ALIBABA CLOUD

阿里云

物联网平台
常见问题

文档版本：20200821

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
`Courier字体`	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.物联网平台产品相关问题

本文介绍物联网平台产品相关常见问题。

什么硬件可以接入物联网平台？

理论上只要您的硬件环境有TCP/IP协议栈，就可以封装物联网提供设备连接和通信的SDK，通过SDK接入物联网平台。

物联网平台提供手机App相关的服务吗？

目前物联网平台只提供设备维度的服务，没有提供手机App相关服务，所以App需要您自己开发。

您可以使用物联网开发服务的移动应用开发工具开发手机App。

物联网平台和消息队列（RocketMQ）都是用于消息通信的产品，它们有什么区别？

物联网平台是专门针对物联网场景的设备通信而开发的产品；消息队列是针对服务端异步通信场景开发的。两者最大的区别在风险控制能力。

因为在物联网场景中，厂家生产的设备大部分都不是自己用，而是给用户使用，那就意味着设备被破解的概率大大增加。所以，风险控制至关重要。物联网平台具有设备认证环节。每个设备都会在物联网平台注册一个唯一的身份标识。在物联网平台上，您可以对每个设备进行管理，包括授权、禁用等管理。一旦某个设备被破解，就可以将风险控制到只损失单个设备，不会波及整个系统。而消息队列没有这样的风险控制机制。

如何获取物联网相关解决方案？

目前物联网平台的整套解决方案一般由第三方的解决方案商和模组商提供。阿里云云市场中有比较成熟的解决方案提供商。

是否支持C语言版的服务器端SDK？

如果服务器端使用C语言SDK接入物联网平台，调用物联网平台云端API，需要使用HTTP协议来发送请求数据。但是，我们不建议用C语言来开发服务器端，因为如果要使用消息回调，还需要用C语言的Web Sever，其开发难度较高。

是否可以把物联网平台接入域名隐藏在自有域名之后？

物联网平台不支持CNAME解析方式，因此不能用CNAME方式把物联网平台接入域名，如 `http://iot.cn-shanghai.aliyuncs.com` ，隐藏在您自有域名之后。物联网平台会对每个请求进行安全认证，如果不是来源于可信域名的请求可能会被拦截。

物联网平台SDK功能是全部免费的吗？

物联网平台提供设备端SDK和服务端SDK供您使用。物联网平台提供的开源SDK有多种证书，例如EDL Epl apache2.0等，请放心使用。

SDK是免费的。物联网平台收费是根据设备在线时长、设备消息数量和固件升级次数计算。目前物联网平台的收费标准，请参见计费方式。

2.通信相关问题

本文介绍设备与物联网平台和物联网平台与服务器端的通信相关问题和解决方法。

设备重复收到消息，如何去重？

由于QoS=1的机制是保证最少收到一次消息，所以可能会出现重复收到消息的情况，但重复消息的消息ID是相同的。设备端可以根据消息ID去重。物联网平台也会尽量减少QoS=1时的消息重复发送。

物联网平台是否会保存消息？

消息一发送到Topic后，物联网平台就会立即将消息转发给订阅了该Topic的设备。

QoS=0时，物联网平台不保存消息。

QoS=1时，消息会保存7天。

服务器端如何获取设备消息？

服务器端可通过以下两种方式获取设备消息。

- 服务端订阅：使用物联网平台的服务端订阅功能，订阅一个或多个消息类型。物联网平台根据您设置的订阅，将产品下所有设备的该类型消息流转至您的服务器。支持以下两种方式的服务端订阅。
  - 参见AMQP服务端订阅及相关文档，设置使用AMQP SDK接收物联网平台流转的设备消息。
  - 参见MNS服务端订阅，设置使用MNS SDK接收物联网平台流转到MNS队列的设备消息。
- 云产品流转：使用规则引擎的云产品流转功能，通过数据流转规则将指定设备数据流转到消息服务（MNS）的主题或消息队列（RocketMQ）的队列中。服务器通过MNS或RocketMQ SDK接收消息。具体设置操作，请参见云产品流转概述。

配置了MNS服务端订阅。但在控制台上发送消息，MNS的队列中并没有收到该消息，为什么？

在控制台发送的消息和调用云端API发送的消息均属于服务端发送的消息，不会流转到MNS队列，只有设备发出的消息（设备上报的消息、设备上下线的状态通知等）才会流转到MNS队列里。

如何判断MNS服务端收到的消息是来自哪个Topic？

发送到MNS队列中的消息格式：`{“messageid”:”12345”,“messagetype”:”status/upload”,“topic”:”null/topic”,“payload”:{data},“timestamp”:1469564576}` 中有Topic字段。您可根据Topic字段来判断。

3.Link SDK运行相关问题

本文介绍使用Link SDK过程中可能遇到的常见问题和解决方法。

Link SDK支持什么环境？

Link SDK是跨平台的，用户可以自行移植到目标平台上运行，开发环境推荐Ubuntu16.04。

Link SDK占用多少RAM？

MQTT协议数据传输通过mbedTLS，Link SDK消耗35 K（RAM）= 8 K（stack）+ 27 K（heap）。

CCP协议下，Link SDK消耗45 K（RAM）= 32 K（stack）+ 13 K（heap）。可以通过修改下面的两个宏减小stack的占用：

- `#define TOPIC_MAX_NUM 64`：Topic数量最大限制。如果设备订阅的Topic数量较小，可以修改为更小的值，如16。
- `#define TOPIC_MAX_LEN 128`：Topic长度最大限制。如果设备订阅的Topic名称长度较小，可以修改为更小的值，如32。

Link SDK的运行需要哪些条件？

运行Link SDK的主要条件是：支持TCP/IP协议栈。

是否支持FreeRTOS 操作系统移植？

支持FreeRTOS 操作系统连接阿里云物联网平台，例如乐鑫的Wi-Fi模组使用的就是FreeRTOS。请参见乐鑫提供的开源代码：FreeRTOS的移植参考代码

如何ECS上使用FreeRTOS 系统？

可以通过导入镜像实现。

导入镜像的方法，请参见导入镜像实践视频。

导入镜像注意事项，请参见导入镜像必读。

是否支持KEIL？

目前Link SDK不支持直接在Keil环境下开发，但是可以在SDK功能配置后，将抽取的代码添加到已有的Keil工程或者通过交叉编译生成目标库供Keil下的工程调用。

如何打开 SDK 日志？

在需要打开日志的地方，调用函数查看日志。具体函数说明如下。

- IOT_OpenLog：开始打印日志信息。可使用一个 `const char *` 作为入参，表示模块名称。
- IOT_SetLogLeve：设置打印的日志等级。入参从1到5。数字越大，日志越详细。
- IOT_CloseLog：停止打印日志信息。入参为空。
- IOT_DumpMemoryStats：打印内存的使用统计情况。入参从1到5。数字越大，日志越详细。

是否支持多线程？

目前 `IOT_*()` 的API都是进程级别，仅支持单进程单线程使用，不支持在同一进程的不同线程并发重入。

数据传输的安全性怎么保证？

设备和服务端之间的链路可以通过TLS加密，并且使用设备身份证书信息（productKey、deviceName、deviceSecret）进行认证，任何一个错误都会导致认证失败。

