STM32 MCU Development

STM32 单片机开发

-- ESP8266 简介及 AT 指令使用说明

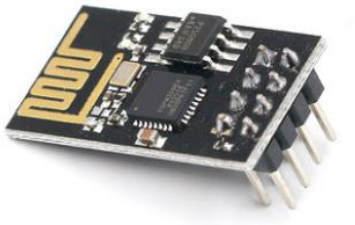
目录

1. ESP8266 介绍

1.1. ESP8266 简介

物联网，万物互联。这里涉及到的最基本的东西就是如何将所有的物联网设备连接在一起。最简单、最广泛使用的就是互联网。乐鑫公司推出的高性能、低功耗串口 WiFi 模块 ESP8266 应该是使用最广泛的一种 WIFI 模块之一了。为什么呢？

ESP8266 内置一个 Tensilica（泰思立达）Xtensa 架构的 32 位处理器 L106，具有 5 级流水线(ARM CortexM3 是 3 级流水线)，最大时钟速度为 160MHz，可以使用高达 16MB 的外部 SPI Flash。 该模块采用串口与 MCU（或其他串口设备）通信，内置 TCP/IP 协议栈，能够实现串口与 WIFI 之间的转换。通过该模块，传统的串口设备只需要简单的串口配置，即可通过 WiFi 传输自己的数据。



WiFi 具有两种功能模式：一种叫 AP(Access Point)模式，一种叫 Station 模式。AP 就是我们平时所说的热点，如 无线路由器，开了热点的手机等，这些 AP 设备可以允许其他设备(如手机，笔记本电脑等)输入热点名(SSID)和密码(也可不设置密码)后连接上网；Station 则是前面说的连接 AP 的设备，如手机，笔记本电脑等。ESP8266 除支持上述两种模式以外，还可以支持第三种模式：AP+Station，即：将 AP 和 Station 的功能合二为一，它主要是实现无线桥接的功能，该模式应用的场景不多，这里不作介绍。

ESP8266 模块自身带有高性能的 MCU，因此它既可以通过串口连接为外部 MCU 提供 WiFi 通信功能；也可以让用户直接在模块内置的 MCU 上基于 RTOS SDK 进行软件编程，开发出具有低功耗、低成本 的 WiFi 连接产品，如市面上绝大部分的 WiFi 智能插座基本上都是直接使用 ESP8266 模块作主控芯片开发的。

STM32 单片机在与 ESP8266 进行串口通信时采用 AT 命令进行通信，这样想要做 ESP8266 WiFi 模块的程序开发的话，那我们首先得学习并熟练掌握 ESP8266 WiFi 模块的 AT 指令。

1.2. AT 命令

AT 即 Attention，它由拨号调制解调器(Modem)的发明者贺氏公司(Hayes)为了控制 Modem 发明的控制协议。协议本身采用文本，每个命令均以 AT 打头,因此得名。90 年代初，AT 指令仅被用于 Modem 操作。

几年后，主要的移动电话生产厂商诺基亚、爱立信、摩托罗拉和 HP 共同为 GSM 研制了一整套 AT 指令，其中就包括对 SMS 的控制。AT 指令在此基础上演化并被加入 GSM 07. 05 标准以及现在的 GSM07. 07 标准，其中拨打电话、收发短信、收发传真等全部由 AT 命令实现。而在随后的 GPRS 控制，3G 模块，以及工业上常用的 PDU，均采用 AT 命令集来控制，这样 AT 指令也就成为了完全标准化和比较健全的标准。

AT 指令是以 AT 作为开头，\r\n 字符结束的字符串，每个指令执行成功与否都有相应的返回。其他的一些非预期的信息(如有人拨号进来、线路无信号等)，模块将有对应的一些信息提示，接收端可做相应的处理。

如下图所示，这是我在树莓派上给 NB-IoT 模块发送的 AT 指令，不同模块的 AT 命令可能不一样的，这要对着模块的 AT 指令手册来查看：

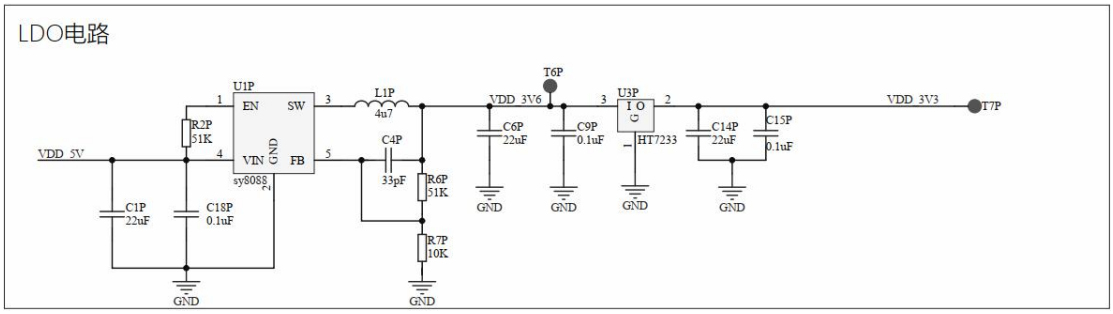
AT 指令可以细分为四种类型：

类型	指令格式	描述
执行指令	AT+<x>	该命令用于执行受模块内部程序控制的变参数不可变的功能。
测试指令	AT+<x>=?	该命令用于该命令用于查询设置指令的参数以及取值范围。
查询指令	AT+<x>?	该命令用于返回参数的当前值。
设置指令	AT+<x>=<…>	该命令用于设置用户自定义的参数值。

