阿里云物联网套件

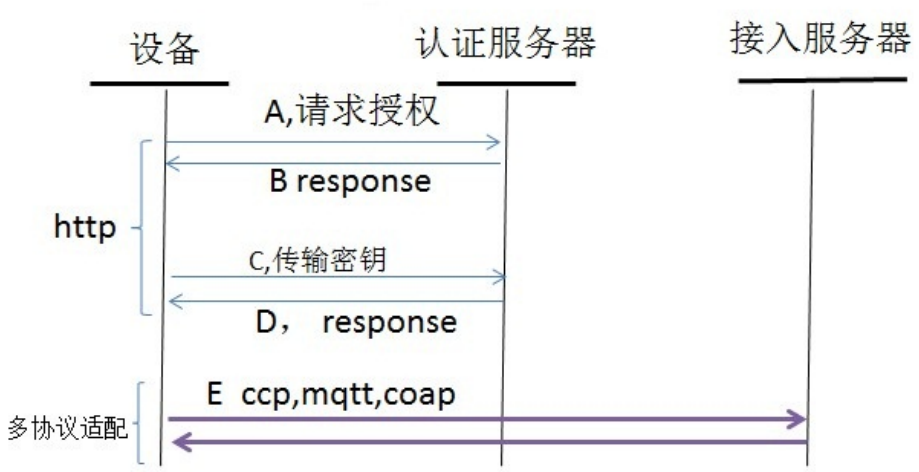
详细手册

阿里云
aliyun.com

详细手册

设备基于CCP接入

IOT接入手册



设备接入云端,主要包含两大流程:

- 设备的认证
- 设备接入数据通道

A，设备使用控制台申请的密钥进行签名请求授权,签名使用 hmacmd5算法(可选 hmacsha1),签名内容=devieceid+appkey,使用key=appSecret+deviceSecret

B，服务器验证签名合法,并返回pubkey证书

C，设备生成随机密钥seedkey,使用pubkey RSA加密传输给服务器

D，服务器确认返回令牌 sid ,同时返回签名sign,签名内容=sid,签名的key=appSecret+deviceSecret,同时返回动态负载的接入服务器ip供设备连接

E，客户端使用私钥seedkey解密令牌,进行数据通信,客户端可选协议模式支持ccp\mqtt\coap...

具体实现请参考设备认证和设备接入文档。

术语&约束

- deviceId : 平台颁发的设备唯一编号, 同deviceSn
- deviceSecret : 平台同时颁发的设备密钥, 和 deviceId 是成对出现
- appKey : 用户在平台当中申请的物联网应用 key
- appSecret : 用户在平台当中申请的物联网应用密钥, 和 appKey 也是成对出现
- seedkey : 设备端随机产生的一串32位长度的hexString，大写字母格式
- sid : 设备端和云端的连接sessionId
- 认证服务器地址 : http://manager.channel.aliyun.com 端口 : 80
- 接入服务器地址： 认证成功后动态返回ip与端口

设备认证流程

注意: 认证需要的参数可以在 iot.console.aliyun.com ，获取产品证书和设备证书 product appKey、product appSecret、deviceId 和 deviceSecret 详细请参考控制台使用手册中的创建产品和添加设备。

1 设备获取服务器的公钥证书以及接入服务器的ip地址等信息

第一步，设备授权认证。发送一个GET/POST请求到http://manager.channel.aliyun.com/iot/auth, 参数如下:

参数名	是否必传	描述
deviceId	必须	申请的deviceId, 例如 :b15f5e5063064fffaccc1e56ad59c29
appKey	必须	用户在阿里云IoT控制台上创建的应用的appKey, 例如: 12345
sign	必须	签名参数计算的规则如下: i. 将所有提交给服务器的参数（sign除外），按照字母顺序排序, 然后将参数值依次拼接起来. 例如 :appKey123456deviceIdb15f5e5063064fffaccc1e56

		ad59c29... 假定变量名content ii. 然后在对这个拼接之后的值(content)进行Hmac签名, 加密使用的key = appSecret+deviceSecret. iii. 得到结果转16进制字符串并转大写 例如 :DF2F6A209E06D25FA154DBE9771ED987 iv. 如果使用MD5，sign=md5(appSecret+content+deviceSecret)，最后转大写
hmac	可选	签名算法，HmacMD5（默认）或 HmacSHA1 或 MD5

- 服务器返回json结构:

```
{"servers":"8.8.8.8:8080|8001","pubkey":"pem key经过base64的字符串
"，"pkVersion":"1.0"，"success":true,"sign":"服务器签名"}
```

i. 获得到的公钥证书需要base64decode，结果是一个pem格式的字符串(x509证书),其中pkVersion是当前证书版本.
ii. 其中servers是用于ccp连接的数据通道服务器地址，端口有多个选择
iii. 至此, 设备端已经完成了认证和获取接入服务器的ip地址
iv. 服务器返回的sign使用以上一致的签名策略对servers、pubkey、pkVersion参数（success除外）签名，供客户端验证（防止dns劫持风险），证书和数据服务器ip可以缓存客户端。

2 确认会话的sessionId, 简称: sid

第二步，设备会话握手，使用公钥传输客户端私钥，同时返回会话id。 发送一个GET/POST请求到http://manager.channel.aliyun.com/iot/sid, 参数如下:

参数名	是否必传	描述

pkVersion	可选	证书版本（如果不传，则默认使用最新版本证书校验）
hmac	可选	签名算法，HmacMD5（默认）或 HmacSHA1 或 MD5
seedKey	必须	1.由客户端随机生成16位byte数组 2. 利用获取到的服务器公钥证书对这个进行RSA/ECB/PKCS1Padding 加密 3. 将加密之后的结果进行Base64 Encode . 具体可参考下方示例代码。
data	必须	1. json结构的字符串, 原始格式：**{deviceId:"xxxx", appKey:"xxxxx"}** 2. 组装好json格式之后做AES-128-ECB, PKCS5 加密, Aes的key使用seedKey二进制数组，最后进行 Base64 Encode. 3. 详情参考下方示例代码
sign	必须	对发送给服务器参数的签名值，规则参考第一步

注意: 最终发送HTTP请求前需要对所有url参数值做urlencode,utf8编码. base64结果通常含有=号之类的特殊字符

返回结果的处理:

```
{"sid":"/GuyJBskn9R1p9DRoyWfcgAjyfjYRMZm1EqDASVd/Uf5N6JOfZFYg2+juz8Wff+S","success":true,"sign":xxxx}
```

i. 利用base64 decode将sid转化成bytes
ii. 使用 AES-128-ECB, PKCS5 解密, Aes的key使用seedKey
iii. 最终得到解密后的 sid
iv. 解密后的sid用于接入服务器凭证进行数据传输，参考 设备接入流程。
v. sid和seedkey可缓存客户端本地，有效期24小时。

> 注意，服务器同时返回了sign签名供客户端验证，防止dns劫持。sign的内容只有sid一个字段，签名规则参考第一步。

错误码

- 当有异常时返回json错误格式：

```
{"errorCode":"InvalidSign","message":"illegle sign!","success":false}
```

  - 其中errorCode有： **InvalidSign**:签名错误，**CertExpired**：证书过期, **InvalidPara**:参数错误,**Reject**:非法请求,**Unknow**:系统异常

seedkey rsa算法的参考代码

```
 //java示例代码
byte[] seedkey = new byte[16];
new Random().nextBytes(seedkey);
Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");

CertificateFactory cf = CertificateFactory.getInstance("X.509");
X509Certificate cert = (X509Certificate) cf
.generateCertificate(new ByteArrayInputStream(pk.getBytes()));
PublicKey publicKey = cert.getPublicKey();

cipher.init(Cipher.ENCRYPT_MODE, publicKey);
//公钥加密
byte[] b1 = cipher.doFinal(seedkey);
String seedKey2server = base64(b1);
seedKey2server = UrlEncode.encode(seedKey2server,"utf-8");
//seedKey2server 就是最终发送服务器的值
```

- aes算法的参考代码

```
  String data = "{deviceId:\" xxxx\", appKey:\"12345 \"}";
SecretKeySpec aesKey = new SecretKeySpec(seedKey, "AES");
cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
cipher.init(Cipher.ENCRYPT_MODE, aesKey);
byte[] b2 = cipher.doFinal(date.getBytes("utf-8"));
//base64
String data2server = base64(b2);
data2server = UrlEncode.encode(data2server,"utf-8");
```

- 嵌入式c的加解密开源库可以参考 https://tls.mbed.org/source-code，后续我们会提供ccp c语言版sdk-source。

设备接入流程

1 ccp协议简介:

- 主要应用于 IoT 场景中 设备-云端 的通信, 以及设备-设备端 的通信.
- 安全, 稳定, 可靠.
- 基于TCP/IP实现, 服务器地址在第一阶段返回，包含主要协议如下:
    - CONNECT : 连接协议, 用于设备端发起和云端的连接协议.
    - RECONNECT : 快速重连协议, 网络异常场景下用于快速和云端发起重连协议.
    - RPC : 设备端向云端发送数据协议.
    - PUSH : 云端向设备短发送数据协议.
    - PING/PONG : 维护TCP连接的心跳协议.
    - Reverse-RPC : 由云端发起,但是期望设备端响应的数据交互协议.
    - PUB/SUB : 设备端的消息订阅和发布协议.
- 目前提供JAVA版demo，请参考快速开始章节

2 CCP协议:

基本协议格式:

固定头								
Bit	8	7	6	5	4	3	2	1
1 byte	MessageType					Compress	QosLevel	HasData
Remaining Length								
n bytes	可变长数值类型，根据可变长算法计算出Connect协议体的长度							
协议体								
n bytes	各协议协议体组装的内容							

MessageType:

MessageType		
协议名称	**值**	**描述**
CONNECT	19	连接云端协议
CONNECT_ACK	2	CONNECT 响应
PUSH	3	消息推送
PUSH_ACK	4	PUSH响应
PING	5	心跳包请求
PONG	6	心跳包响应
DISCONNECT	7	断开连接
RECONNECT	12	快速重连

RECONNECT_ACK	13	快速重连响应
UNKNOWN_SESSION	16	认证失败(会话超时, 或者sid失效)
RPCRequest	17	RPC请求
RPCResponse	18	RPC响应
Reverse_RPCRequest	23	服务端发起的rpc调用,期望客户端响应,有超时时间限制
Reverse_RPCResponse	24	客户端响应
PUBLISH	25	客户端发布消息
PUBLISH_ACK	26	客户端发布消息响应
SUBSCRIBE	27	客户端订阅消息
SUBSCRIBE_ACK	28	客户端订阅消息响应
UNSUBSCRIBE	29	客户端取消订阅
UNSUBSCRIBE_ACK	30	客户端订阅消息响应

字段类型:

可变长数值	这是一个1~n字节的数字,参考附录可变长算法。
字符串	由2部分组成 (2字节number+字符串bytes) 其中2字节无符号的数值,代表了字符串bytes长度。
byte	1字节的无符号位数值
byte[]数组	byte[]数组,n字节

Connect协议体:

Connect协议体			
名称	类型	值	描述
SequenceId	可变长数值	客户端生成	用于关联包相关性
Version	1byte数值	20	connect的版本号
platformId	可变长数值	2	*过期字段*
sid	字符串		设备认证阶段拿到的sessionId
AES加密段			
network	1byte数值	默认00000000	客户端网络信息. 高

			4位网络类型 0未知、2WIFI、3(2g)、4(3g)、5(4g)；低4位运营商类型 0未知、2移动、3联通、4电信.
appAccount	字符串	默认空字符串	过期字段
packageName	字符串	默认空字符串	过期字段
limit	可变长数值	默认50	离线消息最大限制数
keepalive	可变长数值	默认填空字符串	过期字段

CONNECTACK:

ConnectAck协议体			
名称	类型	值	描述
SequenceId	可变长数值	connect协议的sequenceId	用于关联包相关性
StatusCode	1byte数值		参见StatusCode定义
Connection Token	字符串	字符串类型	可以用这个Token在24小时以内做Reconnect
Suggestion IPs	字符串	字符串类型	下一次连接的推荐ip地址, 用于下一次连接就近网络, 返回空则不用处理
keepalive	可变长数值		保持连接的keepalive参数
lastestSDKPath	字符串	默认返回空字符串	过期字段
appId	可变长数值		返回推送系统的AppId

UNKNOWN_SESSION:

UNKNOWN_SESSION协议体			
名称	类型	值	描述
sourceMessageType	1byte数值		产生错误的来源消息类型
StatusCode	1byte数值		参见StatusCode定义

- *statusCode 定义:*

值	16进制	描述
0	0x00	连接成功
1	0x01	协议版本不正确
2	0x02	身份验证失败
3	0x03	证书已过期
4	0x04	无效的token
5	0x05	不安全的数据内容
6	0x06	Token过期
7	0x07	服务器内部错误
8	0x08	app认证packagename不正确 (已过期)
9	0x09	rpc调用出错
10	0x0A	不正确的sid

- *PUSH协议体:*

PUSH协议体			
名称	类型	值	描述
AES加密段			
SequenceId	可变长数值	服务端生成	用于关联包相关性
MessageId	可变长数值		消息的id,服务端可以根据消息id查询
AppId	可变长数值		推送系统的AppId
Content	byte[]数组		数据payload

PUSHACK:

PUSHACK协议体			
名称	类型	值	描述
AES加密段			
SequenceId	可变长数值	来自push协议的值	用于关联包相关性
MessageId	可变长数值		消息的id,服务端可以根据消息id查询
AppId	可变长数值		推送系统的AppId

type	1byte数值	3代表接受,4代表打开,8代表删除	告诉服务端消息已接受/已打开/已删除

PING:

ping只有一字节固定头, 如下:

固定头								
Bit	8	7	6	5	4	3	2	1
1 byte	0	0	1	0	1	1	1	0

PONG:

PONG也只有一字节固定头, 如下:

固定头								
Bit	8	7	6	5	4	3	2	1
1 byte	0	0	1	1	0	1	1	0

- ***DISCONNECT:***

DISCONNECT协议体			
名称	类型	值	描述
AES加密段			
connectionToken	字符串	CONNECTACK返回的token	关闭连接,设备下线

- ***RECONNECT:***

RECONNECT协议体			
名称	类型	值	描述
version	1byte数值	20	版本号
deviceId length	1byte数值		使用RSA加密deviceId得到的byte数组长度
deviceId	byte[]		使用RSA加密之后的deviceId,注意字符

			串需要使用utf8编码
ipswitch flag	1byte数值	默认0	0为没有切换, 1为连接的IP发生了切换
AES加密段			
SequenceId	可变长数值	客户端生成	用于关联包相关性
network	1byte数值	默认00000000	客户端网络信息. 高4位*网络类型 0未知*、*2WIFI*、*3(2g)*、*4(3g)*、*5(4g)*；*低4位*运营商类型 0未知、2移动、3联通、4电信.
ConnectionToken	字符串		与CONNECT协议体参数相同
limit	可变长数值		与CONNECT协议体参数相同
keepalive	可变长数值		与CONNECT协议体参数相同

- ***RECONNECTACK:***

RECONNECTACK协议体			
名称	类型	值	描述
AES加密段			
SequenceId	可变长数值	RECONNECT协议当中的sequenceId	用于关联包相关性
StatusCode	1byte数值		参见StatusCode定义
Connection Token	字符串	字符串类型	用于在24小时内做快速重连的Token
keepalive	可变长数值		保持连接的keepalive参数

RPCREQUEST:

RPC协议体			
名称	类型	值	描述
AES加密段			
SequenceId	可变长数值	客户端生成	用于关联包相关性
RPC Version	可变长数值	默认为2	rpc的版本号
Platform ID	可变长数值	默认为2	默认为2

sid	字符串		认证获得的sid
apiType	1byte数值	默认为3	默认为3
resourceType	1byte数值	默认为2	默认为2
resourceUrl	字符串	格式: appKey;appKey;1	服务的url
contentType	1byte数值	默认为2	默认为2
headers保留位	1byte数值	默认为0	默认为0
rpc content			
rpc content	byte[]数组	rpc payload	rpc 的payload, 透传到用户定义的数据上报地址服务中

- *RPCRESPONSE:*

RPC Response协议体			
名称	类型	值	描述
AES加密段			
SequenceId	可变长数值	RPC Request协议当中的sequenceId	用于关联包相关性
StatusCode	1byte数值		参见StatusCode定义
Response Payload 格式段			
ResponseStatus	1byte数值	0x00代表成功, 0x09代表调用失败	rpc服务的调用状态
如果服务调用成功,则解析下列段			
ContentType	1byte数值		RPC响应的ContentType
response header	1byte数值	默认返回0	RPC响应头
responseConfig 占位	1byte数值	默认返回0	
response payload	byte[]数组		服务的响应payload

- *RRPC:*

Reverse RPCRequest协议体			
名称	类型	值	描述

AES加密段			
SequenceId	可变长数值	服务端生成	用于关联包相关性
AppId	可变长数值		appId
payload length	可变长数值		payload长度
payload	byte[]数组		payload内容

- *RRPCRESPONSE:*

Reverse RPCResponse协议体			
名称	类型	值	描述
AES加密段			
SequenceId	可变长数值	从RRPC请求当中获得	用于关联包相关性
statusCode	1byte数值		参见statusCode
payload length	可变长数值		payload长度
payload	byte[]数组		response payload内容

- *PUBLISH:*

PUBLISH协议体			
名称	类型	值	描述
AES加密段			
SequenceId	可变长数值		用于关联包相关性
topic	字符串		向那个topic发送事件
aliveSecond	可变长数值		需要ACK的消息(至少发送一次的消息)存活的时间，如果该值为0，则说明不需要保存(至多发送一次的消息)
payload	byte[]数组		publish payload

