

中科院声学所在线识别引擎

代码模块接口设计

版 本 历 史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本/状态 | 作者 | 参与者 | 起止日期 | 备注 |
| V1.0 | 段志亮 |  | 2021-11-02 | 创建文件 |
|  |  |  |  |  |

目录

[1.概述 3](#_Toc86847441)

[**1.1 编写目的** 3](#_Toc86847442)

[2.总体设计 4](#_Toc86847443)

[2.1 总体需求 4](#_Toc86847444)

[3.识别服务模块接口设计 5](#_Toc86847445)

[3.1 数据接收及发送模块 5](#_Toc86847446)

[3.1.2 网络通信接口 6](#_Toc86847447)

[3.2 数据解析及封装模块 8](#_Toc86847448)

[3.2.1数据解析接口 8](#_Toc86847449)

[3.3 会话流程处理模块 11](#_Toc86847450)

[3.3.1 流程控制接口 12](#_Toc86847451)

[3.3.2 获取会话接口 13](#_Toc86847452)

[3.3.3 读取语音接口 14](#_Toc86847453)

[3.3.4 语音处理接口 15](#_Toc86847454)

[4.代码目录 19](#_Toc86847455)

[4.1 代码目录结构 19](#_Toc86847456)

[4.2 主要代码文件说明 21](#_Toc86847457)

[4.2.1 redis操作模块 21](#_Toc86847458)

[4.2.2请求解析和返回结果封装模块 21](#_Toc86847459)

[4.2.3 流程控制模块 21](#_Toc86847460)

[4.3 代码编译 21](#_Toc86847461)

# 1.概述

1

**1.1 编写目的**

本文档主要描述识别服务端内部接口设计及接口中参数的数据结构，本文档的读者对象为在线识别系统的开发人员、测试人员、系统维护人员及接入识别系统的第三方业务人员，通过本文档能够从总体上了解识别系统内部的数据流向及处理方式。

本说明给出在线识别系统的接口设计说明，包括接口安全、版本兼容性、数据格式、服务端的异常处理等。

目的在于：

* 为开发人员提供依据；
* 为代码修改、维护提供条件；

# 2.总体设计

2

## 2.1 总体需求

接口需求：

1，接口参数简单明了，参数有实际意义；

2，接口功能定义明确；

3，接口的数据特性明确；

4，接口的规格和技术要求明确；

5，接口程序的数据处理逻辑清晰；

# 3.识别服务模块接口设计

3

识别服务包含接收数据模块、解析及封装数据模块和流程控制模块。

模块间关系如下：



## 3.1 数据接收及发送模块

识别服务和tomcat之间通过socket通信，数据接收及发送模块负责接收tomcat发送过来的请求数据并将处理后的数据返回给tomcat；

处理流程如下：



### 3.1.2 网络通信接口

**void** **CallBackFunc\_Server**(std::string &out\_action,

std::string &out\_msg,

**const** std::string &\_in\_action,

**const** std::string &\_in\_msg)

接口功能：该函数主要和能力接口进行网络通信的数据交互，接收能力接口端发送的请求数据，并将识别后的数据返回给能力接口。

|  |  |  |
| --- | --- | --- |
| 接口参数 | 数据格式 | 参数说明 |
| out\_action | 字符串 | 返回识别信息的编码格式 |
| out\_msg | xml字符串 | 返回的识别信息 |
| \_in\_action | 字符串 | 请求的命令 |
| \_in\_msg | Json字符串 | 请求的数据信息 |

例如：

\_in\_action = grs

\_in\_msg={

"request": {

"cmd": "grs",

"syncid": "1306"

},

"sid": "fb4711c019d4439a97c6d6acad176cf2"

}

out\_action=utf-8

<?xml version="1.0" encoding="utf-8"?>

out\_msg =

<results>

<sid>fb4711c019d4439a97c6d6acad176cf2</sid>

<recStatus>5</recStatus>

<engine\_name>192.168.0.52</engine\_name>

<result>[{ "text": "确认。", "phoneme": "确 认。", "segtime": "0.29 0.45 0.53 0.62", "score": 99.95304107666016 }]</result>

