

深入浅出 玩转物联网平台

云运维工程师从入门到精通

作者：三烽、俏巴



- 官方文档说明快速入门阿里云物联网平台
- 真实源码详解阿里云物联网平台功能操作演示
- 最新独享实例助力开发者快速上手IoT





云服务技术大学
云产品干货高频分享



云服务技术课堂
和大牛零距离沟通



阿里云开发者 " 藏经阁 "

海量免费电子书下载

目录

基础入门篇 4

6 步快速入门阿里云物联网平台 4

物联网平台的三大功能模型（物模型） 9

接入方式篇 22

MQTT.fx：快速接入，轻松体验 22

基于开源 JAVA MQTT Client 连接阿里云 IoT 29

示例代码详解 Coap 协议接入物联网平台（java 实现） 36

HTTP 协议接入物联网平台（Getman 模拟） 47

数据流转篇 53

服务端订阅之 AMQP 53

服务端订阅之 MNS 61

服务端订阅的 4 步排查流程 69

7 个数据流转实例详解规则引擎 73

基于 Topic 消息路由构建 M2M 设备间通信架构 75

基于规则引擎构建 M2M 设备间通信架构 80

高级功能篇 85

8 步详解固件升级流程演示 85

远程配置的 2 种场景演示 92

NTP 时钟同步：如何得出设备上的精确时间 98

基于开源 Java MQTT Client 演示 RRPC 的实现 101

基础入门篇

6 步快速入门阿里云物联网平台

概述

关于阿里云物联网平台的入门案例，可以参考[官方快速入门说明](#)。本文主要演示如何使用物联网平台管理门户现有功能，快速创建一个产品及设备，并实现设备的属性上报，使刚接触该产品的用户能够以最快的方式对阿里云物联网平台有个初步的认知。

操作步骤

1. 登陆阿里云物联网平台[概览界面](#)



2. 快速开始并创建产品和设备

如何连接设备

✕

此引导将通过4个步骤，来帮助您连接设备到阿里云物联网平台：

**注册设备**

设备是平台中物理设备的表示形式和记录。任何物理设备都需要注册才能使用阿里云物联网平台。

**选择设备平台**

选择电脑或者设备运行的平台，自动生成相应的设备开发工具包。

**下载开发工具包**

开发工具包中包括一些重要组件：设备三元组、您选择的软件开发工具包和示例脚本。

**配置和测试设备**

通过使用连接工具包，您可以测试设备是否已正确连接到阿里云物联网平台，并且还可以上报数据以及接收数据。

[开始体验](#)

如何连接设备 / 第一步：注册设备

✕

1

注册设备

2

选择开发包

3

SDK下载

4

配置与测试

注册连接的设备信息

* 产品名称

设备名称 (可选)

[点开展开配置信息 \(这可在正式使用时修改\)](#)

分别填写产品名称
设备名称

[返回](#)[下一步](#)

3. 选择开发包

如何连接设备 / 第二步：选择开发包

注册设备 选择开发包 SDK下载 配置与测试

选择最有利于您连接到阿里云物联网平台的设备平台和开发工具包

选择设备平台：

Linux OS X Windows Android

选择设备连接协议：

MQTT

选择设备开发工具包：

Node.js Java 嵌入式C Android

Java 工具包，开发使用条件：
请确保已安装下面的软件，并且版本不低于指定的版本：JDK 1.8.0.144

返回 下一步

4. 下载工具包

如何连接设备 / 第三步：SDK下载

注册设备 选择开发包 SDK下载 配置与测试

下载设备连接工具包

已创建的实例：

MyQuickStartProduct
ProductKey: [redacted] DeviceName: device1 复制 DeviceSecret: ***** 显示

设备功能：

功能列表 展开

设备开发工具包：

设备三元组	ProductKey: [redacted] DeviceName: device1, DeviceSecret: ***** 显示
设备软件开发工具包	MQTT 协议的 Java SDK
用于接收和发送消息的脚本	start.bat
下载连接工具包	下载 Windows 工具包

返回 下一步

5. 配置与测试

如何连接设备 / 第四步：配置与测试

注册设备

选择开发包

SDK下载

配置与测试

运行脚本，设备连接云端上报数据并下发指令到设备端

第一步：在电脑或者嵌入式设备，解压工具包，可以执行下面的命令行：

```
unzip aliyun_iot_device_quickstart.zip
```

第二步：进入文件，执行脚本，可以执行下面的命令行：

```
./start.bat
```

第三步：运行启动脚本。下面将显示物模型的数据（注：脚本会每隔5秒上报数据，数据为：{"Status": "1", "Data": "Hello, World!"}）

设备日志 检测到设备 未激活

暂无数据

返回 完成

6. 运行程序

6.1 压缩包解压后的文件结构

名称	修改日期	类型	大小
jar	2019/8/31 14:34	文件类	
DS_Store	2019/8/31 14:34	DS_STORE 文件	9 KB
device_id_password.json	2019/8/31 14:34	JSON 文件	1 KB
HelloWorld.java	2019/8/31 14:34	JAVA 文件	8 KB
start.bat	2019/8/31 14:34	Windows 批处理文件	1 KB
start.sh	2019/8/31 14:34	SH 文件	1 KB

6.2 CMD 运行 start.bat

```
C:\Windows\System32\cmd.exe - start.bat
hing/service/property/set
2019-08-31 02:36:38.003 - [MqttSendExecutor.java] - asyncSend(162):MqttSendExecutor:subscribe: topic: [ /sys/alziesMUtYdk/
/device1/thing/service/property/set ]
2019-08-31 02:36:38.009 - [MqttNet.java] - subscribeRpc(217):MqttNet:subscribeRpc(), registerRpcListener
2019-08-31 02:36:38.010 - [MqttDefaultCallback.java] - registerRpcListener(35):MqttDefaultCallback:registerRpcListener()
, topic = /sys/alziesMUtYdk/device1/thing/service/property/set
2019-08-31 02:36:38.015 - [MqttDefaultCallback.java] - connectComplete(52):MqttDefaultCallback:after connectComplete
2019-08-31 02:36:38.024 - [HelloWorld.java] - onSubscribeSuccess(121):HelloWorld:subscribe successfully
2019-08-31 02:36:39.994 - [HelloWorld.java] - reportState(160):HelloWorld:report Hello World.
2019-08-31 02:36:39.996 - [MqttSendExecutor.java] - asyncSend(129):MqttSendExecutor:publish: topic: [ /sys/alziesMUtYdk/d
evice1/thing/event/property/post ]
2019-08-31 02:36:39.997 - [MqttSendExecutor.java] - asyncSend(130):MqttSendExecutor:publish: payload: [ {"id":"230788029
", "method": "thing.event.property.post", "params": {"Status":1, "Data": "Hello, World!", "version": "1.0"} } ]
2019-08-31 02:36:40.000 - [MqttDefaultCallback.java] - deliveryComplete(85):MqttDefaultCallback:deliveryComplete! null
2019-08-31 02:36:40.001 - [HelloWorld.java] - onSuccess(166):HelloWorld:upload successfully
2019-08-31 02:36:44.994 - [HelloWorld.java] - reportState(160):HelloWorld:report Hello World.
2019-08-31 02:36:44.995 - [MqttSendExecutor.java] - asyncSend(129):MqttSendExecutor:publish: topic: [ /sys/alziesMUtYdk/d
evice1/thing/event/property/post ]
2019-08-31 02:36:44.997 - [MqttSendExecutor.java] - asyncSend(130):MqttSendExecutor:publish: payload: [ {"id":"230788029
", "method": "thing.event.property.post", "params": {"Status":1, "Data": "Hello, World!", "version": "1.0"} } ]
2019-08-31 02:36:44.998 - [MqttDefaultCallback.java] - deliveryComplete(85):MqttDefaultCallback:deliveryComplete! null
2019-08-31 02:36:44.999 - [HelloWorld.java] - onSuccess(166):HelloWorld:upload successfully
2019-08-31 02:36:49.993 - [HelloWorld.java] - reportState(160):HelloWorld:report Hello World.
2019-08-31 02:36:49.993 - [MqttSendExecutor.java] - asyncSend(129):MqttSendExecutor:publish: topic: [ /sys/alziesMUtYdk/d
evice1/thing/event/property/post ]
2019-08-31 02:36:49.994 - [MqttSendExecutor.java] - asyncSend(130):MqttSendExecutor:publish: payload: [ {"id":"230788029
", "method": "thing.event.property.post", "params": {"Status":1, "Data": "Hello, World!", "version": "1.0"} } ]
2019-08-31 02:36:49.996 - [MqttDefaultCallback.java] - deliveryComplete(85):MqttDefaultCallback:deliveryComplete! null
2019-08-31 02:36:49.996 - [HelloWorld.java] - onSuccess(166):HelloWorld:upload successfully
```



6.3 如果需要在设备端功能扩展，可以集成压缩包中的 jar 包到应用程序中，在 IDE 中进行相关功能的扩展

设备端开发可以参考[设备端 SDK](#)

物联网平台的三大功能模型（物模型）

简介：物模型指将物理空间中的实体数字化，并在云端构建该实体的数据模型。在物联网平台中，定义物模型即定义产品功能。完成功能定义后，系统将自动生成该产品的物模型。物模型描述产品是什么，能做什么，可以对外提供哪些服务。物模型将产品功能类型分为三类：属性、服务、和事件。定义了这三类功能，即完成了物模型的定义。

功能类型	说明
属性 (Property)	一般用于描述设备运行时的状态，如环境监测设备所读取的当前环境温度等。属性支持 GET 和 SET 请求方式。应用系统可发起对属性的读取和设置请求。
服务 (Service)	设备可被外部调用的能力或方法，可设置输入参数和输出参数。相比于属性，服务可通过一条指令实现更复杂的业务逻辑，如执行某项特定的任务。
事件 (Event)	设备运行时的事件。事件一般包含需要被外部感知和处理的 notification 信息，可包含多个输出参数。如，某项任务完成的信息，或者设备发生故障或告警时的温度等，事件可以被订阅和推送。

使用：设备端可以上报属性和事件；云端可以向设备端发送设置属性和调用服务的指令。

Step By Step

1. 产品物模型 model.json(替换产品 productKey 导入即可)

```
{
  "schema": "https://iotx-tsl.oss-ap-southeast-1.aliyuncs.com/schema.json",
  "profile": {
    "productKey": "alqLU*****"
  },
  "properties": [
    {
      "identifier": "Temperature",
      "name": " 温度 ",
      "accessMode": "rw",
```

```

        "desc": "电机工作温度 ",
        "required": false,
        "dataType": {
            "type": "float",
            "specs": {
                "min": "-55",
                "max": "200",
                "unit": "°C ",
                "step": "0.1"
            }
        }
    },
    ],
    "events": [
        {
            "identifier": "post",
            "name": "post",
            "type": "info",
            "required": true,
            "desc": "属性上报 ",
            "method": "thing.event.property.post",
            "outputData": [
                {
                    "identifier": "Temperature",
                    "name": "温度 ",
                    "dataType": {
                        "type": "float",
                        "specs": {
                            "min": "-55",
                            "max": "200",
                            "unit": "°C ",
                            "step": "0.1"
                        }
                    }
                }
            ]
        }
    ],
    {
        "identifier": "event1",
        "name": "异常事件 ",
        "type": "info",
        "required": false,
        "method": "thing.event.event1.post",
        "outputData": [
            {
                "identifier": "event1",
                "name": "事件参数1 ",
                "dataType": {
                    "type": "int",

```

```

        "specs": {
            "min": "1",
            "max": "100",
            "step": "1"
        }
    }
}
],
{
    "services": [
        {
            "identifier": "set",
            "name": "set",
            "required": true,
            "callType": "async",
            "desc": " 属性设置 ",
            "method": "thing.service.property.set",
            "inputData": [
                {
                    "identifier": "Temperature",
                    "name": " 温度 ",
                    "dataType": {
                        "type": "float",
                        "specs": {
                            "min": "-55",
                            "max": "200",
                            "unit": "℃ ",
                            "step": "0.1"
                        }
                    }
                }
            ],
            "outputData": [
                {
                    "identifier": "Temperature"
                }
            ]
        },
        {
            "identifier": "get",
            "name": "get",
            "required": true,
            "callType": "async",
            "desc": " 属性获取 ",
            "method": "thing.service.property.get",
            "inputData": [
                "Temperature"
            ],
            "outputData": [
                {

```

```

        "identifier": "Temperature",
        "name": " 温度 ",
        "dataType": {
            "type": "float",
            "specs": {
                "min": "-55",
                "max": "200",
                "unit": "°C ",
                "step": "0.1"
            }
        }
    }
}
],
{
    "identifier": "addFuctionServiceAsync",
    "name": " 异步加法服务 ",
    "required": false,
    "callType": "async",
    "method": "thing.service.addFuctionServiceAsync",
    "inputData": [
        {
            "identifier": "add1",
            "name": " 加数 1",
            "dataType": {
                "type": "int",
                "specs": {
                    "min": "1",
                    "max": "100",
                    "step": "1"
                }
            }
        },
        {
            "identifier": "add2",
            "name": " 加数 2",
            "dataType": {
                "type": "int",
                "specs": {
                    "min": "1",
                    "max": "100",
                    "step": "1"
                }
            }
        }
    ],
    "outputData": [
        {
            "identifier": "result",

```

```

        "name": "结果 ",
        "dataType": {
            "type": "int",
            "specs": {
                "min": "1",
                "max": "200",
                "step": "1"
            }
        }
    }
}
],
{
    "identifier": "addFuctionServiceSync",
    "name": "加法服务 ",
    "required": false,
    "callType": "sync",
    "desc": "同步功能 ",
    "method": "thing.service.addFuctionServiceSync",
    "inputData": [
        {
            "identifier": "add2",
            "name": "加数 2",
            "dataType": {
                "type": "int",
                "specs": {
                    "min": "1",
                    "max": "100",
                    "step": "1"
                }
            }
        },
        {
            "identifier": "add1",
            "name": "加数 1",
            "dataType": {
                "type": "int",
                "specs": {
                    "min": "1",
                    "max": "100",
                    "step": "1"
                }
            }
        }
    ]
},
"outputData": [
    {
        "identifier": "result",
        "name": "计算结果 ",

```

```

        "dataType": {
            "type": "int",
            "specs": {
                "min": "1",
                "max": "200",
                "step": "1"
            }
        }
    }
}
]
}
]
}

```

2. 设备端 Code Sample(基于开源 Java MQTT Client)

```

import com.alibaba.fastjson.JSONObject;
import com.alibaba.taro.AliyunIoTSignUtil;
import org.eclipse.paho.client.mqttv3.*;
import org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;

import java.util.HashMap;
import java.util.Map;

public class IoTThingTest {

    public static String productKey = "a1qLU*****";
    public static String deviceName = "device1";
    public static String deviceSecret = "cA9wzIM6bL2QI6DgAaS00FPg*****";
    public static String regionId = "cn-shanghai";

    // 物模型 - 属性上报 topic
    private static String pubTopic = "/sys/" + productKey + "/" + deviceName
+ "/thing/event/property/post";
    // 物模型 - 属性响应 topic
    private static String subTopic = "/sys/" + productKey + "/" + deviceName
+ "/thing/event/property/post_reply";
    // 物模型 - 事件上报 topic
    private static String eventPubTopic = "/sys/" + productKey + "/" +
deviceName + "/thing/event/event1/post";

    // 物模型 - 事件上报响应 topic
    private static String eventSubTopic = "/sys/" + productKey + "/" +
deviceName + "/thing/event/event1/post_reply";

    private static MqttClient mqttClient;

```

```
public static void main(String [] args){

    // 初始化 Client
    initAliyunIoTClient();
    try {
        mqttClient.subscribe(subTopic); // 订阅 Topic
        mqttClient.subscribe(eventSubTopic);
    } catch (MqttException e) {
        System.out.println("error:" + e.getMessage());
        e.printStackTrace();
    }

    // 设置订阅监听
    mqttClient.setCallback(new MqttCallback() {
        @Override
        public void connectionLost(Throwable throwable) {
            System.out.println("connection Lost");
        }
        @Override
        public void messageArrived(String s, MqttMessage mqttMessage)
throws Exception {
            System.out.println("Sub message");
            System.out.println("Topic : " + s);
            System.out.println(new String(mqttMessage.getPayload())); //
打印输出消息 payload

            // 服务调用处理
            if (s.contains("Async") || s.contains("rrpc")) {
                String content = new String((byte[]) mqttMessage.
getPayload());

                System.out.println("服务请求 Topic: " + s);
                System.out.println("服务指令: " + content);

                JSONObject request = JSONObject.parseObject(content);
                JSONObject params = request.getJSONObject("params");
                if (!params.containsKey("add1")) { // 检查入参
                    System.out.println("不包含参数 add1");
                    return;
                }
                Integer input1 = params.getInteger("add1"); // 获取入参
                Integer input2 = params.getInteger("add2"); // 获取入参
                JSONObject response = new JSONObject();
                JSONObject data = new JSONObject();
                data.put("result", input1 + input2);
                response.put("id", request.get("id"));
                response.put("code", 200);
                response.put("data", data);
            }
        }
    });
}
```

```

        String responseTopic = s;
        // 服务响应
        if (s.contains("rrpc")) {
            // 同步服务调用响应 Topic
            responseTopic = s.replace("request", "response");
        } else {
            // 异步服务调用响应 Topic
            responseTopic = responseTopic + "_reply";
        }
        MqttMessage message1 = new MqttMessage(response.
toString().getBytes("utf-8"));
        System.out.println("responseTopic: " + responseTopic);
        mqttClient.publish(responseTopic, message1);
    }
}

@Override
public void deliveryComplete(IMqttDeliveryToken
iMqttDeliveryToken) {

    }

};

// 属性上报
postDeviceProperties();
// 事件上报
postDeviceEvent();
}

/**
 * 初始化 Client 对象
 */
private static void initAliyunIoTClient() {

    try {
        // 构造连接需要的参数
        String clientId = "java" + System.currentTimeMillis();
        Map<String, String> params = new HashMap<>(16);
        params.put("productKey", productKey);
        params.put("deviceName", deviceName);
        params.put("clientId", clientId);
        String timestamp = String.valueOf(System.currentTimeMillis());
        params.put("timestamp", timestamp);
        // cn-shanghai
        String targetServer = "tcp://" + productKey + ".iot-as-
mqtt."+regionId+".aliyuncs.com:1883";

        String mqttclientId = clientId +
"|securemode=3,signmethod=hmacsha1,timestamp=" + timestamp + "|";

