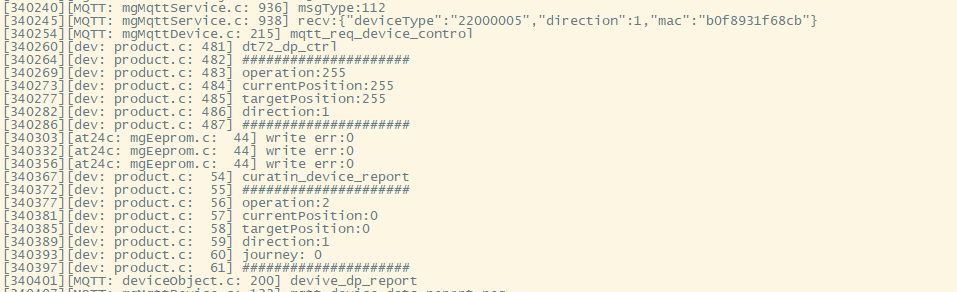
BUG：

1. 设置电机方向时

第一次反转



第二次反转



Operation 255 –> 2 ?

已解决。

1. 原证书验证失败后，从http下载得到的证书，每次开机都要重新下载，无法保持。

已解决。 通过写入模块板载的外部2M flash 中

1. 在正常模式时，APP拖拽位置，电机在执行过程中，APP在此间会每隔一段时间要求上报设备状态。可能会引起dev\_status参数被curtainMsgReport();更改

已解决。在同一进程中，不会同时发生对该参数写，且另外声明report\_dev\_status

1. 只有设备端接收到服务器的指令， 才能将消息队列中的消息发送给服务器

已解决。Select 等待时间并不是NULL，不是阻塞。

1. 设置行程时，dev\_status.direction 该变量没有重新读取eeprom获取

已解决。

1. 存储设备运行的log信息

待解决

1. 增加wifi强度rssi上报功能

待解决

1. 重复设置0~100位置(不包括0，100)，运行多次后，电机发热，行程改变较大

待解决。 方案：设置在运行n次后，默认去运行到0(或100)，从而更新行程

1. 上个星期遗留问题: device\_code eeprom中默认有值，但是验证失败。

已解决。

处理: 初始化(清除)eeprom

mg\_eeprom\_write(SYSTEM\_EE\_ADDR, 0x0, 1024);

mg\_eeprom\_write(KERNEL\_EE\_ADDR, 0x0, 128);

mg\_eeprom\_write(DEVICE\_EE\_ADDR, 0x0, 5120);

mg\_eeprom\_write(SCENE\_EE\_ADDR, 0x0, 25600);

mg\_eeprom\_write(TIMER\_EE\_ADDR, 0x0, 25600);

mg\_eeprom\_write(ASTIME\_EE\_ADDR, 0x0, 2048);

PS: 在moorgen\_service\_init函数中，将打印device\_code信息 删除(完成后)

1. 开机会主动上报所有设备，上报内容只要“mac”,”deviceType”字段

已解决。在void mqtt\_device\_alldp\_report()中，devive\_dp\_report()🡪curtainMsgReport

优化:

1. Eeprom 频繁读写
2. 当在APP拖拽完位置后， 人为手动按键控制电机转动与系统当时记录的位置不匹配，倒是后续APP操作时，都无法运行拖拽设置的位置
3. 第一次设置位置时，原先open，再从位置0到设置的位置

状态:

1设置行程，

1）进入设置行程界面， 在第1菜单(点动)，不会复位 行程标志位

在第2菜单(open or close ) ，会复位行程标志位

2）行程时间B2T\_2，T2B\_t 清0

1. 关闭了 #define mqtt\_log(M, ...) //custom\_log("MQTT", M, ##\_\_VA\_ARGS\_\_)

mgAesEncrypt(udp\_server.buf,udp\_server.aes\_buf,moorgen\_userkey\_get(),udp\_server.bufLen);

mgAesDecrypt(uint8\_t \*in,uint8\_t\*out,uint8\_t \*key, int length);

1. 将MQTT 等待接收消息队列 等待时间 100- > 500
2. void mqtt\_send\_to\_queue( mqtt\_packet\_t \*mqtt\_packet\_send\_p) ,在函数中 处理 消息队列满(注释掉了当发送消息队列为full 时， 释放一个消息)

开机：

1.

