基于 TA 模块的 MCU 应用设计示例

版本 V1.2

2018/3/16

关于本示例

本示例以基于 TA 模块控制的 WIFI 灯为例，介绍 TA 模块的作用以及 TA 模块相关的指令集的使用，包含以下章节。

章	标题	内容
第 1 章	概述	TA 模块的作用
第 2 章	TA 模块指令集用法	用具体实例阐述指令集用法
第 3 章	指令调试	详解通过串口工具、web 端和 APP 进行指令调试
第 4 章	单片机代码示例	提供解析指令集的示例代码
第 5 章	TA 模块使用注意事项	使用 TA 模块应注意的问题

发布说明

日期	版本	发布说明	编制	审核
2018.1.10	V1.1	第二次发布	明国锋	武鹏飞
2018.3.16	V1.2	添加示例代码	明国锋	武鹏飞

1. 概述

TA模块是一个网络透传模块，主要作为MCU与服务器沟通的桥梁。其具体扮演的角色如下图所示（图中数字从小到大排序，依次表示指令的传输方向）：

图 1 ，首先操作MCU端，由MCU端首先发送相应的操作指令：

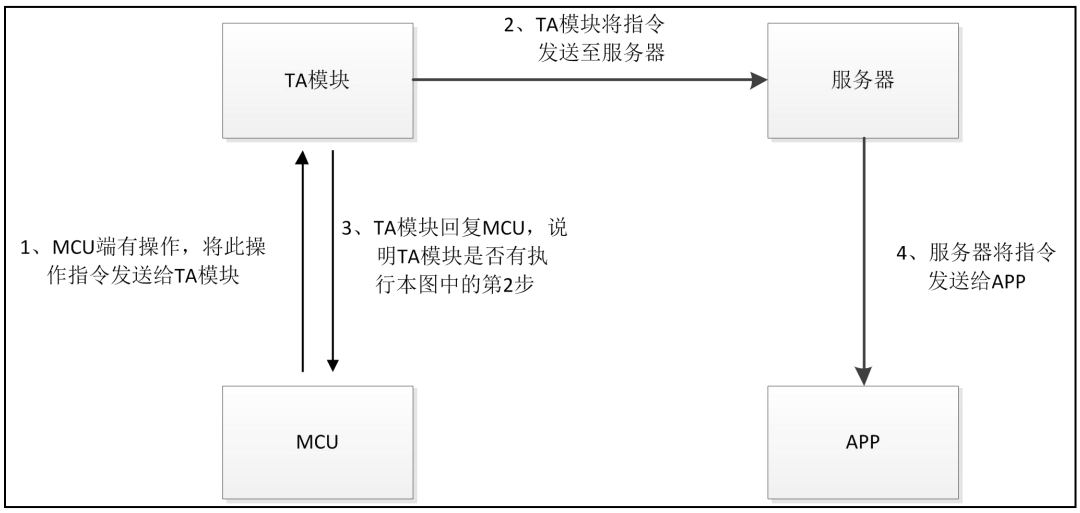
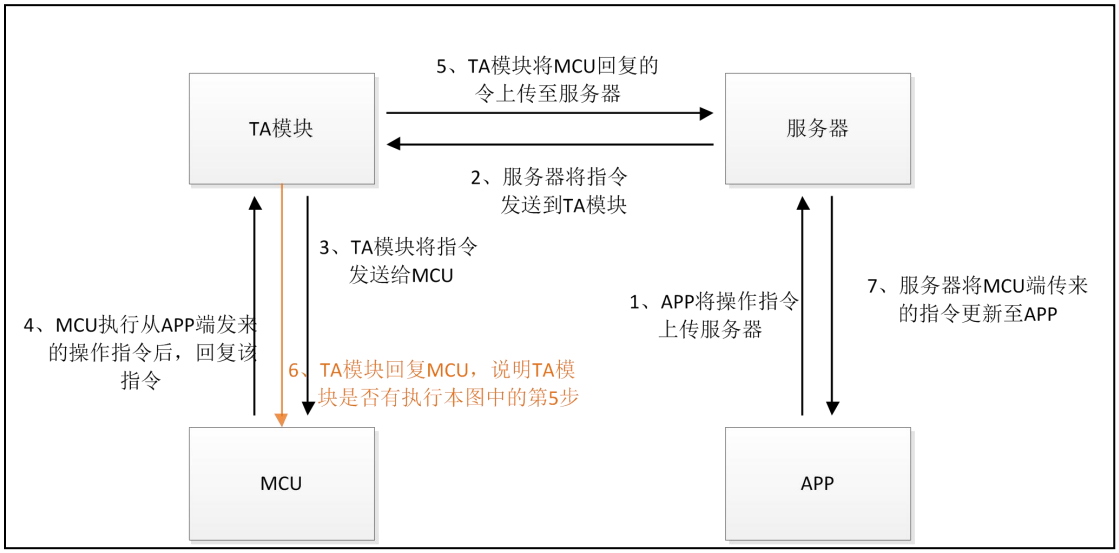


图 2，首先操作 APP 端，由 APP 端首先发送相应的操作指令：



注：图 2 中的褐色箭头与褐色文字相对应。

2. TA 模块指令集用法

2.1、本示例以可用 APP 控制的 wifi 灯为例，简述相关的 AT 指令集的用法。

自定义字段示例：

Wifi 灯开关动作控制：switch

控制开：on

控制关：off

type: 101

Wifi 灯亮度控制：bright

亮度等级：0-100

type: 102

2.2、MCU 发送给 TA 模块的指令以及 TA 模块发送给 MCU 的指令示例（见表 1、表 2）：

注意：

（1）、表中所示指令均为完整指令，包括加号、双引号、等于号、冒号、问号，每条指令均以 ESC 结尾，ESC 是一个字符，其 ASSIC 编码为 0X1B。每条完整指令中没有空格、回车、中文字符！Type 字段和 repeat 字段一定不能丢！由于 word 文档中无法显示 ESC 这个字符，故本节中用“←”表示 ESC。

（2）、指令中所有的数字均为字符，涉及到的标点符号均为英文符号。

（3）、下表中未提及的指令用法与表中指令类似，具体格式参考 TA 模块指令集。

（4）、MCU 发送给 TA 模块单帧数据最大不可超过 256 字节，发送间隔不可小于 200 毫秒。发送频率过快可能会导致 TA 模块重启！

表 1：

<table>
<tr><th>指令含义</th><th>MCU 发送给 TA 模块的指令</th><th>TA 模块回复</th></tr>
<tr><td>开 灯</td><td>AT+UPDATE=”switch”:”on”,”type”:101,”repeat” :1←</td><td rowspan="3">AT+SEND=ok←
表示 TA 模块已将 MCU 端发来的指令发送给服务器

或

AT+SEND=fail←
表示 TA 模块没有把 MCU 端发来的指令发送给服务器
（原因见表下注释 3）</td></tr>
<tr><td>关 灯</td><td>AT+UPDATE=”switch”:”off”,”type”:101,”repeat” :1←</td></tr>
<tr><td>调整亮度至 55（见表下注释 1）</td><td>AT+UPDATE=” bright” :55,”type”:102,”repeat” :1←</td></tr>
<tr><td>调整亮度至 100（见表下注释 2）</td><td>AT+UPDATE=” bright” :100,”type”:102,”repeat” :1←</td><td></td></tr>
<tr><td>查询 TA 模块工作状态</td><td>AT+STATUS?←</td><td>AT+STATUS=4←
表示已连上服务器
（详细描述见表下注释 4）</td></tr>
<tr><td rowspan="4">控制 TA 模块进入或退出配置模式</td><td>AT+SETTING=enterAP←</td><td rowspan="4">AT+SEND=ok←
或
AT+SEND=fail←</td></tr>
<tr><td>AT+SETTING=enterESPTOUCH←</td></tr>
<tr><td>AT+SETTING=exitAP←</td></tr>
<tr><td>AT+SETTING=exitESPTOUCH←</td></tr>
</table>