```



```

        String mqttUsername = deviceName + "&" + productKey;
        String mqttPassword = AliyunIoTSignUtil.sign(params,
deviceSecret, "hmacsha1");

        connectMqtt(targetServer, mqttclientId, mqttUsername,
mqttPassword);

        } catch (Exception e) {
            System.out.println("initAliyunIoTClient error " +
e.getMessage());
        }
    }

    public static void connectMqtt(String url, String clientId, String
mqttUsername, String mqttPassword) throws Exception {

        MemoryPersistence persistence = new MemoryPersistence();
        mqttClient = new MqttClient(url, clientId, persistence);
        MqttConnectOptions connOpts = new MqttConnectOptions();
        // MQTT 3.1.1
        connOpts.setMqttVersion(4);
        connOpts.setAutomaticReconnect(false);
        connOpts.setConnectionTimeout(10);
//        connOpts.setCleanSession(true);
        connOpts.setCleanSession(false);

        connOpts.setUserName(mqttUsername);
        connOpts.setPassword(mqttPassword.toCharArray());
        connOpts.setKeepAliveInterval(60);

        mqttClient.connect(connOpts);
    }

    /**
     * 事件上报
     */
    private static void postDeviceEvent() {

        try {
            // 上报数据
            // 高级版 物模型 - 属性上报 payload
            System.out.println("事件上报");
            String payloadJson = "{\"params\":{\"event1\":23}}";
            MqttMessage message = new MqttMessage(payloadJson.
getBytes("utf-8"));
            message.setQos(1);
            mqttClient.publish(eventPubTopic, message);
        } catch (Exception e) {
            System.out.println(e.getMessage());

```

```

    }
}

/**
 * 汇报属性
 */
private static void postDeviceProperties() {

    try {
        // 上报数据
        // 高级版 物模型 - 属性上报 payload
        System.out.println("上报属性值 ");
        String payloadJson = "{\"params\":{\"Temperature\":\"13\"}}";
        MqttMessage message = new MqttMessage(payloadJson.
getBytes("utf-8"));
        message.setQos(1);
        mqttClient.publish(pubTopic, message);
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}
}
}

```

参考链接: [基于开源 JAVA MQTT Client 连接阿里云 IoT](#)

3. 启动设备查看属性及事件上报情况

3.1 服务端情况

The first screenshot shows the 'device1' page with the '运行状态' (Running Status) tab selected. A red box highlights the temperature value '13°C' and the text '属性上报显示情况' (Attribute report display situation).

The second screenshot shows the 'device1' page with the '事件管理' (Event Management) tab selected. A red box highlights the '时间上报情况' (Time report situation) table, which contains the following data:

时间	设备ID	属性名称	事件类型	属性值
2020-03-18 15:11:12	device1	温度	上报	13

3.2 设备端 reply 情况

```
上报属性值
事件上报
Sub message
Topic : /sys/alqLU*****/device1/thing/event/property/post_reply
{"code":200,"data":{},"id":"null","message":"success","method":"thing.event.
property.post","version":"1.0"}
Sub message
Topic : /sys/alqLU*****/device1/thing/event/event1/post_reply
{"code":200,"data":{},"id":"null","message":"success","method":"thing.event.
event1.post","version":"1.0"}
```

4. 异步服务调用

4.1 说明

通过 [InvokeThingService](#) 或 [InvokeThingsService](#) 接口调用服务，物联网平台采用异步方式下行推送请求，设备也采用异步方式返回结果。此时，服务选择为异步调用方式，物联网平台订阅此处的异步响应 Topic。异步调用的结果，可以使用[规则引擎数据流转功能](#)获取，也可以使用服务端订阅获取。

4.2 Open API [InvokeThingService](#)



4.3 设备端日志

```
Sub message
Topic : /sys/alqLU*****/device1/thing/service/addFuctionServiceAsync
{"method": "thing.service.
addFuctionServiceAsync", "id": "2015524670", "params": {"add2": 2, "add1": 2}, "version": "1.0.0"}
服务请求 Topic: /sys/alqLU*****/device1/thing/service/addFuctionServiceAsync
服务指令: {"method": "thing.service.
addFuctionServiceAsync", "id": "2015524670", "params": {"add2": 2, "add1": 2}, "version": "1.0.0"}
responseTopic: /sys/alqLU*****/device1/thing/service/addFuctionServiceAsync_reply
```

4.4 控制台日志

设备信息	Topic列表	运行状态	事件管理	服务调用	设备影子	文件管理	日志服务	在线调试
请输入服务标识符 Q 1小时 ?								
时间	标识符	服务名称	输入参数	输出参数				
2020/03/08 19:08:02	addFuctionServiceAsync	异步加法服务	["add2":2,"add1":2]	["code":200,"data":{"result":4,"id":"2015524670","message":"","version":"1.0"}]				

4.5 AMQP 服务端订阅获取的服务响应

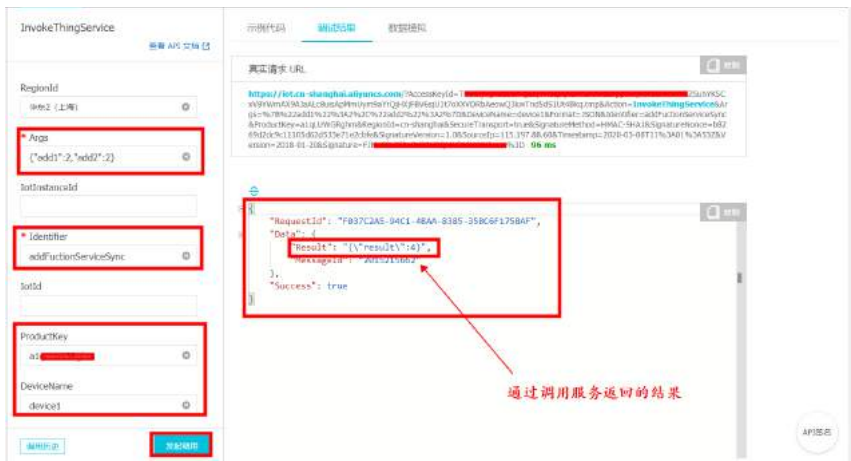
```
Content:{"iotId":"*****","code":200,"data":{"result":4},"requestId":"
2009566194","topic":"/sys/*****/device1/thing/service/addFuctionServiceAsync_
reply","source":"DEVICE","gmtCreate":1583658474852,"productKey":"alqLU****","
deviceName":"device1"}
```

5. 同步服务调用

5.1 说明

通过 [InvokeThingService](#) 或 [InvokeThingsService](#) 接口调用服务，物联网平台直接使用 [RRPC 同步](#) 方式下行推送请求。此时，服务选择为同步调用方式，物联网平台订阅 RRPC 对应 Topic。

5.2 Open API [InvokeThingService](#)



5.3 设备端日志

```
Sub message
Topic : /sys/alqLU*****/device1/rrpc/request/1236608506958775809
{"method":"thing.service.addFuctionServiceSync","id":"2015301903","params":{"add2":2,"add1":2},"version":"1.0.0"}
服务请求 Topic: /sys/alqLU*****/device1/rrpc/request/1236608506958775809
服务指令: {"method":"thing.service.addFuctionServiceSync","id":"2015301903","params":{"add2":2,"add1":2},"version":"1.0.0"}
responseTopic: /sys/alqLU*****/device1/rrpc/response/1236608506958775809
```

5.4 控制台日志

设备信息	Topic列表	运行状态	事件管理	服务调用	设备影子	文件管理	日志服务	在线调试
请输入设备标识符	Q	12时						
时间	标识符	服务名称	输入参数	输出参数				
2020/03/08 19:03:37	addFuctionServiceSync	添加设备	["add2":2,"add1":2]	["code":200,"data":{"result":"","},"version":"1.0.0"]				

更多参考

[设备属性、事件、服务](#)

[同步服务调用](#)

[基于开源 Java MQTT Client 的阿里云物联网平台 RRPC 功能测试](#)

[阿里云物联网平台规则引擎综述](#)

接入方式篇

MQTT.fx：快速接入，轻松体验

简介：本文演示如何使用 MQTT.fx 工具接入物联网平台，进行简单的发布和订阅消息。

概述

MQTT.fx 是一款基于 Eclipse Paho，使用 Java 语言编写的 MQTT 客户端工具，支持通过 Topic 订阅和发布消息。

对于刚入门的新手，可以通过这个第三方软件，以 MQTT 协议快速接入阿里云物联网平台进行体验。

官方文档：[使用 MQTT.fx 接入物联网平台](#)

创建产品和服务

登录阿里云物联网平台的[控制台](#)，创建产品并添加一个设备。

- 创建产品

← 创建产品 (设备模型)

* 产品名称

MQTTfx测试产品

* 所属品类 ?

☐ 标准品类

☒ 自定义品类

* 节点类型



联网与数据

* 联网方式

WiFi

* 数据格式 ?

ICA 标准数据格式 (Alink JSON)

• 添加设备

物联网平台 / 设备管理 / 设备

设备

MQTTfx测试产品

设备列表 批次管理

添加设备 批量添加 DeviceName/备注名称

添加设备 ?

特别说明: deviceName可以为空, 当为空时, 阿里云会颁发全局唯一标识符作为deviceName。

产品MQTTfx测试产品

DeviceName ?MQTTfx_device

备注名称 ?MQTTfx测试设备

确认 取消

MQTT.fx 软件及签名工具下载

访问 MQTT.fx 官网, 下载软件并安装。[官网地址](#)



- 下载连接参数 Password 的生成小工具。[下载地址](#)



具体配置

打开 MQTT.fx 软件, 进行配置。

- 基本信息配置

1. 输入自定义名称 **PK**。
2. 连接域名格式为: `${YourProductKey}.iot-as-mqtt.${region}.aliyuncs.com`
其中, `${YourProductKey}` 为产品对应的 productKey, `${region}` 为产品所

在地域的代码 (例如 cn-shanghai) [地域和可用区](#)。

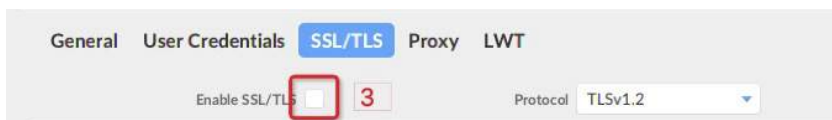
3. 端口: 1883。
4. [特别注意] 这个参数是 ClientID, 格式为: `#{clientId}|securemode=3,signmethod=hmacsha1|` (注意最后面有一个 |, 截图因为长度问题没显示) `{clientId}` 可以自定义, 本文中用 12345 (ClientID 和 clientId 务必区别开); `securemode` 为安全模式, TCP 直连模式设置为 `securemode=3`, TLS 直连为 `securemode=2`, 本文用 TCP 直连; `signmethod` 为算法类型, 支持 `hmacmd5` 和 `hmacsha1`, 本文用 `hmacsha1`。

The screenshot shows the 'MQTT Broker Profile Settings' form. The 'Profile Name' field is labeled '1' and contains 'IoT-Test'. The 'Profile Type' is set to 'MQTT Broker'. The 'Broker Address' field is labeled '2' and contains 'a1LhUsKS74u.iot-as-mqtt.cn-shanghai.aliyunc'. The 'Broker Port' field is labeled '3' and contains '1883'. The 'Client ID' field is labeled '4' and contains '12345|securemode=3,signmethod=hmacsha1|'. There is a 'Generate' button next to the Client ID field.

- User Credentials&SSL/TLS 配置

1. User Name 格式为: `{YourDeviceName}&{YourProductKey}` (注意中间有一个 &)。
2. 使用工具来生成, 详见【 Password 生成】。
3. 因为选择的是 TCP 直连 (`securemode=3`, 所以 SSL/TLS 里面不要勾选)。

The screenshot shows the 'User Credentials' tab in the MQTT.fx configuration. The 'User Name' field is labeled '1' and contains 'MQTTfx_device&a1LhUsKS74u'. The 'Password' field is labeled '2' and contains a series of dots, indicating a masked password.



- Password 生成

1. 使用密码生成工具, 打开 sign.html
2. 拿到 password

填入设备信息:

productKey: a1LhUuKS74u **控制台上直接复制即可**

deviceName: MQTTfx_device

deviceSecret: HBaVW7A9xVhd6qk06VvWlo9cB0mDkrBt

timestamp:

clientId: 12345 **注意这里是 12345**

method: hmacsha1

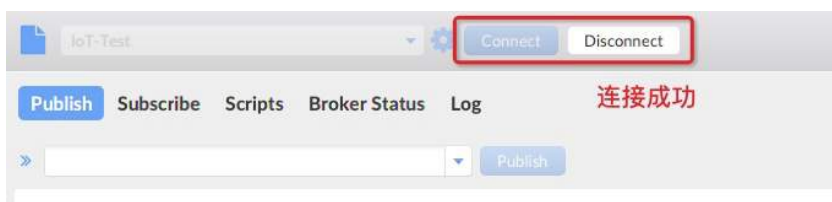
点击这里: [Generate](#)

签名结果:

password: 27F000BC6F097a4419D916EA44B232A399DB8F51

- 连接阿里云物联网平台

1. 配置完成后点击 Connect, 如图所示即为连接成功



2. 控制台查看设备连接情况



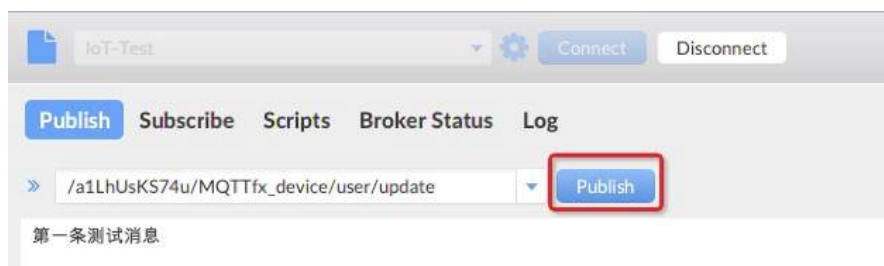
发布和订阅消息

- 通过自定义 topic 发布消息

1. 选择发布的 topic



2. 在 MQTT.fx 上发布消息

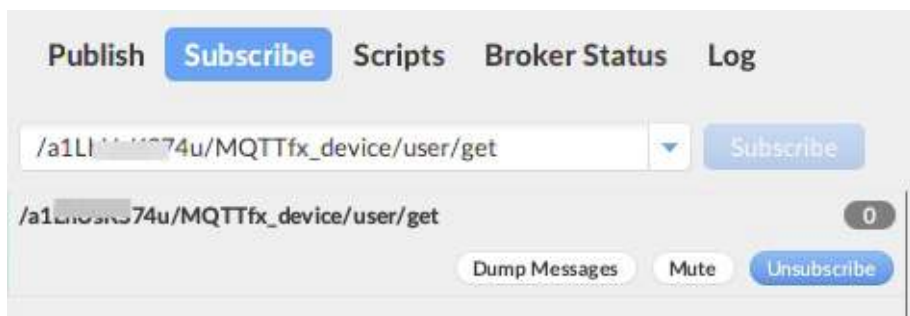


3. 在控制台的日志服务中查看消息



- 通过自定义 topic 订阅消息

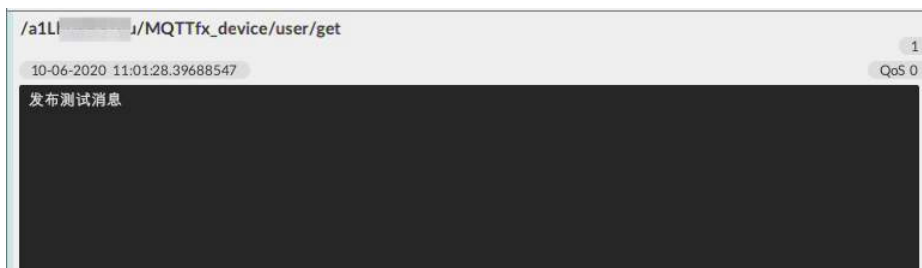
1. 订阅 topic



2. 通过控制台发布消息



3. 在 MQTT.fx 上看到阿里云物联网平台发送过来的消息



基于开源 JAVA MQTT Client 连接阿里云 IoT

概述

在使用阿里云官方 IoT JAVA Device SDK 连接云端测试的时候，发现日志总是会打印一些莫名其妙 Topic 消息的订阅和发布，但是用户并没有操作这些 Topic，这是因为 SDK 底层默认做了很多系统 Topic 的订阅和发布设置，且无法关闭，导致很多测试不能满足预期的测试期望。如果不希望一些系统 Topic 的默认订阅和发布，建议使用开源 MQTT Client 进行 Topic 消息的订阅和发布。

操作步骤

1. 创建产品和设备

参考：[阿里云物联网平台 Quick Start](#) 创建产品和设备部分。

2. pom.xml

```
<dependencies>
  <dependency>
    <groupId>org.eclipse.paho</groupId>
    <artifactId>org.eclipse.paho.client.mqttv3</artifactId>
    <version>1.1.0</version>
  </dependency>
  <dependency>
    <groupId>com.google.guava</groupId>
    <artifactId>guava</artifactId>
    <version>23.0</version>
  </dependency>
</dependencies>
```

