FacePass Android版服务器API文档

版本号	编纂者	修改时间
2.00	叶赛尔	2018-2-27

本文档为旷视科技FacePass产品服务器API文档，面向使用客户，旨在全面详实介绍服务器端API的功能、参数、返回值以及部分使用建议。

实际集成请配套FacePass产品客户端SDK文档，以及示例Demo工程一起完整使用。

文档目录如下：

通用部分

API的通用返回部分被单独列出在这里，以避免在每条API都重复写一遍。

每条API仅写本API内部的返回参数、返回值data部分的json_body、以及特有的错误码。

通用返回参数

参数名	参数类型	出现情况	参数说明
code	Int	必有	返回值，如果协议成功则0，如果失败则参见下方错误码
message	Int	必有	错误信息，长度不超过1024位
data	json data	必有	实际API的返回的业务内容均放在data中，如果没有内容，则返回空的body结构体（不返回NULL，也不缺省data参数）
timestamp	Unix时间戳，精确到毫秒	必有	服务器接到API请求的时间戳(单位为毫秒)
timecost	Int	必有	API从接到请求到完成请求的耗时(单位为毫秒)

通用返回示例

Response

```
{
    "code": 0,
    "message": "",
    "timestamp": 1510058788332,
    "timecost": 237,
    "data": {
        // 每条API的实际返回内容
    }
}
```

通用错误码：

错误类型	错误message	错误码	HTTP_CODE	错误说明
请求内容JSON解析失败	ILLEGAL_JSON_BODY	104000	400	通常是json结构解析失败
参数格式错误	ILLEGAL_ARGUMENT: <args>	104001	400	参数不符合预期格式的失败，后面的args返回错误参数的名称，逗号分隔
缺少必要参数	MISSING_ARGUMENT: <args>	104002	400	缺少必要参数，后面的args返回缺少参数的名称，逗号分隔
图片解码失败	UNSUPPORTED_IMAGE_FORMAT	104100	400	图片解码失败，最常见是因为格式不支持，主要是/face/add接口
图片分辨率过大	INVALID_IMAGE_RESOLUTION	104101	400	图片分辨率过大，主要是/face/add接口
图片过大	INVALID_IMAGE_FILE_SIZE	104102	400	图片文件尺寸过大，主要是/face/add接口
接口不存在			404	接口不存在
内部错误	INTERNAL_ERROR	105000	500	其他内部错误

新增Face

版本

v1

描述

新增人脸，输入图片字节流，检测其中的最大一张人脸，进行质量判断，如果以上过程都正确，则输出对应face_token，反之则报错。

1.输入图片建议约束 分辨率大小、图片格式；

2.如果图片中不包含人脸，则无法新增；

3.约定检测的最小人脸为 100*100 px，这是底库图片符合识别要求的最小尺寸；

4.如果图片中包含多张人脸，则检测最大的人脸（前提是满足脸最小尺寸要求）；

5.如果人脸检测通过，则进行抽特征操作，将图片降解为特征值（这也是实际人脸识别比对用的数据结构）并存储，通过face_token索引；

6.无论成败，均会返回image_id，可以通过/image/query接口获取图片，可以用于入库日志分析；

7.无论成败，只要检测到人脸，均会返回rect，告知算法检测到的人脸框位置；

8.成功时，额外返回face_token；失败时，额外返回失败原因；

9.新增人脸和人脸绑定底库是两个独立的操作，一张人脸首先要新增到系统中，给出face_token，然后face_token绑定到group（可以绑定多个group）。

请求URI

/api/face/v1/add

请求方法

POST

请求参数(form)

<table>
<tr><th>参数名</th><th>是否必选</th><th>参数类型</th><th>参数说明</th></tr>
<tr><td>image</td><td>必选</td><td>Binary(File Upload)</td><td>
<h3>图片属性限制：</h3>
1.分辨率限制：

建议尺寸720p：1280*720
支持最小尺寸：必须大于FaceMin，所以必须大于等于100*100。
支持最大尺寸：4096*4096
分辨率约束的是“面积”，所以长宽比很奇怪的图有可能通过，但前提是其中的人脸满足最小脸像素要求。
如果越界，则报错 "INVALID_IMAGE_RESULUTION”。

2.大小限制：

<= 4M
如果越界，则报错“INVALID_IMAGE_FILE_SIZE”。

3.图片格式限制：

目前支持的格式包括：

png
jpg/jpeg

目标不支持的格式包括：

bmp
gif
svg

如果图片格式或者解析错误，则报错“UNSUPPORTED_IMAGE_FORMAT”。

4.图片控制符限制：

我们会读取图片控制符中的如下字段：

旋转：如果图片本身设置了旋转，我们会忠实的按照旋转后的图片来检测人脸，如果参数有误，则有可能导致人脸翻转或者角度过大导致无法检测入库，所以请务必保证输入图片源的该字段表意正确。

5.其他：

底图的选择总体目标是“清晰且和使用者本人一致”
不建议黑白图
不建议使用强烈ps或者艺术化的图
建议尽量使用近期生活照，不建议使用跨年龄太久的照片
是否化妆、是否戴眼镜，尽量和日常使用习惯一致

<h3>人脸属性限制：</h3>
检测的综合流程为：

检测图片上的人脸；
定位最大人脸，并进行质量判断（包括FaceMin）；
通过质量判断则入库成功，进行特征值抽取；
任意步骤失败则退出。

所以对应的会有如下限制和说明：
1.同框人脸数目：

不管原图上有多少人脸，入库环节仅支持单张人脸；
如果有多张人脸，仅选择最大的一张；
如果最大脸失败，“不会”看第二大的脸，而是直接判失败，所以请确保原图中最大脸就是用户的目标人脸。

2.入库质量判断：
<table>
<tr><th>大类</th><th>子类</th><th>数值</th><th>备注</th></tr>
<tr><td>Pose</td><td>Yaw</td><td><= 20°</td><td>水平角度</td></tr>
<tr><td></td><td>Pitch</td><td><= 20°</td><td>垂直角度</td></tr>
<tr><td></td><td>Roll</td><td><= 20°</td><td>旋转角度</td></tr>
<tr><td>Blur</td><td>Blurriness</td><td><= 0.2</td><td>清晰度</td></tr>
<tr><td>Illumination</td><td>Brightness</td><td>[75, 210]</td><td>人脸平均亮度</td></tr>
<tr><td></td><td>Std_Deviation</td><td><= 80</td><td>人脸亮度标准差</td></tr>
<tr><td>FaceMin</td><td>FaceMin</td><td>>= 100</td><td>最小人脸</td></tr>
</table>

