**系统架构与实现**

目录

[1 系统整体框架 1](#_Toc454807266)

[1.1 系统框架图 1](#_Toc454807267)

[1.2 手机客户端代码 1](#_Toc454807268)

[1.3 服务器端php代码 2](#_Toc454807269)

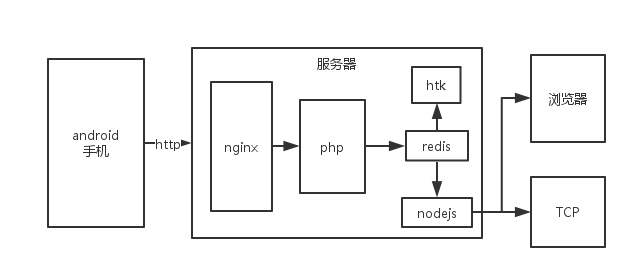
[1.3 HTK代码修改 2](#_Toc454807270)

[1.3 nodejs代码 3](#_Toc454807271)

# 1 系统整体框架

## 1.1 系统框架图

系统框架图如下：



由图中可以看出，系统由android客户端、和服务器构成服务器由nginx+php+htk+redis 和nodejs构成

## 1.2 手机客户端代码

打开eclipse并打开工程ARSNew ，工程代码存放在”语音识别\代码\Android\work\ARSNew”中：其中定义的主要类为RecordActivity 存放在”语音识别\代码\Android\work\ARSNew\src\com\example\OnlineRecRecordActivity.java”中

**public** **class** RecordActivity **extends** Activity {

**private** **static** ClsOscilloscope *clsOscilloscope*=**new** ClsOscilloscope(); //用来绘制波形图

SurfaceView sfv;Paint mPaint;

**private** **static** Button *rec\_or\_stop*;// 开始录音

**private** **static** Button *history*;//录音记录

AudioRecord audioRecord\_;//录音类

**static** AudioTrack *audioTrack\_*;

**static** **final** **int** *frequency\_* = 16000;//分辨率

**static** **final** **int** *channelConfiguration\_* = AudioFormat.~~CHANNEL\_CONFIGURATION\_MONO~~;

**static** **final** **int** *audioEncoding\_* = AudioFormat.*ENCODING\_PCM\_16BIT*;

**private** **static** **final** String *HOST* = "192.168.43.215"; //默认服务器ip可通过输入框修改

**private** **static** **final** **int** *PORT* = 3333;//服务器端口

}

手机端通过该类来上传录音和录制音频工作。

## 1.3 服务器端php代码

系统通过php来接受上传的音频文件代码如下，该文件存放在htk容器内部以下目录中usr/share/nginx/html/upload.php：。

<?php  
$redis = new Redis();  
$redis->pconnect('127.0.0.1', 6379);//链接redis  
  
$name = substr(basename( $\_FILES['file']['name']),0,-4);  
$target\_path = "./uploads/";  
$target\_path = $target\_path . basename( $\_FILES['file']['name']);  
  
$plp\_path = "/home/vrgroup/Desktop/htk/plp/";  
$plp\_name = $plp\_path . $name. ".plp";  
  
if(move\_uploaded\_file($\_FILES['file']['tmp\_name'], $target\_path)) {  
 exec("/usr/local/bin/HCopy -C /home/vrgroup/Desktop/htk/model/config-plp $target\_path $plp\_name");//生成plp特征文件  
 $redis->lPush('filesQueue',$plp\_name);  
 $redis->blPop($name,0);//等待识别结果  
 $getcmd="iconv -f GBK -t utf-8 /home/vrgroup/Desktop/htk/model/out | awk '/$name/,/SENT\_END/'|awk '{if (NF==4){ printf $3}}'";//返回识别结果  
 exec($getcmd,$result);  
 if(!empty($result[0]))echo substr($result[0],11,-9);  
} else{  
 echo "There was an error uploading the file, please try again!" . $\_FILES['uploadedfile']['error'];  
}  
?>

## 1.3 HTK代码修改

通过修改HDecode.c源码并另存为RedisToHDecode.c将redis功能绑定如htk工具包中。该文件存放在htk容器内部的”/home/vrgroup/Desktop/htk/HTKLVRec”目录中，主要修改代码如下：

char name[512];  
 redisContext \*c;  
 redisReply \*reply;  
 const char \*hostname = "127.0.0.1";  
 int port = 6379;  
  
 struct timeval timeout = { 1, 500000 };   
 c = redisConnectWithTimeout(hostname, port, timeout);  
 if (c == NULL || c->err) {  
 if (c) {  
 printf("Connection error: %s\n", c->errstr);  
 redisFree(c);  
 } else {  
 printf("Connection error: can't allocate redis context\n");  
 }  
 exit(1);  
 }  
  
