目录

[课程内容 2](#_Toc523498055)

[课程目标 2](#_Toc523498056)

[课程准备 3](#_Toc523498057)

[课程知识点 3](#_Toc523498058)

[MQTT协议简介 3](#_Toc523498059)

[MQTT协议特点 6](#_Toc523498060)

[C SDK整体框架介绍 7](#_Toc523498061)

[C SDK功能裁剪 10](#_Toc523498062)

[树莓派（3B+）准备工作 11](#_Toc523498063)

[物联网平台侧开发 13](#_Toc523498064)

[创建产品 13](#_Toc523498065)

[添加设备 14](#_Toc523498066)

[设备端开发 14](#_Toc523498067)

[代码开发 14](#_Toc523498068)

[编译C SDK 31](#_Toc523498069)

[运行调试 32](#_Toc523498070)

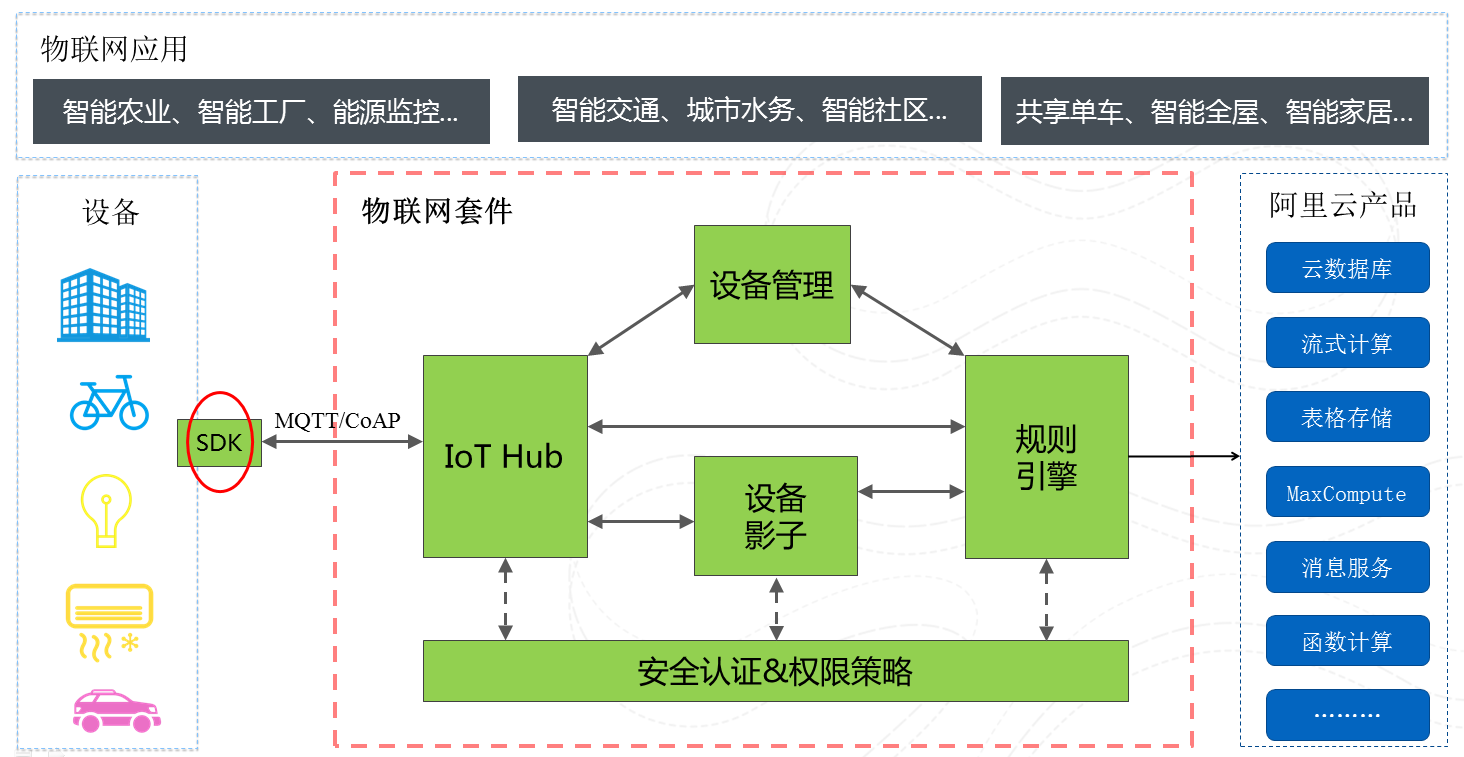
[接入与消息上报 32](#_Toc523498071)

[消息下发 34](#_Toc523498072)

[课程小结 37](#_Toc523498073)

**基于C SDK使用树莓派接入物联网平台**

当用户想要使用嵌入式设备接入物联网平台时，可以使用阿里云物联网平台提供的嵌入式设备快速接入平台的软件包。在这一章中我们以树莓派（3B+）为例，介绍如何在嵌入式设备中使用阿里云物联网平台提供的C语言版SDK接入阿里云物联网平台。



**课程内容**

* 下载并使用阿里云提供的帮助嵌入式设备快速接入云平台的C SDK
* C SDK代码解析
* 基于C SDK帮助树莓派（3B+）接入物联网云平台

**课程目标**

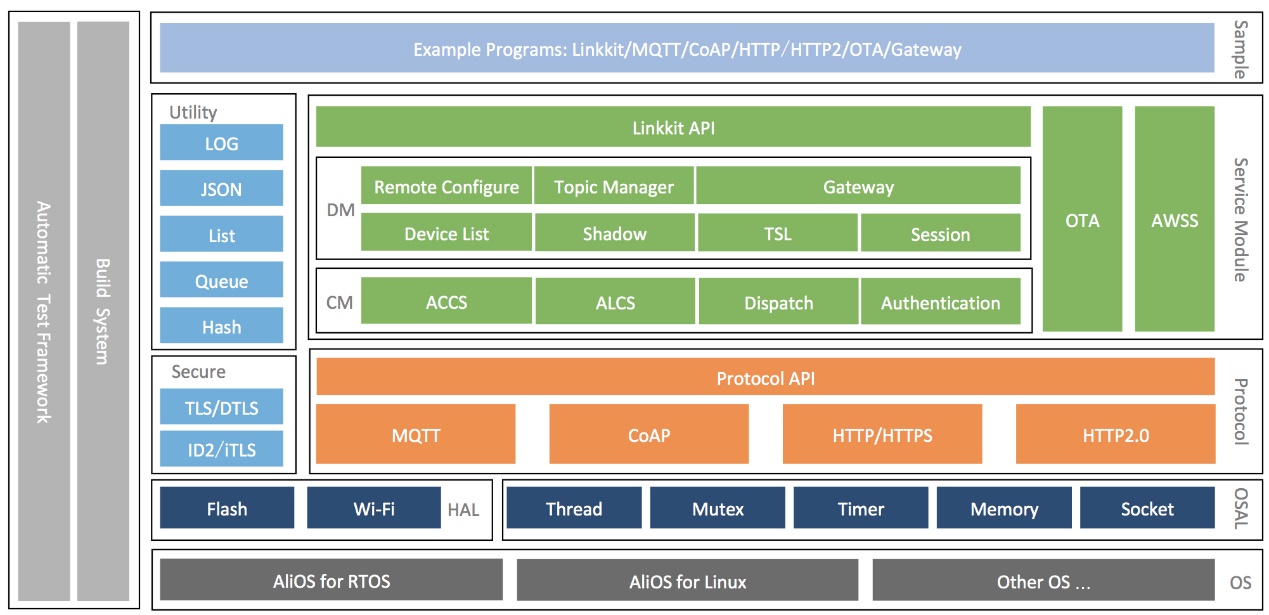
* 熟悉C SDK代码使用方法
* 学习MQTT协议相关知识，了解其发布消息和订阅Topic机制
* 学习嵌入式设备接入物联网云平台流程

