M1设备USB-AOA接口说明

本文将阐述本设备提供的 USB 服务接口，以提供完整的掌纹掌静脉特征提取、特征数据管理、掌纹掌静脉比对的能力以及设备侧时间设置。

USB 连接不需要设置任何ip及端口，只需要确保数据传输通道打开。

目前主要应用场景是和Android设备连接，以下是对已测试场景的记录。

Android AOA模式连接场景

- 使用M1 HOST USB口用数据线连接手机，会自动给 Android 设备发送切换 AOA 模式的控制传输指令。
- M1 会通过数据线给 Android 设备充电。
- 如果 Android 设备没有安装的符合过滤条件的 app，会提示打开 moqi.com.cn 网页。
- 有符合条件的 app 会提示打开应用，符合条件 app 开发详见后续介绍。
- 成功切换到 AOA 模式后，只要不拔掉数据线或者某一方断电，始终还是处于 AOA 模式。
- 数据的接收和发送都先用8个byte来充当头部信息，前四个byte都是0xFE，后四个byte代表发送或接收的有效内容长度，后面app开发中有kotlin的实现。

接口使用约束

为了提供设备则通用SDK的接口能力，我们对此设备的 USB 服务接口设置了以下的约束条件：

- 同一个人的特征，只能传输一份，如果同一个人录入了多份特征，那比中的结果，只能返回最早录入特征的 custom_id;
- 采集掌纹特征 和 比对都是一个等待流程，有结果后会广播给连接的 Android 设备;
- 仅M1设备端处于单机模式，才能支持主动服务请求，当M1设备端处于集群模式，那么仅支持接收比对结果的广播信息；
- 客户端请求数据包使用 json 格式打包数据，且需要传入 type 字段，声明请求的服务类型,举例：

```
{
    "type": "systemtime",
    "date_time": "2024-07-30 12:12:12"
}
```

- 返回的 response，也同样会声明返回的消息类型，使用 json 格式打包数据，举例：
    - `device_id`：设备的ID值，当设备重新初始化，该值会重新生成;
    - `device_id`：设备的SN序列号，该值是固定值;

```
{
    "type": "systemtime",
    "code": "0",
    "device_id" : "4e9a08b0-2e63-98fc-a003-83aea13a02bd",
    "device_sn" : "M100010223500024",
    "message": "操作成功",
    "data": null
}
```

接口详情

采集掌纹掌静脉特征

该接口用于在 M1 设备则采集目标人的掌纹掌静脉生物特征。

Request Parameters:

参数名	类型	是否必填	参考值	说明
type	str	是	"enroll"	请求服务类型
if_enroll_right_hand	bool	是	true	是否是右手掌，默认值是 true

设备进入采集模式，同步返回

参数名	类型	参考值	说明
type	str	"enroll"	返回服务消息类型
code	int	0	状态码，默认为0则表示请求成功
message	string	操作成功	状态码对应的文言描述，默认为"操作成功"

Response example:

```
{
    "type" : "enroll",
    "code" : 0,
    "device_id" : "4e9a08b0-2e63-98fc-a003-83aea13a02bd",
    "device_sn" : "M100010223500024",
    "message" : "操作成功"
}
```

采集成功后，广播采集的结果

- Response Parameters:

参数名	类型	参考值	说明
type	str	"enroll_result"	返回服务消息类型
code	int	0	状态码，默认为0则表示请求成功
message	string	操作成功	状态码对应的文言描述，默认为"操作成功"
data	json	-	返回设备采集掌纹掌静脉生成的特征数据结果
data->is_success	bool	true	是否采集成功
data->is_right_hand	bool	true	是否是右手掌
data->palms	array	[]	json格式的特征数据数组
data->palms->element	json元素	-	json格式的特征数据数组，数组长度为3
data->palms->element->feature	string	-	特征数据

参数名	类型	参考值	说明
data->palms->element->ir_score	float	23.08599853515625	IR图像质量分数
data->palms->element->rgb_score	float	23.08599853515625	RGB图像质量分数

