# 基于STM32节点和阿里云IoT平台的物联网应用开发 系列课程

## 第三章

## 基于STM32的节点设备接入阿里云IoT平台

life.augmented

阿里云

# 课程内容下载、观看



- 视频观看：AI电堂、阿里云大学IoT课堂
- 课件胶片下载：STMCU中文官网、阿里云大学IoT课堂
- 课件项目下载：STMCU中文官网、阿里云大学IoT课堂

STM32公众号

STM中文官网


电堂公众号

阿里云大学

# 第三章内容简介



- 第一节：基于STM32的节点端介绍
  - 硬件平台，软件开发环境
- 第二节：使用Paho MQTT客户端协议栈直连阿里云IoT平台
  - 适用于资源受限的节点设备
- 第三节：使用Linkkit C-SDK和TLS通过MQTT协议直连阿里云IoT平台
  - 适用于资源丰富的节点设备

# STM32-阿里云IoT　联合课件开发

## 第三章．第三节

## 基于阿里Linkkit C-SDK通过MQTT直连阿里云IoT平台

life.augmented

阿里云

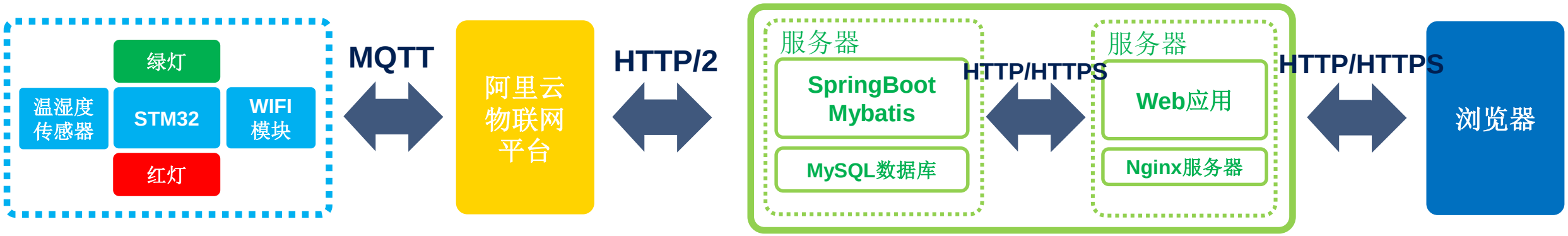
# 第三章.第三节 内容简介



- 项目例程演示
  - 项目例程流程图
  - 演示视频
- Linkkit C-SDK介绍
  - Linkkit C-SDK
  - Linkkit C-SDK如何适配到STM32平台
  - Linkkit C-SDK的使用
- 项目例程介绍
  - 软件包和项目结构
  - 使用CubeMX生成系统初始化框架和代码
  - 网络通信的管理（网络通信抽象层和wifi驱动）
  - 使用Linkkit C-SDK连接阿里云IoT平台
  - 例程参数的存储及Sensor数据的读取

# 课程场景






- 每5秒上报温湿度值，闪烁绿灯
- 温度超【阈值】亮红灯，并在每10秒向用户服务器报警，直到温度恢复【阈值】以下或者收到警报解除消息
- 收到警报解除信息后红灯闪烁
- 温度恢复到【阈值】以下灭红灯

- 湿度值被阿里云IoT转发到用户服务器，进行数据库存储，同时在web端显示近期温湿度数据曲线
- 报警消息被阿里云IoT转发到用户服务器，在web端显示
- 用户通过web端页面解除报警
- 用户通过web端页面设置【阈值】参数

life.augmented | 阿里云

# 运行效果 – 节点端串口打印



## 连接准备

```
*******777*************************************************************
***                  STM32 based AliIoT Client Demo                  ***
***                      With TLS and FREERTOS                       ***
***              FW version 0.3.0 - Mar  8 2019, 12:25:08            ***
***********************************************************************

Application parameter init. Send alarm when temprature>= 38 degrees Celsius,turn off Red LED when temprature<37 degrees Celsius

*** start WiFi provisioning ***

Push the User button (Blue) within the next 5 seconds if you want to update the WiFi network configuration.

Your WiFi parameters need to be entered to proceed.

Enter SSID: [redacted]

You have entered [redacted] as the ssid.

Enter Security Mode (0 - Open, 1 - WEP, 2 - WPA, 3 - WPA2):3
You have entered 3 as the security mode.

Enter password: [redacted]

*** start WiFi initialization ***

> WiFi module initialized.

firmware version is : basic_AT_v2.1.2
OK

*** try to get WiFi MAC address ***

> WiFi module MAC address is: B0:F8:93:17:BB:2C
```

若要重新配置wifi热点，
需要5秒内按下user键

输入热点名称和密码

```
*** try to connect to AP ***

> WiFi module connected to access point.

*** try to get IP address ***

> WiFi module got IP address : 192.168.43.101

Push the User button (Blue) within the next 5 seconds if you want to update the device security parameters or credentials.

Enter Product Key: (example: a1b05Uexxxx)
a1zPqJ3NYxJ
read: --->
a1zPqJ3NYxJ
<---

Enter device name: (example: mydevicename)
2019CT_case2_node001
read: --->
2019CT_case2_node001
<---

Enter device secret: (example: 7o7GJ3odUE7pPnieO7dzxxxxxxxxxxx)
1VDt2s7hUyZVL62oS9VXgT6mnjjRoOCi
read: --->
[redacted]
<---
```

输入product key

输入device name

2019CT_case2_node001

输入device secret

life.augmented

阿里云

# 运行效果 – 节点端串口打印



## 连接准备

```
[inf] guider_print_dev_guider_info(320): ----------------------------------------------
[inf] guider_print_dev_guider_info(321):          ProductKey : a1zPqJ3MYxJ
[inf] guider_print_dev_guider_info(322):          DeviceName : 2019CT_case2_node001
[inf] guider_print_dev_guider_info(323):            DeviceID : a1zPqJ3MYxJ.2019CT_case2_node001
[inf] guider_print_dev_guider_info(327): ----------------------------------------------
[inf] guider_print_dev_guider_info(328):       PartnerID Buf : ,partner_id=example.demo.partner-id
[inf] guider_print_dev_guider_info(329):       ModuleID Buf : ,module_id=example.demo.module-id
[inf] guider_print_dev_guider_info(330):          Guider URL :
[inf] guider_print_dev_guider_info(332):      Guider SecMode : 2 (TLS + Direct)
[inf] guider_print_dev_guider_info(334):    Guider Timestamp : 2524608000000
[inf] guider_print_dev_guider_info(338): ----------------------------------------------
[inf] guider_print_conn_info(297): ------------------------------------------------
[inf] guider_print_conn_info(298):            Host : a1zPqJ3MYxJ.iot-as-mqtt.cn-shanghai.aliyuncs.com
[inf] guider_print_conn_info(299):            Port : 1883
[inf] guider_print_conn_info(304):        ClientID : a1zPqJ3MYxJ.2019CT_case2_node001|securemode=2,timestamp=2524608000000,signmethod=hmacsha1,gw=0,ext=0,partner_id=example
.demo.partner-id,module_id=example.demo.module-id|
[inf] guider_print_conn_info(306):       TLS PubKey : 8027d28 ('-----BEGIN CERTI ...')
[inf] guider_print_conn_info(309): ------------------------------------------------
[inf] iotx_mc_init(2334): MQTT init success!
[inf] _ssl_client_init(196): Loading the CA root certificate ...
cert. version     : 3
serial number     : 04:00:00:00:00:01:15:4B:5A:C3:94
issuer name       : C=BE, O=GlobalSign nv-sa, OU=Root CA, CN=GlobalSign Root CA
subject name      : C=BE, O=GlobalSign nv-sa, OU=Root CA, CN=GlobalSign Root CA
issued  on        : 1998-09-01 12:00:00
expires on        : 2028-01-28 12:00:00
signed using      : RSA with SHA1
RSA key size      : 2048 bits
basic constraints : CA=true
key usage         : Key Cert Sign, CRL Sign
[inf] _ssl_parse_crt(164): crt content:451
[inf] _ssl_client_init(204):  ok (0 skipped)
[inf] _TLSConnectNetwork(286): Connecting to /a1zPqJ3MYxJ.iot-as-mqtt.cn-shanghai.aliyuncs.com/1883...
[inf] _TLSConnectNetwork(299):  ok
[inf] _TLSConnectNetwork(304):   . Setting up the SSL/TLS structure...
[inf] _TLSConnectNetwork(314):  ok
[inf] _TLSConnectNetwork(349): Performing the SSL/TLS handshake...
[inf] _TLSConnectNetwork(357):  ok
[inf] _TLSConnectNetwork(361):   . Verifying peer X.509 certificate..
[inf] _real_confirm(113): certificate verification result: 0x00
[inf] iotx_mc_connect(2724): mqtt connect success!
```

根据设备三元组，Linkkit SDK 进行设备认证

TLS建立 by mbedTLS

life.augmented

阿里云

# 运行效果 – 节点端串口打印



## 连接成功：上线，发布

# 运行效果 – 阿里云IoT平台侧



## 设备上线，发布消息


节点设备定时上报温、湿度；
发布主题；
消息QoS = 1

Publish message to topic:/sys/a1zPqJ3NYxJ/2019CT_case2_node001/thing/event/property/post,QoS=1


设备名称：2019CT_case2_node001

发布当前温度(25.xx)、湿度(29)和温度报警阈值(38)

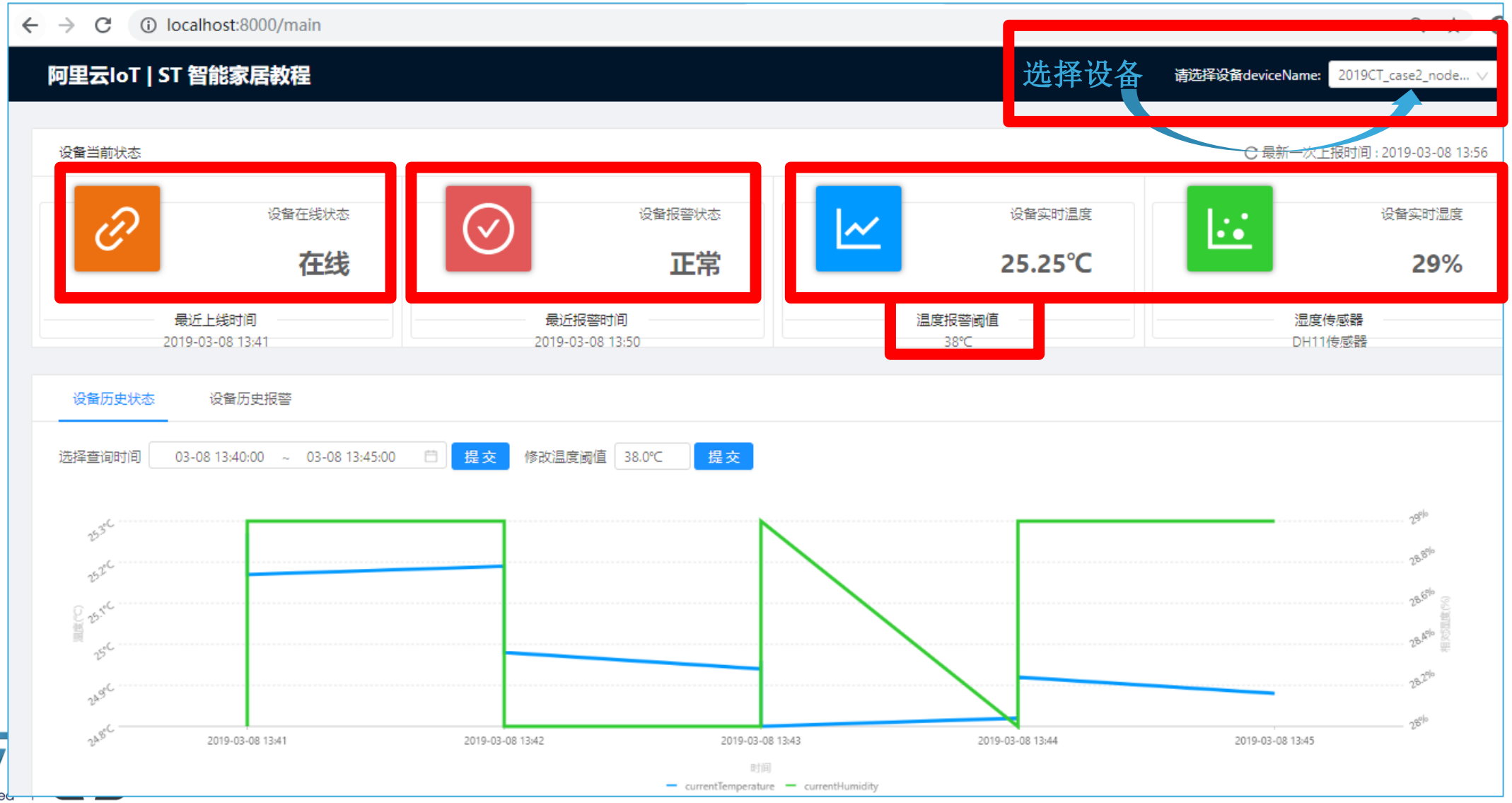
life.augmented

阿里云

# 运行效果 – 应用服务器侧

## 设备上线，发布消息





# 运行效果 – 应用服务器侧



## 用户修改温度报警阈值

# 运行效果 –节点端串口打印



## 温度阈值被改变，触发节点端报警行为

```
**********received message **********
[inf] _demo_message_arrive(247):
Topic: /sys/a1zPqJ3MYxJ/2019CT_case2_node001/thing/service/property/set

Topic Length: 64
[inf] _demo_message_arrive(252):

Payload: {"method":"thing.service.property.set","id":"278051847","params":{"TempThreshold":18},"version":"1.0.0"}

Payload Length: 104

**Need update the Thread*********

[inf] _demo_message_arrive(275):
**updateThread 18*********


**********

********publish success, packet-id=94 ********

[inf] mqtt_client_example(407):
**********packet-id=95, publish topic alarm Value msg={ "method":"thing.event.TempAlarm.post", "id":"123","params":{}}

 packet-id=96, publish topic msg={ "method":"thing.event.property.post", "id":"123","params":{"CurrentHumidity":28,"CurrentTemperature":25.05,"TempThreshold":18},"version":"
1.0.0"}
********publish success, packet-id=95 ********


********publish success, packet-id=96 ********

[inf] mqtt_client_example(407):
**********packet-id=97, publish topic alarm Value msg={ "method":"thing.event.TempAlarm.post", "id":"123","params":{}}

 packet-id=98, publish topic msg={ "method":"thing.event.property.post", "id":"123","params":{"CurrentHumidity":28,"CurrentTemperature":25.03,"TempThreshold":18},"version":"
1.0.0"}
********publish success, packet-id=97 ********
```

收到订阅的主题的消息

数据负载遵循Alink协议：温度阈值 → 18

节点设备发送“报警”事件给IoT平台

节点设备发送“报警”事件给IoT平台

# 项目例程演示 视频

# 第三章.第三节 内容简介



- 项目例程演示
  - 项目例程流程图
  - 演示视频
- Linkkit C-SDK介绍
  - Linkkit C-SDK
  - Linkkit C-SDK如何适配到STM32平台
  - Linkkit C-SDK的使用
- 项目例程介绍
  - 软件包和项目结构
  - 使用CubeMX生成系统初始化框架和代码
  - 网络通信的管理（网络通信抽象层和wifi驱动）
  - 使用Linkkit C-SDK连接阿里云IoT平台
  - 例程参数的存储及Sensor数据的读取

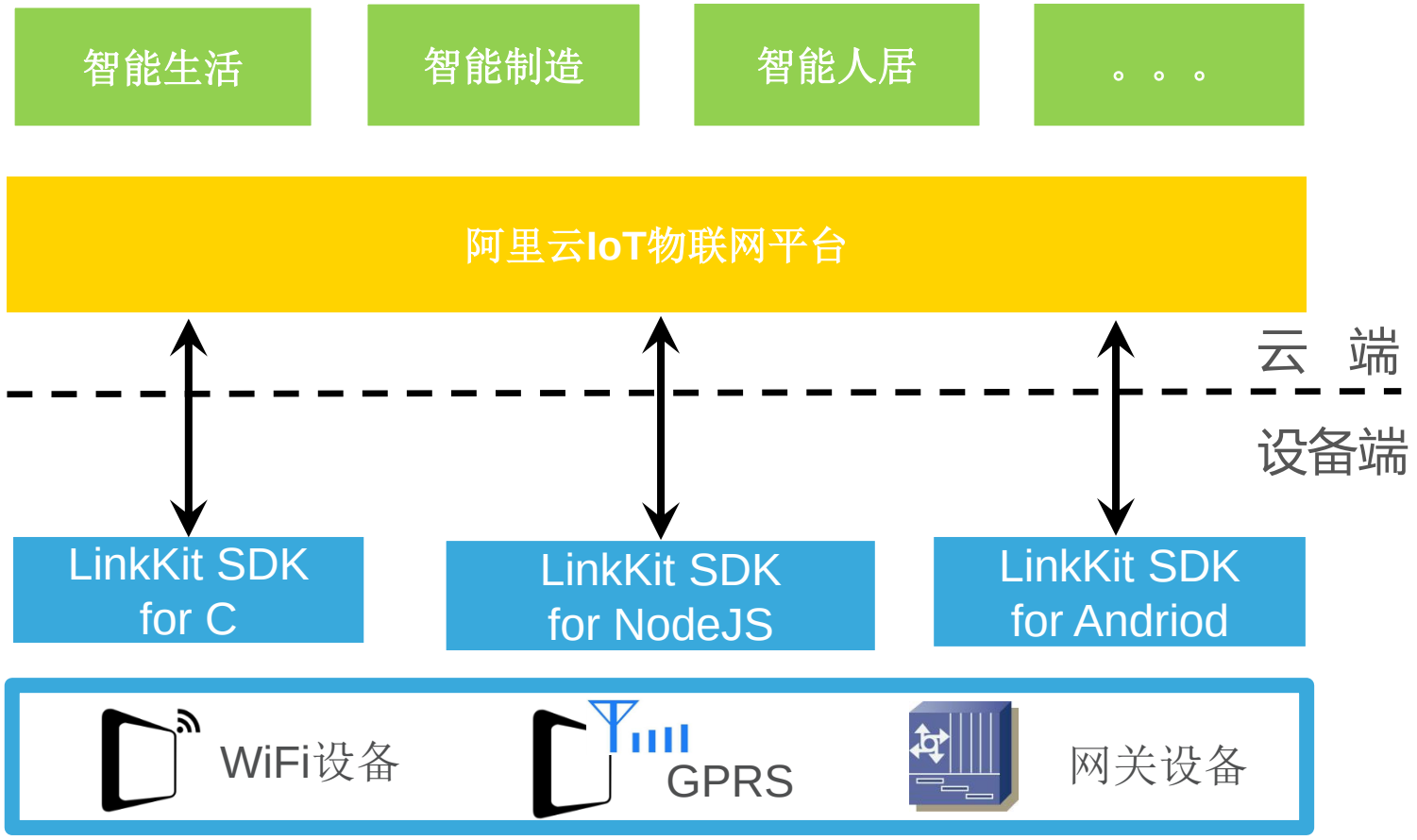
# Linkkit SDK



- 阿里云IoT提供给设备厂商用于快速将设备接入阿里云IoT平台的功能代码集

Linkkit C-SDK 2.3.0

Linkkit C-SDK下载




life.augmented 阿里云

# Linkkit SDK的主要功能

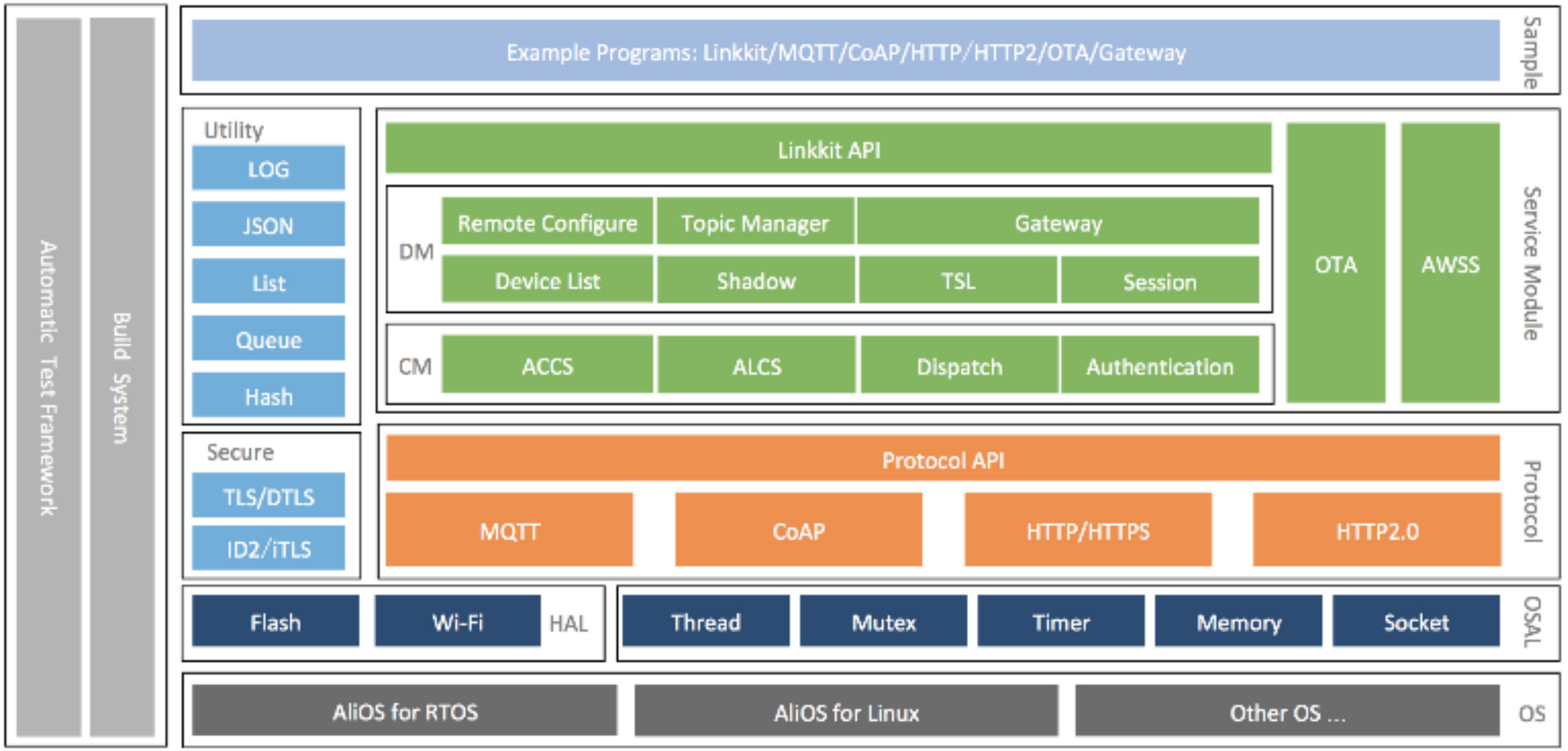


| 基础功能 @ 阿里云IoT平台设备管理.基础版 | |
|---|---|
| 设备连云 | 支持多种协议连接阿里云IoT平台：MQTT、CoAP、HTTP/S、HTTP2(做文件上传) |
| 设备认证 | 多种身份认证方式：一机一密，一型一密 |
| 数据通信 | SDK提供上行、下行数据接口，数据加密功能 |
| **高级功能 @ 阿里云IoT平台设备管理.高级版** | |
| 物模型 | 处理属性上报、设置；服务调用、事件上报 |
| 设备OTA | 设备厂商通过阿里云IoT平台推送固件给设备，设备SDK负责接收接收固件并调用设备升级函数实现固件的烧写 |
| 时间同步 | SDK可从云端获取实时时间，用于设备的定时任务 |
| Wifi配网 | SDK提供多种配网方式：一键配网、手机热点配网、零配、路由器配网 |

更多功能，参见阿里云IoT线上文档：https://help.aliyun.com/document_detail/96623.html

# Linkkit C-SDK架构框图






https://code.aliyun.com/edward.yangx/public-docs/wikis/user-guide/linkkit/SDK_Overview

# Linkkit SDK源文件组织



```
.
+-- build-rules          : 编译构建系统，基于GNU Make和bash脚本
+-- project.mk           : 编译系统配置，指定SDK的目录排布等
+-- LICENSE              : 软件许可证，C-SDK使用的是Apache-2.0版本软件许可证
+-- README.txt           : 简要说明，列出了C-SDK的功能模块和模块内容等
+-- makefile             : 基于GNU Make编译SDK的顶层Makefile
+-- CMakeLists.txt       : 基于cmake编译SDK的顶层CMakeLists.txt
+-- make.settings        : 功能裁剪配置，可编辑该文件来裁剪SDK的内容
|
+-- examples             : 例程目录，演示SDK的使用
|   +-- coap             :     演示如何使用通信模块CoAP的API
|   +-- device-shadow    :     演示如何使用服务模块DeviceShadow的API
|   +-- http             :     演示如何使用通信模块HTTP的API
|   +-- http2            :     演示如何使用通信模块HTTP2的API
|   +-- linkkit          :     演示如何使用服务模块linkkit的API
|   +-- mqtt             :     演示如何使用通信模块MQTT的API
|   +-- ota:             :     演示如何使用服务模块ota的API
|
+-- include              : 头文件目录，列出SDK依赖的HAL接口和向用户提供的API接口
|   +-- iot_export.h     :     列出所有API层函数的声明，是SDK所提供的接口
|   +-- iot_import.h     :     列出所有HAL层函数的声明，是SDK所依赖的接口
|   +-- exports          :     列出各功能点提供的API层接口
|   +-- imports          :     列出各功能点依赖的HAL层接口
|
```

```
+-- src
|   +-- board                  : 跨平台适配目录，一个目标平台对应一个config.xxx.yyy文件
|   |   +-- config.rhino.make  :     适配到AliOS Things系统的编译配置文件
|   |   +-- config.ubuntu.x86  :     适配到Ubuntu系统的编译配置文件
|   |   +-- config.win7.mingw32:     适配到Win7/Win10系统的编译配置文件
|   |
|   +-- infra                  : SDK核心实现中的基础模块目录
|   |   +-- log                :     实现运行时SDK的日志
|   |   +-- system             :     实现全局信息保存，如官方根证书，设备标识ID等
|   |   +-- utils              :     实现工具函数，如连接鉴权时的SHA1摘要计算等
|   +-- protocol               : SDK核心实现中的通信模块目录
|   |   +-- alcs               :     实现设备和手机app之间的本地加密通信
|   |   +-- coap               :     实现设备和阿里云之间的CoAP协议通信
|   |   +-- http               :     实现设备和阿里云之间的HTTP协议通信
|   |   +-- http2              :     实现设备和阿里云之间的HTTP2协议通信
|   |   +-- mqtt               :     实现设备和阿里云之间的MQTT协议通信
|   +-- services               : SDK核心实现中的服务模块目录
|   |   +-- awss               :     实现设备和手机app之间的WiFi配网服务
|   |   +-- linkkit            :     实现设备和阿里云之间的物模型管理服务
|   |   +-- ota                :     实现设备和阿里云之间的固件升级服务
|   |
|   +-- sdk-impl               : API实现目录，iot_export.h中的API在此实现
|   +-- ref-impl               : 参考实现目录，包括加解密库和HAL接口的参考实现
|   |   +-- hal                :     SDK所依赖的HAL接口的参考实现
|   |   +-- tls                :     加解密库的参考实现，由开源软件mbedtls裁剪而成
|   |
|   +-- tools                  : 用于辅助build-rules编译系统的编译脚本文件
|
```

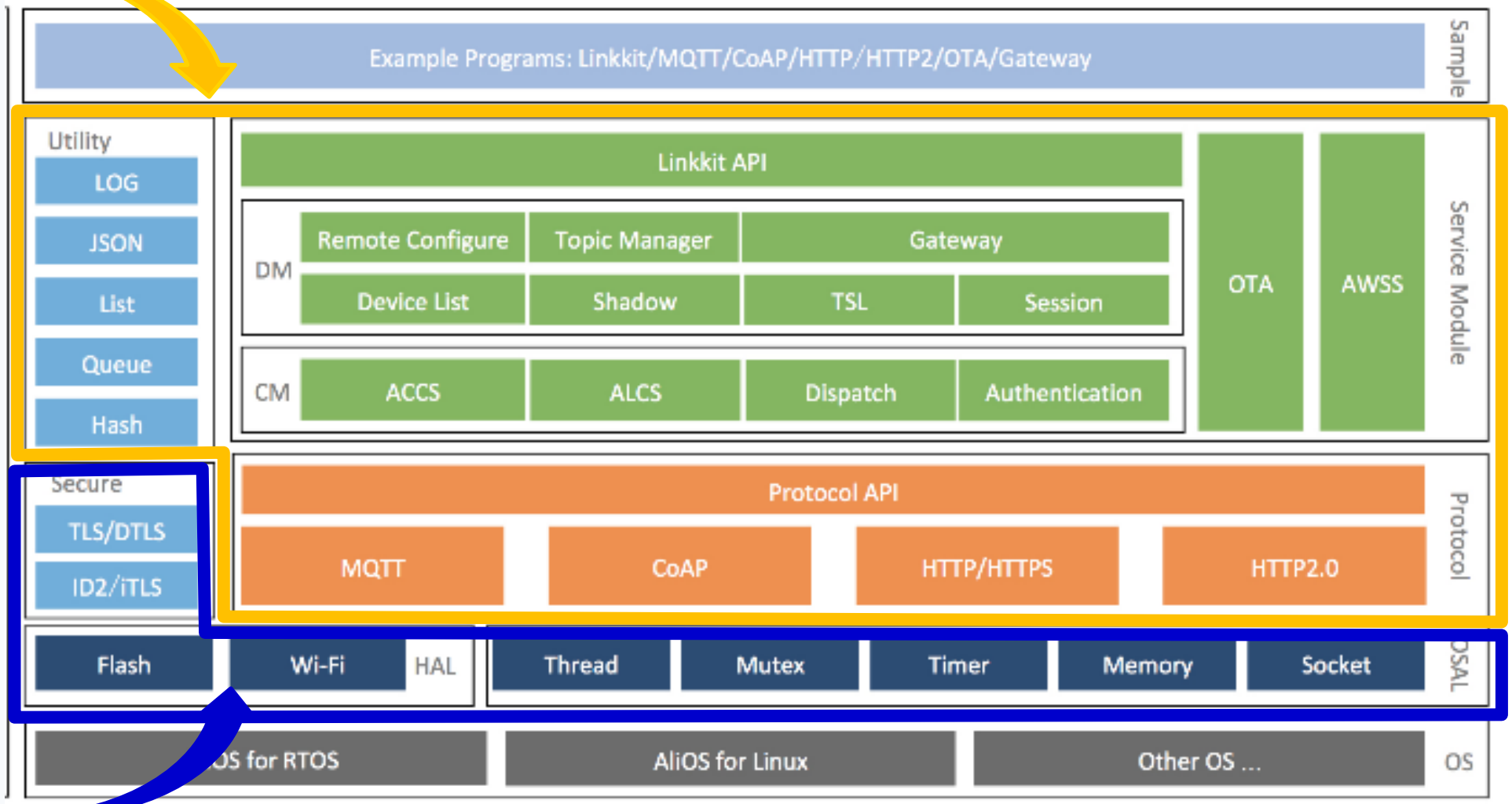
life.augmented

阿里云

# Linkkit C-SDK架构框图



```
+-- src
|   +-- board
|   |   +-- config.rhino.make
|   |   +-- config.ubunt    x86
|   |   +-- config.w        .mingw32
|   |
|   +-- infra
|   |   +-- log
|   |   +-- system
|   |   +-- utils
|   +-- protocol
|   |   +-- alcs
|   |   +-- coap
|   |   +-- http
|   |   +-- http2
|   |   +-- mqtt
|   +-- services
|   |   +-- awss
|   |   +-- linkkit
|   |   +-- ota
|   |
|   +-- sdk-impl
|   +-- ref-impl
|   |   +-- hal
|   |   +-- tls
|   |
|   +-- tools
|
```

