ALIBABA CLOUD

阿里云

智能语音交互
开发前必读

文档版本：20210115

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.使用前须知

初次使用智能语音服务，请先阅读快速开始系列文档，快速体验使用过程，而后按顺序阅读如下文档，以便更好地使用智能语音交互服务。

文档资源	描述
基本概念	介绍与智能语音交互服务相关基本概念。
管理项目	在管理控制台上创建项目、配置参数等。
获取Token	获取访问令牌后再调用智能语音交互服务。
调用语音交互服务	• 一句话识别 • 实时语音识别 • 语音合成 • 录音文件识别 • 录音文件识别极速版 • 长文本语音合成
语音识别自学习工具	优化语音识别效果。

各服务对比

服务	时效性	功能	适用场景	支持的语音格式	支持调用方式	免费调用量	购买
一句话识别	实时	识别一分钟内的短语音	APP语音搜索、语音电话客服、对话聊天、控制口令等场景	PCM（无压缩的PCM或WAV文件）、OPUS	Java/C++/Android/iOS	最大2个并发	可单独购买
实时语音识别	实时	识别长时间的语音数据流	会议演讲、视频直播等长时间不间断语音场景	PCM（无压缩的PCM或WAV文件）	Java/C++/Android/iOS	最大2个并发	可单独购买

服务	时效性	功能	适用场景	支持的语音格式	支持调用方式	免费调用量	购买
语音合成	实时	合成长度不超过300个字符（UTF-8编码）的文本内容	需要人工合成音的场景	PCM、WAV、MP3	Java/C++/Android/iOS	最大2个并发	可单独购买
录音文件识别	非实时	录音文件上传后（文件大小不超过512 MB），针对免费用户，可在24小时内完成识别并返回识别文本；针对付费用户，可在6小时内完成识别并返回识别文本。 **说明** 一次性上传大规模数据（半小时内上传超过500小时时长的录音）的除外，如果您有大规模数据转写需求，可与售前专家联系。	非实时识别场景	支持单轨/双轨的WAV、MP3	Java/C++/GO/.NET/Node.js/PHP/Python	每个自然日最多识别2小时时长的录音文件。	可单独购买
录音文件识别极速版	实时	识别文件大小不超过100 MB，30分钟以内时长的音频，转写完成时间不超过10秒。	短视频编辑工具、 电台和报社字幕内容	AAC、MP3、OPUS、WAV	HTTP POST/Android/iOS	暂不支持免费试用	可单独购买
长文本语音合成	非实时	将超长文本（千字或万字）合成为语音二进制数据。	阅读小说、文章等场景	PCM、WAV、MP3	JAVA/C++/RESTful API	暂不支持免费试用	可单独购买

注意

- 除录音文件识别和录音文件识别极速版以外的其他识别服务只支持单声道（mono）语音数据。
- 识别服务只支持8000Hz/16000Hz采样率、16bit采样位数的音频。

2.基本概念

本文为您介绍智能语音交互服务中的相关概念，以便于更好地理解本产品。

采样率（sample rate）

音频采样率是指录音设备在一秒钟内对声音信号的采样次数，采样频率越高声音的还原就越真实越自然。

目前语音识别服务支持16000Hz和8000Hz两种采样率，其中电话业务一般使用8000Hz，其余业务使用16000Hz。

调用语音识别服务时，如果语音数据采样率高于16000Hz，需要先把采样率转换为16000Hz才能发送给语音识别服务；如果语音数据采样率是8000Hz，请勿将采样率转换为16000Hz，项目中选用支持8000Hz采样率的模型。

采样位数（sample size）

采样值或取样值，即是将采样样本幅度量化。用来衡量声音波动变化的参数，或是声卡的分辨率。数值越大、分辨率越高，发出声音的能力越强。

目前语音识别中常用的采样位数为16bits小端序。即每次采样的音频信息用2字节保存，或者说2字节记录1/16000s的音频数据。

每个采样数据记录的是振幅，采样精度取决于采样位数的大小：

- 1字节（8bit）记录256个数，亦即将振幅划分为256个等级。
- 2字节（16bit）记录65536个数。

其中2字节采样位数已经能够达到CD标准。

语音编码（format）

语音数据存储和传输的方式。注意语音编码和语音文件格式不同，如常见的WAV文件格式，会在其头部定义语音数据的编码，其中的音频数据通常使用PCM、AMR或其他编码。

> **注意**
>
> 在调用智能语音交互服务之前需确认语音数据编码格式是服务所支持的。

声道（sound channel）

录制声音时，在不同空间位置采集的相互独立的音频信号。声道数也就是声音录制时的音源数量。常见的音频数据为单声道或双声道（立体声）。

> **说明**
>
> 除录音文件识别以外的服务**只支持单声道(mono)**语音数据，如果您的数据是双声道或其他，需要先转换为单声道。

逆文本规整（inverse text normalization）

语音转换为文本时使用标准化的格式展示数字、金额、日期和地址等对象，以符合阅读习惯。以下是一些示例。

语音原始文本	开启ITN的识别结果
百分之二十	20%
一千六百八十元	1680元
五月十一号	5月11号
请拨幺幺零	请拨110

项目标识（appkey）

在智能语音交互管理控制台中创建的每个项目都有一个唯一标识，即appkey。当您调用智能语音服务时必须提供appkey，服务通过appkey获得项目的具体配置信息。

当存在多个业务需要智能语音服务，如电话客服场景和手机输入法场景，各场景需要的语音能力是不同的，只有当项目配置与业务场景匹配才能获得最佳效果。

访问标识（access key）

程序访问阿里云API的凭证，登录管理页面，创建并查看访问标识。

访问标识由ID和Secret两部分构成：AccessKey ID是类似身份的标识，AccessKey Secret的作用是签名您的访问参数，防止数据被篡改。两者必须组合使用。其中AccessKey Secret类似登录密码，不要向任何人泄漏。

访问令牌（access token）

访问智能语音交互服务的凭证，提供有效期控制，您可以通过AccessKey ID和AccessKey Secret获取访问令牌。

> **说明**
>
> 对于手机等设备端调用智能语音服务的场景，可以在服务端获取令牌，发送给设备端使用，能够有效避免Access key泄露。

中间结果（intermediate result）

在调用语音识别服务时可以设置是否返回中间结果：

- 设置为false时，只在语音全部识别完后返回一次完整的结果。
- 设置为true时，除了最后一次完整的结果之外，还会在您说话的同时返回中间结果。

如一段语音，识别最终结果是“你好阿里巴巴”。在启用中间结果后，会在您说话的同时返回5次结果，如下所示。

```
你
你好
你好啊
你好阿里
你好阿里巴巴
```

> 说明
>
> - 中间结果可能在后续返回结果中被修正。
> - 中间结果增量返回的字数不固定，并不是每次都比上一次多识别一个字。

task_id

每一个语音服务请求都会有一个唯一的task_id，由SDK自动生成，用于定位问题。

3.获取Token

访问令牌（Access Token）是调用智能语音交互服务的凭证。本文介绍获取访问令牌的方法。

前提条件

已获取AccessKey ID和AccessKey Secret，详情请参见开通服务。

您可以使用阿里云公共SDK调用云端服务获取**Access Token**（调用时需提供阿里云账号的**AccessKey ID**和**AccessKey Secret**）。除此之外，您可根据需求，自行完成获取**Access Token**的客户端请求，详情请参见获取Token协议说明。

> **说明**
>
> 访问令牌使用前需要通过ExpireTime参数获取有效期时间戳，过期需要重新获取。

操作步骤

1. 登录智能语音交互管控台。
2. 默认进入总览，单击页面右上角**点击获取AccessToken**。

   方便快速对接SDK，测试语音服务，该Token仅供测试使用。

   调用云端服务的返回示例如下：

```
{
  "NlsRequestId": "aed8c1af075347819118ff6bf811****",
  "RequestId": "0989F63E-5069-4AB0-822B-5BD2D953****",
  "Token": {
    "ExpireTime": 1527592757,
    "Id": "124fc7526f434b8c8198d6196b0a****",
    "UserId": "12345678****"
  }
}
```

   其中：

   - Id为本次分配的访问令牌Access Token。
   - ExpireTime为此令牌的有效期时间戳，单位：秒。如，`1527592757` 换算为北京时间为：`2018/5/29 19:19:17`，即Token在该时间之前有效。

