

Table of Contents

Rokid Mobile SDK	1.1
修订记录	1.2
快速接入	1.3
集成方式	1.3.1
初始化	1.3.2
调试	1.3.3
Demo 代码	1.3.4
帐号模块 Account	1.4
账号模块介绍	1.4.1
若琪账号体系	1.4.2
自有账号体系	1.4.3
账号登出	1.4.4
用户信息	1.4.5
绑定模块 Binder	1.5
绑定模块介绍	1.5.1
手机蓝牙状态	1.5.2
设备蓝牙扫描	1.5.3
连接设备蓝牙	1.5.4
设备配网	1.5.5
释放设备蓝牙	1.5.6
设备模块 Device	1.6
设备列表	1.6.1
设备基本信息	1.6.2
设备地理位置	1.6.3
设备昵称	1.6.4
设备系统版本	1.6.5
默认设备	1.6.6
恢复出厂设置	1.6.7
设备解绑	1.6.8
设备昵称前缀	1.6.9
ping设备状态	1.6.10
设备夜间模式	1.6.11
设备自定义信息	1.6.12

技能模块 Skill	1.7
闹钟	1.7.1
提醒	1.7.2
智能家居	1.7.3
技能商店	1.7.4
WebBridge	1.8
快速接入	1.8.1
Vui 反馈 模块	1.9
Card 列表	1.9.1
Card 消息	1.9.2
Card 样式说明	1.9.3
发送 ASR	1.9.4
发送 TTS	1.9.5
发送 Topic消息	1.9.6
消息 Event	1.10
Event 介绍	1.10.1
通用消息	1.10.2
帐号相关消息	1.10.3
设备相关信息	1.10.4
内容相关信息	1.10.5
自定义消息	1.10.6
错误码	1.11
FAQ	1.12
服务协议	1.13
免责声明	1.14
Rokid 官方讨论区	1.15

Rokid Mobile SDK 介绍

什么是 Rokid Mobile SDK

Rokid Mobile SDK 提供了整套的移动开发支持，能让开发者的移动应用拥有对搭载 Rokid服务的设备配网、指令发送、技能控制、设备管理等交互能力。

目前支持手机系统有：iOS（苹果）、Android（安卓）。

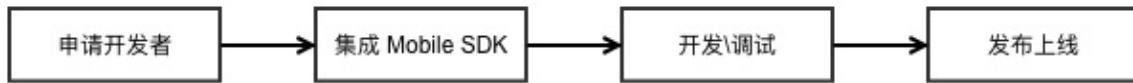
文档介绍

本文档提供了 Rokid Mobile SDK Android 版的接入流程、权限申请、SDK 集成方式、SDK 初始化、各个模块的API说明以及Demo代码。

本文档统一称 Rokid Mobile SDK 为 Mobile SDK

注：本文档 适用与 Mobile SDK 1.3.1 版本。

接入流程



1、申请开发者

需先在[开放平台](#)注册一个帐号，并联系 对接的项目经理 申请 AppKey、AppSecret 和 accessKey。

2、集成 Mobile SDK

[Mobile SDK 集成说明](#)

3、开发调试

可以使用如下开关，进入测试环境和调试功能：

```
RokidMobileSDK.debug();
```

4、发布上线

开发完成并且进行严格测试的 APP，即可正常上线使用。

支持

Rokid Mobile SDK Android Issue

版本更新信息

- v1.3.1

[新增] 账号模块 - 若琪自有账号体系 注册相关接口。

[修改] 账号模块 - 开发者自有账号体系 登录接口。

- v1.3.0

[新增] 绑定模块 - 蓝牙 2.0 配网协议，配网过程中 设备状态的判断

- v1.2.1

[修复] 断网后，长连接没有正常连接问题。 [新增] SDKChannelMessage 长连接全部消息。

[新增] 设备自定义信息存储、获取

- v1.2.0

[优化] SDK 长连接 稳定性

- v1.0.11

[新增] SDKMediaEvent 内容播放消息

- v1.0.9

[新增] Vui sendMessage 接口

- v1.0.8

[新增] Bridge goBack 接口

[优化] Bridge 加载速度，SDKWebClient 改为 SDKWebChromeClient

[修复] Skill 闹钟 ext 字段丢失

[修改] Account token login [新增] Skill-闹钟\提醒 获取列表 回调

[优化] Skill-闹钟\提醒 重复模式 解释

- v1.0.7

[新增] Web Bridge 接入说明

- v1.0.6

[新增] 闹钟、提醒 skill 接口

- v1.0.5

[新增] DeviceTypeId、Token、Debug

- v1.0.4

[新增] 用户单独登录、系统更新 Event

- v1.0.3

[删除] 刷新 Token 接口
[新增] Device 一些接口的返回值

- v1.0.2

[新增] Token 登录
[新增] 刷新 Token 接口

- v1.0.1

[新增] Card 协议解释

- v1.0.0

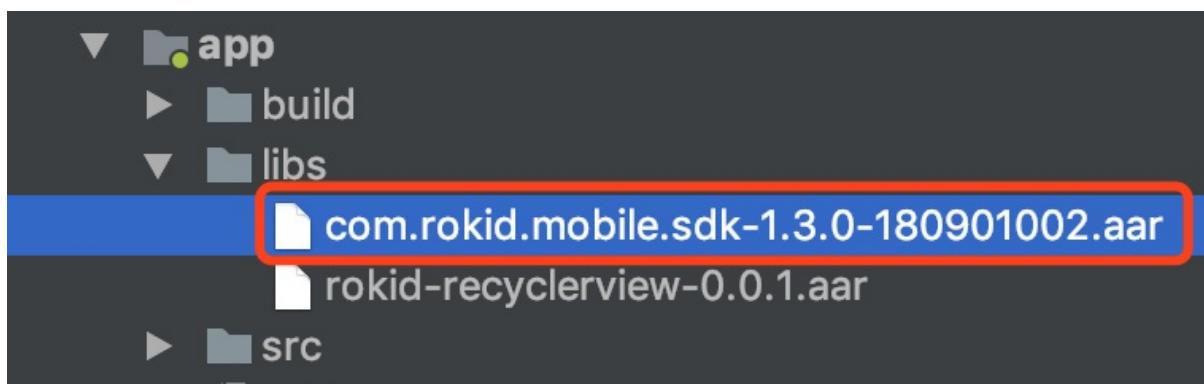
[新增] Rokid Mobile SDK 初版

Mobile SDK 集成方式

导入Mobile SDK

手动导入

1. [下载 Mobile SDK](#)
2. 将 com.rokid.mobile.sdk-XXXXXX.aar 包放入工程的libs目录下，如下图



1. 在工程的 App Module 的 build.gradle 下手动添加依赖，如下代码所示：

```
repositories {  
    flatDir {  
        dirs 'libs'  
    }  
}  
  
dependencies {  
    ....  
    compile(name: 'com.rokid.mobile.sdk-1.3.0-180901002', ext: "aar")  
    ....  
}
```

Gradle 集成

目前不支持，请敬请期待

第三方库依赖

Rokid Mobile SDK 目前需要依赖以下 第三方开源库，请添加到自己工程中：

```
compile 'com.google.code.gson:gson:2.8.0'  
compile 'com.squareup.okhttp3:okhttp:3.9.0'  
compile 'org.greenrobot:eventbus:3.0.0'  
compile 'org.greenrobot:greendao:3.2.0'
```

```
compile 'org.greenrobot:greendao-generator:3.2.0'  
compile 'org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.2.0'  
compile 'org.eclipse.paho:org.eclipse.paho.android.service:1.1.1'  
compile 'com.google.protobuf:protobuf-java:3.5.1'
```

权限依赖

```
<uses-permission android:name="android.permission.INTERNET"/>  
<uses-permission android:name="android.permission.BLUETOOTH"/>  
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>  
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>  
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>  
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>  
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>  
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>  
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>  
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

添加混淆规则

在工程的 proguard-project.txt 里添加以下相关规则：

```
-keep class com.rokid.mobile.** { *; }
```

Mobile SDK 初始化

初始化

使用 Mobile SDK 之前必须要先初始化SDK，否则无法正常使用 Mobile SDK。

注意：

1. 在工程的Application类的onCreate()方法中初始化RokidMobileSDK。
2. appKey、appSecret、accessKey 请按照真实填写，否则会初始化失败。
3. appKey、appSecret、accessKey 需要向 对接的 Rokid 项目经理 申请。

参数说明：

字段	类型	必须？	说明
context	Context	是	Context
appKey	String	是	Rokid 发放的 appKey
appSecret	String	是	Rokid 发放的 appSecret
accessKey	String	是	Rokid 发放的 accessSecret

示例代码：

```
RokidMobileSDK.init(context, appKey, appSecret, accessKey, new InitCompletedCallback {
    @Override
    public void onInitSuccess() {
        Logger.d("onInitSuccess is called.");
        ... // do something
    }

    @Override
    public void onInitFailed(String errorCode, String errorMsg) {
        Logger.e("onInitFailed errorCode=" + errorCode + " errorMsg=" + errorMsg);
        ... // do something
    }
});
```

Mobile SDK 调试

开发者可以 切换到 测试环境， 并打印一些日志用于分析问题。

测试环境

调用这个 API 可以切换到 测试环境， 进行调试开发。

示例代码：

```
RokidMobileSDK.debug();
```

Mobile SDK Demo 代码

介绍

Mobile SDK Demo 主要提供 Mobile SDK API 使用示例。

GitHub 地址：

<https://github.com/Rokid/RokidMobileSDKAndroidDemo>

账号模块

介绍

账号模块 提供了两套账号体系，供开发者使用。

注意：两个账号体系不能同时使用。

- 1. 若琪账号体系
- 2. 自有账号体系

若琪账号体系

开发者 可以直接使用 Rokid 账号体系，进行用户账号注册、登录 等操作。

[若琪账号体系 使用说明](#)

自有账号体系

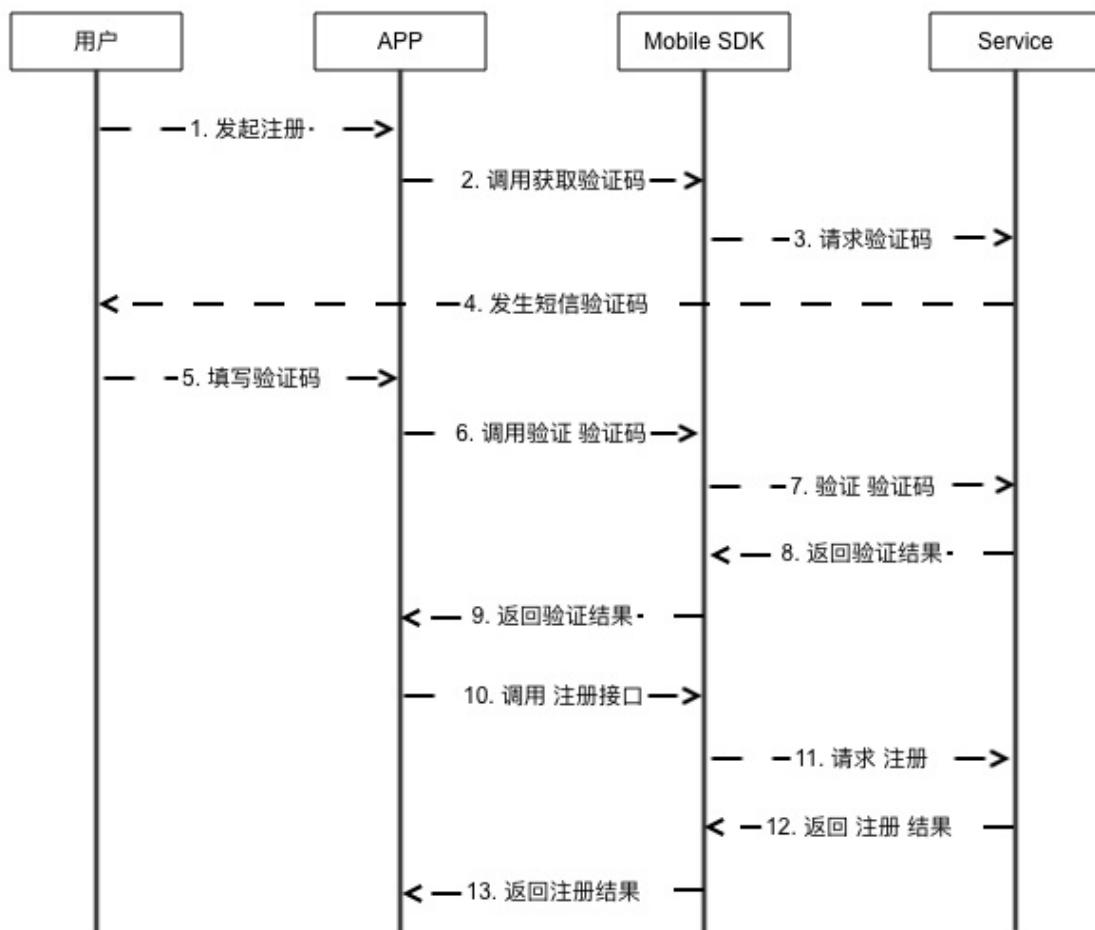
如开发者已经有自己的账号体系了，可以直接使用。

[自有账号体系 使用说明](#)

若琪账号体系

注册流程

1、流程



2、接口

1、获取验证码

参数说明:

字段	类型	必须?	说明
phoneNum	String	是	手机号

举个大栗子:

```
RokidMobileSDK.account.getScode(phoneNum, new IGetScodeResultCallback() {
    @Override
```

```

public void onGetScodeSucceed() {
    Logger.d("onGetScodeSucceed is called.");
    ... // do something
}