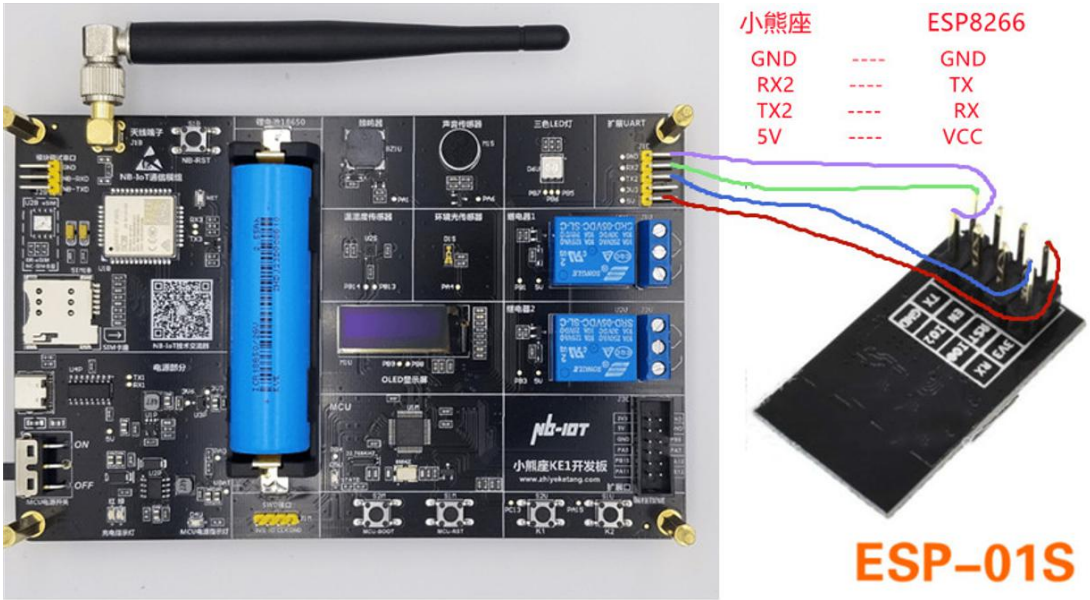
1.3. 硬件连接

ESP8266 WiFi 需要采用 3.3V 供电，但由于小熊座 NB-IoT 开发板上的所有 3.3V 电源均是 LDO 电路由 USB TypeC 接口提供的 5V 降压所得，这里的电流较低，不足以让 WiFi 模块正常工作。

所以在连接 ESP8266 WiFi 模块到小熊座 NB-IoT 开发板上时，有些模块可以用这个 3.3V 正常工作，但有些则不可以。此时可以将供电连到 5V 上，这里存在个风险电压过高可能会烧芯片（好在我买的 ESP-01S 型 WiFi 模块能够正常工作）。如果采用 5V 供电，USB TypeC 接口连接 PC 时尽量别用 USB 3.0 接口，这个接口的电流较高可能导致芯片发热厉害，从而烧坏芯片。



ESP8266 模块提供 TTL 串口通信接口，而小熊座 NB-IoT 开发板上的扩展 UART2 串口也是 TTL 电平，这样可以直接连接 ESP8266 WiFi 模块进行通信。下面是小熊座 NB-IoT 开发板与 ESP8266 串口 WiFi 模块连接的示意图，这里我们连接 5V 供电。



2. 串口转发程序

2.1. 串口转发程序

在前面的课程里，我们已经学习了 STM32 的串口编程使用，能够通过 USB TypeC 接口提供的串口(USART1)实现数据的收发，现在我们使用开发板的 USART2 串口连接了 EPS8266 模块，如果能用 PC 上的串口来调试 ESP8266 WiFi 模块就好了，这样学习起来比较简单、方便。

