ALIBABA CLOUD

# 阿里云

## 智能语音交互
## 实时语音识别

文档版本：20220425

阿里云

# 法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

# 通用约定

| 格式 | 说明 | 样例 |
| --- | --- | --- |
| 危险 | 该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。 | 危险<br>重置操作将丢失用户配置数据。 |
| 警告 | 该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。 | 警告<br>重启操作将导致业务中断，恢复业务时间约十分钟。 |
| 注意 | 用于警示信息、补充说明等，是用户必须了解的内容。 | 注意<br>权重设置为0，该服务器不会再接受新请求。 |
| 说明 | 用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。 | 说明<br>您也可以通过按Ctrl+A选中全部文件。 |
| > | 多级菜单递进。 | 单击**设置**> **网络**> **设置网络类型**。 |
| **粗体** | 表示按键、菜单、页面名称等UI元素。 | 在**结果确认**页面，单击**确定**。 |
| `Courier字体` | 命令或代码。 | 执行 `cd /d C:/window` 命令，进入Windows系统文件夹。 |
| *斜体* | 表示参数、变量。 | `bae log list --instanceid` *Instance_ID* |
| [] 或者 [a\|b] | 表示可选项，至多选择一个。 | `ipconfig [-all\|-t]` |
| {} 或者 {a\|b} | 表示必选项，至多选择一个。 | `switch {active\|stand}` |

# 目录

1.接口说明 05
2.移动端SDK 17
2.1. 接口说明 17
2.2. Android SDK 35
2.3. iOS SDK 40
2.4. 错误码 47
3.Java SDK 56
4.C++ SDK 62
5.C# SDK 80
6.Go SDK 97
7.Python SDK 110
8.Node.js SDK 117
9.微信小程序 123
10.WebSocket协议说明 131

# 1.接口说明

对长时间的语音数据流进行识别，适用于会议演讲、视频直播等长时间不间断识别的场景。

## 使用须知

> **注意**
>
> 如需使用Android或iOS SDK，请参见移动端接口说明。

- 支持的输入格式：PCM（无压缩的PCM或WAV文件）、16 bit采样位数、单声道（mono）。
- 支持的音频采样率：8000 Hz、16000 Hz。
- 支持设置返回结果：是否返回中间识别结果，在后处理中添加标点，将中文数字转为阿拉伯数字输出。
- 支持设置多语言识别：在控制台编辑项目中进行模型选择，详情请参见管理项目。

## 就近地域智能接入

实时语音识别支持就近地域智能接入，域名为 `nls-gateway.aliyuncs.com` 。

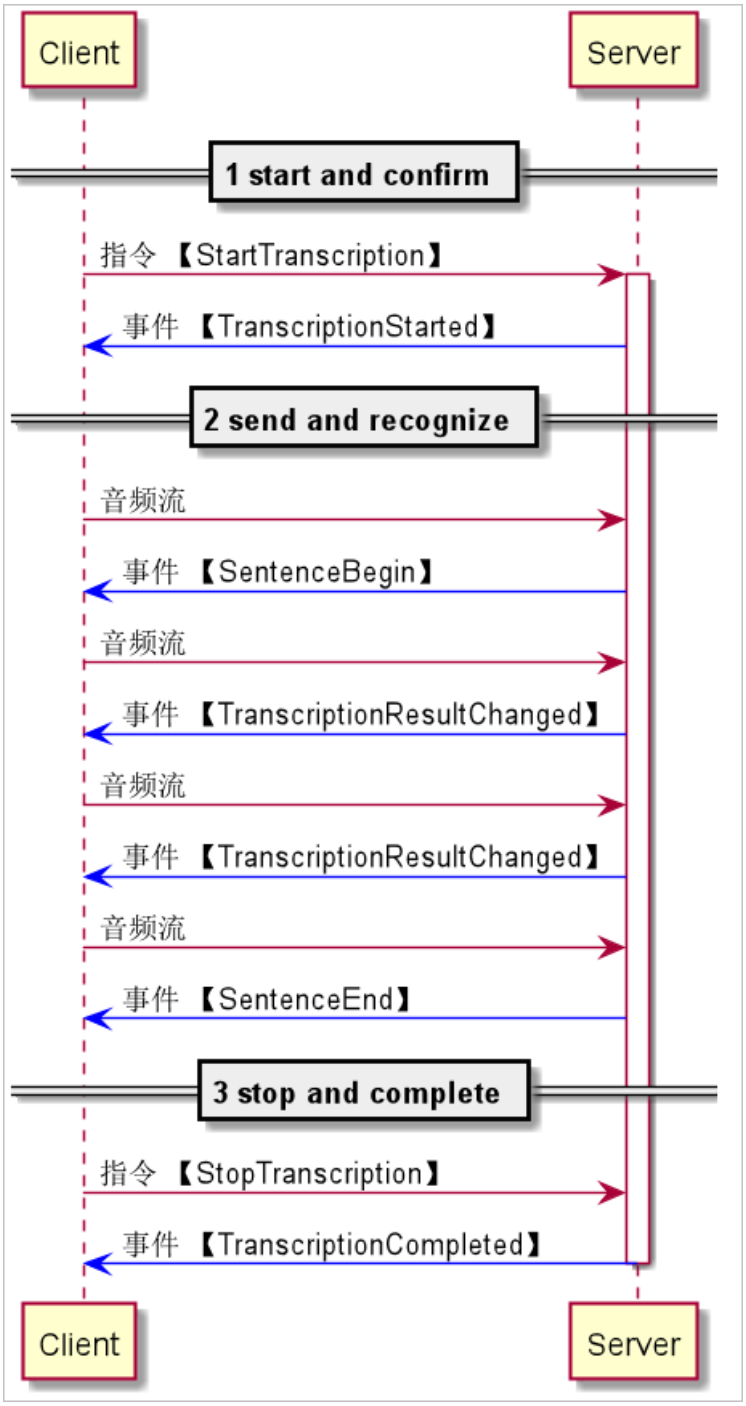
推荐终端用户使用就近地域接入域名。根据调用接口时客户端所在的地理位置，系统会自动解析到最近的某个具体地域的服务器。例如在北京地域发起请求，系统会自动解析到北京地域的服务器，与指定域名 `nls-gateway-cn-beijing.aliyuncs.com` 的实现效果一致。

## 服务地址

| 访问类型 | 说明 | URL |
| --- | --- | --- |
| 外网访问（默认上海地域） | 所有服务器均可使用外网访问URL（SDK中默认设置了外网访问URL）。 | • 上海：`wss://nls-gateway-cn-shanghai.aliyuncs.com/ws/v1`<br>• 北京：`wss://nls-gateway-cn-beijing.aliyuncs.com/ws/v1`<br>• 深圳：`wss://nls-gateway-cn-shenzhen.aliyuncs.com/ws/v1` |

| 访问类型 | 说明 | URL |
| --- | --- | --- |
| ECS内网访问 | 使用阿里云上海、北京、深圳ECS（即ECS地域为华东2（上海）、华北2（北京）、华南1（深圳）），可使用内网访问URL。 ECS的经典网络不能访问AnyTunnel，即不能在内网访问语音服务；如果希望使用AnyTunnel，需要创建专有网络在其内部访问。<br>说明<br>• 使用内网访问方式，将不产生ECS实例的公网流量费用。<br>• 关于ECS的网络类型请参见网络类型。 | • 上海：`ws://nls-gateway-cn-shanghai-internal.aliyuncs.com:80/ws/v1`<br>• 北京：`ws://nls-gateway-cn-beijing-internal.aliyuncs.com:80/ws/v1`<br>• 深圳：`ws://nls-gateway-cn-shenzhen-internal.aliyuncs.com:80/ws/v1` |

## 交互流程




> **说明**
>
> 所有服务端的响应都会在返回信息的header包含用于表示本次识别任务的task_id参数，请记录下该值，如果发生错误，请将task_id和错误信息提交到工单。

## 1. 鉴权

客户端与服务端建立WebSocket连接时，使用Token进行鉴权。关于Token获取请参见获取Token。

## 2. 开始识别

客户端发起请求，服务端确认请求有效。其中在请求消息中需要进行参数设置，各参数由SDK中SpeechTranscriber对象的set方法设置，各参数含义如下：

| 参数 | 类型 | 是否必选 | 说明 |
| --- | --- | --- | --- |
| appkey | String | 是 | 管控台创建的项目Appkey。 |
| format | String | 否 | 音频编码格式，默认是PCM（无压缩的PCM文件或WAV文件），16bit采样位数的单声道。 |
| sample_rate | Integer | 否 | 音频采样率，默认是16000 Hz，根据音频采样率在管控台对应项目中配置支持该采样率及场景的模型。 |
| enable_intermediate_result | Boolean | 否 | 是否返回中间识别结果，默认是False。 |
| enable_punctuation_prediction | Boolean | 否 | 是否在后处理中添加标点，默认是False。 |
| enable_inverse_text_normalization | Boolean | 否 | 是否在后处理中执行ITN，设置为True时，中文数字将转为阿拉伯数字输出，默认是False。<br>**说明**<br>不对词信息进行ITN转换。 |
| customization_id | String | 否 | 自学习模型ID。 |
| vocabulary_id | String | 否 | 定制泛热词ID。 |

| 参数 | 类型 | 是否必选 | 说明 |
| --- | --- | --- | --- |
| max_sentence_silence | Integer | 否 | 语音断句检测阈值，静音时长超过该阈值会被认为断句，参数范围200ms～2000ms，默认值800ms。 |
| enable_words | Boolean | 否 | 是否开启返回词信息，默认是False。 |
| enable_ignore_sentence_timeout | Boolean | 否 | 是否忽略实时识别中的单句识别超时，默认是False。 |
| disfluency | Boolean | 否 | 是否对识别文本进行顺滑（去除语气词，重复说等），默认是False。 |
| speech_noise_threshold | Float | 否 | 噪音参数阈值，参数范围：[-1,1]。取值说明如下：<br>• 取值越趋于-1，噪音被判定为语音的概率越大。<br>• 取值越趋于+1，语音被判定为噪音的概率越大。<br>说明<br>该参数属高级参数，调整需慎重并重点测试。 |
| enable_semantic_sentence_detection | Boolean | 否 | 是否开启语义断句，可选，默认是False。 |

## 3. 接收识别结果

客户端循环发送语音数据，持续接收识别结果：

- SentenceBegin事件表示服务端检测到了一句话的开始。实时语音识别服务的智能断句功能会判断出一句话的开始与结束，举例如下：

```
{
        "header": {
                "namespace": "SpeechTranscriber",
                "name": "SentenceBegin",
                "status": 20000000,
                "message_id": "a426f3d4618447519c9d85d1a0d1****",
                "task_id": "5ec521b5aa104e3abccf3d361822****",
                "status_text": "Gateway:SUCCESS:Success."
        },
        "payload": {
                "index": 1,
                "time": 0
        }
}
```

header对象参数说明：

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| namespace | String | 消息所属的命名空间。 |
| name | String | 消息名称，SentenceBegin表示一个句子的开始。 |
| status | Integer | 状态码，表示请求是否成功，见服务状态码。 |
| status_text | String | 状态消息。 |
| task_id | String | 任务全局唯一ID，请记录该值，便于排查问题。 |
| message_id | String | 本次消息的ID。 |

payload对象参数说明：

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| index | Integer | 句子编号，从1开始递增。 |
| time | Integer | 当前已处理的音频时长，单位是毫秒。 |

- TranscriptionResultChanged事件表示识别结果发生了变化。仅当enable_intermediate_result取值为true时会多次返回此消息，即一句话的中间识别结果，举例如下：

```
{
        "header": {
                "namespace": "SpeechTranscriber",
                "name": "TranscriptionResultChanged",
                "status": 20000000,
                "message_id": "dc21193fada84380a3b6137875ab****",
                "task_id": "5ec521b5aa104e3abccf3d361822****",
                "status_text": "Gateway:SUCCESS:Success."
        },
        "payload": {
                "index": 1,
                "time": 1835,
                "result": "北京的天",
                "confidence": 1.0,
                "words": [{
                        "text": "北京",
                        "startTime": 630,
                        "endTime": 930
                }, {
                        "text": "的",
                        "startTime": 930,
                        "endTime": 1110
                }, {
                        "text": "天",
                        "startTime": 1110,
                        "endTime": 1140
                }]
        }
}
```

header对象参数同上述表格说明，name为TranscriptionResultChanged：表示句子的中间识别结果。

payload对象参数说明：

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| index | Integer | 句子编号，从1开始递增。 |
| time | Integer | 当前已处理的音频时长，单位是毫秒。 |
| result | String | 当前句子的识别结果。 |
| words | List< Word > | 当前句子的词信息，需要将enable_words设置为true。 |

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| confidence | Double | 当前句子识别结果的置信度，取值范围：[0.0,1.0]。值越大表示置信度越高。 |

- SentenceEnd事件表示服务端检测到了一句话的结束，并附带返回该句话的识别结果，举例如下：

```
{
        "header": {
                "namespace": "SpeechTranscriber",
                "name": "SentenceEnd",
                "status": 20000000,
                "message_id": "c3a9ae4b231649d5ae05d4af36fd****",
                "task_id": "5ec521b5aa104e3abccf3d361822****",
                "status_text": "Gateway:SUCCESS:Success."
        },
        "payload": {
                "index": 1,
                "time": 1820,
                "begin_time": 0,
                "result": "北京的天气。",
                "confidence": 1.0,
                "words": [{
                        "text": "北京",
                        "startTime": 630,
                        "endTime": 930
                }, {
                        "text": "的",
                        "startTime": 930,
                        "endTime": 1110
                }, {
                        "text": "天气",
                        "startTime": 1110,
                        "endTime": 1380
                }]
        }
}
```

header对象参数同上述表格说明，name为SentenceEnd表示识别到句子的结束。

payload对象参数说明：

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| index | Integer | 句子编号，从1开始递增。 |
| time | Integer | 当前已处理的音频时长，单位是毫秒。 |

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| begin_time | Integer | 当前句子对应的SentenceBegin事件的时间，单位是毫秒。 |
| result | String | 当前的识别结果。 |
| words | List< Word > | 当前句子的词信息，需要将enable_words设置为true。 |
| confidence | Double | 当前句子识别结果的置信度，取值范围：[0.0,1.0]。值越大表示置信度越高。 |

Word对象参数说明：

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| text | String | 文本。 |
| startTime | Integer | 词开始时间，单位为毫秒。 |
| endTime | Integer | 词结束时间，单位为毫秒。 |

## 4. 结束识别

通知服务端语音数据发送完成，服务端识别结束后通知客户端识别完毕。

## 服务状态码

在服务的每一次响应中，都包含status字段，即服务状态码，此处列举通用错误码、网关错误码、配置错误码表格，如下表所示。

- 通用错误码：

| 错误码 | 原因 | 解决办法 |
| --- | --- | --- |
| 40000001 | 身份认证失败 | 检查使用的令牌是否正确，是否过期。 |

| 错误码 | 原因 | 解决办法 |
| --- | --- | --- |
| 40000002 | 无效的消息 | 检查发送的消息是否符合要求。 |
| 40000010 | 免费试用已到期 | 请前往控制台开通商用服务。 |
| 403 | 令牌过期或无效的参数 | 首先检查使用的令牌是否过期，然后检查参数值设置是否合理。 |
| 40000004 | 空闲超时 | 确认是否长时间（10秒）没有发送数据到服务端。 |
| 40000005 | 请求数量过多 | 检查是否超过了并发连接数或者每秒钟请求数。如果超过并发数，建议从免费版升级到商用版，或者商用版扩容并发资源。 |
| 40000000 | 默认的客户端错误码 | 查看错误消息或提交工单。 |
| 41010120 | 客户端超时错误 | 客户端连续10秒及以上没有发送数据，导致客户端超时错误。 |
| 50000000 | 默认的服务端错误 | 如果偶现可以忽略，重复出现请提交工单。 |
| 50000001 | 内部调用错误 | 如果偶现可以忽略，重复出现请提交工单。 |
| 52010001 | 内部调用错误 | 如果偶现可以忽略，重复出现请提交工单。 |

- 网关错误：

| 错误码 | 原因 | 解决办法 |
| --- | --- | --- |
| 40010001 | 不支持的接口 | 使用了不支持的接口，如果使用SDK请提交工单。 |

| 错误码 | 原因 | 解决办法 |
| --- | --- | --- |
| 40010002 | 不支持的指令 | 使用了不支持的指令，如果使用SDK请提交工单。 |
| 40010003 | 无效的指令 | 指令格式错误，如果使用SDK请提交工单。 |
| 40010004 | 客户端提前断开连接 | 检查是否在请求正常完成之前关闭了连接。 |
| 40010005 | 任务状态错误 | 发送了当前任务状态不能处理的指令。 |

- 配置错误：

| 错误码 | 原因 | 解决办法 |
| --- | --- | --- |
| 40020105 | 应用不存在 | 解析路由时找不到应用。 |
| 40020106 | Appkey和Token不匹配 | 检查应用Appkey是否正确，是否与令牌归属同一个账号。 |
| 40020503 | 子账户鉴权失败 | 使用父账户对调用的子账户授权POP API的访问权限。 |

- 实时语音识别：

| 错误码 | 原因 | 解决办法 |
| --- | --- | --- |
| 41040201 | 客户端10s内停止发送数据 | 检查网络问题，或者检查业务中是否存在不发数据的情况。 |
| 41040202 | 客户端发送数据过快，服务器资源已经耗尽 | 检测客户端发包是否过快，是否按照1:1的实时率来发包。 |
| 41040203 | 客户端发送音频格式不正确 | 请将音频数据的格式转换为SDK目前支持的音频格式发包。 |

| 错误码 | 原因 | 解决办法 |
| --- | --- | --- |
| 41040204 | 客户端调用方法异常 | 客户端应该先调用发送请求接口，在发送请求完毕后再调用其他接口。 |
| 41040205 | 客户端设置MAXSILENCE_PARAM方法异常 | 参数MAXSILENCE_PARAM的取值范围200～6000。 |
| 41050008/41010101 | 采样率不匹配 | 检查实际语音的采样率（实时识别场景ASR只支持8K 16 bit或者16K 16 bit，单声道）、代码调用时设置的采样率（8K或者16K）和控制台上Appkey绑定的ASR模型采样率是否一致。 |
| | 不支持的采样率 | 发起调用时，请注意代码中设置的采样率参数（8000或16000）与管控台上appkey对应的模型（8k或16k）匹配。 |
| 51040101 | 服务端内部错误 | 未知错误。 |
| 51040103 | 实时语音识别服务不可用 | 查看实时语音识别服务是否有任务堆积等导致任务提交失败。 |
| 51040104 | 请求实时语音识别服务超时 | 排查实时语音识别日志。 |
| 51040105 | 调用实时语音识别服务失败 | 检查实时语音识别服务是否启动，端口是否正常开启。 |
| 51040106 | 实时语音识别服务负载均衡失败，未获取到实时语音识别服务的IP地址 | 检查VPC中的实时语音识别服务机器是否有异常。 |
| 51070103 | 后处理服务参数配置错误 | 若使用的模型为非电话—通用行业—英语模型，请通过管控台再次选择模型，刷新服务端参数。若仍无法解决问题，请提交工单。 |

# 2.移动端SDK

## 2.1. 接口说明

对长时间的语音数据流进行识别，适用于会议演讲、视频直播等长时间不间断识别的场景。

### 功能简介

NUI SDK提供更小的工具包和更完善的状态管理。为满足不同用户需求，NUI SDK既能提供全链路的语音能力，同时可做原子能力SDK进行使用，并保持接口的统一。

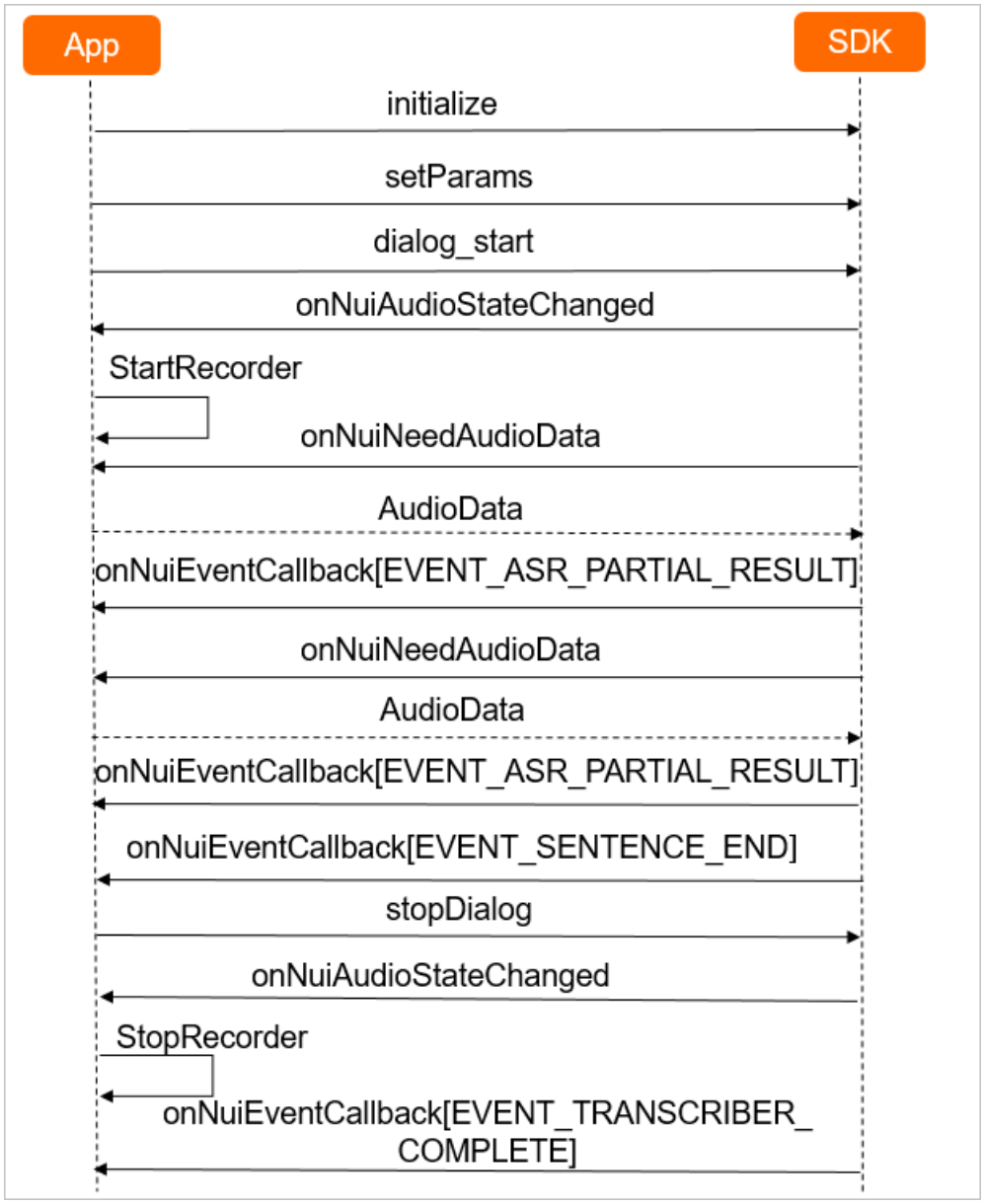
### 使用须知

- 输入格式：PCM编码、16bit采样位数、单声道（mono）。
- 音频采样率：8000Hz/16000Hz。
- 设置返回结果：是否返回中间识别结果、在后处理中添加标点、将中文数字转为阿拉伯数字输出。
- 设置多语言识别：在管控台编辑项目中进行模型选择，详情请参见模型选择。

### 服务地址

| 访问类型 | 说明 | URL |
| --- | --- | --- |
| 外网访问 | 所有服务器均可使用外网访问URL（SDK中默认设置了外网访问URL）。 | wss://nls-gateway.cn-shanghai.aliyuncs.com/ws/v1 |
| 上海ECS内网访问 | 使用阿里云上海ECS（ECS地域为华东2（上海）），可使用内网访问URL。 ECS的经典网络不能访问AnyTunnel，即不能在内网访问语音服务；如果希望使用AnyTunnel，需要创建专有网络在其内部访问。<br>说明<br>• 使用内网访问方式，将不产生ECS实例的公网流量费用。<br>• 关于ECS的网络类型请参见网络类型。 | ws://nls-gateway.cn-shanghai-internal.aliyuncs.com:80/ws/v1 |

### 交互流程




**说明**

所有服务端的响应都会在返回信息的header包含表示本次识别任务的task_id参数，请记录该值，如果出现错误，请将task_id和错误信息提交到工单。

## 1. 鉴权和初始化

客户端在与服务端建立WebSocket连接时，使用Token进行鉴权。关于Token获取请参见获取Token。

初始化参数如下。

| 参数 | 类型 | 是否必选 | 说明 |
| --- | --- | --- | --- |
| workspace | String | 是 | 工作目录路径，SDK从该路径读取配置文件。 |

| 参数 | 类型 | 是否必选 | 说明 |
| --- | --- | --- | --- |
| app_key | String | 是 | 管控台创建项目的appkey。 |
| token | String | 是 | 请确保该Token可以使用并在有效期内。<br>说明<br>Token可以在初始化时设置，也可通过参数设置进行更新。 |
| device_id | String | 是 | 设备标识，唯一表示一台设备（如Mac地址/SN/UniquePsuedoID）。 |
| debug_path | String | 否 | debug目录。当初始化SDK时的save_log参数取值为true时，该目录用于保存中间音频文件。 |
| save_wav | String | 否 | 当初始化SDK时的save_log参数取值为true时，该参数生效。表示是否保存音频debug，该数据保存在debug目录中，需要确保debug_path有效可写。 |

## 2. 开始识别

客户端发起实时语音识别请求前需要进行参数设置，各参数由SDK中setParams接口以JSON格式设置，该参数设置一次即可。各参数含义如下。

| 参数 | 类型 | 是否必选 | 说明 |
| --- | --- | --- | --- |
| appkey | String | 否 | 管控台创建的项目appkey，一般在初始化时设置。 |

| 参数 | 类型 | 是否必选 | 说明 |
| --- | --- | --- | --- |
| token | String | 否 | 如果需要更新，则进行设置。 |
| service_type | Int | 是 | 需要请求的语音服务类型，实时语音识别为“4”。 |
| direct_ip | String | 否 | 支持客户端自行DNS解析后传入IP进行访问。 |
| nls_config | JsonObject | 否 | 访问语音服务相关的参数配置，详情请参见下表。 |