- Response example:


```
{
    "type" : "enroll_result",
    "code" : 0,
    "device_id" : "4e9a08b0-2e63-98fc-a003-83aea13a02bd",
    "device_sn" : "M100010223500024",
    "message" : "操作成功",
    "data" :
    {
        "is_right_hand" : true,
        "is_success" : true,
        "palms" :
        [
            {
                "feature" : "OCkAADgpAACQuGCUoyR7NvO***",
                "ir_score" : 23.08599853515625,
                "rgb_score" : 100.0
            },
            {
                "feature" : "OCkAADgpAACQuGCUoyR7NvO***",
                "ir_score" : 23.08599853515625,
                "rgb_score" : 100.0
            },
            {
                "feature" : "OCkAADgpAACQuGCUoyR7NvO***",
                "ir_score" : 23.08599853515625,
                "rgb_score" : 100.0
            }
        ]
    }
}
```


人员接口:导入人员掌纹掌静脉特征数据到 M1 设备

该接口用于导入人员掌纹掌静脉特征数据到 M1 设备，以便与后续 1:1 以及 1:N 的掌纹掌静脉比对。

- Request Parameters:

参数名	类型	是否必填	参考值	说明
type	str	是	"user_import"	请求服务类型
username	str	否	"TEST"	用户名
custom_id	str	是	"custom_id"	用户ID，要求必须唯一， 如果系统判断库中有存在的ID， 则导入会失败

参数名	类型	是否必填	参考值	说明
expire_time	str	是	"1577976159"	该特征数据失效时间的时间戳(单位是秒), 设备当前时间如果大于此时间, 则特征数据无效，无法比中
right_templates	array	是	["tempalte1","tempalte2","tempalte3"]	右手掌纹掌静脉特征数据字符串数组
left_templates	array	是	["tempalte1","tempalte2","tempalte3"]	左手掌纹掌静脉特征数据字符串数组

- Response Parameters:

参数名	类型	参考值	说明
type	str	"user_import"	返回服务消息类型
code	int	0	状态码，默认为0则表示请求成功
message	string	操作成功	状态码对应的文言描述，默认为"操作成功"
data	json	-	返回设备采集掌纹掌静脉生成的特征数据结果

- Response example:

```
{
    "type" : "user_import",
    "code" : 0,
    "device_id" : "4e9a08b0-2e63-98fc-a003-83aea13a02bd",
    "device_sn" : "M100010223500024",
    "message" : "操作成功",
    "data" : null
}
```

人员接口:获取 M1 设备侧注册人员的列表

该接口用于获取已经在 M1 设备注册的人员列表。

- Request Parameters:

参数名	类型	是否必填	参考值	说明
type	str	是	"user_list"	请求服务类型
limit	int	是	100	分页大小
offset	int	是	0	偏移量

- Response Parameters:

参数名	类型	参考值	说明
type	str	"user_list"	返回服务消息类型
code	int	0	状态码，默认为0则表示请求成功
message	string	操作成功	状态码对应的文言描述，默认为"操作成功"

参数名	类型	参考值	说明
data	json	-	返回的人员数据列表对象
data->limit	int	100	分页大小
data->offset	int	0	偏移量
data->total	int	3	总的人员数据条数
data->list	array	-	人员数据列表
data->list->element->custom_id	str	"10086"	用户ID
data->list->element->username	str	"TEST"	用户名
data->list->element->expire_time	int	1577978989	特征过期时间的时间戳(单位是秒)
data->list->element->template_ids_right	array	["36","37","38"]	入库右掌纹特征数据对应的入库ID
data->list->element->template_ids_left	array	["39","40","41"]	入库左掌纹特征数据对应的入库ID

- Response example:


```
{
    "type" : "user_list",
    "code" : 0,
    "device_id" : "4e9a08b0-2e63-98fc-a003-83aea13a02bd",
    "device_sn" : "M100010223500024",
    "message" : "操作成功",
    "data" :
        {
            "limit" : 100,
            "list" :
            [
                {
                    "custom_id" : "2",
                    "expire_time" : 1577978989,
                    "template_ids_left" :
                    [
                        "39",
                        "40",
                        "41"
                    ],
                    "template_ids_right" :
                    [
                        "36",
                        "37",
                        "38"
                    ],
                    "username" : "Test1"
                }
            ],
            "offset" : 0,
            "total" : 1
        },
}
```

人员接口:根据 custom_id 获取 M1 设备侧获取人员信息

该接口用于获取已经在 M1 设备注册的人员列表。

- Request Parameters:

参数名	类型	是否必填	参考值	说明
type	str	是	"user_info"	请求服务类型
custom_id	str	是	"test"	用户ID

- Response Parameters:

参数名	类型	参考值	说明
type	str	"user_info"	返回服务消息类型
code	int	0	状态码，默认为0则表示请求成功
message	string	操作成功	状态码对应的文言描述，默认为"操作成功"
data	json	-	返回的人员数据
data->custom_id	str	"10086"	用户ID
data->username	str	"TEST"	用户名
data->expire_time	int	1577978989	特征过期时间的时间戳(单位是秒)
data->template_ids_right	array	["36","37","38"]	入库右掌纹特征数据对应的入库ID
data->template_ids_left	array	["39","40","41"]	入库左掌纹特征数据对应的入库ID

- Response example:

```
{
    "type" : "user_info",
    "code" : 0,
    "device_id" : "4e9a08b0-2e63-98fc-a003-83aea13a02bd",
    "device_sn" : "M100010223500024",
    "message" : "操作成功",
    "data" :
    {
        "custom_id" : "2",
        "expire_time" : 1577978989,
        "template_ids_left" :
        [
            "39",
            "40",
            "41"
        ],
        "template_ids_right" :
        [
            "36",
            "37",
            "38"
        ],
        "username" : "Test1"
    }
}
```


人员接口:删除 M1 设备侧人员信息

该接口用于删除 M1 设备侧人员信息。

- Request Parameters:

参数名	类型	是否必填	参考值	说明
type	str	是	"user_delete"	请求服务类型
is_all	bool	是	false	是否删除 M1 设备侧全部人员数据
custom_id	str	否	"test"	用户ID，如果 is_all=false，则此参数生效

- Response Parameters:

参数名	类型	参考值	说明
type	str	"user_delete"	返回服务消息类型
code	int	0	状态码，默认为0则表示请求成功
message	string	操作成功	状态码对应的文言描述，默认为"操作成功"

- Response example:

```
{
    "type" : "user_delete",
    "code" : 0,
    "device_id" : "4e9a08b0-2e63-98fc-a003-83aea13a02bd",
    "device_sn" : "M100010223500024",
    "message" : "操作成功",
    "data" : null
}
```


唤醒 M1 设备进入比对状态

该接口是同步接口，用于唤醒 M1 设备进入比对状态，并等待设备返回比对的结果，默认超时时间是60s。

Request Parameters:

参数名	类型	是否必填	参考值	说明
type	str	是	"match"	请求服务类型

设备进入比对模式，同步返回

Response Parameters:

参数名	类型	参考值	说明
type	str	"match"	返回服务消息类型
code	int	0	状态码，默认为0则表示请求成功
message	string	操作成功	状态码对应的文言描述，默认为"操作成功"

Response example:


```
{
    "type" : "match",
    "code" : 0,
    "device_id" : "4e9a08b0-2e63-98fc-a003-83aea13a02bd",
    "device_sn" : "M100010223500024",
    "message" : "操作成功"
}
```


比对结束后，广播比对的结果

Response Parameters:

参数名	类型	参考值	说明
type	str	"match_result"	返回服务消息类型
code	int	0	状态码，默认为0则表示请求成功
message	string	操作成功	状态码对应的文言描述，默认为"操作成功"
data	json	-	返回的比对结果信息
data->expire_time	int	1880380799	此用户信息的过期时间时间戳

参数名	类型	参考值	说明
data->custom_id	str	"10086"	返回特征匹配的用户ID
data->username	str	"10086"	返回特征匹配的用户名
data->auth_nfc_number	str	"12345678901234567890"	NFC 卡卡号
data->auth_log	json	-	鉴权日志
data->auth_log->auth_method	int	1	认证方式,1 手掌,2 刷卡,3 手掌+卡 4 手掌+密码
data->auth_log->auth_result	int	1	认证结果,1 成功,2 失败,3 无访问权限,4 非活体攻击, 5 认证次数超过限制
data->auth_log->auth_time	int	1723541867	鉴权发生时间时间戳
data->auth_log->custom_id	str	"10086"	返回特征匹配的用户ID
data->auth_log->user_id	int	12	返回特征匹配的用户ID
data->auth_log->user_uuid	str	"b1741c48-894c-4d96-ac57-10f2b994b57f"	云端与设备侧使用的唯一用户ID
data->auth_log->username	str	"testtest"	返回特征匹配的用户名

Response example:

```
{
    "code" : 0,
    "device_id" : "4e9a08b0-2e63-98fc-a003-83aea13a02bd",
    "device_sn" : "M100010223500024",
    "data" :
    {
        "auth_log" :
        {
            "auth_method" : 1.0,
            "auth_result" : 5.0,
            "auth_time" : 1723541867.0,
            "custom_id" : "1",
            "user_id" : 12.0,
            "user_uuid" : "b1741c48-894c-4d96-ac57-10f2b994b57f",
            "username" : "testtest"
        },
        "auth_nfc_number" : "12345678901234567890",
        "custom_id" : "1",
        "expire_time" : 1880380799,
        "username" : "testtest"
    },
    "message" : "操作成功",
    "type" : "match_result"
}
```


