深圳市海凌科电子有限公司

HLK-N10_API 接口说明

版本：V1.0　　修订日期：2020 年 7 月 18 日

使用前说明

本书采用的约定及标志如下。

1．符号约定

带尖括号“ < > ”表示键名、按钮名以及操作员从终端输入的信息；带方括号“ [] ”表示人机界面、菜单条、数据表和字段名等，多级菜单用“→”隔开。如［文件→新建→文件夹］多级菜单表示［文件］菜单下的［新建］子菜单下的［文件夹］菜单项。

2．键盘操作约定

格式	意义
加尖括号的字符	表示键名、按钮名。如 < Enter > 、< Tab > 、< Backspace > 、< a > 等分别表示回车、制表、退格、小写字母 a
< 键 1+键 2 >	表示在键盘上同时按下几个键。如 < Ctrl+Alt+A > 表示同时按下“Ctrl”、“Alt”、“A”这三个键
< 键 1，键 2 >	表示先按第一键，释放，再按第二键。如 < Alt，F > 表示先按 < Alt > 键，释放后，紧接着再按 < F > 键

3．鼠标操作约定

格式	意义
单击	快速按下并释放鼠标的左键
双击	连续两次快速按下并释放鼠标的左键
右击	快速按下并释放鼠标的右键
拖动	按住鼠标的左键不放，移动鼠标

4．标志

小心、注意、警告、危险前使用符号“  ”。

说明、提示、小窍门前使用符号“  ”。

目录

1 概述

本文档是为 OPENCPU 开发提供的上层业务相关接口，包括 socket 套接字/OneNET/CDP/DNS 等，对应的 demo 在 SDK 的路径：userapp/demo。

2 用户二次开发注意事项

用户只能在 userapp 文件夹下开发用户自己的代码，不准在其他文件夹中修改和添加任何代码。

路径 userapp/Inc 中提供了用户可能用到的所有声明，原则上禁止调用平台其他外部头文件声明，socket 套接字除外。

用户进行 OPENCPU 二次开发时，优先推荐使用本文档中介绍的 API 接口方式进行网络上层业务开发，不建议调用扩展 AT 命令进行二次开发，3GPP 标准 AT 命令除外。

3 操作系统适配层

os_adapt.h 为海凌科基于 liteos 操作系统的适配接口，用户可放心调用。

3.1 xy_printf

原型	#define xy_printf(fmt, args...) xy_log_print(USER_LOG, WARN_LOG, fmt, ##args)
功能	用于用户二次开发时从 log 口输出 log 使用，需要使用海凌科提供的 logView 工具打开。如果用户仅想查看自己的 log，不输出芯片内部 log，可以通过 AT+NV=SET,LOG,3 来设置出厂 NV 参数 open_log 值为 3。

注：该接口不可以在锁中断期间调用，否则会断言。

3.2 xy_assert

原型	#define xy_assert(a) {xy_assert_proc(!(a), __FILE__, __LINE__);}
功能	用于软件断言，死机后，会导出 dump 文件以供软件开发定位问题，详情参阅《HLK-N10 Trace32 使用及死机定位指导》

3.3 send_debug_str_to_at_uart

原型	void send_debug_str_to_at_uart(char *buf);
功能	仅用于从 AT 口输出 debug 调试的主动上报 URC，该接口可以在锁中断期间调用，但是不能保证一定会输出到串口。为了对整机不产生影响，调用该接口的调试类 URC，必须以

	“+DBGINFO:” 方式输出

示例	send_debug_str_to_at_uart("+DBGINFO:var nv write!\r\n");
	send_debug_str_to_at_uart("+DBGINFO:NET nv write!\r\n");
	send_debug_str_to_at_uart("+DBGINFO:USER nv write!\r\n");