参数nls_config配置如下。

| 参数 | 类型 | 是否必选 | 说明 |
| --- | --- | --- | --- |
| sr_format | String | 否 | 音频编码格式，支持OPUS编码和PCM原始音频。默认值：OPUS。<br>说明<br>如果使用8000Hz采样率，则只支持PCM格式。 |
| sample_rate | Integer | 否 | 音频采样率，默认值：16000Hz。根据音频采样率在管控台对应项目中配置支持该采样率及场景的模型。 |
| enable_intermediate_result | Boolean | 否 | 是否返回中间识别结果，默认值：False。 |
| enable_punctuation_prediction | Boolean | 否 | 是否在后处理中添加标点，默认值：False。 |

| 参数 | 类型 | 是否必选 | 说明 |
| --- | --- | --- | --- |
| enable_inverse_text_normalization | Boolean | 否 | 是否在后处理中执行ITN。设置为true时，中文数字将转为阿拉伯数字输出，默认值：False。<br>说明<br>不会对词信息进行ITN转换。 |
| customization_id | String | 否 | 自学习模型ID。 |
| vocabulary_id | String | 否 | 定制泛热词ID。 |
| max_sentence_silence | Integer | 否 | 语音断句检测阈值，静音时长超过该阈值被认为断句。取值范围：200ms～2000ms，默认值：800ms。 |
| enable_words | Boolean | 否 | 是否开启返回词信息。默认值：False。 |
| enable_ignore_sentence_timeout | Boolean | 否 | 是否忽略实时识别中的单句识别超时。默认值：False。 |
| disfluency | Boolean | 否 | 是否对识别文本进行顺滑（去除语气词、重复说等）。默认值：False。 |
| vad_model | String | 否 | 设置服务端的vad模型id，默认无需设置。 |

| 参数 | 类型 | 是否必选 | 说明 |
| --- | --- | --- | --- |
| speech_noise_threshold | float | 否 | 噪音参数阈值，取值范围：-1～+1。<br>• 取值越接近-1，判定为语音的概率越大，亦即有可能更多噪声被当作语音被误识别。<br>• 取值越接近+1，判定为噪音的越多，亦即有可能更多语音被当作噪音被拒绝掉。<br>说明<br>该参数属高级参数，调整需谨慎并进行重点测试。 |

## 3. 发送数据

客户端循环发送语音数据，持续接收识别结果：

- EVENT_SENTENCE_START事件表示服务端检测到了一句话的开始。实时语音识别服务的智能断句功能会判断出一句话的开始与结束，如：

```
{
    "header": {
        "namespace": "SpeechTranscriber",
        "name": "SentenceBegin",
        "status": 20000000,
        "message_id": "a426f3d4618447519c9d85d1a0d1****",
        "task_id": "5ec521b5aa104e3abccf3d361822****",
        "status_text": "Gateway:SUCCESS:Success."
    },
    "payload": {
        "index": 1,
        "time": 0
    }
}
```

header对象参数说明：

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| namespace | String | 消息所属的命名空间。 |

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| name | String | 消息名称。SentenceBegin表示一个句子的开始。 |
| status | Integer | 状态码。表示请求是否成功，参见错误码。 |
| message_id | String | 本次消息的ID，由SDK自动生成。 |
| task_id | String | 任务全局唯一ID，请记录该值，便于排查问题。 |
| status_text | String | 状态消息。 |

payload对象参数说明：

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| index | Integer | 句子编号，从1开始递增。 |
| time | Integer | 当前已处理的音频时长，单位：毫秒。 |

- 若enable_intermediate_result设置为true，SDK会持续多次通过onNuiEventCallback回调上报EVENT_ASR_PARTIAL_RESULT事件，即中间识别结果，如：

```
{
    "header": {
        "namespace": "SpeechTranscriber",
        "name": "TranscriptionResultChanged",
        "status": 20000000,
        "message_id": "dc21193fada84380a3b6137875ab****",
        "task_id": "5ec521b5aa104e3abccf3d361822****",
        "status_text": "Gateway:SUCCESS:Success."
    },
    "payload": {
        "index": 1,
        "time": 1835,
        "result": "北京的天",
        "confidence": 1.0,
        "words": [{
            "text": "北京",
            "startTime": 630,
            "endTime": 930
        }, {
            "text": "的",
            "startTime": 930,
            "endTime": 1110
        }, {
            "text": "天",
            "startTime": 1110,
            "endTime": 1140
        }]
    }
}
```

**说明**

header对象中name参数取值为TranscriptionResultChanged表示句子的中间识别结果。其余参数说明参见上述表格。

payload对象参数说明：

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| index | Integer | 句子编号，从1开始递增。 |
| time | Integer | 当前已处理的音频时长，单位：毫秒。 |
| result | String | 当前句子的识别结果。 |

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| words | List< Word > | 当前句子的词信息，需要将enable_words设置为true。 |
| confidence | Double | 当前句子识别结果的置信度，取值范围0.0～1.0，值越大表示置信度越高。 |

- EVENT_SENTENCE_END事件表示服务端检测到了一句话的结束，并返回该句话的识别结果，如：

```
{
    "header": {
        "namespace": "SpeechTranscriber",
        "name": "SentenceEnd",
        "status": 20000000,
        "message_id": "c3a9ae4b231649d5ae05d4af36fd****",
        "task_id": "5ec521b5aa104e3abccf3d361822****",
        "status_text": "Gateway:SUCCESS:Success."
    },
    "payload": {
        "index": 1,
        "time": 1820,
        "begin_time": 0,
        "result": "北京的天气。",
        "confidence": 1.0,
        "words": [{
            "text": "北京",
            "startTime": 630,
            "endTime": 930
        }, {
            "text": "的",
            "startTime": 930,
            "endTime": 1110
        }, {
            "text": "天气",
            "startTime": 1110,
            "endTime": 1380
        }]
    }
}
```

> **说明**
>
> header对象中name参数取值为SentenceEnd表示句子识别结束。其余参数说明参见上述表格。

payload对象参数说明：

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| index | Integer | 句子编号，从1开始递增。 |
| time | Integer | 当前已处理的音频时长，单位：毫秒。 |
| begin_time | Integer | 当前句子对应的SentenceBegin事件的时间，单位：毫秒。 |
| result | String | 当前句子的识别结果。 |
| words | List< Word > | 当前句子的词信息，需要将enable_words设置为true。 |
| confidence | Double | 当前句子识别结果的置信度，取值范围0.0～1.0，值越大表示置信度越高。 |

其中，Word对象：

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| text | String | 文本。 |
| startTime | Integer | 词开始时间，单位：毫秒。 |
| endTime | Integer | 词结束时间，单位：毫秒。 |

## 4. 结束识别

客户端通知服务端语音数据发送完毕，服务端识别结束后通知客户端识别完毕。

## 错误码

## 识别成功

| 错误码 | 错误信息 | 描述 |
| --- | --- | --- |
| 0 | SUCCESS | 成功 |

## 配置或参数错误

| 错误码 | 错误消息 | 描述 | 解决方案 |
| --- | --- | --- | --- |
| 240999 | DEFAULT_ERROR | 内部默认错误。 | 内部未明确错误，可提工单通过日志文件寻求支持。 |
| 240001 | NUI_CONFIG_INVALID | 配置文件错误。 | 配置文件错误，请确认传入的资源路径内是否有资源文件。如果是Android平台，请参考代码样例主动使用copyAssets接口。 |
| 240002 | ILLEGAL_PARAM | 非法参数。 | 请确认传入的格式是否正确，包括字段类型，值范围限制。 |
| 240003 | ILLEGAL_INIT_PARAM | 初始化参数非法。 | 请确认初始化参数格式错误或缺少必须字段。 |
| 240004 | NECESSARY_PARAM_LACK | 缺少必须参数。 | 请确认接口调用时的必须参数。 |
| 240005 | NULL_PARAM_ERROR | 参数为空。 | 确认参数是否为空。 |
| 240006 | NULL_LISTENER_ERROR | 未定义事件回调。 | 确认回调事件是否正确赋值。 |
| 240007 | NULL_DIALOG_ERROR | 无有效对话实例，一般在内部状态错误时发生。 | 请确认接口调用前是否为正确状态，可使用cancel接口恢复idle状态。仍不能解决可提工单获得支持。 |
| 240008 | NULL_ENGINE_ERROR | 无有效引擎实例，请检查是否初始化成功。 | 请确认是否初始化成功。 |
| 240009 | ILLEGAL_DATA | 传入音频数据地址或长度非法。 | 请确认传入的数据长度值。 |

## SDK状态错误

| 错误码 | 错误消息 | 描述 | 解决方案 |
| --- | --- | --- | --- |
| 240010 | ILLEGAL_REENTRANT | 退出后调用SDK接口。 | 不影响功能时可忽略。 |
| 240011 | SDK_NOT_INIT | SDK未正确初始化。 | 确认初始化返回值正确再进行其他接口使用。 |
| 240012 | SDK_ALREADY_INIT | 重复调用SDK初始化接口。 | 确认初始化调用逻辑。 |
| 240013 | DIALOG_INVALID_STATE | 内部对话状态错误。 | 请阅读SDK流程图，确认是否在错误状态下调用接口。 |
| 240014 | STATE_INVALID | SDK内部状态错误。 | 同DIALOG_INVALID_STATE。 |
| 240015 | ILLEGAL_FUNC_CALL | 该模式无法调用接口。 | 请确认接口调用是否合理。 |

## 系统调用错误

| 错误码 | 错误信息 | 描述 | 解决方法 |
| --- | --- | --- | --- |
| 240020 | MEM_ALLOC_ERROR | 内存分配错误。 | 检查内存是否不足。 |
| 240021 | FILE_ACCESS_FAIL | 文件访问错误。 | 检查文件读写权限是否提供。 |
| 240022 | CREATE_DIR_ERROR | 创建目录错误。 | 检查是否有写权限。 |

## SDK内部调用错误

| 错误码 | 错误消息 | 描述 | 解决方法 |
| --- | --- | --- | --- |
| 240030 | CREATE_NUI_ERROR | 引擎创建失败。 | 创建实例失败，一般为系统资源不足。 |
| 240031 | TEXT_DIALOG_START_FAIL | 发起文本理解失败。 | 文本转语义理解失败，检查网络连接或url以及token等信息是否有效。 |

| 错误码 | 错误消息 | 描述 | 解决方法 |
| --- | --- | --- | --- |
| 240032 | TEXT_CANCEL_START_FAIL | 取消文本理解失败。 | 可忽略。 |
| 240033 | WUW_DUPLICATE | 动态唤醒词重复。 | 可忽略。 |

## 本地引擎调用错误

| 错误码 | 错误消息 | 描述 | 解决方法 |
| --- | --- | --- | --- |
| 240040 | CEI_INIT_FAIL | 本地引擎初始化失败。 | 请确认本地引擎的模型是否有效，目录是否可读写。可提交工单跟进。 |
| 240041 | CEI_SET_PARAM_FAIL | 引擎参数设置失败。 | 请提工单。 |
| 240042 | CEI_COMPILE_GRAMMER_FAIL | 语法编译失败。 | 可忽略。 |
| 240043 | CEI_STOP_FAIL | 停止识别失败。 | 请提工单。 |
| 240044 | CEI_CANCEL_FAIL | 取消识别失败。 | 请提工单。 |
| 240045 | CEI_UNLOAD_KWS_FAIL | 取消唤醒词失败。 | 请提工单。 |
| 240046 | GET_WUW_ERROR | 获取唤醒词失败。 | 请提工单。 |

## 音频错误

| 错误码 | 错误消息 | 描述 | 解决方法 |
| --- | --- | --- | --- |
| 240050 | SELECT_RECORDER_ERROR | 选择音频设备错误。 | 内部错误，请提工单。 |
| 240051 | UPDATE_AUDIO_ERROR | 推送音频错误，一般为输入音频长度大于所需音频。 | 确认推送的音频长度是否非法。 |
| 240052 | MIC_ERROR | 连续2s未获取到音频。 | 请确认在音频数据回调中是否正确提供所需长度的音频。 |

## 调用超时错误

| 错误码 | 错误消息 | 描述 | 解决方法 |
| --- | --- | --- | --- |
| 240080 | ENGINE_INIT_TIMEOUT | 初始化引擎超时。 | 如果偶现可忽略，否则请提工单。 |
| 240081 | SET_PARAM_TIMEOUT | 设置参数超时。 | 如果偶现可忽略，否则请提工单。 |
| 240082 | SET_WUW_TIMEOUT | 设置唤醒词超时。 | 如果偶现可忽略，否则请提工单。 |
| 240083 | SELECT_RECORDER_TIMEOUT | 选择录音设备超时。 | 如果偶现可忽略，否则请提工单。 |
| 240084 | STOP_TIMEOUT | 结束对话超时。 | 如果偶现可忽略，否则请提工单。 |
| 240085 | ASR_ENGINE_STOP_TIMEOUT | 结束引擎超时。 | 如果偶现可忽略，否则请提工单。 |
| 240086 | UNLOAD_DYNAMIC_WUW_TIMEOUT | 取消动态唤醒词超时。 | 如果偶现可忽略，否则请提工单。 |
| 240087 | ADD_DYNAMIC_WUW_TIMEOUT | 增加动态唤醒词超时。 | 如果偶现可忽略，否则请提工单。 |
| 240100 | WAIT_TIMEOUT | 引擎接口调用超时。 | 如果偶现可忽略，否则请提工单。 |
| 240101 | HANDLE_API_TIMEOUT | API层接口调用超时。 | 如果偶现可忽略，否则请提工单。 |

## 网络错误

| 错误码 | 错误消息 | 描述 | 解决方法 |
| --- | --- | --- | --- |
| 240060 | CREATE_DA_REQUEST_ERROR | 创建对话助手实例失败。 | 可忽略。 |

| 错误码 | 错误消息 | 描述 | 解决方法 |
| --- | --- | --- | --- |
| 240061 | START_DA_REQUEST_ERROR | 发起对话助手请求失败。 | 可忽略。 |
| 240062 | DEFAULT_NLS_ERROR | 服务端发生错误。说明该错误同时包含服务端返回错误内容，具体请参见服务端错误码。 | 请参考服务端返回码进一步定位。 |
| 240063 | SSL_ERROR | 创建ssl实例错误。 | 偶现请忽略，否则请提交工单。 |
| 240064 | SSL_CONNECT_FAILED | ssl连接失败。 | 连接异常，请检查服务url或者本地网络连接是否正常。 |
| 240065 | HTTP_CONNECT_FAILED | HTTP连接失败。 | 服务连接错误，可通过日志文件查看HTTP返回值确认原因，或提工单跟进。 |
| 240066 | DNS_FAILED | DNS解析失败。 | 请检查本地网络是否正常，DNS服务是否正常。 |
| 240067 | CONNECT_FAILED | socket连接失败。 | 检查网络连接。 |
| 240068 | SERVER_NOT_ACCESS | 服务端无法访问。 | 请检查token是否过期或者url是否正确，仍无法定位请提工单。 |
| 240069 | SOCKET_CLOSED | socket已关闭。 | 偶现可忽略，否则提交工单跟进。 |
| 240070 | AUTH_FAILED | 鉴权失败。 | 请检查是否提供正确的ak_serect，ak_id，app_key，sdk_code，device_id等信息，以及确认是否开通足够配额。 |
| 240071 | HTTPDNS_FAILED | 使用客户端传入的IP连接失败。 | 如果使用直接传入IP进行访问，请确认IP是否可访问。 |
| 240072 | HTTP_SEND_FAILED | 文件转写HTTP发送失败。 | 确认网络连接是否正常。 |

| 错误码 | 错误消息 | 描述 | 解决方法 |
| --- | --- | --- | --- |
| 240073 | HTTP_RECEIVE_FAILED | 文件转写HTTP接收失败。 | 确认网络连接是否正常。 |
| 240074 | HTTP_RESPONSE_ERROR | 文件转写接收内容解析失败。 | 服务端返回内容错误，请提工单跟进。 |
| 240075 | HTTP_SERVER_ERROR | 文件转写服务错误，详细错误请参考服务错误码。 | 请参考服务端错误码进一步确认原因。 |

## 网络超时错误

| 错误码 | 错误消息 | 描述 | 解决方法 |
| --- | --- | --- | --- |
| 240090 | UPDATE_CONTEXT_TIMEOUT | 更新客户端信息超时 | 偶现请忽略，否则提交工单。 |
| 240091 | CONNECTION_TIMEOUT | 网络连接超时 | 偶现请忽略，否则提交工单。 |
| 240092 | PARTIAL_ASR_TIMEOUT | 获取中间识别结果超时 | 偶现请忽略，否则提交工单。 |
| 240093 | ASR_TIMEOUT | 获取最终识别结果超时 | 偶现请忽略，否则提交工单。 |
| 240094 | DIALOG_TIMEOUT | 获取对话理解结果超时 | 偶现请忽略，否则提交工单。 |
| 240095 | WWV_TIMEOUT | 获取云端唤醒确认结果超时 | 偶现请忽略，否则提交工单。 |

## 服务端错误码

当收到EVENT_ASR_ERROR事件，并且错误码为DEFAULT_NLS_ERROR（240062）或HTTP_SERVER_ERROR（240075）时，可以通过错误事件header中status字段获取服务端错误码进行进一步问题定位。

| 错误码 | 原因 | 解决方法 |
| --- | --- | --- |
| 40000001 | 身份认证失败。 | 检查使用的令牌是否正确、是否过期。 |
| 40000002 | 无效的消息。 | 检查发送的消息是否符合要求。 |

| 错误码 | 原因 | 解决方法 |
| --- | --- | --- |
| 403 | 令牌过期或无效的参数 | 1. 检查使用的令牌是否过期。<br>2. 检查参数值设置是否合理。 |
| 40000004 | 空闲超时。 | 确认是否长时间（10秒）未发送数据到服务端。 |
| 40000005 | 请求数量过多。 | 检查是否超过了并发连接数或者每秒钟请求数。如果超过并发数，建议从免费版升级到商用版，或者商用版扩容并发资源。 |
| 40000000 | 默认的客户端错误码。 | 查看错误消息或提交工单。 |
| 41010120 | 客户端超时错误。 | 客户端连续10秒及以上没有发送数据，导致客户端超时错误。 |
| 50000000 | 默认的服务端错误。 | 如果偶现可以忽略，重复出现请提交工单。 |
| 50000001 | 内部调用错误。 | 如果偶现可以忽略，重复出现请提交工单。 |
| 52010001 | 内部调用错误。 | 如果偶现可以忽略，重复出现请提交工单。 |
| 40010001 | 不支持的接口。 | 使用了不支持的接口，如果使用SDK请提交工单。 |
| 40010002 | 不支持的指令。 | 使用了不支持的指令，如果使用SDK请提交工单。 |
| 40010003 | 无效的指令。 | 指令格式错误，如果使用SDK请提交工单。 |
| 40010004 | 客户端提前断开连接。 | 检查是否在请求正常完成之前关闭了连接。 |
| 40010005 | 任务状态错误。 | 发送了当前任务状态不能处理的指令。 |
| 40020105 | 应用不存在。 | 解析路由时找不到应用。 |

| 错误码 | 原因 | 解决方法 |
| --- | --- | --- |
| 40020106 | appkey和token不匹配。 | 检查应用appkey是否正确，是否与令牌归属同一个账号。 |
| 40020503 | 子账户鉴权失败。 | 使用父账户对调用的子账户授权POP API的访问权限。 |
| 41040201 | 客户端10s内停止发送数据。 | 检查网络问题，或者检查业务中是否存在不发数据的情况。 |
| 41040202 | 客户端发送数据过快，服务器资源已经耗尽。 | 检测客户端发包是否过快，是否按照1:1的实时率发包。 |
| 41040203 | 客户端发送音频格式不正确。 | 请将音频数据的格式转换为SDK目前支持的音频格式。 |
| 41040204 | 客户端调用方法异常。 | 客户端应该先调用发送请求接口，发送请求完毕后再调用其他接口。 |
| 41040205 | 客户端设置MAXSILENCE_PARAM方法异常。 | 参数MAXSILENCE_PARAM的范围为200～2000。 |
| 41050008 | 采样率不匹配。 | 检查调用时设置的采样率和管控台上appkey绑定的ASR模型采样率是否一致。 |
| 51040101 | 服务端内部错误。 | 未知错误。 |
| 51040103 | 实时语音识别服务不可用 | 检查实时语音识别服务是否有任务堆积等导致任务提交失败 |
| 51040104 | 请求实时语音识别服务超时。 | 排查实时语音识别日志。 |
| 51040105 | 调用实时语音识别服务失败。 | 检查实时语音识别服务是否启动，端口是否正常开启。 |
| 51040106 | 实时语音识别服务负载均衡失败，未获取到实时语音识别服务的IP地址。 | 检查VPC中的实时语音识别服务机器是否有异常。 |

# 2.2. Android SDK

本文介绍了如何使用阿里云智能语音服务提供的Android NUI SDK，包括SDK下载安装、关键接口及代码示例。

## 前提条件

- 使用SDK前，首先阅读接口说明，详情请参见接口说明。
- 已获取项目Appkey，详情请参见创建项目。
- 已获取Access Token，详情请参见获取Token。

## 下载安装

1. 下载SDK和示例代码。

   > **说明**
   >
   > 请下载后在样例初始化代码中替换您的阿里云账号信息、appkey和token才可运行。

2. 解压ZIP包，在 `app/libs` 目录下获取AAR格式的SDK包，将AAR包集成到您的工程项目中进行依赖。

3. 使用Android Studio打开此工程查看参考代码实现，其中实时语音识别示例代码为SpeechTranscriberActivity.java文件，替换appkey和token后可直接运行。

## SDK关键接口

- **initialize**：初始化SDK。

```
/**
     * 初始化SDK，SDK为单例，请先释放后再次进行初始化。请勿在UI线程调用，可能会引起阻塞。
     * @param callback：事件监听回调，参见下文具体回调。
     * @param parameters：初始化参数，参见接口说明。
     * @param level：log打印级别，值越小打印越多。
     * @param save_log：是否保存log为文件，存储目录为parameter中的debug_path字段值。
     * @return：参见错误码。
     */
    public synchronized int initialize(final INativeNuiCallback callback,
                                       String parameters,
                                       final Constants.LogLevel level,
                                       final boolean save_log)
```

其中，INativeNuiCallback类型包含如下回调。

- **onNuiAudioStateChanged**：根据音频状态进行录音功能的开关。

```
/**
     * 当start/stop/cancel等接口调用时，SDK通过此回调通知App进行录音的开关操作。
     * @param state：录音需要的状态（打开/关闭）。
     */
    void onNuiAudioStateChanged(AudioState state);
```

- **onNuiNeedAudioData**：在回调中提供音频数据。

```
/**
     * 当开始识别时，此回调被连续调用，App需要在回调中进行语音数据填充。
     * @param buffer：用于给用户填充语音的存储区。
     * @param len：需要填充语音的字节数。
     * @return：实际填充的字节数。
     */
    int onNuiNeedAudioData(byte[] buffer, int len);
```

- **onNuiEventCallback**：SDK事件回调。

```
/**
     * SDK主要事件回调
     * @param event：回调事件，参见如下事件列表。
     * @param resultCode：参见错误码，在出现EVENT_ASR_ERROR事件时有效。
     * @param arg2：保留参数。
     * @param kwsResult：语音唤醒功能。
     * @param asrResult：语音识别结果。
     */
    void onNuiEventCallback(NuiEvent event, final int resultCode, final int arg2, KwsRe
sult kwsResult, AsrResult asrResult);
```

- onNuiAudioRMSChanged：音频能量值回调。

```
/**
     * 音频能量值回调
     * @param val：音频数据能量值回调，范围-160至0，一般用于UI展示语音动效
     */
    public void onNuiAudioRMSChanged(float val);
```

事件列表：

| 名称 | 说明 |
| --- | --- |
| EVENT_VAD_START | 检测到人声起点 |
| EVENT_VAD_END | 检测到人声尾点 |
| EVENT_ASR_PARTIAL_RESULT | 语音识别中间结果 |
| EVENT_ASR_RESULT | 语音识别最终结果 |
| EVENT_ASR_ERROR | 根据错误码信息判断出错原因 |
| EVENT_MIC_EEROR | 录音错误 |
| EVENT_SENTENCE_START | 实时语音识别事件，表示检测到一句话开始。 |
| EVENT_SENTENCE_END | 实时语音识别事件，表示检测到一句话结束，返回一句的完整结果。 |
| EVENT_SENTENCE_SEMANTICS | 暂不使用 |
| EVENT_TRANSCRIBER_COMPLETE | 停止语音识别后上报 |

- set_params：以JSON格式设置SDK参数。

```
/**
     * 以JSON格式设置参数
     * @param params：参见接口说明。
     * @return：参见错误码。
     */
    public synchronized int setParams(String params)
```

- startDialog：开始识别。

```
/**
     * 开始识别
     * @param vad_mode：多种模式，对于识别场景，使用P2T。
     * @return：参见错误码。
     */
public synchronized int startDialog(VadMode vad_mode, String dialog_params)
```

- stopDialog：结束识别。

```
/**
     * 结束识别，调用该接口后，服务端将返回最终识别结果并结束任务。
     * @return：参见错误码。
     */
    public synchronized int stopDailog()
```

- release：释放SDK。

```
/**
     * 释放SDK资源
     * @return：参见错误码。
     */
public synchronized int release()
```

## 调用步骤

1. 初始化SDK、录音实例。
2. 根据业务需求设置参数。
3. 调用startDialog开始识别。
4. 根据音频状态回调onNuiAudioStateChanged，打开录音机。
5. 在onNuiNeedAudioData回调中提供录音数据。
6. 在EVENT_ASR_PARTIAL_RESULT和EVENT_SENTENCE_END事件回调中获取识别结果。
7. 调用stopDialog结束识别。
8. 结束调用，使用release接口释放SDK资源。

## Proguard配置

如果代码使用了混淆，请在proguard-rules.pro中配置：

```
-keep class com.alibaba.idst.nui.*{*;}
```

## 代码示例

> **说明**
>
> 接口默认采用GetInstance获得单例，您如果有多例需求，也可以直接new对象进行使用。

**NUI SDK初始化**

```
//拷贝资源
CommonUtils.copyAssetsData(this);
//SDK初始化
int ret = NativeNui.GetInstance().initialize(this, genInitParams(path,path2), Constants.Log
Level.LOG_LEVEL_VERBOSE, true);
```

其中，genInitParams生成为String JSON字符串，包含资源目录和用户信息。其中用户信息包含如下字段。

```
private String genInitParams(String workpath, String debugpath) {
        String str = "";
        try{
            JSONObject object;
            object.put("app_key","");
            object.put("token","");
            object.put("device_id",Utils.getDeviceId());
            object.put("url","wss://nls-gateway.cn-shanghai.aliyuncs.com:443/ws/v1");
            object.put("workspace", workpath);
            object.put("debug_path",debugpath);
            str = object.toString();
        } catch (JSONException e) {
            e.printStackTrace();
        }
  return str;
}
```

