程序介绍

1. 使用接线

板载有 ESP8266 的开发板不需要用杜邦线连接单独的 ESP8266 模块，直接使用跳线帽连接好相关引脚就可以使用板载的 ESP8266 了。 而对于没有板载 ESP8266 的开发板就需要按照例程的接线方式，来外接一个单独的 ESP8266 模块

对于板载有 ESP8266 的开发板，也可以使用外接 ESP8266 模块，但是要先断开板载 ESP8266 与串口的跳帽，断开之后再使用我们的例程

野火 STM32F103 指南者、F103 霸道和 F407 霸天虎开发板板载了 ESP8266 模块（指南者、霸道 V1、霸天虎 V1 板载的 esp8266 的 Flash 只有 512K，烧录不了 1MB 固件，只能使用基础通信例程，可以通过外接野火小智 WIFI_ESP8266-12F 模块等使用物联网 MQTT 例程）

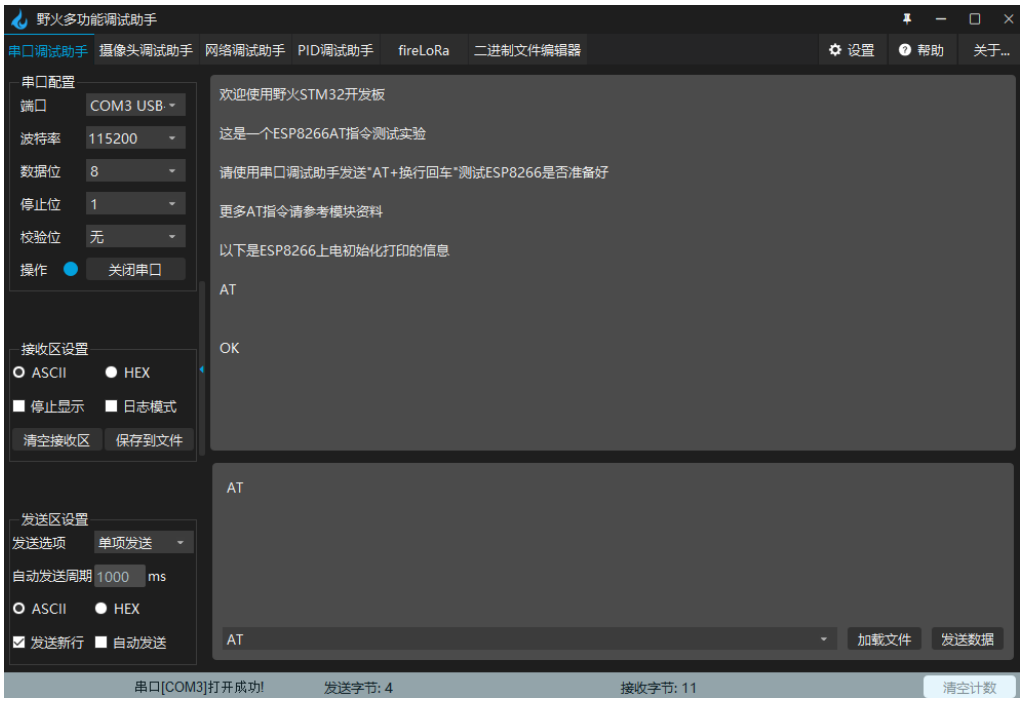
例程的具体引脚分配与接线方式请查看 **ESP8266 模块与 STM32 开发板引脚连接说明.xlsx**，结合实际源码内容查看。

