青莲云嵌入式 SDK ESP8266 版开发使用文档

版本记录：

版本	编写/修订说明	修订人	修订日期	备注
1.0.0	创建文档	王科岩	20160929	

目录

1、概要

适用环境

平台：ESP8266EX

CPU 架构：Xtensa

编译工具链：xtensa-lx106-elf

SDK 目录结构

我们提供一个基于 ESP8266 的静态库，并提供一个与之配套的使用案例。

```
-\sdk_for_esp8266
    -\ QL_ESP8266EX_LIB_0100
        -\ libqinglianyun.a
        -\ qlcloud_interface.h
    -\ QL_ESP8266EX_LIB_0100_DEMO
        -\include
            -\ qlcloud_interface.h
            -\ user_config.h
        -\user
            -\Makefile
            -\user_main.c
        -\ !!!readme!!!.txt
        -\gen_misc.sh
        -\Makefile
```

qlcloud_interface.h 声明了对外的所有功能接口；

user_config.h ESP8266 平台配置头文件；

user_main.c 案例源码。

开发准备

在进行设备开发前，需要先做以下配置准备：

1、登陆 https://www.qinglianyun.com 的管理后台创建产品。创建完产品后，会得到（在产品信息页）产品 ID 和产品密钥。例如下图 1：

图 1 产品信息

2、在功能管理页添加、设置产品的功能项。设备和云端通信时，必须按照定义的数据点进行上传、下发数据。例如下图 2：

图 2 功能点信息

3、了解 ESP8266 模组的开发环境搭建、编译生成固件的步骤，请参考乐鑫官网 http://espressif.com/zh-hans/support/download/documents 提供的文档。

2、功能列表

嵌入式 SDK 提供以下功能：

1) 安全快速联网

2) 获取连接状态

3) 双向消息推送

4) 实时数据上报

5) 实时获取命令

6) 上传配置

7) 更新配置

8) 同步云端时间

9) OTA 升级

3、安全快速联网

该函数初始化设备与青莲云交互的上下文环境 ,该接口调用必须早于文档提及的其他所有 SDK 接口。

调用该返回成功后，会自动与青莲云建立长连接。并且当连接断开时，会自动重连。

函数原型

```
intqlcloud_initialization(
        unsigned int product_id,
        unsigned char*product_key,
        unsigned char*mcu_version,
        unsigned char is_debug_open)
```

参数

product_id	在官网上创建产品时得到的产品 ID

product_key	在官网上创建产品时得到的 32byte 的产品密钥串
mcu_version	MCU 产品的版本号，用于 OTA 升级使用
is_debug_open	是否输出 debug 信息 0-不输出 1-输出 debug 信息由 8266 的串口 0 输出，波特率 74880，数据位 8，校验位 None，停止位 1，流控制 None

返回值

0 成功，-1 失败

示例

```
/**
1000527506-设备所属的产品ID，创建产品时获取得到。如图1产品信息
“3528692539696153db74266afcce6443”-设备所属的产品密钥，创建产品时获取得到。如图1产品信息
“01.01”-MCU版本号
0-表示不输出debug信息
**/
ret = qlcloud_initialization(1000527506, “3528692539696153db74266afcce6443”, "01.01",0);
```

4、获取连接状态

判断与云端连接的状态采用注册回调函数的方式实现，设备连接到云端或从云端断开，该函数被调用。首先,实现处理获取模组联网状态的回调函数：

回调函数原型

```
typedef void (*qlcloud_status_cb)(int is_connected);
```

参数

is_connected	是否已成功联网

0-未联网

1-联网成功

返回值

空

示例

```
void on_conn_handle(int is_connected)
{
    /* 获取云端的联网状态 */
    printf("connect state:%d\r\n", is_connected);
}
```

注册接收联网状态的函数

函数原型

```
void qlcloud_register_status_callback(qlcloud_status_cb status_cb);
```

参数

qlcloud_status_cb	回调函数，用于处理接收联网状态

返回值

空

示例

```
qlcloud_register_status_callback(on_conn_handle);
```

5、双向消息推送

支持双向的消息推送功能：设备可以向云端 push 消息，也可以接收从云端 push 下来的消息。

设备推送消息给云端后，云端自动将该消息推送给与之绑定的控制端（比如手机 APP）。

从控制端（比如手机 APP）推送过来的消息，云端会自动将该消息转送给设备。

支持推送消息的离线功能，当推送消息的接收端不在线时，云端是保存该消息，当接收端上线以后，云端会将消息推送给它。

推送消息到绑定手机

推送消息给云端，云端自动将该消息推送到与之绑定的控制端，比如手机 APP 上。

函数原型

void qlcloud_push_message(char* data, int len)

