

基于讯飞AI开放平台的WebAPI应用项目实践

一、项目概述

1.1 项目背景

人工智能时代来临，人们陷入焦虑的周期将越来越短。AI正以迅猛的发展速度不断替代人类既有的工作岗位。那么人工智能就是万能的吗？拥有所有的技能吗？答案并不是，跨领域推理能力、抽象能力、自我学习的能力、独立地搜索问题、解决问题和团队配合能力、知其然也知其所以然的能力、常识、自我意识、审美、情感等等，这些方面都是机器人所不能拥有的技能。所以企业对我们的这些能力要求很高，尤其是独立地搜索问题、解决问题和团队配合尤为重要。所以，我们希望通过这门实践课程的学习，锻炼学生开发项目的实践能力，同时又能对AI能力的应用开发有比较全面的了解和掌握。

1.2 项目目标

easyAIoT的目标是平台化、简单化讯飞AI能力平台的使用。因此采用了分层化的设计思路和更加轻量化的WebAPI接口，使得应用层的开发者可以很容易的接入，并且更加容易的实现AI能力的调用。

1.3 项目范围

本门课程基于讯飞开放平台提供的WebAPI接口，并进行二次封装，提供APP层的API接口，也提供了基础能力层的接口。APP层接口可以非常容易的开发基于讯飞的AI能力，基础能力层可以不受讯飞平台的限制，通过基础能力提供的接口，可以开发其他平台的AI能力应用，或者使用在其他需要的应用场景。

目前1.0版本提供的C语言的功能如下：

1. AIUI**应用层API接口**；
2. 语音合成**应用层API接口**；
3. 语音实时转写**应用层API接口**；
4. 人脸比对**应用层API接口**；
5. 跨平台TCP/UDP/HTTP/Websocket**基础能力及应用接口**；
6. 跨平台PCM录音接口；
7. 跨平台多线程应用接口；
8. MD5/Base64/Sha256等加解密基础能力接口；
9. Json序列化和反序列化的基础能力接口；
10. string/map/ring buffer等数据结构基础能力接口；

1.4 技术路线

源代码中包含有C语言基础、数据结构与数据处理、网络编程、多线程编程、分层框架设计、技术路线偏向于做基于C语言的应用层开发方向。

使用的硬件平台：vscode（跨平台/protable版本免安装，不仅可以直接放在电脑中运行，也可以把软件和代码插在U盘中直接运行，免除一些机房限制导致的问题）

使用的编译环境：Windows: mingw/cmake; Linux/Mac: gcc/cmake;

使用的语言：C语言

1.5 定义、首字母缩写词和缩略图

序号	名称	描述
1	easyAloT	项目名称，让设备能更容易的获得人工智能的能力
2	AIUI	讯飞提供的一套用于机器交互的接口引擎，使得机器能听的懂
3	IAT	语音识别，把人的声音识别为文字的技术
4	TTS	语音合成 (Text-To-Speech) , 通过机械的、电子的方法产生人造语音的技术
5		

二、功能总览

2.1 项目角色

本项目主要分为：AIUI机器交互应用、人脸比对应用、实时转写应用、语音合成应用；

也可以通过提供的基础能力层完成讯飞平台和其他AI开放平台（例如百度大脑）的AI能力集成；

2.2 项目功能分解

序号	功能	应用层用户	基础能力层用户
1	AIUI能力接口	提供录音、简单的HTTP接入、数据封装与处理；	提供自定义回调接口，方便用户自定义处理；
2	TTS语音合成接口	提供简单的TTS合成接口；	提供开放的回调接口，方便用户自定义处理；
3	IAT语音实时转写接口	提供简单的IAT实时转写功能；	提供开放的接口，供用户自定义语音输入和业务处理；
4	人脸比对接口	提供简单的人脸比对能力，只需要提供appid等验证信息和两张人脸照片即可；	提供基础的接口，可以扩展人脸相关的处理方式；
5	网络接口	提供TCP/UDP/Http/Websocket的C语言应用层接口，可以非常方便的接入对应协议的服务平台；	提供基础的网络接口，可以更加方面的自定义对网络的连接处理；
6	录音接口		■
7	多线程接口	提供跨平台的多线程接口，Windows/Linux平台已验证，Mac暂未测试	■
8	MD5/Base64/Sh256等加解密接口	提供加解密接口，方便通信过程中的加解密处理	■
9	Json接口封装	提供了基于C语言宏的Json序列化和反序列化，比原本的 cJSON 阅读起来更加方便理解；	■
10	string/map/ring buffer接口	提供了基础的数据处理接口，解决了C语言处理数据不方便的问题；	■

三、功能说明

3.1 功能框图

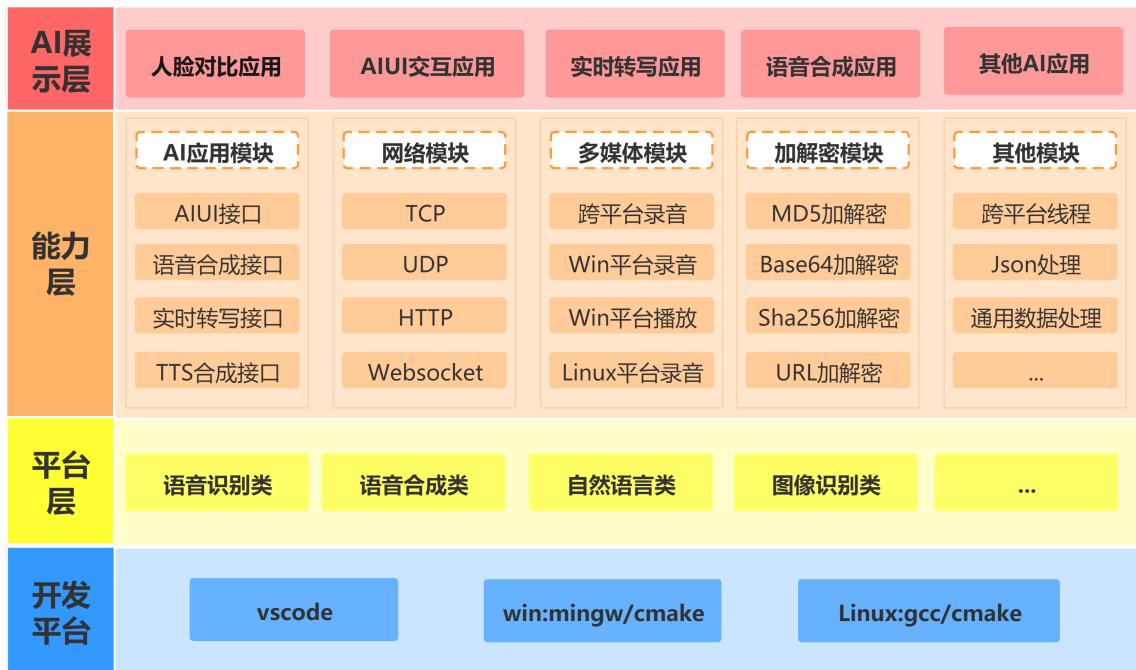
下图是easyAloT项目的分层框架图。

AI展示层：用于展示各种AI能力，通过能力层提供的模块完成；

能力层：由AI应用模块提供的能力接口和其他辅助模块构成；

平台层：目前easyAloT使用C语言适配了讯飞平台层提供的WebAPI能力；

开发平台层：由easyAloT开发使用的主要开发环境、编译器和调试工具组成；



3.2 AIUI模块功能说明

此模块提供语音识别的能力，通过提供的语音识别成文字的过程。

接口说明：

```

1 /**
2 * @brief 获取自然语言处理的结果（使用讯飞AIUIwebAPI引擎）
3 * @param pszAppid      应用APPID
4 * @param pszKey        应用APPKey
5 * @param pszParam      请求参数
6 * @param pAudioData    请求语音数据
7 * @param iAudioLen     请求语音数据长度
8 * @return cstring_t*    返回字符串结构体对象指针，错误返回NULL
9 */
10 cstring_t *getNlpResult(const char *pszAppid, const char *pszKey, const char
11 *pszParam,
                           void *pAudioData, int iAudioLen);

```

实例代码：

```

1 int main(int argc, char *argv[])
2 {
3     // 修改为自己的appid, key、secret和auth_id, 网址https://aiui.xfyun.cn/app/<替换自己的appid>/info
4     const char *pszAppid = "xxxxxxxxxx";
5     const char *pszKey = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
6     const char *pszParam = ""
7     "{\"result_level\":\"plain\",\"auth_id\":\"xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx\",
8      \"data_type\":\"audio\",\"sample_rate\":\"16000\",\"scene\":\"main_box\"}";
9
10    // 1.读取语音文件
11    cstring_t *pAudioData = readFile("../Res/test.pcm");
12    if (!pAudioData)
13    {
14        LOG(ERROR, "read audio file error");

```

```

13         return;
14     }
15     LOG(EDEBUG, "pcm len:%d", pAudioData->length(pAudioData));
16
17     // 2. 提供信息进行语音识别成文本
18     cstring_t *pResult = getNlpResult(pszAppid, pszKey, pszParam,
19     pAudioData->str, pAudioData->len);
20     if (pResult)
21     {
22         LOG(EDEBUG, "识别结果: %s", pResult->str);
23         cstring_del(pResult);
24     }
25
26     cstring_del(pAudioData);
27
28     return 0;
29 }
```

上述代码中的param参数是json格式，说明如下：

```

1 {
2     "result_level": "plain", // 提供结果格式：简单文本格式
3     "auth_id": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx", // 请求的auth_id，平台配置界面
4         中查询
5     "data_type": "audio", // 请求的数据类型，这里是语音，也可以为文本
6     "sample_rate": "16000", // 如果请求的是audio，sample_rate表示音频采样率
7     "scene": "main_box" // 请求场景，未发布版本，需要增加"_box"后缀
8 }
```

程序运行结果：

The screenshot shows the Visual Studio Code interface with the main.c file open. In the terminal tab, the output of the program is displayed, showing the recognized text: "今天全国新冠肺炎新增确诊病例15例". The terminal window title is "1: cppdbg: EasyAIoT.exe".

```

main.c - easyAIoT - Visual Studio Code
main.c - CMakeLists.txt

main.c > main(int, char **)
    LOG(EDEBUG, "pcm len:%d", pAudioData->length(pAudioData));
    cstring_t *pResult = getNlpResult(pszAppid, pszKey, pszParam, pAudioData->str, pAudioData->len);
    if (pResult)
    {
        LOG(EDEBUG, "识别结果: %s", pResult->str);
        cstring_del(pResult);
    }

    cstring_del(pAudioData);

E_DEBUG[1:cppdbg:EasyAIoT.exe] 识别结果: 今天全国新冠肺炎新增确诊病例15例
PS E:\Source\C\easyAIoT>
```

可以看到，返回的结果也是一个json格式，识别出了请求的语音文本为"今天全国新增确诊病例"，服务端返回的结果为："今天全国新冠肺炎新增确诊病例15例"。

注意，获得正确的返回结果，需要在应用中增加相关的技能，按照上例需要在应用中增加相关技能，并点击右上角的保存即可生效。

The screenshot shows the AIUI Open Platform's application configuration page for 'AIBIT Intelligent Robots'. The left sidebar includes sections for Application, Application Information, Application Configuration, Development, Development Tools, IP Whitelist, and Online. The main area has tabs for Scenario Mode (main), Keyword Filtering (with a note about wake words), and a large 'Modify and Save' button. In the center, there's a 'Language Skills' section with a breadcrumb trail: Custom Skills > Custom Q&A > Device Person > Store Skills > Keyword Q&A. A search bar for keywords is also present. A prominent red box highlights the 'Add Related Skills' button. Below it, a skill card for 'COVID-19' is shown with the text 'Coronavirus Disease Query'.

