ALIBABA CLOUD

阿里云

智能语音交互

语音合成

文档版本：20220704

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 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 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 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 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.RESTful API 21
3.移动端SDK 55
3.1. 接口说明 55
3.2. Android SDK 63
3.3. iOS SDK 68
4.Java SDK 74
5.C++ SDK 80
6.Python SDK 95
7.C# SDK 101
8.Go SDK 117
9.Node.js SDK 128
10.微信小程序 133
11.SSML标记语言介绍 141
12.语音合成时间戳功能介绍 193
13.界面化的TTS下载工具 199

1.接口说明

语音合成为您提供将输入文本合成为语音二进制数据的功能。

←返回语音合成产品详情页

功能介绍

- 支持输出PCM、WAV和MP3编码格式数据。
- 支持设置语速、语调和音量。
- 支持设置不同场景及风格的声音。
- 支持一次性合成300字符以内的文字，其中1个汉字、1个英文字母或1个标点均算作1个字符，超过300个字符的内容将会截断。
- 仅支持采用UTF-8编码的文本输入。
- 支持多情感声音调用，具体请参见SSML标记语言介绍中的<emotion>标签。

说明

- 字级别音素边界接口：语音合成服务在输出音频的同时，可输出每个汉字/英文单词在音频中的时间位置，即时间戳。该时间信息可用于驱动虚拟人口型、做视频配音字幕等。详情请参见语音合成时间戳功能介绍。
- 文学场景相关发音人信息，请参见接口说明。
- 如需使用Android或iOS SDK，请参见移动端接口说明。

名称	voice参数值	类型	适用场景	支持语言	支持采样率（Hz）	支持时间戳（字级别音素边界）接口	支持儿化音	声音品质
知米_多情感	zhimi_emo	多种情感女声	通用场景	中文及中英文混合场景	8K/16K	是	否	标准版
知燕_多情感	zhiyan_emo	多种情感女声	通用场景	中文及中英文混合场景	8K/16K	是	否	标准版
知贝_多情感	zhibei_emo	多种情感童声	通用场景	中文及中英文混合场景	8K/16K	是	否	标准版

名称	voice参数值	类型	适用场景	支持语言	支持采样率（Hz）	支持时间戳（字级别音素边界）接口	支持儿化音	声音品质
知甜_多情感	zhitian_emo	多种情感女声	通用场景	中文及中英文混合场景	8K/16K	是	否	标准版
小云	xiaoyun	标准女声	通用场景	中文及中英文混合场景	8K/16K	否	否	lite版
小刚	xiaogang	标准男声	通用场景	中文及中英文混合场景	8K/16K	否	否	lite版
若兮	ruoxi	温柔女声	通用场景	中文及中英文混合场景	8K/16K/24K	否	否	标准版
思琪	siqi	温柔女声	通用场景	中文及中英文混合场景	8K/16K/24K	是	否	标准版
思佳	sijia	标准女声	通用场景	中文及中英文混合场景	8K/16K/24K	否	否	标准版
思诚	sicheng	标准男声	通用场景	中文及中英文混合场景	8K/16K/24K	是	否	标准版
艾琪	aiqi	温柔女声	通用场景	中文及中英文混合场景	8K/16K	是	否	标准版
艾佳	aijia	标准女声	通用场景	中文及中英文混合场景	8K/16K	是	否	标准版

名称	voice参数值	类型	适用场景	支持语言	支持采样率（Hz）	支持时间戳（字级别音素边界）接口	支持儿化音	声音品质
艾诚	aicheng	标准男声	通用场景	中文及中英文混合场景	8K/16K	是	否	标准版
艾达	aida	标准男声	通用场景	中文及中英文混合场景	8K/16K	是	否	标准版
宁儿	ninger	标准女声	通用场景	纯中文场景	8K/16K/24K	否	否	标准版
瑞琳	ruilin	标准女声	通用场景	纯中文场景	8K/16K/24K	否	否	标准版
思悦	siyue	温柔女声	客服场景	中文及中英文混合场景	8K/16K/24K	否	否	标准版
艾雅	aiya	严厉女声	客服场景	中文及中英文混合场景	8K/16K	是	否	标准版
艾夏	aixia	亲和女声	客服场景	中文及中英文混合场景	8K/16K	是	否	标准版
艾美	aimei	甜美女声	客服场景	中文及中英文混合场景	8K/16K	是	否	标准版
艾雨	aiyu	自然女声	客服场景	中文及中英文混合场景	8K/16K	是	否	标准版

名称	voice参数值	类型	适用场景	支持语言	支持采样率（Hz）	支持时间戳（字级别音素边界）接口	支持儿化音	声音品质
艾悦	aiyue	温柔女声	客服场景	中文及中英文混合场景	8K/16K	是	否	标准版
艾婧	aijing	严厉女声	客服场景	中文及中英文混合场景	8K/16K	是	否	标准版
小美	xiaomei	甜美女声	客服场景	中文及中英文混合场景	8K/16K/24K	否	否	标准版
艾娜	aina	浙普女声	客服场景	纯中文场景	8K/16K	是	否	标准版
伊娜	yina	浙普女声	客服场景	纯中文场景	8K/16K/24K	否	否	标准版
思婧	sijing	严厉女声	客服场景	纯中文场景	8K/16K/24K	是	否	标准版
思彤	sitong	儿童音	童声场景	纯中文场景	8K/16K/24K	否	否	标准版
小北	xiaobei	萝莉女声	童声场景	纯中文场景	8K/16K/24K	是	否	标准版
艾彤	aitong	儿童音	童声场景	纯中文场景	8K/16K	是	否	标准版
艾薇	aiwei	萝莉女声	童声场景	纯中文场景	8K/16K	是	否	标准版

名称	voice参数值	类型	适用场景	支持语言	支持采样率（Hz）	支持时间戳（字级别音素边界）接口	支持儿化音	声音品质
艾宝	aibao	萝莉女声	童声场景	纯中文场景	8K/16K	是	否	标准版
Harry	harry	英音男声	英文场景	英文场景	8K/16K	否	否	标准版
Abby	abby	美音女声	英文场景	英文场景	8K/16K	是	否	标准版
Andy	andy	美音男声	英文场景	英文场景	8K/16K	否	否	标准版
Eric	eric	英音男声	英文场景	英文场景	8K/16K	否	否	标准版
Emily	emily	英音女声	英文场景	英文场景	8K/16K	否	否	标准版
Luna	luna	英音女声	英文场景	英文场景	8K/16K	是	否	标准版
Luca	luca	英音男声	英文场景	英文场景	8K/16K	否	否	标准版
Wendy	wendy	英音女声	英文场景	英文场景	8K/16K/24K	否	否	标准版
William	william	英音男声	英文场景	英文场景	8K/16K/24K	否	否	标准版
Olivia	olivia	英音女声	英文场景	英文场景	8K/16K/24K	否	否	标准版
姗姗	shanshan	粤语女声	方言场景	标准粤文（简体）及粤英文混合场景	8K/16K/24K	否	否	标准版

名称	voice参数值	类型	适用场景	支持语言	支持采样率（Hz）	支持时间戳（字级别音素边界）接口	支持儿化音	声音品质
小玥	chuangirl	四川话女声	方言场景	中文及中英文混合场景	8K/16K	否	否	标准版
Lydia	lydia	英中双语女声	英文场景	英文及英中文混合场景	8K/16K	是	否	标准版
艾硕	aishuo	自然男声	客服场景	中文及中英文混合场景	8K/16K	是	否	标准版
青青	qingqing	中国台湾话女声	方言场景	中文场景	8K/16K	否	否	标准版
翠姐	cuijie	东北话女声	方言场景	中文场景	8K/16K	否	是	标准版
小泽	xiaoze	湖南重口音男声	方言场景	中文场景	8K/16K	否	否	标准版
智香	tomoka	日语女声	多语种场景	日文场景	8K/16K	是	否	标准版
智也	tomoya	日语男声	多语种场景	日文场景	8K/16K	是	否	标准版
Annie	annie	美语女声	英文场景	英文场景	8K/16K	是	否	标准版
佳佳	jiajia	粤语女声	方言场景	标准粤文（简体）及粤英文混合场景	8K/16K	是	否	标准版

名称	voice参数值	类型	适用场景	支持语言	支持采样率（Hz）	支持时间戳（字级别音素边界）接口	支持儿化音	声音品质
Indah	indah	印尼语女声	多语种场景	纯印尼语场景	8K/16K	否	否	标准版
桃子	taozi	粤语女声	方言场景	支持标准粤文（简体）及粤英文混合场景	8K/16K	是	否	标准版
柜姐	guijie	亲切女声	通用场景	支持中文及中英文混合场景	8K/16K	是	是	标准版
Stella	stella	知性女声	通用场景	支持中文及中英文混合场景	8K/16K	是	是	标准版
Stanley	stanley	沉稳男声	通用场景	支持中文及中英文混合场景	8K/16K	是	是	标准版
Kenny	kenny	沉稳男声	通用场景	支持中文及中英文混合场景	8K/16K	是	是	标准版
Rosa	rosa	自然女声	通用场景	支持中文及中英文混合场景	8K/16K	是	是	标准版
Farah	farah	马来语女声	多语种场景	仅支持纯马来语场景	8K/16K	否	否	标准版
马树	mashu	儿童剧男声	通用场景	支持中文及中英文混合场景	8K/16K	是	否	标准版

名称	voice参数值	类型	适用场景	支持语言	支持采样率（Hz）	支持时间戳（字级别音素边界）接口	支持儿化音	声音品质
小仙	xiaoxian	亲切女声	直播场景	支持中文及中英文混合场景	8K/16K	是	是	标准版
悦儿	yuer	儿童剧女声	通用场景	仅支持纯中文场景	8K/16K	是	否	标准版
猫小美	maoxiaomei	活力女声	直播场景	支持中文及中英文混合场景	8K/16K	是	是	标准版
知飞	zhifei	激昂解说	超高清场景	支持中文及中英文混合场景	8K/16K	是	否	精品版
知伦	zhilun	悬疑解说	超高清场景	支持中文及中英文混合场景	8K/16K	是	否	精品版
艾飞	aifei	激昂解说	直播场景	支持中文及中英文混合场景	8K/16K	是	是	标准版
亚群	yaqun	卖场广播	直播场景	支持中文及中英文混合场景	8K/16K	是	是	标准版
巧薇	qiaowei	卖场广播	直播场景	支持中文及中英文混合场景	8K/16K	是	是	标准版
大虎	dahu	东北话男声	方言场景	支持中文及中英文混合场景	8K/16K	是	是	标准版

名称	voice参数值	类型	适用场景	支持语言	支持采样率（Hz）	支持时间戳（字级别音素边界）接口	支持儿化音	声音品质
ava	ava	美语女声	英文场景	仅支持纯英文场景	8K/16K	是	否	标准版
艾伦	ailun	悬疑解说	直播场景	支持中文及中英文混合场景	8K/16K	是	是	标准版
杰力豆	jielidou	治愈童声	童声场景	仅支持纯中文场景	8K/16K	是	是	标准版
老铁	laotie	东北老铁	直播场景	仅支持纯中文场景	8K/16K	是	是	标准版
老妹	laomei	吆喝女声	直播场景	仅支持纯中文场景	8K/16K	是	是	标准版
艾侃	aikan	天津话男声	方言场景	仅支持纯中文场景	8K/16K	是	是	标准版
Tala	tala	菲律宾语女声	多语种场景	仅支持菲律宾语场景	8K/16K	否	否	标准版
Tien	tien	越南语女声	多语种场景	仅支持越南语场景	8K/16K	否	否	标准版
Becca	becca	美语客服女声	美式英语	支持纯英语场景	8K/16K	否	否	标准版

多情感声音支持说明

多情感声音支持的情感如下表所示，每个音色支持的情感分类不完全相同，主要包括以下几种：neutral（无情感）、happy（开心）、angry（生气）、sad（悲伤）、fear（害怕）、hate（厌恶）、surprise（惊讶）、arousal（激动）等。

音色名	voice参数值	情感分类（emotion category）
知米_多情感	zhimi_emo	angry, fear, happy, hate, neutral, sad, surprise
知燕_多情感	zhiyan_emo	neutral，happy，angry，sad，fear，hate，surprise，arousal
知贝_多情感	zhibei_emo	neutral，happy，angry，sad，fear，hate，surprise
知甜_多情感	zhitian_emo	neutral，happy，angry，sad，fear，hate，surprise

就近地域智能接入

语音合成支持就近地域智能接入，域名为 `nls-gateway.aliyuncs.com` 。

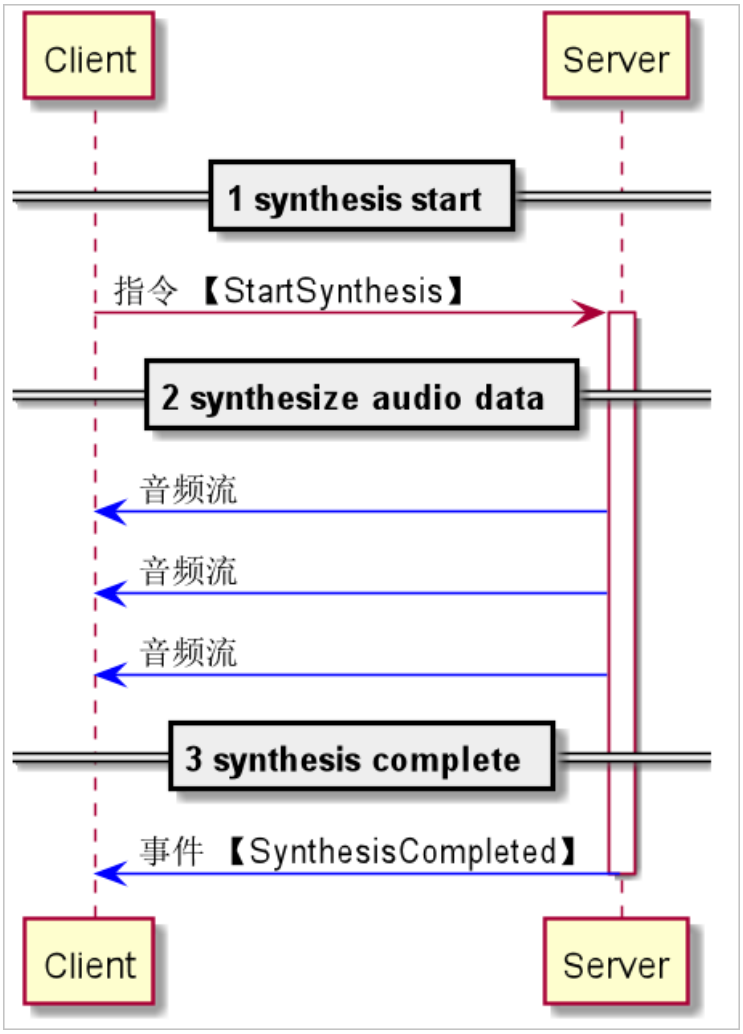
推荐终端用户使用就近地域接入域名。根据调用接口时客户端所在的地理位置，系统会自动解析到最近的某个具体地域的服务器。例如在北京地域发起请求，系统会自动解析到北京地域的服务器，与指定域名 `nls-gateway-cn-beijing.aliyuncs.com` 的实现效果一致。

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` • 北京：`wss://nls-gateway-cn-beijing.aliyuncs.com/ws/v1` • 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。 说明 • 使用内网访问方式，将不产生ECS实例的公网流量费用。 • 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` • 北京：`ws://nls-gateway-cn-beijing-internal.aliyuncs.com:80/ws/v1` • 深圳：`ws://nls-gateway-cn-shenzhen-internal.aliyuncs.com:80/ws/v1`

交互流程



说明

- 上图不包含RESTful API的交互流程，关于RESTful API的交互流程图请参见RESTful API。
- 服务端的响应除了音频流之外，都会在返回信息的header包含本次识别任务的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

   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。

2. 开始合成

   客户端发起语音合成请求，在请求消息中进行参数设置，各参数通过SDK中SpeechSynthesizer对象的set方法设置，含义如下。

参数	类型	是否必需	说明
appkey	String	是	管控台创建的项目Appkey。

参数	类型	是否必需	说明
text	String	是	待合成文本，文本内容必须采用 `UTF-8` 编码，长度不超过300个字符（英文字母之间需要添加空格）。 说明 调用某音色的多情感内容，需要在text中加上ssml-emotion标签，具体请参见<emotion>。
voice	String	否	发音人，默认是 `xiaoyun` 。
format	String	否	音频编码格式，默认值：PCM。支持PCM/WAV/MP3格式。
sample_rate	Integer	否	音频采样率，默认值：16000 Hz。
volume	Integer	否	音量，取值范围：0～100。默认值：50。

参数	类型	是否必需	说明
speech_rate	Integer	否	语速，取值范围：-500～500，默认值：0。 [-500, 0, 500] 对应的语速倍速区间为 [0.5, 1.0, 2.0]。 ○ -500表示默认语速的0.5倍速。 ○ 0表示默认语速的1倍速。1倍速是指模型默认输出的合成语速，语速会依据每一个发音人略有不同，大概每秒钟4个字左右。 ○ 500表示默认语速的2倍速。 计算方法如下： ○ 0.8倍速（1-1/0.8)/0.002 = -125 ○ 1.2倍速（1-1/1.2)/0.001 = 166 说明 ○ 小于1倍速时，使用0.002系数。 ○ 大于1倍速时，使用0.001系数。 实际算法结果取近似值。
pitch_rate	Integer	否	语调，取值范围：-500～500，默认值：0。

3. 接收合成数据

   服务端返回合成的语音二进制数据，SDK接收并处理二进制数据。

4. 结束合成

   语音合成完毕，服务端发送合成完毕事件通知，举例如下。

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

> **说明**
>
> 文档示例将合成的音频保存在文件中，如果您需要播放音频且对实时性要求较高，建议使用流式播放，即边接收语音数据边播放，减少延时。

服务状态码

服务的每一次响应都包含status字段，即服务状态码，各状态码含义如下。

- 通用错误：

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。
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。
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。

- 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。
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。	检查应用Appkey是否正确，是否与令牌归属同一个账号。

- TTS（Text to Speech）错误：

错误码	原因	解决办法
41020001	参数错误。	检查是否传递了正确的参数。
51020001	TTS服务端错误。	如果偶现可以忽略，重复出现请提交工单。

2.RESTful API

语音合成RESTful API支持HTTPS GET和POST两种方法的请求，将待合成的文本上传到服务端，服务端返回文本的语音合成结果，开发者需要保证在语音合成结果返回之前连接不中断。

功能介绍

支持如下设置：

- PCM、WAV、MP3音频格式。
- 8000 Hz、16000 Hz采样率。
- 多种发音人。
- 语速、语调、音量。

> **注意**
>
> - 随着TTS合成效果不断提升，算法的复杂度也越来越高，对您而言，可能会遇到合成耗时变长的情况。因此我们建议您使用流式合成机制。本文档及SDK示例中有相关流式处理示例代码可供参考。
> - 单次调用传入文本不能超过300个字符，超过的字符会被截断。对于更长文本的合成，请参考SDK中的长文本切分及拼接示例。
> - 下载nls-restful-java-demo.zip。

前提条件

- 已准备项目Appkey，详情请参见创建项目。
- 已获取Access Token，详情请参见获取Token。

服务地址

访问类型	说明	URL	Host

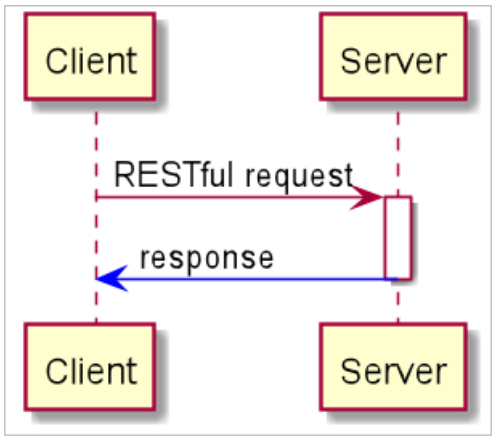
访问类型	说明	URL	Host
外网访问（默认上海地域）	所有服务器均可使用外网访问URL。	• 上海：`nls-gateway-cn-shanghai.aliyuncs.com/stream/v1/tts` • 北京：`nls-gateway-cn-beijing.aliyuncs.com/stream/v1/tts` • 深圳：`nls-gateway-cn-shenzhen.aliyuncs.com/stream/v1/tts`	• 上海：`nls-gateway-cn-shanghai.aliyuncs.com` • 北京：`nls-gateway-cn-beijing.aliyuncs.com` • 深圳：`nls-gateway-cn-shenzhen.aliyuncs.com`
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。 说明 • 使用内网访问方式，将不产生ECS实例的公网流量费用。 • 关于ECS的网络类型请参见网络类型。	• 上海：`nls-gateway-cn-shanghai-internal.aliyuncs.com/stream/v1/tts` • 北京：`nls-gateway-cn-beijing-internal.aliyuncs.com/stream/v1/tts` • 深圳：`nls-gateway-cn-shenzhen-internal.aliyuncs.com/stream/v1/tts`	• 上海：`nls-gateway-cn-shanghai-internal.aliyuncs.com` • 北京：`nls-gateway-cn-beijing-internal.aliyuncs.com` • 深圳：`nls-gateway-cn-shenzhen-internal.aliyuncs.com`

注意

以下将以使用外网访问URL的方式进行介绍。如果您使用的是阿里云上海、北京、深圳ECS，并需要使用内网访问URL，则需要使用HTTP协议，并替换外网访问的URL和Host。以下示例中除了Python，均支持HTTP和HTTPS协议，在使用Python示例时请阅读其注意事项。

交互流程

客户端向服务端发送携带文本内容的HTTPS GET方法或POST方法的请求，服务端返回携带合成语音数据的HTTP响应。



说明

服务端的错误响应都会在返回信息中包含表示本次合成任务的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。

请求参数

语音合成的请求参数如下表所示。

- 如果使用HTTPS GET方法的请求，需要在HTTPS的URL请求参数中设置这些参数。
- 如果使用HTTPS POST方法的请求，需要在HTTPS的请求体（Body）中设置这些参数。

名称	类型	是否必选	描述
appkey	String	是	项目appkey。
text	String	是	待合成的文本，需要为 `UTF-8` 编码。使用GET方法，需要再采用RFC 3986规范进行urlencode编码；使用POST方法不需要urlencode编码。

名称	类型	是否必选	描述
token	String	否	若不设置token参数，需要在HTTP Headers中设置 `X-NLS-Token` 字段来指定Token。
format	String	否	音频编码格式，支持pcm/wav/mp3格式。默认值：pcm。
sample_rate	Integer	否	音频采样率，支持16000 Hz和8000 Hz，默认值：16000 Hz。
voice	String	否	发音人，默认值：xiaoyun。更多发音人请参见接口说明。
volume	Integer	否	音量，取值范围：0~100，默认值：50。
speech_rate	Integer	否	语速，取值范围：-500~500，默认值：0。
pitch_rate	Integer	否	语调，取值范围：-500~500，默认值：0。

GET方法上传文本

一个完整的语音合成RESTful API GET方法的请求包含以下要素：

- URL

协议	URL	方法
HTTPS	nls-gateway-cn-shanghai.aliyuncs.com/stream/v1/tts	GET

- 请求参数

  参见上述请求参数。由URL和请求参数组成的完整请求链接如下所示，在浏览器中打开该链接可直接获取语音合成结果：

```
# text文本："今天是周一，天气挺好的。"
https://nls-gateway-cn-shanghai.aliyuncs.com/stream/v1/tts?appkey=${您的appkey}&token=${您的token}&text=%E4%BB%8A%E5%A4%A9%E6%98%AF%E5%91%A8%E4%B8%80%EF%BC%8C%E5%A4%A9%E6%B0%94%E6%8C%BA%E5%A5%BD%E7%9A%84%E3%80%82&format=wav&sample_rate=16000
```

- HTTPS GET请求头部

名称	类型	是否必选	描述
X-NLS-Token	String	否	服务鉴权Token。

注意

- 服务鉴权Token参数有如下两种设置方式：
  - （推荐）在请求参数 `token` 中设置
  - 在HTTPS Headers的 `X-NLS-Token` 字段设置
- 参数 `text` 必须采用 `UTF-8` 编码，再采用RFC 3986规范进行urlencode编码。如加号 `+` 编码为 `%2B` ，星号 `*` 编码为 `%2A` ， `%7E` 编码为 `~` 。

POST方法上传文本

一个完整的语音合成RESTful API POST请求包含以下要素：

- URL

协议	URL	方法
HTTPS	nls-gateway-cn-shanghai.aliyuncs.com/stream/v1/tts	POST

- HTTPS POST请求头部

名称	类型	是否必选	描述
Content-Type	String	是	必须为“application/json”。表明HTTP请求体的内容为JSON格式字符串。
X-NLS-Token	String	否	服务鉴权Token。

名称	类型	是否必选	描述
Content-Length	long	否	HTTP请求体中内容的长度。

- HTTPS POST请求体

  HTTPS POST请求体由请求参数组成JSON格式的字符串组成，因此在HTTPS POST请求头部中的Content-Type必须设置为"application/json"。示例如下：

```
{
  "appkey":"31f932fb",
  "text":"今天是周一，天气挺好的。",
  "token":"450343c793aaaaaa****",
  "format":"wav"
}
```

> **注意**
>
> - 服务鉴权Token参数有如下两种设置方式：
> - 推荐在请求体通过参数 `token` 设置
> - 在HTTPS Headers的 `X-NLS-Token` 字段设置
> - 使用POST方法的请求，请求体中的请求参数 `text` 必须采用 `UTF-8` 编码，但是不进行urlencode编码，注意与GET方法请求的区别。

响应结果

使用HTTPS GET方法和HTTPS POST方法请求的响应是相同的，响应结果都包含在HTTPS的响应体中。响应结果的成功/失败通过HTTPS Headers的 `Content-Type` 字段来区分：

- 成功响应
  - HTTPS Headers的 `Content-Type` 字段内容为 `audio/mpeg` ，表示合成成功，合成的语音数据在响应体中。
  - HTTPS Header的 `X-NLS-RequestId` 字段内容为请求任务的task_id。
  - 响应内容为合成音频的二进制数据。
- 失败响应
  - HTTPS Headers没有 `Content-Type` 字段，或者 `Content-Type` 字段内容为 `application/json` ，表示合成失败，错误信息在响应体中。

- HTTPS Headers的 `X-NLS-RequestId` 字段内容为请求任务的task_id。
- 响应体内容为错误信息，以JSON格式的字符串表示。如下所示：

```
{
    "task_id":"8f95d0b9b6e948bc98e8d0ce64b0****",
    "result":"",
    "status":40000000,
    "message":"Gateway:CLIENT_ERROR:in post data, json format illegal"
}
```

