RT-THREAD AT 组件应用笔记 - 客户端篇

RT-THREAD 文档中心

上海睿赛德电子科技有限公司版权 ©2019

WWW.RT-THREAD.ORG

Friday 28th September, 2018

目录

!!! abstract “摘要” 本应用笔记介绍了 RT-Thread AT 组件的基本知识和 AT 客户端的使用方法，帮助开发者更好地使用 RT-Thread AT 组件。

1 本文的目的和结构

1.1 本文的目的和背景

随着 AT 命令的逐渐普及，越来越多的嵌入式产品上使用了 AT 命令，AT 命令作为主芯片和通讯模块的协议接口，硬件接口一般为串口，这样主控设备可以通过简单的命令和硬件设计完成多种操作。

虽然 AT 命令已经形成了一定的标准化，但是不同的芯片支持的 AT 命令并没有完全统一，这直接提高了用户使用的复杂性。对于 AT 命令的发送和接收以及数据的解析没有统一的处理方式。并且在使用 AT 设备连接网络时，只能通过命令完成简单的设备连接和数据收发功能，很难做到对上层网络应用接口的适配，不利于产品设备的开发。

为了方便用户使用 AT 命令，简单的适配不同的 AT 模块，RT-Thread 提供了 AT 组件用于 AT 设备的连接和数据通讯。**AT 组件的实现包括客户端的和服务器两部分**。对于嵌入式设备而言，更多的情况下设备使用 AT 组件作为客户端连接服务器设备，所以本文将为大家重点介绍 AT 组件中客户端的主要功能、移植方式和实现原理，并介绍在客户端的基础上实现标准 BSD Socket API，使用 AT 命令完成复杂网络通讯。

1.2 本文的结构

本应用笔记将从以下几个方面来介绍 RT-Thread AT 组件：

- AT 组件介绍
- AT Client 配置与使用
- AT Socket 配置与使用

2 问题阐述

本应用笔记将围绕下面几个问题来介绍 RT-Thread AT 组件。

- 什么是 AT 命令？AT 组件的主要功能是什么？
- 什么是 AT Client，它与 AT Server 之间如何进行数据交互？
- 如何使用 AT 命令实现标准 BSD Socket API，支持多种网络软件包和功能？

想要解决这些问题，就要了解 RT-Thread AT 组件基本原理和功能使用方式。下面开始逐步介绍 AT Client 配置方式、移植以及功能使用，使用户快速上手 AT Client 功能。

3 问题的解决

3.1 AT 命令简介

AT 命令是一种应用于 AT 客户端（AT Client）与 AT 服务器（AT Server）间的设备连接与数据通信的方式。其基本结构如下图所示：

图 1: *AT 命令框架*

3.1.1. AT 命令基本概念

- 一般 AT 命令由三个部分组成，分别是：前缀、主体和结束符。其中前缀由字符 AT 构成；主体由命令、参数和可能用到的数据组成；结束符一般为 `<CR><LF>` (`\r\n`)。
- AT 功能的实现需要 AT Server 和 AT Client 两个部分共同完成；
- AT Server 主要用于接收 AT Client 发送的命令，判断接收的命令及参数格式，并下发对应的响应数据，或者主动下发数据；
- AT Client 主要用于发送命令、等待 AT Server 响应，并对 AT Server 响应数据或主动发送的数据进行解析处理，获取相关信息。
- AT Client 和 AT Server 之间支持多种数据通讯的方式（UART、SPI 等），目前最常用的是串口 UART 通讯方式。
- AT Client 接收到的数据类型分成两种：响应数据和 URC 数据。
  - **响应数据**：AT Client 发送命令之后收到的 AT Server 响应状态和信息；
  - **URC 数据**：AT Server 主动发送给 AT Client 的数据，一般出现在一些特殊的情况，比如 WIFI 连接断开、TCP 接收数据等，这些情况往往需要用户做出相应操作。

3.1.2. AT 组件介绍

AT 组件是基于 RT-Thread 系统的 `AT Server` 和 `AT Client` 的实现，组件完成 AT 命令的发送、命令格式及参数判断、命令的响应、响应数据的接收、响应数据的解析、URC 数据处理等整个 AT 命令数据交互流程。

通过 AT 组件，设备可以作为 AT Client 使用串口连接其他设备完成数据的发送、接收与解析，可以作为 AT Server 让其他设备甚至 PC 设备连接完成数据的响应，也可以在本地 shell 启动 CLI 模式使设备同时支持 AT Server 和 AT Client 功能，多用于设备开发调试。

3.2 AT Client 功能

本文将基于正点原子 STM32F4 探索者开发板和乐鑫 ESP8266 开发板，给出了 AT 组件中 AT Client 功能的配置、移植和使用方式。

下图为本文使用的两个开发板的底板图，开发者可以使用 ESP8266 开发板或模组，若缺少正点原子 STM32F4 探索者开发板可使用其他带额外串口的开发板代替，需确保开发板正常运行 RT-Thread 系统且串口使用正常即可：

