涂鸦智能 WIFI-SDK 说明

版本记录：

版本	编写/修订说明	修订人	修订日期	备注
1.0.0	创建文档	刘康	20160630	
1.0.1	优化部分流程图	刘康	20160715	
1.0.2	修改部分输入参数和返回值描述	胡赛	20160729	
1.0.3	1.增加低配网方式设置 2.更新底层驱动库和烧录 bin 文件	刘康	20170330	
1.0.4	1. 设置 wifi 配置接口说明 2. 增加 SDK 目录结构 3. 增加函数说明	刘康	20180412	

◆ 目录结构：

```
├─── app
│   ├─── tuya_common
│   │   └─── include
│   │       └─── system
│   └─── tuya_user
├─── bin
│   └─── upgrade
├─── extra_include
├─── include
├─── ld
├─── lib
└─── tools
```

（1）应用文件目录

app/tuya_user

（2）引用头文件目录

app/tuya_common/include 和 app/tuya_common/include/system

（3）编译生成 bin 文件目录

bin/upgrade

◆ 函数说明：

（1）设置固件标识名和版本回调函数，用于工厂生产固件校验

```
VOID set_firmware_tp(IN OUT CHAR *firm_name, IN OUT CHAR *firm_ver)
{
```

```
    strcpy(firm_name,APP_BIN_NAME);
    strcpy(firm_ver,USER_SW_VER);
    return;
}
```

（2）应用初始化回调，可设置 wifi 配置模式

```
VOID app_init(VOID)
{
   app_cfg_set(WCM_OLD,NULL);
}
```

（3）gpio 测试回调函数

```
BOOL gpio_func_test(VOID)
{
    return TRUE;
}
```

（4）应用入口函数

```
OPERATE_RET device_init(VOID)
{
    //框架初始化
    //PSM 扇区注册
    //定时器、按键功能创建等
}
```

1.框架基本接口(接入涂鸦云需使用)

1.1.1 获取 sdk 版本号

函数原型	CHAR *tuya_get_sdk_ver(VOID)
功能描述	获取 sdk 版本号
输入参数	无
输出参数	无
返回值	sdk 版本号,如"1.0.0"
备注	无

1.1.2 设置 wifi 配置模式

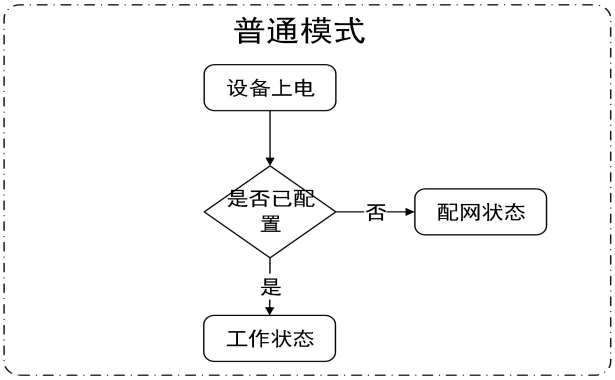
函数原型	VOID app_cfg_set(IN CONST WF_CFG_MTHD_SEL mthd, APP_PROD_CB callback)
功能描述	设置 wifi 配置模式
输入参数	mthd WCM_OLD 普通模式(do not have low power) WCM_LOW_POWER 低功耗模式(with low power) WCM_SPCL_MODE 特殊配网模式(special with low power) WCM_OLD_CPT 兼容模式

	callback
	产测回调函数，说明：
	(1) 普通模式，无产测回调，调用方法
	app_cfg_set(WCM_OLD, NULL);
	(2) 低功耗模式，必须设置产测回调
	app_cfg_set(WCM_LOW_POWER, prod_test);
	(3) 特殊配网模式，必须设置产测回调
	app_cfg_set(WCM_SPCL_MODE, prod_test);
	(4) 兼容模式
	app_cfg_set(WCM_OLD_CPT, prod_test);

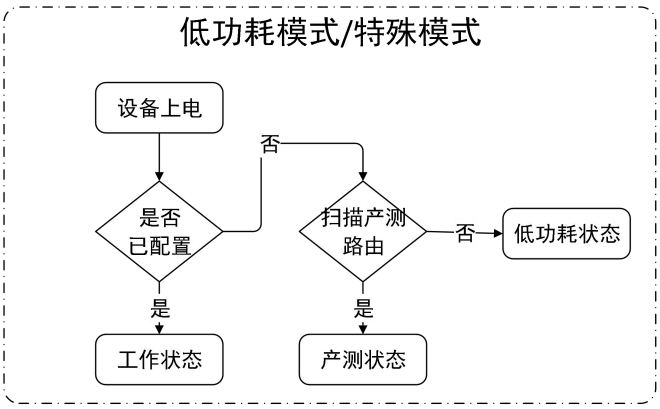
输出参数	无
返回值	无
备注	默认普通模式,必须在 app_init 中调用

各种工作模式状态切换流程：

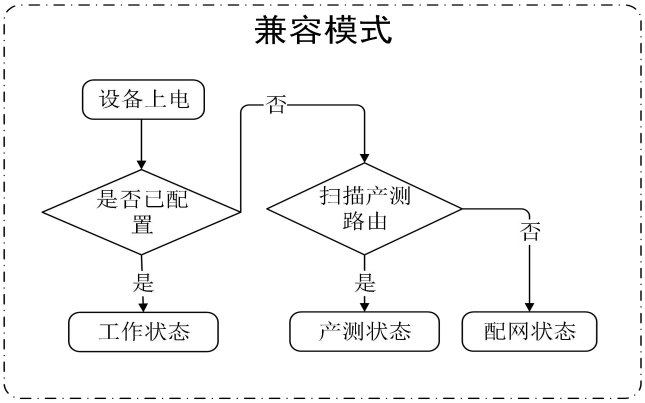
（1）普通模式



（2）低功耗模式/ 特殊配网模式



（3）兼容模式



1.1.3 设备初始化

函数原型	OPERATE_RET tuya_device_init(IN CONST CHAR *product_id, IN CONST SMART_FRAME_CB cb,CONST CHAR *app_ver);
功能描述	注册数据处理函数和应用版本号
输入参数	product_id 产品 ID cb 手机 App 命令回调函数指针, VOID (*)(SMART_CMD_E cmd,cJSON *root) <1> cmd 命令类型 0 表示局域网下发的命令 1 表示外网下发的命令 <2> root 命令数据 例，{"1":100,"2":200}，其中"1"和"2"为数据 ID (dpid)编号,100 和 200 为对应 dpid 的值 app_ver 应用版本号，如"1.0.0"
输出参数	无
返回值	详见返回值列表
备注	初始化设备，注册数据处理函数