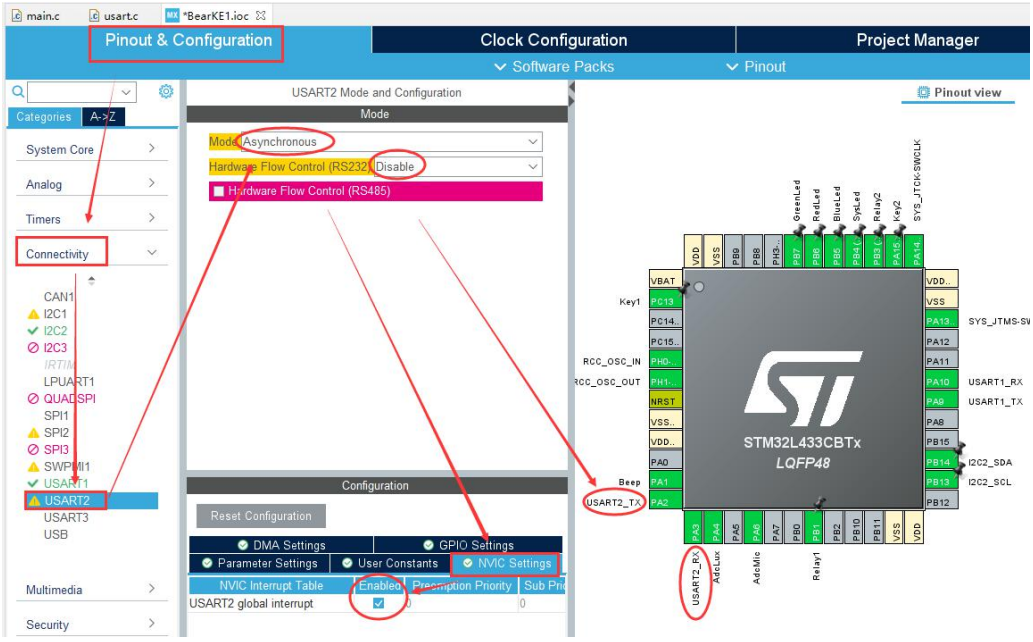
事实上，我们可以在 STM32 单片机上通过软件编程实现一个串口接收转发的程序，即：

- 从 USART1 收到的数据(PC 端发过来的)转发给 USART2(ESP8266 WiFi 模块)；
- 从 USART2 收到的数据(ESP8266 WiFi 模块)转发给 USART1(PC 端)

这样就可以直接使用 PC 上的串口来学习 ESP8266 WiFi 模块的使用了，接下来我们将编程实现该程序。

2.2. STM32CubeMX 配置

在 STM32CubeMX 中使能 USART1、USART2 串口及中断接收，并按 Ctrl+S 生成初始化代码。



2.3. 修改 usart.c/h 文件

修改 usart.c 文件，参考 USART1 的代码添加 USART2 串口及其中断接收处理函数。

```
... ...
在这里参考串口 usart1 添加 usart2 的接收 buffer 相关定义
/* USER CODE BEGIN 0 */
static uint8_t    s_uart1_rxch;
char              g_uart1_rxbuf[256];
uint8_t           g_uart1_bytes;

static uint8_t    s_uart2_rxch;
char              g_uart2_rxbuf[256];
uint8_t           g_uart2_bytes;
/* USER CODE END 0 */
... ...

在这个函数里使能 usart2 的接收中断，中断接收的每个字节将临时存储在 s_uart2_rxch 变量中
void MX_USART2_UART_Init(void)
{
  ... ...
  /* USER CODE BEGIN USART2_Init 2 */
  HAL_UART_Receive_IT(&huart2 , &s_uart2_rxch, 1);
  /* USER CODE END USART2_Init 2 */
}

... ...
/* USER CODE BEGIN 1 */

在这里参考 usart1 添加 usart2 的中断接收回调函数。
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
        if (huart->Instance == USART1)
        {
             if( g_uart1_bytes< sizeof(g_uart1_rxbuf) )
             {
                  g_uart1_rxbuf[g_uart1_bytes++] = s_uart1_rxch;
             }
             HAL_UART_Receive_IT(&huart1 , &s_uart1_rxch, 1);
        }
```

```
        if (huart->Instance == USART2)
        {
            if( g_uart2_bytes< sizeof(g_uart2_rxbuf) )
            {
                g_uart2_rxbuf[g_uart2_bytes++] = s_uart2_rxch;
            }
            HAL_UART_Receive_IT(&huart2 , &s_uart2_rxch, 1);
        }
}

在这里添加 usart1 和 usart2 的收发转发功能函数，main()函数中将会调用。
void uart_forward(void)
{
        if( strstr(g_uart1_rxbuf, "\r\n"))
        {
            HAL_UART_Transmit(&huart2 , (uint8_t *)g_uart1_rxbuf, g_uart1_bytes, 0xFF);
            clear_uart1_rxbuf();
        }

        if (g_uart2_bytes > 0)
        {
            HAL_Delay(100);  /* Wait AT command reply receive over */
            HAL_UART_Transmit(&huart1 , (uint8_t *)g_uart2_rxbuf, g_uart2_bytes, 0xFF);
            clear_uart2_rxbuf();
        }
}
/* USER CODE END 1 */
```

