

AT 库数据手册

修订原因

版本	日期	原因
V1.0	202/12/16	首版发布

1. AT 库概述

1.1. 库简介

[AT 库] 是一款专为嵌入式设备设计的 AT 指令交互开发库，采用分层架构设计，封装了硬件接口、数据收发、网络连接、协议交互等核心功能，旨在简化基于 AT 指令的模块（如 4G、5G、WiFi 模块）开发流程，降低开发难度，提高项目复用性和稳定性。

批注: 在开发初期是想帮助开发人员快速开发 AT 指令网络模块的，但是在首版开发完成阶段，发现本库不受限于 AT 指令，其余指令也可以使用本库，帮助开发人员快速开发产品，若遇到不合理注释，设计，请联系库开发者。;

1.2. 特性

- 分层解耦设计：硬件接口层、核心层、网络层、协议层独立划分，适配不同硬件平台和协议需求；
- 高效数据收发：支持 DMA 环形缓冲区，提供字节、字符串、定长数据、可变参数等多种发送方式；
- 灵活网络管理：内置联网状态机，支持自定义联网指令、超时重试、自动重启模块等功能；
- 可扩展协议支持：提供协议指令注册机制，支持自定义协议数据包构建、解析与校验；
- 轻量可靠：占用资源少，适配嵌入式 MCU 有限的存储和运算能力，关键流程具备超时容错机制。

1.3. 适用场景

- 嵌入式设备与 AT 指令模块的通信开发（4G/5G/WiFi/NB-IoT 等）；
- 基于 AT 指令的网络连接、数据上报、远程控制等场景；
- 需自定义协议与云端 / 服务器交互的嵌入式项目。

1.4. 依赖环境

- 硬件环境：支持 DMA、定时器、UART（或其他通信总线）的嵌入式 MCU；
- 软件环境：支持标准 C 语言的嵌入式开发环境（如 Keil MDK、IAR、GCC 等）；
- 底层依赖：需用户实现硬件外设（UART、DMA、定时器）的初始化代码。

2. AT 库架构

2.1. 库采用自上而下的分层架构，各层职责明确，层间通过接口交互，实现解耦：

层级	核心职责	依赖关系	对外头文件
硬件接口层	封装底层硬件操作，提供统一接口给核心层	依赖用户实现的硬件初始化	AT_HardwarePort.h
核心层	提供数据收发、缓冲区指针管理、状态机扫描等基础功能	硬件接口层	AT_Core.h
网络层	管理模块联网状态，实现联网流程控制、指令交互	核心层	AT_Network.h
协议层	支持自定义协议指令的构建、发送队列、解析与校验	核心层	AT_Protocol.h

注意：用户需要将每层头文件的宏提前填好，然后发给库开发者，进行统一资源分配；

3. AT_HardwarePort.h 硬件接口层简介

建议用户新建一个 xx_HardwarePort.c 模块，存放硬件接口注册代码，脱离应用层；

3.1. 外设资源

3.1.01. 网络模块的通信接口：数据帧格式需要和模块一致(一般 8 位)

3.1.02. DMA 通道：外设-存储器模式，外设端是通信接口的接收，开启循环模式

3.1.03. 网络模块的电源(复位)引脚初始化

3.1.04. 定时器 1ms 溢出的更新中断初始化(不限制定时器，可以实现定时任务即可)，通信接口跟 DMA 都不占用中断资源

3.2. 类型简介

3.2.01. AT_HW_Port_t 结构体

void(*send_byte)(uint8_t byte)	通信接口发送字节函数
void(*wait_data_send_finish)(void)	通信接口数据发送完成函数
void(*module_power)(void)	网络模块开关机(复位)时序函数
uint16_t(*dma_transfer_number_get)(void)	DMA 传输计数器获取函数

该类型用来注册硬件接口

3.3. 函数简介

3.3.01. AT_HW_RegisterState_t **AT_HW_Port_Register**(AT_HW_Port_t *hw_port_cfg) 函数

AT_HW_Port_t *hw_port_cfg	AT_HW_Port_t 类型的指针
AT_HW_RegisterState_t	返回值：成功或者失败，详见硬件接口层头文件