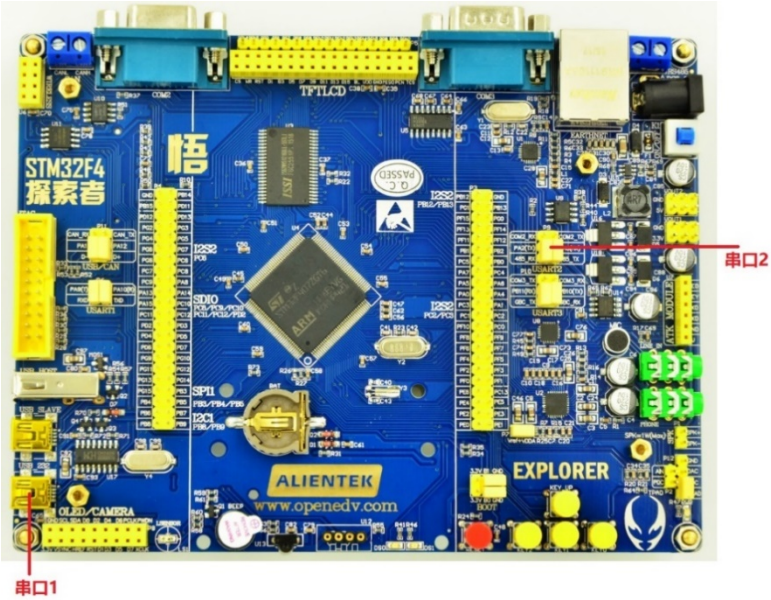

图 2: *STM32F4 底板图*

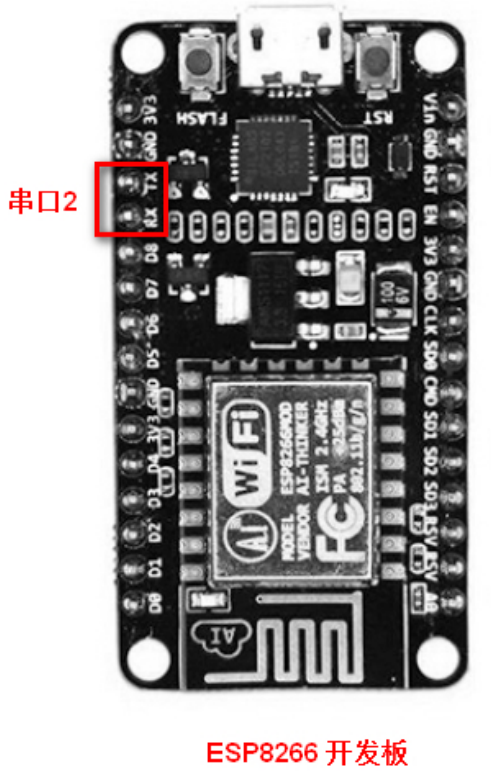

图 3: *ESP8266 底板图*

AT 组件中 AT Client 主要完成 AT 命令的发送和响应数据的接收与解析。这里我们使用正点原子 STM32F4 探索者开发板串口 2 作为 AT Client 连接 ESP8266 开发板的串口 2，ESP8266 开发板的串口 2 作为 AT Server，完成 AT Client 数据收发和解析的功能，下面就具体给出配置、移植和使用方式的介绍。

3.2.1. AT Client 配置

1. 下载 RT-Thread 源码

2. 下载 env 工具

3. 开启 env 工具，进入 `rt-thread\bsp\stm32f4xx-HAL` 目录，在 env 命令行输入 menuconfig 进入配置界面配置工程。

- 配置串口支持：勾选 Using UART1、Using UART2 选项，选择芯片型号为 STM32F407ZG，外部时钟源为 8MHz。

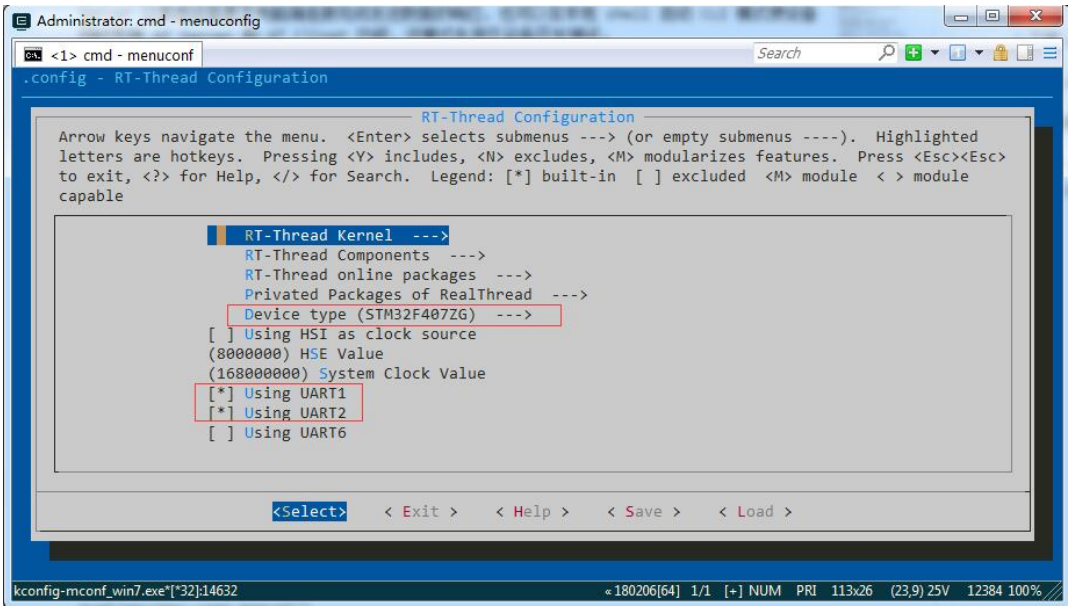

图 4: *配置串口支持*

- 配置 shell 设备：RT-Thread Kernel —> Kernel Device Object —> 修改 the device name for console 为 `uart1`，配置 shell 默认设备为串口 1。

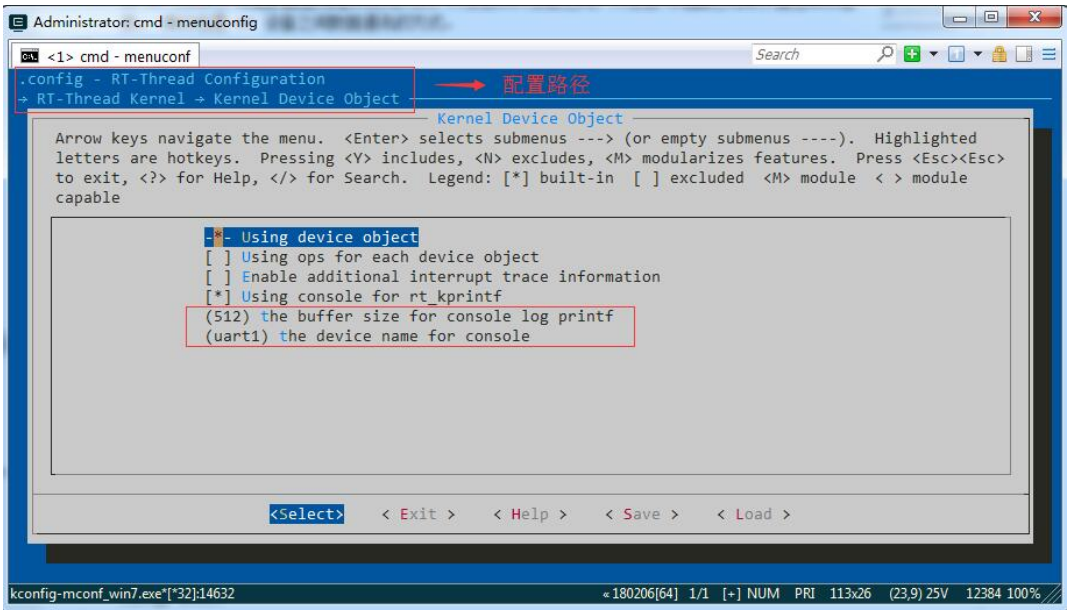

图 5: *配置 shell 设备*

- 开启 AT Client 功能：RT-Thread Components —> Network —> AT commands –> 开启 AT DEBUG，开启 AT Client 支持，目前 AT Client 支持多连接功能，后面需要手动初始化 AT Client。

