SNoti API (V2.1.2.1)

Demo

https://github.com/gizwits/noti-java-demo/tree/v2.0.0-netty

目的

为企业提供 SSL 通讯 API，用于实时推送设备与产品相关的事件。

准备工作

1.一次可接受多个 Product 的消息。每个 Product 需要 auth_id 和 auth_secret 验证。

2.针对每一个 product，都需要新创建一个唯一的 auth_id：

- a.对产品拥有者，auth_id 和 auth_secret 需要向机智云获取，该 auth_id 拥有获取该产品下所有设备消息和控制设备的权限；
- b.对第三方运营商，通过使用 Http API 获取 auth_id 和 auth_secret，获取后，还需要通过 Http API 对已拥有的设备做关联，关联成功后，才能够获取设备消息和控制设备；
- c.第三方运营商申请 auth_id api:https://m2mv4.iotsdk.com:2018/v1/doc#!/product/post_v1_products_product_key_operator
- d.第三方运营商关联设备 api:https://m2mv4.iotsdk.com:2018/v1/doc#!/product/put_v1_products_product_key_operator
- e.第三方运营商取消关联设备 api:https://m2mv4.iotsdk.com:2018/v1/doc#!/product/delete_v1_products_product_key_operator

3.以 product_key+subkey 为唯一主键。其中 subkey(subscription key)为自定义字符串，大小写敏感，长度为 1 到 32 个字符，可包含数字，字母和下划线（即[a-zA-Z0-9]）。

4.选取接收信息的 product_key 下的消息类型。一个 product_key 可支持多种消息类型。目前有 7 种，见推送事件消息字段的 event_type。

过程描述

事件通过 SSL 接口推送。通讯过程如下：

1.客户端以 Client 的身份与本接口（Gizwits Platform）建立 SSL 连接。客户端无需提供证书，只需要信任服务器证书即可；

2.客户端发送登录指令完成验证；

3.客户端实时接受事件消息，并向服务器 ack 事件消息；

4.当客户端在一定时间范围内没有向服务器发送任何消息，需要发 ping 心跳请求，服务器回复 pong 心跳响应。

消息推送

服务端推送消息到客户端。推送工作方式：

1.相同 product_key + subkey 的多个客户端登录连接，消息轮流推送到各客户端；

2.相同 product_key，不同 subkey 的客户端都能接收该 product 下设备的指定类型的消息副本，客户端之间不会相互干扰；

3.当 某客户端未返回的 ack 消息数达到该客户端登录设置的 prefetch_count 值后，消息将不在继续推送给该客户端，但会发送到其他客户端；

4.未返回 ack 的消息，服务端会一直等待不会重发，只有在该客户端断开的情况下，未 ack 的消息会重新发送到连接的客户端；

5.当所有客户端都断开连接后，后续设备的消息会保存在服务端，等待客户端下次接收。

服务地址

- 域名：snoti.gizwits.com
- SSL 服务端口：2017
- HTTP API 服务端口：2018

SSL 的接口协议 的接口协议

消息内容为二进制数据，UTF-8 编码。请注意每个消息后都必须添加”\n”作为消息结尾符。

1. 连接与登陆

客户端和 Gizwits Platform 建立 SSL 连接后，客户端发送以下字符串内容作身份验证(登陆Gizwits Platform)：

```
{
"cmd": "login_req",
"prefetch_count":<uint> (0 < prefetch_count <= 32767, 表示推送没有 ACK 的消息
的最大个数，可不填，默认值是 50)
"data": [{
“product_key": <key string>,
"auth_id": <auth_id string>,
"auth_secret": <auth_secret string>,
"subkey": <subkey string>,
"events":[<event string>,...] (可一个或多个 event)
},...] (可一个或多个 product)
}\n
```

请求字段说明：

字段	是否必须	描述
cmd	必须	登录类型，必须为login_req
prefetch_count	非必须	默认值为50
data.product_key	必须	产品ID
data.auth_id	必须	产品授权ID
data.auth_secret	必须	产品授权密匙
data.subkey	必须	subscription key，为客户端自定义标识，大小写敏感，长度为 1 到 32 个字符，可包含数字，字母和下划线
data.events	必须	客户端接收消息类型，使用逗号隔开的字符串列表，目前支持类型 为 device.attr_fault;device.attr_alert;device.online;device.offline device.status.raw;device.status.kv;datapoints.changed

Gizwits Platform 回复：

```
{
"cmd": "login_res",
"data": {
"result": true | false,
"msg": "ok" | <error msg>
}
}\n
```

如 result 为 false 表示登陆失败，该连接会被关闭。

2. 心跳

客户端和 Gizwits Platform 建立 SSL 连接后，需定期向 Gizwits Platform 发送数据以保持连接的有效性，如 5 分钟内没有向 Gizwits Platform 发送任何数据，应向 Gizwits Platform 发送以下的心跳数据：

```
{
"cmd": "ping"
}\n
```

