

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/portable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/Sha256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

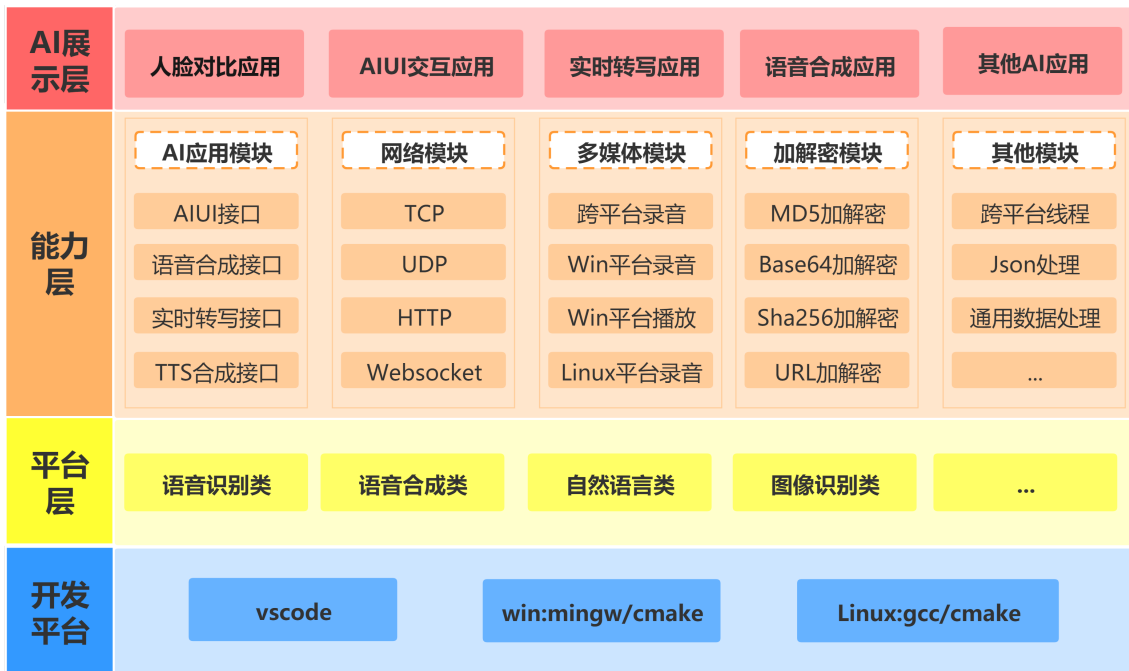
下图是easyAloT项目的分层框架图。

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
    *pszParam,
11                        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 = "xxxxxxx";
5      const char *pszKey = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
6      const char *pszParam = "
      {\"result_level\":\"plain\", \"auth_id\":\"xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx\",
      , \"data_type\":\"audio\", \"sample_rate\":\"16000\", \"scene\":\"main_box\"}";
7
8      // 1.读取语音文件
9      cstring_t *pAudioData = readFile("../Res/test.pcm");
10     if (!pAudioData)
11     {
12         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 cstring_del(pAudioData);
26
27 return 0;
28 }

```

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": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx", //请求的auth_id，平台配置界面
    中查询
4     "data_type": "audio", // 请求的数据类型，这里是语音，也可以为文本
5     "sample_rate": "16000", // 如果请求的是audio，sample_rate表示音频采样率
6     "scene": "main_box" // 请求场景，未发布版本，需要增加"_box"后缀
7 }

```

程序运行结果：

```

E_SEARCH", "score": 0.920714020729648, "slots": [{"begin": 0, "end": 2, "name": "cityMix", "normValue": "全国", "value": "全国"}, {"begin": 2, "end": 4, "name": "add", "normValue": "新增", "value": "新增"}, {"begin": 4, "end": 6, "name": "type", "normValue": "确诊", "value": "确诊"}], "template": "{cityMix}{add}{type}{suffix}", "semanticType": 1, "service": "AIUI.virusSearch", "sessionIsEnd": true, "shouldEndSession": true, "sid": "ara02e14c3c@dx0001132a4c1a094000", "state": null, "text": "今天全国新冠肺炎新增确诊病例15例", "uid": "ara02e14c3c@dx0001132a4c1a094000", "vendor": "AIUI", "version": "79.0", "voice_answer": [{"content": "今天全国新冠肺炎新增确诊病例15例", "type": "TTS"}], "result_id": 1, "sid": "ara02e14c3c@dx0001132a4c1a094000", "code": "0", "desc": "success"}
[EDEBUG][E:\Source\C\easyAIoT\main.c main line:99]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。



3.3 TTS语音合成模块功能说明

接口说明

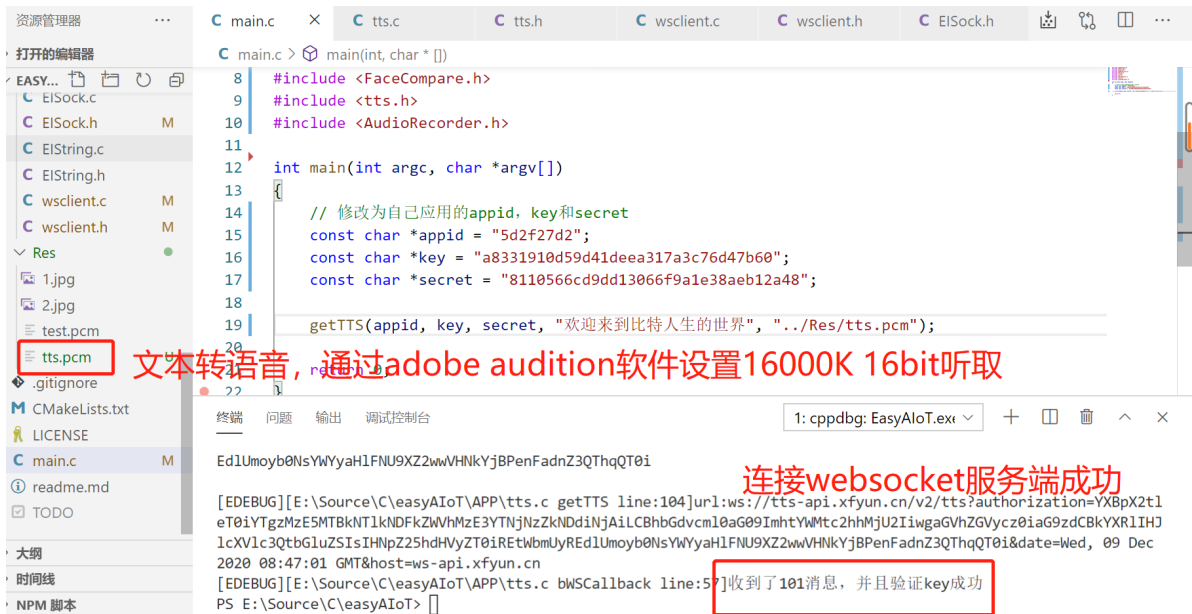
```
1  /**
2   * @brief 文字转语音
3   * @param appid 讯飞appid
4   * @param key 讯飞api key
5   * @param param 讯飞合成参数
6   * @param text 合成的文本内容
7   * @param pathname 合成的结果音频文件路径名称
8   * @return true 合成成功
9   * @return false 合成失败
10  */
11 bool getTTS(const char *appid, const char *key, const char *param,
12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