### 参数设置

以JSON字符串形式进行设置。

```
private String genParams() {
  String params = "";
  try {
    JSONObject nls_config = new JSONObject();
    nls_config.put("enable_intermediate_result", true);
    JSONObject parameters = new JSONObject();
    parameters.put("nls_config", nls_config);
    //选择实时语音识别服务
    parameters.put("service_type", Constants.kServiceTypeSpeechTranscriber);
    params = parameters.toString();
  } catch (JSONException e) {
    e.printStackTrace();
  }
  return params;
}
NativeNui.GetInstance().setParams(genParams());
```

### 开始识别

通过startDialog接口开启监听。

```
NativeNui.GetInstance().startDialog(Constants.VadMode.TYPE_P2T, genDialogParams());
```

**回调处理**

- onNuiAudioStateChanged：录音状态回调，SDK内部维护录音状态，根据该状态的回调进行录音机的开关操作。

```
public void onNuiAudioStateChanged(Constants.AudioState state) {
        Log.i(TAG, "onNuiAudioStateChanged");
        if (state == Constants.AudioState.STATE_OPEN) {
            Log.i(TAG, "audio recorder start");
            mAudioRecorder.startRecording();
        } else if (state == Constants.AudioState.STATE_CLOSE) {
            Log.i(TAG, "audio recorder close");
            mAudioRecorder.release();
        } else if (state == Constants.AudioState.STATE_PAUSE) {
            Log.i(TAG, "audio recorder pause");
            mAudioRecorder.stop();
        }
    }
```

- onNuiNeedAudioData：录音数据回调，在该回调中填充录音数据。

```
public int onNuiNeedAudioData(byte[] buffer, int len) {
        int ret = 0;
        if (mAudioRecorder.getState() != AudioRecord.STATE_INITIALIZED) {
            Log.e(TAG, "audio recorder not init");
            return -1;
        }
        ret = mAudioRecorder.read(buffer, 0, len);
        return ret;
    }
```

- onNuiEventCallback：NUI SDK事件回调，请勿在事件回调中调用SDK的接口，可能引起死锁。

```
public void onNuiEventCallback(Constants.NuiEvent event, final int resultCode, final int
arg2, KwsResult kwsResult, AsrResult asrResult) {
        Log.i(TAG, "event=" + event);
        if (event == Constants.NuiEvent.EVENT_ASR_RESULT) {
            showText(asrView, asrResult.asrResult);
        } else if (event == Constants.NuiEvent.EVENT_ASR_PARTIAL_RESULT ||
                   event == Constants.NuiEvent.EVENT_SENTENCE_END) {
            showText(asrView, asrResult.asrResult);
        } else if (event == Constants.NuiEvent.EVENT_ASR_ERROR) {
            ;
        } else if (event == Constants.NuiEvent.EVENT_TRANSCRIBER_COMPLETE) {
            ;
        }
    }
```

**结束识别**

```
NativeNui.GetInstance().stopDialog();
```

# 2.3. iOS SDK

本文介绍了如何使用阿里云智能语音服务提供的iOS NUI SDK，包括SDK下载安装、关键接口及代码示例。

## 前提条件

- 使用SDK前，请先阅读接口说明，详情请参见接口说明。
- 已获取项目Appkey，详情请参见创建项目。
- 已获取Access Token，详情请参见获取Token。

## 下载安装

1. 下载SDK和示例代码。

   > **说明**
   >
   > 下载后请在样例初始化代码中替换您的阿里云账号信息、Appkey和Token才可运行。为方便集成，2.5.14版本后iOS接口使用纯Object-C接口，不再使用C++混合接口。

2. 解压ZIP包。

   将ZIP包中的nuisdk.framework添加到您的工程中，并在工程Build Phases的Link Binary With Libraries中添加nuisdk.framework。请确保在编译配置的General > Frameworks, Libraries, and Embedded Content中配置nuisdk.framework为Embed & Sign。

3. 使用Xcode打开此工程，工程中提供了参考代码以及一些直接可使用的工具类，例如音频播放录制和文件操作，您可以直接复制源码到您的实际工程进行使用。其中实时转写示例代码为SpeechTranscriberViewController。

## SDK关键接口

- **nui_initialize**：初始化SDK。

```
/**
     * 初始化SDK，SDK为单例，请先释放后再次进行初始化。请勿在UI线程调用，可能引起阻塞。
     * @param parameters: 初始化参数，参见接口说明文档
     * @param level: log打印级别，值越小打印越多
     * @param save_log: 是否保存log为文件，存储目录为parameter中的debug_path字段值
     * @return 参见错误码
     */
-(NuiResultCode) nui_initialize:(const char *)parameters
                  logLevel:(NuiSdkLogLevel)level
                  saveLog:(BOOL)save_log;
```

- **nui_set_params**：以JSON格式设置SDK参数。

```
/**
     * 以JSON格式设置参数
     * @param params: 参数信息请参见接口说明文档
     * @return 参见错误码
     */
    -(NuiResultCode) nui_set_params:(const char *)params;
```

- **nui_dialog_start**：开始识别。

```
/**
     * 开始识别
     * @param vad_mode: 多种模式，对于识别场景，请使用P2T
     * @param dialog_params: 设置识别参数，可不设置
     * @return 参见错误码
     */
    -(NuiResultCode) nui_dialog_start:(NuiVadMode)vad_mode
                      dialogParam:(const char *)dialog_params;
```

- **nui_dialog_cancel**：结束识别。

```
/**
     * 结束识别，调用该接口后，服务端将返回最终识别结果并结束任务
     * @param force: 是否强制结束而忽略最终结果，false表示停止但是等待完整结果返回
     * @return 参见错误码
     */
    -(NuiResultCode) nui_dialog_cancel:(BOOL)force;
```

- **nui_release**：释放SDK。

```
/**
     * 释放SDK资源
     * @return 参见错误码
     */
    -(NuiResultCode) nui_release;
```

- **NeoNuiSdkDelegate**

  onNuiEventCallback：SDK事件回调。

```
/**
     * SDK主要事件回调
     * @param event: 回调事件，参见如下事件列表
     * @param dialog: 会话编号，暂不使用
     * @param wuw: 语音唤醒功能使用
     * @param asr_result: 语音识别结果
     * @param finish: 本轮识别是否结束标志
     * @param resultCode: 参见错误码，在出现EVENT_ASR_ERROR事件时有效
     */
    -(void) onNuiEventCallback:(NuiCallbackEvent)nuiEvent
                        dialog:(long)dialog
                        kwsResult:(const char *)wuw
                        asrResult:(const char *)asr_result
                        ifFinish:(BOOL)finish
                        retCode:(int)code;
```

NuiCallbackEvent事件列表：

| 名称 | 说明 |
| --- | --- |
| EVENT_VAD_START | 检测到人声起点 |

| 名称 | 说明 |
| --- | --- |
| EVENT_VAD_END | 检测到人声尾点 |
| EVENT_ASR_PARTIAL_RESULT | 语音识别中间结果 |
| EVENT_ASR_RESULT | 语音识别最终结果 |
| EVENT_ASR_ERROR | 根据错误码信息判断出错原因 |
| EVENT_MIC_EEROR | 录音错误 |
| EVENT_SENTENCE_START | 实时语音识别事件，检测到一句话开始。 |
| EVENT_SENTENCE_END | 实时语音识别事件，检测一句话结束，返回一句的完整结果。 |
| EVENT_SENTENCE_SEMANTICS | 暂不使用 |
| EVENT_TRANSCRIBER_COMPLETE | 停止语音识别后上报 |

onNuiNeedAudioData：获取音频

```
/**
     * 开始识别时，此回调被连续调用，App需要在回调中进行语音数据填充，语音数据来自App的录音
     * @param audioData: 填充语音的存储区
     * @param len: 需要填充语音的字节数
     * @return: 实际填充的字节数
     */
    -(int) onNuiNeedAudioData:(char *)audioData length:(int)len;
```

onNuiAudioStateChanged：根据音频状态进行录音功能的开关。

```
/**
     * 当start/stop/cancel等接口调用时，SDK通过此回调通知App进行录音的开关操作

     * @param state: 录音需要的状态（打开/关闭）
     */
    -(void) onNuiAudioStateChanged:(NuiAudioState)state;
```

onNuiRmsChanged：音频能量事件。

```
/**
     * SDK主要事件回调
     * @param rms: 语音能量值，范围为-160至0
     */
    -(void) onNuiRmsChanged:(float) rms;
```

## 调用步骤

1. 初始化SDK、录音实例。
2. 根据业务需求配置参数。
3. 调用nui_dialog_start开始识别。
4. 根据音频状态回调audio_state_changed_callback，打开录音机。
5. 在user_data_callback回调中提供录音数据。
6. 在EVENT_ASR_PARTIAL_RESULT和EVENT_SENTENCE_END事件回调中获取识别结果。
7. 调用nui_dialog_cancel结束识别。
8. 结束调用，使用nui_release接口释放SDK资源。

## 代码示例

> **说明**
>
> 接口默认采用get_instance方式获得单例，您如果有多例需求，也可以直接alloc对象进行使用。

- NUI SDK初始化

```
NSString * initParam = [self genInitParams];
    [_nui nui_initialize:[initParam UTF8String] logLevel:LOG_LEVEL_VERBOSE saveLog:save_l
og];
```

其中，genInitParams生成为String JSON字符串，包含资源目录和用户信息。其中用户信息包含如下字段。

```
    [dictM setObject:id_string forKey:@"device_id"];
    [dictM setObject:@"" forKey:@"url"];
    [dictM setObject:@"" forKey:@"app_key"];

    [dictM setObject:@"" forKey:@"token"];
```

- 参数设置

  以JSON字符串形式进行设置。

```
-(NSString*) genParams {
    NSMutableDictionary *nls_config = [NSMutableDictionary dictionary];
    [nls_config setValue:@true forKey:@"enable_intermediate_result"];
    [nls_config setValue:@true forKey:@"enable_voice_detection"];

    NSMutableDictionary *dictM = [NSMutableDictionary dictionary];
    [dictM setObject:nls_config forKey:@"nls_config"];
    [dictM setValue:@(SERVICE_TYPE_SPEECH_TRANSCRIBER) forKey:@"service_type"];

    NSData *data = [NSJSONSerialization dataWithJSONObject:dictM options:NSJSONWritingPre
ttyPrinted error:nil];
    NSString * jsonStr = [[NSString alloc]initWithData:data encoding:NSUTF8StringEncoding
];
    return jsonStr;
}

NSString * parameters = [self genParams];
[_nui nui_set_params:[parameters UTF8String]];
```

- 开始识别

  通过nui_dialog_start接口开启监听。

```
[_nui nui_dialog_start:MODE_P2T dialogParam:[param_string UTF8String]];
```

- 回调处理
  - onNuiAudioStateChanged：录音状态回调，SDK内部维护录音状态，调用时根据该状态的回调进行录音机的开关操作。

```
-(void)onNuiAudioStateChanged:(NuiAudioState)state{
    TLog(@"onNuiAudioStateChanged state=%u", state);
    if (state == STATE_CLOSE || state == STATE_PAUSE) {
        [_voiceRecorder stop:YES];
    } else if (state == STATE_OPEN){
        self.recordedVoiceData = [NSMutableData data];
        [_voiceRecorder start];
    }
}
```

- onNuiNeedAudioData：录音数据回调，在该回调中填充录音数据。

```
-(int)onNuiNeedAudioData:(char *)audioData length:(int)len {
    static int emptyCount = 0;
    @autoreleasepool {
        @synchronized(_recordedVoiceData){
            if (_recordedVoiceData.length > 0) {
                int recorder_len = 0;
                if (_recordedVoiceData.length > len)
                    recorder_len = len;
                else
                    recorder_len = _recordedVoiceData.length;
                NSData *tempData = [_recordedVoiceData subdataWithRange:NSMakeRange(0,
recorder_len)];
                [tempData getBytes:audioData length:recorder_len];
                tempData = nil;
                NSInteger remainLength = _recordedVoiceData.length - recorder_len;
                NSRange range = NSMakeRange(recorder_len, remainLength);
                [_recordedVoiceData setData:[_recordedVoiceData subdataWithRange:range]
];
                emptyCount = 0;
                return recorder_len;
            } else {
                if (emptyCount++ >= 50) {
                    TLog(@"_recordedVoiceData length = %lu! empty 50times.", (unsigned
long)_recordedVoiceData.length);
                    emptyCount = 0;
                }
                return 0;
            }

        }
    }
    return 0;
}
```

- onNuiEventCallback：NUI SDK事件回调，请勿在事件回调中调用SDK的接口，可能引起死锁。

```
-(void)onNuiEventCallback:(NuiCallbackEvent)nuiEvent
                   dialog:(long)dialog
                kwsResult:(const char *)wuw
                asrResult:(const char *)asr_result
                 ifFinish:(bool)finish
                  retCode:(int)code {
    TLog(@"onNuiEventCallback event %d finish %d", nuiEvent, finish);
    if (nuiEvent == EVENT_ASR_PARTIAL_RESULT || nuiEvent == EVENT_SENTENCE_END) {
        TLog(@"ASR RESULT %s finish %d", asr_result, finish);
        NSString *result = [NSString stringWithUTF8String:asr_result];
        [myself showAsrResult:result];
    } else if (nuiEvent == EVENT_ASR_ERROR) {
        TLog(@"EVENT_ASR_ERROR error[%d]", code);
    } else if (nuiEvent == EVENT_MIC_ERROR) {
        TLog(@"MIC ERROR");
        [_voiceRecorder stop:true];
        [_voiceRecorder start];
    }

    if (finish) {
        [myself showStart];
    }

    return;
}
```

- 结束识别

```
[_nui nui_dialog_cancel:NO];
```

# 2.4. 错误码

为方便进行问题定位，本文为您介绍语音识别业务中详细的错误码信息。

## 错误码

### 识别成功

| 错误码 | 错误信息 | 描述 |
| --- | --- | --- |
| 0 | SUCCESS | 成功 |

### 配置或参数错误

| 错误码 | 错误消息 | 描述 | 解决方案 |
| --- | --- | --- | --- |
| 240999 | DEFAULT_ERROR | 内部默认错误。 | 内部未明确错误，可提工单通过日志文件寻求支持。 |
| 240001 | NUI_CONFIG_INVALID | 配置文件错误。 | 配置文件错误，请确认传入的资源路径内是否有资源文件。如果是Android平台，请参考代码样例主动使用copyAssets接口。 |
| 240002 | ILLEGAL_PARAM | 非法参数。 | 请确认传入的格式是否正确，包括字段类型，值范围限制。 |
| 240003 | ILLEGAL_INIT_PARAM | 初始化参数非法。 | 请确认初始化参数格式错误或缺少必须字段。 |
| 240004 | NECESSARY_PARAM_LACK | 缺少必须参数。 | 请确认接口调用时的必须参数。 |
| 240005 | NULL_PARAM_ERROR | 参数为空。 | 确认参数是否为空。 |
| 240006 | NULL_LISTENER_ERROR | 未定义事件回调。 | 确认回调事件是否正确赋值。 |
| 240007 | NULL_DIALOG_ERROR | 无有效对话实例，一般在内部状态错误时发生。 | 请确认接口调用前是否为正确状态，可使用cancel接口恢复idle状态。仍不能解决可提工单获得支持。 |
| 240008 | NULL_ENGINE_ERROR | 无有效引擎实例，请检查是否初始化成功。 | 请确认是否初始化成功。 |
| 240009 | ILLEGAL_DATA | 传入音频数据地址或长度非法。 | 请确认传入的数据长度值。 |

## SDK状态错误

| 错误码 | 错误消息 | 描述 | 解决方案 |
| --- | --- | --- | --- |
| 240010 | ILLEGAL_REENTRANT | 退出后调用SDK接口。 | 不影响功能时可忽略。 |

| 错误码 | 错误消息 | 描述 | 解决方案 |
| --- | --- | --- | --- |
| 240011 | SDK_NOT_INIT | SDK未正确初始化。 | 确认初始化返回值正确再进行其他接口使用。 |
| 240012 | SDK_ALREADY_INIT | 重复调用SDK初始化接口。 | 确认初始化调用逻辑。 |
| 240013 | DIALOG_INVALID_STATE | 内部对话状态错误。 | 请阅读SDK流程图，确认是否在错误状态下调用接口。 |
| 240014 | STATE_INVALID | SDK内部状态错误。 | 同DIALOG_INVALID_STATE。 |
| 240015 | ILLEGAL_FUNC_CALL | 该模式无法调用接口。 | 请确认接口调用是否合理。 |

## 系统调用错误

| 错误码 | 错误信息 | 描述 | 解决方法 |
| --- | --- | --- | --- |
| 240020 | MEM_ALLOC_ERROR | 内存分配错误。 | 检查内存是否不足。 |
| 240021 | FILE_ACCESS_FAIL | 文件访问错误。 | 检查文件读写权限是否提供。 |
| 240022 | CREATE_DIR_ERROR | 创建目录错误。 | 检查是否有写权限。 |

## SDK内部调用错误

| 错误码 | 错误消息 | 描述 | 解决方法 |
| --- | --- | --- | --- |
| 240030 | CREATE_NUI_ERROR | 引擎创建失败。 | 创建实例失败，一般为系统资源不足。 |
| 240031 | TEXT_DIALOG_START_FAIL | 发起文本理解失败。 | 文本转语义理解失败，检查网络连接或url以及token等信息是否有效。 |
| 240032 | TEXT_CANCEL_START_FAIL | 取消文本理解失败。 | 可忽略。 |
| 240033 | WUW_DUPLICATE | 动态唤醒词重复。 | 可忽略。 |

## 本地引擎调用错误

| 错误码 | 错误消息 | 描述 | 解决方法 |
| --- | --- | --- | --- |
| 240040 | CEI_INIT_FAIL | 本地引擎初始化失败。 | 请确认本地引擎的模型是否有效，目录是否可读写。可提交工单跟进。 |
| 240041 | CEI_SET_PARAM_FAIL | 引擎参数设置失败。 | 请提工单。 |
| 240042 | CEI_COMPILE_GRAMMER_FAIL | 语法编译失败。 | 可忽略。 |
| 240043 | CEI_STOP_FAIL | 停止识别失败。 | 请提工单。 |
| 240044 | CEI_CANCEL_FAIL | 取消识别失败。 | 请提工单。 |
| 240045 | CEI_UNLOAD_KWS_FAIL | 取消唤醒词失败。 | 请提工单。 |
| 240046 | GET_WUW_ERROR | 获取唤醒词失败。 | 请提工单。 |

## 音频错误

| 错误码 | 错误消息 | 描述 | 解决方法 |
| --- | --- | --- | --- |
| 240050 | SELECT_RECORDER_ERROR | 选择音频设备错误。 | 内部错误，请提工单。 |
| 240051 | UPDATE_AUDIO_ERROR | 推送音频错误，一般为输入音频长度大于所需音频。 | 确认推送的音频长度是否非法。 |
| 240052 | MIC_ERROR | 连续2s未获取到音频。 | 请确认在音频数据回调中是否正确提供所需长度的音频。 |

## 调用超时错误

| 错误码 | 错误消息 | 描述 | 解决方法 |
| --- | --- | --- | --- |
| 240080 | ENGINE_INIT_TIMEOUT | 初始化引擎超时。 | 如果偶现可忽略，否则请提工单。 |

| 错误码 | 错误消息 | 描述 | 解决方法 |
| --- | --- | --- | --- |
| 240081 | SET_PARAM_TIMEOUT | 设置参数超时。 | 如果偶现可忽略，否则请提工单。 |
| 240082 | SET_WUW_TIMEOUT | 设置唤醒词超时。 | 如果偶现可忽略，否则请提工单。 |
| 240083 | SELECT_RECORDER_TIMEOUT | 选择录音设备超时。 | 如果偶现可忽略，否则请提工单。 |
| 240084 | STOP_TIMEOUT | 结束对话超时。 | 如果偶现可忽略，否则请提工单。 |
| 240085 | ASR_ENGINE_STOP_TIMEOUT | 结束引擎超时。 | 如果偶现可忽略，否则请提工单。 |
| 240086 | UNLOAD_DYNAMIC_WUW_TIMEOUT | 取消动态唤醒词超时。 | 如果偶现可忽略，否则请提工单。 |
| 240087 | ADD_DYNAMIC_WUW_TIMEOUT | 增加动态唤醒词超时。 | 如果偶现可忽略，否则请提工单。 |
| 240100 | WAIT_TIMEOUT | 引擎接口调用超时。 | 如果偶现可忽略，否则请提工单。 |
| 240101 | HANDLE_API_TIMEOUT | API层接口调用超时。 | 如果偶现可忽略，否则请提工单。 |

## 网络错误

| 错误码 | 错误消息 | 描述 | 解决方法 |
| --- | --- | --- | --- |
| 240060 | CREATE_DA_REQUEST_ERROR | 创建对话助手实例失败。 | 可忽略。 |
| 240061 | START_DA_REQUEST_ERROR | 发起对话助手请求失败。 | 可忽略。 |

| 错误码 | 错误消息 | 描述 | 解决方法 |
| --- | --- | --- | --- |
| 240062 | DEFAULT_NLS_ERROR | 服务端发生错误。说明该错误同时包含服务端返回错误内容，具体请参见服务端错误码。 | 请参考服务端返回码进一步定位。 |
| 240063 | SSL_ERROR | 创建ssl实例错误。 | 偶现请忽略，否则请提交工单。 |
| 240064 | SSL_CONNECT_FAILED | ssl连接失败。 | 连接异常，请检查服务url或者本地网络连接是否正常。 |
| 240065 | HTTP_CONNECT_FAILED | HTTP连接失败。 | 服务连接错误，可通过日志文件查看HTTP返回值确认原因，或提工单跟进。 |
| 240066 | DNS_FAILED | DNS解析失败。 | 请检查本地网络是否正常，DNS服务是否正常。 |
| 240067 | CONNECT_FAILED | socket连接失败。 | 检查网络连接。 |
| 240068 | SERVER_NOT_ACCESS | 服务端无法访问。 | 请检查token是否过期或者url是否正确，仍无法定位请提工单。 |
| 240069 | SOCKET_CLOSED | socket已关闭。 | 偶现可忽略，否则提交工单跟进。 |
| 240070 | AUTH_FAILED | 鉴权失败。 | 请检查是否提供正确的ak_serect，ak_id，app_key，sdk_code，device_id等信息，以及确认是否开通足够配额。 |
| 240071 | HTTPDNS_FAILED | 使用客户端传入的IP连接失败。 | 如果使用直接传入IP进行访问，请确认IP是否可访问。 |
| 240072 | HTTP_SEND_FAILED | 文件转写HTTP发送失败。 | 确认网络连接是否正常。 |
| 240073 | HTTP_RECEIVE_FAILED | 文件转写HTTP接收失败。 | 确认网络连接是否正常。 |

| 错误码 | 错误消息 | 描述 | 解决方法 |
| --- | --- | --- | --- |
| 240074 | HTTP_RESPONSE_ERROR | 文件转写接收内容解析失败。 | 服务端返回内容错误，请提工单跟进。 |
| 240075 | HTTP_SERVER_ERROR | 文件转写服务错误，详细错误请参考服务错误码。 | 请参考服务端错误码进一步确认原因。 |

## 网络超时错误

| 错误码 | 错误消息 | 描述 | 解决方法 |
| --- | --- | --- | --- |
| 240090 | UPDATE_CONTEXT_TIMEOUT | 更新客户端信息超时 | 偶现请忽略，否则提交工单。 |
| 240091 | CONNECTION_TIMEOUT | 网络连接超时 | 偶现请忽略，否则提交工单。 |
| 240092 | PARTIAL_ASR_TIMEOUT | 获取中间识别结果超时 | 偶现请忽略，否则提交工单。 |
| 240093 | ASR_TIMEOUT | 获取最终识别结果超时 | 偶现请忽略，否则提交工单。 |
| 240094 | DIALOG_TIMEOUT | 获取对话理解结果超时 | 偶现请忽略，否则提交工单。 |
| 240095 | WWV_TIMEOUT | 获取云端唤醒确认结果超时 | 偶现请忽略，否则提交工单。 |

## 服务端错误码

当收到EVENT_ASR_ERROR事件，并且错误码为DEFAULT_NLS_ERROR（240062）或HTTP_SERVER_ERROR（240075）时，可以通过错误事件header中status字段获取服务端错误码进行进一步问题定位。

| 错误码 | 原因 | 解决方法 |
| --- | --- | --- |
| 40000001 | 身份认证失败。 | 检查使用的令牌是否正确、是否过期。 |
| 40000002 | 无效的消息。 | 检查发送的消息是否符合要求。 |
| 403 | 令牌过期或无效的参数 | 1. 检查使用的令牌是否过期。<br>2. 检查参数值设置是否合理。 |

| 错误码 | 原因 | 解决方法 |
| --- | --- | --- |
| 40000004 | 空闲超时。 | 确认是否长时间（10秒）未发送数据到服务端。 |
| 40000005 | 请求数量过多。 | 检查是否超过了并发连接数或者每秒钟请求数。如果超过并发数，建议从免费版升级到商用版，或者商用版扩容并发资源。 |
| 40000000 | 默认的客户端错误码。 | 查看错误消息或提交工单。 |
| 40000010 | 免费试用已到期。 | 请前往控制台开通商用服务。 |
| 41010120 | 客户端超时错误。 | 客户端连续10秒及以上没有发送数据，导致客户端超时错误。 |
| 50000000 | 默认的服务端错误。 | 如果偶现可以忽略，重复出现请提交工单。 |
| 50000001 | 内部调用错误。 | 如果偶现可以忽略，重复出现请提交工单。 |
| 52010001 | 内部调用错误。 | 如果偶现可以忽略，重复出现请提交工单。 |
| 40010001 | 不支持的接口。 | 使用了不支持的接口，如果使用SDK请提交工单。 |
| 40010002 | 不支持的指令。 | 使用了不支持的指令，如果使用SDK请提交工单。 |
| 40010003 | 无效的指令。 | 指令格式错误，如果使用SDK请提交工单。 |
| 40010004 | 客户端提前断开连接。 | 检查是否在请求正常完成之前关闭了连接。 |
| 40010005 | 任务状态错误。 | 发送了当前任务状态不能处理的指令。 |
| 40020105 | 应用不存在。 | 解析路由时找不到应用。 |

| 错误码 | 原因 | 解决方法 |
| --- | --- | --- |
| 40020106 | appkey和token不匹配。 | 检查应用appkey是否正确，是否与令牌归属同一个账号。 |
| 40020503 | 子账户鉴权失败。 | 使用父账户对调用的子账户授权POP API的访问权限。 |
| 41040201 | 客户端10s内停止发送数据。 | 检查网络问题，或者检查业务中是否存在不发数据的情况。 |
| 41040202 | 客户端发送数据过快，服务器资源已经耗尽。 | 检测客户端发包是否过快，是否按照1:1的实时率发包。 |
| 41040203 | 客户端发送音频格式不正确。 | 请将音频数据的格式转换为SDK目前支持的音频格式。 |
| 41040204 | 客户端调用方法异常。 | 客户端应该先调用发送请求接口，发送请求完毕后再调用其他接口。 |
| 41040205 | 客户端设置MAXSILENCE_PARAM方法异常。 | 参数MAXSILENCE_PARAM的范围为200～2000。 |
| 41050008 | 采样率不匹配。 | 检查调用时设置的采样率和管控台上appkey绑定的ASR模型采样率是否一致。 |
| 51040101 | 服务端内部错误。 | 未知错误。 |
| 51040103 | 实时语音识别服务不可用 | 检查实时语音识别服务是否有任务堆积等导致任务提交失败 |
| 51040104 | 请求实时语音识别服务超时。 | 排查实时语音识别日志。 |
| 51040105 | 调用实时语音识别服务失败。 | 检查实时语音识别服务是否启动，端口是否正常开启。 |
| 51040106 | 实时语音识别服务负载均衡失败，未获取到实时语音识别服务的IP地址。 | 检查VPC中的实时语音识别服务机器是否有异常。 |