**课程准备**

* 运行Windows系统的 PC机一台
* 树莓派（3B+）一块
* micro USB连接线
* 开通阿里云物联网平台*产品*

**课程知识点**

**C SDK整体框架介绍**



C语言版本的SDK是运行在嵌入式设备中，为设备提供阿里云物联网平台接口的软件开发工具包。SDK的架构框图如上图所示， SDK在操作系统上层建立了抽象层，将与操作系统相关的定时器、线程、互斥量、内存管理和套接字等抽象为操作系统抽象层，将与具体硬件设备相关的FLASH和Wi-Fi抽象为硬件抽象层。通过以上抽象设计便于开发者将SDK针对不同的操作系统和硬件设备进行适配后得到一致的功能和体验。

SDK在抽象层的基础上实现了对多种通信协议的支持，以适应不同设备的通信需求与工作环境。物联网通信协议中常用的MQTT协议，CoAP协议，HTTP协议等都以高效轻量的协议栈的方式集成于SDK中。SDK为不同的协议提供了标准的接口，开发者通过引入SDK，可以使设备具有快速入网，稳定通信的能力。在阿里云物联网平台上，不同通信协议的注册设备之间可以进行数据传输与指令下发，与SDK中包含的多种通信协议可以进行无缝的对接，快速实现不同物理设备之间的通信功能。在通信协议部分SDK还具有高效的安全能力，使用ID2认证和TLS/DTLS安全协议，对设备身份进行认证并对用户传输的数据提供保密性和数据完整性。

SDK在通信协议层的基础上建立了服务模块层。设备在SDK提供的软件框架下接入阿里云物联网时，也具备了物模型、设备影子、远程更新等多种高级应用的能力。SDK在服务模块层中封装了Linkkit应用接口，将不同的功能模块封装在SDK统一的接口中。从而屏蔽了底层的实现，为上层用户应用提供一致的应用程序接口。

SDK在封装了服务模块所用到的系统功能，包括设备调试所需的日志打印、设备通信常用的JSON对象解析等功能。SDK为物联网设备开发中所需的多种系统功能提供标准的接口，便于开发者在硬件抽象层的基础上快速便捷的调用所需的系统功能。在用户应用示例中，SDK包含了MQTT，CoAP，HTTP，OTA等多个示例demo，开发者在应用demo的基础上可以快速开发出自己的系统应用。

SDK中所有函数与功能都是在C语言上实现的，遵循GNU标准规范。在GUN编译环境下，开发者可以使用make工具对项目进行编译和链接，得到可执行文件快速对项目进行验证。SDK包含了三层文件目录，第一层是系统设置，应用示例和资源文件目录。第二层中，应用示例下包含mqtt，coap，device-shadow，http和ota五个示例文件夹；资源文件目录下包含通信协议文件夹和系统功能文件夹。第三层中是实现文件与编译配置文件。SDK的目录与文件组成如下。

|  |
| --- |
| +-- build-rules : 编译构建系统, 基于GNU Make和bash脚本  +-- doc : 阿里云*产品*帮助文档页面(https://help.aliyun.com/product/30520.html)的文档镜像  +-- layout.mk : 编译系统配置, 指定SDK的目录排布  +-- LICENSE : 软件许可证, 物联网套件设备端SDK使用的是Apache-2.0版本软件许可证  +-- makefile : 基于GNU Make编译SDK的Makefile  +-- make.settings : 功能裁剪配置, 可编辑该文件裁剪通信模块如MQTT|CoAP, 或裁剪服务模块如OTA|Shadow  +-- README.md : 快速开始导引, 可浏览该文件快速体验如何注册自己的阿里云设备ID, 并结合设备端SDK连接  +-- sample : 例程目录, 演示通信模块和服务模块的使用  | +-- mqtt : 演示如何使用通信模块MQTT的API  | +-- 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  | +-- ota : 演示如何使用服务模块OTA的API  +-- src  +-- sdk-impl : SDK的接口层, 提供总体的头文件, 和一些API的接口封装  +-- sdk-tests : SDK的单元测试  +-- mqtt : 通信模块, 实现以MQTT协议接入  +-- guider : 通信模块, 允许以HTTPS+MQTT协议接入, 保持向下兼容  +-- coap : 通信模块, 实现以CoAP协议接入  +-- http : 通信模块, 实现以HTTP协议接入  +-- ota : 服务模块, 实现基于MQTT|CoAP+HTTP+TLS的固件下载通道  +-- shadow : 服务模块, 实现设备影子  +-- platform : 硬件平台抽象层, 存放不同嵌入式设备的抽象层接口实现, 支撑以上模块形成跨平台  +-- import : 外部输入目录, 存放芯片/模组厂商提供的头文件/二进制库, 如高通NBIoT芯片MDM9206和芯讯通开发板SIM7000C相关文件  +-- configs : 硬件平台编译配置, 存放编译系统需要读入的各平台编译方式, 如交叉编译工具链的设置, 软件模块的裁剪等  +-- scripts : 编译过程将要外部引用的脚本, 用户不必关注  +-- packages : SDK引用的外部软件模块, 用户不必关注  +-- log : 基础模块, 实现运行日志  +-- system : 基础模块, 实现全局信息保存, 如预置的TLS/DTLS连接时的阿里云官方根证书, 设备标识ID等  +-- tls : 基础模块, 实现TLS/DTLS, 来自裁剪过的开源软件mbedtls  +-- utils : 基础模块, 实现工具函数, 如连接鉴权时的SHA1摘要计算, 或NTP服务器的epoch对时等 |