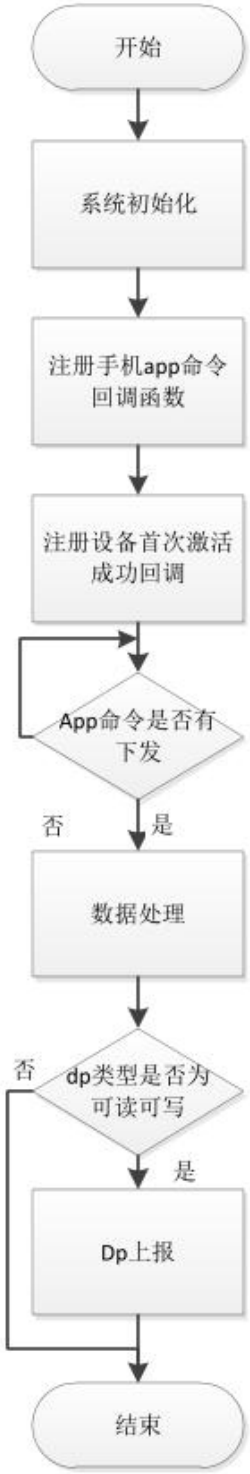


图 1-1 设备初始化流程

1.1.4 设备首次激活成功回调

函数原型	VOID tuya_active_reg(IN CONST SYN_DATA_CB callback)

功能描述	注册产品功能
输入参数	callback 回调函数(用于同步设备状态) 回调函数定义 typedef VOID(*SYN_DATA_CB)(VOID);
输出参数	无
返回值	无
备注	设备首次激活会调用 callback

1.1.5 获取设备 ID

函数原型	CHAR *tuya_get_devid(VOID)
功能描述	获取设备 ID
输入参数	无
输出参数	无
返回值	设备 ID
备注	无

1.1.6 数据上报

函数原型	OPERATE_RET tuya_obj_dp_report(IN CONST CHAR *data)
功能描述	数据上报
输入参数	data 上报的数据,例{"1":100,"2":200}
输出参数	无
返回值	参照返回值列表
备注	调用此接口固件会保存各 DP 的数据状态，如再次上传的 DP 数据与保存的状态相同则忽略上传，推荐使用该函数，可使 APP、云端、固件三方性能最佳

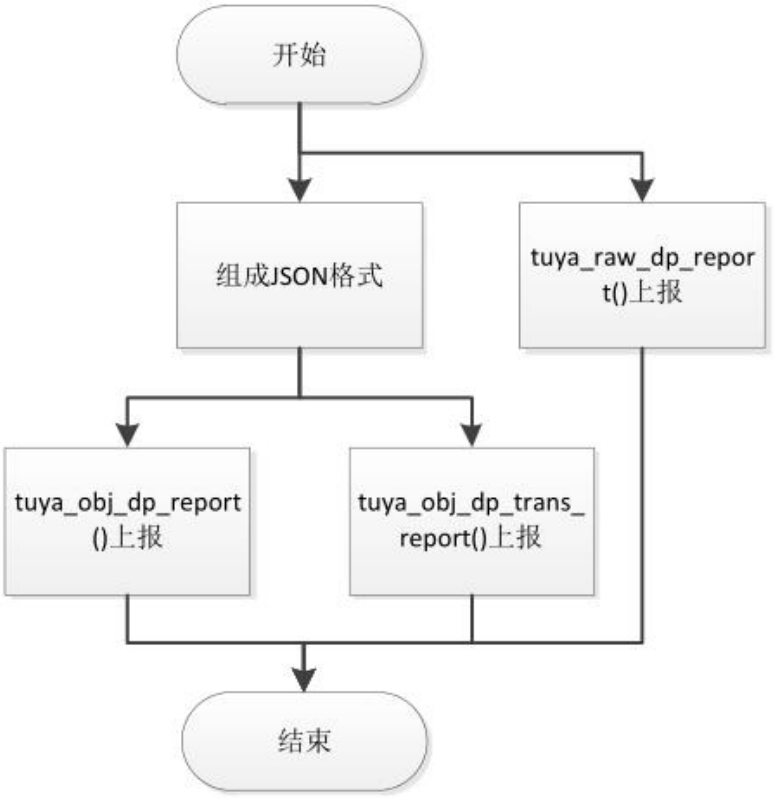


图 1-2 数据上报流程

1.1.7 数据上报(透传)

函数原型	OPERATE_RET tuya_obj_dp_trans_report(IN CONST CHAR *data)
功能描述	数据上报(透传)
输入参数	data 上报的数据,例{"1":100,"2":200}
输出参数	无
返回值	详见返回值列表
备注	数据透传到服务器，固件内部不做状态处理

1.1.8RAW 数据上报

函数原型	OPERATE_RET tuya_raw_dp_report(IN CONST BYTE dpid,IN CONST BYTE *data, IN CONST UINT len)
功能描述	RAW 数据上报
输入参数	dpid 功能点 dp 序号 data 原始二进制数据 len 数据长度
输出参数	无

返回值	详见返回值列表
备注	数据透传到服务器，固件内部不做状态处理

1.1.9 恢复出厂设置

函数原型	VOID tuya_dev_reset_factory(VOID)
功能描述	恢复出厂设置
输入参数	无
输出参数	无
返回值	无
备注	清除配网和设备信息 <1> 设备已激活，调用会将设备重置成 smartconfig 配网状态并清除激活信息 <2> 设备未激活，重复调用该函数会导致设备在 smartconfig、ap 配网状态来回切换

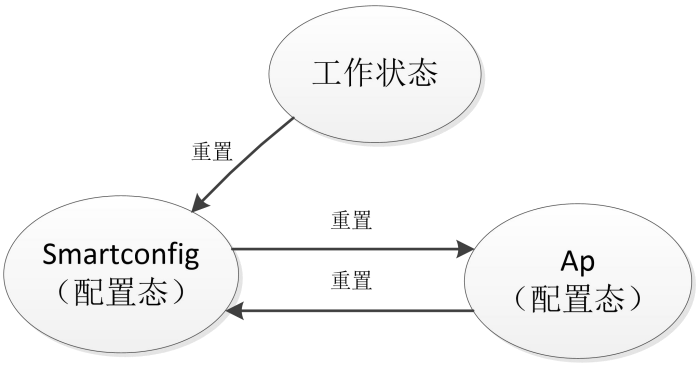


图 1-3 WiFi 状态切换图

1.1.10 恢复出厂设置并切换到指定状态

函数原型	VOID tuya_dev_reset_select(NW_CFG_MODE_E mode)
功能描述	恢复出厂设置并切换到指定状态
输入参数	mode 配置状态 <1> NW_SMART_CFG EZ 状态 <2> NW_AP_CFG AP 状态
输出参数	无
返回值	无
备注	清除配网和设备信息

