

中科院声学所离线识别

数据结构设计说明

目录

[1.概述 5](#_Toc18461)

[1.1简介 5](#_Toc16519)

[1.2 读者对象 5](#_Toc6011)

[3. 数据结构设计 6](#_Toc10681)

[3.1任务消息：TaskInfo 6](#_Toc26273)

[3.2结果消息：ResultInfo 7](#_Toc27183)

[3.3 识别模块结构体、类定义 8](#_Toc12580)

版 本 历 史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本/状态 | 作者 | 参与者 | 起止日期 | 备注 |
| V1.0 | 王超锋 |  | 20211102 |  |
|  |  |  |  |  |
|  |  |  |  |  |

# 1.概述

1

1.1简介

识别模块主要是对语音进行连续语音识别也即全文转写，得到连续语音识别的结果（\*\_sent.txt）。

## 1.2 读者对象

本文档是对离线工程部分的内部代码流程和相关技术点的整体说明，因此读者最好满足如下几点条件后再读此文档：

* 有一定的C和C++语言基础

# 3. 数据结构设计

3

本节主要介绍protobuf消息结构设计和识别模块结构体。

总控端和各服务端之间通信的消息结构的封装采用Google Protocol Buffer( 简称 Protobuf)，类TaskInfo用于表示语音任务，类ResultInfo用于表示处理结果。

### **3.1任务消息：TaskInfo**

message TaskInfo

{

//音频信息

required string serialNum = 1;

required string audioname=2;

required string audiourl = 3;

required int32 channels=4;

required string language=5;

optional string keywordlist = 6;

optional string resultfilePath = 7 ;

optional string xmlfilePath=8;

}

说明：

* serialNum：音频文件的唯一性标识。
* audioname：音频文件的名称，例如：1.V3。
* audiourl ：音频的存放目录，例如：/home/thinkit/tempvoice

根据2,3的拼接得到语音文件的绝对路径。

* channels：音频的声道信息，0：单声道；1:双声道。
* language：音频语种信息，zhn：中文，eng：英语 .......。

若未设置，默认为zhn。

* keywordlist ：关键词列表的绝对路径。

若未设置，则默认使用XML服务端目录下的关键词列表。

* resultfilePath ：结果文本文件存放的绝对目录。

若未设置，则默认与xmlfilePath相同。

* xmlfilePath:最终的xml结果文件存放的绝对路径。

请务必给定该路径，否则任务将处理失败。

总控端从数据库中获取每条语音任务的详细信息后，都将对应生成一个个TaskInfo。当语音任务需要发送给某服务端前，再将TaskInfo序列化为一个字符串，发送给服务端进行处理。

### **3.2结果消息：ResultInfo**

message ResultInfo

{

required string serialNum = 1;

required int32 errorCode = 2;

optional bytes audioname=3;

optional bytes xmlfilePath = 4;

}

说明：

* serialNum：音频文件的唯一性标识。与TaskInfo中serialNum对应。
* errorCode：结果错误码。 0表示正常返回，-1表示分析过程有错误
* audioname：音频文件的名称。与TaskInfo中audioname对应。
* xmlfilePath: xml结果存放路径。与TaskInfo中xmlfilePath对应。

各服务端每处理完一个TaskInfo任务后，将相应构建一个ResultInfo，当需要将结果反馈给总控端前，将ResultInfo序列化为一个字符串，发送给总控端进行后续处理。

### 3.3 识别模块结构体、类定义

本节主要整理识别模块结构体定义及类。

//socket相关

class SocketManagement

{

public:

SocketManagement();

~SocketManagement();

bool InitSocket(string ipaddr,int ports); //初始化socket

bool WaitForClient(); //等待客户端连接

void ReSetSocketState() //重置socket连接状态

{

EnterCriticalSection(&cs\_socket);

bHasReConnet = false;

LeaveCriticalSection(&cs\_socket);

}

SOCKET& GetConnSocket()

{

return m\_connSk;

}

private:

string m\_IpAddr;

int m\_ports;

CRITICAL\_SECTION cs\_socket;

SOCKET m\_connSk;

SOCKET m\_serverSk;

bool bHasReConnet;

};

//线程函数传参

struct ThreadParam

{

int sessId;

};

//将识别结果有序排列

struct SortedResult

{

char\* segResult;

SortedResult\* sortedRstNext;

SortedResult()

{

segResult=NULL;

sortedRstNext = NULL;

}

};