参数

data	推送的消息
len	消息长度

返回值

空

示例

```
/*
*向与该设备绑定的控制端推送一条电量过低的事件
*/
qlcloud_push_message("{\"alert\": \"low battery\"}", strlen("{\"alert\": \"low battery\"}"));
```

接收云端推送的消息

接收云端推送采用注册回调函数的方式实现，一旦有云端推送过来的消息，该函数就会自动被调用。

首先,实现处理获取推送消息的回调函数：

回调函数原型

void (*receive_pushmessage_cb)(char *pushmessage, int pushmessage_len);

参数

pushmessage	接收的推送消息

pushmessage_len　　接收的推送消息长度

返回值

空

示例

```
void get_pushmessage_handle(char *pushmessage, intpushmessage_len)
{
    /*  获取云端推送的 push 消息  */
    printf("Got pushmessage from cloud: length = %d; message = %s \r\n"
                                        , pushmessage_len, pushmessage);
}
```

注册接收推送消息

函数原型

```
void qlcloud_register_getpush_callback(receive_pushmessage_cbgetpush_cb);
```

参数

receive_pushmessage_cb　　回调函数，用于处理接收消息

返回值

空

示例

```
qlcloud_register_getpush_callback(get_pushmessage_handle);
```

6、实时数据上报

实时上传数据至云端。

上传的数据，采用 key=value 的方式组织。

其中，key 必须是在青莲云管理平台上注册的产品功能点对应的变量名。如图 2 功能点信息所示，key 可以为 switch、temperature。

函数原型

voidqlcloud_upload_data(int datatype, char* data, intdata_len)

参数名

datatype	数据类型，目前支持 SAVEDATA_TYPE_QUERY - default SAVEDATA_TYPE_JSON SAVEDATA_TYPE_BINARY
data	上传的数据
data_len	上传的数据长度

返回值

空

示例

```
/**
SAVEDATA_TYPE_QUERY-数据类型
"switch=0"-设备所属的产品数据点，创建产品数据点时获取得到。如图2功能点信息。
**/

qlcloud_upload_data(SAVEDATA_TYPE_QUERY, "switch=0”, strlen("switch=0”));
```

7、实时获取命令

设备实时获取从控制端发来的命令，采用注册回调函数方式来实现。

首先,实现处理命令的回调函数：

回调函数原型

函数原型

```
int (*downloadcmd_cb)(download_cmdinfo* cmdinfo
                      , char *cmdresponse
                      ,int cmdresponse_size)
```

参数

download_cmdinfo	包含云端发送下来的命令，格式为 key=value

cmdresponse	用于存放对命令处理结果数据的 buffer
cmdresponse_size	存放命令处理结果数据的 buffer 的大小

返回值

0 成功， -1 失败

示例

```
inton_downloadcmd_handle(download_cmdinfo *cmdinfo, char *out, int outsize)
{
    /* 获取云端下发的命令 */
printf("Got command from cloud: %s = %s \r\n", cmdinfo->key, cmdinfo->value);
    /* 本地处理该命令  */

    /* 返回处理结果 */
snprintf(out, (size_t)outsize, (const char *)"i know this command.");

return ACK_OK;
}
```

注册命令处理

函数原型

```
void qlcloud_register_cmd_callback(
                        int (*downloadcmd_cb)(download_cmdinfo*, char *, int))
```

参数

downloadcmd_cb	回调函数，当设备接收到命令时触发调用

返回值

空

示例

```
qlcloud_register_cmd_callback(on_downloadcmd_handle);
```

8、上传配置

该功能项为：设备被动上传配置，当控制端（比如手机 APP）发来获取配置请求时触发该逻辑操作。采用注册回调函数方式来实现。

首先,实现应答获取配置的回调函数：

函数原型

```
int (*answer_getconfig_request)(char *request_conf_key
                                , char *answerconf
                                , int answerconf_size)
```

参数

request_conf_key	请求获取配置项名称
answerconf	存放配置信息 buffer
answerconf_size	存放配置信息 buffer 大小

回值

0 成功 -1 失败

示例

```
intanswer_getconfig_request_handle(char *request_conf_key, char *answerconf,
intanswerconf_size)
{
    /*获取云端下发的命令 */
    printf("Got a get config request from cloud. the config key is %s\r\n", request_conf_key);
    /*从本地获取key为request_conf_key的配置信息*/
    /*返回处理结果 */
    snprintf(answerconf, answerconf_size, (const char *)"config u wanted is this.");
    return ACK_OK;
}
```

注册上传配置处理

函数原型

```
voidqlcloud_register_conf_callbacks (
```

int (*answer_getconfig_request)(char *, char *, int)
int (*update_local_config)(char *config_str))

参数

answer_getconfig_request　回调函数，用于应答获取配置请求

update_local_config　　回调函数，用于更新配置到本地

返回值

空

示例

qlcloud_register_conf_callbacks(answer_getconfig_request_handle,update_local_config_handle);