生成的语音文件放在了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。



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 = "xxxxxxx";
5      const char *key = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
6      const char *secret = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
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。



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
9   大
10  */
11 double fFaceCompare(const char *pszAPPID,
12                     const char *pszAPPSecret, const char *pszAPPKey,
13                     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 = "5d2f27d2";
5      const char *key = "a8331910d59d41deea317a3c76d47b60";
6      const char *secret = "8110566cd9dd13066f9a1e38aeb12a48";
7
8      double fScore = fFaceCompare(appid, secret, key, "../Res/1.jpg",
9                                  "../Res/2.jpg");
9      printf("face compare score: %.2f\n", fScore);
10
11      return 0;
12  }
```

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:{
4      "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
   *pvUserData, void *pvData, int iLen);
5
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
   nPort);
18     bool (*bConnect)(struct SsockClient *pstClient);
19     void (*vClose)(struct SsockClient *pstClient);
20     int (*iSend)(struct SsockClient *pstClient, void *pvData, int iLen);
21     bool (*bEventLoop)(struct SsockClient *pstClient, fnSockCallback cb,
   void *pvUserData, int *piLoop, int iTimeout);
22 }SsockClient_t;
23

```

```

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  /**
33   * @brief 网络功能反初始化
34   * @param pstClient      网络功能对象
35   * @return true           网络反初始化成功
36   * @return false         网络反初始化成功
37   */
38  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
   *pvData, int iLen)
3  {
4      printf("bSockCallback called:%s\r\n", pvData);
5      return true;
6  }
7
8  int main(int argc, char *argv[])
9  {
10     int iLoop = 1;
11
12     // 请求头信息
13     const char header[] =
14     {
15         "GET / HTTP/1.1\r\n"
16         "Host: www.baidu.com\r\n"
17         "Connection: keep-alive\r\n"
18         "Accept: */*\r\n\r\n";
19
20     // 声明网络结构体变量
21     SSockClient_t stSockClient;
22
23     // 初始化网络结构体
24     bSockInit(&stSockClient);
25
26     // 创建socket连接对象
27     stSockClient.bCreate(&stSockClient, "14.215.177.39", 80);
28
29     // 连接服务端
30     stSockClient.bConnect(&stSockClient);
31
32     // 发送数据给服务端
33     stSockClient.iSend(&stSockClient, (void *)header, strlen(header) + 1);
34
35     // 等待网络事件，通过bSockCallback处理网络事件
36     stSockClient.bEventLoop(&stSockClient, bSockCallback, NULL, &iLoop,
1000);
37

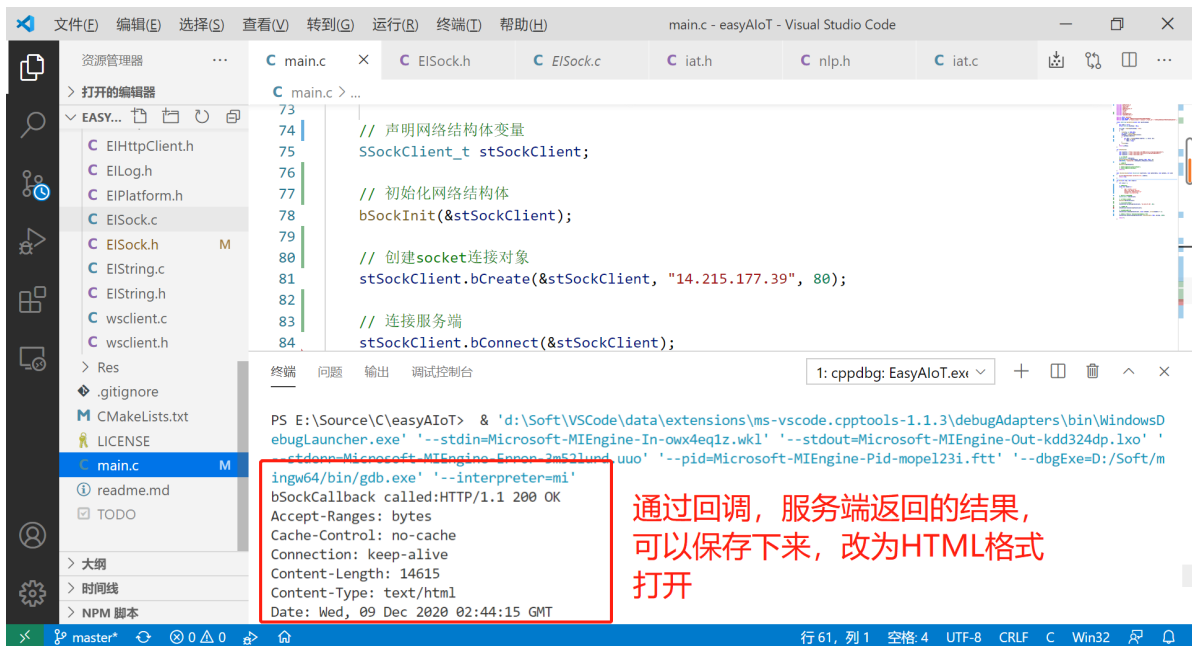
```

```

38     return 0;
39 }