3. 工具类 AliyunIoTSigntUtil

```

import javax.crypto.Mac;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Arrays;
import java.util.Map;

/**
 * AliyunIoTSigntUtil
 */

public class AliyunIoTSigntUtil {
    public static String sign(Map<String, String> params, String
deviceSecret, String signMethod) {
        // 将参数 Key 按字典顺序排序
        String[] sortedKeys = params.keySet().toArray(new String[] {});
        Arrays.sort(sortedKeys);

        // 生成规范化请求字符串
        StringBuilder canonicalizedQueryString = new StringBuilder();
        for (String key : sortedKeys) {
            if ("sign".equalsIgnoreCase(key)) {
                continue;
            }
            canonicalizedQueryString.append(key).append(params.get(key));
        }

        try {
            String key = deviceSecret;
            return encryptHMAC(signMethod, canonicalizedQueryString.
toString(), key);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    /**
     * HMACSHA1 加密
     */
    public static String encryptHMAC(String signMethod, String content, String
key) throws Exception {
        SecretKey secretKey = new SecretKeySpec(key.getBytes("utf-8"),
signMethod);
        Mac mac = Mac.getInstance(secretKey.getAlgorithm());
        mac.init(secretKey);
        byte[] data = mac.doFinal(content.getBytes("utf-8"));
        return bytesToHexString(data);
    }
}

```

```

public static final String bytesToHexString(byte[] bArray) {

    StringBuffer sb = new StringBuffer(bArray.length);
    String sTemp;
    for (int i = 0; i < bArray.length; i++) {
        sTemp = Integer.toHexString(0xFF & bArray[i]);
        if (sTemp.length() < 2) {
            sb.append(0);
        }
        sb.append(sTemp.toUpperCase());
    }
    return sb.toString();
}
}

```

4. main 方法

```

import com.alibaba.taro.AliyunIoTSignUtil;
import org.eclipse.paho.client.mqttv3.*;
import org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;
import java.util.HashMap;
import java.util.Map;

public class IoTDemoPubSubDemo {

    public static String productKey = "*****";
    public static String deviceName = "OpenMQTTDevice";
    public static String deviceSecret = "*****";
    public static String regionId = "cn-shanghai";

    // 物模型 - 属性上报 topic
    private static String pubTopic = "/sys/" + productKey + "/" + deviceName
+ "/thing/event/property/post";
    // 自定义 topic, 在产品 Topic 列表位置定义
    private static String subTopic = "/" + productKey + "/" + deviceName + "/"
+ user/newdatademo";

    private static MqttClient mqttClient;

    public static void main(String [] args){

        initAliyunIoTClient();
        //      ScheduledExecutorService scheduledThreadPool = new
ScheduledThreadPoolExecutor(1,
        //          new ThreadFactoryBuilder().setNameFormat("thread-runner-
%d").build());
        //
        //      scheduledThreadPool.scheduleAtFixedRate(()->postDeviceProperties(),
10,10, TimeUnit.SECONDS);
    }
}

```

```

        // 汇报属性
        postDeviceProperties();
        try {
            mqttClient.subscribe(subTopic); // 订阅 Topic
        } catch (MqttException e) {
            System.out.println("error:" + e.getMessage());
            e.printStackTrace();
        }

        // 设置订阅监听
        mqttClient.setCallback(new MqttCallback() {
            @Override
            public void connectionLost(Throwable throwable) {
                System.out.println("connection Lost");
            }

            @Override
            public void messageArrived(String s, MqttMessage mqttMessage)
                throws Exception {
                System.out.println("Sub message");
                System.out.println("Topic : " + s);
                System.out.println(new String(mqttMessage.getPayload())); //
                打印输出消息 payload
            }

            @Override
            public void deliveryComplete(IMqttDeliveryToken
                iMqttDeliveryToken) {
            }
        });
    }

    /**
     * 初始化 Client 对象
     */
    private static void initAliyunIoTClient() {
        try {
            // 构造连接需要的参数
            String clientId = "java" + System.currentTimeMillis();
            Map<String, String> params = new HashMap<>(16);
            params.put("productKey", productKey);
            params.put("deviceName", deviceName);
            params.put("clientId", clientId);
            String timestamp = String.valueOf(System.currentTimeMillis());
            params.put("timestamp", timestamp);
            // cn-shanghai
            String targetServer = "tcp://" + productKey + ".iot-as-
            mqtt."+regionId+".aliyuncs.com:1883";

```



```
String mqttclientId = clientId +
"|securemode=3,signmethod=hmacsha1,timestamp=" + timestamp + "|";
String mqttUsername = deviceName + "&" + productKey;
String mqttPassword = AliyunIoTSignUtil.sign(params,
deviceSecret, "hmacsha1");

connectMqtt(targetServer, mqttclientId, mqttUsername,
mqttPassword);

    } catch (Exception e) {
        System.out.println("initAliyunIoTClient error " +
e.getMessage());
    }
}

public static void connectMqtt(String url, String clientId, String
mqttUsername, String mqttPassword) throws Exception {

    MemoryPersistence persistence = new MemoryPersistence();
    mqttClient = new MqttClient(url, clientId, persistence);
    MqttConnectOptions connOpts = new MqttConnectOptions();
    // MQTT 3.1.1
    connOpts.setMqttVersion(4);
    connOpts.setAutomaticReconnect(false);
    connOpts.setCleanSession(true);

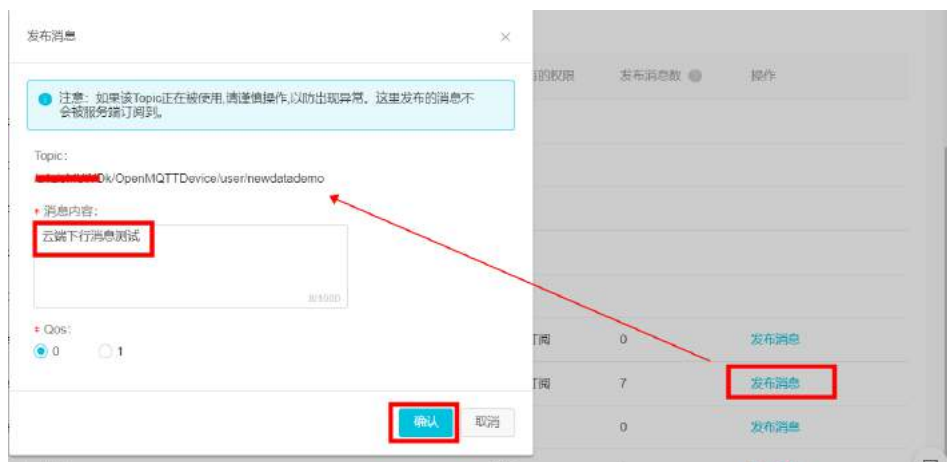
    connOpts.setUserName(mqttUsername);
    connOpts.setPassword(mqttPassword.toCharArray());
    connOpts.setKeepAliveInterval(60);

    mqttClient.connect(connOpts);
}

/**
 * 汇报属性
 */
private static void postDeviceProperties() {

    try {
        // 上报数据
        // 高级版 物模型 - 属性上报 payload
        System.out.println(" 上报属性值 ");
        String payloadJson =
"{" + "params\":" + {" + "Status\":" + 0 + "," + "Data\":" + "15\\"" + "}"}";
        MqttMessage message = new MqttMessage(payloadJson.
getBytes("utf-8"));
        message.setQos(1);
        mqttClient.publish(pubTopic, message);
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}
}
```

5. 运行测试情况





The screenshot shows a Java application window titled "IoTDemoPubSubDemo". The command prompt at the top shows the execution of "C:\Program Files\Java\jdk1.8.0_191\bin\java.exe". The main content area displays the following text:

```
上报属性值
Sub message
Topic : /[REDACTED]/OpenMQTTDevice/user/newdatademo
云端下行消息测试
```

The text is enclosed in a red rectangular border.

参考链接

[基于开源 MQTT 自主接入阿里云 IoT 平台 \(Java\)](#)

[MQTT-TCP 连接通信](#)

示例代码详解 Coap 协议接入物联网平台 (java 实现)

简介：本文介绍基于开源的 CoAP 协议进行对称加密自主接入的流程，并提供 java 示例代码。

概述

阿里云物联网平台支持 CoAP 协议连接通信。CoAP 协议适用在资源受限的低功耗设备上，尤其是 NB-IoT 的设备使用。本文介绍基于开源的 CoAP 协议进行对称加密自主接入的流程，并提供 java 示例代码。

官方文档：[CoAP 连接通信](#)

说明与限制

- Topic 规范和 MQTT Topic 一致，CoAP 协议内 `coap://host:port/topic/topic` 接口对于所有 {topic} 和 MQTT Topic 可以复用。
- 客户端缓存认证返回的 token 是请求的令牌。
- 传输的数据大小依赖于 MTU 的大小，建议在 1 KB 以内。
- 仅华东 2 (上海) 地域支持 CoAP 通信。

接入流程

连接 CoAP 服务器

- endpoint 地址为: `${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com:${port}`
- `${YourProductKey}`: 产品的 ProductKey

- `#{port}`: 端口。使用对称加密时端口为 5682

设备认证

- 认证请求

```
POST /auth
Host: ${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com
Port: 5682
Accept: application/json or application/cbor
Content-Format: application/json or application/cbor
payload: {"productKey": "a1NUjcV****", "deviceName": "ff1a1****", "clientId":
"a1NUjcV****&ff1a1****", "sign": "F9FD53EE0CD010FCA40D14A9FE*****",
"seq": "10"}
```

- 请求参数说明

参数	说明
Method	请求方法。只支持 POST 方法
URL	URL 地址, 取值: /auth
Host	Endpoint 地址。取值格式: YourProductKey.coap.cn-shanghai.link.aliyuncs.com。其中, 变量 {YourProductKey} 需替换为您的产品 Key。
Port	端口, 取值: 5682
Accept	设备接收的数据编码方式。目前, 支持两种方式: application/json 和 application/cbor。
Content-Format	设备发送给物联网平台的上行数据的编码格式, 目前支持两种方式: application/json 和 application/cbor。
payload	设备认证信息内容, JSON 数据格式。具体参数, 请参见 payload 说明 。

上报数据

- 上报数据请求

```
POST /topic/${topic}
Host: ${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com
Port: 5682
Accept: application/json or application/cbor
Content-Format: application/json or application/cbor
payload: ${your_data}
CustomOptions: number:2088(标识 token), 2089(seq)
```

- 请求参数说明

参数	是否必需	说明
Method	是	请求方法。支持 POST 方法。
URL	是	传入格式: /topic/topic。其中, 变量 {topic} 需替换为设备数据上行 Topic。
Host	是	endpoint 地址。传入格式: YourProductKey.coap.cn-shanghai.link.aliyuncs.com。其中, {YourProductKey} 需替换为设备所属产品的 Key。
Port	是	端口。取值: 5682。
Accept	是	设备接收的数据编码方式。目前, 支持两种方式: application/json 和 application/cbor。
Content-Format	是	上行数据的编码格式, 服务端对此不做校验。目前, 支持两种方式: application/json 和 application/cbor。
payload	是	待上传的数据经高级加密标准 (AES) 加密后的数据。加密规则
CustomOptions	是	option 值有 2088 和 2089 两种类型。类型解释

java 示例代码

- pom.xml 依赖

```

<dependency>
  <groupId>org.eclipse.californium</groupId>
  <artifactId>californium-core</artifactId>
  <version>2.0.0-M17</version>
</dependency>
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-lang3</artifactId>
  <version>3.5</version>
</dependency>
<dependency>
  <groupId>commons-codec</groupId>
  <artifactId>commons-codec</artifactId>
  <version>1.13</version>
</dependency>
<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>fastjson</artifactId>
  <version>1.2.61</version>
</dependency>

```

- lotCoapClientWithAes 类

```
/*
 * Copyright © 2019 Alibaba. All rights reserved.
 */

import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Set;
import java.util.SortedMap;
import java.util.TreeMap;

import javax.crypto.Cipher;
import javax.crypto.Mac;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;

import org.apache.commons.codec.DecoderException;
import org.apache.commons.codec.binary.Hex;
import org.apache.commons.lang3.RandomUtils;
import org.eclipse.californium.core.CoapClient;
import org.eclipse.californium.core.CoapResponse;
import org.eclipse.californium.core.Utils;
import org.eclipse.californium.core.coap.CoAP;
import org.eclipse.californium.core.coap.CoAP.Code;
import org.eclipse.californium.core.coap.CoAP.Type;
import org.eclipse.californium.core.coap.MediaTypeRegistry;
import org.eclipse.californium.core.coap.Option;
import org.eclipse.californium.core.coap.OptionNumberRegistry;
import org.eclipse.californium.core.coap.OptionSet;
import org.eclipse.californium.core.coap.Request;
import org.eclipse.californium.elements.exception.ConnectorException;

import com.alibaba.fastjson.JSONObject;

/**
 * CoAP 客户端连接阿里云物联网平台，基于 eclipse californium 开发。
 * 自主接入开发流程及参数填写，请参见：
 * https://help.aliyun.com/document\_detail/57697.html [使用对称加密自主接入]
 */
public class IotCoapClientWithAes {

    // ===== 需要用户填写的参数，开始 =====
    // 地域 ID，当前仅支持华东 2
    private static String regionId = "cn-shanghai";
    // 产品 productKey
    private static String productKey = "*****";
    // 设备名成 deviceName
```

```

private static String deviceName = "****";
// 设备密钥 deviceSecret
private static String deviceSecret = "****";
// 发送的消息内容 payload
private static String payload = "hello coap!!";
// ===== 需要用户填写的参数, 结束 =====

// 定义加密方式 MAC 算法可选以下多种算法 HmacMD5 HmacSHA1, 需与 signmethod 一致。
private static final String HMAC_ALGORITHM = "hmacsha1";

// CoAP 接入地址, 对称加密端口号是 5682。
private static String serverURI = "coap://" + productKey + ".coap." +
regionId + ".link.aliyuncs.com:5682";

// 发送消息用的 Topic。需要在控制台自定义 Topic, 设备操作权限需选择为 "发布"。
private static String updateTopic = "/" + productKey + "/" + deviceName +
"/user/update";

// token option
private static final int COAP2_OPTION_TOKEN = 2088;
// seq option
private static final int COAP2_OPTION_SEQ = 2089;

// 加密算法 sha256
private static final String SHA_256 = "SHA-256";

private static final int DIGITAL_16 = 16;
private static final int DIGITAL_48 = 48;

// CoAP 客户端
private CoapClient coapClient = new CoapClient();

// token 7 天有效, 失效后需要重新获取。
private String token = null;
private String random = null;
@SuppressWarnings("unused")
private long seqOffset = 0;

/**
 * 初始化 CoAP 客户端
 *
 * @param productKey 产品 key
 * @param deviceName 设备名称
 * @param deviceSecret 设备密钥
 */
public void connect(String productKey, String deviceName, String
deviceSecret) {
    try {
        // 认证 uri, /auth

```



```

        String uri = serverURI + "/auth";

        // 只支持 POST 方法
        Request request = new Request(Code.POST, Type.CON);

        // 设置 option
        OptionSet optionSet = new OptionSet();
        optionSet.addOption(new Option(OptionNumberRegistry.CONTENT_
FORMAT, MediaTypeRegistry.APPLICATION_JSON));
        optionSet.addOption(new Option(OptionNumberRegistry.ACCEPT,
MediaTypeRegistry.APPLICATION_JSON));
        request.setOptions(optionSet);

        // 设置认证 uri
        request.setURI(uri);

        // 设置认证请求 payload
        request.setPayload(authBody(productKey, deviceName,
deviceSecret));

        // 发送认证请求
        CoapResponse response = coapClient.advanced(request);
        System.out.println(Utils.prettyPrint(response));
        System.out.println();

        // 解析请求响应
        JSONObject json = JSONObject.parseObject(response.
getResponseText());
        token = json.getString("token");
        random = json.getString("random");
        seqOffset = json.getLongValue("seqOffset");
    } catch (ConnectorException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

/**
 * 发送消息
 *
 * @param topic 发送消息的 Topic
 * @param payload 消息内容
 */
public void publish(String topic, byte[] payload) {
    try {
        // 消息发布 uri, /topic/${topic}
        String uri = serverURI + "/topic" + topic;

        // AES 加密 seq, seq=RandomUtils.nextInt()

```

```

        String shaKey = encod(deviceSecret + "," + random);
        byte[] keys = Hex.decodeHex(shaKey.substring(DIGITAL_16,
DIGITAL_48));
        byte[] seqBytes = encrypt(String.valueOf(RandomUtils.nextInt()).
getBytes(StandardCharsets.UTF_8), keys);

        // 只支持 POST 方法
        Request request = new Request(CoAP.Code.POST, CoAP.Type.CON);

        // 设置 option
        OptionSet optionSet = new OptionSet();
        optionSet.addOption(new Option(OptionNumberRegistry.CONTENT_
FORMAT, MediaTypeRegistry.APPLICATION_JSON));
        optionSet.addOption(new Option(OptionNumberRegistry.ACCEPT,
MediaTypeRegistry.APPLICATION_JSON));
        optionSet.addOption(new Option(COAP2_OPTION_TOKEN, token));
        optionSet.addOption(new Option(COAP2_OPTION_SEQ, seqBytes));
        request.setOptions(optionSet);

        // 设置消息发布 uri
        request.setURI(uri);

        // 设置消息 payload
        request.setPayload(encrypt(payload, keys));

        // 发送消息
        CoapResponse response = coapClient.advanced(request);

        System.out.println("-----");
        System.out.println(request.getPayload().length);
        System.out.println("-----");
        System.out.println(Utils.prettyPrint(response));

        // 解析消息发送结果
        String result = null;
        if (response.getPayload() != null) {
            result = new String(decrypt(response.getPayload(), keys));
        }
        System.out.println("payload: " + result);
        System.out.println();
    } catch (ConnectorException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (DecoderException e) {
        e.printStackTrace();
    }
}

/**