2. 实验操作

基础通信例程：

ESP8266AT 指令测试：

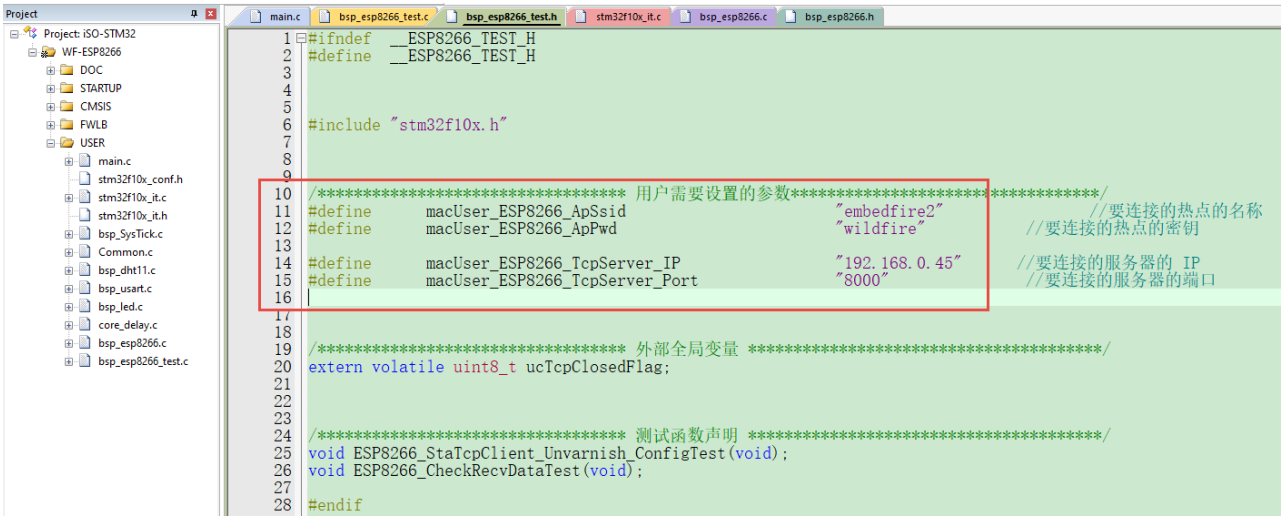
按照 esp8266 接口将开发板与模块连接，用 USB 线连接好电脑和开发板的串口接口，打开电脑上的野火多功能调试助手，波特率设置为 115200，打开与开发板连接的端口， 如下图所示，下载程序后上位机会打印信息，发送 “AT+换行回车” 后，如果 ESP8266 模块工作正常的话开发板会回复 “OK”



WIFI 透传：

1. 设置 WIFI 信息

准备一个具有 WiFi 功能的路由器，把调试使用的电脑连接到该路由器上，并查看该电脑的 IP 地址下载程序之前，程序需要知道你所在的局域网的 WIFI 名称和连接密码（即路由器 WiFi 的名称和密码）、以及你电脑的 IP 地址，这三个信息在 bsp_esp8266_test.h 头文件中修改。如下图中所示：



2. 打开配置串口调试助手和网络调试助手

给开发板上电，把编译好的程序下载到开发板，用 USB 线连接好电脑和开发板的串口接口（板子上面一般标有“USB 转串口”字样），然后打开野火多功能调试助手。

打开串口调试助手： 设置串口号、配置波特率为 115200，打开串口

打开网络调试助手：设置协议类型为 TCP Server、IP 地址、端口号，端口号可以自定义，也可直接用 8080 端口（如果后续步骤一直没有出现连接，尝试换个任意数字的端口号，程序和网络助手中填一致），然后点“启动监听”

