@[toc]

## 一、演示视频

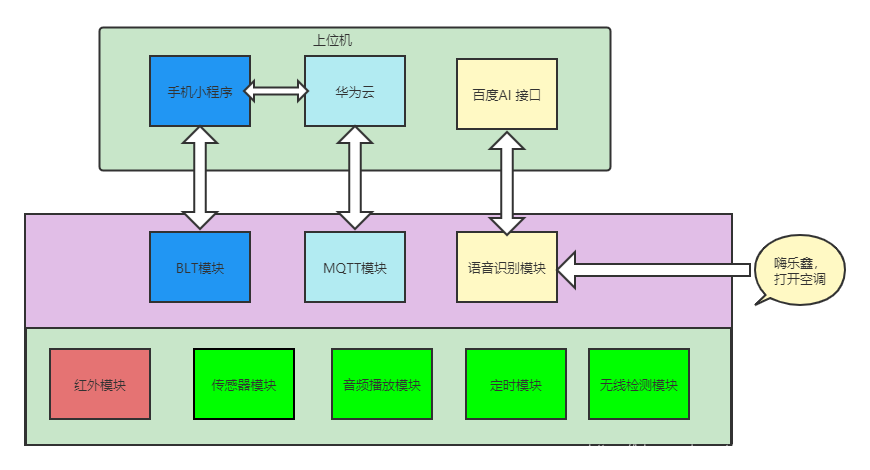
离线版地址：<https://gitee.com/killerp/off_asr> 语音识别优化版地址：<https://gitee.com/killerp/smart_control>

本项目基于esp32a1s模组，设计了一个遥控器，除了实现基本的红外遥控功能，还利用ESP32芯片具备的AI能力，WIFI及蓝牙功能，实现多种方式的输入输出控制，使人们能通过语音，手机远程进行红外遥控。同时又加入温度传感器，并支持将温度数据上传云端，使人们能随时随地查看、分析数据。 项目演示视频如下：

[video(video-19gWGs4l-1617149447766)(type-bilibili)(url-https://player.bilibili.com/player.html?aid=972368801)(image-https://ss.csdn.net/p?http://i0.hdslb.com/bfs/archive/03eaf981b73823d3c328e13cfa1250c6e418c0ca.png)(title-基于esp32语音助手 语音红外遥控)]

## 二、程序框架

项目软件分为上位机端和嵌入式终端，程序框架如图：

位于底层的模块可为其上层的模块提供接口 

## 三、硬件设计

## 四、模块介绍

### 1、语音识别模块

语言识别模块有两种实现方式，一种是离线识别，一种是针对离线版的优化。（目前离线版不再维护）

#### 离线语音识别

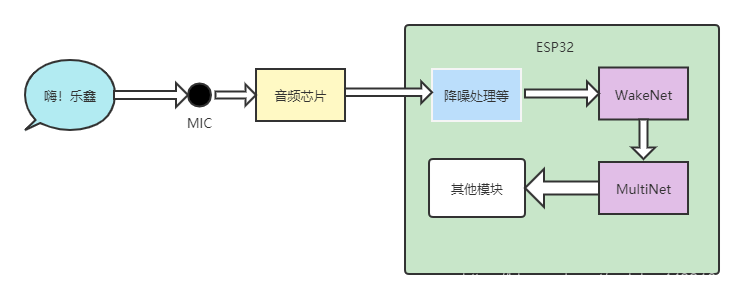
离线语音的实现依赖于乐鑫提供ESP-Skainet库，该库实现了语音唤醒，语言识别功能，可支持100条自定义的语言识别命令。

ESP-Skainet 由两部分组成：

1，唤醒词模型 [WakeNet](https://github.com/espressif/esp-sr/tree/master/wake_word_engine/README_cn.md)：其实现了语音唤醒的功能，目前普通用户能使用限定的唤醒词，如：嗨乐鑫。模型由乐鑫训练。

2，命令词识别模型 [MultiNet](https://github.com/espressif/esp-sr/tree/master/speech_command_recognition/README_cn.md)：其实现语音识别功能，将需要识别的语音命令写入程序，MultiNet就能在程序运行中进行识别，目前支持100条命令词。