3.4 xy_task_realtime_set

原型	void xy_task_realtime_set (int enable)
功能	供用户动态配置用户任务的软 timer 实时性，设为 1 表示用户的软定时会准时到达；设为 0 后，软定时则不会准时到达，进而降功耗。 用户开发中创建的 task 的默认是实时的
参数	enable：lpm_flag 的开关，1 表示打开；0 表示关闭

4 平台基础接口

4.1 xy_work_lock

原型	void xy_work_lock(int lock_dsp);
功能	申请工作锁，期间不会进入深睡 DEEPSLEEP；其中 lock_dsp 指示是否需要锁住 DSP 核。对于 OPENCPU 产品单核形态，如果无需使用 NB 协议栈，则入参设为 0 即可。若需使用 DSP 协议栈，则入参需设为 1

4.2 xy_work_unlock

原型	void xy_work_unlock ()
功能	释放工作锁，若全部锁皆被释放，则会快速进入深睡 DEEPSLEEP。

备注：

1) 对于单一用户实体，如模组类用户，可以使用 xy_fast_power_off(AT 命令：“AT+FASTOFF”)。
2) 对于多实体用户，不得使用 xy_fast_power_off 机制，必须配对使用 xy_work_lock 和 xy_work_unlock，否则会造成其他业务主体正在工作时进入深睡。

4.3 xy_standby_set

原型	void xy_standby_set(int do_open);
功能	standby 睡眠机制的动态开关，1 表示打开 standby 睡眠；0 表示关闭。

4.4 xy_fast_power_off

原型	void xy_fast_power_off(int abnormal_off);
功能	快速进入 DEEPSLEEP，接口内部会强制性进入 DEEPSLEEP，不关心锁数量。此时不响应任何中断唤醒，进而不能被任何事务打断，存在丢中断风险。
参数	abnormal_off 为 0 时，平台内部执行 NV 的保存动作，与 AT+WORKLOCK=0 的差异为不发送 RAI 快速链接释放到网侧，且不关心锁持有数。 abnormal_off 为 1 时，类似直接断电操作，软件快速进行芯片深睡。该参数值慎重！

备注：

1) 对于单一用户实体，如模组类用户，可以使用 xy_fast_power_off(“AT+FASTOFF”)。
2) 对于多实体用户，不得使用 xy_fast_power_off 机制，必须配对使用 xy_work_lock 和 xy_work_unlock，否则会造成其他业务主体正在工作时进入深睡。

4.5 xy_reboot

原型	void xy_reboot(int xy_nv_save,int user_nv_save);
功能	供用户调用以软重启芯片，会重新加载软件，进而提供两个参数供用户进行 NV 的保存设置。一般的，平台内部的 NV 需要保存。用户的 NV 是否保存，由用户通过入参 user_nv_save 决定。具体的 NV 操作接口，参见回调函数 user_flash_hook_by_RB。 软重启过程中，用户设置的 UTC 仍然有效，如果用户希望无效，必须在调用软重启接口之前，调用 xy_rtc_timer_delete 接口删除响应的 URC 设备。
参数	xy_nv_save：平台 NV 的保存标记； user_nv_save：用户 NV 的保存标记；

4.6 is_powenon_from_deepsleep

原型	int is_powenon_from_deepsleep();
功能	指示芯片是否从深睡模式再次上电工作，进而指示 flash 中保存的业务数据是否有效，以便快速恢复云业务。

4.7 user_sys_init

原型	void user_sys_init();
功能	系统初始化时，用户相关的系统初始化接口，主要为驱动、系统资源等初始化。

4.8 user_app_init

原型	void user_app_init();
功能	用户线程任务的开机初始化接口，内部通过位图方式实现了 demo 的测试能力，用户可自行按照产品需求进行修改。

4.9 xy_check_low_vBat

原型	int xy_check_low_vBat(int state)
功能	检测 VBAT 电压是否过低。系统在开机初始化和 flash 擦写操作前，会调用接口识别是否为低压。若是第一次识别为低压，则软重启系统，尝试恢复正常电压；若重启后仍然低压，则系统不再进行任何的 flash 擦写操作，并通过 URC 或全局告知用户进行策略动作。 该接口必须配置出厂 NV(min_mVbat)一起使用，由用户根据自身的产品形态设置合理的低压工作门限。
参数	state：暂未使用；
返回值	1 表示电压过低；0 表示电压正常