异常排查

在调用SDK获取Access Token过程中，出现异常失败需排查问题，请参见错误代码表。

> 说明
>
> 1. 调用方最常见的一个错误描述是：`Request was denied due to user flow control.`，表示接口调用被限流，所以获取Token的请求不宜频繁，否则可能会被阿里云网关拒绝。
> 2. Access Token获取之后可在有效期内多次使用（跨线程、跨进程、跨机器）。

移动端获取Token

推荐在服务端集成下方提供的SDK，获取访问令牌。

1. 在服务端集成Java SDK，通过HTTP GET实现*/gettoken接口。
2. 客户端访问*/gettoken接口获取Token A和有效时间。
3. 使用Token A访问语音服务。每次调用语音服务之前都获取新的Token。
4. 在本地缓存Token，如下次访问在Token A的有效时间内，直接使用Token A；若Token A已经过期，获取新的Token并更新本地缓存。

通过SDK获取Token

调用示例（Java）

从maven服务器下载最新版本SDK，下载demo源码ZIP包。

```
<dependency>
  <groupId>com.alibaba.nls</groupId>
  <artifactId>nls-sdk-common</artifactId>
  <version>2.1.6</version>
</dependency>
```

```
AccessToken token = new AccessToken("your akID", "your akSecret");
token.apply();
String accessToken = token.getToken();
long expireTime = token.getExpireTime();
```

> 说明
>
> 以上Java代码是用在服务端的，Android使用需要在服务端/pc上运行获取Token，出于安全考虑，建议您搭建服务端，客户端请求服务端更新Token。

调用示例（C++）

> 说明
>
> - Linux下安装工具要求如下：
>   - Glibc 2.5及以上
>   - Ccc4或Ccc5
> - Windows下：VS2013或VS2015
>
> Windows平台需要您自己搭建工程。

1. 下载 C++ Token SDK。
2. 编译示例。

   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：

   确认本地系统已安装Cmake2.4及以上版本。

   i. 切换目录：`cd path/to/sdk/lib` 。
   ii. 解压缩文件：`tar -zxvpf linux.tar.gz` 。
   iii. 切换目录：`cd path/to/sdk` 。
   iv. 执行编译脚本：`./build.sh` 。
   v. 切换目录：`cd path/to/sdk/demo` 。
   vi. 执行获取Token示例：`./tokenDemo <yourAccessKeySecret> <yourAccessKeyId>` 。

   不支持Cmake：

   i. 切换目录：`cd path/to/sdk/lib` 。
   ii. 解压缩文件：`tar -zxvpf linux.tar.gz` 。
   iii. 切换目录：`cd path/to/sdk/demo` 。
   iv. 使用g++编译命令编译示例程序：`g++ -o tokenDemo tokenDemo.cpp -I path/to/sdk/include/ -L path/to/sdk/lib/linux -ljsoncpp -lssl -lcrypto -lcurl -luuid -lnlsCommonSdk -D_GLIBCXX_USE_CXX11_ABI=0` 。
   v. 指定库路径：`export LD_LIBRARY_PATH=path/to/sdk/lib/linux/` 。
   vi. 执行获取Token示例：`./tokenDemo <yourAccessKeySecret> <yourAccessKeyId>` 。
3. 调用服务。

   获取访问令牌Token的示例代码如下：

```
#include <iostream>
#include "Token.h"

using std::cout;
using std::endl;
using namespace AlibabaNlsCommon;

//获取访问令牌TokenId
int getTokenId(const char* keySecret, const char* keyId) {
  NlsToken nlsTokenRequest;

  /*设置阿里云账号KeySecret*/
  nlsTokenRequest.setKeySecret(keySecret);
  /*设置阿里云账号KeyId*/
  nlsTokenRequest.setAccessKeyId(keyId);

  /*获取Token. 成功返回0, 失败返回-1*/
  if (-1 == nlsTokenRequest.applyNlsToken()) {
    cout << "Failed: " << nlsTokenRequest.getErrorMsg() << endl; /*获取失败原因*/

    return -1;
  } else {
    cout << "TokenId: " << nlsTokenRequest.getToken() << endl; /*获取TokenId*/
    cout << "TokenId expireTime: " << nlsTokenRequest.getExpireTime() << endl; /*获取Token有效期时间戳(秒)*/

    return 0;
  }
}
```

调用示例（PHP）

> **说明**
>
> 如下使用是阿里云新版PHP SDK，请参见Alibaba Cloud SDK for PHP。如果您已接入阿里云旧版PHP SDK，参见aliyun-openapi-php-sdk，仍然可以继续使用或者更新到新版SDK（推荐）。旧版获取Token的方法请参见基于阿里云aliyun-openapi-php-sdk 获取Token。

安装前请确保环境使用的是PHP 5.5及以上版本，PHP SDK安装方式请参见安装Alibaba Cloud SDK for PHP。

调用步骤：

1. 创建一个全局客户端。

2. 创建API请求，设置参数。

3. 发起请求，处理应答或异常。

```
<?php
require __DIR__ . '/vendor/autoload.php';
use AlibabaCloud\Client\AlibabaCloud;
use AlibabaCloud\Client\Exception\ClientException;
use AlibabaCloud\Client\Exception\ServerException;
/**
 * 第一步：设置一个全局客户端
 * 使用阿里云RAM账号的AccessKey ID和AccessKey Secret进行鉴权。
 */
AlibabaCloud::accessKeyClient(
        "<your-access-key-id>",
        "<your-access-key-secret>")
        ->regionId("cn-shanghai")
        ->asDefaultClient();
try {
  $response = AlibabaCloud::nlsCloudMeta()
                ->v20180518()
                ->createToken()
                ->request();
  print $response . "\n";
  $token = $response["Token"];
  if ($token != NULL) {
    print "Token 获取成功：\n";
    print_r($token);
  }
  else {
    print "token 获取失败\n";
  }
} catch (ClientException $exception) {
  // 获取错误消息
  print_r($exception->getErrorMessage());
} catch (ServerException $exception) {
  // 获取错误消息
  print_r($exception->getErrorMessage());
}
```

通过CommonRequest获取Token

使用阿里云公共SDK获取Access token，建议采用RPC风格的API调用。发起一次RPC风格的CommonAPI请求，需要提供以下参数：

参数名	参数值	说明
domain	nls-meta.cn-shanghai.aliyuncs.com	产品的通用访问域名，固定值。
region_id	cn-shanghai	服务的地域ID，固定值。
action	CreateToken	API的名称，固定值。
version	2019-02-28	API的版本号，固定值。

调用示例（Java）

1. 添加Java依赖。

   添加阿里云Java SDK的核心库（版本为3.7.1）和fastjson库。

```
<dependency>
  <groupId>com.aliyun</groupId>
  <artifactId>aliyun-java-sdk-core</artifactId>
  <version>3.7.1</version>
</dependency>

<!-- http://mvnrepository.com/artifact/com.alibaba/fastjson -->
<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>fastjson</artifactId>
  <version>1.2.49</version>
</dependency>
```

