国网车联网平台与道闸系统平台技术对接方案

一、现状

目前对接道闸系统平台优惠的模式主要分两种，一种是充电时间免费方式，一种是充电免指定小时数。

二、道闸系统平台与国网车联网平台接口实现

总体思路：对于优惠的场站由国网车联网平台配置场站id(场站id=站点id)，在结束充电的时候把配置场站ID的充电记录推给道闸系统平台，道闸系统平台再执行设定的减免逻辑，实现充电停车减免。

三、实现方式

国网车联网平台建议通过http请求开放接口将加密充电订单数据传输到道闸系统平台中。

请求方式:POST

Content-Type:application/json

秘钥:由道闸系统平台定义

初始向量:由道闸系统平台定义

参数加解密方式:AES加解密(加解密工具类如下)

四、接口字段字义

接口名称：

服务提供方：道闸系统平台

服务调用方：国网车联网平台

服务调用机制：充电结束后

输入参数

名称	数据类型	说明	是否必传
data	String	加密后参数下边字段为加密后的属性字段	是
parkId	string	停车场 ID，预留三方停车系统定义(默认与电站 ID 一致) 数据定义：varchar(50)	是
orderNo	string	充电订单号（不允许为空）	是
plateNo	string	默认车牌号(用户设置)	是
startTime	string	充电开始时间 不允许为空 yyyy-MM-dd HH:mm:ss	是
endTime	string	充电结束时间 不允许为空 yyyy-MM-dd HH:mm:ss	是
charingTime	long	充电时间 毫秒	是
stationId	string	电站 ID（不允许为空）	是
stationNo	string	桩编码（不允许为空）	是
stationName	string	电站名称（不允许为空）	是
power	float	充电量（不允许为空） 单位：度，小数点后 2 位	否
elecMoney	float	电费（不允许为空） 单位：元	是
serviceMoney	float	服务费（不允许为空） 单位：元	是

carNums	String[]	车牌号集合 carNums 是当前用户在 e 充电 App 上 我的爱车里面绑定的所有车辆 包含 plateNo 对应的车	是
thirdTradeNo	String	互联互通订单流水	否

返回值：

名称	数据类型	说明	是否必须
parkId	string	车场 ID/站点 id	否
orderNo	string	充电订单号	否
plateNo	string	车牌号(全车牌) plateNo 是当前用户在 e 充电 App 上我的爱车设置的默认车辆 没有设置就随机配置 如果用户有多辆车 plateNo 有可能 就不是当前用户充电的车牌	否
discount_fee	String	减免金额(code=0 时必传)	是
failue_reason	string	失败原因（code!=0 时必传）	是
code	Integer	0 成功 其他值失败 （不允许为空）	是

五、请求示例

未加密前数据:

{"carNums":["京
AA0101"],"charingTime":247000,"elecMoney":0.16,"endTime":"2021-01-21
17:56:00","orEndTodGunoutTime":100,"orderNo":"1140290000019151002101215157901
7","parkId":"3001417","plateNo":"京
AA0101","power":0.16,"serviceMoney":0.48,"stationNo":"1140290000019151","start
Time":"2021-01-21 17:51:53","stationId":"00c0353e-0db4-4d87-a6a6-
3af45f834bad","stationName":"尝试"}
秘钥:

秘钥仅测试环境使用，正式环境待联调结束后由车联网提供

skey:YDwwpoC6syIwqT8m
isvkey:A68DwQdXvN6AMpxV

请求参数(将明文加密)

{

data:j1Ch5d99+OBX1CgrKdVtm1CX5O+hfNgrNMBoB1H/18XOmY4YsGf4ZbZ2bpCIPbtqx7nx1kkaO
EmnV+sUbtaEMWdhgy7FOakBVQUOCgrvcnyH6PO5VbF5roOJcCkIgD2IOHJYxZYtnk7ODs8+1NucumL
Ot7pAOavhqRgqFtTcizrmVPJ1SssEcRisMxPkYRgbT9AZnfmL+hNo9rYgoCrxwvZzXgHOmrVDv4XAD
fyygM5dvp4HqWYo8wvOzJJ1ipkKfc1u8FOOKVtjy8IRX4d9WDfC6OBK4onLWd1K47Rfbsi2L+N6kPB
ZvBeyipAL9rjG9zvIBSJbTGlDwN1knxdKGp+qRs3Hz8OSHbPpT4OYELFVdJikymfkgiy6VMXGOnHZT
ErVTrEEFgV1xOBdbAcTA9T8KvItyEUrM4Zu5vM3OUwKvehxZ1iufOlpNsLW7AmehMVQeKVVkkbHnki
za6WcOUsLK9pdVf6gH+cadE1sPqHiD/TEQH8QJonxFBrjj4IRy8Tj/OhNsYtsQvqcrKKOlQ==