- 错误信息字段如下表

名称	类型	描述
task_id	String	32位请求任务ID，请记录该值，用于排查错误。
result	String	服务结果
status	Integer	服务状态码
message	String	服务状态描述

服务状态码

服务状态码	服务状态描述	解决办法
20000000	请求成功	无。
40000000	默认的客户端错误码	查看错误消息或提交工单。
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。
40000003	无效的参数	检查参数值设置是否合理。
40000004	空闲超时	确认是否长时间没有发送数据到服务端。

服务状态码	服务状态描述	解决办法
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。
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GRPC调用错误	如果偶现可以忽略，重复出现请提交工单。

Java示例

依赖文件内容如下：

```
<dependency>
    <groupId>com.squareup.okhttp3</groupId>
    <artifactId>okhttp</artifactId>
    <version>3.9.1</version>
</dependency>
<!-- http://mvnrepository.com/artifact/com.alibaba/fastjson -->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.42</version>
</dependency>
<dependency>
    <groupId>org.asynchttpclient</groupId>
    <artifactId>async-http-client</artifactId>
    <version>2.5.4</version>
</dependency>
```

示例代码如下：

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;
import com.alibaba.fastjson.JSONObject;
import okhttp3.MediaType;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.RequestBody;
import okhttp3.Response;
public class SpeechSynthesizerRestfulDemo {
    private String accessToken;
    private String appkey;
    public SpeechSynthesizerRestfulDemo(String appkey, String token) {
        this.appkey = appkey;
        this.accessToken = token;
```

```
        this.accessToken = token;
    }
    /**
     * HTTPS GET请求
     */
    public void processGETRequet(String text, String audioSaveFile, String format, int samp
leRate, String voice) {
        /**
         * 设置HTTPS GET请求：
         * 1.使用HTTPS协议
         * 2.语音识别服务域名：nls-gateway-cn-shanghai.aliyuncs.com
         * 3.语音识别接口请求路径：/stream/v1/tts
         * 4.设置必须请求参数：appkey、token、text、format、sample_rate
         * 5.设置可选请求参数：voice、volume、speech_rate、pitch_rate
         */
        String url = "https://nls-gateway-cn-shanghai.aliyuncs.com/stream/v1/tts";
        url = url + "?appkey=" + appkey;
        url = url + "&token=" + accessToken;
        url = url + "&text=" + text;
        url = url + "&format=" + format;
        url = url + "&voice=" + voice;
        url = url + "&sample_rate=" + String.valueOf(sampleRate);
        // voice 发音人，可选，默认是xiaoyun。
        // url = url + "&voice=" + "xiaoyun";
        // volume 音量，范围是0~100，可选，默认50。
        // url = url + "&volume=" + String.valueOf(50);
        // speech_rate 语速，范围是-500~500，可选，默认是0。
        // url = url + "&speech_rate=" + String.valueOf(0);
        // pitch_rate 语调，范围是-500~500，可选，默认是0。
        // url = url + "&pitch_rate=" + String.valueOf(0);
        System.out.println("URL: " + url);
        /**
         * 发送HTTPS GET请求，处理服务端的响应。
         */
        Request request = new Request.Builder().url(url).get().build();
        try {
            long start = System.currentTimeMillis();
            OkHttpClient client = new OkHttpClient();
            Response response = client.newCall(request).execute();
            System.out.println("total latency :" + (System.currentTimeMillis() - start) + "
ms");
            System.out.println(response.headers().toString());
            String contentType = response.header("Content-Type");
            if ("audio/mpeg".equals(contentType)) {
                File f = new File(audioSaveFile);
                FileOutputStream fout = new FileOutputStream(f);
                fout.write(response.body().bytes());
                fout.close();
                System.out.println("The GET request succeed!");
            }
            else {
                // ContentType 为 null 或者为 "application/json"
                String errorMessage = response.body().string();
                System.out.println("The GET request failed: " + errorMessage);
            }
```

```
            response.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    /**
     * HTTPS POST请求
     */
    public void processPOSTRequest(String text, String audioSaveFile, String format, int sa
mpleRate, String voice) {
        /**
         * 设置HTTPS POST请求：
         * 1.使用HTTPS协议
         * 2.语音合成服务域名：nls-gateway-cn-shanghai.aliyuncs.com
         * 3.语音合成接口请求路径：/stream/v1/tts
         * 4.设置必须请求参数：appkey、token、text、format、sample_rate
         * 5.设置可选请求参数：voice、volume、speech_rate、pitch_rate
         */
        String url = "https://nls-gateway-cn-shanghai.aliyuncs.com/stream/v1/tts";
        JSONObject taskObject = new JSONObject();
        taskObject.put("appkey", appkey);
        taskObject.put("token", accessToken);
        taskObject.put("text", text);
        taskObject.put("format", format);
        taskObject.put("voice", voice);
        taskObject.put("sample_rate", sampleRate);
        // voice 发音人，可选，默认是xiaoyun。
        // taskObject.put("voice", "xiaoyun");
        // volume 音量，范围是0~100，可选，默认50。
        // taskObject.put("volume", 50);
        // speech_rate 语速，范围是-500~500，可选，默认是0。
        // taskObject.put("speech_rate", 0);
        // pitch_rate 语调，范围是-500~500，可选，默认是0。
        // taskObject.put("pitch_rate", 0);
        String bodyContent = taskObject.toJSONString();
        System.out.println("POST Body Content: " + bodyContent);
        RequestBody reqBody = RequestBody.create(MediaType.parse("application/json"), bodyC
ontent);
        Request request = new Request.Builder()
            .url(url)
            .header("Content-Type", "application/json")
            .post(reqBody)
            .build();
        try {
            OkHttpClient client = new OkHttpClient();
            Response response = client.newCall(request).execute();
            String contentType = response.header("Content-Type");
            if ("audio/mpeg".equals(contentType)) {
                File f = new File(audioSaveFile);
                FileOutputStream fout = new FileOutputStream(f);
                fout.write(response.body().bytes());
                fout.close();
                System.out.println("The POST request succeed!");
            }
```

```
            else {
                // ContentType 为 null 或者为 "application/json"
                String errorMessage = response.body().string();
                System.out.println("The POST request failed: " + errorMessage);
            }
            response.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public static void main(String[] args) {
        if (args.length < 2) {
            System.err.println("SpeechSynthesizerRestfulDemo need params: <token> <app-key>
");
            System.exit(-1);
        }
        String token = args[0];
        String appkey = args[1];
        SpeechSynthesizerRestfulDemo demo = new SpeechSynthesizerRestfulDemo(appkey, token)
;
        String text = "今天是周一，天气挺好的。";
        // 采用RFC 3986规范进行urlencode编码。
        String textUrlEncode = text;
        try {
            textUrlEncode = URLEncoder.encode(textUrlEncode, "UTF-8")
                .replace("+", "%20")
                .replace("*", "%2A")
                .replace("%7E", "~");
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        System.out.println(textUrlEncode);
        String audioSaveFile = "syAudio.wav";
        String format = "wav";
        int sampleRate = 16000;
        demo.processGETRequet(textUrlEncode, audioSaveFile, format, sampleRate, "siyue");
        //demo.processPOSTRequest(text, audioSaveFile, format, sampleRate, "siyue");
        System.out.println("### Game Over ###");
    }
}
```

Java（流式合成）示例代码如下：

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;
import java.util.concurrent.CountDownLatch;
import io.netty.handler.codec.http.HttpHeaders;
import org.asynchttpclient.AsyncHandler;
import org.asynchttpclient.AsyncHttpClient;
import org.asynchttpclient.AsyncHttpClientConfig;
import org.asynchttpclient.DefaultAsyncHttpClient;
import org.asynchttpclient.DefaultAsyncHttpClientConfig;
```

```
import org.asynchttpclient.HttpResponseBodyPart;
import org.asynchttpclient.HttpResponseStatus;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
/**
 * 此示例演示了
 * 1. TTS的RESTFul接口调用。
 * 2. 启用HTTP chunked机制的处理方式（流式返回）。
 */
public class SpeechSynthesizerRestfulChunkedDemo {
    private static Logger logger = LoggerFactory.getLogger(SpeechSynthesizerRestfulChunkedD
emo.class);
    private String accessToken;
    private String appkey;
    public SpeechSynthesizerRestfulChunkedDemo(String appkey, String token) {
        this.appkey = appkey;
        this.accessToken = token;
    }
    public void processGETRequet(String text, String audioSaveFile, String format, int samp
leRate, String voice, boolean chunked) {
        /**
         * 设置HTTPS GET请求：
         * 1.使用HTTPS协议
         * 2.语音识别服务域名：nls-gateway-cn-shanghai.aliyuncs.com
         * 3.语音识别接口请求路径：/stream/v1/tts
         * 4.设置必须请求参数：appkey、token、text、format、sample_rate
         * 5.设置可选请求参数：voice、volume、speech_rate、pitch_rate
         */
        String url = "https://nls-gateway-cn-shanghai.aliyuncs.com/stream/v1/tts";
        url = url + "?appkey=" + appkey;
        url = url + "&token=" + accessToken;
        url = url + "&text=" + text;
        url = url + "&format=" + format;
        url = url + "&voice=" + voice;
        url = url + "&sample_rate=" + String.valueOf(sampleRate);
        System.out.println("URL: " + url);
        try {
            AsyncHttpClientConfig config = new DefaultAsyncHttpClientConfig.Builder()
                .setConnectTimeout(3000)
                .setKeepAlive(true)
                .setReadTimeout(10000)
                .setRequestTimeout(50000)
                .setMaxConnections(1000)
                .setMaxConnectionsPerHost(200)
                .setPooledConnectionIdleTimeout(-1)
                .build();
            AsyncHttpClient httpClient = new DefaultAsyncHttpClient(config);
            CountDownLatch latch = new CountDownLatch(1);
            AsyncHandler<org.asynchttpclient.Response> handler = new AsyncHandler<org.async
httpclient.Response>() {
                FileOutputStream outs;
                boolean firstRecvBinary = true;
                long startTime = System.currentTimeMillis();
                int httpCode = 200;
```

```
                @Override
                public State onStatusReceived(HttpResponseStatus httpResponseStatus) throws
Exception {
                    logger.info("onStatusReceived status {}", httpResponseStatus);
                    httpCode = httpResponseStatus.getStatusCode();
                    if (httpResponseStatus.getStatusCode() != 200) {
                        logger.error("request error " +  httpResponseStatus.toString());
                    }
                    return null;
                }
                @Override
                public State onHeadersReceived(HttpHeaders httpHeaders) throws Exception {
                    outs = new FileOutputStream(new File("tts.wav"));
                    return null;
                }
                @Override
                public State onBodyPartReceived(HttpResponseBodyPart httpResponseBodyPart)
throws Exception {
                    // 注意：此处一旦接收到数据流，即可向用户播放或者用于其他处理，以提升响应速度。
                    // 注意：请不要在此回调接口中执行耗时操作，可以以异步或者队列形式将二进制TTS语音
流推送到另一线程中。
                    logger.info("onBodyPartReceived " + httpResponseBodyPart.getBodyPartByt
es().toString());
                    if(httpCode != 200) {
                        System.err.write(httpResponseBodyPart.getBodyPartBytes());
                    }
                    if (firstRecvBinary) {
                        firstRecvBinary = false;
                        // 统计第一包数据的接收延迟。实际上接收到第一包数据后就可以进行业务处理了，
比如播放或者发送给调用方。注意：这里的首包延迟也包括了网络建立链接的时间。
                        logger.info("tts first latency " + (System.currentTimeMillis() - st
artTime) + " ms");
                    }
                    // 此处以将语音流保存到文件为例。
                    outs.write(httpResponseBodyPart.getBodyPartBytes());
                    return null;
                }
                @Override
                public void onThrowable(Throwable throwable) {
                    logger.error("throwable {}", throwable);
                    latch.countDown();
                }
                @Override
                public org.asynchttpclient.Response onCompleted() throws Exception {
                    logger.info("completed");
                    logger.info("tts total latency " + (System.currentTimeMillis() - startT
ime) + " ms");
                    outs.close();
                    latch.countDown();
                    return null;
                }
            };
            httpClient.prepareGet(url).execute(handler);
            // 等待合成完成
```

```
            latch.await();
            httpClient.close();
        }catch (Exception e) {
        }
    }
    public static void main(String[] args) {
        if (args.length < 2) {
            System.err.println("SpeechSynthesizerRestfulDemo need params: <token> <app-key>
");
            System.exit(-1);
        }
        String token = args[0];
        String appkey = args[1];
        SpeechSynthesizerRestfulChunkedDemo demo = new SpeechSynthesizerRestfulChunkedDemo(
appkey, token);
        String text = "我家的后面有一个很大的园，相传叫作百草园。现在是早已并屋子一起卖给朱文公的子孙了，连那最末次的相见也已经隔了七八年，其中似乎确凿只有一些野草；但那时却是我的乐园。";
        // 采用RFC 3986规范进行urlencode编码。
        String textUrlEncode = text;
        try {
            textUrlEncode = URLEncoder.encode(textUrlEncode, "UTF-8")
                .replace("+", "%20")
                .replace("*", "%2A")
                .replace("%7E", "~");
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        System.out.println(textUrlEncode);
        String audioSaveFile = "syAudio.wav";
        String format = "wav";
        int sampleRate = 16000;
        // 最后一个参数为true表示使用http chunked机制。
        demo.processGETRequet(textUrlEncode, audioSaveFile, format, sampleRate, "aixia", tr
ue);
        System.out.println("### Game Over ###");
    }
}
```

C++示例

> **说明**
>
> - 下载C++示例。
>
> C++示例使用第三方函数库curl处理HTTPS的请求和响应，使用jsoncpp处理POST请求体的JSON字符串。
> - Linux环境下，运行环境最低要求：Glibc 2.5及以上， Gcc4或Ccc5。
> - Windows下需解压lib目录下的windows.zip库编译使用。

示例目录说明如下：

- CMakeLists.txt：示例工程的CMakeList文件。

- demo：示例文件。

文件名	描述
restfulTtsDemo.cpp	语音合成RESTful API示例。

- include

目录名	描述
curl	curl库头文件目录。
json	jsoncpp库头文件目录。

- lib：包含curl、jsoncpp动态库。

  根据平台不同，使用如下版本软件加载库文件：

  - linux（Glibc：2.5及以上，Gcc4或Gcc5）
  - windows（VS2013、VS2015）

- readme.txt：说明文件。
- release.log：更新记录。
- version：版本号。
- build.sh：示例编译脚本。

编译运行操作步骤：

假设示例文件已解压至 `path/to` 路径下，在Linux终端依次执行如下命令编译运行程序。

- 支持Cmake：

  i. 确认本地系统已安装Cmake 2.4及以上版本。

  ii. `cd path/to/sdk/lib` 。

  iii. `tar -zxvpf linux.tar.gz` 。

  iv. `cd path/to/sdk` 。

  v. `./build.sh` 。

  vi. `cd path/to/sdk/demo` 。

  vii. `./restfulTtsDemo <your-token> <your-appkey>` 。

- 不支持Cmake：

i. `cd path/to/sdk/lib` 。

ii. `tar -zxvpf linux.tar.gz` 。

iii. `cd path/to/sdk/demo` 。

iv. `g++ -o restfulTtsDemo restfulTtsDemo.cpp -I path/to/sdk/include -L path/to/sdk/lib/linux -ljsoncpp -lssl -lcrypto -lcurl -D_GLIBCXX_USE_CXX11_ABI=0` 。

v. `export LD_LIBRARY_PATH=path/to/sdk/lib/linux/` 。

vi. `./restfulTtsDemo <your-token> <your-appkey>` 。

示例代码如下：

```
#ifdef _WIN32
#include <Windows.h>
#endif
#include <iostream>
#include <string>
#include <map>
#include <fstream>
#include <sstream>
#include "curl/curl.h"
#include "json/json.h"
using namespace std;
#ifdef _WIN32
string GBKToUTF8(const string &strGBK) {
    string strOutUTF8 = "";
    WCHAR * str1;
    int n = MultiByteToWideChar(CP_ACP, 0, strGBK.c_str(), -1, NULL, 0);
    str1 = new WCHAR[n];
    MultiByteToWideChar(CP_ACP, 0, strGBK.c_str(), -1, str1, n);
    n = WideCharToMultiByte(CP_UTF8, 0, str1, -1, NULL, 0, NULL, NULL);
    char * str2 = new char[n];
    WideCharToMultiByte(CP_UTF8, 0, str1, -1, str2, n, NULL, NULL);
    strOutUTF8 = str2;
    delete[] str1;
    str1 = NULL;
    delete[] str2;
    str2 = NULL;
    return strOutUTF8;
}
#endif
void stringReplace(string& src, const string& s1, const string& s2) {
    string::size_type pos = 0;
    while ((pos = src.find(s1, pos)) != string::npos) {
        src.replace(pos, s1.length(), s2);
        pos += s2.length();
    }
}
string urlEncode(const string& src) {
```

```
    CURL* curl = curl_easy_init();
    char* output = curl_easy_escape(curl, src.c_str(), src.size());
    string result(output);
    curl_free(output);
    curl_easy_cleanup(curl);
    return result;
}
size_t responseHeadersCallback(void* ptr, size_t size, size_t nmemb, void* userdata)
{
    map<string, string> *headers = (map<string, string>*)userdata;
    string line((char*)ptr);
    string::size_type pos = line.find(':');
    if (pos != line.npos)
    {
        string name = line.substr(0, pos);
        string value = line.substr(pos + 2);
        size_t p = 0;
        if ((p = value.rfind('\r')) != value.npos) {
            value = value.substr(0, p);
        }
        headers->insert(make_pair(name, value));
    }
    return size * nmemb;
}
size_t responseBodyCallback(void* ptr, size_t size, size_t nmemb, void* userData) {
    size_t len = size * nmemb;
    char* pBuf = (char*)ptr;
    string* bodyContent = (string*)userData;
    (*bodyContent).append(string(pBuf, pBuf + len));
    return len;
}
int processGETRequest(string appKey, string token, string text,
                      string audioSaveFile, string format, int sampleRate) {
    CURL* curl = NULL;
    CURLcode res;
    curl = curl_easy_init();
    if (curl == NULL) {
        return -1;
    }
    string url = "https://nls-gateway-cn-shanghai.aliyuncs.com/stream/v1/tts";
    /**
     * 设置HTTPS URL请求参数
     */
    ostringstream oss;
    oss << url;
    oss << "?appkey=" << appKey;
    oss << "&token=" << token;
    oss << "&text=" << text;
    oss << "&format=" << format;
    oss << "&sample_rate=" << sampleRate;
    // voice 发音人，可选，默认是xiaoyun。
    // oss << "&voice=" << "xiaoyun";
    // volume 音量，范围是0~100，可选，默认50。
    // oss << "&volume=" << 50;
```

```
    // speech_rate 语速，范围是-500~500，可选，默认是0。
    // oss << "&speech_rate=" << 0;
    // pitch_rate 语调，范围是-500~500，可选，默认是0。
    // oss << "&pitch_rate=" << 0;
    string request = oss.str();
    cout << request << endl;
    curl_easy_setopt(curl, CURLOPT_URL, request.c_str());
    /**
     * 设置获取响应的HTTPS Headers回调函数
     */
    map<string, string> responseHeaders;
    curl_easy_setopt(curl, CURLOPT_HEADERFUNCTION, responseHeadersCallback);
    curl_easy_setopt(curl, CURLOPT_HEADERDATA, &responseHeaders);
    /**
     * 设置获取响应的HTTPS Body回调函数
     */
    string bodyContent = "";
    curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, responseBodyCallback);
    curl_easy_setopt(curl, CURLOPT_WRITEDATA, &bodyContent);
    /**
     * 发送HTTPS GET请求
     */
    res = curl_easy_perform(curl);
    /**
     * 释放资源
     */
    curl_easy_cleanup(curl);
    if (res != CURLE_OK) {
        cerr << "curl_easy_perform failed: " << curl_easy_strerror(res) << endl;
        return -1;
    }
    /**
     * 处理服务端返回的响应
     */
    map<string, string>::iterator it = responseHeaders.find("Content-Type");
    if (it != responseHeaders.end() && it->second.compare("audio/mpeg") == 0) {
        ofstream fs;
        fs.open(audioSaveFile.c_str(), ios::out | ios::binary);
        if (!fs.is_open()) {
            cout << "The audio save file can not open!";
            return -1;
        }
        fs.write(bodyContent.c_str(), bodyContent.size());
        fs.close();
        cout << "The GET request succeed!" << endl;
    }
    else {
        cout << "The GET request failed: " + bodyContent << endl;
        return -1;
    }
    return 0;
}
int processPOSTRequest(string appKey, string token, string text,
                       string audioSaveFile, string format, int sampleRate) {
```

```
    CURL* curl = NULL;
    CURLcode res;
    curl = curl_easy_init();
    if (curl == NULL) {
        return -1;
    }
    string url = "https://nls-gateway-cn-shanghai.aliyuncs.com/stream/v1/tts";
    /**
     * 设置HTTPS POST URL
     */
    curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
    curl_easy_setopt(curl, CURLOPT_POST, 1L);
    /**
     * 设置HTTPS POST请求头部
     */
    struct curl_slist* headers = NULL;
    // Content-Type
    headers = curl_slist_append(headers, "Content-Type:application/json");
    curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headers);
    /**
     * 设置HTTPS POST请求体
     */
    Json::Value root;
    Json::FastWriter writer;
    root["appkey"] = appKey;
    root["token"] = token;
    root["text"] = text;
    root["format"] = format;
    root["sample_rate"] = sampleRate;
    // voice 发音人，可选，默认是xiaoyun。
    // root["voice"] = "xiaoyun";
    // volume 音量，范围是0~100，可选，默认50。
    // root["volume"] = 50;
    // speech_rate 语速，范围是-500~500，可选，默认是0。
    // root["speech_rate"] = 0;
    // pitch_rate 语调，范围是-500~500，可选，默认是0。
    // root["pitch_rate"] = 0;
    string task = writer.write(root);
    cout << "POST request Body: " << task << endl;
    curl_easy_setopt(curl, CURLOPT_POSTFIELDS, task.c_str());
    curl_easy_setopt(curl, CURLOPT_POSTFIELDSIZE, task.length());
    /**
     * 设置获取响应的HTTPS Headers回调函数
     */
    map<string, string> responseHeaders;
    curl_easy_setopt(curl, CURLOPT_HEADERFUNCTION, responseHeadersCallback);
    curl_easy_setopt(curl, CURLOPT_HEADERDATA, &responseHeaders);
    /**
     * 设置获取响应的HTTPS Body回调函数
     */
    string bodyContent = "";
    curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, responseBodyCallback);
    curl_easy_setopt(curl, CURLOPT_WRITEDATA, &bodyContent);
    /**
     * 发送HTTPS POST请求
```

```
     * 发送HTTPS POST请求
     */
    res = curl_easy_perform(curl);
    /**
     * 释放资源
     */
    curl_slist_free_all(headers);
    curl_easy_cleanup(curl);
    if (res != CURLE_OK) {
        cerr << "curl_easy_perform failed: " << curl_easy_strerror(res) << endl;
        return -1;
    }
    /**
     * 处理服务端返回的响应
     */
    map<string, string>::iterator it = responseHeaders.find("Content-Type");
    if (it != responseHeaders.end() && it->second.compare("audio/mpeg") == 0) {
        ofstream fs;
        fs.open(audioSaveFile.c_str(), ios::out | ios::binary);
        if (!fs.is_open()) {
            cout << "The audio save file can not open!";
            return -1;
        }
        fs.write(bodyContent.c_str(), bodyContent.size());
        fs.close();
        cout << "The POST request succeed!" << endl;
    }
    else {
        cout << "The POST request failed: " + bodyContent << endl;
        return -1;
    }
    return 0;
}
int main(int argc, char* argv[]) {
    if (argc < 3) {
        cerr << "params is not valid. Usage: ./demo your_token your_appkey" << endl;
        return -1;
    }
    string token = argv[1];
    string appKey = argv[2];
    string text = "今天是周一，天气挺好的。";
#ifdef _WIN32
    text = GBKToUTF8(text);
#endif
    string textUrlEncode = urlEncode(text);
    stringReplace(textUrlEncode, "+", "%20");
    stringReplace(textUrlEncode, "*", "%2A");
    stringReplace(textUrlEncode, "%7E", "~");
    string audioSaveFile = "syAudio.wav";
    string format = "wav";
    int sampleRate = 16000;
    // 全局只初始化一次。
    curl_global_init(CURL_GLOBAL_ALL);
    processGETRequest(appKey, token, textUrlEncode, audioSaveFile, format, sampleRate);
    //processPOSTRequest(appKey, token, text, audioSaveFile, format, sampleRate);
```

```
    //processTtsRequest(appKey, token, text, audioSaveFile, format, sampleRate);
    curl_global_cleanup();
    return 0;
}
```