1.1.11 获取设备状态

函数原型	GW_STAT_E tuya_get_gw_status(VOID)
功能描述	获取设备状态
输入参数	无

输出参数	无

返回值	typedef enum {
	UN_INIT = 0, // 未初始化，比如生产信息未写入
	PROD_TEST, // 产品产测模式
	UN_ACTIVE, // 未激活
	ACTIVE_RD, // 激活就绪态
	STAT_WORK, // 正常工作态
	}GW_STAT_E;

备注	无

1.1.12 获取 WIFI 工作状态

函数原型	GW_WIFI_STAT_E tuya_get_wf_status(VOID)
功能描述	获取 WIFI 工作状态
输入参数	无
输出参数	无

返回值	typedef enum {
	STAT_LOW_POWER = 0, //低功耗状态(射频处于关闭状态)
	STAT_UNPROVISION = 0, //EZ 状态(未配置)
	STAT_AP_STA_UNCONN, //AP 状态(未配置)
	STAT_AP_STA_CFG_UNC, //AP 和 STA 混合,STA 未连接状态(保留)
	STAT_AP_STA_CONN, //AP 和 STA 混合,STA 已连接状态(保留)
	STAT_STA_UNCONN, //STA 未连接状态
	STAT_STA_CONN, //STA 已连接状态
	}GW_WIFI_STAT_E;

备注	无

1.1.13 获取 WIFI 信号强度

函数原型	OPERATE_RET tuya_get_wf_rssi(IN UCHAR *ssid, OUT CHAR *rssi)
功能描述	获取 WIFI 信号强度
输入参数	ssid 热点名称
输出参数	rssi 信号强度(单位 dbm)
返回值	详见返回值列表
备注	扫描过程会阻塞，请勿在初始化函数中使用

1.1.14 获取云连接状态

函数原型	BOOL tuya_get_cloud_stat(VOID)

功能描述	获取云连接状态
输入参数	无
输出参数	无
返回值	TRUE 已连接 FALSE 未连接
备注	无

1.1.15 获取设备升级状态

函数原型	BOOL tuya_get_ug_stat(VOID)
功能描述	获取设备升级状态
输入参数	无
输出参数	无
返回值	TRUE 升级中 FALSE 未升级
备注	无

1.1.16 获取本地时间

函数原型	OPERATE_RET tuya_get_local_time(OUT struct tm *st_time)
功能描述	获取本地时间
输入参数	无
输出参数	st_time 时间结构体
返回值	参照返回值列表
备注	设备激活后才能使用

1.1.17 注册 PSM 模块名称和扇区名称

函数原型	OPERATE_RET tuya_psm_register_module(IN CONST CHAR *module_name, IN CONST CHAR *partition_key)
功能描述	注册 PSM 模块名称和扇区名称
输入参数	module_name 模块名称 partition_key 扇区名称
输出参数	无
返回值	OPRT_PSM_E_EXIST 扇区已注册, OPRT_OK 成功，其它为失败
备注	每个扇区最多保存 4K 数据

1.1.18 写入 PSM 数据

函数原型	OPERATE_RET tuya_psm_set_single(IN CONST CHAR *module,IN CONST CHAR *variable,IN CONST CHAR *value)
功能描述	写入 PSM 数据
输入参数	module_name 模块名称 variable 变量名 value 变量值
输出参数	无
返回值	参照返回值列表
备注	只支持字符串类型

1.1.19 读取 PSM 数据

函数原型	OPERATE_RET tuya_psm_get_single(IN CONST CHAR *module, IN CONST CHAR *variable,OUT CHAR *value,IN CONST unsigned max_len)
功能描述	读取 PSM 数据
输入参数	module_name 模块名称 variable 变量名 max_len 缓冲区大小
输出参数	value 变量值
返回值	参照返回值列表
备注	只支持字符串类型

2.按键接口

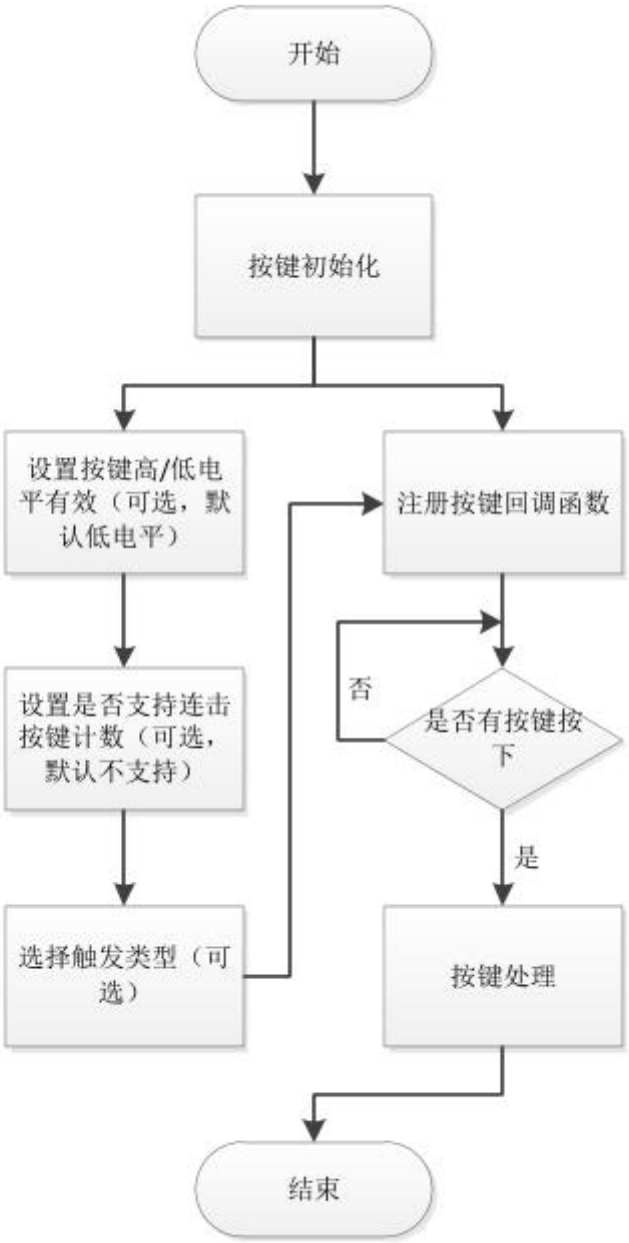


图 2-1 按键使用流程

2.1.1 按键初始化

函数原型	OPERATE_RET tuya_kb_init(VOID)
功能描述	按键初始化
输入参数	无

输出参数	无
返回值	参照返回值列表
备注	无

2.1.2 设置是否支持连击按键计数(即 SEQ_KEY 类型)

