Table of Contents

Part1 Notice

Part2 ablecloud-matrix-app-sdk

Part3 ablecloud-matrix-local-sdk

Part4 ablecloud-matrix-ext-sdk

更新记录

- v2.0.0/2018.6.6

  重构1.X版本，不兼容低版本；

- v2.0.1/2018.6.23

  1. 修复发现设备中物理Id格式错误；
  2. AccoutManager新增功能；

AbleCloud Matrix SDK

AbleCloud Matrix SDK是AbleCloud推出的Android平台上用于快速进行物联网APP开发的软件开发工具包。

ablecloud-matrix-app-sdk：用于与AbleCloud云端交互；

ablecloud-matrix-local-sdk：用于通过局域网，与适配了AbleCloud固件模块的IoT设备交互。

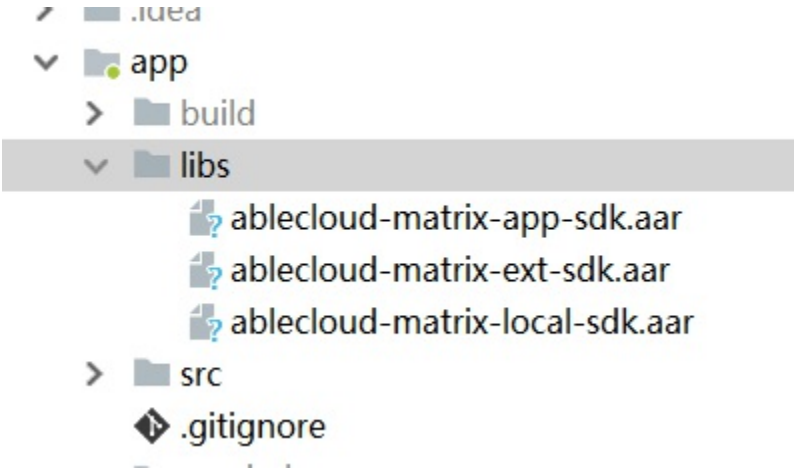
ablecloud-matrix-ext-sdk：用于扩展功能，例如：意见反馈、获取天气信息

1. 配置

1.1 如何配置

AbleCloud提供的SDK均为 `.aar` 格式文件；

1. 将下载完成后的SDK放置项目的libs文件夹下；



2. 在项目的build.gradle文件下，将libs目录作为本地仓库，并将SDK作为依赖，另外添加 `Gson（2.0+)` 、 `OkHttp（3.0+）` ，如果使用头像、文件上传还需要添加七牛的依赖，配置如下:

```
android{
    ...
    repositories {
        flatDir {
            dirs 'libs'
        }
    ...
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
```

```
    compile 'com.qiniu:qiniu-android-sdk:x.x.x'
    compile 'com.google.code.gson:gson:2.x.x'
    compile 'com.squareup.okhttp3:okhttp:3.x.x'
    compile(name: 'ablecloud-matrix-local-sdk', ext: 'aar')
    compile(name: 'ablecloud-matrix-app-sdk', ext: 'aar')
    compile(name: 'ablecloud-matrix-ext-sdk', ext: 'aar')
    ...
}
```

3. 构建项目，完成之后就可以开始使用了。

2. 初始化

2.1 使用前须知

SDK中的耗时操作的回调不在主线程，推荐开发者在开发过程中，如果要在回调成功结束之后进行UI操作，推荐使用RxJava、AsyncTask等完成。

2.2 公有云非国际化配置

```
Matrix.init(context,domainName,domainId,mode,region);
```

- **context**：上下文
- **dominName**：主域
- **dominId**：子域Id
- **mode**：开发环境
  - 测试环境为：0
  - 正式环境为：1
- **region**：所在区域
  - 国内：0
  - 欧洲：4
  - 北美：3
  - 东南亚：1

2.3 公有云国际化配置

使用国际化，意味App连接服务器地址动态获取，所以第一步需要获取对应URL

2.3.1 获取URL

```
Matrix.configGlobal(dominName, mode, country, new MatrixCallback<String>() {
    @Override
    public void success(String s) {
     //获取到该主域下国家和环境对应的URL
```

```
    }

    @Override
    public void error(MatrixError matrixError) {
        e.onError(matrixError);
        }
    });

// mode:测试环境为：0,正式环境为：1
// country:国家代码
```

返回示例：

```
{
    "matchedRegion": {
        "id": "test",
        "redirectDomain": "test.ablecloud.cn:9100",
        "routerDomain": "test.ablecloud.cn:9005",
        "webSocketDomain": "test.ablecloud.cn:9001"
    },
    "regionList": [{
        "id": "test",
        "redirectDomain": "test.ablecloud.cn:9100",
        "routerDomain": "test.ablecloud.cn:9005",
        "webSocketDomain": "test.ablecloud.cn:9001"
    }]
}
```

2.3.2 根据上述返回结果进行初始化配置

开发者可以自主选择连接的环境

```
Matrix.initI18N(activity.this.getApplicationContext(), dominName, dominId, routerDomain
, webSocketDomain, redirectDomain, id);
```

2.4 私有云配置

```
private Configuration privateConfiguration = new Configuration() {
        @Override
        public String getRouterAddress() {return routerAddress;}
        @Override
        public String getGatewayAddress() {return webSocketAddress;}
        @Override
        public String getDomainName() {return dominName;}
        @Override
        public String getRedirectAddress() {return redirectAddresss(可为null);}
```

```
        @Override
        public String getRegionDes() {return regions(可为null);}
        @Override
        public long getDomainId() {return dominId;}
    };
Matrix.init(this, privateConfiguration);
```

UDS通信

Matrix-SDK中通过 `MatrixService` 这个类完成对UDS的请求；

1. 配置MatrixService

```
public class ServiceDataManager extends MatrixService {

    private static volatile ServiceDataManager mServiceDataManager;

    public static ServiceDataManager getInstance() {
        if (mServiceDataManager == null) {
            synchronized (ServiceDataManager.class) {
                if (mServiceDataManager == null)
                    mServiceDataManager = new ServiceDataManager(UDS服务名称, 服务的版本
);
            }
        }
        return mServiceDataManager;
    }
    private ServiceDataManager(String name, int mainVersion) {
        super(name, mainVersion);
    }
}
```

`MatrixService` 这个类是SDK中提供的，开发者在开发的过程中，需要继承自该类，调用方式推荐使用单例的方式。该类与UDS映射，UDS有的接口，开发者均可在这个类中进行封装。

```
public class MatrixService {
    public void requestAsync(String apiName, Map<String, Object> param, MatrixCallback<
String> callback) {}
    ...
}
```

在 `MatrixService` 中提供了 `requestAsync` 方法，用来完成Http请求。

- **apiName**：UDS对应的 `requestName`
- **parm**：请求参数，无参数可为 `null`

2. 使用示例

```
Map deviceParam = new HashMap<>();
deviceParam.put("deviceId", 1);
```

```
ServiceDataManager.getInstance().requestAsync(apiName, DeviceParam, new MatrixCallback<
String>() {
     @Override
     public void success(String s) {
    }

     @Override
     public void error(MatrixError matrixError) {
    }
});
```

账号管理

账号管理的业务逻辑主要有SDK中 `AccountManager` 来管理，开发者可以通过 `Matrix.accountManager()` 来获取实例

1. 涉及的主要数据结构

```
public class User {
    //未赋值
    public static final int TYPE_UNDEFINED = -1;
    //设备用户
    public static final int TYPE_USER = 0;
    //设备管理员
    public static final int TYPE_OWNER = 1;

    /**
     * 用户ID
     */
    public long userId;

    /**
     * 用户电话号码
     */
    public String phone;

    /**
     * 用户邮箱
     */
    public String email;


    /**
     * 用户昵称
     */
    public String nickName;

    /**
     * 用户扩展属性,需要现在平台创建
     */
    public Map<String, Object> userProfile;

    /**
     * 设备用户类别
     */
    public int userType = TYPE_UNDEFINED;
```

```
}
```

2. 相关API

```
void login(String account, String password, final MatrixCallback<User> callback)
```

用户登录

参数：

- account：帐号名，注册时候email或phone任选其一
- password：用户密码

```
void logout()
```

退出登录

```
boolean isLogin()
```

检验是否登录

```
void loginWithOpenId(String thirdPlatformName, String openId, String thirdAccessToken,
final MatrixCallback<User> callback)
```

