STM32 MCU Development

STM32 单片机开发

-- 串口通信之实时监控

目录

在前面的课程内容中，我们介绍了串口通信的基本原理，及使用它实现 printf()打印输出的调试方法。我们知道串口通信主要是解决了一个字节的发送问题，那如果有多个字节的数据在发送时，我们又该如何规范每个字节的意义？

接下来，我们将使用串口实现如下两个功能：

- 串口实时上报采样温湿度值
- 串口实时接收控制 Led 灯亮灭；

1. JSON 协议介绍

1.1. 软件通信协议

此前我们使用了 printf()函数打印了采样的温湿度值，这样我们可以在 PC 上看到采样的温湿度值。但如果 PC 端要将温湿度值存储保存到数据库中呢？这时候需要从打印的字符串里解析出这两个值来，对于这种没有格式的数据解析比较麻烦。

假设我们传感器采样的温度值为 28.88，而相对湿度值为 36.66。一种比较简单的方式就是以“28.88,36.66”这种格式字符串形式上报，两个值之间以逗号隔开，这样在解析的时候就比较方便了。但这时有个问题，对于接收方而言，它怎么知道前面的 28.88 究竟是温度值，还是相对湿度值？

这时候就需要收、发双方都要遵循这样一个约定，逗号前面的字节表示温度、后面的字节表示相对湿度，那这个约定就叫做协议。如果别人不知道这个协议，那即使它收到了这个数据也不能很顺利地解析出相应的数据了。

对于上面这种以字符串形式来上报的通信协议格式，我们叫做**字符流**协议。很显然字符流的协议在通信时需要发送很多字节，如以“28.88,36.66”形式发送温湿度值需要 11 个字节。另外还有一种方式就是采用字节来进行编码发送，这种通信协议叫做**字节流**协议。

如我们规定一个数据报文的前两个字节表示温度，后两个字节表示相对湿度；其中高字节表示整数位，低字节表示小数位，这样我们就只需要 4 个字节就可以实现数据的收发：0x1C 0x58 0x24 0x42

很显然，字符流的通信格式通俗易懂但通信开销较大(字节数较多)，而字节流的通信格式不容易理解但通信开销较小。在嵌入式物联网开发中，**字符流的格式最常见、用得最多的是 JSON 格式**，而**字节流的格式主要是 TLV 协议（Tag、Length、Value）**，基本上所有的字节流协议格式都是它的变种。

1.2. JSON 格式协议

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式，易于人阅读和编写，同时也易于机器解析和生成。它是 JavaScript 编程语言的一个子集，但采用完全独立于语言的文本格式，使用了类似于 C 语言家族的习惯（包括 C, C++, C#, Java, JavaScript, Perl, Python 等），这些特性使 JSON 成为不同编程语言理想的数据交换语言。

JSON 数据现在是现在嵌入式物联网开发中用得最多的一种协议格式，如阿里云、华为云、腾讯云等物联网云平台等无一例外全部支持 JSON 格式的数据报文收发，此外它在互联网(Java、JavaScript、Android)开发中有着更为非常广泛的应用，所以了解 JSON 协议并熟练掌握它的使用非常有必要。

```
{
    "class":"凌云共享单片机",
    "students": [
    { "name":"农好帅" , "ID":"10000" },
    { "name":"杜子腾" , "ID":"10010" },
    { "name":"韦君梓" , "ID":"10086" }
    ]
}
```

JSON 是一种语法，用来序列化对象、数组、数值、字符串、布尔值和 NULL 值。JSON 数据的本质就是一段字符串而已，只不过有不同意义的分隔符将其分割开来而已.我们看上面的符号，里面有[],{}等符号，其中:

- {}表示一个对象，它是一个无序的键值对(属性值)集合，里面的不同键值对用逗号隔开;
- 冒号：表示一个键值对(key:value)，冒号前面是属性的名称(key)，后面是属性的值(value)。值可以是双引号括起来的字符串（string）、数值(number)、true、false、 null、对象（object）或者数组（array），这些结构可以嵌套。
- []表示的是一个数组，它是值的有序集合，值之间使用逗号分隔。

