

mpu6050 工程指导文档

MPU6050 工程使用教程：

烧写本工程编译生成的文件，烧写成功之后，开机，并写入 sn 号。

注：写入 sn 号的方法，本工程是用的 alimqtt 协议，需要在智云服云平台创建 alimqtt 协议的产品，并在产品的调试设备页面中，添加设备 sn 号，可以自行设置 sn 号，只要没有和智云服的 sn 号重复就可以使用。添加完成后会生成 productKey、设备名称、设备秘钥。根据生成的信息，通过串口发送命令：

```
$$write-9999-a15W6M*****-kcz6kqRYcJnSV9AI65N5Laqd*****/
```

即：\$\$write-设备名称-productKey-设备秘钥/

注意：波特率 115200、不发送回车换行、命令最后要加/

如果不准备自己创建产品，直接用这个工程。则需要找群里管理员帮忙生成设备 sn 号，然后写入开发板，并扫描开发包根目录下的小程序二维码，用这个小程序查看数据。



如果准备自己创建产品，可以参考下面的方法，学习创建产品，创建自己的数据点、小程序，就可以用自己的小程序查看数据调试了。

问：如何在智云服创建 alimqtt 协议的产品和数据点等功能？

答：

创建产品：1、百度搜索智云服，在网页上注册账号并登陆。2、进入云平台。3、在产品列表的下发选择添加产品。4、选择通信协议为 ALIMQTT。

创建数据点：在产品的定义功能界面已经有了几个默认的数据点如果需要 GPS 功能，可以通过功能模板勾选 GPS/Beidou、AGPS 两个。如果需要添加其他的数据点，也可以通过自定义功能添加。本工程数据点如下：

显示名称	标识名(fieldid)	读写类型	数据类型	数据长度	单位	小数位	备注	操作
LBS(纬度)	latitudeLbs	只读	数值型	12	°	0	无	 
LBS(经度)	longitudeLbs	只读	数值型	12	°	0	无	 
GPS(纬度)	latitudeGps	只读	数值型	12	°	0	无	 
GPS(经度)	longitudeGps	只读	数值型	12	°	0	无	 
采集时间	collectTime	只读	字符串	20	无	0	无	 
电量百分比	battyPercent	只读	数值型	4	%	0	无	 
信号强度	signallev	只读	数值型	3	dB	0	无	 
运行时长	runTime	只读	数值型	10	s	0	无	 
LED灯	ledSwitch	可写	布尔值	1	无	0	无	 
传输间隔	transTime	可写	数值型	10	s	0	无	 

数据点添加完成之后，必须在程序里对应数据点，一个一个添加数据的上传和解析，参考文档后面的上报数据和接收数据代码解析。标识名要与程序中的一致，否则无法接收到数据。读写类型如果是只读即程序里需要上传的数据，如果是可写即程序里需要根据此数据点下发的数据进行解析。

添加 SN 号：在第三步调试设备中的添加设备里，写入自己定义的 SN 号，如果提示 SN 号重复，请更换 SN 号写入，因为所有客户所有产品的 SN 号都是在智云服的数据库里面，有可能这个 SN 号被占用了。