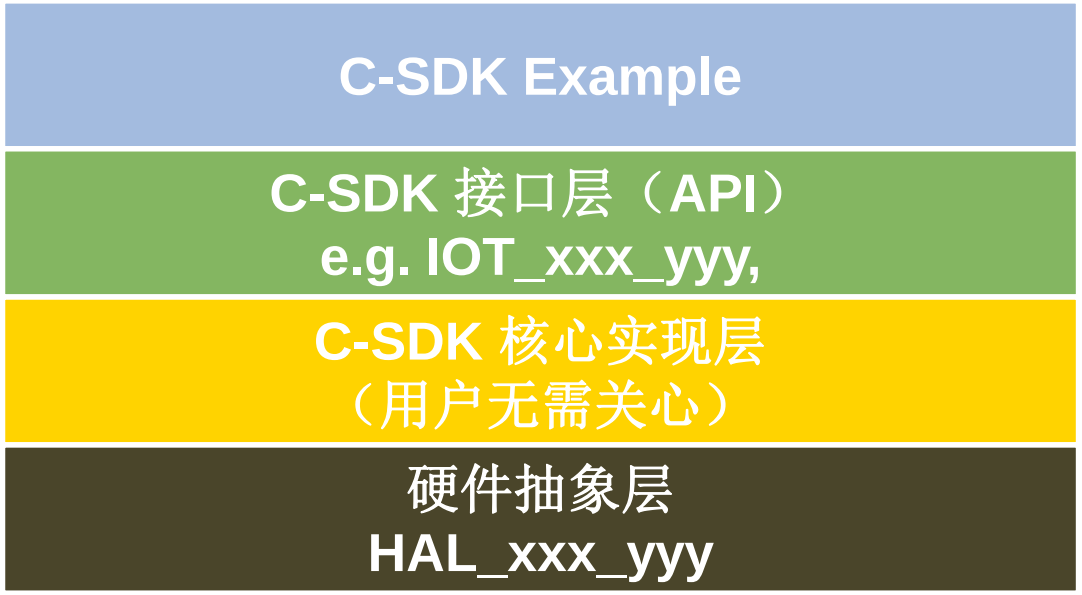


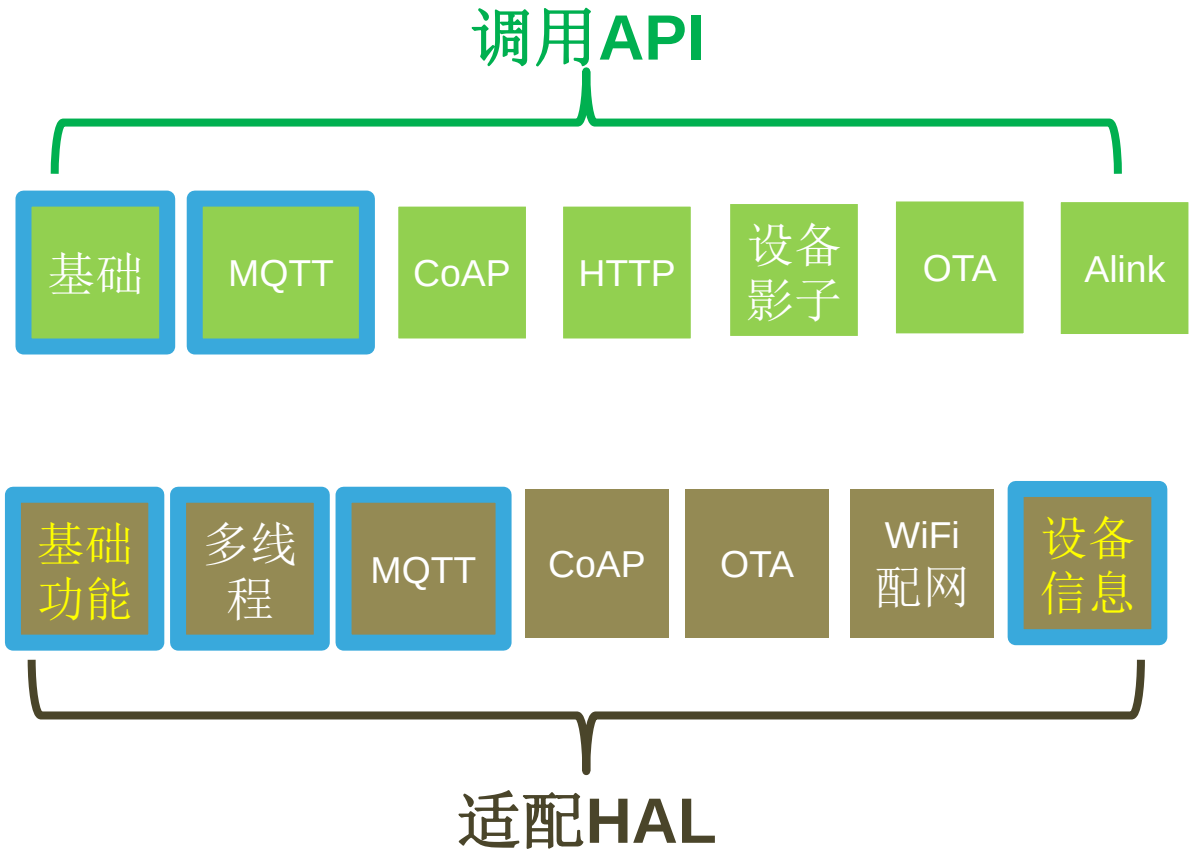

https://code.aliyun.com/edward.yangx/public-docs/wikis/user-guide/linkkit/SDK_Overview

# C-SDK的适配和调用



- 设备端C-SDK的分层




SDK中已提供Linux桌面OS下的实现参考

# 【设备信息】相关HAL



| 函数名 | 说明 |
| --- | --- |
| HAL_GetChipID | 获取芯片ID |
| HAL_GetDeviceID | 获取设备ID (在2.3.1及以后版本中不需要实现） |
| HAL_GetFirmwareVesion | 获取固件版本号, 必须实现 |
| HAL_GetModuleID | 获取模组ID, 用于紧密合作伙伴, 一般客户只需要在此可实现为空函数 |
| HAL_GetPartnerID | 获取合作伙伴ID, 用于紧密合作伙伴, 一般客户只需要在此可实现为空函数 |
| HAL_GetDeviceName | 获取DeviceName, 三元组获取函数之一, 必须实现 |
| HAL_GetDeviceSecret | 获取DeviceSecret, 三元组获取函数之一, 必须实现 |
| HAL_GetProductKey | 获取ProductKey, 三元组获取函数之一, 必须实现 |
| HAL_GetProductSecret | 获取ProductSecret, 三元组获取函数之一, 必须实现 |
| HAL_SetDeviceName | 设置DeviceName, 三元组配置函数之一, 必须实现 |
| HAL_SetDeviceSecret | 设置DeviceSecret, 三元组配置函数之一, 必须实现 |
| HAL_SetProductKey | 设置ProductKey, 三元组配置函数之一, 必须实现 |
| HAL_SetProductSecret | 设置ProductSecret, 三元组配置函数之一, 必须实现 |

# 【设备信息】相关HAL的实现



| 被调用的上下文 | iot_HAL_OS_stm32.c | | |
|---|---|---|---|
| 用于iTLS安全方案时的网络连接；<br><br>上报设备端信息； | HAL_GetPartnerID | example.demo.ST | |
| | HAL_GetModuleID | example.demo.emw3080 | |
| | HAL_GetFirmwareVersion | app-1.0.0-20190101.1000 | |
| | HAL_GetProductKey | 从_product_key读出 | |
| | HAL_GetDeviceName | 从_device_name读出 | |
| | HAL_GetDeviceSecret | 从_device_secret读出 | |
| | HAL_GetProductSecret | 从_product_secret读出 | |
| | HAL_GetDeviceID | 由_product_key. _device_name拼成 | |
| | HAL_GetNetifInfo | 保留参考实现 | |
| 从flash固定位置读出配置写到运行环境的全局变量中 | HAL_SetProductKey | 写_product_key | <iot_HAL_OS_stm32.c>中定义的全局数组变量 |
| | HAL_SetDeviceName | 写_device_name | |
| | HAL_SetDeviceSecret | 写_device_secret | |
| | HAL_SetProductSecret | 写_product_secret | |

# 【基础功能】相关HAL



| 函数名 | 说明 |
| --- | --- |
| HAL_Malloc | 申请一片堆上内存 |
| HAL_Free | 释放一片堆上内存 |
| HAL_SleepMs | 睡眠函数，使当前执行线程睡眠指定的毫秒数 |
| HAL_Snprintf | 打印函数，向内存缓冲区格式化构建一个字符串，参考C99标准库函数 `snprintf` |
| HAL_Printf | 打印函数，用于向串口或其它标准输出打印日志或调试信息 |
| HAL_Vsnprintf | 字符串打印函数，将 `va_list` 类型的变量，打印到指定目标字符串 |
| HAL_UptimeMs | 时钟函数，获取本设备从加电以来到目前时间点已经过去的毫秒数 |
| HAL_Kv_Set | 写入指定KV数据 |
| HAL_Kv_Get | 读取指定KV数据 |
| HAL_Kv_Del | 删除指定KV数据 |
| HAL_Kv_Erase_All | 擦除所有的KV数据 |

# 【基础功能】相关HAL的实现



| 被调用的上下文 | iot_HAL_OS_stm32.c | |
|---|---|---|
| | HAL_Malloc | malloc |
| | HAL_Free | free |
| | HAL_UptimeMs | HAL_GetTick |
| | HAL_SleepMs | HAL_Delay |
| 系统打印到终端或形成新字符串 | HAL_Printf | 保留参考实现 |
| | HAL_Snprintf | |
| | HAL_Vsnprintf | |
| IOT_SetupConnInfo里动态注册时检查设备证书是否在KV中 | HAL_Kv_Get | 没有实现 |
| | HAL_Kv_Set | |
| | HAL_Kv_Del | |

# 【多线程功能】相关HAL的实现



| 被调用的上下文 | iot_HAL_OS_stm32.c | |
|---|---|---|
| MQTT client初始化 (iotx_mc_init) | HAL_MutexCreate | 空函数，直接返回 |
| | HAL_MutexDestroy | |
| | HAL_MutexLock | |
| | HAL_MutexUnlock | |
| 未被本项目例程所需要SDK的子模块调用 | HAL_SemaphoreCreate | 未实现 |
| | HAL_SemaphoreDestroy | |
| | HAL_SemaphorePost | |
| | HAL_SemaphoreWait | |
| | HAL_ThreadCreate | |
| | HAL_ThreadDetach | |
| | HAL_ThreadDelete | |

# 【MQTT上云】相关HAL



| 函数名 | 说明 |
| --- | --- |
| HAL_SSL_Destroy | 销毁一个TLS连接, 用于MQTT功能, HTTPS功能 |
| HAL_SSL_Establish | 建立一个TLS连接, 用于MQTT功能, HTTPS功能 |
| HAL_SSL_Read | 从一个TLS连接中读数据, 用于MQTT功能, HTTPS功能 |
| HAL_SSL_Write | 向一个TLS连接中写数据, 用于MQTT功能, HTTPS功能 |
| HAL_TCP_Destroy | 销毁一个TLS连接, 用于MQTT功能, HTTPS功能 |
| HAL_TCP_Establish | 建立一个TCP连接, 包含了域名解析的动作和TCP连接的建立 |
| HAL_TCP_Read | 在指定时间内, 从TCP连接读取流数据, 并返回读到的字节数 |
| HAL_TCP_Write | 在指定时间内, 向TCP连接发送流数据, 并返回发送的字节数 |
| HAL_Random | 随机数函数, 接受一个无符号数作为范围, 返回0到该数值范围内的随机无符号数 |
| HAL_Srandom | 随机数播种函数, 使 HAL_Random 的返回值每个执行序列各不相同, 类似 `srand` |

# 【MQTT通道】相关HAL的实现

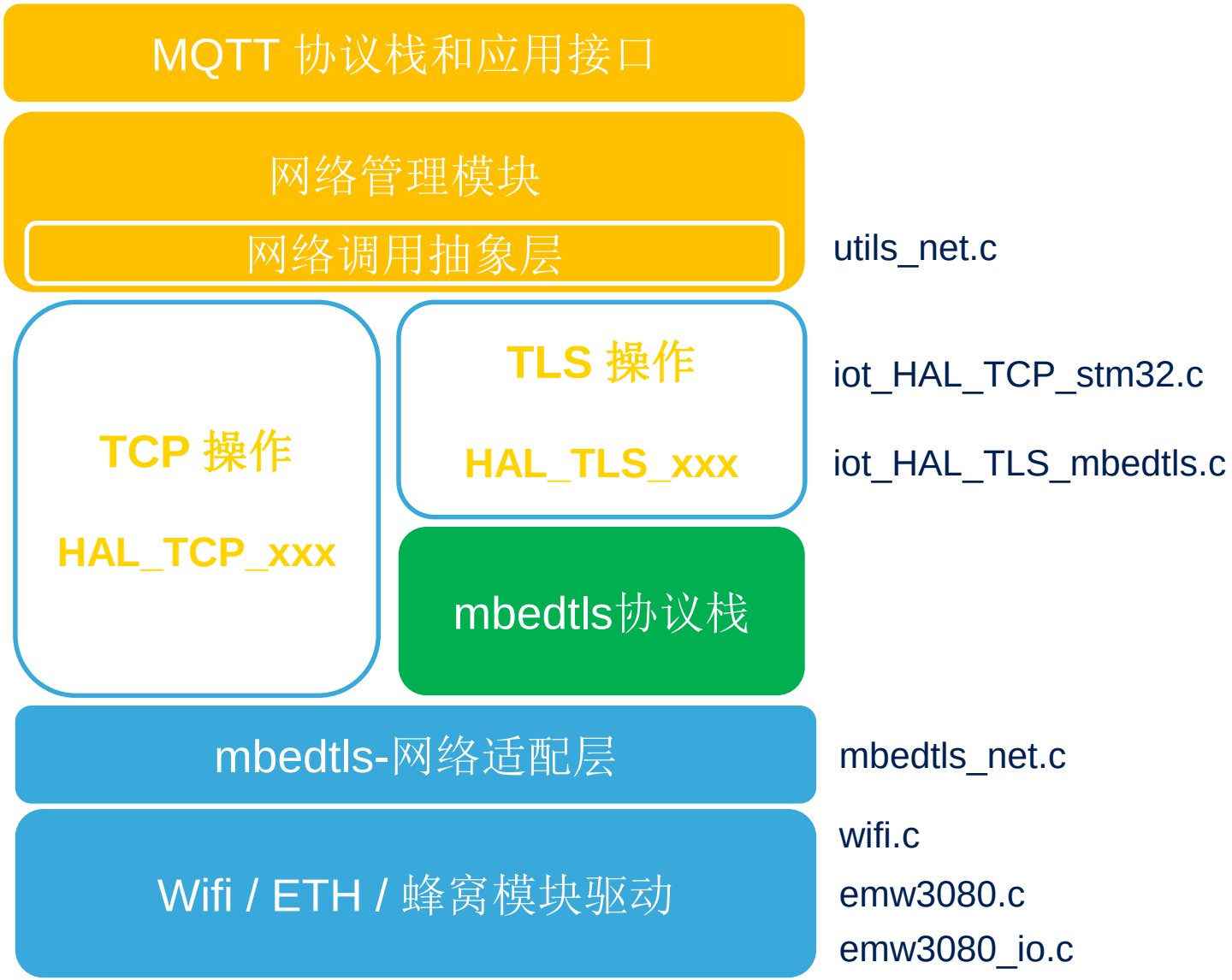


| 被调用的上下文 | iot_HAL_TCP_stm32.c | |
|---|---|---|
| Linkkit C-SDK中网络通信管理层 | HAL_TCP_Establish | 空<br>TCP/IP功能在Wifi模块里实现，调用AT指令控制wifi模块的对应工作 |
| | HAL_TCP_Destroy | |
| | HAL_TCP_Write | |
| | HAL_TCP_Read | |
| **被调用的上下文** | **iot_HAL_TLS_mbedtls.c** | |
| Linkkit C-SDK中网络通信管理层 | HAL_TLS_Establish | 协同mebdtls协议栈，wifi驱动实现安全的TCP通信 |
| | HAL_TLS_Destroy | |
| | HAL_TLS_Read | |
| | HAL_TLS_Write | |
| **被调用的上下文** | **iot_HAL_OS_stm32.c** | |
| | HAL_Random | STM32的硬件随机数单元 |
| 在HAL_Random前调一次 | HAL_Srandom | 空 |

# 网络接口文件的适配



| TCP通道操作 @ iot_HAL_TCP_stm32.c | |
|---|---|
| 建立连接 | HAL_TCP_Establish |
| 销毁连接 | HAL_TCP_Destroy |
| 读 | HAL_TCP_Read |
| 写 | HAL_TCP_Write |
| **TLS通道操作 @ iot_HAL_TLS_mbedtls.c** | |
| 建立连接 | HAL_TLS_Establish |
| 销毁连接 | HAL_TLS_Destroy |
| 读 | HAL_TLS_Read |
| 写 | HAL_TLS_Write |

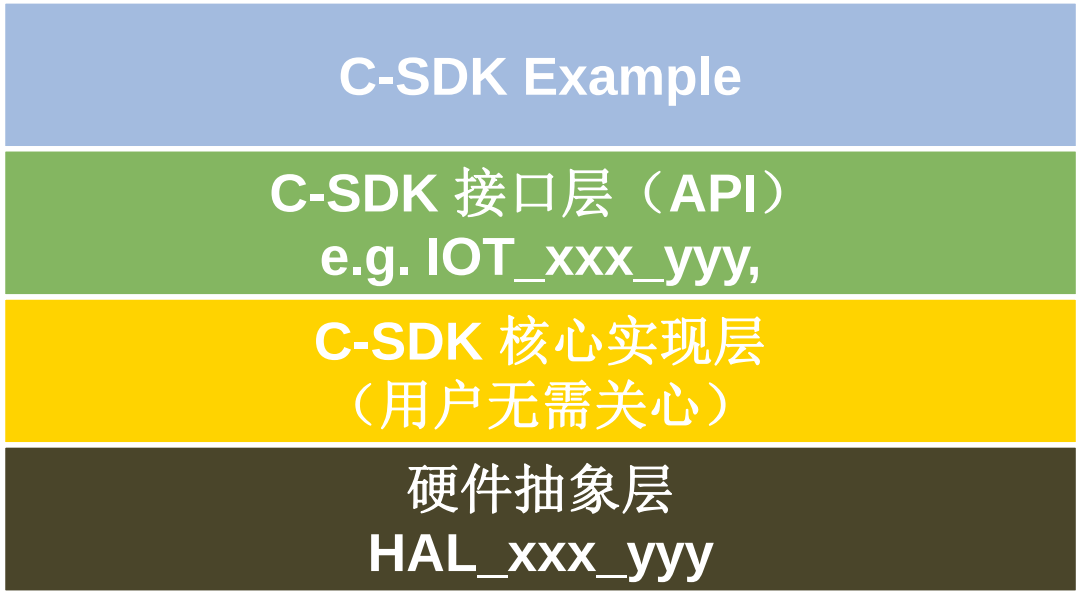



life.augmented | 阿里云

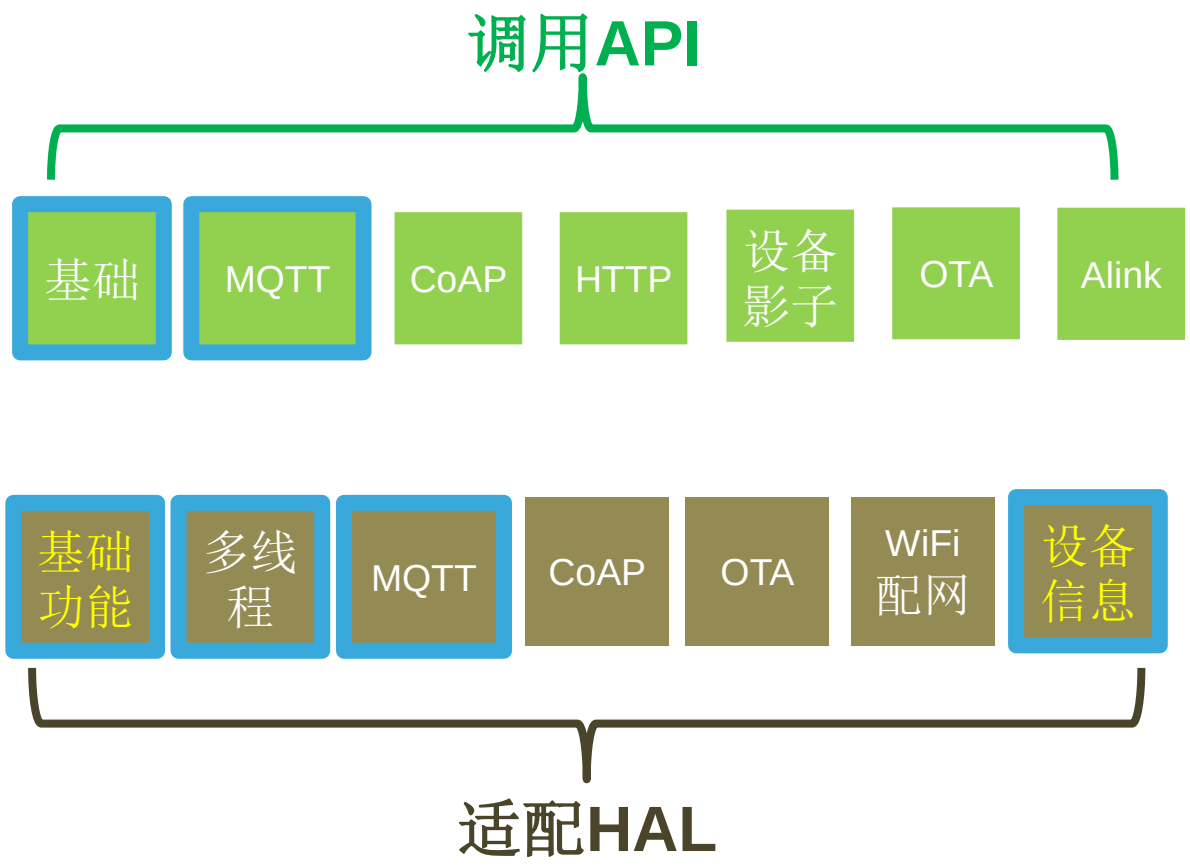
# C-SDK的适配和调用



- 设备端C-SDK的分层




SDK中提供多个mqtt例程




SDK中提供Linux桌面OS下的实现参考

# 【基础】相关API

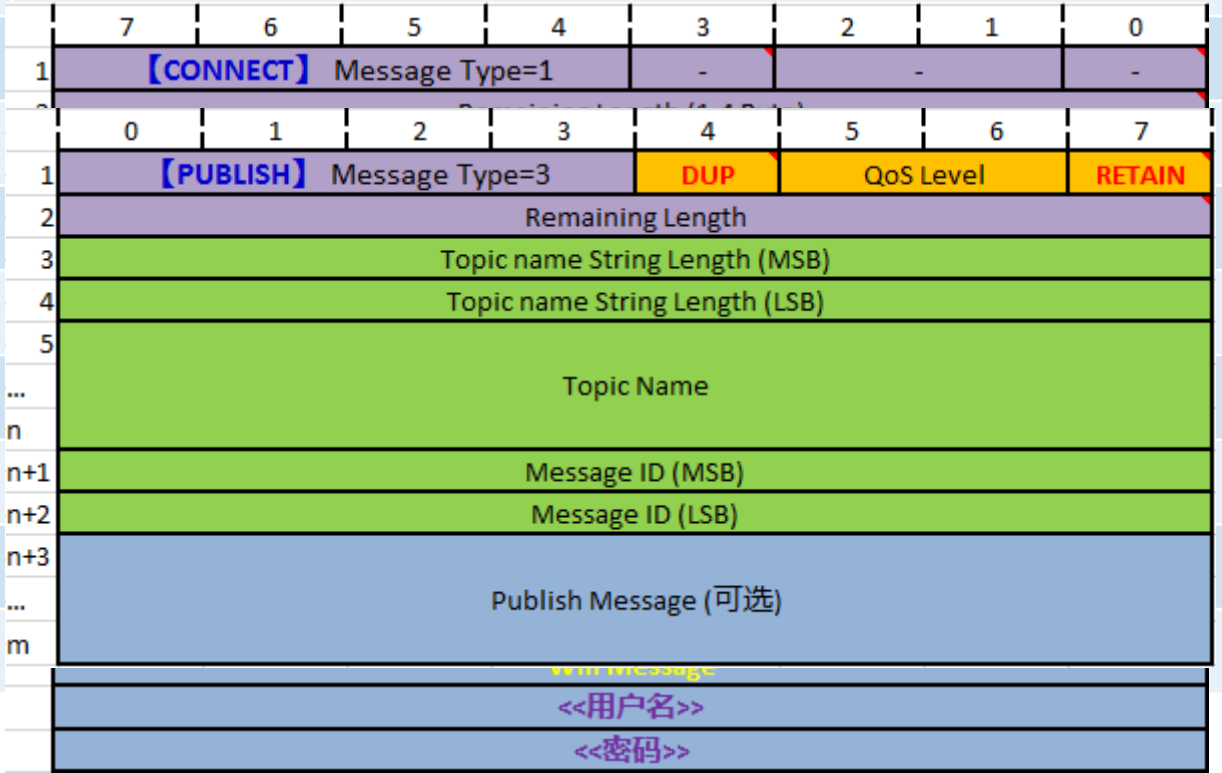


| src\sdk-impl\sdk-impl.c | | |
|---|---|---|
| IOT_OpenLog | #define IOT_OpenLog(arg)<br>#define IOT_CloseLog() IOT_SetLogLevel(IOT_LOG_NONE) | |
| IOT_CloseLog | | |
| IOT_SetLogLevel (LogLevel) | | |
| IOT_DumpMemoryStats (LogLevel) | | |
| IOT_SetupConnInfo | 输入参数 | 输出参数 |
| 设备认证：根据三元组信息计算MQTT连接阿里云IoT平台的参数 | * product_key | typedef struct {<br>uint16_t port;<br>char host_name[HOST_ADDRESS_LEN + 1];<br>char client_id[CLIENT_ID_LEN + 1];<br>char username[USER_NAME_LEN + 1];<br>char password[PASSWORD_LEN + 1];<br>const char *pub_key;<br>} iotx_conn_info_t, *iotx_conn_info_pt |
| | * device_name | |
| | * device_secret | |
| IOT_Ioctl | | |
| IOTX_IOCTL_SET_DOMAIN | IOTX_CLOUD_DOMAIN_SH/SG/JP/US/GER | |
| IOTX_IOCTL_SET_DYNAMIC_REGISTER | 0 – 关闭动态注册； 1 – 使能动态注册 | |

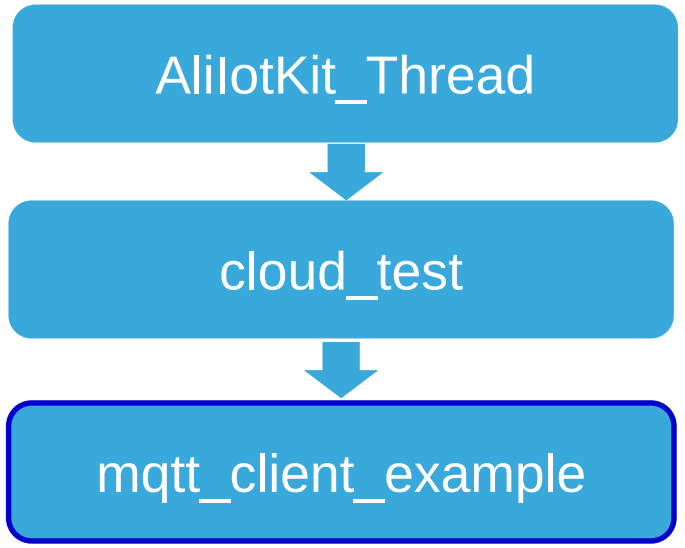
# 【MQTT】相关API



| src/protocol/mqtt/client/mqtt_client.c | | | |
|---|---|---|---|
| IOT_MQTT_Construct | | | |
| port | host | clean_session | request_timeout_ms |
| client_id | username | keeplive_interval_ms | write_buf_size |
| password | pub_key | read_buf_size | handle_event |
| IOT_MQTT_CheckStateNormal | | | |
| IOT_MQTT_Publish | | | |
| IOT_MQTT_Publish_Simple | | | |
| IOT_MQTT_Subscribe | | | |
| IOT_MQTT_Subscribe_Sync | | | |
| IOT_MQTT_Unsubscribe | | | |
| IOT_MQTT_Yield | | | |
| IOT_MQTT_Destroy | | | |




life.augmented | 阿里云




main.c

main.c

mqtt_example.c

1. /* 设备认证 */
2. /* 初始化 MQTT参数*/
3. /* 根据参数构建 MQTT 客户端实例*/
4. /* 订阅业务关心的消息主题 */
5. /* 发布业务关心的消息主题 */
6. /* 等待消息的到来并处理*/

IOT_SetupConnInfo @ sdk-impl.c

IOT_MQTT_Construct
IOT_MQTT_Subscribe
IOT_MQTT_Publish
IOT_MQTT_Yield
@ mqtt_client.c

# 第三章.第三节 内容简介



- 项目例程演示
  - 项目例程流程图
  - 演示视频
- Linkkit C-SDK介绍
  - Linkkit C-SDK
  - Linkkit C-SDK如何适配到STM32平台
  - Linkkit C-SDK的使用
- 项目例程介绍
  - 软件包和项目结构
  - 使用CubeMX生成系统初始化框架和代码
  - 网络通信的管理（网络通信抽象层和wifi驱动）
  - 使用Linkkit C-SDK连接阿里云IoT平台
  - 例程参数的存储及Sensor数据的读取