最小人脸尺寸（FaceMin或者min-face）：
人脸识别对于比对图片的人脸最小尺寸有要求，太小则损失过多精度、影响识别效果。

默认人脸最小尺寸为：长 >= 100px， 宽 >= 100px，注：等号是成立的，也就是100*100的图可以被使用；
理论上没有最大人脸限制，但是人脸过大会让抽特征的过程略慢于往常，但是是一次性操作，默认可以接受；
如果出现大小完全一样的人脸，则算法会选择先检测到的哪一张。

角度pose：大角度会导致人脸信息缺失，我们分三个轴给出限制，越界则报错

roll <= 20°：以鼻子为轴、脸部平面旋转；
yaw <= 20°：左右摇头；
pitch <= 20°：上下点头；

模糊blur：照片模糊会导致识别效果变差，所以入库阶段尽量保证图片不模糊，包括

运动模糊：因为高速运动造成的图片内容“拖尾”；
高斯模糊：因为相机失焦造成像素点和周围融合；
其他模糊：比如照片翻拍等其他因素；

光照illuminatin：人脸识别对于光照整体上还比较敏感，建议光照柔和清晰、无强烈反差

当前版本没有特别优化逆光、背光；
当前版本没有特别优化暗光；
我们建议平均亮度在 [75-210] lux之间，标准差 <= 80 lux；

</td></tr>
</table>

返回参数(json body)

<table>
<tr><th>参数名</th><th>参数类型</th><th>出现情况</th><th>参数说明</th></tr>
<tr><td colspan="4">产品逻辑上，实际上根据新增人脸的实际情况，分为三种不同的类型的结果：

case A.新增人脸成功：一张人脸有效的录入到了系统中

face_token
image_id
rect

case B.新增人脸失败，因为人脸不满足质量判断：失败，但是返回rect

image_id
rect

case C.新增人脸失败，因为没有检测到人脸：失败，并且无法返回rect

image_id

失败时对应的message参见最底端的错误码部分，其他业务参数如下。</td></tr>
<tr><td>image_id</td><td>String</td><td>必有</td><td>image_id可以理解为上传图片的url，是一个长度为24的字符串，用户可以通过image_id来加载原图，具体的API是：
/image/query
协议request传参image_id，会返回原图字节流（如果传参face_token则返回抠脸的图片）。
图片以单独服务的形式存储，每次新增人脸时进行存储，但是每次删除人脸时并不删除原图。
目前新增人脸的图片存储是永久的，对应的，新增人脸历史也是永久存储的，不设过期时间，所以请确保服务器有基本的存储空间用于保存图片，以及不要入库过于海量的底库图片。</td></tr>
<tr><td>rect</td><td>Dictionary</td><td>选有，
能检测到人脸就有，
caseA和caseB有，caseC没有</td><td>人脸框数组，四个值以及顺序分别为left、top、right、bottom，均为整形数值，不含小数点。
根据人脸框和image_id可以从原图中抠出人脸的位置，用于上层业务层应用（比如在原图上画个人脸框）。
只要能检测到人脸（即使不通过FaceMin）就会有rect，是先检测再判断质量，而FaceMin是质量判断的一部分，所以有可能检测到一张小脸，但是因为不满足FaceMin而被干掉了。
再额外提醒说，如果图片中有多张人脸，入库时候只检测“人脸面积最大的‘一’张人脸”，请确保该人脸就是目标人脸。</td></tr>
</table>

参数名	参数类型	出现情况	参数说明
face_token	String	选有， 当且仅当新增人脸成功是才有	24位字符串，入库成功后由系统生成，人脸的唯一标识，可以理解为faceID，用户可以通过face_token绑定/解绑底库，也可以用于face的删除和查询操作。

返回示例

Response


```
{
    "code": 0,
    "message": "",
    "timestamp": 1510058788332,
    "timecost": 237,
    "data": {
            // -------------- demo for face-add result items ----------------
            // There're 3 result opitions for face-add operation
            // Case A: face-add success!
            // Case B: face-add failed, due to face-quality verification failure.
            // Case C: face-add failed, due to no face detected.

            // Face-add case A: face-add success!
            {
              "face_token": "QDeldd1vlJ1v865FO4dqtg==",
              "image_id": "fxCCOPXkjxOmQdqGpODd8A==",
              "rect": {
                "left": 149,
                "top": 305,
                "right": 448,
                "bottom": 604
              }
            }


            // Face-add case B: face-add failed, due to face-quality verification failure, BAD_QUALITY: <reason>.
            {
              "image_id": "3rO0opbSl8hjEsah12t5nw==",
              "rect": {
                "left": 467,
                "top": 140,
                "right": 565,
                "bottom": 238
               }
            }

            // Face-add case C: face-add failed, due to no face detected, FACE_NOT_FOUND.
            {
              "image_id": "B-vVZd2H-xnBTI-6ZwwRBA=="
            }

    }
}
```


错误码

错误类型	错误message	错误码	HTTP_CODE
没有检测到人脸	FACE_NOT_FOUND	104200	400
人脸未通过质量判断	BAD_QUALITY: <reason>	104201	400

删除Face

版本

v1

描述

删除人脸。

1.删除对应的人脸face_token；

2.从group中删除绑定的face_token关系；

3.存储上保留face_token和image_id的历史，用于入库历史和抓拍历史的查询。

请求URI

/api/face/v1/delete

请求方法

POST

请求参数(json body)

参数名	是否必选	参数类型	参数说明
face_token	必选	String	24位字符串，入库成功后由系统生成（参见/face/add），人脸的唯一标识，可以理解为faceID。 face_token的编码规则是： • 长度24位 • 支持英文小写字母、英文大写字母、英文半角的等号/下划线/中划线 但是实际API处理中，并不会严格校验编码格式，而是直接解密成所需的明文数据，如果失败则报错"ILLEGAL_ARGUMENT"。 用户可以通过face_token绑定/解绑底库，也可以用于face的删除和查询操作。 人脸face_token会和各种group拥有绑定关系，删除时候需要跟所有的group解绑、然后再删除。 face_token和group的绑定操作具体参见： /group/bind

请求示例

Request

```
{
    "face_token": "GKBj9-l9tIjsUVdZQ4eZqQ=="
}
```

返回参数(json body)

