IrisEngine SDK

开发指南

北京森博克智能科技有限公司

二〇二二年八月

目录

1 介绍 4

1.1 概述 4
1.2 修订历史 4

2 使用前提 5

2.1 WINDOWS 平台 5
2.1.1 安装 VS2017 再发布包 5

3 SDK 接口函数 6

3.1 IRISENG_INITIATE 6
3.1.1 函数定义 6
3.1.2 示例代码 6
3.1.3 通用设置参数 7
3.1.4 错误码定义 8
3.2 IRISENG_RELEASE 9
3.2.1 函数定义 9
3.2.2 示例代码 9
3.3 IRISENG_LOAD_DEV_PARAMS 10
3.3.1 函数定义 10
3.3.1 示例代码 10
3.4 IRISENG_ENROLL 11
3.4.1 函数定义 11
3.4.2 示例代码 11
3.5 IRISENG_FETCH_ENROLL_DATA 12
3.5.1 函数定义 12
3.5.2 示例代码 13
3.6 IRISENG_IDENTIFY 13
3.6.1 函数定义 13
3.6.2 示例代码 13
3.7 IRISENG_FETCH_SNAP_DATA 14
3.7.1 函数定义 14
3.7.2 示例代码 14
3.8 CALLBACK 14
3.8.1 函数定义 14
3.8.2 说明 15
3.9 IRISENG_STOP 15
3.9.1 函数定义 15
3.9.2 示例代码 15
3.10 IRISENG_ADD_USER 15
3.10.1 函数定义 15
3.10.2 示例代码 16

3.11 IRISENG_FETCH_USER 16
3.11.1 函数定义 *16*
3.11.1 示例代码 *16*
3.12 IRISENG_DELETE_USER 16
3.12.1 函数定义 *16*
3.12.2 示例代码 *17*
3.13 IRISENG_DELETE_ALL_USER 17
3.13.1 函数定义 *17*
3.13.2 示例代码 *17*
3.14 IRISENG_GET_USER_DIR 17
3.14.1 函数定义 *17*
3.14.2 示例代码 *18*
3.15 IRISENG_GET_USER_LIST 18
3.15.1 函数定义 *18*
3.15.2 示例代码 *18*
3.16 IRISENG_CHANGE_CONFIGURE 18
3.16.1 函数定义 *19*
3.16.2 示例代码 *19*
3.17 IRISENG_GET_IMAGE_SIZE 19
3.17.1 函数定义 *19*
3.17.2 示例代码 *19*
3.18 IRISENG_SET_PREVIEW2 19
3.18.1 函数定义 *19*
3.18.2 示例代码 *21*
3.19 CALLBACK 21
3.19.1 函数定义 *21*
3.19.2 说明 *22*
3.20 IRISENG_GET_DEVICE_INFO 22
3.20.1 函数定义 *22*
3.20.2 示例代码 *22*
3.21 IRISENG_GET_DEVICE_INFO_2 22
3.21.1 函数定义 *22*
3.21.2 示例代码 *23*
3.22 IRISENG_GET_ALGOR_VERSION 23
3.22.1 函数定义 *23*
3.22.2 示例代码 *23*