一个设备证书可用于多个设备接入吗？

不可以，一个设备证书只能用于一个设备连接。

报错“err log：[error]rate limiter”，请问是什么原因？

设备被限流，单个设备数据上报上限：QoS0为30条/秒，QoS1为10条/秒。限流后，设备上报的数据就会被丢掉。

如果订阅同一个广播Topic的设备数量超过1,000，怎么办？

广播Topic最多支持1,000个订阅者。如果设备数量超过1,000，可以对设备进行分组，每组设备数量等于或小于1,000。例如，有5000个设备，需分为5组，调用5次广播接口广播消息。

能不能订阅其他设备Topic？

设备只能订阅和发布自己的Topic。如果两个个设备之间需要通信，有两种方式：

基于规则引擎的M2M设备间通信

基于Topic消息路由的M2M设备间通信

广播消息Topic如何填写？

广播Topic格式：`/broadcast/productKey/xxxx`。

广播是否只能针对在线的设备？

广播Topic 默认是QoS=0，且不允许用户设置，因此广播消息只有当前在线设备才能接收到。

4.设备认证相关问题

本文介绍设备认证相关问题和原因。

多台设备是否可以使用相同的证书同时或不同时进行连接认证？

不支持。在同一时间内，只允许使用一个设备证书被一台设备用于连接物联网平台。如果多台设备上烧录相同的设备证书，连接时后连接的设备认证时，先连接的设备将会被断连。

设备端SDK是否支持MQTT协议的断线重连？

支持。

测试场景描述：开发板通过Wi-Fi连接路由器。把网线拔掉后，MQTT协议的设备端会自动尝试和服务器重新建立连接。重连尝试时间间隔是1s、2s、4s、8s、...。C SDK重试的最大间隔时间默认是60秒，即断网超过60s后仍未连接成功，设备端会每隔60秒尝试与服务器进行重连。您可以自定义最大间隔时间。

Android和Java SDK最大间隔时间为128秒，不支持自定义最大间隔时间。

5.MQTT连接相关问题

本文介绍MQTT连接可能出现的问题和解决方法。

云端接入域名和端口号是什么？

接入域名：

- 您购买的实例的接入域名，请在物联网平台控制台**实例管理**页面，单击实例对应的**查看**，进入**实例详情**页查看。
- 公共实例的接入域名：`${YourProductKey}.iot-as-mqtt.${YourRegionId}.aliyuncs.com` 。其中：
  - *${YourProductKey}*：请替换为设备所属产品的的ProductKey。可从物联网平台控制台**设备详情**页获取。
  - *${YourRegionId}*：请参见，替换为您的Region ID。

端口号：1883。

使用MQTT协议连接，不同的设备可以使用相同的clientID连接服务器吗？

clientID需为全局唯一。如果不同的设备使用相同的clientID同时连接物联网平台，那么先连接的那个设备会被强制断开。

MQTT协议版本是多少？

在*MQTT connect packet*中设置MQTT的版本。目前SDK（V2.02）使用MQTT 3.1.1。

可以修改SDK代码中 `src\mqtt\mqtt_client.h IOTX_MC_MQTT_VERSION` 的值，来修改支持的版本，3表示3.1版，4表示3.1.1版。

MQTT进行设备认证时，server返回“400”错误

认证返回400错误，表示鉴权认证失败。请检查设备证书信息ProductKey、DeviceName和DeviceSecret是否正确。

C语言SDK中MQTT是否支持iOS接入？

C语言SDK可以移植到任何能够支持C语言的系统上。如果是iOS系统建议寻找开源的Object-C实现。

目前mqtt-example设备上线后会立刻下线，请问如何修改mqtt-example让设备一直处于上线状态？

mqtt-example程序发送一次消息后会自动退出，可以尝试以下任意一种方式实现长期在线。

- 执行mqtt-example时，使用命令行 `./mqtt-example loop` ，设备会保持长期在线。
- 修改demo代码。mqtt-example的代码在最后会调用IOT_MQTT_Destroy，设备最后会变成离线状态，所以可以修改代码，去掉IOT_MQTT_Unregister 和IOT_MQTT_Destroy。

```
while(1)
{
IOT_MQTT_Yield(pclient, 200);
HAL_SleepMs(100);
}
```

心跳的时间间隔如何设置？

在IOT_MQTT_Construct里面可以设置keepalive_interval_ms的取值。物联网平台使用这个值来作为心跳间隔时间。keepalive_interval_ms的取值范围是60000~300000。

设备端的重连机制是什么？

设备端会在keepalive_interval_ms时间间隔发送ping request，然后等待ping response。

如果设备端在keepalive_interval_ms时间内无法收到ping response，或是在进行send以及recv时发生错误，平台就认为此时网络断开，而需要进行重连。

重连机制是平台内部触发，无需使用者接入。重连时，会重新进行认证。如果认证成功就会开始再次进行MQTT connect。重连会一直持续直到再次连接成功。

云端如何侦测到设备离线？

云端会根据MQTT CONNECT packet里面keepalive的设置，等待ping request。如果在指定时间内没有收到ping request，则认为设备离线。

云端可以接受的最大时延是5秒。

设备端SDK是否支持MQTT和CCP协议的断线重连？

支持。测试场景描述：开发板通过Wi-Fi连接上路由器后，把网线拔掉，MQTT和CCP协议都会自动尝试和server重新建立连接。尝试时间间隔是1s、2s、4s、8s、...，最大间隔时间默认是60s，也就是说断网后超过60s时间仍未连接成功，之后会每隔60s尝试和server重连。您可以设置最大间隔时间。

发布QoS1数据时，偶尔会出现MQTT_PUSH_TO_LIST_ERROR(-42)，如何解决？

需要等待ACK的packet都会存放起来，等待ACK。存放量有上限，当需要等待的packet太多到达上限时，就会触发 `MQTT_PUSH_TO_LIST_ERROR(-42) error` 。

出现错误可能是因为当前网络状态不好，或者是发送的频率过高。如果排除上述两个问题，当前的发送的频率是预期的，那么可以适当的调整IOTX_MC_REPUB_NUM_MAX、IOTX_MC_SUB_REQUEST_NUM_MAX和IOTX_MC_SUB_NUM_MAX的大小。

如果业务允许，也可以把发布的QoS调整成0。

IOT_MQTT_Yield的作用是什么？

IOT_MQTT_Yield的作用是尝试接收数据。因此在需要接收数据时，例如订阅和取消订阅之后，发布QoS1消息之后，或是希望收到发布的数据时，都需要主动调用该函数。

IOT_MQTT_Yield参数timeout的意义是什么？

IOT_MQTT_Yield会尝试接收数据，直到timeout时间到后才会退出。

IOT_MQTT_Yield与HAL_SleepMs的区别

IOT_MQTT_Yield与HAL_SleepMs都是阻塞一段时间，但是IOT_MQTT_Yield实质是去读取数据，而HAL_SleepMs则是系统什么也不做，等待timeout。

如何循环接收消息？

需要循环调用IOT_MQTT_Yield，函数内自动维持心跳和接收数据。

订阅了多个Topic，调用一次IOT_MQTT_Yield，能接收到多个Topic的消息吗？

首先需要确定Topic的权限，是不是同时满足发布和订阅。如果是，调用一次IOT_MQTT_Yield，可以接收到多个packet。

MQTT连接方式，只能通过不停地调用IOT_MQTT_Yield来轮询获取数据吗？

如果使用的TCP/IP协议栈，可以实现TCP主动通知上层有数据到达，可以改动实现事件触发的方式来触发IOT_MQTT_Yield。但是改动比较大，所以还请自行评估是否需要修改。