4.10 xy_getVbatCapacity-用户需二次开发

原型	unsigned int xy_getVbatCapacity()
功能	用于获取当前电池电量，以供业务模块检测电池电量是否过低，若剩余电量不足以完成高耗电场景，如 FOTA 升级等，则需要放弃当前动作，并提醒用户更换电池。 该接口由用户二次开发实现，目前海凌科平台在 xy_is_enough_Capacity 接口中会调用，以裁决当前电量是否足够 FOTA 升级。用户根据自身的业务开发，也可以在适当点调用获取当前电量。

4.11 xy_is_enough_Capacity

原型	int xy_is_enough_Capacity (int state)
功能	用于检查当前电池电量是否低于出厂 NV(min_mah)设置的门限值，若剩余电量不足以完成 FOTA 升级等高耗电场景，则需要放弃当前动作，并提醒用户更换电池。 该接口必须配置出厂 NV(min_mah)一起使用。目前该接口仅在 FOTA 升级之前调用，若发现电量不足，则平台会放弃此次升级。用户根据自身的业务开发，也可以适当调用该接口，以决定是否能执行某高功耗的动作。
参数	state：暂未使用；
返回值	1 表示电量充足；0 表示电量不足

4.12 get_rand_val

原型	int get_rand_val();
功能	用于根据 UTC 的随机因子获取一个 int 型的随机值，用户如果需要其他的随机值，可自行修改实现。

5 flash 操作相关

5.1 注意事项

芯片支持工作期间实时写 flash，但是由于执行写 flash 操作时，会退出 XIP 模式，进而不能运行 flash 上的代码。所以要求所有的中断服务程序中不得调用 flash 上的代码。可以用 __attribute__((section(".ramtext")))方式把某些函数放在 RAM 上。

平台提供了从 USER_FLASH_BASE 开始，大小为 USER_FLASH_LEN_MAX 字节的 flash 空间供用户保存数据到 flash 中。用户不得使用超出设计范围的 flash 空间，否则会造成未知错误。具体使用指导参看《海凌科 HLK-N10 平台开发指南》的“NV 与 flash 的使用”章节。

5.2 user_hook_flash_restore-回调接口

原型	void user_hook_flash_restore();
功能	开机初始化时，对用户的 flash 数据的回调操作，常见于读取用户 NV 和 flash 数据到 RAM 中。

5.3 user_hook_flash_save-回调接口

原型	void user_hook_flash_save();
功能	深睡之前，对用户的 NV 和 RAM 等数据的回写 flash 回调操作。常伴随于 AT+WORKLOCK=0 和 AT+FASTOFF 两个命令之后。

5.4 user_flash_hook_by_fastoff-回调接口

原型	void user_flash_hook_by_fastoff();
功能	执行异常快速深睡 AT+FASTOFF=1 命令(类似直接断电)时，用户 NV 回写 flash 的策略回调接口，用户自由进行私有 NV 的擦除或保存操作。

5.5 user_flash_hook_by_RB-回调接口

原型	void user_flash_hook_by_RB (int user_nv_save);
功能	由正常 AT 命令触发软重启时，用户 NV 回写 flash 的策略回调接口，根据入参进行 NV 的擦除和保存操作即可。目前能够触发软重启的 AT 命令有：AT+NRB/AT+NV=SAVE/AT+NATSPEED。