@Override
public void onGetScodeFailed(String errorCode, String errorMsg) {
    Logger.e("onGetScodeFailed errorCode=" + errorCode + " errorMsg=" + errorMsg);
    ... // do something
}
);

```

2、验证验证码

参数说明:

字段	类型	必须?	说明
phoneNum	String	是	手机号
scode	String	否	验证码

举个大栗子:

```

RokidMobileSDK.account.checkScode(scode, phoneNum, new ICheckScodeResultCallback()
{
    @Override
    public void onCheckScodeSucceed(String newCode) {
        Logger.d("onCheckScodeSucceed is called.");
        ... // do something
    }

    @Override
    public void onCheckScodeFailed(String errorCode, String errorMsg) {
        Logger.e("onCheckScodeFailed errorCode=" + errorCode + " errorMsg=" + errorMsg);
        ... // do something
    }
});

```

3、注册账号

参数说明:

字段	类型	必须?	说明
phoneNum	String	是	手机号
passwd	String	否	密码
scode	String	否	验证码

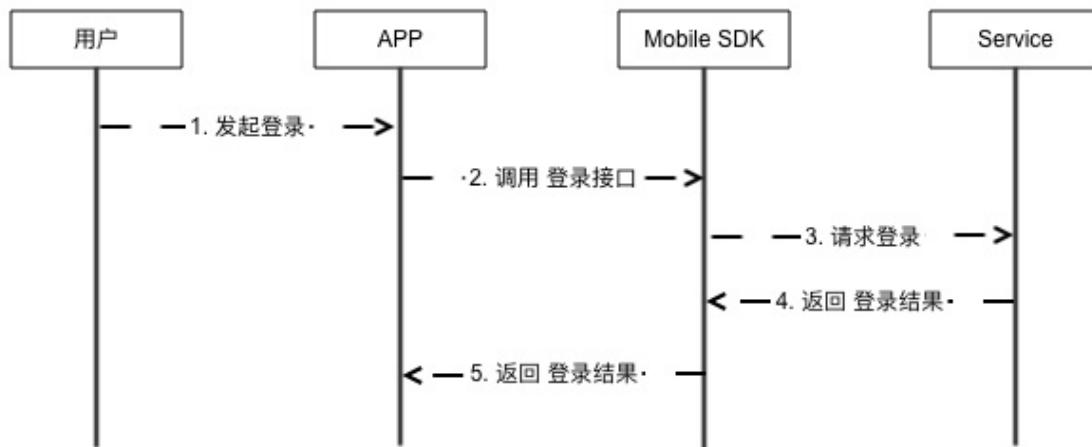
举个大栗子：

```
RokidMobileSDK.account.register(phoneNum, passwd, scode, new IRegisterResultCallback() {
    @Override
    public void onRegisterSucceed() {
        Logger.d("onRegisterSucceed is called.");
        ... // do something
    }

    @Override
    public void onRegisterFailed(String errorCode, String errorMsg) {
        Logger.e("onRegisterFailed errorCode=" + errorCode + " errorMsg=" + errorMsg);
        ... // do something
    }
});
```

登录流程

1、流程



2、接口

参数说明

字段	类型	必须?	说明
phoneNum	String	是	手机号
passwd	String	否	密码

举个大栗子：

Java:

```
RokidMobileSDK.account.login(phoneNum, passwd, new ILoginResultCallback() {  
    @Override  
    public void onLoginSucceed() {  
        Logger.d("onLoginSuccess is called.");  
        ... // do something  
    }  
  
    @Override  
    public void onLoginFailed(String errorCode, String errorMsg) {  
        Logger.e("onLoginFailed errorCode=" + errorCode + " errorMsg=" + errorMsg);  
        ... // do something  
    }  
});
```

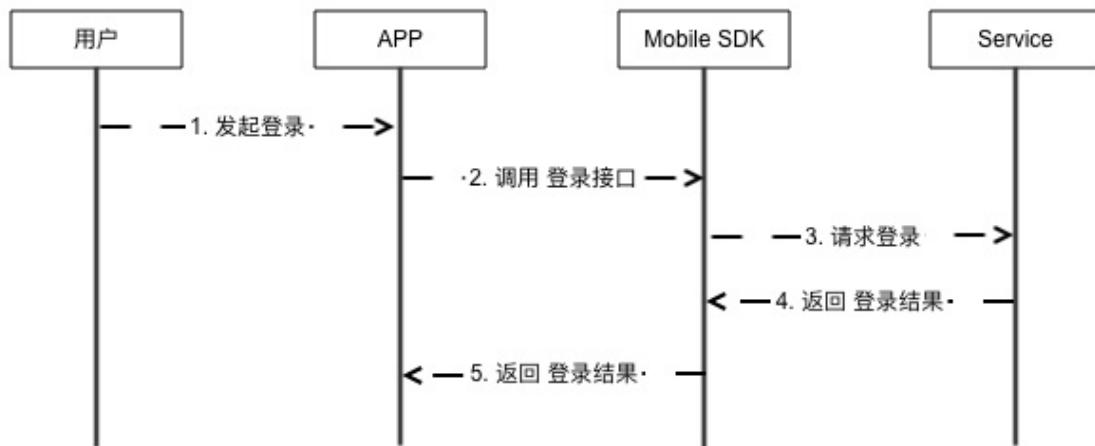
自有帐号体系

注意：

使用自有帐号体系时，开发者必须拥有自己的帐号体系，如果没有请直接使用若琪帐号体系，谢谢。

登录流程

1、流程



2、接口

参数说明：

字段	类型	必须?	说明
userId	String	是	SDK接入方用户Id
token	String	否	SDK接入方用户token

举个大栗子：

Java：

```

RokidMobileSDK.account.thirdpartyLogin(userId, token, new ILoginResultCallback() {
    @Override
    public void onLoginSucceed() {
        Logger.d("onLoginSuccess is called.");
        ... // do something
    }

    @Override
    public void onLoginFailed(String errorCode, String errorMsg) {
        Logger.e("onLoginFailed errorCode=" + errorCode + " errorMsg=" + errorMsg);
    }
}

```

```
    ... // do something  
}  
});
```

登出账号 Logout

登出

接口说明：登出接口，清除 SDK 中的 token、userId 等缓存数据。

举个大栗子：

```
RokidMobileSDK.account.logout(new ILogoutResultCallback() {
    @Override
    public void onLogoutSucceed() {
        Logger.d("onLogoutSucceed is called.");
        ... // do something
    }

    @Override
    public void onLogoutFailed(String errorCode, String errorMsg){
        Logger.e("onLogoutFailed errorCode=" + errorCode + " errorMsg=" + errorMsg)
    ;
        ... // do something
    }
});
```

帐号信息 Account Info

Token

接口说明：获取用户登录的 Token

举个大栗子：

```
String token = RokidMobileSDK.account.getToken();
```

UserId

接口说明：获取用户登录的 UserId

举个大栗子：

```
String userId = RokidMobileSDK.account.getUserId();
```

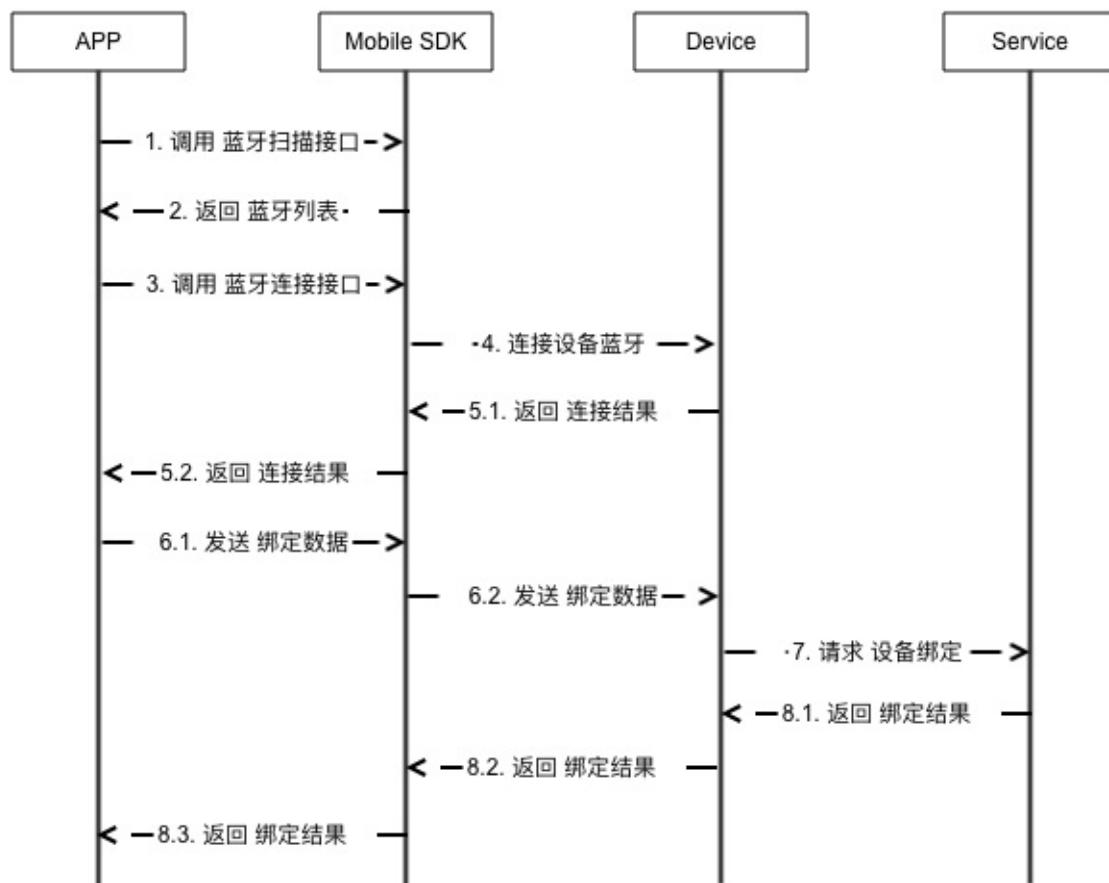
绑定模块

介绍

绑定模块 主要提供给 搭载了 Rokid服务的设备进行配置网络环境。

目前使用蓝牙进行数据传输， 请确认 APP 是否开通蓝牙权限。

配网流程



绑定模块 Binder

查询蓝牙状态

接口说明

查询手机蓝牙状态，是否打开。

举个大栗子：

```
boolean btStatus = RokidMobileSDK.binder.getBTStatus()
```

注册蓝牙状态监听器

接口说明

监听手机蓝牙状态发生改变，手机蓝牙打开 或者 关闭都会调用这个函数。

举个大栗子：

```
RokidMobileSDK.binder.registerBTStateChangeListener(new IBTStateChangeListener() {
    @Override
    public void onBluetoothStateChanged(boolean isOpen) {
        Logger.d("onBluetoothStateChanged ble state=" + isOpen);

        if (!isOpen) {
            // 蓝牙状态变成关
            ... // doSomeThing
            return;
        }

        // 蓝牙状态变成开
        ... // doSomeThing
    }
});
```

设备蓝牙扫描

扫描设备

介绍

需要传入扫描蓝牙设备的名称的前缀，回调均在主线程。

如果传空，是无法获取设备列表。

1、单前缀蓝牙设备

参数说明

字段	类型	必须?	说明
type	String	是	设备名称类型前缀

举个大栗子

```
String type = "Rokid-"
RokidMobileSDK.binder.startBTScan(type, new IBTScanCallBack() {
    @Override
    public void onNewDevicesFound(BTDeviceBean btDeviceBean) {
        if(null == btDeviceBean){
            Logger.w("BTScanCallBack", "newDevicesFound btDeviceBean is null");
            return;
        }

        Logger.i("BTScanCallBack", "newDevicesFound device Name=" + btDeviceBean.getName())
        Logger.i("BTScanCallBack", "newDevicesFound device Address=" + btDeviceBean.getAddress())

        ... // doSomeThing
    });
}
```

2、多前缀蓝牙设备

参数说明

字段	类型	必须?	说明
types	String[]	是	设备名称类型前缀列表

举个大栗子

```
String[] types = {"Rokid-Pebble-", "Rokid-Mini-"}  
RokidMobileSDK.binder.startBTScan(types, new IBTScanCallBack() {  
    @Override  
    public void onNewDevicesFound(BTDeviceBean btDeviceBean) {  
        if(null == btDeviceBean){  
            Logger.w("BTScanCallBack", "newDevicesFound btDeviceBean is null");  
            return;  
        }  
  
        Logger.i("BTScanCallBack", "newDevicesFound device Name=" + btDeviceBean.getName())  
        Logger.i("BTScanCallBack", "newDevicesFound device Address=" + btDeviceBean.getAddress())  
  
        ... // doSomeThing  
    });
```

停止蓝牙扫描

接口说明

仅停止蓝牙扫描

举个大栗子

```
RokidMobileSDK.binder.stopBTScan()
```

停止扫描并清除设备列表

接口说明

停止扫描并且清除底层内存上的设备列表，建议在刷新的时候调用

举个大栗子

```
RokidMobileSDK.binder.stopBTScanAndClearList()
```

蓝牙连接

连接蓝牙设备

接口说明 接口需传入蓝牙名称（蓝牙address重启后会变）

参数说明

字段	类型	必须？	说明
name	String	是	设备名称

举个大栗子

```
RokidMobileSDK.binder.connectBT(name, new IBTConnectCallBack() {
    @Override
    public void onConnectSucceed(BTDeviceBean btDeviceBean) {
        Log.i("BTConnectCallBack", "connectBT Success name=" + btDeviceBean.getName());
        Log.i("BTConnectCallBack", "connectBT Success address=" + btDeviceBean.getAddress());
        ... // doSomeing
    }

