ESP8266 Non-OS SDK API 参考

版本 3.0.2

乐鑫信息科技

版权所有 © 2020

www.espressif.com

# 关于本⼿手册

本⽂文档提供 ESP8266\_NONOS\_SDK 的 API 说明。

发布说明

|  |  |  |
| --- | --- | --- |
| ⽇日期 | 版本 | 发布说明 |
| 2016.03 | V1.5.2 | 更更新章节 3.2、A.5 和 3.3.37 |
| 2016.04 | V1.5.3 | 新增章节 3.5.11 和 3.5.12 更更新章节 3.5.67 和 3.7.9 |
| 2016.05 | V1.5.4 | 新增章节 3.3.8、3.3.46、3.3.47、3.3.48 和 3.7.8 更更新章节 3.7 |
| 2016.07 | V2.0.0 | 更更新章节 3.8.6 和 3.5.65 新增章节 3.9、3.14、3.3.48、3.5.72 和 3.5.73 |
| 2016.11 | V2.0.1 | 将章节 3.5.30 中的函数定义 wifi\_station\_get\_hostname 改为 wifi\_station\_set\_hostname |
| 2017.01 | V2.0.2 | 更更新第 2 章 |
| 2017.05 | V2.1.0 | 新增章节 3.3.49，3.3.50，4.3.6 和 8.2.4 |
| 2017.05 | V2.1.1 | 更更新第 2 章 |
| 2017.06 | V2.1.2 | 更更新章节 3.3.9 |
| 2018.02 | V2.2 | 更更新章节 6.2.1，6.2.3，6.2.4，3.3.49 新增章节 3.4.8，3.4.9，3.5.74，3.5.75 |
| 2018.05 | V2.2.1 | 更更新章节 2.4，3.5.54，3.7 |
| 2018.08 | V3.0 | 新增章节 3.3.50，3.3.51，3.3.52，3.5.76，3.5.77，3.5.78，  3.5.79，附录 A.6 更更新章节 2.5 删除 system\_phy\_freq\_trace\_enable |
| 2019.03 | V3.0.1 | 删除章节 3.5.76，3.5.77，3.5.78，3.5.79 |
| 2020.09 | V3.0.2 | 删除关于⽂文档《ESP8266 Flash 读写说明》的链接 |

⽂文档变更更通知

⽤用户可通过乐鑫官⽹网订阅⻚页⾯面 [*https://www.espressif.com/zh-hans/subscribe*](https://www.espressif.com/zh-hans/subscribe) 订阅技术⽂文档变更更的电⼦子邮件通知。

证书下载

⽤用户可通过乐鑫官⽹网证书下载⻚页⾯面 [*https://www.espressif.com/zh-hans/certificates*](https://www.espressif.com/zh-hans/certificates) 下载产品证书。

# ⽬目录

1. 前⾔言 .........................................................................................................................................1
2. Non-OS SDK .......................................................................................................................2
   1. Non-OS SDK 简介 ......................................................................................................................2
   2. 代码结构 .....................................................................................................................................2
   3. 定时器器 (timer) 和中断 ..................................................................................................................4
   4. 系统性能 .....................................................................................................................................4
   5. 系统存储 .....................................................................................................................................4
3. 应⽤用程序接⼝口 (API) ...............................................................................................................7
   1. 软件定时器器 .................................................................................................................................7
      1. os\_timer\_arm ................................................................................................................7
      2. os\_timer\_disarm ...........................................................................................................7
      3. os\_timer\_setfn ..............................................................................................................8
      4. system\_timer\_reinit .......................................................................................................8
      5. os\_timer\_arm\_us ..........................................................................................................8
   2. 硬件中断定时器器 ..........................................................................................................................8
      1. hw\_timer\_init .................................................................................................................9
      2. hw\_timer\_arm ...............................................................................................................9
      3. hw\_timer\_set\_func ......................................................................................................10
      4. 硬件定时器器示例例 ...........................................................................................................10
   3. 系统接⼝口 ...................................................................................................................................10
      1. system\_get\_sdk\_version ............................................................................................10
      2. system\_restore ...........................................................................................................11
      3. system\_restart ............................................................................................................11
      4. system\_init\_done\_cb ..................................................................................................11
      5. system\_get\_chip\_id ....................................................................................................11
      6. system\_get\_vdd33 .....................................................................................................12
      7. system\_adc\_read ........................................................................................................12
      8. system\_adc\_read\_fast ................................................................................................13
      9. system\_deep\_sleep ....................................................................................................14
      10. system\_deep\_sleep\_set\_option .................................................................................15
      11. system\_phy\_set\_rfoption ............................................................................................15
      12. system\_phy\_set\_powerup\_option ..............................................................................16
      13. system\_phy\_set\_max\_tpw .........................................................................................16
      14. system\_phy\_set\_tpw\_via\_vdd33 ................................................................................16
      15. system\_set\_os\_print ...................................................................................................17
      16. system\_print\_meminfo ...............................................................................................17
      17. system\_get\_free\_heap\_size ........................................................................................17
      18. system\_os\_task ..........................................................................................................17
      19. system\_os\_post ..........................................................................................................18
      20. system\_get\_time .........................................................................................................18
      21. system\_get\_rtc\_time ...................................................................................................19
      22. system\_rtc\_clock\_cali\_proc ........................................................................................19
      23. system\_rtc\_mem\_write ...............................................................................................19
      24. system\_rtc\_mem\_read ...............................................................................................20
      25. system\_uart\_swap ......................................................................................................20
      26. system\_uart\_de\_swap ................................................................................................20
      27. system\_get\_boot\_version ...........................................................................................21
      28. system\_get\_userbin\_addr........................................................................................... 21
      29. system\_get\_boot\_mode .............................................................................................21
      30. system\_restart\_enhance .............................................................................................21
      31. system\_update\_cpu\_req ............................................................................................22
      32. system\_get\_cpu\_freq .................................................................................................22
      33. system\_get\_flash\_size\_map .......................................................................................22
      34. system\_get\_rst\_info ....................................................................................................23
      35. system\_soft\_wdt\_stop ................................................................................................23
      36. system\_soft\_wdt\_restart .............................................................................................23
      37. system\_soft\_wdt\_feed ................................................................................................24
      38. system\_show\_malloc ..................................................................................................24
      39. os\_memset .................................................................................................................24
      40. os\_memcpy ................................................................................................................25
      41. os\_strlen .....................................................................................................................25
      42. os\_printf ......................................................................................................................25
      43. os\_bzero .....................................................................................................................25
      44. os\_delay\_us ................................................................................................................26
      45. os\_install\_putc1 ..........................................................................................................26
      46. os\_random ..................................................................................................................26
      47. os\_get\_random ...........................................................................................................26
      48. user\_rf\_cal\_sector\_set ................................................................................................26
      49. system\_deep\_sleep\_instant .......................................................................................28
      50. system\_partition\_table\_regist .....................................................................................28
      51. system\_partition\_get\_ota\_partition\_size ....................................................................28
      52. system\_partition\_get\_item ..........................................................................................29
   4. SPI Flash 接⼝口 ..........................................................................................................................29
      1. spi\_flash\_get\_id ..........................................................................................................29
      2. spi\_flash\_erase\_sector ...............................................................................................29
      3. spi\_flash\_write ............................................................................................................29
      4. spi\_flash\_read .............................................................................................................30
      5. system\_param\_save\_with\_protect .............................................................................30
      6. system\_param\_load ...................................................................................................31
      7. spi\_flash\_set\_read\_func .............................................................................................31
      8. spi\_flash\_erase\_protect\_enable .................................................................................32
      9. spi\_flash\_erase\_protect\_disable .................................................................................32
   5. Wi-Fi 接⼝口 .................................................................................................................................33
      1. wifi\_get\_opmode ........................................................................................................33
      2. wifi\_get\_opmode\_default............................................................................................ 33
      3. wifi\_set\_opmode .........................................................................................................33
      4. wifi\_set\_opmode\_current ...........................................................................................34
      5. wifi\_station\_get\_config ...............................................................................................34
      6. wifi\_station\_get\_config\_default ..................................................................................34
      7. wifi\_station\_set\_config ...............................................................................................35
      8. wifi\_station\_set\_config\_current ..................................................................................35
      9. wifi\_station\_set\_cert\_key ............................................................................................36
      10. wifi\_station\_clear\_cert\_key .........................................................................................37
      11. wifi\_station\_set\_username .........................................................................................37
      12. wifi\_station\_clear\_username ......................................................................................37
      13. wifi\_station\_connect ...................................................................................................37
      14. wifi\_station\_disconnect ..............................................................................................38
      15. wifi\_station\_get\_connect\_status ................................................................................38
      16. wifi\_station\_scan ........................................................................................................38
      17. scan\_done\_cb\_t .........................................................................................................39
      18. wifi\_station\_ap\_number\_set .......................................................................................39
      19. wifi\_station\_get\_ap\_info .............................................................................................39
      20. wifi\_station\_ap\_change ..............................................................................................40
      21. wifi\_station\_get\_current\_ap\_id ...................................................................................40
      22. wifi\_station\_get\_auto\_connect ...................................................................................40
      23. wifi\_station\_set\_auto\_connect ...................................................................................40
      24. wifi\_station\_dhcpc\_start .............................................................................................41
      25. wifi\_station\_dhcpc\_stop .............................................................................................41
      26. wifi\_station\_dhcpc\_status ..........................................................................................41
      27. wifi\_station\_dhcpc\_set\_maxtry ..................................................................................41
      28. wifi\_station\_set\_reconnect\_policy ..............................................................................42
      29. wifi\_station\_get\_rssi ...................................................................................................42
      30. wifi\_station\_set\_hostname .........................................................................................42
      31. wifi\_station\_get\_hostname .........................................................................................42
      32. wifi\_softap\_get\_config ................................................................................................42
      33. wifi\_softap\_get\_config\_default ...................................................................................43
      34. wifi\_softap\_set\_config ................................................................................................43
      35. wifi\_softap\_set\_config\_current ...................................................................................43
      36. wifi\_softap\_get\_station\_num ......................................................................................43
      37. wifi\_softap\_get\_station\_info .......................................................................................44
      38. wifi\_softap\_free\_station\_info ......................................................................................44
      39. wifi\_softap\_dhcps\_start ..............................................................................................44
      40. wifi\_softap\_dhcps\_stop ..............................................................................................45
      41. wifi\_softap\_set\_dhcps\_lease ......................................................................................45
      42. wifi\_softap\_get\_dhcps\_lease ......................................................................................46
      43. wifi\_softap\_set\_dhcps\_lease\_time .............................................................................46
      44. wifi\_softap\_get\_dhcps\_lease\_time .............................................................................47
      45. wifi\_softap\_reset\_dhcps\_lease\_time ..........................................................................47
      46. wifi\_softap\_dhcps\_status ...........................................................................................47
      47. wifi\_softap\_set\_dhcps\_offer\_option ...........................................................................47
      48. wifi\_set\_phy\_mode .....................................................................................................48
      49. wifi\_get\_phy\_mode .....................................................................................................48
      50. wifi\_get\_ip\_info ...........................................................................................................48
      51. wifi\_set\_ip\_info ...........................................................................................................49
      52. wifi\_set\_macaddr ........................................................................................................49
      53. wifi\_get\_macaddr .......................................................................................................50
      54. wifi\_set\_sleep\_type .....................................................................................................50
      55. wifi\_get\_sleep\_type ....................................................................................................50
      56. wifi\_status\_led\_install .................................................................................................51
      57. wifi\_status\_led\_uninstall .............................................................................................51
      58. wifi\_set\_broadcast\_if ..................................................................................................51
      59. wifi\_get\_broadcast\_if ..................................................................................................52
      60. wifi\_set\_event\_handler\_cb ..........................................................................................52
      61. wifi\_wps\_enable .........................................................................................................53
      62. wifi\_wps\_disable .........................................................................................................54
      63. wifi\_wps\_start .............................................................................................................54
      64. wifi\_set\_wps\_cb .........................................................................................................54
      65. wifi\_register\_send\_pkt\_freedom\_cb ...........................................................................55
      66. wifi\_unregister\_send\_pkt\_freedom\_cb .......................................................................55
      67. wifi\_send\_pkt\_freedom ...............................................................................................56
      68. wifi\_rfid\_locp\_recv\_open ............................................................................................56
      69. wifi\_rfid\_locp\_recv\_close ............................................................................................56
      70. wifi\_register\_rfid\_locp\_recv\_cb ..................................................................................57
      71. wifi\_unregister\_rfid\_locp\_recv\_cb ..............................................................................57
      72. wifi\_enable\_gpio\_wakeup ...........................................................................................57
      73. wifi\_disable\_gpio\_wakeup ..........................................................................................57
      74. wifi\_set\_country ..........................................................................................................58
      75. wifi\_get\_country ..........................................................................................................58
   6. Rate Control 接⼝口 .....................................................................................................................59
      1. wifi\_set\_user\_fixed\_rate .............................................................................................59
      2. wifi\_get\_user\_fixed\_rate .............................................................................................59
      3. wifi\_set\_user\_sup\_rate ...............................................................................................60
      4. wifi\_set\_user\_rate\_limit ...............................................................................................61
      5. wifi\_set\_user\_limit\_rate\_mask ....................................................................................62
      6. wifi\_get\_user\_limit\_rate\_mask ....................................................................................62
   7. 强制休眠接⼝口 ............................................................................................................................63
      1. wifi\_fpm\_open ............................................................................................................63
      2. wifi\_fpm\_close ............................................................................................................63
      3. wifi\_fpm\_do\_wakeup ..................................................................................................63
      4. wifi\_fpm\_set\_wakeup\_cb ...........................................................................................64
      5. wifi\_fpm\_do\_sleep ......................................................................................................64
      6. wifi\_fpm\_set\_sleep\_type............................................................................................. 64
      7. wifi\_fpm\_get\_sleep\_type ............................................................................................65
      8. wifi\_fpm\_auto\_sleep\_set\_in\_null\_mode .....................................................................65
      9. 示例例代码 .....................................................................................................................65
   8. ESP-NOW 接⼝口 ........................................................................................................................68
      1. 结构体 .........................................................................................................................68
      2. esp\_now\_init ...............................................................................................................68
      3. esp\_now\_deinit ...........................................................................................................69
      4. esp\_now\_register\_recv\_cb .........................................................................................69
      5. esp\_now\_unregister\_recv\_cb .....................................................................................69
      6. esp\_now\_register\_send\_cb ........................................................................................70
      7. esp\_now\_unregister\_send\_cb ....................................................................................70
      8. esp\_now\_send ............................................................................................................71
      9. esp\_now\_add\_peer ....................................................................................................71
      10. esp\_now\_del\_peer ......................................................................................................71
      11. esp\_now\_set\_self\_role ................................................................................................71
      12. esp\_now\_get\_self\_role ...............................................................................................72
      13. esp\_now\_set\_peer\_role ..............................................................................................72
      14. esp\_now\_get\_peer\_role ..............................................................................................72
      15. esp\_now\_set\_peer\_key ..............................................................................................72
      16. esp\_now\_get\_peer\_key ..............................................................................................73
      17. esp\_now\_set\_peer\_channel .......................................................................................73
      18. esp\_now\_get\_peer\_channel .......................................................................................73
      19. esp\_now\_is\_peer\_exist ...............................................................................................73
      20. esp\_now\_fetch\_peer ...................................................................................................74
      21. esp\_now\_get\_cnt\_info ................................................................................................74
      22. esp\_now\_set\_kok .......................................................................................................74
   9. Simple-Pair 接⼝口 ......................................................................................................................75
      1. 结构体 .........................................................................................................................75
      2. register\_simple\_pair\_status\_cb ..................................................................................75
      3. unregister\_simple\_pair\_status\_cb ..............................................................................75
      4. simple\_pair\_init ...........................................................................................................76
      5. simple\_pair\_deinit .......................................................................................................76
      6. simple\_pair\_state\_reset ..............................................................................................76
      7. simple\_pair\_ap\_enter\_announce\_mode .....................................................................76
      8. simple\_pair\_sta\_enter\_scan\_mode ............................................................................76
      9. simple\_pair\_sta\_start\_negotiate .................................................................................77
      10. simple\_pair\_ap\_start\_negotiate ..................................................................................77
      11. simple\_pair\_ap\_refuse\_negotiate ...............................................................................77
      12. simple\_pair\_set\_peer\_ref ............................................................................................77
      13. simple\_pair\_get\_peer\_ref............................................................................................ 78
   10. 云端升级 (FOTA) 接⼝口 ...............................................................................................................79
       1. system\_upgrade\_userbin\_check ................................................................................79
       2. system\_upgrade\_flag\_set ...........................................................................................79
       3. system\_upgrade\_flag\_check ......................................................................................79
       4. system\_upgrade\_start ................................................................................................79
       5. system\_upgrade\_reboot .............................................................................................80
   11. Sniffer 相关接⼝口 .......................................................................................................................81
       1. wifi\_promiscuous\_enable ...........................................................................................81
       2. wifi\_promiscuous\_set\_mac ........................................................................................81
       3. wifi\_set\_promiscuous\_rx\_cb ......................................................................................81
       4. wifi\_get\_channel .........................................................................................................82
       5. wifi\_set\_channel .........................................................................................................82
   12. SmartConfig 接⼝口 .....................................................................................................................83
       1. smartconfig\_start ........................................................................................................83
       2. smartconfig\_stop ........................................................................................................84
       3. smartconfig\_set\_type .................................................................................................84
       4. airkiss\_version ............................................................................................................85
       5. airkiss\_lan\_recv ..........................................................................................................85
       6. airkiss\_lan\_pack .........................................................................................................86
   13. SNTP 接⼝口 ................................................................................................................................87
       1. sntp\_setserver ............................................................................................................87
       2. sntp\_getserver ............................................................................................................87
       3. sntp\_setservername ...................................................................................................87
       4. sntp\_getservername ...................................................................................................87
       5. sntp\_init ......................................................................................................................87
       6. sntp\_stop ....................................................................................................................88
       7. sntp\_get\_current\_timestamp ......................................................................................88
       8. sntp\_get\_real\_time .....................................................................................................88
       9. sntp\_set\_timezone ......................................................................................................88

3.13.10.sntp\_get\_timezone .....................................................................................................89

* 1. 11.SNTP 示例例 ..................................................................................................................89
  2. WPA2-Enterprise 接⼝口 ..............................................................................................................90
     1. wifi\_station\_set\_wpa2\_enterprise\_auth ......................................................................90
     2. wifi\_station\_set\_enterprise\_cert\_key ..........................................................................90
     3. wifi\_station\_clear\_enterprise\_cert\_key .......................................................................91
     4. wifi\_station\_set\_enterprise\_ca\_cert ............................................................................91
     5. wifi\_station\_clear\_enterprise\_ca\_cert .........................................................................91
     6. wifi\_station\_set\_enterprise\_username ........................................................................91
     7. wifi\_station\_clear\_enterprise\_username .....................................................................92
     8. wifi\_station\_set\_enterprise\_password ........................................................................92
     9. wifi\_station\_clear\_enterprise\_password .....................................................................92
  3. 10.wifi\_station\_set\_enterprise\_new\_password ...............................................................92

3.14.11.wifi\_station\_clear\_enterprise\_new\_password ............................................................93

3.14.12.wifi\_station\_set\_enterprise\_disable\_time\_check .......................................................93

3.14.13.wifi\_station\_get\_enterprise\_disable\_time\_check .......................................................93

3.14.14.wpa2\_enterprise\_set\_user\_get\_time ..........................................................................93 3.14.15.示例例流程 .....................................................................................................................94

1. TCP/UDP 接⼝口 ..................................................................................................................95
   1. 通⽤用接⼝口 ...................................................................................................................................95
      1. espconn\_delete ..........................................................................................................95
      2. espconn\_gethostbyname ...........................................................................................95
      3. espconn\_port ..............................................................................................................96
      4. espconn\_regist\_sentcb ...............................................................................................96
      5. espconn\_regist\_recvcb ...............................................................................................97
      6. espconn\_sent\_callback ..............................................................................................97
      7. espconn\_recv\_callback ..............................................................................................97
      8. espconn\_get\_connection\_info ....................................................................................97
      9. espconn\_send ............................................................................................................98
      10. espconn\_sent .............................................................................................................99
   2. TCP 接⼝口 ................................................................................................................................100
      1. espconn\_accept .......................................................................................................100
      2. espconn\_regist\_time .................................................................................................100
      3. espconn\_connect .....................................................................................................100
      4. espconn\_regist\_connectcb ......................................................................................101
      5. espconn\_connect\_callback ......................................................................................101
      6. espconn\_set\_opt ......................................................................................................101
      7. espconn\_clear\_opt ...................................................................................................102
      8. espconn\_set\_keepalive ............................................................................................102
      9. espconn\_get\_keepalive ............................................................................................103
      10. espconn\_reconnect\_callback ...................................................................................103
      11. espconn\_regist\_reconcb ..........................................................................................104
      12. espconn\_disconnect ................................................................................................104
      13. espconn\_regist\_disconcb .........................................................................................105
      14. espconn\_abort ..........................................................................................................105
      15. espconn\_regist\_write\_finish .....................................................................................105
      16. espconn\_tcp\_get\_max\_con ......................................................................................106
      17. espconn\_tcp\_set\_max\_con ......................................................................................106
      18. espconn\_tcp\_get\_max\_con\_allow ...........................................................................106
      19. espconn\_tcp\_set\_max\_con\_allow ............................................................................106
      20. espconn\_recv\_hold ...................................................................................................106
      21. espconn\_recv\_unhold ...............................................................................................107
      22. espconn\_secure\_accept ...........................................................................................107
      23. espconn\_secure\_delete ............................................................................................107
      24. espconn\_secure\_set\_size .........................................................................................108
      25. espconn\_secure\_get\_size ........................................................................................108
      26. espconn\_secure\_connect .........................................................................................109
      27. espconn\_secure\_send ..............................................................................................109
      28. espconn\_secure\_sent ...............................................................................................110
      29. espconn\_secure\_disconnect ....................................................................................110
      30. espconn\_secure\_ca\_enable .....................................................................................110
      31. espconn\_secure\_ca\_disable ....................................................................................111
      32. espconn\_secure\_cert\_req\_enable ............................................................................111
      33. espconn\_secure\_cert\_req\_disable ...........................................................................111
      34. espconn\_secure\_set\_default\_certificate ..................................................................112
      35. espconn\_secure\_set\_default\_private\_key ................................................................112
   3. UDP 接⼝口 ................................................................................................................................113
      1. espconn\_create ........................................................................................................113
      2. espconn\_sendto .......................................................................................................113
      3. espconn\_igmp\_join ...................................................................................................113
      4. espconn\_igmp\_leave ................................................................................................114
      5. espconn\_dns\_setserver ............................................................................................114
      6. espconn\_dns\_getserver ...........................................................................................114
   4. mDNS 接⼝口 ............................................................................................................................115
      1. espconn\_mdns\_init ...................................................................................................115
      2. espconn\_mdns\_close ...............................................................................................115
      3. espconn\_mdns\_server\_register ................................................................................115
      4. espconn\_mdns\_server\_unregister ............................................................................116
      5. espconn\_mdns\_get\_servername ..............................................................................116
      6. espconn\_mdns\_set\_servername ..............................................................................116
      7. espconn\_mdns\_set\_hostname .................................................................................116
      8. espconn\_mdns\_get\_hostname ................................................................................116
      9. espconn\_mdns\_disable ............................................................................................117
      10. espconn\_mdns\_enable .............................................................................................117 4.4.11. mDNS 示例例 ...............................................................................................................117
2. 应⽤用相关接⼝口 ...................................................................................................................118
   1. AT 接⼝口 ...................................................................................................................................118
      1. at\_response\_ok ........................................................................................................118
      2. at\_response\_error .....................................................................................................118
      3. at\_cmd\_array\_regist .................................................................................................118
      4. at\_get\_next\_int\_dec ..................................................................................................119
      5. at\_data\_str\_copy ......................................................................................................119
      6. at\_init ........................................................................................................................119
      7. at\_port\_print .............................................................................................................120
      8. at\_set\_custom\_info ...................................................................................................120
      9. at\_enter\_special\_state ..............................................................................................120
      10. at\_leave\_special\_state ..............................................................................................120
      11. at\_get\_version ..........................................................................................................120
      12. at\_register\_uart\_rx\_intr .............................................................................................121
      13. at\_response ..............................................................................................................121
      14. at\_register\_response\_func ........................................................................................121
      15. at\_fake\_uart\_enable ..................................................................................................122
      16. at\_fake\_uart\_rx .........................................................................................................122
      17. at\_set\_escape\_character ..........................................................................................122
   2. JSON 接⼝口 .............................................................................................................................123
      1. jsonparse\_setup .......................................................................................................123
      2. jsonparse\_next .........................................................................................................123
      3. jsonparse\_copy\_value ..............................................................................................123
      4. jsonparse\_get\_value\_as\_int ......................................................................................123
      5. jsonparse\_get\_value\_as\_long ...................................................................................124
      6. jsonparse\_get\_len .....................................................................................................124
      7. jsonparse\_get\_value\_as\_type ...................................................................................124
      8. jsonparse\_strcmp\_value ...........................................................................................124
      9. jsontree\_set\_up ........................................................................................................124
      10. jsontree\_reset ...........................................................................................................125
      11. jsontree\_path\_name .................................................................................................125
      12. jsontree\_write\_int ......................................................................................................125
      13. jsontree\_write\_int\_array ............................................................................................126
      14. jsontree\_write\_string .................................................................................................126
      15. jsontree\_print\_next ...................................................................................................126
      16. jsontree\_find\_next ....................................................................................................126
3. 参数结构体和宏定义 ........................................................................................................127
   1. 定时器器 ....................................................................................................................................127
   2. Wi-Fi 参数 ...............................................................................................................................127
      1. Station 参数 ..............................................................................................................127
      2. SoftAP 参数 ..............................................................................................................127
      3. Scan 参数 .................................................................................................................128
      4. Wi-Fi Event 结构体 ....................................................................................................129
      5. SmartConfig 结构体 ..................................................................................................131
   3. JSON 相关结构体 ...................................................................................................................131
      1. JSON 结构体 ............................................................................................................131
      2. JSON 宏定义 ............................................................................................................132
   4. espconn 参数 .........................................................................................................................133
      1. 回调函数 ...................................................................................................................133
      2. espconn ....................................................................................................................133 6.5. 中断相关宏定义 ......................................................................................................................134
4. 外设驱动接⼝口 ...................................................................................................................136
   1. GPIO 接⼝口 ..............................................................................................................................136
      1. PIN 相关宏定义 .........................................................................................................136
      2. gpio\_output\_set ........................................................................................................136
      3. GPIO 输⼊入输出相关宏 ...............................................................................................137
      4. GPIO 中断 .................................................................................................................137
      5. gpio\_pin\_intr\_state\_set .............................................................................................137
      6. GPIO 中断处理理函数 ...................................................................................................137
   2. UART 接⼝口 ..............................................................................................................................138
      1. uart\_init .....................................................................................................................138
      2. uart0\_tx\_buffer ..........................................................................................................138
      3. uart0\_rx\_intr\_handler ................................................................................................138
      4. uart\_div\_modify ........................................................................................................139
   3. I2C Master 接⼝口 .....................................................................................................................139
      1. i2c\_master\_gpio\_init .................................................................................................139
      2. i2c\_master\_init ..........................................................................................................139
      3. i2c\_master\_start .......................................................................................................140
      4. i2c\_master\_stop .......................................................................................................140
      5. i2c\_master\_send\_ack ...............................................................................................140
      6. i2c\_master\_send\_nack .............................................................................................140
      7. i2c\_master\_checkAck ...............................................................................................140
      8. i2c\_master\_readByte ................................................................................................141
      9. i2c\_master\_writeByte ...............................................................................................141
   4. PWM 接⼝口 ..............................................................................................................................141
      1. pwm\_init ...................................................................................................................141
      2. pwm\_start .................................................................................................................142
      3. pwm\_set\_duty ..........................................................................................................142
      4. pwm\_get\_duty ..........................................................................................................142
      5. pwm\_set\_period .......................................................................................................142
      6. pwm\_get\_period .......................................................................................................143
      7. get\_pwm\_version ......................................................................................................143
   5. SDIO 接⼝口 ..............................................................................................................................143
      1. sdio\_slave\_init ..........................................................................................................143
      2. sdio\_load\_data .........................................................................................................143 7.5.3. sdio\_register\_recv\_cb ...............................................................................................144
5. 附录 .................................................................................................................................145
   1. ESPCONN 编程 ......................................................................................................................145
      1. TCP Client 模式 ........................................................................................................145
      2. TCP Server 模式 .......................................................................................................145
      3. espconn Callback .....................................................................................................146
   2. RTC API 使⽤用示例例 ..................................................................................................................146
   3. Sniffer 说明 .............................................................................................................................148
   4. ESP8266 SoftAP 和 Station 信道定义 ....................................................................................148
   5. ESP8266 启动信息说明 ..........................................................................................................149
   6. ESP8266 信令测试使⽤用说明 ...................................................................................................150

1. 前⾔言

## 1. 前⾔言

ESP8266EX 由乐鑫公司开发，提供了了⼀一套⾼高度集成的 Wi-Fi SoC 解决⽅方案，其低功耗、紧凑设计和⾼高稳定性可以满⾜足⽤用户的需求。

ESP8266EX 拥有完整的且⾃自成体系的 Wi-Fi ⽹网络功能，既能够独⽴立应⽤用，也可以作为从机搭载于其他主机 MCU 运⾏行行。当 ESP8266EX 独⽴立应⽤用时，能够直接从外接 Flash 中启动。内置的⾼高速缓冲存储器器有利利于提⾼高系统性能，并且优化存储系统。此外 ESP8266EX 只需通过 SPI/SDIO 接⼝口或 I2C/UART ⼝口即可作为 Wi-Fi 适配器器，应⽤用到基于任何微控制器器的设计中。

ESP8266EX 集成了了天线开关、射频 balun、功耗放⼤大器器、低噪放⼤大器器、过滤器器和电源管理理模块。这样紧凑的设计仅需极少的外部电路路并且将 PCB 的尺⼨寸降到 ⼩小。

ESP8266EX 还集成了了增强版的 Tensilica’s L106 钻⽯石系列列 32-bit 内核处理理器器，带⽚片上

SRAM。ESP8266EX 可以通过 GPIO 外接传感器器和其他设备。软件开发包 (SDK) 提供了了⼀一些应⽤用的示例例代码。

乐鑫智能互联平台 (ESCP-Espressif Systems’ Smart Connectivity Platform) 表现出来的领先特征有：睡眠/唤醒模式之间的快速切换以实现节能、配合低功耗操作的⾃自适应射频偏

置、前端信号的处理理功能、故障排除和射频共存机制可消除蜂窝/蓝⽛牙/DDR/LVDS/LCD ⼲干扰。

基于 ESP8266EX 物联⽹网平台的 SDK 为⽤用户提供了了⼀一个简单、快速、⾼高效开发物联⽹网产品的软件平台。本⽂文旨在介绍该 SDK 的基本框架，以及相关的 API 接⼝口。主要的阅读对象为需要在 ESP8266 物联⽹网平台进⾏行行软件开发的嵌⼊入式软件开发⼈人员。

## 2. Non-OS SDK

### 2.1. Non-OS SDK 简介

Non-OS SDK 为⽤用户提供了了⼀一套应⽤用程序编程接⼝口 (API)，能够实现 ESP8266 的核⼼心功能改，例例如数据接收/发送、TCP/IP 功能、硬件接⼝口功能，以及基本的系统管理理功能等。⽤用户不不必关⼼心底层⽹网络，如 Wi-Fi、TCP/IP 等的具体实现，只需要专注于物联⽹网上层应⽤用的开发，利利⽤用相应接⼝口实现各种功能即可。

ESP8266 物联⽹网平台的所有⽹网络功能均在库中实现，对⽤用户不不透明。⽤用户应⽤用的初始化功能可以在 *user\_main.c*中实现。 void user\_init(void) 是上层程序的⼊入⼝口函数，给⽤用户提供⼀一个初始化接⼝口，⽤用户可在该函数内增加硬件初始化、⽹网络参数设置、定时器器初始化等功能。

* 对于 *ESP8266\_NONOS\_SDK\_v3.0.0* 及之后版本，请在 *user\_main.c* 增加函数

void ICACHE\_FLASH\_ATTR user\_pre\_init(void)，并且在 user\_pre\_init 中注册⾃自⼰己的 partition table。

* 对于 *ESP8266\_NONOS\_SDK\_v1.5.2* ⾄至 *ESP8266\_NONOS\_SDK\_v2.2.1* 之间的版本，请在 *user\_main.c* 增加函数 void user\_rf\_pre\_init(void) 和 uint32 user\_rf\_cal\_sector\_set(void)，可参考 *IOT\_Demo* 的 *user\_main.c*。⽤用户可在 user\_rf\_pre\_init 中配置 RF 初始化，RF 设置接⼝口为 system\_phy\_set\_rfoption，或者在 Deep-sleep 前调⽤用 system\_deep\_sleep\_set\_option。如果设置为 RF 不不打开，则 ESP8266 Station 及 SoftAP 均⽆无法使⽤用，请勿调⽤用 Wi-Fi 相关接⼝口及⽹网络功能。RF 关闭时，Wi-Fi 射频功能和⽹网络堆栈管理理 API 均⽆无法使⽤用。

对于 *ESP8266\_NONOS\_SDK\_v2.1.0* 及之后版本，⽤用户如果并未使⽤用 DIO-To-QIO flash，可以在 *user\_main.c* 中增加空函数 void user\_spi\_flash\_dio\_to\_qio\_pre\_init(void) 来优化 iRAM 空间。

SDK 中提供了了对 JSON 包的处理理 API，⽤用户也可以采⽤用⾃自定义数据包格式，⾃自⾏行行对数据进⾏行行处理理。

### 2.2. 代码结构

Non-OS SDK 适⽤用于⽤用户需要完全控制代码执⾏行行顺序的应⽤用程序。由于没有操作系统， non-OS SDK 不不⽀支持任务调度，也不不⽀支持基于优先级的抢占。

Non-OS SDK 适合⽤用于事件驱动的应⽤用程序。由于没有操作系统，non-OS SDK 没有单个任务堆栈⼤大⼩小的限制或者执⾏行行时隙要求。

⽽而 RTOS SDK 可⽤用于基于任务的模块化编程。要了了解有关 RTOS SDK 的更更多信息，请参阅[《ESP8266 SDK ⼊入⻔门指南》。](http://espressif.com/sites/default/files/documentation/2a-esp8266-sdk_getting_started_guide_cn.pdf)

Non-OS SDK 中的代码结构具有以下特征：

• Non-OS SDK 不不像基于 RTOS 的应⽤用程序⽀支持任务调度。Non-OS SDK 使⽤用四种类型的函数：

* 应⽤用函数
* 回调函数
* ⽤用户任务
* 中断服务程序 (Interrupt Service Routines, ISR)

应⽤用函数类似于嵌⼊入式 C 编程中的常⽤用 C 函数。这些函数必须由另⼀一个函数调⽤用。应⽤用函数在定义时建议添加 ICACHE\_FLASH\_ATTR 宏，相应程序将存放在 flash 中，被调⽤用时才加载到 cache 运⾏行行。⽽而如果添加了了 IRAM\_ATTR 宏的函数，则会在上电启动时就加载到 iRAM 中。

回调函数是指不不直接从⽤用户程序调⽤用的函数，⽽而是当某系统事件发⽣生时，相应的回调函数由 non-OS SDK 内核调⽤用执⾏行行。这使得开发者能够在不不使⽤用 RTOS 或者轮询事件的情况下响应实时事件。

要编写回调函数，⽤用户⾸首先需要使⽤用相应的 register\_cb API 注册回调函数。回调函数的示例例包括定时器器回调函数和⽹网络事件回调函数。

中断服务程序 (ISR) 是⼀一种特殊类型的回调函数。发⽣生硬件中断时会调⽤用这些函数。当使能中断时，必须注册相应的中断处理理函数。请注意，ISR 必须添加

IRAM\_ATTR。

⽤用户任务可以分为三个优先级：0、1、2。任务优先级为 2 > 1 > 0。即 Non-OS

SDK 多只⽀支持 3 个⽤用户任务，优先级分别为 0、1、2。

⽤用户任务⼀一般⽤用于函数不不能直接被调⽤用的情况下。要创建⽤用户任务，请参阅本⽂文档中的 system\_os\_task() 的 API 描述。例例如，espconn\_disconnect() API 不不能直接在 espconn 的回调函数中调⽤用，因此建议开发者可以在 espconn 回调中创建⽤用户任务来执⾏行行 espconn\_disconnect。

* 如前所述，non-OS SDK 不不⽀支持抢占任务或进程切换。因此开发者需要⾃自⾏行行保证程序的正确执⾏行行，⽤用户代码不不能⻓长期占⽤用 CPU。否则会导致看⻔门狗复位，ESP8266 重启。

如果某些特殊情况下，⽤用户线程必须执⾏行行较⻓长时间（⽐比如⼤大于 500 ms），建议经常调⽤用 system\_soft\_wdt\_feed() API 来喂软件看⻔门狗，⽽而不不建议禁⽤用软件看⻔门狗。

* 请注意，*esp\_init\_data.bin* 和 *blank.bin* ⽂文件⾄至少需要烧录⼀一次，以⽤用于正确的初始化系统。对于 *ESP8266\_NONOS\_SDK\_v2.2.1* 及之前的版本，应⽤用程序必须在 user\_rf\_cal\_sector\_set 中设置 RF 校准扇区。

#### 2.3. 定时器器 (timer) 和中断

* 对于需要进⾏行行轮询的应⽤用，建议使⽤用系统定时器器定期检查事件。
  + 如果使⽤用循环（while 或 for），不不仅效率低下，⽽而且阻塞 CPU，不不建议使

⽤用。

* + 如果需要在定时器器回调中执⾏行行 os\_delay\_us 或 while 或 for，请勿占⽤用 CPU 超过 15 ms。
* 请勿频繁调⽤用定时器器，建议频率不不⾼高于每 5 ms ⼀一次（微秒计时器器则为 100 μs）。有关定时器器使⽤用的详细信息，请参阅 os\_timer\_arm()和相关的 API 说明。
* 微秒定时器器不不是很精确，请在回调中考虑 500 μs 的抖动。如需实现⾼高精度的定时，可以参考驱动程序 (*driver\_lib*) 使⽤用硬件定时器器。请注意，PWM API 不不能与硬件定时器器同时使⽤用。
* 请勿⻓长时间关闭中断。ISR 执⾏行行时间也应当尽可能短（即微秒级）。

### 2.4. 系统性能

* ESP8266 通常的运⾏行行速率为 80 MHz，在⾼高性能应⽤用中也可以配置为160 MHz。请注意，外设不不受 CPU 频率设置的影响，因为它们使⽤用了了不不同的时钟源。
* 设置更更⾼高的时钟频率或者禁⽤用睡眠模式，会导致更更⼤大的功耗，但能获得更更好的性能。

应⽤用程序应考虑两者之间的平衡。

* 添加了了 ICACHE\_FLASH\_ATTR 的代码通常⽐比使⽤用 IRAM\_ATTR 标记的代码执⾏行行得慢。然

⽽而，像⼤大多数嵌⼊入式平台⼀一样，ESP8266 的 iRAM 空间有限，因此建议⼀一般代码添加

ICACHE\_FLASH\_ATTR，仅对执⾏行行效率要求⾼高的代码添加 IRAM\_ATTR 宏。

* Flash 模式和频率直接影响代码执⾏行行速度。将 flash 设置为更更⾼高的频率和 QIO 模式会产

⽣生更更好的性能，但会导致更更⼤大的功耗。

### 2.5. 系统存储

* ESP8266 ⽀支持⾼高达 128 Mbits 的外部 QSPI flash，⽤用于存储代码和数据。也可以使⽤用辅助存储芯⽚片来存储⽤用户数据。
* ESP8266 没有存储⽤用户代码或数据的⾮非易易失性存储。ESP8285 是⼀一款在 ESP8266 的基础上集成了了 flash 的芯⽚片。更更多详细信息请参考 [ESP8285 技术规格书。](http://espressif.com/sites/default/files/documentation/0a-esp8285_datasheet_cn.pdf)
* ESP8266 带有 160 KB 的 RAM，其中 64 KB 为 iRAM，96 KB 为 dRAM。iRAM 进⼀一步分成两块：32 KB iRAM 块运⾏行行标有 IRAM\_ATTR 的代码，另⼀一个 32 KB 块⽤用作 cache，运⾏行行标有 ICACHE\_FLASH\_ATTR 的代码。
* 从 ESP8266\_NonOS\_SDK\_V3.0 开始，增加了了⽀支持使⽤用 iRAM 作为内存的功能，能够多提供约 17 KB 的内存， 对性能可能有⼀一定的影响，请根据实际应⽤用需求设置，并建议做详细测试进⾏行行确认。使⽤用⽅方法如下：
* 在应⽤用中定义 user\_iram\_memory\_is\_enabled 函数并设置返回值为 1。示例例：

#define CONFIG\_ENABLE\_IRAM\_MEMORY 1

#ifdef CONFIG\_ENABLE\_IRAM\_MEMORY uint32 user\_iram\_memory\_is\_enabled(void) { return CONFIG\_ENABLE\_IRAM\_MEMORY;

