_width	否	int	截图宽度。
screenshot_height	否	int	截图高度。

AsyncCallback 异步回调对象。

返回值

异步回调 AsyncCallback，成功返回 Bitmap 对象。

示例

```
Map<String, Object> params = new HashMap<>();
params.put("instance_id", mInstanceIds.get(0));
params.put("screenshot_quality", 80);
params.put("screenshot_width", 1080);
params.put("screenshot_height", 1920);
AsyncCallback<Bitmap> callback = new AsyncCallback<Bitmap>() {
    @Override
    public void onSuccess(Bitmap bitmap) {
    }

    @Override
    public void onFailure(int code, String errorMsg) {
    }
};
getAndroidInstance().getInstanceImage(params, callback);
```

上传文件到实例

上传文件到实例（uploadFile）。
默认上传到 /sdcard/Download 目录下，可使用 path 指定上传目录（仅支持/sdcard/ 下目录）。

输入参数

参数名称	必选	类型	描述
instanceId	是	String	目标云手机实例 ID。
files	是	UploadFileItem[]	要上传的文件数组。
file.fileBytes	是	byte[]	文件二进制内容。
file.file	是	String	文件名（含扩展名）。
file.path	否	String	云端存储路径，如 "/sdcard/Download/"。
callback	否	AsyncCallback<String>	异步回调对象。

返回值

JSON 字符串：**{Code: int; Message: string; FileStatus: [{ CloudPath: string; FileName: string }]}**

参数名称	类型	描述
Code	int	错误码。
Message	String	错误消息。
FileStatus	Object[]	文件状态对象。
FileStatus.CloudPath	String	云端路径。
FileStatus.FileName	String	文件名。

示例

```
private void upload(ArrayList<Uri> fileList, boolean isMedia) {
    UploadFileItem[] files = new UploadFileItem[fileList.size()];
    for (int i = 0; i < fileList.size(); i++) {
        files[i] = new UploadFileItem();
        files[i].fileBytes = readFilesFromUris(fileList.get(i));
        files[i].fileName = getFileNameFromUri(fileList.get(i));
        files[i].filePath = "/sdcard/Movies/";
    }

    AsyncCallback<String> callback = new AsyncCallback<String>() {
        @Override
        public void onSuccess(String response) {
            // 示例数据 {Code=0, Message='null', FileStatus=
[FileStatus{FileName='testDownloadFile',
CloudPath='/sdcard/Download/testDownloadFile'}]}
            UploadResponse uploadResponse = GsonUtils.fromJson(response,
UploadResponse.class);
            LogUtils.d(TAG, "get response: " + uploadResponse);
            if (uploadResponse == null) {
                mResultTextView.setText("文件上传异常: uploadResponse 解析
失败");
                return;
            }
            if (uploadResponse.Code == 0) {
                StringBuilder sb = new StringBuilder("文件上传成功:\n");
                if (uploadResponse.FileStatus != null) {
                    for (UploadResponse.FileStatus status :
uploadResponse.FileStatus) {
```

```
                            sb.append("文件名:
").append(status.FileName).append("\n云端路径: ")
                                    .append(status.CloudPath).append("\n");
                        }
                    }
                    mResultTextView.setText(sb.toString());
                } else {
                    mResultTextView.setText("文件上传失败: " +
uploadResponse.Message);
                }
            }

            @Override
            public void onFailure(int code, String errorMsg) {
                mResultTextView.setText("文件上传失败: " + code + ", " +
errorMsg);
            }
        };
        if (isMedia) {
            getAndroidInstance().uploadMedia(mInstanceIds.get(0), files,
callback);
        } else {
            getAndroidInstance().uploadFile(mInstanceIds.get(0), files,
callback);
        }
    }
```

上传媒体到实例

上传媒体文件到实例（uploadMedia）。
固定上传至 /data/media/0/DCIM 目录，上传完毕会通知系统添加媒体文件到相册中。

输入参数

与“上传文件到实例（uploadFile）”方法一样，但会忽略 file.path 字段，云端固定存储至 /data/media/0/DCIM 目录。

返回值

JSON 字符串：**{Code: int; Message: string;}**

示例

参考“上传文件到实例（uploadFile）”。

获取文件下载地址

获取实例文件的下载地址（getInstanceDownloadAddress）。

输入参数

参数名称	必选	类型	描述
instanceId	是	String	实例 ID。
path	是	String	要下载的云端文件的路径。

返回值

参数名称	类型	描述
address	String	下载地址，失败返回 null。

示例

```
String address =
getAndroidInstance().getInstanceDownloadAddress(mInstanceIds.get(0),
"/sdcard/Download/testDownloadFile");
```