Python示例

说明

- Python 2.x请使用httplib模块；Python 3.x请使用http.client模块。
- 采用RFC 3986规范进行urlencode编码，Python 2.x请使用urllib模块的urllib.quote，Python 3.x请使用urllib.parse模块的urllib.parse.quote_plus。
- 如果使用内网访问URL，请使用HTTP协议，需要替换如下函数，将 `HTTPSConnection` 修改为 `HTTPConnection` ：

```
    # Python 2.x 请使用httplib
    # conn = httplib.HTTPConnection(host)

    # Python 3.x 请使用http.client
    conn = http.client.HTTPConnection(host)
```

```
# -*- coding: UTF-8 -*-
# Python 2.x引入httplib模块。
# import httplib
# Python 3.x引入http.client模块。
import http.client
# Python 2.x引入urllib模块。
# import urllib
# Python 3.x引入urllib.parse模块。
import urllib.parse
import json
def processGETRequest(appKey, token, text, audioSaveFile, format, sampleRate) :
    host = 'nls-gateway-cn-shanghai.aliyuncs.com'
    url = 'https://' + host + '/stream/v1/tts'
    # 设置URL请求参数
    url = url + '?appkey=' + appKey
    url = url + '&token=' + token
    url = url + '&text=' + text
    url = url + '&format=' + format
    url = url + '&sample_rate=' + str(sampleRate)
    # voice 发音人，可选，默认是xiaoyun。
    # url = url + '&voice=' + 'xiaoyun'
    # volume 音量，范围是0~100，可选，默认50。
    # url = url + '&volume=' + str(50)
    # speech_rate 语速，范围是-500~500，可选，默认是0。
    # url = url + '&speech_rate=' + str(0)
    # pitch_rate 语调，范围是-500~500，可选，默认是0。
    # url = url + '&pitch_rate=' + str(0)
    print(url)
    # Python 2.x请使用httplib。
```

```
    # conn = httplib.HTTPSConnection(host)
    # Python 3.x请使用http.client。
    conn = http.client.HTTPSConnection(host)
    conn.request(method='GET', url=url)
    # 处理服务端返回的响应。
    response = conn.getresponse()
    print('Response status and response reason:')
    print(response.status ,response.reason)
    contentType = response.getheader('Content-Type')
    print(contentType)
    body = response.read()
    if 'audio/mpeg' == contentType :
        with open(audioSaveFile, mode='wb') as f:
            f.write(body)
        print('The GET request succeed!')
    else :
        print('The GET request failed: ' + str(body))
    conn.close()
def processPOSTRequest(appKey, token, text, audioSaveFile, format, sampleRate) :
    host = 'nls-gateway-cn-shanghai.aliyuncs.com'
    url = 'https://' + host + '/stream/v1/tts'
    # 设置HTTPS Headers。
    httpHeaders = {
        'Content-Type': 'application/json'
        }
    # 设置HTTPS Body。
    body = {'appkey': appKey, 'token': token, 'text': text, 'format': format, 'sample_rate'
: sampleRate}
    body = json.dumps(body)
    print('The POST request body content: ' + body)
    # Python 2.x请使用httplib。
    # conn = httplib.HTTPSConnection(host)
    # Python 3.x请使用http.client。
    conn = http.client.HTTPSConnection(host)
    conn.request(method='POST', url=url, body=body, headers=httpHeaders)
    # 处理服务端返回的响应。
    response = conn.getresponse()
    print('Response status and response reason:')
    print(response.status ,response.reason)
    contentType = response.getheader('Content-Type')
    print(contentType)
    body = response.read()
    if 'audio/mpeg' == contentType :
        with open(audioSaveFile, mode='wb') as f:
            f.write(body)
        print('The POST request succeed!')
    else :
        print('The POST request failed: ' + str(body))
    conn.close()
appKey = '您的appkey'
token = '您的token'
text = '今天是周一，天气挺好的。'
# 采用RFC 3986规范进行urlencode编码。
textUrlencode = text
```

```
# Python 2.x请使用urllib.quote。
# textUrlencode = urllib.quote(textUrlencode, '')
# Python 3.x请使用urllib.parse.quote_plus。
textUrlencode = urllib.parse.quote_plus(textUrlencode)
textUrlencode = textUrlencode.replace("+", "%20")
textUrlencode = textUrlencode.replace("*", "%2A")
textUrlencode = textUrlencode.replace("%7E", "~")
print('text: ' + textUrlencode)
audioSaveFile = 'syAudio.wav'
format = 'wav'
sampleRate = 16000
# GET请求方式
processGETRequest(appKey, token, textUrlencode, audioSaveFile, format, sampleRate)
# POST请求方式
# processPOSTRequest(appKey, token, text, audioSaveFile, format, sampleRate)
```

PHP示例

说明

PHP示例中使用了cURL函数，要求PHP版本在4.0.2以上，并且确保已安装cURL扩展。

```
<?php
function processGETRequest($appkey, $token, $text, $audioSaveFile, $format, $sampleRate) {
    $url = "https://nls-gateway-cn-shanghai.aliyuncs.com/stream/v1/tts";
    $url = $url . "?appkey=" . $appkey;
    $url = $url . "&token=" . $token;
    $url = $url . "&text=" . $text;
    $url = $url . "&format=" . $format;
    $url = $url . "&sample_rate=" . strval($sampleRate);
    // voice 发音人，可选，默认是xiaoyun。
    // $url = $url . "&voice=" . "xiaoyun";
    // volume 音量，范围是0~100，可选，默认50。
    // $url = $url . "&volume=" . strval(50);
    // speech_rate 语速，范围是-500~500，可选，默认是0。
    // $url = $url . "&speech_rate=" . strval(0);
    // pitch_rate 语调，范围是-500~500，可选，默认是0。
    // $url = $url . "&pitch_rate=" . strval(0);
    print $url . "\n";
    $curl = curl_init();
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, TRUE);
    /**
     * 设置HTTPS GET URL。
     */
    curl_setopt($curl, CURLOPT_URL, $url);
    /**
     * 设置返回的响应包含HTTPS头部信息。
     */
    curl_setopt($curl, CURLOPT_HEADER, TRUE);
    /**
     * 发送HTTPS GET请求。
     */
    curl_setopt($curl, CURLOPT_SSL_VERIFYPEER, FALSE);
```

```
    curl_setopt($curl, CURLOPT_SSL_VERIFYHOST, FALSE);
    $response = curl_exec($curl);
    if ($response == FALSE) {
        print "curl_exec failed!\n";
        curl_close($curl);
        return ;
    }
    /**
     * 处理服务端返回的响应。
     */
    $headerSize = curl_getinfo($curl, CURLINFO_HEADER_SIZE);
    $headers = substr($response, 0, $headerSize);
    $bodyContent = substr($response, $headerSize);
    curl_close($curl);
    if (stripos($headers, "Content-Type: audio/mpeg") != FALSE || stripos($headers, "Conten
t-Type:audio/mpeg") != FALSE) {
        file_put_contents($audioSaveFile, $bodyContent);
        print "The GET request succeed!\n";
    }
    else {
        print "The GET request failed: " . $bodyContent . "\n";
    }
}
function processPOSTRequest($appkey, $token, $text, $audioSaveFile, $format, $sampleRate) {
    $url = "https://nls-gateway-cn-shanghai.aliyuncs.com/stream/v1/tts";
    /**
     * 请求参数，以JSON格式字符串填入HTTPS POST请求的Body中。
     */
    $taskArr = array(
        "appkey" => $appkey,
        "token" => $token,
        "text" => $text,
        "format" => $format,
        "sample_rate" => $sampleRate
        // voice 发音人，可选，默认是xiaoyun。
        // "voice" => "xiaoyun",
        // volume 音量，范围是0~100，可选，默认50。
        // "volume" => 50,
        // speech_rate 语速，范围是-500~500，可选，默认是0。
        // "speech_rate" => 0,
        // pitch_rate 语调，范围是-500~500，可选，默认是0。
        // "pitch_rate" => 0
    );
    $body = json_encode($taskArr);
    print "The POST request body content: " . $body . "\n";
    $curl = curl_init();
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, TRUE);
    /**
     * 设置HTTPS POST URL。
     */
    curl_setopt($curl, CURLOPT_URL, $url);
    curl_setopt($curl, CURLOPT_POST, TRUE);
    /**
     * 设置HTTPS POST请求头部。
```

```
     * */
    $httpHeaders = array(
        "Content-Type: application/json"
    );
    curl_setopt($curl, CURLOPT_HTTPHEADER, $httpHeaders);
    /**
     * 设置HTTPS POST请求体。
     */
    curl_setopt($curl, CURLOPT_POSTFIELDS, $body);
    /**
     * 设置返回的响应包含HTTPS头部信息。
     */
    curl_setopt($curl, CURLOPT_HEADER, TRUE);
    /**
     * 发送HTTPS POST请求。
     */
    $response = curl_exec($curl);
    if ($response == FALSE) {
        print "curl_exec failed!\n";
        curl_close($curl);
        return ;
    }
    /**
     * 处理服务端返回的响应。
     */
    $headerSize = curl_getinfo($curl, CURLINFO_HEADER_SIZE);
    $headers = substr($response, 0, $headerSize);
    $bodyContent = substr($response, $headerSize);
    curl_close($curl);
    if (stripos($headers, "Content-Type: audio/mpeg") != FALSE || stripos($headers, "Conten
t-Type:audio/mpeg") != FALSE) {
        file_put_contents($audioSaveFile, $bodyContent);
        print "The POST request succeed!\n";
    }
    else {
        print "The POST request failed: " . $bodyContent . "\n";
    }
}
$appkey = "您的appkey";
$token = "您的token";
$text = "今天是周一，天气挺好的。";
$textUrlEncode = urlencode($text);
$textUrlEncode = preg_replace('/\+/', '%20', $textUrlEncode);
$textUrlEncode = preg_replace('/\*/', '%2A', $textUrlEncode);
$textUrlEncode = preg_replace('/%7E/', '~', $textUrlEncode);
$audioSaveFile = "syAudio.wav";
$format = "wav";
$sampleRate = 16000;
processGETRequest($appkey, $token, $textUrlEncode, $audioSaveFile, $format, $sampleRate);
// processPOSTRequest($appkey, $token, $text, $audioSaveFile, $format, $sampleRate);
?>
```

Node.js示例

说明

首先安装request依赖，请在您的示例文件所在目录执行如下命令：

```
npm install request --save
```

```
const request = require('request');
const fs = require('fs');
function processGETRequest(appkey, token, text, audioSaveFile, format, sampleRate) {
    var url = 'https://nls-gateway-cn-shanghai.aliyuncs.com/stream/v1/tts';
    /**
     * 设置URL请求参数。
     */
    url = url + '?appkey=' + appkey;
    url = url + '&token=' + token;
    url = url + '&text=' + text;
    url = url + '&format=' + format;
    url = url + '&sample_rate=' + sampleRate;
    // voice 发音人，可选，默认是xiaoyun。
    // url = url + "&voice=" + "xiaoyun";
    // volume 音量，范围是0~100，可选，默认50。
    // url = url + "&volume=" + 50;
    // speech_rate 语速，范围是-500~500，可选，默认是0。
    // url = url + "&speech_rate=" + 0;
    // pitch_rate 语调，范围是-500~500，可选，默认是0。
    // url = url + "&pitch_rate=" + 0;
    console.log(url);
    /**
     * 设置HTTPS GET请求。
     * encoding必须设置为null，HTTPS响应的Body为二进制Buffer类型。
     */
    var options = {
        url: url,
        method: 'GET',
        encoding: null
    };
    request(options, function (error, response, body) {
        /**
         * 处理服务端的响应。
         */
        if (error != null) {
            console.log(error);
        }
        else {
            var contentType = response.headers['content-type'];
            if (contentType === undefined || contentType != 'audio/mpeg') {
                console.log(body.toString());
                console.log('The GET request failed!');
            }
            else {
                fs.writeFileSync(audioSaveFile, body);
                console.log('The GET request is succeed!');
```

```
            }
        }
    });
}
function processPOSTRequest(appkeyValue, tokenValue, textValue, audioSaveFile, formatValue,
sampleRateValue) {
    var url = 'https://nls-gateway-cn-shanghai.aliyuncs.com/stream/v1/tts';
    /**
     * 请求参数，以JSON格式字符串填入HTTPS POST请求的Body中。
    */
    var task = {
        appkey : appkeyValue,
        token : tokenValue,
        text : textValue,
        format : formatValue,
        sample_rate : sampleRateValue
        // voice 发音人，可选，默认是xiaoyun。
        // voice : 'xiaoyun',
        // volume 音量，范围是0~100，可选，默认50。
        // volume : 50,
        // speech_rate 语速，范围是-500~500，可选，默认是0。
        // speech_rate : 0,
        // pitch_rate 语调，范围是-500~500，可选，默认是0。
        // pitch_rate : 0
    };
    var bodyContent = JSON.stringify(task);
    console.log('The POST request body content: ' + bodyContent);
    /**
     * 设置HTTPS POST请求头部。
     */
    var httpHeaders = {
        'Content-type' : 'application/json'
    }
    /**
     * 设置HTTPS POST请求。
     * encoding必须设置为null，HTTPS响应的Body为二进制Buffer类型。
     */
    var options = {
        url: url,
        method: 'POST',
        headers: httpHeaders,
        body: bodyContent,
        encoding: null
    };
    request(options, function (error, response, body) {
        /**
         * 处理服务端的响应。
         */
        if (error != null) {
            console.log(error);
        }
        else {
            var contentType = response.headers['content-type'];
            if (contentType === undefined || contentType != 'audio/mpeg') {
```

```
                console.log(body.toString());
                console.log('The POST request failed!');
            }
            else {
                fs.writeFileSync(audioSaveFile, body);
                console.log('The POST request is succeed!');
            }
        }
    });
}
var appkey = '您的appkey';
var token = '您的token';
var text = '今天是周一，天气挺好的。';
var textUrlEncode = encodeURIComponent(text)
                    .replace(/[!'()*]/g, function(c) {
                        return '%' + c.charCodeAt(0).toString(16);
                    });
console.log(textUrlEncode);
var audioSaveFile = 'syAudio.wav';
var format = 'wav';
var sampleRate = 16000;
processGETRequest(appkey, token, textUrlEncode, audioSaveFile, format, sampleRate);
// processPOSTRequest(appkey, token, text, audioSaveFile, format, sampleRate);
```

.Net示例

说明

示例使用依赖System.Net.Http、System.Web、Newtonsoft.Json.Linq。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Net.Http;
using System.Web;
using Newtonsoft.Json.Linq;
namespace RESTfulAPI
{
    class SpeechSynthesizerRESTfulDemo
    {
        private string appkey;
        private string token;
        public SpeechSynthesizerRESTfulDemo(string appkey, string token)
        {
            this.appkey = appkey;
            this.token = token;
        }
        public void processGETRequest(string text, string audioSaveFile, string format, int
sampleRate)
        {
            /**
```

```
            /**
             * 设置HTTPS GET请求：
             * 1.使用HTTPS协议
             * 2.语音识别服务域名：nls-gateway-cn-shanghai.aliyuncs.com
             * 3.语音识别接口请求路径：/stream/v1/tts
             * 4.设置必须请求参数：appkey、token、text、format、sample_rate
             * 5.设置可选请求参数：voice、volume、speech_rate、pitch_rate
             */
            string url = "http://nls-gateway.aliyuncs.com/stream/v1/tts";
            url = url + "?appkey=" + appkey;
            url = url + "&token=" + token;
            url = url + "&text=" + text;
            url = url + "&format=" + format;
            url = url + "&sample_rate=" + sampleRate.ToString();
            // voice 发音人，可选，默认是xiaoyun。
            // url = url + "&voice=" + "xiaoyun";
            // volume 音量，范围是0~100，可选，默认50。
            // url = url + "&volume=" + 50;
            // speech_rate 语速，范围是-500~500，可选，默认是0。
            // url = url + "&speech_rate=" + 0;
            // pitch_rate 语调，范围是-500~500，可选，默认是0。
            // url = url + "&pitch_rate=" + 0;
            System.Console.WriteLine(url);
            /**
             * 发送HTTPS GET请求，处理服务端的响应。
             */
            HttpClient client = new HttpClient();
            HttpResponseMessage response = null;
            response = client.GetAsync(url).Result;
            string contentType = null;
            if (response.IsSuccessStatusCode)
            {
                string[] typesArray = response.Content.Headers.GetValues("Content-Type").To
Array();
                if (typesArray.Length > 0)
                {
                    contentType = typesArray.First();
                }
            }
            if ("audio/mpeg".Equals(contentType))
            {
                byte[] audioBuff = response.Content.ReadAsByteArrayAsync().Result;
                FileStream fs = new FileStream(audioSaveFile, FileMode.Create);
                fs.Write(audioBuff, 0, audioBuff.Length);
                fs.Flush();
                fs.Close();
                System.Console.WriteLine("The GET request succeed!");
            }
            else
            {
                // ContentType 为 null 或者为 "application/json"
                System.Console.WriteLine("Response status code and reason phrase: " +
                    response.StatusCode + " " + response.ReasonPhrase);
                string responseBodyAsText = response.Content.ReadAsStringAsync().Result;
                System.Console.WriteLine("The GET request failed: " + responseBodyAsText);
```

```
                System.Console.WriteLine("The GET request failed: " + responseBodyAsStr);
            }
        }
        public void processPOSTRequest(string text, string audioSaveFile, string format, in
t sampleRate)
        {
            /**
             * 设置HTTPS POST请求：
             * 1.使用HTTPS协议
             * 2.语音合成服务域名：nls-gateway-cn-shanghai.aliyuncs.com
             * 3.语音合成接口请求路径：/stream/v1/tts
             * 4.设置必须请求参数：appkey、token、text、format、sample_rate
             * 5.设置可选请求参数：voice、volume、speech_rate、pitch_rate
             */
            string url = "https://nls-gateway-cn-shanghai.aliyuncs.com/stream/v1/tts";
            JObject obj = new JObject();
            obj["appkey"] = appkey;
            obj["token"] = token;
            obj["text"] = text;
            obj["format"] = format;
            obj["sample_rate"] = sampleRate;
            // voice 发音人，可选，默认是xiaoyun。
            // obj["voice"] = "xiaoyun";
            // volume 音量，范围是0~100，可选，默认50。
            // obj["volume"] = 50;
            // speech_rate 语速，范围是-500~500，可选，默认是0。
            // obj["speech_rate"] = 0;
            // pitch_rate 语调，范围是-500~500，可选，默认是0。
            // obj["pitch_rate"] = 0;
            String bodyContent = obj.ToString();
            StringContent content = new StringContent(bodyContent, Encoding.UTF8, "applicat
ion/json");
            /**
             * 发送HTTPS POST请求，处理服务端的响应。
             */
            HttpClient client = new HttpClient();
            HttpResponseMessage response = client.PostAsync(url, content).Result;
            string contentType = null;
            if (response.IsSuccessStatusCode)
            {
                string[] typesArray = response.Content.Headers.GetValues("Content-Type").To
Array();
                if (typesArray.Length > 0)
                {
                    contentType = typesArray.First();
                }
            }
            if ("audio/mpeg".Equals(contentType))
            {
                byte[] audioBuff = response.Content.ReadAsByteArrayAsync().Result;
                FileStream fs = new FileStream(audioSaveFile, FileMode.Create);
                fs.Write(audioBuff, 0, audioBuff.Length);
                fs.Flush();
                fs.Close();
                System.Console.WriteLine("The POST request succeed!");
```

```
            }
            else
            {
                System.Console.WriteLine("Response status code and reason phrase: " +
                    response.StatusCode + " " + response.ReasonPhrase);
                string responseBodyAsText = response.Content.ReadAsStringAsync().Result;
                System.Console.WriteLine("The POST request failed: " + responseBodyAsText);
            }
        }
        static void Main(string[] args)
        {
            if (args.Length < 2)
            {
                System.Console.WriteLine("SpeechSynthesizerRESTfulDemo need params: <token>
<app-key>");
                return;
            }
            string token = args[0];
            string appkey = args[1];
            SpeechSynthesizerRESTfulDemo demo = new SpeechSynthesizerRESTfulDemo(appkey, to
ken);
            string text = "今天是周一，天气挺好的。";
            // 采用RFC 3986规范进行urlencode编码。
            string textUrlEncode = text;
            textUrlEncode = HttpUtility.UrlEncode(textUrlEncode, Encoding.UTF8)
                .Replace("+", "%20")
                .Replace("*", "%2A")
                .Replace("%7E", "~");
            System.Console.WriteLine(textUrlEncode);
            string audioSaveFile = "syAudio.wav";
            string format = "wav";
            int sampleRate = 16000;
            demo.processGETRequest(textUrlEncode, audioSaveFile, format, sampleRate);
            //demo.processPOSTRequest(text, audioSaveFile, format, sampleRate);
        }
    }
}
```

GO示例

```
package main
import (
    "fmt"
    "net/url"
    "net/http"
    "io/ioutil"
    "encoding/json"
    "strconv"
    "os"
    "bytes"
    "strings"
)
func processGETRequest(appkey string, token string, text string, audioSaveFile string, form
```

```
at string, sampleRate int) {
    /**
     * 设置HTTPS GET请求：
     * 1.使用HTTPS协议
     * 2.语音识别服务域名：nls-gateway-cn-shanghai.aliyuncs.com
     * 3.语音识别接口请求路径：/stream/v1/tts
     * 4.设置必须请求参数：appkey、token、text、format、sample_rate
     * 5.设置可选请求参数：voice、volume、speech_rate、pitch_rate
     */
    var url string = "https://nls-gateway-cn-shanghai.aliyuncs.com/stream/v1/tts"
    url = url + "?appkey=" + appkey
    url = url + "&token=" + token
    url = url + "&text=" + text
    url = url + "&format=" + format
    url = url + "&sample_rate=" + strconv.Itoa(sampleRate)
    // voice 发音人，可选，默认是xiaoyun。
    // url = url + "&voice=" + "xiaoyun"
    // volume 音量，范围是0~100，可选，默认50。
    // url = url + "&volume=" + strconv.Itoa(50)
    // speech_rate 语速，范围是-500~500，可选，默认是0。
    // url = url + "&speech_rate=" + strconv.Itoa(0)
    // pitch_rate 语调，范围是-500~500，可选，默认是0。
    // url = url + "&pitch_rate=" + strconv.Itoa(0)
    fmt.Println(url)
    /**
     * 发送HTTPS GET请求，处理服务端的响应。
     */
    response, err := http.Get(url)
    if err != nil {
        fmt.Println("The GET request failed!")
        panic(err)
    }
    defer response.Body.Close()
    contentType := response.Header.Get("Content-Type")
    body, _ := ioutil.ReadAll(response.Body)
    if ("audio/mpeg" == contentType) {
        file, _ := os.Create(audioSaveFile)
        defer file.Close()
        file.Write([]byte(body))
        fmt.Println("The GET request succeed!")
    } else {
        // ContentType 为 null 或者为 "application/json"
        statusCode := response.StatusCode
        fmt.Println("The HTTP statusCode: " + strconv.Itoa(statusCode))
        fmt.Println("The GET request failed: " + string(body))
    }
}
func processPOSTRequest(appkey string, token string, text string, audioSaveFile string, for
mat string, sampleRate int) {
    /**
     * 设置HTTPS POST请求：
     * 1.使用HTTPS协议
     * 2.语音合成服务域名：nls-gateway-cn-shanghai.aliyuncs.com
     * 3.语音合成接口请求路径：/stream/v1/tts
     * 4.设置必须请求参数：appkey、token、text、format、sample_rate
```

```
     * 4.设置必须请求参数：appkey、token、text、format、sample_rate
     * 5.设置可选请求参数：voice、volume、speech_rate、pitch_rate
     */
    var url string = "https://nls-gateway-cn-shanghai.aliyuncs.com/stream/v1/tts"
    bodyContent := make(map[string]interface{})
    bodyContent["appkey"] = appkey
    bodyContent["text"] = text
    bodyContent["token"] = token
    bodyContent["format"] = format
    bodyContent["sample_rate"] = sampleRate
    // voice 发音人，可选，默认是xiaoyun。
    // bodyContent["voice"] = "xiaoyun"
    // volume 音量，范围是0~100，可选，默认50。
    // bodyContent["volume"] = 50
    // speech_rate 语速，范围是-500~500，可选，默认是0。
    // bodyContent["speech_rate"] = 0
    // pitch_rate 语调，范围是-500~500，可选，默认是0。
    // bodyContent["pitch_rate"] = 0
    bodyJson, err := json.Marshal(bodyContent)
    if err != nil {
        panic(nil)
    }
    fmt.Println(string(bodyJson))
    /**
     * 发送HTTPS POST请求，处理服务端的响应。
     */
    response, err := http.Post(url, "application/json;charset=utf-8", bytes.NewBuffer([]byt
e(bodyJson)))
    if err != nil {
        panic(err)
    }
    defer response.Body.Close()
    contentType := response.Header.Get("Content-Type")
    body, _ := ioutil.ReadAll(response.Body)
    if ("audio/mpeg" == contentType) {
        file, _ := os.Create(audioSaveFile)
        defer file.Close()
        file.Write([]byte(body))
        fmt.Println("The POST request succeed!")
    } else {
        // ContentType 为 null 或者为 "application/json"
        statusCode := response.StatusCode
        fmt.Println("The HTTP statusCode: " + strconv.Itoa(statusCode))
        fmt.Println("The POST request failed: " + string(body))
    }
}
func main() {
    var appkey string = "您的appkey"
    var token  string = "您的token"
    var text string = "今天是周一，天气挺好的。"
    var textUrlEncode = text
    textUrlEncode = url.QueryEscape(textUrlEncode)
    textUrlEncode = strings.Replace(textUrlEncode, "+", "%20", -1)
    textUrlEncode = strings.Replace(textUrlEncode, "*", "%2A", -1)
    textUrlEncode = strings.Replace(textUrlEncode, "%7E", "~", -1)
```