修改 usart.h 文件，添加相关全局变量及函数声明

```
... ...

/* USER CODE BEGIN Private defines */
extern char                    g_uart1_rxbuf[256];
extern uint8_t                 g_uart1_bytes;
#define clear_uart1_rxbuf()    do { memset(g_uart1_rxbuf, 0, sizeof(g_uart1_rxbuf)); \
                                    g_uart1_bytes=0; } while(0)

在这里添加 uar2 接收 buffer 相关变量声明，并添加一个宏 clear_uart2_rxbuf()用来清除接收 buffer 里的数据
extern char                    g_uart2_rxbuf[256];
extern uint8_t                 g_uart2_bytes;
#define clear_uart2_rxbuf()    do { memset(g_uart2_rxbuf, 0, sizeof(g_uart2_rxbuf)); \
                                    g_uart2_bytes=0; } while(0)
```

```
添加串口接收转发函数的声明
extern void uart_forward(void);

/* USER CODE END Private defines */
... ...
```

2.4. 修改 main.c 文件

修改 main.c 文件，在 main()函数的 while(1)循环里添加 uart_forward()函数调用，并注释掉之前的所有代码，只做串口的接收转发。

```
  ... ...
  while (1)
  {
          uart_forward();
#if 0
          proc_uart1_recv();

          /* check report time arrive or not, use time_after() macro to avoid time overflow */
          if( time_after(HAL_GetTick(), last_time+3000) )
          {
           /* start sample and report data to PC by serial port */
           report_tempRH_json();

           /*update last report time */
           last_time = HAL_GetTick();
          }
#endif
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }... ...
```

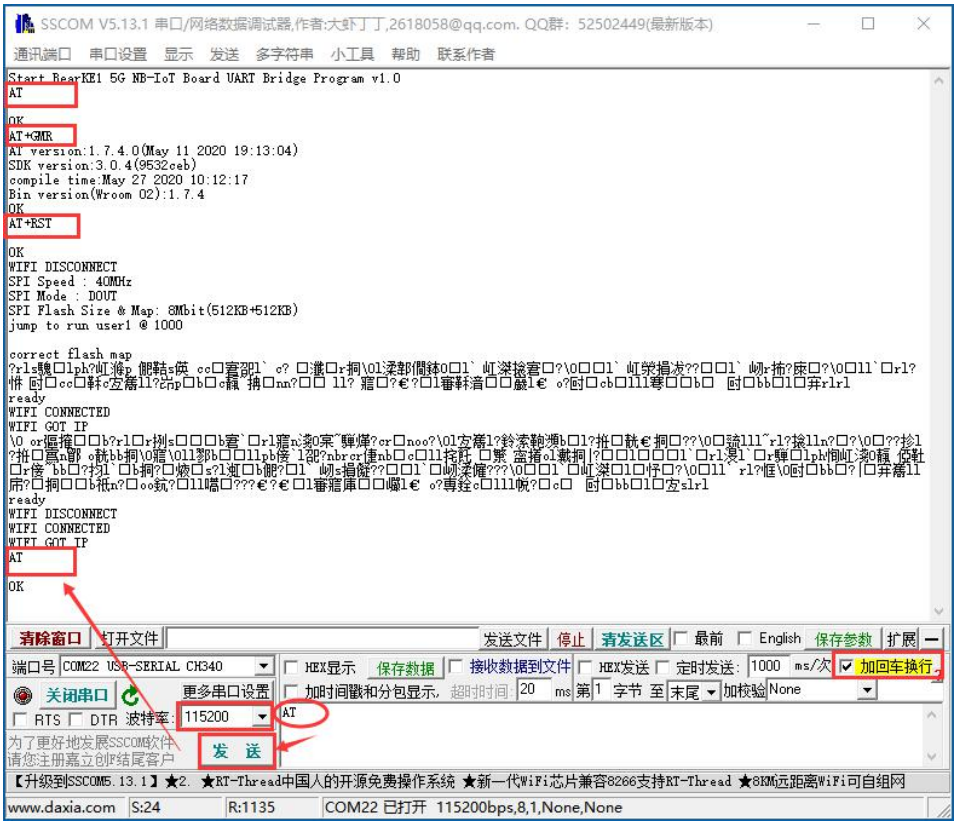
2.5. 运行测试

重新编译并烧录运行程序，使用串口调试助手打开 PC 上相应的串口。EPS8266 模块串口波特率默认为 115200bps，另外 AT 命令都是以回车结尾，所以参考下图做如下设置：

1， 配置串口波特率为 115200；

2， 勾选“加回车换行”选项，这样发送 AT 命令时不需要再添加回车；

接下来我们就可以通过串口给 ESP8266 WiFi 模块发送 AT 命令了，另外 PC 上也能正常收到 ESP8266 WiFi 模块的 AT 命令的回应信息。



3. AT 指令学习

3.1. AT 指令

每一个使用 AT 指令通信的模块都有自己详细的 AT 指令使用说明文档，从上面执行 AT+GMR 命令输出的打印信息可以看到，我们的 ESP8266 WiFi 模块的软件版本为 v1.7.4，其对应的 AT 指令 Version 0.23。模块里的软件固件版本不一样，其 AT 指令可能也不大一样。

要想深入学习了解 ESP8266 WiFi 模块的 AT 指令使用，则需要查阅并学习其 AT 指令手册文档《ESP8266 AT 指令集 V0.23b1_2015.pdf》。这里面详细介绍了该模块所支持的 AT 指令及其使用方法。