函数原型	VOID tuya_set_kb_seq_enable(IN BOOL enable)
功能描述	设置是否支持连击按键计数即 SEQ_KEY 类型
输入参数	enable <1> TRUE 支持连击按键计数事件，可触发 SEQ_KEY 类型事件 <2> FALSE 不支持连击按键计数，仅支持触发 NORMAL_KEY、LONG_KEY 事件
输出参数	无
返回值	无
备注	如果不调用该接口则默认所有的按键处理均支持连击按键计数功能

2.1.3 设置按键高电平有效

函数原型	VOID tuya_set_kb_detect_high_valid(BOOL is_high)
功能描述	设置按键高电平有效
输入参数	is_high <1> TRUE 高电平有效 <2> FALSE 低电平有效
输出参数	无
返回值	无
备注	无

2.1.4 设置按键触发类型

函数原型	VOID tuya_set_kb_trig_type(IN CONST INT gpio_no, IN CONST KEY_TRIGGER_TP_E trig_ty,IN CONST BOOL down_trig_cont)
功能描述	按键初始化
输入参数	gpio_no:0-16 分别对应 IO0-IO16 trig_ty: <1> KEY_UP_TRIG 弹起触发 <2> KEY_DOWN_TRIG 按下触发，仅支持 NORMAL_KEY 以及 LONG_KEY down_trig_cont: <1> 仅当 trig_ty == KEY_DOWN_TRIG 时有效 <2> TURE 如果用户一直按着不放，则间隔 400ms 会再次出发 NORMAL_KEY 事件 <3> FALSE 用户一直按着不放，仅触发一次

输出参数	无
返回值	无
备注	默认如果不调用此函数，则按键事件均为弹起触发类型

2.1.5 注册按键回调函数

函数原型	OPERATE_RET tuya_kb_reg_proc(IN CONST INT gpio_no, IN CONST INT long_key_time,IN CONST KEY_CALLBACK call_back);
功能描述	注册按键回调函数
输入参数	gpio_no:0-16 分别对应 IO0-IO16 long_key_time:长按键触发时间(ms)，如设置为 0 则屏蔽长按键 call_back 按键回调函数, VOID(*)(INT gpio_no,PUSH_KEY_TYPE_E type,INT cnt) gpio_no GPIO 序号 type 按键类型 <1> NORMAL_KEY 普通按键 <2> SEQ_KEY 说明:连击按键,仅对弹起触发的按键类型有效，比如快速按下两次回调参数 type == SEQ_KEY,cnt == 2(cnt 表示连续点击次数) <3> LONG_KEY 长按按键 cnt 按键次数
输出参数	无
返回值	参照返回值列表
备注	无

3.LED 接口

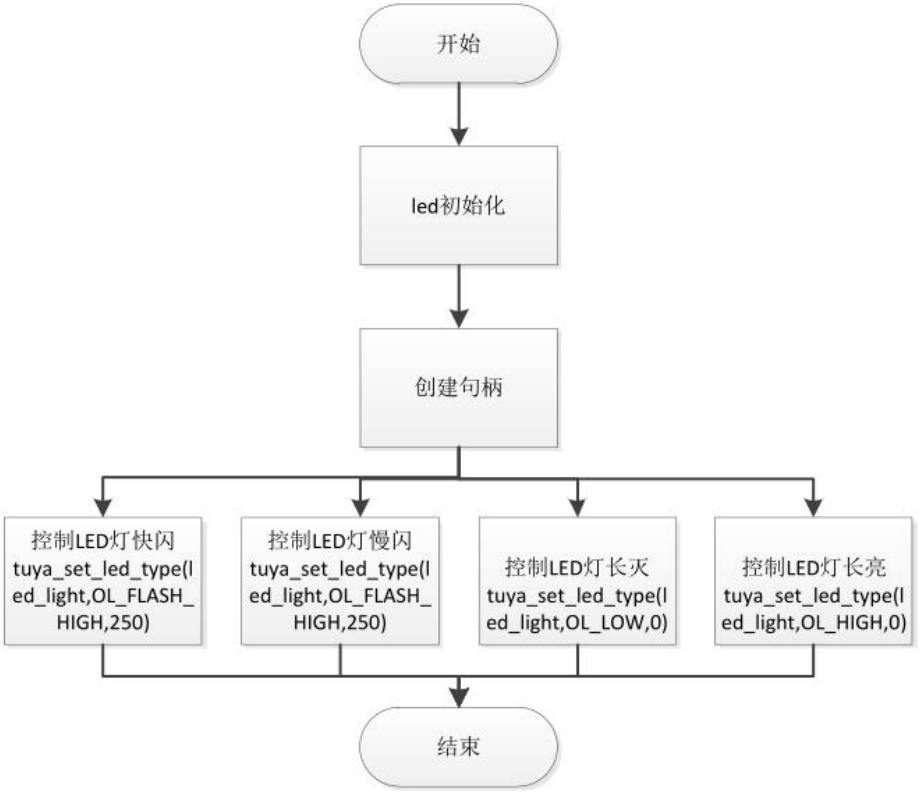


图 3-1 LED 使用流程

3.1.1 创建 LED 控制句柄

函数原型	OPERATE_RET tuya_create_led_handle(IN CONST INT gpio_no, OUT LED_HANDLE *handle)
功能描述	创建 LED 控制句柄
输入参数	gpio_no　GPIO 序号
输出参数	handle　　LED 句柄
返回值	参照返回值列表
备注	无

3.1.2 LED 控制

函数原型	VOID tuya_set_led_type(IN CONST LED_HANDLE handle,IN CONST LED_LT_E type, IN CONST USHORT flh_mstime)
功能描述	按键初始化
输入参数	handle　LED 句柄

	type 控制类型 <1> OL_LOW LED 低电平 <2> OL_HIGH LED 高电平 <3> OL_FLASH_LOW LED 低电平闪烁 <4> OL_FLASH_HIGH LED 高电平闪烁 flh_mstime 闪烁间隔时间
输出参数	无
返回值	无
备注	无

