设备接入 SDK 集成开发指南

版本号：V1.0

2017 年 6 月发布

中国电信股份有限公司上海研究院

目录

1 SDK 说明

接入 SDK 主要包含两个部分，即 CloudKitLib (src/CloudKit)和 Demo (src/Demo)示例代码。其中，CloudKitLib 为对平台接入协议和配网协议的封装库，以源码形式提供。用户将 CloudKitLib 源码集成到自己的工程中，并按流程调用接口，即可完成接入。

1.1 CloudKitLib 说明

CloudKitLib 将平台的接入协议和配网协议完全封装好，用户在使用时根据需要调用对应的接口即可，整体结构如下：

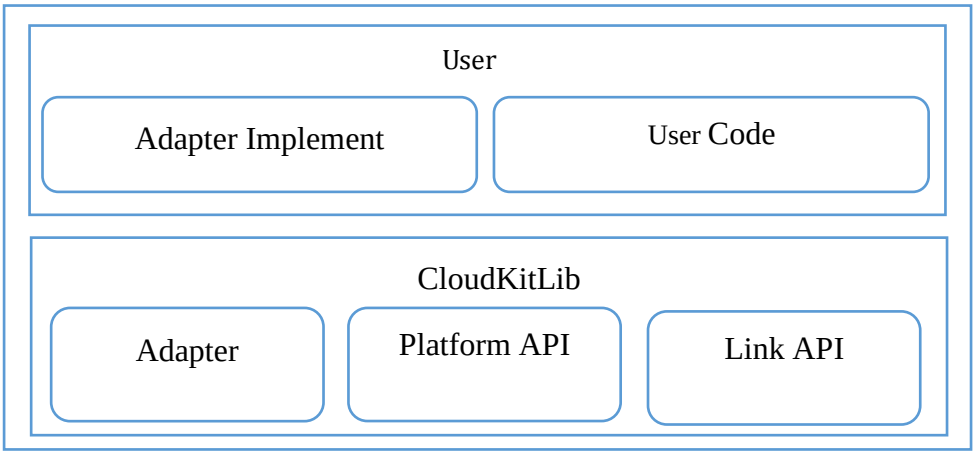

图 1 CloudKitLib 结构图

Adapter：适配层，用于适配不同的硬件环境，要求用户根据各自的硬件环境实现 Adapter 中头文件所定义的接口。

Platform API：平台协议相关接口，定义在 CTSmart2.h 和 CTSmart2Ctx.h 中

Link API：配网相关接口，定义在 CTSmart_link.h 中。

User Code：用户代码

2 开发

2.1 概念定义

2.1.1 UUID

设备的唯一标识码，在申请设备 ID 时会发送给云平台，相同的 UUID 的设备申请的设备 ID 相同。通常情况下为模块的 MAC 地址。

2.1.2 设备 ID（Deviceid）

平台分配的设备 ID 用于标识不同的设备。

2.1.3 产品型号（Model）

平台上分配的产品型号，每款产品不同，用于区分产品。同时平台也会分配一个型号 PIN 码，型号 PIN 码在申请设备 ID 时使用。

2.2 Adapter 实现

使用 CloudKitLib 需要用户首先实现 Adapter 中所定义的接口，主要包含以下部分：

1. CTSmart2AdapterCrypto

   AES 加密接口。

2. CTSmart2NetifIsOK

   判断网络是否可用，如 WiFi 设备为判断是否连接路由器。

3. CTSmart2GetDeviceUUID

   获取设备唯一标识，如为 WiFi 设备则返回 MAC 地址如 AABBCCDDEEFF。

4. CTSmart2Socket

   Socket 相关接口，硬件环境上的 socket 如为 posix 风格接口请参考

Demo/PC 中的代码实现，lwip 的接口请参考 Demo/ESP8266 中的代码实现。

2.3 UserCode

2.3.1 平台接入

平台接入的 API 定义在 CTSmart2.h 和 CTSmart2Ctx.h 中，用户需要引用 CTSmart2.h。

2.3.1.1 初始化

初始化代码中的平台参数中的登陆服务器、ID 分配服务器、平台 ID 需要在开发前进行确认。

```
//初始化
CTSmart2Init();
//创建平台上下文
g_ctx = CTSmart2CtxCreate();
//设置平台参数
CTSmart2CtxPlatformCfg_t platformCfg;
//ServerHost 如为域名则直接填写，如果是 IP 地址则需要以"IP:"开头，如"IP:192.168.0.1"
platformCfg.loginServerHost = "IP:101.95.50.85"; //登陆服务器
platformCfg.loginServerPort = 6779;
platformCfg.applyidServerHost = "IP:101.95.50.85"; //ID 分配服务器
platformCfg.applyidServerPort = 10088;
platformCfg.platformid = 0; //平台 ID
CTSmart2CtxSetPlatformCfg(g_ctx, &platformCfg);
//设置事件处理回调
CTSmart2CtxSetEventHandler(g_ctx, ctsmart2EventHandler);
//设置设备型号 ID，型号 ID 从平台分配，每款产品唯一
CTSmart2CtxSetDeviceModel(g_ctx, MODEL_ID);
//开始设备添加发现
CTSmart2CtxStartAddDiscover(g_ctx);
```

2.3.1.2 申请设备 ID.

