ALIBABA CLOUD

阿里云

生活物联网平台（飞燕平台）
云端开发指南

文档版本：20220712

阿里云

法律声明

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

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

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

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

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

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

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

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

目录

1.概述

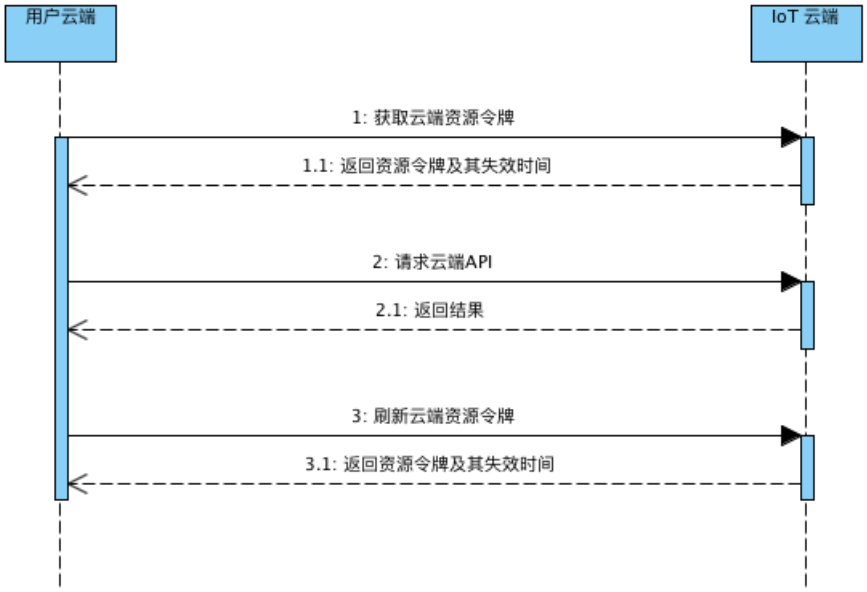
当您自己的云端（web应用或服务）需要调用生活物联网平台提供的云端API时，您需要先了解调用的使用说明、云云交互流程、获取云端唯一身份AppKey的方法、以及调用云端API的示例。

使用说明

- 云端API使用的AppKey与App端使用的AppKey不一致，但是访问环境一致。获取云端AppKey的方法参见本文档下方内容。
- 基于安全考虑，所有云端API调用请使用HTTPS方式，不推荐HTTP方式。
- 所有的API均为POST调用。

云端调用流程介绍

您自己的云端（web应用或服务）调用生活物联网平台提供的云端API时，与阿里云IoT云端（服务端）之间的交互流程如下图所示。



流程说明如下。

1. 您的云端通过调用获取云端资源Token向阿里云IoT云端请求资源令牌（Cloud Token）；阿里云IoT云端收到请求后向您返回Cloud Token，及其对应的失效时间。

   阿里云IoT云端以项目（Project）维度授予Cloud Token，即不同项目的Cloud Token不同。申请Cloud Token时，需携带项目ID。

2. 您的云端使用获取到的Cloud Token调用阿里云IoT云端某个具体的API；阿里云IoT云端向您返回请求的处理结果。
3. 在Cloud Token失效前，您的云端通过调用刷新云端资源Token接口对当前Cloud Token进行续期。

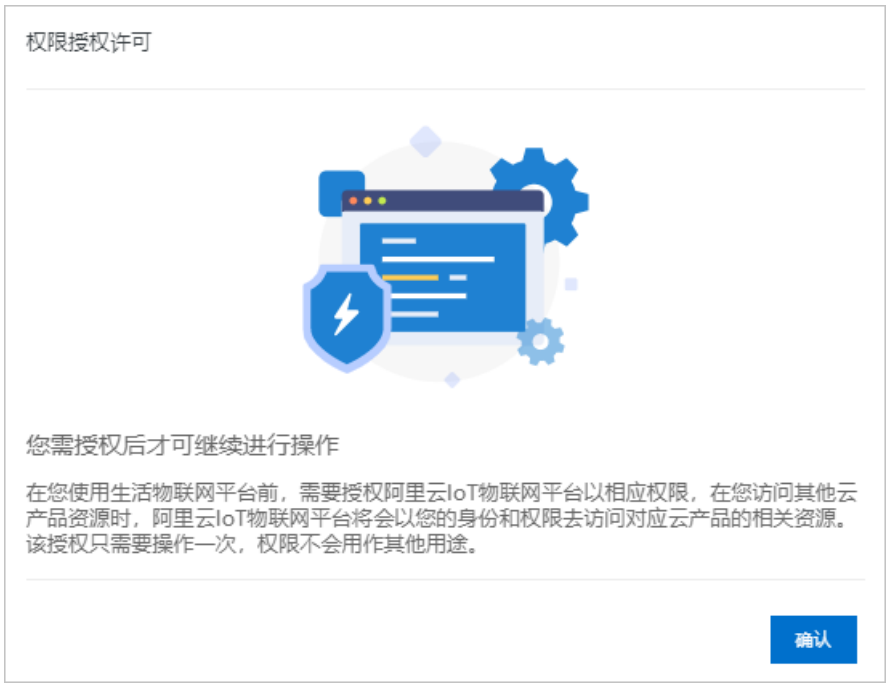
   如果Cloud Token已经失效，则需重新调用获取云端资源Token接口来获取新的Cloud Token及其对应的失效时间。