}

#endif

* 如上设置后，默认使⽤用 IRAM 作为内存，os\_malloc、os\_zalloc 和 os\_calloc 优先从 iRAM 分配，iRAM ⽤用尽后会继续使⽤用 dRAM 分配；
* 或者直接调⽤用 os\_malloc\_iram、os\_zalloc\_iram、os\_calloc\_iram 指定从 iRAM 分配内存，iRAM ⽤用尽后会继续使⽤用 dRAM 分配；直接调⽤用 os\_malloc\_dram、 os\_zalloc\_dram、os\_calloc\_dram 指定从 dRAM 分配内存；
* 如需与旧版本兼容，可使能宏 MEM\_DEFAULT\_USE\_DRAM，os\_malloc、os\_zalloc 和 os\_calloc 将从 dRAM 分配，⽽而 os\_malloc\_iram、os\_zalloc\_iram、 os\_calloc\_iram 可以指定从 iRAM 分配，iRAM ⽤用尽后会继续使⽤用 dRAM 分配。

例例如，在 makefile 中添加：

CONFIGURATION\_DEFINES += -DMEM\_DEFAULT\_USE\_DRAM

在 include/mem.h 中的具体定义如下：

#ifdef MEM\_DEFAULT\_USE\_DRAM

#define os\_malloc os\_malloc\_dram

#define os\_zalloc os\_zalloc\_dram

#define os\_calloc os\_calloc\_dram

#else

#define os\_malloc os\_malloc\_iram

#define os\_zalloc os\_zalloc\_iram

#define os\_calloc os\_calloc\_iram

#endif

• RAM 和 flash 访问必须是 4 字对⻬齐的，请勿直接进⾏行行指针转换。请使⽤用 os\_memcpy 或其他 API 进⾏行行内存操作。

## 3. 应⽤用程序接⼝口 (API)

### 3.1. 软件定时器器

以下软件定时器器接⼝口位于 */ESP8266\_NONOS\_SDK/include/osapi.h*。请注意，以下接⼝口使⽤用的定时器器由软件实现，定时器器的函数在任务中被执⾏行行。因为任务可能被中断，或者被其他⾼高优先级的任务延迟，因此以下 os\_timer 系列列的接⼝口并不不能保证定时器器精确执⾏行行。

如果需要精确的定时，例例如，周期性操作某 GPIO，请使⽤用硬件中断定时器器，具体可参考 *hw\_timer.c*，硬件定时器器的执⾏行行函数在中断⾥里里被执⾏行行。注意：

* 对于同⼀一个 timer，os\_timer\_arm 或 os\_timer\_arm\_us 不不能重复调⽤用，必须先 os\_timer\_disarm。
* os\_timer\_setfn 必须在 timer 未使能的情况下调⽤用，在 os\_timer\_arm 或 os\_timer\_arm\_us 之前或者 os\_timer\_disarm 之后。

3.1.1. os\_timer\_arm

|  |  |
| --- | --- |
| 功能 | 使能毫秒级定时器器 |
| 函数定义 | void os\_timer\_arm (  os\_timer\_t \*ptimer,   uint32\_t milliseconds,   bool repeat\_flag  ) |
| 参数 | * os\_timer\_t \*ptimer：定时器器结构 * uint32\_t milliseconds：定时时间，单位：ms   + 如未调⽤用 system\_timer\_reinit，可⽀支持范围 5 ~ 0x68D7A3   + 如调⽤用了了 system\_timer\_reinit，可⽀支持范围 100 ~ 0x689D0 * bool repeat\_flag：定时器器是否重复 |
| 返回 | ⽆无 |

3.1.2. os\_timer\_disarm

|  |  |
| --- | --- |
| 功能 | 取消定时器器定时 |
| 函数定义 | void os\_timer\_disarm (os\_timer\_t \*ptimer) |
| 参数 | os\_timer\_t \*ptimer：定时器器结构 |
| 返回 | ⽆无 |

3.1.3. os\_timer\_setfn

|  |  |
| --- | --- |
| 功能 | 设置定时器器回调函数。使⽤用定时器器，必须设置回调函数。 |
| 函数定义 | void os\_timer\_setfn(  os\_timer\_t \*ptimer,   os\_timer\_func\_t \*pfunction,   void \*parg  ) |
| 参数 | * os\_timer\_t \*ptimer：定时器器结构 * os\_timer\_func\_t \*pfunction：定时器器回调函数 * void \*parg：回调函数的参数 |
| 返回 | ⽆无 |

3.1.4. system\_timer\_reinit

|  |  |
| --- | --- |
| 功能 | 重新初始化定时器器，当需要使⽤用微秒级定时器器时调⽤用 |
| 注意 | * 同时定义 USE\_US\_TIMER * system\_timer\_reinit 在程序 开始调⽤用，user\_init 的第⼀一句句。 |
| 函数定义 | void system\_timer\_reinit (void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.1.5. os\_timer\_arm\_us

|  |  |
| --- | --- |
| 功能 | 使能微秒级定时器器。 |
| 注意 | * 请定义 USE\_US\_TIMER，并在 user\_init 起始第⼀一句句，先调⽤用 system\_timer\_reinit。 * ⾼高精度为 500 μs。 |
| 函数定义 | void os\_timer\_arm\_us (  os\_timer\_t \*ptimer,   uint32\_t microseconds,   bool repeat\_flag  ) |
| 参数 | * os\_timer\_t \*ptimer：定时器器结构 * uint32\_t microseconds：定时时间，单位：μs， ⼩小定时 0x64 ， ⼤大可输⼊入   0xFFFFFFF   * bool repeat\_flag：定时器器是否重复 |
| 返回 | ⽆无 |

### 3.2. 硬件中断定时器器

以下硬件中断定时器器接⼝口位于 */ESP8266\_NONOS\_SDK/examples/driver\_lib/ hw\_timer.c*。⽤用户可根据 *driver\_lib*⽂文件夹下的 *readme.txt*⽂文件使⽤用。注意：

* 如果使⽤用 NMI 中断源，且为⾃自动填装的定时器器，调⽤用 hw\_timer\_arm 时参数 val 必须⼤大于 100。
* 如果使⽤用 NMI 中断源，那么该定时器器将为 ⾼高优先级，可打断其他 ISR。
* 如果使⽤用 FRC1 中断源，那么该定时器器⽆无法打断其他 ISR。
* hw\_timer.c 的接⼝口不不能跟 PWM 驱动接⼝口函数同时使⽤用，因为⼆二者共⽤用了了同⼀一个硬件定时器器。
* 硬件中断定时器器的回调函数定义，请勿添加 ICACHE\_FLASH\_ATTR宏。
* 使⽤用 hw\_timer.c 的接⼝口，请勿调⽤用 wifi\_set\_sleep\_type(LIGHT\_SLEEP); 将⾃自动睡眠模式设置为 Light-sleep。因为 Light-sleep 在睡眠期间会停 CPU，停 CPU 期间不不能响应 NMI 中断。

3.2.1. hw\_timer\_init

|  |  |
| --- | --- |
| 功能 | 初始化硬件 ISR 定时器器 |
| 函数定义 | void hw\_timer\_init (  FRC1\_TIMER\_SOURCE\_TYPE source\_type,   u8 req ) |
| 参数 | * FRC1\_TIMER\_SOURCE\_TYPE source\_type：定时器器的 ISR 源   + FRC1\_SOURCE：使⽤用 FRC1 中断源   + NMI\_SOURCE：使⽤用 NMI 中断源 * u8 req   + 0：不不⾃自动填装；   + 1：⾃自动填装 |
| 返回 | ⽆无 |

3.2.2. hw\_timer\_arm

|  |  |
| --- | --- |
| 功能 | 使能硬件中断定时器器 |
| 函数定义 | void hw\_timer\_arm (uint32 val) |
| 参数 | uint32 val：定时时间⾃自动填装模式：   * 使⽤用 FRC1 中断源 FRC1\_SOURCE，取值范围：50 ~ 0x199999 μs； * 使⽤用 NMI 中断源 NMI\_SOURCE，取值范围 : 100 ~ 0x199999 μs：   ⾮非⾃自动填装模式，取值范围：10 ~ 0x199999 μs |
| 返回 | ⽆无 |

3.2.3. hw\_timer\_set\_func

|  |  |
| --- | --- |
| 功能 | 设置定时器器回调函数。使⽤用定时器器，必须设置回调函数。 |
| 注意 | 回调函数前不不能添加 ICACHE\_FLASH\_ATTR 宏定义，中断响应不不能存放在 Flash 中。 |
| 函数定义 | void hw\_timer\_set\_func (void (\* user\_hw\_timer\_cb\_set)(void) ) |
| 参数 | void (\* user\_hw\_timer\_cb\_set)(void)：定时器器回调函数，函数定义时请勿添加 ICACHE\_FLASH\_ATTR 宏。 |
| 返回 | ⽆无 |

3.2.4. 硬件定时器器示例例

|  |
| --- |
| #define REG\_READ(\_r) (\*(volatile uint32 \*)(\_r)) #define WDEV\_NOW() REG\_READ(0x3ff20c00) uint32 tick\_now2 = 0; void hw\_test\_timer\_cb(void) {  static uint16 j = 0;  j++;    if( (WDEV\_NOW() - tick\_now2) >= 1000000 )  {  static u32 idx = 1;  tick\_now2 = WDEV\_NOW();  os\_printf("b%u:%d\n",idx++,j);  j = 0;  } }  void ICACHE\_FLASH\_ATTR user\_init(void) {  hw\_timer\_init(FRC1\_SOURCE,1);  hw\_timer\_set\_func(hw\_test\_timer\_cb);  hw\_timer\_arm(100);  } |

### 3.3. 系统接⼝口

系统接⼝口位于 */ESP8266\_NONOS\_SDK/include/user\_interface.h*。 os\_XXX 系列列接⼝口位于 */ESP8266\_NONOS\_SDK/include/osapi.h*。

3.3.1. system\_get\_sdk\_version

|  |  |
| --- | --- |
| 功能 | 查询 SDK 版本信息 |
| 函数定义 | const char\* system\_get\_sdk\_version(void) |
| 参数 | ⽆无 |
| 返回 | SDK 版本信息 |
| 示例例 | printf(“SDK version: %s \n”, system\_get\_sdk\_version()); |

3.3.2. system\_restore

|  |  |
| --- | --- |
| 功能 | 恢复出⼚厂设置。本接⼝口将清除以下接⼝口的设置，恢复默认值：  wifi\_station\_set\_auto\_connect、wifi\_set\_phy\_mode、wifi\_softap\_set\_config 相关，wifi\_station\_set\_config 相关，wifi\_set\_opmode 以及 #define AP\_CACHE 记录的  AP 信息。 |
| 注意 | 恢复出⼚厂设置后，请务必重新启动 system\_restart，再正常使⽤用。 |
| 函数定义 | void system\_restore(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.3.3. system\_restart

|  |  |
| --- | --- |
| 功能 | 系统重启 |
| 注意 | 调⽤用本接⼝口后，ESP8266 模块并不不会⽴立刻重启，请勿在本接⼝口之后调⽤用其他功能接⼝口。 |
| 函数定义 | void system\_restart(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.3.4. system\_init\_done\_cb

|  |  |
| --- | --- |
| 功能 | 在 user\_init 中调⽤用，注册系统初始化完成的回调函数。 |
| 注意 | 接⼝口 wifi\_station\_scan 必须在系统初始化完成后，并且 Station 模式使能的情况下调  ⽤用。 |
| 函数定义 | void system\_init\_done\_cb(init\_done\_cb\_t cb) |
| 参数 | init\_done\_cb\_t cb：系统初始化完成的回调函数 |
| 返回 | ⽆无 |
| 示例例 | void to\_scan(void) { wifi\_station\_scan(NULL,scan\_done); }  void user\_init(void) {  wifi\_set\_opmode(STATION\_MODE);   system\_init\_done\_cb(to\_scan);  } |

3.3.5. system\_get\_chip\_id

|  |  |
| --- | --- |
| 功能 | 查询芯⽚片 ID |
| 函数定义 | uint32 system\_get\_chip\_id (void) |
| 参数 | ⽆无 |
| 返回 | 芯⽚片 ID |

3.3.6. system\_get\_vdd33

|  |  |
| --- | --- |
| 功能 | 测量量 VDD3P3 管脚 3 和 4 的电压值，单位：1/1024V |
| 注意 | * system\_get\_vdd33 必须在 TOUT 管脚悬空的情况下使⽤用。 * TOUT 管脚悬空的情况下，esp\_init\_data\_default.bin (0 ~ 127 byte) 中的第 107 byte 为 vdd33\_const，必须设为 0xFF，即 255。 * 不不同 Wi-Fi 模式下，例例如，Modem-sleep 模式或者普通 Wi-Fi ⼯工作模式时，VDD33 的测量量值会稍有差异。 |
| 函数定义 | uint16 system\_get\_vdd33(void) |
| 参数 | ⽆无 |
| 返回 | VDD33 电压值。单位：1/1024V |

3.3.7. system\_adc\_read

|  |  |
| --- | --- |
| 功能 | 测量量 TOUT 管脚 6 的输⼊入电压，单位：1/1024V |
| 注意 | * system\_adc\_read 必须在 TOUT 管脚接外部电路路情况下使⽤用，TOUT 管脚输⼊入电压范围限定为 0 ~ 1.0V。 * TOUT 管脚接外部电路路的情况下，esp\_init\_data\_default.bin (0 ~ 127 byte) 中的第 107 byte vdd33\_const，必须设为 VDD3P3 管脚 3 和 4 上真实的电源电压，且必须⼩小于 0xFF。 * 第 107 byte vdd33\_const 的单位是 0.1V，有效取值范围是 [18, 36]；当 vdd33\_const 处于⽆无效范围 [0, 18) 或者 (36, 255) 时，使⽤用默认值 3.3V 来优化 RF 电路路⼯工作状态。 * 不不同 Wi-Fi 模式下，例例如，Modem-sleep 模式或者普通 Wi-Fi ⼯工作模式时，ADC 的测量量值会稍有差异。 * 若需要⾼高精度的 ADC，请使⽤用 system\_adc\_read\_fast 接⼝口。 |
| 函数定义 | uint16 system\_adc\_read(void) |
| 参数 | ⽆无 |
| 返回 | TOUT 管脚 6 的输⼊入电压，单位：1/1024V |

3.3.8. system\_adc\_read\_fast

|  |  |  |
| --- | --- | --- |
| 功能 | 快速⾼高精度的 ADC 采样。 | |
| 注意 | * 本接⼝口必须在关闭 Wi-Fi 的状态下使⽤用。如需进⾏行行连续测量量 ADC，则还需要在关闭所有中断的状态下使⽤用。因此，调⽤用 system\_adc\_read\_fast 时，不不能使⽤用 PWM 或者 NMI 类型的硬件定时器器。 * 本接⼝口必须在 TOUT 管脚接外部电路路情况下使⽤用，TOUT 管脚输⼊入电压范围限定为 0 ~ 1.0V。 * TOUT 管脚接外部电路路作为 ADC 输⼊入时，esp\_init\_data\_default.bin (0 ~ 127 byte) 中的 [107] byte vdd33\_const 必须⼩小于 0xFF。 * [107] byte vdd33\_const 的具体⽤用法如下：   + [107] byte = 0XFF 时，内部测量量 VDD33，TOUT 管脚不不能作为外部 ADC 输⼊入；   + [107] byte 有效取值范围是 [18, 36] 时，单位是 0.1V，设置为实际的 VDD33 电源电压，优化 RF 电路路⼯工作状态，TOUT 管脚可以作为外部 ADC 输⼊入；   + [107] byte 有效取值范围是 [0, 18) 或者 (36, 255) 时，使⽤用默认值 3.3V 作为电源电压来优化 RF 电路路⼯工作状态，TOUT 管脚可以作为外部 ADC 输⼊入。 | |
| 函数定义 | void system\_adc\_read\_fast(uint16 \*adc\_addr, uint16 adc\_num, uint8 adc\_clk\_div) | |
| 参数 | * uint16 \*adc\_addr：ADC 连续采样输出的地址指针。 * uint16 adc\_num：ADC 连续采样的点数，输⼊入范围 [1, 65535]。 * uint8 adc\_clk\_div：ADC ⼯工作时钟 = 80M/adc\_clk\_div，输⼊入范围 [8, 32]，推荐值为 8。 | |
| 返回 | ⽆无 | |
| 示例例 | adc\_clk\_div);  {                                  } | extern void system\_adc\_read\_fast(uint16 \*adc\_addr, uint16 adc\_num, uint8 os\_timer\_t timer; void ICACHE\_FLASH\_ATTR ADC\_TEST(void \*p)  wifi\_set\_opmode(NULL\_MODE); ets\_intr\_lock(); //close interrupt  uint16 adc\_addr[10]; uint16 adc\_num = 10; uint8 adc\_clk\_div = 8; uint32 i; system\_adc\_read\_fast(adc\_addr, adc\_num, adc\_clk\_div);  for(i=0; i<adc\_num; i++) os\_printf("i=%d, adc\_v=%d\n", i, adc\_addr[i]);  ets\_intr\_unlock(); //open interrupt  os\_timer\_disarm(&timer); os\_timer\_setfn(&timer, ADC\_TEST, NULL); os\_timer\_arm(&timer,1000,1); |

3.3.9. system\_deep\_sleep

|  |  |
| --- | --- |
| 功能 | 设置芯⽚片进⼊入 Deep-sleep 模式，休眠设定时间后⾃自动唤醒，唤醒后程序从 user\_init 重新运⾏行行。 |
| 注意 | * 硬件需要将 XPD\_DCDC 通过 0Ω 电阻连接到 EXT\_RSTB，⽤用作 Deep-sleep 唤醒。 * system\_deep\_sleep(0) 未设置唤醒定时器器，可通过外部 GPIO 拉低 RST 脚唤醒。 * 本接⼝口设置后，芯⽚片并不不会⽴立刻进⼊入 Deep-sleep，⽽而是等待 Wi-Fi 底层功能安全关闭后，才进⼊入 Deep-sleep 休眠。 |
| 函数定义 | bool system\_deep\_sleep(uint64 time\_in\_us) |
| 参数 | uint64 time\_in\_us：休眠时间，单位：μs   * 参数 time\_in\_us 的理理论 ⼤大值可由公式 (time\_in\_us/cali)<<12 = 2^31 -1 计算。   - 其中 cali = system\_rtc\_clock\_cali\_proc()，表示 RTC 的时钟周期，bit11 ~ bit0 为⼩小数部分，受温度或电源电压变化⽽而偏移，并不不精确，详细可参考 system\_rtc\_clock\_cali\_proc 函数说明。   * 由于计算并不不精确，设置时传⼊入的 time\_in\_us 值需⼩小于理理论 ⼤大值。 |
| 返回 | True，设置成功 False，设置失败 |

3.3.10. system\_deep\_sleep\_set\_option

|  |  |
| --- | --- |
| 功能 | 设置下⼀一次 Deep-sleep 唤醒后的⾏行行为，如需调⽤用此 API，必须在 system\_deep\_sleep 之前调⽤用。默认 option 为 1。 |
| 函数定义 | bool system\_deep\_sleep\_set\_option(uint8 option) |
| 参数 | uint8 option：设置下⼀一次 Deep-sleep 唤醒后的⾏行行为。   * 0：由 esp\_init\_data\_default.bin (0~127 byte) 的 byte 108 和 Deep-sleep 的次数（deep\_sleep\_number，上电时初始化为 0）共同控制 Deep-sleep 唤醒后的⾏行行为，以每 (byte 108 + 1) 次 Deep-sleep 唤醒为周期循环。   + 若 deep\_sleep\_number <= byte 108，则 Deep-sleep 唤醒后不不进⾏行行任何 RF\_CAL，初始电流较⼩小；   + 若 deep\_sleep\_number = byte 108 + 1，则 Deep-sleep 唤醒后的⾏行行为与上电的⾏行行为⼀一致，且将 deep\_sleep\_number 归零； * 1：Deep-sleep 唤醒后的⾏行行为与上电的⾏行行为⼀一致； * 2：Deep-sleep 唤醒后不不进⾏行行 RF\_CAL，初始电流较⼩小； * 4：Deep-sleep 唤醒后不不打开 RF，与 Modem-sleep ⾏行行为⼀一致，这样电流 ⼩小，但是设备唤醒后⽆无法发送和接收数据。 |
| 返回 | true：成功 false：失败 |

3.3.11. system\_phy\_set\_rfoption

|  |  |
| --- | --- |
| 功能 | 设置此次 ESP8266 Deep-sleep 醒来，是否打开 RF。 |
| 注意 | * 本接⼝口只允许在 user\_rf\_pre\_init 中调⽤用。 * 本接⼝口与 system\_deep\_sleep\_set\_option 功能相似， system\_deep\_sleep\_set\_option 在 Deep-sleep 前调⽤用，本接⼝口在 Deep-sleep 醒来初始化时调⽤用，以本接⼝口设置为准。 * 调⽤用本接⼝口前，要求⾄至少调⽤用过⼀一次 system\_deep\_sleep\_set\_option。 |
| 函数定义 | void system\_phy\_set\_rfoption(uint8 option) |
| 参数 | uint8 option：设置下⼀一次 Deep-sleep 唤醒后的⾏行行为。   * 0：由 esp\_init\_data\_default.bin (0~127 byte) 的 byte 108 和 Deep-sleep 的次数（deep\_sleep\_number，上电时初始化为 0）共同控制 Deep-sleep 唤醒后的⾏行行为，以每 (byte 108 + 1) 次 Deep-sleep 唤醒为周期循环。   + 若 deep\_sleep\_number <= byte 108，则 Deep-sleep 唤醒后不不进⾏行行任何 RF\_CAL，初始电流较⼩小；   + 若 deep\_sleep\_number = byte 108 + 1，则 Deep-sleep 唤醒后的⾏行行为与上电的⾏行行为⼀一致，且将 deep\_sleep\_number 归零； * 1：Deep-sleep 唤醒后的⾏行行为与上电的⾏行行为⼀一致； * 2：Deep-sleep 唤醒后不不进⾏行行 RF\_CAL，初始电流较⼩小； * 4：Deep-sleep 唤醒后不不打开 RF，与 Modem-sleep ⾏行行为⼀一致，这样电流 ⼩小，但是设备唤醒后⽆无法发送和接收数据。 |
| 返回 | ⽆无 |

3.3.12. system\_phy\_set\_powerup\_option

|  |  |
| --- | --- |
| 功能 | 设置上电时 RF 初始化的⾏行行为，默认为 option 0。 |
| 函数定义 | void system\_phy\_set\_powerup\_option(uint8 option) |
| 参数 | uint8 option：power up 时，RF 初始化的⾏行行为   * 0：由 esp\_init\_data\_default.bin (0 ~ 127 byte) byte 114 控制 RF 初始化⾏行行为，详细可参考 [ESP8266 SDK ⼊入⻔门指南](http://www.espressif.com/sites/default/files/documentation/2a-esp8266-sdk_getting_started_guide_cn.pdf)。 * 1：RF 初始化仅做 VDD33 和 TX power CAL，耗时约 18 ms，初始电流较⼩小。 * 2：RF 初始化仅做 VDD33 校准，耗时约 2 ms，初始电流 ⼩小。 * 3：RF 初始化进⾏行行全部 RF CAL，耗时约 200 ms，初始电流较⼤大。 |
| 返回 | ⽆无 |

3.3.13. system\_phy\_set\_max\_tpw

|  |  |
| --- | --- |
| 功能 | 设置 RF TX Power ⼤大值，单位：0.25 dBm |
| 函数定义 | void system\_phy\_set\_max\_tpw(uint8 max\_tpw) |
| 参数 | uint8 max\_tpw：RF Tx Power 的 ⼤大值，可参考 esp\_init\_data\_default.bin (0 ~ 127 byte) 的第 34 byte target\_power\_qdb\_0 设置，单位：0.25 dBm，参数范围 [0, 82] |
| 返回 | ⽆无 |

3.3.14. system\_phy\_set\_tpw\_via\_vdd33

|  |  |
| --- | --- |
| 功能 | 根据改变的 VDD33 电压值，重新调整 RF TX Power，单位：1/1024V |
| 注意 | * 在 TOUT 管脚悬空的情况下，VDD33 电压值可通过 system\_get\_vdd33 测量量获得。 * 在 TOUT 管脚接外部电路路情况下，不不可使⽤用 system\_get\_vdd33 测量量 VDD33 电压值。 |
| 函数定义 | void system\_phy\_set\_tpw\_via\_vdd33(uint16 vdd33) |
| 参数 | uint16 vdd33：重新测量量的 VDD33 值，单位：1/1024V，有效值范围：[1900, 3300] |
| 返回 | ⽆无 |

3.3.15. system\_set\_os\_print

|  |  |
| --- | --- |
| 功能 | 开关打印 log 功能 |
| 函数定义 | void system\_set\_os\_print (uint8 onoff) |
| 参数 | uint8 onoff   * 0：打印功能关； * 1：打印功能开 |
| 默认值 | 打印功能开 |
| 返回 | ⽆无 |

3.3.16. system\_print\_meminfo

|  |  |
| --- | --- |
| 功能 | 打印系统内存空间分配，打印信息包括 data/rodata/bss/heap |
| 函数定义 | void system\_print\_meminfo (void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.3.17. system\_get\_free\_heap\_size

|  |  |
| --- | --- |
| 功能 | 查询系统剩余可⽤用 heap 区空间⼤大⼩小 |
| 函数定义 | uint32 system\_get\_free\_heap\_size(void) |
| 参数 | ⽆无 |
| 返回 | uint32：可⽤用 heap 空间⼤大⼩小 |

3.3.18. system\_os\_task

|  |  |
| --- | --- |
| 功能 | 创建系统任务， 多⽀支持创建 3 个任务，优先级分别为 0/1/2 |
| 函数定义 | bool system\_os\_task(  os\_task\_t task,   uint8 prio,   os\_event\_t \*queue,   uint8 qlen  ) |
| 参数 | * os\_task\_t task：任务函数 * uint8 prio：任务优先级，可为 0/1/2；0 为 低优先级。这表示 多只⽀支持建⽴立 3 个任务 * os\_event\_t \*queue：消息队列列指针 * uint8 qlen：消息队列列深度 |
| 返回 | true：成功 false：失败 |
| 示例例 | #define SIG\_RX 0 #define TEST\_QUEUE\_LEN 4 os\_event\_t \*testQueue;    void test\_task (os\_event\_t \*e) {  switch (e->sig) {  case SIG\_RX:  os\_printf(sig\_rx %c/n, (char)e->par);  break;  default:  break;  } }    void task\_init(void) {  testQueue=(os\_event\_t \*)os\_malloc(sizeof(os\_event\_t)\*TEST\_QUEUE\_LEN);  system\_os\_task(test\_task,USER\_TASK\_PRIO\_0,testQueue,TEST\_QUEUE\_LEN);  } |

3.3.19. system\_os\_post

|  |  |
| --- | --- |
| 功能 | 向任务发送消息 |
| 函数定义 | bool system\_os\_post (  uint8 prio,   os\_signal\_t sig,   os\_param\_t par  ) |
| 参数 | * uint8 prio：任务优先级，与建⽴立时的任务优先级对应。 * os\_signal\_t sig：消息类型 * os\_param\_t par：消息参数 |
| 返回 | true：成功 false：失败 |
| 结合上⼀一节的示例例 | void task\_post(void) {  system\_os\_post(USER\_TASK\_PRIO\_0, SIG\_RX, ‘a’); } |
| 打印输出 | sig\_rx a |

3.3.20. system\_get\_time

|  |  |
| --- | --- |
| 功能 | 查询系统时间，单位：μs |
| 函数定义 | uint32 system\_get\_time(void) |
| 参数 | ⽆无 |
| 返回 | 系统时间，单位：μs。 |

3.3.21. system\_get\_rtc\_time

|  |  |
| --- | --- |
| 功能 | 查询 RTC 时间，单位：RTC 时钟周期 |
| 示例例 | 例例如 system\_get\_rtc\_time 返回 10（表示 10 个 RTC 周期）， system\_rtc\_clock\_cali\_proc 返回 5.75（表示 1 个 RTC 周期为 5.75 μs），则实际时间为 10 x 5.75 = 57.5 μs。 |
| 注意 | system\_restart 时，系统时间归零，但是 RTC 时间仍然继续。但是如果外部硬件通过  EXT\_RST 脚或者 CHIP\_EN 脚，将芯⽚片复位后（包括 Deep-sleep 定时唤醒的情况），  RTC 时钟会复位。具体如下：   * 外部复位 EXT\_RST：RTC memory 不不变，RTC timer 寄存器器从零计数 * 看⻔门狗复位：RTC memory 不不变，RTC timer 寄存器器不不变 * system\_restart：RTC memory 不不变，RTC timer 寄存器器不不变 * 电源上电：RTC memory 随机值，RTC timer 寄存器器从零计数 * CHIP\_EN 复位：RTC memory 随机值，RTC timer 寄存器器从零计数 |
| 函数定义 | uint32 system\_get\_rtc\_time(void) |
| 参数 | ⽆无 |
| 返回 | RTC 时间 |

3.3.22. system\_rtc\_clock\_cali\_proc

|  |  |
| --- | --- |
| 功能 | 查询 RTC 时钟周期 |
| 注意 | * RTC 时钟周期含有⼩小数部分。 * RTC 时钟周期会随温度或电源电压变化发⽣生偏移，因此 RTC 时钟适⽤用于在精度可接受的范围内进⾏行行计时，建议 多每分钟调⽤用⼀一次即可。 |
| 函数定义 | uint32 system\_rtc\_clock\_cali\_proc(void) |
| 参数 | ⽆无 |
| 返回 | RTC 时钟周期，单位：μs，bit11 ~ bit0 为⼩小数部分 |
| 示例例 | os\_printf(“clk cal : %d \r\n”,system\_rtc\_clock\_cali\_proc()>>12); 详细 RTC 示例例请⻅见附录。 |

3.3.23. system\_rtc\_mem\_write

|  |  |
| --- | --- |
| 功能 | 由于 Deep-sleep 时，仅 RTC 仍在⼯工作，⽤用户如有需要，可将数据存⼊入 RTC memory 中。提供如下图中的 user data 段共 512 bytes 供⽤用户存储数据。  |<--------system data--------->|<-----------------user data--------------->|  | 256 bytes | 512 bytes | |
| 注意 | RTC memory 只能 4 字节整存整取，函数中参数 des\_addr 为 block number，每 block 4 字节，因此若写⼊入上图 user data 区起始位置，des\_addr 为 256/4 = 64，save\_size 为存⼊入数据的字节数。 |
| 函数定义 | bool system\_rtc\_mem\_write (  uint32 des\_addr,   void \* src\_addr,   uint32 save\_size  ) |
| 参数 | * uint32 des\_addr：写⼊入 rtc memory 的位置，des\_addr >=64 * void \* src\_addr：数据指针 * uint32 save\_size：数据⻓长度，单位：字节 |
| 返回 | true：成功 false：失败 |

3.3.24. system\_rtc\_mem\_read

|  |  |
| --- | --- |
| 功能 | 读取 RTC memory 中的数据，提供如下图中 user data 段共 512 bytes 给⽤用户存储数据。  |<--------system data--------->|<-----------------user data--------------->|  | 256 bytes | 512 bytes | |
| 注意 | RTC memory 只能 4 字节整存整取，函数中参数 des\_addr 为 block number，每 block 4 字节，因此若写⼊入上图 user data 区起始位置，des\_addr 为 256/4 = 64，save\_size 为存⼊入数据的字节数。 |
| 函数定义 | bool system\_rtc\_mem\_read (  uint32 src\_addr,   void \* des\_addr,   uint32 save\_size  ) |
| 参数 | * uint32 des\_addr：写⼊入 rtc memory 的位置，des\_addr >=64 * void \* src\_addr：数据指针 * uint32 save\_size：数据⻓长度，单位：字节 |
| 返回 | true：成功 false：失败 |

3.3.25. system\_uart\_swap

|  |  |
| --- | --- |
| 功能 | UART0 转换。将 MTCK 作为 UART0 RX，MTDO 作为 UART0 TX。硬件上也从 MTDO  (U0RTS) 和 MTCK (U0CTS) 连出 UART0，从⽽而避免上电时从 UART0 打印出 ROM log。 |
| 函数定义 | void system\_uart\_swap (void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.3.26. system\_uart\_de\_swap

|  |  |
| --- | --- |
| 功能 | 取消 UART0 转换，仍然使⽤用原有 UART0，⽽而不不是将 MTCK、MTDO 作为 UART0。 |
| 函数定义 | void system\_uart\_de\_swap (void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.3.27. system\_get\_boot\_version

|  |  |
| --- | --- |
| 功能 | 读取 boot 版本信息 |
| 函数定义 | uint8 system\_get\_boot\_version (void) |
| 参数 | ⽆无 |
| 返回 | boot 版本信息 |
| 注意 | 如果 boot 版本号 >= 3 时，⽀支持 boot 增强模式，详⻅见 system\_restart\_enhance。 |

3.3.28. system\_get\_userbin\_addr

|  |  |
| --- | --- |
| 功能 | 读取当前正在运⾏行行的 user bin（*user1.bin* 或者 *user2.bin*）的存放地址。 |
| 函数定义 | uint32 system\_get\_userbin\_addr (void) |
| 参数 | ⽆无 |
| 返回 | 正在运⾏行行的 user bin 的存放地址 |

3.3.29. system\_get\_boot\_mode

|  |  |
| --- | --- |
| 功能 | 查询 boot 模式 |
| 函数定义 | uint8 system\_get\_boot\_mode (void) |
| 参数 | ⽆无 |
| 返回 | #define SYS\_BOOT\_ENHANCE\_MODE 0  #define SYS\_BOOT\_NORMAL\_MODE 1 |
| 注意 | boot 增强模式：⽀支持跳转到任意位置运⾏行行程序； boot 普通模式：仅能跳转到固定的 *user1.bin*（或 *user2.bin*）位置运⾏行行。 |

3.3.30. system\_restart\_enhance

|  |  |
| --- | --- |
| 功能 | 重启系统，进⼊入 boot 增强模式。 |
| 函数定义 | bool system\_restart\_enhance(  uint8 bin\_type,   uint32 bin\_addr ) |
| 参数 | * uint8 bin\_type：bin 类型   + #define SYS\_BOOT\_NORMAL\_BIN 0 // user1.bin 或者 user2.bin   + #define SYS\_BOOT\_TEST\_BIN 1 // 向乐鑫申请的 test bin * uint32 bin\_addr：bin 的起始地址 |
| 返回 | true：成功 false：失败 |
| 注意 | *SYS\_BOOT\_TEST\_BIN* ⽤用于量量产测试，⽤用户可以向乐鑫申请获得。 |

3.3.31. system\_update\_cpu\_req

|  |  |
| --- | --- |
| 功能 | 设置 CPU 频率。默认为 80 MHz |
| 注意 | 系统总线时钟频率始终为 80 MHz，不不受 CPU 频率切换的影响。UART、SPI 等外设频率由系统总线时钟分频⽽而来，因此也不不受 CPU 频率切换的影响。 |
| 函数定义 | bool system\_update\_cpu\_freq(uint8 freq) |
| 参数 | uint8 freq：CPU 频率  #define SYS\_CPU\_80MHz 80  #define SYS\_CPU\_160MHz 160 |
| 返回 | true：成功 false：失败 |

3.3.32. system\_get\_cpu\_freq

|  |  |
| --- | --- |
| 功能 | 查询 CPU 频率 |
| 函数定义 | uint8 system\_get\_cpu\_freq(void) |
| 参数 | ⽆无 |
| 返回 | CPU 频率，单位：MHz |

3.3.33. system\_get\_flash\_size\_map

|  |  |
| --- | --- |
| 功能 | 查询当前的 Flash size 和 Flash map  Flash map 对应编译时的选项，详细介绍请参考 [ESP8266 SDK ⼊入⻔门指南。](http://www.espressif.com/sites/default/files/documentation/2a-esp8266-sdk_getting_started_guide_cn.pdf) |
| 结构体 | enum flash\_size\_map {  FLASH\_SIZE\_4M\_MAP\_256\_256 = 0,  FLASH\_SIZE\_2M,  FLASH\_SIZE\_8M\_MAP\_512\_512,  FLASH\_SIZE\_16M\_MAP\_512\_512,  FLASH\_SIZE\_32M\_MAP\_512\_512,  FLASH\_SIZE\_16M\_MAP\_1024\_1024,  FLASH\_SIZE\_32M\_MAP\_1024\_1024,  FLASH\_SIZE\_64M\_MAP\_1024\_1024,  FLASH\_SIZE\_128M\_MAP\_1024\_1024,  }; |
| 函数定义 | enum flash\_size\_map system\_get\_flash\_size\_map(void) |
| 参数 | ⽆无 |
| 返回 | flash map |

3.3.34. system\_get\_rst\_info

|  |  |
| --- | --- |
| 功能 | 查询当前启动的信息 |
| 结构体 | enum rst\_reason { REANSON\_DEFAULT\_RST = 0, // normal startup by power on  REANSON\_WDT\_RST = 1, // hardware watch dog reset  // exception reset, GPIO status won’t change  REANSON\_EXCEPTION\_RST = 2,  // software watch dog reset, GPIO status won’t change  REANSON\_SOFT\_WDT\_RST = 3,  // software restart ,system\_restart , GPIO status won’t change  REANSON\_SOFT\_RESTART = 4,  REANSON\_DEEP\_SLEEP\_AWAKE = 5, // wake up from deep-sleep  REANSON\_EXT\_SYS\_RST = 6, // external system reset  };  struct rst\_info { uint32 reason; // enum rst\_reason uint32 exccause;  uint32 epc1; // the address that error occurred  uint32 epc2; uint32 epc3; uint32 excvaddr; uint32 depc;  }; |
| 函数定义 | struct rst\_info\* system\_get\_rst\_info(void) |
| 参数 | ⽆无 |
| 返回 | 启动的信息 |

3.3.35. system\_soft\_wdt\_stop

|  |  |
| --- | --- |
| 功能 | 关闭软件看⻔门狗 |
| 注意 | 请勿将软件看⻔门狗关闭太⻓长时间（⼩小于 5s），否则将触发硬件看⻔门狗复位 |
| 函数定义 | void system\_soft\_wdt\_stop(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.3.36. system\_soft\_wdt\_restart

|  |  |
| --- | --- |
| 功能 | 重启软件看⻔门狗 |
| 注意 | 仅⽀支持在软件看⻔门狗关闭 system\_soft\_wdt\_stop 的情况下，调⽤用本接⼝口 |
| 函数定义 | void system\_soft\_wdt\_restart(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.3.37. system\_soft\_wdt\_feed

|  |  |
| --- | --- |
| 功能 | 喂软件看⻔门狗 |
| 注意 | 仅⽀支持在软件看⻔门狗开启的情况下，调⽤用本接⼝口 |
| 函数定义 | void system\_soft\_wdt\_feed(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.3.38. system\_show\_malloc

|  |  |
| --- | --- |
| 功能 | 打印⽬目前所分配的堆空间所有内存块，包括分配该内存块的⽂文件名、⾏行行号和分配⼤大⼩小。在怀疑有内存泄露露时，可以调⽤用本接⼝口查看当前内存状态。 |
| 注意 | * 在 *user\_config.h* 定义 #define MEMLEAK\_DEBUG。参考   *ESP8266\_NONOS\_SDK\included\mem.h* 开始位置的注释使⽤用。   * 泄露露的内存⼀一般在打印结果中，但打印结果中的内存不不保证⼀一定是泄露露的内存。 * 本接⼝口仅⽤用于调试，⽆无法确保使⽤用本接⼝口后，程序能继续正常执⾏行行，因此请勿在正常运⾏行行情况下，调⽤用本接⼝口。 |
| 函数定义 | void system\_show\_malloc(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.3.39. os\_memset

|  |  |
| --- | --- |
| 功能 | 封装 C 语⾔言函数，在⼀一段内存块中填充某个给定值。 |
| 函数定义 | os\_memset(void \*s, int ch, size\_t n) |
| 参数 | * void \*s：内存块指针 * int ch：填充值 * size\_t n：填充⼤大⼩小 |
| 返回 | ⽆无 |
| 示例例 | uint8 buffer[32];  os\_memset(buffer, 0, sizeof(buffer)); |

3.3.40. os\_memcpy

|  |  |
| --- | --- |
| 功能 | 封装 C 语⾔言函数，内存拷⻉贝。 |
| 函数定义 | os\_memcpy(void \*des, void \*src, size\_t n) |
| 参数 | * void \*des：⽬目标内存块指针 * void \*src：源内存块指针 * size\_t n：拷⻉贝内存⼤大⼩小 |
| 返回 | ⽆无 |
| 示例例 | uint8 buffer[4] = {0}; os\_memcpy(buffer, “abcd”, 4); |

3.3.41. os\_strlen

|  |  |
| --- | --- |
| 功能 | 封装 C 语⾔言函数，计算字符串串⻓长度。 |
| 函数定义 | os\_strlen(char \*s) |
| 参数 | char \*s：字符串串 |
| 返回 | 字符串串⻓长度 |
| 示例例 | char \*ssid = “ESP8266”;  os\_memcpy(softAP\_config.ssid, ssid, os\_strlen(ssid)); |

3.3.42. os\_printf

|  |  |
| --- | --- |
| 功能 | 格式化输出，打印字符串串。 |
| 注意 | * 本接⼝口默认从 UART 0 打印。IOT\_Demo 中的 uart\_init 可以设置波特率，将 os\_printf 改为从 UART 1 打印：os\_install\_putc1((void \*)uart1\_write\_char); * 请勿调⽤用本接⼝口打印超过 125 字节的数据，或者频繁连续调⽤用本接⼝口打印，否则可能会丢失部分待打印数据。 |
| 函数定义 | void os\_printf(const char \*s) |
| 参数 | const char \*s：字符串串 |
| 返回 | ⽆无 |
| 示例例 | os\_printf(“SDK version: %s \n”, system\_get\_sdk\_version()); |

