

中科院声学所在线识别引擎

详细设计说明

版 本 历 史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本/状态 | 作者 | 参与者 | 起止日期 | 备注 |
| V1.0 | 段志亮 |  | 2021-11-03 | 创建文件 |
|  |  |  |  |  |

目录

[1.概述 3](#_Toc86850449)

[1.1编写目的 3](#_Toc86850450)

[1.2 读者对象 3](#_Toc86850451)

[1.3术语定义 4](#_Toc86850452)

[2.总体设计 4](#_Toc86850453)

[2.1 总体需求 5](#_Toc86850454)

[2.2 运行环境 5](#_Toc86850455)

[2.2.1 软件环境 5](#_Toc86850456)

[2.2.2 硬件环境 5](#_Toc86850457)

[2.3服务总体结构设计 6](#_Toc86850458)

[2.3.1 识别服务总体逻辑 6](#_Toc86850459)

[2.3.2服务总体时序流程 7](#_Toc86850460)

[3.识别服务模块设计 7](#_Toc86850461)

[3.1 数据接收及发送模块 8](#_Toc86850462)

[3.2 数据解析及封装模块 9](#_Toc86850463)

[3.3 会话流程处理模块 9](#_Toc86850464)

[4.数据设计 11](#_Toc86850465)

[4.1 与能力接口交互数据设计 11](#_Toc86850466)

[4.1.1 会话开始输入、输出数据 11](#_Toc86850467)

[4.1.2 会话交互输入、输出数据 13](#_Toc86850468)

[4.1.3 获取会话结果输入、输出数据 15](#_Toc86850469)

[4.1.4 会话结束输入、输出参数 16](#_Toc86850470)

[4.2 模块内数据结构设计 17](#_Toc86850471)

[4.2.1 接收网络请求数据 17](#_Toc86850472)

[4.2.2 数据封装数据结构设计 20](#_Toc86850473)

[4.2.3 流程控制数据结构设计 22](#_Toc86850474)

# 1.概述

1

1.1编写目的

本文档主要用于为实现系统功能而进行的系统数据设计说明，具体描述了系统包含的的软件模块的实现流程、功能、接口、数据结构等内容，供项目组开发人员和软件维护人员阅读。

## 1.2 读者对象

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的架构形式及数据流向。

本说明给出在线识别系统的设计说明，包括最终实现的系统必须满足的功能、性能、接口、附属测试工具程序及设计约束等。

目的在于：

* 为开发人员提供依据；
* 为代码修改、维护提供条件；

## 1.3术语定义

语音识别（Automatic Speech Recognition），简称（ASR）。

# 2.总体设计

2

## 2.1 总体需求

功能需求：满足8k16bit pcm、16k16bit pcm语音转文字功能；

接口需求：接口参数简单明了，接口中的每个参数都要有实际意义，保证接口调用流程清晰。

系统验证工具：提供系统完整性，功能正确性的验证工具；

## 2.2 运行环境

### 2.2.1 软件环境

|  |  |  |
| --- | --- | --- |
| 分类 | 名称 | 版本 |
| 操作系统 | Centos | 7.0以上 |
| 数据库 | Redis | 3.2.0 |
| 能力接口 | Tomcat | 9.0.35 |

### 2.2.2 硬件环境

|  |  |  |
| --- | --- | --- |
| 服务器 | 最低配置 | 推荐配置 |
| redis服务器 | CPU：1  内存：256M | CPU：1  内存：1G |
| 识别服务器 | CPU：4  内存：10G  磁盘：20G | CPU：10  内存：15G  磁盘：30G |