Gizwits Platform 回复：

```
{
"cmd": "pong"
}\n
```

如没有及时收到 Gizwits Platform 的回复，可以认为与 Gizwits Platform 的连接已关闭，需重连。

3. 控制设备

客户端和 Gizwits Platform 建立 SSL 连接后，客户端可发送以下字符串内容，控制设备：

```
{
"cmd": "remote_control_req",
"data": [{
“cmd": "write_attrs" | "write" | "write_v1",
"source": "noti",
"data": {
"did": <did string>,
"mac": <mac string>,
"product_key": <product_key string>,
"attrs": {
"name1": <value1>,
"name2": <value2>,
… …
}(Only used with cmd "write_attrs")
OR
"raw": [<byte>, <byte>, <byte>, … ...](Only used with cmd “write“ or “write_v1”)
}
},...] (可一个或多个控制指令)
}\n
```

请求字段说明：

字段	是否必须	描述
cmd	必须	控制设备，必须为 remote_control_req
data	必须	控制指令，数组类型
data.cmd	必须	V4 产品数据点协议格式，填写write_attrs；V4 产品自定义协议格式，填写 write；V1 产品协议格式，填写 write_v1
data.source	必须	固定填写 noti
data.data.did	必须	设备 ID

字段	是否必须	描述
data.data.mac	必须	设备 Mac 地址,长度为 12 的字符串，大小写敏感
data.data.product_key	必须	设备所属产品的标识码
data.data.attrs / data.data.raw	必须	V4 产品数据点协议格式，选择data.data.attrs；V4 产品自定义协议格式，选择data.data.raw；V1 产品协议格式，选择 data.data.raw

Gizwits Platform 回复：

```
{
"cmd": "remote_control_res”,
“result": {
"succeed": ["did1", "did2", … …]
"failed": [
{"did3": <reason str>}
{"did4": <reason str>}
… …
]
}
}\n
```

如协议自身引起的错误，Gizwits Platform 回复 错误响应消息，该消息格式参见下文。

4. 推送事件

数据点编辑事件

```
{
"cmd": "event_push",
"delivery_id": <delivery_id>, (用于 ACK)
"event_type": "datapoints_changed",
"product_key": <product_key string>,
"created_at": <timestamp in seconds, float>
}\n
```

设备上线事件

```
{
"cmd": "event_push",
"delivery_id": <delivery_id>, (用于 ACK)
"event_type": "device_online",
"product_key": <product_key string>,
"did": <did string>,
"mac": <mac string>,
"group_id": <group_id string>,
"created_at": <timestamp in seconds, float>,
"ip": <ip string>,
"country": <country string>,
"region": <region string>,
"city": <city string>
}\n
```

设备下线事件

```
{
"cmd": "event_push",
"delivery_id": <delivery_id>, (用于 ACK)
"event_type": "device_offline",
"product_key": <product_key string>,
"did": <did string>,
"mac": <mac string>,
"group_id": <group_id string>,
"created_at": <timestamp in seconds, float>
}\n
```

故障与报警事件

当有故障与报警事件发生，Gizwits Platform 会向客户端推送以下消息：

```
{
"cmd": "event_push",
"delivery_id": <delivery_id>, (用于 ACK)
"event_type": "attr_fault" | "attr_alert",
"event_id": <uuid string>, (同一设备的同一故障或报警的发生事件与恢复事件共享同一事件 id)
"product_key": <product_key string>,
"did": <did string>,
"mac": <mac string>,
"group_id": <group_id string>,
"created_at": <timestamp in seconds, float>,
"data": {
"attr_name": <故障或报警数据点标识名>,
"attr_displayname": <故障或报警数据点显示名称>
"value": 0 | 1 (0 表示从故障恢复或报警取消，1 表示发生了故障或报警)
}
}\n
```

设备状态事件

```
{
"cmd": "event_push",
"delivery_id": <delivery_id>, (用于 ACK)
"event_type": "device_status_raw",
"product_key": <product_key string>,
"did": <did string>,
"mac": <mac string>,
"group_id": <group_id string>,
"created_at": <timestamp in seconds, float>,
"data": <base64 encoding string> (设备状态原始数据 base64 编码字符串)
}\n
```

如该产品支持数据点解释(机智云通用数据点协议或自定义数据点协议)，则消息格式为message format:

```
{
"cmd": "event_push",
"delivery_id": <delivery_id>, (用于 ACK)
"event_type": "device_status_kv",
"product_key": <product_key string>,
"did": <did string>,
"mac": <mac string>,
"group_id": <group_id string>,
"created_at": <timestamp in seconds, float>,
"data": {
<key1 string>: <value1>,
<key2 string>: <value2>,
...
}
}
```

事件 ACK

客户端每收到一事件消息都需要回复以下 ACK 消息：

```
{
"cmd": "event_ack",
"delivery_id": <delivery_id> (按原数据类型填写)
}\n
```

如 Gizwits Platform 没有收到客户端的回复，该事件会重复推送直至收到 ACK 或过期为止。一般情况下事件按发生时间顺序推送，当prefetch_count > 1 时，有可能因某事件没有 ACK而导致重发而表现为事件没有按时间顺序到达，例如先收到故障恢复或报警取消的事件，所以客户端开发者在处理时应比较 event_id 和 created_at 的值。

5. 错误响应

当客户端发送的消息 Gizwits Platform 不能识别时，Gizwits Platform 返回以下的消息：

```
{
"cmd": "invalid_msg"
“error_code”: <code int>
"msg": <msg string>
}\n
```

客户端开发者应检查客户端发出的消息内容是否正确（注意：如果客户端没有在消息结尾添加”\n”，服务器端会认为还没有收到完整的消息而继续等待更多的消息内容）。