1 介绍

1.1 概述

本文档描述了运行于 Windows/Linux 平台的应用程序如何使用 IrisEngine SDK

1.2 修订历史

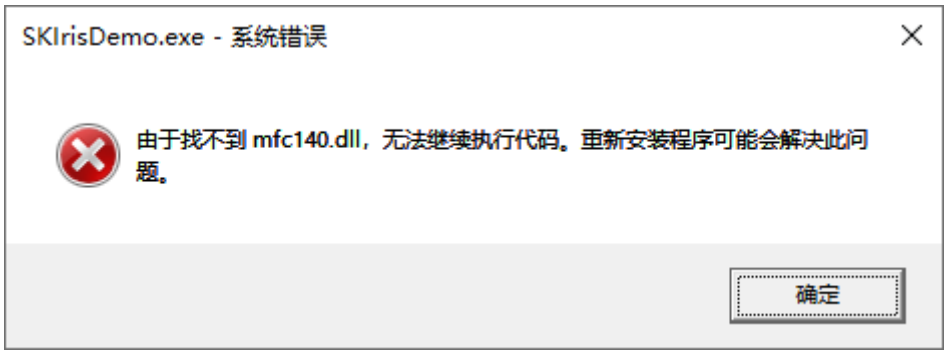
版本	日期	内容
1.0	2019-10-30	初始版本
1.2	2020-03-19	调整和完善接口
1.3	2020-04-20	加入【使用前提】
1.4	2020-12-07	补充安装 VS2017 再发布包的说明
1.5	2021-04-09	加入设备特定参数的说明
1.6	2022-03-20	* 删除关于注册组件的要求和说明 * 加入获取 json 格式的设备信息接口 * 加入获取算法版本和发布日期的接口
1.7	2022-04-20	* 更正关于 DIR_DATA/DIR_EYEDATA 的说明 * 加入有关 load_user_dir 的描述 * 加入有关错误码的描述 * 更新关于 set_preview 的说明
1.8	2022-06-10	加入 Linux 平台的描述
1.9	2022-08-24	加入下列函数的说明： IrisEng_load_dev_params IrisEng_fetch_enroll_data IrisEng_fetch_snap_data
1.11	2023-09-11	加入 Callback 的说明
1.12	2024-02-26	将 IrisEng_set_preview 改为 IrisEng_set_preview2
1.13	2024-12-16	某些接口的详细说明

2 使用前提

2.1 Windows 平台

2.1.1 安装 VS2017 再发布包

运行示例程序 SKIrisDemo，如果出现类似下面的错误：



则需要安装 redistributable packages for Visual Studio 2017，在下载页面中找到

x86: vc_redist.x86.exe

x64: vc_redist.x64.exe

下载并安装

3 SDK 接口函数

3.1 IrisEng_initiate

启动虹膜运行环境并初始化 SDK，在此过程中，配置参数中指定的数据目录下的用户虹膜数据会被加载到 SDK 中。

3.1.1 函数定义

```
int IrisEng_initiate(const char *configuration,
                     IrisEventCB event_callback);
```

参数表：

参数	解释
configuration	Json 格式的参数设置字符串，具体参数说明参见 3.1.3
event_callback	设备事件回调函数，主要提供热插拔事件上报

返回值：

0 ---成功，非 0 ---失败，具体请参照 irisengine_export.h 中关于返回值的定义

后面的接口函数返回类型为 int 的均可参照该返回值定义

3.1.2 示例代码

```
char conf_buf[] =
"{\
        \"enroll_mode\":\"two_eye\",\
        \"identify_mode\":\"any_eye\",\
        \"find_same\":\"off\" \
}";
int ret = IrisEng_initiate(conf_buf, eventCallback);
if(ret != IRIS_ENG_OK) {
        getErrorInfo(ret, msg);
        MessageBox(msg, "初始化失败", MB_ICONERROR);
}
else {
        MessageBox(msg, "初始化成功");
}
```

3.1.3 通用设置参数

参数	取值范围	解释
DIR_APP	可用目录	本 SDK 所使用的根目录，例如： "DIR_APP":"E:\\temp" 缺省时，该目录为%ProgramData%\Simbok\SKIris\
DIR_DATA	可用目录	数据存放的目录，位于 DIR_APP 之下，例如： "DIR_DATA":"E:\\temp\\data" 缺省时，该目录为 DIR_APP\data
DIR_EYEDATA	可用目录	虹膜数据存放的目录，位于 DIR_DATA 之下，例如： "DIR_EYEDATA":"E:\\temp\\data\\eyedata" 缺省时，该目录为 DIR_DATA\data 当"load_user_dir": "on"时，SDK 自动从该目录加载全部用户数据子目录。 用户注册完成后，用户数据子目录也会在该目录下生成
enroll_mode	two_eye left_eye right_eye	分别对应双眼、左眼、右眼注册
identify_mode	two_eye left_eye right_eye any_eye	分别对应双眼、左眼、右眼、任一眼识别
find_same	on/off	指定注册时是否查找当前用户数据库中具有相同虹膜特征的用户 on---查找重复，off---不查找
load_user_dir	on/off	on --- SDK 启动时自动从 DIR_EYEDATA 目录下加载全部