```
    textUrlEncode = strings.Replace(textUrlEncode, "+", "%20", -1)
    fmt.Println(textUrlEncode)
    var audioSaveFile string = "syAudio.wav"
    var format string = "wav"
    var sampleRate int = 16000
    processGETRequest(appkey, token, textUrlEncode, audioSaveFile, format, sampleRate)
    // processPOSTRequest(appkey, token, text, audioSaveFile, format, sampleRate)
}
```

3.移动端SDK

3.1. 接口说明

语音合成提供将输入文本合成为语音二进制数据的功能。

功能介绍

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既能够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。

语音合成功能支持如下能力：

- 支持输出PCM、MP3编码格式数据。
- 支持设置语速、语调、音量。
- 支持设置声音类型，如下表所示。

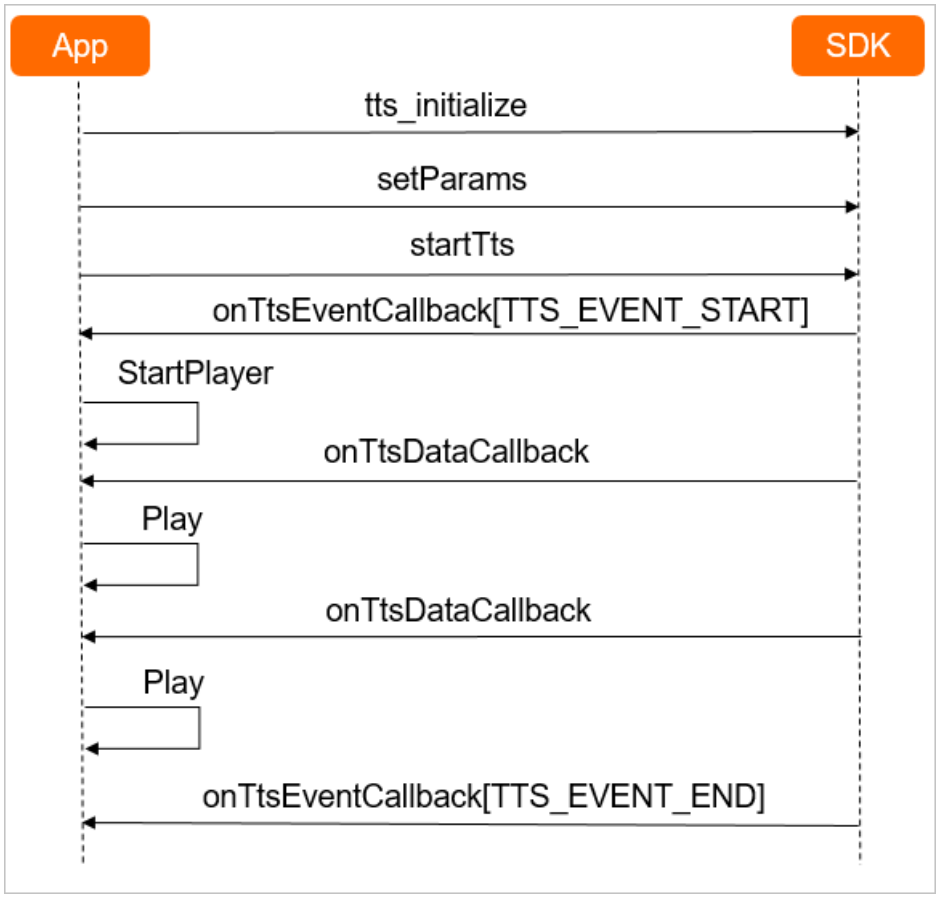
限制条件

- 传入的文本必须采用 `UTF-8` 编码。
- 传入的文本不能超过300个字符，超过300个字符的内容会被截断。

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。 说明 • 使用内网访问方式，将不产生ECS实例的公网流量费用。 • 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

服务端响应除了音频流之外，均会在返回信息的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

   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。
app_key	String	是	管控台创建项目的appkey。

参数	类型	是否必选	说明
token	String	是	请确保该token可以使用并在有效期内。 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，能够唯一表示一台设备（如Mac地址/SN/UniquePsuedoID）。

2. 开始合成

   客户端发起语音合成请求，在请求消息中进行参数设置，各参数通过SDK中setparamTts方法设置，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。
token	String	否	如果需要更新，则进行设置。
direct_host	String	否	支持客户端自行DNS解析后传入IP进行访问。
font_name	String	否	发音人，默认是xiaoyun。
encode_type	String	否	音频编码格式，默认值：pcm。支持格式：pcm、wav、mp3。
sample_rate	String	否	音频采样率，默认值：16000。
volume	String	否	音量，取值范围：0~2，默认值：1.0。

参数	类型	是否必选	说明
speed_level	String	否	语速，取值范围：0～2，默认值：1.0，值越大语速越快。
pitch_level	String	否	语调，取值范围：-500～500，默认值：0，值越大声音越尖锐。
enable_subtitle	String	否	字级别音素边界功能开关，该参数只对支持字级别音素边界接口的发音人有效。“1”表示打开，“0”表示关闭。

3. 接收数据合成

   服务端返回合成的语音二进制数据，SDK接收并处理二进制数据。

4. 结束合成

   语音合成完毕，服务端发送合成完毕事件通知。

错误码

如果语音合成发生错误，SDK将上报TTS_EVENT_ERROR事件，并提供错误信息，如下表所示。

错误码	错误消息	描述
0	TTS_SUCCESS	成功。
140000	TTS_CREATE_FAILED	引擎初始化失败。
140001	TTS_ENGINE_INVALID	引擎未初始化。
140002	TTS_TEXT_ERROR	文本非法，如文本为空。
140003	TTS_MALLOC_FAILED	内存申请失败。
140004	TTS_TEXT_QUEUE_FULL	任务队列已满。
140005	TTS_ASSETPATH_INVALID	资源路径为空。

错误码	错误消息	描述
140006	TTS_HANLDE_INVALID	处理线程不存在。
140007	TTS_CREATE_HANLDE_FAILED	创建处理线程失败。
140008	TTS_AUTH_FAILED	鉴权失败，无法继续使用SDK。
140009	TTS_TEXT_QUEUE_EMPTY	合成队列已空。
140010	TTS_MODE_INVALID	合成模式无效。
140012	TTS_OPEN_FILE_FAILED	打开文件失败。
140013	TTS_STATE_INVALID	SDK的状态机校验失败。
140014	TTS_SYNTHESIZER_INIT_ERROR	合成器初始化失败。
140015	TTS_SYNTHESIZER_RELEASE_ERROR	合成器释放失败。
140016	TTS_SYNTHESIZER_FAILED	合成失败。
140017	TTS_WAIT_TIMEOUT	超时退出。
140018	TTS_CLOSED	没有编译语音合成部分代码。
140100	TTS_PARAM_INVALID	参数无效。
140101	TTS_PARAM_VALUE_INVALID	参数值无效。
140102	TTS_CFG_OPEN_FAILED	配置文件打开失败。
140103	TTS_CFG_WRONG_FORMAT	配置文件格式错误。
140150	TTS_LOG_OPEN_FAILED	创建日志文件失败。

错误码	错误消息	描述
140200	TTS_AM_CREATE_FAILED	播放器创建失败。
140201	TTS_AM_OPEN_FAILED	播放器打开失败。
140210	TTS_DECODER_INIT_FAILED	音频解码器初始化失败。
140211	TTS_DECODER_MALLOC_FAILED	音频解码器申请内存失败。
140212	TTS_DECODER_INPUT_TOO_MANY	单次输入数据过多，将被丢弃。
140213	TTS_DECODER_OUTPUT_TOO_MANY	输出数据过多，超过缓存，会丢失。
140220	TTS_AP_INIT_FAILED	音频处理单元打开失败。
140221	TTS_AP_START_FAILED	音频处理单元启动出错。
140222	TTS_AP_MALLOC_FAILED	音频处理单元内存申请失败。
140230	TTS_BGM_START_FAILED	背景音乐开始失败。
140231	TTS_BGM_DECODE_INVALID	解码器初始化失败。
140232	TTS_BGM_ADD_FAILED	本句加背景音乐失败。
140233	TTS_BGM_MALLOC_FAILED	内存申请失败。
140234	TTS_BGM_OPEN_FILE_FAILED	背景音乐文件打开失败。
140235	TTS_BGM_FILE_FORMAT_ERROR	背景音乐格式错误。
140300	TTS_CACHE_INIT_FAILED	初始化cache失败。
140301	TTS_CACHE_MGR_INVALID	cache管理器未初始化。

错误码	错误消息	描述
140303	TTS_CACHE_CALLBACK_INVALID	回调函数未初始化。
140304	TTS_CACHE_START_READ_FAILED	开始打开cache文件失败。
140305	TTS_CACHE_READ_FAILED	读取cache文件数据失败。
140306	TTS_CACHE_MALLOC_FAILED	cache模块内存申请失败。
140307	TTS_CACHE_DELETE_FAILED	删除失败。
140308	TTS_CACHE_PATH_INVALID	无法创建缓存路径。
140309	TTS_CACHE_LIST_CREATE_FAILED	cache列表创建失败。
140310	TTS_CACHE_FAILED	缓存失败。
140311	TTS_CACHE_TOO_MANY	缓存过多。
140312	TTS_CACHE_PARAM_INVALID	缓存功能的参数设置错误。
140313	TTS_CACHE_RECORDING_OPEN_FAILED	打开本地文件错误。
140352	TTS_FONT_INITLIST_INVALID	fontlist管理器未初始化。
140353	TTS_FONT_CMD_INVALID	命令格式错误。
140354	TTS_FONT_RESPONSE_ERROR	服务端返回格式错误。
140355	TTS_FONT_RESPONSELIST_ERROR	fontlist请求服务端返回格式错误。
140356	TTS_FONT_GET_FONTLIST_FAILED	获取fontlist失败。
140357	TTS_FONT_REQUEST_CMD_ERROR	创建请求指令错误。

错误码	错误消息	描述
140358	TTS_FONT_LOCALMSG_ERROR	本地list文件解析失败。
140360	TTS_FONT_CLOUDMSG_ERROR	云端list解析失败。
140900	TTS_LOCAL_CRE_ENGINE_ERROR	本地引擎初始化失败。
140901	TTS_LOCAL_ENGINE_INVALID	本地引擎未初始化。
140902	TTS_LOCAL_ASSET_ERROR	本地资源校验失败。
140903	TTS_LOCAL_CRE_TASK_ERROR	创建本地task失败。
140904	TTS_LOCAL_TASK_INVALID	Task无效。
140905	TTS_LOCAL_START_FAILED	本地开始合成失败。
141000	TTS_CLOUD_CREATE_FAILED	云端引擎初始化失败。
141001	TTS_CLOUD_ENGINE_INVALID	云端引擎未初始化。
141002	TTS_CLOUD_TASK_FAILED	创建云端Task失败。
141003	TTS_CLOUD_TASK_INVALID	Task无效。
141004	TTS_CLOUD_START_FAILED	云端服务请求失败。
141005	TTS_CLOUD_CANCEL_FAILED	取消任务失败。
141006	TTS_CLOUD_NETWORK_BROKEN	网络较差。
141010	TTS_CLOUD_DNS_FAILED	网络配置DNS有问题。
144001	TTS_CLOUD_AUTH_FAILED	身份认证失败。

错误码	错误消息	描述
144002	TTS_CLOUD_INVALID_MESSAGE	消息无效。
144003	TTS_CLOUD_INVALID_TOKEN	令牌过期或无效。
144004	TTS_CLOUD_WAIT_TIMEOUT	空闲超时。
144005	TTS_CLOUD_EXCEED_CONCURRENCY	请求数量过多。
144100	TTS_CLOUD_INVALID_INTERFACE	接口不支持。
144101	TTS_CLOUD_UNSUPPORTED_ORDER	指令不支持。
144102	TTS_CLOUD_INVALID_ORDER	指令无效。
144103	TTS_CLOUD_CLIENT_DISCONNECT	客户端提前断开连接。
144200	TTS_CLOUD_INVALID_APPKEY	应用不存在。
144300	TTS_CLOUD_INVALID_PARAM	访问云端服务参数错误。
144400	TTS_CLOUD_SERVER_ERROR	语音合成服务端出错。

3.2. 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

- 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。

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，其中语音合成示例代码为TtsBasicActivity.java文件，替换appkey和token后可直接运行。

SDK关键接口

- **tts_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 ticket：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 tts_initialize(INativeTtsCallback callback,
                                           String ticket,
                                           final Constants.LogLevel level,
                                           boolean save_log)
```

其中，INativeTtsCallback类型包含如下回调。

- **onTtsEventCallback**：SDK事件回调。

```
/**
     * 事件回调
     * @param event：回调事件，参见如下事件列表。
     * @param task_id：请求的任务ID。
     * @param ret_code：参见错误码，出现TTS_EVENT_ERROR事件时有效。
     */
    void onTtsEventCallback(TtsEvent event, String task_id, int ret_code);
```

事件列表：

名称	说明
TTS_EVENT_START	语音合成开始，准备播放。
TTS_EVENT_END	语音合成播放结束
TTS_EVENT_CANCEL	取消语音合成
TTS_EVENT_PAUSE	语音合成暂停
TTS_EVENT_RESUME	语音合成恢复
TTS_EVENT_ERROR	语音合成发生错误

- **onTtsDataCallback**：合成数据回调。

```
/**
     * 合成数据回调
     * @param info：使用时间戳功能时，返回JSON格式的时间戳结果。
     * @param info_len：info字段的数据长度，暂不使用。
     * @param data：合成的音频数据，写入播放器。
     */
    void onTtsDataCallback(String info, int info_len, byte[] data);
```

- **setparamTts**：设置参数。

```
/**
     * 以键值对形式设置参数
     * @param param：参数名，参见接口说明。
     * @param value：参数值，参见接口说明。
     * @return：参见错误码。
     */
    public synchronized int setparamTts(String param, String value);
```

- **getparamTts**：获取参数。

```
/**
     * 获取参数值
     * @param param：参数名，参考接口说明。
     * @return：参数值。
     */
    public String getparamTts(String param);
```

- startTts：开始合成。

```
/**
     * 开始合成任务
     * @param priority：任务优先级，请使用"1"。
     * @param taskid：任务ID，可传入32个字节的uuid，或传入空内容由SDK自动生成。
     * @param text：播放的文本内容。
     * @return：参见错误码。
     */
    public synchronized int startTts(String priority, String taskid, String text)
```

- cancelTts：取消合成。

```
/**
     * 取消合成任务
     * @param taskid：传入想要停止的任务ID，如果为空则取消所有任务。
     * @return：参见错误码。
     */
    public synchronized int cancelTts(String taskid)
```

- pauseTts：暂停合成。

```
/**
     * 暂停合成任务
     * @return：参见错误码。
     */
    public synchronized int pauseTts()
```

- resumeTts：恢复合成。

```
/**
     * 恢复暂停的任务
     * @return：参见错误码。
     */
    public synchronized int resumeTts()
```

- tts_release：释放SDK资源。

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

调用步骤

1. 初始化SDK和播放组件。

2. 根据业务需求设置参数。
3. 调用startTts进行播放。
4. 在合成数据回调中，将数据写入播放器进行播放，建议使用流式播放。
5. 收到语音合成结束的回调。

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。

语音合成初始化

```
//拷贝资源
CommonUtils.copyAssetsData(this);
//SDK初始化
int ret = NativeNui.GetInstance().tts_initialize(new INativeTtsCallback() {}, genTicket(path), Constants.LogLevel.LOG_LEVEL_VERBOSE, true);
```

其中，genTicket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 genTicket(String workpath) {
        String str = "";
        try {
            JSONObject object = Auth.getAliYunTicket();
            object.put("workspace", workpath);
            str = object.toString();
        } catch (JSONException e) {
            e.printStackTrace();
        }
        Log.i(TAG, "UserContext:" + str);
        return str;
    }
```

根据需求设置参数

```
NativeNui.GetInstance().setparamTts("font_name", "xiaoyun");
```

启动语音合成

```
NativeNui.GetInstance().startTts("1", "", ttsText);
```

回调处理

- onTtsEventCallback：语音合成事件回调，根据语音合成状态控制播放器。

```
public void onTtsEventCallback(INativeTtsCallback.TtsEvent event) {
                Log.i(TAG, "tts event:" + event);
                if (event == INativeTtsCallback.TtsEvent.TTS_EVENT_START) {
                    mAudioTrack.play();
                    Log.i(TAG, "start play");
                } else if (event == INativeTtsCallback.TtsEvent.TTS_EVENT_END) {
                    Log.i(TAG, "play end");
                } else if (event == TtsEvent.TTS_EVENT_PAUSE) {
                    mAudioTrack.pause();
                    Log.i(TAG, "play pause");
                } else if (event == TtsEvent.TTS_EVENT_RESUME) {
                    mAudioTrack.play();
                }
            }
```

- onTtsDataCallback：语音合成数据回调，将回调中的合成数据写入播放器进行播放。

```
public void onTtsDataCallback(String info, int info_len, byte[] data) {
                if (data.length > 0) {
                    mAudioTrack.setAudioData(data);
                }
            }
```

结束语音合成

```
NativeNui.GetInstance().cancelTts("");
```

3.3. 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

- 阅读接口说明，详情请参见接口说明。
- 已准备项目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。其中语音合成示例代码在TTSViewController类中。

SDK关键接口

- **tts_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，意外下可能引起阻塞。
     * @param parameters: 初始化参数，参考接口说明
     * @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 参考错误码
     */
    -(int) nui_tts_initialize:(const char *)parameters
                  logLevel:(NuiSdkLogLevel)level
                   saveLog:(BOOL)save_log;
```

- **nui_tts_play**: 开始播放

```
/**
     * 开始播放
     * @param priority: 任务优先级，请使用"1"
     * @param taskid: 任务id，可传入32个字节的uuid或者传入空内容由SDK自动生成
     * @param text: 要播放的文本内容
     * @return 参考错误码
     */
    -(int) nui_tts_play:(const char *)priority
             taskId:(const char *)taskid
               text:(const char *)text;
```

- **nui_tts_cancel**：取消播放

```
/**
     * 取消合成任务
     * @param taskid: 传入想要停止的任务id，如果为空则取消所有任务
     * @return 参考错误码
     */
    -(int) nui_tts_cancel:(const char *)taskid;
```

- **nui_tts_pause**：暂停播放

```
/**
     * 暂停
     * @return 参考错误码
     */
    -(int) nui_tts_pause;
```

- **nui_tts_resume**：恢复播放

```
/**
     * 恢复暂停的任务
     * @return 参考错误码
     */
    -(int) nui_tts_resume;
```

- nui_tts_set_param：设置语音合成参数

```
/**
     * 以键值对形式设置参数
     * @param param: 参数名，参考接口说明
     * @param value: 参数值，参考接口说明
     * @return 参考错误码
     */
    -(int) nui_tts_set_param:(const char *)param
                   value:(const char *)value;
```

- nui_tts_get_param：获取参数

```
/**
     * 获取参数值
     * @param param: 参数名，参考接口说明
     * @return 参数值
     */
    -(const char *) nui_tts_get_param:(const char *)param;
```

- nui_tts_release：释放SDK资源

```
/**
     * 释放SDK
     * @return 参考错误码
     */
    -(int) nui_tts_release;
```

- NeoNuiTtsDelegate 事件代理
  - onNuiTtsUserdataCallback：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，语音数据来自App的录音
     * @param info: 在使用时间戳功能时返回时间戳结果，json格式
     * @param info_len: info字段的数据长度
     * @param buffer: 合成的语音数据
     * @param len: 合成的语音长度
     * @param taskid: 本次合成的任务id
     */
    - (void)onNuiTtsUserdataCallback:(char*)info infoLen:(int)info_len buffer:(char*)buffer
len:(int)len taskId:(char*)task_id;
```

  - onNuiTtsEventCallback: 事件回调。

```
/**
     * SDK主要事件回调
     * @param event: 回调事件，参考接口说明
     * @param taskid: 本次合成的任务id
     * @param code: 参考错误码，TTS_EVENT_ERROR时有效
     */
    - (void)onNuiTtsEventCallback:(NuiSdkTtsEvent)event taskId:(char*)taskid code:(int)code
;
```

- NuiSdkTtsEvent事件列表：

名称	说明
TTS_EVENT_START	语音合成开始，准备播放。
TTS_EVENT_END	语音合成播放结束
TTS_EVENT_CANCEL	取消语音合成
TTS_EVENT_PAUSE	语音合成暂停
TTS_EVENT_RESUME	语音合成恢复
TTS_EVENT_ERROR	语音合成发生错误

- onNuiTtsVolumeCallback：合成数据音量回调。

```
/**
     * SDK主要事件回调
     * @param volume: 合成的语音的能量，可用于动画显示等
     * @param taskid: 本次合成的任务id
     */
    -(void)onNuiTtsVolumeCallback:(int)volume taskId:(char*)task_id;
```

调用步骤

1. 初始化SDK和播放组件。
2. 根据业务需要设置参数。
3. 调用nui_tts_play进行播放。
4. 在合成数据回调中，将数据写入播放器进行播放，建议使用流式播放。
5. 收到语音合成结束的回调。

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。

- 语音合成初始化

```
NSString * initParam = [self genInitParams];
[_nui nui_tts_initialize:[initParam UTF8String] logLevel:LOG_LEVEL_VERBOSE saveLog: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。

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

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

- 根据需求设置参数

```
//可以设置发音人、语调和语速等参数，参数列表参见接口说明。
[self.nui nui_tts_set_param:"font_name" value:"xiaoyun"];
```

- 启动语音合成

```
[self.nui nui_tts_play:"1" taskId:"" text:[content UTF8String]];
```

- 回调处理
  - onNuiTtsEventCallback：语音合成事件回调，根据语音合成状态控制播放器。

```
- (void)onNuiTtsEventCallback:(NuiSdkTtsEvent)event taskId:(char*)taskid code:(int)code
{
    TLog(@"onNuiTtsEventCallback event[%d]", event);
    if (event == TTS_EVENT_START) {
        loop_in = TTS_EVENT_START;
        dispatch_async(dispatch_get_main_queue(), ^{
            [self->_voicePlayer play];
        });
    } else if (event == TTS_EVENT_END) {
        loop_in = TTS_EVENT_END;
        dispatch_async(dispatch_get_main_queue(), ^{
            [self->_voicePlayer drain];
        });
    }
}
```

  - onNuiTtsUserdataCallback：语音合成数据回调，将回调中的合成数据写入播放器进行播放。

```
- (void)onNuiTtsUserdataCallback:(char*)info wordIdx:(int)info_len buffer:(char*)buffer
len:(int)len taskId:(char*)task_id {
    TLog(@"onnuiTtsUserdataCallback played text %s index %d", text, word_idx);
    [_voicePlayer write:(char*)buffer Length:(unsigned int)len];
}
```

- 结束语音合成

```
[self.nui nui_tts_cancel:NULL];
```

4.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。

下载安装

1. 从Maven服务器下载最新版本SDK。

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

2. 解压该ZIP文件。
3. 在pom.xml文件所在的目录运行 `mvn package` ，会在target目录生成可执行JAR：nls-example-tts-2.0.0-jar-with-dependencies.jar。
4. 将JAR包拷贝到您应用所在的服务器，用于快速验证及压测服务。
5. 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-tts-2.0.0-jar-with-dependencies.jar com.alibaba.nls.client.SpeechSynthesizerDemo
```

6. 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` ，并发数根据您的购买情况进行选择。

```
java -jar nls-example-tts-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。
- SpeechSynthesizer：语音合成处理类，通过该接口设置请求参数，发送请求。非线程安全。
- SpeechSynthesizerListener：语音合成监听类，监听返回结果。非线程安全。需要实现如下两个抽象方法：

```
/**
 * 接收语音合成二进制数据
 */
abstract public void onMessage(ByteBuffer message);
/**
 * 语音合成结束事件通知
 *
 * @param response
 */
abstract public void onComplete(SpeechSynthesizerResponse response);
```

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。
- SpeechSynthesizer对象不可重复使用，一个语音合成任务对应一个SpeechSynthesizer对象。例如，N个文本要进行N次语音合成任务，创建N个SpeechSynthesizer对象。
- SpeechSynthesizerListener对象和SpeechSynthesizer对象是一一对应的，不能将一个SpeechSynthesizerListener对象设置到多个SpeechSynthesizer对象中，否则不能将各语音合成任务区分开。
- 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。

代码示例

说明

- 示例中使用SDK内置的默认语音合成服务的外网访问服务URL，如果您使用位于阿里云上海地域的ECS，且需要通过内网访问服务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);
  ```

- 示例中将合成的音频保存在文件中，如果您需要播放音频且对实时性要求较高，建议使用流式播放，即边接收语音数据边播放，减少延时。

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import com.alibaba.nls.client.protocol.NlsClient;
import com.alibaba.nls.client.protocol.OutputFormatEnum;
import com.alibaba.nls.client.protocol.SampleRateEnum;
import com.alibaba.nls.client.protocol.tts.SpeechSynthesizer;
import com.alibaba.nls.client.protocol.tts.SpeechSynthesizerListener;
import com.alibaba.nls.client.protocol.tts.SpeechSynthesizerResponse;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
/**
 * 此示例演示了：
 *       语音合成API调用。
 *       动态获取token。
 *       流式合成TTS。
 *       首包延迟计算。
 */
public class SpeechSynthesizerDemo {
    private static final Logger logger = LoggerFactory.getLogger(SpeechSynthesizerDemo.clas
s);
    private static long startTime;
    private String appKey;
    NlsClient client;
    public SpeechSynthesizerDemo(String appKey, String accessKeyId, String accessKeySecret)
{
        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，使用时注意在accessToken.getExpireTime()过期前再次获取。
        AccessToken accessToken = new AccessToken(accessKeyId, accessKeySecret);
        try {
            accessToken.apply();
            System.out.println("get token: " + accessToken.getToken() + ", expire time: " +
accessToken.getExpireTime());
            client = new NlsClient(accessToken.getToken());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public SpeechSynthesizerDemo(String appKey, String accessKeyId, String accessKeySecret,
String url) {
        this.appKey = appKey;
        AccessToken accessToken = new AccessToken(accessKeyId, accessKeySecret);
        try {
            accessToken.apply();
            System.out.println("get token: " + accessToken.getToken() + ", expire time: " +
accessToken.getExpireTime());
            if(url.isEmpty()) {
                client = new NlsClient(accessToken.getToken());
            }else {
                client = new NlsClient(url, accessToken.getToken());
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    private static SpeechSynthesizerListener getSynthesizerListener() {
        SpeechSynthesizerListener listener = null;
        try {
            listener = new SpeechSynthesizerListener() {
                File f=new File("tts_test.wav");
                FileOutputStream fout = new FileOutputStream(f);
                private boolean firstRecvBinary = true;
                //语音合成结束
                @Override
```

```
                public void onComplete(SpeechSynthesizerResponse response) {
                    //调用onComplete时表示所有TTS数据已接收完成，因此为整个合成数据的延迟。该延迟可
能较大，不一定满足实时场景。
                    System.out.println("name: " + response.getName() +
                        ", status: " + response.getStatus()+
                        ", output file :"+f.getAbsolutePath()
                    );
                }
                //语音合成的语音二进制数据
                @Override
                public void onMessage(ByteBuffer message) {
                    try {
                        if(firstRecvBinary) {
                            //计算首包语音流的延迟，收到第一包语音流时，即可以进行语音播放，以提升响
应速度（特别是实时交互场景下）。
                            firstRecvBinary = false;
                            long now = System.currentTimeMillis();
                            logger.info("tts first latency : " + (now - SpeechSynthesizerDe
mo.startTime) + " ms");
                        }
                        byte[] bytesArray = new byte[message.remaining()];
                        message.get(bytesArray, 0, bytesArray.length);
                        fout.write(bytesArray);
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
                @Override
                public void onFail(SpeechSynthesizerResponse response){
                    //task_id是调用方和服务端通信的唯一标识，当遇到问题时需要提供task_id以便排查。
                    System.out.println(
                        "task_id: " + response.getTaskId() +
                            //状态码 20000000 表示识别成功
                            ", status: " + response.getStatus() +
                            //错误信息
                            ", status_text: " + response.getStatusText());
                }
            };
        } catch (Exception e) {
            e.printStackTrace();
        }
        return listener;
    }
    public void process() {
        SpeechSynthesizer synthesizer = null;
        try {
            //创建实例，建立连接。
            synthesizer = new SpeechSynthesizer(client, getSynthesizerListener());
            synthesizer.setAppKey(appKey);
            //设置返回音频的编码格式
            synthesizer.setFormat(OutputFormatEnum.WAV);
            //设置返回音频的采样率
            synthesizer.setSampleRate(SampleRateEnum.SAMPLE_RATE_16K);
            //发音人
```

```
            synthesizer.setVoice("siyue");
            //语调，范围是-500~500，可选，默认是0。
            synthesizer.setPitchRate(100);
            //语速，范围是-500~500，默认是0。
            synthesizer.setSpeechRate(100);
            //设置用于语音合成的文本
            synthesizer.setText("欢迎使用阿里巴巴智能语音合成服务，您可以说北京明天天气怎么样啊");
            // 是否开启字幕功能（返回相应文本的时间戳），默认不开启，需要注意并非所有发音人都支持该参
数。
            synthesizer.addCustomedParam("enable_subtitle", false);
            //此方法将以上参数设置序列化为JSON格式发送给服务端，并等待服务端确认。
            long start = System.currentTimeMillis();
            synthesizer.start();
            logger.info("tts start latency " + (System.currentTimeMillis() - start) + " ms"
);
            SpeechSynthesizerDemo.startTime = System.currentTimeMillis();
            //等待语音合成结束
            synthesizer.waitForComplete();
            logger.info("tts stop latency " + (System.currentTimeMillis() - start) + " ms")
;
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            //关闭连接
            if (null != synthesizer) {
                synthesizer.close();
            }
        }
    }
    public void shutdown() {
        client.shutdown();
    }
    public static void main(String[] args) throws Exception {
        String appKey = "您的appkey";
        String id = "您的AccessKey Id";
        String secret = "您的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);
        }
        SpeechSynthesizerDemo demo = new SpeechSynthesizerDemo(appKey, id, secret, url);
        demo.process();
        demo.shutdown();
    }
```

```
        }
    }
```

