

人脸识别服务

API 参考

文档版本 01
发布日期 2025-01-20



版权所有 © 华为云计算技术有限公司 2025。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HUAWEI和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

地址：贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址：<https://www.huaweicloud.com/>

目录

1 使用前必读.....	1
1.1 概述.....	1
1.2 调用说明.....	1
1.3 终端节点.....	1
1.4 约束与限制.....	2
1.5 基本概念.....	3
2 API 概览.....	5
3 如何调用 API.....	7
3.1 申请服务.....	7
3.2 构造请求.....	8
3.3 认证鉴权.....	10
3.4 返回结果.....	12
4 API.....	15
4.1 人脸检测.....	15
4.2 人脸比对.....	25
4.3 活体检测.....	32
4.3.1 动作活体检测.....	32
4.3.2 静默活体检测.....	40
4.4 人脸搜索.....	47
4.5 人脸库资源管理.....	54
4.5.1 创建人脸库.....	55
4.5.2 查询所有人脸库.....	61
4.5.3 查询人脸库.....	66
4.5.4 删除人脸库.....	71
4.6 人脸资源管理.....	75
4.6.1 添加人脸.....	75
4.6.2 查询人脸.....	83
4.6.3 更新人脸.....	88
4.6.4 删除人脸.....	93
4.6.5 批量删除人脸.....	98
5 公共数据结构.....	104
5.1 自定义字段.....	104

5.2 sort 语法.....	105
5.3 filter 语法.....	105
5.4 消息对象结构.....	109
5.4.1 AllParam.....	109
5.4.2 DetectFace.....	112
5.4.3 Landmark.....	112
5.4.4 Attributes.....	112
5.4.5 FaceQuality.....	114
5.4.6 FaceExpression.....	115
5.4.7 FaceSetFace.....	115
5.4.8 SearchFace.....	116
5.4.9 FaceSetInfo.....	116
5.4.10 BoundingBox.....	117
5.4.11 VideoDetectResult.....	117
5.4.12 LivelessDetectResult.....	118
5.4.13 ServiceInfo.....	119
5.4.14 WarningList.....	119
6 高频报错处理办法.....	121
7 附录.....	123
7.1 状态码.....	123
7.2 错误码.....	125
7.3 获取项目 ID/账号名/AK/SK.....	132
7.3.1 获取项目 ID/账号名.....	132
7.3.2 获取账号 ID.....	133
7.3.3 获取 AK/SK.....	133

1 使用前必读

1.1 概述

欢迎使用人脸识别服务（Face Recognition Service，简称FRS），该服务能够在图像中快速检测人脸、分析人脸关键点信息、获取人脸属性、实现人脸的比对和检索。

人脸识别服务以开放API的方式提供给用户，您可以根据本文档提供的API来使用服务。

在调用人脸识别服务API之前，请确保已经充分了解人脸识别服务相关概念，详细信息请参见人脸识别产品介绍。

1.2 调用说明

人脸识别服务提供了RESTful API，支持您通过HTTPS请求调用，调用方法请参见[如何调用API](#)。

同时人脸识别服务还提供多种编程语言的SDK供您使用。

1.3 终端节点

终端节点（Endpoint）即调用API的[请求地址](#)，不同服务不同区域的终端节点不同，您可以从[地区和终端节点](#)中查询所有服务的终端节点。

人脸识别服务的终端节点如[表1-1](#)所示，请您根据业务需要选择对应区域的终端节点。

表 1-1 人脸识别服务终端节点

区域名称	区域	终端节点（Endpoint）	协议类型	部署的服务
华北-北京四	cn-north-4	face.cn-north-4.myhuaweicloud.com	HTTPS	人脸检测，人脸比对，人脸搜索，静默活体检测，动作活体检测

区域名称	区域	终端节点 (Endpoint)	协议类型	部署的服务
华东-上海一	cn-east-3	face.cn-east-3.myhuaweicloud.com	HTTPS	人脸检测, 人脸比对, 人脸搜索
华南-广州	cn-south-1	face.cn-south-1.myhuaweicloud.com	HTTPS	人脸检测, 人脸比对, 人脸搜索, 动作活体检测
中国-香港	ap-southeast-1	face.ap-southeast-1.myhuaweicloud.com	HTTPS	人脸检测、人脸比对、人脸搜索
亚太-曼谷	ap-southeast-2	face.ap-southeast-2.myhuaweicloud.com	HTTPS	人脸检测、人脸比对、人脸搜索

1.4 约束与限制

受技术与成本多种因素制约, 人脸识别服务存在一些约束限制。其中系统级约束限制, 是所有子服务的约束。除系统级约束限制外, 各子服务还有独立的约束条件。

系统级约束限制

- 只支持识别JPG、PNG、JPEG、BMP格式的图片。
- 每个用户可免费使用10个人脸库, 每个人脸库容量为10万个人脸特征。如需扩容单个人脸库规模, 请联系华为云客服确认扩容规模与价格。
- application/json请求的body中, 请使用标准Json格式。
- Base64编码中请勿使用回车换行。
- 系统不保存用户图片或视频。

人脸检测/比对/搜索

- 人脸比对输入的两张图片总大小小于**8MB**。
- 图片大小小于**8MB**, 由于图片过大会导致图片在网络传输过程中耗时较长, 建议小于**1MB**。
- 图片分辨率小于**4096*4096**, 图片中人脸像素大于**80*80**, 建议**120*120**以上。
- 为保证识别效果, 人脸图片建议要求如下:
 - a. 光照大于200lux、无反光强光阴影现象。
 - b. 人脸无遮挡、整体清晰无拖尾抖动等运动模糊。
 - c. 侧脸不超过30°、俯仰角小于15°、偏转角小于15°、图片中人脸保持竖置正脸。

动作活体检测

- 目前支持检测视频文件，或视频的Base64编码，不支持直接检测视频流，需要用户客户端自己获取视频流并保存成文件，然后调用活体检测接口。
- 视频文件大小不超过8MB，建议客户端压缩到200KB~2MB。
- 限制视频时长1~15秒。
- 建议帧率10fps~30fps。
- 封装格式：mp4、avi、flv、webm、asf、mov。
- 视频编码格式：h261、h263、h264、hevc、vc1、vp8、vp9、wmv3。

静默活体检测

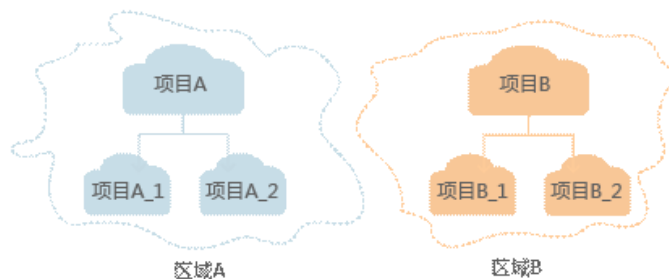
- 图片大小小于8MB，由于过图片过大会导致图片在网络传输过程中耗时较长，建议小于1MB。
- 图片分辨率小于4096*4096，图片中人脸像素大于40*40，建议120*120以上。
- 为保证识别效果，人脸图片建议要求如下：
 - a. 光照大于200lux、无反光强光阴影现象。
 - b. 人脸无遮挡、整体清晰无拖尾抖动等运动模糊。
 - c. 侧脸不超过30°、俯仰角小于15°、偏转角小于15°、图片中人脸保持竖置正脸。

1.5 基本概念

- 账号
用户注册华为云时的账号，账号对其所拥有的资源及云服务具有完全的访问权限，可以重置用户密码、分配用户权限等。由于账号是付费主体，为了确保账号安全，建议您不要直接使用账号进行日常管理工作，而是创建用户并使用他们进行日常管理工作。
- 用户
由账号在IAM中创建的用户，是云服务的使用人员，具有身份凭证（密码和访问密钥）。
在[我的凭证](#)下，您可以查看账号ID和用户ID。通常在调用API的鉴权过程中，您需要用到账号、用户和密码等信息。
- 区域（Region）
从地理位置和网络时延维度划分，同一个Region内共享弹性计算、块存储、对象存储、VPC网络、弹性公网IP、镜像等公共服务。Region分为通用Region和专属Region，通用Region指面向公共租户提供通用云服务的Region；专属Region指只承载同一类业务或只面向特定租户提供业务服务的专用Region。
详情请参见[区域和可用区](#)。
- 可用区（AZ，Availability Zone）
一个AZ是一个或多个物理数据中心的集合，有独立的风火水电，AZ内逻辑上再将计算、网络、存储等资源划分成多个集群。一个Region中的多个AZ间通过高速光纤相连，以满足用户跨AZ构建高可用性系统的需求。
- 项目
华为云的区域默认对应一个项目，这个项目由系统预置，用来隔离物理区域间的资源（计算资源、存储资源和网络资源），以默认项目为单位进行授权，用户可

以访问您账号中该区域的所有资源。如果您希望进行更加精细的权限控制，可以在区域默认的项目中创建子项目，并在子项目中购买资源，然后以子项目为单位进行授权，使得用户仅能访问特定子项目中资源，使得资源的权限控制更加精确。

