

Table of Contents

Rokid 开发者社区文档	1.1
开发者社区介绍	
Rokid开发者社区介绍	2.1
快速开始	
开始开发Rokid技能	3.1
开始接入Rokid语音	3.2
开发Rokid技能	
开始创建第一个技能：我要喝咖啡	4.1
为技能定义语音交互	4.2
Rokid 预定义词表	4.3
Rokid NLP正则表达式使用指南	4.4
Rokid OAuth 指南	4.5
Rokid JS Engine 使用指南	4.6
在技能中使用SSML	4.7
Rokid 语音交互设计指南	4.8
Rokid 示例技能	4.9
为设备开启语音交互	
获取认证文件	5.1
Rokid Android SDK 接入指南	5.2
Linux 设备接入	
Linux 设备开发引导	6.1
云端交互SDK(Speech)	6.1.1
设备拾音SDK(BSiren)	6.1.2
设备拾音服务	6.1.3
服务管理	6.1.4
语音应用管理	6.1.5
音频服务框架	6.1.6
TTS服务	6.1.7

Led阵列	6.1.8
Mic阵列	6.1.9
按键触控输入	6.1.10
蓝牙服务	6.1.11
电量服务	6.1.12
语音应用包管理	6.1.13
媒体播放库	6.1.14
配网服务	6.1.15
Node.js开发者生态	6.1.16
OTA升级	6.1.17

API REFERENCE

Rokid 技能协议文档	7.1
拦截器接口文档	7.2
Rokid 接口文档	7.3
Rokid 客户端 SDK 文档	7.4
Rokid 前端降噪模块 API 文档	7.5

设计语音产品

Rokid 语音产品硬件设计指南	8.1
Rokid 硬件交互设计指南	8.2

智能家居开发

Rokid 智能家居开发文档	9.1
----------------	-----

协议条款

技能发布标准	10.1
服务协议	10.2
免责声明	10.3

支持

Rokid讨论区	11.1
Rokid开发者社区官网	11.2
Rokid 官网	11.3
Rokid 智能家居	11.4

- Rokid 开发者社区文档

Rokid 开发者社区文档

Rokid开发者社区文档，旨在帮助开发者更好的使用社区提供的各种能力。

文档发布在 <https://rokid.github.io/docs/>

以下是文档整体结构。

开发者社区介绍

- [Rokid开发者社区介绍](#)

快速开始

- [开始开发Rokid技能](#)
- [开始接入Rokid语音](#)

开发Rokid技能

- [开始创建第一个技能：我要喝咖啡](#)
- [为技能定义语音交互](#)
- [Rokid 预定义词表](#)
- [Rokid NLP正则表达式使用指南](#)
- [Rokid OAuth 指南](#)
- [Rokid JS Engine 使用指南](#)
- [在技能中使用SSML](#)
- [Rokid 语音交互指南](#)
- [Rokid 示例技能](#)

为设备开启语音交互

- [获取认证文件](#)
- [Rokid Android SDK 接入指南](#)

Linux 设备接入

- [Linux 设备开发引导](#)

API REFERENCE

- [Rokid 技能协议文档](#)
- [拦截器接口文档](#)
- [Rokid 接口文档](#)
- [Rokid 客户端 SDK 文档](#)
- [Rokid 前端降噪模块 API文档](#)

设计语音产品

- [Rokid 语音产品硬件设计指南](#)
- [Rokid 硬件交互设计指南](#)

智能家居开发

- [Rokid 智能家居开发文档](#)

协议条款

- [技能发布标准](#)
- [服务协议](#)
- [免责声明](#)

支持

- [Rokid讨论区](#)
- [Rokid开发者社区官网](#)
- [Rokid 官网](#)
- [Rokid 智能家居](#)

- Rokid开放平台介绍
 - 什么是Rokid开放平台
 - Rokid技能开发工具
 - Rokid语音接入
 - Rokid开放服务的优势
 - 从前端降噪到语音合成的一整套解决方案
 - 经过产品检验的开放方案
 - 不断进步的开放生态
 - 顶尖的语音智能技术
 - Rokid开放服务合作方式
 - 免费使用基础语音服务
 - 可能产生费用的项目
 - 发布时间
 - 支持

Rokid开放平台介绍

Rokid开放平台介绍

什么是Rokid开放平台

Rokid开放平台内含一整套语音解决方案，能够让终端设备拥有语音交互能力，使用户直接使用语音就能与你的设备进行交互。同时依托开放平台提供各种技能（Skills，可以理解为手机中的APPs），终端用户仅通过语音命令就能获取音乐、新闻、天气、问答等各种日常服务。目前这套方案用于Rokid Alien、Rokid Pebble两款产品。

Rokid开放平台语音解决方案建立在云端，将会随着用户的不断使用而变得更加聪明。同时也会因为开发者不断新增的各种技能（Skills）而变得更加好用全面。

Rokid开放平台语音服务包含**Rokid技能开发工具**和**Rokid语音接入**。

Rokid技能开发工具

Rokid技能开发工具帮助开发者为所有搭载Rokid开放服务的设备开发有趣的技能（Skills），实现用户各式各样的语音交互需求。

Rokid语音接入

Rokid语音接入能够为配有麦克风和扬声器的联网硬件设备开启Rokid开放服务所提供的智能、可扩展的语音能力。用户可以直接使用语音让搭载Rokid语音服务的设备播放音乐、查天气、播报新闻，以及其他技能（Skills）所提供的各种功能。

Rokid开放服务的优势

从前端降噪到语音合成的一整套解决方案

Rokid开放平台提供的方案包含从前端硬件的远场拾音、降噪、回声消除等技术，到自然语言识别、语义理解、自然语言合成这一整套的解决方案。

经过产品检验的开放方案

使用Rokid方案的两款产品均是体验最佳的语音智能设备代表。 Rokid方案能够帮助硬件厂商获得相比其他方案更好的拾音、语音交互效果。



左侧为Rokid Pebble，右侧为Rokid Alien，分别在CES2017, CES2016获奖。

不断进步的开放生态

搭载Rokid语音方案的设备可以直接使用经过数次优化的基础语音技能，如天气、新闻等。

另外，在Rokid开放平台中，你可以为自己或全平台的用户开发各类有趣的技能（Skills），来满足用户不断增加的语音需求。

同时，其他开发者开发的公共技能（Skills）也将不断更新至搭载Rokid语音方案的设备中。

顶尖的语音智能技术

Rokid自主研发的语音智能技术，能够在麦克风降噪、远场识别、上下文等多个方面提供业内顶尖的方案，以支持用户丰富深入的语音互动体验。

Rokid开放服务合作方式

免费使用基础语音服务

- 免费提供硬件设计参考方案
- 免费提供远场激活拾音、降噪、回声消除等硬件技术方案
- 免费提供基础语音服务
 - 语音识别（ASR）
 - 自然语言理解（NLP）
 - 语音合成播报（TTS）
- 免费使用Rokid云服务
- 可以使用平台中的所有公共技能（Skills）
- 可以自定义3字以上激活词

可能产生费用的项目

- 涉及版权的技能（Skills）内容
- 需要Rokid提供额外人力支持的部分

- 例如自定义激活词的训练
- 或自定义的TTS人声训练

发布时间

- 2017年4月1日，开放内测。
- 2017年6月1日，正式发布。

支持

- [Rokid开发者社区](#)

- [开始开发Rokid技能](#)
 - [文档结构](#)
 - [什么是Rokid技能开发工具](#)
 - [1 当前支持的技能种类](#)
 - [1.1 不同开放性的技能](#)
 - [1.2 不同类型的技能](#)
 - [2 创建一个技能](#)
 - [2.1 设计您的技能](#)
 - [2.2 为您的技能搭建后端服务](#)
 - [2.3 将您的技能提交审核](#)
 - [3 分享您的技能](#)
 - [快速开始](#)
 - [模板工程](#)
 - [支持](#)

开始开发Rokid技能

为搭载Rokid解决方案的设备开发有趣的技能。

文档结构

- [当前支持的技能种类](#)
- [创建一个技能](#)
- [分享您的技能](#)
- [快速开始](#)
- [模板工程](#)
- [支持](#)

什么是Rokid技能开发工具

Rokid开发者社区所提供的语音方案为用户准备了各种不同场景下的应用，比如音乐、天气、新闻、百科等等，我们将他们称之为：**技能（Skills）**。

Rokid技能开发工具可以帮助您创建各种各样的Rokid技能，以此触达所有搭载Rokid语音方案设备背后的终端用户，向他们提供各种有趣的服务，比如：

- 问一些具体的问题（“若琪，明天杭州会不会下雨。”）
- 玩语音小游戏（“若琪，我要玩文字猜谜。”）
- 控制智能家居设备（“若琪，把卧室的灯打开。”）
- 播放新闻摘要（“若琪，播放最新的新闻。”）

Rokid技能开发工具包含开发工具、文档、以及丰富的示例代码，能够帮助开发者以最快的速度完成技能的开发。在Rokid沉淀的经验之上，专注于更富有创造性的工作。

1 当前支持的技能种类

不同种类的技能可以用来面对不同的业务场景，这是您在开始创建技能时就需要思考的问题。Rokid技能开发工具提供2个维度的4种技能，分别是：

- [不同开放性的技能](#)

- 公开技能
- 私有技能
- 不同类型的技能
 - 自定义技能
 - 预定义技能

1.1 不同开放性的技能

首先，你需要决定是将技能开放给所有搭载Rokid方案的设备，还是仅授权您自己的产品或其他指定产品使用。

① 公开技能

公开属性的技能将会对所有搭载Rokid语音解决方案的设备开放，用户可以通过技能商店轻松开启您开发的技能。

② 私有技能

私有属性的技能无法向所有用户开放，仅针对经过你授权的企业或个人的特定类型的设备开放。用户需要在被授权的设备上才能够使用你的技能。

另外，当选择私有技能时，可以选择创建本地技能，允许开发者收到语义解析的结果后调用设备中的apk。这在为拥有大量本地应用的设备开发技能时特别有用。

1.2 不同类型的技能

① 自定义技能

如果你面对的是复杂的语音交互场景，你需要选择 **自定义技能**。该类型的技能需要你对 **语音交互** 进行自定义。具体来说，你需要定义：

- 技能可处理的具体需求。我们将这些需求称之为 **意图**。举例来说，以下都可以被视为某个技能中的一个意图：
 - 播放音乐
 - 查询天气
 - 播放新闻
 - 其他任何你能想到的一个具体动作
- 用户达成一个意图所有可能的对话方式，我们称之为 **用户语句**。用户语句决定了一个用户与你技能之间的交互方式，举例：
 - “给我放一首歌”(这句话对应了**播放音乐**这个意图)
 - “杭州今天天气怎么样”（这句话对应了**查询天气**这个意图）
 - “今天有什么新闻”（这句话对应了**播放新闻**这个意图）
- 技能理解用户语句并将其转换为意图，有时还需要 **词表** 的帮助。比如：
 - 理解“给我放一首歌”，需要拥有**歌曲库词表**
 - 理解“杭州今天天气怎么样”，需要拥有**城市和日期词表**
 - 理解“今天有什么新闻”，需要拥有**日期词表**
- 此外，Rokid还需要通过 **入口词** 来分辨用户是在和你的技能进行语音交互。用户需要使用含有入口词的语句来唤起你的技能。比如，你为你的音乐技能设定的入口词为“Rokid音乐”。

完成以上定义后，用户就可以用如下语句与你的技能进行交互了：

 | 用户：“若琪，让**Rokid**音乐给我放一首歌”

Rokid将会理解用户的请求，并将**播放音乐**这个意图发送给你的技能。

只要你在[语音交互](#)中预制了充分的用户语句、词表和意图，并且通过代码在后端服务实现这些意图，自定义技能就能够最大限度的满足你预期的用户需求。这种技能最灵活，但也因为需要配置[语音交互](#)，而较为复杂。

② 预定义技能

如果你想创建一个听音乐这样的内容类技能，或是能够开关灯、调节空调温度这样用于智能家居设备的技能，你可以考虑使用[预定义技能](#)。该类型的技能的[语音交互](#)将由Rokid定义且不断更新，你仅需要在后端服务中直接实现对应的意图即可。预定义技能不完全依赖入口词唤起，因此对用户会更加友好。比如用户可以直接说：“若琪，把房间的灯打开。”来使用您创建的智能家居技能。

该类型的技能还在准备中，将在后续开放。

2 创建一个技能

在定义完技能种类、技能名称、技能入口词等关键信息之后，你还需要做以下工作，来完成一个技能的创建。

2.1 设计您的技能

一个好的技能离不开出色的语音交互，这需要您对自然语言、人类对话的基本原理有简单的理解。请阅读[Rokid语音交互指南](#)以了解如何设计出色的语音交互体验。

当您创建一个自定义技能时，您需要通过[定义语音交互](#)将用户请求与您的服务能处理的意图关联起来，并通过定义合理的[用户语句](#)让您的技能能够准确识别用户的各种意图。

2.2 为您的技能搭建后端服务

你可以将您的服务部署在自建或其他任何云端服务器中。

当您在为技能编写后端代码时，请参考我们的[技能协议文档](#)。

在技能的语音交互定义完成、后端服务部署成功之后，你便可以开始对您的技能进行测试了。

2.3 将您的技能提交审核

一旦您完成了上述步骤，就可以准备将您的技能提交至Rokid技能商店，与所有的Rokid用户分享您的技能了。

在此之前，技能需要通过Rokid的审核，您需要：

- 充分测试您的技能
- 完善技能的发布信息
- 完善技能的隐私合规内容

技能提交审核之后，需要耐心等待一段时间。审核完成之后您会收到来自Rokid的通知邮件。

3 分享您的技能

在技能成功通过审核之后，别忘了到[Rokid讨论区](#)与大家分享过您的技能。

快速开始

- [6步创建第一个Rokid技能](#)

模板工程

- [Rokid示例技能](#)

支持

- [Rokid讨论区](#)

- [开始接入Rokid语音](#)
 - [文档结构](#)
 - [Rokid语音接入介绍](#)
 - [Rokid语音接入与Rokid技能](#)
 - [开始接入](#)
 - [1. 注册设备](#)
 - [2. 为您的产品接入Rokid SDK](#)
 - [3. 更多配置](#)
 - [完善您的产品](#)
 - [支持](#)

开始接入Rokid语音

文档结构

- [Rokid语音接入介绍](#)
- [开始接入](#)
- [完善您的产品](#)
- [支持](#)

Rokid语音接入介绍

Rokid语音接入能够为配有麦克风和扬声器的联网硬件设备开启若琪所提供的智能语音交互能力。用户可以直接使用语音让搭载若琪的设备播放音乐、查天气、播报新闻，以及其他技能（Skills）所提供的各种服务。

Rokid语音接入与Rokid技能

当您的设备接入若琪后，即可使用众多丰富有趣的Rokid公有技能，也可以为您的设备创建个性化的私有技能。通过[开始开发Rokid技能](#)了解更多。

开始接入

1. 注册设备

设备信息

在「开发者社区-->语音接入」中注册您的设备，并完善设备名称、设备图片、设备分类及描述。

认证文件

您需要为您的设备创建一个认证文件，以获取接入若琪的权限。具体方法请查看[获取认证文件](#)。

2. 为您的产品接入Rokid SDK

SDK简介

Rokid开发者社区SDK包含Siren、Speech、NLP、ASR、TTS几大模块。

- Siren: 拾音模块，接收HAL的音频数据，算法处理，滤波；
- TTS: 文字转语音；
- Speech和ASR: 都是语音转文字，不同的是Speech专门处理Siren的语音事件。

九步部署SDK

通过[Rokid SDK 接入指南](#)，了解如何快速接入Rokid SDK。

和您的设备做一次互动

您可以创建一个[开始创建第一个技能：我要喝咖啡](#)，并将您的设备添加到此技能的测试设备列表中，就可以与设备进行互动啦。

与开发者分享您的方案

在成功为您的设备开启若琪语音交互能力之后，别忘了到Rokid讨论区与大家分享您接入若琪的心得哦。

3. 更多配置

在提供以上基础能力之外，我们还提供了更多的自定义配置。

核心技能接入（准备中）

核心技能是由Rokid提供的预装技能，在接入若琪之后，您便可以直接使用这些技能，无需额外开发。您也可以在「语音服务-服务接入-核心技能接入」中对您的设备是否接入这些技能进行配置。

拦截器

拦截器可以允许开发者在进入若琪NLP匹配之前或者在若琪NLP处理完成后结果为空时将请求进行拦截，拦截到开发者自己的HTTPS拦截器。您可以在「语音服务-服务接入-拦截器」中对此功能进行配置，详细请了解[拦截器接口文档](#)。

私有技能接入

私有属性的技能无法向所有用户开放，仅针对经过技能开发者授权的企业或个人的特定类型的设备开放。用户需要在被授权的设备上才能够使用私有技能。

当有开发者（包括您自己）向您授权其开发的私有技能后，您可以在「语音服务-服务接入-私有技能接入」的列表中对这些技能进行如下配置：

- 启用/禁用：为当前设备启用或禁用被授权的私有技能。
- 取消授权：取消指定私有技能对您设备的授权。

完善您的产品

以下文档将帮助您打造出优秀的语音交互产品。

- [Rokid 语音产品硬件设计指南](#)：为您提供由Rokid总结的硬件方案设计指南。
- [Rokid 硬件交互设计指南](#)：为您提供硬件交互的设计参考。

支持

- [Rokid讨论区](#)

- 示例技能：我要喝咖啡
 - 您将学会
 - 您需要
 - 1 注册Rokid开发者账号
 - 2 创建一个技能
 - 3 定义技能的语音交互并进行后端配置
 - 4 测试你的技能
 - 5 自定义这个技能
 - 6 发布

示例技能：我要喝咖啡

您将学会

- 如果通过6步快速创建一个Rokid技能。

您需要

- Rokid开发者账号
- GitHub中的示例代码

1 注册Rokid开发者账号

在[Rokid开放平台](#)免费注册一个Rokid开发者账号。

2 创建一个技能

登录Rokid开发后台，





登录后，选择「技能开发工具」，并点击「创建新技能」。



选择你需要的Rokid服务

选择「技能开发」为搭载Rokid技术的设备创造新的能力，或者选择「语音服务」为你的设备添加基于Rokid技术的语音交互能力。
了解更多请参阅[开发者社区介绍](#)。

技能开发工具

使用Rokid技能工具，您可以将自己的技能添加到Rokid，打造出引人入胜的语音体验。

创建技能

语音接入

基于语音服务，用户可以与他们的Rokid产品交互。
包括播放音乐、控制设备、控制智能家居等。

语音接入



技能列表

要了解有关构建Rokid技能的更多信息，你可以通过[Rokid技能开发工具介绍](#)快速了解技能开发工具的使用方法。
同时我们建议你通过[Rokid开发者讨论区](#)与Rokid成员以及众多开发者们交流，获得更多灵感。

创建新技能

技能名称	语言	属性	类型	修改时间	状态	操作
------	----	----	----	------	----	----

给你的技能起个名字

1. 技能属性请选择「公开」。
2. 技能类型请选择「自定义技能」。
3. 为技能起个好名字。
4. 为技能起一个朗朗上口的「入口词」，用户将用他来唤起你的技能。



此处的示例 技能名称，入口词 均为「我要喝咖啡」。

完成后请点击「下一步」。

3 定义技能的语音交互并进行后端配置

首先请查看我们的[技能模板：我要喝咖啡](#)。

定义语音交互

接着在「语音交互」页面中，

- 将模板工程 > voice-interaction 中的 `intent.json` 文件内容，复制进「意图定义」编辑框。

```

1 - 4 "intents": [
2 - 5   {
3 - 6     "intent": "bestcoffeebar",
4 - 7     "slots": [],
5 - 8     "user_says": [
6 - 9       "杭州哪里有好点的咖啡馆",
7 - 10      "杭州哪里好点的咖啡馆",
8 - 11      "杭州哪里有好一点的咖啡馆"
9 - 12    ],
10 - 13  },
11 - 14  {
12 - 15    "intent": "nicedrink",
13 - 16    "slots": [],
14 - 17    "user_says": [
15 - 16      "米萨咖啡馆种好喝",
17 - 18      "米萨咖啡馆种好喝的一种"
19 - 20  ]
21 }]

```

把 `intent.json` 的内容复制进「意图定义」

- 提示：此时可以在该页面右侧的“语音交互测试”中测试配置是否正确。

完成后请点击「下一步」。

完成服务配置

接着在「配置」页面中，

1. 选中「JS Engine」，
2. 将模板工程 > sample-js 中的 js-engine.js 文件内容复制到下方的编辑框中。

The screenshot shows the 'Skill Configuration' page for a skill named '我要喝咖啡'. The left sidebar has tabs for 'Skill Information', 'Voice Interaction', 'Configuration' (which is selected), 'Integration Testing', 'Release', and 'Privacy & Compliance'. The main area has tabs for 'Chinese' and 'English'. Under 'Configuration', there's a note about using the Rokid JS Engine documentation. A red box highlights the code editor containing the following JavaScript code:

```
1- exports.handler = function(event, context, callback) {
2-   var rokid = Rokid.handler(event, context, callback);
3-   rokid.registerHandlers(handlers);
4-   rokid.execute();
5- };
6-
7- var handlers = {
8-   'ROKID_INTENT_WELCOME': function () {
9-     try{
10-       this.emit('tts', {tts: '您好，请问有什么可以帮您。'});
11-     }catch(e){
12-       this.callback(e);
13-     }
14-   },
15-   'bestcoffeebar': function () {
16-     try{
17-       this.emit('tts', {tts: '我看位于向善路89号的永萨咖啡馆很不错。'});
18-     }catch(e){
19-       this.callback(e);
20-     }
21-   }
22- }
```

At the bottom, there are 'Save' and 'Next Step' buttons, with a red arrow pointing to the 'Next Step' button.