注释 1：调整亮度到 55：

指令中的 55 不是阿拉伯数字，而是两个字符‘5’，‘5’的 ASSIC 编码为 0X35。

注释 2：更新亮度到 100：

指令中的 100 不是阿拉伯数字，而是一个字符‘1’和两个字符‘0’，‘1’的 ASSIC 编码为 0X31，‘0’的 ASSIC 编码为 0X30。

注释 3：引起 TA 模块回复的指令为 AT+SEND=fail←的原因有以下几点：

* TA 模块没有连上网络
* MCU 发送的指令格式错误

注释 4： AT+STATUS=4←

此指令中的 4 是字符‘4’，ASSIC 值为 0X34。

此指令中的数字可能是其它数字，具体数字取决于 TA 模块的工作状态，具体描述如下：

0：TA 模块刚启动 1：加入 WIFI 失败 2：加入 WIFI 成功 3：开始注册设备至服务器

4：TA 模块连上服务器 5：TA 模块处于 touch 模式添加设备中 6：进入测试模式

7：烧写出厂数据 8：TA 模块正在升级 9：TA 模块处于 AP 模式添加设备中

注释 5： “sequence”这个字段是 APP 端首先操作发送给 MCU 端的设置（UPDATE）类指令中包含的字段，其后会紧跟一串由 13 位数字字符组成的序列号，收到此类指令后 MCU 端必须执行相应指令并立即回复，MCU 回复的指令中必须包含此字段，该字段后的序列号确认 MCU 回复的是 APP 端下发的哪条指令，确保 APP 端与 MCU 端的状态能正确同步。

表 2：

指令含义	TA 模块发送给 MCU 的指令	MCU 应回复
开灯指令	AT+UPDATE=”switch”:”on”,”sequence”:”0123456789123”←	AT+RESULT=”sequence”:”0123456789123”← 表示 MCU 成功接收数据。 其中”0123456789123“这串数字值不确定(长度固定为 13 位),MCU 回复时这串数字必须与 TA 模块发下来的指令中这串数字一模一样 （为何有这串数字见表上注释 5）
关灯指令	AT+UPDATE=”switch”:”off”,”sequence”:”0123456789123”←	
调整亮度至 55（见表上注释 1）	AT+UPDATE=”bright”:55,”sequence”:”0123456789123”←	
调整亮度至 100（见表上注释 2）	AT+UPDATE=”bright”:100,”sequence”:”0123456789123”←	
告知 MCU TA 模块已连上服务器	AT+START←	无需回复
定时开灯	AT+TIME=”swtch”:”on”←	这两条指令无需回复，但 MCU 端执行完开关动作后，需将最新状态上传至服务器，保证 APP 与 MCU 端状态同步
定时关灯	AT+TIME=”swtch”:”off”←	

3. 指令调试方法

3.1、串口调试工具

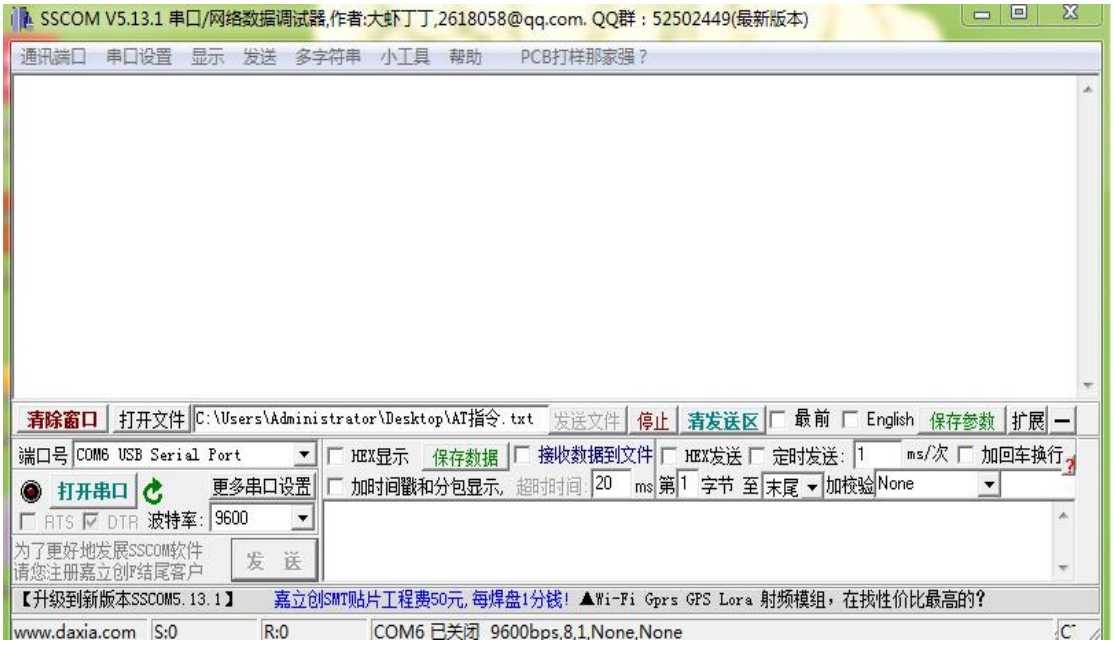
3.1.1、串口调试工具选择

网上可以找到很多种串口工具，但并不是每个串口工具都能显示 ESC 这个字符。在调试过程中，笔者用的串口工具是“sscom5.13.1”，串口通信涉及到的各种参数这个工具上基本都可以设置，我司提供的资料包中有该工具。用户也可用自己顺手的更好的串口工具进行调试。

本文中串口调试部分用的工具均为“sscom5.13.1”。

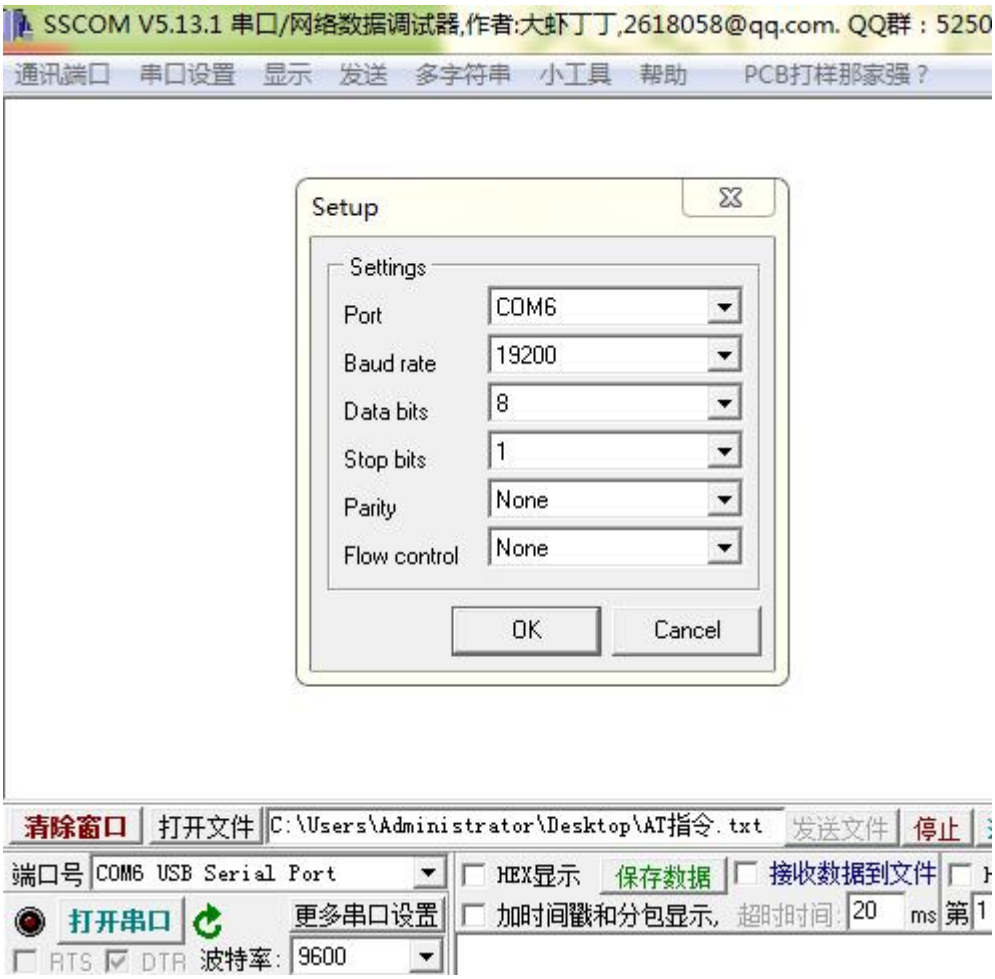
3.1.2、串口调试工具“sscom5.13.1”使用介绍

此工具无需安装，双击 exe 文件即可使用，打开界面如下：



调试 AT 指令时串口调试工具设置、使用步骤如下：

步骤一：点击工具左上角第二个选项“串口设置”，点击后在下拉菜单中选择“打开串口设置”，可看到如下界面：



步骤二：在上述界面中的 Port 选项中选择自己电脑上正在使用的串口号，Baud rate（波特率）选项中选择 19200，Data bits（数据位）选项中选择 8，Stop bits（停止位）选项中中选择 1，Parity（校验位）和 Flow control（流控）选项中都选择 None。选择完成后点击“OK”完成串口设置。

