第一章 ESP32 的 TCP 连接

1. 学习目的及目标

- 掌握 TCP 原理和工作过程
- 掌握乐鑫 ESP32 的 TCP 的程序设计
- 主要掌握 TCP 作为 Client 的详细过程

2. TCP 科普（来自百度百科）

TCP（Transmission Control Protocol 传输控制协议）是一种面向连接的、可靠的、基于字节流的传输层通信协议，由 IETF 的 RFC 793 定义。在简化的计算机网络 OSI 模型中，它完成第四层传输层所指定的功能，用户数据报协议（UDP）是同一层内， 另一个重要的传输协议。在因特网协议族（Internet protocol suite）中，TCP 层是位于 IP 层之上，应用层之下的中间层。不同主机的应用层之间经常需要可靠的、像管道一样的连接，但是 IP 层不提供这样的流机制，而是提供不可靠的包交换。

应用层向 TCP 层发送用于网间传输的、用 8 位字节表示的数据流，然后 TCP 把数据流分区成适当长度的报文段（通常受该计算机连接的网络的数据链路层的最大传输单元（ MTU）的限制）。之后 TCP 把结果包传给 IP 层，由它来通过网络将包传送给接收端实体的 TCP 层。TCP 为了保证不发生丢包，就给每个包一个序号，同时序号也保证了传送到接收端实体的包的按序接收。然后接收端实体对已成功收到的包发回一个相应的确认（ACK）；如果发送端实体在合理的往返时延（RTT）内未收到确认，那么对应的数据包就被假设为已丢失将会被进行重传。TCP 用一个校验和函数来检验数据是否有错误；在发送和接收时都要计算校验和。

2.1. 连接建立

TCP 是因特网中的传输层协议，使用三次握手协议建立连接。当主动方发出 SYN 连接请求后，等待对方回答 SYN+ACK，并最终对对方的 SYN 执行 ACK 确认。这种建立连接的方法可以防止产生错误的连接，TCP 使用的流量控制协议是可变大小的滑动窗口协议。

TCP 三次握手的过程如下：

- 客户端发送 SYN（SEQ=x）报文给服务器端，进入 SYN_SEND 状态。
- 服务器端收到 SYN 报文，回应一个 SYN （SEQ=y）ACK(ACK=x+1）报文，进入 SYN_RECV 状态。
- 客户端收到服务器端的 SYN 报文，回应一个 ACK(ACK=y+1）报文，进入 Established 状态。

2.2. 连接终止

终止一个连接要经过四次握手，这是由 TCP 的半关闭（half-close）造成的。具体过程如下图所示。 [1]

- 某个应用进程首先调用 close，称该端执行“主动关闭”（active close）。该端的 TCP 于是发送一个 FIN 分节，表示数据发送完毕。
- 接收到这个 FIN 的对端执行 “被动关闭”（passive close），这个 FIN 由 TCP 确认。
- 一段时间后，接收到这个文件结束符的应用进程将调用 close 关闭它的套接字。这导致它的 TCP 也发送一个 FIN。

- 接收这个最终 FIN 的原发送端 TCP（即执行主动关闭的那一端）确认这个 FIN

3. TCP 特点和流程

上面的原理很重要，但毕竟我们只是在 API 之上做应用。只需要了解特点和流程。知道特点可以做方案时候考量可行性，流程就是可行后的实施。

3.1. TCP 特点：

- 面向连接的：发数据前要进行连接。
- 可靠的连接：TCP 连接传送的数据，无差错，不丢失，不重复，且按序到达。
- 点到点：TCP 连接传送的数据，无差错，不丢失，不重复，且按序到达
- 最大长度有限：仅 1500 字节。（http 和 websocket 有了用武之地）

3.2. TCP 流程： （本段来源）

TCP 编程的客户端一般步骤是：

① 创建一个 socket，用函数 socket()；
② 设置 socket 属性，用函数 setsockopt();(可选)
③ 绑定 IP 地址、端口等信息到 socket 上，用函数 bind();* 可选
④ 设置要连接的对方的 IP 地址和端口等属性；
⑤ 连接服务器，用函数 connect()；
⑥ 收发数据，用函数 send()和 recv()，或者 read()和 write();
⑦ 关闭网络连接；