完成后请点击「下一步」。

4 测试你的技能

接下来，在「集成测试」页面中，

1. 在「后端服务测试」下，输入「用户语句」，比如“若琪，打开「入口词」。”
2. 你将会看到下方框体中出现经过Rokid语音服务解析的Json「服务请求」，和相应的「服务返回」。
3. （可选）如果需要在机器上进行测试，请根据公司ID、设备Type ID、设备ID来「添加测试设备」。

隐私与合规 + 添加测试设备

后端服务测试
通过服务模拟来测试你的后端服务。

输入语句
打开我要喝咖啡
发送 清空

服务请求
{"device": {
 "basic": {
 "locale": "zh-cn",
 "timestamp": 0,
 "vendor":
 "B488A73BD050F0FD7CF1847FB120975"
 },
 "location": 0,
 "media": {}
},
"request": {
 "content": {}
}}

服务返回
{"form": "cut",
"shouldEndSession": true,
"type": "NORMAL",
"version": "2.0.0",
"voice": {
 "behaviour": "APPEND",
 "item": {
 "tts": "您好，请问有什么可以帮助您。"
 },
 "needEventCallback": false
},
"resType": "INTENT",
"resId": ""}

展开JS日志

提交审核

下一步

此处使用：“打开「我要喝咖啡」”进行测试

隐私与合规 + 添加测试设备

后端服务测试
通过服务模拟来测试你的后端服务。

输入语句
杭州哪里有好点的咖啡馆
发送 清空

服务请求
{"device": {
 "basic": {
 "locale": "zh-cn",
 "timestamp": 0,
 "vendor":
 "B488A73BD050F0FD7CF1847FB120975"
 },
 "location": 0,
 "media": {}
},
"request": {
 "content": {}
}}

服务返回
{"type": "NORMAL",
"version": "2.0.0",
"voice": {
 "behaviour": "APPEND",
 "item": {
 "tts": "我看位于问溪路89号的米萨咖啡就很不错。"
 },
 "needEventCallback": false
},
"resType": "INTENT",
"resId": ""}

展开JS日志

提交审核

下一步

此处使用：“杭州哪里有好点的咖啡馆”进行测试

后端服务测试
通过服务模拟来测试你的后端服务。

输入语句
米萨咖啡最好喝的是哪一种

服务请求

```
{
  "basic": {
    "locale": "zh-cn",
    "timestamp": 0,
    "vendor": "8488A73B0050F0FD7CF1647FB120975"
  },
  "location": {},
  "media": {}
},
{
  "request": {
    "content": {
      "applicationId": "695af843-4b56-4278-
    }
  }
}
```

服务返回

```
{
  "type": "NORMAL",
  "version": "2.0.0",
  "voice": {
    "behaviour": "APPEND",
    "item": {
      "tts": "只要是咖啡都很好喝。"
    }
  },
  "needEventCallback": false
},
{
  "resType": "INTENT",
  "resId": "D08BA1D1CF4C44E2B30646EAC596014B"
}
```

展开JS日志

提交审核 下一步

此处使用：“米萨咖啡最好喝的是哪一种”进行测试

5 自定义这个技能

- 首先，您可以对技能的名称和入口词进行修改，比如起一个新的名字「我要喝橙汁」。

我要喝咖啡
ID: rokid.rsk.695af843-4b56-4278-bda4-633c45893bea

中文

技能信息
语音交互
配置
集成测试
发布
隐私与合规

通用设置

技能名称
技能的名称会在技能商店中向终端用户展示，名称长度应介于2-15个字之间，不能包含特殊字符。
我想喝橙汁

入口词
用来唤起这个技能的关键词。
我想喝橙汁

- 接着您可以修改技能的intent名称和相应的用户语句。比如将问句变为「杭州哪里有好点的鲜榨果汁」。

我要喝咖啡
ID: rokid.rsk.695af843-4b56-4278-bda4-633c45893bea

中文

技能信息
语音交互
配置
集成测试
发布
隐私与合规

意图定义
请在下面的编辑框中，用JSON格式定义用户的意图，详情请见[定义语音交互](#)。

```

1- {
2-   "intents": [
3-     {
4-       "intent": "bestjuice",
5-       "slots": [],
6-       "user_says": [
7-         "杭州哪里有好点的鲜榨果汁"
8-       ]
9-     },
10-    {
11-      "intent": "nicedrink",
12-      "slots": [],
13-      "user_says": [
14-        "算得及汁里面种点好酒"
15-      ]
16-    }
17-  ]
18-}

```

语音交互测试

测试

- 修改Js Engine中的相关回复。

The screenshot shows the Rokid Developer Community interface. In the top navigation bar, there are links for '控制台', '文档中心', '讨论区', 'DrSmile', '退出', and '中文'. Below the navigation, there are tabs for '技能开发' and '语音接入'. A breadcrumb navigation '〈 返回技能列表' is present. The main content area displays a skill card for '我要喝咖啡' (ID: rokid.rsk.695af843-4b56-4278-bda4-633c458938ee). On the left, a sidebar menu includes '技能信息', '语音交互', '配置', '集成测试' (selected), '发布' (disabled), and '隐私与合规'. The right panel contains the skill's code editor with the following JS Engine code:

```

    var handlers = {
      'ROKID_INTENT_WELCOME': function () {
        try {
          this.emit('tts', {tts: '你好，请问有什么可以帮助您。'});
        } catch(e) {
          this.callback(null);
        }
      },
      'bestjuice': function () {
        try {
          this.emit('tts', {tts: '位于尚源路89号的米庐咖啡的现榨果汁也很不错。'});
        } catch(e) {
          this.callback(null);
        }
      },
      'neededrink': function () {
        try {
          this.emit('tts', {tts: '我推荐有橙汁。'});
        } catch(e) {
          this.callback(null);
        }
      }
    };
  
```

4. 通过集成测试页面看看修改后效果吧。

6 发布

将「发布」及「隐私与合规」所需的内容填写完毕后，点击页面最下方亮起的“提交审核”按钮提交发布申请，等待Rokid 审核通过后技能即可发布上线。

需注意：开发者需要通过个人身份认证才可以将技能提交审核。

The screenshot shows the Rokid Developer Community interface for publishing the 'coffee' skill. The top navigation bar and sidebar are identical to the previous screenshot. The main content area displays the skill card for '我要喝咖啡'. The '发布' tab in the sidebar is selected. The publishing form includes fields for '类别' (Category) set to '请选择', '测试说明' (Test Description) with placeholder text '请对您的技能特征进行详细说明，例如账号与权限、使用场景等具体要求。(该内容仅用于Rokid开放平台测试团队，不会向用户展示，至少20个汉字)', '国家和地区' (Country and Region) with options '面向所有地区用户' (Target all regions) and '面向特定地区用户' (Target specific regions), '技能摘要' (Skill Summary) with placeholder text '对自己的技能进行简要描述 (至少10个字, 不超过50个字)', and '技能描述' (Skill Description) with placeholder text '详细描述你的技能。将在技能商店中展示 (至少50个字, 不超过1000个字)'.

- 为技能定义语音交互
 - 概述
 - 理解语音交互的逻辑
 - 意图定义 (intent)
 - 用户输入的数据
 - 用户语句 (user_says)
 - 自定义词表内容 (slot)
 - Confirm用法指南
 - 补全用户的不完整意图
 - Session用法指南
 - 以问天气为例
 - 其他参考

为技能定义语音交互

在创建自定义技能的过程中，您需要为此技能定义语音交互，以覆盖用户的语音交互场景，做出合理的响应。

Tips: 预定义技能不需要您定义语音交互。

概述

- 理解语音交互的逻辑
- 意图定义
- 用户输入的数据
- Confirm用法指南
- Session用法指南
- 其他参考

理解语音交互的逻辑

为自定义技能定义语音交互，实际上是将 用户语句 词表 等用户输入和您的后端服务能够处理的 意图 进行映射关联的过程。

要处理好这个映射关系，你需要弄清楚以下几个主要概念：

1. 意图定义：一个能够声明您后端服务可以处理的意图合集的JSON结构。
2. 用户输入数据：
 - 用户语句：意图中表达用户输入的部分。可以将用户可能的交互语句和您的意图关联起来。您可以为一个意图预置尽可能多的用户语句。
 - 词表：一个词条的列表，这些词条的内容将会被用户语句或者意图所引用。比如您可以把所有的城市作为值放在「城市」词条中，给「杭州今天天气怎么样」这个用户语句引用。

您将会在Rokid开发者社区 > 开发后台 > 技能工具中某一个技能的语音交互页面编辑上述信息。

下面将会为您详细介绍如何进行语音交互的定义。

意图定义 (intent)

在Rokid开发者社区中，意图用来表示用户的一个具体语音请求。如果需要，每一个意图都可以声明自己依赖的词表，也就是需要在语义解析之后需要输出给后端服务的内容。比如在「我要喝咖啡」这个技能的意图定义中，我们会定义一个询问咖啡馆的意图 `bestcoffeebar`，它需要一个依赖词表 `city` 来确定搜索范围。当用户说：“若琪，打开我要喝咖啡 问问杭州哪里有好点的咖啡馆。”时，Rokid开发者社区将把 `bestcoffeebar` 的intent请求发给您的后端服务，同时还将带上词表名称 `city` 中的“杭州”这个值。如果此时未定义 `city` 这个词表依赖，“杭州”这个参数将不会被输出给您的后端服务。

您将在意图定义框中用JSON格式定义一套有效的意图。比如如下案例中定义了两个意

图： `bestcoffeebar` 和 `nicedrink`：

```
{
  "intents": [
    {
      "intent": "bestcoffeebar",
      "slots": [
        {
          "name": "city",
          "type": "LIST_OF_CITIES"
        }
      ],
      "user_says": [
        "... "
      ]
    },
    {
      "intent": "nicedrink"
    }
  ]
}
```

每个意图都拥有三个参数：

- `intent` 参数表示意图名称。
- `slots` 参数表示与上述意图依赖的词表名称和内容。
- `user_says` 参数表示用户语句，将会在下方[用户语句](#)部分详细说明。

在上述例子中，意图 `bestcoffeebar` 依赖的词表是 `city`。

词表通过词表名称 `name` 和词表内容 `type` 来定义。比如，上述例子中的 `city` 通过 `LIST_OF_CITIES` 这个此表内容来定义，其中包含各城市名称的值（比如杭州、北京）。另外，一个词表还可以引用Rokid开发者社区预先定义好并对外开放的词表内容，比如时间等，此部分将会陆续开放。

用户输入的数据

意图定义完成后，您需要定义用户输入数据和意图之间的映射关系。用户输入数据通过词表和用户语句来定义。

用户语句 (`user_says`)

您需要为意图配置任何可能的用户语句，形成映射关系。例如，我们对上例中意图定义的JSON进行完善：

```
{
  "intents": [
    {
      "intent": "bestcoffeebar",
      "slots": [
        {
          "name": "city",
          "type": "LIST_OF_CITIES"
        }
      ]
    }
  ]
}
```

```

        ],
        "user_says": [
            "$city哪里好点的咖啡馆",
            "$city哪里有好一点的咖啡馆",
            "$city哪里有好点的咖啡馆"
        ]
    }
    {
        "intent": "nicedrink",
        "user_says": [
            "米萨咖啡哪种最好喝",
            "米萨咖啡最好喝的是哪一种",
        ]
    }
}
]
}

```

这样，有3句不同的用户语句对应到了意图 `bestcoffeebar`，有2句不同的用户语句对应到了意图 `nicedrink`。意味着用户说出这些语句时，就会匹配到对应的意图。

这里面关于 `$` 的引用，会在下方[自定义词表内容](#)继续说到。

用户语句可以枚举，同时支持正则表达式，具体请参考[Rokid正则表达式使用指南](#)。

请配置尽可能多的用于语句以匹配用户多种多样的表达方式。另外，在后续填写技能发布信息时，您需要从这些语句中选出3句最能代表您技能特点的语句展示在技能说明中告知用户。

特别提醒： 用户语句是句式强匹配，除特殊情况外，不建议在句末添加标点符号，否则可能会导致句式无法匹配。

比如在上例中将语句加上？标点：

杭州哪里好点的咖啡馆？

那么此时只有ASR解析结果为 `杭州哪里好点的咖啡馆？` 的语句才可以被识别，而 `杭州哪里好点的咖啡馆` 则无法被识别。

相反，如果配置的语句为：

杭州哪里好点的咖啡馆

那么语句 `杭州哪里好点的咖啡馆？` 和 `杭州哪里好点的咖啡馆` 以及末尾带有其他标点的句子均能被正确识别。

自定义词表内容 (slot)

在Rokid开发者社区提供的[预定义词表](#)之外，开发者可以自定义自己的词表内容。在上例中，`LIST_OF_CITIES` 词表内容可能包含如下部分：

北京
上海
广州
深圳
杭州

只要有可能被用户说出来的相关词语，你都可以写进词表内容中。Rokid开发者社区将会把用户命中的词表内容直接发送给您的技能。

开发者可以把任何可以抽象的内容定义为词表，并在用户语句中通过 `$ + 词表名称` 进行调用。比如当您希望定义以下几种用户语句时：

我想喝咖啡

```
我要喝咖啡
我想要喝咖啡
我要喝可乐
我想喝果汁
```

可以抽象出两个词表：

```
iwant
```

```
我想
我要
我想要
```

```
drink
```

```
咖啡
可乐
果汁
```

并通过以下语句统一表达：

```
$iwant喝$drink
```

需要注意：

- 一个词表内容可以被多个不同的词表名称引用，用于把同一类的值输出到不同的应用场景中。比如在下例中，`ROKID.NUMBER_ZH` 可以被不同的词表名称引用多次，在用户说出 `两小时十五分钟三十六秒` 的时候，能够输出 `"slots": {"hour": "两", "min": "十五", "sec": "三十六"}`。

```
{
  "intents": [
    {
      {
        "intent": "time",
        "slots": [
          {
            "name": "hour",
            "type": "ROKID.NUMBER_ZH"
          },
          {
            "name": "min",
            "type": "ROKID.NUMBER_ZH"
          },
          {
            "name": "sec",
            "type": "ROKID.NUMBER_ZH"
          }
        ],
        "user_says": [
          "$hour小时",
          "$min分钟",
          "$sec秒",
        ]
      }
    ]
  }
}
```

- 一个技能中，所有词表内容的值的总数，不能超过10万个。

Confirm用法指南

当您的技能需要在某些场景下对用户的意图进行确认时（尤其是一些不完整的意图，比如当用户说“我饿了”时，您需要向用户确认他想吃什么东西），您可以使用confirm功能来完成语音交互。

以下是一个例子。

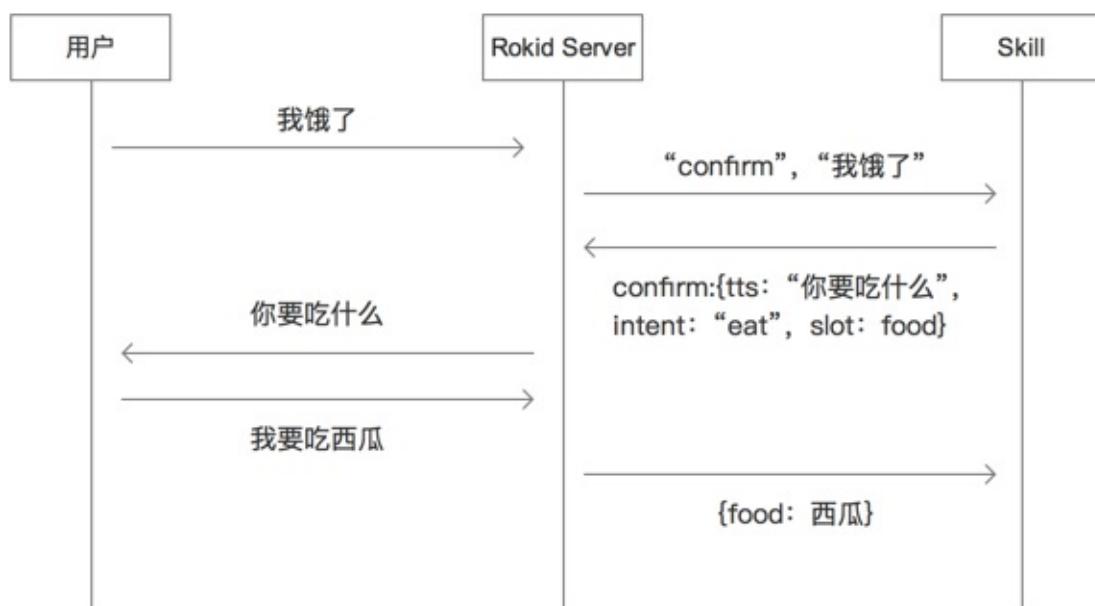
补全用户的不完整意图

当用户在您的技能中说：“若琪，我饿了”这样一个不完整意图时，您希望能够去确认并在用户的帮助下补全这个意图。可以参照如下进行语音交互配置。

```
{
  "intents": [
    {
      "intent": "eat",
      "slots": [
        {
          "name": "food",
          "type": "list"
        }
      ],
      "user_says": [
        "!我要吃$food"
      ]
    },
    {
      "intent": "confirmeat",
      "slots": [ ],
      "user_says": [
        "我饿了"
      ]
    }
  ]
}
```

语音交互流程

以下是上述场景的完整交互流程。



第一步：当用户说“我饿了”时，交由Rokid Server 处理后，告知技能“我饿了”是confirm类型的用户语句。Request内容如下：

```
{
  "request": {
    "reqType": "intent",
    "reqId": "010116000100-ad1f462f4f0946ccb24e9248362c504a",
    "content": {
      "applicationId": "com.rokid.confirm",
      "intent": "confirmeat",
      "slots": {}
    }
  }
}
```

第二步：技能收到Confirm用户语句时，回传Rokid Server Response，内容示例如下：

```
{
  "response": {
    "action": {
      "voice": {
        "action": "PLAY",
        "item": {
          "tts": "你要吃什么"
        }
      }
    },
    "confirm": {
      "confirmIntent": "eat",
      "confirmSlot": "food",
      "optionWords": [
        "西瓜",
        "桔子"
      ]
    }
  }
}
```

Rokid Server处理消息并向用户询问“你要吃什么”。

第三步：用户回答“我要吃西瓜”，Rokid Server处理消息告知技能{food: 西瓜}。

```
{
  "request": {
    "reqType": "intent",
    "reqId": "010116000100-ad1f462f4f0946ccb24e9248362c504a",
    "content": {
      "applicationId": "com.rokid.confirm",
      "intent": "eat",
      "slots": {
        "food": "西瓜"
      }
    }
  }
}
```

如何退出confirm

- 若用户回答的内容，不在预设的词表内，则将重复询问用户，总共三次。三次回答均不符合词表内容将退出confirm。
- 若用户不回答任何内容，超过5分钟后将推出confirm
- 若用户想要强制退出confirm，可以通过“关闭+技能激活词”的句式退出。

使用Js-engine配置Confirm

您可以阅读js-engine 使用指南中 [ttsWithConfirm相关配置](#) 了解更多。

Session用法指南

Session可以通过 `attributes` 来实现。具体协议请参考[技能协议中的Session定义](#)。

Session目前只能作用在同一Domain下，切换Domain会清空Session。此时 `newsession = true`。`attributes` 也会被清空。

以问天气为例

当询问天气技能：“今天天气怎么样”时，会开启一个新的session，此时 `attributes` 为空，`newSession = true`，将会发送如下Request给此技能的后端服务：

```
{
  "context": {
    "application": {
      "applicationId": "xxx"
    },
  },
  "request": {
    "content": {
      "domain": "custom.skill.xxx",
      "intent": "query_weather",
      "slots": {"date": "今天"}
    },
    "reqId": "57792AB32F114788A3E48910FB9CD7BC",
    "reqType": "INTENT"
  },
  "session": {
    "attributes": {},
    "newSession": true
  },
  "version": "2.0.0"
}
```

收到询问天气的请求后，后端服务会给出Response。在这一步中，开发者需要让后端服务把 `city` 值插入到Response的 `attributes` 中，如下：

```
{
  "response": {
    "action": {
      "shouldEndSession": false,
      "type": "NORMAL",
      "version": "2.0.0",
      "voice": {
        "behaviour": "REPLACE_ALL",
        "item": {
          "tts": "杭州今天的天气很好"
        },
        "needEventCallback": false
      }
    }
  },
  "version": "2.0.0",
  "session": {
    "attributes": {
      "city": "杭州"
    }
  }
}
```

```
    }
}
```

之后，用户继续向天气技能询问：“明天呢？”。此时Rokid开发者社区会把 `attributes` 中的city属性带到这次的Request中：

```
{
  "context": {
    "application": {
      "applicationId": "xxx"
    },
    "request": {
      "content": {
        "domain": "custom.skill.xxx",
        "intent": "query_weather",
        "slots": {"date": "明天"}
      },
      "reqId": "57792AB32F114788A3E48910FB9CD7BC",
      "reqType": "INTENT"
    },
    "session": {
      "attributes": {"city": "杭州"},
      "newSession": false
    },
    "version": "2.0.0"
}
```

以此类推。但当切换Domain，比如用户说：“我想听一首歌”时，`session`将会被清空。

其他参考

- [Rokid语音交互指南](#): 能够帮助您更好的定义语音交互。
- [Rokid技能协议文档](#): 能够让您查阅完整的语音交互所需的协议。

- Rokid 预定义词表
 - 1 数字、日期和时间相关
 - 2 常用名词词汇
 - 3 其他词汇
 - 4 ROKID.ANY
 - 举个例子
 - 注意

Rokid 预定义词表

目前已开放的预定义词表，持续更新中。

1 数字、日期和时间相关

数字的预定义词表。

自定义词表类型	简介	支持语言
ROKID.NUMBER_ZH	阿拉伯数字及汉语数字	中文
ROKID.DATE_ZH	描述某年某月某一天	中文
ROKID.RECENT_ZH	描述「近期」的时间词汇	中文
ROKID.YEAR_ZH	描述以「年」为单位的时间词汇	中文
ROKID.MONTH_ZH	以「月」为单位的时间词汇	中文
ROKID.DAY_ZH	以「日」为单位的时间词汇	中文
ROKID.HOUR_ZH	描述以「小时」为单位的时间词汇	中文
ROKID.MINUTE_ZH	以「分钟」为单位的时间词汇	中文
ROKID.SECOND_ZH	以「秒」为单位的时间词汇	中文
ROKID.TIME_ZH	描述具体的时间或时间段	中文

2 常用名词词汇

常用名词的预定义词表。

自定义词表类型	简介	支持语言
ROKID.DEVICE_ZH	家居电器类的词汇	中文
ROKID.LAMPS_ZH	各类灯具词汇	中文
ROKID.COUNTRY_ZH	国家	中文
ROKID.WORLDCITY_ZH	世界城市	中文
ROKID.CN_PROVINCE_ZH	中国省份	中文
ROKID.CN_CITY_ZH	中国城市	中文
ROKID.CN_ARE_ZH	中国县 / 区	中文
ROKID.COLOR_ZH	颜色	中文

ROKID.CURRENCY_ZH	国际货币单位	中文
ROKID.VOICE_ZH	描述声音、音量的词汇	中文
ROKID.EXTRAINFO_ZH	温度单位	中文
ROKID.INFORMATION_ZH	描述信息的词汇	中文
ROKID.ALARM_ZH	描述闹钟的词汇	中文
ROKID.NETWORK_ZH	描述网络的词汇	中文
ROKID.FESTIVAL_ZH	中外节日	中文
ROKID.SOLARTERMS_ZH	二十四节气	中文
ROKID.LUNAR_ZH	描述农历和公历	中文
ROKID.ANIMALYEAR_ZH	生肖年	中文
ROKID.ERAYEAR_ZH	干支年	中文
ROKID.ZONE_ZH	描述早上、晚上等某个时间段的词汇	中文
ROKID.MOVIE_ZH	电影	中文

3 其他词汇

其他词汇的预定义词表。

自定义词表类型	简介	支持语言
ROKID.AH_ZH	语气词	中文
ROKID.ALL_ZH	描述「所有」的词汇	中文
ROKID.YES_ZH	表示「肯定」的词汇	中文
ROKID.NO_ZH	表示「否定」的词汇	中文
ROKID.HIGH_ZH	描述「高、大、升」等词汇	中文
ROKID.LOW_ZH	描述「低、小、降」等词汇	中文
ROKID.NEW_ZH	描述「新」的词汇	中文
ROKID.OLD_ZH	描述「旧」的词汇	中文
ROKID.TRIGGER_ZH	描述「若琪」的词汇	中文

4 ROKID.ANY

ROKID.ANY为提供给用户使用的特殊预定义词表，它可以匹配任何非空输入内容。主要用法：

- 在用户语句中引用。例如，我想听\$any 可以将“听”字之后的任意内容作为词表内容进行歌曲搜索。

举个例子

这里是一个播放音乐的意图，示例如下：

```
{
  "intents": [
    {
      "name": "play_music"
    }
  ]
}
```

```
{  
    "intent": "playmusic",  
    "slots": [  
        {  
            "name": "music"  
            "type": "list_of_music"  
        }  
        {  
            "name": "any",  
            "type": "ROKID.ANY"  
        }  
    ],  
    "user_says": [  
        "我想听$music",  
        "我想听$any"  
    ]  
}
```