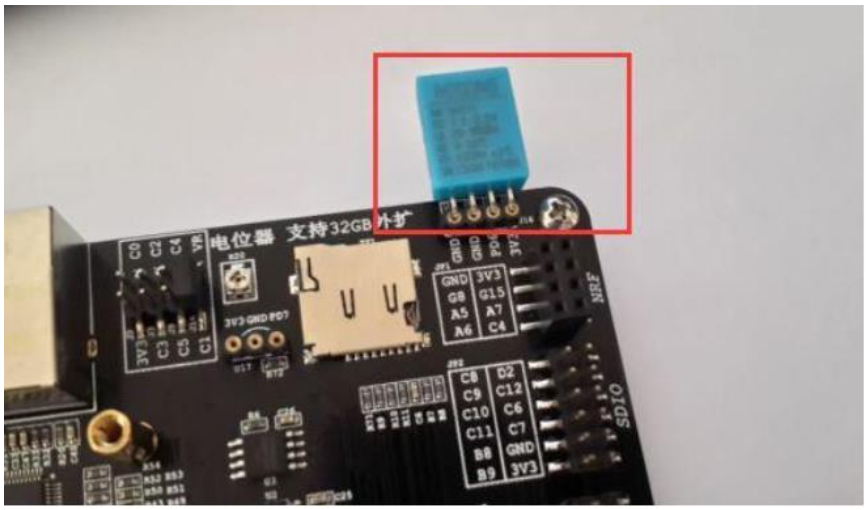
注意：ESP8266 WiFi 模块必须要和电脑处于同一个局域网内，也就是说 WiFi 模块和电脑都要连接同一个路由器，一般不建议用手机当热点使用。网络调试助手不能随意修改本地主机地址，打开软件时显示的是什么就用什么。端口号一般可以随意设置，只要不与电脑中其他程序冲突就行

说明：若无线路由可访问互联网且调试用的电脑具有独立的公网 IP，本程序也支持公网访问（把操作中的 IP 改成电脑的公网 IP）。 ！！对公网访问不熟悉的用户，实验时请直接按以上的说明用局域网方式调试

注意：在网络调试助手里面设置的 IP 地址和端口要和程序里设置的一样，且程序里设置的 IP 地址应当是你电脑的 IP 地址！具体设置如下：

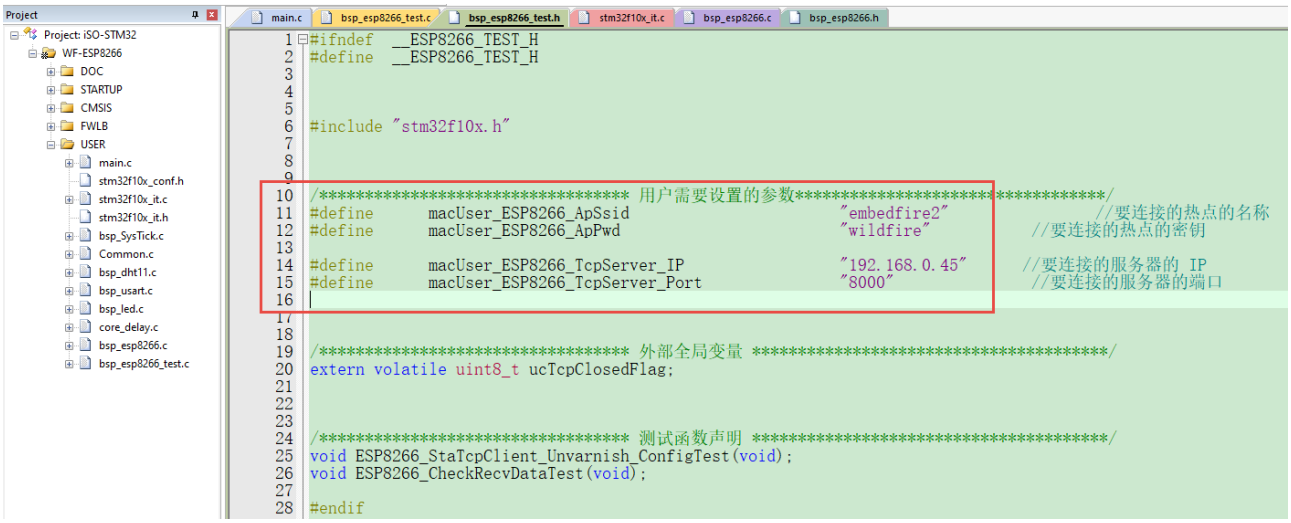
向电脑网络助手上传 DHT11 温湿度：

把开发板读取到的温湿度信息，通过 WIFI 传到局域网的电脑上的网络调试助手上显示。记得在板子上插好 DHT11 温湿度模块。这个程序是在 WIFI 透传的程序中修改而来，实际上就是将透传的数据改成 DHT11 的数据。开发板和 DHT11 连接的图片如下所示