步骤三：测试电脑物理串口是否正常可用，短接电脑串口的“TX”和“RX”引脚，点击串口调试工具界面上的“打开串口”按钮，在发送框中随意输入若干字符，然后点击“发送”按钮，串口工具接收区能正常完全显示发送区的字符（如下图所示），则表示电脑串口正常。

步骤四：测试串口调试工具是否可正常显示 ESC 这个字符。

点击串口调试工具界面上紧贴发送区上边缘的“加时间戳和分包显示”这个选项，然后选中“HEX 发送”（即以 16 进制发送）选项，在发送区输入“1B”，保持物理串口的 RX 和 TX 短接，然后点击发送，可看到如下界面：

此串口调试工具有明显的汉字“收”、“发”来指示哪些数据是工具本身发送出去的，哪些是接收到的数据，“收”、“发”两个字之前中括号里的数据是时间戳，白色菱形和黑色菱形后是发送或接收的数据，通过工具自身发送出去的数据后会跟一个白色的正方形。

从上图可知，ESC 这个字符在此工具中显示为一个向左的箭头“←”，正式用串口发送指令时，复制接收区的“←”放在要发送的指令结尾即可（非 16 进制形式发送指令时）。

步骤五：通过串口调试工具发送正常完整指令

再次点击“HEX 发送”选项，取消 16 进制发送。在发送区里输入正常指令，结尾符 ESC 用上图中接收区的“←”替代，连接好 TA 模块与电脑的串口后，点击串口调试工具界面上的发送按钮即可，发送指令正常可看到如下图所示界面：

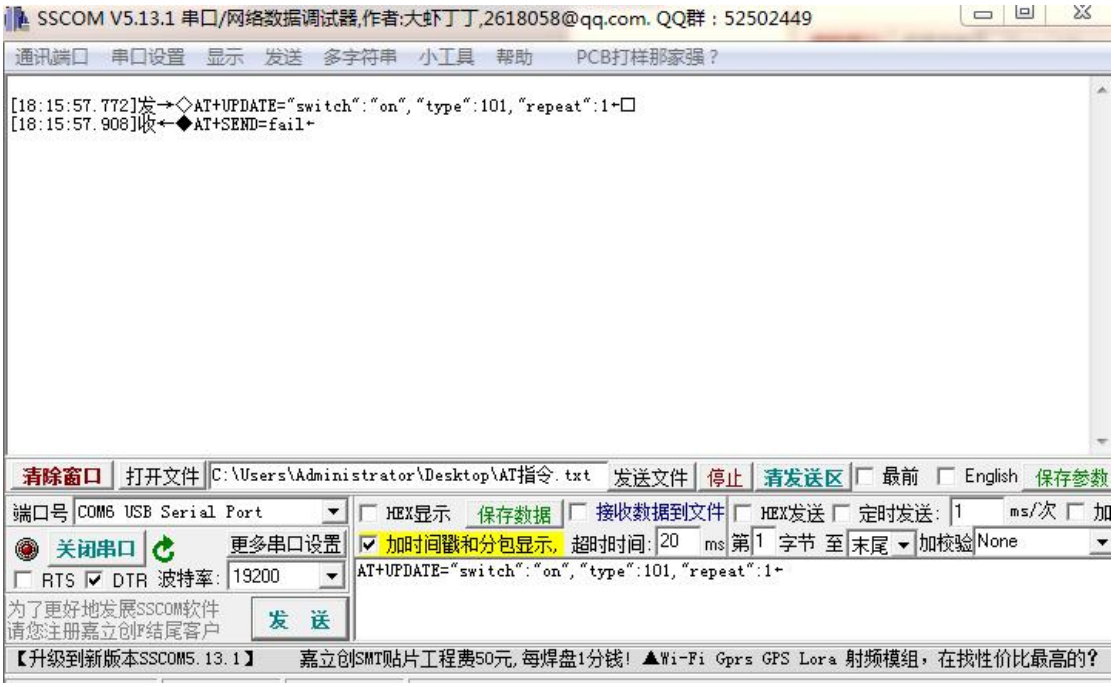
温馨提示：

用其它串口工具调试时，可短接串口的 TX 接 RX，用 16 进制发送 0X1B，用字符显示接收，如果接收有显示字符，这个字符就是该串口工具对 ESC 这个字符的显示，如果之后串口发送指令都是字符形式发送，复制它后把它放在每条指令最后即可，如果所有指令均以 16 进制数字发送，每条指令最后一个字节一定要是 0X1B。

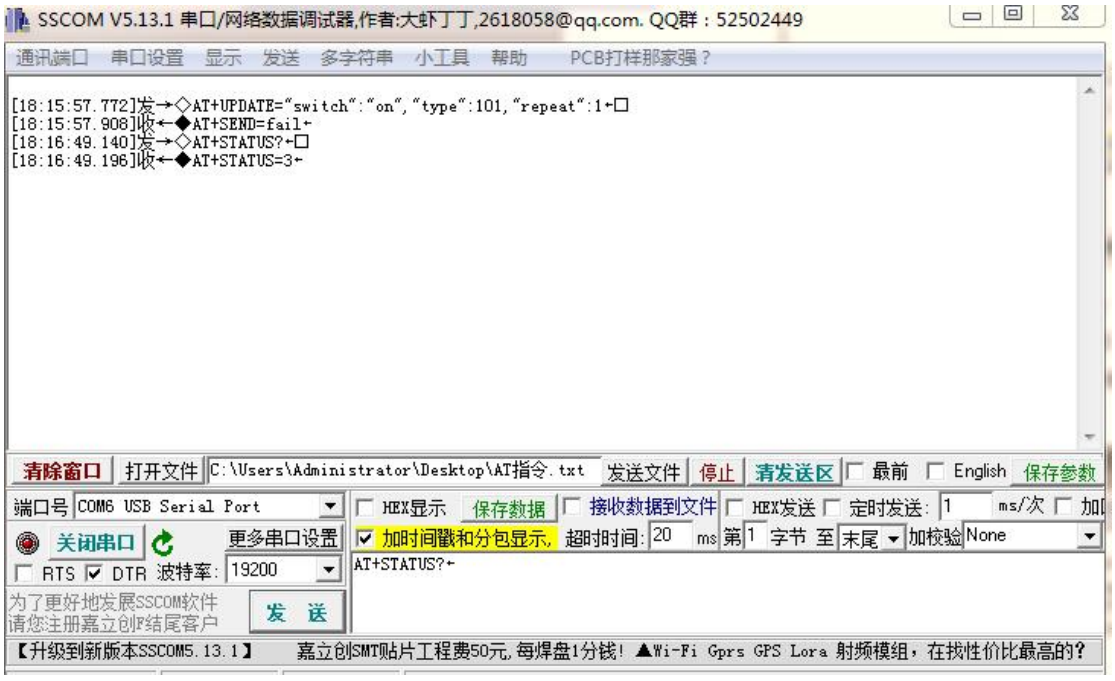
3.1.3、用串口调试工具“sscom5.13.1”调试 AT 指令示例，及出现的问题解析

注意：调试指令前，连接好 TA 模块与电脑的串口。

调试指令 1（设置类指令）：打开 WIFI 灯，在串口调试工具中发送“AT+UPDATE="switch":"on","type":101,"repeat":1”指令，如下图所示：



此时 TA 模块回复的指令为“AT+SEND=fail”，表示 TA 模块没有把打开 WIFI 灯的指令发送给服务器，出现此种情况的原因可能是 TA 模块没有连上网络或者没有被添加到 APP，此时在串口调试工具上发送“AT+STATUS?”这条指令判断 TA 模块的工作状态，获取 TA 模块的工作状态后再进行下一步判断。如下图所示：