3.3.43. os\_bzero

|  |  |
| --- | --- |
| 功能 | 置字符串串 p 的前 n 个字节为零且包含 \0 |
| 函数定义 | void os\_bzero(void \*p, size\_t n) |
| 参数 | * void \*p：要置零的数据的起始地址 * size\_t n：要置零的数据字节数 |
| 返回 | ⽆无 |

3.3.44. os\_delay\_us

|  |  |
| --- | --- |
| 功能 | 延时函数。 ⼤大值 65535 μs |
| 函数定义 | void os\_delay\_us(uint16 us) |
| 参数 | uint16 us：延时时间 |
| 返回 | ⽆无 |

3.3.45. os\_install\_putc1

|  |  |
| --- | --- |
| 功能 | 注册打印接⼝口函数 |
| 函数定义 | void os\_install\_putc1(void(\*p)(char c)) |
| 参数 | void(\*p)(char c)：打印接⼝口函数指针 |
| 返回 | ⽆无 |
| 示例例 | 参考 *UART.c*，uart\_init 中的 os\_install\_putc1((void \*)uart1\_write\_char) 将 os\_printf 改为从 UART 1 打印。否则，os\_printf 默认从 UART 0 打印。 |

|  |  |
| --- | --- |
| 功能 | ⽤用户⾃自定义 RF\_CAL 参数存放在 Flash 的扇区号 |

3.3.46. os\_random

|  |  |
| --- | --- |
| 功能 | 获取随机数 |
| 函数定义 | unsigned long os\_random(void) |
| 参数 | ⽆无 |
| 返回 | 随机数 |

3.3.47. os\_get\_random

|  |  |
| --- | --- |
| 功能 | 获取指定⻓长度的随机数 |
| 函数定义 | int os\_get\_random(unsigned char \*buf, size\_t len) |
| 参数 | * unsigned char \*buf：获得的随机数 * size\_t len：随机数的字节⻓长度 |
| 返回 | true：成功 false：失败 |
| 示例例 | int ret = os\_get\_random((unsigned char \*)temp, 7); os\_printf("ret %d, value 0x%08x%08x\n\r", ret, temp[1], temp[0]); |

3.3.48. user\_rf\_cal\_sector\_set

|  |  |
| --- | --- |
| 注意 | * ⽤用户必须在程序中实现此函数，否则编译链接时会报错。但⽤用户程序⽆无需调⽤用此函数，SDK 底层会调⽤用它，将 RF\_CAL 参数保存在⽤用户指定的 Flash 扇区⾥里里，这将占⽤用⽤用户参数区的⼀一个扇区。 * SDK 预留留的 4 个扇区的系统参数区已经使⽤用，因此 RF\_CAL 参数需要占⽤用到⽤用户参数区的空间，由⽤用户通过此函数设置⼀一个可⽤用扇区供 SDK 底层使⽤用。 * 建议整个系统需要初始化时，或需要重新进⾏行行 RF\_CAL 时，烧录 *blank.bin* 初始化 RF\_CAL 参数区，并烧录 *esp\_init\_data.bin*。注意，*esp\_init\_data.bin* ⾄至少需要烧录⼀一次。 |
| 函数定义 | uint32 user\_rf\_cal\_sector\_set(void) |
| 参数 | ⽆无 |
| 返回 | 存储 RF\_CAL 参数的 Flash 扇区号 |
| 示例例 | 将 RF 参数设置存放在 Flash 倒数第 5 个扇区  uint32 user\_rf\_cal\_sector\_set(void)  {  enum flash\_size\_map size\_map = system\_get\_flash\_size\_map(); uint32 rf\_cal\_sec = 0;  switch (size\_map) {  case FLASH\_SIZE\_4M\_MAP\_256\_256: rf\_cal\_sec = 128 - 5; break;  case FLASH\_SIZE\_8M\_MAP\_512\_512: rf\_cal\_sec = 256 - 5; break;  case FLASH\_SIZE\_16M\_MAP\_512\_512: case FLASH\_SIZE\_16M\_MAP\_1024\_1024:  rf\_cal\_sec = 512 - 5; break;  case FLASH\_SIZE\_32M\_MAP\_512\_512: case FLASH\_SIZE\_32M\_MAP\_1024\_1024:  rf\_cal\_sec = 512 - 5; break;  case FLASH\_SIZE\_64M\_MAP\_1024\_1024:  rf\_cal\_sec = 2048 - 5; break;  case FLASH\_SIZE\_128M\_MAP\_1024\_1024:  rf\_cal\_sec = 4096 - 5; break;  default:  rf\_cal\_sec = 0;  break;  }  return rf\_cal\_sec;  } |

3.3.49. system\_deep\_sleep\_instant

|  |  |
| --- | --- |
| 功能 | 设置芯⽚片⽴立刻进⼊入 Deep-sleep 模式，休眠设定时间后⾃自动唤醒，唤醒后程序从 user\_init 重新运⾏行行。 |
| 注意 | * 硬件需要将 XPD\_DCDC 通过 0Ω 电阻连接到 EXT\_RSTB，⽤用作 Deep-sleep 唤醒。 * system\_deep\_sleep\_instant(0) 未设置唤醒定时器器，可通过外部 GPIO 拉低 RST 脚唤醒。 * 本接⼝口设置后，芯⽚片⽴立刻进⼊入 Deep-sleep 休眠，不不会等待 Wi-Fi 底层功能安全关闭。如需等待 Wi-Fi 安全关闭，可使⽤用接⼝口 system\_deep\_sleep。 |
| 函数定义 | bool system\_deep\_sleep\_instant(uint64 time\_in\_us) |
| 参数 | uint64 time\_in\_us：休眠时间，单位：μs   * 参数 time\_in\_us 的理理论 ⼤大值可由公式 (time\_in\_us/cali)<<12 = 2^32 -1 计算。   - 其中 cali = system\_rtc\_clock\_cali\_proc()，表示 RTC 的时钟周期，bit11 ~ bit0 为⼩小数部分，受温度或电源电压变化⽽而偏移，并不不精确，详细可参考 system\_rtc\_clock\_cali\_proc 函数说明。   * 由于计算并不不精确，设置时传⼊入的 time\_in\_us 值需⼩小于理理论 ⼤大值。 |
| 返回 | True，设置成功 False，设置失败 |

3.3.50. system\_partition\_table\_regist

|  |  |
| --- | --- |
| 功能 | 注册 partition table。 |
| 注意 | * 本接⼝口必须在 user\_pre\_init 中调⽤用注册，如果注册失败，请检查 partition table 的定义。 * 示例例可参考 ESP8266\_NONOS\_SDK/examples/IoT\_Demo/user/user\_main.c。 |
| 函数定义 | bool system\_partition\_table\_regist( const partition\_item\_t\* partition\_table, uint32\_t partition\_num, uint32\_t map  ) |
| 参数 | const partition\_item\_t\* partition\_table：分区表 uint32\_t partition\_num：分区数⽬目  uint32\_t map：flash map；必须与编译烧录时选择的 flash map ⼀一致，否则将会启动异常；建议直接传⼊入宏 SPI\_FLASH\_SIZE\_MAP，它是系统在编译时记录的 flash map 值。 |
| 返回 | True，partition table 注册成功  False，partition table 注册失败 |

3.3.51. system\_partition\_get\_ota\_partition\_size

|  |  |
| --- | --- |
| 功能 | 查询 ota partition 的⼤大⼩小。 |
| 注意 | ota partition 是⽤用于存放 user1.bin 或者 user2.bin 的 flash 分区。 |
| 函数定义 | uint32\_t system\_partition\_get\_ota\_partition\_size(void) |
| 参数 | - |
| 返回 | ota partition 的⼤大⼩小 |

3.3.52. system\_partition\_get\_item

|  |  |
| --- | --- |
| 功能 | 查询指定类型的 partition 信息。 |
| 函数定义 | bool system\_partition\_get\_item(partition\_type\_t type, partition\_item\_t\*  partition\_item) |
| 参数 | partition\_type\_t type：分区类型 partition\_item\_t\* partition\_item：查询到的分区信息 |
| 返回 | True，查询成功 False，查询失败 |

#### 3.4. SPI Flash 接⼝口

SPI Flash 接⼝口位于 */ESP8266\_NONOS\_SDK/include/spi\_flash.h*。

system\_param\_xxx 接⼝口位于 */ESP8266\_NONOS\_SDK/include/user\_interface.h*。

|  |  |
| --- | --- |
| 功能 | 写⼊入数据到 Flash。Flash 读写必须 4 字节对⻬齐。 |

[3.4.1.](https://www.espressif.com/sites/default/files/documentation/99a-esp8266_flash_rw_operation_cn_v1.0.pdf) [spi\_flash\_get\_id](https://www.espressif.com/sites/default/files/documentation/99a-esp8266_flash_rw_operation_cn_v1.0.pdf)

|  |  |
| --- | --- |
| 功能 | 查询 SPI Flash 的 ID |
| 函数定义 | uint32 spi\_flash\_get\_id (void) |
| 参数 | ⽆无 |
| 返回 | spi flash id |

3.4.2. spi\_flash\_erase\_sector

|  |  |
| --- | --- |
| 功能 | 擦除 Flash 扇区 |
| 函数定义 | SpiFlashOpResult spi\_flash\_erase\_sector (uint16 sec) |
| 参数 | uint16 sec：扇区号，从扇区 0 开始计数，每扇区 4 KB |
| 返回 | typedef enum{  SPI\_FLASH\_RESULT\_OK,  SPI\_FLASH\_RESULT\_ERR,  SPI\_FLASH\_RESULT\_TIMEOUT  } SpiFlashOpResult; |

3.4.3. spi\_flash\_write

|  |  |
| --- | --- |
| 函数定义 | SpiFlashOpResult spi\_flash\_write (  uint32 des\_addr,   uint32 \*src\_addr,   uint32 size  ) |
| 参数 | * uint32 des\_addr：写⼊入 Flash ⽬目的地址 * uint32 \*src\_addr：写⼊入数据的指针 * uint32 size：数据⻓长度，单位 byte，必须 4 字节对⻬齐进⾏行行读写 |
| 返回 | typedef enum{  SPI\_FLASH\_RESULT\_OK,  SPI\_FLASH\_RESULT\_ERR,  SPI\_FLASH\_RESULT\_TIMEOUT  } SpiFlashOpResult; |

3.4.4. spi\_flash\_read

|  |  |
| --- | --- |
| 功能 | 从 Flash 读取数据。Flash 读写必须 4 字节对⻬齐。 |
| 函数定义 | SpiFlashOpResult spi\_flash\_read(  uint32 src\_addr,   uint32 \* des\_addr,   uint32 size  ) |
| 参数 | * uint32 des\_addr：写⼊入 Flash ⽬目的地址 * uint32 \*des\_addr： 存放读取到数据的指针 * uint32 size：数据⻓长度，单位 byte，必须 4 字节对⻬齐进⾏行行读写 |
| 返回 | typedef enum {  SPI\_FLASH\_RESULT\_OK,  SPI\_FLASH\_RESULT\_ERR,  SPI\_FLASH\_RESULT\_TIMEOUT  } SpiFlashOpResult; |
| 示例例 | uint32 value;  uint8 \*addr = (uint8 \*)&value;  spi\_flash\_read(0x3E \* SPI\_FLASH\_SEC\_SIZE, (uint32 \*)addr, 4); os\_printf("0x3E sec:%02x%02x%02x%02x\r\n", addr[0], addr[1], addr[2], addr[3]); |

3.4.5. system\_param\_save\_with\_protect

|  |  |
| --- | --- |
| 功能 | 使⽤用带读写保护机制的⽅方式，写⼊入数据到 Flash。Flash 读写必须 4 字节对⻬齐。  Flash 读写保护机制：使⽤用 3 个 sector（4 KB 每 sector）保存 1 个 sector 的数据，sector 0 和 sector 1 互相为备份，交替保存数据，sector 2 作为 flag sector，指示 新的数据保存在 sector 0 还是 sector 1。 |
| 函数定义 | bool system\_param\_save\_with\_protect (  uint16 start\_sec,   void \*param,   uint16 len  ) |
| 参数 | * uint16 start\_sec：读写保护机制使⽤用的 3 个 sector 的起始 sector 0 值。 例例如，IOT\_Demo 中可使⽤用 0x3D000 开始的 3 个 sector（3×4 KB）建⽴立读写保护机制，则参数 start\_sec 传 0x3D。 * void \*param：写⼊入数据的指针 * uint16 len：数据⻓长度，不不能超过 1 个 sector ⼤大⼩小，即 4×1024 |
| 返回 | true：成功 false：失败 |

3.4.6. system\_param\_load

|  |  |
| --- | --- |
| 功能 | 使⽤用带读写保护机制的⽅方式，写⼊入数据到 Flash。Flash 读写必须 4 字节对⻬齐。  Flash 读写保护机制：使⽤用 3 个 sector（4 KB 每 sector）保存 1 个 sector 的数据，sector 0 和 sector 1 互相为备份，交替保存数据，sector 2 作为 flag sector，指示 新的数据保存在 sector 0 还是 sector 1。 |
| 函数定义 | bool system\_param\_load (  uint16 start\_sec, uint16 offset, void \*param,  uint16 len  ) |
| 参数 | * uint16 start\_sec：读写保护机制使⽤用的 3 个 sector 的起始 sector 0 值。 例例如，IOT\_Demo 中可使⽤用 0x3D000 开始的 3 个 sector（3×4 KB）建⽴立读写保护机制，则参数 start\_sec 传 0x3D，请勿传⼊入 0x3E 或者 0x3F。 * uint16 offset：需读取数据，在 sector 中的偏移地址 * void \*param：读取数据的指针 * uint16 len：数据⻓长度，不不能超过 1 个 sector ⼤大⼩小，即 offset+len ≤ 4\*1024 |
| 返回 | true：成功 false：失败 |

3.4.7. spi\_flash\_set\_read\_func

|  |  |
| --- | --- |
| 功能 | 注册⽤用户⾃自定义的 SPI Flash 读取接⼝口函数 |
| 注意 | 仅⽀支持在 SPI overlap 模式下使⽤用，请⽤用户参考  *ESP8266\_NONOS\_SDK\driver\_lib\driver\spi\_overlap.c* |
| 函数定义 | void spi\_flash\_set\_read\_func (user\_spi\_flash\_read read) |
| 参数 | typedef SpiFlashOpResult (\*user\_spi\_flash\_read)(  SpiFlashChip \*spi,  uint32 src\_addr,   uint32 \* des\_addr,   uint32 size  ) |
| 返回 | ⽆无 |

3.4.8. spi\_flash\_erase\_protect\_enable

|  |  |
| --- | --- |
| 功能 | 使能 flash 擦写保护。使能后，将保护 flash 不不会误操作擦写了了正在运⾏行行的应⽤用程序。 |
| 函数定义 | bool spi\_flash\_erase\_protect\_enable(void) |
| 参数 | ⽆无 |
| 返回 | True: 设置成功  False: 设置失败 |

3.4.9. spi\_flash\_erase\_protect\_disable

|  |  |
| --- | --- |
| 功能 | 关闭 flash 擦写保护功能。 |
| 函数定义 | bool spi\_flash\_erase\_protect\_disable(void) |
| 参数 | ⽆无 |
| 返回 | True: 设置成功  False: 设置失败 |

#### 3.5. Wi-Fi 接⼝口

Wi-Fi 接⼝口位于 */ESP8266\_NONOS\_SDK/include/user\_interface.h*。

wifi\_station\_xxx 系列列接⼝口以及 ESP8266 Station 相关的设置、查询接⼝口，请在 ESP8266 Station 使能的情况下调⽤用；

wifi\_softap\_xxx 系列列接⼝口以及 ESP8266 SoftAP 相关的设置、查询接⼝口，请在 ESP8266 SoftAP 使能的情况下调⽤用。

ESP8266 station ⽀支持的认证类型有：OPEN，WEP，WPAPSK，WPA2PSK；⽀支持的加密⽅方式有：AUTO，TKIP，AES，WEP。

ESP8266 softAP ⽀支持的认证类型有：OPEN，WPAPSK，WPA2PSK；⽀支持的加密⽅方式有：AUTO，TKIP，AES；但 group key 加密⽅方式，只⽀支持 TKIP，不不⽀支持 AES。

后⽂文的“Flash 系统参数区”位于 Flash 的 后 16 KB。

3.5.1. wifi\_get\_opmode

|  |  |
| --- | --- |
| 功能 | 查询 Wi-Fi 当前⼯工作模式 |
| 函数定义 | uint8 wifi\_get\_opmode (void) |
| 参数 | ⽆无 |
| 返回 | Wi-Fi ⼯工作模式：   * 0x01：Station 模式 * 0x02：SoftAP 模式 * 0x03：Station+SoftAP 模式 |

3.5.2. wifi\_get\_opmode\_default

|  |  |
| --- | --- |
| 功能 | 查询保存在 Flash 中的 Wi-Fi ⼯工作模式设置 |
| 函数定义 | uint8 wifi\_get\_opmode\_default (void) |
| 参数 | ⽆无 |
| 返回 | Wi-Fi ⼯工作模式：   * 0x01：Station 模式 * 0x02：SoftAP 模式 * 0x03：Station+SoftAP 模式 |

3.5.3. wifi\_set\_opmode

|  |  |
| --- | --- |
| 功能 | 设置 Wi-Fi ⼯工作模式（Station，SoftAP 或者 Station+SoftAP），并保存到 Flash。  默认为 SoftAP 模式 |
| 注意 | * ESP8266\_NONOS\_SDK\_V0.9.2 以及之前版本，设置之后需要调⽤用 system\_restart() 重启⽣生效； * ESP8266\_NONOS\_SDK\_V0.9.2 之后的版本，不不需要重启，即时⽣生效。 * 本设置如果与原设置不不同，会更更新保存到 Flash 系统参数区。 |
| 函数定义 | bool wifi\_set\_opmode (uint8 opmode) |
| 参数 | uint8 opmode：Wi-Fi ⼯工作模式   * 0x01：Station 模式 * 0x02：SoftAP 模式 * 0x03：Station+SoftAP 模式 |
| 返回 | true：成功 false：失败 |

3.5.4. wifi\_set\_opmode\_current

|  |  |
| --- | --- |
| 功能 | 设置 Wi-Fi ⼯工作模式（Station，SoftAP 或者 Station + SoftAP），不不保存到 Flash。 |
| 函数定义 | bool wifi\_set\_opmode\_current (uint8 opmode) |
| 参数 | uint8 opmode：Wi-Fi ⼯工作模式   * 0x01：Station 模式 * 0x02：SoftAP 模式 * 0x03：Station+SoftAP 模式 |
| 返回 | true：成功 false：失败 |

3.5.5. wifi\_station\_get\_config

|  |  |
| --- | --- |
| 功能 | 查询 Wi-Fi Station 接⼝口的当前配置参数。 |
| 函数定义 | bool wifi\_station\_get\_config (struct station\_config \*config) |
| 参数 | struct station\_config \*config：Wi-Fi Station 接⼝口参数指针 |
| 返回 | true：成功 false：失败 |

3.5.6. wifi\_station\_get\_config\_default

|  |  |
| --- | --- |
| 功能 | 查询 Wi-Fi Station 接⼝口保存在 Flash 中的配置参数。 |
| 函数定义 | bool wifi\_station\_get\_config\_default (struct station\_config \*config) |
| 参数 | struct station\_config \*config：Wi-Fi Station 接⼝口参数指针 |
| 返回 | true：成功 false：失败 |

3.5.7. wifi\_station\_set\_config

|  |  |
| --- | --- |
| 功能 | 设置 Wi-Fi Station 接⼝口的配置参数，并保存到 Flash |
| 注意 | * 请在 ESP8266 Station 使能的情况下，调⽤用本接⼝口。 * 如果 wifi\_station\_set\_config 在 user\_init 中调⽤用，则 ESP8266 Station 接⼝口会在系统初始化完成后，⾃自动连接 AP（路路由），⽆无需再调⽤用 wifi\_station\_connect * 否则，需要调⽤用 wifi\_station\_connect 连接 AP（路路由）。 * station\_config.bssid\_set ⼀一般设置为 0 ，仅当需要检查 AP 的 MAC 地址时（多⽤用于有重名 AP 的情况下）设置为 1。 * 本设置如果与原设置不不同，会更更新保存到 Flash 系统参数区。 |
| 函数定义 | bool wifi\_station\_set\_config (struct station\_config \*config) |
| 参数 | struct station\_config \*config：Wi-Fi Station 接⼝口配置参数指针 |
| 返回 | true：成功 false：失败 |
| 示例例 | void ICACHE\_FLASH\_ATTR user\_set\_station\_config(void)  {  char ssid[32] = SSID;  char password[64] = PASSWORD;  struct station\_config stationConf;  stationConf.bssid\_set = 0; //need not check MAC address of AP  os\_memcpy(&stationConf.ssid, ssid, 32);  os\_memcpy(&stationConf.password, password, 64);  wifi\_station\_set\_config(&stationConf); } void user\_init(void)  {  wifi\_set\_opmode(STATIONAP\_MODE); //Set softAP + station mode user\_set\_station\_config();  } |

3.5.8. wifi\_station\_set\_config\_current

|  |  |
| --- | --- |
| 功能 | 设置 Wi-Fi Station 接⼝口的配置参数，不不保存到 Flash |
| 注意 | * 请在 ESP8266 Station 使能的情况下，调⽤用本接⼝口。 * 如果 wifi\_station\_set\_config 在 user\_init 中调⽤用，则 ESP8266 Station 接⼝口会在系统初始化完成后，⾃自动连接 AP（路路由），⽆无需再调⽤用 wifi\_station\_connect * 否则，需要调⽤用 wifi\_station\_connect 连接 AP（路路由）。 * station\_config.bssid\_set ⼀一般设置为 0 ，仅当需要检查 AP 的 MAC 地址时（多⽤用于有重名 AP 的情况下）设置为 1。 * 本设置如果与原设置不不同，会更更新保存到 Flash 系统参数区。 |
| 函数定义 | bool wifi\_station\_set\_config\_current (struct station\_config \*config) |
| 参数 | struct station\_config \*config：Wi-Fi Station 接⼝口配置参数指针 |
| 返回 | true：成功 false：失败 |

3.5.9. wifi\_station\_set\_cert\_key

|  |  |
| --- | --- |
| 功能 | 不不建议使⽤用本接⼝口，请使⽤用 wifi\_station\_set\_enterprise\_cert\_key 代替。设置 ESP8266 Wi-Fi Station 接⼝口连接 WPA2-ENTERPRISE AP 使⽤用的证书。 |
| 注意 | * ⽀支持 WPA2-ENTERPRISE AP 需占⽤用 26 KB 以上的内存，调⽤用本接⼝口时请注意内存是否⾜足够。 * ⽬目前 WPA2-ENTERPRISE 只⽀支持⾮非加密的私钥⽂文件和证书⽂文件，且仅⽀支持 PEM 格式   + ⽀支持的证书⽂文件头信息为：- - - - - BEGIN CERTIFICATE - - - - -   + ⽀支持的私钥⽂文件头信息为：- - - - - BEGIN RSA PRIVATE KEY - - - - -  或者 - - - - - BEGIN PRIVATE KEY - - - - - * 请在连接 WPA2-ENTERPRISE AP 之前调⽤用本接⼝口设置私钥⽂文件和证书⽂文件，在成功连接   AP 后先调⽤用 wifi\_station\_clear\_cert\_key 清除内部状态，应⽤用层再释放私钥⽂文件和证书⽂文件信息。   * 如果遇到加密的私钥⽂文件，请使⽤用 openssl pkey 命令改为⾮非加密⽂文件使⽤用，或者使⽤用 openssl rsa 等命令，对某些私钥⽂文件进⾏行行加密-⾮非加密的转换（或起始 TAG 转化）。 |
| 函数定义 | bool wifi\_station\_set\_cert\_key (  uint8 \*client\_cert, int client\_cert\_len,   uint8 \*private\_key, int private\_key\_len,  uint8 \*private\_key\_passwd, int private\_key\_passwd\_len,) |
| 参数 | uint8 \*client\_cert：⼗十六进制数组的证书指针  int client\_cert\_len：证书⻓长度  uint8 \*private\_key：⼗十六进制数组的私钥指针，暂不不⽀支持超过 2048 的私钥 int private\_key\_len：私钥⻓长度，请勿超过 2048  uint8 \*private\_key\_passwd：私钥的提取密码，⽬目前暂不不⽀支持，请传⼊入 NULL int private\_key\_passwd\_len：提取密码的⻓长度，⽬目前暂不不⽀支持，请传⼊入 0 |
| 返回 | 0：成功  ⾮非 0：失败 |
| 示例例 | 假设私钥⽂文件的信息为 - - - - - BEGIN PRIVATE KEY - - - - - … … … …  那么对应的数组为：uint8 key[]={0x2d, 0x2d, 0x2d, 0x2d, 0x2d, 0x42, 0x45, 0x47, … …  0x00 }; 即各字符的 ASCII 码，请注意，数组必须添加 0x00 作为结尾。 |

3.5.10. wifi\_station\_clear\_cert\_key

|  |  |
| --- | --- |
| 功能 | 不不建议使⽤用本接⼝口，请使⽤用 wifi\_station\_clear\_enterprise\_cert\_key 代替。释放连接 WPA2-ENTERPRISE AP 使⽤用证书占⽤用的资源，并清除相关状态。 |
| 函数定义 | void wifi\_station\_clear\_cert\_key (void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.5.11. wifi\_station\_set\_username

|  |  |
| --- | --- |
| 功能 | 不不建议使⽤用本接⼝口，请使⽤用 wifi\_station\_set\_enterprise\_username 代替。设置连接 WPA2-ENTERPRISE AP 时，ESP8266 Station 的⽤用户名。 |
| 函数定义 | int wifi\_station\_set\_username (uint8 \*username, int len) |
| 参数 | uint8 \*username：⽤用户名称 int len：名称⻓长度 |
| 返回 | 0：成功  其他：失败 |

3.5.12. wifi\_station\_clear\_username

|  |  |
| --- | --- |
| 功能 | 不不建议使⽤用本接⼝口，请使⽤用 wifi\_station\_clear\_enterprise\_username 代替。释放连接 WPA2-ENTERPRISE AP 设置⽤用户名占⽤用的资源，并清除相关状态。 |
| 函数定义 | void wifi\_station\_clear\_username (void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.5.13. wifi\_station\_connect

|  |  |
| --- | --- |
| 功能 | ESP8266 Wi-Fi Station 接⼝口连接 AP |
| 注意 | 请勿在 user\_init 中调⽤用本接⼝口，请在 ESP8266 Station 使能并初始化完成后调⽤用；如果 ESP8266 已经连接某个 AP，请先调⽤用 wifi\_station\_disconnect 断开上⼀一次连接。 |
| 函数定义 | bool wifi\_station\_connect (void) |
| 参数 | ⽆无 |
| 返回 | true：成功 false：失败 |

3.5.14. wifi\_station\_disconnect

|  |  |
| --- | --- |
| 功能 | ESP8266 Wi-Fi Station 接⼝口从 AP 断开连接 |
| 注意 | 请勿在 user\_init 中调⽤用本接⼝口，本接⼝口必须在系统初始化完成后，并且 ESP8266 Station 接  ⼝口使能的情况下调⽤用。 |
| 函数定义 | bool wifi\_station\_disconnect (void) |
| 参数 | ⽆无 |
| 返回 | true：成功 false：失败 |

3.5.15. wifi\_station\_get\_connect\_status

|  |  |
| --- | --- |
| 功能 | 查询 ESP8266 Wi-Fi Station 接⼝口连接 AP 的状态。 |
| 注意 | 若为特殊应⽤用场景：调⽤用 wifi\_station\_set\_reconnect\_policy 关闭重连功能，且未调⽤用 wifi\_set\_event\_handler\_cb 注册 Wi-Fi 事件回调，则本接⼝口失效，⽆无法准确获得连接状态。 |
| 函数定义 | uint8 wifi\_station\_get\_connect\_status (void) |
| 参数 | ⽆无 |
| 返回 | enum{  STATION\_IDLE = 0,  STATION\_CONNECTING,  STATION\_WRONG\_PASSWORD,  STATION\_NO\_AP\_FOUND,  STATION\_CONNECT\_FAIL,  STATION\_GOT\_IP  }; |

3.5.16. wifi\_station\_scan

|  |  |
| --- | --- |
| 功能 | 获取 AP 的信息 |
| 注意 | 请勿在 user\_init 中调⽤用本接⼝口，本接⼝口必须在系统初始化完成后，并且 ESP8266 Station 接  ⼝口使能的情况下调⽤用。 |
| 函数定义 | bool wifi\_station\_scan (struct scan\_config \*config, scan\_done\_cb\_t cb); |
| 结构体 | struct scan\_config {  uint8 \*ssid; // AP’s ssid  uint8 \*bssid; // AP’s bssid  uint8 channel; //scan a specific channel  uint8 show\_hidden; //scan APs of which ssid is hidden.  wifi\_scan\_type\_t scan\_type; // scan type, active or passive  wifi\_scan\_time\_t scan\_time; // scan time per channel  }; |
| 参数 | * struct scan\_config \*config：扫描 AP 的配置参数   + 若 config==null：扫描获取所有可⽤用 AP 的信息   + 若 config.ssid==null && config.bssid==null && config.channel!=null：ESP8266   Station 接⼝口扫描获取特定信道上的 AP 信息。   * + 若 config.ssid!=null && config.bssid==null && config.channel==null：ESP8266   Station 接⼝口扫描获取所有信道上的某特定名称 AP 的信息。   * scan\_done\_cb\_t cb：扫描完成的 callback |
| 返回 | true：成功 false：失败 |

3.5.17. scan\_done\_cb\_t

|  |  |
| --- | --- |
| 功能 | wifi\_station\_scan 的回调函数 |
| 注意 | 请勿在 user\_init 中调⽤用本接⼝口，本接⼝口必须在系统初始化完成后，并且 ESP8266 Station 接  ⼝口使能的情况下调⽤用。 |
| 函数定义 | void scan\_done\_cb\_t (void \*arg, STATUS status) |
| 参数 | * void \*arg：扫描获取到的 AP 信息指针，以链表形式存储，数据结构 struct bss\_info * STATUS status：扫描结果 |
| 返回 | ⽆无 |
| 示例例 | wifi\_station\_scan(&config, scan\_done);  static void ICACHE\_FLASH\_ATTR scan\_done(void \*arg, STATUS status) {  if (status == OK) {  struct bss\_info \*bss\_link = (struct bss\_info \*)arg;  ...  } } |

3.5.18. wifi\_station\_ap\_number\_set

|  |  |
| --- | --- |
| 功能 | 设置 ESP8266 Station 多可记录⼏几个 AP 的信息。  ESP8266 Station 成功连⼊入⼀一个 AP 时，可以保存 AP 的 SSID 和 password 记录。本设置如果与原设置不不同，会更更新保存到 Flash 系统参数区。 |
| 函数定义 | bool wifi\_station\_ap\_number\_set (uint8 ap\_number) |
| 参数 | uint8 ap\_number：记录 AP 信息的 ⼤大数⽬目（ ⼤大值为 5） |
| 返回 | true：成功 false：失败 |

3.5.19. wifi\_station\_get\_ap\_info

|  |  |
| --- | --- |
| 功能 | 获取 ESP8266 Station 保存的 AP 信息， 多记录 5 个。 |
| 函数定义 | uint8 wifi\_station\_get\_ap\_info(struct station\_config config[]) |
| 参数 | struct station\_config config[]：AP 的信息，数组⼤大⼩小必须为 5 |
| 返回 | 记录 AP 的数⽬目 |
| 示例例 | struct station\_config config[5]; int i = wifi\_station\_get\_ap\_info(config); |

3.5.20. wifi\_station\_ap\_change

|  |  |
| --- | --- |
| 功能 | ESP8266 Station 切换到已记录的某号 AP 配置连接 |
| 函数定义 | bool wifi\_station\_ap\_change (uint8 new\_ap\_id) |
| 参数 | uint8 new\_ap\_id：AP 记录的 ID 值，从 0 开始计数 |
| 返回 | true：成功 false：失败 |

3.5.21. wifi\_station\_get\_current\_ap\_id

|  |  |
| --- | --- |
| 功能 | 获取当前连接的 AP 保存记录 ID 值。ESP8266 可记录每⼀一个配置连接的 AP，从 0 开始计数。 |
| 函数定义 | uint8 wifi\_station\_get\_current\_ap\_id (); |
| 参数 | ⽆无 |
| 返回 | 当前连接的 AP 保存记录的 ID 值。 |

3.5.22. wifi\_station\_get\_auto\_connect

|  |  |
| --- | --- |
| 功能 | 查询 ESP8266 Station 上电是否会⾃自动连接已记录的 AP（路路由）。 |
| 函数定义 | uint8 wifi\_station\_get\_auto\_connect(void) |
| 参数 | ⽆无 |
| 返回 | 0：不不⾃自动连接 AP  ⾮非 0：⾃自动连接 AP。 |

3.5.23. wifi\_station\_set\_auto\_connect

|  |  |
| --- | --- |
| 功能 | 设置 ESP8266 Station 上电是否⾃自动连接已记录的 AP（路路由），默认为⾃自动连接。 |
| 注意 | * 本接⼝口如果在 user\_init 中调⽤用，则当前这次上电就⽣生效；如果在其他地⽅方调⽤用，则下⼀一次上电⽣生效。 * 本设置如果与原设置不不同，会更更新保存到 Flash 系统参数区。 |
| 函数定义 | bool wifi\_station\_set\_auto\_connect(uint8 set) |
| 参数 | uint8 set：上电是否⾃自动连接 AP   * 0：不不⾃自动连接 AP * 1：⾃自动连接 AP |
| 返回 | true：成功 false：失败 |

3.5.24. wifi\_station\_dhcpc\_start

|  |  |
| --- | --- |
| 功能 | 开启 ESP8266 Station DHCP client |
| 注意 | * DHCP 默认开启。 * DHCP 与静态 IP 功能 wifi\_set\_ip\_info 互相影响，以 后设置的为准： DHCP 开启，则静态 IP 失效；设置静态 IP，则关闭 DHCP。 |
| 函数定义 | bool wifi\_station\_dhcpc\_start(void) |
| 参数 | ⽆无 |
| 返回 | true：成功 false：失败 |

3.5.25. wifi\_station\_dhcpc\_stop

|  |  |
| --- | --- |
| 功能 | 关闭 ESP8266 Station DHCP client |
| 注意 | * DHCP 默认开启。 * DHCP 与静态 IP 功能 wifi\_set\_ip\_info 互相影响：   DHCP 开启，则静态 IP 失效；设置静态 IP，则 DHCP 关闭。 |
| 函数定义 | bool wifi\_station\_dhcpc\_stop(void) |
| 参数 | ⽆无 |
| 返回 | true：成功 false：失败 |

3.5.26. wifi\_station\_dhcpc\_status

|  |  |
| --- | --- |
| 功能 | 查询 ESP8266 Station DHCP client 状态 |
| 函数定义 | enum dhcp\_status wifi\_station\_dhcpc\_status(void) |
| 参数 | ⽆无 |
| 返回 | enum dhcp\_status {  DHCP\_STOPPED,  DHCP\_STARTED  }; |

3.5.27. wifi\_station\_dhcpc\_set\_maxtry

|  |  |
| --- | --- |
| 功能 | 设置 ESP8266 Station DHCP client ⼤大重连次数。默认会⼀一直重连。 |
| 函数定义 | bool wifi\_station\_dhcpc\_set\_maxtry(uint8 num) |
| 参数 | uint8 num： ⼤大重连次数 |
| 返回 | true：成功 false：失败 |

3.5.28. wifi\_station\_set\_reconnect\_policy

|  |  |
| --- | --- |
| 功能 | 设置 ESP8266 Station 连接 AP 失败或断开后是否重连。默认重连。 |
| 注意 | 建议在 user\_init 中调⽤用本接⼝口 |
| 函数定义 | bool wifi\_station\_set\_reconnect\_policy(bool set) |
| 参数 | bool set   * true：断开则重连 * false：断开不不重连 |
| 返回 | true：成功 false：失败 |

3.5.29. wifi\_station\_get\_rssi

|  |  |
| --- | --- |
| 功能 | 关闭 ESP8266 Station 已连接的 AP 信号强度 |
| 函数定义 | sint8 wifi\_station\_get\_rssi(void) |
| 参数 | ⽆无 |
| 返回 | <10：查询成功，返回信号强度  31：查询失败，返回错误码 |

3.5.30. wifi\_station\_set\_hostname

|  |  |
| --- | --- |
| 功能 | 设置 ESP8266 Station DHCP 分配的主机名称。 |
| 函数定义 | bool wifi\_station\_set\_hostname(char\* hostname) |
| 参数 | char\* hostname：主机名称， ⻓长 32 个字符。 |
| 返回 | true：成功 false：失败 |

3.5.31. wifi\_station\_get\_hostname

|  |  |
| --- | --- |
| 功能 | 查询 ESP8266 Station DHCP 分配的主机名称 |
| 函数定义 | char\* wifi\_station\_get\_hostname(void) |
| 参数 | ⽆无 |
| 返回 | 主机名称 |

3.5.32. wifi\_softap\_get\_config

|  |  |
| --- | --- |
| 功能 | 查询 ESP8266 Wi-Fi SoftAP 接⼝口的当前配置 |
| 函数定义 | bool wifi\_softap\_get\_config(struct softap\_config \*config) |
| 参数 | struct softap\_config \*config：ESP8266 SoftAP 配置参数 |
| 返回 | true：成功 false：失败 |

3.5.33. wifi\_softap\_get\_config\_default

|  |  |
| --- | --- |
| 功能 | 查询 ESP8266 Wi-Fi SoftAP 接⼝口保存在 Flash 中的配置 |
| 函数定义 | bool wifi\_softap\_get\_config\_default(struct softap\_config \*config) |
| 参数 | struct softap\_config \*config：ESP8266 SoftAP 配置参数 |
| 返回 | true：成功 false：失败 |

3.5.34. wifi\_softap\_set\_config

|  |  |
| --- | --- |
| 功能 | 设置 Wi-Fi SoftAP 接⼝口配置，并保存到 Flash |
| 注意 | * 请在 ESP8266 SoftAP 使能的情况下，调⽤用本接⼝口。 * 本设置如果与原设置不不同，将更更新保存到 Flash 系统参数区。 * 因为 ESP8266 只有⼀一个信道，因此 SoftAP+Station 共存模式时，ESP8266 SoftAP 接⼝口会   ⾃自动调节信道与 ESP8266 Station ⼀一致，详细说明请参考附录。 |
| 函数定义 | bool wifi\_softap\_set\_config (struct softap\_config \*config) |
| 参数 | struct softap\_config \*config：ESP8266 Wi-Fi SoftAP 配置参数 |
| 返回 | true：成功 false：失败 |

3.5.35. wifi\_softap\_set\_config\_current

|  |  |
| --- | --- |
| 功能 | 设置 Wi-Fi SoftAP 接⼝口配置，不不保存到 Flash |
| 注意 | * 请在 ESP8266 SoftAP 使能的情况下，调⽤用本接⼝口。 * 因为 ESP8266 只有⼀一个信道，因此 SoftAP+Station 共存模式时，ESP8266 SoftAP 接⼝口会⾃自动调节信道与 ESP8266 Station ⼀一致，详细说明请参考附录。 |
| 函数定义 | bool wifi\_softap\_set\_config\_current (struct softap\_config \*config) |
| 参数 | struct softap\_config \*config：ESP8266 Wi-Fi SoftAP 配置参数 |
| 返回 | true：成功 false：失败 |

3.5.36. wifi\_softap\_get\_station\_num

|  |  |
| --- | --- |
| 功能 | 获取 ESP8266 SoftAP 下连接的 Station 个数 |
| 函数定义 | uint8 wifi\_softap\_get\_station\_num(void) |
| 参数 | ⽆无 |
| 返回 | ESP8266 SoftAP 下连接的 Station 个数 |

3.5.37. wifi\_softap\_get\_station\_info

|  |  |
| --- | --- |
| 功能 | 获取 ESP8266 SoftAP 接⼝口下连⼊入的 Station 的信息，包括 MAC 和 IP |
| 注意 | 本接⼝口基于 DHCP 实现，因此不不⽀支持静态 IP 或者其他没有重新 DHCP 的情况。 |
| 函数定义 | struct station\_info \* wifi\_softap\_get\_station\_info(void) |
| 参数 | ⽆无 |
| 返回 | struct station\_info\*：Station 信息的结构体 |

3.5.38. wifi\_softap\_free\_station\_info

|  |  |
| --- | --- |
| 功能 | 释放调⽤用 wifi\_softap\_get\_station\_info 时结构体 station\_info 占⽤用的空间 |
| 函数定义 | void wifi\_softap\_free\_station\_info(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |
| 示例例 | 获取 MAC 和 IP 信息示例例，注意释放资源： |
| 示例例 1 | struct station\_info \* station = wifi\_softap\_get\_station\_info(); struct station\_info \* next\_station; while(station) {  os\_printf(bssid : MACSTR, ip : IPSTR/n,  MAC2STR(station->bssid), IP2STR(&station->ip));  next\_station = STAILQ\_NEXT(station, next);  os\_free(station); // Free it directly  station = next\_station; } |
| 示例例 2 | struct station\_info \* station = wifi\_softap\_get\_station\_info(); while(station){  os\_printf(bssid : MACSTR, ip : IPSTR/n,  MAC2STR(station->bssid), IP2STR(&station->ip));  station = STAILQ\_NEXT(station, next);  }  wifi\_softap\_free\_station\_info(); // Free it by calling functions |