    @Override
    public void onConnectFailed(BTDeviceBean btDeviceBean, BleException bleException) {
        Log.e("BTConnectCallBack", "connectBT Failed name=" + btDeviceBean.getName());
        Log.e("BTConnectCallBack", "connectBT Failed address=" + btDeviceBean.getAddress());
        Log.e("BTConnectCallBack", "connectBT Failed Exception=" + bleException.toString());
        ... // doSomeing
    }
});
```

发送配网数据

设备配网

接口说明

发送绑定数据

这里会发送到正在连接的蓝牙设备

参数说明

字段	类型	必须?	说明
binderData	DeviceBinderData	是	蓝牙发送信息

举个大栗子

```
// 构建绑定数据
DeviceBinderData binderData = DeviceBinderData.newBuilder()
    .wifiPwd("your wifiPwd")           //wifi密码 (可以为空)
    .wifiSsid("your wifiSsid")         //wifi名字 (可以为空)
    .wifiBssid("your wifiBssid")       //wifi地址 (可以为空)
    .build();

// 发送数据
RokidMobileSDK.binder.sendBTBinderData(binderData, new IBinderCallBack() {
    @Override
    public void sendSuccess(BTDeviceBean btDeviceBean) {
        Log.i("BTSendCallBack", "sendBtData Success name=" + btDeviceBean.getName());
    }

    Log.i("BTSendCallBack", "sendBtData Success address=" + btDeviceBean.getAddress());
    ... // doSomeing
}

@Override
public void sendFailed(BTDeviceBean btDeviceBean, BleException bleException) {

    Log.e("BTSendCallBack", "sendBtData failed name=" + btDeviceBean.getName());
    Log.e("BTSendCallBack", "sendBtData failed address=" + btDeviceBean.getAddress());
    Log.e("BTSendCallBack", " sendBtData failed Exception=" + bleException.toString());
    ... // doSomeing
}
});
```

配网状态

设备配网过程中，状态判断。需要使用 EventBus 接收 消息。

消息体

SDKBinderStatusEvent

状态值判断

状态值	说明	备注
10	wifi连接中	
11	wifi连接成功	
-11	wifi密码错误	
-12	wifi连接超时	
-13	没找到当前wifi	
-14	wifi密码长度不正确	
-98	运营商网络错误	
100	登录中	
101	登录成功	
-101	登录失败	
200	绑定中	
201	绑定成功	
-201	绑定失败	

绑定模块 Binder

释放蓝牙资源

接口说明 建议在配网结束后调用

举个大栗子

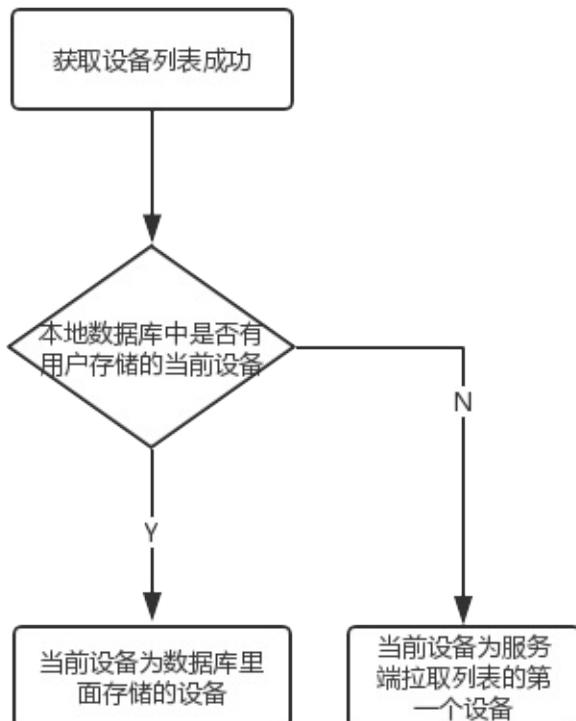
```
RokidMobileSDK.binder.releaseBT()
```

设备模块 Device

获取设备列表

接口说明

目前获取服务端masterId对应的设备列表，
注意 SDKDevice此时里面只有 rokiId, rokidNick, basic_info信息，
底层会默认给用户选择一个当前设备，逻辑图如下：



示例代码

```

RokidMobileSDK.device.getDeviceList(new IGetDeviceCallback() {
    @Override
    public void onGetDeviceListSucceed(List<SDKDevice> deviceList) {
        if(CollectionUtils.isEmpty(devices)){
            // 设备为空
            ... // doSomeing
            return;
        }

        Logger.i("onGetDeviceListSucceed device size="+devices.size());
    }
})
  
```

```

        ... // doSomeing
    }

    @Override
    public void onGetDeviceListFailed(String errorCode, String errorMsg) {
        Logger.e("onGetDeviceListFailed errorCode"+errorCode+", errorMsg="+erro
rMsg );
        ... // doSomeing
    }
});

```

deviceList数据格式：

```

[
{
    //设备昵称
    "deviceNick": "xxx",
    //设备Id
    "deviceId": "0011111111111111",
    "basic_info": {
        //设备地区
        "region": "CN",
        "sn": "02010217020001ED",
        //系统版本号
        "ota": "2.2.2-20171027.091126",
        //mac地址
        "mac": "02:00:00:00:00:00",
        "ip": "183.129.185.66",
        //局域网IP
        "lan_ip": "192.168.1.156",
        //自定义TTS音色
        "ttsList": "[{"name": "111777", "vibrato": 0, "speed": 0, "formant": 0, "tone": 0}, {"name": "234566777777", "vibrato": 4, "speed": 5, "formant": -5, "tone": 5}, {"name": "5555", "vibrato": -5, "speed": 5, "formant": -5, "tone": -5}]",
        //系统当前选择TTS音色
        "tts": "{\"formant": -3, \"speed": -2, \"tone": 1, \"ttsName\": \"蜡笔小新\"}",
        // 设备类型ID
        "device_type_id": "XXXX"
    }
}
]

```

设备模块 Device

获取设备基本信息

接口说明 获取 设备基本信息包括： ip、局域网ip、mac、nick、cy、sn、version、device_type_id

参数说明

字段	类型	必须?	说明
deviceId	String	是	设备ID

示例代码：

```
BasicInfo basicInfo = RokidMobileSDK.device.getBasicInfo(deviceId);
```

设备模块 Device

获取设备地址位置

接口说明 获取设备的loaction信息

参数说明

字段	类型	必须?	说明
deviceId	String	是	设备ID

示例代码

```
RokidMobileSDK.device.getLocation(deviceId, new IGetLocationCallback() {
    @Override
    public void onGetLocationSucceed(RKDeviceLocation location) {
        // location从来没有设置过 里面的所有字段都为空
        Logger.i("getLocationByDeviceId", "getLocationSuccess location=" + location.toString());
        ... // doSomeing
    }

    @Override
    public void onGetLocationFailed(String errorCode, String errorMsg) {
        Logger.e("getLocationByDeviceId", "getLocationFailed errorCode=" + errorCode
                + " errorMsg=" + errorMsg);
        ... // doSomeing
    }
});
```

RKDeviceLoaction数据格式：

```
{
    //邮政编码
    "postalCode": "310023",
    //街道
    "street": "访溪路",
    //身份
    "province": "浙江省",
    //经度
    "longitude": "120.04377191503957",
    //纬度
    "latitude": "30.25401732974572",
    //地区
    "district": "余杭区",
    "locationFrom": "COORDINATES",
    //城市
```

```

    "city": "杭州市",
    //国家
    "country": "中国",
    "ip": "183.129.185.66"
}

```

修改设备地址位置

接口说明 更新设备的 RKDeviceLoaction 信息, RKDeviceLocation 传入更新过的RKDeviceLoaction 对象

参数说明

字段	类型	必须?	说明
deviceId	String	是	设备ID
location	RKDeviceLocation	是	设备位置信息

示例代码

```

// Location构建示例
RKDeviceLocation location = RKDeviceLocation.newBuilder()
    .country("中国")                                //国家
    .province("浙江省")                            //省份
    .city("杭州市")                                //城市
    .district("余杭区")                            //区
    .street("访溪路")                                //街道
    .longitude("120.04377191503957") //经度
    .latitude("30.25401732974572") //维度
    .ip("183.129.185.66")                                //设备ip地址
    .postalCode("310023")                            //邮政编码
    .build();

// 更新位置信息
RokidMobileSDK.device.updateLocation(deviceId, location, new IUpdateLocationCallback() {
    @Override
    public void onUpdateLocationSucceed(Location location) {
        Logger.i("onUpdateLocationSuccess location=" + location.toString());
        ... // doSomeing
    }

    @Override
    public void onUpdateLocationFailed(String errorCode, String errorMsg) {
        Logger.e("onUpdateLocationFailed errorCode=" + errorCode + " errorMsg=" + errorMsg);
        ... // doSomeing
    }
})

```

```
});
```

设备模块 Device

更新设备的昵称

接口说明 更新当前设备的昵称

参数说明

字段	类型	必须?	说明
deviceId	String	是	设备ID
newNick	String	是	新设备名称

示例代码:

```
RokidMobileSDK.device.updateNick(deviceId, newNick, new IUpdateNickNameCallback() {
    @Override
    public void onUpdateNickNameSucceed(String nickName) {
        Log.i("updateNick", "onUpdateNickNameSucceed nickName=" + nickName);
        ... // doSomeing
    }

    @Override
    public void onUpdateNickNameFailed(String errorCode, String errorMsg) {
        Log.e("updateNick", "onUpdateNickNameFailed errorCode=" + errorCode + " error
msg=" + errorMsg);
        ... // doSomeing
    }
});
```

设备模块 Device

获取系统版本信息

接口说明

1. 获取当前设备的系统版本信息 返回的消息将以Event的形式向上抛。
2. 返回值 只表示 获取系统版本信息是否已经发送成功。

参数说明

字段	类型	必须?	说明
deviceId	String	是	设备ID

示例代码:

Kotlin

```
RokidMobileSDK.device.getVersionInfo(deviceId, object : IChannelPublishCallback {
    override fun onSucceed() {
        // TODO
    }

    override fun onFailed() {
        // TODO
    }
})
```

event名称

EventDeviceSysUpdate

event数据格式

```
{
    "from": "deviceId",
    "to": "userId",
    {
        //新版本的文案
        "changelog": "新版本文案",
        //0代表已经是最新版本 1, 代表有新版本可以升级
        "checkCode": 0/1,
        //下载进度
        "downloadProgress": 96,
        //已经下载的大小
        "downloadedSize": 350407968,
        //已经
        "totalSize": 361678140,
```

```

    //true代表升级包下载完毕，可以升级，false代表正在下载
    "updateAvailable": true,
    "url": "http://cnpkg.rokidcdn.com/rokid-os/file/171121194749.zip",
    //版本号
    "version": "2.2.3-20171121.190617"
}
}

```

开始系统升级

接口说明

1. 设备开始下载新的镜像 返回的消息和系统版本信息一致
2. 返回值 只表示 获取系统版本信息是否已经发送成功。

参数说明

字段	类型	必须?	说明
deviceId	String	是	设备ID

示例代码:

Kotlin

```

RokidMobileSDK.device.startSystemUpdate(deviceId, object : IChannelPublishCallback
{
    override fun onSucceed() {
        // TODO
    }

    override fun onFailed() {
        // TODO
    }
})

```

返回值说明

字段	类型	说明
sendSuccess	boolean	发送是否成功

注：设备下载镜像成功后会重启，底层与设备的长连接会断，之后的这些接口将获取不到信息

设备模块 Device

设置默认设备

接口说明

更新当前选择设备

参数说明

字段	类型	必须?	说明
device	SDKDevice	是	若琪设备实体

示例代码:

```
RokidMobileSDK.device.setCurrentDevice(device);
```

获取默认设备

接口说明

获取当前选择的设备。

示例代码:

```
SDKDevice device = RokidMobileSDK.device.getCurrentDevice();
```

设备模块 Device

恢复设备的出厂设置

接口说明

恢复设备的出厂设置

示例代码:

Kotlin

```
RokidMobileSDK.device.resetDevice(deviceId, object : IChannelPublishCallback {
    override fun onSucceed() {
        // TODO
    }

    override fun onFailed() {
        // TODO
    }
})
```

设备模块 Device

解绑设备

接口说明

解绑设备

参数说明

字段	类型	必须?	说明
deviceId	String	是	设备ID

示例代码:

```
RokidMobileSDK.device.unbindDevice(deviceId, new IUnbindDeviceCallback() {
    @Override
    public void onUnbindDeviceSucceed() {
        Log.i("unbinderDevice","onUnbindDeviceSuccess rokidId=" + rokiId);
        ... // doSomeing
    }

    @Override
    public void onUnbindDeviceFailed(String errorCode, String errorMsg) {
        Log.i("unbinderDevice","onUnbindDeviceFailed errorCode=" + errorCode
+ " errorMsg=" + errorMsg + " rokidId=" + rokiId);
        ... // doSomeing
    }
});
```

设备模块 Device

设置设备的昵称前缀

接口说明

给新添加的设备设置一个默认的昵称前缀，如：Pebble

参数说明

字段	类型	必须?	说明
nickPrefix	String	是	昵称前缀

示例代码：

```
RokidMobileSDK.device.setInitDeviceNickPrefix(nickPrefix);
```

注：底层会截取该设备的deviceId后3位，再根据您设置的默认昵称前缀组成该设备的默认昵称，如：
Pebble123。

设备模块 Device

ping设备

接口说明

获取当前设备的在线状态

参数说明

字段	类型	必须?	说明
sddDevice	SDKDevice	是	SDKDevice

示例代码:

```
RokidMobileSDK.device.pingDevice(sddDevice, new IPingDeviceCallback() {

    @Override
    public void onSuccess(String deviceId, boolean isOnline) {
        showToast("获取设备状态成功,deviceId=" + deviceId + ",isOnline=" + isOnline);
    }

