家之安设备管理平台

1.0.1 API 文档

V1.0.1

2020 年 05 月 09 日

版本记录

版本	日期	修改内容	修改人
1.0.0	2019/10/30	初始版本	叶青海
1.0.1	2020/05/12	1、修正排版 2、修改 token 刷新接口参数 refreshToken 3、调整设备命令参数结构定义	叶青海

目录

一、概述

1.1 简述

家之安设备管理平台，是为接入平台的用户提供数据接口服务的管理平台。接入平台在设备管理平台获得授权后，利用本文档提供的 API 接口，进行设备的注册、管理、查询、下行控制等功能。

二、接入说明

2.1 授权说明

在使用家之安设备管理平台 API 之前，需要完成以下授权工作:

1) 申请账号:填写接入平台信息表（基本数据：企业名称、简称、联系人姓名、手机号、用户邮箱、推送 URL 地址以及选择是否加密）并递交给家之安设备管理平台管理者，审核通过后，家之安设备管理平台将账号信息提供给接入平台（通过邮件或其它形式）。

账号信息包含 AppId 、 Appsecret，如果选择加密发送，还会提供消息加密密钥。accessToken 调用接口生成，默认有效期是 7200 秒。

2.2 名词解释

AppId:为家之安设备管理平台对接入平台的唯一身份标识，家之安设备管理平台需通过 AppId 来鉴别应用的身份，接口调用中的请求参数名称使用 appId。

Appsecret:为家之安设备管理平台对接入平台分配的身份密码，用于保证用户使用 API 接口的可靠性，避免应用被伪造，或是被不法人员所使用等。为保障开发者的合法权益，请妥善保存 Appsecret 身份密码，接口调用中的请求参数名称使用 secret。

Token（令牌）:为家之安设备管理平台推送给接入平台的唯一性验证，实现推送指令时，可以识别是家之安设备管理平台推送的数据，确保接入平台数据来源的准确性和安全性。

消息加解密秘钥:若接入平台选择加密通信，则使用该秘钥进行数据加解密。

数据解析格式均为 JSON 格式。

三、开发指南

3.1 开发环境

家之安设备管理平台提供的 API 使用 HTTPS 协议，与开发语言无关，各种开发语言均可支持。

3.2 通用 URL

正式服务器地址:

<API_ADDRESS>= iot.jzasafe.com

备注说明:暂无。

四、接口详细说明

4.1 获取 accessToken

接口功能:

通过 appId 和 secret 获取 accessToken ，其余所有调用请求的 header 中传入 accessToken。

接口原型:

请求方法	POST
请求地址	https://<API_ADDRESS>/api/v1.0.0/auth
请求协议	HTTPS

请求参数:

参数	必选	类型	位置	描述
appId	Y	String（256）	请求体 body	身份标识
secret	Y	String（256）	请求体 body	身份密码

响应参数:

Status Code:200 OK

参数	必选	类型	描述
datas	Y		数据集合
access_token	Y	String（256）	鉴权参数，访问接口的凭证
token_type	Y	String（256）	Access_token 类型，默认 Bearer
refresh_token	Y	String（256）	鉴权参数，用来刷新
expires_in	Y	String（256）	平台生成并返回 access_token 有效期，单位为秒
scope	Y	String（256）	申请权限范围，默认为 all
resp_code	Y	String（256）	请求执行结果。0 失败 1 成功
Resp_msg	Y	String（256）	执行结果返回的信息

请求示例：

```
Method:POST
Request:
https://<API_ADDRESS>/api/v1.0.0/auth
Content-Type:application/json
Body:
{
    "appId":"xxxxxx",
    "secret":"xxxxxx"
}
```

响应示例：

```
Response:
Status Code:200 OK
Content-Type:application/json
Body:
{
    "datas": {
        "access_token": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
        "token_type": "bearer",
        "refresh_token": "xxxxxxxxxxxxxxxxxxxxxxxxxxxx",
        "expires_in": 6329,
        "scope": "all"
    },
    "resp_code": 1,
    "resp_msg": "操作成功"
}
```