2. 调用服务。

   获取访问令牌的示例代码如下：

```
import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONObject;
import com.aliyuncs.CommonRequest;
import com.aliyuncs.CommonResponse;
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.IAcsClient;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.http.MethodType;
```

```
import com.aliyuncs.http.MethodType;
import com.aliyuncs.http.ProtocolType;
import com.aliyuncs.profile.DefaultProfile;

import java.text.SimpleDateFormat;
import java.util.Date;

public class CreateTokenDemo {

  // 地域ID
  private static final String REGIONID = "cn-shanghai";
  // 获取Token服务域名
  private static final String DOMAIN = "nls-meta.cn-shanghai.aliyuncs.com";
  // API版本
  private static final String API_VERSION = "2019-02-28";
  // API名称
  private static final String REQUEST_ACTION = "CreateToken";

  // 响应参数
  private static final String KEY_TOKEN = "Token";
  private static final String KEY_ID = "Id";
  private static final String KEY_EXPIRETIME = "ExpireTime";


  public static void main(String args[]) throws ClientException {
    if (args.length < 2) {
      System.err.println("CreateTokenDemo need params: <AccessKey Id> <AccessKey Secret>");
      System.exit(-1);
    }

    String accessKeyId = args[0];
    String accessKeySecret = args[1];

    // 创建DefaultAcsClient实例并初始化
    DefaultProfile profile = DefaultProfile.getProfile(
        REGIONID,
        accessKeyId,
        accessKeySecret);

    IAcsClient client = new DefaultAcsClient(profile);
    CommonRequest request = new CommonRequest();
```

```
        request.setDomain(DOMAIN);
        request.setVersion(API_VERSION);
        request.setAction(REQUEST_ACTION);
        request.setMethod(MethodType.POST);
        request.setProtocol(ProtocolType.HTTPS);

        CommonResponse response = client.getCommonResponse(request);
        System.out.println(response.getData());
        if (response.getHttpStatus() == 200) {
            JSONObject result = JSON.parseObject(response.getData());
            String token = result.getJSONObject(KEY_TOKEN).getString(KEY_ID);
            long expireTime = result.getJSONObject(KEY_TOKEN).getLongValue(KEY_EXPIRETIME);
            System.out.println("获取到的Token： " + token + "，有效期时间戳(单位：秒): " + expireTime);
            // 将10位数的时间戳转换为北京时间
            String expireDate = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date(expireTi
me * 1000));
            System.out.println("Token有效期的北京时间：" + expireDate);
        }
        else {
            System.out.println("获取Token失败！");
        }
    }
}
```

调用示例（Python）

1. 使用pip安装SDK。

   执行以下命令，通过pip安装Python SDK，版本为2.13.3。

   ```
   pip install aliyun-python-sdk-core==2.13.3 # 安装阿里云SDK核心库
   ```

2. 调用服务。

   示例代码如下：

```
# -*- coding: utf8 -*-

from aliyunsdkcore.client import AcsClient
from aliyunsdkcore.request import CommonRequest

# 创建AcsClient实例
client = AcsClient(
  "<您的AccessKey Id>",
  "<您的AccessKey Secret>",
  "cn-shanghai"
);

# 创建request，并设置参数。
request = CommonRequest()
request.set_method('POST')
request.set_domain('nls-meta.cn-shanghai.aliyuncs.com')
request.set_version('2019-02-28')
request.set_action_name('CreateToken')
response = client.do_action_with_exception(request)

print(response)
```

调用示例（GO）

示例代码如下：

```
package main
import (
  "fmt"
  "github.com/aliyun/alibaba-cloud-sdk-go/sdk"
  "github.com/aliyun/alibaba-cloud-sdk-go/sdk/requests"
)
func main() {
  client, err := sdk.NewClientWithAccessKey("cn-shanghai", "<yourAccessKey Id>", "<yourAccessKey Secret>
")
  if err != nil {
    panic(err)
  }
  request := requests.NewCommonRequest()
  request.Method = "POST"
  request.Domain = "nls-meta.cn-shanghai.aliyuncs.com"
  request.ApiName = "CreateToken"
  request.Version = "2019-02-28"
  response, err := client.ProcessCommonRequest(request)
  if err != nil {
    panic(err)
  }
  fmt.Print(response.GetHttpStatus())
  fmt.Print(response.GetHttpContentString())
}
```

调用示例（Node.js）

1. 安装Node.js SDK。

   建议使用npm完成Node.js依赖模块的安装，所有阿里官方的Node.js SDK都位于@alicloud下。

   假设Node.js SDK下载后的路径为 `/path/to/aliyun-openapi-Node.js-sdk` 。当基于SDK核心库进行开发时，请执行以下命令安装@alicloud/pop-core模块。

   命令中的 `--save` 会将模块写入应用的package.json文件中，作为依赖模块。

   ```
   $ npm install @alicloud/pop-core --save
   ```

   > **说明**
   >
   > 您也可以从GitHub上下载SDK，请参见GitHu下载SDK。

2. 调用服务。

   示例代码如下：

```
var RPCClient = require('@alicloud/pop-core').RPCClient;

var client = new RPCClient({
 accessKeyId: '<yourAccessKey Id>',
 accessKeySecret: '<yourAccessKey Secret>',
 endpoint: 'http://nls-meta.cn-shanghai.aliyuncs.com',
 apiVersion: '2019-02-28'
});

// => returns Promise
// => request(Action, params, options)
client.request('CreateToken').then((result) => {
  console.log(result.Token);
});
```

调用示例（.NET）

1. 安装.NET SDK。

   获取Token的.NET示例使用了阿里云.NET SDK的CommonRequest用来提交获取Token的请求，采用的是RPC风格的POP API调用。阿里云.NET SDK的详细介绍请参见阿里云.Net SDK使用手册。

   .NET SDK CommonRequest的使用方法，请参见使用CommonReques进行调用。

   您只需安装阿里云.NET SDK的核心库。可以选择添加DLL引用或者项目引入任一种方式安装.NET SDK，详细操作步骤请参见使用.NET SDK。

2. 调用服务。

   示例代码如下：

```
using System;
using Aliyun.Acs.Core;
using Aliyun.Acs.Core.Exceptions;
using Aliyun.Acs.Core.Profile;
class Sample
{
  static void Main(string[] args)
  {
    // 构建一个客户端实例，用于发起请求。
    IClientProfile profile = DefaultProfile.GetProfile(
      "cn-shanghai",
      "<yourAccessKey Id>",
      "<yourAccessKey Secret>");
    DefaultAcsClient client = new DefaultAcsClient(profile);

    try
    {
      // 构造请求
      CommonRequest request = new CommonRequest();
      request.Domain = "nls-meta.cn-shanghai.aliyuncs.com";
      request.Version = "2019-02-28";

      // 因为是RPC风格接口，需指定ApiName(Action)。
      request.Action = "CreateToken";

      // 发起请求，并得到响应。
      CommonResponse response = client.GetCommonResponse(request);
      System.Console.WriteLine(response.Data);
    }
    catch (ServerException ex)
    {
      System.Console.WriteLine(ex.ToString());
    }
    catch (ClientException ex)
    {
      System.Console.WriteLine(ex.ToString());
    }
  }
}
```

4.获取Token协议说明

访问令牌（Access Token）是调用智能语音交互服务的凭证。我们提供了基于阿里云公共SDK调用云端服务获取Access Token的方式。本文为您介绍获取Access Token的接口协议说明。

请求报文

客户端向服务端发送获取Token的请求，服务端返回创建Token结果的响应。客户端发送的请求支持使用HTTP/HTTPS协议，请求方法支持GET/POST方法。服务端提供了基于阿里云POP协议的接口，因此客户端需要实现阿里云POP的签名机制。

由于HTTPS协议的请求参数设置与HTTP协议相同，下面以HTTP协议请求为例，介绍如何发送请求获取Token。

- URL

协议	URL	方法
HTTP/1.1	http://nls-meta.cn-shanghai.aliyuncs.com/	GET/POST

- 请求参数

名称	类型	是否必选	说明
AccessKeyId	String	是	阿里云账号AccessKey ID
Action	String	是	POP API名称：CreateToken
Version	String	是	POP API版本：2019-02-28
Format	String	是	响应返回的类型：JSON
RegionId	String	是	服务所在的地域ID：cn-shanghai

名称	类型	是否必选	说明
Timestamp	String	是	请求的时间戳。日期格式按照ISO 8601标准表示，且使用UTC时间，时区：+0。格式：YYYY-MM-DDThh:mm:ssZ。如2019-04-03T06:15:03Z为UTC时间2019年4月3日6点15分03秒。
SignatureMethod	String	是	签名算法：HMAC-SHA1
SignatureVersion	String	是	签名算法版本：1.0
SignatureNonce	String	是	唯一随机数uuid，用于请求的防重放攻击，每次请求唯一，不能重复使用。格式为A-B-C-D-E（A、B、C、D、E的字符位数分别为8、4、4、4、12）。例如，8d1e6a7a-f44e-40d5-aedb-fe4a1c80f434。
Signature	String	是	由所有请求参数计算出的签名结果，生成方法参见下文签名机制。

