浙江省健康数据高铁安诊儿专线

接口规范

第 3 部分：服务接口

（试行）

目 录

前　　言 4
1 范围 5
2 接口清单 5
3 对接模式 5
4 传输规范 5
5 接口明细 5
5.1 在线取号 5
5.1.1 服务说明 5
5.1.1.1 服务概述 5
5.1.1.2 支付模式 6
5.1.1.3 业务流程 6
5.1.1.4 用户动线 8
5.1.2 请求头 10
5.1.3 数据加密加签处理 10
5.1.3.1 签名&加密流程 11
5.1.3.2 请求体解密示例 12
5.1.3.3 响应体加密示例 12
5.1.3.4 Java 代码示例 12
5.1.3.5 maven 依赖 22
5.1.4 接口定义 22
【10000】患者取号列表 22
【20000】患者取号 25
【30000】自费支付成功通知 26
【40000】查询号源状态 27
5.2 排队叫号 28
5.2.1 服务说明 28
5.2.1.1 服务概述 28
5.2.1.3 业务流程 29
5.2.1.4 用户动线 29
5.2.2 请求头 32
5.2.3 数据加密加签处理 33
5.2.3.1 签名&加密流程 33
5.2.3.2 请求体解密示例 34
5.2.3.3 响应体加密示例 34
5.2.3.4 Java 代码示例 34
5.2.3.5 maven 依赖 44
5.2.4 接口定义 45
【10000】 获取医院排队叫号信息 45
5.3 费用清单 52
5.3.1 服务说明 52
5.3.1.1 服务概述 52
5.3.1.2 接入模式 52
5.2.1.3 用户动线 52
5.3.2 接口定义 53
5.3.3 请求示例 72

5.3.3.1 请求业务参数明文 72
5.3.3.2 请求业务参数密文 SM4 解密 72
5.3.3.3 实际请求参数 72
5.3.4 响应示例 73
5.3.4.1 响应业务参数明文 73
5.3.4.2 响应业务参数密文 SM4 解密 75
5.3.4.3 实际响应参数 75
5.3.4.4 注意事项 76
5.4 检查预约 76
5.5 在线缴费 76
5.6 报告查询 76

前 言

《浙江省健康数据高铁安诊儿专线接口规范》分为以下部分：
——第 1 部分：总则；
——第 2 部分：消息接口；
——第 3 部分：服务接口；

本部分为浙江省健康数据高铁安诊儿专线接口规范的第 3 部分。
本部分起草单位：浙江省卫生健康信息中心。

1 范围

本部分规定了浙江省健康数据高铁安诊儿专线服务接口的一系列传输方式、字段参数、请求响应示例。
本部分适用于各级医疗机构信息系统与安诊儿服务平台之间的业务服务接口交互。

2 接口清单

表 1 服务接口分类

服务类别	交易触发点	服务提供方	服务调用方
在线取号	业务交易，实时	医疗卫生机构	浙江省在线取号平台
排队叫号	业务交易，实时	医疗卫生机构	浙江省排队叫号平台
预约挂号	业务交易，实时	医疗卫生机构	浙江省预约挂号平台
费用查询	业务交易，实时	医疗卫生机构	安诊儿服务平台
检查预约	业务交易，实时	医疗卫生机构	安诊儿服务平台
在线缴费	业务交易，实时	医疗卫生机构	安诊儿服务平台
报告查询	业务交易，实时	医疗卫生机构	安诊儿服务平台

3 对接模式

接入模式类型分为：**医疗卫生机构直接接入、医疗卫生机构通过地市平台接入**两类
医疗卫生机构直接接入：医疗卫生机构直接对接业务接口。
医疗卫生机构通过地市平台接入：省平台对接地市平台，接口与单院接入模式基本相同，只增加医疗机构编码字段，用于地市平台区分下属机构。

4 传输规范

1 、请求方式：https

2 、数据格式：统一采用json 数据格式

3 、字符集编码：统一采用utf-8

5 接口明细

5.1 在线取号

5.1.1 服务说明

5.1.1.1 服务概述

安诊儿服务平台在线取号服务能力复用**浙江省在线取号平台**。源端医疗卫生机构通过接入浙江省在线取号平台，完成在线取号服务能力的接入。

浙江省在线取号平台是聚焦卫生领域“最多跑一次”要求，优化就医流程，减少患者线下就医排队时间的健康便民服务平台，通过统一获取各级医疗机构预约挂号信息，实现患者在线取号，直接诊间就医。

5.1.1.2 支付模式

以挂号费支付方式分为：**先取号再支付模式、先支付再取号**模式两类：

先取号再支付模式：用户在线取号时挂号费医院挂帐托收，线下就诊后统一院内结算，此种模式所有的费用与省平台无关，相应逃费风险由医院自行承担，省平台只做取号动作。

先支付再取号模式：在线支付模式支持自费支付、医保支付两类：

-自费支付：用户在线取号时挂号费在线支付， 完成支付后，省平台才会向医院 his 做取号动作。开通自费结算模式的医院需先完成与省电子健康卡平台的对接，支付交易走电子健康卡相应交易接口，平台提供统一交易流水账单，各接入医院自行对账。

特别说明：

1、自费结算模式，在取号列表中，需要返回那些号源是要在线结算结算的，如果该号源需要在线结算，则用户点击取号的时候，省平台将会给用户创建订单，并让用户去支付。

2、自费结算模式，如果支付失败，则认为取号失败，用户可以再次发起取号

3、自费结算模式，如果支付成功，则省平台会调用【30000】接口通知医院，医院需要进行院内取号动作，如果院内取号动作失败，省平台将直接发起退款操作。

4、居民端如果支付成功之后，需要获取取号结果，如果这个时候医院并未明确返回院内取号结果，就会调用【40000】接口轮询取号结果。

-医保支付：基于安诊儿小程序与开通了医保移动支付的医疗机构绑定支付插件授权，通过支付宝医保移动支付插件，通过医保移动支付将挂号费收入医院账户。

特别说明：

1、机构需首先开通医保移动支付，获得国家医保局下发正式密钥；

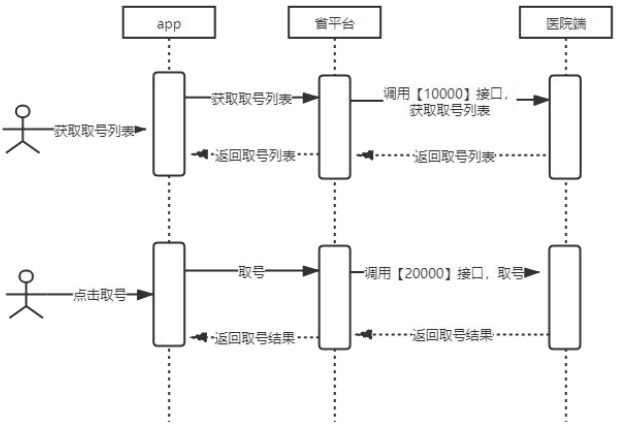
2、需完成安诊儿小程序医保插件与医院绑定，支付宝侧支持处理；

3、居民端如果支付成功之后，需要获取取号结果，会调用【40000】接口轮询取号结果。

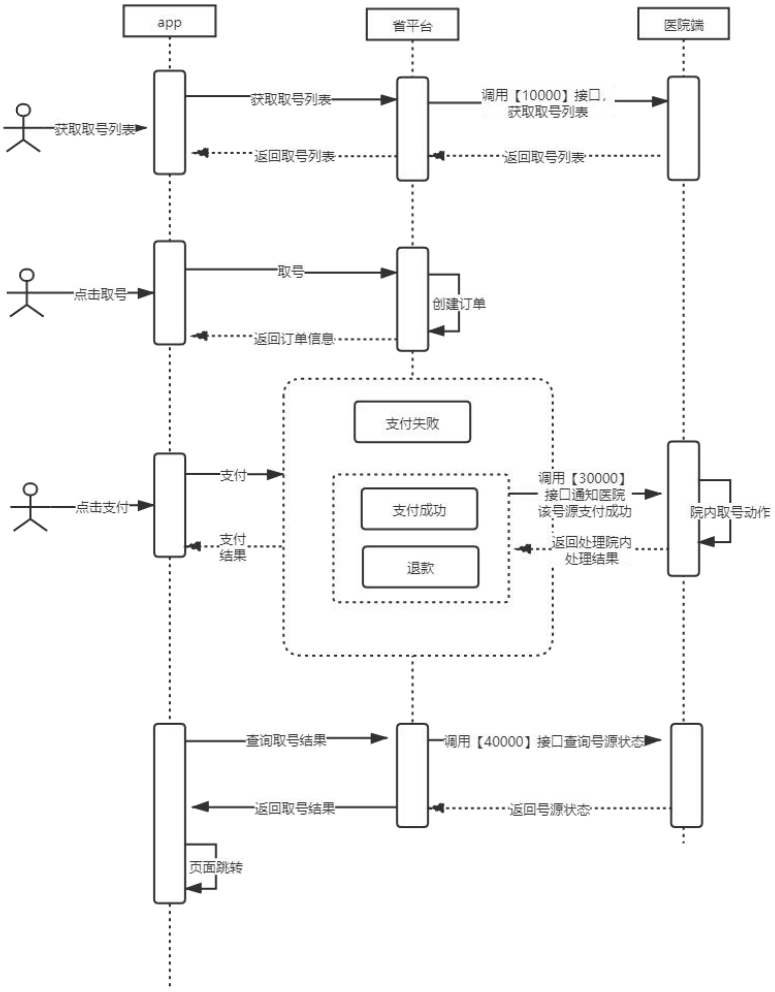
5.1.1.3 业务流程

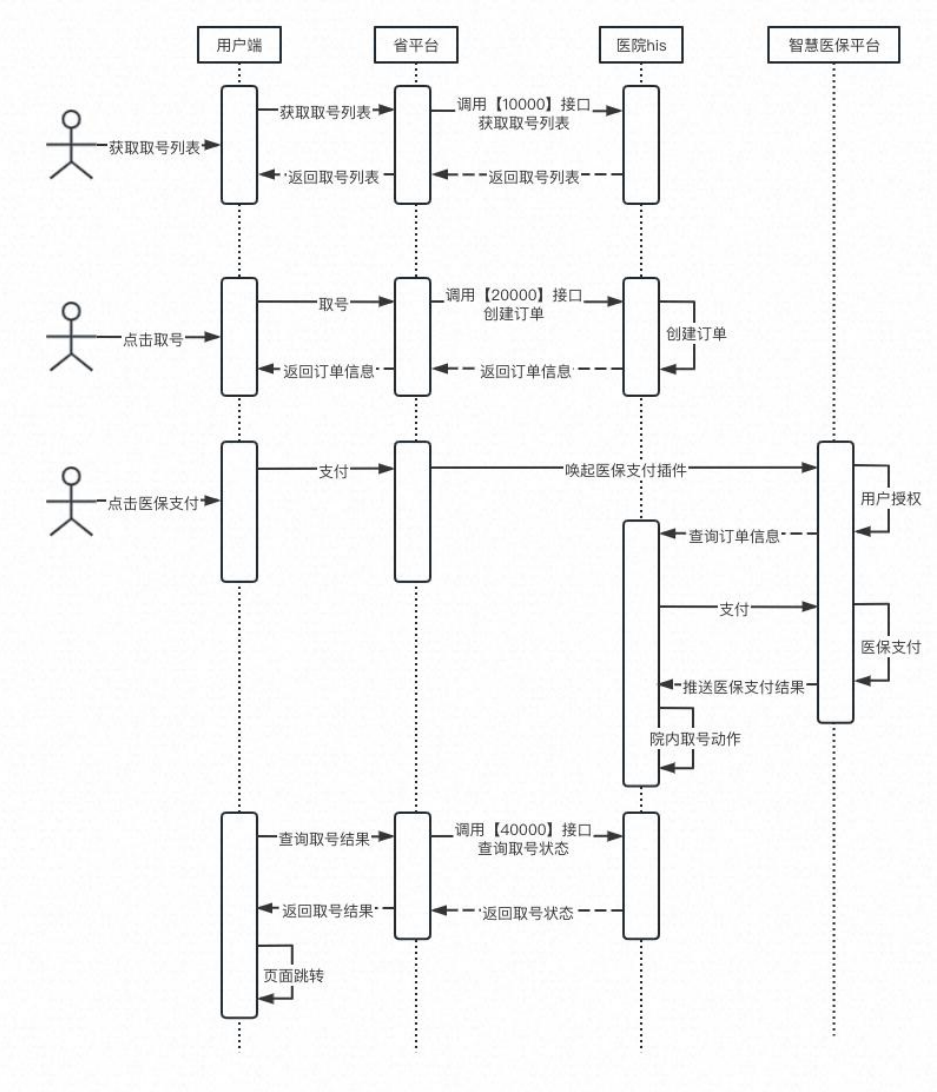
表 2 取号流程时序图

院内结算


app
省平台
医院端
获取取号列表
获取取号列表
调用【10000】接口，获取取号列表
返回取号列表
返回取号列表
点击取号
取号
调用【20000】接口，取号
返回取号结果
返回取号结果