```

```

* 生成认证请求内容
*
* @param productKey 产品 key
* @param deviceName 设备名字
* @param deviceSecret 设备密钥
* @return 认证请求
*/
private String authBody(String productKey, String deviceName, String
deviceSecret) {

    // 构建认证请求
    JSONObject body = new JSONObject();
    body.put("productKey", productKey);
    body.put("deviceName", deviceName);
    body.put("clientId", productKey + "." + deviceName);
    body.put("timestamp", String.valueOf(System.currentTimeMillis()));
    body.put("signmethod", HMAC_ALGORITHM);
    body.put("seq", DIGITAL_16);
    body.put("sign", sign(body, deviceSecret));

    System.out.println("----- auth body -----");
    System.out.println(body.toJSONString());

    return body.toJSONString();
}

/**
* 设备端签名
*
* @param params 签名参数
* @param deviceSecret 设备密钥
* @return 签名十六进制字符串
*/
private String sign(JSONObject params, String deviceSecret) {

    // 请求参数按字典顺序排序
    Set<String> keys = getSortedKeys(params);

    // sign、signmethod、version、resources 除外
    keys.remove("sign");
    keys.remove("signmethod");
    keys.remove("version");
    keys.remove("resources");

    // 组装签名明文
    StringBuffer content = new StringBuffer();
    for (String key : keys) {
        content.append(key);
        content.append(params.getString(key));
    }
}

```

```

    }

    // 计算签名
    String sign = encrypt(content.toString(), deviceSecret);
    System.out.println("sign content=" + content);
    System.out.println("sign result=" + sign);

    return sign;
}

/**
 * 获取 JSON 对象排序后的 key 集合
 *
 * @param json 需要排序的 JSON 对象
 * @return 排序后的 key 集合
 */
private Set<String> getSortedKeys(JSONObject json) {
    SortedMap<String, String> map = new TreeMap<String, String>();
    for (String key : json.keySet()) {
        String vlaue = json.getString(key);
        map.put(key, vlaue);
    }
    return map.keySet();
}

/**
 * 使用 HMAC_ALGORITHM 加密
 *
 * @param content 明文
 * @param secret 密钥
 * @return 密文
 */
private String encrypt(String content, String secret) {
    try {
        byte[] text = content.getBytes(StandardCharsets.UTF_8);
        byte[] key = secret.getBytes(StandardCharsets.UTF_8);
        SecretKeySpec secretKey = new SecretKeySpec(key, HMAC_ALGORITHM);
        Mac mac = Mac.getInstance(secretKey.getAlgorithm());
        mac.init(secretKey);
        return Hex.encodeHexString(mac.doFinal(text));
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}

/**
 * SHA-256
 *
 * @param str 待加密的报文

```

```

    */
    private String encod(String str) {
        MessageDigest messageDigest;
        String encdeStr = "";
        try {
            messageDigest = MessageDigest.getInstance(SHA_256);
            byte[] hash = messageDigest.digest(str.getBytes(StandardCharsets.
UTF_8));
            encdeStr = Hex.encodeHexString(hash);
        } catch (NoSuchAlgorithmException e) {
            System.out.println(String.format("Exception@encod: str=%s;",
str));
            e.printStackTrace();
            return null;
        }
        return encdeStr;
    }

    // AES 加解密算法
    private static final String IV = "543yhjy97ae7fyfg";
    private static final String TRANSFORM = "AES/CBC/PKCS5Padding";
    private static final String ALGORITHM = "AES";

    /**
     * key length = 16 bits
     */
    private byte[] encrypt(byte[] content, byte[] key) {
        return encrypt(content, key, IV);
    }

    /**
     * key length = 16 bits
     */
    private byte[] decrypt(byte[] content, byte[] key) {
        return decrypt(content, key, IV);
    }

    /**
     * aes 128 cbc key length = 16 bits
     */
    private byte[] encrypt(byte[] content, byte[] key, String ivContent) {
        try {
            SecretKeySpec keySpec = new SecretKeySpec(key, ALGORITHM);
            Cipher cipher = Cipher.getInstance(TRANSFORM);
            IvParameterSpec iv = new IvParameterSpec(ivContent.
getBytes(StandardCharsets.UTF_8));
            cipher.init(Cipher.ENCRYPT_MODE, keySpec, iv);
            return cipher.doFinal(content);
        } catch (Exception ex) {
            System.out.println(

```

```

        String.format("AES encrypt error, %s, %s, %s", content,
Hex.encodeHex(key), ex.getMessage());
        return null;
    }
}

/**
 * aes 128 cbc key length = 16 bits
 */
private byte[] decrypt(byte[] content, byte[] key, String ivContent) {
    try {
        SecretKeySpec keySpec = new SecretKeySpec(key, ALGORITHM);
        Cipher cipher = Cipher.getInstance(TRANSFORM);
        IvParameterSpec iv = new IvParameterSpec(ivContent.
getBytes(StandardCharsets.UTF_8));
        cipher.init(Cipher.DECRYPT_MODE, keySpec, iv);
        return cipher.doFinal(content);
    } catch (Exception ex) {
        System.out.println(String.format("AES decrypt error, %s, %s, %s",
Hex.encodeHex(content),
Hex.encodeHex(key), ex.getMessage()));
        return null;
    }
}

public static void main(String[] args) throws InterruptedException {
    IotCoapClientWithAes client = new IotCoapClientWithAes();
    client.conenct(productKey, deviceName, deviceSecret);
    client.publish(updateTopic, payload.getBytes(StandardCharsets.
UTF_8));
}
}

```

注意事项

1. coap 协议是短连接, 和 mqtt 长连接不同, 所以在控制台的设备行为日志里看不到记录, 需要注意。
2. coap 协议发送的 payload 有大小限制, 不能超过 1KB, 如果超过的话这个请求会被拒绝, 消息发不到平台。代码中 `System.out.println(request.getPayload().length);` 就是用来打印 payload 长度的。

HTTP 协议接入物联网平台 (Getman 模拟)

简介：本文将使用 Getman 模拟设备模拟 HTTP 请求，进行接入测试。

概述

阿里云物联网平台支持设备使用 HTTP 接入，目前仅支持 HTTPS 协议。本文将使用 [Getman](#) 模拟设备，进行接入测试。官方文档：[HTTP 连接通信](#)

说明与限制

- HTTP 通信方式适合单纯的数据上报的场景。
- HTTP 服务器地址为：<https://iot-as-http.cn-shanghai.aliyuncs.com>。
- 目前，仅中国（上海）地域支持 HTTP 通信。
- 只支持 HTTPS 协议和 POST 请求方式。
- Topic 规范和 MQTT 的 Topic 规范一致。使用 HTTP 协议连接，上报数据请求：[https://iot-as-http.cn-shanghai.aliyuncs.com/topic/ \\${topic}](https://iot-as-http.cn-shanghai.aliyuncs.com/topic/${topic})。其中，**Topic** 变量 `${topic}` 的值可以与 MQTT 连接通信的 Topic 相复用。不支持以 `?query_String=xxx` 格式传参。
- 数据上行接口传输的数据大小限制为 128 KB。
- 设备认证请求的 HTTP header 中的 Content-Type 必须为 application/json。
- 数据上报请求的 HTTP header 中的 Content-Type 必须为 application/octet-stream。
- 设备认证返回的 token 会在一定周期后失效（目前 token 有效期是 7 天），请务必考虑 token 失效逻辑的处理。

接入流程

认证设备，获取设备的 token

- endpoint 地址: <https://iot-as-http.cn-shanghai.aliyuncs.com>
- 认证设备请求

```
POST /auth HTTP/1.1Host: iot-as-http.cn-shanghai.aliyuncs.comContent-Type: application/jsonbody: {"version":"default","clientId":"mylight1000002", "signmethod":"hmacsha1", "sign":"487014", "productKey":"ZG1E", "deviceName":"NlwaS**", "timestamp":"1501668289957"}
```

- 请求参数说明

参数	说明
Method	请求方法。支持 POST 方法。
URL	/auth, URL 地址, 只支持 HTTPS。
Host	endpoint 地址: iot-as-http.cn-shanghai.aliyuncs.com
Content-Type	设备发送给物联网平台的上行数据的编码格式。目前只支持 application/json。若使用其他编码格式, 会返回参数错误。
body	设备认证信息。JSON 数据格式。具体信息, 请参见 body 说明

- 返回示例

```
body:
{
  "code": 0, // 业务状态码
  "message": "success", // 业务信息
  "info": {
    "token": "6944e5bfb92e4d4ea3918d1eda3942f6" // 需本地保存
  }
}
```

上报数据，发送数据到某个自定义 Topic

- 上报数据请求

```
POST /topic/${topic} HTTP/1.1
Host: iot-as-http.cn-shanghai.aliyuncs.com
password:${token}
Content-Type: application/octet-stream
body: ${your_data}
```


- 请求参数说明

参数	说明
Method	请求方法。支持 POST 方法。
URL	/topic/ topic 。其中，变量 {topic} 需替换为数据发往的目标 Topic。只支持 HTTPS。
Host	endpoint 地址: <code>iot-as-http.cn-shanghai.aliyuncs.com</code>
password	放在 Header 中的参数，取值为调用设备认证接口 auth 返回的 token 值。
Content-Type	设备发送给物联网平台的上行数据的编码格式。目前仅支持 <code>application/octet-stream</code> 。若使用其他编码格式，会返回参数错误。
body	发往 \${topic} 的数据内容。

- 返回示例

```
body:
{
  "code": 0, // 业务状态码
  "message": "success", // 业务信息
  "info": {
    "messageId": 892687627916247040,
  }
}
```

Getman 模拟过程

认证设备

- URL 和 Header



- body

Request

Body

```
{  
  "version": "default",  
  "clientId": "12345",  
  "signatureMethod": "hmacsha1",  
  "signature": "27F0C0B0E03084419D916EA44B232A399DB8F51",  
  "productKey": "a1[REDACTED]74u",  
  "deviceName": "MC[REDACTED]Device"  
}
```

sign 可以使用工具快速计算得到，请参考[此处](#) password 的计算

- 返回结果

Response

Body	Header
<pre>{ "code": 0, "info": { "token": "^1^1591773294379^3055f6955de1764" }, "message": "success" }</pre>	

上报数据

- 使用自定义 topic

设备信息

Topic列表

物模型数据

设备影子

文件管理

日志服务

在线调试

基础通信 Topic

物模型通信 Topic

自定义topic

自定义 Topic 列表

设备的Topic	设备具有的权限
/a1LhK374u/MQTTfx_device/user/update	发布
/a1LhK374u/MQTTfx_device/user/update/error	发布
/a1LhK374u/MQTTfx_device/user/get	订阅

- URL 和 Header

POST

https://iot-as-http.cn-shanghai.aliyuncs.com/topic/a1LhK374u/MQTTfx_device/user/upto

✓

Request

Body

Header

Content-Type

Host: iot-as-http.cn-shanghai.aliyuncs.com
Content-Type: application/octet-stream
password: **1591773294379*3055f6955de1764 ← token

- body

Request

Body

Header

Content-Type

```

{
  "abc":123
}

```

- 返回结果

Response

Body

Header

```
{
  "code": 0,
  "info": {
    "messageId": 1270616254155330600
  },
  "message": "success"
}
```

数据流转篇

服务端订阅之 AMQP

概述

服务端可以直接订阅产品下所有类型的消息：设备上报消息、设备状态变化通知、网关发现子设备上报、设备生命周期变更、设备拓扑关系变更。配置服务端订阅后，物联网平台会将产品下所有设备的已订阅类型的消息转发至您的服务端。

AMQP (Advanced Message Queuing Protocol) 即高级消息队列协议。您配置 AMQP 服务端订阅后，物联网平台会将产品下所有已订阅类型的消息，通过 AMQP 通道推送至您的服务端。

本文通过 MQTT.fx 模拟设备接入，[MQTT.fx 接入物联网平台](#)。通过 Java SDK 进行服务端订阅演示，[SDK 下载地址](#)。

消息流转示意图



AMQP 服务端订阅优势

- 支持多消费组。同一个账号，可以在开发环境下使消费组 A 订阅产品 A，同时在正式环境下使消费组 B 订阅产品 B。
- 方便排查问题。支持查看客户端状态、查看堆积和消费速率。
- 线性扩展。在消费者能力足够，即客户端机器足够的情况下，可轻松线性扩展推送能力。
- 实时消息优先推送，消息堆积不会影响服务。设备实时消息直接推送，推送限流或失败时进入堆积队列，堆积态消息采用降级模式，不会影响实时推送能力。即使消费者的客户端宕机，或因消费能力不足堆积了消息，消费端恢复后，设备生成的消息也可以和堆积消息并行发送，使设备优先恢复可用态。

操作流程

1. 在物联网平台的控制台上创建对应产品的 AMQP 服务端订阅，并勾选需要推送的消息类型。

编辑订阅

* 产品
MQTT.fx测试产品

* 订阅类型
AMQP

* 消费组
服务端订阅测试

* 推送消息类型

☒ 设备上报消息 ☒ 设备状态变化通知

☒ 设备生命周期变更 ☐ 物模型历史数据上报

☐ 固件升级状态通知

保存 取消

2. 填写相关参数，并启动消费端。[参数说明](#)

- 添加依赖如下：

```
<dependencies>
  <!-- amqp 1.0 qpid client -->
  <dependency>
    <groupId>org.apache.qpid</groupId>
    <artifactId>qpid-jms-client</artifactId>
    <version>0.47.0</version>
  </dependency>
  <!-- util for base64 -->
  <dependency>
    <groupId>commons-codec</groupId>
    <artifactId>commons-codec</artifactId>
    <version>1.10</version>
  </dependency>
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.22</version>
  </dependency>
  <!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-log4j12 -->
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.7.22</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-nop</artifactId>
    <version>1.7.2</version>
  </dependency>
</dependencies>
```

- 代码示例如下：

```
import java.net.URI;
import java.util.Hashtable;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.jms.MessageListener;
```

```

import javax.jms.MessageProducer;
import javax.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import org.apache.commons.codec.binary.Base64;
import org.apache.qpid.jms.JmsConnection;
import org.apache.qpid.jms.JmsConnectionListener;
import org.apache.qpid.jms.message.JmsInboundMessageDispatch;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class AmqpJavaClientDemo {

    private final static Logger logger = LoggerFactory.
        getLogger(AmqpJavaClientDemo.class);

    public static void main(String[] args) throws Exception {
        // 参数说明, 请参见上一篇文档: AMQP 客户端接入说明。
        String accessKey = "*****";
        String accessSecret = "*****";
        String consumerGroupId = "*****";
        long timeStamp = System.currentTimeMillis();
        // 签名方法: 支持 hmacmd5, hmacsha1 和 hmacsha256
        String signMethod = "hmacsha1";
        // 控制台服务端订阅中消费组状态页客户端 ID 一栏将显示 clientId 参数。
        // 建议使用机器 UUID、MAC 地址、IP 等唯一标识等作为 clientId。便于您区分识别不同的
        客户端。
        String clientId = "*****";

        //UserName 组装方法, 请参见上一篇文档: AMQP 客户端接入说明。
        String userName = clientId + "|authMode=aksign"
            + ",signMethod=" + signMethod
            + ",timestamp=" + timeStamp
            + ",authId=" + accessKey
            + ",consumerGroupId=" + consumerGroupId
            + "|";
        //password 组装方法, 请参见上一篇文档: AMQP 客户端接入说明。
        String signContent = "authId=" + accessKey + "&timestamp=" +
            timeStamp;
        String password = doSign(signContent, accessSecret, signMethod);
        // 按照 qpid-jms 的规范, 组装连接 URL。
        String connectionUrl = "failover:(amqps://${UID}.iot-amqp.cn-
            shanghai.aliyuncs.com:5671?amqp.idleTimeout=80000) "
            + "?failover.maxReconnectAttempts=10&failover.
            reconnectDelay=30";

        Hashtable<String, String> hashtable = new Hashtable<>();
        hashtable.put("connectionfactory.SBCF", connectionUrl);
        hashtable.put("queue.QUEUE", "default");
    }
}

```



```

        hashtable.put(Context.INITIAL_CONTEXT_FACTORY, "org.apache.qpid.jms.
jndi.JmsInitialContextFactory");
        Context context = new InitialContext(hashtable);
        ConnectionFactory cf = (ConnectionFactory)context.lookup("SBCF");
        Destination queue = (Destination)context.lookup("QUEUE");
        // Create Connection
        Connection connection = cf.createConnection(userName, password);
        ((JmsConnection) connection).
addConnectionListener(myJmsConnectionListener);
        // Create Session
        // Session.CLIENT_ACKNOWLEDGE: 收到消息后, 需要手动调用 message.
        acknowledge()
        // Session.AUTO_ACKNOWLEDGE: SDK 自动 ACK (推荐)
        Session session = connection.createSession(false, Session.AUTO_
ACKNOWLEDGE);
        connection.start();
        // Create Receiver Link
        MessageConsumer consumer = session.createConsumer(queue);
        consumer.setMessageListener(messageListener);
    }

    private static MessageListener messageListener = new MessageListener() {
        @Override
        public void onMessage(Message message) {
            try {
                byte[] body = message.getBody(byte[].class);
                String content = new String(body);
                String topic = message.getStringProperty("topic");
                String messageId = message.getStringProperty("messageId");
                logger.info("receive message"
                    + ", topic = " + topic
                    + ", messageId = " + messageId
                    + ", content = " + content);
                System.out.println("topic:" + topic);
                System.out.println("payload:" + content);
                System.out.println(" ");
                // 如果创建 Session 选择的是 Session.CLIENT_ACKNOWLEDGE, 这里需要手
                动 ACK。
                message.acknowledge();
                // 如果要对收到的消息做耗时的处理, 请异步处理, 确保这里不要有耗时逻辑。
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    };

    private static JmsConnectionListener myJmsConnectionListener = new
JmsConnectionListener() {
        /**

