

中科院声学所离线识别引擎

总控端概要设计说明

版 本 历 史

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

目录

[1.概述 3](#_Toc27418)

[1.1简介 3](#_Toc4290)

[1.2 读者对象 3](#_Toc13187)

[1.3术语定义 4](#_Toc18048)

[2.总体设计 5](#_Toc14461)

[2.1 总体需求 5](#_Toc19360)

[2.2 运行环境 6](#_Toc22805)

[2.2.1 软件环境 6](#_Toc5512)

[2.2.2 硬件环境 6](#_Toc17772)

[2.3系统总体结构设计 7](#_Toc2810)

[3.模块设计 9](#_Toc27600)

[3.1任务获取 9](#_Toc24894)

[3.2结果处理 11](#_Toc29654)

[3.3任务发送 13](#_Toc29525)

[3.4结果接收 15](#_Toc24372)

[4.配置及测试设计 16](#_Toc27632)

[4.1 配置文件设计 16](#_Toc26211)

[4.2 测试功能点 18](#_Toc8199)

[5.可靠性设计 19](#_Toc5968)

[6.扩展性设计 20](#_Toc10053)

[7.维护性设计 21](#_Toc11946)

[8.易用性设计 22](#_Toc26781)

# 1.概述

1

**1.1简介**

总控端取出任务之后，经过语音转码、任务信息存入数据库，分别把任务发往识别引擎、ITN引擎、标点引擎、聚类引擎、叠音引擎，最终得到各个模块的中间结果文件。

## 1.2 读者对象

本文档的读者对象为离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。

本说明给出离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：获取redis里任务依次发往各服务端；

性能需求：正常处理不崩溃；

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 |
| Mysql | 5.7.6 |
| 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系统总体结构设计

总控端部分主要包括一个任务接收线程，一个结果接收线程，一个任务发送线程，一个结果处理线程组成。



任务获取线程：阻塞的从Redis获取待处理任务，经过语音转码，任务信息记录数据库等操作，存入任务队列。

任务处理线程：从结果队列中取出服务端返回处理结果，根据服务端返回的处理码和是否走到最后一个服务端做相应处理。

任务发送线程:从任务队列中取已经转码好的语音，发送到识别服务端。

# 3.模块设计

3

总控端完成的功能有：从Redis任务列表中获取任务，控制语音任务在各个引擎服务端之间按一定顺序进行处理，记录任务处理状态与结果等信息到数据库。总控端总共有“3N+2”个线程，包括：1个主函数线程，1个原始任务获取线程，3N个关于任务发送、结果获取、结果处理的线程，N表示系统中开启的引擎服务端端的个数。

原始任务获取线程主要负责从Redis任务队列中获取语音任务、插入语音任务到数据库、对原始语音进行转码、构造任务结构体TaskInfo压入第一个服务端的任务队列中。

## 3.1任务获取

原始任务获取线程对应线程函数为GetTaskThread(),具体处理流程如下：



主要函数介绍

int beginProcess(string serialNum,string audioName,string audioUrl,string XMLfilePath,SRVMANAGE\* firstServer,int in\_channels,string stereo\_on,string ResultType); //语音任务进行转码，获取语音时长，任务压入第一个服务端任务队列

int insertTask(string serialNum,string audioName,string audioUrl,string XMLfilePath,string getResult); //往数据表task\_info中插入任务

std::map<std::string, std::string> getValInTaskInfo(string serialNum, string keyWord); //根据serialNum, 获取数据库中的值

bool audio2pcm(TaskInfo \*& aNewTask,string stereo\_on); //将语音转码为8k\_16bit\_pcm格式

float getAudioTimeLen(string audioFullName); // 获取语音时长，单位：秒

int setAudioTimeLen(string serialNum,float audioTimeLen); //设置语音时长

## 3.2结果处理

结果处理线程从当前服务端的结果队列中循环获取结果，根据任务是否被成功处理、是否还有下一个引擎服务端来进行不同的处理。当任务被成功处理且当前服务端不是最后一个服务端时，此任务将被压入当前服务端的下一个服务端的任务队列，以此实现任务在各个服务端有序进行处理。