3.5.39. wifi\_softap\_dhcps\_start

|  |  |
| --- | --- |
| 功能 | 开启 ESP8266 SoftAP DHCP server |
| 注意 | * DHCP 默认开启。 * DHCP 与静态 IP 功能 wifi\_set\_ip\_info 互相影响，以 后设置的为准： DHCP 开启，则静态 IP 失效；设置静态 IP，则关闭 DHCP。 |
| 函数定义 | bool wifi\_softap\_dhcps\_start(void) |
| 参数 | ⽆无 |
| 返回 | true：成功 false：失败 |

3.5.40. wifi\_softap\_dhcps\_stop

|  |  |
| --- | --- |
| 功能 | 关闭 ESP8266 SoftAP DHCP server。默认开启 DHCP。 |
| 函数定义 | bool wifi\_softap\_dhcps\_stop(void) |
| 参数 | ⽆无 |
| 返回 | true：成功 false：失败 |

3.5.41. wifi\_softap\_set\_dhcps\_lease

|  |  |  |
| --- | --- | --- |
| 功能 | 设置 ESP8266 SoftAP DHCP server 分配 IP 地址的范围 | |
| 注意 | * 设置的 IP 分配范围必须与 ESP8266 SoftAP IP 在同⼀一⽹网段。 * 本接⼝口必须在 ESP8266 SoftAP DHCP server 关闭 wifi\_softap\_dhcps\_stop 的情况下设置。 * 本设置仅对下⼀一次使能的 DHCP server ⽣生效 wifi\_softap\_dhcps\_start，如果 DHCP server 再次被关闭，则需要重新调⽤用本接⼝口设置 IP 范围；否则之后 DHCP server 重新使能，会使   ⽤用默认的 IP 地址分配范围。 | |
| 函数定义 | bool wifi\_softap\_set\_dhcps\_lease(struct dhcps\_lease \*please) | |
| 参数 | struct dhcps\_lease {  struct ip\_addr start\_ip;  struct ip\_addr end\_ip; }; | |
| 返回 | true：成功 false：失败 | |
| 示例例 | 或者 | void dhcps\_lease\_test(void) { struct dhcps\_lease dhcp\_lease; const char\* start\_ip = “192.168.5.100”; const char\* end\_ip = “192.168.5.105”; dhcp\_lease.start\_ip.addr = ipaddr\_addr(start\_ip); dhcp\_lease.end\_ip.addr = ipaddr\_addr(end\_ip); wifi\_softap\_set\_dhcps\_lease(&dhcp\_lease);  }  void dhcps\_lease\_test(void) {  struct dhcps\_lease dhcp\_lease;  IP4\_ADDR(&dhcp\_lease.start\_ip, 192, 168, 5, 100); IP4\_ADDR(&dhcp\_lease.end\_ip, 192, 168, 5, 105); wifi\_softap\_set\_dhcps\_lease(&dhcp\_lease);  } void user\_init(void) {  struct ip\_info info; wifi\_set\_opmode(STATIONAP\_MODE); //Set softAP + station mode wifi\_softap\_dhcps\_stop();  IP4\_ADDR(&info.ip, 192, 168, 5, 1);  IP4\_ADDR(&info.gw, 192, 168, 5, 1);  IP4\_ADDR(&info.netmask, 255, 255, 255, 0);  wifi\_set\_ip\_info(SOFTAP\_IF, &info); dhcps\_lease\_test(); wifi\_softap\_dhcps\_start();  } |

3.5.42. wifi\_softap\_get\_dhcps\_lease

|  |  |
| --- | --- |
| 功能 | 查询 ESP8266 SoftAP DHCP server 分配 IP 地址的范围 |
| 注意 | 本接⼝口仅⽀支持在 ESP8266 SoftAP DHCP server 使能的情况下查询。 |
| 函数定义 | bool wifi\_softap\_get\_dhcps\_lease(struct dhcps\_lease \*please) |
| 返回 | true：成功 false：失败 |

3.5.43. wifi\_softap\_set\_dhcps\_lease\_time

|  |  |
| --- | --- |
| 功能 | 设置 ESP8266 SoftAP DHCP server 的租约时间。默认为 120 分钟。 |
| 注意 | 本接⼝口仅⽀支持在 ESP8266 SoftAP DHCP server 使能的情况下查询。 |
| 函数定义 | bool wifi\_softap\_set\_dhcps\_lease\_time(uint32 minute) |
| 参数 | uint32 minute：租约时间，单位：分钟，取值范围：[1，2880] |
| 返回 | true：成功 false：失败 |

3.5.44. wifi\_softap\_get\_dhcps\_lease\_time

|  |  |
| --- | --- |
| 功能 | 查询 ESP8266 SoftAP DHCP server 的租约时间。 |
| 注意 | 本接⼝口仅⽀支持在 ESP8266 SoftAP DHCP server 使能的情况下查询。 |
| 函数定义 | uint32 wifi\_softap\_get\_dhcps\_lease\_time(void) |
| 返回 | 租约时间，单位：分钟 |

3.5.45. wifi\_softap\_reset\_dhcps\_lease\_time

|  |  |
| --- | --- |
| 功能 | 复位 ESP8266 SoftAP DHCP server 的租约时间。恢复到 120 分钟。 |
| 注意 | 本接⼝口仅⽀支持在 ESP8266 SoftAP DHCP server 使能的情况下查询。 |
| 函数定义 | bool wifi\_softap\_reset\_dhcps\_lease\_time(void) |
| 返回 | true：成功 false：失败 |

3.5.46. wifi\_softap\_dhcps\_status

|  |  |
| --- | --- |
| 功能 | 获取 ESP8266 SoftAP DHCP server 状态 |
| 函数定义 | enum dhcp\_status wifi\_softap\_dhcps\_status(void) |
| 参数 | ⽆无 |
| 返回 | enum dhcp\_status {  DHCP\_STOPPED,  DHCP\_STARTED  }; |

3.5.47. wifi\_softap\_set\_dhcps\_offer\_option

|  |  |
| --- | --- |
| 功能 | 设置 ESP8266 SoftAP DHCP server 属性 |
| 结构体 | enum dhcps\_offer\_option{  OFFER\_START = 0x00,  OFFER\_ROUTER = 0x01,  OFFER\_END  }; |
| 函数定义 | bool wifi\_softap\_set\_dhcps\_offer\_option(uint8 level, void\* optarg) |
| 参数 | * uint8 level：OFFER\_ROUTER，设置 router 信息 * void\* optarg：bit0, 0 禁⽤用 router 信息；bit0, 1 启⽤用 router 信息；默认为 1 |
| 返回 | true：成功 false：失败 |
| 示例例 | uint8 mode = 0；  wifi\_softap\_set\_dhcps\_offer\_option(OFFER\_ROUTER, &mode); |

3.5.48. wifi\_set\_phy\_mode

|  |  |
| --- | --- |
| 功能 | 设置 ESP8266 物理理层模式 (802.11 b/g/n) |
| 函数定义 | bool wifi\_set\_phy\_mode(enum phy\_mode mode) |
| 参数 | enum phy\_mode mode : 物理层模式  enum phy\_mode {  PHY\_MODE\_11B = 1,  PHY\_MODE\_11G = 2,  PHY\_MODE\_11N = 3  }; |
| 返回 | true：成功 false：失败 |

3.5.49. wifi\_get\_phy\_mode

|  |  |
| --- | --- |
| 功能 | 查询 ESP8266 物理理层模式 (802.11 b/g/n) |
| 函数定义 | enum phy\_mode wifi\_get\_phy\_mode(void) |
| 参数 | ⽆无 |
| 返回 | enum phy\_mode{  PHY\_MODE\_11B = 1,  PHY\_MODE\_11G = 2,  PHY\_MODE\_11N = 3  }; |

3.5.50. wifi\_get\_ip\_info

|  |  |
| --- | --- |
| 功能 | 查询 Wi-Fi Station 接⼝口或者 SoftAP 接⼝口的 IP 地址 |
| 注意 | 在 user\_init 中，由于初始化尚未完成，⽆无法通过本接⼝口查询到有效 IP 地址。 |
| 函数定义 | bool wifi\_get\_ip\_info(  uint8 if\_index,   struct ip\_info \*info ) |
| 参数 | uint8 if\_index：获取 Station 或者 SoftAP 接⼝口的信息  #define STATION\_IF 0x00  #define SOFTAP\_IF 0x01 struct ip\_info \*info：获取到的 IP 信息 |
| 返回 | true：成功 false：失败 |

3.5.51. wifi\_set\_ip\_info

|  |  |
| --- | --- |
| 功能 | 设置 Wi-Fi Station 或者 SoftAP 的 IP 地址 |
| 注意 | * 本接⼝口设置静态 IP，请先关闭对应 DHCP 功能 wifi\_station\_dhcpc\_stop 或者 wifi\_softap\_dhcps\_stop * 设置静态 IP，则关闭 DHCP；DHCP 开启，则静态 IP 失效。 |
| 函数定义 | bool wifi\_set\_ip\_info(  uint8 if\_index,   struct ip\_info \*info ) |
| 参数 | uint8 if\_index：设置 Station 或者 SoftAP 接⼝口  #define STATION\_IF 0x00  #define SOFTAP\_IF 0x01 struct ip\_info \*info：获取到的 IP 信息 |
| 返回 | true：成功 false：失败 |
| 示例例 | wifi\_set\_opmode(STATIONAP\_MODE); //Set softAP + station mode struct ip\_info info; wifi\_station\_dhcpc\_stop(); wifi\_softap\_dhcps\_stop();    IP4\_ADDR(&info.ip, 192, 168, 3, 200);  IP4\_ADDR(&info.gw, 192, 168, 3, 1); IP4\_ADDR(&info.netmask, 255, 255, 255, 0); wifi\_set\_ip\_info(STATION\_IF, &info);    IP4\_ADDR(&info.ip, 10, 10, 10, 1);  IP4\_ADDR(&info.gw, 10, 10, 10, 1);  IP4\_ADDR(&info.netmask, 255, 255, 255, 0); wifi\_set\_ip\_info(SOFTAP\_IF, &info); wifi\_softap\_dhcps\_start(); |

3.5.52. wifi\_set\_macaddr

|  |  |
| --- | --- |
| 功能 | 设置 MAC 地址 |
| 注意 | * 本接⼝口必须在 user\_init 中调⽤用 * ESP8266 SoftAP 和 Station MAC 地址不不同，请勿将两者设置为同⼀一 MAC 地址 * ESP8266 MAC 地址第⼀一个字节的 bit 0 不不能为 1。例例如，MAC 地址可以设置为 1a:XX:XX:XX:XX:XX，但不不能设置为 15:XX:XX:XX:XX:XX。 |
| 函数定义 | bool wifi\_set\_macaddr(  uint8 if\_index,   uint8 \*macaddr  ) |
| 参数 | uint8 if\_index：设置 Station 或者 SoftAP 接⼝口  #define STATION\_IF 0x00  #define SOFTAP\_IF 0x01 uint8 \*macaddr：MAC 地址 |
| 返回 | true：成功 false：失败 |
| 示例例 | wifi\_set\_opmode(STATIONAP\_MODE);  char sofap\_mac[6] = {0x16, 0x34, 0x56, 0x78, 0x90, 0xab}; char sta\_mac[6] = {0x12, 0x34, 0x56, 0x78, 0x90, 0xab}; wifi\_set\_macaddr(SOFTAP\_IF, sofap\_mac); wifi\_set\_macaddr(STATION\_IF, sta\_mac); |

3.5.53. wifi\_get\_macaddr

|  |  |
| --- | --- |
| 功能 | 查询 MAC 地址 |
| 函数定义 | bool wifi\_get\_macaddr(  uint8 if\_index,   uint8 \*macaddr  ) |
| 参数 | uint8 if\_index：查询 Station 或者 SoftAP 接⼝口  #define STATION\_IF 0x00  #define SOFTAP\_IF 0x01 uint8 \*macaddr：MAC 地址 |
| 返回 | true：成功 false：失败 |

3.5.54. wifi\_set\_sleep\_type

|  |  |
| --- | --- |
| 功能 | 设置省电模式。设置为 NONE\_SLEEP\_T，则关闭省电模式。 |
| 注意 | * 默认为 Modem-sleep 模式。 * Light Sleep 为了了降低功耗，将 TCP timer tick 由原本的 250ms 改为了了 3s，这将导致 TCP timer 超时时间相应增加；如果⽤用户对 TCP timer 的准确度有要求，请使⽤用 modem sleep 或者 deep sleep 模式。 |
| 函数定义 | bool wifi\_set\_sleep\_type(enum sleep\_type type) |
| 参数 | enum sleep\_type type：省电模式 |
| 返回 | true：成功 false：失败 |

3.5.55. wifi\_get\_sleep\_type

|  |  |
| --- | --- |
| 功能 | 查询省电模式。 |
| 函数定义 | enum sleep\_type wifi\_get\_sleep\_type(void) |
| 参数 | ⽆无 |
| 返回 | enum sleep\_type {  NONE\_SLEEP\_T = 0;  LIGHT\_SLEEP\_T,  MODEM\_SLEEP\_T  }; |

3.5.56. wifi\_status\_led\_install

|  |  |
| --- | --- |
| 功能 | 注册 Wi-Fi 状态 LED。 |
| 函数定义 | void wifi\_status\_led\_install (  uint8 gpio\_id,   uint32 gpio\_name,   uint8 gpio\_func  ) |
| 参数 | * uint8 gpio\_id：GPIO ID * uint8 gpio\_name：GPIO MUX 名称 * uint8 gpio\_func：GPIO 功能 |
| 返回 | ⽆无 |
| 示例例 | 使⽤用 GPIO0 作为 Wi-Fi 状态 LED  #define HUMITURE\_WIFI\_LED\_IO\_MUX PERIPHS\_IO\_MUX\_GPIO0\_U  #define HUMITURE\_WIFI\_LED\_IO\_NUM 0  #define HUMITURE\_WIFI\_LED\_IO\_FUNC FUNC\_GPIO0 wifi\_status\_led\_install(HUMITURE\_WIFI\_LED\_IO\_NUM,  HUMITURE\_WIFI\_LED\_IO\_MUX, HUMITURE\_WIFI\_LED\_IO\_FUNC) |

3.5.57. wifi\_status\_led\_uninstall

|  |  |
| --- | --- |
| 功能 | 注销 Wi-Fi 状态 LED |
| 函数定义 | void wifi\_status\_led\_uninstall () |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.5.58. wifi\_set\_broadcast\_if

|  |  |
| --- | --- |
| 功能 | 设置 ESP8266 发送 UDP ⼴广播包时，从 Station 接⼝口还是 SoftAP 接⼝口发送。默认从 SoftAP 接⼝口发送。 |
| 注意 | 如果设置仅从 Station 接⼝口发 UDP ⼴广播包，会影响 ESP8266 SoftAP 的功能，DHCP server ⽆无法使⽤用。需要使能 SoftAP 的⼴广播包功能，才可正常使⽤用 ESP8266 SoftAP。 |
| 函数定义 | bool wifi\_set\_broadcast\_if (uint8 interface) |
| 参数 | uint8 interface   * 1：station * 2：SoftAP * 3：Station 和 SoftAP 接⼝口均发送 |
| 返回 | true：成功 false：失败 |

3.5.59. wifi\_get\_broadcast\_if

|  |  |
| --- | --- |
| 功能 | 查询 ESP8266 发送 UDP ⼴广播包时，从 Station 接⼝口还是 SoftAP 接⼝口发送。 |
| 函数定义 | uint8 wifi\_get\_broadcast\_if (void) |
| 参数 | ⽆无 |
| 返回 | * 1：Station * 2：SoftAP * 3：Station 和 SoftAP 接⼝口均发送 |

3.5.60. wifi\_set\_event\_handler\_cb

|  |  |
| --- | --- |
| 功能 | 注册 Wi-Fi event 处理理回调 |
| 函数定义 | void wifi\_set\_event\_handler\_cb(wifi\_event\_handler\_cb\_t cb) |
| 参数 | wifi\_event\_handler\_cb\_t cb：回调函数 |
| 返回 | ⽆无 |

|  |  |  |
| --- | --- | --- |
| 示例例 | { | void wifi\_handle\_event\_cb(System\_Event\_t \*evt)  os\_printf("event %x\n", evt->event); switch (evt->event) { case EVENT\_STAMODE\_CONNECTED:  os\_printf("connect to ssid %s, channel %d\n",  evt->event\_info.connected.ssid, evt->event\_info.connected.channel); break;  case EVENT\_STAMODE\_DISCONNECTED:  os\_printf("disconnect from ssid %s, reason %d\n",  evt->event\_info.disconnected.ssid, evt->event\_info.disconnected.reason); break;  case EVENT\_STAMODE\_AUTHMODE\_CHANGE:  os\_printf("mode: %d -> %d\n",  evt->event\_info.auth\_change.old\_mode, evt->event\_info.auth\_change.new\_mode); break;  case EVENT\_STAMODE\_GOT\_IP:  os\_printf("ip:" IPSTR ",mask:" IPSTR ",gw:" IPSTR,  IP2STR(&evt->event\_info.got\_ip.ip),  IP2STR(&evt->event\_info.got\_ip.mask), IP2STR(&evt->event\_info.got\_ip.gw)); os\_printf("\n"); break;  case EVENT\_SOFTAPMODE\_STACONNECTED:  os\_printf("station: " MACSTR "join, AID = %d\n",  MAC2STR(evt->event\_info.sta\_connected.mac), evt->event\_info.sta\_connected.aid); break;  case EVENT\_SOFTAPMODE\_STADISCONNECTED:  os\_printf("station: " MACSTR "leave, AID = %d\n",  MAC2STR(evt->event\_info.sta\_disconnected.mac), evt->event\_info.sta\_disconnected.aid); break; default:  break;  } |
|  | }  void user\_init(void)  {  // TODO: add your own code here....  wifi\_set\_event\_handler\_cb(wifi\_handle\_event\_cb);  } | |

3.5.61. wifi\_wps\_enable

|  |  |
| --- | --- |
| 功能 | 使能 Wi-Fi WPS 功能 |
| 注意 | WPS 功能必须在 ESP8266 Station 使能的情况下调⽤用。 |
| 结构体 | typedef enum wps\_type { WPS\_TYPE\_DISABLE=0,  WPS\_TYPE\_PBC,  WPS\_TYPE\_PIN,  WPS\_TYPE\_DISPLAY,  WPS\_TYPE\_MAX,  }WPS\_TYPE\_t; |
| 函数定义 | bool wifi\_wps\_enable(WPS\_TYPE\_t wps\_type) |
| 参数 | WPS\_TYPE\_t wps\_type：WPS 的类型，⽬目前仅⽀支持 WPS\_TYPE\_PBC |
| 返回 | true：成功 false：失败 |

3.5.62. wifi\_wps\_disable

|  |  |
| --- | --- |
| 功能 | 关闭 Wi-Fi WPS 功能，释放占⽤用的资源。 |
| 函数定义 | bool wifi\_wps\_disable(void) |
| 参数 | ⽆无 |
| 返回 | true：成功 false：失败 |

3.5.63. wifi\_wps\_start

|  |  |
| --- | --- |
| 功能 | WPS 开始进⾏行行交互 |
| 注意 | WPS 功能必须在 ESP8266 Station 使能的情况下调⽤用。 |
| 函数定义 | bool wifi\_wps\_start(void) |
| 参数 | ⽆无 |
| 返回 | true：成功开始交互，并不不表示 WPS 成功完成 false：失败 |

3.5.64. wifi\_set\_wps\_cb

|  |  |
| --- | --- |
| 功能 | 设置 WPS 回调函数，回调函数中将传⼊入 WPS 运⾏行行状态。WPS 不不⽀支持 WEP 加密⽅方式。 |
| 回调及参数结构体 | typedef void (\*wps\_st\_cb\_t)(int status); enum wps\_cb\_status {  WPS\_CB\_ST\_SUCCESS = 0,  WPS\_CB\_ST\_FAILED,  WPS\_CB\_ST\_TIMEOUT,  WPS\_CB\_ST\_WEP, // WPS failed because that WEP is not supported. WPS\_CB\_ST\_SCAN\_ERR, // can not find the target WPS AP  }; |
| 注意 | * 如果回调函数的传⼊入参数状态为 WPS\_CB\_ST\_SUCCESS，表示成功获得 AP 密钥，请调⽤用   wifi\_wps\_disable 关闭 WPS 功能释放资源，并调⽤用 wifi\_station\_connect 连接 AP。   * 否则，表示 WPS 失败，可以创建⼀一个定时器器，间隔⼀一段时间后调⽤用 wifi\_wps\_start 再次尝试 WPS，或者调⽤用 wifi\_wps\_disable 关闭 WPS 并释放资源。 |
| 函数定义 | bool wifi\_set\_wps\_cb(wps\_st\_cb\_t cb) |
| 参数 | wps\_st\_cb\_t cb：回调函数 |
| 返回 | true：成功 false：失败 |

3.5.65. wifi\_register\_send\_pkt\_freedom\_cb

|  |  |
| --- | --- |
| 功能 | 注册 freedom 发包的回调函数。freedom 发包功能，即⽀支持发送⽤用户⾃自定义 802.11 的包。 |
| 注意 | * freedom 发包必须等前⼀一个包发送完毕，进⼊入发包回调 freedom\_outside\_cb\_t 之后，才能发下⼀一个包。 * 设置发送回调函数可以⽤用来判别包是否发送成功（IEEE802.11 MAC 底层是否发送成功）。   使⽤用发送回调函数请注意如下情况：   * + 针对单播包：     - 回调函数状态显示成功时，对⽅方应⽤用层实际没有收到的状况。原因：       1. 存在流氓设备进⾏行行攻击       2. 加密密钥设置错误       3. 应⽤用层丢包   若需要更更强地发包保证发包成功率，请在应⽤用层实现发包握⼿手机制。   * + - 回调函数状态显示失败时，对⽅方应⽤用层实际已收到的状况。原因：       1. 信道繁忙，未收到对⽅方ACK。   请注意应⽤用层发包重传，接收⽅方需要检测重传包。   * + 针对组播包（包括⼴广播包）：     - 回调函数状态显示成功，表示组播包已成功发送     - 回调函数状态显示失败，表示组播包发送失败 |
| 回调函数定义 | typedef void (\*freedom\_outside\_cb\_t)(uint8 status); status：0，发包成功；其他值，发包失败。 |
| 参数 | freedom\_outside\_cb\_t cb：回调函数 |
| 返回 | 0：注册成功 -1：注册失败 |

3.5.66. wifi\_unregister\_send\_pkt\_freedom\_cb

|  |  |
| --- | --- |
| 功能 | 注销 freedom 发包的回调函数。 |
| 函数定义 | void wifi\_unregister\_send\_pkt\_freedom\_cb(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.5.67. wifi\_send\_pkt\_freedom

|  |  |
| --- | --- |
| 功能 | 发包函数。 |
| 注意 | * 发送包必须是完整的 802.11 包，⻓长度不不包含 FCS。发包⻓长度必须⼤大于 ⼩小 802.11 头，即   24 字节，且不不能超过 1400 字节，否则返回发包失败。   * duration 域填写⽆无效，由 ESP8266 底层程序决定，⾃自动填充。 * 发包速率限制成管理理包速率，与系统的发包速率⼀一致。 * ⽀支持发送：⾮非加密的数据包，⾮非加密的 beacon/probe req/probe resp。 * 不不⽀支持发送：所有加密包 (即包头中的加密 bit 必须为 0，否则返回发包失败)，控制包，除 beacon/probe req/probe resp 以外的其他管理理包。 * freedom 发包必须等前⼀一个包发送完毕，进⼊入发包回调之后，才能发下⼀一个包。 |
| 函数定义 | int wifi\_send\_pkt\_freedom(uint8 \*buf, int len，bool sys\_seq) |
| 参数 | * uint8 \*buf：数据包指针 * int len：数据包⻓长度 * bool sys\_seq：是否跟随系统的 802.11 包 sequence number，如果跟随系统，将会在每次发包后⾃自加 1 |
| 返回 | 0：成功 -1：失败 |

3.5.68. wifi\_rfid\_locp\_recv\_open

|  |  |
| --- | --- |
| 功能 | 开启 RFID LOCP (Location Control Protocol) 功能，⽤用于接收 WDS 类型的包。 |
| 函数定义 | int wifi\_rfid\_locp\_recv\_open(void) |
| 参数 | ⽆无 |
| 返回 | 0：成功  其他值：失败 |

3.5.69. wifi\_rfid\_locp\_recv\_close

|  |  |
| --- | --- |
| 功能 | 关闭 RFID LOCP (Location Control Protocol) 功能。 |
| 函数定义 | void wifi\_rfid\_locp\_recv\_close(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.5.70. wifi\_register\_rfid\_locp\_recv\_cb

|  |  |
| --- | --- |
| 功能 | 注册 WDS 收包回调。仅在收到的 WDS 包的第⼀一个 MAC 地址为组播地址时，才会进⼊入回调函数。 |
| 回调函数定义 | typedef void (\*rfid\_locp\_cb\_t)(uint8 \*frm, int len, int rssi); |
| 参数 | * uint8 \*frm：指向 802.11 包头的指针 * int len：数据包⻓长度 * int rssi：信号强度 |
| 返回 | 0：成功  其他值：失败 |

3.5.71. wifi\_unregister\_rfid\_locp\_recv\_cb

|  |  |
| --- | --- |
| 功能 | 注销 WDS 收包回调。 |
| 函数定义 | void wifi\_unregister\_rfid\_locp\_recv\_cb(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.5.72. wifi\_enable\_gpio\_wakeup

|  |  |
| --- | --- |
| 功能 | 使能 GPIO 唤醒 Light-sleep 模式的功能。 |
| 注意 | 在⾃自动 Light-sleep 休眠 wifi\_set\_sleep\_type(LIGHT\_SLEEP\_T); 的情况下，由 GPIO 触发 ESP8266 从 Light-sleep 唤醒之后，如需再次进⼊入休眠时，将判断唤醒 GPIO 的状态:   * 如果 GPIO 仍然处于唤醒状态，则进⼊入 Modem-sleep 休眠； * 如果 GPIO 不不处于唤醒状态，则进⼊入 Light-sleep 休眠。 |
| 函数定义 | void wifi\_enable\_gpio\_wakeup(uint32 i, GPIO\_INT\_TYPE intr\_status) |
| 参数 | uint32 i：GPIO 号，取值范围：[0, 15]  GPIO\_INT\_TYPE intr\_status：GPIO 触发唤醒的状态 |
| 返回 | ⽆无 |
| 示例例 | 设置 GPIO12 低电平时，将 ESP8266 从 Light-sleep 模式唤醒。  GPIO\_DIS\_OUTPUT(12);  PIN\_FUNC\_SELECT(PERIPHS\_IO\_MUX\_MTDI\_U, FUNC\_GPIO12); wifi\_enable\_gpio\_wakeup(12, GPIO\_PIN\_INTR\_LOLEVEL); |

3.5.73. wifi\_disable\_gpio\_wakeup

|  |  |
| --- | --- |
| 功能 | 取消 GPIO 唤醒 Light-sleep 模式的功能。 |
| 函数定义 | void wifi\_disable\_gpio\_wakeup(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.5.74. wifi\_set\_country

|  |  |
| --- | --- |
| 功能 | 设置 WiFi 国家码 |
| 函数定义 | bool wifi\_set\_country(wifi\_country\_t \*country) |
| 参数 | wifi\_country\_t \*country: 国家码信息 |
| 注意 | * 默认国家码为 {.cc="CN", .schan=1, .nchan=13, policy=WIFI\_COUNTRY\_POLICY\_AUTO} * 当 policy=WIFI\_COUNTRY\_POLICY\_AUTO，ESP8266 的国家码会在连上 AP 后，⾃自动更更改为与 AP ⼀一致；当与 AP 断开连接后，⼜又回到原设置值。 * 当 policy=WIFI\_COUNTRY\_POLICY\_MANUAL，ESP8266 的国家码将始终保持为设置值。 * 在 station+softAP 模式下，如果 ESP8266 station 的国家码信息改变，softAP 端 probe response/beacon 中的 country IE 国家码信息也会同样改变。 * 国家码信息不不保存在 flash，重新上电后，需要重新配置。 |
| 返回 | true：成功 false：失败 |

3.5.75. wifi\_get\_country

|  |  |
| --- | --- |
| 功能 | 获取当前 WiFi 国家码 |
| 函数定义 | bool wifi\_get\_country(wifi\_country\_t \*country) |
| 参数 | wifi\_country\_t \*country: 国家码信息 |
| 返回 | true：成功 false：失败 |

#### 3.6. Rate Control 接⼝口

Wi-Fi Rate Control 接⼝口位于 */ESP8266\_NONOS\_SDK/include/user\_interface.h*。 1. wifi\_set\_user\_fixed\_rate

|  |  |  |
| --- | --- | --- |
| 功能 | 设置 ESP8266 Station 或 SoftAP 发数据的固定 rate 和 mask | |
| 参数定义 | enum FIXED\_RATE {  PHY\_RATE\_48 = 0x8,  PHY\_RATE\_24 = 0x9,  PHY\_RATE\_12 = 0xA,  PHY\_RATE\_6 = 0xB,  PHY\_RATE\_54 = 0xC,  PHY\_RATE\_36 = 0xD,  PHY\_RATE\_18 = 0xE,  PHY\_RATE\_9 = 0xF, }  #define FIXED\_RATE\_MASK\_NONE | (0x00) |
|  | #define FIXED\_RATE\_MASK\_STA | (0x01) |
|  | #define FIXED\_RATE\_MASK\_AP (0x02)  #define FIXED\_RATE\_MASK\_ALL (0x03) | |
| 注意 | * 当 enable\_mask 的对应 bit 为 1，ESP8266 Station 或 SoftAP 才会以固定 rate 发送数据。 * 如果 enable\_mask 设置成 0，则 ESP8266 Station 和 SoftAP 均不不会以固定 rate 发送数据。 * ESP8266 Station 和 SoftAP 共享同⼀一个 rate，不不⽀支持分别设置为不不同 rate 值。 | |
| 函数定义 | int wifi\_set\_user\_fixed\_rate(uint8 enable\_mask, uint8 rate) | |
| 参数 | * uint8 enable\_mask   + 0x00：禁⽤用固定 rate   + 0x01：固定 rate ⽤用于 ESP8266 Station 接⼝口   + 0x02：固定 rate ⽤用于 ESP8266 SoftAP 接⼝口   + 0x03：固定 rate ⽤用于 ESP8266 Station+SoftAP * uint8 rate：固定 rate 值 | |
| 返回 | 0：成功  其他：失败 | |

1. wifi\_get\_user\_fixed\_rate

|  |  |
| --- | --- |
| 功能 | 获取已经设置的固定 rate 的 mask 和 rate 值 |
| 函数定义 | int wifi\_get\_user\_fixed\_rate(uint8 \*enable\_mask, uint8 \*rate) |
| 参数 | * uint8 \*enable\_mask：mask 的指针 * uint8 \*rate：rate 的指针 |
| 返回 | 0：成功  其他：失败 |

1. wifi\_set\_user\_sup\_rate

|  |  |
| --- | --- |
| 功能 | 设置 ESP8266 beacon、probe req/resp 等包⾥里里的 support rate 的 IE 中⽀支持的 rate 范围。⽤用于将 ESP8266 ⽀支持的通信速率告知通信对⽅方，以限制对⽅方设备的发包速率。 |
| 注意 | 本接⼝口⽬目前仅⽀支持 802.11g 模式，后续会增加⽀支持 802.11b。 |
| 参数定义 | enum support\_rate {  RATE\_11B5M = 0,  RATE\_11B11M = 1,  RATE\_11B1M = 2,  RATE\_11B2M = 3,  RATE\_11G6M = 4,  RATE\_11G12M = 5,  RATE\_11G24M = 6,  RATE\_11G48M = 7,  RATE\_11G54M = 8,  RATE\_11G9M = 9,  RATE\_11G18M = 10,  RATE\_11G36M = 11,  }; |
| 函数定义 | int wifi\_set\_user\_sup\_rate(uint8 min, uint8 max) |
| 参数 | • uint8 min：support rate 下限值，仅⽀支持从 enum support\_rate 中取值。 • uint8 max：support rate 上限值，仅⽀支持从 enum support\_rate 中取值。 |
| 返回 | 0：成功  其他：失败 |
| 示例例 | wifi\_set\_user\_sup\_rate(RATE\_11G6M, RATE\_11G24M); |

1. wifi\_set\_user\_rate\_limit

|  |  |
| --- | --- |
| 功能 | 设置 ESP8266 发包的初始速率范围。重传速率则不不受此接⼝口限制。 |
| 参数定义 | enum RATE\_11B\_ID {  RATE\_11B\_B11M = 0,   RATE\_11B\_B5M = 1,  RATE\_11B\_B2M = 2,  RATE\_11B\_B1M = 3,  }  enum RATE\_11G\_ID {  RATE\_11G\_G54M = 0,  RATE\_11G\_G48M = 1,  RATE\_11G\_G36M = 2,  RATE\_11G\_G24M = 3,  RATE\_11G\_G18M = 4,  RATE\_11G\_G12M = 5,  RATE\_11G\_G9M = 6,  RATE\_11G\_G6M = 7  RATE\_11G\_B5M = 8,  RATE\_11G\_B2M = 9,  RATE\_11G\_B1M = 10  }  enum RATE\_11N\_ID {  RATE\_11N\_MCS7S = 0,   RATE\_11N\_MCS7 = 1,  RATE\_11N\_MCS6 = 2,  RATE\_11N\_MCS5 = 3,  RATE\_11N\_MCS4 = 4,  RATE\_11N\_MCS3 = 5,  RATE\_11N\_MCS2 = 6,  RATE\_11N\_MCS1 = 7,  RATE\_11N\_MCS0 = 8,   RATE\_11N\_B5M = 9,  RATE\_11N\_B2M = 10,  RATE\_11N\_B1M = 11   } |
| 函数定义 | bool wifi\_set\_user\_rate\_limit(uint8 mode, uint8 ifidx, uint8 max, uint8 min) |
| 参数 | * uint8 mode：设置模式   #define RC\_LIMIT\_11B 0  #define RC\_LIMIT\_11G 1  #define RC\_LIMIT\_11N 2   * uint8 ifidx：设置接⼝口   0x00 - ESP8266 station 接口  0x01 - ESP8266 soft-AP 接口   * uint8 max：速率上限。请从第⼀一个参数 mode 对应的速率枚举中取值。 * uint8 min：速率下限。请从第⼀一个参数 mode 对应的速率枚举中取值。 |
| 返回 | true：成功 false：失败 |
| 示例例 | 设置 11G 模式下的 ESP8266 station 接⼝口的速率，限制为 ⼤大 18M， ⼩小 6M。  wifi\_set\_user\_rate\_limit(RC\_LIMIT\_11G, 0, RATE\_11G\_G18M, RATE\_11G\_G6M); |

1. wifi\_set\_user\_limit\_rate\_mask

|  |  |
| --- | --- |
| 功能 | 设置使能受 wifi\_set\_user\_rate\_limit 限制速率的接⼝口。 |
| 参数定义 | #define LIMIT\_RATE\_MASK\_NONE (0x00）  #define LIMIT\_RATE\_MASK\_STA (0x01)  #define LIMIT\_RATE\_MASK\_AP (0x02)  #define LIMIT\_RATE\_MASK\_ALL (0x03) |
| 函数定义 | bool wifi\_ set\_user\_limit\_rate\_mask(uint8 enable\_mask) |
| 参数 | uint8 enable\_mask   * 0x00：ESP8266 Station+SoftAP 接⼝口均不不受限制 * 0x01：ESP8266 Station 接⼝口开启限制 * 0x02：ESP8266 SoftAP 接⼝口开启限制 * 0x03：ESP8266 Station+SoftAP 接⼝口均开启限制 |
| 返回 | true：成功 false：失败 |

1. wifi\_get\_user\_limit\_rate\_mask

|  |  |
| --- | --- |
| 功能 | 查询当前受 wifi\_set\_user\_rate\_limit 限制速率的接⼝口。 |
| 函数定义 | uint8 wifi\_get\_user\_limit\_rate\_mask(void) |
| 参数 | ⽆无 |
| 返回 | * 0x00：ESP8266 Station+SoftAP 接⼝口均不不受限制 * 0x01：ESP8266 Station 接⼝口开启限制 * 0x02：ESP8266 SoftAP 接⼝口开启限制 * 0x03：ESP8266 Station+SoftAP 接⼝口均开启限制 |

### 3.7. 强制休眠接⼝口

强制休眠接⼝口位于 */ESP8266\_NONOS\_SDK/include/user\_interface.h*。

使⽤用强制休眠功能，必须先设置 Wi-Fi ⼯工作模式为 NULL\_MODE。从强制休眠中唤醒

ESP8266，或者休眠时间到，进⼊入唤醒回调 (由 wifi\_fpm\_set\_wakeup\_cb 注册)后，先关闭强制休眠功能，才能再设置 Wi-Fi ⼯工作模式为 Station、SoftAP 或 Station+SoftAP 的正常⼯工作模式运⾏行行，具体可参考本章节后⽂文提供的“示例例代码”。

注意，

* 定时器器会影响进⼊入 Light-sleep 模式，如需 Light-sleep 休眠，请先将定时器器关闭。
* Light Sleep 为了了降低功耗，将 TCP timer tick 由原本的 250ms 改为了了 3s，这将导致 TCP timer 超时时间相应增加；如果⽤用户对 TCP timer 的准确度有要求，请使⽤用 modem sleep 或者 deep sleep 模式。

3.7.1. wifi\_fpm\_open

|  |  |
| --- | --- |
| 功能 | 开启强制休眠功能 |
| 函数定义 | void wifi\_fpm\_open (void) |
| 默认值 | 强制 sleep 功能关闭 |
| 返回 | ⽆无 |

3.7.2. wifi\_fpm\_close

|  |  |
| --- | --- |
| 功能 | 关闭强制休眠功能 |
| 函数定义 | void wifi\_fpm\_close (void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.7.3. wifi\_fpm\_do\_wakeup

|  |  |
| --- | --- |
| 功能 | 唤醒 MODEM\_SLEEP\_T 类型的强制休眠 |
| 注意 | 本接⼝口仅⽀支持在强制休眠功能开启的情况下调⽤用，可在 wifi\_fpm\_open 之后调⽤用；在 wifi\_fpm\_close 之后，不不可以调⽤用。 |
| 函数定义 | void wifi\_fpm\_do\_wakeup (void) |
| 返回 | ⽆无 |

3.7.4. wifi\_fpm\_set\_wakeup\_cb

|  |  |
| --- | --- |
| 功能 | 设置强制休眠的定时唤醒功能超时，系统醒来后的回调函数 |
| 注意 | * 本接⼝口仅⽀支持在强制休眠功能开启的情况下调⽤用，可在 wifi\_fpm\_open 之后调⽤用；在 wifi\_fpm\_close 之后，不不可以调⽤用。 * 仅在定时唤醒 wifi\_fpm\_do\_sleep 且参数不不为 0xFFFFFFF 功能的定时时间到，系统醒来，才会进⼊入唤醒回调 fpm\_wakeup\_cb\_func。 * MODEM\_SLEEP\_T 类型的强制休眠被 wifi\_fpm\_do\_wakeup 唤醒，并不不会进⼊入唤醒回调。 |
| 函数定义 | void wifi\_fpm\_set\_wakeup\_cb(void (\*fpm\_wakeup\_cb\_func)(void)) |
| 参数 | void (\*fpm\_wakeup\_cb\_func)(void)：回调函数 |
| 返回 | ⽆无 |

3.7.5. wifi\_fpm\_do\_sleep

|  |  |
| --- | --- |
| 功能 | 让系统强制休眠，休眠时间到后，系统将⾃自动醒来。 |
| 注意 | * 本接⼝口仅⽀支持在强制休眠功能开启的情况下调⽤用，可在 wifi\_fpm\_open 之后调⽤用；在 wifi\_fpm\_close 之后，不不可以调⽤用。 * 本接⼝口返回 0 表示休眠设置成功，但并不不表示⽴立即进⼊入休眠状态。系统会在进⼊入底层相关任务处理理时，进⾏行行休眠。请勿在调⽤用本接⼝口后，⽴立即调⽤用其他 Wi-Fi 相关操作。 |
| 函数定义 | int8 wifi\_fpm\_do\_sleep (uint32 sleep\_time\_in\_us) |
| 参数 | uint32 sleep\_time\_in\_us：休眠时间，单位：us，取值范围：10000 ~  268435455(0xFFFFFFF)  如果参数设置为 0xFFFFFFF，则系统将⼀一直休眠，直⾄至：   * 若 wifi\_fpm\_set\_sleep\_type 设置为 LIGHT\_SLEEP\_T，可被 GPIO 唤醒。 * 若 wifi\_fpm\_set\_sleep\_type 设置为 MODEM\_SLEEP\_T，可被 wifi\_fpm\_do\_wakeup 唤醒。 |
| 返回 | 0：休眠设置成功  -1：强制休眠的状态错误，休眠失败  -2：强制休眠功能未开启，休眠失败 |

3.7.6. wifi\_fpm\_set\_sleep\_type

|  |  |  |  |
| --- | --- | --- | --- |
| 功能 | 设置系统强制休眠的休眠类型。 | | |
| 注意 | 如需调⽤用本接⼝口，请在 wifi\_fpm\_open 之前调⽤用。 | | |
| 函数定义 | void wifi\_fpm\_set\_sleep\_type (enum sleep\_type type) | | |
| 参数 |  | enum sleep\_type{ NONE\_SLEEP\_T =  LIGHT\_SLEEP\_T, | 0, |
|  | }; | MODEM\_SLEEP\_T, |  |
| 返回 | ⽆无 |  |  |

3.7.7. wifi\_fpm\_get\_sleep\_type

|  |  |
| --- | --- |
| 功能 | 查询系统强制休眠的休眠类型。 |
| 函数定义 | enum sleep\_type wifi\_fpm\_get\_sleep\_type (void) |
| 参数 | ⽆无 |
| 返回 | enum sleep\_type{  NONE\_SLEEP\_T = 0,  LIGHT\_SLEEP\_T,  MODEM\_SLEEP\_T,  }; |