4.UART 接口

图 4-1 串口使用流程

4.1.1 打印串口号设置

函数原型	void print_port_init(UART_Port uart_no)
功能描述	打印串口号设置
输入参数	uart_no 串口号
输出参数	无

返回值	无
备注	默认打印波特率 74880 如 uart_io==UART1，则打印信息由 IO2 端口输出，tysdk 默认所有的打印信息由 IO2 端口输出，用户在编写应用代码时可通过该接口重配置

4.1.2 打印串口参数设置

函数原型	void print_port_full_init(UART_Port uart_no, UART_BautRate bit_rate, UART_WordLength data_bits, UART_ParityMode parity,UART_StopBits stop_bits);
功能描述	打印串口参数设置
输入参数	uart_no 串口号 bit_rate 波特率(300-3686400) data_bits 数据位 parity 奇偶校验位 stop_bits 停止位
输出参数	无
返回值	无
备注	如 uart_io==UART1，则打印信息由 IO2 端口输出，tysdk 默认所有的打印信息由 IO2 端口输出，用户在编写应用代码时可通过该接口重配置

4.1.3 读取 GPIO 电平

函数原型	INT tuya_read_gpio_level(USHORT gpio_no)
功能描述	读取 GPIO 电平
输入参数	gpio_no:0-16 分别对应 IO0-IO16
输出参数	无
返回值	0 低电平 1 高电平
备注	无

4.1.4 UART0 串口初始化配置

函数原型	void user_uart_full_init(UART_BautRate bit_rate,UART_WordLength data_bits,\ UART_ParityMode parity,UART_StopBits stop_bits);
功能描述	UART0 串口初始化配置
输入参数	bit_rate 波特率(300-3686400) data_bits 数据位 parity 奇偶校验位 stop_bits 停止位
输出参数	无
返回值	无

备注	如果采用该接口初始化 UART0，IO15 电路需做特殊处理，具体可见乐鑫原厂文档。采用该端口进行串口通讯的好处是，可以避免 8266 启动后默认信息输出对用户控制板的数据干扰

4.1.5 UART0 串口初始化配置(默认 UART0)

函数原型	void user_uart_raw_full_init(UART_BautRate bit_rate,UART_WordLength data_bits,\ UART_ParityMode parity,UART_StopBits stop_bits);
功能描述	UART0 串口初始化配置(默认 UART0)
输入参数	bit_rate 波特率(300-3686400) data_bits 数据位 parity 奇偶校验位 stop_bits 停止位
输出参数	无
返回值	无
备注	esp8266 默认 UART0 的收发 IO 端口

4.1.6 UART0 串口初始化（esp8266 的收发端口设置为 I015:TX IO13:RX）

函数原型	void user_uart_init(UART_BautRate bit_rate)
功能描述	UART0 串口初始化（esp8266 的收发端口设置为 I015:TX IO13:RX）
输入参数	bit_rate 波特率(300-3686400)
输出参数	无
返回值	无
备注	默认 data_bits==8,parity==无,stop_bits==0 如果采用该接口初始化 UART0，IO15 电路需做特殊处理，具体可见乐鑫原厂文档。采用该端口进行串口通讯的好处是，可以避免 8266 启动后默认信息输出对用户控制板的数据干扰。

4.1.7 UART0 串口初始化(esp8266 的收发端口设置为 IO3:RX IO1:TX)

函数原型	void user_uart_raw_init(UART_BautRate bit_rate)
功能描述	UART0 串口初始化(esp8266 的收发端口设置为 IO3:RX IO1:TX)
输入参数	bit_rate 波特率(300-3686400)
输出参数	无
返回值	无
备注	默认 data_bits==8,parity==无,stop_bits==0, esp8266 默认 UART0 的收发 IO 端口

4.1.8 读串口缓冲数据大小

函数原型	uint16 user_uart_read_size(void)
功能描述	读串口缓冲数据大小
输入参数	无
输出参数	无
返回值	读到的缓冲数据长度
备注	无

4.1.9 读取串口数据

函数原型	uint16 user_uart_read_data(uint8 *out,uint16 out_len)
功能描述	读取串口数据
输入参数	out_len 缓冲大小
输出参数	out 读取的数据
返回值	读取的数据长度
备注	无

4.1.10 写串口数据

函数原型	void user_uart_write_data(uint8 *in,uint16 in_len)
功能描述	写串口数据
输入参数	in 要写的数据 in_len 数据长度
输出参数	无
返回值	无
备注	无

5.定时器接口

5.1.1 添加一个系统定时器

函数原型	OPERATE_RET sys_add_timer(IN CONST P_TIMER_FUNC pTimerFunc, IN CONST PVOID pTimerArg, OUT TIMER_ID *p_timerID)
功能描述	添加一个系统定时器
输入参数	pTimerFunc: 定时器处理函数 pTimerArg: 定时器处理参数
输出参数	p_timerID：定时器 ID 编号
返回值	参照返回值列表

备注	无

5.1.2 停止一个定时器

函数原型	OPERATE_RET sys_stop_timer(IN CONST TIMER_ID timerID)
功能描述	停止一个定时器
输入参数	timerID：定时器 ID 编号
输出参数	无
返回值	参照返回值列表
备注	无

5.1.3 定时器是否运行

函数原型	BOOL IsThisSysTimerRun(IN CONST TIMER_ID timerID)
功能描述	定时器是否运行
输入参数	timerID：定时器 ID 编号
输出参数	无
返回值	TRUE 正在运行 FALSE 未运行
备注	无

5.1.4 启动一个定时器

函数原型	OPERATE_RET sys_start_timer(IN CONST TIMER_ID timerID, IN CONST TIME_MS timeCycle, IN CONST TIMER_TYPE timer_type)
功能描述	启动一个定时器
输入参数	timerID：定时器 ID 编号 timeCycle: 定时周期(单位毫秒) timer_type: 定时器类型 <1> TIMER_ONCE 单次执行 <2> TIMER_CYCLE 循环执行
输出参数	无
返回值	参照返回值列表
备注	无

6.系统层接口

6.1.1 系统休眠

函数原型	VOID SystemSleep(IN CONST TIME_MS msTime)
功能描述	系统休眠
输入参数	msTime: 休眠时间(单位毫秒)
输出参数	无
返回值	无
备注	无

