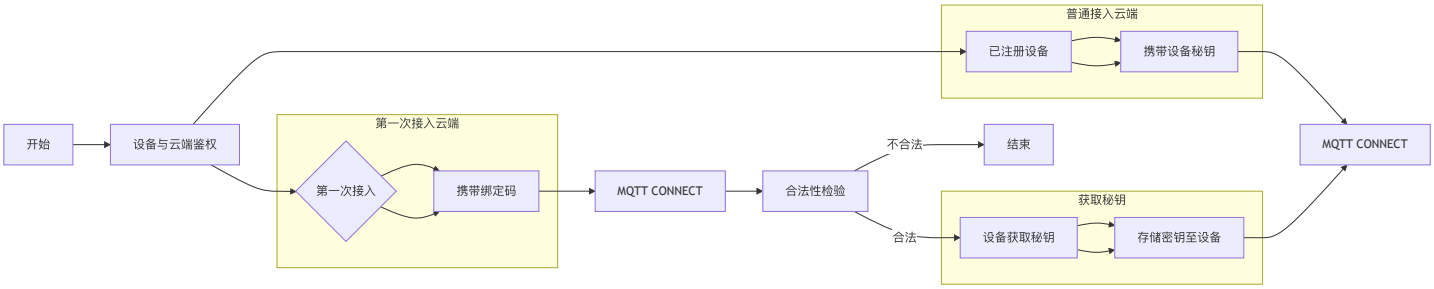
[20240808]M1设备端MQTT接口和交互过程

本文根据功能列出云端和设备端之间的全部 MQTT 接口.

业务流程

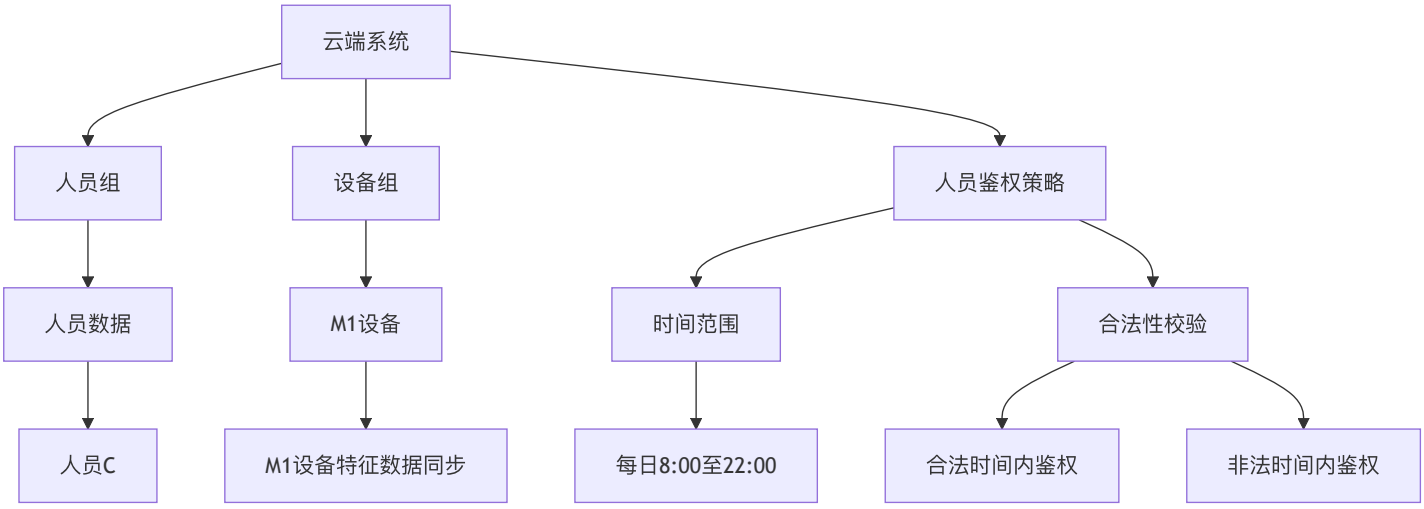
设备与云端建立联系

- 所有与云端尝试建立连接的设备,都需要鉴权;
  - 第一次建立连接,设备需要携带由云端生成的绑定码以及设备侧本地生成的设备UUID请求,云端通过读取此绑定码,确定此设备UUID的接入是合法的;
    - 检验合法后,设备端与云端建立了合法的 MQTT 通讯,需要第一时间通过设备获取秘钥接口获取由云端生成的秘钥,存储到设备本地存储;
    - 后续就可以使用此秘钥,使用设备普通接入云端的接口,尝试与云端建立合法的MQTT连接;
  - 如果设备已经注册,后续的连接请求,都需要携带已经保存的设备秘钥尝试与云端建立合法的MQTT连接;
- 建立连接的过程使用 MQTT 协议的 CONNECT 方法;
- 设备获取秘钥接口需要在云端与设备端建立合法的 MQTT 的通讯之后,才能被合法访问;
  - 云端需要记录设备UUID以及下发的秘钥信息,以便后续的鉴权;
- 通过设备UUID标记每一个与云端建立联系的设备,为后续的设备组以及人员组的业务逻辑打下基础.