```

```

    * 连接成功建立。
    */
    @Override
    public void onConnectionEstablished(Uri remoteUri) {
        logger.info("onConnectionEstablished, remoteUri:{}, remoteUri;", remoteUri);
    }

    /**
     * 尝试过最大重试次数之后，最终连接失败。
     */
    @Override
    public void onConnectionFailure(Throwable error) {
        logger.error("onConnectionFailure, {}", error.getMessage());
    }

    /**
     * 连接中断。
     */
    @Override
    public void onConnectionInterrupted(Uri remoteUri) {
        logger.info("onConnectionInterrupted, remoteUri:{}, remoteUri;");
    }

    /**
     * 连接中断后又自动重连上。
     */
    @Override
    public void onConnectionRestored(Uri remoteUri) {
        logger.info("onConnectionRestored, remoteUri:{}, remoteUri;");
    }

    @Override
    public void onInboundMessage(JmsInboundMessageDispatch envelope) {}

    @Override
    public void onSessionClosed(Session session, Throwable cause) {}

    @Override
    public void onConsumerClosed(MessageConsumer consumer, Throwable
cause) {}

    @Override
    public void onProducerClosed(MessageProducer producer, Throwable
cause) {}
};

/**
 * password 签名计算方法，请参见上一篇文档：AMQP 客户端接入说明。
 */

```

```

private static String doSign(String toSignString, String secret, String
signMethod) throws Exception {
    SecretKeySpec signingKey = new SecretKeySpec(secret.getBytes(),
signMethod);
    Mac mac = Mac.getInstance(signMethod);
    mac.init(signingKey);
    byte[] rawHmac = mac.doFinal(toSignString.getBytes());
    return Base64.encodeBase64String(rawHmac);
}
}

```

3. 启动消费端后，在控制台进行查看

← 服务端订阅测试 编辑

消费组 ID: HlWYKaswVcqpU3LmyC8000100 复制 创建时间: 2020/06/21 16:02:55

消费组状态: 订阅产品

基本信息 重置配置

消息消费速率: 0 条/分钟 消息堆积量: 0 条 最近消费时间: -

在线客户端列表

客户端 ID	客户端 IP/Port	最后上线时间
sanfeng-test	10.10.10.170:233-24189	2020/06/21 19:02:21.231

4. 设备上线并上报一条消息

IoT-Test Connect Disconnect

Publish Subscribe Scripts Broker Status Log

» /a1l...MQTTfx_device/user/update Publish

{ "test": "abc" }

服务端订阅之 MNS

简介：本文结合物联网平台最新推出的独享实例，在新的实例下面创建产品及设备，进行历史属性的上报测试，并进行 MNS 历史属性服务端订阅。

概述

阿里云物联网平台不仅支持设备即时属性的上报，也支持因为某些原因，没有及时上报的属性数据，通过历史属性上报的方式进行上报，历史属性上报 Topic: /sys/{productKey}/{deviceName}/thing/event/property/history/post。本文结合物联网平台最新推出的独享实例，在新的实例下面创建产品及设备，进行历史属性的上报测试，并进行 MNS 历史属性服务端订阅。

操作步骤

1. 创建独享实例，在独享实例下面创建产品和设备





2. 获取独享实例设备端 MQTT 接入点



3. 设备端接入, [参考链接](#)

```
import com.alibaba.taro.AliyunIoTSignUtil;
import org.eclipse.paho.client.mqttv3.*;
```

```

import org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;
import java.util.HashMap;
import java.util.Map;

public class IoTDemoPubSubDemo {

    public static String productKey = "g028S*****";
    public static String deviceName = "device1";
    public static String deviceSecret = "aKEHNK1w0UBvGcAlBA7gf2eHC*****";
    public static String regionId = "cn-shanghai";
    public static String instanceId = "iot-instc-public-cn-0pp1g*****";

    // 物模型 - 历史属性上报 topic
    private static String pubTopic = "/sys/" + productKey + "/" + deviceName
+ "/thing/event/property/history/post";

    private static MqttClient mqttClient;

    public static void main(String [] args){

        initAliyunIoTClient();
        postDeviceProperties();
    }

    /**
     * 初始化 Client 对象
     */
    private static void initAliyunIoTClient() {

        try {
            // 构造连接需要的参数
            String clientId = "java" + System.currentTimeMillis();
            Map<String, String> params = new HashMap<>(16);
            params.put("productKey", productKey);
            params.put("deviceName", deviceName);
            params.put("clientId", clientId);
            String timestamp = String.valueOf(System.currentTimeMillis());
            params.put("timestamp", timestamp);
            // MQTT 设备接入 公网终端节点 (Endpoint)
            String targetServer = "tcp://" + instanceId + ".iot-as-
mqtt."+regionId+".iothub.aliyuncs.com:1883";

            String mqttclientId = clientId +
"|securemode=3,signmethod=hmacsha1,timestamp=" + timestamp + "|";
            String mqttUsername = deviceName + "&" + productKey;
            String mqttPassword = AliyunIoTSignUtil.sign(params,
deviceSecret, "hmacsha1");

            connectMqtt(targetServer, mqttclientId, mqttUsername,
mqttPassword);

```

```

        } catch (Exception e) {
            System.out.println("initAliyunIoTClient error " +
e.getMessage());
        }
    }

    public static void connectMqtt(String url, String clientId, String
mqttUsername, String mqttPassword) throws Exception {

        MemoryPersistence persistence = new MemoryPersistence();
        mqttClient = new MqttClient(url, clientId, persistence);
        MqttConnectOptions connOpts = new MqttConnectOptions();
        // MQTT 3.1.1
        connOpts.setMqttVersion(4);
        connOpts.setAutomaticReconnect(false);
//        connOpts.setCleanSession(true);
        connOpts.setCleanSession(false);

        connOpts.setUserName(mqttUsername);
        connOpts.setPassword(mqttPassword.toCharArray());
        connOpts.setKeepAliveInterval(60);

        mqttClient.connect(connOpts);
    }

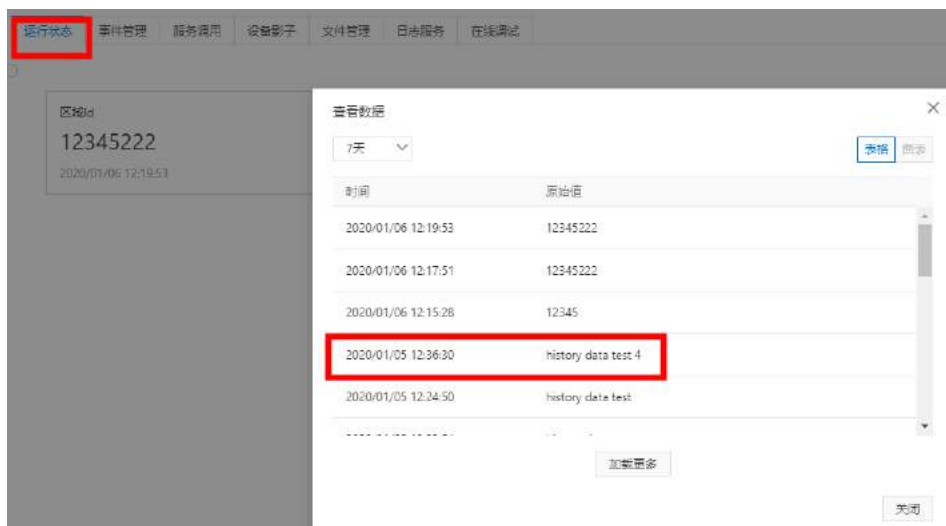
    /**
     * 汇报属性
     */
    private static void postDeviceProperties() {

        try {
            // 上报数据
            // 高级版 物模型 - 属性上报 payload
            System.out.println("历史上报属性值");
            String payloadJson = "{ \"id\": 123, \"version\": \"1.0\",
\"method\": \"thing.event.property.history.post\", \"params\": [ {
\"identity\": { \"productKey\": \"g028S*****\", \"deviceName\": \"device1\"
}, \"properties\": [ { \"AreaId\": { \"value\": \"history data test 4\",
\"time\": 1578198990000 } } ] ] }";
            MqttMessage message = new MqttMessage(payloadJson.
getBytes("utf-8"));
            message.setQos(1);
            mqttClient.publish(pubTopic, message);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}

```


payload 格式[参考](#)。

4. 设备运行状态查看

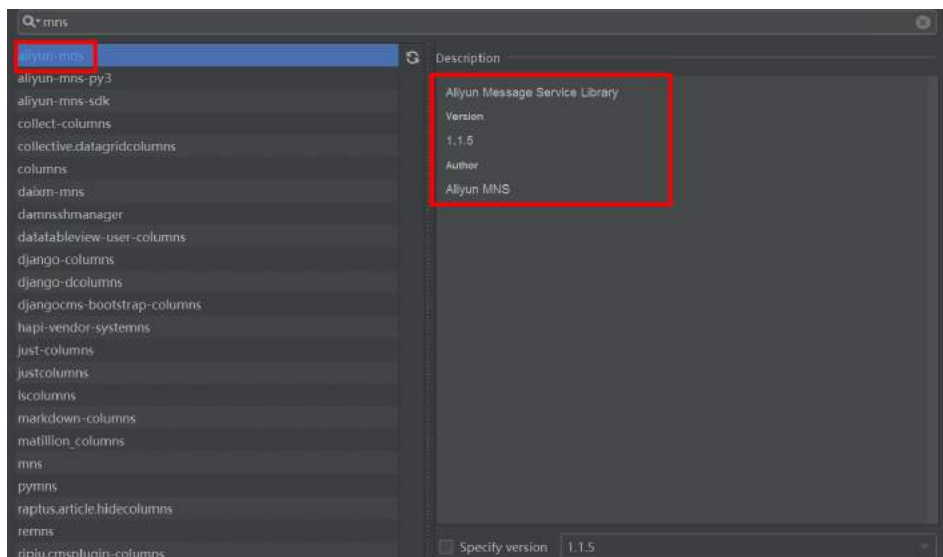


5. MNS 服务端订阅配置



6. 使用 MNS Python SDK 获取 MNS 订阅到 Queue 中的消息

6.1 安装 SDK



6.2 Code Sample

```
#init my_account, my_queue
from mns.account import Account
import sys
import json
import base64

endpoint = "http://18482178*****.mns.cn-shanghai.aliyuncs.com/"
accid = "LTAIOZZg*****"
acckey = "v7CjUJCMk7j9aKduMAQLjy*****"
token = ""
my_account = Account(endpoint, accid, acckey, token)
queue_name = "aliyun-iot-g028S*****"
my_queue = my_account.get_queue(queue_name)

# 循环读取删除消息直到队列空
#receive message 请求使用 long polling 方式, 通过 wait_seconds 指定长轮询时间为 3 秒

## long polling 解析:
### 当队列中有消息时, 请求立即返回;
### 当队列中没有消息时, 请求在 MNS 服务器端挂 3 秒钟, 在这期间, 有消息写入队列, 请求会立即返回消息, 3 秒后, 请求返回队列没有消息;
```

```

wait_seconds = 3
print("%sReceive And Delete Message From Queue%s\nQueueName:%s\nWaitSeconds:%s\n" % (10*"=", 10*"=", queue_name, wait_seconds))
while True:
    # 读取消息
    try:
        recv_msg = my_queue.receive_message(wait_seconds)
        print("Receive Message Succeed! ReceiptHandle:%s MessageBody:%s\nMessageID:%s" % (recv_msg.receipt_handle, recv_msg.message_body, recv_msg.message_id))
        print("Message Body: ", base64.b64decode(json.loads(recv_msg.message_body)['payload'])) # 转发到mns的messagebody经过了base64编码, 获取具体消息的内容需要做base64解码
    except Exception as e:
        #except MNSServerErrorException as e:
        if e.type == u"QueueNotExist":
            print("Queue not exist, please create queue before receive message.")
            sys.exit(0)
        elif e.type == u"MessageNotExist":
            print("Queue is empty!")
            sys.exit(0)
        print("Receive Message Fail! Exception:%s\n" % e)
        continue

    # 删除消息
    try:
        my_queue.delete_message(recv_msg.receipt_handle)
        print("Delete Message Succeed! ReceiptHandle:%s" % recv_msg.receipt_handle)
    except Exception as e:
        print("Delete Message Fail! Exception:%s\n" % e)
print("Delete Message Fail! Exception:%s\n" % e)

```

6.3 Test Result

```

Receive Message Succeed! ReceiptHandle:8-2zuDHj20LzZz8zcFz0z6YEzcSFqxyIKefW
MessageBody:{'payload': 'eyJkZXZpY2VUeXB1IjoiaQ3VzdG9tQ2F0ZWdvcnkiLCJpb3RJRZCI6I1Y1WGJXekluY0EyOW41aWNib3RYZzZyODAwIiwicmVxdWVzdElkIjoimTIzIiwicHJvZHVhdEtleSI6I1mcwMjhhTRlo4RlJDIiwiaWZ210Q3JlYXRlIjoxNTc4MjknZi00OTk4MDAwLCJ2YWx1ZSI6Imhpc3RvcnkzZGF0YSB0ZXN0*****', 'messageType': 'thing_history', 'topic': '/g028S*****/device1/thing/event/property/history/post', 'messageid': '1214069604465248256', 'timestamp': '1578291724'} MessageID:5F8092E53A67656C7F811CD50E19A150Message Body: b'{"deviceType": "CustomCategory", "deviceId": "V5XbWzIncA29n5icbo*****", "requestId": "123", "productKey": "g028S*****", "gmtCreate": "1578291724668", "deviceName": "device1", "items": {"AreaId": {"time": "1578198990000", "value": "history data test 4"}}}'
DEBUG: v7CjUJCMk7j9aKduMAQLjyCmb8cmCm hAbKzezvRlqeVXC18CzChL4OZZk= DELETE

```

更多参考

[使用 MNS 服务端订阅](#)

[Python SDK 队列使用手册](#)

服务端订阅的 4 步排查流程

概述

服务端可以直接订阅产品下所有类型的消息：设备上报消息、设备状态变化通知、网关发现子设备上报、设备生命周期变更、设备拓扑关系变更。配置服务端订阅后，物联网平台会将产品下所有设备的已订阅类型的消息转发至您的服务端。

注意事项

- 消息流转链路



- 对于老的 HTTP2 订阅，建议尽快更新至 AMQP 订阅

排查过程

一、检查控制台相应参数配置

1. 相应产品订阅类型（AMQP 还是 MNS）

- 如果是 AMQP 订阅，检查这个产品选择的消费组（例如：如果只选择了消费组 A，但是服务端代码里用的是消费组 B 的消费组 ID，自然订阅不到消息）。

×

已选择 1 个消费组

- MNS 订阅不涉及消费组概念，检查产品对应的队列是否正常创建即可。

数据源	数据源名称	数据源类型	数据源地址	数据源描述	数据源状态	操作
数据源	alyun-iot-productKey	MNS	2019/08/30 14:38:10	编辑	删除	
数据源	alyun-iot-productKey	MNS	2019/08/07 20:36:37	编辑	删除	
数据源	alyun-iot-productKey	AMQP	2019/08/06 10:09:08	编辑	删除	

2. 勾选的推送消息类型

- 如果只勾选了设备状态变化通知，那服务端自然订阅不到“设备上报消息”了。

* 推送消息类型

☒ 设备上报消息 ?

☒ 设备状态变化通知

☐ 设备生命周期变更 ?

☐ 物模型历史数据上报

☐ 固件升级状态通知 ?