		用户数据子目录到运行时内存，此为缺省值 off --- SDK 启动时不主动加载用户数据目录。一般用于当上层应用使用数据库来管理用户数据时，此时上层应用可通过调用 IrisEng_add_user（参见 3.10）接口将用户数据加载到 SDK 中。
save_user_dir	on/off	指定当用户注册完成后，图像和特征数据是否要保存在 DIR_EYEDATA 下的用户子目录中 缺省为 on 该项为 off 时，注册成功后不保存用户子目录。如需获取注册后的用户数据，应使用 3.5 接口取出该用户数据
save_snap_file	on/off	指定当用户识别成功后，识别到图像是否要保存在 DIR_EYEDATA/snapshot 子目录下 缺省为 off，识别成功后不保存本次识别出的图像。如需获取识别出的图像，应使用 3.7 接口取出该图像

3.1.4 错误码定义

调用接口返回值非 0 时，表示发生错误，具体错误码和含义如下：

宏定义	数值	解释
IRIS_ENG_OK	0	成功
IRIS_ENG_ERR_NO_DEV	1	未发现 Simbok 专用采集模组接入系统
IRIS_ENG_ERR_DEV_CONNECT	2	连接采集模组时失败
IRIS_ENG_ERR_DEV_ILLEGAL	3	接入的采集模组未经过正常的出厂设定
IRIS_ENG_ERR_DEV_CAPTURE	4	启动图像采集失败
IRIS_ENG_ERR_DEV_HOTPLUG_ENABLE	6	启动模组热插拔监测失败
IRIS_ENG_ERR_DEV_FRAME_PARAM	11	设置图像帧参数失败
IRISDEV_ERR_EU_CTRL	12	控制采集模组的红外/LED/测距时失败
IRISDEV_ERR_EXPIRE_DEV	15	已过期的采集模组

IRIS_ENG_ERR_NOT_INIT	20	SDK 尚未初始化
IRIS_ENG_ERR_ALREADY_INIT	21	SDK 已初始化（不能重复初始化）
IRIS_ENG_ERR_WRONG_CONFIG	22	错误的配置参数（必须是合法的 json 格式）
IRIS_ENG_ERR_MAX_USER_NUM	23	超过最大用户数上限
IRIS_ENG_ERR_USER_ID_ILLEGAL	24	用户 ID 不合法
IRIS_ENG_ERR_USER_EXIST	25	用户 ID 已存在
IRIS_ENG_ERR_NO_USER	26	没有用户存在
IRIS_ENG_ERR_ALREADY_RUN	27	已处于某种运行状态（注册/识别）
IRIS_ENG_ERR_USER_NOT_EXIST	28	指定的用户 ID 不存在
IRIS_ENG_ERR_SOUND_PLAY	30	声音播放故障
IRIS_ENG_ERR_MISSING_CONFIG	31	config 项缺失
IRIS_ENG_ERR_UNALLOWED_REGION	32	不允许在该区域使用
IRIS_ENG_ERR_NO_CIQ_LIBRARY	33	未找到第三方的图像检测库
IRIS_ENG_ERR_SYSTEM	50	系统级错误（内存分配、线程等）