获取日志文件下载地址

获取云手机实例的日志文件的下载地址（getInstanceDownloadLogcatAddress）。

输入参数

参数名称	必选	类型	描述
instanceId	是	String	实例 ID。
recentDays	是	int	表示下载几天内修改过的 logcat 日志文件。

返回值

参数名称	类型	描述
address	String	下载地址，失败返回 null。

示例

```
String address =
getAndroidInstance().getInstanceDownloadAddress(mInstanceIds.get(0), 3);
```

下载文件

下载文件（downloadFile）。

输入参数

参数名称	必选	类型	描述
params	是	Map<String, Object>	下载参数，包含： • instance_id：云手机实例 ID。 • path：要下载的云端文件的路径。
localFile	是	File	下载文件存储到本地的 File，应用需确保有写入权限。
callback	否	AsyncCallback <String>	异步回调对象。

返回值

异步回调结果。

示例

```
String localPath = "/sdcard/Download/testDownloadFile";
String cloudPath = "/sdcard/Download/testDownloadFile";
Map<String, Object> params = new HashMap<>();
params.put("instance_id", mInstanceIds.get(0));
params.put("path", cloudPath);
AsyncCallback<String> callback = new AsyncCallback<String>() {
    @Override
    public void onSuccess(String response) {
        mResultTextView.setText("文件下载成功");
    }

    @Override
    public void onFailure(int code, String errorMsg) {
        mResultTextView.setText("文件下载失败: " + code + ", " +
errorMsg);
    }
};
```

```
getAndroidInstance().downloadFile(params, new File(localPath),
callback);
```

切换输入法

切换输入法（switchIME）。

输入参数

参数名称	必选	类型	描述
ime	是	String	• "cloud"：云端输入法。 • "local"：本地输入法。

示例

```
getAndroidInstance().switchIME("local");
```

设置设备 GPS 信息

设置设备 GPS 信息（setLocation），支持批量实例操作。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { Longitude: double; Latitude: double } }

参数名称	必选	类型	描述
Longitude	是	double	经度。
Latitude	是	double	纬度。

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    JsonObject param = new JsonObject();
    param.addProperty("Longitude", 121.4737);
    param.addProperty("Latitude", 31.2304);
```

```
    params.put(instanceId, param);
}
getAndroidInstance().setLocation(params, mBatchCallback);
```

设置设备分辨率

设置设备分辨率（setResolution）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { Width: int; Height: int; DPI?: int } };

参数名称	必选	类型	描述
Width	是	int	宽。
Height	是	int	高。
DPI	否	int	像素密度。

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    JsonObject param = new JsonObject();
    param.addProperty("Width", 1080);
    param.addProperty("Height", 1920);
    param.addProperty("DPI", 420);
    params.put(instanceId, param);
}
getAndroidInstance().setResolution(params, mBatchCallback);
```

粘贴文本

粘贴文本（paste）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { Text: string } }

参数名称	必选	类型	描述
Text	是	String	文本。

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    JsonObject param = new JsonObject();
    param.addProperty("Text", "粘贴的文本内容");
    params.put(instanceId, param);
}
getAndroidInstance().paste(params, mBatchCallback);
```

发送文本到剪切板

发送文本到剪切板（sendClipboard）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { Text: string } }

参数名称	必选	类型	描述
Text	是	String	文本。

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    JsonObject param = new JsonObject();
    param.addProperty("Text", "剪贴板内容");
    params.put(instanceId, param);
}
getAndroidInstance().sendClipboard(params, mBatchCallback);
```

版权所有：腾讯云计算（北京）有限责任公司

摇一摇

摇一摇（shake）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { } }

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    params.put(instanceId, new JsonObject());
}
getAndroidInstance().shake(params, mBatchCallback);
```

设置设备传感器信息

设置设备传感器信息（setSensor）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { Type: 'accelerometer' | 'gyroscope'; Values: double[] } }

参数名称	必选	类型	描述
Type	是	String	传感器 leasing accelerometer（加速器），gyroscope（陀螺仪）。
Accuracy	否	int	精度。
Values	是	double[]	传感器对应的值，数组长度为 3，分别表示 x/y/z 轴的值。

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
```

```
for (String instanceId : mInstanceIds) {
    JsonObject param = new JsonObject();
    param.addProperty("Type", "accelerometer");
    param.addProperty("Accuracy", 3);

    JsonArray values = new JsonArray();
    values.add(0.5);
    values.add(-0.3);
    values.add(0.8);
    param.add("Values", values);

    params.put(instanceId, param);
}
getAndroidInstance().setSensor(params, mBatchCallback);
```