该函数用来注册库依赖的硬件接口

其中网络模块开关机时序函数可以按照我这个示例写，可以减少 Delay(示例代码是三极管开关的电路，所以是高电平拉低电源引脚)

```
gpio_bit_write(GPIOA, GPIO_PIN_1, RESET);  
Delay_ms(10);  
gpio_bit_write(GPIOA, GPIO_PIN_1, SET);
```

4. AT_Core.h 核心层简介

4.1.核心层快速使用说明

- (1) 用户提供 AT_Core_SerialBuf_t 类型中对应的 3 个缓冲区跟长度, 其中 recv_buf, recv_line_buf 缓冲区的长度要一致;
- (2) 用户通过 AT_Core_SerialBuf_Config 函数配置缓冲区(该配置要在通信接口,DMA 初始化之前完成);
- (3) 用户调用下列函数配置通信接口接收超时阈值

AT_Core_SerialRecv_SM_ScanThreshold_Set,
AT_Core_SerialRecv_SM_TimeoutThreshold_Set

- (4) 用户将该函数放入 1ms 扫描 1 次的任务中 AT_Core_SerialRecvTimeout_Task;

4.2.类型简介

4.2.01. AT_Core_SerialBuf_t 结构体

uint8_t *recv_buf	DMA 传输目的缓冲区 默认: NULL
uint8_t *recv_line_buf	DMA 环形缓冲区(大小跟 recv_buf 一致 默认: NULL)
uint8_t *temp_buf	可变参数函数发送缓冲区, 发送任务状态 机寻找 '>', "OK" 缓冲区 默认: NULL
uint16_t recv_buf_size	recv_buf recv_buf 长度 默认: 0
uint32_t temp_buf_size	temp_buf 长度 默认: 0L;

该类型是用来配置缓冲区的

4.3. 函数简介

4.3.01. void **AT_Core_SerialBuf_Config**(AT_Core_SerialBuf_t *serial_buf_cfg);

AT_Core_SerialBuf_t *serial_buf_cfg	AT_Core_SerialBuf_t 类型的指针
配置缓冲区 接口函数	

4.3.02. void **AT_Core_SerialRecv_SM_ScanThreshold_Set**(uint32_t threshold);

uint32_t threshold	阈值 默认: 1
--------------------	-------------

通信接口接收状态机扫描计数器阈值设置;

4.3.03. void **AT_Core_SerialRecv_SM_TimeoutThreshold_Set**(uint32_t threshold);

uint32_t threshold	阈值 默认: 3
--------------------	-------------

通信接口接收状态机超时阈值设置;

4.3.04. void **AT_Core_SerialRecvTimeout_Task**(void);

通信接口接收超时任务,放入周期 1ms 的任务区中;

4.3.05. void **AT_Core_SendByte**(uint8_t byte);

uint8_t byte	待发送的字节数据
发送字节函数;	

4.3.06. void **AT_Core_SendString**(const char *string);

const char *string	待发送的字符串数据
发送字符串函数	

4.3.07. void **AT_Core_SendFixedLengthData**(const char *string, uint32_t length);

const char *string	待发送数据
uint32_t length	待发送的数据长度

发送定长数据函数;

4.3.08. int **AT_Core_Printf**(const char *string, ...);

const char *string	格式符字符串
...	格式符, %s, %d, %c,...

格式符数据发送函数;

4.3.9. void **AT_Core_ModulePower**(void);

网络模块开关机(复位)函数

4.3.10. uint16_t **AT_Core_DMA_TransferNumber_Get**(void);

uint16_t	返回值: DMA 传输计数器数量
DMA 传输计数器获取函数;	

4.3.11. char ***AT_Core_RecvBuf_Check**(const char *string);

const char *string	待校验的字符串
char *	返回值: 找到则返回 string 在 DMA 环形缓冲区首地址指针, 未找到返回 NULL

返回在 DMA 环形缓冲区中 string 第一次出现的地方, 未找到返回 NULL;

5. AT_Network.h 网络层简介

5.1. 网络层快速使用说明

- (1) 调用 `AT_Network_Menu_MempoolInit` 函数初始化内存池;
- (2) 定时调用 `AT_Network_Task` 函数, 建议 300ms 以上;
- (3) 实现发送网络指令函数

```
void TT_Send_CSQ(void) { AT_Core_SendString("AT+CSQ\r\n"); }
```