6.1.2 系统重启

函数原型	VOID SystemReset(VOID)
功能描述	系统重启
输入参数	无
输出参数	无
返回值	无
备注	无

6.1.3 比较两个字符串是否相等

函数原型	int strcasecmp(const char *s1, const char *s2)
功能描述	比较两个字符串是否相等
输入参数	s1: 字符串 s1 s2: 字符串 s2
输出参数	无
返回值	0 字符串相等　　非 0 字符串不等
备注	无

6.1.4 ASCII 码转 HEX

函数原型	void ascs2hex(unsigned char *hex,unsigned char *ascs,int srclen)
功能描述	ASCII 码转 HEX
输入参数	ascs: ASCII 码 srclen: ASCII 码长度

输出参数	hex: 转换后的 hex 数据
返回值	无
备注	无

7.任务接口

7.1.1 任务创建

函数原型	OPERATE_RET CreateAndStart(OUT THRD_HANDLE *pThrdHandle,\ IN CONST P_THRD_FUNC pThrdFunc,\ IN CONST PVOID pThrdFuncArg,\ IN CONST STACK_SIZE stack_size,\ IN CONST TRD_PRI pri,\ IN CONST CHAR *thrd_name);
功能描述	任务创建
输入参数	pThrdHandle 任务句柄 pThrdFunc 任务处理函数 pThrdFuncArg 任务参数 stack_size 指定任务堆栈大小 pri 任务优先级 thrd_name 任务名称
输出参数	无
返回值	参照返回值列表
备注	无

7.1.2 任务删除

函数原型	OPERATE_RET ThrdJoin(IN CONST THRD_HANDLE thrdHandle, OUT VOID **ppThrdRet);
功能描述	任务删除
输入参数	thrdHandle 任务句柄
输出参数	ppThrdRet 任务退出码
返回值	参照返回值名称
备注	无

8.信号量接口

8.1.1 信号量创建

函数原型	SEM_HANDLE CreateSemaphore(VOID)
功能描述	信号量创建
输入参数	无
输出参数	无
返回值	信号量句柄
备注	无

8.1.2 信号量初始化

函数原型	OPERATE_RET InitSemaphore(IN CONST SEM_HANDLE semHandle, IN CONST UINT semCnt, IN CONST UINT sem_max)
功能描述	信号量初始化
输入参数	semHandle 信号量句柄 semCnt 信号量初始值 sem_max 信号量最大值
输出参数	无
返回值	参照返回值名称
备注	无

8.1.3 信号量发送

函数原型	OPERATE_RET PostSemaphore(IN CONST SEM_HANDLE semHandle)
功能描述	信号量发送
输入参数	semHandle 信号量句柄
输出参数	无
返回值	参照返回值名称
备注	无

8.1.4 信号量接收

函数原型	OPERATE_RET WaitSemaphore(IN CONST SEM_HANDLE semHandle)
功能描述	信号量接收

输入参数	semHandle 信号量句柄
输出参数	无
返回值	参照返回值名称
备注	无

9.互斥锁接口

9.1.1 互斥锁创建

函数原型	OPERATE_RET CreateMutexAndInit(OUT MUTEX_HANDLE *pMutexHandle)
功能描述	互斥锁创建
输入参数	无
输出参数	pMutexHandle 互斥锁句柄
返回值	参照返回值名称
备注	无

9.1.2 互斥锁锁定

函数原型	OPERATE_RET MutexLock(IN CONST MUTEX_HANDLE mutexHandle)
功能描述	互斥锁锁定
输入参数	pMutexHandle 互斥锁句柄
输出参数	无
返回值	参照返回值名称
备注	无

9.1.3 互斥锁解锁

函数原型	OPERATE_RET MutexUnLock(IN CONST MUTEX_HANDLE mutexHandle)
功能描述	互斥锁解锁
输入参数	pMutexHandle 互斥锁句柄
输出参数	无
返回值	参照返回值名称
备注	无

10.消息队列接口

10.1.1 消息队列创建

函数原型	OPERATE_RET CreateMsgQueAndInit(OUT MSG_QUE_HANDLE *pMsgQueHandle)
功能描述	消息队列创建
输入参数	无
输出参数	pMsgQueHandle 消息管理结构句柄
返回值	参照返回值名称
备注	无

10.1.2 消息添加

函数原型	OPERATE_RET AddMsgNodeToQueue(IN CONST MSG_QUE_HANDLE msgQueHandle, IN CONST MSG_ID msgID, IN CONST P_MSG_DATA pMsgData, IN CONST MSG_DATA_LEN msgDataLen, IN CONST MSG_TYPE msgType)
功能描述	消息添加
输入参数	pMsgQueHandle 消息管理结构句柄 msgID 消息 ID pMsgData 消息数据 msgDataLen 消息数据长度 msgType 消息类型
输出参数	无
返回值	参照返回值名称
备注	先进先出方式，系统顺序执行消息

10.1.3 获取指定 ID 的消息节点

函数原型	OPERATE_RET GetMsgNodeFromQueue(IN CONST MSG_QUE_HANDLE msgQueHandle, IN CONST MSG_ID msgID, OUT P_MSG_LIST *ppMsgListNode)
功能描述	获取指定 ID 的消息节点
输入参数	pMsgQueHandle 消息管理结构句柄 msgID 消息 ID
输出参数	pMsgListNode 消息节点
返回值	参照返回值名称
备注	无

10.1.4 获取首先入链的消息节点

函数原型	OPERATE_RET GetFirstMsgFromQueue(IN CONST MSG_QUE_HANDLE msgQueHandle, OUT P_MSG_LIST *ppMsgListNode)
功能描述	获取首先入链的消息节点
输入参数	pMsgQueHandle 消息管理结构句柄
输出参数	pMsgListNode 消息节点
返回值	参照返回值名称
备注	无

10.1.5 获取队列中的消息节点总数

函数原型	OPERATE_RET GetMsgNodeNum(IN CONST MSG_QUE_HANDLE msgQueHandle, OUT PINT pMsgNodeNum)
功能描述	获取队列中的消息节点总数
输入参数	pMsgQueHandle 消息管理结构句柄
输出参数	pMsgNodeNum 消息节点总数
返回值	参照返回值名称
备注	无

10.1.6 从链中删除消息节点并释放消息节点内存

函数原型	OPERATE_RET DelAndFreeMsgNodeFromQueue(IN CONST MSG_QUE_HANDLE msgQueHandle, IN CONST P_MSG_LIST pMsgListNode)
功能描述	从链中删除消息节点并释放消息节点内存
输入参数	pMsgQueHandle 消息管理结构句柄 pMsgListNode 删除的消息节点
输出参数	无
返回值	参照返回值名称
备注	无