无

返回示例

Response

```
{
    "code": 0,
    "message": "",
    "timestamp": 1510058788332,
    "timecost": 237,
    "data": {
    }
}
```

错误码

错误类型	错误message	错误码	HTTP_CODE
人脸不存在	FACE_TOKEN_NOT_EXIST	104202	400

查询人脸信息

版本

v1

描述

查询人脸。

根据face_token，查询对应的人脸信息。

人脸信息和人脸新增时一致，返回image_id，rect，time。

请求URI

/api/face/v1/query

请求方法

POST

请求参数(json body)

参数名	是否必选	参数类型	参数说明
face_token	必选	String	24位字符串，入库成功后由系统生成（参见/face/add），人脸的唯一标识，可以理解为faceID。 face_token的编码规则是： • 长度24位 • 支持英文小写字母、英文大写字母、英文半角的等号/下划线/中划线 但是实际API处理中，并不会严格校验编码格式，而是直接解密成所需的明文数据，如果失败则报错"ILLEGAL_ARGUMENT"。

请求示例

Request

```
{
    "face_token": "GKBj9-l9tIjsUVdZQ4eZqQ=="
}
```

返回参数(json body)

（和face/add的返回值类似，更详细的逻辑和参数说明可以参见face/add API说明）

参数名	参数类型	出现情况	参数说明
image_id	String	必有	image_id可以理解为上传图片的url，是一个长度为24的字符串，用户可以通过iamge_id来加载原图，具体的API是： /image/query 协议request传参image_id，会返回原图字节流（如果传参face_token则返回抠脸的图片）。
rect	Array	必有	人脸框数组，四个值以及顺序分别为left、top、right、bottom，均为整形数值，不含小数点。 根据人脸框和image_id可以从原图中抠出人脸的位置，用于GUI显示为主。

返回示例

Response

```
{
    "code": 0,
    "message": "",
    "timestamp": 1510058788332,
    "timecost": 237,
    "data": {
            "image_id": "MKfyUsg9E8MFf3aCYivkKA==",
            "rect": {
                "left": 17,
                "top": 55,
                "right": 115,
```

```
                "bottom": 153
            }
        }
    }
```

错误码

错误类型	错误message	错误码	HTTP_CODE
人脸不存在	FACE_TOKEN_NOT_EXIST	104202	400

图片下载

版本

v1

描述

图片内容下载。

系统目前存储的图片一共有两类：

1.新增人脸无论成败都会存储原图，永久存储，用于客户查询入库历史以及查询底库人脸；

2.识别记录无论成败都会存储识别图，默认30天过期自动删除，用于客户查询识别历史。

查询方式也有两种，通过传入不同参数来控制：

1.传入image_id，则返回原图；

2.传入face_token，则返回原图中人脸抠图，目前的抠图规则是：

- 抠图的高度等于2倍rect的高，宽度等于2倍rect的宽；
- 宽度扩大，保持中心点不变；
- 高度扩大，上部扩大3/5，下部扩大2/5（换算一下，中心点向上扩展了1/10）。

请求URI

/api/image/v1/query

请求方法

GET

请求参数(json_body)

参数名	是否必选	参数类型	参数说明
image_id	二选一	String	指定image_id返回原图。 字符串，新增人脸或者人脸识别后生成，图片的唯一标识。
face_token		String	指定face_token返回人脸抠图。 24位字符串，新增人脸成功后由系统生成（参见/face/add），人脸的唯一标识。

说明：如果同时传image_id和face_token，则仅处理face_token。

请求示例

请求image_id:

Request

```
/api/image/v1/query?image_id=9ggf4MCYpavxfdEK8OHjLw==
```

请求face_token:

Request

```
/api/image/v1/query?face_token=z1BgYCmnTbiHTNtX1EeeEA==
```

返回参数

File Stream

创建Group

版本

v1

描述

新增底库。

底库是一个容器的概念，本产品的人脸识别都是基于“一张图”和“一个底库”的比对，而不是暴露直接的1:1的比对接口供用户轮询，原因是算法底层对1:N检索进行了大量的优化，目前的实现方式是综合性价比最高的。

新增人脸只是在系统中加入了一张人脸并抽取特征值（对应的用face_token进行索引），但真正要用于比对，则必须让face_token绑定某个group_name，然后让客户端抓拍图和group_name对应的group进行比较，从而实现1:N的人脸识别检索。

请求URI

/api/group/v1/create

请求方法

POST

请求参数(json body)

参数名	是否必选	参数类型	参数说明

参数名	是否必选	参数类型	参数说明
group_name	必选	String	group的名称，由用户作为参数指定，用于group的增删查操作，也是绑定/解绑face_token的group唯一标识。 参数限制（如果不满足，则报参数错误的通用错误码）： • 支持4-16位长度 • 支持英文小写字母、英文大写字母、数字、半角下划线 • 建议正则：/^[a-zA-Z0-9_-]{4,16}

请求示例

Request

```
{
  "group_name" : "group_a"
}
```

返回参数(json body)

无

返回示例

Response

```
{
    "code": 0,
    "message": "",
    "timestamp": 1510058788332,
    "timecost": 237,
    "data": {
    }
}
```

错误码

错误类型	错误message	错误码	HTTP_CODE
group_name已存在	GROUP_NAME_CONFLICT	104300	400

删除Group

版本

v1

描述

删除底库group，对应的也删除了group和face_token的绑定关系。

请求URI

/api/group/v1/delete

请求方法

POST

请求参数(json body)

参数名	是否必选	参数类型	参数说明
group_name	必选	String	group的名称，由用户作为参数指定，用于group的增删查操作，也是绑定/解绑face_token的group唯一标识。 参数限制（如果不满足，则报参数错误的通用错误码）： • 支持4-16位长度 • 支持英文小写字母、英文大写字母、数字、下划线 • 建议正则：/^[a-zA-Z0-9_-]{4,16}

请求示例

Request

```
{
  "group_name" : "group_a"
}
```

返回参数(json body)

无

返回示例

Response

```
{
    "code": 0,
    "message": "",
    "timestamp": 1510058788332,
    "timecost": 237,
    "data": {
    }
}
```

错误码

错误类型	错误message	错误码	HTTP_CODE
group_name不存在	GROUP_NAME_NOT_EXIST	104301	400

查询Group信息

版本

v1

描述

通过参数group_name，查询group的详细信息。

请求URI

/api/group/v1/query

请求方法

POST

请求参数(json body)