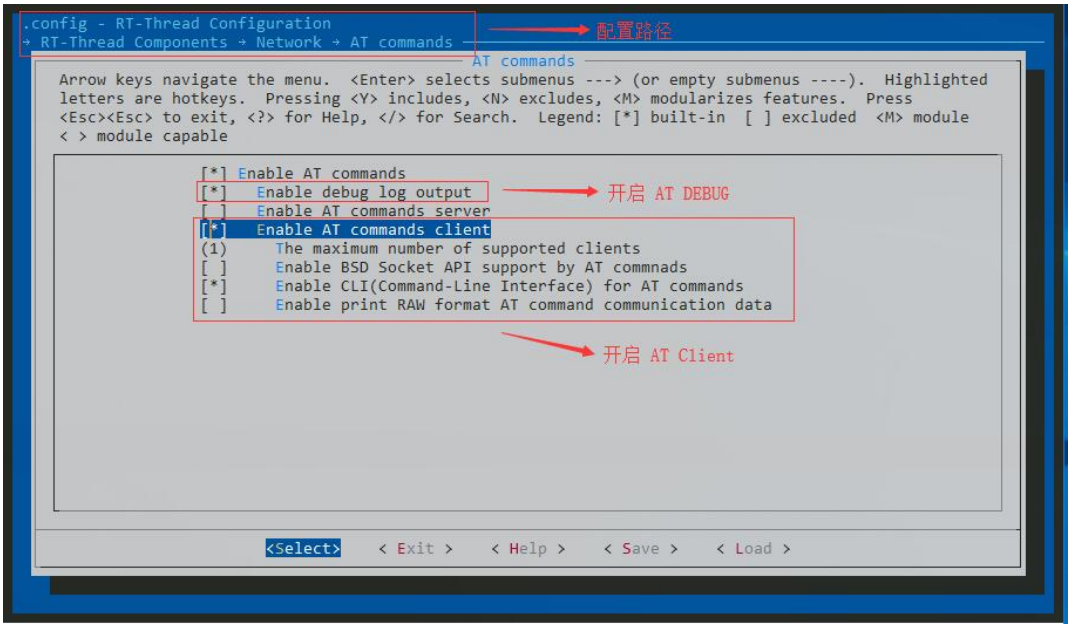

图 6: *AT Client 配置*

AT Client 配置选项介绍如下：

- Enable debug log output：配置开启调试日志；
- Enable AT commands client：配置开启 AT 客户端；
- The muxinum number of supported clients: 配置最大同时支持的客户端数量，该例程使用单客户端连接，配置为 1 即可。
- Enable BSD Socket API support by AT commands: 配置开启 BSD Socket API 支持，该例程没有使用可不开启。
- Enable CLI(Command-Line Interface) for AT commands: 配置开启 AT 命令行交互模式。
- Enable print RAW format AT command communication data：配置开启收发数据实时打印功能。

4. 配置完成，保存并退出配置选项，输入命令 scons –target=mdk5 生成 keil 工程；

3.2.2. AT Client 移植

AT Client 的移植主要是对 **URC 数据（服务器主动下发数据）**的处理，实现获取不同的 URC 数据时执行相应的操作函数的功能。**如果不考虑 AT Client URC 数据处理可忽略移植部分。**

对于 URC 数据，AT 组件中已经提供完善的 URC 数据判断和处理方式，下面给出 AT Client URC 数据处理流程，开发者可以定义修改对应设备的 URC 处理完成 AT Client 的移植。

注：“OK” 和 “ERROR” 为正常命令响应结果判断字符串，切勿作为 URC 数据设置到 URC 列表中。

```
/* URC 数据相关结构体定义 */
struct at_urc
{
    const char *cmd_prefix;                                 //URC 数据前缀
    const char *cmd_suffix;                                 //URC 数据后缀
    void (*func)(const char *data, rt_size_t size);         //URC 数据执行函
        数
};

static void urc_func(const char *data, rt_size_t size)
{
    /* 自定义 URC 数据处理方式 */
    LOG_D("URC data : %.*s", size, data);
}

static struct at_urc urc_table[] = {
    {"ready",            "\r\n",    urc_func},
    {"WIFI CONNECTED",   "\r\n",    urc_func},
    {"WIFI DISCONNECT",  "\r\n",    urc_func},
};

int at_client_obj_init(void)
{
    /* 初始化 AT Client */
    at_client_init("uart2", 512);

    /* 添加多个 URC 结构体至 URC 列表中，当接收到同时匹配 URC 前缀和后缀
        的数据，执行 URC 函数 */
    at_set_urc_table(urc_table, sizeof(urc_table) / sizeof(urc_table[0]))
        ;

    return RT_EOK;
}
```