```

程序运行结果



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解析四个功能
5   * @param pszUrl 访问的http url
6   * @param pszExtraHeader 额外的头文件
7   * @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
8   * @param isize body的长度
9   *
10  * @return 调用成功返回true，否则返回false
11  *
12  * @note 注意返回结果在pstHttpInfo的stResponse成员中，并且调用完成后
13  *        要通过bHttpClose来释放pstHttpInfo中的资源
14  */
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
11      bHttpClose(&stHttpInfo);
12
13      return 0;
14  }

```

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

```

终端 问题 输出 调试控制台 1: cppdbg: EasyAloT.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

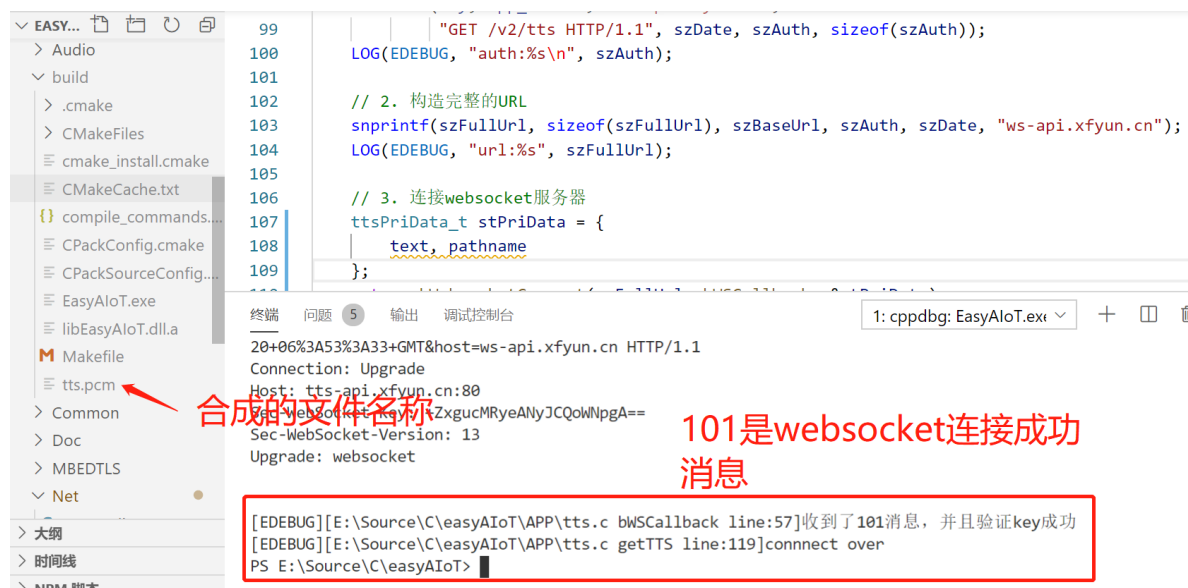
```

1  int main(int argc, char *argv[])
2  {
3      const char *appid = "xxxxxxx";
4      const char *app_secret = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
5      const char *app_key = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
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 audition软件，设置为16000K 16bit听声音是否和文本内容一致。



```
99 "GET /v2/tts HTTP/1.1", szDate, szAuth, sizeof(szAuth));
100 LOG(EDEBUG, "auth:%s\n", szAuth);
101
102 // 2. 构造完整的URL
103 snprintf(szFullUrl, sizeof(szFullUrl), szBaseUrl, szAuth, szDate, "ws-api.xfyun.cn");
104 LOG(EDEBUG, "url:%s", szFullUrl);
105
106 // 3. 连接websocket服务器
107 ttsPriData_t stPriData = {
108     text, pathname
109 };
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653

```

```

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
      // 是传递给回调的参数
4      AudioConfig_t stAudioConfig = {16000, 16, 1, RecordCB,
      (void*)"test.pcm"};
5      // 打开音频设备
6      Recorder.open(&stAudioConfig);
7      // 开始录音, 阻塞函数, 需要关闭录音在其他地方调用Recorder.close()
8      Recorder.start();
9      关闭录音设备
10     Recorder.close();
11
12     return 0;
13 }

```

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*
pvData, int32_t iLen)
2  {
3      uint32_t byteswritten = 0;
4      static uint32_t totalWritten = 0;
5      const char *pathname = (const char *)pvUserData;
6      static FILE* fp = NULL;
7      if (!fp) {
8          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 /**
12  * @brief 线程创建
13  * @param iThreadNum 线程创建的个数
14  * @param funArray 线程处理函数组
15  */
16 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 doStomething 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
6   *                          个字节使用16进制转化为字符串
7   */
7 void vMD5(const unsigned char *msg, unsigned int len, unsigned char
  *digest);
8
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
  *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-> d1428978a06bcefed0e94ab70a5d1498

```

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:ZWFzeUFJb1Q=, 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
char *key, int len_key, unsigned char *out);
18

```

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      3&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;
15 }

