阿里云物联网套件

产品组件

产品组件

IoT Hub

IoT Hub

IoT Hub帮助设备连接阿里云IoT，并进行安全可靠的数据通信。具有以下几大功能：

高性能的扩展能力

支持线性动态扩展，可以支撑十亿设备同时连接。

全链路加密

整个通信链路以RSA，AES加密，保证数据传输的安全。

支持设备多种协议接入

- 支持CoAP协议，CoAP接入文档请参考设备基于CoAP接入
- 支持开源的MQTT协议，MQTT接入文档请参考设备基于MQTT接入

消息实时到达

当设备与IoT Hub建立数据通道后，IoT Hub会与设备保持长连接，减少握手时间，保证消息的实时到达。

支持数据透传

IoT Hub支持二进制透传的方式将设备数据传到自己的服务器上，物联网套件不会保存设备业务数据，从而保证用户对数据的安全可控性。

支持多种通信模式

IoT Hub支持RRPC以及Pub/Sub两种通信模式，满足用户不同的应用场景。

IoT Hub支持多种设备协议

MQTT

MQTT协议被广泛用在嵌入式设备的消息传输上。详情请访问MQTT

COAP

CoAP是一种软件协议旨在用于非常简单的电子设备,让他们通过互联网交互通信。详情请访问COAP

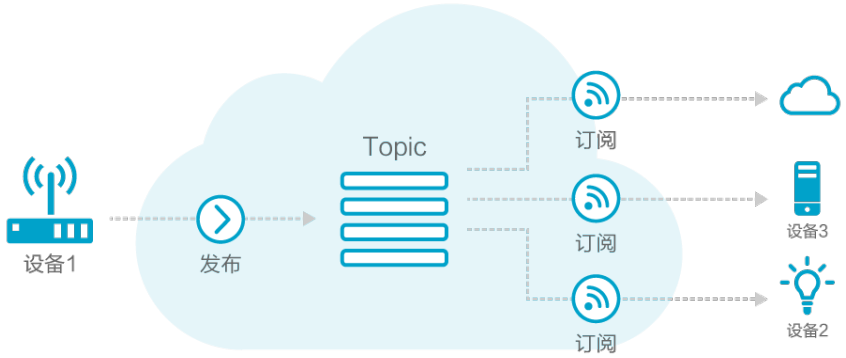
HTTPS

设备通过HTTPS协议连接IoT Hub。设备可以通过POST方式实现Pub消息到某个Topic。

IoT套件支持两种通信模式，PUB/SUB以及RRPC，用户可以根据自己的业务灵活使用两种通信模式。

PUB/SUB

PUB/SUB是基于Topic进行消息的路由转发，让设备端或者服务端可以发布订阅消息，实现异步的通信，参考下面的示意图。



IoT套件维护所有Topic的发布订阅用户列表，当消息发送到Topic，IoT套件会检查该Topic的所有订阅用户，然后将消息转发给所有订阅了该Topic的设备。

RRPC

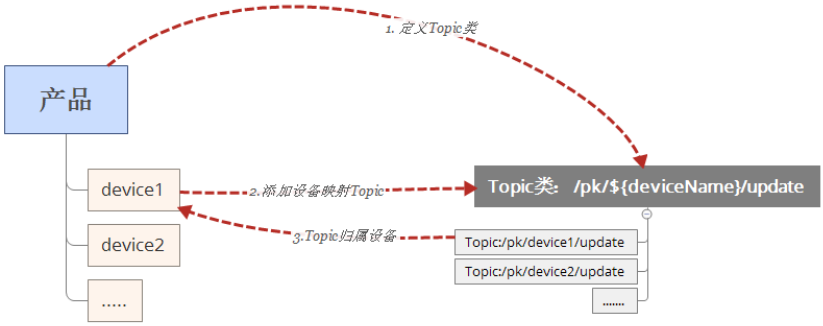
基于开源协议MQTT封装了同步的通信模式，服务端下发指令给设备可以同步得到设备端的响应。

MQTT协议和CoAP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。

讲解Topic之前，先介绍一下另一个概念：**Topic类**

Topic类

为了方便海量设备基于海量Topic进行Pub/Sub通信，简化授权操作，物联网套件增加Topic类的概念，用户只需在产品下定义Topic类，套件会自动将Topic映射到对应的设备上，不需要用户单独为每个设备授权Topic。



如下图:

Topic类的特性：

- Topic类是一类Topic的集合，举个例子，Topic类：/pk/${deviceName}/update是具体Topic：/pk/device1/update或者/pk/device2/update的集合。
- Topic类格式必须以“/”进行分层，区分每个类目。其中前两个类目已经规定好，第一个代表产品标识ProductKey，第二类目${deviceName}通配deviceName
- 其余类目命名只能包含字母，数字和下划线(_)命名每级类目，每级类目不能为空

注意：Topic类不能用于通信，pub/sub是基于具体的Topic通信。举个例子，用户不能使用/pk/${deviceName}/update进行通信，只能使用/pk/device1/update或者/pk/device2/update通信。