**C SDK功能裁剪**

SDK支持根据配置文件进行快速裁剪以适配具有不同资源的物联网设备。SDK的裁剪是通过修改第一层目录中的make.setting文件中相关变量的值实现的，根据文件中配置变量的值，系统会选择对应的文件进行编译，得到具备不同功能的可执行文件。在裁剪的过程中，我们需要配置FEATURE\_MQTT\_COMM\_ENABLED ，FEATURE\_MQTT\_DIRECT，FEATURE\_MQTT\_DIRECT\_NOTLS，FEATURE\_COAP\_COMM\_ENABLED ， FEATURE\_HTTP\_COMM\_ENABLED等多个变量的值来使能不同的通道开关。部分配置说明如下表，具体配置 在阿里云官网会不断的同步更新，以官网相关文档为准。

|  |  |
| --- | --- |
| 配置选项 | 含义 |
| FEATURE\_MQTT\_COMM\_ENABLED | 是否使能MQTT通道功能的总开关 |
| FEATURE\_MQTT\_DIRECT | 用MQTT直连模式或HTTPS三方认证模式做设备认证 |
| FEATURE\_MQTT\_DIRECT\_NOTLS | MQTT直连模式认证时, 是否关闭MQTT over TLS |
| FEATURE\_COAP\_COMM\_ENABLED | 是否使能CoAP通道功能的总开关 |
| FEATURE\_HTTP\_COMM\_ENABLED | 是否使能Https通道功能的总开关 |

我们可以对配置选项进行不同的赋值选择我们需要的功能而关闭其他的功能，以节省嵌入式设备上的资源。例如，当我们只需要MQTT协议而不需要其他通信方式时，可以将配置文件中变量设置为对应的值。

|  |
| --- |
| FEATURE\_MQTT\_COMM\_ENABLED = y  FEATURE\_MQTT\_DIRECT = y  FEATURE\_MQTT\_DIRECT\_NOTLS = n  FEATURE\_COAP\_COMM\_ENABLED = n  FEATURE\_HTTP\_COMM\_ENABLED = n |

**树莓派（3B+）准备工作**

在进行C SDK的开发时，我们需要准备好完整的开发环境。C SDK默认支持Linux，Win\_mingw32和Win\_VC60平台环境，在其他平台下开发时需要对硬件抽象层进行适配。

本次课程以树莓派为例，介绍真实设备基于C SDK的接入开发。在进行开发和接入之前，首先介绍在树莓派3B+平台上开发环境的准备过程。

1、安装Raspbian

在安装系统前要确保手头拥有如下工具：树莓派 3B+、SD卡（建议容量最少8G）、读卡器、键盘、HDMI接口线、带有HDMI输入接口的显示器和Micro USB接口线。

安装步骤：

* 下载NOOBS系统安装工具、SD卡格式化工具。
* 格式化SD卡。
* 解压下载的NOOBS压缩包，将解压后的所有文件直接复制到格式化之后的SD卡。
* 插卡上电后打开屏幕按照引导流程安装系统，安装成功后配置WiFi网络。

首先我们到树莓派官方网站下载系统固件，固件下载地址为：<https://www.raspberrypi.org/downloads/>，进入网站后选择NOOBS，之后下载如图所示NOOBS工具。



然后开始下载SD卡格式化工具，我们使用官方推荐的链接地址下载：https://www.sdcard.org/downloads/formatter\_4/eula\_windows/。下载并安装完成后，使用读卡器将SD卡插入电脑，打开该工具直接点击Format即可。

SD卡格式化之后将NOOBS解压出来的所有文件直接拷贝到SD卡中。拷贝完毕后将SD卡插到树莓派中，将显示器和键盘与树莓派连接，上电启动后按照引导流程安装RASPBBERY系统。

2、安装工具软件

操作系统安装成功之后，我们需要安装一些开发必备的工具。在下载工具软件之前，我们首先要使树莓派接入网络，启动树莓派后使用图形界面配置Wi-Fi账号与密码。

连接Wi-Fi成功后，我们需要下载一些运行C SDK必备的依赖软件。C SDK的最新源码被托管在官方GitHub平台上，为了获取最新的源码，首先需要安装git工具。然后使用git工具将最新的源码克隆至本地。对源码进行编译时需要gcc编译器与make工具。幸运的是，我们可以通过一条简单的指令完成所有工具软件的安装。安装指令为：sudo apt-get install -y build-essential make git gcc

软件安装完成后，为了检测软件是否安装成功，我们可以通过软件的版本查询指令查看软件的版本。系统返回对应的版本号码说明软件安装成功，若系统返回错误，请检查网络后重新安装。版本查询指令为：

|  |  |  |
| --- | --- | --- |
| 工具软件 | 版本查询指令 | 系统返回 |
| git | sudo git --version | git version 2.7.4 |
| gcc | sudo gcc -v | gcc version 5.4.0 20160609 |
| make | sudo make --version | GNU Make 4.1 |

3、获取C SDK源码

在git安装成功后，我们便可以从官方网站https://github.com/aliyun/iotkit-embedded获取最新的C SDK源码了，通过clone指令将源码克隆至本地home目录下。在文件目录的操作过程中常常需要管理员权限，所以我们先获取系统管理员权限。获取权限的指令为 su，输入指令后，根据系统的提示输入管理员密码便可获取权限。获取管理员权限指令过程如下。

|  |
| --- |
| pi@xxx:~$ su  Password:  root@xxx:/home # |

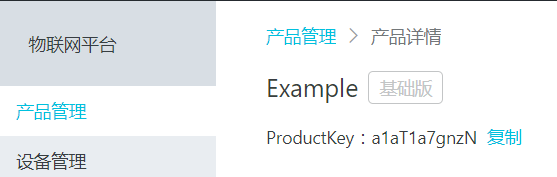
获取管理员权限后，接着进入系统home目录下克隆源码。开发者也可根据实际需求设置为其他目录。运行cd ~/指令进入系统home目录，然后运行git clone https://github.com/aliyun/iotkit-embedded指令克隆源码至home目录下。获取源码指令过程如下。

|  |
| --- |
| root@xxx:# cd ~/  root@xxx:~# git clone https://github.com/aliyun/iotkit-embedded  Cloning into 'iotkit-embedded'...  remote: Counting objects: 13046, done.  remote: Compressing objects: 100% (3/3), done.  remote: Total 13046 (delta 0), reused 1 (delta 0), pack-reused 13043  Receiving objects: 100% (13046/13046), 61.31 MiB | 81.00 KiB/s, done.  Resolving deltas: 100% (5349/5349), done.  Checking connectivity... done. |