以上所列错误码适用于所有 SDK 接口的返回值。

详细宏定义也可以参照 SDK 包中 inc\irisengine_export.h

3.2 IrisEng_release

释放虹膜运行环境

3.2.1 函数定义

```
int IrisEng_release();
```

3.2.2 示例代码

略

3.3 IrisEng_load_dev_params

不同的设备模组因自身结构和应用场合的不同，需要使用一些特殊的内部参数值，例如：

```
{
    "E6": {
        "THD_FOCUS_RANGE":"40-0-60",
        "THD_RADIUS_L":105,
        "THD_RADIUS_H":130,

        "THD_FOCUS2":40,
        "THD_RADIUS_L2":95,
        "THD_RADIUS_H2":135
    },

    "U5000": {
        "THD_RADIUS_L":95,
        "THD_RADIUS_H":115,
        "THD_RADIUS_L2":90,
        "THD_RADIUS_H2":120
    }
}
```

对该函数的调用要紧跟在 IrisEng_initiate 后进行。

3.3.1 函数定义

```
int IrisEng_load_dev_params(const char *params);
```

参数表：

参数	解释
params	Json 格式的参数设置字符串，具体参数说明参见 3.1.3 第一级 key 总是设备类型名称 内容完全对应 SDK 包中的 param_dev.cfg 文件

3.3.1 示例代码

略

3.4 IrisEng_enroll

注册虹膜用户

3.4.1 函数定义

```
int IrisEng_enroll(const char *user_id, int overwrite,
                   CallBack enroll_callback);
```

参数表：

参数	解释
user_id	注册用户 ID
overwrite	1---覆盖原有用户（无论该用户是否存在），0---不覆盖（如果该用户已存在则返回错误）
enroll_callback	结果回调函数，参见 3.8

3.4.2 示例代码

```
    void callBack(const char* user_id, iris_callback_result *result)
    {
        char msg[1024];

        sprintf(msg, "result:%d, eye:%d, finish:%d", result->result,
result->eye, result->finish);
        std::cout << msg << std::endl;

        if(result->finish) {
            switch(state) {
            case ENROLL:
                if(result->result == IRIS_RESULT_OK) {
                    sprintf(msg, "%s enrolled", user_id);
                }
                else if(result->result == IRIS_RESULT_FAIL) {
                    sprintf(msg, "find user '%s' with same iris trait",
user_id);
                }
                else if(result->result == IRIS_RESULT_TIMEOUT) {
                    sprintf(msg, "enroll timeout");
                }
                state = NOTHING;
                break;
```

```
            case IDENTIFY:
            case IDENTIFY2:
                if(result->result == IRIS_RESULT_OK) {
                    sprintf(msg, "%s identified", user_id);
                }
                else if(result->result == IRIS_RESULT_FAIL) {
                    sprintf(msg, "no user identified");
                }
                else if(result->result == IRIS_RESULT_FAIL_LR_MATCH) {
                    sprintf(msg, "find different user with same
left/right iris");
                }
                else if(result->result == IRIS_RESULT_TIMEOUT) {
                    sprintf(msg, "identify timeout");
                }
                if(state == IDENTIFY) {
                    state = NOTHING;
                }
                break;

            default:
                break;
            }
            std::cout << msg << std::endl;
        }
    }


    int ret = IrisEng_enroll( strName, 1, myCallback );
    if (IRIS_ENG_OK != ret) {
        return;
    }
```

3.5 IrisEng_fetch_enroll_data

将刚刚完成注册的用户数据从 SDK 缓存中取出

3.5.1 函数定义

```
    int IrisEng_fetch_enroll_data( const char *user_id,
                                   char *data,
                                   int len);
```

参数表：

参数	解释

user_id	NULL---识别功能 非 NULL---对该 user_id 的验证功能
data	提前分配的地址空间，用来存储获取的用户数据
len	data 所指向的地址空间大小

3.5.2 示例代码

```
const char* user = "aaa";
int data_len = IrisEng_get_enroll_data_len(user);
char* data = (char*)malloc(data_len+1);
int ret = IrisEng_fetch_enroll_data(user, data, data_len+1);
```