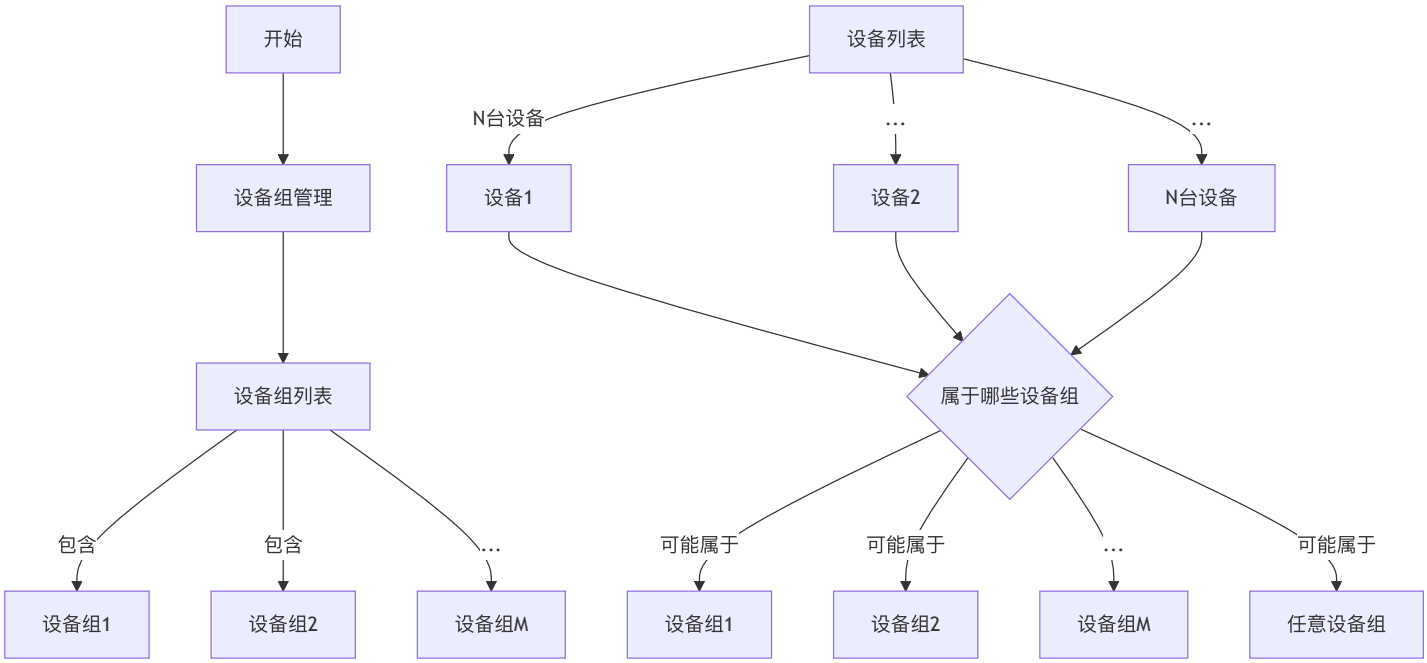


设备组与人员组

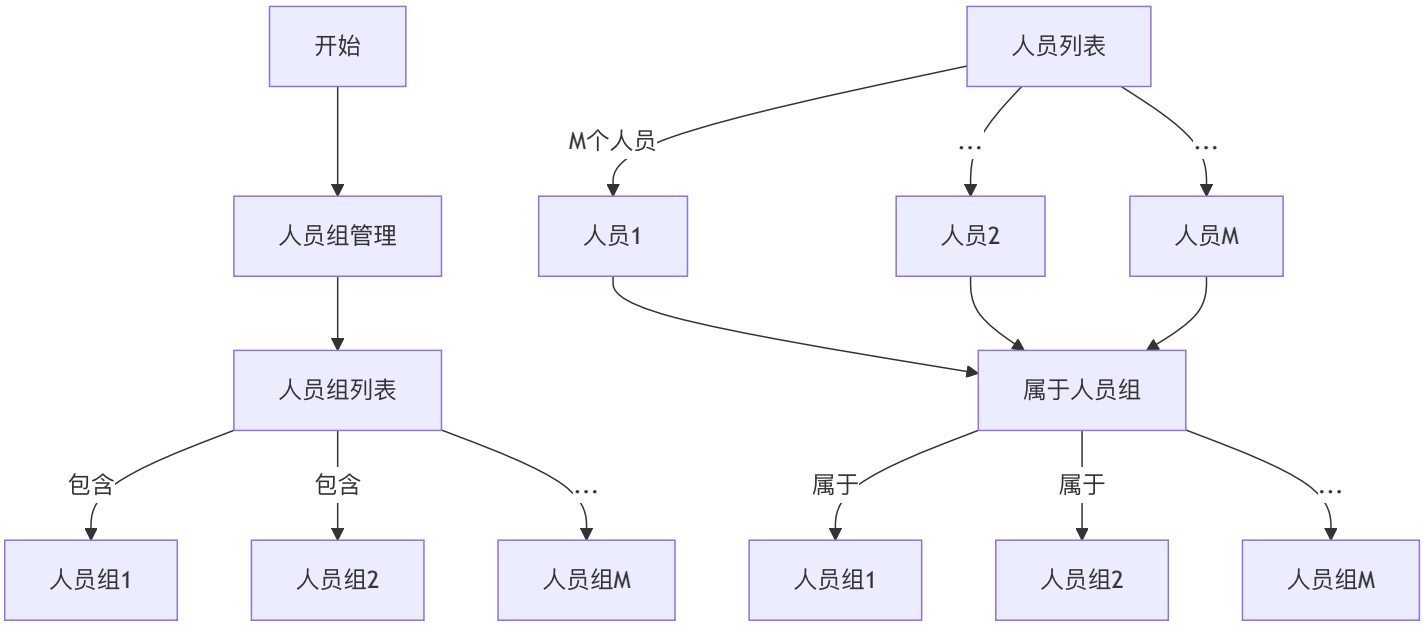
云端需要管理 N 个 M1 设备以及 M 个人员信息,因此我们设计了设备组以及人员组的概念,用于控制每一个 M1 设备的人员数据、人员鉴权策略.



N 台设备可以在 M 个设备组,每台设备可以在任意一个设备组中:

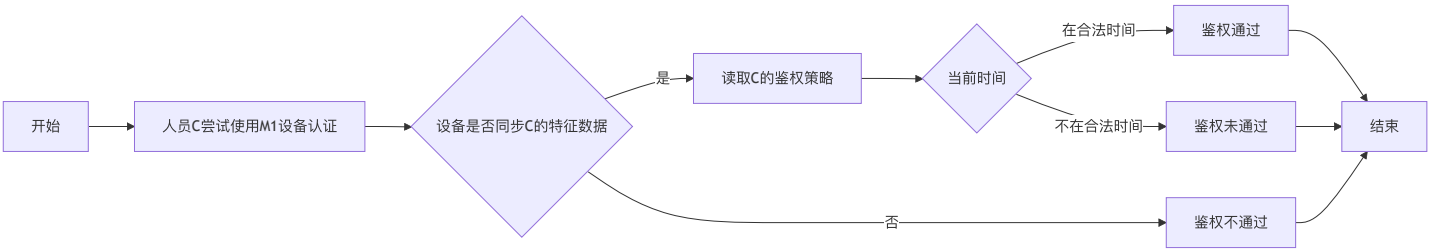


同理,一个人员,也可以在过个人员组内:



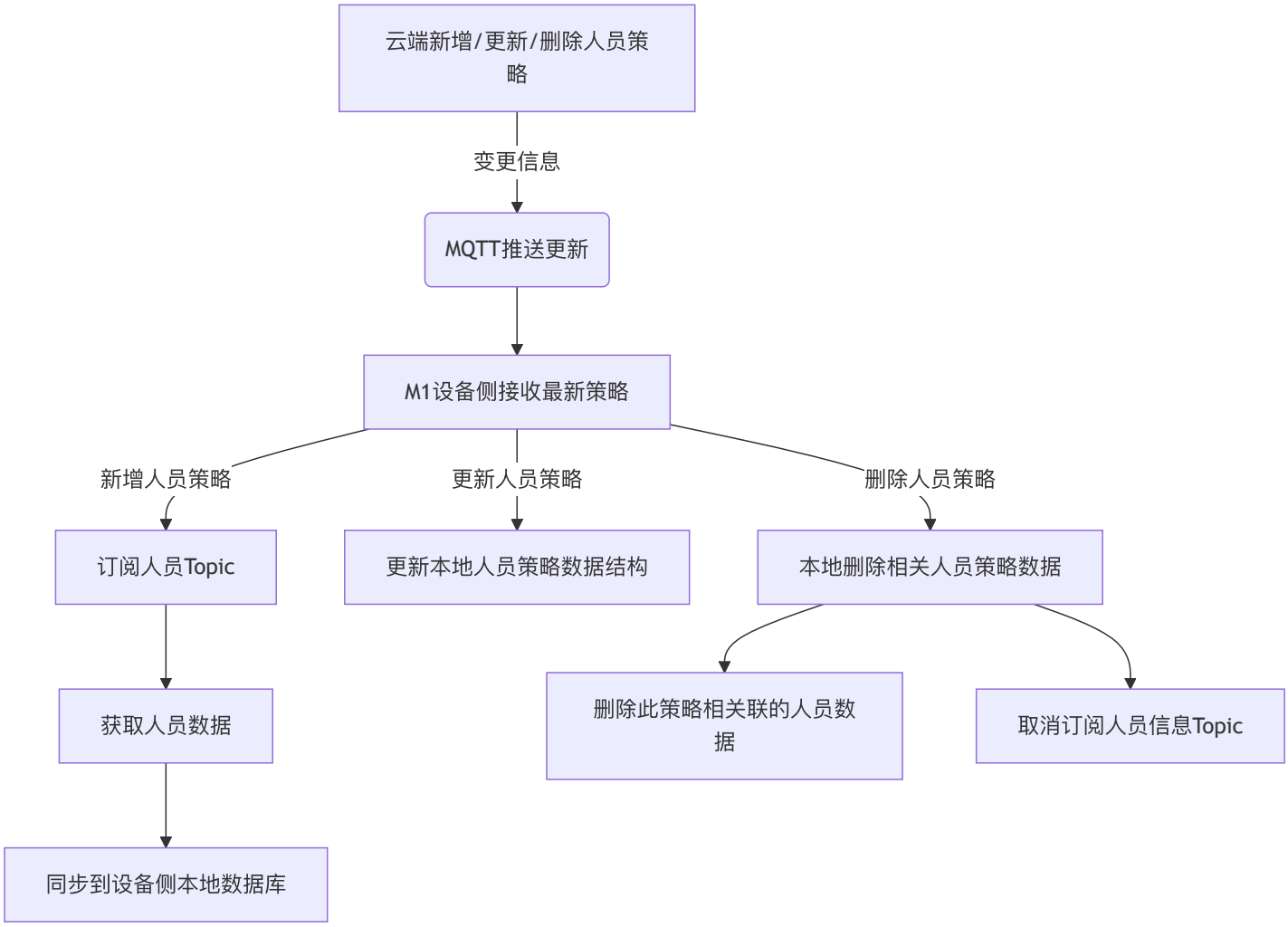
所谓的人员鉴权策略,它定义了一组 M1 设备以及对应的一组人员,在一组时间范围内组内人员是否可以合法地通过 M1 的身份识别校验.通过这三组数据,云端控制每一个 M1 设备需要同步的人员数据以及校验策略,控制每一个 M1 设备的库容,以更好地发挥 M1 的比对资源,提升比对的比中精度.当一个人 C 已经在 M1 云端系统录入了掌纹数据,并尝试使用 M1 设备认证身份,存在以下几种情况:

- 当前 M1 设备已经同步此人 C 的特征数据:
  - 识别出人员 C,读取此人的鉴权策略,假设此人合法通过的时间是每日的 8:00~22:00 则
    - 在合法时间内,鉴权通过;
    - 不在合法时间内,鉴权未通过;
- 当前 M1 设备并没有同步此人 C 的特征数据,则无法识别比中,鉴权不通过;