接下来我们将讲解使用 ESP8266 收发数据所需要的一些基本命令。需要注意的是如果所使用的配置命令带有_CUR 后缀，则表示只更改当前系统配置，并不写入 Flash 存储器中，重启复位后失效。如果想要模块复位之后也还保持原配置，则需要使用带_DEF 后缀的命令，该命令将会把配置写入到 Flash 存储器中，系统重启后仍然有效。

3.2. WiFi 初始化命令

AT 命令	说明
AT	AT 命令用来确认模块是否正常工作以及串口通信是否正常
AT+GMR	获取 WiFi 模块的软件固件版本信息
AT+RST	重启、复位 WiFi 模块
AT+CWMODE_CUR=1	设置 ESP8266 WiFi 模块工作在 Station 模式
AT+CWDHCP_CUR=1,1	设置使能 ESP8266 WiFi 模块 Station 模式的 DHCP 服务
AT+CIPSTA_CUR	静态设置 ESP8266 的 IP 地址，子网掩码和默认网关

WiFi 技术在做无线局域网组网时有两个功能模块：AP(Acess Point) 和 Station。

- **AP 模式**提供无线接入功能，允许其它无线设备接入并提供数据访问，无线路由器就工作在该模式下。我们的手机开热点即工作在该模式下，因为是由软件模拟实现的，所以叫 SoftAP；

- **Station 模式**即无线终端，它可以连接 AP 并加入到无线网络中，其本身并不提供无线接入，一般我们 PC 上的无线网卡就工作在该模式下；

ESP8266 WiFi 模块即可以工作在 AP 模式下，这样就可以作热点让手机、笔记本等连接上来组建无线局域网实现无线通信；也可以工作 Station 模式下连接无线路由器实现上网功能。上面的 AT+CWMODE_CUR 命令就可以配置 ESP8266 模块的工作模式：

1. 其值为 1 配置为 Station 模式，即无线网卡工作模式；
2. 其值为 2 配置为 SoftAP 模式，即无线路由器工作模式；
3. 其值为 3 配置为 SoftAP+Station 模式，即无线桥接模式(WDS)，在信号不好时作无线中继使用；

我们知道，计算机网络在局域网通信时每台机器都需要一个独一无二的 IP 地址，而我们的手机和 PC 平时都没有配置，怎么就有 IP 地址能够上网了呢？ 这是因为我们的 PC、手机默认都开启了 DHCP 服务，该服务使得我们在连上 WiFi 时就能够从路由器上动态获取 IP 地址。

在 ESP8266 模块上，我们可以使用 AT+CWDHCP_CUR 命令开启 Station 模式和 AP 模式的 DHCP 服务。显然，当工作在 Station 模式时，DHCP 将工作在客户端模式下，用来动态请求获取 IP 地址；而当工作在 AP 模式时，DHCP 则工作中服务器模式下，用来给连接上来的无线设备动态分配 IP 地址。

如果我们想要静态设置固定 IP 地址，则可以使用 AT+CIPSTA_CUR 命令来设置，其使用方法为：

AT+CIPSTA_CUR=“192.168.0.100”,"192.168.0.1","255.255.255.0"

- **192.168.0.1** 是默认网关地址，其值应该为 WiFi 模块所连接的无线路由器的 LAN 口 IP 地址,不同的路由器这个 IP 地址不一样，需要要根据自己路由器的配置来修改；

- **192.168.0.100** 是 ESP8266 要配置的 IP 地址，它不能与其他设备 IP 地址相冲突，其值应该根据路由器的 IP 来调整。 假设路由器的 IP 地址为 192.168.1.1，则该 IP 地址一般为 192.168.1.x（如 192.168.1.100）,实际应该为多少应该由后面的子网掩码(255.255.255.0)来决定；

- 255.255.255.0 为子网掩码，其主要是用来划分网络号和主机号，其值应该与路由器上的子网掩码号保持一致。

3.3. 子网掩码划分

在计算机网络设置中，IP 地址被子网掩码划分成网络号和主机号两个部分，这里的网络号和设备号与日常所见的固话号码里的区号和座机号一样。例如对于固话号码 02766668888，其中前 3 位 027 为武汉市区号，剩下的 66668888 即为座机号。

- 如果另外有个座机号码为 02766665555，其前三位也为 027，即区号相同，这时两台座机互相通信就为市话通信较为便宜；
- 如果另外有个座机号码为 01066667777，其前三位为 010，此时区号并不相同，此时两台座机互相通信则为长途通信，需要电话交换机转接发送，所以通话费用较贵；

将 IP 地址与子网掩码作按位与运算，即可将 IP 地址划分成网络号和主机号两部分。在局域网内通信时，只有网络号相同的两台主机才能互相通信。假设某个路由器的 IP 地址为 192.168.3.1，其子网掩码为 255.255.255.0，那我们将 32 位的 IP 地址和子网掩码转换成二进制形式，然后做按位的与运算：