10.1.7 释放消息队列所占用内存

函数原型	OPERATE_RET ReleaseMsgQue(IN CONST MSG_QUE_HANDLE msgQueHandle)
功能描述	释放消息队列所占用内存
输入参数	pMsgQueHandle 消息管理结构句柄
输出参数	无

返回值	参照返回值名称
备注	无

10.1.8 投递消息

函数原型	OPERATE_RET PostMessage(IN CONST MSG_QUE_HANDLE msgQueHandle, IN CONST MSG_ID msgID, IN CONST P_MSG_DATA pMsgData, IN CONST MSG_DATA_LEN msgDataLen)
功能描述	投递消息
输入参数	msgQueHandle 消息处理句柄 msgID 消息 ID pMsgData 消息数据 msgDataLen 消息数据长度
输出参数	无
返回值	参照返回值名称
备注	递送一个消息至模块(消息先进先执行)

10.1.9 投递紧急消息

函数原型	OPERATE_RET PostInstancyMsg(IN CONST MSG_QUE_HANDLE msgQueHandle,\ IN CONST MSG_ID msgID, IN CONST P_MSG_DATA pMsgData,\ IN CONST MSG_DATA_LEN msgDataLen)
功能描述	投递紧急消息
输入参数	msgQueHandle 消息处理句柄 msgID 消息 ID pMsgData 消息数据 msgDataLen 消息数据长度
输出参数	无
返回值	参照返回值名称
备注	无

10.1.10 等待消息

函数原型	OPERATE_RET WaitMessage(IN CONST MSG_QUE_HANDLE msgQueHandle, OUT P_MSG_LIST *ppMsgListNode)
功能描述	等待消息
输入参数	msgQueHandle 消息处理句柄
输出参数	ppMsgListNode 消息节点

返回值	参照返回值名称
备注	WaitMessage 成功需调用，消息处理完后需调用 DelAndFreeMsgNodeFromQueue 释放消息

附录

1.返回值列表

返回值宏定义	返回值	描述
OPRT_OK	0	执行成功
OPRT_COM_ERROR	1	通用错误
OPRT_INVALID_PARM	2	无效的入参
OPRT_MALLOC_FAILED	3	内存分配失败
OPRT_INIT_MUTEX_ATTR_FAILED	4	初始化同步属性失败
OPRT_SET_MUTEX_ATTR_FAILED	5	设置同步属性失败
OPRT_DESTROY_MUTEX_ATTR_FAILED	6	销毁同步属性失败
OPRT_INIT_MUTEX_FAILED	7	初始化互斥量失败
OPRT_MUTEX_LOCK_FAILED	8	互斥量加锁失败
OPRT_MUTEX_TRYLOCK_FAILED	9	互斥量尝试加锁失败
OPRT_MUTEX_LOCK_BUSY	10	互斥量忙
OPRT_MUTEX_UNLOCK_FAILED	11	互斥量解锁失败
OPRT_MUTEX_RELEASE_FAILED	12	互斥量释放失败
OPRT_INIT_SEM_FAILED	13	初始化信号量失败
OPRT_WAIT_SEM_FAILED	14	等待信号量失败
OPRT_POST_SEM_FAILED	15	释放信号量失败
OPRT_THRD_STA_UNVALID	16	线程状态非法
OPRT_THRD_CR_FAILED	17	线程创建失败
OPRT_THRD_JOIN_FAILED	18	线程 JOIN 函数调用失败
OPRT_THRD_SELF_CAN_NOT_JOIN	19	自身线程不能调用 JOIN 函数
OPRT_TIMERID_EXIST	20	定时器 ID 已存在
OPRT_TIMERID_NOT_FOUND	21	未找到指定定时器 ID
OPRT_TIMERID_UNVALID	22	定时器 ID 非法
OPRT_GET_IDLE_TIMERID_ERROR	23	获取空闲定时器 ID 错误
OPRT_MSG_NOT_FOUND	24	未找到指定消息
OPRT_MSG_LIST_EMPTY	25	消息链空
OPRT_PSM_FLH_RET_ERR	26	PSM 擦除失败
OPRT_PSM_FLH_TM_ERR	27	PSM 擦除超时
OPRT_PSM_E_INVAL	28	
OPRT_PSM_E_IO	29	

OPRT_PSM_E_EXIST	30	
OPRT_PSM_E_NOENT	31	
OPRT_PSM_FAIL	32	
OPRT_PSM_E_NOSPC	33	
OPRT_PSM_E_METADATA_CRC	34	
OPRT_PSM_E_CRC	35	
OPRT_WIFI_SCAN_FAIL	36	
OPRT_WF_MAC_SET_FAIL	37	
OPRT_WF_CONN_FAIL	38	
OPRT_WF_NW_CFG_FAIL	39	
OPRT_SET_SOCK_ERR	41	
OPRT_SOCK_CONN_ERR	42	
OPRT_CR_MUTEX_ERR	43	
OPRT_CR_TIMER_ERR	44	
OPRT_CR_THREAD_ERR	45	
OPRT_BUF_NOT_ENOUGH	46	
OPRT_URL_PARAM_OUT_LIMIT	47	
OPRT_HTTP_OS_ERROR	48	
OPRT_HTTP_PR_REQ_ERROR	49	
OPRT_HTTP_SD_REQ_ERROR	50	
OPRT_HTTP_RD_ERROR	51	
OPRT_HTTP_AD_HD_ERROR	52	
OPRT_HTTP_GET_RESP_ERROR	53	
OPRT_HTTP_AES_INIT_ERR	54	
OPRT_HTTP_AES_OPEN_ERR	55	
OPRT_HTTP_AES_SET_KEY_ERR	56	
OPRT_HTTP_AES_ENCRYPT_ERR	57	
OPRT_TY_WS_PART_ERR	58	
OPRT_CR_CJSON_ERR	59	
OPRT_PSM_SET_ERROR	60	
OPRT_PSM_GET_ERROR	61	
OPRT_CJSON_PARSE_ERR	62	
OPRT_CJSON_GET_ERR	63	
OPRT_CR_HTTP_URL_H_ERR	64	
OPRT_HTTPS_HANDLE_FAIL	65	
OPRT_HTTPS_RESP_UNVALID	66	
OPRT_MEM_PARTITION_EMPTY	67	
OPRT_MEM_PARTITION_FULL	68	
OPRT_MEM_PARTITION_NOT_FOUND	69	
OPRT_CR_QUE_ERR	70	
OPRT_SND_QUE_ERR	71	
OPRT_NOT_FOUND_DEV	72	
OPRT_NOT_FOUND_DEV_DP	73	