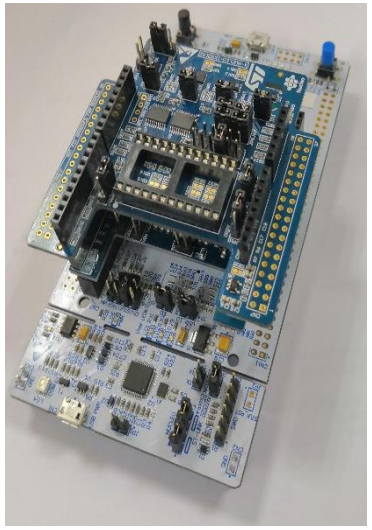
# 节点端系统框图



X-NUCLEO-IKS01A2 Sensor扩展板

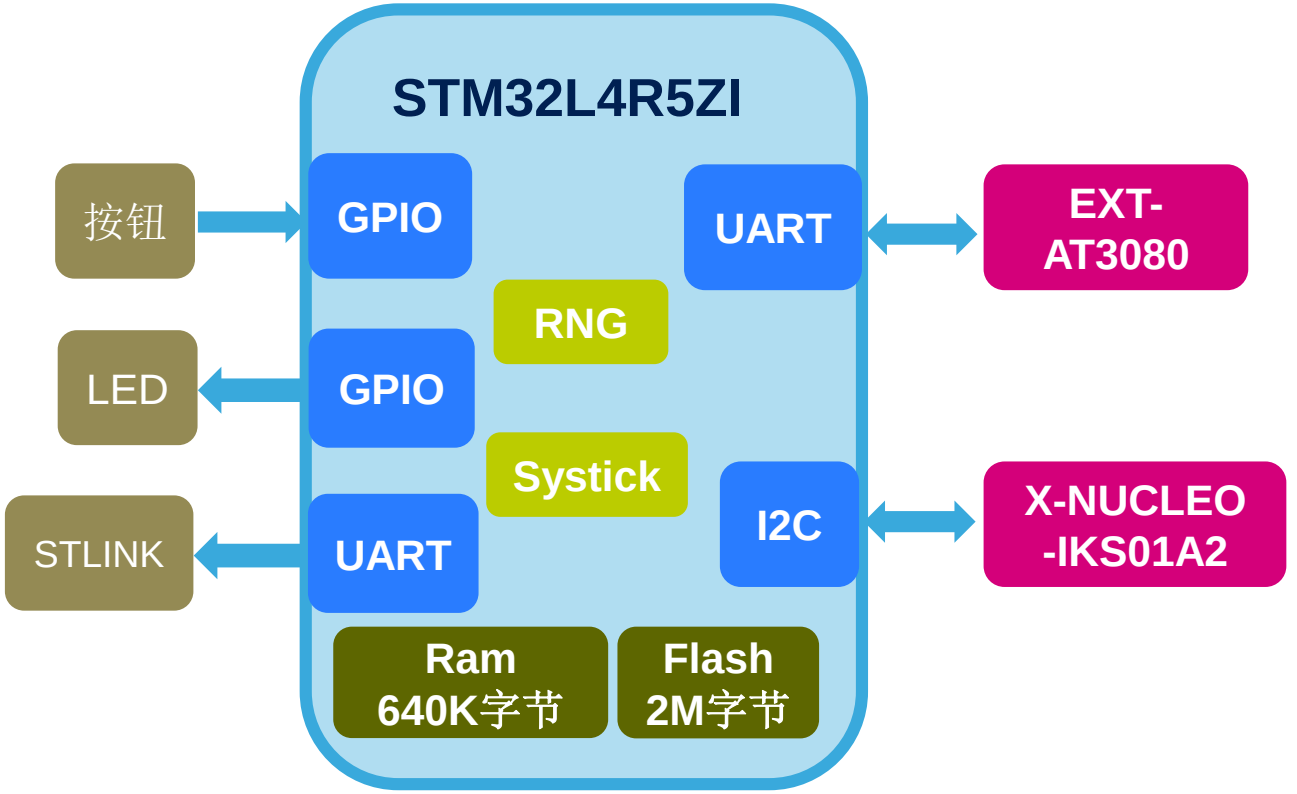
STM32L4R5ZI

Arduino Uno接口



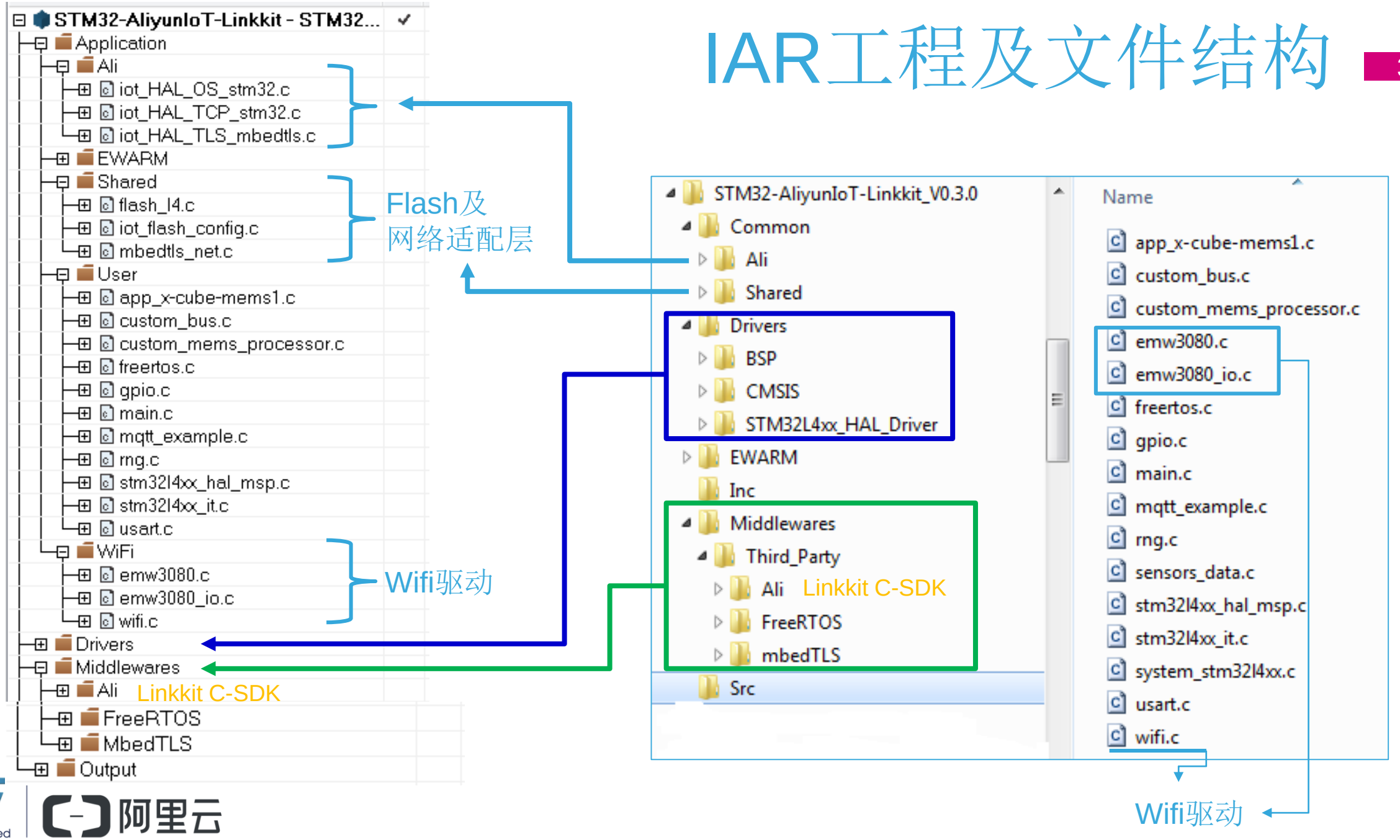
NUCLEO-L4R5ZI

EXT-AT3080 Wifi扩展板



# IAR工程及文件结构



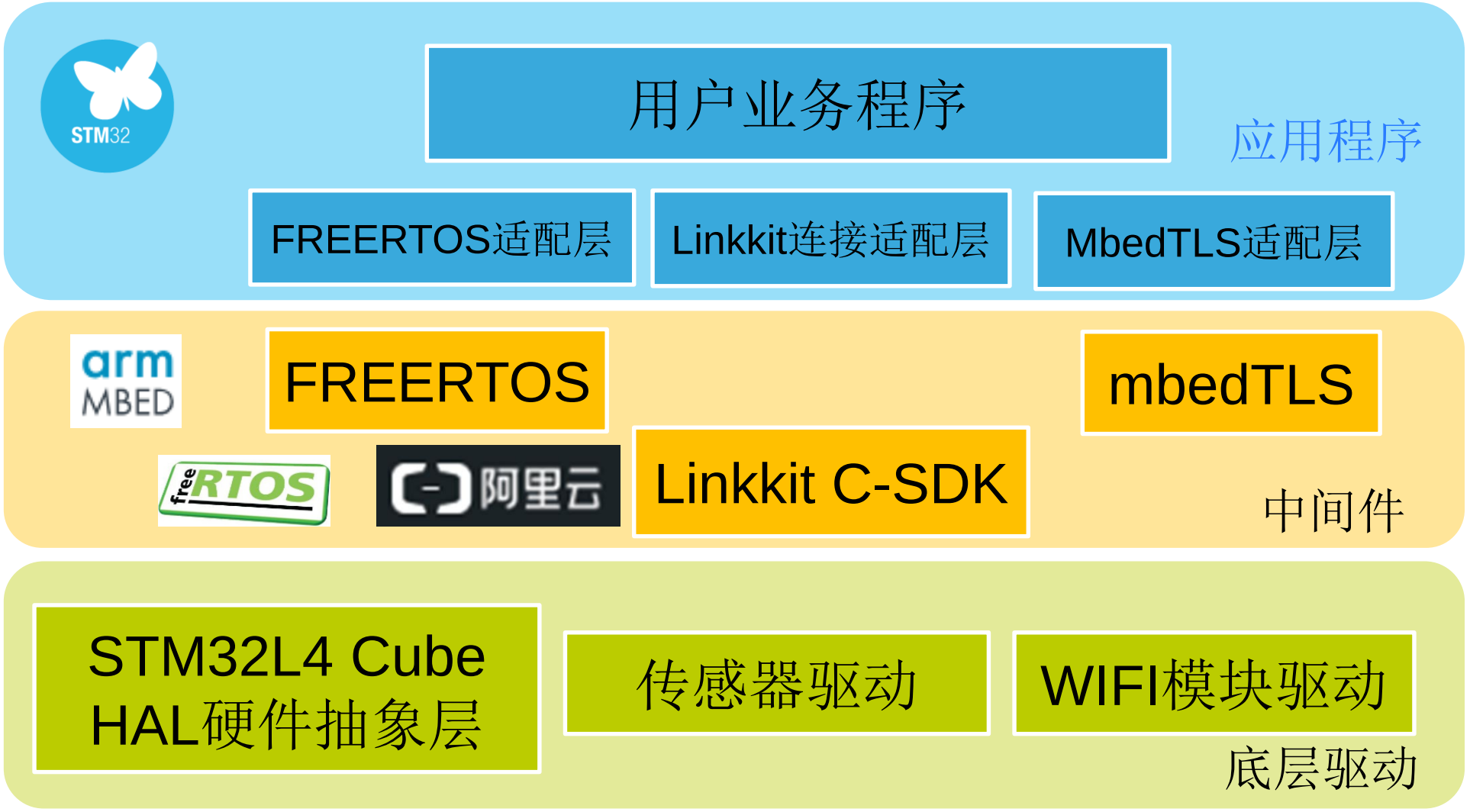



life.augmented

阿里云

# 项目例程软件架构




用户业务程序
应用程序
FREERTOS适配层
Linkkit连接适配层
MbedTLS适配层
FREERTOS
mbedTLS
Linkkit C-SDK
中间件
STM32L4 Cube
HAL硬件抽象层
传感器驱动
WIFI模块驱动
底层驱动

# MCU外设的使用



- 与WIFI扩展板的接口定义

| NCULEO板接口编号 | 引脚编号 | 引脚名 | STM32引脚 | 外设配置 |
|---|---|---|---|---|
| CN10 | 14 | D1 | PD8 | USART3_TX |
| CN10 | 16 | D0 | PD9 | USART3_RX |

- 与传感器扩展板的接口定义

| NCULEO板接口编号 | 引脚编号 | 引脚名 | STM32引脚 | 外设配置 |
|---|---|---|---|---|
| CN7 | 2 | D15 | PB8 | I2C1_SCL |
| CN7 | 4 | D14 | PB9 | I2C1_SDA |

- 虚拟串口接口定义

| | STM32引脚 | 外设配置 |
|---|---|---|
| 连到STLINK USB 虚拟串口 | PG7 | LPUART1_TX |
| | PG8 | LPUART1_RX |

- 与USER按键的接口定义

| NCULEO板 | STM32引脚 | 外设配置 | 功能 |
|---|---|---|---|
| 蓝色User按键 | PC13 | 外部中断，下降沿触发 | 切换允许流程至等待用户输入WIFI配网，三元组等信息 |

- 与LED灯的接口定义

| NCULEO板 | STM32引脚 | 外设配置 | 功能 |
|---|---|---|---|
| LD1(绿） | PC7 | GPIO 输出 | 每次上传温湿度信息时，闪烁一次 |
| LD3(红) | PB14 | GPIO 输出 | 高温报警提醒 |

- 片上外设

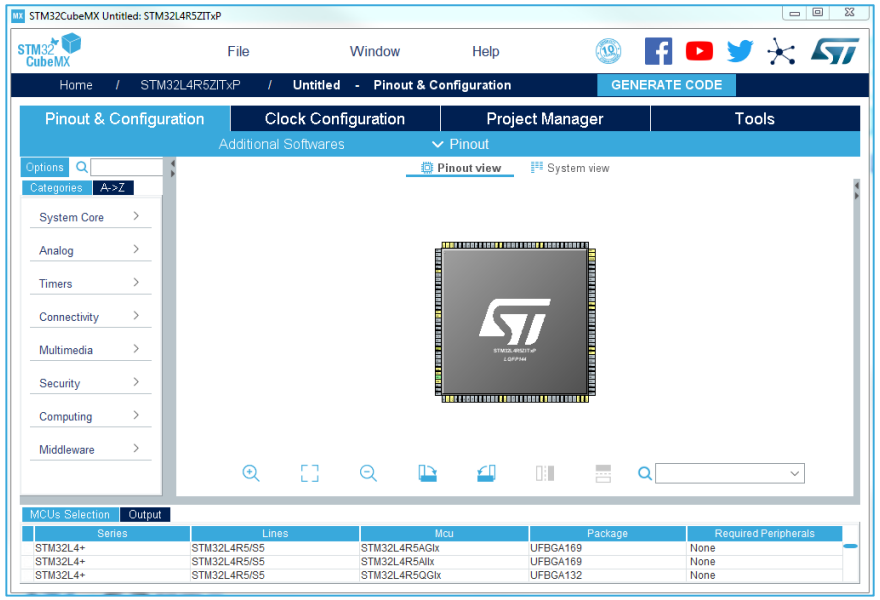
| NCULEO板 | STM32引脚 | STM32外设 | 功能 |
|---|---|---|---|
| N.A | N.A | Systick | FreeRTOS系统滴答 |
| N.A | N.A | RNG | 产生真随机数 |

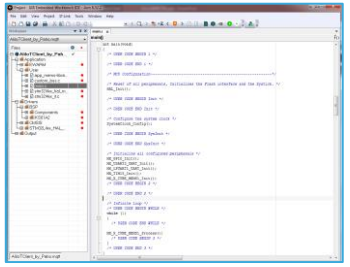
# 使用CubeMX初始化系统





步骤一：选择MCU型号




步骤三：生成初始工程

步骤二：

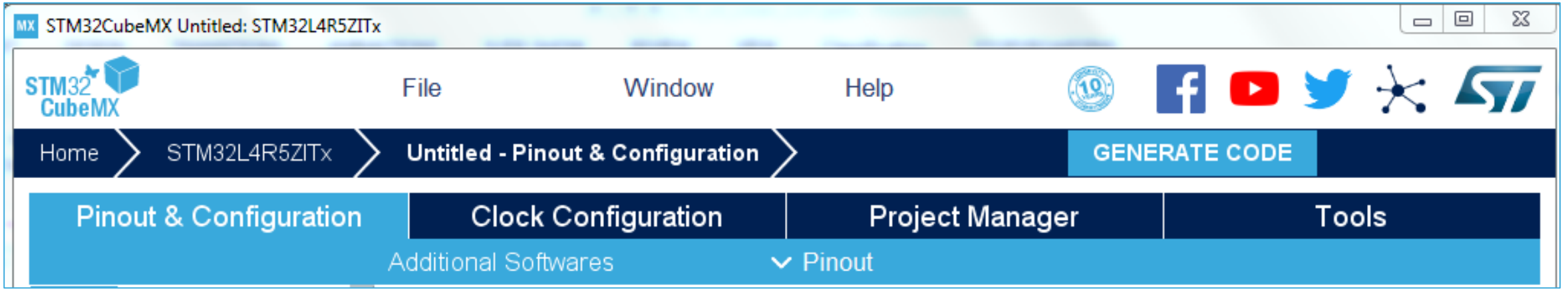
- 引脚/外设配置
  (UART
  /I2C
  /EXT)
- 时钟配置
- 插件配置
- FREERTOS配置

# 使用CubeMX初始化系统：步骤一

- MCU型号：STM32L4R5ZITx


# 使用CubeMX初始化系统：步骤一

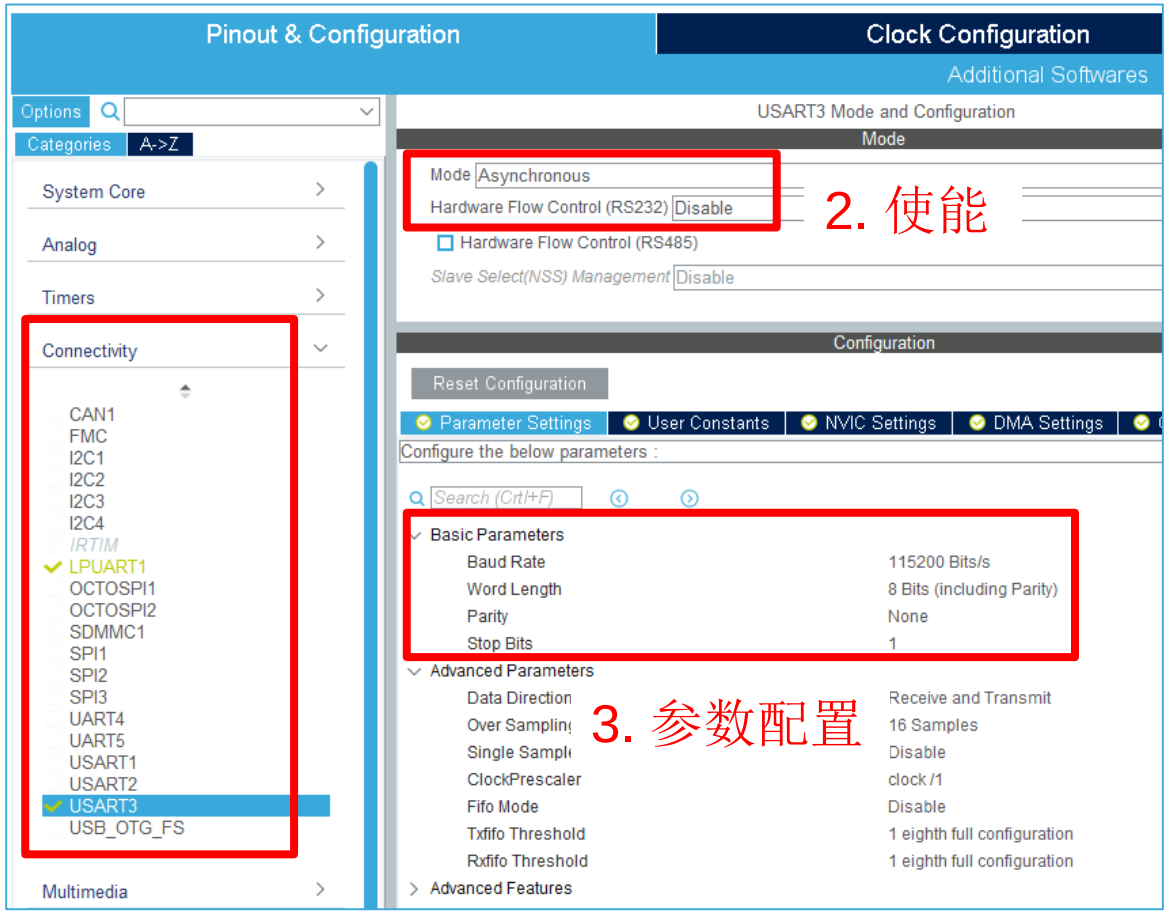





- 引脚和配置
  - 外设使能和配置
  - 外设在引脚上的映射
  - 中间件配置
- 时钟配置
  - 时钟树参数的智能配置
- 项目管理
  - 管理初始代码的生成
  - 管理初始项目的生成
- 工具
  - 功耗估计

# 使用CubeMX初始化系统：步骤二

- 分别使能两个串口
  - 与WIFI模块通信的USART3
  - 打印程序运行信息的LPUART1
- 参数配置
  - 波特率：115200
  - 数据长度：8bit
  - 1位停止位
  - 无奇偶校验



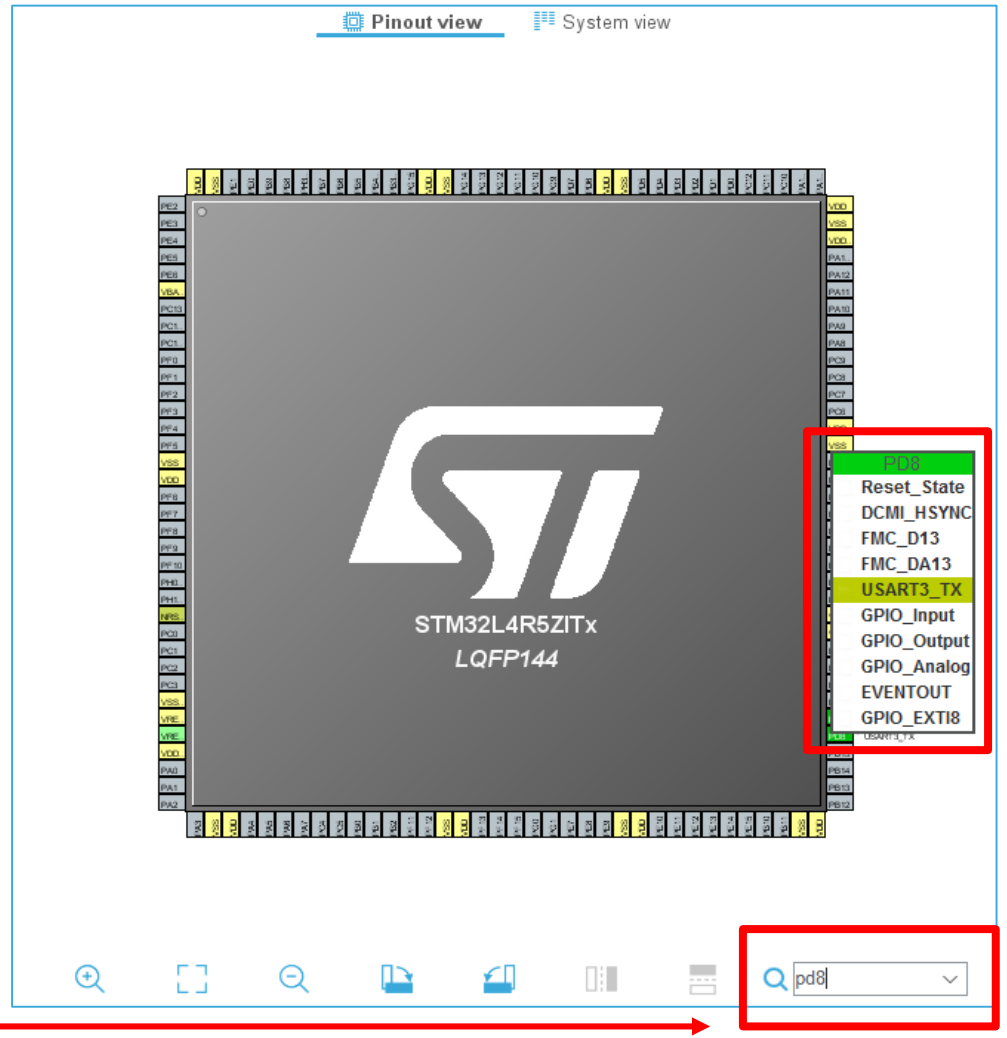

1. 选择目标串口外设

# 使用CubeMX初始化系统：步骤二

- 将功能落实到实际所用引脚上

| Wifi扩展板接口 | 外设配置 | CubeMX默认分配的引脚 | 例程实际使用的引脚 |
|---|---|---|---|
| D1 | USART3_TX | PC4 | PD8 |
| D0 | USART3_RX | PC5 | PD9 |

| | 外设配置 | CubeMX默认分配的引脚 | 例程实际使用的引脚 |
|---|---|---|---|
| STLINK虚拟串口 | LPUART1_TX | PC1 | PG7 |
| | LPUART1_RX | PC0 | PG8 |




2. 选择该引脚上的“串口发送”功能

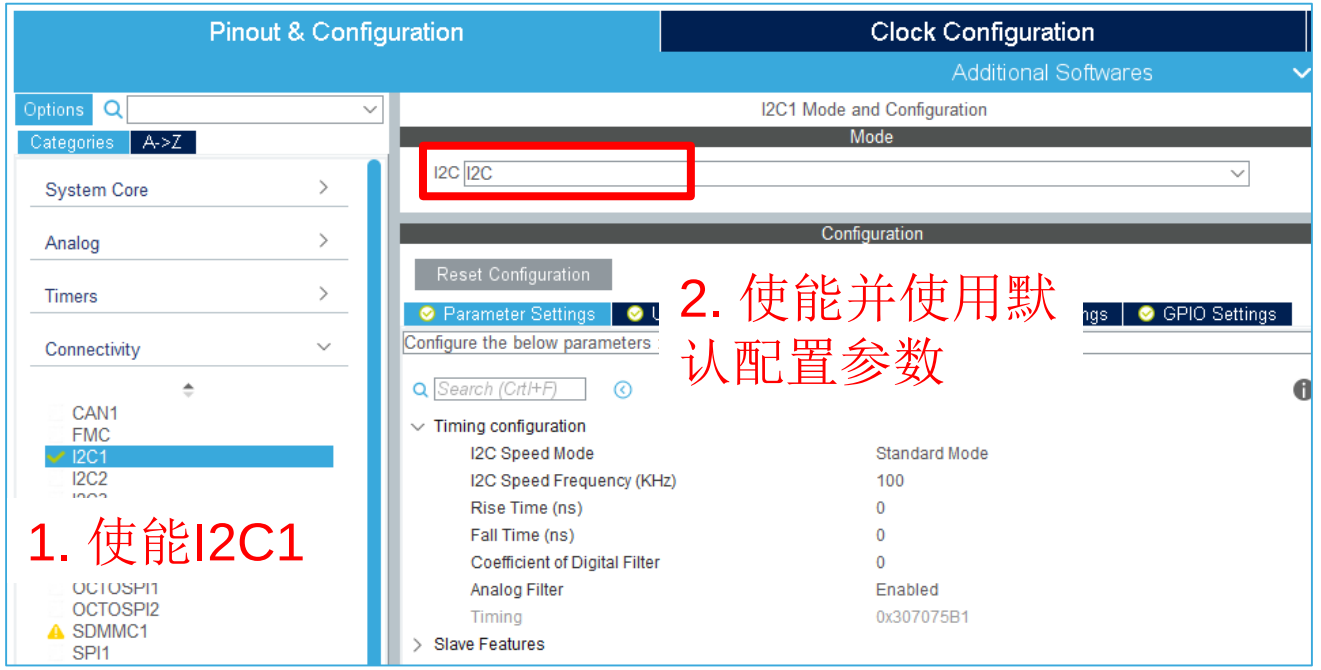
1. 查找需要配置的引脚

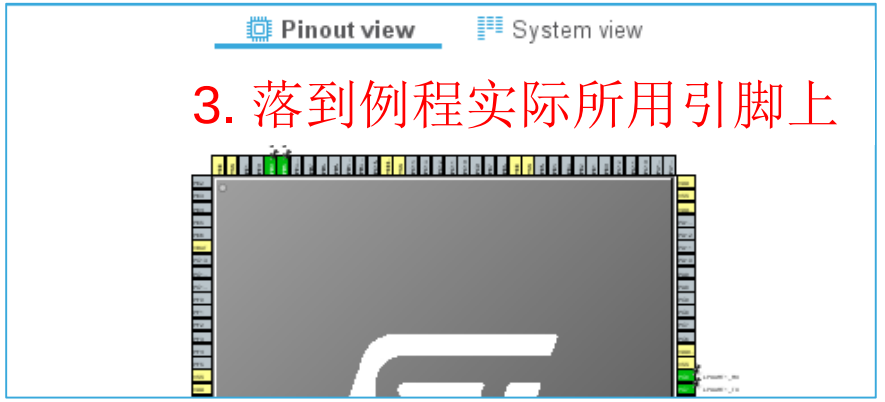
以UART3.TX @ PD.8为例

# 使用CubeMX初始化系统：步骤二

- 使能I2C
  - 与传感器扩展板通信的I2C1
- 参数配置
  - 使用默认配置即可
- 修改引脚定义

| Sensor扩展板接口 | 外设配置 | CubeMX默认分配的引脚 | 例程实际使用的引脚 |
|---|---|---|---|
| | I2C1_SDA | PG13 | PB9 |
| | I2C1_SCL | PG14 | PB8 |

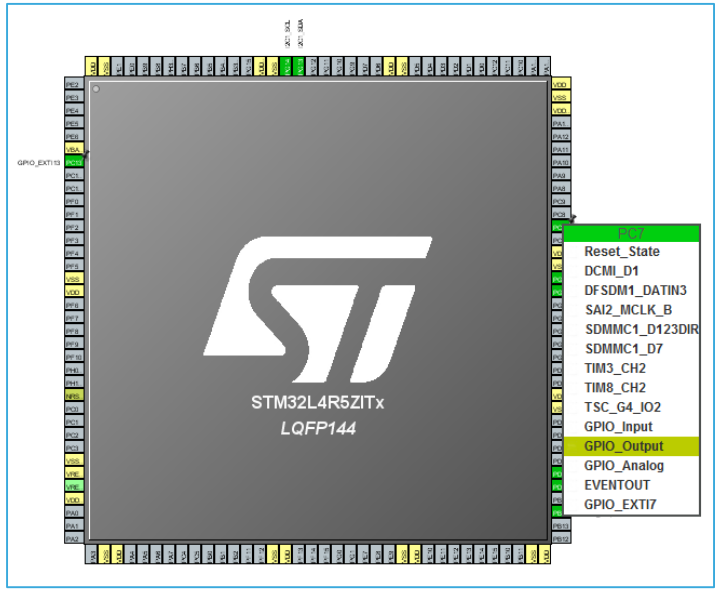


# 使用CubeMX初始化系统：步骤二

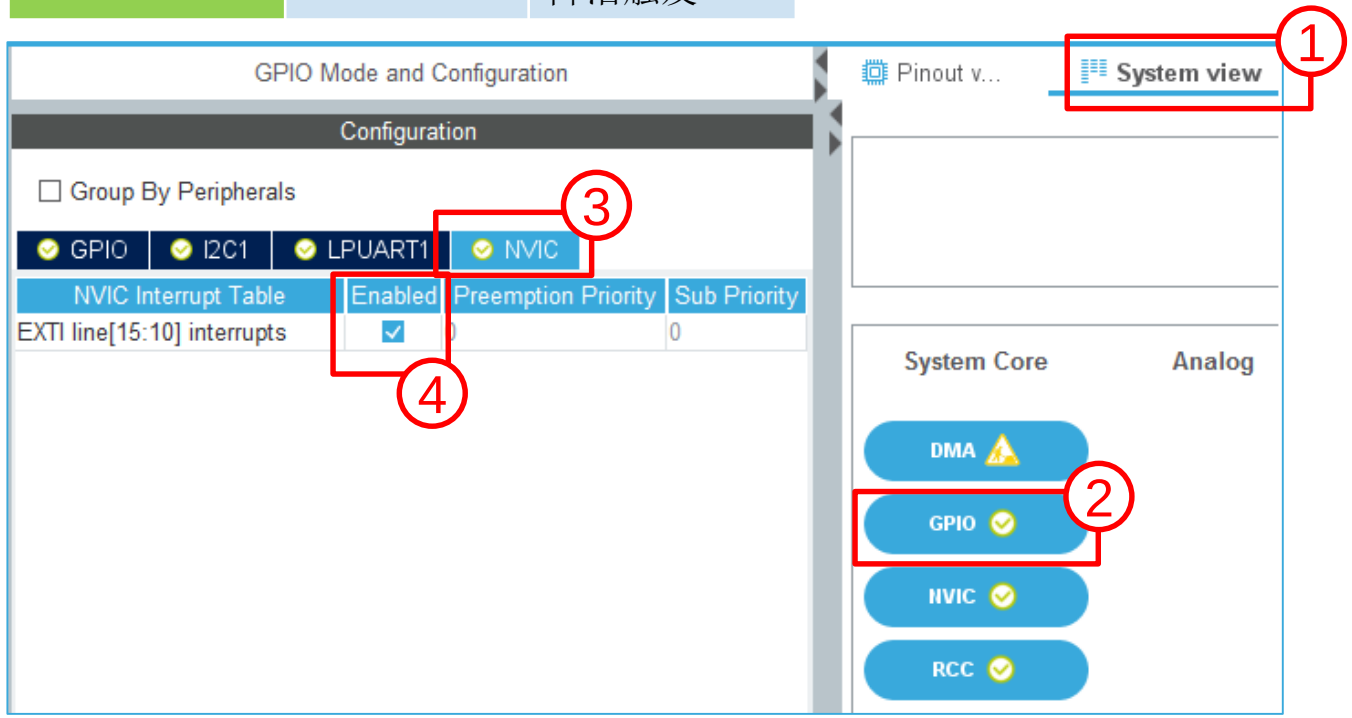
- 控制LED灯
  - PC7和PB14配置成：GPIO_Output

| NCULEO板 | STM32引脚 | 外设配置 |
|---|---|---|
| LD1(绿） | PC7 | GPIO 输出 |
| LD3(红) | PB14 | GPIO 输出 |




