ALIBABA CLOUD

阿里云

Java SDK

文档版本：20210302

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.工程配置

1. 配置Maven仓库地址，在pom添加如下maven仓库依赖。

```
<repositories>
  <repository>
    <id>alimaven</id>
    <name>aliyun maven</name>
    <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
  </repository>
</repositories>
```

2. maven 依赖，在工程 pom.xml dependencies 下添加如下依赖。

```
<dependencies>
  <dependency>
    <groupId>com.aliyun.alink.linksdk</groupId>
    <artifactId>iot-linkkit-java</artifactId>
    <version>1.2.0.1</version>
    <scope>compile</scope>
  </dependency>
  <dependency>
    <groupId>com.aliyun.alink.linksdk</groupId>
    <artifactId>public-cmp-java</artifactId>
    <version>1.3.6</version>
    <scope>compile</scope>
  </dependency>
  <dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.1</version>
    <scope>compile</scope>
  </dependency>
  <dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.40</version>
    <scope>compile</scope>
  </dependency>
</dependencies>
```

JAR 依赖方式

Java版设备端SDK依赖列表如下，可以到 阿里云Maven仓库 下载并依赖。

type	groupId	artifactId	version	description
LinkKit SDK	com.aliyun.alink.linksdk	iot-linkkit-java	1.2.0.1	LinkKit API
LinkKit SDK	com.aliyun.alink.linksdk	iot-device-manager-java	1.2.0.2	device manager
LinkKit SDK	com.aliyun.alink.linksdk	public-channel-gateway-java	1.2.2	gateway
LinkKit SDK	com.aliyun.alink.linksdk	public-tmp-java	1.0.2	物模型
LinkKit SDK	com.aliyun.alink.linksdk	public-cmp-java	1.3.5	连接管理
LinkKit SDK	com.aliyun.alink.linksdk	iot-apiclient	1.0.0	https请求
LinkKit SDK	com.aliyun.alink.linksdk	network-core	1.0.0	基础网络请求库
LinkKit SDK	com.aliyun.alink.linksdk	iot-mqttclient	1.0.6	长连接
LinkKit SDK	com.aliyun.alink.linksdk	tools-java	1.0.1	-
base SDK	com.google.code.gson	gson	2.8.1	json解析库
base SDK	com.alibaba	fastjson	1.2.40	json解析库
base SDK	com.squareup.okhttp3	okhttp	3.0.1	-
base SDK	com.squareup.okio	okio	1.6.0	-
base SDK	com.aliyun.alink.linksdk	opensource-paho-java	1.2.1	Mqtt协议实现

Java SDK Demo

Java SDK提供Demo，供您参考使用。单击下载Java SDK Demo。下载本Demo将默认您同意本软件许可协议。接口定义参见 Api Reference。

2.认证与连接

本文介绍如何进行 SDK 初始化，建立设备与云端的连接。

设备认证

Java SDK目前只支持设备证书方式对设备进行身份认证，也即使用ProductKey、DeviceName、DeviceSecret来认证设备。也支持动态注册的预注册方式来动态获取DeviceSecret，但**Java SDK目前只支持在物联网平台的华东2站点进行动态注册**。

若使用动态注册，那么开发者需要先使用动态注册获取设备的DeviceSecret，将其持久化存在本地后再连接物联网平台。若设备已通过动态注册获取到了设备的DeviceSecret，那么设备重启不能再执行动态注册，而是从持久化存储中获取三元组，然后执行初始化建联。