在线结算


app
省平台
医院端
获取取号列表
获取取号列表
调用【10000】接口，获取取号列表
返回取号列表
返回取号列表
点击取号
取号
创建订单
返回订单信息
支付失败
点击支付
支付
支付结果
支付成功
退款
调用【30000】接口通知医院该号源支付成功
返回处理院内处理结果
院内取号动作
查询取号结果
调用【40000】接口查询号源状态
返回取号结果
返回号源状态
页面跳转



5.1.1.4 用户动线

表 3 医保支付在线取号用户动线图

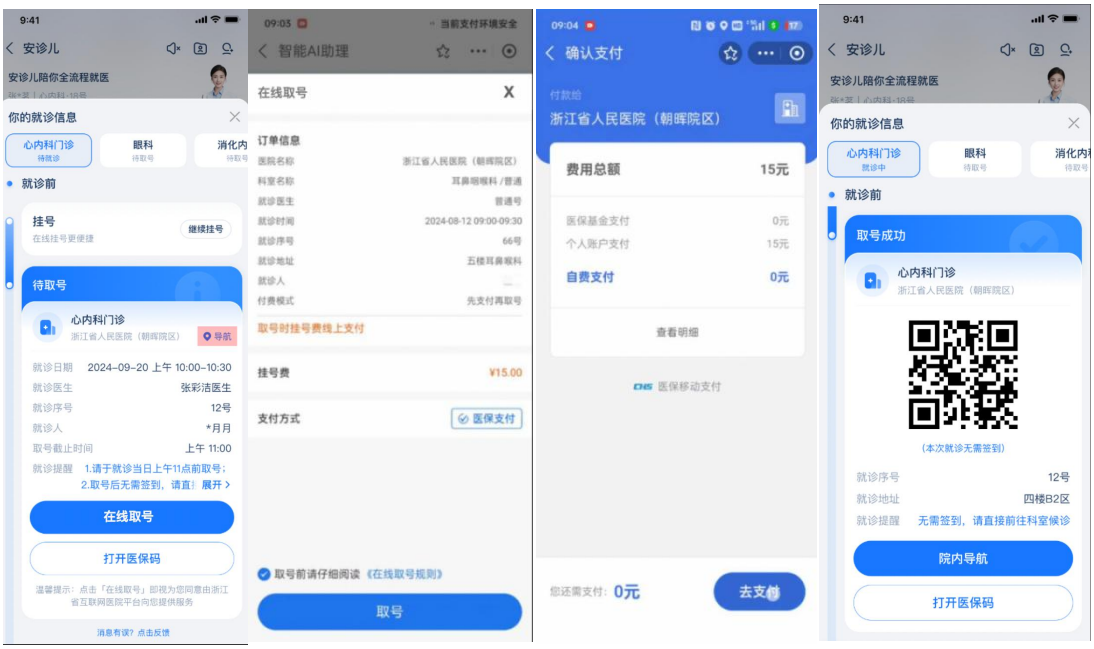


表 4 自费支付在线取号用户动线图

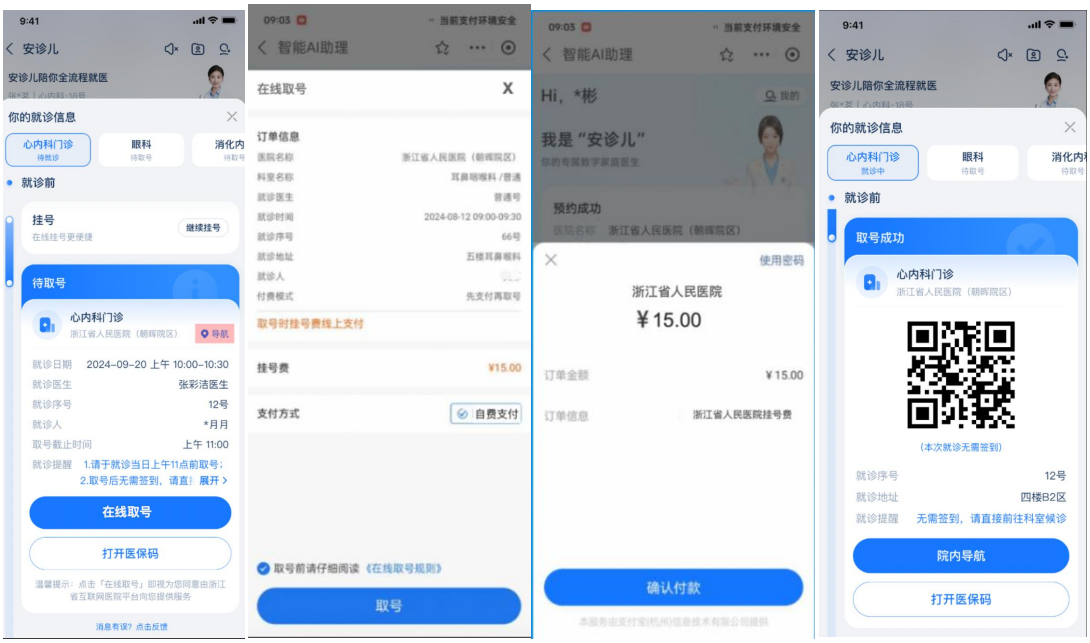


表 5 自费支付在线取号用户动线图

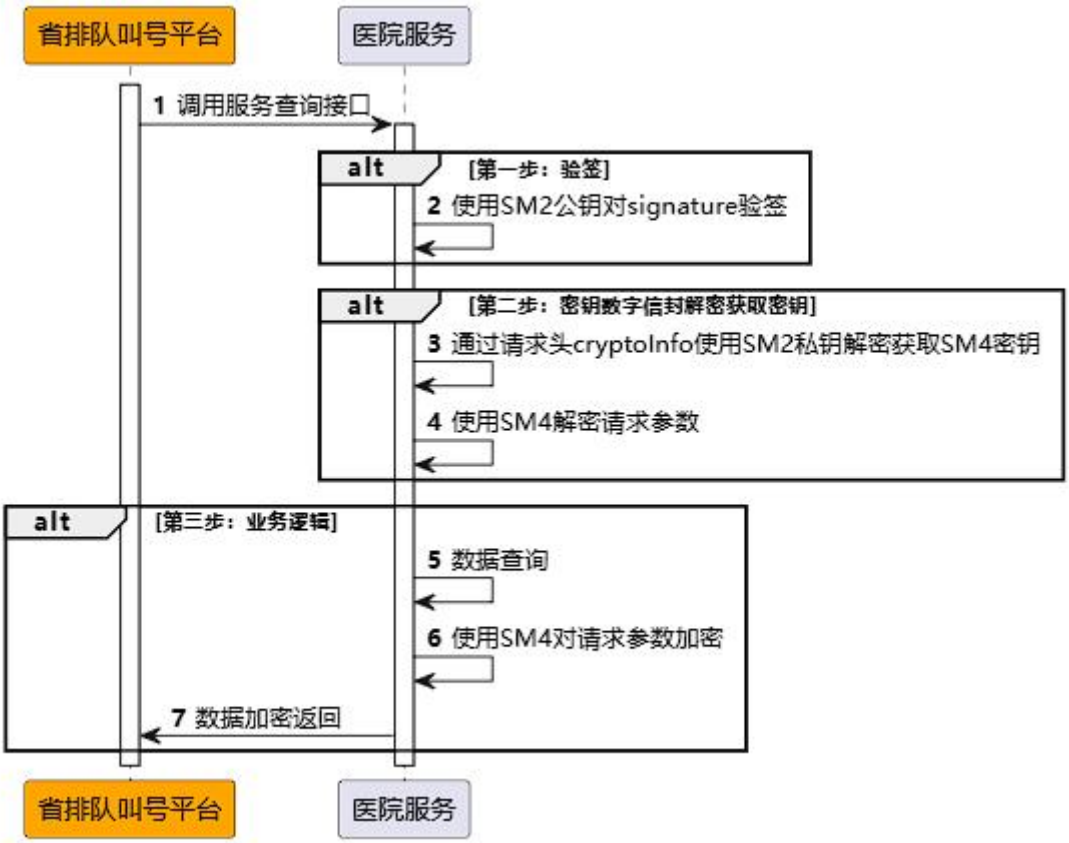
5.1.2 请求头

字段名	是否必填	字段类型	字段说明
med_org_code	是	String	医疗机构编码
med_hos_code	是	String	院区编码
request_id	是	String	请求 id
timestamp	是	String	请求时间戳
signature	是	String	签名信息
cryptoInfo	是	String	密钥数字信封（SM2 加密后的 SM4 密钥）

说明：可通过头部的参数来区分不同医疗机构和院区

5.1.3 数据加密加签处理

表 6 加解密流程时序图



5.1.3.1 签名&加密流程

5.1.3.1.1 签名规则

为了请求过程的安全性，业务各方之间的请求需要实现加签，以保证数据的安全性。请求方实现对请求参数数据的加签，服务方根据 Header 请求参数 signature，对签名进行验签，签名不合法的请求将会被拒绝，同时对 timestamp 时间戳进行校验，超出 10 秒范围内的请求将会被拒绝。报文的签名及验证使用 SM2 算法，具体处理机制要求如下:

1）对 Header 所有非空业务参数(signature 不参与签名)按照 key 的 ascii 顺序进行升序排序，然后拼接成一串待加签字符串，格式为：key=value&key=value。

签名示例：

cryptoInfo=0402b88dc4be88f212b92fa1dc092574d35670266d827c1c8220f6b6d382230e467407535510ae2ee77b0873b6e5839b2278e9a4048b1a5dec32e4c4feb897f46d2faf770595619f4b76476a4ad911679fcc5ed811e57ad98cbc502ba8d524c5a18fee6622e5e033e8412605052737e9bb4278eac2eee5ecefe463df6aa5ebf1828&med_hos_code=med_hos_code_001&med_org_code=med_org_code_001&request_id=406bf230-cdda-41b7-936f-99462e58797f&timestamp=1742113790749