3.3 TTS语音合成模块功能说明

接口说明

```
1 /**
2  * @brief 文字转语音
3  * @param appid 讯飞appid
4  * @param key 讯飞api key
5  * @param param 讯飞合成参数
6  * @param text 合成的文本内容
7  * @param pathname 合成的结果音频文件路径名称
8  * @return true 合成成功
9  * @return false 合成失败
10 */
11 bool getTTS(const char *appid, const char *key, const char *param,
12             const char *text, const char *pathname);
```

示例代码

```
1 int main(int argc, char *argv[])
2 {
3     // 修改为自己应用的appid, key和secret
4     const char *appid = "xxxxxxxxxx";
5     const char *key = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
6     const char *secret = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
7
8     getTTS(appid, key, secret, "欢迎来到比特人生的世界", "../Res/tts.pcm");
9
10    return 0;
11 }
```

程序运行结果

生成的语言文件放在了Res/tts.pcm中，可以打开音频软件AU，设置16000采样，16bit深的音频信息来听取声音内容和要转写的文字内容是否一致。

```
文本转语音，通过adobe audition软件设置16000K 16bit听取
```

```
连接websocket服务端成功
```

```
PS E:\Source\C\easyAIoT>收到了101消息，并且验证key成功
```

3.4 IAT语音转写功能说明

接口说明

```
1 /**
2 * @brief 语音转写
3 * @param appid      应用appid
4 * @param key        应用apikey
5 * @param secret    应用secret
6 */
7 void iat(const char *appid, const char *key, const char *secret);
```

示例代码

```
1 int main(int argc, char *argv[])
2 {
3     // 修改为自己应用的appid, key和secret
4     const char *appid = "xxxxxxxxxx";
5     const char *key = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
6     const char *secret = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
7
8     LOG(EDEBUG, "尝试说一些话，看看有什么有趣的事情会发生(•'◡'•)");
9     iat(appid, key, secret);
10
11     return 0;
12 }
```

程序运行结果

程序运行后，可以一边说话，一边看返回结果。

The screenshot shows a code editor with a terminal window at the bottom. The terminal output includes a websocket handshake and a log message indicating a successful device open. A red box highlights the log message: "[EDEBUG][E:\Source\C\easyAIoT\APP\iat.c RecordCB line:134]record device open success 测试实时撰写的效果，可以一边说话，然后会一边转写。".

```
81 }
82 }
83 int main(int argc, char *argv[])
84 {
85     const char *appid = "5d2f27d2";
86     const char *key = "a8331910d59d41deea317a3c76d47b60";
87     const char *secret = "8110566cd9dd13066f9a1e38aeb12a48";
88
89     iat(appid, key, secret);
90
91     return 0;
92 }
```

终端 问题 2 输出 调试控制台 1: cppdbg: EasyAIoT.exe +

```
Sec-WebSocket-Key: YAsK9fldXKbKaXbHtrD57Q==
Sec-WebSocket-Version: 13
Upgrade: websocket

[EDEBUG][E:\Source\C\easyAIoT\APP\iat.c bWSCallback line:184]websocket handshake success
[EDEBUG][E:\Source\C\easyAIoT\APP\iat.c RecordCB line:134]record device open success
测试实时撰写的效果，可以一边说话，然后会一边转写。
PS E:\Source\C\easyAIoT>
```

3.5 人脸比对接口功能说明

接口说明

```
1 /**
2  * @brief 讯飞人脸对比功能
3  * @param pszAppID      讯飞appid
4  * @param pszAppSecret   讯飞api secret
5  * @param pszAppKey      讯飞app key
6  * @param pszImagePath1 比较的图片1
7  * @param pszImagePath2 比较的图片2
8  * @return float          比较的相似度，取值0-1，0.67以上阈值是同一个人的可行性很大
9  */
10 double fFaceCompare(const char *pszAppID,
11                      const char *pszAppSecret, const char *pszAppKey,
12                      const char *pszImagePath1, const char *pszImagePath2);
```

示例代码

```
1 int main(int argc, char *argv[])
2 {
3     // 修改为自己应用的appid, key和secret
4     const char *appid = "xxxxxxxxxx";
5     const char *key = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
6     const char *secret = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
7
8     double fScore = fFaceCompare(appid, secret, key, "../Res/1.jpg",
9                                  "../Res/2.jpg");
10    printf("face compare score:%.2f\n", fScore);
11
12    return 0;
13 }
```

使用的两张图片如下



程序运行结果

```

1 [EDEBUG] [E:\Source\C\easyAIoT\APP\FaceCompare.c fFaceCompare line:144]body:
{"header":
 {"code":0,"message":"success","sid":"ase000d52ea@hu17642958dc90212882"},"payload": {"face_compare_result": {"compress": "raw", "encoding": "utf8", "format": "json", "text": "ewoJInJldCIgOiAwL AoJInNjb3JlIiA6IDAuOTc1NTI1Nzk2NDEzNDIxNjMKfQo="}}}
2 [EDEBUG] [E:\Source\C\easyAIoT\APP\FaceCompare.c fGetScoreResult line:74]szResponse:{ "header":
 {"code":0,"message":"success","sid":"ase000d52ea@hu17642958dc90212882"},"payload": {"face_compare_result": {"compress": "raw", "encoding": "utf8", "format": "json", "text": "ewoJInJldCIgOiAwL AoJInNjb3JlIiA6IDAuOTc1NTI1Nzk2NDEzNDIxNjMKfQo="}}}
3 [EDEBUG] [E:\Source\C\easyAIoT\APP\FaceCompare.c fGetScoreResult line:90]score json:{ "ret" : 0,
5 "score" : 0.97552579641342163
6 }
7
8 [EDEBUG] [E:\Source\C\easyAIoT\APP\FaceCompare.c fGetScoreResult line:101]ret:0, score:0.98
9 face compare score:0.98

```

以上两张图片的得分0.98，代表识别的两张图片非常像。

3.6 网络接口功能说明

3.6.1 tcp功能

接口说明

```

1 /**
2 * @brief 网络接收回调
3 */
4 typedef bool (*fnSockCallback)(struct ssockclient *pstClient, void
5 *pvUserData, void *pvData, int iLen);
6 /**
7 * @brief 网络通信结构体，包含创建网络、连接网络、关闭网络，发送数据和网络通信事件循环
8 */
9 typedef struct ssockclient {
10     sock_t          iSocket;
11     int             iRecvBufferSize;
12     int             iSendBufferSize;
13     int             iProtocol;
14     struct in_addr  iServerIp;
15     short           nServerPort;
16
17     bool (*bCreate)(struct ssockclient *pstClient, const char *ip, short
18 nPort);
19     bool (*bConnect)(struct ssockclient *pstClient);
20     void (*vClose)(struct ssockclient *pstClient);
21     int (*iSend)(struct ssockclient *pstClient, void *pvData, int iLen);
22     bool (*bEventLoop)(struct ssockclient *pstClient, fnSockCallback cb,
23 void *pvUserData, int *piLoop, int iTimeOut);
24 }ssockclient_t;

```

```

24 /**
25 * @brief 网络功能初始化
26 * @param pstClient 网络功能对象
27 * @return true 网络初始化成功
28 * @return false 网络初始化失败
29 */
30 bool bSockInit(ssockclient_t *pstClient);

31 /**
32 * @brief 网络功能反初始化
33 * @param pstClient 网络功能对象
34 * @return true 网络反初始化成功
35 * @return false 网络反初始化成功
36 */
37 bool bSockUninit(ssockclient_t *pstClient);

```

示例代码

```

1 // 回调函数，当服务端有数据发送回来时执行这个回调
2 bool bSockCallback(struct ssockclient *pstClient, void *pvUserData, void
3 *pvData, int iLen)
4 {
5     printf("bSockCallback called:%s\r\n", pvData);
6     return true;
7 }
8
9 int main(int argc, char *argv[])
10 {
11     int iLoop = 1;
12
13     // 请求头信息
14     const char header[] =
15     {
16         "GET / HTTP/1.1\r\n"
17         "Host: www.baidu.com\r\n"
18         "Connection: keep-alive\r\n"
19         "Accept: */*\r\n\r\n"};
20
21     // 声明网络结构体变量
22     SsockClient_t stSockClient;
23
24     // 初始化网络结构体
25     bSockInit(&stSockClient);
26
27     // 创建socket连接对象
28     stSockClient.bCreate(&stSockClient, "14.215.177.39", 80);
29
30     // 连接服务端
31     stSockClient.bConnect(&stSockClient);
32
33     // 发送数据给服务端
34     stSockClient.iSend(&stSockClient, (void *)header, strlen(header) + 1);
35
36     // 等待网络事件，通过bSockCallback处理网络事件
37     stSockClient.bEventLoop(&stSockClient, bSockCallback, NULL, &iLoop,
1000);

```

```
38     return 0;
39 }
```

程序运行结果

The screenshot shows the Visual Studio Code interface. On the left is the sidebar with project files like EIHttpClient.h, EILog.h, EIPlatform.h, EI Socket.c, EI Socket.h, EIString.c, EIString.h, wsclient.c, and wsclient.h. The main editor window displays the main.c file with code related to socket operations. Below the editor is the terminal window, which contains the output of a debugger session. A red box highlights the terminal output, which includes a command to start a debugger and the resulting HTTP response from a server. To the right of the terminal, a red box contains the text: "通过回调，服务端返回的结果，可以保存下来，改为HTML格式打开".

```
// 声明网络结构体变量
SSockClient_t stSockClient;
// 初始化网络结构体
bSockInit(&stSockClient);
// 创建socket连接对象
stSockClient.bCreate(&stSockClient, "14.215.177.39", 80);
// 连接服务端
stSockClient.bConnect(&stSockClient);

通过回调，服务端返回的结果，  
可以保存下来，改为HTML格式  
打开
```

3.6.2 HTTP功能

接口说明

```
1 /**
2 * @brief 连接http服务端
3 *
4 * @param pstHttpInfo httpclient的结构体，包含了请求、响应、网络连接、http解析四个功
5 * 能
6 * @param pszUrl 访问的http url
7 * @param pszExtraHeader 额外的头文件
8 * @param pvBody 如果是post消息，需要提供这一项，将会填充到http请求的body中
9 * @param isize body的长度
10 *
11 * @return 调用成功返回true，否则返回false
12 *
13 * @note 注意返回结果在pstHttpInfo的stResponse成员中，并且调用完成后
14 *       要通过bHttpClose来释放pstHttpInfo中的资源
15 */
16 bool bConnectHttpServer(SEIHttpInfo_t *pstHttpInfo, const char *pszUrl,
                           const char *pszExtraHeader, void* pvBody, int isize);
```

示例代码

```
1 int main(int argc, char *argv[])
2 {
3     SEIHttpInfo_t stHttpInfo;
4     const char *pszUrl = "http://www.baidu.com";
5
6     // 连接HTTP服务器
7     bConnectHttpServer(&stHttpInfo, pszUrl, NULL, NULL, 0);
8     LOG(EDEBUG, "status:%d", stHttpInfo.stResponse.iStatus);
```

```

9     LOG(EDEBUG, "body:%d", stHttpInfo.stResponse.pstBody->sBuffer);
10    bHttpClose(&stHttpInfo);
11
12    return 0;
13 }

