IMI 系统第三方平台使用说明文档

版本：V 1.0.2

2017 年 7 月

文档修订记录

版本编号	*变化状态	变更内容和范围	变更日期	变更人	批准日期	批准人
1.0.0	A	用户信息第三方使用说明文档初稿	2017/07/05	胡靖宇		
1.0.1	M	减少第三方客户端的安全问题，把相关逻辑移到第三方服务器做处理	2017/07/12	胡靖宇		
1.0.2	A	增加第三方服务器和客户端的 SDK 描述和说明	2017/07/18	胡靖宇		

*变化状态：A——增加，M——修改，D——删除，N——正式发布

目录

1.本文阅读对象

第三方系统（政务，民生等）集成 IMI 身份认证涉及的技术架构师，研发工程师，测试工程师，系统运维工程师。

注意：

- IMI 系统针对第三方平台现在只开放了身份认证平台的实名认证用户的信息获取/使用功能，更多的开放能力还在整理之中
- 文档中的伪代码描述和 SDK 说明性内容，是为了让本文的阅读对象能理解整体设计思想，具体 SDK 使用方法和定义，以正式发布的 SDK 文档和工具包为准

2.术语和约定

第三方：集成 IMI 身份认证功能的公司或机构。

登录：使用 IMI 登录，第三方可获得用户基本信息，包含用户名、手机号和 vportId。

授权：使用 IMI 授权，第三方可同时获得用户基本信息和用户实名认证（身份）信息，实名认证（身份）信息包含身份证信息等。

签名：指非对称加密里面的私钥签名，用于验证身份。

验签：指验证签名有效性。

Keystore：非对称加密算法中的私钥保存方法，一般保存在第三方服务器。

HTTPS：HTTPS（全称：Hyper Text Transfer Protocol over Secure Socket Layer），是以安全为目标的 HTTP 通道，用于安全的 HTTP 数据传输。

APP：特指第三方 APP。

SDK：特指 IMI SDK，本文对应的 IMI SDK 版本是，Android SDK 2.1.0，iOS SDK 2.1.0。

回调函数：由第三方来实现并提供给 SDK 函数内部使用的函数和方法，比如我们常用的 Listener 设计模式。

Mapping server：部署在 IMI 服务器上，用于网页和手机 APP 之间、不同手机之间或不同 APP 之间进行数据交换。

Topic id：网页和手机 APP 之间、不同手机之间或 APP 使用 Mapping server 交换数据时，临时建立的数据通信通道标识。

url schemes：网页和 Android 或 iOS 应用之间约定的一种 url，主要用于手机网页直接调起手机 APP。

第三方服务器：指需要集成 IMI 身份认证功能的 APP

或网站所对应的后台服务器。

IMI 跳转网页：为了在微信公众号打开 IMI 登录或授权界面做的一个跳转网页。

3.项目背景

随着互联网成为各行各业的基础设施，网上海量的数据成为金矿。如何保证数据的真实和安全，并让数据产生价值，推动了区块链技术的发展，区块链也被认为是互联网新时代的“风口”。

《“十三五”国家信息化规划》首次把区块链技术列入国家信息化规划中，并将其放在提升中国政府治理能力、推动经济转型升级的战略地位上。

佛山市禅城区政府始终拥抱信息化发展这样一个大趋势，从“一门式”到大数据，再到区块链，禅城历经数年之功，积累了海量数据。

在这过程中，禅城也发现，如果“一门式”要进化到“0 门式”，真正让数据跑腿，不得不解决身份信息等数据的真实性和安全性问题。

站在风口，顺势而为，以 IMI 身份认证平台为代表的区块链技术直击痛点，它以区块链技术作为基础平台，结合个人数字空间和大数据平台，保证了所有数据都和真实身份相关联，对数据的存储和使用做到

可信和可靠，并可以按照数据拥有者的要求做到数据的开放和共享。

简单得来说，IMI 区块链系统不光使用数据来证明“我是我”,而且还能使用数据来完善“我是我”，最后还能使用技术实现“我的数据我做主”。

IMI 平台除了针对政务、民生提高服务能力的同时，也给各行各业相关产业信息化发展带来同步推动和提高。通过 IMI 平台，我们所实现的政务和民生的“0 门式”工作模式，同样也可以同步推广到社会发展的各行各业,让社会上的各行各业也同步实现“0 门式”工作模式。

IMI 区块链平台是个开放的平台，平台上每个实名认证的用户可以是自然人，也可以是法人。作为各个行业和产业链的法人实体，他们可以充分利用 IMI 区块链平台来实现下面应用场景：

1. 自己的身份和数据，通过政府/权威机构背书，可以通过平台得到互相确认，实现自己数据的“0 门式”工作模式。
2. 在获取用户授权的情况下，可以直接获取用户的信息和数字资产，及各种可信数据。

   这样可以直接使用互联网方式解决用户的身份和数据的确认问题，实现用户数据的“0 门式”

工作模式。

3. IMI 区块链平台是一个开放的平台，权威机构和产业联盟可以加入并使用 IMI 平台来建立自己的产业链服务，实现产业各个利益方和使用者的“0 门式”工作模式。

比如食品安全管理，产业供应链管理，金融投资服务等等。

4. IMI 系统介绍

IMI 系统本质上是一个数字身份和数字空间平台，它的技术目标是建立一个 - **我是我，隐私和信任共存** - 这样特性的个人数字身份和个人数字空间：

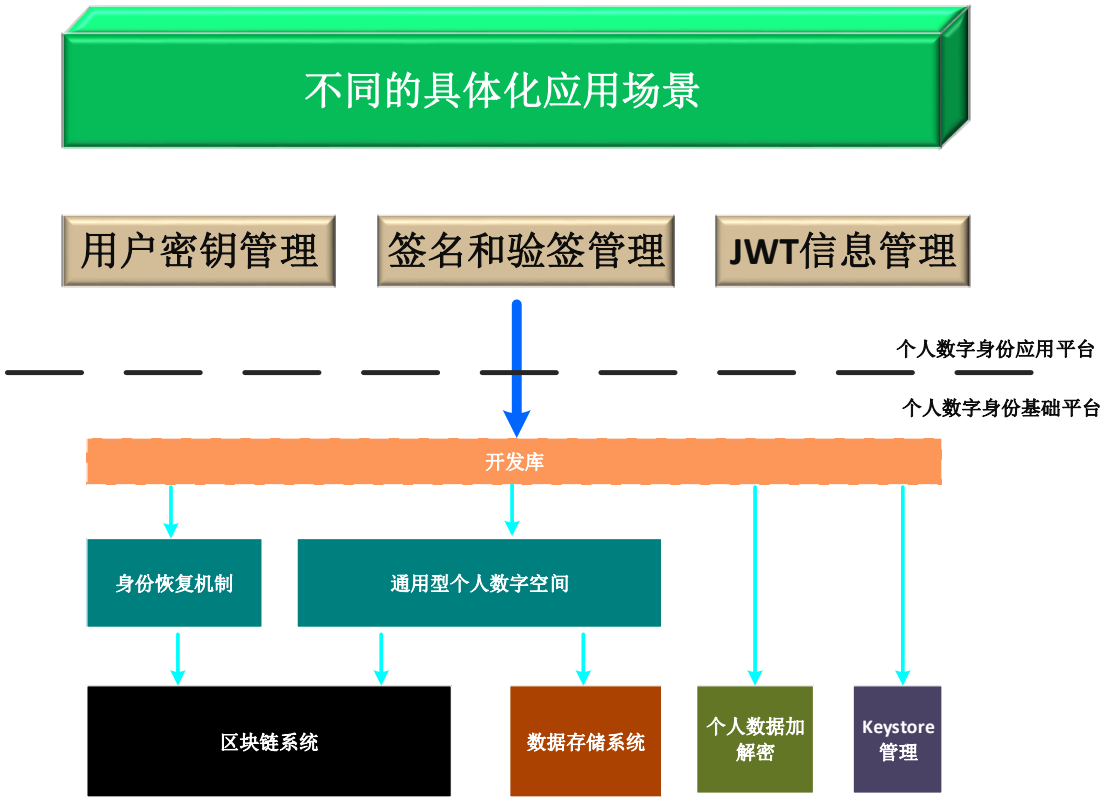
- 数字身份控制权可恢复
  - 个人使用区块链私钥的方式获取数字身份的控制权
  - 个人通过数字身份恢复机制可以更新私钥，并以新的私钥重新获取数字身份的控制权
- 建立可信任个人数字空间
  - 建立本人声明和他人认证数字身份相关的个人数据，使用区块链算法保证数据的可信任性
  - 只有本人可以在自己的个人数字空间保存相

关的个人数据

- 本人对个人的隐私数据进行加密

- 我的数据我做主
  - 本人对个人的隐私数据进行解密
  - 查看数字身份相关的个人数据需要得到本人授权

为了建立这样一个平台，我们需要实现下面这样一个基础架构系统来实现我们的技术目标：



所以整个系统的建设需要实现下面几个关键功能组件的整合：

1. 手机端应用

为了覆盖最大范围内的手机用户，我们需要在iOS/Android 二个手机系统上实现我们的应用。

2. 实名认证应用系统

为了对 IMI 系统的数字身份进行实名认证，我们需要为权威机构（政府部门）实现一套实名认证应用系统，允许权威机构的工作人员通过柜台应用系统完成用户的实名认证过程；允许权威机构的工作人员通过柜台应用系统完成用户的APP 身份恢复过程。

3. 区块链系统

建立一个区块链系统，并能基于这个区块链系统完成一套 - **我是我，隐私和信任共存** - 个人数字身份和个人数字空间的管理系统。

4. 数据存储系统

为了让数据存储逻辑和区块链系统完美组合，达到区块链数据去中心化、不可篡改、可追溯、开放共享的目标，我们需要建立一个符合并具有区块链特性的数据存储系统。

5. 数字身份/数字空间管理系统

为了建设一个数字身份可控制和可管理的完整的个人数字身份和个人数字空间，需要我们基于区块链和数据存储系统之上建立起完整的数字

身份和数字空间管理系统：

- 身份恢复机制
- 通用型的个人数字空间
- 个人数据加解密机制
- Keystore 管理机制

6.开发库

我们的 IMI 系统是一个开放的平台，需要提供 SDK/开发库允许第三方应用和平台来使用、建设和完善数字身份和数字空间。

当前我们对于第三方应用平台的开发支持如下：

- Java 后端平台（JDK1.7 及以上）
- PHP 后端开发（PHP5.6 及以上）
- Javascript 前端开发
- Android 应用开发
- iOS 应用开发

5.用户个人数据的独特性

IMI 系统为了实现用户对于数据的自主控制权 -“我的数据我做主”，用户的个人数据具有下面的存在特殊性：

- 个人数据是以用户独一无二的加密方式进行数据存储

- 服务器无法获取用户的个人数据
- 用户只有通过 IMI APP 才能对个人数据进行解密操作

个人数据的存在特殊性，决定了用户的个人数据只能由用户通过 IMI APP 才能获取，也决定了用户的个人数据只能通过 IMI APP 才能进行数据分享。

那么我们就需要实现一种技术方式，能让第三方应用或者第三方平台和 IMI APP 之间简单、方便得进行数据分享操作，这种技术实现方式我们称作为 Mapping Server 数据通讯方式。

6. Mapping Server 的设计思想

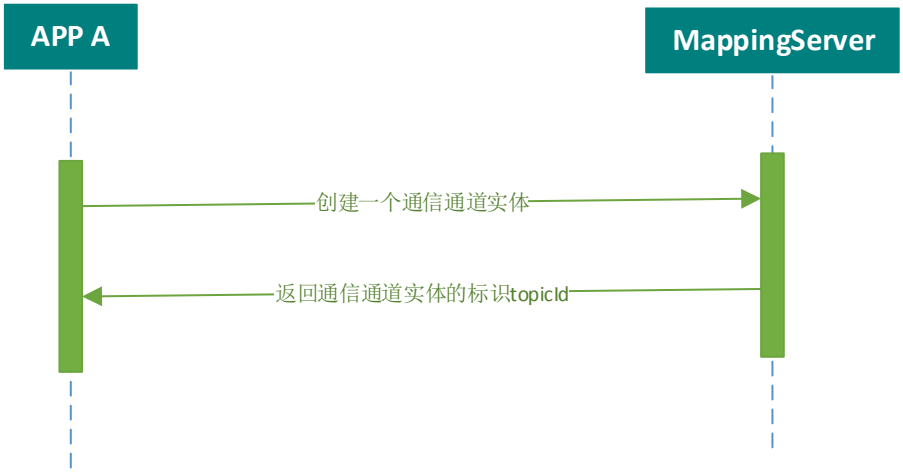
我们设计了 Mapping Server 是为了让 APP 和 APP，APP 和第三方网站之间，能借助于 HTTPS 协议建立起数据通信通道，为二者之间的数据提供一种基于网络传输的数据交互能力。

我们通过 Mapping Server 建立起来的数据交互能力不要求 APP 和 APP， APP 和第三方网站之间直接建立网络通道，并解决了以下一些问题：

1. 避免了端到端之间直接建立网络连接时可能存在的网络条件限制，比如 APP 双方可能都在防火墙之内，导致双方 APP 无法建立端到端网络连接
2. 保护了 APP 双方设备的安全性和隐私性