图 1-1 项目隔离模型



2 API 概览

人脸识别服务所提供的API，均符合RESTful API设计规范，如表2-1所示。

表 2-1 人脸识别 API

类型	API	说明
人脸检测	人脸检测	人脸检测是对输入图片进行人脸检测和分析，输出人脸在图像中的位置、人脸关键属性。
人脸比对	人脸比对	人脸比对是将两个人脸进行比对，来判断是否为同一个人，返回比对置信度。如果传入的图片中包含多个人脸，选取最大的人脸进行比对。
活体检测	动作活体检测	通过判断视频中的人物动作与传入动作列表是否一致来识别视频中人物是否为活体。如果有多张人脸出现，则选取最大的人脸进行判定。
	静默活体检测	基于人脸图片中可能存在的畸变、摩尔纹、反光、倒影、边框等信息，判断图片中的人脸是否来自于真人活体，有效抵御纸质翻拍照、电子翻拍照以及视频翻拍等各种攻击方式。静默活体检测支持单张图片，不支持多人脸图片。
人脸搜索	人脸搜索	人脸搜索是指在已有的人脸库中，查询与目标人脸相似的一张或者多张人脸，并返回相应的置信度。
人脸库资源管理	创建人脸库	创建用于存储人脸特征的人脸库。您最多可以创建10个人脸库，每个人脸库最大容量为10万个人脸特征。如有更大规格的需求请联系客服。
	查询所有人脸库	查询当前用户所有人脸库的状态信息。
	查询人脸库	查询人脸库当前的状态。

类型	API	说明
	删除人脸库	删除人脸库以及其中所有的人脸。人脸库数据为用户隐私数据，该数据无备份，删除时请谨慎操作。
人脸资源管理	添加人脸	添加人脸到人脸库中，检测到传入的单张图片中存在多少张人脸，则增加多少张人脸到人脸库当中。
	查询人脸	查询指定人脸库中人脸信息。
	更新人脸	根据人脸ID (face_id) 更新单张人脸信息。
	删除人脸	根据指定字段删除人脸库中人脸。
	批量删除人脸	自定义筛选条件，批量删除人脸库中的符合指定条件的多张人脸。

3 如何调用 API

3.1 申请服务

在调用API之前，必须先申请开通服务，下面是申请服务的相关步骤。

📖 说明

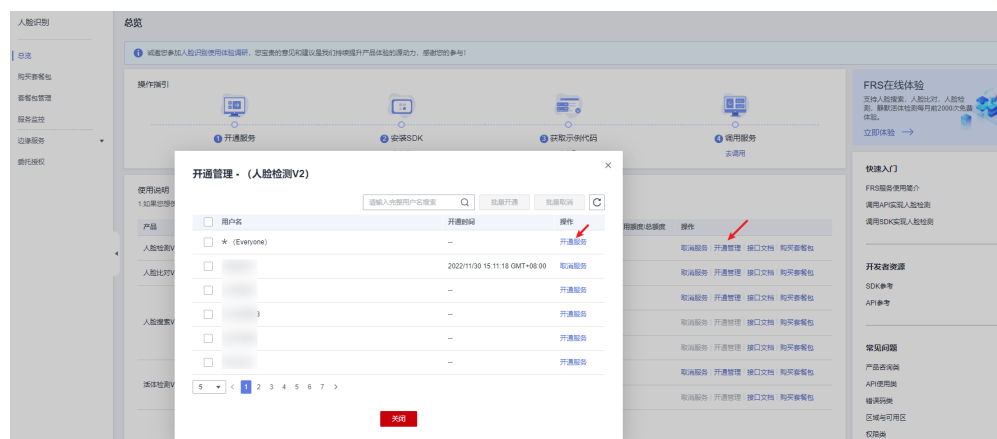
申请服务前，必须先注册云账号，并完成实名认证。

申请步骤

1. 登录[人脸识别管理控制台](#)。
2. 根据业务需求，选择服务部署区域，开通所需服务（例如：人脸检测），单击右侧的“开通服务”。

香港、曼谷区域API也可以使用中国站账号进行开通；如您需要使用国际站账号开通香港、曼谷区域API，请联系[客服](#)获取开通支持。

图 3-1 开通服务



说明

- 由于应用可能需要使用对象存储服务（OBS）中的数据，人脸识别服务需要您授权可以操作对象存储服务。单击左侧“委托授权”，完成OBS授权，已授权过的服务，该页面提示“已授权”。
- OBS授权时，如果提示委托已达上限，则需要您登录到[统一身份认证服务](#)，对委托进行删除或创建新的委托。
- 服务状态显示“已开通”时，即可调用服务的API。

图 3-2 服务授权



3.2 构造请求

本节介绍REST API请求的组成，并以调用人脸检测为例说明如何调用API，您还可以通过这个视频教程了解如何构造请求调用API：<https://bbs.huaweicloud.com/videos/102987>。

请求 URI

请求URI由如下部分组成。

{URI-scheme}://{Endpoint} / {resource-path} ? {query-string}

表 3-1 请求 URI

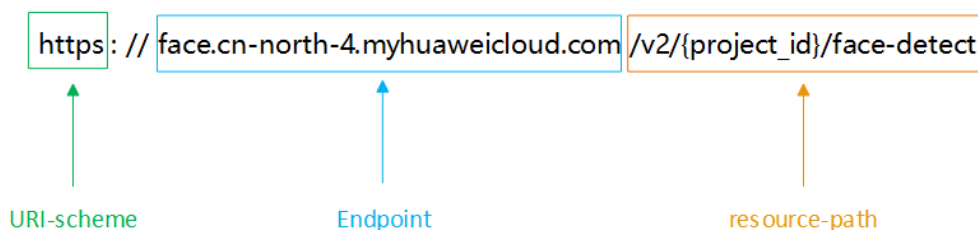
参数	说明
URI-scheme	传输请求的协议，当前所有API均采用HTTPS协议。
Endpoint	承载REST服务端点的服务器域名或IP，不同服务在不同区域时，对应Endpoint不同，可以从 终端节点 中获取。 例如IAM服务在“华北-北京四”区域的Endpoint为“iam.cn-north-4.myhuaweicloud.com”。

参数	说明
resource-path	资源路径，即API访问路径。从具体API的URI模块获取。 例如“获取用户Token”API的resource-path为“/v3/auth/tokens”。 例如调用人脸检测接口，API的resource-path为“/v2/{project_id}/face-detect”。
query-string	查询参数，可选，查询参数前面需要带一个“？”，形式为“参数名=参数取值”，例如“limit=10”，表示查询不超过10条数据。

例如，您需要在“华北-北京四”区域调用人脸检测API，URI如下所示。其中face.cn-north-4.myhuaweicloud.com表示“华北-北京四”区域的Endpoint，{project_id}表示与区域对应的项目id，可在“[我的凭证](#)”页面获取。

```
https://face.cn-north-4.myhuaweicloud.com/v2/{project_id}/face-detect
```

图 3-3 URI 示意图



说明

为查看方便，服务每个具体API的URI，只给出resource-path部分，并将请求方法写在一起。这是因为URI-scheme都是HTTPS，而Endpoint在同一个区域也相同，所以简洁起见将这两部分省略。

请求方法

HTTP请求方法（也称为操作或动词），它告诉服务你正在请求什么类型的操作。

表 3-2 HTTP 方法

方法	说明
GET	请求服务器返回指定资源。
PUT	请求服务器更新指定资源。
POST	请求服务器新增资源或执行特殊操作。
DELETE	请求服务器删除指定资源，如删除对象等。
HEAD	请求服务器资源头部。

方法	说明
PATCH	请求服务器更新资源的部分内容。 当资源不存在的时候，PATCH可能会去创建一个新的资源。

人脸检测API的URI部分，您可以看到其请求方法为“POST”，则其请求为：
POST https://face.cn-north-4.myhuaweicloud.com/v2/{project_id}/face-detect

请求消息头

附加请求头字段，如指定的URI和HTTP方法所要求的字段。例如定义消息体类型的请求头“Content-Type”，请求鉴权信息等。

- **Content-Type:** 消息体的类型（格式），必选，默认取值为“application/json”，有其他取值时会在具体接口中专门说明。
- **X-Auth-Token:** 用户Token，可选，当使用Token方式认证时，必须填充该字段。用户Token请参考[认证鉴权](#)。

添加消息头后的请求如下所示。

```
POST https://face.cn-north-4.myhuaweicloud.com/v2/{project_id}/face-detect
Content-Type: application/json
x-auth-token: MIlaBgYJKoZlhvcNAQcC...
```

请求消息体

请求消息体通常以结构化格式发出，与请求消息头中Content-type对应，传递除请求消息头之外的内容。若请求消息体中参数支持中文，则中文字符必须为UTF-8编码。

每个接口的请求消息体内容不同，也并不是每个接口都需要有请求消息体（或者说消息体为空），GET、DELETE操作类型的接口就不需要消息体，消息体具体内容需要根据具体接口而定。

例如，对于人脸检测接口，您可以从接口的请求部分看到所需的请求参数及参数说明。将消息体加入后的请求如下所示。

```
POST https://face.cn-north-4.myhuaweicloud.com/v2/{project_id}/face-detect
Request Header:
Content-Type: application/json
X-Auth-Token: MIINRwYJKoZlhvcNAQcCoIINODCCDTQCAQExDT...

Request Body:
{
  "image_base64": "/9j/4AAQSkZJRgABAgEASABIAAD...",
  "attributes": "2,12,13"
}
```

到这里为止这个请求需要的内容就具备齐全了，您可以使用curl、Postman或直接编写代码等方式发送请求调用API。

3.3 认证鉴权

调用接口有如下两种认证方式，您可以选择其中一种进行认证鉴权。

- Token认证：通过Token认证通用请求。
- AK/SK认证：通过AK (Access Key ID) /SK (Secret Access Key)加密调用请求。