第三方登录

参数：

- thirdPlatformName：第三方类型，必须是以下之中的一种： qq、weibo、weixin、jingdong、facebook、twitter、instagram
- openId：通过第三方登录获取的openId
- thirdAccesssToken：通过第三方登录获取的accessToken

```
void associateThirdPlatfrom(@NonNull ThirdPlatform thirdPlatform, String openId, String
 accessToken,MatrixCallback<Void> callback)
```

关联第三方平台

参数：

- thirdPlatform：ThirdPlatform枚举
- openId：通过第三方登录获取的openId
- thirdAccesssToken：通过第三方登录获取的accessToken

```
void associatePhone(String phone, String password, String nickName, String verifyCode,
MatrixCallback<Void> callback)
```

给未设置手机号的User关联手机号

参数：

- phone：手机号码
- password：当前账号的密码
- nickName：名字
- verifyCode：验证码

```
void associateEmail(String email, String password, String nickName, String verifyCode,
MatrixCallback<Void> callback)
```

给未设置邮箱的User关联邮箱

参数：

- phone：需要关联的邮箱
- password：当前账号的密码
- nickName：名字
- verifyCode：验证码

```
void listAllPlatformInfo(MatrixCallback<List<ThirdPlatformInfo>> callback)
```

获取当前账号所关联的所有第三方平台

```
void register(String phone, String email, String password, String verifyCode, String ni
ckname, final MatrixCallback<User> callback)
```

用户通过手机号或者邮箱注册，二者均可，当选择手机号注册，则email设置为空，如果二者均填写以手机号为准

参数：

- phone：手机号
- email：邮箱号
- password：密码
- verifyCode：验证码

```
void requireVerifyCode(String account, int template, MatrixCallback<Void> callback)
```

发送验证码，在注册、忘记密码、通过验证码登录的时候，获取验证码；如果有需求要在这三种情况验证码为不同格式，在AbleCloud云端设置不同的短信模板/邮箱模板

参数：

- account：接收短信或者邮件的账户
- template：短信模板

```
void checkVerifyCode(String account, String verifyCode, final MatrixCallback<Boolean> c
allback)
```

检测验证码是否正确

参数：

- account：账号
- verifyCode：验证码

```
void changePassword(String oldPassword, String newPassword, MatrixCallback<User> callba
ck)
```

登录状态下修改密码

参数：

- oldPassword：旧密码
- newPassword：新密码

```
void resetPassword(String account, String password, String verifyCode, final MatrixCall
```

```
back<User> callback)
```

重置密码

参数：

- account：重置密码的账号
- password：新密码
- verifyCode：验证码

```
void getUserProfile(final MatrixCallback<Map<String, Object>> callback)
```

获取用户的附加属性，使用之前需在平台的用户服务添加用户的附加属性

```
setUserProfile(Map<String, Object> userProfile, final MatrixCallback<Void> callback)
```

修改用户的附加属性，变量更新

```
void uploadAvatar(byte[] data, final MatrixCallback<Integer> progress, final MatrixCall
back<String> callback)
```

上传用户的头像，以二进制的方式，上传成功，之后会将存储照片的URL返回，开发者可以将这个URL以用户的附加属性进行存储。

参数：

- data：二进制byte数据
- progress：上传进度

```
void uploadAvatar(File file, MatrixCallback<Integer> progress, final MatrixCallback<Str
ing> callback)
```

上传用户的头像，以文件的形式

参数：

- file：头像图片的file
- progress：上传进度

```
void checkAccountExist(String account, final MatrixCallback<Boolean> callback)
```

判断账号是否存在

参数：

- account：账号

```
void changePhone(String phone, String password, String verifyCode, MatrixCallback<Void>
 callback)
```

给当前user修改手机号

参数：

- phone：要更换的手机号
- password：密码
- verifyCode：验证码

```
void changeEmail(String email, String password, String verifyCode, MatrixCallback<Void>
 callback)
```

给当前user修改邮箱

参数：

- email：要更换的邮箱
- password：密码
- verifyCode：验证码

```
void changeNickName(String nickName, MatrixCallback<Void> callback)
```

修改该账号的昵称

参数：

- nickName：要修改的昵称

```
void loginWithVerifyCode(String account, String verifyCode, MatrixCallback<User> callback)
```

通过验证码登录

参数：

- account：账号
- verifyCode：验证码

设备管理

设备管理的业务由 `BindManager` 来处理，开发者可通过 `Matrix.bindManager()` 来获取 `BindManager` 的实例，开发业务逻辑；

1.涉及的主要数据结构

```
//在绑定成功、分享码绑定设备，listDevice这三个接口会涉及到该数据结构
public class Device {
    public static final int CLOUD_ONLINE = 1;
    public static final int LOCAL_ONLINE = 1 << 1;

    /**
     * 物理Id
     */
    public String physicalDeviceId;

    /**
     * 设备的逻辑id，是设备联云激活成功之后，AbleCloud云给设备分配的Id，不是设备物理地址，或者其他产品Id
     */
    public long deviceId;

    /**
     * 设备的名称
     */
    public String name;

    /**
     * 设备的子域
     */
    public long subDomainId;

    /**
     * 子域名称
     */
    public String subDomainName;

    /**
     * 管理员的Id
     */
    public long owner;

    /**
     * 在线状态 0不在线 1云端在线 2局域网在线 3云端和局域网同时在线
```

```
     */
    public int status;

    /**
     * 设备在云端创建的属性
     */
    public Map<String, Object> attributes;

    /**
     * 设备的绑定时间
     */
    public String bindTime;
}
```

2. 设备的绑定/解绑

```
void bindDevice(String subDomain, String physicalDeviceId, String name, final MatrixCallback<Device> callback)
```

绑定设备，需要注意的是绑定设备成功的关键是设备云端在线、并且该设备处于未绑定的状态，如果出现绑定失败的情况请先排除这两种情况。

参数：

- subDomain：设备子域
- physicalDeviceId：物理Id/mac地址
- name：设备的昵称（绑定成功的设备可通过 `changname()` 来为设备重命名）

```
void unbindDevice(long deviceId, MatrixCallback<Void> callback)
```

解绑设备

参数：

- deviceId：设备的逻辑id，是设备联云激活成功之后，AbleCloud云给设备分配的Id，不是设备物理地址，或者其他产品Id

---

```
void getBoundStatus(String physicalDeviceId, final MatrixCallback<Boolean> callback)
```

判断设备是否绑定

参数：

- physicalDeviceId：设备的物理地址

```
void getOnlineStatus(long deviceId, final MatrixCallback<Boolean> callback)
```

根据deviceId判断设备是否在线

参数：

- deviceId：设备的逻辑id，是设备联云激活成功之后，AbleCloud云给设备分配的Id，不是设备物理地址，或者其他产品Id

---

```
void getOnlineStatus(String physicalDeviceId, String subDomin, final MatrixCallback<Boolean> callback)
```

根据physicalDeviceId和subdomin获取设备的在线状态，这个方法，可以在设备未绑定的情况下调用，在AP配网设置WiFi信息成功之后，可调用该接口检测是否在线，若在线则可执行绑定操作

参数：

- physicalDeviceId：设备的物理地址
- subDomin：子域

---

3. 分享设备

```
void shareDevice(long deviceId, String account, MatrixCallback<Void> callback)
```

设备的管理员分享设备给指定账户

参数：

- deviceId：设备的逻辑id，是设备联云激活成功之后，AbleCloud云给设备分配的Id，不是设备物理地址，或者其他产品Id
- account：接收分享设备的账号

```
void unshareDevice(long deviceId, long userId, MatrixCallback<Void> callback)
```

设备的管理员解除给某一用户的分享

参数：

- deviceId：设备的逻辑id，是设备联云激活成功之后，AbleCloud云给设备分配的Id，不是设备物理地址，或者其他产品Id
- userId：待解除用户的用户Id，可通过listUser获取

```
void fetchShareCode(long deviceId, final MatrixCallback<String> callback
```