(4) 实现接收网络指令函数

```
char *TT_Recv_CSQ(void) { return AT_Core_RecvBuf_Check("OK"); }
```

(5) 调用此函数,传入上面的 2 个参数

AT_Network_Directive_Add(CSQ ,TT_Send_CSQ, TT_Recv_CSQ,); 即可注册成功;

(6) 其中第一个参数建议用户自行声明枚举, 方便管理指令的唯一标识;

(7) 调用 AT_Network_StartLinkDirective_Set 函数设置网络连接起始指令, 用户可自行决定从哪条指令开始连接;

5.2.类型简介

5.2.01. AT_Network_MenuState_t 枚举

AT_NETWORK_MENU_RAM_ERROR	超出内存池上限
AT_NETWORK_MENU_PARAM_DEFAULT	指针参数非法,例如 NULL
AT_NETWORK_MENU_DIRECTIVE_INDEX_REPEAT	网络指令的唯一标识重复
AT_NETWORK_MENU_NO_FOUND_INDEX	未找到该标识的网络指令
AT_NETWORK_MENU_SUCCEED	成功

该类型是 AT_Network.h 网络层内一些接口函数的返回状态;

5.2.02. AT_Network_t 结构体

Uint8_t network_flag	网络在线,离线标志
uint8_t iccid[25]	Sim 卡号
uint8_t imei[20]	网络模块全球唯一标识码

该类型是网络状态标志位, 用户可定义一个该类型的全局变量(也可使用自身的)

5.2.03. AT_Network_Param_t 结构体

uint32_t speed_threshold	网络发送,接收速率阈值 默认:300
uint32_t timeout_threshold	网络超时阈值 默认:7000
uint8_t retry_threshold	网络指令重试阈值 默认:3

该类型用来配置网络参数;

5.3. 函数简介

5.3.01. void AT_Network_Menu_MempoolInit(void); 函数

该函数是用来初始化内存池的,在网络模块初始化完成之前执行;

5.3.02. AT_Network_MenuState_t AT_Network_Directive_Add(uint32_t

directive_index, void(*send_command)(void), char *(recv_command)(void));

uint32_t directive_index	网络指令的唯一标识
void(*send_command)(void)	发送函数指针
char *(recv_command)(void))	接收函数指针
AT_Network_MenuState_t	返回值

该函数用来添加网络指令,在 AT_Network_Menu_MempoolInit 函数之后执行, 网络菜单指令可以任意的添加跟删除, 并且可以通过平台远程下发指令, 删除或者添加网络指令, 前提条件是用户提前构建好待添加的指令;

注意: 网络指令的执行顺序是根据用户添加的顺序执行

5.3.02. AT_Network_MenuState_t AT_Network_Directive_Delete(uint32_t

directive_index);

uint32_t directive_index	待删除的网络指令唯一标识
AT_Network_MenuState_t	返回值

该函数用来删除网络指令,可以在运行过程中通过某种事件删除网络指令,不会破坏网络菜单, 库内会对网络链表进行修复;

5.3.03. AT_Network_MenuState_t AT_Network_StartLinkDirective_Set(uint32_t

start);

uint32_t start	网络指令唯一标识
AT_Network_MenuState_t	返回值

该函数用来设置起始网络指令;

5.3.04. AT_Network_MenuState_t **AT_Network_CurrentLinkDirective_Set**(uint32_t current);

uint32_t current	网络指令唯一标识
AT_Network_MenuState_t	返回值

该函数用来设置当前网络连接指令

5.3.05. void **AT_Network_ParamConfig**(AT_Network_Param_t *network_param_cfg);

AT_Network_Param_t*network_param_cfg	AT_Network_Param_t 类型的指针
--------------------------------------	--------------------------

该函数用来配置网络参数;

5.3.06. void **AT_Network_Task**(void);

该函数是网络任务, 需要定时执行, 由 AT_Network_Param_t 类型的 speed_threshold 成员决定;

5.3.07. static inline int **Find_FirstZero**(uint32_t value)

uint32_t value	待寻找 0 的值
int	返回值: 从最低位开始,0 在 value 的位置

该函数是为了兼容不同编译器编写;

5.4. 宏简介

AT_NETWORK_MENU_MEMPOOL_SIZE	网络菜单内存池容量
------------------------------	-----------

该宏需要用户自行决定好值后, 发给库开发者, 进行容量分配;