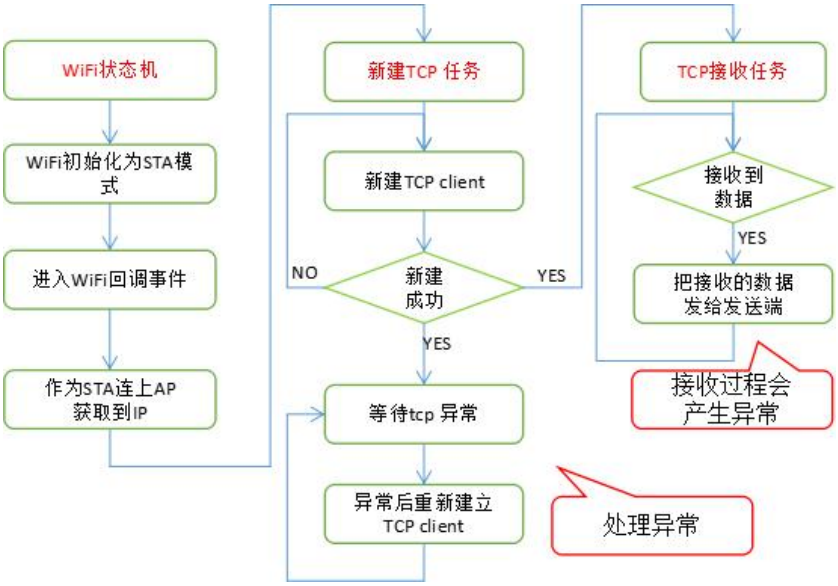
TCP 编程的服务器端一般步骤是：

1. 创建一个 socket，用函数 socket()；
2. 设置 socket 属性，用函数 setsockopt();（可选）
3. 绑定 IP 地址、端口等信息到 socket 上，用函数 bind();
4. 开启监听，用函数 listen()；
5. 接收客户端上来的连接，用函数 accept()；
6. 收发数据，用函数 send()和 recv()，或者 read()和 write();
7. 关闭网络连接； closesocket();
8. 关闭监听；

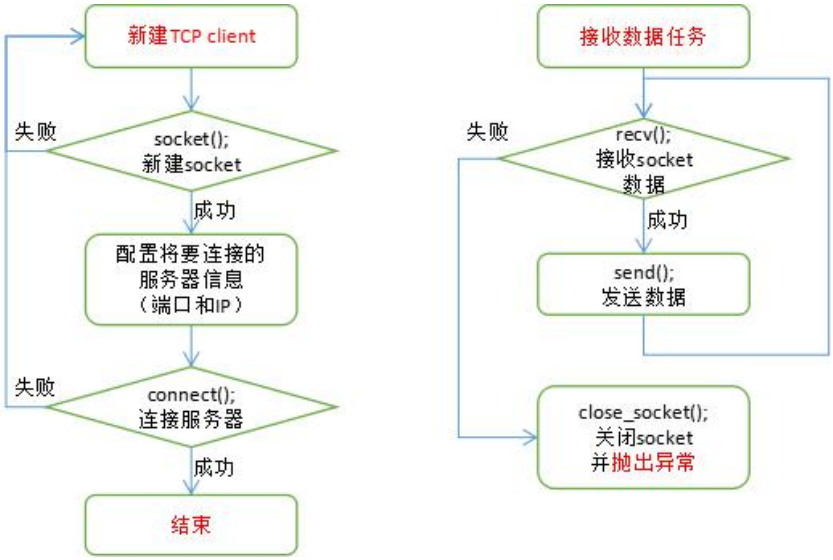
4. TCP 和 UDP（下一章讲）互怼

5. 软件设计

5.1. ESP32 的 TCP Client（Server 类似）主逻辑



5.2. TCP Client 的新建任务和接收任务详细过程逻辑



5.3. ESP32 的 TCP 接口介绍

ESP32 使用的是 LwIP，LwIP 是特别适用于嵌入式设备的小型开源 TCP/IP 协议栈，对内存资源占用很小。ESP-IDF 即是移植了 LwIP 协议栈。学习了解 LwIP，给大家推荐本书，《嵌入式网络那些事:LwIP 协议深度剖析与实战演练》。