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，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的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 ◦ test1.wav ◦ test2.wav ◦ 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，SDK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
./syDemo
```

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Git Hub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speechSynthesizerDemo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。
CreateSynthesizerRequest	创建一句话识别对象，线程安全，支持高并发请求。
releaseSynthesizerRequest	销毁语音合成对象，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。
getBinaryData	获取云端返回的二进制数据。
getAllResponse	获取云端返回的识别结果。

- 识别接口

  SpeechSynthesizerRequest：语音合成请求对象，用于语音合成及长文本语音合成。

接口名	功能描述
setOnSynthesisCompleted	设置语音合成结束回调函数。
setOnChannelClosed	设置通道关闭回调函数。
setOnTaskFailed	设置错误回调函数。
setOnBinaryDataReceived	设置语音合成二进制音频数据接收回调函数。
setOnMetaInfo	设置文本对应的日志信息接收回调函数。
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。
setText	待合成音频文本内容text设置。

接口名	功能描述
setVoice	发音人voice设置。
setVolume	音量volume设置。
setFormat	输出音频编码格式Format设置。
setSampleRate	音频采样率设置。
setSpeechRate	语速设置。
setPitchRate	语调设置。
setEnableSubtitle	是否开启字幕功能。
start	启动SpeechSynthesizerRequest。
cancel	不会与服务端确认关闭，直接关闭语音合成过程。

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，并通过工单系统反馈给阿里云。
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阿里云。
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。
10000013	`The sent data is null or dataSize <= 0.`	发送错误。请检查发送参数是否正确。

错误码	错误描述	解决方案
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SDK内置的默认外网访问服务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speechSynthesizerRequest的对象中设置内网访问的URL。

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

- 示例中将合成的音频保存在文件中，如果您需要播放音频且对实时性要求较高，建议使用流式播放，即边接收语音数据边播放，减少延时，而无需等待合成结束后再处理语音流。
- 完整示例，参见SDK压缩包中demo目录的speechSynthesizerDemo.cpp文件。

```
#include<string.h>
#include<unistd.h>
#include<pthread.h>
#include<stdlib.h>
#include<ctime>
#include<string>
#include<iostream>
#include<vector>
#include<fstream>
#include<sys/time.h>
#include"nlsClient.h"
#include"nlsEvent.h"
#include"nlsToken.h"
#include "speechSynthesizerRequest.h"

using namespace AlibabaNlsCommon;
using AlibabaNls::NlsClient;
using AlibabaNls::NlsEvent;
using AlibabaNls::LogDebug;
using AlibabaNls::LogInfo;
using AlibabaNls::SpeechSynthesizerRequest;

// 自定义线程参数。
struct ParamStruct {
```

```
struct ParamStruct {
    std::string text;
    std::string token;
    std::string appkey;
    std::string audioFile;
};
// 自定义事件回调参数。
struct ParamCallBack {
    std::string binAudioFile;
    std::ofstream audioFile;
    uint64_t startMs;
};

//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，并更新该全局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;
uint64_t getNow() {
    struct timeval now;
    gettimeofday(&now, NULL);
    return now.tv_sec * 1000 * 1000 + now.tv_usec;
}
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 在接收到云端返回合成结束消息时，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。
//@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 OnSynthesisCompleted(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("OnSynthesisCompleted: %s\n", tmpParam->binAudioFile.c_str());
    // 获取消息的状态码，成功为0或者20000000，失败时对应失败的错误码。
    // 当前任务的task id，方便定位问题，作为和服务端交互的唯一标识建议输出。
    printf("OnSynthesisCompleted: status code=%d, task id=%s\n", cbEvent->getStatusCode(), cbEvent->getTaskId());
    // 获取服务端返回的全部信息。
```

```
    //printf("OnSynthesisCompleted: all response=%s\n", cbEvent->getAllResponse());
}

//@brief 合成过程发生异常时，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。
//@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 OnSynthesisTaskFailed(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("OnSynthesisTaskFailed: %s\n", tmpParam->binAudioFile.c_str());
    // 当前任务的task id。
    printf("OnSynthesisTaskFailed: status code=%d, task id=%s, error message=%s\n",
cbEvent->getStatusCode(), cbEvent->getTaskId(), cbEvent->getErrorMessage());
}

//@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 OnSynthesisChannelClosed(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("OnSynthesisChannelClosed: %s\n", tmpParam->binAudioFile.c_str());
    printf("OnSynthesisChannelClosed: %s\n", cbEvent->getAllResponse());
    tmpParam->audioFile.close();
    delete tmpParam; //识别流程结束，释放回调参数。
}

//@brief 文本上报服务端后，收到服务端返回的二进制音频数据，SDK内部线程通过BinaryDataRecved事件上报给
用户。
//@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 OnBinaryDataRecved(NlsEvent* cbEvent, void* cbParam) {
    ParamCallBack* tmpParam = (ParamCallBack*)cbParam;
    if(tmpParam->startMs > 0 ) {
        // 重要提示：一旦获取到语音流，如第一次从服务端返回合成语音流，即可以进行播放或者其他处理。示例
为保存到本地文件。
        // 第一次收到语音流数据，计算TTS合成首包延迟。另外此处计算首包延迟时也包括了start操作（即本程
序连接公共云服务端的时间），而该时间受不同网络因素影响可能有较大差异。
        uint64_t now = getNow();
        printf("first latency = %lld ms, task id = %s\n",
                (now - tmpParam->startMs) / 1000, cbEvent->getTaskId());
        tmpParam->startMs = 0;
    }
     // 演示如何打印/使用用户自定义参数示例。
    printf("OnBinaryDataRecved: %s\n", tmpParam->binAudioFile.c_str());
    const std::vector<unsigned char>& data = cbEvent->getBinaryData(); // getBinaryData()
：获取文本合成的二进制音频数据。
    printf("OnBinaryDataRecved: status code=%d, task id=%s, data size=%d\n",
cbEvent->getStatusCode(), cbEvent->getTaskId(), data.size());
    // 以追加形式将二进制音频数据写入文件。
    if (data.size() > 0) {
        tmpParam->audioFile.write((char*)&data[0], data.size());
    }
```

```
}

//@brief 返回tts文本对应的日志信息，增量返回对应的字幕信息。
//@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 OnMetaInfo(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("OnBinaryDataRecved: %s\n", tmpParam->binAudioFile.c_str());
    printf("OnMetaInfo: task id=%s, respose=%s\n", cbEvent->getTaskId(), cbEvent->getAllRes
ponse());
}

 // 工作线程
void* pthreadFunc(void* arg) {
    // 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;
    }
    // 1：初始化自定义回调参数
   ParamCallBack* cbParam = new ParamCallBack;
   cbParam->binAudioFile = tst->audioFile;
  cbParam->audioFile.open(cbParam->binAudioFile.c_str(), std::ios::binary | std::ios::out);

   // 2：创建语音识别SpeechSynthesizerRequest对象
    SpeechSynthesizerRequest* request = NlsClient::getInstance()->createSynthesizerRequest(
);
    if (request == NULL) {
        printf("createSynthesizerRequest failed.\n");
        cbParam->audioFile.close();
        return NULL;
    }
    request->setOnSynthesisCompleted(OnSynthesisCompleted, cbParam); // 设置音频合成结束回调函
数
    request->setOnChannelClosed(OnSynthesisChannelClosed, cbParam); // 设置音频合成通道关闭回
调函数
    request->setOnTaskFailed(OnSynthesisTaskFailed, cbParam); // 设置异常失败回调函数
    request->setOnBinaryDataReceived(OnBinaryDataRecved, cbParam); // 设置文本音频数据接收回调
函数
    request->setOnMetaInfo(OnMetaInfo, cbParam); // 设置字幕信息
    request->setAppKey(tst->appkey.c_str());
    request->setText(tst->text.c_str()); // 设置待合成文本，必填参数。文本内容为UTF-8编码。
    request->setVoice("siqi");              // 发音人，可选参数，默认是xiaoyun。
    request->setVolume(50);              // 音量，范围是0~100，可选参数，默认50。
    request->setFormat("wav");             // 音频编码格式，可选参数，默认是WAV。支持：PCM/WAV/M
P3。
    request->setSampleRate(8000);          // 音频采样率，支持8000/16000。可选参数，默认是16000
。
    request->setSpeechRate(0);              // 语速，范围是-500~500。可选参数，默认是0。
    request->setPitchRate(0);              // 语调，范围是-500~500。可选参数，默认是0。
    //request->setEnableSubtitle(true);       //是否开启字幕，可选。注意并不是所有发音人都支持字幕
功能。
```

```
    request->setToken(tst->token.c_str()); // 设置账号校验token，必填参数。
    cbParam->startMs = getNow();

    // 3：start()为异步操作。成功返回BinaryRecv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()->releaseSynthesizerRequest(request); // 调用start()失败，释
放request对象。
        cbParam->audioFile.close();
        return NULL;
    }
    //4：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();      //5：识别结束，释放request对象。
    NlsClient::getInstance()->releaseSynthesizerRequest(request);
    return NULL;
}

// 合成单个文本数据
int speechSynthesizerFile(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;
        }
    }
    ParamStruct pa;
    pa.token = g_token;
    pa.appkey = appkey;
    // 说明：Windows平台下，合成文本中如果包含中文，请将本CPP文件设置为带签名的UTF-8编码或者GB2312编
码。
    pa.text = "今天天气很棒，适合去户外旅行.";
    pa.audioFile = "syAudio.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;
}

 // 合成多个文本数据。
// SDK多线程指一个文本数据对应一个线程，非一个文本数据对应多个线程。
// 示例代码为同时开启2个线程合成2个文件。
// 免费用户并发连接不能超过2个。
#define AUDIO_TEXT_NUMS 2
#define AUDIO_TEXT_LENGTH 64
#define AUDIO_FILE_NAME_LENGTH 32
int speechSynthesizerMultFile(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;
        }
    }
    const char syAudioFiles[AUDIO_TEXT_NUMS][AUDIO_FILE_NAME_LENGTH] =
    {
        "syAudio0.wav",
        "syAudio1.wav"
    };
    const char texts[AUDIO_TEXT_NUMS][AUDIO_TEXT_LENGTH] =
    {
        "今日天气真不错，我想去操作踢足球.",
        "明天有大暴雨，还是宅在家里看电影吧."
    };
    ParamStruct pa[AUDIO_TEXT_NUMS];
    for (int i = 0; i < AUDIO_TEXT_NUMS; i ++) {
        pa[i].token = g_token;
        pa[i].appkey = appkey;
        pa[i].text = texts[i];
        pa[i].audioFile = syAudioFiles[i];
    }
    std::vector<pthread_t> pthreadId(AUDIO_TEXT_NUMS);      // 启动工作线程，同时识别音频文件。
    for (int j = 0; j < AUDIO_TEXT_NUMS; j++) {
        pthread_create(&pthreadId[j], NULL, &pthreadFunc, (void *)&(pa[j]));
    }
    for (int j = 0; j < AUDIO_TEXT_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-Synthesizer.txt。LogDebug表示输出所
有级别日志。
    int ret = NlsClient::getInstance()->setLogConfig("log-synthesizer", LogDebug);
    if (-1 == ret) {
        printf("set log failed\n");
        return -1;
    }
     //启动工作线程
     NlsClient::getInstance()->startWorkThread(4);      //合成单个文本
    speechSynthesizerFile(appkey.c_str());      //合成多个文本
    //speechSynthesizerMultFile(appkey.c_str());
    //所有工作完成，进程退出前，释放nlsClient->releaseInstance()非线程安全
```

```
    //所有工作完成，进程退出前，释放nlsClient。releaseInstance()非线程安全。
    NlsClient::releaseInstance();
    return 0;
}
```

6.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

语音合成对应的类为**NlsSpeechSynthesizer**，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。 • 若未已填写**token**字段，则需要填写该字段。 • 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。 • 若未已填写**token**字段，则需要填写该字段。 • 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_metainfo	Function	如果**start**方法中通过**ex**参数传递**enable_subtitle**，则会返回对应字幕信息。回调参数包含以下两种： • JSON形式的字符串 • 用户自定义参数 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_data	Function	当存在合成数据后的回调参数。回调参数包含以下两种： • 对应**start**方法中**aformat**的二进制音频数据 • 用户自定义参数 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： • JSON形式的字符串 • 用户自定义参数 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_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

同步开始语音合成，如果wait_complete为True（默认），则会阻塞直到所有音频合成完毕（on_completed返回之后）返回，否则会立即返回。

- 参数说明

参数	类型	参数说明
text	String	要合成的文字。
aformat	String	合成出来音频的格式，默认为pcm。
voice	String	发音人，默认为xiaoyun。
sample_rate	Integer	识别音频采样率，默认值：16000Hz。
volume	Integer	音量大小，取值范围0~100，默认值：50。
speech_rate	Integer	语速，取值范围-500~500，默认值：0。
pitch_rate	Integer	语调，取值范围-500~500，默认值：0。
wait_complete	Boolean	是否阻塞到合成完成。
start_timeout	Integer	和云端连接建立超时，默认值：10秒。
completed_timeout	Integer	从连接建立到合成完成超时，默认值：60秒

参数	类型	参数说明
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. 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

代码示例

> **说明**
>
> - 本示例中将合成的音频保存在文件中，如果您需要播放音频且对实时性要求较高，建议使用流式播放，即边接收语音数据边播放，减少延时。
> - 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"



TEXT='大壮正想去摘取花瓣，谁知阿丽和阿强突然内讧，阿丽拿去手枪向树干边的阿强射击，两声枪响，阿强直接倒入水中'
```

```
#以下代码会根据上述TEXT文本反复进行语音合成
class TestTts:
    def __init__(self, tid, test_file):
        self.__th = threading.Thread(target=self.__test_run)
        self.__id = tid
        self.__test_file = test_file

    def start(self, text):
        self.__text = text
        self.__f = open(self.__test_file, "wb")
        self.__th.start()

    def test_on_metainfo(self, message, *args):
        print("on_metainfo message=>{}".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))
        try:
            self.__f.close()
        except Exception as e:
            print("close file failed since:", e)

    def test_on_data(self, data, *args):
        try:
            self.__f.write(data)
        except Exception as e:
            print("write data failed:", e)

    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))
        tts = nls.NlsSpeechSynthesizer(
                    url=URL,
                    akid=AKID,
                    aksecret=AKKEY,
                    appkey=APPKEY,
                    on_metainfo=self.test_on_metainfo,
                    on_data=self.test_on_data,
                    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 = tts.start(self.__text, voice="ailun")
            print("{}: tts done with result:{}".format(self.__id, r))
            time.sleep(1)
```

```
            time.sleep(1)

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

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

7.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。

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的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<version>-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**speechSynthesizerDemo**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。其中 `{ProjectRoot}` 为SDK源码路径。
7. 若需要对生成的库文件、测试exe、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。

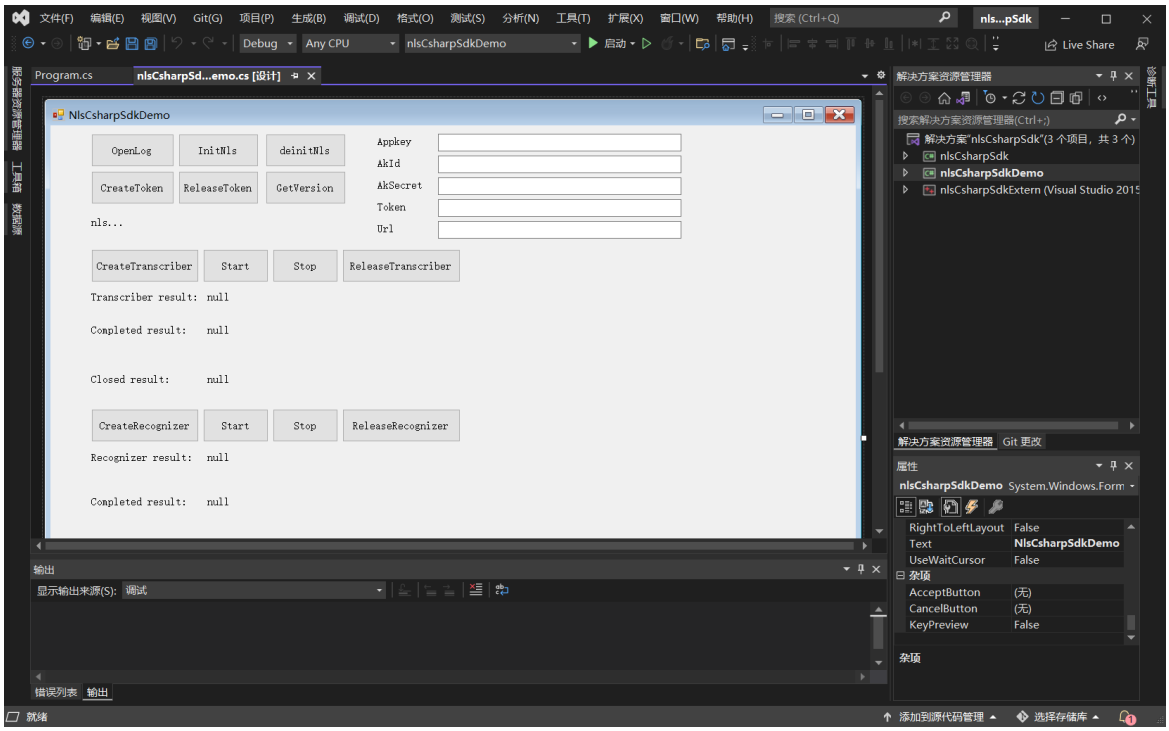
C#编译

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：

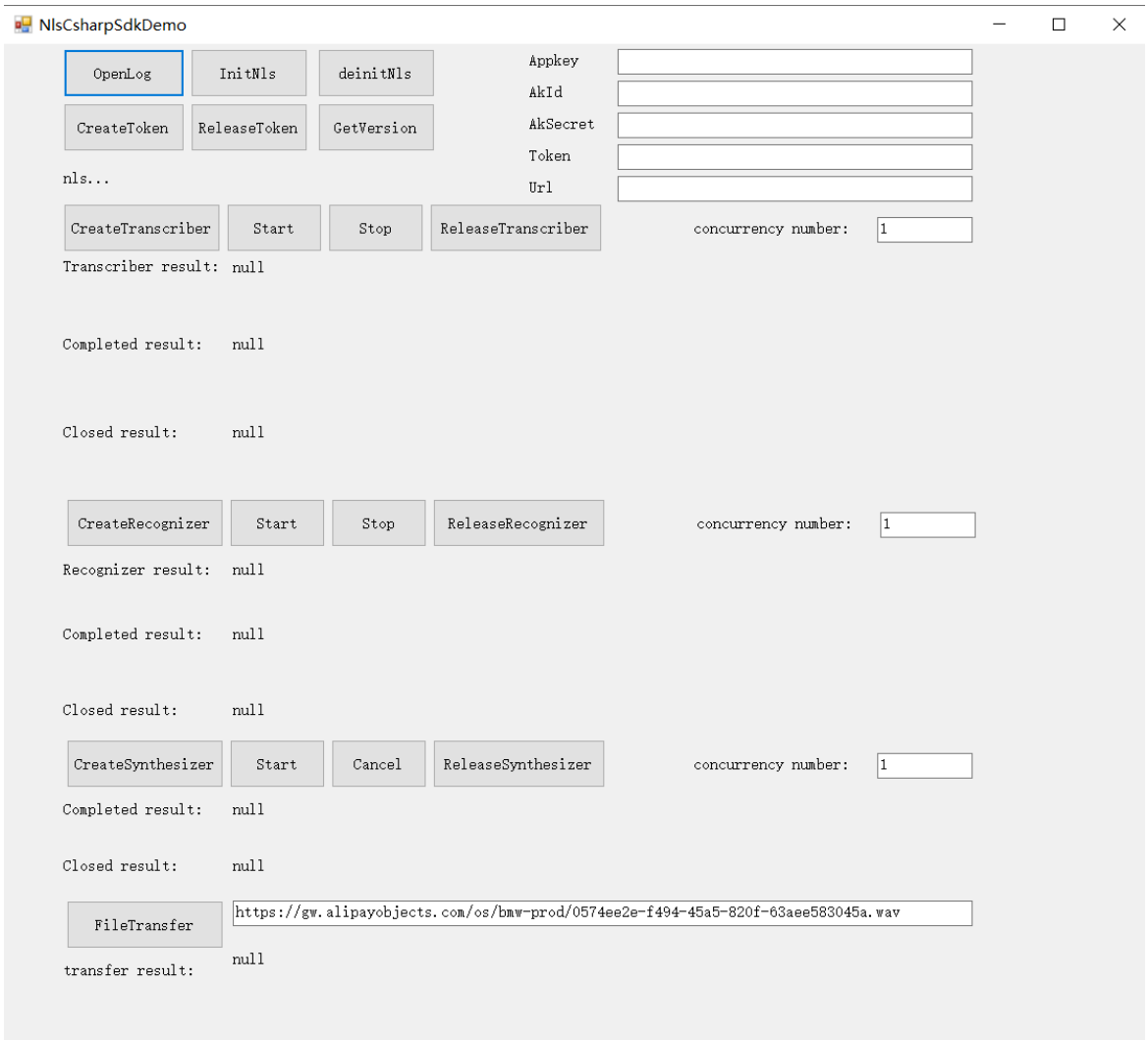
`{ProjectRoot}\nlsCsharpSdk\nlsCsharpSdkDemo\bin\Debug` 。即分别将下方动态库拷贝至 `{ProjectRoot}\nlsCsharpSdk\nlsCsharpSdkDemo\bin\Debug` 目录下。

> **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CreateSynthesizer创建语音合成请求，然后单击Start开始工作，会在nlsCsharpSdkDemo.exe所在路径生成保存音频数据的<taskId>.pcm文件，单击Stop后停止工作，然后单击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。
CreateSynthesizerRequest	创建语音合成对象，线程安全，支持高并发请求。
CreateSynthesizerRequest	销毁语音合成对象，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。

- 识别接口

  SpeechSynthesizerRequest：语音合成请求对象，用于语音合成及长文本语音合成。

接口名	功能描述
SetOnSynthesisCompleted	设置语音合成结束回调函数。
SetOnChannelClosed	设置通道关闭回调函数。
SetOnTaskFailed	设置错误回调函数。

接口名	功能描述
SetOnBinaryDataReceived	设置语音合成二进制音频数据接收回调函数。
SetOnMetaInfo	设置文本对应的日志信息接收回调函数。
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。
SetText	待合成音频文本内容text设置。
SetVoice	发音人voice设置。
SetVolume	音量volume设置。
SetFormat	输出音频编码格式Format设置。
SetSampleRate	音频采样率设置。
SetSpeechRate	语速设置。
SetPitchRate	语调设置。
SetEnableSubtitle	是否开启字幕功能。
Start	启动SpeechSynthesizerRequest。
Cancel	不会与服务端确认关闭，直接关闭语音合成过程。

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。
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。

错误码	错误描述	解决方案
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。
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内置的默认语音合成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SpeechSynthesizerRequest的对象中设置内网访问的URL。

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

namespace nlsCsharpSdkDemo
{
    public partial class nlsCsharpSdkDemo : Form
    {
        private NlsClient nlsClient;
        private SpeechSynthesizerRequest syPtr;
        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 string cur_sy_completed;
        static string cur_sy_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;
                }

                if (cur_sr_result != null && cur_sr_result.Length > 0)
                {
                    srResult.Text = cur_sr_result;
                }
                if (cur_sr_completed != null && cur_sr_completed.Length > 0)
                {
                    srCompleted.Text = cur_sr_completed;
```

```
                srCompleted.Text = cur_sr_completed;
            }
            if (cur_sr_closed != null && cur_sr_closed.Length > 0)
            {
                srClosed.Text = cur_sr_closed;
            }

            if (cur_sy_completed != null && cur_sy_completed.Length > 0)
            {
                syCompleted.Text = cur_sy_completed;
            }
            if (cur_sy_closed != null && cur_sy_closed.Length > 0)
            {
                syClosed.Text = cur_sy_closed;
            }
            Thread.Sleep(200);
        }
    }

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

        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 SynthesizerCallback
    private CallbackDelegate DemoOnBinaryDataReceived =
        (ref NLS_EVENT_STRUCT e) =>
        {
            FileStream fs;
            String filename = e.taskId + ".pcm";
            if (File.Exists(filename))
            {
                fs = new FileStream(filename, FileMode.Append, FileAccess.Write);
            }
            else
            {
                fs = new FileStream(filename, FileMode.Create, FileAccess.Write);
            }
            fs.Write(e.binaryData, 0, e.binaryDataSize);
            fs.Close();
        };
    private CallbackDelegate DemoOnSynthesisClosed =
        (ref NLS_EVENT_STRUCT e) =>
        {
            System.Diagnostics.Debug.WriteLine("DemoOnSynthesisClosed = {0}", e.msg);
            cur_sy_closed = "msg : " + e.msg;
```

```
            };
        private CallbackDelegate DemoOnSynthesisTaskFailed =
            (ref NLS_EVENT_STRUCT e) =>
            {
                System.Diagnostics.Debug.WriteLine("DemoOnSynthesisTaskFailed = {0}", e.msg
);
                cur_sy_completed = "msg : " + e.msg;
            };
        private CallbackDelegate DemoOnSynthesisCompleted =
            (ref NLS_EVENT_STRUCT e) =>
            {
                System.Diagnostics.Debug.WriteLine("DemoOnSynthesisCompleted = {0}", e.msg)
;
                cur_sy_completed = "result : " + e.msg;
            };
        private CallbackDelegate DemoOnMetaInfo =
            (ref NLS_EVENT_STRUCT e) =>
            {
                System.Diagnostics.Debug.WriteLine("DemoOnMetaInfo = {0}", e.msg);
                //cur_sy_completed = "metaInfo : " + e.msg;
            };
        #endregion

        // create synthesizer
        private void button12_Click(object sender, EventArgs e)
        {
            syPtr = nlsClient.CreateSynthesizerRequest(TtsVersion.ShortTts);
            if (syPtr.native_request != IntPtr.Zero)
            {
                nlsResult.Text = "CreateSynthesizerRequest Success";
            }
            else
            {
                nlsResult.Text = "CreateSynthesizerRequest Failed";
            }
            cur_sy_closed = "null";
            cur_sy_completed = "null";
        }

        // start synthesizer
        private void button10_Click(object sender, EventArgs e)
        {
            int ret = -1;
            if (syPtr.native_request != IntPtr.Zero)
            {
                syPtr.SetAppKey(syPtr, appKey);
                syPtr.SetToken(syPtr, token);
                syPtr.SetUrl(syPtr, url);
                syPtr.SetText(syPtr, "今天天气真不错，我想去操场踢足球。");
                syPtr.SetVoice(syPtr, "siqi");
                syPtr.SetVolume(syPtr, 50);
                syPtr.SetFormat(syPtr, "wav");
                syPtr.SetSampleRate(syPtr, 16000);
                syPtr.SetSpeechRate(syPtr, 0);
                syPtr.SetPitchRate(syPtr, 0);
```

```
                syPtr.SetPitchRate(syPtr, 0);
                syPtr.SetEnableSubtitle(syPtr, true);

                syPtr.SetOnSynthesisCompleted(syPtr, DemoOnSynthesisCompleted, IntPtr.Zero)
;
                syPtr.SetOnBinaryDataReceived(syPtr, DemoOnBinaryDataReceived, IntPtr.Zero)
;
                syPtr.SetOnTaskFailed(syPtr, DemoOnSynthesisTaskFailed, IntPtr.Zero);
                syPtr.SetOnChannelClosed(syPtr, DemoOnSynthesisClosed, IntPtr.Zero);
                syPtr.SetOnMetaInfo(syPtr, DemoOnMetaInfo, IntPtr.Zero);

                ret = syPtr.Start(syPtr);
            }

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

        // cancel synthesizer
        private void button9_Click(object sender, EventArgs e)
        {
            int ret = -1;
            if (syPtr.native_request != IntPtr.Zero)
                ret = syPtr.Cancel(syPtr);

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

        private void btnSYrelease_Click(object sender, EventArgs e)
        {
            if (syPtr.native_request != IntPtr.Zero)
            {
                nlsClient.ReleaseSynthesizerRequest(syPtr);
                syPtr.native_request = IntPtr.Zero;
                nlsResult.Text = "ReleaseSynthesizerRequest Success";
            }
            else
            {
                nlsResult.Text = "ReleaseSynthesizerRequest is nullptr";
            }
            cur_sy_closed = "null";
            cur_sy_completed = "null";
```

```
            }
        }
    }
```