取消

二、检查上报的数据格式

- 大多数情况都是设备端进行属性上报，但是在服务端订阅不到消息。此时看一下控制台上的日志，找到物模型上报的相关日志，检查物模型解析是否正常。如果物模型解析失败，该条消息是不会推送到服务端的。



时间	TraceID	MessageID	DeviceName	业务消息(空部)	操作	内容	状态
2020/06/21 19:27:34.801	0a30265d15927388547 996391d20be	1274665260 095440906	MQTTfx_device	设备到云消息	/sys/thing/model57du/ MQTTfx_device...	<?Content="Publish message to..."	200
2020/06/21 19:27:34.823	0a30265d15927388547 996391d20be	1274665260 095440906	MQTTfx_device	物模型上报	thing.event.property post	{"Reason":"request parameter error"}	400
2020/06/21 19:27:34.823	0a30265d15927388547 996391d20be	-	MQTTfx_device	物模型上报	Check	{"Reason":"request parameter error"}	400

三、检查是不是开启了多个客户端进行订阅

- 如果一和二都检查无误，需要检查一下是不是启动了多个客户端。



四、提交工单，提供相应信息

如果上述排查都无法定位问题，请提供以下信息：

- 设备三元组信息。
- 消息的 messageID。

3. 如果是设备上下线的消息，提供设备上下线的日志截图（也就是日志里的设备行为）。
4. MNS 订阅实际上是规则引擎的数据流转，提供上行消息分析里（Transmit to MNS……）日志的相关内容。

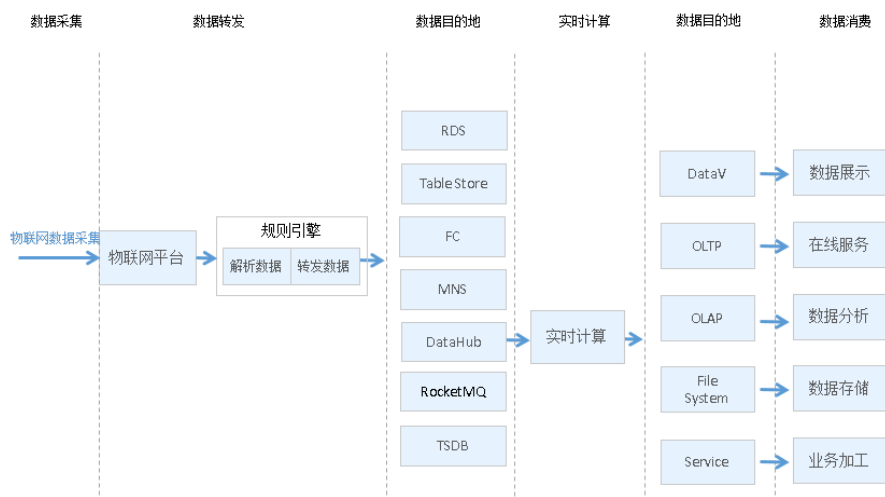
7 个数据流转实例详解规则引擎

简介：针对官方文档目前这部分示例介绍的比较简单，数据转发部分介绍的较少，本文对规则引擎常见的使用场景做一个整理，方便使用者结合自己的业务场景参考。

概述

使用物联网平台规则引擎的数据流转功能，当设备基于 Topic 进行通信时，您可以在规则引擎的数据流转中，编写 SQL 对 Topic 中的数据进行处理，并配置转发规则将处理后的数据转发到其他设备 Topic 或阿里云其他服务。针对官方文档目前这部分示例介绍的比较简单，数据转发部分介绍的较少，本文对规则引擎常见的使用场景做一个整理，方便使用者结合自己的业务场景参考。

原理图



数据流转实例

[阿里云物联网平台数据转发到 DataHub 示例](#)

[阿里云物联网平台数据转发到函数计算示例](#)

[阿里云物联网平台设备上下线信息通过规则引擎流转到 RDS 示例](#)

[阿里云物联网平台设备数据转发到消息队列 RocketMQ 全链路测试](#)

[阿里云物联网平台数据转发到表格存储 \(Table Store\) 示例参考](#)

[阿里云物联网平台数据转发到消息服务 \(MNS\) 示例](#)

[阿里云物联网平台数据转发到时序时空数据库 \(TSDB\) 示例](#)

参考链接

[云产品流转概述](#)

[SQL 表达式](#)

[数据流转使用示例](#)

基于 Topic 消息路由构建 M2M 设备间通信架构

简介：本章节以 Node JS SDK 为例，使用基于 Topic 消息路由的 M2M 设备间通信，主要介绍如何基于物联网平台构建一个 M2M 设备间通信架构。

概述

M2M (即 Machine-to-Machine) 是一种端对端通信技术。本章节以 Node JS SDK 为例，使用基于 Topic 消息路由的 M2M 设备间通信，主要介绍如何基于物联网平台构建一个 M2M 设备间通信架构。

实验步骤

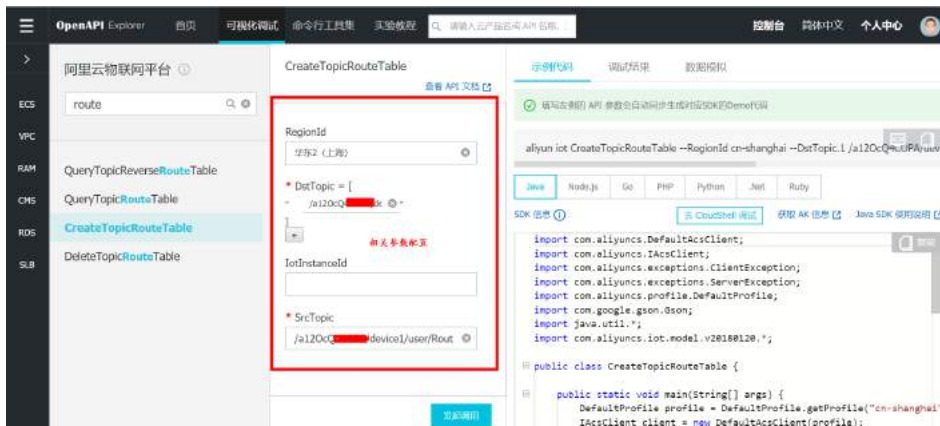
第一部分：配置相关

1. 产品、设备、Topic 的创建参考[链接](#)

消息路由建立

本部分目前不支持门户直接配置，需要基于管理 API: [CreateTopicRouteTable](#) 来建立消息路由关系。

测试可以直接使用 [OpenAPI](#) 来快速实现相关功能，本地集成相关功能直接基于 SDK 即可。



2. JAVA SDK Demo

```
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.IAcsClient;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.exceptions.ServerException;
import com.aliyuncs.iot.model.v20170420.CreateTopicRouteTableRequest;
import com.aliyuncs.iot.model.v20170420.CreateTopicRouteTableResponse;
import com.aliyuncs.profile.DefaultProfile;
import com.google.gson.Gson;
import java.util.*;

public class CreateTopicRouteTable {

    public static void main(String[] args) {
        DefaultProfile profile = DefaultProfile.getProfile("cn-shanghai",
"LTAI0ZZg*****", "v7CjUJCMk7j9aKduMAQLjy*****");
        IAcsClient client = new DefaultAcsClient(profile);

        CreateTopicRouteTableRequest request = new
CreateTopicRouteTableRequest();
        request.setRegionId("cn-shanghai");

        List<String> dstTopicList = new ArrayList<String>();
        dstTopicList.add("/a120cQ4****/device2/user/RouteData");
        request.setDstTopics(dstTopicList);
        request.setSrcTopic("/a120cQ4****/device1/user/RouteData");

        try {
            CreateTopicRouteTableResponse response = client.
getAcsResponse(request);
            System.out.println(new Gson().toJson(response));
        } catch (ServerException e) {
            e.printStackTrace();
        } catch (ClientException e) {
```

```

        System.out.println("ErrCode:" + e.getErrCode());
        System.out.println("ErrMsg:" + e.getErrMsg());
        System.out.println("RequestId:" + e.getRequestId());
    }
}
}

```

注意：SDK 版本差异按照实际版本调整即可。

3. 查询路由关系

```

import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.IAcsClient;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.exceptions.ServerException;
import com.aliyuncs.iot.model.v20170420.QueryTopicRouteTableRequest;
import com.aliyuncs.iot.model.v20170420.QueryTopicRouteTableResponse;
import com.aliyuncs.profile.DefaultProfile;
import com.google.gson.Gson;

public class QueryTopicRouteTable {

    public static void main(String[] args) {
        DefaultProfile profile = DefaultProfile.getProfile("cn-shanghai",
            "LTAIOZZgY*****", "v7CjUJCMk7j9aKduMAQLjy*****");
        IAcsClient client = new DefaultAcsClient(profile);

        QueryTopicRouteTableRequest request = new
            QueryTopicRouteTableRequest();
        request.setRegionId("cn-shanghai");
        request.setTopic("/a120cQ4****/device1/user/RouteData");

        try {
            QueryTopicRouteTableResponse response = client.
                getAcsResponse(request);
            System.out.println(new Gson().toJson(response));
        } catch (ServerException e) {
            e.printStackTrace();
        } catch (ClientException e) {
            System.out.println("ErrCode:" + e.getErrCode());
            System.out.println("ErrMsg:" + e.getErrMsg());
            System.out.println("RequestId:" + e.getRequestId());
        }
    }
}

```

运行结果

```

{"requestId":"9404FD71-7461-478E-B064-0AEB15C91111","success":true,"dstTopic":["/a120cQ4****/device2/user/RouteData"]}

```

第二部分：客户端代码相关

4. Node JS SDK 安装参考[链接](#)

5. 设备端业务代码

device1

```
// node 引入包名
const iot = require('alibabacloud-iot-device-sdk');
// 浏览器、微信小程序，支付宝小程序引入 ./dist 编译的 js 文件
// const iot = require('./dist/alibabacloud-iot-device-sdk.js');
// js 版本下载地址：
//   https://github.com/aliyun/alibabacloud-iot-device-sdk/tree/master/dist
或
//   alibabacloud-iot-device-sdk.js 下载地址 https://unpkg.com/alibabacloud-iot-device-sdk@1.2.4/dist/alibabacloud-iot-device-sdk.js 或
//   alibabacloud-iot-device-sdk.min.js 下载地址 https://unpkg.com/alibabacloud-iot-device-sdk@1.2.4/dist/alibabacloud-iot-device-sdk.min.js
//

const device = iot.device({
  productKey: 'al20cQ4****',
  deviceName: 'device1',
  deviceSecret: '3yWqKtWxN7VPuWEEDEn4eKWN*****',
  // 支付宝小程序和微信小程序额外需要配置协议参数
  // "protocol": 'alis://', "protocol": 'wxs://',
});
device.on('connect', () => {
  console.log('connect successfully!');
  // 发送消息到指定的 Topic，等待规则引擎转发到另外的一个设备的 Topic
  device.publish('/al20cQ4****/device1/user/RouteData', '{"key1": "value1 test"}');
  device.publish('/al20cQ4****/device1/user/RouteData', 'This is my test job.');
```

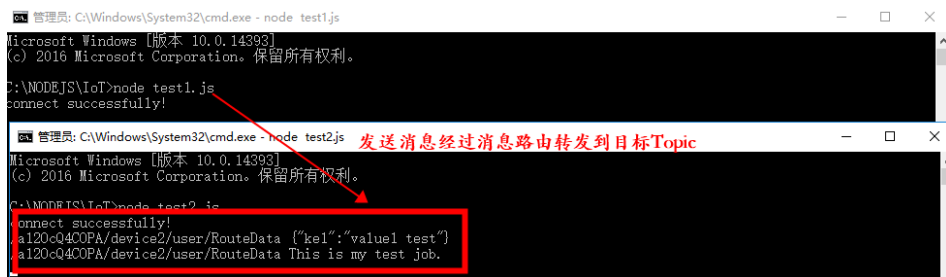
device2

```
// node 引入包名
const iot = require('alibabacloud-iot-device-sdk');
// 浏览器、微信小程序，支付宝小程序引入 ./dist 编译的 js 文件
// const iot = require('./dist/alibabacloud-iot-device-sdk.js');
// js 版本下载地址：
//   https://github.com/aliyun/alibabacloud-iot-device-sdk/tree/master/dist
或
```

```
// alibabacloud-iot-device-sdk.js 下载地址 https://unpkg.com/alibabacloud-
iot-device-sdk@1.2.4/dist/alibabacloud-iot-device-sdk.js 或
// alibabacloud-iot-device-sdk.min.js 下载地址 https://unpkg.com/
alibabacloud-iot-device-sdk@1.2.4/dist/alibabacloud-iot-device-sdk.min.js
//

const device = iot.device({
  productKey: 'al20cQ4****',
  deviceName: 'device2',
  deviceSecret: 'X9fzX9u0aIOORNghPyfYKq22*****'
  // 支付宝小程序和微信小程序额外需要配置协议参数
  // "protocol": 'alis://', "protocol": 'wxs://',
});
// 定于规则引擎转发过来的消息
device.subscribe('/al20cQ4****/device2/user/RouteData');
device.on('connect', () => {
  console.log('connect successfully!');
});
device.on('message', (topic, payload) => {
  console.log(topic, payload.toString());
});
```

8. 测试运行



参考链接

[基于 Topic 消息路由的 M2M 设备间通信](#)

基于规则引擎构建 M2M 设备间通信架构

简介：本章节以 Node JS SDK 为例，使用规则引擎数据流转来实现 M2M 设备间通信，主要介绍如何基于物联网平台构建一个 M2M 设备间通信架构。

概述

M2M (即 Machine-to-Machine) 是一种端对端通信技术。本章节以 Node JS SDK 为例，使用规则引擎数据流转来实现 M2M 设备间通信，主要介绍如何基于物联网平台构建一个 M2M 设备间通信架构。

实验步骤

第一部分：配置相关

1. 创建产品

节点类型

* 节点类型

☒ 设备 ☐ 网关

* 是否接入网关

☐ 是 ☒ 否

连网与数据

* 连网方式

WiFi

* 数据格式

ICA 标准数据格式 (Alink JSON)

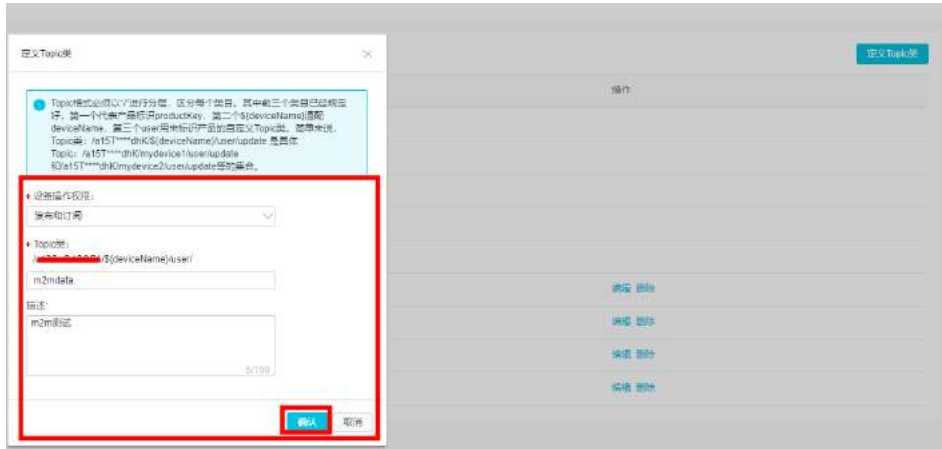
* 使用 ID² 认证

☐ 是 ☒ 否

更多信息

使用文档

完成 取消



2. 产品下面添加设备 1 和设备 2



3. 规则引擎配置



4. 启动规则引擎



第二部分：客户端代码相关

5. Node JS 设备端 SDK 安装

```
npm install alibabacloud-iot-device-sdk --save
```

6. 设备端业务代码

设备 1

```
// node 引入包名
const iot = require('alibabacloud-iot-device-sdk');
// 浏览器、微信小程序，支付宝小程序引入 ./dist 编译的 js 文件
// const iot = require('./dist/alibabacloud-iot-device-sdk.js');
// js 版本下载地址：
//   https://github.com/aliyun/alibabacloud-iot-device-sdk/tree/master/dist
// 或
//   alibabacloud-iot-device-sdk.js 下载地址 https://unpkg.com/alibabacloud-iot-device-sdk@1.2.4/dist/alibabacloud-iot-device-sdk.js 或
//   alibabacloud-iot-device-sdk.min.js 下载地址 https://unpkg.com/alibabacloud-iot-device-sdk@1.2.4/dist/alibabacloud-iot-device-sdk.min.js
//

const device = iot.device({
  productKey: 'al20cQ4****',
  deviceName: 'device1',
  deviceSecret: '3yWqKtWxN7VPuWEEDEn4eKWNNTUv****'
  // 支付宝小程序和微信小程序额外需要配置协议参数
  // "protocol": 'alis://', "protocol": 'wxs://',
});

device.on('connect', () => {
  console.log('connect successfully!');
  // 发送消息到指定的 Topic，等待规则引擎转发到另外的一个设备的 Topic
  device.publish('/al20cQ4****/device1/user/m2mdata', '{"ke1": "value1 test"}');
});
```

设备 2

```
// node 引入包名
const iot = require('alibabacloud-iot-device-sdk');
// 浏览器、微信小程序，支付宝小程序引入 ./dist 编译的 js 文件
// const iot = require('./dist/alibabacloud-iot-device-sdk.js');
// js 版本下载地址：
//   https://github.com/aliyun/alibabacloud-iot-device-sdk/tree/master/dist
```

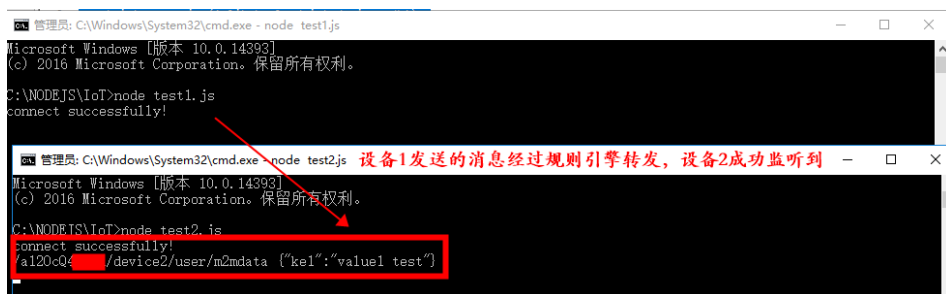
```

或
// alibabacloud-iot-device-sdk.js 下载地址 https://unpkg.com/alibabacloud-
iot-device-sdk@1.2.4/dist/alibabacloud-iot-device-sdk.js 或
// alibabacloud-iot-device-sdk.min.js 下载地址 https://unpkg.com/
alibabacloud-iot-device-sdk@1.2.4/dist/alibabacloud-iot-device-sdk.min.js
//

const device = iot.device({
  productKey: 'a120cQ4****',
  deviceName: 'device2',
  deviceSecret: 'X9fzX9u0aIOORNghPyfYKq22IL7Q****'
  // 支付宝小程序和微信小程序额外需要配置协议参数
  // "protocol": 'alis://', "protocol": 'wxs://',
});
// 定于规则引擎转发过来的消息
device.subscribe('/a120cQ4****/device2/user/m2mdata');
device.on('connect', () => {
  console.log('connect successfully!');
});
device.on('message', (topic, payload) => {
  console.log(topic, payload.toString());
});