5.6 erase_flash_hook-回调接口

原型	void erase_flash_hook(int state);
功能	通过 AT+RESET 命令、异常断言、硬看门狗、PIN RESET 按键等，执行软 reset 后，对 flash 空间的回调接口。对于出厂 NV，用户可以自行决定是否恢复到原始出厂 NV；平台的运行态 NV，建议擦除；对于用户空间的 flash 数据，即 USER_FLASH_BASE 开始的空间，用户根据自己产品需要决定是否擦除。
参数	0：AT+RESET 触发的重启; 1：assert or watchdog 触发的重启; 2：PIN reset 触发的重启

5.7 xy_Flash_Read

原型	void xy_Flash_Read(int addr,unsigned char *data, int size);
功能	供用户读取从 USER_FLASH_BASE 开始的用户 flash 数据内容到 data 空间中。由于读取大数据耗费时间过长，平台内部提供了 DMA 硬加速器。
参数	addr：用户的 FLASH 地址，最小值为 USER_FLASH_BASE，最大值为 USER_FLASH_BASE 和 USER_FLASH_LEN_MAX 的空间和; data：读取内存的存放空间，由用户申请该空间 size：读取的大小

5.8 xy_Flash_Write

原型	int xy_Flash_Write(int addr,unsigned char *data, int size);
功能	写用户数据到 flash 中，内部先执行 erase 擦除动作，再执行 write 写操作。该接口常见于用户的出厂 NV 等动态运行相关参数保存，要求不能够改变同一个扇区中其他部分的内容值。 由于 FLASH 仅支持 10 万次的写操作，不建议用户运行过程中频繁调用该接口，而应该在 sys_hook_func.c 中的回调函数执行写操作，即软重启、深睡等才执行写动作。 为了防止低压时造成 flash 擦写异常，接口内部一旦识别为低压，则放弃操作，返回 0 表示失败。
参数	addr：用户的 FLASH 地址，最小值为 USER_FLASH_BASE，最大值为 USER_FLASH_BASE+USER_FLASH_LEN_MAX; data：用户数据首地址，由用户申请该空间并赋值 size：待写入的数据长度
返回值	bool 类型，1 表示成功；0 表示失败

5.9 xy_Flash_Write_no_erase

原型	int xy_Flash_Write_no_erase(UINT32 addr,void *data, int size);
功能	写用户数据到 flash 中，不执行擦除操作。使用该接口的前提是能够保证待写入的 flash 空间已被擦除为全 FF，否则会造成脏数据。 该接口常见于顺序保存用户数据，如 FOTA、传感器类数据等，一般与 xy_Flash_Erase 接口配套使用，即先擦除一大段 flash，再一小段一小段分时写入 为了防止低压时造成 flash 擦写异常，接口内部一旦识别为低压，则放弃操作。
参数	ddr：用户的 FLASH 地址，最小值为 USER_FLASH_BASE，最大值为 USER_FLASH_BASE+ USER_FLASH_LEN_MAX； data：用户数据首地址，由用户申请该空间并赋值 size：待写入的数据长度
返回值	bool 类型，1 表示成功；0 表示失败

5.10 xy_Flash_Erase

原型	int xy_Flash_Erase(UINT32 addr, UINT32 size);
功能	擦除用户 flash 空间。芯片不支持实时的写 flash 操作，该接口仅允许在上述的各回调接口中调用。该接口属于暴力擦除，请谨慎使用！ 为了防止低压时造成 flash 擦写异常，接口内部一旦识别为低压，则放弃操作。
参数	addr：用户的 FLASH 地址，最小值为 USER_FLASH_BASE，最大值为 USER_FLASH_BASE+ USER_FLASH_LEN_MAX； size：待擦除的长度
返回值	bool 类型，1 表示成功；0 表示失败

6 时间定时相关

6.1 RTC 硬定时器

RTC 定时器目前只有一个硬件资源，平台通过软件方式实现了多个软 RTC ID，目前能给用户使用的为 RTC_TIMER_USER1 和 RTC_TIMER_USER2。RTC 定时器在深睡时不掉电，超时后能够唤醒芯片执行 RTC 中断。对于 PIN_RESET、assert、watchdog 等异常重启，RTC 会被重新复位，用户任务每次上电初始化时，需要识别当前用户 RTC 是否存在，若不存在，必须重设 RTC 定时器。