**物联网平台侧开发**

**创建产品**

我们首先需要在物联网平台上创建好*产品*和设备，登录物联网平台控制台https://iot.console.aliyun.com/，选择区域为“华东2（上海）”。在“*产品*管理”页面，点击右侧的“创建*产品*”按钮。输入*产品*名称，并选择节点类型后，确定即可。注意：*产品*名称必须保证在账号内唯一。此处创建的示例*产品*名称为Example，节点类型为“设备”。



**添加设备**

*产品*创建成功后，进入“设备管理”栏，选择刚刚创建好的基础版*产品*，点击“添加设备”，输入设备名称即可。用户可以自定义设备名称，保证名称在*产品*内唯一即可。



设备创建完成后，平台会自动生成该设备的三元组信息，该信息在后续设备端开发中需要写入设备，用于设备接入物联网平台时的鉴权。完成*产品*和设备的创建之后，接下来我们进行设备端的开发工作。

**设备端开发**

**代码开发**

我们将在C SDK的MQTT例程基础上开发一个树莓派与物联网云平台双向通信的应用。应用中设备连接到阿里云IoT，并通过MQTT协议进行Pub/Sub通信将用户数据发送到云平台上注册设备的topic。接下来我们开始SDK的开发工作。

首先进入目录~/iotkit-embedded/sample/mqtt，在该目录下有一个文本文件CMakeLists.txt，该文件为编译列表文件。还有四个c文件，分别为mqtt\_multi\_region-example.c，mqtt\_rrpc-example.c，mqtt-example.c，mqtt\_multi\_thread-example.c文件。

其中mqtt\_multi\_region-example.c程序支持多区域服务站点接入，通过设置预编译命令对接入的站点进行设置，可以使设备接入美国，日本，德国等不同的站点以满足不同地区开发者的应用需求。mqtt\_rrpc-example.c文件是在mqtt协议上实现的同步通信示例，适用于云端指令下行并同步设备返回结果的场景。mqtt-example.c文件是基础的mqtt例程，示例中设备接入华东站点进行数据通信。mqtt\_multi\_thread-example.c文件是多线程mqtt通信的例程，例程中对不同的topic创建了不同的线程，每个线程管理对应的topic。本节我们以mqtt-example.c文件为基础进行应用开发。在~/iotkit-embedded/sample/mqtt目录下运行vim mqtt-example.c命令打开文件。mqtt-example.c文件是基础的mqtt例程，文件中包含一个mqtt客户端函数和一个消息解析函数，接下来我们对代码进行细致的分析（考虑到文件篇幅，文件中部分内容有所省略，在关键的指令后注释上序号并分段进行解释）。

|  |
| --- |
| #include<stdio.h>//①  #include <stdlib.h>  #include <string.h>  #include <stdarg.h>  #include "iot\_import.h"  #include "iot\_export.h"  #ifdefined(MQTT\_ID2\_AUTH)&&defined(ON\_DAILY) //②  #define PRODUCT\_KEY "9rx2yMNV5l0"  #define DEVICE\_NAME "sh\_online\_sample\_mqtt"  #define DEVICE\_SECRET "v9mqGzepKEphLhXmAoiaUIR2HZ7XwTky"  /\*省略部分内容\*/  #define PRODUCT\_KEY "yfTuLfBJTiL"  #define DEVICE\_NAME "TestDeviceForDemo"  #define DEVICE\_SECRET "fSCl9Ns5YPnYN8Ocg0VEel1kXFnRlV6c"  #endif  char \_\_product\_key[PRODUCT\_KEY\_LEN + 1];  char \_\_device\_name[DEVICE\_NAME\_LEN + 1];  char \_\_device\_secret[DEVICE\_SECRET\_LEN + 1];  #define TOPIC\_UPDATE "/"PRODUCT\_KEY"/"DEVICE\_NAME"/update"  #define TOPIC\_ERROR "/"PRODUCT\_KEY"/"DEVICE\_NAME"/update/error"  #define TOPIC\_GET "/"PRODUCT\_KEY"/"DEVICE\_NAME"/get"  #define TOPIC\_DATA "/"PRODUCT\_KEY"/"DEVICE\_NAME"/data"  /\*省略部分内容\*/  Void event\_handle(void \*pcontext,void \*pclient,iotx\_mqtt\_event\_msg\_pt msg) //③  {  uintptr\_t packet\_id = (uintptr\_t)msg->msg;  iotx\_mqtt\_topic\_info\_pt topic\_info = (iotx\_mqtt\_topic\_info\_pt)msg->msg;  switch (msg->event\_type) {  case IOTX\_MQTT\_EVENT\_UNDEF:  EXAMPLE\_TRACE("undefined event occur.");  break;  case IOTX\_MQTT\_EVENT\_DISCONNECT:  EXAMPLE\_TRACE("MQTT disconnect.");  break;  /\*省略部分内容\*/  case IOTX\_MQTT\_EVENT\_PUBLISH\_NACK:  EXAMPLE\_TRACE("publish nack, packet-id=%u", (unsigned int)packet\_id);  break;  case IOTX\_MQTT\_EVENT\_PUBLISH\_RECVEIVED:  EXAMPLE\_TRACE("topic message arrived but without any related handle: topic=%.\*s, topic\_msg=%.\*s",  topic\_info->topic\_len,  topic\_info->ptopic,  topic\_info->payload\_len,  topic\_info->payload);  break;  case IOTX\_MQTT\_EVENT\_BUFFER\_OVERFLOW:  EXAMPLE\_TRACE("buffer overflow, %s", msg->msg);  break;  default:  EXAMPLE\_TRACE("Should NOT arrive here.");  break;  }  } |

文件开始第①部分通过include指令包含了四个系统头文件，分别引入了系统输入与输出，内存操作，可变参数支持等标准库。然后包含了iot\_import.h和iot\_export.h两个头文件。iot\_import.h文件与iot\_export.h文件位于~/iotkit-embedded/src/sdk-impl目录下，在iot\_import.h文件中申明了硬件抽象层的应用程序接口，在iot\_export.h申明了系统功能函数应用程序接口。头文件将硬件抽象层接口与系统功能接口引入mqtt-example文件，便于系统功能被用户调用。

文件中第②部分为设备鉴权信息的配置指令，配置指令中运用宏命令定义了PRODUCT\_KEY，DEVICE\_NAME和DEVICE\_SECRET。这三个字符串分别对应上一节在物联网云平台上注册设备的*产品*密钥，设备名称与设备密钥。这三个信息用于设备接入物联网云平台时的鉴权与mqtt协议中topic的定义。我们在例程上进行开发时，需要将对应的信息替换为上一节中注册生成的信息。