Token 认证

说明

需要使用同一个Token鉴权时，可以缓存起来，避免频繁调用。

关于Token有效期的详细说明请参见[获取IAM用户Token（使用密码）](#)。

Token在计算机系统中代表令牌（临时）的意思，拥有Token就代表拥有某种权限。Token认证就是在调用API的时候将Token加到请求消息头，从而通过身份认证，获得操作API的权限。

username、*domainname*、*project name*可登录控制台“[我的凭证](#)”页面获取。*password*为用户密码。



获取Token时，如果出现账密报错“The username or password is wrong.”，请参见[如何处理账密报错](#)。

POST <https://iam.cn-north-4.myhuaweicloud.com/v3/auth/tokens>
Content-Type: application/json

Request Body:

```
{
  "auth": {
    "identity": {
      "methods": [
        "password"
      ],
      "password": {
        "user": {
          "name": "username",
          "password": "*****",
          "domain": {
            "name": "domainname"
          }
        }
      }
    },
    "scope": {
      "project": {
        "name": "project name"
      }
    }
  }
}
```

如下图所示，返回的响应消息头中“x-subject-token”就是需要获取的用户Token。获取Token之后，您就可以使用Token认证调用FRS服务API。

您还可以通过这个视频教程了解如何使用Token认证：<https://bbs.huaweicloud.com/videos/101333>。

图 3-4 获取用户 Token 响应消息头

```

connection → keep-alive
content-type → application/json
date → Tue, 12 Feb 2019 06:52:13 GMT
server → Web Server
strict-transport-security → max-age=31536000; includeSubdomains;
transfer-encoding → chunked
via → proxy A
x-content-type-options → nosniff
x-download-options → noopen
x-frame-options → SAMEORIGIN
x-iam-trace-id → 218d45ab-d674-4995-af3a-2d0255ba41b5

x-subject-token
→ IAMToken(218d45ab-d674-4995-af3a-2d0255ba41b5)
[{"sub": "218d45ab-d674-4995-af3a-2d0255ba41b5", "aud": "iam:sts:token", "exp": 3600, "iss": "iam:sts:token", "jti": "218d45ab-d674-4995-af3a-2d0255ba41b5"}]

x-xss-protection → 1; mode=block;
    
```

AK/SK 认证

说明

AK/SK 签名认证方式仅支持消息体大小 12MB 以内，12MB 以上的请求请使用 Token 认证。

AK/SK 认证就是使用 AK/SK 对请求进行签名，在请求时将签名信息添加到消息头，从而通过身份认证。

- AK (Access Key ID)：访问密钥 ID。与私有访问密钥关联的唯一标识符；访问密钥 ID 和私有访问密钥一起使用，对请求进行加密签名。
- SK (Secret Access Key)：与访问密钥 ID 结合使用的密钥，对请求进行加密签名，可标识发送方，并防止请求被修改。

使用 AK/SK 认证时，您可以基于签名算法使用 AK/SK 对请求进行签名，也可以使用专门的签名 SDK 对请求进行签名。详细的签名方法和 SDK 使用方法请参见 [API 签名指南](#)。

须知

签名 SDK 只提供签名功能，与服务提供的 SDK 不同，使用时请注意。
AK/SK 获取方式请参考 [获取 AK/SK](#)。

3.4 返回结果

请求发送以后，您会收到响应，包含：状态码、响应消息头和响应消息体。

状态码

状态码是一组从 1xx 到 5xx 的数字代码，状态码表示了请求响应的状态，完整的状态码列表请参见 [状态码](#)。

对于[获取用户Token](#)接口，如果调用后返回状态码为“201”，则表示请求成功。

响应消息头

对应请求消息头，响应同样也有消息头，如“Content-type”。

表 3-3 公共响应消息头

消息头名称	说明	是否必选
Content-Type	用于指明发送给接收者的实体正文的媒体类型。 类型：字符串。 默认值：application/json; charset=UTF-8	是
X-request-id	此字段携带请求ID号，以便任务跟踪。 类型：字符串。request_id-timestamp-hostname（request_id在服务器端生成UUID，timestamp为当前时间戳，hostname为处理当前接口的服务器名称）。 默认值：无。	否
X-ratelimit	此字段携带总计流控请求数。 类型：整型。 默认值：无。	否
X-ratelimit-used	此字段携带剩下请求数。 类型：整型。 默认值：无。	否
X-ratelimit-window	此字段携带流控单位。 类型：字符串。单位按照分钟、小时、天。 默认值：小时。	否

响应消息体

响应消息体通常以结构化格式返回，与响应消息头中Content-type对应，传递除响应消息头之外的内容。

对于人脸检测接口，返回如下消息体。

```
{
  "faces": [
    {
      "bounding_box": {
        "width": 174,
        "top_left_y": 37,
        "top_left_x": 22,
        "height": 174
      },
      "attributes": {
        "age": 35,
        "quality": {
          "total_score": 0.5869140625,
          "blur": 0.385498046875,

```

```
    "pose": 0.3349609375,  
    "occlusion": 0.392333984375,  
    "illumination": 0.3408203125  
  },  
  "expression": {  
    "type": "happy",  
    "probability": 0.74  
  }  
}  
]  
}
```

当接口调用出错时，会返回错误码及错误信息说明，错误响应的Body体格式如下所示。

```
{  
  "error_code": "FRS.0019",  
  "error_msg": "The service has not been subscribed."  
}
```

其中，error_code表示错误码，error_msg表示错误描述信息，具体请参见[错误码](#)。

4 API

4.1 人脸检测

功能介绍

人脸检测是对输入图片进行人脸检测和分析，输出人脸在图像中的位置、人脸关键属性。若照片中存在多张人脸，则返回所有符合条件的人脸特征信息。

前提条件：

请确保您已开通[人脸识别服务](#)，具体操作方法请参见[申请服务](#)。

约束限制：

- 只支持识别JPG、PNG、JPEG、BMP格式的图片。
- application/json请求的body中，请使用标准Json格式。
- Base64编码中请勿使用回车换行。
- 系统不保存用户图片。
- 图片大小**小于8MB**，由于过大图片会导致时延较长，并且图片信息量不大，建议**小于1MB**。
- 图片分辨率**小于4096*4096**，图片中人脸像素**大于80*80**，建议**120*120**以上。
- 为保证识别效果，人脸图片建议要求如下：
 - a. 光照大于200lux、无反光强光阴影现象。
 - b. 人脸无遮挡、整体清晰无拖尾抖动等运动模糊。
 - c. 侧脸不超过30°、俯仰角小于15°、偏转角小于15°、图片中人脸保持竖置正脸。
- 支持人脸图片旋转角检测，返回人脸图片顺时针旋转角度，值为0°、90°、180°和270°。
- 其他的约束限制信息请参见[约束与限制](#)章节。

建议：

- 由于过大图片对识别算法精度无明显提升，同时会导致时延较长，建议传入图片**小于1MB**，一般**500KB**左右足够。

- OBS上存储的图片也建议小于1MB。
- 图片中人脸像素建议120*120以上。

调试

您可以在[API Explorer](#)中调试该接口，支持自动认证鉴权。API Explorer可以自动生成SDK代码示例，并提供SDK代码示例调试功能。

URI

POST /v2/{project_id}/face-detect

表 4-1 路径参数

参数	是否必选	参数类型	描述
project_id	是	String	项目ID，获取方法请参见 获取项目ID/账号名/AK/SK 。

请求参数

表 4-2 请求 Header 参数

参数	是否必选	参数类型	描述
X-Auth-Token	是	String	用户Token。 用于获取操作API的权限。获取方法请参见 认证鉴权 。
Content-Type	是	String	发送的实体的MIME类型，参数值为“application/json”。

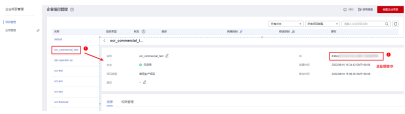

参数	是否必选	参数类型	描述
Enterprise-Project-Id	否	String	<p>企业项目ID。FRS支持通过企业项目管理（EPS）对不同用户组和用户的资源使用，进行分账，当前仅支持按需计费模式。</p> <p>获取方法：进入“企业项目管理”页面，单击企业名称，在企业项目详情页获取Enterprise-Project-Id（企业项目ID）。</p>  <p>说明 创建企业项目后，在传参时，有以下三类场景。</p> <ul style="list-style-type: none"> 携带正确的ID，正常使用FRS服务，账单的企业项目会被分类到企业ID对应的企业项目中。 携带格式正确但不存在的ID，正常使用FRS服务，账单的企业项目会显示对应不存在的企业项目ID。 不携带ID或格式错误ID（包含特殊字符等），正常使用FRS服务，账单的企业项目会被分类到"default"中。

表 4-3 请求 Body 参数

参数名	是否必选	参数类型	说明
image_url	与image_file、image_base64三选一	String	<p>图片的URL路径，目前仅支持华为云上OBS的URL，使用时只需保证FRS有权限读取该OBS桶的数据。开通读取权限的操作请参见服务授权。</p> <p>OBS URL格式如下，可在OBS控制台获取。 https://BucketName.obs.xxx.com/ObjectName</p> 

