智能语音交互

语音合成(TTS)

语音合成(TTS)

简介

注意：要使用语音合成服务需要先注册阿里云账号，并开通智能语音服务，具体步骤请参考账号和服务申请。

语音合成服务（TTS），就是将文本转成语音的服务。阿里云语音服务为用户提供语音合成的基础服务，服务器将需要进行合成的文本传送到服务器端，服务器进行语音合成后，以语音数据流的形式返回给SDK，用户可直接进行语音数据的播放或存储。

同时，阿里云语音合成服务提供给用户丰富的接口，可以设置不同的发音人、语速、音量等，获取的语音形式也很丰富，包括PCM、WAV、MP3等格式，可以极大的简化语音类App的开发。

支持的app_key

「语音合成」支持的app_key列表如下，此列表与「一句话识别」支持的app_key列表一致，用户可根据需要选择app_key。若用户单独使用「语音合成」，推荐使用nls-service。

app_key	备注
nls-service	推荐使用。可用于「一句话识别」和「语音合成」
nls-service-streaming	可用于「一句话识别」和「语音合成」
nls-service-tv	可用于「一句话识别」和「语音合成」
nls-service-shopping	可用于「一句话识别」和「语音合成」
nls-service-care	可用于「一句话识别」和「语音合成」
nls-service-en-opus	可用于「一句话识别」和「语音合成」

调用限制

目前免费用户调用限制10个并发。

Java SDK

功能介绍

语音Java SDK提供提供将文本转为普通话语音的语音合成功能。

获取appkey

获取appkey

SDK下载地址

语音合成JavaSDK

压缩包内为Java Demo工程，SDK jar包在src/main/java/resources/目录下，用户需要在IDE中操作将jar包导入。

- IDEA 中的操作方法为在jar包上右键，选”Add as library...”。
- Eclipse中的操作方法为在工程上右键，选“Build Path”->“Add Library...”

示例说明

SDK调用顺序

创建一个NlsClient的实例并调用init()方法来初始化客户端

提取语音数据并创建语音识别请求，至少填写appKey及需要识别的语音数据的格式。创建一个NlsListener的实现类。

调用NlsClient的createNlsFuture（第2步中的listener实例作为入参之一，用来处理返回结果）方法获取future，通过future的sendVoice方法来发送语音数据并在listener中处理返回结果。

通过future的sendFinishSignal来结束语音文件的发送，ASR服务收到这个结束信号后，会返回处理结果。

如有多个识别需要，重复步骤2-4。

调用NlsClient的close()方法来关闭客户端并释放资源。

SDK调用注意事项

NlsClient创建一次可以重复使用,每次创建消耗性能

NlsClient使用了netty的框架，创建时比较消耗时间和资源，但创建之后可以重复利用。建议调用程序将NlsClient的创建和关闭与程序本身的生命周期结合。

每一次调用语音识别请求时注入的NlsListener只对本次语音识别的生命周期负责。

每次调用createNlsFuture(NlsRequest req, com.alibaba.idst.nls.event.NlsListener listener)方法做语音识别和对话时，注入的listener只对本次识别起作用。其他的语音识别进程不会触发该listener。

NlsRequest里面的语音格式请与语音文件的格式保持一致。目前SDK支持pcm和opu格式，opu格式请参考opus编解码文档进行压缩。

sdk只会根据该格式来切分和发送语音文件，如果两个不一致，后续的一切都是错误的。

NlsListener的实现类里面，处理返回结果的代码尽量耗时剪短，最好不要涉及I/O操作。

NlsListener的实现类中的处理方法是在NlsClient的语音识别线程中调用的，太长的操作时间导致线程不能及时释放会影响其他识别进程的顺利进行，同时也会影响整个程序的loading。I/O操作应该禁止在该实现类中出现，最好使用触发的方式将操作转移到其他线程中去进行。

并发或多线程支持，如果需要在您的应用支持多个并发请求，请不要重复创建NlsClient对象。正确的做法是构建不同的NlsRequest对象，同时创建不同的NlsListener，并传入NlsRequest对象。这样就可以并发不同请求并且拿到正确的相应的结果。

重要接口说明

com.alibaba.idst.nls.NlsClient

语音sdk对外暴露的类，调用程序通过调用该类的init()、close()、createNlsFuture()等方法来打开、关闭或发送语音数据。

初始化NlsClient

public void init((boolean sslMode, int port)

- 说明 初始化NlsClient，创建好websocket client factory

参数

- sslMode 是否采用ssl模式，一般设置为true
- port 端口号，一般为443

返回值 null

设置服务器地址

public void setHost(String host)

- 说明 设置服务器地址

参数

返回值 null

实例化NlsFuture请求

public NlsFuture createNlsFuture(NlsRequest req, com.alibaba.idst.nls.event.NlsListener listener)

- 说明 实例化NlsFuture请求，传入请求和监听器

参数

- req 请求对象
- listener 回调数据的监听器

返回值 future

关闭NlsClient

public void close()

- 说明 关闭websocket client factory，释放资源

参数

null

返回值 null

com.alibaba.idst.nls.event.NlsListener

语音识别是一个非常漫长的过程，为了不影响服务端的正常工作，语音sdk被设计成异步模式，调用程序将语音客户端建立好并将语音数据交给sdk后就可以离开。在收到语音数据之后的处理需要通过实现该接口来完成。每一个识别过程创建的时候都需要注入侦听者以响应语音识别的结果。

识别结果回调

void onMessageReceived(NlsEvent e);

- 说明 识别结果回调

参数

- NlsEvent Nls服务结果的对象 识别结果在e.getResponse()中

返回值 null

识别失败回调

void onOperationFailed(NlsEvent e);

- 说明 识别失败回调

参数

- NlsEvent Nls服务结果的对象 错误信息在e.getErrorMessage()中

返回值 null

socket 连接关闭的回调

void onChannelClosed(NlsEvent e);

- 说明 socket 连接关闭的回调

参数

- NlsEvent Nls服务结果的对象

返回值 null

com.alibaba.idst.nls.protocol.NlsRequest

语音识别的请求封装对象。调用程序需要至少在请求对象里设定好调用者的appKey和语音数据的格式以便后台服务能正确识别并识别语音。

初始化Nls 请求：NlsRequest mNlsRequest = new NlsRequest(proto)

public void setAppKey(String app_key)

- 说明 设置应用的appkey，appkey需要先申请再使用。
- 参数
    - app_key
- 返回值 null

public void authorize(String id, String secret)

- 说明 数加认证模块，所有的请求都必须通过authorize方法认证通过，才可以使用。 id和secret需要申请获取。
- 参数
    - id id。
    - secret 对应密钥。
- 返回值 null

public void setTtsRequest(String tts_text)

- 说明 TTS 服务调用接口，传入文本，返回语音结果
- 参数
    - tts_text 用户输入的文本请求
- 返回值 null

public void setTtsRequest(String tts_text, String sample_rate)

- 说明 TTS 服务调用接口，传入文本，返回语音结果
- 参数
    - tts_text 用户输入的文本请求
    - sample_rate 返回语音采样率 支持8k，16k等
- 返回值 null

public void setTtsVoice(String voiceName)

- 说明 设置tts返回发音人，有两个选项 xiaoyun(女声)、xiaogang(男声)。
- 参数
    - voiceName xiaoyun/xiaogang
- 返回值 null

public void setTtsNus(int nus)

说明 nus模式选择，0为参数，1为拼接（电话客户场景请选0）

参数是指最终合成语音的时候是通过一组参数生成的语音数据，拼接的意思是通过拼接原始的录音数据得到语音数据

- 参数
    - nus 取值范围0或1，默认1
- 返回值 null

public void setTtsEncodeType(String encodeType)

- 说明 设置tts返回语音的格式类型，目前支持pcm、wav、alaw、mp3。
- 参数
  - encodeType 类型
- 返回值 null

public void setTtsSpeechRate(String speechRate)

- 说明 语速，默认输入为0，阈值-500~500，从慢到快。
- 参数
  - speechRate 语速
- 返回值 null

public void setTtsVolume(String volume)

- 说明 音量大小默认50，阈值0-100。
- 参数
  - volume 音量
- 返回值 null

public void setTtsBackgroundMusic(int id)

- 说明 设置开启背景音乐
- 参数
  - id 音乐id 0，1
- 返回值 null

public void setTtsPitchRate(int pitchRate)

- 说明 设置tts发音语调，-500非常低沉，500非常高亢
- 参数
  - pitchRate 调整语调,-500~500
- 返回值 null

com.alibaba.idst.nls.internal.protocol.NlsRequestProto

NlsRequest对象初始化时需要注入的参数。

public void setApp_id(String app_id)

- 设置application_id.

public void setApp_user_id(String app_user_id)

- 设置app_user_id.

com.alibaba.idst.nls.NlsFuture

NlsFuture是具体操作语音的对象类，语音数据的发送/接收都通过这个类进行操作。

public NlsFuture sendVoice(byte[] data, int offset, int length)

- 说明 语音识别时，发送语音文件的方法
- 参数
    - data byte[]类型的语音流
    - offset
    - length
- 返回值 NlsFuture

public NlsFuture sendFinishSignal()

- 说明 语音识别结束时，发送结束符

  参数

  返回值 NlsFuture

public byte[] read()

- 说明 TTS（文本转语音）时，输出语音流
- 参数
- 返回值 byte[]

public boolean await(int timeout)

- 说明 请求超时时间
- 参数
    - timeout 请求发送出去后，等待服务端结果返回的超时时间，单位ms
- 返回值 true false

com.alibaba.idst.nls.protocol.NlsResponse

语音识别的返回结果封装对象。返回结果告知语音识别是否成功，语音识别结果或部分结果(视调用程序需要的返回方式而定)，语音识别是否已经结束。

- 说明 TTS（文本转对话）的状态等文本的提升结果，语音结果通过future.read()获取
- 参数
    - null
- 返回值 String

错误码

状态	status_code	CloseFrame状态码	HTTP语义
成功	200	1000	成功处理

请求格式有误	400	4400	错误请求
需要鉴权信息	401	4401	请求要求身份验证
鉴权失败	403	4403	服务器拒绝请求
超出最大并发量	429	4429	太多请求
请求超时	408	4408	处理请求超时
处理出错	500	4500	服务器内部错误
服务不可用	503	4503	服务不可用