<keyword/>

<keywordtime/>

<bg>1</bg>

<ed>1</ed>

</results>

## 3.2 数据解析及封装模块

识别服务需要将接收到的请求数据进行解析处理，tomcat发送到识别服务端是数据形式为json字符串，识别服务将json字符串解析为本地方便使用的结构体形式；

识别服务返回数据到tomcat需要按照设计的格式进行返回，需要使用数据的封装模块，将返回数据进行整理封装，然后通过socket方式返回到tomcat端。

### 3.2.1数据解析接口

**bool** **deserialize**(redis\_param\* item, **const** **char**\* buffer, size\_t size)

接口功能：解析接收到的数据为服务方便使用的结构形式。

|  |  |  |
| --- | --- | --- |
| 接口参数 | 数据格式 | 参数说明 |
| item | 结构体 | 接收解析后数据 |
| buffer | Json字符串 | 待解析json字符串 |
| size | 整数 | 待解析json字符串长度 |

例如：

item结构信息如下

**struct** redis\_param{

std::string cmd; //请求命令(ssb/auw/grs/sse)

std::string sid; //会话id(8909b222c9554cf9a61cad64aa44dc6e)

std::string auf; //采样率 (audio/L16;rate=8000)

std::string aue; //语音是否压缩(raw)

std::string data; //请求数据

std::string syncid;//语音包序号(1,2,3...n)

};

buffer={

"request": {

"cmd": "grs",

"syncid": "1306"

},

"sid": "8909b222c9554cf9a61cad64aa44dc6e"

}

size=82

3.2.2数据封装接口

std::string **getXmlStr**(ACand\* cand, asr\_resp\_t res\_data, **int** selectNum, **char**\* encoding, **int** realTimeContent,**int** soundType)

接口功能：封装返回的识别数据。

|  |  |  |
| --- | --- | --- |
| 接口参数 | 数据格式 | 参数说明 |
| cand | 结构体 | 存放识别结果信息 |
| res\_data | 结构体 | 会话信息 |
| selectNum | 整数 | 输出结果数量 |
| encoding | 字符串 | 编码格式 |
| realTimeContent | 整数 | 区分返回结果的形式 |
| soundType | 整数 | 一句话开始结束标识 |

例如：

cand结构信息如下

**struct** ACand

{

**char** \*text; // 候选文本结果

**float** score; // 候选得分，0.00~100.00，candNum个

**float** startTime; // 开始时刻，单位为秒s

**float** endTime; // 结束时刻，单位为秒s

ACand \*next; // 到下一个的指针

};

res\_data结构信息如下

**typedef** **struct** \_asr\_resp\_t

{

std::string sid; //会话id

**bool** isLast; //是否是最后一包

std::string result; //识别结果

**float** confidence; //置信度

std::string phoneme; //识别结果分词

std::string segtime; //结果分词时间

std::string keyword; //关键词

std::string keywordtime;//关键词时间

**float** startframe; //是否是语音起点

**float** endframe; //是否是语音尾点

std::string engine\_name; //服务器地址

} asr\_resp\_t;

selectNum =1

encoding =utf-8

realTimeContent=0

soundType =1

## 3.3 会话流程处理模块

会话流程处理是识别服务的核心模块，负责从会话开始到结束的整个流程控制；

会话开始从redis中获取会话信息，将当前服务的信息存入redis；tomcat获取到服务信息后，通过socket方式将语音流数据发送到识别服务，识别服务调用引擎接口对语音进行识别处理，语音发送结束后，返回识别结果信息，结束会话。

会话控制流程图如下：



### 3.3.1 流程控制接口

static void \*DecodeProcThread(void \*parameter)

接口功能：该函数主要实现数据处理工程的逻辑控制功能；

|  |  |  |
| --- | --- | --- |
| 接口参数 | 数据格式 | 参数说明 |
| parameter | 指针 | 存放控制流程会话信息 |