3.2.3. AT Client 示例添加

下载 AT Client 示例代码，添加到打开的 keil 工程中，如下图所示：

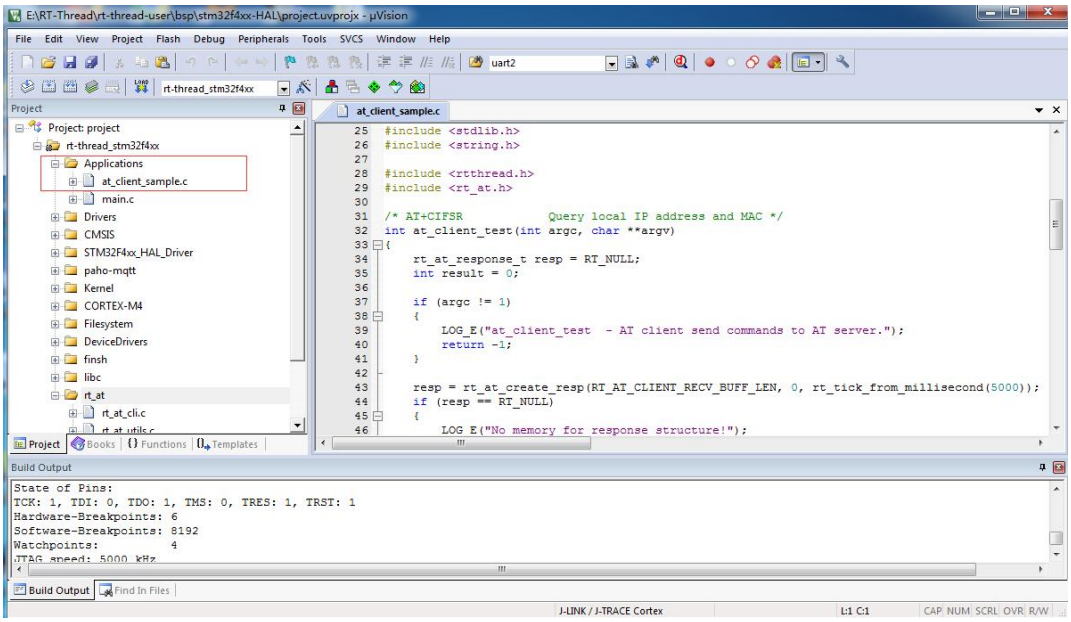

图 7: *AT Client 示例添加*

示例添加完成，就可以编译、下载程序到开发板，之后打开 PC 上串口工具，这里使用 xshell 工具，选择正确的串口（配置串口参数为 115200-8-1-N、无流控），然后按下复位后就可以在串口 1 连接的终端上看到 RT-Thread 系统启动日志。

系统初始化成功之后，在 shell 中执行 `at_client_init uart2` 命令，这里的 `uart2` 为开发板中作为 AT client 的设备名。然后可以看到 AT Client 的初始化日志，说明 AT Client 功能配置启动成功，如下图所示：

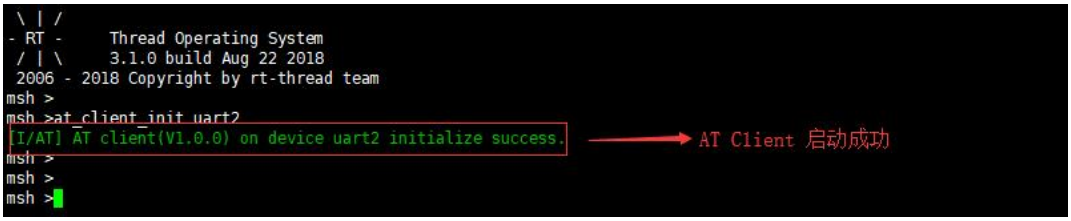

图 8: *AT Client 成功启动*

3.2.4. AT Client 使用

1.AT Client 模式

该模式下正点原子 STM32F4 探索者开发板串口 2 作为 AT Client，ESP8266 开发板作为 AT Server，进行数据交互模式，在本地 shell 中输入 `at_client_test` 命令，该 shell 命令用于发送 AT 命令到服务器，并且接收和解析服务器响应数据，如下图所示过程：

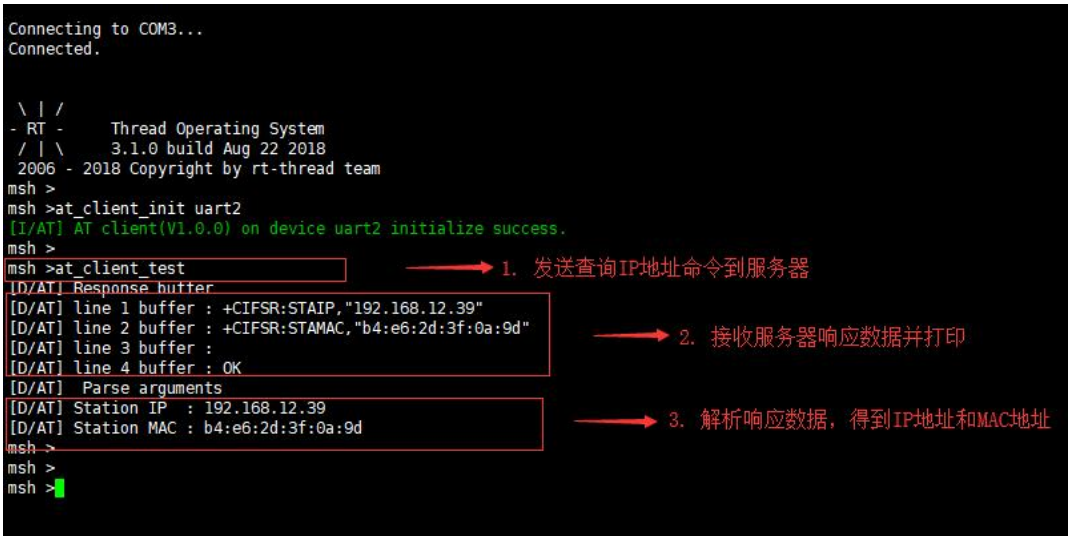

图 9: *AT Client 运行示例*

2.AT Client CLI 模式

AT Client CLI 功能可以转发本地 shell 输入的数据到设备连接的 AT Server 串口设备上，并在本地 shell 上实时显示 AT Client 串口接收到的数据。在本地 shell 中执行 `at client` 命令进入 AT Client CLI 模式即可进行数据的收发。通过 AT Client CLI 模式，用户可以很方便的完成与 AT Server 的连接与调试，极大的提高开发效率。

下图演示了 AT Client CLI 功能的使用和退出：

图 10: *AT Client CLI 运行示例*

3.2.5. AT Client 使用流程

本文使用的 AT Client 示例代码演示了 AT Client 的整个使用流程，示例代码完成 STM32F4 设备 AT 命令的发送并接收和解析 ESP8266 设备的响应数据。代码的使用和平