中止 M1 设备当前任务并回到默认页面

该接口用于中止 M1 设备处理的任务，恢复到默认状态。

Request Parameters:

参数名	类型	是否必填	参考值	说明
type	str	是	"stop"	请求服务类型

Response Parameters:

参数名	类型	参考值	说明
type	str	"stop"	返回服务消息类型
code	int	0	状态码，默认为0则表示请求成功
message	string	操作成功	状态码对应的文言描述，默认为"操作成功"

Response example:


```
{
    "type" : "stop",
    "code" : 0,
    "device_id" : "4e9a08b0-2e63-98fc-a003-83aea13a02bd",
    "device_sn" : "M100010223500024",
    "message" : "操作成功"
}
```

设置 M1 设备当前时间

该接口用于设置 M1 设备的系统时间。

- Request Parameters:

参数名	类型	是否必填	参考值	说明
type	str	是	"systemtime"	请求服务类型
date_time	str	是	"2024-07-30 12:12:12"	时间字符，格式为 "yyyy-MM-dd HH:mm:ss"

- Response Parameters:

参数名	类型	参考值	说明
type	str	"systemtime"	返回服务消息类型
code	int	0	状态码，默认为0则表示请求成功
message	string	操作成功	状态码对应的文言描述，默认为"操作成功"

USB 连接心跳广播

该接口是 M1 设备侧对连接的 websocket client 端进行心跳广播，用于判断连接是否正常。

Response Parameters:

参数名	类型	参考值	说明
type	str	"heartbeat"	返回服务消息类型
code	int	0	状态码，默认为0则表示请求成功
message	string	操作成功	状态码对应的文言描述，默认为"操作成功"

Response example:

```
{
    "type" : "heartbeat",
    "code" : 0,
    "device_id" : "4e9a08b0-2e63-98fc-a003-83aea13a02bd",
    "device_sn" : "M100010223500024",
    "message" : "操作成功"
}
```

接口类型列表

```
QString WEBSOCKET_TASK_TYPE_HEARTBEAT="heartbeat";
QString WEBSOCKET_TASK_TYPE_SYSTEMTIME="systemtime";
QString WEBSOCKET_TASK_TYPE_STOP="stop";
QString WEBSOCKET_TASK_TYPE_MATCH="match";
QString WEBSOCKET_TASK_TYPE_MATCH_RESULT="match_result";
QString WEBSOCKET_TASK_TYPE_ENROLL="enroll";
QString WEBSOCKET_TASK_TYPE_ENROLL_RESULT="enroll_result";
QString WEBSOCKET_TASK_TYPE_UNKNOWN="known";
QString WEBSOCKET_TASK_TYPE_USER_IMPORT="user_import";
QString WEBSOCKET_TASK_TYPE_USER_LIST="user_list";
QString WEBSOCKET_TASK_TYPE_USER_INFO="user_info";
QString WEBSOCKET_TASK_TYPE_USER_DELETE="user_delete";
```

状态码映射表

code	说明
0	操作成功
-1	操作失败
-2	输入参数非法
-3	程序运行出错
-4	所需文件未找到
-999	未知的错误类型
-100	数据库执行失败
-200	设备服务执行失败
-400	设备服务执行失败
10001	用户信息未找到
10005	用户信息保存到数据库失败
10008	用户信息删除失败
10009	删除特征数据失败
10012	用户 ID 已存在
10001	用户信息未找到

APP开发

1. app/src/main/res/xml/ 下新建 accessory_usb_filter.xml

增加以下过滤条件，当连接数据线后可提示打开该应用

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <usb-accessory model="m1-f8" manufacturer="moqi" version="1.0" />
</resources>
```

2. AndroidManifest.xml 在要显示Activity页面使用 accessory_usb_filter.xml

```xml
<activity
    android:name=".YourActivity"
    android:screenOrientation="portrait"
    android:exported="true">

    <intent-filter>
        <action android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED" />
    </intent-filter>

    <meta-data
        android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED"
        android:resource="@xml/accessory_usb_filter" />
</activity>
```

3. 获取 AOA 模式连接的设备