平台维护了与基站同步的世界时间，前提是 NB 终端 attach 小区成功，若未 attach 成功，则本地的世界时间是从 2018/10/1 开始的。如果用户需要正确的世界时间，需要确保 attach 成功后，再使用 RTC 接口功能。RTC 采用的是 32K 晶振，精度低，连续运行若干天后，会造成偏差变大，若用户需要较精确的世界时间，可以自行在合适的时机重新出发小区 attach。

6.2 xy_rtc_set_time

原型	int xy_rtc_set_time(struct xy_wall_clock *rtctime,int zone_sec);
功能	用于设置 RTC 当前的世界时间。平台提供了两种世界时间设置能力，即开机后 attach 成功，通过基站获取当前时间，主动上报命令为 “+CTZEU:”。另一种就是通过该接口由用户手动更新当前的世界时间。 为了保证世界时间的准确性，平台默认从基站获取最新的世界时间，并允许用户通过该接口进行更新。平台内部始终使用最新的世界时间。
参数	rtctime：当前设置的世界时间，例如 2020/10/1 zone_sec：时区秒数，例如北京时间位于东八区，即 8*60*60 秒；如果 rtctime 已经考虑了时区，填 0 即可
返回值	bool 型，目前始终为 0，即未使用返回值

6.3 xy_rtc_get_time

原型	int xy_rtc_get_time(struct xy_wall_clock *rtctime);
功能	用于获取当前的世界时间，返回值中已考虑了时区，即在中国返回的是北京时间。 用户使用该接口时，需要关注是否能够获取到有效值。
参数	rtctime：获取到的世界时间，例如 2020/10/1
返回值	bool 型，返回值为 0，表示当前没有有效的世界时间，常见于未 attach 成功。

6.4 xy_rtc_get_UT_offset

原型	int xy_rtc_get_UT_offset(struct xy_wall_clock *wall_clock,int offset_sec);
功能	用于获取某个时刻偏移的世界时间。用户使用该接口时，需要关注是否能够获取到有效值。offset_sec 时间偏移可以灵活使用，如设为 0 时，该接口则与 xy_rtc_get_time 功能一致，即获取当前的世界时间；再如 offset_sec 设为(0-8*60*60)，即向前 8 小时的世界时间，则对应当前北京时间的世界时间。
参数	wall_clock：获取到的世界时间，例如 2020/10/1 offset_sec：秒偏移，可正可负
返回值	bool 型，当值为 0，表示当前没有有效的世界时间，常见于未 attach 成功。

6.5 xy_rtc_get_sec

原型	int xy_rtc_get_sec (int zone_sec);
功能	用于获取当前的世界时间的秒数值，即相对于 1970/1/1 已过去多少秒。 用户使用该接口时，需要关注是否能够获取到有效值。
参数	zone_sec：时区秒数，平台内部记录的是北京时间，如果需要获取当前世界时间秒数，入参填写 8*60*60 即可
返回值	秒数，返回值为 0，表示当前没有有效的世界时间，常见于未 attach 成功。

6.6 xy_rtc_timer_create

原型	int xy_rtc_timer_create(char timer_id,int sec_offset,rtc_timeout_cb_t callback, void *data);
功能	注册用户的 RTC 定时器。由于 RTC 硬件只有一个，软件上实现了定时复用，目前仅支持两个用户定时器，如果不够用，请与海凌科 FAE 联系。考虑到 NB 系统的世界时间不能始终维护更新，进而定时器仅支持秒数偏移，不支持设置严格的世界时间格式的定时功能。
参数	timer_id：查看 USER_RTC_TIMER_ID sec_offset：秒数偏移 callback：超时回调函数，正常深睡会保存 data：超时回调入参