2）生成签名：对拼接的字符串使用 SM2 算法通过私钥生成加签字符串 signature

5.1.3.1.2 签名示例

SM2 公钥	04b674b6edfd08f754410b21cc3c8b522f318a1f1b27403bc63a290b7e1790d03d967d06c46e69357793967ff42a34d77ed5d7bb40d31b7f5ab0b0840017cff114
SM2 私钥	a99d6978656355b570a19cc707b2ea68ed033c3fa436df9c0a15e84a06069c7b
SM4 密钥	726be1646879423e26eb7977c4d80c48

密钥数字信封 cryptoInfo（使用 SM2 私钥加密后的 SM4 密钥）	04eafaa325aea8f19463820c1e35553433bc1240a597bf085f908b73d39341f04cfcf95c6662a10c5b55188c326c9d04ec503c52ce28b76eb24f6d097b35337e6c6bda04a5ab4e5612fcc591fc207cde14fa648767ac7185e07303b7c76043b74ae62f4ec61ce255c35a75b3b5121a5037b55cc8f1f6f546d701cae56e34361008
加签前参数内容	cryptoInfo=0402b88dc4be88f212b92fa1dc092574d35670266d827c1c8220f6b6d382230e46740753510ae2ee77b0873b6e5839b2278e9a4048b1a5dec32e4c4feb897f46d2faf770595619f4b76476a4ad911679fcc5ed811e57ad98cbc502ba8d524c5a18fee6622e5e033e8412605052737e9bb4278eac2eee5ecefe463df6aa5ebf1828&med_hos_code=med_hos_code_001&med_org_code=med_org_code_001&request_id=406bf230-cdda-41b7-936f-99462e58797f&timestamp=1742113790749
加签后字符串(签名)	3045022100941e831d2a86ee1eac96d71a8ab13d5b4a44fdd27f324e4cf8e4a99bd27706150220069fd9a529a1aab78fddea63901e0cec0b1988849a2179cc5708d659d8e97c02

5.1.3.2 请求体解密示例

数字信封 SM4 密钥	726be1646879423e26eb7977c4d80c48
请求体 Body（SM4 加密密文）	41cacc5f8ea74405a36f445d5df4bb837993d52bcb65ae22b1afe591e67698f4380e8bf135ebb5a82e3d2723e96cd328d4e577701dbc748e060fee87cf5e9decfd774a960980c45646cb71fe9a4e42e67d4b7ea7ec083cd6c46ce8e92d722d54272abc6f0629031b20dcd9d8762aa200
业务请求参数明文	{"order_status":"3","name":"测试人员","idcard_value":"330100********0000","idcard_type":"01"}
最终请求体 Body	{"order_status":"3","name":"测试人员","idcard_value":"330100********0000","idcard_type":"01"}

5.1.3.3 响应体加密示例

数字信封 SM4 密钥	726be1646879423e26eb7977c4d80c48
Body 参数明文	{"response_code":"1","response_data":"...省略","response_msg":"success"}
业务参数 SM4 加密密文	31b7be894984b26f3acc23f3f2ad162a6131bb143013b38b59a0937e43d4287659c4b9ebd0998194dbb881cf81d0abfc3206b907af244fe0576da8ed746d8c7d3f693641504c13ec5bcd415d9932e0dd
最终响应体 Body	04eafaa325aea8f19463820c1e35553433bc1240a597bf085f908b73d39341f04cfcf95c6662a10c5b55188c326c9d04ec503c52ce28b76eb24f6d097b35337e6c6bda04a5ab4e5612fcc591fc207cde14fa648767ac7185e07303b7c76043b74ae62f4ec61ce255c35a75b3b5121a5037b55cc8f1f6f546d701cae56e34361008

5.1.3.4 Java 代码示例

```
import cn.hutool.core.util.HexUtil;

import cn.hutool.core.util.StrUtil;

import cn.hutool.crypto.SmUtil;

import cn.hutool.crypto.asymmetric.KeyType;

import cn.hutool.crypto.asymmetric.SM2;

import com.alibaba.fastjson.JSON;
```

```
import com.alibaba.fastjson.JSONObject;

import com.example.demo.utils.HttpUtils;

import org.apache.commons.collections4.MapUtils;

import org.junit.jupiter.api.Test;



import java.util.*;



/**

 * 在线取号请求流程-代码示例

 */

public class PickupRequestDemo {









    // sm4 秘钥

    static String SM4_KEY = "726be1646879423e26eb7977c4d80c48";

    // sm2 公钥

    static String SM2_PUBLIC_KEY =
"04b674b6edfd08f754410b21cc3c8b522f318a1f1b27403bc63a290b7e1790d03d967d06c46e69357793967ff42a34d77ed5d7bb40d31b7f5ab0b
0840017cff114";

    // sm2 私钥

    static String SM2_PRIVATE_KEY = "a99d6978656355b570a19cc707b2ea68ed033c3fa436df9c0a15e84a06069c7b";







    /**

     * 查询 his 患者取号列表信息
```

```
 */
@Test
public void queryPickupList() {
    // SM2 公钥加密 sm4 秘钥字符串
    String sm2EncryptSm4Key = sm2Encrypt(SM4_KEY, SM2_PUBLIC_KEY);
    // Header 参数
    HashMap<String, String> header = new HashMap<>();
    header.put("request_id", UUID.randomUUID().toString());
    header.put("med_org_code", "med_org_code_001");
    header.put("med_hos_code", "med_hos_code_001");
    header.put("cryptoInfo", sm2EncryptSm4Key);
    header.put("timestamp", String.valueOf(System.currentTimeMillis()));

    String preSignature = buildPreSignature(header);
    System.out.println("预签名字符串：" + preSignature);
    String signature = sm2Sign(SM2_PRIVATE_KEY, preSignature);
    header.put("signature", signature);
    System.out.println("signature：" + signature);

    //  请求体参数
    JSONObject requestBody = new JSONObject();
    requestBody.put("name", "测试人员");
    requestBody.put("idcard_type", "01");
    requestBody.put("idcard_value", "330100********0000");
    requestBody.put("order_status", "3");
```

```
        // 请求体参数 Json
        String requestBodyJson = requestBody.toJSONString();
        System.out.println(requestBodyJson);
        // 请求体参数 Json 加密
        String requestBodyJsonEncrypt = sm4Encrypt(requestBodyJson, SM4_KEY);
        System.out.println(requestBodyJsonEncrypt);

        // 请求 his 接口
        String result = HttpUtils.post("请求 his 患者取号列表接口", "https://ip:port/get/pickup/list", requestBodyJsonEncrypt, header);

        // 解密 ...
        String responseBody = sm4Decrypt(result, SM4_KEY);
        // 解析
        JSONObject jsonObject = JSON.parseObject(responseBody);
        // 处理业务 ...省略
    }

    /**
     * 获取医院患者取号列表信息查询接口-his 处理代码示例
     */
    public String getPickupList(HashMap<String, String> header, String requestBody) {
        String requestId = header.get("request_id");
        String medOrgCode = header.get("med_org_code");
        String medHosCode = header.get("med_hos_code");
        String cryptoInfo = header.get("cryptoInfo");
```

```
String timestamp = header.get("timestamp");

String signature = header.get("signature");



// 校验签名处理

HashMap<String, String> checkSignMap = new HashMap<>();

checkSignMap.put("request_id", requestId);

checkSignMap.put("med_org_code", medOrgCode);

checkSignMap.put("med_hos_code", medHosCode);

checkSignMap.put("cryptoInfo", cryptoInfo);

checkSignMap.put("timestamp", timestamp);

// 获取预签名字符串

String checkSignPreSignature = buildPreSignature(checkSignMap);

// 验签结果

boolean checkResult = sm2Verify(SM2_PUBLIC_KEY, checkSignPreSignature, signature);

if (!checkResult) {

    // 验签失败  错误响应...

}

// 校验 timestamp 参数是否超出 10 秒  如超出则错误响应  ...



// 解析数字信封

String sm4Key = sm2Decrypt(cryptoInfo, SM2_PRIVATE_KEY);

System.out.println("sm4key：" + sm4Key);



// SM4 密文解析出明文 json

String requestJson = sm4Decrypt(requestBody, sm4Key);
```

```
        JSONObject jsonObject = JSON.parseObject(requestJson);
        String idcard_value = jsonObject.getString("idcard_value");
        String idcard_type = jsonObject.getString("idcard_type");
        String other___ = jsonObject.getString("...省略");
        // his 处理具体业务  ...



        //  响应结果
        JSONObject responseObject = new JSONObject();
        responseObject.put("response_code", "1");
        responseObject.put("response_msg", "success");
        responseObject.put("response_data", "...省略");
        //  响应体 json
        String responseObjectJSON = responseObject.toJSONString();
        System.out.println("响应体 json：" + responseObjectJSON);
        //  对响应体 json 进行 sm4 加密
        String responseObjectJSONEncrypt = sm4Encrypt(responseObjectJSON, sm4Key);


        System.err.println(responseObjectJSONEncrypt);
        return responseObjectJSONEncrypt;
    }



    /**
     * SM2 私钥签名
```

```
     *
     * @param privateKey  私钥
     * @param content       待签名内容
     * @return  签名值
     */
    public static String sm2Sign(String privateKey, String content) {
        SM2 sm2 = new SM2(privateKey, null);
        return sm2.signHex(HexUtil.encodeHexStr(content));
    }

    /**
     * SM2 公钥验签
     *
     * @param publicKey  公钥
     * @param content     原始内容
     * @param sign          签名
     * @return  验签结果
     */
    public static boolean sm2Verify(String publicKey, String content, String sign) {
        SM2 sm2 = new SM2(null, publicKey);
        return sm2.verifyHex(HexUtil.encodeHexStr(content), sign);
    }

    /**
     * SM2 公钥加密
```

```
     *
     * @param content      原文
     * @param publicKey SM2 公钥
     * @return
     */
    public static String sm2Encrypt(String content, String publicKey) {
        SM2 sm2 = new SM2(null, publicKey);
        return sm2.encryptHex(content, KeyType.PublicKey);
    }

    /**
     * SM2 私钥解密
     *
     * @param encryptStr SM2 加密字符串
     * @param privateKey SM2 私钥
     * @return
     */
    public static String sm2Decrypt(String encryptStr, String privateKey) {
        SM2 sm2 = new SM2(privateKey, null);
        return StrUtil.utf8Str(sm2.decrypt(encryptStr, KeyType.PrivateKey));
    }

    /**
     * SM4 加密
     *
```

```
     * @param content  原文
     * @param key          SM4 秘钥
     * @return
     */
    public static String sm4Encrypt(String content, String key) {
        byte[] sm4KeyByte = HexUtil.decodeHex(key);
        String encryptBody = SmUtil.sm4(sm4KeyByte).encryptHex(content);
        return encryptBody;
    }

    /**
     * SM4 解密
     *
     * @param content SM4 加密字符串
     * @param key          SM4 秘钥
     * @return
     */
    public static String sm4Decrypt(String content, String key) {
        byte[] sm4KeyByte = HexUtil.decodeHex(key);
        String decryptBody = SmUtil.sm4(sm4KeyByte).decryptStr(content);
        return decryptBody;
    }

    /**
```

```
     *  生成预签名字符串
     *
     * @param header
     * @return
     */
    public static String buildPreSignature(HashMap<String, String> header) {
        if (MapUtils.isEmpty(header)) {
            return null;
        }
        List<String> keys = new ArrayList<>(header.keySet());
        Collections.sort(keys); //  按 ASCII 升序排序

        StringBuilder sb = new StringBuilder();
        for (String key : keys) {
            Object value = header.get(key);
            //  处理 null 和不同类型值
            String valueStr = (value != null) ? value.toString() : "";

            if (sb.length() > 0) {
                sb.append("&");
            }
            sb.append(key).append("=").append(valueStr);
        }
        return sb.toString();
    }
```

```
}
```