发送 App binder 消息

发送 App binder 消息（sendTransMessage）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { PackageName: string; Msg: string } }

参数名称	必选	类型	描述
PackageName	是	String	包名。
Msg	是	String	消息。

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    JsonObject param = new JsonObject();
    param.addProperty("PackageName", "com.example.app");
    param.addProperty("Msg", "消息内容");
    params.put(instanceId, param);
}
```

```
getAndroidInstance().sendTransMessage(params, mBatchCallback);
```

查询实例属性

查询实例属性（describeInstanceProperties）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { } }

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    params.put(instanceId, new JsonObject());
}
getAndroidInstance().describeInstanceProperties(params, mBatchCallback);
```

修改实例属性

修改实例属性（modifyInstanceProperties）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: **param** }，其中 param 内容参考：

```
{
  "RequestID": "modify_req_20230901001",
  "DeviceInfo": {
    "Brand": "Samsung",
    "Model": "Galaxy S24"
  },
  "ProxyInfo": {
    "Enabled": true,
    "Protocol": "socks5",
    "Host": "proxy.example.com",
    "Port": 1080,
    "User": "user123",
    "Password": "pass123"
  },
```

```
  "GPSInfo": {
    "Longitude": 121.4737,
    "Latitude": 31.2304
  },
  "SIMInfo": {
    "State": 1 // sim 状态。 1：未插入 sim 卡 5：sim 卡已就绪
  },
  "LocaleInfo": {
    "Timezone": "Asia/Shanghai"
  },
  "LanguageInfo": {
    "Language": "zh",
    "Country": "CN"
  },
  "ExtraProperties": [
    {
      "Key": "custom_property1",
      "Value": "value1"
    },
    {
      "Key": "custom_property2",
      "Value": "value2"
    }
  ]
}
```

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    JsonObject param = new JsonObject();
    param.addProperty("RequestID", UUID.randomUUID().toString());

    JsonObject deviceInfo = new JsonObject();
    deviceInfo.addProperty("Brand", "brand");
    deviceInfo.addProperty("Model", "model");
    param.add("DeviceInfo", deviceInfo);

    JsonObject proxyInfo = new JsonObject();
```

```
    proxyInfo.addProperty("Enabled", true);
    proxyInfo.addProperty("Protocol", "socks5");
    proxyInfo.addProperty("Host", "proxy.example.com");
    proxyInfo.addProperty("Port", 1080);
    proxyInfo.addProperty("User", "user");
    proxyInfo.addProperty("Password", "password");
    param.add("ProxyInfo", proxyInfo);

    JsonObject gpsInfo = new JsonObject();
    gpsInfo.addProperty("Longitude", 121.4737);
    gpsInfo.addProperty("Latitude", 31.2304);
    param.add("GPSInfo", gpsInfo);

    JsonObject simInfo = new JsonObject();
    simInfo.addProperty("State", 1);
    simInfo.addProperty("PhoneNumber", "1234567890");
    simInfo.addProperty("IMSI", "imsi");
    simInfo.addProperty("ICCID", "iccid");
    param.add("SIMInfo", simInfo);

    JsonObject localeInfo = new JsonObject();
    localeInfo.addProperty("Timezone", "Asia/Shanghai");
    param.add("LocaleInfo", localeInfo);

    JsonObject languageInfo = new JsonObject();
    languageInfo.addProperty("Language", "zh");
    languageInfo.addProperty("Country", "CN");
    param.add("LanguageInfo", languageInfo);

    JsonArray extraProperties = new JsonArray();
    JsonObject extraProp = new JsonObject();
    extraProp.addProperty("Key", "key");
    extraProp.addProperty("Value", "value");
    extraProperties.add(extraProp);
    param.add("ExtraProperties", extraProperties);

    params.put(instanceId, param);
}
getAndroidInstance().modifyInstanceProperties(params, mBatchCallback);
```

查询已安装第三方应用

查询已安装第三方应用（listUserApps）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { } }

返回值

类型：BatchTaskResponse
其中每个实例 ID 返回的 JsonElement 内容参考：