获取云端Appkey

请您根据以下操作获取云端唯一身份AppKey。

1. 登录生活物联网控制台。
2. 选择项目名称，并单击**项目设置**。

3. （可选）如果您的账户还没完成授权，此时界面弹出**权限授权许可**对话框，您需要为平台授权。
   i. 在**权限授权许可**对话框中，单击**确认**。



ii. 在**云资源访问授权**页面，单击**同意授权**。

4. 选择**设置数据同步**。在云端接口调用区域中，显示了线上环境的AppKey和AppSecret。

调用示例

生活物联网平台提供了多种语言的云端API调用示例，详细调用示例请参见如何调用云端API。

2.数据HTTP/2方式推送

HTTP/2转储功能适用于生活物联网平台与企业服务器之间的消息流转。通过集成和使用HTTP/2 SDK，即可实现身份认证、消息接收的能力。我们推荐使用HTTP/2的方式推送设备数据（如设备状态数据、设备控制记录等），用户信息数据等。

前提条件

启动设备数据同步，并配置要同步数据的产品。详细参见设置数据同步。

说明 当开启数据同步后，集成HTTP/2客户端SDK来订阅数据。如果通过控制台关闭数据同步再开启时，客户端SDK需要重新进行连接流程，否则无法正常接收数据。如果切换不同的HTTP/2客户端，需要把之前的客户端断开，再连接新的客户端。否则新客户端无法正常接收数据。

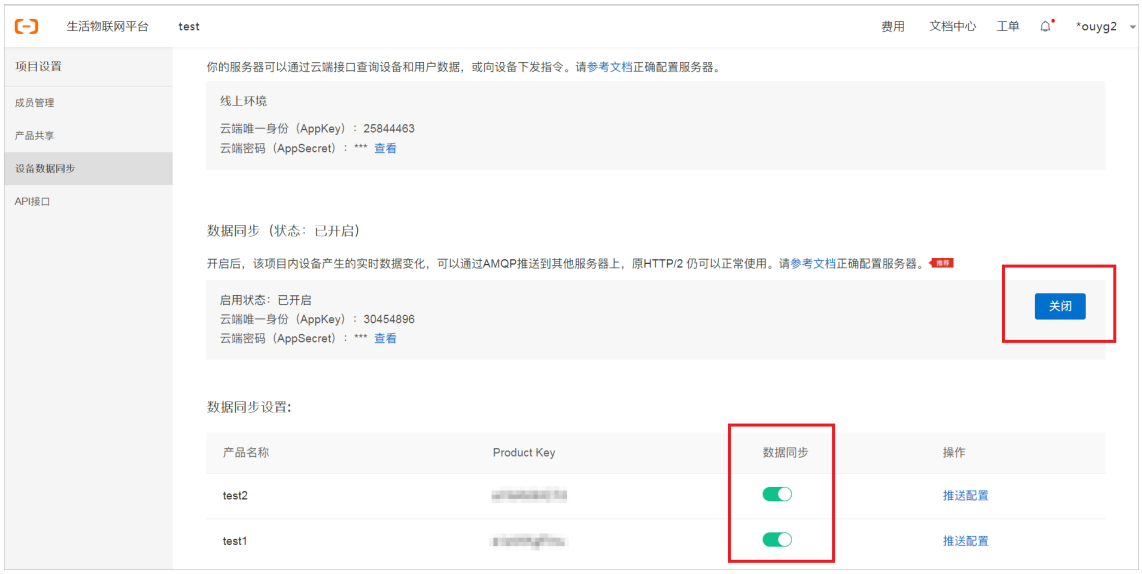
HTTP/2 SDK使用

1. 引入依赖。

   在项目中添加Maven依赖，Maven信息如下。

```
<dependency>
  <groupId>com.aliyun.openservices</groupId>
  <artifactId>iot-client-message</artifactId>
  <version>1.1.5</version>
</dependency>
<dependency>
    <groupId>commons-codec</groupId>
    <artifactId>commons-codec</artifactId>
    <version>1.11</version>
</dependency>
```

2. 认证身份信息。
   - 身份认证需要使用AppKey和AppSecret，该信息可以从控制台中获取。



- 认证身份信息需要profile信息，且Profile需要提供EndPoint、AppKey和AppSecret用于鉴权。

```
Profile profile = Profile.getAppKeyProfile("${END_POINT}", "${APP_KEY}", "${APP_SECRET}");
```