在整个程序运行过程中，WakeNet和MultiNet两个模型以进程的形式运行，实时地读取麦克风的音频输入，将音频经过降噪处理等，输入模型，输出结果。其程序运行如图所示：

 以下代码在main()中运行，该代码初始化了wakenet，multinet 模型，并创建一个音频流通道（ESP-ADF中的概念），该通道的作用即上图中箭头所示。main()以一个RTOS的任务运行，在循环中实时读取通道的音频，输入模型识别。

命令词识别成功后，在asr\_multinet\_control函数中编写对命令词的响应动作，如调用红外模块提供的接口打开空调等。

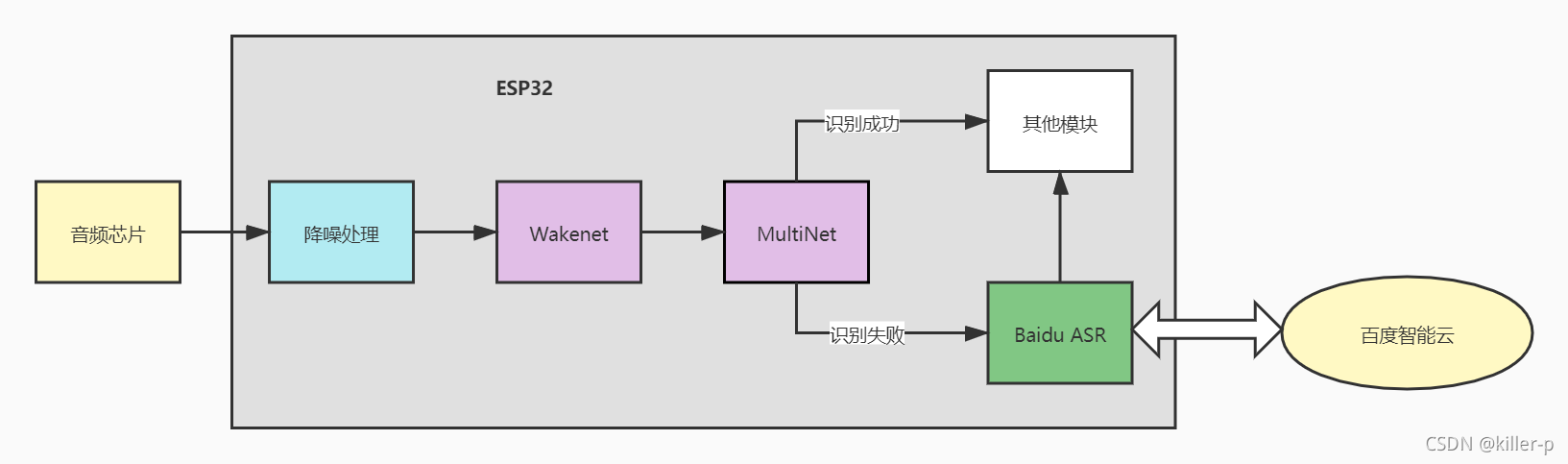
ESP\_LOGI(TAG, "Initialize SR wn handle");  
 esp\_wn\_iface\_t \*wakenet; //唤醒模型  
 model\_coeff\_getter\_t \*model\_coeff\_getter; //神经网络系数获取  
 model\_iface\_data\_t \*model\_wn\_data; //识别模型的数据  
 const esp\_mn\_iface\_t \*multinet = &MULTINET\_MODEL; //识别模型  
  
