目录

1 功能介绍

- 支持单轨/双轨的 WAV 格式、MP3 格式的录音文件识别；
- 支持两种调用方式：轮询方式和回调方式；
- 支持自学习平台和热词；
- 支持 8000Hz、16000Hz 的采样率；
- 支持汉语普通话、方言、欧美英语等多种模型识别；
- 支持设置有效时间段信息，用来排除一些不必要时间段的识别；
- 录音文件地址链接支持 HTTP/HTTPS(推荐)、FTP 协议，或者存放在录音文件识别所在服务器的本地路径（注意 docker 环境的目录映射）。

2 调用限制

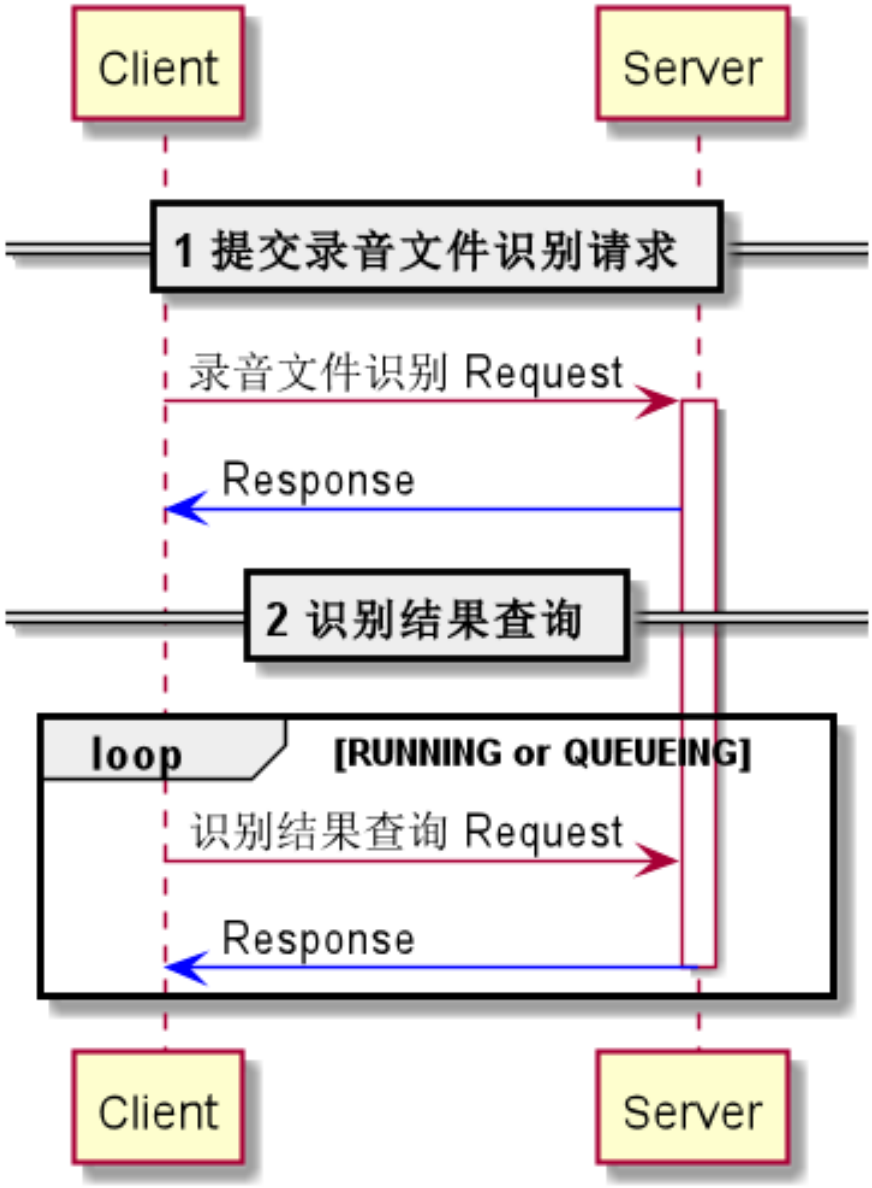
- 文件大小需控制在 512MB 以下；
- 提交录音文件识别请求后，在 24 小时内完成识别，识别结果在服务端可保存 72 小时；
- 录音文件访问权限需要保证能被服务端访问和下载。

3 使用步骤

轮询方式：

1. 了解您的录音文件格式和采样率，是否与选取的模型匹配。
2. 用户把录音文件存放在某服务器上或者本地目录下，保证录音文件识别服务能访问和下载。
3. 客户端提交录音文件识别请求，正常服务端会返回该请求任务的 ID，用以查询识别结果。
4. 客户端进行识别结果的查询，通过步骤 3 获取的请求任务 ID 查询录音文件识别的结果，目前识别的结果在服务端可保存 72 小时。