我们的这个例程是直接怼的是标准 socket 接口（内部是 LwIP 封装的），没有用 LwIP 的，关于 LwIP 的接口讲解在 Websocket 中讲解，用法都是一样，知道流程后，API 调用即可，处理好异常。流程+接口，打遍无敌手。LwIP 的教程可以参考安富莱、野火的文档。

在 src/include/lwip/socket.h 文件中可以看到下面的宏定义，lwip 的 socket 也提供标准的 socket 接口函数。

```
1  #if LWIP_COMPAT_SOCKETS
2  #define accept(a,b,c)         lwip_accept(a,b,c)
3  #define bind(a,b,c)           lwip_bind(a,b,c)
```

```
#define shutdown(a,b)           lwip_shutdown(a,b)
#define closesocket(s)          lwip_close(s)
#define connect(a,b,c)          lwip_connect(a,b,c)
#define getsockname(a,b,c)      lwip_getsockname(a,b,c)
#define getpeername(a,b,c)      lwip_getpeername(a,b,c)
#define setsockopt(a,b,c,d,e) lwip_setsockopt(a,b,c,d,e)
#define getsockopt(a,b,c,d,e) lwip_getsockopt(a,b,c,d,e)
#define listen(a,b)             lwip_listen(a,b)
#define recv(a,b,c,d)           lwip_recv(a,b,c,d)
#define recvfrom(a,b,c,d,e,f) lwip_recvfrom(a,b,c,d,e,f)
#define send(a,b,c,d)           lwip_send(a,b,c,d)
#define sendto(a,b,c,d,e,f)     lwip_sendto(a,b,c,d,e,f)
#define socket(a,b,c)           lwip_socket(a,b,c)
#define select(a,b,c,d,e)       lwip_select(a,b,c,d,e)
#define ioctlsocket(a,b,c)      lwip_ioctl(a,b,c)

#if LWIP_POSIX_SOCKETS_IO_NAMES
#define read(a,b,c)             lwip_read(a,b,c)
#define write(a,b,c)            lwip_write(a,b,c)
#define close(s)                lwip_close(s)
```

- 新建 socket 函数：socket();
- 连接函数：connect();
- 关闭 socket 函数：close();
- 获取 socket 错误代码：getsocketopt();
- 接收数据函数：recv();
- 发送数据函数：send();
- 绑定函数：bing();
- 监听函数：listen();
- 获取连接函数：accept();

更多更详细接口请参考官方指南。

5.4. ESP32 的 TCP 总结

初始化 wifi 配置后，程序会根据 WIFI 的实时状态，在回调函数中给出状态返回，所以只需要在回调中进行相关操作，STA 开始事件触发 TCP 进行连接，连接上后就可以进行数据的交互。其中对连接的异常情况做出来显得异常重要，TCP 是可靠的，不能玩成地摊货。

5.5. TCP 新建任务编写

只讲 Client，server 看源码。

```
esp_err_t create_tcp_client()
{
    ESP_LOGI(TAG, "will connect gateway ssid : %s port:%d",
             TCP_SERVER_ADRESS, TCP_PORT);
    //新建 socket
    connect_socket = socket(AF_INET, SOCK_STREAM, 0);
```

```
    if (connect_socket < 0)
    {
        //打印报错信息
        show_socket_error_reason("create client", connect_socket);
        //新建失败后，关闭新建的 socket，等待下次新建
        close(connect_socket);
        return ESP_FAIL;
    }
    //配置连接服务器信息：端口+ip
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(TCP_PORT);
    server_addr.sin_addr.s_addr = inet_addr(TCP_SERVER_ADRESS);
    ESP_LOGI(TAG, "connectting server...");
    //连接服务器
    if (connect(connect_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0)
    {
        //打印报错信息
        show_socket_error_reason("client connect", connect_socket);
        ESP_LOGE(TAG, "connect failed!");
        //连接失败后，关闭之前新建的 socket，等待下次新建
        close(connect_socket);
        return ESP_FAIL;
    }
    ESP_LOGI(TAG, "connect success!");
    return ESP_OK;
}
```