台有关，开发者可以根据自己使用的平台修改示例代码并运行，主要修改命令的名称和解析的方式。下面通过示例代码介绍一下 AT Client 的具体使用流程：

```
#include <stdlib.h>
#include <string.h>
#include <rtthread.h>
#include <at.h>

/* AT+CIFSR            Query local IP address and MAC */
int at_client_test(int argc, char **argv)
{
    at_response_t resp = RT_NULL;
    int result = 0;

    if (argc != 1)
    {
        LOG_E("at_client_test  - AT client send commands to AT server.");
        return -1;
    }

    /* 创建响应结构体，设置最大支持响应数据长度为 256 字节
    （最大响应长度用户根据实际需求自定义），响应数据行数无限制，超时时间
       为 5 秒 */
    resp = at_create_resp(256, 0, rt_tick_from_millisecond(5000));
    if (resp == RT_NULL)
    {
        LOG_E("No memory for response structure!");
        return -2;
    }

    /* 关闭回显功能 */
    at_exec_cmd(resp, "ATE0");

    /* AT  Client 发送查询 IP 地址命令并接收 AT Server 响应 */
    /* 响应数据及信息存放在 resp 结构体中 */
    result = at_exec_cmd(resp, "AT+CIFSR");
    if (result != RT_EOK)
    {
        LOG_E("AT client send commands failed or return response error!")
            ;
        goto __exit;
    }

    /* 按行数循环打印接收到的响应数据 */
    {
```

```
        const char *line_buffer = RT_NULL;

        LOG_D("Response buffer");
        for(rt_size_t line_num = 1; line_num <= resp->line_counts;
            line_num++)
        {
            if((line_buffer = at_resp_get_line(resp, line_num)) !=
                RT_NULL)
            {
                LOG_D("line %d buffer : %s", line_num, line_buffer);
            }
            else
            {
                LOG_E("Parse line buffer error!");
            }
        }
    }
    /* 按自定义表达式（sscanf 解析方式）解析数据，得到对应数据 */
    {
        char resp_arg[AT_CMD_MAX_LEN] = { 0 };
        /* 自定义数据解析表达式 ，用于解析两双引号之间字符串信息 */
        const char * resp_expr = "%*[^\"]\"%[^\"]\"";

        LOG_D(" Parse arguments");
        /* 解析响应数据中第一行数据，得到对应 IP 地址 */
        if (at_resp_parse_line_args(resp, 1, resp_expr, resp_arg) == 1)
        {
            LOG_D("Station IP  : %s", resp_arg);
            memset(resp_arg, 0x00, AT_CMD_MAX_LEN);
        }
        else
        {
            LOG_E("Parse error, current line buff : %s", at_resp_get_line
                (resp, 4));
        }

        /* 解析响应数据中第二行数据，得到对应 MAC 地址 */
        if (at_resp_parse_line_args(resp, 2, resp_expr, resp_arg) == 1)
        {
            LOG_D("Station MAC : %s", resp_arg);
        }
        else
        {
            LOG_E("Parse error, current line buff : %s", at_resp_get_line
                (resp, 5));
            goto __exit;
```

```
        }
    }
__exit:
    if(resp)
    {
        /* 删除 resp 结构体 */
        at_delete_resp(resp);
    }

    return result;
}
/* 设置当前 AT 客户端最大支持的一次接收数据的长度 */
#define AT_CLIENT_RECV_BUFF_LEN         512
int at_client_test_init(int argc, char **argv)
{
    if (argc != 2)
    {
        rt_kprintf("at_client_init <dev_name>   -- AT client initialize.\
            n");
        return -RT_ERROR;
    }

    at_client_init(argv[1], AT_CLIENT_RECV_BUFF_LEN);

    return RT_EOK;
}
#ifdef FINSH_USING_MSH
#include <finsh.h>
/* 添加 AT Client 测试命令到 shell  */
MSH_CMD_EXPORT(at_client_test, AT client send cmd and get response);
/* 添加 AT Client 初始化命令到 shell  */
MSH_CMD_EXPORT_ALIAS(at_client_test_init, at_client_init, initialize AT
    client);
#endif
```

- 整个示例为单客户端示例，可以直接使用单客户端模式 API。
- AT Client 使用流程大致如下：at_create_resp() 创建响应结构体 —> at_exec_cmd() 发送命令并接收响应 —> at_resp_get_line()/at_resp_parse_line_args() 打印或解析响应数据 —> at_delete_resp() 删除响应结构体。
- at_exec_cmd() 函数完成对传入 AT 命令的发送和响应数据的接收，响应数据以按行的形式存放与结构体中，便于数据按行打印或及解析。
- 打印或解析数据时对于不同的命令的响应数据有不同的数据解析方式，需要自定义数据解析的表达式，这要求开发者提前知道发送命令的具体响应结构，可以通过查看设

备 AT 命令手册了解。

3.3 AT Socket 功能（进阶）

为了方便开发者使用 AT 组件进行网络相关操作，降低 RT-Thread 系统对单独协议栈网络连接的依赖，RT-Thread 系统在 AT 组件和 SAL 组件的基础上推出了 AT Socket 功能。

AT Socket 功能是建立在 AT Client 功能基础上，主要作用是使用 AT 命令完成设备连接网络并进行数据通讯，AT Socket 功能提供的网络接口支持标准的 BSD Socket API，并通过 SAL 组件的接口抽象，实现接口统一化。

AT Socket 功能使设备无需进行网络连接，直接使用串口完成设备联网功能，简化了设备开发的软硬件设计，方便开发者开发。此外，不同于传统的软件网络协议栈，AT Socket 网络功能的运行主要是在串口连接的 AT Server 设备上完成，根据不同的 AT Server 设备，可同时支持 5-6 个 socket，这样极大了降低了 AT Client 设备上 MCU 资源占用，提高 MCU 工作效率，确保数据通讯的质量和硬件的资源的合理分配。AT Socket 功能目前占用最少资源体积约为 20K ROM 、3K RAM（支持 5 个 Socket）。

AT Socket 功能对于不同的 AT 设备需要完成移植适配过程，目前已经完成多种设备的适配，包括：ESP8266、M26 、MC20 等，适配的方式通过 at_device 软件包 给出。下面主要通过 ESP8266 设备，对 AT Socket 功能的配置和使用进行介绍。

3.3.1. AT Socket 配置

AT Socket 功能的使用依赖于如下几个组件：

- **AT 组件**：AT Socket 功能基于 AT Client 功能的实现；
- **SAL 组件**：SAL 组件主要是 AT Socket 接口的抽象，实现标准 BSD Socket API；
- **at_device 软件包**：针对不同设备的 AT Socket 移植和示例文件，以软件包的形式给出；

下面主要介绍 env 中配置 AT Socket 功能的整个流程：

1. 开启 env 工具，进入 `rt-thread\bsp\stm32f4xx-HAL` 目录，在 env 命令行输入 menuconfig 进入配置界面配置工程。

2. 开启 AT Socket：RT-Thread Components —> Network —> AT commands —> Enable BSD Socket API support by AT commnads 开启 AT Socket 支持。