这样，对我们要上报的采样温度和相对湿度值，我们可以使用下面这种 JSON 格式数据来上报：

```
{"Temperature":"31.05", "Humidity":"48.00"}
```

2. 串口实时上报温湿度

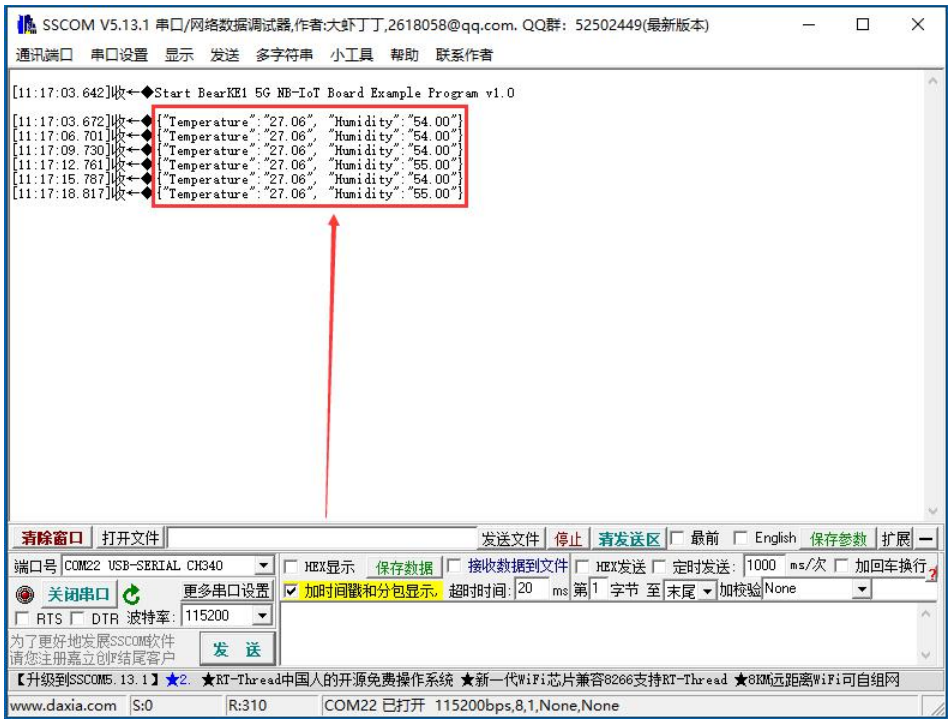
2.1. 源码修改

```
... ...
在文件开始处添加用户头文件 string.h 头文件，后面的 memset()和 snprintf()函数定义在该头文件中 ：
/* USER CODE BEGIN Includes */
#include <string.h>
#include "dht11.h"
/* USER CODE END Includes */

... ...

在这里添加 report_tempRH_json()函数的声明，它定义在 main()函数之后 ：
/* USER CODE BEGIN 0 */
static int report_tempRH_json(void);
/* USER CODE END 0 */

... ...

在 while()循环开始处相应位置添加 DHT11 采样并使用 JSON 协议格式上报温湿度代码：
  while (1)
  {
    if( report_tempRH_json() < 0 )
    {
        printf("ERROR: UART report temperature and relative humidity failure\r\n");
    }
    HAL_Delay(3000);

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }

... ...

在这里添加 report_tempRH_json()函数的定义
/* USER CODE BEGIN 4 */
int report_tempRH_json(void)
{
```

```
    char         buf[128];
    float        temperature, humidity;

    if ( DHT11_SampleData(&temperature, &humidity) < 0 )
    {
         return -1;
    }

    memset(buf, 0, sizeof(buf));
    snprintf(buf, sizeof(buf), "{\"Temperature\":\"%.2f\", \"Humidity\":\"%.2f\"}", temperature,
humidity);

    HAL_UART_Transmit(&huart1 , (uint8_t *)buf, strlen(buf), 0xFFFF);

    return 0;
}
/* USER CODE END 4 */

... ...
```