```
{
  "AppList": [
    {
      "PackageName": "com.android.chrome",
      "LastUpdateTimeMs": 1667232000000,
      "FirstInstallTimeMs": 1635724800000,
      "VersionName": "115.0.5790.166",
      "Label": "Chrome"
    },
    {
      "PackageName": "com.tencent.mm",
      "LastUpdateTimeMs": 1667328405000,
      "FirstInstallTimeMs": 1635811200000,
      "VersionName": "8.0.35",
      "Label": "微信"
    },
    {
      "PackageName": "com.zhihu.android",
      "LastUpdateTimeMs": 1667256803000,
      "FirstInstallTimeMs": 1635897600000,
      "VersionName": "8.27.0",
      "Label": "知乎"
    }
  ],
  "Code": 0,
  "Msg": "xx"
}
```

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    params.put(instanceId, new JsonObject());
}
```

```
getAndroidInstance().listUserApps(params, mBatchCallback);
```

修改前台应用保活状态

修改前台应用保活状态（modifyKeepFrontAppStatus）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { PackageName: string; Enable: boolean; RestartInterValSeconds: int; } }

参数名称	必选	类型	描述
PackageName	是	String	包名。
Enable	是	Boolean	是否保活。
RestartInterValSeconds	是	int	重新拉起最长间隔。

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    JsonObject param = new JsonObject();
    param.addProperty("PackageName", "com.example.app");
    param.addProperty("Enable", true);
    param.addProperty("RestartInterValSeconds", 5);
    params.put(instanceId, param);
}
getAndroidInstance().modifyKeepFrontAppStatus(params, mBatchCallback);
```

查询前台应用保活状态

查询前台应用保活状态（describeKeepFrontAppStatus）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { } }

返回值

类型：BatchTaskResponse
其中每个实例 ID 返回的 JsonElement 内容参考：

```
{
  "PackageName": "com.example.app",
  "Enable": true,
  "RestartInterValSeconds": 3,
  "Code": 0,
  "Msg": "xx"
}
```

参数名称	类型	描述
PackageName	String	包名。
Enable	Boolean	是否保活。
RestartInterValSeconds	int	重新拉起最长间隔。

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    params.put(instanceId, new JsonObject());
}
getAndroidInstance().describeKeepFrontAppStatus(params, mBatchCallback);
```

卸载应用

卸载应用（unInstallByPackageName）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { PackageName: string; } }

参数名称	必选	类型	描述

PackageName	是	String	包名。

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    JsonObject param = new JsonObject();
    param.addProperty("PackageName", "com.example.app");
    params.put(instanceId, param);
}
getAndroidInstance().unInstallByPackageName(params, mBatchCallback);
```

启动应用

启动应用（startApp）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { PackageName: string; ActivityName: string } }

参数名称	必选	类型	描述
PackageName	是	String	包名。
ActivityName	是	String	启动的 Activity 名称。

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    JsonObject param = new JsonObject();
    param.addProperty("PackageName", "com.example.app");
    param.addProperty("ActivityName", "com.example.app.MainActivity");
    params.put(instanceId, param);
```

```
}
getAndroidInstance().startApp(params, mBatchCallback);
```

停止应用

停止应用（stopApp）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { PackageName: string; } }

参数名称	必选	类型	描述
PackageName	是	String	包名。

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    JsonObject param = new JsonObject();
    param.addProperty("PackageName", "com.example.app");
    params.put(instanceId, param);
}
getAndroidInstance().stopApp(params, mBatchCallback);
```

清除应用数据

清除应用数据（clearAppData）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { PackageName: string; } }

参数名称	必选	类型	描述
PackageName	是	String	包名。

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    JsonObject param = new JsonObject();
    param.addProperty("PackageName", "com.example.app");
    params.put(instanceId, param);
}
getAndroidInstance().clearAppData(params, mBatchCallback);
```

启用应用

启用应用（enableApp）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { PackageName: string; } }

参数名称	必选	类型	描述
PackageName	是	String	包名。

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    JsonObject param = new JsonObject();
    param.addProperty("PackageName", "com.example.app");
    params.put(instanceId, param);
}
getAndroidInstance().enableApp(params, mBatchCallback);
```

禁用应用

禁用应用（disableApp）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { PackageName: string; } }

参数名称	必选	类型	描述
PackageName	是	String	包名。

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    JsonObject param = new JsonObject();
    param.addProperty("PackageName", "com.example.app");
    params.put(instanceId, param);
}
getAndroidInstance().disableApp(params, mBatchCallback);
```

摄像头播放媒体文件

摄像头播放媒体文件（startCameraMediaPlay）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { FilePath: string; Loops: int }; }