```

程序运行结果

status的返回值为http的状态码，200代表正常返回，body是服务端返回的内容，以上代码会返回网址的首页信息，因此会看到终端输出的是网页的内容信息。



```

57
58     const char *pszUrl = "http://www.baidu.com";
59
60     // 连接HTTP服务器
61     bConnectHttpServer(&stHttpInfo, pszUrl, NULL, NULL, 0);
62     LOG(EDEBUG, "status:%d", stHttpInfo.stResponse.iStatus);
63     LOG(EDEBUG, "body:%s", stHttpInfo.stResponse.pstBody->sBuffer);
64
65     bHttpClose(&stHttpInfo);
66
67     return 0;
}

```

终端 问题 输出 调试控制台 1: cppdbg: EasyAIoT.exe + ×

```

document.cookie="NOJS=;expires=Sat, 01 Jan 2000 00:00:00 GMT";
}

</script>

<script src="http://ss.bdimg.com/static/superman/js/components/hotsearch-8f112f3361.js"></script>
<script defer src="//hectorstatic.baidu.com/cd37ed75a9387c5b.js"></script>
</body>

</html>

```

PS E:\Source\C\easyAIoT>

3.6.3 Websocket功能

接口说明

```

1 /**
2  * @brief 连接到websocket服务端
3  * @param c_pszurl      websocket服务器的URL
4  * @param cb            websocket服务器的响应信息
5  * @return true         连接成功
6  * @return false        连接失败
7 */
8 bool bwebsocketConnect(const char *c_pszurl, fnwebsocketcallback cb);

```

示例代码

```

1 int main(int argc, char *argv[])
2 {
3     const char *appid = "xxxxxxxxxx";
4     const char *app_secret = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
5     const char *app_key = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
6
7     getTTS(appid, app_key, app_secret, "欢迎来到比特人生的世界", "./tts.pcm");
8
9     return 0;
10}

```

程序运行结果

合成的结果保存到了tts.pcm中，可以打开adobe audiom软件，设置为16000K 16bit听声音是否和文本内容一致。



The screenshot shows a terminal window with the following log output:

```
20+06%3A53%3A33+GMT&host=ws-api.xfyun.cn HTTP/1.1
Connection: Upgrade
Host: tts-api.xfyun.cn:80
Upgrade: websocket
Sec-WebSocket-Version: 13
[EDebug][E:\Source\C\easyAIoT\APP\tts.c bWS Callback line:57]收到了101消息，并且验证key成功
[EDebug][E:\Source\C\easyAIoT\APP\tts.c getTTS line:119]connect over
PS E:\Source\C\easyAIoT>
```

A red arrow points from the text "合成的文件名称" to the file name "tts.pcm" in the project tree. A red box highlights the log message "[EDebug][E:\Source\C\easyAIoT\APP\tts.c bWS Callback line:57]收到了101消息，并且验证key成功". To the right of this box, the text "101是websocket连接成功消息" is displayed.

3.7 录音接口功能说明

接口说明

```
1  typedef void*    AudioHandle;
2 /**
3  * @brief 录音回调
4  * @param pvHandle      录音对象, AudioRecorder_t类型的指针
5  * @param iType         录音状态, AUDIO_OPEN表示录音设备打开成功, AUDIO_DATA表示录
6  * 音数据
7  * @param pvUserData   用户传递到回调的私有数据
8  * @param pData         录音数据, 当iType=AUDIO_DATA时有效
9  * @param iLen          录音数据长度, 当iType=AUDIO_DATA时有效
10 */
11 typedef void     (*RecordCallback)(void* pvHandle, int32_t iType, void*
12 pvUserData, void* pData, int32_t iLen);
13 /**
14 * @brief 录音配置结构体
15 */
16 typedef struct AudioConfig {
17     uint32_t           iSampleRate;        /* 每秒采样次数 */
18     uint8_t            byBitsPerSample;    /* 每次采样深度 */
19     uint8_t            byChannels;        /* 采样通道 */
20     RecordCallback    pfCallback;        /* 获取音频数据的回调 */
21     void*             pvUserData;       /* 传递给回调的参数 */
22 }AudioConfig_t;
23 /**
24 * @brief 录音功能结构体
25 */
26 typedef struct AudioRecorder {
27     /**
28     * @brief 打开录音设备
29     * @param pAudioConfig  录音配置信息
30     * @return bool         false打开失败, true打开成功
31 }
```

```

31     */
32     bool (*open)(AudioConfig_t* pAudioConfig);
33 /**
34     * @brief 开始流式录音，录音信息通过open时注册的回调函数返回给用户
35     */
36     bool (*start)(void);
37 /**
38     * @brief 停止录音，异步
39     */
40     void (*stop)(void);
41 /**
42     * @brief 关闭录音设备
43     */
44     void (*close)(void);
45     AudioHandle             pvAudioHandle;      /* 音频设备的句柄 */
46     AudioConfig_t            stAudioConfig;       /* 音频设备的配置 */
47     bool                     bRecording;         /* 是否正在录音 */
48 }AudioRecorder_t;
49
50 /**
51 * @brief 录音对象单例
52 */
53 extern AudioRecorder_t Recorder;

```

示例代码

```

1 int main()
2 {
3     // 16000K采样，16bit采样深度，1是单声道，RecordCB是音频信息的回调函数，test.pcm
4     // 是传递给回调的参数
5     AudioConfig_t stAudioConfig = {16000, 16, 1, RecordCB,
6     (void*)"test.pcm"};
7     // 打开音频设备
8     Recorder.open(&stAudioConfig);
9     // 开始录音，阻塞函数，需要关闭录音在其他地方调用Recorder.close()
10    Recorder.start();
11    // 关闭录音设备
12    Recorder.close();
13
14    return 0;
15 }

```

程序运行结果

程序运行成功后，可以通过编写回调函数，来获取音频数据，上面程序的回调函数可以通过以下的回调处理来保存录音数据

```

1 static void RecordCB(void* pvHandle, int32_t iType, void* pvUserData, void*
2 pvData, int32_t iLen)
3 {
4     uint32_t byteswritten = 0;
5     static uint32_t totalwritten = 0;
6     const char *pathname = (const char *)pvUserData;
7     static FILE* fp = NULL;
8     if (!fp) {
9         fp = fopen(pathname, "wb+");

```

```

9    }
10   if (AUDIO_DATA == iType) {
11     if (fp) {
12       bytesWritten = fwrite(pvData, iLen, 1, fp);
13       totalWritten += bytesWritten*iLen;
14       LOG(EDEBUG, "totalWritten:%d, iLen:%d", totalWritten, iLen);
15       if (totalWritten > 32000*5) {
16         Recorder.stop();
17       }
18     }
19   }
20   else if (AUDIO_CLOSE == iType) {
21     if (fp) fclose(fp);
22     fp = NULL;
23     Recorder.stop();
24   }
25 }
```

pvUserData参数是通过stAudioConfig传递进来的"test.pcm"，可以在回调中使用这个参数，把录取的音频数据保存到test.pcm中。保存之后可以通过adobe audition软件打开听取声音是否正常。

3.8 多线程接口功能说明

接口说明

```

1  typedef struct ThreadFun ThreadFun;
2  struct ThreadFun{
3    void* params; //线程函数的参数
4 #if defined(_WIN32)
5    unsigned int (*fun)(void *params); //线程函数指针
6 #else
7    void* (*fun)(void *params); //线程函数指针
8 #endif
9  };
10 /**
11 * @brief 线程创建
12 * @param iThreadNum      线程创建的个数
13 * @param funArray        线程处理函数组
14 */
15 void vStartThread(int iThreadNum, ThreadFun funArray[]);
```

示例代码

```

1 #include <SimpleThread.h>
2 static void *doSomething(void *params)
3 {
4   printf("doSomething thread:%s\r\n", (const char *)params);
5   return NULL;
6 }
7
8 int main()
9 {
10   ThreadFun funArr[1];
11   funArr[0].fun = doSomething;
12   funArr[0].params = (void *)"easyAIot";
```

```
13     vStartThread(1, funArr);
14     printf("main thread\n");
15     return 0;
16 }
```

程序运行结果

```
1 | main thread
2 | doSomething thread:easyAIot
```

3.9 加解密接口功能说明

3.9.1 MD5加密

接口说明

```
1 /**
2  * @brief md5加密
3  * @param initial_msg      加密消息串
4  * @param initial_len       加密消息串长度
5  * @param digest            加密后的内容，注意：长度固定16字节，在使用时一般会把每一个字节使用16进制转化为字符串
6  */
7 void vMD5(const unsigned char *msg, unsigned int len, unsigned char
8           *digest);
9 /**
10 * @brief md5加密
11 * @param initial_msg      加密消息串
12 * @param initial_len       加密消息串长度
13 * @param digest            加密后的32字节小写字符串内容
14 */
15 void md5String(const unsigned char *msg, unsigned int len, unsigned char
16                 *digest);
```

示例代码

```
1 int main(int argc, char *argv[])
2 {
3     const char *pszString = "easyAIoT";
4     char szBuf[33] = {0};
5
6     md5String((uint8_t *)pszString, strlen(pszString), szBuf);
7     printf("%s md5-> %s\n", pszString, szBuf);
8
9     return 0;
10}
```

程序运行结果

```
1 | easyAIoT md5-> d1428978a06bcfed0e94ab70a5d1498
```

3.9.2 Base64加解密

接口说明

```
1 /**
2 * @brief Base64编码
3 * @param pcData 需要编码的数据
4 * @param pcBase64 Base64编码后的数据
5 * @param pcDataLen 需要编码数据的长度
6 * @return int 编码后数据的长度
7 */
8 int iBase64Encode(const char *pcData, char *pcBase64, int pcDataLen);
9
10 /**
11 * @brief Base64解码
12 * @param base64 需要解码的Base64字符串
13 * @param dedata 解码后的数据内容
14 * @return int 解码后数据的长度
15 */
16 int iBase64Decode(const char *base64, unsigned char *dedata);
```

示例代码

```
1 #include <base64.h>
2
3 int main(int argc, char *argv[])
4 {
5     const char *pcPlain = "easyAIoT";
6     char szEncode[32] = {0};
7     char szDecode[32] = {0};
8
9     int iEncodeLen = iBase64Encode(pcPlain, szEncode, strlen(pcPlain));
10    int iDecodeLen = iBase64Decode(szEncode, szDecode);
11
12    printf("encode:%s, len:%d\n", szEncode, iEncodeLen);
13    printf("decode:%s, len:%d\n", szDecode, iDecodeLen);
14
15    return 0;
16 }
```

程序运行结果

```
1 encode:ZWFezeUFJb1Q=, len:12
2 decode:easyAIoT, len:8
```

3.9.3 Sha256加解密

接口说明

```
1 /**
2 * @brief sha256加密
```

```
3 * @param input 加密源字符串
4 * @param ilen 加密源字符串大小
5 * @param out 加密字符串，大于等于65字节
6 */
7 void sha256(const unsigned char *input, size_t ilen, unsigned char *out);
8
9 /**
10 * @brief HamcSHA256加密
11 * @param data 加密数据
12 * @param len 加密数据长度
13 * @param key 加密key
14 * @param len_key 加密key长度
15 * @param out 加密内容，out长度大于等于65
16 */
17 void hamcSha256String(const unsigned char *data, size_t len, const unsigned
18 char *key, int len_key, unsigned char *out);
```