完整示例

```
package com.demo;
import com.alibaba.idst.nls.protocol.NlsResponse;
import com.alibaba.idst.nls.NlsClient;
import com.alibaba.idst.nls.NlsFuture;
import com.alibaba.idst.nls.event.NlsEvent;
import com.alibaba.idst.nls.event.NlsListener;
import com.alibaba.idst.nls.protocol.NlsRequest;

public class TTSDemo implements NlsListener {
private NlsClient client = new NlsClient();

public TTSDemo() {
System.out.println("init Nls client...");
// 初始化NlsClient
client.init();
}

public void shutDown() {
System.out.println("close NLS client");
// 关闭客户端并释放资源
client.close();
System.out.println("demo done");
}


public void startTTS() {
File file = new File("tts.wav");
if(!file.exists()) {
try {
file.createNewFile();
} catch (Exception e) {
e.printStackTrace();
}
}
NlsRequest req = new NlsRequest();
String appkey = "nls-service";
req.setAppKey(appkey); // 设置语音文件格式
```

```
req.setTtsRequest(tts_text); //传入测试文本，返回语音结果
req.setTtsEncodeType("wav");//返回语音数据格式，支持pcm,wav.alaw
req.setTtsVolume(30); //音量大小默认50，阈值0-100
req.setTtsSpeechRate(0); //语速，阈值-500~500
req.setTtsBackgroundMusic(1);//背景音乐编号
req.authorize("", ""); // 请替换为用户申请到的Access Key ID和Access Key Secret
try {
FileOutputStream fileOutputStream = new FileOutputStream(file);
NlsFuture future = client.createNlsFuture(req, this); // 实例化请求,传入请求和监听器
int total_len = 0;
byte[] data ;
while((data = future.read()) != null) {
fileOutputStream.write(data, 0, data.length);
total_len += data.length;
System.out.println("tts length " + data.length);
}
fileOutputStream.close();
System.out.println("tts audio file size is :" + total_len);
future.await(10000); // 设置服务端结果返回的超时时间

} catch (Exception e) {
e.printStackTrace();
}
}


@Override
public void onMessageReceived(NlsEvent e) {
NlsResponse response = e.getResponse();
String result = "";
int statusCode = response.getStatus_code();
if (response.getTts_ret() != null) {
result += "\nget tts result: statusCode=[" + statusCode + "], " + response.getTts_ret();
}

if (result != null) {
System.out.println(result);
}
else {
System.out.println(response.jsonResults.toString());
}
}

@Override
public void onOperationFailed(NlsEvent e) {
//识别失败的回调
System.out.print("on operation failed: ");
System.out.println(e.getErrorMessage());
}

@Override
public void onChannelClosed(NlsEvent e) {
//socket 连接关闭的回调
System.out.println("on websocket closed.");
}
```

```
/**
* @param args
*/
public static void main(String[] args) {
TTSDemo ttsDemo = new TTSDemo();
ttsDemo.startTTS();
ttsDemo.shutDown();
}
}
```

C++ SDK

一、SDK 使用说明

功能介绍

C++ SDK提供将文字转为语音的功能。

SDK下载地址

文字转语音C++ SDK

demo 编译，运行步骤

1. Linux环境下，请确认本地系统以安装Cmake，最低版本2.4
2. 解压SDK压缩包，执行[./build.sh]编译demo
3. 编译完毕，进入demo目录，执行demo

- 由于链接顺序对编译会造成影响，如果提示编译错误，可以调整CMakeLists.txt调整-lrealTimeUnity -lopus前后顺序
- 由于此sdk没有采用c11，如果报C++11编译错误，可以在CMakeLists.txt中添加-D_GLIBCXX_USE_CXX11_ABI=0

SDK调用顺序

创建一个的NlsSpeechCallback实例callbck，并分别设置结果返回、操作错误和通道关闭的回调函

数.

创建一个NlsClient的对象nlc，该对象只需创建一次并且可以重复使用。

准备好config.txt配置文件，其内包含app-key、url等参数，详细见本文末尾的示例。

通过调用2步中的nlc对象的createAsrRequest方法获得一个NlsRequest 对象的指针(用完记得释放)，该NlsRequest对象不能重复使用，但是可以重复创建。

调用4中返回的NlsRequest对象的Authorize方法进行设置用户id和scret。

调用4中 NlsRequest对象的start方法.

start返回之后，调用4中NlsRequest对象的stop方法

- 注意：Request的create()——>start()——>stop()——>delete操作，请在一个线程内部完成。不要夸线程。否则会有异常。Demo.cpp内部实现亦是在一个线程内部完成。

SDK调用注意事项

sdk采用ISO 标准c++编写，运行环境最低要求：glibc:2.5 gcc 版本：4.1.2.

sdk内部采用pthread多线程机制，因此在linux环境下可以直接使用，如果需要在windows环境下使用，需要额外安装能够在windows环境下运行的pthread的支持库(ftp://sourceware.org/pub/pthreads-win32/).

针对linux环境，sdk提供的库文件为librealTimeUnity.so 和librealTimeUnity-32.so，分别对应64位运行环境和32位运行环境.

sdk依赖于第三方库，主要是(libssl.a libcrypto.a libopus.so.0)，其中前两个来自于openssl，版本为1.0.2j，最后一个是用于opus编解码，版本不限.

sdk提供的第三方库均为64位，如需32位，请根据如上版本自行下载源码进行编译.

重要接口说明

NlsClient：语音SDK的Client，相当于所有语音相关处理类的factory，全局创建一个实例即可。线程安全。可通过NlsClient创建语音识别request对象。

```
    /**
* @brief 设置日志文件与存储路径
* @param logOutputFile 日志文件
* @param logLevel 日志级别，默认1（Error : 1、Warn : 2、Debug : 3）
* @return 成功则返回0，失败返回-1
*/
int setLogConfig(const char* logOutputFile, int loglevel);

/**
* @brief 创建实时语音识别Request对象
* @param onResultReceivedEvent 事件回调接口
* @param config 配置文件
* @return 成功返回NlsRequest对象，否则返回NULL
*/
NlsRequest* createRealTimeRequest(NlsSpeechCallback* onResultReceivedEvent, const char* config);

/**
* @brief 创建一句话识别Request对象
* @param onResultReceivedEvent 事件回调接口
* @param config 配置文件
* @return 成功返回NlsRequest对象，否则返回NULL
*/
NlsRequest* createAsrRequest(NlsSpeechCallback* onResultReceivedEvent, const char* config);

/**
* @brief 创建语音合成Request对象
* @param onResultReceivedEvent 事件回调接口
* @param config 配置文件
* @return 成功返回NlsRequest对象，否则返回NULL
*/
NlsRequest* createTtsRequest(NlsSpeechCallback* onResultReceivedEvent, const char* config);

/**
* @brief 创建NLu Request对象
* @param onResultReceivedEvent 事件回调接口
* @param config 配置文件
* @return 成功返回NlsRequest对象，否则返回NULL
*/
NlsRequest* createNluRequest(NlsSpeechCallback* onResultReceivedEvent, const char* config);

/**
* @brief NlsClient对象实例
* @param sslInitial 是否初始化openssl 线程安全，默认为true
* @return NlsClient对象
*/
static NlsClient* getInstance(bool sslInitial = true);
```

NlsRequest：语音识别请求对象，通过调用该类的start()、stop()、sendAudio()等方法来打开、关闭或发送语音数据。

```
    /**
* @brief 参数设置
* @param value appKey字符串
* @return 成功则返回0，否则返回-1
*/
```

```
int SetParam(const char* str_key, const char* str_value); // add format

/**
* @brief 启动Request
* @note 阻塞操作，等待服务端响应、或10秒超时才会返回
* @return 成功则返回0，否则返回-1
*/
int Start();

/**
* @brief 会与服务端确认关闭，正常停止SpeechRecognizerRequest链接操作
* @note 阻塞操作，等待服务端响应才会返回
* @return 成功则返回0，否则返回-1
*/
int Stop();

/**
* @brief 不会与服务端确认关闭，直接关闭SpeechRecognizerRequest链接
* @note 正常情况下，建议使用stop结束操作
* @return 成功则返回0，否则返回-1
*/
int Cancel();

/**
* @brief 授权校验
* @param id
* @param scret
* @return 成功则返回实际发送长度，失败返回-1
*/
int Authorize(const char* id, const char* scret);

/**
* @brief 发送语音数据
* @param data 语音数据
* @param dataSize 语音数据长度
* @return 成功则返回实际发送长度，失败返回-1
*/
int SendAudio(char* data, size_t num_byte);
```

NlsSpeechCallback：回调函数集合的对象，语音结果、异常等回调的统一入口。

```
    /**
* @brief 设置结果消息接受回调函数
* @param _event 回调方法
* @param para 用户传入参数, 默认为NULL
* @return void
*/
void setOnMessageReceiced(NlsCallbackMethod onMessageReceived,void*para=NULL);

/**
* @brief 设置错误回调函数
* @note 在请求过程中出现错误时，触发该回调。用户可以在事件的消息头中检查状态码和状态消息，以确认失败的具体原因
。
* @param _event 回调方法
* @param para 用户传入参数, 默认为NULL
```

```
* @return void
*/
void setOnOperationFailed(NlsCallbackMethod _event, void*para=NULL);

/**
* @brief 设置通道关闭回调函数
* @note 在请求过程中通道关闭时，触发该回调.
* @param _event 回调方法
* @param para 用户传入参数, 默认为NULL
* @return void
*/
void setOnChannelClosed(NlsCallbackMethod _event, void*para=NULL);

/**
* @brief 二进制音频数据接收回调函数
* @note 仅在tts语音合成中触发该回调
* @param _event 回调方法
* @param para 用户传入参数, 默认为NULL
* @return void
*/
void setOnBinaryDataReceived(NlsCallbackMethod _event, void*para=NULL);
```

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

```
    /**
* @brief 获取状态码
* @note 正常情况为0或者200，失败时对应失败的错误码。错误码参考SDK文档说明。
* @return string
*/
std::string getStausCode();

/**
* @brief 获取云端返回的结果
* @note json格式
* @return string
*/
std::string getResponse();

/**
* @brief 获取NlsRequest操作过程中出现失败时的错误信息
* @note 在Failed回调函数中使用
* @return string
*/
std::string getErrorMessage();

/**
* @brief 获取云端返回的二进制数据
* @note 仅用于tts语音合成功能
* @return vector<unsigned char>
*/
std::vector<unsigned char> getBinaryData();

/**
* @brief 获取当前所发生Event的类型
* @note Event值参照NlsClient.h说明
```

```
* @return EventType
*/
EventType getMsgType();

/**
* @brief 获取当前request对象id
* @return string
*/
std::string getId();

/**
* @brief 设置当前request对象id
* @return
*/
void setId(std::string id);
```