8.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	访问的公有云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。 • 若未填写**Token**字段，则需要填写该字段。 • 若已填写**Token**字段，则该字段可不填写。
Akkey	String	阿里云账号AccessKey Secret。 • 若未填写**Token**字段，则需要填写该字段。 • 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	访问的公有云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。

语音合成

1.SpeechSynthesisStartParam

实时语合成参数。

参数	类型	参数说明
Voice	String	发音人，默认值：“xiaoyun”。
Format	String	音频格式，默认使用WAV。
SampleRate	Integer	采样率，默认值：16000 Hz。

参数	类型	参数说明
Volume	Integer	音量，范围为0~100，默认值：50。
SpeechRate	Integer	语速，范围为-500~500，默认值：0。
PitchRate	Integer	音高，范围为-500~500，默认值：0。
EnableSubtitle	Boolean	是否开启字幕功能，默认不开启。 • true：开启。 • false：不开启。

2.func DefaultSpeechSynthesisParam() SpeechSynthesisStartParam

创建一个默认的语音合成参数。

- 参数说明：无。
- 返回值：

  SpeechSynthesisStartParam：默认参数。

3.func NewSpeechSynthesis(...) (*SpeechSynthesis, error)

创建一个新的语音合成对象。

- 参数说明：

参数	类型	参数说明
config	*ConnectionConfig	参见建立连接相关内容。
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。
synthesisresult	func([]byte, interface{})	语音合成数据回调参数。
metainfo	func(string, interface{})	字幕数据回调，需要参数中**EnableSubtitle**为true。
completed	func(string, interface{})	合成完毕结果回调参数。

参数	类型	参数说明
closed	func(interface{})	连接断开回调参数。
param	interface{}	用户自定义参数。

- 返回值：无。

4.func (tts *SpeechSynthesis) Start(text string, param SpeechSynthesisStartParam, extra map[string]interface{}) (chan bool, error)

给定文本和参数进行语音合成。

- 参数说明：

参数	类型	参数说明
text	String	待合成文本。
param	SpeechRecognitionStartParam	实时合成参数。
extra	map[string]interface{}	额外 `key:value` 参数。

- 返回值：
  - chan bool：语音合成完成通知管道。
  - error：错误异常。

5.func (tts *SpeechSynthesis) Shutdown()

强制停止语音合成。

- 参数说明：无。
- 返回值：无。

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。 • true：禁止。 • 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。 • true：允许。 • 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。

方法名	方法说明
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"
        "io"
        "log"
        "os"
        "os/signal"
        "sync"
        "time"

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

const (
    AKID  = "Your AKID"
```

```
        AKKEY = "Your AKKEY"
        //online key
        APPKEY = "Your APPKEY"
        TOKEN  = "Your TOKEN"
)

type TtsUserParam struct {
        F           io.Writer
        Logger      *nls.NlsLogger
}

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

        p.Logger.Println("TaskFailed:", text)
}

func onSynthesisResult(data []byte, param interface{}) {
        p, ok := param.(*TtsUserParam)
        if !ok {
                log.Default().Fatal("invalid logger")
                return
        }
        p.F.Write(data)
}

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

        p.Logger.Println("onCompleted:", text)
}


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

        p.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(60 * time.Second):
                {
                        logger.Println("Wait timeout")
                        return errors.New("wait timeout")
                }
        }
        return nil
}

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

const (
        TEXT = "你好小德，今天天气怎么样。"
)

func testMultiInstance(num int) {
        param := nls.DefaultSpeechSynthesisParam()
        config, _ := nls.NewConnectionConfigWithAKInfoDefault(nls.DEFAULT_URL, APPKEY, AKID
, AKKEY)
        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)
                        fname := fmt.Sprintf("ttsdump%d.wav", id)
                        ttsUserParam := new(TtsUserParam)
                        fout, err := os.OpenFile(fname, os.O_RDWR|os.O_TRUNC|os.O_CREATE, 0
666)
                        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)
                        ttsUserParam.F = fout
                        ttsUserParam.Logger = logger
      tts, err := nls.NewSpeechSynthesis(config, logger,
                                onTaskFailed, onSynthesisResult, nil,
                                onCompleted, onClose, ttsUserParam)
                        if err != nil {
                                logger.Fatalln(err)
                                return
                        }

                        for {
                                lk.Lock()
```

```
                                lk.Lock()
                                reqNum++
                                lk.Unlock()
                                logger.Println("SR start")
                                ch, err := tts.Start(TEXT, param, nil)
                                if err != nil {
                                        lk.Lock()
                                        fail++
                                        lk.Unlock()
                                        tts.Shutdown()
                                        continue
                                }

                                err = waitReady(ch, logger)
                                if err != nil {
                                        lk.Lock()
                                        fail++
                                        lk.Unlock()
                                        tts.Shutdown()
                                        continue
                                }
                                logger.Println("Synthesis done")
                                tts.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)
}
```

9.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"
```

语音合成

Class: SpeechSynthesizer

SpeechSynthesizer类用于进行语音合成。

- 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(voice)

返回一个默认的推荐参数，其中voice为您自行提供，采样率为16000 Hz，格式为WAV，音量50，语速语调皆为0，不开启字幕。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：

参数	类型	参数说明
voice	String	发音人。

- 返回值：

  object类型对象，字段如下：

```
{
    "voice": voice,
    "format": "wav",
    "sample_rate": 16000,
    "volume": 50,
    "speech_rate": 0,
    "pitch_rate": 0,
    "enable_subtitle": false
}
```

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
meta	字幕回调。	1	String类型，字幕信息。
data	合成音频回调。	1	Buffer类型，合成音频数据。
completed	语音合成完成。	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。

- 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	语音合成参数。
enablePing	Boolean	是否自动向云端发送ping请求，默认false。 • true：发送。 • 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。

代码示例

```
"use strict"

require('log-timestamp')(`${process.pid}`)
const fs = require("fs")
const Nls = require("alibabacloud-nls")
const sleep = (waitTimeInMs) => new Promise(resolve => setTimeout(resolve, waitTimeInMs))
const util = require("util")
const readline = require("readline")
```

```
const readline = require("readline")
const args = process.argv.slice(2)
//const Memwatch = require("node-memwatch-new")

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

let b1 = []
let loadIndex = 0
//let hd = new Memwatch.HeapDiff()
let needDump = true

async function runOnce(line) {
  console.log(`speak: ${line}`)
  loadIndex++

  //let dumpFile = fs.createWriteStream(`${process.pid}.wav`, {flags:"w"})
  let tts = new Nls.SpeechSynthesizer({
    url: URL,
    appkey:APPKEY,
    token:TOKEN
  })

  tts.on("meta", (msg)=>{
    console.log("Client recv metainfo:", msg)
  })

  tts.on("data", (msg)=>{
    console.log(`recv size: ${msg.length}`)
    //console.log(dumpFile.write(msg, "binary"))
  })

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

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

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

  let param = tts.defaultStartParams()
  param.text = line
  param.voice = "aixia"
  try {
    await tts.start(param, true, 6000)
  } catch(error) {
    console.log("error on start:", error)
    return
  } finally {
    //dumpFile.end()
```

```
    //dumpFile.end()
  }
  console.log("synthesis done")
  await sleep(2000)
}

async function test() {
  console.log("load test case:", args[0])
  const fileStream = fs.createReadStream(args[0])
  const rl = readline.createInterface({
    input: fileStream,
    crlfDelay: Infinity
  })

  for await (const line of rl) {
    b1.push(line)
  }

  while (true) {
    for (let text of b1) {
      await runOnce(text)
    }
  }
}

test()
```

10.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。

语音合成

Class: SpeechSynthesizer

SpeechSynthesizer类用于进行语音合成。

- 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(voice)

返回一个默认的推荐参数，其中voice为您自行提供，采样率为16000 Hz，格式为WAV，音量50，语速语调皆为0，不开启字幕。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：

参数	类型	参数说明
voice	String	发音人。

- 返回值：

  object类型对象，字段如下：

```
{
    "voice": voice,
    "format": "wav",
    "sample_rate": 16000,
    "volume": 50,
    "speech_rate": 0,
    "pitch_rate": 0,
    "enable_subtitle": false
}
```

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
meta	字幕回调。	1	String类型，字幕信息。
data	合成音频回调。	1	Arraybuffer类型，合成音频数据。
completed	语音合成完成。	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。

- 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	语音合成参数。

- 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。

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/tts/tts.js
```

```
const app = getApp()
const SpeechSynthesizer = require("../../utils/tts")
const formatTime = require("../../utils/util").formatTime
const sleep = require("../../utils/util").sleep
const getToken = require("../../utils/token").getToken
const fs = wx.getFileSystemManager()
Page({

    /**
     * 页面的初始数据
     */
    data: {
        ttsStart: false,
        ttsText: ""
    },

    /**
     * 生命周期函数--监听页面加载
     */
    onLoad: async function (options) {
        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 tts = new SpeechSynthesizer({
            url: app.globalData.URL,
            appkey:app.globalData.APPKEY,
            token:this.data.token
        })

        tts.on("meta", (msg)=>{
            console.log("Client recv metainfo:", msg)
        })

        tts.on("data", (msg)=>{
            console.log(`recv size: ${msg.byteLength}`)
            //console.log(dumpFile.write(msg, "binary"))
            if (this.data.saveFile) {
                try {
                  fs.appendFileSync(
                      this.data.saveFile,
                      msg,
                      "binary"
                  )
                  console.log(`append ${msg.byteLength}`)
                } catch (e) {
                  console.error(e)
                }
            } else {
                console.log("save file empty")
            }
```

```
        })

        tts.on("completed", async (msg)=>{
            console.log("Client recv completed:", msg)
            await sleep(500)
            fs.close({
                fd : this.data.saveFd,
                success: (res)=> {
                    let ctx = wx.createInnerAudioContext()
                    ctx.autoplay = true
                    ctx.src = this.data.saveFile
                    ctx.onPlay(() => {
                        console.log('start playing..')
                    })
                    ctx.onError((res) => {
                        console.log(res.errMsg)
                        console.log(res.errCode)
                        fs.unlink({
                            filePath: this.data.saveFile,
                            success: (res)=>{
                                console.log(`remove ${this.data.saveFile} done`)
                                this.data.saveFile = null
                                this.data.saveFd = null
                            },
                            failed: (res)=>{
                                console.log("remove failed:" + res.errMsg)
                            }
                        })
                    })
                    ctx.onEnded((res)=>{
                        console.log("play done...")
                        fs.unlink({
                            filePath: this.data.saveFile,
                            success: (res)=>{
                                console.log(`remove ${this.data.saveFile} done`)
                                this.data.saveFile = null
                                this.data.saveFd = null
                            },
                            failed: (res)=>{
                                console.log("remove failed:" + res.errMsg)
                            }
                        })
                    })
                },
                fail : (res)=>{
                    console.log("saved file error:" + res.errMsg)
                }
            })
        })

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

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

        this.data.tts = tts
    },

    /**
     * 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 () {

    },

    /**
     * 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 () {
```

```
    },
    textInput: function(e) {
        this.setData({
            ttsText:e.detail.value
        })
    },
    onTtsStart: function() {
        if (!this.data.ttsText || !this.data.tts) {
            console.log("text empty")
            wx.showToast({
                title: "文本为空",
                icon: "error",
                duration: 1000,
                mask: true
              })
            return
        }
        if (this.data.ttsStart) {
            wx.showToast({
                title: "正在合成请稍候",
                icon: "error",
                duration: 1000,
                mask: true
              })
            return
        } else {
            this.data.ttsStart = true
        }
        console.log("try to synthesis:" + this.data.ttsText)
        let save = formatTime(new Date()) + ".wav"
        let savePath = wx.env.USER_DATA_PATH + "/" + save
        console.log(`save to ${savePath}`)
        fs.open({
            filePath:savePath,
            flag : "a+",
            success: async (res)=> {
                console.log(`open ${savePath} done`)
                this.data.saveFd = res.fd
                this.data.saveFile = savePath
                let param = this.data.tts.defaultStartParams()
                param.text = this.data.ttsText
                param.voice = "aixia"
                try {
                    await this.data.tts.start(param)
                    console.log("tts done")
                    this.data.ttsStart = false
                } catch(e) {
                    console.log("tts start error:" + e)
                }
            },
            fail: (res)=> {
                console.log(`open ${savePath} failed: ${res.errMsg}`)
            }
        })
```

```
    }
})
```

11.SSML标记语言介绍

本文为您介绍SSML（Speech Synthesis Markup Language）标记语言的功能、标签使用及示例。

概述

SSML是一种基于XML的语音合成标记语言。与纯文本的合成相比，使用SSML可以充实合成的内容，为最终合成效果带来更多变化。SSML不仅控制语音合成能读什么，更能控制语音合成可以怎么读，包括控制断句分词方式、发音、速度、停顿、声调和音量等特征，甚至加入背景音乐。

> **说明**
>
> 阿里巴巴语音合成服务的SSML实现基于W3C的语音合成标记语言版本1.0。但并不支持W3C包含的所有的标记类型，而是从业务角度出发，将支持的标记类型最大程度与业务需求绑定。

使用方式

> **说明**
>
> - 目前仅中文及英文声音支持SSML功能，中文和英文所支持的SSML标签及内容也有略有差别，详情请参考文档下方各标签的介绍和示例。
> - 所有文本需放在<speak></speak>标签之内，且每个语音合成任务只能包含一个<speak></speak>标签。
> - 标签内的文字内容如果包含XML的特殊字符，需要做字符转义，常用的特殊字符对应关系如下。
> - "（双引号）对应："
> - '（撇号或单引号）对应：'
> - &（表示和的符号）对应：&
> - <（小于号）对应：<
> - >（大于号）对应：>

将带标签的文本作为text参数值，上传至语音合成服务，以Java SDK为例：

```
SpeechSynthesizer synthesizer = new SpeechSynthesizer(client, getSynthesizerListener());
String text = "<speak>请闭上眼睛休息一下<break time=\"500ms\"/>好了，请睁开眼睛。</speak>";
synthesizer.setText(text);
```

发送给语音合成服务的请求内容如下：

```
{
    "payload": {
        "volume": 50,
        "sample_rate": 16000,
        "format": "wav",
        "text": "<speak>请闭上眼睛休息一下<break time=\"500ms\"/>好了，请睁开眼睛。</speak>"
    },
    "context": {
        "sdk": {
            "name": "nls-sdk-java",
            "version": "2.0.4"
        }
    },
    "header": {
        "namespace": "SpeechSynthesizer",
        "name": "StartSynthesis",
        "message_id": "5fdf78c0dd574b6897f3cb204dd0****",
        "appkey": "fd4er4aa****",
        "task_id": "6e1be78ef5804c50a2c5a8b92de1****"
```

标签

<speak>

- 描述

  <speak>标签是所有待支持SSML标签的根节点。一切需要调用SSML标签的文本都要包含在<speak></speak>中。

- 语法

```
<speak>需要调用SSML标签的文本</speak>
```

- 属性

  <speak>标签支持如下属性。

属性名称	属性类型	属性值	是否必选	描述
voice	String	线上可调用的发音人的名称代号，为全小写的voice参数值，如”siyue”。	否	阿里巴巴语音合成特有标签。在合成时，指定发音人，优先级高于接口请求参数 `voice` 指定的发音人。 **注意** “xiaoyue”暂不支持在voice中设置。

属性名称	属性类型	属性值	是否必选	描述
encodeType	String	PCM/WAV/MP3	否	阿里巴巴语音合成特有标签。在合成时，指定音频文件格式，优先级高于接口请求参数 `format` 指定的文件格式。
sampleRate	String	8000/16000/24000/48000	否	阿里巴巴语音合成特有标签。在合成时，指定音频的采样率，优先级高于接口请求参数 `sample_rate` 指定的音频采样率。
rate	String	[-500,500]之间整数。默认值为0。 ○ 大于0表示加快语速。 ○ 小于0表示减慢语速。	否	阿里巴巴语音合成特有标签。在合成时，指定音频的语速，优先级高于接口请求参数 `speech_rate` 指定的语速。
pitch	String	[-500,500]之间整数。默认值为0。 ○ 大于0表示升高音高。 ○ 小于0表示降低音高。	否	阿里巴巴语音合成特有标签。在合成时，指定音频的音高，优先级高于接口请求参数 `pitch_rate` 指定的音高。
volume	String	[0,100]之间整数。默认值为50。 ○ 大于50表示增大音量。 ○ 小于50表示减小音量。	否	阿里巴巴语音合成特有标签。在合成时，指定音频的音量，优先级高于接口请求参数 `volume` 指定的音量。

属性名称	属性类型	属性值	是否必选	描述
effect	String	robot/lolita/lowpass/echo/eq/lpfilter/hpfilter	否	阿里巴巴语音合成特有标签。使用该标签可以使合成的语音产生不同的声音效果。 ○ robot：机器人音效。 ○ lolita：萝莉音效。 ○ lowpass：低通音效。 ○ echo：回声音效。 ○ eq：均衡器。 ○ lpfilter：低通滤波器。 ○ hpfilter：高通滤波器。 **说明** ○ 其中eq、lpfilter、hpfilter是高级效果器，您可以通过 `effectValue` 自定义效果器的效果。 ○ 一个SSML只支持一种音效，不可以写多个effect属性。 ○ 选择使用音效功能会增加系统延时。

属性名称	属性类型	属性值	是否必选	描述
effectValue	String	当effect取值为eq/lpfilter/hpfilter时，使用该参数修改效果器的默认效果。	否	○ eq（均衡器）：系统默认使用8个等级。对应的频率为[“40 Hz”，“100 Hz”，“200 Hz”，“400 Hz”，“800 Hz”，“1600 Hz”，“4000 Hz”，“12000 Hz”]；对应的带宽为[“1.0q”，“1.0q”，“1.0q”，“1.0q”，“1.0q”，“1.0q”，“1.0q”]。在使用过程中，需要输入8个等级对应的增益，其取值范围为[-20 dB, 20 dB]。例如，effectValue=” 1 1 1 1 1 1 1 1”。是一个以空格分割的8个整数组成的字符串。数值为0表示不调整对应频率的增益。 ○ lpfilter：输入低通滤波器的频率值。取值为(0, 目标采样率/2]之间的整数。例如effectValue=” 800”。 ○ hpfilter：输入高通滤波器的频率值。取值为(0, 目标采样率/2]之间的整数。例如effectValue=“1200”。
bgm	String	线上可调用的背景音乐的名称。参见bgm属性说明。	否	阿里巴巴语音合成特有标签。为合成的语音添加指定的背景音乐。
backgroundMusicVolume	String	[0,100]之间的整数。默认值为50。 ○ 大于50表示增大音量。 ○ 小于50表示减小音量。	否	阿里巴巴语音合成特有标签。控制背景音乐的音量。

其中，bgm属性说明如下。

服务内嵌URL	自定义背景音URL
目前阿里巴巴语音合成服务内嵌如下几款背景音乐供您体验： ○ 背景音乐1.wav ○ 背景音乐2.wav ○ 背景音乐3.wav	您可以根据需求，使用自定义的背景音。需要将背景音存放在阿里云的OSS上，并且所在的存储空间至少为公共读权限，请参见创建存储空间。使用HTTP/HTTPS协议生成文件访问链接，请参见上传文件。 音频要求： ○ 采样率16 kHz、单声道WAV格式。 ○ 不超过2 MB。 ○ 合成时长超出背景音时长时，背景音将随合成音频循环播放（如果背景音不是WAV格式，可使用ffmpeg将其转为WAV格式： `ffmpeg -i 输入音频 -acodec pcm_s16le -ac 1 -ar 16000 目标.wav` ）。 ○ 标签内的URL如果包含XML的特殊字符，需要做字符转义。 ○ 位深度要求16位。 **注意** 您需要对上传的音频版权承担相应的法律责任。

- 标签关系

  <speak>标签可以包含文本和以下标签：

  - <break>
  - <s>
  - <w>
  - <phoneme>
  - <say-as>

- 示例

  - 空属性

```
<speak>
  需要调用SSML标签的文本
</speak>
```

  音频效果：SSML-speak1.mp3

- voice属性

```
<speak voice="xiaogang">
  我是男声。
</speak>
```

音频效果：SSML-speak2.mp3

- encodeType属性

```
<speak encodeType="mp3">
  我可以生成压缩格式的音频。
</speak>
```

音频效果：SSML-encode.mp3

- sampleRate属性

```
<speak sampleRate="8000">
  看看我的文件大小吧，是16000采样率音频的一半。
</speak>
```

音频效果：SSML-speak4.mp3

- rate属性

```
<speak rate="200">
  我的语速比正常人快。
</speak>
```

音频效果：SSML-speak5.mp3

- pitch属性

```
<speak pitch="-100">
  我的音高却比别人低。
</speak>
```

音频效果：SSML-speak6.mp3

- volume属性

```
<speak volume="80">
  我的音量也很大。
</speak>
```

音频效果：SSML-speak7.mp3

- 属性组合（空格分隔）

```
<speak rate="200" pitch="-100" volume="80">
  所以放在一起，我的声音是这样的。
</speak>
```

音频效果：SSML-speak8.mp3

- effect属性

```
<speak effect="robot">
  你喜欢机器人瓦力吗?
</speak>
```

音频效果：SSML-speak9.mp3

- bgm属性

```
<speak bgm="http://nls.alicdn.com/bgm/2.wav" backgroundMusicVolume="30" rate="-500" vol
ume="40">
  <break time="2s"/>
  阴崖老木苍苍烟
  <break time="700ms"/>
  雨声犹在竹林间
  <break time="700ms"/>
  绵蕝固知裨国计
  <break time="700ms"/>
  绵州风物总堪怜
  <break time="2s"/>
</speak>
```

音频效果：SSML-speak10.mp3

<emotion>

- 描述

  <emotion>用于多情感声音合成，该标签是可选标签。

- 语法

```
<emotion category="happy" intensity="1.0">今天天气真不错！</emotion>
```

- 属性

  <emotion>标签支持如下属性。

属性名称	属性类型	属性值	是否必须	描述
category	String	枚举值：如neutral, happy。	是	指定说话情绪。每个发音人支持的情绪参见下表所示。
intensity	String	浮点值：数值范围[0.01,2.0]。	否	指定情绪强度。默认值为1.0，表示预定义的情绪强度。最小值为0.01，导致目标情绪略有倾向。最大值为2.0，导致目标情绪强度加倍。

其中，多情感声音支持说明如下。

音色名	voice参数值	情感分类（emotion category）
知米_多情感	zhimi_emo	angry, fear, happy, hate, neutral, sad, surprise
知燕_多情感	zhiyan_emo	neutral，happy，angry，sad，fear，hate，surprise，arousal
知贝_多情感	zhibei_emo	neutral，happy，angry，sad，fear，hate，surprise
知甜_多情感	zhitian_emo	neutral，happy，angry，sad，fear，hate，surprise

- 标签关系

  <emotion>标签可以包含文本和以下标签：

  - <s>
  - <sub>
  - <say-as>
  - <w>
  - <phoneme>
  - <prosody>
  - <soundEvent />
  - <break/>
  - <vhml/>

- 示例

  - 空属性

```
<speak voice="zhitian_emo">
    <emotion category="happy" intensity="1.0">今天天气真不错！</emotion>
</speak>
```

    音频效果：SSML-emotion.wav

<break>

- 描述

  用于在文本中插入停顿，该标签是可选标签。

- 语法

```
# 空属性
<break/>
# 带time属性
<break time="string"/>
```

- 属性

> **说明**
>
> 使用无属性的<break>标签时，停顿时长为“1s”。

属性名称	属性类型	属性值	是否必选	描述
time	String	[number]s/[number]ms	否	以秒/毫秒为单位设置停顿的时长（如“2s“、“50ms”）。 ○ [number]s：以秒为单位，[number]取值范围为[1, 10]的整数。 ○ [number]ms：以毫秒为单位，[number]取值范围为[50, 10000]的整数。

- 标签关系

  <break>是空标签，不能包含任何标签。如果SSML结构中存在<s>标签，请把<break>写在<s>里面，表示对当前段落或句子设置停顿。

- 示例