5.1.3.5 maven 依赖

```
<dependency>
    <groupId>cn.hutool</groupId>
    <artifactId>hutool-crypto</artifactId>
    <version>5.8.35</version>
</dependency>
```

5.1.4 接口定义

【10000】患者取号列表

地址：由医院提供给在线取号平台

请求方式：post 请求

说明：在线取号平台根据患者证件号+患者姓名向医院获取患者取号列表，其中患者证件号+患者姓名确定唯一用户信息，取号列表只返回挂号时间为当天的，请求参数为 json 字符串。

请求参数：

字段名	是否必填	字段类型	字段说明
name	是	String	患者姓名
idcard_type	是	String	证件类型（见《浙江省健康数据高铁安诊儿专线接口规范 第 1 部分：总则》5.1 章节 CVX-SFZJLBDM 中的代码）
idcard_value	是	String	证件号
order_status	是	String	预约号状态： 0 未取号 1 已取号 2 已取消 3 全部
request_id	是	String	请求 id，用于幂等控制

timestamp	是	String	发送请求的时间戳
signature	是	String	加签字符串

同步返回参数：

字段名	是否必填	字段类型	字段说明
request_id	是	String	请求 id，和请求方传入的 id 保持一致
response_code	是	String	请求结果，1 表示成功，其他表示失败
response_msg	是	String	提示信息
response_data	是	String	返回业务数据。格式为 json 数组，json 数组中的每个对象为业务对象。如果取号列表为空的时候则返回"[]"的加密字符串

业务数据 response_data：

字段名	是否必填	字段类型	字段说明
order_id	是	String	预约 id
is_pay	否	String	1 表示需要在线支付　其他或不填表示不需要在线支付（医保和自费都需要传 1）
order_no	否	String	医院订单号，如果 is_pay 为 1，且需要通过自费渠道支付时，订单号必填（医保支付不填） 同一个号源需要保证订单号唯一（即多次查询该号源返回的订单号为同一个）
medical_insurance_code	否	String	医院的医保编码（需要医保支付时返回），对应医院在医保平台的国标编码+分院编号（org_code+sub_org_code）

order_status	是	String	预约号状态： 0 未取号　1 已取号　2 已取消 3 全部
name	是	String	患者姓名
pId	是	String	门诊号
dept_name	是	String	科室名称
treatment_name	是	String	诊疗科室
sch_date	是	String	就诊时间 格式：yyyy-MM-dd HH:mm:ss
ampm	是	String	就诊场次　am 表示上午，pm 表示下午
doctor_name	是	String	医生姓名
reg_class	是	String	号类：1 普通科 2 名医　3 专家科
reg_fee	是	String	挂号费（单位：元）
disable	否	String	号源是否可取号 0 表示不可取号　1 表示可取号 默认是 1 可取号 由于有些号源存在医保结算问题，部分医院不想走让该号源走自费结算，但又想在省平台展示，增加此字段来解决该问题
remark	否	String	如果不能取号，则需要说明不能取号的原因
reg_num	是	String	就诊序号
reg_addr	是	String	就诊地址
org_id	否	String	院区编码

oper_date	否	String	取号时间 格式：yyyy-MM-dd HH:mm:ss
appointment_treatment_no	是	String	预约流水号

【20000】患者取号

地址：由医院提供给在线取号平台

请求方式：post 请求

说明：在线取号平台根据【10000】患者取号列表接口返回的预约 id（order_id），向医院发起在线取号。请求参数为 json 字符串。医院托收、在线支付-医保支付模式调用。

请求参数：

字段名	是否必填	字段类型	字段说明
order_id	是	String	预约 id
medical_insurance_token	否	String	医保电子凭证 token，医保用户选择医保移动支付付款并取号时必传

同步返回参数：

字段名	是否必填	字段类型	字段说明
response_data	否	String	返回业务数据。格式为 json 数组，json 数组中的每个对象为业务对象。如果取号列表为空的时候则返回"[]"的加密字符串
response_code	是	String	请求结果，1 表示成功，其他表示失败
response_msg	是	String	提示信息

业务数据 response_data：

字段名	是否必填	字段类型	字段说明

medical_insurance_number	是	String	移动支付流水号，对应浙江医保支付门户定点医药机构接口规范查询费用列表(ORG_002)接口，org_trace_no 字段，HIS 系统跟踪号（唯一），医保支付时医保中心会调用 org002 接口进行费用列表查询
org_card_no	是	String	院内医保建档卡号
org_card_type	是	String	院内医保建档卡类型

【30000】自费支付成功通知

地址：由医院提供给在线取号平台

请求方式：post 请求

说明：在线取号平台根据【10000】患者取号列表接口返回的 order_no，在省平台创建订单并唤起第三方支付，支付成功后，将通知医院，医院接受通知后完成院内取号动作，并将结果返回至省平台，如果院内操作失败，省平台将直接发起退款操作。请求参数为 json 字符串。

该接口仅纯自费模式时调用，医保+自费的混合支付不调用。

备注：

1、对账：在线取号所有产生的支付和退款流水均可通过省平台电子健康卡或第三方支付渠道进行对账，对账功能取决于每家医院。如果医院原来对账是和第三方支付渠道对账，则直接拿着 bq_trade_no 去第三方支付渠道对账即可，如果医院原来是和省平台电子健康卡平台对账，则走原来电子健康卡对账功能即可，详情参考电子健康卡 13004、13005、13007 接口

2、退款：如果医院在收到取号支付成功之后，由于患者在医院某种原因，需要发起退款，则可以走电子健康卡退款接口即可，详情参考电子健康卡 13003 接口

请求参数：

字段名	是否必填	字段类型	字段说明
order_no	是	String	医院订单号
bq_trade_no	是	String	支付单号，用于对账

pay_channel_trade_no	是	String	第三方交易流水号
pay_channel	是	String	支付渠道(1 支付宝,2 银联)
pay_time	是	String	付款时间 yyyy-MM-dd HH:mm:ss
pay_amount	是	String	支付金额，单位为：元
request_id	是	String	请求 id，用于幂等控制
timestamp	是	String	发送请求的时间戳
signature	是	String	加签字符串

同步返回参数：

字段名	是否必填	字段类型	字段说明
response_code	是	String	请求结果，1 表示成功，其他表示失败， 如果是失败，省平台将会直接发起退款
response_msg	是	String	提示信息

【40000】查询号源状态

地址：由医院提供给在线取号平台

请求方式：post 请求

说明：在线取号平台根据预约 id（order_id），查询取号结果。请求参数为 json 字符串。

请求参数：

字段名	是否必填	字段类型	字段说明
order_id	是	String	预约 id
appointment_date	是	String	号源时间,考虑到有些医院预约 id 是当天

			唯一，所以查询会带上号源时间
request_id	是	String	请求 id，用于幂等控制
timestamp	是	String	发送请求的时间戳
signature	是	String	加签字符串

同步返回参数：

字段名	是否必填	字段类型	字段说明
response_code	是	String	请求结果，1 表示成功，其他表示失败
response_msg	是	String	提示信息
response_data	是	String	返回业务数据。格式为 json 数组，json 数组中的每个对象为业务对象。如果取号列表为空的时候则返回"[]"的加密字符串

业务数据 response_data：

字段名	是否必填	字段类型	字段说明
order_status	是	String	预约号状态： 0 未取号 1 已取号 2 已取消

5.2 排队叫号

5.2.1 服务说明

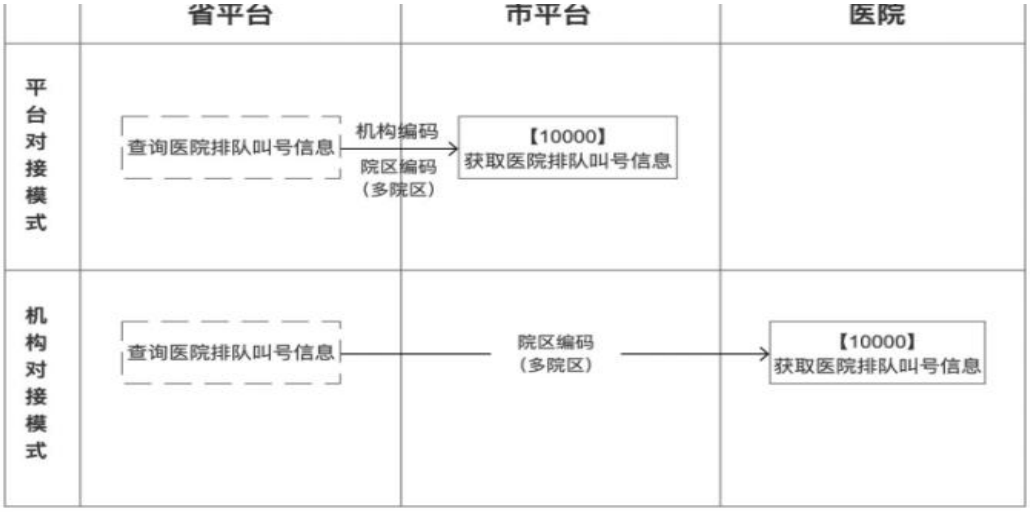
5.2.1.1 服务概述

安诊儿服务平台在线取号服务能力复用**浙江省排队叫号平台**。源端医疗服务机构通过接入浙江省排队叫号平台，完成排队叫号服务能力的接入。

浙江省排队叫号平台是响应政府数字化改革的要求，通过集成各医院的排队叫号资源，搭建的全省统一的排队叫号平台，支持查看个人以及医院的排队叫号队列信息，为省内居民提供便捷的线上就医体验。