1. 设置 WIFI 信息

准备一个具有 WiFi 功能的路由器，把调试使用的电脑连接到该路由器上，并查看该电脑的 IP 地址下载程序之前，程序需要知道你所在的局域网的 WIFI 名称和连接密码（即路由器 WiFi 的名称和密码）、以及你电脑的 IP 地址，这三个信息在 bsp_esp8266_test.h 头文件中修改。如下图中所示：



2. 打开配置串口调试助手和网络调试助手

给开发板上电，把编译好的程序下载到开发板，用 USB 线连接好电脑和开发板的串口接口（板子上面一般标有“USB 转串口”字样），然后打开野火多功能调试助手。

打开串口调试助手： 设置串口号、配置波特率为 115200，打开串口

打开网络调试助手：设置协议类型为 TCP Server、IP 地址、端口号，端口号可以自定义，也可直接用 8080 端口（如果后续步骤一直没有出现连接，尝试换个任意数字的端口号，程序和网络助手中填一致），然后点“启动监听”

注意：在网络调试助手里面设置的 IP 地址和端口要和程序里设置的一样，且程序里设置的 IP 地址应当是你电脑的 IP 地址！具体设置如下：

手机 app 控制例程：

例程与手机在同一局域网内建立 TCP 连接后通讯，程序中根据 bsp_esp8266_test.h 中的 #define BUILTAP_TEST 宏选择控制模块为 AP 还是 STA 模式

例程中默认注释 BUILTAP_TEST 宏以使用 STA 模式

STA 模式下，在 bsp_esp8266_test.c 中与之前例程一样，填写好需要连接的热点名字和密码，手机也连接同一个热点。执行程序后查看串口打印中模块连上热点后获取的 IP，打开 APP 填写以上 IP 和程序中端口号即可连接

AP 模式下，在 bsp_esp8266_test.c 中填写一个自定义的开启热点名字与密码（使用 OPEN 无需密码，改为 WPA2_PSK 启用密码），和下面填写自身 IP（STA 模式时使用 DHCP 获取 IP 与这处无关）与端口号。 执行程序后手机连接模块开启的热点，打开 APP 填写以上 IP 和程序中端口号即可连接

手机 APP 控制过程通过不同按钮发送不同的字符串，开发板程序在接收完一次数据后对字符串解析执行对应的流程

见例程中 bsp_esp8266_test.c 中的 ESP8266_CheckRecv_SendDataTest 函数流程，此时流程中未开启透传模式，利用 strstr 先定位接收数据中自定义字符串的位置，再依次偏移获取后续的字符做判断，可以在第一个 if 语句中打断点， debug 执行，用手机 APP 按一次按钮发送数据，停在断点后用 watch 工具查看 Data_RX_BUF 的内容方便理解字符串解析流程

例程中使用的手机 APP 年代久远，目前没有源码文件，从上面描述与结合源码理解，本质还是建立 TCP 连接后发送自定义数据，因此这个过程可以用任意类似网络调试助手功能的手机 APP 代替，可以自己修改简化解析的字符串过程

APP 可以控制开发板上面的 RGB 灯，蜂鸣器，开发板会将 DHT11 采集的温湿度信息发送给 APP。APP 界面如下图所示：

1.APP 中的 IP 地址和端口号需要填的是 WIFI 模块 ESP8266 内部的 Server 信息，这在例程里已经设置好了（可以修改），根据例程里的进行填写，旁边的连接按钮默认是白色，当点击连接成功之后会变成黄褐色

