Wifi：

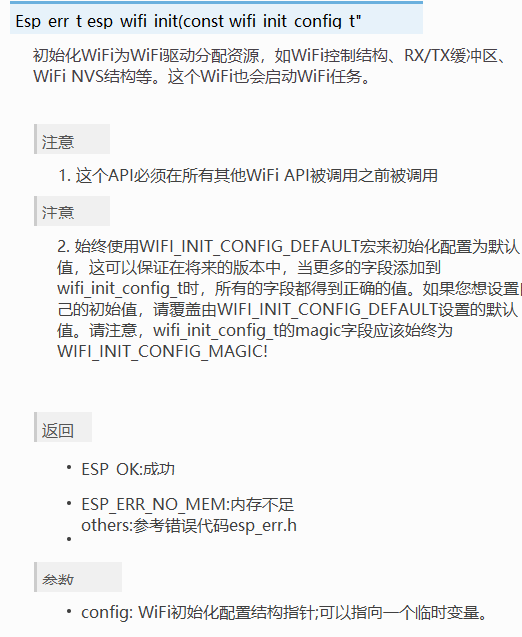
[WIFI的AP（Access Point）和STA（Station）指什么\_M2嵌入式-CSDN博客\_accesspoint](https://blog.csdn.net/weixin_42445727/article/details/94737103)

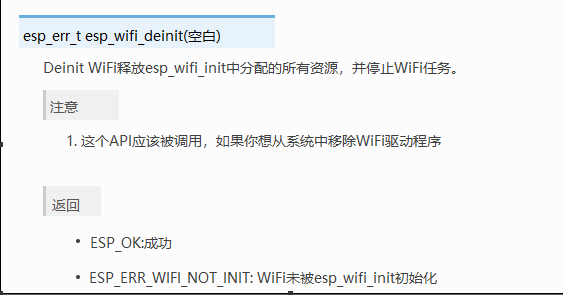
[无线AP、WiFi、WLAN是什么关系？\_清之我心的博客-CSDN博客\_ap wifi](https://blog.csdn.net/niutianzhuang/article/details/79727977)

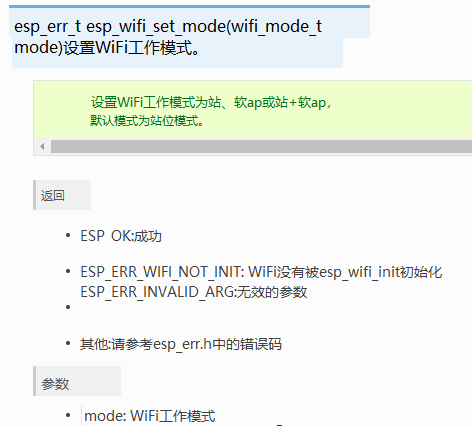
[ESP32学习笔记（5）——WiFi接口使用(STA和AP模式)\_Leung的博客-CSDN博客\_esp32接口](https://blog.csdn.net/qq_36347513/article/details/115863228)

[ESP32作为接入点AP - noticeable - 博客园 (cnblogs.com)](https://www.cnblogs.com/noticeable/p/7754633.html)