- 控制USER按键
  - PC13配置成：GPIO_EXTI13
  - 使能EXT13外部中断

| NCULEO板 | STM32引脚 | 外设配置 |
|---|---|---|
| 蓝色User按键 | PC13 | 外部中断，下降沿触发 |

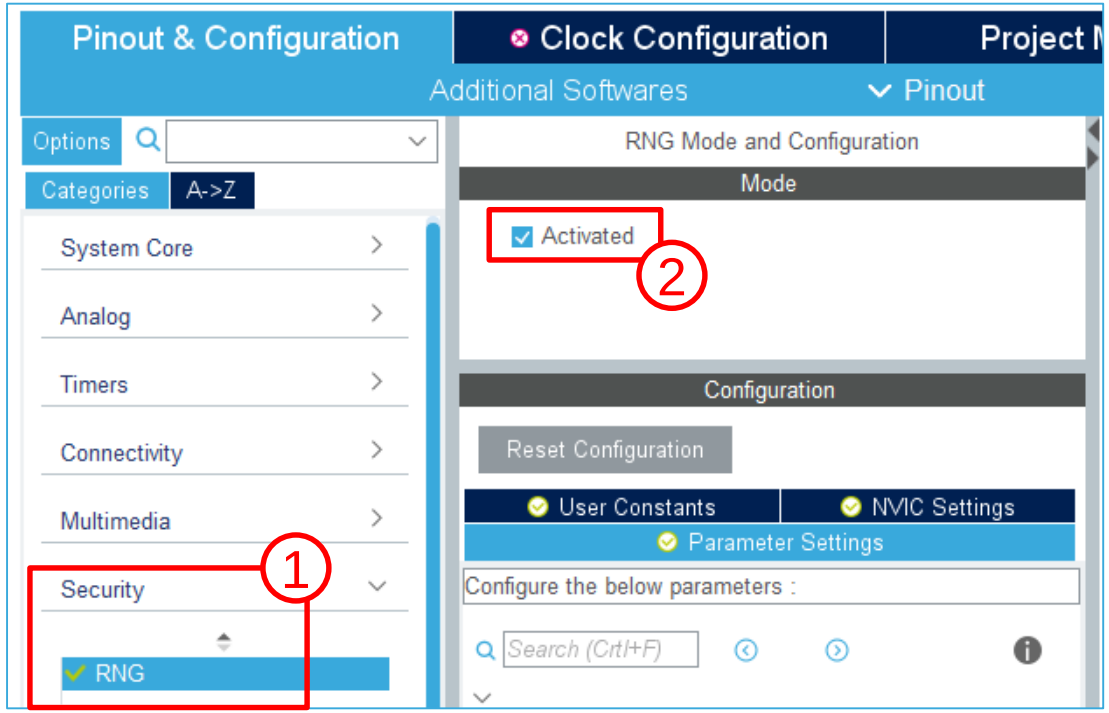



life.augmented
阿里云

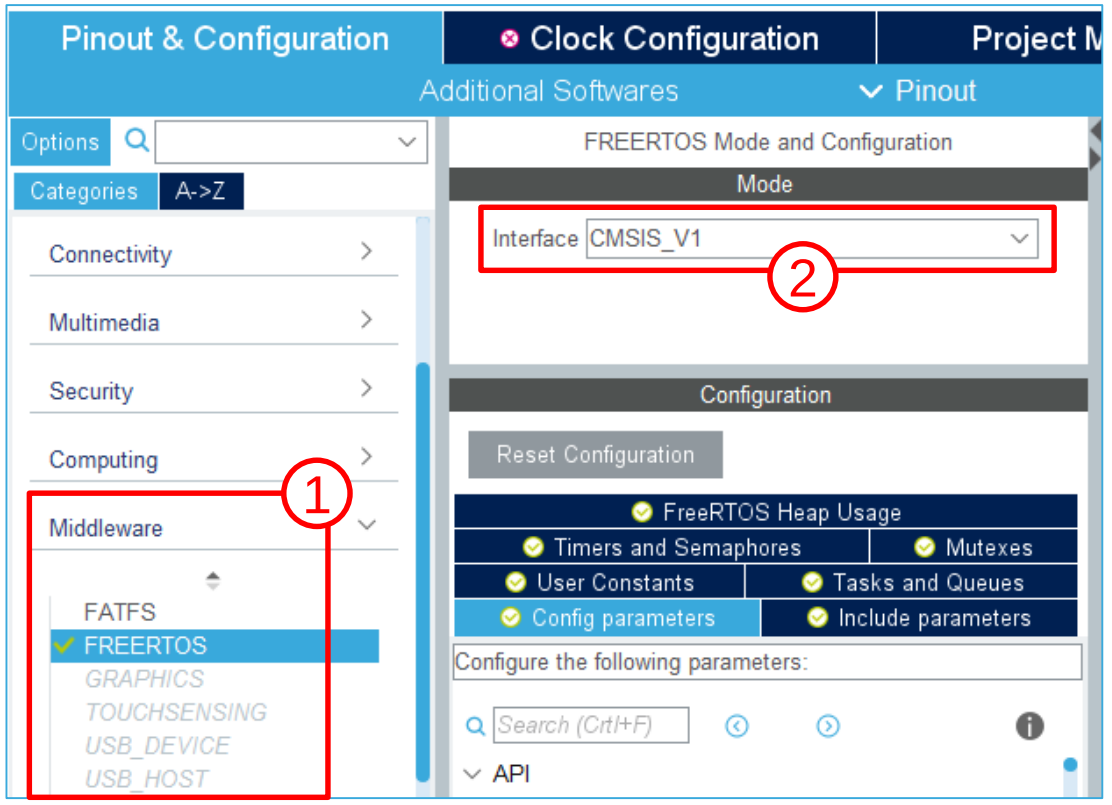
# 使用CubeMX初始化系统：步骤二



- 使能RNG硬件模块




- 使能FreeRTOS软件模块（中间件）

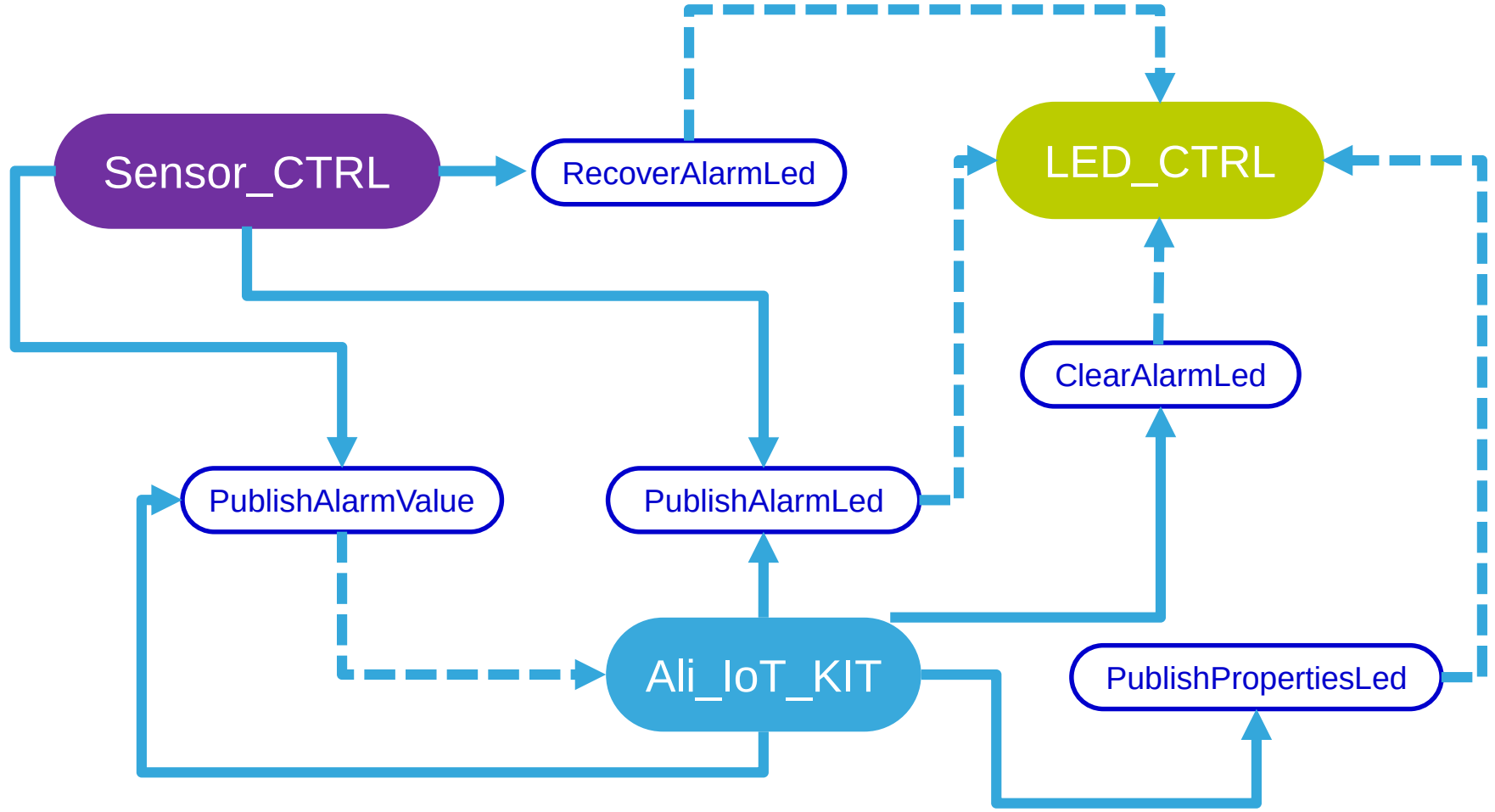


# 规划FreeRTOS的使用



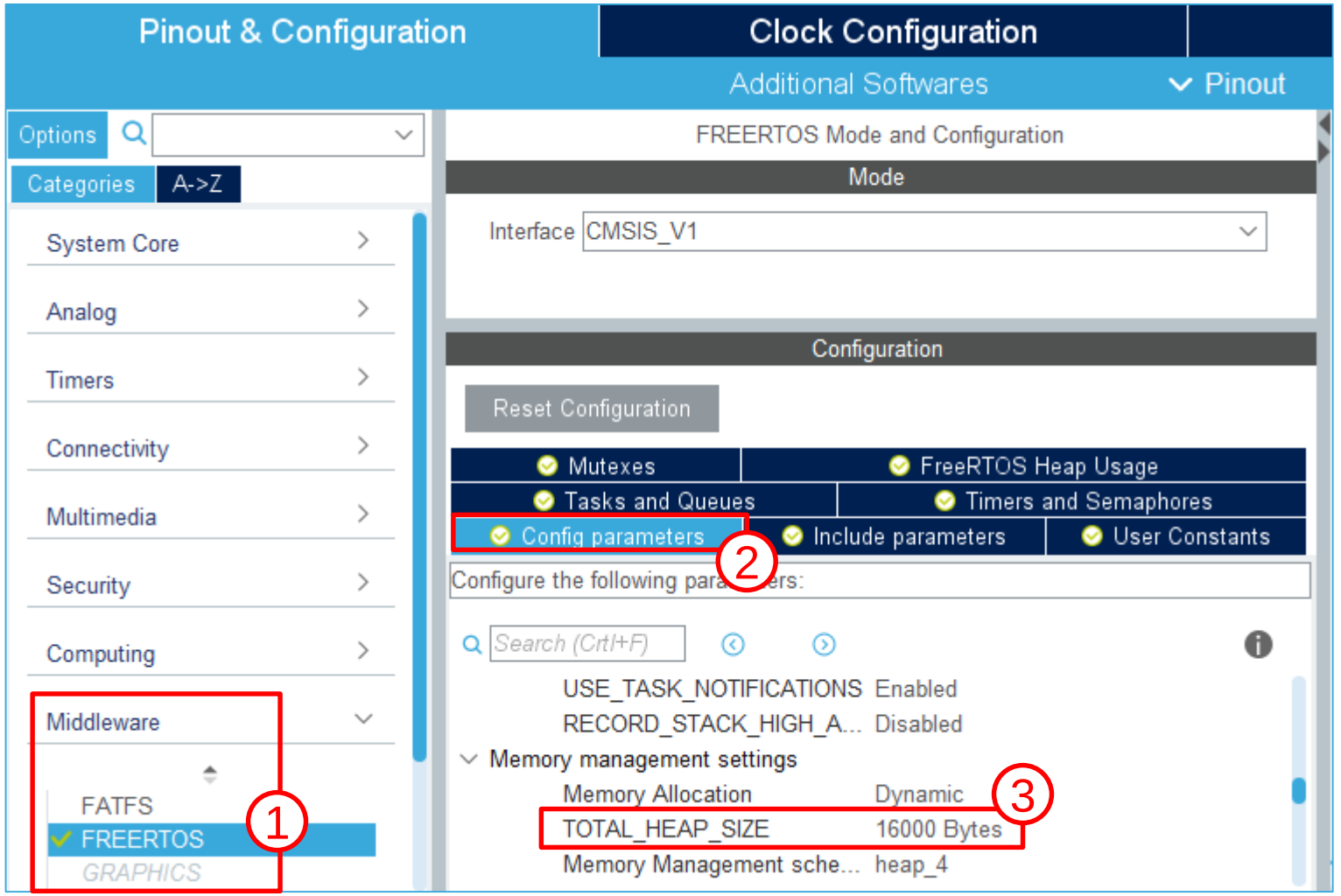
- 3个应用任务
  - Ali_IoT_KIT
  - LED_CTRL
  - Sensor_CTRL
- 5个同步信号量
  - PublishPropertiesLed
  - PublishAlarmValue
  - PublishAlarmLed
  - RecoverAlarmLed
  - ClearAlarmLed