设备 ID 用于唯一标识一款设备，在开始连接平台之前先要连接 ID 分配服务器分配设备 ID。

```
//使用型号 ID 和型号 PIN 码进行设备 ID 申请，型号 ID 和型号 PIN 由平台分配
CTSmart2CtxApplyDeviceid(g_ctx, MODEL_ID, MODEL_PIN);
```

申请到设备 ID 后，会通过事件回调给用户，需要将设备 ID 进行永久化存储避免每次连接平台都进行申请。

2.3.1.3 添加设备

在 APP 绑定设备需要将设备切换到添加发现，一般逻辑是先进行配网（详见 WiFi 配网），配网成功连上路由器后，调用接口 CTSmart2CtxStartAddDiscover，将设备切换到添加发现状态，此时在 APP 上搜索设备（发送 UDP 广播包），SDK 会向 APP 回复设备信息（要求设备同 APP 处于同一局域网并且已经有设备 ID），完成绑定。需要注意的是调用 CTSmart2CtxStartAddDiscover 进入添加发现状态的超时时间为 1 分钟，1 分钟之后 SDK 会退出添加发现状态。

```
//进入添加发现状态，超时 1 分钟
CTSmart2CtxStartAddDiscover(g_ctx);

//回复 APP 添加发现
else if(event == CTSmart2EVENT_RESP_ADD_DISCOVER)
{
    SysLog("resp add discover");
}
```

2.3.1.4 连接平台

```
//设置设备 ID 和 PIN 码
CTSmart2CtxSetDeviceid(g_ctx, g_deviceid, g_devicepin);
//开始连接
CTSmart2CtxStartConnect(g_ctx);
```

2.3.1.5 状态上报

平台同设备之间的控制、状态上报都通过属性来进行，不同的属性的 ID 不同，如将开关定义为属性 1，温度定义为属性 2，则将设备自身的开关状态通过属性 1 来上传，温度状态通过属性 2 来上传。属性类型包括两种，数值属性与文本属性。

```
//注册属性值
CTSmart2CtxRegistProperty(g_ctx, 1, CTSmart2_PROPERTY_TYPE_NUM);
CTSmart2CtxRegistProperty(g_ctx, 2, CTSmart2_PROPERTY_TYPE_NUM);

//同步属性值
CTSmart2PropertyValue_t value;
value.num = 0;
CTSmart2CtxSyncPropertyValue(g_ctx, 1, value);
CTSmart2CtxSyncPropertyValue(g_ctx, 2, value);
```

2.3.1.6 控制

APP 来的控制命令会通过事件回调通知用户，具体处理详见**事件处理**部分。

2.3.1.7 事件处理

```
static void ctsmart2EventHandler(CTSmart2Ctx_t *ctx, muint8_t event, void *arg)
{
    //控制
    if(event == CTSmart2EVENT_OPT)
    {
```

```
        CTSmart2EventOptArg_t *opt = arg;
        uint8_t i;
        for(i = 0; i < opt->propertyNum; i++)
        {
            CTSmart2Property_t *property = &opt->propertys[i];
            //收到控制后需要将被控制后的属性值进行同步
            CTSmart2CtxSyncPropertyValue(g_ctx, property->pid, property->value);

            SysLog("control propertyid:%d value:%d", property->pid, property->value.num);
        }
    }
    //申请设备 ID
    else if(event == CTSmart2EVENT_APPLYID)
    {
        //申请到设备 ID 和 PIN 码，应该写入 flash 保存
        CTSmart2EventApplyidArg_t *applyidArg = arg;
        strcpy(g_deviceid, applyidArg->deviceid);
        strcpy(g_devicepin, applyidArg->devicepin);
        SysLog("deviceid:%s pin:%s", applyidArg->deviceid, applyidArg->devicepin);
    }
    //查询
    else if(event == CTSmart2EVENT_QUERY)
    {
        //如果设备状态已经实时通过 CTSmart2CtxSyncPropertyValue 同步则不需要进行任何处理，否则需要将属性同步一遍。
        SysLog("Query");
    }
    //更新服务器时间
    else if(event == CTSmart2EVENT_UPDATE_SERVER_TIME)
    {
        CTSmart2EventUpdaeServerTimeArg_t *timearg = arg;
        SysLog("timezone:%d seconds:%d", timearg->timezone, timearg->seconds);
    }
    //解绑成功
    else if(event == CTSmart2EVENT_UNBIND_RESULT)
```

```
    {
        SysLog("unbind success.");
    }
    //固件升级
    else if(event == CTSmart2EVENT_UPGRADE)
    {

    }
}
```