## 2.3服务总体结构设计

### 2.3.1 识别服务总体逻辑

识别服务总体逻辑图如下：



### 2.3.2服务总体时序流程

服务总体时序流程如下所示：



# 3.识别服务模块设计

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。

## 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：



**相关接口介绍：**

**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。

## 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（详见数据设计），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。

相关接口介绍：

**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。

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。

## 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：



相关接口介绍：

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；

# 4.数据设计

4

## 4.1 与能力接口交互数据设计

### 4.1.1 会话开始输入、输出数据

会话开始输入数据：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| aue | 字符串 | 非编码原始语音 |
| auf | 字符串 | 语音采样率 |
| cmd | 字符串 | 会话状态 |
| sid | 字符串 | 会话Id |

如下实例：

{

"request": {

"aue": "raw",

"auf": "audio/L16;rate=8000",

"cmd": "ssb"

},

"sid": "16e374f04d8348e487dab492552537a8"

}

会话开始输出数据：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| ret | 整型 | 响应状态 |
| sid | 字符串 | 会话Id |
| addr | 字符串 | 识别服务地址信息 |

**实例如下：**

{  
    **"response"**:{  
        **"ret"**:**0**  
    },  
    **"sid"**:**"16e374f04d8348e487dab492552537a8"**,  
    **"addr"**:**"192.168.0.52:10200"**  
}

### 4.1.2 会话交互输入、输出数据

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| data | 字符串 | Base64加密的语音数据 |
| cmd | 字符串 | 会话状态 |
| syncid | 字符串 | 语音包序号 |
| sid | 字符串 | 会话Id |

实例如下：

{  
    **"request"**:{  
        **"data"**:**""**,  
        **"cmd"**:**"auw"**,  
        **"syncid"**:**"1"**  
    },  
    **"sid"**:**"16e374f04d8348e487dab492552537a8"**  
}

输出数据：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| sid | 字符串 | 会话Id |
| recStatus | 字符串 | 识别状态 |
| engine\_name | 字符串 | 引擎地址 |
| result | json字符串 | 识别结果 |
| bg | 字符串 | 检测到有效语音 |
| ed | 字符串 | 检测到有效语音尾点 |

**实例如下：**

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

<results>

<sid>**16e374f04d8348e487dab492552537a8**</sid>

<recStatus>5</recStatus>

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

<result>[{ "text": "你好。"]</result>

<bg>1</bg>

<ed>0</ed>

</results>

### 4.1.3 获取会话结果输入、输出数据

输入参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| cmd | 字符串 | 会话状态 |
| syncid | 字符串 | 序号 |
| sid | 字符串 | 会话Id |

**实例如下：**

{  
    **"request"**:{  
        **"cmd"**:**"grs"**,  
        **"syncid"**:**"1306"**  
    },  
    **"sid"**:**"16e374f04d8348e487dab492552537a8"**  
}

输出参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| sid | 字符串 | 会话Id |
| recStatus | 字符串 | 识别状态 |
| engine\_name | 字符串 | 引擎地址 |
| result | json字符串 | 识别结果 |
| bg | 字符串 | 检测到有效语音 |
| ed | 字符串 | 检测到有效语音尾点 |

**实例如下：**

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

<results>

<sid>**16e374f04d8348e487dab492552537a8**</sid>

<recStatus>5</recStatus>

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

<result>[{ "text": "你好。", "phoneme": "你好。", "segtime": "0.31 0.86", "score": 1.0 }]</result>

<bg>1</bg>

<ed>1</ed>

</results>

### 4.1.4 会话结束输入、输出参数

输入参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| cmd | 字符串 | 会话状态 |
| sid | 字符串 | 会话Id |

实例如下：

{  
    **"request"**:{  
        **"cmd"**:**"sse"**  
    },  
    **"sid"**:**"16e374f04d8348e487dab492552537a8"**  
}

输出参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| ret | 整型 | 会话结束状态 |
| sid | 字符串 | 会话Id |

**实例如下：**

{  
    **"ret"**:0,  
    **"sid"**:**"16e374f04d8348e487dab492552537a8"**  
}

## 4.2 模块内数据结构设计

### 4.2.1 接收网络请求数据

**结构体如下：**

**typedef** **struct** Data

{

**char** action[32];

**char** \*msg;

}Data\_t;

|  |  |  |
| --- | --- | --- |
| **结构体参数** | **类型** | **说明** |
| action | 字符串 | 请求命令（取值ssb、auw、grs、sse） |
| msg | json字符串 | 请求的数据信息 |