- 任务间的同步



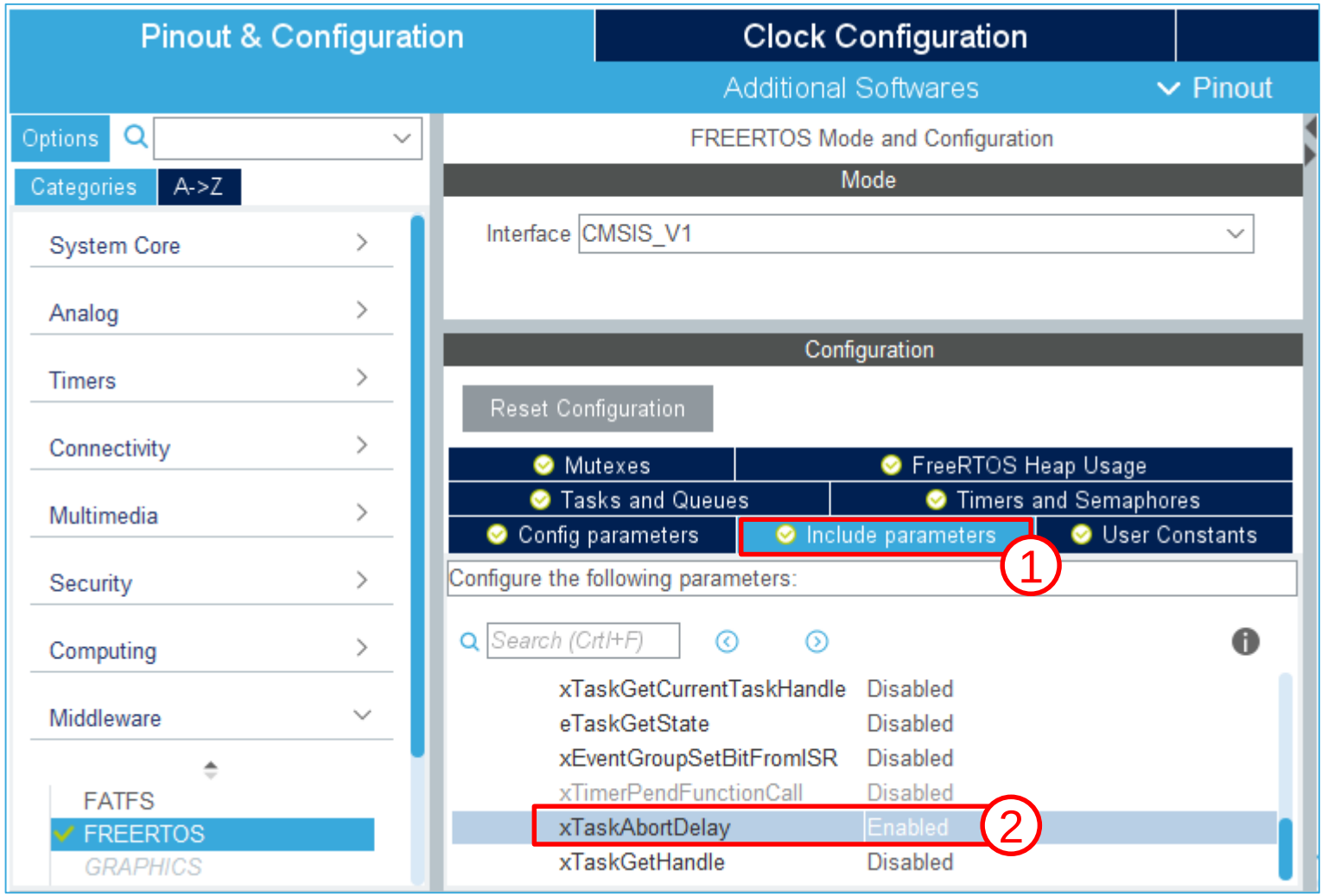
# 使用CubeMX初始化系统：步骤二

- 配置FREERTOS的堆大小



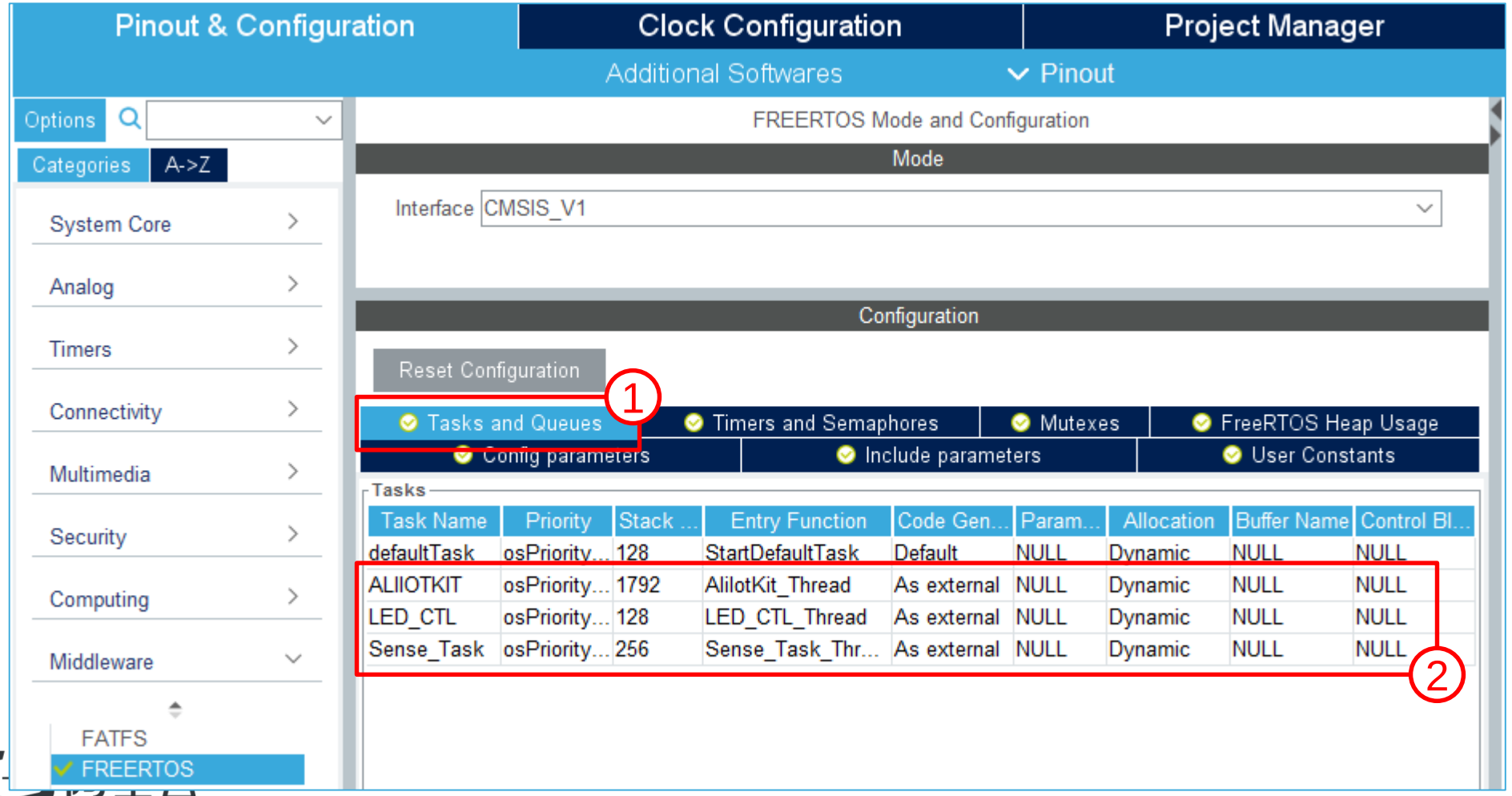
# 使用CubeMX初始化系统：步骤二

- 配置FREERTOS：使能xTaskAbortDelay函数



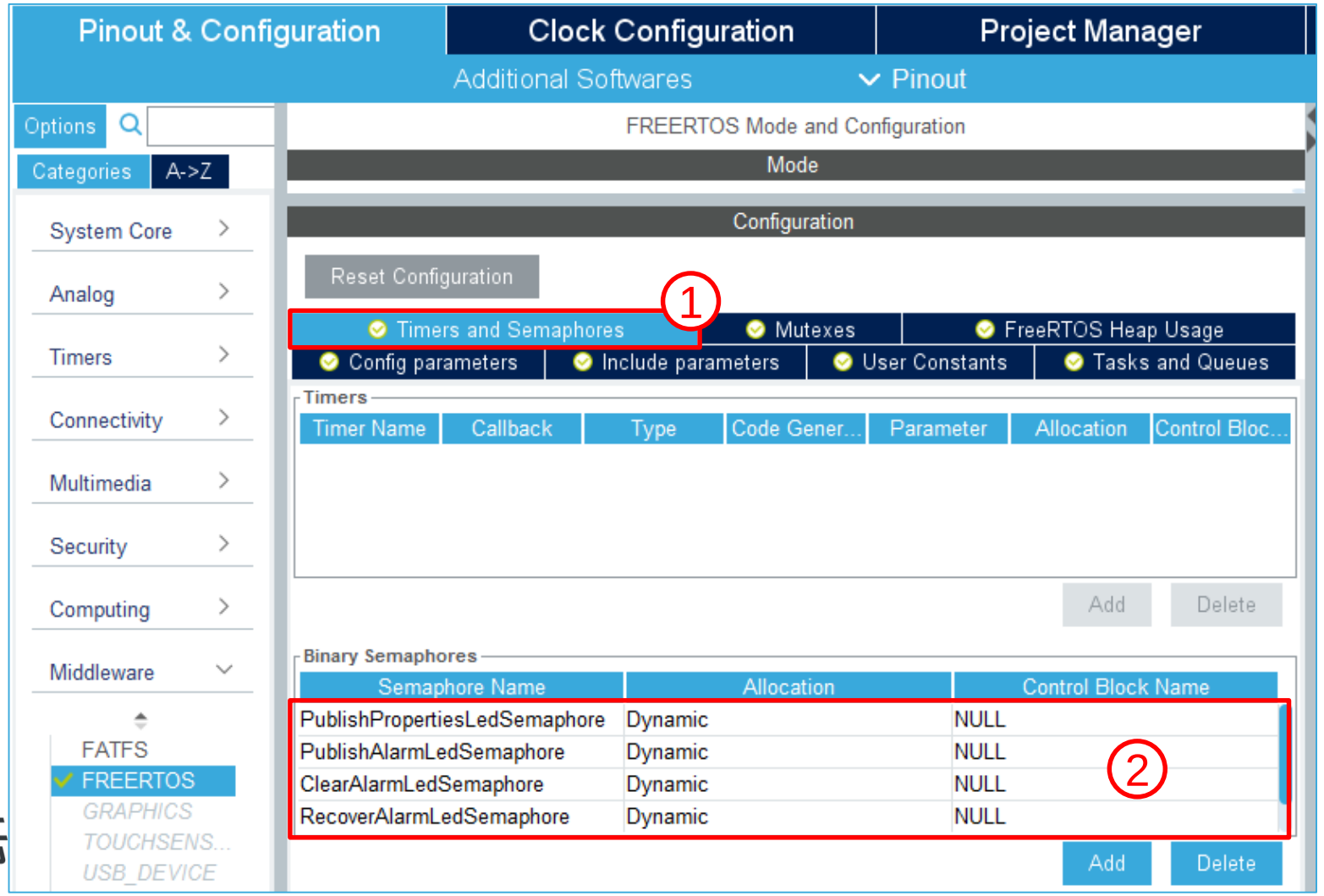
# 使用CubeMX初始化系统：步骤二

- 创建任务：任务名/优先级/任务堆栈/任务函数/函数声明



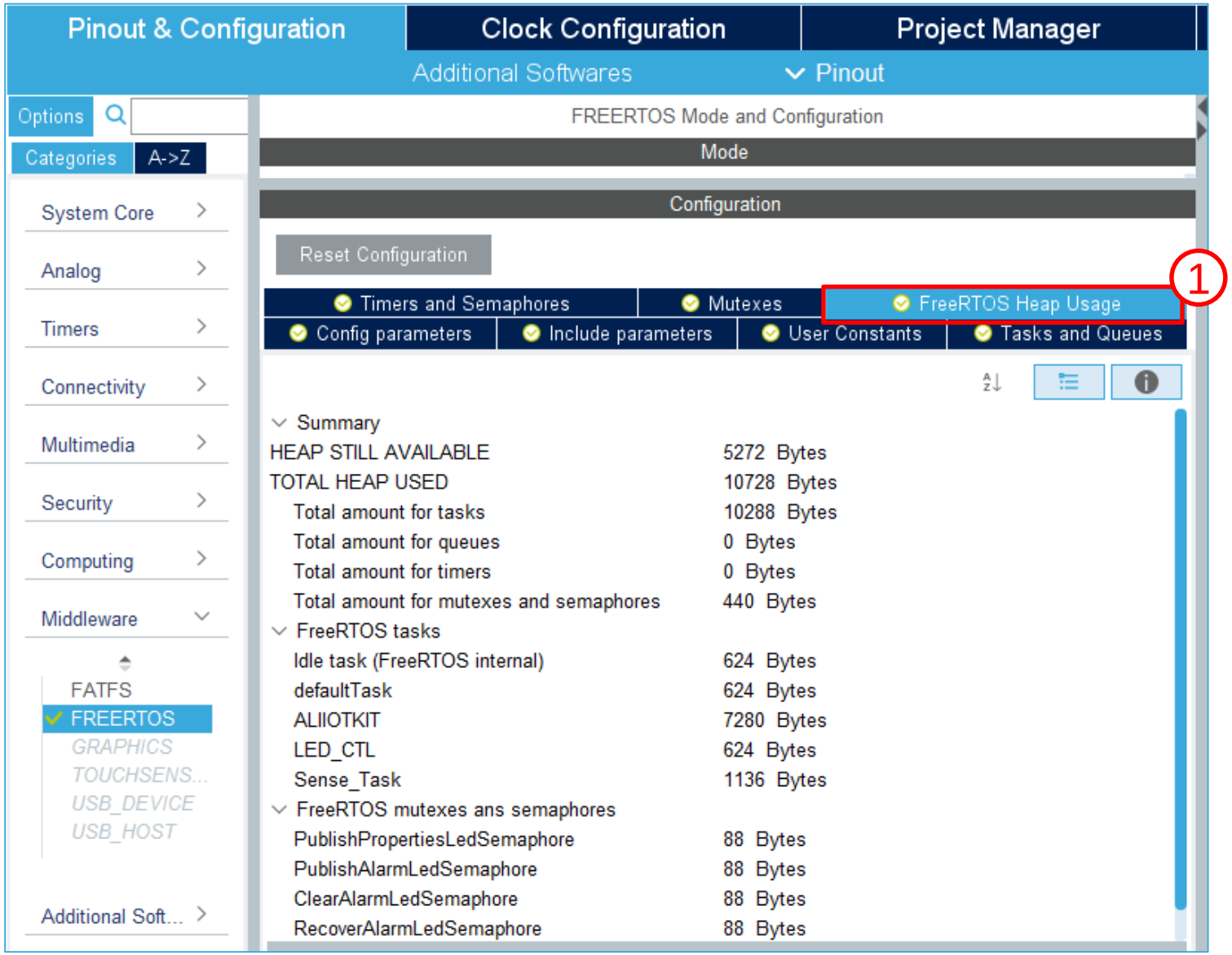
# 使用CubeMX初始化系统：步骤二

- 创建用于任务同步的信号量



# FREERTOS内存使用情况

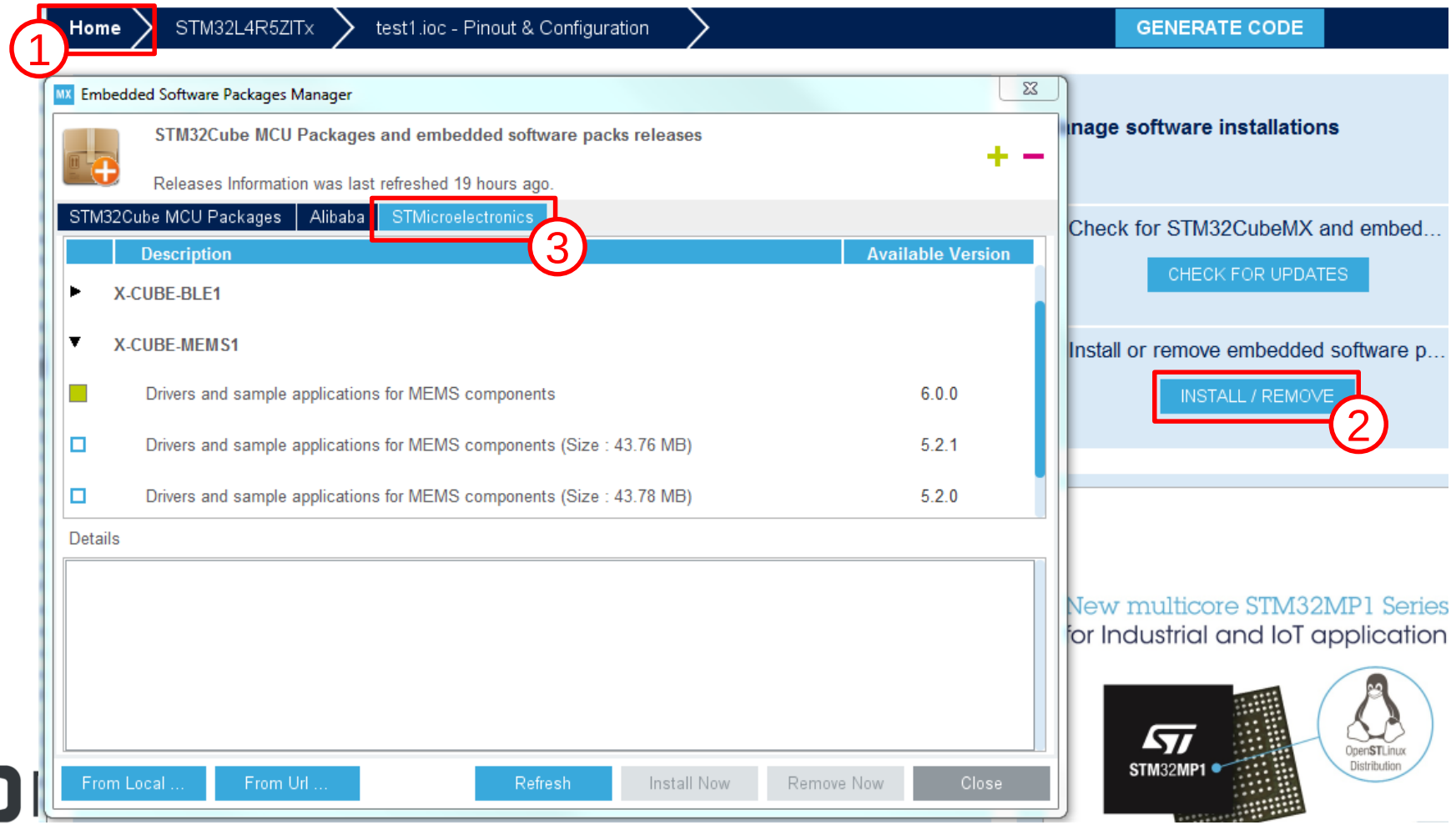




# 使用CubeMX初始化系统：步骤二



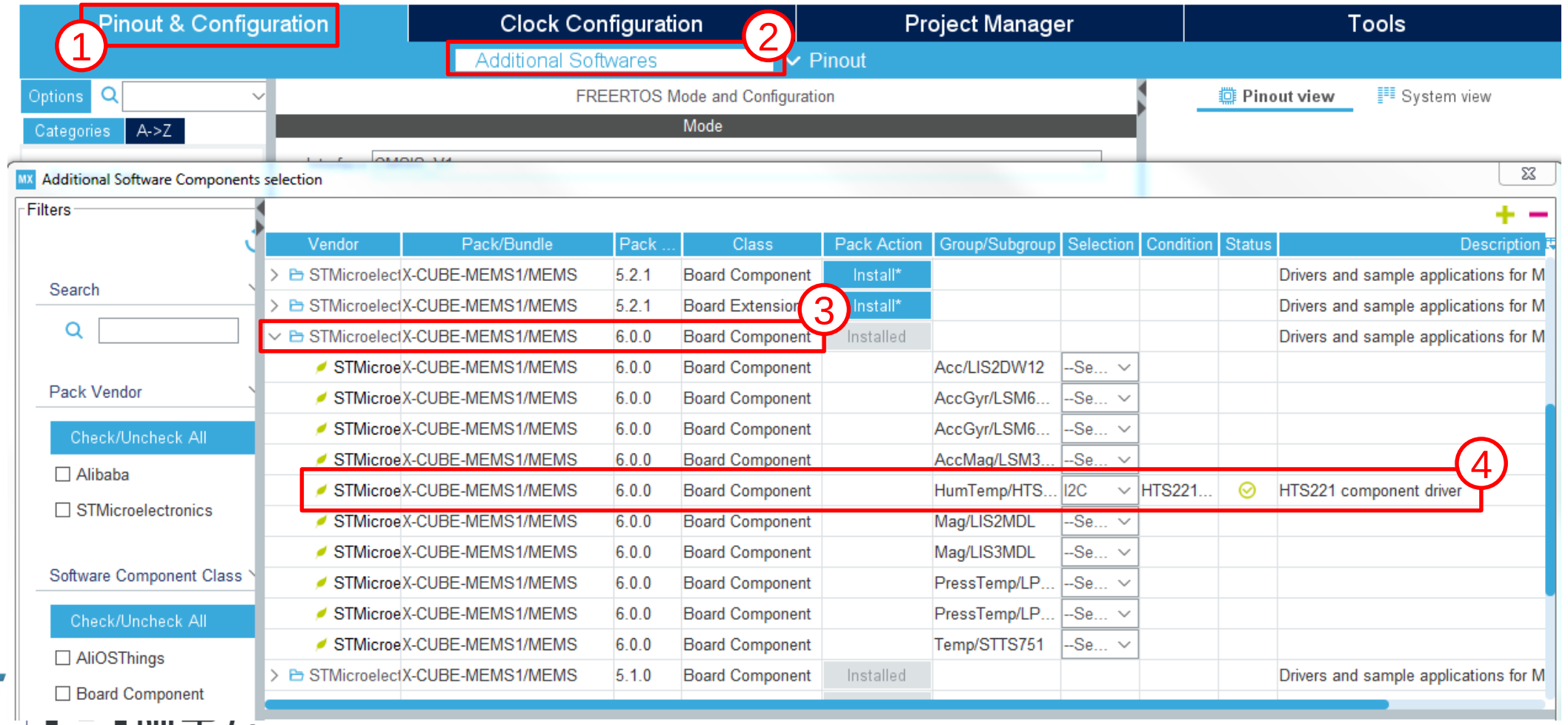
- 安装MEMS传感器驱动（CubeMX插件）



# 使用CubeMX初始化系统：步骤二

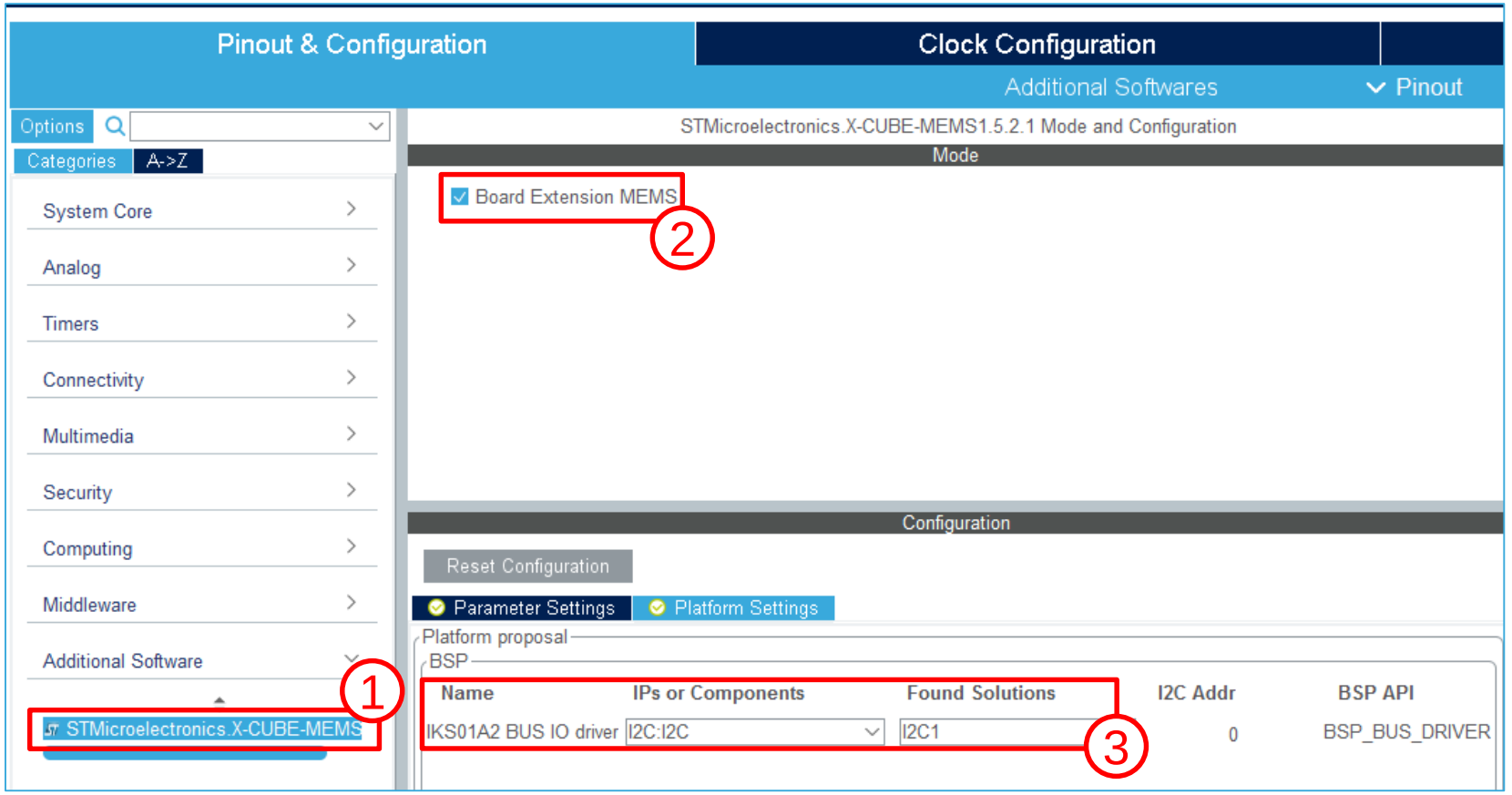


- 选择所需要的传感器（HTS221）及其驱动



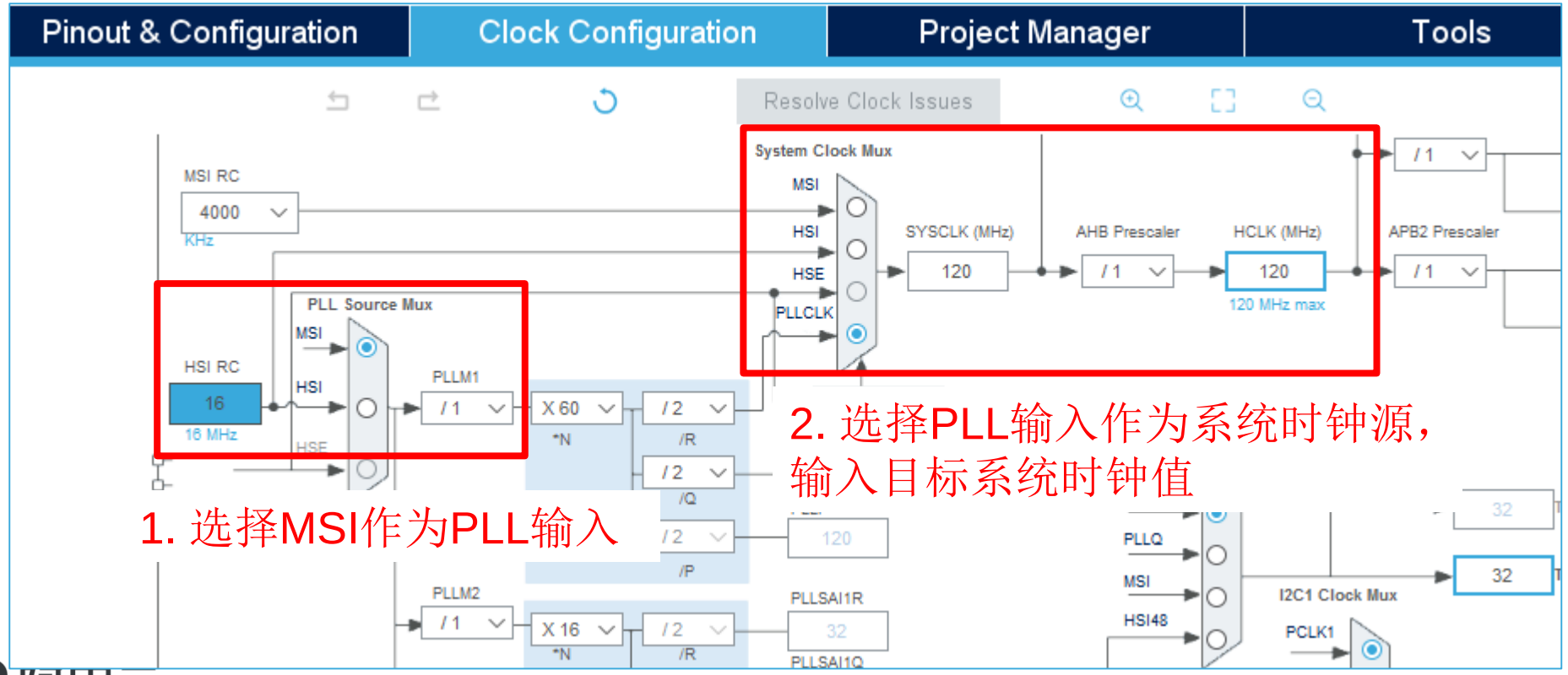
# 使用CubeMX初始化系统：步骤二

- 对温湿度传感器驱动进行BSP配置：使用I2C1



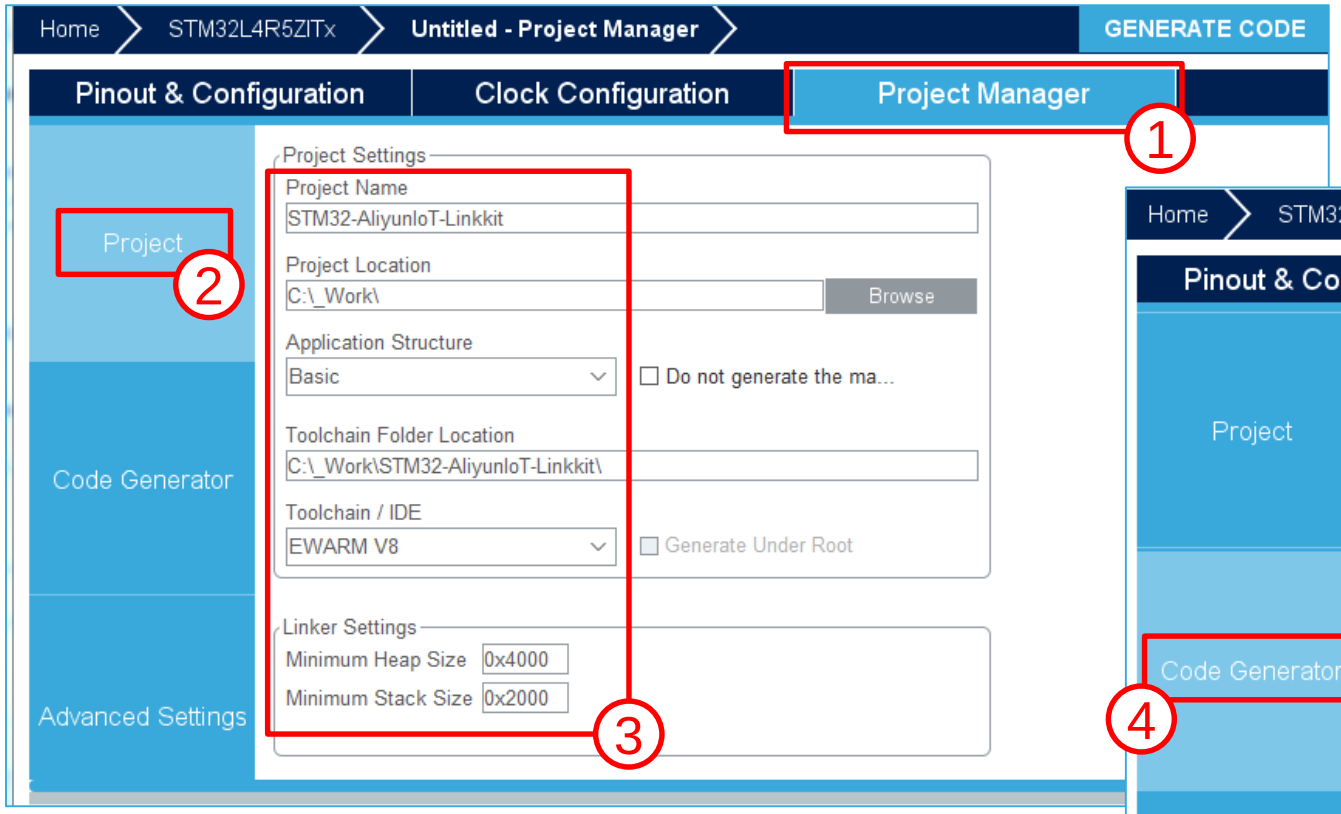
# 使用CubeMX初始化系统：步骤二

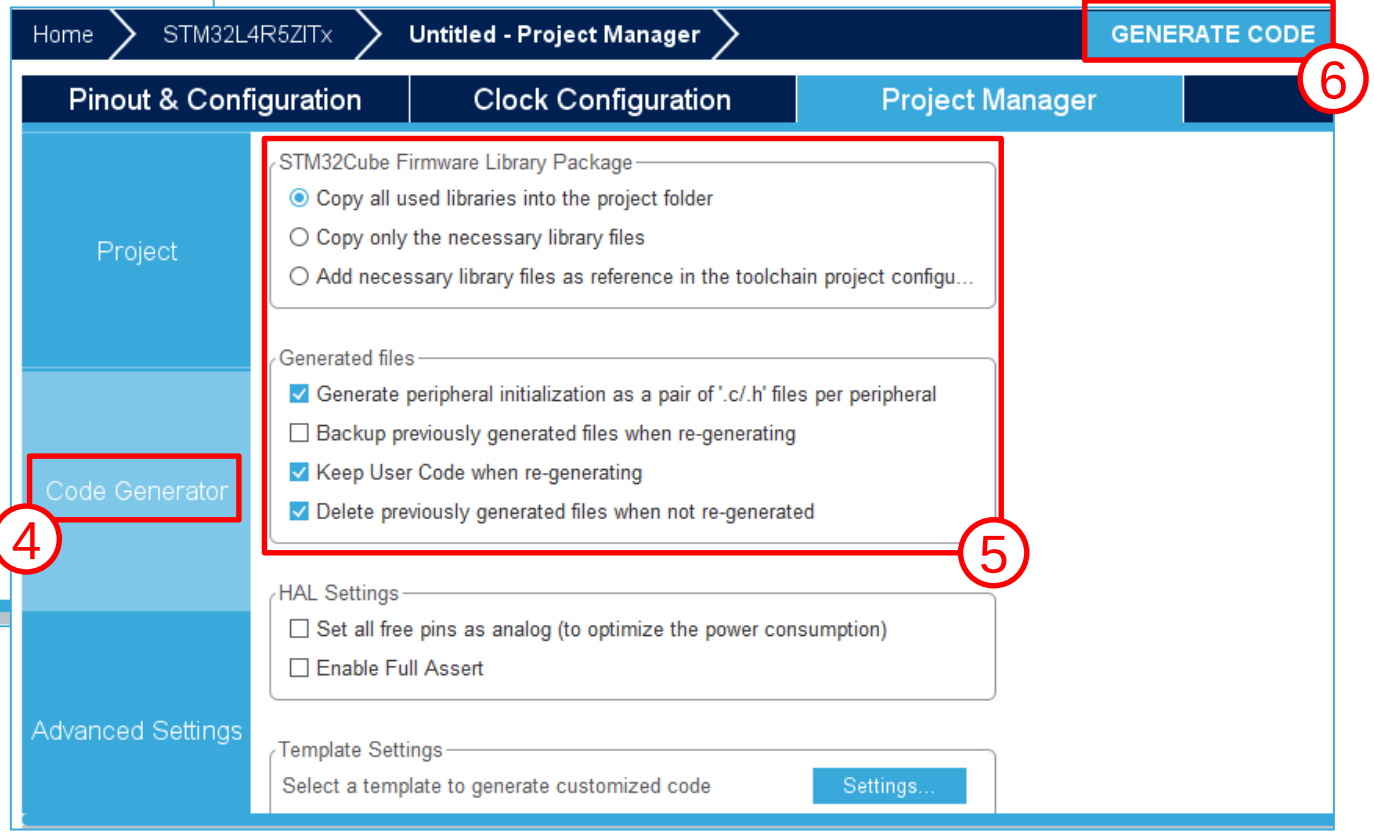
- 时钟配置
  - PLL源选择：MSI
  - 系统时钟：120MHz



# 使用CubeMX初始化系统：步骤三

- 生成IAR初始工程

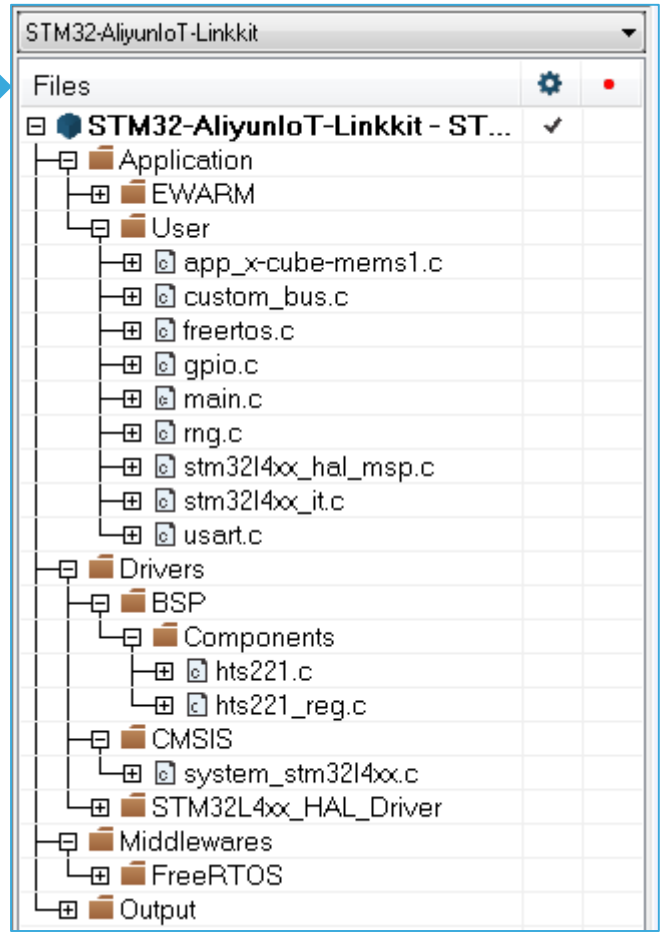


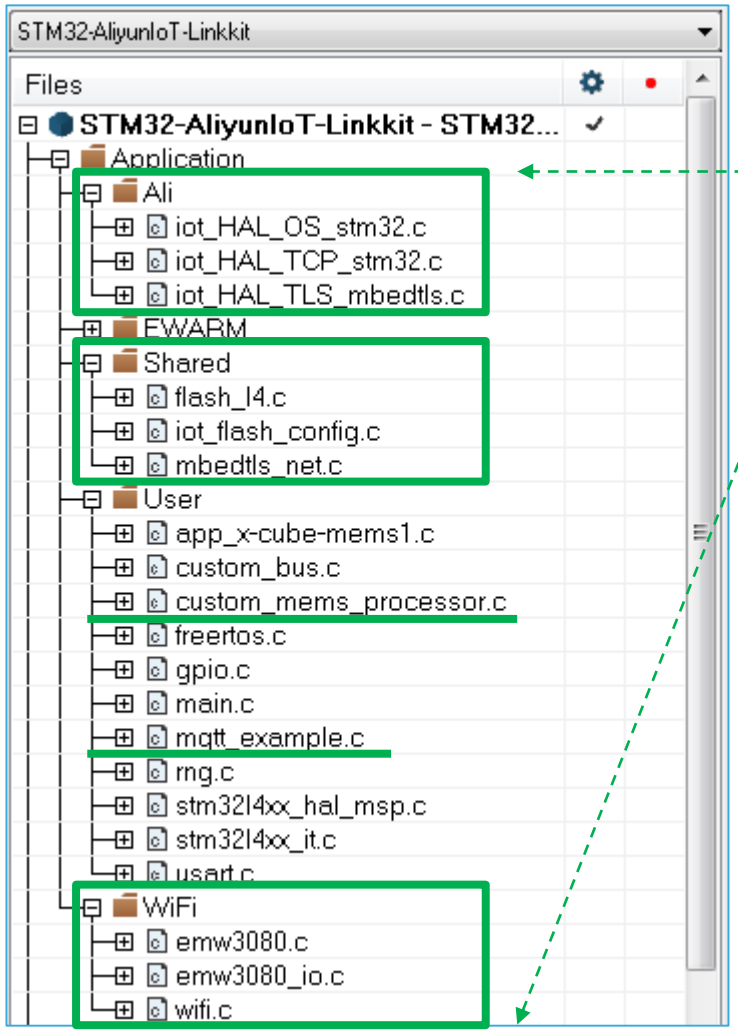
# 项目例程：初始 vs.交付

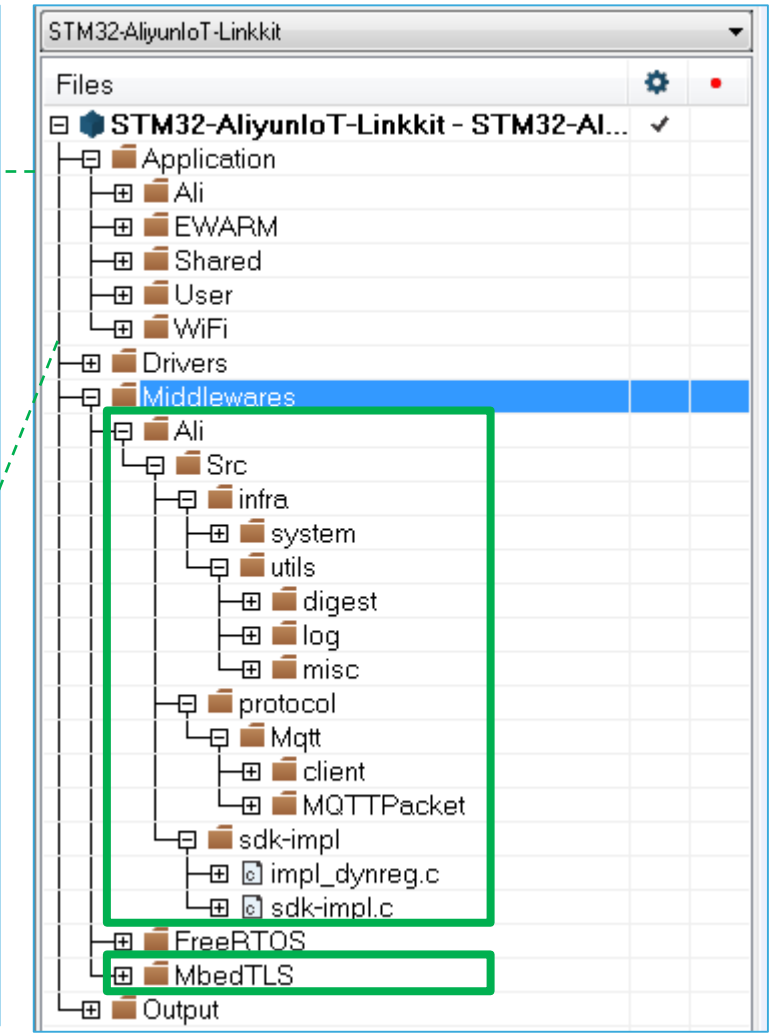


自动生成IAR初始工程文件




用户添加代码



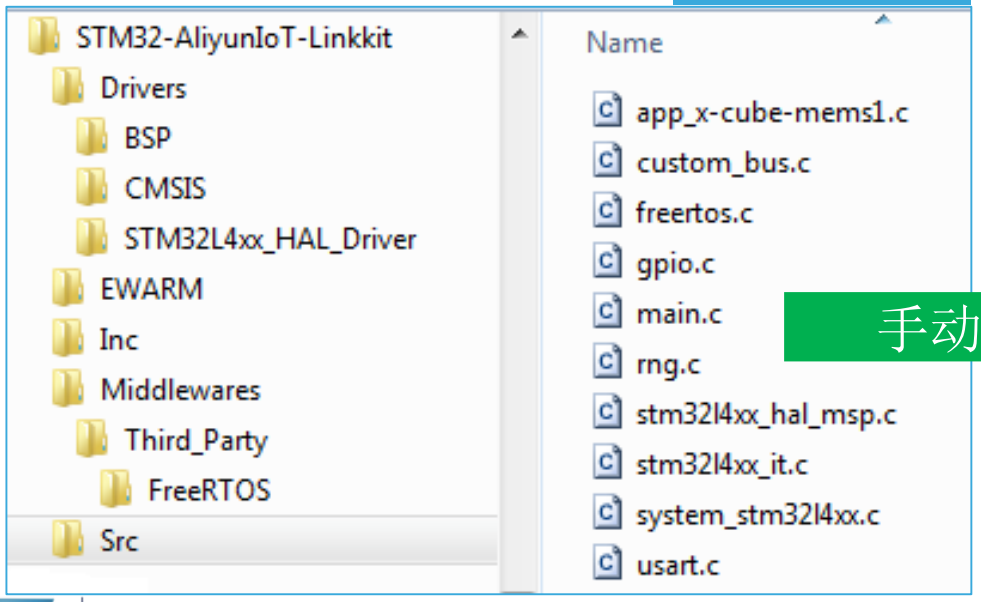



# 从初始工程项目 开始...



- 需要用户手动添加文件
  - 阿里云IoT平台设备端C-SDK
  - mbedTLS协议栈
  - Wifi模块驱动
  - 应用逻辑

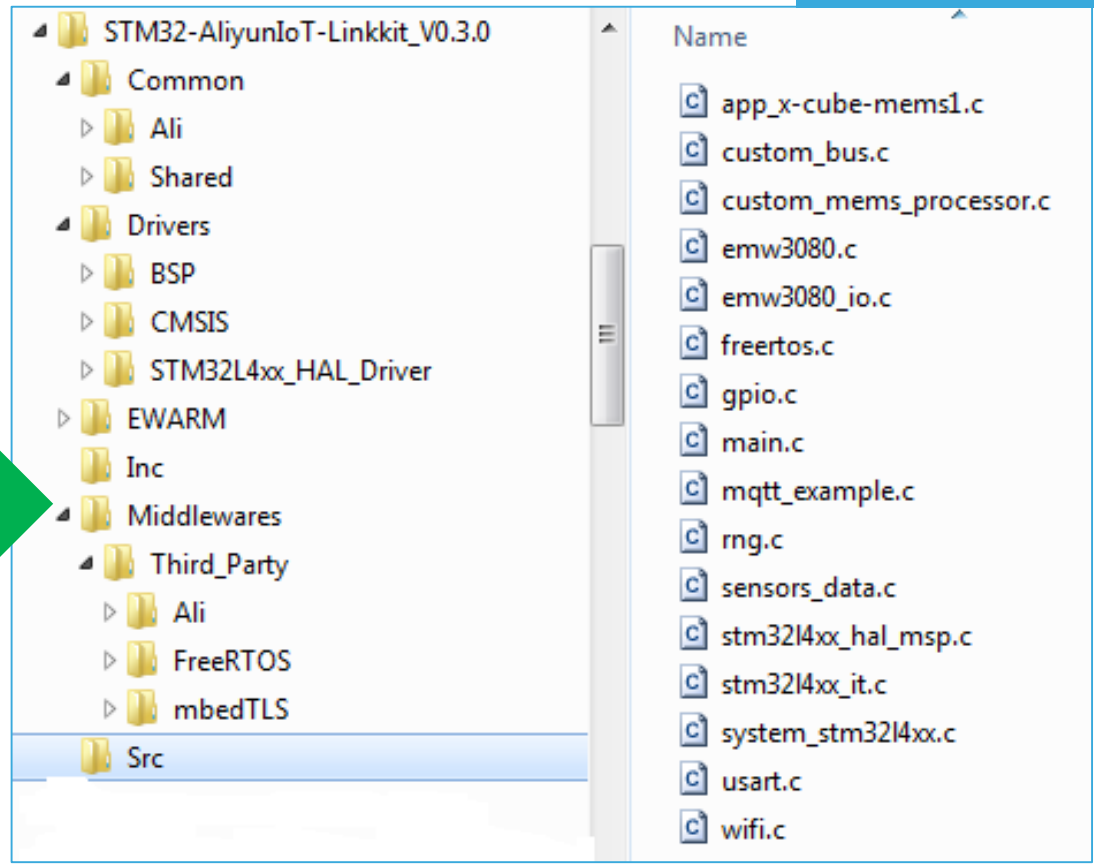
初始工程




手动添加

- 新增include路径
- 新增预编译宏定义

目标工程




life.augmented

阿里云

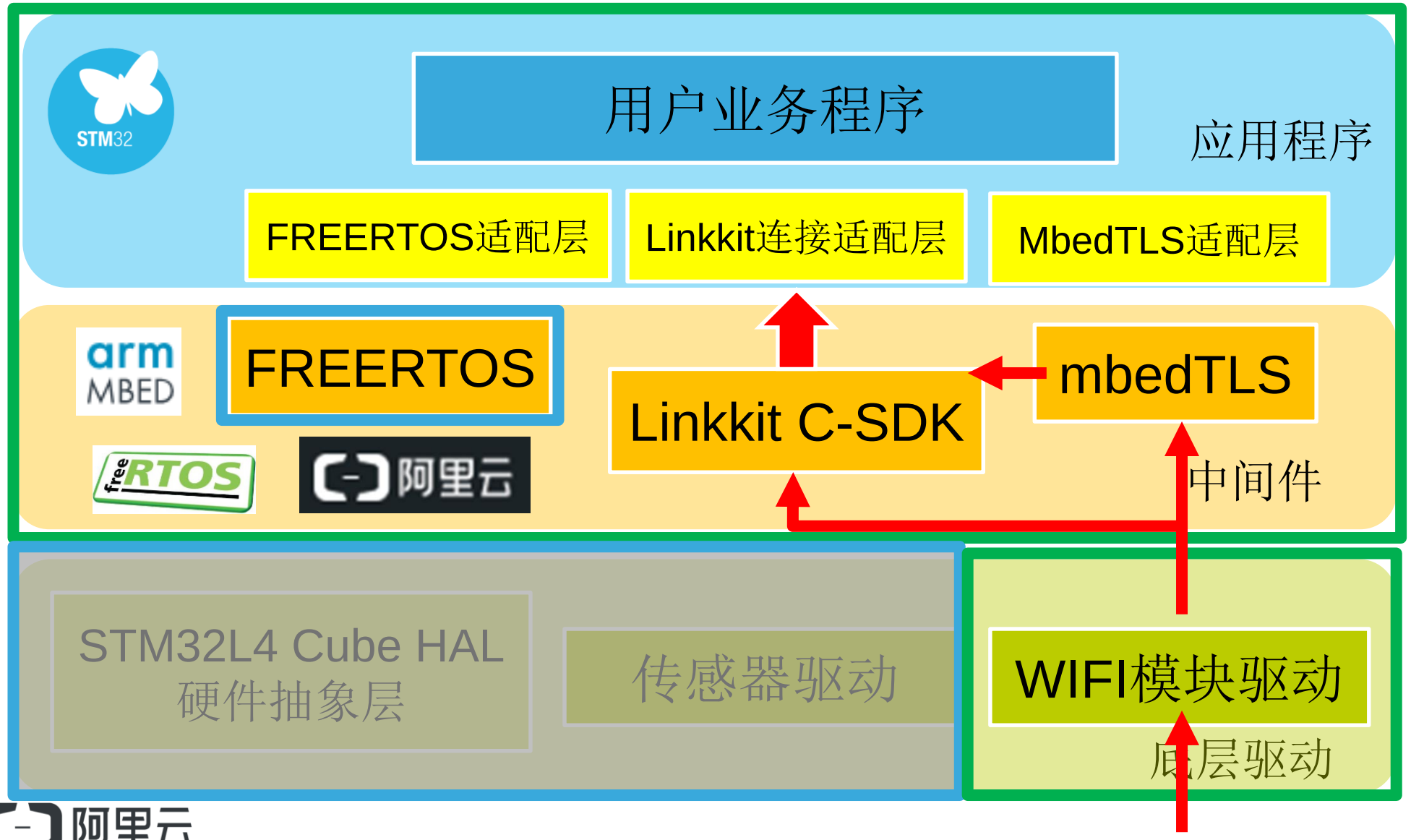
# 项目例程软件架构




用户业务程序
应用程序
FREERTOS适配层
Linkkit连接适配层
MbedTLS适配层
FREERTOS
Linkkit C-SDK
mbedTLS
中间件
需要由用户添加...
STM32L4 Cube HAL
硬件抽象层
传感器驱动
WIFI模块驱动
底层驱动
CubeMX自动生成