请输入设备名称

添加设备

productKey	设备名称	设备秘钥	操作
a15W6[REDACTED]	9999	kc[REDACTED]N5LaqdSMzHWUTC	删除

创建小程序：在第四步 App 开发中，上传小程序背景图片，并且设置小程序名称即可。

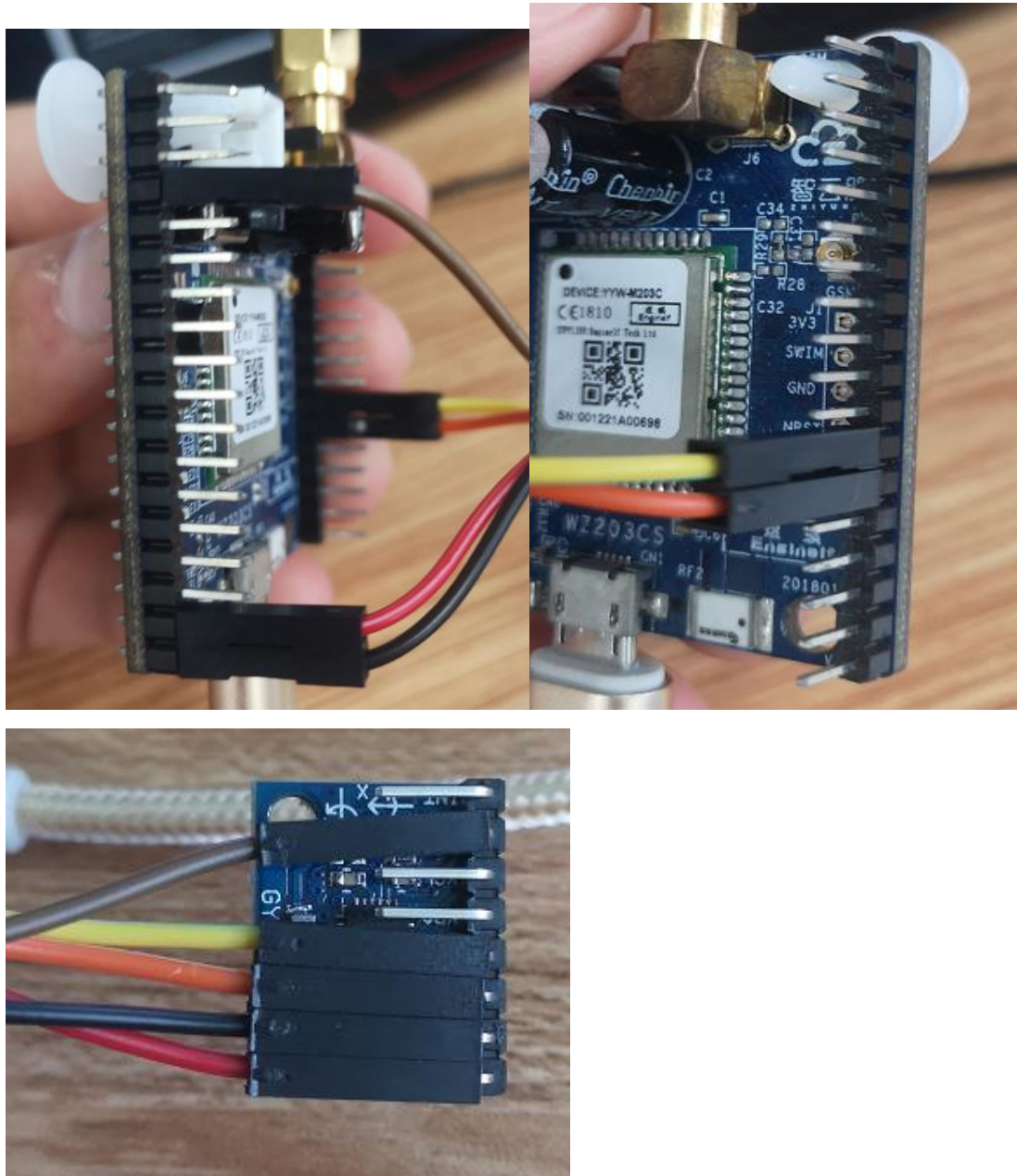
删除产品和小程序：在第五步发布里面，右下角有删除功能，此删除就是删除创建的产品和小程序。值得注意的是开发时右上角的发布不能点，点了之后该产品就无法修改数据点，无法调试，无法删除了。

最后需要联系智云服工作人员，在阿里云上把对应产品的服务端订阅打开。即可使用这个产品和小程序。否则小程序无法接收到数据。我们这边后台处理过该产品之后，除非创建新的产品，否则不需要我们再处理。