```

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",
   "mandarin");
8
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",
   "audio/L16;rate=16000");
12 JSON_SERIALIZE_ADD_STRING_TO_OBJECT(json_data_obj, "encoding", "raw");
13 JSON_SERIALIZE_ADD_STRING_TO_OBJECT(json_data_obj, "audio",
   "exSI6ICJlbiIsCgkgICAgInBvc2l0aw9uIjogImZhbnHNlIgoJf");
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",
   json_common_obj);
18 JSON_SERIALIZE_ADD_OBJECT_TO_OBJECT(json_frame_obj, "business",
   json_business_obj);
19 JSON_SERIALIZE_ADD_OBJECT_TO_OBJECT(json_frame_obj, "data", json_data_obj);
20 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": "exSI6ICJ1biIsCgkgICAgInBvc2l0aw9uIjogImZhbHNlIgoJf"
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
   程序跳出方式
5  #define JSON_DESERIALIZE_GET_INT(json_doc, key, value, ret, jump)
6
7  // 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
   为程序跳出方式
8  #define JSON_DESERIALIZE_GET_DOUBLE(json_doc, key, value, ret, jump)
9
10 // 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
   为程序跳出方式
11 #define JSON_DESERIALIZE_GET_STRING(json_doc, key, value, ret, jump)
12
13 // 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
   为程序跳出方式
14 #define JSON_DESERIALIZE_GET_STRING_COPY(json_doc, key, value, len, ret,
   jump)
15
16 // 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,
   jump为程序跳出方式
17 #define JSON_DESERIALIZE_GET_ARRAY(json_doc, key, value, ret, jump)
18
19 // 根据json对象json_doc生成sub_item迭代器
20 #define JSON_DESERIALIZE_ARRAY_FOR_EACH_START(json_doc, sub_item, pos,
   total)
21
22 // 数组迭代结束标识
23 #define JSON_DESERIALIZE_ARRAY_FOR_EACH_END()
24
25 // 获取json_doc对象中的Json对象, 键值为key, 值放到value中, 正确ret值为0, 否则为负值,
   jump为失败后的跳转方式
26 #define JSON_DESERIALIZE_GET_OBJECT(json_doc, key, value, ret, jump)
27
28 // Json反序列结束标识
29 #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":
5             "ewoJInJldCIGOiAwLAoJInNjb3JlIiA6IDAuOTk2MTg2MDC3NTk0Nzu3MDgKfQo="
6         }
7     }
8 }

```

反序列化代码

```

1
2 JSON_DESERIALIZE_START(json_root, szResponse, iRet);
3     JSON_DESERIALIZE_GET_OBJECT(json_root, "payload", payload_obj, iRet,
4     JSON_CTRL_BREAK);
5     JSON_DESERIALIZE_GET_OBJECT(payload_obj, "face_compare_result",
6     result_obj, iRet, JSON_CTRL_BREAK);
7     JSON_DESERIALIZE_GET_STRING(result_obj, "text", pText, iRet,
8     JSON_CTRL_NULL);
9     if (pText) text->appendStr(text, pText, strlen(pText));
10    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的值存放到了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 allocated;
13     // 动态分配
14     size_t len;
15     // 字符串长度
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
14    should be called when we're finished with a map.
15
16 // 获取给定key的map值，返回的是指向值得指针
17 map_get(m, key)
18
19 // 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
20 map_set(m, key, value)
21
22 // 把指定为key的数据从map中移除
23 map_remove(m, key)
24
25 // 获取map迭代器，返回一个map_iter_t类型，可以通过map_next遍历
26 map_iter(m)
27 Returns a map_iter_t which can be used with map_next() to iterate all the
28    keys in the map.
29
30 // 通过map_next迭代map中的数据
31 map_next(m, iter)
32 Uses the map_iter_t returned by map_iter() to iterate all the keys in the
33    map. map_next() returns a key with each call and returns NULL when there are
34    no more keys.
35
36 const char *key;
37 map_iter_t iter = map_iter(&m);
38
39 while ((key = map_next(&m, &iter))) {
40     printf("%s -> %d", key, *map_get(&m, key));
41 }
```

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  /**
2   * Initializes the ring buffer pointed to by <em>buffer</em>.
3   * This function can also be used to empty/reset the buffer.
4   * @param buffer The ring buffer to initialize.
5   */
6  void ring_buffer_init(ring_buffer_t *buffer);
7
8  /**
9   * Adds a byte to a ring buffer.
10   * @param buffer The buffer in which the data should be placed.
11   * @param data The byte to place.
12   */
13  void ring_buffer_queue(ring_buffer_t *buffer, char data);
14
15  /**
16   * Adds an array of bytes to a ring buffer.
17   * @param buffer The buffer in which the data should be placed.
18   * @param data A pointer to the array of bytes to place in the queue.
19   * @param size The size of the array.
20   */
21  void ring_buffer_queue_arr(ring_buffer_t *buffer, const char *data,
22                             ring_buffer_size_t size);
23
24  /**
25   * Returns the oldest byte in a ring buffer.
26   * @param buffer The buffer from which the data should be returned.
27   * @param data A pointer to the location at which the data should be placed.
28   * @return 1 if data was returned; 0 otherwise.
29   */
30  uint8_t ring_buffer_dequeue(ring_buffer_t *buffer, char *data);
31
32  /**