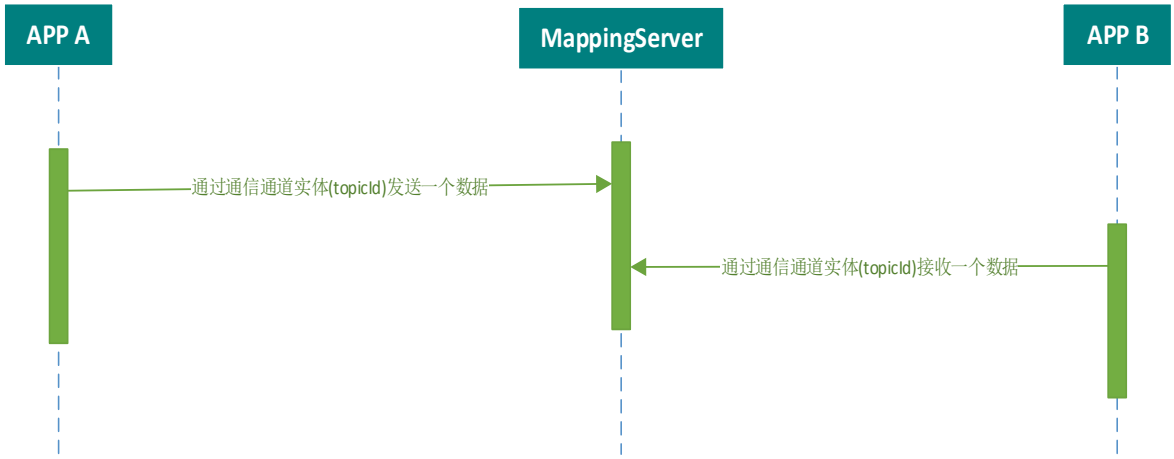
一般的使用流程如下：

1. 创建一个数据通信通道的实体：



2. APP A 可以通过某种数据共享的方式（比如，二维码扫描方式，或者 iOS/Android 启动应用传参方式，或者 H5 网页传参方式）传递通信通道实体标识 topicId(包括一些附加参数)给业务逻辑的另一方。

3. 通过通信通道实体进行数据交互：



基于这个数据交互的能力，APP 和 APP 之间，APP 和第三方网站之间，可以按照业务逻辑来定义双方的交互数据和交互流程，以完成相关的业务逻辑。

7. APP ID 的设计思想

第三方 APP 在使用 IMI 系统功能（SDK）的时候，对于 IMI 系统来说其实存在着一个第三方 APP 安全身份管理的问题。

IMI 系统有责任和义务来确认第三方 APP 的身份安全性并可以对第三方 APP 的身份进行有效的管理，以便保护 IMI APP 用户的个人数据和隐私问题。

为了解决这个问题，我们为每个第三方应用或者第三方平台设计了一个数字身份，我们称之为 APP ID。

IMI 系统本质是以数字身份为载体的数据空间和数据平台，IMI 系统为数字身份和数字空间的控制和管

理已经提供了系统化的解决方案。通过数字身份，第三方 APP 可以系统化得获取访问控制和数据管理的能力。

IMI 数字身份 ID 在技术文档内部被统称为 vportId，它用于标识自然人和法人（包括法人的附属组织机构）所对应的数字身份和数字空间。APP ID 也是一个 vportId，但是它从附属关系上来说，是从属于一个指定的法人数字身份 ID。法人可以通过这个 APP ID 来对这个 APP 的数字身份控制权和数字空间进行有效的管理。

如何对 APP 的数字身份控制权和数字空间进行有效的管理？这一切都是通过数字身份所对应的私钥来进行访问控制的。

所以我们继续延伸到一个话题：如何可以安全有效得对服务器保存的私钥进行存储和管理。

8. Keystore 的设计思想

为了密钥的安全，系统一般都是对密钥进行再次加密后存储，所以就有了 Keystore 的概念。

用户可以针对一个 Keystore 设置一个对应的 password，Keystore 算法使用 password 对密钥内容

进行加密存储。同样，用户只有使用 password 才能从对应的 Keystore 中计算出对应的密钥内容。

为了防止黑客窃取 Keystore 后暴力破解用户保存在 Keystore 里面的私钥内容，我们在 Keystore 加密算法里面使用了 scrypt 算法。scrypt 不仅计算所需时间长，而且占用的内存也多，使得并行计算多个结果异常困难，因此利用 rainbow table 进行暴力攻击更加困难，这样可以极大地降低 Keystore 里面的私钥内容被暴力破解的可能性。

Keystore 的数据存储结构如下，仅做参考：


```
{
    "crypto" : {
        "cipher" : "aes-128-ctr",
        "cipherparams" : {
            "iv" : "83dbcc02d8ccb40e466191a123791e0e"
        },
        "ciphertext" :
"d172bf743a674da9cdad04534d56926ef8358534d458fffccd4e6ad2fbde479c",
        "kdf" : "scrypt",
        "kdfparams" : {
            "dklen" : 32,
            "n" : 262144,
            "r" : 1,
            "p" : 8,
            "salt" :
"ab0c7876052600dd703518d6fc3fe8984592145b591fc8fb5c6d43190334ba19"
        },
        "mac" :
"2103ac29920d71da29f15d75b4a16dbe95cfd7ff8faea1056c33131d846e3097"
    },
    "id" : "3198bc9c-6672-5ab3-d995-4942343ae5b6",
    "version" : 3
```

```
}
```

使用这样一种 Keystore 的密钥管理方式，可以保证黑客在无 password 的情况下,无法获取对应的密钥内容；或者说，无 password 的情况下要获取密钥内容是及其困难的、代价巨大的。

用户可以在启动服务的时候，以手工输入参数的方式把 password 传递给服务，由服务通过 password 从 Keystore 中获取对应的密钥内容。

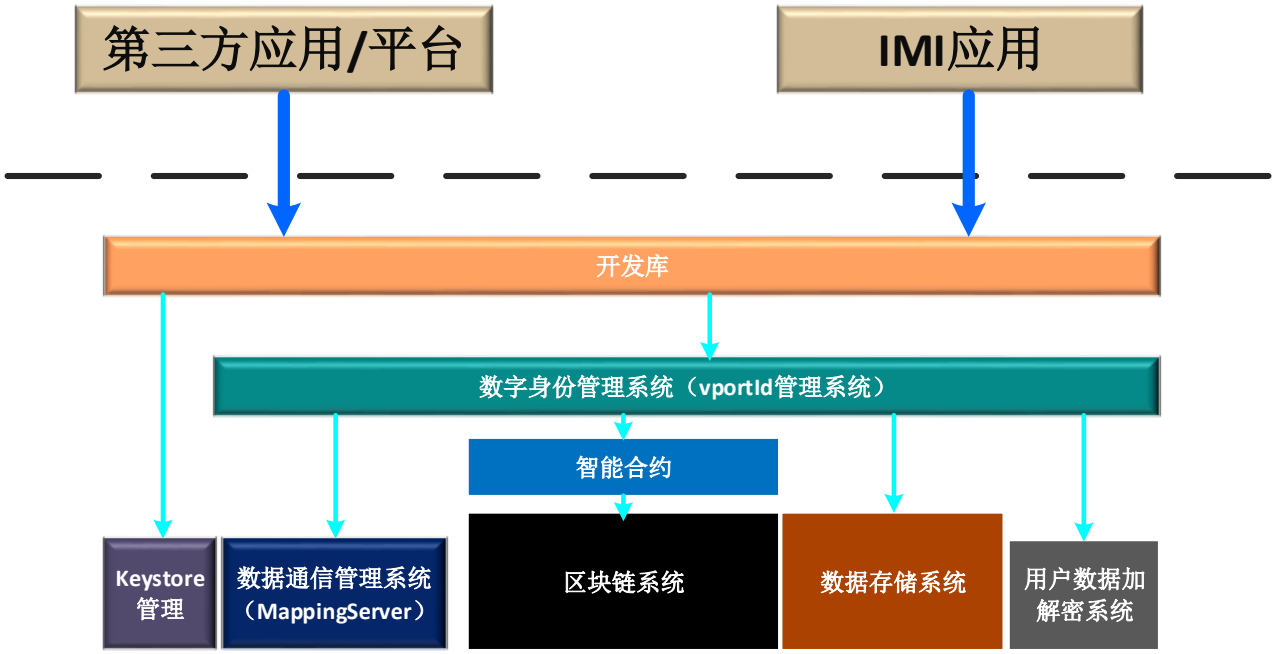
9.用户数据获取/使用的技术分析

通过上面的背景介绍，我们再来总结用户个人数据在获取和使用上，所采用的技术方式和技术特点：

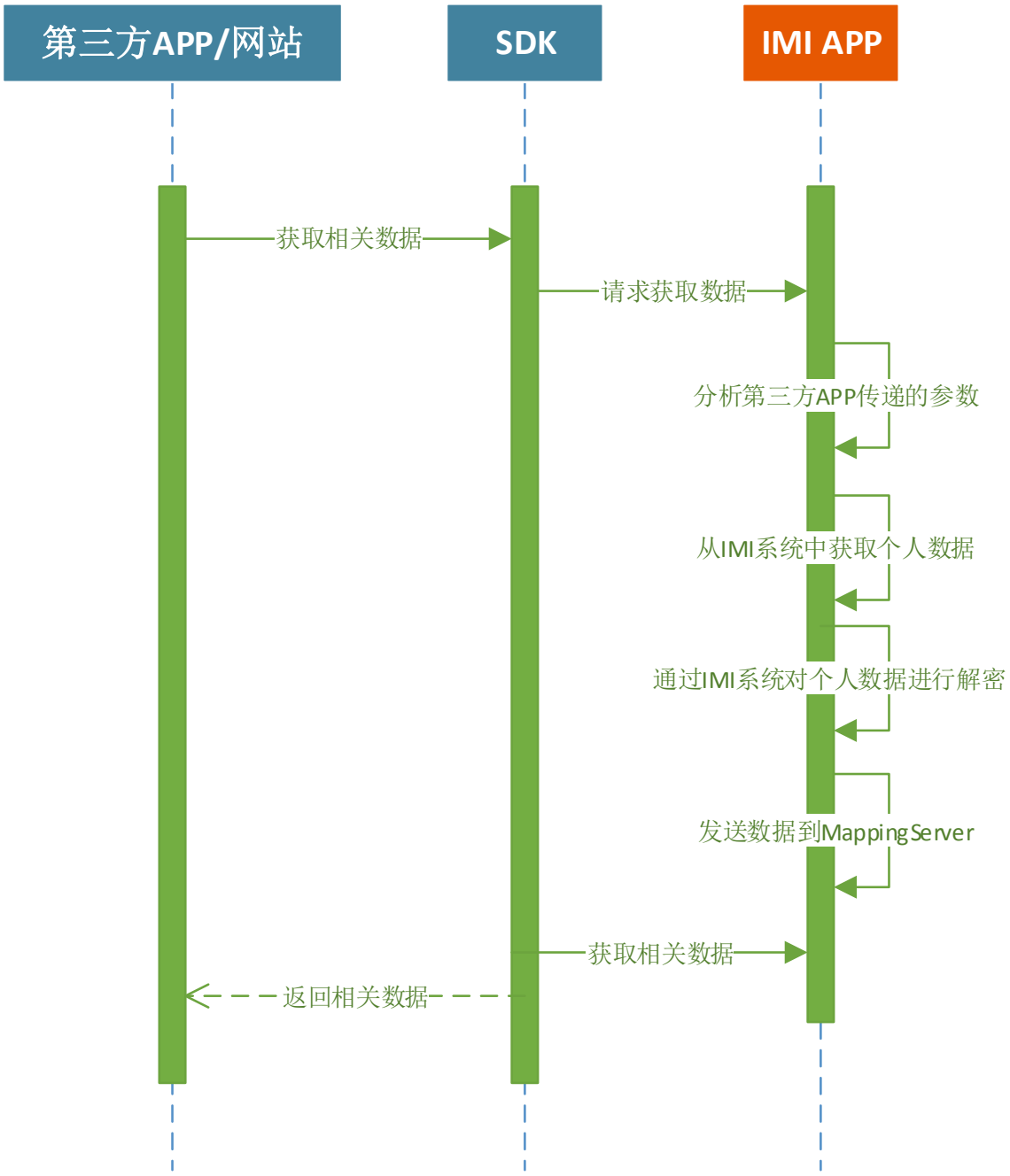
1. 第三方 APP 和第三方平台在 IMI 系统中是一个从属于指定法人的一个 APP 数字身份：
   APP 数字身份 vportId（APP ID）的存在必要性
2. APP 数字身份也是通过私钥的安全控制方式在 IMI 系统中进行访问和管理：
   Keystore 的存在必要性
3. 第三方 APP/网站和 IMI APP 是通过 Mapping Server 进行数据通信和数据传输的：
   Mapping Server 的存在必要性

4. 所有 APP、网页和服务器之间的点对点通讯都是使用 HTTPS 数据传输方式来保证数据的安全性：数据隐私性的保证

我们通过一张图来表示用户数据获取/使用功能相关的整体架构（使用 3D 图标表示直接关联部分）：



数据的获取过程简单介绍如下：



10. 对第三方平台的技术要求

第三方 APP/平台要能对接 IMI 系统的功能，需要具有下面 4 个基本的技术能力：

1. 第三方对于其网站和 APP 需要有对应的后台服务器做支撑
2. 第三方 APP/服务器在技术上有能力作为 HTTPS 的客户端使用 HTTPS 通讯方式和 IMI 服务器完成安全网络通讯
3. IMI 技术团队提供 Java/PHP 的服务器端 SDK,用于在后台服务器上完成相关的服务和管理