- *PUBLISHACK:*

PUBLISHACK协议体			
名称	类型	值	描述
AES加密段			

SequenceId	可变长数值		用于关联包相关性
code	1byte数值		表明pub成功或失败,0表示成功,其他为失败,1为没有权限,其他未知的为8

- *SUBSCRIBE:*

SUBSCRIBE协议体			
名称	类型	值	描述
AES加密段			
SequenceId	可变长数值		用于关联包相关性
topic_size	1byte数值		topic数量,最少1个,最多64个
topic			
topic	字符串	string类型,注意MSB/LSB	topic名称,允许出现 * 或者 #

SUBSCRIBEACK:

SUBSCRIBEACK协议体			
名称	类型	值	描述
AES加密段			
SequenceId	可变长数值		用于关联包相关性
返回码, 至少1个,可能多个,每个code占1个字节			
codes length	1byte数字		所有codes的总长度
单个code	1byte数值		code
返回消息, 至少1个,可能多个,每个code占n个字节			
messages length	1byte数值		所有messages的总长度
单个message	字符串		message

UnSubscribe:

UnSubscribe协议体			
名称	类型	值	描述
AES加密段			
SequenceId	可变长数值		用于关联包相关

			性
topic,1至多个			
topic	字符串		topic名称,允许出现 * 或者 #

UNSUBSCRIBEACK:

SUBSCRIBEACK协议体			
名称	类型	值	描述
AES加密段			
SequenceId	可变长数值		用于关联包相关性
返回码, 至少1个,可能多个,每个code占1个字节			
codes length	1byte数值		所有codes的总长度
单个code	1byte数值		code

可变长算法

CCP协议中字段类型为可变长数值，都需要使用该算法来实现。

The algorithm for encoding a decimal number (X) into the variable length encoding scheme is as follows:

```
do
  digit = X MOD 128
  X = X DIV 128
  // if there are more digits to encode, set the top bit of this digit
  if ( X > 0 )
    digit = digit OR 0x80
  endif
  'output' digit
while ( X> 0 )
```

where MOD is the modulo operator (% in C), DIV is integer division (/ in C), and OR is bit-wise or (| in C). The algorithm for decoding the Number Length field is as follows:

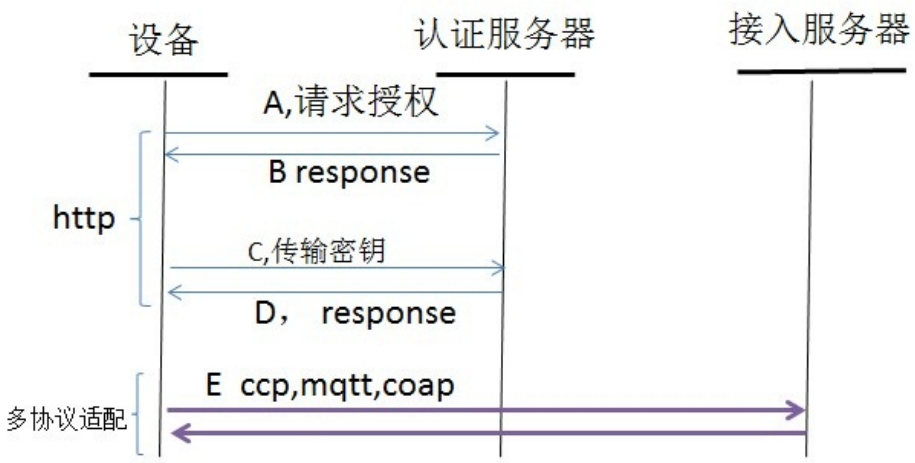
```
multiplier = 1
value = 0
do
  digit = 'next digit from stream'
```

```
    value += (digit AND 127) * multiplier
    multiplier *= 128
  while ((digit AND 128) != 0)
```

where AND is the bit-wise and operator (& in C).LS When this algorithm terminates, value contains the Remaining Length in bytes.

设备基于MQTT接入

IOT接入手册



设备使用mqtt接入云端,主要包含两大流程:

- 设备获取证书
- 设备接入数据通道

A，设备使用控制台申请的密钥进行签名请求授权,签名使用 hmacmd5算法(可选 hmacsha1、md5),签名内容=devieceid+appkey,使用key=appSecret+deviceSecret

B，服务器验证签名合法,并返回pubkey证书（X.509格式base64）

C，设备使用pubkey证书TLS协议连接MQTT

出于安全考虑算法不建议使用md5，具体实现请参考**设备认证**和**设备接入**文档。

术语&约束

- deviceId : 平台颁发的设备唯一编号, 同deviceSn
- deviceSecret : 平台同时颁发的设备密钥, 和 deviceId 是成对出现
- appKey : 用户在平台当中申请的物联网应用 key
- appSecret : 用户在平台当中申请的物联网应用密钥, 和 appKey 也是成对出现
- 认证服务器地址 : http://manager.channel.aliyun.com 端口 : 80
- 接入服务器地址： 认证成功后动态返回ip与端口

设备认证流程(MQTT)

注意: 用户在 iot.console.aliyun.com 申请并获取appKey、appSecret、deviceId 和 deviceSecret 详细请参考控制台使用手册中的创建产品和添加设备。

1 设备获取服务器的公钥证书以及接入服务器的ip地址等信息

第一步，设备授权认证。发送一个GET/POST请求到http://manager.channel.aliyun.com/iot/auth, 参数如下:

- deviceId : 申请的deviceId, 例如: b15f5e5063064fffaccc1e56ad59c29
- appKey : 用户在阿里云IoT控制台上创建的应用的appKey, 例如: 12345
- hmac: 签名算法，可选 HmacMD5（默认） 或 HmacSHA1 或 MD5，不传则使用默认值
- sign : 签名参数计算的方式如下:
    - 将上述所有的参数（sign除外），按照字母顺序排序, 然后将参数和值依次拼接起来. 例如:appKey123456deviceIdb15f5e5063064fffaccc1e56ad59c29...
    - 然后在对这个拼接之后的值(content)进行HmacMd5加签, 加密使用的key = appSecret+deviceSecret.
    - 得到结果转16进制字符串并转大写 例如 :DF2F6A209E06D25FA154DBE9771ED987
    - 如果使用MD5，sign=md5(appSecret+content+deviceSecret)，并转大写

服务器返回json结构:

```
 {"servers":"8.8.8.8:8080|8001","pubkey":"pem key经过base64的字符串
" , "pkVersion":"1.0" , "success":true,"sign":"服务器签名"}
```

- 获得到的公钥证书需要base64decode，结果是一个pem格式的字符串,其中pkVersion是当前证书版本.
- 其中servers是用于mqtt连接的数据通道服务器地址，端口有多个选择
- 至此, 设备端已经完成了认证和获取mqtt服务器的ip地址，下一步请参考设备接入
- 服务器返回的sign使用以上一致的签名策略对servers、pubkey、pkVersion参数（success除外）签名，供客户端验证（可以不做，防止dns劫持风险）

错误码

- 当有异常时返回json错误格式：

```
{"errorCode":"InvalidSign","message":"illegle sign!","success":false}
```

  - 其中errorCode有： **InvalidSign**:签名错误，**CertExpired**：证书过期,**InvalidPara**:参数错误,**Reject**:非法请求,**Unknow**:系统异常

设备接入流程

1. MQTT协议

目前阿里云支持设备通过MQTT协议接入(兼容 3.1 以及 3.1.1 版本协议). 具体的协议请参考 MQTT 3.1.1 或 MQTT 3.1 协议文档.

目前阿里云支持 TLSV1.1, TLSV1.2 版本的协议来建立安全连接

使用示例

Java示例

可以采用MQTT 推荐的客户端 Paho 来进行MQTT的连接. 本次示例采用其中的JAVA客户端来进行连接.

- 第一步: 首先通过设备认证，获取到对应的配置信息.
- 第二步: 通过获取到的公钥证书来配置SSL，与阿里云进行连接
- 第三步: 通过MQTT协议与阿里云进行设备通信.