其中，EndPoint是连接节点，具体取值如下表所示。

区域	End Point
中国内地	https://ilop.iot-as-http2.cn-shanghai.aliyuncs.com:443
新加坡	https://ilop.iot-as-http2.ap-southeast-1.aliyuncs.com:443
美国（弗吉尼亚）	https://ilop.iot-as-http2.us-east-1.aliyuncs.com:443
德国（法兰克福）	https://ilop.iot-as-http2.eu-central-1.aliyuncs.com:443

3. 接收云端消息。

   首先需要创建消息接收的客户端对象client，并传入上面身份认证的profile信息。当消息接收的客户端和服务端建立连接后，服务端会立即向消息接收的客户端推送已订阅的消息，因此建立连接时需要提供默认消息接收的回调接口，用于处理云端推送的消息。

   代码片段示例如下。

```
MessageClient messageClient = MessageClientFactory.messageClient(profile);
MessageCallback messageCallback = new MessageCallback() {
        public Action consume(MessageToken messageToken) {
            System.out.println("receive : " + new String(messageToken.getMessage().getPayload()));
            return Action.CommitSuccess;
        }
    };
messageClient.connect(messageCallback);
```

- 消息接收的回调接口

用户需要实现MessageCallback接口的consume方法，并调用客户端对象client的 `setMessageListener()` 方法设置回调接口。

代码片段示例如下。

```
MessageCallback messageCallback = new MessageCallback() {
    public Action consume(MessageToken messageToken) {
        System.out.println("receive : " + new String(messageToken.getMessage().getPayload()));
        return Action.CommitSuccess;
    }
};
messageClient.setMessageListener(messageCallback);
```

- 指定Topic回调

  对于消息接收的回调接口，可以指定带通配符的Topic，如 `/${ProductKey}/{$DeviceName}/#` ，指定Topic的回调优先级高于通用的回调，一条消息匹配到多个Topic时，按英文字母顺序优先调用，仅会回调一次。

  关于topic的概念请参见什么是Topic。

  示例如下。

  ```
  // 当收到消息的Topic匹配到"/Product_A/Device_B/update"时，会优先调用该回调
  messageClient.setMessageListener("/Product_A/Device_B/#",messageCallback);
  ```

- 通用回调

  设置该回调后，对于未设置指定的topic回调的消息，均会调用该回调。示例如下。

  ```
  // 当收到消息的topic未匹配到已指定的Topic时，调用该回调。该通用回调的设置效果等同于messageClient.connect(messageCallback)的。
  messageClient.setMessageListener(messageCallback);
  ```

完整示例如下。

```
public static void main(String[] args) throws UnsupportedEncodingException {
    Profile profile = Profile.getAppKeyProfile("要连接的<END_POINT>", "您的<AppKey>", "您的<AppSecret>");
    MessageCallback messageCallback = new MessageCallback() {
        public Action consume(MessageToken messageToken) {
            System.out.println("receive : " + new String(messageToken.getMessage().getPayload()));
            return Action.CommitSuccess;
        }
    };
    MessageClient messageClient = MessageClientFactory.messageClient(profile);
    messageClient.setMessageListener(messageCallback);
    messageClient.connect(messageCallback);
    try
    {
        System.in.read();
    } catch (Exception e) {}
}
```

> **说明** 消息推送失败时，平台会重新推送，重试策略如下。
>
> - 如果对端不在线或未回复ack消息，则会造成消息堆积，堆积的消息转为离线消息。
> - 离线消息每隔5min重试推送一次（每次推送10条）。对端如果成功接收了消息，则重试策略会继续推送剩余的离线消息（推送失败的消息，下一次继续推送）。
> - 离线消息最多会保存7天，如果7天后仍然无法推送成功，则会被删除。
> - 离线消息会进入单独的队列，不会影响后续消息的实时推送。

消息格式

- 物的属性变更消息

  topic：`/${productKey}/${deviceName}/thing/event/property/post`

  消息字段说明如下。

参数	子参数	子参数	类型	含义
deviceType			String	设备所属品类
gmtCreate			Long	数据流转消息产生时间，自1970-1-1起流逝的毫秒值
iotId			String	设备的唯一ID
productKey			String	设备所属产品的唯一标识符
deviceName			String	设备名称
items			JSON	变更的状态列表
	attribute		String	发生变更的属性，具体取值由具体情况确定
		value	具体数据类型由具体情况确定	变更值
		time	Long	设备属性发生变化的时间，自1970-1-1起流逝的毫秒值

  消息示例如下。

```
{
    "deviceType": "SmartDoor",
    "iotId": "Xzf15db9xxxxxxxxWR001046b400",
    "productKey": "a17xxxxTYNA",
    "gmtCreate": 153xxxx145304,
    "deviceName": "Xzf15xxxxucTHBgUo6WR",
    "items": {
        "WIFI_Rx_Rate": {
            "value": 74274,
            "time": 1534299145344
        }
    }
}
```