参数名	是否必选	参数类型	参数说明
group_name	必选	String	group的名称，由用户作为参数指定，用于group的增删查操作，也是绑定/解绑face_token的group唯一标识。 参数限制（如果不满足，则报参数错误的通用错误码）： • 支持4-16位长度 • 支持英文小写字母、英文大写字母、数字、下划线 • 建议正则：/^[a-zA-Z0-9_-]{4,16}

请求示例

Request

```
{
  "group_name" : "group_a"
}
```

返回参数

参数名	参数类型	出现情况	参数说明
group_name	String	必有	group的名称，由用户作为参数指定，用于group的增删查操作，也是绑定/解绑face_token的group唯一标识。
faces	Array	必有	当前group下绑定的人脸信息，数组形式给出，如果没有绑定任何人脸，则返回空数组，不缺省，不给NULL。
/faces 数组中的item内容如下			
face_token	String	必有	24位字符串，入库成功后由系统生成，人脸的唯一标识，可以理解为faceID，用户可以通过face_token绑定/解绑底库，也可以用于face的删除和查询操作。

返回示例

Response

```
{
    "code": 0,
    "message": "",
    "timestamp": 1510058788332,
    "timecost": 237,
    "data": {
            "group_name": "group_a",
            "faces": [
                "QDeldd1vlJ1v865FO4dqtg==",
                "OPNt2cUFlrmSgwLDUf4LJg=="
            ]
    }
}
```

错误码

错误类型	错误message	错误码	HTTP_CODE
group_name不存在	GROUP_NAME_NOT_EXIST	104301	400

查询全部Group

版本

v1

描述

查询全部底库group，列出对应的group_name。

我们默认group操作、face操作均是SDK底层操作，而客户会基于SDK在上层提供业务层的face和group列表。

SDK的group和user操作，是为了完成底层核心的数据结构，以完成快速的算法操作，并提供有效的结果给上层。

而上层业务层则更多时候是为了实际的业务逻辑操作，已经GUI交互。

所以，在SDK层面，不建议直接使用/group/list和/face/list进行GUI操作，也不提供诸如游标、翻页等功能。

请求URI

/api/group/v1/list

请求方法

GET

请求参数(json body)

无

请求示例

无

返回参数(json body)

参数名	参数类型	出现情况	参数说明
groups	Array	必有，如果没有查询到group，则给空数组	返回数组列表，每一项为一个group_name字符串，规则如下： • 支持4-16位长度 • 支持英文小写字母、英文大写字母、数字、下划线 • 建议正则：/^[a-zA-Z0-9_-]{4,16} 如果整个系统中一个group都没有，则返回空数组(不返回NULL，也不缺省groups字段）。

返回示例

Response

```
{
    "code": 0,
    "message": "",
    "timestamp": 1510058788332,
    "timecost": 237,
    "data": {
        "groups": [
            "group_a",
            "group_b"
        ]
    }
}
```

错误码

无

绑定Face到Group

版本

v1

描述

绑定一张人脸到底库，以进行识别。

这里汇总一下之前几个核心api的信息到一起，解释一下这块的核心逻辑：

1.“新增人脸”，主要目的是“人脸准入”，确保一张图片符合人脸质量判断需要，可以用于检索。但是这张脸并无法直接被比对，因为所有搜索都是基于group_name的；

2.所有搜索基于group_name，是因为算法底层对这块进行了各种优化（主要是并行计算方面），所以没有暴露直接的face_token操作给用户，而是由算法黑盒比对；

3.人脸的唯一标识是face_token，底库的唯一标识是group_name，人脸url的唯一标识是image_id；

4.为了真正让比对生效，绑定face_token到group_name，再对应的在客户端配置目标为group_name即可；

5.重复绑定不报错，正常再次绑定即可。

请求URI

/api/group/v1/bind

请求方法

POST

请求参数(json body)

参数名	是否必选	参数类型	参数说明
group_name	必选	String	group的名称，由用户作为参数指定，用于group的增删查操作，也是绑定/解绑face_token的group唯一标识。 参数限制（如果不满足，则报参数错误的通用错误码）： • 支持4-16位长度 • 支持英文小写字母、英文大写字母、数字、下划线 • 建议正则：/^[a-zA-Z0-9_-]{4,16}
face_token	必选	String	24位字符串，入库成功后由系统生成（参见/face/add），人脸的唯一标识，可以理解为faceID。 用户可以通过face_token绑定/解绑底库，也可以用于face的删除和查询操作。

请求示例

Request

```
{
  "group_name" : "group_a",
  "face_token" : "GKBj9-l9tIjsUVdZQ4eZqQ=="
}
```

返回参数

无

返回示例

Response

```
{
    "code": 0,
    "message": "",
    "timestamp": 1510058788332,,
    "timecost": 237,
    "data": {
    }
}
```

错误码

无

Group解绑Face

版本

v1

描述

这是 /group/bind的对立操作，解绑一张人脸和底库的绑定关系，更严格的说，是face_token和group_name的绑定关系。

解绑之后，对目标group_name的搜索就不再包含此face_token，但是目前支持门禁机配置多个group，所以其他group_name会继续识别（因为没有解绑）。

所以如果目的是为了让门禁机不再识别某个人脸，则请务必确保该门禁机上所有配置的group都完成了解绑操作。

如果对应的绑定关系不存在，则需要报错，解绑失败。

请求URI

/api/group/v1/unbind

请求方法

POST

请求参数(json body)

参数名	是否必选	参数类型	参数说明
group_name	必选	String	group的名称，由用户作为参数指定，用于group的增删查操作，也是绑定/解绑face_token的group唯一标识。 参数限制（如果不满足，则报参数错误的通用错误码）： • 支持4-16位长度 • 支持英文小写字母、英文大写字母、数字、下划线 • 建议正则：/^[a-zA-Z0-9_-]{4,16}
face_token	必选	String	24位字符串，入库成功后由系统生成（参见/face/add），人脸的唯一标识，可以理解为faceID。 用户可以通过face_token绑定/解绑底库，也可以用于face的删除和查询操作。

请求示例

Request

```
{
  "group_name" : "group_a",
  "face_token" : "GKBj9-l9tIjsUVdZQ4eZqQ=="
}
```

返回参数

无

返回示例

Response

```
{
    "code": 0,
    "message": "",
    "timestamp": 1510058788332,
    "timecost": 237,
    "data": {
    }
}
```