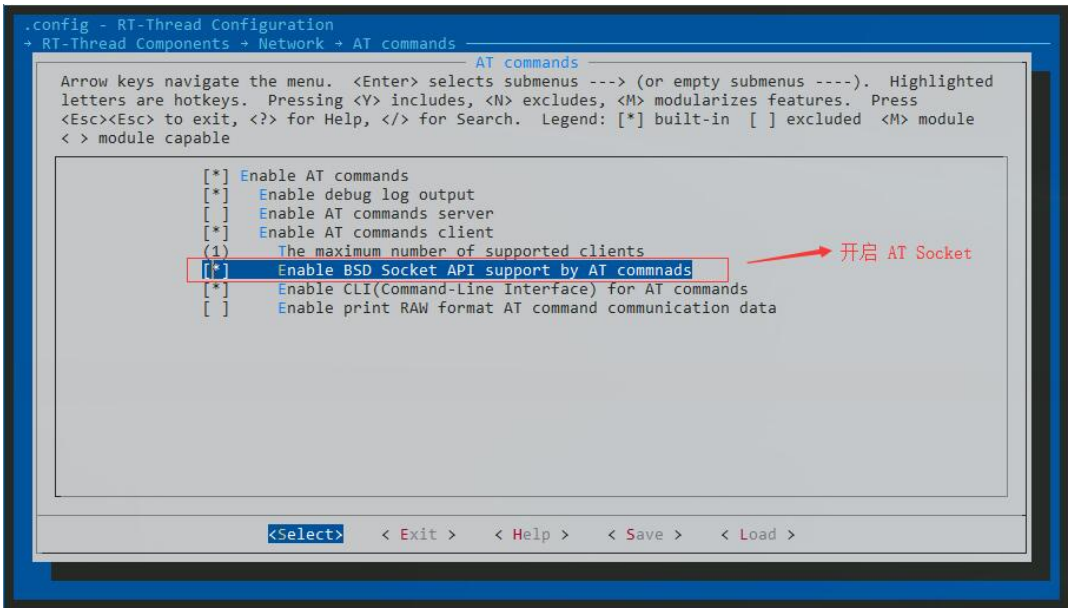

图 11: *配置开启 AT Socket*

3.AT Socket 功能配置成功后，需要开启 at_device 软件包，at_device 软件包需要配置使用的 AT 模块型号和 AT Client 设备名称，确保正确运行：RT-Thread online packages —> IoT - internet of things —> 开启 AT DEVICE 软件包支持，配置 AT Socket device modules 为 ESP8266 设备，配置 AT Client 设备名称和最大支持的接收数据长度，配置 wifi ssid 和 wifi password 用于设备联网，版本使用最新 `laster` 版本。

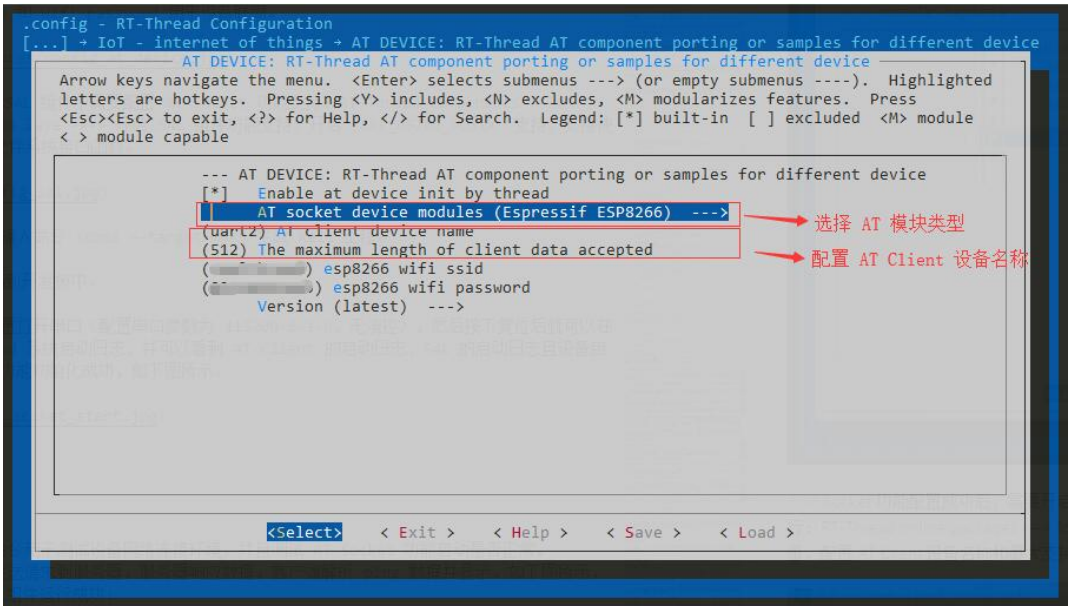

图 12: *配置 at device 软件包*

4. 之后需要开启 SAL 组件支持，在 SAL 组件中需要配置 AT Socket 功能支持：RT-Thread Components —> Network —> Socket abstraction laye —> 开启 SAL 组件功能支

持，开启 SAL_USING_POSIX 支持，支持使用 read/write、poll/select 等文件系统接口函数。

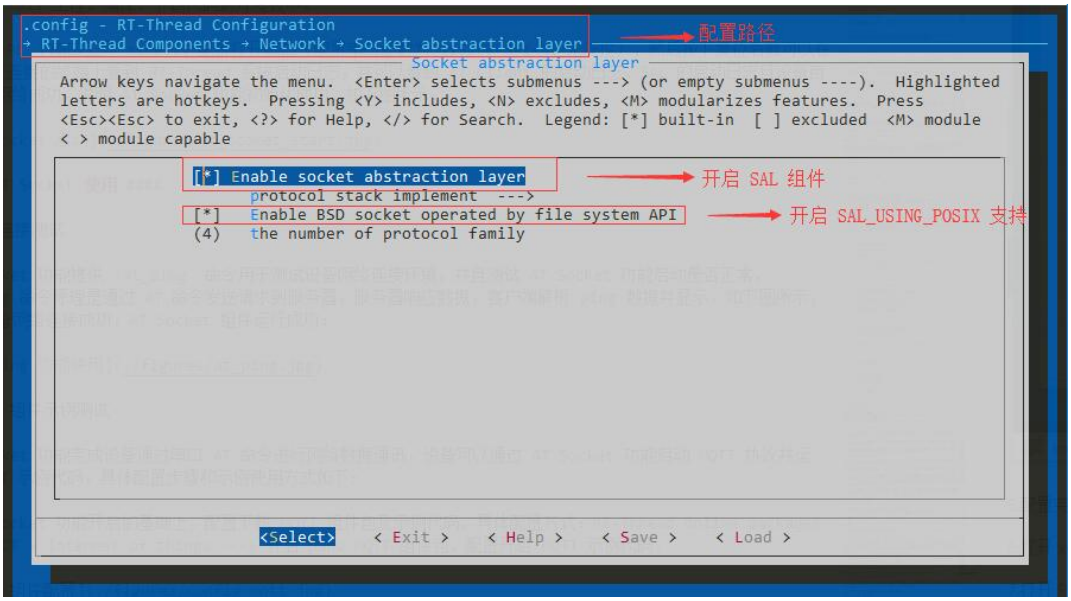

图 13: *开启 SAL 支持*

5. 配置完成，保存并退出配置选项，输入命令 scons –target=mdk5 生成 keil 工程。

6. 打开 keil 工程，编译、下载代码到开发板中。

7. 打开 PC 上串口工具 xshell，配置打开串口（配置串口参数为 115200-8-1-N、无流控），然后按下复位后就可以在串口 1 连接的终端上看到 RT-Thread 系统启动日志，并可以看到 AT Client 的启动日志、SAL 的启动日志且设备自动连接网络成功，说明 AT Socket 功能初始化成功，如下图所示。