5.6. TCP 接收任务代码

```
void recv_data(void *pvParameters)
{
    int len = 0;           //长度
    char databuff[1024];    //缓存
    while (1)
    {
        //清空缓存
        memset(databuff, 0x00, sizeof(databuff));
        //读取接收数据
        len = recv(connect_socket, databuff, sizeof(databuff), 0);
        //异常标记
        g_rxtx_need_restart = false;
        if (len > 0)
        {
            //打印接收到的数组
            ESP_LOGI(TAG, "recvData: %s", databuff);
            //接收数据回发
            send(connect_socket, databuff, strlen(databuff), 0);
```

```
        }
        else
        {
            //打印错误信息
            show_socket_error_reason("recv_data", connect_socket);
            //服务器故障，标记重连
            g_rxtx_need_restart = true;

            break;
        }
    }
    close_socket();
    //标记重连
    g_rxtx_need_restart = true;
    vTaskDelete(NULL);
}
```

5.7. TCP 异常处理

```
static void tcp_connect(void *pvParameters)
{
    。。。。。。。
        while (1)
        {
            vTaskDelay(3000 / portTICK_RATE_MS);
            //重新建立 client，和新建一样一样
            if (g_rxtx_need_restart)
            {
                vTaskDelay(3000 / portTICK_RATE_MS);
                ESP_LOGI(TAG, "reStart create tcp client...");
                //建立 client
                int socket_ret = create_tcp_client();
                if (socket_ret == ESP_FAIL)
                {
                    ESP_LOGE(TAG, "reStart create tcp socket error,stop...");
                    continue;
                }
                else
                {
                    ESP_LOGI(TAG, "reStart create tcp socket succeed...");
                    //重新建立完成，清除标记
                    g_rxtx_need_restart = false;
                    //建立 tcp 接收数据任务
                    xTaskCreate(&recv_data, "recv_data", 4096, NULL, 4, &tx_rx_task);
                }
            }
    }
```

```
    vTaskDelete(NULL);
}
    return ESP_OK;
}
```