错误消息：

resp_code	说明
0	1、 appId 或 secret 错误。 2、 appId 或 secret 不能为空

4.2 刷新 accessToken

接口功能:

accessToken 将要过期的时候，重新刷新获取 accessToken。

接口原型:

请求方法	POST
请求地址	https://<API_ADDRESS>/api/v1.0.0/refreshToken
请求协议	HTTPS

请求参数:

参数	必选	类型	位置	描述
appId	Y	String（256）	请求体 body	身份标识
secret	Y	String（256）	请求体 body	身份密码
refreshToken	Y	String（256）	请求体 body	刷新 accessToken，用来获取新的 accessToken

响应参数

Status Code:200 OK

参数	必选	类型	描述
datas	Y		数据集合
access_token	Y	String（256）	鉴权参数，访问接口的凭证
token_type	Y	String（256）	Access_token 类型，默认 Bearer
refresh_token	Y	String（256）	鉴权参数，用来刷新
expires_in	Y	String（256）	平台生成并返回 access_token 有效期，单位为秒
scope	Y	String（256）	申请权限范围，默认为 all
resp_code	Y	String（256）	请求执行结果。0 失败 1 成功
Resp_msg	Y	String（256）	执行结果返回的信息

请求示例:

Method:POST

Request:
https://<API_ADDRESS>/api/v1.0.0/refreshToken

Header:
    Authorization:Bearer xxxxxxxxxxxxxxxxxxx // 此处为 Auth 接口获取的 accessToken

Content-Type:application/json
Body:

```
{
  "appId":"xxxx",
  "secret":"xxxxxx",
  "refreshToken":"xxxxxxxxxxxxx"
}
```

响应示例:

Response:
Status Code:200 OK
Content-Type:application/json
Body:

```
{
    "datas": {
        "access_token": "xxxxxxxxxxxxxxxx",
        "token_type": "bearer",
        "refresh_token": "xxxxxxxxxxxxxx",
        "expires_in": 7199,
        "scope": "all"
    },
    "resp_code": 1,
    "resp_msg": "刷新成功"
}
```

错误消息:

resp_code	说明
0	1、 appId 或 secret 错误 2、 refreshToken 错误 3、 appid、secret 和 refreshToken 都不能为空

4.3 设备注册

接口功能:

设备注册是将设备从家之安设备管理平台分配给接入平台，通过返回值判断注册结果。

接口原型:

请求方法	POST
请求地址	https://<API_ADDRESS>/api/v1.0.0/registerDevice
请求协议	HPPTS

请求参数:

参数	必选	类型	位置	描述
device_qrcode	Y	String（256）	请求体 body	设备二维码（24 位编码）

响应参数:

Status Code:200 OK

请求示例:

Method:POST

Request:

https://<API_ADDRESS>/api/v1.0.0/registerDevice

Header:

appId:XXXXX

Authorization:Bearer xxxxxxxxxxxxxxxxxxx // 此处为 Auth 接口获取的 accessToken

Content-Type:application/json

Body:

```
{
  "device_qrcode": "xxxx"
}
```

响应示例:

Response:

Status Code:200 OK

Content-Type:application/json

Body:

```
{
    "resp_code": 1,
    "resp_msg": "设备已注册"
}
```

错误消息:

resp_code	说明
0	1、 设备不存在 2、 设备已使用 3、 appId 不存在 4、 请勿重复绑定
2000	4、Invalid access token: xxxxxxxxxxxxxxxxxxxx

4.4 查询设备详情

接口功能:

单个设备查看详情是已知设备二维码的情况下查看设备的明细。会返回待查设备的基本信息、设备当前状态等。

接口原型

请求方法	POST
请求地址	https://<API_ADDRESS>/api/v1.0.0/getDeviceInfo
请求协议	HTTPS

请求参数:

参数	必选	类型	位置	描述
device_qrcode	Y	String（256）	请求体 body	设备二维码

响应参数:

Status Code:200 OK

参数	必选	类型	描述
datas	Y	String(256)	数据集合
device_status	Y	String(256)	0 正常 1 故障（包括离线）2 报警
last_time	Y	String(256)	设备最后上线时间
device_qrcode	Y	String(256)	设备二维码
production_date	Y	String(256)	设备生产日期
info_upspace	Y	String(256)	上传间隔
space_unit	Y	String(256)	上传间隔单位
device_type	Y	String(256)	设备类型
model_name	Y	String(256)	设备型号全称
isp_type	Y	String(256)	设备数据上传平台类型 IOT 电信 ONENET 移动
imsi	Y	String(256)	IMSI
imei	Y	String(256)	IMEI
gprs_code	Y	String(256)	GPRS 号码
sim_sn	Y	String(256)	SN
device_addr	Y	String(256)	设备地址码
resp_code	Y	String(256)	请求执行结果。0 失败 1 成功
resp_msg	Y	String(256)	执行结果返回的信息

请求示例:

```
Method:POST
Request:
https://<API_ADDRESS>/api/v1.0.0/getDeviceInfo
Header:
    appId:XXXXX //即 appId
    Authorization:Bearer xxxxxxxxxxxxxxxxxxx // 此处为 Auth 接口获取的
accessToken
    Content-Type:application/json
Body:
{
  "device_qrcode":"xxxx" // 此处为设备二维码
}
```

响应示例:

```
Response:
Status Code:200 OK
Content-Type:application/json
Body:
{
    "datas": {
        "device_status": "0",
        "device_qrcode": "XXXXXXXXX",
        "production_date": "2019-10-04",
        "space_unit": "天",
        "device_type": "烟雾报警器",
        "isp_type": "ONENET",
        "imsi": "XXXXXXX",
        "model_name": "KD-122NB 移动 NB 烟雾报警器",
        "sim_sn": "XXXXXXXXX",
        "last_time": "2019-10-30 10:24:30",
        "imei": "XXXXXXXX",
        "info_upspace": "2",
        "gprs_code": "XXXXXXXXX",
        "device_addr ": "XXXXXXXXX"
    },
    "resp_code": 1,
    "resp_msg": "查询设备详情成功"
}
```

错误消息:

resp_code	说明
0	1、 设备不属于此用户或暂未注册 2、 设备不属于此用户或暂未注册 3、 appId 不存在
2000	4、Invalid access token: xxxxxxxxxxxxxxxxxxxx

4.5 发送指令

接口功能:

设备管理平台向设备发指令，通过返回值判断执行结果。

接口原型:

请求方法	POST
请求地址	https://<API_ADDRESS>/api/v1.0.0/sendCommand
请求协议	HPPTS

请求参数:

参数	必选	类型	位置	描述
device_qrcode	Y	String（256）	请求体 body	设备二维码（24 位编码）
downType	Y	String（256）	请求体 body	指令类型 ctrl ：控制指令 set ：设置指令 query ：查询指令
method	Y	String（256）	请求体 body	命令方法
paras	Y	String（256） JSON 格式	请求体 body	命令参数 {"sirentime":"1"}

设备能力 json 定义:

```
{
    "ctrl": { //指令类型
            "SIREN_ALARM": { //命令方法
                    "name": "响铃",//按钮名称
                    "wechat": "detail",//用途定义
                    "web": "dialog",//用途定义
                    "buttons": [{ //自定义按钮，非必有
                            "name_zh": "关闭",
                            "value": "0"
                    }, {"name_zh": "开启","value": "255"
                    }],
                    "paras": [{ //参数定义
                            "name": "时间", //参数名称
                            "value": "sirentime",// 参数属性，组装 paras 数组内容
                            "unit": "秒",//参数单位
                            "isSet": true,//是否需要输入参数数值
                            "max": "255",//参数上限
                            "min": "0",//参数下限
                            "remark": "0 代表关闭，255 时代表一直响"//参数注解
                            "default": "2"//isSet==false,直接获取组装数据
                    }]
            }
    }
}
```

注:设备支持的命令在设备上报数据‘device_ability’字段中体现，详见数据推送说明。

响应参数:

Status Code:200 OK

请求示例:

```
Method:POST
Request:
https://<API_ADDRESS>/api/v1.0.0/sendCommand
Header:
    appId:XXXXX
    Authoriation:Bearer xxxxxxxxxxxxxxxxxxx // 此处为 Auth 接口获取
的 accessToken
    Content-Type:application/json
Body:
{
  "device_qrcode":"xxxx", // 设备二维码
  "downType":"xxxxxx" // 指令类型
  "method":"xxxxxx" // 指令方法
  "paras":{device_cmd: "07"} // 指令参数
}
```

响应示例:

```
Response:
Status Code:200 OK
Content-Type:application/json
Body:
{
    "resp_code": 1,
    "resp_msg": "下发命令成功"
}
```

错误消息:

resp_code	说明
0	1、 设备不属于此用户或暂未注册 2、 appId 不存在 3、 下发命令失败 4、 设备不支持下行命令 5、 设备不支持这条下行命令
2000	6、 Invalid access token: xxxxxxxxxxxxxxxxxxxx

4.6 数据推送

接口功能:

家之安设备管理平台以 HTTPS POST 请求形式向接入平台注册地址推送数据，推送数据相关信息以 JSON 串的形式置于 HTTPS 请求中的 body 部分。

接入平台在接收数据时，会接收到数据的明文消息或者密文消息。明文消息中以“type”的数值区分不同的消息，而密文消息与明文消息的区别是“msg”部分经过了加密处理。下面将分别对明文消息、密文消息以及消息相关字段和密文的加密算法进行说明。

接收到的数据由 Toekn 令牌验证数据的有效性，推送示例参见:附录 II 推送数据。

注意事项:

数据推送使用的为 Token（令牌），在填写第三方平台注册信息后，由设备管理平台提供。

接口原型:

请求方法	POST
请求地址	创建应用平台时填写的 URL
请求协议	HTTPS

请求参数

明文

参数	必选	类型	描述
device_qrcode	Y	varchar（256）	设备二维码
msg	Y	varchar（256）	明文
nonce	Y	varchar（8）	八位随机码
msgSignature	Y	varchar（256）	MD5 签名

密文

密文消息中的“enc_msg”是由明文消息中的“msg”采用 AES 加密算法而来。

参数	必选	类型	描述
device_qrcode	Y	varchar（256）	设备二维码
enc_msg	Y	varchar（256）	密文
nonce	Y	varchar（8）	八位随机码
msgSignature	Y	varchar（256）	MD5 签名

明文消息示例:

1、设备上报数据:

```
{
  "protocol_ver": "KDNB01",  // 协议版本
  "device_qrcode": "XXXXX", // 设备二维码
  "data": {   // 设备上报数据
    "att": "10000000", // 设备属性
    "val": "00000000",// 状态值
    "csq": "16",// 信号强度
    "sub": "01",// 序列号总数
    "sta": "00000000",// 设备状态
    "ver": "6A64", // 版本号
    "mod": "00", // 布防撤防
    "num": "000D",// 数据流水号
    "nid": "F002622E",// 经过的网关地址码
    "ack": "00", // 是否需要回码
    "typ": "01",// 类型 code
    "sid": "00",//设备序列号
    "voltage": "0",// 电压
    "act": "report",// 动作
    "zon": "1000",// 工作区
    "sensor": "0",// 传感器故障
    "cmd": "00",// 指令
    "net": "00",// 网络状态
    "did": "F002622E",//联网状态
    "seq": "0026"//型号
  },
  "device_model": "KD-122NB-OneNET", //型号名称
  "device_status": "0", //设备状态 0 正常 1 故障（包括离线）2 报警
  "device_type": "烟雾报警器",// 类型名称
  "device_ability": { // 设备能力
    "silence": "01"//静音
  },
  "type": "1",// 信息类型  1 设备上报数据 2 离线数据 3 解绑设备数据
}
```

"device_ability":设备属性，是否允许静音等属性

″protocol_ver″:协议版本，不同的协议版本号规定了 data 字段内设备上报的数据格式，上述例子只是其中一种情况，会提供单位的协议版本号对应的数据格式定义。其余字段通用。