更多详细介绍参见SDK压缩包内部include/NlsClient.h。

错误码

服务端错误码：

状态	status_code	CloseFrame状态码	HTTP语义
成功	200	1000	成功处理
请求格式有误	400	4400	错误请求
需要鉴权信息	401	4401	请求要求身份验证
鉴权失败	403	4403	服务器拒绝请求
超出最大并发量	429	4429	太多请求
请求超时	408	4408	处理请求超时
处理出错	500	4500	服务器内部错误
服务不可用	503	4503	服务不可用
服务不可用	503	4503	服务不可用

SDK错误码

状态	status_code
“status_code not found”	999
“GetInetAddressByHostname fail”	1028
“asr frame type must be text”	1030
“Json reader fail”	1031
“Authorization mustn’ t be null”	10002
“get time fail”	10003
“PING no implementaion”	10004

“ConnectToHttp fail”	10005
“Json convert fail”	10006
“SSLv23_client_method fail”	10015
“SSL_CTX_new fail”	10016
“SSL_new fail”	10017
“not support mode”	10020
“ssl connect fail”	参考SSL官方错误码
“convert to utf8 error”	参考系统错误码

config.txt 内容如下：

```
#注意：
#1. 以#开头的行为注释，处理时会直接跳过该行。
#2. 配置中每行只能出现一条配置且以key:value的形式出现，切忌中间、行头、行尾不要出现空格，除非key, value字段中本身含有空格
Url:wss://nls.dataapi.aliyun.com:443
AppKey:nls-service
#Language:EN/CHN
#TtsEnable tts请求为true
TtsEnable:true
#TtsEncodeType 编码格式，支持pcm，wav，mp3。默认值是pcm
TtsEncodeType:pcm
#TtsVolume 音量，范围是0到100
TtsVolume:100
#TtsNus 合成方式，0表示HTS，1表示NUS，2表示RUS
TtsNus:0
#TtsVoice 说话人名称，支持xiaoyun，xiaogang，xiaokubao
TtsVoice:xiaoyun
#TtsSpeechRate 语速，范围是-500到500。
TtsSpeechRate:0
#BackgroundMusicId 背景音乐ID，0表示不添加背景音乐
BackgroundMusicId:1
#BackgroundMusicOffset 在添加背景音乐时，指定背景音乐的偏移
BackgroundMusicOffset:0
#BstreamAttached 是否附加流
BstreamAttached:false
#TtsReference 指定发音
#TtsReference:
#TtsPitchRate 语调，范围是-500到500。
TtsPitchRate:0
#TtsSampleRate 采样率，支持16000或8000
TtsSampleRate:16000
#TtsReq 需要转为语音的文字
TtsReq:人民公园不但风景美丽，而且还给我们带来了无穷的乐趣。
```

完整示例

```
#include <string>
```

```
#include <iostream>
#include <vector>
#include <fstream>
#include <sstream>
#include "pthread.h"
#include "NlsClient.h"
#include <stdlib.h>
#include <string.h>
#include <vector>
#ifdef _WINDOWS_COMPILE_
#include <windows.h>
#else
#include <unistd.h>
#endif
using namespace std;

NlsSpeechCallback* gCallback = NULL;
NlsClient* gNlc = NULL;
vector<unsigned char> databuf;

struct ParamStruct {
std::string filename;
std::string config;
int threadcount;
int count;
std::string id;
std::string key;
bool isSendData;
};

void* func(void* arg) {
try {
ParamStruct* tst = (ParamStruct*)arg;
if (tst == NULL) {
std::cout << "filename is not valid" << endl;
return 0;
}

/*******************第一种调用方式***************************/
//创建一句话识别 NlsRequest，第一个参数为gCallback对象，第二个参数为config.txt文件
NlsRequest* request = gNlc->createTtsRequest(gCallback, tst->config.c_str());
/*********************************************************/

/*****************以参数设置方式创建request******************
NlsRequest* request = nlc.createTtsRequest(gCallback, NULL);
request->SetParam("AppKey","nls-service");
request->SetParam("TtsEnable","true");
request->SetParam("TtsEncodeType","wav");
request->SetParam("TtsVolume","100");
request->SetParam("Url","wss://nls.dataapi.aliyun.com:443");
request->SetParam("Id","1234567");
request->SetParam("TtsNus","0");
request->SetParam("TtsVoice","xiaoyun");
request->SetParam("TtsSpeechRate","0");
request->SetParam("BackgroundMusicId","1");
request->SetParam("BackgroundMusicOffset","0");
```

```
request->SetParam("BstreamAttached","false");
**********************************************************/
if (request == NULL) {
std::cout << "createTtsRequest fail" << endl;
return 0;
}

request->SetParam("Id", "1234567");
request->Authorize(tst->id.c_str(), tst->key.c_str());

/*
*start()为非异步操作，发送start指令之后，会等待服务端响应、或超时之后才返回
*/
if (request->Start() < 0) {
std::cout << "start fail" << endl;
delete request;
request = NULL;
return 0;
}

/*
*stop()非异步，在接受到服务端响应，或者超时之后，才会返回.
*返回之后，可以调用delete request
*/
request->Stop();

delete request;
request = NULL;

if (databuf.size() > 0) {
ofstream fss;
fss.open(tst->filename.c_str(), ios::binary | ios::out);
fss.write((char*)&databuf[0], databuf.size());
fss.close();
}
} catch (const char* e) {
std::cout << e << endl;
}
return 0;
}

void OnResultDataRecved(NlsEvent* str,void* para=NULL) {
ParamStruct* tst = (ParamStruct*)para;
std::cout << "Id:" << (*(int*)para) << " " << str->getResponse() << endl;
}

void OnOperationFailed(NlsEvent* str, void* para=NULL) {
ParamStruct* tst = (ParamStruct*)para;
std::cout << "Id:" << (*(int*)para) << " " << str->getErrorMessage() << endl;
}

void OnChannelCloseed(NlsEvent* str, void* para=NULL) {
ParamStruct* tst = (ParamStruct*)para;
std::cout << "Id:" << (*(int*)para) << " " << str->getResponse() << endl;
}
```

```
void OnBinaryDataRecved(NlsEvent* str, void* para=NULL) {
vector<unsigned char> data = str->getBinaryData();
databuf.insert(databuf.end(), data.begin(), data.end());
}

int main(int arc, char* argv[]) {
try {
if (arc <= 7) {
cout << "param is not valid. Usage: demo test.wav config.txt 1 1 [your-id] [your-scret] 0" << endl;
return -1;
}

gNlc = NlsClient::getInstance();
int ret = gNlc->setLogConfig("log-tts.txt",3);

int id = 123434;
gCallback = new NlsSpeechCallback();
gCallback->setOnMessageReceiced(OnResultDataRecved, &id);
gCallback->setOnOperationFailed(OnOperationFailed, &id);
gCallback->setOnChannelClosed(OnChannelCloseed, &id);
gCallback->setOnBinaryDataReceived(OnBinaryDataRecved, &id);

ParamStruct pa;
pa.filename = argv[1];
pa.config = argv[2];
pa.threadcount = atoi(argv[3]);
pa.count = atoi(argv[4]);
pa.id = argv[5];
pa.key = argv[6];
pa.isSendData = strcmp(argv[7], "0") == 0 ? false : true;

const int ThreadCount = pa.threadcount;
vector<pthread_t> tarr(ThreadCount);
for (int i = 0; i < pa.count; i++) {
for (int j = 0; j < ThreadCount; j++)
pthread_create(&tarr[j], NULL, &func, (void*)&pa);
for (int j = 0; j < ThreadCount; j++)
pthread_join(tarr[j], NULL);
}

if(gNlc != NULL) {
delete gNlc;
gNlc = NULL;
}

delete gCallback;
gCallback = NULL;
} catch (exception e) {
cout << e.what() << endl;
}

return 0;

}
```

Android SDK

功能介绍

语音Android SDK提供提供将文本转为普通话语音的语音合成功能。

阿里云语音服务SDK（NLSClinet），是运行于android平台的基础语音识别、自然语言理解和语音合成的基础服务，本服务通过Jar包和.so库的形式导入，用户通过本指南可以方便的接入。

SDK下载地址

语音合成AndroidSDK

获取appkey

获取appkey

开发包目录

文件夹	内容
libs/	
libs/NlsClientSdk.jar	NLSClinet的服务包
libs/armeabi	
libs/armeabi/libztcodec2.so	
libs/armeabi-v7a	
libs/armeabi-v7a/libztcodec2.so	
libs/x86	
libs/x86/libztcodec2.so	