1. 假设「十年」这首歌不在您的 `music` 词表中，用户说“我想听十年”，将无法通过 `我想听$music` 命中 `playmusic` 这个意图。
2. 在引用了 `ROKID.ANY`，并添加了 `我想听$any` 语句之后，“我想听十年”将能够命中 `playmusic` 意图。并将「十年」作为词表的值传递给您的后端服务。
3. 您可以参考这些命中 `any` 词表的歌曲，补充到音乐的自定义词表中。

注意

- 请勿将ANY作为整句用户语句使用。
- 请勿在词表中引用 `ROKID.ANY` 这个值。

- Rokid NLP正则表达式使用指南

Rokid NLP正则表达式使用指南

目前支持的正则符号

符号	说明	样例
^	仅匹配此符号后的内容	^\$location的天气
\$ (放在句尾)	仅匹配此符号前的内容	我想知道\$any\$
\$ (后接英文字符)	表示引用某个词表 (slot)	我要听\$singer的歌
?	此符号前的字符或子表达式在匹配时可有可无	听一? 首歌
x y	匹配x或y	听一(首 个 曲)歌
()	改变运算符的优先级	听一(首 个 曲)歌

简单实践

休眠

假设你想为你的设备设计一个休眠语句，比如：

- “若琪，没事了。”

同时，你并不希望以下语句会误触发休眠操作，

- “若琪，你没事了呀。”

此时，你可以用到 ^ \$ 两种语法：

```
^没事了$
```

调整音量

假设你需要为你的设备设计一个调整音量的语句，比如：

- ”若琪，声音小一点。“

同时，你希望这个语句具备更广的兼容性，希望被下面的语句命中，

- ”若琪，声音再小一点。“
- ”若琪，音量小一点好吗。“

那么此时，语句的正则表达式应为：

```
$volume再?$low($ah$ha)?$
```

解释

再? : 表示「再」这个字在此句的匹配时可有可无
\$low: 表示引用low这个词表 (slot)
(\$ah\$ha)? : 表示括号内的两个词表所含的内容在匹配时可有可无
句尾的\$: 表示此句需要绝对匹配。若不加句尾的\$, 此句后再跟一些无关的字也可以命中。

- Rokid OAuth 使用指南
 - 使用场景
 - 交互流程
 - 创建一个需要授权的技能
 - 一、准备工作
 - 二、OAuth相关的服务端开发

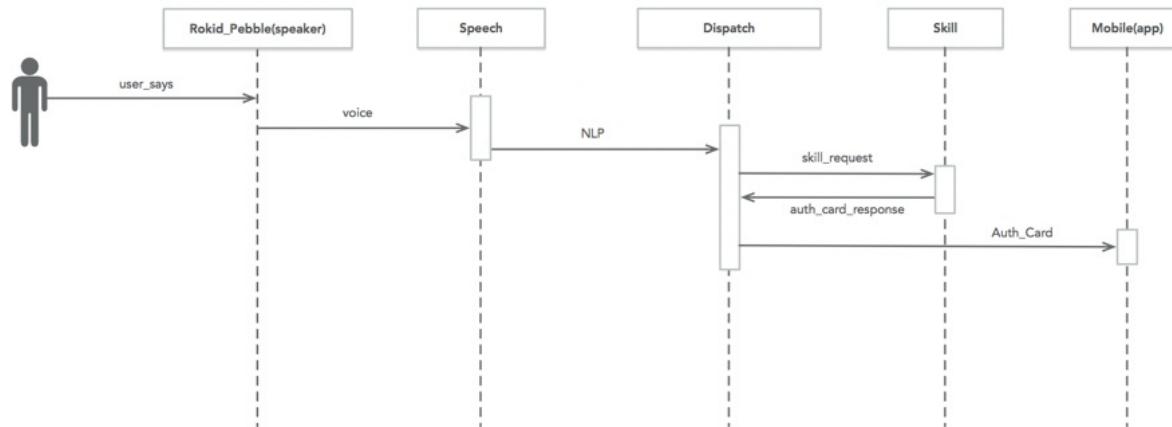
Rokid OAuth 使用指南

使用场景

当你开发的技能需要通过用户授权在喜马拉雅等第三方应用上的授权信息来提升技能的个性化体验时，可以按照本指南的流程来一步步实现。

本文档适用于第三方对接Rokid OAuth2标准授权服务。Rokid开放平台标准授权登录流程采用OAuth2标准授权码(authorization code)模式。

交互流程



第一步：用户与设备产生语音交互后，经Rokid Speech、Rokid Dispatch服务处理后，交由相应技能的服务进行进一步处理。当技能服务发现用户需要提供授权信息时，会向Rokid Dispatch推送Auth Card response，Rokid Dispatch接受到response会向Rokid App推送登录授权所需要的url。

第二步：若已登录，则获取用户的token信息。若没有授权信息则会向用户推送登录url，用户登录并授权后即可获得用户在第三方应用中的授权信息。

第三步：获取了用户的授权信息后，将会通知第三方应用已获取成功授权。

创建一个需要授权的技能

一、准备工作

以喜马拉雅为例创建一个需要授权的技能。

1. 在喜马拉雅开放平台创建一个技能，并且把oauth2授权回调地址填写到如下图所示位置处。回调地址为：<https://account.rokid.com/skills/callback.do>

我的应用 / 编辑应用

编辑应用

应用名称	test.rokid
应用简介	rokid auth test
应用类型	<input checked="" type="checkbox"/> Android <input checked="" type="checkbox"/> iOS
Android客户端包名	test.rokid
Android客户端签名 (正式环境)	rokid
Android客户端签名 (测试环境)	rokid
iOS客户端AppleID	rokid
iOS客户端BundleID	rokid
oauth2授权回调地址	请输入应用授权回调地址

2. 在喜马拉雅创建应用成功后可以获得该应用的clientID和clientSecret，如下图所示：

喜马拉雅开放平台 首页 文档 我的应用 SDK下载 N33方舟 -

我的应用

test.rokid	appkey: XXXXXXXXXX	appsecret: XXXXXXXXXX 隐藏	编辑
应用状态: 正常			
创建时间: 2017-07-13			

3. 在[Rokid开发者社区](#)新建一个技能

4. 填写技能基本信息及语音交互信息

5. 在「配置」页面填写终端服务及授权信息，如下图：

JS Engine

HTTPS

将后端服务部署在服务器上，并通过URL接收所有事件。请注意，该URL必须拥有一个可信任的SSL证书。

https://ximalaya-skill-demo.rokid.com

认证key (选填)

认证key用于访问https服务时身份校验，支持大小写英文和数字，最长不超过36位。

是否需要用户授权

是

否

选择您使用的授权过期时间格式。

Client Id

请填写第三方应用授权的Client Id。

2b45ac72420c8d46c41859c6be6eb57d

步骤2中页面的app key

Client Secret

请填写第三方应用授权的Client Secret。

00ea4f838c8dead2e020e809d3cc3501

步骤2中页面的app secret

登录 url

请填写第三方应用的登录页面url。

https://m.ximalaya.com/login?fromUri=

喜马拉雅的H5登录链接地址，fromUri为登录成功后跳转地址，如果需要此参数，必须在此写死

授权 url

请填写第三方应用的授权页面url。

https://api.ximalaya.com/oauth2/v2/authorize

喜马拉雅OAUTH 2 授权链接地址

Token url

填写用户登录的Token url。

https://api.ximalaya.com/oauth2/v2/access_token

喜马拉雅OAUTH token信息取得链接地址

绑定成功通知 url (选填)

请填写绑定成功后，第三方应用接收通知的url。

+ 增加

read

额外参数 (选填)

如 device_id=1234，如有多个参数，请用'&'连接，如 device_id=1234&myparam=abc

device_id=rokit&client_os_type=3

喜马拉雅OAUTH 2 所需的额外参数，其他第三方开放平台如果也需要额外参数请按照此格式书写以此

二、OAuth相关的服务端开发

1. 解析接收到Rokit request参数，参数信息例子如下

```
{
  "context": {
    "application": {
      "applicationId": "R69DCB84C84E4966AF690552705018EF"
    },
    "device": {
      "basic": {
        "deviceId": "0202021716000025",
        "deviceType": "98EA4B548AEB4A329D21615B9ED060E5",
        "locale": "zh-cn",
        "masterId": "13646829663",
        "timestamp": 0,
        "vendor": "910B21C0AF987D67AE7B2D50D6197421"
      }
    }
  }
}
```

```

    },
    "location": {},
    "media": {}
},
"user": {
    "accountLinkedId": "aa8afcfc9c9159ac9e06aab35cde7a475",
    "userId": "8502D6F85557559C3F62B1D40247EE24"
},
"request": {
    "content": {
        "applicationId": "R69DCB84C84E4966AF690552705018EF",
        "intent": "ROKID.INTENT.WELCOME",
        "slots": {}
    },
    "reqId": "C73901742BFA4AF7AAE0B63A24AFE4F3",
    "reqType": "INTENT"
},
"session": {
    "attributes": {},
    "newSession": false,
    "sessionId": "6A0B65595DCE4F4D87244FD2B652C0F0"
},
"version": "2.0.0"
}

```

- 上述json中 `$.context.user.accountLinkedId` 处为空时，则需要发送card要求技能用户完成授权操作，返回给rokid的reponse 如下所示：

```

{
    "version": "2.0.0",
    "startWithActiveWord": false,
    "appId": "RA66AF851FEE4CE6AF5979525AE1246A",
    "session": {
        "attributes": null
    },
    "response": {
        "action": {
            "version": "2.0.0",
            "type": "NORMAL",
            "form": "cut",
            "shouldEndSession": true,
            "voice": {
                "action": "PLAY",
                "item": {
                    "tts": "你还没有绑定喜马拉雅账号还不能使用本技能，请通过手机app绑定喜马拉雅账号以获得更加的体验"
                }
            },
            "media": null
        },
        "card": {
            "type": "ACCOUNT_LINK"
        },
        "resType": "INTENT",
        "respId": "03EA49EB0F484BA2823773F4FB9FAF19"
    }
}

```

注：

- 返回Rokid的response参数中如 `card` 处设定 `type` 为“AUTH”，则Rokid会给推送一个申请授权的push消息到用户的Rokid App上（推送功能将会在下一个版本的Rokid App实现），此时用户点击推送消息中的链接，并按照提示的操作完成授权流程。
- 上面Rokid request json中的 `$.context.user.accountLinkedId` 处，不为空时，则用户已登录第三方开放平台

- 使用步骤1中的Rokid request json参数中 `$.context.user.accountLinkedId` 完成请求第三方开放平台资源操作。

- Rokid JS Engine 使用指南
 - 使用JS脚本更快速的开发技能
 - 目录
 - 1. 本期更新-2017.08
 - 2. JS脚本基本内容
 - 2.1 固定写法部分
 - 2.2 开发者编写基本内容handlers
 - 2.3 关于调用callback
 - 2.4 语音交互中对应配置
 - 3. response配置项
 - 3.1 tts相关配置
 - 3.2 ttsWithConfirm相关配置
 - 3.3 media相关配置
 - 4. 在Rokid对象中封装的工具
 - 5. 关于调试
 - 6. 关于日志
 - 7. Sample
 - 8. Q&A

Rokid JS Engine 使用指南

欢迎使用Rokid-JS-Engine，很高兴大家可以通过编辑JS脚本来搭建技能服务。

使用JS脚本更快速的开发技能

使用JS脚本，我们的目标是帮助您更快速地构建技能，同时可以让您避免不必要的复杂性。使用JS脚本的方式有以下优势：

- 无需服务器：开发者不需要服务器去提供服务。
- 无需https服务：开发者不需要自己搭建复杂的https服务。

目录

- 1.本期更新（2017.07.11）
- 2.JS脚本基本内容
 - 2.1固定写法部分
 - 2.2开发者编写基本内容handlers
 - 2.3关于调用callback
 - 2.4语音交互中对应配置
- 3.response配置项
 - 3.1tts相关配置
 - 3.2ttsWithConfirm相关配置
 - 3.3media相关配置
- 4.在Rokid对象中封装的工具
- 5.关于调试
- 6.关于日志
- 7.Sample
- 8.Q&A

1. 本期更新-2017.08

- 新增历史日志查看入口，在“配置”页面有历史运行日志。仅保存最近5天及10万条日志数据。ps：技能上线后想查看线上运行日志须查看技能线上版本。
- 新增ttsWithConfirm交互方式。详情参见3.2节
- 新增支持使用语音合成标记语言（Speech Synthesis Markup Language，简称SSML），可以对TTS进行诸如插入一段音频、改变语速、改变发音人等更加灵活的自定义。<https://rokid.github.io/docs/2-RokidDocument/1-SkillsKit/ssml-document.html>

2. JS脚本基本内容

开发者可以利用编写JS脚本实现各自所需的技能意图函数实现不同的功能。

2.1 固定写法部分

这部分如没有特别需求，所有开发者都可以通用如下代码：

```
exports.handler = function(event, context, callback) {
  var rokid = Rokid.handler(event, context, callback);
  rokid.registerHandlers(handlers);
  rokid.execute();
};
```

我们通过Rokid对象封装了一些工具供大家使用。

- 首先，通过Rokid.handler(event, context, callback)来使用Rokid-sdk。
- 接下来，我们需要处理我们技能意图(intent)，通过rokid.registerHandlers()以简单来注册您所需的技能意图。
- 最后，通过rokid.execute()触发技能意图。

以上三步是必须的。

2.2 开发者编写基本内容handlers

其中handlers，为大家所要写的意图技能处理函数，在“配置”编写js脚本，比如：

```
var handlers = {
  'HelloWorldSample':function() {
    try{
      this.emit(':tts',{tts:'Hello World!'}, {sessionKey:sessionValue});
      //正常完成意图函数时callback
      this.callback();
    }catch(error){
      //报错时callback错误
      this.callback(error);
    }
  },
  'ConfirmSample':function() {
    try{
      this.emit(':ttsWithConfirm',{tts:'Hello World!', confirm: {confirmIntent:'confirmIntent', confirmSlot:'confirmSlot'}, {sessionKey:sessionValue}});
      //正常完成意图函数时callback
      this.callback();
    }catch(error){
      //报错时callback错误
      this.callback(error);
    }
  },
  'MediaSample': function () {
```

```

try{
    //正常完成意图函数时callback
    this.emit(':media', { itemType: 'AUDIO', url:'s.rokidcdn.com/temp/rokid-ring.mp3' },{sessionKey:sessionValue});
} catch(error){
    //报错时callback错误
    this.callback(error)
}
};


```

2.3 关于调用callback

上述中`this.callback`是在意图函数运行完成后必须调用的，且须注意`this`的指向。如不调用，则`jsEngine`将会误认为脚本未执行完毕而导致无法输出结果。

2.4 语音交互中对应配置

"HelloWorldSample"与"MediaSample"对应于"语音交互"中的intent如下：

```

{
  "intents": [
    {
      "intent": "HelloWorldSample",
      "slots": [],
      "user_says": [
        "你好"
      ],
      {
        "intent": "confirmSample",
        "slots": [],
        "user_says": [
          "询问"
        ]
      },
      {
        "intent": "MediaSample",
        "slots": [],
        "user_says": [
          "播放"
        ]
      }
    ]
  }
}


```

"语音交互"的intent, slot等request信息可在`Rokid.param`（下文有介绍）中获取。

3. response配置项

本节讲述开发者在意图函数中最终需要emit最终结果。

3.1 tts相关配置

```

this.emit(':tts',{tts:'Hello World!'}, {sessionKey:sessionValue})

```

上述语法是rokid-sdk用于同步响应对象的"tts"响应方法。注：传入":tts"法的参数对象中`tts`属性是必须的，且其类型须为`string`或`number!` `session`根据需求选择。

开发者相关字段 (tts)

this.emit(":tts",{},{})第二个参数如下 (tts相关信息配置项) :

字段	类型	默认值
type	string	NORMAL
form	string	cut
shouldEndSession	boolean	true
action	string	PLAY
tts	string	无 (必填)

this.emit(":tts",{},{})第三个参数如下 (session配置项) :

字段	类型	默认值
sessionKey	string	无 (自定义选填)
sesessionValue	string	无 (自定义选填)

3.2 ttsWithConfirm相关配置

```
this.emit(':ttsWithConfirm',{tts:'Hello World!', confirmIntent:'confirmIntent', confirmSlot:'confirmSlot'}, {ses  
sionKey:sessionValue})
```

上述语法是rokid-sdk用于同步响应对象的“ttsWithConfirm”响应方法。注：传入":ttsWithConfirm"法的参数对象中tts(类型string|number)、confirmIntent(string),confirmSlot(string)属性是必须的。session根据需求选择。

开发者相关字段 (ttsWithConfirm)

this.emit(":ttsWithConfirm",{},{})第二个参数如下 (tts相关信息配置项) :

字段	类型	默认值
type	string	NORMAL
form	string	cut
shouldEndSession	boolean	true
action	string	PLAY
tts	string或number	无 (必填)
confirmIntent	string	无 (必填)
confirmSlot	string	无 (必填)
optionWords	array	无 (选填)

this.emit(":ttsWithConfirm",{},{})第三个参数如下 (session配置项) :

字段	类型	默认值
sessionKey	string	无 (自定义选填)
sesessionValue	string	无 (自定义选填)

3.3 media相关配置

```
this.emit(':media', { itemType: 'AUDIO', url:'s.rokidcdn.com/temp/rokid-ring.mp3' },{sessionKey:sessionValue})
```

上述语法是rokid-sdk用于同步响应对象的"media"响应方法。注：传入":media"方法的参数对象中"url"和属性"itemType"是必须的！ session根据需求选择。

开发者相关字段（media）

this.emit(":media",{},{})第二个参数如下（media相关信息配置项）：

字段	类型	默认值
type	string	NORMAL
form	string	cut
shouldEndSession	boolean	true
action	string	PLAY
token	string	无
itemType(对应文档item里的type)	string	无 (必填)
url	string	无 (必填)
offsetInMilliseconds	number	0

this.emit(":media",{},{})第三个参数如下（session配置项）：

字段	类型	默认值
sessionKey	string	无 (自定义选填)
sesessionValue	string	无 (自定义选填)

具体字段定义可参见：https://rokid.github.io/docs/3-ApiReference/cloud-app-development-protocol_cn.html#3-response

4. 在Rokid对象中封装的工具

开发者可直接调用封装在Rokid对象中的所有工具方法，现有如下：

- Rokid.handler(event,contxt,callback):用于调用Rokid-sdk。
- Rokid.request(options,callback):异步请求，具体使用方法参见<https://www.npmjs.com/package/request>。
- Rokid.sync_request(method,url,options):同步请求，需把返回的数据通过Rokid.resHandler()进行buffer处理。具体使用方法参见：<https://www.npmjs.com/package/sync-request>
- Rokid.resHandler(content):buffer处理函数，如通过Rokid.sync_request请求必须通过此函数处理，再提交":tts"或":media"。
- Rokid.param:可根据“集成测试”中服务请求的数据结构获取一些数据。如：
 - slots: Rokid.param.request.content.slots
 - intent: Rokid.param.request.content.intent
 - session: Rokid.param.session.attributes
 - userId: Rokid.param.context.user.userId (只有在设备上测试才能获取，集成测试中获取不到)
- Rokid.dbServer:开发者可用此对象中的get, set, delete方法进行数据增删改查。
 - get:Rokid.dbServer.get(key, callback); key必须为string类型
 - set:Rokid.dbServer.set(key, value, callback); key,value必须为string类型。key值开发者可自定义（若每个用户

- 都可能存取数据，则推荐使用userId）。
- delete:Rokid.dbServer.delete(key,callback); key必须为string类型

数据存取基本操作如下：

```
var handlers = {
  'set':function() {
    var data = JSON.stringify([0,1,2,3,4]); //字符串存入数据库
    Rokid.dbServer.set('user001',data, (error, result) => {
      if(error) {
        this.emit(':tts',{tts: error});
        this.callback();
      }else{
        this.emit(':tts',{tts: result});
        this.callback();
      }
    });
  },
  'get':function() {
    Rokid.dbServer.get('user001', (error, result) => {
      if(error) {
        this.emit(':tts',{tts: error});
        this.callback();
        return;
      }else{
        result = JSON.parse(result); //根据取出的数据进行处理
        this.emit(':tts',{tts: result[0]});
        this.callback();
      }
    });
  },
  'delete':function() {
    Rokid.dbServer.delete('user001', (error, result) => {
      if(error) {
        this.emit(':tts',{tts: error});
        this.callback();
        return;
      }else{
        this.emit(':tts',{tts: result});
        this.callback();
      }
    });
  }
};
```

5. 关于调试

在“配置”页里，目前已支持单点测试，仅测试js应用脚本正确性。

- 配置测试用例，其中的intent和slot是需要开发者在默认用例基础上对应于“语音交互”中作相应调整。
- 调试结果中将测试用例作为request, response和log日志则是根据应用脚本产出的真实结果。

确保js应用脚本的正确性情况下，可在“集成测试”页中进行全链路集成测试。

6. 关于日志

- 在应用脚本中可调用console.log()输出你想要看的日志。
- 须保证在“命中语音指令”的情况下才可查看日志。
- 此日志仅是应用脚本执行的日志，不包括其他链路的日志。
- 尽量在脚本中采用try-catch中调用callback传回状态，详细参见本文sample。

7. Sample

```

var data = [
    "床前明月光，疑是地上霜。举头望明月，低头思故乡。",
    "白日依山尽，黄河入海流。欲穷千里目，更上一层楼。",
    "千山鸟飞绝，万径人踪灭。孤舟蓑笠翁，独钓寒江雪。",
    "松下问童子，言师采药去。只在此山中，云深不知处。",
    "向晚意不适，驱车登古原。夕阳无限好，只是近黄昏。"
];

exports.handler = function(event, context, callback) {
    var rokid = Rokid.handler(event, context, callback);
    rokid.registerHandlers(handlers);
    rokid.execute();
};

var handlers = {
    'LaunchRequest': function () {
        //这里不是意图函数完成的地方，因此不需要callback
        this.emit('GetNewFactIntent');
    },
    'GetNewFactIntent': function () {
        try{
            var factArr = data;
            var factIndex = Math.floor(Math.random() * factArr.length);
            var randomFact = factArr[factIndex];
            var speechOutput = randomFact;
            this.emit(':tts', {tts:speechOutput});
            this.callback();
        }catch(error){
            this.callback(error);
        }
    },
    'sync_NewsRequest': function () {
        //同步请求
        try{
            var result = Rokid.sync_request('GET', 'https://www.toutiao.com/hot_words/');
            //buffer处理
            result = Rokid.resHandler(result);
            var resStr = '';
            result.forEach(function(item){
                //根据返回结果进行不同处理。在此返回的一个数组，因此forEach处理。
                resStr += item + ','
            })
            this.emit(':tts', {tts: resStr });
            this.callback();
        }catch(error){
            this.callback(error);
        }
    },
    'ansync_NewsRequest': function () {
        //异步请求，注意this指向问题
        var ttsRes = '', self = this;
        Rokid.request({
            method: 'GET',
            url: 'https://www.toutiao.com/hot_words/',
            headers:{
                'User-Agent': 'request'
            }
        }, function (error, response, body) {
            if (error) {
                self.callback(error);
            }
            JSON.parse(body).forEach(function (item) {
                //根据返回结果进行不同处理。在此返回的一个数组，因此forEach处理。
                ttsRes += item + ',';
            })
        });
    }
};