```
<speak>
  请闭上眼睛休息一下<break time="500ms"/>好了，请睁开眼睛。
</speak>
```

  音频效果：SSML-break.mp3

<s>

- 描述

  用于表示文本的句子结构，该标签是可选标签。

- 语法

```
<s>文本</s>
```

- 属性

  无。

- 标签关系

  <s>标签可以包含文本和以下标签：

  ○ <break>
  ○ <w>
  ○ <phoneme>
  ○ <say-as>

- 示例

```
<speak><s>这是第一句话</s><s>这是第二句话</s></speak>
```

音频效果：SSML-s.mp3

<sub>

- 描述

  使用别名替换标签内文本。

- 语法

```
<sub alias="string"></sub>
```

- 属性

属性名称	属性类型	属性值	是否必选	描述
alias	String	替换后的内容	是	用于替换标签内的文本。

- 标签关系

  <sub>标签仅包括文本。

- 示例

```
<speak><sub alias="网络协议标准">W3C</sub></speak>
```

音频效果：SSML-sub.mp3

<w>

- 描述

  用于表示文本的词语结构，该标签是可选标签。英文文本通常采用空格来进行分词，一般无需使用此标签。

- 语法

```
<w>文本</w>
```

- 属性

  无。

- 标签关系

```
<w>文本</w>
```

- 示例

```
<speak>南京市长<w>江大桥</w>今天发表了演讲。</speak>
```

音频效果：SSML-w.mp3

<phoneme>

- 描述

  用于控制标签内文本的读音，该标签是可选标签。英文文本不支持该标签。

- 语法

```
<phoneme alphabet="string" ph="string">文本</phoneme>
```

- 属性

属性名称	属性类型	属性值	是否必选	描述
alphabet	String	py	是	“py”表示拼音。
ph	String	标签内文本对应的拼音串	是	拼音用法的赋值规范： ○ 字与字的拼音用空格分隔，拼音的数目必须与字数相等。 ○ 每个拼音由发音和音调组成，音调为1~5的数字编号，其中”5”表示轻声。

- 标签关系

  <phoneme>标签仅包括文本。

- 示例

```
<speak>
   去<phoneme alphabet="py" ph="dian3 dang4 hang2">典当行</phoneme>把这个玩意<phoneme alphabet="py" ph="dang4 diao4">当掉</phoneme>
</speak>
```

  音频效果：SSML-phoneme.mp3

<soundEvent>

- 描述

  提示音标签，可以在SSML合成过程中，通过该标签在任意位置插入提示音。

- 语法

```
<soundEvent src="URL"/>
```

- 属性

属性名称	属性类型	属性值	是否必选	描述
src	String	URL提示音资源路径	是	您可以根据需求，使用自定义提示音。需要将提示音存放在阿里云OSS上，并且所在的存储空间至少为公共读权限，请参见创建存储空间，使用HTTP/HTTPS协议生成文件访问链接请参见上传文件。 音频要求： ○ 采样率16 kHz、单声道WAV格式。 ○ 不超过2 MB。 ○ 位深度要求16位。 注意 您需要对上传的音频版权承担相应的法律责任。

- 标签关系

  <soundEvent>是空标签，不可以包含任何标签。

- 示例

```
<speak>
  一匹马受了惊吓<soundEvent src="http://nls.alicdn.com/sound-event/horse-neigh.wav"/>人们四散躲避
</speak>
```

音频效果：SSML-sound-event.mp3

<say-as>

- 描述

  用于指示出标签内文本的信息类型，进而按照该类型的默认发音方式发音。

- 语法

```
<say-as interpret-as="string">文本</say-as>
```

- 属性

属性名称	属性类型	属性值	是否必选	描述

属性名称	属性类型	属性值	是否必选	描述
interpret-as	String	cardinal/digits/telephone/name/address/id/characters/punctuation/date/time/currency/measure	是	指示出标签内文本的信息类型： ○ cardinal：按整数或小数发音 ○ digits：按数字发音。 ○ telephone：按电话号码常用方式发音。 ○ name：按人名发音。 ○ address：按地址发音。 ○ id：适用于账户名、昵称等 ○ characters：将标签内的文本按字符一一读出。 ○ punctuation：将标签内的文本按标点符号的方式读出来。 ○ date：按日期发音。 ○ time：按时间发音。 ○ currency：按金额发音。 ○ measure：按计量单位发音。

- 各<say-as>类型支持范围
  - cardinal

格式	示例	中文输出	说明
数字串	145	一百四十五	整数输入范围：20位以内的正负整数，[-99999999999999999999,99999999999999999999]。 小数输入范围：对小数点后小数的位数没有特殊限制，建议不超过10位。
负号+数字串	-145	负一百四十五	
以逗号分隔3位数字串	10,000	一万	
负号+以逗号分隔3位数字串	-10,124	负一万一百二十四	
数字串+小数点+2个零	10.00	十	
负号+数字串+小数点+2个零	-110.00	负一百一十	

格式	示例	中文输出	说明
数字串+小数点+数字串	79.090	七十九点零九零	
负号+数字串+小数点+数字串	-79.001	负七十九点零零一	

格式	示例	英文输出	说明
数字串	145	one hundred forty five	整数输入范围：13位以内的正负整数，[-999999999999,999999999999999]。 小数输入范围：对小数点后小数的位数没有特殊限制，建议不超过10位。
以零开头的数字串	0145	one hundred forty five	
负号+数字串	-145	minus hundred forty five	
以逗号分隔三位数字串	60,000	sixty thousand	
负号+以逗号分隔三位数字串	-208,000	minus two hundred eight thousand	
数字串+小数点+零	12.00	twelve	
数字串+小数点+数字串	12.34	twelve point three four	
以逗号分隔三位数字串+小数点+数字串	1,000.1	one thousand point one	
负号+数字串+小数点+数字串	-12.34	minus twelve point three four	
负号+以逗号分隔三位数字串+小数点+数字串	-1,000.1	minus one thousand point one	
（以逗号分隔三位）数字串+连词符+（以逗号分隔三位）数字	1-1,000	one to one thousand	

格式	示例	英文输出	说明
其它默认读法	012.34	twelve point three four	无
	1/2	one half	
	-3/4	minus three quarters	
	5.1/6	five point one over six	
	-3 1/2	minus three and a half	
	1,000.3^3	one thousand point three to the power of three	
	3e9.1	three times ten to the power of nine point one	
	23.10%	twenty three point one percent	

- digits

格式	示例	中文输出	说明
数字串	129090909	一二九零九零九零九	对数字串的长度没有特殊限制，建议不超过20位。 当数字串超过10位时，每个数字后插入停顿。

格式	示例	英文输出	说明
数字串	12034	one two zero three four	对数字串的长度没有特殊限制，建议不超过20位。 当数字串以空格或连词符分组时，分组之间会插入逗号而产生适当停顿，支持最长5个分组。
数字串+空格或连词符+数字串+空格或连词符+数字串+空格或连词符+数字串	1-23-456 7890	one, two three, four five six, seven eight nine zero	

- telephone

格式	示例	中文输出	说明
座机号	4930286	四九三 零二八六	支持7~8位座机号，支持空格和“-”作为分隔符。 其中，7位座机号支持“3-4”的数字分隔方式；8位座机号支持“4-4”的数字分隔方式。
	493 0286	四九三 零二八六	
	493-0286	四九三 零二八六	
	62552560	六二五五 二五六零	
	6255 2560	六二五五 二五六零	
	6255-2560	六二五五 二五六零	
	4930286-109	四九三 零二八六 转幺零九	

格式	示例	中文输出	说明
座机号+分机号	4930286转109	四九三 零二八六 转幺零九	支持1~4位分机号。
	4930286分机109	四九三 零二八六 分机幺零九	
	4930286分机号109	四九三 零二八六 分机号幺零九	
区号+座机号	01062552560	零幺零 六二五五 二五六零	支持区号：010、02x、03xx、04xx、05xx、07xx、08xx、09xx。
	010 62552560	零幺零 六二五五 二五六零	
	010 6255 2560	零幺零 六二五五 二五六零	
	010 6255-2560	零幺零 六二五五 二五六零	
	010-62552560	零幺零 六二五五 二五六零	
	010-6255-2560	零幺零 六二五五 二五六零	
	(010)62552560	零幺零 六二五五 二五六零	
	03198907098	零三幺九 八九零 七零九八	
	0319-8907098	三幺九 八九零 七零九八	
	010 62552560-109	零幺零 六二五五 二五六零 转幺零九	

格式	示例	中文输出	说明
区号+座机号+分机号	010-62552560-109	零幺零 六二五五 二五六零 转幺零九	无
	(010)62552560-109	零幺零 六二五五 二五六零 转幺零九	
	(010)62552560转109	零幺零 六二五五 二五六零 转幺零九	
	(010)62552560分机109	零幺零 六二五五 二五六零 分机幺零九	
	(010)62552560分机号109	零幺零 六二五五 二五六零 分机号幺零九	
国家代码+区号+座机号	86-010-62791627	八六 零幺零 六二七九 幺六二七	支持国家代码：86、(86)、+86、(+86)、0086。并统一读为“八六”。
	(86)10-62791627	八六 幺零 六二七九 幺六二七	
	+86-010-62791627	八六 零幺零 六二七九 幺六二七	
	0086-10-62791627	八六 幺零 六二七九 幺六二七	
	(+86)-10-6279 1627	八六 幺零 六二七九 幺六二七	
国家代码+区号+座机号+分机号	(86)21-58118818-207	八六 二幺 五八幺幺 八八幺八 转二零七	无
	(86)021-5811-8818-207	八六 零二幺 五八幺幺 八八幺八 转二零七	
	(86)021-58118818转207	八六 零二幺 五八幺幺 八八幺八 转二零七	

格式	示例	中文输出	说明
	(86)21-5811-8818分机207	八六 二幺 五八幺幺 八八幺八 分机二零七	
	+86-021-58118818分机号207	八六 零二幺 五八幺幺 八八幺八分机号二零七	
手机号	139 0000 5678	幺三九 零零零零 五六七八	支持11位手机号，支持3-3-5、3-4-4两种数字分隔方式
	139-000-05678	幺三九 零零零 零五六七八	
	139 000 05678	幺三九 零零零 零五六七八	
国家代码+手机号	+86-13900005678	八六 幺三九 零零零零 五六七八	无
	(+86)-139-0000-5678	八六 幺三九 零零零零 五六七八	
	+8613900005678	八六 幺三九 零零零零 五六七八	
	0086-139 000 05678	八六 幺三九 零零零 零五六七八	
服务号	123	幺二三	■ 支持常用的服务号。 ■ 支持以400/800开头的10位服务号，支持以“3-3-4”的数字分隔方式。 ■ 支持以12530/17951/12593开头的16位号码。
	95678	九五六七八	
	4008110510	四零零 八幺幺 零五幺零	
	800-810-8888	八零零 八幺零 八八八八	
	1253013520638377	幺二五三零 幺三五 二零六三 八三七七	

格式	示例	中文输出	说明
其他	(86)(21)9899-80800-0909	八六 二幺 九八九九 八零八零零 零九零九	支持“数字串+分隔符（左右括号、-）”方式。

格式	示例	英文输出	说明
数字串	12034	one two oh three four	对数字串的长度没有特殊限制，建议不超过20位。当数字串以空格或连词符分组时，分组之间会插入逗号而产生适当停顿，支持最长5个分组。
数字串+空格或连词符+数字串+空格或连词符+数字串	1-23-456 7890	one, two three, four five six, seven eight nine oh	
加号+数字串+空格或连词符+数字串	+43-211-0567	plus four three, two one one, oh five six seven	
左括号+数字串+右括号+空格+数字串+空格或连词符+数字串	(21) 654-3210	(two one) six five four, three two one oh	

- address

格式	示例	中文输出	说明
常用地址格式	元和镇嘉元30-9	元和镇嘉元三十杠九	支持常用地址格式。此处地址指标准的邮寄地址。
	市台路388弄1107-1108号	市台路三八八弄幺幺零七杠幺幺零八号	
	华润二十四城六期锦云府3-1-3205	华润二十四城六期锦云府三杠一杠三二零五	
	圣华名都大厦2幢2006室	圣华名都大厦二幢二零零六室	
	五常街道庭院5幢4单元201	五常街道庭院五幢四单元二零幺	
	芙蓉江路150弄19号	芙蓉江路幺五零弄十九号	

英文文本不支持该标签。

- id

格式	示例	输出	说明
字符串	dell0101	D E L L 零 一 零 一	大小写英文字符、阿拉伯数字0~9、下划线。 输出的空格表示每个字符之间插入停顿，即字符一个一个地读。
	myid_1998	M Y I D 下划线 一 九 九 八	
	AiTest	A I T E S T	

英文文本该标签功能同标签characters。

- characters

格式	示例	中文输出	说明

格式	示例	中文输出	说明
字符串	ISBN 1-001-099098-1	I S B N 一 杠 零 零 一 杠 零 九 九 零 九 八 杠 一	支持中文汉字、大小写英文字符、阿拉伯数字0~9以及部分全角和半角字符。 输出的空格表示每个字符之间插入停顿，即字符一个一个地读。标签内的文本如果包含XML的特殊字符，需要做字符转义。
	x10b2345_u	x 一 零 b 二 三 四 五 下划线 u	
	v1.0.1	v 一 点 零 点 一	
	版本号2.0	版本号二 点 零	
	苏M MA000	苏M M A 零 零 零	
	空中客车A330	空中客车A 三 三 零	
	型号s01 s02和s03	型号s 零 一 s 零二 和s 零 三	
	空中客车A330	空中客车A 三 三 零	
	αβγ	阿尔法 贝塔 伽玛	

格式	示例	英文输出	说明
字符串	*b+3$.c-0'=α	asterisk B plus three dollar dot C dash zero apostrophe equals alpha	支持中文汉字、大小写英文字符、阿拉伯数字0~9以及部分全角和半角字符。 输出的空格表示每个字符之间插入停顿，即字符一个一个地读。 标签内的文本如果包含XML的特殊字符，需要做字符转义。

- punctuation

格式	示例	中文输出	说明
标点符号	…	省略号	支持常见中英文标点。输出的空格表示每个字符之间插入停顿，即字符一个一个地读。 标签内的文本如果包含XML的特殊字符，需要做字符转义。
	……	省略号	
	!"#$%&	叹号 双引号 井号 dollar 百分号 and	
	'()*+	单引号 左括号 右括号 星号 加号	
	,-./:;	逗号 杠 点 斜杠 冒号 分号	
	<=>?@	小于 等号 大于 问号 at	
	[\]^_	左方括号 反斜线 右方括号 脱字符 下划线	

英文文本该标签功能同标签characters。

- date

格式	示例	中文输出	说明
xx年	71年	七一年	支持2位和4位年份。其中： ■ 2位年份支持60年~99年、00年~09年、10年~19年。 ■ 4位年份支持1000年~1999年、2000年~2099年。
	04年	零四年	
	19年	一九年	
	1011年	一零一一年	
	1998年	一九九八年	

格式	示例	中文输出	说明
	2008年	二零零八年	
xx年xx月	98年4月	九八年四月	当月份为1到9月时，支持开头带“0”和不带“0”两种写法。例如“1908年4月”和“1908年04月”。
	1998年04月	一九九八年四月	
	08年8月	零八年八月	
	2008年8月	二零零八年八月	
xx年xx月xx日xx年xx月xx号	98年4月23日	九八年四月二十三日	当日期为1到9日时，支持开头带“0”和不带“0”两种写法。例如“1908年4月8日”和“1908年04月08日”。
	1998年04月23日	一九九八年四月二十三日	
	08年8月8号	零八年八月八号	
	2008年08月08号	二零零八年八月八号	
xx年xx月xx日xx年xx月xx号	98年4月23日	九八年四月二十三日	当日期为1到9日时，支持开头带“0”和不“0”两种写法。例如“1908年4月8日”和“1908年04月08日”。
	1998年04月23日	一九九八年四月二十三日	
	08年8月8号	零八年八月八号	
	2008年08月08号	二零零八年八月八号	
xx月xx号	3月20日	三月二十日	无
	08月07号	八月七号	

格式	示例	中文输出	说明
年月缩写	2018/08	二零一八年八月	支持“/”、“-”、“.”作为缩写的分隔符。
	2018-08	二零一八年八月	
	2018.08	二零一八年八月	
年月日缩写	2018/08/08	二零一八年八月八日	
	2018-8-8	二零一八年八月八日	
	2018.08.08	二零一八年八月八日	
xx年xx月xx日~xx年xx月xx日xx年xx月xx号~xx年xx月xx号	04年9月1日~30日	零四年九月一日至三十日	支持“~”、“-”作为“至”的缩写标志。
	2004年09月01号-2008年06月08号	二零零四年九月一号至二零零八年六月八号	
xx年xx月xx日~xx日xx年xx月xx号~xx号	04年9月1日~30日	零四年九月一日至三十日	
	2004年09月01号-2008年06月08号	二零零四年九月一号至二零零八年六月八号	
xx年xx月~xx年xx月	01年04月~10年04月	零一年四月至一零年四月	
	2001年04月~2010年04月	二零零一年四月至二零一零年四月	
	10月1日~10月7日	十月一日至十月七日	

格式	示例	中文输出	说明
xx月xx日~xx月xx日 xx月xx号~xx月xx号	10月01号~10月07号	十月一号至十月七号	
xx月xx日~xx日xx月xx号~xx号	10月1日~7日	十月一日至七日	
	10月01号~07号	十月一号至七号	
年月日缩写~年月日缩写	2018/03/03~2019/01/01	二零一八年三月三日至二零一九年一月一日	支持“/”、“.”作为缩写的分隔符，支持“~”、“-”作为“至”的缩写标志。
	1997.9.9~1998.9.9	一九九七年九月九日至一九九八年九月九日	
月日缩写~月日缩写	10/20~10/31	十月二十日至十月三十一日	
xx~xx月xx月~xx月	1~10月	一至十月	
	1月~10月	一月至十月	
月日年缩写	10/20/2018	二零一八年十月二十日	仅支持4位的年份，仅支持“/”作为日期的分隔符，仅支持“月/日/年”的书写方式。

格式	示例	英文输出	说明
	2000/01	two thousand, oh one	
	1900-01	nineteen hundred, oh one	
	2001-02	twenty oh one, oh two	

格式	示例	英文输出	说明
四位数字/两位数字或四位数字-两位数字	2019-20	twenty nineteen, twenty	跨年度。
	1998-99	nineteen ninety eight, ninety nine	
	1999-00	nineteen ninety nine, oh oh	
以1或2开头的四位数字	2000	two thousand	四位数字年份。
	1900	nineteen hundred	
	1905	nineteen oh five	
	2021	twenty twenty one	
星期几-星期几 或 星期几~星期几 或 星期几&星期几	mon-wed	monday to wednesday	星期几的范围标签内的文本如果包含XML的特殊字符，需要做字符转义。
	tue~fri	tuesday to friday	
	sat&sun	saturday and sunday	
DD-DD MMM, YYYY 或 DD~DD MMM, YYYY 或 DD&DD MMM, YYYY	19-20 Jan, 2000	the nineteen to the twentieth of january two thousand	DD表示两位数字日期，MMM表示月份的三字母缩写或完整单词，YYYY表示以1或2开头的四位数字年份。
	01 ~ 10 Jul, 2020	the first to the tenth of july twenty twenty	
	05&06 Apr, 2009	the fifth and the sixth of april two thousand nine	

格式	示例	英文输出	说明
MMM DD-DD 或 MMM DD~DD 或 MMM DD&DD	Feb 01 - 03	feburary the first to the third	MMM表示月份的三字母缩写或完整单词，DD表示两位数字日期。
	Aug 10~20	august the tenth to the twentieth	
	Dec 11&12	december the eleventh and the twelfth	
MMM-MMM 或 MMM~MMM 或 MMM&MMM	Jan-Jun	january to june	MMM表示月份的三字母缩写或完整单词。
	jul ~ dec	july to december	
	sep&oct	september and october	
YYYY-YYYY 或 YYYY~YYYY	1990 - 2000	nineteen ninety to two thousand	YYYY表示以1或2开头的四位数字年份。
	2001~2021	two thousand one to twenty twenty one	
WWW DD MMM YYYY	Sun 20 Nov 2011	sunday the twentieth of november twenty eleven	
WWW DD MMM	Sun 20 Nov	sunday the twentieth of november	
WWW MMM DD YYYY	Sun Nov 20 2011	sunday november the twentieth twenty eleven	

格式	示例	英文输出	说明
WWW MMM DD	Sun Nov 20	sunday november the twentieth	WWW表示星期几的三字母缩写或完整单词，DD表示两位数字日期，MMM表示月份的三字母缩写或完整单词，MM表示两位数字月份（或三字母缩写或完整单词），YYYY表示以1或2开头的四位数字年份。
WWW YYYY-MM-DD	Sat 2010-10-01	aturday october the first twenty ten	
WWW YYYY/MM/DD	Sat 2010/10/01	saturday october the first twenty ten	
WWW MM/DD/YYYY	Sun 11/20/2011	sunday november the twentieth twenty eleven	
MM/DD/YYYY	11/20/2011	november the twentieth twenty eleven	
YYYY	1998	nineteen ninety eight	
	10 Mar, 2001	the tenth of march two thousand one	
	10 Mar	the tenth of march	
	Mar 2001	march two thousand one	
	Fri. 10/Mar/2001	friday the tenth of march two thousand one	
	Mar 10th, 2001	march the tenth two thousand one	

格式	示例	英文输出	说明
其它默认读法	Mar 10	march the tenth	无
	2001/03/10	march the tenth two thousand one	
	2001-03-10	march the tenth two thousand one	
	2000s	two thousands	
	2010's	twenty tens	
	1900's	nineteen hundreds	
	1990s	nineteen nineties	

- time

格式	示例	中文输出	说明
时刻	12:00	十二点	
	12:00:00点	十二点	
	10:20分	十点二十分	
	10:20:30	十点二十分三十秒	
	09:18:14	九点十八分十四秒	
	11:00~12:00	十一点到十二点	
	09:00-14:00	九点到十四点	

格式	示例	中文输出	说明
时刻~时刻	11:00~11:30	十一点到十一点三十分	支持常用时间和时间范围格式。
	11:00-12:18	十一点到十二点十八分	
	10:30~11:00	十点三十分到十一点	
	09:28-10:00	九点二十八分到十点	
	10:20~11:20	十点二十分到十一点二十分	
	06:00~08:00	六点到八点	
	上午10:20~下午13:30	上午十点二十分到下午十三点三十分	
	5:00 am	凌晨五点整	
	5:30 am	凌晨五点半	
	5:20:12 am	凌晨五点二十分十二秒	
	7:00 am	上午七点整	
	7:30 AM	上午七点半	
	7:20:12 a.m.	上午七点二十分十二秒	
	07:08:12 A.M.	上午七点零八分十二秒	
	5:00 pm	下午五点整	
	5:30 PM	下午五点半	

时间缩写 格式	示例	中文输出	说明
	5:20:12 p.m.	下午五点二十分十二秒	
	05:09:12 P.M.	下午五点零九分十二秒	
	9:00 pm	晚上九点整	
	9:30 pm	晚上九点半	
	9:20:12 PM	晚上九点二十分十二秒	
	9:02:12 P.M.	晚上九点零二分十二秒	
	12:00 pm	中午十二点整	
	12:30 p.m.	中午十二点半	
	12:20:12 PM	中午十二点二十分十二秒	

格式	示例	英文输出	说明
HH:MM AM或PM	09:00 AM	nine A M	HH表示一或两位数字小时，MM表示两位数字分钟，AM/PM表示上/下午。
	09:03 PM	nine oh three P M	
	09:13 p.m.	nine thirteen p m	
HH:MM	21:00	twenty one hundred	
HHMM	100	one oclock	
	8:00 am - 05:30 pm	eight a m to five p m	

格式 时刻-时刻	示例	英文输出	说明 支持常见时间格式和范围。
	7:05~10:15 AM	seven oh five to ten fifteen A M	
	09:00-13:00	nine oclock to thirteen hundred	

- currency

格式	示例	中文输出	说明
数字+金额标识符	12.00 RMB	十二人民币	支持AUD（澳元）、CAD（加元）、HKD（港币）、JPY（日元）、USD（美元）、CHF（瑞士法郎）、NOK（挪威克朗）、SEK（瑞典克朗）、GBP（英镑）、RMB（人民币）、CNY（元）和EUR（欧元）。 支持的数字格式包括：整数、小数以及以逗号分隔的国际写法。
	12.50 RMB	十二点五零人民币	
	12,000,000 RMB	一千二百万人民币	
	12,000,000.00 RMB	一千二百万人民币	
	12,000.35 RMB	一万两千点三五人民币	
金额标识符+数字	$12	十二美元	支持 CAD（加元）、$（美元）、Fr（法郎）、kr（丹麦克朗）、£（英镑）、¥（元）和 €（欧元）。 支持的数字格式包括：整数、小数以及以逗号分隔的国际写法。
	$12.00	十二美元	
	$12.12	二点一二美元	
	$12,000	一万两千美元	
	$12,000.00	一万两千美元	
	$12,000.99	一万两千点九九美元	

格式	示例	中文输出	说明
其他默认读法	1213	一千二百一十三	无
	1213 KML	一千二百一十三K M L	
	1213.00 KML	一千二百一十三K M L	
	1213.9 KML	一千二百一十三点九K M L	
	1,000 KML	一千K M L	
	1,000.00 KML	一千K M L	
	1,000.98 KML	一千点九八K M L	
	12,000	一万两千	

格式	示例	英文输出	说明

格式	示例	英文输出	说明
数字+金额识别符	1.00 RMB	one yuan	支持的数字格式：整数、小数以及以逗号分隔的国际写法。 支持的金额识别符： CN¥ (yuan) CNY (yuan) RMB (yuan) AUD (australian dollar) CAD (canadian dollar) CHF (swiss franc) DKK (danish krone) EUR (euro) GBP (british pound) HKD (hong kong dollar) JPY (japanese yen) NOK (norwegian krone) SEK (swedish krona) SGD (singapore dollar) USD (united states dollar)
	2.02 CNY	two point zero two yuan	
	1,000.23 CN¥	one thousand point two three yuan	
	1.01 SGD	one singapore dollar and one cent	
	2.01 CAD	two canadian dollars and one cent	
	3.1 HKD	three hong kong dollars and ten cents	
	1,000.00 EUR	one thousand euros	

格式	示例	英文输出	说明
金额识别符+数字	US$ 1.00	one US dollar	支持的数字格式：整数、小数以及以逗号分隔的国际写法。 支持的金额识别符： US$ (US dollar) CA$ (Canadian dollar) AU$ (Australian dollar) SG$ (Singapore dollar) HK$ (Hong Kong dollar) C$ (Canadian dollar) A$ (Australian dollar) $ (dollar) £ (pound) € (euro) CN¥ (yuan) CNY (yuan) RMB (yuan) AUD (australian dollar) CAD (canadian dollar) CHF (swiss franc) DKK (danish krone) EUR (euro) GBP (british pound) HKD (hong kong dollar) JPY (japanese yen) NOK (norwegian krone) SEK (swedish krona) SGD (singapore dollar) USD (united states dollar)
	$0.01	one cent	
	JPY 1.01	one japanese yen and one sen	
	£1.1	one pound and ten pence	
	€2.01	two euros and one cent	
	USD 1,000	one thousand united states dollars	

格式	示例	英文输出	说明
数字+量词+金额识别符 或 金额识别符+数字+量词	1.23 Tn RMB	one point two three trillion yuan	支持的量词格式包括： thousand million billion trillion Mil (million) mil (million) Bil (billion) bil (billion) MM (million) Bn (billion) bn (billion) Tn (trillion) tn (trillion) K(thousand) k (thousand) M (million) m (million)
	$1.2 K	one point two thousand dollars	

- measure

格式	示例	中文输出	说明
数字+中文单位	2片	两片	
	120公顷	一百二十公顷	
	100多毫克	一百多毫克	
	100来米	一百来米	
	100余人	一百余人	
	1厘米20毫米	一厘米二十毫米	

格式	示例	中文输出	说明
	120.00平方公里	一百二十平方公里	支持常见中文单位及单位缩写。
数字+单位缩写	120.56 cm^2	一百二十点五六平方厘米	
	120 m^2 56 cm^2	一百二十平方米五十六平方厘米	
	100 m 12 cm 6 mm	一百米十二厘米六毫米	
范围	10~15 kg	十至十五千克	
	10.24~789.82亩	十点二四至七百八十九点八二亩	
	10米~15米	十米至十五米	
	10.24 cm~19.08 cm	十点二四厘米至十九点零八厘米	
数字+单位+"/"+单位	10元/斤	十元每斤	
	199~299元/件	一百九十九至二百九十九元每件	
	299.99元/g~399.99元/g	二百九十九点九九元每克至三百九十九点九九元每克	
其他默认读法	12扎	十二扎	
	30 rm	三十r m	
	4万万同胞	四万万同胞	
	12.897微克	十二点八九七微克	

格式	示例	英文输出	说明
数字+计量单位	1.0 kg	one kilogram	支持的数字格式：整数、小数以及以逗号分隔的国际写法。 支持常见单位缩写。
	1,234.01 km	one thousand two hundred thirty four point zero one kilometres.	
计量单位	mm^2	square millimetre	

- <say-as>常见符号读法如下表所示。

符号	中文读法	英文读法
!	叹号	exclamation mark
“	双引号	double quote
#	井号	pound
$	dollar	dollar
%	百分号	percent
&	and	and
‘	单引号	left quote
(	左括号	left parenthesis
)	右括号	right parenthesis
*	星	asterisk
+	加	plus

符号	中文读法	英文读法
,	逗号	comma
-	杠	dash
.	点	dot
/	斜杠	slash
:	零冒号	solon
;	分号	semicolon
<	小于	less than
=	等号	equals
>	大于	greater than
?	问号	question mark
@	at	at
[	左方括号	left bracket
\	反斜线	back slash
]	右方括号	right bracket
^	脱字符	caret
_	下划线	underscore
`	反引号	back quote

符号	中文读法	英文读法
{	左花括号	left brace
|	竖线	vertical bar
}	右花括号	right brace
~	波浪线	tilde
!	叹号	exclamation mark
“	左双引号	left double quote
”	右双引号	right double qute
‘	左单引号	left quote
’	右单引号	right quote
(	左括号	left parenthesis
)	右括号	right parenthesis
,	逗号	comma
。	句号	full stop
—	杠	em dash
:	冒号	colon
;	分号	semicolon
?	问号	question mark

符号	中文读法	英文读法
、	顿号	enumeration comma
…	省略号	ellipsis
……	省略号	ellipsis
《	左书名号	left guillemet
》	右书名号	right guillemet
￥	人民币符号	yuan
≥	大于等于	greater than or equal to
≤	小于等于	less than or equal to
≠	不等于	not equal
≈	约等于	approximately equal
±	加减	plus or minus
×	乘	times
π	派	pi
Α	阿尔法	alpha
Β	贝塔	beta
Γ	伽玛	gamma
Δ	德尔塔	delta

符号	中文读法	英文读法
E	艾普西龙	epsilon
Z	捷塔	zeta
E	依塔	epsilon
Θ	西塔	theta
I	艾欧塔	iota
K	喀帕	kappa
Λ	拉姆达	lambda
M	缪	mu
N	拗	nu
Ξ	克西	ksi
O	欧麦克轮	omicron
Π	派	pi
P	柔	rho
Σ	西格玛	sigma
T	套	tau
Y	宇普西龙	upsilon
Φ	fai	phi

符号	中文读法	英文读法
Χ	器	chi
Ψ	普赛	psi
Ω	欧米伽	omega
α	阿尔法	alpha
β	贝塔	beta
γ	伽玛	gamma
δ	德尔塔	delta
ε	艾普西龙	epsilon
ζ	捷塔	zeta
η	依塔	eta
θ	西塔	theta
ι	艾欧塔	iota
κ	喀帕	kappa
λ	拉姆达	lambda
μ	缪	mu
ν	拗	nu
ξ	克西	ksi

符号	中文读法	英文读法
ο	欧麦克轮	omicron
π	派	pi
ρ	柔	rho
σ	西格玛	sigma
τ	套	tau
υ	宇普西龙	upsilon
ϕ	fai	phi
χ	器	chi
ψ	普赛	psi
ω	欧米伽	omega

- <say-as>常见计量单位如下表所示。

格式	类别	中文示例	英文示例
缩写	长度	nm（纳米）、μm（微米）、mm（毫米）、cm（厘米）、m（米）、km（千米）、ft（英尺）、in（英寸）	nm (nanometre), μm (micrometre), mm (millimetre), cm (centimetre), m (metre), km (kilometre), ft (foot), in (inch)
	面积	cm^2（平方厘米）、m^2（平方米）、km^2（平方千米）、SqFt（平方英尺）	cm^2 (square centimetre), m^2 (square metre), km2 (square kilometre), SqFt (square foot)
	体积	cm^3（立方厘米）、m^3（立方米）、km^3（立方千米）、mL（毫升）、L（升）、gallon（加仑）	cm^3 (cubic centimetre), m^3 (cubic metre), km^3 (cubic kilometre), mL (millilitre), L (millilitre), gal (gallon)
	重量	μg（微克）、mg（毫克）、g（克）、kg（千克）	μg (microgram), mg (microgram), g (gram), kg (kilogram)
	时间	min（分）、sec（秒）、ms（毫秒）	min (minute), sec (second), ms (millisecond)
	电磁	μA（微安）、mA（毫安）、Ω（欧姆）、Hz（赫兹）、kHz（千赫兹）、MHz（兆赫兹）、GHz（吉赫兹）、V（伏）、kV（千伏）、kWh（千瓦时）	μA (microamp), mA (milliamp), Hz (hertz), kHz (kilohertz), MHz (megahertz), GHz (gigahertz), V (volt), kV (kilovolt), kWh (kilowatt hour)
	声音	dB（分贝）	dB (decibel)
	气压	Pa（帕）、kPa（千帕）、Mpa（兆帕）	Pa (pascal), kPa (kilopascal), MPa (megapascal)
其它常见单位		支持不限于上述类别的中文单位，例如“米”、“秒”、“美元”、“毫升每瓶”等。以及中文量词，例如“架”、“场”、“头”、“部”、“盆”等。	支持不限于上述类别的计量单位，例如 tsp (teaspoon), rpm (round per minute), KB (kilobyte), mmHg (milimetre of mercury) 等。

- 标签关系

  <say-as>标签仅包括文本。

- 示例
  - cardinal