Topic

Topic的一些特性：

- Topic归属于相对应的设备，不能被其他设备用于pub/sub通信。举个例子，Topic：/pk/**device1**/update归属于设备**device1**，只能被device1用于发布订阅消息，而不能被设备device2用于发布订阅消息。
- **Topic是根据deviceName从Topic类中映射动态创建而来的，只有当deviceName创建存在时，对应的Topic才会创建，而且这个Topic只能被该设备通信使用。**
- Topic格式和Topic类一致，区别在于Topic类第二个类目是一个通配符${deviceName},而Topic则是具体的deviceName。

通配符

通配符	描述

#	这个通配符必须出现在最后一个类目，代表本级及下级所有类目，例如Topic为/productKey/#，这不仅可以代表/productKey/device1/update,也可以代表/productKey/device1/update/error
+	代表本级所有类目，例如Topic为/productKey/+/update，可以代表/productKey/device1/update，又可以代表/productKey/device2/update。**Topic类中${deviceName}就是+的一种用法，只不过规定通配的内容必须是deviceName**

- 同一个类目只能出现一个通配符，例如/productKey/##或者/productKey/++/tem都不合法。

注意：**通配符只能在配置规则引擎时使用**

安全是IoT最重要的话题。

阿里云物联网套件提供多重防护保障设备云端安全。

- 每个设备需要具备物联网套件颁发的证书才能连接IoT Hub；
- 设备与云端通信采用AES、RSA算法加密，从而保证数据传输安全;
- 为了安全地基于数据通道传输数据，设备有责任保证他们的证书安全;
- 数据到达IoT Hub之后,IoT Hub通过权限机制保障数据安全转发到其他阿里云服务或者其他设备。

设备身份

物联网套件为设备颁发证书，包括产品证书和设备证书，设备证书与设备是一对一的关系，确保设备的唯一合法性。设备通过CoAP、HTTPS、或者MQTT协议接入IoT Hub之前，都需要进行设备认证，设备认证需要携带产品证书和设备证书进行认证。

通信安全

阿里云物联网套件采用RSA-1024bit、AES算法来保证数据传输安全。在认证过程中您需要使用阿里云颁发的根证书 CA_pub 文件，点击下载

数据路由安全

数据从设备经过物联网套件再路由到阿里云的任何一个节点，都有权限机制保证其安全。

- 设备基于Topic向套件发布或者订阅消息，前提是必须具有Topic的权限；
- 设备数据从物联网套件路由到阿里云其他节点，例如数据库，队列等等，都需要对应的权限，不然无法写入。

Topic的权限操作

对Topic的权限操作，阿里云物联网套件提供的有以下几种：

操作	描述
发布	该操作表示可以往Topic中发布消息
订阅	该操作表示可以从Topic中订阅消息
发布和订阅	该操作表示既可以对某Topic发布消息，也可以对某Topic订阅消息

操作的限制

阿里云物联网套件不仅能够支持设备端发布订阅消息，同时也支持服务端发布消息给设备。但是操作也有一些限制：

- 设备只能对其Topic列表中的Topic进行发布订阅消息；
- 服务端基于AK只能对该账号下的Topic和设备发送消息；
- 服务端不能对带有通配符的Topic进行发布消息。

规则引擎

本文档主要描述处理数据中SQL表达式。

为了容易理解，我们把一条规则抽象为一条sql表达(类试mysql语法)：

例子：SELECT crypto(userId,’SHA1’) md2, (a+1) al, color c, config.flag flag, deviceId(),CASE col.a WHEN 1 THEN ‘Y’ ELSE ‘N’ END flagFROM “/12345/#” WHEREc is not null and b<0

select参数和where条件可以使用消息的payload属性作为列，不支持子查询。

1，FROM “topic” 当有符合topic规则的消息到达时，消息的payload数据以json形式被上下文环境使用(如果消息格式不合法,将忽略此消息),您可以使用topic()函数引用具体的topic值。

注意，如果您选的topic来自别人共享的，则配置的topic需要与提供的topic模式匹配，比如 共享的topic：/123/+/abc，输入 /123/dfdf(不合法)，/123/+/abc(合法)，/123/567/abc(合法)，详情请查看控制台配置

2，SELECT

select的属性来源于消息的payload，可以使用json表达式形式引用，也可以来源于函数比如deviceName()。

比如假定 设备发送的消息payload={a:123,b:45,c:’ ccc’ ,d:{a:55,b:’ 222’ ,c:{a:88}},e:[{e1:1},{e1:2}]} ， 对于（ select d.c.a a1,a a2，deviceName() as d1 ），则a1=88，a2=123 ,d1=asdfdsf23434 。 建议参数使用别名,比如d.c.a **a1**。

3，WHERE

规则触发条件，条件表达式。当符合topic的消息到达时，这条消息触发规则的条件，条件表达式列表见下方表格描述。

4，json表达式 select和where可以直接使用json表达式。 json表达式支持属性，也支持数组。如果payload数据解析出错将会导致规则运行失败。 转发数据action中的表达式需要使用 ${表达式} 来使用。

例子，假定
payload={ “tem” :50,” colors” :[{ “c” :” red” },{ “c” :” blue” }],” config” :{ “isonline” :0}}, 则tem=50，colors[0].c=red,config.isonline=0 。当消息规则匹配时，select属性列表可用于 转发动作参数引用。比如规则：select deviceName() as device,temperature t1 from xxxx，对于ots存储的action，配置中可以${device}、${t1} 分别引用到select中属性device t1。