动态注册

下面是动态注册的代码说明：LinkKit Init Params 初始化参数。

```
// ####### 一型一密动态注册接口开始 ######
/**
 * 注意：动态注册成功，设备上线之后，不能再次执行动态注册，云端会返回已注册。
 */
DeviceInfo deviceInfo = new DeviceInfo();
deviceInfo.productKey = "xx"; //必填
deviceInfo.deviceName = "xx"; //必填
deviceInfo.productSecret = "xx"; //必填
LinkKitInitParams params = new LinkKitInitParams();
IoTMqttClientConfig config = new IoTMqttClientConfig();
config.productKey = deviceInfo.productKey;
config.deviceName = deviceInfo.deviceName;
params.mqttClientConfig = config;
params.deviceInfo = deviceInfo;
final CommonRequest request = new CommonRequest();
request.setPath("/auth/register/device");
LinkKit.getInstance().deviceRegister(params, request, new IoTCallback() {
  public void onFailure(CommonRequest commonRequest, Exception e) {
    ALog.e(TAG, "动态注册失败 " + e);
  }
  public void onResponse(CommonRequest commonRequest, CommonResponse commonResponse) {
    if (commonResponse == null || StringUtils.isEmptyString(commonResponse.getData())) {
      ALog.e(TAG, "动态注册失败 response=null");
      return;
    }
    try {
      ResponseModel<Map<String, String>> response = new Gson().fromJson(commonResponse.getData(),
```

```
new TypeToken<ResponseModel<Map<String, String>>>() {
        }.getType());
        if (response != null && "200".equals(response.code)) {
          ALog.d(TAG, "动态注册成功" + (commonResponse == null ? "" : commonResponse.getData()));
          /** 获取 deviceSecret, 存储到本地，然后执行初始化建联
           * 这个流程只能走一次，获取到 secret 之后，下次启动需要读取本地存储的三元组，
           * 直接执行初始化建联，不可以再走动态初始化
           */
          // deviceSecret = response.data.get("deviceSecret");
          // init(pk,dn,ds);
          return;
        }
      } catch (Exception e) {
      }
      ALog.d(TAG, "动态注册失败" + commonResponse.getData());
    }
});
// ####### 一型一密动态注册接口结束 ######
```

SDK 初始化

SDK初始化，即使用设备的证书连接物联网平台。设备如果初始化失败，比如网络不通导致连接失败，根据业务场景开发者可以选择是否再次进行初始化连接物联网平台，SDK此时不会自动尝试连接物联网平台；若SDK初始化成功之后，设备与云端的连接异常断开，那么SDK将负责自动重新连接物联网平台。onInitDone表示初始化成功，onError表示初始化失败。

设备可以接入物联网平台的公共实例或者企业实例，SDK初始化时需要根据实例类型来指定设备连接的域名：

- 公共实例

  公共实例的域名类似：{productKey}.iot-as-mqtt.***RegionID***.aliyuncs.com"，其中的***RegionID***即代表相应的站点，其取值可以从“地域和可用区”中查看，需要注意的是物联网平台这个服务并不是所有的阿里云地域都进行了部署，因此这里填入的RegionID一定要和产品创建的地域相同，比如“华东2（上海）”的RegionID是cn-shanghai，假设产品的productKey是abc，那么最终的域名为：abc.iot-as-mqtt.cn-shanghai.aliyuncs.com。SDK中的默认域名为华东2，若设备接入阿里云华东2站点，那么可以不用指定域名。

- 企业实例

用户需要在企业实例中查看实例的接入URL，请参见文档“实例管理”中的“查看实例终端节点”章节了解如何查看实例详情，其中MQTT接入的域名如下图所示，用户无需在域名前添加productKey：

下面的代码演示设备接入华东2站点时SDK初始化的代码：

```
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;
/*
 *是否接受离线消息
 *对应 mqtt 的 cleanSession 字段
 */
config.receiveOfflineMsg = false;
params.mqttClientConfig = config;
/**
 *设置初始化三元组信息，用户传入
 */
DeviceInfo deviceInfo = new DeviceInfo();
deviceInfo.productKey = pk;
deviceInfo.deviceName = dn;
deviceInfo.deviceSecret = ds;
params.deviceInfo = deviceInfo;
/**
 * 设置设备当前的初始状态值，属性需要和云端创建的物模型属性一致
 *如果这里什么属性都不填，物模型就没有当前设备相关属性的初始值。
 *用户调用物模型上报接口之后，物模型会有相关数据缓存。
*/
Map propertyValues = new HashMap(); // 示例// propertyValues.put(“LightSwitch”, new ValueWrapper.Boo
leanValueWrapper(0));params.propertyValues = propertyValues;
LinkKit.getInstance().init(params, new ILinkKitConnectListener() {
  public void onError(AError aError) {
    ALog.e(TAG, “Init Error error = ”+aError);
  }
  public void onInitDone(InitResult initResult) {
    ALog.i(TAG, "onInitDone result=" + initResult);
  }
});
```