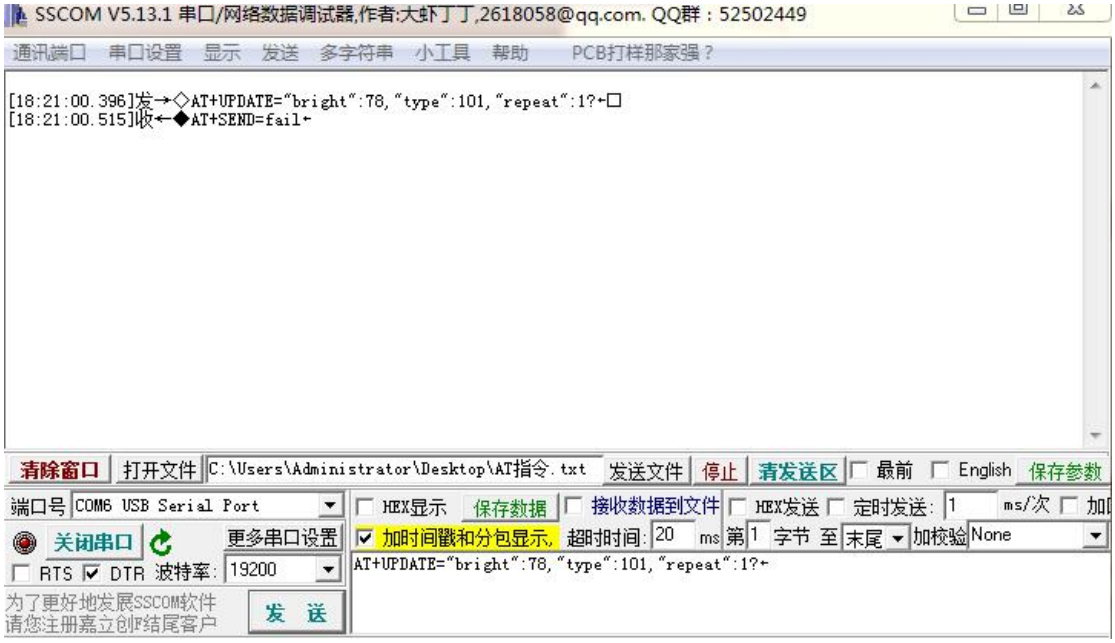


TA 模块回复的指令是“AT+STATUS=3”，“3”表示开始注册设备到服务器，即 TA 模块还没有被 APP 添加，解决方法是用 APP 添加设备，具体添加步骤见本文 3.2.1 章节。

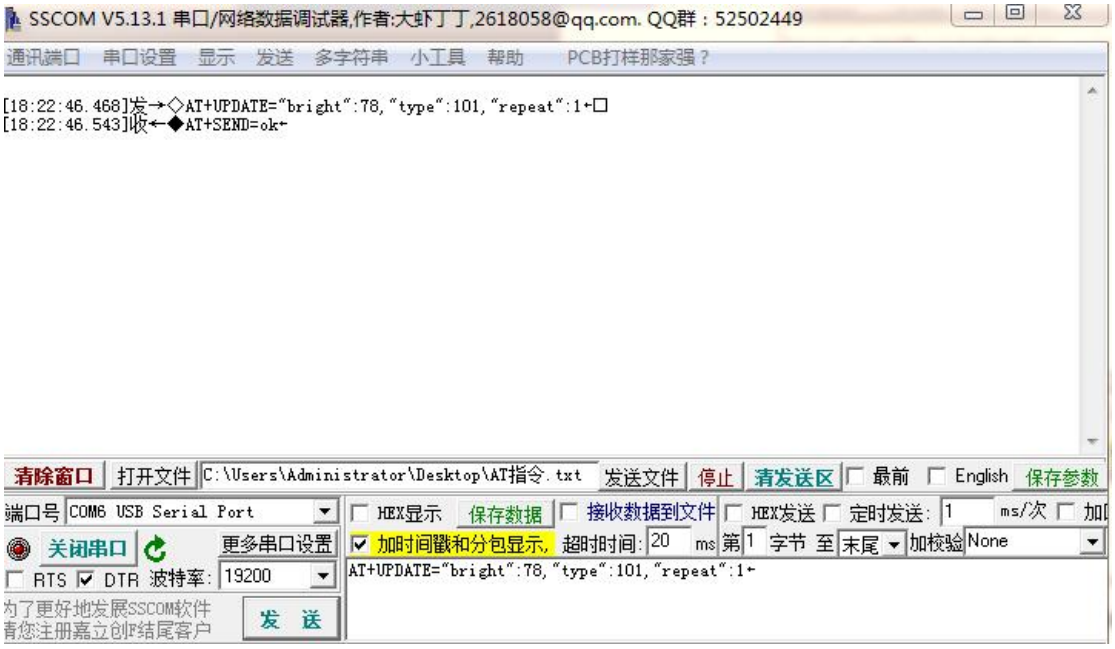
如果 TA 模块回复的指令是“AT+STATUS=5”或“AT+STATUS=9”，则表示 APP 正在添加 TA 模块，等 APP 添加成功后即可发送开灯指令；如果 TA 模块回复的指令是“AT+STATUS=1”，则表示 TA 模块没有连上路由器，可检查路由器是否断电，或路由器是否损坏；

如果 TA 模块回复的指令是“AT+STATUS=2”，则表示 TA 模块已连上路由器，但可能路由器没有连上外网；如果 TA 模块回复的指令是“AT+STATUS=4”，则表示 TA 模块已正常连上服务器，出现的问题可能是指令格式不对；如果 TA 模块回复的指令是“AT+STATUS=6”或“AT+STATUS=7”，则表示 TA 模块进入测试模式或正在烧录出厂数据，若连续几次均是如此，请联系我们；如果 TA 模块回复的指令是“AT+STATUS=8”，则表示 TA 模块正在进行固件升级，请等候 TA 固件升级成功后再发送指令；如果 TA 模块无回复，则检查 TA 模块是否正常供电，检查指令格式是否正确。

调试指令 2（设置类指令）：调节 wifi 灯亮度，在串口调试工具中发送“AT+UPDATE="bright":78,”type”:101,”repeat”:1”指令，如下图所示：



TA 模块回复的指令是“AT+SEND=fail”，按照调试指令 1 中的方法进行分析后，发现是串口调试工具指令发送的指令格式有误：指令结束符前多了一个“？”，把它去掉后重新发送，如下图所示：



可见，指令修改正确后，TA 模块回复“AT+SEND=ok”，表示 TA 模块已将指令发送给服务器。

调试指令 3（查询自定义字段）：查询 WIFI 灯的亮度等级，在串口调试工具中发送“AT+QUERY="bright"”指令，如下图所示：

TA 模块首先回复“AT+SEND=ok”，表示 TA 模块已将 MCU 端发送的查询指令发送给服务器，再回复“AT+RESULT="bright":78”，表示当前 wifi 灯的亮度等级为 78（调试指令 2 中设置过）。

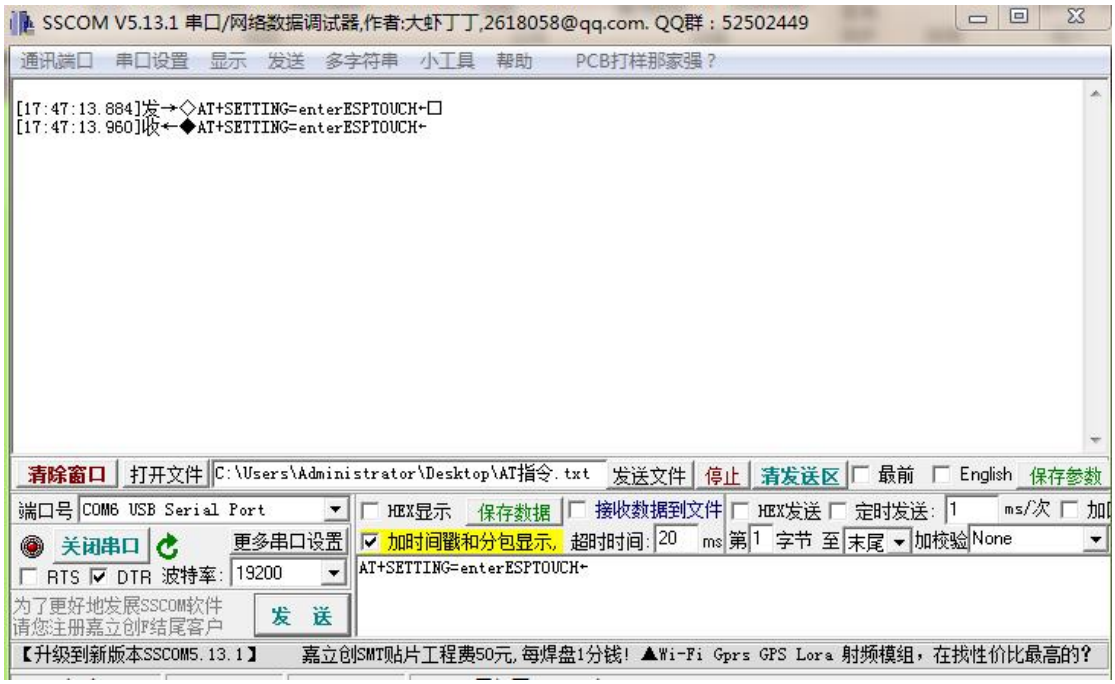
调试指令 4（查询指定字段）：查询当前 GMT 时间，在串口调试工具中发送“AT+GMT？”指令，如下图所示：