″seq″:4 位 ascii 设备型号序列码，服务器可以据此得到产品具体型号（关联图片，技术参数,状态信息定义等）

″att″字段定义(2 位 Hex string，对应十六进制数位定义):

bit7: 1->device support start hush
bit6: 1->device support start alarm
bit5: 1->device support start test
bit4: 1->device support stop hush
bit3: 1->device support stop alarm
bit2: 1->device support stop test
bit1: 1->device support boardcast
bit0: default

″sta″ 字段缺省定义(2 位 Hex string，对应十六进制数位定义):

b0:1->LowBattery, 0->Normal;
b1:1->AC PowerDown,0->Normal;
b2:1->Sensor Fault,0->Normal;
b3-b7:自定义

（不同产品型号具体位代表状态可能有所不同）

″zon″字段定义(火灾报警设备固定为“1000”HexString)

″mod″字段定义(火灾报警设备固定为“00”HexString)

″cmd″字段定义(2 位 Hex string，对应十六进制数):

0->normal	正常
1->normal alarm	报警
7->test alarm	测试
8->nop	空

2、设备离线通知第三方平台:

```
{
    ″device_qrcode″:″″, //设备二维码
    ″type″:″2″//推送数据格式类型：
              1 设备上报数据 2 离线数据 3 解绑设备数据
}
```

3、解绑设备通知第三方平台:

```
{
    ″device_qrcode″:″″, //设备二维码
    ″type″:″3″//推送数据格式类型：
              1 设备上报数据 2 离线数据 3 解绑设备数据
}
```

五、应用说明

5.1 设备二维码

每一个可以通过 NB、GPRS、WIFI 等设备都有唯一的二维码和设备序列码。二维码总长度 24 位，以‘KD’字母开头，最后 8 位也是设备序列码，其余 14 位由不特定字符串的 MD5 散列码构成，所有字母大写。

5.2 设备序列码

设备序列码唯一，位置一般位于二维码下方，以‘SN:’开头，后接一个字母和 7 位数字构成。

5.3 应用流程

获取家之安设备（KD-122LANB 烟雾报警器）；
扫码设备扫码二维码获取二维码信息；
接入方平台通过设备二维码信息到家之安设备平台获取设备详情；接入方平台到家之安设备平台注册设备；
注册后的设备上报数据通过家之安设备平台推送到接入方平台；注册后的设备接入方也可以通过家之安设备平台下发命令给设备。