- 本服务使用了解析json的gson.jar,需要用户自行导入。

集成指南

导入服务包

将下载的服务包解压后，将NlsClientSdk.jar，以及对应架构的.so包导入你的项目中的libs/目录下。

Android权限管理

在AndroidManifest.xml文件中添加以下权限申请：

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.WRITE_SETTINGS" />
```

Proguard配置

如果代码使用了混淆，请在proguard-rules.pro中配置：

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

重要接口说明

com.alibaba.idst.nls.NlsClient

语音服务的核心服务类，客户端程序通过调用该类的cancel()、start()、stop()等方法来打开、关闭或发送语音数据。

全局配置

public static void configure(Context appContext)

- 说明 全局配置，在初始化NlsClient前需要被调用，建议放在Application的onCreate()里
- 参数
  - appContext 传入ApplicationContext参数，用于识别引擎内部访问和Android上下文相关的资源
- 返回值 null

实例化NlsClient对象

public static NlsClient newInstance(Context context,NlsListener nlsListener,StageListener stageListener, NlsRequest nlsRequest)

- 说明 获得一个NlsClient, 通过该方法实例化NlsClient。
- 参数
    - context 传入Context参数，用于识别引擎内部访问和Android上下文相关的资源
    - nlsListener 识别相关的回调接口，用于通知客户端识别结果
    - stageListener 引擎状态回调接口，用于通知客户端当前的引擎状态以及录音音量等
- 返回值 NlsClient

TTS请求接口

ppublic boolean PostTtsRequest(String InputText,String SampleRate)

- 说明 TTS 服务调用接口，传入文本，返回语音结果
- 参数
    - InputText 用户输入的文本请求
    - SampleRate 用户传入码率 如16000
- 返回值 true:请求成功发送 false:发送失败

其它接口：

Log开关

public static void openLog(boolean isOpen)

- 说明 log开关，标识是否需要通过logcat打印识别引擎的相关log
- 参数
    - isOpen true:打印log false:关闭log
- 返回值 null

com.alibaba.idst.nls.internal.protocol.NlsRequest

语音服务的请求封装对象。调用程序需要至少在请求对象里设定好调用者的appKey和语音数据的格式以便后台服务能正确识别并翻译语音。

初始化Nls 请求：NlsRequest mNlsRequest = new NlsRequest()

public void setApp_key(String app_key)

- 说明 设置应用的appkey，appkey需要先申请再使用。
- 参数
    - app_key
- 返回值 null

public void authorize(String id, String secret)

- 说明 数加认证模块，所有的请求都必须通过authorize方法认证通过，才可以使用。 id和secret需要申请获取。
- 参数

- id id。
- secret 对应密钥。

- 返回值 null

public void setTtsEncodeType(String encodeType)

- 说明 设置tts返回语音的格式类型，目前支持pcm、wav、alaw。
- 参数
  - encodeType 类型
- 返回值 null

public void setTtsSpeechRate(String speechRate)

- 说明 语速，默认输入为0，阈值-500~500。
- 参数
  - speechRate 语速
- 返回值 null

public void setTtsVolume(String volume)

- 说明 音量大小默认50，阈值0-100。
- 参数
  - volume 音量
- 返回值 null

com.alibaba.idst.nls.internal.protocol.NlsRequestProto

NlsRequest对象初始化时需要注入的参数。

public void setApp_id(String app_id)

- 设置application_id.

public void setApp_user_id(String app_user_id)

- 设置app_user_id.

public void setQuery_type(String query_type)

- 设置query_type.

com.alibaba.idst.nls.StageListener

语音服务引擎状态变更回调接口，服务状态的改变、音量大小的回调、语音文件的生成通过本接口获取。

录音开始的回调

public void onStartRecording(NlsClient recognizer)

- 说明 录音开始

参数

返回值 NlsClient

录音结束的回调

public void onStopRecording(NlsClient recognizer)

- 说明 录音结束

参数

返回值 NlsClient

识别开始的回调

public void onStartRecognizing(NlsClient recognizer)

- 说明 识别开始

参数

返回值 NlsClient

识别结束的回调

public void onStopRecognizing(NlsClient recognizer)

- 说明 识别结束

参数

返回值 NlsClient

音量大小回调

public void onVoiceVolume(int volume)

说明 获取音量

参数

- volume 录音音量，范围0-100

返回值 null

com.alibaba.idst.nls.NlsListener

语音服务结果的回调类，语音服务是一个相对较长过程，为了不影响客户端的正常工作，语音sdk被设计成异步模式，调用程序将语音客户端建立好并将语音数据交给sdk后就可以离开。在收到语音数据之后的处理需要通过实现该接口来完成。每一个识别过程创建的时候都需要注入侦听者以响应语音服务的结果。

tts文本转语音，语音流回调接口

public void onTtsResult(int status, byte[] ttsResult)

- 说明 tts识别结束时回调接口，识别结束时回调。
- 参数
  - status 标识识别结果的状态，详情见下方。
  - result 返回识别结果，结果byte[]类型。

返回值 null

status 状态：

- 客户端错误码

字段名	值	含义
TTS_BEGIN	6	tts 语音流开始
TTS_TRANSFERRING	7	tts 中间结果
TTS_OVER	8	tts 语音流结束
CONNECT_ERROR	530	socket请求失败

- 服务端返回结果错误码

状态	status_code	CloseFrame状态码	HTTP语义
成功	200	1000	成功处理
请求格式有误	400	4400	错误请求
需要鉴权信息	401	4401	请求要求身份验证
鉴权失败	403	4403	服务器拒绝请求
超出最大并发量	429	4429	太多请求
请求超时	408	4408	处理请求超时
处理出错	500	4500	服务器内部错误
服务不可用	503	4503	服务不可用

错误码

- 客户端错误码

字段名	错误码	含义
SUCCESS	0	成功
RECOGNIZE_ERROR	1	识别失败
USER_CANCEL	520	用户取消
CONNECT_ERROR	530	网络及通讯异常
NOTHING	540	语音服务异常
RECORDING_ERROR	550	录音及语音识别异常
ERROR_CLICK_TOOMUCH	570	用户点击过快

- 服务端返回结果错误码

状态	status_code	CloseFrame状态码	HTTP语义
成功	200	1000	成功处理
请求格式有误	400	4400	错误请求
需要鉴权信息	401	4401	请求要求身份验证
鉴权失败	403	4403	服务器拒绝请求
超出最大并发量	429	4429	太多请求
请求超时	408	4408	处理请求超时
处理出错	500	4500	服务器内部错误
服务不可用	503	4503	服务不可用

完整示例

```
package com.alibaba.idst.nlsdemo;

import android.app.Activity;
import android.media.AudioFormat;
import android.media.AudioManager;
import android.media.AudioTrack;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
```

```
import android.widget.Toast;

import com.alibaba.idst.R;
import com.alibaba.idst.nls.NlsClient;
import com.alibaba.idst.nls.NlsListener;
import com.alibaba.idst.nls.internal.protocol.NlsRequest;
import com.alibaba.idst.nls.internal.protocol.NlsRequestProto;

public class PublicTtsActivity extends Activity {
private static final String TAG = "PublicTtsActivity";
private TextView UserContent;
private Button Send_User_Content;
private NlsClient mNlsClient;

private NlsRequest mNlsRequest;
private Context context;

int iMinBufSize = AudioTrack.getMinBufferSize(16000,
AudioFormat.CHANNEL_CONFIGURATION_MONO,
AudioFormat.ENCODING_PCM_16BIT);
AudioTrack audioTrack=new AudioTrack(AudioManager.STREAM_MUSIC, 16000,
AudioFormat.CHANNEL_CONFIGURATION_MONO, AudioFormat.ENCODING_PCM_16BIT,
iMinBufSize, AudioTrack.MODE_STREAM) ; //使用audioTrack播放返回的pcm数据

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);

setContentView(R.layout.activity_public_tts);
UserContent = (TextView) findViewById(R.id.UserContent_tts);
Send_User_Content = (Button) findViewById(R.id.send_tts);
context = getApplicationContext();
mNlsRequest = initNlsRequest();
String appkey = "nls-service"; //请设置简介页面的Appkey

mNlsRequest.setApp_key(appkey); //appkey请从 简介页面的appkey列表中获取
mNlsRequest.initTts(); //初始化tts请求

NlsClient.openLog(true);
NlsClient.configure(getApplicationContext()); //全局配置
mNlsClient = NlsClient.newInstance(this, mRecognizeListener, null ,mNlsRequest); //实例化NlsClient
initTtsContentButton();


}

private NlsRequest initNlsRequest(){
NlsRequestProto proto = new NlsRequestProto(context);
proto.setApp_user_id("xxx"); //设置用户名
return new NlsRequest(proto);

}

private void initTtsContentButton(){
Send_User_Content.setOnClickListener(new View.OnClickListener() {
@Override
```

```
public void onClick(View view) {
String user_input = UserContent.getText().toString();
if (user_input.equals("")){
Toast.makeText(PublicTtsActivity.this, "输入不能为空！", Toast.LENGTH_LONG).show();
}else {
mNlsRequest.setTtsEncodeType("pcm"); //返回语音数据格式，支持pcm,wav.alaw
mNlsRequest.setTtsVolume(50); //音量大小默认50，阈值0-100
mNlsRequest.setTtsSpeechRate(0);//语速，阈值-500~500
mNlsClient.PostTtsRequest(user_input); //用户输入文本
mNlsRequest.authorize("", ""); //请替换为用户申请到的数加认证key和密钥
audioTrack.play();
}
}
});
}


private NlsListener mRecognizeListener = new NlsListener() {
@Override
public void onTtsResult(int status, byte[] ttsResult){
switch (status) {
case NlsClient.ErrorCode.TTS_BEGIN :
audioTrack.play();
Log.e(TAG, "tts begin");
audioTrack.write(ttsResult, 0, ttsResult.length);
break;
case NlsClient.ErrorCode.TTS_TRANSFERRING :
Log.e(TAG,"tts transferring"+ttsResult.length);
audioTrack.write(ttsResult, 0, ttsResult.length);
break;
case NlsClient.ErrorCode.TTS_OVER :
audioTrack.stop();
Log.e(TAG,"tts over");
break;
case NlsClient.ErrorCode.CONNECT_ERROR :
Toast.makeText(PublicTtsActivity.this, "CONNECT ERROR", Toast.LENGTH_LONG).show();
break;
}
}
} ;



@Override
protected void onDestroy() {
audioTrack.release();
super.onDestroy();
}