```

```
        });
        self.emit(':tts', { tts: ttsRes });
        self.callback();
    });
},
'MediaRequest': function () {
    try{
        this.emit(':media', { itemType: 'AUDIO', url: 's.rokidcdn.com/temp/rokid-ring.mp3' });
        this.callback();
    }catch(error){
        this.callback(error);
    }
}
};
```

8. Q&A

Q：为什么我有时候结果输出的会包含[Object, Object]？

A: 需要注意在emit的时候写入{tts:xxx}对象时，xxx必须为string类型，如果object+string就可能出现以上现象。想要输出object的内容须用JSON.stringify()将其转换为string。

Q：为什么我的脚本一直超时？

A: 1、脚本允许运行时间为2s，如在2s内未完成则会超时。 2、需要在emit之后立即调用callback。 3、在调用emit及callback时请注意“上下文this”的指向。

Q：同步请求处理总是报Unexpected token < in JSON at position 0；

A: 开发者使用的请求地址，其返回体必须是json格式的。比如“<http://xx.com/xx.html>”这类地址是不可用的。

- 在您的技能中使用 SSML
 - 目录
 - 如何在您技能的Response中使用SSML
 - 目前可使用的SSML标签
 - audio
 - break
 - p
 - phoneme
 - prosody
 - s
 - say-as
 - speak
 - sub
 - voice
 - word
 - 特别说明

在您的技能中使用 SSML

若琪的文字转语音（Text to Speech，简称TTS，下同）会自动对标点符号做基本处理，比如在语句的 | , 。之后将会插入短停顿。能够满足基本的使用场景。

如果您需要对TTS进行诸如插入一段音频、改变语速、改变发音人等更加灵活的自定义，可以使用下述的语音合成标记语言（Speech Synthesis Markup Language，简称SSML，下同）。

目录

- 如何在您技能的Response中使用SSML
- 目前可使用的SSML标签
 - audio
 - break
 - p
 - phoneme
 - prosody
 - s
 - say-as
 - speak
 - sub
 - voice
 - word
- 特别说明

如何在您技能的Response中使用SSML

直接在返回的TTS中加入SSML格式，完整Voice Response协议请查看[Rokid 技能协议文档Voice部分](#)。

```
"item": {
  "tts": "<speak>Rokid TTS语音支持SSML。</speak>"
}
```

目前可使用的SSML标签

audio

在TTS语句中插入一段音频，可用做TTS回复的背景音，或者特定的音效回复。

参数	可选值
src	<p>背景音仅支持通过URL提供的 .wav 文件，并且需要满足以下要求。</p> <ul style="list-style-type: none"> • WAV文件必须被放在一个可靠的HTTPS终端中，且HTTPS域名需要有合法的SSL证书。 • WAV文件不能够含有个性化定制的内容以及敏感内容。 • 音频文件的长度需限制在90秒内。 • 比特率为16 bit，单声道 mono，采样率为24000 HZ。 <p>您需要使用下方推荐的转换工具将音频文件转换为符合上述要求的WAV文件。</p>

像如下例子中一样，在TTS回复中用 speak 标签包裹 audio 标签后，您的回复将带有背景音。

```
<speak>
    欢迎成为Rokid开发者。
    <audio src="s.rokidcdn.com/temp/rokid-ring.wav" />
    您会听到一段悦耳的铃声。
</speak>
```

注意：

- 单条response中最多只能插入5个音频文件
- 单挑response中的所有音频长度加总不能超过90秒。

将音频文件转换为Rokid可用的格式

您需要使用转换工具将需要插入的TTS中的音频转为WAV文件，16 bit，mono 24000HZ。推荐使用开源的命令行工具[FFmpeg](#)。下方的命令可以帮您将 <input-file> 装换为 audio 标签可用的WAV文件。

MP3转WAV:

```
ffmpeg -i <input-file.mp3> -acodec pcm_s16le -ac 1 -ar 24000 <output-file.wav>
```

PCM转WAV:

```
ffmpeg -f s16le -ar 24k -ac 1 -i <input-file.pcm> <output-file.wav>
```

break

在TTS语句中插入停顿，

参数	可选值
strength	<ul style="list-style-type: none"> • none: 无停顿。常用于消除默认停顿的部分。 • x-weak: 无停顿。和'none'效果一致。 • weak: 标准停顿。等同于逗号的效果。和'medium'效果一致。 • medium: 标准停顿。 • strong: 句子停顿。等同于句号以及`标签的效果。 • x-strong: 段落停顿。等同于`标签的效果。
time	具体停顿时长，秒的绝对值。上限10s，可以精确到小数点后1位。

```
<speak>
    这句话之后会有3.3秒钟的停顿 <break time="3.3s"/>
    然后才会继续说。
</speak>
```

p

表示一个段落。将会在段尾为TTS语音插入一个强停顿，等同于 `<break strength="x-strong">`。

```
<speak>
    <p>这是第一段话。说完之后将会停顿一会。</p>
    <p>这是第二段话。</p>
</speak>
```

phoneme

为TTS指定不同的发音。

参数	可选值
alphabet	指明音标规范 <ul style="list-style-type: none"> • py: 拼音。
ph	音标的具体序列。

```
<speak>
    我的指甲又 <phoneme alphabet="py" ph="zhang3/chang2">长长</phoneme>了。
</speak>
```

prosody

对TTS的语速、声调、音量做调整。

参数	可选值
rate	语句的发音速度 <ul style="list-style-type: none"> • 枚举值: x-slow, slow, medium, fast, x-fast • 相对值: <ul style="list-style-type: none"> ◦ +x表示加速, 最高+12 ◦ -x表示减速, 最低-12 ◦ 0表示无变化
pitch	语句的语调 <ul style="list-style-type: none"> • 枚举值: x-low, low, medium, high, x-high • 相对值: <ul style="list-style-type: none"> ◦ +x表示升高语调, 最高+12 ◦ -x表示降低语调, 最低-12 ◦ 0表示无变化
volume	声音的高低 <ul style="list-style-type: none"> • 枚举值: silent, x-soft, soft, medium, loud, x-loud • 相对值: <ul style="list-style-type: none"> ◦ +x表示调高音量, 最高+12 ◦ -x表示降低音量, 最低-12 ◦ 0表示无变化

```
<speak>
    这句话是普通音量。
    <prosody volume="x-loud">第二句话增加了音量。</prosody>。
    接下来，<prosody rate="x-slow">语速会变慢</prosody>。
    我还可以把我的音调变高。
    <prosody pitch="x-high"> 比如这样 </prosody>,
    但同样可以降低 <prosody pitch="low">变成这样</prosody>。
</speak>
```

s

表示一个句子。将会在句后插入一个停顿。等同于：

- 在句末加入句号。
- 插入一个停顿 <break strength="strong"/>

```
<speak>
    <s>这是一个句子</s>
    <s>刚才你听到了停顿</s>
    这个句子句末的句号将提供相同的停顿效果。
</speak>
```

say-as

为单词或语句指定以何种方式进行发音。

参数	可选值
interpret-as	<ul style="list-style-type: none"> • cardinal, number: 按数值发音。比如“316”，将读作“三百一十六”。 • digits: 按数字串发音。比如“666”，将读作“六六六”。 • address: 将地址缩写读作改地址的全称（暂时仅支持英文）。

```
<speak>
    昨天逛超市花了<say-as interpret-as="cardinal">216</say-as>块。
    你的房号是 <say-as interpret-as="digits">666</say-as>。
</speak>
```

speak

SSML的根元素。

```
<speak>
    这句话是没有任何SSML语法的效果。
</speak>
```

sub

将指定的单词或句子替换为特定的读音，需要搭配 `alias` 参数来使用。

参数	可选值
alias	为指定的单词或句子替换读音。

```
<speak>
    今天的比分是4 <sub alias="比">: </sub>2。
</speak>
```

voice

使用不同的发音人进行发音。

参数	可选值
name	<ul style="list-style-type: none"> normal: 正常若琪。 robot: 机器人。 sweet: 甜美女生。 crayon: 蜡笔小新。

```
<speak>
    接下来出场的是蜡笔小新,
    <voice name="crayon">大家好, 我是蜡笔小新</voice>。
</speak>
```

word

使指定的词组不被拆分以保证不会有停顿。

```
<speak>
    这双<word>球鞋</word>是C罗送给我的。
</speak>
```

特别说明

1. SSML语法不区分大小写。
2. 若一个句子中嵌套了多个SSML用法，效果将会叠加。比如对整句进行了加速和蜡笔小新的效果叠加，将会以蜡笔小新的声音快速读完该句话。
3. 若某部分的SSML出现了拼写错误，该部分会按照未添加任何SSML效果的语句来输出。比如这两个例子：`<word>球鞋</wrod>`、`<phoneme alphabet="py" ph="zhang3/canhg2">长长</phoneme>`。
4. 未在上述列出的用法暂不支持，若被您使用在句子中，若琪将按照普通语句输出。
5. 如果您在定义SSML时出现标签不完整的情况，比如：`<p>这是一个错误示例</p>`、`<s>这是第二个错误示例</s>`，该句TTS将不会被输出，并返回 `ssml syntax error`。

- Rokid语音交互指南
 - 通过Rokid开放平台，您主要可以通过语音实现2种用途
 - 用户可以通过三种基本方式与您的Rokid技能互动
 - 1. 完整意图
 - 2. 部分意图
 - 3. 无意图
 - 从用户那获取信息和提供反馈
 - 反馈作用和提示类型
 - 为用户提供反馈选项
 - 提供智能反馈推送
 - 过多音频内容造成的干扰性
 - 反馈信息的几点建议
 - 与用户进行确认
 - 处理对话错误

Rokid语音交互指南

本文档将与您分享如何使用Rokid开放平台快速为自己的产品打造一个更好的语音交互体验。

适用于：

- 为Rokid开发新的技能
- 为搭载Rokid解决方案的设备设计完整交互

如果您是为**Rokid**设备开发新技能，那您主要应该考虑用户和机器的语音交互。机器上的一些其他交互，可以复用Rokid设计好的通用方式。针对不同的需求，今后也会考虑添加用于机器的定制交互模板。

如果您是为搭载**Rokid**解决方案的设备设计完整交互，那么除了语音(Text-To-Speech,简称TTS)作为主要反馈之外，您还应该考虑音效、光效、界面（包括机器和手机上的）等其他外界因素。

通过Rokid开放平台，您主要可以通过语音实现2种用途

1. 问若琪一个问题。
 - 用户：“若琪，今天的农历日期是什么？”
 - 农历日历：“今天农历是正月初十，是石头节，适宜嫁娶，出游，设宴。”
2. 让若琪帮您做一件事。
 - 用户：“若琪，帮我把咖啡煮上。”
 - 智能家居：“好的，咖啡已经煮上了，2分钟后可以饮用。”

用户可以通过三种基本方式与您的Rokid技能互动

1. 完整意图

单个请求数句包含了技能入口词和一个意图的全部参数，能够使Rokid直接回答。

- 用户：“若琪，请告诉我现在中国天津的天气是多少度？”
- 天气：“现在天津的天气是零下-2度，会有小雪，晚上转晴，建议穿羽绒服。”

小贴士：尽可能给出完整意图的例句以培养用户给出完整意图的习惯。给用户的反馈应该最快最简洁。更多反馈内容可以使用手机卡片（后续开放）去补充。

2. 部分意图

部分意图的请求包含了技能入口词，但是意图的参数并不完全，需要用户进行额外补充。在此情况下，您应该向用户指出3个以下比较重要的选项。如果选项多于3个，则建议把最后一个选项换成“更多”，并通过手机卡片展示给用户。在用户激活“更多”后，继续展示另外三个选项作为下一步的操作。

- 用户：“若琪，帮我打开滴滴打车。”
- 滴滴打车：“好的，滴滴打车可以帮您叫车，送餐和代驾。请问您需要什么服务？”
- 用户：“打车。”
- 滴滴打车：“好的，请问要去哪里？”
- 用户：“西湖。”
- 滴滴打车：“好的，行程预估价格是XX元。需要现在帮您叫车吗？”
- 用户：“好的。”

3. 无意图

用户对您的技能不了解，只是想尝试玩一下且并没有任何真正的目的。这个时候还是应该根据部分意图让用户尽量了解您技能的目标和最优的3个选项。

- 用户：“若琪，帮我打开滴滴打车。”
- 滴滴打车：“好的，滴滴打车可以帮您叫车，送餐和代驾。请问您需要什么服务？”
- 用户：“打车。”
- 滴滴打车：“好的，请问要去哪里？”
- 用户：“...”
- 滴滴打车：(过了xx秒)“您可以告诉我想去的地方。”
 - 用户：“西湖吧”
 - 滴滴打车：“好的，....”
- 滴滴打车：(如果没有任何有效反馈)“期待你的下一次用车需求。再见。”

从用户那获取信息和提供反馈

反馈作用和提示类型

- 如果无法做到一次性给出用户答案和执行命令，则需要通过反馈从用户那里获取更多的信息。
- 反馈提示的类型有：
 - 发散型：有可能会接收到各式各样的回复。
 - 菜单型：有可能会给出用户一组选项（3-5个之间）
 - 从问型：当对话出现错误或者不明确时，从问型反馈会帮助用户从错误中恢复出来，或者引导用户一步步给出正确指令或者操作。

为用户提供反馈选项

- 明确提示用户需要反馈并给出选项内容让用户选择。选项不能超过3个，同时也要防止过多重复的词语。
- 如果用户的问题会使反馈产生很多选项（多于3个），则可以按优先级给出最重要的3个选项进行提示：
 - 网易云音乐：“您可以和我说‘打开网易云音乐’并让我‘播放热门歌曲’、‘收藏当前歌曲’或者‘播放王菲的流年’。”
- 如果一个应用比较复杂，在给出两个选项的同时可以给出“更多”“帮助”的提示入口。之后如果用户选择了“更多”，“帮助”，则显示出余下选项：
 - 网易云音乐：“您可以跟我说‘打开网易云音乐’并告诉我让我‘播放热门歌曲’、‘收藏当前歌曲’或者‘更多’。”
 - 用户：“更多。”
 - 网易云音乐：“您可以说‘播放R&B列表’、‘单曲循环播放’、‘顺序播放’、‘播放王菲的歌曲’。”
- 在给出选项之后，永远记得要让用户进行选择或者给出确认答复。

- 选项也可以是给出肯定或者否定让用户进行选择。

提供智能反馈推送

- 如果有一个技能只有一个选项，进入技能之后不要让用户选择，而是直接去做或者根据之前已有的数据去学习猜测用户心理反馈并给出具有极大正确性的反馈。
- 根据不同的用户画像提供不同类型的服务，例如老人首推的技能为医药陪护类的，而工作上班族首推的技能应该是实时资讯或者提示效率的技能等。

过多音频内容造成的干扰性

- TTS不像图像内容一样可以快速提取或者有选择性略过，所以如果用TTS作为提示应该要简洁明了。
- 如果信息不全不足以给出建议，应该一步步引导用户将信息补全，而不能一下子让用户把所有内容都给出来。
- 利用手机app作为补充，给出详细信息。可以首先给出例子，这个例子必须要是一个完整意图用来作为引导，否则难以达到好的效果。
- 确认TTS要在会产生重大影响的时候给出，不要过多使用确认：
 - 在网络上公开个人信息时候
 - 影响他人时候
 - 金钱交易

反馈信息的几点建议

- 在技能反馈的开头植入技能名称和Intent关键字会让用户有一种预先浏览了应用梗概的感觉：
 - 若琪：“现在播放来自网易云音乐的热播榜单Top10列表。”
- 如果需要显示的内容过多过长，则需要把信息分解成为一组一组的方式，每组3-5个选项，同样按照优先级把最重要信息优先显示（例如第一组3个选项）。
- 根据用户提问去判断反馈选项数量。如果用户想要的是在几个选项之间进行选择，则可以优先显示3个选项，之后继续询问用户是否需要给出更多选项；如果用户想要的是最优选项，则直接展示最优的那个，之后询问用户是否需要了解其他选项。
- 反馈信息应更多思考听觉的舒适度，而不是视觉。
- 尽量减少使用晦涩难懂的官方、技术术语。

与用户进行确认

- 确认的作用是给用户建立对机器的信心，帮助用户了解机器是懂用户的。
- 确认有两种使用途径：
 - 显性的，作为不确定时需要让用户进行反馈从而进行下一个步骤。
 - 隐性的，作为一个让用户参考自己位置坐标的提示。
- 显性确认是用明显的反问方式去跟用户确认机器听到的话然后从用户那边得到正确的回复。
- 隐性确认是用一种不易被察觉到的方式复述机器听到的关键字但同时又能保证对话的自然流畅性。隐形确认不需要用户肯定的答复但是如果用户察觉到了回复有误则有可能会自己主动自发的再复述一遍他刚才的需求。

小贴士：当机器有很大的信心给出正确回答时，推荐使用隐性确认。否则推荐使用显性确认。

处理对话错误

- 鼓励用户回答：当进入15s的拾音状态时，系统应该判断是否为人声或者杂音。如果判断为杂音则继续拾音，当15s结束系统仍处于无反馈时，系统应自动给出提示鼓励用户回答。
- 使用主动问询式的引导：当用户没有给全意图的时候，需要机器主动的一步步引导把意图补全或者结束对话。

- Rokid技能开发工具示例代码
 - 技能
 - 示例代码

Rokid技能开发工具示例代码

此部分是Rokid技能开发工具（Rokid Skills kit）的示例代码，允许开发者在**Rokid技能开发工具**中快速创建一个skill以体验Rokid提供的语音交互能力。

技能

- [我要喝咖啡](#): 一个简单的示例技能，实现最简单的问答。

示例代码

- [JS示例代码](#): 一个简单的JS示例代码合集，示范如何使用JS Engine开发Rokid技能。

- **获取认证文件**
 - **认证文件的用途**
 - **1创建认证文件**
 - 1.1 注册Rokid开发者账号
 - 1.2 创建一个设备
 - 1.3 获得认证文件
 - **2 使用方法**
 - 2.1 直接调用API时的配置方法
 - 2.2 使用C++ SDK时的配置方法
 - 2.3 在Android设备上使用TTS、Speech SDK时的配置方法

获取认证文件

认证文件的用途

用于调用Rokid开发者社区提供的各项服务时的认证工作。

1创建认证文件

1.1 注册Rokid开发者账号

在[Rokid开放平台](#)免费注册一个Rokid开发者账号。

1.2 创建一个设备

登录Rokid开发者社区，



Rokid 开发者社区 首页 文档中心 讨论区 登录 注册 简体中文

为您的产品开启语音能力

将Rokid的技能和语音服务智能集成到您的产品中
为您的应用程序和设备添加创新独特的自然语言交互体验

Rokid应用场景

智能家居 影音娱乐 应用接入

选择「语音接入」，



Rokid 开发者社区 文档中心 讨论区 DrSmile 退出

选择你需要的Rokid服务

选择「技能开发」为搭载Rokid技术的设备创造新的能力，或者选择「语音服务」为你的设备添加基于Rokid技术的语音交互能力。了解更多请参阅[开发者社区介绍](#)。

 技能开发 轻松为Rokid语音设备添加技能	 语音接入 为你的设备开启语音交互能力
---	--

选择「创建新设备」，

The screenshot shows the Rokid developer community interface. At the top, there is a navigation bar with links for '文档中心' (Documentation Center), '讨论区' (Discussion Zone), 'DrSmile' (user profile), and '退出' (Logout). Below the navigation bar, the page title is '设备列表' (Device List). A large blue banner at the top says '为你的设备注册Rokid语音服务' (Register your device for Rokid voice service) and '为你的设备开启基于Rokid的语音交互能力' (Enable your device's voice interaction capability based on Rokid). On the right side of the banner, there is a blue button labeled '创建新设备' (Create New Device), which is highlighted with a red arrow pointing towards it.

给您的设备起个名字，

The screenshot shows the 'Create New Device' setup page. The top navigation bar is identical to the previous one. The main content area has a title '给您的设备起个名字' (Give your device a name). On the left, there is a sidebar with sections: '未定义' (Not defined), '基本信息' (Basic Information) which is expanded, '认证文件' (Certification files), '设备详情' (Device details), and '服务接入' (Service access). In the main area, there is a section for '公司/个人名称' (Company/Personal Name) with the placeholder 'DrSmile'. Below it, there is a section for '设备名称' (Device Name) with the placeholder '为你的设备起一个名称, 将会展示给终端用户' (Give your device a name, it will be displayed to the end user). An input field labeled '输入设备名称' (Enter device name) is present. At the bottom, there are two buttons: '保存' (Save) and '下一步' (Next Step), with the 'Next Step' button highlighted with a red arrow.