3.7.8. wifi\_fpm\_auto\_sleep\_set\_in\_null\_mode

|  |  |
| --- | --- |
| 功能 | 设置在关闭 Wi-Fi 模式 wifi\_set\_opmode(NULL\_MODE) 的情况下，是否⾃自动进⼊入 Modem-sleep 模式。 |
| 函数定义 | void wifi\_fpm\_auto\_sleep\_set\_in\_null\_mode (uint8 req) |
| 参数 | uint8 req   * 0：关闭 Wi-Fi 模式后，不不⾃自动进⼊入 Modem-sleep 模式 * 1：关闭 Wi-Fi 模式后，⾃自动进⼊入 Modem-sleep 模式。 |
| 返回 | ⽆无 |

3.7.9. 示例例代码调⽤用强制休眠接⼝口，在需要的情况下强制关闭 RF 电路路以降低功耗。

|  |
| --- |
| ⚠**注意：**  强制休眠接⼝口调⽤用后，并不不会⽴立即休眠，⽽而是等到系统 *idle task* 执⾏行行时才进⼊入休眠。请参考下述示例例使  ⽤用。 |

示例例⼀一：Modem-sleep 模式强制进⼊入 Modem-sleep 模式，即强制关闭 RF。

|  |
| --- |
| #define FPM\_SLEEP\_MAX\_TIME 0xFFFFFFF    void fpm\_wakup\_cb\_func1(void)  {  wifi\_fpm\_close(); // disable force sleep function  wifi\_set\_opmode(STATION\_MODE); // set station mode  wifi\_station\_connect(); // connect to AP }    void user\_func(...)  {  wifi\_station\_disconnect();  wifi\_set\_opmode(NULL\_MODE); // set WiFi mode to null mode.  wifi\_fpm\_set\_sleep\_type(MODEM\_SLEEP\_T); // modem sleep  wifi\_fpm\_open(); // enable force sleep  #ifdef SLEEP\_MAX  /\* For modem sleep, FPM\_SLEEP\_MAX\_TIME can only be wakened by calling wifi\_fpm\_do\_wakeup. \*/  wifi\_fpm\_do\_sleep(FPM\_SLEEP\_MAX\_TIME);  #else  // wakeup automatically when timeout.  wifi\_fpm\_set\_wakeup\_cb(fpm\_wakup\_cb\_func1)； // Set wakeup callback   wifi\_fpm\_do\_sleep(50\*1000);  #endif  }  #ifdef SLEEP\_MAX  void func1(void)  {  wifi\_fpm\_do\_wakeup();  wifi\_fpm\_close(); // disable force sleep function  wifi\_set\_opmode(STATION\_MODE); // set station mode  wifi\_station\_connect(); // connect to AP  }  #endif |

示例例⼆二：Light-sleep 模式

强制进⼊入 Light-sleep 模式，即强制关闭 RF 和 CPU，需要设置⼀一个回调函数，以便便唤醒后程序继续运⾏行行。注意，定时器器会影响进⼊入 Light-sleep 模式，如需休眠，请先将定时器器关闭。

|  |
| --- |
| #define FPM\_SLEEP\_MAX\_TIME 0xFFFFFFF void fpm\_wakup\_cb\_func1(void)  {  wifi\_fpm\_close(); // disable force sleep function  wifi\_set\_opmode(STATION\_MODE); // set station mode  wifi\_station\_connect(); // connect to AP } |

|  |
| --- |
| #ifndef SLEEP\_MAX // Wakeup till time out.  void user\_func(...)  {  wifi\_station\_disconnect();  wifi\_set\_opmode(NULL\_MODE); // set WiFi mode to null mode.  wifi\_fpm\_set\_sleep\_type(LIGHT\_SLEEP\_T); // light sleep  wifi\_fpm\_open(); // enable force sleep  wifi\_fpm\_set\_wakeup\_cb(fpm\_wakup\_cb\_func1); // Set wakeup callback   wifi\_fpm\_do\_sleep(50\*1000);  }  #else  // Or wake up by GPIO  void user\_func(...)  {  wifi\_station\_disconnect();  wifi\_set\_opmode(NULL\_MODE); // set WiFi mode to null mode.  wifi\_fpm\_set\_sleep\_type(LIGHT\_SLEEP\_T); // light sleep  wifi\_fpm\_open(); // enable force sleep  PIN\_FUNC\_SELECT(PERIPHS\_IO\_MUX\_MTDI\_U, FUNC\_GPIO12); wifi\_enable\_gpio\_wakeup(12, GPIO\_PIN\_INTR\_LOLEVEL);    wifi\_fpm\_set\_wakeup\_cb(fpm\_wakup\_cb\_func1); // Set wakeup callback   wifi\_fpm\_do\_sleep(FPM\_SLEEP\_MAX\_TIME);   ...  }  #endif |

#### 3.8. ESP-NOW 接⼝口

ESP-NOW 接⼝口位于 */ESP8266\_NONOS\_SDK/include/espnow.h*。

ESP-NOW 详细介绍请参考⽂文档 [ESP-NOW ⽤用户指南](http://www.espressif.com/sites/default/files/documentation/esp-now_user_guide_cn.pdf)，软件接⼝口使⽤用时的注意事项如下：

* ESP-NOW ⽬目前不不⽀支持组播包；
* ESP8266\_NONOS\_SDK\_V2.1 及之后版本，ESP-NOW ⽀支持发送⼴广播包，但请注意，仅⽀支持不不加密的⼴广播包；
* 建议 slave 和 combo ⻆角⾊色对应 ESP8266 SoftAP 模式或者 SoftAP+Station 共存模式；controller ⻆角⾊色对应 Station 模式；
* 当 ESP8266 处于 SoftAP+Station 共存模式时，若作为 slave 或 combo ⻆角⾊色，将从 SoftAP 接⼝口通信；若作为 controller ⻆角⾊色，将从 Station 接⼝口通信；
* ESP-NOW 不不实现休眠唤醒功能，因此如果通信对⽅方的 ESP8266 Station 正处于休眠状态，ESP-NOW 发包将会失败；
* ESP8266 Station 模式下， 多可设置 10 个加密的 ESP-NOW peer，加上不不加密的设备，总数不不超过 20 个；
* ESP8266 SoftAP 模式或者 SoftAP+station 模式下， 多设置 6 个加密的 ESPNOW peer，加上不不加密的设备，总数不不超过 20 个。

3.8.1. 结构体

enum esp\_now\_role {

ESP\_NOW\_ROLE\_IDLE = 0,

ESP\_NOW\_ROLE\_CONTROLLER,

ESP\_NOW\_ROLE\_SLAVE,

ESP\_NOW\_ROLE\_COMBO, // both slave and controller

ESP\_NOW\_ROLE\_MAX,

};

3.8.2. esp\_now\_init

|  |  |
| --- | --- |
| 功能 | 初始化 ESP-NOW 功能 |
| 函数定义 | init esp\_now\_init(void) |
| 参数 | ⽆无 |
| 返回 | 0：成功  其它：失败 |

3.8.3. esp\_now\_deinit

|  |  |
| --- | --- |
| 功能 | 卸载 ESP-NOW 功能 |
| 函数定义 | int esp\_now\_deinit(void) |
| 参数 | ⽆无 |
| 返回 | 0：成功  其它：失败 |

3.8.4. esp\_now\_register\_recv\_cb

|  |  |
| --- | --- |
| 功能 | 注册 ESP-NOW 收包的回调函数 |
| 注意 | 当收到 ESP-NOW 的数据包，进⼊入收包回调函数  typedef void (\*esp\_now\_recv\_cb\_t)(u8 \*mac\_addr, u8 \*data, u8 len) 回调函数的 3 个参数分别为：   * u8 \*mac\_add：发包⽅方的 MAC 地址 * u8 \*data：收到的数据 * u8 len：数据⻓长度 |
| 函数定义 | int esp\_now\_register\_recv\_cb(esp\_now\_recv\_cb\_t cb) |
| 参数 | esp\_now\_recv\_cb\_t cb：回调函数 |
| 返回 | 0：成功  其它：失败 |

3.8.5. esp\_now\_unregister\_recv\_cb

|  |  |
| --- | --- |
| 功能 | 注销 ESP-NOW 收包的回调函数 |
| 函数定义 | int esp\_now\_unregister\_recv\_cb(void) |
| 参数 | ⽆无 |
| 返回 | 0：成功  其它：失败 |

3.8.6. esp\_now\_register\_send\_cb

|  |  |
| --- | --- |
| 功能 | 设置 ESP-NOW 发包回调函数 |
| 注意 | * 当发送了了 ESP-NOW 的数据包，进⼊入收包回调函数   void esp\_now\_send\_cb\_t(u8 \*mac\_addr, u8 status)   * 回调函数的 2 个参数分别为：   + u8 \*mac\_addr：发包对⽅方的⽬目标 MAC 地址   + u8 status：发包状态；0，成功；否则，失败。对应结构体：   mt\_tx\_status {  MT\_TX\_STATUS\_OK = 0, MT\_TX\_STATUS\_FAILED,  }   * 发包回调函数不不判断密钥是否匹配，如果使⽤用密钥加密，请⾃自⾏行行确保密钥正确。 * 设置发送回调函数可以⽤用来判别包是否发送成功（IEEE802.11 MAC 底层是否发送成功）。   使⽤用发送回调函数请注意如下情况：   * + 针对单播包：     - 回调函数状态显示成功时，对⽅方应⽤用层实际没有收到的状况。原因：       1. 存在流氓设备进⾏行行攻击       2. 加密密钥设置错误       3. 应⽤用层丢包   若需要更更强地发包保证发包成功率，请在应⽤用层实现发包握⼿手机制。   * + - 回调函数状态显示失败时，对⽅方应⽤用层实际已收到的状况。原因：       1. 信道繁忙，未收到对⽅方ACK。   请注意应⽤用层发包重传，接收⽅方需要检测重传包。   * + 针对组播包（包括⼴广播包）：     - 回调函数状态显示成功，表示组播包已成功发送     - 回调函数状态显示失败，表示组播包发送失败 |
| 函数定义 | u8 esp\_now\_register\_send\_cb(esp\_now\_send\_cb\_t cb) |
| 参数 | esp\_now\_send\_cb\_t cb：回调函数 |
| 返回 | 0：成功  其它：失败 |

3.8.7. esp\_now\_unregister\_send\_cb

|  |  |
| --- | --- |
| 功能 | 注销 ESP-NOW 发包的回调函数，不不再报告发包状态。 |
| 函数定义 | int esp\_now\_unregister\_send\_cb(void) |
| 参数 | ⽆无 |
| 返回 | 0：成功  其它：失败 |

3.8.8. esp\_now\_send

|  |  |
| --- | --- |
| 功能 | 发送 ESP-NOW 数据包 |
| 函数定义 | int esp\_now\_send(u8 \*da, u8 \*data, int len) |
| 参数 | * u8 \*da：⽬目的 MAC 地址；如果为 NULL，则遍历 ESP-NOW 维护的所有 MAC 地址进⾏行行发送，否则，向指定 MAC 地址发送 * u8 \*data：要发送的数据 * int len：数据⻓长度 |
| 返回 | 0：成功  其它：失败 |

3.8.9. esp\_now\_add\_peer

|  |  |
| --- | --- |
| 功能 | 增加 ESP-NOW 匹配设备，将设备 MAC 地址存⼊入 ESP-NOW 维护的列列表。 |
| 函数定义 | int esp\_now\_add\_peer(u8 \*mac\_addr, u8 role,u8 channel, u8 \*key, u8 key\_len) |
| 参数 | * u8 \*mac\_addr：匹配设备的 MAC 地址 * u8 channel：匹配设备的信道值 * u8 \*key：与该匹配设备通信时，需使⽤用的密钥，⽬目前仅⽀支持 16 字节的密钥 * u8 key\_len：密钥⻓长度，⽬目前⻓长度仅⽀支持 16 字节 |
| 返回 | 0：成功  其它：失败 |

3.8.10. esp\_now\_del\_peer

|  |  |
| --- | --- |
| 功能 | 删除 ESP-NOW 匹配设备，将设备 MAC 地址从 ESP-NOW 维护的列列表中删除。 |
| 函数定义 | int esp\_now\_del\_peer(u8 \*mac\_addr) |
| 参数 | u8 \*mac\_addr：要删除设备的 MAC 地址 |
| 返回 | 0：成功  其它：失败 |

3.8.11. esp\_now\_set\_self\_role

|  |  |
| --- | --- |
| 功能 | 设置⾃自身 ESP-NOW 的⻆角⾊色 |
| 函数定义 | int esp\_now\_set\_self\_role(u8 role) |
| 参数 | u8 role：⻆角⾊色类型，详⻅见 esp\_now\_role |
| 返回 | 0：成功  其它：失败 |

3.8.12. esp\_now\_get\_self\_role

|  |  |
| --- | --- |
| 功能 | 查询⾃自身 ESP-NOW 的⻆角⾊色 |
| 函数定义 | u8 esp\_now\_get\_self\_role(void) |
| 参数 | ⽆无 |
| 返回 | ⻆角⾊色类型 |

3.8.13. esp\_now\_set\_peer\_role

|  |  |
| --- | --- |
| 功能 | 设置指定匹配设备的 ESP-NOW ⻆角⾊色。如果重复设置，新设置会覆盖原有设置。 |
| 函数定义 | int esp\_now\_set\_peer\_role(u8 \*mac\_addr，u8 role) |
| 参数 | • u8 \*mac\_addr：指定设备的 MAC 地址 • u8 role：⻆角⾊色类型，详⻅见 esp\_now\_role |
| 返回 | 0：成功  其它：失败 |

3.8.14. esp\_now\_get\_peer\_role

|  |  |
| --- | --- |
| 功能 | 查询指定匹配设备的 ESP-NOW ⻆角⾊色 |
| 函数定义 | int esp\_now\_get\_peer\_role(u8 \*mac\_addr) |
| 参数 | u8 \*mac\_addr：指定设备的 MAC 地址 |
| 返回 | ⻆角⾊色类型，详⻅见 esp\_now\_role  否则，失败 |

3.8.15. esp\_now\_set\_peer\_key

|  |  |
| --- | --- |
| 功能 | 设置指定匹配设备的 ESP-NOW 密钥。如果重复设置，新设置会覆盖原有设置。 |
| 函数定义 | int esp\_now\_set\_peer\_key(u8 \*mac\_addr，u8 \*key，u8 key\_len) |
| 参数 | * u8 \*mac\_addr：指定设备的 MAC 地址 * u8 \*key：密钥指针，⽬目前仅⽀支持 16 字节的密钥；如果传 NULL，则清除当前密钥 * u8 key\_len：密钥⻓长度，⽬目前⻓长度仅⽀支持 16 字节 |
| 返回 | 0：成功  其它：失败 |

3.8.16. esp\_now\_get\_peer\_key

|  |  |
| --- | --- |
| 功能 | 查询指定匹配设备的 ESP-NOW 密钥 |
| 函数定义 | int esp\_now\_set\_peer\_key(u8 \*mac\_addr，u8 \*key，u8 \*key\_len) |
| 参数 | * u8 \*mac\_addr：指定设备的 MAC 地址 * u8 \*key：查询到的密钥指针，请使⽤用 16 字节的 buffer 保存密钥 * u8 key\_len：查询到的密钥⻓长度 |
| 返回 | * 0：成功 * >0：找到⽬目标设备，但未获得 key * <0：失败 |

3.8.17. esp\_now\_set\_peer\_channel

|  |  |
| --- | --- |
| 功能 | 记录指定匹配设备的信道值。  当与该指定设备进⾏行行 ESP-NOW 通信时，   * 先调⽤用 esp\_now\_get\_peer\_channel 查询该设备所在信道； * 再调⽤用 wifi\_set\_channel 与该设备切换到同⼀一信道进⾏行行通信； * 通信完成后，请注意切换回原所在信道。 |
| 函数定义 | int esp\_now\_set\_peer\_channel(u8 \*mac\_addr，u8 channel) |
| 参数 | * u8 \*mac\_addr：指定设备的 MAC 地址 * u8 channel：信道值，⼀一般为 1 ~ 13，部分地区可能⽤用到 14 |
| 返回 | 0：成功  其它：失败 |

3.8.18. esp\_now\_get\_peer\_channel

|  |  |
| --- | --- |
| 功能 | 查询指定匹配设备的信道值。ESP-NOW 要求切换到同⼀一信道进⾏行行通信。 |
| 函数定义 | int esp\_now\_get\_peer\_channel(u8 \*mac\_addr) |
| 参数 | u8 \*mac\_addr：指定设备的 MAC 地址 |
| 返回 | 1 ~ 13（部分地区可能到 14），成功否则，失败 |

3.8.19. esp\_now\_is\_peer\_exist

|  |  |
| --- | --- |
| 功能 | 根据 MAC 地址判断设备是否存在 |
| 函数定义 | int esp\_now\_is\_peer\_exist(u8 \*mac\_addr) |
| 参数 | u8 \*mac\_addr：指定设备的 MAC 地址 |
| 返回 | * 0：设备不不存在 * >0：出错，查询失败 * <0：设备存在 |

3.8.20. esp\_now\_fetch\_peer

|  |  |
| --- | --- |
| 功能 | 查询当前指向的 ESP-NOW 配对设备的 MAC 地址，并将内部游标指向 ESP-NOW 维护列列表的后⼀一个设备或重新指向 ESP-NOW 维护列列表的第⼀一个设备。 |
| 注意 | * 本接⼝口不不可重⼊入。 * 第⼀一次调⽤用本接⼝口时，参数必须为 true，让内部游标指向 ESP-NOW 维护列列表的第⼀一个设备。 |
| 函数定义 | u8 \*esp\_now\_fetch\_peer(bool restart) |
| 参数 | bool restart   * true：将内部游标重新指向 ESP-NOW 维护列列表的第⼀一个设备 * false：将内部游标指向 ESP-NOW 维护列列表的后⼀一个设备 |
| 返回 | * NULL：不不存在已关联的 ESP-NOW 设备 * 否则，当前指向的 ESP-NOW 配对设备的 MAC 地址指针 |

3.8.21. esp\_now\_get\_cnt\_info

|  |  |
| --- | --- |
| 功能 | 查询已经匹配的全部设备总数和加密的设备总数。 |
| 函数定义 | int esp\_now\_get\_cnt\_info(u8 \*all\_cnt, u8 \*encryp\_cnt) |
| 参数 | * u8 \*all\_cnt：已经匹配的全部设备总数 * u8 \*encryp\_cnt：加密的设备总数 |
| 返回 | 0：成功  其它：失败 |

3.8.22. esp\_now\_set\_kok

|  |  |
| --- | --- |
| 功能 | 设置⽤用于将通信密钥加密的主密钥 (key of key)。所有设备的通信均共享同⼀一主密钥，如不不设置，则使⽤用默认主密钥给通信密钥加密。  如需调⽤用本接⼝口，请在 esp\_now\_add\_peer 和 esp\_now\_set\_peer\_key 接⼝口之前调⽤用。 |
| 函数定义 | int esp\_now\_set\_kok(u8 \*key, u8 len) |
| 参数 | * u8 \*key：主密钥指针 * u8 len：主密钥⻓长度，⽬目前⻓长度仅⽀支持 16 字节 |
| 返回 | 0：成功  其它：失败 |

#### 3.9. Simple-Pair 接⼝口

Simple-Pair 接⼝口位于 */ESP8266\_NONOS\_SDK/include/simple\_pair.h* 中。

3.9.1. 结构体

typedef enum {

SP\_ST\_STA\_FINISH = 0, // station 端协商结束

SP\_ST\_AP\_FINISH = 0, // AP 端协商结束

SP\_ST\_AP\_RECV\_NEG, // AP 收到 station 发来的协商请求

SP\_ST\_STA\_AP\_REFUSE\_NEG, // station 收到 AP 发来的协商拒绝

/\* definitions below are error codes \*/

SP\_ST\_WAIT\_TIMEOUT, // 错误：协商过程超时

SP\_ST\_SEND\_ERROR, // 错误：发送数据出错

SP\_ST\_KEY\_INSTALL\_ERR, // 错误：密钥安装错误

SP\_ST\_KEY\_OVERLAP\_ERR, // 错误：同一个 MAC 地址有多个密钥

SP\_ST\_OP\_ERROR, // 错误：操作错误

SP\_ST\_UNKNOWN\_ERROR, // 错误：未知错误

SP\_ST\_MAX,

} SP\_ST\_t;

3.9.2. register\_simple\_pair\_status\_cb

|  |  |
| --- | --- |
| 功能 | 注册 Simple-Pair 的状态回调函数 |
| 函数定义 | int register\_simple\_pair\_status\_cb(simple\_pair\_status\_cb\_t cb) |
| 回调函数定义 | typedef void (\*simple\_pair\_status\_cb\_t)(u8 \*sa, u8 status);   * u8 \*sa：对⽅方设备的源 MAC 地址 * u8 status：状态枚举值，详⻅见 SP\_ST\_t 定义 |
| 参数 | simple\_pair\_status\_cb\_t cb：状态回调函数 |
| 返回 | 0：成功  其它：失败 |

3.9.3. unregister\_simple\_pair\_status\_cb

|  |  |
| --- | --- |
| 功能 | 注销 Simple-Pair 的状态回调函数 |
| 函数定义 | void unregister\_simple\_pair\_status\_cb(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.9.4. simple\_pair\_init

|  |  |
| --- | --- |
| 功能 | 初始化 Simple-Pair 功能 |
| 函数定义 | int simple\_pair\_init(void) |
| 参数 | ⽆无 |
| 返回 | 0：成功  其它：失败 |

3.9.5. simple\_pair\_deinit

|  |  |
| --- | --- |
| 功能 | 反初始化 Simple-Pair 功能 |
| 函数定义 | void simple\_pair\_deinit(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.9.6. simple\_pair\_state\_reset

|  |  |
| --- | --- |
| 功能 | 重置 Simple-Pair 状态。当需要重新启动 Simple-Pair 时，可调⽤用本接⼝口重置状态。 |
| 函数定义 | int simple\_pair\_state\_reset(void) |
| 参数 | ⽆无 |
| 返回 | 0：成功  其它：失败 |

3.9.7. simple\_pair\_ap\_enter\_announce\_mode

|  |  |
| --- | --- |
| 功能 | Simple-Pair 的 AP 端进⼊入 announce 模式 |
| 函数定义 | int simple\_pair\_ap\_enter\_announce\_mode(void) |
| 参数 | ⽆无 |
| 返回 | 0：成功  其它：失败 |

3.9.8. simple\_pair\_sta\_enter\_scan\_mode

|  |  |
| --- | --- |
| 功能 | Simple-Pair 的 Station 端进⼊入 scan 模式 |
| 函数定义 | int simple\_pair\_sta\_enter\_scan\_mode(void) |
| 参数 | ⽆无 |
| 返回 | 0：成功  其它：失败 |

3.9.9. simple\_pair\_sta\_start\_negotiate

|  |  |
| --- | --- |
| 功能 | Simple-Pair 的 Station 端开始协商 |
| 函数定义 | int simple\_pair\_sta\_start\_negotiate(void) |
| 参数 | ⽆无 |
| 返回 | 0：成功  其它：失败 |

3.9.10. simple\_pair\_ap\_start\_negotiate

|  |  |
| --- | --- |
| 功能 | Simple-Pair 的 AP 端同意协商 |
| 函数定义 | int simple\_pair\_ap\_start\_negotiate(void) |
| 参数 | ⽆无 |
| 返回 | 0：成功  其它：失败 |

3.9.11. simple\_pair\_ap\_refuse\_negotiate

|  |  |
| --- | --- |
| 功能 | Simple-Pair 的 AP 端拒绝协商 |
| 函数定义 | int simple\_pair\_ap\_refuse\_negotiate(void) |
| 参数 | ⽆无 |
| 返回 | 0：成功  其它：失败 |

3.9.12. simple\_pair\_set\_peer\_ref

|  |  |
| --- | --- |
| 功能 | 设置需要协商的设备的参数。仅向系统设置信息，不不会安装 Key 或进⾏行行其他动作。   * 若作为 Station 端，则需要在 simple\_pair\_sta\_start\_negotiate 之前设置 * 若作为 AP 端，则需要在 simple\_pair\_ap\_start\_negotiate 或者 simple\_pair\_ap\_refuse\_negotiate 之前设置。 |
| 函数定义 | int simple\_pair\_set\_peer\_ref(u8 \*peer\_mac, u8 \*tmp\_key, u8 \*ex\_key) |
| 参数 | * u8 \*peer\_mac：需要协商的对端设备的 MAC 地址，⻓长度为 6 字节。不不能为 NULL * u8 \*tmp\_key：⽤用于加密 Simple-Pair 通信的临时密钥，⻓长度为 16 字节。不不能为 NULL * u8 \*ex\_key：需要交换的 Key，⻓长度为 16 字节。如果为 NULL，则系统默认使⽤用全 0 为 Key |
| 返回 | 0：成功  其它：失败 |

3.9.13. simple\_pair\_get\_peer\_ref

|  |  |
| --- | --- |
| 功能 | 获取设置的参数值。⽆无需获取的参数，传⼊入 NULL 即可。 |
| 函数定义 | int simple\_pair\_get\_peer\_ref(u8 \*peer\_mac, u8 \*tmp\_key, u8 \*ex\_key) |
| 参数 | * u8 \*peer\_mac：需要协商的设备 MAC 地址，⻓长度为 6 字节 * u8 \*tmp\_key：⽤用于加密 Simple-Pair 通信的临时密钥，⻓长度为 16 字节 * u8 \*ex\_key：需要交换的 Key，⻓长度为 16 字节。如未设置，则查询到全 0 值 |
| 返回 | 0：成功  其它：失败 |

#### 3.10. 云端升级 (FOTA) 接⼝口

云端升级接⼝口位于 */ESP8266\_NONOS\_SDK/include/user\_interface.h* 和 *upgrade.h* 中。

3.10.1. system\_upgrade\_userbin\_check

|  |  |
| --- | --- |
| 功能 | 查询 user bin |
| 函数定义 | uint8 system\_upgrade\_userbin\_check() |
| 参数 | ⽆无 |
| 返回 | 0x00 : UPGRADE\_FW\_BIN1, i.e. *user1.bin*  0x01 : UPGRADE\_FW\_BIN2, i.e. *user2.bin* |

3.10.2. system\_upgrade\_flag\_set

|  |  |
| --- | --- |
| 功能 | 设置升级状态标志 |
| 注意 | * 若调⽤用 system\_upgrade\_start 升级，本接⼝口⽆无需调⽤用。 * 若⽤用户调⽤用 spi\_flash\_write ⾃自⾏行行写 Flash 实现升级，新软件写⼊入完成后，将 flag 置为 UPGRADE\_FLAG\_FINISH， 再调⽤用 system\_upgrade\_reboot 重启运⾏行行新软件。 |
| 函数定义 | void system\_upgrade\_flag\_set(uint8 flag) |
| 参数 | uint8 flag:  #define UPGRADE\_FLAG\_IDLE 0x00  #define UPGRADE\_FLAG\_START 0x01  #define UPGRADE\_FLAG\_FINISH 0x02 |
| 返回 | ⽆无 |

3.10.3. system\_upgrade\_flag\_check

|  |  |
| --- | --- |
| 功能 | 查询升级状态标志 |
| 函数定义 | uint8 system\_upgrade\_flag\_check() |
| 参数 | ⽆无 |
| 返回 | #define UPGRADE\_FLAG\_IDLE 0x00  #define UPGRADE\_FLAG\_START 0x01  #define UPGRADE\_FLAG\_FINISH 0x02 |

3.10.4. 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：升级服务器器的相关参数 |
| 返回 | true：开始升级 false：已经在升级过程中，⽆无法开始升级 |

3.10.5. system\_upgrade\_reboot

|  |  |
| --- | --- |
| 功能 | 重启系统，运⾏行行新软件 |
| 函数定义 | void system\_upgrade\_reboot (void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

#### 3.11. Sniffer 相关接⼝口

Sniffer 接⼝口位于 */ESP8266\_NONOS\_SDK/include/user\_interface.h*。 3.11.1. wifi\_promiscuous\_enable

|  |  |
| --- | --- |
| 功能 | 开启混杂模式 (sniffer) |
| 注意 | * 仅⽀支持在 ESP8266 单 Station 模式下，开启混杂模式 * 混杂模式中，ESP8266 Station 和 SoftAP 接⼝口均失效 * 若开启混杂模式，请先调⽤用 wifi\_station\_disconnect 确保没有连接 * 混杂模式中请勿调⽤用其他 API，请先调⽤用 wifi\_promiscuous\_enable(0) 退出 sniffer |
| 函数定义 | void wifi\_promiscuous\_enable(uint8 promiscuous) |
| 参数 | uint8 promiscuous   * 0：关闭混杂模式 * 1：开启混杂模式 |
| 返回 | ⽆无 |
| 示例例 | ⽤用户可以向乐鑫申请 sniffer demo |

3.11.2. wifi\_promiscuous\_set\_mac

|  |  |
| --- | --- |
| 功能 | 设置 sniffer 模式时的 MAC 地址过滤，可过滤出发给指定 MAC 地址的包（也包含⼴广播包）。 |
| 注意 | * 本接⼝口需在 wifi\_promiscuous\_enable(1) 使能混杂模式后调⽤用； * MAC 地址过滤仅对当前这次的 sniffer 有效； * 如果停⽌止 sniffer，⼜又再次 sniffer，需要重新设置 MAC 地址过滤。 |
| 函数定义 | void wifi\_promiscuous\_set\_mac(const uint8\_t \*address) |
| 参数 | const uint8\_t \*address：MAC 地址 |
| 返回 | ⽆无 |
| 示例例 | char ap\_mac[6] = {0x16, 0x34, 0x56, 0x78, 0x90, 0xab}; wifi\_promiscuous\_set\_mac(ap\_mac); |

3.11.3. wifi\_set\_promiscuous\_rx\_cb

|  |  |
| --- | --- |
| 功能 | 注册混杂模式下的接收数据回调函数，每收到⼀一包数据，都会进⼊入注册的回调函数。 |
| 函数定义 | void wifi\_set\_promiscuous\_rx\_cb(wifi\_promiscuous\_cb\_t cb) |
| 参数 | wifi\_promiscuous\_cb\_t cb：回调函数 |
| 返回 | ⽆无 |

3.11.4. wifi\_get\_channel

|  |  |
| --- | --- |
| 功能 | 获取信道号 |
| 函数定义 | uint8 wifi\_get\_channel(void) |
| 参数 | ⽆无 |
| 返回 | 信道号 |

3.11.5. wifi\_set\_channel

|  |  |
| --- | --- |
| 功能 | 设置信道号，⽤用于混杂模式 |
| 函数定义 | bool wifi\_set\_channel (uint8 channel) |
| 参数 | uint8 channel：信道号 |
| 返回 | true：成功 false：失败 |

#### 3.12. SmartConfig 接⼝口

Smart Config 接⼝口位于 */ESP8266\_NONOS\_SDK/include/smartconfig.h*。

AirKiss 接⼝口位于 */ESP8266\_NONOS\_SDK/include/airkiss.h*。开启 SmartConfig 功能前，请先确保 AP 已经开启。

3.12.1. smartconfig\_start

|  |  |
| --- | --- |
| 功能 | 开启快连模式，快速连接 ESP8266 Station 到 AP。ESP8266 抓取空中特殊的数据包，包含⽬目标 AP 的 SSID 和 password 信息，同时，⽤用户需要通过⼿手机或者电脑⼴广播加密的 SSID 和 password 信息。 |
| 注意 | * 仅⽀支持在单 Station 模式下调⽤用本接⼝口 * SmartConfig 过程中，ESP8266 Station 和 SoftAP 失效 * smartconfig\_start 未完成之前不不可重复执⾏行行 smartconfig\_start，请先调⽤用 smartconfig\_stop 结束本次快连。 * SmartConfig 过程中，请勿调⽤用其他 API；先调⽤用 smartconfig\_stop，再使⽤用其他 API。 |
| 结构体 | typedef enum {  SC\_STATUS\_WAIT = 0, // 连接未开始，请勿在此阶段开始连接  SC\_STATUS\_FIND\_CHANNEL, // 请在此阶段开启 APP 进行配对连接  SC\_STATUS\_GETTING\_SSID\_PSWD,  SC\_STATUS\_LINK,  SC\_STATUS\_LINK\_OVER, // 获取到 IP，连接路由完成  } sc\_status; |
| 函数定义 | bool smartconfig\_start(sc\_callback\_t cb, uint8 log) |
| 参数 | * sc\_callback\_t cb：SmartConfig 状态发⽣生改变时，进⼊入回调函数。 * 传⼊入回调函数的参数 status 表示 SmartConfig 状态：   + 当 status 为 SC\_STATUS\_GETTING\_SSID\_PSWD 时，参数 void \*pdata 为 sc\_type \* 类型的指针变量量，表示此次配置是 AirKiss 还是 ESP-TOUCH；   + 当 status 为 SC\_STATUS\_LINK 时，参数 void \*pdata 为 struct station\_config 类型的指针变量量；   + 当 status 为 SC\_STATUS\_LINK\_OVER 时，参数 void \*pdata 是移动端的 IP 地址的指针，   4 个字节。（仅⽀支持在 ESP-TOUCH ⽅方式下，其他⽅方式则为 NULL）   * + 当 status 为其他状态时，参数 void \*pdata 为 NULL * uint8 log   + 1：UART 打印连接过程   + 否则：UART 仅打印连接结果。打印信息仅供调试使⽤用，正常⼯工作时，应避免   SmartConfig 过程中进⾏行行串串⼝口打印。 |
| 返回 | true：成功 false：失败 |
| 示例例 | void ICACHE\_FLASH\_ATTR  smartconfig\_done(sc\_status status, void \*pdata)  {  switch(status) {  case SC\_STATUS\_WAIT:  os\_printf("SC\_STATUS\_WAIT\n");  break;  case SC\_STATUS\_FIND\_CHANNEL:  os\_printf("SC\_STATUS\_FIND\_CHANNEL\n");  break;  case SC\_STATUS\_GETTING\_SSID\_PSWD:  os\_printf("SC\_STATUS\_GETTING\_SSID\_PSWD\n");  sc\_type \*type = pdata;  if (\*type == SC\_TYPE\_ESPTOUCH) {  os\_printf("SC\_TYPE:SC\_TYPE\_ESPTOUCH\n");  } else {  os\_printf("SC\_TYPE:SC\_TYPE\_AIRKISS\n");  }  break;  case SC\_STATUS\_LINK:  os\_printf("SC\_STATUS\_LINK\n");  struct station\_config \*sta\_conf = pdata;  wifi\_station\_set\_config(sta\_conf);  wifi\_station\_disconnect();  wifi\_station\_connect();  break;  case SC\_STATUS\_LINK\_OVER:  os\_printf("SC\_STATUS\_LINK\_OVER\n");  if (pdata != NULL) {  uint8 phone\_ip[4] = {0};  memcpy(phone\_ip, (uint8\*)pdata, 4);  os\_printf("Phone ip: %d.%d.%d.  %d\n",phone\_ip[0],phone\_ip[1],phone\_ip[2],phone\_ip[3]);  }  smartconfig\_stop();  break;  }  }  smartconfig\_start(smartconfig\_done); |

3.12.2. smartconfig\_stop

|  |  |
| --- | --- |
| 功能 | 关闭快连模式，释放 smartconfig\_start 占⽤用的内存。 |
| 注意 | * 若快连成功，连上⽬目标 AP 后，调⽤用本接⼝口释放 smartconfig\_start 占⽤用的内存。 * 若快连失败，调⽤用本接⼝口退出快连模式，释放占⽤用的内存。 |
| 函数定义 | bool smartconfig\_stop(void) |
| 参数 | ⽆无 |
| 返回 | true：成功 false：失败 |

3.12.3. smartconfig\_set\_type

|  |  |
| --- | --- |
| 功能 | 设置快连模式的协议类型。 |
| 注意 | 如需调⽤用本接⼝口，请在 smartconfig\_start 之前调⽤用。 |
| 函数定义 | bool smartconfig\_set\_type(sc\_type type) |
| 参数 | typedef enum {  SC\_TYPE\_ESPTOUCH = 0,  SC\_TYPE\_AIRKISS,  SC\_TYPE\_ESPTOUCH\_AIRKISS,  } sc\_type; |
| 返回 | true：成功 false：失败 |

3.12.4. airkiss\_version

|  |  |
| --- | --- |
| 功能 | 获得 AirKiss 库的版本信息。 |
| 注意 | 版本信息的实际⻓长度未知。 |
| 函数定义 | const char\* airkiss\_version(void) |
| 参数 | ⽆无 |
| 返回 | AirKiss 库的版本信息。 |

3.12.5. airkiss\_lan\_recv

|  |  |
| --- | --- |
| 功能 | ⽤用于 AirKiss 内⽹网发现功能，可参考微信官⽹网内⽹网发现功能介绍 [http://iot.weixin.qq.com](http://iot.weixin.qq.com/) 内⽹网发现功能⼤大致流程为：创建⼀一个 UDP 传输，在 UDP 的 espconn\_recv\_callback 中，将接收到的 UDP 报⽂文传⼊入 airkiss\_lan\_recv 函数，若函数返回 AIRKISS\_LAN\_SSDP\_REQ，则调⽤用 airkiss\_lan\_pack 打包响应报⽂文，通过 UDP 传输回复给发送⽅方。本函数⽤用于接收 AirKiss 发来的 UDP 数据包并解析。 |
| 函数定义 | int airkiss\_lan\_recv( const void\* body, unsigned short length, const airkiss\_config\_t\* config) |
| 参数 | * const void\* body：接收到的 UDP 数据 * unsigned short length：有效的数据⻓长度 * airkiss\_config\_t\* config：AirKiss 结构体 |
| 返回 | * >=0：成功 * <0：失败 * 具体可参考 airkiss\_lan\_ret\_t |

3.12.6. airkiss\_lan\_pack

|  |  |
| --- | --- |
| 功能 | ⽤用于 AirKiss 内⽹网发现功能，将⽤用户数据组织成 AirKiss 内⽹网探测的 UDP 数据包格式。 |
| 函数定义 | int airkiss\_lan\_pack(  airkiss\_lan\_cmdid\_t ak\_lan\_cmdid, void\* appid, void\* deviceid, void\* \_datain,  unsigned short inlength, void\* \_dataout,  unsigned short\* outlength, const airkiss\_config\_t\* config) |
| 参数 | * airkiss\_lan\_cmdid\_t ak\_lan\_cmdid：发包的类型 * void\* appid：微信公众号，必须从微信获得 * void\* deviceid：设备 ID 值，必须从微信获得 * void\* \_datain：待组包的⽤用户数据 * unsigned short inlength：⽤用户数据⻓长度 * void\* \_dataout：⽤用户数据完成 AirKiss 内⽹网探测组包后的数据 * unsigned short\* outlength：组包后的数据⻓长度 * const airkiss\_config\_t\* config：AirKiss 结构体 |
| 返回 | * >=0：成功 * <0：失败 * 具体可参考 airkiss\_lan\_ret\_t |

#### 3.13. SNTP 接⼝口

SNTP 接⼝口位于 */ESP8266\_NONOS\_SDK/include/sntp.h*。

3.13.1. sntp\_setserver

|  |  |
| --- | --- |
| 功能 | 通过 IP 地址设置 SNTP 服务器器，⼀一共 多⽀支持设置 3 个 SNTP 服务器器 |
| 函数定义 | void sntp\_setserver(unsigned char idx, ip\_addr\_t \*addr) |
| 参数 | unsigned char idx：SNTP 服务器器编号， 多⽀支持 3 个 SNTP 服务器器（0 ~ 2）；0 号为主服务器器，1 号和 2 号为备⽤用服务器器。  ip\_addr\_t \*addr：IP 地址；⽤用户需⾃自⾏行行确保，传⼊入的是合法 SNTP 服务器器 |
| 返回 | ⽆无 |

3.13.2. sntp\_getserver

|  |  |
| --- | --- |
| 功能 | 查询 SNTP 服务器器的 IP 地址，对应的设置接⼝口为：sntp\_setserver |
| 函数定义 | ip\_addr\_t sntp\_getserver(unsigned char idx) |
| 参数 | unsigned char idx：SNTP 服务器器编号， 多⽀支持 3 个 SNTP 服务器器（0 ~ 2） |
| 返回 | IP 地址 |

3.13.3. sntp\_setservername

|  |  |
| --- | --- |
| 功能 | 通过域名设置 SNTP 服务器器，⼀一共 多⽀支持设置 3 个 SNTP 服务器器 |
| 函数定义 | void sntp\_setservername(unsigned char idx, char \*server) |
| 参数 | unsigned char idx：SNTP 服务器器编号， 多⽀支持 3 个 SNTP 服务器器（0 ~ 2）；0 号为主服务器器，1 号和 2 号为备⽤用服务器器。  char \*server：域名；⽤用户需⾃自⾏行行确保，传⼊入的是合法 SNTP 服务器器 |
| 返回 | ⽆无 |

3.13.4. sntp\_getservername

|  |  |
| --- | --- |
| 功能 | 查询 SNTP 服务器器的域名，仅⽀支持查询通过 sntp\_setservername 设置的 SNTP 服务器器 |
| 函数定义 | char \* sntp\_getservername(unsigned char idx) |
| 参数 | unsigned char idx：SNTP 服务器器编号， 多⽀支持 3 个 SNTP 服务器器（0 ~ 2） |
| 返回 | 服务器器域名 |