文件中第③部分为设备接收信息处理函数。event\_handle函数包含三个参数第一个参数void \*pcontext为系统进程切换到event\_handle时所需的上下文，本例程中event\_handle函数为用户调用，所以该变量赋值为NULL。event\_handle函数的第二个参数为void \*pclient，pclient为一个void指针，指向event\_handle函数的调用者的mqtt资源，pclient变量实现了资源信息的传递。event\_handle函数的第三个参数为iotx\_mqtt\_event\_msg\_pt msg，该变量的内容是event\_handle函数所需要解析的对象，event\_handle函数从该对象中获取消息的类型，内容和状态。

|  |
| --- |
| static void \_demo\_message\_arrive(void \*pcontext, void \* pclient, iotx\_mqtt\_event\_msg\_pt msg) //④  {  iotx\_mqtt\_topic\_info\_pt ptopic\_info = (iotx\_mqtt\_topic\_info\_pt) msg->msg;  /\* print topic name and topic message \*/  EXAMPLE\_TRACE("----");  EXAMPLE\_TRACE("packetId: %d", ptopic\_info->packet\_id);  EXAMPLE\_TRACE("Topic: '%.\*s' (Length: %d)",  ptopic\_info->topic\_len,  ptopic\_info->ptopic,  ptopic\_info->topic\_len);  EXAMPLE\_TRACE("Payload: '%.\*s' (Length: %d)",  ptopic\_info->payload\_len,  ptopic\_info->payload,  ptopic\_info->payload\_len);  EXAMPLE\_TRACE("----");  } |

文件中第④部分为设备接收消息处理函数。\_demo\_message\_arrive函数包含三个参数第一个参数void \*pcontext为系统进程切换时所需的上下文，第二个参数为void \*pclient，pclient为一个void指针，指向调用者的mqtt资源，第三个参数为iotx\_mqtt\_event\_msg\_pt msg，该变量的内容是函数所需要解析的对象。与event\_handle函数不同的是，\_demo\_message\_arrive只解析物联网云平台下行的消息，不对设备接入过程中的消息解析。

|  |
| --- |
| #ifndef MQTT\_ID2\_AUTH  int mqtt\_client(void) //⑤  {  int rc = 0, msg\_len, cnt = 0;  void \*pclient;  iotx\_conn\_info\_pt pconn\_info;  iotx\_mqtt\_param\_t mqtt\_params;  iotx\_mqtt\_topic\_info\_t topic\_msg;  char msg\_pub[128];  char \*msg\_buf = NULL, \*msg\_readbuf = NULL;  if (NULL == (msg\_buf = (char \*)HAL\_Malloc(MQTT\_MSGLEN))) {  EXAMPLE\_TRACE("not enough memory");  rc = -1;  goto do\_exit;  }  if (NULL == (msg\_readbuf = (char \*)HAL\_Malloc(MQTT\_MSGLEN))) {  EXAMPLE\_TRACE("not enough memory");  rc = -1;  goto do\_exit;  }  HAL\_GetProductKey(\_\_product\_key); //⑥  HAL\_GetDeviceName(\_\_device\_name);  HAL\_GetDeviceSecret(\_\_device\_secret);  /\* Device AUTH \*/  if (0 != IOT\_SetupConnInfo(\_\_product\_key, \_\_device\_name, \_\_device\_secret, (void \*\*)&pconn\_info)) {  EXAMPLE\_TRACE("AUTH request failed!");  rc = -1;  goto do\_exit;  }  /\* Initialize MQTT parameter \*/  memset(&mqtt\_params, 0x0, sizeof(mqtt\_params)); //⑦  mqtt\_params.port = pconn\_info->port;  mqtt\_params.host = pconn\_info->host\_name;  mqtt\_params.client\_id = pconn\_info->client\_id;  mqtt\_params.username = pconn\_info->username;  mqtt\_params.password = pconn\_info->password;  mqtt\_params.pub\_key = pconn\_info->pub\_key;  mqtt\_params.request\_timeout\_ms = 2000;  mqtt\_params.clean\_session = 0;  mqtt\_params.keepalive\_interval\_ms = 60000;  mqtt\_params.pread\_buf = msg\_readbuf;  mqtt\_params.read\_buf\_size = MQTT\_MSGLEN;  mqtt\_params.pwrite\_buf = msg\_buf;  mqtt\_params.write\_buf\_size = MQTT\_MSGLEN;  mqtt\_params.handle\_event.h\_fp = event\_handle;  mqtt\_params.handle\_event.pcontext = NULL;  /\* Construct a MQTT client with specify parameter \*/  pclient = IOT\_MQTT\_Construct(&mqtt\_params);  if (NULL == pclient) {  EXAMPLE\_TRACE("MQTT construct failed");  rc = -1;  goto do\_exit;  }  /\* Initialize topic information \*/  memset(&topic\_msg, 0x0, sizeof(iotx\_mqtt\_topic\_info\_t));  strcpy(msg\_pub, "update: hello! start!"); //⑧  topic\_msg.qos = IOTX\_MQTT\_QOS1;  topic\_msg.retain = 0;  topic\_msg.dup = 0;  topic\_msg.payload = (void \*)msg\_pub;  topic\_msg.payload\_len = strlen(msg\_pub);  rc = IOT\_MQTT\_Publish(pclient, TOPIC\_UPDATE, &topic\_msg);  if (rc < 0) {  IOT\_MQTT\_Destroy(&pclient);  EXAMPLE\_TRACE("error occur when publish");  rc = -1;  goto do\_exit;  }  EXAMPLE\_TRACE("\n publish message: \n topic: %s\n payload: \%s\n rc = %d", TOPIC\_UPDATE, topic\_msg.payload, rc);  /\* Subscribe the specific topic \*/  rc = IOT\_MQTT\_Subscribe(pclient, TOPIC\_DATA, IOTX\_MQTT\_QOS1, \_demo\_message\_arrive, NULL);  if (rc < 0) {  IOT\_MQTT\_Destroy(&pclient);  EXAMPLE\_TRACE("IOT\_MQTT\_Subscribe() failed, rc = %d", rc);  rc = -1;  goto do\_exit;  }  /\* Initialize topic information \*/  memset(msg\_pub, 0x0, 128);  strcpy(msg\_pub, "data: hello! start!");  memset(&topic\_msg, 0x0, sizeof(iotx\_mqtt\_topic\_info\_t));  topic\_msg.qos = IOTX\_MQTT\_QOS1;  topic\_msg.retain = 0;  topic\_msg.dup = 0;  topic\_msg.payload = (void \*)msg\_pub;  topic\_msg.payload\_len = strlen(msg\_pub);  rc = IOT\_MQTT\_Publish(pclient, TOPIC\_DATA, &topic\_msg);  EXAMPLE\_TRACE("\n publish message: \n topic: %s\n payload: \%s\n rc = %d", TOPIC\_DATA, topic\_msg.payload, rc);  IOT\_MQTT\_Yield(pclient, 200);  do {  /\* Generate topic message \*/  cnt++;  msg\_len = snprintf(msg\_pub, sizeof(msg\_pub), "{\"attr\_name\":\"temperature\", \"attr\_value\":\"%d\"}", cnt);  if (msg\_len < 0) {  EXAMPLE\_TRACE("Error occur! Exit program");  rc = -1;  break;  }  topic\_msg.payload = (void \*)msg\_pub;  topic\_msg.payload\_len = msg\_len;  rc = IOT\_MQTT\_Publish(pclient, TOPIC\_DATA, &topic\_msg);  if (rc < 0) {  EXAMPLE\_TRACE("error occur when publish");  rc = -1;  break;  }  #ifdef MQTT\_ID2\_CRYPTO  EXAMPLE\_TRACE("packet-id=%u,publish topic msg='0x%02x%02x%02x%02x'...", (uint32\_t)rc, msg\_pub[0], msg\_pub[1], msg\_pub[2], msg\_pub[3] );  #else  EXAMPLE\_TRACE("packet-id=%u, publish topic msg=%s", (uint32\_t)rc, msg\_pub);  #endif  /\* handle the MQTT packet received from TCP or SSL connection \*/  IOT\_MQTT\_Yield(pclient, 200);  /\* infinite loop if running with 'loop' argument \*/  if (user\_argc >= 2 && !strcmp("loop", user\_argv[1])) {  HAL\_SleepMs(2000);  cnt = 0;  }  } while (cnt < 1);  IOT\_MQTT\_Yield(pclient, 200);  IOT\_MQTT\_Unsubscribe(pclient, TOPIC\_DATA);  IOT\_MQTT\_Yield(pclient, 200);  IOT\_MQTT\_Destroy(&pclient);  do\_exit:  if (NULL != msg\_buf) {  HAL\_Free(msg\_buf);  }  if (NULL != msg\_readbuf) {  HAL\_Free(msg\_readbuf);  }  return rc;  }  #endif /\* MQTT\_ID2\_AUTH \*/ |