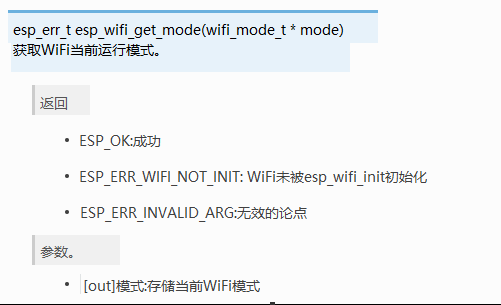
1. **SoftAP**



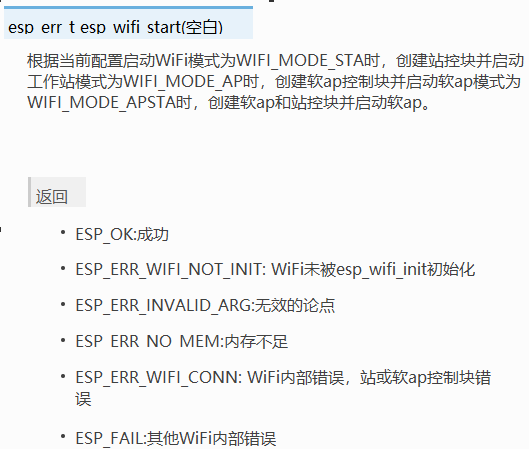








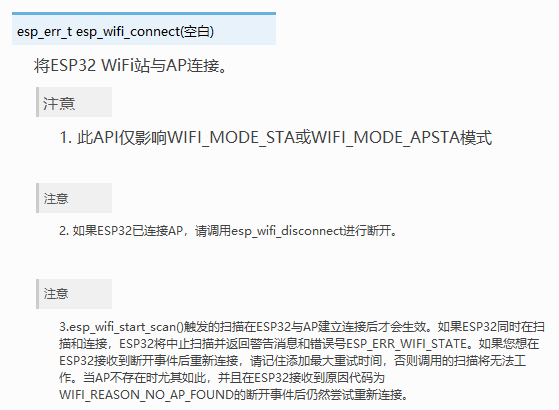


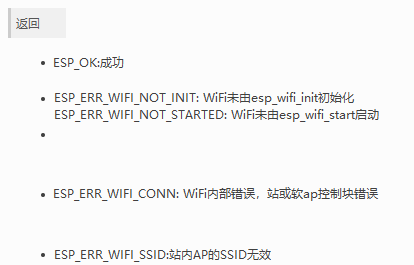




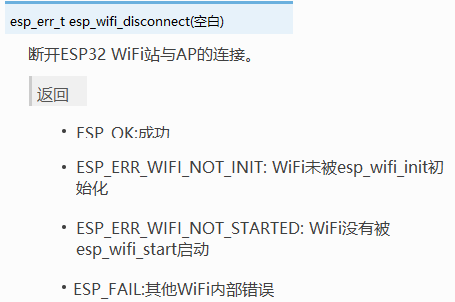








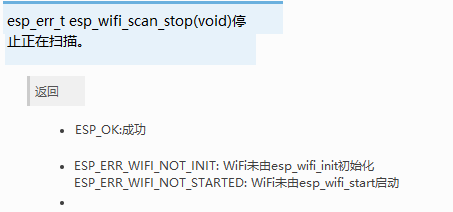




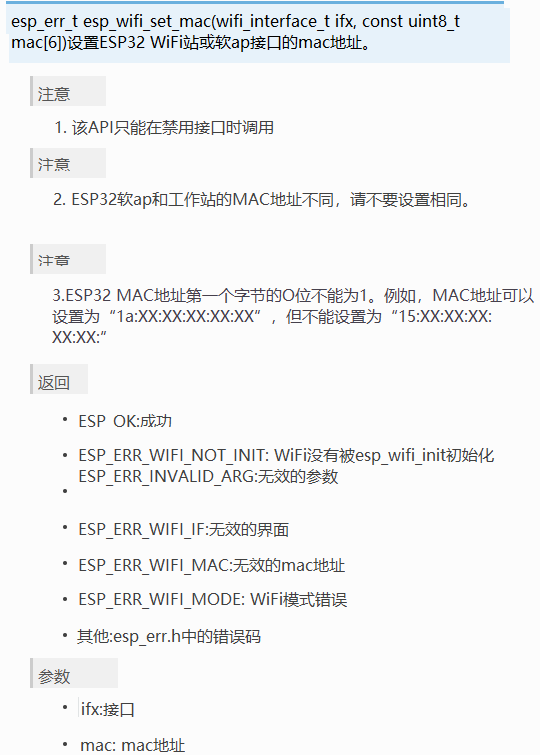




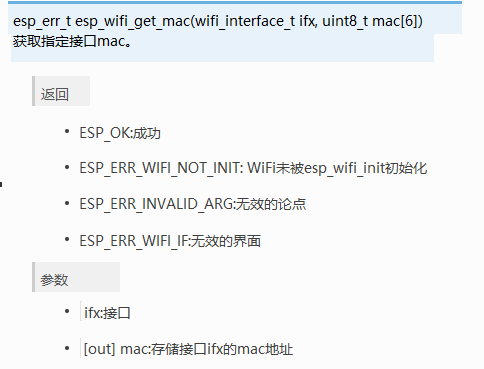




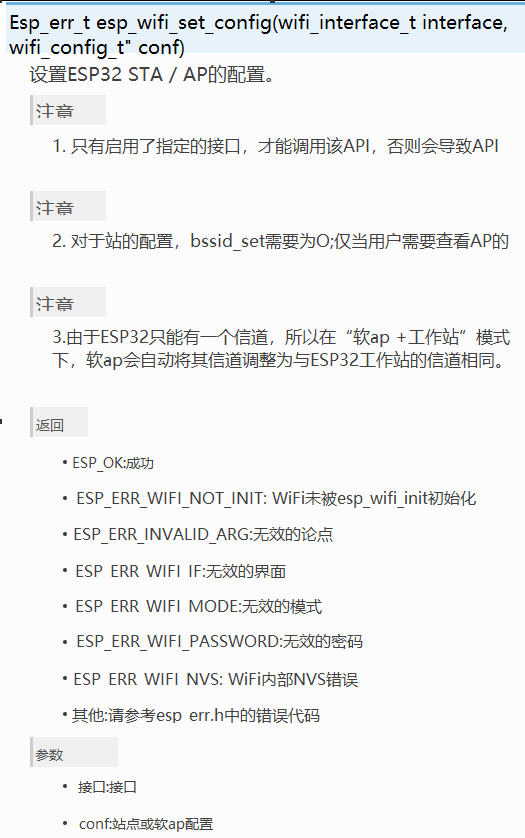
















示例创建过程：

添加相应头文件：

#include <string.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "esp\_system.h"

#include "esp\_wifi.h"

#include "esp\_event.h"

#include "esp\_log.h"

#include "nvs\_flash.h"

#include "lwip/err.h"

#include "lwip/sys.h"

1. 创建并初始化wifi相关参数

    //创建LwIP核心任务并初始化相关工作

**ESP\_ERROR\_CHECK**(**esp\_netif\_init**());

    // 创建系统事件任务并初始化应用程序事件的回调函数

**ESP\_ERROR\_CHECK**(**esp\_event\_loop\_create\_default**());

    // 创建具有TCP / IP堆栈的默认网络接口实例绑定AP。