192.168.0.1：	**1100 0000**	**1010 1000**	**0000 0011**	**0000 0001**
255.255.255.0：	**1111 1111**	**1111 1111**	**1111 1111**	**0000 0000**
&	---	---	---	---
	1100 0000	1010 1000	0000 0011	0000 0000
	192	168	3	0

从上面的运算过程我们可以看出，将 IP 地址 192.168.3.1 与子网掩码 255.255.255.0 作按位的与运算，其结果为 192.168.3.0，这个值即为该 IP 地址的网络号。另外从中也可以看出，子网掩码 255.255.255.0 即从高位开始有 24 个位为连续的 1，其意思就是将 IP 地址的前 24 位“掩掉”作为网络号(这也就是 netmask 的意思)，那剩下的 8 个位(0000 0001)就是其设备号，其值为 1。

假设 ESP8266 通过 WiFi 连接到了该路由器下，然后我们静态设置其 IP 地址为 192.168.1.100，子网掩码为 255.255.255.0，此时使用相同的方法将 IP 地址与子网掩码作按位与运算，其结果(网络号)为 192.168.1.0。显然此时 ESP8266 的网络号 192.168.1.0 与路由器的网络号 192.168.3.0 不一致，则 ESP8266 即使连上了 WiFi，也不能与无线路由器通信。

显然，为了保证我们的 WiFi 模块连上无线路由器后能够正常上网，必须要根据所连路由器的 IP 地址来配置。比较简单的一种方式是将子网掩码号设置得与路由器一致，即 255.255.255.0；然后将 IP 地址的前 24 位(即前三个字段网络号由子网掩码决定)与路由器保持一致，即 192.168.3；后面的 8 位(即最后一个字段主机位)与其它主机不冲突即可，如 100。此时的 IP 地址为 **192.168.3**.100，子网掩码为 255.255.255.0，这样就可以与无线路由器正常通信了。

显然，如果要静态配置设备的 IP 地址则必须要知道无线路由器的网络配置，而使用 DHCP 动态获取 IP 地址就比较简单方便，直接有路由器动态分配给我们即可。但使用 DHCP 动态获取也有个缺陷，即我们不知道 ESP8266 的 IP 地址，因为它是由路由器动态分配的。

3.4. 无线连接命令

AT 命令	说明
AT+CWJAP_CUR="Router_SSID","Password"	该命令用来连接指定的无线路由器
AT+CIPSTA_CUR?	该命令用来查看 ESP8266 当前的 IP 地址
AT+PING="192.168.0.1"	该命令用来测试与目标主机的连通性

AT+CWJAP_CUR 命令用来临时连接指定的无线路由器，其中第一个参数为无线路由器的热点名(专业术语叫 SSID，Service Set Identifier)，第二个参数为无线路由器的连接密码；

使用 **AT+CWJAP_CUR** 命令连上无线路由器后，我们需要使用 **AT+CIPSTA_CUR?** 命令查看 DHCP 是否正常获取到 IP 地址，如果没有获取到 IP 地址肯定不能通信。

在获取到 IP 地址后，我们还可以使用 **AT+PING** 命令测试目标主机是否可达，如 ping 一下路由器的 IP 地址看是否能够跟路由器正常通信，ping 一下 **114.114.114.114** 看是否能正常上网。

3.5. 数据收发命令

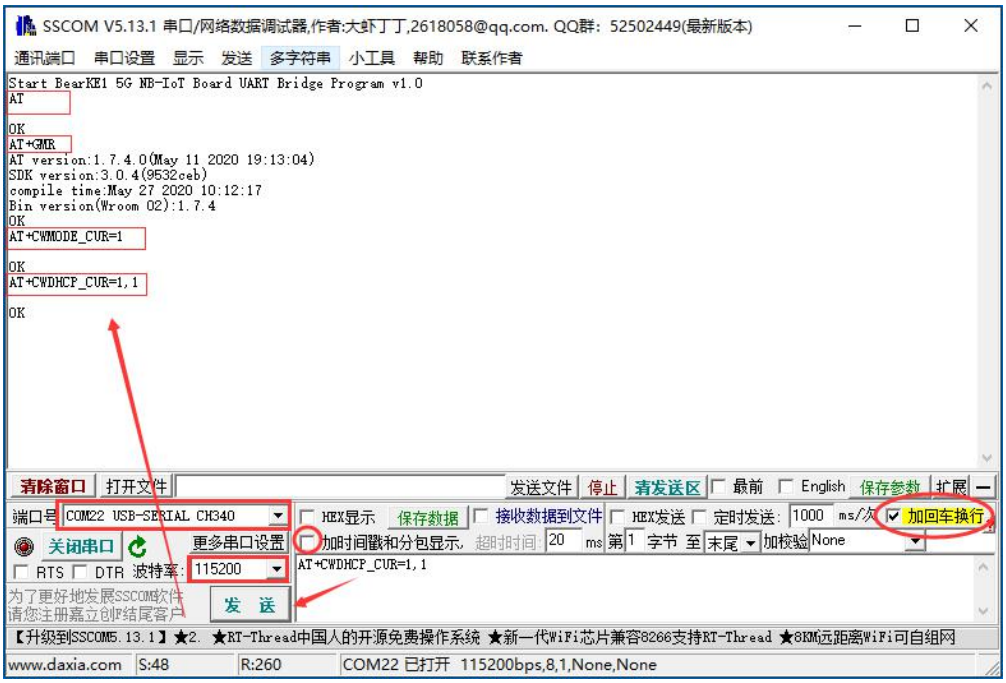
AT 命令	说明
AT+CIPMUX=0	该命令用来禁用多个 socket 连接，一般只连接一个目标服务器。
AT+CIPSTART="TCP","192.168.0.100",12345	该命令用来连接指定的目标 socket 服务器
AT+CIPSEND=5	该命令用来发送 5 个字节的数据，在收到模块回应的‘>’字符后，开始输入要发送的数据内容。
AT+CIPCLOSE	该命令用来断开 Socket 连接