整体的业务流程：

- 云端新增、更新、删除人员策略,包括变更人员组成员、变更设备组或者是修改了鉴权策略;
- 更新完成后,根据以上变化,需要通过 MQTT 的云端下发权限策略到设备侧 推送到涉及到的 M1 设备;
- M1 设备侧通过订阅的云端下发权限策略到设备侧 接口,接收到最新的人员策略以及涉及到的人员,开始对本地人员策略进行增/删/改的操作,以及本地人员库进行更新;
  - 在本地新增人员策略,同时会订阅相关人员的 topic 云端下发新增更新删除人员信息,获取人员数据,同步到设备侧本地的数据库;
  - 更新人员策略只能更新人员策略数据结构本身,涉及到的人员列表暂时不支持同步更新;
  - 需要删除人员策略,则会在设备侧本地删除相关人员策略数据,同时删除此策略相关的人员数据,被删除的人员同时也会取消从云端订阅人员信息云端下发新增更新删除人员信息;



鉴权策略

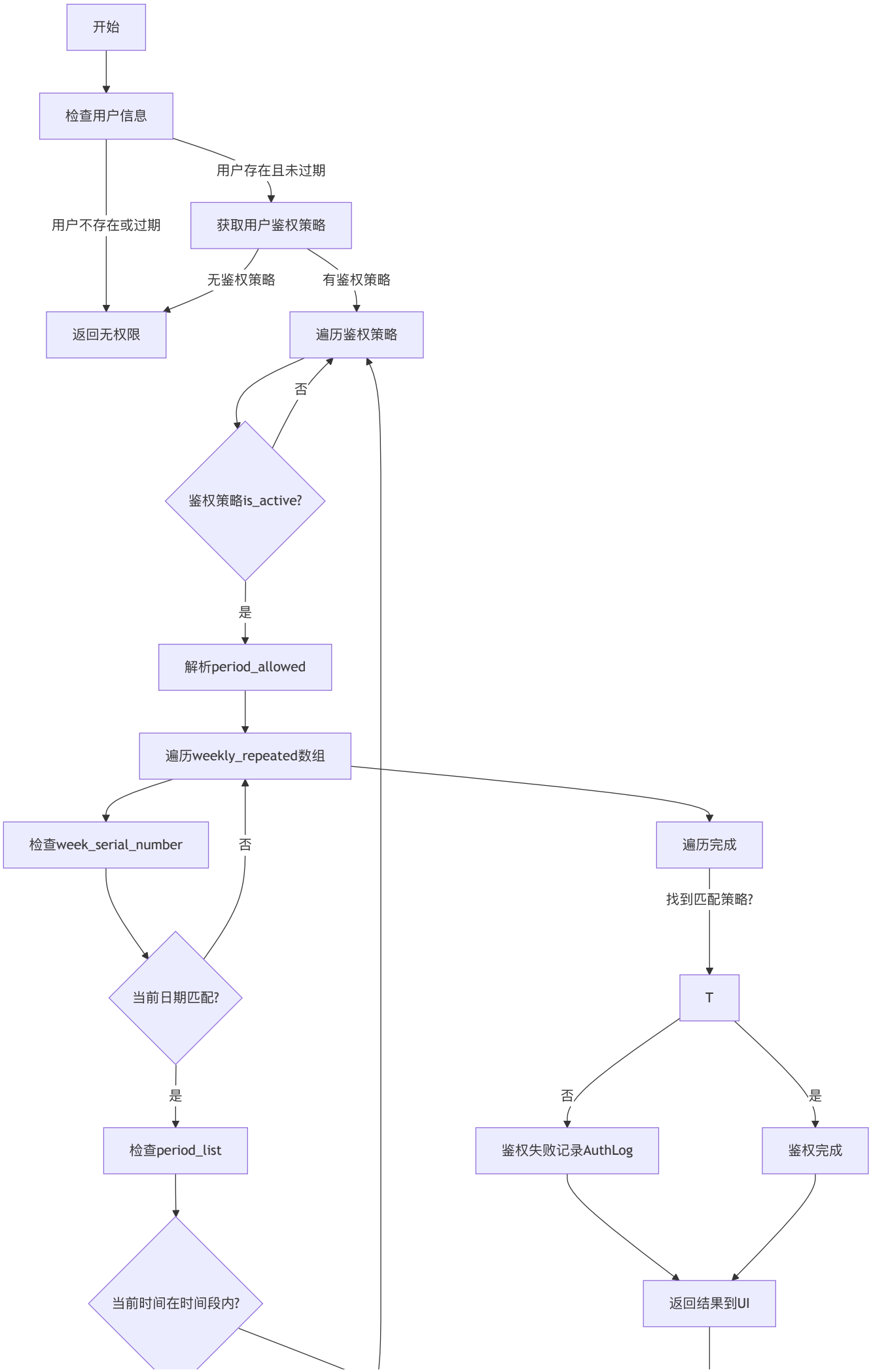
鉴权策略数据结构中,真正提供鉴权策略规则的是 **std::string period_allowed** json字符串字段,其格式如下:

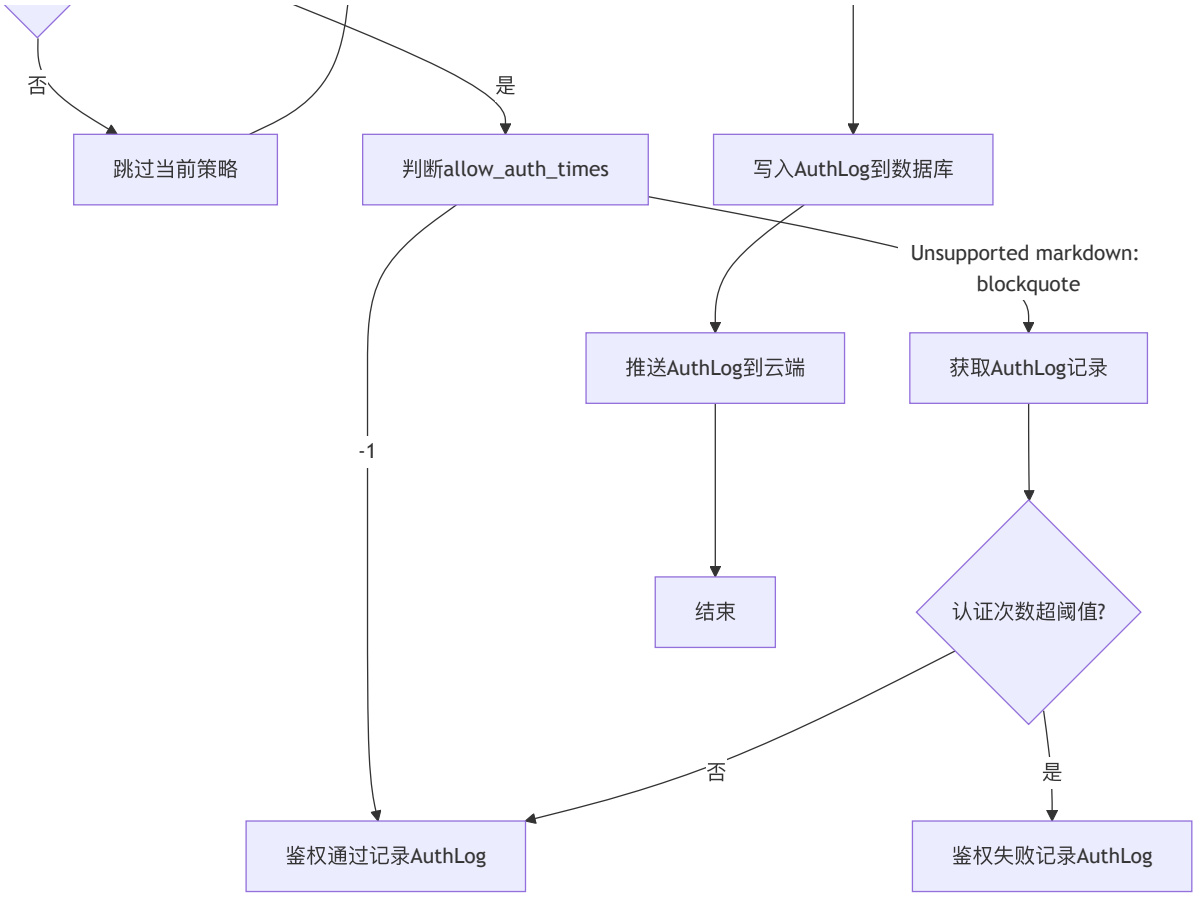
```
{
  "weekly_repeated": [
    {
      "period_list": [
        {
          "end_time": "23:59",
          "start_time": "00:00"
        }
      ],
      "allow_auth_times": -1,
      "week_serial_number": 1
    },
    {
      "period_list": [
        {
          "end_time": "23:59",
          "start_time": "00:00"
        }
      ],
      "allow_auth_times": -1,
      "week_serial_number": 2
    },
    {
      "period_list": [
        {
          "end_time": "23:59",
          "start_time": "00:00"
        }
      ],
      "allow_auth_times": -1,
      "week_serial_number": 3
    },
    {
      "period_list": [
        {
          "end_time": "23:59",
          "start_time": "00:00"
        }
      ],
      "allow_auth_times": -1,
      "week_serial_number": 4
    },
    {
      "period_list": [
        {
          "end_time": "23:59",
          "start_time": "00:00"
        }
      ],
      "allow_auth_times": -1,
      "week_serial_number": 5
    },
    {
      "period_list": [
        {
```

```
          "end_time": "23:59",
          "start_time": "00:00"
        }
      ],
      "allow_auth_times": -1,
      "week_serial_number": 6
    },
    {
      "period_list": [
        {
          "end_time": "23:59",
          "start_time": "00:00"
        }
      ],
      "allow_auth_times": -1,
      "week_serial_number": 7
    }
  ]
}
```