客户端与服务端的交互流程如图所示：



回调方式：

1. 了解您的录音文件格式和采样率，是否与选取的模型匹配。
2. 用户把录音文件存放在服务器上，保证录音文件识别服务能访问和下载。
3. 服务端将识别结果发送到回调 URL。

4 接口调用方式

录音文件识别服务是以 RESTful API 方式提供录音文件识别接口。需要识别的录音文件必须放在某服务器上，可以被录音文件识别服务访问到。
录音文件识别 RESTful API 包括两部分：POST 方式的“录音文件识别请求接口”，GET 方式的“录音文件识别结果查询接口”。

4.1 录音文件识别请求接口

- 当采用轮询方式时，提交录音文件识别请求，获取任务 ID，供后续轮询使用。
- 当采用回调方式时，提交录音文件识别请求和回调 URL，请求完成后会把识别结果发送到回调地址。

录音文件识别请求的 HTTP 报文实例：

```
POST /stream/v1/filetrans HTTP/1.1
Content-Type: application/json; charset=utf-8
Content-Length: 166
Host: gateway 所在 IP:8101

{"appkey":"default","file_link":"https://aliyun-nls.oss-cn-
hangzhou.aliyuncs.com/asr/fileASR/examples/nls-sample-16k.wav","token":"default"}
```

URL

URL 中指定了请求路径**/stream/v1/filetrans**。

协议	URL	方法
HTTP/1.1	http://gateway 所在 IP:8101/stream/v1/filetrans	POST

HTTP 请求头部

名称	类型	是否必需	描述
Content-Type	String	是	必须为“application/json”，表明 HTTP Body 的内容为 JSON 格式字符串
Content-Length	long	否	HTTP Body 中内容的长度

HTTP 请求体

HTTP 请求体传入的是输入参数组成的 JSON 格式的字符串，因此在 HTTP 请求头部的 Content-Type 必须设置为"application/json"。输入参数如下表所示：

属性	类型	是否必需	说明
appkey	String	是	业务方或者业务场景的标记，专有云为 default
debug	String	否	是否输出 debug 日志，存放在 data 目录下的 processor.log 中，值为 true 或者 false
token	String	是	服务鉴权 Token，专有云为 default
file_link	String	是	存放录音文件的地址链接，支持 HTTP/HTTPS(推荐)、FTP 协议，或者存放在录音文件识别所在服务器的本地路径，格式为 file:/录音文件绝对路径。（使用 docker 部署时，请将录音文件放在软件包的 data/nls-filetrans 目录下，并在使用时，将 file_link 设置为：file:/home/admin/nls-filetrans/disk/${要识别的录音文件名} ）

enable_callback	Boolean	否	是否启用回调功能，默认值为 false
callback_url	String	否	回调用户服务的地址，enable_callback=true 时，本字段必填，URL 支持 HTTP 和 HTTPS
auto_split	Boolean	否	是否开启智能分轨（只支持 8000Hz 采样率单通道语音）
valid_times	List< ValidTime >	否	有效时间段信息，用来排除一些不必要的时间段
customization_id	String	否	使用定制模型特性的模型 id
class_vocabulary_id	Map	否	使用类热词特性的词表 ID，Map 的 Key 是类名，Value 是类热词词表 ID，设置方式见 JSON 字段示例
vocabulary_id	String	否	使用泛热词特性的词表 id

其中，ValidTime 对象的参数描述：

属性	类型	是否必须	说明
begin_time	Int	是	有效时间段的起始点时间偏移(单位: ms)
end_time	Int	是	有效时间段的结束点时间偏移(单位: ms)
channel_id	Int	是	有效时间段的作用音轨序号(从 0 开始)

输入参数组成的 JSON 字段示例：

```
{
  "appkey": "default",
  "token": "default",
  "file_link": "https://aliyun-nls.oss-cn-hangzhou.aliyuncs.com/asr/fileASR/examples/nls-sample-16k.wav",
  "auto_split": false,
```

```
  "vocabulary_id": "您的泛热词 ID",
  "valid_times": [
    {
      "begin_time": 200,
      "end_time": 2000,
      "channel_id": 0
    }
  ],
  "class_vocabulary_id": {
    "PERSON": "您的人名 ID",
    "ADDRESS": "您的地名 ID"
  }
}
```

响应结果

提交录音文件识别请求后，服务端返回该请求的响应：

- 返回 HTTP 状态：200 表示成功，更多状态码请查阅 HTTP 状态码；
- 输出参数：包含在响应的 Body 中，JSON 格式字符串：

属性	类型	是否必须	说明
task_id	String	是	识别任务 ID，用于识别结果的查询
status	Int	是	状态码，见服务码说明
status_message	String	是	状态说明，见服务码说明

输出参数示例：

```
{
  "header": {
    "appkey": "default",
    "gw_task_id": "0e7b7d5b0908431a9d482724950986de",
    "message_id": "serverf3b539f460184ea58ef72d6141",
    "namespace": "SpeechFileTranscriber",
    "status": 21050000,
    "status_message": "SUCCESS",
    "task_id": "fe28a941f7bc11e8a77e671efe28de2b"
  }
}
```

