# 1. AT客户端框架

在之前的三篇教程中，我们都是直接使用串口助手发送AT指令与模组通信，本篇教程就来探索一下如何使用 MCU 中的串口模组交互。

什么是AT客户端

在使用AT指令的时候，直接发送AT指令的一端称为客户端(AT Client)，接收AT指令并返回响应的一端称为服务端(AT Server)。

ESP8266、M26、BC35-G这些通信模组都是接收我们发送的AT指令，所以称为AT命令服务端，MCU 需要向模组主动发送AT指令，称为AT客户端，它们之间的通信架构如下：

AT命令通信架构

为什么需要AT客户端框架

首先来看上图中的三个数据流：

发送AT指令：可以直接调用HAL库提供的API发送，AT框架并无太大作用；

等待接收返回结果：可以直接调用HAL库的API使用中断方式接收；

接收服务端主动发送的数据：可以直接调用HAL库的API使用中断方式接收；

三条数据流都可以调用HAL库的API直接实现呀，为什么要设计一层AT框架呢？

在直接调用HAL库实现的时候，首先无法保证每次模组向 MCU 发送的数据都能完整的被接收，所以，我们需要设计一层串口驱动以保证数据在任何时候都可以被完整的接收进缓冲区。

其次，在接收数据之后，难点在于对数据的处理，判断AT指令发送的数据是不是正常的返回结果，从返回结果中提取有效信息等等，这些如果每条指令接收之后，都去写代码依次判断，代码量陡增暂且不说，编程的难度也是直接上升，所以，我们需要基于串口驱动，在保证数据被完整接收的前提之上，再根据AT命令通信的特点，设计一层AT框架，专门负责解析数据，提取有效信息。

# 2. 剖析串口驱动框架实现

串口驱动直接使用LiteOS提供的驱动框架实现，由于其特殊性，最底层的驱动框架实现文件放在了工程目录中，调用HAL库提供的API实现：

串口驱动实现文件

uart\_at.c文件中，主要完成了两个功能：

串口初始化

实现串口驱动框架的读写，并注册串口设备到系统中

## 2.1. 串口初始化

串口初始化函数的调用架构如图：

串口初始化函数架构

其中默认初始化的是LPUART1，如果将其它串口作为AT指令的串口，修改这两行代码即可：

修改AT默认串口

## 2.2. ring\_buffer

ring\_buffer是专门实现的用户存放接收数据的缓冲区，用户只需要调用read和write操作缓冲区即可，其实现文件在iot-link SDK的IoT\_LINK\_1.0.0\iot\_link\link\_misc路径下：

ring\_buffer源码

ring\_buffer在串口初始化函数中被调用初始化：

ring\_buffer初始化

缓冲区大小在宏定义中声明：

ring\_buffer大小

初始化之后，向LiteOS注册的中断服务函数只需要调用ring\_buffer\_write向缓冲区不停的写入接收到的数据，即可保证串口数据被完整的接收。

## 2.3. 串口驱动框架实现

串口驱动框架中，因为已经有了初始化函数，所以只需要实现read函数和write函数即可，实现的函数架构如下：

串口驱动框架

因为数据全部保存在了ring\_buffer中，所以串口驱动的read API实现用缓冲区提供的读取函数实现即可。

实现read和write两个函数之后，调用如下的宏定义，即可将设备和驱动注册到系统中：

OSDRIV\_EXPORT(uart\_at\_driv,CONFIG\_AT\_DEVICENAME,(los\_driv\_op\_t \*)&s\_at\_op,NULL,O\_RDWR);

CONFIG\_AT\_DEVICENAME由用户指定，不重复即可，在iot\_link\_config.h文件中，稍后会讲解。

# 3. 剖析AT客户端框架

AT客户端框架的实现源码在SDK的IoT\_LINK\_1.0.0\iot\_link\at文件夹下：

AT框架源码

AT框架的架构如下：

AT框架架构

如图，因为串口设备已经注册到了系统中，所以AT框架的底层发送和接收函数直接调用LiteOS设备驱动框架提供的API实现，除了上述图中的这些，还涉及到大量的使用信号量、互斥锁、字符串比较等函数进行AT指令匹配处理，提取结果的代码，这些不是理解AT框架的重点，所以图中未给出。

在实现了AT框架之后，最终留给用户使用的接口只要三个，即可完成AT指令的交互，非常简洁：

at\_init：初始化AT框架，启动AT数据接收引擎(优先级为10)

at\_command：发送AT指令并匹配指定的返回结果

at\_oobregister：监控AT主动上报的数据

接下来，我们以ESP8266模组入网为例，讲述如何使用AT框架提供的简洁API与模组交互。

# 4. AT客户端框架的使用

AT框架使能及配置

经过上面的讲解，完整的AT框架其实包括设备驱动框架和AT框架实现两部分，所以首先需要在配置文件中使能驱动框架和AT框架。

打开之前新建的HelloWorld工程（如果没有可以参考之前的教程新建一个HelloWorld工程），在.sdkconfig中进行配置，如图：

使能驱动框架和AT框架

使能之后，不仅驱动框架的源码和AT框架的源码会被加入工程，还会进行自动初始化。

打开SDK中下的IoT\_LINK\_1.0.0\iot\_link目录中的link\_main.c文件，其中在link\_main函数即可看到：

首先是驱动框架的初始化：

自动初始化

其次是串口硬件和AT框架的初始化：

自动初始化

在自动初始化的时候，可以看到串口通信波特率由宏定义CONFIG\_AT\_BAUDRATE指定，串口设备注册到系统的名称由宏定义CONFIG\_AT\_DEVICENAME指定，那么，这两个宏定义在哪里指定呢？