根据设备的deviceId，获取分享码，只有管理员可以获取 ，默认一小时内生效，开发者可以根据获取的分享码，生成二维码，开发二维码分享功能

参数：

- deviceId：设备的逻辑id，是设备联云激活成功之后，AbleCloud云给设备分配的Id，不是设备物理地址，或者其他产品Id
- callback：分享码

```
void adoptShareCode(String shareCode, final MatrixCallback<Device> callback)
```

接收通过 `fetchShareCode()` 获取的分享码，调用成功之后，该用户就获得了改分享码对应设备的使用权

- shareCode：分享码

```
void changeOwner(long deviceId, long userId, MatrixCallback<Void> callback)
```

设备管理员权限转让

- deviceId：设备的逻辑id，是设备联云激活成功之后，AbleCloud云给设备分配的Id，不是设备物理地址，或者其他产品Id
- userId：新的管理员Id（要求该用户也已经绑定过该设备）

```
void listUsers(long deviceId, final MatrixCallback<List<User>> callback)
```

获取所有可控制该设备的user

参数：

- deviceId：设备的逻辑id，是设备联云激活成功之后，AbleCloud云给设备分配的Id，不是设备物理地址，或者其他产品Id

---

4. 设备信息

```
void getDeviceProfile(String subDomain, long deviceId, final MatrixCallback<Map<String,
 Object>> callback)
```

获取设备的附加属性，开发如果需要使用设备附加属性，须得在云端先添加设备的附加属性

参数：

- subDomain：子域
- deviceId：设备的逻辑id，是设备联云激活成功之后，AbleCloud云给设备分配的Id，不是设备物理地址，或者其他产品Id
- callback：设备的附加属性，以 `key-value` 的形式返回

```
void setDeviceProfile(String subDomain, long deviceId, Map<String, Object> deviceProfil
e, MatrixCallback<Void> callback)
```

更新设备的附加属性，增量更新

参数：

- subDomain：子域
- deviceId：设备的逻辑id，是设备联云激活成功之后，AbleCloud云给设备分配的Id，不是设备物理地址，或者其他产品Id
- deviceProfile：需要更新的设备属性，以 `key-value` 的形式

```
void listDevices(final MatrixCallback<List<Device>> callback)
```

获取该用户已经绑定的所有设备，包括被分享的设备

```
void changeName(long deviceId, String name, MatrixCallback<Void> callback)
```

重命名设备的名称

参数：

- deviceId：设备的逻辑id，是设备联云激活成功之后，AbleCloud云给设备分配的Id，不是设备物理地址，或者其他产品Id
- name：设备的新昵称

定时任务

定时任务的业务是由 `DeviceTimerManager` 来完成的，开发者可以通过以下方式创建定时任务的实例

```
DeviceTimerManager mDevicenew = new DeviceTimerManager(deviceId)
```

定时任务的主要功能针对单个设备通过AbleCloud云定时给设备下发设置好的控制指令，对定时任务的描述均封装在 `DeviceTask` 类中，定时任务在执行完之后，云端是不会删除这个定时任务，只会将状态改为停止，对应 `DeviceTask` 中的status由1->0；

注意：

1、传入**SDK**的时间戳会被格式化为**"yyyy-MM-dd HH:mm:ss"**。

2、**timeCycle**需要在**timePoint**时间点的基础上,选择循环方式。

- **TimeCycle.ONCE**：单次循环
- **TimeCycle.HOUR**：在每小时的 `mm:ss` 时间点循环执行
- **TimeCycle.DAY**：在每天的 `HH:mm:ss` 时间点循环执行
- **TimeCycle.MONTH**：在每月的 `dd HH:mm:ss` 时间点循环执行
- **TimeCycle.YEAR**:在每年的 `MM-dd HH:mm:ss` 时间点循环执行
- **TimeCycle.parse("week[0,1,2,3,4,5,6]")**：在每星期的 `HH:mm:ss` 时间点循环执行(如周一，周五重复，则表示为 `week[1,5]`)

1. 涉及的数据结构

```
public class DeviceTask {
    private Long id;    //任务id，在编辑和删除任务的时候，需要指定taskId
    private String name; //任务的名字
    private String desc; //任务描述
    private Integer status; //任务执行状态 0停止 1执行
    private TimeRule timeRule; //执行的时间点和周期
    private Command command; //下发的指令
    private String createTime; //任务创建时间 开发者无需关心
    private String modifyTime; //任务编辑时间 开发者无需关心

    //设置执行的指令，开发者至于要构建一个MatrixMessage即可，关于MatrixMessage请看设备通信一节中MatrixMessage
    public DeviceTask setCommand(MatrixMessage message) {
        command = new Command();
        command.deviceCmd = new DeviceCmd();
        command.deviceCmd.code = String.valueOf(message.code);
        command.deviceCmd.binary = Base64.encodeToString(message.getContent(), Base64.N
```

```
O_WRAP);
        return this;
    }
    //设置执行周期和时间点
    public DeviceTask setTimeRule(long timePoint, TimeCycle timeCycle) {
        this.timeRule = new TimeRule(timePoint, timeCycle);
        return this;
    }
    //设置时区，如果不设置，则采取UTC时间
    public void setTimeZone(TimeZone timeZone) {
        timeRule.timeZone = timeZone.getID();
    }
}
```

```
public class DeviceTaskGroup {
    private String id; //任务组id
    private String name; //任务组名字
    private List<DeviceTask> tasks = new ArrayList<>(); //存放该任务组的所有任务
}
```

2. 单个定时任务

```
 private void createTimerTask() {
        //创建Task并且指定该Task的name
        final DeviceTask tripDeviceTask = new DeviceTask(name);
        //添加任务描述
        tripDeviceTask.setDesc(timerDesc);
         //执行的时间戳（毫秒）,执行一次
        tripDeviceTask.setTimeRule(timestamp, TimeCycle.ONCE);
         //定时执行的时区
        tripDeviceTask.setTimeZone(TimeZone.getDefault());
        //设置控制指令,msgCode为该条指令的功能码
        MatrixMessage message = new MatrixMessage(msgCode, new byte[]{0, 0, 0, 0});
        tripDeviceTask.setCommand(message);

        mDisposable = Completable
                .create(new CompletableOnSubscribe() {
                    @Override
                    public void subscribe(final CompletableEmitter e) throws Exception
{
                        mDeviceTimerManager.addTask(tripDeviceTask, new MatrixCallback<
DeviceTask>() {
                            @Override
                            public void success(DeviceTask deviceTask) {e.onComplete();
}
                            @Override
```

```
                            public void error(MatrixError matrixError) {e.onError(matri
xError);}
                        });
                    }
                })
                .subscribeOn(Schedulers.newThread())
                .observeOn(AndroidSchedulers.mainThread())
                .subscribe(new Action() {
                    @Override
                    public void run() throws Exception {
                    //处理创建成功的逻辑
                    }
                }, new Consumer<Throwable>() {
                    @Override
                    public void accept(Throwable throwable) throws Exception {
                    //处理创建失败的逻辑
                    }
                });
    }
```

添加单个定时任务

参数：

- deviceTask：定时任务

```
void listTasks(final MatrixCallback<List<DeviceTask>> callback)
```

获取指定设备的所有的定时任务

```
void updateTask(long taskId, DeviceTask deviceTask, final MatrixCallback<DeviceTask> ca
llback)
```

更新定时任务，更新后的任务默认开启

参数：

- taskId：更新任务的taskId
- deviceTask：修改后的定时任务

```
void deleteTask(long taskId, MatrixCallback<Void> callback)
```

删除一个定时任务

参数：

- taskId：更新任务的taskId

```
void enableTask(long taskId, MatrixCallback<Void> callback)
```

打开定时任务

参数：

- taskId：更新任务的taskId

```
void disableTask(long taskId, MatrixCallback<Void> callback)
```