3.13.5. sntp\_init

|  |  |
| --- | --- |
| 功能 | SNTP 初始化 |
| 函数定义 | void sntp\_init(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.13.6. sntp\_stop

|  |  |
| --- | --- |
| 功能 | SNTP 关闭 |
| 函数定义 | void sntp\_stop(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.13.7. sntp\_get\_current\_timestamp

|  |  |
| --- | --- |
| 功能 | 查询当前距离基准时间 (1970.01.01 00：00：00 GMT + 8) 的时间戳，单位：秒 |
| 函数定义 | uint32 sntp\_get\_current\_timestamp() |
| 参数 | ⽆无 |
| 返回 | 距离基准时间的时间戳 |

3.13.8. sntp\_get\_real\_time

|  |  |
| --- | --- |
| 功能 | 查询实际时间 (GMT + 8) |
| 函数定义 | char\* sntp\_get\_real\_time(long t) |
| 参数 | long t：与基准时间相距的时间戳 |
| 返回 | 实际时间 |

3.13.9. sntp\_set\_timezone

|  |  |
| --- | --- |
| 功能 | 设置时区信息 |
| 注意 | 调⽤用本接⼝口前，请先调⽤用 sntp\_stop |
| 函数定义 | bool sntp\_set\_timezone (sint8 timezone) |
| 参数 | sint8 timezone：时区值，参数范围：-11 ~ 13 |
| 返回 | true：成功 false：失败 |
| 示例例 | sntp\_stop();  if( true == sntp\_set\_timezone(-5) ) {  sntp\_init();  } |

3.13.10.sntp\_get\_timezone

|  |  |
| --- | --- |
| 功能 | 查询时区信息 |
| 函数定义 | sint8 sntp\_get\_timezone (void) |
| 参数 | ⽆无 |
| 返回 | 时区值，参数范围：-11 ~ 13 |

3.13.11.SNTP 示例例

Step 1. enable sntp

ip\_addr\_t \*addr = (ip\_addr\_t \*)os\_zalloc(sizeof(ip\_addr\_t)); sntp\_setservername(0, ”us.pool.ntp.org”); // set server 0 by domain name sntp\_setservername(1, ”ntp.sjtu.edu.cn”); // set server 1 by domain name ipaddr\_aton(“210.72.145.44”, addr); sntp\_setserver(2, addr); // set server 2 by IP address sntp\_init(); os\_free(addr);

Step 2. set a timer to check sntp timestamp

LOCAL os\_timer\_t sntp\_timer; os\_timer\_disarm(&sntp\_timer); os\_timer\_setfn(&sntp\_timer, (os\_timer\_func\_t \*)user\_check\_sntp\_stamp, NULL); os\_timer\_arm(&sntp\_timer, 100, 0);

Step 3. timer callback

|  |
| --- |
| void ICACHE\_FLASH\_ATTR user\_check\_sntp\_stamp(void \*arg){ uint32 current\_stamp; current\_stamp = sntp\_get\_current\_timestamp(); if(current\_stamp == 0){ os\_timer\_arm(&sntp\_timer, 100, 0);  } else{  os\_timer\_disarm(&sntp\_timer); os\_printf(“sntp: %d, %s \n”,current\_stamp,  sntp\_get\_real\_time(current\_stamp));  }  } |

#### 3.14. WPA2-Enterprise 接⼝口

ESP8266 Station 接⼝口⽀支持连接到 WPA2\_Enterprise 企业级加密的 AP。

WPA2\_Enterprise 接⼝口位于 */ESP8266\_NONOS\_SDK/include/wpa2\_enterprise.h*。

3.14.1. wifi\_station\_set\_wpa2\_enterprise\_auth

|  |  |
| --- | --- |
| 功能 | 使能 WPA2\_Enterprise 企业级加密的验证。   * 连接 WPA2\_Enterprise AP，需调⽤用 wifi\_station\_set\_wpa2\_enterprise\_auth(1); 使能验证。 * 之后如再需连接普通 AP，则需要先调⽤用 wifi\_station\_set\_wpa2\_enterprise\_auth(0); 清除状态。 |
| 函数定义 | int wifi\_station\_set\_wpa2\_enterprise\_auth(int enable) |
| 参数 | int enable   * 0：清除当前 WPA2\_Enterprise 状态； * ⾮非 0：使能 WPA2\_Enterprise 企业级加密的验证。 |
| 返回 | 0：成功  其它：失败 |

3.14.2. wifi\_station\_set\_enterprise\_cert\_key

|  |  |
| --- | --- |
| 功能 | 设置 ESP8266 Station 接⼝口连接 WPA2\_Enterprise AP 所需的⽤用户证书及密钥，⽤用于 EAP-TLS 认证。 |
| 注意 | * ⽀支持 WPA2-ENTERPRISE AP 需占⽤用 26 KB 以上的内存，调⽤用本接⼝口时请注意内存是否⾜足够。 * ⽬目前 WPA2-ENTERPRISE 只⽀支持⾮非加密的私钥⽂文件和证书⽂文件，且仅⽀支持 PEM 格式   + ⽀支持的证书⽂文件头信息为：- - - - - BEGIN CERTIFICATE - - - - -   + ⽀支持的私钥⽂文件头信息为：- - - - - BEGIN RSA PRIVATE KEY - - - - -  或者 - - - - - BEGIN PRIVATE KEY - - - - - * 请在连接 WPA2\_Enterprise AP 之前调⽤用本接⼝口设置私钥⽂文件和证书⽂文件，在成功连接 AP 后先调⽤用 wifi\_station\_clear\_enterprise\_cert\_key 清除内部状态，应⽤用层再释放私钥⽂文件和证书⽂文件信息。 * 如果遇到加密的私钥⽂文件，请使⽤用 openssl pkey 命令改为⾮非加密⽂文件使⽤用，或者使⽤用 openssl rsa 等命令，对某些私钥⽂文件进⾏行行加密-⾮非加密的转换（或起始 TAG 转化）。 |
| 函数定义 | int wifi\_station\_set\_enterprise\_cert\_key( u8 \*client\_cert, int client\_cert\_len, u8 \*private\_key, int private\_key\_len, u8 \*private\_key\_passwd, int private\_key\_passwd\_len) |
| 参数 | uint8 \*client\_cert：⼗十六进制数组的证书指针  int client\_cert\_len：证书⻓长度  uint8 \*private\_key：⼗十六进制数组的私钥指针，暂不不⽀支持超过 2048 的私钥 int private\_key\_len：私钥⻓长度，请勿超过 2048 uint8 \*private\_key\_passwd：私钥的提取密码，⽬目前暂不不⽀支持，请传⼊入 NULL int private\_key\_passwd\_len：提取密码的⻓长度，⽬目前暂不不⽀支持，请传⼊入 0 |
| 返回 | 0：成功  ⾮非 0：失败 |
| 示例例 | 假设私钥⽂文件的信息为 - - - - - BEGIN PRIVATE KEY - - - - - … … … …  那么对应的数组为：uint8 key[]={0x2d, 0x2d, 0x2d, 0x2d, 0x2d, 0x42, 0x45, 0x47, … …  0x00 }; 即各字符的 ASCII 码，请注意，数组必须添加 0x00 作为结尾。 |

3.14.3. wifi\_station\_clear\_enterprise\_cert\_key

|  |  |
| --- | --- |
| 功能 | 释放连接 WPA2\_Enterprise AP 使⽤用⽤用户证书和密钥所占⽤用的资源，并清除相关状态。 |
| 函数定义 | void wifi\_station\_clear\_enterprise\_cert\_key (void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

|  |  |
| --- | --- |
| 功能 | 设置连接 WPA2\_Enterprise AP 时，ESP8266 Station 的⽤用户名。 |

3.14.4. wifi\_station\_set\_enterprise\_ca\_cert

|  |  |
| --- | --- |
| 功能 | 设置 ESP8266 Station 接⼝口连接 WPA2\_Enterprise AP 使⽤用的根证书。EAP-TTLS/PEAP 认证⽅方法可选对根证书进⾏行行验证。 |
| 函数定义 | int wifi\_station\_set\_enterprise\_ca\_cert(u8 \*ca\_cert, int ca\_cert\_len) |
| 参数 | * u8 \*ca\_cert：⼗十六进制数组的根证书指针 * int ca\_cert\_len：根证书⻓长度 |
| 返回 | 0：成功  其它：失败 |

3.14.5. wifi\_station\_clear\_enterprise\_ca\_cert

|  |  |
| --- | --- |
| 功能 | 释放连接 WPA2\_Enterprise AP 使⽤用根证书占⽤用的资源，并清除相关状态。 |
| 函数定义 | void wifi\_station\_clear\_enterprise\_ca\_cert (void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.14.6. wifi\_station\_set\_enterprise\_username

|  |  |
| --- | --- |
| 注意 | WPA2\_Enterprise 企业级加密⽅方法调⽤用本接⼝口设置⽤用户身份，   * 对于 EAP-TTLS 以及 EAP-PEAP 认证必须设置，并且是⽤用在了了认证的第⼆二阶段，只有认证服务器器⽀支持的⽤用户身份才能通过认证。 * 对于 EAP-TLS 认证⽅方法则为可选项，即使没有设置⽤用户名，也可以通过匿匿名的身份通过验证。 |
| 函数定义 | int wifi\_station\_set\_enterprise\_username (u8 \*username, int len) |
| 参数 | * u8 \*username：⽤用户名称 * int len：名称⻓长度 |
| 返回 | 0：成功  其它：失败 |

3.14.7. wifi\_station\_clear\_enterprise\_username

|  |  |
| --- | --- |
| 功能 | 释放连接 WPA2\_Enterprise AP 设置⽤用户名占⽤用的资源，并清除相关状态。 |
| 函数定义 | void wifi\_station\_clear\_enterprise\_username (void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.14.8. wifi\_station\_set\_enterprise\_password

|  |  |
| --- | --- |
| 功能 | 设置⽤用户密码，⽤用于通过 EAP-TTLS/EAP-PEAP 认证⽅方法。 |
| 函数定义 | int wifi\_station\_set\_enterprise\_password (u8 \*password, int len) |
| 参数 | * u8 \*password：⽤用户密码 * int len：密码⻓长度 |
| 返回 | 0：成功  其它：失败 |

3.14.9. wifi\_station\_clear\_enterprise\_password

|  |  |
| --- | --- |
| 功能 | 释放连接 WPA2\_Enterprise AP 设置密码占⽤用的资源，并清除相关状态。 |
| 函数定义 | void wifi\_station\_clear\_enterprise\_password (void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.14.10.wifi\_station\_set\_enterprise\_new\_password

|  |  |
| --- | --- |
| 功能 | 设置新⽤用户密码，针对 MSCHAPV2 ⽅方法。 |
| 函数定义 | int wifi\_station\_set\_enterprise\_new\_password (u8 \*new\_password, int len) |
| 参数 | * u8 \*new\_password：新⽤用户密码 * int len：密码⻓长度 |
| 返回 | 0：成功  其它：失败 |

3.14.11.wifi\_station\_clear\_enterprise\_new\_password

|  |  |
| --- | --- |
| 功能 | 释放连接 WPA2\_Enterprise AP 设置新⽤用户密码占⽤用的资源，并清除相关状态。 |
| 函数定义 | void wifi\_station\_clear\_enterprise\_new\_password (void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

3.14.12.wifi\_station\_set\_enterprise\_disable\_time\_check

|  |  |
| --- | --- |
| 功能 | 设置认证时，是否检查过期时间。默认情况下，认证过程中不不检查过期时间。 |
| 函数定义 | void wifi\_station\_set\_enterprise\_disable\_time\_check(bool disable) |
| 参数 | bool disable   * true：不不检查过期时间 * false：检查过期时间，需调⽤用 wpa2\_enterprise\_set\_user\_get\_time 注册回调函数。 |
| 返回 | ⽆无 |

3.14.13.wifi\_station\_get\_enterprise\_disable\_time\_check

|  |  |
| --- | --- |
| 功能 | 查询认证时，是否检查过期时间。 |
| 函数定义 | bool wifi\_station\_get\_enterprise\_disable\_time\_check (void) |
| 参数 | ⽆无 |
| 返回 | true：不不检查过期时间 false：检查过期时间 |

3.14.14.wpa2\_enterprise\_set\_user\_get\_time

|  |  |
| --- | --- |
| 功能 | 设置认证过程中从⽤用户获得时间的回调函数。  需调⽤用 wifi\_station\_set\_enterprise\_disable\_time\_check(false); 使能检查过期时间。 |
| 函数定义 | void wpa2\_enterprise\_set\_user\_get\_time(get\_time\_func\_t cb) |
| 参数 | get\_time\_func\_t cb：回调函数 |
| 返回 | ⽆无 |
| 示例例 | static int sys\_get\_current\_time(struct os\_time \*t)  {  t->sec = CURRENT\_TIME; // User set current time.  return 0;  } //Set Callback  wpa2\_enterprise\_set\_user\_get\_time(sys\_get\_current\_time); //Enable Time check  wifi\_station\_set\_enterprise\_disable\_time\_check(false); |

3.14.15.示例例流程如需连接 WPA2\_Enterprise 加密的 AP，流程如下：

1. wifi\_station\_set\_config 配置需连接的 AP 信息。
2. wifi\_station\_set\_wpa2\_enterprise\_auth(1); 使能 WPA2\_Enterprise 加密验证。
   1. 若为 EAP-TLS 认证，调⽤用 wifi\_station\_set\_enterprise\_cert\_key 设置证书和密钥。可选调⽤用 wifi\_station\_set\_enterprise\_username 设置⽤用户名。
   2. 若为 EAP-TTLS 或 EAP-PEAP 认证，调⽤用

wifi\_station\_set\_enterprise\_username 以及

wifi\_station\_set\_enterprise\_password，设置⽤用户名和密码。可选调⽤用 wifi\_station\_set\_enterprise\_ca\_cert 设置根证书。

1. wifi\_station\_connnect 连接 AP。
2. 成功连接 AP 或连接 AP 失败并不不再重试后，调⽤用

wifi\_station\_clear\_enterprise\_XXX 对应接⼝口释放资源。

## 4. TCP/UDP 接⼝口

位于 *ESP8266\_NONOS\_SDK/include/espconn.h*。

⽹网络相关接⼝口可分为以下⼏几类：

* **通⽤用接⼝口：**TCP 和 UDP 均可以调⽤用的接⼝口。
* **TCP APIs：**仅建⽴立 TCP 连接时，使⽤用的接⼝口。
* **UDP APIs：**仅收发 UDP 包时，使⽤用的接⼝口。
* **mDNS APIs：**mDNS 相关接⼝口。

### 4.1. 通⽤用接⼝口

4.1.1. espconn\_delete

|  |  |
| --- | --- |
| 功能 | 删除传输连接。 |
| 注意 | 对应创建传输的接⼝口如下：  TCP：espconn\_accept  UDP：espconn\_create |
| 函数定义 | sint8 espconn\_delete(struct espconn \*espconn) |
| 参数 | struct espconn \*espconn：对应⽹网络传输的结构体 |
| 返回 | 0：成功其它：失败，返回错误码   * ESPCONN\_ARG：未找到参数 espconn 对应的⽹网络传输 * ESPCONN\_INPROGRESS：参数 espconn 对应的⽹网络连接仍未断开，请先调⽤用 espconn\_disconnect 断开连接，再进⾏行行删除。 |

4.1.2. espconn\_gethostbyname

|  |  |
| --- | --- |
| 功能 | DNS 功能 |
| 函数定义 | err\_t espconn\_gethostbyname(  struct espconn \*pespconn,   const char \*hostname,   ip\_addr\_t \*addr,   dns\_found\_callback found ) |
| 参数 | * struct espconn \*espconn：对应⽹网络传输的结构体 * const char \*hostname：域名字符串串的指针 * ip\_addr\_t \*addr：IP 地址 * dns\_found\_callback found：DNS 回调函数 |
| 返回 | err\_t   * ESPCONN\_OK：成功 * ESPCONN\_ISCONN：失败，错误码含义：已经连接 * ESPCONN\_ARG：失败，错误码含义：未找到参数 espconn 对应的⽹网络传输 |
| 示例例 | 请参考 *IoT\_Demo*：  ip\_addr\_t esp\_server\_ip; LOCAL void ICACHE\_FLASH\_ATTR  user\_esp\_platform\_dns\_found(const char \*name, ip\_addr\_t \*ipaddr, void \*arg) {  struct espconn \*pespconn = (struct espconn \*)arg; if (ipaddr != NULL)  os\_printf(user\_esp\_platform\_dns\_found %d.%d.%d.%d/n,  \*((uint8 \*)&ipaddr->addr), \*((uint8 \*)&ipaddr->addr + 1),  \*((uint8 \*)&ipaddr->addr + 2), \*((uint8 \*)&ipaddr->addr + 3));  }  void dns\_test(void) {  espconn\_gethostbyname(pespconn,“iot.espressif.cn”, &esp\_server\_ip,  user\_esp\_platform\_dns\_found); } |

4.1.3. espconn\_port

|  |  |
| --- | --- |
| 功能 | 获取 ESP8266 可⽤用的端⼝口 |
| 函数定义 | uint32 espconn\_port(void) |
| 参数 | ⽆无 |
| 返回 | 端⼝口号 |

4.1.4. espconn\_regist\_sentcb

|  |  |
| --- | --- |
| 功能 | 注册⽹网络数据发送成功的回调函数 |
| 函数定义 | sint8 espconn\_regist\_sentcb(  struct espconn \*espconn,   espconn\_sent\_callback sent\_cb ) |
| 参数 | struct espconn \*espconn：对应⽹网络传输的结构体 espconn\_connect\_callback connect\_cb：成功接收⽹网络数据的回调函数 |
| 返回 | 0：成功  其它：失败，返回错误码 ESPCONN\_ARG：未找到参数 espconn 对应的⽹网络传输 |

4.1.5. espconn\_regist\_recvcb

|  |  |
| --- | --- |
| 功能 | 注册成功接收⽹网络数据的回调函数 |
| 函数定义 | sint8 espconn\_regist\_recvcb(  struct espconn \*espconn,   espconn\_recv\_callback recv\_cb ) |
| 参数 | struct espconn \*espconn：对应⽹网络传输的结构体 espconn\_connect\_callback connect\_cb：成功接收⽹网络数据的回调函数 |
| 返回 | 0：成功  其它：失败，返回错误码 ESPCONN\_ARG：未找到参数 espconn 对应的⽹网络传输 |

|  |  |
| --- | --- |
| 功能 | 查询某个 TCP 连接或者 UDP 传输的远端信息。⼀一般在 espconn\_recv\_callback 中调⽤用。 |
| 函数定义 | sint8 espconn\_get\_connection\_info(  struct espconn \*espconn,   remot\_info \*\*pcon\_info,   uint8 typeflags  ) |

4.1.6. espconn\_sent\_callback

|  |  |
| --- | --- |
| 功能 | ⽹网络数据发送成功的回调函数，由 espconn\_regist\_sentcb 注册 |
| 函数定义 | void espconn\_sent\_callback (void \*arg) |
| 参数 | void \*arg：回调函数的参数，⽹网络传输的结构体 espconn 指针。  注意，本指针为底层维护的指针，不不同回调传⼊入的指针地址可能不不⼀一样，请勿依此判断⽹网络连接。可根据 espconn 结构体中的 remote\_ip, remote\_port 判断多连接中的不不同⽹网络传输。 |
| 返回 | ⽆无 |

4.1.7. espconn\_recv\_callback

|  |  |
| --- | --- |
| 功能 | 成功接收⽹网络数据的回调函数，由 espconn\_regist\_recvcb 注册 |
| 函数定义 | void espconn\_recv\_callback (  void \*arg,   char \*pdata,   unsigned short len  ) |
| 参数 | * void \*arg：回调函数的参数，⽹网络传输结构体 espconn 指针。注意，本指针为底层维护的指针，不不同回调传⼊入的指针地址可能不不⼀一样，请勿依此判断⽹网络连接。可根据 espconn 结构体中的 remote\_ip、remote\_port 判断多连接中的不不同⽹网络传输。 * char \*pdata：接收到的数据 * unsigned short len：接收到的数据⻓长度 |
| 返回 | ⽆无 |

4.1.8. espconn\_get\_connection\_info

|  |  |
| --- | --- |
| 参数 | * struct espconn \*espconn：对应⽹网络连接的结构体 * remot\_info \*\*pcon\_info：连接 client 信息 * uint8 typeflags   + 0：正常 server   + 1：SSL server |
| 返回 | 0：成功  其它：失败，返回错误码 ESPCONN\_ARG：未找到参数 espconn 对应的 TCP 连接 |
| 示例例 | void user\_udp\_recv\_cb(void \*arg, char \*pusrdata, unsigned short length) {  struct espconn \*pesp\_conn = arg;  remot\_info \*premot = NULL;  if (espconn\_get\_connection\_info(pesp\_conn,&premot,0) == ESPCONN\_OK){  pesp\_conn->proto.tcp->remote\_port = premot->remote\_port;  pesp\_conn->proto.tcp->remote\_ip[0] = premot->remote\_ip[0];  pesp\_conn->proto.tcp->remote\_ip[1] = premot->remote\_ip[1];  pesp\_conn->proto.tcp->remote\_ip[2] = premot->remote\_ip[2];  pesp\_conn->proto.tcp->remote\_ip[3] = premot->remote\_ip[3]; espconn\_sent(pesp\_conn, pusrdata, os\_strlen(pusrdata));  }  } |

4.1.9. espconn\_send

|  |  |
| --- | --- |
| 功能 | 通过 WiFi 发送数据 |
| 注意 | * ⼀一般情况，请在前⼀一包数据发送成功，进⼊入 espconn\_sent\_callback 后，再调⽤用 espconn\_send 发送下⼀一包数据。 * 如果是 UDP 传输，请在每次调⽤用 espconn\_send 前，设置 espconn->proto.udp->remote\_ip 和 remote\_port 参数，因为 UDP ⽆无连接，远端信息可能被更更改。 |
| 函数定义 | sint8 espconn\_send(  struct espconn \*espconn,   uint8 \*psent,   uint16 length  ) |
| 参数 | struct espconn \*espconn：对应⽹网络传输的结构体 uint8 \*psent：发送的数据 uint16 length：发送的数据⻓长度 |
| 返回 | 0：成功其它：失败，返回错误码   * ESPCONN\_ARG：未找到参数 espconn 对应的⽹网络传输 * ESPCONN\_MEM：空间不不⾜足 * ESPCONN\_MAXNUM：底层发包缓存已满，发包失败 * ESPCONN\_IF：UDP 发包失败 |

4.1.10. espconn\_sent

[@deprecated] 本接⼝口不不建议使⽤用，建议使⽤用 espconn\_send 代替。

|  |  |
| --- | --- |
| 功能 | 通过 WiFi 发送数据 |
| 注意 | * ⼀一般情况，请在前⼀一包数据发送成功，进⼊入 espconn\_sent\_callback 后，再调⽤用 espconn\_send 发送下⼀一包数据。 * 如果是 UDP 传输，请在每次调⽤用 espconn\_send 前，设置 espconn->proto.udp->remote\_ip 和 remote\_port 参数，因为 UDP ⽆无连接，远端信息可能被更更改。 |
| 函数定义 | sint8 espconn\_sent(  struct espconn \*espconn,   uint8 \*psent,   uint16 length  ) |
| 参数 | struct espconn \*espconn：对应⽹网络传输的结构体 uint8 \*psent：发送的数据 uint16 length：发送的数据⻓长度 |
| 返回 | 0：成功其它：失败，返回错误码   * ESPCONN\_ARG：未找到参数 espconn 对应的⽹网络传输 * ESPCONN\_MEM：空间不不⾜足 * ESPCONN\_MAXNUM：底层发包缓存已满，发包失败 * ESPCONN\_IF：UDP 发包失败 |

#### 4.2. TCP 接⼝口

TCP 接⼝口仅⽤用于 TCP 连接，请勿⽤用于 UDP 传输。

4.2.1. espconn\_accept

|  |  |
| --- | --- |
| 功能 | 创建 TCP server，建⽴立侦听 |
| 函数定义 | sint8 espconn\_accept(struct espconn \*espconn) |
| 参数 | struct espconn \*espconn：对应⽹网络连接的结构体 |
| 返回 | 0：成功其它：失败，返回错误码   * ESPCONN\_ARG：未找到参数 espconn 对应的 TCP 连接 * ESPCONN\_MEM：空间不不⾜足 * ESPCONN\_ISCONN：连接已经建⽴立 |

4.2.2. espconn\_regist\_time

|  |  |
| --- | --- |
| 功能 | 注册 ESP8266 TCP server 超时时间，时间值仅作参考，并不不精确。 |
| 注意 | * 请在 espconn\_accept 之后，连接未建⽴立之前，调⽤用本接⼝口。本接⼝口不不能⽤用于 SSL 连接。 * 如果超时时间设置为 0，ESP8266 TCP server 将始终不不会断开已经不不与它通信的 TCP client，不不建议这样使⽤用。 |
| 函数定义 | sint8 espconn\_regist\_time(  struct espconn \*espconn,   uint32 interval,   uint8 type\_flag  ) |
| 参数 | * struct espconn \*espconn：对应⽹网络连接的结构体 * uint32 interval：超时时间，单位：秒， ⼤大值：7200 秒 * uint8 type\_flag   + 0：对所有 TCP 连接⽣生效   + 1：仅对某⼀一 TCP 连接⽣生效 |
| 返回 | 0：成功  其它：失败，返回错误码 ESPCONN\_ARG：未找到参数 espconn 对应的 TCP 连接 |

4.2.3. espconn\_connect

|  |  |
| --- | --- |
| 功能 | 连接 TCP server（ESP8266 作为 TCP client）。 |
| 注意 | * 如果 espconn\_connect 失败，返回⾮非零值，连接未建⽴立，不不会进⼊入任何 espconn callback。 * 建议使⽤用 espconn\_port 接⼝口，设置⼀一个可⽤用的端⼝口号。 |
| 函数定义 | sint8 espconn\_connect(struct espconn \*espconn) |
| 参数 | struct espconn \*espconn：对应⽹网络连接的结构体 |
| 返回 | 0：成功其它：失败，返回错误码   * ESPCONN\_ARG：未找到参数 espconn 对应的 TCP 连接 * ESPCONN\_MEM：空间不不⾜足 * ESPCONN\_ISCONN：连接已经建⽴立 * ESPCONN\_RTE：路路由异常 |

4.2.4. espconn\_regist\_connectcb

|  |  |
| --- | --- |
| 功能 | 注册 TCP 连接成功建⽴立后的回调函数。 |
| 函数定义 | sint8 espconn\_regist\_connectcb(  struct espconn \*espconn,  espconn\_connect\_callback connect\_cb ) |
| 参数 | * struct espconn \*espconn：对应⽹网络连接的结构体 * espconn\_connect\_callback connect\_cb：成功建⽴立 TCP 连接后的回调函数 |
| 返回 | 0：成功  其它：失败，返回错误码 ESPCONN\_ARG：未找到参数 espconn 对应的 TCP 连接 |

4.2.5. espconn\_connect\_callback

|  |  |
| --- | --- |
| 功能 | 成功建⽴立 TCP 连接的回调函数，由 espconn\_regist\_connectcb 注册。ESP8266 作为 TCP server 侦听到 TCP client 连⼊入；或者 ESP8266 作为 TCP client 成功与 TCP server 建⽴立连接。 |
| 函数定义 | void espconn\_connect\_callback (void \*arg) |
| 参数 | * void \*arg：回调函数的参数，对应⽹网络连接的结构体 espconn 指针。 * 注意，本指针为底层维护的指针，不不同回调传⼊入的指针地址可能不不⼀一样，请勿依此判断⽹网络   连接。可根据 espconn 结构体中的 remote\_ip, remote\_port 判断多连接中的不不同⽹网络传输。 |
| 返回 | ⽆无 |

4.2.6. espconn\_set\_opt

|  |  |
| --- | --- |
| 功能 | 设置 TCP 连接的相关配置，对应清除配置标志位的接⼝口为 espconn\_clear\_opt |
| 注意 | * SSL 连接不不⽀支持使⽤用本接⼝口 * ⼀一般情况下，⽆无需调⽤用本接⼝口；如需设置 espconn\_set\_opt 请在 espconn\_connect\_callback 中调⽤用 |
| 函数定义 | sint8 espconn\_set\_opt(  struct espconn \*espconn,   uint8 opt ) |
| 参数 | * struct espconn \*espconn：对应⽹网络连接的结构体 * uint8 opt：TCP 连接的相关配置，参考 espconn\_option   + bit 0：1，TCP 连接断开时，及时释放内存，⽆无需等待 2 分钟才释放占⽤用内存；   + bit 1：1，关闭 TCP 数据传输时的 nalge 算法；   + bit 2：1，使能 write finish callback，进⼊入此回调表示 espconn\_send 要发送的数据已经写⼊入 2920 字节的 write buffer 等待发送或已经发送；   + bit 3：1，使能 keep alive； |
| 返回 | 0：成功  其它：失败，返回错误码 ESPCONN\_ARG：未找到参数 espconn 对应的 TCP 连接 |

4.2.7. espconn\_clear\_opt

|  |  |
| --- | --- |
| 功能 | 清除 TCP 连接的相关配置 |
| 函数定义 | sint8 espconn\_clear\_opt(  struct espconn \*espconn,   uint8 opt ) |
| 结构体 | enum espconn\_option{  ESPCONN\_START = 0x00，  ESPCONN\_REUSEADDR = 0x01，  ESPCONN\_NODELAY = 0x02，  ESPCONN\_COPY = 0x04，  ESPCONN\_KEEPALIVE = 0x08，  ESPCONN\_END  } |
| 参数 | * struct espconn \*espconn：对应⽹网络连接的结构体 * uint8 opt：清除 TCP 连接的相关配置，参考 espconn\_option |
| 返回 | 0：成功  其它：失败，返回错误码 ESPCONN\_ARG：未找到参数 espconn 对应的 TCP 连接 |

4.2.8. espconn\_set\_keepalive

|  |  |
| --- | --- |
| 功能 | 设置 TCP keep alive 的参数 |
| 注意 | * ⼀一般情况下，不不需要调⽤用本接⼝口 * 如果设置，请在 espconn\_connect\_callback 中调⽤用，并先设置 espconn\_set\_opt 使能 keep alive |
| 函数定义 | sint8 espconn\_set\_keepalive(struct espconn \*espconn, uint8 level, void\* optarg) |
| 结构体 | enum espconn\_level{  ESPCONN\_KEEPIDLE,  ESPCONN\_KEEPINTVL,  ESPCONN\_KEEPCNT  } |
| 参数 | * struct espconn \*espconn：对应⽹网络连接的结构体 * uint8 level：默认设置为每隔 ESPCONN\_KEEPIDLE 时⻓长进⾏行行⼀一次 keep alive 探查，如果报⽂文   ⽆无响应，则每隔 ESPCONN\_KEEPINTVL 时⻓长探查⼀一次， 多探查 ESPCONN\_KEEPCNT 次；若始  终⽆无响应，则认为⽹网络连接断开，释放本地连接相关资源，进⼊入  espconn\_reconnect\_callback。注意，时间间隔设置并不不可靠精准，仅供参考，受其他⾼高优先级任务执⾏行行的影响。参数说明如下：   * + ESPCONN\_KEEPIDLE：设置进⾏行行 keep alive 探查的时间间隔，单位：秒   + ESPCONN\_KEEPINTVL：keep alive 探查过程中，报⽂文的时间间隔，单位：秒   + ESPCONN\_KEEPCNT：每次 keep alive 探查，发送报⽂文的 ⼤大次数 * void\* optarg：设置参数值 |
| 返回 | 0：成功  其它：失败，返回错误码 ESPCONN\_ARG：未找到参数 espconn 对应的 TCP 连接 |

4.2.9. espconn\_get\_keepalive

|  |  |
| --- | --- |
| 功能 | 查询 TCP keep alive 的参数 |
| 函数定义 | sint8 espconn\_set\_keepalive(struct espconn \*espconn, uint8 level, void\* optarg) |
| 结构体 | enum espconn\_level{  ESPCONN\_KEEPIDLE,  ESPCONN\_KEEPINTVL,  ESPCONN\_KEEPCNT  } |
| 参数 | * struct espconn \*espconn：对应⽹网络连接的结构体 * uint8 level   + ESPCONN\_KEEPIDLE：设置进⾏行行 keep alive 探查的时间间隔，单位：秒   + ESPCONN\_KEEPINTVL：keep alive 探查过程中，报⽂文的时间间隔，单位：秒   + ESPCONN\_KEEPCNT：每次 keep alive 探查，发送报⽂文的 ⼤大次数 * void\* optarg：设置参数值 |
| 返回 | 0：成功  其它：失败，返回错误码 ESPCONN\_ARG：未找到参数 espconn 对应的 TCP 连接 |

4.2.10. espconn\_reconnect\_callback

|  |  |
| --- | --- |
| 功能 | TCP 连接异常断开时的回调函数，相当于出错处理理回调，由 espconn\_regist\_reconcb 注册。 |
| 函数定义 | void espconn\_reconnect\_callback (void \*arg, sint8 err) |
| 参数 | * void \*arg：回调函数的参数，对应⽹网络连接的结构体 espconn 指针。注意，本指针为底层维护的指针，不不同回调传⼊入的指针地址可能不不⼀一样，请勿依此判断⽹网络连接。可根据 espconn 结构体中的 remote\_ip、remote\_port 判断多连接中的不不同⽹网络传输。 * sint8 err：异常断开的错误码。   + ESCONN\_TIMEOUT：超时出错断开   + ESPCONN\_ABRT：TCP 连接异常断开   + ESPCONN\_RST：TCP 连接复位断开   + ESPCONN\_CLSD：TCP 连接在断开过程中出错，异常断开   + ESPCONN\_CONN：TCP 未连接成功   + ESPCONN\_HANDSHAKE：TCP SSL 握⼿手失败   + ESPCONN\_PROTO\_MSG：SSL 应⽤用数据处理理异常 |
| 返回 | ⽆无 |

4.2.11. espconn\_regist\_reconcb

|  |  |
| --- | --- |
| 功能 | 注册 TCP 连接发⽣生异常断开时的回调函数，可以在回调函数中进⾏行行重连。 |
| 注意 | espconn\_reconnect\_callback 功能类似于出错处理理回调，任何阶段出错时，均会进⼊入此回调。例例如，espconn\_sent 失败，则认为⽹网络连接异常，也会进⼊入 espconn\_reconnect\_callback；⽤用户可在 espconn\_reconnect\_callback 中⾃自⾏行行定义出错处理理。 |
| 函数定义 | sint8 espconn\_regist\_reconcb(  struct espconn \*espconn,  espconn\_reconnect\_callback recon\_cb ) |
| 参数 | * struct espconn \*espconn：对应⽹网络连接的结构体 * espconn\_reconnect\_callback recon\_cb：回调函数 |
| 返回 | 0：成功其它：失败，返回错误码 ESPCONN\_ARG：未找到参数 espconn 对应的 TCP 连接 |

4.2.12. espconn\_disconnect

|  |  |
| --- | --- |
| 功能 | 断开 TCP 连接 |
| 注意 | 请勿在 espconn 的任何 callback 中调⽤用本接⼝口断开连接。如有需要，可以在 callback 中使⽤用任务触发调⽤用本接⼝口断开连接。 |
| 函数定义 | sint8 espconn\_disconnect(struct espconn \*espconn) |
| 参数 | struct espconn \*espconn：对应⽹网络连接的结构体 |
| 返回 | 0：成功  其它：失败，返回错误码 ESPCONN\_ARG：未找到参数 espconn 对应的 TCP 连接 |

4.2.13. espconn\_regist\_disconcb

|  |  |
| --- | --- |
| 功能 | 注册 TCP 连接正常断开成功的回调函数 |
| 函数定义 | sint8 espconn\_regist\_disconcb(  struct espconn \*espconn,  espconn\_connect\_callback discon\_cb ) |
| 参数 | * struct espconn \*espconn：对应⽹网络连接的结构体 * espconn\_connect\_callback connect\_cb：回调函数 |
| 返回 | 0：成功其它：失败，返回错误码 ESPCONN\_ARG：未找到参数 espconn 对应的 TCP 连接 |

4.2.14. espconn\_abort

|  |  |
| --- | --- |
| 功能 | 强制断开 TCP 连接 |
| 注意 | 请勿在 espconn 的任何 callback 中调⽤用本接⼝口断开连接。如有需要，可以在 callback 中使⽤用任务触发调⽤用本接⼝口断开连接。 |
| 函数定义 | sint8 espconn\_abort(struct espconn \*espconn) |
| 参数 | struct espconn \*espconn：对应⽹网络连接的结构体 |
| 返回 | 0：成功  其它：失败，返回错误码 ESPCONN\_ARG：未找到参数 espconn 对应的 TCP 连接 |

4.2.15. espconn\_regist\_write\_finish

|  |  |
| --- | --- |
| 功能 | 注册所有需发送的数据均成功写⼊入 write buffer 后的回调函数。  请先调⽤用 espconn\_set\_opt 使能 write buffer。 |
| 注意 | * 本接⼝口不不能⽤用于 SSL 连接。 * write buffer ⽤用于缓存 espconn\_send 将发送的数据， 多缓存 8 包数据，write buffer 的容量量为 2920 字节。 * 由 espconn\_set\_opt 设置使能 write\_finish\_callback 回调。 * 对发送速度有要求时，可以在 write\_finish\_callback 中调⽤用 espconn\_send 发送下⼀一包，   ⽆无需等到 espconn\_sent\_callback |
| 函数定义 | sint8 espconn\_regist\_write\_finish (  struct espconn \*espconn,  espconn\_connect\_callback write\_finish\_fn ) |
| 参数 | struct espconn \*espconn：对应⽹网络连接的结构体 espconn\_connect\_callback write\_finish\_fn：回调函数 |
| 返回 | 0：成功  其它：失败，返回错误码 ESPCONN\_ARG：未找到参数 espconn 对应的 TCP 连接 |

4.2.16. espconn\_tcp\_get\_max\_con

|  |  |
| --- | --- |
| 功能 | 查询允许的 TCP ⼤大连接数。 |
| 函数定义 | uint8 espconn\_tcp\_get\_max\_con(void) |
| 参数 | ⽆无 |
| 返回 | 允许的 TCP ⼤大连接数 |

4.2.17. espconn\_tcp\_set\_max\_con

|  |  |
| --- | --- |
| 功能 | 设置允许的 TCP ⼤大连接数。在内存⾜足够的情况下，建议不不超过 10。默认值为 5。 |
| 函数定义 | sint8 espconn\_tcp\_set\_max\_con(uint8 num) |
| 参数 | uint8 num：允许的 TCP ⼤大连接数 |
| 返回 | 0：成功  其它：失败，返回错误码 ESPCONN\_ARG：未找到参数 espconn 对应的 TCP 连接 |

4.2.18. espconn\_tcp\_get\_max\_con\_allow

|  |  |
| --- | --- |
| 功能 | 查询 ESP8266 某个 TCP server 多允许连接的 TCP client 数⽬目 |
| 函数定义 | sint8 espconn\_tcp\_get\_max\_con\_allow(struct espconn \*espconn) |
| 参数 | struct espconn \*espconn：对应 TCP server 的结构体 |
| 返回 | >0： 多允许连接的 TCP client 数⽬目  <0：失败，返回错误码 ESPCONN\_ARG：未找到参数 espconn 对应的 TCP 连接 |

4.2.19. espconn\_tcp\_set\_max\_con\_allow

|  |  |
| --- | --- |
| 功能 | 设置 ESP8266 某个 TCP server 多允许连接的 TCP client 数⽬目 |
| 函数定义 | sint8 espconn\_tcp\_set\_max\_con\_allow(struct espconn \*espconn, uint8 num) |
| 参数 | struct espconn \*espconn：对应 TCP server 的结构体 uint8 num：允许的 TCP ⼤大连接数 |
| 返回 | 0：成功  其它：失败，返回错误码 ESPCONN\_ARG：未找到参数 espconn 对应的 TCP 连接 |

4.2.20. espconn\_recv\_hold

|  |  |
| --- | --- |
| 功能 | 阻塞 TCP 接收数据 |
| 注意 | 调⽤用本接⼝口会逐渐减⼩小 TCP 的窗⼝口，并不不是即时阻塞，因此建议预留留 1460\*5 字节左右的空间时候调⽤用，且本接⼝口可以反复调⽤用。 |
| 函数定义 | sint8 espconn\_recv\_hold(struct espconn \*espconn) |
| 参数 | struct espconn \*espconn：对应⽹网络连接的结构体 |
| 返回 | 0：成功其它：失败，返回错误码 ESPCONN\_ARG：未找到参数 espconn 对应的 TCP 连接 |

4.2.21. espconn\_recv\_unhold

|  |  |
| --- | --- |
| 功能 | 解除 TCP 收包阻塞，即对应的阻塞接⼝口 espconn\_recv\_hold |
| 注意 | 本接⼝口实时⽣生效。 |
| 函数定义 | sint8 espconn\_recv\_unhold(struct espconn \*espconn) |
| 参数 | struct espconn \*espconn：对应⽹网络连接的结构体 |
| 返回 | 0：成功  其它：失败，返回错误码 ESPCONN\_ARG：未找到参数 espconn 对应的 TCP 连接 |