参数名	是否必选	参数类型	说明
image_file	与image_url、image_base64三选一	File	本地图片文件，要求： <ul style="list-style-type: none"> • 图片不能超过8MB，建议小于1MB。 • 上传文件时，请求格式为multipart。
image_base64	与image_file、image_url三选一	String	图像数据，Base64编码，要求： <ul style="list-style-type: none"> • Base64编码后大小不超过8MB，建议小于1MB。 • 图片为JPG/JPEG/BMP/PNG格式。
attributes	否	String	是否返回人脸属性，希望获取的属性列表，多个属性间使用逗号(,)隔开。目前支持的属性有： <ul style="list-style-type: none"> • 2: 年龄 • 4: 装束(帽子、眼镜) • 6: 口罩 • 7: 发型 • 8: 胡须 • 11: 图片类型 • 12: 质量 • 13: 表情 • 21: 人脸图片旋转角(顺时针偏转角度)，支持0°、90°、180°和270°图片旋转。

响应参数

状态码：200

表 4-4 响应 Body 参数

参数	参数类型	描述
faces	Array of DetectFace objects	检测到的人脸。调用失败时无此字段。

表 4-5 DetectFace

参数	参数类型	描述
bounding_box	BoundingBox object	人脸在图像中的位置。
attributes	Attributes object	人脸关键属性，比如头部姿态。

表 4-6 BoundingBox

参数	参数类型	描述
width	Integer	人脸图像所在矩形框的宽度。
top_left_y	Integer	矩形框左上角纵坐标。
top_left_x	Integer	矩形框左上角横坐标。
height	Integer	人脸图像所在矩形框的高度。

表 4-7 Attributes

参数	参数类型	描述
dress	Dress object	包含glass和hat两个属性结果。
glass	String	是否戴眼镜： <ul style="list-style-type: none"> ● yes: 戴眼镜 ● dark: 戴墨镜 ● none: 未戴眼镜 ● unknown: 未知
hat	String	是否戴帽子： <ul style="list-style-type: none"> ● yes: 戴帽子 ● none: 未戴帽子 ● unknown: 未知
age	Integer	年龄。
mask	String	是否戴口罩： <ul style="list-style-type: none"> ● yes: 戴口罩 ● none: 未戴口罩 ● unknown: 未知

参数	参数类型	描述
beard	String	胡须： <ul style="list-style-type: none"> • yes: 有胡须 • none: 无胡须 • unknown: 未知
phototype	String	图片类型： <ul style="list-style-type: none"> • idcard: 证件照 • monitor: 摄像头监控 • internet photo: 网络图片
quality	FaceQuality object	图片中人脸的遮挡度、模糊度、光照强度、姿态角度。
hair	String	发型： <ul style="list-style-type: none"> • long: 长发 • short: 短发 • unknown: 未知
expression	expression object	人脸表情，包括中性、高兴、害怕、惊讶、伤心、生气、厌恶。
face_angle	Integer	人脸图片旋转角（顺时针偏转角度），支持0°、90°、180°和270°图片旋转。

表 4-8 Dress

参数	参数类型	描述
glass	String	是否戴眼镜： <ul style="list-style-type: none"> • yes: 戴眼镜 • dark: 戴墨镜 • none: 未戴眼镜 • unknown: 未知
hat	String	是否戴帽子： <ul style="list-style-type: none"> • yes: 戴帽子 • none: 未戴帽子 • unknown: 未知

表 4-9 FaceQuality

参数	参数类型	描述	推荐值
total_score	Double	人脸质量总分，取值范围[0-1]，分值越大质量越高。	大于0.45
blur	Double	模糊度，取值范围[0-1]，分值越大模糊问题越严重。	小于0.3
pose	Double	姿态，取值范围[0-1]，分值越大姿态问题越严重。	小于0.3
occlusion	Double	遮挡，取值范围[0-1]，分值越大遮挡问题越严重。	小于0.3
illumination	Double	光照，取值范围[0-1]，分值越大光照问题越严重。	小于0.3

表 4-10 expression

参数	参数类型	描述
type	String	人脸表情类型： <ul style="list-style-type: none"> • neutral: 中性 • happy: 高兴 • fear: 害怕 • surprise: 惊讶 • sad: 伤心 • angry: 生气 • disgust: 厌恶 • unknown: 图片质量问题导致未识别
probability	Double	表情置信度，取值范围[0-1]。

状态码： 400

表 4-11 响应 Body 参数

参数	参数类型	描述
error_code	String	调用失败时的错误码，具体请参考 错误码 。调用成功时无此字段。
error_msg	String	调用失败时的错误信息。调用成功时无此字段。

请求示例

X-Auth-Token值获取方法请参见[快速入门](#)。

- 检测图片中是否有人脸，待检测图片通过图片的base64编码传入

```
POST https://{endpoint}/v2/{project_id}/face-detect
Request Header:
Content-Type: application/json
X-Auth-Token: MIINRwYJKoZIhvcNAQcCoIINODCCDTQCAQExDT...

Request Body:
{
  "image_base64": "/9j/4AAQSkZJRgABAgEASABIAAD...",
  "attributes": "2,12,13"
}
```

- 检测图片中是否有人脸，待检测图片通过图片文件传入

```
POST https://{endpoint}/v2/{project_id}/face-detect
Request Header:
X-Auth-Token: MIINRwYJKoZIhvcNAQcCoIINODCCDTQCAQExDT...

Request Body:
image_file: File(图片文件)
attributes: "2,12,13"
```

- 检测图片中是否有人脸，待检测图片通过图片的url传入

```
POST https://{endpoint}/v2/{project_id}/face-detect
Request Header:
Content-Type: application/json
X-Auth-Token: MIINRwYJKoZIhvcNAQcCoIINODCCDTQCAQExDT...

Request Body:
{
  "image_url": "https://<bucket-name>.<endpoint>/<object-name>",
  "attributes": "2,12,13"
}
```

- 使用Python3语言读取本地图片，并判断图片中是否有人脸

```
# -*- coding:utf-8 -*-

import requests
import base64

endpoint = '开通服务所在region的人脸识别服务域名'
project_id = '开通服务所在region的用户项目ID'
token = '用户获取得到的实际token值'
headers = {'Content-Type': 'application/json', 'X-Auth-Token': token}

url = "https://{endpoint}/v2/{project_id}/face-detect".format(endpoint=endpoint, project_id=project_id)
imagepath = r'./data/face-demo.png'
with open(imagepath, "rb") as bin_data:
    image_data = bin_data.read()
    image_base64 = base64.b64encode(image_data).decode("utf-8")
    body = {"image_base64": image_base64, "attributes": "2,12,13"}
    response = requests.post(url, headers=headers, json=body, verify=False)
    print(response.text)
```

- 使用Java语言读取图片的base64编码，并判断图片中是否有人脸

```
import com.huawei.trace.http.apache.httpclient.TraceApacheHttpClientBuilder;
import org.apache.http.HttpHeaders;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.config.Registry;
import org.apache.http.config.RegistryBuilder;
import org.apache.http.conn.socket.ConnectionSocketFactory;
import org.apache.http.conn.socket.PlainConnectionSocketFactory;
import org.apache.http.conn.ssl.NoopHostnameVerifier;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
```

```
import org.apache.http.entity.ContentType;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;
import org.apache.http.ssl.SSLContextBuilder;
import org.apache.http.ssl.TrustStrategy;
import org.apache.http.util.EntityUtils;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.security.KeyManagementException;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.concurrent.TimeUnit;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.SSLContext;

/**
 * 此demo仅供测试使用，强烈建议使用SDK
 * 使用前需已配置HttpClient jar包。jar包可通过下载SDK获取
 */
public class FaceDetect {
    protected static HttpClientBuilder buildClient(HttpClientBuilder httpClientBuilder) {
        SSLContext sslContext = null;
        try {
            sslContext = new SSLContextBuilder().loadTrustMaterial(null, new TrustStrategy() {
                public boolean isTrusted(X509Certificate[] arg0, String arg1) throws CertificateException {
                    return true;
                }
            }).build();
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e.getMessage());
        } catch (KeyManagementException e) {
            throw new RuntimeException(e.getMessage());
        } catch (KeyStoreException e) {
            throw new RuntimeException(e.getMessage());
        }
        httpClientBuilder.setSSLContext(sslContext);
        httpClientBuilder.setConnectionTimeToLive(30, TimeUnit.SECONDS);
        HostnameVerifier hostnameVerifier = NoopHostnameVerifier.INSTANCE;
        SSLConnectionSocketFactory sslSocketFactory = new SSLConnectionSocketFactory(sslContext,
        hostnameVerifier);
        Registry<ConnectionSocketFactory> socketFactoryRegistry
            = RegistryBuilder.<ConnectionSocketFactory>create().register("http",
            PlainConnectionSocketFactory.getSocketFactory()).register("https", sslSocketFactory).build();
        PoolingHttpClientConnectionManager connMgr = new
        PoolingHttpClientConnectionManager(socketFactoryRegistry);
        connMgr.setMaxTotal(200);
        connMgr.setDefaultMaxPerRoute(100);
        httpClientBuilder.setConnectionManager(connMgr);
        return httpClientBuilder;
    }

    public static String doPost(String url, String jsonStr, String token, CloseableHttpClient client) {
        HttpPost post = new HttpPost(url);
        StringEntity entity = new StringEntity(jsonStr, ContentType.APPLICATION_JSON);
        post.setEntity(entity);
        post.setHeader(HttpHeaders.CONTENT_TYPE, "application/json");
        //time unit is milliseconds
        RequestConfig requestConfig =
        RequestConfig.custom().setConnectTimeout(3000).setSocketTimeout(3000).build();
        post.setConfig(requestConfig);
        post.setHeader("X-Auth-Token", token);
        HttpResponse response = null;
        String result = "";
        try {
```

```
response = client.execute(post);
HttpEntity responseBody = response.getEntity();
if (responseBody == null) {
    System.out.println("the response body is null.");
    return result;
} else {
    byte[] byteStream = EntityUtils.toByteArray(responseBody);
    System.arraycopy(byteStream, 0, new byte[byteStream.length], 0, byteStream.length);
    result = new String(byteStream, StandardCharsets.UTF_8);
}
} catch (IOException e) {
    e.printStackTrace();
}
return result;
}

public static void main(String[] args) {
    CloseableHttpClient client = buildClient(TraceApacheHttpClientBuilder.create()).build();
    // endpoint和project_id需要替换成实际信息。
    String url = "https://{{endpoint}}/v2/{{project_id}}/face-detect";
    // image_base64需要替换成实际的值
    String jsonStr = "{ \"image_base64\": \"9j/4AAQSkZJRgABAgEASABIAAD...\"}";
    String token = "对应region的token";
    String result = doPost(url, jsonStr, token, client);
    System.out.println(result);
}
}
```