```

7. 测试运行



第三部分：问题排查

对于使用过程中设备 2 无法正常监听到消息的情况，可以结合运维监控的日志服务进行排查。



参考链接

基于规则引擎的 M2M 设备间通信

Node JS SDK 环境要求与配置

高级功能篇

8 步详解固件升级流程演示

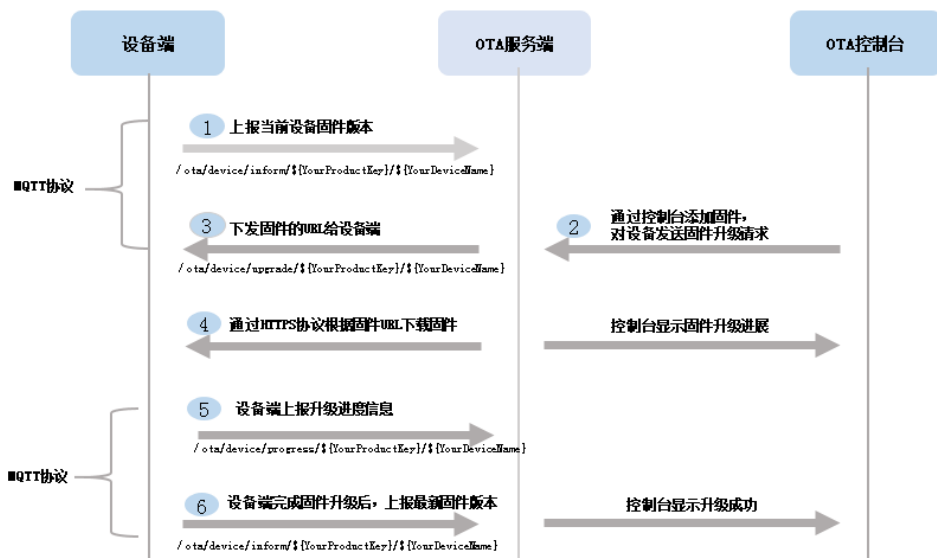
概述

OTA (Over-the-Air Technology) 即空中下载技术。阿里云物联网平台支持通过 OTA 方式进行设备固件升级。本文以 MQTT 协议下的固件升级为例，介绍 OTA 固件升级流程、数据流转使用的 Topic 和数据格式。本文使用 MQTT.fx 客户端模拟设备，进行固件升级流程的操作演示。[MQTT.fx 接入物联网平台](#)

官方文档：[固件升级](#)

OTA 固件升级流程

1. 流程图如下：



2. 相关 Topic 如下：

- 设备端通过以下 Topic 上报固件版本给物联网平台。

```
/ota/device/inform/${YourProductKey}/${YourDeviceName}
```

- 设备端订阅以下 Topic 接收物联网平台的固件升级通知。

```
/ota/device/upgrade/${YourProductKey}/${YourDeviceName}
```

- 设备端通过以下 Topic 上报固件升级进度。

```
/ota/device/progress/${YourProductKey}/${YourDeviceName}
```

操作演示

1. 在物联网平台控制台的 " 监控运维 "—" 固件升级 " 中添加固件。

添加固件



* 固件类型 ?

整包

差分

* 固件名称 ?

演示固件

* 所属产品

MQTTfx测试产品

* 固件模块

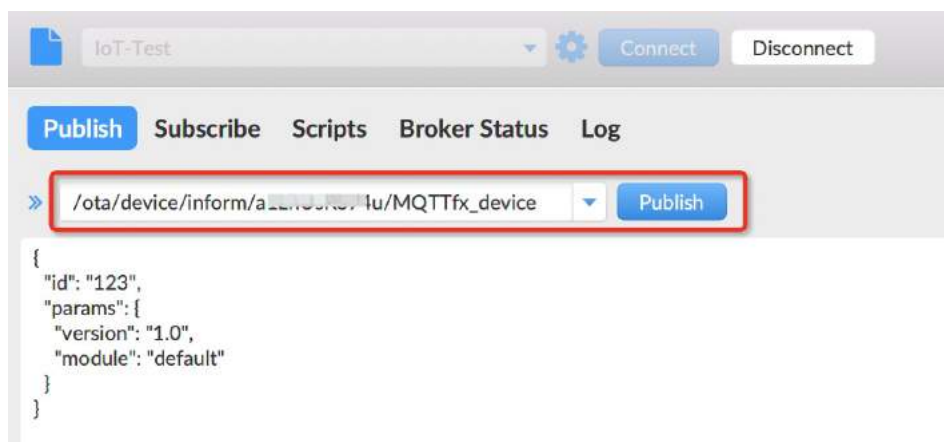
☒ 选择模块 ☐ 新增模块 ?

default

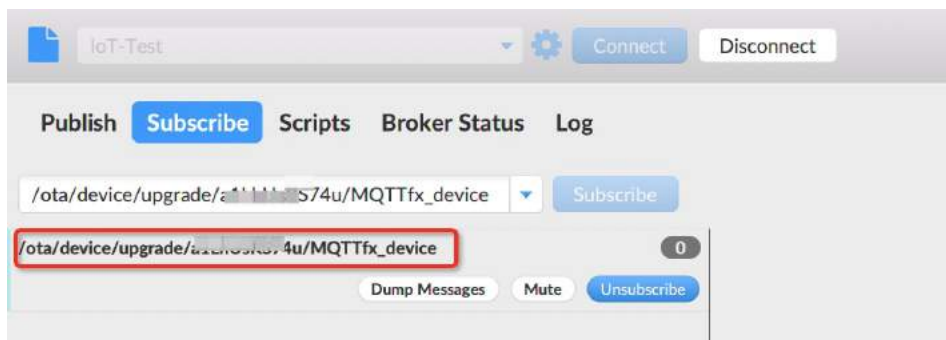
* 固件版本号 ?

2.0

2. 设备端通过 MQTT 协议推送当前设备固件版本号到 Topic: `/ota/device/inform/${YourProductKey}/${YourDeviceName}`。消息格式



3. 设备端订阅 Topic: `/ota/device/upgrade/${YourProductKey}/${YourDeviceName}`。控制台对设备发起固件升级请求后，设备端会通过该 Topic 收到固件的存储地址 URL。



4. 控制台触发升级操作。

验证固件

×

* 待升级版本号

1.0 ×

▽

* 待验证设备

MQTTfx_device ×

▽

设备升级超时时间（分钟） ?

10

确定

取消

- 根据版本号来判断设备端 OTA 升级是否成功。
- 设备离线时，不能接收服务端推送的升级消息。通过 MQTT 协议接入物联网平台的设备再次上线后，物联网平台系统自动检测到设备上线，OTA 服务端验证该设备是否需要升级。如果需要升级，再次推送升级消息给设备，否则，不推送消息。
- 设备需在固件 URL 下发后的 24 小时内下载固件，否则该 URL 失效。
- 升级成功的唯一判断标志是设备上报正确的版本号。即使升级进度上报为 100%，如果不上报新固件版本号，也视为升级失败。

远程配置的 2 种场景演示

简介：使用远程配置功能，可在不用重启设备或中断设备运行情况下，在线远程更新设备的系统参数、网络参数等配置信息。本文使用 MQTT.fx 客户端模拟设备，进行远程配置两种场景的演示。

概述

使用远程配置功能，可在不用重启设备或中断设备运行情况下，在线远程更新设备的系统参数、网络参数等配置信息。本文使用 MQTT.fx 客户端模拟设备，进行远程配置两种场景的演示。[MQTT.fx 接入物联网平台](#)

官方文档：[远程配置](#)

与固件升级对比

很多场景下，开发者需要更新设备的配置信息，包括设备的系统参数、网络参数、本地策略等。通常情况下，是通过固件升级更新设备的配置信息。但是，这将加大固件版本的维护工作，并且需要设备中断运行以完成更新。为了解决上述问题，物联网平台提供远程配置更新功能，设备无需重启或中断运行即可在线完成配置信息更新。

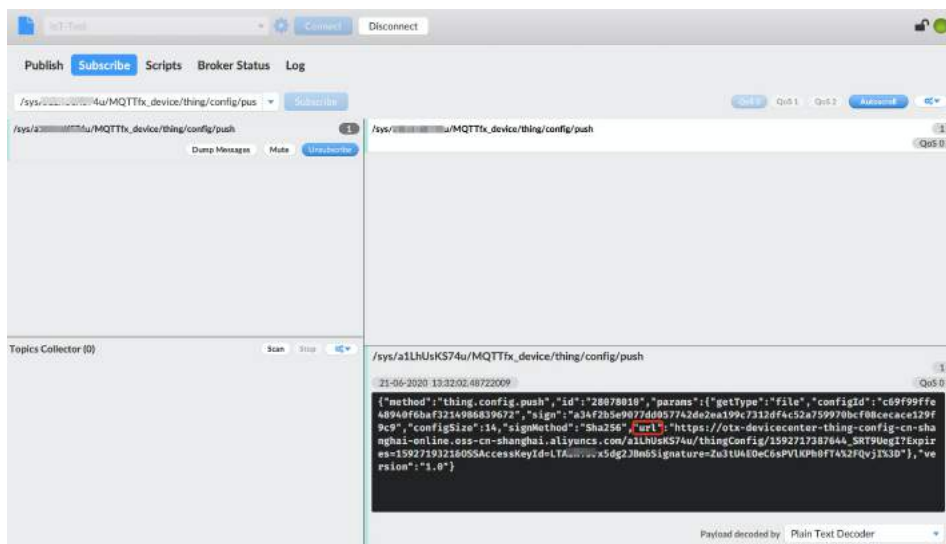
前提条件

- 已在物联网平台控制台开通远程配置服务。如果未开通，登录物联网平台的控制台，选择监控运维 > 远程配置，然后单击开通服务。
- 设备端 SDK 已开启支持远程配置服务。（特指设备端 C SDK）。需要在设备端 SDK 中定义 `FEATURE_SERVICE_OTA_ENABLED = y`。SDK 提供接口 `linkkit_cota_init`，用于初始化远程配置（Config Over The Air, COTA）。

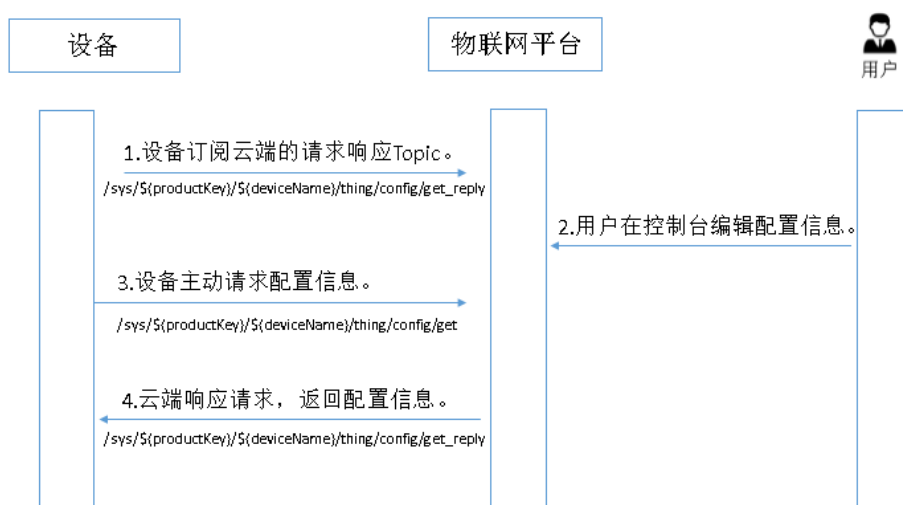
2. 在控制台上打开产品的远程配置开关，并进行批量更新。



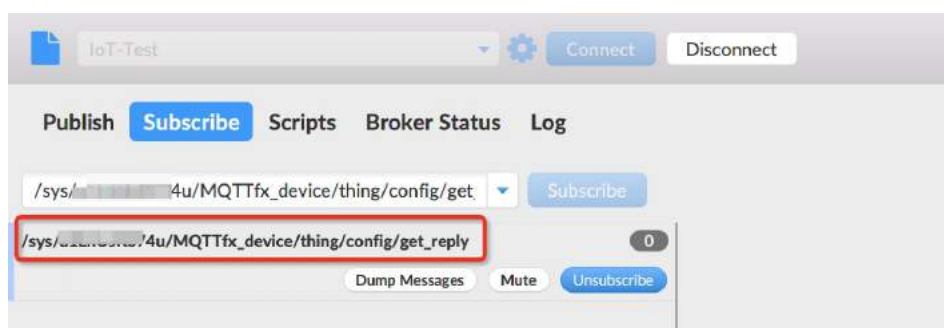
3. 设备端接收云端下发的配置文件下载链接后，自行更新配置。



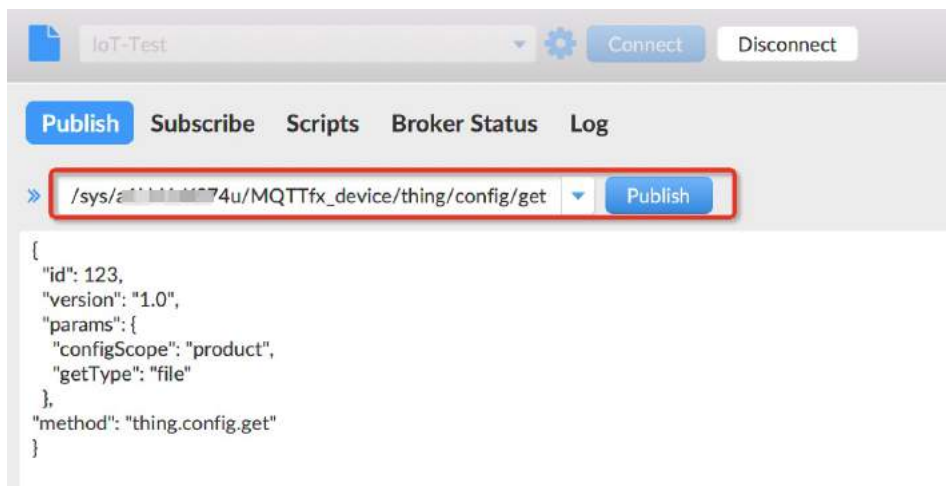
场景二：设备主动请求配置信息



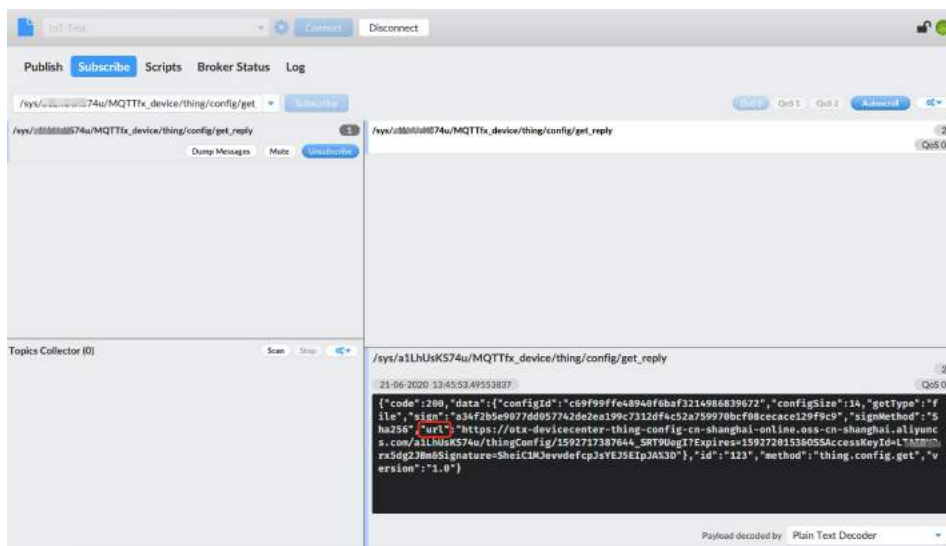
1. 设备端上线并订阅云端响应设备请求配置信息的 Topic: `/sys/${productKey}/${deviceName}/thing/config/get_reply`。



2. 在物联网平台控制台中，开启远程配置，编辑配置信息。详细步骤请参见以上场景一的第 2 步，但是不进行批量更新。
3. 设备通过 Topic: `/sys/${productKey}/${deviceName}/thing/config/get` 主动查询最新的配置信息。格式请参考：[请求数据格式](#)



4. 设备端接收云端下发的配置文件下载链接后，自行更新配置。



说明

- 必须开启产品的远程配置功能后，才可以编辑配置信息。
- 切换为关闭状态，即关闭该产品的远程配置功能。
- 产品配置模板适用于该产品下的所有设备。目前，不支持在控制台向单个设备

推送配置文件。

- 远程配置文件为 JSON 格式。物联网平台对配置内容没有特殊要求，但系统会对提交的配置文件进行 JSON 格式校验，避免错误格式引起配置异常。
- 配置文件最大支持 64 KB。编辑框右上角将实时显示当前文件的大小。超过 64KB 的配置文件无法提交。
- 批量更新频率限制：一小时内仅允许操作一次。
- 云端下发的配置文件下载链接有效期为 30 分钟，设备端需要及时下载。

NTP 时钟同步：如何得出设备上的精确时间

简介：由于部分嵌入式设备资源受限，且系统本身不包含 NTP 服务，导致了在端上没有精确时间戳。阿里云物联网平台提供了 NTP 时钟同步功能，通过计算设备端与云端的时间差，来得出设备上的精确时间。

概述

物联网平台提供 NTP 服务，解决嵌入式设备资源受限，系统不包含 NTP 服务，端上没有精确时间戳的问题。

物联网平台借鉴 NTP 协议原理，将云端作为 NTP 服务器。设备端发送一个特定 Topic 给云端，payload 中带上发送时间。云端回复时在 payload 中加上云端的接收时间和发送时间。设备端收到回复后，再结合自己本地当前时间，得出一共 4 个时间。一起计算出设备端与云端的时间差，从而得出端上当前的精确时间。

