云端升级实现方案

Status	
Current version	V1.2
Author	Yu Fei
Completion Date	2014.7.7
Reviewer	Wu Jiangang
Completion Date	2014.7.7

[ ] CONFIDENTIAL

[ ] INTERNAL

[√] PUBLIC

版本信息

日期	版本	撰写人	审核人	修改说明
2014.6.17	1.0	巫建刚		初稿
2014.7.1	1.1	巫建刚		添加升级方式
2014.7.7	1.2	喻菲		升级的详细说明

目录

1. 前言

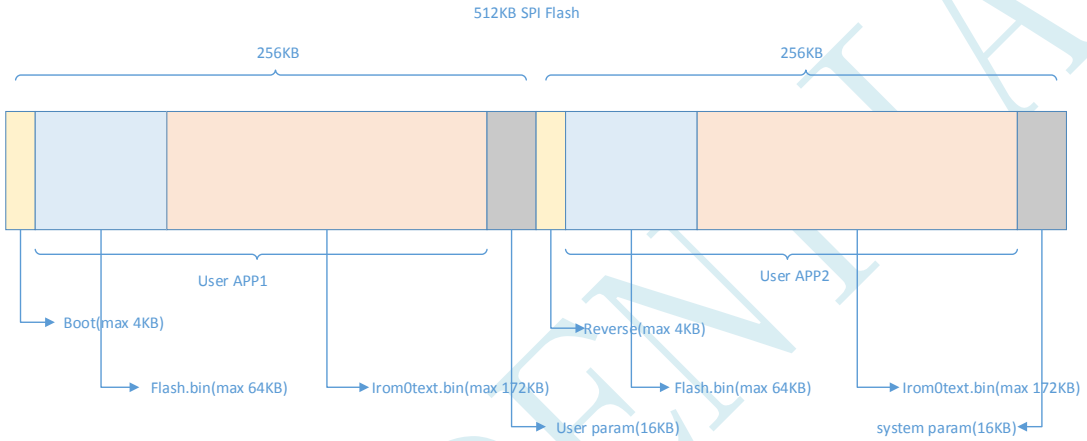
本文主要介绍 ESP8266 基于服务器实现云端升级软件的方法，当中涉及到 ESP8266 底层 firmware 在 SPI Flash 中的存放，ESP8266、Server、APP之间实现 upgrade 所需要的通信过程。

主要供 firmware 开发以及 server 开发参考。

2. 底层框架

为了实现云端升级功能，重新定义了代码功能及存放空间。Espressif 提供 boot.bin 用于初始 boot 选择；将原来的 Flash.bin 和 irom0text.bin 合并为 user1.bin（或者 user2.bin），实现互相备份升级。