本工程的主要功能：实现 GPS 定位功能并上传 GPS 数据到服务器；实现深度休眠并定时唤醒，深度休眠功耗为 200 多 uA；实现 MPU6050 外接后六轴数值获取并上传到服务器；实现运动阈值的判断并进行事件响应；通过小程序可以查看数据并配置功能。

MPU6050 连接:

如下图



引脚连接: SCL-RI, SDA-DCD, VCC-3V3, GND-GND, ADO-VBAT

软件配置:

```
void proc_subtask2(s32 TaskId)
{
    Ql_Sleep(3000);
    MPU_Wakeup();
    while(1)
    {
        Ql_Sleep(200);
        MPU_Get_Accelerometer();
        MPU_Get_Gyroscope();
    }
}

void proc_subtask1(s32 TaskId)
{
    Ql_Sleep(1000);
    while(1){
        Ql_Sleep(200);
        if(mpu6050SportLevel >= SPORTLEVEL)
        {
            // 部署任务
            // 例如:
            mprintf("mpu6050SportLevel:%d\tSPORTLEVEL=%d\r\n", mpu6050SportLevel, SPORTLEVEL);
            #if 0
            if(flag_sms)
                SMS_TextMode_Send();//发送短信
            #endif
            mpu6050SportLevel = 0;
        }
    }
}
```

实时获取六轴的值

mpu6050运动超过了SPORTLEVEL等级就会走这里，可以在这里编程自己的任务

扫描根目录下的小程序二维码，进入小程序。然后搜索自己的 sn 号，此 sn 号为前面写入的 sn 号，不是标签上贴的 sn 号。

在小程序里可以看到加速度和陀螺仪的 xyz 值，并且可以配置运动阈值。

运动阈值对应 mpu6050 状态：

- 0: mpu6050 完全静止不动。
- 1: mpu6050 稍微有一点点动。
- 2: mpu6050 一般运动。
- 3: mpu6050 快速运动。
- 4: mpu6050 非常快运动。

2, 3, 4 的运动等级有误差，有兴趣的朋友可以自行修改程序，获取运动等级。

最后，设置好阈值，接好 mpu6050，写好 sn 号。开机，晃动 mpu6050，即可触发软件配置的任务。

GPIO:

通过 Q1_GPIO_Init 函数进行初始化为输出模式，通过 Q1_GPIO_SetLevel 设置输出高低电平，来控制 LED 灯的开关，由下图可知刚一开机 LED1 和 LED1 会都亮一下。

```
Enum_PinName LED1=PINNAME_SIM2_CLK;  
Enum_PinName LED2=PINNAME_SIM2_DATA;  
Enum_PinName LED3=PINNAME_SIM2_RST;
```

```
void power_drv_init(void)  
{  
    Q1_GPIO_Init(LED1, PINDIRECTION_OUT, 0, PINPULLSEL_PULLUP);  
    Q1_GPIO_Init(LED2, PINDIRECTION_OUT, 0, PINPULLSEL_PULLUP);  
  
    LED1_H;|  
    LED2_H;  
}
```

```
:  
: #ifdef _USE_LED_MIS_  
:  
: #define LED1_L    Q1_GPIO_SetLevel(LED1, PINLEVEL_LOW)  
: #define LED1_H    Q1_GPIO_SetLevel(LED1, PINLEVEL_HIGH)  
: #define LED2_L    Q1_GPIO_SetLevel(LED2, PINLEVEL_LOW)  
: #define LED2_H    Q1_GPIO_SetLevel(LED2, PINLEVEL_HIGH)  
: #define LED3_L    Q1_GPIO_SetLevel(LED3, PINLEVEL_LOW)  
: #define LED3_H    Q1_GPIO_SetLevel(LED3, PINLEVEL_HIGH)  
:  
: #else  
:  
:  
:
```


UART:

在 UartInit 函数中通过 QI_UART_Register 函数注册 UART1 和 UART2，传的第二个参数是 Callback_UART_Hdlr，这是一个回调函数。相当于中断处理函数，即用来处理 UART 事件。如下图，EVENT_UART_READY_TO_READ 事件就是读取到串口有数据，然后通过判断是哪个串口走到对应的函数进行处理。如果是 UART_PORT1 则走到 User_Command_Ays；如果是 UART_PORT2 则走到 Uart2UserCommandEncode 里面。

```
4:
5: static void Callback_UART_Hdlr(Enum_SerialPort port, Enum_UARTEventType msg, bool level, void* param)
6: {
7:     s32 lens;
8:     u8 RxBuf_Uart[800] = {0};
9:     s32 iRet = 0;
10:    switch (msg)
11:    {
12:        case EVENT_UART_READY_TO_READ:
13:        {
14:            lens = ReadSerialPort(port, RxBuf_Uart, sizeof(RxBuf_Uart));
15:            if (lens < 0) break;
16:            else
17:            {
18:                if (port == UART_PORT1) User_Command_Ays(RxBuf_Uart, lens);
19:                if (port == UART_PORT2) Uart2UserCommandEncode(RxBuf_Uart, lens);
20:            }
21:        }
22:        break;
23:        case EVENT_UART_READY_TO_WRITE:
24:        break;
25:        default:
26:        break;
27:    }
28:    } // end switch msg
29: } // end callback_UART_Hdlr
30:
31:
32: void UartInit(void)
33: {
34:     s32 ret;
35:     ret = QI_UART_Register(UART_PORT1, Callback_UART_Hdlr, NULL);
36:     ret = QI_UART_Open(UART_PORT1, 115200, FC_NONE);
37:     ret = QI_UART_Register(UART_PORT2, Callback_UART_Hdlr, NULL);
38:     ret = QI_UART_Open(UART_PORT2, 9600, FC_NONE);
39: }
```

在 User_Command_Ays 里可以添加自己的串口命令，如开发包已经有的写入 sn 号的串口命令：

```
void User_Command_Ays(u8* buf, u8 len)
{
    u8* p;
    u8 mybuf[20];
    u16 tem;
    p = (u8*)strstr((const char*)buf, "$write"); alimqtt版本写入sn
    if (p != NULL)
    {
        Get_mqtt_conf(p);
        systemset.snuser=1;
        SaveFlashParamsNew(&systemset);
        printf("+SN %s\r\n", (u8*)systemset.SN);
        printf("+secret %s\r\n", (u8*)systemset.secret);
        printf("+key %s\r\n", (u8*)systemset.key);
        ble_server_sta=BLESERVER_DEVICE_PREINIT0;
        SendMsg2KernelForBLEStart();
        SendMsg2KernelForCheckGSMorGPRS();
    }
    p = (u8*)strstr((const char*)buf, "$setsn"); mqtt版本写入sn
    if (p != NULL)
    {
        QI_memset(systemset.SN, 0, sizeof(systemset.SN));
        Get_Str_Use(systemset.SN, p);
        systemset.snuser=1;
        SaveFlashParamsNew(&systemset);
        printf("+SN %s\r\n", (u8*)systemset.SN);
        ble_server_sta=BLESERVER_DEVICE_PREINIT0;
        SendMsg2KernelForBLEStart();
        SendMsg2KernelForCheckGSMorGPRS();
    }
}
```

Timer:

先通过 Ql_Timer_Register 注册指定的定时器，再通过 Ql_Timer_Start 开启定时器，Ql_Timer_Start 的第二个参数代表定时器的周期，最短为 1ms。然后在指定的回调函数如 Callback_led_Timer 中，进行编程，需要注意的是定时器中不能加处理时间过长的事件如 sleep 等。通过 Ql_Timer_Stop 可以停止定时器。

```
void Led_Timer_init(u32 TIMER_ID, u32 ms)
{
    Ql_Timer_Register(TIMER_ID, Callback_led_Timer, NULL);
    Ql_Timer_Start(TIMER_ID, ms, TRUE);
}

static void Callback_led_Timer(u32 timerId, void* param)
{
    if(Ql_GPIO_GetLevel(LED1))
    {
        LED1_H;
    }
    else
    {
        LED1_L;
    }
}
```