- 物的事件变更消息

  topic：`/${productKey}/${deviceName}/thing/event/{tsl.event.identifier}/post`

  消息字段说明如下。

参数	子参数	类型	含义
deviceType		String	设备所属品类
iotId		String	设备的唯一ID
productKey		String	设备所属产品的唯一标识符
deviceName		String	设备名称
identifier		String	事件标识符，对应事件的identifier
name		String	事件名称
type		String	事件类型
time		Long	设备上报value对应的时间，自1970-1-1起流逝的毫秒值
value		JSON	变更的事件属性列表：key-value键值对
	key	String	属性key
	value	具体数据类型由具体情况确定	属性取值

  消息示例如下。

```
{
    "deviceType": "SmartDoor",
    "identifier": "Doorxxxxication",
    "iotId": "Xzf15db9xxxxxxxxx01046b400",
    "name": "开门通知",
    "time": 1534319108982,
    "type": "info",
    "productKey": "a17xxxxTYNA",
    "deviceName": "Xzf15xxxxucTHBgUo6WR",
    "value": {
        "KeyID": "x8xxxxxkDY",
        "LockType": 3
    }
}
```

- 设备服务返回消息

  topic： `/${productKey}/${deviceName}/thing/downlink/reply/message`

  消息字段说明如下。

参数	类型	含义
gmtCreate	Long	数据流转消息产生时间，自1970-1-1起流逝的毫秒值
iotId	String	设备的唯一ID
productKey	String	设备所属产品的唯一标识符
deviceName	String	设备名称
requestId	String	阿里云产生和设备通信的信息ID
code	Integer	调用的结果信息
message	String	结果信息说明
topic	String	服务调用下行时使用的topic
data	Object	设备返回的结果，非透传之间返回设备结果，透传则需要经过脚本转换

  消息示例如下。

```
{
  "gmtCreate": 151xxxx39881,
  "iotId": "4z819VQHxxxxxxxxxxxx7ee200",
  "productKey": "p1gxxxxBd",
  "deviceName": "xxxxxxxxxx",
  "requestId": "1234",
  "code": 200,
  "message": "success",
  "topic": "/sys/p1gsv0teUBd/xxxxxxxxxx/thing/service/property/set",
  "data": {}
}
```

- 物的状态变更消息

  为了提高消息有效性，设备上下线过于频繁时，会对消息进行筛检。

  topic： `/${productKey}/${deviceName}/mqtt/status`

  消息字段说明如下。

参数	子参数	类型	含义
deviceType		String	设备所属品类
gmtCreate		Long	数据流转消息产生时间，自1970-1-1起流逝的毫秒值
iotId		String	设备的唯一ID
action		String	设备状态变更动作： ○ online：上线动作 ○ offline：下线动作
productKey		String	设备所属产品的唯一标识符
deviceName		String	设备名称
status		JSON	状态信息，元素包括：value-状态值，time-发生变化的时间
	time	Long	设备上下线状态发生变化的时间，自1970-1-1起流逝的毫秒值
	value	String	设备上下线状态 value状态值定义： ○ 1：在线 ○ 0：离线

  消息示例如下。

```
{
    "deviceType": "SmartDoor",
    "iotId": "Xzf15dxxxxxxxxxxxxxxx01046b400",
    "action": "online",
    "productKey": "a17xxxxxxTYNA",
    "gmtCreate": 153xxxx1368,
    "deviceName": "Xzf1xxxxxxxxxxxgUo6WR",
    "status": {
        "time": 1534319611368,
        "value": "1"
    }
}
```

- 用户绑定变更消息

  用户绑定/解绑设备产生的回流消息，用于同步用户与设备的绑定、解绑。

  topic: `/${productKey}/${deviceName}/thing/awss/enrollee/user`

  消息字段说明如下。

参数	子参数	类型	含义
bind		bool	true-绑定；false-解绑
productKey		String	设备所属产品的唯一标识符
deviceName		String	设备名称
iotId		String	设备的唯一ID
messageCreateTime		JSON	消息创建时间
identityInfos		list	用户信息列表
	identityId	String	用户身份ID
	scopeId	String	隔离ID
	tenantId	String	租户ID
	owned	Integer	拥有标记 ○ 0：分享者 ○ 1：拥有者
params		Map	扩展参数（暂未使用）

```
{
  "bind":true,
  "productKey": "123xxxx569",
  "deviceName": "deviceNamexxxx34",
  "iotId": "",
  "messageCreateTime": 151xxxx9881,
  "identityInfos":[
     {
       "identityId":"50xxxxxxxxxxx62060259",
       "scopeId":"",
       "tenantId":"1D89B5xxxxxxxxxxxxxxxx861678FF",
       "owned":1
     }
  ],
  "params":{
  }
}
```

3.数据AMQP方式推送

AMQP（Advanced Message Queuing Protocol，高级消息队列协议）转储功能适用于生活物联网平台与企业服务器之间的消息流转。通过集成和使用AMQP SDK，即可实现身份认证、消息接收的能力。我们推荐使用AMQP的方式推送设备数据（如设备状态数据、设备控制记录等），用户信息数据等。