备注：对于低功耗产品，如果用户设置的周期性定时小于半小时，请务必与海凌科的 FAE 联系。

6.7 xy_rtc_timer_delete

原型	int xy_rtc_timer_delete(char timer_id);
功能	删除注册用户的 RTC 定时器。
参数	timer_id：查看 USER_RTC_TIMER_ID

6.8 xy_rtc_next_offset_by_ID

原型	int xy_rtc_next_offset_by_ID(char timer_id);
功能	获取某用户定时器 ID 的下一次超时的秒数偏移。该接口常用于异常重启等保护使用，即重启后查看是否有效，若无效，则重设该定时器。
参数	timer_id：查看 USER_RTC_TIMER_ID
返回值	整型，返回值为 0 表示当前 ID 设置无效

6.9 set_rtc_by_day_offset

原型	int set_rtc_by_day_offset(char timer_id,rtc_timeout_cb_t callback, void *data,int hour_start,int hour_span,int min,int sec);
功能	用于设置每天的某个时间段的 RTC 定时。为了减少同批产品同时唤醒联网，产品开发时务必设置时间段，以减轻基站压力，提供通信成功率。例如，用户想每天 15:00--18:00 随机做事，则设置： set_rtc_by_day_offset(RTC_TIMER_USER1,user_rtc_timeout_cb,NULL,15,3,0,0)
参数	timer_id：查看 USER_RTC_TIMER_ID 大于等于 RTC_TIMER_USER_BASE 且小于 RTC_TIMER_MAX； callback：超时回调函数，正常深睡会保存 data：超时回调入参 hour_start：开始的小时点，24 小时表示方式 hour_span：小时的跨度，即在多少小时内随机设置 RTC min：分钟 sec：秒数

6.10 set_rtc_by_week

原型	int set_rtc_by_week(char timer_id,rtc_timeout_cb_t callback, void *data,int day_week,int hour_start,int hour_span,int min,int sec);
功能	用于设置每周的某天的某个时间段的 RTC 定时。为了减少同批产品同时唤醒联网，产品开发时务必设置时间段，以减轻基站压力，提供通信成功率。例如，用户想每周的星期日的 15:00--20:00 随机做事，则设置： et_rtc_by_week(RTC_TIMER_USER1,user_rtc_timeout_cb,NULL,7,15,5,0,0)
参数	timer_id：查看 USER_RTC_TIMER_ID 大于等于 RTC_TIMER_USER_BASE 且小于 RTC_TIMER_MAX； callback：超时回调函数，正常深睡会保存 data：超时回调入参 day_week：取值 1-7，对应星期一到星期日 hour_start：开始的小时点，24 小时表示方式 hour_span：小时的跨度，即在多少小时内随机设置 RTC

	min：分钟 sec：秒数

6.11 timer 软定时

timer 软定时，是由操作系统提供的，不支持深睡唤醒，上电后必须重新设置。

6.12 xy_Timer_Create

原型	UINT32 xy_Timer_Create(char* name, UINT32 *pTimerId,UINT32 ulTimerTick_ms,const UINT32 uxAutoReload, xyTimerCallbackFunc callback_func);
功能	创建软定时器。由于芯片深睡时，会关闭 timer 时钟，进而无法深睡唤醒。对于需要深睡唤醒的定时功能，请使用 RTC 定时器。
参数	name：软定时名 pTimerId：软定时 ID，接口内部对其赋值 ulTimerTick_ms：定时时长，毫秒 uxAutoReload：是否为周期性定时器，建议设为 TIMER_NO_AUTO_RELOAD callback_func：超时回调函数