# 3.Java SDK

本文介绍如何使用阿里云智能语音服务提供的Java SDK，包括SDK的安装方法及SDK代码示例。

## 前提条件

- 在使用SDK之前，请先阅读接口说明，详情请参见接口说明。
- 从2.1.0版本开始原有nls-sdk-long-asr更名为nls-sdk-transcriber。升级时需确认已删除nls-sdk-long-asr，并按编译提示添加相应回调方法。

## 下载安装

1. 从maven服务器下载最新版本SDK，下载Demo源码ZIP包。

```
<dependency>
      <groupId>com.alibaba.nls</groupId>
      <artifactId>nls-sdk-transcriber</artifactId>
      <version>2.2.1</version>
</dependency>
```

   Demo解压后，在pom目录运行 `mvn package` ，会在target目录生成可执行JAR：nls-example-transcriber-2.0.0-jar-with-dependencies.jar，将JAR包拷贝到目标服务器，用于快速验证及服务压测。

2. 服务验证。

   运行如下代码，并按提示提供相应参数。运行后在命令执行目录生成logs/nls.log。

```
java -cp nls-example-transcriber-2.0.0-jar-with-dependencies.jar com.alibaba.nls.client
.SpeechTranscriberDemo
```

3. 服务压测。

   运行如下代码，并按提示提供相应参数。其中阿里云服务url参数为：`wss://nls-gateway.cn-shanghai.aliyuncs.com/ws/v1` ，语音文件为16k采样率PCM格式文件，并发数根据您的购买情况进行选择。

```
java -jar nls-example-transcriber-2.0.0-jar-with-dependencies.jar
```

> **说明**
>
> 自行压测超过2路并发将产生费用。

## 关键接口

- NlsClient：语音处理客户端，利用该客户端可以进行一句话识别、实时语音识别和语音合成的语音处理任务。该客户端为线程安全，建议全局仅创建一个实例。
- SpeechTranscriber：实时语音识别类，通过该接口设置请求参数，发送请求及声音数据。非线程安全。
- SpeechTranscriberListener：实时语音识别结果监听类，监听识别结果。非线程安全。

更多介绍，请参见Java API接口说明。

**注意**

SDK调用注意事项：

- NlsClient使用了Netty框架，NlsClient对象的创建会消耗一定时间和资源，一经创建可以重复使用。建议调用程序将NlsClient的创建和关闭与程序本身的生命周期相结合。
- SpeechTranscriber对象不可重复使用，一个识别任务对应一个SpeechTranscriber对象。例如，N个音频文件要进行N次识别任务，创建N个SpeechTranscriber对象。
- SpeechTranscriberListener对象和SpeechTranscriber对象是一一对应的，不能在不同SpeechTranscriber对象使用同一个SpeechTranscriberListener对象，否则不能将各识别任务区分开。
- Java SDK依赖Netty网络库，如果您的应用依赖Netty，其版本需更新至4.1.17.Final及以上。

## 示例代码

**说明**

- 下载nls-sample-16k.wav。

  示例中使用的音频文件为16000Hz采样率，请在管控台中将appkey对应项目的模型设置为**通用**模型，以获取正确的识别结果；如果使用其他音频，请设置为支持该音频场景的模型，关于模型设置，请参见管理项目。
- 示例中使用了SDK内置的默认外网访问服务端URL，如果您需要使用阿里云上海ECS内网访问服务端URL，则在创建NlsClient对象时，设置内网访问的URL：

```
client = new NlsClient("ws://nls-gateway.cn-shanghai-internal.aliyuncs.com/ws/v1"
, accessToken);
```

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import com.alibaba.nls.client.protocol.InputFormatEnum;
import com.alibaba.nls.client.protocol.NlsClient;
import com.alibaba.nls.client.protocol.SampleRateEnum;
import com.alibaba.nls.client.protocol.asr.SpeechTranscriber;
import com.alibaba.nls.client.protocol.asr.SpeechTranscriberListener;
import com.alibaba.nls.client.protocol.asr.SpeechTranscriberResponse;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
/**
 * 此示例演示了：
 * ASR实时识别API调用。
 * 动态获取token。
 * 通过本地模拟实时流发送。
 * 识别耗时计算。
 */
public class SpeechTranscriberDemo {
    private String appKey;
    private NlsClient client;
    private static final Logger logger = LoggerFactory.getLogger(SpeechTranscriberDemo.clas
```

```
s);

    public SpeechTranscriberDemo(String appKey, String id, String secret, String url) {
        this.appKey = appKey;
        //应用全局创建一个NlsClient实例，默认服务地址为阿里云线上服务地址。
        //获取token，实际使用时注意在accessToken.getExpireTime()过期前再次获取。
        AccessToken accessToken = new AccessToken(id, secret);
        try {
            accessToken.apply();
            System.out.println("get token: " + ", expire time: " + accessToken.getExpireTim
e());
            if(url.isEmpty()) {
                client = new NlsClient(accessToken.getToken());
            }else {
                client = new NlsClient(url, accessToken.getToken());
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static SpeechTranscriberListener getTranscriberListener() {
        SpeechTranscriberListener listener = new SpeechTranscriberListener() {
            //识别出中间结果。仅当setEnableIntermediateResult为true时，才会返回该消息。
            @Override
            public void onTranscriptionResultChange(SpeechTranscriberResponse response) {
                System.out.println("task_id: " + response.getTaskId() +
                    ", name: " + response.getName() +
                    //状态码“20000000”表示正常识别。
                    ", status: " + response.getStatus() +
                    //句子编号，从1开始递增。
                    ", index: " + response.getTransSentenceIndex() +
                    //当前的识别结果。
                    ", result: " + response.getTransSentenceText() +
                    //当前已处理的音频时长，单位为毫秒。
                    ", time: " + response.getTransSentenceTime());
            }

            @Override
            public void onTranscriberStart(SpeechTranscriberResponse response) {
                //task_id是调用方和服务端通信的唯一标识，遇到问题时，需要提供此task_id。
                System.out.println("task_id: " + response.getTaskId() + ", name: " + respon
se.getName() + ", status: " + response.getStatus());
            }

            @Override
            public void onSentenceBegin(SpeechTranscriberResponse response) {
                System.out.println("task_id: " + response.getTaskId() + ", name: " + respon
se.getName() + ", status: " + response.getStatus());

            }

            //识别出一句话。服务端会智能断句，当识别到一句话结束时会返回此消息。
            @Override
            public void onSentenceEnd(SpeechTranscriberResponse response) {
```

```
            public void onSentenceEnd(SpeechTranscriberResponse response) {
                System.out.println("task_id: " + response.getTaskId() +
                    ", name: " + response.getName() +
                    //状态码“20000000”表示正常识别。
                    ", status: " + response.getStatus() +
                    //句子编号，从1开始递增。
                    ", index: " + response.getTransSentenceIndex() +
                    //当前的识别结果。
                    ", result: " + response.getTransSentenceText() +
                    //置信度
                    ", confidence: " + response.getConfidence() +
                    //开始时间
                    ", begin_time: " + response.getSentenceBeginTime() +
                    //当前已处理的音频时长，单位为毫秒。
                    ", time: " + response.getTransSentenceTime());
            }

            //识别完毕
            @Override
            public void onTranscriptionComplete(SpeechTranscriberResponse response) {
                System.out.println("task_id: " + response.getTaskId() + ", name: " + respon
se.getName() + ", status: " + response.getStatus());
            }

            @Override
            public void onFail(SpeechTranscriberResponse response) {
                //task_id是调用方和服务端通信的唯一标识，遇到问题时，需要提供此task_id。
                System.out.println("task_id: " + response.getTaskId() +  ", status: " + res
ponse.getStatus() + ", status_text: " + response.getStatusText());
            }
        };

        return listener;
    }

    //根据二进制数据大小计算对应的同等语音长度。
    //sampleRate：支持8000或16000。
    public static int getSleepDelta(int dataSize, int sampleRate) {
        // 仅支持16位采样。
        int sampleBytes = 16;
        // 仅支持单通道。
        int soundChannel = 1;
        return (dataSize * 10 * 8000) / (160 * sampleRate);
    }

    public void process(String filepath) {
        SpeechTranscriber transcriber = null;
        try {
            //创建实例、建立连接。
            transcriber = new SpeechTranscriber(client, getTranscriberListener());
            transcriber.setAppKey(appKey);
            //输入音频编码方式。
            transcriber.setFormat(InputFormatEnum.PCM);
            //输入音频采样率。
            transcriber.setSampleRate(SampleRateEnum.SAMPLE_RATE_16K);
```

```
            //是否返回中间识别结果。
            transcriber.setEnableIntermediateResult(false);
            //是否生成并返回标点符号。
            transcriber.setEnablePunctuation(true);
            //是否将返回结果规整化，比如将一百返回为100。
            transcriber.setEnableITN(false);

            //设置vad断句参数。默认值：800ms，有效值：200ms～2000ms。
            //transcriber.addCustomedParam("max_sentence_silence", 600);
            //设置是否语义断句。
            //transcriber.addCustomedParam("enable_semantic_sentence_detection",false);
            //设置是否开启顺滑。
            //transcriber.addCustomedParam("disfluency",true);
            //设置是否开启词模式。
            //transcriber.addCustomedParam("enable_words",true);
           //设置vad噪音阈值参数，参数取值为-1～+1，如-0.9、-0.8、0.2、0.9。
            //取值越趋于-1，判定为语音的概率越大，亦即有可能更多噪声被当成语音被误识别。
            //取值越趋于+1，判定为噪音的越多，亦即有可能更多语音段被当成噪音被拒绝识别。
            //该参数属高级参数，调整需慎重和重点测试。
            //transcriber.addCustomedParam("speech_noise_threshold",0.3);
            //设置训练后的定制语言模型id。
            //transcriber.addCustomedParam("customization_id","你的定制语言模型id");
            //设置训练后的定制热词id。
            //transcriber.addCustomedParam("vocabulary_id","你的定制热词id");
            //设置是否忽略单句超时。
            transcriber.addCustomedParam("enable_ignore_sentence_timeout",false);
            //vad断句开启后处理。
            //transcriber.addCustomedParam("enable_vad_unify_post",false);

            //此方法将以上参数设置序列化为JSON发送给服务端，并等待服务端确认。
            transcriber.start();

            File file = new File(filepath);
            FileInputStream fis = new FileInputStream(file);
            byte[] b = new byte[3200];
            int len;
            while ((len = fis.read(b)) > 0) {
                logger.info("send data pack length: " + len);
                transcriber.send(b, len);
                //本案例用读取本地文件的形式模拟实时获取语音流并发送的，因为读取速度较快，这里需要设置
sleep。
                //如果实时获取语音则无需设置sleep，如果是8k采样率语音第二个参数设置为8000。
                int deltaSleep = getSleepDelta(len, 16000);
                Thread.sleep(deltaSleep);
            }

            //通知服务端语音数据发送完毕，等待服务端处理完成。
            long now = System.currentTimeMillis();
            logger.info("ASR wait for complete");
            transcriber.stop();
            logger.info("ASR latency : " + (System.currentTimeMillis() - now) + " ms");
        } catch (Exception e) {
            System.err.println(e.getMessage());
        } finally {
```

```
            if (null != transcriber) {
                transcriber.close();
            }
        }
    }

    public void shutdown() {
        client.shutdown();
    }

    public static void main(String[] args) throws Exception {
        String appKey = "填写appkey";
        String id = "填写AccessKey Id";
        String secret = "填写AccessKey Secret";
        String url = ""; // 默认值：wss://nls-gateway.cn-shanghai.aliyuncs.com/ws/v1。

        if (args.length == 3) {
            appKey   = args[0];
            id       = args[1];
            secret   = args[2];
        } else if (args.length == 4) {
            appKey   = args[0];
            id       = args[1];
            secret   = args[2];
            url      = args[3];
        } else {
            System.err.println("run error, need params(url is optional): " + "<app-key> <Ac
cessKeyId> <AccessKeySecret> [url]");
            System.exit(-1);
        }
        //本案例使用本地文件模拟发送实时流数据。您在实际使用时，可以实时采集或接收语音流并发送到ASR服
务端。
        String filepath = "nls-sample-16k.wav";
        SpeechTranscriberDemo demo = new SpeechTranscriberDemo(appKey, id, secret, url);
        demo.process(filepath);
        demo.shutdown();
    }
}
```

# 4.C++ SDK

本文介绍如何使用阿里云智能语音服务提供的C++ SDK，包括SDK的安装方法及SDK代码示例。

## SDK下载

> **说明**
>
> - 当前最新版本：3.1.13，支持Linux、Windows及Android平台。发布日期：2022年04月18日。
> - 使用SDK前，请先阅读接口说明，详情请参见接口说明。
> - 该版本C++ SDK API 3.1和上一版本API 2.0（已下线）定义有区别，本文以当前版本为例进行介绍。

可通过以下两种方法获取SDK。

- 方法一：从GitHub获取最新源码，详细编译和运行方式可见下文，或查看源码中的readme.md。

```
git clone --depth 1 https://github.com/aliyun/alibabacloud-nls-cpp-sdk
```

- 方法二：直接从下文表中选取需要SDK包进行下载。其中SDK源码包为SDK原始代码，需要通过下文编译方法生成集成所需的库文件。其他对应平台的SDK包内含相关库文件、头文件，无需编译。

| 最新SDK包 | 平台 | MD5 |
| --- | --- | --- |
| alibabacloud-nls-cpp-sdk3.1.13-master_d1b6694.zip | SDK源码 | f9f0b479c7008abcc20089e38df8a4ae |
| NlsCppSdk_Linux_3.1.13_d1b6694.tar.gz | Linux | 7849eee151310b22a069ec1c77b93583 |
| NlsCppSdk_Android_3.1.13_d1b6694.tar.gz | Android | 0107c1ac58c92b94f35f6d98e2cbf6cf |
| NlsCppSdk_Windows_3.1.13_d1b6694.zip | Windows | 9a658492e7dba4977451e74f4c88ebe0 |

其中：

- alibabacloud-nls-cpp-sdk<version>-master_<github commit id>.zip为SDK源码包。
- NlsCppSdk_<平台>_<版本号>_<github commit id>.tar.gz为对应平台下开发需要的SDK包，详见内部readme.md。

## SDK包文件说明

- scripts/build_linux.sh：SDK源码中，以Linux平台为例的示例编译脚本。
- CMakeLists.txt：SDK源码中，以Linux/Android平台为例的示例代码工程CMakeList文件。

- demo目录：SDK包中，集成示例代码，以Linux平台为例，如下表所示。

| 文件名 | 描述 |
| --- | --- |
| speechRecognizerDemo.cpp | 一句话识别示例。 |
| speechSynthesizerDemo.cpp | 语音合成示例。 |
| speechTranscriberDemo.cpp | 实时语音识别示例。 |
| fileTransferDemo.cpp | 录音文件识别示例。 |

- resource目录：SDK源码中，语音服务范例音频，可用于功能测试，如下表所示。

| 文件名 | 描述 |
| --- | --- |
| ◦ test0.wav<br>◦ test1.wav<br>◦ test2.wav<br>◦ test3.wav | 测试音频（16k采样频率、16bit采样位数的音频文件）。 |

- include：SDK源码中的头文件，如下表所示。

| 文件名 | 描述 |
| --- | --- |
| nlsClient.h | SDK实例。 |
| nlsEvent.h | 回调事件说明。 |
| nlsGlobal.h | SDK全局头文件。 |
| nlsToken.h | SDK Access Token实例。 |
| iNlsRequest.h | NLS请求基础头文件。 |
| speechRecognizerRequest.h | 一句话识别。 |

| 文件名 | 描述 |
| --- | --- |
| speechSynthesizerRequest.h | 语音合成、长文本语音合成。 |
| speechTranscriberRequest.h | 实时音频流识别。 |
| FileTrans.h | 录音文件识别。 |

- lib：SDK库文件。
- readme.md：SDK说明。
- release.log：版本说明。
- version：版本号。

## 编译运行

### Linux平台编译

1. 安装工具的最低版本要求如下：
   - CMake 3.1
   - Glibc 2.5
   - Gcc 4.1.2
2. 在Linux终端运行如下脚本。
   i. 进入SDK源码的根目录。
   ii. 生成SDK库文件和可执行程序：srDemo（一句话识别）、stDemo（实时语音识别）、syDemo（语音合成）、daDemo（语音对话）。

```
./scripts/build_linux.sh
```

   iii. 查看范例使用方式。

```
cd build/demo
./stDemo
```

### Android平台编译

支持arm64-v8a、armeabi、armeabi-v7a、x86、x86_64编译指令，在SDK源码运行如下编译脚本。

```
./scripts/build_android.sh #默认增量编译，生成arm64-v8a架构Debug版本
./scripts/build_android.sh all debug arm64-v8a    #全量编译，生成arm64-v8a架构Debug版本
./scripts/build_android.sh incr debug arm64-v8a   #增量编译，生成arm64-v8a架构Debug版本
./scripts/build_android.sh all release arm64-v8a    #全量编译，生成arm64-v8a架构Release版本
./scripts/build_android.sh incr release arm64-v8a    #增量编译，生成arm64-v8a架构Release版本
```

### Windows平台编译

此方式使用已经编译好的第三方库进行编译。

1. 下载alibabacloud-nls-cpp-sdk<version>-master_<github commit id>.zip并解压到本地，或从GitHub获取最新代码。
2. 进入scripts目录，使用文本编辑工具分别打开build_windows_64_prebuild.bat和build_windows_64_package.bat，修改解压缩工具WinRAR。

   例如，您的个人电脑WinRAR.exe所在路径为 `C:\Program Files (x86)\WinRAR\WinRAR.exe` ，则修改脚本文件中第三行为 `set winRar="C:\Program Files (x86)\WinRAR\WinRAR.exe"` 为您的个人电脑中WinRAR路径。
3. 双击批处理脚本build_windows_64_prebuild.bat，从而解压SDK源码中包含的各第三方库，并把依赖头文件释放到合适位置。
4. 使用Visual Studio（VS2015及以上版本）打开nlsCppSdk.sln，直接编译需要的范例工程。

   > **说明**
   > - 需要确认好各项目属性中的目标平台版本和平台工具集，按需选择。例如，您的目标平台版本为 `10.0.19041.0` ，平台工具集为 `Visual Studio 2015(v140)` 。
   > - 目前支持Debug_x64，Release_x64，Debug_win32和Release_win32。本文档说明均以x64为例。
5. 右键单击speechTranscriberDemo项目，单击生成进行编译。
6. 编译Debug_x64和Release_x64。
   - Debug生成物路径： `{ProjectRoot}\build\build_win64\nlsCppSdk\x64\Debug`
   - Release生成物路径： `{ProjectRoot}\build\build_win64\nlsCppSdk\x64\Release`

   各路径包含所有生成的dll和测试exe文件。其中 `{ProjectRoot}` 为SDK源码路径。
7. 若需要对生成的库文件、测试exe文件、头文件、说明文档等对外披露的文件进行打包，则在scripts目录下双击运行build_windows_64_package.bat，生成 `{ProjectRoot}\build\install\NlsSdk3.X_win64.zip` 。其中 `{ProjectRoot}` 为SDK源码路径。

## 关键接口

- 基础接口

- NlsClient：语音处理客户端，利用该客户端可以进行一句话识别、实时语音识别和语音合成的语音处理任务。该客户端为线程安全，建议全局仅创建一个实例。

| 接口名 | 功能描述 |
| --- | --- |
| getInstance | 获取（创建）NlsClient实例。 |
| setLogConfig | 设置日志文件与存储路径。 |
| startWorkThread | 启动工作线程数，默认1即启动一个线程，若-1则启动CPU核数的线程数。在高并发的情况下建议选择-1。 |
| releaseInstance | 销毁NlsClient对象实例。 |
| getVersion | 获取SDK版本号。 |
| createTranscriberRequest | 创建实时语音识别对象，线程安全，支持高并发请求。 |
| releaseTranscriberRequest | 销毁实时语音识别对象，需要在当前请求的closed事件后调用。 |

- NlsToken：创建Token对象，用于申请获取TokenId。申请新Token时需要先获取有效时间戳，若超过有效时间则再申请。若在有效时间内多次申请Token会导致TokenId错误而无法使用。

| 接口名 | 功能描述 |
| --- | --- |
| setAccessKeyId | 设置阿里云账号AccessKey ID。 |
| setKeySecret | 设置阿里云账号AccessKey Secret。 |
| applyNlsToken | 申请获取TokenId。 |
| getToken | 获取TokenId。 |
| getExpireTime | 获取Token有效期时间戳（秒）。 |

- NlsEvent：事件对象，您可以从中获取Request状态码、云端返回结果、失败信息等。

| 接口名 | 功能描述 |
| --- | --- |
| getStatusCode | 获取状态码，正常情况为0或者20000000，失败时对应失败的错误码。 |
| getErrorMessage | 在TaskFailed回调中，获取NlsRequest操作过程中出现失败时的错误信息。 |
| getTaskId | 获取任务的TaskId。 |
| getAllResponse | 获取云端返回的识别结果。 |
| getResult | 获取中间识别结果和最终结果。 |
| getSentenceIndex | 获取实时语音检测的句子编号。 |
| getSentenceTime | 获取实时语音检测的句子的音频时长，单位是毫秒。 |
| getSentenceBeginTime | 对应的SentenceBegin事件的时间，单位是毫秒。 |

- 识别接口

  SpeechTranscriberRequest：实时语音识别请求对象，用于长语音实时识别。

| 接口名 | 功能描述 |
| --- | --- |
| setOnTaskFailed | 设置错误回调函数。 |
| setOnTranscriptionStarted | 设置实时音频流识别开始回调函数。 |
| setOnSentenceBegin | 设置一句话开始回调。 |
| setOnSentenceEnd | 设置一句话结束回调函数。 |
| setOnTranscriptionResultChanged | 设置实时音频流识别中间结果回调函数。 |
| setOnTranscriptionCompleted | 设置服务端结束服务回调函数。 |

| 接口名 | 功能描述 |
| --- | --- |
| setOnChannelClosed | 设置通道关闭回调函数。 |
| setAppKey | 设置AppKey。 |
| setToken | 口令认证。所有的请求都必须通过SetToken方法认证通过，才可以使用。 |
| setUrl | 设置服务URL地址。 |
| setIntermediateResult | 设置是否返回中间识别结果。 |
| setPunctuationPrediction | 设置是否在后处理中添加标点。 |
| setInverseTextNormalization | 设置是否在后处理中执行数字转换。 |
| setFormat | 设置音频数据编码格式。 |
| setSampleRate | 音频采样率设置。 |
| setSemanticSentenceDetection | 设置是否使用语义断句。 |
| setMaxSentenceSilence | 设置vad阈值。 |
| setCustomizationId | 设置定制模型。 |
| setVocabularyId | 使用该接口也可以把项目和热词关联起来。如果同时配置了控制台热词和SDK，则SDK配置优先级最高，将会覆盖控制台热词配置。 |
| setTimeout | 设置Socket接收超时时间。 |
| setEnableNlp | 设置是否开启NLP服务。 |
| setNlpModel | 设置NLP模型名称，开启NLP服务后必填。 |

| 接口名 | 功能描述 |
| --- | --- |
| setSessionId | 设置Session ID。 |
| setOutputFormat | 设置输出文本的编码格式。 |
| setPayloadParam | 参数设置。 |
| setContextParam | 设置用户自定义参数。 |
| AppendHttpHeaderParam | 设置用户自定义ws阶段http header参数。 |
| start | 启动SpeechTranscriberRequest。 |
| stop | 会与服务端确认关闭，正常停止实时音频流识别操作。 |
| cancel | 不会与服务端确认关闭，直接关闭实时音频流识别过程。 |
| control | 要求服务端更新识别参数。 |
| sendAudio | 发送语音数据。 |

## C++ SDK错误码

| 错误码 | 错误描述 | 解决方案 |
| --- | --- | --- |
| 10000001 | `SSL: couldn’t create a ……!` | 建议重试。 |
| 10000002 | openssl官方错误描述 | 根据描述提示处理之后，重试。 |
| 10000003 | 系统错误描述 | 根据系统错误描述提示处理。 |
| 10000004 | `URL: The url is empty.` | 检查是否设置云端URL地址。 |

| 错误码 | 错误描述 | 解决方案 |
| --- | --- | --- |
| 10000005 | `URL: Could not parse WebSocket url.` | 检查是否正确设置云端URL地址。 |
| 10000006 | `MODE: unsupport mode.` | 检查是否正确设置了语音功能模式。 |
| 10000007 | `JSON: Json parse failed.` | 服务端发送错误响应内容，请提供task_id，并反馈至阿里云。 |
| 10000008 | `WEBSOCKET: unkown head type.` | 服务端发送错误WebSocket类型，请提供task_id，并通过工单系统反馈给我们。 |
| 10000009 | `HTTP: connect failed.` | 与云端连接失败，请检查网络后，重试。 |
| HTTP协议官方状态码 | `HTTP: Got bad status.` | 根据HTTP协议官方描述提示处理。 |
| 系统错误码 | `IP: ip address is not valid.` | 根据系统错误描述提示处理。 |
| 系统错误码 | `ENCODE: convert to utf8 error.` | 根据系统错误描述提示处理。 |
| 10000010 | `please check if the memory is enough.` | 内存不足，请检查本地机器内存。 |
| 10000011 | `Please check the order of execution.` | 接口调用顺序错误。接收到Failed/Complete事件时，SDK内部会关闭连接。此时再调用send会上报错误。 |
| 10000012 | `StartCommand/StopCommand Send failed.` | 参数错误。请检查参数设置是否正确。 |

| 错误码 | 错误描述 | 解决方案 |
| --- | --- | --- |
| 10000013 | `The sent data is null or dataSize <= 0.` | 发送错误。请检查发送参数是否正确。 |
| 10000014 | `Start invoke failed.` | 调用start方法超时错误。请调用stop方法释放资源，重新开始识别流程。 |
| 10000015 | `connect failed.` | 调用connect方法失败。请调用stop方法释放资源，重新开始识别流程。 |

## 服务端响应状态码

关于服务状态码，请参见接口说明。

## 代码示例

**说明**

- 示例中使用的音频文件为16000 Hz采样率，管控台设置的模型为**通用**模型。如果使用其他音频，请设置为支持该音频场景的模型。关于模型设置，请参见管理项目。
- 示例中使用了SDK内置的默认实时语音识别服务的外网访问服务端URL，如果您使用阿里云上海ECS且需要使用内网访问服务端URL，则在创建speechTranscriberRequest的对象中设置内网访问的URL。