```
<speak>
  <say-as interpret-as="cardinal">12345</say-as>
</speak>
```

中文音频效果：SSML-say-as_Cardinal.mp3

```
<speak>
  <say-as interpret-as="cardinal">10234</say-as>
</speak>
```

英文音频效果：en-SSML-say-as_cardinal.mp3

  - digits

```
<speak>
  <say-as interpret-as="digits">12345</say-as>
</speak>
```

中文音频效果：SSML-say-as_digit.mp3

```
<speak>
  <say-as interpret-as="digits">10234</say-as>
</speak>
```

英文音频效果：en-SSML-say-as_digits.mp3

  - telephone

```
<speak>
  <say-as interpret-as="telephone">12345</say-as>
</speak>
```

中文音频效果：SSML-say-as_Telephone.mp3

```
<speak>
  <say-as interpret-as="telephone">10234</say-as>
</speak>
```

英文音频效果：en-SSML-say-as_telephone.mp3

  - name

```
<speak>
  她的曾用名是<say-as interpret-as="name">曾小凡</say-as>
</speak>
```

音频效果：SSML-say-as_Name.mp3

- address

```
<speak>
  <say-as interpret-as="address">富路国际1号楼3单元304</say-as>
</speak>
```

音频效果：SSML-say-as_Address.mp3

- id

```
<speak>
  <say-as interpret-as="id">myid_1998</say-as>
</speak>
```

音频效果：SSML-say-as_id.mp3

- characters

```
<speak>
  <say-as interpret-as="characters">希腊字母αβ</say-as>
</speak>
```

中文音频效果：SSML-say-as_characters.mp3

```
<speak>
  <say-as interpret-as="characters">*b+3.c$=α</say-as>
</speak>
```

英文音频效果：en-SSML-say-as_characters.mp3

- punctuation

```
<speak>
  <say-as interpret-as="punctuation"> -./:;</say-as>
</speak>
```

音频效果：SSML-say-as_punctuation.mp3

- date

```
<speak>
  <say-as interpret-as="date">1000-10-10</say-as>
</speak>
```

中文音频效果：SSML-say-as_date.mp3

```
<speak>
  <say-as interpret-as="date">10-01-2020</say-as>
</speak>
```

英文音频效果：en-SSML-say-as_date.mp3

- time

```
<speak>
  <say-as interpret-as="time">5:00am</say-as>
</speak>
```

中文音频效果：SSML-say-as_time.mp3

```
<speak>
  <say-as interpret-as="time">0500</say-as>
</speak>
```

英文音频效果：en-SSML-say-as_time.mp3

- currency

```
<speak>
  <say-as interpret-as="currency">13,000,000.00RMB</say-as>
</speak>
```

中文音频效果：SSML-say-as_currency.mp3

```
<speak>
  <say-as interpret-as="currency">$1,000.01</say-as>
</speak>
```

英文音频效果：en-SSML-say-as_currency.mp3

- measure

```
<speak>
  <say-as interpret-as="measure">100m12cm6mm</say-as>
</speak>
```

中文音频效果：SSML-say-as_measure.mp3

```
<speak>
  <say-as interpret-as="measure">1,000.01kg</say-as>
</speak>
```

英文音频效果：en-SSML-say-as_measure.mp3

综合示例

本示例将详细展示SSML的使用方法，您可以直接拷贝到管控台的项目功能配置中，测试效果。音频效果：综合示例.mp3。

> **说明**
>
> - 示例样音中旁白发音人使用的是定制声音，暂没有开放。体验定制声音，请参见语音合成声音定制。
> - 语音合成每次合成请求的文本字数不能超过300字符，且只能使用一次<speak>标签，以下示例需要按照<speak>标签分多次进行合成测试。

```
<speak>
  相传北宋年间
```

```
  相传北宋年间，
  <say-as interpret-as="date">1121-10-10</say-as>，
  <say-as interpret-as="address">开封城</say-as>
  郊外的早晨笼罩在一片
  <sub alias="双十一">11.11</sub>
  前买买买的欢乐海洋中。一支运货的骡队刚进入城门
  <soundEvent src="http://nls.alicdn.com/sound-event/bell.wav"/>
  一个肤白貌美
  <phoneme alphabet="py" ph="de5">地</phoneme>
  姑娘便拦下第一排的小哥<say-as interpret-as="name">阿发。</say-as>
</speak>

<speak voice="xiaomei">
  “亲，本店今日特惠，鞋履全场
  <say-as interpret-as="digits">199</say-as>
  减
  <say-as interpret-as="cardinal">100</say-as>，
  走过路过不要错过”。
</speak>

<speak voice="sicheng" rate="150">
  “不啦不啦，赶着上货，已经
  <say-as interpret-as="time">09:59:59</say-as>
  了，再晚就供应链断裂了”。
</speak>

<speak>
  <say-as interpret-as="name">阿发</say-as>
  擦了擦汗，带着运货队伍，径直穿过闹巷，耳边充斥着各种叫卖声：
</speak>

<speak voice="ninger" rate="200">
  最新花色现染布匹，买两尺送一尺；
</speak>

<speak voice="xiaobei">
  爆款纱帽头盔，7天无理由退货；
</speak>

<speak voice="sijia">
  专治大小方脉，调理男人妇人疑难杂症。
</speak>

<speak>
  突然，一匹马不知怎么受了惊，在路上嘶鸣狂奔
  <soundEvent src="http://nls.alicdn.com/sound-event/horse-neigh.wav"/>
  一个孩子也吓坏了，跌跌撞撞地扑向大人怀里
  <break time="50ms"/>大喊道：
</speak>

<speak voice="sitong" rate="150">
  “妈妈，妈妈！”
</speak>

<speak>
```

```
<speak>
  这时，
  <say-as interpret-as="name">阿发</say-as>
  心想
</speak>

<speak effect="robot" pitch="-100">
  “吓死宝宝了！”
</speak>

<speak>
  于是他赶紧捂住了
  <phoneme alphabet="py" ph="he2 bao1">钱包</phoneme>，
  继续赶路送货。一路上，
  <say-as interpret-as="address">开封城</say-as>
  的繁荣景象给
  <say-as interpret-as="name">阿发</say-as>
  留下了深刻的印象。
</speak>

<speak bgm="http://nls.alicdn.com/bgm/2.wav" backgroundMusicVolume="30" rate="-200">
  物换星移，繁华落尽，于是他在购物狂欢之余握起画笔，勾勒出一幅长卷，并命名为《清明上河图》。
</speak>
```

12.语音合成时间戳功能介绍

语音实时合成服务在输出音频流的同时，可输出每个汉字/英文单词在音频中的时间位置，即时间戳，时间戳功能又叫字级别音素边界接口。该时间信息可用于驱动虚拟人口型、做视频配音字幕等。

> **注意**
>
> - 只有支持字级别音素边界接口的发音人才有此功能。
> - 由于HTTP协议传输方式的限制，短文本语音合成RESTful API不支持返回时间戳信息。
> - TTS服务返回的字幕是基于发音的，所以不能直接用于上屏，需要使用您的原始文本。
> - 如果用于上屏，可以基于返回的结果，定位每个句子的句首和句尾时间戳。

参数设置

在客户端设置请求参数 `enable_subtitle` 为 `true` ，开启时间戳功能。

以Java SDK为例，其设置方式如下。

```
// 是否开启字幕功能（返回对应文本的相应时间戳），默认不开启。
synthesizer.addCustomedParam("enable_subtitle", true);
```

服务端响应

服务端返回的带字幕信息的响应MetaInfo事件。

参数	类型	说明
subtitles	List	时间戳信息。

其中，SubtitleItem格式如下。

参数	类型	说明
text	String	文本信息。
begin_time	Integer	文本对应tts语音开始时间戳，单位ms。
end_time	Integer	文本对应tts语音结束时间戳，单位ms。

参数	类型	说明
phoneme	无	不支持打印该字对应的phone系列，默认输出为 `"null"` 。
begin_index	Integer	该字在整句中的开始位置，从0开始。
end_index	Integer	该字在整句中的结束位置，从0开始。

返回示例

```
{
    "header": {
        "message_id": "05450bf69c53413f8d88aed1ee60****",
        "task_id": "640bc797bb684bd6960185651307****",
        "namespace": "SpeechSynthesizer",
        "name": "MetaInfo",
        "status": 20000000,
        "status_message": "GATEWAY|SUCCESS|Success."
    },
    "payload": {
        "subtitles": [
            {
                "begin_index": 0,
                "phoneme": "null",
                "end_time": 180,
                "end_index": 1,
                "begin_time": 0,
                "text": "你"
            },
            {
                "begin_index": 1,
                "phoneme": "null",
                "end_time": 299,
                "end_index": 2,
                "begin_time": 180,
                "text": "好"
            }
        ]
    }
}
```

字级别时间戳代码示例

说明

- 示例中使用SDK内置的默认语音合成服务的外网访问服务URL，如果您使用位于阿里云上海地域的ECS，且需要通过内网访问服务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);
```

- 示例中将合成的音频保存在文件中，如果您需要播放音频且对实时性要求较高，建议使用流式播放，即边接收语音数据边播放，减少延时。

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.util.List;
import java.util.Map;

import com.alibaba.fastjson.JSONArray;
import com.alibaba.nls.client.AccessToken;
import com.alibaba.nls.client.protocol.NlsClient;
import com.alibaba.nls.client.protocol.OutputFormatEnum;
import com.alibaba.nls.client.protocol.SampleRateEnum;
import com.alibaba.nls.client.protocol.tts.SpeechSynthesizer;
import com.alibaba.nls.client.protocol.tts.SpeechSynthesizerListener;
import com.alibaba.nls.client.protocol.tts.SpeechSynthesizerResponse;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
/**
 * 此示例演示了：
 *       语音合成API调用。
 *       动态获取token。
 *       流式合成TTS。
 *       首包延迟计算。
 *       字级别时间戳返回。
 */
public class SpeechSynthesizerDemo {
    private static final Logger logger = LoggerFactory.getLogger(SpeechSynthesizerDemo.clas
s);
    private static long startTime;
    private String appKey;
    NlsClient client;
    public SpeechSynthesizerDemo(String appKey, String accessKeyId, String accessKeySecret)
{
        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，使用时注意在accessToken.getExpireTime()过期前再次获取。
        AccessToken accessToken = new AccessToken(accessKeyId, accessKeySecret);
        try {
            accessToken.apply();
            System.out.println("get token: " + accessToken.getToken() + ", expire time: " +
accessToken.getExpireTime());
            client = new NlsClient(accessToken.getToken());
```

```
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public SpeechSynthesizerDemo(String appKey, String accessKeyId, String accessKeySecret,
String url) {
        this.appKey = appKey;
        AccessToken accessToken = new AccessToken(accessKeyId, accessKeySecret);
        try {
            accessToken.apply();
            System.out.println("get token: " + accessToken.getToken() + ", expire time: " +
accessToken.getExpireTime());
            if(url.isEmpty()) {
                client = new NlsClient(accessToken.getToken());
            }else {
                client = new NlsClient(url, accessToken.getToken());
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    private static SpeechSynthesizerListener getSynthesizerListener() {
        SpeechSynthesizerListener listener = null;
        try {
            listener = new SpeechSynthesizerListener() {
                File f=new File("tts_test.wav");
                FileOutputStream fout = new FileOutputStream(f);
                private boolean firstRecvBinary = true;
                //语音合成结束
                @Override
                public void onComplete(SpeechSynthesizerResponse response) {
                    //调用onComplete时表示所有TTS数据已接收完成，因此为整个合成数据的延迟。该延迟可
能较大，不一定满足实时场景。
                    System.out.println("name: " + response.getName() +
                        ", task_id: " + response.getTaskId() +
                        ", status: " + response.getStatus()+
                        ", output file :"+f.getAbsolutePath()
                    );
                }
                //语音合成的语音二进制数据
                @Override
                public void onMessage(ByteBuffer message) {
                    try {
                        if(firstRecvBinary) {
                            //计算首包语音流的延迟，收到第一包语音流时，即可以进行语音播放，以提升响
应速度（特别是实时交互场景下）。
                            firstRecvBinary = false;
                            long now = System.currentTimeMillis();
                            logger.info("tts first latency : " + (now - SpeechSynthesizerDe
mo.startTime) + " ms");
                        }
                        byte[] bytesArray = new byte[message.remaining()];
                        message.get(bytesArray, 0, bytesArray.length);
                        fout.write(bytesArray);
```

```
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
                @Override
                public void onMetaInfo(SpeechSynthesizerResponse response) {
                    // 调用onMetaInfo时表示返回字级别时间戳
                    System.out.println("name: " + response.getName() + ", task_id: " + resp
onse.getTaskId());
                    JSONArray subtitles = (JSONArray)response.getObject("subtitles");
                    List<Map> subtitleList = subtitles.toJavaList(Map.class);
                    for (Map word : subtitleList) {
                        System.out.println("current subtitle: " + word);
                    }
                }
                @Override
                public void onFail(SpeechSynthesizerResponse response){
                    //task_id是调用方和服务端通信的唯一标识，当遇到问题时需要提供task_id以便排查。
                    System.out.println(
                        "task_id: " + response.getTaskId() +
                            //状态码 20000000 表示识别成功
                            ", status: " + response.getStatus() +
                            //错误信息
                            ", status_text: " + response.getStatusText());
                }
            };
        } catch (Exception e) {
            e.printStackTrace();
        }
        return listener;
    }
    public void process() {
        SpeechSynthesizer synthesizer = null;
        try {
            //创建实例，建立连接。
            synthesizer = new SpeechSynthesizer(client, getSynthesizerListener());
            synthesizer.setAppKey(appKey);
            //设置返回音频的编码格式
            synthesizer.setFormat(OutputFormatEnum.WAV);
            //设置返回音频的采样率
            synthesizer.setSampleRate(SampleRateEnum.SAMPLE_RATE_16K);
            //发音人
            synthesizer.setVoice("siyue");
            //语调，范围是-500~500，可选，默认是0。
            synthesizer.setPitchRate(100);
            //语速，范围是-500~500，默认是0。
            synthesizer.setSpeechRate(100);
            //设置用于语音合成的文本
            synthesizer.setText("欢迎使用阿里巴巴智能语音合成服务，您可以说北京明天天气怎么样啊");
            // 是否开启字幕功能（返回相应文本的时间戳），默认不开启，需要注意并非所有发音人都支持该参
数。
            synthesizer.addCustomedParam("enable_subtitle", true);
            //此方法将以上参数设置序列化为JSON格式发送给服务端，并等待服务端确认。
            long start = System.currentTimeMillis();
```

```
            synthesizer.start();
            logger.info("tts start latency " + (System.currentTimeMillis() - start) + " ms"
);
            SpeechSynthesizerDemo.startTime = System.currentTimeMillis();
            //等待语音合成结束
            synthesizer.waitForComplete();
            logger.info("tts stop latency " + (System.currentTimeMillis() - start) + " ms")
;
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            //关闭连接
            if (null != synthesizer) {
                synthesizer.close();
            }
        }
    }
    public void shutdown() {
        client.shutdown();
    }
    public static void main(String[] args) throws Exception {
        String appKey = "您的Appkey";
        String id = "您的AccessKey Id";
        String secret = "您的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);
        }
        SpeechSynthesizerDemo demo = new SpeechSynthesizerDemo(appKey, id, secret, url);
        demo.process();
        demo.shutdown();
    }
}
```

13.界面化的TTS下载工具

本文为您介绍在熟悉代码的情况下，如何在控制台通过界面化工具合成并下载音频。

前提条件

- 已开通智能语音交互服务，请参见开通服务。
- 已创建项目，请参见创建项目。

使用限制

目前此功能只针对标准版TTS声音。

操作步骤

> **注意**
>
> 该合成能力调用的是语音合成服务，100字符以内（含100字符）每试听或下载一次，均算作一次服务调用。例如，合成一句话之后，试听发现有个别读音不自然，更换同音字或更改了标点符号，再次试听满意后下载，则计费次数为3次（2次试听和1次下载）。具体计费信息请参见计费说明。

1. 登录智能语音交互管理控制台。
2. 在左侧导航栏单击**全部项目**。
3. 找到需要操作的项目，单击右侧**操作**栏中的**项目功能配置**。

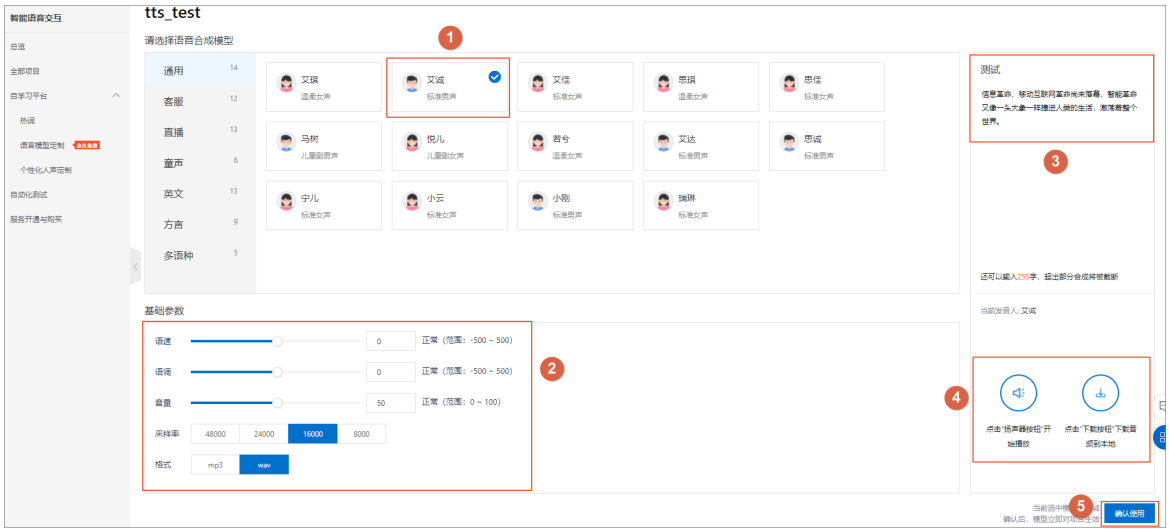
- 如果您已配置语音合成服务，单击右侧**测试**文本框的**语音合成**，直接输入文字，试听并下载。

- 如果您还没有配置语音合成服务，单击**语音合成**模块内的**去配置**。

a. 选择语音合成模型。

b. 设置**基础参数**。

c. 在**测试**文本框内输入不超过300字符的文字。

d. 单击**播放**试听合成效果，单击**下载**可将音频下载到本地。

e. 单击右下角**确认使用**，保存配置信息。