示例代码

```
1 int main(int argc, char *argv[])
2 {
3     const char *input = "easyAIoT";
4     const char *key = "123456";
5     char out[65];
6     char hamc_out[65] = {0};
7
8     sha256(input, strlen(input), out);
9     printf("out:%s\n", out);
10
11    hamcSha256String(input, strlen(input), key, strlen(key), hamc_out);
12    printf("hamc_out:%s\n", hamc_out);
13
14    return 0;
15 }
```

程序运行结果

```
1 out:72db9c2cd52e183d7f3a367c8f33a226090c2c73b049731908c8341093c87d77
2 hamc_out:a46e68f86c13dbfb00a834f223cc68c516915f31c7736cdd5e1664ba69c32343
```

3.9.4 URL加解密

接口说明

```

1 /**
2 * @brief url网址编码，便于网络传输
3 * @param in          编码前网址
4 * @param out         编码后网址
5 */
6 void urlencode(char in[], char out[]);
7
8 /**
9 * @brief url网址解码，便于查看
10 * @param in          解码前网址
11 * @param out         解码后网址
12 */
13 void urldecode(char in[], char out[]);

```

示例代码

```

1 int main(int argc, char *argv[])
2 {
3     const char *in = "http://www.easyaiot.com/index.html?param1=1 2
4 &param2=1:2";
5     char enout[64], deout[64];
6
7     urlencode(in, enout);
8     urldecode(enout, deout);
9
10    printf("in:%s\n", in);
11    printf("enout:%s\n", enout);
12    printf("deout:%s\n", deout);
13
14    return 0;
}

```

程序运行结果

```

1 in:http://www.easyaiot.com/index.html?param1=1 2 3&param2=1:2
2 enout:http%3A//www.easyaiot.com/index.html?param1=1+2+3&param2=1%3A2
3 deout:http://www.easyaiot.com/index.html?param1=1 2 3&param2=1:2

```

3.10 Json接口功能说明

3.10.1 Json序列化

接口说明

```

1 // 创建Json对象，序列化对象名: json_obj
2 #define JSON_SERIALIZE_CREATE_OBJECT_START(json_obj)
3 // 创建序列化数组 (key, value) 到json_obj对象中
4 #define JSON_SERIALIZE_ADD_ARRAY_TO_OBJECT(json_obj, key, value)
5 // 创建序列化对象 (key, value) 到json_obj对象中
6 #define JSON_SERIALIZE_ADD_OBJECT_TO_OBJECT(json_obj, key, value)
7 // 增加一个字符串键值对 (key, value) 到json_obj对象中
8 #define JSON_SERIALIZE_ADD_STRING_TO_OBJECT(json_obj, key, value)
9 // 增加一个整型键值对 (key, value) 到json_obj对象中

```

```

10 #define JSON_SERIALIZE_ADD_INT_TO_OBJECT(json_obj, key, value)
11 // 创建一个数组Json对象
12 #define JSON_SERIALIZE_CREATE_ARRAY_START(json_array)
13 // 增加一个数组Json对象 (key, value) 到json_array对象中
14 #define JSON_SERIALIZE_ADD_ARRAY_TO_ARRAY(json_array, sub_json_array)
15 // 增加一Json对象 (key, value) 到数组对象json_array中
16 #define JSON_SERIALIZE_ADD_OBJECT_TO_ARRAY(json_array, json_obj)
17 // 创建Json结束符
18 #define JSON_SERIALIZE_CREATE_END(json_obj)
19 // JSON序列化为字符串
20 #define JSON_SERIALIZE_STRING(json_doc, str, len)

```

示例代码

```

1 JSON_SERIALIZE_CREATE_OBJECT_START(json_common_obj);
2 JSON_SERIALIZE_ADD_STRING_TO_OBJECT(json_common_obj, "app_id", "123456");
3
4 JSON_SERIALIZE_CREATE_OBJECT_START(json_business_obj);
5 JSON_SERIALIZE_ADD_STRING_TO_OBJECT(json_business_obj, "language", "zh_cn");
6 JSON_SERIALIZE_ADD_STRING_TO_OBJECT(json_business_obj, "domain", "iat");
7 JSON_SERIALIZE_ADD_STRING_TO_OBJECT(json_business_obj, "accent",
8 "mandarin");
9
9 JSON_SERIALIZE_CREATE_OBJECT_START(json_data_obj);
10 JSON_SERIALIZE_ADD_INT_TO_OBJECT(json_data_obj, "status", 0);
11 JSON_SERIALIZE_ADD_STRING_TO_OBJECT(json_data_obj, "format",
12 "audio/L16;rate=16000");
13 JSON_SERIALIZE_ADD_STRING_TO_OBJECT(json_data_obj, "encoding", "raw");
14 // JSON_SERIALIZE_CREATE_END(json_data_obj);
15
16 JSON_SERIALIZE_CREATE_OBJECT_START(json_frame_obj);
17 JSON_SERIALIZE_ADD_OBJECT_TO_OBJECT(json_frame_obj, "common",
18 json_common_obj);
19 JSON_SERIALIZE_ADD_OBJECT_TO_OBJECT(json_frame_obj, "business",
20 json_business_obj);
21 JSON_SERIALIZE_STRING(json_frame_obj, pszRequest, iLen);
21 JSON_SERIALIZE_CREATE_END(json_frame_obj);

```

程序运行结果

```

1 {
2     "common": {
3         "app_id": "123456"
4     },
5     "business": {
6         "language": "zh_cn",
7         "domain": "iat",
8         "accent": "mandarin"
9     },
10    "data": {
11        "status": 0,
12        "format": "audio/L16;rate=16000",
13        "encoding": "raw",

```

```
14     "audio": "exSI6ICJlbiiscgkgICAgInBvc210aw9uijogImZhbHNlIgoJf"
15 }
16 }
```

3.10.2 Json反序列化

接口说明

```
1 // 根据json字符串json_string创建json对象json_root
2 #define JSON_DESERIALIZE_START(json_root, json_string, ret)
3
4 // 根据json对象json_doc, 获取键值为key的整型变量放到value中, 返回值放入ret中, jump为
5 // 程序跳出方式
6 #define JSON_DESERIALIZE_GET_INT(json_doc, key, value, ret, jump)
7
8 // 根据json对象json_doc, 获取键值为key的浮点型变量放到value中, 返回值放入ret中, jump
9 // 为程序跳出方式
10 #define JSON_DESERIALIZE_GET_DOUBLE(json_doc, key, value, ret, jump)
11
12 // 根据json对象json_doc, 获取键值为key的字符串指针赋值给value, 返回值放入ret中, jump
13 // 为程序跳出方式
14 #define JSON_DESERIALIZE_GET_STRING(json_doc, key, value, ret, jump)
15
16 // 根据json对象json_doc, 获取键值为key的字符串变量放到value中, 返回值放入ret中, jump
17 // 为程序跳出方式
18 #define JSON_DESERIALIZE_GET_STRING_COPY(json_doc, key, value, len, ret,
19 jump)
20
21 // 根据json对象json_doc, 获取键值为key的Json数组变量放到value中, 返回值放入ret中,
22 // jump为程序跳出方式
23 #define JSON_DESERIALIZE_GET_ARRAY(json_doc, key, value, ret, jump)
24
25 // 根据json对象json_doc生成sub_item迭代器
26 #define JSON_DESERIALIZE_ARRAY_FOR_EACH_START(json_doc, sub_item, pos,
27 total)
28
29 // Json反序列结束标识
30 #define JSON_DESERIALIZE_END(json_root, ret)
```

示例代码

反序列的Json字符串

```
1 {
2     "payload": {
3         "face_compare_result": {
4             "text": "ewoJInJldCigOiAwLAoJInNjb3JlIiA6IDAuOTk2MTg2MDC3NTk0NzU3MDgkfQo="
5         }
6     }
7 }
```

反序列化代码

```
1 JSON_DESERIALIZE_START(json_root, szResponse, iRet);
2     JSON_DESERIALIZE_GET_OBJECT(json_root, "payload", payload_obj, iRet,
3     JSON_CTRL_BREAK);
4     JSON_DESERIALIZE_GET_OBJECT(payload_obj, "face_compare_result",
5     result_obj, iRet, JSON_CTRL_BREAK);
5     JSON_DESERIALIZE_GET_STRING(result_obj, "text", pText, iRet,
6     JSON_CTRL_NULL);
6     if (pText) text->appendStr(text, pText, strlen(pText));
7 JSON_DESERIALIZE_END(json_root, iRet);
```

程序运行结果

运行结果是通过JSON_DESERIALIZE_GET_STRING把text的值存放到pText中。

3.11 基础数据处理接口

3.11.1 string数据处理

接口说明

```
1 // 创建一个新的字符串对象cs
2 cstring_new(cs);
3 // 创建一个新的字符串对象cstr, 长度为len
4 cstring_new_len(cstr, len);
5 // 使用cstring_new的cstr需要通过cstring_del释放资源
6 cstring_del(cstr);
7
8 typedef struct cstring
9 {
10     char *str;
11     // 字符串
12     size_t alloced;
13     // 动态分配
14     size_t len;
15     // 字符长度
16
17     struct cstring *(*create)(size_t len);
18     // 创建字符串
19     void (*destory)(struct cstring *);
20     // 释放字符串
21     void (*appendStr)(struct cstring *cs, const char *str, size_t len);
22     // 追加字符串
23     void (*appendChar)(struct cstring *cs, char c);
24     // 追加字符
```

```

18     void (*appendInt)(struct cstring *cs, int val);
// 追加整型
19     void (*frontStr)(struct cstring *cs, const char *str, size_t len);
// 从头插入字符串
20     void (*frontChar)(struct cstring *cs, char c);
// 从头插入字符
21     void (*frontInt)(struct cstring *cs, int val);
// 从头插入整型
22     void (*clear)(struct cstring *cs);
// 清空字符串内容
23     void (*truncate)(struct cstring *cs, size_t len);
// 截断字符串
24     void (*dropBegin)(struct cstring *cs, size_t len);
// 丢掉前面指定len的字符
25     void (*dropEnd)(struct cstring *cs, size_t len);
// 丢掉后面指定len的字符
26     size_t (*length)(const struct cstring *cs);
// 求取字符串长度
27     const char *(*peek)(const struct cstring *cs);
// 获取字符串首地址
28     char *(*dump)(const struct cstring *cs, size_t *len);
// 获取字符串内容
29 } cstring_t;

```

示例代码

```

1 int main()
2 {
3     cstring_new(cs);
4
5     cs->appendStr(cs, "123", 0);
6     cs->appendChar(cs, '4');
7     cs->appendInt(cs, '4');
8     printf("%s \n", cs->peek(cs));
9
10    cs->frontStr(cs, "789", 0);
11    printf("%s \n", cs->peek(cs));
12
13    cs->dropBegin(cs, 2);
14    printf("%s \n", cs->peek(cs));
15
16    cs->dropEnd(cs, 2);
17    printf("%s \n", cs->peek(cs));
18
19    cstring_del(cs);
20    return 0;
21 }