第⑥部分为硬件抽象层提供的获取设备鉴权三元组的系统调用。在一型一密的动态注册方式中，通过该系统调用获取该设备对应的鉴权信息。获取鉴权信息后调用函数IOT\_SetupConnInfo将鉴权变量值注册到系统资源。

接下来的第⑦部分对mqtt资源参数进行了设置，包括地址，端口以及验证信息等多个参数。设置参数收调用IOT\_MQTT\_Construct函数构造一个mqtt客户端资源并返回给pclient指针。

第⑧部分调用strcpy函数设置消息内容为"update: hello! start!"，接着设置消息QoS等级为IOTX\_MQTT\_QOS1。然后调用IOT\_MQTT\_Publish函数将消息发送到物联网云平台。IOT\_MQTT\_Publish函数有三个参数，第一个参数pclient是mqtt资源对象指针，用于传递mqtt资源参数。第二个参数TOPIC\_UPDATE为mqtt协议中的topic，该参数的值对应到物联网云平台中设备的topic。也就是消息将要发送的topic。第三个参数topic\_msg为消息变量的地址，用于消息值的传递。IOT\_MQTT\_Publish调用的返回值保存于变量rc中，通过rc的值判断发送函数是否运行成功。若发送失败，则调用IOT\_MQTT\_Destroy函数析构客户端资源对象，释放内存并退出。

接着调用IOT\_MQTT\_Subscribe函数订阅云端topic。IOT\_MQTT\_Subscribe包含5个参数，第一个参数pclient为客户端资源指针，将客户端资源传入IOT\_MQTT\_Subscribe函数。第二个参数为TOPIC\_DATA，表示订阅的云端topic。第三个参数IOTX\_MQTT\_QOS1表示订阅的topic的消息传输QoS等级。第四个参数\_demo\_message\_arrive为消息接收解析函数指针，将消息解析函数注册到mqtt客户端资源中。最后一个参数为上下文。此处设置为NULL。IOT\_MQTT\_Subscribe订阅函数将返回值保存与变量rc中，根据rc的值判断订阅函数是否运行正常。若订阅失败，返回值为-1，析构客户端对象并释放资源。

消息订阅成功后调用IOT\_MQTT\_Publish函数进行消息的发送，每次发送消息后，调用IOT\_MQTT\_Yield函数，阻塞当前进程，切换至接收消息解析函数，对物联网云平台返回给设备的消息进行解析。接着运行了一个do{ }while( )循环，重复消息的发送与解析过程。