前提条件

开启设备数据同步，并配置要同步数据的产品。详细参见设置数据同步。

> **说明** 当开启数据同步后，集成AMQP客户端SDK来订阅数据。如果通过控制台关闭数据同步再开启时，客户端SDK需要重新进行连接流程，否则无法正常接收数据。如果切换不同的AMQP客户端，需要把之前的客户端断开，再连接新的客户端。否则新客户端无法正常接收数据。

AMQP SDK使用

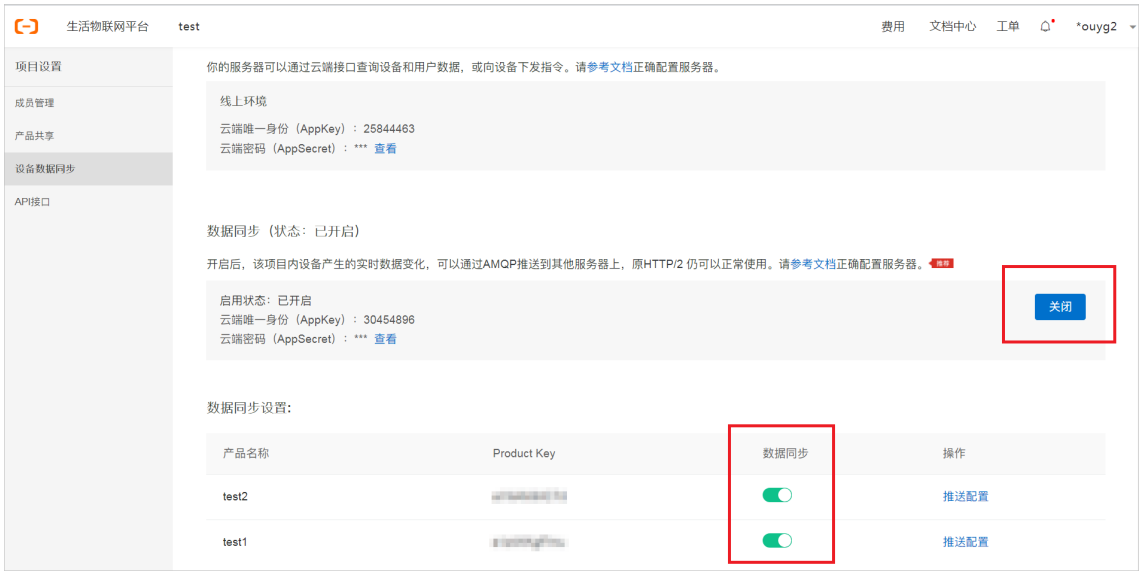
1. 引入依赖。

   AMQP SDK为开源SDK。如果您使用Java开发语言，推荐使用Apache Qpid JMS客户端。在项目中添加Maven依赖，Maven信息如下。

   > **说明** 目前生活物联网平台仅支持Java语言开发，暂未提供其他语言版本，请以生活物联网平台官网为主。

```
<!-- 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>
```

2. 认证身份信息。
   - 身份认证需要使用AppKey和AppSecret，该信息可以从控制台中获取。



   - 认证身份信息需要使用EndPoint、AppKey和AppSecret用于鉴权。

     其中，EndPoint是连接节点，具体取值如下表所示。

区域	End Point
中国内地	amqps://ilop.iot-amqp.cn-shanghai.aliyuncs.com:5671
新加坡	amqps://ilop.iot-amqp.ap-southeast-1.aliyuncs.com:5671
美国（弗吉尼亚）	amqps://ilop.iot-amqp.us-east-1.aliyuncs.com:5671
德国（法兰克福）	amqps://ilop.iot-amqp.eu-central-1.aliyuncs.com:5671

3. 接收云端消息。

   首先需要创建消息接收的客户端对象client，并传入上面身份认证的profile信息。当消息接收的客户端和服务端建立连接后，服务端会立即向消息接收的客户端推送已订阅的消息，因此建立连接时需要提供默认消息接收的回调接口，用于处理云端推送的消息。