注意：如果出现错误，请将如上响应的 JSON 字符串反馈给对接人。

快速测试

测试音频文件：**nls-sample-16k.wav**，采样率为 16000Hz，请确认模型是否支持。
使用 cURL 命名行可以快速进行测试：

```
curl -H "Content-Type:application/json" -XPOST "http://gateway 所在
IP:8101/stream/v1/filetrans" -d '{"token":"default", "appkey":"default",
"auto_split":false,"file_link":"https://aliyun-nls.oss-cn-
hangzhou.aliyuncs.com/asr/fileASR/examples/nls-sample-16k.wav"}'
```

4.2 录音文件识别结果查询接口

在提交录音文件识别请求后，用户可以根据请求任务的 task_id 轮询识别结果。
录音文件识别结果查询报文实例：

```
GET
/stream/v1/filetrans?appkey=default&token=default&task_id=39b09d73fdd411e882ce3b46b16
f53ad HTTP/1.1
Host: gateway 所在 IP:8101
```

HTTP 请求行指定了 URL 和请求参数。

URL

URL 中指定了请求路径**/stream/v1/filetrans**。

协议	URL	方法
HTTP/1.1	http://gateway 所在 IP:8101/stream/v1/filetrans	GET

请求参数

用户通过提交录音文件识别请求获得的任务 ID 作为识别结果的查询接口参数，可以获取识别结果。在接口调用过程中，需要设置一定的查询时间间隔。

属性	类型	是否必须	说明
token	String	是	服务鉴权 token，专有云使用 default
appkey	String	是	业务方或者业务场景的标记，专有云使用 default
task_id	String	是	提交录音文件识别请求获取的任务 ID

如上 URL 和请求参数组成的完整请求链接为：

```
http://gateway 所在 IP:8101/stream/v1/filetrans?token=default&appkey=default&task_id={您的 task_id}
```

响应结果

提交录音文件识别结果查询请求后，服务端返回该查询请求的响应：

- 返回 HTTP 状态：200 表示成功，更多状态码请查阅 HTTP 状态码；
- 输出参数：包含在响应的 Body 中，JSON 格式字符串：

其中 header 结构体参数如下：

属性	类型	是否必须	说明
task_id	String	是	识别任务 ID
status	Int	是	状态码
status_message	String	是	状态说明
message_id	String	是	本次请求的 ID
gw_task_id	String	否	gateway 请求 ID，用于调试

其中，在 status_message 为 SUCCEED，playload 中包含完整的识别结果：

属性	类型	是否必须	说明
sentences	List< SentenceResult >	是	识别结果列表，以句子为单位

单句结果描述：

属性	类型	是否必须	说明
channel_id	Int	是	该句所属音轨 ID
begin_time	Int	是	该句的起始时间偏移，单位为毫秒
end_time	Int	是	该句的结束时间偏移，单位为毫秒
text	String	是	该句的识别文本结果
emotion_value	Int	是	情绪能量值 1-10，值越高情绪越强烈
silence_duration	Int	是	本句与上一句之间的静音时长，单位为秒
speech_rate	String	是	本句的平均语速，单位为每分钟字数

输出示例:

成功响应：


```
{
  "header": {
    "appkey": "default",
    "gw_task_id": "9bcb96a6a3354684b1f0a0bd5e3c9c36",
    "message_id": "server3d4f438d4a924613ac9762d68a",
    "namespace": "SpeechFileTranscriber",
    "status": 21050000,
    "status_message": "SUCCESS",
    "task_id": "7d54d835fd0b11e88a3b85575a6c6d40"
  },
  "payload": {
    "sentences": [
      {
        "silence_duration": 0,
        "end_time": 2365,
        "speech_rate": 177,
        "begin_time": 340,
        "text": "北京的天气。",
        "channel_id": 0,
        "emotion_value": 5
```

```
      }
    ]
  }
}
```

如果采用回调地址，格式如下：

```
{
  "result": [
    {
      "begin_time": 0,
      "channel_id": 0,
      "emotion_value": 6,
      "end_time": 354312,
      "silence_duration": 0,
      "speech_rate": 3,
      "text": "北京的天气。"
    }
  ],
  "task_id": "3bebda30f78a11e88cdc6d93bc2acfa3",
  "status_code": 21050000,
  "status_text": "SUCCESS",
  "request_time": 1543903371860,
  "solve_time": 1543903384845,
  "biz_duration": 11072,
  "enable_callback": true
}
```

- request_time 表示录音文件识别请求的时间；
- solve_time 表示录音文件识别完成的时间；
- biz_duration 表示语音的时长。

正在识别：

```
{
  "header": {
    "appkey": "default",
    "gw_task_id": "0e7b7d5b0908431a9d482724950986de",
    "message_id": "serverf3b539f460184ea58ef72d6141",
    "namespace": "SpeechFileTranscriber",
    "status": 21050001,
    "status_message": "RUNNING",
    "task_id": "fe28a941f7bc11e8a77e671efe28de2b"
  }
}
```