设备侧整体的校验流程如下：

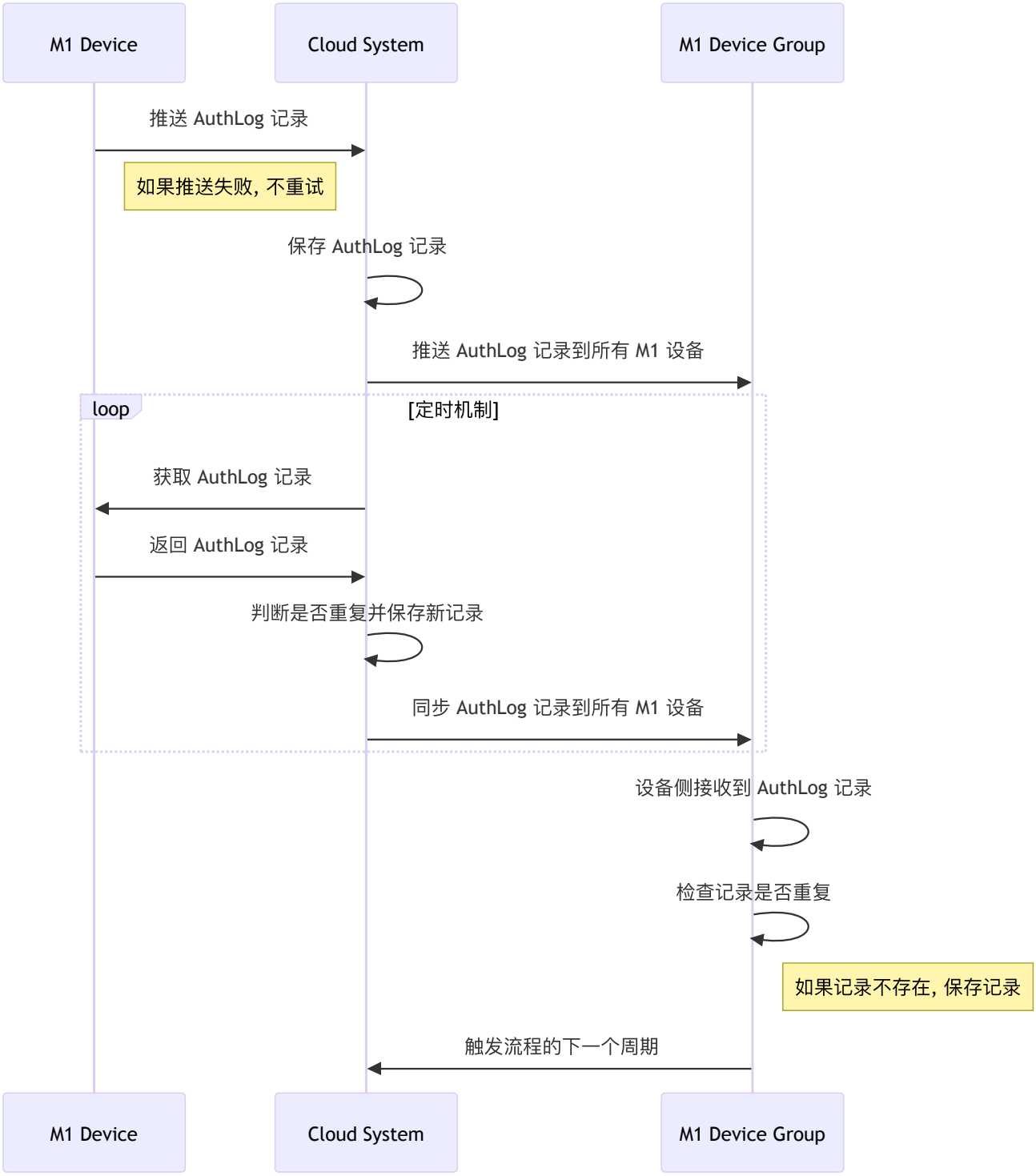
1. 使用 user_uuid 获取该用户的信息,检查该用户是否存在,以及该用户是否过期(通过User.expire_time判断),如果用户不存在或者用户过期则返回无权限,退出下面的流程;
2. 使用 user_uuid 获取这个用户拥有的鉴权策略,如果该用户不存在任何的鉴权策略对象,则返回无权限,退出下面的流程;
3. 遍历所有该用户的鉴权策略表;
4. 如果鉴权策略失效(is_active=0),则跳过;
5. 读取鉴权策略中的 `period_allowed` 字段字符数据,还原成 json 数据结构,从主键 weekly_repeated 获取到一个数组,获取周重复的规则(目前也仅支持周重复规则);
6. 遍历 weekly_repeated 数组中的元素：
   i. week_serial_number: 每个元素代表一周的某一天,使用该值定义,取值范围是正整数 [1,7];
   ii. period_list: 定义一天中某一个合法的时间段,元素内包含 start_time、end_time 代表开始时间、结束时间,格式是 HH:MM;
   iii. allow_auth_times: 允许认证次数, 其中 -1=无限制,1=一次,2:=两次,以此类推;
7. 如果当前时间不在对应星期 period_list 中,则跳过;
8. period_list 包含当前时间,则判断 allow_auth_times 字段值:
   i. 当 allow_auth_times = -1,则鉴权通过,生成当前用户新的 `AuthResult::PASS` 的 AuthLog 记录,直接跳到步骤11;
   ii. 当 allow_auth_times > 0, 则根据当前时间从本地获取该用户的 AuthLog 记录,判断在规则时间段内,认证次数是否超过阈值,未超过则鉴权通过,生成当前用户新的 `AuthResult::PASS` 的 AuthLog 记录,否则鉴权失败,生成当前用户新的 `AuthResult::AUTH_ATTEMPTS_OVER_LIMIT` 的 AuthLog 记录,直接跳到步骤11;
9. period_list 不包含当前时间,跳过,读取下一个鉴权策略,回到步骤3:
10. 遍历完所有鉴权策略,没有找到符合时间段要求的鉴权策略,鉴权失败,生成当前用户新的 `AuthResult::NO_ACCESS` 的 AuthLog 记录,进入步骤11;
11. 鉴权完成返回结果到UI,将 AuthLog 记录写入数据库,同时将 AuthLog 记录推送到云端;


开始
检查用户信息
用户存在且未过期
用户不存在或过期
获取用户鉴权策略
无鉴权策略
有鉴权策略
返回无权限
遍历鉴权策略
否
鉴权策略is_active?
是
解析period_allowed
遍历weekly_repeated数组
检查week_serial_number
否
遍历完成
找到匹配策略?
当前日期匹配?
T
是
否
是
检查period_list
鉴权失败记录AuthLog
鉴权完成
当前时间在时间段内?
返回结果到UI



以上流程仅展示单台M1的鉴权策略校验流程,我们可以知道,限制用户鉴权次数的基础是 AuthLog 记录,当我们需要在多台 M1 设备上限制某些用户的鉴权次数,我们需要将 AuthLog 记录同步到其他关联的 M1 设备,我们统称这些设备为设备组。在设备组之间同步 Authlog 记录,需要由云端系统发起维护:

1. 设备侧完成了上一步完整的鉴权流程,生成 AuthLog 记录,使用设备侧上传认证记录接口将 AuthLog 记录推送到云端,需要提醒的是,如果推送失败,设备侧不会尝试重试;
2. 云端收到设备侧上报的 AuthLog 记录,在云端保存,同时根据业务需要,将 AuthLog 记录推送给设备组内的所有 M1 设备;
3. 由于M1设备侧的硬件资源有限,以及网络情况的不可控性,设备侧无法保证 AuthLog 记录的同步,所以云端需要定时检查设备的 AuthLog,以及将 AuthLog 记录同步到设备组内的所有 M1 设备;
4. 通过云端请求获取设备侧的认证日志接口,云端可以主动获取设备侧的 AuthLog 记录,且通过字段的值判断数据是否重复，如果不重复则作为新的记录保存;
5. 根据具体的业务需要周期性地推送认证记录(通常只需要推送 `AuthResult::PASS` 的认证记录),利用云端下发新增/删除认证记录接口将 AuthLog 记录重新同步到设备组内的所有 M1 设备;
6. 设备侧接收到云端下发的 AuthLog 记录,会根据以下规则判定是否重复
   i. 执行
   ```
   select * from auth_log where user_uuid =? and auth_time =? and auth_method =? and auth_result =?
   ```
   结果的行数,如果结果行数为 0,则表示该 AuthLog 记录本地不存在,保存在本地,否则忽略;
7. 通过步骤2以及定时机制,周而复始的触发步骤4的流程,从而保证 AuthLog 记录在设备组中的一致性;


M1 Device
Cloud System
M1 Device Group
推送 AuthLog 记录
如果推送失败, 不重试
保存 AuthLog 记录
推送 AuthLog 记录到所有 M1 设备
loop
[定时机制]
获取 AuthLog 记录
返回 AuthLog 记录
判断是否重复并保存新记录
同步 AuthLog 记录到所有 M1 设备
设备侧接收到 AuthLog 记录
检查记录是否重复
如果记录不存在, 保存记录
触发流程的下一个周期
M1 Device
Cloud System
M1 Device Group

云端表结构参考

设备组表结构参考

```
CREATE TABLE iot_platform.device_group (
 id serial4 NOT NULL,
 "name" varchar(30) NOT NULL,
 create_time timestamp NOT NULL,
 CONSTRAINT device_group_pkey PRIMARY KEY (id)
);
CREATE INDEX device_group_name_index ON iot_platform.device_group USING btree (name);

CREATE TABLE iot_platform.device_group_mapping (
 device_group_id serial4 NOT NULL,
 device_id serial4 NOT NULL,
 CONSTRAINT device_group_mapping_pkey PRIMARY KEY (device_group_id, device_id)
);
CREATE INDEX device_group_id_device_group_mapping_fkey ON iot_platform.device_group_mapping USING btree
CREATE INDEX device_id_device_group_mapping_fkey ON iot_platform.device_group_mapping USING btree (devic
```

人员组表结构参考

```
CREATE TABLE iot_platform.person_group (
 id serial4 NOT NULL,
 "name" varchar(30) NOT NULL,
 create_time timestamp NOT NULL,
 is_default bool NULL DEFAULT false,
 CONSTRAINT person_group_pkey PRIMARY KEY (id)
);
CREATE INDEX person_group_name_index ON iot_platform.person_group USING btree (name);

CREATE TABLE iot_platform.person_group_mapping (
 person_id serial4 NOT NULL,
 person_group_id serial4 NOT NULL,
 CONSTRAINT person_group_mapping_pkey PRIMARY KEY (person_id, person_group_id)
);
CREATE INDEX person_group_id_person_group_mapping_fkey ON iot_platform.person_group_mapping USING btree
CREATE INDEX person_id_person_group_mapping_fkey ON iot_platform.person_group_mapping USING btree (perso
```

人员鉴权策略表结构参考

```
# period_allowed 是 json 的字符串,定义了人员鉴权策略的时间范围以及相关策略,需要云端与设备侧共同约定解析逻辑

CREATE TABLE iot_platform.group_access_strategy (
 id serial4 NOT NULL,
 person_group_id serial4 NOT NULL,
 device_group_id serial4 NOT NULL,
 period_allowed jsonb NOT NULL,
 is_active bool NOT NULL DEFAULT false,
 CONSTRAINT group_access_strategy_pkey PRIMARY KEY (id),
 CONSTRAINT group_access_strategy_device_group_id_fkey FOREIGN KEY (device_group_id) REFERENCES iot_plat
 CONSTRAINT group_access_strategy_person_group_id_fkey FOREIGN KEY (person_group_id) REFERENCES iot_plat
);
CREATE INDEX device_group_id_group_access_strategy_fkey ON iot_platform.group_access_strategy USING btre
CREATE INDEX person_group_id_group_access_strategy_fkey ON iot_platform.group_access_strategy USING btre
```

MQTT TOPIC

设备端 SUBSCRIBE TOPIC

- 设备侧订阅云端的设备信息更新: device/info/{device_uuid}
- 远程设备控制: device/control/{device_uuid}
- 云端的触发固件升级请求: device/ota/request/{device_uuid}/+
- 云端下发权限策略到设备侧: v2/device/strategy/{deivce_uuid}
- 云端验证输入 NFC 卡号是否已绑定结果: rpc/response/nfc_verify/{device_uuid}/+
- 云端返回设备侧搜索的人员数据: rpc/response/person_info/{device_uuid}/+
- 设备侧同步人员基本信息到云端结果: v2/rpc/response/person/{person_uuid}/{device_uuid}/+
- 云端下发新增/更新/删除人员信息: v2/person/{person_uuid}
- 云端下发新增/删除认证记录: v2/auth_log/{person_uuid}
- 云端同步设备人员列表: v2/rpc/response/person/list/{device_uuid}/+
- 云端返回设备侧它需要订阅的topics列表: v2/rpc/response/device/subscription/{device_uuid}/+
- 云端请求获取设备侧的认证日志: rrpc/request/auth_log/list/{device_uuid}/+
- 云端返回设备密钥的签名: rrpc/response/device/register/{device_uuid}/{request_id}

设备端 PUBLISH TOPIC

- 响应,在不同阶段发送response报文更新升级状态: device/ota/response/{device_uuid}/{request_id}
- 设备侧上传认证记录: device/auth_record/{device_uuid}
- 上传报警事件: device/alert_event/{device_uuid}
- 设备侧推送设备信息到云端: device/info/{device_uuid}
- 通知 broker 我要把自己删了: device/delete_me/{device_uuid}
- 尝试在云端验证输入 NFC 卡号是否已绑定: rpc/request/nfc_verify/{device_uuid}/{request_id}
- 设备侧请求云端搜索获取人员信息: rpc/request/person_info/{device_uuid}/
- 设备侧同步人员基本信息到云端: v2/rpc/request/person/{person_uuid}/{device_uuid}
- 设备侧请求云端同步本设备人员列表: v2/rpc/request/person/list/{device_uuid}
- 设备侧请求从云端获取需要在本地订阅的topics列表: v2/rpc/request/device/subscription/{device_uuid}
- 回复云端请求获取设备侧的认证日志: rrpc/response/auth_log/list/{device_uuid}/{request_id}
- 设备接入,获取秘钥: rrpc/request/device/register/{device_uuid}/+

设备相关

设备到云端注册

分成两种情况,第一次注册以及非第一次注册,参数存在部分区别.

设备第一次接入云端

设备第一次接入云端进行注册,输入云端提供的绑定码,设备尝试请求绑定.

request（设备 → 云端）

```
Method: connect
Payload:
{
    "username": "${DEVICE_ID}&&${LOCAL_TIMESTAMP}&&${RANDOM_INT}",
    "password": "${云端提供的绑定码}"
}
```

设备普通接入云端

设备断电重启后,会尝试使用本地存储的秘钥(通过设备获取密钥接口获取)重新与云端建立连接.

password 生成的逻辑参考,参数包括 DEVICE_SECRET、DEVICE_ID、TIMESTAMP 和 NONCE:

- DEVICE_SECRET -> 是云端返回的秘钥
- DEVICE_ID -> 设备的ID
- TIMESTAMP -> 本地生成的时间戳
- NONCE -> 本地生成的整数随机数

```
// decodedKey = DEVICE_SECRET
// message = $DEVICE_SECRET + $DEVICE_ID + $TIMESTAMP + $NONCE
std::string crypto_calculate_HMACSHA256(const std::string &decodedKey, const std::string &message) {
  std::string calculated_hmac;
  CryptoPP::HMAC<CryptoPP::SHA256> hmac(reinterpret_cast<const CryptoPP::byte *>(decodedKey.data()), dec
  CryptoPP::StringSource give_me_a_name(
      message, true,
      new CryptoPP::HashFilter(hmac, new CryptoPP::Base64Encoder(new CryptoPP::StringSink(calculated_hma
  return calculated_hmac;
}
```

request（设备 → 云端）

```
Method: connect
Payload:
{
    "username": "${DEVICE_ID}&&${LOCAL_TIMESTAMP}&&${RANDOM_INT}&&HMACSHA256",
    "password": ${加密后的password}
}
```