参数名称	必选	类型	描述
FilePath	是	String	文件路径。
Loops	是	int	循环次数，负数表示无限循环。

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    JsonObject param = new JsonObject();
    param.addProperty("FilePath", "/sdcard/media.mp4");
    param.addProperty("Loops", 3);
    params.put(instanceId, param);
}
getAndroidInstance().startCameraMediaPlay(params, mBatchCallback);
```

摄像头停止播放媒体文件

摄像头停止播放媒体文件（stopCameraMediaPlay）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { }; }

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    params.put(instanceId, new JsonObject());
}
getAndroidInstance().stopCameraMediaPlay(params, mBatchCallback);
```

查询当前摄像头媒体播放状态

查询当前摄像头媒体播放状态（describeCameraMediaPlayStatus）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { }; }

返回值

类型：BatchTaskResponse
其中每个实例 ID 返回的 JsonElement 内容参考：

```
{
  "FilePath": "/sdcard/video.mp4",
```

```
  "Loops": 3,
  "Code": 0,
  "Msg": "xx"
}
```

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    params.put(instanceId, new JsonObject());
}
getAndroidInstance().describeCameraMediaPlayStatus(params,
mBatchCallback);
```

摄像头显示图片

摄像头显示图片（displayCameraImage）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { FilePath: string; }; }

参数名称	必选	类型	描述
FilePath	是	String	文件路径。

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    JsonObject param = new JsonObject();
    param.addProperty("FilePath", "/sdcard/image.png");
    params.put(instanceId, param);
}
getAndroidInstance().displayCameraImage(params, mBatchCallback);
```

增加后台保活应用

增加后台保活应用（addKeepAliveList）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { AppList: string[] } }

参数名称	必选	类型	描述
AppList	是	String[]	App 列表。

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    JsonObject param = new JsonObject();
    JsonArray appList = new JsonArray();
    appList.add("com.example.app");
    param.add("AppList", appList);
    params.put(instanceId, param);
}
getAndroidInstance().addKeepAliveList(params, mBatchCallback);
```

移除后台保活应用

移除后台保活应用（removeKeepAliveList）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { AppList: string[] } }

参数名称	必选	类型	描述
AppList	是	String[]	App 列表。

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
```

```
for (String instanceId : mInstanceIds) {
    JsonObject param = new JsonObject();
    JsonArray appList = new JsonArray();
    appList.add("com.example.app");
    param.add("AppList", appList);
    params.put(instanceId, param);
}
getAndroidInstance().removeKeepAliveList(params, mBatchCallback);
```

覆盖后台保活应用

覆盖后台保活应用（setKeepAliveList）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { AppList: string[] } }

参数名称	必选	类型	描述
AppList	是	String[]	App 列表。

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    JsonObject param = new JsonObject();
    JsonArray appList = new JsonArray();
    appList.add("com.example.app");
    param.add("AppList", appList);
    params.put(instanceId, param);
}
getAndroidInstance().setKeepAliveList(params, mBatchCallback);
```

查询后台保活应用

查询后台保活应用（describeKeepAliveList）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: {}; }

返回值

类型：BatchTaskResponse
其中每个实例 ID 返回的 JsonElement 内容参考：

```
{
  "AppList": ["com.wechat", "com.alipay", "com.dingtalk"],
  "Code": 0,
  "Msg": "xx"
}
```

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    params.put(instanceId, new JsonObject());
}
getAndroidInstance().describeKeepAliveList(params, mBatchCallback);
```

清空后台保活应用

清空后台保活应用（clearKeepAliveList）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { }; }

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    params.put(instanceId, new JsonObject());
}
getAndroidInstance().clearKeepAliveList(params, mBatchCallback);
```

静音开关

静音开关（mute）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { Mute: boolean } }

参数名称	必选	类型	描述
Mute	是	Boolean	是否静音。

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    JsonObject param = new JsonObject();
    param.addProperty("Mute", true);
    params.put(instanceId, param);
}
getAndroidInstance().mute(params, mBatchCallback);
```

媒体库文件搜索

媒体库文件搜索（mediaSearch）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { Keyword: string } }

参数名称	必选	类型	描述
Keyword	是	String	关键字。

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    JsonObject param = new JsonObject();
    param.addProperty("Keyword", "搜索关键词");
    params.put(instanceId, param);
}
```

```
getAndroidInstance().mediaSearch(params, mBatchCallback);
```

重启实例

重启实例（reboot）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { }; }

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    params.put(instanceId, new JsonObject());
}
getAndroidInstance().reboot(params, mBatchCallback);
```