9、更新配置

该功能项为：设备更新配置，当控制端（比如手机 APP）发来更新配置请求时触发该逻辑操作。采用注册回调函数方式来实现。

回调函数，当设备接收到最新配置请求消息时触发调用

函数原型

int (*update_local_config)(char *config_str)

参数

config_str　　获取到的最新配置信息

回值

0 成功　-1 失败

示例

```
intupdate_local_config_handle(char *config_str)
{
    /*获取到新的配置信息*/
    printf("Got a newest config from cloud:%s \r\n", config_str);
```

```
    /*将新的配置更新到本地  */
    /*返回处理结果*/
    return ACK_OK;
}
```

注册更新配置处理

函数原型

```
voidqlcloud_register_conf_callbacks (
                int (*answer_getconfig_request)(char *, char *, int)
                int (*update_local_config)(char *config_str))
```

参数

answer_getconfig_request　回调函数，用于应答获取配置请求

update_local_config　　回调函数，用于更新配置到本地

返回值

空

示例

```
qlcloud_register_conf_callbacks(answer_getconfig_request_handle,update_local_config_handle);
```

10、同步时间

获取云端实时时间戳，用于同步时间

函数原型

unsigned int qlcloud_get_onlinetime()

说明

获取云端实时时间戳，用于同步时间

参数

空

返回值

空

示例

```
cur_st = qlcloud_get_onlinetime();
```

11、OTA 升级

OTA 升级采用注册回调函数的方式来实现。

OTA 升级包含了两个逻辑部分：OTA 升级包下发到设备；OTA 升级指令下发到设备。

因此，需要实现两个处理函数来分别处理 OTA 数据包、OTA 升级指令。

详细的 OTA 升级包逻辑请参看《青莲云 OTA 说明文档》。

OTA 升级包数据处理

处理云端推送的 OTA 升级包数据，数据包是分段传输，该回调会被触发调用多次

原型函数

```
int (*ota_binfile_chunk_cb)(binfile_chunk_data *)
```

参数

binfile_chunk_data	OTA 数据包，包含： 当前包的数据 当前数据长度 是否是最后一个分片包 当前分片包在整个文件中的偏移量

返回值

0 成功， -1 失败

示例

```
/**
 **回调函数，接收 OTA 升级包数据
 **/
intprocess_ota_binfile_chunk(binfile_chunk_data *chunkinfo)
{
if (chunkinfo == NULL || chunkinfo->data == NULL || chunkinfo->datalen<= 0) {
printf("MCU: get_ota_binfile_chunkchunkinfo == null. \r\n");
return ACK_ERR;
    }
printf("MCU: current chunk's offset = %d; datalen = %d.\r\n", chunkinfo->offset, chunkinfo->datalen);
if (chunkinfo->isend) {
printf("MCU: current chunk is last one.\r\n");
    }
 /* 处理收取每个一个 chunk 数据包， 包含四个信息:
     ** 1、是否是最后一个 chunk 数据块
     ** 2、当前 chunk 数据块在整个数据中的偏移
     ** 3、当前 chunk 数据块的长度
     ** 4、当前 chunk 数据块的数据
     **/

printf("MCU: get_ota_binfile_chunk success.\r\n");

return ACK_OK;
}
```

OTA 升级指令处理

当 OTA 升级包全部被接收以后，云端根据升级策略向设备发送升级指令，这时触发 OTA 升级处理逻辑。

函数原型

```
int (*ota_upgrade_cb)()
```

参数

空

返回值

0 成功， -1 失败

示例

```
/**
 **回调函数，处理 OTA 升级指令
 **/
intprocess_ota_upgrade_command()
{
 /* 处理云端发来的升级指令
    ** 根据 MCU 自身逻辑，处理先前接收到的 chunk 数据
    */

printf("Got ota upgrade command from cloud.\r\n");

return ACK_OK;
}
```

注册 OTA 升级接口

该函数注册 OTA 升级需要的上述两个处理接口，需要 OTA 功能的 Device，必须使用该接口进行注册

函数原型

```
voidqlcloud_register_ota_callbacks (
                                ota_binfile_chunk_cbchunk_cb,
                                ota_upgrade_cbcmd_cb)
```

参数

chunk_cb	类型为 ota_binfile_chunk_cb 的回调函数 处理云端推送的 OTA 升级包数据，数据包是分段传输 该回调会被触发调用多次，该接口需用户实现
cmd_cb	类型为 ota_upgrade_cb 的回调函数

处理云端发送的 OTA 升级指令，只有当 OTA 升级包全部下载到本地以后，云端才会推送升级指令，该接口需用户实现

返回值

空

示例

```
qlcloud_register_ota_callbacks(process_ota_binfile_chunk,process_ota_upgrade_command);
```