说明

- 使用GET方法，需要将请求参数设置在请求行中：`/?请求参数字符串` 。
- 使用POST方法，需要将请求参数设置在请求体中。

- HTTP请求头部

HTTP请求头部由“关键字-值”对组成，每行一对，关键字和值用英文冒号“:”分隔，内容如下：

名称	类型	是否必选	描述
Host	String	否	HTTP请求的服务器域名：`nls-meta.cn-shanghai.aliyuncs.com`，一般根据请求链接自动解析。
Accept	String	否	指定客户端能够接收的内容类型：`application/json`，不设置默认为 */* 。
Content-type	String	POST方法必须设置	指定POST方法请求体数据格式：`application/x-www-form-urlencoded`。

报文示例：

- HTTP GET请求报文

```
GET /?Signature=O0s6pfeOxtFM6YKSZKQdSyPR9Vs%3D&AccessKeyId=LTAF3sAA****&Action=CreateToken&Format=JSON&RegionId=cn-shanghai&SignatureMethod=HMAC-SHA1&SignatureNonce=a1f01895-6ff1-43c1-ba15-6c109fa00106&SignatureVersion=1.0&Timestamp=2019-03-27T09%3A51%3A25Z&Version=2019-02-28 HTTP/1.1
Host: nls-meta.cn-shanghai.aliyuncs.com
User-Agent: curl/7.49.1
Accept: */*
```

- HTTP POST请求报文

```
POST / HTTP/1.1
Host: nls-meta.cn-shanghai.aliyuncs.com
User-Agent: curl/7.49.1
Accept: */*
Content-type: application/x-www-form-urlencoded
Content-Length: 276

SignatureVersion=1.0&Action=CreateToken&Format=JSON&SignatureNonce=8d1e6a7a-f44e-40d5-aedb-fe4a1c80f434&Version=2019-02-28&AccessKeyId=LTAF3sAA****&Signature=oT8A8RgvFE1tMD%2B3hDbGuoMQSi8%3D&SignatureMethod=HMAC-SHA1&RegionId=cn-shanghai&Timestamp=2019-03-25T09%3A07%3A52Z
```

响应结果：

发送获取Token的HTTP请求后，会受到服务端响应，结果以JSON字符串的形式保存在响应中，GET方法和POST方法的响应结果相同。

- 成功响应

  HTTP状态码为200，响应字段如下表所示。

参数	类型	说明
Token	Token对象	包含Token值和有效期时间戳。
Id	String	请求分配的Token值。
ExpireTime	Long	Token的有效期时间戳（单位：秒。例如1553825814换算为北京时间为：2019/3/29 10:16:54，即Token在该时间之前有效。）。

```
HTTP/1.1 200 OK
Date: Mon, 25 Mar 2019 09:29:24 GMT
Content-Type: application/json; charset=UTF-8
Content-Length: 216
Connection: keep-alive
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: X-Requested-With, X-Sequence, _aop_secret, _aop_signature
Access-Control-Max-Age: 172800
Server: Jetty(7.2.2.v20101205)

{"NlsRequestId":"dd05a301b40441c99a2671905325****","RequestId":"E11F2DC2-0163-4D97-A704-0BD280
45****","ErrMsg":"","Token":{"ExpireTime":1553592564,"Id":"88916699****","UserId":"150151111111****
"}}
```

响应体JSON字符串内容如下。

```
{
  "NlsRequestId": "dd05a301b40441c99a2671905325****",
  "RequestId": "E11F2DC2-0163-4D97-A704-0BD28045****",
  "ErrMsg": "",
  "Token": {
    "ExpireTime": 1553592564,
    "Id": "889******166",
    "UserId": "150**********151"
  }
}
```

- 失败响应

  HTTP状态码为非200，响应字段说明如下表。

参数	类型	说明
RequestId	String	请求ID
Message	String	失败响应的错误信息
Code	String	失败响应的错误码

> **说明**
>
> 请根据错误码和错误信息提示检查请求参数是否设置正确，如无法排查，请将响应信息提交工单。

以重传入阿里云账号的AccessKey Id错误为例，响应消息如下。

```
HTTP/1.1 404 Not Found
Date: Thu, 28 Mar 2019 07:23:01 GMT
Content-Type: application/json; charset=UTF-8
Content-Length: 290
Connection: keep-alive
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: X-Requested-With, X-Sequence, _aop_secret, _aop_signature
Access-Control-Max-Age: 172800
Server: Jetty(7.2.2.v20101205)

{"Recommend":"https://error-center.aliyun.com/status/search?Keyword=InvalidAccessKeyId.NotFound&source=PopGw","Message":"Specified access key is not found.","RequestId":"A51587CB-5193-4DB8-9AED-CD4365C2****","HostId":"nls-meta.cn-shanghai.aliyuncs.com","Code":"InvalidAccessKeyId.NotFound"}
```

响应体JSON字符串内容如下。

```
{
  "Recommend": "https://error-center.aliyun.com/status/search?Keyword=InvalidAccessKeyId.NotFound&source=PopGw",
  "Message": "Specified access key is not found.",
  "RequestId": "A51587CB-5193-4DB8-9AED-CD4365C2****",
  "HostId": "nls-meta.cn-shanghai.aliyuncs.com",
  "Code": "InvalidAccessKeyId.NotFound"
}
```

签名机制

服务端POP API对每个接口访问请求的发送者都要进行身份验证，所以无论使用HTTP协议还是HTTPS协议提交的请求，都需要在请求中包含签名信息。通过签名机制，服务端可以确认哪位用户在做API请求，并能确认请求在网络传输过程中是否被篡改。

安全验证流程：

计算签名时，需要阿里云账号的AccessKey Id和AccessKey Secret，使用HMAC-SHA1算法进行对称加密。其工作流程如下。

1. 请求端根据API请求内容（包括HTTP请求参数和请求体）生成签名字符串。
2. 请求端使用阿里云账号的AccessKey Id和AccessKey Secret对第一步生成的签名字符串进行签名，获得该API请求的数字签名。

3. 请求端把API请求内容和数字签名一同发送给服务端。
4. 服务端在接收到请求后会重复1、2步骤（服务端会在后台获取该请求使用的用户密钥）并计算出该请求期望的数字签名。
5. 服务端用期望的数字签名和请求端发送过来的数字签名做对比，若完全一致则认为该请求通过验证。否则拒绝该请求。

生成请求的签名字符串：

1. 构造规范化的请求字符串。

   将HTTP请求参数（不包括Signature）构造成规范化的请求字符串。规范化步骤：

   i. 参数排序。按参数名的字典顺序，对请求参数进行大小写敏感排序。

   ii. 参数编码。对排序后的请求参数进行规范化设置。

   请求参数名和值都要使用 `UTF-8` 字符集进行URL编码，URL编码规则如下：

   - 对于字符 A-Z、a-z、0-9以及字符 `-` 、`_` 、`.` 、`~` 不编码。
   - 对于其他字符编码成“%XY”的格式，其中XY是字符对应ASCII码的16进制表示。比如英文的双引号（”）对应的编码就是%22。
   - 对于扩展的 `UTF-8` 字符，编码成“%XY%ZA...”的格式。
   - 需要说明的是英文空格要被编码为%20，而不是加号 `+` 。

   > **说明**
   >
   > 一般支持URL编码的库（比如Java中的java.net.URLEncoder）都是按照“application/x-www-form-urlencoded”的MIME类型的规则进行编码的。实现时可以直接使用此类方式进行编码，然后把编码后的字符串中：加号 `+` 替换为 `%20` ，星号 `*` 替换为 `%2A` ， `%7E` 替换为波浪号 `~` ，即可得到上述规则描述的编码字符串。

   iii. 使用等号 `=` 连接URL编码后的参数名和参数值： `percentEncode(参数Key) + “=” + percentEncode(参数值)` 。

   iv. 使用与（ `&` ）号连接第c步URL编码后的请求参数对，如 `Action=CreateToken&Format=JSON` 。

   > **说明**
   >
   > 字符串中第一个参数名前面不需要 `&` 符号。

   v. 返回规范化的请求字符串。

   示例如下：

```
String percentEncode(String value) throws UnsupportedEncodingException {
  return value != null ? URLEncoder.encode(value, URL_ENCODING)
      .replace("+", "%20")
      .replace("*", "%2A")
      .replace("%7E", "~") : null;
}

// 对参数Key排序
String[] sortedKeys = queryParamsMap.keySet().toArray(new String[] {});
Arrays.sort(sortedKeys);

// 对排序的参数进行编码、拼接
for (String key : sortedKeys) {
  canonicalizedQueryString.append("&")
      .append(percentEncode(key)).append("=")
      .append(percentEncode(queryParamsMap.get(key)));
}
queryString = canonicalizedQueryString.toString().substring(1);
```