TA 模块回复的指令为“AT+GMT=2018-03-13 09:37:56”，表示当前的 GMT 时间为 2018 年 3 月 13 日 9 点 37 分 56 秒，此时北京时间为 2018 年 3 月 13 日 17 点 37 分 56 秒，GMT 时间加 8 小时就是北京时间。

调试指令 5（查询指定字段）：查询 TA 模块 wifi 信号强度，在串口调试工具中发送“AT+RSSI？”指令，如下图所示：

TA 模块回复的指令为“AT+RSSI=-56”，表示当前 TA 模块的 wifi 信号强度为-56dbm，此数值越接近 0，表示信号强度越好。

调试指令 6（设置类指令）：设置 TA 模块进入 touch 配置模式，在串口调试工具中发送“AT+SETTING=enterESPTOUCH”指令，如下图所示：



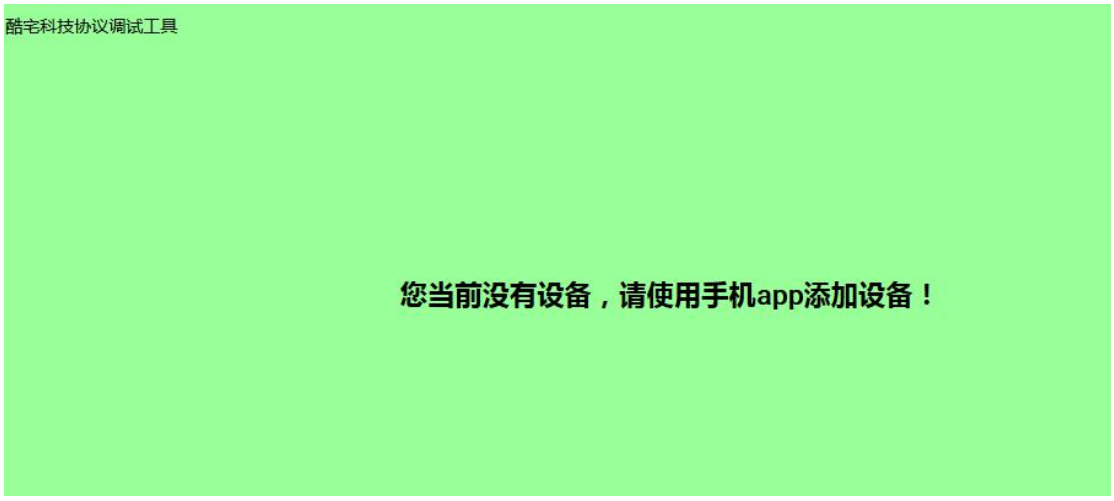
TA 模块下发的指令为“AT+SETTING=enterESPTOUCH”，表示 TA 模块已进入 touch 配置模式，可以被 APP 添加。

3.2、web 调试工具

3.2.1、登陆 web 调试工具，添加设备

登陆 web 调试工具(http://54.222.241.185/testing/index.html#/user/login)后

可看到如下界面：



此时设备没有被 APP 添加，web 调试窗口看不到任何设备，连接好 TA 模块与电脑的串口后，通过串口调试工具向 TA 模块发送“AT+SETTING=enterESPTOUCH↵”这条指令，设置 TA 模块进入 touch 配置模式，登陆 APP，可看到如下界面：

点击界面下端中间的“+”添加设备，进入如下界面：

点击“下一步”，进入如下界面：

输入正确 WIFI 密码后点击“下一步”，进入如下界面，等待设备添加成功：

设备添加成功后会出现如下界面：

设备名称可自行修改，修改完成后点击完成添加。示例中将设备名称改为“Wifi 灯”，添加完成后的界面如下所示：

至此，已完成设备添加，接下来重新登录 web 调试工具，可看到如下界面：

当设备（TA 模块）被 APP 添加成功后，在 web 调试端可清楚地看到自己给设备的命名（上图左上角的 name）,名称下方是设备（示例为 TA 模块，如果是我司其它产品，则为其它模块）出厂时的唯一编号（deviceid），编号下方是 APP 界面类型（ui），设备不同，APP 界面也不同，但同一系列产品，操作方法一样，右上方显示设备的在线情况，右下方是 web 端的命令输入区。

3.2.2、用 web 调试工具，调试设备

在上图中的白色框内，以 json 字符串形式输入控制字段，然后点击输入框下的发送按钮即可将指令发送至 MCU。

Web 调试工具在不修改第三行中的“action”字符串时，默认是发送“UPDATE”类的设置指令，故发送“UPDATE”类的指令时只需在发送框中输入自定义的控制字段即可。如打开 WIFI 灯，可输入下图所示的指令：

如果 MCU 在 5 秒内没有正确回复该指令，web 端会提示出现超时错误（上图中的倒数第二段）。如将灯光调节至 60 亮度等级，可输入下图所示指令：

串口调试工具收到的指令如下图所示：

Web 调试工具端下发的指令跟 APP 端操作下发的指令相同，MCU 端只要及时回复该指令即可。

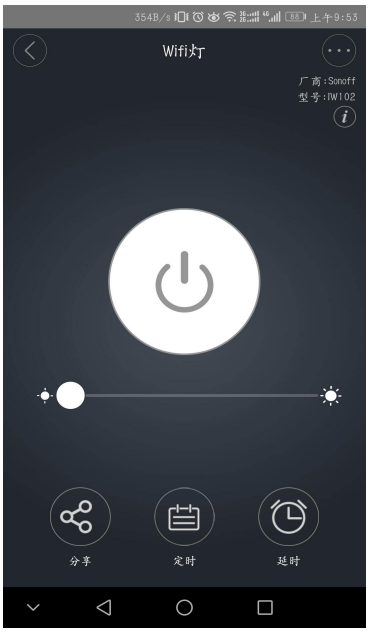
如果 TA 模块的 WIFI 信号发生一定范围的变化，TA 模块会主动上传当前的的 WIFI 信号强度，如下图所示：

上图中“RSSI”后跟的是 WIFI 信号强度，“staMac”后跟的是 TA 模块的 Mac 地址，“fwVersion”后跟的是 TA 模块的固件版本号，“from”后跟的当前信息的发送方，“device”表示发送方为设备（TA 模块），如果是“app”则表示发送方为 APP。

3.2.3、直接在 APP 端操作，配合串口调试工具调试设备

在 3.2.2 节中可知，APP 刚添加完 wifi 灯时，APP 界面如下所示：

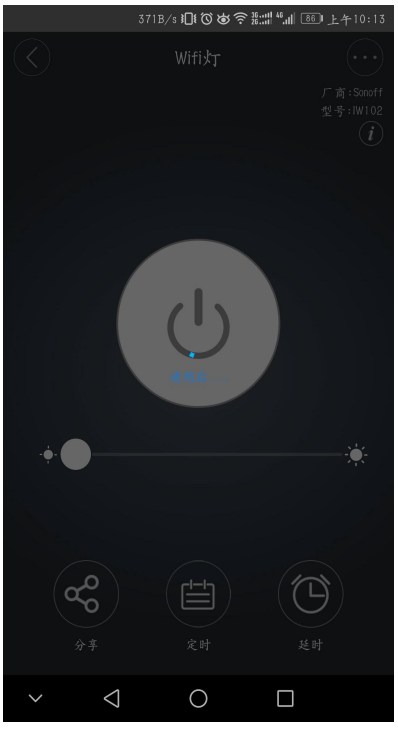
此时点击灰色区域中右下角的“>”图标可进入如下界面：



上图中中间的白色大按钮为开关按钮，开关按钮正下方为亮度设置的亮度条，亮度条上的大白点所在位置即代表当前灯的亮度等级，当大白点在亮度条在最左边时，亮度等级最低，在最右边时亮度等级最高。亮度条下有设备的分享按钮，点击它可将设备的控制权分享给其他人，还有定时按钮，点击此按钮后可设置指定时间开/关灯，还有延时按钮，点击此按钮可实现延时开/关灯，上图中所示为 Wifi 灯刚被添加完成时的状态(亮度最低，灯为关闭状态)，每种设备被添加完成时状态由设备被添加时的状态决定，并非每次都一样（前提是每当设备重新连上服务器时，MCU 要及时上传设备的最新状态）。