修改流程是：

调整utils_net.c里面socket的API，变成可以由TCP数据到达时回调的API。

当TCP主动通知上层有数据到达时，通知到MQTT服务器。让MQTT服务器内部执行IOT_MQTT_Yield，这样就可以不需要外部调用IOT_MQTT_Yield来读取数据。

如果TCP无法做到主动上报数据，但OS支持多线程，也可以在MQTT-example里面再起一个thread，在这个thread里面以下代码用于接收数据。收到数据时，触发主线程进行数据处理，而主线程大部分时间可以用于处理其他逻辑。

```
while(1)
{
IOT_MQTT_Yiled(pclient, 200);
HAL_SleepMs(200);
}
```

如果使用的系统也不支持多线程，就只能把IOT_MQTT_Yield的timeout时间间隔减小，然后提高调用的频率，在每次调用的时间间隔内执行其他操作，从而做到尽量减少对其他操作的阻塞。

什么情况下会发生订阅超时（subscribe timeout）？

在2倍request_timeout_ms时间内，系统未接收到SUBACK packet时，会触发订阅超时，并通过event_handle函数发送超时通知。

请在订阅之后，立刻执行IOT_MQTT_Yield尝试读取SUBACK，请勿使用HAL_SleepMs。

订阅时，返回IOTX_MQTT_EVENT_SUBCRIBE_NACK

请检查Topic的操作权限是否为订阅。

如果发布报错“no authorization”，请确认是否为发布权限。

MQTT发布的消息体大小限制

MQTT的协议包受限于IOT_MQTT_Construct里参数的write_buf和read_buf的大小。

MQTT协议包大小不能超过256 KB。超过大小限制的消息会被丢弃。

MQTT协议pub消息payload格式是怎么样的？

物联网平台没有制定pub消息payload的具体字段有那些。您根据应用场景制定自己的协议，然后以JSON格式放到pub消息载体里面传给服务端。

ota_mqtt升级的时候报错“mqtt read buffer is too short”

MQTT设置的buffer过小，即mqtt_param的pread_buf和pwrite_buf申请过小造成的。可以根据实际需要修改OTA_MQTT_MSGLEN的大小。

是否可以使用MQTT直连的方式进行OTA升级？

OTA升级时，必须使用HTTPS进行固件下载。MQTT只接收版本更新指令，与MQTT的连接方式无关。阿里云不支持HTTP下载固件，因此如果设备没有SSL通信的能力，则不能使用OTA服务。

打开MQTT over TLS，运行时提示MQTT创建失败，返回错误码0x2700

如果关闭MQTT over TLS则可以成功地订阅和发布信息；打开MQTT over TLS时，建连失败。首先确认mbedtls是否做了修改，这是用于传输层和应用层之间加密的功能，不能随意更改。mbedtls没有修改，则考虑系统时间是否正确，系统时间不对也会导致证书校验失败。

进行MQTT连接的时候，是否需要root.crt证书验证？

若使用TLS进行MQTT接入，需要下载根证书。

若使用物联网平台提供的demo进行开发，无需再下载根证书，demo中已自带证书。

物联网平台支持哪些QoS Level？

在MQTT协议、CCP协议下，阿里云物联网平台均支持QoS 0和QoS 1，但不支持QoS 2。

6.CoAP连接相关问题

本文介绍设备使用CoAP协议连接物联网平台可能出现的问题和解决方法。

CoAP协议允许 over UDP 接入吗？

目前物联网平台的CoAP连接只支持DTLS，不支持CoAP over UDP。

CoAP协议接入物联网平台的URI是什么？

在调用 IOT_CoAP_Init 的时候，可以设置其参数iotx_coap_config_t里面的p_url。

如果p_url为NULL，SDK会自动使用 `IOTX_ONLINE_DTLS_SERVER_URL` 这个URL。

```
#define IOTX_ONLINE_DTLS_SERVER_URL “coaps://%s.iot-as-coap.cn-shanghai.aliyuncs.com:5684”
```

CoAP协议接入物联网平台的地址、端口等详细信息，请参见CoAP连接通信。

IOT_CoAP_DeviceNameAuth认证函数是阻塞等待指定时间吗？

目前这个版本是非阻塞、异步的。在这个接口里面不会阻塞等待结果，而是在IOT_CoAP_Yield里面进行处理。

函数IOT_CoAP_Yield 是半阻塞。调用一次，会等2秒钟。有数据，则收取数据进行处理；没数据，则2秒后超时。

等待时间默认2,000毫秒，可自行修改宏COAP_WAIT_TIME_MS 来定义。

CoAP的客户端在提交认证时，需要字段IOTX_DEVICE_ID是什么意义？

这个字段是自定义的信息，例如可以填设备SN等信息。凡是想携带到云端的设备信息都可以填这里面。

如果发送到云端的数据不是JSON会出现什么错误？

目前除了支持JSON格式外，也可以支持cbor格式。与云端通信，需要使用指定格式，否则可能会出现无法解析的问题。

调用IOT_CoAP_Init方法时里面coap_malloc返回NULL，这是什么原因引起的？

请查看一下coap_malloc函数有没有适配实现。平台移植后，hal相关的底层函数都需要用户自行实现。

CoAP协议支持数据下行吗？

目前HTTP协议和CoAP协议暂不支持数据下行。MQTT协议支持数据上、下行，推荐使用。

7.HTTP连接相关问题

本文介绍设备使用HTTP协议连接物联网平台可能出现的问题和解决方法。

HTTPS进行设备认证时，server返回的错误码代表什么意思？

设备连接认证时，返回的错误码信息，请参见HTTP连接通信文档中的错误码说明。

HTTPS接入认证的时候，body中的sign参数怎么生成？

sign计算方法：`hmacmd5(deviceSecret,content)`。具体请参见HTTP连接通信。

服务器返回代码“302 Found，The requested resource resides temporarily under a different URI”

不能直接访问根目录路径的，要加上接入域名：`https://iot-as-http.cn-shanghai.aliyuncs.com/auth`

用浏览器调用HTTP请求出现跨域错误

目前阿里云不支持浏览器HTTP跨域接入。

HTTP协议支持数据下行吗？

目前HTTP协议和CoAP协议暂不支持数据下行。MQTT协议支持数据上、下行，推荐使用。

8.设备移植相关问题

本文介绍IoT_SDK V2.0在高通MDM9206平台上移植用户遇到过的问题原因说明。

编译问题

- ArmCC不支持C语言中变量定义和逻辑语句混杂，声明段落必须在逻辑段落之前完整结束。
- ArmCC不支持用花括号 `{}` 的方式定义匿名结构体实例作为右值。
- ArmCC处理枚举类型时，gcc及其变种按照int处理，而armcc有严格的编译时检查，多按unsigned int处理。
- ArmCC不支持在if、while、for逻辑的判断语句中有变量赋值。
- 函数内多分支情况，ArmCC对不可达分支处的return语句严格报错导致编译失败。
- ArmCC不支持返回值被声明为枚举类型的函数，实际上返回值是int等gcc所认为的等价数据类型。
- 标准库的头文件内容有差异。gcc编译时可以找到的定义如size_t、 typeof等，在armcc上没有。
- ArmCC不支持GNU风格扩展函数， 如vasnprintf、asprintf。
- ArmCC不支持GNU风格扩展数据结构，如匿名结构体、结构体匿名成员赋值等。