```

```

32  * Returns the <em>len</em> oldest bytes in a ring buffer.
33  * @param buffer The buffer from which the data should be returned.
34  * @param data A pointer to the array at which the data should be placed.
35  * @param len The maximum number of bytes to return.
36  * @return The number of bytes returned.
37  */
38  ring_buffer_size_t ring_buffer_dequeue_arr(ring_buffer_t *buffer, char
39  *data, ring_buffer_size_t len);
40  /**
41  * Peeks a ring buffer, i.e. returns an element without removing it.
42  * @param buffer The buffer from which the data should be returned.
43  * @param data A pointer to the location at which the data should be placed.
44  * @param index The index to peek.
45  * @return 1 if data was returned; 0 otherwise.
46  */
47  uint8_t ring_buffer_peek(ring_buffer_t *buffer, char *data,
48  ring_buffer_size_t index);
49  /**
50  * Returns whether a ring buffer is empty.
51  * @param buffer The buffer for which it should be returned whether it is
52  empty.
53  * @return 1 if empty; 0 otherwise.
54  */
55  inline uint8_t ring_buffer_is_empty(ring_buffer_t *buffer) {
56  return (buffer->head_index == buffer->tail_index);
57  }
58  /**
59  * Returns whether a ring buffer is full.
60  * @param buffer The buffer for which it should be returned whether it is
61  full.
62  * @return 1 if full; 0 otherwise.
63  */
64  inline uint8_t ring_buffer_is_full(ring_buffer_t *buffer) {
65  return ((buffer->head_index - buffer->tail_index) & RING_BUFFER_MASK) ==
66  RING_BUFFER_MASK;
67  }
68  /**
69  * Returns the number of items in a ring buffer.
70  * @param buffer The buffer for which the number of items should be
71  returned.
72  * @return The number of items in the ring buffer.
73  */
74  inline ring_buffer_size_t ring_buffer_num_items(ring_buffer_t *buffer) {
75  return ((buffer->head_index - buffer->tail_index) & RING_BUFFER_MASK);
76  }

```

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
   pDate中
3   */
4  typedef struct DateTime {
5      time_t (*now)(void);
6      bool (*format)(const char *fmt, char *pDate, int iLen);
7  }DateTime_t;
8
9  /**
10   * @brief datetime对象
11   */
12  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 /**
9  * @brief 写入文件内容
10 * @param  pathname      要写入的文件
11 * @param  data          写入的数据
12 * @param  len           写入数据的长度
13 * @return size_t        真实写入的大小
14 */
15 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

示例代码

程序运行结果

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。

4.1.1 vscode软件下载

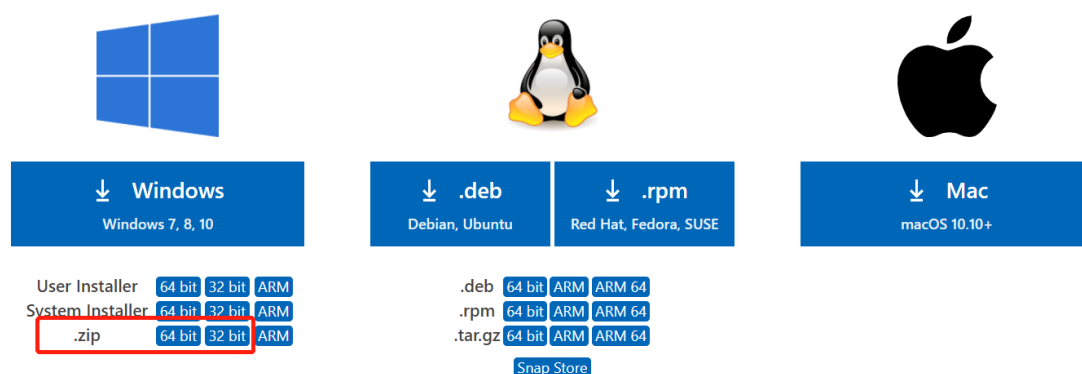
vscode支持windows平台的portable免安装版本，再也不用下载之后还要在系统上安装了，可怜的C盘终于不用塞那么多软件了，如果你喜欢，甚至可以把软件放在U盘里，插到电脑上来使用，更加便捷。