512KB 的 SPI Flash 空间分布如下图所示：



1、 Boot，存放 boot.bin；

该 boot 为二级 boot，区分于 rom 中的 boot，存放在 SPI Flash 的开始 0~4KB 空间内；

2、 User APP1，存放 user1.bin（即 Flash.bin 和 irom0text.bin）；

用户程序区，如 Sensor、Plug 等应用，存放在 SPI Flash 的 4KB~240KB 空间；

3、 User param，用户参数存放区；

存放在 SPI Flash 的 240KB~256KB 空间内，4 个 sector；

4、 Reverse，保留空间；

存放在 SPI Flash 的 256KB~260KB 空间，为了使 User APP1 和 User APP2 的地址空间偏移一致；

5、 User App2，，存放 user2.bin（即 Flash.bin 和 irom0text.bin）；

用户程序区，如 Sensor、Plug 等应用，存放在 SPI Flash 的 260KB~496KB 空间；

6、 System param，系统参数存放区；

存放在 SPI Flash 的 496KB~512KB 空间内，4 个 sector。

Boot 程序的作用是根据 system param 中的 flag，判断是运行 User APP1 还是 User APP2。

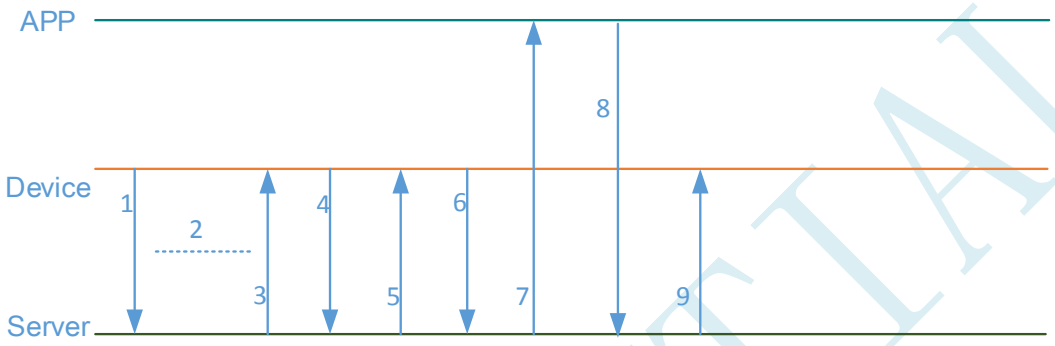
我们可以认为 User APP1 和 User APP2 是应用的两个备份，在实际烧录中，就是 user1.bin 和 user2.bin，通过 Boot 在两个之间进行切换。

例如，初始烧入 boot.bin 和版本 v1.0 的 user1.bin，system param 中的 flag 标志为使用 user1；当服务器上的软件上传更新 v1.1 的 user1.bin 和 user2.bin 时，服务器推送通知，设备根据当前使用的是 user1，会从服务器下载 v1.1 的 user2 到 flash 260KB 之后的空间，下载完成后，推送消息给用户，用户控制设备是否重启使用新版本软件（即 v1.1 的 user2），如果用户选择重启，则修改 system param 中的 flag 标志为使用 user2，设备重启，使用 v1.1 的 user2 软件。再下次升级，则下载 user1 到 flash 的前半段，覆盖之前 v1.0 的 user1 软件。

对于同一份代码，由于 icache 在 SPI Flash 上的映射关系，需要通过修改链接文件，在编译的时候生成两套 firmware（user1.bin 和 user2.bin），存放在服务器上，由设备收到升级请求后，根据设备目前使用软件情况，判断应该下载 user1 还是 user2。

3.升级过程

将更新后的 firmware 上传到服务器后，点击“升级”，服务器回推送消息升级的条件为服务器上 firmware 的版本高于设备端的版本，升级过程如下图：



1. 设备激活，在激活过程中，上报设备的版本信息给服务器，服务器将对应设备的版本信息存于数据库。
2. 正常的使用过程。
3. 服务器端的 firmware 更新，推送升级消息到设备。
4. 设备根据 device key 以及升级路径请求对应 firmware。
5. 下载升级对应的 firmware 到 SPI Flash。
6. 设备发送升级完成消息到服务器。
7. 服务器将升级完成消息推送给 APP。
8. APP 发反向控制消息，使设备重启进入新版本，消息先发至 Server。
9. Server 将重启消息转发至 Device，Device 重启进入新版本。

以上所有数据流均采用 SSL 加密，从第 4 步开始，需要结合 device key 才能获取到 server 的数据，从而保证 upagrade 过程的安全性。

注：

由于温湿度计会进入休眠，升级流程将有所不同。目前仅实现了开关一类支持反向控制的设备的云端升级，对于温湿度计这类不支持反向控制的设备的云端升级，会在后续实现。

4. 使用指南

User1 和 user2 实际就是同一份软件，放在了 flash 的不同位置，然后交替使用；以达到使用其中一份软件时，在后台下载另一份软件作为更新升级，下载完成后，系统重启，使用更新升级的软件。

4.1. 如何生成 user1.bin 和 user2.bin

对于同一份代码，由于 icache 在 SPI Flash 上的映射关系，需要通过修改链接文件后，分别编译，生成两套 firmware（user1.bin 和 user2.bin）。

生成方法如下：

1) 打开编译器，先编译 user1.bin，执行指令 make APP=1
2) 执行 gen_misc_plus.bat user1，在路径“\esp_iot_sdk\bin\upgrade”下生成 user1.bin；
3) 执行 make clean，清除之前的编译信息；
4) 再编译 user2.bin，执行指令 make APP=2
5) 执行命令 gen_misc_plus.bat user2，在路径“\esp_iot_sdk\bin\upgrade”下生成 user2.bin