响应示例

状态码：200

成功响应样例

```
{
  "faces": [
    {
      "bounding_box": {
        "width": 174,
        "top_left_y": 37,
        "top_left_x": 22,
        "height": 174
      },
      "attributes": {
        "age": 35,
        "quality": {
          "total_score": 0.5869140625,
          "blur": 0.385498046875,
          "pose": 0.3349609375,
          "occlusion": 0.392333984375,
          "illumination": 0.3408203125
        },
        "expression": {
          "type": "happy",
          "probability": 0.74
        }
      }
    }
  ]
}
```

状态码：400

失败响应样例

```
{
  "error_code": "FRS.0019",
  "error_msg": "The service has not been subscribed."
}
```

状态码

状态码请参见[状态码](#)。

错误码

错误码请参见[错误码](#)

4.2 人脸比对

功能介绍

人脸比对是将两个人脸进行比对，来判断是否为同一个人，返回比对置信度。如果传入的图片中包含多个人脸，选取最大的人脸进行比对。

前提条件：

请确保您已开通[人脸识别服务](#)，具体操作方法请参见[申请服务](#)。

约束限制：

- 人脸比对输入的两张图片总大小。
- 只支持识别JPG、PNG、JPEG、BMP格式的图片。
- application/json请求的body中，请使用标准Json格式。
- Base64编码中请勿使用回车换行。
- 系统不保存用户图片。
- 图片大小小于**8MB**，由于过大图片会导致时延较长，并且图片信息量不大，建议**小于1MB**。
- 图片分辨率**小于4096*4096**，图片中人脸像素**大于80*80**，建议120*120以上。
- 为保证识别效果，人脸图片建议要求如下：
 - a. 光照大于200lux、无反光强光阴影现象。
 - b. 人脸无遮挡、整体清晰无拖尾抖动等运动模糊。
 - c. 侧脸不超过30°、俯仰角小于15°、偏转角小于15°、图片中人脸保持竖置正脸。
- 具体的约束限制信息请参见[约束与限制](#)章节。

建议：

- 由于过大图片对识别算法精度无明显提升，同时会导致时延较长，建议传入图片**小于1MB**，一般**500KB**左右足够。
- OBS上存储的图片也建议**小于1MB**。
- 图片中人脸像素建议**120*120**以上。

调试

您可以在[API Explorer](#)中调试该接口，支持自动认证鉴权。API Explorer可以自动生成SDK代码示例，并提供SDK代码示例调试功能。

URI

POST /v2/{project_id}/face-compare

表 4-12 路径参数

参数	是否必选	参数类型	描述
project_id	是	String	项目ID，获取方法请参见 获取项目ID/账号名/AK/SK 。

请求参数

表 4-13 请求 Header 参数


参数	是否必选	参数类型	描述
X-Auth-Token	是	String	用户Token。 用于获取操作API的权限。获取方法请参见 认证鉴权 。
Content-Type	是	String	发送的实体的MIME类型，参数值为“application/json”。
Enterprise-Project-Id	否	String	<p>企业项目ID。FRS支持通过企业项目管理（EPS）对不同用户组和用户的资源使用，进行分账，当前仅支持按需计费模式。</p> <p>获取方法：进入“企业项目管理”页面，单击企业名称，在企业项目详情页获取Enterprise-Project-Id（企业项目ID）。</p>  <p>说明 创建企业项目后，在传参时，有以下三类场景。</p> <ul style="list-style-type: none"> 携带正确的ID，正常使用FRS服务，账单的企业项目会被分类到企业ID对应的企业项目中。 携带格式正确但不存在的ID，正常使用FRS服务，账单的企业项目会显示对应不存在的企业项目ID。 不携带ID或格式错误ID（包含特殊字符等），正常使用FRS服务，账单的企业项目会被分类到"default"中。

表 4-14 请求 Body 参数

参数名	参数类型	是否必选	说明
image1_url	String	与image1_file、image1_base64三选一	<p>图片的URL路径，目前仅支持华为云上OBS的URL，使用时只需保证FRS有权限读取该OBS桶的数据。开通读取权限的操作请参见服务授权。</p> <p>OBS URL格式如下，可在OBS控制台获取。 https://BucketName.obs.xxx.com/ObjectName</p> 
image1_file	File	与image1_url、image1_base64三选一	<p>本地图片文件，要求：</p> <ul style="list-style-type: none"> ● 图片不能超过8MB，建议小于1MB。 ● 上传文件时，请求格式为 multipart。
image1_base64	String	与image1_file、image1_url三选一	<p>图像数据，Base64编码，要求：</p> <ul style="list-style-type: none"> ● Base64编码后大小不超过8MB，建议小于1MB。 ● 图片为JPG/JPEG/BMP/PNG格式。
image2_url	String	与image2_file、image2_base64三选一	<p>图片的URL路径，目前仅支持华为云上OBS的URL，且人脸识别服务有权限读取该OBS桶的数据。开通读取权限的操作请参见服务授权。</p>
image2_file	File	与image2_url、image2_base64三选一	<p>本地图片文件，要求：</p> <ul style="list-style-type: none"> ● 图片不能超过8MB，建议小于1MB。 ● 上传文件时，请求格式为 multipart。
image2_base64	String	与image2_file、image2_url三选一	<p>图像数据，Base64编码，要求：</p> <ul style="list-style-type: none"> ● Base64编码后大小不超过8MB，建议小于1MB。 ● 图片为JPG/JPEG/BMP/PNG格式。

响应参数

状态码：200

表 4-15 响应 Body 参数

参数	参数类型	描述
image1_face	CompareFace object	第1幅图像中检测到的人脸，DetectFace结构见 DetectFace 。调用失败时无此字段。
image2_face	CompareFace object	第2幅图像中检测到的人脸，DetectFace结构见 DetectFace 。调用失败时无此字段。
similarity	Double	人脸相似度，1表示最大，0表示最小，值越大表示越相似。一般情况下超过0.93即可认为是同一个人。如果图片质量较低，也会影响相似度。调用失败时无此字段。

表 4-16 CompareFace

参数	参数类型	描述
bounding_box	BoundingBox object	人脸在图像中的位置。

表 4-17 BoundingBox

参数	参数类型	描述
width	Integer	人脸图像所在矩形框的宽度。
top_left_y	Integer	矩形框左上角纵坐标。
top_left_x	Integer	矩形框左上角横坐标。
height	Integer	人脸图像所在矩形框的高度。

状态码：400

表 4-18 响应 Body 参数

参数	参数类型	描述
error_code	String	调用失败时的错误码，具体请参考 错误码 。调用成功时无此字段。
error_msg	String	调用失败时的错误信息。调用成功时无此字段。

请求示例

X-Auth-Token值获取方法请参见[快速入门](#)。

- 判断两张人脸图片的相似度，待检测图片通过图片的base64编码传入

```
POST https://{endpoint}/v2/{project_id}/face-compare
Request Header:
Content-Type: application/json
X-Auth-Token: MIINRwYJKoZIhvcNAQcCoIINODCCDTQCAQExDT...

Request Body:
{
  "image1_base64": "/9j/4AAQSkZJRgABAQEASABIAAD...",
  "image2_base64": "/9j/4AAQSkZJRgABAQEASABIAAD..."
}
```

- 判断两张人脸图片的相似度，待检测图片通过图片文件传入

```
POST https://{endpoint}/v2/{project_id}/face-compare
Request Header:
X-Auth-Token: MIINRwYJKoZIhvcNAQcCoIINODCCDTQCAQExDT...

Request Body:
image1_file: File(图片文件)
image2_file: File(图片文件)
```

- 判断两张人脸图片的相似度，待检测图片通过图片的url传入

```
POST https://{endpoint}/v2/{project_id}/face-compare
Request Header:
Content-Type: application/json
X-Auth-Token: MIINRwYJKoZIhvcNAQcCoIINODCCDTQCAQExDT...

Request Body:
{
  "image1_url": "https://<bucket-name>.<endpoint>/<object-name>",
  "image2_url": "https://<bucket-name>.<endpoint>/<object-name>"
}
```