**esp\_netif\_create\_default\_wifi\_ap**();

**ESP\_ERROR\_CHECK**(**esp\_event\_handler\_instance\_register**(WIFI\_EVENT,

                                                        ESP\_EVENT\_ANY\_ID,

                                                        &wifi\_event\_handler,

                                                        NULL,

                                                        NULL));

//初始化Wi-Fi

//定义一个名为cfg的wifi\_init\_config\_t结构体，wifi\_init\_config\_t的参数可由menuconfig配置

wifi\_init\_config\_t cfg = **WIFI\_INIT\_CONFIG\_DEFAULT**();

//初始名为cfg的结构体

**ESP\_ERROR\_CHECK**(**esp\_wifi\_init**(&cfg));

1. 对wifi进行配置并启动wifi

    //Wi-Fi配置

    wifi\_config\_t wifi\_config = {

        .ap = {

            .ssid = EXAMPLE\_ESP\_WIFI\_SSID,

            .ssid\_len = **strlen**(EXAMPLE\_ESP\_WIFI\_SSID),

            .channel = EXAMPLE\_ESP\_WIFI\_CHANNEL,

            .password = EXAMPLE\_ESP\_WIFI\_PASS,

            .max\_connection = EXAMPLE\_MAX\_STA\_CONN,

            .authmode = WIFI\_AUTH\_WPA\_WPA2\_PSK

        },

};