若设备连接的不是华东2站点的公共实例，而是连接企业实例或者其它站点的公共实例，那么需要对连接的实例的域名进行设置：

```
// 设置 Mqtt 请求域名 LinkKitInitParams 初始化参数
IoTMqttClientConfig clientConfig = new IoTMqttClientConfig();
clientConfig.channelHost = "xxx";
linkKitInitParams.mqttClientConfig = clientConfig;
```

对于使用企业实例的客户，我们提供了demo用于演示设备如何连接企业实例，客户下载demo之后需要在device_id.json中文件中修改endpoint字段为自己实例的MQTT接入的域名。

SDK 反初始化

如果需要注销初始化，调用如下接口。

```
// 取消注册 notifyListener，notifyListener对象需和注册的时候是同一个对象
LinkKit.getInstance().unRegisterOnNotifyListener(notifyListener);
LinkKit.getInstance().deinit();
```

其他设置

日志开关

打开SDK内部日志输出开关：

```
ALog.setLevel(ALog.LEVEL_DEBUG);
```

连接状态监听

如果需要监听设备的上下线信息，云端下发的所有数据，可以设置以下监听器。

```
IConnectNotifyListener notifyListener = new IConnectNotifyListener() {
  @Override
  public void onNotify(String connectId, String topic, AMessage aMessage) {
    // 云端下行数据回调
    // connectId 连接类型 topic 下行 topic; aMessage 下行数据
    //String pushData = new String((byte[]) aMessage.data);
    // pushData 示例 {"method":"thing.service.test_service","id":"123374967","params":{"vv":60},"version":
"1.0.0"}
    // method 服务类型； params 下推数据内容
}
  @Override
  public boolean shouldHandle(String connectId, String topic) {
    // 选择是否不处理某个 topic 的下行数据
    // 如果不处理某个topic，则onNotify不会收到对应topic的下行数据
    return true; //TODO 根基实际情况设置
  }
  @Override
  public void onConnectStateChange(String connectId, ConnectState connectState) {
    // 对应连接类型的连接状态变化回调，具体连接状态参考 SDK ConnectState
  }
}
// 注册下行监听，包括长连接的状态和云端下行的数据
LinkKit.getInstance().registerOnNotifyListener(notifyListener);
```

Mqtt 连接参数

- 设置Mqtt Keep-Alive 时间

```
interval 单位秒 MqttConfigure.setKeepAliveInterval(int interval);
```

```
- qos设置
MqttPublishRequest request = new MqttPublishRequest();
// 支持 0 和 1， 默认0
request.qos = 0;
request.isRPC = false;
request.topic = topic.replace("request", "response");
String resId = topic.substring(topic.indexOf("rrpc/request/")+13);
request.msgId = resId;
// TODO 用户根据实际情况填写 仅做参考
request.payloadObj = "{\"id\":\"" + resId + "\", \"code\":\"200\"" + ",\"data\":{} }";
```

- cleanSession 设置

```
/**
 * 设置 Mqtt 初始化参数
 */
IoTMqttClientConfig config = new IoTMqttClientConfig();
config.productKey = deviceInfoData.productKey;
config.deviceName = deviceInfoData.deviceName;
config.deviceSecret = deviceInfoData.deviceSecret;
config.channelHost = pk + ".iot-as-mqtt." + deviceInfoData.region + ".aliyuncs.com:1883";
/**
 * 是否接受离线消息
 * 对应 receiveOfflineMsg = !cleanSession, 默认不接受离线消息
 */
config.receiveOfflineMsg = false;
params.mqttClientConfig = config;
```