错误码

错误类型	错误message	错误码	HTTP_CODE
绑定关系不存在	GROUP_BINDING_NOT_EXIST	104303	400

人脸识别

版本

v1

描述

人脸识别识别请求。
这条协议是由客户端发起，向服务器端请求人脸识别结果。
请求参数包括目标群组的名称，必要的信息字段，和对应的图片内容。
请配合客户端SDK的FacePassHandler的feedFrame函数阅读。

请求URI

/api/service/recognize/v1/

请求方法

POST

请求参数(multipart/form-data)

是否必选	参数名	参数类型	参数说明
必选	group_name	String	人脸搜索的group名字。 在服务器端通过/group/create/时由客户指定生成，用于group的增删查操作，也是绑定/解绑face_token的group唯一标识。
必选	face_data	String	将Message对象进行JSON编码再进行序列化以后的结果。 这个序列化传输包含了人脸识别所需的相关参数，一些客户端处理的中间结果、阈值、以及部分控制参数。 这个字段来源是feedFrame的返回值，代表了一帧中的人脸信息。注意，如果一帧中有多张人脸，则一个Message会同时包含多个人脸信息，建议将所有人脸图片都通过image_${track_id}参数传入，不同的人脸将根据track_id进行结果合并。
必选	image_${track_id}	File	图片内容。 这个字段和face_data配套使用，对应的是一帧人脸检测出来的人脸图片信息，同样会依据track_id进行结果合并。

请求示例

form表单形式，请参考实际demo程序。

返回参数

返回参数同样是序列化之后的结果，需要在客户端进行解析。

返回示例

Response


```
{
    "code": 0,
    "message": "",
    "timestamp": 1510058788332,
    "timecost": 237,
    "data": "7zvOjop716X7LGnVrIRFfD___bFWfmxzux4psUZJiGTniz5zV04pTE7YYorro5TSY9nqaU5XhcPcv1PjnjtbutJ1_mZZVSD2If5AamVmLtgwnOyh9wxje4oH8b8VHoL2aU9Bg6L8aI5Vpfe3R_RpQ6A65lvNPRltB0QTX
}
```


错误码

错误类型	错误message	错误码	HTTP_CODE
group_name不存在	GROUP_NAME_NOT_EXIST	104301	400

人脸比对

版本

v1

描述

传入两张图，进行1:1比对。
给入的两张图会进行人脸检测，同时如果开启了活体判断参数，则会同时进行活体检测。
人脸检测失败，则会报错（不符合标准的照片进行compare没有意义，我们也无法保障结果具有实际意义）。
如果以上环节均正确，则会返回一个compre的分值，用于判断是否通过。
分值在[0，100]之间，越高表示越相似（同一个人），越低表示越不相似（不同的两个人）。
通常做法是分值会和一个阈值进行比较，这里一般的建议阈值是70。
说明，和一般的图片使用规则不同，compare接口目前不对图片质量进行严格的校验，原因是这是一个很底层的API，客户会用于各种复杂场景，很难给出有效的质量体系。
所以我们的策略是，API返回值中给出各种细节的参数，如果用户可以自行限制质量体系（并且无视我们的比对分值）。

请求URI

/api/service/compare/v1

请求方法

POST

请求参数(form)

参数名	是否必选	参数类型	参数说明
image_1	必选	binary file	用于比对的两张图，支持jpg, png, bmp图片格式。 **图片属性限制：** 1.分辨率限制： • 建议尺寸720p：1280*720 • 支持最小尺寸：必须大于FaceMin，所以必须大于等于50*50。 • 支持最大尺寸：2560*2560 • 分辨率约束的是“面积”，所以长宽比很奇怪的图有可能通过，但前提是其中的人脸满足最小脸像素要求。 • 如果越界，则抛出FacePassException 信息 "INVALID_IMAGE_RESULUTION”。 2.图片控制符限制： • 我们会读取图片控制符中的如下字段： • 旋转：如果图片本身设置了旋转，我们会忠实的按照旋转后的图片来检测人脸，如果参数有误，则有可能导致人脸翻转或者角度过大导致无法检测入库，所以请务必保证输入图片源的该字段表意正确。

参数名	是否必选	参数类型	参数说明
image_2			3.其他： • 底图的选择总体目标是“清晰且和使用者本人一致” • 不建议黑白图 • 不建议使用强烈ps或者艺术化的图 • 建议尽量使用近期生活照，不建议使用跨年龄太久的照片 • 是否化妆、是否戴眼镜，尽量和日常使用习惯一致 **人脸属性限制：** 1:1场景下图片输入来源复杂，所以这个场景下不进行质量判断。 返回值中会给出详细的质量分数（3D-Pose和Blurness），如果确实需要限制，客户可以自行判断。
enable_liveness	必选	boolean	是否判断活体。 说明：因为活体检测比较占用资源，所以在高频使用的时候建议置为false。 另外，如果Bitmap本身没有被检测到人脸，会直接报错，不再浪费资源进行活体判断。

返回参数(json body)