   完整代码示例如下。

```
import java.net.URI;
import java.util.Hashtable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;
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.cla
ss);
    //业务处理异步线程池，线程池参数可以根据您的业务特点调整，或者您也可以用其他异步方式处理接收到
的消息。
    private final static ExecutorService executorService = new ThreadPoolExecutor(
        Runtime.getRuntime().availableProcessors(),
        Runtime.getRuntime().availableProcessors() * 2, 60, TimeUnit.SECONDS,
        new LinkedBlockingQueue<>(50000));
    public static void main(String[] args) throws Exception {
        String appKey = "${YourAppkey}"; //在控制台自有App，单击密钥对应的查看，可显示App的A
pp Key和App Secret 创建App时系统
        String appSecret = "${YourAppSecret}";
        String consumerGroupId = "${YourAppkey}";
        long random = xxxxx;
        //建议使用机器UUID、MAC地址、IP等唯一标识等作为clientId。便于您区分识别不同的客户端。
        String clientId = "${YourClientId}";
        String userName = clientId + "|authMode=appkey"
                + ",signMethod=" + "SHA256"
                + ",random=" + random
                + ",appKey=" + appKey
                + ",groupId=" + consumerGroupId + "|";
        String signContent = "random=" + random;
        String password = doSign(signContent, appSecret, "HmacSHA256");
        String connectionUrlTemplate = "failover:(${AMQPEndPointUrl}?amqp.idleTimeout=8
0000)"
                + "?failover.maxReconnectAttempts=10&failover.reconnectDelay=30";
        Hashtable<String, String> hashtable = new Hashtable<>();
        hashtable.put("connectionfactory.SBCF",connectionUrlTemplate);
        hashtable.put("queue.QUEUE", "default");
```

```
        hashtable.put(Context.INITIAL_CONTEXT_FACTORY, "org.apache.qpid.jms.jndi.JmsIni
tialContextFactory");
        Context context = new InitialContext(hashtable);
        ConnectionFactory cf = (ConnectionFactory)context.lookup("SBCF");
        Destination queue = (Destination)context.lookup("QUEUE");
        // 创建连接。
        Connection connection = cf.createConnection(userName, password);
        ((JmsConnection) connection).addConnectionListener(myJmsConnectionListener);
        // 创建会话。
        // 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();
        // 创建Receiver连接。
        MessageConsumer consumer = session.createConsumer(queue);
        consumer.setMessageListener(messageListener);
    }
    private static MessageListener messageListener = new MessageListener() {
        @Override
        public void onMessage(Message message) {
            try {
                //1.收到消息之后一定要ACK。
                // 推荐做法：创建Session选择Session.AUTO_ACKNOWLEDGE，这里会自动ACK。
                // 其他做法：创建Session选择Session.CLIENT_ACKNOWLEDGE，这里一定要调message
.acknowledge()来ACK。
                // message.acknowledge();
                //2.建议异步处理收到的消息，确保onMessage函数里没有耗时逻辑。
                // 如果业务处理耗时过程过长阻塞住线程，可能会影响SDK收到消息后的正常回调。
                executorService.submit(() -> processMessage(message));
            } catch (Exception e) {
                logger.error("submit task occurs exception ", e);
            }
        }
    };
    /**
     * 在这里处理您收到消息后的具体业务逻辑。
     */
    private static void processMessage(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);
        } catch (Exception e) {
            logger.error("processMessage occurs error ", e);
        }
    }
    private static JmsConnectionListener myJmsConnectionListener = new JmsConnectionLis
tener() {
        /**
         * 连接成功建立。
```

```
            连接成功建立。
             */
            @Override
            public void onConnectionEstablished(URI remoteURI) {
                logger.info("onConnectionEstablished, 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) {}
        };
        /**
         * 计算签名，password组装方法，请参见AMQP客户端接入说明文档。
         */
        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 Hex.encodeHexString(rawHmac);
        }
}
```

> **说明** 消息推送失败时，平台会重新推送，重试策略如下。
>
> - 如果对端不在线或未回复ack消息，则会造成消息堆积，堆积的消息转为离线消息。
> - 离线消息每隔5min重试推送一次（每次推送10条）。对端如果成功接收了消息，则重试策略会继续推送剩余的离线消息（推送失败的消息，下一次继续推送）。
> - 离线消息最多会保存 1 天，如果 1 天后仍然无法推送成功，则会被删除。
> - 离线消息会进入单独的队列，不会影响后续消息的实时推送。

消息格式

- 物的属性变更消息

  topic：`/${productKey}/${deviceName}/thing/event/property/post`

  消息字段说明如下。

参数	子参数	子参数	类型	含义
deviceType			String	设备所属品类
gmtCreate			Long	数据流转消息产生时间，自1970-1-1起流逝的毫秒值
iotId			String	设备的唯一ID
productKey			String	设备所属产品的唯一标识符
deviceName			String	设备名称
items			JSON	变更的状态列表
	attribute		String	发生变更的属性，具体取值由具体情况确定
		value	具体数据类型由具体情况确定	变更值
		time	Long	设备属性发生变化的时间，自1970-1-1起流逝的毫秒值

  消息示例如下。

```
{
    "deviceType": "SmartDoor",
    "iotId": "Xzf15db9xxxxxxxxWR001046b400",
    "productKey": "a17xxxxTYNA",
    "gmtCreate": 153xxxx145304,
    "deviceName": "Xzf15xxxxucTHBgUo6WR",
    "items": {
        "WIFI_Rx_Rate": {
            "value": 74274,
            "time": 1534299145344
        }
    }
}
```