 // Initialize wakeNet model data  
 get\_wakenet\_iface(&wakenet); //初始化唤醒模型  
 get\_wakenet\_coeff(&model\_coeff\_getter); //获取系数  
 model\_wn\_data = wakenet->create(model\_coeff\_getter, DET\_MODE\_90); //创建唤醒模型，设置灵敏度90%  
  
 int wn\_num = wakenet->get\_word\_num(model\_wn\_data); //唤醒词数量  
 for (int i = 1; i <= wn\_num; i++)  
 {  
 char \*name = wakenet->get\_word\_name(model\_wn\_data, i); //唤醒词文本  
 ESP\_LOGI(TAG, "keywords: %s (index = %d)", name, i);  
 }  
 float wn\_threshold = wakenet->get\_det\_threshold(model\_wn\_data, 1); //获取唤醒阈值  
 int wn\_sample\_rate = wakenet->get\_samp\_rate(model\_wn\_data); //唤醒词采样率16k  
 int audio\_wn\_chunksize = wakenet->get\_samp\_chunksize(model\_wn\_data); //内存块大小  
 ESP\_LOGI(TAG, "keywords\_num = %d, threshold = %f, sample\_rate = %d, chunksize = %d, sizeof\_uint16 = %d", wn\_num, wn\_threshold, wn\_sample\_rate, audio\_wn\_chunksize, sizeof(int16\_t));  
  
 model\_iface\_data\_t \*model\_mn\_data = multinet->create(&MULTINET\_COEFF, 4000); //语音识别时间，single模式下最大5s  
 int audio\_mn\_chunksize = multinet->get\_samp\_chunksize(model\_mn\_data); //识别内存块  
 int mn\_num = multinet->get\_samp\_chunknum(model\_mn\_data); //唤醒词数量  
 int mn\_sample\_rate = multinet->get\_samp\_rate(model\_mn\_data); //采样率16k  
 ESP\_LOGI(TAG, "keywords\_num = %d , sample\_rate = %d, chunksize = %d, sizeof\_uint16 = %d", mn\_num, mn\_sample\_rate, audio\_mn\_chunksize, sizeof(int16\_t));  
  
 //选择所需的较大的内存块  
 int size = audio\_wn\_chunksize;  
 if (audio\_mn\_chunksize > audio\_wn\_chunksize)  
 {  
 size = audio\_mn\_chunksize;  
 }  
 int16\_t \*buffer = (int16\_t \*)malloc(size \* sizeof(short)); //buffer用于缓存经过流水线处理的音频  
  
 /\*[ac101]-->i2s\_stream-->filter-->raw-->[SR]\*/  
 audio\_pipeline\_handle\_t pipeline; //音频输入流水线  
 audio\_element\_handle\_t i2s\_stream\_reader, filter, raw\_read; //流水线车间  
  
 bool enable\_wn = true; //唤醒使能  
 uint32\_t mn\_count = 0;  
  
 ESP\_LOGI(TAG, "[ 1 ] Start codec chip");  
  
 ESP\_LOGI(TAG, "[ 2.0 ] Create audio pipeline for recording");  
 //流水线初始化  
 audio\_pipeline\_cfg\_t pipeline\_cfg = DEFAULT\_AUDIO\_PIPELINE\_CONFIG();  
 pipeline = audio\_pipeline\_init(&pipeline\_cfg);  
 mem\_assert(pipeline);  
  
 //i2s初始化，用于与ac101通信  
 ESP\_LOGI(TAG, "[ 2.1 ] Create i2s stream to read audio data from codec chip");  
 i2s\_stream\_cfg\_t i2s\_cfg = I2S\_STREAM\_CFG\_DEFAULT();  
 i2s\_cfg.i2s\_config.sample\_rate = 48000;  
 i2s\_cfg.type = AUDIO\_STREAM\_READER; //输入流  
 i2s\_stream\_reader = i2s\_stream\_init(&i2s\_cfg);  
  
 ESP\_LOGI(TAG, "[ 2.2 ] Create filter to resample audio data");  
 //滤波器初始化，将源采样率变为16k  
 rsp\_filter\_cfg\_t rsp\_cfg = DEFAULT\_RESAMPLE\_FILTER\_CONFIG();  
 rsp\_cfg.src\_rate = 48000;  
 rsp\_cfg.src\_ch = 2;  
 rsp\_cfg.dest\_rate = 16000;  
 rsp\_cfg.dest\_ch = 1;  
 filter = rsp\_filter\_init(&rsp\_cfg);  
  