连接好电脑和 TA 模块的串口，点击上图中的开关按钮，打开 wifi 灯，可看到串口调试工具收到如下图所示指令：

此时 APP 的状态如下图所示：

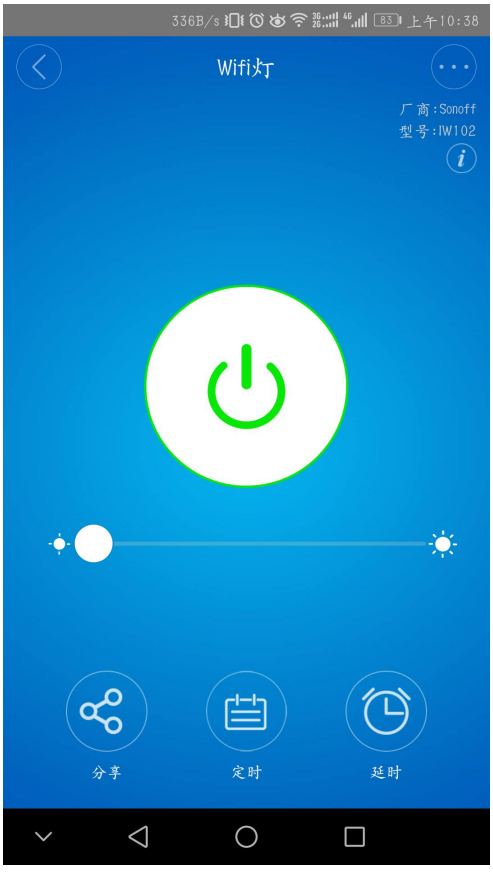


上此时 APP 界面整体变暗，处于不可操作状态，界面中心出现“请稍后”的提示信息，5 秒钟后

恢复为没有点击开关按钮的状态。出现此种情况的原因是 MCU 没有及时回复 APP（通过 TA 模块）下发的指令，导致 APP 端操作不成功。

此时可用串口调试工具模拟 MCU 给 TA 模块发指令，在调试工具的发送窗口输入：AT+RESULT="sequence":←（此格式的指令用于回复 UPDATE 类指令），等在 APP 端点击开关按钮后，立即将串口调试工具接收到的指令中的序列号复制到发送区的结束符之前，点击发送（此处要求手速快，完成整个操作时间不能超过 5 秒，否则无效，即 MCU 在回复 TA 模块的指令时，越快回复越好），如下图所示：

此时 APP 界面如下所示：



此时 APP 界面整体色调变亮，开关按键边缘边缘绿色，表示 APP 端开灯操作成功，

产品不一样，APP 界面显示的开关状态不一样，用户需保证自己正在开发的产品与 APP 匹配。

下面是一个定时器任务示例，具体操作如下：

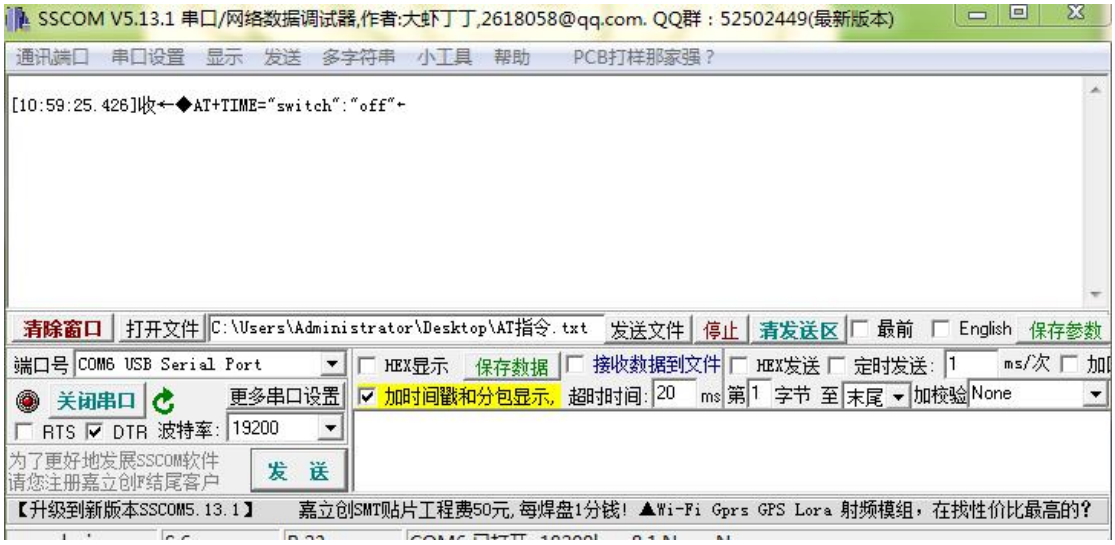
在上图 APP 界面的下方有一个“定时”按钮，点击它进入如下界面：

点击界面下方的“添加定时器按钮”，进入如下界面：

因为 Wifi 灯当前是打开状态，现在设置在 10 点 59 分关闭，设置结果如下：

然后点击”完成”即可。

定时时间到了，可在串口调试工具上看到 TA 模块在定时器指定时间到时给 MCU 发送的指令，如下图所示：



MCU 接收到此指令时无需回复，但要执行此定时器任务，任务执行完后，将最新状态上传，保证 MCU 端的状态与 APP 端状态同步。

4. 单片机代码示例

本示例主要描述 MCU 端怎样接收和解析 TA 模块从串口发来的数据。

4.1、整体思路

由于 TA 模块与 MCU 是通过串口进行的通信，为保证通信质量，MCU 应将串口设置为中断，且定为较高优先级；若 MCU 内存资源十分受限，指令缓冲区可设置为环形缓冲区，且留给环形缓冲区的大小至少是最长的 AT 指令长度的 2 倍以上（本示例用的是环形缓冲区）；对 AT 指令进行分段分类解析。以 wifi 灯为例，程序解析时 AT 指令分类如下：

指令大类	细分类 1	细分类 2
START	无	无
UPDATE	switch	on
		off
	bright	无
TIME	switch	on
		off
SETTING	enterESPTOUCH	无
	enterAP	
	exitESPTOUCH	
	exitAP	
SEND	ok	无
	fail	
STATUS	无	无

4.2、具体示例代码

```
#define 0X1B                ESC
#define AT_UPDATE           ("AT+UPDATE")
#define AT_SEND             ("AT+SEND")
#define AT_NOTIFY           ("AT+NOTIFY")
#define AT_STATUS           ("AT+STATUS")
#define AT_SETTING          ("AT+SETTING")
#define AT_TIME             ("AT+TIME")
#define AT_START            ("AT+START")

#define BRIGHT              ("\"bright\":")
#define SWITCH              ("\"switch\":")
#define FAIL                ("fail")
#define OK                  ("ok")
#define ON                  ("\"on\"")
```

```
#define OFF                     ("\"off\"")
#define SEQUENCE                ("=\"sequence\":")
#define ENTER_TOUCH             ("enterESPTOUCH")
#define ENTER_AP                ("enterAP")
#define EXIT_TOUCH              ("exitESPTOUCH")
#define EXIT_AP                 ("exitAP")

#define ON_OFF_TYPE             ("101")
#define BRIGHT_TYPE             ("102")

void receivedCommandControl (void)
{
    unsigned char i = 3;
    /* 设置解析单条指令的缓冲区，缓冲区大小为至少为最长 AT 指令的长度加 1 */
    unsigned char command_analyse_buf[MAX_SIZE+1];   /*< MAX_SIZE 为最长 AT 指令的长度 */
    unsigned char command_family_buf[10];/*< 设置指令大类（如 UPDATE、START）的缓冲区 */
    /* 获取可以处理的单条完整指令 */
    if (1 == getCommandToAnalyse(command_analyse_buf))
    {
        /* 获取等号前的指令大类 */
        while ((command_analyse_buf[i] != '=')
               && (i < 10)
               && (command_analyse_buf[i] != ESC))   /*< ”AT+“后跟的指令大类最大长度为 9 */
        {
            /* 每条指令均以”AT+“三个字符开始，”+“到”=“之间的字符串为指令大类 */
            command_family_buf[i-3] = command_analyse_buf[i];
            i++;
        }

        command_family_buf[i-3] = '\0';

        if (strstr(AT_START,command_family_buf))
        {
            analyseStartCommand(); /*< 处理 START 类指令 */
            return;
        }
        else if (strstr(AT_UPDATE,command_family_buf))
        {
            analyseUpdateCommand(command_analyse_buf); /*< 处理 UPDATE 类指令 */
            return;
        }
        else if (strstr(AT_TIME,command_family_buf))
        {
```

```
            analyseTimeCommand(command_analyse_buf); /*< 处理 TIME 类指令 */
            return;
        }
        else if (strstr(AT_SETTING,command_family_buf))
        {
            analyseSettingCommand(command_analyse_buf); /*< 处理 SETTING 类指令 */
            return;
        }
        else if (strstr(AT_SEND,command_family_buf))
        {
            analyseSendCommand(command_analyse_buf);   /*< 处理 SEND 类指令 */
            return;
        }
        else if (strstr(AT_STATUS,command_family_buf))
        {
            analyseStatusCommand(command_analyse_buf); /*< 处理 STATUS 类指令 */
            return;
        }
    }
}
```