   - Keystore 管理
   - 签名和验签
   - 用户数据通过 Mapping Server 接口访问来获取

注意：IMI 系统的服务器接口都需要使用用户的私钥做签名，才能访问

4. 第三方的后台服务器需要实现用户数据获取/使用服务,以提供给 JavaScript SDK 或 Native APP SDK 做数据展示使用

   - 获取 Mapping Server 通道标识参数(topicId, openId, scope, v)
   - 通过通道标识参数来获取用户的个人信息

注意：服务器不能直接把对应用户的全信息返回给网页/APP，以防止出现 IMI 用户隐私泄露问题

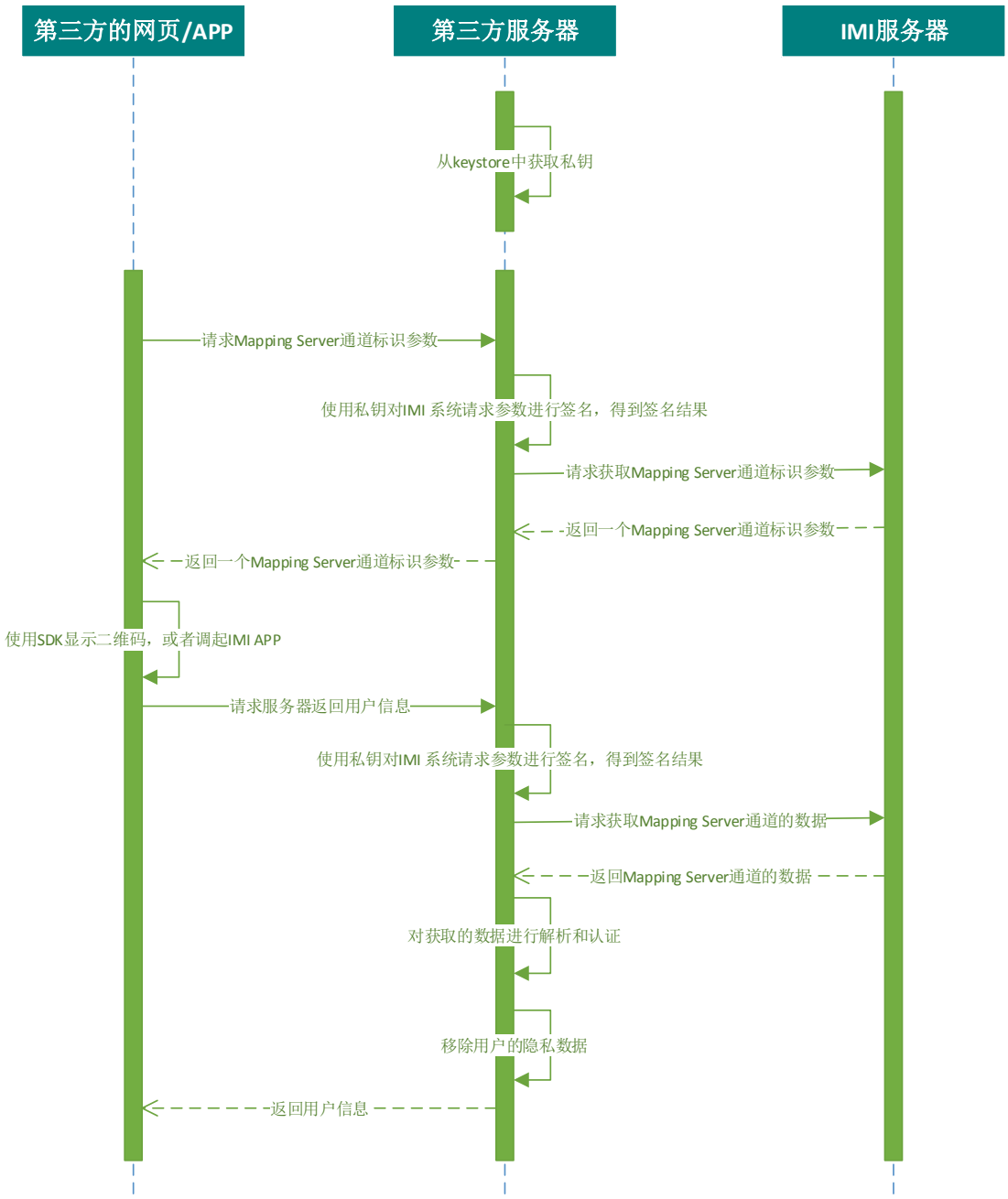
5. 第三方的网页/APP 使用 JavaScript SDK 或

Native APP SDK 完成应用场景的交互流程

- 通过服务器接口获取 Mapping Server 通道标识参数，并显示为二维码，或者调起 IMI APP

- 通过服务器接口获取用户信息，并根据应用场景需要显示用户的相关信息

比如，第三方服务器实现用户数据服务如下：



第三方服务器要能对接 IMI 系统的功能，需要具有下面的运行环境：

1. 如果服务器是 Java 运行环境，需要有 JDK1.7 及以上的版本环境
2. 如果服务器是 PHP 运行环境，需要有 PHP5.6 及以

上的版本环境

11. 第三方平台的应用场景

用户个人数据获取/使用的第三方应用场景有以下二大类：

1. 第三方网站通过浏览器 SDK 和 IMI APP 交互；交互方式如下：
   - IMI APP 扫描 PC 登录页面上的二维码信息
   - 手机网页调起 IMI APP

这里所获取的信息又分成二类信息：

①第三方网站获取用户基本信息

②第三方网站获取用户实名认证信息

2. 第三方 APP 通过 Native APP SDK 和 IMI APP 交互；交互方式如下：
   - 第三方 APP 扫描 IMI APP 上的二维码信息
   - 第三方 APP 调起 IMI APP