}
```

iOS SDK

功能介绍

语音iOS SDK提供提供将文本转为普通话语音的语音合成功能。

SDK下载地址

语音合成iOSSDK

获取appkey

获取appkey

重要接口说明

发送语音请求的对象及方法NlsRequest.h

语音请求初始化方法

- (instancetype)init;

- 说明 语音识别、语音合成的语音请求初始化方法
- 返回值 self

设置语音请求的appkey

- (void)setAppkey:(NSString *)appKey;

- 说明 设置语音请求的appkey
- 参数
  - appKey
- 返回值 无

设置发送的请求是否需要带语音数据

- (void)setBstreamAttached:(BOOL)bstreamAttached;

- 说明 设置发送的请求是否需要带语音数据。若发送的是语音识别请求，则bstreamAttached为YES；若发送的是语音合成请求，则bstreamAttached为NO

- 参数
  - bstreamAttached 请求是否需要带语音数据
- 返回值 无

设置TTS文本

- (void)setTTSText:(NSString *)text;

- 说明 设置TTS文本
- 参数
  - text TTS文本
- 返回值 无

设置TTS语音编码格式

- (void)setTtsEncodeType:(NSString *)encode_type;

- 说明 设置TTS语音编码格式
- 参数
  - encode_type 语音数据编码，取值范围pcm,wav或alaw，默认pcm
- 返回值 无

设置TTS采样率

(void)setTTSSampleRate:(NSString *)sampleRate;

- 参数
  - sampleRate 采样率，取值范围8000～16000，默认16000
- 返回值 无

设置TTS语速

- (void)setTtsSpeechRate:(NSInteger)speechRate;

- 说明 设置TTS语音编码格式
- 参数
  - speechRate 播放速率,取值范围-500~500，默认0
- 返回值 无

设置TTS音量

- (void)setTtsVolume:(NSInteger)volume;

- 参数
  - volume 播放音量,取值范围0～100，默认50
- 返回值 无

设置TTS模式

- (void)setTtsNus:(NSInteger)nus;

- 参数
  - nus模式，取值范围0或1，默认1
- 返回值 无

数加验证

- (void)Authorize:(NSString *)authId withSecret:(NSString *)secret;

- 说明 数加验证，未经过数加验证的语音请求均为非法请求。
- 参数
  - authId 数加验证的ak_id
  - secret 数加验证的ak_secret
- 返回值 无

将语音请求NlsRequest对象转换成JSON字符串

+ (NSString *)getJSONStringfromNlsRequest:(NlsRequest *)nlsRequest;

- 说明 将语音请求NlsRequest对象转换成JSON字符串形式。
- 参数
  - nlsRequest NlsRequest对象
- 返回值 NlsRequest的JSON字符串

将对象转换成JSONString方法

+ (NSString *)getJSONString:(id)obj options:(NSJSONWritingOptions)options error:(NSError)error;**

- 说明 将object转换成JSONString。
- 参数
  - obj 被转化对象
  - options NSJSONWritingOptions
  - error NSError
- 返回值 NlsRequest的JSON字符串

将对象转换成NSDictionary方法

+ (NSDictionary *)getObjectData:(id)obj;

- 说明 将object转换成NSDictionary。
- 参数
  - obj 被转化对象
- 返回值 NSDictionary

语音服务SDK的核心类NlsRecognizer.h

语音服务SDK的核心类，封装了录音设备的初始化，压缩处理，语音检测（VAD）等复杂逻辑，自动的将语音数据同步传送到语音服务器上。开发者只需要传递正确delegate的，就能完成语音识别和语音合成

语音合成的关键回调函数

-(void)recognizer:(NlsRecognizer *)recognizer didCompleteTTSWithVoiceData:(NSData*)voiceData error:(NSError*)error;

- 说明 接收服务器返回的语音数据，多次回调，每次返回不大于8004字节的NSData数据，前4个字节为数据长度相关的数据。直到返回byte前4个字节为0000，停止回调。
- 参数
    - recognizer NlsRecognizer
    - voiceData 服务器传回的语音数据
    - error 语音合成错误和异常 NSError
- 返回值 无

开始返回语音合成数据

-(void)recognizerDidStartRecieveTTSData:(NlsRecognizer *)recognizer;

- 参数
    - NlsRecognizer
- 返回值 无

结束返回语音合成数据

-(void)recognizerDidStopRecieveTTSData:(NlsRecognizer *)recognizer;

- 参数
    - NlsRecognizer
- 返回值 无

设置SDK工作模式

NlsRecognizer @property(nonatomic,assign,readwrite) kNlsRecognizerMode mode;

- 说明 设置语音SDK的工作模式，若不设置，则为kMODE_RECOGNIZER

设置SDK是否监听App状态

NlsRecognizer @property(nonatomic,assign,readwrite) BOOL cancelOnAppEntersBackground;

- 说明 设置SDK是否监听App状态，缺省为NO。如果设为YES，则SDK会监听App状态，一旦切换到后台，就自动取消请求。

配置语音服务模块的基础参数

+(void)configure;

- 说明 配置语音服务模块的基础参数，请在App启动的时候调用
- 参数 无
- 返回值 无

初始化NlsRecognizer

-(id)initWithNlsRequest:(NlsRequest *)nlsRequest svcURL:(NSString *)svcURL;

- 说明 初始化NlsRecognizer，注意: 在其他地方调用，可以用dispatch_once的方式调用。
- 参数
  - nlsRequest 语音请求NlsRequest
  - svcURL 语音服务地址
- 返回值 无

语音主服务是否可用

+(BOOL)isServiceAvailable;

- 说明 语音主服务是否可用，开发者可以根据测返回值，调整UI行为
- 返回值 返回语音主服务当前是否可用。

发送语音合成请求

-(void)sendText;

- 说明 用于语音合成TTS的启动方法。网络请求在后台继续，如果有识别结果返回，则会通过delegate的didCompleteRecognizingWithResult回调方法返回文本结果，通过didCompleteTTSWithVoiceData回调方法返回语音数据。
- 返回值 无

参数和错误码说明

发送TTS语音请求的参数：

```
    {
"requests" : {
"tts_in" : {
"text" : "",// 输入的文本
"format" : "normal",
"encode_type" : "pcm",// 输入的语音格式(编码类型)，默认pcm
"sample_rate" : "16000",采样率，默认16000
"version" : "1.0"// 协议版本号
"volume": 50,
"speech_rate": 0,
```

```
"nus": 1
},
"context": {
"auth": {}
}
},
"app_key" : "",
"bstream_attached" : true,// 请求包的后面是不是还接着二进制语音流。
"version" : "4.0"// 协议版本号
}
```

返回的识别结果result是一个NlsRecognizerResult的对象：

```
   {
"status" : "1",// 服务器状态，0为失败，非零为成功
"id" : "",// 透传系统始终的uuid,服务端配置是否返回
"finish" : "1",// 0为未结束，非零为结束，识别是否已经结束
"results" : {
"tts_out" : {
"encode_type" : "pcm",//输出的语音数据编码
"id" : "",// 透传系统始终的uuid,服务端配置是否返回
"status" : "OK",// 服务器状态，0为失败，非零为成功
"speech_key_prefix" : ""// tts前缀
},
"out" : {}//保留字段
},
"bstream_attached" : false,// 应答包的后面是不是还接着二进制语音流。
"version" : "4.0"// 协议版本号
}
```

若识别发生错误，recognizer:didCompleteTTSWithVoiceData:error:的回调函数中error不为nil。相应错误码的对应表如下所示。

- 客户端错误码

字段名	错误码	含义
kERR_NO_ERROR	0	成功
kERR_GENERIC_ERROR	1	识别失败
kERR_USER_CANCELED	520	用户取消
kERR_NETWORK_ERROR	530	网络及通讯异常
kERR_SERVICE_ERROR	540	语音服务异常或被降级
kERR_VOICE_ERROR	550	录音及语音识别异常
kERR_MIC_ERROR	560	Mic无法访问或硬件异常
kERR_TOOSHORT_ERROR	570	用户点击过快

- 服务端返回结果错误码

状态	status_code	CloseFrame状态码	HTTP语义

成功	200	1000	成功处理
请求格式有误	400	4400	错误请求
需要鉴权信息	401	4401	请求要求身份验证
鉴权失败	403	4403	服务器拒绝请求
超出最大并发量	429	4429	太多请求
请求超时	408	4408	处理请求超时
处理出错	500	4500	服务器内部错误
服务不可用	503	4503	服务不可用

完整示例

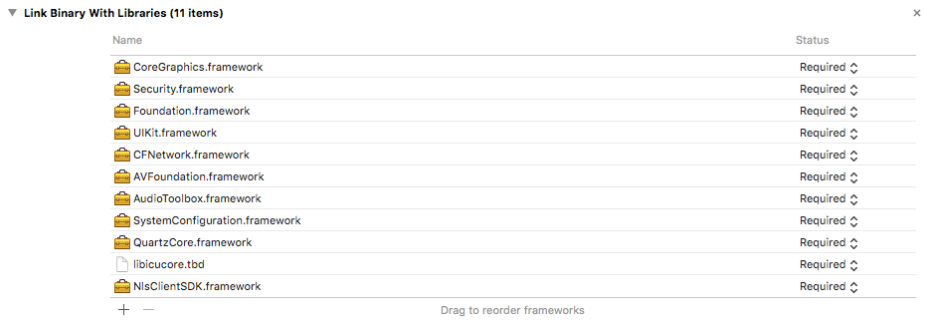
创建应用

使用Xcode创建iOS application应用工程。

添加Framework

在Xcode工程中需要引入所需要的framework，NlsClientSDK.framework。

添加方法：选中工程，点击TARGETS，在右侧的Build Phases中选择 Link Binary With Libraries，点击上图中左下角的+号，在弹出界面中依次添加所依赖的framework。




```
    $lipo -info NlsClientSDK
Architectures in the fat file: NlsClientSDK are: armv7 i386 x86_64 arm64
```

引入头文件

在需要调用SDK的文件中，添加如下头文件:

```
    #import <NlsClientSDK/NlsClientSDK.h>
```

语音服务注册

AppDelegate中注册语音服务：

```
    #import "AppDelegate.h"
#import "ViewController.h"
#import <NlsClientSDK/NlsClientSDK.h>

@interface AppDelegate ()
@end

@implementation AppDelegate
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
-
#warning configure语音服务，必须在调用语音服务前执行该方法。
[NlsRecognizer configure];
……
 }
```

实现语音识别功能

ViewController中实现语音识别方法：

```
    #import "ViewController.h"
#import <NlsClientSDK/NlsClientSDK.h>

@interface ViewController ()<NlsRecognizerDelegate>

@property(nonatomic,strong) NlsRecognizer *recognizer;

@end

@implementation ViewController

- (void)viewWillAppear:(BOOL)animated
{
[super viewWillAppear:animated];

// 检查语音主服务是否可用
if([NlsRecognizer isServiceAvailable]) {
NSLog(@"当前语音服务可用");
}
else {
NSLog(@"当前语音服务不可用");
}

// 监测语音服务状态
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(asrStatusChanged:)
name:kNlsRecognizerServiceStatusChanged object:nil];

}

#pragma mark - Actions
- (void)onTtsButtonClick:(id)sender {
```

```
// 初始化语音请求类
NlsRequest * nlsRequest = [[NlsRequest alloc] init];
#warning appkey请从 "快速开始" 帮助页面的appkey列表中获取
[nlsRequest setAppkey:@""]; // requested
[nlsRequest setBstreamAttached:NO]; // requested 不带语音数据
[nlsRequest setTtsText:self.inputTextView.text]; // requested
#warning 请修改为您在阿里云申请的数字验证串Authorize withSecret
[nlsRequest Authorize:@"" withSecret:@""]; // requested Access Key ID 和 Access Key Secret

// 初始化语音服务核心类
NlsRecognizer *r = [[NlsRecognizer alloc] initWithNlsRequest:nlsRequest svcURL:nil]; // requested
r.delegate = self;
r.cancelOnAppEntersBackground = YES;
r.enableUserCancelCallback = YES;
self.recognizer = r;

// print nlsRequest
NSString *nlsRequestJSONString = [NlsRequest getJSONStringfromNlsRequest:nlsRequest];
NSLog(@"setupTtsIn : %@",nlsRequestJSONString);

//开始语音合成
[self.recognizer sendText];
self.audioPlayer = [[NLSPlayAudio alloc] init];

}


#pragma mark - Notification Callbacks
-(void)asrStatusChanged:(NSNotification*)notify{
//处理网络变化
}