创建认证文件,

1.3 获得认证文件

至此，获取认证文件的步骤已经结束，您已经可以根据上图处提供的 key secret account_id device_id 获取Rokid 提供的服务了。

页面中后续的「设备详情」和「服务接入」是对设备进行更多的配置，不在此进行讨论。

2 使用方法

2.1 直接调用API时的配置方法

直接通过API调用Rokid语音服务时，按照[Rokid开发者社区接口文档](#)中的要求，填入相应的参数即可。其中提到的 `key` `secret` `account_id` `device_type_id` 均通过上一步中的创建认证文件获得。

但需要注意，`device_id` 需要由开发者自行指定，由6~15位的字母和数字组成，不能含有空格和特殊符号。此ID每个设备唯一。

2.2 使用C++ SDK时的配置方法

通过调用SDK中的config方法来配置认证信息：

```
void config(String key, String value)
```

同样的，`device_id` 需要开发者自行指定，用以区分每个不同的设备。

具体请查看[Rokid客户端SDK文档中的示例](#)。

2.3 在Android设备上使用TTS、Speech SDK时的配置方法

TTS SDK

SDK解包后 `etc` 目录下文件最终需要放到安卓设备的 `/system/etc` 目录下。其中 `tts_sdk.json` 提供TTS SDK需要的配置信息，包括连接服务器的认证信息。

首先需要将创建认证文件中获得的认证信息填入到此文件中，具体需要修改如下几项：`key` `secret` `device_type_id` `device_id`

再将 `tts_sdk.json` 文件push到设备中：

```
cd etc  
adb push . /system/etc/
```

Speech SDK

使用上述TTS SDK中的方式，修改 `speech_sdk.json` 中的相关信息。

- Rokid Open Voice SDK快速集成指南
 - 简介
 - 1 目录结构
 - 2 配置
 - 3 处理权限问题
 - 4 添加系统启动项
 - 5 编译
 - 6 调试

Rokid Open Voice SDK快速集成指南

简介

Rokid开放平台SDK包含Siren、NLP、ASR、TTS几大模块。要使用Rokid开放平台的SDK，首先需要有一套 [Android 源码](#)，然后下载以下SDK模块：

- [rokid-blacksiren](#) 前端拾音降噪，寻向，音墙
- [rokid-openvoice-sdk](#) 包含NLP语音识别，语义理解；ASR语音识别；TTS语音合成
- [rokid-openvoice-sdk-deps-poco](#) SDK依赖
- [rokid-openvoice-sdk-deps-protobuf](#) (Android 4.4 不需要)

Android 6.0 使用

```
git clone https://github.com/Rokid/rokid-openvoice-sdk-deps-protobuf -b android23
```

命令获取protobuf。另外，这里为大家提供了[rokid-openvoice-sample-android](#)示例代码帮忙大家快速集成。

接下来会从以下6个步骤完整讲述如何为自己的项目部署Rokid开放平台的SDK：

- 1 目录结构
- 2 配置
- 3 处理权限问题
- 4 添加系统启动项
- 5 编译
- 6 调试

1 目录结构

```
drwxrwxr-x  9 daixiang daixiang 4096 Jun 16 11:36 rokid-blacksiren/
drwxrwxr-x 10 daixiang daixiang 4096 Jun 16 19:26 rokid-openvoice-sample-android/
drwxrwxr-x  8 daixiang daixiang 4096 Jun 15 20:26 rokid-openvoice-sdk/
drwxrwxr-x 10 daixiang daixiang 4096 Jun 15 20:26 rokid-openvoice-sdk-deps-poco/
drwxrwxr-x  4 daixiang daixiang 4096 Jun 15 20:36 rokid-openvoice-sdk-deps-protobuf/
openvoice$
```

命名建议与上图一致

`rokid-openvoice-sample-android` 与整个的业务逻辑相关，其中包含一个C进程和一个Java进程，以及MIC HAL。C进程用于为Siren提供pcm流，然后传递由Siren滤波降噪过的纯净语音给NLP或ASR，NLP或ASR经过云端处理返回结果，还有一个最重要的点就是维持Siren与NLP或ASR之间的状态。Java进程用于解析NLP或ASR返回结果，处理应用层逻辑。

2 配置

- 1、进入[Rokid开放平台](#)申请Rokid账号，已经有Rokid账号的同学可直接登录（但需进行部分信息补全）。
- 2、登录后点选「语音接入」进行设备认证信息申请。
- 3、具体做法：语音接入 > 创建新设备 > 填写设备名称 > 创建认证文件。之后您将获得：
account_id、device_type_id、device_id、secret、key

通过以上信息，您就可以获取Rokid语音服务了，然后把申请得到的账号信息写入到 `/rokid-openvoice-sample-android/etc/openvoice-profile.json` 文件中。

```
{
  'host': 'apigwws.open.rokid.com',
  'port': '443',
  'branch': '/api',
  'ssl_roots_pem': '/system/etc/roots.pem',
  'key': 'your_key',
  'device_type_id': 'your_device_type_id',
  'device_id': 'your_device_id',
  'api_version': '1',
  'secret': 'your_secret',
  'codec': 'opus'
}
```

请添加如下内容到 `/device/xxxx/p230/p230.mk`，每个人的路径是不一样的。这一步是为了让SDK编译出来的东西打包进system.img。

注意SDK存放位置

```
include openvoice/rokid-openvoice-sample-android/device/xxxx/p230/openvoice.mk
```

如果你是Android 5.0 的代码，你还需要添加如下内容，这一步后面会想办法去掉，不过目前还是必不可少的

```
/build/core/definitions.mk 最后一行添加下面这一句
include openvoice/rokid-openvoice-sample-android/build/core/definitions.mk
```

3 处理权限问题

Android 基于Linux引入了selinux，这是专门为Linux设计的一套安全机制。它有三种工作模式，Android4.2 以前一直处于Permission模式，之后便工作在enforcing模式，这就需要我们根据selinux的规范添加相应的.te文件，建议在开发阶段设回Permission，省掉一大堆权限问题。

```
/system/core/init/init.cpp

static selinux_enforcing_status selinux_status_from_cmdline() {
    selinux_enforcing_status status = SELINUX_ENFORCING;
    修改为
    selinux_enforcing_status status = SELINUX_PERMISSIVE;
}
```

4 添加系统启动项

进入到 `/device/xxxx/common/products/mbox/init.xxxx.rc` 加入如下内容，每个人的目录是不一样的。因为rokid-openvoice-sample-android中的C进程编译出来为runtime，所以我们加在这里，让init进程帮我们启动。

```
service runtime /system/bin/runtime
    class main
    user root
    group root root
```

5 编译

如果你的开发板使用的是USB MIC，你需要定义宏 **USB_AUDIO_DEVICE**，具体操作如下：

```
/rokid-openvoice-sample-android/Android.mk

LOCAL_CPPFLAGS += -DUSB_AUDIO_DEVICE
```

如果你不使用Java或TTS接口，请修改如下内容，否则你需要使用

```
git clone https://github.com/Rokid/rokid-openvoice-sdk-deps-fastjson.git
```

获取依赖Jar包。

```
/rokid-openvoice-sdk/Android.mk

include $(LOCAL_PATH)/JavaLibrary.mk
修改为
#include $(LOCAL_PATH)/JavaLibrary.mk
```

单编：

先编译 Andorid，然后使用mm依次编译：rokid-openvoice-sdk-deps-protobuf，rokid-openvoice-sdk-deps-poco，rokid-openvoice-sdk，rokid-blacksiren，rokid-openvoice-sample-android。

整编：

```
./build/envsetup.sh
lunch <your config>
make aprotoc -j8
make -j8
```

6 调试

以上是一套完整的集成流程，到目前为止已经完成了50%的工作。调试主要围绕MIC这块，拾音以及语音识别准确度都围绕在这里，建议大家分两步：

第一步： 测试硬件是否OK，我们使用tinycap命令抓取pcm裸数据，这个命令是从kernel层直接拿数据，不经过HAL层，这样可以定位问题。然后用Audacity工具分析波形，要想识别效果好就不能有杂音出现。

```
adb shell
stop runtime
cd /sdcard
tinycap file.wav [-D card] [-d device] [-c channels] [-r rate] [-b bits] [-p period_size] [-n n_periods]

card      : 选择一个MIC设备，可以执行`cat /proc/asound/cards` 下查看
device    : 0代表输入，1代表输出
```

```

channels      : 通道数
rate          : 采样率, Siren采样率最小必须为48K
bits          : 比特率, 32/24/16
period_size   : 周期
periods       : 点

例:
tinycap 1.wav -D 0 -d 0 -c 8 -r 48000 -b 32 -p 1024 -n 8
adb pull sdcard/1.wav ./

```

第二步：拾音，也就是Siren，这里教大家做一个简单的配置。

1. ./rokid-blacksiren/resource/blacksiren.json

```

mic_num        : 总的MIC个数(aec没有则不计算在内)
mic_channel_num: 对应的channel数, 一般是一对一
mic_sample_rate: 大于等于48000

```

```
{
  "basic_config": {
    "mic_num":8,
    "mic_channel_num":8,
    "mic_sample_rate":48000,
    "mic_aec":1
  }
}
```

2. ./rokid-blacksiren/resource/cn/r2ssp.cfg

```

audio.rate  : 等同于上面的 sample_rate
aec         : (重要)全称为Acoustic Echo Chancellor, 可以消除各种延迟的回声, 这个需要内核做修改, 大家可以关闭掉(置0)
aec.mics    : 实际使用的MIC通道(不包括aec)
aec.ref.mics: aec的虚拟通道(aec是关闭可以不用管)

```

```
r2ssp.audio.rate=48000      第一步: 测试硬件是否OK, 我们使用tinycap命令抓取pcm裸数据, 这个命令是从kernel中直接调用的, 所以可以定位问题。然后用Audacity工具分析波形, 要想识别效果好就不能有杂音出现。
r2ssp.aec=1
r2ssp.aec.mics=0,1,2,3      adb shell
r2ssp.aec.ref.mics=4,5      stop runtime
```

```

mic.num        : 总的MIC个数(aec没有则不计算在内)
mic.pos(0...n): MIC方位, 程三维立体型, 需要游标尺精确测量, 寻向角度与此相关, 不影响拾音。如图0--3为实际MIC, 需要精确方位, 后面全部都是AEC虚拟通道, 不需要方位。大家根据自己的硬件进行配置(详细测量方式见附件1)
sl.poc        : 实际使用的MIC通道(不包括aec)
bf.mics       : 同上

```

```

r2ssp.mic.num=8
r2ssp.mic.pos.0=0.00000000,0.03000000,0.00000000
r2ssp.mic.pos.1=-0.03000000,0.00000000,0.00000000
r2ssp.mic.pos.2=0.00000000,-0.03000000,0.00000000
r2ssp.mic.pos.3=0.00300000,0.00000000,0.00000000

r2ssp.mic.pos.4=0.0f,0.0f,0.0f
r2ssp.mic.pos.5=0.0f,0.0f,0.0f
r2ssp.mic.pos.6=0.0f,0.0f,0.0f
r2ssp.mic.pos.7=0.0f,0.0f,0.0f

r2ssp.sl.mics=0,1,2,3
r2ssp.bf.mics=0,1,2,3

```

```

r2ssp.mic.num=8
r2ssp.mic.pos.0=0.00000000,0.03000000,0.00000000
r2ssp.mic.pos.1=-0.03000000,0.00000000,0.00000000
r2ssp.mic.pos.2=0.00000000,-0.03000000,0.00000000
r2ssp.mic.pos.3=0.00300000,0.00000000,0.00000000

r2ssp.mic.pos.4=0.0f,0.0f,0.0f
r2ssp.mic.pos.5=0.0f,0.0f,0.0f
r2ssp.mic.pos.6=0.0f,0.0f,0.0f
r2ssp.mic.pos.7=0.0f,0.0f,0.0f

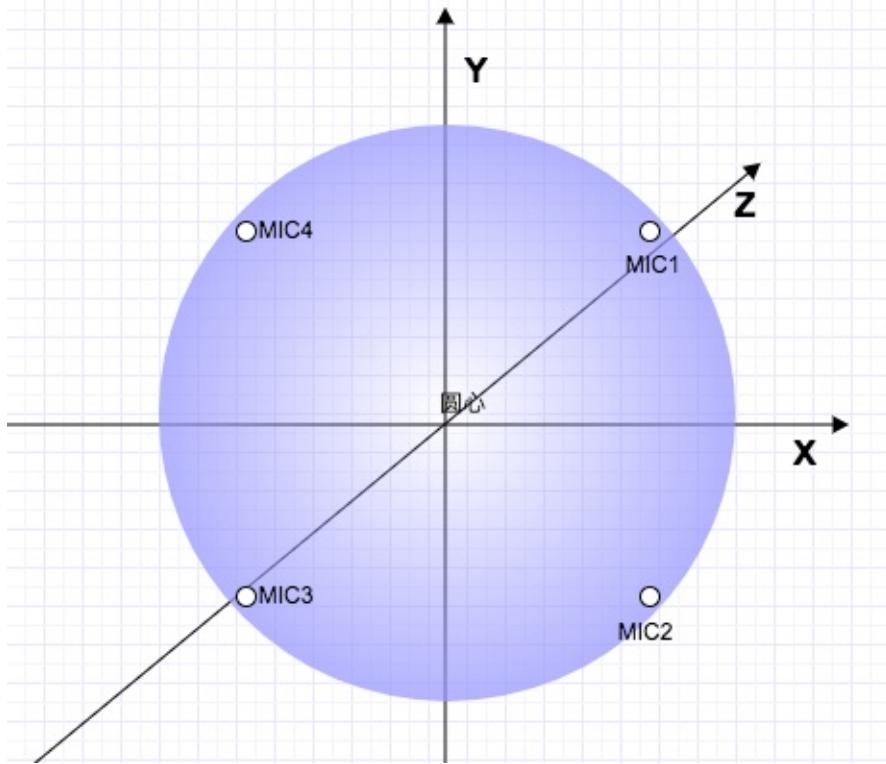
r2ssp.sl.mics=0,1,2,3
r2ssp.bf.mics=0,1,2,3

```

到这为止整个集成就大告完成，后续优化主要针对MIC硬件，以及添加AEC来改善拾音，下一章就会讲解编译中普遍遇到的问题。

附件1:

如图，MIC可以呈三维立体设计，但一般都是二维的。不管怎样，只要找到圆心，顺时针测量从圆心到MIC之间的距离。例如上面pos.0的坐标：x=0.00000000, y=0.03000000, z=0.00000000, 单位为“米”。



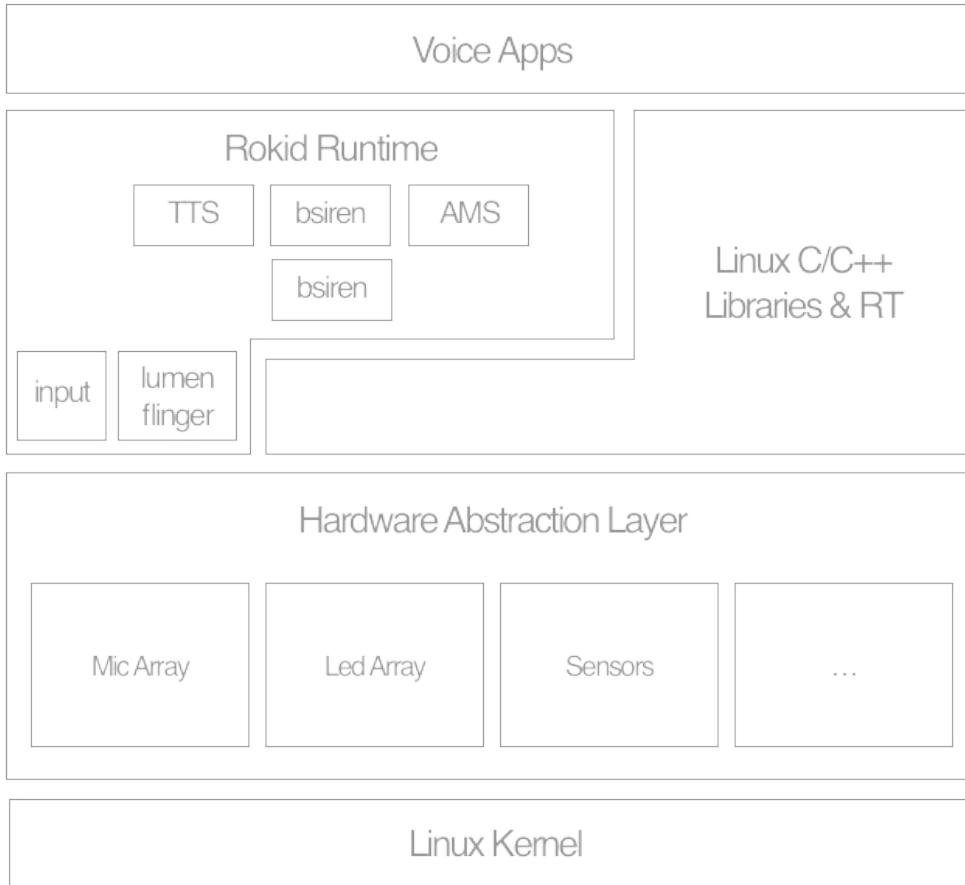
- 开发者社区 Linux 设备开发引导
 - 支持的开发板
 - 系统架构图
 - 系统特性
 - 编译
 - Amlogic芯片
 - Rokid对厂商代码的修改

开发者社区 Linux 设备开发引导

支持的开发板

芯片厂商	芯片型号	开发板内部代号	编译框架	状态
Amlogic	A112	nana_l	buildroot	支持
Amlogic	S905D	nana_t	buildroot	支持
Amlogic	S905D	nana_t2	buildroot	支持
Amlogic	S905D	rm101	buildroot	支持
Amlogic	S905D	rp102	buildroot	支持
Amlogic	A113	banban_m	buildroot	开发中

系统架构图



系统特性

模块	功能介绍	代码位置	实现语言	状态
AMS	Application Manager Service Rokid语音应用的生命期调度、事件分发框架	robot/openvoice/ams	C	开发中
Speech SDK	封装了与Rokid云服务交互协议，包括ASR、NLP、TTS等开发服务	robot/openvoice/speech	C++	支持
Blacksiren SDK	输入麦克风数据，经内部拾音算法及云端服务（调用SpeechSDK），输出语音识别结果、各种拾音事件	robot/openvoice/blacksiren	C++	支持
openvoice_proc Service	将Blacksiren封装成服务	robot/openvoice_proc	C++	支持
Linux systemd	Linux Init System -- systemd		C/C++	支持
openvoice app Zygote	由该进程负责启动所有语音应用	robot/service/zygote	C/C++	支持
PulseAudio	提供Audio服务及路由机制	buildroot原生	C	支持

TTSFlinger Service	提供设备端的语音转文字服务	robot/services/ttsflinger	C++	支持
Lumenflinger	提供灯光渲染服务	robot/services/lumenflinger	C++	支持
BTFlinger	提供蓝牙功能	robot/services/btflinger	C	支持
系统电量服务	系统电量服务		C/C++	支持
应用包管理	应用安装升级		C/Node.JS	支持
OTA	系统升级		C	支持
蓝牙配网服务	提供通过蓝牙来配置Wifi网络		C	支持
热点配网	提供通过设备开启热点方式来配置网络		C	开发中
CloudClient C 版	实现Cloud Skill功能 / 应用支持（如天气、新闻、音乐等），适用小内存系统		C	开发中
CloudClient Nodejs版本	实现Cloud Skill功能 / 应用支持（如天气、新闻、音乐等Cloud 应用）、适用大内存系统		C/Node.JS	支持
系统音量控制	提供系统及的音量控制服务		C/C++/Node.JS	支持
灯光寻向指示	提供唤醒、对话时的寻向指示		C/C++/Node.JS	支持
媒体播放器	媒体播放器	robot/external/librplayer	C/C++	支持
蓝牙音乐应用	媒体播放器	robot/apps/xxx	C/C++	支持
聊天应用	提供系统语音聊天服务		C/C++/Node.JS	支持
Android ADB	提供ADB支持，方便开发		C++	支持
Android HAL	提供Android HAL，方便实现 Mic Array, Led Array, Sensor等		C++	支持
Android Binder	提供进程间通讯机制		C++	支持
Input Manager	提供按键、触摸、鼠标事件 SDK		C++	支持

编译

Amlogic芯片

Rokid对厂商代码的修改

U-Boot

修改了厂商代码，支持Rokid的板级配置目录

Kernel

修改了厂商代码，支持Rokid多型号板子的DST配置目录

BuildRoot Package

扩展的buildroot_external

rokid_br_external 是Rokid通过BuildRoot的external机制，将Rokid提供的包或第三方库的编译配置放在此处

FFWT

需要使用Rokid对该包的配置，核心的语音算法会依赖该动态库

NE10

需要使用Rokid对该包的配置，核心的语音算法会依赖该动态库

NodeJS

需要使用Rokid对该包的配置，CloudClient NodeJS版会依赖该配置

TinyPlay

需要使用Rokid对该包的配置，目前Mic Array使用了Tinyplay接口读取数据，而Amlogic源码释放出的Tinyplay 版本存在超过2个channel时读取音频数据会存在Bug，所以需要使用Rokid目前配置的版本。

编译指令

目前支持64位版本，32后续会考虑支持。

```
source rokid_br_external/build/setenv.sh
```

输出

```
Environment setting is OK!
Just type 'lunch' and you will get a list of choices, or you can type 'lunch [choice]' to lunch directly.
```

```
lunch
```

输出

```
You are building on Linux
echo Lunch menu... pick a combo:
1. nana_t_s905d_release
2. nana_l_a112_release
3. rm101_s905d_release
4. rp102_s905d_release
5. banban_m_a113_release
6. nana_t2_s905d_release

Which would you like? [2]
```

目前使用注意事项

关于刷机

如果你的设备是mini没有接串口，无法在U-Boot下进入刷机模式,请看如下指示：刷机镜像是：
output/rm101_s905d/images/aml_upgrade_package.img 先打开PC端的windows版的amlogic刷机工具，Amlogic也提供了linux版本。

mini让他进入update模式的指令（无串口，无法进入uboot的情况下），adb shell登录到设备上，敲如下指令：

```
fw_setenv bootcmd "run update"
reboot
```

之后PC端的刷机程序就会检测到设备进入刷机模式，按软件的刷机提示刷机即可。

关于配置网络

由于目前配网模块还在开发中，需要手动配置：

```
vi /etc/wpa_supplicant.conf
```

将

```
4 network={
5   key_mgmt=NONE
6 }
```

修改为：

```
network={
  ssid="你的WiFi网络名"
  psk="你的密码"
}
```

然后启动网络

```
/etc/init.d/S42wifi stop
/etc/init.d/S42wifi start
```


- TTSFlinger

TTSFlinger

TTSFlinger 使用 C/S 架构，服务端默认启动，开发者需要通过调用客户端接口与 TTSFlinger 通讯。客户端头文件目录位于：

```
$ cat /usr/include/rokid/tts/tts_client.h
```

除了要 `include` 头文件外，还需要链接 `/usr/lib/librktts.so`。

API

```
int tts_init();
// tts初始化接口

int tts_speak(const char* content, void* userdata);
// tts播放接口，参数content: 要播放内容, userdata: 用户私有数据

void tts_cancel(int id, void* userdata);
// tts取消借口，参数id: 代表要取消的tts的id, userdata: 用户私有数据

void tts_set(struct tts_callback *func);
// tts设置的接口，参数func代表要实现的接口

void tts_destory();
// tts退出接口
```

回调接口

```
void (*onStart)(int id, void* userdata);
// tts服务开始事件

void (*onCancel)(int id, void* userdata);
// tts服务取消事件

void (*onComplete)(int id, void* userdata);
// tts服务结束事件

void (*onError)(int id, int err, void* userdata);
// tts服务出错事件
```

示例

```
#include <rokid/tts/tts_client.h>

void onStart(int id, void* userdata) {
    // todo
}

void onCancel(int id, void* userdata) {
    // todo
}

void onComplete(int id, void* userdata) {
    // todo
}

void onError(int id, int err, void* userdata) {
    // todo
}
```

```
int main(int argc, char** argv) {
    int id;
    int complete;
    struct tts_callback func = {
        onStart, onCancel, onComplete, onError
    };
    tts_init();
    tts_set(&func);
    id = tts_speak((const char *)argv[1], &complete);
    return 0;
}
```


- Node.js 开发者生态

Node.js 开发者生态

- 本地开发者工具
- 技能框架介绍
 - 工程结构
 - 安装
 - 调试
 - 测试
- 系统接口
 - 语音合成
 - 媒体播放
 - 灯光控制

RokidOS 提供了完整的 Node.js 语音技能接口及框架。

本地开发者工具

RokidOS 为 Node.js 开发者提供了 `rokidos-cli` 包：

```
$ npm install rokidos-cli -g
```

语音技能框架介绍

RokidOS 内置了一套完整的语音技能框架，该框架遵循基本的 Node.js 规则，只需在根目录创建 `app.js` 及 `package.json` 即可。

工程结构

RokidOS 本地技能的工程结构如下：

`package.json`

```
$ npm init
```

可通过上述命令生成一个通用模版，然后更新文件的如下字段：

- `name`：应用名，会作为安装后目录的名字
- `version`：应用版本号，请开发者遵守语义化版本更新你的应用
- `metadata`：应用安装所需的信息
 - `type`：应用类型，可选：`scene` 和 `cut`
 - `skills`：使用该字段配置应用需要处理的技能 ID

`app.js`

下面是一个示例的入口脚本：

```
'use strict';

const app = require('@rokid/ams')();
const tts = require('@rokid/tts');
const player = require('@rokid/player');
```

```

app.on('create', function() {
    // the app is created
});

app.on('resume', function() {
    // the app is resume
});

app.on('pause', function() {
    // the app is pause
});

app.on('request', function(data, action, event) {
    // upstream voice data
});
app.start();

```

app 对象支持如下事件：

事件名	描述
create	在应用第一次被唤起时调用
restart	在应用再次被创建时调用
resume	在应用被置为栈顶恢复时调用
pause	在应用被其他应用占用时调用
stop	在应用被停止时调用
destroy	在应用被销毁时调用