查询所有应用列表

查询所有应用列表（listAllApps）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { }

返回值

类型：BatchTaskResponse
其中每个实例 ID 返回的 JsonElement 内容参考：

```
{
  "AppList": [
    {
      "PackageName": "com.android.chrome",
      "LastUpdateTimeMs": 1667232000000,
      "FirstInstallTimeMs": 1635724800000,
      "VersionName": "115.0.5790.166",
      "Label": "Chrome"
    },
    {
```

```
      "PackageName": "com.tencent.mm",
      "LastUpdateTimeMs": 1667328405000,
      "FirstInstallTimeMs": 1635811200000,
      "VersionName": "8.0.35",
      "Label": "微信"
    },
    {
      "PackageName": "com.zhihu.android",
      "LastUpdateTimeMs": 1667256803000,
      "FirstInstallTimeMs": 1635897600000,
      "VersionName": "8.27.0",
      "Label": "知乎"
    }
  ],
  "Code": 0,
  "Msg": "xx"
}
```

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    params.put(instanceId, new JsonObject());
}
getAndroidInstance().listAllApps(params, mBatchCallback);
```

关闭应用至后台

关闭应用至后台（moveAppBackground）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { }; }

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    params.put(instanceId, new JsonObject());
}
```

```
getAndroidInstance().moveAppBackground(params, mBatchCallback);
```

新增应用安装黑名单

新增应用安装黑名单（addAppInstallBlackList）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { AppList: string[] } }

参数名称	必选	类型	描述
AppList	是	String[]	App 列表。

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    JsonObject param = new JsonObject();
    JsonArray appList = new JsonArray();
    appList.add("com.example.app");
    param.add("AppList", appList);
    params.put(instanceId, param);
}
getAndroidInstance().addAppInstallBlackList(params, mBatchCallback);
```

移除应用安装黑名单

移除应用安装黑名单（removeAppInstallBlackList）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { AppList: string[] } }

参数名称	必选	类型	描述
AppList	是	String[]	App 列表。

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    JsonObject param = new JsonObject();
    JsonArray appList = new JsonArray();
    appList.add("com.example.app");
    param.add("AppList", appList);
    params.put(instanceId, param);
}
getAndroidInstance().removeAppInstallBlackList(params, mBatchCallback);
```

覆盖应用安装黑名单

覆盖应用安装黑名单（setAppInstallBlackList）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { AppList: string[] } }

参数名称	必选	类型	描述
AppList	是	String[]	App 列表。

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    JsonObject param = new JsonObject();
    JsonArray appList = new JsonArray();
    appList.add("com.example.app");
    param.add("AppList", appList);
    params.put(instanceId, param);
}
getAndroidInstance().setAppInstallBlackList(params, mBatchCallback);
```

查询应用安装黑名单

查询应用安装黑名单（describeAppInstallBlackList）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { }; }

返回值

类型：BatchTaskResponse
其中每个实例 ID 返回的 JsonElement 内容参考：

```
{
  "AppList": ["com.wechat", "com.alipay", "com.dingtalk"],
  "Code": 0,
  "Msg": "xx"
}
```

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    params.put(instanceId, new JsonObject());
}
getAndroidInstance().describeAppInstallBlackList(params,
mBatchCallback);
```

清空应用安装黑名单

清空应用安装黑名单（clearAppInstallBlackList）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { }; }

返回值

类型：BatchTaskResponse

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    params.put(instanceId, new JsonObject());
}
```

```
getAndroidInstance().clearAppInstallBlackList(params, mBatchCallback);
```

获取系统导航栏显示状态

获取系统导航栏显示状态（getNavVisibleStatus）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { }; }

返回值

类型：BatchTaskResponse
其中每个实例 ID 返回的 JsonElement 内容参考：{ Visible: boolean}

参数名称	类型	描述
Visible	Boolean	是否可见。

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    params.put(instanceId, new JsonObject());
}
getAndroidInstance().getNavVisibleStatus(params, mBatchCallback);
```

获取系统媒体音量大小

获取系统媒体音量大小（getSystemMusicVolume）。

输入参数

类型：Map<String, JsonObject> params，内容为 { [InstanceId: string]: { }; }

返回值

类型：BatchTaskResponse
其中每个实例 ID 返回的 JsonElement 内容参考：

```
{
    "Volume":15,
    "Code": 0,
    "Msg": "xx"
```

```
}
```

示例

```
Map<String, JsonObject> params = new HashMap<>();
for (String instanceId : mInstanceIds) {
    params.put(instanceId, new JsonObject());
}
getAndroidInstance().getSystemMusicVolume(params, mBatchCallback);
```