参数名	是否必选	参数类型	参数说明
score	必选	Float	1:1 compare的分值，（0,100]的float。 只有在result = 0 的时候，这个值才会被赋予 > 0 的数值，否则为缺省值 0。 分值越高表示越相似（同一个人），越低表示越不相似（不同的两个人）。 通常做法是分值会和一个阈值进行比较，建议阈值参见compare_threshold字段。
compare_threshold	必选	Float	1:1 compare的建议阈值，(0, 100]的Float。 如果客户不希望采用SDK的建议阈值，也可以自行设定，一般的原则是： 通过率 = 本人比本人正确的次数 / 本人比本人的总次数； 误识率 = 本人比他人正确的次数 / 本人比他人的总次数； 阈值提高，通过率下降，误识率也下降，直观上说，就是自己比自己更容易比不过了，但是自己比他人也更不容易出错了。 客户理论上可以根据实际需要调整自己的使用阈值，如有困难，可以咨询支持人员。
detection_result_1 detection_result_2	必选	Dictionary	/service/compare接口的前两个参数image1和image2的人脸检测结果，这是一个字典结构，包含rect和pose和blur。 “rect”：表示image1/image2中的人脸框位置，如果是检测失败（即result = 1的情形），则置为NULL，否则为left、top、right、bottom的字典数据结构； “pose”/"blur"：分别表示角度和模糊度；这里需要说明的是，这两个质量判断并不影响返回值。原因如前文描述，这个API的应用场景过于宽泛，且很多场景有人值守、可做时候校验，所以普遍希望API能够尽可能放宽要求。一个典型的实际例子是身份证读卡器的输出照片，质量其实比较差（blur可能很不理想），但是我们依旧希望能过通过质量判断体系进行最终的比对。所以说，我们默认不卡角度和模糊度，如果用户真的需要卡，可以二次开发，针对返回值中的这两个参数进行更上层的业务判断。
liveness_threshold	必选	Float	活体阈值，Float型，服务器内部配置的阈值参数，当前版本可以通过/service/config进行这个阈值的配置。
liveness_score1 liveness_score1	必选	Float	/service/compare接口的前两个参数image1和image2的活体检测结果，Float型，越大表示越是真人，越小表示越像活体攻击。 liveness_score1和liveness_score2的任意一个低于liveness_threshold，就意味着活体检测失败，对应的 result = 2 。
/detection_result_1 和 /detection_result_2			
rect	必选	Dictionary	人脸框数组，左上角为(0,0)，四个值为left、top、right、bottom，均为Int型，不含小数点。 通常我们建议rect能大于 50*50，再低可能会引起compare结果的不可控。
pose	必选	Dictionary	人脸角度数组，四个值为yaw、pitch、roll，均为Float型，用角度值表示。 0表示没有偏转，正负表示两个不同的方向，通常我们都建议三个值落在±20°以内，如果过大可能引起compare结果的不可控。 • roll：以鼻子为轴、脸部平面旋转； • yaw：左右摇头； • pitch：上下点头。
blur	必选	Float	人脸模糊程度，[0,1]之间的Float，越大越模糊，通常我们建议 blur <= 0.2f，如果过大可能引起compare结果的不可控。

Response返回示例


```
{
    "code": 0,
    "message": "",
    "timecost": 433,
    "timestamp": 1519639954744,
    "data": {
        "compare_threshold": 69.101395,
        "score": 33.288754,
        "detection_result_1": {
            "rect": {
                "bottom": 604,
                "left": 149,
                "right": 448,
                "top": 305
            },
            "pose": {
                "yaw": -4.085426,
                "pitch": 12.494804,
                "roll": 0.4202854
            },
            "blur": 0.0036446378
        },
        "detection_result_2": {
            "rect": {
                "bottom": 715,
                "left": 168,
                "right": 639,
                "top": 244
            },
            "pose": {
                "yaw": -4.5025926,
                "pitch": 11.365435,
                "roll": 3.101541
            },
            "blur": 0.003414875
        },
        "liveness_threshold": 70,
        "liveness_score_1": 0,
        "liveness_score_2": 0
    }
}
```

错误码

错误类型	错误message	错误码	HTTP_CODE
没有检测到人脸	FACE_NOT_FOUND	104200	400
活体检测失败	COMPARE_LIVENESS_FAILED	104901	400
图片校验失败	IMAGE_VALIDATE_FAILED	104103	400

底库同步

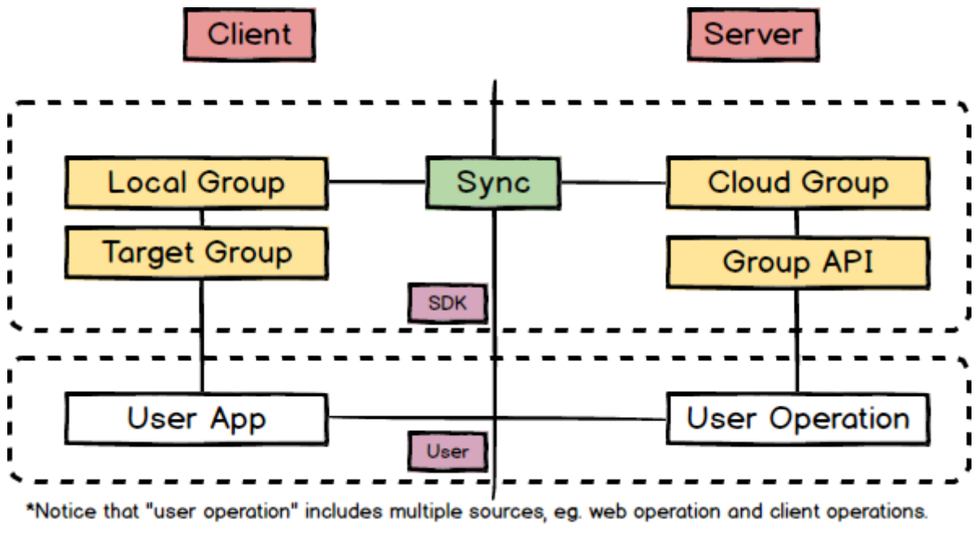
版本

v1

描述

注意：以下逻辑仅针对“云离线模式”有效，和抓拍模式和脱机模式无关。
如果想全面了解三种模式的使用，请参加FacePass的产品介绍文档。

核心逻辑如下：



严格意义上讲，我们说的“同步”并非真正意义上的同步，而是“云端统一copy下发”，核心原则是：

- 唯一性：云端有一个唯一的copy，是其他所有的标准，用户能修改的也仅是云端copy，不能直接修改客户端。
- 单向性：只允许云端覆盖客户端，不允许客户端覆盖云端，所以也不存在多个客户端同时修改造成的merge。
- 实时性：当云端发生梗概的时候，需要尽可能快的同步到本地客户端。

下面描述一下同步的几个关键点：

1.什么时候触发同步?

回答：当云端和客户端的group版本不一致的时候发生。

客户端会发起一条协议/service/sync，带上本地的group的name、id和version。

groupName是客户在本地指定的，用于检索的底库；

groupId是sync时真正的索引（确保唯一性），本地刚指定groupName的时候是不带Id的，同步完一次就会用云端groupId覆盖本地，后续sync就一直带上Id了；

version是本地具有groupId的时候，判断云和端版本是否一致的参数。

所以两种情况下会触发同步：

a.本地没有groupId（因为新建的时候指定的是groupName）；

b.本地有groupId，但是本地的groupVersion和云端对应的groupVersion不同；。

2.sync做了什么?

a.如果是本地没有groupId，则云端下发该groupName对应的groupId和全部facetoken，对应覆盖到本地；

b.如果本地有groupId，则云端比较local的version和云端的version，如果version一致则无增量变化；

如果不一致，如果云端版本更更高，则云端merge最近这两个版本之间的增量覆盖或者直接用云端全量覆盖版本（local version过老，云端无法查询）。

增量的部分会作为返回值（/service/sync)发给客户端，客户端调用handleSyncResultData函数进行本地增量更新。

3.云端什么情况下会升版本号?