5.4 应用场景

设备烟雾报警->设备上报数据到接入方平台->接入方平台获取报警信息->接入方平台下发静音命令（设备支持）->设备静音。

NB 烟雾报警器正常每 12 小时上报一次数据->接入方平台获取设备状态信息。

附录 I 推送数据

JAVA 开发代码

数据接收

```
@RestController
@Slf4j
@RequestMapping("/msg")
public class ThirtyPartInfoController {
        // token（Token 令牌）
        private static String token = "xxxxxxxx";
        // 加密密钥
        private static String aeskey = "xxxxxxx";

        // 对称加密
        public static final String KEY_ALGORITHM = "AES";
        public static final String CIPHER_ALGORITHM_CBC = "AES/CBC/PKCS5Padding";
        public static final String CIPHER_ALGORITHM_ECB = "AES/ECB/PKCS5Padding";

        @PostMapping("/receive")
        @ResponseBody
        public String receive(@RequestBody String body) throws UnsupportedEncodingException,
Exception {
                log.info("data recieve : body string ---- " + body);
                String result = null;
                // 明文模式
                ThirtyPartyDataUtil.BodyObj obj = ThirtyPartyDataUtil.resolveBody(body,
false);
                if (obj != null && obj.getMsg() != null) {// 明文
                        // 签名验证
                        boolean dataRight = ThirtyPartyDataUtil.checkSignature(obj, token);

                        if (dataRight) {
                                byte[] infoBytes =
Base64.getDecoder().decode(obj.getMsg().toString());
                                String msgString = new String(infoBytes, "utf-8");
                                log.info("data receive: content" + msgString);
                                return 200 + msgString;
                        } else {
                                log.error("签名验证失败");
                                return "签名验证失败";
                        }
                } else { // 加密模式
                        obj = ThirtyPartyDataUtil.resolveBody(body, true);

                        if (obj != null && StringUtils.isNotBlank(obj.getMsgSignature())) {
                                boolean dataRight = ThirtyPartyDataUtil.checkSignature(obj,
token);
                                if (dataRight) {
                                        // String msg = ThirtyPartyDataUtil.decrypt(obj,
aeskey);]
                                        byte[] msg =
decrypt(Base64.getDecoder().decode(obj.getMsg().toString()), aeskey.getBytes());
                                        log.info("解密数据:" + new String(msg,
StandardCharsets.UTF_8));
                                        result = new String(msg, StandardCharsets.UTF_8);
                                        try {
                                                JSONObject jsonStr =
JSONObject.parseObject(result);
                                                log.info("合法 json 格式");
```

```
                                        } catch (Exception e) {
                                                log.info("非法 json 格式");
                                        }
                                        log.info(result);
                                        return result;
                                } else {
                                        log.error("数据签名验证失败");
                                        return "error";
                                }
                        } else {
                                log.info("数据为空");
                                return "null data";
                        }
                }

        }

        /**
         * 验证本地 token 与服务器发送的 token 是否一致
         *
         * @param msg       验证消息
         * @param nonce     随机串
         * @param signature 签名
         * @return msg 数据
         * @throws UnsupportedEncodingException
         */
        @PostMapping("/check")
        public String check(@RequestParam(value = "msg") String msg, @RequestParam(value =
"nonce") String nonce,
                            @RequestParam(value = "signature") String signature) throws
UnsupportedEncodingException {
                log.info("url&token check: msg:{} nonce{} signature:{}", msg, nonce,
signature);
                if (ThirtyPartyDataUtil.checkToken(msg, nonce, signature, token)) {
                        return msg;
                } else {
                        return "error";
                }
        }


        public static byte[] encrypt(byte[] data, byte[] key) throws Exception {
                Cipher cipher = Cipher.getInstance(CIPHER_ALGORITHM_ECB);
                cipher.init(Cipher.ENCRYPT_MODE, new SecretKeySpec(key, KEY_ALGORITHM));
                return cipher.doFinal(data);
        }

        public static byte[] encrypt(byte[] data, byte[] key, byte[] iv) throws Exception {
        Cipher cipher = Cipher.getInstance(CIPHER_ALGORITHM_CBC);
        cipher.init(Cipher.ENCRYPT_MODE, new SecretKeySpec(key, KEY_ALGORITHM), new
IvParameterSpec(iv));
        return cipher.doFinal(data);
    }

        public static byte[] decrypt(byte[] data, byte[] key) throws Exception {
                Cipher cipher = Cipher.getInstance(CIPHER_ALGORITHM_ECB);
                cipher.init(Cipher.DECRYPT_MODE, new SecretKeySpec(key, KEY_ALGORITHM));
                return cipher.doFinal(data);
        }

        public static byte[] decrypt(byte[] data, byte[] key, byte[] iv) throws Exception {
                Cipher cipher = Cipher.getInstance(CIPHER_ALGORITHM_CBC);
                cipher.init(Cipher.DECRYPT_MODE, new SecretKeySpec(key, KEY_ALGORITHM), new
IvParameterSpec(iv));
                return cipher.doFinal(data);
        }
}
```