4.2.22. espconn\_secure\_accept

|  |  |
| --- | --- |
| 功能 | 创建 SSL server，侦听 SSL 握⼿手 |
| 注意 | * ⽬目前仅⽀支持建⽴立⼀一个 SSL server，本接⼝口只能调⽤用⼀一次，并且仅⽀支持连⼊入⼀一个 SSL client。 * 如果 SSL 加密⼀一包数据⼤大于 espconn\_secure\_set\_size 设置的缓存空间，ESP8266 ⽆无法处理理，SSL 连接断开，进⼊入 espconn\_reconnect\_callback。 * SSL 相关接⼝口与普通 TCP 接⼝口底层处理理不不⼀一致，请不不要混⽤用。SSL 连接时，仅⽀支持使⽤用 espconn\_secure\_XXX 系列列接⼝口和 espconn\_regist\_XXXcb 系列列注册回调函数的接⼝口，以及 espconn\_port 获得⼀一个空闲端⼝口。 * 如需创建 SSL server，必须先调⽤用 espconn\_secure\_set\_default\_certificate 和 espconn\_secure\_set\_default\_private\_key 传⼊入证书和密钥。 |
| 函数定义 | sint8 espconn\_secure\_accept(struct espconn \*espconn) |
| 参数 | struct espconn \*espconn：对应⽹网络连接的结构体 |
| 返回 | 0：成功其它：失败，返回错误码   * ESPCONN\_ARG：未找到参数 espconn 对应的 SSL 连接 * ESPCONN\_MEM：空间不不⾜足 * ESPCONN\_ISCONN：连接已经建⽴立 |

4.2.23. espconn\_secure\_delete

|  |  |
| --- | --- |
| 功能 | 删除 ESP8266 作为 SSL server 的连接。 |
| 函数定义 | sint8 espconn\_secure\_delete(struct espconn \*espconn) |
| 参数 | struct espconn \*espconn：对应⽹网络连接的结构体 |
| 返回 | 0：成功其它：失败，返回错误码   * ESPCONN\_ARG：未找到参数 espconn 对应的 SSL 连接 * ESPCONN\_INPROGRESS：参数 espconn 对应的 SSL 连接仍未断开，请先调⽤用 espconn\_secure\_disconnect 断开连接，再进⾏行行删除。 |

4.2.24. espconn\_secure\_set\_size

|  |  |
| --- | --- |
| 功能 | 设置加密 (SSL) 数据缓存空间的⼤大⼩小 |
| 注意 | 默认缓存⼤大⼩小为 2KB；如需更更改，请在加密 (SSL) 连接建⽴立前调⽤用：  在 espconn\_secure\_accept（ESP8266 作为 SSL server）之前调⽤用；或者 espconn\_secure\_connect（ESP8266 作为 SSL client）之前调⽤用 |
| 函数定义 | bool espconn\_secure\_set\_size (uint8 level, uint16 size) |
| 参数 | * uint8 level：设置 ESP8266 SSL server/client   + 0x01：SSL client   + 0x02：SSL server   + 0x03：SSL client 和 SSL server * uint16 size：加密数据缓存的空间⼤大⼩小，取值范围：1 ~ 8192，单位：字节，默认值为   2048 |
| 返回 | true：成功 false：失败 |

4.2.25. espconn\_secure\_get\_size

|  |  |
| --- | --- |
| 功能 | 查询加密 (SSL) 数据缓存空间的⼤大⼩小 |
| 函数定义 | sint16 espconn\_secure\_get\_size (uint8 level) |
| 参数 | • uint8 level：设置 ESP8266 SSL server/client   * 0x01：SSL client * 0x02：SSL server * 0x03：SSL client 和 SSL server |
| 返回 | 加密 (SSL) 数据缓存空间的⼤大⼩小 |

4.2.26. espconn\_secure\_connect

|  |  |
| --- | --- |
| 功能 | 加密 (SSL) 连接到 TCP SSL server（ESP8266 作为 TCP SSL client） |
| 注意 | * 如果 espconn\_secure\_connect 失败，返回⾮非零值，连接未建⽴立，不不会进⼊入任何 espconn callback。 * ⽬目前 ESP8266 作为 SSL client 仅⽀支持⼀一个连接，本接⼝口只能调⽤用⼀一次，或者调⽤用 espconn\_secure\_disconnect 断开前⼀一次连接，才可以再次调⽤用本接⼝口建⽴立 SSL 连接； * 如果 SSL 加密⼀一包数据⼤大于 espconn\_secure\_set\_size 设置的缓存空间，ESP8266 ⽆无法处理理，SSL 连接断开，进⼊入 espconn\_reconnect\_callback * SSL 相关接⼝口与普通 TCP 接⼝口底层处理理不不⼀一致，请不不要混⽤用。SSL 连接时，仅⽀支持使⽤用 espconn\_secure\_XXX 系列列接⼝口和 espconn\_regist\_XXXcb 系列列注册回调函数的接⼝口，以及 espconn\_port 获得⼀一个空闲端⼝口。 |
| 函数定义 | sint8 espconn\_secure\_connect (struct espconn \*espconn) |
| 参数 | struct espconn \*espconn：对应⽹网络连接的结构体 |
| 返回 | 0：成功其它：失败，返回错误码   * ESPCONN\_ARG：未找到参数 espconn 对应的 SSL 连接 * ESPCONN\_MEM：空间不不⾜足 * ESPCONN\_ISCONN：连接已经建⽴立 |

4.2.27. espconn\_secure\_send

|  |  |
| --- | --- |
| 功能 | 发送加密数据 (SSL) |
| 注意 | * 请在上⼀一包数据发送完成，进⼊入 espconn\_sent\_callback 后，再发下⼀一包数据。 * 每⼀一包数据明⽂文的上限值为 1024 字节，加密后的报⽂文上限值是 1460 字节。 |
| 函数定义 | sint8 espconn\_secure\_send (  struct espconn \*espconn,   uint8 \*psent,   uint16 length  ) |
| 参数 | struct espconn \*espconn：对应⽹网络连接的结构体 uint8 \*psent：发送的数据 uint16 length：发送的数据⻓长度 |
| 返回 | 0：成功  其它：失败，返回错误码 ESPCONN\_ARG：未找到参数 espconn 对应的 SSL 连接 |

4.2.28. espconn\_secure\_sent

[@deprecated] 本接⼝口不不建议使⽤用，建议使⽤用 espconn\_secure\_send 代替。

|  |  |
| --- | --- |
| 功能 | 发送加密数据 (SSL) |
| 注意 | * 请在上⼀一包数据发送完成，进⼊入 espconn\_sent\_callback 后，再发下⼀一包数据。 * 每⼀一包数据明⽂文的上限值为 1024 字节，加密后的报⽂文上限值是 1460 字节。 |
| 函数定义 | sint8 espconn\_secure\_sent (  struct espconn \*espconn,   uint8 \*psent,   uint16 length  ) |
| 参数 | struct espconn \*espconn：对应⽹网络连接的结构体 uint8 \*psent：发送的数据 uint16 length：发送的数据⻓长度 |
| 返回 | 0：成功其它：失败，返回错误码 ESPCONN\_ARG：未找到参数 espconn 对应的 SSL 连接 |

4.2.29. espconn\_secure\_disconnect

|  |  |
| --- | --- |
| 功能 | 断开加密连接 (SSL) |
| 注意 | 请勿在 espconn 的任何 callback 中调⽤用本接⼝口断开连接。如有需要，可以在 callback 中使⽤用任务触发调⽤用本接⼝口断开连接。 |
| 函数定义 | sint8 espconn\_secure\_disconnect(struct espconn \*espconn) |
| 参数 | struct espconn \*espconn：对应⽹网络连接的结构体 |
| 返回 | 0：成功其它：失败，返回错误码 ESPCONN\_ARG：未找到参数 espconn 对应的 SSL 连接 |

4.2.30. espconn\_secure\_ca\_enable

|  |  |
| --- | --- |
| 功能 | 开启 SSL CA 认证功能 |
| 注意 | * CA 认证功能，默认关闭，详细介绍可参考⽂文档 [ESP8266 SSL 加密使⽤用⼿手册](https://www.espressif.com/sites/default/files/documentation/5a-esp8266_sdk_ssl_user_manual_cn.pdf)。 * 如需调⽤用本接⼝口，请在加密 (SSL) 连接建⽴立前调⽤用：   + 在 espconn\_secure\_accept（ESP8266 作为 SSL server）之前调⽤用；   + 或者 espconn\_secure\_connect（ESP8266 作为 SSL client）之前调⽤用 |
| 函数定义 | bool espconn\_secure\_ca\_enable (uint8 level, uint32 flash\_sector) |
| 参数 | * uint8 level：设置 ESP8266 SSL server/client   + 0x01：SSL client   + 0x02：SSL server   + 0x03：SSL client 和 SSL server * uint32 flash\_sector：设置 CA 证书 *esp\_ca\_cert.bin* 烧录到 Flash 的位置。例例如，参数传⼊入 0x3B，则对应烧录到 Flash 0x7B000 |
| 返回 | true：成功 false：失败 |

|  |  |
| --- | --- |
| 功能 | 关闭 ESP8266 作为 SSL client 时的证书认证功能 |

4.2.31. espconn\_secure\_ca\_disable

|  |  |
| --- | --- |
| 功能 | 关闭 SSL CA 认证功能 |
| 注意 | * CA 认证功能，默认关闭，详细介绍可参考⽂文档 [ESP8266 SSL 加密使⽤用⼿手册](https://www.espressif.com/sites/default/files/documentation/5a-esp8266_sdk_ssl_user_manual_cn.pdf)。 * 如需调⽤用本接⼝口，请在加密 (SSL) 连接建⽴立前调⽤用：   + 在 espconn\_secure\_accept（ESP8266 作为 SSL server）之前调⽤用；   + 或者 espconn\_secure\_connect（ESP8266 作为 SSL client）之前调⽤用 |
| 函数定义 | bool espconn\_secure\_ca\_disable (uint8 level) |
| 参数 | • uint8 level：设置 ESP8266 SSL server/client   * 0x01：SSL client * 0x02：SSL server * 0x03：SSL client 和 SSL server |
| 返回 | true：成功 false：失败 |

4.2.32. espconn\_secure\_cert\_req\_enable

|  |  |
| --- | --- |
| 功能 | 使能 ESP8266 作为 SSL client 时的证书认证功能 |
| 注意 | * 证书认证功能，默认关闭。如果服务器器端不不要求认证证书，则⽆无需调⽤用本接⼝口。 * 如需调⽤用本接⼝口，请在 espconn\_secure\_connect 之前调⽤用。 |
| 函数定义 | bool espconn\_secure\_cert\_req\_enable (uint8 level, uint32 flash\_sector) |
| 参数 | * uint8 level：仅⽀支持设置为 0x01 ESP8266 作为 SSL client * uint32 flash\_sector：设置密钥 *esp\_cert\_private\_key.bin* 烧录到 Flash 的位置，例例如，参数传⼊入 0x7A，则对应烧录到 Flash 0x7A000。请注意，不不要覆盖了了代码或系统参数区域。 |
| 返回 | true：成功 false：失败 |

4.2.33. espconn\_secure\_cert\_req\_disable

|  |  |
| --- | --- |
| 注意 | 证书认证功能，默认关闭。 |
| 函数定义 | bool espconn\_secure\_ca\_disable (uint8 level) |
| 参数 | uint8 level：仅⽀支持设置为 0x01 ESP8266 作为 SSL client |
| 返回 | true：成功 false：失败 |

4.2.34. espconn\_secure\_set\_default\_certificate

|  |  |
| --- | --- |
| 功能 | 设置 ESP8266 作为 SSL server 时的证书 |
| 注意 | * *ESP8266\_NONOS\_SDK/examples/IoT\_Demo*中提供使⽤用示例例 * 本接⼝口必须在 espconn\_secure\_accept 之前调⽤用，传⼊入证书信息 |
| 函数定义 | bool espconn\_secure\_set\_default\_certificate (const uint8\_t\* certificate, uint16\_t  length) |
| 参数 | const uint8\_t\* certificate：证书指针 uint16\_t length：证书⻓长度 |
| 返回 | true：成功 false：失败 |

4.2.35. espconn\_secure\_set\_default\_private\_key

|  |  |
| --- | --- |
| 功能 | 设置 ESP8266 作为 SSL server 时的密钥 |
| 注意 | * *ESP8266\_NONOS\_SDK/examples/IoT\_Demo* 中提供使⽤用示例例 * 本接⼝口必须在 espconn\_secure\_accept 之前调⽤用，传⼊入密钥信息 |
| 函数定义 | bool espconn\_secure\_set\_default\_private\_key (const uint8\_t\* key, uint16\_t length) |
| 参数 | const uint8\_t\* key：密钥指针 uint16\_t length：密钥⻓长度 |
| 返回 | true：成功 false：失败 |

#### 4.3. UDP 接⼝口

4.3.1. espconn\_create

|  |  |
| --- | --- |
| 功能 | 建⽴立 UDP 传输。 |
| 注意 | 请注意设置 remote\_ip 和 remote\_port 参数，请勿设置为 0。 |
| 函数定义 | sin8 espconn\_create(struct espconn \*espconn) |
| 参数 | struct espconn \*espconn：对应⽹网络连接的结构体 |
| 返回 | 0：成功其它：失败，返回错误码   * ESPCONN\_ARG：未找到参数 espconn 对应的 UDP 连接 * ESPCONN\_MEM：空间不不⾜足 * ESPCONN\_ISCONN：连接已经建⽴立 |

4.3.2. espconn\_sendto

|  |  |
| --- | --- |
| 功能 | UDP 发包接⼝口 |
| 函数定义 | sin16 espconn\_sendto(struct espconn \*espconn, uint8 \*psent, uint16 length) |
| 参数 | struct espconn \*espconn：对应⽹网络连接的结构体 uint8 \*psent：待发送的数据 uint16 length：发送的数据⻓长度 |
| 返回 | 0：成功其它：失败，返回错误码   * ESPCONN\_ARG：未找到参数 espconn 对应的 UDP 传输 * ESPCONN\_MEM：空间不不⾜足 * ESPCONN\_IF：UDP 发包失败 |

4.3.3. espconn\_igmp\_join

|  |  |
| --- | --- |
| 功能 | 加⼊入多播组 |
| 注意 | 请在 ESP8266 Station 已连⼊入路路由的情况下调⽤用。 |
| 函数定义 | sint8 espconn\_igmp\_join(ip\_addr\_t \*host\_ip, ip\_addr\_t \*multicast\_ip) |
| 参数 | ip\_addr\_t \*host\_ip：主机 IP ip\_addr\_t \*multicast\_ip：多播组 IP |
| 返回 | 0：成功  其它：失败，返回错误码 ESPCONN\_MEM：空间不不⾜足 |

4.3.4. espconn\_igmp\_leave

|  |  |
| --- | --- |
| 功能 | 退出多播组 |
| 函数定义 | sint8 espconn\_igmp\_leave(ip\_addr\_t \*host\_ip, ip\_addr\_t \*multicast\_ip) |
| 参数 | ip\_addr\_t \*host\_ip：主机 IP  ip\_addr\_t \*multicast\_ip：多播组 IP |
| 返回 | 0：成功  其它：失败，返回错误码 ESPCONN\_MEM：空间不不⾜足 |

4.3.5. espconn\_dns\_setserver

|  |  |
| --- | --- |
| 功能 | 设置默认 DNS server |
| 注意 | 本接⼝口必须在 ESP8266 DHCP client 关闭 wifi\_station\_dhcpc\_stop 的情况下使⽤用。 |
| 函数定义 | void espconn\_dns\_setserver(uint8 numdns, ip\_addr\_t \*dnsserver) |
| 参数 | uint8 numdns：DNS server ID，⽀支持设置两个 DNS server，ID 分别为 0 和 1 ip\_addr\_t \*dnsserver：DNS server IP |
| 返回 | ⽆无 |

4.3.6. espconn\_dns\_getserver

|  |  |
| --- | --- |
| 功能 | 查询 DNS server IP |
| 函数定义 | ip\_addr\_t espconn\_dns\_getserver(uint8 numdns) |
| 参数 | uint8 numdns：DNS server ID，⽀支持传⼊入 0 或 1 |
| 返回 | DNS server IP |

#### 4.4. mDNS 接⼝口

4.4.1. espconn\_mdns\_init

|  |  |
| --- | --- |
| 功能 | mDNS 初始化 |
| 注意 | * 若为 SoftAP+Station 模式，请先调⽤用 wifi\_set\_broadcast\_if(STATIONAP\_MODE); * 若使⽤用 ESP8266 Station 接⼝口，请获得 IP 后，再调⽤用本接⼝口初始化 mDNS * txt\_data 必须为 key = value 的形式 |
| 结构体 | struct mdns\_info{  char \*host\_name;  char \*server\_name;  uint16 server\_port;  unsigned long ipAddr;  char \*txt\_data[10];  }; |
| 函数定义 | void espconn\_mdns\_init(struct mdns\_info \*info) |
| 参数 | struct mdns\_info \*info：mDNS 结构体 |
| 返回 | ⽆无 |

4.4.2. espconn\_mdns\_close

|  |  |
| --- | --- |
| 功能 | 关闭 mDNS，对应开启 mDNS 的 API：espconn\_mdns\_init |
| 函数定义 | void espconn\_mdns\_close(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

4.4.3. espconn\_mdns\_server\_register

|  |  |
| --- | --- |
| 功能 | 注册 mDNS 服务器器 |
| 函数定义 | void espconn\_mdns\_server\_register(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

4.4.4. espconn\_mdns\_server\_unregister

|  |  |
| --- | --- |
| 功能 | 注销 mDNS 服务器器 |
| 函数定义 | void espconn\_mdns\_server\_unregister(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

4.4.5. espconn\_mdns\_get\_servername

|  |  |
| --- | --- |
| 功能 | 查询 mDNS 服务器器名称 |
| 函数定义 | char\* espconn\_mdns\_get\_servername(void) |
| 参数 | ⽆无 |
| 返回 | 服务器器名称 |

4.4.6. espconn\_mdns\_set\_servername

|  |  |
| --- | --- |
| 功能 | 设置 mDNS 服务器器名称 |
| 函数定义 | void espconn\_mdns\_set\_servername(const char \*name) |
| 参数 | const char \*name：服务器器名称 |
| 返回 | ⽆无 |

4.4.7. espconn\_mdns\_set\_hostname

|  |  |
| --- | --- |
| 功能 | 设置 mDNS 主机名称 |
| 函数定义 | void espconn\_mdns\_set\_hostname(char \*name) |
| 参数 | char \*name：主机名称 |
| 返回 | ⽆无 |

4.4.8. espconn\_mdns\_get\_hostname

|  |  |
| --- | --- |
| 功能 | 查询 mDNS 主机名称 |
| 函数定义 | char\* espconn\_mdns\_get\_hostname(void) |
| 参数 | ⽆无 |
| 返回 | 主机名称 |

4.4.9. espconn\_mdns\_disable

|  |  |
| --- | --- |
| 功能 | 去能 mDNS，对应使能 API：espconn\_mdns\_enable |
| 函数定义 | void espconn\_mdns\_disable(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

4.4.10. espconn\_mdns\_enable

|  |  |
| --- | --- |
| 功能 | 使能 mDNS |
| 函数定义 | void espconn\_mdns\_enable(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

4.4.11. mDNS 示例例

定义 mDNS 信息时，请注意 host\_name 和 server\_name 不不能包含特殊字符（例例如“.”符号），或者协议名称（例例如不不能定义为“http”）。

|  |
| --- |
| struct mdns\_info info; void user\_mdns\_config() {  struct ip\_info ipconfig; wifi\_get\_ip\_info(STATION\_IF, &ipconfig); info->host\_name = “espressif”;  info->ipAddr = ipconfig.ip.addr; //ESP8266 station IP  info->server\_name = “iot”;  info->server\_port = 8080;  info->txt\_data[0] = “version = now”;  info->txt\_data[1] = “user1 = data1”;  info->txt\_data[2] = “user2 = data2”;  espconn\_mdns\_init(&info);  } |

### 5. 应⽤用相关接⼝口

#### 5.1. AT 接⼝口

AT 接⼝口位于 */ESP8266\_NONOS\_SDK/include/at\_custom.h*。

AT 接⼝口的使⽤用示例例，请参考 *ESP8266\_NONOS\_SDK/examples/at/user/user\_main.c*。

5.1.1. at\_response\_ok

|  |  |
| --- | --- |
| 功能 | AT 串串⼝口 (UART0) 输出 OK |
| 函数定义 | void at\_response\_ok(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

5.1.2. at\_response\_error

|  |  |
| --- | --- |
| 功能 | AT 串串⼝口 (UART0) 输出 ERROR |
| 函数定义 | void at\_response\_error(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

5.1.3. at\_cmd\_array\_regist

|  |  |
| --- | --- |
| 功能 | 注册⽤用户⾃自定义的 AT 指令。请仅调⽤用⼀一次，将所有⽤用户⾃自定义 AT 指令⼀一并注册。 |
| 函数定义 | void at\_cmd\_array\_regist (  at\_function \* custom\_at\_cmd\_arrar,   uint32 cmd\_num ) |
| 参数 | at\_function \* custom\_at\_cmd\_arrar：⽤用户⾃自定义的 AT 指令数组 uint32 cmd\_num：⽤用户⾃自定义的 AT 指令数⽬目 |
| 返回 | ⽆无 |
| 示例例 | 请参考 *ESP8266\_NONOS\_SDK/examples/at/user/user\_main.c* |

5.1.4. at\_get\_next\_int\_dec

|  |  |
| --- | --- |
| 功能 | 从 AT 指令⾏行行中解析 int 型数字 |
| 函数定义 | bool at\_get\_next\_int\_dec (char \*\*p\_src,int\* result,int\* err) |
| 参数 | * char \*\*p\_src：参数 \*p\_src 为接收到的 AT 指令字符串串 * int\* result：从 AT 指令中解析出的 int 型数字 * int\* err：解析处理理时的错误码   + 1：数字省略略时，返回错误码 1   + 3：只发现’-’时，返回错误码 3 |
| 返回 | true：正常解析到数字（数字省略略时，仍然返回 true，但错误码会为 1）；  false：解析异常，返回错误码；异常可能：数字超过 10 bytes，遇到 \r 结束符，只发现 - 字符。 |
| 示例例 | 请参考 *ESP8266\_NONOS\_SDK/examples/at/user/user\_main.c* |

5.1.5. at\_data\_str\_copy

|  |  |
| --- | --- |
| 功能 | 从 AT 指令⾏行行中解析字符串串 |
| 函数定义 | int32 at\_data\_str\_copy (char \* p\_dest, char \*\* p\_src,int32 max\_len) |
| 参数 | * char \* p\_dest：从 AT 指令⾏行行中解析到的字符串串 * char \*\*p\_src：参数 \*p\_src 为接收到的 AT 指令字符串串 * int32 max\_len：允许的 ⼤大字符串串⻓长度 |
| 返回 | 解析到的字符串串⻓长度：  >=0：成功，则返回解析到的字符串串⻓长度  <0：失败，返回 -1 |
| 示例例 | 请参考 *ESP8266\_NONOS\_SDK/examples/at/user/user\_main.c* |

5.1.6. at\_init

|  |  |
| --- | --- |
| 功能 | AT 初始化 |
| 函数定义 | void at\_init (void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |
| 示例例 | 请参考 *ESP8266\_NONOS\_SDK/examples/at/user/user\_main.c* |

5.1.7. at\_port\_print

|  |  |
| --- | --- |
| 功能 | 从 AT 串串⼝口 (UART0) 输出字符串串 |
| 函数定义 | void at\_port\_print(const char \*str) |
| 参数 | const char \*str：字符串串 |
| 返回 | ⽆无 |
| 示例例 | 请参考 *ESP8266\_NONOS\_SDK/examples/at/user/user\_main.c* |

5.1.8. at\_set\_custom\_info

|  |  |
| --- | --- |
| 功能 | 开发者⾃自定义 AT 版本信息，可由指令 AT+GMR 查询到。 |
| 函数定义 | void at\_set\_custom\_info (char \*info) |
| 参数 | char \*info：版本信息 |
| 返回 | ⽆无 |

5.1.9. at\_enter\_special\_state

|  |  |
| --- | --- |
| 功能 | 进⼊入 AT 指令执⾏行行态，此时不不响应其他 AT 指令，返回 busy |
| 函数定义 | void at\_enter\_special\_state (void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

5.1.10. at\_leave\_special\_state

|  |  |
| --- | --- |
| 功能 | 退出 AT 指令执⾏行行态 |
| 函数定义 | void at\_leave\_special\_state (void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

5.1.11. at\_get\_version

|  |  |
| --- | --- |
| 功能 | 查询乐鑫提供的 AT lib 版本号 |
| 函数定义 | uint32 at\_get\_version (void) |
| 参数 | ⽆无 |
| 返回 | 乐鑫 AT lib 版本号 |

5.1.12. at\_register\_uart\_rx\_intr

|  |  |
| --- | --- |
| 功能 | 设置 UART0 RX 是由⽤用户使⽤用，还是由 AT 使⽤用。 |
| 注意 | * 本接⼝口可以重复调⽤用。 * 运⾏行行 AT BIN，UART0 RX 默认供 AT 使⽤用。 |
| 函数定义 | void at\_register\_uart\_rx\_intr (at\_custom\_uart\_rx\_intr rx\_func) |
| 参数 | at\_custom\_uart\_rx\_intr：注册⽤用户使⽤用 UART0 的 Rx 中断处理理函数；如果传 NULL，则切换为 AT 使⽤用 UART0 |
| 返回 | ⽆无 |
| 示例例 | void user\_uart\_rx\_intr (uint8\* data, int32 len)  {  // UART0 rx for user  os\_printf(“len=%d \r\n”,len);  os\_printf(data);    // change UART0 for AT  at\_register\_uart\_rx\_intr(NULL);  }  void user\_init(void) {  at\_register\_uart\_rx\_intr(user\_uart\_rx\_intr); } |

5.1.13. at\_response

|  |  |
| --- | --- |
| 功能 | 设置 AT 响应 |
| 注意 | * 默认情况下，at\_response 从 UART0 TX 输出，与 at\_port\_print 功能相同。 * 如果调⽤用了了 at\_register\_response\_func，at\_response 的字符串串成为 response\_func 的参数，由⽤用户⾃自⾏行行处理理。 |
| 函数定义 | void at\_response (const char \*str) |
| 参数 | const char \*str：字符串串 |
| 返回 | ⽆无 |

5.1.14. at\_register\_response\_func

|  |  |
| --- | --- |
| 功能 | 注册 at\_response 的回调函数。调⽤用了了 at\_register\_response\_func，at\_response 的字符串串将传⼊入 response\_func，由⽤用户⾃自⾏行行处理理。 |
| 函数定义 | void at\_register\_response\_func (at\_custom\_response\_func\_type response\_func) |
| 参数 | at\_custom\_response\_func\_type：at\_response 的回调函数 |
| 返回 | ⽆无 |

5.1.15. at\_fake\_uart\_enable

|  |  |
| --- | --- |
| 功能 | 使能模拟 UART，开发者可⽤用于实现⽹网络 AT 指令，或者 SDIO AT 指令。 |
| 函数定义 | bool at\_fake\_uart\_enable(bool enable, at\_fake\_uart\_tx\_func\_type func) |
| 参数 | bool enable：使能模拟 UART at\_fake\_uart\_tx\_func\_type func：模拟 UART Tx 的回调函数 |
| 返回 | true：成功 false：失败 |

5.1.16. at\_fake\_uart\_rx

|  |  |
| --- | --- |
| 功能 | 模拟 UART RX，开发者可⽤用于实现⽹网络 AT 指令，或者 SDIO AT 指令。 |
| 函数定义 | uint32 at\_fake\_uart\_rx(uint8\* data, uint32 length) |
| 参数 | uint8\* data：模拟 UART Rx 收到的数据 uint32 length：数据⻓长度 |
| 返回 | 如果执⾏行行成功，则返回值与 length 相同；否则，执⾏行行失败。 |

5.1.17. at\_set\_escape\_character

|  |  |
| --- | --- |
| 功能 | 设置 AT 指令的转义字符，⽀支持设置为符号 !、#、$、@、&、\ 的其中之⼀一，默认转义字符为  \。 |
| 函数定义 | bool at\_set\_escape\_character(uint8 ch) |
| 参数 | uint8 ch：转义字符，⽀支持传⼊入符号 !、#、$、@、&、\ 的其中之⼀一。 |
| 返回 | true：成功 false：失败 |

#### 5.2. JSON 接⼝口

位于 *ESP8266\_NONOS\_SDK/include/json/jsonparse.h & jsontree.h*。

5.2.1. jsonparse\_setup

|  |  |
| --- | --- |
| 功能 | JSON 解析初始化 |
| 函数定义 | void jsonparse\_setup(  struct jsonparse\_state \*state,   const char \*json,   int len  ) |
| 参数 | struct jsonparse\_state \*state：JSON 解析指针 const char \*json：JSON 解析字符串串 int len：字符串串⻓长度 |
| 返回 | ⽆无 |

|  |  |
| --- | --- |
| 功能 | 解析JSON 格式为整型数据 |
| 函数定义 | int jsonparse\_get\_value\_as\_int(struct jsonparse\_state \*state) |
| 参数 | struct jsonparse\_state \*state：JSON 解析指针 |

5.2.2. jsonparse\_next

|  |  |
| --- | --- |
| 功能 | 解析 JSON 格式下⼀一个元素 |
| 函数定义 | int jsonparse\_next(struct jsonparse\_state \*state) |
| 参数 | struct jsonparse\_state \*state：JSON 解析指针 |
| 返回 | int：解析结果 |

5.2.3. jsonparse\_copy\_value

|  |  |
| --- | --- |
| 功能 | 复制当前解析字符串串到指定缓存 |
| 函数定义 | int jsonparse\_copy\_value(  struct jsonparse\_state \*state,   char \*str,   int size  ) |
| 参数 | struct jsonparse\_state \*state：JSON 解析指针 char \*str：缓存指针 int size：缓存⼤大⼩小 |
| 返回 | int：复制结果 |

5.2.4. jsonparse\_get\_value\_as\_int

|  |  |
| --- | --- |
| 返回 | int：解析结果 |

5.2.5. jsonparse\_get\_value\_as\_long

|  |  |
| --- | --- |
| 功能 | 解析 JSON 格式为为⻓长整型数据 |
| 函数定义 | long jsonparse\_get\_value\_as\_long(struct jsonparse\_state \*state) |
| 参数 | struct jsonparse\_state \*state：JSON 解析指针 |
| 返回 | int：解析结果 |

5.2.6. jsonparse\_get\_len

|  |  |
| --- | --- |
| 功能 | 解析 JSON 格式数据⻓长度 |
| 函数定义 | int jsonparse\_get\_value\_len(struct jsonparse\_state \*state) |
| 参数 | struct jsonparse\_state \*state：JSON 解析指针 |
| 返回 | int：解析⻓长度 |

5.2.7. jsonparse\_get\_value\_as\_type

|  |  |
| --- | --- |
| 功能 | 解析 JSON 格式数据类型 |
| 函数定义 | int jsonparse\_get\_value\_as\_type(struct jsonparse\_state \*state) |
| 参数 | struct jsonparse\_state \*state：JSON 解析指针 |
| 返回 | int：JSON 格式数据类型 |

5.2.8. jsonparse\_strcmp\_value

|  |  |
| --- | --- |
| 功能 | ⽐比较解析 JSON 数据与特定字符串串 |
| 函数定义 | int jsonparse\_strcmp\_value(struct jsonparse\_state \*state, const char \*str) |
| 参数 | struct jsonparse\_state \*state：JSON 解析指针 const char \*str：字符串串缓存 |
| 返回 | int：⽐比较结果 |

5.2.9. jsontree\_set\_up

|  |  |
| --- | --- |
| 功能 | ⽣生成 JSON 格式数据树 |
| 函数定义 | void jsontree\_setup(  struct jsontree\_context \*js\_ctx,   struct jsontree\_value \*root,   int (\* putchar)(int) ) |
| 参数 | struct jsontree\_context \*js\_ctx：JSON 格式树元素指针 struct jsontree\_value \*root：根树元素指针 int (\* putchar)(int)：输⼊入函数 |
| 返回 | ⽆无 |

5.2.10. jsontree\_reset

|  |  |
| --- | --- |
| 功能 | 设置 JSON 数 |
| 函数定义 | void jsontree\_reset(struct jsontree\_context \*js\_ctx) |
| 参数 | struct jsontree\_context \*js\_ctx：JSON 格式树指针 |
| 返回 | ⽆无 |

5.2.11. jsontree\_path\_name

|  |  |
| --- | --- |
| 功能 | 获取 JSON 树参数 |
| 函数定义 | const char \*jsontree\_path\_name(  const struct jsontree\_cotext \*js\_ctx,   int depth ) |
| 参数 | struct jsontree\_context \*js\_ctx：JSON 格式树指针 int depth：JSON 格式树深度 |
| 返回 | char\*：参数指针 |

5.2.12. jsontree\_write\_int

|  |  |
| --- | --- |
| 功能 | 整型数写⼊入 JSON 树 |
| 函数定义 | void jsontree\_write\_int(  const struct jsontree\_context \*js\_ctx,   int value ) |
| 参数 | struct jsontree\_context \*js\_ctx：JSON 树指针 int value：整型数 |
| 返回 | ⽆无 |

5.2.13. jsontree\_write\_int\_array

|  |  |
| --- | --- |
| 功能 | 整型数组写⼊入 JSON 树 |
| 函数定义 | void jsontree\_write\_int\_array(  const struct jsontree\_context \*js\_ctx,   const int \*text,   uint32 length  ) |
| 参数 | struct jsontree\_context \*js\_ctx：JSON 树指针 int \*text：数组⼊入⼝口地址 uint32 length：数组⻓长度 |
| 返回 | ⽆无 |

5.2.14. jsontree\_write\_string

|  |  |
| --- | --- |
| 功能 | 字符串串写⼊入 JSON 树 |
| 函数定义 | void jsontree\_write\_string(  const struct jsontree\_context \*js\_ctx,   const char \*text ) |
| 参数 | struct jsontree\_context \*js\_ctx：JSON 格式树指针 const char\* text：字符串串指针 |
| 返回 | ⽆无 |

5.2.15. jsontree\_print\_next

|  |  |
| --- | --- |
| 功能 | 获取 JSON 树下⼀一个元素 |
| 函数定义 | int jsontree\_print\_next(struct jsontree\_context \*js\_ctx) |
| 参数 | struct jsontree\_context \*js\_ctx：JSON 树指针 |
| 返回 | int：JSON 树深度 |

5.2.16. jsontree\_find\_next

|  |  |
| --- | --- |
| 功能 | 查找 JSON 树元素 |
| 函数定义 | struct jsontree\_value \*jsontree\_find\_next(  struct jsontree\_context \*js\_ctx,   int type ) |
| 参数 | struct jsontree\_context \*js\_ctx：JSON 树指针  int：类型 |
| 返回 | struct jsontree\_value \*：JSON 树元素指针 |

### 6. 参数结构体和宏定义

#### 6.1. 定时器器

typedef void ETSTimerFunc(void \*timer\_arg); typedef struct \_ETSTIMER\_ { struct \_ETSTIMER\_ \*timer\_next; uint32\_t timer\_expire; uint32\_t timer\_period; ETSTimerFunc \*timer\_func; void \*timer\_arg; } ETSTimer;

#### 6.2. Wi-Fi 参数

6.2.1. Station 参数

|  |  |  |
| --- | --- | --- |
| typedef struct { int8 rssi; AUTH\_MODE authmode; } wifi\_fast\_scan\_threshold\_t;  struct station\_config { uint8 ssid[32]; uint8 password[64]; uint8 bssid\_set; // Note: If bssid\_set is 1, station will just connect to the router  // with both ssid[] and bssid[] matched. Please check about this. uint8 bssid[6]; wifi\_fast\_scan\_threshold\_t threshold;  }; | | |
| ⚠**注意：**  *BSSID* 表示 *AP* 的 *MAC* 地址，⽤用于多个 *AP* 的 *SSID* 相同的情况。如果station\_config.bssid\_set==1， station\_config.bssid必须设置，否则连接失败。⼀一般情况下，station\_config.bssid\_set设置为 *0*。 |

6.2.2. SoftAP 参数

typedef enum \_auth\_mode { AUTH\_OPEN = 0,

AUTH\_WEP,

AUTH\_WPA\_PSK,

AUTH\_WPA2\_PSK,

AUTH\_WPA\_WPA2\_PSK

} AUTH\_MODE;

struct softap\_config { uint8 ssid[32]; uint8 password[64]; uint8 ssid\_len; uint8 channel; // support 1 ~ 13 uint8 authmode; // Don’t support AUTH\_WEP in soft-AP mode uint8 ssid\_hidden; // default 0 uint8 max\_connection; // default 4, max 4 uint16 beacon\_interval; // 100 ~ 60000 ms, default 100 };

|  |
| --- |
| ⚠**注意：**  如果 softap\_config.ssid\_len==0，读取 *SSID* 直⾄至结束符，否则，根据softap\_config.ssid\_len设置 *SSID* 的⻓长度。 |

6.2.3. Scan 参数

|  |
| --- |
| struct scan\_config { uint8 \*ssid; uint8 \*bssid; uint8 channel; uint8 show\_hidden; // Scan APs which are hiding their SSID or not. wifi\_scan\_type\_t scan\_type; // scan type, active or passive wifi\_scan\_time\_t scan\_time; // scan time per channel };  struct bss\_info { STAILQ\_ENTRY(bss\_info) next;  uint8 bssid[6]; uint8 ssid[32]; uint8 ssid\_len; uint8 channel; sint8 rssi; AUTH\_MODE authmode; uint8 is\_hidden; // SSID of current AP is hidden or not sint16 freq\_offset; // AP’s frequency offset sint16 freqcal\_val; uint8 \*esp\_mesh\_ie; uint8 simple\_pair; CIPHER\_TYPE pairwise\_cipher; CIPHER\_TYPE group\_cipher; uint32\_t phy\_11b:1; uint32\_t phy\_11g:1; uint32\_t phy\_11n:1; uint32\_t wps:1; uint32\_t reserved:28; }; typedef void (\* scan\_done\_cb\_t)(void \*arg, STATUS status); |

6.2.4. Wi-Fi Event 结构体

|  |
| --- |
| enum { EVENT\_STAMODE\_CONNECTED = 0,  EVENT\_STAMODE\_DISCONNECTED,  EVENT\_STAMODE\_AUTHMODE\_CHANGE,  EVENT\_STAMODE\_GOT\_IP,  EVENT\_STAMODE\_DHCP\_TIMEOUT,  EVENT\_SOFTAPMODE\_STACONNECTED,  EVENT\_SOFTAPMODE\_STADISCONNECTED,  EVENT\_SOFTAPMODE\_PROBEREQRECVED,  EVENT\_OPMODE\_CHANGED,  EVENT\_SOFTAPMODE\_DISTRIBUTE\_STA\_IP,  EVENT\_MAX };  enum {  REASON\_UNSPECIFIED = 1,  REASON\_AUTH\_EXPIRE = 2,  REASON\_AUTH\_LEAVE = 3,  REASON\_ASSOC\_EXPIRE = 4,  REASON\_ASSOC\_TOOMANY = 5,  REASON\_NOT\_AUTHED = 6,  REASON\_NOT\_ASSOCED = 7,  REASON\_ASSOC\_LEAVE = 8,  REASON\_ASSOC\_NOT\_AUTHED = 9,  REASON\_DISASSOC\_PWRCAP\_BAD = 10, /\* 11h \*/  REASON\_DISASSOC\_SUPCHAN\_BAD = 11, /\* 11h \*/  REASON\_IE\_INVALID = 13, /\* 11i \*/  REASON\_MIC\_FAILURE = 14, /\* 11i \*/  REASON\_4WAY\_HANDSHAKE\_TIMEOUT = 15, /\* 11i \*/  REASON\_GROUP\_KEY\_UPDATE\_TIMEOUT = 16, /\* 11i \*/  REASON\_IE\_IN\_4WAY\_DIFFERS = 17, /\* 11i \*/  REASON\_GROUP\_CIPHER\_INVALID = 18, /\* 11i \*/  REASON\_PAIRWISE\_CIPHER\_INVALID = 19, /\* 11i \*/  REASON\_AKMP\_INVALID = 20, /\* 11i \*/  REASON\_UNSUPP\_RSN\_IE\_VERSION = 21, /\* 11i \*/  REASON\_INVALID\_RSN\_IE\_CAP = 22, /\* 11i \*/  REASON\_802\_1X\_AUTH\_FAILED = 23, /\* 11i \*/  REASON\_CIPHER\_SUITE\_REJECTED = 24, /\* 11i \*/  REASON\_BEACON\_TIMEOUT = 200,  REASON\_NO\_AP\_FOUND = 201,  REASON\_AUTH\_FAIL = 202,  REASON\_ASSOC\_FAIL = 203,  REASON\_HANDSHAKE\_TIMEOUT = 204,  };  typedef struct {  uint8 ssid[32]; uint8 ssid\_len; |

|  |  |
| --- | --- |
| uint8 bssid[6]; uint8 channel;  } Event\_StaMode\_Connected\_t;  typedef struct {  uint8 ssid[32]; uint8 ssid\_len; uint8 bssid[6]; uint8 reason;  } Event\_StaMode\_Disconnected\_t;  typedef struct {  uint8 old\_mode; uint8 new\_mode;  } Event\_StaMode\_AuthMode\_Change\_t;  typedef struct {  struct ip\_addr ip; struct ip\_addr mask; struct ip\_addr gw;  } Event\_StaMode\_Got\_IP\_t;  typedef struct {  uint8 mac[6]; uint8 aid;  } Event\_SoftAPMode\_StaConnected\_t;  typedef struct {  uint8 mac[6]; struct ip\_addr ip; uint8 aid;  } Event\_SoftAPMode\_Distribute\_Sta\_IP\_t;  typedef struct {  uint8 mac[6]; uint8 aid;  } Event\_SoftAPMode\_StaDisconnected\_t;  typedef struct {  int rssi; uint8 mac[6];  } Event\_SoftAPMode\_ProbeReqRecved\_t;  typedef struct {  uint8 old\_opmode; uint8 new\_opmode;  } Event\_OpMode\_Change\_t;  typedef union {  Event\_StaMode\_Connected\_t | connected; |
| Event\_StaMode\_Disconnected\_t disconnected;  Event\_StaMode\_AuthMode\_Change\_t auth\_change;  Event\_StaMode\_Got\_IP\_t got\_ip;  Event\_SoftAPMode\_StaConnected\_t sta\_connected;  Event\_SoftAPMode\_Distribute\_Sta\_IP\_t distribute\_sta\_ip; Event\_SoftAPMode\_StaDisconnected\_t sta\_disconnected;  Event\_SoftAPMode\_ProbeReqRecved\_t ap\_probereqrecved;  Event\_OpMode\_Change\_t opmode\_changed;  } Event\_Info\_u;  typedef struct \_esp\_event { uint32 event; Event\_Info\_u event\_info;  } System\_Event\_t; | |