```

程序运行结果

```

1 123452
2 789123452
3 9123452
4 91234

```

3.11.2 map数据处理

接口说明

```
1 功能:  
2 所有的map都是通过宏定义实现的.  
3  
4 // 创建一个map结构体, T可以为自定义类型  
5 map_t(T)  
6 Creates a map struct for containing values of type T.  
7  
8 // 初始化map, map在使用前必须要调用map_init  
9 map_init(m)map_init  
10  
11 // 反初始化map, 释放使用的资源  
12 map_deinit(m)  
13 Deinitialises the map, freeing the memory the map allocated during use; this  
should be called when we're finished with a map.  
14  
15 // 获取给定key的map值, 返回的是指向值得指针  
16 map_get(m, key)  
17  
18 // 设置一个指定key的值value到map中, 操作正常返回0, 否则失败返回-1  
19 map_set(m, key, value)  
20  
21 // 把指定为key的数据从map中移除  
22 map_remove(m, key)  
23  
24 // 获取map迭代器, 返回一个map_iter_t类型, 可以通过map_next遍历  
25 map_iter(m)  
26 Returns a map_iter_t which can be used with map_next() to iterate all the  
keys in the map.  
27  
28 // 通过map_next迭代map中的数据  
29 map_next(m, iter)  
30 Uses the map_iter_t returned by map_iter() to iterate all the keys in the  
map. map_next() returns a key with each call and returns NULL when there are  
no more keys.  
31  
32 const char *key;  
33 map_iter_t iter = map_iter(&m);  
34  
35 while ((key = map_next(&m, &iter))) {  
36     printf("%s -> %d", key, *map_get(&m, key));  
37 }
```

示例代码

```

1 map_int_t m;
2 map_init(&m);
3
4 map_set(&m, "testkey", 123);
5
6 int *val = map_get(&m, "testkey");
7 if (val) {
8     printf("value: %d\n", *val);
9 } else {
10    printf("value not found\n");
11 }
12
13 map_deinit(&m);

```

程序运行结果

```
1 | value: 123
```

3.11.3 循环buffer数据处理

接口说明

```

1 // ring buffer初始化
2 void ring_buffer_init(ring_buffer_t *buffer);
3 // 存入data字符数据到buffer中
4 void ring_buffer_queue(ring_buffer_t *buffer, char data);
5 // 存放长度位size的数据到ring buffer中
6 void ring_buffer_queue_arr(ring_buffer_t *buffer, const char *data,
ring_buffer_size_t size);
7 // 从ring buffer中获取一个字符数据到data中
8 uint8_t ring_buffer_dequeue(ring_buffer_t *buffer, char *data);
9 // 从ring buffer中获取一个长度为len的数据到data中, 返回真实的数据长度
10 ring_buffer_size_t ring_buffer_dequeue_arr(ring_buffer_t *buffer, char
*data, ring_buffer_size_t len);
11 // 从ring buffer中index索引处获取一个字符到data中, 这个操作并不会影响ring buffer的内容
12 uint8_t ring_buffer_peek(ring_buffer_t *buffer, char *data,
ring_buffer_size_t index);
13 // 判断ring buffer是否为空
14 inline uint8_t ring_buffer_is_empty(ring_buffer_t *buffer) {
15     return (buffer->head_index == buffer->tail_index);
16 }
17 // 判断ring buffer内容是否已经满了
18 inline uint8_t ring_buffer_is_full(ring_buffer_t *buffer) {
19     return ((buffer->head_index - buffer->tail_index) & RING_BUFFER_MASK) ==
RING_BUFFER_MASK;
20 }
21 // 判断ring buffer剩余的内容大小, 以字节大小返回
22 inline ring_buffer_size_t ring_buffer_num_items(ring_buffer_t *buffer) {
23     return ((buffer->head_index - buffer->tail_index) & RING_BUFFER_MASK);
24 }
```

示例代码

```

1 #include <ringbuffer.h>
2
3 int main(void) {
4     int i, cnt;
5     char buf[50];
6
7     /* 创建循环buffer */
8     ring_buffer_t ring_buffer;
9     ring_buffer_init(&ring_buffer);
10
11    /* 往ring buffer中写入内容 */
12    ring_buffer_queue_arr(&ring_buffer, "Hello, easyAIoT!", 16);
13
14    // 获取ring buffer大小
15    cnt = ring_buffer_num_items(&ring_buffer);
16
17    /* 获取buffer中的内容 */
18    ring_buffer_dequeue_arr(&ring_buffer, buf, sizeof(buf));
19
20    printf("ring buffer size:%d, content:%s\n", cnt, buf);
21
22    return 0;
23
24    return 0;
25 }

```

程序运行结果

```
1 | ring buffer size:16, content:Hello, easyAIoT!
```

3.12 其他接口

3.12.1 时间接口

接口说明

```

1 /**
2 * @brief datetime结构体，通过now获取当前时间，通过format对时间进行格式化输出，输出到
3 pDate中
4 */
5 typedef struct DateTime {
6     time_t (*now)(void);
7     bool (*format)(const char *fmt, char *pDate, int iLen);
8 }DateTime_t;
9
10 /**
11 * @brief datetime对象
12 */
13 extern DateTime_t datetime;

```

示例代码

```
1 // 获取当前时间戳
2 time_t time = datetime.now();
3
4 //获取GMT时间
5 char szDate[32];
6 datetime.format("GMT", szDate, sizeof(szDate));
7
8 printf("now:%d, time:%s\n", time, szDate);
```

程序运行结果

```
1 now:1607614344, time:Thu, 10 Dec 2020 15:32:25 GMT
```

3.12.2 文件接口

接口说明

```
1 /**
2 * @brief 读取文件内容
3 * @param pathname 需要读取的文件路径名称
4 * @return cstring_t* 读取数据存放的字符串结构体指针，读取失败返回NULL
5 */
6 cstring_t * readFile(const char *pathname);

7 /**
8 * @brief 写入文件内容
9 * @param pathname 要写入的文件
10 * @param data 写入的数据
11 * @param len 写入数据的长度
12 * @return size_t 真实写入的大小
13 */
14 size_t writeFile(const char *pathname, void *data, int len);
```

示例代码

```
1 writeFile("test.txt", "easyAIoT", 8);
2 cstring_t * pContent = readFile("test.txt");
3 printf("file content:%s", pContent->str);
```

程序运行结果

```
1 file content:easyAIoT
```

3.12.3 讯飞authorization接口

接口说明

```

1 /**
2 * @brief 构造鉴权参数authorization字段，参考讯飞云相关文档
3 * @param pszAPIKey      用户的APPID
4 * @param pszBaseUrl     请求的URL地址
5 * @param pszRequestLine 请求request-line
6 * @param pszGMTDate    请求GMT时间戳
7 * @param pAuthData     获取到的Auth数据，如果失败数据内容长度为0
8 * @param iLen           pAuthData参数的数组长度
9 */
10 void vGetAuth(const char *pszAPPKey, const char *pszAPPSecret, const char
*pszBaseUrl,
11               const char *pszRequestLine, const char *pszGMTDate, char *pAuthData,
12               int iLen);

```

示例代码

```

1 #include <iFlyAuth.h>
2
3 int main(int argc, char *argv[])
4 {
5     char szDate[64] = {0};
6     char szAuthData[256] = {0};
7
8     // 获取GMT时间戳
9     datetime.format("GMT", szDate, sizeof(szDate));
10
11    // 构造authorization请求参数
12    vGetAuth("apikey", "appsecret", "api.xf-yun.com",
13              "POST /v1/private/s67c9c78c HTTP/1.1", szDate,
14              szAuthData, sizeof(szAuthData));
15    printf("auth:%s\n", szAuthData);
16    return 0;
17 }

```

程序运行结果

```

1 auth:YXBpx2t1et0iYXbpa2V5IiwgYwxnb3JpdGhtPSJobWFjLXNoYTI1NiISIGH1YWRLcnM9Imhv
c3QgZGF0ZSByZXF1ZXN0LwxpbmUiLCBzaWduYXR1cmU9IkNXLzzUwxobVztQVVDRW5oOE9XV2FZa
k4zz2MybXF1NX1DYmUrQzJBRUk9Ig==

```

四、环境搭建

4.1 vscode环境搭建

vscode是微软近几年推出的一款[免费](#)、[开源](#)、[跨平台](#)、资源占用(较)少、架构优良、插件扩展丰富(不会像eclipse一样卡顿)的一款IDE代码编辑、编译、调试一体化的工具，是目前即全面又好用的IDE开发环境。

可能以下的配置过程略显麻烦，我们也提供了一个完整的环境，方便大家直接使用开发代码的protable版本，适用于windows的32和64位的版本：

- ```

1 | 32位下载链接: https://pan.baidu.com/s/1YRwBz-EsxpNGmm1yHk3CLW
2 | 提取码: 1234
3 |
4 | 64位下载链接: https://pan.baidu.com/s/11vZujX_a_0Y1QykP0w4bqw
5 | 提取码: 1234

```

以上只要把解压后的CMake/bin和mingw下的bin目录配置到系统的path中就可以直接使用了。

## 4.1.1 vscode软件下载

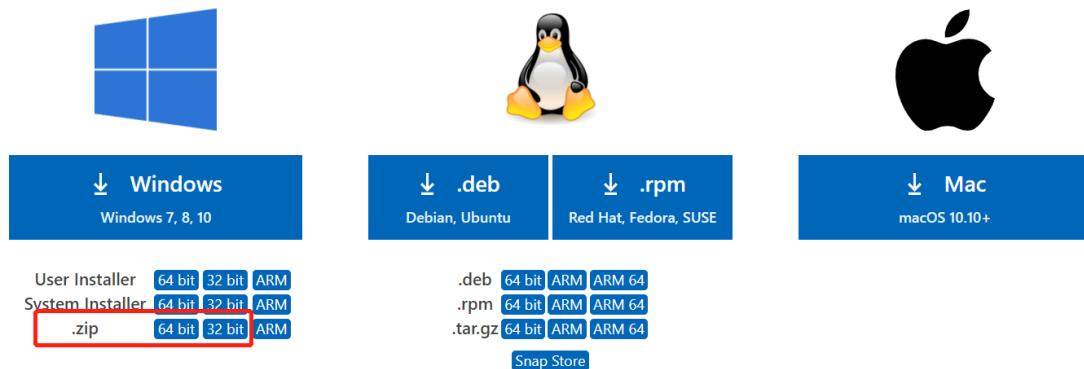
vscode支持windows平台的portable免安装版本，再也不用下载之后还要在系统上安装了，可怜的C盘终于不用塞那么多软件了，如果你喜欢，甚至可以把软件放在U盘里，插到电脑上来使用，更加便捷。

由于是便捷版本，插件下载时必须设置相关目录才能被正常使用，接下来看看怎么使用这款神奇的软件吧。

**便携版下载地址：**

直接在浏览器搜索vscode即可，[点击下载](#)

根据自己的平台选择。windows平台选择zip为免安装版本，根据自己的电脑选择对应的版本，目前台式电脑还是x86的天下（苹果M1芯片将改变x86架构在PC端一统江湖的地位），对于目前电脑都是64bit的了，选择64bit即可。



下载解压就可以用了，但是作为“真正”的便携，我们还需要为它做一些事情，有关描述，vscode开发者，也就是微软，给了详细的[说明](#)，流程如下：

### 1、使能便携模式

Window/Linux用户：

解压zip压缩包，并创建data目录

- ```

1 | |- VSCode-win32-x64-1.50.1
2 |   |- Code.exe (or code executable)
3 |   |- data
4 |   |- ...