    @Override
    public void onFailed(String deviceId, String errorCode, String errorMsg)
    {
        showToast("获取设备状态失败, deviceId" + deviceId + ",errorCode=" + errorCode + "errorMsg= " + errorMsg);
    }
});
```

设备模块 Device

获取设备夜间模式

接口说明

获取设备夜间模式

参数说明

字段	类型	必须?	说明
deviceId	String	是	设备ID

示例代码:

```
String deviceId = "XXXXXX";
RokidMobileSDK.device.getNightMode(deviceId, new IGetDeviceNightMode() {
    @Override
    public void onGetDeviceNightModeSucceed(NightModeBean nightModeBean) {
        // 获取成功
    }

    @Override
    public void onGetDeviceNightModeFailed(String errorCode, String errorMessage) {
        // 获取失败
    }
});
```

更新设备夜间模式

接口说明

更新设备夜间模式

参数说明

字段	类型	必须?	说明
deviceId	String	是	设备ID

NightModeBean 对象结构如下:

字段	类型	必须?	说明
action	String	是	open: 打开夜间模式 close:

startTime	String	是	设备ID
endTime	String	是	设备ID

示例代码：

```
String deviceId = "XXXXXX";

NightModeBean nightModeBean = new NightModeBean();
nightModeBean.setAction("open")
nightModeBean.setStartTime("23:00")
nightModeBean.setEndTime("7:00")

RokidMobileSDK.device.updateNightMode(deviceId, nightModeBean, new IUpdateCustomInfoCallback() {
    @Override
    public void onUpdateDeviceNightModeSucceed() {
        callback.onUpdateDeviceNightModeSucceed();
    }

    @Override
    public void onUpdateDeviceNightModeFailed(String errorCode, String errorMsg) {
        callback.onUpdateDeviceNightModeFailed(errorCode, errorMsg);
    }
});
```

设备配置信息 Device

获取设备配置信息

根据业务需求获取设备 配置的各种信息。

参数说明

字段	类型	必须	备注
deviceId	String	是	设备Id
namespace	String	是	存储空间、根据业务填写
key	String	是	信息存储Key

示例代码:

Kotlin

```

val deviceId: String = "XXX"
val namespace: String = "YYY"
val key: String = "ZZZ"

RokidMobileSDK.device.getServiceInfo(deviceId, namespace, key,
    object : SDKServiceInfoCallback{
        override fun onSuccess(result: String?) {
            // TODO ...
        }

        override fun onFailure(errorCode: String?, errorMsg: String?) {
            // TODO ...
        }
})

```

存储、更新设备信息

字段	类型	必须	备注
deviceId	String	是	设备Id
namespace	String	是	存储空间、根据业务填写
kvMapJsonString	String	是	信息存储信息键值对 转换成的 jsonString

```

val deviceId: String = "XXX"
val namespace: String = "YYY"
val kvMapJsonString: String = "ZZZ"

```

```
RokidMobileSDK.device.storeServiceInfo(deviceId, namespace, kvMapJsonString, object
: SDKServiceInfoCallback {
    override fun onSuccess(result: String?) {
    }

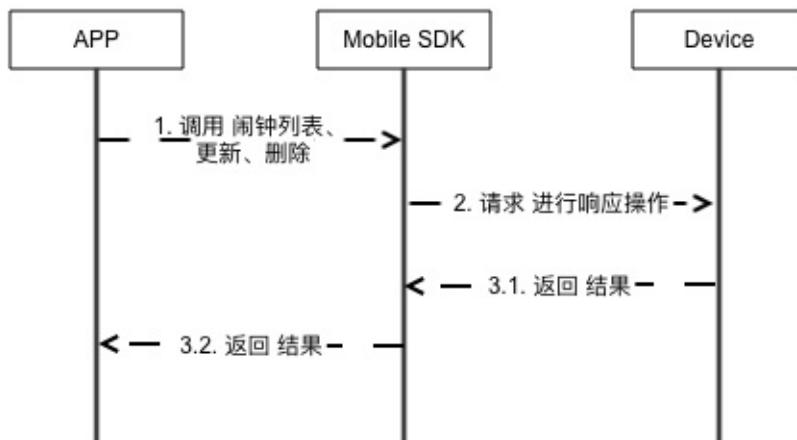
    override fun onFailure(errorCode: String?, errorMsg: String?) {
    }

})
```

技能模块 Skill

闹钟 Alarm

流程



获取闹钟列表

请求获取设备上的闹钟列表：

Java:

```

RokidMobileSDK.skill.alarm().getList(deviceId, new GetAlarmListCallback {
    @Override
    public void onSucceed(List<SDKAlarm> alarmList) {
        //...
    }

    @Override
    public void onFailed(String errorCode, String errorMessage) {
        //...
    }
})
  
```

Kotlin:

```

RokidMobileSDK.skill.alarm().getList(skill_alarm_device_id.selectedItem.toString(),

object : GetAlarmListCallback {
    override fun onSucceed(alarmList: MutableList<SDKAlarm>?) {
        // ...
    }
})
  
```

```

    }

    override fun onFailed(errorCode: String?, errorMessage: String?) {
        // ...
    }
})

```

SDKAlarm 字段说明：

参数	类型	必要？	说明
id	int	是	闹钟Id
year	int	是	年
month	int	是	月
day	int	是	日
hour	int	是	小时
minute	int	是	分钟
repeatType	String	是	重复模式
repeatText	String	是	重复模式的文案
ext	Map	是	扩展字段，根据自己业务进行扩展

注：目前只有Lua版Linux系统支持该字段 ext字段是手机App与系统通信特有的字段，添加或修改时传入，获取列表时原样返回，以下划线_开始的key是预定义key。ext字段可以为空；因为暂时没有删除字段的接口，所以修改时(SpecificTime)需要传入所有的key和value。系统不支持时间完全相同的闹钟，所以更新和删除时不会校验ext是否匹配。

名称	类型	描述
_ringtone	string	闹钟铃声地址，会覆盖全局的闹钟主题

第三方需求可以由他们自定义字段，比如小雅小雅的标签需求

添加闹钟

添加一个闹钟。

Java:

```

SDKAlarm alarm = SDKAlarm.builder()
    .year(2018)
    .month(3)
    .day(3)
    .hour(14)
    .minute(30)
    .repeatType(SDKRepeatType.EVERY_MONDAY)

```

```
.build();

RokidMobileSDK.skill.alarm().add(deviceId, alarm);
```

Kotlin:

```
val sdkAlarm = SDKAlarm().apply {
    year = 2018
    month = 3
    day = 3
    hour = 14
    minute = 30
    repeatType = SDKRepeatType.EVERY_MONDAY
}

RokidMobileSDK.skill.alarm().add(deviceId!!, sdkAlarm)
```

SDKRepeatType 解释:

```
SDKRepeatType.ONCE // 仅此一次
SDKRepeatType.EVERYDAY // 每天
SDKRepeatType.WEEKDAY // 工作日
SDKRepeatType.WEEKEND // 每周末
SDKRepeatType.EVERY_MONDAY // 每周一
SDKRepeatType.EVERY_TUESDAY // 每周二
SDKRepeatType.EVERY_WEDNESDAY // 每周三
SDKRepeatType.EVERY_THURSDAY // 每周四
SDKRepeatType.EVERY_FRIDAY // 每周五
SDKRepeatType.EVERY_SATURDAY // 每周六
SDKRepeatType.EVERY_SUNDAY // 每周日
```

删除一个闹钟

删除一个闹钟:

Java:

```
RokidMobileSDK.skill.alarm().delete(deviceId, alarm, new IChannelPublishCallback()
{
    @Override
    public void onSucceed() {
        // TODO
    }

    @Override
    public void onFailed() {
        // TODO
    }
})
```

```

    }

});
```

Kotlin:

```

RokidMobileSDK.skill.alarm().delete(deviceId, alarm, object : IChannelPublishCallback {
    override fun onSucceed() {
        // TODO
    }

    override fun onFailed() {
        // TODO
    }
})
```

注：字段说明 请参考上面 6.1.1

更新闹钟

更新一个闹钟：

Java:

```

RokidMobileSDK.skill.alarm().update(deviceId, oldAlarm, newAlarm, new IChannelPublishCallback() {
    @Override
    public void onSucceed() {
        // TODO
    }

    @Override
    public void onFailed() {
        // TODO
    }
});
```

Kotlin:

```

RokidMobileSDK.skill.alarm().update(deviceId, oldAlarm, newAlarm, object : IChannelPublishCallback {
    override fun onSucceed() {
        // TODO
    }
})
```

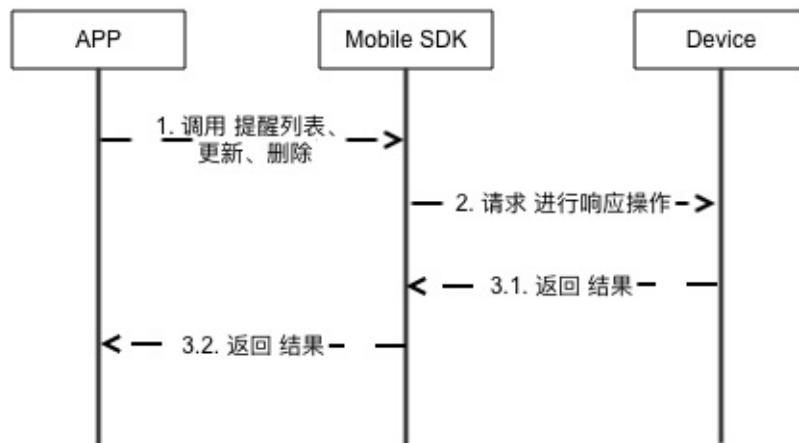
```
override fun onFailed() {  
    // TODO  
}  
  
})
```

注：字段说明 请参考上面 1 和 2

技能模块 Skill

提醒 Remind

流程



提醒列表

请求获取设备上的提醒列表：

Java:

```

RokidMobileSDK.skill.remind().getList(deviceId, new GetRemindListCallback {
    @Override
    public void onSucceed(List<SDKRemind> remindList) {
        // ...
    }

    @Override
    public void onFailed(String errorCode, String errorMessage) {
        // ...
    }
})
  
```

Kotlin:

```

RokidMobileSDK.skill.remind().getList(deviceId, object : GetRemindListCallback {
    override fun onSucceed(remindList: MutableList<SDKRemind>?) {
        // ...
    }
})
  
```

```

        override fun onFailed(errorCode: String?, errorMessage: String?) {
            //...
        }
    })
}

```

SDKRemind 字段说明：

参数	类型	必要？	说明
id	int	是	提醒Id
year	int	是	年
month	int	是	月
day	int	是	日
hour	int	是	小时
minute	int	是	分钟
repeatType	String	是	重复模式
repeatText	String	是	重复模式文案
content	String	是	提醒内容
ext	Map	是	扩展字段，根据自己业务进行扩展

删除一个提醒

删除一个提醒：

Java:

```

RokidMobileSDK.skill.remind().delete(deviceId, remind, new IChannelPublishCallback(
) {
    @Override
    public void onSucceed() {
        // TODO
    }

    @Override
    public void onFailed() {
        // TODO
    }
});

```

Kotlin:

```

RokidMobileSDK.skill.remind().delete(deviceId, remind, object : IChannelPublishCall

```

```
back {  
    override fun onSucceed() {  
        // TODO  
    }  
  
    override fun onFailed() {  
        // TODO  
    }  
}  
)
```

注：字段说明 请参考上面 1

技能模块 Skill

智能家居 H5

介绍

智能家居模块是已 H5 的形式集成到 SDK 中，所以必须实现 SDK webbridge 才能正常打开使用此模块。

URL和环境

- release环境: <https://s.rokidcdn.com/homebase/tob/index.html#/homes/index?theme=default>
Rokid Mobile SDK 环境为 release
- pre环境: <https://s.rokidcdn.com/homebase/tob/pre/index.html#/homes/index?theme=default>
Rokid Mobile SDK 环境为 release
- dev环境: <https://s.rokidcdn.com/homebase/tob/dev/index.html#/homes/index?theme=default>
Rokid Mobile SDK 环境为 Debug

接入说明

请参照 SDKWebbridge 的快速集成文档集成 Bridge，并且必须实现如下方法。

Kotlin:

```
// 响应用户手指按下
override fun touchDown() {
}

// 响应用户手指抬起
override fun touchUp() {
}

// 当前 Webview 打开一个新页面
override fun open(title: String, url: String) {
}

// 在一个新的 Activity (webview) 打开一个新页面
override fun openNewWebView(title: String, url: String) {
}

// 使用 外部浏览器（系统默认浏览器） 打开一个页面
override fun openExternal(url: String) {
}
```

```
// 跳回到 一个页面
override fun goBack(module: String, page: String) {
    if (module == "homebase" && page == "index") {
        // TODO 跳回 智能家居首页，并且清除栈中 其他智能家居页面。
    }
}
```

字段解释：

名称	类型	必须？	描述
module	String	是	固定值：homebase
page	String	是	页面表示： index：智能家居首页

技能模块 Skill

技能商店 H5

URL

- release环境: <https://skill.rokid.com/storev2>

快速接入说明

1、必须继承 SDKWebview

2、必须实现如下方法。

例子:

Kotlin

```
// 当前 Webview 打开一个新页面
override fun open(title: String, url: String) {
}

// 在一个新的 Activity (webview) 打开一个新页面
override fun openNewWebView(title: String, url: String) {
}

// 使用 外部浏览器（系统默认浏览器） 打开一个页面
override fun openExternal(url: String) {
}
```

RKWebBridge

简介

如果需要 接入 智能家居 等一些 H5 页面， 需要接入 RKWebBridge， 否则 H5 页面无法正常使用。

快速接入

我们 提供了 封装好的 SDKWebChromeClient、SDKWebview， 方便开发者集成， 请安装下面 Demo 代码使用即可， 具体 Native UI 组件可根据APP业务需求进行实现。

DemoWebChromeClient:

```
public class DemoWebChromeClient extends SDKWebChromeClient {  
  
    public DemoWebChromeClient(RKWebBridge webBridge) {  
        super(webBridge);  
    }  
  
}
```

DemoWebView:

```
public class DemoWebView extends SDKWebview {  
  
    private void init(Context context) {  
        this.setWebViewClient(new DemowebViewClient(webBridge));  
        // ...  
    }  
  
    // 用户手指按下  
    @Override  
    public void touchDown() {  
    }  
  
    // 用户手指移动  
    @Override  
    public void touchMove() {  
    }  
  
    // 用户手指抬起  
    @Override  
    public void touchUp() {  
    }  
}
```

```
// 关闭当前页面
@Override
public void close() {
}

// 在当前的 webview , 打开Url
@Override
public void open(String title, String url) {
}

// 在一个新的 Activity 中打开Url
@Override
public void openNewWebView(String title, String url) {
}

// 使用外部浏览器 打开Url
@Override
public void openExternal(String url) {
}

// 显示 Toast
@Override
public void showToast(String message) {
}

// 显示 加载中UI组件
@Override
public void showLoading(String message) {
}

// 隐藏 加载中UI组件
@Override
public void hideLoading() {
}

// 设置 标题栏标题
@Override
public void setTitle(String title) {
}

// 设置 标题栏风格
@Override
public void setTitleBarStyle(String style) {
}

// 设置 标题栏 右侧按钮
@Override
public void setTitleBarRight(TitleBarButton titleBarButton){
}

// 设置 标题栏 右侧按钮小红点状态
```

```
@Override  
public void setTitleBarRightDotState(boolean state) {  
}  
  
// 显示 异常UI组件  
@Override  
public void errorView(boolean state, String retryUrl) {  
}  
  
}
```

(1) 标题栏风格：请标题栏风格 根据业务需求设置。

(2) 标题栏 右侧 RightButton 参数解释：

名称	类型	必须？	描述
icon	String	否	按钮图片
text	String	否	按钮标题
loadSelf	Bool	否	默认false：是否在当前页面打开
targetUrl	String	是	close：关闭当前页面</br> \：打开这个url

Vui 反馈 模块

获取Card 列表

默认一次拉取25条

参数说明

字段	类型	必须?	说明
maxDbId	int	是	card的id
pageSize	int	是	分页大小

示例代码:

```
RokidMobileSDK.vui.getCardList(maxDbId, pageSize, new IGetCardsCallback() {
    @Override
    public void onGetCardsSucceed(final List<CardMsgBean> cardInfoList, boolean hasMore) {
        Logger.d("getCardListFromService success ");
        if (CollectionUtils.isEmpty(cardInfoList) || !hasMore) {
            Logger.d("getCardListFromService success but card list is empty or hasMore false");
            // 服务端没有更多数据了
        } else {
            // 拿到cards
        }
    }

    @Override
    public void onGetCardsFailed(String errorCode, String errorMsg) {
        Logger.e("getCardListFromService Failed errorCode=" + errorCode + " ;errorMsg=" + errorMsg);
        // ...
    }
});
```

返回参数说明

字段	类型	说明
cardInfoList	List	card列表，按照时间的顺序排列（最新的card在数组尾端）
hasMore	boolean	服务端是否还有数据

接收卡片消息

流程



接收 Card 消息

监听Card event 消息

示例代码:

```

@Subscribe(threadMode = ThreadMode.BACKGROUND)
public void onReceivedCard(CardMsgBean cardMessage) {
    Logger.d("Receiver the Card message event: ", cardMessage.toString());
    if (TextUtils.isEmpty(cardMessage.getMsgTxt())) {
        Logger.e("This card message is invalid.");
        return;
    }
    // .....
}
  
```

消息的格式:

```

{
    //card Id
    "dbId": 0,
    //应用的appid
    "from": "E33FCE60E7294A61B84C43C1A171DFD8",
    //用户的id
    "to": "userId",
    //时间戳
    "msgStamp": "Dec 14, 2017 4:22:24 PM",
    //消息内容 (5.4会详细说明)
    "msgTxt": "{\"type\":\"Chat\",\"template\":\"{\\\"tts\\\\":\\\"我是若琪，很高兴认识你\\\"}\",\"feedback\":{\"voiceUrl\":null,\"voice\":\\\"你好\\\"},\"appid\":\"E33FCE60E7294A61B84C43C1A171DFD8\"}",
    "topic": "card"
}
  
```

}

Vui 模块

Card 样式说明

chat样式

消息格式：

```
{  
    "type": "Chat",  
    "template": "xxx",  
    //asr内容 (你对若琪说的话)  
    "feedback": {  
        "voiceUrl": null,  
        "voice": "你好"  
    },  
    //来自于应用的 appid  
    "appid": "E33FCE60E7294A61B84C43C1A171DFD8"  
}
```

template :

```
{"tts":"我是若琪，很高兴认识你"}
```

图示：



Brief summary样式

消息格式：

```
{  
    "appid": "com.rokid.alarm1",
```

```

"feedback": {
    "voice": "帮我设置一个明天早上十点的闹钟"
},
"id": "475b8d28-6a06-4453-993f-84c724876794",
"template": "xxxx",
"type": "Summary"
}

```

template :

```

{
    "buttons": [
        {
            "title": "查看已设置的闹钟",
            //点击跳转的url
            "url": "rokid://app/alarm?deviceId\u003d02010217020001ED"
        }
    ],
    "icon": "https://s.rokidcdn.com/mobile-app/icon/card/alarm.png",
    "items": [
        {
            //点击跳转的url
            "linkUrl": "rokid://app/alarm?deviceId\u003d02010217020001ED",
            "subtitle": "2017年12月15日",
            "title": "10:00"
        }
    ],
    "title": "闹钟",
    "type": "Brief"
}

```



图示：

simple summary样式

消息格式：

```
{  
    "appid": "com.rokid.xxxx",  
    "feedback": {  
        "voice": "xxxx"  
    },  
    "id": "475b8d28-6a06-4453-993f-84c724876722",  
    "template": "xxxx",  
    "type": "Summary"  
}
```

template：

```
{  
    "icon": "https://s.rokidcdn.com/mobile-app/icon/card/tips.png",  
    "title": "若琪",  
    "type": "simple",  
    "items": [  
        {  
            "title": "你好，主人！",  
            "contents": [  
                "我已经准备好与你进行第一次对话，试试对我说：“若琪，介绍一下你自己”\n想了解我的  
更多技能吗？"  
            ]  
        }  
    ],  
    "buttons": [  
        {  
            "title": "查看「若琪技能」",  
            "url": "https://skill.rokid.com/store"  
        }  
    ]  
}
```

图示：



image summary样式

消息格式：

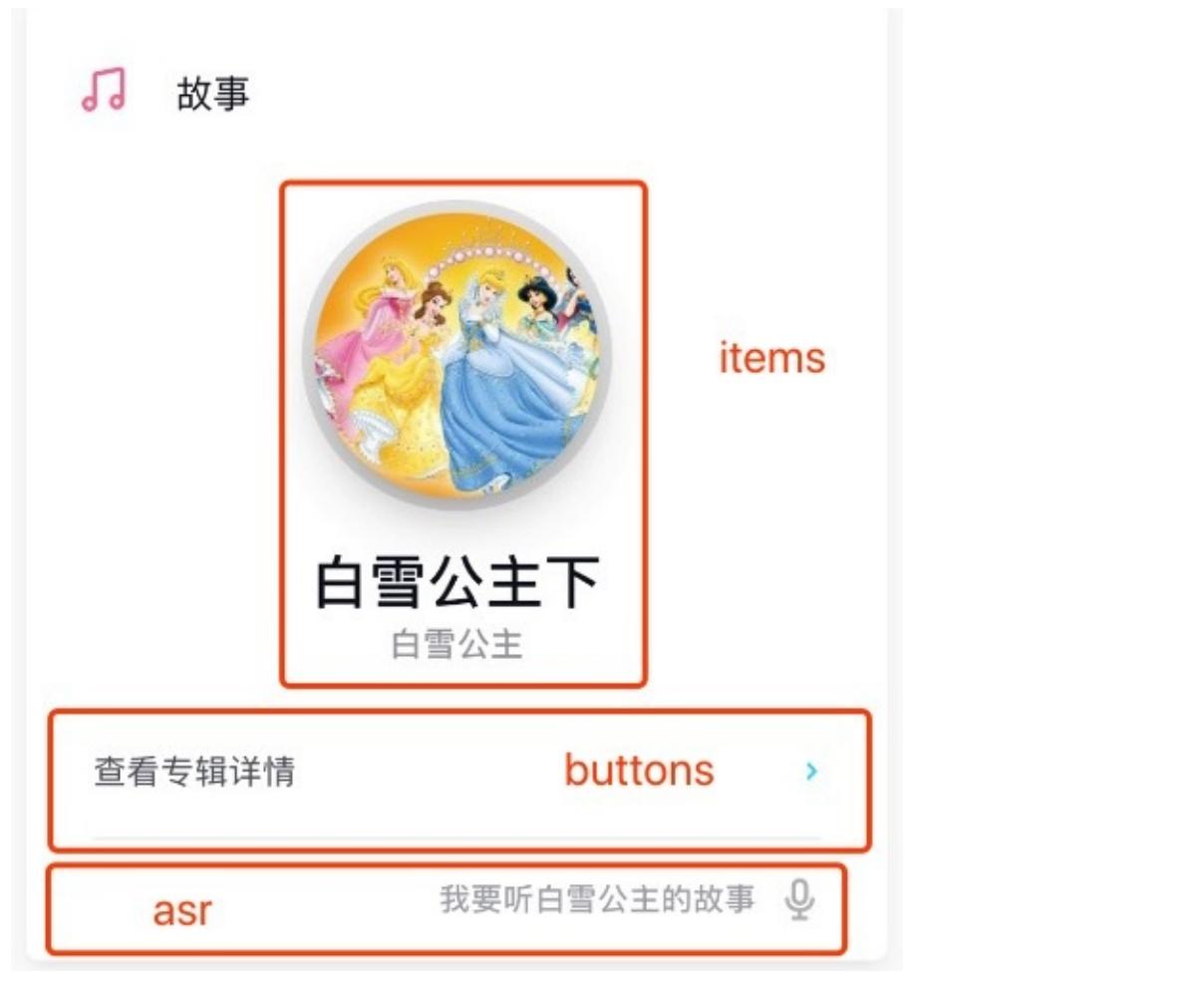
```
{  
    "dbId": 0,  
    "from": "02010217020001ED",  
    "msgStamp": "Dec 14, 2017 6:22:02 PM",  
    "msgTxt": "xxx",  
    "topic": "card"  
}
```

template：

```
{  
    "type": "Image",  
    "title": "故事",  
    "subtitle": "",  
    "icon": "https://s.rokidcdn.com/mobile-app/icon/card/music.png",  
    "items": [  
        {  
            "title": "白雪公主下",  
            "subtitle": "白雪公主",  
            "imageUrl": "https://rokidstorycdn.rokid.com/story/album/4663853/wKgJK1d7ZleCvqS6AAK2Ys1hXsw742_mobile_large.jpg",  
            "imageType": "Circle",  
            "linkUrl": "rokid://media/v3/detail?id\u003dea7c48fea1654376acbc6d837c6b8d22\u0026appId\u003dR7C638312DA94C54BFE5B3BE2FE33E44"  
        }  
    ]  
}
```

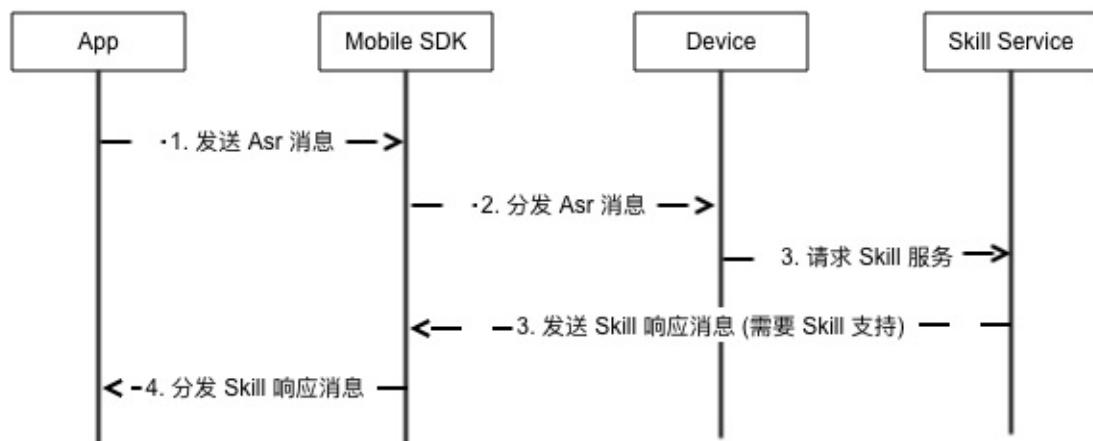
```
        },
    ],
    "buttons": [
        {
            "title": "查看专辑详情",
            "url": "rokid://media/v3/detail?id\u003dea7c48fea1654376acbc6d837c6b8d22\u0026appId\u003dR7C638312DA94C54BFE5B3BE2FE33E44"
        }
    ]
}
```

图示：



Vui 模块

流程



发送ASR

发送 ASR，就是发送 指定，让设备根据指令进行操作。

参数说明

字段	类型	必须?	说明
deviceId	String	是	设备Id
asr	String	是	你对设备说的话

示例代码：

Java:

```

RokidMobileSDK.vui.sendAsr(deviceId, asr, new IChannelPublishCallback() {
    @Override
    public void onSucceed() {
        // TODO
    }

    @Override
    public void onFailed() {
        // TODO
    }
});
```

Kotlin:

```
RokidMobileSDK.vui.sendAsr(deviceId, asr, object : IChannelPublishCallback {
    override fun onSucceed() {
        // TODO
    }

    override fun onFailed() {
        // TODO
    }
})
```

返回参数说明

字段	类型	说明
sendSuccess	boolean	是否发送成功

创建 ASR 纠错

纠正设备错误识别的 ASR 内容。

参数说明

字段	类型	必须?	说明
originText	String	是	原始的 ASR 内容
targetText	String	是	纠正后的 ASR 内容

示例代码:

```
RokidMobileSDK.vui.asrCorrectCreate(originText,
    targetText,
    new IGetAsrErrorCallback() {
        @Override
        public void onGetAsrErrorSucceed(AsrErrorCorrectBean asrErrorCorrectBean) {
            //do something
        }

        @Override
        public void onGetAsrErrorFailed(String errorCode, String errorMessage) {
            //do something
        }
    });
}
```

asrErrorCorrectBean数据格式：

```
[
  {
    // ASR纠错ID
    "id": "xxx",
    // 用户ID
    "accountId": "xxx",
    // 原始的ASR内容
    "originText": "xxx",
    // 纠正后的ASR内容
    "targetText": "xxx",
    // ASR纠错的创建时间
    "createTime": "xxx",
    // ASR纠错的更新时间
    "updateTime": "xxx"
  }
]
```

查询 ASR 纠错

查询这条 ASR 是否被纠错过。

参数说明

字段	类型	必须?	说明
originText	String	是	原始的 ASR 内容

示例代码：

```
RokidMobileSDK.vui.asrCorrectFind(originText, new IGetAsrErrorCallback() {
    @Override
    public void onGetAsrErrorSucceed(AsrErrorCorrectBean asrErrorCorrectBea
n) {
        //do something
    }

    @Override
    public void onGetAsrErrorFailed(String errorCode, String errorMsg) {
        //do something
    }
});
```

更新 ASR 纠错

修改该条 ASR 的纠错内容。

参数说明

字段	类型	必须?	说明
asrId	long	是	ASR纠错ID
originText	String	是	原始的 ASR 内容
targetText	String	是	纠正后的 ASR 内容

示例代码:

```
RokidMobileSDK.vui.asrCorrectUpdate(asrId,
    originText,
    targetText,
    new IGetAsrErrorManageCallback() {
        @Override
        public void onGetAsrErrorManageListSucceed() {
            //do something
        }

        @Override
        public void onGetAsrErrorManageFailed(String errorCode, String
errorMsg) {
            //do something
        }
    });
}
```

删除 ASR 纠错

删除该条 ASR 的纠错内容。

参数说明

字段	类型	必须?	说明
asrId	long	是	ASR纠错ID

示例代码:

```
RokidMobileSDK.vui.asrCorrectDelete(asrId, new IGetAsrErrorManageCallback() {
    @Override
    public void onGetAsrErrorManageListSucceed() {
        //do something
    }

    @Override
    public void onGetAsrErrorManageFailed(String errorCode, String errorMsg)
});
```

```
{  
    //do something  
}  
});
```

获取 ASR 纠错历史列表

获取该用户全部的 ASR 纠错内容。

参数说明

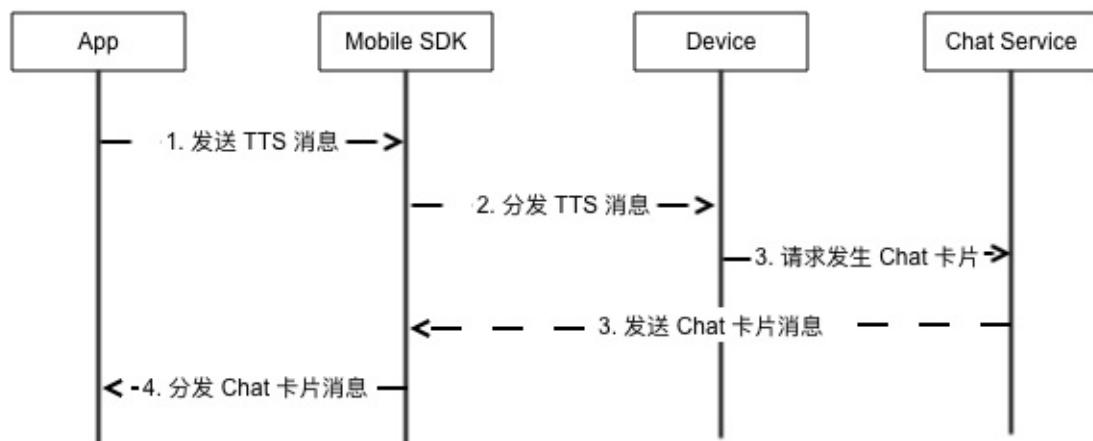
字段	类型	必须?	说明
pageNumber	int	是	页码
pageSize	int	是	每页显示的数量

示例代码:

```
RokidMobileSDK.vui.asrCorrectHistory(pageNumber,  
    pageSize,  
    new IGetAsrErrorListCallback() {  
        @Override  
        public void onGetAsrErrorListSucceed(List<AsrErrorCorrectBean> list) {  
            //do something  
        }  
  
        @Override  
        public void onGetAsrErrorListFailed(String errorCode, String errorMsg)  
        {  
            //do something  
        }  
    });
```

Vui 模块

流程



发送TTS

发送 TTS，就是发送一条能让设备立即说的内容。

参数说明

字段	类型	必须?	说明
deviceId	String	是	设备Id
tts	String	是	让设备说的话

示例代码:

Java:

```

RokidMobileSDK.vui.sendTts(deviceId, tts, new IChannelPublishCallback() {
    @Override
    public void onSucceed() {
        // TODO
    }

    @Override
    public void onFailed() {
        // TODO
    }
});
  
```

Kotlin:

```
RokidMobileSDK.vui.sendTts(deviceId, tts, object : IChannelPublishCallback {  
    override fun onSucceed() {  
        // TODO  
    }  
  
    override fun onFailed() {  
        // TODO  
    }  
  
})
```

返回参数说明

字段	类型	说明
sendSuccess	boolean	是否发送成功

Vui 模块

发送Topic消息

发送Topic消息，可以指定消息类型，

参数说明

字段	类型	必须？	说明
deviceId	String	是	设备Id
topic	String	是	消息主题
text	String	是	消息文本

示例代码：

Java:

```
RokidMobileSDK.vui.sendMessage(deviceId, topic, text, new IChannelPublishCallback()
{
    @Override
    public void onSucceed() {
        // TODO
    }

    @Override
    public void onFailed() {
        // TODO
    }
});
```

Kotlin:

```
RokidMobileSDK.vui.sendMessage(deviceId, topic, text, object : IChannelPublishCallback {
    override fun onSucceed() {
        // TODO
    }

    override fun onFailed() {
        // TODO
    }
})
```

返回参数说明

字段	类型	说明
sendSuccess	boolean	是否发送成功

消息 Event

简介

Mobile SDK 发送的一些 Event 消息，根据业务需求 接收 并实现具体功能。

流程



支持库

****目前Event 使用 EventBus 进行封装，请引入EventBus 库****

```
compile 'org.greenrobot:eventbus:3.0.0'
```

通用消息

介绍

此消息体可以接收 Mobile SDK 发出的全部未经过 Mobile SDK 处理的原始消息。

消息体

SDKChannelMessage

例子

Kotlin

```
@Subscribe(threadMode = ThreadMode.MAIN)
public fun onReceivedSDKChannelMessage(channelMessage: SDKChannelMessage){
    // TODO
}
```

ChannelMessageBean 说明：

参数	类型	说明
sourceDevice	ChannelDeviceBean	发送消息对象
deviceDevice	ChannelDeviceBean	接收消息对象
topic	String	消息类型
text	String	消息结构体

ChannelDeviceBean 说明：

参数	类型	说明
accountId	String	账号Id
deviceId	String	设备Id
deviceTypeId	String	设备类型

帐号 消息

帐号被登出

当前用户登录失效、异地登录时，都会发送这个消息。

消息体

EventUserLoginInvalid

例子

Kotlin

```
@Subscribe(threadMode = ThreadMode.MAIN)
public fun onReceivedUserEvent(eventUserLoginInvalid: EventUserLoginInvalid){
    // TODO
}
```

设备消息

当前选择的设备发生变化

消息体

EventCurrentDeviceChange

例子

Kotlin

```
@Subscribe(threadMode = ThreadMode.BACKGROUND)
public fun onReceived(eventCurrentDeviceChange: EventCurrentDeviceChange){
    // TODO
}
```

EventCurrentDeviceChange 说明：

参数	类型	说明
deviceId	String	当前设备Id

当前设备状态发生变化

消息体

EventCurrentDeviceStatus

例子

Kotlin

```
@Subscribe(threadMode = ThreadMode.BACKGROUND)
public fun onReceived(eventCurrentDeviceStatus: EventCurrentDeviceStatus){
    // TODO
}
```

EventCurrentDeviceStatus 说明：

参数	类型	说明
deviceId	String	当前设备Id

isOnline	boolean	是否在线
----------	---------	------

设备状态发生变化

消息体

EventDeviceStatus

例子

Kotlin

```
@Subscribe(threadMode = ThreadMode.BACKGROUND)
public fun onReceived(eventDeviceStatus: EventDeviceStatus){
    // TODO
}
```

EventDeviceStatus 说明：

参数	类型	说明
deviceId	String	当前设备Id
isOnline	boolean	是否在线

设备解绑

消息体

EventUnbind

例子

Kotlin

```
@Subscribe(threadMode = ThreadMode.BACKGROUND)
public fun onReceived(eventUnbind: EventUnbind){
    // TODO
}
```

EventUnbind 说明：

参数	类型	说明

deviceId	String	当前设备Id
----------	--------	--------

音量发生变化

消息体

EventVolumeChange

例子

Kotlin

```
@Subscribe(threadMode = ThreadMode.BACKGROUND)
public fun onReceived(eventVolumeChange: EventVolumeChange){
    // TODO
}
```

EventVolumeChange 说明：

参数	类型	说明
event	String	固定值：ON_VOLUME_CHANGE
from	String	设备Id
to	String	用户Id
volumeTemplate.mediaCurrent	String	媒体当前音量
volumeTemplate.mediaCurrentmediaTotal	String	媒体最大音量

系统更新信息

消息体

EventDeviceSysUpdate

例子

Kotlin

```
@Subscribe(threadMode = ThreadMode.BACKGROUND)
public fun onReceived(eventDeviceSysUpdate: EventDeviceSysUpdate){
    // TODO
}
```

参数	类型	说明
from	String	消息来源设备Id
to	String	用户Id
versionInfo.getCheckCode	int	0:表示没有更新 ,1:表示有更新
versionInfo.currentVersion	String	当前版本
versionInfo.changelog	String	版本更新信息

媒体消息

简介

此消息主要是设备响应语音指令后发送过来的事件。

消息体

SDKMediaEvent

例子

Kotlin

```
@Subscribe(threadMode = ThreadMode.MAIN)
fun onReceivedMediaMessage(sdkMediaEvent: SDKMediaEvent) {
    // TODO
}
```

SDKMediaEvent 说明：

参数	类型	说明
appId	String	技能id
version	String	内容协议版本号
event	String	事件名称
template.controlInfo.state	String	内容播放状态: PLAYING: 播放中; PAUSED: 暂停播放; STOPPED: 停止播放;

自定义消息

简介

此消息 用来接收 用户自定义TOPIC消息 发送过来的事件。

消息体

ChannelMessageBean

例子

Kotlin