6.2.5. SmartConfig 结构体

|  |
| --- |
| typedef enum {  SC\_STATUS\_WAIT = 0, // 连接未开始，请勿在此阶段开始连接  SC\_STATUS\_FIND\_CHANNEL, // 请在此阶段开启 APP 进行配对连接  SC\_STATUS\_GETTING\_SSID\_PSWD,  SC\_STATUS\_LINK,  SC\_STATUS\_LINK\_OVER, // 获取到 IP，连接路由完成  } sc\_status; typedef enum { SC\_TYPE\_ESPTOUCH = 0,  SC\_TYPE\_AIRKISS,  SC\_TYPE\_ESPTOUCH\_AIRKISS,  } sc\_type; |

#### 6.3. JSON 相关结构体

6.3.1. JSON 结构体

|  |
| --- |
| struct jsontree\_value { uint8\_t type; };  struct jsontree\_pair { const char \*name; struct jsontree\_value \*value; };  struct jsontree\_context { struct jsontree\_value \*values[JSONTREE\_MAX\_DEPTH]; uint16\_t index[JSONTREE\_MAX\_DEPTH]; int (\* putchar)(int); uint8\_t depth; |
| uint8\_t path; int callback\_state; };  struct jsontree\_callback { uint8\_t type;  int (\* output)(struct jsontree\_context \*js\_ctx); int (\* set)(struct jsontree\_context \*js\_ctx,   struct jsonparse\_state \*parser); };  struct jsontree\_object { uint8\_t type; uint8\_t count; struct jsontree\_pair \*pairs; };  struct jsontree\_array { uint8\_t type; uint8\_t count; struct jsontree\_value \*\*values; };  struct jsonparse\_state { const char \*json; int pos; int len; int depth; int vstart; int vlen; char vtype; char error; char stack[JSONPARSE\_MAX\_DEPTH];  }; |

6.3.2. JSON 宏定义

#define JSONTREE\_OBJECT(name, ...) / static struct jsontree\_pair jsontree\_pair\_##name[] = {\_\_VA\_ARGS\_\_}; / static struct jsontree\_object name = { / JSON\_TYPE\_OBJECT, / sizeof(jsontree\_pair\_##name)/sizeof(struct jsontree\_pair), / jsontree\_pair\_##name }

#define JSONTREE\_PAIR\_ARRAY(value) (struct jsontree\_value \*)(value) #define JSONTREE\_ARRAY(name, ...) / static struct jsontree\_value\* jsontree\_value\_##name[] = {\_\_VA\_ARGS\_\_}; / static struct jsontree\_array name = { /

JSON\_TYPE\_ARRAY, /

sizeof(jsontree\_value\_##name)/sizeof(struct jsontree\_value\*), / jsontree\_value\_##name }

6.4. espconn 参数

6.4.1. 回调函数

/\*\* callback prototype to inform about events for a espconn \*/ typedef void (\* espconn\_recv\_callback)(void \*arg, char \*pdata, unsigned short len); typedef void (\* espconn\_callback)(void \*arg, char \*pdata, unsigned short len); typedef void (\* espconn\_connect\_callback)(void \*arg);

6.4.2. espconn

|  |
| --- |
| typedef void\* espconn\_handle; typedef struct \_esp\_tcp { int remote\_port; int local\_port; uint8 local\_ip[4]; uint8 remote\_ip[4]; espconn\_connect\_callback connect\_callback; espconn\_reconnect\_callback reconnect\_callback; espconn\_connect\_callback disconnect\_callback; espconn\_connect\_callback write\_finish\_fn;  } esp\_tcp;  typedef struct \_esp\_udp { int remote\_port; int local\_port; uint8 local\_ip[4]; uint8 remote\_ip[4]; } esp\_udp;  /\*\* Protocol family and type of the espconn \*/ enum espconn\_type { ESPCONN\_INVALID = 0,  /\* ESPCONN\_TCP Group \*/  ESPCONN\_TCP = 0x10,  /\* ESPCONN\_UDP Group \*/  ESPCONN\_UDP = 0x20, };  enum espconn\_option{  ESPCONN\_START = 0x00，  ESPCONN\_REUSEADDR = 0x01，  ESPCONN\_NODELAY = 0x02，  ESPCONN\_COPY = 0x04，  ESPCONN\_KEEPALIVE = 0x08， ESPCONN\_MANUALRECV = 0x10,  ESPCONN\_END  }  enum espconn\_level{  ESPCONN\_KEEPIDLE, |
| ESPCONN\_KEEPINTVL,  ESPCONN\_KEEPCNT  }  /\*\* Current state of the espconn. Non-TCP espconn are always in state ESPCONN\_NONE! \*/ enum espconn\_state { ESPCONN\_NONE,  ESPCONN\_WAIT,  ESPCONN\_LISTEN,  ESPCONN\_CONNECT,  ESPCONN\_WRITE,  ESPCONN\_READ,  ESPCONN\_CLOSE  };  /\*\* A espconn descriptor \*/ struct espconn { /\*\* type of the espconn (TCP, UDP) \*/ enum espconn\_type type; /\*\* current state of the espconn \*/ enum espconn\_state state; union { esp\_tcp \*tcp; esp\_udp \*udp; } proto; /\*\* A callback function that is informed about events for this espconn \*/ espconn\_recv\_callback recv\_callback; espconn\_sent\_callback sent\_callback; uint8 link\_cnt; void \*reverse; // reversed for customer use  }; |

#### 6.5. 中断相关宏定义

|  |
| --- |
| /\* interrupt related \*/  #define ETS\_SPI\_INUM 2 #define ETS\_GPIO\_INUM 4  #define ETS\_UART\_INUM 5 #define ETS\_UART1\_INUM 5  #define ETS\_FRC\_TIMER1\_INUM 9  /\* disable all interrupts \*/  #define ETS\_INTR\_LOCK() ets\_intr\_lock()  /\* enable all interrupts \*/  #define ETS\_INTR\_UNLOCK() ets\_intr\_unlock()  /\* register interrupt handler of frc timer1 \*/ #define ETS\_FRC\_TIMER1\_INTR\_ATTACH(func, arg) \ ets\_isr\_attach(ETS\_FRC\_TIMER1\_INUM, (func), (void \*)(arg)) |

|  |
| --- |
| /\* register interrupt handler of GPIO \*/ #define ETS\_GPIO\_INTR\_ATTACH(func, arg) \ ets\_isr\_attach(ETS\_GPIO\_INUM, (func), (void \*)(arg))  /\* register interrupt handler of UART \*/ #define ETS\_UART\_INTR\_ATTACH(func, arg) \ ets\_isr\_attach(ETS\_UART\_INUM, (func), (void \*)(arg))  /\* register interrupt handler of SPI \*/ #define ETS\_SPI\_INTR\_ATTACH(func, arg) \ ets\_isr\_attach(ETS\_SPI\_INUM, (func), (void \*)(arg))  /\* enable a interrupt \*/  #define ETS\_INTR\_ENABLE(inum) ets\_isr\_unmask((1<<inum))  /\* disable a interrupt \*/  #define ETS\_INTR\_DISABLE(inum) ets\_isr\_mask((1<<inum))  /\* enable SPI interrupt \*/  #define ETS\_SPI\_INTR\_ENABLE() ETS\_INTR\_ENABLE(ETS\_SPI\_INUM)  /\* enable UART interrupt \*/  #define ETS\_UART\_INTR\_ENABLE() ETS\_INTR\_ENABLE(ETS\_UART\_INUM)  /\* disable UART interrupt \*/  #define ETS\_UART\_INTR\_DISABLE() ETS\_INTR\_DISABLE(ETS\_UART\_INUM)  /\* enable frc1 timer interrupt \*/  #define ETS\_FRC1\_INTR\_ENABLE() ETS\_INTR\_ENABLE(ETS\_FRC\_TIMER1\_INUM)  /\* disable frc1 timer interrupt \*/  #define ETS\_FRC1\_INTR\_DISABLE() ETS\_INTR\_DISABLE(ETS\_FRC\_TIMER1\_INUM)  /\* enable GPIO interrupt \*/  #define ETS\_GPIO\_INTR\_ENABLE() ETS\_INTR\_ENABLE(ETS\_GPIO\_INUM) /\* disable GPIO interrupt \*/  #define ETS\_GPIO\_INTR\_DISABLE() ETS\_INTR\_DISABLE(ETS\_GPIO\_INUM) |

### 7. 外设驱动接⼝口

外围设备驱动可以参考 */ESP8266\_NONOS\_SDK/driver\_lib*。

#### 7.1. GPIO 接⼝口

GPIO 相关接⼝口位于 */ESP8266\_NONOS\_SDK/include/eagle\_soc.h & gpio.h*。

使⽤用示例例可参考 */ESP8266\_NONOS\_SDK/examples/IoT\_Demo/user/user\_plug.c*。

7.1.1. PIN 相关宏定义

以下宏定义控制 GPIO 管脚状态：

|  |  |  |
| --- | --- | --- |
| PIN\_PULLUP\_DIS(PIN\_NAME) | 管脚上拉屏蔽 | 示例例：  // Use MTDI pin as GPIO12.  PIN\_FUNC\_SELECT(PERIPHS\_IO\_MUX\_MTDI\_U,  FUNC\_GPIO12); |
| PIN\_PULLUP\_EN(PIN\_NAME) | 管脚上拉使能 |
| PIN\_FUNC\_SELECT(PIN\_NAME, FUNC) | 管脚功能选择 |

7.1.2. gpio\_output\_set

|  |  |
| --- | --- |
| 功能 | 设置 GPIO 属性 |
| 函数定义 | void gpio\_output\_set( uint32 set\_mask,uint32 clear\_mask,   uint32 enable\_mask,   uint32 disable\_mask  ) |
| 参数 | uint32 set\_mask：设置输出为⾼高的位，对应位为1，输出⾼高，对应位为0，不不改变状态  uint32 clear\_mask：设置输出为低的位，对应位为1，输出低，对应位为0，不不改变状态  uint32 enable\_mask：设置使能输出的位 uint32 disable\_mask：设置使能输⼊入的位 |
| 返回 | ⽆无 |
| 示例例 | gpio\_output\_set(BIT12, 0, BIT12, 0)：设置 GPIO12 输出⾼高电平； gpio\_output\_set(0, BIT12, BIT12, 0)：设置 GPIO12 输出低电平；  gpio\_output\_set(BIT12, BIT13, BIT12|BIT13, 0)：设置 GPIO12 输出⾼高电平， GPIO13 输出低电平；  gpio\_output\_set(0, 0, 0, BIT12)：设置 GPIO12 为输⼊入 |

7.1.3. GPIO 输⼊入输出相关宏

|  |  |
| --- | --- |
| GPIO\_OUTPUT\_SET(gpio\_no, bit\_value) | 设置 gpio\_no 管脚输出 bit\_value，与上⼀一节的输出⾼高低电平的示例例相同。 |
| GPIO\_DIS\_OUTPUT(gpio\_no) | 设置 gpio\_no 管脚输⼊入，与上⼀一节的设置输⼊入示例例相同。 |
| GPIO\_INPUT\_GET(gpio\_no) | 获取 gpio\_no 管脚的电平状态。 |

7.1.4. GPIO 中断

|  |  |
| --- | --- |
| ETS\_GPIO\_INTR\_ATTACH(func, arg) | 注册 GPIO 中断处理理函数 |
| ETS\_GPIO\_INTR\_DISABLE() | 关 GPIO 中断 |
| ETS\_GPIO\_INTR\_ENABLE() | 开 GPIO 中断 |

7.1.5. gpio\_pin\_intr\_state\_set

|  |  |
| --- | --- |
| 功能 | 设置 GPIO 中断触发状态 |
| 函数定义 | void gpio\_pin\_intr\_state\_set(  uint32 i,  GPIO\_INT\_TYPE intr\_state ) |
| 参数 | uint32 i：GPIO pin ID，例例如设置 GPIO14，则为 GPIO\_ID\_PIN(14); GPIO\_INT\_TYPE intr\_state ：中断触发状态： typedef enum {  GPIO\_PIN\_INTR\_DISABLE = 0,  GPIO\_PIN\_INTR\_POSEDGE = 1,  GPIO\_PIN\_INTR\_NEGEDGE = 2,  GPIO\_PIN\_INTR\_ANYEDGE = 3,  GPIO\_PIN\_INTR\_LOLEVEL = 4,  GPIO\_PIN\_INTR\_HILEVEL = 5  } GPIO\_INT\_TYPE; |
| 返回 | ⽆无 |

7.1.6. GPIO 中断处理理函数在 GPIO 中断处理理函数内，需要做如下操作来清除响应位的中断状态：

|  |
| --- |
| uint32 gpio\_status; gpio\_status = GPIO\_REG\_READ(GPIO\_STATUS\_ADDRESS); //clear interrupt status  GPIO\_REG\_WRITE(GPIO\_STATUS\_W1TC\_ADDRESS, gpio\_status); |

#### 7.2. UART 接⼝口

默认情况下，UART0 作为系统的打印信息输出接⼝口，当配置为双 UART 时，UART0 作为数据收发接⼝口，UART1 作为打印信息输出接⼝口。使⽤用时，请确保硬件连接正确。

关于 UART 的详细介绍，请参考 [ESP8266 技术参考](http://www.espressif.com/sites/default/files/documentation/esp8266-technical_reference_cn.pdf)。

7.2.1. uart\_init

|  |  |
| --- | --- |
| 功能 | 双 UART 模式，两个 UART 波特率初始化 |
| 函数定义 | void uart\_init(  UartBautRate uart0\_br,  UartBautRate uart1\_br ) |
| 参数 | UartBautRate uart0\_br：UART0 波特率  UartBautRate uart1\_br：UART1 波特率 |
| 波特率 | typedef enum {  BIT\_RATE\_9600 = 9600,  BIT\_RATE\_19200 = 19200,  BIT\_RATE\_38400 = 38400,  BIT\_RATE\_57600 = 57600,  BIT\_RATE\_74880 = 74880,  BIT\_RATE\_115200 = 115200,  BIT\_RATE\_230400 = 230400,  BIT\_RATE\_460800 = 460800,  BIT\_RATE\_921600 = 921600  } UartBautRate; |
| 返回 | ⽆无 |

7.2.2. uart0\_tx\_buffer

|  |  |
| --- | --- |
| 功能 | 通过 UART0 输出⽤用户数据 |
| 函数定义 | void uart0\_tx\_buffer(uint8 \*buf, uint16 len) |
| 参数 | uint8 \*buf：数据缓存 uint16 len：数据⻓长度 |
| 返回 | ⽆无 |

7.2.3. uart0\_rx\_intr\_handler

|  |  |
| --- | --- |
| 功能 | UART0 中断处理理函数，⽤用户可在该函数内添加对接收到数据包的处理理。 |
| 函数定义 | void uart0\_rx\_intr\_handler(void \*para) |
| 参数 | void \*para：指向数据结构 RcvMsgBuff 的指针 |
| 返回 | ⽆无 |

7.2.4. uart\_div\_modify

|  |  |
| --- | --- |
| 功能 | 设置 UART 波特率。 |
| 函数定义 | void uart\_div\_modify(uint8 uart\_no, uint32 DivLatchValue) |
| 参数 | uint8 uart\_no：UART 号，UART0 或者 UART1 uint32 DivLatchValue: 分频参数 |
| 返回 | ⽆无 |
| 示例例 | void ICACHE\_FLASH\_ATTR UART\_SetBaudrate(uint8 uart\_no, uint32 baud\_rate)  {  uart\_div\_modify(uart\_no, UART\_CLK\_FREQ /baud\_rate);  } |

#### 7.3. I2C Master 接⼝口

ESP8266 不不能作为 I2C 从设备，但可以作为 I2C 主设备，对其他 I2C 从设备（例例如⼤大多数数字传感器器）进⾏行行控制与读写。

每个 GPIO 管脚内部都可以配置为开漏漏模式 (open-drain)，从⽽而可以灵活的将 GPIO ⼝口⽤用作 I2C data 或 clock 功能。同时，芯⽚片内部提供上拉电阻，以节省外部的上拉电阻。关于 I2C 的详细介绍，请参考 [ESP8266 技术参考](http://www.espressif.com/sites/default/files/documentation/esp8266-technical_reference_cn.pdf)。

7.3.1. i2c\_master\_gpio\_init

|  |  |
| --- | --- |
| 功能 | 设置 GPIO 为 I2C master 模式 |
| 函数定义 | void i2c\_master\_gpio\_init (void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

7.3.2. i2c\_master\_init

|  |  |
| --- | --- |
| 功能 | 初始化 I2C |
| 函数定义 | void i2c\_master\_init(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

7.3.3. i2c\_master\_start

|  |  |
| --- | --- |
| 功能 | 设置 I2C 进⼊入发送状态 |
| 函数定义 | void i2c\_master\_start(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

7.3.4. i2c\_master\_stop

|  |  |
| --- | --- |
| 功能 | 设置 I2C 停⽌止发送 |
| 函数定义 | void i2c\_master\_stop(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

7.3.5. i2c\_master\_send\_ack

|  |  |
| --- | --- |
| 功能 | 发送 I2C ACK |
| 函数定义 | void i2c\_master\_send\_ack (void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

7.3.6. i2c\_master\_send\_nack

|  |  |
| --- | --- |
| 功能 | 发送 I2C NACK |
| 函数定义 | void i2c\_master\_send\_nack (void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

7.3.7. i2c\_master\_checkAck

|  |  |
| --- | --- |
| 功能 | 检查 I2C slave 的 ACK |
| 函数定义 | bool i2c\_master\_checkAck (void) |
| 参数 | ⽆无 |
| 返回 | true：获取 I2C slave ACK  false：获取 I2C slave NACK |

7.3.8. i2c\_master\_readByte

|  |  |
| --- | --- |
| 功能 | 从 I2C slave 读取⼀一个字节 |
| 函数定义 | uint8 i2c\_master\_readByte (void) |
| 参数 | ⽆无 |
| 返回 | uint8：读取到的值 |

7.3.9. i2c\_master\_writeByte

|  |  |
| --- | --- |
| 功能 | 向 I2C slave 写⼀一个字节 |
| 函数定义 | void i2c\_master\_writeByte (uint8 wrdata) |
| 参数 | uint8 wrdata：数据 |
| 返回 | ⽆无 |

#### 7.4. PWM 接⼝口

本⽂文档仅简单介绍 *pwm.h* 中的 PWM 相关接⼝口，详细的 PWM 介绍⽂文档请参考 [ESP8266 技术参考](http://www.espressif.com/sites/default/files/documentation/esp8266-technical_reference_cn.pdf)。

PWM 驱动接⼝口函数不不能跟 *hw\_timer.c* 的接⼝口同时使⽤用，因为它们共⽤用了了同⼀一个硬件定时器器。PWM 不不⽀支持进⼊入 Deep sleep 模式，也请勿调⽤用

wifi\_set\_sleep\_type(LIGT\_SLEEP); 将⾃自动睡眠模式设置为 Light-sleep。因为 Lightsleep 在睡眠期间会停 CPU，停 CPU 期间不不能响应 NMI 中断。

7.4.1. pwm\_init

|  |  |
| --- | --- |
| 功能 | 初始化 PWM，包括 GPIO 选择，周期和占空⽐比。⽬目前仅⽀支持调⽤用⼀一次。 |
| 函数定义 | void pwm\_init(  uint32 period, uint8 \*duty,  uint32 pwm\_channel\_num, uint32 (\*pin\_info\_list)[3]) |
| 参数 | uint32 period：PWM 周期 uint8 \*duty：各路路 PWM 的占空⽐比 uint32 pwm\_channel\_num：PWM 通道数  uint32 (\*pin\_info\_list)[3]：PWM 各通道的 GPIO 硬件参数。本参数是⼀一个 n \* 3 的数组指针，数组中定义了了 GPIO 的寄存器器，对应 PIN 脚的 IO 复⽤用值和 GPIO 对应的序号 |
| 返回 | ⽆无 |
| 示例例 | 初始化⼀一个三通道的 PWM：  uint32 io\_info[][3] =  {{PWM\_0\_OUT\_IO\_MUX,PWM\_0\_OUT\_IO\_FUNC,PWM\_0\_OUT\_IO\_NUM},  {PWM\_1\_OUT\_IO\_MUX,PWM\_1\_OUT\_IO\_FUNC,PWM\_1\_OUT\_IO\_NUM},  {PWM\_2\_OUT\_IO\_MUX,PWM\_2\_OUT\_IO\_FUNC,PWM\_2\_OUT\_IO\_NUM}}; pwm\_init(light\_param.pwm\_period, light\_param.pwm\_duty, 3, io\_info); |

7.4.2. pwm\_start

|  |  |
| --- | --- |
| 功能 | PWM 开始。每次更更新 PWM 设置后，都需要重新调⽤用本接⼝口进⾏行行计算。 |
| 函数定义 | void pwm\_start (void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

|  |  |
| --- | --- |
| 功能 | 设置 PWM 周期，单位：μs。例例如，1KHz PWM，参数为 1000 μs。 |
| 注意 | 设置完成后，需要调⽤用 pwm\_start ⽣生效。 |
| 函数定义 | void pwm\_set\_period(uint32 period) |
| 参数 | uint32 period：PWM 周期，单位：μs |

7.4.3. pwm\_set\_duty

|  |  |
| --- | --- |
| 功能 | 设置 PWM 某个通道信号的占空⽐比。设置各路路 PWM 信号⾼高电平所占的时间，duty 的范围随  PWM 周期改变， ⼤大值为：Period \* 1000 /45。例例如，1KHz PWM，duty 范围是：0 ~ 22222 |
| 注意 | 设置完成后，需要调⽤用 pwm\_start ⽣生效。 |
| 函数定义 | void pwm\_set\_duty(uint32 duty, uint8 channel) |
| 参数 | uint32 duty：设置⾼高电平时间参数，占空⽐比的值为 (duty\*45)/ (period\*1000)  uint8 channel：当前要设置的 PWM 通道，取值范围依据实际使⽤用了了⼏几路路 PWM，在 *IOT\_Demo* 中取值在 #define PWM\_CHANNEL 定义的范围内。 |
| 返回 | ⽆无 |

7.4.4. pwm\_get\_duty

|  |  |
| --- | --- |
| 功能 | 获取某路路 PWM 信号的 duty 参数，占空⽐比的值为 (duty\*45)/ (period\*1000) |
| 函数定义 | uint8 pwm\_get\_duty(uint8 channel) |
| 参数 | uint8 channel：当前要查询的 PWM 通道，取值范围依据实际使⽤用了了⼏几路路 PWM，在 *IOT\_Demo* 中取值在 #define PWM\_CHANNEL 定义的范围内。 |
| 返回 | 对应某路路 PWM 信号的 duty 参数 |

7.4.5. pwm\_set\_period

|  |  |
| --- | --- |
| 返回 | ⽆无 |

7.4.6. pwm\_get\_period

|  |  |
| --- | --- |
| 功能 | 查询 PWM 周期 |
| 函数定义 | uint32 pwm\_get\_period(void) |
| 参数 | ⽆无 |
| 返回 | PWM 周期，单位：μs |

7.4.7. get\_pwm\_version

|  |  |
| --- | --- |
| 功能 | 查询 PWM 版本信息 |
| 函数定义 | uint32 get\_pwm\_version(void) |
| 参数 | ⽆无 |
| 返回 | PWM 版本信息 |

#### 7.5. SDIO 接⼝口

ESP8266 仅⽀支持作为 SDIO slave。

7.5.1. sdio\_slave\_init

|  |  |
| --- | --- |
| 功能 | 初始化 SDIO |
| 函数定义 | void sdio\_slave\_init(void) |
| 参数 | ⽆无 |
| 返回 | ⽆无 |

7.5.2. sdio\_load\_data

|  |  |
| --- | --- |
| 功能 | 加载数据到 SDIO buffer 中，并通知 SDIO host 读取。 |
| 函数定义 | int32 sdio\_load\_data(const uint8\* data, uint32 len) |
| 参数 | const uint8\* data：待传输的数据 uint32 len：数据⻓长度 |
| 返回 | 实际成功加载到 SDIO buffer 中的数据⻓长度。  ⽬目前不不⽀支持加载部分数据，如果数据超过 SDIO buffer 可加载容量量，将返回 0，数据加载失败。 |

7.5.3. sdio\_register\_recv\_cb

|  |  |
| --- | --- |
| 功能 | 注册 SDIO 收到 host 数据的回调函数。 |
| 回调函数定义 | typedef void(\*sdio\_recv\_data\_callback)(uint8\* data, uint32 len) 注册的回调函数不不能放在 cache 中，即回调函数前不不能添加 ICACHE\_FLASH\_ATTR 宏定义。 |
| 函数定义 | bool sdio\_register\_recv\_cb(sdio\_recv\_data\_callback cb) |
| 参数 | sdio\_recv\_data\_callback cb：回调函数 |
| 返回 | true：注册成功 false：注册失败 |

# A. 附录

## A.1. ESPCONN 编程

可参考乐鑫提供的示例例代码：

<https://github.com/espressif/esp8266-nonos-sample-code><https://github.com/espressif/esp8266-rtos-sample-code>

A.1.1. TCP Client 模式

|  |
| --- |
| ⚠**注意：**   * *ESP8266* ⼯工作在 *Station* 模式下，需确认 *ESP8266* 已经连接 *AP*（路路由）分配到 *IP* 地址，启⽤用 *client* 连接。 * *ESP8266* ⼯工作在 *SoftAP* 模式下，需确认连接 *ESP8266* 的设备已被分配到 *IP* 地址，启⽤用 *client* 连接。 |

步骤如下：

1. 依据⼯工作协议初始化 espconn 参数；
2. 注册连接成功的回调函数和连接失败重连的回调函数；
   * 调⽤用 espconn\_regist\_connectcb 和 espconn\_regist\_reconcb
3. 调⽤用 espconn\_connect 建⽴立与 TCP Server 的连接；
4. TCP连接建⽴立成功后，在连接成功的回调函数 espconn\_connect\_callback 中，注册接收数据的回调函数，发送数据成功的回调函数和断开连接的回调函数。
   * 调⽤用 espconn\_regist\_recvcb、espconn\_regist\_sentcb 和 espconn\_regist\_disconcb
5. 在接收数据的回调函数，或者发送数据成功的回调函数中，执⾏行行断开连接操作时，建议适当延时⼀一定时间，确保底层函数执⾏行行结束。

A.1.2. TCP Server 模式

|  |
| --- |
| ⚠**注意：**   * *ESP8266* ⼯工作在 *Station* 模式下，需确认 *ESP8266* 已经分配到 *IP* 地址，再启⽤用 *server* 侦听。 * *ESP8266* ⼯工作在 *SoftAP* 模式下，可以直接启⽤用 *server* 侦听。 |

步骤如下：

1. 依据⼯工作协议初始化 espconn参数；
2. 注册连接成功的回调函数和连接失败重连的回调函数；
   * 调⽤用 espconn\_regist\_connectcb 和 espconn\_regist\_reconcb
3. 调⽤用 espconn\_accept 侦听 TCP 连接；
4. TCP 连接建⽴立成功后，在连接成功的回调函数 espconn\_connect\_callback中，注册接收数据的回调函数，发送数据成功的回调函数和断开连接的回调函数。
   * 调⽤用 espconn\_regist\_recvcb、espconn\_regist\_sentcb 和 espconn\_regist\_disconcb

A.1.3. espconn Callback

|  |  |  |  |
| --- | --- | --- | --- |
| **注册函数** | | **回调函数** | **说明** |
| espconn\_regist\_connectcb | | espconn\_connect\_callback | TCP 连接建⽴立成功 |
| espconn\_regist\_reconcb | | espconn\_reconnect\_callback | TCP 连接发⽣生异常⽽而断开 |
| espconn\_regist\_sentcb | | espconn\_sent\_callback | TCP 或 UDP 数据发送完成 |
| espconn\_regist\_recvcb | | espconn\_recv\_callback | TCP 或 UDP 数据接收 |
| espconn\_regist\_write\_finish | | espconn\_write\_finish\_callback | 数据成功写⼊入 TCP 数据缓存 |
| espconn\_regist\_disconcb | | espconn\_disconnect\_callback | TCP 连接正常断开 |
| ⚠**注意：**   * 回调函数中传⼊入的指针arg，对应⽹网络连接的结构体espconn指针。该指针为 *SDK* 内部维护的指针，不不同回调传⼊入的指针地址可能不不⼀一样，请勿依此判断⽹网络连接。可根据espconn结构体中的remote\_ip、remote\_port判断多连接中的不不同⽹网络传输。 * 如果espconn\_connect（或者espconn\_secure\_connect）失败，返回⾮非零值，连接未建⽴立，不不会进   ⼊入任何 espconn *callback*。   * 请勿在espconn 任何回调中调⽤用espconn\_disconnect（或者espconn\_secure\_disconnect）断开连接。如有需要，可以在espconn回调中使⽤用触发任务的⽅方式（system\_os\_task和system\_os\_post）调⽤用espconn\_disconnect（或者espconn\_secure\_disconnect）断开连接。 | | | |

## A.2. RTC API 使⽤用示例例

以下测试示例例，可以验证 RTC 时间和系统时间，在 system\_restart时的变化，以及读写 RTC memory。

|  |
| --- |
| #include "ets\_sys.h"  #include "osapi.h" #include "user\_interface.h"  os\_timer\_t rtc\_test\_t;  #define RTC\_MAGIC 0x55aaaa55 |

|  |
| --- |
| typedef struct {  uint64 time\_acc; uint32 magic ; uint32 time\_base;  }RTC\_TIMER\_DEMO;  void rtc\_count() { RTC\_TIMER\_DEMO rtc\_time; static uint8 cnt = 0; system\_rtc\_mem\_read(64, &rtc\_time, sizeof(rtc\_time)); if(rtc\_time.magic!=RTC\_MAGIC){ os\_printf("rtc time init...\r\n"); rtc\_time.magic = RTC\_MAGIC; rtc\_time.time\_acc= 0; rtc\_time.time\_base = system\_get\_rtc\_time();  os\_printf("time base : %d \r\n",rtc\_time.time\_base);  } os\_printf("==================\r\n"); os\_printf("RTC time test : \r\n"); uint32 rtc\_t1,rtc\_t2; uint32 st1,st2; uint32 cal1, cal2; rtc\_t1 = system\_get\_rtc\_time(); st1 = system\_get\_time(); cal1 = system\_rtc\_clock\_cali\_proc(); os\_delay\_us(300); st2 = system\_get\_time(); rtc\_t2 = system\_get\_rtc\_time(); cal2 = system\_rtc\_clock\_cali\_proc(); os\_printf(" rtc\_t2-t1 : %d \r\n",rtc\_t2-rtc\_t1); os\_printf(" st2-t2 : %d \r\n",st2-st1); os\_printf("cal 1 : %d.%d \r\n", ((cal1\*1000)>>12)/1000, ((cal1\*1000)>>12)%1000 ); os\_printf("cal 2 : %d.%d \r\n",((cal2\*1000)>>12)/1000,((cal2\*1000)>>12)%1000 ); os\_printf("==================\r\n\r\n"); rtc\_time.time\_acc += ( ((uint64)(rtc\_t2 - rtc\_time.time\_base)) \* ( (uint64) ((cal2\*1000)>>12)) ) ; os\_printf("rtc time acc : %lld \r\n",rtc\_time.time\_acc); os\_printf("power on time : %lld us\r\n", rtc\_time.time\_acc/1000); os\_printf("power on time : %lld.%02lld S\r\n", (rtc\_time.time\_acc/10000000)/100, (rtc\_time.time\_acc/10000000)%100); rtc\_time.time\_base = rtc\_t2; |
| system\_rtc\_mem\_write(64, &rtc\_time, sizeof(rtc\_time)); os\_printf("------------------------\r\n"); if(5 == (cnt++)){ os\_printf("system restart\r\n"); system\_restart();  }else{ os\_printf("continue ...\r\n");  } }  void user\_init(void) { rtc\_count(); os\_printf("SDK version:%s\n", system\_get\_sdk\_version()); os\_timer\_disarm(&rtc\_test\_t); os\_timer\_setfn(&rtc\_test\_t,rtc\_count,NULL); os\_timer\_arm(&rtc\_test\_t,10000,1);  } |

A.3. Sniffer 说明

关于 sniffer 的详细说明，请参考 [ESP8266 技术参考。](http://www.espressif.com/sites/default/files/documentation/esp8266-technical_reference_cn.pdf)

## A.4. ESP8266 SoftAP 和 Station 信道定义

虽然 ESP8266 ⽀支持 SoftAP+Station 共存模式，但是 ESP8266 实际只有⼀一个硬件信道。因此在 SoftAP+Station 模式时，ESP8266 SoftAP 会动态调整信道值与 ESP8266 Station ⼀一致。

这个限制会导致 ESP8266 SoftAP+Station 模式时⼀一些⾏行行为上的不不便便，⽤用户请注意。例例如：情况⼀一

1. 如果 ESP8266 Station 连接到⼀一个路路由（假设路路由信道号为 6）
2. 通过接⼝口 wifi\_softap\_set\_config 设置 ESP8266 SoftAP
3. 若设置值合法有效，该 API 将返回 true，但信道号仍然会⾃自动调节成与 ESP8266 Station 接⼝口⼀一致，在这个例例⼦子⾥里里也就是信道号为 6。因为 ESP8266 在硬件上只有⼀一个信道，由 ESP8266 Station 与 SoftAP 接⼝口共⽤用。

情况⼆二

1. 调⽤用接⼝口 wifi\_softap\_set\_config 设置 ESP8266 SoftAP（例例如信道号为 5）
2. 其他 Station 连接到 ESP8266 SoftAP
3. 将 ESP8266 Station 连接到路路由（假设路路由信道号为 6）
4. ESP8266 SoftAP 将⾃自动调整信道号与 ESP8266 Station ⼀一致（信道 6） 5. 由于信道改变，之前连接到 ESP8266 SoftAP 的 Station 的 Wi-Fi 连接断开。

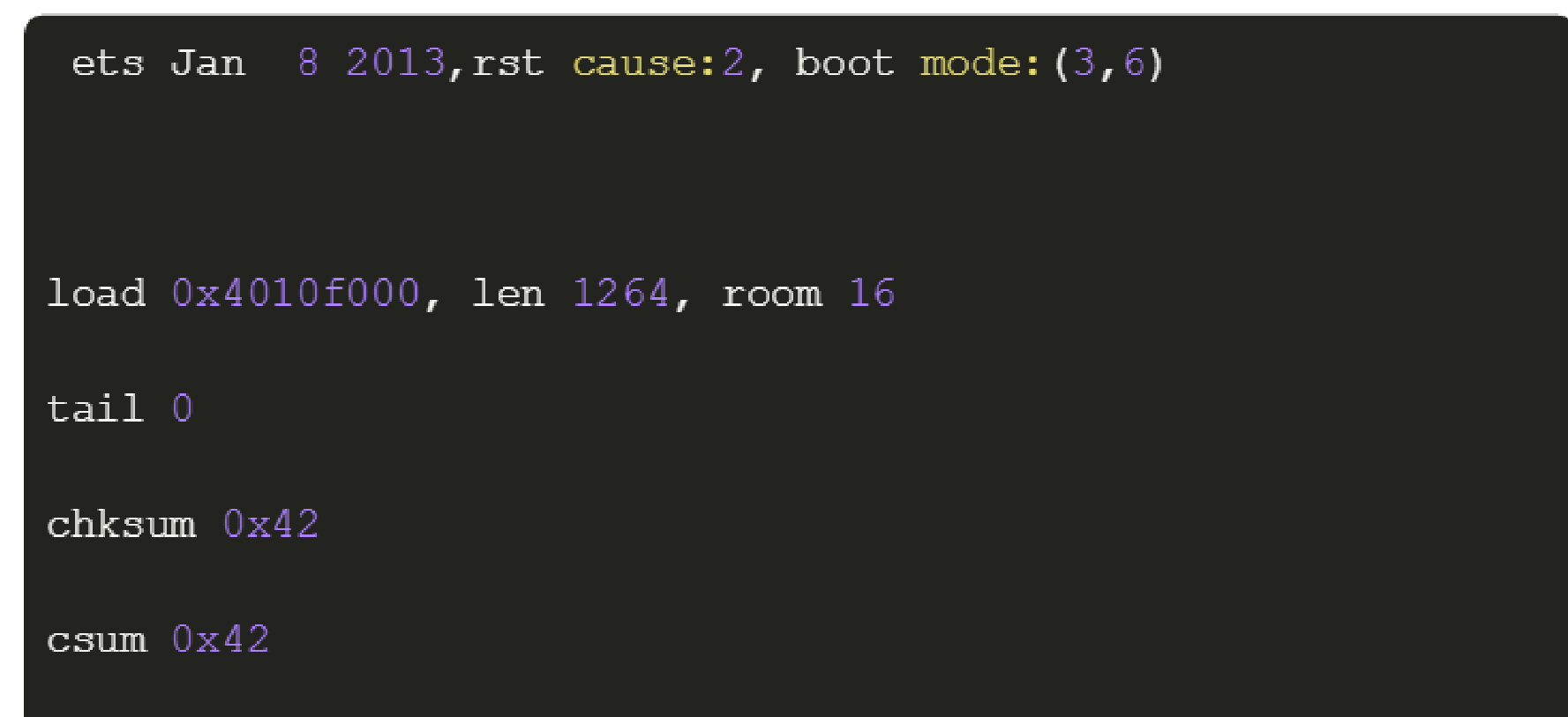
情况三

1. 其他 Station 与 ESP8266 SoftAP 建⽴立连接
2. 如果 ESP8266 Station ⼀一直尝试扫描或连接某路路由，可能导致 ESP8266 SoftAP 端的连接断开。
3. 因为 ESP8266 Station 会遍历各个信道查找⽬目标路路由，意味着 ESP8266 其实在不不停切换信道，ESP8266 SoftAP 的信道也因此在不不停更更改。这可能导致 ESP8266 SoftAP 端的原有连接断开。
4. 这种情况，⽤用户可以通过设置定时器器，超时后调⽤用 wifi\_station\_disconnect 停⽌止 ESP8266 Station 不不断连接路路由的尝试；或者在初始配置时，调⽤用

wifi\_station\_set\_reconnect\_policy 和 wifi\_station\_set\_auto\_connect 禁⽌止 ESP8266 Station 尝试重连路路由。

## A.5. ESP8266 启动信息说明

ESP8266 启动时，将从 UART0 以波特率 74880 打印如下启动信息：

!

其中可供⽤用户参考的启动信息说明如下：

|  |  |  |
| --- | --- | --- |
| **启动信息** |  | **说明** |
| rst cause | 1：上电 |  |
| 2：外部复位 |  |
| **启动信息** | **说明** | |
|  | 4：硬件看⻔门狗复位 | |
| boot mode  第⼀一个参数 | 1：ESP8266 处于 UART-down 模式，可通过 UART 下载固件 | |
| 3：ESP8266 处于 Flash-boot 模式，从 Flash 启动运⾏行行 | |
| chksum | chksum 与 csum 值相等，表示启动过程中 Flash 读取正确 | |

## A.6. ESP8266 信令测试使⽤用说明

ESP8266\_NonOS\_SDK\_V3.0 增加了了信令测试功能，该功能默认关闭。在

user\_interface.h 定义了了函数 wifi\_enable\_signaling\_measurement() 和

wifi\_disable\_signaling\_measurement() 可⽤用于开启/关闭信令测试功能。建议使⽤用罗德施瓦茨公司的 CMW500 信令测试仪进⾏行行测试。具体使⽤用步骤如下：

1. 在系统启动后，调⽤用 wifi\_set\_opmode 使能 sta 模式，调⽤用 wifi\_enable\_signaling\_measurement 开启信令测试模式；
2. 如果需要测试 11n 或者 11b，可以调⽤用 wifi\_set\_phy\_mode 将模式设置为 11n 或者

11b；如果⽆无需测试，则⽆无需调⽤用，ESP8266 默认在 11g 模式；

1. 调⽤用 wifi\_station\_connect 连接测试仪。

免责申明和版权公告

本⽂文中的信息，包括供参考的 URL 地址，如有变更更，恕不不另⾏行行通知。

⽂文档“按现状”提供，不不负任何担保责任，包括对适销性、适⽤用于特定⽤用途或⾮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。本⽂文档不不负任何责任，包括使⽤用本⽂文档内信息产⽣生的侵犯任何专利利权⾏行行为的责任。本⽂文档在此未以禁⽌止反⾔言或其他⽅方式授予任何知识产权使⽤用许可，不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。蓝⽛牙标志是 Bluetooth SIG 的注册商标。

|  |
| --- |
| 乐鑫 IOT 团队  [*www.espressif.com*](http://www.espressif.com/) |

⽂文中提到的所有商标名称、商标和注册商标均属其各⾃自所有者的财产，特此声明。

版权归© 2020 乐鑫所有。保留留所有权利利。