3.自定义MQTT Topic通信

Link SDK 提供了与云端长链接的基础能力接口，用户可以直接使用这些接口完成自定义 Topic 相关的功能。提供的基础能力包括：发布、订阅、取消订阅、RRPC、订阅下行。

调用上行请求接口，SDK 封装了上行Publish请求、订阅Subscribe和取消订阅unSubscribe等接口。

```
/**
 * 发布
 *
 * @param request  发布请求
 * @param listener 监听器
 */
void publish(ARequest request, IConnectSendListener var2);
/**
 * 订阅
 *
 * @param request   订阅请求
 * @param listener 监听器
 */
void subscribe(ARequest request, IConnectSubscribeListener var2);
/**
 * 取消订阅
 *
 * @param request 取消订阅请求
 * @param listener 监听器
 */
void unsubscribe(ARequest request, IConnectUnscribeListener var2);
```

调用示例：MqttPublishRequest 类路径参见 com.aliyun.alink.linksdk.cmp.connect.channel.MqttPublishRequest。

```
// 发布
MqttPublishRequest request = new MqttPublishRequest();
// topic 用户根据实际场景填写
request.topic = "/sys/" + pk + "/" + dn + "/thing/deviceinfo/update";
/**
 * 订阅回复的 replyTopic
 * 如果业务有相应的响应需求，可以设置 replyTopic，且 isRPC=true
 */
//     request.replyTopic = request.topic + "_reply";
/**
```

```
/
 * isRPC = true; 表示先订阅 replyTopic，然后再发布；
 * isRPC = false; 不会订阅回复
 */
//     request.isRPC = true;
/**
 * 设置请求的 qos
 */
request.qos = 0;
// 更新标签 仅做测试
// payloadObj 替换成用户需要发布的数据 json String
//示例 属性上报 {"id":"160865432","method":"thing.event.property.post","params":{"LightSwitch":1},"versi
on":"1.0"}
request.payloadObj = "{\"id\":2, \"params\":{\"version\":\"1.0.0\"}}";
LinkKit.getInstance().publish(request, new IConnectSendListener() {
  @Override
  public void onResponse(ARequest aRequest, AResponse aResponse) {
    // publish 结果
    ALog.d(TAG, "onResponse " + (aResponse==null?"":aResponse.data));
  }
  @Override
  public void onFailure(ARequest aRequest, AError aError) {
    // publish 失败
    ALog.d(TAG, "onFailure " + (aError==null?"":(aError.getCode()+aError.getMsg())));
  }
});
// 订阅
MqttSubscribeRequest request = new MqttSubscribeRequest();
// topic 用户根据实际场景填写
request.topic = "/sys/" + pk + "/" + dn + "/thing/deviceinfo/update";
request.isSubscribe = true;
LinkKit.getInstance().subscribe(request, new IConnectSubscribeListener() {
  @Override
  public void onSuccess() {
    // 订阅成功
    ALog.d(TAG, "onSuccess ");
  }
  @Override
  public void onFailure(AError aError) {
    // 订阅失败
    ALog.d(TAG, "onFailure " + (aError==null?"":(aError.getCode()+aError.getMsg())));
```

```
    }
});
// 取消订阅
MqttSubscribeRequest request = new MqttSubscribeRequest();
// topic 用户根据实际场景填写
request.topic = "/sys/" + pk + "/" + dn + "/thing/deviceinfo/update";
request.isSubscribe = false;
LinkKit.getInstance().unsubscribe(request, new IConnectUnscribeListener() {
    @Override
    public void onSuccess() {
        // 取消订阅成功
        ALog.d(TAG, "onSuccess ");
    }
    @Override
    public void onFailure(AError aError) {
        // 取消订阅失败
        ALog.d(TAG, "onFailure " + (aError==null?"":(aError.getCode()+aError.getMsg())));
    }
});
```