5.2.1.3 业务流程

表 7 业务流程图



5.2.1.4 用户动线

表 8 浙里办排队叫号用户动线

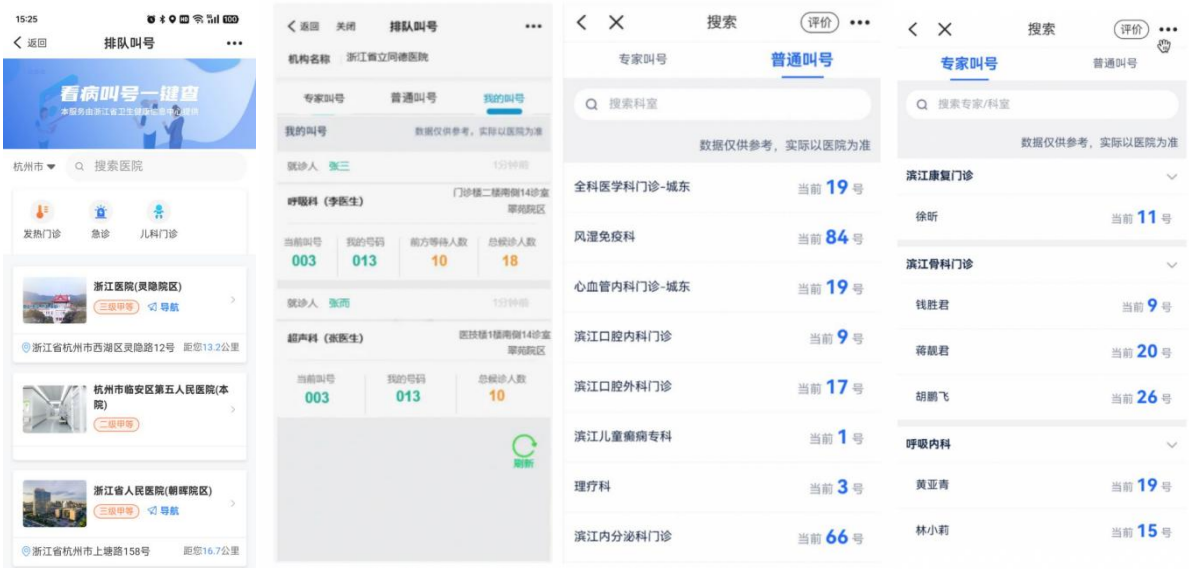


表 9 安诊儿查询叫号信息用户动线

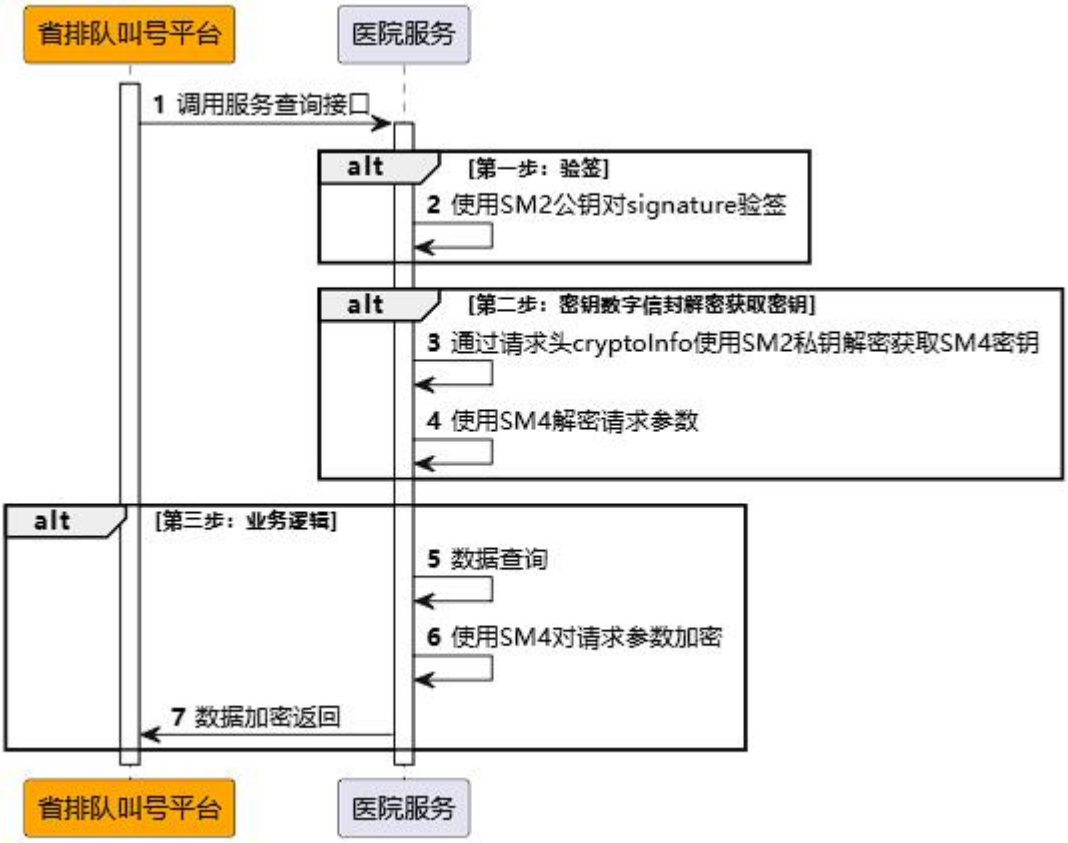

9:41
安诊儿
安诊儿陪你全流程就医
你的就诊信息
心内科门诊
就诊中
眼科
待取号
就诊提醒
无需签到，请直接前往科室候诊
叫号中
心内科门诊
浙江省人民医院（朝晖院区）
12
你的序号
4号
当前序号
8人
前方候诊
32分钟
预计等待
就诊地址
四楼B2区
就诊医生
张医生
刷新排队情况
刷新时间：10:30 请以医院实际叫号为准
已到号，请尽快就诊
心内科门诊
浙江省人民医院（朝晖院区）
12
你的序号
12
当前叫号
诊室
17号诊室
就诊地址
四楼B2区
就诊医生
张医生
已到号，请尽快前往诊室就诊
已过号，请重新签到
心内科门诊
浙江省人民医院（朝晖院区）
过号
12
你的序号
13
当前叫号
就诊地址
四楼B2区
就诊医生
张医生
已过号，请重新签到

5.2.2 请求头

字段名	是否必填	参与签名	字段类型	字段说明
request_id	是	是	String	请求唯一标识(随机生成不重复即可)
timestamp	是	是	String	请求时间戳
signature	是	是	String	加签字符串（规则见加 签机制）
org_code	是	是	String	机构编码
hos_code	是	是	String	院区编码
cryptoInfo	是	是	String	密钥数字信封（SM2 加密后的 SM4 秘钥）

5.2.3 数据加密加签处理

表 10 加解密流程时序图



5.2.3.1 签名&加密流程

5.2.3.1.1 签名规则

为了请求过程的安全性，业务各方之间的请求需要实现加签，以保证数据的安全性。请求方实现对请求参数数据的加签，服务方根据 Header 请求参数 signature，对签名进行验签，签名不合法的请求将会被拒绝，同时对 timestamp 时间戳进行校验，超出 10 秒范围内的请求将会被拒绝。报文的签名及验证使用 SM2 算法，具体处理机制要求如下:

1）对 Header 所有非空业务参数(signature 不参与签名)按照 key 的 ascii 顺序进行升序排序，然后拼接成一串待加签字符串，格式为：key=value&key=value。

签名示例：

```
cryptoInfo=04eafaa325aea8f19463820c1e35553433bc1240a597bf085f908b73d39341f04cfcf95c6662a10c5b55188c326c9d04ec503c52ce28b76eb24f6d097b35337e6c6bda04a5ab4e5612fcc591fc207cde14fa648767ac7185e07303b7c76043b74ae62f4ec61ce255c35a75b3b5121a5037b55cc8f1f6f546d701cae56e34361008&hos_code=hos_code_001&org_code=org_code_001&request_id=7761fa6d-5ca0-4f4e-a8f0-c78f52ee4abd&timestamp=1742110076829
```

2）生成签名：对拼接的字符串使用 SM2 算法通过私钥生成加签字符串 signature

5. 2. 3. 1. 2 签名示例

SM2 公钥	04b674b6edfd08f754410b21cc3c8b522f318a1f1b27403bc63a290b7e1790d03d967d06c46e69357793967ff42a34d77ed5d7bb40d31b7f5ab0b0840017cff114
SM2 私钥	a99d6978656355b570a19cc707b2ea68ed033c3fa436df9c0a15e84a06069c7b
SM4 密钥	726be1646879423e26eb7977c4d80c48
密钥数字信封 cryptoInfo（使用 SM2 私钥加密后的 SM4 密钥）	04eafaa325aea8f19463820c1e35553433bc1240a597bf085f908b73d39341f04cfcf95c6662a10c5b55188c326c9d04ec503c52ce28b76eb24f6d097b35337e6c6bda04a5ab4e5612fcc591fc207cde14fa648767ac7185e07303b7c76043b74ae62f4ec61ce255c35a75b3b5121a5037b55cc8f1f6f546d701cae56e34361008
加签前参数内容	cryptoInfo=04eafaa325aea8f19463820c1e35553433bc1240a597bf085f908b73d39341f04cfcf95c6662a10c5b55188c326c9d04ec503c52ce28b76eb24f6d097b35337e6c6bda04a5ab4e5612fcc591fc207cde14fa648767ac7185e07303b7c76043b74ae62f4ec61ce255c35a75b3b5121a5037b55cc8f1f6f546d701cae56e34361008&hos_code=hos_code_001&org_code=org_code_001&request_id=7761fa6d-5ca0-4f4e-a8f0-c78f52ee4abd&timestamp=1742110076829
加签后字符串(签名)	3046022100bdb527e86a85fbe1e17f577a177a7328a6d0f9d247ae795accafb33ad2d1d198022100a564bdac01384790252d0b4bbef6ca286a1bb9ed6a3b57197c7ee0d59ff51e18

5.2.3.2 请求体解密示例

数字信封 SM4 密钥	726be1646879423e26eb7977c4d80c48
请求体 Body（SM4 加密密文）	02a1efbc250f047f73923a84c73931fc23f36823cfd5a9483ca1432527cab960ebc11e79d6a310c24b7148c0097313cb1d4e7d26b0531b0dd261c271a80fbfb1b62c16d0cdb201f89b868686ed4c83fbb19250cce547c92650068f196521870165a1ef5fffdf7a2521acb1fde098a607
业务请求参数明文	{"name":"测试人员","idcard_value":"330100********0000","idcard_type":"01","search_type":"1"}
最终请求体 Body	{"name":"测试人员","idcard_value":"330100********0000","idcard_type":"01","search_type":"1"}

5.2.3.3 响应体加密示例

数字信封 SM4 密钥	726be1646879423e26eb7977c4d80c48
Body 参数明文	{"response_code":"1","response_data":"...省略","response_msg":"success"}
业务参数 SM4 加密密文	31b7be894984b26f3acc23f3f2ad162a6131bb143013b38b59a0937e43d4287659c4b9ebd0998194dbb881cf81d0abfc3206b907af244fe0576da8ed746d8c7d3f693641504c13ec5bcd415d9932e0dd
最终响应体 Body	31b7be894984b26f3acc23f3f2ad162a6131bb143013b38b59a0937e43d4287659c4b9ebd0998194dbb881cf81d0abfc3206b907af244fe0576da8ed746d8c7d3f693641504c13ec5bcd415d9932e0dd