由于是便携版本，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 |   |- ...
```

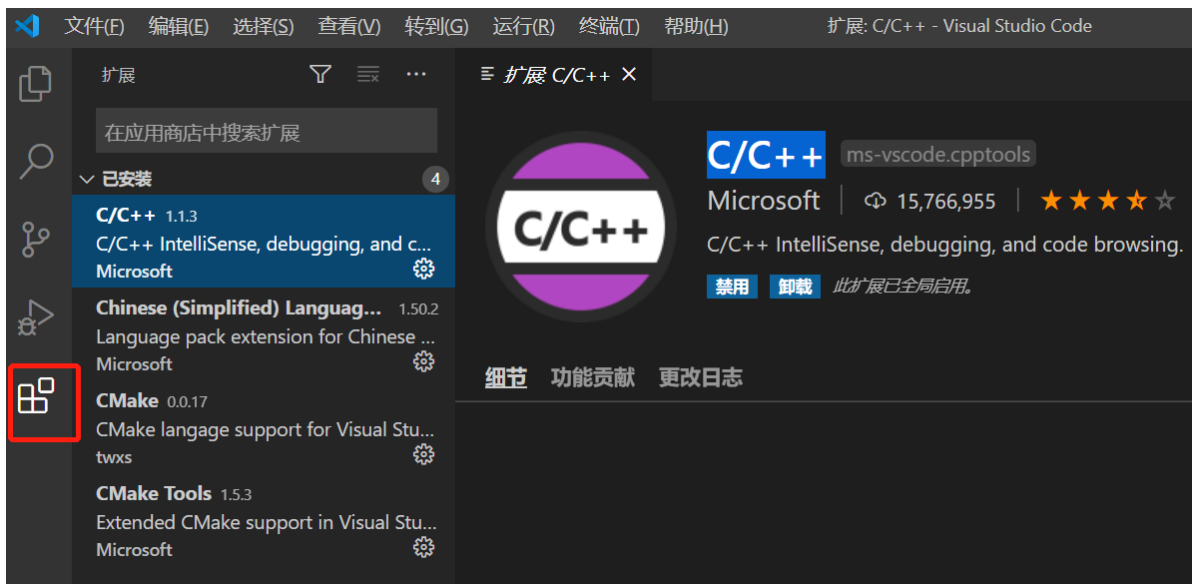
vscode在运行的过程中会产生一些临时文件，如果你想“收留”它们的话，可以在data目录下创建一个TMP目录用于存放vscode的一些临时数据内容。

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;

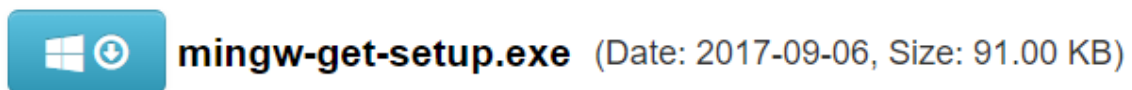
以上这些可以通过左侧侧边栏扩展功能窗口，在搜索框中输入这些插件的名字来下载获得，下载好后，这些插件会存放 to 上一节创建的数据/extensions目录下。快捷键ctrl+shift+x



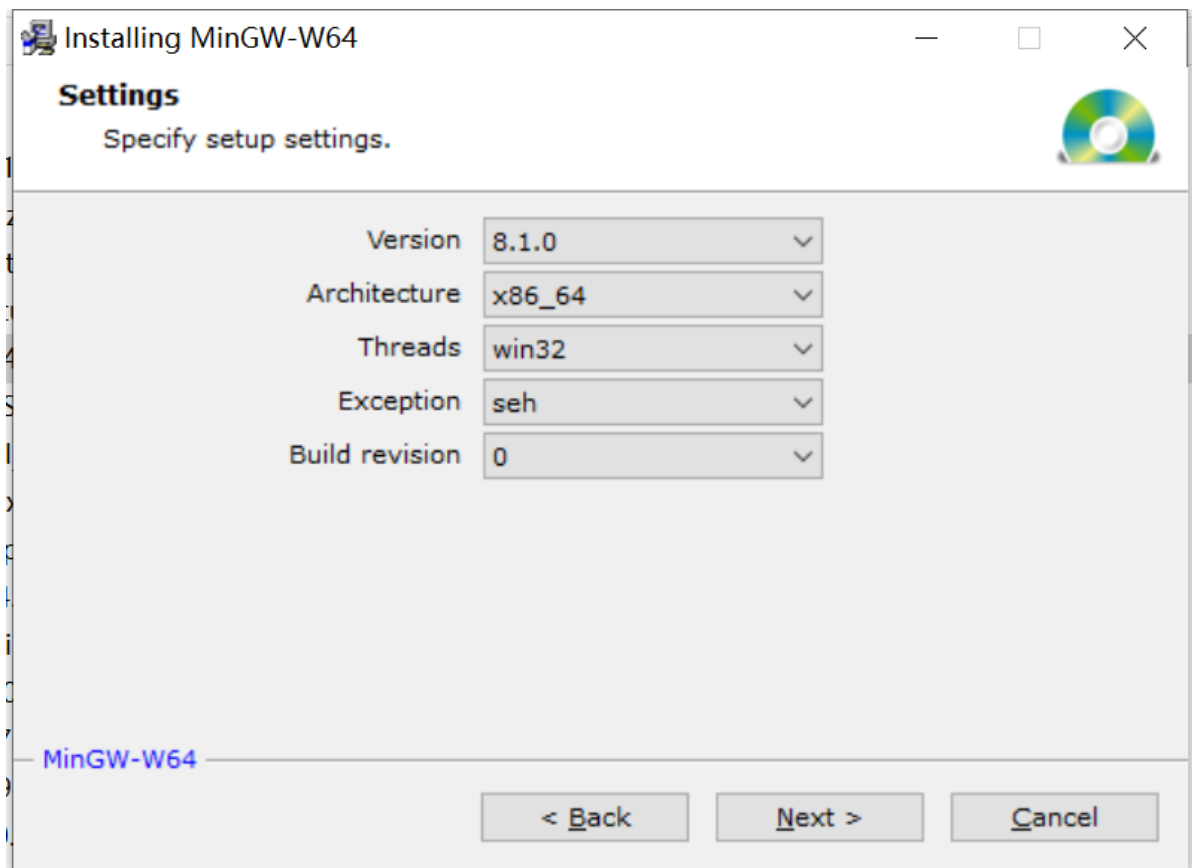
4.1.3 Mingw工具下载

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](#)



它是一个MinGW的在线下载工具, 可以在线下载需要的组件, 选择的配置信息如下:

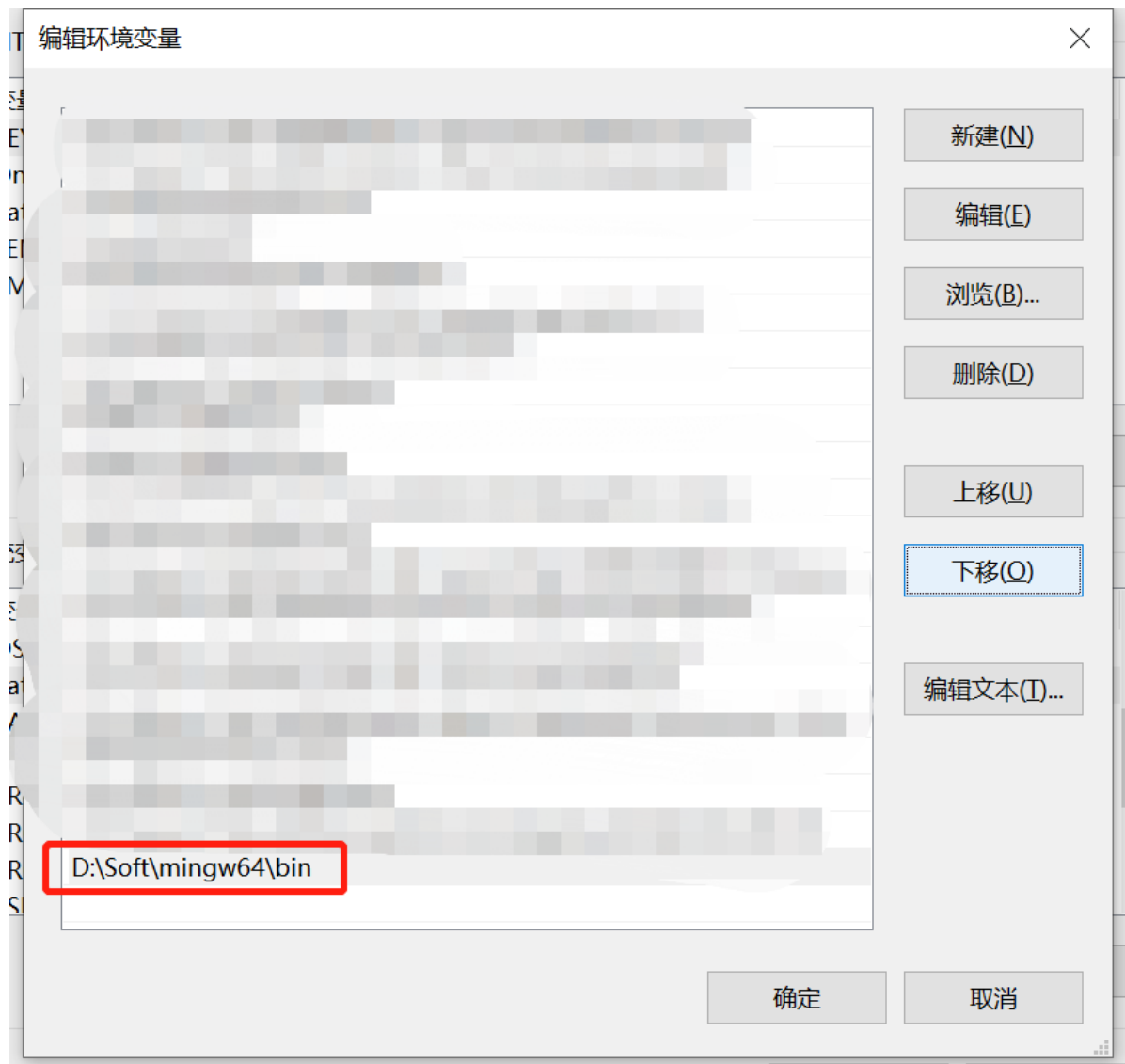