如上所示，函数 receivedCommandControl 为命令解析函数，对命令的处理步骤为：通过函数 getCommandToAnalyse 获取单条可解析的指令——>解析出该指令具体分属哪类指令——>按指令类别对指令进行针对性解析。下文将对 receivedCommandControl 调用的函数进行详细解析：

符串查找函数（strstr）：

```
static unsigned char* strstr(char *src,char *sub)
{
    const char *bp=src;
    const char *sp=sub;
    if(src==NULL||NULL==sub)
    {
        return NULL;
    }
    while(*src)
    {
        bp=src;
        sp=sub;
        do{
            if(!*sp)
                return src;
            }while(*bp++==*sp++);
        src+=1;
    }
```

```
    return NULL;
}
```

获取单条指令函数：

```
    /*
     * 函数名称：getCommandToAnalyse
     * 函数作用：获取可以执行的完整格式指令
     * 参      数：analyse_buf -> 取出可处理的完整指令
     * 返 回 值：1：已接收到完整指令
     *           0：没有接收到完整指令
     */
    static unsigned char getCommandToAnalyse(unsigned char* analyse_buf)
    {
        unsigned char read_byte = 0;
        static unsigned char frame_header[4] = {0}; /*< 帧头字符缓冲区 */
        static unsigned char frame_index = 0;

        while(commandBufReadAvailable() /*< 函数 commandBufReadAvailable 用于判断环形缓冲区中是否有数据 */
              && frame_index < 3)
        {
            read_byte = commandRead(); /*< 读取环形缓冲区的一个字节 */

            /* 获取帧头“AT+” */
            if(0 == frame_index)
            {
                if('A' == read_byte)
                {
                    frame_header[frame_index] = read_byte;
                    frame_index++;
                }
            }
            else if (1 == frame_index)
            {
                if('T' == read_byte)
                {
                    frame_header[frame_index] = read_byte;
                    frame_index++;
                }
                else
                {
                    frame_index = 0;
                }
            }
            else if (2 == frame_index)
```

```
        {
            if('+' == read_byte)
            {
                frame_header[frame_index] = read_byte;
                frame_index++;
                break;
            }
            else
            {
                frame_index = 0;
            }
        }
}

/* 已取到帧头 */
if (3 == frame_index)
{
    for (frame_index = 0;frame_index < 3;frame_index++)
    {
        analyse_buf[frame_index] = frame_header[frame_index];
    }
}

if (1 == ESCCheak())   /* 函数 ESCCheak 用于检查缓冲区中是否有结尾符 ESC */
{
    /* 取出整帧数据 */
    while(commandBufReadAvailable())
    {
        analyse_buf[frame_index] = commandRead();

        if (frame_index > MAX_SIZE
            && analyse_buf[frame_index] != ESC) /*< 数据超长 */
        {
            frame_index = 0;
            analyse_buf = NULL;
            break;
        }

        if(analyse_buf[frame_index] == ESC)
        {
            /* 取出整帧数据 ESC 后加'\0',方便后续字符串处理 */
            analyse_buf[frame_index+1] = '\0';
            frame_index = 0;   /*< 清楚帧头缓冲区 */
```

```
                    return 1;
                }
                frame_index++;
            }
        }
        return 0;
    }
```

分类指令解析函数 1（START 类指令解析）：

```
static void analyseStartCommand(void)
{
    /*接收到 AT+START 指令，表示 TA 模块已连上服务器，此处可自行处理任务，例如上传状态信息等*/
    return;
}
```

分类指令解析函数 2（UPDATE 类指令解析）：

TA 模块传给 MCU 的 UPDATE 类指令示例：

```
/* update类完整指令 */
/* AT+UPDATE="sequence":"1516870911281","switch":"on"ESC */
/* AT+UPDATE="sequence":"1516870911281","switch":"off"ESC */
/* AT+UPDATE="sequence":"1516870911281","bright":50ESC */
/* AT+UPDATE="sequence":"1516870911281","bright":100ESC */
```

完整代码：

```
/*
 *函数名称： analyseUpdateCommand
 *函数作用：处理 UPDATE 类指令
 *输入参数：command，待处理的完整指令
 *返回值：无(此处可根据实际需要设置相应的返回值)
 */
static void analyseUpdateCommand(unsigned char* command)
{
    unsigned char command_family_buf[14]; /*< 指令解析缓冲区 */
    unsigned char bright_temp = 0;
    unsigned char command_lenth = 0;
    unsigned char i = 9; /*< update 类指令中的 bright 和 switch 关键字从第 39 字节开始,等号在第 10 个字节 */
    command_lenth = strlen(command);
    if ((command_lenth < 49) || (command_lenth > 52))/*< update 类指令最短为 49 字节,最长为 52 字节 */
    {
        return;
    }
    while (i < 21)
    {
        command_family_buf[i-9] = command[i];
        i++;
    }
```

```
if (':' != command[20])     /*< 冒号在第 21 字节 */
{
    return;
}
command_family_buf[i-9] = '\0';
if (strstr(SEQUENCE,command_family_buf))
{
    if ('\"' != command[21]) /*< 序列号前的引号 */
    {
        return;
    }
    i = 22; /*< 序列号正式开始 */
    while (i < 35)
    {
        command_family_buf[i-22] = command[i];
        i++;
    }
    command_family_buf[i-22] = '\0';
    for (i = 0;i<13;i++)
    {
        if (command_family_buf[i] < '0' || command_family_buf[i] > '9')
        {
            return; /*< 序列号必须是纯数字字符 */
        }
    }
    if ('\"' != command[35]) /*< 第 36 字节是序列号后的引号 */
    {
        return;
    }
    if (',' != command[36]) /*< 第 37 字节是序列号后的逗号 */
    {
        return;
    }
    for (i=37;i<46;i++) /*< switch 后的冒号在第 46 字节 */
    {
        command_family_buf[i-37] = command[i];
    }
    command_family_buf[i-37] = '\0'; /*< 此时 i = 46，开关指令及亮度均从指令的第 46 字节开始 */
    if (strstr(BRIGHT,command_family_buf))  /*< bright 类 */
    {
        while (i < 50) /* bright 类最长为 50 字节，包含 ESC */
        {
            command_family_buf[i-46] = command[i];
```

```
            if (ESC == command[i])
            {
                break;
            }
            i++;
        }
        if(ESC != command_family_buf[i-46])/*< 从上述循环中出来，最后一个字节必须是 ESC */
        {
            return;
        }
        command_family_buf[i] = '\0';
        while ('\0' != command_family_buf[i])
        {
            if (command_family_buf[i] < '0' || command_family_buf[i] > '9')
            {
                return; /*< 亮度必须是纯数字字符 */
            }
            i++;
        }
        bright_temp = atoi(command_family_buf);
        lightLevelSet(bright_temp); /*< 设置灯光亮度 */
        commandAnswer(command);        /*< 回复指令 */
    }
    else if (strstr(SWITCH,command_family_buf)) /*< switch 类 */
    {
        while (ESC != command[i]) /*如果数据超长不会进到此处*/
        {
            command_family_buf[i-46] = command[i];
            i++;
        }
        command_family_buf[i-46] = '\0';
        if (strstr(ON,command_family_buf))
        {
            lightSet(ON);     /*< 先执行开灯指令 */
            commandAnswer(command);
        }
        else if(strstr(OFF,command_family_buf))
        {
            lightSet(OFF);     /*< 先执行关灯指令 */
            commandAnswer(command); /*< 回复指令 */
        }
    }
}
```