异常返回：

以文件下载失败为例：

```
{
  "header": {
    "appkey": "default",
    "gw_task_id": "4d5bdde998c3407b88e40483b1c6d1ea",
    "message_id": "server876566d6a0b0473c93183f053d",
    "namespace": "SpeechFileTranscriber",
    "status": 41050002,
    "status_message": "FILE_DOWNLOAD_FAILED",
    "task_id": "c6ac64a7fd1411e88a3b85575a6c6d40"
  }
}
```

注意：如果出现错误，请将如上响应的 JSON 字符串反馈给对接人。

快速测试

使用 cURL 命名行可以快速进行测试：

```
curl -XGET http://gateway 所在
IP:8101/stream/v1/filetrans?token=default&appkey=default&task_id={您的 task_id}
```

4.3 服务状态码

正常状态码：

状态码	状态描述	状态含义	解决方案
21050000	SUCCESS	成功	POST 方式的识别请求接口调用成功，或者 GET 方式的识别结果查询接口调用成功
21050001	QUEUEING	录音文件识别任务排队中	请稍后再发送 GET 方式的识别结果查询请求
21050002	RUNNING	录音文件识别任务运行中	请稍后再发送 GET 方式的识别结果查询请求

21050003	SUCCESS_WITH_NO_VALID_FRAGMENT	识别结果查询接口调用成功，但是没有识别到语音	检查录音文件是否有语音，或者语音时长太短

错误状态码：

说明：状态码 4 开头表示客户端错误，5 开头表示服务端错误。

状态码	状态描述	状态含义	解决方案
41050001	USER_BIZDURATION_QUOTA_EXCEED	用户单日时间超限	如业务量较大，请联系商务洽谈
41050002	FILE_DOWNLOAD_FAILED	文件下载失败	检查录音文件路径是否正确，或阿里云的服务器是否可以正常访问下载
41050003	FILE_CHECK_FAILED	文件格式错误	检查录音文件是否是单轨/双轨的 WAV 格式、MP3 格式
41050004	FILE_TOO_LARGE	文件过大	检查录音文件大小是否超过 128MB
41050005	FILE_NORMALIZE_FAILED	文件归一化失败	检查录音文件是否有损坏，是否可以正常播放
41050006	FILE_PARSE_FAILED	文件解析失败	检查录音文件是否有损坏，是否可以正常播放
41050007	MKV_PARSE_FAILED	MKV 解析失败	检查录音文件是否有损坏，是否可以正常播放
41050008	UNSUPPORTED_SAMPLE_RATE	采样率不支持	检查录音文件采样率是否是 8000、16000

41050009	UNSUPPORTED_ASR_GROUP	ASR 分组不支持	确认下 ak 和 appkey 是否一致
41050010	FILE_TRANS_TASK_EXPIRED	录音文件识别任务过期	TaskId 不存在，或者已过期
41050011	REQUEST_INVALID_FILE_URL_VALUE	请求 file_link 参数非法	请确认 file_link 参数格式是否正确
41050012	REQUEST_INVALID_CALLBACK_VALUE	请求 callback_url 参数非法	请确认 callback_url 参数格式是否正确，是否为空
41050013	REQUEST_PARAMETER_INVALID	请求参数无效	确认下请求 task 值为有效 JSON 格式字符串
41050014	REQUEST_EMPTY_APPKEY_VALUE	请求参数 app_key 值为空	请确认是否设置了 app_key 参数值
41050015	REQUEST_APPKEY_UNREGISTERED	请求参数 app_key 未注册	请确认请求参数 app_key 值是否设置正确，或者是否与阿里云账号的 AccessKey ID 同一个账号
51050000	INTERNAL_ERROR	内部通用错误	如果偶现可以忽略，重复出现请提交工单
51050001	VAD_FAILED	VAD 失败	如果偶现可以忽略，重复出现请提交工单
51050002	RECOGNIZE_FAILED	内部 alisr 识别失败	如果偶现可以忽略，重复出现请提交工单
51050003	RECOGNIZE_INTERRUPT	内部 alisr 识别中断	如果偶现可以忽略，重复出现请提交工单

51050004	OFFER_INTERRUPT	内部写入队列中断	如果偶现可以忽略，重复出现请提交工单
51050005	FILE_TRANS_TIMEOUT	内部整体超时失败	如果偶现可以忽略，重复出现请提交工单
51050006	FRAGMENT_FAILED	内部分断失败	如果偶现可以忽略，重复出现请提交工单