```

macOS用户：

解压后，创建code-portable-data目录

```
1 | - visual studio Code.app  
2 | - code-portable-data
```

如果处于隔离模式，需要输入以下信息：

```
1 | xattr -dr com.apple.quarantine Visual\ Studio\ Code.app
```

如果是Insiders（开发版）模式，创建的目录名称是**code-insiders-portable-data**

设置完整的便携模式流程

1. 下载VS Code ZIP包
2. 创建data或者code-portable-data目录
3. 拷贝目前系统上的Code目录到data目录下，并重新命名为user-data：
 - Windows %APPDATA%\Code
 - macOS \$HOME/Library/Application Support/Code
 - Linux \$HOME/.config/Code
4. 拷贝扩展包（下载的插件）目录到data目录下：
 - Windows %USERPROFILE%\.vscode\extensions
 - macOS ~/.vscode/extensions
 - Linux ~/.vscode/extensions

最终的目录结构大概是这样：

```
1 | - VSCode-win32-x64-1.50.1  
2 |   | - Code.exe (or code executable)  
3 |   | - data  
4 |   |   | - user-data  
5 |   |   | - ...  
6 |   |   | - extensions  
7 |   |   | - ...  
8 |   | - ...
```

vscode在运行的过程中会产生一些临时文件，如果你想“收留”它们的话，可以在data目录下创建一个TMP目录用于存放vscode的一些临时数据内容。

4.1.2 插件下载

工欲善其事必先利其器，vscode只是一个IDE环境，丰富的插件才是vscode的灵魂所在。本项目中必选的插件如下：

- **C/C++ IntelliSense**：必选，负责代码编辑、调试、代码浏览。比如好用的代码提示的功能；
- **CMake Tools**：必选，负责CMake工程的配置、创建、构建、调试；
- **Chinese (Simplified) Language Pack**：可选，IDE环境的中文支持；
- **CMake**：可选，编写CMake文件时提供CMake语法支持和提示信息，需要手动写CMake配置文件必备；

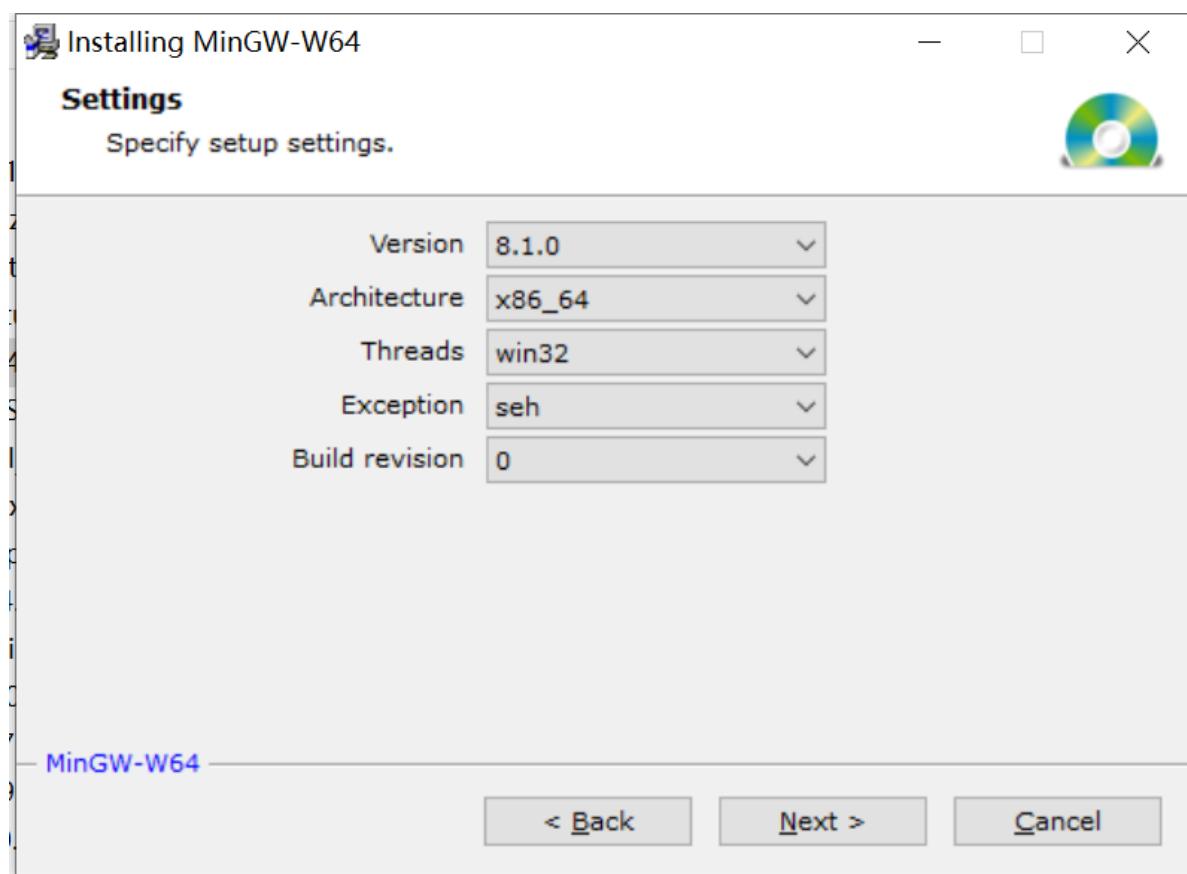
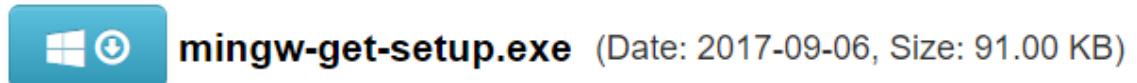
以上这些可以通过左侧侧边栏扩展功能窗口，在搜索框中输入这些插件的名字来下载获得，下载好后，这些插件会存放到上一节创建的data/extensions目录下。快捷键ctrl+shift+x



4.1.3 Mingw工具下载

MinGW (Minimalist GNU for Windows)，主要提供了针对 win32 应用的 GCC等工具集。另外如果想在windows平台模拟Linux的开发环境，也可以使用Cygwin，只适合学习，效率受到了模拟环境的影响。

[下载地址](#)

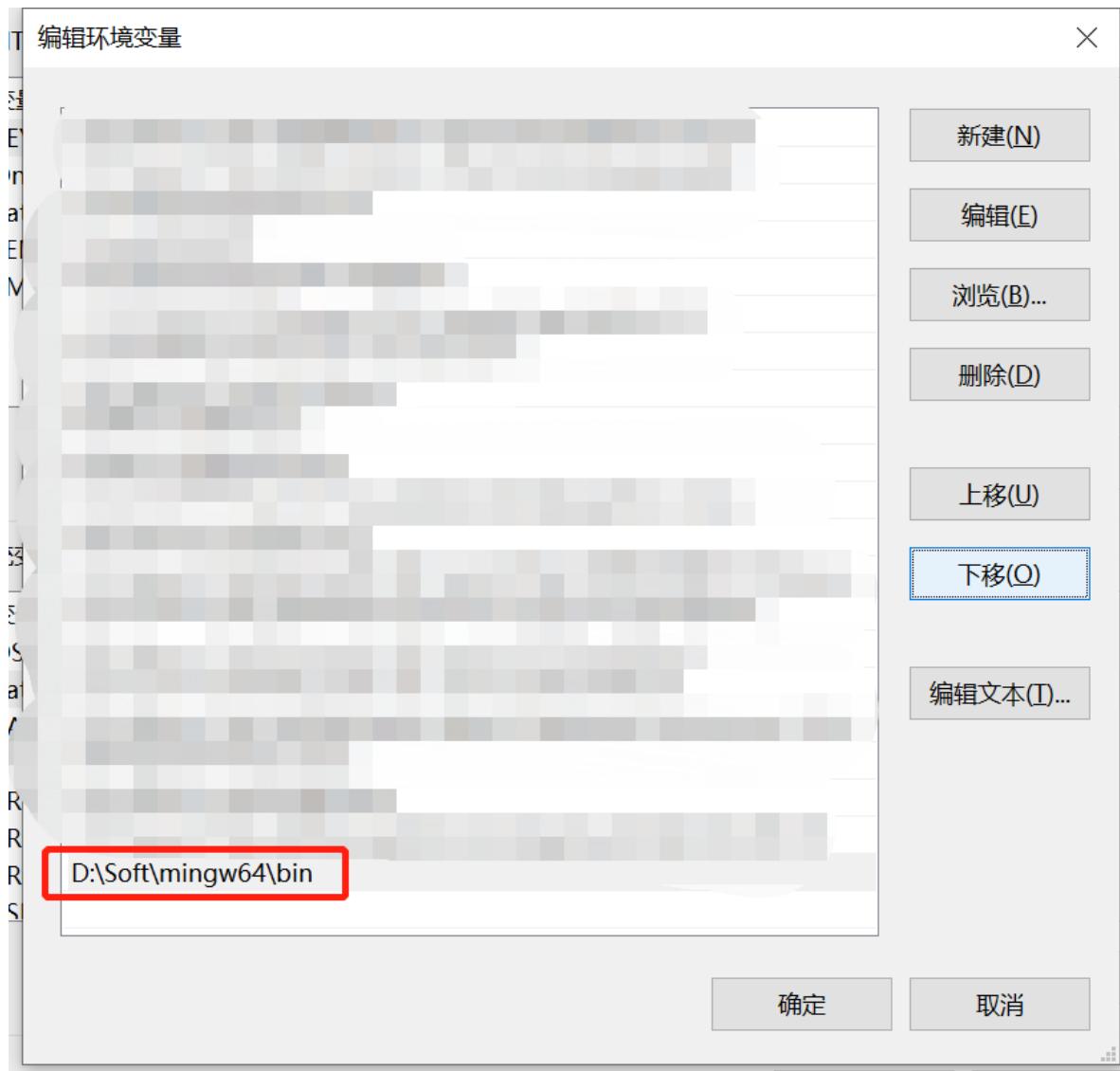


由于线上下载速度可能会比较慢，可以通过以下给定的网盘下来离线版本，这里为了兼容Windows平台的录音库，并没有使用Posix版本。

链接: <https://pan.baidu.com/s/16zyws1R-pNz0QQIGEQggDw>

提取码: 1234

解压之后, 把mingw64\bin目录设置到系统PATH变量中;