由于线下载速度可能会比较慢, 可以通过以下给定的网盘下来离线版本, 这里为了兼容Windows平台的录音库, 并没有使用Posix版本。

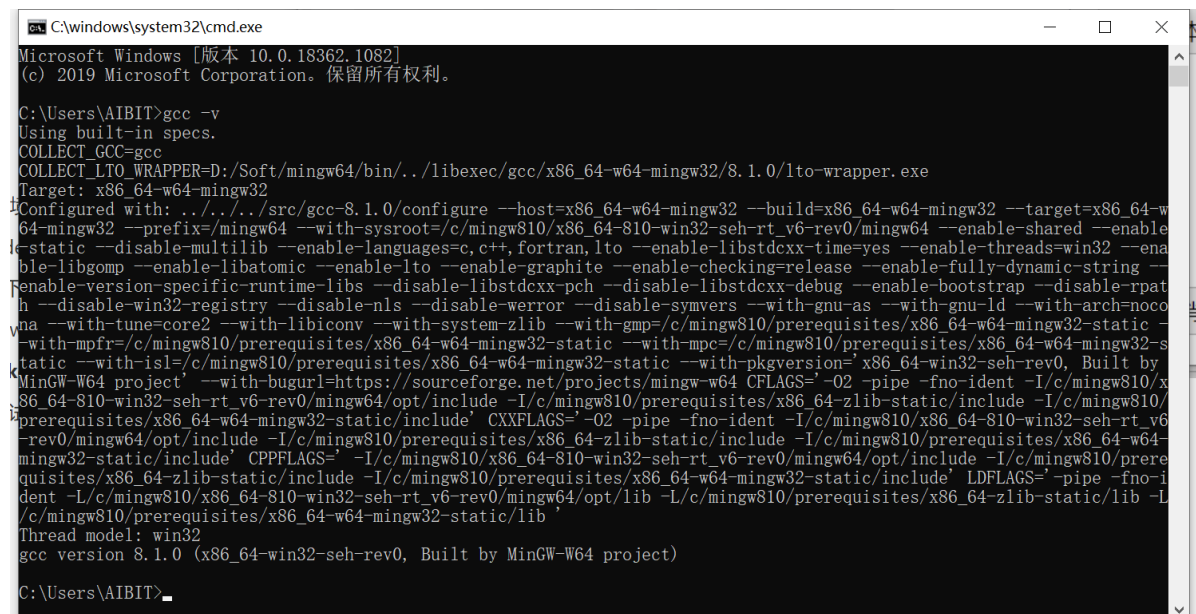
链接: <https://pan.baidu.com/s/16zvw51R-pNZoQQIGEQggDw>

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。



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/1Xccl2MEFUouxZkQKzEhG3w>

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Makfile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

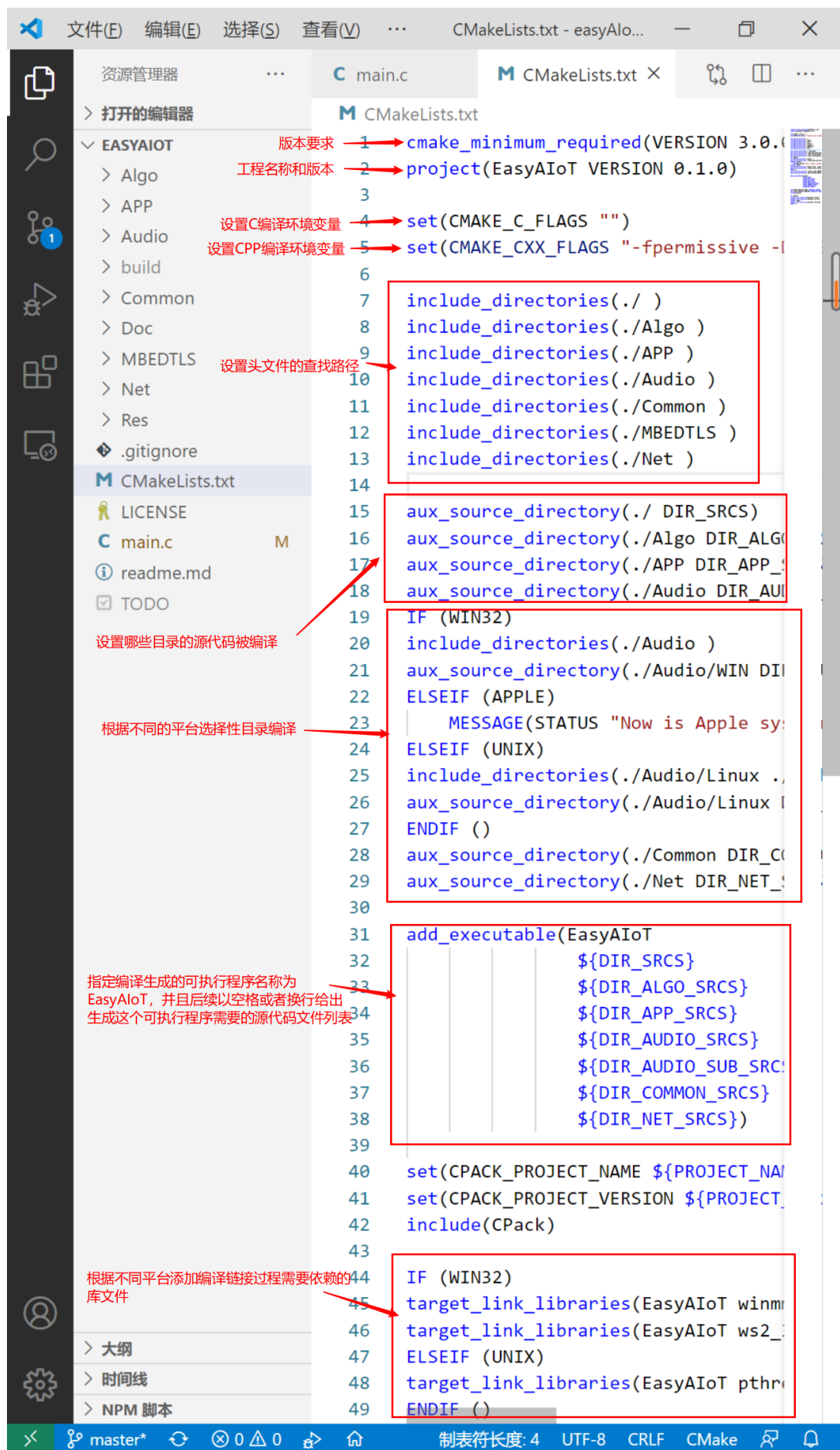
首先，ctrl+shift+x快捷键打开扩展，在应用商店中搜索CMake Tools，这个插件是VSCode支持CMake功能的插件；

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。



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