 //raw初始化，缓存经过处理的音频数据  
 ESP\_LOGI(TAG, "[ 2.3 ] Create raw to receive data");  
 raw\_stream\_cfg\_t raw\_cfg = {  
 .out\_rb\_size = 8 \* 1024,  
 .type = AUDIO\_STREAM\_READER, //输入流  
 };  
 raw\_read = raw\_stream\_init(&raw\_cfg);  
  
 //将各个车间流连接到流水线  
 ESP\_LOGI(TAG, "[ 3 ] Register all elements to audio pipeline");  
 audio\_pipeline\_register(pipeline, i2s\_stream\_reader, "i2s");  
 audio\_pipeline\_register(pipeline, raw\_read, "raw");  
 audio\_pipeline\_register(pipeline, filter, "filter");  
 ESP\_LOGI(TAG, "[ 4 ] Link elements together [codec\_chip]-->i2s\_stream-->filter-->raw-->[SR]");  
 const char \*link\_tag[3] = {"i2s", "filter", "raw"};  
 audio\_pipeline\_link(pipeline, &link\_tag[0], 3);  
  
 //运行流水线  
 ESP\_LOGI(TAG, "[ 5 ] waiting to be awake");  
 audio\_pipeline\_run(pipeline);  
  
 while (1)  
 {  
 //读取raw的音频到buffer  
 raw\_stream\_read(raw\_read, (char \*)buffer, size \* sizeof(short));  
 if (enable\_wn)  
 {  
 //检测buffer是否有唤醒词  
 if (wakenet->detect(model\_wn\_data, (int16\_t \*)buffer) == WAKE\_UP)  
 {  
  
 LED\_ON;  
 ESP\_LOGI(TAG, "wake up");  
 enable\_wn = false;  
 }  
 }  
 else  
 {  
 mn\_count++;  
 //检测buffer中是否有命令词  
 int commit\_id = multinet->detect(model\_mn\_data, buffer);  
 //进入命令词控制函数  
 if (asr\_multinet\_control(commit\_id) == ESP\_OK)  
 {  
 LED\_OFF;  
 enable\_wn = true;  
 mn\_count = 0;  
 }  
 if (mn\_count == mn\_num)  
 {  
 ESP\_LOGI(TAG, "stop multinet");  
 LED\_OFF;  
 enable\_wn = true;  
 mn\_count = 0;  
 }  
 }  
 }  
  
 ESP\_LOGI(TAG, "[ 6 ] Stop audio\_pipeline");  
  
 audio\_pipeline\_stop(pipeline);  
 audio\_pipeline\_wait\_for\_stop(pipeline);  
 audio\_pipeline\_terminate(pipeline);  
  
 /\* Terminate the pipeline before removing the listener \*/  
 audio\_pipeline\_remove\_listener(pipeline);  
  
 audio\_pipeline\_unregister(pipeline, raw\_read);  
 audio\_pipeline\_unregister(pipeline, i2s\_stream\_reader);  
 audio\_pipeline\_unregister(pipeline, filter);  
  
 /\* Release all resources \*/  
 audio\_pipeline\_deinit(pipeline);  
 audio\_element\_deinit(raw\_read);  
 audio\_element\_deinit(i2s\_stream\_reader);  
 audio\_element\_deinit(filter);  
  
 ESP\_LOGI(TAG, "[ 7 ] Destroy model");  
 wakenet->destroy(model\_wn\_data);  
 model\_wn\_data = NULL;  
 free(buffer);  
 buffer = NULL;

进入项目的配置程序

idf.py menuconfig>ESP Speech Recognition>Add speech commands

即可添加命令词

#### 优化语音识别

相对于离线版，优化版除了具有本地的语音识别之外，还加入了百度的语音识别API，当本地的语音识别未能匹配到命令时，将音频数据发送到百度语音识别接口，并对返回的文本结果进行解析，逻辑如图（百度智能云语音识别返回结果是utf-8编码的字符串，所以代码编辑器中最好设置utf-8编码.）  **语音模块的工作流程如下：**