运行问题

- 使用标准库函数tolower()，需要对应引用头文件*ctype.h*，否则运行会出现卡死情况。
- 使用高通的串口打印函数，不能用 `%s` 格式打印0长度的字符串。
- HTTP Client的代码中，如 `src/utils/misc/utils_httpc.c` ，使用了C标准库函数sscanf()， 由于高通的C库问题，这些代码运行到时会卡死。

9.MQTT连接掉线相关问题

本文介绍MQTT连接掉线的排查方法。

为什么设备一直上下线？

如果发现设备反复上下线，很有可能同一个设备证书用于多个设备连接认证。例如有两个设备，但使用了同一个设备证书。设备1先上线，设备2后上线。设备2连接认证时，系统会把设备1的连接断掉，而设备1又会重连，再将设备2断掉，如此循环。可以通过日志服务，根据设备信息查询日志情况。

单设备每分钟最大连接请求次数为5次，超出5次的连接请求将被拒绝。

如何判断设备掉线的原因？

如果不是设备主动断开连接，可以通过设备日志来判断原因。

如果出现类似 `Keepalive timeout after xx sec` 这样的日志，说明设备没有及时发送心跳包给物联网平台。服务器容忍5秒的延迟，如果还是没有收到ping包，服务器会关闭与设备的连接。

如何检查网络问题？

如果您的本地环境无法连接服务器，请检查下本地网络情况。常用的网络检查命令：

- `ping ${productkey}.iot-as-mqtt.cn-shanghai.aliyuncs.com`：用于检测是否可以连接物联网平台服务器。
- `telnet ${productkey}.iot-as-mqtt.cn-shanghai.aliyuncs.com 1883`：用于检测1883端口连接情况。

如果以上的检测结果显示没有问题，但是接入仍然不成功，请检查本地防火墙策略。建议可通过traceroute、tcpdump分析网络具体原因。

另外，除了MQTT规范标准默认的1883端口，您也可以尝试使用443端口去连接MQTT server。在防火墙默认设置的情况下，443端口被拦截的几率小很多。

如果您仍不确定问题所在，可以提交工单，我们将协助您进行排查。

10.控制台使用问题

本文介绍物联网平台控制台使用相关问题。

如何查看线上日志？

在物联网平台控制台左侧导航栏，选择**监控运维 > 日志服务**，即可在**日志服务**下查看日志。

是否可以更换产品和设备的地域？

不支持变更地域。

如何激活设备？

只要使用该设备证书信息的设备成功连接物联网平台服务端后，该设备即被激活。

已有的三个自定义Topic类：get、update和error。这些Topic消息的来源分别是什么？

- error：设备发送到物联网平台的错误信息。
- get：设备从服务端订阅Topic消息。
- update：设备向物联网平台发送的消息。

您可以根据您的需要自定义Topic。请参见文档自定义Topic。

11.查询设备状态

11.1. 调用API获取设备状态

物联网很多业务场景中，时常需要获取设备的实时状态，以便根据不同状态（在线或离线）做不同处理。阿里云物联网平台提供多个云端API来获取设备的状态信息。本文介绍这些API的调用方法。

IoT 物联网 阿里云物联网平台 调用API接口 查询设备状态

原理

以下五个API可以获得设备状态。请根据业务需要，选择调用的接口。

API	描述	优缺点
GetDeviceStatus	获取单个设备的状态。	通过会话来获取设备状态，返回结果可能会因为网络和心跳包延迟而延时更新。
BatchGetDeviceState	批量获取多个设备的状态。	
QueryDeviceDetail	查询单个设备的详细信息 除设备状态外，还可以获得其他设备信息。	
BatchQueryDeviceDetail	批量查询多个设备的详细信息 除设备状态外，还可以获得其他设备信息。	
RRpc	向指定设备发送查询状态的请求消息，并同步返回响应。	该接口查询到的设备状态信息准确度高。 说明 本文示例中，只介绍调用RRpc查询设备状态的服务端SDK配置；更完整的设备状态查询配置方法，请参见服务端检测设备是否在线。

实现

本文示例使用Java SDK，需准备Java开发环境。

在Maven项目中，需添加如下pom依赖，安装阿里云IoT SDK。

```
<dependency>
<groupId>com.aliyun</groupId>
<artifactId>aliyun-java-sdk-core</artifactId>
<version>3.5.1</version>
</dependency>
<dependency>
<groupId>com.aliyun</groupId>
<artifactId>aliyun-java-sdk-iot</artifactId>
<version>6.11.0</version>
</dependency>
<dependency>
<groupId>commons-codec</groupId>
<artifactId>commons-codec</artifactId>
<version>1.13</version>
</dependency>
```

*Config.*参数值中，需传入您的阿里云账号AccessKey信息和设备信息。

```
// 地域ID，根据您的物联网平台服务地域获取对应ID，https://help.aliyun.com/document_detail/40654.html
private static String regionId = "cn-shanghai";
// 您的阿里云账号AccessKey ID
private static String accessKeyID = "Config.accessKey";
// 您的阿里云账号AccesseKey Secret
private static String accessKeySecret = "Config.accessKeySecret";
// 要查询的设备所属产品的ProductKey
private static String productKey = "Config.productKey";
// 要查询的设备的名称DeviceName
private static String deviceName = "Config.deviceName";
```