相关数据结构

批量操作返回值

批量操作返回值（**BatchTaskResponse**）。

数据结构

参数名称	类型	描述
Code	int	• Code 为 0 表示所有操作成功完成，非0值表示操作失败或部分失败。 • Code：10001, Msg: "invalid param"。 • Code：10002, Msg: "invalid token"。 • Code：10003, Msg: "invalid operate"。
Msg	String	暂无意义。
SuccResult	Map<String, JsonElement>	成功操作的实例结果映射： • key：云手机实例 ID。 • value：该实例操作成功的返回结果对象，使用 JsonElement 封装，具体内容取决于操作类型，见具体操作接口说明。
FailResult	Map<String, JsonElement>	失败操作的实例结果映射： key：云手机实例 ID。 value：该实例操作失败的返回结果对象，使用 JsonElement 封装，通常包含错误码和错误信息。例如： `{ "Code": 10100, "Msg": "cai operation denied" }`

使用示例

```
AndroidInstance instance = TcrSdk.getInstance().getAndroidInstance();
Map<String, JsonObject> params = new HashMap<>();
String[] instanceIds = {"instance1", "instance2"};
for (String instanceId : instanceIds) {
    JsonObject param = new JsonObject();
    param.addProperty("Width", 1080);
    param.addProperty("Height", 1920);
    param.addProperty("DPI", 420);
    params.put(instanceId, param);
}
AsyncCallback<BatchTaskResponse> callback = new
AsyncCallback<BatchTaskResponse>() {
    public void onSuccess(BatchTaskResponse response) {
        // 遍历成功实例
        for (Entry<String, JsonElement> entry :
response.SuccResult.entrySet()) {
            String instanceId = entry.getKey();
            JsonElement result = entry.getValue();
            // 处理结果
        }
        // 遍历失败实例
        for (Entry<String, JsonElement> entry :
response.FailResult.entrySet()) {
            String instanceId = entry.getKey();
            JsonElement result = entry.getValue();
            // 处理错误
        }
    }

    @Override
    public void onFailure(int code, String msg) {
        // 处理错误
        Log.e(TAG, "Operation failed: " + code + ", " + msg);
    }
};
instance.setResolution(params, callback);
```

事件回调

最近更新时间：2025-09-03 14:10:22

会话事件

TcrSession#TcrEvent 事件枚举，通过 TcrSession#Observer#onEvent() 方法回调。

会话生命周期事件

事件代码	事件含义	事件详细描述
STATE_INITED	会话已初始化	详情参见 创建会话 。 事件数据为 String (client Session)。
STATE_CONNECTED	会话已连接	详情参见 启动会话 。 无事件数据。
STATE_RECONNECTING	会话正在重连	在网络中断时 SDK 会尝试进行会话重连，App 会收到此事件。App 可以在 UI 上提示用户，例如“网络连接中...”。 无事件数据。
STATE_CLOSED	会话已关闭	在 创建会话 失败、启动会话 失败、主动 释放会话 、或者其他异常原因导致会话结束的时候，App 会收到此事件。 收到此事件后，会话已关闭，无法再使用。App 可以记录错误日志，上报运营分析等。 事件数据为 Integer 类型，具体值范围和含义参考 `TcrCode#SessionStopBaseCode` 。

会话状态事件

事件代码	事件含义	事件详细描述
CLIENT_STATS	性能数据	App 会定时收到此事件；App 可以记录和分析性能数据。 事件数据为 StatsInfo 类型。
CLIENT_IDLE	客户端空闲	当用户长时间（ `TcrSessionConfig#idleThreshold` 配置）未操作鼠标、键盘或屏幕时，App 会收到此事件。App 可以根据自己的业务逻辑进行处理，如主动结束会话、忽略等。 无事件数据。
CLIENT_LOW_FPS	客户端低帧率	当客户端帧率连续10秒保持低水平时（ `TcrSessionConfig#lowFpsThreshold` 配置），App 会收到此事件。App 可以根据自己的业务逻辑进行处理，如提醒用户、忽略等。