需要注意的事项

在进行MQTT CONNECT协议设置的时候，

```
需要对ClientId以及UserName属性进行特殊的设置. 以便阿里云来验证您当前连接的身份信息.
对于ClientId的属性设置，需要将 AppKey+":"+DeviceId 设置为属性的值.
对于UserName属性， 则需要将 AppKey+AppSecret+DeviceId+DeviceSecret 连接之后，进行MD5加密，然后将加密后的值转换成大写.
另外，注意的是，Connect指令中的KeepAlive时间需要设置超过60秒以上，否则连接时，阿里云会拒绝该连接，返回ConAck错误码0x03.
```

具体DEMO连接 ： MQTT-JAVA-DEMO

请使用JDK7来跑DEMO程序. 运行DEMO之前，请首先修改 com.alibaba.iot.demo.util.Config 类，将对应的配置信息填写完整

具体代码如下:

获取证书，利用IotAuthUtil类来处理. 具体的流程参考 设备认证的流程.

```
    //Step 1: 获取配置信息
System.out.println("开始获取配置信息!");
Map<String, String> result = IotAuthUtil.auth();
//1.1 得到公钥的BASE64编码以后的字符串数据
String pubKey = result.get("pubkey");
//1.2 得到连接的目的地IP与端口
String servers = result.get("servers");
String targetServer = servers.substring(0, servers.indexOf("|"));

//1.3 得到BASE64字符串解码以后的公钥证书文件
byte[] pubKeyByteContent = Base64Util.decode(pubKey);
```

第二步，配置TLS/SSL信息

```
       InputStream is = new ByteArrayInputStream(pubKeyByteContent);
InputStream caInput = new BufferedInputStream(is);
CertificateFactory cf = CertificateFactory.getInstance("X.509");
Certificate ca = null;
try {
ca = cf.generateCertificate(caInput);
} catch (CertificateException e) {
e.printStackTrace();
} finally {
caInput.close();
}
String keyStoreType = KeyStore.getDefaultType();
KeyStore keyStore = KeyStore.getInstance(keyStoreType);
keyStore.load(null, null);
keyStore.setCertificateEntry("ca", ca);
String tmfAlgorithm = TrustManagerFactory.getDefaultAlgorithm();
TrustManagerFactory tmf = TrustManagerFactory.getInstance(tmfAlgorithm);
tmf.init(keyStore);
SSLContext context = SSLContext.getInstance("TLSV1.1");
context.init(null, tmf.getTrustManagers(), null);
SSLSocketFactory socketFactory = context.getSocketFactory();
```

第三步, 与阿里云建立 TLS/SSL 连接 .

需要注意的 在配置MQTT连接时. Connect协议当中 clientId 属性值为 appkey:deviceId username 属性值为 ToUpperCase(MD5_32(appkey+appSecret+deviceId+deviceSecret)) 这样以便阿里云对当前连接的有效性进行验证.

```
    final String topic = Config.topic;
String broker = "ssl://" + targetServer;
//客户端ID格式: appkey + deviceId.
String clientId = Config.appKey + ":" + Config.deviceId;
final MqttClient sampleClient = new MqttClient(broker, clientId, persistence);
MqttConnectOptions connOpts = new MqttConnectOptions();
connOpts.setMqttVersion(4);// MQTT 3.1.1
connOpts.setSocketFactory(socketFactory);
```

```
String signUserName = signUserName();
connOpts.setUserName(signUserName);
connOpts.setKeepAliveInterval(65);
System.out.println("进行连接, 目的地: " + broker);
sampleClient.connect(connOpts);
```

第四步, 进行消息Publish

```
    String content = "Message From DeviceId:" + Config.deviceId2;
MqttMessage message = new MqttMessage(content.getBytes());
message.setQos(1);
sampleClient.publish(topic, message);
System.out.println("消息发布成功!");
```

C客户端DEMO

请下载 Linux-C-MQTT-Demo,解压后，根据README文件来构建与部署即可.

FreeRTOS 版本

使用方式请解压后参见README文档.

下载链接:FreeRTOS DEMO

控制台使用手册

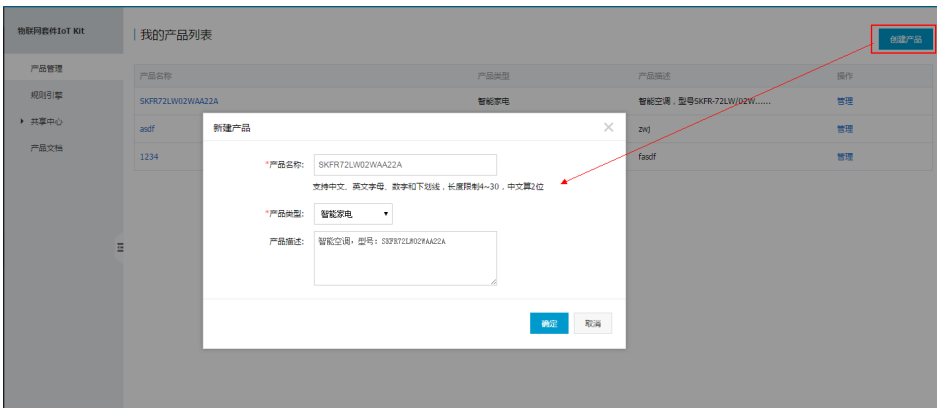
本文档主要介绍产品管理控制台的使用说明，帮助用户快速使用阿里云物联网套件。

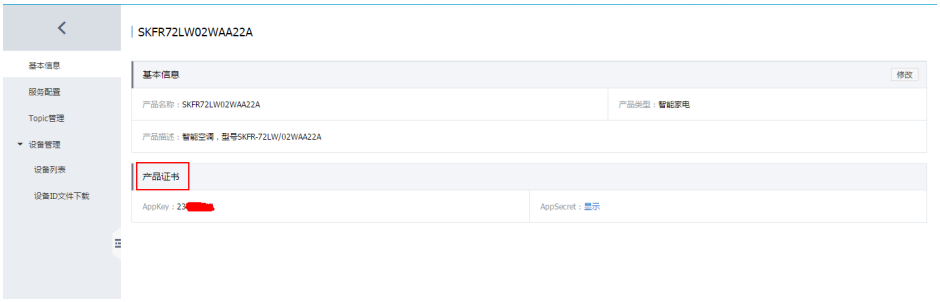
开通阿里云IoT

以aliyun账号直接进入IoT控制台，如果还没有开通阿里云物联网套件服务，则需要申请开通。

创建产品

初步进入控制台后，需要创建产品。点击创建产品。产品相当于某一类设备的集合，用户可以根据产品管理其设备等。

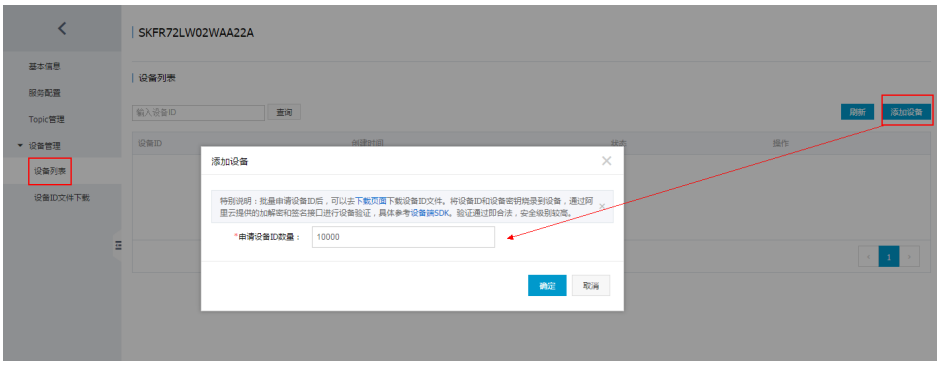


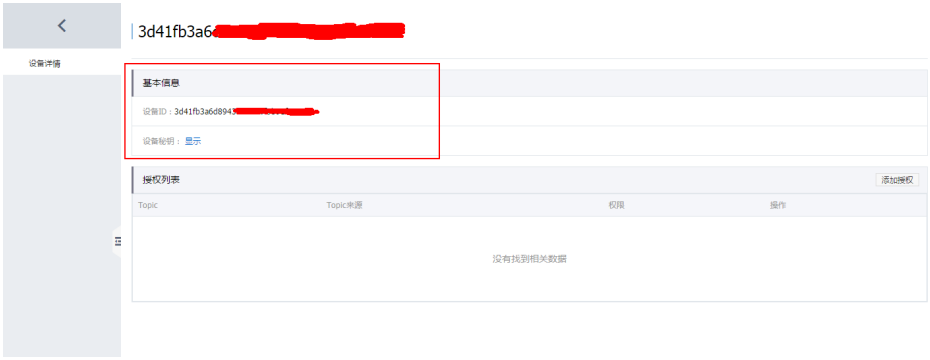


- 产品名称：对产品命名。产品名称在账号内保持唯一。
- 产品类型：选择设备对应的产品类型，方便阿里云IoT进行管理以及别人共享时搜索该产品。例如智能家电。
- **产品证书：创建产品之后，阿里云IoT会为产品颁发唯一标识符。**
    - AppKey：阿里云IoT为产品颁发的唯一标识符
    - APPSecret：阿里云IoT为产品颁发的产品秘钥，与AppKey成对出现。