关联结构数据

**struct** redis\_param{

std::string cmd;

std::string sid;

std::string auf;

std::string aue;

std::string data;

std::string syncid;

};

在解析请求数据时，该结构体用来存放解析出的json元素数据。

|  |  |  |
| --- | --- | --- |
| **结构体参数** | 类型 | 说明 |
| cmd | 字符串 | 请求命令 |
| sid | 字符串 | 会话id |
| auf | 字符串 | 采样率 |
| aue | 字符串 | 语音是否压缩 |
| data | 字符串 | 请求数据 |
| syncid | 字符串 | 语音包序号 |

数据解析主要代码处理逻辑如下：

item是redis\_param结构体对象，接收解析出的数据；

**if**(document.HasMember("request"))

{

**const** rapidjson::Value& items = document["request"];

**if**(items.HasMember("cmd") && items["cmd"].IsNull()==**false**)

{

item->cmd = items["cmd"].GetString();

}

**if**(items.HasMember("syncid") && items["syncid"].IsNull()==**false**)

{

item->syncid = items["syncid"].GetString();

}

**if**(items.HasMember("auf") && items["auf"].IsNull()==**false**)

{

item->auf = items["auf"].GetString();

}

**if**(items.HasMember("aue") && items["aue"].IsNull()==**false**)

{

item->aue = items["aue"].GetString();

}

**if**(items.HasMember("data") && items["data"].IsNull()==**false**)

{

item->data = items["data"].GetString();

}

}

### 4.2.2 数据封装数据结构设计

返回给能力接口数据是xml形式，结构如下：

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

<results>

<sid></sid>

<recStatus></recStatus>

<engine\_name></engine\_name>

<result></result>

<bg></bg>

<ed></ed>

</results>

|  |  |  |
| --- | --- | --- |
| Xml参数 | 类型 | 说明 |
| sid | 字符串 | 会话id |
| recStatus | 字符串 | 识别状态 |
| engine\_name | 字符串 | 识别引擎地址 |
| result | 字符串 | 识别结果 |
| bg | 字符串 | 语音开始标识 |
| ed | 字符串 | 语音结束标识 |

关联结构数据：

该结构体存放引擎处理的识别信息，封装返回结果从该结构体中获取识别信息。

**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

**#ifdef** ENHANCED\_RESULT

**char** \* phone;//phoneme string

**char** \* segTime;//segTime string

**#endif**

ACand (){

text = 0;

next = 0;

score = startTime = endTime = 0;

phone = 0;

segTime = 0;

};

};

数据封装主要代码逻辑如下：

Value text1(kStringType);

std::string restmp = cand->text;

deleteAllMark(restmp," ");

text1.SetString(restmp.c\_str(), allocator);

doc.AddMember("text",text1,allocator);

Value phoneme1(kStringType);

phoneme1.SetString(cand->phone, allocator);

doc.AddMember("phoneme",phoneme1,allocator);

Value segtime1(kStringType);

segtime1.SetString(cand->segTime, allocator);

doc.AddMember("segtime",segtime1,allocator);

Value score1(kStringType);

score1.SetFloat(cand->score);

doc.AddMember("score",score1,allocator);

StringBuffer buffer;

PrettyWriter<StringBuffer> pretty\_writer(buffer);

doc.Accept(pretty\_writer);

resultXML += buffer.GetString();

### 4.2.3 流程控制数据结构设计

接收引擎识别结果数据结构：

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

{

**int** sessId;

**bool** isFirstRecog;

queue<rec\_t \* > mRecArray;

