语音降噪

TBD.

一、产品优势

TBD

二、产品功能

TBD

三、应用场景

TBD

三、API文档

1、接口说明

- 集成同声传译时，需按照以下要求:

内容	说明
传输方式	http[s] (为提高安全性，强烈推荐https)
请求地址前缀	http[s]: //voiceid.abcpen.com/denoise*注：服务器IP不固定，为保证您的接口稳定，请勿通过指定IP的方式调用接口，使用域名方式调用*
接口鉴权	签名机制，详情请参照下方[鉴权说明]
字符编码	UTF-8
响应格式	统一采用JSON格式
开发语言	任意，只要可以向笔声云服务发起HTTP请求的均可

2. 鉴权说明

在调用业务接口时，请求方需要对请求进行签名，服务端通过签名来校验请求的合法性。
鉴权根据application_id, application_secret, timestamp 和signature这几个参数做组合计算，其中application_id, application_secret请在笔声开放平台自主申请。鉴权如下:

- **注意，目前线上版本鉴权尚没有加入；加入后的验证规则和下面的实现代码没有差别。**

(1). Python示例代码

```python
def generate_signature(application_key: str, application_secret: str) -> str:
    timestamp: str = str(int(time.time()))
    message = f"{application_key}{timestamp}"
    signature = hmac.new(application_secret.encode("utf-8"), message.encode("utf-8"), hashlib.sha256).hexdigest()
    return signature
```

(2) . Java示例代码

```java
import java.nio.charset.StandardCharsets;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.time.Instant;

public class SignatureGenerator {
    public static String generateSignature(String applicationKey, String applicationSecret) {
        String timestamp = String.valueOf(Instant.now().getEpochSecond());
        String message = applicationKey + timestamp;
        try {
            Mac sha256Hmac = Mac.getInstance("HmacSHA256");
            SecretKeySpec secretKey = new SecretKeySpec(applicationSecret.getBytes(StandardCharsets.UTF_8), "HmacSHA256");
            sha256Hmac.init(secretKey);
            byte[] hmacDigest = sha256Hmac.doFinal(message.getBytes(StandardCharsets.UTF_8));
            return bytesToHex(hmacDigest);
        } catch (NoSuchAlgorithmException | InvalidKeyException e) {
            e.printStackTrace();
            return null;
        }
    }

    private static String bytesToHex(byte[] bytes) {
        StringBuilder hexString = new StringBuilder();
        for (byte b : bytes) {
            String hex = Integer.toHexString(0xff & b);
            if (hex.length() == 1) {
                hexString.append('0');
            }
            hexString.append(hex);
        }
        return hexString.toString();
    }
```

```
}
```

(3). Kotlin示例代码

```kotlin
import java.security.MessageDigest
import javax.crypto.Mac
import javax.crypto.spec.SecretKeySpec
import java.time.Instant

fun generateSignature(applicationKey: String, applicationSecret: String): String {
    val timestamp = Instant.now().epochSecond.toString()
    val message = "$applicationKey$timestamp"
    val hmac = Mac.getInstance("HmacSHA256")
    hmac.init(SecretKeySpec(applicationSecret.toByteArray(Charsets.UTF_8),
"HmacSHA256"))
    val hmacDigest = hmac.doFinal(message.toByteArray(Charsets.UTF_8))
    return bytesToHex(hmacDigest)
}

private fun bytesToHex(bytes: ByteArray): String {
    val hexChars = CharArray(bytes.size * 2)
    for (i in bytes.indices) {
        val v = bytes[i].toInt() and 0xFF
        hexChars[i * 2] = hexArray[v.ushr(4)]
        hexChars[i * 2 + 1] = hexArray[v and 0x0F]
    }
    return String(hexChars)
}

private val hexArray = "0123456789abcdef".toCharArray()
```

(4). nodejs示例代码

```javascript
const crypto = require('crypto');

function generateSignature(applicationKey, applicationSecret) {
  const timestamp = Math.floor(Date.now() / 1000).toString();
  const message = applicationKey + timestamp;
  const hmac = crypto.createHmac('sha256', applicationSecret);
  hmac.update(message);
  const signature = hmac.digest('hex');
  return signature;
}

// 示例使用
const applicationKey = 'your_application_key';
```

```
const applicationSecret = 'your_application_secret';
const signature = generateSignature(applicationKey, applicationSecret);
console.log(signature);
```