完整代码示例如下：

```
/*
* Copyright © 2019 Alibaba. All rights reserved.
*/
package com.aliyun.iot.demo.checkstatus;

import java.util.ArrayList;
import java.util.List;

import org.apache.commons.codec.binary.Base64;
```

```
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.exceptions.ServerException;
import com.aliyuncs.iot.model.v20180120.BatchGetDeviceStateRequest;
import com.aliyuncs.iot.model.v20180120.BatchGetDeviceStateResponse;
import com.aliyuncs.iot.model.v20180120.BatchGetDeviceStateResponse.DeviceStatus;
import com.aliyuncs.iot.model.v20180120.BatchQueryDeviceDetailRequest;
import com.aliyuncs.iot.model.v20180120.BatchQueryDeviceDetailResponse;
import com.aliyuncs.iot.model.v20180120.BatchQueryDeviceDetailResponse.DataItem;
import com.aliyuncs.iot.model.v20180120.GetDeviceStatusRequest;
import com.aliyuncs.iot.model.v20180120.GetDeviceStatusResponse;
import com.aliyuncs.iot.model.v20180120.QueryDeviceDetailRequest;
import com.aliyuncs.iot.model.v20180120.QueryDeviceDetailResponse;
import com.aliyuncs.iot.model.v20180120.RRpcRequest;
import com.aliyuncs.iot.model.v20180120.RRpcResponse;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.profile.IClientProfile;

public class GetDeviceStatusByApi {

// ====================需要用户填写的参数开始==========================
// 修改Config.*的参数为您的实际信息
// 地域ID，根据您的物联网平台服务地域获取对应ID，https://help.aliyun.com/document_detail/40654.html
private static String regionId = "cn-shanghai";
// 用户账号AccessKey ID
private static String accessKeyID = "Config.accessKey";
// 用户账号AccesseKey Secret
private static String accessKeySecret = "Config.accessKeySecret";
// 要查询的设备所属的产品ProductKey
private static String productKey = "Config.productKey";
// 要查询的设备名称deviceName
private static String deviceName = "Config.deviceName";
// ====================需要用户填写的参数结束==========================

private static DefaultAcsClient client = null;

private static DefaultAcsClient getClient(String accessKeyID, String accessKeySecret) {

if (client != null) {
return client;
}
```

```
try {
IClientProfile profile = DefaultProfile.getProfile(regionId, accessKeyID, accessKeySecret);
DefaultProfile.addEndpoint(regionId, regionId, "Iot", "iot." + regionId + ".aliyuncs.com");
client = new DefaultAcsClient(profile);
} catch (Exception e) {
System.out.println("create Open API Client failed !! exception:" + e.getMessage());
}

return client;
}

/**
* 设备状态获取
* 方法一、二、三、四是基于状态查询的，获取的状态值可能会因为网络和心跳包延迟而延时更新；
* 方法五是基于同步通信的，结果比较精准
*
* @param args
* @throws ServerException
* @throws ClientException
*/
public static void main(String[] args) throws ServerException, ClientException {

// 获取服务端请求客户端
DefaultAcsClient client = getClient(accessKeyID, accessKeySecret);

GetDeviceStatusByApi api = new GetDeviceStatusByApi();

// 方法一
api.ByGetDeviceStatus(client);

// 方法二
api.ByBatchGetDeviceState(client);

// 方法三
api.ByQueryDeviceDetail(client);

// 方法四
api.ByBatchQueryDeviceDetail(client);

// 方法五
```

```
// 方法五
api.ByRRpc(client);
}

/**
* 查询单设备运行状态
* GetDeviceStatus https://help.aliyun.com/document_detail/69617.html
*
* @param client 服务端请求客户端
* @throws ServerException
* @throws ClientException
*/
public void ByGetDeviceStatus(DefaultAcsClient client) throws ServerException, ClientException {

// 填充请求
GetDeviceStatusRequest request = new GetDeviceStatusRequest();
request.setProductKey(productKey); // 目标设备产品key
request.setDeviceName(deviceName); // 目标设备名

// 获取结果
GetDeviceStatusResponse response = (GetDeviceStatusResponse) client.getAcsResponse(request);
if (response != null && response.getSuccess()) {
GetDeviceStatusResponse.Data data = response.getData();
// ONLINE：设备在线。
// OFFLINE：设备离线。
// UNACTIVE：设备未激活。
// DISABLE：设备已禁用。
if ("ONLINE".equals(data.getStatus())) {
System.out.println("GetDeviceStatus 检测：" + deviceName + " 设备在线");
} else { // 其他状态归结为设备不在线，也可以根据业务情况自行区分处理
System.out.println("GetDeviceStatus 检测：" + deviceName + " 设备不在线");
}
} else {
System.out.println("GetDeviceStatus 检测：" + "接口调用不成功，可能设备 " + deviceName + " 不存在");
}
}

/**
* 批量查询设备运行状态
* BatchGetDeviceState https://help.aliyun.com/document_detail/69906.html
*
```

```
* @param client 服务端请求客户端
* @throws ServerException
* @throws ClientException
*/
public void ByBatchGetDeviceState(DefaultAcsClient client) throws ServerException, ClientException {

// 填充请求
BatchGetDeviceStateRequest request = new BatchGetDeviceStateRequest();
request.setProductKey(productKey); // 目标设备产品key
List<String> deviceNames = new ArrayList<String>(); // 目标设备名列表
deviceNames.add(deviceName);
request.setDeviceNames(deviceNames);

// 获取结果
BatchGetDeviceStateResponse response = (BatchGetDeviceStateResponse) client.getAcsResponse(re
quest);
if (response != null && response.getSuccess()) {
List<DeviceStatus> dsList = response.getDeviceStatusList();
for (DeviceStatus ds : dsList) {
// ONLINE：设备在线。
// OFFLINE：设备离线。
// UNACTIVE：设备未激活。
// DISABLE：设备已禁用。
if ("ONLINE".equals(ds.getStatus())) {
System.out.println("BatchGetDeviceState 检测：" + ds.getDeviceName() + " 设备在线");
} else { // 其他状态归结为设备不在线，也可以根据业务情况自行区分处理
System.out.println("BatchGetDeviceState 检测：" + ds.getDeviceName() + " 设备不在线");
}
}
} else {
System.out.println("BatchGetDeviceState 检测：" + "接口调用不成功，可能设备 " + deviceName + " 不存在")
;
}
}

/**
* 查询单设备详细信息
* QueryDeviceDetail https://help.aliyun.com/document_detail/69594.html
*
* @param client 服务端请求客户端
* @throws ServerException
```

```
* @throws ClientException
*/
public void ByQueryDeviceDetail(DefaultAcsClient client) throws ServerException, ClientException {

// 填充请求
QueryDeviceDetailRequest request = new QueryDeviceDetailRequest();
request.setProductKey(productKey); // 目标设备产品key
request.setDeviceName(deviceName); // 目标设备名

// 获取结果
QueryDeviceDetailResponse response = (QueryDeviceDetailResponse) client.getAcsResponse(request
);
if (response != null && response.getSuccess()) {
QueryDeviceDetailResponse.Data data = response.getData();
// ONLINE：设备在线。
// OFFLINE：设备离线。
// UNACTIVE：设备未激活。
// DISABLE：设备已禁用。
if ("ONLINE".equals(data.getStatus())) {
System.out.println("QueryDeviceDetail 检测：" + deviceName + " 设备在线");
} else { // 其他状态归结为设备不在线，也可以根据业务情况自行区分处理
System.out.println("QueryDeviceDetail 检测：" + deviceName + " 设备不在线");
}
} else {
System.out.println("QueryDeviceDetail 检测：" + "接口调用不成功，可能设备 " + deviceName + " 不存在");
}
}

/**
* 批量查询设备详细信息
* BatchQueryDeviceDetai https://help.aliyun.com/document_detail/123470.html
*
* @param client 服务端请求客户端
* @throws ServerException
* @throws ClientException
*/
public void ByBatchQueryDeviceDetail(DefaultAcsClient client) throws ServerException, ClientExceptio
n {

// 填充请求
```

```
BatchQueryDeviceDetailRequest request = new BatchQueryDeviceDetailRequest();
request.setProductKey(productKey); // 目标设备产品key
List<String> deviceNames = new ArrayList<String>(); // 目标设备名列表
deviceNames.add(deviceName);
request.setDeviceNames(deviceNames);

// 获取结果
BatchQueryDeviceDetailResponse response = (BatchQueryDeviceDetailResponse) client.getAcsRespon
se(request);
if (response != null && response.getSuccess()) {
List<DataItem> diList = response.getData();
for (DataItem di : diList) {
// ONLINE：设备在线。
// OFFLINE：设备离线。
// UNACTIVE：设备未激活。
// DISABLE：设备已禁用。
if ("ONLINE".equals(di.getStatus())) {
System.out.println("BatchQueryDeviceDetail 检测：" + di.getDeviceName() + " 设备在线");
} else { // 其他状态归结为设备不在线，也可以根据业务情况自行区分处理
System.out.println("BatchQueryDeviceDetail 检测：" + di.getDeviceName() + " 设备不在线");
}
}
} else {
System.out.println("BatchQueryDeviceDetail 检测：" + "接口调用不成功，可能设备 " + deviceName + " 不存
在");
}
}

/**
* RRPC 检测设备状态
* RRpc https://help.aliyun.com/document_detail/69797.html
*
* @param client 服务端请求客户端
* @throws ServerException
* @throws ClientException
*/
public void ByRRpc(DefaultAcsClient client) throws ServerException, ClientException {

// 填充请求
RRpcRequest request = new RRpcRequest();
request.setProductKey(productKey);// 目标设备产品key
```

```
request.setDeviceName(deviceName);// 目标设备名
request.setRequestBase64Byte(Base64.encodeBase64String("Hello World".getBytes())); // 消息内容，必须base64编码字符串
request.setTimeout(5000); // 响应超时设置，可根据实际需要填写数值

// 获取结果
RRpcResponse response = (RRpcResponse) client.getAcsResponse(request);
if (response != null) { // 不要使用response.getSuccess()判断，非SUCCESS都是false；直接使用RrpcCode判断即可
// UNKNOWN：系统异常
// SUCCESS：成功
// TIMEOUT：设备响应超时
// OFFLINE：设备离线
// HALFCONN：设备离线（设备连接断开，但是断开时间未超过一个心跳周期）
if ("SUCCESS".equals(response.getRrpcCode())) {
System.out.println("RRPC 检测：" + deviceName + " 设备在线");
} else if (response.getRrpcCode() == null) {
System.out.println("RRPC 检测：" + deviceName + " 设备可能不存在");
} else { // 其他状态归结为设备不在线，也可以根据业务情况自行区分处理
System.out.println("RRPC 检测：" + deviceName + " 设备不在线:");
}
// RRPC 还可以实现更复杂的设备状态检测方法
// 请参考 https://help.aliyun.com/document_detail/101133.html
} else {
System.out.println("RRPC 检测：" + "接口调用不成功");
}
}
}
```