6. 测试流程和效果展示

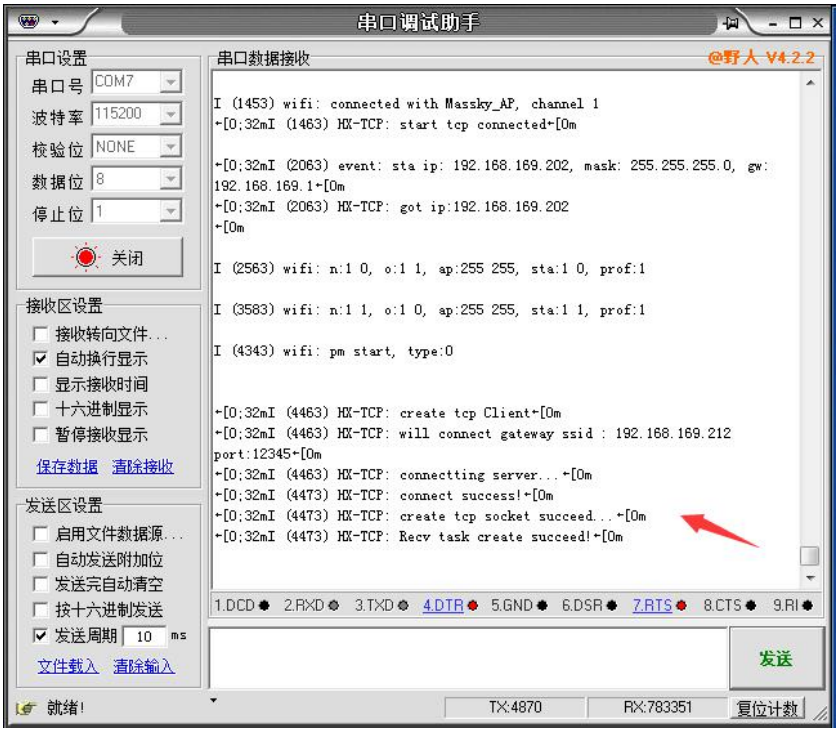
6.1. 测试流程

- **Client 测试**
  - 修改 AP 和 STA 的账号密码
  - #define TCP_SERVER_CLIENT_OPTION FALSE //esp32 作为 client
  - 修改作为 client 连接 server 的 IP(电脑/手机)和 Port
  - 使用手机或者电脑使用助手工具建立 server，让 esp32 自动连接
- **Server 测试**
  - #define TCP_SERVER_CLIENT_OPTION TRUE //esp32 作为 server
  - 修改作为 Server 时监听的 Port
  - 手机或者电脑直连 ESP32 的 AP
  - 使用 TCP 助手工具作为 Client，连接 esp32 的 server