2.APP 中设备状态栏返回的是设备的状态信息， LED 的亮灭和温湿度信息，因为可以同时 5 台手机控制，当一台手机控制板子的 LED 改变状态时。另一台手机界面的状态变化可以通过这个按钮选择自动更新还是手动更新，自动更新的时间一般为 3s 最合适

3.APP 中三个 LED 的图标可控制开发板中三个 LED 的亮灭，灭的时候全部显示灰色，亮的时候三个 LED 对应板子上的 3 个 LED 灯，如果板子上的是 RGB 灯，则对应 RGB 灯的红绿蓝三种颜色亮起

4.温度和湿度是开发板上的 DHT11 传回来的信息，前提是开发板上插有温湿度传感器 DHT11

5.蜂鸣器按钮可以控制蜂鸣器的开和关

注意：有一些手机连接了 WiFi 模块的热点之后，会检测到和提示 WiFi 模块没有网络可用，会自动切换回数据流量，导致 APP 连接模块失败，解决办法就是暂时先关闭数据流量

3. 例程说明

基础通信例程：

ESP8266AT 指令测试：

以指南者开发板为例，ESP8266_Init 是 ESP8266 的初始化函数，在该函数中初始化了 ESP8266 用到的 GPIO 引脚，以及初始化了 ESP8266 用到的 USART3，并且开启了串口中断，代码如下。

ESP8266_Init 函数

```
void ESP8266_Init ( void )
{
   /* 初始化 ESP8266 用到的 GPIO 引脚 */
   ESP8266_GPIO_Config ();

   /* 初始化 ESP8266 用到的 USART */
   ESP8266_USART_Config ();

   /* 复位引脚拉高 */
   macESP8266_RST_HIGH_LEVEL();

   /* CH 使能*/
   macESP8266_CH_ENABLE();
}
```

在 main 函数中调用了 ESP8266_Init 对 ESP8266 进行初始化，以及初始化与电脑连接的串口 USART1，然后用 USART1 打印了关于本例程的提示信息。 最后进入主循环中，程序会不断检测 USART1 和连接到 ESP8266 的 USART3 是否接收到数据， 如果 USART1 接收到电脑串口助手发送过来的数据，则将接收到的数据发送给 ESP8266，ESP8266 接收到有效数据后也会通过 USART3 返回信息。

数据流向如下所示：

串口助手发送 AT 指令：电脑串口助手 -> USART1 接收 -> USART3 发送 -> ESP8266

ESP8266 返回消息：ESP8266 -> USART3 接收 -> USART1 发送 -> 电脑串口助手

main.c

```
int main(void)
{
   /*初始化 USART 配置模式为 115200 8-N-1，中断接收*/
   USART_Config();
   ESP8266_Init();

   printf("欢迎使用野火 STM32 开发板\n\n");
   printf("这是一个 ESP8266AT 指令测试实验\n\n");
   printf("请使用串口调试助手发送\"AT+换行回车\"测试 ESP8266 是否准备好\n\n");
   printf("更多 AT 指令请参考模块资料\n\n");
   printf("以下是 ESP8266 上电初始化打印的信息\n\n");
```

```
  while(1)
  {
        if(strUSART_Fram_Record .InfBit .FramFinishFlag == 1)  //如果接收到了串口调试助手的数据
        {
           strUSART_Fram_Record .Data_RX_BUF[strUSART_Fram_Record .InfBit .FramLength] = '\0';
           Usart_SendString(macESP8266_USARTx ,strUSART_Fram_Record .Data_RX_BUF);      //数据从
串口调试助手转发到ESP8266
           strUSART_Fram_Record .InfBit .FramLength = 0;                                //接收数
据长度置零
           strUSART_Fram_Record .InfBit .FramFinishFlag = 0;                            //接收标
志置零
     }
        if(strEsp8266_Fram_Record .InfBit .FramFinishFlag)                              //如果接
收到了ESP8266 的数据
        {
           strEsp8266_Fram_Record .Data_RX_BUF[strEsp8266_Fram_Record .InfBit .FramLength] =
'\0';
           Usart_SendString(DEBUG_USARTx ,strEsp8266_Fram_Record .Data_RX_BUF);         //数据从
ESP8266 转发到串口调试助手
           strEsp8266_Fram_Record .InfBit .FramLength = 0;                              //接收数
据长度置零
           strEsp8266_Fram_Record.InfBit.FramFinishFlag = 0;                            //接收标
志置零
        }
  }
}
```