同时也支持下列 BlackSiren 下发的事件如下：

- vad_start
- vad_data
- vad_end
- vad_cancel
- wake_cmd
- sleep

更复杂的例子可以参考 `/opt/apps/` 目录下的内置应用。

安装

```

$ cd /path/to/app/directory
$ rokid install

```

如果你想单独构建出应用包，则使用 `rokid build` 即可。

调试

首先确保你已正确安装应用，并且已通过 USB 连接到你要调试的 RokidOS 设备：

```

$ rokid debug

```

测试

在项目目录下创建 `tests` 目录，然后在该目录下创建测试样例如下：

```
test('test voice play', (t) => {
  t.send('asr', 'nlp', 'action');
  // t.assert([event], [value], [callback]);
});
```

在这里返回的对象 `t` 中，分别提供了两个方法：`send()` 和 `assert()`。

`send(asr, nlp, action)`

该方法用于向测试设备发送指定的 `asr`、`nlp` 对象和 `action` 对象。

`assert(event, val, callback)`

该方法用于在 `t.send()` 之后，捕获来自测试设备发送的探针事件，现支持的事件如下：

- `voice`
- `media.start`
- `media.stop`

`callback` 在事件发生后被调用，因此你可以在 `media.stop` 的回调中，继续执行一些链式用例。例如：

```
test('chainable case example', (t) => {
  t.send(asr, nlp, action);
  t.assert('media.stop', true, () => {
    t.send(asr2, nlp2, action2);
    t.assert('voice', 'some text to speak');
  });
});
```

当你写完测试用例后，使用下面的命令来执行测试：

```
$ rokid test
```

系统接口

为了让开发者完成各种开放性的技能需求，RokidOS 为本地技能开发者提供了如下系统基础接口：

语音合成

样例：

```
const tts = require('@rokid/tts');
tts.play('text to speech');
```

该模块提供以下方法：

- `play(text, callback)` 让若琪说出指定的文字内容
 - `{String} text` 需要让若琪说的文字，支持 `SSML`
 - `{Function} callback` 当 `tts` 完成、取消或其他方式结束时，会通过这个回调函数通知用户
- `stop()` 停止播放当前内容

媒体播放

样例：

```
const player = require('@rokid/player');
player.play(url);
player.stop();
player.resume();
player.pause();
```

该模块提供以下方法：

- `play(url)` 播放某个 `url` 的媒体
 - `{String} url` 媒体地址
- `stop()` 停止当前应用中所有的媒体播放
- `pause()` 暂停当前应用中所有的媒体播放
- `resume()` 恢复当前应用中所有的媒体播放

灯光控制

样例：

```
const light = require('@rokid/lumen');
light.fill('red', 'all');
light.update({
  [1]: 'red',
  [2]: 'green',
});
```

该模块提供以下方法：

- `fill(color, sets)` 向指定的 LED 填充单个颜色
 - `{String} color` RGB 颜色值，支持的格式：#000000、red
 - `{Array} sets` 表示需要填充颜色的 LED 坐标，目前 LED 阵列为一维
- `update(dataWithSets)` 更底层的填充方法，一般用于更复杂的灯光渲染需求
 - `{Object} dataWithSets` 该对象是 LED-Color 的映射关系
- `gradients(dataWithSets, ms)` 提供灯光的渐变控制

Cloud App 开发协议

Rokid 开放平台

版本: 1.0.0-alpha

大纲

- 简介
 - 一些概念
- Request
 - 协议概览
 - Session定义
 - Context定义
 - Request定义
- Response
 - 协议概览
 - Action定义

1. 简介

本文是对在[Rokid开放平台](#)上开发CloudApp的协议的详细描述。

1.1 一些概念

在了解本文所描述协议之前，需要对一下概念作如下说明：

- CloudApp - 在[Rokid开放平台](#)上接入的云端应用，可以理解为遵循本文所描述的协议开发的某种云端服务或小应用。
- CloudDispatcher - 用于向CloudApp传递请求和分发CloudApp返回结果的模块。
- CloudApiClient - 用于处理CloudApp返回结果的设备端的执行容器。
- RokidMobileSDK - 与[Rokid开放平台](#)向关联的手机端SDK，用于对CloudApp的信息扩展展示或第三方授权。
- TTS - Text To Speech的缩写，这是机器人的语音表达方式。

2. Request

Request is used to fetch response from CloudApps, which is sent by CloudDispatcher. Currently IntentRequest and EventRequest are available. IntentRequest is created according an NLP intent. And EventRequest is created when an event occurs.

Request 是CloudDispatcher产生的用于向CloudApp获取对应返回结果的请求。目前有两种类型的请求：一种是IntentRequest，一种是EventRequest。IntentRequest 是根据语音识别和语义理解（NLP）的结果创建的，其中会带有（NLP）的信息。EventRequest是在当有某种事件发生时产生的，并通过CloudDispatcher转发给当前CloudApp，比如当某个TTS播放结束的时候会产生一个TTS结束的事件，当前CloudApp可以选择处理或者不处理。

2.1 Request 协议预览

2.1.1 Request Header

为了保证Https链接访问的安全性，在http请求的header中增加了Singature来校验请求是否来源于Rokid。

Http Header中相关内容的示例如下：

```
Content-Type: application/json; charset=utf-8
Signature: DAF1E1062C21E3BB80A55BA32F41D935
```

您可以在「技能开发的-配置」中，填写您自定义的认证key（支持大小写英文和数字，最长不超过36位）。

认证key的生成规则为： Signature = MD5(Secret + MD5(Body))

2.1.2 Request Body

*Request*的整体协议定义如下所示：

```
{
  "version": "2.0.0",
  "session": {
    "sessionId": "D75D1C9BECE045E9AC4A87DA86303DD6",
    "newSession": true,
    "attributes": {
      "key1": "value1",
      "key2": "value2"
    }
  },
  "context": {
    "application": {
      "applicationId": "application id for requested CloudApp"
    },
    "device": {
      "basic": {
        "vendor": "vendor id",
        "deviceType": "device type",
        "deviceId": "010116000100",
        "locale": "zh-cn",
        "timestamp": 1478009510909
      },
      "screen": {
        "x": "640",
        "y": "480"
      },
      "media": {
        "state": "PLAYING / PAUSED / IDLE"
      },
      "voice": {
        "state": "PLAYING / PAUSED / IDLE"
      },
      "location": {
        "latitude": "30.213322455923485",
        "longitude": "120.01190010997654"
      }
    },
    "user": {
      "userId": "user id string"
    }
}
```

```

    },
    "request": {
        "reqType": "intent / event",
        "reqId": "010116000100-ad1f462f4f0946ccb24e9248362c504a",
        "content": {
            "applicationId": "com.rokid.cloud.music",
            "intent": "play_random",
            "slots": {}
        }
    }
}

```

- **request** - 协议中真正代表此次Request的实体，会明确给出请求的类型**RequestType**和请求的内容**RequestContent**。

2.2 Session定义

Session 向所请求的CloudApp表明了会话的信息，每一次对CloudApp的请求都会产生会话信息，会话的信息和状态由开放平台的系统更新。**Session**也提供了**attributes**字段留给CloudApp来保存一些上下文信息。具体阐述如下：

```

"session": {
    "sessionId": "D75D1C9BECE045E9AC4A87DA86303DD6",
    "newSession": true,
    "attributes": {}
}

```

字段	类型	可能值
sessionId	string	每次会话的唯一ID，由系统填充
newSession	boolean	true / false (由系统填充)
attributes	key-value map	一个string-string map

- **sessionId** - 每次会话的唯一ID，由系统填充
- **newSession** - 向CloudApp表明此次会话是新的会话还是已经存在的会话
- **attributes** - 为CloudApp提供**attributes**字段留保存上下文信息的字段

2.3 Context定义

Context 向所请求的CloudApp提供了当前的设备信息，用户信息和应用状态，用以帮助CloudApp更好的去管理逻辑，状态以及对应的返回结果。

```

"context": {
    "application": {},
    "device": {},
    "user": {}
}

```

字段	类型	可能值
application	ApplicationInfo object	ApplicationInfo对象，目前只有应用ID

device	DeviceInfo object	<i>DeviceInfo</i> 对象
user	UserInfo object	<i>UserInfo</i> 对象

2.3.1 ApplicationInfo

ApplicationInfo 包含了当前的应用信息，目前只有**applicationId**可用。

```
"application": {
    "applicationId": "application id for requested CloudApp"
}
```

字段	类型	可能值
applicationId	string	应用ID字符串

- **applicationId** - *CloudApp*在*Rokid*开放平台上的唯一ID.

2.3.2 DeviceInfo

DeviceInfo 是对此次请求发生时当前设备信息的描述。

```
"device": {
    "basic":{},
    "screen":{},
    "media": {},
    "voice": {},
    "location": {}
}
```

字段	类型	可能值
basic	BasicInfo object	<i>BasicInfo</i> 对象
screen	ScreenInfo object	<i>ScreenInfo</i> 对象
media	MeidaStatus object	当前设备上 <i>CloudAppClient</i> 的 <i>MediaPlayer</i> 状态
location	LocationInfo object	当前设备的地理位置信息

- **basic** - 展示了当前设备的基础信息，主要包含设备制造信息、时间信息、国家文字信息。
- **screen** - 展示了当前设备的屏幕信息，主要包含屏幕的分辨率信息。
- **meida** - 向*CloudApp*表明当前设备上*CloudAppClient*中的*MediaPlayer*的状态信息。
- **location** - 向*CloudApp*提供当前设备的地理位置信息。

2.3.2.1 BasicInfo

```
"basic":{
    "vendor": "vendor id",
    "deviceType": "device type",
    "deviceId": "010116000100",
    "locale": "zh-cn",
    "timestamp": 1478009510909
}
```

字段	类型	可能值
vendor	string	注册生产商ID
deviceType	string	该生产商设定的设备型号

deviceId	string	该型号下的设备ID
locale	string	国家及语言, 标准 <code>locale</code> 格式
timestamp	long	当前时间, <code>unix timestamp</code>

- **vendor** - 生产商ID, 通过在网站注册生产商生成, 保证全局唯一
- **deviceType** - 设备型号ID, 通过在网站注册设备型号生成, 保证生产商内部唯一
- **deviceId** - 设备ID, 由生产商自行生成, 保证设备型号内部唯一
- **locale** - 国家及语言, 采用标准`locale`格式, language-country
- **timestamp** - 当前时间, 使用设备当前的时间戳, unix timestamp

2.3.2.2 ScreenInfo

当前设备的显示设备信息:

```
"screen": {
    "x": "640",
    "y": "480"
}
```

字段	类型	可能值
x	string	X 方向上的像素大小
y	string	Y 方向上的像素大小

- **x** - X 方向上的像素大小
- **y** - Y 方向上的像素大小
- 根据给出的屏幕分辨率信息, 通常来讲, 如果 **x** 比 **y** 大, 那么该屏幕会被认为是横屏 **landscape**, 反过来则是竖屏 **portrait**.

2.3.2.3 MediaStatus

当前设备上CloudAppClient中MediaPlayer的状态:

```
"media": {
    "state": "PLAYING / PAUSED / IDLE"
}
```

字段	类型	可能值
state	string	<code>PLAYING / PAUSED / IDLE</code>

- **state** - 表明当前播放状态. 当前有 **PLAYING**、**PAUSED** 和 **IDLE** 三种状态可用。
 - **PLAYING**: 代表当前有媒体正在播放;
 - **PAUSED**: 代表当前媒体被暂停, 可以执行继续播放 (`RESUME`) 操作;
 - **IDLE**: 代表当前媒体播放器为空闲状态, 没有任何媒体数据。

2.3.2.4 VoiceStatus

参见上述[2.3.2.3 MediaStatus](#)

2.3.2.5 LocationInfo

```
"location": {
    "latitude": "30.213322455923485",
    "longitude": "120.01190010997654"
}
```

当当前设备存在地理位置信息时会通过`location`提供给CloudApp。基于地理位置的CloudApp可以根据此信息来处理逻辑。`location` 信息目前包括 **纬度(latitude)** 和 **经度(longtitude)**。

2.3.3 UserInfo

`Userinfo` 展示了与当前设备绑定的用户信息，通常是设备对应手机应用的账号。

```
"user": {
  "userId": "user id string"
}
```

字段	类型	可能值
userId	string	用户ID

2.4 Request定义

`Request` 是当前请求的真正内容：

```
"request": {
  "reqType": "INTENT / EVENT",
  "reqId": "010116000100-ad1f462f4f0946ccb24e9248362c504a",
  "content": {}
}
```

字段	类型	可能值
reqType	string	<code>INTENT / EVENT</code>
reqId	string	当前请求的唯一ID
content	request content object	<code>IntentRequest</code> 或 <code>EventRequest</code> 的对象

- **reqType** - 表明请求的类型：`INTENT` 和 `EVENT` 分别对应 `IntentRequest` 和 `EventRequest`。
- **reqId** - 每次请求都会对应一个唯一ID用以区分每一次的请求。请求ID将会与返回ID一一对应。
- **content** - `IntentRequest` 和 `EventRequest` 对应的具体内容，下面会具体介绍。

2.4.1 IntentRequest

`IntentRequest` 是基于 `NLP` 的结果产生的请求，其中包括了 `NLP` 的所有信息：**ApplicationId**、**Intent** 和 **Slots**。`IntentRequest` 将会发给对应的 `CloudApp` 根据 `intent` 和 `slots` 进行相应的逻辑处理。

```
"content": {
  "applicationId": "com.rokid.cloud.music",
  "intent": "play_random",
  "slots": {}
}
```

字段	类型	可能值
applicationId	string	<code>CloudApp</code> 对应的 <code>applicationId</code>
intent	string	<code>CloudApp</code> 对应的 <code>nlp intent</code>
slots	string	<code>CloudApp</code> 对应的 <code>nlp slots</code>

- **applicationId**, **intent** 和 **slots** 均为 `NLP` 结果的基本元素。分别表明了一句话所代表的领域，意图和完成这个意图

所需要的参数。

2.4.2 EventRequest

当CloudAppClient在执行中发生了一个事件，则会产生一个EventRequest。CloudApp 可以根据自己的需要选择处理或者不处理当前收到的事件。

```
"content": {
    "event": "Media.NEAR_FINISH",
    "extra": {
        "key1": "value1",
        "key2": "value2"
    }
}
```

字段	类型	可能值
event	string	事件类型
extra	string-string map	自定义字段，目前暂无定义，作扩展用

- **event** - 表明了是具体的事件类型。
 - **Voice.STARTED** - 当Voice开始播放时发生
 - **Voice.FINISHED** - 当Voice停止时发生，此处停止可能是被打断，可能是播放完成，也可能是播放失败，但都作为统一的事件抛出。
 - **Media.START_PLAYING** - 当MediaPlayer开始播放时发生。
 - **Media.PAUSED** - 当MediaPlayer中途停止时发生。
 - **Media.NEAR_FINISH** - 当播放内容结束前15秒时发生，当总时长不足15秒时，会在**Media.START_PLAYING***后发生。
 - **Media.FINISHED** - 当播放内容结束时发生。
 - 更多的事件会在未来的版本更迭中给出
- **extra** - 针对media类型的eventrequest支持如下扩展字段：

```
"content": {
    "event": "Media.PAUSED",
    "extra": {
        "media": {
            "token": "MediaItem里面的token",
            "progress": "当前进度",
            "duration": "音频文件的总长度"
        }
    }
}
```

3. Response

根据之前的描述，Response是 CloudApp 向客户端的返回结果。

3.1 协议概览

整体协议示例如下：

```
{
  "version": "2.0.0",
  "session": {
    "attributes": {
      "key1": "value1",
      "key2": "value2"
    }
  }
}
```

```

        "key2": "value2"
    },
},
"response": {
    "action": { // for Rokid device

        "version": "2.0.0",

        "type": "NORMAL / EXIT",

        "form": "scene/cut/service",

        "shouldEndSession": true,

        "voice": {
            "action": "PLAY/PAUSE/RESUME/STOP",
            "item": {
                "tts": "tts content"
            }
        },
        "media": {
            "action": "PLAY/PAUSE/RESUME/STOP",
            "item": {
                "token": "xxxx",
                "type": "AUDIO/VIDEO",
                "url": "media streaming url",
                "offsetInMilliseconds": 0
            }
        },
        "display": {
            // TBD
        },
        "confirm": {
            "confirmIntent": "nlp intent to confirm",
            "confirmSlot": "nlp slot to confirm",
            "optionWords": ["word1", "word2"],
        }
    }
}
}

```

- **version** - 表明了Response协议的版本，必须由 *CloudApp* 填充。当前协议版本是 `2.0.0`。
- **session** - 表示当前应用的session，与Request中的信息一致，*CloudApp* 可以在 *attributes* 里填充自己需要的上下文信息用于后面的请求。
- **response** - 返回给 *CloudAppClient* 的Response内容。包括了 `card` 和 `action` 两个部分。`card` 会在之后的协议更新中作详细说明。

3.2 Action定义

Action 目前包括两种类型：`voice` 和 `media`。`voice` 表示了语音交互的返回。`media` 是对媒体播放的返回。

```

"action": {
    "version": "2.0.0",
    "type": "NORMAL/EXIT",
    "form": "scene/cut/service",
    "shouldEndSession": true,
    "voice": {},
    "media": {},
    "display": {},
    "confirm": {}
}

```

字段	类型	可能值
version	string	action 协议的版本, 当前为 2.0.0
type	string	NORMAL / EXIT
form	string	scene / cut / service
shouldEndSession	boolean	true / false
voice	voice object	Voice对象
media	media object	Media对象
confirm	confirm object	Confirm对象

- **version** - 表明 action 协议版本, 当前版本为: 2.0.0.
- **type** - 当前action的类型: NORMAL 或 EXIT。当 type 是 NORMAL 时, voice 和 media 会同时执行; 当 type 是 EXIT 时, action会立即退出, 并且在这种情况下, voice 和 media 将会被忽略。
- **form** - 当前action的展现形式: scene、cut、service。scene的action会在被打断后压栈, cut的action会在被打断后直接结束, service会在后台执行, 但没有任何界面。该字段在技能创建时被确定, 无法由cloud app更改。
- **shouldEndSession** - 表明当此次返回的action执行完后 CloudAppClient 是否要退出, 同时, 当 shouldEndSession 为 true 时, CloudAppClient 将会忽略 EventRequests, 即在action执行过程中不会产生 EventRequest。
- **confirm** - 表明此次返回中, 是否存在需要confirm的内容。[了解用法指南](#)。
 - **confirmIntent**: 表明此次返回对哪一个intent进行confirm。
 - **confirmSlot**: 表明此次返回对哪一个slot进行confirm。
 - **optionWords**: 可选项。表明此次返回中在confirmSlot之上需要新增的confirm选项, 用于需要动态新增 confirm内容的场景。

3.2.1 Voice

Voice 定义了 CloudApp 返回的语音交互内容。具体定义如下:

```
"voice": {
  "action": "PLAY/PAUSE/RESUME/STOP",
  "item": {}
}
```

字段	类型	可能值
action	string	PLAY / PAUSE / RESUME / STOP
item	item object	voice 的 item 对象

- **action** - 表示对当前voice的操作, 可以播放 (PLAY)、暂停 (PAUSE) 、继续播放 (RESUME) 和停止 (STOP) (具体Action行为参照Media的Action行为, 但是目前暂未实现, PAUSE以及RESUME操作) ;
- **item** - 定义了voice的具体内容, 将会在 3.2.1.1 中详细描述。

3.2.1.1 Item

Item定义了voice的具体内容。

```
"item": {
  "tts": "tts content"
}
```

字段	类型	可能值

tts	string	tts 内容
-----	--------	--------

- **tts** - 定义了需要播报的TTS内容。

3.2.2 Media

Media 用来播放CloudApp返回的流媒体内容。有 *audio* 和 *video* 两种类型，目前第一版暂时只对 *audio* 作了支持，后续会支持 *video*。

```
"media": {
    "action": "PLAY/PAUSE/RESUME/STOP",
    "item": {}
}
```

字段	类型	可能值
action	string	PLAY / PAUSE / RESUME / STOP
item	media item object	media的具体内容

- **action** - 定义了对MediaPlayer的操作，目前只支持 4 种操作：**PLAY**, **PAUSE**, **RESUME** 和 **STOP**。其中，只有**PLAY**接受**item**数据。
 - **PLAY**: 如果有item数据，则按照最新的item从头开始播放，如果没有item**数据，且原来有在播放的内容，则从原来播放的内容开始播放
 - **PAUSE**: 暂停当前播放的内容，播放的进度等数据不会丢失（可以直接通过**RESUME**指令直接恢复原来的播放状态）
 - **RESUME**: 继续播放（从原来的播放进度播放）
 - **STOP**: 停止播放，并且清空当前的播放进度，但是播放内容不清
- **item** - 定义了具体的播放内容，如下：

3.2.2.1 Item

```
"item": {
    "token": "xxxx",
    "type": "AUDIO/VIDEO",
    "url": "media streaming url",
    "offsetInMilliseconds": 0
}
```

字段	类型	可能值
token	string	用于鉴权的token，由CloudApp填充和判断
type	string	AUDIO / VIDEO
url	string	可用的流媒体播放链接
offsetInMilliseconds	long	毫秒数值，表明从哪里开始播放

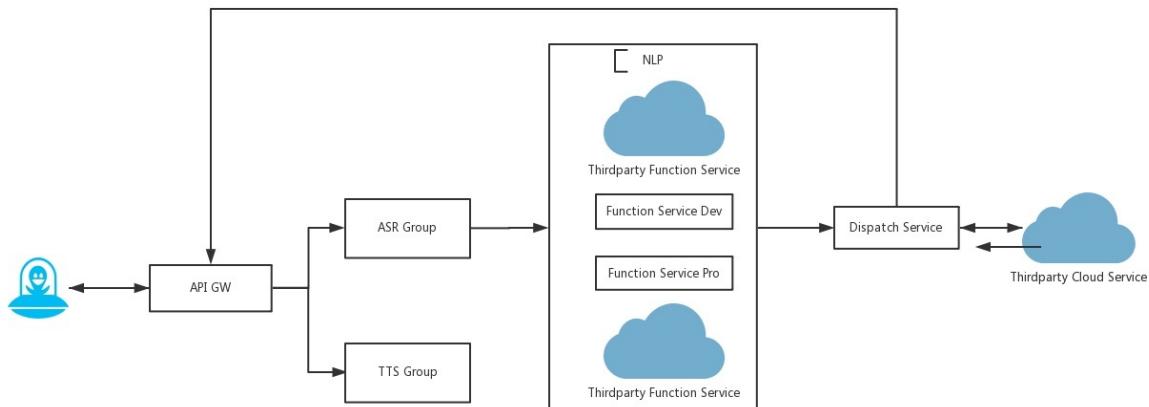
- **token** - 用于鉴权的token，由CloudApp填充和判断。
- **type** - 表明了当前媒体类型：**AUDIO** 或 **VIDEO**，有且只能取其一。
- **url** - 为MediaPlayer指明可用的流媒体播放链接。
- **offsetInMilliseconds** - 毫秒数值，告诉MediaPlayer开始播放的位置。有效范围从0到歌曲整体播放时长。

- 拦截器接口文档
 - NLP Interceptor

拦截器接口文档

语义理解（NLP）的拦截器（interceptor），可以允许开发者在进入我们NLP匹配之前或者是asr结果在若琪的NLP处理完成后结果为空时将请求进行拦截，拦截到开发者自己的https拦截器。