#pragma mark - RecognizerDelegate
-(void) recognizer:(NlsRecognizer *)recognizer didCompleteRecognizingWithResult:(NlsRecognizerResult*)result
error:(NSError*)error{
//处理识别结果和错误信息
}

-(void) recognizer:(NlsRecognizer *)recognizer didCompleteTTSWithVoiceData:(NSData*)voiceData
error:(NSError*)error{
//处理服务器返回的语音数据

Byte *messageByte = (Byte *)[voiceData bytes];
NSString *pre4Str = [[NSString alloc]init];
for (int i = 0; i < 4; i ++) {
pre4Str = [pre4Str stringByAppendingString: [NSString stringWithFormat:@"%d",messageByte[i]]];
}

if (![pre4Str isEqual:@"0000"]) {
NSUInteger len = [voiceData length];
Byte *voiceByte = (Byte*)malloc(len - 4);
for(int i=0;i< len - 4;i++){
voiceByte[i] = messageByte[i + 4];
}
//对语音数据进行处理
} else {
```

```
NSLog(@"voiceData stop");
}

}
-(void)recognizerDidStartRecieveTTSData:(NlsRecognizer *)recognizer {
//处理开始语音合成事件
}

-(void)recognizerDidStopRecieveTTSData:(NlsRecognizer *)recognizer {
//处理结束语音合成事件
}

@end
```

FAQ

问题1 : bitcode。

```
ld: 'xxx/NlsClientSDK.framework/NlsClientSDK(NlsRecognizer.o)' does not contain bitcode. You must rebuild it with bitcode enabled (Xcode setting ENABLE_BITCODE), obtain an updated library from the vendor, or disable bitcode for this target. for architecture arm64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
```

解决1 : 打开项目 - targets - build settings - Enable Bitcode-设置为No。

问题2 : 数加验证失败4403。

```
{
NSLocalizedDescription = "server closed connection, code:4403, reason:Unauthorized AppKey [xxx], wasClean:1";
}
```

解决2 : 检查数加验证的 ak_id 和 ak_secret 是否正确;检查 appkey 填写是否正确。

```
#warning 请修改为您在阿里云申请的数字验证串Authorize withSecret
[nlsRequest Authorize:@"ak_id" withSecret:@"ak_secret"]; // requested


#warning appkey请从 "快速开始" 帮助页面的appkey列表中获取。
[nlsRequest setAppkey:@"your_appkey"]; // requested
```

问题3 : Assertion failed。

解决3 : 检查在开始语音识别前，是否注册;检查NlsRecognizer初始化时svcURL是否为nil。

```
//Config appkey.
[NlsRecognizer configure];
```

```
NlsRecognizer *r = [[NlsRecognizer alloc] initWithNlsRequest:nlsRequest svcURL:nil];
```

问题4 : 错误码4400。

```
{
NSLocalizedDescription = "server closed connection, code:4400, reason:illegal params, operation forbidden,
wasClean:1";
}
```

解决4 : 检查appkey是否为空，填写正确的appkey。

```
#warning 请修改为您在阿里云申请的APP_KEY
[nlsRequest setAppkey:@"your_appkey"]; // requested
```

语音合成REST接口

1. 简介

语音合成REST API支持以POST方式上传UTF-8编码的合成文本。服务端支持以16bitpcm、alaw编码或16bit wav / pcm文件返回生成语音。语音合成服务的请求地址为：https://nlsapi.aliyun.com/speak

2. 请求格式

- 语音合成请求实例

```
POST http://nlsapi.aliyun.com/speak?encode_type=pcm&voice_name=xiaoyun&volume=50
Date: Tue, 21 Mar 2017 11:42:00 GMT
Content-type: text/plain
Authorization: Dataplus *****
Accept: audio/pcm, application/json
Content-Length: 36

需要被转换的文本
```

2.1 参数配置

语音合成服务在处理请求时，将在URL地址中检查如下参数。若参数未设置，将以缺省值合成语音。

名称	类型	需求	缺省值	描述
encode_type	String	选填	pcm	合成语音的编码

				格式，支持 pcm/wav/mp3/ alaw
voice_name	String	选填	xiaoyun	xiaogang - 男 ，xiaoyun - 女
volume	int	选填	50	0～100
sample_rate	int	选填	16000	抽样频率率 8000 / 16000
speech_rate	int	选填	0	语速 -500～500
pitch_rate	int	选填	0	语调 -500～500
tts_nus	int	选填	1	0 - 通过参数合成语音，1 - 拼接原始录音
background_music_id	int	选填	无	播放语音时可选背景音乐，0，1
background_music_offset	int	选填	0	背景音乐播放偏移时长，毫秒。当启用背景音乐时生效
background_music_volume	int	选填	50	背景音乐音量 ，当启用背景音乐时生效 ，0～100

2.2 语音合成文本

请以UTF-8格式编码后将需要合成的语音文本在POST body中上传，单次请求限制为300个UTF-8字符，即每个汉字、数字、字母都算一个字符。

2.3 HTTP Header

名称	类型	需求	描述
Authorization	String	必填	鉴权， 详见 **2.3.1**
Content-type	String	必填	text/plain
Accept	String	必填	固定为audio/*, application/json，请将星号替换为所需编码格式 ，pcm / wav / alaw
Date	String	必填	鉴权，HTTP 1.1协议中规定的GMT时间 ，例如：Wed, 05 Sep. 2012 23:00:00 GMT

2.3.1 Authorization Header

调用阿里巴巴智能语音交互平台的任何功能前都需经过严格的鉴权验证。在处理用户请求前，服务端会校验Authorization Header以确保用户请求在传输过程中没有被恶意篡改或替换。

```
Authorization: Dataplus access_id:signature
```

*Authorization*以固定字符串Dataplus开头，开发者需要将从阿里云申请到的access_id和经过计算的signature以:分隔并以**Base64编码**后加入Header。

2.3.1.1 signature的计算

与阿里云标准校验规范稍有不同，signature的计算需要首先对body内容进行MD5和Base64编码，然后将编码结果与 Method，Accept，Content-Type和Date 合并产生特征值，最后用阿里云取得的access_key对特征值进行HMAC-SHA1加密生成signature。这里和标准方法的区别主要在于拼接特征值时不需要urlpath。

```
// 1.对body进行MD5+BASE64加密
String bodyMd5 = MD5Base64(body);

// 2.特征值
String feature = method + "\n" + accept + "\n" + bodyMd5 + "\n" + content_type + "\n" + date;

// 2.对特征值HMAC-SHA1加密
String signature = HMACSha1(feature, access_secret);
```

2.3.1.2 计算 MD5+BASE64

```
public static String MD5Base64(String s) throws UnsupportedEncodingException {
if (s == null)
return null;
String encodeStr = "";
//string 编码必须为utf-8
byte[] utfBytes = s.getBytes("UTF-8");
MessageDigest mdTemp;
try {
mdTemp = MessageDigest.getInstance("MD5");
mdTemp.update(utfBytes);
byte[] md5Bytes = mdTemp.digest();
BASE64Encoder b64Encoder = new BASE64Encoder();
encodeStr = b64Encoder.encode(md5Bytes);
} catch (Exception e) {
throw new Error("Failed to generate MD5 : " + e.getMessage());
}
return encodeStr;
}
```

2.3.1.3 计算 HMAC-SHA1

```
public static String HMACSha1(String data, String key) {
```

```
String result;
try {
SecretKeySpec signingKey = new SecretKeySpec(key.getBytes(), "HmacSHA1");
Mac mac = Mac.getInstance("HmacSHA1");
mac.init(signingKey);
byte[] rawHmac = mac.doFinal(data.getBytes());
result = (new BASE64Encoder()).encode(rawHmac);
} catch (Exception e) {
throw new Error("Failed to generate HMAC : " + e.getMessage());
}
return result;
}
```

2.3.2 Content-Type Header

Content-Type header 规定为text / plain

2.3.3 Accept Header

Accept Header需要允许audio和json两种返回格式。当语音合成成功（包括中途失败），服务器将返回在请求中要求的语音流。请求失败将返回json字符串。

3. 语音合成返回

当语音合成请求成功时，服务端将返回16bit pcm、alaw编码或16bit wav / pcm格式的语音文件。如在合成过程中出现错误，后续语音将不再传输并且没有其他的错误返回。

如果语音合成请求失败，服务端将在response body中返回json格式文件

```
{
"request_id":"6262b55940044d95afc11d02ddcea377",
"error_code":80103,
"error_message":"authorization failed!"
}
```

4. 代码示例

4.1 请求DEMO

```
package com.alibaba.idst.nls;

import java.io.File;
import java.io.IOException;
import java.io.BufferedReader;
import java.io.OutputStream;
import java.io.FileReader;
```

```
import java.util.UUID;
import com.alibaba.idst.nls.request.TtsRequest;
import com.alibaba.idst.nls.response.HttpResponse;
import com.alibaba.idst.nls.utils.HttpUtil;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class HttpTtsDemo {
private static Logger logger = LoggerFactory.getLogger(HttpTtsDemo.class);
private String url = "http://nlsapi.aliyun.com/speak?";
private static String tts_text = "薄雾浓云愁永昼。瑞脑消金兽。佳节又重阳，玉枕纱厨，半夜凉初透。东篱把酒黄昏后。有暗香盈袖。莫道不消魂，帘卷西风，人比黄花瘦。";

public static void main(String[] args) throws IOException {

//请使用https://ak-console.aliyun.com/ 页面获取的Access 信息
//请提前开通智能语音服务(https://data.aliyun.com/product/nls)
String ak_id = args[0];
String ak_secret = args[1];

//设置TTS的参数,详细参数说明详见文档部分2.1 参数配置
HttpTtsDemo ttsDemo=new HttpTtsDemo();
TtsRequest ttsRequest = new TtsRequest();
ttsRequest.setEncodeType("wav");
ttsRequest.setVoiceName("xiaoyun");
ttsRequest.setVolume(50);
ttsRequest.setSampleRate(16000);
ttsRequest.setSpeechRate(0);
ttsRequest.setPitchRate(0);
ttsRequest.setTtsNus(1);
ttsRequest.setBackgroundMusicId(0);
ttsRequest.setBackgroundMusicOffset(0);
ttsRequest.setBackgroundMusicVolume(100);