5. 示例代码

5.1 Java Demo

依赖：

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

代码：

```
import com.alibaba.fastjson.JSONObject;
import okhttp3.*;

import java.io.IOException;

public class FileTransRESTfulDemo {

   /**
    * 常量字段，请勿修改
    */
   private static final String KEY_APPKEY = "appkey";
```

```
    private static final String KEY_TOKEN = "token";
    private static final String KEY_FILE_LINK = "file_link";
    private static final String KEY_HEADER = "header";
    private static final String KEY_PAYLOAD = "payload";
    private static final String KEY_STATUS_MESSAGE = "status_message";
    private static final String KEY_TASK_ID = "task_id";

    private static final String STATUS_SUCCESS = "SUCCESS";
    private static final String STATUS_SUCCESS_WITH_NO_VALID_FRAGMENT =
"SUCCESS_WITH_NO_VALID_FRAGMENT";
    private static final String STATUS_RUNNING = "RUNNING";
    private static final String STATUS_QUEUEING = "QUEUEING";

    private String appkey;
    private String token;

    public FileTransRESTfulDemo(String appkey, String token) {
        this.appkey = appkey;
        this.token = token;
    }

    public String submitFileTransRequest(String url, String fileLink) {
        /**
         * 设置 HTTP POST 请求
         * 1.使用 HTTP 协议
         * 2.录音文件识别服务域名：您的 gateway 的域名: ip:port
         * 3.录音文件识别请求路径：/stream/v1/filetrans
         * 4.设置必须请求参数：appkey、token、file_link
         * 5.可选请求参数：auto_split、enable_callback、callback_url 等
         */
        JSONObject taskObject = new JSONObject();
        taskObject.put(KEY_APPKEY, appkey);
        taskObject.put(KEY_FILE_LINK, fileLink);
        taskObject.put(KEY_TOKEN, token);
        String task = taskObject.toJSONString();
        System.out.println(task);

        RequestBody requestBody = RequestBody.create(MediaType.parse("application/json"), task);
        Request request = new Request.Builder()
                .url(url)
                .header("Content-Type", "application/json")
                .post(requestBody)
                .build();

        String taskId = null;
        try {
            OkHttpClient client = new OkHttpClient();
            /**
             * 发送 HTTP POST 请求，处理服务端返回的响应
             */
            Response response = client.newCall(request).execute();
            int responseCode = response.code();
            String responseBody = response.body().string();
            System.out.println("请求结果：" + responseBody);
            response.close();
```

```
        if (200 == responseCode) {
            JSONObject result = JSONObject.parseObject(responseBody);
            result = result.getJSONObject(KEY_HEADER);
            String statusMessage = result.getString(KEY_STATUS_MESSAGE);
            if (STATUS_SUCCESS.equals(statusMessage)) {
                taskId = result.getString(KEY_TASK_ID);
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }

    return taskId;
}

public String getFileTransResult(String url, String taskId) {
    /**
     * 设置 HTTP GET 请求
     * 1.使用 HTTP 协议
     * 2.录音文件识别服务域名：您的 gateway 的域名: ip:port
     * 3.录音文件识别请求路径：/stream/v1/filetrans
     * 4.设置必须请求参数：appkey、token、task_id
     */
    url = url + "?appkey=" + appkey;
    url = url + "&token=" + token;
    url = url + "&task_id=" + taskId;

    System.out.println("url: " + url);

    Request request = new Request.Builder()
            .url(url)
            .get()
            .build();

    String statusMessage = null;
    String result = null;
    OkHttpClient client = new OkHttpClient();
    while (true) {
        try {
            /**
             * 发送 HTTP GET 请求，处理服务端返回的响应
             */
            Response response = client.newCall(request).execute();
            int responseCode = response.code();
            String responseBody = response.body().string();
            System.out.println("识别查询结果：" + responseBody);
            response.close();

            if (200 != responseCode) {
                break;
            }

            JSONObject rootObj = JSONObject.parseObject(responseBody);
            // 从 header 中获取状态信息
```

```
            JSONObject headerObj = rootObj.getJSONObject(KEY_HEADER);
            statusMessage = headerObj.getString(KEY_STATUS_MESSAGE);
            if (STATUS_RUNNING.equals(statusMessage) ||
STATUS_QUEUEING.equals(statusMessage)) {
                // 继续轮询，注意设置轮询时间间隔
                Thread.sleep(3000);
            }
            else {
                // 状态信息为成功，返回识别结果；状态信息为异常，返回空
                if (STATUS_SUCCESS.equals(statusMessage) ||
STATUS_SUCCESS_WITH_NO_VALID_FRAGMENT.equals(statusMessage)) {
                    result = rootObj.getJSONObject(KEY_PAYLOAD).toString();
                }
                break;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    return result;
  }

  public static void main(String args[]) {
    if (args.length < 1) {
        System.err.println("FileTransRESTfulDemo need params: <gateway 所在 IP>");
        System.exit(-1);
    }

    String ip = args[0];
    String appkey = "default";
    String token = "default";
    String port = "8101";

    String url = "http://" + ip + ":" + port + "/stream/v1/filetrans";
    String fileLink = "https://aliyun-nls.oss-cn-hangzhou.aliyuncs.com/asr/fileASR/examples/nls-
sample-16k.wav";

    FileTransRESTfulDemo demo = new FileTransRESTfulDemo(appkey, token);

    // 第一步：提交录音文件识别请求，获取任务 ID 用于后续的识别结果轮询
    String taskId = demo.submitFileTransRequest(url, fileLink);
    if (taskId != null) {
        System.out.println("录音文件识别请求成功，task_id: " + taskId);
    }
    else {
        System.out.println("录音文件识别请求失败！");
        return;
    }

    // 第二步：根据任务 ID 轮询识别结果
    String result = demo.getFileTransResult(url, taskId);
    if (result != null) {
        System.out.println("录音文件识别结果查询成功：" + result);
    }
```

```
        else {
            System.out.println("录音文件识别结果查询失败！ ");
        }
    }
}
```