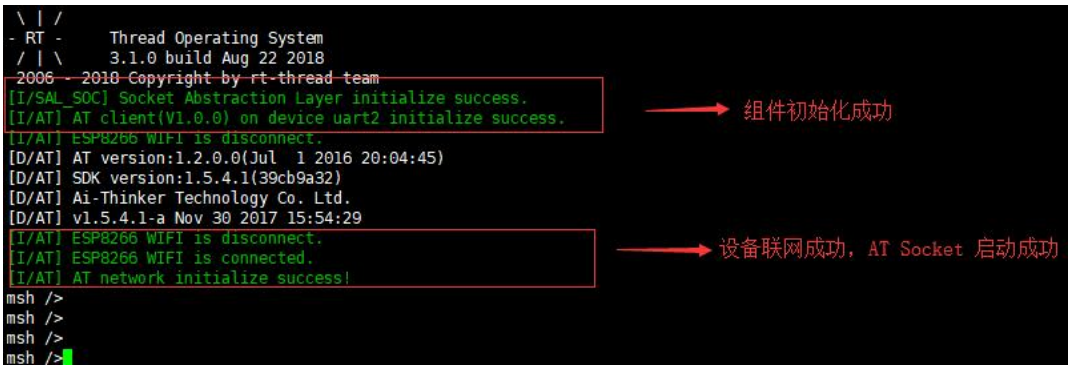

图 14: *AT Socket 启动*

3.3.2. AT Socket 使用

1. 网络连接测试

AT Socket 功能提供 at_ping 命令用于测试设备网络连接环境，并且测试 AT Socket 功能启动是否正常，at_ping 命令原理是通过 AT 命令发送请求到服务器，服务器响应数据，

客户端解析 ping 数据并显示。如下图所示，命令设备网络连接成功，AT Socket 组件运行成功：

```
msh />
msh />
msh />
msh />at_ping baidu.com
32 bytes from baidu.com icmp_seq=1 time=62 ms
32 bytes from baidu.com icmp_seq=2 time=36 ms
32 bytes from baidu.com icmp_seq=3 time=59 ms
32 bytes from baidu.com icmp_seq=4 time=33 ms
msh />
msh />
msh />
```

图 15: *at_ping 功能使用*

2.MQTT 组件示例测试

AT Socket 功能完成设备通过串口 AT 命令进行网络数据通讯，设备可以通过 AT Socket 功能启动 MQTT 协议并运行 MQTT 示例代码，具体配置步骤和示例使用方式如下：

- AT Socket 功能开启的基础上，配置下载 MQTT 组件包及示例代码，具体配置方式：RT-Thread online packages —> IOT - internet of things —> 开启 paho MQTT 组件包，配置开启 MQTT 示例代码。

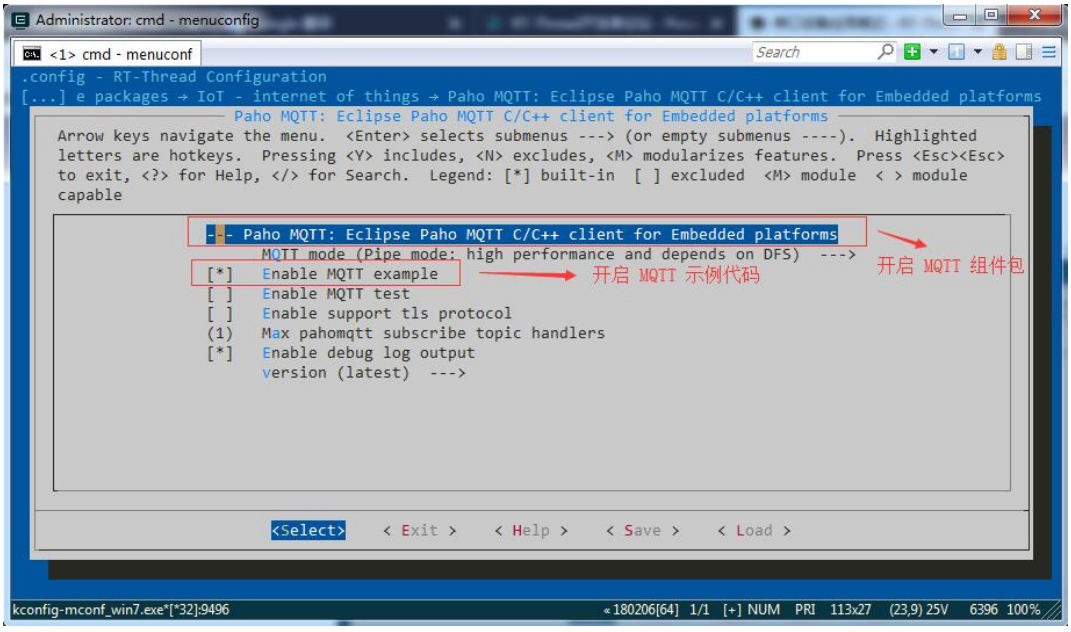

图 16: *MQTT 组件配置*

- 配置完成，保存并退出配置选项，scons 重新生成工程，编译下载代码到开发板中。
- 打开串口工具，系统启动成功，输入 `mq_start` 命令启动 MQTT 协议，启动完成之后输入 `mq_pub mqtt_test_data` 命令，用于向固定的 MQTT Topic 发送数据，同时 MQTT 服务器会立刻向该 Topic 发送同样数据，MQTT 示例测试完成，如下图所示：