麦克风输入的音频经过音频解码芯片，通过IIS进入到ESP32内存，经过降噪处理输入WakeNet检测是否存在唤醒词，若有唤醒词则进入语音识别，将音频输入MultiNet匹配命令词，同时缓存音频数据。

若匹配成功则会执行命令内容，若经过设置的时间（eg 4s）匹配失败，则将缓存的音频数据通过Baidu ASR发送到百度智能云并读取返回的文本结果，对结果解析出命令内容执行。

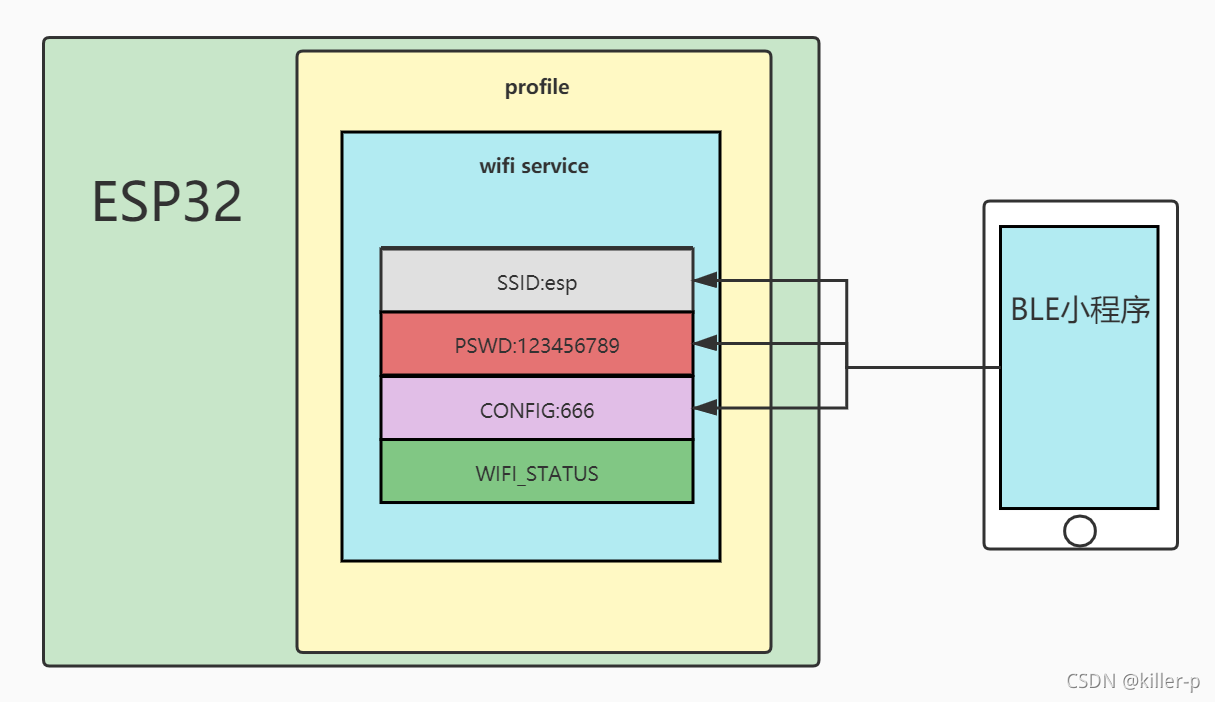
### 2、BLE模块

ESP32支持蓝牙双模，目前仅使用蓝牙进行配网（即给esp32发送wifi名称和密码，使esp32连接到wifi）。所以使用ble更合适。

BLE（低功耗蓝牙），适合数据量较小的场合，ESP32支持完整的BLE协议栈，以ESP32作为BLE从机，发送广播、建立GATT Server并等待客户端连接。客户端以微信小程序BLE为例。小程序与ESP32蓝牙通信过程如下图：

ESP32建立一张profile、并创建一个wifi service、用于配置ESP32的wifi。小程序可向wii service 的相应属性中写入数据，来配置ESP32的wifi。