2.2. 运行测试

重新编译并烧录运行程序，使用串口调试助手监控串口输出，这时候会发现每隔 3s 单片机就会打印 JSON 格式的采样温湿度值。



3. 串口实时接收控制 LED

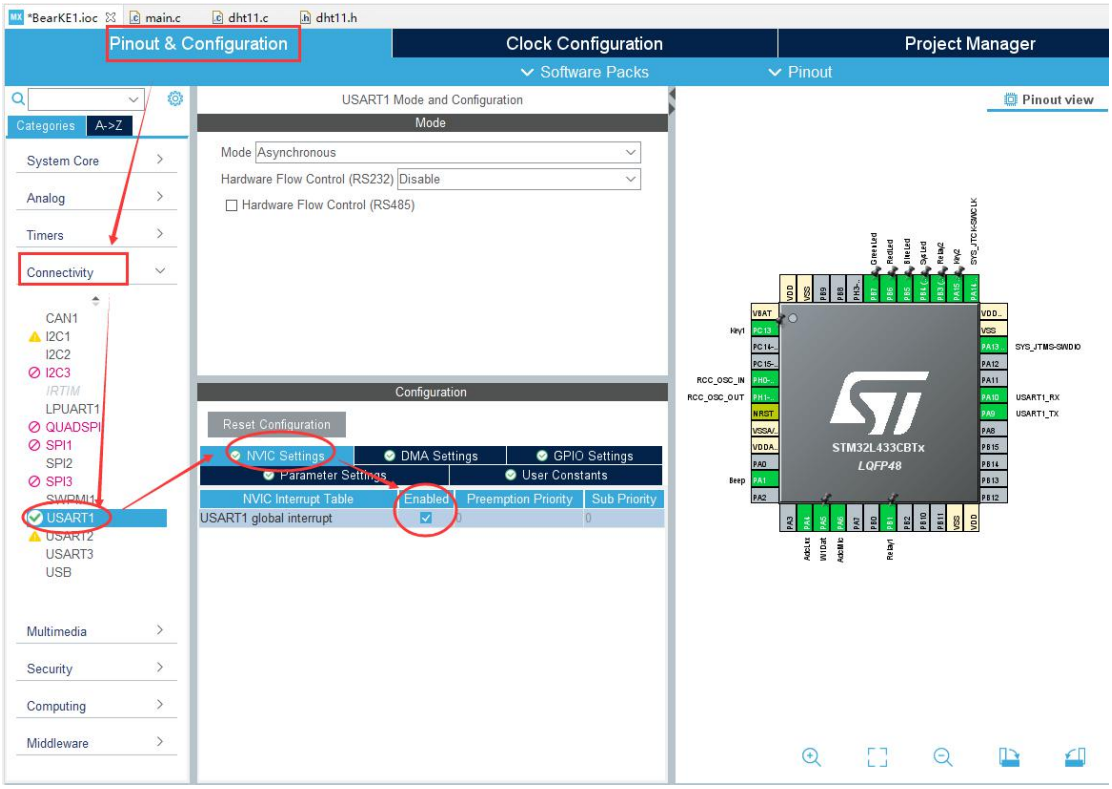
前面讲解了使用标准库函数 snprintf()将采样的温湿度值打包成 JSON 数据报文，并通过串口发送到 PC 上的程序。接下来，我们将实现使用串口接收 PC 端下发的 JSON 格式 Led 控制指令报文，解析并实现 Led 灯的控制。

其 JSON 格式控制数据报文将定义如下：

```
{"RedLed":"on", "GreenLed":"on", "BlueLed":"on"}
{"RedLed":"off", "GreenLed":"off", "BlueLed":"off"}
```