5.2.3.4 Java 代码示例

```
import cn.hutool.core.util.HexUtil;

import cn.hutool.core.util.StrUtil;
```

```
import cn.hutool.crypto.SmUtil;

import cn.hutool.crypto.asymmetric.KeyType;

import cn.hutool.crypto.asymmetric.SM2;

import com.alibaba.fastjson.JSON;

import com.alibaba.fastjson.JSONObject;

import com.example.demo.utils.HttpUtils;

import org.apache.commons.collections4.MapUtils;

import org.junit.jupiter.api.Test;



import java.util.*;



/**

 * 排队叫号请求流程-代码示例

 */

public class CallQueueRequestDemo {



    // sm4 秘钥

    static String SM4_KEY = "726be1646879423e26eb7977c4d80c48";

    // sm2 公钥

    static String SM2_PUBLIC_KEY =
"04b674b6edfd08f754410b21cc3c8b522f318a1f1b27403bc63a290b7e1790d03d967d06c46e69357793967ff42a34d77ed5d7bb40d31b7f5ab0b
0840017cff114";

    // sm2 私钥
```

```
static String SM2_PRIVATE_KEY = "a99d6978656355b570a19cc707b2ea68ed033c3fa436df9c0a15e84a06069c7b";



/**
 * 查询 His 队列信息
 */
@Test
public void queryQueueInfo() {

    // SM2 公钥加密 sm4 秘钥字符串
    String sm2EncryptSm4Key = sm2Encrypt(SM4_KEY, SM2_PUBLIC_KEY);
    // Header 参数
    HashMap<String, String> header = new HashMap<>();
    header.put("request_id", UUID.randomUUID().toString());
    header.put("org_code", "org_code_001");
    header.put("hos_code", "hos_code_001");
    header.put("cryptoInfo", sm2EncryptSm4Key);
    header.put("timestamp", String.valueOf(System.currentTimeMillis()));

    String preSignature = buildPreSignature(header);
    System.out.println("预签名字符串：" + preSignature);
    String signature = sm2Sign(SM2_PRIVATE_KEY, preSignature);
    header.put("signature", signature);
```

```
        System.out.println("signature：" + signature);



        // 请求体参数

        JSONObject requestBody = new JSONObject();

        requestBody.put("name", "测试人员");

        requestBody.put("idcard_type", "01");

        requestBody.put("idcard_value", "330100********0000");

        requestBody.put("search_type", "1");

        // 请求体参数 Json

        String requestBodyJson = requestBody.toJSONString();

        // 请求体参数 Json 加密

        String requestBodyJsonEncrypt = sm4Encrypt(requestBodyJson, SM4_KEY);



        // 请求 his 排队叫号接口

        String result = HttpUtils.post("请求 his 排队叫号接口", "https://ip:port/get/queue/info", requestBodyJsonEncrypt, header);



        // 解密 ...

        String responseBody = sm4Decrypt(result, SM4_KEY);

        // 解析

        JSONObject jsonObject = JSON.parseObject(responseBody);

        // 处理业务 ...省略

    }
```

```
/**
 * 获取医院排队叫号信息查询接口-his 处理代码示例
 */
public String getQueueInfo(HashMap<String, String> header, String requestBody) {
    String requestId = header.get("request_id");
    String orgCode = header.get("org_code");
    String hosCode = header.get("hos_code");
    String cryptoInfo = header.get("cryptoInfo");
    String timestamp = header.get("timestamp");
    String signature = header.get("signature");

    // 校验签名处理
    HashMap<String, String> checkSignMap = new HashMap<>();
    checkSignMap.put("request_id", requestId);
    checkSignMap.put("org_code", orgCode);
    checkSignMap.put("hos_code", hosCode);
    checkSignMap.put("cryptoInfo", cryptoInfo);
    checkSignMap.put("timestamp", timestamp);
    // 获取预签名字符串
    String checkSignPreSignature = buildPreSignature(checkSignMap);
    // 验签结果
    boolean checkResult = sm2Verify(SM2_PUBLIC_KEY, checkSignPreSignature, signature);
    if (!checkResult) {
```

```
        // 验签失败  错误响应...

    }

    // 校验 timestamp 参数是否超出 10 秒  如超出则错误响应  ...



    // 解析数字信封

    String sm4Key = sm2Decrypt(cryptoInfo, SM2_PRIVATE_KEY);

    System.out.println("sm4key：" + sm4Key);



    // SM4 密文解析出明文 json

    String requestJson = sm4Decrypt(requestBody, sm4Key);

    JSONObject jsonObject = JSON.parseObject(requestJson);

    String idcard_value = jsonObject.getString("idcard_value");

    String idcard_type = jsonObject.getString("idcard_type");

    String other___ = jsonObject.getString("...省略");

    // his 处理具体业务  ...



    // 响应结果

    JSONObject responseObject = new JSONObject();

    responseObject.put("response_code", "1");

    responseObject.put("response_msg", "success");

    responseObject.put("response_data", "...省略");

    // 响应体 json
```

```
        String responseObjectJSON = responseObject.toJSONString();
        System.out.println("响应体 json：" + responseObjectJSON);
        // 对响应体 json 进行 sm4 加密
        String responseObjectJSONEncrypt = sm4Encrypt(responseObjectJSON, sm4Key);

        return responseObjectJSONEncrypt;
    }


    /**
     * SM2 私钥签名
     *
     * @param privateKey  私钥
     * @param content       待签名内容
     * @return  签名值
     */
    public static String sm2Sign(String privateKey, String content) {
        SM2 sm2 = new SM2(privateKey, null);
        return sm2.signHex(HexUtil.encodeHexStr(content));
    }

    /**
     * SM2 公钥验签
```

```
     *
     * @param publicKey  公钥
     * @param content      原始内容
     * @param sign           签名
     * @return  验签结果
     */
    public static boolean sm2Verify(String publicKey, String content, String sign) {
        SM2 sm2 = new SM2(null, publicKey);
        return sm2.verifyHex(HexUtil.encodeHexStr(content), sign);
    }

    /**
     * SM2 公钥加密
     *
     * @param content      原文
     * @param publicKey SM2 公钥
     * @return
     */
    public static String sm2Encrypt(String content, String publicKey) {
        SM2 sm2 = new SM2(null, publicKey);
        return sm2.encryptHex(content, KeyType.PublicKey);
    }
```

```
/**
 * SM2 私钥解密
 *
 * @param encryptStr SM2 加密字符串
 * @param privateKey SM2 私钥
 * @return
 */
public static String sm2Decrypt(String encryptStr, String privateKey) {
    SM2 sm2 = new SM2(privateKey, null);
    return StrUtil.utf8Str(sm2.decrypt(encryptStr, KeyType.PrivateKey));
}

/**
 * SM4 加密
 *
 * @param content  原文
 * @param key      SM4 秘钥
 * @return
 */
public static String sm4Encrypt(String content, String key) {
    byte[] sm4KeyByte = HexUtil.decodeHex(key);
    String encryptBody = SmUtil.sm4(sm4KeyByte).encryptHex(content);
    return encryptBody;
```

```
}


/**
 * SM4 解密
 *
 * @param content SM4 加密字符串
 * @param key         SM4 秘钥
 * @return
 */
public static String sm4Decrypt(String content, String key) {
        byte[] sm4KeyByte = HexUtil.decodeHex(key);
        String decryptBody = SmUtil.sm4(sm4KeyByte).decryptStr(content);
        return decryptBody;
}




/**
 *  生成预签名字符串
 *
 * @param header
 * @return
 */
public static String buildPreSignature(HashMap<String, String> header) {
```

```
        if (MapUtils.isEmpty(header)) {
            return null;
        }
        List<String> keys = new ArrayList<>(header.keySet());
        Collections.sort(keys); //  按 ASCII 升序排序

        StringBuilder sb = new StringBuilder();
        for (String key : keys) {
            Object value = header.get(key);
            //  处理 null 和不同类型值
            String valueStr = (value != null) ? value.toString() : "";

            if (sb.length() > 0) {
                sb.append("&");
            }
            sb.append(key).append("=").append(valueStr);
        }
        return sb.toString();
    }
}
```

5.2.3.5 maven 依赖

```
<dependency>
  <groupId>cn.hutool</groupId>
  <artifactId>hutool-crypto</artifactId>
```

```
    <version>5.8.35</version>
</dependency>
```

5.2.4 接口定义

【10000】 获取医院排队叫号信息

地址： 由医院提供给省平台

请求方式：post 请求

特别说明：

1. 返回当前就诊人排队叫号信息：search_type=1 ，需要返回就诊人在当前就诊科室的排队叫号信息（ 如在 多个科室候诊，就返回多个科室的排队叫号信息）；
2. 返回某一科室排队叫号信息：search_type=2 ，根据科室名称（模糊搜索）返回科室下的排队叫号信息；
3. 返回医生当前排队叫号信息：search_type=3 ，根据医生名字（模糊搜索）返回该医生排队叫号信息；
4. 返回医院所有排队叫号信息：search_type=4 ，返回医院当前所有科室的排队叫号信息；

备注：每次请求时候，需要返回当前时间点最新的排队叫号状态。

请求参数：

字段名	是否必填	字段类型	字段说明
search_type	是	String	1 根据患者查询 2 根 据科室查询 3 根据医生查询 4 全 部查询
name	否	String	患者姓名 search_type 为 1 时必填
idcard_type	否	String	证件类型（见《浙江省健康数据高铁安诊儿专线接口规范 第 1 部分：总则》5.1 章节 CVX-SFZJLBDM 中的代码） search_type 为 1 时必 填
idcard_value	否	String	证件号 search_type 为1 时必填
dept_name	否	String	科室名称，支持模糊搜索 search_type 为 2 时 dept_name 必填
doctor_name	否	String	医生名字，支持模糊搜索 search_type 为 3 时必填
reg_class	否	String	0 普通号、1 专家号（包括名医、特需）2 检验检查

registration_treatment_no	否	String	挂号流水号
page_num	是	Integer	页码
page_size	是	Integer	当前页大小

（2）同步返回参数：

字段名	是否必填	字段类型	字段说明
response_code	是	String	请求结果，1 表示成功 其他表示失败
response_msg	是	String	提示信息
response_data	是	JSONArray	如果未找到满足条件的叫号信息，则返回[]

业务数据 response_data：

1 、科室下面的一位医生对应 doctor_list 中的一条记录；

2 、若为普通号无法确认医生名字时，doctor_list 中 doctor_name 传 “普通号 ” ，doctor_id 传 “0 ”；

3 、若为普通号能确认医生名字时，doctor_list 中 doctor_name 与 doctor_id 传对应医生的姓名及 id；

4 、若 search_type=1 ，只返回就诊人对应的医生（ 或普通号）。

5 、若 search_type=3 ，只返回所需的医生（ 或普通号）。

6 、若没有找到满足条件的医生（ 或普通号） ，则返回[]。

字段名	是否必填	字段类型	字段说明
doctor_list	是	JSONArray	医生/诊室队列
dept_code	是	String	科室编码
dept_name	是	String	科室名称
fever_clinic_sign	是	String	发热门诊标志 0-否 1- 是

业务数据 doctor_list：

字段名	是否必填	字段类型	字段说明
doctor_name	是	String	队列名称（ 医生/诊室/其他队列)
doctor_id	是	String	队列 ID（ 医生/诊室/其他队列)

current_user_name	是	String	当前就诊患者姓名
current_reg_num	是	String	当前就诊序号
current_ampm	是	String	当前就诊患者是上午的号还是下午的号，am 表示上午，pm 表示下午
reg_num	否	String	患者就诊序号，入参有患者信息时，需要返回该字段
reg_class	是	String	0 普通号 、1 专家号（ 包括名医 、特需） 2 检验检查
reg_addr	是	String	就诊地址
today_appointment_ num	否	String	当前挂号人数
current_wait_num	是	String	当前候诊人数
patient_wait_num	否	String	患者前方等待人数
patient_wait_time	否	Int	患者前方等待时间（单位：分钟）
is_call	否	Int	叫号状态，0 未叫，1 已叫，2 过号
reg_type	否	String	号源类别，0门诊 、1 治疗 、2 检验 、3 检查
registration_treatmen t_no	否	String	挂号流水号

doctor_num	是	String	当前接诊医生数

请求示例

请求头部：

{

"request_id":"7ab3599ae3ca4bee882595aeffb18424",

"timestamp":"1581737772069",

"signature":"6be0a9a9372f7313307f75ba7c2d825f",

"org_code": "432112312",

"hos_code": "01",

"cryptoInfo":"042335391514c4504e828ea1858400a5f1bb5aa62c58628189882deb5954c124ddcbfabaa6868fd49f4ae6c2c5def809f391802932bb276e7aca540b10a233a988603a159ce6801292b48e4b48366b7a27e8219dfecb73e9686ccf1d668d659f70f3cfcfb100726d9ec8463741cee0df61cb8d5fb47bb264f49f56d55815042deb"

}

请求体:

{

"name": "小明",

"idcard_type": "01",

"idcard_value": "123",

"search_type":"1"

}

请求返回内容：

{

"response_code": "1",

"response_msg": "success",

"response_data": [{

"dept_name": "测试科室",

"dept_code": "P001",

"fever_clinic_sign":"1",

"doctor_list": [{

"doctor_name": "医生名称",

"doctor_id": "张晓红",

"current_user_name": "王",

"current_reg_num": "66",

"reg_num": "78",

"current_ampm": "am",

"reg_class": "1",

"reg_addr": "5 楼肤科 3 诊室",

"today_appointment_num":"124",

"current_wait_num":"58",

"doctor_num":"2"

}]

}]

