阿里云物联网套件

快速开始

为了无法计算的价值 | 阿里云

快速开始

本文以Linux下C语言版SDK为例, 演示如何快速体验让设备连接到阿里云IoT, 并通过MQTT协议的PUB/SUB实现一个简单的M2M通信过程.

详细说明请访问官方WiKi和官方SDK首页

第一步: 在控制台中创建设备

登录IoT控制台, 创建产品及在产品下创建设备和Topic类, 具体步骤如下:

- 创建产品, 可得到ProductKey
- 在产品下创建设备, 可得到DeviceName, DeviceSecret
- 定义Topic: $(PRODUCT_KEY)/$(DEVICE_NAME)/data, 并设置权限为: 设备具有发布与订阅 **(此步骤非常重要)**

具体请参考控制台使用手册文档中的创建产品, 添加设备以及获取设备Topic部分.

第二步: 填充设备参数

> **备注:**如果您还没有SDK源码, 请到SDK下载页面, 下载最新版本.

将sample程序文件中的设备参数替换为您在控制台申请到的设备参数.

将 sample/mqtt/mqtt-example.c 中以下星号字符串表示的设备参数替换为第一步中获得的值:

```
#if defined(MQTT_ID2_AUTH) && defined(TEST_ID2_DAILY)
#define PRODUCT_KEY "********************"
#define DEVICE_NAME "********************"
#define DEVICE_SECRET "********************"
#else
// TODO: 在以下段落替换下列宏定义为你在IoT控制台申请到的设备信息
#define PRODUCT_KEY "********************"
#define DEVICE_NAME "********************"
#define DEVICE_SECRET "********************"
#endif
```

完成编辑并保存后, 进行下一步

第三步: 编译SDK

- 返回顶层目录

执行make指令, 编译SDK, 命令如下

```
make distclean
make
```

编译成功后, 在相应目录生成样例可执行程序.

第四步: 执行样例程序

执行目录 output/release/bin/ 下的可执行程序:

```
cd output/release/bin
./mqtt-example
```

样例程序的基本逻辑流程为:

1. 创建一个MQTT客户端
2. 订阅主题 $(PRODUCT_KEY)/$(DEVICE_NAME)/data
3. 向该主题发布消息

其它

编译输出的说明

编译顺利完成后, 输出在 output/release/ 目录:

```
+-- bin
| +-- coap-example
| +-- ...
| +-- mqtt-example
+-- include
| +-- exports
| | +-- iot_export_coap.h
| | +-- ...
| | +-- iot_export_shadow.h
| +-- imports
| | +-- iot_import_coap.h
| | +-- ...
| | +-- iot_import_ota.h
| +-- iot_export.h
| +-- iot_import.h
```

```
+-- lib
| +-- libiot_platform.a
| +-- libiot_sdk.a
| +-- libiot_tls.a
+-- src
+-- coap-example.c
+-- http-example.c
+-- Makefile
+-- mqtt-example.c
```

说明如下:

文件/目录	说明
bin/mqtt-example	用MQTT协议连接阿里云IoT的样例程序
include/	使用libiot_sdk.a时需要的头文件, 存放在这个目录
lib/libiot_platform.a	硬件平台抽象层, libiot_sdk.a的工作是建立在它的基础上的
lib/libiot_sdk.a	SDK的核心层, 基于libiot_platform.a提供连接云端的能力
src/Makefile	示例用Makefile, 演示如何在SDK之外链接本SDK提供的库文件
src/mqtt-example.c	样例对应C代码, 可在此目录下输入make, 编译生成可执行程序

Java版本

Java版请参考 JAVA-SDK使用(MQTT)