这里所获取的信息又分成二类信息：

①第三方 APP 获取用户基本信息

②第三方 APP 获取用户实名认证信息

11.1 第三方网站通过浏览器 JavaScript SDK 和 IMI APP 交互

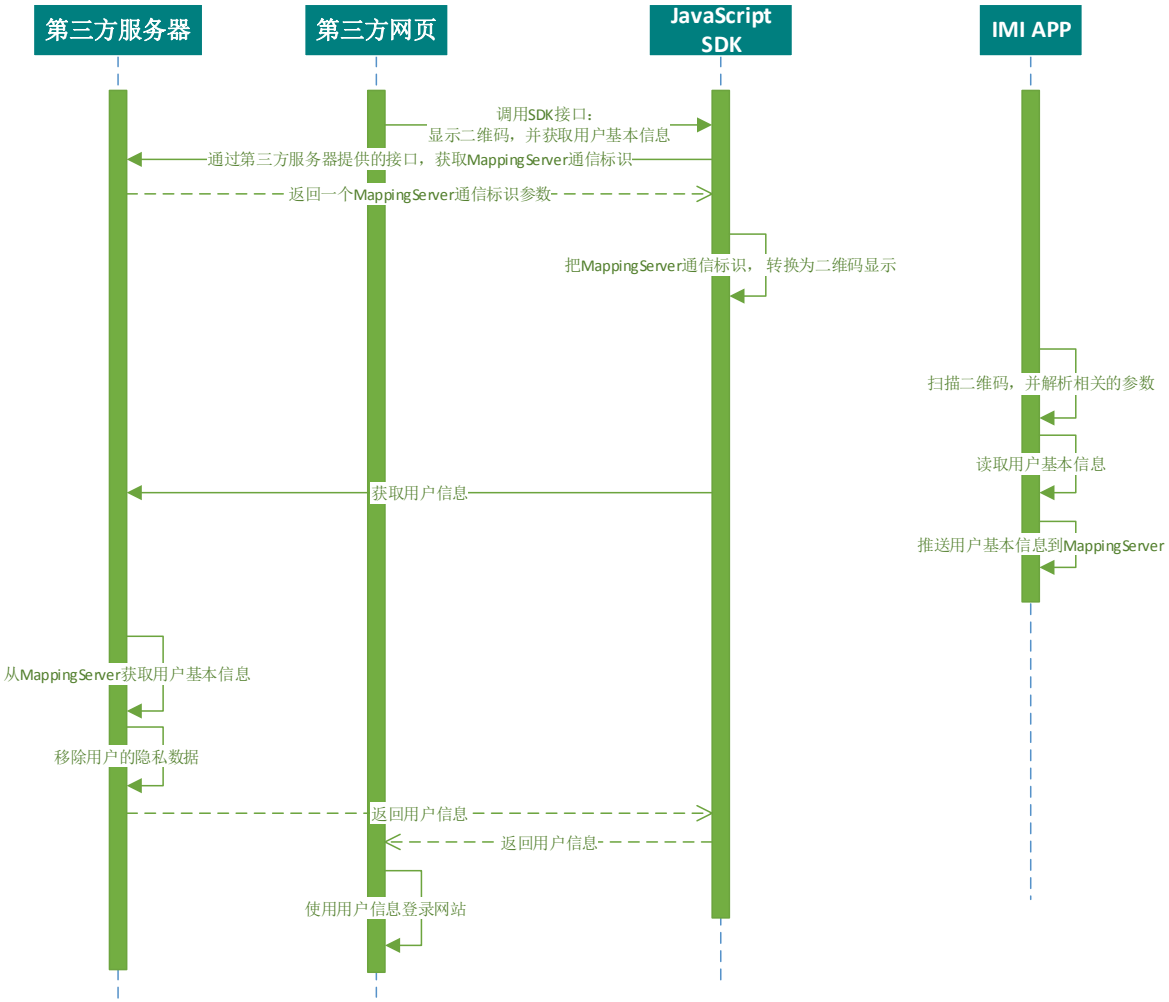
这里主要分为二大类应用场景：

1.交互双方属于不同的设备

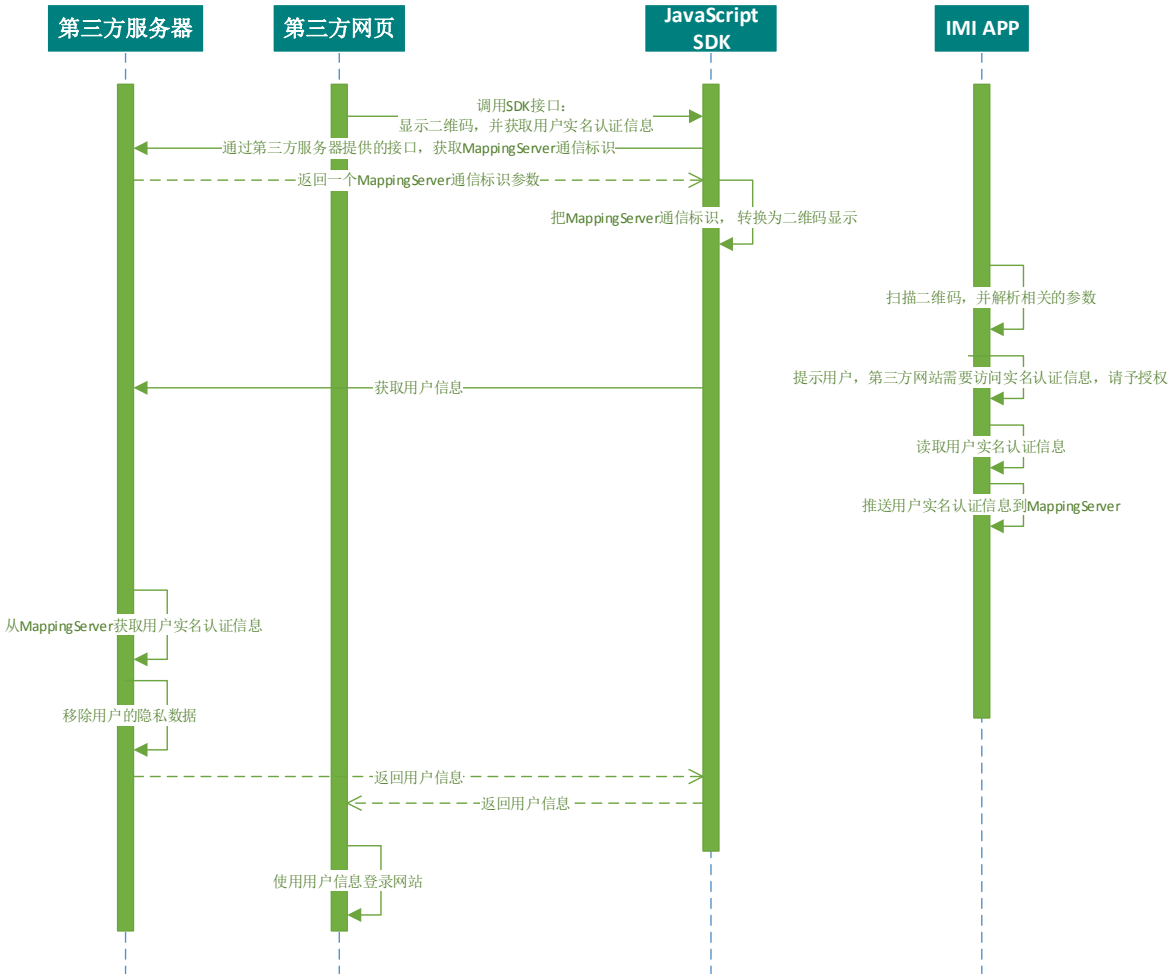
比如：PC <-> 手机

IMI APP 扫描 PC 登录页面上的二维码信息

①第三方网站获取用户基本信息



②第三方网站获取用户实名认证信息

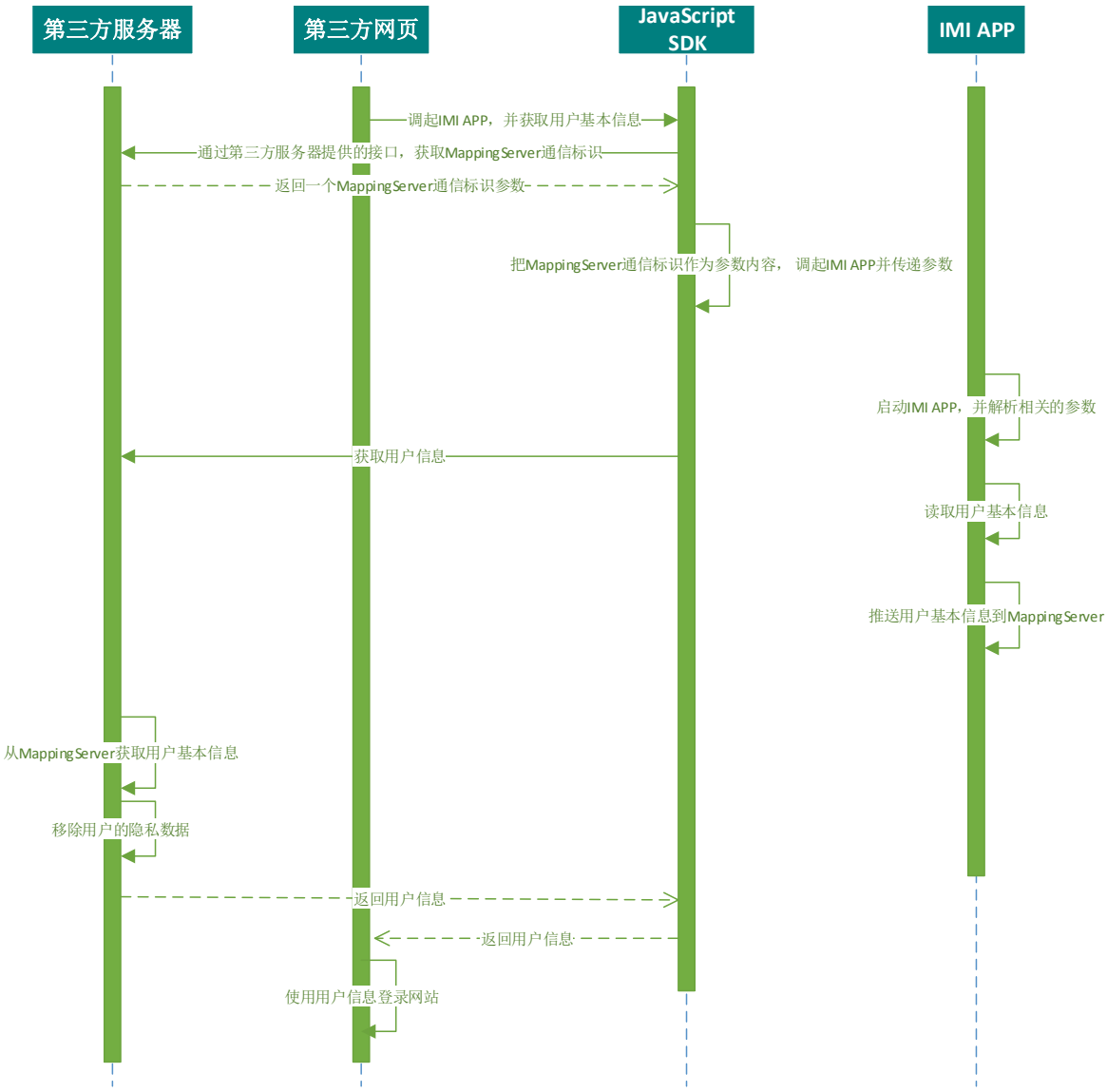


2. 交互双方属于同一个设备

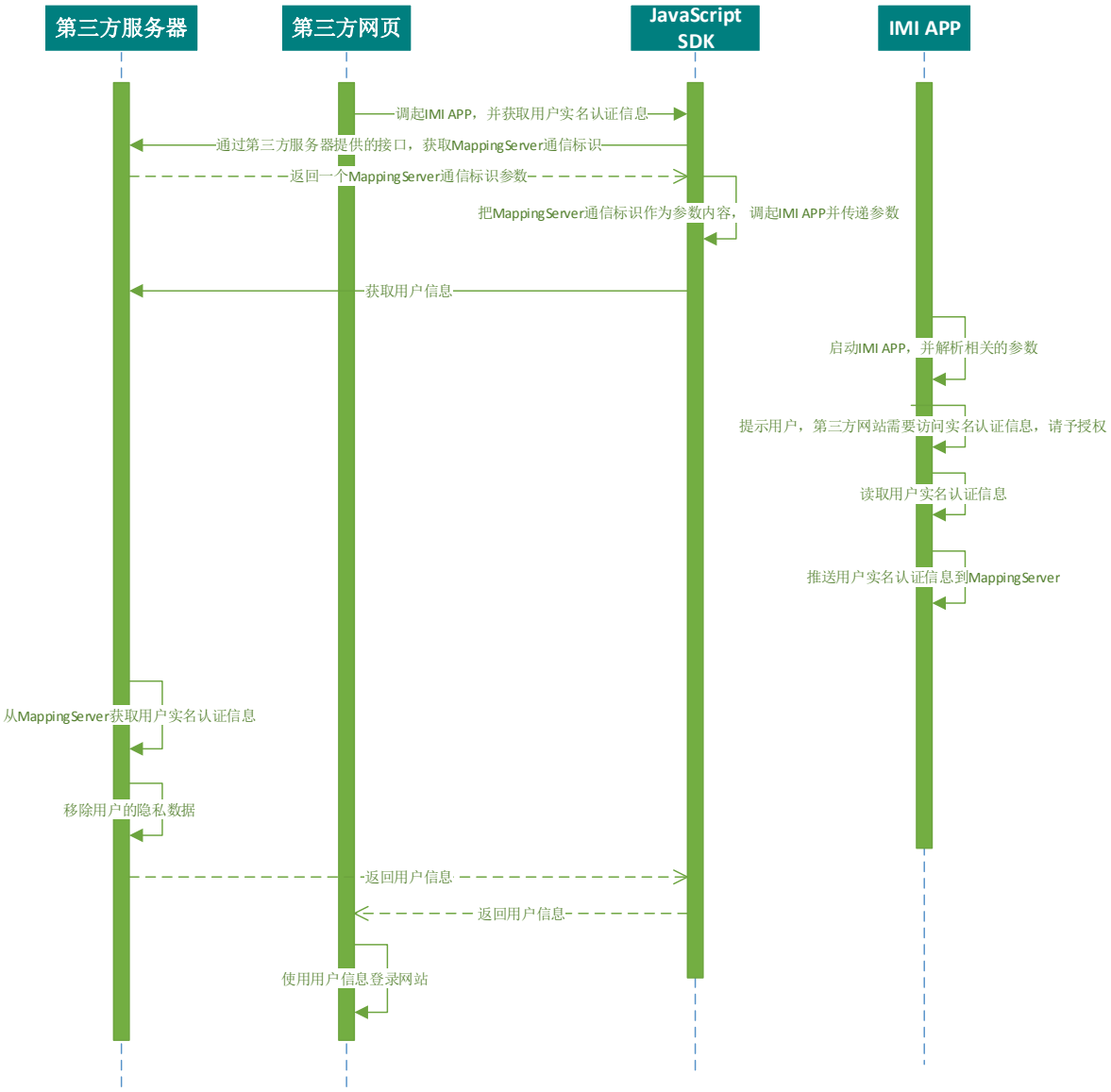
比如：手机 A <-> 手机 A

手机网页调起 IMI APP

①第三方网站获取用户基本信息



②第三方网站获取用户实名认证信息



11.2 第三方 APP 通过 Native SDK 和 IMI APP 交互

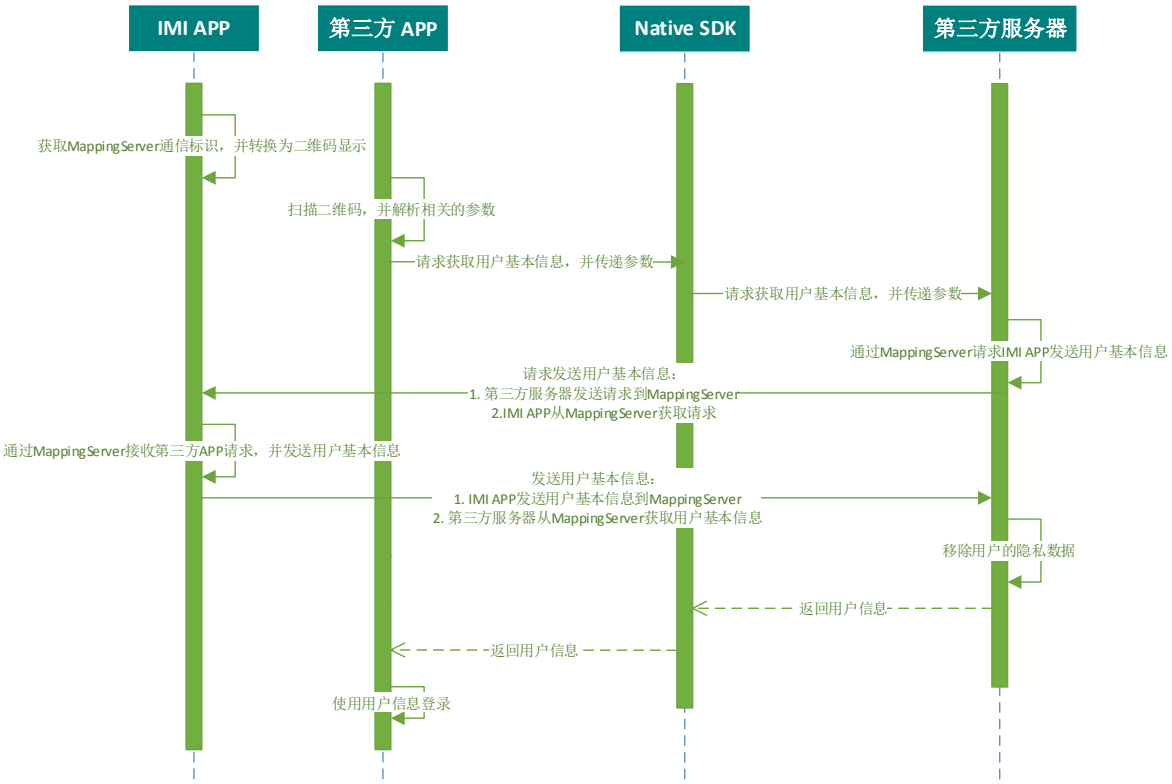
这里主要分为二大类应用场景：

1. 交互双方属于不同的设备

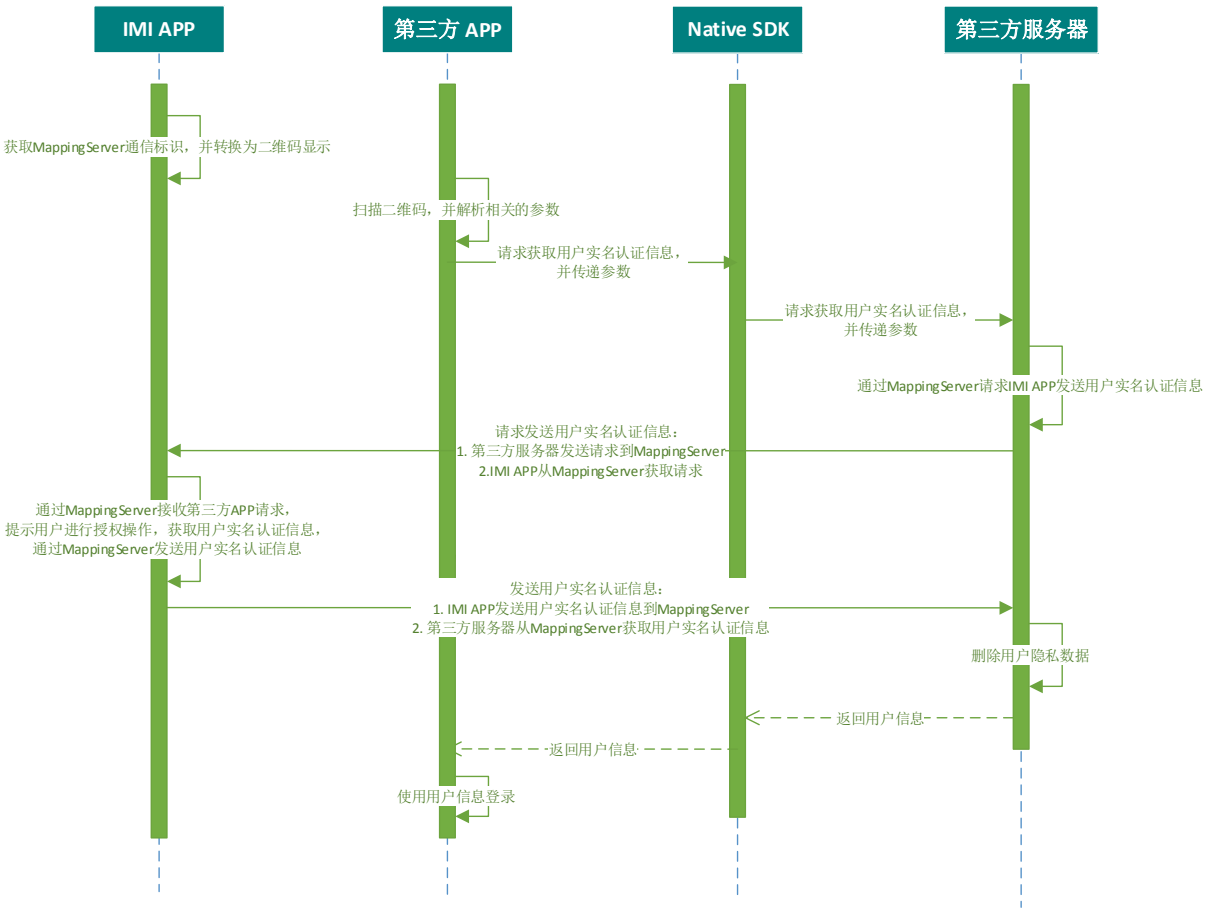
比如：手机 A <-> 手机 B

第三方 APP 扫描 IMI APP 上的二维码信息

①第三方 APP 获取用户基本信息



②第三方 APP 获取用户实名认证信息

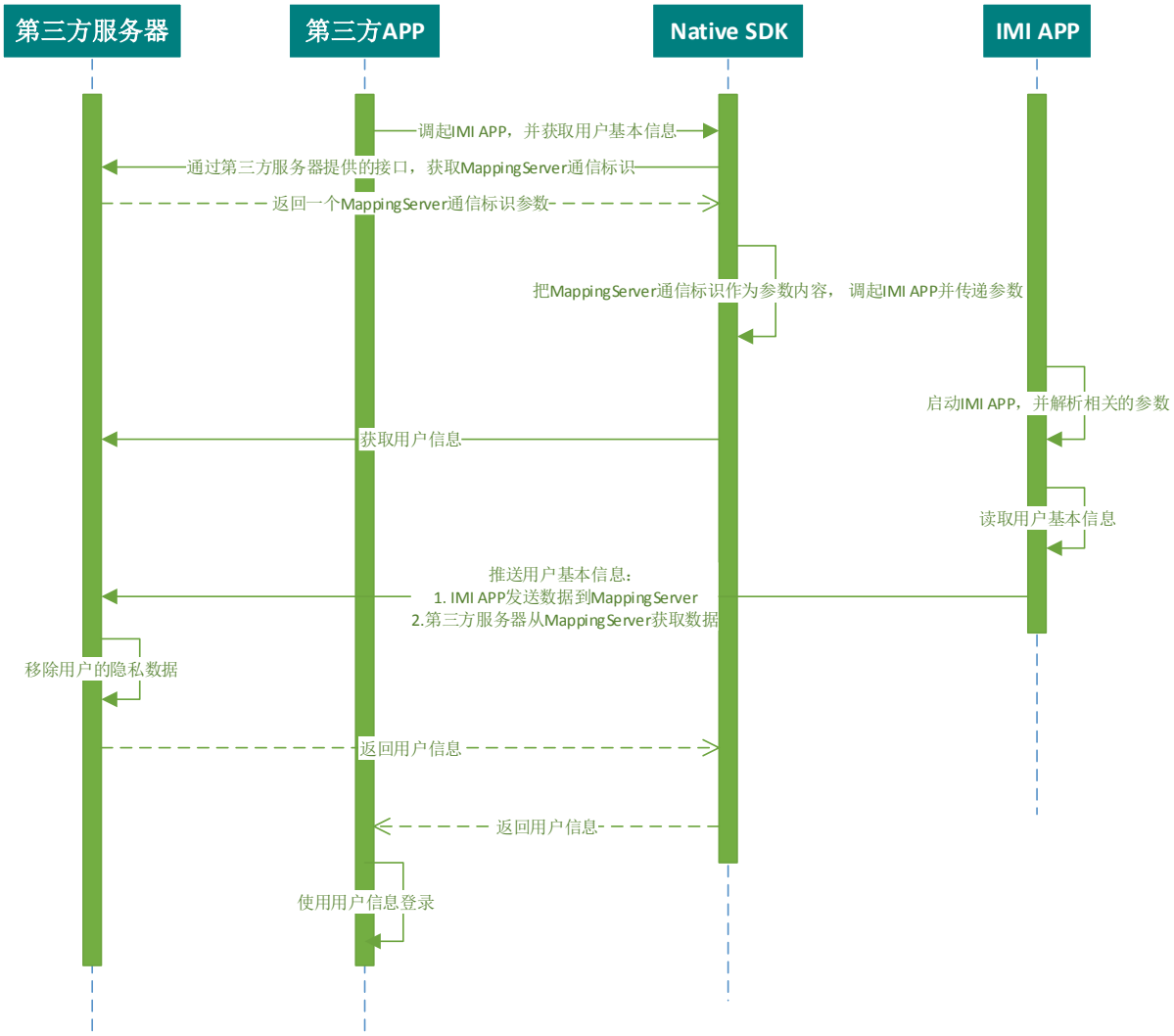


2.交互双方属于同一个设备

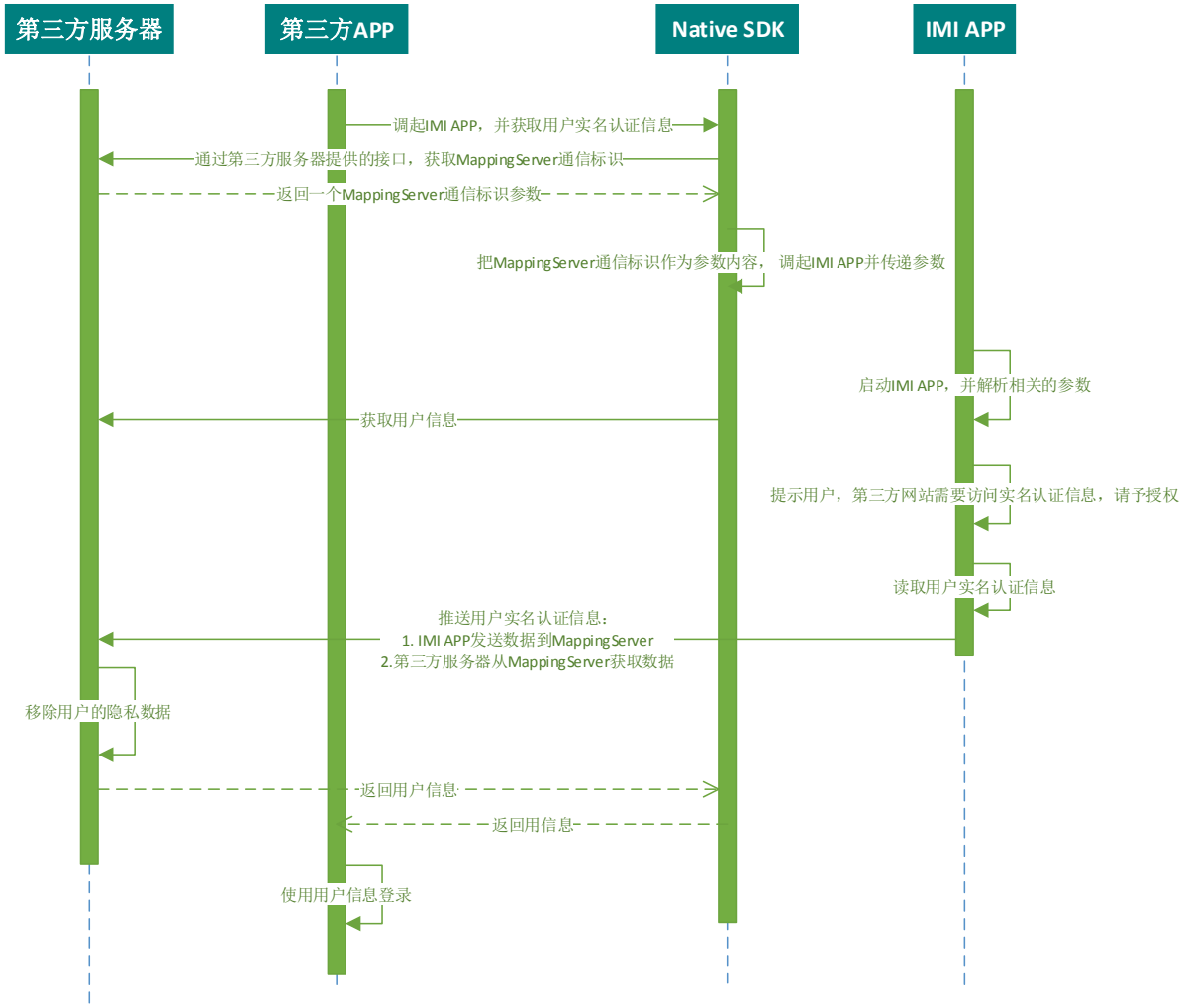