(5). go示例代码

```
package main

import (
    "crypto/hmac"
    "crypto/sha256"
    "encoding/hex"
    "fmt"
    "time"
)

func generateSignature(applicationKey string, applicationSecret string) string {
    timestamp := fmt.Sprintf("%d", time.Now().Unix())
    message := applicationKey + timestamp
    hmacKey := []byte(applicationSecret)
    hmacData := []byte(message)
    hmacSha256 := hmac.New(sha256.New, hmacKey)
    hmacSha256.Write(hmacData)
    signature := hex.EncodeToString(hmacSha256.Sum(nil))
    return signature
}

func main() {
    applicationKey := "your_application_key"
    applicationSecret := "your_application_secret"
    signature := generateSignature(applicationKey, applicationSecret)
    fmt.Println(signature)
}
```

(6) Rust示例代码

```
use hmac::{Hmac, Mac, NewMac};
use sha2::Sha256;
use std::time::{SystemTime, UNIX_EPOCH};

fn generate_signature(application_key: &str, application_secret: &str) -> String {
    let timestamp = SystemTime::now()
        .duration_since(UNIX_EPOCH)
        .unwrap()
        .as_secs()
        .to_string();
    let message = format!("{}{}", application_key, timestamp);
    let mut hmac =
```

```
        Hmac::<Sha256>::new_varkey(application_secret.as_bytes()).expect("HMAC
initialization failed");
    hmac.update(message.as_bytes());
    let signature = hex::encode(hmac.finalize().into_bytes());
    signature
}

fn main() {
    let application_key = "your_application_key";
    let application_secret = "your_application_secret";
    let signature = generate_signature(application_key, application_secret);
    println!("{}", signature);
}
```

(7) vue示例代码

```
<template>
  <div>
    <button @click="generateSignature">Generate Signature</button>
    <p>Timestamp: {{ timestamp }}</p>
    <p>Signature: {{ signature }}</p>
  </div>
</template>

<script>
import crypto from 'crypto';

export default {
  name: 'SignatureGenerator',
  data() {
    return {
      applicationKey: 'test1',
      applicationSecret: '2258ACC4-199B-4DCB-B6F3-C2485C63E85A',
      timestamp: null,
      signature: null,
    };
  },
  methods: {
    generateSignature() {
      const timestamp = Math.floor(Date.now() / 1000).toString();
      const message = this.applicationKey + timestamp;
      const hmac = crypto.createHmac('sha256', this.applicationSecret);
      hmac.update(message);
      const signature = hmac.digest('hex');
      this.timestamp = timestamp;
      this.signature = signature;
    },
  },
};
</script>
```

2、鉴权访问

根据上述鉴权说明，生成X-App-Key， X-App-Signature和X-Timestamp这三个参数；将这三个参数放入http(s)请求体头部（header），向服务端发起请求。具体参考后面的示例代码。

注意：

- 下面的API路径中，目前去掉了"/v1"路径；目前去掉了appid，appsecret验证。正式版本这两个都会加上。
- 所有输入参数中，目前有app_id参数，正式版本中都会被appid, appsecret的联合验证方式所替代。
- 别的不会发生变化。

3、接口访问

（1）发起降噪请求

- 客户将降噪的文件上传到云端，获得一个url链接（不须要笔声云自己的云端存储）
- POST表单方式提交
- API路径

/denoise/denoiseRequest

- 输入参数定义
  - 常规参数，属于鉴权信息，生成X-App-Key， X-App-Signature和X-Timestamp这三个参数
  - 具体参数定义

参数	类型	是否必须	默认值	备注
task_id	string	是	无	开发者app_id, 目前任意填入字母数字串；生产环境会被appid/appsecret替换。
audio_url	string	是	无	待降噪的语音url， https/http形式的url链接

- 返回参数定义

参数	类型	备注
data	string	子参数：1. task_id: 待降噪的任务id，后续用接口/denoise/denoiseReply获取处理结果；2. input_audio_url：客户传入的待降噪的url链接；3. callback_url, 客户的回调地址；注意回调地址和通过/denoise/denoiseReply只能二选一。

参数	类型	备注
code	string	状态码, 如“0”， 具体参考[错误码]
msg	string	状态码对应字符串，如"success"

如:


```
{"code":"0","msg":"success","data":{"audio_task":{"task_id":"6734da2d-14e3-46b4-ad40-8581efa70c24","input_audio_url":"https://zmeet-1258547067.cos.ap-shanghai.myqcloud.com/voiceid/wuhan/20230316/9d6d081e-3047-4480-b622-c9ccddcb2092.wav"}}}
```


(2) 获取降噪的结果

- 根据客户输入的任务id，获得返回处理的结果。
- POST表单方式提交
- API路径

/denoise/denoiseReply

- 输入参数定义
  - 常规参数，属于鉴权信息，生成X-App-Key， X-App-Signature和X-Timestamp这三个参数
- 具体参数定义

参数	类型	是否必须	默认值	备注
app_id	string	是	无	开发者app_id, 目前任意填入字母数字串；生产环境会被appid/appsecret替换。
audio_task_id	string	是	无	语音的task id，通过/denoise/denoiseRequest 接口返回。