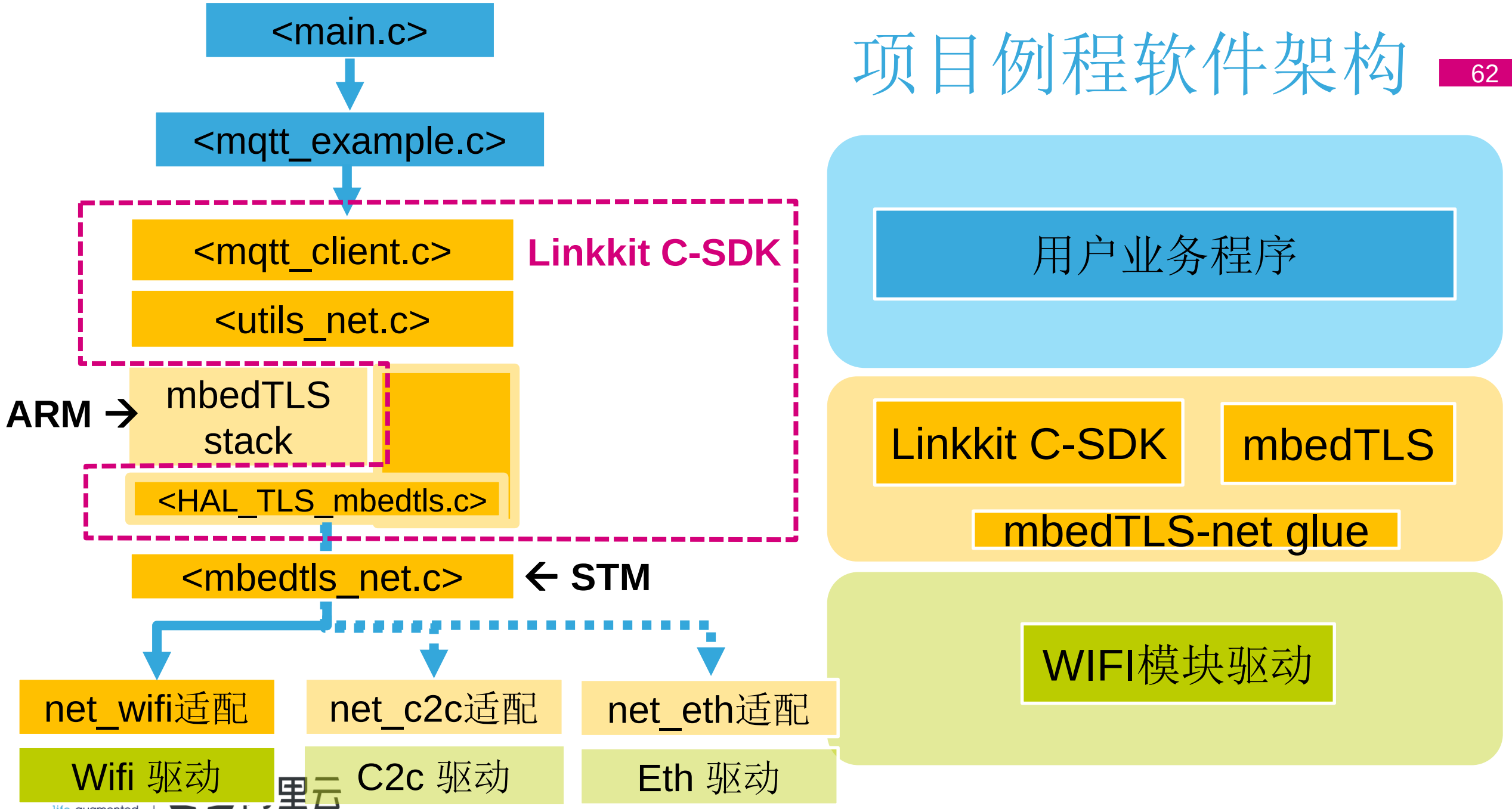
# 项目例程软件架构




用户业务程序
应用程序
FREERTOS适配层
Linkkit连接适配层
MbedTLS适配层
FREERTOS
Linkkit C-SDK
mbedTLS
中间件
STM32L4 Cube HAL
硬件抽象层
传感器驱动
WIFI模块驱动
底层驱动

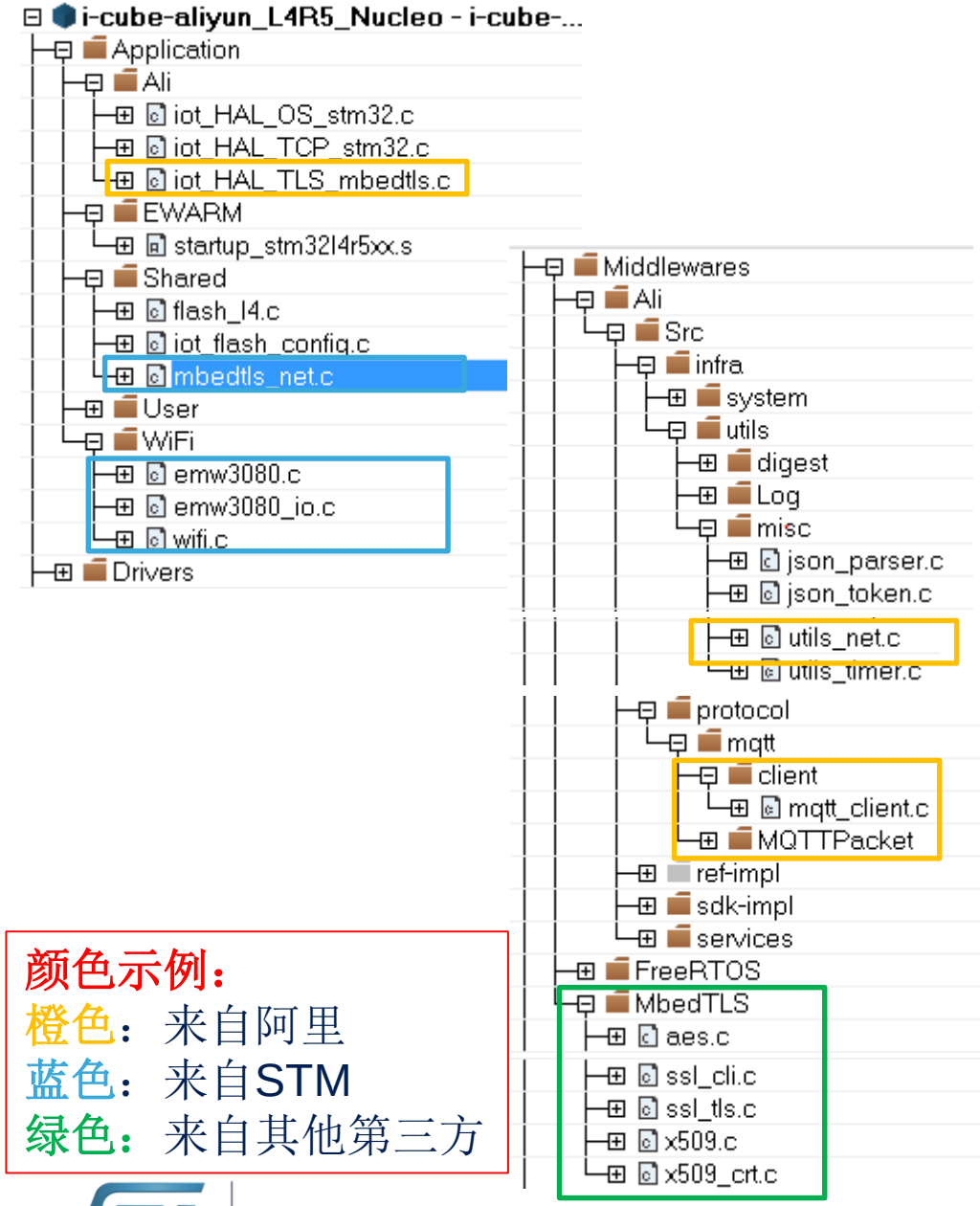
# 项目例程软件架构





# 网络数据流



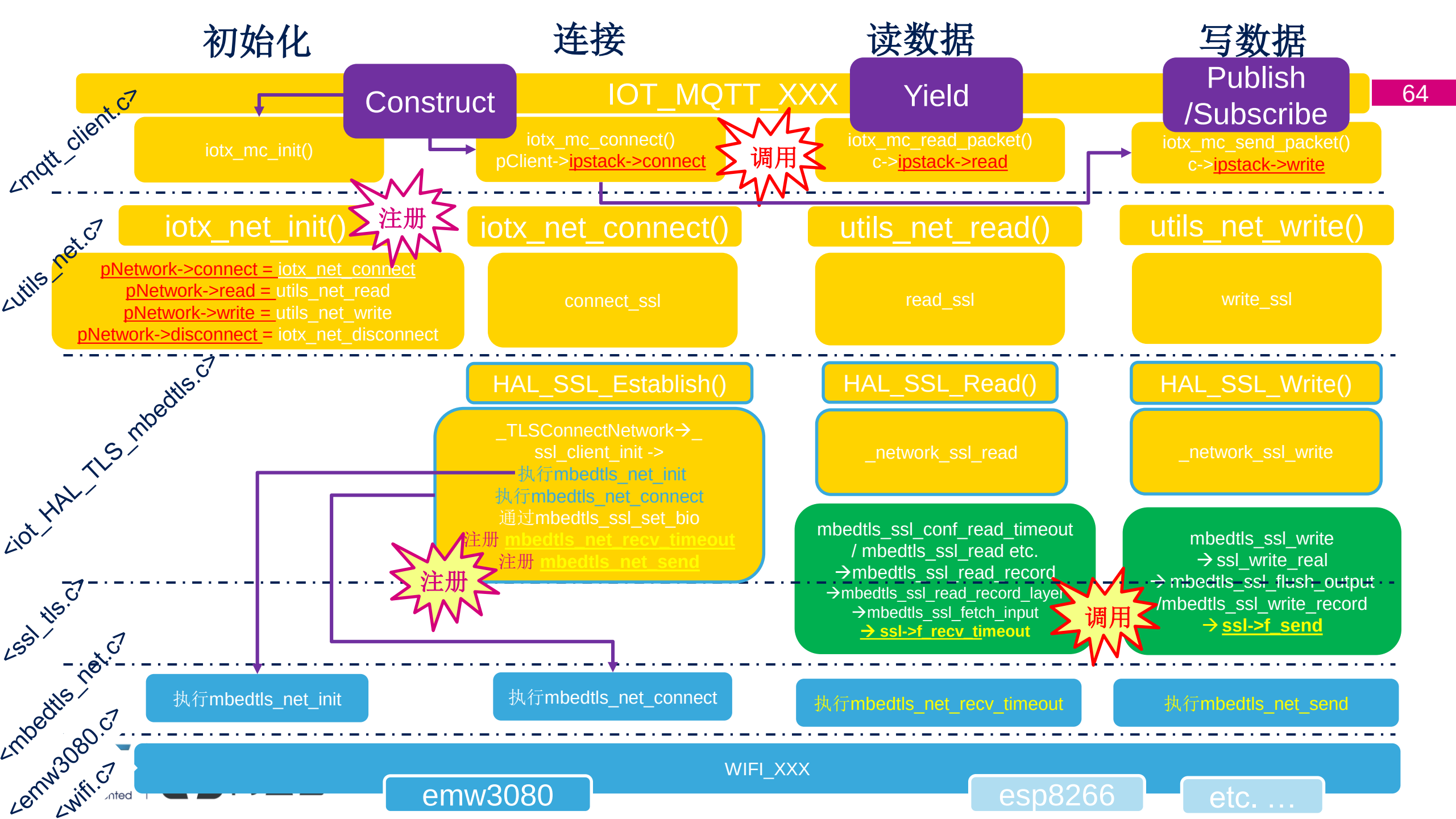



颜色示例：
橙色：来自阿里
蓝色：来自STM
绿色：来自其他第三方

| | |
|---|---|
| iot_HAL_TLS_mbedtls.c | linkkit的网络适配文件 |
| mbedtls_net.c | ST的mbedtls到net连接层 |
| *wifi_*.c | ST的wifi驱动模块 |
| utils_net.c | linkkit的网络管理 |
| mqtt_client.c | linkkit的mqtt客户端api封装 |
| *ssl_tls*.c | mbedtls协议栈 |

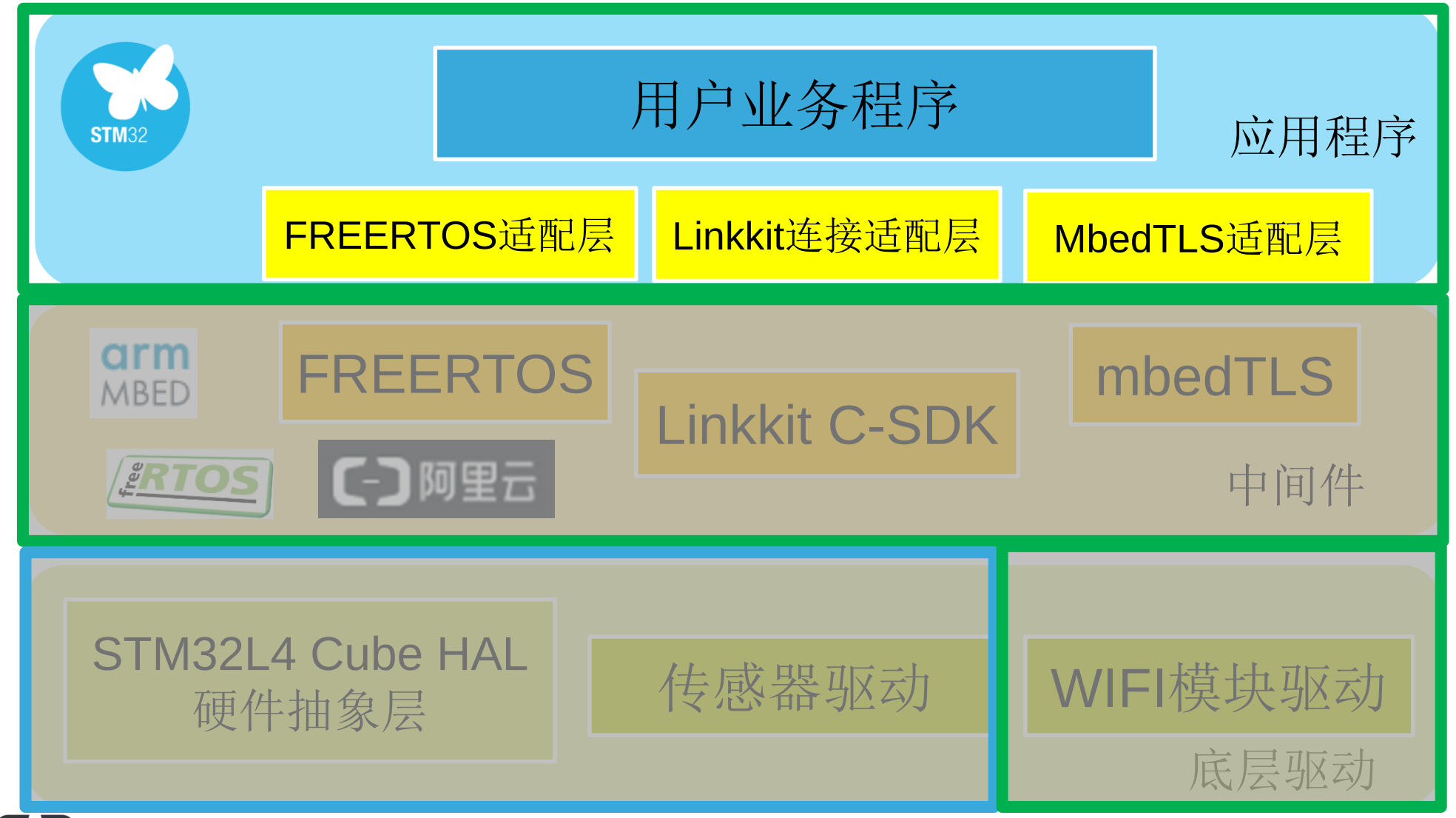
*按照代码在IAR项目里的自下往上排列顺序，非逻辑调用顺序

life.augmented 阿里云


初始化
连接
读数据
写数据
Construct
IOT_MQTT_XXX
Yield
Publish /Subscribe
<mqtt_client.c>
iotx_mc_init()
iotx_mc_connect()
pClient->ipstack->connect
调用
iotx_mc_read_packet()
c->ipstack->read
iotx_mc_send_packet()
c->ipstack->write
<utils_net.c>
iotx_net_init()
注册
iotx_net_connect()
utils_net_read()
utils_net_write()
pNetwork->connect = iotx_net_connect
pNetwork->read = utils_net_read
pNetwork->write = utils_net_write
pNetwork->disconnect = iotx_net_disconnect
connect_ssl
read_ssl
write_ssl
<iot_HAL_TLS_mbedtls.c>
HAL_SSL_Establish()
HAL_SSL_Read()
HAL_SSL_Write()
_TLSConnectNetwork→_
ssl_client_init ->
执行mbedtls_net_init
执行mbedtls_net_connect
通过mbedtls_ssl_set_bio
注册 mbedtls_net_recv_timeout
注册 mbedtls_net_send
_network_ssl_read
_network_ssl_write
注册
<ssl_tls.c>
mbedtls_ssl_conf_read_timeout
/ mbedtls_ssl_read etc.
→mbedtls_ssl_read_record
→mbedtls_ssl_read_record_layer
→mbedtls_ssl_fetch_input
→ ssl->f_recv_timeout
调用
mbedtls_ssl_write
→ ssl_write_real
→ mbedtls_ssl_flush_output
/mbedtls_ssl_write_record
→ ssl->f_send
<mbedtls_net.c>
执行mbedtls_net_init
执行mbedtls_net_connect
执行mbedtls_net_recv_timeout
执行mbedtls_net_send
<emw3080.c>
<wifi.c>
WIFI_XXX
emw3080
esp8266
etc. …

# 项目例程软件架构

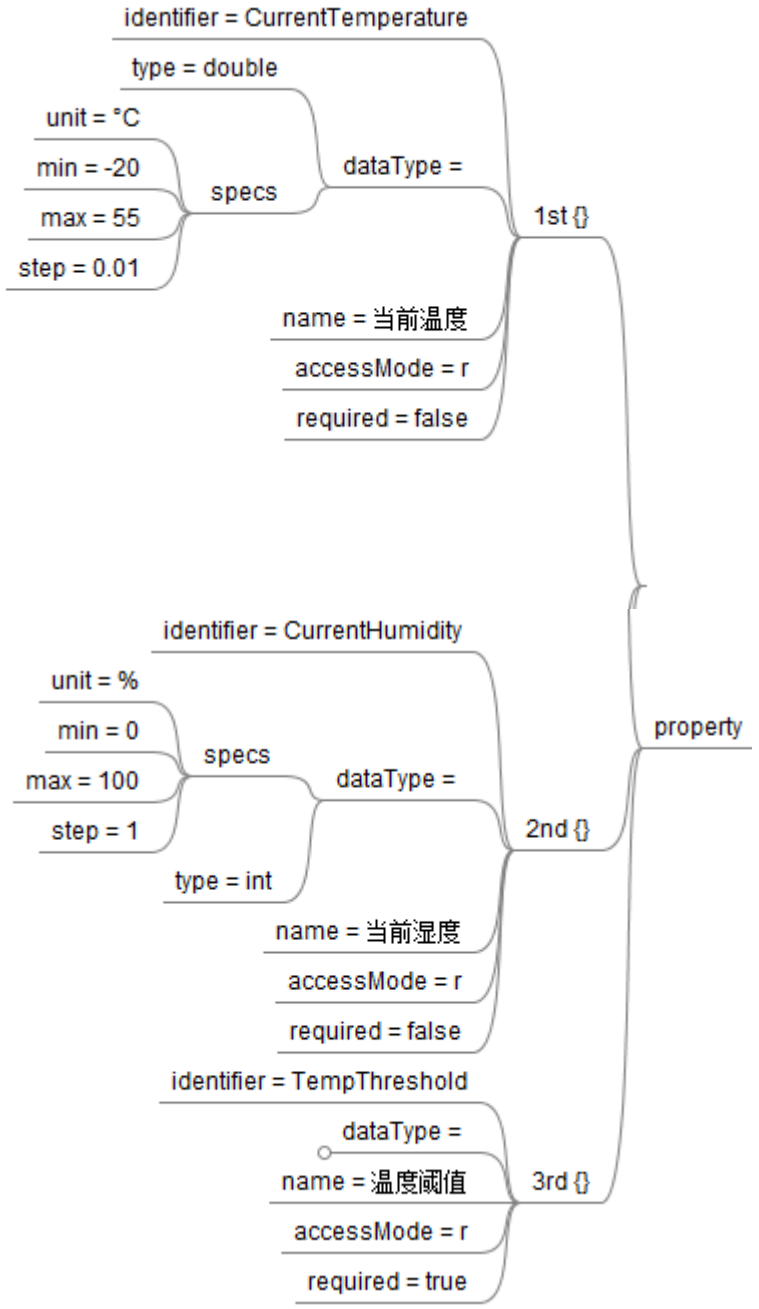



用户业务程序
应用程序
FREERTOS适配层
Linkkit连接适配层
MbedTLS适配层
FREERTOS
Linkkit C-SDK
mbedTLS
中间件
STM32L4 Cube HAL
硬件抽象层
传感器驱动
WIFI模块驱动
底层驱动

# 三个属性



- 当前温度
  - CurrentTemperature
- 当前湿度
  - CurrentHumidity
- 温度阈值
  - TempThreshold



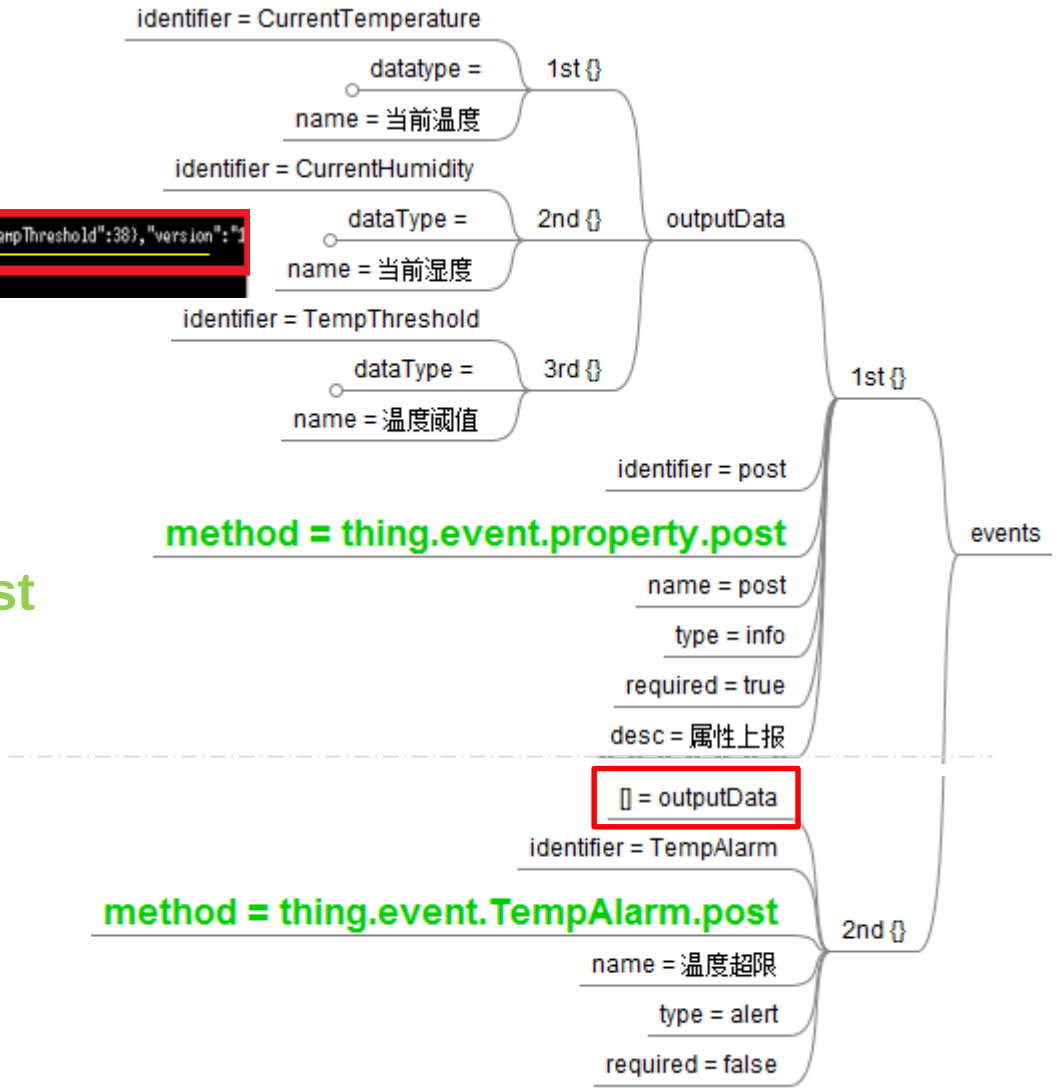
# 两个事件



- （三个）属性上报
  - **method = thing.event.property.post**

```
packet-id=8, publish topic msg={ "method":"thing.event.property.post", "id":"123" "params":{"CurrentHumidity":28,"CurrentTemperature":25.25,"TempThreshold":38},"version":"1
.0.0"}
********publish success, packet-id=8 ********
```

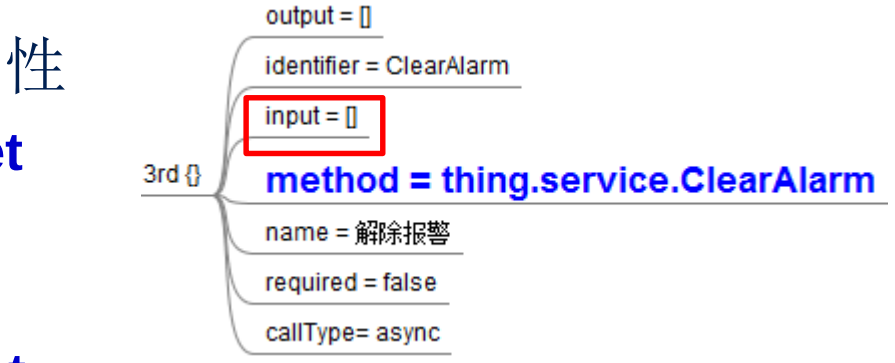
- 温度超限报警
  - **method = thing.event.TempAlarm.post**