String url = ttsDemo.url+"encode_type="+ttsRequest.getEncodeType()
+"&voice_name="+ttsRequest.getVoiceName()
+"&volume="+ttsRequest.getVolume()
+"&sample_rate="+ttsRequest.getSampleRate()
+"&speech_rate="+ttsRequest.getSpeechRate()
+"&pitch_rate="+ttsRequest.getPitchRate()
+"&tts_nus="+ttsRequest.getTtsNus()
+"&background_music_id="+ttsRequest.getBackgroundMusicId()
+"&background_music_offset="+ttsRequest.getBackgroundMusicOffset()
+"&background_music_volume="+ttsRequest.getBackgroundMusicVolume();

logger.info("TTS request is: {}",url);

String fileName = UUID.randomUUID().toString().replace("-","");
//tts demo 会在项目根目录生产语音文件
HttpResponse response = HttpUtil.sendTtsPost(tts_text,ttsRequest.getEncodeType(), fileName ,url, ak_id, ak_secret);

}
}
```

4.2 TTS参数类

```
package com.alibaba.idst.nls.request;

public class TtsRequest {

private String encode_type;
private String voice_name;
private int volume;
private int sample_rate;
private int speech_rate;
private int pitch_rate;
private int tts_nus;
private int background_music_id;
private int background_music_offset;
private int background_music_volume;

public String getEncodeType() {
return encode_type;
}

public void setEncodeType(String encode_type) {
this.encode_type = encode_type;
}

public String getVoiceName() {
return voice_name;
}

public void setVoiceName(String voice_name) {
this.voice_name = voice_name;
}

public int getVolume() {
return volume;
}

public void setVolume(int volume) {
this.volume = volume;
}

public int getSampleRate() {
return sample_rate;
}

public void setSampleRate(int sample_rate) {
this.sample_rate = sample_rate;
}

public int getSpeechRate() {
return speech_rate;
}

public void setSpeechRate(int speech_rate) {
this.speech_rate = speech_rate;
}
```

```
public int getPitchRate() {
return pitch_rate;
}

public void setPitchRate(int pitch_rate) {
this.pitch_rate = pitch_rate;
}

public int getTtsNus() {
return tts_nus;
}

public void setTtsNus(int tts_nus) {
this.tts_nus = tts_nus;
}

public int getBackgroundMusicId() {
return background_music_id;
}

public void setBackgroundMusicId(int background_music_id) {
this.background_music_id = background_music_id;
}

public int getBackgroundMusicOffset() {
return background_music_offset;
}

public void setBackgroundMusicOffset(int background_music_offset) {
this.background_music_offset = background_music_offset;
}

public int getBackgroundMusicVolume() {
return background_music_volume;
}

public void setBackgroundMusicVolume(int background_music_volume) {
this.background_music_volume = background_music_volume;
}
}
```

4.3 请求服务的HttpUtil类

```
package com.alibaba.idst.nls.utils;

import java.io.*;
import java.net.HttpURLConnection;
import java.net.URL;
import java.security.MessageDigest;
import java.text.SimpleDateFormat;
import java.util.*;
import javax.crypto.spec.SecretKeySpec;
import com.alibaba.idst.nls.response.HttpResponse;
import javax.crypto.Mac;
```

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import sun.misc.BASE64Encoder;

public class HttpUtil {
static Logger logger = LoggerFactory.getLogger(HttpUtil.class);

/*
* 计算MD5+BASE64
*/
public static String MD5Base64(byte[] s) throws UnsupportedEncodingException {
if (s == null) {
return null;
}
String encodeStr = "";
// string 编码必须为utf-8
MessageDigest mdTemp;
try {
mdTemp = MessageDigest.getInstance("MD5");
mdTemp.update(s);
byte[] md5Bytes = mdTemp.digest();
BASE64Encoder b64Encoder = new BASE64Encoder();
encodeStr = b64Encoder.encode(md5Bytes);
/*
* java 1.8以上版本支持 Encoder encoder = Base64.getEncoder(); encodeStr =
* encoder.encodeToString(md5Bytes);
*/
} catch (Exception e) {
throw new Error("Failed to generate MD5 : " + e.getMessage());
}
return encodeStr;
}

/*
* 计算 HMAC-SHA1
*/
public static String HMACSha1(String data, String key) {
String result;
try {
SecretKeySpec signingKey = new SecretKeySpec(key.getBytes(), "HmacSHA1");
Mac mac = Mac.getInstance("HmacSHA1");
mac.init(signingKey);
byte[] rawHmac = mac.doFinal(data.getBytes());
result = (new BASE64Encoder()).encode(rawHmac);
/*
* java 1.8以上版本支持 Encoder encoder = Base64.getEncoder(); result =
* encoder.encodeToString(rawHmac);
*/
} catch (Exception e) {
throw new Error("Failed to generate HMAC : " + e.getMessage());
}
return result;
}

/*
* 等同于javaScript中的 new Date().toUTCString();
```

```
*/
public static String toGMTString(Date date) {
SimpleDateFormat df = new SimpleDateFormat("E, dd MMM yyyy HH:mm:ss z", Locale.UK);
df.setTimeZone(new java.util.SimpleTimeZone(0, "GMT"));
return df.format(date);
}

/*
* 发送POST请求
*/
public static HttpResponse sendAsrPost(byte[] audioData, String audioFormat, int sampleRate, String url,
String ak_id, String ak_secret) {

BufferedReader in = null;
String result = "";
HttpResponse response = new HttpResponse();
try {
URL realUrl = new URL(url);
/*
* http header 参数
*/
String method = "POST";
String accept = "application/json";
String content_type = "audio/" + audioFormat + ";samplerate=" + sampleRate;
int length = audioData.length;
String date = toGMTString(new Date());
// 1.对body做MD5+BASE64加密
String bodyMd5 = MD5Base64(audioData);
String md52 = MD5Base64(bodyMd5.getBytes());
String stringToSign = method + "\n" + accept + "\n" + md52 + "\n" + content_type + "\n" + date;
// 2.计算 HMAC-SHA1
String signature = HMACSha1(stringToSign, ak_secret);
// 3.得到 authorization header
String authHeader = "Dataplus " + ak_id + ":" + signature;
// 打开和URL之间的连接
HttpURLConnection conn = (HttpURLConnection) realUrl.openConnection();
// 设置通用的请求属性
conn.setRequestProperty("accept", accept);
conn.setRequestProperty("content-type", content_type);
conn.setRequestProperty("date", date);
conn.setRequestProperty("Authorization", authHeader);
conn.setRequestProperty("Content-Length", String.valueOf(length));
// 发送POST请求必须设置如下两行
conn.setDoOutput(true);
conn.setDoInput(true);
// 获取URLConnection对象对应的输出流
OutputStream stream = conn.getOutputStream();
// 发送请求参数
stream.write(audioData);
// flush输出流的缓冲
stream.flush();
stream.close();
response.setStatus(conn.getResponseCode());
// 定义BufferedReader输入流来读取URL的响应
if (response.getStatus() == 200) {
in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
```

```
} else {
in = new BufferedReader(new InputStreamReader(conn.getErrorStream()));
}
String line;
while ((line = in.readLine()) != null) {
result += line;
}
if (response.getStatus() == 200) {
response.setResult(result);
response.setMessage("OK");
} else {
response.setMessage(result);
}
System.out.println("post response status code: [" + response.getStatus() + "], response massage : ["
+ response.getMessage() + "] ,result :[" + response.getResult() + "]");
} catch (Exception e) {
System.out.println("发送 POST 请求出现异常！" + e);
e.printStackTrace();
}
// 使用finally块来关闭输出流、输入流
finally {
try {
if (in != null) {
in.close();
}
} catch (IOException ex) {
ex.printStackTrace();
}
}
return response;
}

/*
* 发送POST请求
*/
public static HttpResponse sendTtsPost(String textData, String audioType, String audioName, String url,
String ak_id, String ak_secret) {
BufferedReader in = null;
FileOutputStream fileOutputStream = null;
String result = "";
HttpResponse response = new HttpResponse();
try {
URL realUrl = new URL(url);
/*
* http header 参数
*/
String method = "POST";
String content_type = "text/plain";
String accept = "audio/" + audioType + ",application/json";
int length = textData.length();
String date = toGMTString(new Date());
// 1.对body做MD5+BASE64加密
String bodyMd5 = MD5Base64(textData.getBytes("UTF-8"));
String stringToSign = method + "\n" + accept + "\n" + bodyMd5 + "\n" + content_type + "\n" + date;
// 2.计算 HMAC-SHA1
String signature = HMACSha1(stringToSign, ak_secret);
```

```
// 3.得到 authorization header
String authHeader = "Dataplus " + ak_id + ":" + signature;
// 打开和URL之间的连接
HttpURLConnection conn = (HttpURLConnection) realUrl.openConnection();
// 设置通用的请求属性
conn.setRequestProperty("accept", accept);
conn.setRequestProperty("content-type", content_type);
conn.setRequestProperty("date", date);
conn.setRequestProperty("Authorization", authHeader);
conn.setRequestProperty("Content-Length", String.valueOf(length));
// 发送POST请求必须设置如下两行
conn.setDoOutput(true);
conn.setDoInput(true);
// 获取URLConnection对象对应的输出流
OutputStream stream = conn.getOutputStream();
// 发送请求参数
stream.write(textData.getBytes("UTF-8"));
// flush输出流的缓冲
stream.flush();
stream.close();
response.setStatus(conn.getResponseCode());
// 定义BufferedReader输入流来读取URL的响应
InputStream is = null;
String line = null;
if (response.getStatus() == 200) {
is = conn.getInputStream();
} else {
in = new BufferedReader(new InputStreamReader(conn.getErrorStream()));
while ((line = in.readLine()) != null) {
result += line;
}
}
File ttsFile = new File(audioName + "." + audioType);
fileOutputStream = new FileOutputStream(ttsFile);
byte[] b = new byte[1024];
int len = 0;
while (is != null && (len = is.read(b)) != -1) { // 先读到内存
fileOutputStream.write(b, 0, len);
}
if (response.getStatus() == 200) {
response.setResult(result);
response.setMessage("OK");
System.out.println("post response status code: [" + response.getStatus()
+ "], generate tts audio file :" + audioName + "." + audioType);
} else {
response.setMessage(result);
System.out.println("post response status code: [" + response.getStatus() + "], response massage : ["
+ response.getMessage() + "]");
}
} catch (Exception e) {
System.out.println("发送 POST 请求出现异常！" + e);
e.printStackTrace();
}
// 使用finally块来关闭输出流、输入流
finally {
try {
```

```
if (fileOutputStream != null) {
fileOutputStream.close();
}
if (in != null) {
in.close();
}
} catch (IOException ex) {
ex.printStackTrace();
}
}
return response;
}
}
```