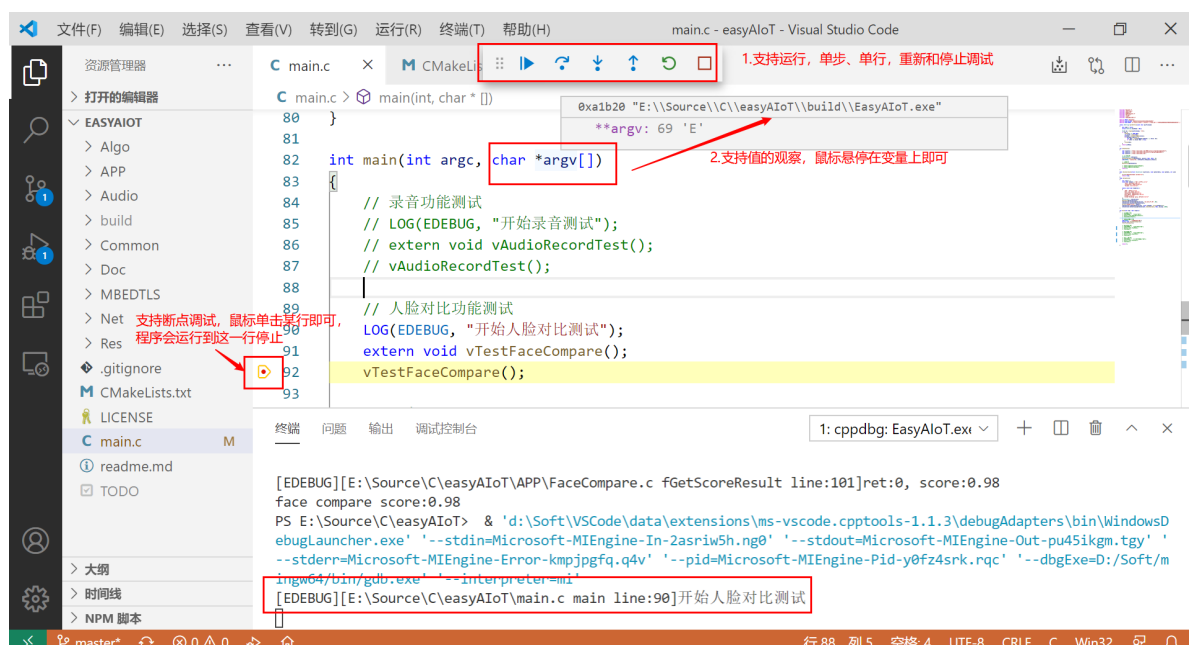
这一个步骤是和上一个步骤同步完成的，但是要注意的是，如果发现undefined reference类似的错误是在这个阶段导致的，需要开发者增加缺失的符号，可能来自自己的代码，也可能是缺少第三方或者系统库的依赖。缺少第三方或系统库可以通过在CMakeLists.txt文件中增加对应的target_link_libraries依赖库文件即可（注意依赖库的名称，比如依赖库的名字为libpthread.so, 只需要填写pthread即可，不需要前面的lib，也不需要后面的.so）

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。

TODO：图例，一些离散的问题

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流程图

TODO：程序流程图

5.3 编码

针对目标的流程有了，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。

TODO：编码过程

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。

TODO：测试过程和结果演示