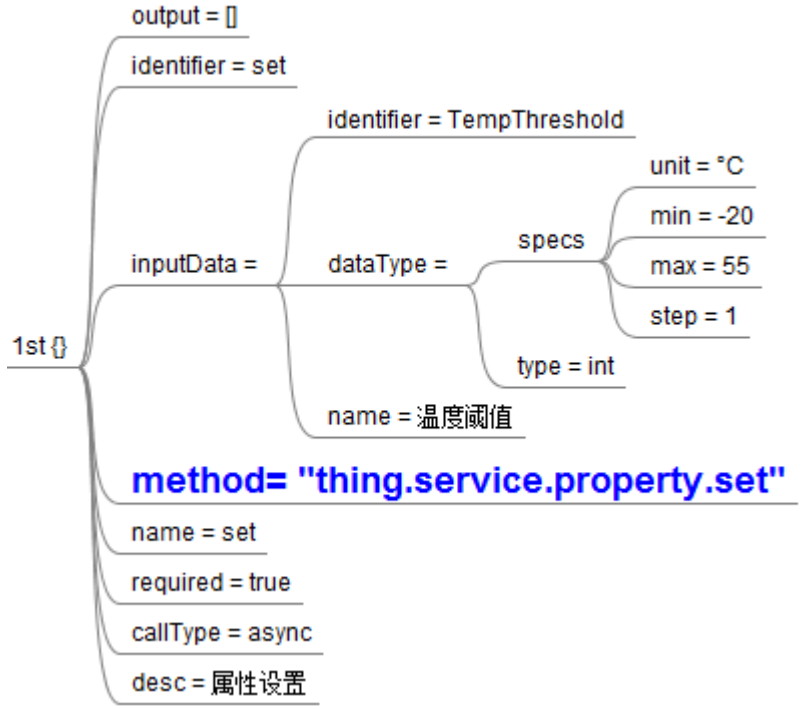



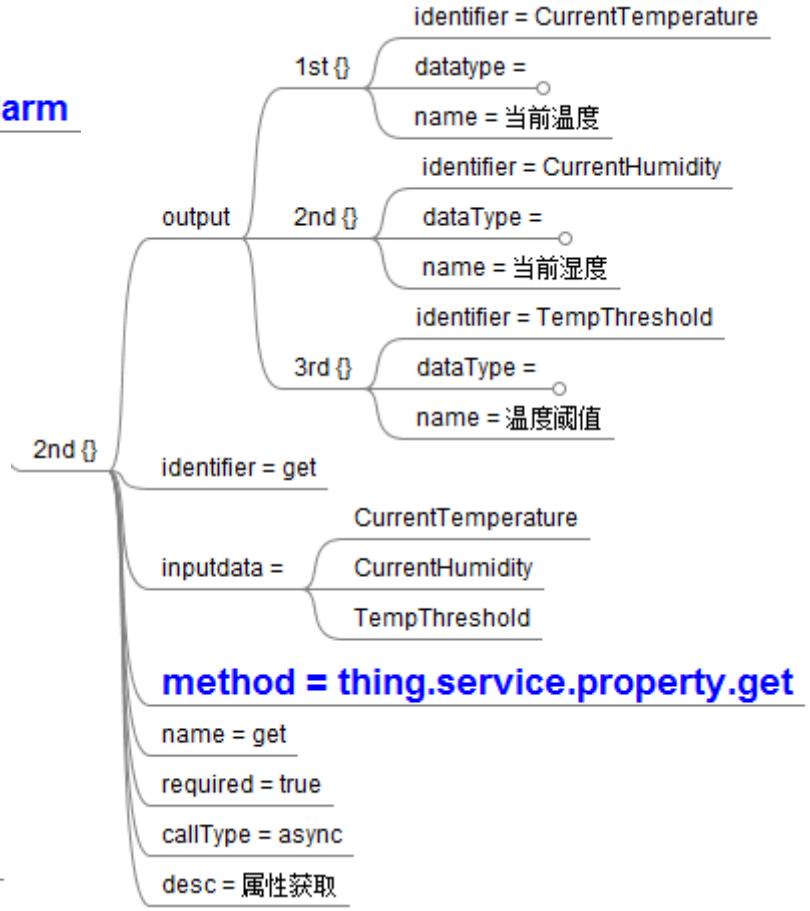
# 三个服务



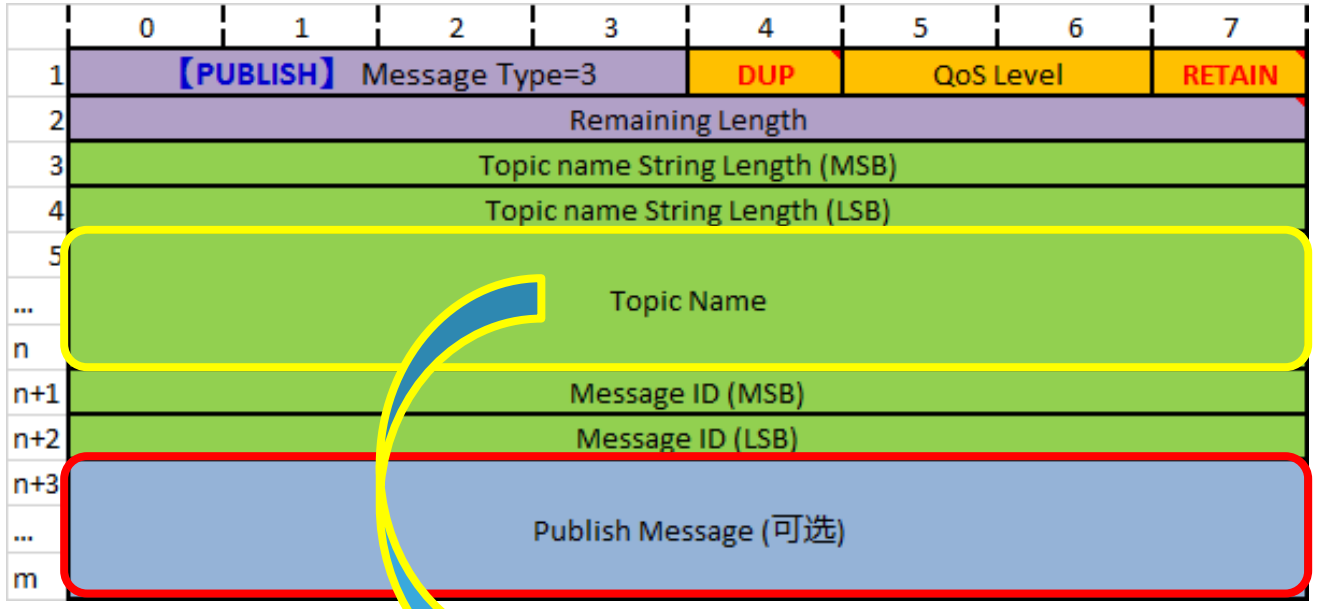
- 设置“温度阈值”这个属性
  - **thing.service.property.set**
- 获得（三个）属性
  - **thing.service.property.get**
- 解除警报
  - **thing.service.ClearAlarm**



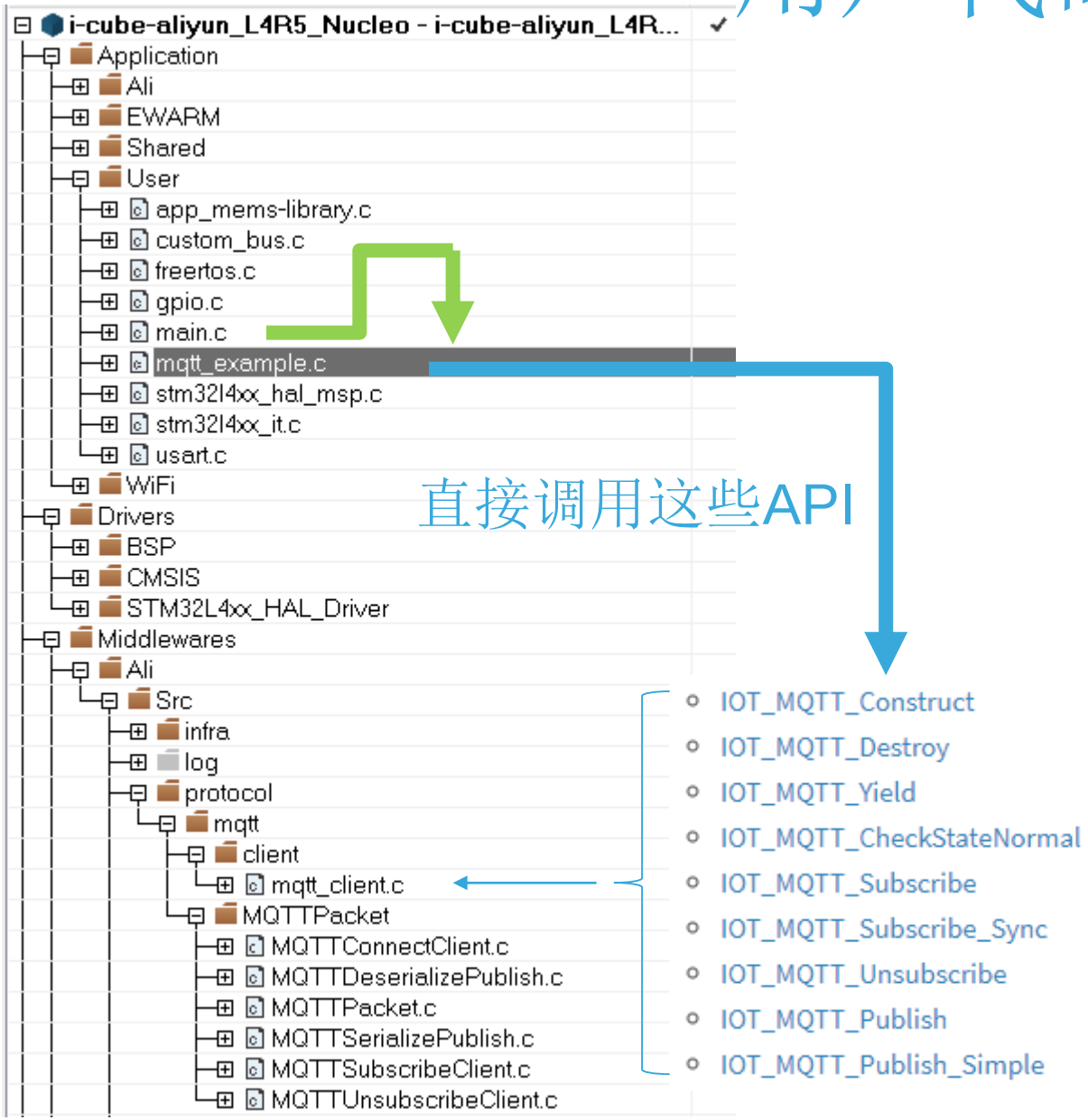
# 通信模型




| | MQTT主题 | 模型 (MCU) | 消息方向 | MQTT Message |
|---|---|---|---|---|
| 设置温度阈值 | /sys/${productKey}/${deviceName}/**thing.service.property.set** | 订阅 | 下行 | “id”:”1”,”version”:”1.0”,”method”:”左边字符串”,**”params”:{“TempThreshold”: XXX}** |
| 上报属性 | /sys/${productKey}/${deviceName}/thing.event.property.post | 发布 | 上行 | “id”:”2”,”version”:”1.0”,”method”:”左边字符串”,**”params”:{“CurrentTemperature”: YYY, “CurrentHumidity”: ZZZ}** |
| 温度报警 | /sys/${productKey}/${deviceName}/thing.event.TempAlarm.post | 发布 | 上行 | “id”:”3”,”version”:”1.0”,”method”:”左边字符串”,**”params”:{}** |
| 解除警报 | /sys/${productKey}/${deviceName}/**thing.service.ClearAlarm** | 订阅 | 下行 | |

# 用户代码如何调用MQTT API






AliIotKit_Thread

cloud_test

mqtt_client_example

main.c

mqtt_example.c

IOT_SetupConnInfo

IOT_MQTT_Construct

IOT_MQTT_Subscribe

IOT_MQTT_Publish

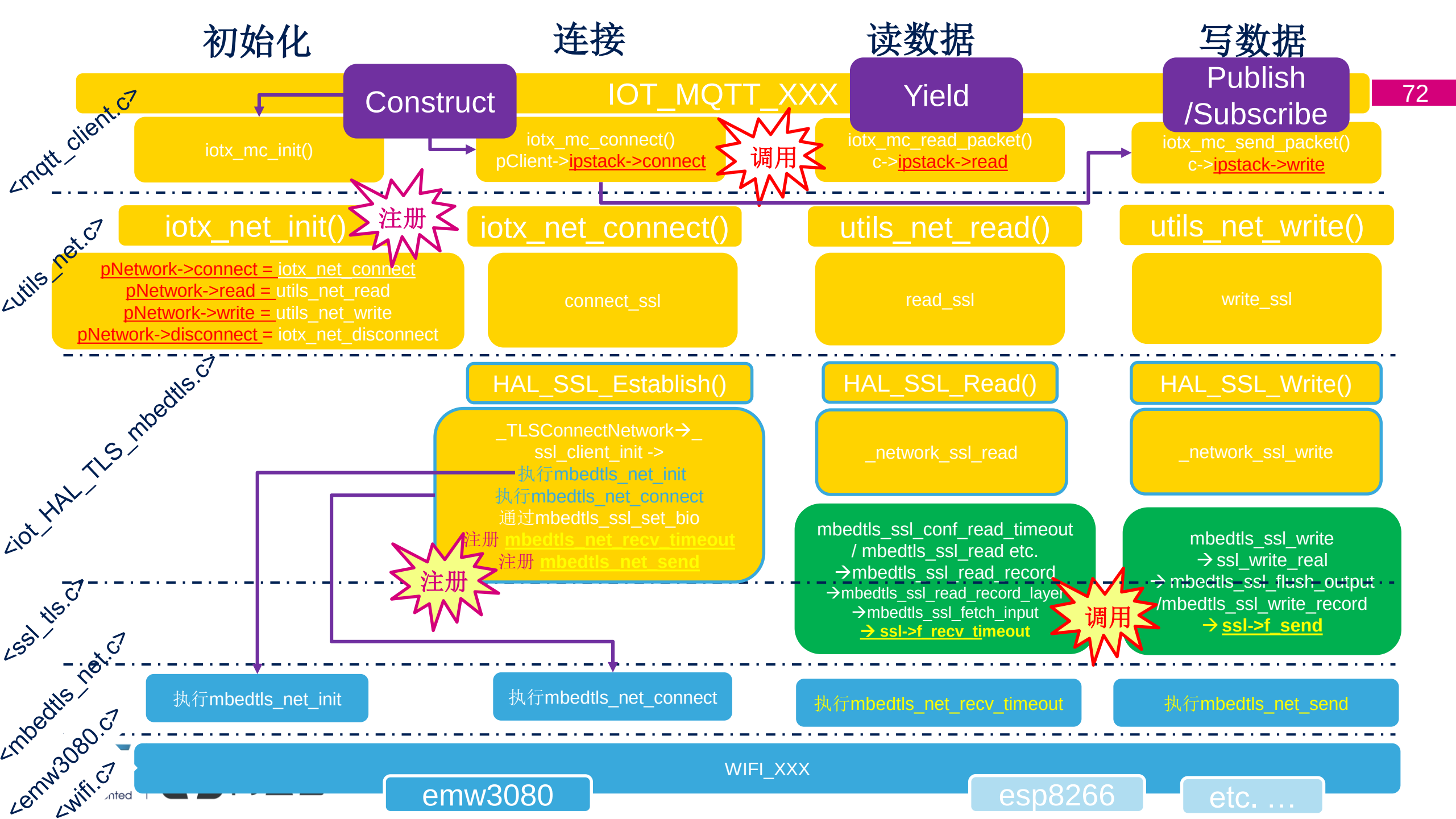
IOT_MQTT_Yield

mqtt_client.c

# 用户代码如何调用MQTT API



| | |
|---|---|
| getDeviceSecret(&tempsecret)<br>getProductKey(&product_key_temp)<br>getDeviceName(&device_name_temp) | 从flash中读出(之前用户输入的)设备三元组信息：产品key，设备名，设备密钥 |
| **IOT_SetupConnInfo** (product_key_temp, device_name_temp, tempsecret, (void **)&pconn_info) | |
| 根据设备三元组计算MQTT连接的几个参数：host_name, port, clientID, username, password, TLS连接时的服务器证书，并传递给mqtt_params | |
| **IOT_MQTT_Construct** (&mqtt_params) | |
| 1. iotx_mc_init (pclient, pInitParams) | 注册网络操作函数：连接、断开、读、写到connect_ssl, disconnect_ssl, read_ssl, write_ssl |
| 2. iotx_mc_connect (pclient) | 执行网络连接，<br>注册ssl读、写函数到mbedtls_net_send和mbedtsl_net_recv_timeout |
| | 执行网络写操作，发送MQTT connect报文 |
| 3. iotx_report_devinfo | 执行网络写操作，发送MQTT publish报文 |


初始化
连接
读数据
写数据
IOT_MQTT_XXX
Construct
Yield
Publish /Subscribe
<mqtt_client.c>
iotx_mc_init()
iotx_mc_connect()
pClient->ipstack->connect
调用
iotx_mc_read_packet()
c->ipstack->read
iotx_mc_send_packet()
c->ipstack->write
注册
<utils_net.c>
iotx_net_init()
iotx_net_connect()
utils_net_read()
utils_net_write()
pNetwork->connect = iotx_net_connect
pNetwork->read = utils_net_read
pNetwork->write = utils_net_write
pNetwork->disconnect = iotx_net_disconnect
connect_ssl
read_ssl
write_ssl
<iot_HAL_TLS_mbedtls.c>
HAL_SSL_Establish()
HAL_SSL_Read()
HAL_SSL_Write()
_TLSConnectNetwork→_
ssl_client_init ->
执行mbedtls_net_init
执行mbedtls_net_connect
通过mbedtls_ssl_set_bio
注册 mbedtls_net_recv_timeout
注册 mbedtls_net_send
_network_ssl_read
_network_ssl_write
注册
<ssl_tls.c>
mbedtls_ssl_conf_read_timeout
/ mbedtls_ssl_read etc.
→mbedtls_ssl_read_record
→mbedtls_ssl_read_record_layer
→mbedtls_ssl_fetch_input
→ ssl->f_recv_timeout
调用
mbedtls_ssl_write
→ ssl_write_real
→ mbedtls_ssl_flush_output
/mbedtls_ssl_write_record
→ ssl->f_send
<mbedtls_net.c>
执行mbedtls_net_init
执行mbedtls_net_connect
执行mbedtls_net_recv_timeout
执行mbedtls_net_send
<emw3080.c>
<wifi.c>
WIFI_XXX
emw3080
esp8266
etc. …

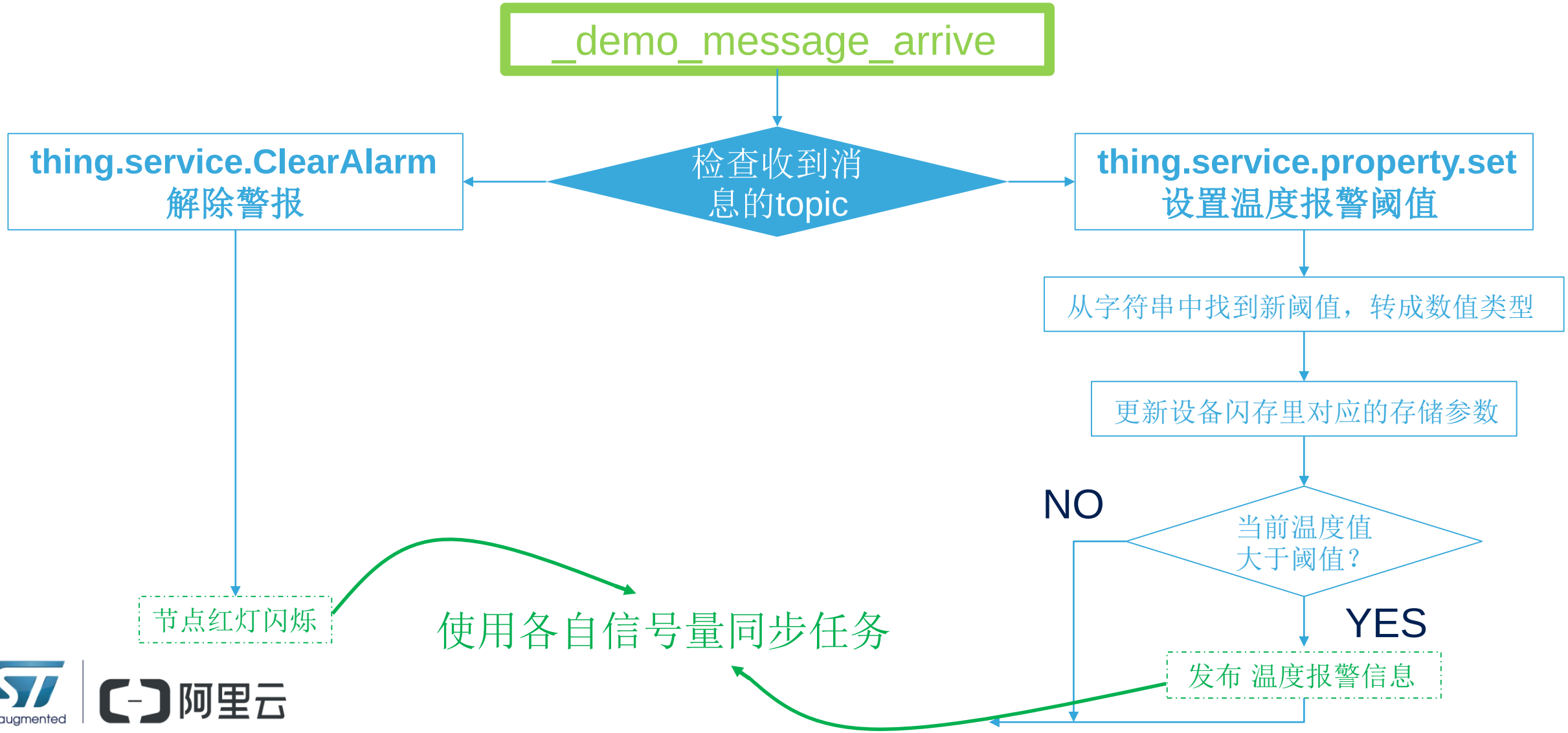
# 用户代码如何调用MQTT API



| | |
|---|---|
| Temp_Topic主题赋值： | /sys/%s/%s/thing/service/property/set |
| **IOT_MQTT_Subscribe** (pclient, Temp_Topic, IOTX_MQTT_QOS1, _demo_message_arrive, NULL) | |
| Temp_Topic主题赋值： | /sys/%s/%s/thing/service/ClearAlarm |
| **IOT_MQTT_Subscribe** (pclient, Temp_Topic, IOTX_MQTT_QOS1, _demo_message_arrive, NULL) | |
| Temp_Topic主题赋值： | /sys/%s/%s/thing/event/TempAlarm/post |
| **IOT_MQTT_Publish** (pclient, Temp_Topic, &topic_msg) | |
| Temp_Topic主题赋值： | /sys/%s/%s/thing/event/property/post |
| **IOT_MQTT_Publish** (pclient, Temp_Topic, &topic_msg) | |
| **IOT_MQTT_Yield** (pclient, 1000) | |
| 1. iotx_mc_keepalive | 发送ping包，或者重连 |
| 2. iotx_mc_cycle | 执行网络读操作(iotx_mc_read_packet)，解析收到的包<br>subscribe : handler->handle.h_fp = messageHandler(即_demo_message_arrive)<br>PUBLISH : 执行注册好的回调函数 handle->h_fp(handle->pcontext, c, msg) |

# MQTT订阅消息的回调函数





# 用户输入运行参数



- 需要保存在节点设备闪存中的信息
  - WIFI配网参数（用户串口输入）
  - 设备三元组信息（用户串口输入）
  - 温度报警阈值（云端下发）

```
typedef struct {
  uint64_t magic;                          /**< The USER_CONF_MAGIC magic word signals that the structu
  char ssid[USER_CONF_WIFI_SSID_MAX_LENGTH];  /**< Wifi network SSID. */
  char psk[USER_CONF_WIFI_PSK_MAX_LENGTH];    /**< Wifi network PSK. */
  uint8_t security_mode;                   /**< Wifi network security mode. See @ref wifi_network_secur
} wifi_config_t;

typedef struct {
  uint64_t magic;                          /**< The USER_CONF_MAGIC magic word signals that the structu
  char product_key[USER_CONF_PRODUCT_KEY_LENGTH];
  char device_name[USER_CONF_DEVICE_NAME_LENGTH];
  char device_secret[USER_CONF_DEVICE_SECRET_LENGTH];
  char region_id[USER_CONF_REGION_ID_LENGTH];
} iot_config_t;

typedef struct {
  uint64_t magic;                          /**< The USER_CONF_MAGIC magic word signals that the structu
  uint8_t temprature_threshold;
}app_config_t;
/** Static user configuration data which must survive reboot and firmware update. */
typedef struct {
  wifi_config_t wifi_config;
  iot_config_t iot_config;
  app_config_t app_config;
} user_config_t;
```

iot_flash_config.h

```
*** start WiFi provisioning ***
Push the User button (Blue) within the next 5 seconds if you want to update the WiFi network configuration.
Your WiFi parameters need to be entered to proceed.
Enter SSID: [redacted]
You have entered [redacted] as the ssid.
Enter Security Mode (0 - Open, 1 - WEP, 2 - WPA, 3 - WPA2):3
You have entered 3 as the security mode.
Enter password: [redacted]
*** start WiFi initialization ***
> WiFi module initialized.
firmware version is : basic_AT_v2.1.2
OK
*** try to get WiFi MAC address ***
> WiFi module MAC address is: B0:F8:93:17:BB:2C
```

若要重新配置wifi热点，需要5秒内按下user键

输入热点名称和密码

```
Push the User button (Blue) within the next 5 seconds if you want to update the device security parameters or credentials.
Enter Product Key: (example: a1b05Uexxxx)
a1zPqJ3NYxJ
read: --->
a1zPqJ3NYxJ
<---
Enter device name: (example: mydevicename)
2019CT_case2_node001
read: --->
2019CT_case2_node001
<---
Enter device secret: (example: 7o7GJ3odUE7pPnieO7dzxxxxxxxxxxx)
1VDt2s7hUyZVL62oS9VXgT6nnjjRoOCi
read: --->
[redacted]
<---
```

输入product key

输入device name

2019CT_case2_node001

输入device secret

life.augmented

阿里云

# Demo参数存储



- MCU用户闪存容量2M，页大小4K（双bank）
- 取末尾32K作为用户参数存储区


- link文件中定义该区域

```
define symbol __ICFEDIT_region_FIXED_LOC_start__ = 0x081F8000;
define region uninit_fixed_loc = mem:[from __ICFEDIT_region_FIXED_LOC_start__ size 32K];
```

- iot_flash_config.c将全局变量指定在此区域

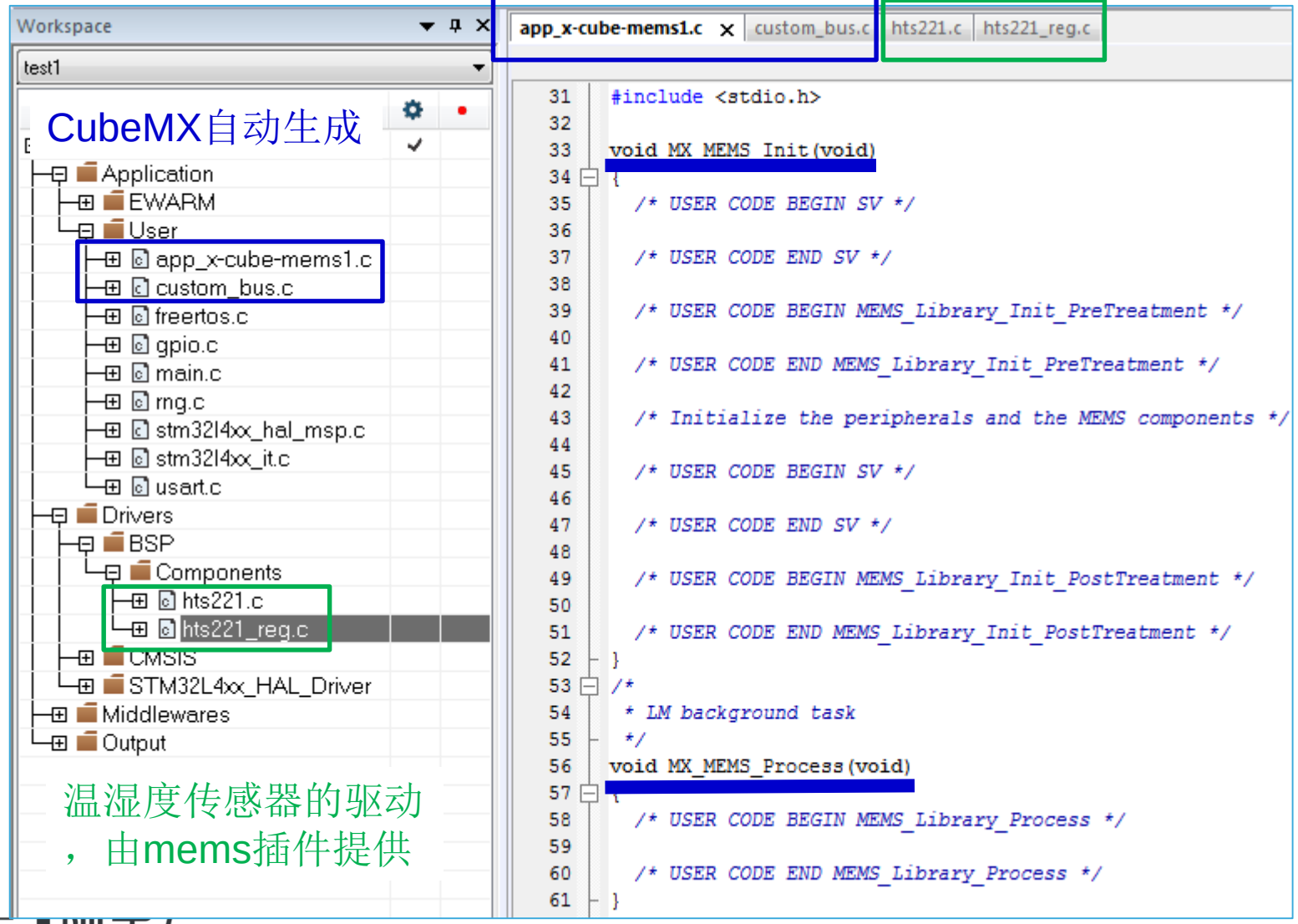
```
#ifdef __ICCARM__  /* IAR */
__no_init const user_config_t lUserConfig @ "UNINIT_FIXED_LOC";
#elif defined ( __CC_ARM   )/* Keil / armcc */

#elif defined ( __GNUC__ )      /*GNU Compiler */

#endif
```

# Sensor数据的读取

自动生成IAR初始工程及相关源文件…



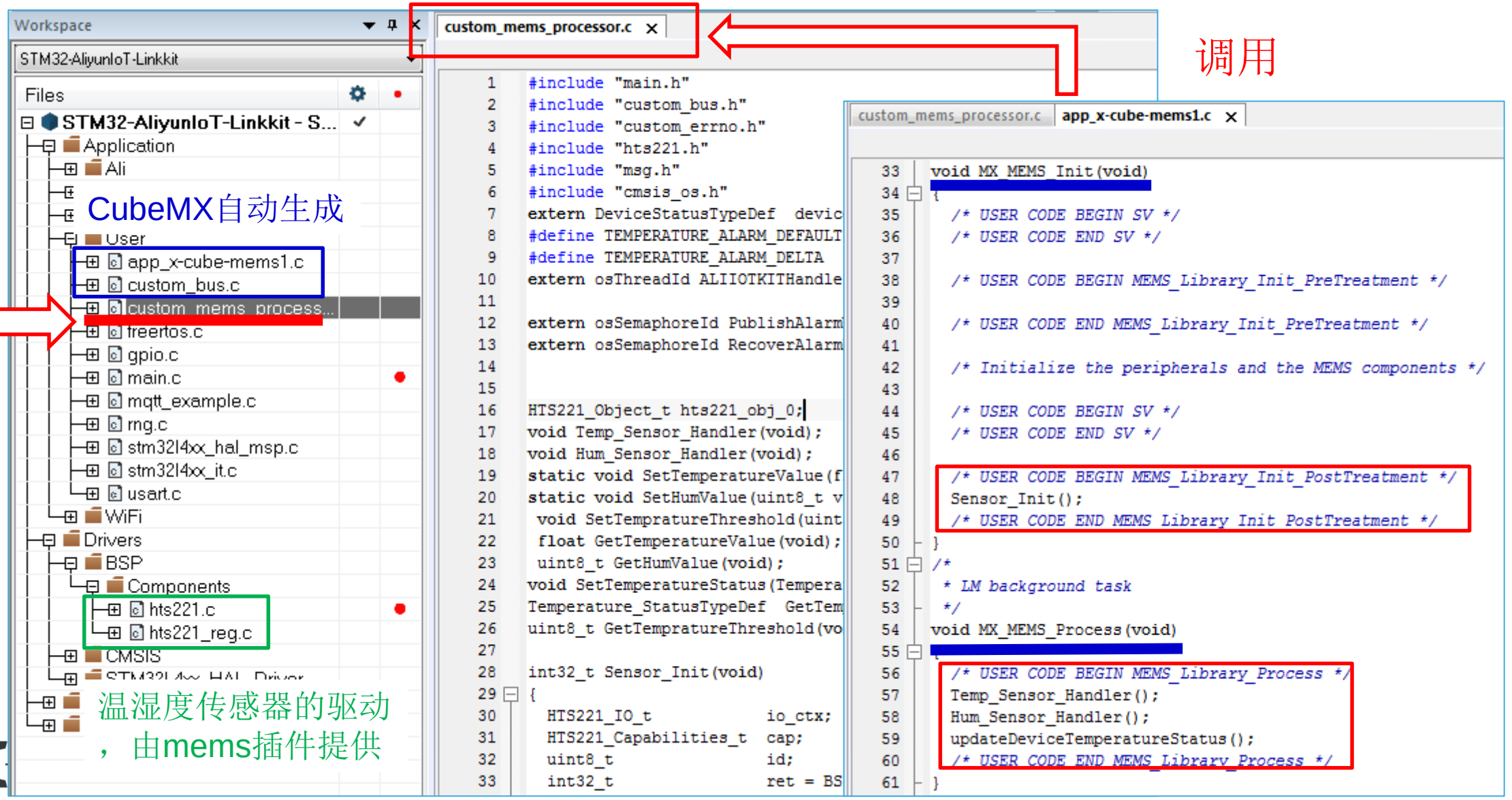

## Custom_bus.c

- BSP_I2C1_Init
- BSP_I2C1_DeInit
- BSP_I2C1_IsReady
- BSP_I2C1_WriteReg
- BSP_I2C1_ReadReg
- BSP_I2C1_WriteReg16
- BSP_I2C1_ReadReg16