添加设备

创建完产品之后，可以为该产品添加设备。进入产品管理页面下的设备列表，点击添加设备。





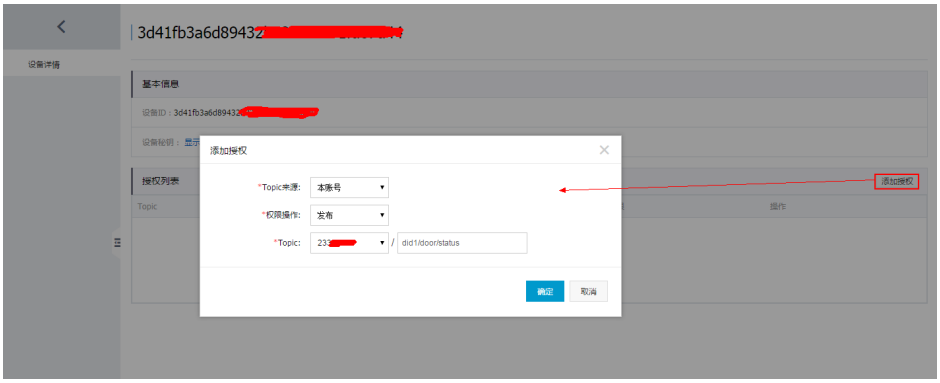
添加完设备之后，控制台向用户展示已经添加的设备，用户可以进入设备详情查看单个设备的信息。

- **设备证书：添加设备之后，阿里云IoT为设备颁发的唯一标识符**
  - 设备ID：即deviceId，是阿里云IoT给设备颁发的唯一标识符
  - 设备秘钥：即device_secret,是阿里云IoT为设备颁发的设备秘钥，用于认证加密，与设备ID成对出现。

设备授权

如果客户的应用场景适合用Pub/Sub通信方式，那是基于Topic进行通信。阿里云IoT规定**设备必须具备权限才能操作某个Topic**，具体详情请参考文档**身份和安全**的授权部分。

进入设备详情页，为设备添加操作Topic的权限。



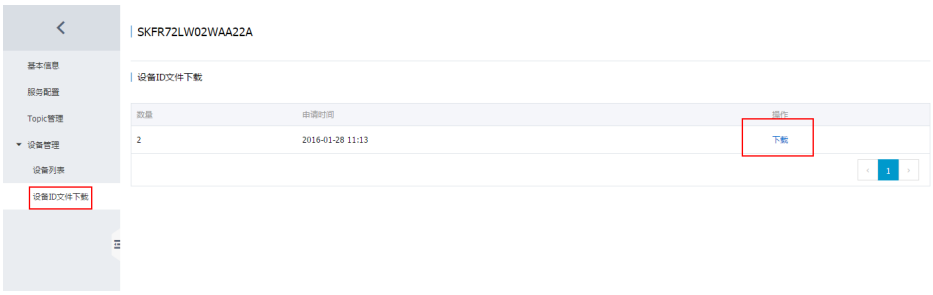
例子中，代表的就是该设备具有往Topic：23XXXXX/did1/door/status中发布消息的权限。

备注:

- Topic来源：操作的Topic属于哪个厂商；
- 权限操作：对Topic的操作，包括发布,订阅，以及发布和订阅。
- Topic：具体Topic定义，请参考文档Topic。**授权操作的Topic，可以使用通配符**

除了控制台添加授权，您也可以通过api接口动态授权，请参考api文档

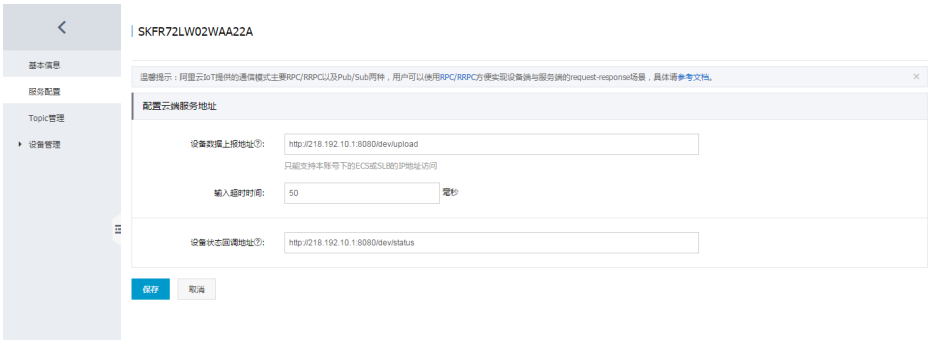
批量下载设备ID



特别注意：设备接入阿里云IoT必须携带产品证书和设备证书进行认证，用户应该保证证书的安全,不能泄露。具体请根据协议来选择参考设备接入文档

服务配置

阿里云物联网套件提供远程调用（RPC）的服务，方便设备端实时请求云端，实现设备端与云端request-response的场景。



特别注意：服务配置的所有功能目前只支持CCP协议接入的设备，MQTT协议接入的设备不能使用该功能

配置设备数据上报服务

- 设备数据上报地址：用户将部署的服务地址注册到阿里云IoT，然后设备调用CCP协议中RPC方法（详细请参考CCP协议中的RPC部分），这样设备数据就能通过数据通道转发到自己的服务地址。
- 超时时间：服务超时时间，范围在10-1000毫秒

配置设备状态回调地址

用户将设备状态回调地址注册到阿里云IoT,阿里云IoT会根据该地址主动将设备状态以HTTP方式通知用户，这样用户就能实时知道自己设备状态。具体的传输的数据格式请参考附录。

特别注意：设备数据上报地址目前只能支持本账号下的ECS或SLB的IP地址访问

云端实时请求设备端

云端实时请求设备端，并得到设备的返回结果，例如控制设备得到结果。这部分在控制台没有相应功能提供，但是阿里云物联网套件提供相应的API接口完成该功能。

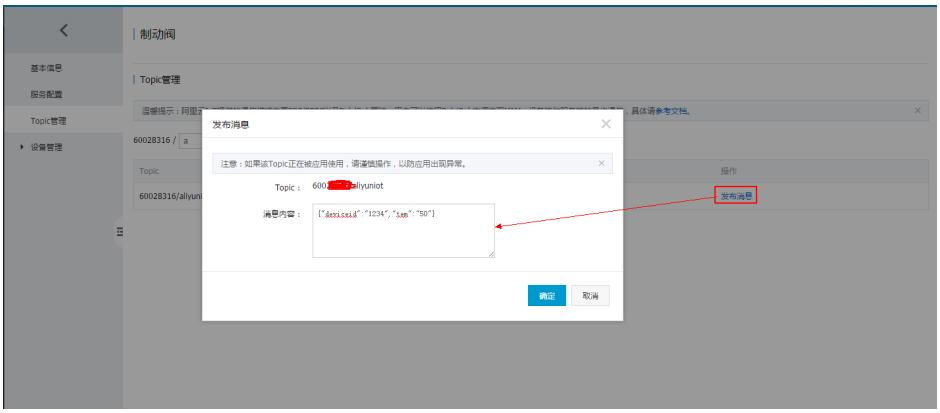
- 首先设备要基于CCP协议实现RRPC,具体请参考文档CCP协议中的RRPC部分
- 然后，服务端可以调用RRPC的API控制设备，参考文档RRPC。

Topic管理

CCP协议和MQTT协议都支持Pub/Sub的通信方式，Pub/Sub是基于Topic路由转发消息的。Topic是动态创建的，用户可以通过服务端调用OpenAPI或者设备端调用协议方法往某个Topic中发布消息，云端自动判断该Topic是否存在，不存在即创建。具体请参考Topic文档。

这里可以查询Topic，查询支持前缀匹配。

对查询出来的Topic，可以查看该Topic的消息发布数，也可以通过控制台往该Topic中发布消息。



特别提醒：消息内容将会转换成二进制分发给订阅者。所以如果发布的是JSON格式的消息，订阅者只能拿到转码过后的二进制数据，如果想要拿到JSON数据，需要再进行UTF-8转换成JSON。

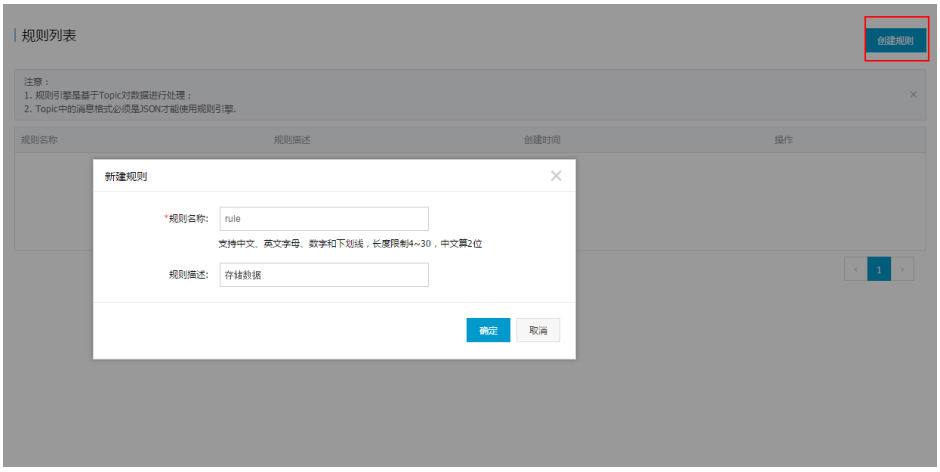
规则引擎使用手册

当用户是基于Topic进行通信时，则可以使用规则引擎对Topic中的数据进行处理，然后转发到阿里云其他服务上，例如可以转发到RDS、Table Store中进行存储，例如可以转发到推送服务将消息推送给手机App，例如可以转发到流式计算中进行实时计算，例如可以转发到消息中间件DataHub、LogHub上进而与流计算配合提供服务，等等。当然也可以使用规则引擎将Topic中的数据处理转发到另一个Topic中实现M2M通信。