- group被创建；
- group被删除；
- group中的face_token发生了bind；
- gorup中的face_token发生了unbind；

4.综合云+端说一下具体的同步策略?

如果不清楚target/local/cloud三个group层的概念，请参见第二章底库管理部分。

target group的调整(get/set函数)是一个同步（非异步）操作，所以默认和local group是一致的。

但是local group和cloud group当然会因为version不一致等原因出现差异，具体逻辑如下：

（注：sync是根据local group的，所以不存在“local不存在但是cloud存在”的状态）

场景	客户端			服务器		策略
	local group	group id	version	cloud group	version	
客户端配置里新加了一个groupName	有	无	无	有	有	等于全量更新group信息到本地，覆盖local version
	有	无	无	无	无	不报错，不删除local，保留local version
云端删除了目标group	有	有/无	有/无	无	无	不报错，不删除local，保留local version
云端某个group增删了faceToken	有	有/无	有/无	有	有且一致	什么都不做
	有	有/无	有/无	有	有，不一致	更新本地copy为云端，覆盖local version（即使local更大）

5.sync是什么形式的协议?

http的get请求而已。处于简化，我们建议实际的sync都是客户端发起的。即使有能力做长连，那么长连也仅仅是通知客户端需要更新，从而客户端主动发起一次协议而已。

6.说一下协议是什么，怎么发?

我们对FacePassHandler提供了两个新的函数：

a.FacePassSyncRequestData getSyncRequestData()，这个函数用于发http协议是封包，去掉用/service/sync。

b.void handelSyncResultData(String syncResultData)，这里的syncResultData是/service/sync的返回值，

7.那么发起sync的场景是什么呢?

理论上讲，我们提供了以上两个接口+一条云端API之后，客户随时可以发起sync，无限可能，这里我们枚举集中典型场景：

- A.SDK自己实现一套心跳机制，隔一段时间sync一次；（最简单，但是有延时，体验不好）
- B.用户自己实现一套心跳机制，捎带上SDK提供的getSyncRequestData，隔一段时间sync一次；（SDK更简单，但是客户需要麻烦一点）
- C.SDK自己实现一套推送机制，云端有更新则推送客户端，客户端接到通知则sync；（长连有点麻烦，还有适配风险；做精致了需要云端有IP到groupname的映射表，糙的话全推送）
- D.用户自己实现一套推送机制，携带下云端更新的版本号，客户端接到通知则sync；（能省掉一个长连，对SDK来说非常简单，但是客户需要麻烦一点）。

8.升级的时候有什么要注意的吗?

以上提到的所有内容，都是默认info/search_model对云和端是一致的。如果不一致，一般是升级过程中会出现，请参见升级策略。

发送sync协议的时候是会带上模型版本的，如果模型版本和云端的实际模型版本不一致，则会视为“所有feature需要被更新”，云端下发的syncResult会要求删除本地所有feature/face_token数据，然后全部换成新模型的数据。

9.如果混用“云离线模式”和“脱机模式”，会有什么结果?

当前版本在客户端SDK提供了直接修改客户端底库的API，如有需要，客户可以无视云端而在本地完成完整的底库管理。

通常情况下我们不建议两者混用，但是如果真的使用了，原则是“云端无条件覆盖本地”，技术上说，任何本地的修改都会把本地底库的version置为0，一旦发起/service/sync，就会因为version的滞后而被无条件覆盖了。

请求URI

/api/service/sync/v1

请求方法

POST

请求参数(json body)

参数名	是否必选	参数类型	参数说明
info	必选	Dictionary	信息字段，用于存放一些控制信息，当前版本仅支持识别模型版本号。
groups	必选	List<Group>	底库信息，同步的核心内容，主要是group名字和版本号，供云端决策。
/info			
search_model	必选	String	算法模型版本，如果算法模型版本和云端不一致，则会下发用新模型抽取的特征值，下发到本地。 此操作目前仅在升级过程中遇到，日常默认不会发生。
version	必选	String	客户端的版本号
Group			
group_id	可选	Long	实际sync的索引（不是group_name）。 当本地新建一个group的时候，是没有group_id的，这是默认云端根据group_name全量下载； 经过一次sync后，本地就会缓存该group_name对应的group_id，后续再次针对这个group的sync就都会带上了。
group_name	必选	String	如果请求没有传递group_id, 则使用group_name查询最新的group_id, 并进行全量同步
local_version	必选	Long	客户端本地group的版本，long型，>=1，有内容的版本从1开始计数递增，在云端生成。 修改仅有两个来源： 1.本地通过setLocalGroup新建一个本地库的时候，version为0； 2.sync成功之后，会用cloud_version来覆盖local_version。

请求示例

Request

```
{
  "info": {
    "search_model": "1017",
    "version": "v1.0.0"
  },
  "groups": [
    {
      "group_name": "group_a",
      "group_id": 0,
      "local_version": 0
    }
  ]
}
```

返回参数

返回参数为二进制码流, 供客户端Handler直接解析, 没有设计成直接的json结构，是因为一个feature大约4k，2000人底库就是8M，json解析过慢。

返回示例

Response

二进制流

错误码

无

底库同步状态查询

版本

v1

描述

查询门禁机设备的底库同步状态，仅在云离线模式下有效，本质上讲是调用/service/sync/协议后的sync状态。

返回值是一个长列表，每一项是一个设备，每个设备包含同步状态细节。

请求URI

/api/service/sync/v1/status

请求方法

GET

请求参数

无

返回参数(json body)