> 说明
>
> 完整代码实现请参见完整示例的 `canonicalizedQuery` 函数。

构造规范化的请求字符串：

```
AccessKeyId=LTA******3s2&Action=CreateToken&Format=JSON&RegionId=cn-shanghai&SignatureMethod=HMAC-SHA1&SignatureNonce=f20b1beb-e5dc-4245-9e96-aa582e905c1a&SignatureVersion=1.0&Timestamp=2019-04-03T03%3A40%3A13Z&Version=2019-02-28
```

2. 构造待签名字符串。

   将HTTP请求的方法（GET）、URL编码的URL路径（/）、第1步获取的规范化请求字符串使用与（ & ）符号连接成待签名字符串： `HTTPMethod + "&" + percentEncode("/") + "&" + percentEncode(queryString)` 。

   构造签名字符串代码示例：

```
StringBuilder strBuilderSign = new StringBuilder();
strBuilderSign.append(HTTPMethod);
strBuilderSign.append("&");
strBuilderSign.append(percentEncode(urlPath));
strBuilderSign.append("&");
strBuilderSign.append(percentEncode(queryString));

stringToSign = strBuilderSign.toString();
```

> 说明
>
> 完整代码实现请参见完整示例的 `createStringToSign` 函数。

构造的签名字符串：

```
GET&%2F&AccessKeyId%3DLTA******F3s%26Action%3DCreateToken%26Format%3DJSON%26RegionId%3Dcn-shanghai%26SignatureMethod%3DHMAC-SHA1%26SignatureNonce%3Da237e025-07ea-4d87-bb04-d9b2712d871d%26SignatureVersion%3D1.0%26Timestamp%3D2019-04-19T03%253A31%253A40Z%26Version%3D2019-02-28
```

3. 计算签名。
   - 签名采用HMAC-SHA1算法 + Base64，编码采用 `UTF-8` 。
   - 根据AccessKey Secret，将第2步构造的待签名字符串使用HMAC-SHA1算法计算出对应的数字签名。其中，计算签名时使用的AccessKey Secret必须在其后面增加一个与字符 `&` 。
   - 签名也要做URL编码。

   计算签名的代码示例：

```
signature = Base64( HMAC-SHA1(stringToSign, accessKeySecret + "&") );
// 进行URL编码
signature = percentEncode(signature)
```

> 说明
>
> 完整代码实现请阅读完整示例的 `sign` 函数。

计算得到的签名：

```
# 签名串
AKIktdPUMCV12fTh667BLXeuCtg=
# URL编码后的结果
AKIktdPUMCV12fTh667BLXeuCtg%3D
```

计算签名后，将签名的键值对用符号 = 连接，并使用符号 & 添加到第1步获取的请求字符串中，作为HTTP GET请求参数，发送到服务端，获取Token。

```
String queryStringWithSign = "Signature=" + signature + "&" + queryString;
```

快速测试

您可以使用以下参数计算签名，比较计算结果是否一致。

> 说明
>
> AccessKey Id和AccessKey Secret没有提供真实数据，Timestamp是过期的时间，使用这些参数计算的签名，获取Token时会失败，仅用于计算签名测试对比。

- AccessKeyId:

  `my_access_key_id`

- AccessKeySecret:

  `my_access_key_secret`

- Timestamp:

  `2019-04-18T08:32:31Z`

- SignatureNonce:

  `b924c8c3-6d03-4c5d-ad36-d984d3116788`

请求参数如下:

```
AccessKeyId:my_access_key_id
Action:CreateToken
Version:2019-02-28
Timestamp:2019-04-18T08:32:31Z
Format:JSON
RegionId:cn-shanghai
SignatureMethod:HMAC-SHA1
SignatureVersion:1.0
SignatureNonce:b924c8c3-6d03-4c5d-ad36-d984d3116788
```

1. 规范化请求字符串。

   ```
   AccessKeyId=my_access_key_id&Action=CreateToken&Format=JSON&RegionId=cn-shanghai&SignatureMethod=HMAC-SHA1&SignatureNonce=b924c8c3-6d03-4c5d-ad36-d984d3116788&SignatureVersion=1.0&Timestamp=2019-04-18T08%3A32%3A31Z&Version=2019-02-28
   ```

2. 构造待签名字符串。

```
GET&%2F&AccessKeyId%3Dmy_access_key_id%26Action%3DCreateToken%26Format%3DJSON%26RegionId%3Dcn-shanghai%26SignatureMethod%3DHMAC-SHA1%26SignatureNonce%3Db924c8c3-6d03-4c5d-ad36-d984d3116788%26SignatureVersion%3D1.0%26Timestamp%3D2019-04-18T08%253A32%253A31Z%26Version%3D2019-02-28
```

3. 计算得到签名。

```
hHq4yNsPitlfDJ2L0nQPdugdEzM=
# URL编码后的结果
hHq4yNsPitlfDJ2L0nQPdugdEzM%3D
```

4. 得到携带签名的请求字符串。

```
Signature=hHq4yNsPitlfDJ2L0nQPdugdEzM%3D&AccessKeyId=my_access_key_id&Action=CreateToken&Format=JSON&RegionId=cn-shanghai&SignatureMethod=HMAC-SHA1&SignatureNonce=b924c8c3-6d03-4c5d-ad36-d984d3116788&SignatureVersion=1.0&Timestamp=2019-04-18T08%3A32%3A31Z&Version=2019-02-28
```

5. 组合成HTTP请求链接。

```
http://nls-meta.cn-shanghai.aliyuncs.com/?Signature=hHq4yNsPitlfDJ2L0nQPdugdEzM%3D&AccessKeyId=my_access_key_id&Action=CreateToken&Format=JSON&RegionId=cn-shanghai&SignatureMethod=HMAC-SHA1&SignatureNonce=b924c8c3-6d03-4c5d-ad36-d984d3116788&SignatureVersion=1.0&Timestamp=2019-04-18T08%3A32%3A31Z&Version=2019-02-28
```

6. 替换第5步获取的HTTP请求链接，使用浏览器或者curl，获取Token。

```
curl "http://nls-meta.cn-shanghai.aliyuncs.com/?Signature=${您的签名}&AccessKeyId=${您的AccessKeyId}&Action=CreateToken&Format=JSON&RegionId=cn-shanghai&SignatureMethod=HMAC-SHA1&SignatureNonce=${您的请求UUID}&SignatureVersion=1.0&Timestamp=${您的请求时间戳}&Version=2019-02-28"
```

完整示例

> **说明**
>
> 本文提供Java、Python语言示例代码，您可根据协议和示例实现其他语言的客户端程序。

- Java示例

  依赖文件：

```
<!-- http://mvnrepository.com/artifact/com.alibaba/fastjson -->
<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>fastjson</artifactId>
  <version>1.2.49</version>
</dependency>

<dependency>
  <groupId>com.squareup.okhttp3</groupId>
  <artifactId>okhttp</artifactId>
  <version>3.9.1</version>
</dependency>
```