数据解析

```
/**
 * @author [jza]
 * @Date [2019年10月22日]
 * @Description: [家之安平台第三方平台数据推送工具类 ]
 */
@Slf4j
public class ThirtyPartyDataUtil {

        public static final String KEY_ALGORITHM = "AES";
        public static final String CIPHER_ALGORITHM_CBC = "AES/CBC/PKCS5Padding";
        public static final String CIPHER_ALGORITHM_ECB = "AES/ECB/PKCS5Padding";

        private static MessageDigest mdInst;

        static {
                try {
                        mdInst = MessageDigest.getInstance("MD5");
                        Security.addProvider(new BouncyCastleProvider());
                } catch (Exception e) {
                        e.printStackTrace();
                        log.error(e.getMessage());
                }
        }

        /**
         * 功能描述：验证本地token与服务器是否一致      *
         * @param msg       消息
         * @param nonce     8位随机串
         * @param signature 签名
         * @param token     实时token
         * @return token 校验成功true，失败false
         * @throws UnsupportedEncodingException
         */
        public static boolean checkToken(String msg, String nonce, String signature, String
token)
                        throws UnsupportedEncodingException {
                byte[] paramB = new byte[token.length() + 8 + msg.length()];
                // 复制token
                System.arraycopy(token.getBytes(), 0, paramB, 0, token.length());
                // 复制随机串
                System.arraycopy(nonce.getBytes(), 0, paramB, token.length(), 8);
                // 复制消息
                System.arraycopy(msg.getBytes(), 0, paramB, token.length() + 8,
msg.length());

                String sign = Base64.encode(mdInst.digest(paramB));
                log.info("url&token validation: result {}, detail receive:{} calculate:{}",
                                sign.equals(signature.replace(' ', '+')), signature, sign);

                return sign.equals(signature.replace(' ', '+'));
        }

        /**
         * 验证数据签名
         *
         * @param obj   消息对象
         * @param token 家之安平台token
         * @return
         */
        public static boolean checkSignature(BodyObj obj, String token) {
                // obj 为 token长度 + 8 位随机码 + 消息长度
                byte[] signature = new byte[token.length() + 8 +
obj.getMsg().toString().length()];
```

```
                System.arraycopy(token.getBytes(), 0, signature, 0, token.length());

                System.arraycopy(obj.getNonce().getBytes(), 0, signature, token.length(), 8);

                System.arraycopy(obj.getMsg().toString().getBytes(), 0, signature,
token.length() + 8,
                                obj.getMsg().toString().length());

                mdInst.update(signature);

                String calSignature = Base64.encode(mdInst.digest());

                log.info("check signature: result:{} receive sig:{},calculate sig:{}",
                                calSignature.equals(obj.getMsgSignature()),
obj.getMsgSignature(), calSignature);

                return calSignature.equals(obj.getMsgSignature());
        }

        /**
         * Data 解密
         *
         * @param data 加密内容
         * @param key  加密密钥
         * @return
         * @throws Exception
         */
        public static byte[] decrypt(byte[] data, byte[] key) throws Exception {
                Cipher cipher = Cipher.getInstance(CIPHER_ALGORITHM_ECB);
                cipher.init(Cipher.DECRYPT_MODE, new SecretKeySpec(key, KEY_ALGORITHM));
                return cipher.doFinal(data);
        }

        /**
         * MD5 加密
         * @param source
         * @param salt
         * @param toUpperCase
         * @return
         */
        public static String md5(byte[] signature) {
                mdInst.update(signature);
                String calSignature = Base64.encode(mdInst.digest());
                return calSignature;
        }


        /**
         * @param body      推送数据的 body 部分
         * @param encrypted 是否被加密
         * @return
         */
        public static BodyObj resolveBody(String body, boolean encrypted) {
                JSONObject jsonMsg;
                BodyObj obj = new BodyObj();
                try {
                        jsonMsg = new JSONObject(body);
                        obj.setNonce(jsonMsg.getString("nonce"));
                        obj.setMsgSignature(jsonMsg.getString("msgSignature"));
                        if (encrypted) {// 加密
                                if (!jsonMsg.has("enc_msg")) {
                                        return null;
                                }
                                obj.setMsg(jsonMsg.getString("enc_msg"));
                        } else {// 未加密
                                if (!jsonMsg.has("msg")) {
                                        return null;
```

```
                        }
                        obj.setMsg(jsonMsg.getString("msg"));
                    }
                } catch (JSONException e) {
                    log.error(e.getMessage());
                    e.printStackTrace();
                }
                return obj;
            }

            /**
             * @author [yeqh]
             * @Date [2019年10月26日]
             * @Description: [消息体 ]
             */
            @Data
            public static class BodyObj {

                // 消息内容
                private Object msg;

                // 8位随机码
                private String nonce;

                // 数据签名
                private String msgSignature;

                // 设备二维码
                private String device_qrcode;
            }
}
```