// 注：EXAMPLE\_ESP\_WIFI\_SSID：Wifi名字，由宏指定

// EXAMPLE\_ESP\_WIFI\_PASS：Wifi密码，由宏指定

// #define EXAMPLE\_ESP\_WIFI\_SSID      "Test\_Wifi"

// #define EXAMPLE\_ESP\_WIFI\_PASS      "746169690"

// CONFIG\_ESP\_WIFI\_CHANNEL：Wifi通道，由宏指定

// CONFIG\_ESP\_MAX\_STA\_CONN：Wifi最大连接数，由宏指定

// #define CONFIG\_ESP\_WIFI\_CHANNEL 1

// #define CONFIG\_ESP\_MAX\_STA\_CONN 4

    if (**strlen**(EXAMPLE\_ESP\_WIFI\_PASS) == 0) {

        wifi\_config.ap.authmode = WIFI\_AUTH\_OPEN;

}

//设置模式为接入点（SoftAP）模式

**ESP\_ERROR\_CHECK**(**esp\_wifi\_set\_mode**(WIFI\_MODE\_AP));

//初始化接入点（SoftAP）模式的配置

**ESP\_ERROR\_CHECK**(**esp\_wifi\_set\_config**(WIFI\_IF\_AP, &wifi\_config));

**ESP\_ERROR\_CHECK**(**esp\_wifi\_start**());

//打印Wifi启动日志

**ESP\_LOGI**(TAG, "wifi\_init\_softap finished. SSID:%s password:%s channel:%d",

             EXAMPLE\_ESP\_WIFI\_SSID, EXAMPLE\_ESP\_WIFI\_PASS, EXAMPLE\_ESP\_WIFI\_CHANNEL);

注:（一）和（二）均放在函数 void wifi\_init\_softap(void) 中，提升代码可读性

1. 创建Wifi连接或断开的时间处理函数

//当有终端连接（WIFI\_EVENT\_AP\_STACONNECTED）或断开（WIFI\_EVENT\_AP\_STADISCONNECTED）时，触发以下函数

static void **wifi\_event\_handler**(void\* arg, esp\_event\_base\_t event\_base,

                                    int32\_t event\_id, void\* event\_data)

{

    if (event\_id == WIFI\_EVENT\_AP\_STACONNECTED) {

        wifi\_event\_ap\_staconnected\_t\* event = (wifi\_event\_ap\_staconnected\_t\*) event\_data;

**ESP\_LOGI**(TAG, "station "MACSTR" join, AID=%d",

**MAC2STR**(event->mac), event->aid);

    } else if (event\_id == WIFI\_EVENT\_AP\_STADISCONNECTED) {

        wifi\_event\_ap\_stadisconnected\_t\* event = (wifi\_event\_ap\_stadisconnected\_t\*) event\_data;

**ESP\_LOGI**(TAG, "station "MACSTR" leave, AID=%d",

**MAC2STR**(event->mac), event->aid);

    }}

1. 初始化NVS内存

//Initialize NVS

    esp\_err\_t ret = **nvs\_flash\_init**();

    if (ret == ESP\_ERR\_NVS\_NO\_FREE\_PAGES || ret == ESP\_ERR\_NVS\_NEW\_VERSION\_FOUND) {

**ESP\_ERROR\_CHECK**(**nvs\_flash\_erase**());

      ret = **nvs\_flash\_init**();

    }

**ESP\_ERROR\_CHECK**(ret);

1. 调用 void wifi\_init\_softap(void)函数 启动Wifi

**ESP\_LOGI**(TAG, "ESP\_WIFI\_MODE\_AP");

**wifi\_init\_softap**();

完整示例代码：

#include <string.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "esp\_system.h"

#include "esp\_wifi.h"

#include "esp\_event.h"

#include "esp\_log.h"

#include "nvs\_flash.h"

#include "lwip/err.h"

#include "lwip/sys.h"

/\* The examples use WiFi configuration that you can set via project configuration menu.