特别注意：

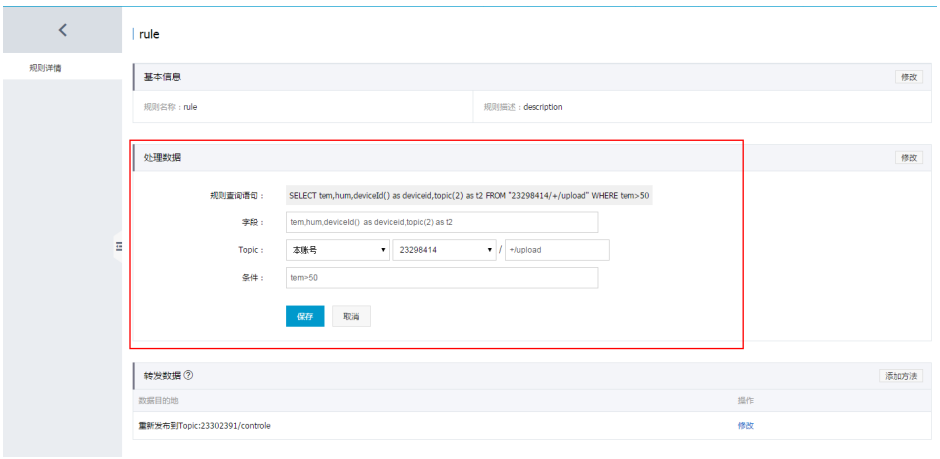
1. 规则引擎是基于Topic对数据进行处理
2. Topic中的消息格式必须是JSON才能使用规则引擎
3. 规则引擎是通过SQL对Topic中的数据进行处理

创建规则



处理数据

创建完规则，即可编写SQL对某一Topic中的数据进行处理，详细SQL规则请参考文档表达式



图中例子表达的规则：将key为tem和hum对应的数据从Topic:232*****/+/upload中的JSON消息中提取出来，而且利用规则引擎规定的函数deviceId()和topic()将特定的数据提取出来（规则引擎支持的函数详情请参考文档函数），再通过条件tem>50对数据进行过滤，最后得到处理过后的数据，以便进行下一步转发该数据。

特别注意：

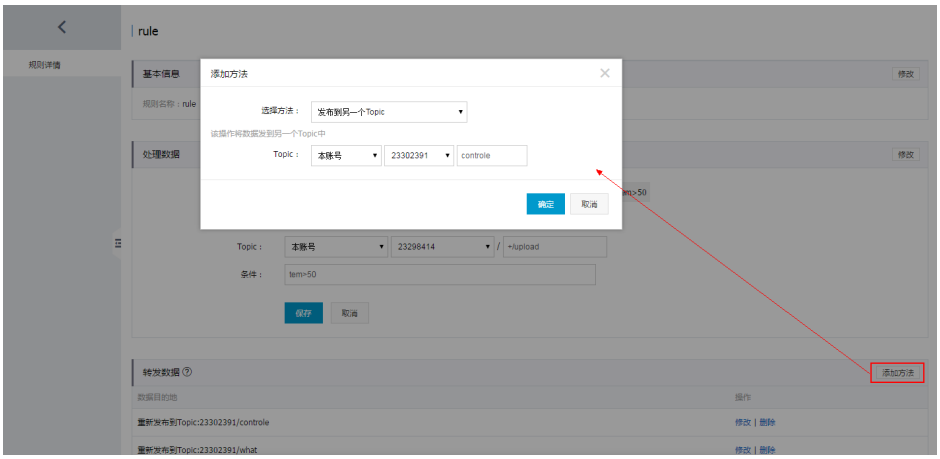
1. Topic支持通配符
2. 规则引擎中的SQL表达式支持标准SQL语法，而且会提供额外的函数方便实现丰富的场景。

转发数据

通过编写SQL对Topic中的消息进行处理，然后可以添加方法对处理过后的消息进行转发操作。下面将详细介绍目前已经上线的转发操作，而且我们的方式会不断增加，提供更多更完善的服务给用户。

发布消息到另一个Topic

将Topic中的消息处理过后，发送到另一个Topic，可以实现M2M场景，但是又不局限于M2M，帮助节省服务端的开发。

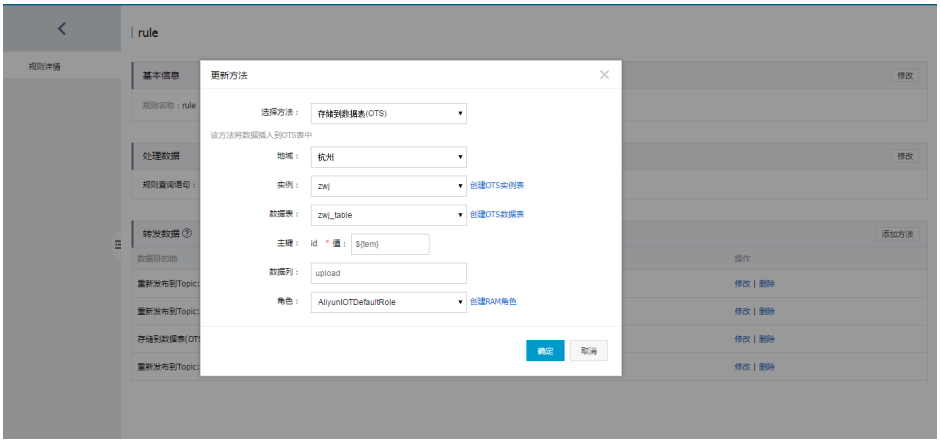


特别注意：

1. 重新发送的Topic不支持通配符, 但可以使用${}表达式引用上下文值，比如${a}将解析为 select xxx as a 对应的值。
2. 可以发送到其他厂商的Topic中

存储表格存储(Table Store)

可以将处理过后的消息，通过配置方法存储到表格存储（Table Store）中。想了解更多表格存储的信息，请参考表格存储（Table Store）

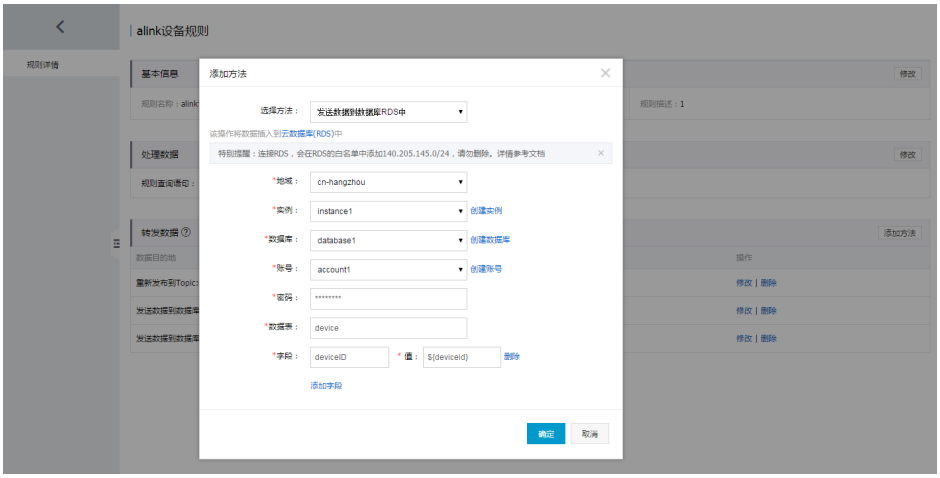


操作注意事项：

- 用户需要在控制台上选择Table Store数据表，用于数据存储。如果没有资源，则需要用户创建数据表
- 创建Table Store数据表必须创建主键，当用户确定数据表之后，需要输入该表主键的值，这里，我们提供了多种机制可供选择：
    - 用户可以输入常量。举个例子，主键值可以输入1，这就意味着当规则触发时，该主键就会存储值1
    - 用户可以输入已经利用SQL处理过后的Topic数据作为主键的值。举个例子，主键值支持转义符${tem}，这就意味着当规则触发时，该转义符就会替换成SQL中select属性中key为tem的值。${表达式} 中的表达式也支持对json的引用，比如tem.a。详情请参考表达式。
- 通过SQL处理过后的数据都会存储在Table Store数据表中**数据列**下，用户可以对数据列进行命名，默认是**payload**
- **阿里云IoT不能操作用户的Table Store数据表，必须经过用户的授权才能对用户的数据表进行写数据**。所以，用户需要创建一个具有Table Store写入权限的角色，然后将该角色赋予给阿里云IoT，这样阿里云IoT才能将处理过后的数据写入数据表中。

存储到云数据库（RDS）中

用户可以通过在控制台上配置方法将物联网套件中的数据路由转发到云数据库（RDS）中。想要了解更多RDS信息的，请参考文档云数据库（RDS）。



操作说明：

- 首先用户需要根据自己的业务选择数据库进行数据存储。用户需要先选择地域，然后根据地域选择实例，最后根据实例选择数据库。如果没有资源，那就需要去RDS控制台创建相应的资源。
- 选择完数据库之后，需要选择对该数据库**具有读写权限**的账号，如果没有，则需要去RDS控制台创建账号。输入该账号密码，让物联网套件去连接RDS进而写数据。
- 输入数据库中已经建立的数据表名，确定之后，数据将会写进这张表中。

确定数据表之后，需要将规则引擎筛选出来的数据对应存到数据表中的字段中。举个例子，将规则引擎筛选出来的JSON数据对应的存到RDS数据表中的字段中.