3.6 IrisEng_identify

启动虹膜识别

3.6.1 函数定义

```
int IrisEng_identify(const char *user_id, int continuous,
                     CallBack identify_callback);
```

参数表：

参数	解释
user_id	NULL --- 用于识别 非 NULL --- 用于对该 user_id 的验证
continuous	1---连续识别，0---单次识别
identify_callback	结果回调函数，参见 3.8

3.6.2 示例代码

```
int ret = IrisEng_identify( NULL, 1, myCallback );
if (IRIS_ENG_OK != ret) {
        getErrorInfo(ret, msg);
        MessageBox(msg, "识别失败", MB_ICONERROR);
        return;
}
```

3.7 IrisEng_fetch_snap_data

将刚刚识别成功的用户数据从 SDK 缓存中取出

3.7.1 函数定义

```
int IrisEng_fetch_snap_data(char *data, int len);
```

3.7.2 示例代码

```
int data_len = IrisEng_get_snap_data_len();
char* data = (char*)malloc(data_len+1);
int ret = IrisEng_fetch_snap_data(data, data_len+1);
```

3.8 CallBack

注册/识别的结果回调函数

3.8.1 函数定义

```
typedef void (*CallBack)(const char* user_id,
                         iris_callback_result *cb_result);
```

参数表：

参数	解释
user_id	注册/识别到的用户 ID
cb_result	result：表示结果，0 为成功。具体参见 irisengine_export.h 中的 IRIS_RESULT 定义 eye：0-左眼，1-右眼 finish：表示本次回调是否是注册/识别的最后一次回调 例如： 1. 双眼注册时，会发生 2 次回调，结果可能是： result:0, eye:0,finish:0 result:0, eye:1,finish:1 表示左右眼均注册成功

	2. 任一眼识别时，只发生一次回调，结果可能是： result:0, eye:1,finish:1 表示右眼识别成功

3.8.2 说明

在此回调函数中不要进行长耗时任务的处理，也不要调用注册/识别的停止或启动等接口。

3.9 IrisEng_stop

停止正在运行的虹膜注册/识别

3.9.1 函数定义

```
int IrisEng_stop();
```

3.9.2 示例代码

略

3.10 IrisEng_add_user

将用户数据动态加入到 SDK 运行内存中。

一般情况下，当 load_user_dir 为 off 时才使用该接口。

3.10.1 函数定义

```
int IrisEng_add_user(const char *user_id,
                     const unsigned char *left_iris,
                     const unsigned char *right_iris);
```

参数表：

参数	解释
user_id	添加的用户 ID

left_iris	左眼虹膜特征码，可以为 NULL
right_iris	右眼虹膜特征码，可以为 NULL

3.10.2 示例代码

略

3.11 IrisEng_fetch_user

将用户数据从 SDK 运行内存中取出

3.11.1 函数定义

```
int IrisEng_fetch_user(const char *user_id,
                       bool *left_exist, unsigned char *left_iris,
                       bool *right_exist, unsigned char *right_iris);
```

参数表：

参数	解释
user_id	待取出的用户 ID
left_exist	返回值，左眼数据是否存在
right_exist	返回值，右眼数据是否存在
left_iris	左眼虹膜特征码
right_iris	右眼虹膜特征码

3.11.1 示例代码

略

3.12 IrisEng_delete_user

将用户数据从 SDK 中删除

3.12.1 函数定义

```
int IrisEng_delete_user(const char *user_id, int dir_removed);
```

参数表：

参数	解释
user_id	删除的用户 ID
dir_removed	1---删除该用户的数据目录，0---不删除目录 数据目录：3.1.3 中 DIR_EYEDATA 目录下该用户的子目录