  ```
  request->setUrl("ws://nls-gateway.cn-shanghai-internal.aliyuncs.com/ws/v1");
  ```

- 完整示例，参见SDK压缩包中Demo目录的speechTranscriberDemo.cpp文件。

```
#include <pthread.h>
#include <unistd.h>
#include <ctime>
#include <stdlib.h>
#include <string.h>
#include <string>
#include <vector>
#include <fstream>
#include "nlsClient.h"
#include "nlsEvent.h"
#include "speechTranscriberRequest.h"
#include "nlsToken.h"

#define FRAME_SIZE 3200
#define SAMPLE_RATE 16000

using namespace AlibabaNlsCommon;
```

```
using AlibabaNls::NlsClient;
using AlibabaNls::NlsEvent;
using AlibabaNls::LogDebug;
using AlibabaNls::LogInfo;
using AlibabaNls::SpeechTranscriberRequest;

// 自定义线程参数。
struct ParamStruct {
    std::string fileName;
    std::string token;
    std::string appkey;
};
// 自定义事件回调参数。
struct ParamCallBack {
    int userId;
    char userInfo[10];
};

//全局维护一个服务鉴权token和其对应的有效期时间戳，
//每次调用服务之前，首先判断token是否已经过期。
//如果已经过期，则根据AccessKey ID和AccessKey Secret重新生成一个token，并更新这个全局的token和其有效期时间戳。
//说明：只需在token即将过期时进行重新生成。所有的服务并发可共用一个token。
std::string g_akId = "";
std::string g_akSecret = "";
std::string g_token = "";
long g_expireTime = -1;
int generateToken(std::string akId, std::string akSecret, std::string* token, long* expireTime) {
    NlsToken nlsTokenRequest;
    nlsTokenRequest.setAccessKeyId(akId);
    nlsTokenRequest.setKeySecret(akSecret);
    if (-1 == nlsTokenRequest.applyNlsToken()) {
        // 获取失败原因。
        printf("generateToken Failed: %s\n", nlsTokenRequest.getErrorMsg());
        return -1;
    }
    *token = nlsTokenRequest.getToken();
    *expireTime = nlsTokenRequest.getExpireTime();
    return 0;
}

//@brief 获取sendAudio发送延时时间 //@param dataSize 待发送数据大小
//@param sampleRate 采样率：16k/8K
//@param compressRate 数据压缩率，例如压缩比为10:1的16k OPUS编码，此时为10，非压缩数据则为1。
//@return 返回sendAudio之后需要sleep的时间。
//@note 对于8k pcm 编码数据，16位采样，建议每发送1600字节 sleep 100 ms.
//              对于16k pcm 编码数据，16位采样，建议每发送3200字节 sleep 100 ms.
//              对于其它编码格式(OPUS)的数据，由于传递给SDK的仍然是PCM编码数据，
//              按照SDK OPUS/OPU 数据长度限制，需要每次发送640字节 sleep 20ms.
unsigned int getSendAudioSleepTime(int dataSize, int sampleRate, int compressRate) {
    // 仅支持16位采样
    const int sampleBytes = 16;      // 仅支持单通道
    const int soundChannel = 1;      // 当前采样率，采样位数下每秒采样数据的大小。
```

```
    int bytes = (sampleRate * sampleBytes * soundChannel) / 8;     // 当前采样率，采样位数下每
毫秒采样数据的大小。
    int bytesMs = bytes / 1000;     // 待发送数据大小除以每毫秒采样数据大小，以获取sleep时间。
    int sleepMs = (dataSize * compressRate) / bytesMs;
    return sleepMs;
}

//@brief 调用start()，成功与云端建立连接，SDK内部线程上报started事件。
//@param cbEvent 回调事件结构，详见nlsEvent.h。
//@param cbParam 回调自定义参数，默认为NULL。可以根据需求自定义参数。
void onTranscriptionStarted(NlsEvent* cbEvent, void* cbParam) {
    ParamCallBack* tmpParam = (ParamCallBack*)cbParam;
    // 演示如何打印/使用用户自定义参数示例。
    printf("onTranscriptionStarted: %d\n", tmpParam->userId);
    // 当前任务的task id，方便定位问题。
    printf("onTranscriptionStarted: status code=%d, task id=%s\n",
            cbEvent->getStatusCode(), cbEvent->getTaskId());

    // 获取服务端返回的全部信息
    //printf("onTranscriptionStarted: all response=%s\n", cbEvent->getAllResponse());
}

//@brief 服务端检测到了一句话的开始，SDK内部线程上报SentenceBegin事件。
//@param cbEvent 回调事件结构，详情参见nlsEvent.h。
//@param cbParam 回调自定义参数，默认为NULL，可以根据需求自定义参数。
void onSentenceBegin(NlsEvent* cbEvent, void* cbParam) {
    ParamCallBack* tmpParam = (ParamCallBack*)cbParam;
    // 演示如何打印/使用用户自定义参数示例。
    printf("onSentenceBegin: %d\n", tmpParam->userId);
    printf("onSentenceBegin: status code=%d, task id=%s, index=%d, time=%d\n",
            cbEvent->getStatusCode(), cbEvent->getTaskId(),
            cbEvent->getSentenceIndex(), //句子编号，从1开始递增。
            cbEvent->getSentenceTime() //当前已处理的音频时长，单位：毫秒。
    );

    // 获取服务端返回的全部信息
    //printf("onTranscriptionStarted: all response=%s\n", cbEvent->getAllResponse());
}

//@brief 服务端检测到了一句话结束，SDK内部线程上报SentenceEnd事件。
//@param cbEvent 回调事件结构，详情参见nlsEvent.h。
//@param cbParam 回调自定义参数，默认为NULL，可以根据需求自定义参数。
void onSentenceEnd(NlsEvent* cbEvent, void* cbParam) {
    ParamCallBack* tmpParam = (ParamCallBack*)cbParam;
    // 演示如何打印
    //使用用户自定义参数示例。
    printf("onSentenceEnd: %d\n", tmpParam->userId);
    printf("onSentenceEnd: status code=%d, task id=%s, index=%d, time=%d, begin_time=%d, re
sult=%s\n",
            cbEvent->getStatusCode(),
            cbEvent->getTaskId(),
            cbEvent->getSentenceIndex(), //句子编号，从1开始递增。
            cbEvent->getSentenceTime(), //当前已处理的音频时长，单位：毫秒。
            cbEvent->getSentenceBeginTime(), // 对应的SentenceBegin事件的时间。
            cbEvent->getResult()     // 当前句子的完整识别结果
```

```
            cbEvent->getResult()     // 当前句子的完整识别结果。
    );
    // 获取服务端返回的全部信息
    //printf("onTranscriptionStarted: all response=%s\n", cbEvent->getAllResponse());
}

//@brief 识别结果发生了变化，SDK在接收到云端返回的最新结果时，其内部线程上报ResultChanged事件。
//@param cbEvent 回调事件结构，详情参见nlsEvent.h。
//@param cbParam 回调自定义参数，默认为NULL，可以根据需求自定义参数。
void onTranscriptionResultChanged(NlsEvent* cbEvent, void* cbParam) {
    ParamCallBack* tmpParam = (ParamCallBack*)cbParam;
    // 演示如何打印/使用用户自定义参数示例。
    printf("onTranscriptionResultChanged: %d\n", tmpParam->userId);
    printf("onTranscriptionResultChanged: status code=%d, task id=%s, index=%d, time=%d, re
sult=%s\n",
            cbEvent->getStatusCode(),
            cbEvent->getTaskId(),
            cbEvent->getSentenceIndex(), //句子编号，从1开始递增。
            cbEvent->getSentenceTime(), //当前已处理的音频时长，单位：毫秒。
            cbEvent->getResult()     // 当前句子的完整识别结果
    );
    // 获取服务端返回的全部信息
    //printf("onTranscriptionStarted: all response=%s\n", cbEvent->getAllResponse());
}

//@brief 服务端停止实时音频流识别时，SDK内部线程上报Completed事件。
//@note 上报Completed事件之后，SDK内部会关闭识别连接通道。此时调用sendAudio会返回-1，请停止发送。
//@param cbEvent 回调事件结构，详情参见nlsEvent.h。
//@param cbParam 回调自定义参数，默认为NULL，可以根据需求自定义参数。
void onTranscriptionCompleted(NlsEvent* cbEvent, void* cbParam) {
    ParamCallBack* tmpParam = (ParamCallBack*)cbParam;
    // 演示如何打印/使用用户自定义参数示例。
    printf("onTranscriptionCompleted: %d\n", tmpParam->userId);
    printf("onTranscriptionCompleted: status code=%d, task id=%s\n",
            cbEvent->getStatusCode(),
            cbEvent->getTaskId());
}

//@brief 识别过程（包含start()、send()、stop()）发生异常时，SDK内部线程上报TaskFailed事件。
//@note 上报TaskFailed事件之后，SDK内部会关闭识别连接通道。此时调用sendAudio会返回-1，请停止发送。
//@param cbEvent 回调事件结构，详情参见nlsEvent.h。
//@param cbParam 回调自定义参数，默认为NULL，可以根据需求自定义参数。
void onTaskFailed(NlsEvent* cbEvent, void* cbParam) {
    ParamCallBack* tmpParam = (ParamCallBack*)cbParam;
    // 演示如何打印/使用用户自定义参数示例。
    printf("onTaskFailed: %d\n", tmpParam->userId);
    printf("onTaskFailed: status code=%d, task id=%s, error message=%s\n",
            cbEvent->getStatusCode(),
            cbEvent->getTaskId(),
            cbEvent->getErrorMessage()
    );
    // 获取服务端返回的全部信息
    //printf("onTaskFailed: all response=%s\n", cbEvent->getAllResponse());
}
```

```
//@brief SDK内部线程上报语音表单结果事件
//@param cbEvent 回调事件结构，详情参见nlsEvent.h。
//@param cbParam 回调自定义参数，默认为NULL，可以根据需求自定义参数。
void onSentenceSemantics(NlsEvent* cbEvent, void* cbParam) {
    ParamCallBack* tmpParam = (ParamCallBack*)cbParam;
    // 演示如何打印/使用用户自定义参数示例。
    printf("onSentenceSemantics: %d\n", tmpParam->userId);
    // 获取服务端返回的全部信息。
    printf("onSentenceSemantics: all response=%s\n", cbEvent->getAllResponse());
}

//@brief 识别结束或发生异常时，会关闭连接通道，SDK内部线程上报ChannelCloseed事件。
//@param cbEvent 回调事件结构，详情参见nlsEvent.h。
//@param cbParam 回调自定义参数，默认为NULL，可以根据需求自定义参数。
void onChannelClosed(NlsEvent* cbEvent, void* cbParam) {
    ParamCallBack* tmpParam = (ParamCallBack*)cbParam;
    delete tmpParam; //识别流程结束，释放回调参数。
}

// 工作线程
void* pthreadFunc(void* arg) {
    int sleepMs = 0;
    ParamCallBack *cbParam = NULL;
    //初始化自定义回调参数，以下两变量仅作为示例表示参数传递，在demo中不起任何作用。
    //回调参数在堆中分配之后，SDK在销毁request对象时会一并销毁。
    cbParam = new ParamCallBack;
    cbParam->userId = 1234;
    strcpy(cbParam->userInfo, "User.");
    // 0: 从自定义线程参数中获取token，配置文件等参数。
    ParamStruct* tst = (ParamStruct*)arg;
    if (tst == NULL) {
        printf("arg is not valid\n");
        return NULL;
    }
    /* 打开音频文件，获取数据 */
    std::ifstream fs;
    fs.open(tst->fileName.c_str(), std::ios::binary | std::ios::in);
    if (!fs) {
        printf("%s isn't exist..\n", tst->fileName.c_str());
        return NULL;
    }
    //2: 创建实时音频流识别SpeechTranscriberRequest对象。
    SpeechTranscriberRequest* request = NlsClient::getInstance()->createTranscriberRequest(
);
    if (request == NULL) {
        printf("createTranscriberRequest failed.\n");
        return NULL;
    }
    request->setOnTranscriptionStarted(onTranscriptionStarted, cbParam);                //
设置识别启动回调函数
    request->setOnTranscriptionResultChanged(onTranscriptionResultChanged, cbParam);    //
设置识别结果变化回调函数
    request->setOnTranscriptionCompleted(onTranscriptionCompleted, cbParam);            //
设置语音转写结束回调函数
```

```
    request->setOnSentenceBegin(onSentenceBegin, cbParam);                          //
设置一句话开始回调函数
    request->setOnSentenceEnd(onSentenceEnd, cbParam);                              //
设置一句话结束回调函数
    request->setOnTaskFailed(onTaskFailed, cbParam);                                //
设置异常识别回调函数
    request->setOnChannelClosed(onChannelClosed, cbParam);                          //
设置识别通道关闭回调函数
    request->setAppKey(tst->appkey.c_str());            // 设置appkey，必选参数。
    request->setFormat("pcm");                          // 设置音频数据编码格式，默认值PCM。

    request->setSampleRate(SAMPLE_RATE);                // 设置音频数据采样率，可选参数，目前支
持16000/8000，默认值16000。
    request->setIntermediateResult(true);               // 设置是否返回中间识别结果，可选参数，
默认false。
    request->setPunctuationPrediction(true);            // 设置是否在后处理中添加标点，可选参数
，默认false。
    request->setInverseTextNormalization(true);         // 设置是否在后处理中执行数字转写，可选
参数，默false。
    //语音断句检测阈值，一句话之后静音长度超过该值，即本句结束，合法参数范围200ms～2000ms，默认值800m
s。
    //request->setMaxSentenceSilence(800);
    //request->setCustomizationId("TestId_123"); //定制模型id，可选。
    //request->setVocabularyId("TestId_456"); //定制泛热词id，可选。
    // 用于传递某些定制化、高级参数设置，参数格式为JSON格式： {"key": "value"}
    //request->setPayloadParam("{\"vad_model\": \"farfield\"}");
    //设置是否开启词模式。
    request->setPayloadParam("{\"enable_words\": true}");
    //语义断句，默认false，非必需则不建议设置。
    //request->setPayloadParam("{\"enable_semantic_sentence_detection\": false}");
    //是否开启顺滑，默认不开启，非必需则不建议设置。
    //request->setPayloadParam("{\"disfluency\": true}");
    //设置vad的模型，默认不设置，非必需则不建议设置。
    //request->setPayloadParam("{\"vad_model\": \"farfield\"}");
    //设置是否忽略单句超时
    //request->setPayloadParam("{\"enable_ignore_sentence_timeout\": false}");
    //vad断句开启后处理，默认不设置，非必需则不建议设置。
    //request->setPayloadParam("{\"enable_vad_unify_post\": true}");
    request->setToken(tst->token.c_str());
    //3: start()为异步操作。成功返回started事件，失败返回TaskFailed事件。
    if (request->start() < 0) {
        printf("start() failed. may be can not connect server. please check network or fire
walld\n");
        NlsClient::getInstance()->releaseTranscriberRequest(request); // start()失败，释放re
quest对象。
        return NULL;
    }
    while (!fs.eof()) {
        uint8_t data[FRAME_SIZE] = {0};
        fs.read((char *)data, sizeof(uint8_t) * FRAME_SIZE);
        size_t nlen = fs.gcount();
        if (nlen <= 0) {
            continue;
        }
```

```
        //4: 发送音频数据。sendAudio返回-1表示发送失败，需要停止发送。
        int ret = request->sendAudio(data, nlen);
        if (ret < 0) {
            // 发送失败，退出循环数据发送。
            printf("send data fail.\n");
            break;
        }
         //语音数据发送控制:
        //语音数据是实时的，不需要sleep控制速率。
        //语音数据来自文件，发送时需要控制速率，使单位时间内发送的数据大小接近单位时间原始语音数据存储的大小。
        sleepMs = getSendAudioSleepTime(nlen, SAMPLE_RATE, 1);  // 根据发送数据大小、采样率、数据压缩比，获取sleep时间。
        //5: 语音数据发送延时控制
        usleep(sleepMs * 1000);
    }
    // 关闭音频文件
    fs.close();
    //6: 通知云端数据发送结束
    //stop()为异步操作，失败返回TaskFailed事件。
    request->stop();
    //7: 识别结束，释放request对象。
    NlsClient::getInstance()->releaseTranscriberRequest(request);
    return NULL;
}

//识别单个音频数据
int speechTranscriberFile(const char* appkey) {
    // 获取当前系统时间戳，判断token是否过期。
    std::time_t curTime = std::time(0);
    if (g_expireTime - curTime < 10) {
        printf("the token will be expired, please generate new token by AccessKey-ID and AccessKey-Secret.\n");
        if (-1 == generateToken(g_akId, g_akSecret, &g_token, &g_expireTime)) {

            return -1;
        }
    }
    ParamStruct pa;
    pa.token = g_token;
    pa.appkey = appkey;
    pa.fileName = "test0.wav";
    pthread_t pthreadId;     // 启动一个工作线程，用于识别。
    pthread_create(&pthreadId, NULL, &pthreadFunc, (void *)&pa);
    pthread_join(pthreadId, NULL);
    return 0;
}

//识别多个音频数据
//SDK多线程指一个音频数据对应一个线程，多个音频数据对应多个线程。
//示例代码为同时开启2个线程识别2个文件。
//免费用户并发连接不能超过2个。
#define AUDIO_FILE_NUMS 2
#define AUDIO_FILE_NAME_LENGTH 32
```

```
int speechTranscriberMultFile(const char* appkey) {
    // 获取当前系统时间戳，判断token是否过期。
    std::time_t curTime = std::time(0);
    if (g_expireTime - curTime < 10) {
        printf("the token will be expired, please generate new token by AccessKey-ID and Ac
cessKey-Secret.\n");
        if (-1 == generateToken(g_akId, g_akSecret, &g_token, &g_expireTime)) {

            return -1;
        }
    }
    char audioFileNames[AUDIO_FILE_NUMS][AUDIO_FILE_NAME_LENGTH] =
    {
        "test0.wav",
        "test1.wav"
    };
    ParamStruct pa[AUDIO_FILE_NUMS];
    for (int i = 0; i < AUDIO_FILE_NUMS; i ++) {
        pa[i].token = g_token;
        pa[i].appkey = appkey;
        pa[i].fileName = audioFileNames[i];
    }
    std::vector<pthread_t> pthreadId(AUDIO_FILE_NUMS);
    // 启动2个工作线程，同时识别2个音频文件。
    for (int j = 0; j < AUDIO_FILE_NUMS; j++) {
        pthread_create(&pthreadId[j], NULL, &pthreadFunc, (void *)&(pa[j]));
    }
    for (int j = 0; j < AUDIO_FILE_NUMS; j++) {
        pthread_join(pthreadId[j], NULL);
    }
    return 0;
}

int main(int arc, char* argv[]) {
    if (arc < 4) {
        printf("params is not valid. Usage: ./demo <your appkey> <your AccessKey ID> <your
AccessKey Secret>\n");
        return -1;
    }
    std::string appkey = argv[1];
    g_akId = argv[2];
    g_akSecret = argv[3];
    // 根据需要设置SDK输出日志，可选。此处表示SDK日志输出至log-Transcriber.txt，LogDebug表示输出所
有级别日志。
    int ret = NlsClient::getInstance()->setLogConfig("log-transcriber", LogDebug);
    if (-1 == ret) {
        printf("set log failed\n");
        return -1;
    }
    //启动工作线程
    NlsClient::getInstance()->startWorkThread(4);      // 识别单个音频数据
    speechTranscriberFile(appkey.c_str());      // 识别多个音频数据
    // speechTranscriberMultFile(appkey.c_str());
    // 所有工作完成，进程退出前释放nlsClient，releaseInstance()非线程安全。
    NlsClient::releaseInstance();
```

```
    NlsClient::releaseInstance();
    return 0;
}
```

# 5.C# SDK

本文介绍如何使用阿里云智能语音服务提供的C# SDK，包括SDK的安装方法及SDK代码示例。

## SDK下载

> 说明
> - 当前最新版本：3.1.13，该版本基于C++ SDK API 3.1.x。发布日期：2022年4月18日。
> - 使用SDK前，请先阅读接口说明，详情请参见接口说明。

可通过以下两种方法获取SDK。

- 方法一：从GitHub获取最新源码，详细编译和运行方式可见下文，或查看源码中的readme.md。

```
git clone --depth 1 https://github.com/aliyun/alibabacloud-nls-cpp-sdk
```

- 方法二：直接从下文表中选取需要SDK包进行下载。其中SDK源码包为SDK原始代码，需要通过下文编译方法生成集成所需的库文件。其他对应平台的SDK包内含相关库文件、头文件，无需编译。

| 最新SDK包 | 平台 | MD5 |
| --- | --- | --- |
| alibabacloud-nls-cpp-sdk3.1.13-master_d1b6694.zip | SDK源码 | f9f0b479c7008abcc20089e38df8a4ae |
| NlsCsharpSdk_Windows_3.1.13_d1b6694.zip | Windows | 0953ff23142b671464d8cfe9fbdf2cf4 |

其中：

- alibabacloud-nls-cpp-sdk-master_<github commit id>.zip为SDK源码包。
- NlsCsharpSdk_Windows_<版本号>_<github commit id>.zip为Windows平台下开发需要的SDK包，详见内部readme.md。

## Windows平台C#编译运行

### 编译第三方库

此方式使用已经编译好的第三方库进行编译，首先编译Windows平台Cpp SDK库。

1. 下载alibabacloud-nls-cpp-sdk<version>-master_<github commit id>.zip并解压到本地，或从GitHub获取最新代码。
2. 进入scripts目录，使用文本编辑工具分别打开build_windows_64_prebuild.bat和build_windows_64_package.bat，修改解压缩工具WinRAR。

例如，您的个人电脑WinRAR.exe所在路径为 `C:\Program Files (x86)\WinRAR\WinRAR.exe` ，则修改脚本文件中第三行为