下面是中断服务函数，DEBUG_USART_IRQHandler 是单片机与电脑连接的串口中断服务函数，而 macESP8266_USART_INT_FUN 是单片机与 ESP8266 连接的串口中断服务函数。 当串口接收到数据或者接收数据完毕后都会产生中断，我们通过 USART_GetITStatus 函数来判断这两个不同的中断（USART_IT_RXNE 和 USART_IT_IDLE）并进行相应的处理。

stm32f10x_it.c

```
// 单片机与电脑连接的串口中断服务函数
void DEBUG_USART_IRQHandler(void)
{
  uint8_t ucCh;

  /* 判断串口是否产生接收数据中断 */
  if ( USART_GetITStatus ( DEBUG_USARTx, USART_IT_RXNE ) != RESET )
  {
     ucCh  = USART_ReceiveData( DEBUG_USARTx );

     if ( strUSART_Fram_Record .InfBit .FramLength < ( RX_BUF_MAX_LEN - 1 ) )
//预留1 个字节写结束符
           strUSART_Fram_Record .Data_RX_BUF [ strUSART_Fram_Record .InfBit .FramLength ++ ]  =
ucCh;

  }

  /* 判断串口是否产生空闲中断 */
  if ( USART_GetITStatus( DEBUG_USARTx, USART_IT_IDLE ) == SET )                   //数据帧接收完毕
  {
     strUSART_Fram_Record .InfBit .FramFinishFlag = 1;

     ucCh = USART_ReceiveData( DEBUG_USARTx );                                     //由软件序列清除
中断标志位(先读USART_SR，然后读USART_DR)
  }
}
```

```
// 单片机与 ESP8266 连接的串口服务函数
void macESP8266_USART_INT_FUN ( void )
{
   uint8_t ucCh;

   /* 判断串口是否产生接收数据中断 */
   if ( USART_GetITStatus ( macESP8266_USARTx, USART_IT_RXNE ) != RESET )
   {
      ucCh  = USART_ReceiveData( macESP8266_USARTx );

      if ( strEsp8266_Fram_Record .InfBit .FramLength < ( RX_BUF_MAX_LEN - 1 ) )       //预留 1 个
字节写结束符
            strEsp8266_Fram_Record .Data_RX_BUF [ strEsp8266_Fram_Record .InfBit .FramLength ++ ]
= ucCh;
   }

   /* 判断串口是否产生空闲中断 */
   if ( USART_GetITStatus( macESP8266_USARTx, USART_IT_IDLE ) == SET )        //数据帧接收完毕
   {
      strEsp8266_Fram_Record .InfBit .FramFinishFlag = 1;

      ucCh = USART_ReceiveData( macESP8266_USARTx );                          //由软件序列清除中断
标志位(先读 USART_SR，然后读 USART_DR)

   }
}
```