#define REGISTER\_TYPE\_DEVICECODE "deviceCode"

不为0(默认为0)，表示该设备已在服务器注册

若未注册，则发送：send msgType:1，"mac", "deviceType"，"channelCode" 进行注册

若已注册，则发送: send:{ "fwVersion": "0.1.7", "intranet": "192.168.2.128", "ssid": "Dooya\_PH" }

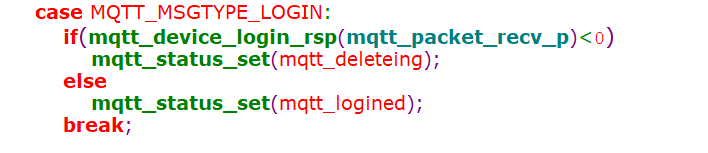
2.

在上一步发送注册信息(即登陆)， 服务器返回: {"accessToken":"46b4d619-ddb0-4341-a874-5b5592ab7e7e","refreshToken":"87e17deb-44a9-45d7-97aa-00ace1d6267a","retCode":"0000"}

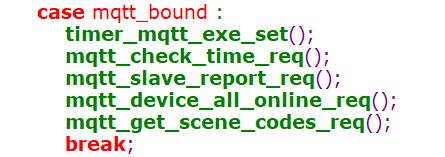
判断retCode 值是否为”0000”, 表示登陆成功。 并将accessToken ， refreshToken存入MOORGEN\_SYS\_INFO 中

3.

登陆后，根据



判断是否设备已经在服务器端被删除，若没有，则进入mqtt\_status\_set(logined)



1. timer\_mqtt\_exe=1;
2. mqtt\_log("mqtt\_check\_time\_req"); msgType:19
3. 若为hub，上报管理的所有从机
4. 若为hub，上报在线的从机
5. 场景请求， msgType:20

4.

接收到服务器: msgType:19 (mqtt时间请求回复)

recv:{"retCode":"0000","time":"2018-09-18 08:44:08","timeZone":"Asia/Shanghai"}



将"time":"2018-09-18 08:44:08" 的时间信息更新为系统时间(写入RTC)。

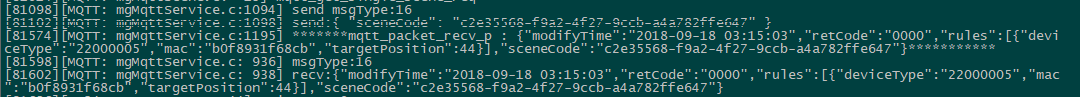
接收到服务器: msgType:20(场景信息)



recv:{"currentPage":0,"pageAlias":"8f80ykm1","retCode":"0000","scenes":[],"totalPage":0}

scenes“”[]数组中存放”sceneCode” ，"modifyTime" 信息。

获取“scenes“”[]数组个数(即在APP设置的场景数量)，一个个查找对比eeprom中写入的场景（根据"sceneCode"），是否存在,不存在，则向服务器发送mqtt\_get\_single\_scene\_req(sceneCode);从服务器获取该场景的信息，

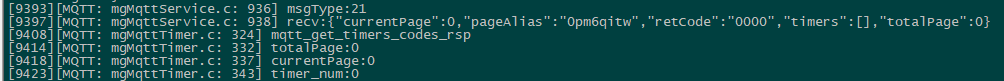
 若存在，对比"modifyTime" 是否一致，不一致，则向服务器发送mqtt\_get\_single\_scene\_req(sceneCode); 跟上一步(不存在)一样处理

最后发送了 send msgType:21 (获取定时器信息)



5.

根据上一步，发送定时器请求，收到服务器



处理跟上一步场景类似。

最后又执行了

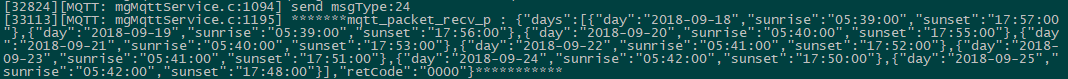
1. moorgen\_sys\_info.is\_local\_server = 0
2. mqtt\_device\_alldp\_report(); //将"operation" ,"currentPosition", "targetPosition",

// "direction", "currentState"

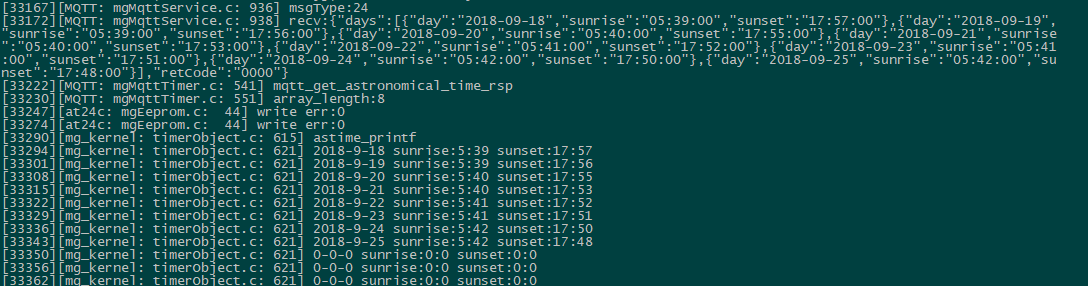


7.