比如：手机 A <-> 手机 A

第三方 APP 调起 IMI APP

①第三方 APP 获取用户基本信息



②第三方 APP 获取用户实名认证信息



12. 第三方平台的开发流程

因为 IMI 系统当前还没有完成法人的身份认证功能的开发，所以第三方平台的 APP ID（应用的 vportId）无法通过系统申请来完成。

当前给第三方平台的开发流程是先对接，在对接中由 IMI 系统工程师为第三方平台申请对应的 APP ID（vportId）和 Keystore，第三方平台使用对应的 APP

ID、Keystore 及开发库来完成相关的功能对接工作：

1. 通过 Keystore 和对应的 password 可以获取第三方平台数字身份 APP ID 所关联的密钥
2. 通过 APP ID、密钥和 Java/PHP SDK 可以在服务器上实现 IMI 系统服务接口访问所需要的签名回调函数；这个回调函数会被 Java/PHP SDK 使用以便能完成服务器上的相关操作逻辑
3. 第三方后台服务器需要实现二个服务并以 HTTP 接口的方式开放给网页/APP 使用：
   - MappingServer 通信通道标识的获取服务
   - IMI 用户信息的获取服务
4. 需要第三方 APP/网页通过后台开放的二个服务在第三方 APP/网页代码里面实现二个回调函数，以便 APP SDK 或者 JavaScript SDK 能使用这二个回调函数来完成相关的应用场景

在 IMI 系统完成法人的身份认证功能后，第三方可以通过 IMI 系统申请对应的 APP ID 和 Keystore，然后在第三方服务器上用新的 APP ID 和 Keystore 替代原来的文件内容就可以继续使用，后续没有额外的开发工作。