WIFI 透传：

本实验使用 ESP8266 模块实现串口 WiFi 透传功能。本实验中的 ESP8266 工作在 STA 模式，作为 TCP CLIENT。

本实验网络连接模型：开发板 ESP8266<——–>无线路由<———>调试用的电脑。

在 main 函数中需要初始化串口 1，用于打印调试信息，配置 DWT 计数器用于延时函数，初始化 RGB 彩灯，初始化 WiFi 模块使用的接口和外设。

main.c

```
int main ( void )
{
   /* 初始化 */
   USART_Config ();                                                    //初始化串口 1
   CPU_TS_TmrInit();                                                   //初始化 DWT 计数器，用于
延时函数
   LED_GPIO_Config();                                                  //初始化 RGB 彩灯
   ESP8266_Init ();                                                    //初始化 WiFi 模块使用的接
口和外设
   //        DHT11_Init ();                                                  //初始化 DHT11
   //        SysTick_Init ();                                                //配置 SysTick
为 10ms 中断一次，在中断里读取传感器数据


   printf ( "\r\n 野火 WF-ESP8266 WiFi 模块测试例程\r\n" );                    //打印测试例程
提示信息
   printf ( "\r\n 在网络调试助手或者串口调试助手上发送以下命令可以控制板载 RGB 灯\r\n" );     //打印测试
例程提示信息
   printf
( "\r\nLED_RED\r\nLED_GREEN\r\nLED_BLUE\r\nLED_YELLOW\r\nLED_PURPLE\r\nLED_CYAN\r\nLED_WHITE\r\nL
ED_RGBOFF\r\n" );
```

```
    ESP8266_StaTcpClient_Unvarnish_ConfigTest();                    //对 ESP8266 进行配置

    printf ( "\r\n 在网络调试助手或者串口调试助手  发送以下命令控制板载 RGB 灯：\r\n" );    //打印测试例程提示信息
    printf ( "\r\nLED_RED\r\nLED_GREEN\r\nLED_BLUE\r\nLED_YELLOW\r\nLED_PURPLE\r\nLED_CYAN\r\nLED_WHITE\r\nLED_RGBOFF\r\n" );
    printf ( "\r\n 观察 RGB 灯的状态变化\r\n" );

    while ( 1 )
    {

        ESP8266_CheckRecvDataTest(); // ESP8266 检查一次是否接收到了数据

    }

}
```

另外，还要在 ESP8266_StaTcpClient_Unvarnish_ConfigTest 这个函数里面对 ESP8266 进行一些配置，然后才进入主循环执行 ESP8266_CheckRecvDataTest 函数。

我们来看一下 ESP8266_StaTcpClient_Unvarnish_ConfigTest 函数和 ESP8266_CheckRecvDataTest 函数都做了些什么。

首先是 ESP8266_StaTcpClient_Unvarnish_ConfigTest 函数，拉高 CH_PD 引脚，使能模块，对 WF-ESP8266 模块进行 AT 测试启动，选择 WF-ESP8266 模块的工作模式为 STA 模式， WF-ESP8266 模块连接外部 WiFi，WF-ESP8266 模块启动多连接，WF-ESP8266 模块连接外部服务器，配置 WF-ESP8266 模块进入透传发送。设置完毕后就可以开始透传了。

接着进入主循环，在 ESP8266_CheckRecvDataTest 函数里面，会一直检查是否接收到数据，如果接收到了串口调试助手的数据，就转发给 ESP82636，如果接收到了 ESP8266 的数据，就转发给串口调试助手。同时 ESP8266_CheckRecvDataTest 函数里面还会检查 ESP8266 有没有断开连接，若是断连的话会让 ESP8266 退出透传模式，并且重新连接热点和服务器。