设备获取密钥

设备第一次接入云端进行注册后,云端通过 RRPC 向设备发送云端为设备生成的设备密钥

request（设备 → 云端）设备通知云端已经完成接入流程,可以下发设备密钥

```
Topic: rrpc/request/device/register/{device_uuid}/+
Method: Publish
```

response（云端 → 设备）云端通过 RRPC 向设备下发设备密钥

```
Topic: rrpc/request/device/register/{device_uuid}/{request_id}
Method: Subscribe
Payload:
{
    "uuid": UUID,
    "device_secret": String
}
```

云端 → 设备:收到设备密钥后的响应

```
Topic: rrpc/response/device/register/{device_uuid}/{request_id}
Method: Publish
Payload:
{
    "uuid": UUID,
    "signature": String // 签名字段内容为 request_id&&device_secret
}
```

设备上传基本信息

设备第一次接入云端进行注册后,设备需要向云端上传自己的基本信息

message（设备 → 云端）

```
Topic: device/info/{device_uuid}
Method: Publish
Payload:
{
    "uuid": UUID,
    "name": String,
    "location": String,
    "remark": String // Optional
}
```

云端向设备下发设备基本信息更新

设备成功接入云端后,用户在云端对设备基本信息更新后,云端会将更新后的设备基本信息下发到设备

设备通过 Subscribe 相应 topic 监听云端的设备信息更新

```
Topic: device/info/{device_uuid}
Method: Subscribe
```

message 云端 → 设备 下发设备基本信息

```
Topic: device/info/{device_uuid}
Method: Publish
Payload:
{
    "uuid": UUID,
    "name": String,
    "location": String,
    "remark": String // Optional
}
```

设备侧从云端获取需要在本地订阅的topics列表

设备启动后,会首先尝试与云端建立通讯,然后请求云端返回需要设备侧订阅的 topic 列表.

message 设备 → 云端,设备侧请求从云端获取需要在本地订阅的topics列表:

```
Topic: v2/rpc/request/device/subscription/{device_uuid}
Method: Publish
```

设备侧需要订阅返回消息的 topic

```
Topic: v2/rpc/response/device/subscription/{device_uuid}/+
Method: Subscribe
```

message 云端 → 设备,云端返回设备侧它需要订阅的topics列表

```
Topic: v2/rpc/response/device/subscription/{device_uuid}/{request_id}
Method: Publish
Payload:["topic1", "topic2", "topic3"]
```

设备控制

云端需要删除设备,会将相关信息下发到设备.设备通过 Subscribe 相应 topic 监听云端的设备消息下发.目前仅支持删除设备,并要求设备重置.

```
Topic: device/control/{device_uuid}
Method: Subscribe
```

message 云端 → 设备 下发设备基本信息

```
Topic: device/control/{device_uuid}
Method: Publish
Payload:
{
    "operation": REMOVE_DEVICE( 1 )
}
```

设备恢复出厂设置

设备恢复出厂设置后需要向云端同步删除请求,云端会将设备删除

message（设备 → 云端）

```
Topic: device/delete_me/{device_uuid}
Method: Publish
Payload:
{
    "uuid": UUID,
    "name": String,
    "location": String,
    "remark": String // Optional
}
```

人员相关

设备侧同步人员基本信息到云端

人员在设备端录入后会上传到云端,目前支持新增/更新两个方法.

message（设备 → 云端）

```
Topic: v2/rpc/request/person/{person_uuid}/{device_uuid}
Method: Publish
Payload:
{
    "method": INSERT( 1 )/UPDATE( 2 )
    "person_info":
      {
        "id": int
        "uuid": UUID,
        "custom_id": String
        "name": String,
        "auth_method": SINGLE_PALM( 1 )/SINGLE_NFC( 2 )/FACTORS_PALM_NFC( 3 )/FACTORS_PALM_P
        "device_create_time": timestamp
        "role_type": DEVICE_ADMIN( 1 )/DEVICE_USER( 2 ),
        "nfc_card_number": String,
        "password": String,
        "salt": String,
        "expire_time": timestamp,
      }
    templates_left: std::vector<std::vector<uint8_t>> ,
    templates_right: std::vector<std::vector<uint8_t>> ,
    "update_right": boolean,
    "update_left": boolean
}
```

message（云端 → 设备）

```
Topic: v2/rpc/response/person/{person_uuid}/{device_uuid}/+
Method: Subcribe
Payload:
{
    "code": 0,
    "message": String,
    "data": {}
}
```

搜索获取人员信息

设备侧通过 rpc 的形式向云端同步请求人员信息,payload 内根据 query_by 字段声明使用什么进行查询

设备通过 Publish 到 rpc request 的 topic 进行请求,在 payload 内根据 query_by 字段声明使用什么进行查询

```
Topic: rpc/request/person_info/{device_uuid}/
Payload:
{
    "query_by": UUID( 1 )/CUSTOM_ID( 2 ),
    "uuid": UUID,
    "custom_id": String
}
```

设备 Subscribe rpc 的 response topic

```
Topic: rpc/response/person_info/{device_uuid}/+
Method: Subscribe
```

云端下发到设备侧的数据:

```
Topic: rpc/response/person_info/{device_uuid}/{request_id}
Payload: // person_info
{
    "uuid": UUID,
    "name": String,
    "device_create_time": timestamp,
    "auth_method": SINGLE_PALM( 1 )/SINGLE_NFC( 2 )/FACTORS_PALM_NFC( 3 )/FACTORS_PALM_PASSWO
    "role_type": DEVICE_ADMIN( 1 )/DEVICE_USER( 2 ),
    "nfc_card_number": String,
    "password": String,
    "salt": String,
    "custom_id": String,
    "expire_time": timestamp,
    "person_group_id_list": List<Integer> // 人员组 id List
    "has_template": boolean // 云端数据库是否包含人员掌信息
}
```

云端下发新增/更新/删除人员信息

人员信息在云端进行更改后需要下发到存有该人员的设备进行数据同步.

设备通过 Subscribe 相应 topic 监听云端的人员消息下发,支持新增/更新/删除.

```
Topic: v2/person/{person_uuid}
Method: Subscribe
```

message 云端 → 设备 下发人员信息

```
Topic: v2/person/{person_uuid}
Method: Publish
Payload:
{
    "method": INSERT( 1 )/UPDATE( 2 )/DELETE( 3 )
    "person_info": // person_info 需要将 json 数据转成 json string 格式赋值
      {
        "uuid": UUID,
        "custom_id": String
        "name": String,
        "auth_method": SINGLE_PALM( 1 )/SINGLE_NFC( 2 )/FACTORS_PALM_NFC( 3 )/FACTORS_PALM_P
        "device_create_time": timestamp
        "role_type": DEVICE_ADMIN( 1 )/DEVICE_USER( 2 ),
        "nfc_card_number": String,
        "password": String,
        "salt": String,
        "expire_time": timestamp,
      }
    templates_left: std::vector<std::vector<uint8_t>> ,
    templates_right: std::vector<std::vector<uint8_t>> ,
    "update_right": boolean,
    "update_left": boolean
}
```

> person_info 需要将 json 数据转成 json string 格式赋值

设备侧请求云端同步本设备人员列表

设备侧请求云端同步本设备人员列表,获取到人员列表后,逐一尝试搜索本地的数据库,发现本地缺失的人员,则尝试订阅相关人员的信息并同步回本地.