在，开机30s左右会主动请求天文时时间。



并将服务器发来的连续8天天文时时间存储。



指令码:

typedef enum

{

MQTT\_MSGTYPE\_ERR = 0,

MQTT\_MSGTYPE\_REGISTER = 1,

MQTT\_MSGTYPE\_ACTIVITE = MQTT\_MSGTYPE\_REGISTER,

MQTT\_MSGTYPE\_LOGIN,

MQTT\_MSGTYPE\_SLAVE\_REPORT,

MQTT\_MSGTYPE\_STATUS\_REPORT,

MQTT\_MSGTYPE\_CLOUD\_STORAGE,

MQTT\_MSGTYPE\_DEVICE\_ONLINE,

MQTT\_MSGTYPE\_DEVICE\_OFFLINE,

MQTT\_MSGTYPE\_COMMISSION\_RESULT,

MQTT\_MSGTYPE\_COMMAND\_RESULT,

MQTT\_MSGTYPE\_ERROR\_REPORT,

MQTT\_MSGTYPE\_TRANSPARENT\_REPORT,

MQTT\_MSGTYPE\_GET\_STORAGE,

MQTT\_MSGTYPE\_DATA\_REPORT,

MQTT\_MSGTYPE\_BIND\_USER,

MQTT\_MSGTYPE\_GET\_SCENES,

MQTT\_MSGTYPE\_GET\_SINGLE\_SCENE,

MQTT\_MSGTYPE\_GET\_TIMERS,

MQTT\_MSGTYPE\_GET\_SINGLE\_TIMER,

MQTT\_MSGTYPE\_GET\_SYSTEM\_TIME = 19,

MQTT\_MSGTYPE\_GET\_SCENES\_CODE = 20,

MQTT\_MSGTYPE\_GET\_TIMERS\_CODE,

MQTT\_MSGTYPE\_GET\_USER\_CODE = 22,

MQTT\_MSGTYPE\_CLIENT\_TIMER\_NOTIFY,

MQTT\_MSGTYPE\_GET\_ASTRONOMICAL\_TIME,

MQTT\_MSGTYPE\_RATE\_PUSH\_REPORT=26,

MQTT\_MSGTYPE\_REQUEST\_MAX,

MQTT\_MSGTYPE\_SUBSCRIBE\_START = 100,

MQTT\_MSGTYPE\_REBOOT = MQTT\_MSGTYPE\_SUBSCRIBE\_START,

MQTT\_MSGTYPE\_RELOGIN,

MQTT\_MSGTYPE\_RESET,

MQTT\_MSGTYPE\_DEVICE\_REPORT\_SUBSCRIBE,

MQTT\_MSGTYPE\_START\_COMMISSION,

MQTT\_MSGTYPE\_DELETE\_SLAVE,

MQTT\_MSGTYPE\_CLOSE\_SLAVE,

MQTT\_MSGTYPE\_OPEN\_SLAVE,

MQTT\_MSGTYPE\_CONFIGURATION,

MQTT\_MSGTYPE\_UPDATE,

MQTT\_MSGTYPE\_TRANSPARENT,

MQTT\_MSGTYPE\_DISCONNECT,

MQTT\_MSGTYPE\_DEVICE\_CONTROL,

MQTT\_MSGTYPE\_UPDATE\_SCENE,

MQTT\_MSGTYPE\_UPDATE\_TIMER,

MQTT\_MSGTYPE\_SCENE\_CONTROL,

MQTT\_MSGTYPE\_DEL\_SCENE,

MQTT\_MSGTYPE\_DEL\_TIMER,

MQTT\_MSGTYPE\_REGET\_USER\_CODE,

MQTT\_MSGTYPE\_ASTRONOMICAL\_TIME\_GET,

MQTT\_MSGTYPE\_UNBIND,

MQTT\_MSGTYPE\_CLEAR\_SCENE,

MQTT\_MSGTYPE\_CLEAR\_TIMER,

MQTT\_MSGTYPE\_SUBSCRIBE\_MAX

} mqtt\_msgtype\_t;