6.13 xy_Timer_Start

原型	UINT32 xy_Timer_Start(UINT32 *pTimerId);
功能	开启软定时计时。
参数	pTimerId：软定时器 ID

6.14 xy_Timer_Delete

原型	UINT32 xy_Timer_Delete(UINT32 *pTimerId);
功能	删除软定时。
参数	pTimerId：软定时器 ID

6.15 xy_Timer_Stop

原型	UINT32 xy_Timer_Stop(UINT32 *pTimerId);
功能	停止软定时计时。
参数	pTimerId：软定时器 ID

6.16 xy_Timer_Reset

原型	UINT32 xy_Timer_Reset(UINT32 *pTimerId);
功能	重新开始软定时计时。
参数	pTimerId：软定时器 ID

6.17 xy_get_working_time

原型	unsigned int xy_get_working_time();
功能	用于系统从上电开始已经过了多少秒了，常用于用户的运行时间异常监控。
返回值	秒数，系统已运行多少秒。

6.18 xy_get_last_work_offset

原型	int xy_get_last_work_offset();
功能	该接口用于查询上次正常深睡到现在过了多少秒，以供用户进行异常监控
返回值	秒数，即上次深睡到现在过了多少秒，若为 0，则表示上次未进行正常的深睡流程

7 AT 命令相关

7.1 register_app_at_urc

原型	void register_app_at_urc(char *at_prefix, inform_act_func func);
功能	注册主动上报的处理回调接口。
参数	at_prefix：AT 前缀 func：对应的回调函数

7.2 register_app_at_req

原型	void register_app_at_req(char *at_prefix, ser_req_func func);
功能	注册请求 AT 的处理回调接口。
参数	at_prefix：AT 前缀 func：对应的回调函数

7.3 drop_unused_urc

原型	int drop_unused_urc(char *buf);
功能	用于供用户选择过滤不关心的主动上报。
参数	buf：URC 主动上报字符串

7.4 at_ReqAndRsp_to_ps

原型	int at_ReqAndRsp_to_ps(char *req_at, char *info_fmt, void **pval,int timeout);
功能	用于发送 AT 请求给平台，并同步处理对应的应答结果。由于接口内部是根据 AT 前缀来匹配中间结果的，所以对于非标准化的 AT 请求中间结果不适用，需要按照主动上报方式进行解析处理，参考 register_app_at_urc 接口即可。 该接口只能用于获取 3GPP 相关的 AT 命令信息，不得用于平台业务的能力获取，如 onenet、CDP、socket 等。
参数	req_at：AT 请求字符串 info_fmt：参数格式化字符串，如"%d,%2d,%32s"，其中数字为参数存放的内存大小 pval：解析后参数值存放的内存数组 timeout：等待结果的超时时长，超时后上报 ATERR_WAIT_RSP_TIMEOUT
返回值	参见宏值 AT_XY_ERR

注意事项：

a) info_fmt 中对于字符串解析时，用户必须确保字符串的存放地址足够长，否则会造成没有‘\0’结束符，进而一旦使用 string 的 C 库函数，会存在越界的风险。例如：

```
char pri_dns[16] = {0};
char sec_dns[16] = {0};
void *p[] = {pri_dns,sec_dns};
at_ReqAndRsp_to_ps (at_str ,"%16s,%16s", at_buf, p);
```

b) 一旦输入的 at_buf 中，DNS 地址为 16 字节，就会造成 pri_dns 和 sec_dns 没有 ‘\0’ 结束符，进而极易造成内存访问越界，如 strlen(pri_dns)值可能很大。正确做法为：

```
char pri_dns[17] = {0};
char sec_dns[17] = {0};
void *p[] = {pri_dns,sec_dns};
at_ReqAndRsp_to_ps (at_str ,"%16s,%16s", at_buf, p);
```

7.5 at_parse_param

原型	int at_parse_param(char *fmt, char *buf, void **pval);
功能	用于解析 AT 命令参数字符串的每个参数值。
参数	Fmt：参数格式化字符串，如"%d,%2d,%1d"，其中数字为参数存放的内存大小 buf：AT 命令参数字符串，不含前缀内容 pval：解析后参数值存放的内存数组
返回值	参见宏值 AT_XY_ERR