6. AT_Protocol.h 协议层简介

6.1. 协议层快速使用说明

构建协议指令示例

(1) AT_Protocol_Task 函数放入主循环扫描

(2) 实现构建协议指令数据包函数

```
void TT_Login_Build(uint8_t port)
{
    … 构建数据包 …
    构建完成后将数据包起始地址,长度传给 协议发送任务状态机;
    AT_Protocol_Send_SM_DataPacket_Write(JsonStructure.buf, JsonStructure.index);
}
```

(3) 实现接收解析协议指令函数

```
char *TT_Login_Recv(char *addr)
{ /* 保留 */ return NULL; }
```

由于登录指令可以不用校验平台返回的数据, 故写一个空的返回函数, 如果该指令需要接收解析的, 那么此函数内就是用户的解析代码, addr 是

AT_Protocol_Directive_Add 函数的第 4 个参数, 也就是指令标识的起始地址;

(4) 用户自行定义该协议指令的索引唯一标识, 建议使用枚举来管理

(5) 调用 AT_Protocol_Directive_Add(TT_LOGIN, TT_Login_Build, TT_Login_Recv,
“\”msgType:500\””);即可

(6) 用户如果想发送协议指令, 只需要 AT_Protocol_SendQueue_Enqueue((TT_LOGIN,
0); 调用此函数, 返回值不为 0 代表入队成功, 开始发送;

设备状态标志位使用方法

(1) AT_Protocol_DeviceState_Task 函数放入 1ms 的任务区扫描

(2) 通过 AT_Protocol_DeviceStateFlag_Get 函数判断设备状态标志位是否置位, 并且可以通过 AT_Protocol_DeviceStateFlag_Set, AT_Protocol_DeviceStateFlag_Clear 函数进行置位跟清除位标志;

(3) 例如心跳指令标志位置位后, 可以执行 AT_Protocol_SendQueue_Enqueue 函数入队, 进行发送;

6.2. 类型简介

6.2.01. AT_Protocol_MenuState_t 枚举

AT_PROTOCOL_MENU_EXCEED	协议菜单内的指令超出范围
AT_PROTOCOL_MENU_PARAM_DEFAULT	传入的指针参数非法,例如 NULL
AT_PROTOCOL_MENU_SUCCEED	成功

该类型是协议层一些函数的返回状态;

6.2.02. AT_Protocol_ModuleParam_t 结构体

char *send_cmd	模块发送命令 默认: "AT+QISEND=0,%d\r\n"
char *recv_urc	模块接收到服务器数据后上报的 URC 默认: "+QIURC: \\"recv\\""
uint16_t send_max_len	模块单次发送最大长度 默认: 1440

该类型用来注册网络模块参数;

6.3. 函数简介

6.3.01. void AT_Protocol_Task(void);

该函数是协议任务函数, 主循环调用;

6.3.02. AT_Protocol_MenuState_t AT_Protocol_Directive_Add(uint32_t

```
directive_index, void(*build_func)(uint8_t port), char *(*recv_func)(char *addr),
char *directive);
```

uint32_t directive_index	协议指令的唯一标识
void(*build_func)(uint8_t port)	构建协议指令数据包函数
char *(*recv_func)(char *addr),	接收协议指令函数
char *directive	指令标识, 例如 EBCharge#1..., 则传入 EBCharge;
AT_Protocol_MenuState_t	返回值

该函数是用来添加协议指令的, 由于协议指令不会随意的删除, 故不提供删除函数, 再初始化的时候就需要完成协议指令菜单的生成;

6.3.03. AT_Protocol_MenuState_t **AT_Protocol_Send_SM_DataPacket_Write**(char *packet_addr, uint32_t packet_len);

char *packet_addr	数据包起始地址指针
uint32_t packet_len	数据包长度
AT_Protocol_MenuState_t	返回值

该函数是在构建完数据包后(build_func 函数), 将数据包起始地址跟长度传给协议发送任务状态机;

6.3.04. uint8_t **AT_Protocol_SendQueue_Enqueue**(uint32_t directive_index, uint8_t port);

uint32_t directive_index	协议指令的唯一标识
uint8_t port	指定端口
uint8_t	返回值: 0 失败, 1 成功

该函数是协议指令入队, 将协议指令构建函数的唯一标识, 端口(无需指定端口传 0), 传入, 返回值为 1 即代表入队成功, 入队成功后会立马开始发送

6.3.05. AT_Protocol_MenuState_t

AT_Protocol_ModuleParamConfig(AT_Protocol_ModuleParam_t *module_param_cfg);