11.2. 设备端检测自己是否在线

基于MQTT接入的设备靠心跳保活，但心跳是周期性的、且自动收发和超时重连，这些特性给主动检测设备端是否在线带来了一定难度。本文提供通过消息收发是否正常判定设备是否在线的原理、流程、实现方式。

原理

如果设备可以发送、接收消息，那么该设备的通信是没问题的，并且一定在线。

消息收发是物联网平台的核心能力。因此，这种判定方法不会因为物联网平台架构升级或业务变动而变化，也不会因为设备使用的客户端不同而不同。是设备端检测自己是否在线最通用的一种原理。

该原理的一种特殊实现就是设备端消息的自发自收。

流程

1. 创建Topic = `/yourProductKey/yourDeviceName/user/checkstatus` 。

   Topic可以自定义，但权限必须为发布和订阅。

2. 设备端订阅上一步创建的Topic 。

3. 设备端发送消息 `{"id":123,"version":"1.0","time":1234567890123}` ，请一定使用QoS=0 。

   消息内容可自定义，但建议使用此格式。

   参数说明：

字段	类型	说明
id	Object	用于验证收发的消息是否是同一个，请自行业务层保证唯一
version	String	版本号固定1.0
time	Long	发送消息的时间戳，可以计算消息来回的延时，评估当前的通信质量

4. 设备端收到消息上一步发送的消息。

   离线判定逻辑

   - 严格的：发送消息后，5秒内没有收到消息算失败，出现1次失败，判定为离线
   - 普通的：发送消息后，5秒内没有收到消息算失败，连续2次失败，判定为离线
   - 宽松的：发送消息后，5秒内没有收到消息算失败，连续3次失败，判定为离线