5.2 Python Demo

注意：Python 2.x 请使用 httplib 模块；Python 3.x 请使用 http.client 模块。

```
# -*- coding: UTF-8 -*-

import json
import time

# Python 2.x 引入 httplib 模块
# import httplib

# Python 3.x 引入 http.client 模块
import http.client



def submitFileTransRequest(host, appKey, token, fileLink) :

    url = 'http://' + host + '/stream/v1/filetrans'

    # 设置 HTTP Headers
    httpHeaders = {
        'Content-Type': 'application/json'
        }

    # 设置 HTTP Body
    body = {'appkey': appKey, 'token': token, 'file_link': fileLink}
    body = json.dumps(body)

    print('The POST request body content: ' + body)

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

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

    conn.request(method='POST', url=url, body=body, headers=httpHeaders)

    # 处理服务端返回的响应
    response = conn.getresponse()
    print('Response status and response reason:')
    print(response.status ,response.reason)
```

```
    body = response.read()
    print('请求结果：' + str(body))

    taskId = None
    try:
        body = json.loads(body)
        result = body['header']
        statusMessage = result['status_message']
        if 'SUCCESS' == statusMessage :
            taskId = result['task_id']

    except ValueError:
        print('The response is not json format string')

    conn.close()

    return taskId



def getFileTransResult(host, appKey, token, taskId) :

    url = 'http://' + host + '/stream/v1/filetrans'

    # 设置 URL 请求参数
    url = url + '?appkey=' + appKey
    url = url + '&token=' + token
    url = url + '&task_id=' + taskId
    print(url)

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

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

    result = None

    while True:
        conn.request(method='GET', url=url)
        # 处理服务端返回的响应
        response = conn.getresponse()
        print('Response status and response reason:')
        print(response.status ,response.reason)

        body = response.read()
        print('识别查询结果：' + str(body))

        if 200 != response.status :
            break

        try:
            rootObj = json.loads(body)
            headerObj = rootObj['header']
```

```
            statusMessage = headerObj['status_message']
            if 'RUNNING' == statusMessage or 'QUEUEING' == statusMessage :
                # 继续轮询
                time.sleep(3)
            else :
                if 'SUCCESS' == statusMessage or 'SUCCESS_WITH_NO_VALID_FRAGMENT' ==
statusMessage :
                    result = rootObj['payload']

                break
        except ValueError:
            print('The response is not json format string')

    conn.close()

    return result



appKey = 'default'
token = 'default'

host = 'gateway 所在 IP:8101'
fileLink = 'https://aliyun-nls.oss-cn-hangzhou.aliyuncs.com/asr/fileASR/examples/nls-sample-
16k.wav'

# 第一步：提交录音文件识别请求，获取任务 ID 用于后续的识别结果轮询
taskId = submitFileTransRequest(host, appKey, token, fileLink)
print(type(taskId))
if taskId is None:
    print('录音文件识别请求失败！')
else:
    print('录音文件识别请求成功，task_id：' + str(taskId))

# 第二步：根据任务 ID 轮询识别结果
result = getFileTransResult(host, appKey, token, taskId)
if result is None :
    print('录音文件识别结果查询失败！')
else :
    print('录音文件识别结果查询成功：' + str(result))
```

5.3 C++Demo

说明：

C++ Demo 使用了第三方库 curl 处理 HTTP 的请求和响应，使用 jsoncpp 处理请求和响应中 JSON 格式字符串的设置。Demo 相关文件在 nls-cpp-example/nls-cpp-restful 目录下：

目录说明：

- CMakeLists.txt Demo 工程的 CMakeList 文件；

- demo

文件名	描述
restfulFileTransDemo.cpp	录音文件识别 RESTful API Demo

- include

文件名	描述
curl	curl 库头文件目录
json	json 库头文件目录

- lib 根据平台不同，可以选择 linux 版本（glibc:2.5 及以上, Gcc4, Gcc5）、windows 版本（VS2013、VS2015）。
- readme.txt 说明
- release.log 更新记录
- version 版本号
- build.sh demo 编译脚本