```
val usbManager = context.getSystemService(USB_SERVICE) as UsbManager
if (usbManager.accessoryList != null && usbManager.accessoryList.isNotEmpty()) {
    val usbAccessory = usbManager.accessoryList[0]
}
```

4. 注册广播并申请权限

```
private val mAccessoryPermissionReceiver: BroadcastReceiver = object : BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent) {
        // 权限申请成功后会在此广播接收到，继续做后续的连接处理
        val accessory =
            intent.getParcelableExtra<Parcelable>(UsbManager.EXTRA_ACCESSORY) as UsbAccessory?
        prepareToConnect(accessory)
    }
}

val permissionIntent = PendingIntent.getBroadcast(
    this, 0,
    Intent("您的自定义ACTION"), 0
)
val filter = IntentFilter("您的自定义ACTION")
filter.addAction(UsbManager.ACTION_USB_ACCESSORY_DETACHED)
registerReceiver(mAccessoryPermissionReceiver, filter)
usbManager.requestPermission(usbAccessory, permissionIntent)
```

5. 建立连接并获取读写流

```
private fun prepareToConnect(accessory: UsbAccessory?) {
    val usbManager = context.getSystemService(USB_SERVICE) as UsbManager
    fileDescriptor = usbManager.openAccessory(accessory)
    if (fileDescriptor == null) {
        return
    }
    fd = fileDescriptor?.fileDescriptor
    inputStream = FileInputStream(fd)
    outputStream = FileOutputStream(fd)
}
```

6. 向 M1 写入json数据

```
fun sendMessageWithUSB(message: String) {
    val data = message.toByteArray(Charsets.UTF_8)
    var dataSize = data.size
    Log.e(USB_TAG, "发送数据长度: $dataSize，内容: $message")

    // 先发送一次本次发送的数据大小
    val headInfo = ByteArray(8) { (-2).toByte() }
    headInfo[4] = (dataSize shr 24 and 0xFF).toByte()
    headInfo[5] = (dataSize shr 16 and 0xFF).toByte()
    headInfo[6] = (dataSize shr 8 and 0xFF).toByte()
    headInfo[7] = (dataSize and 0xFF).toByte()
    Log.d(USB_TAG, "发送数据头部信息: ${headInfo.contentToString()}")
    outputStream!!.write(headInfo)
    outputStream!!.flush()

    // 每块最大100k的方式循环将整个内容发送，块的大小可以自己设置
    while (dataSize > 0) {
        val allocateSize = if (dataSize > (1024 * 100)) (1024 * 100) else dataSize
        val buffer = ByteBuffer.allocate(allocateSize)
        buffer.put(data)
        outputStream!!.write(buffer.array())
        outputStream!!.flush()
        dataSize -= allocateSize
    }
}
```

7. 开启线程不断从 M1 读取数据

```
fun readMessageWithUSB() {
while (true) {
    // 读第一次，从头部信息中读出要读取的总大小
    var contentSize = 0  // 记录要读取的总大小
    val headInfo = ByteArray(8)
    val headSize = inputStream!!.read(headInfo)
    val flag: Byte = -2
    if (headSize == 8 && headInfo[0] == flag && headInfo[1] == flag
        && headInfo[2] == flag && headInfo[3] == flag
    ) {
        contentSize =
            ((headInfo[4].toInt() and 0xFF) shl 24) or ((headInfo[5].toInt() and 0xFF) shl 16) or ((headI
        Log.d(USB_TAG, "即将接收json字符串的总大小: $contentSize")
    } else {
        continue
    }

    // 循环读取，直到将总大小完全读出
    var alreadyReadSize = 0 // 记录读取到了哪个位置
    val buffer = ByteArray(contentSize) // 存放最终完整的内容
    while (contentSize > 0) {
        val currReadBuffer = ByteArray(maxBufferLength)
        val currReadSize = inputStream!!.read(currReadBuffer)
        if (currReadSize > 0) {
            System.arraycopy(
                currReadBuffer,
                0,
                buffer,
                alreadyReadSize,
                currReadSize
            )
            alreadyReadSize += currReadSize
            contentSize -= currReadSize
        }
    }
    val readContent = String(buffer, Charsets.UTF_8)
    if (readContent.startsWith("{") && readContent.endsWith("}")) {
        // 对读取到的完整json内容做处理
        dispatchMessage(readContent)
    } else {
        Log.d(USB_TAG, "接收数据非json字符串: $readContent")\
    }

    delay(100)
}
```