   > 说明 您可以根据自己的情况，自定义离线判定逻辑。

实现

为方便体验，本例基于Java SDK Demo开发，实现设备端检测自己是否在线的严格判定逻辑。

Java SDK开发具体细节，请查看相关文档。

> 说明 您可以根据自己的喜好，选择不同的设备端SDK进行开发。

首先，下载Demo工程，添加本类，并填写设备证书信息。设备端代码如下：

```
import java.io.UnsupportedEncodingException;


import com.aliyun.alink.dm.api.DeviceInfo;
import com.aliyun.alink.dm.api.InitResult;
import com.aliyun.alink.linkkit.api.ILinkKitConnectListener;
import com.aliyun.alink.linkkit.api.IoTMqttClientConfig;
import com.aliyun.alink.linkkit.api.LinkKit;
import com.aliyun.alink.linkkit.api.LinkKitInitParams;
import com.aliyun.alink.linksdk.cmp.connect.channel.MqttPublishRequest;
import com.aliyun.alink.linksdk.cmp.connect.channel.MqttSubscribeRequest;
import com.aliyun.alink.linksdk.cmp.core.base.AMessage;
```

```
import com.aliyun.alink.linksdk.cmp.core.base.ARequest;
import com.aliyun.alink.linksdk.cmp.core.base.AResponse;
import com.aliyun.alink.linksdk.cmp.core.base.ConnectState;
import com.aliyun.alink.linksdk.cmp.core.listener.IConnectNotifyListener;
import com.aliyun.alink.linksdk.cmp.core.listener.IConnectSendListener;
import com.aliyun.alink.linksdk.cmp.core.listener.IConnectSubscribeListener;
import com.aliyun.alink.linksdk.tools.AError;

public class CheckDeviceStatusOnDevice {

// ====================需要用户填写的参数，开始============================
// 产品productKey，设备证书参数之一
private static String productKey = "";
// 设备名字deviceName，设备证书参数之一
private static String deviceName = "";
// 设备密钥deviceSecret，设备证书参数之一
private static String deviceSecret = "";
// 消息通信的Topic，需要在控制台定义，权限必须为发布和订阅
private static String checkStatusTopic = "/" + productKey + "/" + deviceName + "/user/checkstatus";
// ====================需要用户填写的参数结束============================

// 接收到的消息
private static String subInfo = "";

public static void main(String[] args) throws InterruptedException {

CheckDeviceStatusOnDevice device = new CheckDeviceStatusOnDevice();

// 初始化
device.init(productKey, deviceName, deviceSecret);

// 下行数据监听
device.registerNotifyListener();

// 订阅Topic
device.subscribe(checkStatusTopic);

// 测试设备状态
System.out.println("we will check device online status now.");
device.checkStatus();
```

```
// 准备测试设备离线状态，请拔掉网线
System.out.println("pls close network，we will check device offline status after 60 seconds.");
for (int i = 0; i < 6; i++) {
Thread.sleep(10000);
}
device.checkStatus();
}

/**
* 测试设备状态
*
* @throws InterruptedException
*/
public void checkStatus() throws InterruptedException {

// ----------------------------------------------------------------------
// 要发送的消息，可以自定义，建议使用当前格式
// ----------------------------------------------------------------------
// Field | Tyep | Desc
// ----------------------------------------------------------------------
// id | Object | 用于验证收发的消息是否是同一个，请自行业务层保证唯一
// ----------------------------------------------------------------------
// version | String | 版本号固定1.0
// ----------------------------------------------------------------------
// time | Long | 发送消息的时间戳，可以计算消息来回的延时，评估当前的通信质量
// ----------------------------------------------------------------------
String payload = "{\"id\":123, \"version\":\"1.0\",\"time\":" + System.currentTimeMillis() + "}";

// 发送消息
publish(checkStatusTopic, payload);

// 严格的离线判定逻辑：发送消息后，5秒内没有收到消息算失败，出现1次失败，判定为离线
boolean isTimeout = true;
for (int i = 0; i < 5; i++) {
Thread.sleep(1000);
if (!subInfo.isEmpty()) {
isTimeout = false;
break;
}
}
```

```
if (!isTimeout && payload.equals(subInfo)) {
System.out.println("Device is online !!");
} else {
System.out.println("Device is offline !!");
}

// 置空接收到的消息，方便下一次测试
subInfo = "";
}

/**
* 初始化
*
* @param pk productKey
* @param dn devcieName
* @param ds deviceSecret
* @throws InterruptedException
*/
public void init(String pk, String dn, String ds) throws InterruptedException {

LinkKitInitParams params = new LinkKitInitParams();

// 设置 MQTT 初始化参数
IoTMqttClientConfig config = new IoTMqttClientConfig();
config.productKey = pk;
config.deviceName = dn;
config.deviceSecret = ds;
params.mqttClientConfig = config;

// 设置初始化设备证书信息，用户传入
DeviceInfo deviceInfo = new DeviceInfo();
deviceInfo.productKey = pk;
deviceInfo.deviceName = dn;
deviceInfo.deviceSecret = ds;

params.deviceInfo = deviceInfo;

LinkKit.getInstance().init(params, new ILinkKitConnectListener() {
@Override
public void onInitDone(InitResult initResult) {
System.out.println("init success !!");
```

```
}

@Override
public void onError(AError aError) {
System.out.println("init failed !! code=" + aError.getCode() + ",msg=" + aError.getMsg() + ",subCode="
+ aError.getSubCode() + ",subMsg=" + aError.getSubMsg());
}
});

// 确保初始化成功后才执行后面的步骤，可以根据实际情况适当延长这里的延时
Thread.sleep(2000);
}

/**
* 监听下行数据
*/
public void registerNotifyListener() {
LinkKit.getInstance().registerOnNotifyListener(new IConnectNotifyListener() {
@Override
public boolean shouldHandle(String connectId, String topic) {
// 只处理特定Topic的消息
if (checkStatusTopic.equals(topic)) {
return true;
} else {
return false;
}
}

@Override
public void onNotify(String connectId, String topic, AMessage aMessage) {
// 接收消息
try {
subInfo = new String((byte[]) aMessage.getData(), "UTF-8");
} catch (UnsupportedEncodingException e) {
e.printStackTrace();
}
}

@Override
public void onConnectStateChange(String connectId, ConnectState connectState) {
}
```

```
}
});
}

/**
* 发布消息
*
* @param topic 发送消息的Topic
* @param payload 发送的消息内容
*/
public void publish(String topic, String payload) {
MqttPublishRequest request = new MqttPublishRequest();
request.topic = topic;
request.payloadObj = payload;
request.qos = 0;
LinkKit.getInstance().getMqttClient().publish(request, new IConnectSendListener() {
@Override
public void onResponse(ARequest aRequest, AResponse aResponse) {
}

@Override
public void onFailure(ARequest aRequest, AError aError) {
}
});
}

/**
* 订阅消息
*
* @param topic 订阅消息的Topic
*/
public void subscribe(String topic) {
MqttSubscribeRequest request = new MqttSubscribeRequest();
request.topic = topic;
LinkKit.getInstance().getMqttClient().subscribe(request, new IConnectSubscribeListener() {
@Override
public void onSuccess() {
}

@Override
public void onFailure(AError aError) {
```

```
}
});
}

}
```

> **说明** 检测到设备离线后，尽量不要选择主动重连。
>
> 物联网平台规定一个设备1分钟只允许尝试接入平台5次，超过会触发限流，限制设备接入。此时停止接入，等待1分钟即可解除限制。
>
> 设备端应注意退避，切勿触发限流。

11.3. 服务端检测设备是否在线

基于MQTT接入的设备靠心跳保活，但心跳具有周期性、自动收发包和超时重连等特性，这些特性给主动检测设备端是否在线带来了一定难度。服务端虽提供了GetDeviceStatus和BatchGetDeviceState接口查询设备状态，但是调用API获取设备状态是基于会话实现的，会话保活是建立在心跳基础上的。本文提供通过使用RRPC来判定设备是否在线的原理、流程、实现方式。

原理

如果设备可以接收服务端下发的消息并作出响应，则该设备通信正常，设备一定在线。

消息收发是物联网平台的核心能力。因此，这种判定方法不会因为物联网平台架构升级或业务变动而变化，也不会因为设备使用的客户端不同而不同。这是服务端检测设备是否在线最通用的一种原理。

物联网平台提供的RRPC能力就是该原理的一种特殊实现。

流程

1. 设备端订阅RRPC请求的Topic是 `/sys/${yourProductKey}/${yourDeviceName}/rrpc/request/+` 。
2. 服务端调用RRpc接口发送指令，例如 `{"id":123,"version":"1.0","time":1234567890123}` 。

   消息内容可自定义，但建议使用此格式。

   参数说明如下表。

字段	类型	说明
id	Object	消息ID，用于验证收发的消息是否是同一条。请自行在业务层生成，并保证其唯一性。
version	String	版本号。目前版本号固定为1.0。
time	Long	发送消息的时间戳，可以用于计算消息来回的延时，评估当前的通信质量。

3. 设备端接收指令并响应RRPC请求。接收的消息格式： `{"id":123,"version":"1.0","time":1234567890123}`

   离线判定逻辑如下。

- 严格：发送消息后，5秒内没有收到消息算失败，出现1次失败，判定为离线。
- 普通：发送消息后，5秒内没有收到消息算失败，连续2次失败，判定为离线。
- 宽松：发送消息后，5秒内没有收到消息算失败，连续3次失败，判定为离线。

> **说明** 您可以根据自己的情况，自定义离线判定逻辑。

实现

为方便体验，本例基于设备端Java SDK Demo和服务端Java SDK Demo开发。

> **说明** 您可以根据实际情况，选择不同的设备端SDK和服务端SDK进行开发。

分别下载Demo工程，服务端添加CheckDeviceStatusOnServer类，设备端添加Device类，填写您的阿里云账号AccessKey信息和设备证书信息。