关闭一个定时任务

参数：

- taskId：更新任务的taskId

---

3. 定时任务组

与设备单个定时任务的区别是：将多个定时任务进行分组管理。 以下场景举例：用户设置每天早上9点上班，下午17点下班。那么我们希望在每天8:30自动关闭空调，下午16:30提前打开空调，实现代码如下：

```
private void addTaskGroup() {
        DeviceTaskGroup deviceTaskGroup = new DeviceTaskGroup(name);

        DeviceTask closeTask = new DeviceTask("Close air condition");
        closeTask.setDesc("Go to work");
        MatrixMessage closeMessage = new MatrixMessage(msgCode, new byte[]{1, 0, 0});
        closeTask.setCommand(closeMessage);
        closeTask.setTimeRule(上午9点对应的时间戳, TimeCycle.parse("week[1,2,3,4,5]"));
        closeTask.setTimeZone(TimeZone.getDefault());
        deviceTaskGroup.add(closeTask);

        DeviceTask openTask = new DeviceTask("Open air condition");
        openTask.setDesc("Go off work");
        MatrixMessage openMessage = new MatrixMessage(msgCode, new byte[]{1, 1, 2});
```

```
        openTask.setCommand(openMessage);
        openTask.setTimeRule(16:30对应的时间戳, TimeCycle.parse("week[1,2,3,4,5]"));
        openTask.setTimeZone(TimeZone.getDefault());
        deviceTaskGroup.add(openTask);

        mCreatePlanTaskDisable = Completable
                .create(new CompletableOnSubscribe() {
                    @Override
                    public void subscribe(final CompletableEmitter e) throws Exception
{
                            mDeviceTimerManager.addTaskGroup(deviceTaskGroup, new Matri
xCallback<DeviceTaskGroup>() {
                                @Override
                                public void success(DeviceTaskGroup deviceTaskGroup) {
                                    e.onComplete();}
                                @Override
                                public void error(MatrixError matrixError) {
                                    e.onError(matrixError);}
                            });}})
                .subscribeOn(Schedulers.newThread())
                .observeOn(AndroidSchedulers.mainThread())
                .subscribe(new Action() {
                    @Override
                    public void run() throws Exception {
                        L.i("定时任务组创建成功");
                    }
                }, new Consumer<Throwable>() {
                    @Override
                    public void accept(Throwable throwable) throws Exception {
                        L.e("定时任务组创建失败");
                    }
                });

    }
```

添加定时任务组

参数：

- deviceTaskGroup：定时任务组

```
void listTaskGroups(final MatrixCallback<List<DeviceTaskGroup>> callback)
```

获取该设备所有的定时任务组

```
void updateTaskGroup(String groupId, DeviceTaskGroup taskGroup, final MatrixCallback<DeviceTaskGroup> callback)
```

更新定时任务组，更新后的任务组默认开启

参数：

- groupId：需要更新的任务组Id
- taskGroup：新的任务组

```
void deleteTaskGroup(String groupId, MatrixCallback<Void> callback)
```

删除任务组

参数：

- groupId：需要删除的任务组Id

```
void enableTaskGroup(String groupId, MatrixCallback<Void> callback)
```

打开指定任务组

参数：

- groupId：需要打开的任务组Id

```
void disableTaskGroup(String groupId, MatrixCallback<Void> callback)
```

关闭指定任务组

参数：

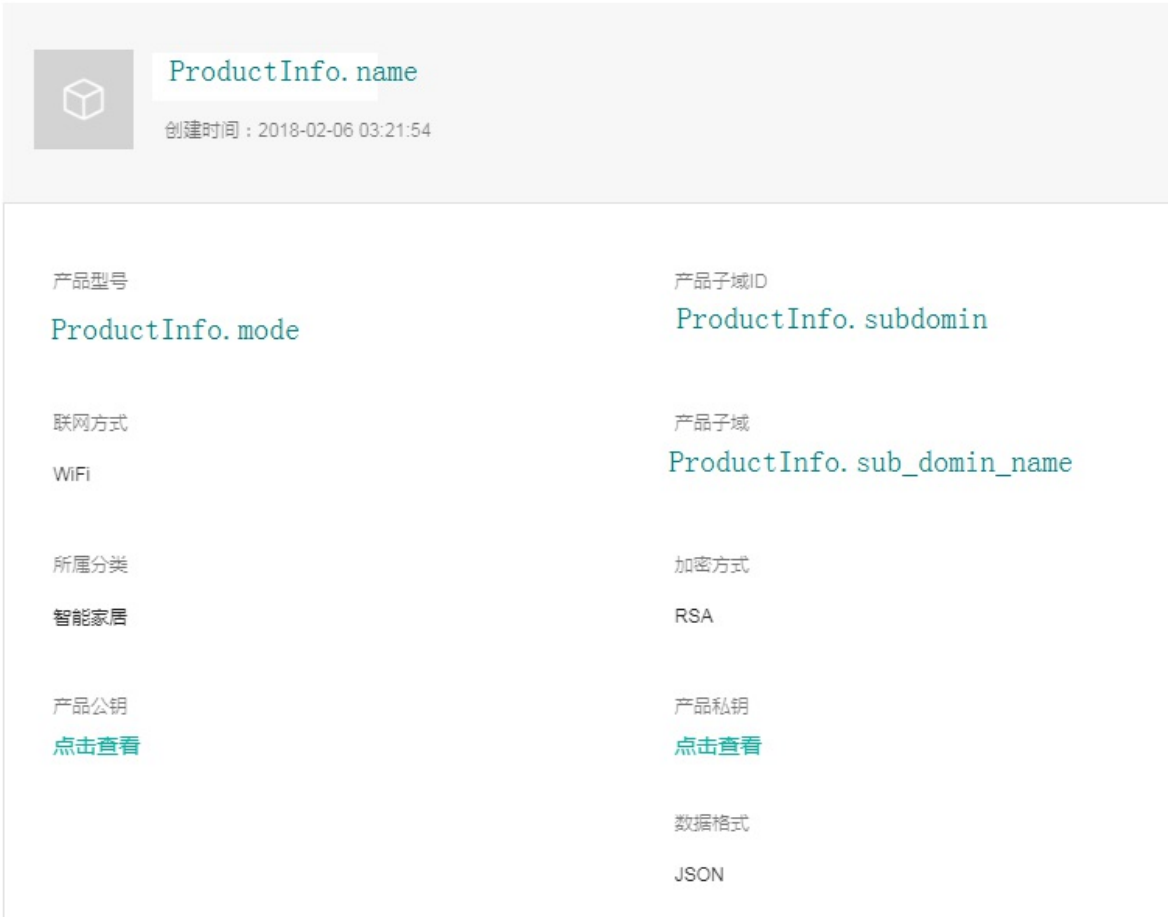
- groupId：需要关闭的任务组Id

产品管理

产品管理的业务逻辑由 `ProductManager` 来完成，开发者通过 `Matrix.productManager()` 获取实例。开发者通过产品管理功能，获取在AbleCloud云端创建的产品信息，包括：产品的名称、产品型号、主子域信息；

1. 涉及的数据结构

```
public class ProductInfo {
    public String domain;
    public String domain_name;
    public String sub_domain;
    public String sub_domain_name;
    public String name; //产品名称
    public String model; //产品型号
    public String productImageUrl;//产品的图片地址
    public String description; //产品型号
}
```



2. API

```
void getAllProducts(final MatrixCallback<List<ProductInfo>> callback)
```

获取该主域下所有的产品信息

参数：

- callback：所有产品的集合

```
void getProduct(String subDomain, final MatrixCallback<ProductInfo> callback)
```

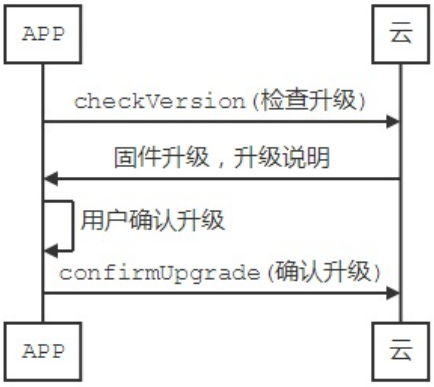
获取指定产品的信息

参数：

- subDomin：产品的子域

OTA

OTA的业务由 `OTAManager` 来处理，开发者可通过 `Matrix.otaManager()` 来获取实例开发对应功能，以下是大致流程



1. 检查升级

```
private void checkVersion() {
        VersionQuery versionRequest = new VersionQuery(subDomainName, otaType);
        versionRequest.setDeviceId(device.deviceId);
        Matrix.otaManager().checkVersion(versionRequest, new MatrixCallback<VersionResp
onse>() {
            @Override
            public void success(VersionResponse versionResponse) {
                //hasUpgrade为true则有新版，false则无新版本
                boolean hasUpgrade = versionResponse.hasUpgrade();
            }
            @Override
            public void error(final MatrixError matrixError) {
               //处理check失败的逻辑
            }
        });
    }
```