注意事项：

a) info_fmt 中对于字符串解析时，用户必须确保字符串的存放地址足够长，否则会造成没有‘\0’结束符，进而一旦使用 string 的 C 库函数，会存在越界的风险。例如：

```
char pri_dns[16] = {0};
char sec_dns[16] = {0};
void *p[] = {pri_dns,sec_dns};
at_parse_param("%16s,%16s", at_buf, p);
```

b) 一旦输入的 at_buf 中，DNS 地址为 16 字节，就会造成 pri_dns 和 sec_dns 没有 ‘\0’ 结束符，进而极易造成内存访问越界，如 strlen(pri_dns)值可能很大。正确做法为：

```
char pri_dns[17] = {0};
char sec_dns[17] = {0};
void *p[] = {pri_dns,sec_dns};
at_parse_param("%16s,%16s", at_buf, p);
```

7.6 send_rsp_str_to_ext

原型	#define send_rsp_str_to_ext(a) {send_msg_2_atctl(AT_USER_SEND_ATSTR_TO_FARPS, a, strlen(a));}
功能	用于供用户发送 AT 请求的应答结果和 URC 主动上报给底板 MCU，常见于 at_user_req_serv 注册的具体 AT 请求的应答处理。该接口不可以在锁中断期间调用。

示例	rsp_cmd = xy_malloc(36);
	xy_printf("USER_SERV_MSG_1\n");
	snprintf(rsp_cmd, 36, "\r\n+INFO:USER_SERV_MSG_1\r\n");
	strcat(rsp_cmd, "\r\nOK\r\n");
	send_rsp_str_to_ext(rsp_cmd);
	xy_free(rsp_cmd);

7.7 send_urc_to_ext

原型	#define send_urc_to_ext(a)
功能	供用户任务发送主动上报给外部 MCU。

7.8 hexstr2bytes

原型	int hexstr2bytes(char* src, int src_len, char* dst, int dst_size);
功能	将十六进制 ASCII 码，转换为二进制码流，如：“AB235E3C”转化为 0XAB235E3C
参数	src：十六进制 ASCII 码字符串 src_len：十六进制 ASCII 码字符串长度 dst：转换后的二进制码流存放首地址 dst_size：目标内存长度
返回值	转换后的二进制码流有效长度，正常为 src_len/2 的值；-1 表示转换发生错误

7.9 bytes2hexstr

原型	int bytes2hexstr(char* src, signed long src_len, char* dst, signed long dst_size)
功能	将二进制码流，转换为十六进制 ASCII 码，如：0XAB235E3C 转化为 “AB235E3C”
参数	src：二进制码流存放首地址 src_len：二进制码流长度 dst：转换后的十六进制 ASCII 码字符串首地址 dst_size：目标内存长度
返回值	转换后的十六进制 ASCII 码字符串有效长度，正常为 src_len*2 的值；-1 表示转换发生错误

7.10 at_transparent_data_hook

原型	void at_transparent_data_hook(char *buf, unsigned long data_len, int fd);
功能	用户数据的传输过程中，在保证传输质量的前提下，通过无线的方式这组数据不发生任何形式的改变传输到了最终接收者。该接口在 AT 命令模式下，发”ATD*98\r\n” / ”ATD*99\r\n” 以及 “+++” 命令对状态的切换，达到用户数据透传的目的。用户可自行按照产品需求进行修改，具体细节请参考文档《海凌科 HLK-N10 平台扩展 AT 命令》。

7.11 switch_AT_cmd_hook

原型	void switch_AT_cmd_hook(int open_at_mode);
功能	用户数据的传输过程中，通过此 api 接口设置是否为 AT 命令模式，参数 1 为 AT 命令模式，log 打印：\r\nAT CMD MODE\r\n；参数 0 为数据透传模式，log 打印：\r\nCONNECTING\r\n；其他参数无效，保持原状。

版本历史

资料版本	日期	资料更新说明
V1.0	2020/7/18	初始版本