- 使用Python3语言检测两张人脸图片的相似度

```
# -*- coding:utf-8 -*-

import requests
import base64

endpoint = '开通服务所在region的人脸识别服务域名'
project_id = '开通服务所在region的用户项目ID'
token = '用户获取得到的实际token值'
headers = {'Content-Type': 'application/json', 'X-Auth-Token': token}

url = "https://{endpoint}/v2/{project_id}/face-compare".format(endpoint=endpoint,
project_id=project_id)

image1_file_path = r'./data/face-demo1.png'
image2_file_path = r'./data/face-demo2.png'
with open(image1_file_path, "rb") as bin_data:
    image1_data = bin_data.read()
with open(image2_file_path, "rb") as bin_data:
    image2_data = bin_data.read()
image1_base64 = base64.b64encode(image1_data).decode("utf-8")
image2_base64 = base64.b64encode(image2_data).decode("utf-8")
body = {"image1_base64": image1_base64,"image2_base64": image2_base64}
response = requests.post(url, headers=headers, json=body, verify=False)
print(response.text)
```

- 使用Java语言检测两张人脸图片的相似度

```
import com.huawei.trace.http.apache.httpclient.TraceApacheHttpClientBuilder;
import org.apache.http.HttpEntity;
import org.apache.http.HttpHeaders;
import org.apache.http.HttpResponse;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.client.methods.HttpPost;
```

```
import org.apache.http.config.Registry;
import org.apache.http.config.RegistryBuilder;
import org.apache.http.conn.socket.ConnectionSocketFactory;
import org.apache.http.conn.socket.PlainConnectionSocketFactory;
import org.apache.http.conn.ssl.NoopHostnameVerifier;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import org.apache.http.entity.ContentType;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;
import org.apache.http.ssl.SSLContextBuilder;
import org.apache.http.ssl.TrustStrategy;
import org.apache.http.util.EntityUtils;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.security.KeyManagementException;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.concurrent.TimeUnit;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.SSLContext;

/**
 * 此demo仅供测试使用，强烈建议使用SDK
 * 使用前需已配置HttpClient jar包。jar包可通过下载SDK获取
 */

public class FaceCompare {
    protected static HttpClientBuilder buildClient(HttpClientBuilder httpClientBuilder) {
        SSLContext sslContext = null;
        try {
            sslContext = new SSLContextBuilder().loadTrustMaterial(null, new TrustStrategy() {
                public boolean isTrusted(X509Certificate[] arg0, String arg1) throws CertificateException {
                    return true;
                }
            }).build();
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e.getMessage());
        } catch (KeyManagementException e) {
            throw new RuntimeException(e.getMessage());
        } catch (KeyStoreException e) {
            throw new RuntimeException(e.getMessage());
        }
        httpClientBuilder.setSSLContext(sslContext);
        httpClientBuilder.setConnectionTimeToLive(30, TimeUnit.SECONDS);
        HostnameVerifier hostnameVerifier = NoopHostnameVerifier.INSTANCE;
        SSLConnectionSocketFactory sslSocketFactory = new SSLConnectionSocketFactory(sslContext,
            hostnameVerifier);
        Registry<ConnectionSocketFactory> socketFactoryRegistry
            = RegistryBuilder.<ConnectionSocketFactory>create().register("http",
                PlainConnectionSocketFactory.getSocketFactory()).register("https", sslSocketFactory).build();
        PoolingHttpClientConnectionManager connMgr = new
        PoolingHttpClientConnectionManager(socketFactoryRegistry);
        connMgr.setMaxTotal(200);
        connMgr.setDefaultMaxPerRoute(100);
        httpClientBuilder.setConnectionManager(connMgr);
        return httpClientBuilder;
    }

    public static String doPost(String url, String jsonStr, String token, CloseableHttpClient client) {
        HttpPost post = new HttpPost(url);
        StringEntity entity = new StringEntity(jsonStr, ContentType.APPLICATION_JSON);
        post.setEntity(entity);
        post.setHeader(HttpHeaders.CONTENT_TYPE, "application/json");
        //time unit is milliseconds
        RequestConfig requestConfig =
```

```
RequestConfig.custom().setConnectTimeout(3000).setSocketTimeout(3000).build();
post.setConfig(requestConfig);
post.setHeader("X-Auth-Token", token);
HttpResponse response = null;
String result = "";
try {
    response = client.execute(post);
    HttpEntity responseBody = response.getEntity();
    if (responseBody == null) {
        System.out.println("the response body is null.");
        return result;
    } else {
        byte[] byteStream = EntityUtils.toByteArray(responseBody);
        System.arraycopy(byteStream, 0, new byte[byteStream.length], 0, byteStream.length);
        result = new String(byteStream, StandardCharsets.UTF_8);
    }
} catch (IOException e) {
    e.printStackTrace();
}
return result;
}

public static void main(String[] args) {
    CloseableHttpClient client = buildClient(TraceApacheHttpClientBuilder.create()).build();
    // endpoint和project_id需要替换成实际信息。
    String url = "https://{{endpoint}}/v2/{{project_id}}/face-compare";
    // image_base64需要替换成实际的值
    String jsonStr = "{ \"image1_base64\": \"\n/9j/\n4AAQSkZJRgABAgEASABIAAD...\", \"image2_base64\": \"\n/9j/4AAQSkZJRgABAgEASABIAAD...\"}";
    String token = "对应region的token";
    String result = doPost(url, jsonStr, token, client);
    System.out.println(result);
}
}
```

响应示例

状态码：200

成功响应样例

```
{
  "image1_face": {
    "bounding_box": {
      "width": 174,
      "top_left_y": 37,
      "top_left_x": 22,
      "height": 174
    }
  },
  "similarity": 0.4078676104545593,
  "image2_face": {
    "bounding_box": {
      "width": 118,
      "top_left_y": 28,
      "top_left_x": 94,
      "height": 118
    }
  }
}
```

状态码：400

失败响应样例

```
{
  "error_code": "FRS.0501",
  "error_msg": "Detect no face, check out your picture."
}
```

状态码

状态码请参见[状态码](#)。

错误码

错误码请参见[错误码](#)。

4.3 活体检测

4.3.1 动作活体检测

功能介绍

动作活体检测是通过判断视频中的人物动作与传入动作列表是否一致来识别视频中人物是否为活体，进行真人检测。如果有多张人脸出现，则选取最大的人脸进行判定。新老用户均可用该接口。

前提条件：

请确保您已开通[人脸识别服务](#)，具体操作方法请参见[申请服务](#)。

约束限制：

- 目前支持检测视频文件，或视频的Base64编码，不支持直接检测视频流，需要用户客户端自己获取视频流并保存成文件，然后调用活体检测接口。
- 视频文件大小**不超过8MB**，建议客户端压缩到**200KB~2MB**。
- application/json请求的body中，请使用标准Json格式。
- Base64编码中请勿使用回车换行。
- 系统不保存用户视频。
- 具体的约束限制信息请参见[约束与限制](#)章节。

建议：

- 建议帧率**10fps ~ 30fps**。
- 视频文件的大小建议客户端压缩到**200KB~2MB**。

调试

您可以在[API Explorer](#)中调试该接口，支持自动认证鉴权。API Explorer可以自动生成SDK代码示例，并提供SDK代码示例调试功能。

URI

POST /v1/{project_id}/live-detect

表 4-19 路径参数

参数	是否必选	参数类型	描述
project_id	是	String	项目ID，获取方法请参见 获取项目ID/账号名/AK/SK 。

请求参数

表 4-20 请求 Header 参数


参数	是否必选	参数类型	描述
X-Auth-Token	是	String	用户Token。 用于获取操作API的权限。获取方法请参见 认证鉴权 。
Content-Type	是	String	发送的实体的MIME类型，参数值为“application/json”。
Enterprise-Project-Id	否	String	<p>企业项目ID。FRS支持通过企业项目管理（EPS）对不同用户组和用户的资源使用，进行分账，当前仅支持按需计费模式。</p> <p>获取方法：进入“企业项目管理”页面，单击企业名称，在企业项目详情页获取Enterprise-Project-Id（企业项目ID）。</p>  <p>说明 创建企业项目后，在传参时，有以下三类场景。</p> <ul style="list-style-type: none"> 携带正确的ID，正常使用FRS服务，账单归到企业ID对应的企业项目中。 携带错误的ID，正常使用FRS服务，账单的企业项目会被分类为“未归集”。 不携带ID，正常使用FRS服务，账单的企业项目会被分类为“default”。

表 4-21 请求 Body 参数

参数名	是否必选	参数类型	说明
video_url	与video_file、video_base64三选一	String	<p>视频的URL路径，目前仅支持华为云上OBS的URL，使用时只需保证FRS有权限读取该OBS桶的数据。开通读取权限的操作请参见服务授权。</p> <p>视频要求：</p> <ul style="list-style-type: none"> • 视频Base64编码后大小不超过8MB，建议客户端压缩到200KB~2MB。 • 限制视频时长1~15秒。 • 建议帧率10fps~30fps。 • 封装格式：mp4、avi、flv、webm、asf、mov。 • 视频编码格式：h261、h263、h264、hevc、vc1、vp8、vp9、wmv3。
video_file	与video_url、video_base64三选一	File	<p>本地视频文件。上传文件时，请求格式为multipart。视频要求：</p> <ul style="list-style-type: none"> • 视频文件大小不超过8MB，建议客户端压缩到200KB~2MB。 • 限制视频时长1~15秒。 • 建议帧率10fps~30fps。 • 封装格式：mp4、avi、flv、webm、asf、mov。 • 视频编码格式：h261、h263、h264、hevc、vc1、vp8、vp9、wmv3。