检查升级的时候需要构造一个 `VersionQuery` 实例，参数分别为 `subDominName` , `otaType` ， `otaType` 有两种取值分别为：

- VersionQuery.TYPE_MCU：检查MCU升级情况
- VersionQuery.TYPE_WIFI：检查WiFi固件升级情况

2. 确认升级

```
private void confirmUpgrade() {
        UpgradeRequest upgradeRequest = new UpgradeRequest(subDomainName, deviceId, tar
getVersion, otaType);
        Matrix.otaManager().confirmUpgrade(upgradeRequest, new MatrixCallback<Void>() {
            @Override
            public void success(Void aVoid) {// confirmUpgrade success}
            @Override
            public void error(MatrixError matrixError) {//confirmUpgrade error}
        });
    }
```

确认升级的时候需要构造一个 `UpgradeRequest` 实例

参数分别为：

- subDomainName：子域
- deviceId：设备的逻辑id，是设备联云激活成功之后，AbleCloud云给设备分配的Id，不是设备物理地址，或者其他产品Id
- targetVersion：确认升级的版本，是 `checkVersion` 方法返回的 `VersionResponse` 中 `targetVersion`
- otaType：同检查升级的ota类型

数据订阅

在实际的开发过程中，类似对数据的实时性需求不可避免，所以SDK中提供了对应的API来完成对数据的实时更新，目前支持三类型数据实时订阅：

- 设备属性
- UDS数据集
- 自定义数据

在AbleCloud平台进行设备开发，强烈建议将设备的功能参数定义为属性。利用属性的相关功能(协议解析等)，开发者可以对属性数据进行持久化存储并获取历史数据，设备属性的详细介绍 。

设备属性订阅和加速上报

本节包括设备属性订阅、加速上报；

要进行属性订阅，需要先在平台创建设备属性，如下：

1. 设备属性订阅

要完成设备的属性订阅，完整的过程需要经过三个步骤

1. 订阅
2. 创建和添加监听
3. 移除监听(防止内存泄漏)

```
void subscribeProperty(String subDomain, long deviceId, MatrixCallback<Void> callback)
```

订阅一台设备的属性

参数：

- subDomain：子域
- deviceId：设备的逻辑id，是设备联云激活成功之后，AbleCloud云给设备分配的Id，不是设备物理地址，或者其他产品Id

---

```
void registerPropertyReceiver(PropertyReceiver receiver)
```

注册设备属性的监听

参数：

- receiver：监听，需要实现PropertyReceiver接口

```
public interface PropertyReceiver {
    //这里的value是全量推送，所有的设备属性
    void onPropertyReceive(String subDomain, long deviceId, String value);
```

```
}
```

```
void unsubscribeProperty(String subDomain, long deviceId, MatrixCallback<Void> callback)
```

取消一台设备的订阅

参数：

- subDomain：子域
- deviceId：设备的逻辑id，是设备联云激活成功之后，AbleCloud云给设备分配的Id，不是设备物理地址，或者其他产品Id

```
void unsubscribeAllProperty()
```

取消所有设备的订阅关系

```
void unregisterPropertyReceiver(PropertyReceiver receiver)
```

移除设备的监听

注意：可以每一台设备进行订阅，但是设备属性监听只有一个，所有的设备的订阅数据，均通过这个 `receiver` 来完成，如果有多台设备中只有一台设备取消订阅，可以只通过 `unsubscribeProperty` 完成即可

2. 加速上报

在开发过程中，当APP打开时，有些设备属性的值是比较看重实时性的，例如温度、PM2.5、TDS值等，这时候需要设备加快上报的速率；

```
void enableDeviceFastReport(String subDomain, long deviceId, int interval, final Matrix
Callback<Void> callback)
```

加速设备上报

参数：

- subDomain：子域
- deviceId：设备的逻辑id，是设备联云激活成功之后，AbleCloud云给设备分配的Id，不是设备物理地址，或者其他产品Id
- interval：控制设备上报数据的间隔时间 范围限制为：(0, 60] 秒。

```
void disableDeviceFastReport(String subDomain, long deviceId, MatrixCallback<Void> callback)
```

取消设备的加速上报

参数：

- subDomain：子域
- deviceId：设备的逻辑id，是设备联云激活成功之后，AbleCloud云给设备分配的Id，不是设备物理地址，或者其他产品Id

---

```
void disableAllDeviceFastReport()
```

取消所有设备的加速上报

订阅设备在线状态

AbleCloud支持对设备状态的监听，需要注意的是：设备由”在线->离线“的状态的敏感度比较低，这是因为当云端收不到三次设备发出的心跳包才认为设备已经离线了，从而避免由于网络状况不稳定带来的干扰，而设备有”离线->在线“的状态很灵敏。

```
void subscribeOnlineStatus(String subDomain, long deviceId, MatrixCallback<Void> callba
ck)
```

订阅在线状态

参数：

- subDomain：子域
- deviceId：设备的逻辑id，是设备联云激活成功之后，AbleCloud云给设备分配的Id，不是设备物理地址，或者其他产品Id

---

```
void registerOnlineStatusListener(OnlineStatusListener listener)
```

注册在线状态的监听

参数：

- listener：监听

```
public interface OnlineStatusListener {
   void onStatusChanged(String subDomain, long deviceId, boolean online);
}
```

```
void unsubscribeOnlineStatus(String subDomain, long deviceId, MatrixCallback<Void> call
back)
```

取消订阅在线状态

参数：

- subDomain：子域
- deviceId：设备的逻辑id，是设备联云激活成功之后，AbleCloud云给设备分配的Id，不是设备物理地址，或者其他产品Id

---

```
void unsubscribeAllOnlineStatus()
```

取消所有设备的订阅

```
void unregisterOnlineStatusListener(OnlineStatusListener listener)
```

移除在线状态的监听

参数：

- listener：监听

注意：可以每一台设备进行订阅，但是设备在线状态监听只有一个，所有设备的订阅数据，均通过这个 **`listener`** 来完成，如果有多台设备中只有一台设备取消订阅，可以只通过 **`unsubscribeOnlineStatus`** 完成即可

订阅UDS数据集

对于存储在数据集中的数据，APP可以进行实时消息订阅，进行订阅后，APP和云端建立长连接，数据集中的数据发生改变后，云端会将最新的数据实时推送给APP

使用之前, 需要先到AbleCloud中控制台查看对应的数据集

> 1. 数据集对象中涉及到字符串比较多, 如果遇到订阅失败请检查拼写是否正确
> 2. 如果一个数据集有两个及以上主键, 则必须按顺序订阅, 即如果要订阅主键二, 则实现方式为订阅主键一和主键二, 不能单独订阅主键二
> 3. 如果主键是设备Id, 那么该设备必须与当前用户是绑定关系

```
void subscribe(String className, Map<String, Object> primaryKey, int opType, MatrixCallback<Void> callback)
```

订阅数据集

参数：

- className：数据表名，例如上述图中：indoor_air
- primaryKey：主键 例如上述图：`primaryKey.put("deviceId",6)` 监听deviceId==6的数据
- opType：
  - ClassTopic.OPTYPE_CREATE：数据创建
  - ClassTopic.OPTYPE_REPLACE：数据替换
  - ClassTopic.OPTYPE_UPDATE：数据更新
  - ClassTopic.OPTYPE_DELETE：数据删除
  - ClassTopic.OPTYPE_ALL：数据所有写操作

```
void registerDataReceiver(ClassDataReceiver receiver)
```

注册监听

参数：

- receiver：监听

```
public interface ClassDataReceiver {
    /**
     * 接收实时数据
     *
     * @param className 数据集名称
     * @param value     JSON格式数据
     */
    void onReceive(String className, int opType, String value);
}
```

```
void unsubscribe(String className, Map<String, Object> primaryKey, int opType, MatrixCallback<Void> callback)
```

取消订阅

参数：

- className：数据表名，例如上述图中：indoor_air
- primaryKey：主键 例如上述图：`primaryKey.put("deviceId",6)` 监听deviceId==6的数据
- opType：
  - ClassTopic.OPTYPE_CREATE：数据创建
  - ClassTopic.OPTYPE_REPLACE：数据替换
  - ClassTopic.OPTYPE_UPDATE：数据更新
  - ClassTopic.OPTYPE_DELETE：数据删除
  - ClassTopic.OPTYPE_ALL：数据所有写操作

```
void unregisterDataReceiver(ClassDataReceiver receiver)
```