注意：

1. Linux 环境下，运行环境最低要求：Glibc 2.5 及以上， Gcc4、Gcc5。
2. Windows 下需要用户自己搭建 demo 工程。
3. C++ demo 的下载包里自带了测试音频 sample.pcm。

编译运行：

```
1. 请确认本地系统以安装 Cmake，最低版本 2.4
2. cd path/to/sdk/lib
3. tar -zxvpf linux.tar.gz
4. cd path/to/sdk
```

5. 执行[./build.sh]编译 demo
6. 编译完毕，进入 demo 目录，执行[./restfulFileTransDemo <gateway 所在 IP>]

如果不支持 cmake，可以尝试手动编译：
1: cd path/to/sdk/lib
2: tar -zxvpf linux.tar.gz
3: cd path/to/sdk/demo
4: g++ -o restfulFileTransDemo restfulFileTransDemo.cpp -I path/to/sdk/include -L path/to/sdk/lib/linux -ljsoncpp -lssl -lcrypto -lcurl -D_GLIBCXX_USE_CXX11_ABI=0
5: export LD_LIBRARY_PATH=path/to/sdk/lib/linux/
6: ./restfulFileTransDemo <gateway 所在 IP>

Windows 平台需要用户自己搭建工程。

代码：

```
#ifdef _WIN32
#include <Windows.h>
#else
#include <unistd.h>
#endif

#include <iostream>
#include <string>
#include <map>
#include <fstream>
#include <sstream>

#include "curl/curl.h"
#include "json/json.h"

using namespace std;

#define KEY_APPKEY "appkey"
#define KEY_TOKEN "token"
#define KEY_FILE_LINK "file_link"
#define KEY_HEADER "header"
#define KEY_PAYLOAD "payload"
#define KEY_STATUS_MESSAGE "status_message"
#define KEY_TASK_ID "task_id"

#define STATUS_SUCCESS "SUCCESS"
#define STATUS_SUCCESS_WITH_NO_VALID_FRAGMENT
"SUCCESS_WITH_NO_VALID_FRAGMENT"
#define STATUS_RUNNING "RUNNING"
#define STATUS_QUEUEING "QUEUEING"
```

```
#ifdef _WIN32
string UTF8ToGBK(const string& strUTF8) {
    int len = MultiByteToWideChar(CP_UTF8, 0, strUTF8.c_str(), -1, NULL, 0);
    unsigned short * wszGBK = new unsigned short[len + 1];
    memset(wszGBK, 0, len * 2 + 2);

    MultiByteToWideChar(CP_UTF8, 0, (char*)strUTF8.c_str(), -1, (wchar_t*)wszGBK, len);

    len = WideCharToMultiByte(CP_ACP, 0, (wchar_t*)wszGBK, -1, NULL, 0, NULL, NULL);

    char *szGBK = new char[len + 1];
    memset(szGBK, 0, len + 1);
    WideCharToMultiByte(CP_ACP, 0, (wchar_t*)wszGBK, -1, szGBK, len, NULL, NULL);

    string strTemp(szGBK);
    delete[] szGBK;
    delete[] wszGBK;

    return strTemp;
}

#endif

size_t responseBodyCallback(void* ptr, size_t size, size_t nmemb, void* userData) {
    size_t len = size * nmemb;
    char* pBuf = (char*)ptr;
    string response = string(pBuf, pBuf + len);
#ifdef _WIN32
    response = UTF8ToGBK(response);
#endif

    string* bodyContent = (string*)userData;
    (*bodyContent).append(response);

    return len;
}

string submitFileTransRequest(const string& url, const string& appKey, const string& token, const
string& fileLink) {
    CURL* curl = NULL;
    CURLcode curlCode = CURLE_OK;

    curl = curl_easy_init();
    if (curl == NULL) {
        return "";
    }

    // 设置 HTTP POST URL
    curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
    curl_easy_setopt(curl, CURLOPT_POST, 1L);

    // 设置 HTTP POST 请求头部
    struct curl_slist* headers = NULL;
    // Content-Type
    headers = curl_slist_append(headers, "Content-Type:application/json");
    curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headers);
```

```
    // 设置 HTTP POST 请求体
    Json::Value root;
    Json::FastWriter writer;
    root[KEY_APPKEY] = appKey;
    root[KEY_TOKEN] = token;
    root[KEY_FILE_LINK] = fileLink;

    string task = writer.write(root);
    cout << "POST request Body: " << task << endl;

    curl_easy_setopt(curl, CURLOPT_POSTFIELDS, task.c_str());
    curl_easy_setopt(curl, CURLOPT_POSTFIELDSIZE, task.length());

    // 设置获取响应的 HTTP Body 回调函数
    string bodyContent = "";
    curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, responseBodyCallback);
    curl_easy_setopt(curl, CURLOPT_WRITEDATA, &bodyContent);


    // 发送 HTTP POST 请求
    curlCode = curl_easy_perform(curl);
    // 获取 HTTP 响应状态编码
    long httpCode = 0;
    curl_easy_getinfo(curl, CURLINFO_RESPONSE_CODE, &httpCode);

    // 释放资源
    curl_slist_free_all(headers);
    curl_easy_cleanup(curl);

    if (curlCode != CURLE_OK) {
        cerr << "curl_easy_perform failed: " << curl_easy_strerror(curlCode) << endl;