- 物的事件变更消息

  topic：`/${productKey}/${deviceName}/thing/event/{tsl.event.identifier}/post`

  消息字段说明如下。

参数	子参数	类型	含义
deviceType		String	设备所属品类
iotId		String	设备的唯一ID
productKey		String	设备所属产品的唯一标识符
deviceName		String	设备名称
identifier		String	事件标识符，对应事件的identifier
name		String	事件名称
type		String	事件类型
time		Long	设备上报value对应的时间，自1970-1-1起流逝的毫秒值
value		JSON	变更的事件属性列表：key-value键值对
	key	String	属性key
	value	具体数据类型由具体情况确定	属性取值

  消息示例如下。

```
{
    "deviceType": "SmartDoor",
    "identifier": "Doorxxxxication",
    "iotId": "Xzf15db9xxxxxxxxx01046b400",
    "name": "开门通知",
    "time": 1534319108982,
    "type": "info",
    "productKey": "a17xxxxTYNA",
    "deviceName": "Xzf15xxxxucTHBgUo6WR",
    "value": {
        "KeyID": "x8xxxxxkDY",
        "LockType": 3
    }
}
```

- 设备服务返回消息

  topic： `/${productKey}/${deviceName}/thing/downlink/reply/message`

  消息字段说明如下。

参数	类型	含义
gmtCreate	Long	数据流转消息产生时间，自1970-1-1起流逝的毫秒值
iotId	String	设备的唯一ID
productKey	String	设备所属产品的唯一标识符
deviceName	String	设备名称
requestId	String	阿里云产生和设备通信的信息ID
code	Integer	调用的结果信息
message	String	结果信息说明
topic	String	服务调用下行时使用的topic
data	Object	设备返回的结果，非透传之间返回设备结果，透传则需要经过脚本转换

  消息示例如下。

```
{
  "gmtCreate": 151xxxx39881,
  "iotId": "4z819VQHxxxxxxxxxxxx7ee200",
  "productKey": "p1gxxxxBd",
  "deviceName": "xxxxxxxxxx",
  "requestId": "1234",
  "code": 200,
  "message": "success",
  "topic": "/sys/p1gxxxxeUBd/xxxxxxxxxx/thing/service/property/set",
  "data": {}
}
```

- 物的状态变更消息

  为了提高消息有效性，设备上下线过于频繁时，会对消息进行筛检。

  topic： `/${productKey}/${deviceName}/mqtt/status`

  消息字段说明如下。

参数	子参数	类型	含义
deviceType		String	设备所属品类
gmtCreate		Long	数据流转消息产生时间，自1970-1-1起流逝的毫秒值
iotId		String	设备的唯一ID
action		String	设备状态变更动作： ○ online：上线动作 ○ offline：下线动作
productKey		String	设备所属产品的唯一标识符
deviceName		String	设备名称
status		JSON	状态信息，元素包括：value-状态值，time-发生变化的时间
	time	Long	设备上下线状态发生变化的时间，自1970-1-1起流逝的毫秒值
	value	String	设备上下线状态 value状态值定义： ○ 1：在线 ○ 0：离线

  消息示例如下。

```
{
    "deviceType": "SmartDoor",
    "iotId": "Xzf15dxxxxxxxxxxxxxxx01046b400",
    "action": "online",
    "productKey": "a17xxxxxxTYNA",
    "gmtCreate": 153xxxx1368,
    "deviceName": "Xzf1xxxxxxxxxxxgUo6WR",
    "status": {
        "time": 1534319611368,
        "value": "1"
    }
}
```

- 用户绑定变更消息

  用户绑定/解绑设备产生的回流消息，用于同步用户与设备的绑定、解绑。

  topic: `/${productKey}/${deviceName}/thing/awss/enrollee/user`

  消息字段说明如下。

参数	子参数	类型	含义
bind		bool	true-绑定；false-解绑
productKey		String	设备所属产品的唯一标识符
deviceName		String	设备名称
iotId		String	设备的唯一ID
messageCreateTime		Long	消息创建时间
identityInfos		list	用户信息列表
	identityId	String	用户身份ID
	scopeId	String	隔离ID
	tenantId	String	租户ID
	owned	Integer	拥有标记 ○ 0：分享者 ○ 1：拥有者
params		Map	扩展参数（暂未使用）

```
{
  "bind":true,
  "productKey": "123xxxx569",
  "deviceName": "deviceNamexxxx34",
  "iotId": "",
  "messageCreateTime": 151xxxx9881,
  "identityInfos":[
     {
       "identityId":"50xxxxxxxxxxx62060259",
       "scopeId":"",
       "tenantId":"1D89B5xxxxxxxxxxxxxxxx861678FF",
       "owned":1
     }
  ],
  "params":{
  }
}
```