13. 第三方平台 SDK 概述

从前面的介绍内容可以知道，第三方平台的 IMI 功能的开发工作量主要集中在第三方后台服务器，智信禅城 IMI 系统会提供相关的 demo 给第三方平台做参考。

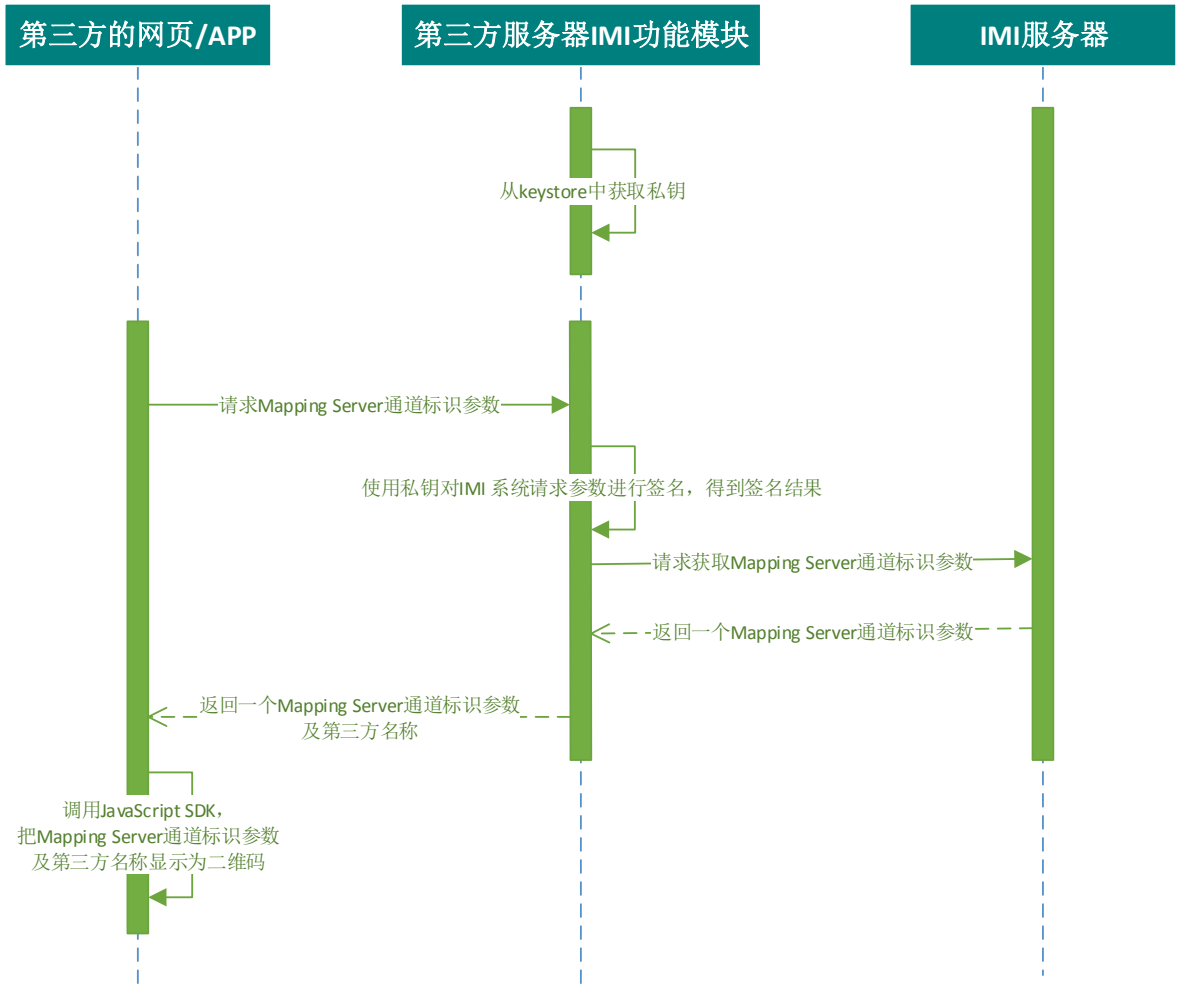
我们这里先来依次介绍一下第三方平台需要完成的功能和涉及的 SDK 接口定义。

注意：文档中的伪代码描述和 SDK 说明性内容，是为了让本文的阅读对象能理解整体设计思想，具体 SDK 使用方法和定义，以正式发布的 SDK 文档和工具包为准

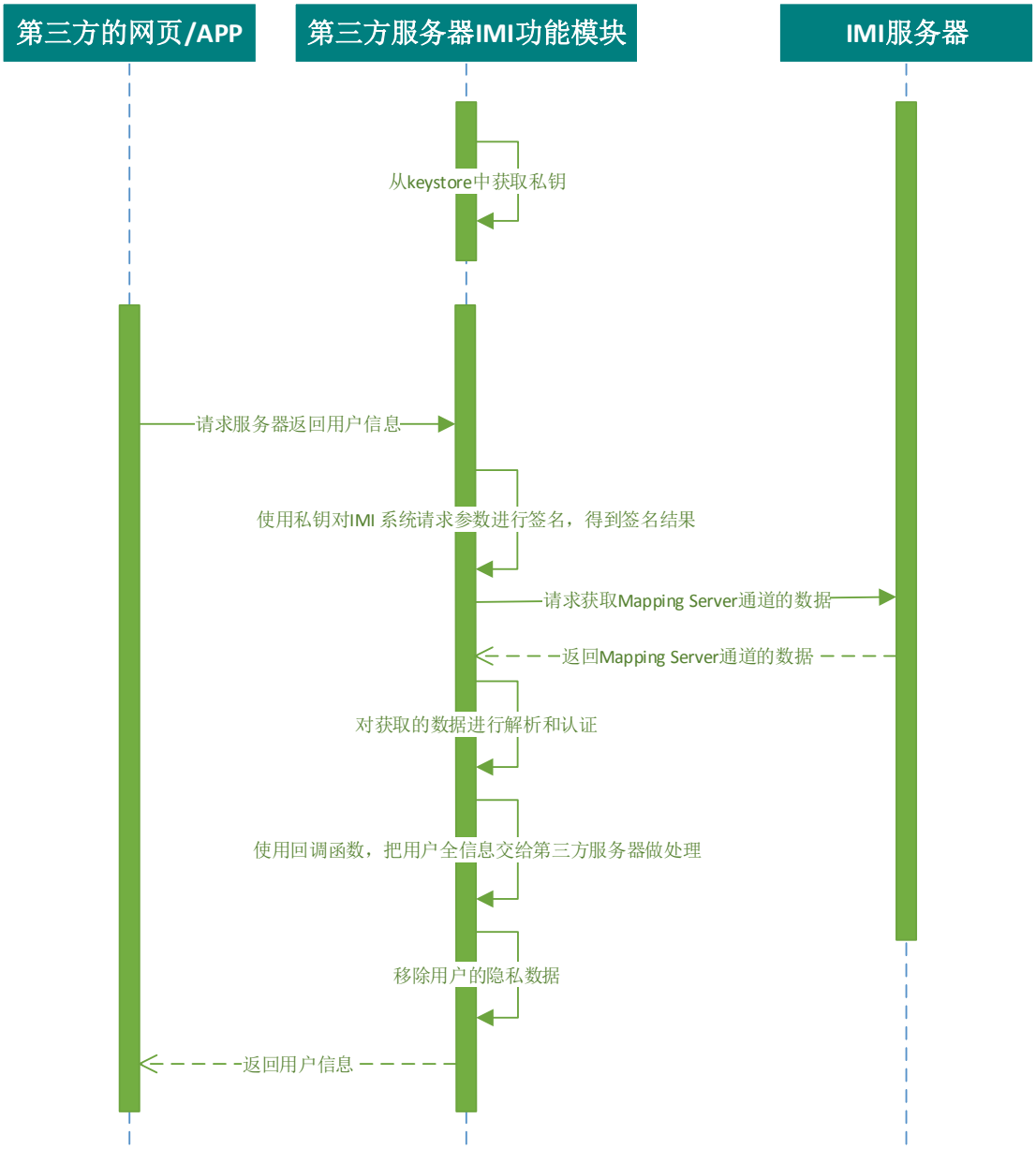
13.1 第三方服务器 SDK

第三方服务器为网页/APP 提供下面二个 HTTP 服务接口：

1. MappingServer 通信通道标识参数（及附加信息，现在是第三方名称）的获取服务



2. IMI 用户信息的获取服务



智信禅城 IMI 系统为第三方服务器提供了下列 SDK 函数（以 Java SDK 方式进行描述）：

13.3.1. SDK 里面需要用到的各种实体对象

● 数据通信代理对象

```
/**
 * @Title: 数据通信代理信息
 */
class PeerProxy {
    // 通过 MappingServer 通信代理和对方建立起通讯连接时，需要的相关代理信息
    private String openId;
    private String topicId;

    // 和 IMI APP 数据交互时，需要的相关代理信息
    private String[] scope;
    private String version;

    private PeerProxy(String openId, String topicId, String[] scope, String
version) {
        this.openId = openId;
        this.topicId = topicId;

        this.scope = scope;
        this.version = version;
    }

    public static PeerProxy create(String openId, String topicId, String[]
scope, String version) {
            return new PeerProxy(openId, topicId, scope, version);
    }

    public String getOpenId() {
        return openId;
    }

    public String getTopicId() {
        return topicId;
    }

    public String[] getScope() {
        return scope;
    }

    public String getVersion() {
        return version;
    }
}
```

● 数据签名者信息对象

```
class Issuer {
    private String name;
    private String vPortId;
    private String publicKey;
    private boolean hashed;

    private Issuer(String name, String vPortId, String publicKey, boolean
hashed) {
        this.name = name;
        this.vPortId = vPortId;
        this.publicKey = publicKey;
        this.hashed = hashed;
    }

    public static Issuer create(String name, String vPortId, String
publicKey, boolean hashed) {
        return new Issuer(name, vPortId, publicKey, hashed);
    }

    public String getName() {
        return name;
    }

    public String getVPortId() {
        return vPortId;
    }

    public String getPublicKey() {
        return publicKey;
    }

    public boolean getHashed() {
        return hashed;
    }
}
```

● 用户基本信息对象

```
/**
```

```
 * @Title: 登录用户信息，也被称作用户基本信息
 */
class LoginUserInfo {
    // 获取的登录用户信息的 JWT 串
    private String jwt;

    // 从 JWT 串中解析出来的登录用户的 vPortId，登录名字，手机号码
    private String vPortId;
    private String nickName;
    private String mobile;

    // 从 JWT 串中解析出来的签名者信息
    private Issuer issuer;

    public boolean setup(String jwt) {
        // 对 jwt 进行解析，如果解析失败，返回 false
        this.vPortId = 解析出来的登录用户的 vPortId;
        this.nickName = 解析出来的登录用户的昵称;
        this.mobile = 解析出来的登录用户的手机号码;

        验证签名是否正确;

        验证签名者 issuer 信息是否真实;

        if (任何的解析失败 || 验证签名错误 || 验证签名者信息错误) {
            this.vPortId = null;
            this.nickName = null;
            this.mobile = null;

            this.issuer = null;

            return false;
        }

        this.jwt = jwt;

        return true;
    }

    public String getJwt() {
        return jwt;
    }

    public String getVPortId() {
```

```
        return vPortId;
    }

    public String getNickName() {
        return nickName;
    }

    public String getMobile() {
        return mobile;
    }

    public Issuer getIssuer() {
        return issuer;
    }
}
```

● 用户实名认证信息对象

```
/**
 * @Title: 实名认证身份证信息
 */
class IdentityCard {
    // 获取的身份证信息的 JWT 串
    private String jwt;

    // 从 JWT 串中解析出来的居民身份号码，名字，签发机关，有效期开始日期，有效期结束日期
    private String cin;
    private String name;
    private String sex;
    private String authority;
    private java.util.Date dateIssue;
    private java.util.Date dateExpiry;

    // 从 JWT 串中解析出来的签名者信息
    private Issuer issuer;

    public boolean setup(String jwt) {
        // 对 jwt 进行解析，如果解析失败，返回 false
        this.cin = 解析出来的居民身份号码;
        this.name = 解析出来的居民姓名;
        this.sex = 解析出来的居民性别;
        this.authority = 解析出来的签发机关;
```

```
        this.dateIssue = 解析出来的有效期开始日期;
        this.dateExpiry = 解析出来的有效期结束日期;

        验证签名是否正确;

        验证签名者 issuer 信息是否真实;

        if (任何的解析失败 || 验证签名错误 || 验证签名者信息错误) {
            this.cin = null;
            this.name = null;
            this.authority = null;
            this.dateExpiry = null;

            this.issuer = null;

            return false;
        }

        this.jwt = jwt;

        return true;
    }

    public String getJwt() {
        return jwt;
    }

    public String getId() {
        return id;
    }

     public String getName() {
         return name;
     }

     public String getAuthority() {
         return authority;
     }

     public java.util.Date getDateIssue() {
         return dateIssue;
     }

     public java.util.Date getDateIssue() {
```

```
        return dateExpiry;
    }

    public Issuer getIssuer() {
        return issuer;
    }
}
```

- 第三方服务器在应用逻辑之间用于传递附加信息的上下文对象接口

```
/**
 * @Title:第三方服务器在应用逻辑之间用于传递附加信息的上下文对象接口
 */
public interface AppContext {
}
```

- 第三方服务器通过 IMI SDK 所获取的用户信息结果对象

```
/**
 * @Title:第三方服务器通过 IMI SDK 所获取的用户信息结果对象，包括用户信息对象列表和上下文对象
 */
public class UserInfoResult {
    // 获取的用户信息列表，表达方式为<信息类型，信息对象>，
    // 信息类型字符串值和对象的 Java 类型名保持一致，
    // 比如”LoginUserInfo”，”IdentityCard”
    // 由第三方服务器解析使用
    private List<String, Object> userInfs;

    // 透传给第三方服务器使用的上下文对象，由第三方服务器解析使用
    // 如果第三方在调用传参的时候给的是个 null，这里也是个 null 值
    private AppContext context;

    private UserInfoResult(List<String, Object> userInfs, AppContext context) {
        this.userInfs = userInfs,
        this.context = context;
    }
```

```
    public static UserInfoResult setup(List<String, Object> userInfs,
AppContext context) {
        return new UserInfoResult(userInfs, context);
    }

    public List<String, Object> getUserInfs() {
        return userInfs;
    }

    public AppContext getContext() {
        return context;
    }
}
```

13.3.2. 第三方服务器需要实现的上下文对象

- 第三方服务器在应用逻辑之间用于传递附加信息而需要实现的上下文对象（根据具体业务逻辑来定义，不是必选项）

```
class LoginContext implements AppContext {
    // 这里是一个使用上下文对象的例子：
    // 1. 第三方服务器获取一个异步请求的上下文对象，
    //    后续需要通过它完成 HTTP Response 操作
    // 2. 第三方服务器在获取 IMI 用户的全信息后，
    //    就默认在后台为客户端 APP 完成了用户登录操作
    // 3. 后台完成用户登录操作后，需要返回给客户端所登录用户对应的 token 值，
    //    以便第三方 APP 能使用 token 做更多的请求操作
    // 当然，还可以在上下文对象里面为第三方 APP 返回更多的附加信息
    private javax.servlet.AsyncContext asyncContext;
    private String token;
    private String error;

    public LoginContext(javax.servlet.AsyncContext asyncContext) {
        this.asyncContext = asyncContext;
    }

    public javax.servlet.AsyncContext getAsyncContext () {
        return asyncContext;
```

```
    }

    public String getToken() {
        return token;
    }

    public String setToken(String token) {
        this.token = token;
    }

    public String getError() {
        return error;
    }

    public String setError(String error) {
        this.error = error;
    }
}
```

13.3.3. 密钥管理服务相关的 SDK

- 加载 Keystore 文件接口

```
public class KeyData {
        String privateKey;      // 私钥
        String publicKey;       // 公钥
        String accountAddress;  // 账户地址
};

public class KeyStoreUtils {
        public static KeyData loadKeyStore(String password, String filePath);
}
```

测试例子如下：

```
String password = "12345678";
String directoryPath = "/home/ks";
String fileName= "my-test-ks";
String filePath = directoryPath + File.separator + fileName;

KeyData keyData = KeyStoreUtils.loadKeyStore(password, filePath);

System.out.printf("keyData=[%s]\n", keyData.toString());
```

13.3.4. 点对点通信通道获取相关的 SDK

● 获取点对点通道对象接口

```
public class MappingSever {
        public static PeerProxy getPeerProxy(KeyData keyData, String openId,
String[] scope, String version) throws java.lang.Exception;
};
```

13.3.5. IMI 用户信息获取相关的 SDK

● 获取用户信息接口

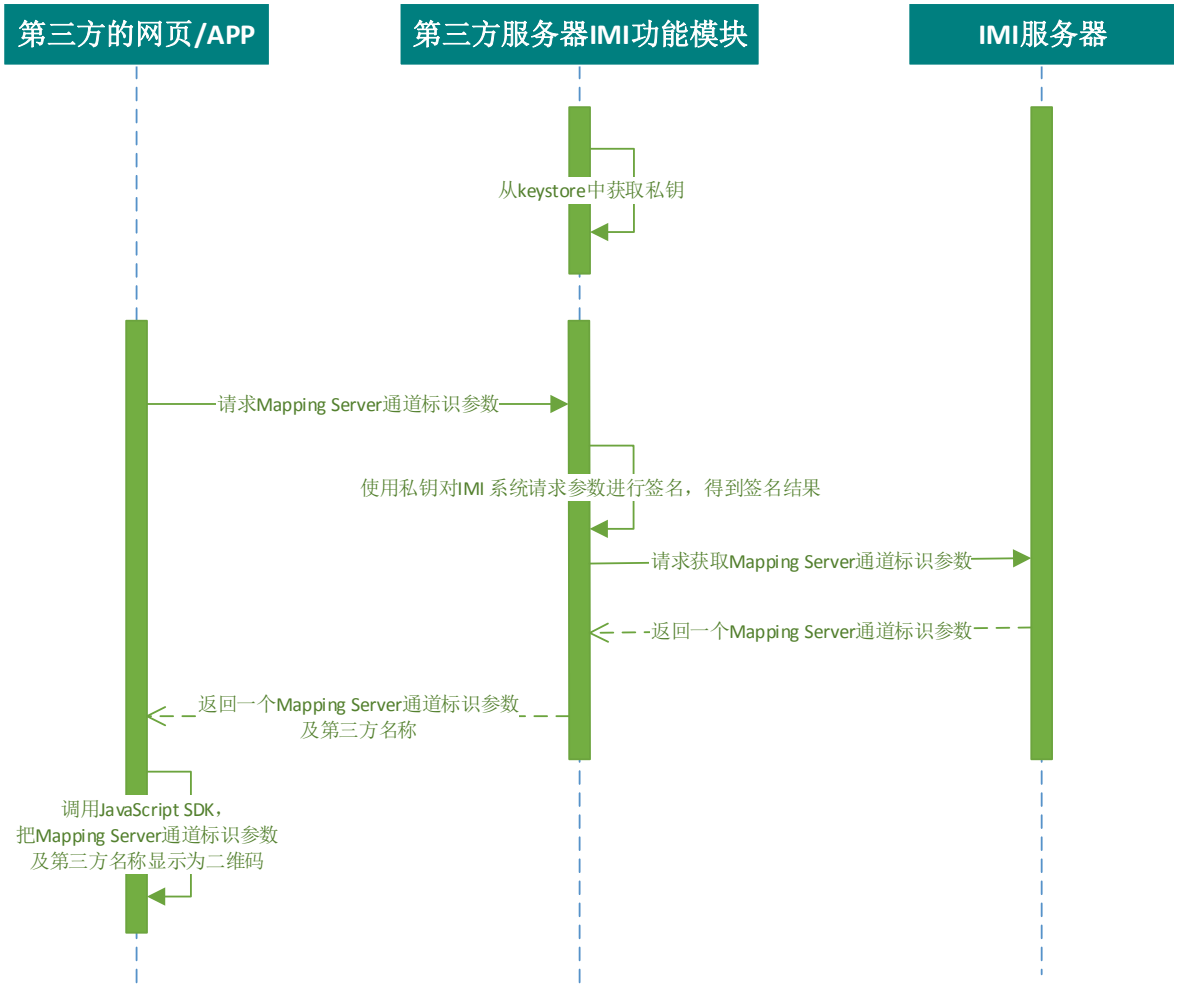
```
import org.apache.http.concurrent;

public class UserInfoProxy {
    /**
     * @Title: queryUserInfo
     * @Description: 获取 IMI 用户的信息（异步接口）
     * @param proxy              通过点对点数据通信的代理对象
     * @param issuer             请求发起者的签名信息
     * @param keyData            请求发起者的 KeyData
     * @param FutureCallback<UserInfoResult>  使用 FutureCallback 回调的方式，
     *                                        在业务完成后返回用户信息结果对象
     * @param AppContext   服务器应用逻辑的上下文对象，SDK 在内部不会对它进行
     *                     任何修改，在调用 Future/FutureCallback 函数的时候
     *                     把 AppContext 作为参数传递给 UserInfoResult
     * @return Future<UserInfoResult>  异步执行结果
     */
    public static Future<UserInfoResult> queryUserInfo(PeerProxy proxy,
Issuer issuer, KeyData keyData, FutureCallback<UserInfoResult> callback,
AppContext appContext);
}
```