移除监听

订阅自定义数据

用户可以自定义数据订阅

在完成订阅自定义数据，开发者需要构建一个 `Topic`

```
public class Topic {
    /**
     * 创建自定义Topic
     *
     * @param subDomain 子域,可以为空，空表示主域级别
     * @param type      自定义类别
     * @param key       自定义关键字
     * @return
     */
    public static Topic customTopic(String subDomain, String type, String key) {
        if (isPreserved(type)) {
            throw new IllegalArgumentException("Type \"" + type + "\" preserved for int
ernal use");
        }
        return new Topic(subDomain, type, key);
    }
}
```

其中的 `type` 和 `key` 需要云端支持

```
void subscribe(Topic topic, MatrixCallback<Void> callback)
```

订阅

```
void unsubscribe(Topic topic, MatrixCallback<Void> callback)
```

取消订阅

```
void registerDataReceiver(MatrixReceiver<TopicData> receiver)
```

添加监听

```
void unregisterDataReceiver(MatrixReceiver<TopicData> receiver)
```

移除监听

文件存储

AbleCloud支持文件的上传和下载，SDk中关于文件下载以及存储的业务由 `BlobStoreManager` 来处理，开发者可通过 `Matrix.blobStoreManager()` 来获得实例，完成业务逻辑，需要注意的是，要完成下载到sdcard和将sdcard的文件上传需要额外增加以下权限：

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

1. 请求参数的数据结构

1.1 下载数据结构

```
public class DownloadBlob {
    public final String bucket; //可理解为文件存储在云端的目录
    public final String name;   //文件名
    public final long expireTime; //下载的过程中会产生一个临时的下载链接，为保证安全，需要给这个链接设置一个有效时间,单位是秒
}
//上传文件时若UploadBlob中AccessType为PUBLIC，则expireTime参数无效；默认情况为false
//如:24*60*60代表url链接有效时间，即1天

//开发者自己维护通过这两个参数保证上传的所有文件在云端不会重名，建议通过UUID的方式命名文件或者以用户 / 设备唯一标识命名bucket），重目录重名的情况下原文件会被覆盖。另外可通过这两个参数获取到下载的url。
```

1.2 上传数据结构

```
public class UploadBlob {
    public final String bucket;//见上
    public final String name;
    public final AccessType accessType; //文件的权限

     public enum AccessType {
        PUBLIC, PRIVATE
    }
}
```

2. 上传文件

```
void uploadBlob(final UploadBlob uploadBlob, final byte[] data, final MatrixCallback<Integer> progress, final MatrixCallback<String> callback)
```

通过二进制的方式上传文件

参数：

- uploadBlob：上传文件信息
- data：文件的二进制数组
- progress：进度回调
- callback：上传成功后返回的URL

```
void uploadFile(final UploadBlob uploadBlob, final File inputFile, final MatrixCallback
<Integer> progress, final MatrixCallback<String> callback)
```

通过文件形式上传

参数：

- uploadBlob：上传文件信息
- inputFile：文件对象
- progress：进度回调
- callback：上传成功后返回的URL

---

3. 下载文件

```
void downloadBlob(DownloadBlob downloadBlob, final MatrixCallback<Integer> progress, fi
nal MatrixCallback<byte[]> callback)
```

二进制方式下载文件

参数：

- downloadBlob：下载文件信息
- progress：下载进度
- callack：文件信息

```
void downloadFile(DownloadBlob downloadBlob, final File outputFile, final MatrixCallbac
k<Integer> progress, final MatrixCallback<Void> callback)
```

下载文件在本地生成具体文件（注意申请写入sdcard权限）

参数：

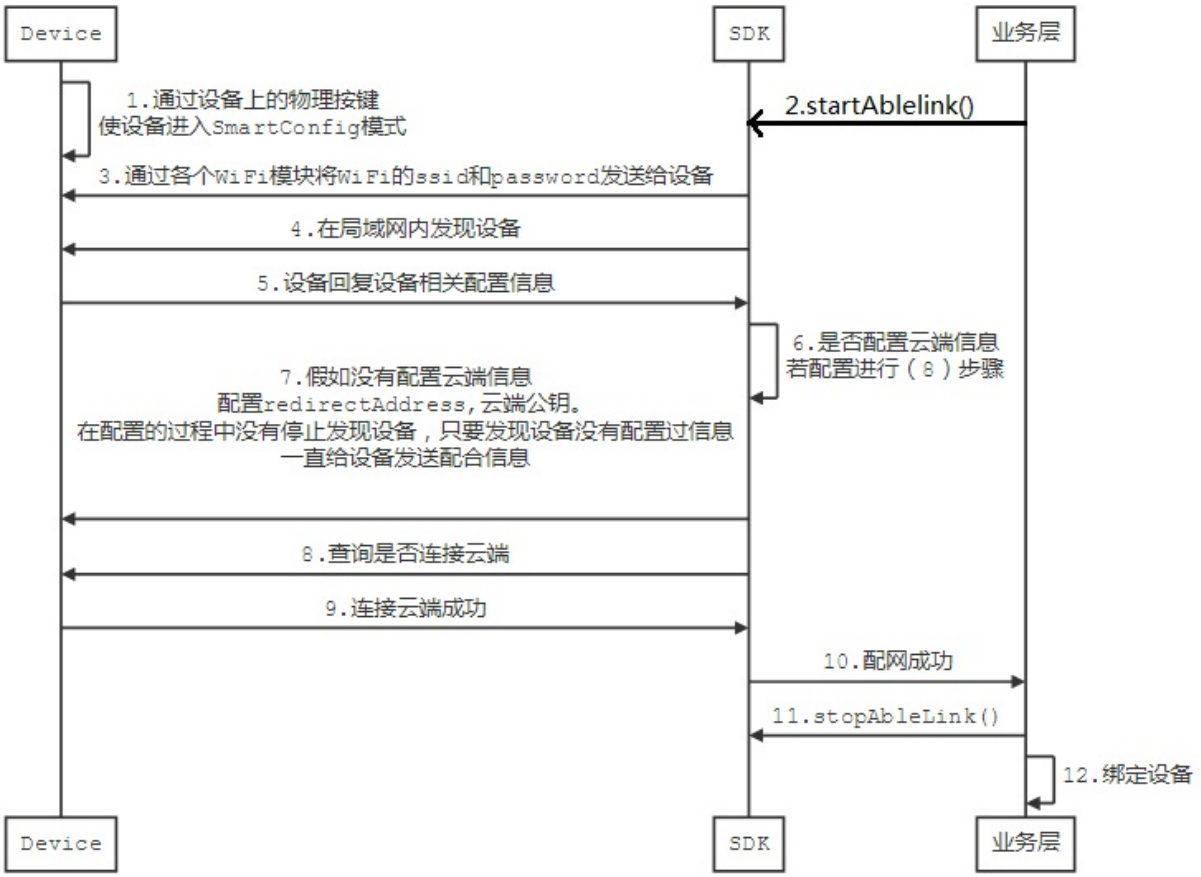
- downloadBlob：下载文件信息
- outputFile：本地文件信息
- progress：下载进度
- callack：文件信息

SmartConfig配网

关于SmartConfig可分为以下几类，开发者可根据实际情况去调用对应的API

- 国际化SmartConfig配网（**WiFI**固件版本**V7**及以上）
  - 纯局域网配网
  - 普通SmartConfig配网(设备需要连接云端)
- 非国际化SmartConfig配网（**WiFI**固件版本低于**V7**）
  - 纯局域网配网
  - 普通SmartConfig配网(设备需要连接云端)

> SmartConfig配网需要各个厂家的配网库支持，目前AbleCloud支持的配网库有：
>
> ```
> public enum DeviceType {
>     HIFLYING, //汉枫模块
>     MXCHIP,   //庆科模块
>     MX_LEGACY,//乐鑫模块
>     QC_SNIFFER,
>     ESP8266,
>     REALTEK,
>     AI6060H,
>     BROADLINK,//古北模块
>     QC_LTLINK
> }
> ```
>
> ## 1.国际化**SmartConfig**配网流程