设置完成后, 打开cmd命令, 输入gcc -v, 得到如下信息, 表示安装成功。

```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 10.0.18362.1082]
(c) 2019 Microsoft Corporation. 保留所有权利。

C:\Users\AIBIT>gcc -v
Using built-in specs.
COLLECT_GCC= gcc
COLLECT_LTO_WRAPPER=D:/Soft/mingw64/bin/../libexec/gcc/x86_64-w64-mingw32/8.1.0/lto-wrapper.exe
Target: x86_64-w64-mingw32
Configured with: ../../src/gcc-8.1.0/configure --host=x86_64-w64-mingw32 --build=x86_64-w64-mingw32 --target=x86_64-w64-mingw32 --prefix=/mingw64 --with-sysroot=/c/mingw810/x86_64-810-win32-seh-rt_v6-rev0/mingw64 --enable-shared --enable-threads=win32 --enable-static --disable-multilib --enable-languages=c,c++,fortran,lto --enable-libstdcxx-time=yes --enable-libatomic --enable-graphite --enable-checking=release --enable-fuzzy-string --enable-version-specific-runtime-libs --disable-libstdcxx-pch --disable-libstdcxx-debug --enable-bootstrap --disable-rpath --enable-win32-registry --disable-nls --disable-werror --disable-symvers --with-gnu-as --with-gnu-ld --with-arch=native --with-tune=core2 --with-libiconv --with-system=zlib --with-gmp=/c/mingw810/prerequisites/x86_64-w64-mingw32-static --with-mpfr=/c/mingw810/prerequisites/x86_64-w64-mingw32-static --with-mpc=/c/mingw810/prerequisites/x86_64-w64-mingw32-static --with-isl=/c/mingw810/prerequisites/x86_64-w64-mingw32-static --with-pkgversion=x86_64-win32-seh-rev0, Built by MinGW-W64 project' --with-bugurl=https://sourceforge.net/projects/mingw-w64/CFLAGS=-O2 -pipe -fno-ident -I/c/mingw810/x86_64-810-win32-seh-rt_v6-rev0/mingw64/include -I/c/mingw810/prerequisites/x86_64-zlib-static/include -I/c/mingw810/prerequisites/x86_64-w64-mingw32-static/include' CXXFLAGS=-O2 -pipe -fno-ident -I/c/mingw810/x86_64-810-win32-seh-rt_v6-rev0/mingw64/include -I/c/mingw810/prerequisites/x86_64-zlib-static/include -I/c/mingw810/prerequisites/x86_64-w64-mingw32-static/include' CPPFLAGS=-I/c/mingw810/x86_64-810-win32-seh-rt_v6-rev0/mingw64/include -I/c/mingw810/prerequisites/x86_64-w64-mingw32-static/include' LDFLAGS=-pipe -fno-ident -L/c/mingw810/x86_64-810-win32-seh-rt_v6-rev0/mingw64/opt/lib -L/c/mingw810/prerequisites/x86_64-zlib-static/lib -L/c/mingw810/prerequisites/x86_64-w64-mingw32-static/lib'
Thread model: win32
gcc version 8.1.0 (x86_64-win32-seh-rev0, Built by MinGW-W64 project)

C:\Users\AIBIT>
```

4.1.4 cmake工具下载

CMake是自动工程管理工具。当源代码变多的时候，使用GCC需要自己维护编译的过程，通过CMake可以更加方便的管理。

CMake的下载：按照自己的平台下载对应的软件安装即可。

提供额外的链接：<https://pan.baidu.com/s/1XccI2MEFUouxZkQKzEhG3w>

提取码：1234

安装完成后，打开cmd命令行输入"cmake -version"验证cmake工具是否能够正常运行。

```
1 C:\Users\AIBIT>cmake -version
2 cmake version 3.17.5
3
4 CMake suite maintained and supported by Kitware (kitware.com/cmake).
```

看到cmake版本的展示说明软件安装成功。

4.2 编译与调试

一个源代码工程可能由多个源文件组成，并且这些源文件可能由.c/.cpp或者其他语言的编译文件组成，这些源代码怎么编译呢？这就需要一个编译管理的工具，跨平台通用的管理工具叫做Make，我们只需要提供Makefile文件即可，这个文件中描述了我们要怎样编译源代码文件。

接下来的问题是，Makefile本身维护也很复杂，特别在目录多了后。那么有没有一款可以跨平台的代码编译管理工具呢？CMake即是最好的选择，仅仅写一些简单的描述，就会生成编译需要的Makefile文件，从而通过Make编译生成需要的最终程序运行文件。

在VSCode中怎样使用CMake来维护咱们的源代码工程呢？一起来看看吧。

4.2.1 使用CMake管理工程

CMake管理编译工程的流程：



1. CMake环境配置

首先，ctrl+shift+x快捷键打开扩展，在应用商店中搜索CMake Tools，这个插件是VSCode支持CMake功能的插件；

其次，需要安装CMake程序，这个在之前章节中有下载使用说明；

最后，通过ctrl+shift+p打开命令面板，输入cmake:scan for Kits和cmake:select a kit查找编译器和选择编译器

2. CMakeLists.txt配置

CMakeLists.txt是一个描述文件，用于生成Makefile文件以便指导源代码的编译。既然是描述文件就会有自己的语法，官方提供了教程。本文不具体介绍语法，通过项目提供的CMakeLists.txt文件，来说明介绍。

The screenshot shows a code editor with a CMakeLists.txt file open. The file contains several sections of CMake configuration code, each highlighted with a red box and annotated with Chinese comments:

- 版本要求** (Line 1): `cmake_minimum_required(VERSION 3.0.0)`
- 工程名称和版本** (Line 2): `project(EasyAIoT VERSION 0.1.0)`
- 设置C编译环境变量** (Line 4): `set(CMAKE_C_FLAGS "")`
- 设置CPP编译环境变量** (Line 5): `set(CMAKE_CXX_FLAGS "-fpermissive -fPIC")`
- 设置头文件的查找路径** (Line 10): `include_directories(..)`, `include_directories(../Algo)`, `include_directories(../APP)`, `include_directories(../Audio)`, `include_directories(../Common)`, `include_directories(../MBEDTLS)`, `include_directories(../Net)`
- 设置哪些目录的源代码被编译** (Line 17): `aux_source_directory(.. DIR_SRCS)`, `aux_source_directory(../Algo DIR_ALGO_SRCS)`, `aux_source_directory(../APP DIR_APP_SRCS)`, `aux_source_directory(../Audio DIR_AUDIO_SRCS)`
- 根据不同的平台选择性目录编译** (Line 23):
 - 根据不同的平台添加编译链接过程需要依赖的库文件** (Line 44): `IF (WIN32)`, `target_link_libraries(EasyAIoT winmm)`, `target_link_libraries(EasyAIoT ws2_32)`
 - 指定编译生成的可执行程序名称为EasyAIoT，并且后续以空格或者换行给出生成这个可执行程序需要的源代码文件列表** (Line 33): `add_executable(EasyAIoT ${DIR_SRCS} ${DIR_ALGO_SRCS} ${DIR_APP_SRCS} ${DIR_AUDIO_SRCS} ${DIR_AUDIO_SUB_SRCS} ${DIR_COMMON_SRCS} ${DIR_NET_SRCS})`
 - 根据不同的平台添加编译链接过程需要依赖的库文件** (Line 45): `ELSEIF (UNIX)`, `target_link_libraries(EasyAIoT pthread)`
 - 根据不同的平台选择性目录编译** (Line 23): `ENDIF ()`
- 指定编译生成的可执行程序名称为EasyAIoT，并且后续以空格或者换行给出生成这个可执行程序需要的源代码文件列表** (Line 33): `set(CPACK_PROJECT_NAME ${PROJECT_NAME})`, `set(CPACK_PROJECT_VERSION ${PROJECT_VERSION})`, `include(CPack)`
- 根据不同的平台添加编译链接过程需要依赖的库文件** (Line 45): `IF (WIN32)`, `target_link_libraries(EasyAIoT winmm)`, `target_link_libraries(EasyAIoT ws2_32)`
- 根据不同的平台选择性目录编译** (Line 45): `ELSEIF (UNIX)`, `target_link_libraries(EasyAIoT pthread)`
- 根据不同的平台选择性目录编译** (Line 49): `ENDIF ()`

3. Makefile构建

这个步骤的目标是通过CMakeLists.txt生成真正的Makefile工程文件。可以通过ctrl+shift+p命令面板输入cmake:configure，成功后会在当前目录下生成build目录

4. 源代码编译

这个步骤的主要目标是通过Makefile工程文件生成可执行的程序的过程。可以直接通过F7快捷键即可进行编译的过程，也可以通过ctrl+shift+p命令面板输入cmake:build来完成

5. 链接生成可执行文件

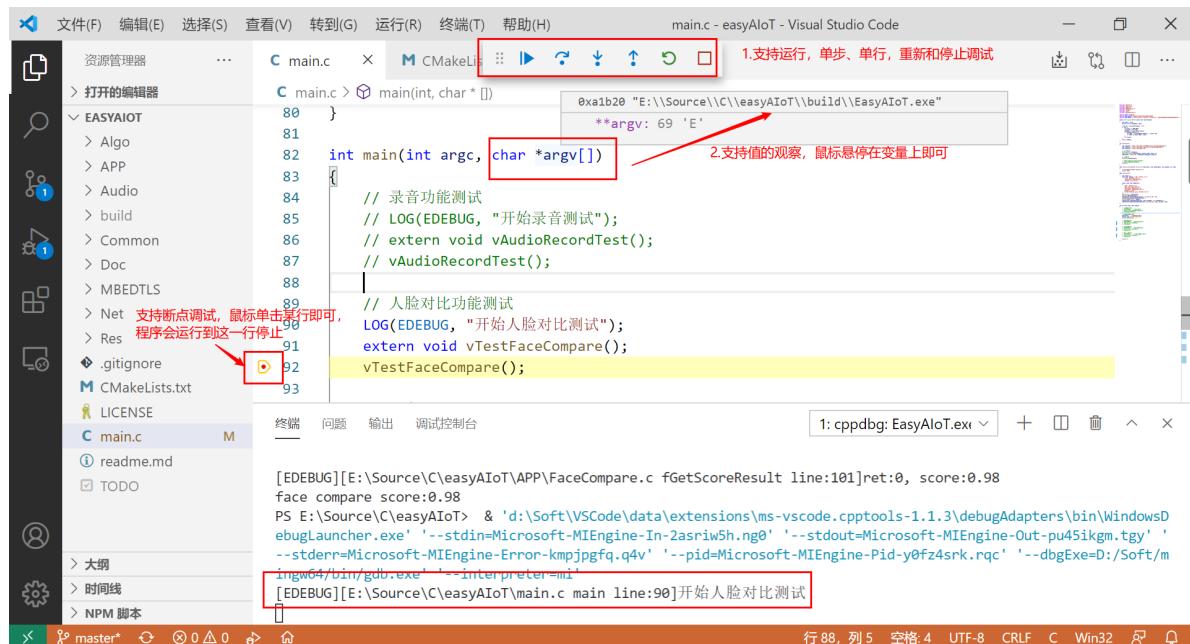
这一个步骤是和上一个步骤同步完成的，但是要注意的是，如果发现undefined reference类似的错误是在这个阶段导致的，需要开发者增加缺失的符号，可能来自自己的代码，也可能是缺少第三方或者系统库的依赖。缺少第三方或系统库可以通过在CMakeLists.txt文件中增加对应的target_link_libraries依赖库文件即可（注意依赖库的名称，比如依赖库的名字为libpthread.so，只需要填写pthread即可，不需要前面的lib，也不需要后面的.so）

6. 程序运行

编译成功后，直接使用VSCode提供Ctrl+F5进行程序的运行调试即可。运行程序会启动内部调试窗口，开发者可通过查看终端来进行代码的简单调试。更加复杂的调试可参考后面的内容

4.2.2 程序的调试过程

程序的调试非常重要，不仅能够帮助开发者找到程序的BUG，也是分析代码最重要的手段，强烈建议初学者就应该学习掌握。VSCode提供了非常简单直观的调试界面，通过调试的过程图来说明。