5.3 费用清单

5.3.1 服务说明

5.3.1.1 服务概述

用户院内门急诊、住院等环节收费清单查询，支撑安诊儿云陪诊流程中通过缴费卡片展示完整的费用明细。

5.3.1.2 接入模式

接入机构按照此接口规范开发费用清单接口，并通过市平台将接口代理到政务网，安诊儿服务平台通过接口查询获取用户费用清单。

5.2.1.3 用户动线

表 11 安诊儿费用清单查询用户动线

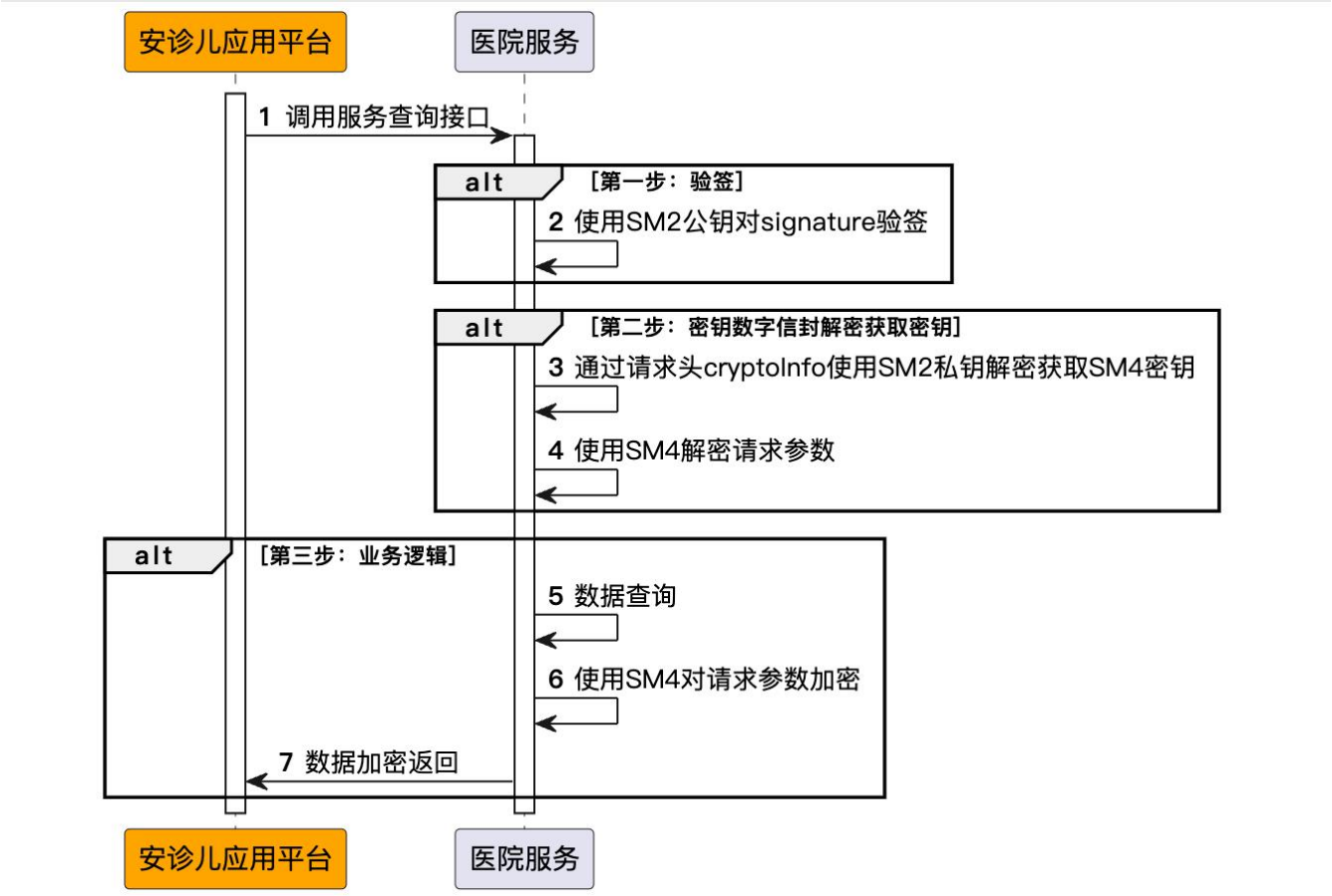
5.3.2 接口定义

请按照以下要求实现接口，后提供请求 api 的路径

如：http(s)://host:port/fyqd-query。

- **URL**：自定义
- **Method**：POST
- **需要登录**：否
- **需要鉴权**：否

说明：安诊儿专线服务调用医院接口时，采用数字信封的加密方式，流程见下图



安诊儿专线服务请求 his 流程-代码示例

```
import cn.hutool.core.util.HexUtil;

import cn.hutool.core.util.StrUtil;

import cn.hutool.crypto.SmUtil;

import cn.hutool.crypto.asymmetric.KeyType;

import cn.hutool.crypto.asymmetric.SM2;

import com.alibaba.fastjson.JSON;

import com.alibaba.fastjson.JSONObject;

import org.apache.commons.collections4.MapUtils;

import org.junit.jupiter.api.Test;

import java.io.Serializable;
```

```
import java.util.*;



/**
 * 安诊儿专线服务请求his流程-代码示例
 */
public class RequestHisDemoFile {



    /**
     * 安诊儿服务平台请求his流程实例
     * 包含数字信封加密过程
     */
    @Test
    public void angelServerFyqdQuery() {
        // sm2公私钥
        String SM2_PUBLIC_KEY =
"04b674b6edfd08f754410b21cc3c8b522f318a1f1b27403bc63a290b7e1790d03d967d06c46e69357793967ff42a34
d77ed5d7bb40d31b7f5ab0b0840017cff114";
        String SM2_PRIVATE_KEY =
"a99d6978656355b570a19cc707b2ea68ed033c3fa436df9c0a15e84a06069c7b";
        // 医疗机构统一社会信用代码
        String medOrgCode = "medOrgCode_001";
        // 院区编码
        String medHosCode = "medHosCode_001";
```

```
// 区域平台归属单位的18位统一社会信用代码
String platformCode = "platformCode_001";


// 平台调用安诊儿秘钥服务获取SM4秘钥 ...
String sm4Key = AngelAccessServer.getSm4Key(medOrgCode, medHosCode, platformCode);


System.out.println("------------------ ↓安诊儿平台请求his示例↓ ----------------");
HashMap<String, String> header = new HashMap<>();
// 交易编码：查询
header.put("tradeCode", "1003");
// 数据集编码
header.put("datasetCode", "HDSD.FYQD");
// 请求唯一标识 随机数
header.put("requestId", UUID.randomUUID().toString());
// 医疗机构统一社会信用代码
header.put("medOrgCode", medOrgCode);
// 院区编码
header.put("medHosCode", medHosCode);
// 区域平台归属单位的18位统一社会信用代码 非区域平台接入时不可填入
header.put("platformCode", platformCode);
// 请求时间戳（13位毫秒级，例：1735704000000）
header.put("timestamp", String.valueOf(System.currentTimeMillis()));
```

```
// 密钥数字信封（SM2 加密后的 SM4 秘钥）
String cryptoInfo = sm2Encrypt(sm4Key, SM2_PUBLIC_KEY);
header.put("cryptoInfo", cryptoInfo);
System.out.println("cryptoInfo：" + cryptoInfo);

// 预签名字符串
String preSignature = buildPreSignature(header);
String sign = sm2Sign(SM2_PRIVATE_KEY, preSignature);
// 加签字符串
header.put("signature", sign);

// 业务请求参数
JSONObject businessParam = new JSONObject();
businessParam.put("tyshxydm", medOrgCode);
businessParam.put("yqdm", medHosCode);
businessParam.put("jfdbm", "jfdbm");
businessParam.put("...", "...省略");
// 业务请求体 json 字符串
String jsonString = businessParam.toJSONString();
String bizRequestEncrypt = sm4Encrypt(jsonString, sm4Key);

// 最终请求体
```

```
    JSONObject bizRequest = new JSONObject();
    bizRequest.put("bizRequest", bizRequestEncrypt);
    // 最终请求体 json
    String bizRequestJson = bizRequest.toJSONString();

    // 调用 his 接口
    String result = fyqd_query(header, bizRequestJson);
    System.out.println("请求 his 响应结果：" + result);
    // 解析结果 后续操作 ...
    System.out.println("----------------- ↑安诊儿平台请求 his 示例↑ -----------------");
}

/**
 * 费用清单查询接口-his 处理代码示例
 *
 * @param header
 * @param requestBody
 * @return
 */
public String fyqd_query(HashMap<String, String> header, String requestBody) {
    System.out.println("---------------- ↓his 接收安诊儿平台请求处理↓ ----------------");
```

```
        // sm2 公私钥

        String SM2_PRIVATE_KEY =
"a99d6978656355b570a19cc707b2ea68ed033c3fa436df9c0a15e84a06069c7b";

        String SM2_PUBLIC_KEY =
"04b674b6edfd08f754410b21cc3c8b522f318a1f1b27403bc63a290b7e1790d03d967d06c46e69357793967ff42a34
d77ed5d7bb40d31b7f5ab0b0840017cff114";



        // 交易编码：查询

        String tradeCode = header.get("tradeCode");

        // 数据集编码

        String datasetCode = header.get("datasetCode");

        // 请求唯一标识 随机数 his 服务系统需日志打印

        String requestId = header.get("requestId");

        // 医疗机构统一社会信用代码

        String medOrgCode = header.get("medOrgCode");

        // 院区编码

        String medHosCode = header.get("medHosCode");

        // 区域平台归属单位的 18 位统一社会信用代码

        String platformCode = header.get("platformCode");

        // 请求时间戳（13 位毫秒级，例：1735704000000）

        String timestamp = header.get("timestamp");

        // 密钥数字信封（SM2 加密后的 SM4 秘钥）

        String cryptoInfo = header.get("cryptoInfo");
```

```
// 加签字符串
String signature = header.get("signature");


// 校验签名处理
HashMap<String, String> checkSignMap = new HashMap<>();
checkSignMap.put("tradeCode", tradeCode);
checkSignMap.put("datasetCode", datasetCode);
checkSignMap.put("requestId", requestId);
checkSignMap.put("medOrgCode", medOrgCode);
checkSignMap.put("medHosCode", medHosCode);
checkSignMap.put("platformCode", platformCode);
checkSignMap.put("timestamp", timestamp);
checkSignMap.put("cryptoInfo", cryptoInfo);


// 获取预签名字符串
String preSignature = buildPreSignature(checkSignMap);
// 验签结果
boolean checkResult = sm2Verify(SM2_PUBLIC_KEY, preSignature, signature);
if (!checkResult) {
    // 验签失败 错误响应...
}
// 校验 timestamp 参数是否超出 10 秒 如超出则错误响应 ...
```

```
// 解析数字信封
String sm4Key = sm2Decrypt(cryptoInfo, SM2_PRIVATE_KEY);
System.out.println("sm4key：" + sm4Key);
// SM4 密文
JSONObject requestObject = JSON.parseObject(requestBody);
String bizRequestEncrypt = requestObject.getString("bizRequest");
// 请求体明文 Json 字符串
String requestJson = sm4Decrypt(bizRequestEncrypt, sm4Key);
JSONObject bizRequest = JSON.parseObject(requestJson);
bizRequest.getString("tyshxydm");
bizRequest.getString("yqdm");
bizRequest.getString("jfdbm");
bizRequest.getString("...");
// 处理具体业务 ....

// 响应对象 ....
JSONObject responseBiz = new JSONObject();
responseBiz.put("ydzflsh", "ydzflsh");
responseBiz.put("ynybjdkh", "ynybjdkh");
responseBiz.put("...", "省略...");
// 响应对象 Json
```

```
        String responseBizJson = responseBiz.toJSONString();
        // 响应对象加密
        String responseBizEncrypt = sm4Encrypt(responseBizJson, sm4Key);

        JSONObject responseObject = new JSONObject();
        responseObject.put("traceId", UUID.randomUUID().toString());
        responseObject.put("requestId", requestId);
        responseObject.put("success", true);
        responseObject.put("responseBiz", responseBizEncrypt);
        System.out.println("---------------- ↑his接收安诊儿平台请求处理↑ ----------------");

        return responseObject.toJSONString();
    }


    /**
     * 返回密钥信息
     */
    public class SecretKeyResponse implements Serializable {

        private String cryptoInfo;
        private String version;
```

```
    public String getCryptoInfo() {
        return cryptoInfo;
    }

    public void setCryptoInfo(String cryptoInfo) {
        this.cryptoInfo = cryptoInfo;
    }

    public String getVersion() {
        return version;
    }

    public void setVersion(String version) {
        this.version = version;
    }
}

/**
 * SM2 私钥签名
 *
 * @param privateKey 私钥
```

```
     * @param content    待签名内容
     * @return 签名值
     */
    public static String sm2Sign(String privateKey, String content) {
        SM2 sm2 = new SM2(privateKey, null);
        return sm2.signHexFromHex(HexUtil.encodeHexStr(content));
    }

    /**
     * SM2 公钥验签
     *
     * @param publicKey 公钥
     * @param content   原始内容
     * @param sign      签名
     * @return 验签结果
     */
    public static boolean sm2Verify(String publicKey, String content, String sign) {
        SM2 sm2 = new SM2(null, publicKey);
        return sm2.verifyHex(HexUtil.encodeHexStr(content), sign);
    }

    /**
```

```
     * SM4 加密
     *
     * @param content 原文
     * @param key     SM4 秘钥
     * @return
     */
    public static String sm4Encrypt(String content, String key) {
        byte[] sm4KeyByte = HexUtil.decodeHex(key);
        String encryptBody = SmUtil.sm4(sm4KeyByte).encryptHex(content);
        return encryptBody;
    }


    /**
     * SM4 解密
     *
     * @param content SM4 加密字符串
     * @param key     SM4 秘钥
     * @return
     */
    public static String sm4Decrypt(String content, String key) {
        byte[] sm4KeyByte = HexUtil.decodeHex(key);
        String decryptBody = SmUtil.sm4(sm4KeyByte).decryptStr(content);
```

```
    return decryptBody;
}

/**
 * 生成预签名字符串
 *
 * @param header
 * @return
 */
public static String buildPreSignature(HashMap<String, String> header) {
    if (MapUtils.isEmpty(header)) {
        return null;
    }
    List<String> keys = new ArrayList<>(header.keySet());
    Collections.sort(keys); // 按 ASCII 升序排序

    StringBuilder sb = new StringBuilder();
    for (String key : keys) {
        Object value = header.get(key);
        // 处理 null 和不同类型值
        String valueStr = (value != null) ? value.toString() : "";
```

```
            if (sb.length() > 0) {
                sb.append("&");
            }
            sb.append(key).append("=").append(valueStr);
        }
        return sb.toString();
    }

    /**
     * SM2 公钥加密
     *
     * @param content   原文
     * @param publicKey SM2 公钥
     * @return
     */
    public static String sm2Encrypt(String content, String publicKey) {
        SM2 sm2 = new SM2(null, publicKey);
        return sm2.encryptHex(content, KeyType.PublicKey);
    }

    /**
     * SM2 私钥解密
```

```
     *
     * @param encryptStr SM2 加密字符串
     * @param privateKey SM2 私钥
     * @return
     */
    public static String sm2Decrypt(String encryptStr, String privateKey) {
        SM2 sm2 = new SM2(privateKey, null);
        return StrUtil.utf8Str(sm2.decrypt(encryptStr, KeyType.PrivateKey));
    }
}
```