   If you'd rather not, just change the below entries to strings with

   the config you want - ie #define EXAMPLE\_WIFI\_SSID "mywifissid"

\*/

// #define EXAMPLE\_ESP\_WIFI\_SSID      CONFIG\_ESP\_WIFI\_SSID

// #define EXAMPLE\_ESP\_WIFI\_PASS      CONFIG\_ESP\_WIFI\_PASSWORD

#define **EXAMPLE\_ESP\_WIFI\_SSID**      "Test\_Wifi"

#define **EXAMPLE\_ESP\_WIFI\_PASS**      "746169690"

// #define EXAMPLE\_ESP\_WIFI\_CHANNEL   CONFIG\_ESP\_WIFI\_CHANNEL

// #define EXAMPLE\_MAX\_STA\_CONN       CONFIG\_ESP\_MAX\_STA\_CONN

static const char \*TAG = "wifi softAP";

//当有终端连接（WIFI\_EVENT\_AP\_STACONNECTED）或断开（WIFI\_EVENT\_AP\_STADISCONNECTED）时，触发以下函数

static void **wifi\_event\_handler**(void\* arg, esp\_event\_base\_t event\_base,

                                    int32\_t event\_id, void\* event\_data)

{

    if (event\_id == WIFI\_EVENT\_AP\_STACONNECTED) {

        wifi\_event\_ap\_staconnected\_t\* event = (wifi\_event\_ap\_staconnected\_t\*) event\_data;

**ESP\_LOGI**(TAG, "station "MACSTR" join, AID=%d",

**MAC2STR**(event->mac), event->aid);

    } else if (event\_id == WIFI\_EVENT\_AP\_STADISCONNECTED) {

        wifi\_event\_ap\_stadisconnected\_t\* event = (wifi\_event\_ap\_stadisconnected\_t\*) event\_data;

**ESP\_LOGI**(TAG, "station "MACSTR" leave, AID=%d",

**MAC2STR**(event->mac), event->aid);

    }

}

void **wifi\_init\_softap**(void)

{

    //创建LwIP核心任务并初始化相关工作

**ESP\_ERROR\_CHECK**(**esp\_netif\_init**());

    // 创建系统事件任务并初始化应用程序事件的回调函数

**ESP\_ERROR\_CHECK**(**esp\_event\_loop\_create\_default**());

    // 创建具有TCP / IP堆栈的默认网络接口实例绑定AP。

**esp\_netif\_create\_default\_wifi\_ap**();

**ESP\_ERROR\_CHECK**(**esp\_event\_handler\_instance\_register**(WIFI\_EVENT,

                                                        ESP\_EVENT\_ANY\_ID,

                                                        &wifi\_event\_handler,

                                                        NULL,

                                                        NULL));

    //初始化Wi-Fi

    wifi\_init\_config\_t cfg = **WIFI\_INIT\_CONFIG\_DEFAULT**();

**ESP\_ERROR\_CHECK**(**esp\_wifi\_init**(&cfg));

    //Wi-Fi配置

    wifi\_config\_t wifi\_config = {

        .ap = {

            .ssid = EXAMPLE\_ESP\_WIFI\_SSID,

            .ssid\_len = **strlen**(EXAMPLE\_ESP\_WIFI\_SSID),

            .channel = EXAMPLE\_ESP\_WIFI\_CHANNEL,

            .password = EXAMPLE\_ESP\_WIFI\_PASS,

            .max\_connection = EXAMPLE\_MAX\_STA\_CONN,

            .authmode = WIFI\_AUTH\_WPA\_WPA2\_PSK

        },

};

    if (**strlen**(EXAMPLE\_ESP\_WIFI\_PASS) == 0) {

        wifi\_config.ap.authmode = WIFI\_AUTH\_OPEN;