|  |
| --- |
| #ifdef MQTT\_ID2\_AUTH  #include "tfs.h"  char \_\_device\_id2[TFS\_ID2\_LEN + 1];  int mqtt\_client\_secure( )//⑨  {  int rc = 0, msg\_len, cnt = 0;  void \*pclient;  iotx\_conn\_info\_pt pconn\_info;  iotx\_mqtt\_param\_t mqtt\_params;  iotx\_mqtt\_topic\_info\_t topic\_msg;  char msg\_pub[128];  char \*msg\_buf = NULL, \*msg\_readbuf = NULL;  char topic\_update[IOTX\_URI\_MAX\_LEN] = {0};  char topic\_error[IOTX\_URI\_MAX\_LEN] = {0};  char topic\_get[IOTX\_URI\_MAX\_LEN] = {0};  char topic\_data[IOTX\_URI\_MAX\_LEN] = {0};  if (NULL == (msg\_buf = (char \*)HAL\_Malloc(MQTT\_MSGLEN))) {  EXAMPLE\_TRACE("not enough memory");  rc = -1;  goto do\_exit;  }  if (NULL == (msg\_readbuf = (char \*)HAL\_Malloc(MQTT\_MSGLEN))) {  EXAMPLE\_TRACE("not enough memory");  rc = -1;  goto do\_exit;  }  HAL\_GetProductKey(\_\_product\_key);  HAL\_GetID2(\_\_device\_id2);  /\* Device AUTH \*/  rc = IOT\_SetupConnInfoSecure(\_\_product\_key, \_\_device\_id2, \_\_device\_id2, (void \*\*)&pconn\_info);  if (rc != 0) {  EXAMPLE\_TRACE("AUTH request failed!");  goto do\_exit;  }  HAL\_Snprintf(topic\_update,IOTX\_URI\_MAX\_LEN,TOPIC\_UPDATE\_FMT,\_\_product\_key,\_\_device\_id2);  HAL\_Snprintf(topic\_error,IOTX\_URI\_MAX\_LEN,TOPIC\_ERROR\_FMT,\_\_product\_key,\_\_device\_id2);  HAL\_Snprintf(topic\_get,IOTX\_URI\_MAX\_LEN,TOPIC\_GET\_FMT,\_\_product\_key,\_\_device\_id2);  HAL\_Snprintf(topic\_data,IOTX\_URI\_MAX\_LEN,TOPIC\_DATA\_FMT,\_\_product\_key,\_\_device\_id2);  /\* Initialize MQTT parameter \*/  memset(&mqtt\_params, 0x0, sizeof(mqtt\_params));  mqtt\_params.port = pconn\_info->port;  mqtt\_params.host = pconn\_info->host\_name;  mqtt\_params.client\_id = pconn\_info->client\_id;  mqtt\_params.username = pconn\_info->username;  mqtt\_params.password = pconn\_info->password;  mqtt\_params.pub\_key = pconn\_info->pub\_key;  mqtt\_params.request\_timeout\_ms = 2000;  mqtt\_params.clean\_session = 0;  mqtt\_params.keepalive\_interval\_ms = 60000;  mqtt\_params.pread\_buf = msg\_readbuf;  mqtt\_params.read\_buf\_size = MQTT\_MSGLEN;  mqtt\_params.pwrite\_buf = msg\_buf;  mqtt\_params.write\_buf\_size = MQTT\_MSGLEN;  mqtt\_params.handle\_event.h\_fp = event\_handle;  mqtt\_params.handle\_event.pcontext = NULL;  /\* Construct a MQTT client with specify parameter \*/  pclient = IOT\_MQTT\_ConstructSecure(&mqtt\_params);  if (NULL == pclient) {  EXAMPLE\_TRACE("MQTT construct failed");  rc = -1;  goto do\_exit;  }  /\* Subscribe the specific topic \*/  rc = IOT\_MQTT\_Subscribe(pclient, topic\_data, IOTX\_MQTT\_QOS1, \_demo\_message\_arrive, NULL);  if (rc < 0) {  IOT\_MQTT\_Destroy(&pclient);  EXAMPLE\_TRACE("IOT\_MQTT\_Subscribe() failed, rc = %d", rc);  rc = -1;  goto do\_exit;  }  HAL\_SleepMs(1000);  /\* Initialize topic information \*/  memset(&topic\_msg, 0x0, sizeof(iotx\_mqtt\_topic\_info\_t));  strcpy(msg\_pub, "message: hello! start!");  topic\_msg.qos = IOTX\_MQTT\_QOS1;  topic\_msg.retain = 0;  topic\_msg.dup = 0;  topic\_msg.payload = (void \*)msg\_pub;  topic\_msg.payload\_len = strlen(msg\_pub);  rc = IOT\_MQTT\_Publish(pclient, topic\_data, &topic\_msg);  EXAMPLE\_TRACE("rc = IOT\_MQTT\_Publish() = %d", rc);  do {  /\* Generate topic message \*/  cnt++;  msg\_len = snprintf(msg\_pub, sizeof(msg\_pub), "{\"attr\_name\":\"temperature\", \"attr\_value\":\"%d\"}", cnt);  if (msg\_len < 0) {  EXAMPLE\_TRACE("Error occur! Exit program");  rc = -1;  break;  }  topic\_msg.payload = (void \*)msg\_pub;  topic\_msg.payload\_len = msg\_len;  rc = IOT\_MQTT\_Publish(pclient, topic\_data, &topic\_msg);  if (rc < 0) {  EXAMPLE\_TRACE("error occur when publish");  rc = -1;  break;  }  EXAMPLE\_TRACE("packet-id=%u, publish topic msg='0x%02x%02x%02x%02x'...",  (uint32\_t)rc, msg\_pub[0], msg\_pub[1], msg\_pub[2], msg\_pub[3] );  /\* handle the MQTT packet received from TCP or SSL connection \*/  IOT\_MQTT\_Yield(pclient, 200);  /\* infinite loop if running with 'loop' argument \*/  if (user\_argc >= 2 && !strcmp("loop", user\_argv[1])) {  HAL\_SleepMs(2000);  cnt = 0;  }  } while (cnt < 1);  IOT\_MQTT\_Unsubscribe(pclient, TOPIC\_DATA);  HAL\_SleepMs(200);  IOT\_MQTT\_Destroy(&pclient);  do\_exit:  if (NULL != msg\_buf) {  HAL\_Free(msg\_buf);  }  if (NULL != msg\_readbuf) {  HAL\_Free(msg\_readbuf);  }  return rc;  }  #endif /\* MQTT\_ID2\_AUTH\*/  int main(int argc, char \*\*argv)  {  IOT\_OpenLog("mqtt");  IOT\_SetLogLevel(IOT\_LOG\_DEBUG);  user\_argc = argc;  user\_argv = argv;  HAL\_SetProductKey(PRODUCT\_KEY); //⑩  HAL\_SetDeviceName(DEVICE\_NAME);  HAL\_SetDeviceSecret(DEVICE\_SECRET);  #ifndef MQTT\_ID2\_AUTH  mqtt\_client();  #else  mqtt\_client\_secure();  #endif  IOT\_DumpMemoryStats(IOT\_LOG\_DEBUG);  IOT\_CloseLog();  EXAMPLE\_TRACE("out of sample!");  return 0;  } |

文件中第⑨部分为具有安全加密的客户端资源函数，在mqtt\_client函数的基础上增加了消息加密传输的功能，其他部分与mqtt\_client函数功能一 致，此处不再进行分析。

最后我们分析一下main函数，main函数是系统运行的函数入口。系统完成相关的初始化工作进入main函数后首先调用系统函数IOT\_OpenLog设置日志模块。然后调用IOT\_SetLogLevel函数设置打印日志等级为IOT\_LOG\_DEBUG。接着将系统调用main时传入的参数保存于变量user\_argc和user\_argv中。

第⑩部分为硬件抽象层设置鉴权信息接口，将鉴权信息保存于内存中。接下来是一个预编译指令，根据参数设置情况运行mqtt\_client客户端或者运行mqtt\_client\_secure客户端。最后一部分是系统退出时调用的释放资源的函数。

在学习了mqtt-example文件的基础上，我们将mqtt-example.c代码中的设备的鉴权信息替换自己所创建的设备的三元组信息，即可得到用户应用程序代码。

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <stdarg.h>  #include "iot\_import.h"  #include "iot\_export.h"  #define PRODUCT\_KEY " a1aT1a7gnzN"  #define DEVICE\_NAME " test"  #define DEVICE\_SECRET "r0JpkrBZV1U89rGYTJtEJTLPeunOn7qp" |

