阿里云
物联网平台

常见问题

文档版本：20190624

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
	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。	 **禁止:** 重置操作将丢失用户配置数据。
	该类警示信息可能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，恢复业务所需时间约10分钟。
	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。	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。	单击 确定。
`courier` 字体	命令。	执行 `cd /d C:/windows` 命令，进入Windows系统文件夹。
*## *	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。	`swich` *`{stand | slave}`*

目录

法律声明.....I
通用约定.....I
1 设备端检测自己是否在线.....1
2 服务端检测设备是否在线.....7

1 设备端检测自己是否在线

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

原理

如果设备可以发送、接收消息，那么该设备的通信是没问题的，并且一定在线。

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

该原理的一种特殊实现就是设备端消息的自发自收。

流程

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

   Topic可以自定义，但权限必须为发布和订阅。
2. 设备端订阅上一步创建的Topic 。
3. 设备端发送消息`{"id":123,"version":"1.0","time":1234567890123}`，请一定使用QoS=0 。

   消息内容可自定义，但建议使用此格式。

   参数说明：

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

4. 设备端收到消息上一步发送的消息。

离线判定逻辑

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

说明:

您可以根据自己的情况，自定义离线判定逻辑。

实现

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

Java SDK开发具体细节，请查看相关文档。

说明:

您可以根据自己的喜好，选择不同的设备端*SDK*进行开发。

首先，下载Demo工程，添加本类，并填写设备证书信息。设备端代码如下：

```
import java.io.UnsupportedEncodingException;

import com.aliyun.alink.dm.api.DeviceInfo;
import com.aliyun.alink.dm.api.InitResult;
import com.aliyun.alink.linkkit.api.ILinkKitConnectListener;
import com.aliyun.alink.linkkit.api.IoTMqttClientConfig;
import com.aliyun.alink.linkkit.api.LinkKit;
import com.aliyun.alink.linkkit.api.LinkKitInitParams;
import com.aliyun.alink.linksdk.cmp.connect.channel.MqttPublishRequest
;
import com.aliyun.alink.linksdk.cmp.connect.channel.MqttSubscr
ibeRequest;
import com.aliyun.alink.linksdk.cmp.core.base.AMessage;
import com.aliyun.alink.linksdk.cmp.core.base.ARequest;
import com.aliyun.alink.linksdk.cmp.core.base.AResponse;
import com.aliyun.alink.linksdk.cmp.core.base.ConnectState;
import com.aliyun.alink.linksdk.cmp.core.listener.IConnectNotifyListen
er;
import com.aliyun.alink.linksdk.cmp.core.listener.IConnectSendListener
;
import com.aliyun.alink.linksdk.cmp.core.listener.IConnectSubscribeLis
tener;
import com.aliyun.alink.linksdk.tools.AError;

public class CheckDeviceStatusOnDevice {

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

```
 private static String deviceSecret = "";
 // 消息通信的Topic，需要在控制台定义，权限必须为发布和订阅
 private static String checkStatusTopic = "/" + productKey + "/" +
deviceName + "/user/checkstatus";
 // ==================需要用户填写的参数结束==========================

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

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

  CheckDeviceStatusOnDevice device = new CheckDeviceStatusOnDevice();

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

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

  // 订阅Topic
  device.subscribe(checkStatusTopic);

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

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

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

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

  // 发送消息
```

```
  publish(checkStatusTopic, payload);

  // 严格的离线判定逻辑：发送消息后，5秒内没有收到消息算失败，出现1次失败，判定为离
线
  boolean isTimeout = true;
  for (int i = 0; i < 5; i++) {
   Thread.sleep(1000);
   if (!subInfo.isEmpty()) {
    isTimeout = false;
    break;
   }
  }
  if (!isTimeout && payload.equals(subInfo)) {
   System.out.println("Device is online !!");
  } else {
   System.out.println("Device is offline !!");
  }

  // 置空接收到的消息，方便下一次测试
  subInfo = "";
 }

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

  LinkKitInitParams params = new LinkKitInitParams();

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

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

  params.deviceInfo = deviceInfo;

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

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

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

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

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

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

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

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

 /**
  * 订阅消息
  *
```

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

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

}
```

说明:

检测到设备离线后，尽量不要选择主动重连。

物联网平台规定一个设备1分钟只允许尝试接入平台5次，超过会触发限流，限制设备接入。此时停止接入，等待1分钟即可解除限制。

设备端应注意退避，切勿触发限流。

2 服务端检测设备是否在线

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

原理

如果设备可以接收服务端下发的消息并作出响应，那么该设备通信是没问题的，并且设备一定在线。

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

物联网平台提供的RRPC能力就是该原理的一种特殊实现。

流程

1. 设备端订阅RRPC请求的Topic =`/sys/yourProductKey/yourDeviceName/rrpc/request/+`。

   格式固定。

2. 服务端调用*RRpc*接口发送指令`{"id":123,"version":"1.0","time":1234567890123}`。

   消息内容可自定义，但建议使用此格式。

   参数说明：

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

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

   离线判定逻辑

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

   > 说明:
   > 您可以根据自己的情况，自定义离线判定逻辑。

实现

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

> 说明:
> 您可以根据自己的喜好，选择不同的设备端*SDK*和服务端*SDK*进行开发。

分别下载Demo工程，服务端添加CheckDeviceStatusOnServer类，设备端添加Device类，填写AccessKey信息和设备三元组信息。

设备端代码如下：

```
import java.io.UnsupportedEncodingException;

import com.aliyun.alink.dm.api.DeviceInfo;
import com.aliyun.alink.dm.api.InitResult;
import com.aliyun.alink.linkkit.api.ILinkKitConnectListener;
import com.aliyun.alink.linkkit.api.IoTMqttClientConfig;
import com.aliyun.alink.linkkit.api.LinkKit;
import com.aliyun.alink.linkkit.api.LinkKitInitParams;
import com.aliyun.alink.linksdk.cmp.connect.channel.MqttPublishRequest
;
import com.aliyun.alink.linksdk.cmp.connect.channel.MqttSubscr
ibeRequest;
import com.aliyun.alink.linksdk.cmp.core.base.AMessage;
import com.aliyun.alink.linksdk.cmp.core.base.ARequest;
import com.aliyun.alink.linksdk.cmp.core.base.AResponse;
import com.aliyun.alink.linksdk.cmp.core.base.ConnectState;
import com.aliyun.alink.linksdk.cmp.core.listener.IConnectNotifyListen
er;
import com.aliyun.alink.linksdk.cmp.core.listener.IConnectSendListener
;
import com.aliyun.alink.linksdk.cmp.core.listener.IConnectSubscribeLis
tener;
import com.aliyun.alink.linksdk.tools.AError;

public class Device {

    // ====================需要用户填写的参数，开
始==========================
```

```
    // 产品productKey，设备证书参数之一
    private static String productKey = "";
    // 设备名字deviceName，设备证书参数之一
    private static String deviceName = "";
    // 设备密钥deviceSecret，设备证书参数之一
    private static String deviceSecret = "";
    // 消息通信的Topic，无需创建和定义，直接使用即可
    private static String rrpcTopic = "/sys/" + productKey + "/" +
deviceName + "/rrpc/request/+";
    // ===================需要用户填写的参数，结
束============================

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

        Device device = new Device();

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

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

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

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

        LinkKitInitParams params = new LinkKitInitParams();

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

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

        params.deviceInfo = deviceInfo;

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

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

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

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

            @Override
            public void onNotify(String connectId, String topic,
AMessage aMessage) {
                // 接收RRPC请求并回复RRPC响应
                try {
                    String response = topic.replace("/request/", "/
response/");
                    publish(response, new String((byte[]) aMessage.
getData(), "UTF-8"));
                } catch (UnsupportedEncodingException e) {
                    e.printStackTrace();
                }
            }

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

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

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

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

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

}
```

服务端代码如下：

```
import java.io.UnsupportedEncodingException;

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

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

public class CheckDeviceStatusOnServer extends BaseTest {

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

    public static void main(String[] args) throws ServerException,
ClientException, UnsupportedEncodingException {

        //
 ----------------------------------------------------------------------
```

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

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

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

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

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

    public static DefaultAcsClient getClient() {

        DefaultAcsClient client = null;

        try {
            IClientProfile profile = DefaultProfile.getProfile("cn-
shanghai", accessKeyID, accessKeySecret);
            DefaultProfile.addEndpoint("cn-shanghai", "cn-shanghai", "
Iot", "iot.cn-shanghai.aliyuncs.com");
            client = new DefaultAcsClient(profile);
        } catch (Exception e) {
```

```
            System.out.println("init client failed !! exception:" + e.
getMessage());
        }

        return client;
    }
}
```

说明:

检测流程在服务端主动触发。