示例代码：

```
import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONObject;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.Response;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import javax.xml.bind.DatatypeConverter;
import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.text.SimpleDateFormat;
import java.util.Arrays;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.SimpleTimeZone;
import java.util.UUID;
public class CreateToken {
  private final static String TIME_ZONE = "GMT";
  private final static String FORMAT_ISO8601 = "yyyy-MM-dd'T'HH:mm:ss'Z'";
  private final static String URL_ENCODING = "UTF-8";
  private static final String ALGORITHM_NAME = "HmacSHA1";
  private static final String ENCODING = "UTF-8";
  private static String token = null;
```

```
private static long expireTime = 0;
/**
 * 获取时间戳
 * 必须符合ISO8601规范，并需要使用UTC时间，时区为+0。
 */
public static String getISO8601Time(Date date) {
  Date nowDate = date;
  if (null == date) {
    nowDate = new Date();
  }
  SimpleDateFormat df = new SimpleDateFormat(FORMAT_ISO8601);
  df.setTimeZone(new SimpleTimeZone(0, TIME_ZONE));
  return df.format(nowDate);
}
/**
 * 获取UUID
 */
public static String getUniqueNonce() {
  UUID uuid = UUID.randomUUID();
  return uuid.toString();
}
/**
 * URL编码
 * 使用UTF-8字符集按照RFC3986规则编码请求参数和参数取值。
 */
public static String percentEncode(String value) throws UnsupportedEncodingException {
  return value != null ? URLEncoder.encode(value, URL_ENCODING).replace("+", "%20")
      .replace("*", "%2A").replace("%7E", "~") : null;
}
/***
 * 将参数排序后，进行规范化设置，组合成请求字符串。
 * @param queryParamsMap  所有请求参数
 * @return 规范化的请求字符串
 */
public static String canonicalizedQuery( Map<String, String> queryParamsMap) {
  String[] sortedKeys = queryParamsMap.keySet().toArray(new String[] {});
  Arrays.sort(sortedKeys);
  String queryString = null;
  try {
    StringBuilder canonicalizedQueryString = new StringBuilder();
    for (String key : sortedKeys) {
```

```
            canonicalizedQueryString.append("&")
                .append(percentEncode(key)).append("=")
                .append(percentEncode(queryParamsMap.get(key)));
        }
        queryString = canonicalizedQueryString.toString().substring(1);
        System.out.println("规范化后的请求参数串：" + queryString);
    } catch (UnsupportedEncodingException e) {
        System.out.println("UTF-8 encoding is not supported.");
        e.printStackTrace();
    }
    return queryString;
}
/**
 * 构造签名字符串
 * @param method     HTTP请求的方法
 * @param urlPath    HTTP请求的资源路径
 * @param queryString 规范化的请求字符串
 * @return 签名字符串
 */
public static String createStringToSign(String method, String urlPath, String queryString) {
    String stringToSign = null;
    try {
        StringBuilder strBuilderSign = new StringBuilder();
        strBuilderSign.append(method);
        strBuilderSign.append("&");
        strBuilderSign.append(percentEncode(urlPath));
        strBuilderSign.append("&");
        strBuilderSign.append(percentEncode(queryString));
        stringToSign = strBuilderSign.toString();
        System.out.println("构造的签名字符串：" + stringToSign);
    } catch (UnsupportedEncodingException e) {
        System.out.println("UTF-8 encoding is not supported.");
        e.printStackTrace();
    }
    return stringToSign;
}
/**
 * 计算签名
 * @param stringToSign    签名字符串
 * @param accessKeySecret  阿里云AccessKey Secret加上与号&
 * @return 计算得到的签名
```

```
     @return 计算得到的签名
     */
    public static String sign(String stringToSign, String accessKeySecret) {
        try {
            Mac mac = Mac.getInstance(ALGORITHM_NAME);
            mac.init(new SecretKeySpec(
                accessKeySecret.getBytes(ENCODING),
                ALGORITHM_NAME
            ));
            byte[] signData = mac.doFinal(stringToSign.getBytes(ENCODING));
            String signBase64 = DatatypeConverter.printBase64Binary(signData);
            System.out.println("计算的得到的签名：" + signBase64);
            String signUrlEncode = percentEncode(signBase64);
            System.out.println("UrlEncode编码后的签名：" + signUrlEncode);
            return signUrlEncode;
        } catch (NoSuchAlgorithmException e) {
            throw new IllegalArgumentException(e.toString());
        } catch (UnsupportedEncodingException e) {
            throw new IllegalArgumentException(e.toString());
        } catch (InvalidKeyException e) {
            throw new IllegalArgumentException(e.toString());
        }
    }
    /***
     * 发送HTTP GET请求，获取token和有效期时间戳。
     * @param queryString 请求参数
     */
    public static void processGETRequest(String queryString) {
        /**
         * 设置HTTP GET请求
         * 1. 使用HTTP协议
         * 2. Token服务域名：nls-meta.cn-shanghai.aliyuncs.com
         * 3. 请求路径：/
         * 4. 设置请求参数
         */
        String url = "http://nls-meta.cn-shanghai.aliyuncs.com";
        url = url + "/";
        url = url + "?" + queryString;
        System.out.println("HTTP请求链接：" + url);
        Request request = new Request.Builder()
            .url(url)
```

```
            .header("Accept", "application/json")
            .get()
            .build();
        try {
            OkHttpClient client = new OkHttpClient();
            Response response = client.newCall(request).execute();
            String result = response.body().string();
            if (response.isSuccessful()) {
                JSONObject rootObj = JSON.parseObject(result);
                JSONObject tokenObj = rootObj.getJSONObject("Token");
                if (tokenObj != null) {
                    token = tokenObj.getString("Id");
                    expireTime = tokenObj.getLongValue("ExpireTime");
                }
                else{
                    System.err.println("提交获取Token请求失败: " + result);
                }
            }
            else {
                System.err.println("提交获取Token请求失败: " + result);
            }
            response.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public static void main(String args[]) {
        if (args.length < 2) {
            System.err.println("CreateTokenDemo need params: <AccessKey Id> <AccessKey Secret>");
            System.exit(-1);
        }
        String accessKeyId = args[0];
        String accessKeySecret = args[1];
        System.out.println(getISO8601Time(null));
        // 所有请求参数
        Map<String, String> queryParamsMap = new HashMap<String, String>();
        queryParamsMap.put("AccessKeyId", accessKeyId);
        queryParamsMap.put("Action", "CreateToken");
        queryParamsMap.put("Version", "2019-02-28");
        queryParamsMap.put("Timestamp", getISO8601Time(null));
        queryParamsMap.put("Format", "JSON");
```

```
queryParamsMap.put("RegionId", "cn-shanghai");
queryParamsMap.put("SignatureMethod", "HMAC-SHA1");
queryParamsMap.put("SignatureVersion", "1.0");
queryParamsMap.put("SignatureNonce", getUniqueNonce());
/**
 * 1.构造规范化的请求字符串
 */
String queryString = canonicalizedQuery(queryParamsMap);
if (null == queryString) {
  System.out.println("构造规范化的请求字符串失败！");
  return;
}
/**
 * 2.构造签名字符串
 */
String method = "GET"; // 发送请求的 HTTP 方法，GET
String urlPath = "/"; // 请求路径
String stringToSign = createStringToSign(method, urlPath, queryString);
if (null == stringToSign) {
  System.out.println("构造签名字符串失败");
  return;
}
/**
 * 3.计算签名
 */
String signature = sign(stringToSign, accessKeySecret + "&");
if (null == signature) {
  System.out.println("计算签名失败!");
  return;
}
/**
 * 4.将签名加入到第1步获取的请求字符串
 */
String queryStringWithSign = "Signature=" + signature + "&" + queryString;
System.out.println("带有签名的请求字符串：" + queryStringWithSign);
/**
 * 5.发送HTTP GET请求，获取token。
 */
processGETRequest(queryStringWithSign);
if (token != null) {
  System.out.println("获取的Token：" + token + ", 有效期时间戳（秒）：" + expireTime);
```

```
        System.out.println("获取的Token：" + token + "，有效期时间戳（秒）：" + expireTime);
        // 将10位数的时间戳转换为北京时间
        String expireDate = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date(expireTime
* 1000));
        System.out.println("Token有效期的北京时间：" + expireDate);
    }
  }
}
```

- Python示例

> 说明
>
> - Python版本要求：Python3.4及以上。
> - 安装Python HTTP库Requests：
>
> ```
> pip install requests
> ```

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import base64
import hashlib
import hmac
import requests
import time
import uuid
from urllib import parse
class AccessToken:
  @staticmethod
  def _encode_text(text):
    encoded_text = parse.quote_plus(text)
    return encoded_text.replace('+', '%20').replace('*', '%2A').replace('%7E', '~')
  @staticmethod
  def _encode_dict(dic):
    keys = dic.keys()
    dic_sorted = [(key, dic[key]) for key in sorted(keys)]
    encoded_text = parse.urlencode(dic_sorted)
    return encoded_text.replace('+', '%20').replace('*', '%2A').replace('%7E', '~')
  @staticmethod
  def create_token(access_key_id, access_key_secret):
    parameters = {'AccessKeyId': access_key_id,
            'Action': 'CreateToken',
```