- 返回参数定义

参数	类型	备注
data	string	子参数有：1. task_id, 客户输入的任务id， 2. input_audio_url, 客户输入的原始噪声url， 3. clean_audio_url, AI引擎处理的降噪后的干净的语音url; 其中status具体定义如下： status = 1, 处理完毕. status = 2, 正在处理中； status = 3, 处理失败

参数	类型	备注
code	string	状态码, 如“0”， 具体参考[错误码]
msg	string	状态码对应字符串，如"success"

- 返回结果实例


```
{"code":"0","msg":"success","data":{"audio_task":"{\n  \"status\": 1,\n  \"task_id\": \"96944ecb-2c76-4ec6-8f69-c3eedd885172\",\n  \"input_audio_url\": \"https://zmeet-1258547067.cos.ap-shanghai.myqcloud.com/voiceid/wuhan/20230316/9d6d081e-3047-4480-b622-c9ccddcb2092.wav\",\n  \"clean_audio_url\": \"https://zos.abcpen.com/denoise/abcpen/20230427/c592810a-d69e-4511-a16a-0ed877319f12.wav\"\n}"}}
```


###

4、接口调用实例

（1）、Python实例

```
import requests
import uuid
import random
import os
from pathlib import Path
from time import perf_counter
import json
import time


def voice_denoise_request():
    """发起降噪请求"""
    app_id = "abcpen0"
    values = {'audio_url': "https://zmeet-1258547067.cos.ap-shanghai.myqcloud.com/voiceid/wuhan/20230316/9d6d081e-3047-4480-b622-c9ccddcb2092.wav",
              "app_id": app_id}
    url = "https://voiceid.abcpen.com/denoise/denoiseRequest"
    r = requests.post(url,  data=values)
    print(f"audio denoise request: {r.text}\n")
    return r.text

def voice_denoise_reply(task_id:str):
    """获得降噪处理的结果"""
    print(f"voice_denoise_reply task_id==============================>>>: {task_id}, type: {type(task_id)}\n")
    app_id = "abcpen0"
```

```
    values = {'audio_task_id': task_id,
              "app_id": app_id}
    url = "https://voiceid.abcpen.com/denoise/denoiseReply"
    r = requests.post(url,  data=values)
    print(f"audio denoise reply: {r.text}\n")

def denoise():
    try:

        txt = voice_denoise_request()
        task_id = json.loads(txt)
        task_id = task_id["data"]["audio_task"]["task_id"]
        time.sleep(10)
        voice_denoise_reply(task_id)
    except Exception as err:
        print(f"meet exception {err}")

if __name__ == "__main__":
    #voiceid()
    denoise()
```

(2) 、Java实例

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.HashMap;
import java.util.Map;

public class VoiceDenoiser {
    public static void main(String[] args) {
    voiceDenoise();
}

public static String voiceDenoiseRequest() throws IOException {
    String appId = "abcpen0";
    String audioUrl = "https://zmeet-1258547067.cos.ap-
shanghai.myqcloud.com/voiceid/wuhan/20230316/9d6d081e-3047-4480-b622-
c9ccddcb2092.wav";

    URL url = new URL("https://voiceid.abcpen.com/denoise/denoiseRequest");
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setRequestMethod("POST");
    conn.setRequestProperty("Content-Type", "application/json");
    conn.setRequestProperty("Accept", "application/json");

    Map<String, String> values = new HashMap<>();
```

```
    values.put("audio_url", audioUrl);
    values.put("app_id", appId);

    String jsonInputString = new Gson().toJson(values);

    conn.setDoOutput(true);
    try (var os = conn.getOutputStream()) {
        byte[] input = jsonInputString.getBytes("utf-8");
        os.write(input, 0, input.length);
    }

    BufferedReader in = new BufferedReader(
            new InputStreamReader(conn.getInputStream()));
    String inputLine;
    StringBuffer response = new StringBuffer();
    while ((inputLine = in.readLine()) != null) {
        response.append(inputLine);
    }
    in.close();

    System.out.println("audio denoise request: " + response.toString());
    return response.toString();
}

public static void voiceDenoiseReply(String taskId) throws IOException {
    String appId = "abcpen0";

    URL url = new URL("https://voiceid.abcpen.com/denoise/denoiseReply");
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setRequestMethod("POST");
    conn.setRequestProperty("Content-Type", "application/json");
    conn.setRequestProperty("Accept", "application/json");