注，

上传服务器时，需要将上述 user1.bin 和 user2.bin 两个文件都上传至服务器，云端升级时，由设备根据自身的运行情况选择其一下载。

如无需云端升级功能，则采用原本的编译及烧录方法即可，esp_iot_sdk_v0.8 兼容之前的编译和烧录方式。之前版本的编译及烧录，参见文档“Espressif IoT SDK 使用手册”。

4.2. 初次烧录

初次使用时，需先将支持云端升级功能的 boot.bin 和 user1.bin，通过烧录工具“XTCOM_UTIL”烧录到 flash 中，之后就可以通过云端进行软件升级。烧录方

法参见文档“Espressif IoT SDK 使用手册”。具体如下：

boot.bin ： 由 Espressif 提供，烧录到地址 0x0000；

user1.bin ： 参照 4.1 节编译生成，烧录到地址 0x1000；

master-device-key.bin ： 向 Espressif 服务器申请，烧录到地址 0x3e000；

blank.bin ： 由 Espressif 提供，烧录到地址 0x7e000。

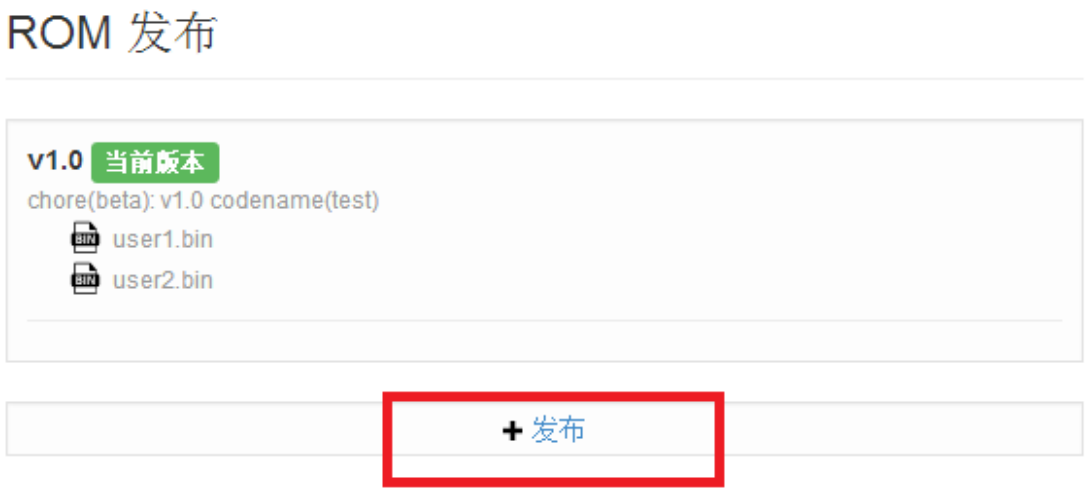
注，

初次烧录时，无需烧录 user2.bin；后续软件升级时，将更新版本的 user1.bin 和 user2.bin 两个文件都上传至服务器即可，由设备根据自身的运行情况选择其一进行云端升级。

4.3. Website 操作说明

1） 用户名、密码登录网站 http://iot.espressif.cn/#/ ，点击“产品管理”。

2） 选择需要升级的产品，进入该产品页面后，看到“ROM 发布”。例如，我们更新上述第一项产品 humiture 的软件，则点击“humiture”，进入产品 humiture 后，在页面右侧看到 ROM 发布。



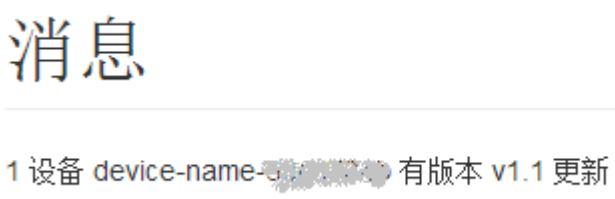
3） 点击“ROM 发布”中的“发布”，上传新版本的软件。“填写 version”-> “选择软件属性”-> “选择文件”添加新版本 user1.bin –> 点击“+”符号 -> “选择文件”添加新版本 user2.bin -> “保存”。corename 自定义发布代号，不填也可以。