数组的使用说明

数组表达式需要使用双引号，比如设备消息为： { a:[{v:1},{v:2}] } ,那么select中引用a[0]的写法为：select ".a[0]" data1,".a[1].v" data2,则data1= [{v:1}] ,data2=2

条件表达式支持列表

操作符	描述	举例
=	相等	color = ‘red’
<>	不等于	color <> ‘red’
AND	逻辑与	color = ‘red’ AND siren = ‘on’
OR	逻辑或	color = ‘red’ OR siren =

		‘on’
( )	括号代表一个整体	color = ‘red’ AND (siren = ‘on’ OR isTest)
+	算术加法	4 + 5
-	算术减	5 - 4
/	除	20 / 4
*	乘	5 * 4
%	取余数	20 % 6
<	小于	5 < 6
<=	小于或等于	5 <= 6
>	大于	6 > 5
>=	大于或等于	6 >= 5
函数调用	支持函数，详细列表请参考 函数章节。	deviceId()
JSON属性表达式	可以从消息payload以json表达式提取属性.	state.desired.color,a.b.c[0].d
CASE … WHEN … THEN … ELSE …END	Case 表达式	CASE col WHEN 1 THEN ‘Y’ WHEN 0 THEN ‘N’ ELSE ‘’ END as flag
IN	仅支持枚举，不支持子查询	比如 where a in(1,2,3); 不支持以下形式： where a in(select xxx)
like	匹配某个字符, 仅支持%号通配符, 代表匹配任意字符串	比如where c1 like ‘%abc’, where c1 not like ‘%def%’

在规则引擎中，我们提供了多种函数帮助用户实现多样化的场景，用户可以在编写SQL时使用这些函数。

如何使用函数？

当您需要对数据做一定处理，或者有些上下文信息比如（当前设备id），可以使用函数来实现。

例子：SELECT case flag when 1 then '开灯' when 2 then '关灯' else '' end flag，deviceName(),abs(temperature) tmr FROM "/topic/#" WHERE temperature>10 and topic(2)='123'

注意：通常单引号代表常量，双引号代表变量。 比如 select “a” a1, ‘a’ a2, a a3; 那么 a1与a3等效，而a2等于常量’ a’ .

支持函数列表

函数名	函数说明
deviceName()	返回当前设备名称

attribute(key)	返回key所对应的设备属性值
topic(number)	返回topic分段信息. 比如这个topic：/12345/678, topic() 返回 “/12345/678”，topic(1) 返回 “12345” , topic(2) 返回 “678”.
crypto(field,String)	对field的值进行加密。第二个参数为算法字符串，可选：’ MD2’， ‘MD5’,’ SHA1’， ‘SHA-256’， ‘SHA-384’， ‘SHA-512’
payload(textEncoding)	返回设备publish消息的payload转字符串， (字符编码默认 UTF-8)，即 payload()默认 等价于 payload(‘utf-8’).
abs(number)	返回绝对值.
asin(number)	返回 number的反正弦
upper(string)	返回大写字符
newuuid()	返回一个随机uuid字符串
rand()	返回[0~1)之间随机数
timestamp(format)	返回当前系统时间,format可选。如果为空则返回当前系统时间戳毫秒值，比如 timestamp() = 1232323233, timestamp(‘yyyy-MM-dd HH:mm:ss.SSS’)=2016-05-30 12:00:00.000
replace(source, substring, replacement)	对某个目标列值进行替换，比如 replace(field,’ a’ ,’ 1’).
concat(string1, string2)	字符串连接，比如concat(field,’ a’).
cos(number)	返回 number的余弦.
cosh(number)	返回 number的双曲余弦（hyperbolic cosine）.
endswith(input, suffix)	判断input是否以suffix结尾.
exp(number)	返回指定数字的指定次幂
floor(number)	返回一个最接近它的整数,它的值小于或等于这个浮点数
nanvl(value, default)	返回属性值如果是null则返回default
log(n, m)	返回自然对数，如果不传m，则返回log(n)
mod(n, m)	n%m 余数
sin(n)	返回 n的正弦.
sinh(n)	返回 n的双曲正弦（hyperbolic sine）.
tan(n)	返回 n的正切.
tanh(n)	返回n的双曲正切（hyperbolic tangent）.
power(n,m)	返回n的m次幂.
lower(string)	返回小写字符串

设备影子

设备影子

设备影子是一个 JSON 文档，用于存储设备上报状态、应用程序期望状态信息。每个设备有且只有一个设备影子，设备可以通过 MQTT获取和设置设备影子以此来同步状态，这个同步可以是影子同步给设备，也可以是设备同步给影子；应用程序也可以通过阿里云POP API获取和设置设备影子以此来获取设备最新状态或者下发期望状态给设备。

应用场景

场景一：

由于网络的不稳定，设备频繁的上下线。应用程序想获取一下当前的设备状态，当请求发出时，正好这个时候设备掉线，无法获得设备状态，可是下一秒这个设备又连上来了，这时候应用程序就蒙圈了，因为他不知道该什么时候请求？