参数名	参数类型	出现情况	参数说明
devices	Array	必有	如果什么设备都没有，给空数组
/devices			

参数名	参数类型	出现情况	参数说明
ip	String	必有	device的IP地址。 用户实际上会维护自己的设备列表，但是并没有和SDK的同步机制，目前先提供IP，在必要的时候可以做关联。
version	String	必有	客户端版本，这里指的是客户端的版本。
sync_status	Dictionary	必有	同步的状态，详细参见具体value说明。 这里着重说明一下，由于sync过程可能是客户自己手动操作的，SDK本身并不控制http层， 所以，云端记录的是最新的一次同步记录返给客户端的数据，而不是客户端的最终结果。 如果要求本协议实时体现客户端最终状态，则需要客户自行增加某种握手策略。 最简单的方式是在同步完成之后再无脑sync一次，直到确保云端不再返回增量更新内容（云和端的版本完全一致了）。
/sync_status			
sync_time	Unix时间戳，精确到毫秒	必有	最近一次同步发生的时间。
info	Dictionary	必有	信息字段，用于存放一些控制信息，当前版本仅支持识别模型版本号。
groups	List<Group>	必有	底库信息，同步的核心内容，主要是group名字和版本号，供云端决策。 字典的key是groupName，不做额外说明，后面仅描述value部分，具体参见示例。
/info			
search_model	String	必有	算法模型版本，如果算法模型版本和云端不一致，则会下发用新模型抽取的特征值，下发到本地。 此操作目前仅在升级过程中遇到，日常默认不会发生。
Group			
group_name	String	必有	如果请求没有传递group_id, 则使用group_name查询最新的group_id, 并进行全量同步
group_id	Long	必有	实际sync的索引（不是group_name）。 当本地新建一个group的时候，是没有group_id的，这是默认云端根据group_name全量下载； 经过一次sync后，本地就会缓存该group_name对应的group_id，后续再次针对这个group的sync就都会带上了
local_version	Long	必有	客户端本地group的版本，long型，>=1，有内容的版本从1开始计数递增，在云端生成。 修改仅有两个来源： 1.本地通过setLocalGroup新建一个本地库的时候，version为0； 2.sync成功之后，会用cloud_version来覆盖local_version。
cloud_version	Long	必有	客户端本地group的版本，long型，>=1，有内容的版本从1开始计数递增，在云端生成。 如果云端不存在这个group，则version为0； 如果云端新建了一个group，且name不重复，version = 1； 如果云端新建了一个group，且name重复（但是被删除了），version ++； 如果云端发生了group/bind和/unbind，version ++。
result_code	Int	必有	0=成功，1=失败，注意，这个指的是操作本身成败，并不严格对应于“version改变”这个事实。
result_version	Long	必有	result_version = result_code=0 ? cloud_version : local_version。
result	Dictionary	必有	实际增删facetoken的结果，如果没有发生任何变化，不为NULL，不缺省，内部字段也依旧存在。
/groups/result			
facetoken_added	Int	必有	本次sync增加的facetoken数量，如果没有新增则为0，不为NULL，不缺省。
facetoken_deleted	Int	必有	本次sync删除的facetoken数量，如果没有新增则为0，不为NULL，不缺省。

返回示例

Response

```
{
  "code": 0,
  "message": "",
  "timestamp": 1510058788332,
  "timecost": 237,
  "data": {
    "devices": [
      {
        "ip": "192.168.0.1",
        "version": "v1.0.1",
        "sync_status": {
          "sync_time": 15038483743000,
          "info": {
            "search_model": "1017"
          },
          "groups": [
            // 注意，这里云端并无法保证客户端完成了同步，所以是记录了最后一次同步的信息，而已
            {
              "group_name": "group_a",
              "group_id": 100,
              "local_version": 10001,
              "cloud_version": 10001,
              "result_code": 0,          // 0=成功，1=失败，注意，这个指的是操作本身成败，并不严格对应于“version改变”这个事实
              "result_version": 10001,  // result_version = result_code=0 ? cloud_version : local_version
              "result": {
                "facetoken_added": 0,
                "facetoken_deleted": 0 // 这里屏蔽掉了具体的feature，因为量很大，只返回修改的int出来
              }
            }
          ]
        }
      }
      // log上传v1.0.0不做，由于是新字段能向前兼容。
    ]
  }
}
```

错误码

无

阈值配置

版本

v1

描述

客户可以对本接口进行服务器端阈值的配置，具体包括1:N人脸识别、1:1人脸比对、活体这三个阈值。

这个配置仅对服务器识别的模式生效，也就是所谓的“抓拍模式”下生效。

生效的方式是，识别之后的返回值，会同时带上当前配置的阈值，客户端本身只是根据得分和阈值的比较来得出结果。

客户端SDK也会配置一套阈值参数，但是仅是针对客户端本地识别使用，和本API配置的服务器阈值是独立的。

请求URI

/api/service/config/v1

请求方法

POST

请求参数(form)

参数名	是否必选	参数类型	参数说明
recognize_threshold	可选	Float	需要进行阈值配置的参数。 有两种使用方法： • 传入(0, 100]区间的float值，则代表修改阈值至指定参数值； • 传入-1，则代表恢复阈值为系统默认配置。 三个阈值参数相互独立，但是如果有一个不在参数允许范围内就会整体报错。 阈值为空或者缺省表单参数是，系统不做任何处理。 阈值修改会在下次请求时生效。 推荐阈值： recognize_threshold: 78.0f compare_threshold: 70.0f liveness_threshold: 50.0f
compare_threshold			
liveness_threshold			

返回参数(json body)

参数名	是否必选	参数类型	参数说明
recognize_threshold	必选	Float	1:N人脸识别的阈值， 建议阈值78
compare_threshold	必选	Float	1:1人脸比对的阈值，建议阈值70
liveness_threshold	必选	Float	活体检测阈值，建议阈值50

Response返回示例


```
{
    "code": 0,
    "message": "",
    "timecost": 173,
    "timestamp": 1519872450463,
    "data": {
        "compare_threshold": 78,
        "liveness_threshold": 70,
        "recognize_threshold": 50
    }
}
```


错误码

错误类型	错误message	错误码	HTTP_CODE
参数错误	ILLEGAL_ARGUMENT	104001	400

版本查询

版本

v1

描述

查询当前服务器软件版本。

这个API默认客户没有实际的使用价值，主要用于问题定位时查询版本号。

请求URI

/api/version/v1

请求方法

GET

请求参数

无

返回参数(json body)

参数名	参数类型	出现情况	参数说明
version	String	必有	当前服务器版本，默认三位"v1.0.0"格式。
detail	Dictionary	必有	其他模型版本，内容无固定格式，以每个版本的实际需要为准。 不可缺省，不为NULL。

返回示例

Response

```
{
    "code": 0,
    "message": "",
    "timestamp": 1510058788332,
    "timecost": 237,
    "data": {
        "version": "v1.0.1",
        "detail": {
          "liveness": "panorama.koala.1107",
          "feature": "large.v4"
        }
    }
}
```


错误码

无