假如规则引擎的SQL：SELECT tem FROM mytopic. 假如RDS数据库有一张表，表中有tem字段，类型是String。 **存储可以在控制台上配置，字段填入的是RDS数据表中的字段，例如tem，值填入的是规则引擎筛选出来的JSON字段，例如${tem},这里要强调两点，需要使用${}，如果不使用的话，存到表中的将会是一个常量，例如填入tem，那数据表存入就是tem这个常量；字段与值的数据类型必须保持一致，不然无法存储。**

特别注意：

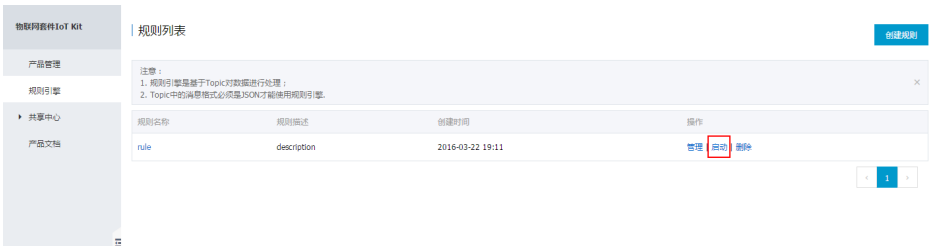
- 物联网套件为了连接RDS,会在RDS的白名单中添加140.205.145.0/24。这个IP不能删除，不然物联网套件就无法连接RDS，进而也就无法将数据写进RDS数据库中。

下图展示的就是RDS控制台的白名单，当用户使用规则引擎将数据写进RDS某个数据库实例时，就会出现在白名单中出现140.205.145.0/24这个IP。

- 物联网套件为了连接RDS数据库，需要使用用户的账号去连接，这就需要用户提供账号和密码给物联网套件。**这里面需要强调的是物联网套件取得用户的账号后，只是负责将规则匹配的数据写进数据库中，不会做其他操作。**
- 物联网套件只是负责将数据写进RDS数据表中，不会去帮用户去创建数据表，所以用户需要在RDS中自行创建数据表。
- **存储需要用户自行保证字段与值的数据类型一致，不然会导致写入不成功；值应该使用函数${}，这样才能存入JSON数据中的value，不然的话将会存入一个常量。**

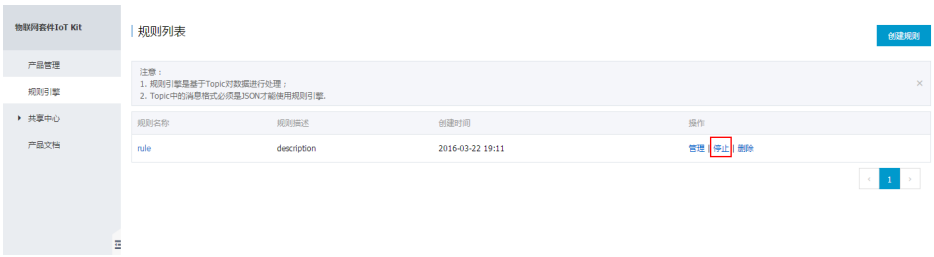
启动规则

当编写完SQL，添加好方法之后，用户就可以启动规则。这样一旦有消息Pub到SQL语法中的Topic里时，规则就触发，执行规则。



停止规则

用户也可以停止规则，这样规则就不会被触发。



本文档主要讲述使用共享中心的两种主要场景。用户可以基于共享中心实现设备跨厂商互联互通。**共享中心都是基于Topic进行共享的。**

场景一：设备数据共享

基于物联网套件可以实现设备数据共享，这样开发者可以开发出多样化的物联网应用。

假设厂商A生产智能门，厂商B生产智能灯，这两家厂商各自的云平台是相互独立的。现在厂商B想根据厂商A智能门的开关状态决定自己的开关状态。基于阿里云物联网套件如何实现呢？

首先厂商A将设备的消息发布到某个Topic中，然后厂商A将Topic的订阅权限共享给厂商B，厂商B获得权限就可以订阅设备的数据，进而作相应的业务处理。

第一步：发布消息到Topic

- 厂商A登录IoT控制台。
- 创建智能门的产品，在该产品下添加设备，为某设备添加授权，使设备具有往Topic：23XXXXX/door/status 发布消息的权限。具体操作请参考产品管理中的创建产品、添加设备和设备授权部分文档。
- 厂商A可以基于CCP协议或者MQTT协议将设备接入阿里云IoT，详情请参考文档设备基于CCP接入和设备基于MQTT接入
- 然后厂商A就可以基于CCP协议或者MQTT协议中的PUBLISH方法向Topic：23XXXXX/door/status发消息；当然也可以基于OpenAPI向Topic：23XXXXX/door/status发消息。

第二步：共享Topic的订阅权限操作

厂商A进入共享中心，将Topic：23XXXXXX/door/status的订阅权限共享。

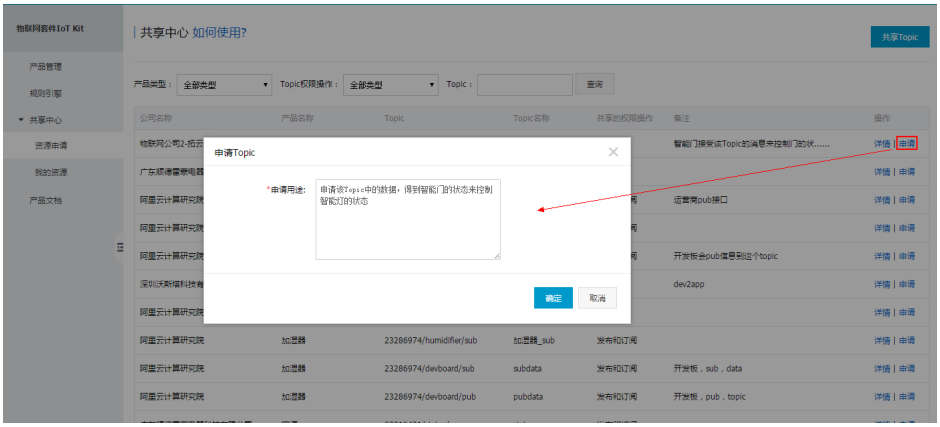
备注:

- 共享的权限操作：可以让申请者拥有该Topic何种权限操作，该例子是订阅。
- 消息格式：表示该Topic中消息的格式，这个对于申请者非常重要，因为他需要依赖这个来理解你的数据，然后进行数据的处理。

特别提醒：共享Topic必须是企业认证客户。

第三步：申请共享出来的Topic

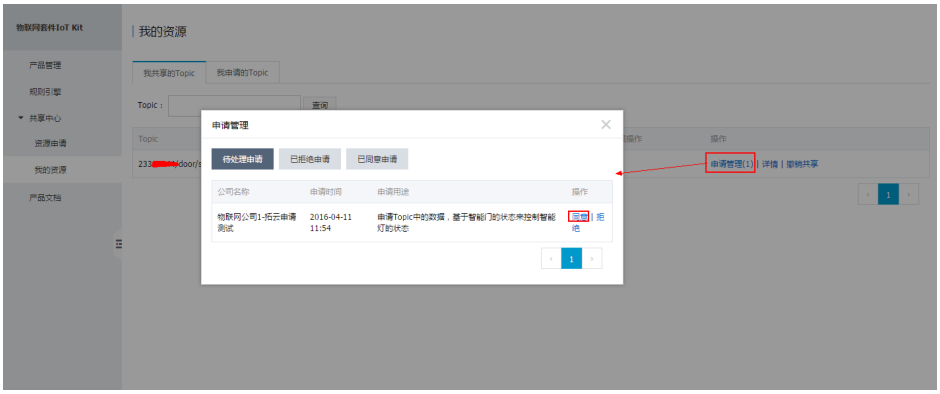
前面厂商A已经将Topic共享出来，现在厂商B可以登录IoT控制台进入共享中心查询该Topic进行申请。