当有设备影子这个机制时，这个问题就很好解决。因为影子会存储设备最新状态，一旦设备状态有变化，设备就会把状态同步到影子。这样应用程序在请求设备当前状态时，只需要获取影子中的状态即可，不需要关心设备是否在线，随时随地请求。

场景二：

假如设备网络稳定，有很多应用程序来请求设备状态，那就意味着设备需要根据这些请求响应多次，哪怕这些响应的结果都是一样的，这样做根本就是没有必要的，而且设备本身处理能力有限，可能根本负载这种被请求多次的情况。

当有设备影子这个机制时，这个问题就比较好解。设备只需要主动同步状态一次给设备影子，然后多个应用程序只需要请求设备影子即可获取设备最新状态，这就做到了应用程序和设备的解耦，设备的能力得到了解放。

场景三：

假设设备网络不稳定（这种情况在无线通信环境下太普遍了），设备频繁的上下线。这时候应用程序发送控制指令给设备，而设备刚好这时候掉线，那就意味着指令无法下达到设备，而且这种情况是大

概率事件，这种情况对于大部分用户是不可接受的。当然这种场景也可以通过QoS=1或者2来实现，但是这种方式对于服务端的压力比较大，一般不建议使用。

当有设备影子这个机制时，这个问题就比较好解。应用程序不需要care设备是否在线，只需要发送指令，指令会带着时间戳保存在设备影子，当设备掉线重连时就获取指令，并根据时间戳来确定是否执行，当设备判断指令过期可以选择不执行指令。因为也有很多场景并不是因为无线通信的不稳定造成的，设备并不是那种频繁上下线的场景，而是真的掉线了，这种场景指令失败那是正常的，而不应该出现设备再上线突然执行过期指令的场景，这个设备影子也预防了这种情况，就是通过指令附带时间戳来解决。

设备影子JSON格式：

```
{
"state": {
"desired": {
"attribute1": integer2,
"attribute2": "string2",
...
"attributeN": boolean2
},
"reported": {
"attribute1": integer1,
"attribute2": "string1",
...
"attributeN": boolean1
}
},
"metadata": {
"desired": {
"attribute1": {
"timestamp": timestamp
},
"attribute2": {
"timestamp": timestamp
},
...
"attributeN": {
"timestamp": timestamp
}
},
"reported": {
"attribute1": {
"timestamp": timestamp
},
"attribute2": {
"timestamp": timestamp
},
...
"attributeN": {
```

```
"timestamp": timestamp
}
}
},
"timestamp": timestamp,
"version": version
}
```

JSON属性描述：

属性	描述
desired	设备的预期状态。应用程序可以向本文档desired部分写入数据来更新事物的状态，且无需直接连接到该设备
reported	设备的报告状态。设备可以向本文档reported部分写入数据，报告其最新状态。应用程序可以读取本文档部分，获取设备的状态
metadata	这部分当用户更新State文档，设备影子服务会自动更新，不需要用户更新。文档 state 部分的元数据的信息，其中包括 state 部分中每个属性的时间戳（以 Epoch 时间表示），让您能够确定它们的更新时间
timestamp	影子文档的最新更新时间
version	用户主动更新version，那设备影子会检查请求中的version是否大于当前版本，如果大于的话，则更新设备影子，并将version更新到请求的版本，反之拒绝更新设备影子。文档每次更新时，此版本号都会递增，用于确保正在更新的文档为最新版本

示例影子JSON：

```
{
"state" : {
"desired" : {
"color" : "RED",
"sequence" : [ "RED", "GREEN", "BLUE" ]
},
"reported" : {
"color" : "GREEN"
}
},
"metadata" : {
"desired" : {
"color" : {
"timestamp" : 1469564492
},
"sequence" : {
"timestamp" : 1469564492
```

```
}
},
"reported" : {
"color" : {
"timestamp" : 1469564492
}
}
},
"timestamp" : 1469564492,
"version" : 1
}
```

空白部分：

- 仅当设备影子文档具有预期状态时，它才包含“desired”部分。例如以下没有“desired”部分的文档同样为有效影子JSON文档：

```
{
"state" : {
"reported" : {
"color" : "red",
}
},
"metadata" : {
"reported" : {
"color" : {
"timestamp" : 1469564492
}
}
},
"timestamp" : 1469564492,
"version" : 1
}
```

- “reported”部分也可以为空，例如以下没有”reported”部分的文档同样为有效影子JSON文档

```
{
"state" : {
"desired" : {
"color" : "red",
}
},
"metadata" : {
"desired" : {
"color" : {
"timestamp" : 1469564492
}
}
},
"timestamp" : 1469564492,
"version" : 1
}
```

数组：

- 设备影子支持数组，但将其视为正常值进行处理，因为对数组的更新将替换整个数组。无法更新数组的某个部分。
- 初始状态：

```
{
"reported" : { "colors" : ["RED", "GREEN", "BLUE" ] }
}
```