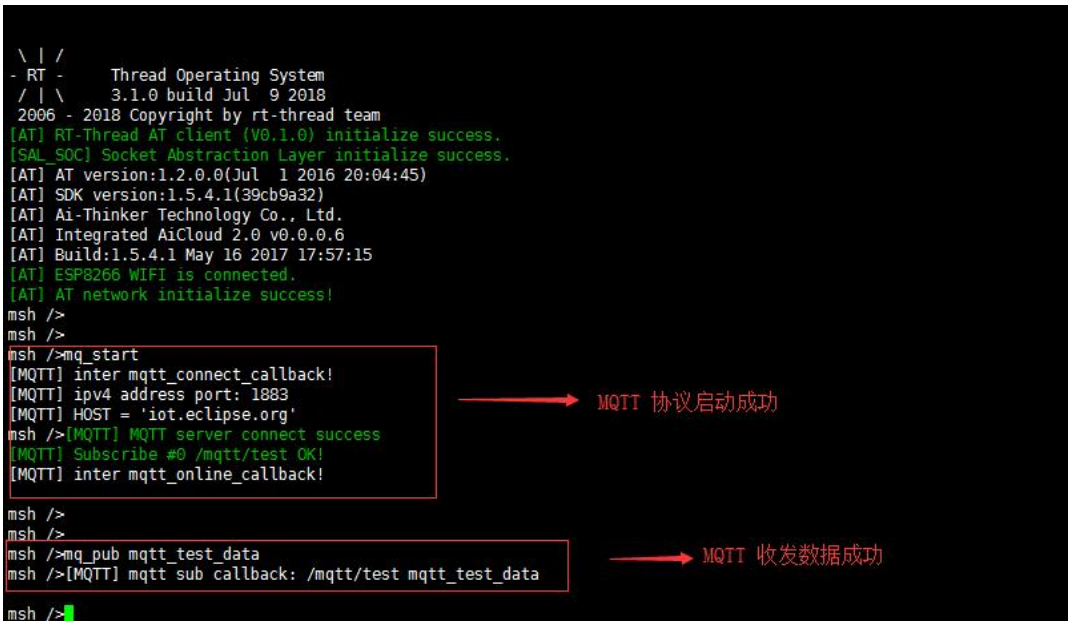

图 17: *MQTT 示例运行*

上述展示了正点原子 STM32F4 设备在未连接网络的情况下使用 AT Socket 功能运行 MQTT 网络示例，实现了 AT Socket 网络数据收发的功能，目前 AT Socket 功能只支持设备作为网络客户端连接服务器，这也符合嵌入式设备多用于客户端设备的特性。AT Socket 目前已经支持多种网络相关组软件包和功能，如下所示：

- MQTT 软件包
- webclient 软件包
- mbedtls 软件包
- onenet 软件包
- NTP 时间查询功能
- iperf 网络测试功能
- at_ping 网络测试功能

更多 AT 组件以及 AT Socket 功能的使用请参考《RT-Thread 编程手册》中 **AT 组件介绍**章节。

4 常见问题

1. 开启 AT 命令收发数据实时打印功能，shell 上日志显示错误怎么办？

- 提高 shell 对应串口设备波特率为 921600，提高串口打印速度，防止数据量过大时打印显示错误。

2. AT Socket 功能启动时，编译提示 `The AT socket device is not selected, please select it through the env menuconfig` 怎么办？

- 该错误因为开启 AT Socket 功能之后，默认开启 at device 软件包中为配置对应的设备型号，进入 at device 软件包，配置设备为 ESP8266 设备，配置 WIFI 信息，重新 scons 生成工程，编译下载。

3. esp8266 设备连上网络后过一段时间 wifi 自动断开重连怎么办？

- 该错误一般为 esp8266 设备供电问题，可以使用万用表查看设备当前电压情况，如果出现供电不足问题，可以为 esp8266 vin 接口添加额外供电。

5 参考

5.1 本文所有相关的 API

5.1.1. API 列表

API	位置
at_client_init()	at_client.c
at_create_resp()	at_client.c
at_delete_resp()	at_client.c
at_exec_cmd()	at_client.c
at_resp_get_line()	at_client.c
at_resp_parse_line_args()	at_client.c
at_set_urc_table()	at_client.c
at_client_port_init()	at_client.c

5.1.2. 核心 API 详解

5.1.3. AT Client 初始化

```
int at_client_init(const char *dev_name,  rt_size_t recv_bufsz);
```

AT Client 初始化函数，属于应用层函数，需要在使用 AT Client 功能或者使用 AT CLI 功能前调用。**at_client_init** 函数完成对 AT Client 设备初始化、AT Client 使用的线程、信号量、互斥锁等资源初始化。

参数	描述
dev_name	AT 客户端使用设备名称
recv_bufsz	AT 客户端最大支持接收数据长度
返回	**描述**
>=0	成功
-1	失败
-5	失败，内存不足

5.1.4. 创建响应结构体

```
at_response_t at_create_resp(rt_size_t buf_size);
```

该函数用于创建指定最大返回数据长度的结构体对象，用于后面接收并解析发送命令响应数据。

参数	描述
buf_size	最大支持返回数据的长度
返回	**描述**
!= NULL	成功，返回指向响应结构体的指针
= NULL	失败

5.1.5. 删除响应结构体

```
void at_delete_resp(at_response_t resp);
```

该函数用于删除创建的响应结构体对象，一般与 **at_create_resp** 创建函数成对出现。

参数	描述
resp	准备删除的响应结构体指针
返回	**描述**
无	无

5.1.6. 发送命令并接收响应

```
rt_err_t at_exec_cmd(at_response_t resp, rt_size_t resp_line, const char
    *cmd_expr, ...);
```

该函数用于 AT Client 发送命令到 AT Server，并等待接收响应，是 AT Client 中的重要命令发送函数。

参数	描述
resp	响应结构体指针
resp_line	需要返回的数据行数，0 表示全部返回，>0 表示返回固定行数
cmd_expr	发送命令的表达式
...	发送命令的数据
返回	**描述**
>=0	成功
-1	失败
-2	失败，接收响应超时

5.1.7. 解析指定行号的响应数据

```
int at_resp_parse_line_args(at_response_t resp, rt_size_t resp_line,
    const char *resp_expr, ...);
```

该函数用于在 AT Server 响应数据中获取指定行号的一行数据, 并解析该行数据中的参数。

参数	描述
resp	响应结构体指针
resp_line	需要解析数据的行号
resp_expr	自定义的参数解析表达式
...	解析的参数
返回	**描述**
>0	成功，返回解析成功的参数个数

参数	描述
=0	失败，无匹参配数解析表达式的参数
-1	失败，参数解析错误

5.1.8. URC 数据列表初始化

```
void at_set_urc_table(const struct at_urc *table, rt_size_t size);
```

该函数用于初始化开发者自定义的 URC 数据列表，主要在 AT Client 移植函数中使用。

参数	描述
table	URC 数据结构体数组指针
size	URC 数据的个数
返回	**描述**
无	无