下行数据监听

下行数据监听可以通过 RRPC 方式或者注册一个下行数据监听器实现。

```
/**
 * RRPC 接口
 * RRPC：先订阅 topic A；云端需要的时候调用设备的服务，通过 topic A下发数据；设备收到数据，回复云端；
 * 另外一种方式是：单独调用订阅接口，然后在 registerOnNotifyListener 接收对应 topic 的下行数据，
 * 并回复云端；
 * @param topic   订阅 topic
 * @param listener 监听器
 */
void registerResource(AResource var1, IResourceRequestListener var2);
/**
 * 注册下行数据监听器,所有已订阅的 topic 下行数据都会在这里返回
 *
 * @param listener 监听器
 */
void registerOnNotifyListener(IConnectNotifyListener listener);
/**
 * 取消注册下行监听器
 *
 * @param listener 监听器
 */
void unRegisterOnNotifyListener(IConnectNotifyListener listener);
```

调用示例：

```
/**
 * 下行数据接收&处理
 * 设备连接状态变化
 */
private IConnectNotifyListener notifyListener = new IConnectNotifyListener() {
  public void onNotify(String connectId, String topic, AMessage aMessage) {
    // 云端下行数据通知
  }
  public void onConnectStateChange(String connectId, ConnectState connectState) {
    // 设备连接状态通知
  }
  public boolean shouldHandle(String connectId, String topic){
    return true; // 根据实际场景设置
  }
};
/**
 * 所有topic的下行数据入口（前提是先订阅了该 topic，才会在这里收到）
 * 如果想要在这里收到对应 topic 的下行数据，需要先订阅该 topic
 */
public void registerNotifyListener(){
  LinkKit.getInstance().registerOnNotifyListener(notifyListener);
}
/**
 * 取消注册下行的监听器，该 listener 需要保持和注册的 listener 是同一个对象
 */
public void unregisterNotifyListener() {
  LinkKit.getInstance().unRegisterOnNotifyListener(notifyListener);
}
```

RRPC 调用示例：

```
final CommonResource resource = new CommonResource();
resource.topic = "/ext/rrpc/+/" + productKey + "/" + deviceName + "/get";
resource.replyTopic = resource.topic;
LinkKit.getInstance().registerResource(resource, new IResourceRequestListener() {
 @Override
 public void onHandleRequest(AResource aResource, ResourceRequest resourceRequest, IResourceRespon
seListener iResourceResponseListener) {
  // 收到云端数据下行
  ALog.d(TAG, "onHandleRequest aResource=" + aResource + ", resourceRequest=" + resourceRequest + ", iR
```

```
esourceResponseListener=" + iResourceResponseListener);
  // 下行数据解析示例
  //          String downstreamData = new String((byte[]) resourceRequest.payloadObj);
  // 示例 {"id":"269297015","version":"1.0","method":"thing.event.property.post","params":{"lightData":{"v
v":12}}}
  // 如果数据是json，且包含id字段，格式可以按照如下示例回复，传输数据请根据实际情况定制
  //          if (aResource instanceof  CommonResource) {
  //            ((CommonResource) aResource).replyTopic = resourceRequest.topic;
  //          }
  //          if (iResourceResponseListener != null) {
  //            AResponse response = new AResponse();
  //
  //            response.data = "{\"id\":\"123\", \"code\":\"200\"" + ",\"data\":{} }";
  //            iResourceResponseListener.onResponse(aResource, resourceRequest, response);
  //          }
  // 如果不一定是json格式，可以参考如下方式回复
  MqttPublishRequest rrpcResponse = new MqttPublishRequest();
  rrpcResponse.topic = resourceRequest.topic;
  rrpcResponse.payloadObj ="xxx";
  LinkKit.getInstance().publish(rrpcResponse,null);
 }
 @Override
 public void onSuccess() {
  // 注册资源成功
  ALog.d(TAG, "onSuccess ");
 }
 @Override
 public void onFailure(AError aError) {
  // 注册资源失败
  ALog.d(TAG, "onFailure " + getError(aError));
 }
});
```