- 更新：

```
{
"reported" : { "colors" : ["RED"] }
}
```

最终状态：

```
{
"reported" : { "colors" : ["RED"] }
}
```

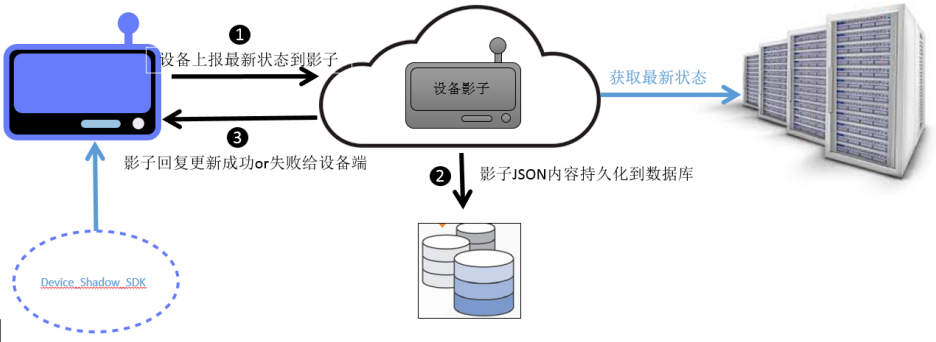
设备影子topic

- IoT套件为每个设备预定义了两个Topic实现数据流转，定义的Topic都以固定格式呈现。
- /shadow/update/${productKey}/${deviceName}
- 设备和应用程序发布消息到此Topic，套件收到该topic的消息后会将消息中的状态更新到设备影子中。
- /shadow/get/${productKey}/${deviceName}
- 设备影子会更新状态到该Topic，设备订阅此Topic的消息。

示例：

- 接下来以产品灯泡1号下面某个具体灯泡这个设备为例，productkey:10000；deviceName:lightbulb，设备以QoS=1发布订阅定义的两个Topic，举例说明设备，设备影子以及应用程序之间的通信。

设备主动上报状态：



处理流程图

当灯泡lightbulb联网时，上报状态，它将向/shadow/update/10000/lightbulb这个Topic发送消息。发送的JSON消息格式：

```
{
"method": "update",
"state": {
"reported": {
"color": "red"
}
},
"version": 1
}
```

method	表示设备或者应用程序对请求设备影子的操作类型，设备影子更新完之后的返回类型。当设备或者应用程序在更新状态的时候，固定为”update”，必填字段
state	包含设备发送给shadow的具体状态。在上报状态中，reported为必须字段，包含汇报的设备状态信息，这些信息会同步到设备影子文档中的reported部分
version	设备影子会检查请求中的version是否大于当前的version。只有在大于的情况，设备影子才会接受设备端的请求，更新设备影子，并将version更新到相应的版本

- 当设备影子接受到灯泡上报状态时，成功更新影子文档

```
{
"state" : {
"reported" : {
"color" : "red"
}
},
"metadata" : {
"reported" : {
"color" : {
"timestamp" : 1469564492
```

```
}
}
},
"timestamp" : 1469564492,
"version" : 1
}
```

- 更新设备影子之后，设备影子会返回结果给设备（灯泡），即发消息到 /shadow/get/10000/lightbulb中，设备订阅该Topic。
- 若更新成功，发送到Topic中的消息为：

```
{
"method":"reply",
"payload": {
"status":"success",
"version": 1
},
"timestamp": 1469564576
}
```

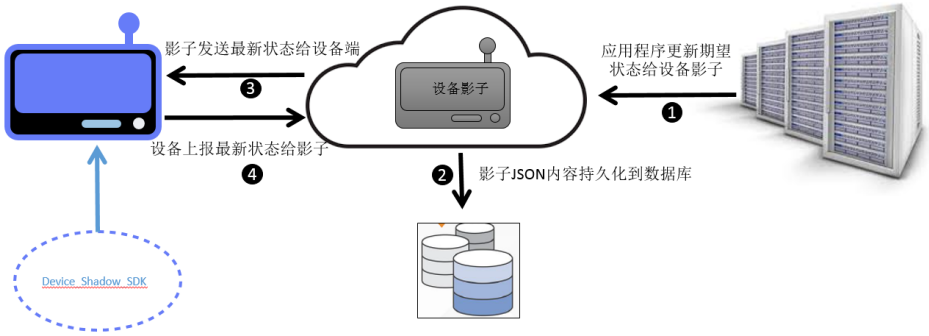
- 若更新失败，发送到Topic中的消息为：

```
{
"method":"reply",
"payload": {
"status":"error",
"content": {
"errorcode": "${errorcode}",
"errormessage": "${errormessage}"
}
},
"timestamp": 1469564576
}
```

errorCode	errorMessage
400	不正确的json格式
401	影子json缺少method信息
402	影子json缺少state字段
403	影子json version不是数字
404	影子json缺少reported字段
405	影子json reported属性字段为空
406	影子json method是无效的方法
407	影子内容为空
408	影子reported属性个数超过128个
409	影子版本冲突

500	服务端处理异常

应用程序改变设备状态:



处理流程图：

应用程序下发指令给设备影子，进而更改灯泡状态。应用程序发消息到Topic: /shadow/update/10000/lightbulb/中，消息如下：

```
{
"method": "update",
"state": {
"desired": {
"color": "green"
}
},
"version": 2
}
```