本文通过 MQTT.fx 模拟设备端进行功能测试，工具接入请参考 [MQTT.fx 接入物联网平台使用说明](#)

使用说明

- 相关 Topic

```
/ext/ntp/${YourProductKey}/${YourDeviceName}/request  
/ext/ntp/${YourProductKey}/${YourDeviceName}/response
```

- 时间戳格式

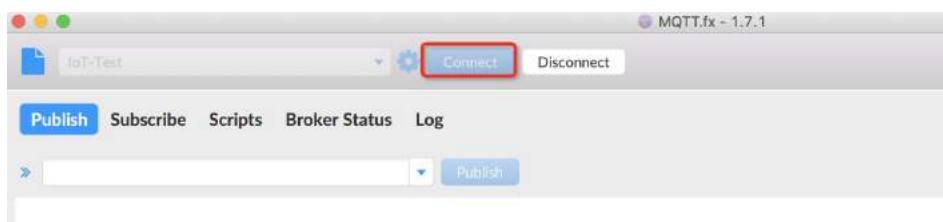
```
{"deviceSendTime":"1571724098000"}
```

说明

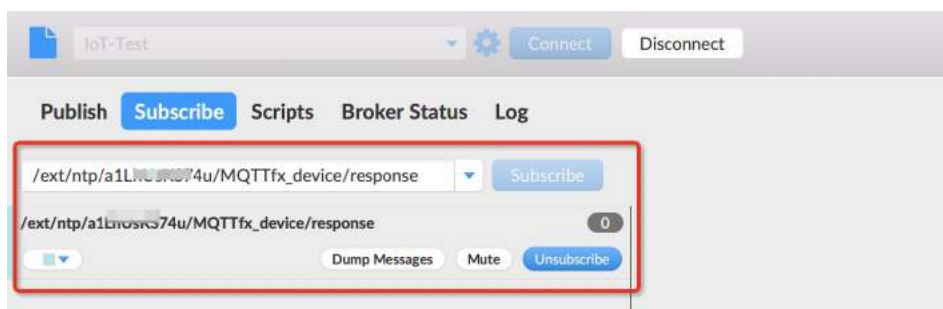
1. 时间戳数字的格式，支持 Long 和 String。默认为 Long 类型。
2. NTP 服务目前仅支持 QoS=0 的消息。

示例

- 通过 MQTT.fx 模拟设备成功和平台进行连接



- 设备端订阅 Topic: /ext/ntp/\${YourProductKey}/\${YourDeviceName}/response



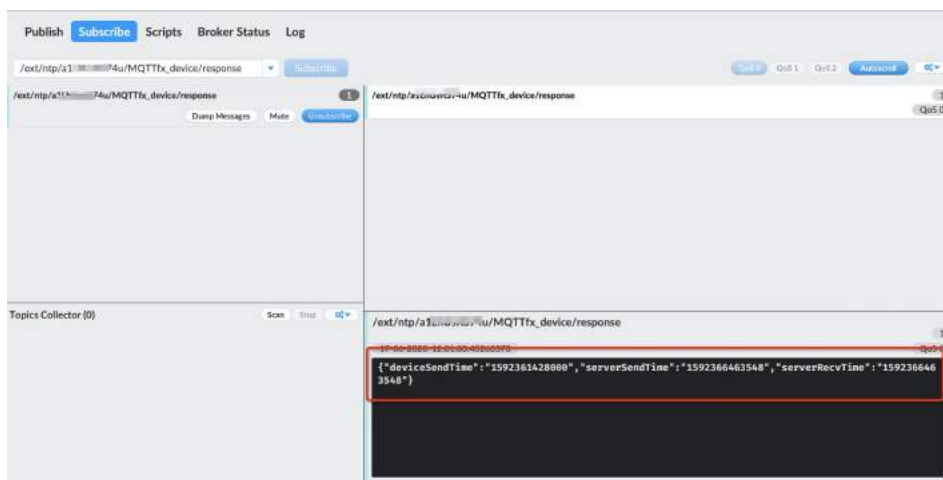
- 设备端向 Topic: /ext/ntp/\${YourProductKey}/\${YourDeviceName}/request 发送一条 QoS=0 的消息，payload 中带上设备当前的时间戳，单位为毫秒。

```
{"deviceSendTime": "1592361428000"}
```



- 设备端收到服务端回复的消息

```
{ "deviceSendTime": "1592361428000", "serverSend Time": "1592366463548",  
  "serverRecvTime": "1592366463548" }
```



- 设备端计算出当前精确的 unix 时间

设备端收到服务端的时间记为 deviceRecvTime ，则设备上的精确时间为： $(\text{serverRecvTime} + \text{serverSendTime} + \text{deviceRecvTime} - \text{deviceSendTime}) / 2$ 。

官方文档

请参考：[NTP 服务](#)

基于开源 Java MQTT Client 演示 RRPC 的实现

简介：本文主要基于开源 Java MQTT Client，分别针对系统 Topic 和自定义 Topic，演示阿里云物联网平台 RRPC 的实现。

概述

MQTT 协议是基于 PUB/SUB 的异步通信模式，不适用于服务端同步控制设备端返回结果的场景。物联网平台基于 MQTT 协议制定了一套请求和响应的同步机制，无需改动 MQTT 协议即可实现同步通信。物联网平台提供 API 给服务端，设备端只需要按照固定的格式回复 PUB 消息，服务端使用 API，即可同步获取设备端的响应结果。RRPC：Revert-RPC。RPC (Remote Procedure Call) 采用客户机 / 服务器模式，用户不需要了解底层技术协议，即可远程请求服务。RRPC 则可以实现由服务端请求设备端并能够使设备端响应的功能。本文主要基于开源 Java MQTT Client，分别针对系统 Topic 和自定义 Topic，演示阿里云物联网平台 RRPC 的实现。

RRPC 原理



1. 物联网平台收到来自用户服务器的 RRPC 调用，下发一条 RRPC 请求消息给设备。消息体为用户传入的数据，Topic 为物联网平台定义的 Topic，其中含有唯一的 RRPC 消息 ID。
2. 设备收到下行消息后，按照指定 Topic 格式（包含之前云端下发的唯一的 RRPC 消息 ID）回复一条 RRPC 响应消息给云端，云端提取出 Topic 中的消息 ID，和之前的 RRPC 请求消息匹配上，然后回复给用户服务器。
3. 如果调用时设备不在线，云端会给用户服务器返回设备离线的错误；如果设备没有在超时时间内（8 秒内）回复 RRPC 响应消息，云端会给用户服务器返回超时错误。

更多原理介绍可以参考[阿里云官方文档](#)。

实验测试

准备工作

1. 创建产品和设备

参考：[阿里云物联网平台 Quick Start](#) 创建产品和设备部分。

2. 创建自定义 Topic

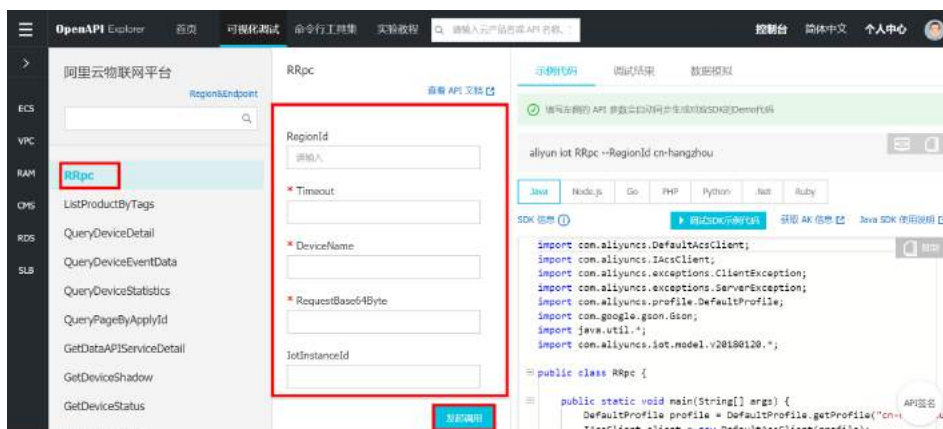


3. 开源 SDK 的使用

参考：[基于开源 JAVA MQTT Client 连接阿里云 IoT](#)

4. 服务端 RRPC API 的调用：调用该接口向指定设备发送请求消息，并同步返回响应

这里我们使用 Open API Explorer 快速测试调用，RRPC 测试[地址](#)。



系统 Topic 测试

参数:

RRPC 调用的系统 Topic 格式如下:

- RRPC 请求消息 Topic: /sys/\${YourProductKey}/\${YourDeviceName}/rrpc/request/\${messageld}
- RRPC 响应消息 Topic: /sys/\${YourProductKey}/\${YourDeviceName}/rrpc/response/\${messageld}
- RRPC 订阅 Topic: /sys/\${YourProductKey}/\${YourDeviceName}/rrpc/request/+

1. 设备端 Code

```
import com.alibaba.taro.AliyunIoTSignUtil;
import org.eclipse.paho.client.mqttv3.*;
import org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;
import java.util.HashMap;
import java.util.Map;

public class IoTRRPCForSysTopic {

    // 设备三元组信息
    public static String productKey = "*****";
    public static String deviceName = "*****";
    public static String deviceSecret = "*****";
    public static String regionId = "cn-shanghai";

    // RRPC 系统 Topic
    private static String subTopic = "/sys/" + productKey + "/" + deviceName +
"/rrpc/request/+";
    private static MqttClient mqttClient;

    public static void main(String [] args) {

        initAliyunIoTClient();

        // RRPC 订阅 Topic
        try {
            mqttClient.subscribe(subTopic);
        } catch (MqttException e) {
```



```

        e.printStackTrace();
    }

    mqttClient.setCallback(new MqttCallback() {
        @Override
        public void connectionLost(Throwable cause) {
            System.out.println("connectionLost:" + cause.getMessage());
        }

        @Override
        public void messageArrived(String topic, MqttMessage message)
            throws Exception {
            System.out.println("message: " + new String(message.
                getPayload()));

            // 根据 RRPC 请求消息 Topic, 构建 RRPC 响应消息 Topic
            String responseTopic = topic.replace("request", "response");
            MqttMessage message1 = new MqttMessage("resonse demo".
                getBytes("utf-8"));
            mqttClient.publish(responseTopic, message1);
        }

        @Override
        public void deliveryComplete(IMqttDeliveryToken token) {
            System.out.println("IMqttDeliveryToken: " + token);
        }
    });
}

/**
 * 初始化 Client
 */
private static void initAliyunIoTClient() {
    try {
        String clientId = "java" + System.currentTimeMillis();

        Map<String, String> params = new HashMap<>(16);
        params.put("productKey", productKey);
        params.put("deviceName", deviceName);
        params.put("clientId", clientId);
        String timestamp = String.valueOf(System.currentTimeMillis());
        params.put("timestamp", timestamp);

        // cn-shanghai
        String targetServer = "tcp://" + productKey + ".iot-as-mqtt." +
            regionId + ".aliyuncs.com:1883";

        String mqttclientId = clientId +
            "|securemode=3,signmethod=hmachal,timestamp=" + timestamp + "|";
    }
}

```

```

        String mqttUsername = deviceName + "&" + productKey;
        String mqttPassword = AliyunIoTSignUtil.sign(params,
deviceSecret, "hmacsha1");

        connectMqtt(targetServer, mqttclientId, mqttUsername,
mqttPassword);

        } catch (Exception e) {
            System.out.println("initAliyunIoTClient error " +
e.getMessage());
        }
    }

    public static void connectMqtt(String url, String clientId, String
mqttUsername, String mqttPassword) throws Exception {

        MemoryPersistence persistence = new MemoryPersistence();
        mqttClient = new MqttClient(url, clientId, persistence);
        MqttConnectOptions connOpts = new MqttConnectOptions();
        // MQTT 3.1.1
        connOpts.setMqttVersion(4);
        connOpts.setAutomaticReconnect(false);
        connOpts.setCleanSession(true);

        connOpts.setUserName(mqttUsername);
        connOpts.setPassword(mqttPassword.toCharArray());
        connOpts.setKeepAliveInterval(60);

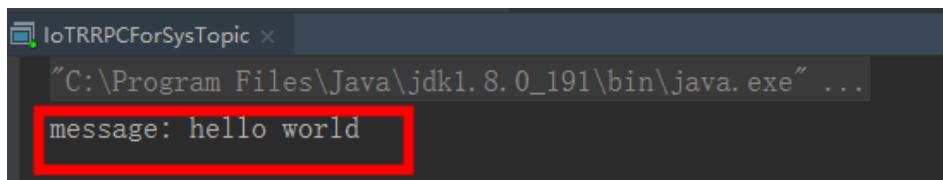
        mqttClient.connect(connOpts);
    }
}

```

2. 服务端下发消息



3. 设备端订阅情况



```
IoTRRPCForSysTopic x
"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...
message: hello world
```

自定义 Topic 测试

参数

RRPC 调用自定义 Topic 的格式如下:

- RRPC 请求消息 Topic: /ext/rrpc/\${messageld}/\${topic}
- RRPC 响应消息 Topic: /ext/rrpc/\${messageld}/\${topic}
- RRPC 订阅 Topic: /ext/rrpc/+/\${topic}

1. 设备端 Code

```
import com.alibaba.taro.AliyunIoTSignUtil;
import org.eclipse.paho.client.mqttv3.*;
import org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;
import java.util.HashMap;
import java.util.Map;

public class IoTRRPCForPersonalTopic {

    // 设备三元组信息
    public static String productKey = "*****";
    public static String deviceName = "*****";
    public static String deviceSecret = "*****";
    public static String regionId = "cn-shanghai";

    // RRPC 自定义 Topic
    private static String personalTopic = "/" + productKey + "/" + deviceName
+ "/user/rrpcdemo";
    private static String subTopic = "/ext/rrpc/+" + personalTopic;
    private static MqttClient mqttClient;
```

```

public static void main(String [] args){

    initAliyunIoTClient();

    try {
        mqttClient.subscribe(subTopic);
    } catch (MqttException e) {
        e.printStackTrace();

        System.out.println(e.getMessage());
    }

    mqttClient.setCallback(new MqttCallback() {
        @Override
        public void connectionLost(Throwable cause) {
            System.out.println("connectionLost");
        }

        @Override
        public void messageArrived(String topic, MqttMessage message)
throws Exception {
            System.out.println("topic: " + topic);
            System.out.println("message: " + new String(message.
getPayload()));

            // 设置响应 Topic 及 message
            MqttMessage msg = new MqttMessage("demo".getBytes());
            mqttClient.publish(topic,msg);
        }

        @Override
        public void deliveryComplete(IMqttDeliveryToken token) {
            System.out.println("IMqttDeliveryToken: " + token);
        }
    });
}

/**
 * 初始化 Client
 */
private static void initAliyunIoTClient() {

    try {
        String clientId = "java" + System.currentTimeMillis();

        Map<String, String> params = new HashMap<>(16);
        params.put("productKey", productKey);
        params.put("deviceName", deviceName);
        params.put("clientId", clientId);
    }
}

```

```
String timestamp = String.valueOf(System.currentTimeMillis());
params.put("timestamp", timestamp);

// cn-shanghai
String targetServer = "tcp://" + productKey + ".iot-as-
mqtt."+regionId+".aliyuncs.com:1883";

// 从云端下发自定义格式 Topic 的 RRPC 调用命令到设备端时，设备端必须在进行
MQTT CONNECT 协议设置时，在 clientId 中增加 ext=1 参数。
String mqttclientId = clientId +
"|securemode=3,signmethod=hmacsha1,timestamp=" + timestamp + ",ext=1|";
String mqttUsername = deviceName + "&" + productKey;
String mqttPassword = AliyunIoTSignUtil.sign(params,
deviceSecret, "hmacsha1");

connectMqtt(targetServer, mqttclientId, mqttUsername,
mqttPassword);

} catch (Exception e) {
    System.out.println("initAliyunIoTClient error " +
e.getMessage());
}
}

public static void connectMqtt(String url, String clientId, String
mqttUsername, String mqttPassword) throws Exception {

    MemoryPersistence persistence = new MemoryPersistence();
    mqttClient = new MqttClient(url, clientId, persistence);
    MqttConnectOptions connOpts = new MqttConnectOptions();
    // MQTT 3.1.1
    connOpts.setMqttVersion(4);
    connOpts.setAutomaticReconnect(false);
    connOpts.setCleanSession(true);

    connOpts.setUserName(mqttUsername);
    connOpts.setPassword(mqttPassword.toCharArray());
    connOpts.setKeepAliveInterval(60);

    mqttClient.connect(connOpts);
}
}
```

注意

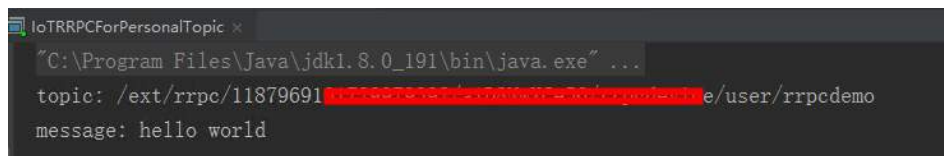
从云端下发自定义格式 Topic 的 RRPC 调用命令到设备端时，设备端必须在进

行 MQTT CONNECT 协议设置时，在 clientId 中增加 ext=1 参数。

2. 服务端下发消息



3. 设备端订阅情况



参考链接

[调用系统 Topic](#)

[调用自定义 Topic](#)



云服务技术大学
云产品干货高频分享



云服务技术课堂
和大牛零距离沟通



阿里云开发者 " 藏经阁 "
海量免费电子书下载