下图是整个语音的后端处理流程图：



图中的「Thirdparty Function Service」即为https拦截器

NLP Interceptor

- Request方式: POST
- Request参数:

参数名	类型	描述
version	string	接口版本号
account_id	string	account id,平台管理后台获取
device_type_id	string	设备类型ID
device_id	string	设备ID
sentence	string	待解析语句
language	string	语言
stack	string	当前app

- Response参数:

参数名	类型	描述	默认值	必填
app_id	string	skill id	无	是
intent	string	intent	无	否
slots	object	slots	无	否

- [rokid-openvoice](#)
 - [开放平台接口定义文档](#)
 - [文档简介](#)
 - [认证](#)
 - [Speech 接口](#)
 - [protobuf 定义](#)
 - [AuthRequest](#)
 - [AuthResponse](#)
 - [AsrRequest](#)
 - [AsrHeader](#)
 - [AsrResponse](#)
 - [TtsRequest](#)
 - [TtsHeader](#)
 - [TtsResponse](#)
 - [VoiceSpeechRequest](#)
 - [TextSpeechRequest](#)
 - [SpeechHeader](#)
 - [SpeechResponse](#)

rokid-openvoice

开放平台接口定义文档

文档版本:V0.2

Rokid openvoice开放服务包含以下四部分功能,

- 设备认证
- 语音识别 (ASR)
- 自然语言合成 (TTS)
- 语音交互 (SPCH)

文档简介

此文档用于定义开放平台上云端应用接口开发协议，协议遵循 [grpc](#) 协议。

认证

采用[grpc](#)的[调用证书](#)方案，约定如下：

参数	类型	描述	默认值
key	string	开放接口Key,在管理平台获取	无, 必填
device_type_id	string	设备类型ID	无, 必填
device_id	string	设备ID，只能由6~15位的字母和数字组成，不能含有空格和特殊符号。 由开发者自己生成管理，用于标志每个设备的唯一性。	无, 必填

service	string	asr,tts,spch	无, 必填
version	string	接口版本号	无, 必填
time	string	unix时间戳	无, 必填
sign	string	由以上几项+secret按约定的加密方式生成	无, 必填

sign的生成加密方式：

key={key}&device_type_id={device_type_id}&device_id={device_id}&service={service}&version={version}&time={time}&secret={secret}

的utf8字符串的md5值

其中{xxx}由xxx的值替代

key及secret由开发方通过管理平台获取，并保管。

Speech 接口

protobuf 定义

```

syntax = "proto3";

package rokid.open;

service Speech {
    rpc auth(AuthRequest) returns (AuthResponse) { }

    rpc asr(stream AsrRequest) returns (stream AsrResponse) { }

    rpc tts(TtsRequest) returns (stream TtsResponse) { }

    rpc speechv(stream VoiceSpeechRequest) returns (stream SpeechResponse) { }

    rpc speecht(TextSpeechRequest) returns (SpeechResponse) { }
}

message AuthRequest {
    string key                = 1;
    string device_type_id     = 2;
    string device_id          = 3;
    string service            = 4;
    string version             = 5;
    string timestamp           = 6;
    string sign                = 7;
}

message AuthResponse {
    // 0: success
    // -1: failed
    int32 result              = 1;
}

message AsrRequest {
    oneof request_content {
        AsrHeader header      = 1;
        bytes voice           = 2;
    }
}

```

```

}

message AsrHeader {
    int32 id          = 1;
    string lang        = 2;
    string codec       = 3;
    // vt = voice trigger
    string vt          = 4;
}

message AsrResponse {
    string asr         = 1;
}

message TtsRequest {
    TtsHeader header   = 1;
    string text        = 2;
}

message TtsHeader {
    int32 id          = 1;
    string disclaimer = 2;
    string codec       = 3;
}

message TtsResponse {
    string text        = 1;
    bytes voice        = 2;
}

message VoiceSpeechRequest {
    oneof request_content {
        SpeechHeader header = 1;

        bytes voice        = 2;
    }
}

message TextSpeechRequest {
    SpeechHeader header = 1;

    string asr          = 2;
}

message SpeechHeader {
    int32 id = 1;

    // zh
    // en
    string lang = 2;

    // pcm
    // opu
    // opu2
    string codec = 3;

    // vt = voice trigger
    string vt = 4;

    // stack of applications, "app1:app2:app3...."
    string stack = 5;

    // json format
    string device = 6;
}

message SpeechResponse {
    string asr          = 1;
}

```

```

    string nlp          = 2;
    string action        = 3;
}

```

AuthRequest

在同一条连接上第一个请求必须为Auth()。

参数	类型	描述	默认值
key	string	开放接口Key,在管理平台获取	无, 必填
device_type_id	string	设备类型ID	无, 必填
device_id	string	设备ID	无, 必填
service	string	asr,tts,spch	无, 必填
version	string	接口版本号	无, 必填
time	string	unix时间戳	无, 必填
sign	string	由以上几项+secret按约定的加密方式生成	无, 必填

sign的生成加密方式：

```
key={key}&device_type_id={device_type_id}&device_id={device_id}&service={service}&version={version}&time={time}&secret={secret}
```

的utf8字符串的md5值

其中{xxx}由xxx的值替代

key及secret由开发方通过管理平台获取，并保管。

AuthResponse

请求响应参数详见：[golang代码例子](#)

参数	类型	描述
result	int32	认证结果(成功:0, 失败:-1)

AsrRequest

一个请求一个AsrRequest{AsrHeader}后跟多个AsrRequest{voice}。PCM为 RIFF (little-endian) data, WAVE audio, Microsoft PCM, 16 bit, mono 16000 Hz语音。

请求响应参数详见：[golang代码例子](#)

参数	类型	描述	默认值
header	AsrHeader	帮助识别voice语音流的参数配置	无
voice	bytes	需要识别的voice语音流	无

AsrHeader

参数	类型	描述	默认值

id	int32	唯一标识，用于跟踪一个完整的请求，处理及响应事件。	0
lang	string	语音流的语言，目前支持zh-CN, en-US。	zh-CN
codec	string	语音流的编码，目前支持PCM, OPU, OPU2。	PCM
vt	string	激活词，即用于唤醒设备的名字，如"若琪"。	空

AsrResponse

参数	类型	描述
asr	string	asr实时识别的结果

TtsRequest

请求响应参数详见：[golang代码例子](#)

参数	类型	描述	默认值
header	TtsHeader	配置如何将text转换成voice语音流	无
text	string	需要转换的text文本	无

TtsHeader

PCM格式为 RIFF (little-endian) data, WAVE audio, Microsoft PCM, 16 bit, mono 24000 Hz语音。注意，这里的PCM与Asr的PCM格式不一致，因此不能将tts的输出直接作为asr的输入。

参数	类型	描述	默认值
id	int32	唯一标识，用于跟踪一个完整的请求，处理及响应事件。	0
claimer	string	发音者，如"zh", "zhangsan", "rose"	"zh"
codec	string	语音流的编码，目前支持PCM, OPU, OPU2。	PCM

TtsResponse

参数	类型	描述
text	string	voice语音中包含的文字
voice	bytes	合成的voice语音

VoiceSpeechRequest

请求响应参数详见：[golang代码例子](#)

参数	类型	描述	默认值
header	TtsHeader	配置如何将text转换成voice语音流	无
voice	bytes	需要识别的voice语音流	无

TextSpeechRequest

请求响应参数详见：[golang代码例子](#)

参数	类型	描述	默认值
header	TtsHeader	配置如何将text转换成voice语音流	无
asr	string	需要识别的asr文本	无

SpeechHeader

参数	类型	描述	默认值
id	int32	唯一标识，用于跟踪一个完整的请求，处理及响应事件。	0
lang	string	语音流的语言，目前支持zh-CN, en-US。	zh-CN
codec	string	语音流的编码，目前支持PCM, OPU, OPU2。	PCM
vt	string	激活词，即用于唤醒设备的名字，如"若琪"。	空
stack	string	设备当前的应用栈信息。	空
device	string	设备上的状态信息，为json结构。	空

stack结构: "applicationId1:applicationId2:application3"按照应用被调用的早晚逆序排列，当前应用在第一个。

SpeechResponse

参数	类型	描述
asr	string	asr实时识别的结果
nlp	string	nlp识别的结果
action	string	cloud app处理的结果

asr: 标准字符串，返回实时识别结果。

nlp: json字符串，结构如下

```
"content": {
  "applicationId": "com.rokid.cloud.music",
  "intent": "play_random",
  "slots": {}
}
```

action: json字符串，结构详见 [action定义](#)

Tts接口定义

调用接口

```

bool prepare(const char* config_file);

void release();

bool prepared();

int speak(const char* content, TtsCallback* cb);

void stop(int id);

// 'key': disclaimer      'value':
//       codec           'value': opu2 (default)
//                         opu
//                         pcm
void config(const char* key, const char* value);

```

回调接口

```

virtual void onStart(int id) = 0;

virtual void onText(int id, const char* text) = 0;

virtual void onVoice(int id, const unsigned char* data, int length) = 0;

virtual void onStop(int id) = 0;

virtual void onComplete(int id) = 0;

virtual void onError(int id, int err) = 0;

```

示例

```

class DemoCallback : public TtsCallback {
public:
    void onStart(int32_t id) {
        ...
    }

    void onData(int32_t id, const uint8_t* data, uint32_t length) {
        ...
    }

    ...
};

Tts tts;
DemoCallback cb;
// 配置服务器地址
tts.config("server_address", "apigw.open.rokid.com:443");
// 配置ssl cert文件路径(sdk源码中提供了roots.pem)
tts.config("ssl_roots_pem", "/system/etc/roots.pem");
// 配置语音服务认证信息
// key, secret, device_type_id需要申请
// device_id即设备序列号, 由用户自定, 用户可用它区分自己的设备
tts.config("key", "your_rokid_key");
tts.config("device_type_id", "your_device_type_id");

```

```
tts.config("secret", "your_secret");
tts.config("device_id", "your_device_id");
// api版本号
tts.config("api_version", "1");
if (!tts.prepare()) {
    log("tts prepare failed");
    return;
}
// 'cb' 回调会在另一个单独线程进行
int32_t id = tts.speak("你好", cb);
...
...
// quit tts
// 'release' 阻塞, 直至所有tts创建的线程退出
tts.release();
```

speech接口定义

```
struct SpeechResult {
    int32_t id;
    uint32_t err;
    std::string asr;
    std::string nlp;
    std::string action;
};

class Speech {
public:
    virtual bool prepare(const char* config_file = NULL) = 0;

    virtual void release() = 0;

    virtual int32_t put_text(const char* text) = 0;

    virtual int32_t start_voice() = 0;

    virtual void put_voice(int32_t id, const uint8_t* data, uint32_t length) = 0;

    virtual void end_voice(int32_t id) = 0;

    virtual void cancel(int32_t id) = 0;

    // poll speech results
    // block current thread if no result available
    // if Speech.release() invoked, poll() will return -1
    //
    // return value: 0 speech result
    //             1 stream result start
    //             2 stream result end
    //             3 speech cancelled
    //             4 speech occurs error, see SpeechResult.err
    //             -1 speech sdk released
    virtual int32_t poll(SpeechResult& res) = 0;

    virtual void config(const char* key, const char* value) = 0;
};

Speech* new_speech();

void delete_speech(Speech* speech);
```

示例

```
Speech* speech = rokid::speech::new_speech();
```

```

// 配置信息同tts示例，这里不再列出
// speech->config(...);
if (!speech->prepare()) {
    log("speech prepare failed");
    rokid::speech::delete_speech(speech);
    return;
}

... create callback thread and run ...

int32_t id1 = speech->put_text("你好");
int32_t id2 = speech->start_voice();
speech->put_voice(id2, voice_data, data_length);
speech->put_voice(id2, more_voice_data, data_length);
...
speech->end_voice();
...
speech->release();
callback_thread.join();
rokid::open::delete_speech(speech);

// callback thread
SpeechResult result;
int32_t r;
while (true) {
    r = speech->poll(result);
    // speech quit
    if (r < 0)
        break;
    ... handler speech result ...
}

```

asr接口定义

调用接口

```

bool prepare();

bool prepared();

void release();

int start(AsrCallback* cb);

void voice(int id, const unsigned char* data, int length);

void end(int id);

void cancel(int id);

void config(const char* key, const char* value);

```

回调接口

```

virtual void onStart(int id) = 0;

virtual void onData(int id, const char* text) = 0;

virtual void onStop(int id) = 0;

```

```
virtual void onComplete(int id) = 0;  
virtual void onError(int id, int err) = 0;
```

示例

与 tts 接口用法相似

nlp 接口定义

调用接口

```
bool prepare();  
bool prepared();  
void release();  
  
int request(const char* asr, NlpCallback* cb);  
  
void cancel(int32_t id);  
  
void config(const char* key, const char* value);
```

回调接口

```
virtual void onNlp(int id, const char* nlp) = 0;  
  
virtual void onStop(int id) = 0;  
  
virtual void onError(int id, int err) = 0;
```

示例

与 tts 接口用法相似

- Siren API v4
 - 简介
 - 配置文件字段
 - basic_config 基础配置
 - alg_config 算法配置
 - alg_def_vt 默认激活词配置
 - debug_config DEBUG选项
 - 数据结构与回调方法
 - 1. siren_input_if_t
 - 2. siren_proc_callback_t
 - Siren接口说明
 - 1. 初始化
 - 2. 打开语音处理音频流
 - 3. 打开裸数据音频流
 - 4. 关闭数据处理音频流
 - 5. 关闭裸数据音频流
 - 6. 关闭音频流
 - 7. 强制设置当前siren的激活/睡眠状态
 - 8. 强制设置当前寻向角度
 - 9. 关闭siren软件栈
 - 10. 添加唤醒激活词
 - 11. 删除激活词
 - 12. 查询所有激活词

Siren API v4

项目地址：<https://github.com/Rokid/rokit-blacksiren>

简介

Siren包括C的API和Java的API， API形式完全等价，下面详细分析API的时候会给出相关说明。Siren使用JSON形式的配置文件，配置文件可以通过文件绝对路径或者直接字符串两种方式进行加载。

配置文件字段

basic_config 基础配置

`mic_num` : 麦克风数目（包括AEC参考音源数目）
`mic_channel1_num` : 麦克风通道数目（包括参考音源通道），一般情况下和`mic_num`相同
`mic_sample_rate` : 麦克风音频采样率
`mic_audio_byte` : 麦克风音频位宽，一般取值为2（16位），4（32位）两种
`mic_frame_length` : 建议的每一语音帧的时长，建议语音帧长度为10ms。
`siren_ipc` : 使用channel的方式进行ipc还是share memory，目前仅支持channel
`siren_channel_rmem` : channel写缓存的大小
`siren_channel_wmem` : channel读缓存的大小
`siren_input_err_retry_num` : 输入音频流出错时重试的最大连续次数，已废弃 `siren_input_err_retry_timeout` : 两次重试间间隔的时间，单位时毫秒，已废弃

alg_config 算法配置

`alg_use_legacy_config_file` : 是否使用老siren的ssp配置文件方式, 已废弃
`alg_legacy_config_file_path` : 使用老siren的配置文件方式, 指出ssp配置文件位置, 已废弃
`alg_lan` : 当前语言配置, zh/en, 默认zh, 已废弃
`alg_rs_mics` : 降采样通道配置, 数组形式, 告知需要降采样的通道
`alg_aec` : 是否进行aec
`alg_aec_mics` : aec音频通道, 数组形式, 告知aec的音频数据通道
`alg_aec_ref_mics` : 参考音源通道, 数组形式, 用来告知AEC算法哪些通道是参考通道
`alg_aec_shield` : 默认200.0f
`alg_aec_aff_cpus` : aec处理线程的亲和性
`alg_aec_mat_aff_cpus` : aec矩阵运算的亲和性
`alg_raw_stream_sl_direction` : 裸数据流sl方向
`alg_raw_stream_bf` : 裸数据流是否需要bf处理
`alg_raw_stream_agc` : 裸数据是否需要agc处理
`alg_vt_enable` : 是否需要vt事件
`alg_vad_enable` : 是否需要vad事件, 此事前端处理流程退化成raw stream `alg_vad_mics` : vad使用的音频通道, 数组形式
`alg_mic_pos` : 所有麦克风的位置, 每个位置由x,y,z三个double坐标描述。
`alg_sl_mics` : 寻向使用的音频通道。
`alg_bf_mics` : 波束成形使用的音频通道。
`alg_opus_compress` : 是否输出opus编码后的语音
`alg_vt_phomod` : 存放音素对应表的绝对路径
`alg_dt_dnnmod` : 存放DNN模型的绝对路径 `alg_rs_delay_on_left_right_channel` : 左右声道是否存在不一致的delay, 常发生在i2s采集的情况下
`raw_stream_channel_num` : 裸音频流输出的通道数, 默认为1。
`raw_stream_sample_rate` : 裸音频流输出的采样率, 默认为16000。
`raw_stream_byte` : 裸音频输出的位宽, 默认为2。

alg_def_vt 默认激活词配置

`vt_type` : 激活词类型, 1是激活词, 2是睡眠词, 3是热词
`vt_word` : 激活词, utf-8的汉字
`vt_phone` : 激活词音素, 拼音, 假如激活词是“你好”, 则这里填ni2hao3, 声调轻声使用数字5
`vt_block_avg_score` : 所有phone声学平均得分门限
`vt_block_min_score` : 单个phone声学最低得分门限 `vt_left_check` : 是否进行前置静音检测
`vt_right_check` : 是否进行后置静音检测
`vt_remote_check_with_aec` : 是否在aec状态下进行远程asr确认, 如果为true, 则不会发出_nocmd类型事件
`vt_remote_check_without_aec` : 是否在非aec状态下进行远程asr确认
`vt_local_classify_check` : 是否进行本地激活二次确认
`vt_classify_shield` : 本地激活二次确认门限 `vt_nnet_path` : 激活词模型绝对路径

debug_config DEBUG选项

`debug_record_path` : 录音结果存放绝对路径
`debug_mic_array_record` : 是否针对输入进行录音
`debug_pre_result_record` : 是否针对预处理语音进行录音
`debug_proc_result_record` : 是否针对处理过的语音进行录音, 如果`alg_opus_compress`为true则录音得到的opus编码的数据, 否则是16K, 32float, 1ch的PCM流
`debug_rs_record` : 是否针对降采样的语音进行录音
`debug_aec_record` : 是否针对自身音源消除后的语音进行录音

```
debug_bf_record : 是否针对波束成形后的语音进行录音
debug_bf_raw_record : 废弃 debug_vad_record : 是否对vad的结果进行录音
debug_opu_record : 废弃
```

数据结构与回调方法

1. siren_input_if_t

功能

音频流输入回调函数集合

说明

该结构体由和音频输入相关得回调函数指针组成

1. 初始化音频流

函数功能

当调用init_siren后，并且配置成功后，回调该接口，该方法应该分配音频流相关资源

原型

```
int (*init_input_stream)(void *token)
```

参数

参数	类型	说明
token	void *	通过init_siren传入的token

返回值

操作成功返回0，否则init_siren会返回siren_status_error。

2. 结束音频流

函数功能

当调用destroy_siren时回调该方法，该方法应该释放音频流相关资源

原型

```
void (*release_input_stream)(void *token)
```

参数

参数	类型	说明
token	void *	通过init_siren传入的token

返回值

无

3. 打开音频流

函数功能

该方法应该打开音频流，该回调将在第一次调用start_siren_process_stream或start_siren_raw_stream时被调用

原型

```
int (*start_input_stream)(void *token)
```

参数

参数	类型	说明
token	void *	通过init_siren传入的token

返回值

返回0将告知blacksiren打开音频流成功，返回非0会告知blacksiren打开音频流失败，此时blacksiren的录音线程将不会启动

4. 停止音频流

函数功能

该方法应该暂停音频流，该回调将在最后一次调用stop_siren_process_stream或stop_siren_raw_stream时被调用

原型

```
void (*stop_input_stream)(void *token)
```

参数

参数	类型	说明
token	void *	通过init_siren传入的token

返回值

无

5. 从音频流中读取音频数据

函数功能

该方法应该根据配置正确返回音频流，音频流的大小和格式根据配置决定，缓冲区由siren负责分配和释放，该方法应该阻塞输入至少length字节的音频数据。

原型

```
int (*read_input_stream)(void *token, char *buffer, int length)
```

参数

参数	类型	说明
token	void *	通过init_siren传入的token
buffer	char *	需要填写音频流数据的缓冲区，不要超过length的长度
length	int	音频缓冲区的长度

返回值

返回值0表示从音频流成功读取了length字节的音频数据，其他值表示出错，出错后siren会调用 stop_input_stream最后重新尝试调用start_input_stream重新开始读取音频流数据。如果多次出错，则调用on_err_input_stream

6. 处理音频流错误

函数功能

当read_input_stream多次出错后，会回调该方法

原型

```
void (*on_err_input_stream)(void *token)
```

参数

参数	类型	说明
token	void *	通过init_siren传入的token

返回值

无

2. siren_proc_callback_t

功能

语音处理事件的回调接口

1. 语音事件回调接口

函数功能

当发生特定语音事件时，siren会回掉该接口

原型

```
void (on_voice_event*)(void *token, voice_event_t *voice_event)
```

参数

参数	类型	说明
token	void *	通过init_siren传入的token
voice_event	voice_event_t	语音信息

voice_event_t结构体成员如下

成员	类型	说明
event	siren_event_t	语音事件，详见下面语音事件说明
length	int	如果包含寻向信息指的是激活词中文的长度，如果是语音帧则是语音数据的长度
flag	int	可以使用HAS_VOICE, HAS_SL和HAS_VT来判断是否是语音帧，是否包含寻向信息，是否包含激活信息
sl	double	寻向角度

background_energy	double	归一化后的背景声能量，越大表明当前帧能量越大
background_threshold	double	背景声能量阈值，越大表明背景越吵
vt	vt_event_t	激活信息，具体见下面说明
buff	void *	如果HAS_VOICE，则是语音数据，长度由length表明，如果HAS_VT，则表示激活词名字字符串长度，如果HAS_SL，则为0

vt_event_t结构体成员如下

成员	类型	说明
start	int	在下面第一个SIREN_EVENT_VAD_DATA语音帧中激活词所在的开始，单位是float
end	int	语音帧中激活词结尾
energy	float	激活词的能量绝对值

返回值

无

语音事件说明

SIREN_EVENT_VAD_START : VAD_START事件，通常可以认为是一个语音帧的开始。 SIREN_EVENT_VAD_DATA : VAD_DATA事件，携带语音信息的语音帧。

SIREN_EVENT_VAD_END : VAD_END事件，语音帧的结束帧。

SIREN_EVENT_VAD_CANCLE : 因为误激活引发的VAD_CANCLE事件。

SIREN_EVENT_WAKE_VAD_START :

SIREN_EVENT_WAKE_VAD_DATA :

SIREN_EVENT_WAKE_VAD_END :

SIREN_EVENT_WAKE_PRE : 疑似激活词的开始，不能作为激活标准，已废弃。

SIREN_EVENT_WAKE_NOCMD : 单独激活事件。 SIREN_EVENT_WAKE_CMD : 以激活词开头，以其他语音结尾的混合激活事件，需要通过asr来进一步判断激活情况。

SIREN_EVENT_WAKE_CANCLE : 可以本地判断的误激活事件。

SIREN_EVENT_SLEEP : 睡眠激活词。

SIREN_EVENT_HOTWORD :