4.智能云存储开发指南

本文档主要介绍智能云存储套餐在开发过程中的使用规则及注意事项。

对接智能云存储即视为您同意阿里云Link Visual视频云存储服务条款。智能云存储涉及到计费，具体收费策略请参见Link Visual收费策略。

套餐领用和冻结

生活物联网平台提供一定量的免费云存储，每个设备只能领用一次，换一个账号绑定不可以再领取。 免费赠送的套餐查询和领取的接口有App客户端调用的接口和云云调用的接口两种。App客户端调用的接口，设计初衷是考虑到有的客户不具备云端开发能力，提供App客户端调用接口方便您开发的App直接调用领取开通云存储。云云调用的接口，是为了便于具备云端开发能力的客户可以通过云云对接的接口统一管理其自有App免费套餐和付费套餐的开通使用。建议您统一使用云云对接的接口。购买的云存储套餐生效后，不支持退款。提供了云存储套餐停用的接口，便于您自行管理C端用户的云存储套餐冻结停用。API接口请参见设置用户的云存储套餐状态。

> **说明**
>
> 开通Link Visual视频服务的项目中，所有视频产品的设备，C端用户默认可以免费领取3个月的7天循环事件云存储月套餐。
>
> 为了提高C端用户云存储套餐的购买率，您可以调用支付宝的周期性扣款接口，在云存储到期时提示C端用户自动续费。

套餐与设备和用户的关系

设备只有一个管理员用户（通过配网绑定的用户），管理员可以把设备分享给其他用户。Link Visual提供的云云对接接口不限制购买的用户类别，建议您的自有App限制管理员才能购买套餐。您的云端调用Link Visual的购买接口，为用户购买开通云存储，Link Visual会记录下购买用户，购买的云存套餐归属于当前设备和管理员A。套餐会按购买先后顺序生效，新购买的套餐从当前有效套餐结束时间点往后追加。

设备的云存储套餐及套餐有效期内产生的数据，不随设备解绑而转移。即设备被解绑后重新绑定新的管理员用户，在这之前购买云存储套餐及设备产生的数据仍然归属于原管理员用户，不能转移给新的管理员用户。此时新管理员用户没有云存储套餐及数据。如果原管理员用户重新绑定该设备，原云存储套餐及数据在有效期内依然生效。

套餐生效顺序

C端用户给设备领取免费的云存储套餐或者购买云存储套餐，默认按照开通的先后生效。套餐查询接口都会返回每个套餐详情，包括开始时间、结束时间。免费套餐的领取接口和付费套餐的开通接口，都有入参（immediateUse）来指定该套餐开通后是否要调整为立即生效。举个例子：用户当前已经有了A、B两个套餐，如果再新购买一个套餐C，则默认是在A、B两个套餐之后生效；如果厂商云端指定新购套餐C要立即生效，也就是指定immediateUse=true，那么新购买的套餐C立即生效，用户的套餐使用顺序调整为C、A、B。

如果在使用过程中需要调整套餐的先后顺序，那么可以通过设置云存储套餐立即生效的接口来调整。免费领用的套餐调用设置免费云存储套餐立即生效这个接口来调整，付费购买的套餐调用设置云存储套餐立即生效这个接口来调整。例如：用户当前已经有了A、B、C三个套餐，三个套餐的生效顺序是A->B->C，A、B是付费套餐，C为免费套餐。如果用户需要将C调整为立即生效，那么调用设置免费云存储套餐立即生效的接口将C套餐设置为立即生效，这时候新的套餐顺序为C->A->B；如果后续又希望调整为A套餐立即生效，那么调用设置云存储套餐立即生效的接口将A套餐设置为立即生效，这时候新的套餐顺序重新调整为A->C->B。

套餐转移

考虑到C端用户在使用云存储套餐的过程可能会遇到设备有问题需要退换货的情形，支持C端用户将原先设备购买的云存储套餐转移到新的设备上。API接口请参见用户云存套餐转移。

套餐与录像计划

开通云存储套餐后C端用户的设备才有生成云端录像的配额。事件类型的套餐只能录制事件联动产生的事件录像；连续类型的套餐可以录制连续的录像，同时过程中发生的事件录像也可以录制。开通录像计划后，云端才能根据计划决定何时、如何录制生成录像。

录像计划包括连续的录像计划和事件联动的录像计划。开通录像计划需要先配置一个录像计划，再绑定录像计划到设备。只有绑定录像计划到设备，云端才会根据这个设备的录像计划进行录像生成的调度。因此，C端用户设备使用云存储录像的必要条件如下。

- 为设备领取免费云存储套餐或者购买开通云存储套餐。
- 为设备设置一个录像计划。

考虑到C端用户在App上购买了云存储套餐之后，有可能因为没有去开通录像计划而云端不会去调度生成云存储录像，从而引起用户疑惑。云端开通云存储套餐的接口，支持默认开通全天的录像计划，需要设置enableDefaultPlan入参为true。如果用户对录像计划有自定义的需求，则可以通过App调用录像计划的相关接口进行设置。API接口请参见视频服务。

> **说明**
>
> 说明 enableDefaultPlan为true，云端则会为该设备开通全天的连续录像计划、全天的事件联动录像计划。该计划的默认参数为所有类型的事件都进行录像、预录时长5秒、录像时长10秒。