需要注意的是：纯局域网下是没有云端配置信息，默认的是测试环境配置信息；

非国际化配网流程

2.具体API

2.1 国际化SmartConfig配网

```
  MatrixLocal.smartModeNetConfigManager()
      //需要连接云端，如果是纯局域网，configNetType(SmartModeNetConfigManager.NET.LOCAL)
      .configNetType(SmartModeNetConfigManager.NET.CLOUD)
      //这里采用了乐鑫配网
      .startAblelink(DeviceType.MX_LEGACY, NET_SSID, PASSWORD, 60 * 1000, new MatrixCal
lback<LocalDevice>() {
          @Override
          public void success(LocalDevice localDevice) {
            //配网成功
          }

          @Override
          public void error(MatrixError matrixError) {
             //配网失败
          }
      });
```

2.2 非国际化SmartConfig配网(需要连接云端)

```
MatrixLocal
    .localDeviceManager()
    .startAblelink(DeviceType type, String ssid, String password, int timeout_ms, Matri
xCallback<LocalDevice> callback);
```

参数：

- type：WiFi模块类型
- ssid：WiFi名称
- password：WiFi密码
- time_ms：超时时间

2.3 非国际化纯局域网SmartConfig配网

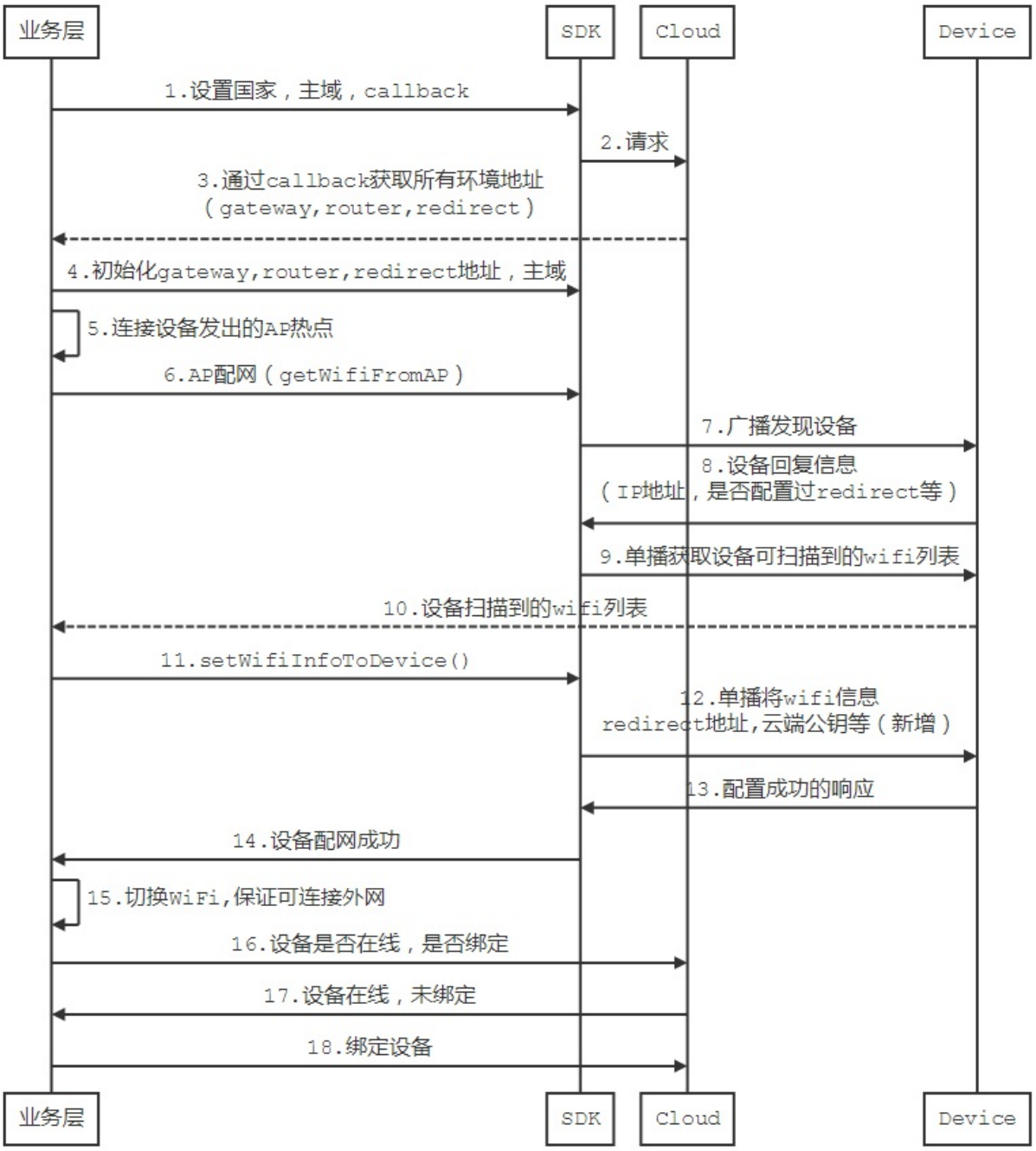
```
private void startAbleLink() {
        final DeviceActivator activator = DeviceActivator.of(DeviceType.MX_LEGACY);
        activator.startSmartConfig("ssid", "psw", 60 * 1000);
        AbleFind ableFind = new AbleFind(60 * 1000, 500, new MatrixReceiver<LocalDevice
>() {
            @Override
            public void onReceive(LocalDevice localDevice) {
                //发现设备
                activator.stopSmartConfig();
                //如果想要停止发现 调用stopFind()
            }
        });
        ableFind.execute(new MatrixCallback<Void>() {
            @Override
            public void success(Void aVoid) {

            }

            @Override
            public void error(MatrixError matrixError) {
            }
        });
    }
```

AP配网

关于AP配网可分为以下几类，开发者可根据实际情况去调用对应的API

- 国际化AP配网（**WiFI**固件版本**V7**及以上）
- 非国际化AP配网（**WiFI**固件版本低于**V7**）

1.国际化AP配网



非国际化AP配网

2.具体API

2.1 国际化AP配网

开发者可通过 `MatrixLocal.apModeNetConfigManager()` 获得配网实例，完成配网；

第一步：获取设备可扫描到的**WiFi**列表

```
MatrixLocal
    .apModeNetConfigManager()
    .getWififromDevice(60 * 1000, new MatrixCallback<List<MatrixWifiInfo>>() {
         @Override
        public void success(List<MatrixWifiInfo> matrixWifiInfos) {
            //获取wifi列表成功，根据设备返回的wifi信息，检查设备是否能发现目标WiFi
            for (MatrixWifiInfo matrixWifiInfo : matrixWifiInfos) {
                if (matrixWifiInfo.ssid.equals(SSID)) {
                   //setWifiToAP，将指定的WiFi信息发送给设备
                 }
             }
        }

        @Override
        public void error(MatrixError matrixError) {
            //获取wifi列表失败
        }
    });
```

第二步：将指定**WiFi**信息发送给设备

```
MatrixLocal
     .apModeNetConfigManager()
     .setWifiToAP(SSID, PASSWORD, 60 * 1000, new MatrixCallback<LocalDevice>() {
        @Override
        public void success(LocalDevice localDevice) {
            //设备配网成功,切换WiFi使手机能够连接外网（纯局域网可省略）
        }

        @Override
        public void error(MatrixError matrixError) {
            //设备配网失败
        }
     });
```

第三步：切换到目标**WiFi**，使手机和设备处于同一局域网

切换WiFi方法

第四步：发现设备

> 通过BindManager#getOnlineStatus(String physicalDeviceId, String subDomin, final MatrixCallback callback)获取设备是否在线（纯局域网可省略），

2.2 非国际化AP配网

开发者可通过 `MatrixLocal.localDeviceManager()` 获得配网实例，完成配网；

第一步：获取设备可扫描到的**WiFi**列表

```
MatrixLocal
    .localDeviceManager()
    .getWifiFromAP(GET_OR_WIFI_TIMEOUT_MS, new MatrixCallback<List<MatrixWifiInfo>>() {
         @Override
         public void success(List<MatrixWifiInfo> matrixWifiInfos) {
         //发现成功
            for (MatrixWifiInfo matrixWifiInfo : matrixWifiInfos) {
                if (matrixWifiInfo.ssid.equals(mWifi_ssid)) {
                   //setWifiToAP
                 }
             }
         }

         @Override
         public void error(MatrixError matrixError) {
        //发现失败
        }
});
```