}

分类指令解析函数 3（TIME 类指令解析）：

```
/*
 *函数名称： analyseTimeCommand
 *函数作用：处理 TIME 类指令
 *输入参数：command，待处理的完整指令
 *返回值：无(此处可根据实际需要设置相应的返回值)
 */
static void analyseTimeCommand(unsigned char* command)
{
    unsigned char command_family_buf[11];
    unsigned char command_lenth = 0;
    unsigned char i = 0;
    /* time 类完整指令 */
    /* AT+TIME="switch":"on" */
    /* AT+TIME="switch":"off" */

    command_lenth = strlen(command);
    if ((command_lenth < 22) || (command_lenth > 23)) /*< 当前 time 类的指令只有 22 字节和 23 字节 */
    {
        return;
    }

    if ('=' != command[7]) /*< 第 8 字节必须为等号 */
    {
        return;
    }
    for (i = 8;i < 17;i++)
    {
        command_family_buf[i-8] = command[i];
    }
    command_family_buf[i-8] = '\0'; /*< 此时 i=17 */

    if (strstr(SWITCH,command_family_buf))
    {
        while (ESC != command[i])
        {
            command_family_buf[i-17] = command[i];
            i++;
        }
        command_family_buf[i-17] = '\0';
        if (strstr(ON,command_family_buf))
        {
            lightSet(ON);
        }
        else if(strstr(OFF,command_family_buf))
        {
            lightSet(OFF);
        }
        else
```

```
        {
            /* 无需处理 */
        }
    }
    else
    {
        /* 无需处理 */
    }
}
```

分类指令解析函数 4（SETTING 类指令解析）：

```
/*
 *函数名称： analyseSettingCommand
 *函数作用：处理 SETTING 类指令
 *输入参数：command，待处理的完整指令
 *返回值：无(此处可根据实际需要设置相应的返回值)
 */
static void analyseSettingCommand(unsigned char* command)
{
    unsigned char command_family_buf[15];
    unsigned char command_lenth = 0;
    unsigned char i = 11; /*< setting 类指令等号后的指令从第 11 字节开始 */

    /* AT+SETTING=enterESPTOUCH */
    /* AT+SETTING=enterAP */
    /* AT+SETTING=exitESPTOUCH */
    /* AT+SETTING=exitAP */

    command_lenth = strlen(command);

    if ((command_lenth < 17) || (command_lenth > 25)) /*< setting 类指令最短为 18 字节，最长为 25 字节 */
    {
        return;
    }
    if ('=' != command[10])
    {
        return;/*< 第 11 字节必须为等号 */
    }
    while (ESC != command[i])
    {
        command_family_buf[i-11] = command[i];
        i++;
    }
    command_family_buf[i-11] = '\0';
    if (strstr(ENTER_TOUCH,command_family_buf))       /**< AT+SETTING=enterESPTOUCH*/
    {
```

```
        /*TA 模块已进入 touch 配置模式，等待 APP 添加，此时 MCU 不可上传信息*/
    }
    else if(strstr(ENTER_AP,command_family_buf))       /**< AT+SETTING=enterAP*/
    {
        /*TA 模块已进入 AP 配置模式，等待 APP 添加，此时 MCU 不可上传信息*/
    }
    else if(strstr(EXIT_AP,command_family_buf)
            || strstr(EXIT_TOUCH,command_family_buf))       /**< 退出配置，会有短暂时间离线 */
    {
        /*TA 模块已退出配置模式，请等待 TA 模块上线*/
    }
    else
    {
        /*< 传入指令有误，自行处理 */
    }
}
```

分类指令解析函数 5（SEND 类指令解析）：

```
/*
 *函数名称： analyseSendCommand
 *函数作用：处理 SEND 类指令
 *输入参数：command，待处理的完整指令
 *返回值：无(此处可根据实际需要设置相应的返回值)
 */
static void analyseSendCommand(unsigned char* command)
{
    unsigned char command_family_buf[37];
    unsigned char command_lenth = 0;
    unsigned char i = 0;

    /* AT+SEND=ok */
    /* AT+SEND=fail */
    command_lenth = strlen(command);

    if ((command_lenth != 11) && (command_lenth != 13))/*< 返回的 seng 指令长度只可能为 11 或 13 字节 */
    {
        return;
    }
    if ('=' != command[7])
    {
        return; /*< 第 8 字节必须为等号*/
    }
    i= 8;
    while (ESC != command[i])
```

```
    {
        command_family_buf[i-8] = command[i];
        i++;
    }
    command_family_buf[i-8] = '\0';
    if (strstr(FAIL,command_family_buf))
    {
        /* MCU 指令上传失败，自行处理 */
        commandSendFailControl();
    }
    else if(strstr(OK,command_family_buf))
    {
        /* MCU 指令上传成功，自行处理 */
        commandSendOkControl();
    }
    else
    {
        /** 无需处理 */
    }
}
```

分类指令解析函数 6（STATUS 类指令解析）：

```
/*
 *函数名称： analyseStatusCommand
 *函数作用：处理 STATUS 类指令
 *输入参数：command，待处理的完整指令
 *返回值：无(此处可根据实际需要设置相应的返回值)
 */
static void analyseStatusCommand(unsigned char* command)
{
    unsigned char command_lenth = 0;

    /* AT+STATUS=5 */
    command_lenth = strlen(command);

    if (command_lenth != 12) /*< status 指令只有 12 字节 */
    {
        return;
    }

    if ('=' != command[9])
    {
        return; /*< 第 10 个字节必须为等号 */
    }
```

```
    if (ESC != command[11])
    {
        return; /*< 第 12 个字节必须为 ESC */
    }
    if (command[10] < '0'
        || command[10] > '9')
    {
        return; /*< 第 11 个字节必须为出数字字符 */
    }

    /* TA 模块当前工作状态 = command[10] - 48 ,
       此处自行处理,如标记 TA 模块状态 TA_status = command[10] - 48*/
}
```

5. TA 模块使用注意事项

TA 模块使用时应注意以下问题：

1、设备连上网络后请及时上传当前设备最新状态，保证 APP 端与设备端同步，包括开机连网和中途断网后重新连接网络。

2、模块连上服务器后主动发送给 MCU ” AT+START “这条指令，如果没收到这条指令，可通过” AT+STATUS? “这条指令来查询模块工作状态。

3、” AT+TIME “类的指令是 APP 端定时器任务的时间到了模块发送给 MCU 的指令，MCU 无需回复，只需执行 APP 端指定的动作即可。

如在 APP 端的定时器中定时在今晚十点打开 WIFI 灯，在今晚十点的时候模块发送指令：AT+TIME=” switch” :” on” ，MCU 收到指令开灯，然后上传灯的最新状态即可。

4、MCU 判断模块与服务器之间的连接可用” AT+STATUS? “指令进行查询，模块判断模块与 MCU 之间的连接状态是通过” AT+WATCHDOG “这条指令，模块如果 10 秒内没收到” AT+WATCHDOG “会认为 MCU 出现故障，此时会在模块的 GPIO14 脚输出 100 毫秒的低电平。

5、如果 TA 模块断线，MCU 端的操作可照常处理，待模块重新上线后，更新 MCU 端最新状态到服务器即可。

免责申明和版权公告

本文中的信息，包括供参考的 URL 地址，如有变更，恕不另行通知。文档“按现状”提供，不负任何担保责任，包括对适销性、适用于特定用途或非侵权性的任何担保，和任何提案、规格或样品在他处提到的任何担保。本文档不负任何责任，包括使用本文档内信息产生的侵犯任何专利权行为的责任。本文档在此未以禁止反言或其他方式授予任何知识产权使用许可，不管是明示许可还是暗示许可。Wi-Fi 联盟成员标志归 Wi-Fi 联盟所有。ESP 商标为乐鑫公司注册商标文中提到的所有商标名称、商标和注册商标属其各自所有者的财产，特此声明。

2018/3/14