例如：

**parameter指针变量包含如下结构体信息**

**typedef** **struct** \_asr\_param\_t

{

**int** sessId; //线程序号

**bool** isFirstRecog;//会话开始标识

queue<rec\_t \* > mRecArray; //识别结果队列

**int** res\_num; //结果数量

} asr\_param\_t;

### 3.3.2 获取会话接口

**int** **RedisProcesser::GetSsbRequest**(redis\_param \*rparam)

接口功能：该函数接口主要实现阻塞等待从redis队列中获取新的会话信息，同时将自身所在服务器ip和端口写入redis队列中。

|  |  |  |
| --- | --- | --- |
| 接口参数 | 数据格式 | 参数说明 |
| rparam | 结构体 | 存放获取到的会话信息 |

例如：

结构体信息如下

**struct** redis\_param{

std::string cmd; //请求命令(ssb/auw/grs/sse)

std::string sid; //会话id(8909b222c9554cf9a61cad64aa44dc6e)

std::string auf; //采样率 (audio/L16;rate=8000)

std::string aue; //语音是否压缩(raw)

std::string data; //请求数据

std::string syncid;//语音包序号(1,2,3...n)

};

### 3.3.3 读取语音接口

**int** **getRequest**(redis\_param &rparam,**int** num)

接口功能：主要从指定的队列中读取语音数据据，如果读取数据超时返回异常信息。

|  |  |  |
| --- | --- | --- |
| 接口参数 | 数据格式 | 参数说明 |
| rparam | 结构体 | 存放获取到的会话信息 |
| num | 整数 | 队列索引 |

例如：

结构体信息如下

**struct** redis\_param{

std::string cmd; //请求命令(ssb/auw/grs/sse)

std::string sid; //会话id(8909b222c9554cf9a61cad64aa44dc6e)

std::string auf; //采样率 (audio/L16;rate=8000)

std::string aue; //语音是否压缩(raw)

std::string data; //请求数据

std::string syncid;//语音包序号(1,2,3...n)

};

num = 1 //队列索引

### 3.3.4 语音处理接口

**int** **processAuw**(asr\_var\_t &threadVar,

asr\_var\_temp\_t &threadVarTemp,

**void** \*parameter,

redis\_param &rparam,

std::string &bitRate,

Data\_t \*out)

接口功能：主要实现对数据数据的识别处理，并返回对应的识别信息。

|  |  |  |
| --- | --- | --- |
| 接口参数 | 数据格式 | 参数说明 |
| threadVar | 结构体 | 存放对接收数据处理后的信息 |
| threadVarTemp | 结构体 | 存放每次请求的临时信息 |
| parameter | 指针 | 当前会话的状态信息 |
| rparam | 结构体 | 存放客户端请求参数 |
| bitRate | 字符串 | 语音采样率 |
| out | 结构体指针 | 输出识别信息 |

例如；

**typedef** **struct** \_asr\_var\_t

{

**char** recName[MAX\_PATH]; //保存接收语音的文件名字

**char** pcmName[MAX\_PATH]; //保存转码语音的文件名字

u\_int readLen; // 从客户端收到的"本包"语音长度

**int** sendLen; // 原始语音缓存buffer（pcmBuf）的长度

**int** decodeLen; // 原始语音缓存buffer中已经解码的长度

**int** dataSize; // 可以解码的语音长度

FILE \*fpRec; // 原始语音文件

**void** \*fpPcm; // 解码后语音文件

**int** inSampleRate; //采样率

**int** compressRate; //压缩比例

**int** isPcmType; //语音格式

**bool** isCheckHeader; //是否检测语音格式

**bool** isNeedRestart; //是否需要启动引擎

asr\_head\_t inviteHead; //请求信息

**unsigned** **char** \*recBuf; // 源语音数据包存放

**unsigned** **char** \*pcmBuf; // 源语音数据存放buffer，缓存数据

**short** \*voicedata; // 解码后的语音数据，发送到decoder中识别

**int** sessId;

**char** coding\_type[MAX\_RES\_LEN]; //g711音频编码格式：A-law或U-law

} asr\_var\_t;