结果处理线程对应线程函数为ResultProcThread(),流程图如下：



主要函数介绍

int handleXMLfile(string xmlfile,string &XMLResult,string oriAudioFullname); //处理xml文件

void removeFile(string file); //删除文件

void removeResultFile(string resultfilepath, string basename,string ResultType); //删除中间结果文件及转码后语音

int setEndTime(string serialNum,string voiceStatus,string failCause, string XMLResult); //写语音处理结果到数据库

int setFailCause(string serialNum, string failCause); //记录失败原因

## 3.3任务发送

任务发送线程主要是从当前服务端的任务列表获取任务后，使用当前服务端的通信套接字，发送给当前服务端。

任务发送线程对应线程函数为TaskSendThread(),流程图如下：



图6 TaskSendThread()函数流程图

## 3.4结果接收

结果接收线程使用当前服务端的套接字，接收当前服务端的分析结果，并将结果放入当前服务端的结果队列。

结果接收线程对应的线程函数为ResultReceiveThread(),流程图如下：



# 4.配置及测试设计

4

## 4.1 配置文件设计

设置配置文件，方便使用不同控制逻辑，支持不同的功能及结果输出，总控端配置文件实现如下目的：

* 通过参数控制支持特殊功能
* 检测异常数据
* 日志输出
* 引擎资源加载
* 数据库连接
* 服务地址配置

配置文件主要参数如下表：

|  |  |  |
| --- | --- | --- |
| 参数 | 取值 | 说明 |
| MysqlServerIP | 字符串 | 数据库IP |
| MysqlServerPort | 字符串 | 数据库端口 |
| MysqlDatabase | 字符串 | 数据库库名 |
| MysqlUser | 字符串 | 数据库用户名 |
| MysqlEncrypt | 字符串 | 数据库密码是否加密 |
| MysqlPassword | 字符串 | 数据库密码 |
| processorId | 字符串 | 引擎服务器标识 |
| redisCluster | false:true | Redis是否集群 |
| redisIp | 字符串 | Redis地址 |
| redisPort | 字符串 | Redis端口 |
| redisIsPass | false:true | Redis单机是否有密码 |
| redisPass | 字符串 | Redis密码 |
| redisTaskList | 范围0－1 | Redis队列名 |
| xmlPathPostfix | 范围0－2 | 存放xml结果目录后缀 |
| DeleteTempResult | false:true | 是否删除中间结果文件 |
| DeleteOriginalWav | false:true | 是否删除原始语音 |
| maxHandleThread | int | 并行处理语音最大线数 |
| checkPeriod | 字符串 | 超时检查频率 |
| taskCtrlShell | 字符串 | 文件同步脚本 |
| wavFormat | int | 语音格式 |
| logEvents | 范围0－8 | 日志级别 |
| logMaxLen | 100 | 单个日志文件的最大长度 |

## 4.2 测试功能点

根据设计需求，对服务必须满足的功能点进行验证测试。

|  |  |
| --- | --- |
| 测试功能 | 期望结果 |
| 服务启动验证 | 正确配置服务参数，启动服务，CPU、内存满足情况下，服务能够正常启动。 |
| 单声道语音 | 参数传递正确情况下，能正确转码。 |
| 双声道语音 | 参数传递正确情况下，能正确转码。 |
| 不删除原始语音 | 配置不删除原始语音，不删除原始语音 |
| 删除原始语音 | 配置删除原始语音，删除原始语音 |
| 不删除中间结果文件 | 配置不删除中间结果文件，不删除中间结果文件 |
| 删除中间结果文件 | 配置删除中间结果文件，删除中间结果文件 |
| 日志级别 | 调整不同的日志级别，日志输出内容不同 |
| 日志文件大小 | 调整日志文件大小，保存日志文件大小不同 |

# 5.可靠性设计

5

总控端部分设计考虑了实际应用场景的复杂性和输入的多样性的情况，内部加入了大部分实际情况中会出现的异常情况的处理方案，socket连接异常断开重连、redis连接异常断开重连、数据库操作异常重试、文件完整性检查等异常事件的处理；并且整套系统测试按照严格的7\*24小时压力测试标准进行测试，同时也针对所有异常数据进行异常情况测试，保证引擎的稳定性和可靠性。

# 6.扩展性设计

6

总控端控制语音任务在各个引擎服务端之间按一定顺序进行处理，3N个关于任务发送、结果获取、结果处理的线程，N表示系统中开启的引擎服务端端的个数，可方便加入新的服务端模块，只需增加相关服务端配置，无需修改总控端代码。系统本身遵循模块化设计，可以随时添加定制化功能，满足不同应用场景的需要。

# 7.维护性设计

7

离线系统为了方便后期的引擎维护，采用标准c/c++函数同时在较低编译器上进行编译。

总控端可配置cfg.cfg文件部分参数，保证了引擎的可配性和可维护性。常见参数如下：

#是否删除中间结果

DeleteTempResult=false

#是否删除原始语音

DeleteOriginalWav=false

#日志级别

logEvents=8

# 8.易用性设计

8

系统安装部署时有一键安装部署脚本，只需要执行脚本即可安装引擎转写系统及所需的中间件（如ffmpeg、sox、redis等），不需要安装人员逐个安装，减少操作量。系统启停均有对应的脚本，实现自动启动、停止服务。