4.4 请求结果类

```
package com.alibaba.idst.nls.response;

public class HttpResponse {
private int status;
private String result;
private String message;

public int getStatus() {
return status;
}

public void setStatus(int status) {
this.status = status;
}

public String getResult() {
return result;
}

public void setResult(String result) {
this.result = result;
}

public String getMessage() {
return message;
}

public void setMessage(String message) {
this.message = message;
}
}
```

长文本合成Demo

TTS 长文本合成Demo

长文本合成Demo是为了满足用户有超过300字以上的文本，并需要合成一条语音结果的实现。本Demo基于Java SDK 实现，用户需自定配置Java SDK的调用环境。

SDK 调用：

```
package com.alibaba.idst.nls.demo;

import com.alibaba.idst.nls.NlsClient;
import com.alibaba.idst.nls.NlsFuture;
import com.alibaba.idst.nls.event.NlsEvent;
import com.alibaba.idst.nls.event.NlsListener;
import com.alibaba.idst.nls.protocol.NlsRequest;
import com.alibaba.idst.nls.protocol.NlsRequestASR;
import com.alibaba.idst.nls.protocol.NlsRequestProto;
import com.alibaba.idst.nls.protocol.NlsResponse;
import com.alibaba.idst.nls.utils.PcmToWav;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.Arrays;
import java.util.UUID;

/**
* Created by songsong.sss on 16/12/12.
*/
public class LongTtsDemo implements NlsListener {
static Logger logger = LoggerFactory.getLogger(LongTtsDemo.class);

private NlsClient client = new NlsClient();
public String appKey = null;
public String auth_Id = null;
public String auth_Secret = null;
public String tts_text ;
private String fileName = UUID.randomUUID().toString();


public LongTtsDemo() {
}

public void shutDown() {
logger.info("close NLS client");
client.close();
logger.info("demo done");
}

public void start() {
logger.info("init Nls client...");
```

```
client.init();

tts_text = "百草堂与三味书屋 鲁迅 \n" +
"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。\n" +
"不必说碧绿的菜畦，光滑的石井栏，高大的皂荚树，紫红的桑葚；也不必说鸣蝉在树叶里长吟，肥胖的黄蜂伏在菜花上，轻捷的叫天子(云雀)忽然从草间直窜向云霄里去了。\n"+
"单是周围的短短的泥墙根一带，就有无限趣味。油蛉在这里低唱，蟋蟀们在这里弹琴。翻开断砖来，有时会遇见蜈蚣；还有斑蝥，倘若用手指按住它的脊梁，便会啪的一声，\n"+
"从后窍喷出一阵烟雾。何首乌藤和木莲藤缠络着，木莲有莲房一般的果实，何首乌有臃肿的根。有人说，何首乌根是有像人形的，吃了便可以成仙，我于是常常拔它起来，牵连不断地拔起来，\n" +
"也曾因此弄坏了泥墙，却从来没有见过有一块根像人样。如果不怕刺，还可以摘到覆盆子，像小珊瑚珠攒成的小球，又酸又甜，色味都比桑葚要好得远。";

}

public void sayIt() throws Exception {

int ttsTextLength = tts_text.length();
String[] longTexts;
int i = 0;
boolean isHead = false; //标识是否是第一个头文件
String tts_part_text;
File file = new File(fileName+".pcm");
if (!file.exists()) {
try {
file.createNewFile();
} catch (Exception e) {
e.printStackTrace();
}

}
FileOutputStream outputStream = new FileOutputStream(file, true);
longTexts = processLongText(tts_text);
//处理文本,文本长度以50为限,截取为多个文件.
while (ttsTextLength > 0) {
tts_part_text = "";
if (ttsTextLength > 50) {
if (i == 0) {
isHead = true;
} else {
isHead = false;
}
for (; i < longTexts.length; i++) {
tts_part_text = tts_part_text + longTexts[i];
if (i < longTexts.length - 1 && tts_part_text.length() + longTexts[i + 1].length() >= 50) {
i = i + 1;
break;

}
}
} else {
if (i == 0) {
isHead = true;
}
for (; i < longTexts.length; i++) {
```

```
tts_part_text = tts_part_text + longTexts[i];
}
}
NlsRequest req = new NlsRequest();
req.setApp_key("nls-service");
req.setTts_req(tts_part_text, "16000");
req.setTtsEncodeType("wav");
req.setTtsVoice("xiaoyun");//男声:xiaogang
req.setTtsVolume(50);
req.setTtsBackgroundMusic(1, 0);
req.authorize(auth_Id, auth_Secret);

NlsFuture future = client.createNlsFuture(req, this);
int total_len = 0;
byte[] data;
while ((data = future.read()) != null) {
if (data.length == 8044 ) {
// 去掉wav头,同时将多条wav转成一条pcm
logger.debug("data length:{} , and head is:{}", (data.length - 44), isHead ? "true" : "false");
outputStream.write(data, 44, data.length - 44);
} else {
outputStream.write(data, 0, data.length);
}
total_len += data.length;
}

logger.info("tts audio file size is :" + total_len);
future.await(10000);

ttsTextLength = ttsTextLength - tts_part_text.length();
}
outputStream.close();
//将pcm转为wav,可以直接播放. 格式为:16kHz采样率,16bit,单声道
PcmToWav.copyWaveFile(fileName+".pcm",fileName+".wav");
logger.debug("close the wav file!");
}


@Override
public void onMessageReceived(NlsEvent e) {
NlsResponse response = e.getResponse();
String result = "";
if (response.getDs_ret() != null) {
result = "get ds result: " + response.getDs_ret();
}
if (response.getAsr_ret() != null) {
result += "\nget asr result: " + response.getAsr_ret();
}
if (response.getTts_ret() != null) {
result += "\nget tts result: " + response.getTts_ret();
}
if (response.getGds_ret() != null) {
result += "\nget gds result: " + response.getGds_ret();
}
if (!result.isEmpty()) {
logger.info(result);
```

```
} else if (response.jsonResults != null) {
logger.info(response.jsonResults.toString());
} else {
logger.info("get an acknowledge package from server.");
}
}

@Override
public void onOperationFailed(NlsEvent e) {
logger.error("Error message is: {}, Error code is: {}", e.getErrorMessage(),
Integer.valueOf(e.getResponse().getStatus_code()));
}


//切分长文本
public static String[] processLongText(String text) {
text = text.replaceAll("、", "、|");
text = text.replaceAll(" , ", " , |");
text = text.replaceAll("。", "。|");
text = text.replaceAll(" ; ", " ; |");
text = text.replaceAll(" ? ", " ? |");
text = text.replaceAll(" ! ", " ! |");
text = text.replaceAll(",", ",|");
text = text.replaceAll(";", ";|");
text = text.replaceAll("\\?", "?|");
text = text.replaceAll("!", "!|");
String[] texts = text.split("\\|");
return texts;

}


@Override
public void onChannelClosed(NlsEvent e) {
logger.info("on websocket closed.");
}

/**
* @param args
*/
public static void main(String[] args) throws Exception {
LongTtsDemo lun = new LongTtsDemo();

if (args.length < 4) {
logger.info("NlsDemo <app-key> <Id> <Secret>");
System.exit(-1);
}

lun.appKey = args[0];
lun.auth_Id = args[1];
lun.auth_Secret = args[2];


lun.start();
lun.sayIt();
lun.shutDown();
```

```
}

}
```

pcm 转 wav 工具类：

```
package com.alibaba.idst.nls.utils;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

/**
* Created by songsong.sss on 2016/12/12.
*/
public class PcmToWav {
private static int frequency = 16000;
private static final int RECORDER_BPP = 16;
private static int recBufSize = 640;
public static void copyWaveFile(String inFilename,String outFilename){
FileInputStream in = null;
FileOutputStream out = null;
long totalAudioLen = 0;
long totalDataLen = totalAudioLen + 36;
long longSampleRate = frequency;
int channels = 1;
long byteRate = RECORDER_BPP * frequency * channels/8;

byte[] data = new byte[recBufSize];

try {
in = new FileInputStream(inFilename);
out = new FileOutputStream(outFilename);
totalAudioLen = in.getChannel().size();
totalDataLen = totalAudioLen + 36;

//AppLog.logString("File size: " + totalDataLen);

WriteWaveFileHeader(out, totalAudioLen, totalDataLen,
longSampleRate, channels, byteRate);

while(in.read(data) != -1){
out.write(data);
}

in.close();
out.close();
} catch (FileNotFoundException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}
}
```

```
private static void WriteWaveFileHeader(
FileOutputStream out, long totalAudioLen,
long totalDataLen, long longSampleRate, int channels,
long byteRate) throws IOException {

byte[] header = new byte[44];

header[0] = 'R'; // RIFF/WAVE header
header[1] = 'I';
header[2] = 'F';
header[3] = 'F';
header[4] = (byte) (totalDataLen & 0xff);
header[5] = (byte) ((totalDataLen >> 8) & 0xff);
header[6] = (byte) ((totalDataLen >> 16) & 0xff);
header[7] = (byte) ((totalDataLen >> 24) & 0xff);
header[8] = 'W';
header[9] = 'A';
header[10] = 'V';
header[11] = 'E';
header[12] = 'f'; // 'fmt ' chunk
header[13] = 'm';
header[14] = 't';
header[15] = ' ';
header[16] = 16; // 4 bytes: size of 'fmt ' chunk
header[17] = 0;
header[18] = 0;
header[19] = 0;
header[20] = 1; // format = 1
header[21] = 0;
header[22] = (byte) channels;
header[23] = 0;
header[24] = (byte) (longSampleRate & 0xff);
header[25] = (byte) ((longSampleRate >> 8) & 0xff);
header[26] = (byte) ((longSampleRate >> 16) & 0xff);
header[27] = (byte) ((longSampleRate >> 24) & 0xff);
header[28] = (byte) (byteRate & 0xff);
header[29] = (byte) ((byteRate >> 8) & 0xff);
header[30] = (byte) ((byteRate >> 16) & 0xff);
header[31] = (byte) ((byteRate >> 24) & 0xff);
header[32] = (byte) (1 * 16 / 8); // block align
header[33] = 0;
header[34] = RECORDER_BPP; // bits per sample
header[35] = 0;
header[36] = 'd';
header[37] = 'a';
header[38] = 't';
header[39] = 'a';
header[40] = (byte) (totalAudioLen & 0xff);
header[41] = (byte) ((totalAudioLen >> 8) & 0xff);
header[42] = (byte) ((totalAudioLen >> 16) & 0xff);
header[43] = (byte) ((totalAudioLen >> 24) & 0xff);
out.write(header, 0, 44);
}
}
```