}

返回值（*要加密*下方是未加密的格式）

加密前数据：


```
{
    "code": 0,
    "failue_reason": "减免成功",
    "discount_fee": "8"
}
```


返回内容（将明文加密）：

{

    "data": "HZBcnmqCIfhuXjdy4gZUSpO7OxV3stBs1pIey+LpyROowueIqDOiouQ=="

}

车联网订单推送道闸流程 e 充电 app 绑车->充电-->充电结束->推送订单数据给三方道闸（由三方道闸系统根据订单数据处理减免逻辑）

注：车联网推送订单数据前提 用户充电之前 要确认绑车 不绑车的订单 车联网不推送 充电后绑车或不绑车 订单数据车联网不推送

六、加解密工具类: 加密 encrypt2 解密 decrypt2

```
import org.apache.commons.codec.binary.Base64;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import sun.misc.BASE64Decoder;

import sun.misc.BASE64Encoder;

import javax.crypto.Cipher;

import javax.crypto.spec.IvParameterSpec;

import javax.crypto.spec.SecretKeySpec;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;


/**

 * Created by Administrator on 2017/10/13 0013.

 */

public class AESUtil {


    private static final Logger LOGGER =
LoggerFactory.getLogger(AESUtil.class);

    /**
```

```
     * 密钥

     */

    private static final String SKEY = "abcdefgabcdefg13";



    // 加密  sSrc:加密字符串     sKey:密钥     ivStr:初始向量

    public static String encrypt2(String sSrc, String sKey, String ivStr)
throws Exception {

        if (sKey == null) {

            System.out.println("Key 为空 null");

            return null;

        }

        // 判断 Key 是否为 16 位

        if (sKey.length() != 16) {

            System.out.println(String.format("Key 长度不是 16 位,sKey:%s
length :%d", sKey, sKey.length()));

            return null;

        }

        byte[] raw = sKey.getBytes("UTF-8");

        SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");

        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");//"算法/模
式/补码方式"

        IvParameterSpec iv = new IvParameterSpec(ivStr.getBytes("UTF-8"));//使
用 CBC 模式，需要一个向量 iv，可增加加密算法的强度 1234567890123456

        cipher.init(Cipher.ENCRYPT_MODE, skeySpec, iv);
```

```
        // sSrc= escapeChar(sSrc);
        byte[] encrypted = cipher.doFinal(sSrc.getBytes("UTF-8"));

        String str = new BASE64Encoder().encode(encrypted);
        str = str.replaceAll("\r\n", "");//window
        str = str.replaceAll("\n", "");//linux

        return str;//new BASE64Encoder().encode(encrypted);//此处使用 BASE64 做转码功能，同时能起到 2 次加密的作用。
    }

    // 解密  sSrc:解密密文  sKey:密钥    ivStr:初始向量
    public static String decrypt2(String sSrc, String sKey, String ivStr) throws Exception {
        try {
            // 判断 Key 是否正确
            if (sKey == null) {
                System.out.println("Key 为空 null");
                return null;
            }
            // 判断 Key 是否为 16 位
            if (sKey.length() != 16) {
```

```
                System.out.println(String.format("Key 长度不是 16 位,sKey:%s
length :%d", sKey, sKey.length()));

                return null;

            }

            byte[] raw = sKey.getBytes("UTF-8");

            SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");

            Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");

            IvParameterSpec iv = new IvParameterSpec(ivStr.getBytes("UTF-8"));

            cipher.init(Cipher.DECRYPT_MODE, skeySpec, iv);

            byte[] encrypted1 = new BASE64Decoder().decodeBuffer(sSrc);//先用
base64 解密

            try {

                byte[] original = cipher.doFinal(encrypted1);

                String originalString = new String(original, "UTF-8");

                // originalString= unEscapeChar(originalString);

                return originalString;

            } catch (Exception e) {

                e.printStackTrace();

                System.out.println(e.toString());

                return null;

            }

        } catch (Exception ex) {

            System.out.println(ex.toString());
```

```
        return null;
    }
}


public static String getHmacMd5Str(String key, String data) {
    String result = "";
    try {
        byte[] keyByte = key.getBytes("UTF-8");
        byte[] dataByte = data.getBytes("UTF-8");