}

**ESP\_ERROR\_CHECK**(**esp\_wifi\_set\_mode**(WIFI\_MODE\_AP));

**ESP\_ERROR\_CHECK**(**esp\_wifi\_set\_config**(WIFI\_IF\_AP, &wifi\_config));

    //Wi-Fi启动

**ESP\_ERROR\_CHECK**(**esp\_wifi\_start**());

**ESP\_LOGI**(TAG, "wifi\_init\_softap finished. SSID:%s password:%s channel:%d",

             EXAMPLE\_ESP\_WIFI\_SSID, EXAMPLE\_ESP\_WIFI\_PASS, EXAMPLE\_ESP\_WIFI\_CHANNEL);

}

void **app\_main**(void)

{

    //Initialize NVS

    esp\_err\_t ret = **nvs\_flash\_init**();

    if (ret == ESP\_ERR\_NVS\_NO\_FREE\_PAGES || ret == ESP\_ERR\_NVS\_NEW\_VERSION\_FOUND) {

**ESP\_ERROR\_CHECK**(**nvs\_flash\_erase**());

      ret = **nvs\_flash\_init**();

    }

**ESP\_ERROR\_CHECK**(ret);

**ESP\_LOGI**(TAG, "ESP\_WIFI\_MODE\_AP");

**wifi\_init\_softap**();

}

1. **Station**

[ESP32入门基础之ESP32作为 WIFI Station\_while（1）-CSDN博客\_esp32 wifi中继](https://blog.csdn.net/qq_42900996/article/details/116132421)

ESP32 idf4.4 自带例程：

/\* WiFi station Example

   This example code is in the Public Domain (or CC0 licensed, at your option.)

   Unless required by applicable law or agreed to in writing, this

   software is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR

   CONDITIONS OF ANY KIND, either express or implied.

\*/

#include <string.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "freertos/event\_groups.h"

#include "esp\_system.h"

#include "esp\_wifi.h"

#include "esp\_event.h"

#include "esp\_log.h"

#include "nvs\_flash.h"

#include "lwip/err.h"

#include "lwip/sys.h"

/\* The examples use WiFi configuration that you can set via project configuration menu

   If you'd rather not, just change the below entries to strings with

   the config you want - ie #define EXAMPLE\_WIFI\_SSID "mywifissid"

\*/

// #define EXAMPLE\_ESP\_WIFI\_SSID      CONFIG\_ESP\_WIFI\_SSID

// #define EXAMPLE\_ESP\_WIFI\_PASS      CONFIG\_ESP\_WIFI\_PASSWORD

#define **EXAMPLE\_ESP\_WIFI\_SSID** "CDTAC-3F"

#define **EXAMPLE\_ESP\_WIFI\_PASS** "CDTAC20170417"

#define **EXAMPLE\_ESP\_MAXIMUM\_RETRY** CONFIG\_ESP\_MAXIMUM\_RETRY

/\* FreeRTOS event group to signal when we are connected\*/

static EventGroupHandle\_t s\_wifi\_event\_group;

/\* The event group allows multiple bits for each event, but we only care about two events:

 \* - we are connected to the AP with an IP

 \* - we failed to connect after the maximum amount of retries \*/

#define **WIFI\_CONNECTED\_BIT** BIT0

#define **WIFI\_FAIL\_BIT** BIT1

static const char \*TAG = "wifi station";

static int s\_retry\_num = 0;

static void **event\_handler**(void \*arg, esp\_event\_base\_t event\_base,

                          int32\_t event\_id, void \*event\_data)