3.1. STM32CubeIDE 配置

配置使能串口 USART1 的中断，使用中断来接收串口上收到的数据。配置好后，按 Ctrl+S 重新生成代码。



3.2. 串口中断接收处理

修改 usart.c 文件，添加串口中断处理函数及接收 buffer：

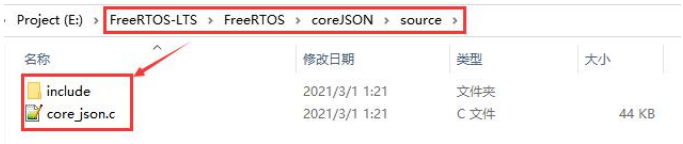
```
... ...
在这里添加串口 usart1 的接收 buffer 相关定义
/* USER CODE BEGIN 0 */
static uint8_t    s_uart1_rxch;
char              g_uart1_rxbuf[256];
uint8_t           g_uart1_bytes;
/* USER CODE END 0 */
... ...

在这个函数里使能 usart1 的接收中断，中断接收的每个字节存储在 s_uart1_rxch 变量中
void MX_USART1_UART_Init(void)
{
  ... ...
  /* USER CODE BEGIN USART1_Init 2 */
  HAL_UART_Receive_IT(&huart1 , &s_uart1_rxch, 1);
  /* USER CODE END USART1_Init 2 */
}
... ...
/* USER CODE BEGIN 1 */
... ...
保留之前添加的 printf()重定向实现代码.

在这里添加串口中断接收的回调函数，还要 g_uart1_rxbuf 没有满，就将中断接收到的 1 字节数据 s_uart1_rxch 存储到 g_uart1_rxbuf 中，并将 g_uart1_bytes 自加。
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);
        }
}
/* USER CODE END 1 */
```

修改 usart.h 文件，添加相关全局变量的声明：

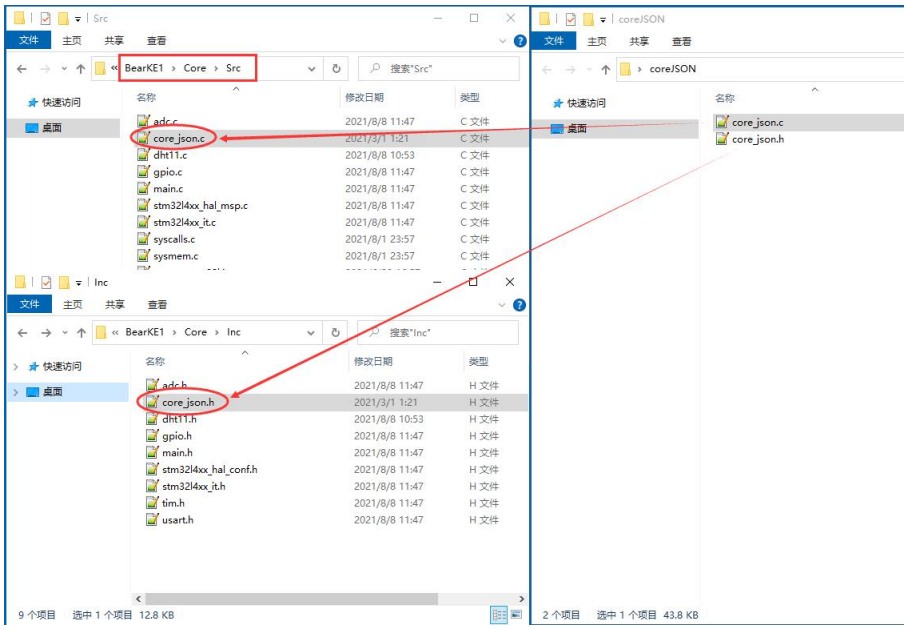
```
... ...
/* USER CODE BEGIN Includes */
#include <stdio.h>
#include <string.h>
/* USER CODE END Includes */
... ...
在这里添加 uart1 接收 buffer 相关变量声明，并添加一个宏 clear_uart1_rxbuf()用来清除接收 buffer 里的数据
/* USER CODE BEGIN Private defines */
extern char              g_uart1_rxbuf[256];
extern uint8_t           g_uart1_bytes;

#define clear_uart1_rxbuf() do { memset(g_uart1_rxbuf, 0, sizeof(g_uart1_rxbuf)); \
                                 g_uart1_bytes=0; } while(0)
/* USER CODE END Private defines */
```