```
@Subscribe(threadMode = ThreadMode.MAIN)
fun onReceiveCustomMessage(customEvent: ChannelMessageBean) {
    // TODO
}
```

ChannelMessageBean 说明：

参数	类型	说明
sourceDevice	ChannelDeviceBean	发送消息对象
deviceDevice	ChannelDeviceBean	接收消息对象
topic	String	自定义消息名称
text	String	自定义消息内容

ChannelDeviceBean 说明：

参数	类型	说明
accountId	String	账号Id
deviceId	String	设备Id
deviceTypeId	String	设备类型

错误码 Error Code

错误码说明：

errorCode	errorMessage	说明
binder_BT_NAME_EMPTY	device name is empty please check your DeviceName	传入的name为空
binder_BT_ADDRESS_EMPTY	found device address empty	name对应的address为 空
binder_BT_DEVICE_NOT_FOUND	device not found during getRemoteDevice	发现蓝牙设备失败
binder_BT_PHONE_DISABLE	ble is disable please check bt state	手机蓝牙在连接过程中 关闭
binder_BT_CONNECT_ERROR	ble connect error	蓝牙连接失败
binder_BT_binder_DATA_ILLEGAL	binderData is illegal please check your userId,wifiSsid,wifiPwd	发送数据非法: 1.userId不能为空; 2.wifiSsid和 wifiPwd不能同时为空; 3. DevicebinderData不能为空)
binder_BT_SERVICES_NOT_FOUND	ble remote service not found sdk BTEngine service uuid error	蓝牙服务获取失败
binder_BT_CHARACTER_NOT_FOUND	ble remote service not found sdk BTEngine character uuid error	蓝牙服务特征值获取失 败
binder_BT_DISCONNECT	ble has been disConnect	蓝牙断开连接
binder_BT_SEND_DATA_ERROR	ble writeCharacteristi error during sendbinderData	蓝牙写入数据过程中失 败
ping_device_RC_DISCONNECT	ble writeCharacteristi error during sendbinderData	SDK与系统长连接断开
ping_device_ID_EMPTY	get device status fail,deviceld is null	设备ID为空
ping_device_USER_ID_EMPTY	ble writeCharacteristi error during sendbinderData	用户ID为空
ping_device_SAME_TASK_RUNNING	ble writeCharacteristi error during sendbinderData	相同的ping个线程在运 行

Rokid 开发者社区服务协议

为了营造规范、有序、安全的开发者社区环境，并利用先进的互联网技术给用户带来便利或更好的体验，芋头科技（杭州）有限公司（下文简称“甲方”）与您（以下简称“开发者”或“乙方”）就开发者社区的使用等相关事项，在杭州市余杭区签订本协议。

特别提示：

开发者通过网络页面点击确认或以其他方式选择接受本协议或使用开发者社区的服务，即表示开发者同意并接受本协议条款，请开发者仔细阅读本协议的全部内容（特别是以粗体标注的内容）。

若开发者在签订本协议前已与甲方或相关方签订了与开发者社区相关的《开发者社区开发者协议》、《服务市场协议》和《安全检测服务协议》，则本协议将在签订日起取代前述协议。

一、定义

1.1 开发者社区：即Rokid开发者社区，指由甲方所拥有并独立经营的、供开发者使用接口等资源以开发、展示、销售和管理开发者应用或服务的平台系统。开发者社区网站登录入口为<https://developer.rokid.com>，但不限于该网站及该网站内的相关页面。

1.2 开发者/乙方：指与甲方签订本协议入驻开发者社区的单位或个人。开发者通过开发者社区获取接口和技术文档开发、完善应用以便服务自身或服务其他用户。开发者可以在接口文档中查看该接口具体由哪个接口提供者提供。

1.3 服务市场：指在开发者社区上，为乙方应用或服务提供发布及订购的自助服务市场和管理系统。

1.4 接口：指接口提供者开发并拥有知识产权的软件或程序，开发者可按照开发者社区的规则将其集成到用户应用或服务中。

1.5 接口提供者：指通过开发者社区向开发者提供接口信息和技术文档的单位或个人，以便开发者开发应用产品及提供服务。

1.6 用户：指所有使用开发者提供的应用或服务的单位或个人。

1.7 应用或服务：指由开发者通过开发者社区上接口提供者所提供的接口和/或技术材料所开发或完善的应用程序及相关服务。开发者通过开发者社区等渠道发布其技能及相关服务。

1.8 法律法规：指适用于相关人士或涉及的标的物的全国性法律、地方性法律、行政法规、部门规章、通知、法令和其它立法、执法或者司法裁决（包括但不限于任何最高人民法院解释）或中华人民共和国相关主管部门发布的规范性文件，无论是否具有法律或者行政法规的地位。

二、协议构成

本协议包括协议正文及甲方已发布的或将来可能发布的关于开发者社区的各类规则（含业务规范，下同）、通知、公告等（以下合称“规则”）。所有上述规则、通知和公告等为本协议不可分割的组成部分，与本协议正文具有同等法律效力。如上述规则、通知和公告等与本协议有冲突，则以后发布者为准。甲方有权根据需要不时地制定、修改本协议和/或各类规则，并通过在开发者社区发布通知的方式

进行公告，而无需单独通知开发者。除非规则或本协议另行规定生效时间，否则甲方制定、修改的协议和规则一经公布即自动生效，成为本协议的一部分。如开发者继续使用开发者社区服务的，则视为开发者接受并同意遵守修改后的协议和规则。

三、特别提示

甲方和开发者均同意并理解：（1）甲方、接口提供者不参与开发者技能的研发、运营等任何活动。因开发者产生的任何纠纷、责任等，以及开发者违反法律法规或本协议约定引发的任何后果，均由开发者独立承担责任，与甲方及接口提供者无关；如侵害到甲方、接口提供者或其他第三方合法权益的，开发者须自行承担全部责任和赔偿一切损失。如甲方或接口提供者因开发者的应用或开发者原因承担法律责任或遭受任何损失，则甲方或接口提供者有权在承担该等责任后向开发者进行追偿，以便使自身不受损失。

（2）除本协议外，如接口提供者就接口使用等事项另行制定使用规范或规则的，开发者需同时遵守该等使用规范或规则。

（3）开发者基于自身使用接口提供者提供的接口，或向用户提供含有接口提供者所提供接口的应用或服务的，开发者应按甲方流程进行操作，并通过甲方或接口提供者审核。

（4）本协议给予开发者的所有授权仅限于开发者用于进行常规开发和测试，且开发者社区有权对开发者的请求次数予以限制。开发者如需进行商业化运营，需获得甲方认证并另行签署协议。

四、甲方服务内容

4.1 甲方是开发者社区的运营者，仅提供中立的平台服务，开发者应自行负责其应用的创作、开放、编辑、加工、修改、测试、运营及维护等，并自行承担相应的费用。若开发者申请在开发者社区展示其应用或服务，开发者同意，甲方有权根据法律法规规定对开发者的应用或服务进行审查，并自行决定是否展示开发者的应用或服务，或从开发者社区中删除该应用或服务。尽管如此，开发者同意，甲方不应因其审核行为而被认定为是应用或服务的共同提供方，且不因审核而对开发者的应用或服务承担任何责任。

4.2 如存在下列情况，甲方有权立即停止为开发者提供本协议项下的服务，并追究开发者的法律责任。甲方有权通过规则、通知等形式进一步补充或说明本条所规定的内容：

（1）开发者违反国家相关法律法规的规定，包括但不限于从事恐怖活动、危害网络安全、洗钱、颠覆政府、煽动民族仇恨等；

（2）开发者违反本协议约定；

（3）甲方收到司法机关或政府机关或监管组织或接口提供者的相关要求；

（4）甲方收到用户或其他第三方通知，投诉某个开发者或其应用或服务存在违反法律规定、存在风险或瑕疵、侵犯他人合法权益或违反其与接口提供者约定的等情形；

（5）甲方认为开发者的行为及所开发的应用或服务可能影响开发者社区的正常运营的；

（6）甲方认为开发者存在违反法律处规定或不当的行为，包括但不限于未经许可获取、存储、披露用户信息、可能危及网络或用户安全等情形；

(7) 甲方认为存在其它可能损害甲方、接口提供者、用户或其他第三方的合法权益的行为。

4.3 甲方有权查阅开发者和用户的注册、交易数据及交易行为，如果甲方发现其中反映可能存在第4.2条规定所列的情形或任何其他问题，甲方有权向开发者发出询问或要求改正的通知，或者直接作出其认为合适的处理，包括但不限于删除相关信息，删除应用或停止对该开发者提供本协议项下的服务。

4.4 开发者同意，甲方有权在开发者的应用或服务或其他相关页面上公开开发者的真实主体信息，以便用户了解应用或服务的提供者身份。

五、开发者承诺和服务使用规范

5.1 开发者需使用Rokid账户和密码登录开发者社区，开发者须同时遵守《Rokid用户协议》以及甲方另行要求开发者签订的协议以及甲方发布的规则等。

5.2 开发者应妥善保管登录开发者社区的账户（包括但不限于Rokid账户或将来开发者社区允许登录的其他账户，以下简称“账户”）和密码、手机等联系方式和通讯工具。甲方通过账户、密码识别开发者身份及指令，使用开发者账户、密码的操作视为开发者本人的操作。开发者同意，如果因开发者的账户、密码遗失、泄露、被盗等所导致的损失及责任由开发者自行承担。

5.3 为保障开发者社区的有序运营和安全性，开发者声明其使用开发者社区服务及相关接口、技术文档时应遵守法律法规。且开发者仅得按本协议及与接口服务提供者的约定（如有）自行使用开发者社区上的权限和接口等，不得为任何第三方申请接口或本协议项下的其他服务。

5.4 开发者保证应具备经营、签署及履行本协议所需各项授权、证照、批准和资质，并保持联系方式的畅通。上述资料如有任何变更，开发者应立即向甲方提交变更后资料。变更资料未经核实前，甲方可完全依赖变更前的资料行事，由此产生的一切风险由开发者自行承担。如果甲方自行发现，或第三方通知发现开发者提供的上述信息不真实或不准确的，甲方有权终止向开发者提供开发者社区服务，由此产生的一切法律责任由开发者自行承担。

5.5 开发者理解并同意，为保护平台安全，甲方有权选择开发者，并可对开发者接入的信息系统实行审查，包括但不限于技术水平审查、安全水平审查等，并根据审查结果向开发者提出防入侵、防病毒等措施建议。若开发者提供的信息系统仍无法符合保护用户数据安全的要求，甲方有权拒绝或终止提供开发者社区服务。但基于甲方并非专业机构，该审查并不保证开发者的应用、服务或行为完全合法合规或完全无风险，也并不代表甲方对开发者的任何行为提供担保、许可或向第三人承担任何共同责任。

5.6 开发者同意接收来自甲方及其关联公司、合作伙伴发出的邮件、信息。

5.7 未经甲方书面同意，开发者及其员工不得使用甲方及其关联公司（包括但不限于芋头、Rokid、若琪等）的商号、商标、服务标志、企业名称、其他标志、标识、用语等进行任何活动，亦不得对外宣传与甲方存在除本协议之外的任何其他关联。

5.8 开发者应负责审核使用其应用的用户的真实身份、资质，并在司法机关、行政机关等有权机关或甲方要求时提供用户的身份信息；如接口提供者需要开发者到用户经营场所进行巡检或提交相应数据统计资料的，开发者应积极配合。

5.9 开发者在开发者社区展示、销售其应用或服务的，开发者应自行按照相关法规运营应用或服务，并自行承担全部责任，包括但不限于：

- (1) 依照相关法律法规的规定，保留应用或服务的访问、使用等日志记录；
- (2) 自行对应用或服务中由用户使用应用或服务产生的内容（包括但不限于留言、消息、评论、名称等）进行审查，保证其不违反相关法律、法规、政策的规定以及公序良俗等；
- (3) 自行缴纳税费。如用户要求开具发票，则开发者应为用户开具或按照用户要求出具合法有效的收费凭证；
- (4) 按照甲方的要求提供著作权、专利权、商标权等相关权利证明文件。

5.10 如出现以下情形，甲方有权对开发者展示或销售的应用或服务进行下架处理、停止向开发者提供服务等措施：

- (1) 违反法律法规、本协议及开发者社区和/或服务市场相关规则及开发者社区相关规则；
- (2) 故意传播计算机病毒等破坏性程序以及任何危害计算机信息网络安全，或者应用或服务中含有计算机病毒、木马、间谍软件或任何其他可能妨碍、干扰开发者社区和/或服务市场正常运营的内容；
侵犯甲方、用户或任何其他第三方的知识产权或其他合法权益；
- (3) 从事任何“二清”、洗钱、暴力、反政府等违法违规行为，或为违法违规行为提供便利；
- (4) 其它违反开发者社区和/或服务市场规则规定的行为。

5.11 若开发者决定停止使用本协议项下的服务、或停止提供应用或服务的，导致用户不能再使用开发者应用或服务的，开发者应当向所有用户至少提前30天发出书面通知，并退还用户已支付但未使用部分的相应费用，并与用户达成一致的解决方案；因此产生的损失及责任由开发者承担。且开发者理解并同意，用户使用开发者提供的应用和服务发生的任何纠纷应由用户和开发者自行协商解决，甲方不承担任何责任。但为了维护用户权益，甲方有权引导用户与开发者自行解决纠纷；或由甲方协助开发者解决用户纠纷。甲方有权以普通或非专业人员的知识水平标准自行判断开发者是否为用户提供了优质、合理、安全的服务。甲方可制定相应的投诉规则规定具体细则。

5.12 开发者社区仅提供给开发者进行常规的开发和测试，每个开发者账号下每日服务（含接口）调用次数限制为1000次。

5.13 开发者明确理解并同意，如因其违反有关法律或者本协议之规定，使甲方及其关联公司遭受任何损失，受到任何第三方的索赔，或任何行政管理部门的处罚的，除本协议另有规定外，开发者应对上述损失、索赔或处罚向甲方及其关联公司进行赔偿。

六、数据及隐私

6.1 为了开发者更便利地使用开发者社区服务，开发者同意其在接口提供者处的相关数据（如有）可由接口提供者同步给甲方。