{

    if (event\_base == WIFI\_EVENT && event\_id == WIFI\_EVENT\_STA\_START)

    {

**esp\_wifi\_connect**();

    }

    else if (event\_base == WIFI\_EVENT && event\_id == WIFI\_EVENT\_STA\_DISCONNECTED)

    {

        if (s\_retry\_num < EXAMPLE\_ESP\_MAXIMUM\_RETRY)

        {

**esp\_wifi\_connect**();

            s\_retry\_num++;

**ESP\_LOGI**(TAG, "retry to connect to the AP");

        }

        else

        {

**xEventGroupSetBits**(s\_wifi\_event\_group, WIFI\_FAIL\_BIT);

        }

**ESP\_LOGI**(TAG, "connect to the AP fail");

    }

    else if (event\_base == IP\_EVENT && event\_id == IP\_EVENT\_STA\_GOT\_IP)

    {

        ip\_event\_got\_ip\_t \*event = (ip\_event\_got\_ip\_t \*)event\_data;

**ESP\_LOGI**(TAG, "got ip:" IPSTR, **IP2STR**(&event->ip\_info.ip));

        s\_retry\_num = 0;

**xEventGroupSetBits**(s\_wifi\_event\_group, WIFI\_CONNECTED\_BIT);

    }

}

void **wifi\_init\_sta**(void)

{

s\_wifi\_event\_group = **xEventGroupCreate**();//作用暂时未知

//  s1.1：主任务通过调用函数 esp\_netif\_init() 创建一个 LwIP 核心任务，并初始化 LwIP 相关工作。

//  s1.2：主任务通过调用函数 esp\_event\_loop\_create() 创建一个系统事件任务，并初始化应用程序事件的回调函数。在此情况下，该回调函数唯一的动作就是将事件中继到应用程序任务中。

//  s1.3：主任务通过调用函数 esp\_netif\_create\_default\_wifi\_sta() 创建有 TCP/IP 堆栈的默认网络接口实例绑定 station。

//  s1.4：主任务通过调用函数 esp\_wifi\_init() 创建 Wi-Fi 驱动程序任务，并初始化 Wi-Fi 驱动程序。

//  s1.5：主任务通过调用 OS API 创建应用程序任务。

    //创建LwIP核心任务并初始化相关工作（初始化底层TCP/IP栈。）

**ESP\_ERROR\_CHECK**(**esp\_netif\_init**());

    // 创建系统事件任务并初始化应用程序事件的回调函数

**ESP\_ERROR\_CHECK**(**esp\_event\_loop\_create\_default**());

    //创建默认的WIFI STA。在任何初始化错误的情况下，这个API中止

**esp\_netif\_create\_default\_wifi\_sta**();

    //定义一个名为cfg的wifi\_init\_config\_t结构体，wifi\_init\_config\_t的参数可由menuconfig配置

    wifi\_init\_config\_t cfg = **WIFI\_INIT\_CONFIG\_DEFAULT**();

    //初始名为cfg的结构体

**ESP\_ERROR\_CHECK**(**esp\_wifi\_init**(&cfg));

    //定义一个叫instance\_any\_id的句柄

    esp\_event\_handler\_instance\_t instance\_any\_id;

    //定义一个叫instance\_got\_ip的句柄

    esp\_event\_handler\_instance\_t instance\_got\_ip;

**ESP\_ERROR\_CHECK**(**esp\_event\_handler\_instance\_register**(WIFI\_EVENT,

                                                        ESP\_EVENT\_ANY\_ID,

                                                        &event\_handler,

                                                        NULL,

                                                        &instance\_any\_id));

**ESP\_ERROR\_CHECK**(**esp\_event\_handler\_instance\_register**(IP\_EVENT,

                                                        IP\_EVENT\_STA\_GOT\_IP,

                                                        &event\_handler,

                                                        NULL,

                                                        &instance\_got\_ip));

    //初始化wifi的配置

    wifi\_config\_t wifi\_config = {

        .sta = {

            .ssid = EXAMPLE\_ESP\_WIFI\_SSID,

            .password = EXAMPLE\_ESP\_WIFI\_PASS,

            /\* Setting a password implies station will connect to all security modes including WEP/WPA.