注意，从 WiFi 模块上接收数据并不需要使用 AT 指令，这是因为 ESP8266 收到数据后将会自动转发到相应的串口上，我们只需要该串口上接收数据即可。

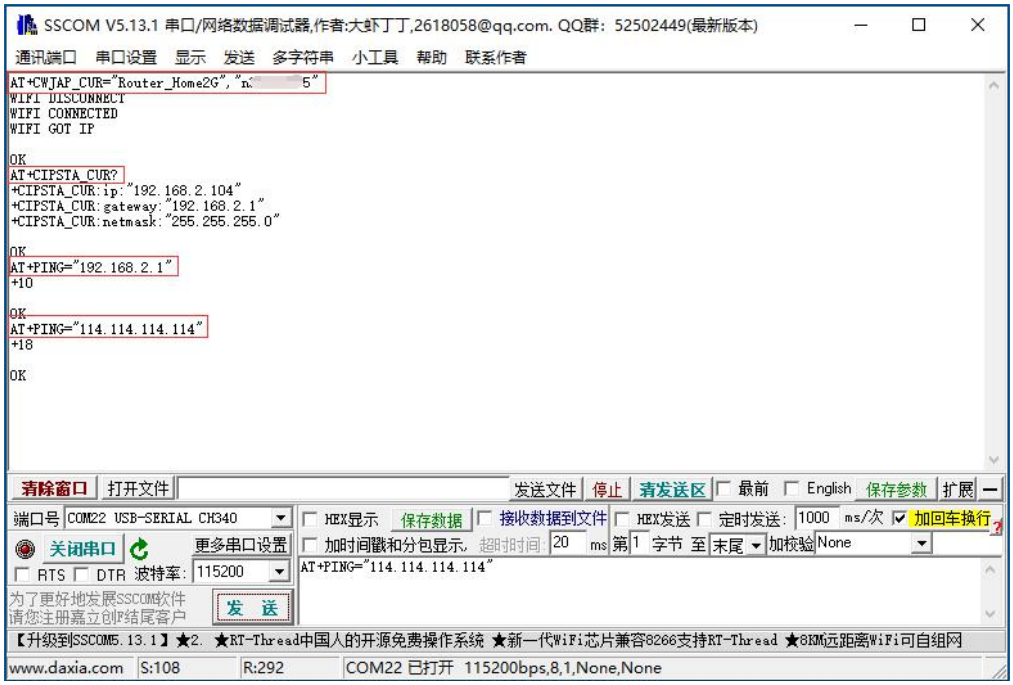
上面只是讲解了少部分常用的 AT 命令，如果实现其他功能（如进入低功耗模式、设置上电启动配置），则需要阅读相关 AT 命令手册。下面就在了解了上面这些命令的基础上，在 PC 上实现 ESP8266 上的数据收发。

4. AT 命令使用示例

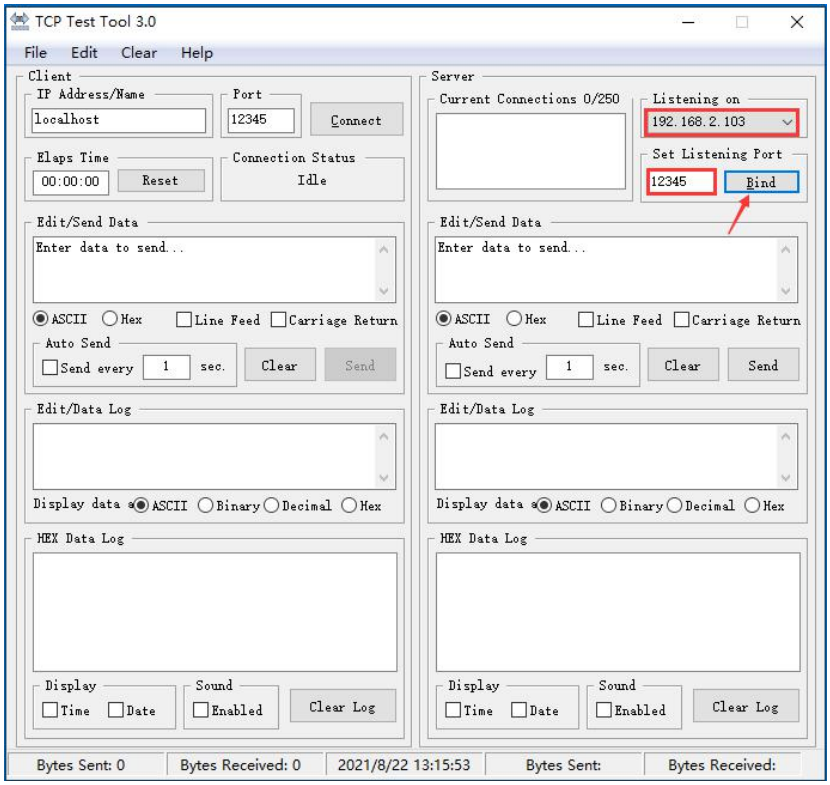
首先运行串口调试软件，并正确打开、配置好串口。接下来输入并设置相应的 WiFi 初始化命令。配置 ESP8266 工作在 Station 模式，并启动 DHCP 动态获取 IP 地址。