如图，向SSID、PSWD、CONFIG分别写入下值，ESP32会自动连接到路由器esp



在这里插入图片描述

### 3、MQTT模块

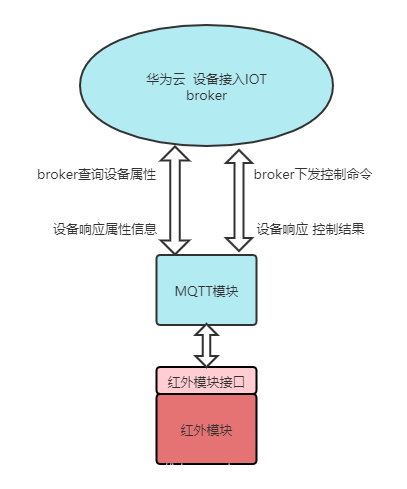
该模块使用mqtt\_client库实现mqtt客户端进程的登陆，订阅，发布等功能。MQTT broker则使用华为云的设备接入IOT平台。

华为云平台为每个设备定义了以下多个主题，用于最基本的mqtt消息上传和下发，本模块参考了[设备接入Iota->API参考->设备侧mqtt接口参考](https://support.huaweicloud.com/api-iothub/iot_06_v5_3011.html) **上传** - 设备消息上报：设备无法按照产品模型中定义的属性格式进行数据上报时，将设备的自定义数据通过设备消息上报接口上报给平台，平台将设备上报的消息转发给应用服务器或华为云其他云服务上进行存储和处理。

* 设备属性上报：用于设备按产品模型中定义的格式将属性数据上报给平台。
* 网关批量属性上报：用于网关设备将多个设备的属性数据一次性上报给平台。
* 设备事件上报：用于设备按产品模型中定义的格式将事件数据上报给平台。

**下发** - 平台消息下发：用于平台下发自定义格式的数据给设备。

* 平台设置设备属性：设备的产品模型中定义了平台可向设备设置的属性，应用服务器可通过属性设置的方式修改指定设备的属性值。
* 平台查询设备属性：应用服务器通过属性查询的方式，实时查询指定设备的属性数据。
* 平台命令下发：应用服务器按产品模型中定义的命令格式下发控制命令给设备。
* 平台事件下发：应用服务器按产品模型中定义的事件格式下发事件给设备。

目前实现的软件框架如图所示：  #### 3.1、命令下发 该功能实现参考[华为云平台命令下发文档](https://console.huaweicloud.com/iotdm/?region=cn-north-4&locale=zh-cn#/dm-portal/device/all-device/device-detail/60b9c9383744a602a5cb9bf3_smart_control_01?appId=7e7b135b6d664d92be89ed696290524d&productId=60b9c9383744a602a5cb9bf3) 华为云平台下发的mqtt命令topic格式固定为：

$oc/devices/{device\_id}/sys/commands/request\_id={request\_id}

数据段中的json数据是命令的具体内容（由用户定义），如空调的控制命令如下：

{"paras":{"ac\_power":1,"ac\_temp":27,"ac\_wind\_speed":2,"ac\_mode":0},"service\_id":"ac\_control","command\_name":"ac\_control"}

#### 3.2、设备属性上报

设备端可主动上报设备的属性（空调的状态，温度），mqtt消息发布的topic为：

Topic: $oc/devices/{device\_id}/sys/properties/report

并将设备属性以json格式放在数据段中。

#### 3.3、平台查询设备属性

平台通过给该设备主题发布消息，通知设备端上报属性信息，主题格式为：

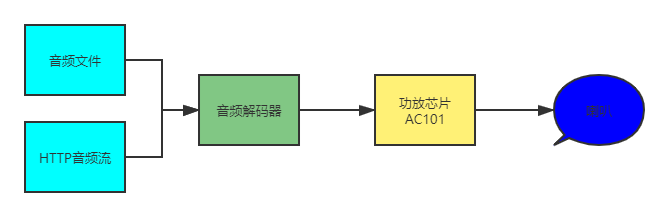
$oc/devices/{device\_id}/sys/properties/get/request\_id={request\_id}