             \* However these modes are deprecated and not advisable to be used. Incase your Access point

             \* doesn't support WPA2, these mode can be enabled by commenting below line \*/

            .threshold.authmode = WIFI\_AUTH\_WPA2\_PSK,

            .pmf\_cfg = {

                .capable = true,

                .required = false},

        },

    };

    //设置模式为站（station）模式

**ESP\_ERROR\_CHECK**(**esp\_wifi\_set\_mode**(WIFI\_MODE\_STA));

    //初始化Station模式的Wifi配置

**ESP\_ERROR\_CHECK**(**esp\_wifi\_set\_config**(WIFI\_IF\_STA, &wifi\_config));

    //启动wifi

**ESP\_ERROR\_CHECK**(**esp\_wifi\_start**());

**ESP\_LOGI**(TAG, "wifi\_init\_sta finished.");

    /\* Waiting until either the connection is established (WIFI\_CONNECTED\_BIT) or connection failed for the maximum

     \* number of re-tries (WIFI\_FAIL\_BIT). The bits are set by event\_handler() (see above) \*/

    EventBits\_t bits = **xEventGroupWaitBits**(s\_wifi\_event\_group,

                                           WIFI\_CONNECTED\_BIT | WIFI\_FAIL\_BIT,

                                           pdFALSE,

                                           pdFALSE,

                                           portMAX\_DELAY);

    /\* xEventGroupWaitBits() returns the bits before the call returned, hence we can test which event actually

     \* happened. \*/

    if (bits & WIFI\_CONNECTED\_BIT)

    {

**ESP\_LOGI**(TAG, "connected to ap SSID:%s password:%s",

                 EXAMPLE\_ESP\_WIFI\_SSID, EXAMPLE\_ESP\_WIFI\_PASS);

    }

    else if (bits & WIFI\_FAIL\_BIT)

    {

**ESP\_LOGI**(TAG, "Failed to connect to SSID:%s, password:%s",

                 EXAMPLE\_ESP\_WIFI\_SSID, EXAMPLE\_ESP\_WIFI\_PASS);

    }

    else

    {

**ESP\_LOGE**(TAG, "UNEXPECTED EVENT");

}

    /\* The event will not be processed after unregister \*/

**ESP\_ERROR\_CHECK**(**esp\_event\_handler\_instance\_unregister**(IP\_EVENT, IP\_EVENT\_STA\_GOT\_IP, instance\_got\_ip));

**ESP\_ERROR\_CHECK**(**esp\_event\_handler\_instance\_unregister**(WIFI\_EVENT, ESP\_EVENT\_ANY\_ID, instance\_any\_id));

**vEventGroupDelete**(s\_wifi\_event\_group);

}

void **app\_main**(void)

{

    // Initialize NVS

    esp\_err\_t ret = **nvs\_flash\_init**();

    if (ret == ESP\_ERR\_NVS\_NO\_FREE\_PAGES || ret == ESP\_ERR\_NVS\_NEW\_VERSION\_FOUND)

    {

**ESP\_ERROR\_CHECK**(**nvs\_flash\_erase**());

        ret = **nvs\_flash\_init**();

    }

**ESP\_ERROR\_CHECK**(ret);

**ESP\_LOGI**(TAG, "ESP\_WIFI\_MODE\_STA");

**wifi\_init\_sta**();

}

1. **Wifi-Easyconnect**

[学一点Wi-Fi：DPP（WiFi Easy Connect）\_NIMEI583的博客-CSDN博客\_dpp协议](https://blog.csdn.net/qq_23087099/article/details/116765256)