设备端代码如下：

```
import java.io.UnsupportedEncodingException;

import com.aliyun.alink.dm.api.DeviceInfo;
import com.aliyun.alink.dm.api.InitResult;
import com.aliyun.alink.linkkit.api.ILinkKitConnectListener;
import com.aliyun.alink.linkkit.api.IoTMqttClientConfig;
import com.aliyun.alink.linkkit.api.LinkKit;
import com.aliyun.alink.linkkit.api.LinkKitInitParams;
import com.aliyun.alink.linksdk.cmp.connect.channel.MqttPublishRequest;
import com.aliyun.alink.linksdk.cmp.connect.channel.MqttSubscribeRequest;
import com.aliyun.alink.linksdk.cmp.core.base.AMessage;
import com.aliyun.alink.linksdk.cmp.core.base.ARequest;
import com.aliyun.alink.linksdk.cmp.core.base.AResponse;
import com.aliyun.alink.linksdk.cmp.core.base.ConnectState;
import com.aliyun.alink.linksdk.cmp.core.listener.IConnectNotifyListener;
import com.aliyun.alink.linksdk.cmp.core.listener.IConnectSendListener;
import com.aliyun.alink.linksdk.cmp.core.listener.IConnectSubscribeListener;
import com.aliyun.alink.linksdk.tools.AError;

public class Device {

// ====================需要用户填写的参数，开始============================
// 产品productKey，设备证书参数之一
private static String productKey = "";
// 设备名字deviceName，设备证书参数之一
private static String deviceName = "";
// 设备密钥deviceSecret，设备证书参数之一
```

```
private static String deviceSecret = "";
// 消息通信的Topic，无需创建和定义，直接使用即可
private static String rrpcTopic = "/sys/" + productKey + "/" + deviceName + "/rrpc/request/+";
// ====================需要用户填写的参数，结束==========================

public static void main(String[] args) throws InterruptedException {

Device device = new Device();

// 初始化
device.init(productKey, deviceName, deviceSecret);

// 下行数据监听
device.registerNotifyListener();

// 订阅Topic
device.subscribe(rrpcTopic);
}

/**
* 初始化
*
* @param pk productKey
* @param dn devcieName
* @param ds deviceSecret
* @throws InterruptedException
*/
public void init(String pk, String dn, String ds) throws InterruptedException {

LinkKitInitParams params = new LinkKitInitParams();

// 设置 MQTT 初始化参数
IoTMqttClientConfig config = new IoTMqttClientConfig();
config.productKey = pk;
config.deviceName = dn;
config.deviceSecret = ds;
params.mqttClientConfig = config;

// 设置初始化设备证书信息，用户传入
DeviceInfo deviceInfo = new DeviceInfo();
deviceInfo.productKey = pk;
```

```
deviceInfo.productKey = pk;
deviceInfo.deviceName = dn;
deviceInfo.deviceSecret = ds;

params.deviceInfo = deviceInfo;

LinkKit.getInstance().init(params, new ILinkKitConnectListener() {
public void onError(AError aError) {
System.out.println("init failed !! code=" + aError.getCode() + ",msg=" + aError.getMsg() + ",subCode="
+ aError.getSubCode() + ",subMsg=" + aError.getSubMsg());
}

public void onInitDone(InitResult initResult) {
System.out.println("init success !!");
}
});

// 确保初始化成功后才执行后面的步骤，可以根据实际情况适当延长这里的延时
Thread.sleep(2000);
}

/**
* 监听下行数据
*/
public void registerNotifyListener() {
LinkKit.getInstance().registerOnNotifyListener(new IConnectNotifyListener() {
@Override
public boolean shouldHandle(String connectId, String topic) {
// 只处理特定Topic的消息
if (topic.contains("/rrpc/request/")) {
return true;
} else {
return false;
}
}

@Override
public void onNotify(String connectId, String topic, AMessage aMessage) {
// 接收RRPC请求并回复RRPC响应
try {
String response = topic.replace("/request/", "/response/");
```

```
publish(response, new String((byte[]) aMessage.getData(), "UTF-8"));
} catch (UnsupportedEncodingException e) {
e.printStackTrace();
}
}

@Override
public void onConnectStateChange(String connectId, ConnectState connectState) {
}
});
}

/**
* 发布消息
*
* @param topic 发送消息的Topic
* @param payload 发送的消息内容
*/
public void publish(String topic, String payload) {
MqttPublishRequest request = new MqttPublishRequest();
request.topic = topic;
request.payloadObj = payload;
request.qos = 0;
LinkKit.getInstance().getMqttClient().publish(request, new IConnectSendListener() {
@Override
public void onResponse(ARequest aRequest, AResponse aResponse) {
}

@Override
public void onFailure(ARequest aRequest, AError aError) {
}
});
}

/**
* 订阅消息
*
* @param topic 订阅消息的Topic
*/
public void subscribe(String topic) {
MqttSubscribeRequest request = new MqttSubscribeRequest();
```

```
request.topic = topic;
LinkKit.getInstance().getMqttClient().subscribe(request, new IConnectSubscribeListener() {
@Override
public void onSuccess() {
}

@Override
public void onFailure(AError aError) {
}
});
}

}
```

服务端代码如下：

```
import java.io.UnsupportedEncodingException;

import org.apache.commons.codec.binary.Base64;

import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.exceptions.ServerException;
import com.aliyuncs.iot.model.v20170420.RRpcRequest;
import com.aliyuncs.iot.model.v20170420.RRpcResponse;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.profile.IClientProfile;

public class CheckDeviceStatusOnServer extends BaseTest {

// ====================需要用户填写的参数，开始============================
// 用户账号AccessKey
private static String accessKeyID = "";
// 用户账号AccesseKeySecret
private static String accessKeySecret = "";
// 产品productKey，设备证书参数之一
private static String productKey = "";
// 设备名字deviceName，设备证书参数之一
private static String deviceName = "";
// ====================需要用户填写的参数，结束============================
```

```
public static void main(String[] args) throws ServerException, ClientException, UnsupportedEncodingEx
ception {

// ----------------------------------------------------------------------
// 要发送的消息，可以自定义，建议使用当前格式
// ----------------------------------------------------------------------
// Field | Tyep | Desc
// ----------------------------------------------------------------------
// id | Object | 用于验证收发的消息是否是同一个，请自行业务层保证唯一
// ----------------------------------------------------------------------
// version | String | 版本号固定1.0
// ----------------------------------------------------------------------
// time | Long | 发送消息的时间戳，可以计算消息来回的延时，评估当前的通信质量
// ----------------------------------------------------------------------
String payload = "{\"id\":123, \"version\":\"1.0\",\"time\":" + System.currentTimeMillis() + "}";

// 构建RRPC请求
RRpcRequest request = new RRpcRequest();
request.setProductKey(productKey);
request.setDeviceName(deviceName);
request.setRequestBase64Byte(Base64.encodeBase64String(payload.getBytes()));
request.setTimeout(5000);

// 获取服务端请求客户端
DefaultAcsClient client = getClient();

// 发起RRPC请求
RRpcResponse response = (RRpcResponse) client.getAcsResponse(request);

// RRPC响应处理
// 这个不能看response.getSuccess()，这个仅表明RRPC请求发送成功，不代表设备接收成功和响应成功
// 需要根据RrpcCode来判定，参考文档https://help.aliyun.com/document_detail/69797.html
if (response != null && "SUCCESS".equals(response.getRrpcCode())) {
if (payload.equals(new String(Base64.decodeBase64(response.getPayloadBase64Byte()), "UTF-8"))) {
System.out.println("Device is online");
} else {
System.out.println("Device is offline1");
}
} else {
System.out.println("Device is offline");
}
```

```
}
}

public static DefaultAcsClient getClient() {

DefaultAcsClient client = null;

try {
IClientProfile profile = DefaultProfile.getProfile("cn-shanghai", accessKeyID, accessKeySecret);
DefaultProfile.addEndpoint("cn-shanghai", "cn-shanghai", "Iot", "iot.cn-shanghai.aliyuncs.com");
client = new DefaultAcsClient(profile);
} catch (Exception e) {
System.out.println("init client failed !! exception:" + e.getMessage());
}

return client;
}
}
```

说明 需服务端主动触发RRPC调用，检测是否能及时收到设备端响应结果。