- 当应用程序发出更新请求，设备影子会更新其文档，那么影子文档将变为:

```
{
"state" : {
"reported" : {
"color" : "red"
},
"desired" : {
"color" : "green"
}
},
"metadata" : {
"reported" : {
"color" : {
"timestamp" : 1469564492
}
},
"desired" : {
"color" : {
"timestamp" : 1469564576
```

```
}
}
},
"timestamp" : 1469564576,
"version" : 2
}
```

- 更新设备影子之后，设备影子会返回结果给设备（灯泡），即发消息到/shadow/get/10000/lightbulb中，这返回的消息由设备影子决定其构成.

```
{
"method":"control",
"payload": {
"status":"success",
"state": {
"reported": {
"color": "red"
},
"desired": {
"color": "green"
}
},
"metadata": {
"reported": {
"color": {
"timestamp": 1469564492
}
},
"desired" : {
"color" : {
"timestamp" : 1469564576
}
}
}
},
"version": 2,
"timestamp": 1469564576
}
```

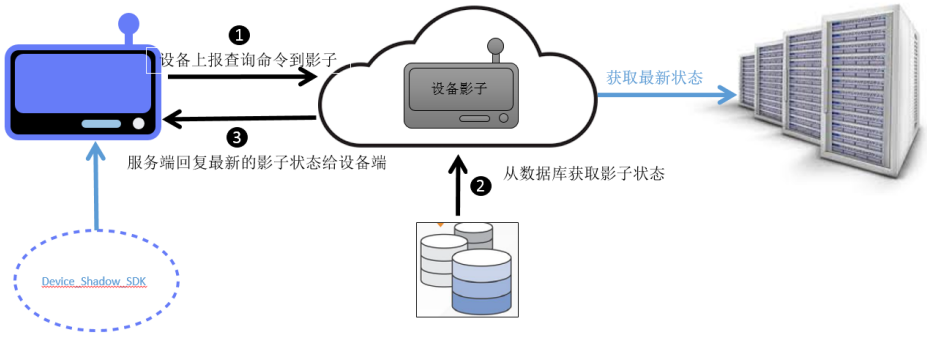
- 如果此时灯泡在线，并且订阅了Topic：/shadow/get/10000/lightbulb，就会收到消息，并根据desired文档的值更新状态（当然也可以选择不更新，例如有时间戳判断指令过期），将灯泡颜色变成绿色，更新完之后，上报最新状态，发消息到Topic：/shadow/update/10000/lightbulb中，消息如下：

```
{
"method": "update",
"state": {
"desired":"null"
},
"version": 3
}
```

- 上报状态之后，设备影子会同步更新，此时的影子文档如下：

```
{
"state" : {
"reported" : {
"color" : "green"
}
},
"metadata" : {
"reported" : {
"color" : {
"timestamp" : 1469564577
}
},
"desired" : {
"timestamp" : 1469564576
}
},
"version" : 3
}
```

设备主动获取设备影子内容:



处理流程图：

灯泡想要获取设备影子中保存的灯泡最新状态，先发一个固定消息到Topic: /shadow/update/10000/lightbulb中，具体的消息如下：

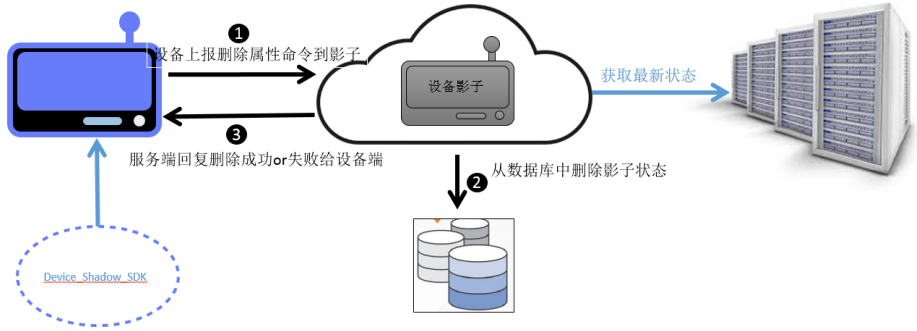
```
{
"method": "get"
}
```

- 当设备影子收到这条消息时，会发消息到/shadow/get/10000/lightbulb中，灯泡订阅该Topic，获得消息, 消息内容如下：

```
{
"method":"reply",
"payload": {
```

```
"status":"success",
"state": {
"reported": {
"color": "red"
},
"desired": {
"color": "green"
}
},
"metadata": {
"reported": {
"color": {
"timestamp": 1469564492
}
},
"desired": {
"color": {
"timestamp": 1469564492
}
}
}
},
"version": 2,
"timestamp": 1469564576
}
```


设备端删除影子属性



- 处理流程图：
- 灯泡想要删除设备影子中保存的灯泡某条属性状态，发送删除影子属性的json内容到Topic: /shadow/update/10000/lightbulb中，具体的消息如下：


```
删除影子某一属性shadow json格式

{
"method": "delete",
"state": {
"reported": {
"color": "null",
"temperature":"null"
}
},
"version": 1
```

```
}

删除影子全部属性的shadow json格式

{
"method": "delete",
"state": {
"reported":"null"
},
"version": 1
}
```