GPS:

上述文档中，需要关心的是函数 GpsOpen 打开 GPS 功能，函数 LbsAEpoStalnit 可以打开 EPO 功能，EPO 功能是辅助 GPS 定位用的。在 GPS 打开、EPO 打开之后，调用 RIL_GPS_Read 循环读取 GPS 数据，如果 GPS 信号良好，即能完成定位。

```
s32 GetGpsLocation(u32 timeout, u8 op)
{
    u8 times=0;
    u8 rdBuff[1000];
    s32 iRet = RIL_AT_SUCCESS;
    s8 ptTimeout = 0;
    double temp=0.000;

    if (op)
    {
        if(gpspowersta == 0)iRet = GpsOpen();
        if (RIL_AT_SUCCESS != iRet)
        {
            mprintf("Power on GPS fail, iRet = %d.\r\n", iRet);
            return -1;
        }
        else
        {
            SendMsg2KernelForSendPMTKData();
            Q1_Sleep(500);
            mprintf("my_tcp_state = %d\r\n", my_tcp_state);
            if(my_tcp_state == STATE_GPRS_REGISTER)
            {
                mprintf("_EPO_ENABLE_\r\n");
                Q1_Sleep(1000);
                LbsAEpoStalnit();
            }
            Q1_memset(rdBuff, 0, sizeof(rdBuff));
            while (timeout)
            {
                timeout--;
                times++;
                Q1_Sleep(1000);
                iRet = RIL_GPS_Read("ALL", rdBuff); //RMC GGA
                if (RIL_AT_SUCCESS != iRet)
                {
                    mprintf("Read %d information failed.\r\n", iRet);
                }
                else
                {
                    mprintf("%s\r\n", rdBuff);
                    mprintf("ll:%s,la:%s\r\n",&lbsJingdu,&lbsWeidu);
                    GpsAnalysis((u8*)rdBuff);
                    GPS_package(&gpsx, &my_core_data);
                }
                mprintf("satellitenum %d\r\n",gpsx.satellitenum);
                if(gpsx.useorno != 65 && times > 10 && gpsx.satellitenum < 4) //gps 信号不好 10秒搜不到星推出
                {
                    mprintf("gps rssi is too low\r\n");
                }
            }
        }
    }
}
```