6.2 开发者同意，在法律法规允许的情况下，开发者社区中有关开发者的应用的运营数据（指用户或开发者在使用开发者社区中产生的相关数据，包括但不限于用户或开发者在提交、操作行为中形成的数据及各类交易数据、用户注册信息以及用户使用开发者应用所形成的数据）和用户数据（指用户在使用开发者社区、应用等过程中产生的与用户有关的数据，包括但不限于用户提交的语音数据、图像数

据、用户操作行为形成的数据) 的全部权利均归属于甲方, 且甲方可将上述数据同步给接口提供者, 以便其改进服务和接口。未经甲方事先书面同意, 开发者不得为本协议约定之外的目的使用前述数据, 不得为任何目的擅自保存、使用或授权他人使用前述数据。

6.3 开发者应自行对使用开发者社区而获取的各种数据, 采取合理、安全的技术措施确保其安全, 但甲方有权根据政府部门、司法机关等有权机关的合法要求提供前述数据。

6.4 开发者收集、使用用户数据或代用户向甲方提交用户信息, 应取得用户的明确授权。如甲方或接口提供者认为开发者收集、使用用户数据的方式, 存在违反法律法规情况以及可能损害用户体验或利益, 甲方或接口提供者有权要求开发者删除相关数据并不得再以该方式收集、使用用户数据, 否则甲方有权停止向开发者提供开发者社区服务。

6.5 开发者不得将用户信息用于不利于用户的目的, 包括但不限于代替用户与开发者自身签署合同, 或代替用户确认信息。因违反本约定引起的纠纷及后果由开发者自行负责解决和承担。因此给甲方或接口提供者或用户带来任何损失的, 开发者应负责予以赔偿。

6.6 一旦开发者停止使用本协议项下的服务, 或甲方终止提供本协议项下服务的, 开发者应立即删除其从开发者社区及各接口中获得的各种数据(包括各种备份), 包括但不限于前述运营数据和用户数据, 且不得再以任何方式进行使用。

七、服务费用及开发者收费

7.1 开发者社区部分服务(含接口)可能以收费方式提供, 如开发者使用该等收费服务, 应遵守收费服务相关的规则。甲方为开发者社区向开发者提供的服务付出大量的成本, 除开发者社区明示的收费业务外, 开发者社区向开发者提供的服务目前是免费的。如未来开发者社区向开发者收取合理费用, 开发者社区会提前通过法定程序并以本协议第十三条约定的方式通知开发者。

7.2 甲方或接口服务提供者可能根据实际需要修改和变更收费服务的收费标准、收费方式, 甲方或接口服务提供者也可能会对曾经面向社会公众免费的服务开始收费。前述修改、变更或开始收费前, 甲方将在开发者社区相应页面进行通知或公告。如果开发者不同意上述修改、变更或付费内容, 则应停止使用该服务。

7.3 甲方同意, 开发者有权就开发者社区中的应用或服务向用户收费, 但为保证交易安全必须采用甲方规则中认可的支付渠道。

7.4 开发者同意其应用或服务的订购数据以甲方系统记录为准。

八、违约责任

8.1 如开发者违反本协议、相关规则的, 甲方有权根据相关协议、规则采取中止或终止向开发者提供本协议项下的服务及本协议等措施。

8.2 开发者因违反本协议而给用户造成损失的, 开发者应向用户赔偿; 如甲方为了保证用户权益先行向用户补偿的, 开发者应在收到甲方支付通知后10个工作日内向甲方返还甲方先行补偿给用户的款项。如逾期未支付, 开发者应每日按照应付金额的0.05%向甲方支付滞纳金。

8.3 如因开发者违反本协议约定或开发者的其他原因导致甲方或其关联公司遭受损失的，包括但不限于甲方或其关联公司向用户或第三方支付赔偿，或向政府缴纳罚款，则甲方有权或开发者不可撤销地授权甲方或其关联公司按甲方要求从开发者的待结算费用（如有）中直接扣除相应的费用，同时开发者不可撤销地授权甲方或其关联公司可按甲方要求从开发者的账户中直接扣除相应费用；上述扣除费用仍不足以抵偿甲方或其关联公司损失的（包括如律师费等费用），开发者还应当补足。

8.4 开发者未经甲方书面同意将其开发成果投入商业运营或以其他方式允许最终用户使用的，甲方有权立即终止向开发者提供服务，并要求开发者支付违约金，违约金以开发者因该等开发成果所获得的所有直接、间接收益之总和计算。

九、协议的解除和转让

9.1如有下列情形发生，甲方有权单方面解除本协议，终止向开发者提供服务：

- (1) 开发者为非法目的使用开发者社区服务的；
- (2) 开发者使用开发者社区服务或提供的应用损害甲方、接口提供者、用户或其他第三方合法权益的；
- (3) 开发者违反法律法规或本协议约定或违反其与接口提供者约定的。

9.2开发者同意，除第9.1条所述情形外，甲方和接口提供者有权根据风险及自身业务运营情况需要，随时终止向开发者提供开发者社区服务及接口的部分或全部，因此导致开发者无法使用服务或服务受到限制，或导致用户无法使用开发者应用或服务的，甲方不构成违约，接口提供者也不承担任何法律责任。

9.3甲方有权将其在本协议项下的权利和义务全部或部分转让给第三方，并通过开发者社区或甲方服务市场发布公告。

十、服务终止后的处理

开发者停止使用开发者社区后，或甲方终止向开发者提供本协议项下的服务后，甲方可能不再转发任何未曾阅读或发送的信息给开发者、用户或其他第三方，同时甲方亦不就终止服务而对开发者或用户或任何第三方承担任何责任。但甲方将保存开发者保留原账户中或与之相关的任何信息。对于开发者过往的违约行为，甲方仍可依据本协议向开发者追究违约责任。

十一、责任限制和免责

11.1开发者明确理解和同意，甲方不保证接口提供者的接口或技术文档一定能满足开发者的需要，不保证接口的正常运行，不保证甲方能够不中断地或无故障地提供本协议项下的服务。

11.2对发布在开发者社区的各服务市场的应用或服务的开发、运营、支持和维护，开发者同意独立承担所有的风险和后果。对于开发者发布在甲方服务市场上的任何应用或内容，甲方不承担任何责任。

11.3甲方服务市场仅为开发者发布应用或服务提供技术支持。本协议的签署并不代表甲方成为用户与开发者进行交易的参与者，甲方不对开发者和用户交易承担任何责任。

11.4 开发者同意，因下列原因导致无法正常提供开发者社区服务的，甲方及接口提供者不承担任何责任：（1）甲方或接口提供者对其各自系统及设备进行停机维护或升级；（2）因台风、地震、洪水、雷电、恐怖袭击、等不可抗力原因；（3）用户的电脑软硬件和通信线路、供电线路出现故障的；（4）开发者或用户操作不当或通过非甲方授权或认可的方式使用开发者社区或甲方服务市场的；（5）因病毒、木马、恶意程序攻击、网络拥堵、系统不稳定、系统或设备故障、通讯故障、电力故障、第三方服务瑕疵或政府行为等原因。尽管有前款约定，甲方将采取合理行动积极促使服务恢复正常。

11.5 在任何情况下，甲方及接口提供者均不对任何间接性、后果性、惩戒性、偶然性、特殊性或惩罚性的损害承担责任（即使甲方已被告知该等损失的可能性）。

十二、知识产权

12.1 除12.3条规定的情形外，开发者社区上所有内容，包括但不限于著作、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由甲方或其他权利人依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经甲方或其他权利人书面同意任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表甲方网站程序或内容，或对开发者社区内的接口或技术材料的部分或整体进行分解、反向工程、反编译、修改、调整。

12.2 除本协议另有规定外，甲方及接口提供者授予开发者有限的、非排他性的、仅限于中华人民共和国境内（为避免歧义，不包含香港、澳门和台湾）的、可随时终止的和不可再分发的许可，仅限于开发者自己访问和使用开发者社区开发、测试、显示其应用。

12.3 开发者享有其开发的应用的著作权、商标权、专利权等，但其中涉及与接口相关或者其他与甲方所提供的代码或技术资料互动或相关的部分除外。如该等应用中使用了第三方软件或技术，开发者保证其已经获得了第三方的合法授权，否则因此产生的任何第三方索赔或其他责任均应由开发者承担。

12.4 如开发者通过甲方开发者社区提交或发布应用或服务，即表明开发者授予甲方服务期内的、非排他性的、完全给付并免费的全球性许可，允许甲方使用、复制、再许可、重设格式、修改、删除、添加、公开显示、重现、分发和执行其应用或服务，以及将其存储和缓存在甲方指定服务器上。

12.5 开发者理解并同意，甲方不保证开发者社区提供的接口中完全不侵犯任何第三方的著作权、专利权、商业秘密等知识产权或其他合法权益，如上述接口中使用技术、软件涉嫌侵犯第三方知识产权等合法权益的，甲方有权立即终止向开发者提供涉嫌侵权的服务，甲方不承担因此导致的任何的损失和责任。

十三、通知和送达

13.1 开发者在注册成为开发者社区用户，并接受甲方提供的服务时，开发者应该向甲方提供真实有效的联系方式(包括电子邮件地址、联系电话等)，对于联系方式发生变更的情况，开发者有义务及时更新有关信息，并保持可被联系的状态。甲方将向开发者的上述联系方式的其中之一或其中若干送达各类通知，此类通知的内容可能对开发者的权利义务产生重大的有利或不利影响，请开发者务必及时关注。开发者在此授权甲方通过注册时填写的或注册后修改的手机号码或者电子邮箱向开发者发送可能感兴趣的的商品广告信息、促销优惠等商业性信息;开发者如果不愿意接收此类信息，有权通过相应的退订功能进行退订。

13.2 甲方通过上述联系方式向开发者发出通知，其中以电子邮件的方式发出的书面通知，包括但不限于在开发者社区发布公告，向开发者提供的联系电话发送手机短信，向开发者提供的电子邮件地址发送电子邮件，向开发者的账号发送即时信息、系统消息以及站内信信息，在发布或发送成功后即视为送达；以纸质载体发出的书面通知，按照提供联系地址交邮后的第五个自然日即视为送达。对于在开发者社区上因交易活动引起的任何纠纷，开发者同意司法机关(包括但不限于法院)可以通过手机短信、电子邮件或其他即时信息等现代通讯方式或邮寄方式向开发者送达法律文书(包括但不限于诉讼文书)。开发者指定接收法律文书的手机号码、电子邮箱或即时通讯账号等联系方式为开发者在开发者社区注册、更新时提供的手机号码、电子邮箱联系方式以及即时通讯账号，司法机关向上述联系方式发出法律文书即视为送达。开发者指定的邮寄地址为法定联系地址或提供的有效联系地址。开发者同意司法机关可采取以上一种或多种送达方式向开发者送达法律文书，司法机关采取多种方式向开发者送达法律文书，送达时间以上述送达方式中最先送达的为准。

13.3 开发者同意上述送达方式适用于各个司法程序阶段。如进入诉讼程序的，包括但不限于一审、二审、再审、执行以及督促程序等。你应当保证所提供的联系方式是准确、有效的，并进行实时更新。如果因提供的联系方式不确切，或未及时告知变更后的联系方式，使法律文书无法送达或未及时送达，由开发者承担由此可能产生的法律后果，如相应文书视为送达，法院/仲裁庭可缺席审理不再通过公告送达。

十四、其他约定

14.1 非代理和禁止委托：甲方与开发者确认双方在本协议项下的安排、约定均不能视为双方存在任何代理关系；开发者确认其将不会在其任何宣传、市场、销售等活动或资料、文件中声称或误导其他人相信开发者是甲方代理。

14.2 本协议之解释与适用，以及与本协议有关的争议，均应依照中华人民共和国法律予以处理，并以上海仲裁委员会为管辖机构。

Rokid开发者社区免责声明

《Rokid开发者社区免责声明》（以下简称“本声明”）是杭州芋头科技有限公司及其关联公司（以下简称“我们”）及其运营合作单位（以下简称“合作单位”）关于我们的开发者社区网站 <https://developer.rokid.com> 与我们的产品、程序及服务（包括但不限于若琪以及相关的应用）所作的声明。

我们在此特别提醒您（或称“用户”，指注册、登录、使用、浏览本服务的个人或组织）认真阅读、充分理解本声明。请您审慎阅读并选择接受或不接受本声明（未成年人应在法定监护人陪同下阅读）。除非您接受本声明所有条款，否则您无权注册、登录或使用本声明所涉相关服务。您的注册、登录、使用等行为将被视为对本声明全部内容的认可。

您对本声明的接受即表示自愿接受全部条款的约束，包括接受我们公司对条款随时所做的任何修改。本声明可由我们随时更新，更新后的声明一旦公布即代替原来的声明，恕不再另行通知，用户可在官方网站查阅最新版声明。在我们修改本声明相关条款之后，如果用户不接受修改后的条款，请立即停止使用我们提供的服务，用户继续使用我们提供的服务将被视为已接受了修改后的声明。

1. 如发生下述情形，我们不承担任何法律责任：

- 被第三方识别的风险；
- 依据法律规定或相关政府部门的要求提供您的个人信息；
- 由于您的使用不当或其他自身原因而导致任何个人信息的泄露；
- 由于您的使用不当或您的电脑软硬件和通信线路、供电线路出现故障导致不能正常提供服务的情形；
- 系统停机维护或升级导致不能正常提供服务的情形；
- 任何由于黑客攻击，电脑病毒的侵入，非法内容信息、骚扰信息的屏蔽，政府管制以及其他任何网络、技术、通信线路、信息安全*全管理措施等原因造成的服务中断、受阻等不能满足用户要求的情形；
- 用户因第三方如运营商的通讯线路故障、技术问题、网络、电脑故障、系统不稳定及其他因不可抗力造成的损失的情形；
- 用户之间通过我们的官方网站或我们产品、程序及服务与其他用户交往，因受误导或欺骗而导致或可能导致的任何心理、生理上的伤害以及经济上的损失；
- 我们产品、程序及服务明文声明，不以明示、默示或以任何形式对我们及其合作公司服务之及时性、安全性、准确性做出担保。

2. 用户信息及第三方产品或服务信息均由用户及第三方自行提供并上传，由用户及第三方对其提供并上传的所有信息承担相应法律责任。用户所发布的任何内容并不代表和反映我们的任何观点或政策，我们对此不承担任何责任。

3. 用户的购买行为应当基于真实的消费需求，不得存在对产品及/或服务实施恶意购买、恶意维权等扰乱正常交易秩序的行为。基于维护交易秩序及交易安全的需要，一旦发现上述情形时我们可主动执行关闭相关交易订单等操作，用户自行承担此风险。

4. 我们的开发者社区网站和我们产品、程序及服务包含其他用户提供的用户内容。您与其他用户的互动仅属于您与其他用户之间的行为，我们不控制且不对以上用户内容承担法律责任，也没有检查、监控、审批、核准以上用户内容的义务。您对因使用该用户内容以及与其他用户互动产生的

风险承担法律责任。我们的官方网站和我们产品、程序及服务对此类行为不承担任何法律责任。

5. 我们因制止用户的违约或侵权行为导致的风险，由用户自行承担。
6. 在任何情况下，我们均不对任何间接性、后果性、惩罚性、偶然性、特殊性或刑罚性的损害，包括因用户使用我们服务而遭受的利润损失，承担责任。尽管本协议中可能含有相悖的规定，我们对您承担的全部责任，无论因何原因或何种行为方式，始终不超过您在注册期内因使用我们服务而支付给我们的费用(如有)。
7. 我们不保证为向用户提供便利而设置的外部链接的准确性和完整性，同时，对于该外部链接指向的不由我们实际控制的任何网页上的内容，我们不承担任何责任。
8. 我们对相关声明享有版权及其修改权、更新权和最终解释权。