SIREN_EVENT_VOICE_PRINT : 声纹事件，包含声纹信息。

SIREN_EVENT_DIRTY :

Siren接口说明

1. 初始化

函数功能

初始化Siren软件栈

函数原型

```
siren_t init_siren(void *token, const char *path, siren_input_if_t *input)
```

参数

参数	类型	说明
token	void *	将在后续回调方法中被调用
path	const char *	JSON配置文件所在的本地文件绝对地址
input	siren_input_if_t *	siren语音输入接口

返回值

返回siren对象，如果失败则返回nullptr，siren对象用于后续操作的第一个参数

2. 打开语音处理音频流

函数功能

打开语音处理流，此时siren将源源不断的从siren_input_if_t提供的输入接口中读取音频数据，直到stop_siren_process_stream或stop_siren_stream被调用

函数原型

```
void start_siren_process_stream(siren_t siren, siren_proc_callback_t *proc_callback)
```

参数

参数	类型	说明
siren	siren_t	siren对象
proc_callback	siren_proc_callback_t *	处理结果回调接口

返回值

无

3. 打开裸数据音频流

函数功能

打开裸数据流，此时siren将源源不断的从siren_input_if_t提供的输入接口中读取音频数据，直到stop_siren_raw_stream或stop_siren_stream被调用

函数原型

```
void start_siren_raw_stream(siren_t siren)
```

参数

参数	类型	说明
siren	siren_t	siren对象

返回值

无

4. 关闭数据处理音频流

函数功能

关闭数据处理音频流，如果此时没有打开裸数据音频流，那么将调用siren_input_if_t的stop_stream方法

函数原型

```
void stop_siren_process_stream(siren_t siren)
```

参数

参数	类型	说明
siren	siren_t	siren对象

返回值

无

5. 关闭裸数据音频流

函数功能

关闭裸数据音频流，如果此时没有打开语音处理音频流，那么将调用siren_input_if_t的stop_stream方法

函数原型

```
void stop_siren_raw_stream(siren_t siren)
```

参数

参数	类型	说明
siren	siren_t	siren对象

返回值

无

6. 关闭音频流

函数功能

强制关闭音频流，将调用siren_input_if_t的stop_stream方法

函数原型

```
void stop_siren_stream(siren_t siren)
```

参数

参数	类型	说明
siren	siren_t	siren对象

返回值

无

7. 强制设置当前siren的激活/睡眠状态

函数功能

强制设置siren的激活/睡眠状态

函数原型

```
void set_siren_state(siren_t siren, siren_state_t state, state_changed_callback_t *callback)
```

参数

参数	类型	说明
siren	siren_t	siren对象
state	siren_state_t	可以在siren_state_awake和siren_state_sleep中选择
callback	state_changed_callback_t *	如果不是NULL则调用为异步，否则为同步，异步调用会在完成时调用callback接口

返回值

无

8. 强制设置当前寻向角度

函数功能

强制设置水平和垂直寻向角度

函数原型

```
void set_siren_steer(siren_t siren, float ho, float ver)
```

参数

参数	类型	说明
siren	siren_t	siren对象
ho	float	水平角度
ver	float	垂直角度

返回值

无

9. 关闭siren软件栈

函数功能

关闭siren软件栈，回收所有内存

函数原型

```
void destroy_siren(siren_t siren)
```

参数

参数	类型	说明
siren	siren_t	siren对象

返回值

无

10. 添加唤醒激活词

函数功能

添加一个激活词

函数原型

```
siren_vt_t add_vt_word(siren_t siren, siren_vt_word *word, bool use_default_config)
```

参数

参数	类型	说明
siren	siren_t	siren对象
word	siren_vt_word *	激活词信息
use_default_config	bool	是否自动使用默认配置

siren_vt_word 结构体说明

成员	类型	说明
vt_type	int	激活词类型，1是激活词，2是睡眠词，3是热词
vt_word	std::string	激活词，utf-8任意文字
vt_phone	std::string	激活词拼音，在add_vt_word成功调用后，这里会自动转化成音素
use_default_config	bool	是否对算法相关配置使用默认配置
alg_config	siren_vt_alg_config	激活词算法相关配置

siren_vt_alg_config 结构体说明

成员	类型	说明
vt_block_avg_score	float	见默认激活词配置中的说明， 默认值为4.2
vt_block_min_score	float	见默认激活词配置中的说明， 默认值为2.7
vt_left_sil_det	bool	见默认激活词配置中的说明， 默认值为true
vt_right_sil_det	bool	见默认激活词配置中的说明， 默认值为false
vt_remote_check_with_aec	bool	见默认激活词配置中的说明， 默认为false
vt_remote_check_without_aec	bool	见默认激活词配置中的说明， 默认值为false
vt_local_classify_check	bool	见默认激活词配置中的说明， 默认值为false
vt_classify_shield	float	见默认激活词配置中的说明， 默认值为-0.3
nnet_path	std::string	见默认激活词配置中的说明， 默认值为""

返回值

成功添加激活词，返回SIREN_VT_OK 激活词已经存在，返回SIREN_VT_DUP 添加激活词失败，返回SIREN_VT_ERROR

11. 删除激活词

函数功能

删除一个指定激活词，使用激活词名字来删除

函数原型

```
siren_vt_t remove_vt_word(siren_t siren, const char *word)
```

参数

参数	类型	说明
siren	siren_t	siren对象
word	const char *	激活词名字

返回值

成功删除一个激活词，返回SIREN_VT_OK 删除激活词失败，返回SIREN_VT_ERROR 激活词不存在，返回SIREN_VT_NO_EXIT

12. 查询所有激活词

函数功能

查询所有激活词

函数原型

```
int get_vt_word(siren_t siren, siren_vt_word **words)
```

参数

参数	类型	说明
siren	siren_t	siren对象
words	siren_vt_word *	所有激活词数组，数目由返回值给出

返回值

-1表示错误，其他表示激活词数目

- Rokid语音产品硬件设计指南
 - Rokid开放平台技术优势
 - 典型产品示例
 - 工业设计和结构设计指南
 - 工业设计指南
 - 麦克风阵列排布
 - 结构设计原则
 - 硬件芯片方案
 - 支持业界主流应用处理器AP
 - 第三方降噪方案的集成
 - 麦克风选型参考
 - 麦克风数据格式

Rokid语音产品硬件设计指南

Rokid将基于自身打磨多年的历代产品经验，进行总结和归纳，并免费开放给用户。用户可以立即将Rokid优秀的语音整套解决方案，轻松地集成到具有麦克风和喇叭的联网设备上，打造业界优秀的语音交互产品(VoiceUI or VUI products)。

Rokid开放平台技术优势

- Rokid拥有完整自主知识产权的全功能语音链条；
- 远场语音识别的优化：Rokid积累了数年的远场语音真实数据，并用于自有的ASR引擎训练优化，针对智能家居、远距离控制等应用场景具备更好的识别率；
- 软硬件全流程的技术方案：从麦克风选型、阵列设计，到前端语音处理算法、云端建模，Rokid能够做到硬件和软件的全栈式优化的技术方案；
- 顶尖的本地和云端建模核心技术：Rokid的语音唤醒、识别、合成以及语义理解，均采用了业界最新最先进的算法建模技术，包括CTC、DeepCNN、LSTM等End To End的语音语义建模算法，结合真实的用户数据，能够达到业界最好的识别和理解水平；
- 高度灵活和可定制的云端语义理解技术：我们的NLU开放平台可由用户来完成全部订制，并可为用户提供专门的理解模型的训练；
- TTS定制和变声技术：为用户提供TTS模型订制以及变声订制（其他平台很少提供这样的服务）；

典型产品示例

开启若琪VUI的方式有两种：触摸以及语音激活；触摸可以通过遥控器或者手机APP，语音则是叫一声“若琪”，类似若琪和若琪•月石的激活词。

			
应用场景	触摸(<0.5m)	2-3米激活词	3-5米激活词

智能音箱		•	•
遥控器	•		
手机APP	•		
OTT/机顶盒	•	•	•
智能家居		•	•
车载		•	•

工业设计和结构设计指南

工业设计指南

为保证语音体验的完好，建议水平设置；如果产品设计所需，麦克风的倾斜角度必须<30度。

麦克风阵列排布

① 均匀圆形阵列

数量	设计半径R
4	18mm<R<=30mm
6	25mm<R<=43mm
8	30mm<R<=56mm
16	60mm<R<=110mm

② 均匀线性阵列

数量	设计间距L
2-8	30mm<L<=60mm(43mm最佳)

结构设计原则

麦克风导音孔：不影响频响效果；腔体设计、避震等case by case；

硬件芯片方案

支持业界主流应用处理器AP

Rokid开放平台的远距离拾音技术可轻松地移植到支持Android与Linux平台的AP：

- Amlogic
- Samsung
- Qualcomm
- Rockchip
- Allwinner
- Others

第三方降噪方案的集成

Rokid开放平台可以将业界优秀的语音激活与降噪芯片厂商，集成到平台中，并与云端的语音识别技术配合，共同提供整体解决方案：

- XMOS
- Others

麦克风选型参考

语音方案支持业界主流麦克风：

- I2S
- PDM
- Analog

建议用户在保证成本的同时，从信噪比SNR、灵敏度Sensitivity、相位一致性以及SPL曲线特征等参数进行选择。

麦克风数据格式

① 如果直接对接云端不通过我们的前端降噪模块，所需数据：

- 采样频率：16K
- 16bit
- PCM/OPUS编码的音频文件

② 如果使用前端降噪，则需要按照mic阵列的参数完成一个配置文件，前端降噪支持：

- 通道数量：2,4,6,8路
- 采样频率：48K
- 32bit的PCM语音流

- Rokid 硬件交互设计指南
 - 系统整体
 - 灯光
 - A. 灯光的颜色, 含义和用途
 - B. 灯效
 - C. 光停留长短和闪烁频率
 - D. 光的强弱 (亮度值和范围)
 - 音效
 - E. 组成
 - F. 作用

Rokid 硬件交互设计指南

系统整体

对于Rokid产品整个系统流程方面的主要交互上的布局和细节内容会考虑如下内容。

对于交互的整个流程中，我们应遵循如下几点：

- 如何准确的引导使用者进入某项功能中, 这个功能可能是基础配网或者某个应用。
- 其次，我们应该给出足够的提示和引导帮助使用者了解如何才能使用这个产品的功能。
- 同时我们应该定义使用者在什么环境下才能使用此功能
- 最后我们要定义符合什么条件的用户才能使用此功能。

整体上，Rokid的各种产品应达到交互上的统一。即是不管有没有UI界面显示他们给人交互上的反馈应该是一致的。总体上的交互层级应该是由TTS>灯光>音效>界面。但是音效和TTS相比于UI和灯光更加具有干扰性，应该尽量避免几种途径的重复叠加使用。但是如果事情是很重要紧急的，则应该采用2种及以上的途径提示用户去解决。

TTS:

- 交互的重点应该放在TTS的准确性上而其他感官的交互应作为辅助TTS的功能。
- TTS应该作为解释重要信息和明确指令的传达但是过多使用TTS和TTS过长会让用户有觉得啰嗦厌烦心里。

灯光:

- 灯光的交互应该做到能第一时间传达感官上快, 出问题了和没有问题等的一些可以直观理解的信息。利用不同颜色的灯光可以传达不同的指令例如绿光是没问题, 通过。红光是有问题。除了颜色之外光还可以有强弱, 不同光交替的效果和光闪耀的频率等用于传达速度和系统状态等的问题。但是大面积的光容易打扰到用户应尽量避免长时间和高强度大面积的光。

界面:

- 可以演变成多种形式增加娱乐性, 但是不应该让用户过多依赖UI的显示从而削弱了TTS的功能。

音效:

- 音效可以分为韵律性的和音标。带韵律性的音效有助于传达给用户美好的体验和丰富他们的感官享受而音标则是像图标一样是具有象征性意义的某种系统音效。他可以拟物也可以作为重复性或者具有某种特殊意义的语音上的符号。

灯光

灯光应该作为Rokid设备除TTS之外主要的交互途径。

灯光的交互应当快速的传达直观信息便于用户对产品状态的迅速理解和判断。需要注意的是大面积的光容易打扰到用户应尽量避免长时间和高强度大面积的光。

灯光的呈现包括：

- 颜色 (含义和用途)
- 不同灯效
- 光停留的长短和闪烁频率
- 光的强弱 (亮度值和范围)

A. 灯光的颜色，含义和用途

- 灯光可以利用单一或者多种颜色叠加的不同呈现。不同颜色的灯光可以传达不同的指令例如绿光是没问题，通过。红光是有问题。
- Pebble采用的颜色主要有如下几种：
 - 蓝色：主要用于系统未准备好前及系统配置的灯光显示
 - 开机时
 - 开机加载时
 - 升级时
 - 白色：主要用于系统准备好后的功能正常运转的显示
 - 寻向灯
 - 系统确认成功
 - 系统非异常反馈
 - 按键正常反馈
 - 指令加载
 - 开机后
 - 音量
 - 橙色：主要用于非紧急情况的提示反馈
 - 系统加载超时
 - 等待配网/连接不稳定
 - 低电量提示 (<50%)
 - 红色：主要用于紧急灯光的提示
 - 低电量提示 (<20%)马上就要被动关机
 - 网络/蓝牙连接失败中断
 - 禁止拾音
 - 绿色：主要用于电量相关显示
 - 充电电量
 - 混色：主要用于屏保显示
 - 呼吸光
 - 屏保光

B. 灯效

- 寻向灯
 - 从两边向声源处汇聚
- 灯圈全亮
 - 开机时，给人一种很快迅速的感觉

- 确认灯效, 放在成功完成某项指令时
 - 提示时
- 呼吸灯
 - TTS波形呼吸, 作为TTS时灯效
- 刻度灯效 (双边)
 - 表现系统进度可知如系统升级, 开机加载时
 - 音量+/-
 - 充电量多少
- 跑马灯 (快速/慢速)
 - 慢速为间隔灯珠跑马灯而快速为单颗灯珠跑马灯有最亮100%和最暗的灯珠渐变
 - 系统未知时间/进程加载 (可先慢速再快速)
 - 系统已知时间/进程加载 (快速)
 - 自动检测 (慢速)
 - 系统等待输入 (慢速)
- 屁股灯
 - 可用于显示电量多少和是否充电的状态。
 - 关机充电时
- 扫尾灯
 - 从一点向2边延伸和收回灯光。
 - 通电一刹那

C. 光停留长短和闪烁频率

- 光停留长短应根据其状态是否改变。例如拾音时指向灯只会在用户说话时候显示。TTS时也会根据系统回馈的长短同时配合灯光。
 - 所有灯效如有出现或消失需控制在300ms以内
 - 状态成功确认为0.5s闪烁一次, 状态提示0.5s闪烁2次, 状态警示为1s闪烁1次
 - 按键灯效的显示停留0.5s
 - 灯圈长亮时状态有开机一开始, 静音键按下的状态时
 - 屁股灯闪烁0.5s一次循环
 - 普通跑马灯为连续, 600ms一圈
 - 有进度条的灯在非用户主动行为造成结果显示下为长亮
 - 刻度灯效停留3s
 - 屏保灯为2s一换
 - 呼吸为2s一循环
 - 渐隐渐退的衔接为30ms以内

D. 光的强弱 (亮度值和范围)

- Pebble下每颗光珠如有显示, 光的default亮度值为100%。在一些灯效下可以允许亮度值有所调整:
 - 跑马灯
 - 呼吸灯
 - 关机灯

音效

音效和TTS的区别是它可以通过较少的语音提示去传达信息同时又不像TTS一样繁琐。但是它的弊端也在于信息的准确度和充足性不够，所以不适合单独去呈现新的复杂的信息。所以音效有时需要和其他反馈的途径去配合给用户呈现。

E. 组成

- 音调
 - 音调的高低会产生层级上的变化，比如菜单当中位于最高层的或者上面的音调会高。位于菜单最底层的或者下面的音调会低。
- 音质
 - 优美的音质会让人愉悦。嘈杂的音质会让人反感。
 - 采用不同的乐器音质也会给用户不同的感觉。例如：电子乐给人科技感更强而古典乐器偏木制的则会有质感，感觉更情调一些。
- 音量
 - 音量的从低到高也可以帮助提示用户。
- 时长
 - 时长可以帮助用户提示进程。

F. 作用

- 提醒提示（重要/一般）
 - 按照事情的紧急程度和重要性使用不同的音效。主要是要看是否有干扰到用户正常的操作。是应该停下来手边的事去关注还是可以了解一下但不需要马上解决。
 - 对于紧急重要的，应该加强音量和音调。如果是想显示一个进程的过程而不是状态，可以考虑延长时长。
- 重要信息提示：是用户要马上采取处理才能不影响正常操作的错误和失败的音效。
 - 联网中断
 - 蓝牙配对失败
 - 低电量 (<20%)
 - 系统异常
 - 开机进程加载失败
 - 指令识别失败（例如系统无法执行TTS反馈，无法执行智能家居操作）
- 一般信息反馈：用音效显示进程或者结果（好的/坏的）他的角色就是辅助灯光或者TTS
 - 音量调节
 - 静音键
 - 手机app恢复出厂设置
 - 进程反馈/成功确认（例如蓝牙配对成功，开机系统加载好）
 - 开机系统不稳定
 - 指令执行（例如音乐应用收藏，取消收藏）
- 旋律
 - 用户知道
 - 闹钟旋律
 - 秒表旋律
 - 开关机（例如开机系统加载完成的从低到高的）

	状态细节	灯光	音效
开机	开始启动	颜色：蓝色 停留时长：2-3s 灯光细节：灯长亮	N/A

	启动中-启动好	颜色: 蓝色 停留时长: 直到状态转换 灯光细节: 单颗拖尾带刻度加载	Y
关机	N/A	颜色: 当前光 - 深蓝色30% 停留时长: 直到状态转换 灯光细节: 深蓝色灯光闪烁3次保持当前颜色, 开始做渐暗效果直到关机完成	Y
休眠	N/A	N/A	N/A
屏保	N/A	颜色: 蓝紫色系30% 停留时长: 直到状态转换 灯光细节: 进入屏保时, 先在0.5秒内从30%过渡到5%亮度的基础底灯, 保持2秒后开始渐变多彩色, 位置随机, 一个颜色过渡时间为3秒, 然后花3秒钟从30%暗到5%, 再过度到下一个颜色	Y
激活	考虑是否可以设置语音或者按键激	颜色: 底色蓝30%+寻向灯白色100% 停留时长: 直到状态转换 灯光细节: 首次拾音从尾部扫到拾音区; 在拾音状态没消失前, 拾音方向 转换会直接在新方向直接出现不需要扫尾。	Y (连续拾音是没有音效的)
加载	系统未启动好: e.g., 系统开机加载	颜色: 蓝色 停留时长: 直到状态转换 灯光细节: 单颗拖尾带刻度加载	N/A
	系统启动好: e.g., NLP加载	颜色: 白色100% 停留时长: 直到状态转换 灯光细节: 单颗跑马灯600ms一圈	N/A
应用	TTS	颜色: 白色20%~60% 停留时长: 直到状态转换 灯光细节: 波形闪动用灯光模拟声音的频率明暗闪动	N/A
	执行中	N/A	N/A
电量	关机电量显示: 满电	颜色: 绿色100% 停留时长: 直到状态转换 灯光细节: 屁股灯全亮持续	N/A
	关机电量显示: >20%	颜色: 橙色100% 停留时长: 直到状态转换 灯光细节: 屁股灯全亮持续	N/A
	关机电量显示: <20%	颜色: 红色100% 停留时长: 直到状态转换 灯光细节: 屁股灯全亮持续	N/A
电源	拔下电源	N/A	Y
	插上电源	N/A	Y
音量	加减音量	颜色: 底光深蓝色30%+白色100% 停留时长: 直到状态转换 灯光细节: 两边从尾部灯加一格	Y
配网	连接中	颜色: 橘色100% 停留时长: 直到状态转换 灯光细节: 4个一组跑马灯600ms呼吸一次	N/A
升级	升级中	颜色: 整圈底光深蓝色30%+刻度蓝色100%按比例 加载 停留时长: 直到状态转换 灯光细节: 从前面两端前后扫尾加载直到尾部	N/A
麦克		颜色: 红色100%	

风开启		停留时长：直到状态转换 灯光细节：灯全亮保持直到状态取消	Y
麦克风关闭	还原最前面一个状态	N/A	Y
提示	失败 低电量；连接超时 系统异常	颜色：红色100% 停留时长：直到状态转换 灯光细节：灯全亮0.5s闪烁2次	Y
	成功	颜色：白色100% 停留时长：直到状态转换 交互体验：N/A 灯光细节：灯全亮0.5s闪烁一次	Y

- Rokid技能发布标准

Rokid技能发布标准

该标准适用于所有提交至Rokid开发者社区审核的公有技能（私有技能的要求不在本审核标准范围内）。本标准的内容会随着开发者社区的建设而不断完善。

如果您的技能包含或者涉及以下被禁止的内容，Rokid开发者社区将拒绝您技能的上线申请，并通过与开发者帐户相关联的电子邮件地址通知您。

商标，知识产权和品牌

- 未经Rokid授权或同意，擅自编造与rokid之间的合作关系。
- 侵犯第三方的知识产权（包括版权，商标和宣传权）。
- 擅自使用第三方公司的品牌或擅自编造与第三方公司之间的合作关系。

健康相关

- 收集用户的身体健康或精神状况有关的信息，并企图通过此方法获利。
- 发布健康有关的技能，但在技能描述中并没有包括免责声明，声明该技能不能替代专业健康建议。示例：“此技能不提供专业医疗建议，不能替代专业医疗咨询或诊断。专业医疗问题请前往医院接受医疗咨询。如果遇到紧急健康状况，请拨打120。”
- 通过技能名称或技能描述暗示可以提供专业救援。

收费相关

- 提供一个非Rokid授权的第三方收费技能商店。
- 通过利益手段诱导用户使用其他技能。
- 暗示或诱导用户向银行账号或者第三方支付账户捐款。

广告

- 包含任何第三方产品或服务的广告，且广告不是该技能的核心功能。

色情、暴力

- 具有色情内容的描述。
- 包含极端血腥、恐怖，令人不安的内容或过度暴力的描述。
- 暗示有组织犯罪，恐怖主义或其他非法活动，意图破坏地方或国家政府安定。

宗教，种族和文化

- 提及强迫婚姻或可购买伴侣的信息。
- 能够预测未来等迷信信息。
- 针对任何群体或个人的贬损评论或仇恨言论。
- 包含非法或政治相关内容，这些引用会误导对原文意图的理解。
- 包含促进仇恨言论，煽动种族仇恨或性别仇恨，或支持这种信仰的团体或组织。

内容

- 包含非法下载盗版软件的途径。

- 包含如何加入非法组织的具体建议。
- 包含诱导违法行为（如卖淫）的信息。
- 包含如何制造或建造危险物资（如如何建造炸弹，消声器，甲型实验室等）信息。
- 包含关于药物或处方药的虚假声明，例如声称治愈所有疾病或提供非法销售处方药的信息。
- 包含诱导用户参与赌博的信息。
- 包含诱导未成年人吸烟、酗酒信息。
- 包含对他人的侮辱、恶意诽谤、过度的恶作剧信息。

- Rokid 开发者社区服务协议

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/或《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，且不因该等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。

5.12 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，存在违法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求，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。未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

《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。