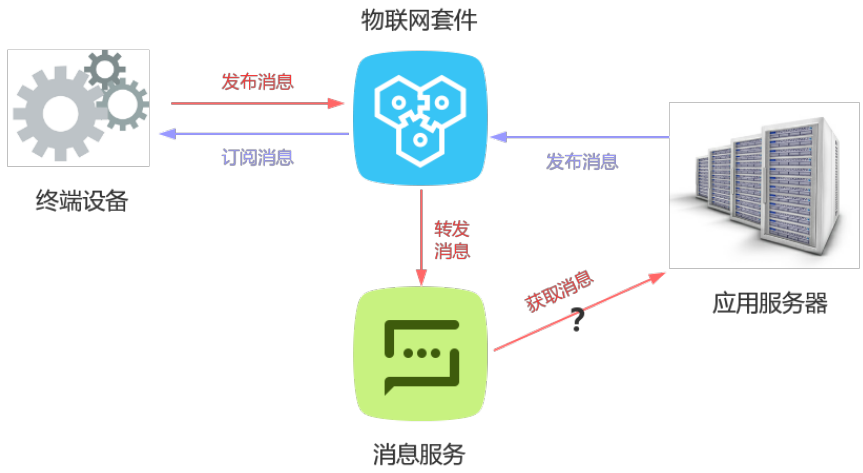
服务器接入的场景：

1. 你的服务器端需要快速的接收设备发送的消息
2. 你的服务器需要实时接收设备上下线的通知消息
3. 你的服务器需要通过SDK 来主动发送消息给设备 这部分参考API接入

本文针对上述1和2的场景快速接入说明：

物联网套件核心功能就是设备连接和设备通信，简单理解就是物联网套件充当了设备网关的角色，帮助设备实现与云端的双向通信。

物联网套件配合消息服务MNS来提供服务端订阅设备消息的功能，这种方式优势是MNS可以保证消息的可靠性，避免了服务端不可用时的消息丢失，同时MNS在处理大量消息并发时有削峰填谷的作用，保证服务端不会因为突然的并发压力导致服务不可用。这篇文档主要讲述的就是当设备的数据发送到物联网套件之后，用户的服务端如何获取设备的数据。就是下图中？的链路：



物联网套件主要提供两种方式，让用户的服务端获取设备数据。

- 用户简单配置，物联网套件会将设备的数据透传写入**MNS队列**中，用户服务端从队列中获取数据
- 用户可以使用物联网套件的规则引擎对设备数据进行计算处理，再利用规则引擎将数据转发到**MNS主题**中，然后用户服务端从主题中获取设备数据。

> 队列模型支持一对一发送和接收消息；
>
> 主题模型支持一对多发布和订阅消息，并且支持多种消息推送方式；

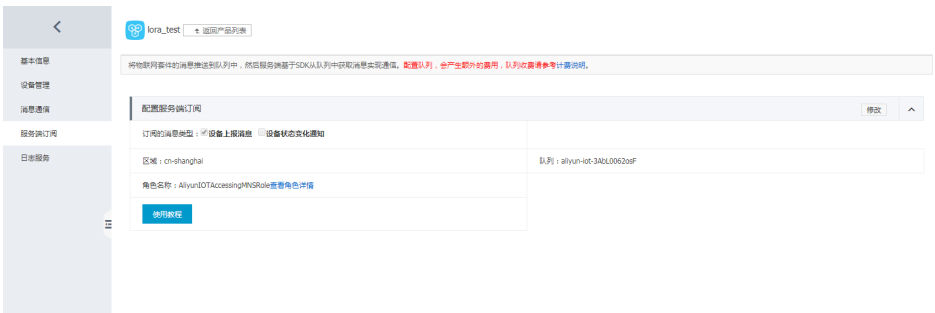
本文档主要介绍队列方式，至于主题的那种使用方式请参考规则引擎MNS文档。

使用步骤

1. 控制台配置服务端订阅

选择推送给服务端订阅的消息类型，请参考**控制台服务端订阅使用文档**。配置完成后IoT会自动在MNS华东2区域下创建aliyun-iot-${productKey}队列。

> 特别说明：IoT创建的队列命名规则是aliyun-iot-${productkey}，${productkey}是该产品的productkey。更新配置后大概要1分钟生效。



名词解释：

- 订阅的消息类型:

1. 设备上报消息:指的是产品下所有设备Topic列表中具有发布权限的Topic中的消息，例如产品下面有三个Topic类，其中有/pk/${deviceName}/get:订阅、/pk/${deviceName}/update:发布、/pk/${deviceName}/update/error:发布。那么设备上报消息指的是，/pk/${deviceName}/update和/pk/${deviceName}/update/error对应的所有Topic中的消息。选中后保存，系统会把这些Topic中的消息写到上面默认创建的MNS队列里
2. 设备状态变化通知:指的是一旦该产品下的设备状态变化时，例如上线，下线，套件产生的消息。选中后保存，系统会推送设备上下线消息到上面默认创建的MNS队列里

- 区域:IoT默认在该区域创建Queue,下面在使用MNS SDK获取Endpoint 需要在MNS控制台选择该区域（region）
- 队列: IoT会自动到MNS华东2下创建aliyun-iot-${productKey}队列。具体可以到MNS控制台查看。
- 角色名称：用户授权IoT访问用户MNS系统的角色，IoT系统根据这个授权角色写入消息到用户的MNS消息队列，否则消息无权写入。