    Map<String, String> values = new HashMap<>();
    values.put("audio_task_id", taskId);
    values.put("app_id", appId);

    String jsonInputString = new Gson().toJson(values);

    conn.setDoOutput(true);
    try (var os = conn.getOutputStream()) {
        byte[] input = jsonInputString.getBytes("utf-8");
        os.write(input, 0, input.length);
    }

    BufferedReader in = new BufferedReader(
            new InputStreamReader(conn.getInputStream()));
    String inputLine;
    StringBuffer response = new StringBuffer();
    while ((inputLine = in.readLine()) != null) {
        response.append(inputLine);
    }
```

```
    in.close();

    System.out.println("audio denoise reply: " + response.toString());
}

public static void voiceDenoise() {
    try {
        String txt = voiceDenoiseRequest();
        String taskId = (String) new
JSONObject(txt).getJSONObject("data").getJSONObject("audio_task").get("task_id");
        Thread.sleep(10000);
        voiceDenoiseReply(taskId);
    } catch (Exception e) {
        System.out.println("meet exception " + e.getMessage());
    }
}
```

(3) 、Kotlin实例

```
import com.google.gson.Gson
import org.json.JSONObject
import java.io.BufferedReader
import java.io.InputStreamReader
import java.net.HttpURLConnection
import java.net.URL

fun main() {
voiceDenoise()
}

fun voiceDenoiseRequest(): String {
val appId = "abcpen0"
val audioUrl = "https://zmeet-1258547067.cos.ap-
shanghai.myqcloud.com/voiceid/wuhan/20230316/9d6d081e-3047-4480-b622-
c9ccddcb2092.wav"
val url = URL("https://voiceid.abcpen.com/denoise/denoiseRequest")
val conn = url.openConnection() as HttpURLConnection
conn.requestMethod = "POST"
conn.setRequestProperty("Content-Type", "application/json")
conn.setRequestProperty("Accept", "application/json")

val values = HashMap<String, String>()
values["audio_url"] = audioUrl
values["app_id"] = appId

val jsonInputString = Gson().toJson(values)

conn.doOutput = true
val os = conn.outputStream
val input: ByteArray = jsonInputString.toByteArray(charset("utf-8"))
```

```
os.write(input, 0, input.size)

val inReader = BufferedReader(InputStreamReader(conn.inputStream))
var inputLine: String?
val response = StringBuffer()
while (inReader.readLine().also { inputLine = it } != null) {
    response.append(inputLine)
}
inReader.close()

println("audio denoise request: ${response.toString()}")
return response.toString()
}

fun voiceDenoiseReply(taskId: String) {
val appId = "abcpen0"
val url = URL("https://voiceid.abcpen.com/denoise/denoiseReply")
val conn = url.openConnection() as HttpURLConnection
conn.requestMethod = "POST"
conn.setRequestProperty("Content-Type", "application/json")
conn.setRequestProperty("Accept", "application/json")

val values = HashMap<String, String>()
values["audio_task_id"] = taskId
values["app_id"] = appId

val jsonInputString = Gson().toJson(values)

conn.doOutput = true
val os = conn.outputStream
val input: ByteArray = jsonInputString.toByteArray(charset("utf-8"))
os.write(input, 0, input.size)

val inReader = BufferedReader(InputStreamReader(conn.inputStream))
var inputLine: String?
val response = StringBuffer()
while (inReader.readLine().also { inputLine = it } != null) {
    response.append(inputLine)
}
inReader.close()

println("audio denoise reply: ${response.toString()}")
}

fun voiceDenoise() {
try {
val txt = voiceDenoiseRequest()
val taskId =
JSONObject(txt).getJSONObject("data").getJSONObject("audio_task").get("task_id") as
String
Thread.sleep(10000)
voiceDenoiseReply(taskId)
```

```
} catch (e: Exception) {
println("meet exception ${e.message}")
}
}
```

5. 错误码

错误码	描述	说明	处理方式
0	success	成功	
-1	in progress	识别中	请继续重试
-2	audio encode error	音频编码错误	请编码成正确的格式，再提交请求
10105	illegal access	没有权限	检查apiKey，ip，ts等授权参数是否正确
10106	invalid parameter	无效参数	上传必要的参数， 检查参数格式以及编码
10107	illegal parameter	非法参数值	检查参数值是否超过范围或不符合要求
10110	no license	无授权许可	检查参数值是否超过范围或不符合要求
10700	engine error	引擎错误	提供接口返回值，向服务提供商反馈
16003	basic component error	基础组件异常	重试或向服务提供商反馈
10800	over max connect limit	超过授权的连接数	确认连接数是否超过授权的连接数

四、价格套餐

待定