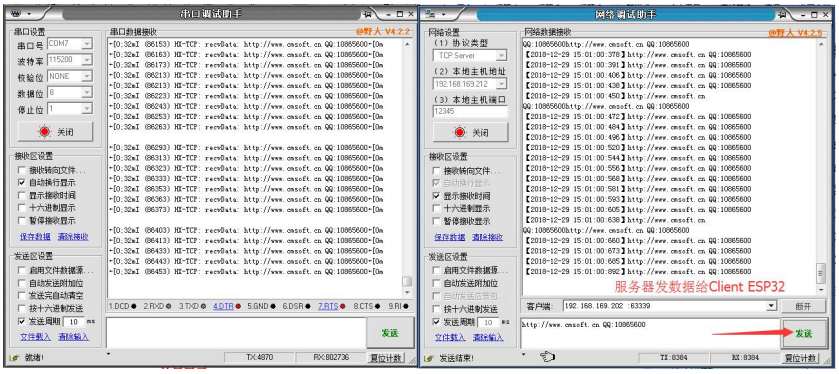
6.2. 效果展示

- **Client 效果展示**

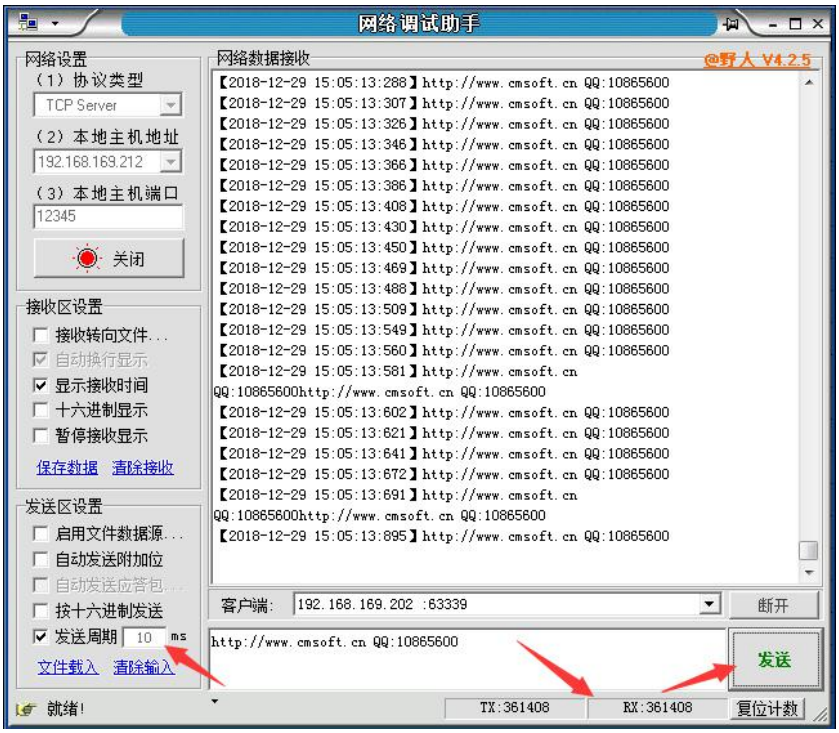
先建服务器，等 ESP32 过来连接。



测试发送数据



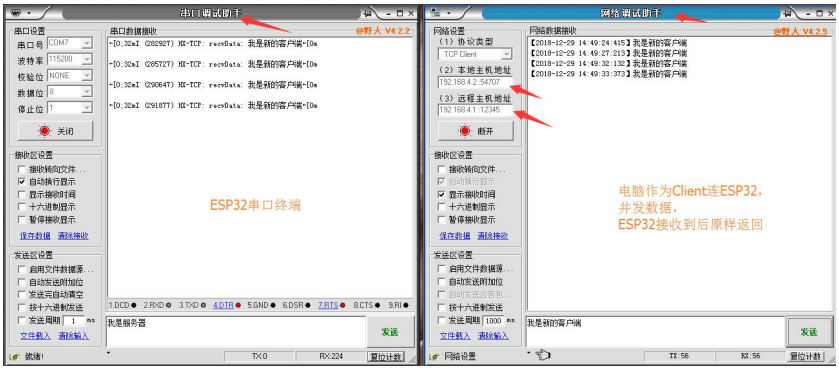
压力小测



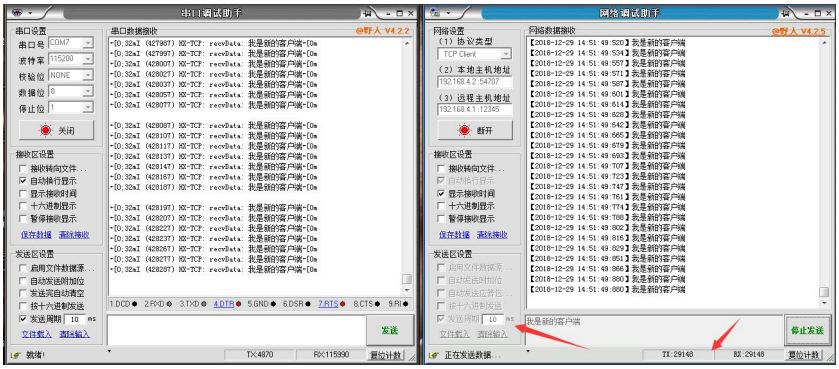
➢ Server 效果展示

连接 ESP32 的 AP

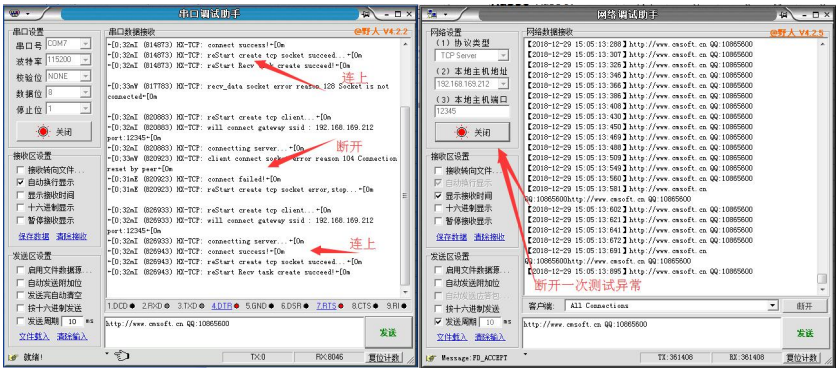
测试发送数据



压力小测



测试异常



7. TCP 总结

- 底层重原理，应用中流程+接口。
- 压力小测不丢包，自己移植要大测产品稳定性。
- 源码地址：https://github.com/xiaolongba/wireless-tech