数据段为json数据，包含要求上报的属性。

#### 3.4、应用侧接口

还可以通过华为云提供的应用侧接口，查询，控制设备的信息，实现在手机/网页端对设备的管理。

### 4、音频播放模块

本项目板子上装备了两个喇叭，可以用于播放音频，可播放 本地音频 和 网络音频流 。如图所示，由于嵌入式设备内存有限，故本地音频主要是一些简单的提示语音，其优点是播放速度快，响应及时；而HTTP音频流能灵活播放音频，但其受限于网络环境，故将二者结合互补。  #### 4.1、播放本地mp3 该模块的实现方式是在PC端，将mp3文件转换成二进制的音频文件，将音频文件嵌入到芯片的flash中。在程序运行时可通过读取flash中的二进制音频文件，经过mp3解码，输出到音频芯片播放。

由于flash大小有限，故音频文件只能很短小，适合一些提示性短语，其播放速度也更快。 具体实现细节，参考：[esp32播放本地mp3](https://blog.csdn.net/weixin_44821644/article/details/107934841)

#### 4.3、文本转语音

文本转语音的核心就是依靠百度AI平台的文本转语音接口，设备端只需要准备好文本数据，发送到AI平台的文本转语音接口，并从该接口中获取音频数据，将该音频数据输出到音频芯片进行播放。

实现的详细代码参考： [esp32a1s 百度文本转语音实现](https://blog.csdn.net/weixin_44821644/article/details/109107687)

### 5、红外模块

目前的家具，电器中，仍有很多电器使用红外遥控，特别是空调。同类型的设备使用的红外协议大同小异，并且都可以查看其协议内容。

红外模块可发送不同类型，品牌的产品的红外控制信号，同时具备**学习功能**，能通过遥控器，学习设备的红外协议。

#### 5.1、红外发射

红外遥控是通过特定的LED发射信号，设备接收该信号来响应。

在ESP32端来看，所有的红外信号都可看成一个方波信号，ESP32只要能产生所需要的方波信号，结合发射硬件，就能实现红外发射。ESP32中提供了一个叫RMT控制器的设备来控制信号，可利用该RMT控制器，产生所需的方波信号。

如何使用RMT产生方波，可参考：[红外发射与接收](https://blog.csdn.net/weixin_44821644/article/details/108592858)；

了解了如何产生方波，但更重要的是如何获取方波信号的电平状态和持续时长，这些信息就是红外传输的具体内容。很幸运的是，一些开源红外遥控码库提供了这些信息，本模块使用的Irext就是一个开源万能红外遥控码库、编解码压缩算法以及免费周边服务。

将所需要的码库二进制文件下载到文件系统，就能在程序运行时，通过文件系统打开码库，输入电器类型及遥控命令，即可生成一个特定的数组，该数组表示了方波信号的电平高低及电平时长，即可产生所需的方波信号。

如何使用irext实现该功能，请参考：[基于irext实现万能遥控器](https://blog.csdn.net/weixin_44821644/article/details/109292942)

#### 5.2、红外学习

红外学习主要用于快速配置空调协议。用本地空调遥控器，给esp32发射制冷模式 开机 26° 一级风速 自动扫风的特定命令，程序会从本地码库中匹配对应的协议。

红外接收的重点是解析输入的信号并将其与本地数据库进行匹配。目前由人工分别完成了对格力，美的，海尔三种空调的不同协议的分析并生成数据库，并在代码中实现对他们的识别。

红外接收任务负责接收与识别的实现，由parse\_items()和ir\_code\_lib\_update()完成信号识别，匹配。

//接收到的红外信号  
struct RX\_signal  
{  
 uint32\_t item\_num; //item数量  
 uint32\_t lowlevel; //低电平时间 us  
 uint32\_t highlevel\_1; //高电平1的时间  
 uint32\_t highlevel\_0; //高电平0的时间  
 uint32\_t encode; //由0和1组成的编码  
};  
/\*  
 \* 红外接收任务 rmt\_rx\_start()执行后才会接收数据  
 \* 接收的数据以item的数据结构存放到nvs  
\*/  
void rmt\_ir\_rxTask(void \*agr)  
{  
 size\_t rx\_size = 0;  
  