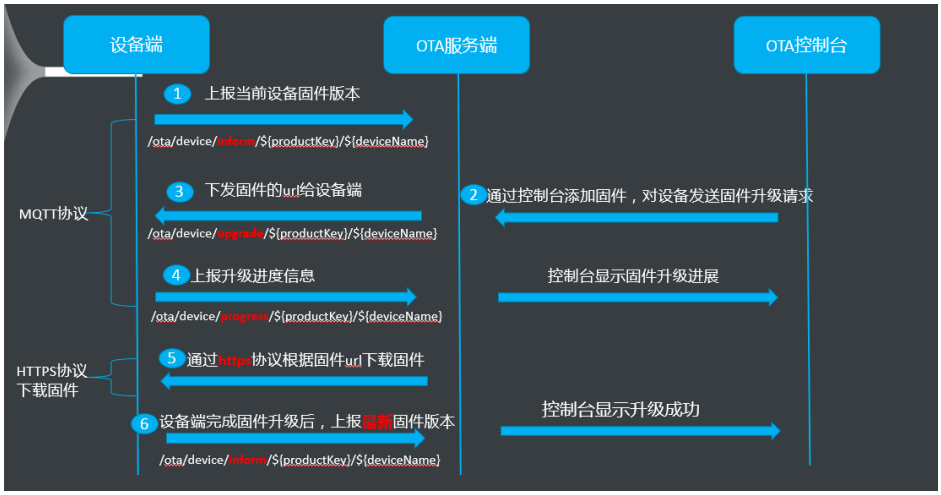
- 删除属性只需要把method置为” delete” ，并且属性的值设置为” null” 即可。

背景介绍：

- OTA服务作为物联设备的基础服务，当物联设备固件发现重要bug或紧急安全漏洞时，设备端可以通过OTA服务进行固件升级，将bug及安全风险降至最低。

固件升级流程

以MQTT协议为例：



固件升级topic

```
设备端上报固件版本给云端
/ota/device/inform/${productKey}/${deviceName}

设备端订阅该topic接收云端固件升级通知
/ota/device/upgrade/${productKey}/${deviceName}

设备端上报固件升级进度
/ota/device/progress/${productKey}/${deviceName}

设备端请求是否固件升级
/ota/device/request/${productKey}/${deviceName}
```

A.MQTT固件升级

1. 首先设备连接OTA服务，必须上报版本号。

- 设备端通过mqtt协议pub当前设备固件版本号到topic /ota/device/inform/${productKey}/${deviceName}，内容格式如下：

```
{
"id": 1,
"params": {
"version":"xxxxxxxx"
}
}
```

id：消息ID号，保留值。
version：设备当前固件版本号。

2.用户在固件升级控制台上可以添加固件，**此处需要注意的是：上传的固件类型必须是bin文件类型，否则上传不成功。**

3.用户在控制台对指定版本号的设备进行升级，升级分为几种：验证固件、批量升级、再次升级，这些本质上都是对设备的一次升级操作，**但是批量升级的前提是：该固件已通过验证，未验证的固件不可以进行批量升级**

- 验证固件：执行该功能后，控制台会返回此次验证的结果，选取的设备升级成功或者失败，升级记录可以在控制台的设备升级列表中查看。
- 批量升级：执行批量升级后，可以在控制台上设备升级列表中查看升级结果。
- 再次升级：该功能主要针对升级失败的设备进行操作，对于待升级，正在升级，升级成功的设备，是无效的，不可操作的。

4.触发升级操作之后，设备会收到OTA服务推送的固件的URL地址

- 设备端订阅topic /ota/device/upgrade/${productKey}/${deviceName}，控制台对设备发起固件升级请求后，设备端会通过该topic收到固件的URL，格式如下：

```
{
"code": "1000",
"data": {
"size": 432945,
"version": "2.0.0",
"url": "https://iotx-ota-pre.oss-cn-
shanghai.aliyuncs.com/nopoll_0.4.4.tar.gz?Expires=1502955804&OSSAccessKeyId=XXXXXXXXXXXXXXXXXXXX&Sig
nature=XfgJu7P6DWWejstKJgXJEH0qAKU%3D&security-
token=CAISuQJ1q6Ft5B2yfSjIpK6MGsyN1Jx5jo6mVnfBglIPTvlvt5D50Tz2IHtIf3NpAusdsv03nWxT7v4flqFyTINVAEvYZ
JOPKGrGR0DzDbDasumZsJbo4f%2FMQBqEaXPS2MvVfJ%2BzLrf0ceusbFbpjzJ6xaCAGxypQ12iN%2B%2Fr6%2F5gdc9
FcQSkL0B8ZrFsKxBltdUROFbIKP%2BpKWSKuGfLC1dysQcO1wEP4K%2BkkMqH8Uic3h%2Boy%2BgJt8H2PpHhd9NhX
uV2WMzn2%2FdtJOiTknxR7ARasaBqhelc4zqA%2FPPlWgAKvkXba7aIoo01fV4jN5JXQfAU8KLO8tRjofHWmojNzBJAA
PpYSSy3Rvr7m5efQrrybY1lLO6iZy%2BVio2VSZDxshI5Z3McKARWct06MWV9ABA2TTXXOi40BOxuq%2B3JGoABXC54
TOlo7%2F1wTLTsCUqzzeIiXVOK8CfNOkfTucMGHkeYeCdFkm%2FkADhXAnrnGf5a4FbmKMQph2cKsr8y8UfWLC6IzvJ
sClXTnbJBMeuWIqo5zIynS1pm7gf%2F9N3hVc6%2BEeIk0xfl2tycsUpbL2FoaGk6BAF8hWSWYUXsv59d5Uk%3D",
"md5": "93230c3bde425a9d7984a594ac55ea1e"
},
"id": 1507707025,
```

```
"message": "success"
}

size：固件文件内容大小。
md5：固件内容的MD5值，32位的hex String。
url：固件的URL，带有时效30分钟，超过这个时间后再下载，服务端拒绝访问。
version：固件的目的版本号。
```