请求头

参数	**是否必选**	**值**	**说明**
tradeCode	是	1003	交易编码：查询
datasetCode	是	HDSD.FYQD	费用清单数据集
requestId	是	请求唯一 ID	同就诊事件上传的含义一样
medOrgCode	是	统一社会信用代码	同就诊事件上传的含义一样
medHosCode	是	院区编码	同就诊事件上传的含义一样

platformCode	条件必选	区域平台归属单位的 18 位统一社会信用代码	区域平台归属单位的 18 位统一社会信用代码。（单家医疗机构接入时不可传值），通过区域平台接入方式请求时必传。
timestamp	是	请求时间戳	请求时间戳（13 位毫秒级，例：1735704000000）
signature	是	签名字符串	加签字符串（规则见加签机制）
cryptoInfo	是	SM4 秘钥	密钥数字信封（SM2 加密后的 SM4 秘钥，该值需要参与签名）

请求参数

参数	**是否必填**	**类型**	**说明**
tyshxydm	是	String	统一社会信用代码。同就诊事件信息上传的一样
yljgmc	是	String	医疗机构名称。同就诊事件信息上传的一样
yqdm	是	String	院区代码。同就诊事件信息上传的一样
yqmc	否	String	院区名称。同就诊事件信息上传的一样
sfzjlbdm	是	String	身份证件类别代码。同就诊事件信息上传的一样
sfzjhm	是	String	患者姓名。同就诊事件信息上传的一样
hzxm	是	String	患者姓名。同就诊事件信息上传的一样

ghlsh	是	String	挂号流水号。同就诊事件信息上传的一样
jfdbm	否	String	缴费单编码，关联订单待支付事件。字段为空时，返回当前号源全部缴费单信息。同就诊事件信息上传的一样
jzkh	否	String	院内就诊卡号，关联费用清单数据集，同就诊事件信息上传的一样
jzklx	否	String	院内就诊卡类型，关联费用清单数据集，具体以医疗医院实际卡类型为准。同就诊事件信息上传的一样

响应参数

参数	**类型**	**说明**
ydzflsh	String	移动支付流水号，对应浙江医保支付门户定点医药机构接口规范查询费用列表(ORG002)接口中的 org_trace_no 字段，HIS 系统跟踪号(唯一)，医保支付时医保中心会调用 ORG002 接口进行费用列表查询。
ynybjdkh	String	院内医保建档卡号：对应浙江医保支付门户定点医药机构接口规范 ORG001 接口返回的 his_hust_id（患者院内证件号）字段值一致。
ynybklx	String	院内医保建档卡号：对应浙江医保支付门户定点医药机构接口规范 ORG001 接口返回的 his_hust_id（患者院内证件号）字段值一致。

fyqdlb	Array	费用清单列表
-fydlmc	String	费用大类名称
-fydldm	String	费用大类代码
-fydlje	Int	费用大类金额，本类收费项目各金额的总计，计量单位为分
-fyqd	Array	费用清单，明细数据
--xmmc	String	收费项目名称
--xmdm	String	收费项目代码
--xmsl	Int	收费项目数量
--xmdw	String	收费项目单位
--xmdj	Int	收费项目单价，计量单位为分
--xmzje	Int	收费项目总金额，计量单位为分
--sjxmdm	String	事件项目代码，关联费用清单数据集；关联检验支付完成事件、检验取号完成事件、检验完成事件、报告产生事件）
--zfzt	String	支付完成状态。0.未支付 1.已支付 2.已退费
--zflx	String	支付渠道类型。1.自费 2.医保

注意，以上所有字段信息就是就诊事件上传来的信息，经过安诊儿应用平台组装业务请求参数，向医疗机构实现的接口发起请求，以获取费用清单结果。

5.3.3 请求示例

5.3.3.1 请求业务参数明文

```
{
    "tyshxydm": "aliqua",
    "yqdm": "est pariatur deserunt",
    "yljgmc": "Excepteur",
    "yqmc": "Ut Excepteur",
    "sfzjlbdm": "sed ut",
    "sfzjhm": "Excepteur proident amet ut",
    "hzxm": "esse ut proident",
    "ghlsh": "commodo nisi",
    "jfdbm": "tempor minim esse",
    "jzkh": "dolor laborum Duis id irure",
    "jzklx": "amet voluptate"
}
```

5.3.3.2 请求业务参数密文 SM4 解密

通过 SM2 私钥解密 cryptoInfo 信息拿到 SM4 密钥，使用 SM4 密钥对请求参数解密；

5.3.3.3 实际请求参数

```
{
  "requestBiz":"请求业务参数密文 encoded"
}
```

5.3.4 响应示例

5.3.4.1 响应业务参数明文


```
{
    "ydzflsh": "asdyuqjnx",
    "ynybjdkh": "number",
    "ynybklx": "type",
    "fyqdlb": [
        {
            "fydlmc": "nostrud",
            "fydldm": "et nulla adipisicing est eiusmod",
            "fydlje": "aliquip ullamco deserunt",
            "fyqd": [
                {
                    "xmmc": "velit quis elit",
                    "xmdm": "nulla non ad",
                    "xmsl": "exercitation fugiat commodo enim consequat",
                    "xmdw": "do",
                    "xmdj": "adipisicing",
                    "xmzje": "ut in",
                    "sjxmdm": "deserunt amet",
                    "zfzt": "magna Lorem",
                    "zflx": "qui occaecat proident consequat incididunt"
                }
```

```
        ]
    },
    {
        "fydlmc": "nulla ullamco irure ad veniam",
        "fydldm": "in amet ipsum in aliquip",
        "fydlje": "Lorem culpa",
        "fyqd": [
            {
                "xmmc": "laborum",
                "xmdm": "in dolor adipisicing Ut Duis",
                "xmsl": "ipsum Ut reprehenderit aliquip pariatur",
                "xmdw": "ipsum dolore nulla eu magna",
                "xmdj": "est ad in",
                "xmzje": "velit id",
                "sjxmdm": "adipisicing officia ullamco cupidatat",
                "zfzt": "labore velit",
                "zflx": "dolore"
            },
            {
                "xmmc": "dolor nostrud quis",
                "xmdm": "nostrud",
                "xmsl": "laboris labore voluptate mollit dolore",
```

```
                "xmdw": "dolore aliqua commodo",
                "xmdj": "veniam",
                "xmzje": "consectetur dolor reprehenderit",
                "sjxmdm": "non minim veniam",
                "zfzt": "adipisicing labore in",
                "zflx": "est"
            }
        ]
    }
]
}
```


5. 3. 4. 2　响应业务参数密文 SM4 解密

通过 SM2 私钥解密 cryptoInfo 信息拿到 SM4 密钥，使用 SM4 密钥对响应结果加密；

5.3.4.3 实际响应参数


```
{
    "success": Boolean,
    "errCode": "String",
    "errMsg": "String",
    "responseBiz": "响应业务参数密文",
    "requestId":"String",
    "traceId":"String"
}
```

5.3.4.4 注意事项

上述请求示例中的请求参数是调用方按照接口规定，使用各医院对应的加密密钥。按照加密算法，组装请求参数对服务提供方发起请求，服务提供方在接受到请求之后，需要对请求参数 进行 SM4 解密，得到请求的明文参数。根据明文参数组装费用清单的查询业务。将查询的结果组装成响应明文格式，之后进行 SM4 加密。组装成实际响应参数格式返回。

5.4 检查预约

服务由医疗卫生机构提供，包含待预约项目查询、检查项目预约、预约记录查询等接口，支持用户在线自主预约医疗机构开具并完成支付的检查项目。

5.5 在线缴费

服务由医疗卫生机构提供，包含费用明细查询、费用预结算、创建支付订单、取消费用预结算、支付结果轮询、支付结果回调、费用结算、查询费用记录等接口，支持用户完成门急诊等诊疗费用在线结算。

5.6 报告查询

服务由医疗卫生机构提供，包含报告列表查询接口和报告详情查询接口（检查、检验等），支持用户实时在线查询院内出具的检验检查报告。