 RingbufHandle\_t rb = NULL;  
 rmt\_get\_ringbuf\_handle(rx\_channel, &rb); //获取红外接收器接收的数据 放在ringbuff中  
  
 rmt\_rx\_stop(rx\_channel); //暂停接收  
 while (rb)  
 {  
  
 //从ringbuff读取items 会进入阻塞 直到ringbuff中有新的数据  
 rmt\_item32\_t \*item = (rmt\_item32\_t \*)xRingbufferReceive(rb, &rx\_size, portMAX\_DELAY);  
 if (item)  
 {  
 ESP\_LOGI(TAG, "rx\_size = %u", rx\_size);  
  
 //!红外线接收器有干扰，需要滤波  
 if (rx\_size > 30)  
 {  
 struct RX\_signal sig; //接收信号结构体  
 size\_t item\_num = rx\_size / 4; //一个item32bit  
 sig.item\_num = item\_num;  
 sig.highlevel\_1 = 0;  
 sig.highlevel\_0 = 0;  
 sig.encode = 0;  
  
 //解析item  
 parse\_items(item, item\_num, &sig);  
  
 ir\_code\_lib\_update(&sig); //更新ac\_handle  
 rmt\_rx\_stop(rx\_channel); //暂停接收  
 xSemaphoreGive(IR\_sem); //释放信号量  
 }  
  
 //解析出数据后释放ringbuff的空间  
 vRingbufferReturnItem(rb, (void \*)item);  
 }  
 }  
 vTaskDelete(NULL);  
}

### 6、传感器模块

目前我们的板子上添加了ds18b20温度传感器，用于准确的读取室内的温度。

### 7、定时模块

定时模块主要用于记录当前的年份，月份，日期，时间，提供精确到秒的定时提醒服务，支持大时间尺度上的定时，如一天，两天甚至更长时间的定时提醒。其定时的时钟源是freertos的系统时钟。 其实现参考博客：[基于esp32 的时间系统](https://blog.csdn.net/weixin_44821644/article/details/115172987)

### 8，无线检测模块

暂时未完成，预期功能为：能实时检测人体/物体在一定空间内的活动状态，如人体是否静止OR活动中，从而来判断空间内是否有人类活动。

其工作原理是利用ESP32的wifi无线信号在复杂空间中传播时，在发送端和接收端之间因为物体移动影响电磁波的多径效应，可用系统的信道频率响应来描述多径传播特性。

#### 8.1、使用相关系数进行人体检测

根据谈青青的论文，wifi子载波之间相关系数会随着空间环境的变换而变换，当室内无人体活动时，各个载波之间的相关系数相对小，当空间内有人活动时，各个载波会因人体活动而产生变换，此时相关系数会增大。

我的实验：提取2s内的载波振幅，计算相关系数，发现无论何种情况，大部分的子载波的相关系数都接近1，与论文不是很符合，且人体的活动反而导致了相关系数的减小，与论文也不符合。但初期的实验确实证实相关系数可以 反映人体的活动。

#### 8.2、使用振幅平均值进行人体检测

根据其他论文，一定时间窗口2s内，计算载波的振幅的均值。当空间环境稳定的情况下，振幅均值应该不会出现过大的波动，当人体活动时，振幅均值会受到影响而产生波动。对于esp32的51条子载波，不同的子载波对活动的灵敏度不一样。有的子载波会产生较大的波动，而有的子载波没有明显变化。描述子载波的幅度均值变化程度可以用方差，但实验过程中，全部子载波的方差值似乎不能很好的反映人体活动。但单独一条子载波的振幅均值对人体活动却比较敏感。

#### 8.3、路由器与esp32相对位置对实验的影响

实验路由器有两种：一般路由器和手机热点。在相关系数实验中，以手机热点为路由的人体检测预期较好，但换成一般路由器则效果很差。

将路由器摆放到较高位置进行实验，实验效果也不明显。



在这里插入图片描述