一般的调试流程：

- 设置断点
- 运行程序
- 断点处单步执行程序，观察变量值的变化

从而发现问题和解决问题的过程

五、AI能力应用开发流程

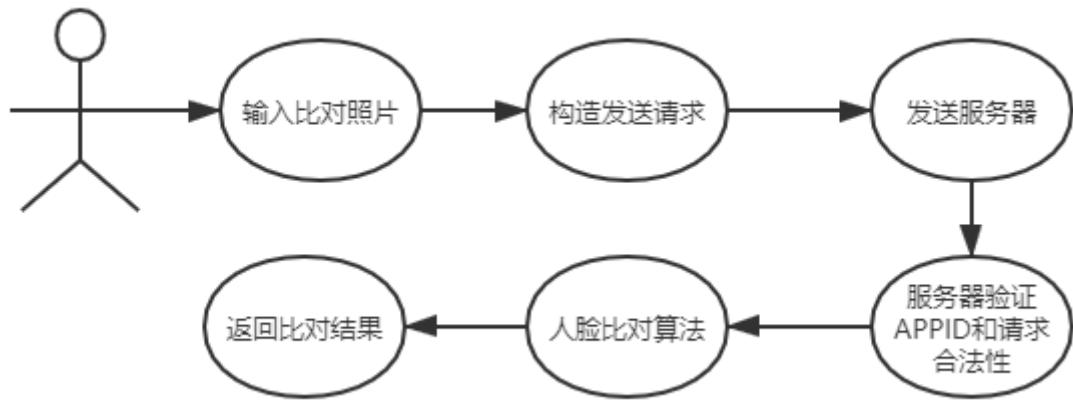
以人脸比对作为说明。

5.1 开发流程

项目开发对于研发人员一般分为需求确认、技术调研、系统程序设计、软件编码、测试验收等环节。

以人脸比对应用举例，需求就是**完成一个基本的人脸比对应用，能够比较给定的两张照片相似度，具体形式不限制**。为了完成这个需求，需要做技术上的调研，分析抽象出人脸比对由哪些基础技术组成，再一个个的攻克这些基础问题。

用例图



使用到的技术

输入比对照片：文件读取（两张照片读取）；

构造发送请求：NTP时间、Base64加解密、MD5加密、URL加解密；

发送服务器：HTTP客户端Post请求；

服务器验证APPID和请求合法性：提供APPID、APIKEY、APP PARAM、Json序列化；

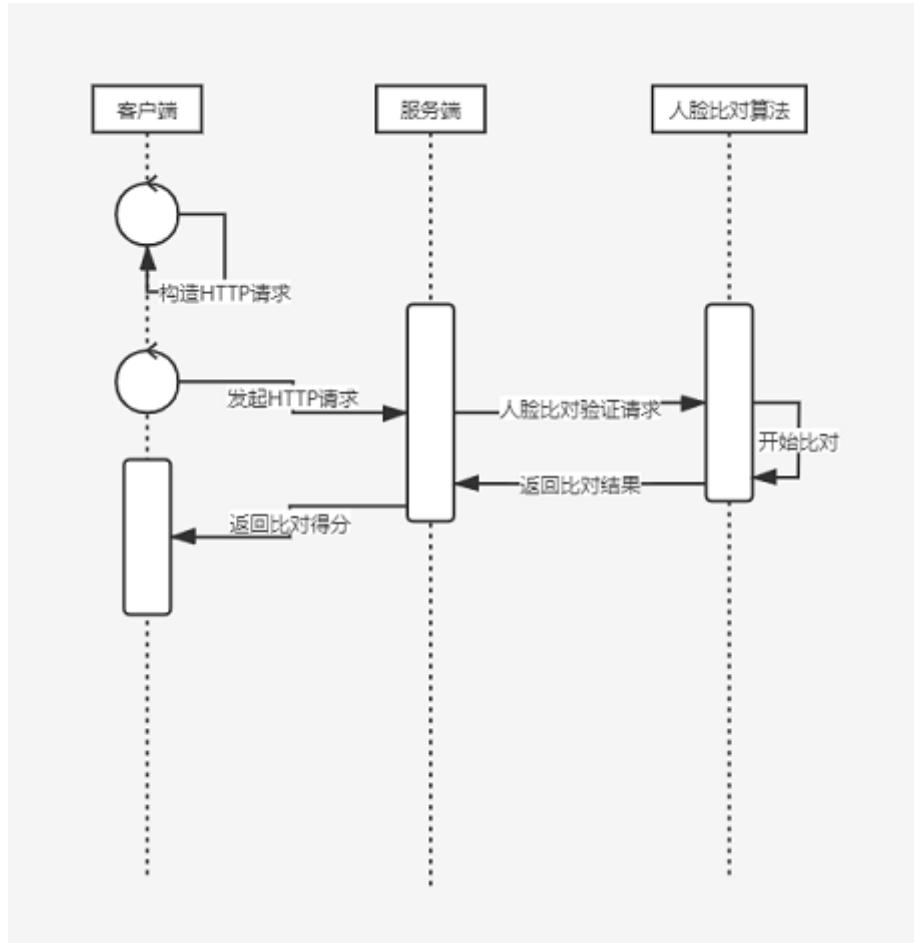
人脸比对算法：获得HTTP人脸比对服务的接口URL；

返回比对结果：JSON数据反序列化；

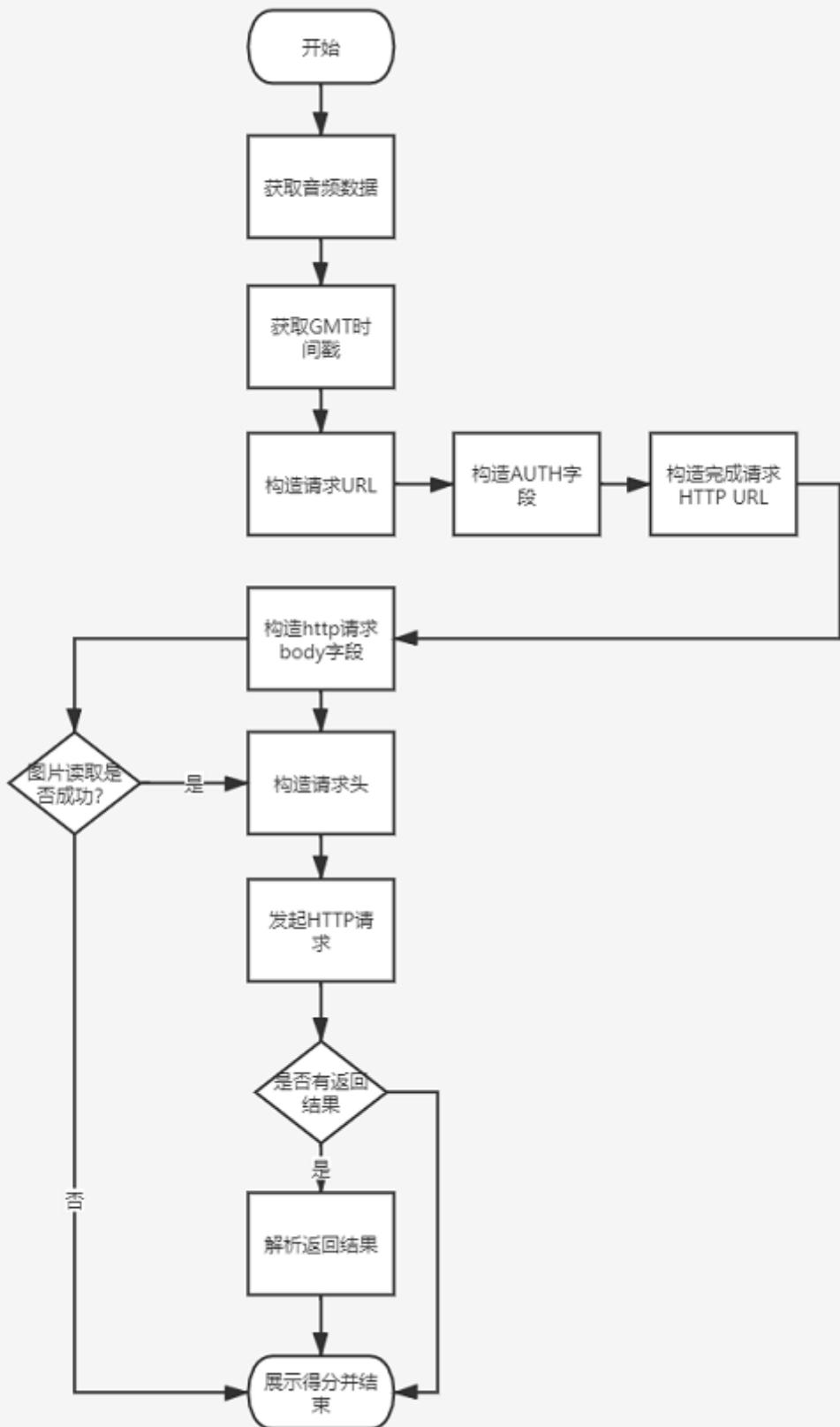
5.2 程序流程图

以上明确了人脸比对应用到的技术和问题，这里假设已经掌握了这些基础技术，那么怎么组合这些技术，按照一定的流程来完成需求呢？来看一下整体的程序时序和流程图

人脸比对时序图



人脸比对程序流程图



5.3 编码

针对目标的时序和流程有了，流程涉及到的知识也掌握了，接下来就可以通过编码的方式应用这些技术点完成这个流程，从而完成需求的过程。

```

1 double fFaceCompare(const char *pszAPPID,
2                     const char *pszAPPSecret, const char *pszAPPKey,
3                     const char *pszImagePath1, const char *pszImagePath2)

```

```

4  {
5      double fScore = 0.0f;
6      char szDate[64];
7      char szAuthData[512];
8      char szFullUrl[1024];
9      char *szBaseUrl = "http://api.xf-yun.com/v1/private/s67c9c78c?
host=%s&date=%s&authorization=%s";
10
11     // 构造authorization请求参数
12     datetime.format("GMT", szDate, sizeof(szDate));
13     vGetAuth(pszAPPKey, pszAPPSecret, "api.xf-yun.com",
14             "POST /v1/private/s67c9c78c HTTP/1.1", szDate,
15             szAuthData, sizeof(szAuthData));
16
17     // 构造完整的请求URL
18     snprintf(szFullUrl, sizeof(szFullUrl), szBaseUrl, "api.xf-yun.com",
19             szDate, szAuthData);
20     LOG(EDEBUG, "url:%s", szFullUrl);
21
22     // 构造body数据
23     cstring_t *pBody = pstGenBody(pszAPPID, pszImagePath1, pszImagePath2);
24     if (!pBody) {
25         LOG(ERROR, "empty body error");
26         return .0f;
27     }
28     // LOG(EDEBUG, "body:#%s#", pBody->str);
29
30     // 构造需要的额外HTTP头
31     const char szHeader[] = {
32         "content-type: application/json\r\n"
33     };
34
35     // 发起HTTP Post语音识别的请求
36     SEIHttpInfo_t stHttpInfo;
37     bConnectHttpServer(&stHttpInfo, szFullUrl, szHeader,
38                         (void *)pBody->peek(pBody), (int)pBody-
39                         >length(pBody));
40     LOG(EDEBUG, "status:%d", stHttpInfo.stResponse.iStatus);
41     LOG(EDEBUG, "body:%s", stHttpInfo.stResponse.pstBody->sBuffer);
42
43     // 获得人脸对比得分
44     fScore = fGetScoreResult(stHttpInfo.stResponse.pstBody->sBuffer);
45
46     // 关闭HTTP
47     bHttpClose(&stHttpInfo);
48     cstring_del(pBody);
49
50     return fScore;
51 }
```

5.4 测试及结果

程序是否按照既定的流程走了呢？有没有可能在不同的场景下会偏离流程，而出现不可预知的结果呢？测试在这里就起到了这样的作用：列举出用户使用的所有可能的场景，对场景的流程进行列举，再通过这些场景的流程去验证程序的流程正确性的过程。

测试过程可以选择

1. 不同角度的人脸照片；
2. 不同分辨率图片大小；
3. 不同明暗度的图片；
4. 不同格式的图片；
5. 扩展自己的测试用例；



可能是同一个人



[上传本地图片](#)

[上传本地图片](#)