        byte[] hmacMd5Byte = getHmacMd5Bytes(keyByte, dataByte);
        StringBuffer md5StrBuff = new StringBuffer();
        for (int i = 0; i < hmacMd5Byte.length; i++) {
            if (Integer.toHexString(0xFF & hmacMd5Byte[i]).length() == 1)
                md5StrBuff.append("0").append(
                        Integer.toHexString(0xFF & hmacMd5Byte[i]));
            else
                md5StrBuff.append(Integer
                        .toHexString(0xFF & hmacMd5Byte[i]));
        }
        result = md5StrBuff.toString().toUpperCase();
```

```
        } catch (Exception e) {
            // logger.error("error getHmacMd5Str()",e);
            e.printStackTrace();
        }
        return result;

    }

    /**
     * 将待加密数据 data，通过密钥 key，使用 hmac-md5 算法进行加密，然后返回加密结果。 参照 rfc2104 HMAC 算法介绍实现。
     * @param key
     *            密钥
     * @param data
     *            待加密数据
     * @return 加密结果
     * @throws NoSuchAlgorithmException
     */
```

```
    public static byte[] getHmacMd5Bytes(byte[] key, byte[] data)
            throws NoSuchAlgorithmException {
        /*
         * HmacMd5 calculation formula: H(K XOR opad, H(K XOR ipad, text))
         * HmacMd5 计算公式：H(K XOR opad, H(K XOR ipad, text))
         * H 代表 hash 算法，本类中使用 MD5 算法，K 代表密钥，text 代表要加密的数据 ipad 为 0x36，opad 为 0x5C。
         */
        int length = 64;
        byte[] ipad = new byte[length];
        byte[] opad = new byte[length];
        for (int i = 0; i < 64; i++) {
            ipad[i] = 0x36;
            opad[i] = 0x5C;
        }
        byte[] actualKey = key; // Actual key.
        byte[] keyArr = new byte[length]; // Key bytes of 64 bytes length
        /*
         * If key's length is longer than 64,then use hash to digest it and use
         * the result as actual key. 如果密钥长度，大于 64 字节，就使用哈希算法，计算其摘要，作为真正的密钥。
         */
```

```
if (key.length > length) {
    actualKey = md5(key);
}
for (int i = 0; i < actualKey.length; i++) {
    keyArr[i] = actualKey[i];
}
/*
 * append zeros to K 如果密钥长度不足64字节，就使用0x00补齐到64字节。
 */
if (actualKey.length < length) {
    for (int i = actualKey.length; i < keyArr.length; i++)
        keyArr[i] = 0x00;
}

/*
 * calc K XOR ipad 使用密钥和ipad进行异或运算。
 */
byte[] kIpadXorResult = new byte[length];
for (int i = 0; i < length; i++) {
    kIpadXorResult[i] = (byte) (keyArr[i] ^ ipad[i]);
}
```

```
        /*
         * append "text" to the end of "K XOR ipad" 将待加密数据追加到K XOR ipad计算结果后面。
         */
        byte[] firstAppendResult = new byte[kIpadXorResult.length + data.length];
        for (int i = 0; i < kIpadXorResult.length; i++) {
            firstAppendResult[i] = kIpadXorResult[i];
        }
        for (int i = 0; i < data.length; i++) {
            firstAppendResult[i + keyArr.length] = data[i];
        }

        /*
         * calc H(K XOR ipad, text) 使用哈希算法计算上面结果的摘要。
         */
        byte[] firstHashResult = md5(firstAppendResult);

        /*
         * calc K XOR opad 使用密钥和opad进行异或运算。
         */
        byte[] kOpadXorResult = new byte[length];
```

```
for (int i = 0; i < length; i++) {
    kOpadXorResult[i] = (byte) (keyArr[i] ^ opad[i]);
}

/*
 * append "H(K XOR ipad, text)" to the end of "K XOR opad" 将H(K XOR
 * ipad, text)结果追加到K XOR opad结果后面
 */
byte[] secondAppendResult = new byte[kOpadXorResult.length
        + firstHashResult.length];
for (int i = 0; i < kOpadXorResult.length; i++) {
    secondAppendResult[i] = kOpadXorResult[i];
}
for (int i = 0; i < firstHashResult.length; i++) {
    secondAppendResult[i + keyArr.length] = firstHashResult[i];
}

/*
 * H(K XOR opad, H(K XOR ipad, text)) 对上面的数据进行哈希运算。
 */
byte[] hmacMd5Bytes = md5(secondAppendResult);
return hmacMd5Bytes;
```

```
    }

    private static byte[] md5(byte[] str) throws NoSuchAlgorithmException {
        MessageDigest md = MessageDigest.getInstance("MD5");
        md.update(str);
        return md.digest();
    }

}
```