```
set winRar="C:\Program Files (x86)\WinRAR\WinRAR.exe"
```

为您的个人电脑中WinRAR路径。

3. 双击批处理脚本**build_windows_64_prebuild.bat**，从而解压SDK源码中包含的各第三方库，并把依赖头文件释放到合适位置。
4. 使用Visual Studio（VS2015及以上版本）打开**nlsCppSdk.sln**，直接编译需要的范例工程。

> **说明**
>
> - 需要确认好各项目**属性**中的**目标平台版本**和**平台工具集**，按需选择。例如，您的**目标平台版本**为 `10.0.19041.0` ，**平台工具集**为 `Visual Studio 2015(v140)` 。
> - 目前支持Debug_x64，Release_x64，Debug_win32和Release_win32。本文档说明均以x64为例。

5. 右键单击**speechTranscriberDemo**项目，单击**生成**进行编译。
6. 编译Debug_x64和Release_x64。
   - Debug生成物路径： `{ProjectRoot}\build\build_win64\nlsCppSdk\x64\Debug`
   - Release生成物路径： `{ProjectRoot}\build\build_win64\nlsCppSdk\x64\Release`

   各路径包含所有生成的dll和测试exe文件。其中 `{ProjectRoot}` 为SDK源码路径。
7. 若需要对生成的库文件、测试exe文件、头文件、说明文档等对外披露的文件进行打包，则在**scripts**目录下双击运行**build_windows_64_package.bat**，生成 `{ProjectRoot}\build\install\NlsSdk3.X_win64.zip` 。其中 `{ProjectRoot}` 为SDK源码路径。

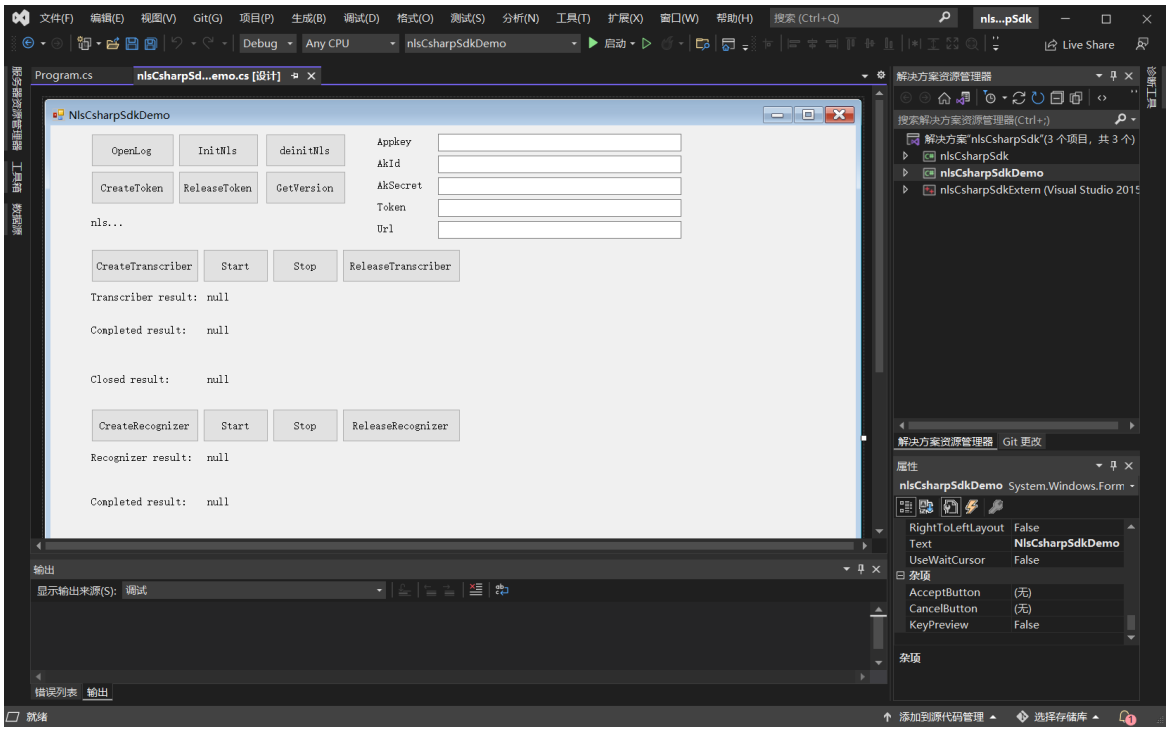
## C#编译

必须先完成Windows平台Cpp SDK编译，C#依赖Windows Cpp SDK的库和头文件。完成以上编译后开始进行C#编译。

1. 使用Visual Studio（VS2015及以上版本）打开**nlsCsharpSdk.sln**，进行C#工程编译。

> **说明**
>
> 需要确认**nlsCsharpSdkExtern**的**平台工具集**和**目标平台版本**，以及**nlsCsharpSdkDemo**和**nlsCsharpSdk**的.NET版本。

2. 右键nlsCsharpSdkDemo项目，选择**生成**，生成所有生成物。




> **说明**
>
> `{ProjectRoot}\nlsCsharpSdk\nlsCsharpSdkDemo\bin\Debug` 路径下包含所有生成物，`{ProjectRoot}` 为SDK源码路径。
>
> - **nlsCsharpSdk.dll**为NLS C# SDK，可进行发布Nupkg。
> - **nlsCsharpSdkExtern.dll**为NLS C#与Cpp互操作层。
> - **nlsCsharpSdkDemo.exe**为NLS C# UI Demo。

3. 以Debug版本为例，运行还需要一些NLS Cpp SDK相关依赖库，需要将依赖库和测试音频文件搬移到**nlsCsharpSdkDemo.exe**能依赖的路径：

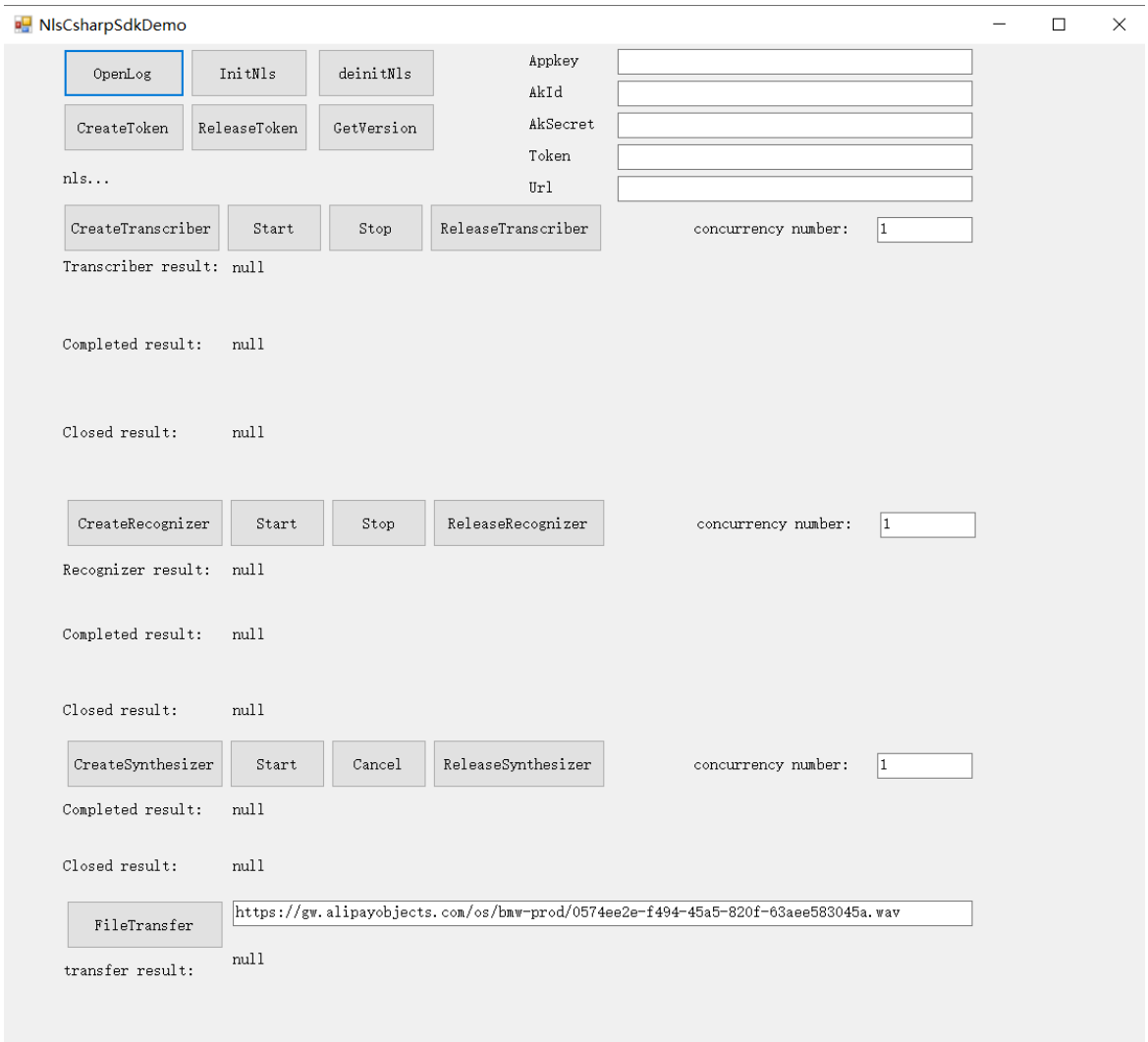
`{ProjectRoot}\nlsCsharpSdk\nlsCsharpSdkDemo\bin\Debug` 。即分别将下方动态库拷贝至 `{ProjectRoot}\nlsCsharpSdk\nlsCsharpSdkDemo\bin\Debug` 目录下。

> **说明**
>
> 正常情况下在工程编译完成后系统会进行自动搬运，您可跳过此步骤。若目标路径无以下文件，则需要进行手动搬运。

```
{ProjectRoot}\build\install\NlsSdk3.X_win64\lib\14.0\x64\Debug\nlsCppSdk.dll
{ProjectRoot}\build\install\NlsSdk3.X_win64\lib\14.0\x64\Debug\libcrypto-1_1-x64.dll
{ProjectRoot}\build\install\NlsSdk3.X_win64\lib\14.0\x64\Debug\libssl-1_1-x64.dll
{ProjectRoot}\build\install\NlsSdk3.X_win64\lib\14.0\x64\Debug\libcurld.dll (release版为libcurl.dll)
{ProjectRoot}\build\install\NlsSdk3.X_win64\lib\14.0\x64\Debug\libeay32.dll
{ProjectRoot}\build\install\NlsSdk3.X_win64\lib\14.0\x64\Debug\ssleay32.dll
{ProjectRoot}\build\install\NlsSdk3.X_win64\lib\14.0\x64\Debug\pthreadVC2.dll
```

`{ProjectRoot}\nlsCsharpSdk\nlsCsharpSdkDemo\bin\Debug\audio_files\` 目录下，需要包含命名为test0.wav、test1.wav、test2.wav、test3.wav的四个音频文件，用于进行语音识别。这些WAV文件可从 `resource/audio` 目录拷贝。若缺少这些文件，系统将会在运行语音识别过程中，因找不到文件而退出。

4. 双击**nlsCsharpSdkDemo.exe**，运行如下：




5. （可选）单击**OpenLog**可开启日志记录。
6. 单击**Init Nls**初始化SDK。
7. 填入Appkey、AccessKey ID、AccessKey Secret后，单击**CreateToken**生成Token，合法Token会在Token栏显示。

8. 单击CreateTranscriber创建实时语音识别请求，然后单击Start开始工作，会在最下方实时显示识别结果。单击Stop进行停止，单击ReleaseTranscriber释放请求。

## 关键接口

```
using nlsCsharpSdk;
```

- 基础接口
  - NlsClient：语音处理客户端，利用该客户端可以进行一句话识别、实时语音识别和语音合成的语音处理任务。该客户端为线程安全，建议全局仅创建一个实例。

| 接口名 | 功能描述 |
| --- | --- |
| SetLogConfig | 设置日志文件与存储路径。 |
| StartWorkThread | 启动工作线程数，默认1即启动一个线程，若-1则启动CPU核数的线程数。在高并发的情况下建议选择-1。这一接口有初始化NlsClient的作用。 |
| ReleaseInstance | 销毁NlsClient对象实例，即释放NLS SDK。 |
| GetVersion | 获取SDK版本号。 |
| CreateTranscriberRequest | 创建实时语音识别对象，线程安全，支持高并发请求。 |
| ReleaseTranscriberRequest | 销毁实时语音识别对象，需要在当前请求的closed事件后调用。 |

  - NlsToken：创建Token对象，用于申请获取TokenId。申请新Token时需要先获取有效时间戳，若超过有效时间则再申请。若在有效时间内多次申请Token会导致TokenId错误而无法使用。

| 接口名 | 功能描述 |
| --- | --- |
| SetAccessKeyId | 设置阿里云账号AccessKey ID。 |
| SetKeySecret | 设置阿里云账号AccessKey Secret。 |
| ApplyNlsToken | 申请获取TokenId。 |
| GetToken | 获取TokenId。 |
| GetExpireTime | 获取Token有效期时间戳（秒）。 |

- NlsEvent：事件对象，您可以从中获取Request状态码、云端返回结果、失败信息等。

| 接口名 | 功能描述 |
| --- | --- |
| statusCode | 获取状态码，正常情况为0或者20000000，失败时对应失败的错误码。 |
| taskId | 获取任务的TaskId。 |
| msg | 获取云端返回的response结果。 |
| result | 获取云端返回的识别结果。 |
| sentenceIndex | 获取实时语音检测的句子编号。 |
| sentenceTime | 获取实时语音检测的句子的音频时长，单位是毫秒。 |
| sentenceBegineTime | 对应的SentenceBegin事件的时间，单位是毫秒。 |

- 识别接口

  SpeechTranscriberRequest：实时语音识别请求对象，用于长语音实时识别。

| 接口名 | 功能描述 |
| --- | --- |
| SetOnTaskFailed | 设置错误回调函数。 |
| SetOnTranscriptionStarted | 设置实时音频流识别开始回调函数。 |
| SetOnSentenceBegin | 设置一句话开始回调。 |
| SetOnSentenceEnd | 设置一句话结束回调函数。 |
| SetOnTranscriptionResultChanged | 设置实时音频流识别中间结果回调函数。 |
| SetOnTranscriptionCompleted | 设置服务端结束服务回调函数。 |
| SetOnChannelClosed | 设置通道关闭回调函数。 |

| 接口名 | 功能描述 |
| --- | --- |
| SetAppKey | 设置Appkey |
| SetToken | 口令认证。所有的请求都必须通过SetToken方法认证通过，才可以使用。 |
| SetUrl | 设置服务URL地址。 |
| SetIntermediateResult | 设置是否返回中间识别结果。 |
| SetPunctuationPrediction | 设置是否在后处理中添加标点。 |
| SetInverseTextNormalization | 设置是否在后处理中执行数字转换。 |
| SetFormat | 设置音频数据编码格式。 |
| SetSampleRate | 音频采样率设置。 |
| SetSemanticSentenceDetection | 设置是否使用语义断句。 |
| SetMaxSentenceSilence | 设置vad阈值。 |
| SetCustomizationId | 设置定制模型。 |
| SetVocabularyId | 使用该接口也可以把项目和热词关联起来。如果同时配置了控制台热词和SDK，则SDK配置优先级最高，将会覆盖控制台热词配置。 |
| SetTimeout | 设置Socket接收超时时间。 |
| SetEnableNlp | 设置是否开启NLP服务。 |
| SetNlpModel | 设置NLP模型名称，开启NLP服务后必填。 |
| SetSessionId | 设置Session ID。 |
| SetOutputFormat | 设置输出文本的编码格式。 |

| 接口名 | 功能描述 |
| --- | --- |
| SetPayloadParam | 参数设置。 |
| SetContextParam | 设置用户自定义参数。 |
| AppendHttpHeaderParam | 设置用户自定义ws阶段http header参数。 |
| Start | 启动SpeechTranscriberRequest。 |
| Stop | 会与服务端确认关闭，正常停止实时音频流识别操作。 |
| Cancel | 不会与服务端确认关闭，直接关闭实时音频流识别过程。 |
| Control | 要求服务端更新识别参数。 |
| SendAudio | 发送语音数据。 |

## 错误码

| 错误码 | 错误描述 | 解决方案 |
| --- | --- | --- |
| 10000001 | `SSL: couldn’t create a …!` | 建议重试。 |
| 10000002 | openssl官方错误描述 | 根据描述提示处理后，重试。 |
| 10000003 | 系统错误描述 | 根据系统错误描述提示处理。 |
| 10000004 | `URL: The url is empty.` | 检查是否设置云端URL地址。 |
| 10000005 | `URL: Could not parse WebSocket url.` | 检查是否正确设置云端URL地址。 |

| 错误码 | 错误描述 | 解决方案 |
| --- | --- | --- |
| 10000006 | `MODE: unsupport mode.` | 检查是否正确设置了语音功能模式。 |
| 10000007 | `JSON: Json parse failed.` | 服务端发送错误响应内容，请提供task_id，并反馈给阿里云。 |
| 10000008 | `WEBSOCKET: unkown head type.` | 服务端发送错误WebSocket类型，请提供task_id，并通过工单系统反馈给我们。 |
| 10000009 | `HTTP: connect failed.` | 与云端连接失败，请检查网络后，重试。 |
| HTTP协议官方状态码 | `HTTP: Got bad status.` | 根据HTTP协议官方描述提示处理。 |
| 系统错误码 | `IP: ip address is not valid.` | 根据系统错误描述提示处理。 |
| 系统错误码 | `ENCODE: convert to utf8 error.` | 根据系统错误描述提示处理。 |
| 10000010 | `please check if the memory is enough.` | 内存不足，请检查本地机器内存。 |
| 10000011 | `Please check the order of execution.` | 接口调用顺序错误（接收到Failed/complete事件时，SDK内部会关闭连接。此时再调用send方法会上报错误）。 |
| 10000012 | `StartCommand/StopCommand Send failed.` | 参数错误。请检查参数设置是否正确。 |

| 错误码 | 错误描述 | 解决方案 |
| --- | --- | --- |
| 10000013 | `The sent data is null or dataSize <= 0.` | 发送错误。请检查发送参数是否正确。 |
| 10000014 | `Start invoke failed.` | 调用start方法超时错误。请调用stop方法释放资源，重新开始识别流程。 |
| 10000015 | `connect failed.` | 调用connect方法失败。请调用stop方法释放资源，重新开始识别流程。 |

## 服务端响应状态码

关于服务状态码，请参见服务状态码。

## 代码示例

**说明**

- 示例中使用的音频文件为16000 Hz采样率，管控台设置的模型为**通用**模型。如果使用其他音频，请设置为支持该音频场景的模型。关于模型设置，请参见管理项目。
- 示例中使用了SDK内置的默认实时语音识别服务的外网访问服务URL，如果您使用阿里云上海ECS且需要使用内网访问URL，则在创建SpeechTranscriberRequest的对象中设置内网访问的URL。

```
SpeechTranscriberRequest srPtr;
srPtr.SetUrl(srPtr, "ws://nls-gateway.cn-shanghai-internal.aliyuncs.com/ws/v1")
```

- 以下为简要示例，完整示例请参见SDK压缩包中**demo**目录的**nlsCsharpSdkDemo.cs**文件。

```
using System;
using System.IO;
using System.Threading;
using System.Windows.Forms;
using nlsCsharpSdk;

namespace nlsCsharpSdkDemo
{
    public partial class nlsCsharpSdkDemo : Form
    {
        private NlsClient nlsClient;
        private SpeechTranscriberRequest stPtr;
        private NlsToken tokenPtr;
        private UInt64 expireTime;

        private string appKey;
```

```
        private string akId;
        private string akSecret;
        private string token;
        private string url;

        static bool running;
        static string cur_nls_result;

        static bool st_send_audio_flag = false;
        static bool st_audio_loop_flag = false;
        static Thread st_send_audio;
        static string cur_st_result;
        static string cur_st_completed;
        static string cur_st_closed;

        private void FlushLab()
        {
            while (running)
            {
                if (cur_nls_result != null && cur_nls_result.Length > 0)
                {
                    nlsResult.Text = cur_nls_result;
                }

                if (cur_st_result != null && cur_st_result.Length > 0)
                {
                    stResult.Text = cur_st_result;
                }
                if (cur_st_completed != null && cur_st_completed.Length > 0)
                {
                    stCompleted.Text = cur_st_completed;
                }
                if (cur_st_closed != null && cur_st_closed.Length > 0)
                {
                    stClosed.Text = cur_st_closed;
                }

                Thread.Sleep(200);
            }
        }
        private void STAudioLab()
        {
            string file_name = System.Environment.CurrentDirectory + @"\audio_files\test3.w
av";
            System.Diagnostics.Debug.WriteLine("st audio file_name = {0}", file_name);
            FileStream fs = new FileStream(file_name, FileMode.Open, FileAccess.Read);
            BinaryReader br = new BinaryReader(fs);

            while (st_audio_loop_flag)
            {
                if (st_send_audio_flag)
                {
                    byte[] byData = br.ReadBytes((int)640);
                    if (byData.Length > 0)
```

```
                    {
                        stPtr.SendAudio(stPtr, byData, (UInt64)byData.Length, EncoderType.E
NCODER_PCM);
                    }
                    else
                    {
                        br.Close();
                        fs.Dispose();
                        fs = new FileStream(file_name, FileMode.Open, FileAccess.Read);
                        br = new BinaryReader(fs);
                    }
                }
                Thread.Sleep(20);
            }
            br.Close();
            fs.Dispose();
        }

        public nlsCsharpSdkDemo()
        {
            InitializeComponent();
            System.Windows.Forms.Control.CheckForIllegalCrossThreadCalls = false;//设置该属
性 为false

            nlsClient = new NlsClient();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            int ret = nlsClient.SetLogConfig("nlsLog", LogLevel.LogDebug, 400, 10);
            if (ret == 0)
                nlsResult.Text = "OpenLog Success";
            else
                nlsResult.Text = "OpenLog Failed";
        }

        private void button1_Click_1(object sender, EventArgs e)
        {
            string version = nlsClient.GetVersion();
            nlsResult.Text = version;
        }

        private void button1_Click_2(object sender, EventArgs e)
        {
            nlsClient.StartWorkThread(-1);
            nlsResult.Text = "StartWorkThread and init NLS success.";
            running = true;
            Thread t = new Thread(FlushLab);
            t.Start();
        }

        private void button1_Click_3(object sender, EventArgs e)
        {
            nlsClient.ReleaseInstance();
```

```
            nlsResult.Text = "Release NLS success.";
        }

        #region TranscriberButton
        // create transcriber
        private void button1_Click_4(object sender, EventArgs e)
        {
            stPtr = nlsClient.CreateTranscriberRequest();
            if (stPtr.native_request != IntPtr.Zero)
            {
                nlsResult.Text = "CreateTranscriberRequest Success";
            }
            else
            {
                nlsResult.Text = "CreateTranscriberRequest Failed";
            }
            cur_st_result = "null";
            cur_st_closed = "null";
            cur_st_completed = "null";
        }

        // release transcriber
        private void button2_Click(object sender, EventArgs e)
        {
            if (stPtr.native_request != IntPtr.Zero)
            {
                nlsClient.ReleaseTranscriberRequest(stPtr);
                stPtr.native_request = IntPtr.Zero;
                nlsResult.Text = "ReleaseTranscriberRequest Success";
            }
            else
            {
                nlsResult.Text = "TranscriberRequest is nullptr";
            }
            cur_st_result = "null";
            cur_st_closed = "null";
            cur_st_completed = "null";
        }

        // start transcriber
        private void button3_Click(object sender, EventArgs e)
        {
            int ret = -1;
            if (stPtr.native_request != IntPtr.Zero)
            {
                stPtr.SetAppKey(stPtr, appKey);
                stPtr.SetToken(stPtr, token);
                stPtr.SetUrl(stPtr, url);
                stPtr.SetFormat(stPtr, "pcm");
                stPtr.SetSampleRate(stPtr, 16000);
                stPtr.SetIntermediateResult(stPtr, true);
                stPtr.SetPunctuationPrediction(stPtr, true);
                stPtr.SetInverseTextNormalization(stPtr, true);
```

```
                stPtr.SetOnTranscriptionStarted(stPtr, DemoOnTranscriptionStarted, IntPtr.Z
ero);
                stPtr.SetOnChannelClosed(stPtr, DemoOnTranscriptionClosed, IntPtr.Zero);
                stPtr.SetOnTaskFailed(stPtr, DemoOnTranscriptionTaskFailed, IntPtr.Zero);
                stPtr.SetOnSentenceEnd(stPtr, DemoOnSentenceEnd, IntPtr.Zero);
                stPtr.SetOnTranscriptionResultChanged(stPtr, DemoOnTranscriptionResultChang
ed, IntPtr.Zero);

                ret = stPtr.Start(stPtr);

                if (st_audio_loop_flag == false)
                {
                    st_audio_loop_flag = true;
                    st_send_audio = new Thread(STAudioLab);
                    st_send_audio.Start();
                }
            }

            if (ret != 0)
            {
                nlsResult.Text = "Transcriber Start failed";
            }
            else
            {
                nlsResult.Text = "Transcriber Start success";
            }
        }

        // stop transcriber
        private void btnSTstop_Click(object sender, EventArgs e)
        {
            int ret = -1;
            if (stPtr.native_request != IntPtr.Zero)
                ret = stPtr.Stop(stPtr);

            st_send_audio_flag = false;
            st_audio_loop_flag = false;
            if (ret != 0)
            {
                nlsResult.Text = "Transcriber Stop failed";
            }
            else
            {
                nlsResult.Text = "Transcriber Stop success";
            }
        }
        #endregion

        #region Info
        private void textBox1_TextChanged(object sender, EventArgs e)
        {
            akId = tAkId.Text;
        }

        private void tAppKey_TextChanged(object sender, EventArgs e)
```

```
        private void tAppKey_TextChanged(object sender, EventArgs e)
        {
            appKey = tAppKey.Text;
        }

        private void tAkSecret_TextChanged(object sender, EventArgs e)
        {
            akSecret = tAkSecret.Text;
        }

        private void tToken_TextChanged(object sender, EventArgs e)
        {
            token = tToken.Text;
        }

        private void tUrl_TextChanged(object sender, EventArgs e)
        {
            url = tUrl.Text;
        }
        #endregion

        #region TokenButton
        // create token
        private void button3_Click_1(object sender, EventArgs e)
        {
            int ret = -1;
            tokenPtr = nlsClient.CreateNlsToken();
            if (tokenPtr.native_token != IntPtr.Zero)
            {
                if (akId != null && akSecret != null & akId.Length > 0 && akSecret.Length >
0)
                {
                    tokenPtr.SetAccessKeyId(tokenPtr, akId);
                    tokenPtr.SetKeySecret(tokenPtr, akSecret);

                    ret = tokenPtr.ApplyNlsToken(tokenPtr);
                    if (ret == -1)
                    {
                        System.Diagnostics.Debug.WriteLine("ApplyNlsToken failed");
                        nlsResult.Text = tokenPtr.GetErrorMsg(tokenPtr);
                    }
                    else
                    {
                        System.Diagnostics.Debug.WriteLine("ApplyNlsToken success");
                        token = tokenPtr.GetToken(tokenPtr);
                        tToken.Text = token;
                        expireTime = tokenPtr.GetExpireTime(tokenPtr);
                        nlsResult.Text = "ExpireTime:" + expireTime.ToString();
                    }
                }
                else
                {
                    nlsResult.Text = "CreateToken Failed, akId or Secret is null";
                }
            }
```

```
            else
            {
                nlsResult.Text = "CreateToken Failed";
            }
        }

        // release token
        private void button4_Click(object sender, EventArgs e)
        {
            if (tokenPtr.native_token != IntPtr.Zero)
            {
                nlsClient.ReleaseNlsToken(tokenPtr);
                tokenPtr.native_token = IntPtr.Zero;
                nlsResult.Text = "ReleaseNlsToken Success";
            }
            else
            {
                nlsResult.Text = "ReleaseNlsToken is nullptr";
            }
        }
        #endregion

        #region TranscriberCallback
        private CallbackDelegate DemoOnTranscriptionStarted =
            (ref NLS_EVENT_STRUCT e) =>
            {
                System.Diagnostics.Debug.WriteLine("DemoOnTranscriptionStarted = {0}", e.ms
g);
                cur_st_completed = "msg : " + e.msg;

                st_send_audio_flag = true;
            };
        private CallbackDelegate DemoOnTranscriptionClosed =
            (ref NLS_EVENT_STRUCT e) =>
            {
                System.Diagnostics.Debug.WriteLine("DemoOnTranscriptionClosed = {0}", e.msg
);
                cur_st_closed = "msg : " + e.msg;
            };
        private CallbackDelegate DemoOnTranscriptionTaskFailed =
            (ref NLS_EVENT_STRUCT e) =>
            {
                System.Diagnostics.Debug.WriteLine("DemoOnTranscriptionTaskFailed = {0}", e
.msg);
                cur_st_completed = "msg : " + e.msg;
                st_send_audio_flag = false;
                st_audio_loop_flag = false;
            };
        private CallbackDelegate DemoOnTranscriptionResultChanged =
            (ref NLS_EVENT_STRUCT e) =>
            {
                System.Diagnostics.Debug.WriteLine("DemoOnTranscriptionResultChanged = {0}"
, e.result);
                cur_st_result = "middle result : " + e.result;
```