```
            'Format': 'JSON',
            'RegionId': 'cn-shanghai',
            'SignatureMethod': 'HMAC-SHA1',
            'SignatureNonce': str(uuid.uuid1()),
            'SignatureVersion': '1.0',
            'Timestamp': time.strftime("%Y-%m-%dT%H:%M:%SZ", time.gmtime()),
            'Version': '2019-02-28'}
    # 构造规范化的请求字符串
    query_string = AccessToken._encode_dict(parameters)
    print('规范化的请求字符串: %s' % query_string)
    # 构造待签名字符串
    string_to_sign = 'GET' + '&' + AccessToken._encode_text('/') + '&' + AccessToken._encode_text(query_
string)
    print('待签名的字符串: %s' % string_to_sign)
    # 计算签名
    secreted_string = hmac.new(bytes(access_key_secret + '&', encoding='utf-8'),
                bytes(string_to_sign, encoding='utf-8'),
                hashlib.sha1).digest()
    signature = base64.b64encode(secreted_string)
    print('签名: %s' % signature)
    # 进行URL编码
    signature = AccessToken._encode_text(signature)
    print('URL编码后的签名: %s' % signature)
    # 调用服务
    full_url = 'http://nls-meta.cn-shanghai.aliyuncs.com/?Signature=%s&%s' % (signature, query_string)
    print('url: %s' % full_url)
    # 提交HTTP GET请求
    response = requests.get(full_url)
    if response.ok:
      root_obj = response.json()
      key = 'Token'
      if key in root_obj:
        token = root_obj[key]['Id']
        expire_time = root_obj[key]['ExpireTime']
        return token, expire_time
    print(response.text)
    return None, None
if __name__ == "__main__":
  # 用户信息
  access_key_id = '您的AccessKeyId'
  access_key_secret = '您的AccessKeySecret'
```