**typedef** **struct** \_asr\_var\_temp\_t

{

u\_int recBytes; //接收语音总长度

u\_int conBytes; //pcm语音总长度

g72x\_state state; //g72x实例

**bool** isSendData; //是否发送语音

**bool** isAbort; //识别是否终止

**bool** isStart; //引擎是否启动

u\_int preIdx; //前一语音包序号

**int** pcmLen; //pcm语音包长度

**int** opt\_adpcm; //语音格式临时变量

u\_int in\_buffer; //语音转码临时变量

**int** in\_bits; //语音转码临时变量

**struct** timeval start\_recog, end\_recog, start\_result;//识别时间统计

**long** mtime\_recog, mtime\_result, seconds, useconds, differ; //识别时间统计

} asr\_var\_temp\_t;

**parameter指针变量包含如下结构体信息**

**typedef** **struct** \_asr\_param\_t

{

**int** sessId; //线程序号

**bool** isFirstRecog;//会话开始标识

queue<rec\_t \* > mRecArray; //识别结果队列

**int** res\_num; //结果数量

} asr\_param\_t;

**struct** redis\_param{

std::string cmd; //请求命令(ssb/auw/grs/sse)

std::string sid; //会话id(8909b222c9554cf9a61cad64aa44dc6e)

std::string auf; //采样率 (audio/L16;rate=8000)

std::string aue; //语音是否压缩(raw)

std::string data; //请求数据

std::string syncid;//语音包序号(1,2,3...n)

};

bitRate=8000

out->msg=<?xml version="1.0" encoding="utf-8"?><results><sid>8909b222c9554cf9a61cad64aa44dc6e</sid><recStatus>5</recStatus><engine\_name>192.168.0.52</engine\_name><result>[{

"text": "确认。",

"phoneme": "确 认。",

"segtime": "0.29 0.45 0.53 0.62",

"score": 99.95304107666016

}]</result><keyword></keyword><keywordtime></keywordtime><bg>1</bg><ed>1</ed></results>

# 4.代码目录

4

## 4.1 代码目录结构

decoder-code

├── amr //amr语音解码库

├── include //依赖头文件目录

├── lib //依赖库目录

├── src

│  ├── amr\_decode.cpp //amr语音解码函数定义

│  ├── amr\_decode.h

│  ├── CommonDefine.h //共用变量声明文件

│  ├── CommonMethod.cpp

│  ├── CommonMethod.h //共用方法声明文件

│  ├── Encode.cpp

│  ├── Encode.h //base64编解码

│  ├── g711.c

│  ├── g721.c

│  ├── g723\_24.c

│  ├── g723\_40.c

│  ├── g72x.c

│  ├── g72x.h //g72x编解码

│  ├── hpmp.h

│  ├── Makefile //编译脚本

│  ├── Makefile.common.variables

│  ├── Markup.cpp

│  ├── Markup.h //xml解析

│  ├── RedisProcesser.cpp

│  ├── RedisProcesser.h //redis操作

│  ├── server.cpp //识别逻辑控制

│  ├── server.h

│  ├── wavwriter.c //保存语音

│  └── wavwriter.h

└── third\_party //日志第三方库

└── rapidjson //json解析

## 4.2 主要代码文件说明

### 4.2.1 redis操作模块

RedisProcesser.cpp该cpp文件中主要包含连接redis、读写redis操作方法，作为 共用方法类供其他模块使用。

### 4.2.2请求解析和返回结果封装模块

CommonMethod.cpp该cpp定义了数据解析、数据封装、异常数据封装、字符串拼接处理等操作方法。

### 4.2.3 流程控制模块

server.cpp该cpp功能为启动线程池，调用其他模块控制每个线程的处理流程，异常出现时能正常释放会话，新的会话进来后能正常处理。

## 4.3 代码编译

1，编译第三方库

执行命令 sh build\_thirdparty.sh，编译出的头文件在include目录下，库文件在lib目录下；

2，编译decoder可执行文件

进入src目录执行 make命令，生成二进制文件decoder，将此二进制文件替换到部署包的Decoder/bin目录下即可。