配置完成后，iot套件会对设备数据做一层封装，以JSON的格式写入queue中，消息格式如下：

```
{
"messageid":"12345",
"messagetype":"status/upload",
"topic":"null/topic",
"payload": {data},
"timestamp": 1469564576
}
```

- messageid：IoT套件生成的消息ID，64位大小
- messagetype：指的是消息类型，目前包括设备状态通知和设备上报消息，分别对应status和upload
- topic：表示该消息源自套件中的哪个topic，当messageType=status时，topic为null，当messageType=upload时，topic为具体的设备Topic，例如/pk/mydevice/update
- payload：**数据为Base64 Encode的数据。**当messageType=status时，数据是套件的通知数据，具体格式见下文；当messageType=upload时，data即为设备发布到Topic中的原始数据。
- timestamp：时间戳，以Epoch时间表示

对应的设备状态数据格式参考:

设备上下线通知

data为JSON字符串，格式如下：

```
data=
{
"status":"online"(或offline),//设备状态
"productKey":"xxxxxxxxxx",//产品标识
```

```
"deviceName":"xxxxxxxxxx",//设备标识
"time":"2017-10-11 10:11:12.234", //发送通知时间点
"lastTime":"2017-10-11 10:11:12.123" //状态变更时最后一次通信时间
"clientIp":"xxx.xxx.xxx.xxx" //设备端公网出口IP
}
```

2. 用户服务端接入MNS的SDK，使用queue模式订阅IoT默认创建的消息队列aliyun-iot-${productKey}

服务端接收消息代码示例：

以MNS的JAVA版SDK使用为例，在工程的pom.xml文件里添加如下依赖：

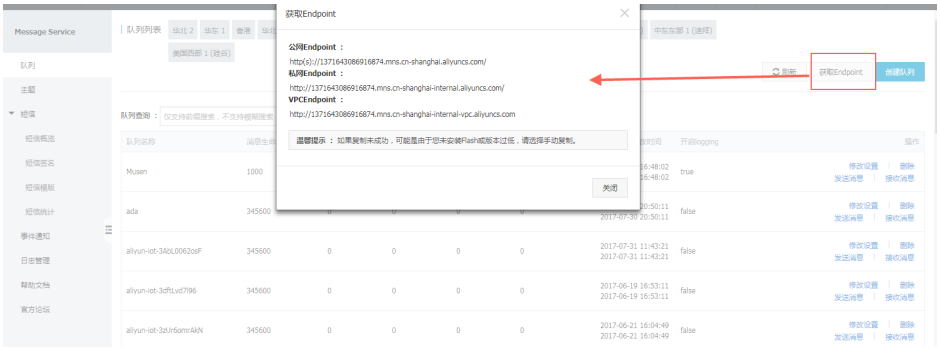
```
<dependency>
<groupId>com.aliyun.mns</groupId>
<artifactId>aliyun-sdk-mns</artifactId>
<version>1.1.5</version>
</dependency>
```

接收mns消息的代码：

```
CloudAccount account = new CloudAccount(
$AccessKeyId,
$AccessKeySecret,
$AccountEndpoint);
```

- AccessKeyId和AccessKeySecret是和阿里云账号绑定的，在下图所示位置获取：

- AccountEndpoint可以在MNS的控制台上获取，因为IoT是在**华东1**区域创建队列，所以请选择**华东1**获取Endpoint，如下图所示：




```
MNSClient client = account.getMNSClient();
CloudQueue queue = client.getQueueRef("aliyun-iot-11111111111"); //参数请输入IoT自动创建的队列名称，例如上面截图中的aliyun-iot-3AbL0062osF

while (true) {
```

```
// 获取消息
Message popMsg = queue.popMessage(10); //长轮询等待时间为10秒
if (popMsg != null) {
System.out.println("PopMessage Body: "
+ popMsg.getMessageBodyAsRawString()); //获取原始消息
queue.deleteMessage(popMsg.getReceiptHandle()); //从队列中删除消息
} else {
System.out.println("Continuing");
}
}
```

启动程序完成对MNS队列的监听

其他版本SDK请参考MNS官方文档

3.发送消息，查看您的服务器是否正常接收

模拟设备发送消息

设备接入请参考设备端快速接入(MQTT)

下发指令给设备

服务端对指定的Topic发消息，设备订阅该Topic接收指令。具体请参考两个接口，RRPC和Pub，根据自己业务情况选择。