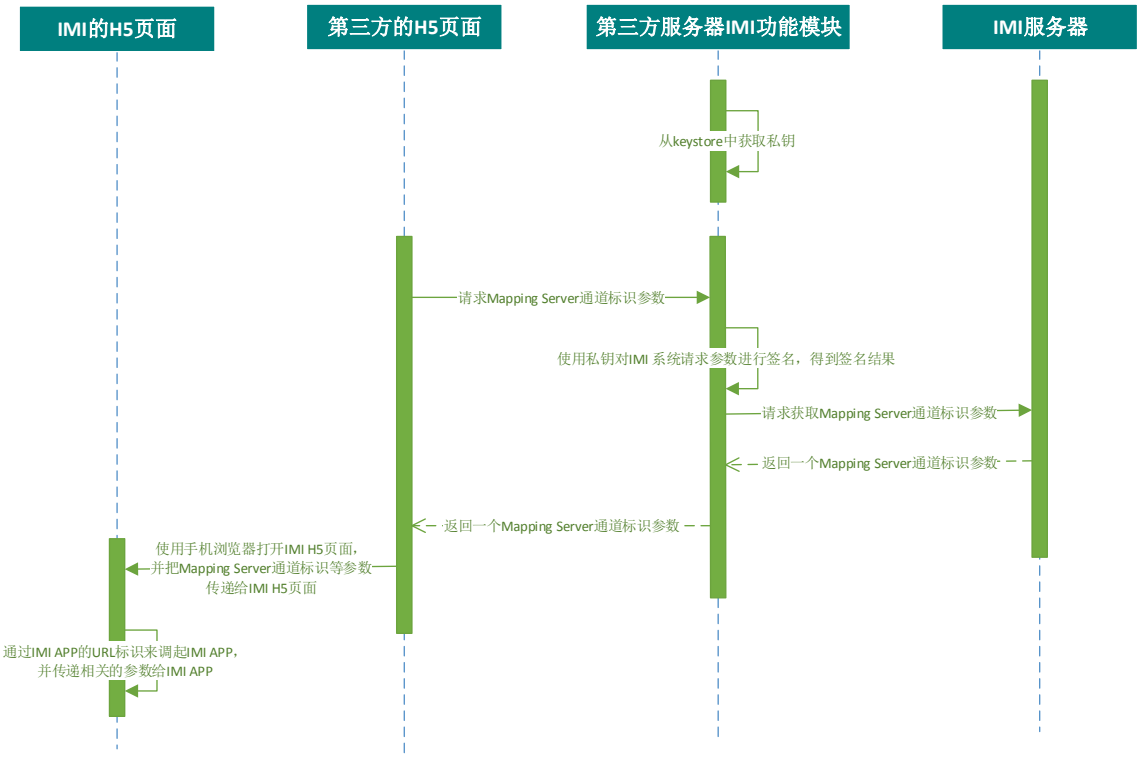
13.2 第三方网页客户端 JavsScript SDK

第三方网页客户端 SDK 主要提供下面三个功能：

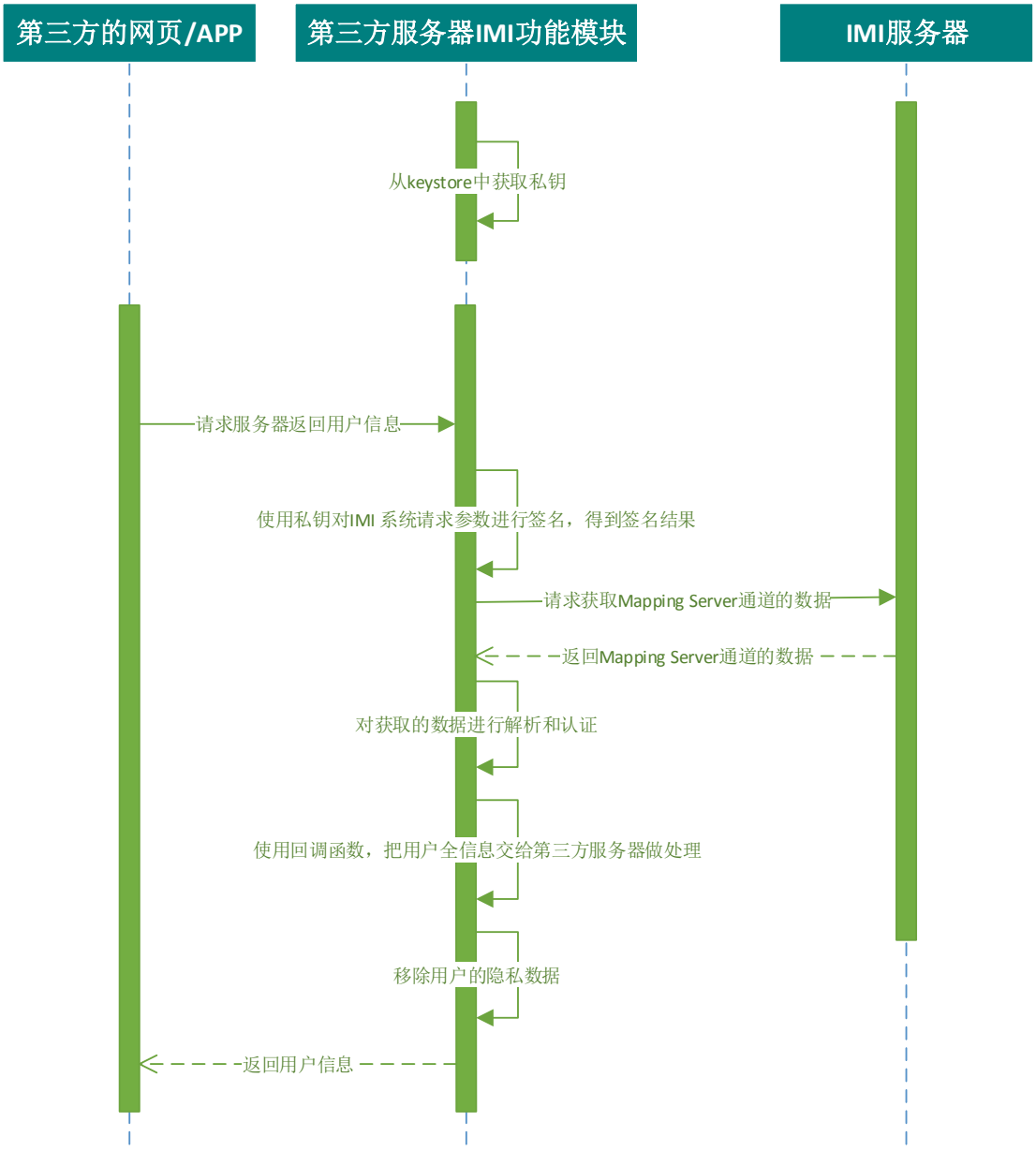
1. 二维码的显示功能（网站版本）



2. 调起 IMI APP 并请求用户授权功能(H5 小程序版本)


IMI的H5页面
第三方的H5页面
第三方服务器IMI功能模块
IMI服务器
从keystore中获取私钥
请求Mapping Server通道标识参数
使用私钥对IMI 系统请求参数进行签名，得到签名结果
请求获取Mapping Server通道标识参数
返回一个Mapping Server通道标识参数
返回一个Mapping Server通道标识参数
使用手机浏览器打开IMI H5页面，
并把Mapping Server通道标识等参数
传递给IMI H5页面
通过IMI APP的URL标识来调起IMI APP，
并传递相关的参数给IMI APP

3.获取 IMI 用户信息



注意：

为了保护 IMI 用户的数据隐私权，第三方服务器返回给网页客户端的用户信息不能包含用户的隐私数据。除非第三方有技术能力来保证访问第三方服务器接口的网页客户端是个可信的授权网页客户端，而不是来自于一个钓鱼网站。

智信禅城 IMI 系统为第三方网页客户端提供了下列 SDK 函数（以 JavaScript SDK 方式进行描述）：

13.2.1. SDK 里面需要用到的各种实体对象

- IMI 用户信息

```
// IMI 用户信息
// 通过 vportId 可以唯一对应第三方服务器后台所保存的 IMI 用户全信息
// 信息中额外附加了 name，主要是为了网页页面显示方便
function UserInfo(vportId, name) {
    this.vportId = vportId;
    this.name = name;
}
```

- IMI 二维码信息

```
// IMI 二维码信息
// 这些参数都会以二维码的形式显示在网页上，提供给 IMI APP 做扫描识别使用
function QrCodeInfo(openId, topicId, scope, version) {
    this.openId = openId;
    this.topicId = topicId;
    this.scope = scope;
    this.version = version;
}
```

13.2.2. 第三方网页需要实现的回调函数

- 第三方网页需要实现的二个回调函数

```
function vPortProxy() {
    this.getQrCodeInfo = function() {
        // 这是第三方网页需要实现的回调函数，
        // 第三方网页通过第三方服务器获取相关的参数值，
        // 并返回实体对象，提供给智信禅城 IMI SDK 使用
        var qrCodeInfo = new QrCodeInfo(openId, topicId, scope, version);
```

```
        return qrCodeInfo;
    }

    this.getUserInfo = function(qrCodeInfo) {
        // 这是第三方网页需要实现的回调函数，
        // 第三方网页使用 topicId 和 scope 信息从第三方服务器获取
        // 智信禅城 IMI 用户信息{vPortId, name}
        // 如果错误或者无法获取信息，返回 null
        // 如果获得 IMI 用户信息，返回 new UserInfo(vportId, name)
        var userInfo = new UserInfo(vportId, name);
        return userInfo;
    }
}
```

13.2.3. 在网页上显示获取用户信息的二维码

```
function vPortUserInfo(vPortProxy) {
    this.vPortProxy = vPortProxy;

    this.sleep = function(time) {
        return new Promise((resolve) => setTimeout(resolve, time));
    }

    // 显示二维码的 SDK 函数接口
    this.showQrCode = function() {
        var qrCodeInfo = this.vPortProxy.getQrCodeInfo();
        // 使用 qrCodeInfo 信息，显示为二维码提供给 IMI APP 做扫描
    }

    // 通过第三方服务器获取用户信息的 SDK 函数接口
    this.getUserInfo = function(qrCodeInfo, blockms) {
        var n = blockms/2000
        for (var i = 0; i < n; i++) {
            this.sleep(2000);
            var userInfo = this.vPortProxy.getUserInfo(qrCodeInfo);
            // 做额外的信息处理？

            if (userInfo != null)
                break;
        }

        return userInfo;
```

```
    }
}
```

13.2.4. 第三方 H5 页面通过 IMI 手机端 H5 网页调起 IMI APP 并获取授权（H5 小程序版本）

```
function imiAppAuthorize(vPortProxy) {
    this.vPortProxy = vPortProxy;

    // 使用手机端浏览器打开 IMI H5 网页来调起 IMI APP，
    // 在获取 IMI APP 用户的授权操作后返回到第三方 H5 网页
}
```

13.2.5. 在网页上获取用户信息

```
function vPortUserInfo(vPortProxy) {
    this.vPortProxy = vPortProxy;

    this.sleep = function(time) {
        return new Promise((resolve) => setTimeout(resolve, time));
    }

    this.showQrCode = function() {
        var qrCodeInfo = vPortProxy.getQrCodeInfo();
        // 使用 qrCodeInfo 信息，显示为二维码提供给 IMI APP 做扫描
    }

    // IMI APP 扫描二维码后，网页获取 IMI 用户信息的 SDK 函数接口
    // 注意：
    // 这个接口是和服务器逻辑一起使用的，
    // 服务器已经获取到了 IMI 用户的全信息，
    // 网页只需要通过 vportId 就可以唯一对应服务器内部的 IMI 用户
    this.getUserInfo = function(qrCodeInfo, blockms) {
        var n = blockms/2000
        for (var i = 0; i < n; i++) {
            this.sleep(2000);
            var userInfo = this.vPortProxy.getUserInfo();
            // 做额外的信息处理？
```