3.3. 添加 JSON 解析库

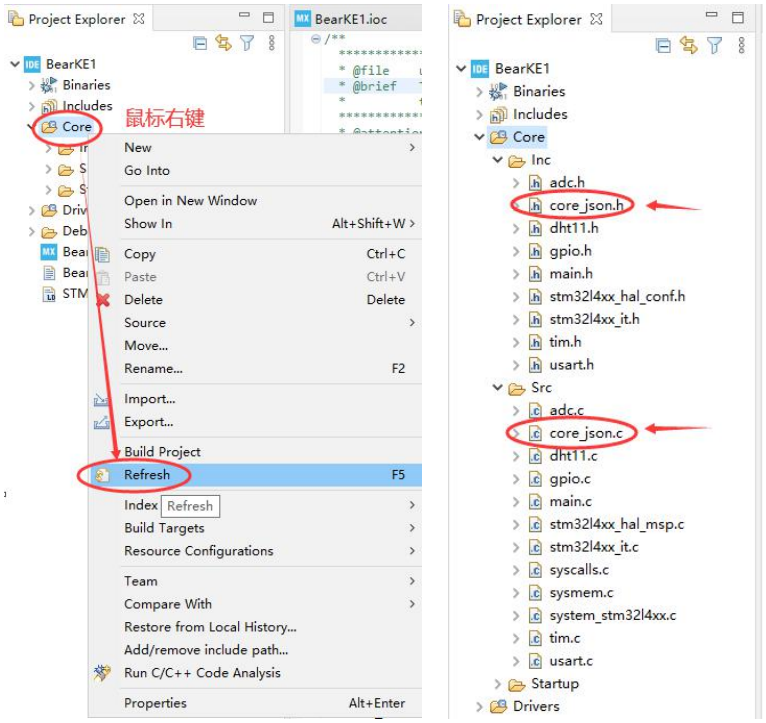
JSON 作为广泛使用的数据通信格式，每种编程语言都会有很多解析库，而 C 语言中经常用到的 JSON 解析库有 cJSON，但因为嵌入式单片机中的 Flash、RAM 有限，这里我们就使用单片机中常用的免费开源操作系统 FreeRTOS 里自带的 coreJSON 库：



该库只有两个 C 文件，我们将其直接拷贝到 STM32CubeIDE 的项目源码下即可。



接下来如下图所示鼠标右键点在”Core”上并刷新，这样刚拷贝过去的 coreJSON 源码文件将会自动加入到项目中去了。



3.4. 修改 Led 驱动函数 API

之前我们封装的 Led 驱动代码，是通过 enum 里的编号来控制的：

```
... ...
enum
{
        SysLed,
        RedLed,
        GreenLed,
        BlueLed,
        LedMax,
};
extern void turn_led(int which, int status);
... ...
```

但现在 JSON 串里用的是字符串名称来控制 LED 灯，这样我们修改一下 Led 灯的驱动接口，让字符串和这个整形索引值建立一种映射关系，从而简化接下来的操作代码：

修改 gpio.h 头文件：

```
... ...

enum
{
        SysLed,
        RedLed,
        GreenLed,
        BlueLed,
        LedMax,
};

enum
{
        Relay1,
        Relay2,
        RelayMax,
};

#define OFF       0
#define ON        1

将 gpio.c 中定义的 gpio_t 结构体定义移到头文件中来，并在它里面添加一个 char *name 的字段，其它 C 文件会用到这个结构体。
typedef struct gpio_s
{
        const char         *name;
        GPIO_TypeDef       *group;
        uint16_t           pin;
} gpio_t;

在这里声明 gpio.c 中定义的继电器和 led 定义
extern gpio_t     relays[RelayMax];
extern gpio_t     leds[LedMax] ;

... ...
```