MQTT:

```
void Mqtt_InitConnect_Start(void)
{
    u8 cent=0;
    s32 ret;
    int app_retcode=0;
    do {
        switch (user_mqtt_clint_sta)
        {
            case USER_MQTT_GPRS_INIT: //检查网络状态
                ret=Check_SystemGprsSta(70); //35s 超时
                if(ret!=0)
                {
                    user_mqtt_error("!!!MQTT GPRS CHECK ERROR!!!");
                    printfErr2F("!!!MQTT GPRS CHECK ERROR!!!\r\n");
                    savemode = -1;
                    SendMsg2KernelForGetLocation(&savemode);
                    return ;
                    //break;
                }
                LBSDataInit();
                user_mqtt_clint_sta=USER_MQTT_PARMA_INIT;
                break;
            case USER_MQTT_PARMA_INIT: //初始化用户MQTT参数
                user_mqtt_yeinfo.heartping=0;
                alink_mqtt_topic_for_reg_init();
                alink_mqtt_topic_for_pub_init();
                alink_mqtt_topic_for_sub_init();
                user_mqtt_info("topic_for_alinkpub=%s",topic_for_alinkpub);
                user_mqtt_info("topic_for_alinksub=%s",topic_for_alinksub);
                user_mqtt_clint_sta=USER_MQTT_AUTH_INIT;
                break;
            case USER_MQTT_AUTH_INIT: //初始化mqtt用户登录信息
                ret=user_get_puser_info();
                if(ret==0)
                {
                    user_mqtt_auth_init();
                    user_mqtt_clint_sta=USER_MQTT_CONNECTING;
                }
                break;
            case USER_MQTT_CONNECTING: //mqtt连接服务器
                app_pclient = IOT_MQTT_Construct(&mqtt_params);
                if (NULL == app_pclient)
                {
                    user_mqtt_error("MQTT construct failed");
                    break;
                }
                user_mqtt_info("MQTT construct OK.");
                app_retcode = IOT_MQTT_Subscribe(app_pclient, topic_for_alinksub, IOTX_MQTT_QOS1, aliot_mqtt_msg_arrived, NULL);
                if (app_retcode < 0)
                {
                    IOT_MQTT_Destroy(&app_pclient);
                    user_mqtt_error("subscribe failed, retcode = %d", app_retcode);
                    break;
                }
                user_mqtt_clint_sta = USER_MQTT_PUB_REGISTER_MSG;
                user_mqtt_info("MQTT connect OK.");
                break;
            case USER_MQTT_PUB_REGISTER_MSG: //发送注册信息
                memset(&topic_msg, 0x0, sizeof(iotx_mqtt_topic_info_t));
                topic_msg.qos = IOTX_MQTT_QOS1;
                topic_msg.retain = 0;
                topic_msg.dup = 0;
                topic_msg.payload = (void*)mqtt_msg_pub;
                topic_msg.payload_len = alink_msg_data(mqtt_msg_pub,0);
                topic_msg.payload_len = zyf_msg_register_data(mqtt_msg_pub,0);
                app_retcode = IOT_MQTT_Publish(app_pclient, topic_for_reg, &topic_msg);
                if (app_retcode < 0)
                {
                    user_mqtt_error("error %d when publish", app_retcode);
                    user_mqtt_clint_sta = USER_MQTT_CONNECTING;
                    break;
                }
                user_mqtt_clint_sta = USER_MQTT_CLINT_OK;
                user_mqtt_info("USER_MQTT_CLINT_OK");
                Q1_Sleep(1000);
                SendMsg2KernelForSendSaveData();
                break;
            default:
                if(user_mqtt_clint_sta==USER_MQTT_CLINT_OK){
                    user_mqtt_info("USER_MQTT_CLINT_OK");
                }
                else {
                    user_mqtt_error("should not goto here %d", __LINE__);
                }
                break;
        }
    }
}
```

- 1、检测网络状态：保证 GPRS 网络正常
- 2、初始化用户 MQTT 参数
用于发布的主题：

```

void alink_mqtt_topic_for_pub_init(void)
{
    char product_key[IOTX_PRODUCT_KEY_LEN] = {0};
    char device_name[IOTX_DEVICE_NAME_LEN] = {0};

    HAL_GetProductKey(product_key);
    HAL_GetDeviceName(device_name);
    Q1_memset(topic_for_alinkpub, 0, sizeof(topic_for_alinkpub));

    #if 1
        Q1_strcat(topic_for_alinkpub, "/");
        Q1_strcat(topic_for_alinkpub, product_key);
        Q1_strcat(topic_for_alinkpub, "/");
        Q1_strcat(topic_for_alinkpub, device_name);
        Q1_strcat(topic_for_alinkpub, "/user/post");
    #endif
}

```

用于订阅的主题:

```

void alink_mqtt_topic_for_sub_init(void)
{
    char product_key[IOTX_PRODUCT_KEY_LEN] = {0};
    char device_name[IOTX_DEVICE_NAME_LEN] = {0};

    HAL_GetProductKey(product_key);
    HAL_GetDeviceName(device_name);
    Q1_memset(topic_for_alinksub, 0, sizeof(topic_for_alinksub));

    #if 1
        Q1_strcat(topic_for_alinksub, "/");
        Q1_strcat(topic_for_alinksub, product_key);
        Q1_strcat(topic_for_alinksub, "/");
        Q1_strcat(topic_for_alinksub, device_name);
        Q1_strcat(topic_for_alinksub, "/user/set");
    #endif
}