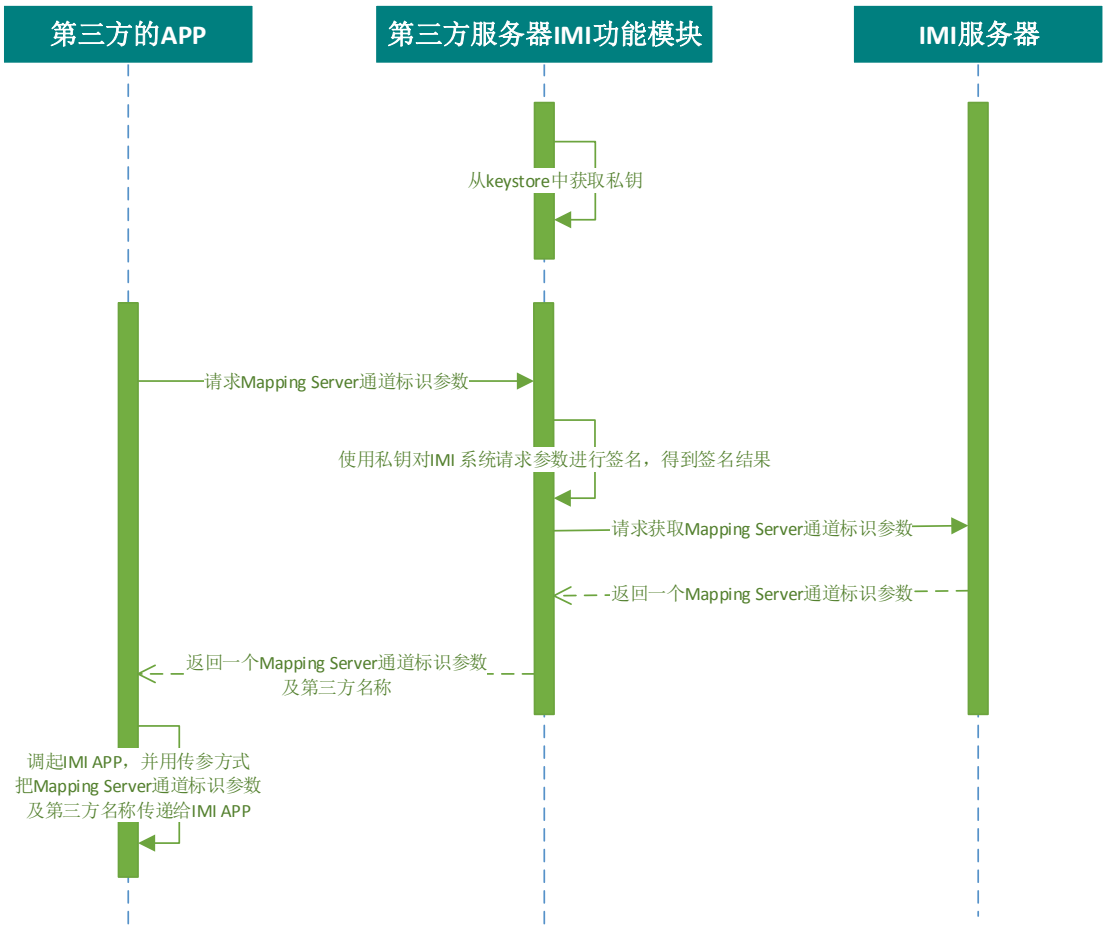
```
            if (userInfo != null)
                break;
        }

        return userInfo;
    }
}
```

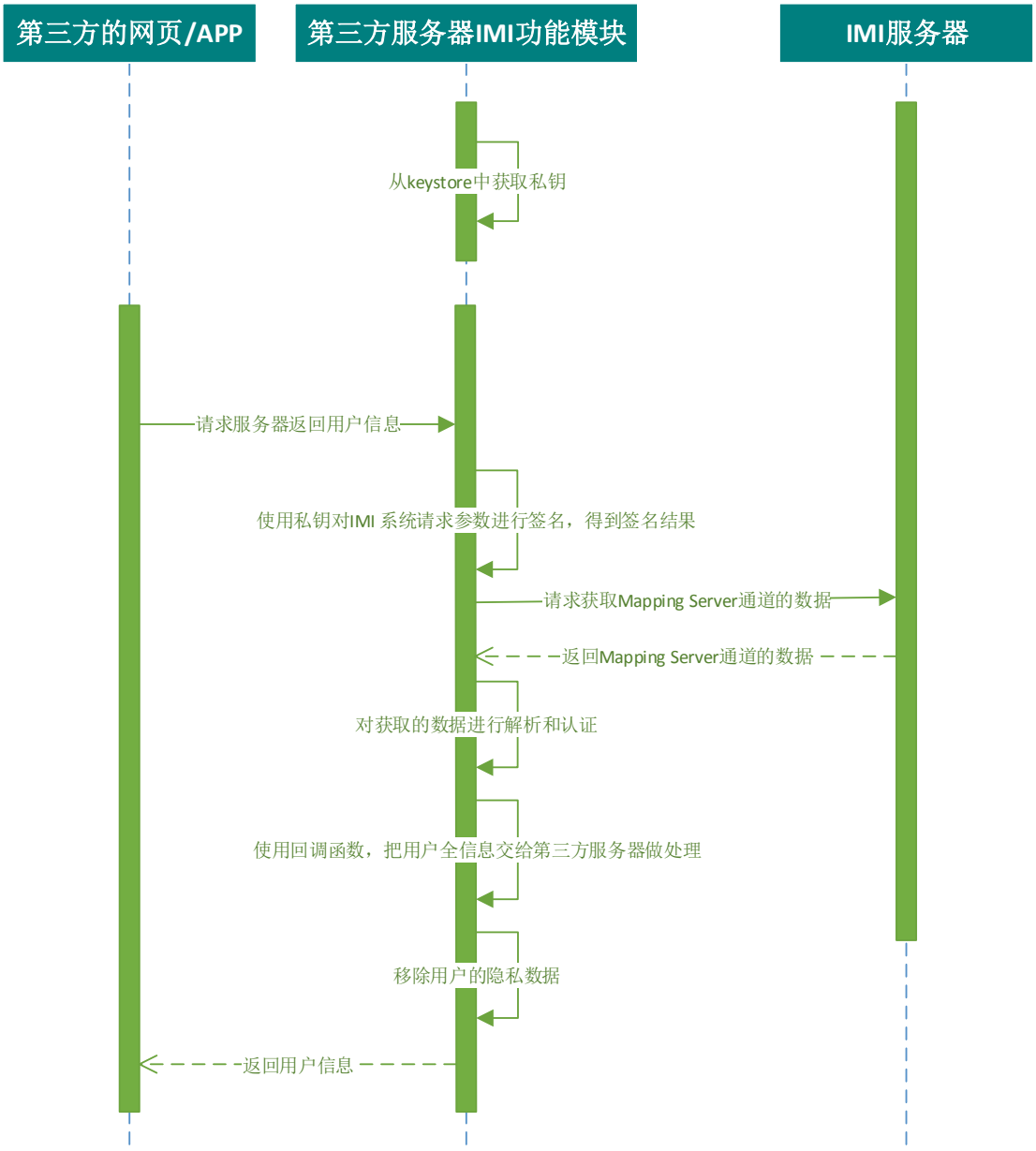
13.3 第三方 Android 客户端 SDK

第三方 Android 客户端 SDK 主要提供下面二个功能：

1. 通过第三方服务器获取 MappingServer 通信标识等参数，调起 IMI APP 并以系统传参的方式把 MappingServer 通信标识等参数传递给 IMI APP：



4. 获取 IMI 用户信息



注意：

为了保护 IMI 用户的数据隐私权，第三方服务器返回给 APP 客户端的用户信息不能包含用户的隐私数据。除非第三方有技术能力来保证访问第三方服务器接口的 APP 客户端是个可信的授权 APP 客户端，而不是

来自于一个钓鱼网站。

智信禅城 IMI 系统为第三方 Android 客户端提供了下列 SDK 函数（以 Java SDK 方式进行描述）：

13.3.1. SDK 里面需要用到的各种实体对象

- 数据通信代理信息

```
/**
 * @Title: 数据通信代理信息
 */
class PeerProxy {
    // 通过 MappingServer 通信代理和对方建立起通讯连接时，需要的相关代理信息
    private String openId;
    private String topicId;

    // 和 IMI APP 数据交互时，需要的相关代理信息
    private String[] scope;
    private String version;

    private PeerProxy(String openId, String topicId, String[] scope, String
version) {
        this.openId = openId;
        this.topicId = topicId;

        this.scope = scope;
        this.version = version;
    }

    public static PeerProxy create(String openId, String topicId, String[]
scope, String version) {
            return new PeerProxy(openId, topicId, scope, version);
    }

    public String getOpenId() {
```

```
        return openId;
    }

    public String getTopicId() {
        return topicId;
    }

    public String[] getScope() {
        return scope;
    }

    public String getVersion() {
        return version;
    }
}
```

● IMI 用户信息

```
/**
 * @Title: IMI 用户信息
 *          通过 vportId 可以唯一对应第三方服务器后台所保存的 IMI 用户全信息
 *          信息中额外附加了 name，主要是为了客户端显示方便

 */
class UserInfo {
    // 第三方服务器在后台获取到用户的全信息后，返回给客户端的用户信息
    //
    private String vportId;
    private String name;

    private UserInfo(String vportId, String name) {
        this.vportId = vportId;
        this.name = name;
    }

    public static UserInfo create(String vportId, String name) {
        return new UserInfo(vportId, name);
    }

    public String getVportId() {
        return vportId;
    }

    public String getName() {
        return name;
```

```
    }
}
```

- 第三方 APP 和第三方服务器之间用于传递附加信息的上下文对象接口

```
/**
 * @Title: 第三方 APP 和第三方服务器之间用于传递附加信息（比如登陆完成后用户的
token 信息，或者登陆失败的原因信息）的上下文对象接口
 */
public interface AppContext {
}
```

13.3.2. 第三方 APP 需要实现的上下文对象

- 第三方 APP 和第三方服务器之间用于传递附加信息而需要实现的上下文对象（根据具体业务逻辑来定义，不是必选项）

```
class LoginContext implements AppContext {
    // 这里是一个使用上下文对象的例子：
    // 1. 第三方服务器在获取 IMI 用户的全信息后，
    //    就默认在后台为客户端 APP 完成了用户登录操作
    // 2. 后台完成用户登录操作后，需要返回给客户端所登录用户对应的 token 值，
    //    以便第三方 APP 能使用 token 做更多的请求操作
    // 当然，还可以在上下文对象里面为第三方 APP 返回更多的附加信息
    private String token;
    private String error;

    public String getToken() {
        return token;
    }

    public String setToken(String token) {
        this.token = token;
    }

    public String getError() {
        return error;
    }
```

```
    public String setError(String error) {
        this.error = error;
    }
}
```

13.3.3. 第三方 APP 需要实现的回调处理函数

- 第三方 APP 需要通过后台服务接口实现的回调处理函数（提供给 SDK 内部调用来完成 IMI 业务）

```
public interface IMIServiceHandler {
    /**
     * @Title: getPeerProxy
     * @Description: 通过第三方服务器获取智信禅城 IMI 的数据通信代理信息
     * @param scope   用户信息的类型集合
     * @param version 代理信息的版本
     * @return PeerProxy  返回数据通信代理信息
     */
    public PeerProxy getPeerProxy(String[] scope, String version) throws
java.lang.Exception;

    /**
     * @Title: getUserInfo
     * @Description: 通过第三方服务器获取智信禅城 IMI 的用户信息和附加信息，
     *               并把服务器传递过来的附加信息以上下文对象的方式来返回
     * @param PeerProxy   数据通信代理信息
     * @param AppContext  第三方服务器需要传递给 APP 的附加信息通过上下文对象返回
     * @return UserInfo   返回结果：
     *                    如果获取用户信息失败，返回 null
     *                    如果获取用户信息成功，返回用户信息对象
     */
    public UserInfo getUserInfo(PeerProxy peerProxy, AppContext context)
throws java.lang.Exception;
}
```

13.3.4. 第三方 App 调起 IMI APP

```
public class IMIUserAuthorize {
    /**
     * @Title: queryUserAuthorize
     * @Description: 通过第三方服务器获取相关参数，
     *               然后通过系统调起 IMI APP 并传递相关参数给 IMI APP
     * @param scope   用户信息的类型集合
     * @param version 代理信息的版本
     * @param handler 第三方 APP 实现的回调处理函数，提供给 SDK 实现 IMI 业务功能
     * @return PeerProxy  返回数据通信代理信息
     */
     public static PeerProxy queryUserAuthorize(String[] scope, String
version, IMIServiceHandler handler) throws java.lang.Exception;
}
```

13.3.5. 第三方 APP 获取用户信息和 APP 上下文对象

```
public class IMIUserAuthorize {
    /**
     * @Title: getUserInfo
     * @Description: 通过第三方服务器获取 IMI 用户信息和附加信息，
     *               并把服务器传递过来的附加信息以上下文对象的方式来返回
     * @param peerProxy  数据通信代理信息
     * @param AppContext  第三方服务器需要传递给 APP 的附加信息通过上下文对象返回
     * @param handler  第三方 APP 实现的回调处理函数，提供给 SDK 实现 IMI 业务功能
     * @return UserInfo   返回结果：
     *                    如果获取用户信息失败，返回 null
     *                    如果获取用户信息成功，返回用户信息对象
     */

   public static UserInfo getUserInfo(PeerProxy peerProxy, AppContext
 context, IMIServiceHandler handler) throws java.lang.Exception;
}
```