第二步：将指定**WiFi**信息发送给设备

```
MatrixLocal
    .localDeviceManager()
     .setWifiToAP(
         mWifi_ssid, mWifi_psw
        , GET_OR_WIFI_TIMEOUT_MS
        , new MatrixCallback<Void>() {
            @Override
            public void success(Void aVoid) {
                //设置wifi成功，下一步切换wifi,调用发现接口，发现设备
            }

            @Override
            public void error(MatrixError matrixError) {
                //设置WiFi失败
            }
```

```
        });
```

第三步：切换到目标**WiFi**，使手机和设备处于同一局域网

切换WiFi方法

第四步：发现设备

```
MatrixLocal
    .localDeviceManager()
    .findLinkingDevice(
        FINDLINK_TIME_OUT_MS
        , INTER_VAL_MS
        , new MatrixCallback<LocalDevice>() {
            @Override
            public void success(LocalDevice localDevice) {
            //发现设备成功，可以进行绑定
            }

            @Override
            public void error(MatrixError matrixError) {
             //发现设备失败
            }
        });
```

设备通信

说明：在设备尚未开发完成时，在管理后台可以启动虚拟设备用于APP的调试。虚拟设备和真实设备使用方法相同，需要先绑定再使用。虚拟设备能够显示APP发到设备的指令，上报数据到云端、填入数据供APP查询。

1. 向设备发送消息

```
public void sendToDevice() {
       byte[] content = {0x12,0x25};
        MatrixMessage message = new MatrixMessage(code, content);
        Matrix.bindManager().sendDevice(getSubDomainName()
                                        ,physicalDeviceId
                                        , BindManager.Mode.CLOUD_ONLY
                                        , message
                                        , new MatrixCallback<MatrixMessage>() {
            @Override
            public void success(MatrixMessage matrixMessage) {
                //下发控制成功
            }

            @Override
            public void error(MatrixError matrixError) {
               //下发控制失败
            }
        });
    }
```

参数：

- subDomin：子域
- physicalDeviceId：设备的物理地址
- mode：枚举类型，发送的模式有四种，`Mode.LOCAL_ONLY`，`Mode.CLOUD_ONLY`，`Mode.LOCAL_FIRST`，`Mode.CLOUD_FIRST`
- message：下发指令的载体

关于**MatrixMessage**

```
public class MatrixMessage {
    public int code;
    protected byte[] content;

    public MatrixMessage(int code, byte[] content) {
        this.code = code;
```

```
        this.content = content;
    }
}
//在下发指令的时候，根据协议，需要知道该条控制指令的code(功能码)，以及具体的指令
//eg: MatrixMessage message = new MatrixMessage(65,new byte[]{0xDF,0x23,0x45})
```

2.纯局域网控制

注意：在进行局域网控制之前，请先调用 `MatrixLocal.localDeviceManager().findDevice()` 来发现该局域网下设备。纯局域网下发控制给设备，有两个重载方法供开发者调用；

```
void sendDevice(String physicalDeviceId, MatrixMessage message, MatrixCallback<MatrixMessage> callback)
```

```
void sendDevice(final LocalDevice local, final MatrixMessage message, final int timeoutMs, final MatrixCallback<? extends MatrixMessage> callback)
```

参数：

- physicalDeviceId：设备的物理Id
- message：控制指令的载体
- timeoutMs：超时时间（毫秒）

3.局域网订阅

3.1 数据订阅

```
void registerLocalDataReceiver(DataReceiver receiver)

public interface DataReceiver {
        /**
         * Receive local device data
         *
         * @param subDomainId      Device subdomain id
         * @param physicalDeviceId Device physical device
         * @param matrixMessage    Device data message
         */
        void onDataReceive(long subDomainId, String physicalDeviceId, MatrixMessage matrixMessage);
    }
```

注册局域网数据监听

```
void unregisterLocalDataReceiver(DataReceiver receiver)
```

取消局域网数据监听

3.2 设备数量变化订阅

当局域网有新设备上线，或者离线的时候可通过注册设备数量变化监听来知晓

```
 void registerLocalDeviceObserver(DataSetObserver observer)
```

注册设备数量变化监听

```
void unregisterLocalDeviceObserver(DataSetObserver observer)
```

移除设备数量变化监听

```
 List<LocalDevice> getWatchedDevices()
```

当发现本地设备变化，开发者调用 `getWatchDevices()` 来发现本地所有的设备

意见反馈

AbleCloud提供APP端的用户意见反馈接口。主要的业务逻辑 `FeedbackHelper` 来负责，开发者可以开发用户提交意见的页面。用户意见反馈可以反馈的项由开发者自己定义。 使用意见反馈前,需要先在控制台设置反馈项参数；

1. 代码示例

```
final File[] attachments = {new File(dir, "feedback1.png"), new File(dir, "feedback2.pn
g")};

Observable.fromArray(attachments).flatMapSingle(new Function<File, SingleSource<String>
>() {
    @Override
    public SingleSource<String> apply(final File file) throws Exception {
        return Single.create(new SingleOnSubscribe<String>() {
            @Override
            public void subscribe(final SingleEmitter<String> e) throws Exception {
                FeedbackHelper.uploadAttachment(file, null, new MatrixCallback<String>(
) {
                    @Override
                    public void success(String s) {e.onSuccess(s);}
                    @Override
                    public void error(MatrixError matrixError) {e.onError(matrixError);
}
                });
            }
        }).timeout(BLOB_TIMEOUT_MS, TimeUnit.MILLISECONDS);
    }
}).collectInto(new ArrayList<String>(), new BiConsumer<List<String>, String>() {
    @Override
    public void accept(List<String> strings, String s) throws Exception {
        strings.add(s);
    }
}).flatMapCompletable(new Function<ArrayList<String>, CompletableSource>() {
    @Override
```

```
    public CompletableSource apply(final ArrayList<String> strings) throws Exception {
        return Completable.create(new CompletableOnSubscribe() {
            @Override
            public void subscribe(CompletableEmitter e) throws Exception {
                Map<String, Object> extend = new HashMap<>();
                extend.put("description", "feedback: " + new Date().toString());
                extend.put("pictures", strings);
                FeedbackHelper.submitFeedback(null, null, extend, new CompletableCallba
ck<>(e, Void.class));
            }
        }).timeout(TIMEOUT_MS, TimeUnit.MILLISECONDS);
    }
}).blockingAwait();
```

示例中涉及两个方法

```
static void uploadAttachment(File file, MatrixCallback<Integer> progress, MatrixCallbac
k<String> callback)
```

上传意见反馈中的图片

参数

- file：图片
- progress：上传进度
- callback：图片存储的地址（URL）

```
static void submitFeedback(String subDomain, String type, Map<String, Object> extend, f
inal MatrixCallback<Void> callback)
```

提交意见反馈

参数：

- subDomin：子域，可以为空,也可以指定subDomain产品
- type：预留字段，可以为空
- extend：提交的具体信息，key为在云端创建的字段，value为提交的值

室外天气

SDK可以获取到室外的pm2.5, AQI(空气质量)以及天气状况，由 `WeatherHelper` 来处理；

```
static void getCurrent(String dataType, String area, MatrixCallback<String> callback)
```

获取实时天气情况

参数：

- dataType：可以是 `WeatherHelper.TYPE_PM25` , `WeatherHelper.TYPE_AQI` , `WeatherHelper.TYPE_WEATHER` 中的一种
- area：城市名称

```
static void getHistory(String dataType, String area, int time, TimeUnit unit, final Mat
rixCallback<String> callback)
```

获取历史天气情况

参数：

- dataType：可以是 `WeatherHelper.TYPE_PM25` , `WeatherHelper.TYPE_AQI` , `WeatherHelper.TYPE_WEATHER` 中的一种
- area：城市名称
- time：最近{time}个小时的数据
- unit：时间单位， `TimeUnit.DAYS` 和 `TImeUnit.HOURS` 中的一种