设备 Subscribe 相应 topic 监听云端的人员列表数据下发.

message 云端 → 设备 下发人员列表

```
Topic: v2/rpc/response/person/list/{device_uuid}/+
Method: Subscribe
Payload:[
  "uuid1",
  "uuid2",
  "uuid3"
]
```

设备 Publish 相应 topic 请求云端的人员列表数据下发.

```
Topic: v2/rpc/request/person/list/{device_uuid}
Method: Publish
```

设备侧从云端获取人员组数据

设备上的人员录入需要云端人员组信息,设备通过 rpc 请求向云端请求全部人员组数据

设备需要 Subscribe 相应 Topic 监听云端的人员组下发

```
Topic: rpc/response/person_group/{device_uuid}/+
Method: Subscribe
```

response（云端 → 设备）

```
Topic: rpc/response/person_group/{device_uuid}/{request_uuid}
Method: Publish
Payload:
[{
    "id": int, // Group ID
    "name": String // Group Name
}]
```

request（设备 → 云端）请求云端返回人员组数据

```
Topic: rpc/request/person_group/{device_uuid}/+
Method: Publish
```

云端下发权限策略到设备侧

设备接入云端后会在云端上被分配权限策略,云端会将设备的权限策略下发给设备

设备通过 Subscribe 相应 topic 监听云端的权限策略下发

```
Topic: v2/device/strategy/{deivce_uuid}
Method: Subscribe
```

message 云端 → 设备 下发设备权限策略

```
# 新增权限策略
Topic: v2/device/strategy/{deivce_uuid}
Method: Publish
Payload:
{
    "method": CREATE( 1 ),
    "strategies": [{
        "person_uuid_list": ["ID1","ID2"],
        "entity_strategy_id": int,
        "group_strategy_id": int,
        "period_allowed": {
            "weekly_repeated": [{
                "week_serial_number": int,
                "period_list": [{
                    "start_time": String,
                    "end_time": String
                }]
            }]
        },
        "is_active": boolean
    }]
}

# 更新权限策略
Topic: v2/device/strategy/{deivce_uuid}
Method: Publish
Payload:
{
    "method": UPDATE( 2 ),
    "strategies": [{
        "group_strategy_id": int,
        "period_allowed": {
            "weekly_repeated": [{
                "week_serial_number": int,
                "period_list": [{
                    "start_time": String,
                    "end_time": String
                }]
            }]
        },
        "is_active": boolean
    }]
}

########################################
# !!!!!!!!!!!! ATTENTION !!!!!!!!!!!!
# 删除权限策略
# 实际该方法是在设备侧直接将所有用户删除了
########################################
Topic: v2/device/strategy/{deivce_uuid}
Method: Publish
Payload:
{
    "method": DELETE( 3 ),
    "strategies": [{
```

```
        "person_uuid_list": ["ID1","ID2"],
        "entity_strategy_id": int,
        "group_strategy_id": int,
        "is_active": boolean
    }]
}
```

验证输入NFC卡号是否已绑定

设备端上传人员信息前需要判断当前人员 NFC 卡号是否已绑定

设备需要 Subscribe 相应 Topic 监听云端相应 response

```
Topic: rpc/response/nfc_verify/{device_uuid}/+
Method: Subscribe
```

response（云端 → 设备）

```
Topic: rpc/response/nfc_verify/{device_uuid}/{request_uuid}
Method: Publish
Payload:
{
    "code": SUCCESS( 1 )/NON_UNIQUE_NFC_NUMBER( 4 )/INVALID_ARGUMENT_FORMAT( 8 ),
    "message": "",
    "data": {
        "person_uuid": "",
        "person_name": ""
    }
}
```

日志

设备侧上传认证记录

人员在设备上刷手认证后,设备会将生成的认证记录上传到云端.

message（设备 → 云端）

```
Topic: device/auth_record/{device_uuid}
Method: Publish
Payload:
{
    "device_created_id": int,
    "auth_method": SINGLE_PALM( 1 )/SINGLE_NFC( 2 )/FACTORS_PALM_NFC( 3 )/FACTORS_PALM_PASSW
    "auth_result": AuthResult,
    "person_name": String,
    "auth_time": timestamp,
    "person_name": String,
    "person_uuid": String
}
```

云端下发新增/删除认证记录

云端可以将其他设备上传的认证记录,下发到指定的设备让其保存,或者是命令设备删除某些用户某时间段内的认证记录,用于鉴权策略.

设备通过 Subscribe 相应 topic 监听云端的人员消息下发,支持新增/更新/删除.

```
Topic: v2/auth_log/{person_uuid}
Method: Subscribe
```

message 云端 → 设备 下发数据

- 新增认证记录,只允许同步 auth_result 为 PASS 的数据:

```
Topic: v2/auth_log/{person_uuid}
Method: Publish
Payload:
{
    "method": INSERT( 1 )
    "auth_log":
      {
        "user_uuid": UUID,
        "user_id": String,
        "custom_id": String,
        "username": String,
        "auth_time": timestamp,
        "auth_method": SINGLE_PALM( 1 )/SINGLE_NFC( 2 )/FACTORS_PALM_NFC( 3 )/FACTORS_PALM_P,
        "auth_result": PASS = 1
      }
}
```

- 删除某一个人的认证记录,指定 user_uuid 以及 auth_time 的开始、结束时间的时间戳,闭区间 [auth_time_begin, auth_time_end]:

```
Topic: v2/auth_log/{person_uuid}
Method: Publish
Payload:
{
    "method": DELETE( 3 )
    "auth_log":
      {
        "user_uuid": UUID,
        "auth_time_begin": timestamp,
        "auth_time_end": timestamp
      }
}
```

云端请求获取设备侧的认证日志

> 设备侧每次生成的 Authlog 认证日志,都会同步推送到云端,但是当网络、服务异常导致推送失败,设备端并不会尝试重新推送,原因是设备则本身的性能有限, 这里的认证一致性需要云端自己来保证,因此提供此接口.

设备通过 Subscribe 相应 topic 监听云端的触发获取认证日志请求

```
Topic: rrpc/request/auth_log/list/{device_uuid}/+
Method: Subscribe
```

message 云端 → 设备 发送云端触发录入的请求

```
Topic: rrpc/request/auth_log/list/{device_uuid}/{request_id}
Method Publish
Payload:
{
    "user_uuid": UUID,   // [选填]用户UUID,如未填入则查询user_uuid不为null的认证日志
    "custom_id": String, // [选填]用户自定义ID
    "auth_time_begin": timestamp, // [选填]认证开始时间
    "auth_time_end": timestamp,   // [选填]认证结束时间
    "auth_method": int, // [选填]认证方式 SINGLE_PALM( 1 )/SINGLE_NFC( 2 )/FACTORS_PALM_NFC( 3 )/FACTORS_F
    "auth_result": int, // [选填]认证结果 PASS = 1, REJECT = 2, NO_ACCESS = 3, NON_LIVING = 4, AUTH_ATTEMF
}
```

message 设备 → 云端 回复云端触发录入的请求

```
Topic: rrpc/response/auth_log/list/{device_uuid}/{request_id}
Method Publish
Payload:
{
  "data": [
  {
   "auth_method" : 2,
   "auth_result" : 1,
   "auth_time" : 1723812887,
   "custom_id" : "test",
   "id" : 19,
   "user_id" : 11,
   "user_uuid" : "96813953-f91c-4103-9d3f-1e8c9c336403",
   "username" : "NFC卡"
  },
  {
   "auth_method" : 2,
   "auth_result" : 1,
   "auth_time" : 1723812897,
   "custom_id" : "test",
   "id" : 20,
   "user_id" : 11,
   "user_uuid" : "96813953-f91c-4103-9d3f-1e8c9c336403",
   "username" : "NFC卡"
  }
 ],
  "rows": 2,
  "code": 0, // 0表示成功,其他为异常情况
  "message": "success"
}
```

告警事件

设备生成告警事件后,会上传到云端

message（设备 → 云端）

```
Topic: device/alert_event/{device_uuid}
Method: Publish
Payload:
{
    "device_created_id": int,
    "alert_type": CONTINUOUS_AUTHENTICATION_FAILURE( 1 )/NON_LIVING_AUTH( 2 )/DURESS_ALARM
    "alert_detail": {
        "continuous_failed_time": String,
        "auth_person": String,
        "auth_method": SINGLE_PALM( 1 )/SINGLE_NFC( 2 )/FACTORS_PALM_NFC( 3 )/FACTORS_PALM_P
    }
    "alert_time": timestamp
}
```

云端触发设备进入掌纹掌静脉录入

云端需要响应 web 前端的触发设备录入请求,并需要同步的回复前端当前设备是否可以进行录入,整个链路都是同步的,又由于
是云端需求设备进行回应,故使用 rrpc 来实现

设备通过 Subscribe 相应 topic 监听云端的触发录入请求

```
Topic: rrpc/request/person/enroll/{device_uuid}/+
Method: Subscribe
```

message 云端 → 设备 发送云端触发录入的请求

```
Topic: rrpc/request/person/enroll/{device_uuid}/{request_id}
Method Publish
Payload:
{
    "id": int,
    "uuid": UUID,
    "custom_id": String
    "name": String,
    "auth_method": SINGLE_PALM( 1 )/SINGLE_NFC( 2 )/FACTORS_PALM_NFC( 3 )/FACTORS_PALM_P
    "device_create_time": timestamp
    "role_type": DEVICE_ADMIN( 1 )/DEVICE_USER( 2 ),
    "device_update_time": timestamp
    "nfc_card_number": String,
    "password": String,
    "salt": String,
    "expire_time": timestamp,
    "hasLeftTemplate": boolean,
    "hasRightTemplate": boolean
}
```

message 设备 → 云端 回复云端触发录入的请求

```
Topic: rrpc/response/person/enroll/{device_uuid}/{request_id}
Method Publish
Payload:
{
code: SUCCESS( 1 )/NOT_HOME_PAGE( 2 )
message: String
}
```