OPRT_DP_ATTR_ILLEGAL	74	
OPRT_DP_TYPE_PROP_ILLEGAL	75	
OPRT_DP_REPORT_CLOUD_ERR	76	
OPRT_NO_NEED_SET_PRODINFO	77	
OPRT_NW_INVALID	78	
OPRT_SELECT_ERR	79	
OPRT_SELECT_TM	80	
OPRT_SEND_ERR	81	
OPRT_DEV_NOT_BIND	82	
OPRT_FW_UG_FAILED	83	
OPRT_VER_FMT_ERR	84	
OPRT_FW_NOT_EXIST	85	
OPRT_SEM_CR_ERR	86	
OPRT_SELECT_TIMEOUT	87	
OPRT_GW_MQ_OFFLILNE	88	
OPRT_NOT_SPT_CLX_DP	89	
OPRT_RECV_ERR	90	
OPRT_UG_PKG_NOT_ENOUGH	91	
OPRT_SCMA_INVALID	92	
OPRT_PRODECT_KEY_NULL	93	
OPRT_DEVICE_VER_NULL	94	
OPRT_MSG_OUT_OF_LMT	95	
OPRT_NOT_FOUND_AUTH_SSID	96	
OPRT_SOCKET_FAULT	97	
OPRT_MQ_PUBLISH_TIMEOUT	98	
OPRT_GW_NOT_EXIST	99	
OPRT_GW_SCHEMA_SIZE_LMT_OUT	100	
OPRT_DEV_DP_CNT_INVALID	101	
OPRT_TOKEN_OVERTIME OPRT_CR_TIMER_FAILED OPRT_PUB_NO_PERMISSION	102 103 104	

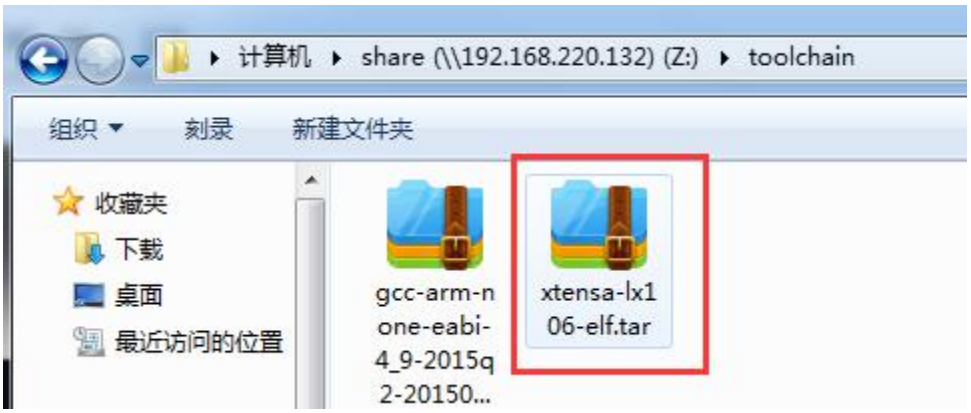
2.编译环境搭建

(1) 交叉编译器下载

可先在 windows 下下载完交叉编译器(也可在 ununtu 下直接下载)，下载地址为：http://bbs.espressif.com/viewtopic.php?f=57&t=2

(2) 交叉编译器安装

将在 windows 下下载完成后的交叉编译器 xtensa-lx106-elf.tar.bz2 复制至共享目录下,我是专门新建了一个 toolchain 目录放置一系列交叉编译器安装文件



(3) 解压交叉编译器

在 linux 终端下切换至/home/share/samba/toolchain 目录;解压 xtensa-lx106-elf.tar.bz2 至/usr/bin 目录下,输入命令:

sudo tar -jxvf xtensa-lx106-elf.tar.bz2 -C /usr/bin

(4)修改目录属性

这一步非常重要,不然编译的时候会提示无法编译

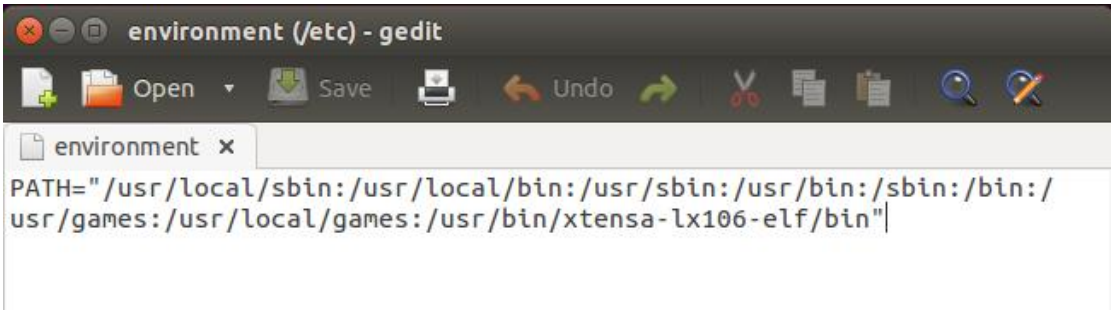
sudo chmod 777 -R /usr/bin/xtensa-lx106-elf

(5)编辑环境变量

打开/etc/environment 文件.输入命令:

```
sudo gedit /etc/environment
```

修改/etc/environment 文件(我习惯编辑这个文件),在文件末尾添加:/usr/bin/xtensa-lx106-elf/bin 至当前环境变量下



(6)执行 source /etc/environment

使环境变量生效.生效后,可以 echo $PATH 查看是否设置成功

3.应用编译方法

1.进入 tysdk_for_esp8266/app 目录

(1)编译有日志输出的固件，例如 sample_pk，版本号 1.0.0

sh build_app.sh sample_pk 1.0.0

(2)编译无日志输出的固件，例如 sample_pk，版本号 1.0.0

sh build_app_release.sh sample_pk 1.0.0

2.编译生成 bin 文件说明

编译生成 bin 文件在/bin/upgrade 目录中

(1)sample_pk(1)_1.0.0.bin 为可下载的固件文件，从 0x1000 地址烧录

(2) sample_pk_ug_1.0.0.bin 为升级文件，用作固件升级使用

3.固件烧录说明

(1)文件烧录地址

BOOT 文件	boot_v1.4(b1).bin	0x0000
应用文件	sample_pk(1)_1.0.0.bin	0x1000
系统参数文件	esp_init_data_default.bin	0xfc000
填充文件	blank.bin	0xfe000

(2)建议文件打包成 1M 的 target 文件，适用 1M/2M/4M Byte 的 flash，只烧录 1M 大小即可