AT_Protocol_ModuleParam_t*module_param_cfg	AT_Protocol_ModuleParam_t 类型的指针
AT_Protocol_MenuState_t	返回值

该函数用来配置网络模块的参数, 例如发送命令, 接收 URC, 发送最大长度…;

6.3.06. void **AT_Protocol_DeviceState_Task** (void);

该函数是扫描设备状态, 内部嵌入了, 端口状态指令计数器, 心跳指令计数器, 心跳超时计数器, 以及相应的阈值, 当超过阈值, 将会置位相应标志位, 如下

登录指令标志	AT_PROTOCOL_LOGIN_SEND_FLAG
--------	-----------------------------

端口状态指令标志	AT_PROTOCOL_PORT_STATE_SEND_FLAG
心跳指令标志	AT_PROTOCOL_BEAT_SEND_FLAG
心跳超时标志	AT_PROTOCOL_BEAT_TIMEOUT_FLAG

6.3.07. void **AT_Protocol_PortStateThreshold_Set**(uint32_t value);

uint32_t value	值 默认: 60000
----------------	----------------

该函数是 端口状态指令计数器阈值设置;

6.3.08. void **AT_Protocol_BeatThreshold_Set**(uint32_t value);

uint32_t value	值 默认: 30000
----------------	----------------

该函数是 心跳指令计数器阈值设置;

6.3.09. void **AT_Protocol_BeatTimeoutThreshold_Set**(uint32_t value);

uint32_t value	值 默认: 50000
----------------	----------------

该函数是 心跳超时计数器阈值设置;

6.3.10. void **AT_Protocol_RetryThreshold_Set**(uint32_t value);

uint32_t value	值 默认: 0x000EFFFFU
----------------	----------------------

该函数是 协议发送任务状态机重试次数阈值设置

举例,当发送 "AT+QISEND=0,%d\r\n" 后,模块超过这个 SumTime,会置位 AT_PROTOCOL_BEAT_TIMEOUT_FLAG 离线标志

* Freq_Ref: 168MHZ(基准频率)

* Once_Ref: 0.00001057s(基准频率下重试 1 次的时间)

* 计算不同频率下单次重试时间公式: Once_Now = Once_Ref * (Freq_Ref / Freq_Now)

* 总时间公式: SumTime = Once_Now * value,单位 s;

6.3.11. `uint32_t AT_Protocol_DeviceStateFlag_Get(void);`

<code>uint32_t</code>	返回值: 设备状态标志位对应的变量
-----------------------	-------------------

该函数返回设备状态标志位对应的变量, 用户可以通过此变量判断心跳指令, 心跳超时等标志位是否置位;

6.3.12. `void AT_Protocol_DeviceStateFlag_Set(uint32_t device_state_flag);`

<code>uint32_t device_state_flag</code>	设备状态标志位(只能传入 位)
---	-----------------

该函数是 设备状态标志位设置;

6.3.13. `void AT_Protocol_DeviceStateFlag_Clear(uint32_t device_state_flag);`

<code>uint32_t device_state_flag</code>	设备状态标志位 (只能传入 位)
---	------------------

该函数是 设备状态标志位清除;

6.1.14. 设备状态标志位

<code>AT_PROTOCOL_LOGIN_SEND_FLAG</code>	登录指令标志位
<code>AT_PROTOCOL_PORT_STATE_SEND_FLAG</code>	端口状态指令标志位
<code>AT_PROTOCOL_BEAT_SEND_FLAG</code>	心跳指令标志位
<code>AT_PROTOCOL_BEAT_TIMEOUT_FLAG</code>	心跳超时标志位

6.4. 宏简介

<code>AT_PROTOCOL_MENU_SIZE</code>	协议菜单大小
<code>AT_PROTOCOL_LINK_BAG_MAX</code>	连包最大处理次数
<code>AT_PROTOCOL_SEND_TASK_QUEUE_MAX</code>	发送任务队列大小

该宏需要用户自行根据项目决定好值后, 发库开发者分配资源;

编写人: 王土成