固件升级

推送升级固件请求

云端创建固件发布任务后,会在设定的升级时间向设备发送固件升级请求

设备通过 Subscribe 相应 topic 监听云端的固件升级请求

```
Topic: device/ota/request/{device_uuid}/+
Method: Subscribe
```

message 云端 → 设备 下发设备固件升级请求和路径,设备侧需要自己根据配置的 server ip 和端口拼成 url

```
Topic: device/ota/request/{device_uuid}/{request_id}
Method: Publish
Payload:
{
    "image_url": "/ota/firmware/download/xxx.zip",
    "version": "X.X.X"
}
```

固件升级状态更新

设备收到固件升级请求后,会做出响应,在不同阶段发送 response 报文更新升级状态

message 设备 → 云端 下发设备固件升级请求和路径,设备侧需要自己根据配置的 server ip 和端口拼成 url

```
Topic: device/ota/response/{device_uuid}/{request_id}
Method: Publish
Payload:
{
    "status": int,
    "message": "" // String
}
```

设备侧下载OTA升级包

设备端收到推送升级固件请求后，设备会把 SERVER_IP 和推送升级固件请求的 image_url 字段组合生成 url 请求地址。

随后设备会向此 url 请求地址发送 http 请求，来获取下载 OTA 升级包。

设备发送的 http 请求格式如下

```
Method: GET
Url: ${SERVER_IP:SERVER_PORT}${image_url}
```

- SERVER_IP: IP 地址，设备接入 MQTT 服务的 IP 地址。
- **SERVER_PORT: 端口号，基于约定，它的值是 MQTT 服务端口号 + 1, 如 MQTT 的服务端口号是2803，那么 SERVER_PORT 就是 2803 + 1 = 2804** ;

开发注意事项

关于timestamp时间戳的最大值限制

在设备端我们使用 Sqlite3 作为数据库存储数据,当前 M1 固件中使用的版本是 3.21.0

```
[root@RV1126_RV1109:~]# sqlite3 --version
3.21.0 2017-10-24 18:55:49 1a584e499906b5c87ec7d43d4abce641fdf017c42125b083109bc77c4de48827
```

为保持最好的兼容性,兼顾M1本身硬件平台的性能平衡,我们使用 Sqlite3 的 INTEGER 作为时间戳的存储类型,因此时间戳的最大值为 2147483647,即 2038-01-19 03:14:07 左右。

> Attention: timestamp <= 2147483647

设备侧的表结构

User

```
struct User {
  int id;                          // 自增 id,仅本地使用
  std::string uuid;                // 16 位的 uuid, 用于云平台进行数据同步,index
  std::string username;            // not null, index
  int gender;                      // 0 男,1 女,-1 default
  int role_type;                   // 用户类型,字段值: 1 普通用户,2 管理员,默认 1
  std::string template_ids;        // 特征 template id 数组,存储时以用 , 风格的字符串分隔
  std::string template_ids_left;   // 特征 template id 左手数组,存储时以用 , 风格的字符串分隔
  std::string auth_nfc_number;     // nfc 卡号
  std::string auth_password;       // 密码,加盐 hash
  std::string auth_password_salt;  // 密码加盐 hash 的盐值
  std::string custom_id;           // 第三方用户 id
  int auth_method;  // 认证方式,-1 未知,1 单一手掌,2 单一卡,3 双重: 掌+卡,4 双重 掌+密码
  int create_time;  // 创建时间,精确到秒级的 timestamp
  int update_time;  // 修改时间,精确到秒级的 timestamp
  int expire_time;  // 过期时间,精确到秒级的 timestamp
  Json::Value user_details;  // 用户消息信息,由第三方定义
};
```

AccessStrategy

```
struct AccessStrategy {
  int id;                      // 自增 id,仅本地使用
  int entity_strategy_id;      // 单体权限策略 id
  int group_strategy_id;       // 组权限策略 id
  std::string user_uuid;       // 人员 uuid
  std::string period_allowed;  // 许可时间,json 字符串, 向上返回的时候将在 db 层转化为 Json::Value
  int is_active;               // 该策略是否可用,0 不可用,1 可用,默认 0
  int create_time;             // 创建时间,秒级时间戳
  int update_time;             // 修改时间,秒级时间戳
};
```

AlertEventLog

```
struct AlertEventLog {
  int id;              // 自增 id
  int alert_type;      // 告警事件类型:  1 连续认证失败
  int alert_time;      // 告警时间,也是记录创建时间,精确到秒级的 timestamp
  std::string detail;  // 告警详情,Json 字符串
};
```

AuthLog

```
struct AuthLog {
  int id;                 // 自增 id
  int user_id;            // 用户 id 用于显示编号,默认-1,表示没有用户信息
  std::string custom_id;  // 第三方用户 id
  std::string user_uuid;  // 冗余字段,uuid 用于与平台同步
  std::string username;   // 冗余字段,实际查询时该表不会与 user 表 join
  int auth_time;          // 认证时间,也是记录创建时间,精确到秒级的 timestamp
  int auth_method;        // 认证方式,1 手掌,2 刷卡,3 手掌+卡 4 手掌+密码
  int auth_result;        // 认证结果,1 成功,2 失败,3 无访问权限,4 非活体攻击
};
```

设备侧枚举类型

```
/**
 * user_info 表的 gender 字段
 */
enum UserGender { MALE = 0, FEMALE = 1, UNKNOWN = -1 };

/**
 * user_info 表的 role_type 字段
 */
enum UserRoleType { NORMAL_PERSON = 1, MANAGER = 2 };

/**
 * user_info 表的 auth_method 字段
 */
enum UserInfoAuthMethod {
  SINGLE_PALM_OR_SINGLE_NFC = 1,
  FACTORS_PALM_NFC = 3,
  FACTORS_PALM_PASSWORD = 4
};

/**
 * auth_log 表的 auth_method 字段
 */
enum AuthLogAuthMethod {
  AUTH_SINGLE_PALM = 1,
  AUTH_SINGLE_NFC = 2,
  AUTH_FACTORS_PALM_NFC = 3,
  AUTH_FACTORS_PALM_PASSWORD = 4
};

/**
 * auth_log 表的 auth_result 字段
 */
enum AuthResult {
  PASS = 1,
  REJECT = 2,
  NO_ACCESS = 3,
  NON_LIVING = 4,
  AUTH_ATTEMPTS_OVER_LIMIT = 5
};

/**
 * alert_event_log 表的 alert_type 字段
 */
enum AlertType {
  CONTINUOUS_AUTHENTICATION_FAILURE = 1,  //连续认证
  NON_LIVING_AUTH = 2,                    //非活体攻击
  DURESS_AUTHENTICATION = 3,              //胁迫认证
  FIRE_ALARM = 6,                         //消防告警
  FIRE_ALARM_RELEASE = 7,                 //消防告警解除
  DOOR_SENSOR_TIMEOUT_ALARM = 8,          //门磁状态告警
  DOOR_SENSOR_TIMEOUT_ALARM_RELEASE = 9,  //门磁状态告警解除
  ILLEGAL_DISASSEMBLY = 10,               //防拆告警
```

```
  ILLEGAL_DISASSEMBLY_RELEASE = 11        //防拆告警解除
};
```

UpdateLog

- 20240826
  - 更新了设备侧下载OTA升级包接口描述，说明固件下载地址的约定;
- 20240820
  - 增加关于timestamp时间戳的最大值限制章节;
  - 鉴权策略章节增加同步AuthLog认证记录的描述;
  - 增加新的topic云端请求获取设备侧的认证日志: rrpc/request/auth_log/list/{device_uuid}/+;
  - 增加新的topic回复云端请求获取设备侧的认证日志: rrpc/response/auth_log/list/{device_uuid}/{request_id};
- 20240808
  - 增加新的 topic= `v2/auth_log/{person_uuid}` ,并添加接口描述云端下发新增/删除认证记录
  - AuthLog 中的 AuthResult 枚举类型新增 AUTH_ATTEMPTS_OVER_LIMIT 枚举值,并添加相关逻辑;
  - 业务流程增加鉴权策略章节;
- 20240805
  - 业务流程章节增加整体的业务流程描述;
- 20240804
  - 添加业务流程章节
  - 添加云端触发设备进入掌纹掌静脉录入章节;
  - 添加设备侧从云端获取人员组数据章节;