2.3.1.8 固件升级

- 设置版本号

```
//设置版本号，固件升级时用到
char version[] = VERSION;
CTSmart2CtxSetDeviceVer(g_ctx, version);
```

- 处理升级通知，由平台推送

```
//固件升级通知
else if(event == CTSmart2EVENT_OTA_NOTIFY)
{
CTSmart2EventUpgradeArg_t *upgradeArg = arg;
mlog("ota notify type:%d version:%d.%d.%d.%d",
            upgradeArg->type,
            upgradeArg->version[0],
                upgradeArg->version[1],
                upgradeArg->version[2],
          upgradeArg->version[3]);
            g_otaType = upgradeArg->type;
            memcpy(g_version, upgradeArg->version, 4);
            CTSmart2CtxStartOTADownload(ctx, upgradeArg, 0);
}
```

- 接收固件文件

```
//固件数据
    else if(event == CTSmart2EVENT_OTA_DATA)
    {
        CTSmart2EventOTADataArg_t *dataArg = arg;
        if(dataArg->status == CTSmart2_OTA_DATA_STAUTS_SUCCESS)
        {
            mlog("ota success");
            CTSmart2CtxStopOTADownlaod(ctx);
            CTSmart2CtxUpdateOTAStatus(ctx, g_otaType,
CTSmart2_OTA_STATUS_UPGRAD_SUCCESS, 0, g_version);
//TODO 固件下载完成，延时几秒等待升级进度传到平台后再可以调用底层接口完成程
序
        }
        else if(dataArg->status == CTSmart2_OTA_DATA_STAUTS_FAIL)
        {
            mlog("ota download fail");
            CTSmart2CtxStopOTADownlaod(ctx);
            CTSmart2CtxUpdateOTAStatus(ctx, g_otaType,
CTSmart2_OTA_STATUS_UPGRAD_FAIL, 0, g_version);
        }
        else
        {
////NOTICE 固件数据是分块发送
            //TODO 将接收到固件数据保存到 Flash 中
        }
    }
```

2.3.2 WiFi 配网

2.3.2.1 设备要求

无法满足以下以下能力要求的设备或模块将无法进行 WiFi 配网。

1. 能够切换信道
2. 能够设置为混杂模式，接收 802.11 无线帧

3. 提供一种进入配网模式的控制方式，例如一个按键

2.3.2.2 配网流程

- 开始配网

```
//初始化
ctsmartlink_init();
//切换到混杂模式并接收 802.11 无线帧
WifiSnifferStart(wifiSnifferCallback);
```

- 100ms 循环切换信道

```
static uint8_t chn = 1;
//未锁定信道时，循环 100ms 切换信道
if(!g_lockChn && g_running - g_lastSwitchChnTime > 100)
{
    g_lastSwitchChnTime = g_running;
    WifiSnifferSetChannel(chn);
    chn++;
    if(chn > 14)
    {
        chn = 1;
    }
}
```

- 解析 802.11 帧数据

```
static WifiSnifferStatus_t wifiSnifferCallback(const uint8_t *data, uint16_t len)
{
    ctsmartlink_status_t status = ctsmartlink_recv(data, len);
    //锁定信道
    if(status == CTSmartLINK_STATUS_LOCK)
    {
        g_lockChn = 1;
    }
    //配网完成
    else if(status == CTSmartLINK_STATUS_COMPLETE)
    {
        ctsmartlink_res_t res;
        ctsmartlink_get_res(&res);

        SysLog("got ssid:%s pwd:%s", res.ssid, res.pwd);

        g_startNetCfg = 0;
        WifiSnifferStop();

        //连接路由器
        WifiJoinAp(res.ssid, res.pwd, WIFI_AUTH_WPA_WPA2_PSK);
        WifiReconnect();

//开始设备添加发现，开始设备添加发现之后才能被 APP 发现并添加
        CTSmart2CtxStartAddDiscover(g_ctx);
    }
}
```