```
            };
        private CallbackDelegate DemoOnSentenceEnd =
            (ref NLS_EVENT_STRUCT e) =>
            {
                System.Diagnostics.Debug.WriteLine("DemoOnSentenceEnd = {0}", e.msg);
                cur_st_completed = "sentenceEnd : " + e.result;
            };
        #endregion
    }
}
```

# 6.Go SDK

本文介绍如何使用阿里云智能语音服务提供的Go SDK，包括SDK的安装方法及SDK代码示例。

## 前提条件

在使用SDK前，请先阅读接口说明，详情请参见接口说明。

## 下载安装

说明

- SDK支持Go 1.16及以上版本。
- 请确认已经安装Golang环境，并完成基本配置。

1. 下载并安装SDK。

   通过以下命令完成SDK下载和安装。

   ```
   go get github.com/aliyun/alibabacloud-nls-go-sdk
   ```

2. 导入SDK。

   在代码中通过添加以下字段导入SDK。

   ```
   import ("github.com/aliyun/alibabacloud-nls-go-sdk")
   ```

## SDK常量

| 常量 | 含义 |
| --- | --- |
| SDK_VERSION | SDK版本。 |
| PCM | PCM音频格式。 |
| WAV | WAV音频格式。 |
| OPUS | OPUS音频格式。 |
| OPU | OPU音频格式。 |
| DEFAULT_DISTRIBUTE | 获取Token时使用的默认区域，默认为"cn-shanghai"。 |
| DEFAULT_DOMAIN | 获取Token时使用的默认URL，默认为"nls-meta.cn-shanghai.aliyuncs.com"。 |

| 常量 | 含义 |
| --- | --- |
| DEFAULT_VERSION | 获取Token时使用的协议版本，默认为"2019-02-28"。 |
| DEFAULT_URL | 默认公有云URL，默认为"wss://nls-gateway.cn-shanghai.aliyuncs.com/ws/v1"。 |

## 建立连接

> **注意**
>
> 使用**Akid**和**Akkey**来获取Token时，建议针对Token进行缓存，并根据获取Token时获得的过期时间参数进行及时更新；请勿频繁调用获取Token的接口，否则会造成云端限流。

### 1.ConnectionConfig

用于建立连接的基础参数。

参数说明：

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| Url | String | 访问的公共云URL，如果不确定，可以使用常量**DEFAULT_URL**。 |
| Token | String | 访问Token，详情可参见获取Token。 |
| Akid | String | 阿里云账号AccessKey ID。<br>• 若未填写**Token**字段，则需要填写该字段。<br>• 若已填写**Token**字段，则该字段可不填写。 |
| Akkey | String | 阿里云账号AccessKey Secret。<br>• 若未填写**Token**字段，则需要填写该字段。<br>• 若已填写**Token**字段，则该字段可不填写。 |
| Appkey | String | 对应项目Appkey。 |

### 2.func NewConnectionConfigWithToken(url string, appkey string, token string) *ConnectionConfig

通过URL、Appkey和Token创建连接参数。

- 参数说明：

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| Url | String | 访问的公共云URL，如果不确定，可以使用常量DEFAULT_URL。 |
| Appkey | String | 对应项目Appkey。 |
| Token | String | 访问Token，详情可参见获取Token。 |

- 返回值：

*ConnectionConfig：连接配置对象指针。

### 3. func NewConnectionConfigFromJson(jsonStr string) (*ConnectionConfig, error)

通过JSON字符串来创建连接参数。

- 参数说明：

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| jsonStr | String | 描述连接参数的JSON字符串，有效字段如下：url，token，akid，akkey，appkey。其中必须包含url和appkey，如果包含Token则不需要包含akid和akkey。 |

- 返回值：

*ConnectionConfig：连接配置对象指针。

## 实时语音识别

### 1. SpeechTranscriptionStartParam

实时语音识别参数。

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| Format | String | 音频格式，默认值：PCM。取值为OPUS、OPU和PCM，如果是OPUS或OPU时，您需要自行编码。 |
| SampleRate | Integer | 采样率，默认值：16000 Hz。 |

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| EnableIntermediateResult | Boolean | 是否打开中间结果返回。<br>• true：打开。<br>• false：关闭。 |
| EnablePunctuationPredition | Boolean | 是否打开标点预测。<br>• true：打开。<br>• false：关闭。 |
| EnableInverseTextNormalization | Boolean | 是否打开ITN（逆文本规整，inverse text normalization）。<br>• true：打开。<br>• false：关闭。 |
| MaxSentenceSilence | Integer | 语音断句检测阈值。静音时长超过该阈值会被认为断句，合法参数范围200～2000毫秒，默认值800秒。 |
| enable_words | Boolean | 是否开启返回词信息。默认不开启。<br>• true：开启。<br>• false：不开启。 |

## 2. func DefaultSpeechTranscriptionParam() SpeechTranscriptionStartParam

创建一个默认参数。

- 参数说明：无。
- 返回值：

  SpeechTranscriptionStartParam：默认参数。

## 3. func NewSpeechTranscription(...) (*SpeechTranscription, error)

创建一个实时语音识别对象。

- 参数说明：

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| config | *ConnectionConfig | 参见建立连接相关内容。 |

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| logger | *NlsLogger | 参见SDK日志相关内容。 |
| taskfailed | func(string, interface{}) | 识别过程中的错误处理回调参数，interface{}为用户自定义参数。 |
| started | func(string, interface{}) | 建连完成回调参数。 |
| sentencebegin | func(string, interface{}) | 一句话开始。 |
| sentenceend | func(string, interface{}) | 一句话结束。 |
| resultchanged | func(string, interface{}) | 识别中间结果回调参数。 |
| completed | func(string, interface{}) | 最终识别结果回调参数。 |
| closed | func(interface{}) | 连接断开回调参数。 |
| param | interface{} | 用户自定义参数。 |

- 返回值：
  - *SpeechRecognition：识别对象指针。
  - error：错误异常。

## 4. func (st *SpeechTranscription) Start(param SpeechTranscriptionStartParam, extra map[string]interface{}) (chan bool, error)

开始实时语音识别。

- 参数说明：

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| param | SpeechTranscriptionStartParam | 实时语音识别参数。 |
| extra | map[string]interface{} | 额外 `key:value` 参数。 |

- 返回值：
  - chan bool：同步start完成的管道。
  - error：错误异常。

## 5. func (st *SpeechTranscription) Stop() (chan bool, error)

停止实时语音识别。

- 参数说明：无。
- 返回值：
  - chan bool：同步stop完成的管道。
  - error：错误异常。

## 6. func (st *SpeechTranscription) Ctrl(param map[string]interface{}) error

发送控制命令，可先查看接口说明。

- 参数说明：无。

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| param | map[string]interface{} | 自定义控制命令，该字典内容会以 `key:value` 形式合并进请求的payload段中。 |

- 返回值：

  error：错误异常。

## 7. func (st *SpeechTranscription) Shutdown()

强制停止实时语音识别。

- 参数说明：无。
- 返回值：无。

## 8. func (sr *SpeechTranscription) SendAudioData(data []byte) error

发送音频，音频格式必须和参数中一致。

- 参数说明：

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| data | []byte | 音频数据。 |

- 返回值：

error：错误异常。

## SDK日志

### 1.func DefaultNlsLog() *NlsLogger

用于创建全局唯一的默认日志对象，默认日志以NLS为前缀，输出到标准错误。

- 参数说明：无。
- 返回值：

NlsLogger：日志对象指针。

### 2.func NewNlsLogger(w io.Writer, tag string, flag int) *NlsLogger

创建一个新的日志。

- 参数说明：

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| w | io.Writer | 任意实现io.Writer接口的对象。 |
| tag | String | 日志前缀，会打印到日志行首部。 |
| flag | Integer | 日志flag，具体请参见Go官方log文档。 |

- 返回值：

NlsLogger：日志对象指针。

### 3.func (logger *NlsLogger) SetLogSil(sil bool)

设置日志是否输出到对应的io.Writer。

- 参数说明：

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| sil | Boolean | 是否禁止日志输出。<br>• true：禁止。<br>• false：允许。 |

- 返回值：无。

### 4.func (logger *NlsLogger) SetDebug(debug bool)

设置是否打印Debug日志，仅影响通过Debugf或Debugln进行输出的日志。

- 参数说明：

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| debug | Boolean | 是否允许Debug日志输出。<br>• true：允许。<br>• false：禁止。 |

- 返回值：无。

## 5. func (logger *NlsLogger) SetOutput(w io.Writer)

设置日志输出方式。

- 参数说明：

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| w | io.Writer | 任意实现io.Writer接口的对象。 |

- 返回值：无。

## 6. func (logger *NlsLogger) SetPrefix(prefix string)

设置日志行的标签。

- 参数说明：

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| prefix | String | 日志行标签，会输出在日志行行首。 |

- 返回值：无。

## 7. func (logger *NlsLogger) SetFlags(flags int)

设置日志属性。

- 参数说明：

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| flags | Integer | 日志属性，具体请参见Go官方文档。 |

- 返回值：无。

## 8. 日志打印

日志打印方法：

| 方法名 | 方法说明 |
| --- | --- |
| func (l *NlsLogger) Print(v ...interface{}) | 标准日志输出。 |
| func (l *NlsLogger) Println(v ...interface{}) | 标注日志输出，行尾自动换行。 |
| func (l *NlsLogger) Printf(format string, v ...interface{}) | 带Format的日志输出，Format方式见具体请参见Go官方文档。 |
| func (l *NlsLogger) Debugln(v ...interface{}) | Debug信息日志输出，行尾自动换行。 |
| func (l *NlsLogger) Debugf(format string, v ...interface{}) | 带Format的Debug信息日志输出。 |
| func (l *NlsLogger) Fatal(v ...interface{}) | 致命错误日志输出，输出后自动进程退出。 |
| func (l *NlsLogger) Fatalln(v ...interface{}) | 致命错误日志输出，行尾自动换行，输出后自动进程退出。 |
| func (l *NlsLogger) Fatalf(format string, v ...interface{}) | 带Format的致命错误日志输出，输出后自动进程退出。 |
| func (l *NlsLogger) Panic(v ...interface{}) | 致命错误日志输出，输出后自动进程退出并打印崩溃信息。 |
| func (l *NlsLogger) Panicln(v ...interface{}) | 致命错误日志输出，行尾自动换行，输出后自动进程退出并打印崩溃信息。 |
| func (l *NlsLogger) Panicf(format string, v ...interface{}) | 带Format的致命错误日志输出，输出后自动进程退出并打印崩溃信息。 |

## 代码示例

```
package main

import (
        "errors"
        "flag"
        "fmt"
        "log"
        "os"
        "os/signal"
        "sync"
```

```
        "sync"
        "time"

        "github.com/aliyun/alibabacloud-nls-go-sdk"
)

const (
    AKID  = "Your AKID"
        AKKEY = "Your AKKEY"
        //online key
        APPKEY = "Your APPKEY"
        TOKEN  = "Your TOKEN"
)

func onTaskFailed(text string, param interface{}) {
        logger, ok := param.(*nls.NlsLogger)
        if !ok {
                log.Default().Fatal("invalid logger")
                return
        }

        logger.Println("TaskFailed:", text)
}

func onStarted(text string, param interface{}) {
        logger, ok := param.(*nls.NlsLogger)
        if !ok {
                log.Default().Fatal("invalid logger")
                return
        }

        logger.Println("onStarted:", text)
}

func onSentenceBegin(text string, param interface{}) {
        logger, ok := param.(*nls.NlsLogger)
        if !ok {
                log.Default().Fatal("invalid logger")
                return
        }

        logger.Println("onSentenceBegin:", text)
}

func onSentenceEnd(text string, param interface{}) {
        logger, ok := param.(*nls.NlsLogger)
        if !ok {
                log.Default().Fatal("invalid logger")
                return
        }

        logger.Println("onSentenceEnd:", text)
}

func onResultChanged(text string, param interface{}) {
```

```
        logger, ok := param.(*nls.NlsLogger)
        if !ok {
                log.Default().Fatal("invalid logger")
                return
        }

        logger.Println("onResultChanged:", text)
}

func onCompleted(text string, param interface{}) {
        logger, ok := param.(*nls.NlsLogger)
        if !ok {
                log.Default().Fatal("invalid logger")
                return
        }

        logger.Println("onCompleted:", text)
}

func onClose(param interface{}) {
        logger, ok := param.(*nls.NlsLogger)
        if !ok {
                log.Default().Fatal("invalid logger")
                return
        }

        logger.Println("onClosed:")
}

func waitReady(ch chan bool, logger *nls.NlsLogger) error {
        select {
        case done := <-ch:
                {
                        if !done {
                                logger.Println("Wait failed")
                                return errors.New("wait failed")
                        }
                        logger.Println("Wait done")
                }
        case <-time.After(20 * time.Second):
                {
                        logger.Println("Wait timeout")
                        return errors.New("wait timeout")
                }
        }
        return nil
}

var lk sync.Mutex
var fail = 0
var reqNum = 0

func testMultiInstance(num int) {
        pcm, err := os.Open("tests/test1.pcm")
```

```
        if err != nil {
                log.Default().Fatalln(err)
        }

        buffers := nls.LoadPcmInChunk(pcm, 320)
        param := nls.DefaultSpeechTranscriptionParam()
     config := nls.NewConnectionConfigWithAKInfoDefault(nls.DEFAULT_URL, APPKEY, AKID, AKKE
Y)
        var wg sync.WaitGroup
        for i := 0; i < num; i++ {
                wg.Add(1)
                go func(id int) {
                        defer wg.Done()
                        strId := fmt.Sprintf("ID%d   ", id)
                        logger := nls.NewNlsLogger(os.Stderr, strId, log.LstdFlags|log.Lmic
roseconds)
                        logger.SetLogSil(false)
                        logger.SetDebug(true)
                        logger.Printf("Test Normal Case for SpeechRecognition:%s", strId)
                        st, err := nls.NewSpeechTranscription(config, logger,
                                onTaskFailed, onStarted,
                                onSentenceBegin, onSentenceEnd, onResultChanged,
                                onCompleted, onClose, logger)
                        if err != nil {
                                logger.Fatalln(err)
                                return
                        }

                        test_ex := make(map[string]interface{})
                        test_ex["test"] = "hello"

                        for {
                                lk.Lock()
                                reqNum++
                                lk.Unlock()
                                logger.Println("ST start")
                                ready, err := st.Start(param, test_ex)
                                if err != nil {
                                        lk.Lock()
                                        fail++
                                        lk.Unlock()
                                        st.Shutdown()
                                        continue
                                }

                                err = waitReady(ready, logger)
                                if err != nil {
                                        lk.Lock()
                                        fail++
                                        lk.Unlock()
                                        st.Shutdown()
                                        continue
                                }
```

```
                                for _, data := range buffers.Data {
                                        if data != nil {
                                                st.SendAudioData(data.Data)
                                                time.Sleep(10 * time.Millisecond)
                                        }
                                }

                                logger.Println("send audio done")
                                ready, err = st.Stop()
                                if err != nil {
                                        lk.Lock()
                                        fail++
                                        lk.Unlock()
                                        st.Shutdown()
                                        continue
                                }

                                err = waitReady(ready, logger)
                                if err != nil {
                                        lk.Lock()
                                        fail++
                                        lk.Unlock()
                                        st.Shutdown()
                                        continue
                                }

                                logger.Println("Sr done")
                                st.Shutdown()
                        }
                }(i)
        }

        wg.Wait()
}

func main() {
        coroutineId := flag.Int("num", 1, "coroutine number")
        flag.Parse()
        log.Default().Printf("start %d coroutines", *coroutineId)

        c := make(chan os.Signal, 1)
        signal.Notify(c, os.Interrupt)
        go func() {
                for range c {
                        lk.Lock()
                        log.Printf(">>>>>>>>>REQ NUM: %d>>>>>>>>>FAIL: %d", reqNum, fail)
                        lk.Unlock()
                        os.Exit(0)
                }
        }()
        testMultiInstance(*coroutineId)
}
```

# 7.Python SDK

本文介绍如何使用阿里云智能语音服务提供的Python SDK，包括SDK的安装方法及SDK代码示例。

## 前提条件

在使用SDK前，请先阅读接口说明，详情请参见接口说明。

## 下载安装

> **说明**
>
> - SDK仅支持Python3，暂不支持Python2。
> - 请确认已安装Python包管理工具setuptools。如果没有安装，请在终端使用以下命令安装：
>
> ```
> pip install setuptools
> ```

1. 下载Python SDK。
2. 安装SDK依赖。

   进入SDK根目录使用如下命令安装SDK依赖：

   ```
   python -m pip install -r requirements.txt
   ```

3. 安装SDK。

   依赖安装完成后使用如下命令安装SDK：

   ```
   python -m pip install .
   ```

4. 安装完成后通过以下代码导入SDK。

   ```
   # -*- coding: utf-8 -*-
   import nls
   ```

> **说明**
>
> 上述命令均需要在SDK根目录中执行。

## 多线程和多并发

在CPython中，由于存在全局解释器锁，同一时刻只有一个线程可以执行Python代码（虽然某些性能导向的库可能会去除此限制）。如果您想更好地利用多核心计算机的计算资源，推荐你使用multiprocessing或concurrent.futures.ProcessPoolExecutor。 如果你想要同时运行多个I/O密集型任务，则多线程仍然是一个合适的模型。

如果单解释器有太多线程，将会在线程间切换造成更多消耗，有可能会导致SDK出现错误。不建议使用超过200线程，推荐使用multiprocessing技术或者手动使用脚本创建多个解释器。

## 关键接口

实时语音识别对应的类为**nls.NlsSpeechTranscriber**，其核心方法如下：

## 1. 初始化（__init__）

- 参数说明

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| url | String | 网关WebSocket URL地址，默认为 `wss://nls-gateway.cn-shanghai.aliyuncs.com/ws/v1` 。 |
| akid | String | 阿里云账号AccessKey ID，默认为None。<br>• 若未已填写**token**字段，则需要填写该字段。<br>• 若已填写**token**字段，则该字段可不填写。 |
| aksecret | String | 阿里云账号AccessKey Secret，默认为None。<br>• 若未已填写**token**字段，则需要填写该字段。<br>• 若已填写**token**字段，则该字段可不填写。 |
| token | String | 访问Token，详情可参见获取Token。 |
| on_start | Function | 当实时语音识别就绪时的回调参数。回调参数包含以下两种：<br>• JSON形式的字符串<br>• 用户自定义参数<br>其中，用户自定义参数为下方**callback_args**字段中返回的参数内容。 |
| on_sentence_begin | Function | 当实时语音识别开始时的回调参数。回调参数包含以下两种：<br>• JSON形式的字符串<br>• 用户自定义参数<br>其中，用户自定义参数为下方**callback_args**字段中返回的参数内容。 |

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| on_sentence_end | Function | 当实时语音识别结束时的回调参数。回调参数包含以下两种：<br>● JSON形式的字符串<br>● 用户自定义参数<br>其中，用户自定义参数为下方callback_args字段中返回的参数内容。 |
| on_result_changed | Function | 当实时语音识别返回中间结果时的回调参数。回调参数包含以下两种：<br>● JSON形式的字符串<br>● 用户自定义参数<br>其中，用户自定义参数为下方callback_args字段中返回的参数内容。 |
| on_completed | Function | 当实时语音识别返回最终识别结果时的回调参数。回调参数包含以下两种：<br>● JSON形式的字符串<br>● 用户自定义参数<br>其中，用户自定义参数为下方callback_args字段中返回的参数内容。 |
| on_error | Function | 当SDK或云端出现错误时的回调参数。回调参数包含以下两种：<br>● JSON形式的字符串<br>● 用户自定义参数<br>其中，用户自定义参数为下方callback_args字段中返回的参数内容。 |
| on_close | Function | 当和云端连接断开时的回调参数。回调参数为用户自定义参数，即用户自定义参数为下方callback_args字段中返回的参数内容。 |
| callback_args | List | 用户自定义参数列表，列表中的内容会打包（pack）成List数据结构传递给各个回调的最后一个参数。 |

- 返回值：无

## 2. start

同步开始实时语音识别，该方法会阻塞当前线程直到实时语音识别就绪（on_start回调返回）。

- 参数说明

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| aformat | String | 要识别音频格式，支持PCM，OPUS，OPU，默认值：PCM。<br>SDK不会自动将PCM编码成OPUS或OPU，如果需要使用OPUS或OPU，您可自行编码实现。 |
| sample_rate | Integer | 识别音频采样率，默认值：16000Hz。 |
| ch | Integer | 音频通道数，默认值：1，目前仅支持单通道。 |
| enable_intermediate_result | Boolean | 是否返回中间结果，默认值：False。 |
| enable_punctutation_prediction | Boolean | 是否进行识别结果标点预测，默认值：False。 |
| enable_inverse_text_normalization | Boolean | 是否进行ITN（逆文本规整，inverse text normalization），默认值：False。 |
| timeout | Integer | 阻塞超时，默认值：10秒。 |
| ping_interval | Integer | Ping包发送间隔，默认值：8秒。无需间隔可设置为0或None。 |
| ping_timeout | Integer | 是否检查Pong包超时，默认值：None。None为不检查Pong包是否超时。 |
| ex | Dict | 用户提供的额外参数，该字典内容会以 `key:value` 形式合并进请求的payload段中，具体可参见接口说明章节中的请求数据。 |

- 返回值：Boolean类型，False为失败，True为成功。

## 3. stop

停止实时语音识别，并同步等待on_completed回调结束。

- 参数说明

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| timeout | Integer | 阻塞超时，默认值：10秒。 |

- 返回值：Boolean类型，False为失败，True为成功。

## 4. shutdown

强行关闭当前请求，重复调用无副作用。

- 参数说明：无
- 返回值：无

## 5. send_audio

发送二进制音频数据，发送数据的格式需要和start中的aformat对应。

- 参数说明

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| pcm_data | Bytes | 要发送的二进制音频数据，格式需要和上一次调用时start中的aformat相对应。<br>SDK不会自动将PCM编码成OPUS或OPU，如果需要使用OPUS或OPU，您可自行编码实现。 |

- 返回值：Boolean类型，False为失败，True为成功。

## 6. ctrl

发送控制命令。

- 参数说明

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| ex | Dict | 自定义控制命令，该字典内容会以 `key:value` 形式合并进请求的payload段中，具体可参见接口说明章节中的请求数据。 |

- 返回值：Boolean类型，False为失败，True为成功。

## 代码示例

**说明**

- 本示例中使用的音频文件为16000Hz采样率，PCM格式，您可以使用*tests*文件夹下的test1.pcm，请在智能语音交互管控台将Appkey对应项目的模型设置为**通用**模型，以获取准确的识别结果；如果使用其他音频，请设置为支持该音频场景的模型。关于模型设置，请参见管理项目。
- 本示例中使用SDK内置的默认外网访问服务端URL，如果您使用阿里云上海地域的ECS，并需要通过内网访问服务端URL，请使用如下URL：`URL="ws://nls-gateway.cn-shanghai-internal.aliyuncs.com/ws/v1"` 。

```
import time
import threading
import sys

import nls


URL="wss://nls-gateway.cn-shanghai.aliyuncs.com/ws/v1"
AKID="yourAccessKeyId"
AKKEY="yourAccessKeySecret"
APPKEY="yourAppkey"

#以下代码会根据音频文件内容反复进行实时语音识别（文件转写）
class TestSt:
    def __init__(self, tid, test_file):
        self.__th = threading.Thread(target=self.__test_run)
        self.__id = tid
        self.__test_file = test_file

    def loadfile(self, filename):
        with open(filename, "rb") as f:
            self.__data = f.read()

    def start(self):
        self.loadfile(self.__test_file)
        self.__th.start()

    def test_on_sentence_begin(self, message, *args):
        print("test_on_sentence_begin:{}".format(message))

    def test_on_sentence_end(self, message, *args):
        print("test_on_sentence_end:{}".format(message))

    def test_on_start(self, message, *args):
        print("test_on_start:{}".format(message))

    def test_on_error(self, message, *args):
        print("on_error args=>{}".format(args))

    def test_on_close(self, *args):
        print("on_close: args=>{}".format(args))
```

```
    def test_on_result_chg(self, message, *args):
        print("test_on_chg:{}".format(message))

    def test_on_completed(self, message, *args):
        print("on_completed:args=>{} message=>{}".format(args, message))


    def __test_run(self):
        print("thread:{} start..".format(self.__id))
        sr = nls.NlsSpeechTranscriber(
                    url=URL,
                    akid=AKID,
                    aksecret=AKKEY,
                    appkey=APPKEY,
                    on_sentence_begin=self.test_on_sentence_begin,
                    on_sentence_end=self.test_on_sentence_end,
                    on_start=self.test_on_start,
                    on_result_changed=self.test_on_result_chg,
                    on_completed=self.test_on_completed,
                    on_error=self.test_on_error,
                    on_close=self.test_on_close,
                    callback_args=[self.__id]
                )
        while True:
            print("{}: session start".format(self.__id))
            r = sr.start(aformat="pcm",
                    enable_intermediate_result=True,
                    enable_punctutation_prediction=True,
                    enable_inverse_text_normalization=True)

            self.__slices = zip(*(iter(self.__data),) * 640)
            for i in self.__slices:
                sr.send_audio(bytes(i))
                time.sleep(0.01)

            sr.ctrl(ex={"test":"tttt"})
            time.sleep(1)

            r = sr.stop()
            print("{}: sr stopped:{}".format(self.__id, r))
            time.sleep(1)

def multiruntest(num=500):
    for i in range(0, num):
        name = "thread" + str(i)
        t = TestSt(name, "tests/test1.pcm")
        t.start()

nls.enableTrace(True)
multiruntest(1)
```

# 8.Node.js SDK

本文介绍如何使用阿里云智能语音服务提供的Node.js SDK，包括SDK的安装方法及SDK代码示例。

## 前提条件

在使用SDK前，请先阅读接口说明，详情请参见接口说明。

## 下载安装

说明

- SDK支持nodev14及以上版本。
- 请确认已经安装nodejs&npm环境，并完成基本配置。

1. 下载并安装SDK。

   通过以下命令完成SDK下载和安装。

```
npm install alibabacloud-nls
```

2. 导入SDK。

   在代码中使用require或者import导入SDK。

```
const Nls = require('alibabacloud-nls')
//Nls内部含SpeechRecognition, SpeechTranscription, SpeechSynthesizer
//以下为使用import导入SDK
//import { SpeechRecognition } from "alibabacloud-nls"
//import { SpeechTranscription } from "alibabacloud-nls"
//import { SpeechSynthesizer } from "alibabacloud-nls"
```

## 实时语音识别

### Class: SpeechTranscription

SpeechTranscription类用于进行实时语音识别。

- 构造函数参数说明：

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| config | Object | 连接配置对象。 |

- config object说明：

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| url | String | 服务URL地址。 |

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| token | String | 访问Token，详情可参见获取Token。 |
| appkey | String | 对应项目Appkey。 |

## defaultStartParams()

返回一个默认的推荐参数，其中Format为PCM，采样率为16000 Hz，中间结果、标点预测和ITN均为打开状态。您在拿到默认对象后可以根据自身需求，结合接口说明中的参数列表来添加和修改参数。

- 参数说明：无。
- 返回值：

  object类型对象，字段如下：