参数名	是否必选	参数类型	说明
video_base64	与video_file、video_url三选一	String	<p>视频数据，Base64编码，要求：</p> <ul style="list-style-type: none"> • Base64编码后大小不超过8MB，建议客户端压缩到200KB~2MB。 • 限制视频时长1~15秒。 • 建议帧率10fps~30fps。 • 封装格式：mp4、avi、flv、webm、asf、mov。 • 视频编码格式：h261、h263、h264、hevc、vc1、vp8、vp9、wmv3。
actions	是	String	<p>动作代码顺序列表，英文逗号(,)分隔。建议单动作，目前支持的动作有：</p> <ul style="list-style-type: none"> • 1：左摇头 • 2：右摇头 • 3：点头 • 4：嘴部动作 • 5：眨眼 <p>说明 仅当actions的传参顺序和视频中的动作顺序一致时返回true。例如，视频中人物动作顺序为点头、嘴部动作，传参顺序需为3,4。 左右摇头动作建议角度15-30度，嘴部动作建议张嘴距离大于3厘米。</p>
action_time	否	String	<p>该参数为动作时间数组拼接的字符串，数组的长度和actions的数量一致，每一项代表了对应次序动作的起始时间和结束时间，单位为距视频开始的毫秒数。</p>
nod_threshold	否	double	<p>该参数为点头动作幅度的判断门限，取值范围:[1,90]，默认为10，单位为度。该值设置越大，则越难判断为点头。</p>

响应参数

状态码：200

表 4-22 响应 Body 参数

参数	参数类型	描述
video-result	video-result object	活体检测结果，video-result数据结构请见 VideoDetectResult 。调用失败时无此字段。
warning-list	Array of WarningList objects	警告信息列表，warning-list数据结构请见 WarningList 。调用失败时无此字段。

表 4-23 video-result

参数	参数类型	描述
alive	Boolean	是否是活体。
actions	Array of ActionsList objects	动作列表。
picture	String	返回检测出最大人脸图片的base64编码。 其中，图片格式为jpg，分辨率与视频分辨率相等。

表 4-24 ActionsList

参数	参数类型	描述
confidence	Double	置信度，取值范围0~1。
action	Integer	动作编号，取值范围：[1,2,3,4]. 1：左摇头 2：右摇头 3：点头 4：嘴部动作 5：眨眼

表 4-25 WarningList

参数	参数类型	描述
warningCode	Integer	警告ID。

参数	参数类型	描述
warningMsg	String	告警消息。

状态码： 400

表 4-26 响应 Body 参数

参数	参数类型	描述
error_code	String	调用失败时的错误码，具体请参考 错误码 。调用成功时无此字段。
error_msg	String	调用失败时的错误信息。调用成功时无此字段。

请求示例

X-Auth-Token值获取方法请参见[快速入门](#)。

- 通过传入视频的BASE64编码，判断视频中的人物动作与传入动作列表是否一致来识别视频中人物是否为活体

```
POST https://{endpoint}/v1/{project_id}/live-detect
Request Header:
Content-Type: application/json
X-Auth-Token: MIINRwYJKoZlhvcNAQcCoIINODCCDTQCAQExDT...

Request Body:
{
  "video_base64": "/9j/4AAQSkZJRgABAQEASABIAAD...",
  "actions": "1,3,2",
  "action_time": "1000-3000,4000-7000,9000-12000",
  "nod_threshold": 10
}
```

- 通过传入视频文件，判断视频中的人物动作与传入动作列表是否一致来识别视频中人物是否为活体

```
POST https://{endpoint}/v1/{project_id}/live-detect
Request Header:
X-Auth-Token: MIINRwYJKoZlhvcNAQcCoIINODCCDTQCAQExDT...

Request Body:
video_file: File(视频文件)
actions: 1,3,2
action_time: 1000-3000,4000-7000,9000-12000
nod_threshold: 10
```

- 通过传入视频URL，判断视频中的人物动作与传入动作列表是否一致来识别视频中人物是否为活体

```
POST https://{endpoint}/v1/{project_id}/live-detect
Request Header:
Content-Type: application/json
X-Auth-Token: MIINRwYJKoZlhvcNAQcCoIINODCCDTQCAQExDT...

Request Body:
{
  "video_url": "https://<bucket-name>.<endpoint>/<object-name>",
  "actions": "1,3,2",
  "action_time": "1000-3000,4000-7000,9000-12000",
}
```

```
"nod_threshold": 10
}
```

- 使用Python3语言读取视频文件，判断视频中的人物动作与传入动作列表是否一致来识别视频中人物是否为活体

```
# -*- coding:utf-8 -*-
```

```
import requests
import base64
```

```
endpoint = '开通服务所在region的人脸识别服务域名'
project_id = '开通服务所在region的用户项目ID'
token = "用户获取到的实际token值"
headers = {'Content-Type': 'application/json', 'X-Auth-Token': token}
```

```
url = "https://{endpoint}/v1/{project_id}/live-detect".format(endpoint=endpoint, project_id=project_id)
video_file_path = r'./data/face-video-demo.mp4'
with open(video_file_path, "rb") as bin_data:
    video_data = bin_data.read()
video_base64 = base64.b64encode(video_data).("utf-8")
body = {"video_base64": video_base64, "actions": "1,2,3", "action_time":
"1000-3000,4000-7000,9000-12000"}
response = requests.post(url, headers=headers, json=body, verify=False)
print(response.text)
```

- 使用Java语言读取视频base64编码，判断视频中的人物动作与传入动作列表是否一致来识别视频中人物是否为活体

```
import com.huawei.trace.http.apache.httpclient.TraceApacheHttpClientBuilder;
import org.apache.http.HttpEntity;
import org.apache.http.HttpHeaders;
import org.apache.http.HttpResponse;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.config.Registry;
import org.apache.http.config.RegistryBuilder;
import org.apache.http.conn.socket.ConnectionSocketFactory;
import org.apache.http.conn.socket.PlainConnectionSocketFactory;
import org.apache.http.conn.ssl.NoopHostnameVerifier;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import org.apache.http.entity.ContentType;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;
import org.apache.http.ssl.SSLContextBuilder;
import org.apache.http.ssl.TrustStrategy;
import org.apache.http.util.EntityUtils;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.security.KeyManagementException;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.concurrent.TimeUnit;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.SSLContext;
```

```
/**
```

```
 * 此demo仅供测试使用，强烈建议使用SDK
```

```
 * 使用前需已配置HttpClient jar包。jar包可通过下载SDK获取
```

```
 */
```

```
public class LiveDetect {
    protected static HttpClientBuilder buildClient(HttpClientBuilder httpClientBuilder) {
        SSLContext sslContext = null;
        try {
            sslContext = new SSLContextBuilder().loadTrustMaterial(null, new TrustStrategy() {
                public boolean isTrusted(X509Certificate[] arg0, String arg1) throws CertificateException {
                    return true;
                }
            });
        } catch (Exception e) {
            // ignore
        }
    }
}
```

```
    }
    }).build();
} catch (NoSuchAlgorithmException e) {
    throw new RuntimeException(e.getMessage());
} catch (KeyManagementException e) {
    throw new RuntimeException(e.getMessage());
} catch (KeyStoreException e) {
    throw new RuntimeException(e.getMessage());
}
httpClientBuilder.setSSLContext(sslContext);
httpClientBuilder.setConnectionTimeToLive(30, TimeUnit.SECONDS);
HostnameVerifier hostnameVerifier = NoopHostnameVerifier.INSTANCE;
SSLConnectionSocketFactory sslSocketFactory = new SSLConnectionSocketFactory(sslContext,
hostnameVerifier);
Registry<ConnectionSocketFactory> socketFactoryRegistry
    = RegistryBuilder.<ConnectionSocketFactory>create().register("http",
    PlainConnectionSocketFactory.getSocketFactory()).register("https", sslSocketFactory).build();
PoolingHttpClientConnectionManager connMgr = new
PoolingHttpClientConnectionManager(socketFactoryRegistry);
connMgr.setMaxTotal(200);
connMgr.setDefaultMaxPerRoute(100);
httpClientBuilder.setConnectionManager(connMgr);
return httpClientBuilder;
}

public static String doPost(String url, String jsonStr, String token, CloseableHttpClient client) {
    HttpPost post = new HttpPost(url);
    StringEntity entity = new StringEntity(jsonStr, ContentType.APPLICATION_JSON);
    post.setEntity(entity);
    post.setHeader(HttpHeaders.CONTENT_TYPE, "application/json");
    //time unit is milliseconds
    RequestConfig requestConfig =
RequestConfig.custom().setConnectTimeout(3000).setSocketTimeout(3000).build();
    post.setConfig(requestConfig);
    post.setHeader("X-Auth-Token", token);
    HttpResponse response = null;
    String result = "";
    try {
        response = client.execute(post);
        HttpEntity responseBody = response.getEntity();
        if (responseBody == null) {
            System.out.println("the response body is null.");
            return result;
        } else {
            byte[] byteStream = EntityUtils.toByteArray(responseBody);
            System.arraycopy(byteStream, 0, new byte[byteStream.length], 0, byteStream.length);
            result = new String(byteStream, StandardCharsets.UTF_8);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return result;
}

public static void main(String[] args) {
    CloseableHttpClient client = buildClient(TraceApacheHttpClientBuilder.create()).build();
    // endpoint和project_id需要替换成实际信息。
    String url = "https://{{endpoint}}/v1/{{project_id}}/live-detect";
    String jsonStr = "{ \"video_base64\": \"9j/4AAQSkZJRgABAgEASABIAAD...\", \"actions\": \"1,2,3\", \"action_time\": \"1000-3000,4000-7000,9000-12000\"}";
    String token = "对应region的token";
    String result = doPost(url, jsonStr, token, client);
    System.out.println(result);
}
}
```