```
    token, expire_time = AccessToken.create_token(access_key_id, access_key_secret)
    print('token: %s, expire time(s): %s' % (token, expire_time))
    if expire_time:
        print('token有效期的北京时间：%s' % (time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(expire_time))))
```

5.语音识别自学习工具

在语音识别服务中心，如果您的业务领域有部分词汇默认识别效果不好可以使用热词功能； 如果您需要的语音识别服务场景不在所提供的模型范围内，或者需要对标准模型进行更进一步优化，可以使用语音模型定制功能，达成优化目的。 通过自学习工具的有效使用，能够提高场景语音识别率。

在管理控制台训练定制模型的操作，请参见管理自学习模型。

训练语料说明

自学习平台为您提供热词和语言模型定制功能。

- 热词功能：能够对人名、地名或业务专属热词进行针对性识别，详情请参见热词概述。
- 语言模型定制功能：可以对阿里云提供的标准模型进行再一步优化，尤其是专有名词和高频词汇，有较好优化效果，详情请参见概语言模型定制功能概述。

应用举例

某地即将召开一场红学研究会，为记录会场嘉宾的讲话内容，主办方选择接入阿里云智能语音服务进行会议转写。首先开发人员注册并开通阿里云智能语音服务，为提高发言内容的识别率，采用自学习平台进行模型训练优化。

1. 选择基础模型：采用**多语言通用**。
2. 采集训练语料：由于会议核心是讨论红楼梦相关的议题，开发人员将红楼梦的原文进行处理，根据标点做裁剪，将原文每句话保存为训练文本中的一行。
3. 操作训练模型：通过自学习平台提交语料，采用训练出的模型，可以有效识别出贾宝玉等红楼梦中的词汇，从而获得理想的识别效果。

6.智能语音服务1.0升级2.0

本文带您开启智能语音服务全新2.0版本。

步骤一：开通服务

首先开通智能语音2.0服务，请参见开通服务。

语音识别服务免费试用版提供最大不超过2路并发，如需更多并发，请选择升级为商用版，升级前请阅读计费方式，详情请参见产品定价。

步骤二：登录管控台

智能语音管控台为2.0语音服务所特有。

步骤三：创建项目

项目创建完成会生成appkey，此处appkey和智能语音服务1.0中appkey功能相同，都是使用SDK时的关键参数。

智能语音服务1.0中appkey是由官网文档给出的一组固定字符串，供所有用户使用。

智能语音服务2.0中针对不同用户的不同项目会生成不同appkey。

> 说明
>
> 管理1.0版本服务的项目，仍需要使用官网给出的固定appkey。升级2.0版本以后需要在控制台重新配置项目，使用新生成的项目appkey。

步骤四：配置模型

对项目进行功能配置，根据音频文件选择合适的场景。

> 说明
>
> 场景设置与音频文件采样率不匹配会导致无法识别、识别不准确等各种问题。

步骤五：升级SDK

智能语音服务1.0版本与2.0版本的SDK区别如下：

- 相同点：都是基于websocket长连接。基本流程一致：建立连接—发送请求—发送语音—异步接收结果—关闭连接。
- 不同点：鉴权方式不同。2.0版本需要先获取访问令牌Token，请求服务时用Token鉴权，详情请参见获取Token。

1. 下载相关语言SDK示例。

   使用您的AccessKey ID、AccessKey Secret和appkey运行示例代码，详细操作请参见运行SDK。

2. 替换SDK。

   将智能语音1.0版本的SDK替换为2.0版本的SDK，将项目中相应代码进行变更、重构。

   以Java SDK为例，SDK对应关系如下。

服务	智能语音服务1.0	智能语音服务2.0
一句话识别	nls-service-sdk	nls-sdk-recognizer
实时语音识别	nls-realtime-sdk	nls-sdk-transcriber
语音合成	nls-service-sdk	nls-sdk-tts

7.服务使用问题

本文为您解答服务使用的相关问题。

appkey是什么？

您可能有多个业务需要智能语音服务，如客服场景和司法场景，每个场景需要的语音能力是不同的。appkey就是用来区别不同业务场景的标识。只有在appkey项目中设置了合适的场景，才能获得理想的效果。

“流式”模式和“非流式”模式识别的区别？

“非流式”模式也叫“普通”模式，“普通”模式下服务判断用户整句话说完后才返回一次识别结果。
而“流式”模式下用户一边说话一边返回识别结果，在句子结束的识别结果前会有很多中间结果。

语音识别服务支持哪些编码格式的音频？

每种服务支持的格式不尽相同，请参见各服务中的说明。您可以使用常见音频编辑软件如Audacity，查看音频文件的编码格式。

语音识别服务支持哪些采样率？

目前语音识别服务仅支持16KHz和8KHz两种采样率，其他采样率如48KHz建议重采样到16 KHz，再调用语音识别服务。请注意选择和语音文件采样率对应的appkey。

怎么查看音频文件的采样率？

可以使用常见音频编辑软件如Audacity查看音频文件的采样率，也可以使用开源命令行工具FFmpeg查看。

语音识别服务支持离线功能吗？

目前不支持本地离线的语音识别，必须将音频数据发送至服务端做识别。

智能语音交互能的域名是什么？

智能语音交互服务的域名：wss://nls-gateway.cn-shanghai.aliyuncs.com/ws/v1 。

> **说明**
>
> HTTP协议需开放80端口；HTTPS协议需开放443端口。

现在有对识别结果进行敏感词屏蔽吗？

目前未提供此功能。您获取识别结果后可以按需处理。

语音识别服务支持英文识别吗？

支持英文识别。请在管控台的配置项目语音识别模型中选择英文模型。要求语音采样率为16 KHz。

该服务支持但不限于英式、美式及中式等口音的英语。

语音识别服务支持方言识别吗？

语音识别服务支持方言识别。具体方言模型请在管控台中进行设置。更多信息，请参见管理项目。

语音识别能否自动断开多句话？

实时语音识别服务可以断开多句话；一句话识别服务的每个请求只对应一句话，无法断开。

免费用户有什么限制？

- 一句话识别/实时语音识别：最多同时发送2路语音识别。
- 录音文件识别：每自然日最多识别2小时时长的录音文件。

服务请求时长限制？

- 一句话识别支持60s以内的实时语音。
- 实时语音识别不限制时长。

token重新获取会不会导致已获取的token失效？

token重新获取不会影响已获取token的有效性，有效性只与时间有关。token的有效期通过服务端响应消息的ExpireTime参数获取。更多信息，请参见获取Token协议说明。

是否可以提供服务的IP白名单？

由于服务器的IP范围很广，不能提供IP白名单。

实时语音识别，识别慢、超时问题？

排查方式：

1. 运行阿里云提供的示例，和您的服务对比运行状态，记录并提供日志信息。
2. 记录请求对应的taskid，方便排查问题。
3. 客户端使用TCPDump（Linux）/Wireshark（Windows）等抓包工具，确定网络状况。

8.SDK及接口调用问题

本文为您解答SDK及接口调用相关问题。

为什么语音识别准确率很低，有时只识别出几个字？

请检查音频数据的采样率与管控台应用的模型是否一致，以及音频是否是单通道录音。

> 说明
>
> 只有录音文件识别支持双通道的录音。

确认调用方式和采样率都没问题，识别还是不准确怎么办？

您可以通过如下两种方式提高识别准确率：

- 使用自定义热词功能，快速、实时提高准确率，详情请参见热词概述。
- 开通自学习模型训练，通过模型定制的方式提高大量文本的识别率，详情请参见语言模型定制概述。

音频数据必须连续发送吗？

音频数据必须连续发送。

服务端在超过一定时间未接到语音数据（10秒）会超时断开连接，返回40000004错误信息。如果需要再次发送数据，客户端需要重新发起请求。

音频数据发送中断后，为什么还会收到服务器发回的数据？

音频数据未连续发送超时中断后，服务器上如果还有之前未处理完的数据，就会继续返回这些数据的识别结果，但是整个句子的识别已经是错误的了。

语音识别的返回结果JSON中endtime =-1是什么意思？

表示当前句子未结束。当语音识别模式为“流式”时，才会存在中间结果。

C++ SDK语音合成时传入的文本没有采用UTF-8编码会有什么错误信息？

如果传入的文本没有采用UTF-8编码，在文本中含有中文字符时，语音合成SDK调用start函数会失败，返回错误信息：`Socket recv failed, errorCode: 0`。错误码为0表示服务端已经关闭了连接，此时应检查传入的文本是否采用UTF-8编码。

服务端返回的状态码都有哪些？

- HTTP状态`200`表示请求成功。
- HTTP状态`4XX`表示客户端错误。
- HTTP状态`5XX`表示服务端错误。

具体状态码，请参见各服务中的说明。

录音文件识别存在一次请求返回两次相同的结果？

如果是如下情况属正常现象：您提交的语音文件是双声道，且两个声道语音内容一样。

9.移动端SDK使用问题

本文为您介绍移动端SDK的常见问题。

为什么语音识别准确率很低，有时只识别出几个字？

请检查音频数据的采样率与管控台应用的模型是否一致，以及音频是否是单通道录音（有录音文件识别支持双通道录音的识别）。

初始化失败的可能原因？

请检查是否使用正确的AccessKey ID、AccessKey secret生成Access Token，并填入正确Appkey、Access Token、workspace等必选参数。

开始识别失败的可能原因？

SDK为单例模式，请确认上一个识别已经结束后再开始新的识别。

为何开始识别后没有识别结果？

确认如下信息：

- 初始化成功。
- 开始识别接口调用成功，且正确使用参数vad_mode。
- 有音频状态回调返回且已正确开启录音。

无问题的情况下，若仍然没有识别结果，则一般会有EVENT_ASR_ERROR事件发生，根据事件携带的错误码进行定位。

如何进行log控制与音频保存？

SDK提供多级log控制，在初始化接口中配置。

同时提供音频数据的保存方便问题定位，需要设置save_wav和debug_path初始化参数，详情请参见接口说明。

> **说明**
>
> 实时语音识别的save_wav和debug_path参数含义与一句话识别相同。

调用上有什么限制？

SDK已经对语音服务的访问做了封装，对您而言只要调用开始接口，在回调中进行适当事件处理。一般需要处理错误事件和识别结果事件。注意不能在回调中直接调用SDK的接口，可能导致死锁发生。

为什么链接不到framework？

framework中代码采用Objective-C和C++混合编写而成，所以需要使用.mm后缀文件进行调用，同时请确保工程的头文件路径与库文件路径设置正确。

SDK错误事件如何定位原因？

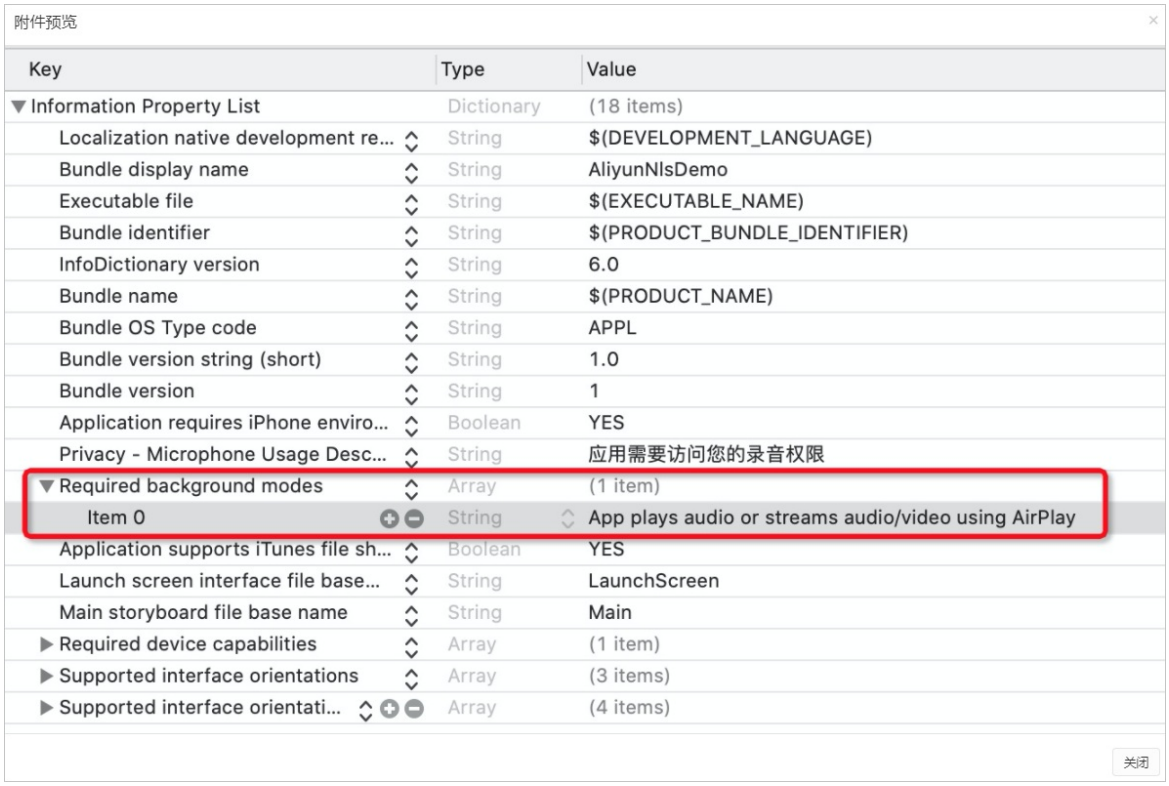
在SDK上报错误事件时，首先根据错误码查看原因，一般能够根据错误码获得问题的初步原因。

如果错误码为DEFAULT_NLS_ERROR，需要查看事件返回的结果字段，其中包含服务端的错误码、错误信息和task_id，然后根据这些信息进行分析或提交工单获得支持。

是否支持后台处理？

SDK本身不限制前后台，iOS SDK的样例工程默认仅支持前台处理，如果您需要支持后台处理，可以做如下修改：

1. 在工程Info.list中添加Required background modes配置，并在该配置下添加item，Value设置为 `App plays audio or streams audio/video using AirPlay` 。



2. 在录音模块中进入后台时，不停止录音。亦即NLSVoiceRecorder.m中_appResignActive接口中不做停止录音调用。


附件预览
NUIdemo
NUIdemo
audio
NLSVoiceRecorder.m
-_appResignActive
347 - (void)_unregisterForBackgroundNotifications {
348 [[NSNotificationCenter defaultCenter] removeObserver:self];
349 }
350
351
352 - (void)_appResignActive {
353 _inBackground = true;
354 // AudioSessionSetActive(NO);
355 }
356
357 - (void)_appEnterForeground {
358 _inBackground = false;
359 }
360
361 @end
362
关闭