```
{
    "format": "pcm",
    "sample_rate": 16000,
    "enable_intermediate_result": true,
    "enable_punctuation_predition": true,
    "enable_inverse_text_normalization": true
}
```

## on(which, handler)

设置事件回调。

- 参数说明：

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| which | String | 事件名称。 |
| handler | Function | 回调函数。 |

支持的回调事件如下：

| 事件名称 | 事件说明 | 回调函数参数个数 | 回调函数参数说明 |
| --- | --- | --- | --- |
| started | 实时语音识别开始。 | 1 | String类型，开始信息。 |
| changed | 实时语音识别中间结果。 | 1 | String类型，中间结果信息。 |

| 事件名称 | 事件说明 | 回调函数参数个数 | 回调函数参数说明 |
| --- | --- | --- | --- |
| completed | 实时语音识别完成。 | 1 | String类型，完成信息。 |
| closed | 连接关闭。 | 0 | 无。 |
| failed | 错误。 | 1 | String类型，错误信息。 |
| begin | 提示句子开始。 | 1 | String类型，相关信息。 |
| end | 提示句子结束。 | 1 | String类型，相关信息。 |

- 返回值：无。

## async start(param, enablePing, pingInterval)

根据param发起一次一句话识别，param可以参考defaultStartParams方法的返回，具体参数见接口说明。

- 参数说明：

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| param | Object | 实时语音识别参数。 |
| enablePing | Boolean | 是否自动向云端发送ping请求，默认false。<br>• true：发送。<br>• false：不发送。 |
| pingInterval | Number | 发ping请求间隔时间，默认6000，单位为毫秒。 |

- 返回值： Promise对象，当started事件发生后触发resolve，并携带started信息；当任何错误发生后触发reject，并携带异常信息。

## async close(param)

停止一句话识别。

- 参数说明：

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| param | Object | 实时语音识别结束参数。 |

- 返回值：

  Promise对象，当completed事件发生后触发resolve，并携带completed信息；当任何错误发生后触发reject，并携带异常信息。

## shutdown()

强制断开连接。

- 参数说明：无。
- 返回值：无。

## sendAudio(data)

发送音频，音频格式必须和参数中一致。

- 参数说明：

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| data | Buffer | 二进制音频数据。 |

- 返回值：无。

## 代码示例

```
"use strict"

const Nls = require("alibabacloud-nls")
const fs = require("fs")
const sleep = (waitTimeInMs) => new Promise(resolve => setTimeout(resolve, waitTimeInMs))

const URL = "wss://nls-gateway.cn-shanghai.aliyuncs.com/ws/v1"
const APPKEY = "Your Appkey"
const TOKEN = "Your Token"

let audioStream = fs.createReadStream("test1.pcm", {
  encoding: "binary",
  highWaterMark: 1024
})
let b1 = []

audioStream.on("data", (chunk) => {
  let b = Buffer.from(chunk, "binary")
  b1.push(b)
})
```

```
audioStream.on("close", async ()=>{
  while (true) {
    let st = new Nls.SpeechTranscription({
      url: URL,
      appkey:APPKEY,
      token:TOKEN
    })

    st.on("started", (msg)=>{
      console.log("Client recv started:", msg)
    })

    st.on("changed", (msg)=>{
      console.log("Client recv changed:", msg)
    })

    st.on("completed", (msg)=>{
      console.log("Client recv completed:", msg)
    })

    st.on("closed", () => {
      console.log("Client recv closed")
    })

    st.on("failed", (msg)=>{
      console.log("Client recv failed:", msg)
    })

    st.on("begin", (msg)=>{
      console.log("Client recv begin:", msg)
    })

    st.on("end", (msg)=>{
      console.log("Client recv end:", msg)
    })

    try {
      await st.start(st.defaultStartParams(), true, 6000)
    } catch(error) {
      console.log("error on start:", error)
      continue
    }

    try {
      for (let b of b1) {
        if (!st.sendAudio(b)) {
          throw new Error("send audio failed")
        }
        await sleep(20)
      }
    } catch(error) {
      console.log("sendAudio failed:", error)
      continue
    }
```

```
    try {
      console.log("close...")
      await st.close()
    } catch(error) {
      console.log("error on close:", error)
    }
    await sleep(2000)
  }
})
```

# 9.微信小程序

本文介绍如何使用阿里云智能语音服务提供的微信小程序SDK，包括SDK的安装方法及SDK代码示例。

## 前提条件

在使用SDK前，请先阅读接口说明，详情请参见接口说明。

## 下载安装

> 说明
>
> - 微信基础库要求2.4.4及以上版本。
> - 请确认已经安装微信小程序开发环境，并完成基本配置。具体可参见微信开发者工具。
> - 需要提前将如下URL添加到微信小程序后台服务器域名中：
>  - request合法域名：`https://nls-meta.cn-shanghai.aliyuncs.com`
>  - socket合法域名：`wss://nls-gateway.cn-shanghai.aliyuncs.com`

1. 下载并安装SDK。

   通过Github下载对应SDK代码。

2. 导入SDK。

   您可将下载好的代码放入工程合适目录下，然后根据目录位置通过require进行导入。

## 获取Token

### getToken

获取Token并以**akid**和**akkey**为key缓存对应Token，如果缓存的Token过期则自动刷新并获取。缓存机制请参见微信小程序文档的数据缓存部分。

- 参数说明：无。
- 返回值：String类型的Token。

### getTokenInner

直接获取Token，不带任何缓存机制，适用于用户自定义缓存方式。

> 注意
>
> **频繁调用该接口会被服务端拒绝访问。**

- 参数说明：无。
- 返回值：String类型的Token。

## 实时语音识别

### Class: SpeechTranscription

SpeechTranscription类用于进行实时语音识别。

- 构造函数参数说明：

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| config | Object | 连接配置对象。 |

- config object说明：

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| url | String | 服务URL地址。 |
| token | String | 访问Token，详情可参见获取Token。 |
| appkey | String | 对应项目Appkey。 |

## defaultStartParams()

返回一个默认的推荐参数，其中Format为PCM，采样率为16000 Hz，中间结果、标点预测和ITN均为打开状态。您在拿到默认对象后可以根据自身需求，结合接口说明中的参数列表来添加和修改参数。

- 参数说明：无。
- 返回值：

  object类型对象，字段如下：

```
{
    "format": "pcm",
    "sample_rate": 16000,
    "enable_intermediate_result": true,
    "enable_punctuation_predition": true,
    "enable_inverse_text_normalization": true
}
```

## on(which, handler)

设置事件回调。

- 参数说明：

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| which | String | 事件名称。 |

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| handler | Function | 回调函数。 |

支持的回调事件如下：

| 事件名称 | 事件说明 | 回调函数参数个数 | 回调函数参数说明 |
| --- | --- | --- | --- |
| started | 实时语音识别开始。 | 1 | String类型，开始信息。 |
| changed | 实时语音识别中间结果。 | 1 | String类型，中间结果信息。 |
| completed | 实时语音识别完成。 | 1 | String类型，完成信息。 |
| closed | 连接关闭。 | 0 | 无。 |
| failed | 错误。 | 1 | String类型，错误信息。 |
| begin | 提示句子开始。 | 1 | String类型，相关信息。 |
| end | 提示句子结束。 | 1 | String类型，相关信息。 |

- 返回值：无。

## async start(param)

根据param发起一次一句话识别，param可以参考defaultStartParams方法的返回，具体参数见接口说明。

- 参数说明：

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| param | Object | 实时语音识别参数。 |

- 返回值： Promise对象，当started事件发生后触发resolve，并携带started信息；当任何错误发生后触发reject，并携带异常信息。

## async close(param)

停止一句话识别。

- 参数说明：

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| param | Object | 实时语音识别结束参数。 |

- 返回值：

  Promise对象，当completed事件发生后触发resolve，并携带completed信息；当任何错误发生后触发reject，并携带异常信息。

## shutdown()

强制断开连接。

- 参数说明：无。
- 返回值：无。

## sendAudio(data)

发送音频，音频格式必须和参数中一致。

- 参数说明：

| 参数 | 类型 | 参数说明 |
| --- | --- | --- |
| data | ArrayBuffer | 二进制音频数据。 |

- 返回值：无。

## 代码示例

以下代码示例仅供参考，代码中使用微信小程序自带录音功能，实际使用时，需要考虑微信小程序的限制，以及前端页面设计和具体业务功能。

```
// pages/st/st.js

const app = getApp()

const getToken = require("../../utils/token").getToken
const SpeechTranscription = require("../../utils/st")
const sleep = require("../../utils/util").sleep

Page({

    /**
     * 页面的初始数据
     */
    data: {
        stStart : false,
        stResult : "未开始识别"
    },

    /**
```

```
     * 生命周期函数--监听页面加载
     */
    onLoad: async function (options) {
        wx.getRecorderManager().onFrameRecorded((res)=>{
            if (res.isLastFrame) {
                console.log("record done")
            }
            if (this.data.st && this.data.stStart) {
                console.log("send " + res.frameBuffer.byteLength)
                this.data.st.sendAudio(res.frameBuffer)
            }
        })
        wx.getRecorderManager().onStart(()=>{
            console.log("start recording...")
        })
        wx.getRecorderManager().onStop((res) => {
            console.log("stop recording...")
            if (res.tempFilePath) {
                wx.removeSavedFile({
                    filePath:res.tempFilePath
                })
            }
        })
        wx.getRecorderManager().onError((res) => {
            console.log("recording failed:" + res)
        })

        try {
            this.data.token = await getToken(app.globalData.AKID,
                 app.globalData.AKKEY)
        } catch (e) {
            console.log("error on get token:", JSON.stringify(e))
            return
        }
        let st = new SpeechTranscription({
            url : app.globalData.URL,
            appkey: app.globalData.APPKEY,
            token: this.data.token
        })

        st.on("started", (msg)=> {
            console.log("Client recv started")
            this.setData({
                stResult : msg
            })
        })

        st.on("changed", (msg)=>{
            console.log("Client recv changed:", msg)
            this.setData({
                stResult : msg
            })
        })
```

```
        st.on("completed", (msg)=>{
            console.log("Client recv completed:", msg)
            this.setData({
                stResult : msg
            })
        })

        st.on("begin", (msg)=>{
            console.log("Client recv sentenceBegin:", msg)
            this.setData({
                stResult : msg
            })
          })

          st.on("end", (msg)=>{
            console.log("Client recv sentenceEnd:", msg)
            this.setData({
                stResult : msg
            })
          })

        st.on("closed", () => {
            console.log("Client recv closed")
        })

        st.on("failed", (msg)=>{
            console.log("Client recv failed:", msg)
            this.setData({
                stResult : msg
            })
        })

        this.data.st = st
    },

    /**
     * 生命周期函数--监听页面初次渲染完成
     */
    onReady: function () {

    },

    /**
     * 生命周期函数--监听页面显示
     */
    onShow: function () {

    },

    /**
     * 生命周期函数--监听页面隐藏
     */
    onHide: function () {
```

```
    },

    /**
     * 生命周期函数--监听页面卸载
     */
    onUnload: function () {
        console.log("st onUnload")
        this.data.stStart = false
        wx.getRecorderManager().stop()
        if (this.data.st) {
            this.data.st.shutdown()
        } else {
            console.log("st is null")
        }
    },

    /**
     * 页面相关事件处理函数--监听用户下拉动作
     */
    onPullDownRefresh: function () {

    },

    /**
     * 页面上拉触底事件的处理函数
     */
    onReachBottom: function () {

    },

    /**
     * 用户点击右上角分享
     */
    onShareAppMessage: function () {

    },
    onStStart: async function() {
        if (!this.data.st) {
            console.log("st is null")
            return
        }

        if (this.data.stStart) {
            console.log("st is started!")
            return
        }
        let st = this.data.st
        try {
            await st.start(st.defaultStartParams())
            this.data.stStart = true
        } catch (e) {
            console.log("start failed:" + e)
            return
        }
```

```
        wx.getRecorderManager().start({
            duration: 600000,
            numberOfChannels: 1,
            sampleRate : 16000,
            format: "PCM",
            frameSize: 4
        })
    },

    onStStop: async function() {
        wx.getRecorderManager().stop()
        await sleep(500)
        if (this.data.stStart && this.data.st) {
            try {
                console.log("prepare close st")
                await this.data.st.close()
                this.data.stStart = false
            } catch(e) {
                console.log("close st failed:" + e)
            }
        }
    }
})
```

# 10.WebSocket协议说明

如果您不希望引入阿里云智能语音交互产品SDK，或者目前提供的Java、C或C++的SDK不能满足您的要求，可以基于本文描述自行开发代码访问阿里语音服务。

## 功能介绍

阿里云智能语音交互产品通过WebSocket协议对外提供实时语音流语音转写功能，支持长语音。其中指令、事件皆为WebSocket协议Text类型的DataFrame，音频流需要以Binary Frame的形式上传至服务端，调用时序需要符合协议要求的交互流程。发送语音数据使用Websocket的二进制帧BinaryFrame，具体可参见Data Frames。

- 支持的输入格式：PCM（无压缩的PCM或WAV文件）、16bit采样位数、单声道（mono）。
- 支持的音频采样率：8000Hz/16000Hz。
- 支持设置返回结果：是否返回中间识别结果，在后处理中添加标点，将中文数字转为阿拉伯数字输出。
- 支持设置多语言识别：在控制台编辑项目中进行模型选择，详情请参见管理项目。

## 鉴权

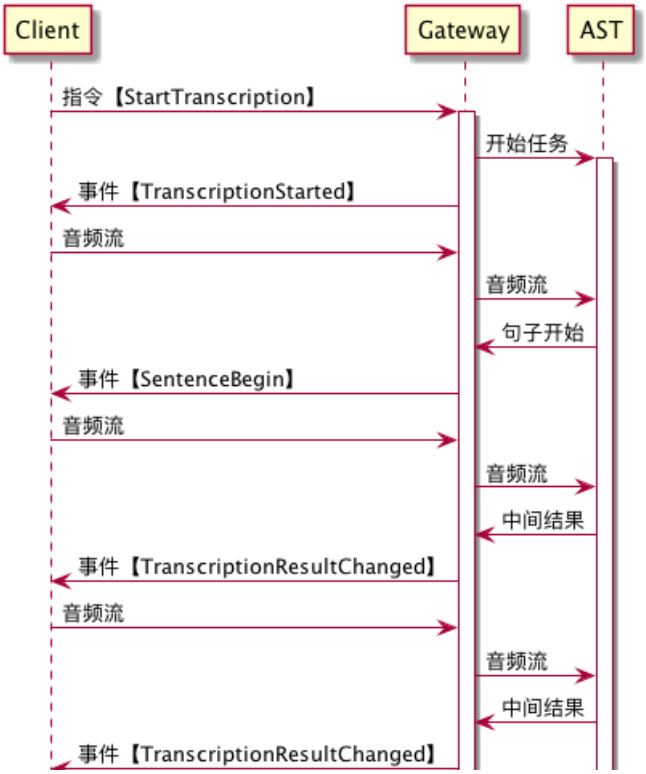
服务端通过临时Token进行鉴权，请求时需要在URL中携带Token参数，Token获取方式请参见获取Token。获取Token之后通过如下方式访问语音服务端。

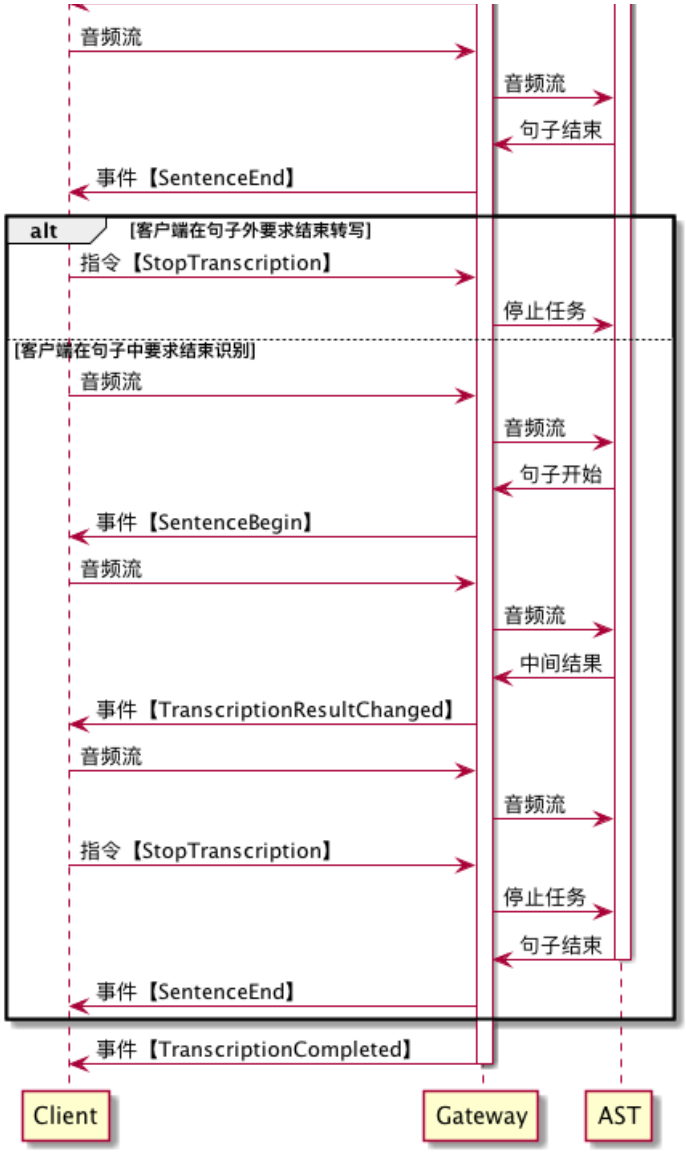
| 访问类型 | 说明 | URL |
| --- | --- | --- |
| 外网访问 | 所有服务器均可使用外网访问URL。 | wss://nls-gateway.cn-shanghai.aliyuncs.com/ws/v1?token=<your token> |

| 访问类型 | 说明 | URL |
| --- | --- | --- |
| 上海ECS内网访问 | 使用阿里云上海ECS（即ECS地域为华东2（上海）），可使用内网访问URL。ECS的经典网络不能访问AnyTunnel，即不能在内网访问语音服务；如果希望使用AnyTunnel，需要创建专有网络在其内部访问。<br>说明<br>• 使用内网访问方式，将不产生ECS实例的公网流量费用。<br>• 关于ECS的网络类型请参见网络类型。 | ws://nls-gateway.cn-shanghai-internal.aliyuncs.com:80/ws/v1?token=<your token> |

## 交互流程

指令及音频流需要严格按照下图所示顺序发送，否则会导致和服务端交互失败。






## 指令

请求指令用于控制语音识别任务的起止，标识任务边界，以JSON格式的Text Frame方式发送服务端请求，需要在Header中设置请求的基础信息。指令由Header和Payload两部分组成，其中Header部分为统一格式，不同指令的Payload部分格式各不相同。

## 1、Header格式说明

Header格式如下：

| 参数 | 类型 | 是否必选 | 说明 |
| --- | --- | --- | --- |
| appkey | String | 是 | 管控台创建的项目Appkey。 |
| message_id | String | 是 | 当次消息请求ID，随机生成32位唯一ID。 |

| 参数 | 类型 | 是否必选 | 说明 |
| --- | --- | --- | --- |
| task_id | String | 是 | 整个实时语音识别的会话ID，整个请求中需要保持一致，32位唯一ID。 |
| namespace | String | 是 | 访问的产品名称，固定为“SpeechTranscriber”。 |
| name | String | 是 | 指令名名称，包含StartTranscription和StopTranscription指令。具体请参见StartTranscription指令和StopTranscription指令。 |

## 2、StartTranscription指令

Payload对象参数说明：

| 参数 | 类型 | 是否必选 | 说明 |
| --- | --- | --- | --- |
| format | String | 否 | 音频编码格式，默认是PCM（无压缩的PCM文件或WAV文件），16bit采样位数的单声道。 |
| sample_rate | Integer | 否 | 音频采样率，默认是16000Hz，根据音频采样率在管控台对应项目中配置支持该采样率及场景的模型。 |
| enable_intermediate_result | Boolean | 否 | 是否返回中间识别结果，默认是False。 |
| enable_punctuation_prediction | Boolean | 否 | 是否在后处理中添加标点，默认是False。 |
| enable_inverse_text_normalization | Boolean | 否 | 是否在后处理中执行ITN，设置为true时，中文数字将转为阿拉伯数字输出，默认是False。<br>说明<br>不对词信息进行ITN转换。 |
| customization_id | String | 否 | 自学习模型ID。 |

| 参数 | 类型 | 是否必选 | 说明 |
| --- | --- | --- | --- |
| vocabulary_id | String | 否 | 定制泛热词ID。 |
| max_sentence_silence | Integer | 否 | 语音断句检测阈值，静音时长超过该阈值会被认为断句，参数范围200ms ~ 2000ms，默认值800ms。 |
| enable_words | Boolean | 否 | 是否开启返回词信息，默认是False。 |
| enable_ignore_sentence_timeout | Boolean | 否 | 是否忽略实时识别中的单句识别超时，默认是False。 |
| disfluency | Boolean | 否 | 是否对识别文本进行顺滑（去除语气词，重复说等），默认是False。 |
| speech_noise_threshold | Float | 否 | 噪音参数阈值，参数范围：[-1,1]。取值说明如下：<br>• 取值越趋于-1，噪音被判定为语音的概率越大。<br>• 取值越趋于+1，语音被判定为噪音的概率越大。<br>说明<br>该参数属高级参数，调整需慎重并重点测试。 |
| enable_semantic_sentence_detection | Boolean | 否 | 是否开启语语义断句，默认是False。 |

示例代码如下：

```
{
    "header": {
        "message_id": "05450bf69c53413f8d88aed1ee60****",
        "task_id": "640bc797bb684bd6960185651307****",
        "namespace": "SpeechTranscriber",
        "name": "StartTranscription",
        "appkey": "17d4c634****"
    },
    "payload": {
        "format": "opus",
        "sample_rate": 16000,
        "enable_intermediate_result": true,
        "enable_punctuation_prediction": true,
        "enable_inverse_text_normalization": true
    }
}
```

## 3、StopTranscription指令

StopTranscription指令要求服务端停止语音转写，Payload为空。示例代码如下：

```
{
    "header": {
        "message_id": "05450bf69c53413f8d88aed1ee60****",
        "task_id": "640bc797bb684bd6960185651307****",
        "namespace": "SpeechTranscriber",
        "name": "StopTranscription",
        "appkey": "17d4c634****"
    }
}
```

# 事件

## 1、TranscriptionStarted事件

TranscriptionStarted事件表示服务端已经准备好了进行识别，客户端可以发送音频数据了。

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| session_id | String | 客户端请求时传入session_id的话则原样返回，否则由服务端自动生成32位唯一ID。 |

示例格式如下：

```
{
    "header": {
        "message_id": "05450bf69c53413f8d88aed1ee60****",
        "task_id": "640bc797bb684bd6960185651307****",
        "namespace": "SpeechTranscriber",
        "name": "TranscriptionStarted",
        "status": 20000000,
        "status_message": "GATEWAY|SUCCESS|Success."
    },
    "payload": {
        "session_id": "1231231dfdf****"
    }
}
```

## 2、SentenceBegin事件

SentenceBegin事件表示服务端检测到了一句话的开始。

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| index | Integer | 句子编号，从1开始递增。 |
| time | Integer | 句子开始时间相对整个音频流的开始时间，单位是毫秒。 |

示例格式如下：

```
{
    "header": {
        "message_id": "05450bf69c53413f8d88aed1ee60****",
        "task_id": "640bc797bb684bd6960185651307****",
        "namespace": "SpeechTranscriber",
        "name": "SentenceBegin",
        "status": 20000000,
        "status_message": "GATEWAY|SUCCESS|Success."
    },
    "payload": {
        "index": 1,
        "time": 320
    }
}
```

## 3、TranscriptionResultChanged事件

TranscriptionResultChanged事件表示识别结果发生了变化。

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| index | Integer | 句子编号，从1开始递增。 |
| time | Integer | 当前已处理的音频时长，单位是毫秒。 |
| result | String | 当前的识别结果。 |
| words | Word | 词信息。 |
| status | Integer | 状态码。 |

Word 结构：

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| text | String | 文本。 |
| startTime | Integer | 词开始时间。 |
| endTime | Integer | 词结束时间。 |

示例格式如下：

```
{
    "header":{
        "message_id":"05450bf69c53413f8d88aed1ee60****",
        "task_id":"640bc797bb684bd6960185651307****",
        "namespace":"SpeechTranscriber",
        "name":"TranscriptionResultChanged",
        "status":20000000,
        "status_message":"GATEWAY|SUCCESS|Success."
    },
    "payload":{
        "index":1,
        "time":1800,
        "result":"今年双十一",
        "words":[
            {
                "text":"今年",
                "startTime":1,
                "endTime":2
            },
            {
              "text":"双十一",
              "startTime":2,
              "endTime":3
            }
        ]
    }
}
```

## 4、SentenceEnd事件

SentenceEnd事件表示服务端检测到了一句话的结束。

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| index | Integer | 句子编号，从1开始递增。 |
| time | Integer | 当前已处理的音频时长，单位是毫秒。 |
| begin_time | Integer | 这句话对应的SentenceBegin事件的时间，单位是毫秒。 |
| result | String | 当前的识别结果。 |
| confidence | Double | 结果置信度，取值范围[0.0,1.0]，值越大表示置信度越高。 |

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| words | Word | 词信息。 |
| status | Integer | 状态码，默认值 20000000，开启 `enable_ignore_sentence_timeout` 后如果遇到超时会返回错误码51040104并保持当前连接。 |
| stash_result | StashResult | 暂存结果，开启语意断句后会返回下一句未断句中间结果。 |

StashResult结构：

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| sentenceId | Integer | 句子编号，从1开始递增。 |
| beginTime | Integer | 句子开始时间。 |
| text | String | 转写内容。 |
| currentTime | Integer | 当前处理时间。 |

示例格式如下：

```
{
    "header": {
        "message_id": "05450bf69c53413f8d88aed1ee60****",
        "task_id": "640bc797bb684bd6960185651307****",
        "namespace": "SpeechTranscriber",
        "name": "SentenceEnd",
        "status": 20000000,
        "status_message": "GATEWAY|SUCCESS|Success."
    },
    "payload": {
        "index": 1,
        "time": 3260,
        "begin_time": 1800,
        "result": "今年双十一我要买电视"
    }
}
```

## 5、TranscriptionCompleted事件

TranscriptionCompleted事件表示服务端已停止了语音转写。示例格式如下：

```
{
    "header": {
        "message_id": "05450bf69c53413f8d88aed1ee60****",
        "task_id": "640bc797bb684bd6960185651307****",
        "namespace": "SpeechTranscriber",
        "name": "TranscriptionCompleted",
        "status": 20000000,
        "status_message": "GATEWAY|SUCCESS|Success."
    }
}
```