		无事件数据。

云端状态事件

事件代码	事件含义	事件详细描述
OPEN_URL	打开链接	当在云机上点击了可打开的 URL 链接时，App 会收到此事件。App 可以处理 URL 点击事件，例如打开浏览器。 事件数据为 URL 字符串。
CAMERA_STATUS_CHANGED	摄像头状态改变	当云机上的摄像头状态发生改变时，App 会收到此事件。App 可以处理摄像头状态事件，例如调整 UI 等。 事件数据为 JSON 格式的 String 类型，包含分辨率宽度、高度和摄像头状态，示例数据： `{ "status": "open_back/open_front/close", "height": 720, "width": 1280 }` 其中 status 取值和含义： • "open_back"：打开后置摄像头。 • "open_front"：打开前置摄像头。 • "close"：关闭摄像头。
IME_TYPE	输入法类型	在刚连接上云机时，或者云机 切换输入法 时，App 会收到此事件。App 可以切换本地的输入状态或 UI。 事件数据为 JSON 格式的 String 类型，包含云机当前的 IME 状态，示例数据： `{"ime_type":"cloud"/"local"}` 其中： • "cloud"：云端输入法。 • "local"：本地输入法。
INPUT_STATUS_CHANGED	输入状态改变	当云机上的输入状态发生改变时，App 会收到此事件。App 可以处理更新本地的输入状态，例如禁用用户输入控件。 事件数据为 JSON 格式的 String 类型，包含状态信息。示例数据： `{status:"disabled/enabled"}` 其中： • disabled：禁止用户输入 • enabled：允许用户输入
SCREEN_CONFIG_CHAN	屏幕配置改变	当云机上的屏幕配置发生改变时，App 会收到此事件。App 可以更新本地的页面屏幕配置方向，以和云端一致，例如切换

GE		Activity 的横竖屏。 事件数据为 ScreenConfig 类型，包含屏幕方向、角度等信息。
VIDEO_STREAM_CONFIG_CHANGED	视频流配置改变	当云机串流的视频流配置发生改变时，App 会收到此事件。App 可以按需更新解码器、渲染视图的参数。 事件数据为 VideoStreamConfig 类型，包含视频宽度、高度等信息。

INPUT_STATE_CHANGE	输入框状态改变	当云机上的输入框状态（如焦点）发生改变时，App 会收到此事件。App 可以处理更新本地的输入框状态，例如同步文本内容。
		事件数据为 JSON 格式的 String 类型，包含输入框类型、文本内容等信息。示例数据：
		`{ "field_type": "normal_input|autologin_input|unfocused", "status": "disabled/enabled", "input_type": 32, "text": "xxx", "start_index": 0 }`
		其中：
		• field_type：
		  ○ normal_input：点击普通窗口的输入框。
		  ○ autologin_input：【废弃】点击登录窗口的输入框（登录窗口由云端 RunningConfig 配置），可以配合 ack 的 auto_login 消息进行自动登录。
		  ○ unfocused：从输入框移出光标。
		• status（只有端游支持，在自动登录过程中为了避免客户干扰，会忽略所有客户端发过来的控制消息）：
		  ○ disabled：禁止客户端输入。
		  ○ enabled：允许客户端输入。
		• input_type：Android 输入框的类型。
		• text：输入框文字，只在 Android 平台支持。
		• start_index：表示当前光标在文本中的位置，只在 Android 平台支持。

云手机事件

事件代码	事件含义	事件详细描述
CAI_TRANS_MESSAGE	应用透传消息	当云端应用向客户端发送透传消息时，App 会收到此事件。App 可以根据自己的业务逻辑进行处理。

		事件数据为 JsonObject 类型，包含包名和消息内容。示例数据：
		{
		"package_name": "com.example.myapplication",
		"msg": "test msg"
		}

CAI_SYSTEM_USAGE	云手机负载信息	每10s云端应用向客户端发送的负载信息。App 可以根据自己的业务逻辑进行处理。
		事件数据为 JsonObject 类型，包含 CPU、内存和 GPU 使用率等。示例数据：
		{
		"cpu_usage": 30.1,
		"mem_usage": 7.2,
		"gpu_usage": 12.0
		}

CAI_CLIPBOARD	剪贴板同步	当云手机剪贴板内容发生变化时，App 会收到此事件。App 可以根据自己的业务逻辑进行处理。 事件数据为 JsonObject 类型，包含剪贴板文本内容。示例数据： {"text": "abc"}

CAI_NOTIFICATION	收到通知消息	当云手机上有新通知时，App 会收到此事件。App 可以根据自己的业务逻辑进行处理。
		事件数据为 JsonObject 类型，包含应用包名、通知标题和正文。示例数据：
		{
		"package_name": "com.tencent.qq",
		"title": "11",
		"text": "22"
		}

CAI_SYSTEM_STATUS	系统状态更新	每1秒云端应用向客户端发送的系统状态。App 可以根据自己的业务逻辑进行处理。
		事件数据为 JsonObject 类型，包含导航栏可见性和媒体音量。示例数据：
		{
		"nav_visible": false,
		"music_volume": 8
		}