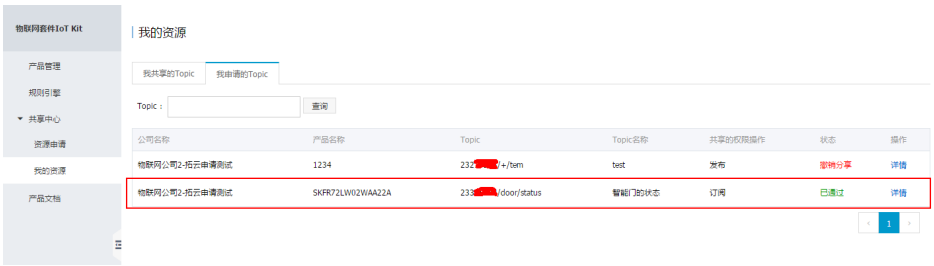
特别提醒：申请Topic必须也是企业认证客户。

第四步：处理申请Topic请求

当厂商B申请完Topic之后，厂商A需要对该请求进行处理。



如果同意，厂商B就拥有权限操作该Topic。



第五步：设备从Topic中订阅消息

厂商B有了权限之后，可以通过OpenAPI或者SDK订阅该Topic得到Topic中的消息，然后得到数据开发自己的应用程序控制灯的状态；或者可以登录控制台，直接将该Topic的订阅权限授权给某一个设备，这一步可以参考产品管理中设备授权文档，这样设备可以不经过服务端接受到数据，作相应的处理。

场景二：设备控制共享

基于物联网套件可以实现设备控制共享，这样开发者可以在经过其他厂商的同意的情况下控制其设备，进而开发出多样化的物联网应用。

假设厂商A生产智能灯，厂商B生产机器人，这两家厂商各自的云平台是相互独立的。现在智能机器人想根据主人的提示控制厂商A灯的开关。基于阿里云物联网套件如何实现呢？

首先厂商A需要订阅Topic中的消息，然后厂商A将该Topic的发布权限共享给厂商B，厂商B获得权限就可以发布消息到该Topic，最后厂商A订阅到消息进而作相应的业务处理。

第一步：从Topic中订阅消息

- 厂商A登录IoT控制台。
- 创建智能灯的产品，在该产品下添加设备，为某设备添加授权，使设备具有往Topic：23XXXXX/light/controle 订阅消息的权限。具体操作请参考产品管理中的创建产品、添加设备和设备授权部分文档。
- 厂商A可以基于CCP协议或者MQTT协议将设备接入阿里云IoT，详情请参考文档设备基于CCP接入和设备基于MQTT接入
- 然后厂商A就可以基于CCP协议或者MQTT协议中的SUBSCRIBE方法从Topic：23XXXXX/light/controle订阅消息；当然也可以基于OpenAPI向Topic：23XXXXX/door/status发消息。

备注:

- Topic来源：操作的Topic属于哪个厂商；
- 权限操作：对Topic的操作，包括发布,订阅，以及发布和订阅。
- Topic：具体Topic定义。具体参考文档Topic

第二步：共享Topic的发布权限操作

厂商A进入共享中心将Topic：23XXXXXX/light/controle的发布权限共享。

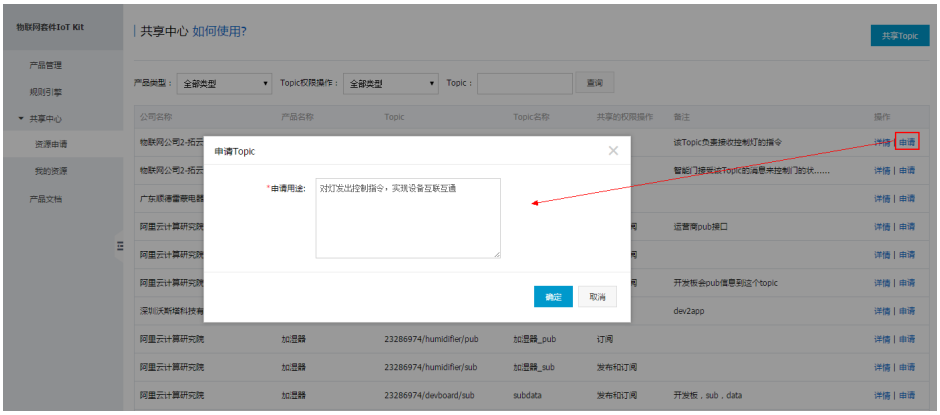
备注:

- 共享的权限操作：可以让申请者拥有该Topic何种权限操作，该例子是发布。
- 消息格式：表示该Topic中消息的格式，这个对于申请者非常重要，因为他需要依赖这个来理解你的数据，然后发相同格式的数据到Topic中。

特别提醒：共享Topic必须是企业认证客户。

第三步：申请共享出来的Topic

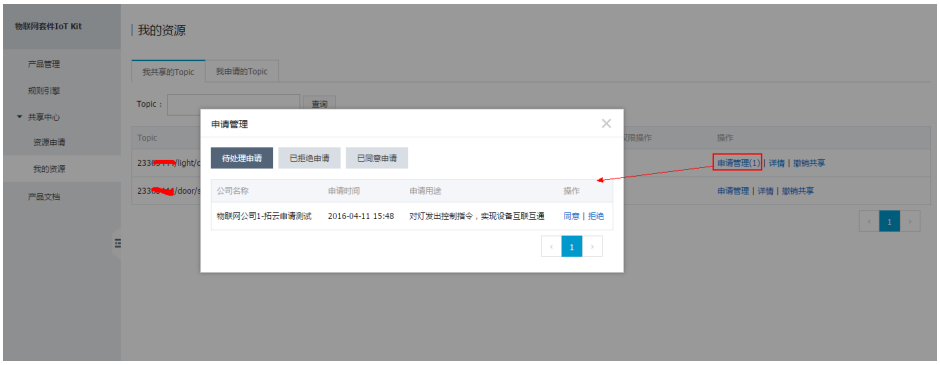
前面厂商A已经将Topic：23XXXXXX/light/controle共享出来，现在厂商B可以登录IoT控制台进入共享中心查询该Topic进行申请。



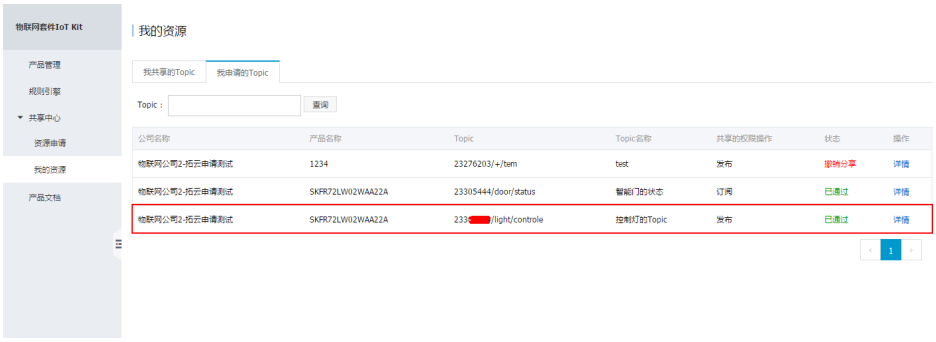
特别提醒：申请Topic必须也是企业认证客户。

第四步：处理申请Topic请求

当厂商B申请完Topic之后，厂商A需要对该请求进行处理。



如果同意，厂商B就拥有权限操作该Topic。



第五步：设备往Topic中发布消息

厂商B拥有该Topic的发布消息的权限之后，可以通过OpenAPI或者SDK发布消息到该Topic控制灯；或者可以登录控制台，直接将该Topic的发布权限授权给某一个设备，这一步可以参考**产品管理**中设备授权文档，这样设备可以不经过服务端发布消息到该Topic，然后被另一个厂商的设备订阅，实现跨厂商M2M的场景。

服务端收到设备回调数据格式

设备状态回调数据格式

若关注设备激活或者上下线状态的话，控制台提供状态回调。对应的回调数据格式如下:

- 设备激活数据格式

参数名为 data, 参数值为json格式，如下.

```
data=
{
    "status":"active",
    "appkey":"xxx",
    "appId":"12312", //推送时使用,可以忽略该参数
```

```
    "isDev":true,
    "deviceSn":"sss", //与设备ID相同.
    "deviceId":"deviceId",
    "ip":"deviceIp", //设备当前IP
    "time":"2015-12-23 00:00:00" //发送回调时间点.
}
```

- 设备上下线回调

参数名为 data, 参数值为json格式，如下.

```
data=
{
    "status":"online"(或offline),
    "appId":"12231",
    "deviceSn":"sss",
    "deviceId":"deviceId",
    "time":"2015-12-23 00:00:00"
}
```