**int** res\_num;

} asr\_param\_t;

|  |  |  |
| --- | --- | --- |
| 结构体参数 | 类型 | 说明 |
| SessId | 整数 | 线程编号 |
| isFirstRecog | 布尔类型 | 会话开始标识 |
| mRecArray | 结构体队列 | 存放识别结果 |
| res\_num | 整数 | 识别结果数量 |

流程控制结构体：

该结构体参与到整个会话流程，从会话开始创建文件保存语音到发送语音到识别引擎，最后到会话结束释放资源都需要该结构体参与控制。

**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; //线程编号

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

} asr\_var\_t;

主要代码逻辑如下：

会话开始主要逻辑

**int** **processSsb**(asr\_var\_t &threadVar, asr\_var\_temp\_t &threadVarTemp,

**void** \*parameter, redis\_param &rparam, std::string &bitRate)

{

**int** count = 0;

//等待上一个会话sessId取未及时取走的结果

**while**(g\_ResultQueue[threadVar.sessId]->size()!=0 && count++<50)

{

usleep(100\*1000);

}

destroyData(g\_TaskQueue[threadVar.sessId]);

destroyData(g\_ResultQueue[threadVar.sessId]);

RedisProcesser \*redisProc = **new** RedisProcesser(RedisPasswd,RedisCluster,

redisKey,RedisPasswdOn,opt\_ListenPort,LocalIP);

redisProc->setSessionId(threadVar.sessId);

**if** (redisProc->GetSsbRequest(&rparam) != 0)

{

LOG\_ERROR("[Id:%d]get session faill",threadVar.sessId);

safe\_delete(redisProc);

PopThreadNum(rparam.sid);

**return** 1;

}

}

语音处理主要逻辑

int processPcm(asr\_var\_t &threadVar,redis\_param rparam,asr\_var\_temp\_t &threadVarTemp,Data\_t \*out)

{

if (!threadVar.isCheckHeader && threadVar.readLen >= 4) //第一包数据检测4字节数据头,判断语音格式

{

checkFirst4Byte(threadVar, threadVarTemp);

}

//保存语音逻辑

if ((opt\_SaveVoice == 1 || opt\_SaveVoice == 3) && (threadVar.fpRec != NULL))

{

if(threadVar.inviteHead.idx == 1)

fwrite(threadVar.recBuf+4, threadVar.readLen-4, 1, threadVar.fpRec);

else

fwrite(threadVar.recBuf, threadVar.readLen, 1, threadVar.fpRec);

}

if (opt\_SaveVoice == 2 || opt\_SaveVoice == 3)

{

if (NULL != threadVar.fpPcm)

{

if(threadVar.inviteHead.idx == 1)

{

LOG\_INFO("[Id:%d][sid:%s]write to file %d bytes.",

threadVar.sessId, threadVar.inviteHead.sid, threadVar.readLen-4);

wav\_write\_data(threadVar.fpPcm, threadVar.recBuf+4, threadVar.readLen-4);

}

else

{

LOG\_INFO("[Id:%d][sid:%s]write to file %d bytes.",

threadVar.sessId, threadVar.inviteHead.sid, threadVar.readLen);

wav\_write\_data(threadVar.fpPcm, threadVar.recBuf, threadVar.readLen);

}

}

}

if(opt\_sendToEngine == 1)

{

if(opt\_guding == 1)

{

sendToTbnr(threadVar,threadVar.recBuf,threadVar.readLen,true);

}

else

{

if(threadVar.inviteHead.idx == 1)

TBNR\_SendData((char\*)(threadVar.recBuf+4), threadVar.readLen-4,handleId[threadVar.sessId], TSR\_RAW\_16);

else

TBNR\_SendData((char\*)threadVar.recBuf, threadVar.readLen,handleId[threadVar.sessId], TSR\_RAW\_16); //发送语音到识别引擎

}

}

threadVarTemp.isSendData = true;

getAuwResult(threadVar, rparam.sid, out,false); //获取识别信息

threadVarTemp.conBytes += threadVar.readLen;

return 0;

}