接下来我们使用 AT 命令让 ESP8266 连接到我们 PC 连接的无线路由器，并使用 AT+PING 命令测试网络的联通性。



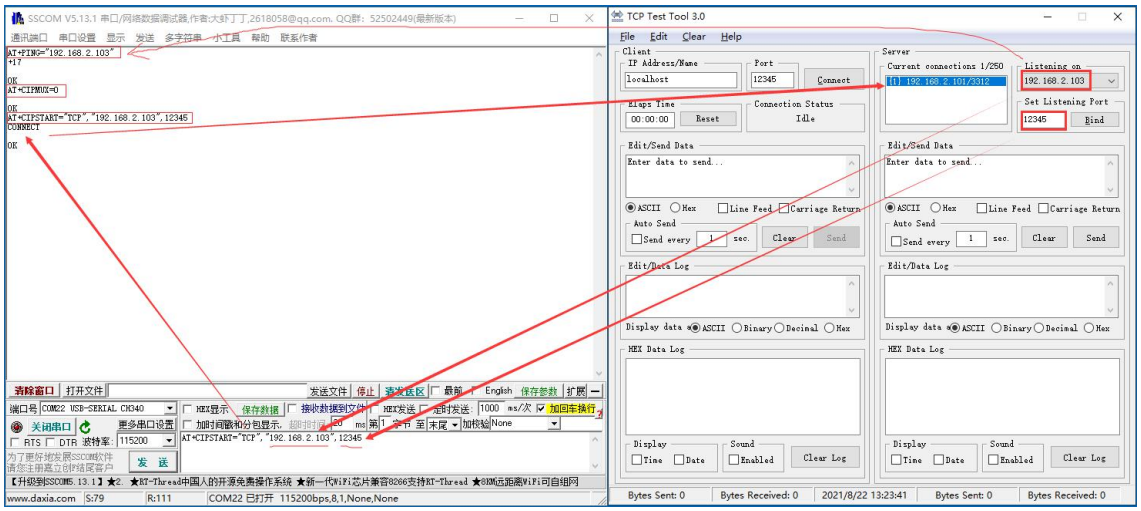
计算机网络通信都是基于 C/S 模式(Client/Server)，此时我们可以运行 TCP Test Tools 工具软件，并配置让其监控端口号 12345。



Windows 系统的防火墙默认会封 Ping 报文以及这里的 Socket 连接，为方便接下来的测试，在“Windows 安全中心里”关闭 Windows 系统下所有防火墙。

接下来，可以使用 AT 命令连接 Socket 服务器：

1. AT+PING="192.168.2.103"命令测试看与 PC 是否能够正常通信；
2. AT+CIPMUX=0 命令禁用多路 socket 连接；
3. AT+CIPSTART="TCP","192.168.2.103",12345 命令连接 Socket 服务器的 12345 端口；



待正常连接到 socket 服务器后，接下来我们就可以实现数据的收发了。其中数据的收发过程为：

1. 发送 AT+CIPSEND=5 命令，其中 5 为我们要发送的数据长度；
2. ESP8266 收到数据发送命令后，将会返回字符'>'表示等待开始接收要发送的数据；
3. ESP8266 在收到到该符号后，开始输入数据"Hello"。如果数据长度<5 个字节，WiFi 模块则继续等到 5 个字节后才发送；如果长度>=5 个字节，则只发送 5 个字节；
4. 这时候 TCP Test Tools 服务器程序将会收到"Hello"字符串；
5. 我们在 TCP Test Tools 服务端给客户端发送数据，这时候串口调试助手上将会收到"+IPD,7:Goodbye"字符串。其中"+IPD"为接收到数据的前导标志,7 为接收到的数据长度，冒号后面就是接收到的数据内容；
6. Socket 用完之后，可以使用 AT+CIPCLOSE 命令断开 Socket 连接。