3.12.2 示例代码

略

3.13 IrisEng_delete_all_user

将全部用户数据从 SDK 中删除

3.13.1 函数定义

```
int IrisEng_delete_all_user(int dir_removed);
```

参数表：

参见 3.12.1 中的说明

3.13.2 示例代码

略

3.14 IrisEng_get_user_dir

获得用户虹膜数据所在的目录。

该目录可以由上层应用在初始化时通过 DIR_EYEDATA 配置项指定给 SDK，如果不指定，SDK 会使用缺省目录，通过该接口来获知是哪个目录

3.14.1 函数定义

```
int IrisEng_get_user_dir(char* user_dir);
```

参数表：

参数	解释
user_dir	返回用户数据（即 EYEDATA）的目录 作为参数传入时，该参数指向提前分配的内存地址

3.14.2 示例代码

略

3.15 IrisEng_get_user_list

获得用户数量和整个用户列表字符串。

3.15.1 函数定义

```
int IrisEng_get_user_list(int *user_num, char* list);
```

参数表：

参数	解释
user_num	返回用户目录下包含的用户数量
list	返回以逗号分隔的所有用户列表的字符串，例如：”a1,a2,a3”

3.15.2 示例代码

```
char* l_user_list = NULL;
int l_user_num;

IrisEng_get_user_list(&l_user_num, NULL);
if(l_user_num == 0)
    return;
l_user_list = (char*)calloc(l_user_num, IRIS_USER_ID_LENGTH);
irisEng_get_user_list(&l_user_num, l_user_list);
```