将main函数中DoRecognition (dec, datafn)函数调用替换为如下代码  
 while (1){  
 printf("BRPOP filesQueue 0\n");  
 printf("block for redis filesQueue. to wakeup it use 'LPUSH filesQueue plpfile' --plpfile was the absolute path of php file to be recognitioned\n");  
 reply = redisCommand(c,"BRPOP filesQueue 0");  
 printf("GET file: %s\n", reply->element[1]->str);  
 DoRecognition (dec, reply->element[1]->str);  
  
 s=reply->element[1]->str;  
 while(\*s!='.'&&\*s!='\0')s++;  
 \*s='\0';  
 sprintf(name,"%s",reply->element[1]->str+30);  
 printf("%s",name);  
  
 redisCommand(c,"LPUSH %s %s",name,"1");  
 freeReplyObject(reply);  
 }

## 1.3 nodejs代码

为了实现识别结果的多平台分发，系统采用nodejs来完成该任务，为了让nodejs支持浏览器，TCP客户端等方式分发数据，编写如下脚本来完成这些功能，该文件存放在nodejs容器中”/home/admin”目录中。

var app = require('express')();  
var http = require('http').Server(app);  
var io = require('socket.io')(http);  
var redis = require('redis');  
var net = require('net');  
  
function getIPAdress(){   
 var interfaces = require('os').networkInterfaces();   
 for(var devName in interfaces){   
 var iface = interfaces[devName];   
 for(var i=0;i<iface.length;i++){   
 var alias = iface[i];   
 if(alias.family === 'IPv4' && alias.address !== '127.0.0.1' && !alias.internal){   
 return alias.address;   
 }   
 }   
 }   
}   
app.get('/', function(req, res){  
 var sp="<script src='http://";  
 sp=sp + getIPAdress();  
 sp=sp + ":3003/socket.io/socket.io.js'></script>";  
 sp=sp + "<script> var socket = io('http://";  
 sp=sp + getIPAdress();  
 sp=sp + ":3003');socket.on('message', function (data) {document.getElementById('demo').innerHTML=data;});</script>"   
 res.send(sp+'<h1 id="demo">Welcome Realtime Server Hucd</h1>');  
});  
  
io.on('connection', function(socket){  
 console.log('a user connected');  
 //监听用户发布聊天内容  
 socket.on('message', function(obj){  
 //向所有客户端广播发布的消息  
 io.emit('message', obj);  
 console.log(obj);  
 });  
   
});  
  
http.listen(3003, function(){  
 console.log('listening on \*:3003');  
});  
//------------------------------tcp socket---------------------------------------------------------  
clientList=[];  
var tcp\_server = net.createServer(function (socket) {  
 console.log('客户端: ' + "连接成功");  
 clientList.push(socket);  
 socket.on('data', function (data) {  
 console.log('DATA '+ data);  
 //socket.write(data);  
 });  
  
 socket.on('close', function (data) {  
 console.log('客户端: ' + "断开连接");  
 });  
  
 socket.on('error', function (exc) {  
 console.log("ignoring exception: " + exc);  
 });  
});  
function broadcast(message) {  
 var cleanup = []  
 for(var i=0;i<clientList.length;i+=1) {  
 if(clientList[i].writable) { // 先检查 sockets 是否可写  
 clientList[i].write(message)  
 } else {  
 cleanup.push(clientList[i]) // 如果不可写，收集起来销毁。销毁之前要 Socket.destroy() 用 API 的方法销毁。  
 clientList[i].destroy()  
 }  
 } //Remove dead Nodes out of write loop to avoid trashing loop index  
 for(i=0;i<cleanup.length;i+=1) {  
 clientList.splice(clientList.indexOf(cleanup[i]), 1)  
 }  
}  
tcp\_server.listen(3004);  
  
var redisclient = redis.createClient();  
redisclient.on('connect',function(){  
 redisclient.set('author', 'testauthor', redis.print);  
 redisclient.get('author', redis.print);  
 redisclient.get('hello', redis.print);  
 redisclient.subscribe("root");  
});  
redisclient.on("message", function (channel, message) {  
 io.emit('message', message);  
 broadcast(message);  
 console.log(channel + ": " + message);  
});