```

用于注册的主题:

```

void alink_mqtt_topic_for_reg_init(void)
{
    char product_key[IOTX_PRODUCT_KEY_LEN] = {0};
    char device_name[IOTX_DEVICE_NAME_LEN] = {0};

    HAL_GetProductKey(product_key);
    HAL_GetDeviceName(device_name);

    #if 1
        Q1_strcat(topic_for_reg, "/");
        Q1_strcat(topic_for_reg, product_key);
        Q1_strcat(topic_for_reg, "/");
        Q1_strcat(topic_for_reg, device_name);
        Q1_strcat(topic_for_reg, "/user/register");
    #endif
}

```

3、初始化 MQTT 登录信息

```

s8 user_get_puser_info(void)
{
    u32 res = 0;
    iotx_dev_meta_info_t meta;
    memset(&meta, 0, sizeof(iotx_dev_meta_info_t));
    memset(&g_mqtt_signout, 0, sizeof(iotx_sign_mqtt_t));

    HAL_GetProductKey(meta.product_key);
    HAL_GetDeviceName(meta.device_name);
    HAL_GetDeviceSecret(meta.device_secret);

    res = IOT_Sign_MQTT(IOTX_CLOUD_REGION_SHANGHAI, &meta, &g_mqtt_signout);
    if (res == 0)
    {
        user_mqtt_info("signout.hostname: %s", g_mqtt_signout.hostname);
        user_mqtt_info("signout.port : %d", g_mqtt_signout.port);
        user_mqtt_info("signout.clientid: %s", g_mqtt_signout.clientid);
        user_mqtt_info("signout.username: %s", g_mqtt_signout.username);
        user_mqtt_info("signout.password: %s", g_mqtt_signout.password);

        memset(&mqtt_params, 0, sizeof(iotx_mqtt_param_t));
        mqtt_params.port = g_mqtt_signout.port;
        mqtt_params.host = g_mqtt_signout.hostname;
        mqtt_params.client_id = g_mqtt_signout.clientid;
        mqtt_params.username = g_mqtt_signout.username;
        mqtt_params.password = g_mqtt_signout.password;

        return 0;
    }

    return -1;
} « end user_get_puser_info »

```

```

signout.hostname: a12YKLwgXmc.iot-as-mqtt.cn-shanghai.aliyuncs.com
signout.port : 1883
signout.clientid: a12YKLwgXmc.111122220000 |
timestamp=2524608000000, securemode=3, signmethod=hmacsha256, gw=0, ext=0, _v=sdk-c-3.0.1 ||
signout.username: 111122220000@a12YKLwgXmc
signout.password: 2623D5497DF946CD5042F6D6482E8475C0B2E6786312851BFA1B4523EB6BBB5E

```

由打印数据可以看到，这里是在用在智云服生成的设备 SN 和 productkey 登陆到阿里云平台，user_mqtt_autu_init 用于设置基本的 mqtt 参数。

4、MQTT 连接服务器

通过 IOT_MQTT_Construct 连接到阿里云，再通过 IOT_MQTT_Subscribe 注册一个回调函数 aliot_mqtt_msg_arrived。此函数的功能如果是如果订阅的主题有消息发布，就会走到这个回调函数进行处理这个消息。

5、发送注册主题

向智云服发送 productkey、sn 等数据，作用相当于在智云服激活目前的设备，不然在小程序无法搜索到这个设备信息。

低功耗:

注意使用到的函数为 zyf_protocol.c 里面定义的，不是 uart.c 里面定义的。

1、只休眠 STM8:

```
setmcu2eint();  
setmcu2sleep(min);  
setmcu2pinoff();  
...
```

发送命令让stm8休眠

2、休眠 STM8 的同时也休眠 203C 并定时全部唤醒:

```
setmcu2poweroff203c(&msg.param1);  
Q1_PowerDown(1);
```