3.16 IrisEng_change_configure

可在 SDK 运行时动态调整某些个别的参数配置。

3.16.1 函数定义

```
int IrisEng_change_configure(const char *configuration);
```

参数表：

参见 3.1.3 中的具体说明

3.16.2 示例代码

略

3.17 IrisEng_get_image_size

获得虹膜图像的像素宽度和高度。

3.17.1 函数定义

```
int IrisEng_get_image_size(int *height, int *width);
```

参数表：

参数	解释
height	虹膜图像高度（像素单位）
width	虹膜图像宽度（像素单位）

3.17.2 示例代码

略

3.18 IrisEng_set_preview2

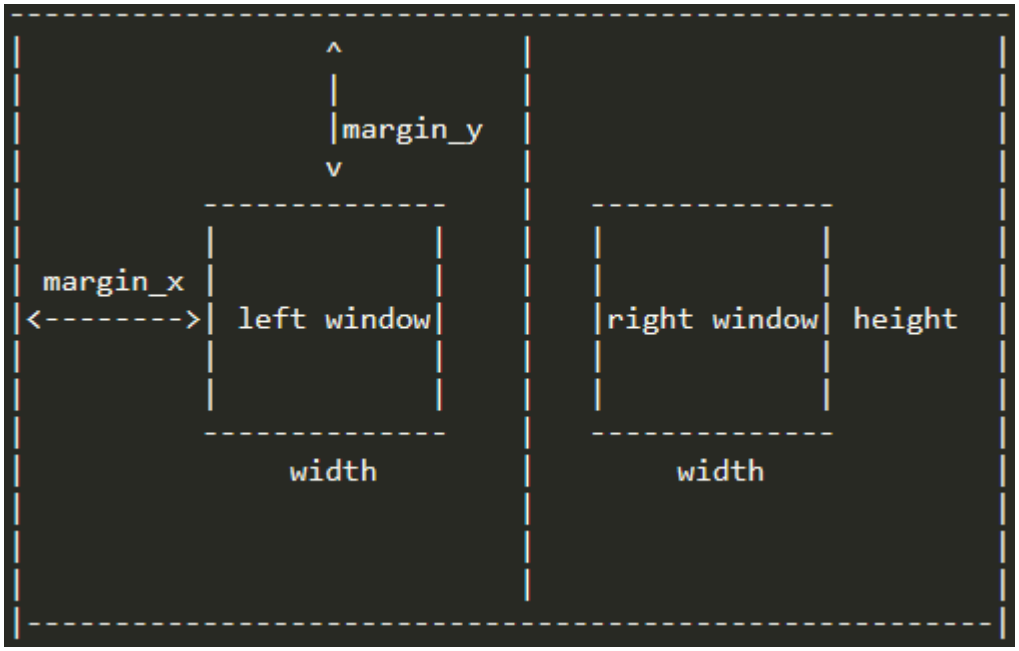
设置图像实时预览相关的参数

3.18.1 函数定义

```
int IrisEng_set_preview2(const char* mode,
                         IMAGE_PIXEL_FORMAT format,
                         int width, int height,
                         int margin_x, int margin_y,
                         PreviewCB callback);
```

参数表：

参数	解释
mode	选择输出图像的区域 "whole" --- 完整图像区域 输出完整图像区域，并按指定的 width/height 来进行缩放 "custom"--- 定制区域 输出左右眼各自中央截取区域的拼接图像 * 对于 AF800 等模组，只会输出完整图像区域，并且不进行任何缩放
format	IMAGE_PIXEL_RGB565 = 0, IMAGE_PIXEL_RGB888 = 1, IMAGE_PIXEL_ARGB8888 = 2, IMAGE_PIXEL_RAW = 3, //直接使用模组采集到的原始图像数据，比如 JPEG。一般情况下不用此格式
width	用于"custom"时，为左/右眼截取区域的宽度。用于"whole"时，为经过缩放的输出图像的实际宽度
height	用于"custom"时，为左/右眼截取区域的高度。用于"whole"时，为经过缩放的输出图像的实际高度
margin_x	用于"custom"，为左眼截取区域的左边缘和整体图像最左边的横向距离（或右眼截取区域的右边缘和整体图像最右边的横向距离）。-1 表示居中
margin_y	用于"custom"，为截取区域的上边缘和整体图像最上边的纵向距离。-1 表示居中
callback	预览回调函数，参见 3.19



3.18.2 示例代码

略

3.19 CallBack

预览回调函数

3.19.1 函数定义

```
typedef void (* PreviewCB)(const unsigned char* data, int len,
                           int width, int height, int flag);
```

参数表：

参数	解释
data	图像数据指针
len	图像数据长度
width	图像宽度
height	图像高度
flag	图像标记 -1 --- 全局图像 0 --- 左眼

	1 --- 右眼

3.19.2 说明

在此回调函数中不要进行长耗时任务的处理，也不要调用注册/识别的停止或启动等接口。

3.20 IrisEng_get_device_info

获取虹膜设备信息

3.20.1 函数定义

```
int IrisEng_get_device_info(IrisDeviceInfo *dev_info);
```

其中 IrisDeviceInfo 的定义如下：

```
typedef struct {
    char fw_ver[32];                //固件版本
    char fw_date[32];               //固件日期
    char manufacturer[32];          //设备厂家
    char logo[32];                  //厂家 LOGO
    char sn[32];                    //设备序列号，格式为" SK-E6-00011005"
//其中的 E6 为设备类型,
    unsigned int max_user_num;      //设备允许最大用户数
} IrisDeviceInfo;
```

3.20.2 示例代码

略

3.21 IrisEng_get_device_info_2

获取虹膜设备信息

3.21.1 函数定义

```
int IrisEng_get_device_info_2(char* dev_info, int len)
```

其中 dev_info 包含内容和前述 IrisDeviceInfo 基本相同，但格式为 json 格式，方便后续信息的扩展或调整

3.21.2 示例代码

略

3.22 IrisEng_get_algor_version

获取当前算法版本和发布日期

3.22.1 函数定义

```
int IrisEng_get_algor_version(char *version, char* date)
```

参数表：

参数	解释
version	算法版本号 三段式，如: “2.3.1”
date	算法发布日期 如: ”2022-03-01”

3.22.2 示例代码

略