**编译C SDK**

完成了应用代码的修改后，在进行编译之前我们还需要配置编译选项文件。系统根据编译选项文件中变量的值对不同的功能模块进行编译，生成可执行文件。编译选项文件位于~/iotkit-embedded下，文件名为make.setttings，文件中定义了控制功能模块是否参与编译的10个变量。我们根据应用中使用的模块对变量进行赋值，在这个应用中，我们使用了mqtt模块，对编译变量的配置为：

|  |
| --- |
| FEATURE\_MQTT\_COMM\_ENABLED = y  FEATURE\_MQTT\_DIRECT = y  FEATURE\_MQTT\_DIRECT\_NOTLS = n  FEATURE\_MQTT\_DIRECT\_NOITLS = y  FEATURE\_COAP\_COMM\_ENABLED = n  FEATURE\_HTTP\_COMM\_ENABLED = y  FEATURE\_SUBDEVICE\_ENABLED = n  FEATURE\_CMP\_ENABLED = y  FEATURE\_DM\_ENABLED = y  FEATURE\_SERVICE\_OTA\_ENABLED = y |

现在我们成功完成了应用程序的编写和编译选项的配置，接下来的工作就是对这个工程进行编译，得到我们所需的可执行文件。工程的编译是通过Linux下的make工具完成的。在上一节的准备工作中，我们已经安装了make工具，此时只需要一条简单的命令就可以完成工程的编译。但是，在编译之前，我们需要了解一下make工具的使用方法。make是Linux中的一个命令工具，负责执行我们编写的关于文件编译与链接的命令。通常，我们将需要编译与链接的文件按照make中的规则进行组织，然后调用gcc中的命令对文件进行编译与链接，生成可执行目标文件。这些关于make中所需的文件组织与编译命令都被存放于文件名为makefile的文件中。在我们运行make命令时，系统会找到该文件然后执行文件中存放的命令。现在我们运行cd ~/iotkit-embedded命令进入~/iotkit-embedded目录。然后运行ls命令查看该目录下的文件，发现SDK中已经有编写完成的makefile文件！接下来就可以开始工程的编译啦。

首先运行make distclean命令，执行distclean中的内容，清除先前编译产生的目标文件。接下来运行make命令，对工程进行编译。编译完成时系统会输出编译结果。

|  |
| --- |
| 30.96% [ dm ] 122448 Bytes  21.49% [ utils ] 84992 Bytes  14.36% [ mqtt ] 56784 Bytes  10.35% [ cmp ] 40944 Bytes  5.52% [ shadow ] 21848 Bytes  5.48% [ ota ] 21664 Bytes  4.85% [ system ] 19184 Bytes  2.67% [ http ] 10576 Bytes  1.74% [ cota ] 6872 Bytes  1.39% [ fota ] 5488 Bytes  1.19% [ log ] 4712 Bytes |

**运行调试**

**接入物联网平台并上报消息**

1、运行目标文件

make工具编译项目后，生成的目标文件保存于~/iotkit-embedded/output/release/bin目录中，接下来运行 cd ~/iotkit-embedded/output/release/bin进入该目录，用ls指令查看目录中的文件，发现mqtt-example目标文件就在该目录下。最后运行./mqtt-example loop 命令循环执行mqtt客户端程序。

2、设备上线

执行./mqtt-example loop命令后设备开始运行，开始日志打印系统初始化成功的状态信息。状态信息如下。

|  |
| --- |
| [inf] iotx\_device\_info\_init(40): device\_info created successfully!  [dbg] iotx\_device\_info\_set(50): start to set device info!  [dbg] iotx\_device\_info\_set(64): device\_info set successfully! |

系统创建好设备资源后开始连接物联网云平台进行设备鉴权。并输出连接成功信息，连接日志如下。

|  |
| --- |
| [inf] \_ssl\_client\_init(176): 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(144): crt content:451  [inf] \_ssl\_client\_init(184): ok (0 skipped)  [inf] \_TLSConnectNetwork(346): Connecting to /a1aT1a7gnzN.iot-as-mqtt.cn-shanghai.aliyuncs.com/1883...  [inf] mbedtls\_net\_connect\_timeout(291): setsockopt SO\_SNDTIMEO timeout: 10s  [inf] \_TLSConnectNetwork(359): ok  [inf] \_TLSConnectNetwork(364): . Setting up the SSL/TLS structure...  [inf] \_TLSConnectNetwork(374): ok  [inf] \_TLSConnectNetwork(409): Performing the SSL/TLS handshake...  [inf] \_TLSConnectNetwork(417): ok  [inf] \_TLSConnectNetwork(421): . Verifying peer X.509 certificate..  [inf] \_real\_confirm(93): certificate verification result: 0x00  [inf] iotx\_mc\_connect(2035): mqtt connect success! |

终端打印日志mqtt connect success!表示设备已经成功接入物联网云平台，此时我们在物联网云平台的设备管理中，设备test的状态从未激活变成在线。



3、设备上发消息

设备接入网络后开始向物联网云平台发送消息，设备发送的第一个消息为“update: hello! IoT!”。该消息被发送到update这个topic。在物联网云平台中进入*产品*管理页面，点击日志服务栏，查看设备上行消息。



**消息下发**

设备成功与物联网云平台连接后，可以进行消息的上发，也可以接收云平台下发的消息，实现设备与云平台的双向通信。接下来我们通过物联网云平台将消息发送到设备上。

在用户应用程序中，我们编写了设备订阅mqtt中data对应的topic。当服务端向该topic发布消息时，消息被推送至设备上。在物联网云平台上注册设备时系统默认创建了update，error与get三个topic，没有创建data对应的topic。那么就让我们登陆物联网云平台创建自己的topic吧。首先进入IoT控制台（https://iot.console.aliyun.com）。点击左侧导航栏中*产品*管理，查看Example*产品*。



点击查看进入*产品*页面，选择消息通信栏，然后点击定义topic类，进入topic设置页面。



进入topic设置页面后，网页弹出设置topic信息界面，输入topic类名为data,设置设备操作权限为发布和订阅。然后点击确认。此时topic创建完成。



现在我们已经完了所有下发消息需要的准备工作。接下来重新运行应用程序，设备上线后，登陆物联网云平台，进入设备管理页面，选择topic列表。在topic列表中第一个topic就是我们刚刚创建的data。点击该topic栏的发送消息，进入消息发送页面。



进入消息发送页面后，输入消息内容。然后点击确认按钮。网页显示消息成功发送。



此时，我们返回设备日志打印窗口，物联网云平台下发的消息被设备接收并成功解析，设备将解析后的消息“hello device!”打印在输出日志上。

|  |
| --- |
| \_demo\_message\_arrive|117 :: ----  \_demo\_message\_arrive|118 :: packetId: 0  \_demo\_message\_arrive|122 :: Topic: '/ a1aT1a7gnzN/test/data' (Length: 26)  \_demo\_message\_arrive|126 :: Payload: 'hello device!' (Length: 13)  \_demo\_message\_arrive|127 :: ---- |

**课程小结**

我们已经成功完成了第一个基于SDK开发的应用程序，应用在mqtt例程的基础上实现了设备与云平台的双向通信功能。当然这只是SDK功能的冰山一角，开发者在SDK的基础上可以设计出更高级的应用程序。