5.设备收到URL之后，通过**HTTPS协议**根据URL下载固件，**注意URL是有时效限制的，目前限制时间是30分钟，超过这个时间后，会拒绝访问。**

- 下载固件过程中，设备端必须pub升级进度到topic /ota/device/progress/${productKey}/${deviceName}，内容格式如下：

```
{
"id": 1
"params": {
"step":"1",
"desc":" xxxxxxxx "
}
}
id：消息ID号，保留值。
step：[1, 100] 下载进度比
"-1" 代表升级失败，
"-2" 代表下载失败，
"-3" 代表校验失败，
"-4" 代表烧写失败
desc: 当前步骤的描述信息，如果发生异常可以用此字段承载错误信息
```

6.设备端完成固件升级后，需要pub最新的固件版本到topic/ota/device/inform/${productKey}/${deviceName}，如果上报的版本与OTA服务要求的版本一致就认为升级成功，反之失败。

注意事项

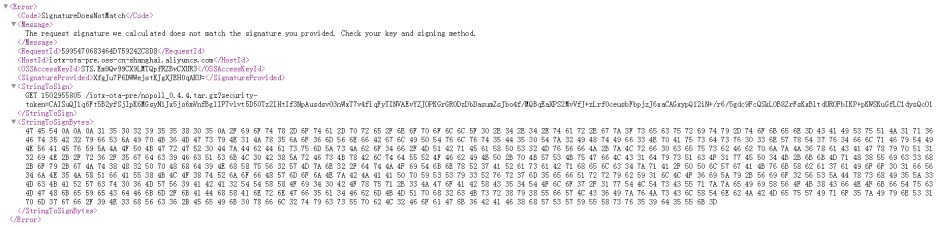
- 设备固件版本号只需要在系统启动过程中上报一次即可，不需要周期循环上报。
- 服务端检测设备端OTA是否升级成功是根据版本号来判断的，例如：设备端初始的固件版本是1.0.0，要升级的固件版本是2.0.0，设备启动过程中pub固件版本1.0.0到topic /ota/device/inform/${productKey}/${deviceName}，在设备端下载固件过程中不要上报任何版本给服务端，设备端下载完固件并且做完OTA升级后，pub固件版本2.0.0到服务端，这样服务端就认为整个升级流程成功。如果设备端下载固件过程中仍然上报固件版本1.0.0给服务端，服务端就判断上报的固件版本1.0.0和预期的固件版本2.0.0不匹配，就认为升级失败。
- 从OTA服务端控制台发起批量升级，并不代表意味着升级开始（设备升级操作记录状态是*待升级*），实际升级的开始以OTA系统接收到设备上报的升级进度为准（设备升级操作记录状态是*升级中*）。

  设备离线时接收不到服务端推送的升级消息，当设备上线后，会主动通知服务端上线消息,OTA服务端收到设备上线消息，会先验证该设备是否需要升级，如果需要会再次推送升级消息给设备, 否则，不推

送消息。

常见下载固件错误

签名错误，如果设备端获取的固件的url不全or手动修改了url内容就会出现如下错误



访问拒绝，URL过期导致，目前过期时间为30分钟。

```
▼<Error>
    <Code>AccessDenied</Code>
    <Message>Request has expired.</Message>
    <RequestId>5995498D7444FA88AE536F77</RequestId>
    <HostId>iotx-ota-pre.oss-cn-shanghai.aliyuncs.com</HostId>
    <Expires>2017-08-17T07:43:24.000Z</Expires>
    <ServerTime>2017-08-17T07:45:17.000Z</ServerTime>
  </Error>
```

B.HTTPS固件升级

- 用户在控制台触发升级操作
- 设备主动向服务端推送升级验证消息（topic: /ota/device/request/${productkey}/${devicename}）,服务端接收到该消息后，验证该设备在服务端是否存在于待升级列表中，如果存在，则设备获取服务端的固件信息。
- 后期设备推送进度消息和设备当前版本号与MQTT连接方式的过程一样，此部分请参考MQTT接入的设备。

C.COAP固件升级

- 用户在控制台触发升级操作
- 设备主动向服务端推送升级验证消息（topic: /ota/device/request/${productkey}/${devicename}）， 服务端接收到该消息后，验证该设备在服务端是否存在于待升级列表中，如果存在，则设备获取服务端的固件信息。
- 后期设备推送进度消息和设备当前版本号与MQTT连接方式的过程一样，此部分请参考MQTT接入的设备。