修改 gpio.c 源文件：

```
... ...
gpio_t    leds[LedMax] =
{
        { "SysLed",   SysLed_GPIO_Port,   SysLed_Pin},
        { "RedLed",   RedLed_GPIO_Port,   RedLed_Pin},
        { "GreenLed", GreenLed_GPIO_Port, GreenLed_Pin},
        { "BlueLed",  BlueLed_GPIO_Port,  BlueLed_Pin},
};

gpio_t    relays[RelayMax] =
{
        { "Relay1", Relay1_GPIO_Port,   Relay1_Pin},
        { "Relay2", Relay2_GPIO_Port,   Relay2_Pin},
};
... ...
```

3.5. 接收数据 JSON 解析

修改 main.c 文件，添加串口接收数据解析代码并控制 LED 灯：

```
  ... ...
在文件开始处添加用户头文件 core_json.h 头文件：
/* USER CODE BEGIN Includes */
#include <string.h>
#include "dht11.h"
#include "core_json.h"
/* USER CODE END Includes */

  ... ...

在这里添加 parser_led_json()函数的声明，它定义在 main()函数之后 ：
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
static int report_tempRH_json(void);
static int parser_led_json(char *json_string, int bytes);
static void proc_uart1_recv(void);
/* USER CODE END 0 */
```

在 while()循环开始处相应位置添加解析 JSON 数据并控制 Led 函数代码：

```
  while (1)
  {
    proc_uart1_recv();

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }

... ...
```

在这里添加 report_tempRH_json()函数的定义

```
/* USER CODE BEGIN 4 */
... ...
```

保留之前添加的 report_tempRH_json()函数定义，在其后再添加 parser_led_json()和 proc_uart1_recv()函数的定义：

```
int parser_led_json(char *json_string, int bytes)
{
   JSONStatus_t            result;
    char                   save;
    char                  *value;
    size_t                 valen;
    int                    i;

    printf("DBUG: Start parser JSON string: %s\r\n", json_string);

          result = JSON_Validate(json_string, bytes);

          /* JSON document is valid so far but incomplete */
          if( JSONPartial == result )
          {
           printf("WARN: JSON document is valid so far but incomplete!\r\n");
           return 0;
          }

          /* JSON document is not valid JSON */
          if( JSONSuccess != result )
          {
           printf("ERROR: JSON document is not valid JSON!\r\n");
           return -1;
          }
```

```
        /* Parser and set LED status */
        for(i=0; i<LedMax; i++)
        {
         result = JSON_Search( json_string, bytes,  leds[i].name, strlen(leds[i].name), &value,
&valen);
         if( JSONSuccess == result )
         {
               save = value[valen];
               value[valen] = '\0';

               if( !strncasecmp(value, "on", 2) )
               {
                    printf("DBUG: turn %s on\r\n", leds[i].name);
                    turn_led(i, ON);
               }
               else if( !strncasecmp(value, "off", 3) )
               {
                    printf("DBUG: turn %s off\r\n", leds[i].name);
                    turn_led(i, OFF);
               }

               value[valen] = save;
         }
        }

        return 1;
}

void proc_uart1_recv(void)
{
          if( g_uart1_bytes > 0 )
          {
              HAL_Delay(200);
              if( 0 != parser_led_json(g_uart1_rxbuf, g_uart1_bytes) )
              {
               clear_uart1_rxbuf();
              }
          }
}
... ...
```

3.6. 运行测试

重新编译并烧录运行程序，PC 上运行串口调试助手监控串口，通过串口发送控制三个 led 全亮的 JSON 格式字符串指令 {"RedLed":"on", "BlueLed":"on", "GreenLed":"on"} ，这时三个 Led 灯将全亮；而发送 {"RedLed":"off", "BlueLed":"off", "GreenLed":"off"} 指令将会让三个 Led 都灭。