bsp_esp8266_test.c

```
/**
* @brief  ESP8266 StaTcpClient Unvarnish 配置测试函数
* @param  无
* @retval 无
*/
void ESP8266_StaTcpClient_Unvarnish_ConfigTest(void)
{
    printf( "\r\n 正在配置 ESP8266 ......\r\n" );
    printf( "\r\n 使能 ESP8266 ......\r\n" );
    macESP8266_CH_ENABLE();
    while( ! ESP8266_AT_Test() );

    printf( "\r\n 正在配置工作模式 STA ......\r\n" );
    while( ! ESP8266_Net_Mode_Choose ( STA ) );

    printf( "\r\n 正在连接 WiFi ......\r\n" );
    while( ! ESP8266_JoinAP ( macUser_ESP8266_ApSsid, macUser_ESP8266_ApPwd ) );

    printf( "\r\n 禁止多连接 ......\r\n" );
    while( ! ESP8266_Enable_MultipleId ( DISABLE ) );

    printf( "\r\n 正在连接 Server ......\r\n" );
    while( !  ESP8266_Link_Server ( enumTCP, macUser_ESP8266_TcpServer_IP,
```

```
macUser_ESP8266_TcpServer_Port, Single_ID_0 ) );

   printf( "\r\n 进入透传发送模式 ......\r\n" );
   while( ! ESP8266_UnvarnishSend () );

   printf( "\r\n 配置 ESP8266 完毕\r\n" );
   printf ( "\r\n 开始透传......\r\n" );
}


/**
* @brief  ESP8266 检查是否接收到了数据，检查连接和掉线重连
* @param  无
* @retval 无
*/
void ESP8266_CheckRecvDataTest(void)
{
   uint8_t ucStatus;
   uint16_t i;

   /* 如果接收到了串口调试助手的数据 */
   if(strUSART_Fram_Record.InfBit.FramFinishFlag == 1)
   {
      for(i = 0;i < strUSART_Fram_Record.InfBit.FramLength; i++)
      {
         USART_SendData( macESP8266_USARTx ,strUSART_Fram_Record.Data_RX_BUF[i]); //转发给
ESP82636
         while(USART_GetFlagStatus(macESP8266_USARTx,USART_FLAG_TC)==RESET){}      //等待发送完成
      }
      strUSART_Fram_Record .InfBit .FramLength = 0;                               //接收数据长度
置零
      strUSART_Fram_Record .InfBit .FramFinishFlag = 0;                           //接收标志置零
      Get_ESP82666_Cmd(strUSART_Fram_Record .Data_RX_BUF);                        //检查一下是不
是点灯命令
   }

   /* 如果接收到了 ESP8266 的数据 */
   if(strEsp8266_Fram_Record.InfBit.FramFinishFlag)
   {
      for(i = 0;i < strEsp8266_Fram_Record .InfBit .FramLength; i++)
      {
         USART_SendData( DEBUG_USARTx ,strEsp8266_Fram_Record .Data_RX_BUF[i]);    //转发给串口调
试助手
         while(USART_GetFlagStatus(DEBUG_USARTx,USART_FLAG_TC)==RESET){}
      }
      strEsp8266_Fram_Record .InfBit .FramLength = 0;                             //接收数据长度置
零
      strEsp8266_Fram_Record.InfBit.FramFinishFlag = 0;                           //接收标志置零
      Get_ESP82666_Cmd(strEsp8266_Fram_Record .Data_RX_BUF);                      //检查一下是不是
点灯命令
   }

   if ( ucTcpClosedFlag )                                                  //检测是否失去连接
   {
      ESP8266_ExitUnvarnishSend ();                                        //退出透传模式

      do ucStatus = ESP8266_Get_LinkStatus ();                             //获取连接状态
      while ( ! ucStatus );

      if ( ucStatus == 4 )                                                 //确认失去连接后重连
      {
         printf ( "\r\n 正在重连热点和服务器 ......\r\n" );

         while ( ! ESP8266_JoinAP ( macUser_ESP8266_ApSsid, macUser_ESP8266_ApPwd ) );
```

```
        while ( !   ESP8266_Link_Server ( enumTCP, macUser_ESP8266_TcpServer_IP,
macUser_ESP8266_TcpServer_Port, Single_ID_0 ) );

        printf ( "\r\n 重连热点和服务器成功\r\n" );

      }

      while ( ! ESP8266_UnvarnishSend () );

   }
}
```