不用的模组波特率不同，设备名称当然也不尽相同，所以这两个设置在工程目录中的OS\_CONFIG/iot\_link\_config.h中，这里我们使用ESP8266模组实验，设置如图：

串口设备配置

发送AT命令

发送AT指令的API原型及参数说明如下：

/\*\*

\* @brief:use this function to register a function that monitor the URC message

\* @param[in]:cmd, the command to send

\* @param[in]:cmdlen, the command length

\* @param[in]:index, the command index, if you don't need the response, set it to NULL; this must be a string

\* @param[in]:respbuf, if you need the response, you should supply the buffer

\* @param[in]:respbuflen,the respbuf length

\* @param[in]:timeout, the time you may wait for the response;and the unit is ms

\*

\* @return:0 success while -1 failed

\* \*/

int at\_command(const void \*cmd, size\_t cmdlen,const char \*index,\

void \*respbuf,size\_t respbuflen,uint32\_t timeout);

接下来我们在Demo文件夹之下创建一个文件夹at\_test\_demo，用于存放实验文件，并在该文件夹之下新建本节所使用的实验文件at\_esp8266\_demo.c：

新建实验文件

然后在文件中编辑以下内容：

#include <osal.h>

#include <string.h>

#include <at.h>

#define SSID "FAST\_88A6"

#define PASSWD "18324701020"

static bool\_t esp8266\_atcmd(const char \*cmd,const char \*index)

{

int ret = 0;

ret = at\_command((unsigned char \*)cmd,strlen(cmd),index,NULL,0,5000);

if(ret >= 0)

{

return true;

}

else

{

return false;

}

}

static int at\_esp8266\_demo\_entry()

{

char cmd[64];

int ret;

/\* 测试AT是否OK，超时时间5S \*/

memset(cmd,0,64);

snprintf(cmd,64,"AT\r\n");

while(false == esp8266\_atcmd(cmd, "OK"))

{

printf("AT Test fail, repeat.\r\n");

}

printf("AT test ok.\r\n");

/\* 关闭回显 \*/

memset(cmd,0,64);

snprintf(cmd,64,"ATE0\r\n");

ret = esp8266\_atcmd(cmd, "OK");

if(ret == false)

{

printf("ATE0 test fail.\r\n");

}

else

{

printf("ATE0 test ok.\r\n");

}

/\* 设置模式为AP+STA \*/

memset(cmd,0,64);

snprintf(cmd,64,"AT+CWMODE=3\r\n");

ret = esp8266\_atcmd(cmd, "OK");

if(ret == false)

{

printf("AT+CWMODE=3 test fail.\r\n");

}

else

{

printf("AT+CWMODE=3 test ok.\r\n");

}

/\* 连接路由器 \*/

memset(cmd,0,64);

snprintf(cmd,64,"AT+CWJAP=\"%s\",\"%s\"\r\n", SSID, PASSWD);

while(false == esp8266\_atcmd(cmd, "OK"))

{

printf("try to join AP:%s fail, repeat.\r\n", SSID);

}

printf("AT+CWMODE=3 test ok.\r\n");

return 0;

}

int standard\_app\_demo\_main()

{

osal\_task\_create("at\_esp8266\_demo",at\_esp8266\_demo\_entry,NULL,0x800,NULL,12);

return 0;

}

结果：

### 测试结果

获取AT指令返回结果并提取有效信息

对于AT命令返回的结果，如果其中存放有效信息，我们可以在调用at\_command时传入一个缓冲区，如下，发送查询模组的ip地址，并从中提取出ip地址：

在连接路由器的代码之后，添加如下代码：

/\* 获取ip地址 \*/

const char cs\_cmd[] = "AT+CIFSR\r\n";

char buffer[150];

char \*str;

uint8\_t ip[4];

memset(buffer,0,150);

ret = at\_command(cs\_cmd,strlen(cs\_cmd),"OK", buffer, 150, 5000);

if(ret < 0)

{

printf("AT+CIFSR test fail.\r\n");

}

else

{

printf("AT+CIFSR test ok.\r\n");

/\* 提取ip地址 \*/

str = strstr(buffer,"STAIP");

str = str + 7;

sscanf(str,"%d.%d.%d.%d",&ip[0],&ip[1],&ip[2],&ip[3]);

printf("ip: %d.%d.%d.%d\r\n", ip[0], ip[1], ip[2], ip[3] );

}

### 实验结果如图：

### 实验结果