        return "";
    }

    /**
     * 处理服务端返回的响应
     */
    cout << "请求结果：" << bodyContent << endl;

    string taskId = "";
    if (200 == httpCode) {
        Json::Value rootObj;
        Json::Reader jsonReader;
        if (jsonReader.parse(bodyContent, rootObj)) {
            Json::Value headerObj = rootObj[KEY_HEADER];
            string statusMessage = headerObj[KEY_STATUS_MESSAGE].asString();
            if (statusMessage.compare(STATUS_SUCCESS) == 0) {
                taskId = headerObj[KEY_TASK_ID].asString();
            }
        }
    }

    return taskId;
}
```

```
string getFileTransResult(const string& url, const string& appKey, const string& token, const string& taskId) {
    CURL* curl = NULL;
    CURLcode curlCode = CURLE_OK;

    curl = curl_easy_init();
    if (curl == NULL) {
        return "";
    }

    // 设置 HTTP URL 请求参数
    ostringstream oss;
    oss << url;
    oss << "?appkey=" << appKey;
    oss << "&token=" << token;
    oss << "&task_id=" << taskId;

    string request = oss.str();
    cout << request << endl;

    curl_easy_setopt(curl, CURLOPT_URL, request.c_str());

    // 设置获取响应的 HTTP Body 回调函数
    string bodyContent = "";
    curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, responseBodyCallback);
    curl_easy_setopt(curl, CURLOPT_WRITEDATA, &bodyContent);

    string result = "";
    while (true) {
        // 每次发送查询请求前，先清除上次查询到的内容
        bodyContent = "";

        // 发送 HTTP GET 请求
        curlCode = curl_easy_perform(curl);

        long httpCode = 0;
        curl_easy_getinfo(curl, CURLINFO_RESPONSE_CODE, &httpCode);

        cout << "识别查询结果： " << bodyContent << endl;

        if (curlCode != CURLE_OK || httpCode != 200) {
            cerr << "curl_easy_perform failed: " << curl_easy_strerror(curlCode) << endl;
            break;
        }

        /**
         * 处理服务端返回的响应
         */
        Json::Value rootObj;
        Json::Reader jsonReader;
        if (jsonReader.parse(bodyContent, rootObj)) {
            Json::Value headerObj = rootObj[KEY_HEADER];
            string statusMessage = headerObj[KEY_STATUS_MESSAGE].asString();
```

```
        if (statusMessage.compare(STATUS_RUNNING) == 0 ||
statusMessage.compare(STATUS_QUEUEING) == 0) {
            // 继续轮询
#if defined(_WIN32)
            Sleep(3000);
#else
            usleep(3000 * 1000);
#endif
        }
        else {
            // 状态信息为成功，返回识别结果；状态信息为异常，返回空
            if (statusMessage.compare(STATUS_SUCCESS) == 0 ||
                statusMessage.compare(STATUS_SUCCESS_WITH_NO_VALID_FRAGMENT) == 0)
{
                Json::FastWriter jsonWriter;
                result = jsonWriter.write(rootObj[KEY_PAYLOAD]);
            }

            break;
        }
    }
  }

  // 释放资源
  curl_easy_cleanup(curl);

  return result;
}


int main(int argc, char* argv[]) {
  if (argc < 2) {
    cerr << "params is not valid. Usage: ./demo <gateway 所在 IP>" << endl;
    return -1;
  }

  string ip = argv[1];
  string appKey = "default";
  string token = "default";
  string port = "8101";

  string fileLink = "https://aliyun-nls.oss-cn-hangzhou.aliyuncs.com/asr/fileASR/examples/nls-
sample-16k.wav";

  string url = "http://" + ip + ":" + port + "/stream/v1/filetrans";

  // 全局只初始化一次
  curl_global_init(CURL_GLOBAL_ALL);

  // 第一步：提交录音文件识别请求，获取任务 ID 用于后续的识别结果轮询
  string taskId = submitFileTransRequest(url, appKey, token, fileLink);
  if (taskId.empty()) {
    cerr << "录音文件识别请求失败！" << endl;

    curl_global_cleanup();
```

```
        return -1;
    }
    else {
        cout << "录音文件识别请求成功，task_id：" << taskId << endl;
    }

    // 根据任务 ID 轮询识别结果
    string result = getFileTransResult(url, appKey, token, taskId);
    if (result.empty()) {
        cerr << "录音文件识别结果查询失败！" << endl;
    }
    else {
        cout << "录音文件识别结果查询成功：" << result << endl;
    }

    curl_global_cleanup();

    return 0;
}
```