注，请将 user1.bin 和 user2.bin 两个文件都上传至服务器。

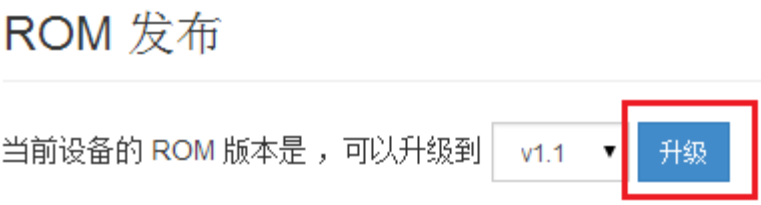
4） 保存后，会在“ROM 发布”中看到新上传的 v1.1 版本软件，此时可以选择“设置为当前版本”，进行软件版本更新，设备将收到更新消息，进行升级。

5） 在弹出的提示框中点击“确定”，更新 v1.1 成为“当前版本”。

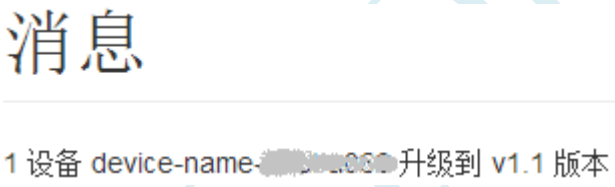
6） 该产品的用户将收到软件更新的消息提醒。



7） 该产品下的各个设备的使用者收到消息后，可进行软件升级。开发者可以选择“设备开发”-> 点击可升级的设备（如上述 device-name-XXXXXXXX）-> “ROM 发布” -> 选择版本，点击升级。



8） 设备依据当前自己正在运行的软件，从服务器下载更新后的版本。例如，设备当前正在运行 user1，则收到软件更新消息后，设备会从服务器下载新版本的 user2。下载完成后，用户收到通知。



9） 新版本软件下载到设备后，用户可以重启设备，自动运行新版本软件。在设备界面，“RPC 请求”-> action= 输入“sys_reboot”-> 点击“请求”。

10） 设备重启后，自动运行新版本软件。

4.4.Curl 指令说明

本文 4.3 节中说明的通过 website 进行的云端升级操作，同样可以通过 PC 发送 curl 指令实现。

下述 PC 侧，指用户可通过 PC 发送 curl 指令控制；设备侧，是指设备收到的数据，或者设备自发向 Server 发送的数据，无需用户操作。对于设备而言，无论控制来自 website 还是来自 PC curl 指令，设备侧的信息是一样的。

4.4.1.升级请求

✧ PC 侧

请求升级的 curl 指令为：

Linux/Cygwin curl：

```
curl -X GET -H "Content-Type:application/json" -H "Authorization: token HERE_IS_THE_OWNER_KEY" 'http://114.215.177.97/v1/device/rpc/?deliver_to_device=true&action=sys_upgrade&version=v1.1'
```

Windows curl：

```
curl -X GET -H "Content-Type:application/json" -H "Authorization: token HERE_IS_THE_OWNER_KEY" "http://114.215.177.97/v1/device/rpc/?deliver_to_device=true&action=sys_upgrade&version=v1.1"
```

注意，上述红色 v1.1 为举例，需填入实际将升级的新版本号。

✧ 设备侧

设备收到的数据格式如下：

```
{"body": {}, "nonce": 855881582, "get": {"action": "sys_upgrade", "version": "v1.1", "deliver_to_device": "true"}, "token": "HERE_IS_THE_OWNER_KEY", "meta": {"Authorization": "token HERE_IS_THE_OWNER_KEY"}, "path": "/v1/device/rpc/", "post": {}, "method": "GET", "deliver_to_device": true}
```

4.4.2. 下载完成

✧ 设备侧

设备收到升级请求后，从服务器下载新版本 bin 文件，下载完成后，设备向服务器发送如下数据，通知已将新版本软件下载成功。

```
{"path": "/v1/messages/", "method": "POST", "meta": {"Authorization": "token
 HERE_IS_THE_MASTER_DEVICE_KEY"},"get":{"action":"device_upgrade_success"},
"body":{"pre_rom_version":"v1.0","rom_version":"v1.1"}}
```