交付的IAR例程项目

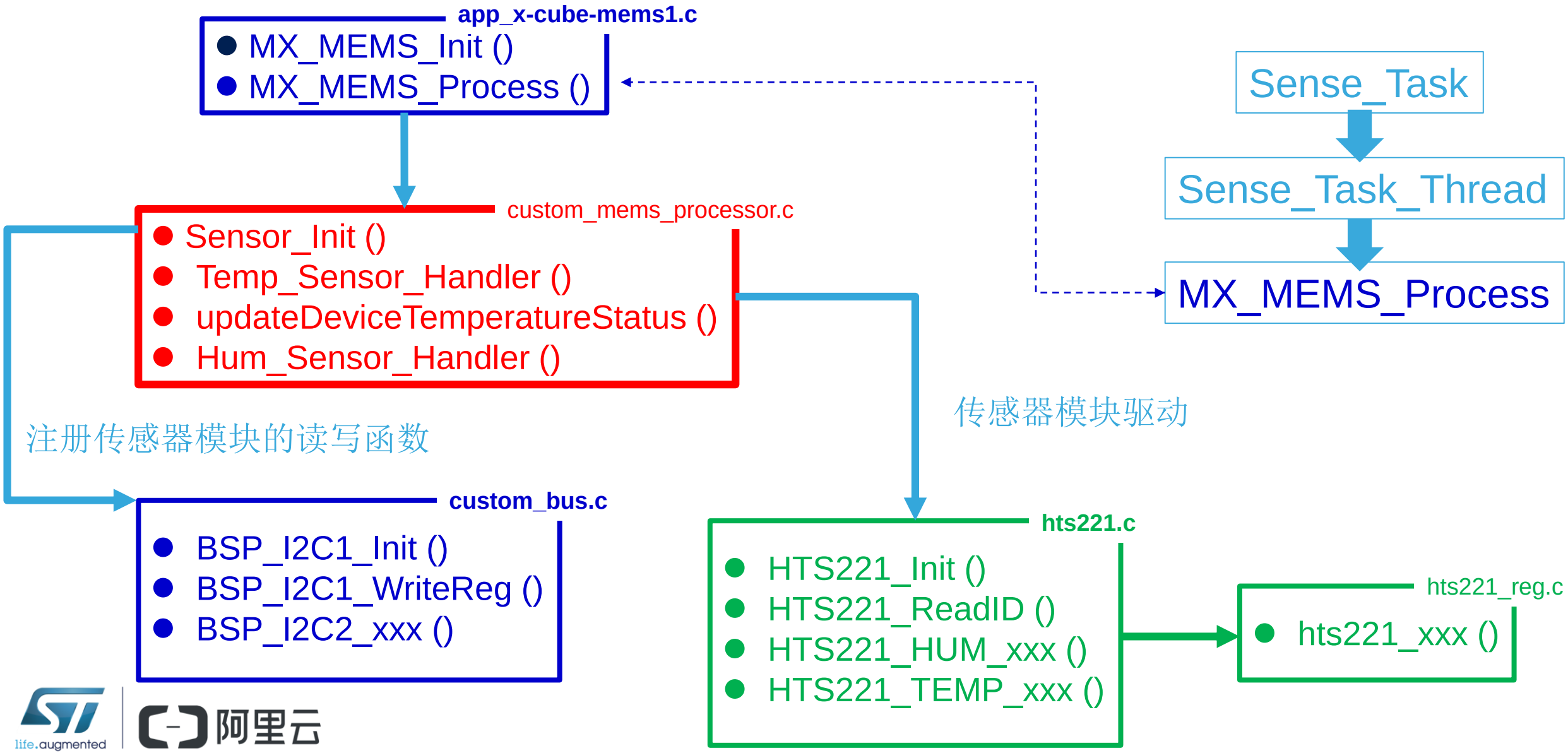
# Sensor数据的读取





# Sensor数据的读取



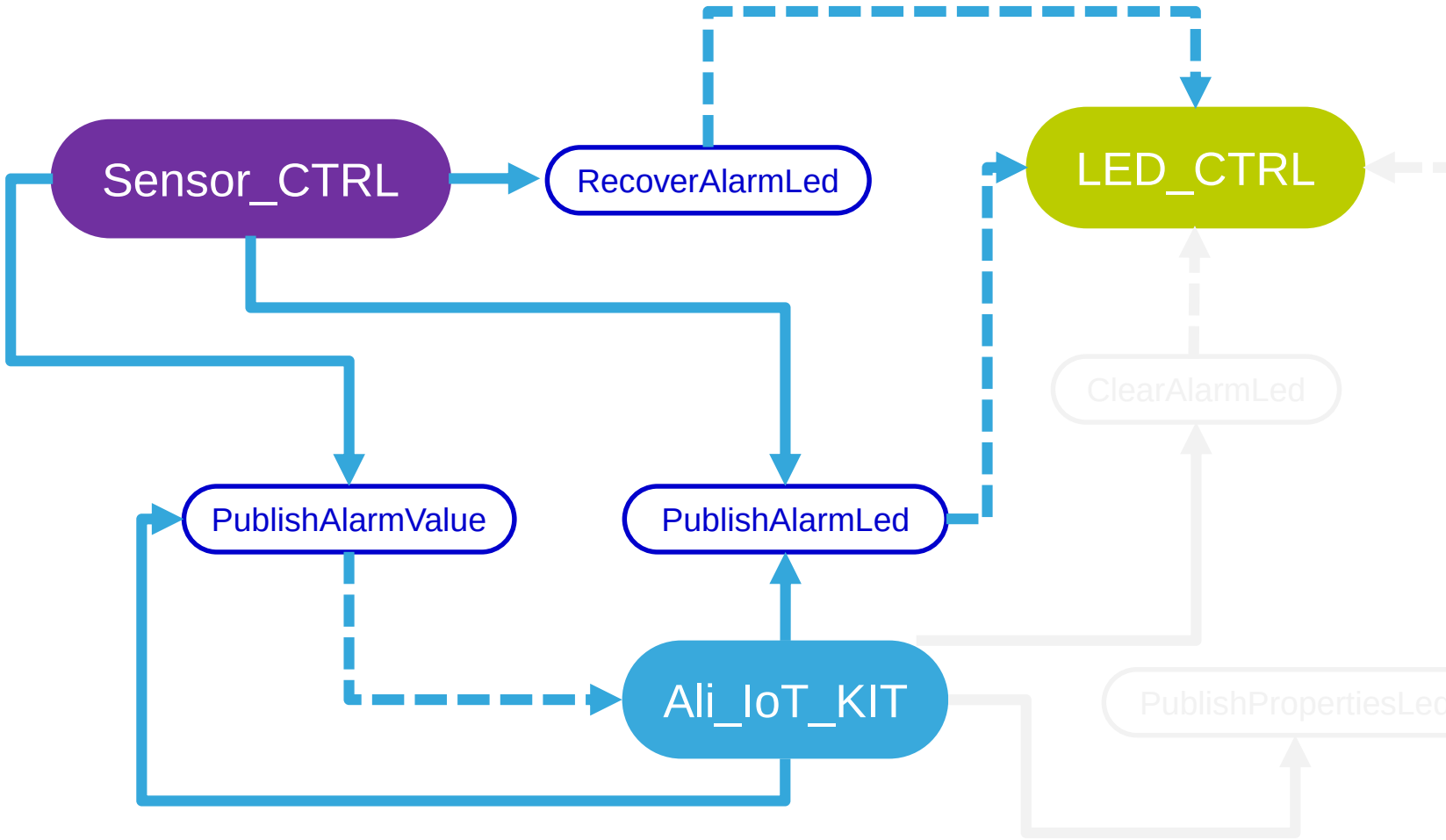


# Sensor_CTRL任务



- 3个应用任务
  - Ali_IoT_KIT
  - LED_CTRL
  - Sensor_CTRL
- 5个同步信号量
  - PublishPropertiesLed
  - PublishAlarmValue
  - PublishAlarmLed
  - RecoverAlarmLed
  - ClearAlarmLed

- 任务间的同步



main.c

main()

```
101  int main(void)
102  {
103    /* USER CODE BEGIN 1 */
104
105    /* USER CODE END 1 */
106
107    /* MCU Configuration--------------------------
108
109    /* Reset of all peripherals, Initializes the
110    HAL_Init();
111
112    /* USER CODE BEGIN Init */
113
114    /* USER CODE END Init */
115
116    /* Configure the system clock */
117    SystemClock_Config();
118
119    /* USER CODE BEGIN SysInit */
120
121    /* USER CODE END SysInit */
122
123    /* Initialize all configured peripherals */
124    MX_GPIO_Init();
125    MX_LPUART1_UART_Init();
126    MX_USART3_UART_Init();
127    MX_RNG_Init();
128    /* USER CODE BEGIN 2 */
129
130    /* USER CODE END 2 */
131
132    /* Call init function for freertos objects
133    MX_FREERTOS_Init();
134
135    /* Start scheduler */
136    osKernelStart();
137
138    /* We should never get here as control is no
139
140    /* Infinite loop */
141    /* USER CODE BEGIN WHILE */
142    while (1)
143    {
144      /* USER CODE END WHILE */
145
146      /* USER CODE BEGIN 3 */
147    }
148    /* USER CODE END 3 */
149  }
```

main.c

main()

```
240  main(void)
241  {
242    /* USER CODE BEGIN 1 */
243
244    /* USER CODE END 1 */
245
246    /* MCU Configuration--------------------------------------------------------*/
247
248    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
249    HAL_Init();
250
251    /* USER CODE BEGIN Init */
252
253    /* USER CODE END Init */
254
255    /* Configure the system clock */
256    SystemClock_Config();
257
258    /* USER CODE BEGIN SysInit */
259
260    /* USER CODE END SysInit */
261
262    /* Initialize all configured peripherals */
263    MX_GPIO_Init();
264    MX_LPUART1_UART_Init();
265    MX_USART3_UART_Init();
266    MX_RNG_Init();
267    /* USER CODE BEGIN 2 */
268
269
270    msg_info("\n");
271    msg_info("*********************************************************************************\n");
272    msg_info("***                    STM32 based AliIoT Client Demo                         ***\n");
273    msg_info("***                        With TLS and FREERTOS                              ***\n");
274    msg_info("***              FW version %d.%d.%d - %s, %s              ***\n",
275             FW_VERSION_MAJOR, FW_VERSION_MINOR, FW_VERSION_PATCH, __DATE__, __TIME__);
276    msg_info("*********************************************************************************\n");
277    msg_info("\n");
278
279    MX_MEMS_Init();
280    BSP_WIFI_Init();
281
282    /* USER CODE END 2 */
283
284    /* Call init function for freertos objects (in freertos.c) */
285    MX_FREERTOS_Init();
286
287    /* Start scheduler */
288    osKernelStart();
```

```
292    /* Infinite loop */
293    /* USER CODE BEGIN WHILE
294    while (1)
295    {
296      /* USER CODE END WHILE
297
298      /* USER CODE BEGIN 3 *
299    }
300    /* USER CODE END 3 */
301  }
```

# 项目例程流程图



<main.c>

HAL_Init

SystemClock_Config

MX_GPIO_Init
MX_LPUART1_UART_Init
MX_USART3_UART_Init
MX_RNG_Init

msg_info (…..)
MX_MEMS_Init
BSP_WIFI_Init

MX_FREERTOS_Init

osKernelStart

while (1);

<freeRTOS.c>

建立5个同步信号量
信号量处于等待状态
建立3个任务

<main.c>

ALIIOTKIT : AliIotKit_Thread
>> cloud_test
>> mqtt_client_example

LED_CTL : LED_CTL_Thread
……

Sense_Task : Sense_Task_Thread
>> MX_MEMS_Process

life.augmented

阿里云

# 项目例程流程图



## 初始化

系统初始化
（时钟，GPIO，串口，I2C，RNG，sensor 模块）

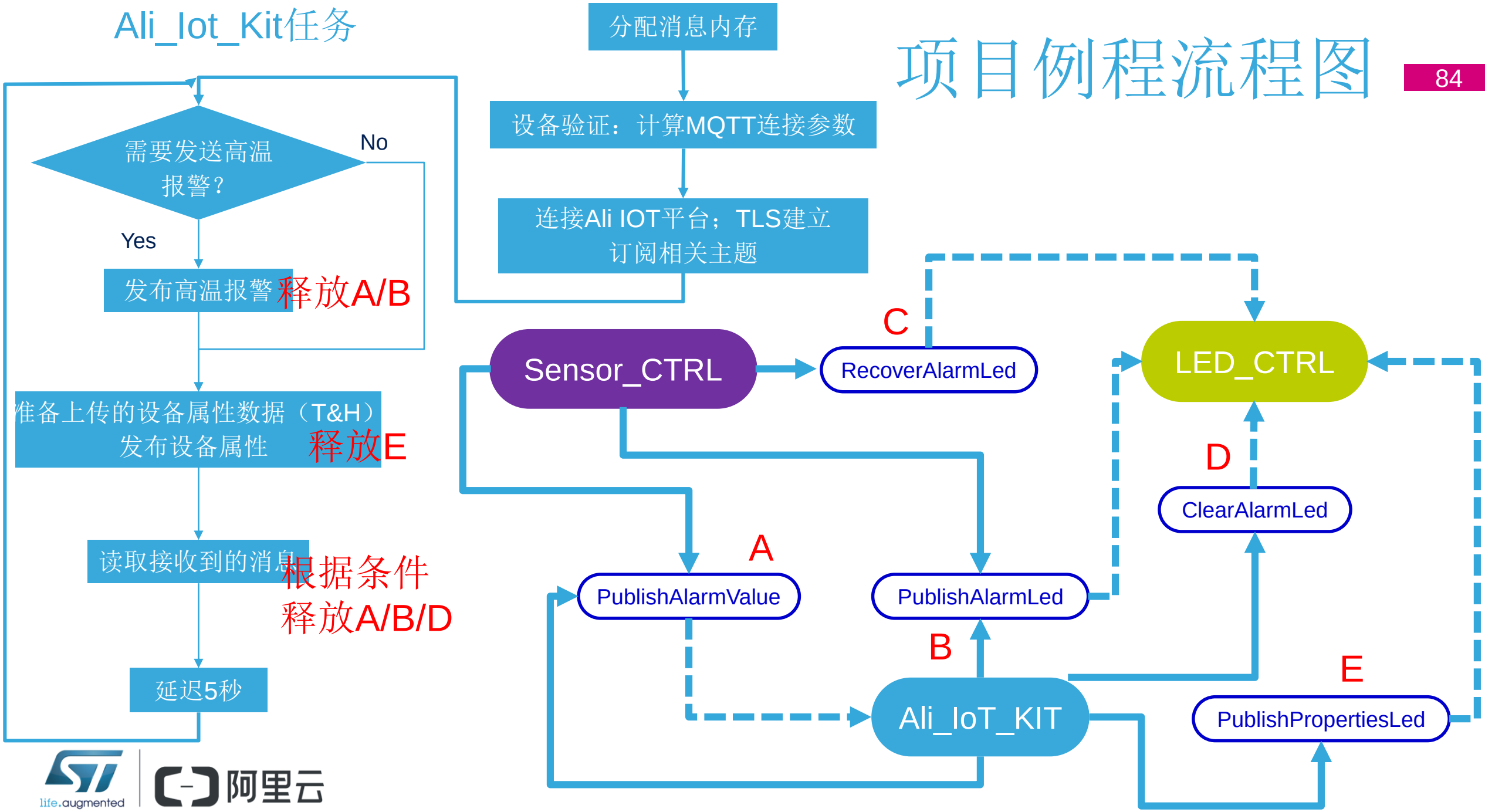
平台初始化
（wifi 模块，获取wifi连接参数，连接热点）

FREERTOS初始化
（LED/ALIIOT/SENSE 任务创建，信号量创建）

启动FreeRTOS

# 项目例程流程图

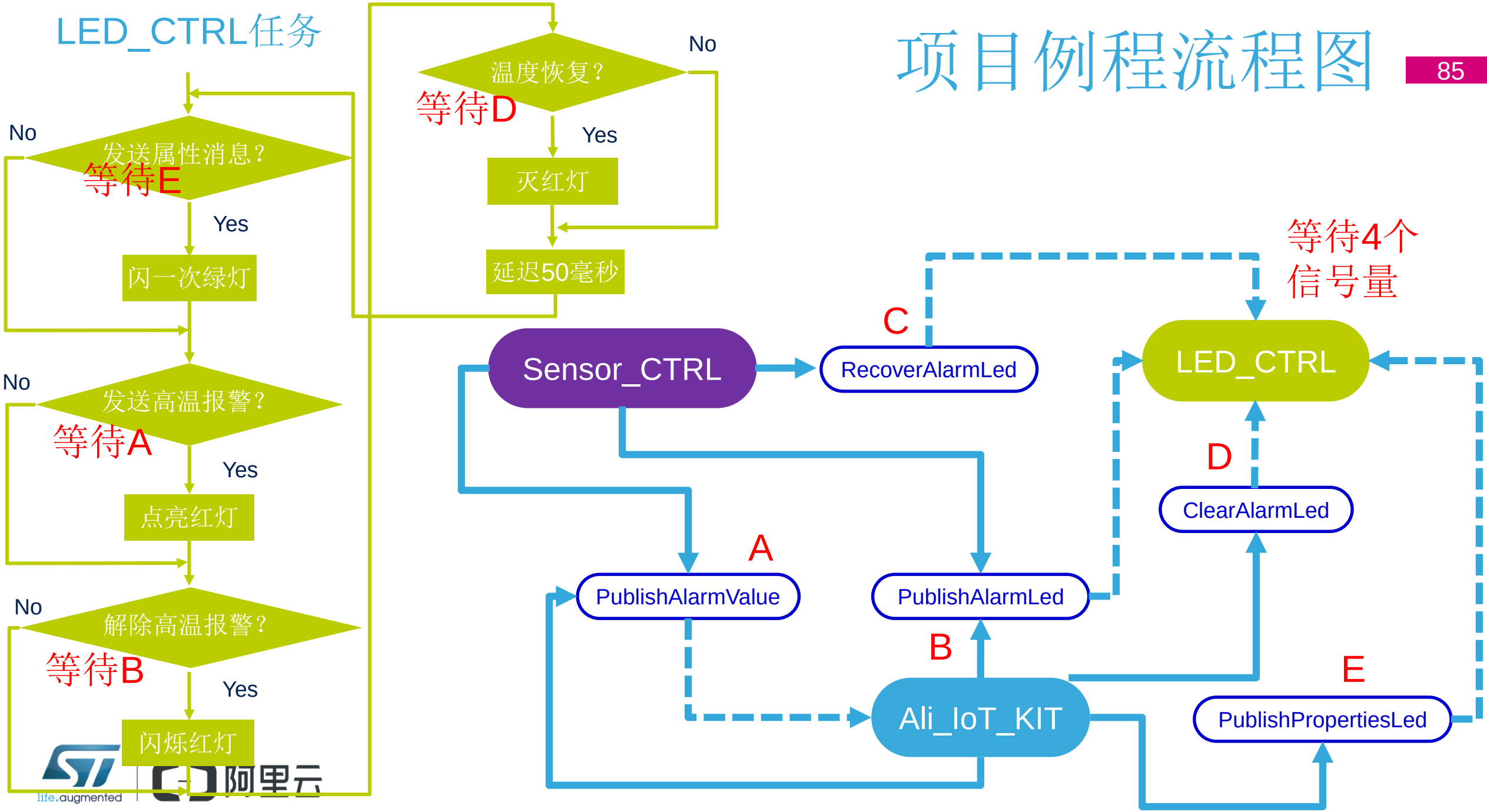





life.augmented 阿里云

# 项目例程流程图

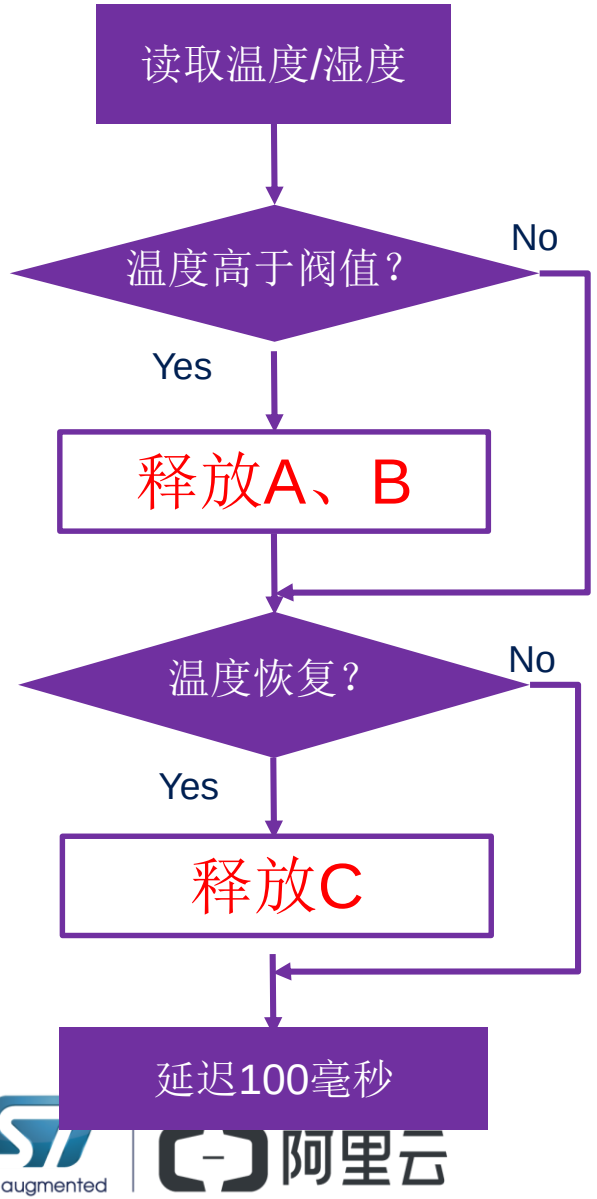


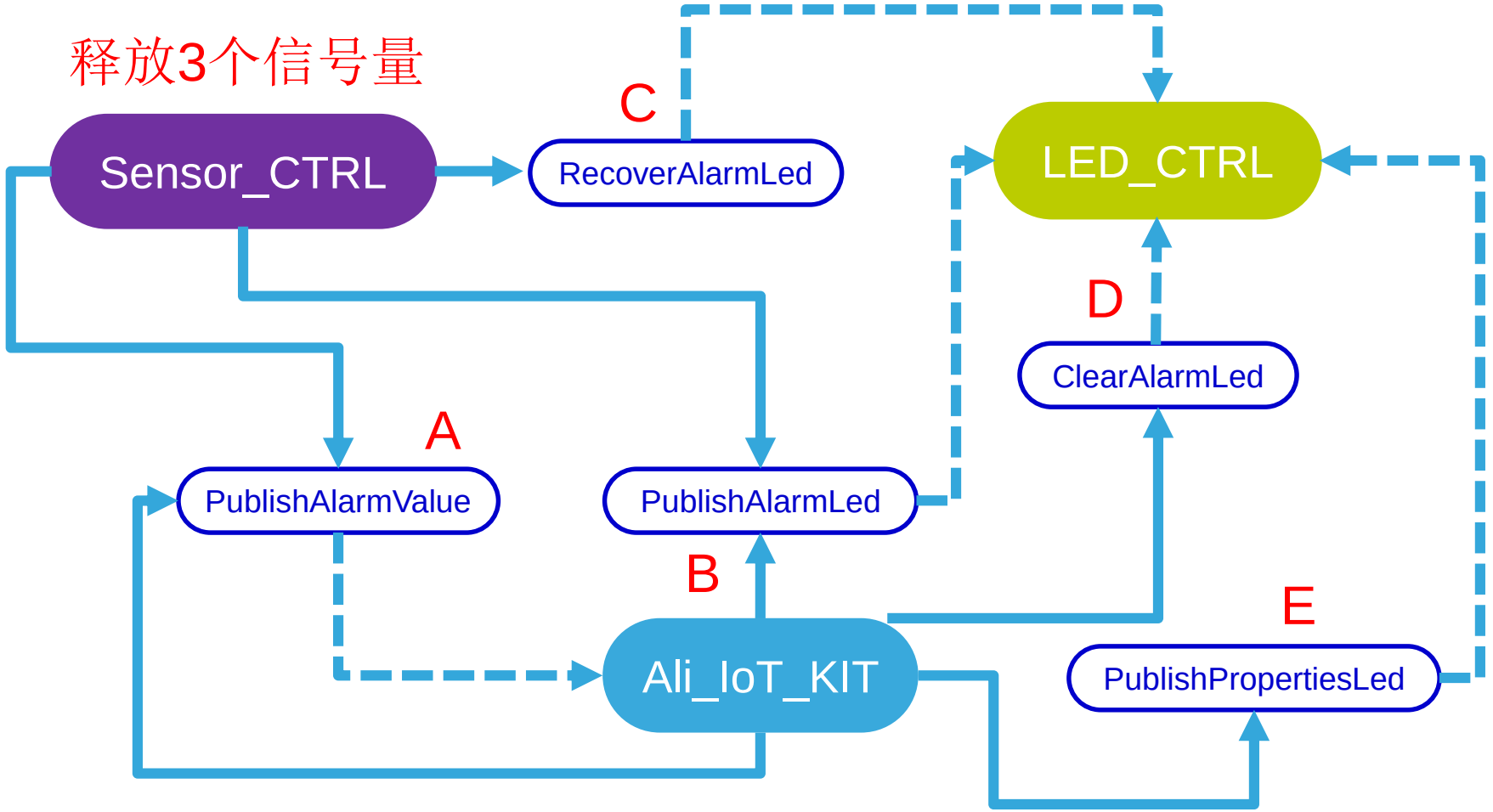

LED_CTRL任务
温度恢复？
No
等待D
Yes
灭红灯
延迟50毫秒
No
发送属性消息？
等待E
Yes
闪一次绿灯
No
发送高温报警？
等待A
Yes
点亮红灯
No
解除高温报警？
等待B
Yes
闪烁红灯
Sensor_CTRL
C
RecoverAlarmLed
等待4个
信号量
LED_CTRL
D
ClearAlarmLed
A
PublishAlarmValue
PublishAlarmLed
B
Ali_IoT_KIT
E
PublishPropertiesLed

# Sensor_CTRL任务

# 项目例程流程图





# 内存占用



- 总内存占用
  - Flash: 140 K字节
  - Ram：20 K字节
- 主要模块内存占用

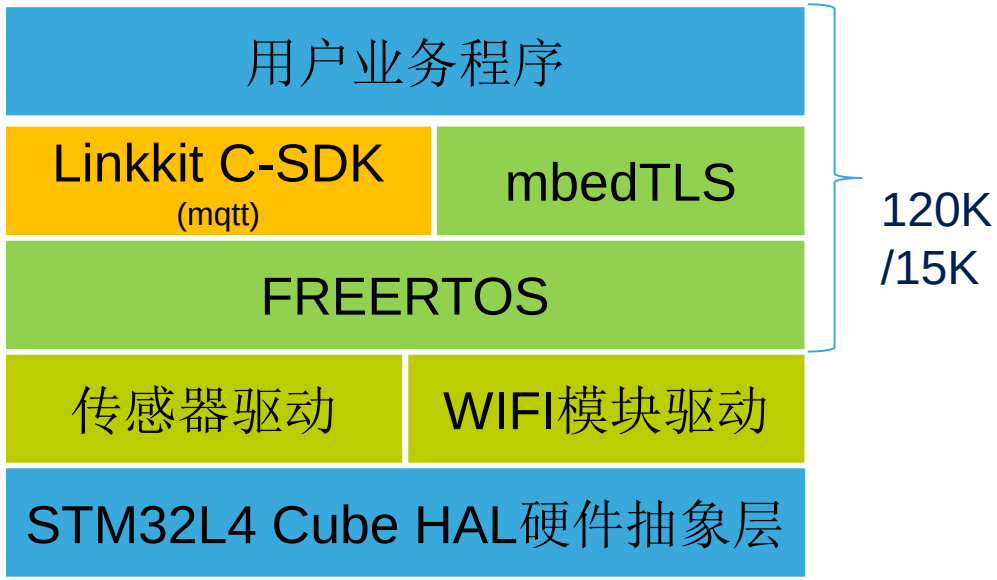
| 模块 | Flash-code | Flash-data | RAM |
|---|---|---|---|
| Wifi驱动 | 2.45 K | 168 | 6.3 K |
| Sensor驱动 | 2.1 K | 151 | 176 |
| STM32L4 HAL | 12.6 K | 11 | 36 |
| FreeRTOS | 5.3 K | 4 | 13 K |
| Linkkit C-SDK (mqtt) | 35.7 K | 11.1 K | 740 |
| mbedTLS | 40 K | 14.8 K | 60 |
| Application | 8.7 K | 5.5 K | 576 |

# 内存占用对比



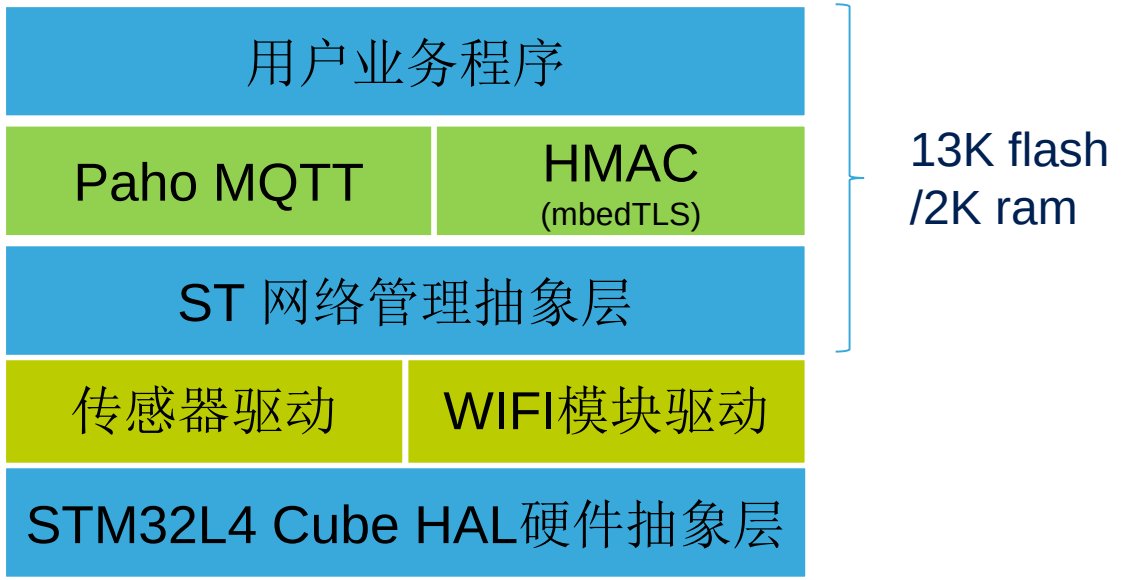
## 第二种连接方式

⌘ 140K Flash
⌘ 20K RAM




## 第一种连接方式

⌘ 52K Flash
⌘ 10K RAM



# 第四章内容预告



- 第一节：综合软件架构介绍
  - 软件架构，知识结构梳理
- 第二节：后端服务开发
  - 后端框架，初始化首个项目
- 第三节：后端服务开发体验
  - 系统应用后端开发详解