4.4.3. 重启升级

✧ PC 侧

新版本软件下载完成后，设备重启运行新软件。设备重启升级的 curl 指令为：

Linux/Cygwin curl：

```
curl -X GET -H "Content-Type:application/json" -H "Authorization: token HERE
_IS_THE_OWNER_KEY" 'http://114.215.177.97/v1/device/rpc/?deliver_to_device=
true&action=sys_reboot'
```

Windows curl：

```
curl -X GET -H "Content-Type:application/json" -H "Authorization: token HERE
_IS_THE_OWNER_KEY" "http://114.215.177.97/v1/device/rpc/?deliver_to_device=
true&action=sys_reboot"
```

✧ 设备侧

设备收到的数据格式如下：

```
{"body": {}, "nonce": 856543282, "get": {"action": "sys_reboot",
"deliver_to_device": "true"}, "token": "HERE_IS_THE_OWNER_KEY", "meta":
{"Authorization": "token HERE_IS_THE_OWNER_KEY"}, "path": "/v1/device/rpc/",
"post": {}, "method": "GET", "deliver_to_device": true}
```

设备收到 sys_reboot 后，会自动重启，运行新版本软件。

5. 软件实现

5.1. struct upgrade_server_info

云端升级服务的相关信息结构体。

```
struct upgrade_server_info{
    uint8 ip[4];            // IP 地址
    uint16 port;            // 端口号
    uint8 upgrade_flag;     // 标志（true－upgrade 完成；false－upgrade 失败）
    uint32 check_times;     // 超时计时器（ms）
    uint8 *url;             // url 指令
    upgrade_states_check_callback check_cb;     // 回调函数
    struct espconn *pespconn;                   // 指针
};
```

5.2. upgrade 函数

在 esp_iot_sdk_v0.8 中实现了云端升级软件的功能，提供以下接口：

system_upgrade_userbin_check ：检查当前运行 user1.bin 还是 user2.bin

system_upgrade_start ：升级开始。

system_upgrade_reboot ：系统重启，运行新版本软件。

用户如何调用上述接口，实现云端升级，可参考以下函数：

user_esp_platform_upgrade_begin ：开始下载升级文件。

user_esp_platform_upgrade_rsp : upgrade 回调函数

5.2.1. system_upgrade_userbin_check

功能：	检查当前正在使用的 firmware 是 user1 还是 user2。 若当前正在使用 user1.bin，则下载新版本的 user2.bin；若当前使用 user2.bin，则下载新版本的 user1.bin
函数定义：	uint8 system_upgrade_userbin_check() 参数：None。
返回值：	0x00 : UPGRADE_FW_BIN1 ，即 user1.bin 0x01 : UPGRADE_FW_BIN2 ，即 user2.bin

5.2.2. system_upgrade_start

功能：	配置参数，开始升级。
函数定义：	bool system_upgrade_start (struct upgrade_server_info *server) 参数： struct upgrade_server_info *server - server 相关的参数。
返回值：	True ：开始进行升级。 False ：已正在升级过程中，无法执行 upgrade start。

5.2.3. system_upgrade_reboot

功能：	重启系统，运行新版本软件。 用于新版软件从服务器下载成功后，由指令控制是否 reboot，用户可自行选择。
函数定义：	void system_upgrade_reboot (void) 参数：None

返回值：	None

5.2.4. user_esp_platform_upgrade_begin

功能：	准备开始下载升级文件，配置结构体 upgrade_server_info。
函数定义：	user_esp_platform_upgrade_begin (struct espconn *pespconn, struct upgrade_server_info *server, char *version) 参数： struct espconn *pespconn – connection 相关的参数。 struct upgrade_server_info *server – server 相关的参数。 char *version – 新版本信息。
返回值：	None

5.2.5. user_esp_platform_upgrade_rsp

功能：	upgrade 回调函数。 在 user_esp_platform_upgrade_begin 中注册为 server 的回调函数。当 upgrade timer（即 check_times）超时，或者 upgrade 完成，此回调函数会被调用。
函数定义：	esp_platform_upgrade_rsp(void *arg) 参数： void *arg – 即 upgrade_server_info 的指针。
返回值：	None