响应样例

状态码：200

成功响应样例

```
{
  "video-result": {
    "alive": true,
    "actions": [
      {
        "confidence": 0.823,
        "action": 1
      },
      {
        "confidence": 0.823,
        "action": 3
      },
      {
        "confidence": 0.823,
        "action": 2
      }
    ],
    "picture": "/9j/4AAQSkZJRgABAQEAYABgAAD/2w..."
  },
  "warning-list": []
}
```

状态码：400

失败响应样例

```
{
  "error_code": "FRS.0701",
  "error_msg": "Parse video data failed."
}
```

状态码

状态码请参见[状态码](#)。

错误码

错误码请参见[错误码](#)。

4.3.2 静默活体检测

功能介绍

静默活体检测是基于人脸图片中可能存在的畸变、摩尔纹、反光、倒影、边框等信息，判断图片中的人脸是否来自于真人活体，进行真人检测，有效抵御纸质翻拍照、电子翻拍照以及视频翻拍等各种攻击方式。静默活体检测支持单张图片，也支持多人脸图片，当图片中有多个脸时，会检测其中最大的人脸是否为活体。新老用户均可用该接口。

前提条件：

请确保您已开通[人脸识别服务](#)，具体操作方法请参见[申请服务](#)。

约束限制：

- 只支持识别JPG、PNG、JPEG、BMP格式的图片。
- application/json请求的body中，请使用标准Json格式。
- Base64编码中请勿使用回车换行。
- 系统不保存用户图片。

- 图片大小小于2MB，由于过大图片会导致时延较长，并且图片信息量不大，建议小于1MB。
- 图片分辨率小于4096*4096，图片中人脸像素大于40*40，建议120*120以上。
- 为保证识别效果，人脸图片建议要求如下：
 - a. 光照大于200lux、无反光强光阴影现象。
 - b. 人脸无遮挡、整体清晰无拖尾抖动等运动模糊。
 - c. 侧脸不超过30°、俯仰角小于15°、偏转角小于15°、图片中人脸保持竖置正脸。
- 其他的约束限制信息请参见1.4 约束限制章节。

建议：

- 由于过大图片对识别算法精度无明显提升，同时会导致时延较长，建议传入图片小于1MB，一般500KB左右足够。
- OBS上存储的图片也建议小于1MB。
- 图片中人脸像素建议120*120以上。

调试

您可以在API Explorer中调试该接口，支持自动认证鉴权。API Explorer可以自动生成SDK代码示例，并提供SDK代码示例调试功能。

URI

POST /v1/{project_id}/live-detect-face

表 4-27 路径参数

参数	是否必选	参数类型	描述
project_id	是	String	项目ID，获取方法请参见 获取项目ID/账号名/AK/SK 。

请求参数

表 4-28 请求 Header 参数

参数	是否必选	参数类型	描述
X-Auth-Token	是	String	用户Token。 用于获取操作API的权限。获取方法请参见 认证鉴权 。
Content-Type	是	String	发送的实体的MIME类型，参数值为“application/json”。

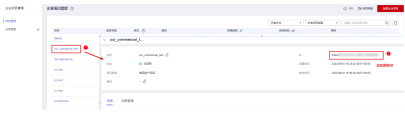
参数	是否必选	参数类型	描述
Enterprise-Project-Id	否	String	<p>企业项目ID。FRS支持通过企业项目管理（EPS）对不同用户组和用户的资源使用，进行分账，当前仅支持按需计费模式。</p> <p>获取方法：进入“企业项目管理”页面，单击企业名称，在企业项目详情页获取Enterprise-Project-Id（企业项目ID）。</p>  <p>说明 创建企业项目后，在传参时，有以下三类场景。</p> <ul style="list-style-type: none"> 携带正确的ID，正常使用FRS服务，账单归到企业ID对应的企业项目中。 携带错误的ID，正常使用FRS服务，账单的企业项目会被分类为“未归集”。 不携带ID，正常使用FRS服务，账单的企业项目会被分类为“default”。

表 4-29 请求 Body 参数

参数名	是否必选	参数类型	说明
image_url	与image_file、image_base64三选一	String	<p>图片的URL路径，目前仅支持华为云上OBS的URL，使用时只需保证FRS有权限读取该OBS桶的数据。开通读取权限的操作请参见申请服务。</p>
image_file	与image_url、image_base64三选一	File	<p>本地图片文件，要求：</p> <ul style="list-style-type: none"> 图片不能超过8MB。 上传文件时，请求格式为multipart。

参数名	是否必选	参数类型	说明
image_base64	与image_file、image_url三选一	String	图像数据，Base64编码，要求： <ul style="list-style-type: none"> • Base64编码后大小不超过8MB。 • 图片为JPG/JPEG/BMP/PNG格式。

响应参数

状态码：200

表 4-30 响应 Body 参数

参数	参数类型	描述
result	result object	静默活体检测结果，LivelessDetectResult结构见表 结构格式说明表 。调用失败时无此字段。
warning-list	Array of WarningList objects	警告信息列表。调用失败时无此字段。

表 4-31 result

参数	参数类型	描述
alive	Boolean	是否是活体。
confidence	Double	置信度，取值范围0~1。 当confidence>0.5时，alive=true，否则alive=false。
picture	String	检测出最大人脸的图片base64字符串。

表 4-32 WarningList

参数	参数类型	描述
warningCode	Integer	警告ID。
warningMsg	String	告警消息。

状态码：400

表 4-33 响应 Body 参数

参数	参数类型	描述
error_code	String	调用失败时的错误码，具体请参考 错误码 。调用成功时无此字段。
error_msg	String	调用失败时的错误信息。调用成功时无此字段。

请求示例

X-Auth-Token值获取方法请参见[快速入门](#)。

- 通过传入图片的BASE64编码，判断图片中的人脸是否来自于真人活体

```
POST https://{endpoint}/v1/{project_id}/live-detect-face
Request Header:
Content-Type: application/json
X-Auth-Token: MIINRwYJKoZIhvcNAQcColINODCCDTQCAQExDT...
Request Body:
{
  "image_base64": "/9j/4AAQSkZJRgABAgEASABIAAD"
}
```

- 通过传入图片文件，判断图片中的人脸是否来自于真人活体

```
POST https://{endpoint}/v1/{project_id}/live-detect-face
Request Header:
X-Auth-Token: MIINRwYJKoZIhvcNAQcColINODCCDTQCAQExDT...
Request Body:
image_file: File(图片文件)
```

- 通过传入图片URL，判断图片中的人脸是否来自于真人活体

```
POST https://{endpoint}/v1/{project_id}/live-detect-face
Request Header:
Content-Type: application/json
X-Auth-Token: MIINRwYJKoZIhvcNAQcColINODCCDTQCAQExDT...
Request Body:
{
  "image_url": "https://<bucket-name>.<endpoint>/<object-name>"
}
```

- 使用Python3语言读取本地图片，判断图片中的人脸是否来自于真人活体

```
# -*- coding:utf-8 -*-

import requests
import base64

endpoint = '开通服务所在region的人脸识别服务域名'
project_id = '开通服务所在region的用户项目ID'
token = "用户获取得到的实际token值"
headers = {'Content-Type': 'application/json', 'X-Auth-Token': token}

url = "https://{endpoint}/v1/{project_id}/live-detect-face".format(endpoint=endpoint,
project_id=project_id)
image_file_path = './data/face-demo.png'
with open(image_file_path, "rb") as bin_data:
    image_data = bin_data.read()
image_base64 = base64.b64encode(image_data).decode("utf-8")
body = {"image_base64": image_base64}
response = requests.post(url, headers=headers, json=body, verify=False)
print(response.text)
```

- 使用Java语言读取图片base64编码，判断图片中的人脸是否来自于真人活体

```
import com.huawei.trace.http.apache.httpclient.TraceApacheHttpClientBuilder;
import org.apache.http.HttpEntity;
import org.apache.http.HttpHeaders;
import org.apache.http.HttpResponse;
```



```
import org.apache.http.client.config.RequestConfig;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.config.Registry;
import org.apache.http.config.RegistryBuilder;
import org.apache.http.conn.socket.ConnectionSocketFactory;
import org.apache.http.conn.socket.PlainConnectionSocketFactory;
import org.apache.http.conn.ssl.NoopHostnameVerifier;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import org.apache.http.entity.ContentType;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;
import org.apache.http.ssl.SSLContextBuilder;
import org.apache.http.ssl.TrustStrategy;
import org.apache.http.util.EntityUtils;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.security.KeyManagementException;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.concurrent.TimeUnit;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.SSLContext;

/**
 * 此demo仅供测试使用，强烈建议使用SDK
 * 使用前需已配置HttpClient jar包。jar包可通过下载SDK获取
 */

public class LiveDetectFace {
    protected static HttpClientBuilder buildClient(HttpClientBuilder httpClientBuilder) {
        SSLContext sslContext = null;
        try {
            sslContext = new SSLContextBuilder().loadTrustMaterial(null, new TrustStrategy() {
                public boolean isTrusted(X509Certificate[] arg0, String arg1) throws CertificateException {
                    return true;
                }
            }).build();
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e.getMessage());
        } catch (KeyManagementException e) {
            throw new RuntimeException(e.getMessage());
        } catch (KeyStoreException e) {
            throw new RuntimeException(e.getMessage());
        }
        httpClientBuilder.setSSLContext(sslContext);
        httpClientBuilder.setConnectionTimeToLive(30, TimeUnit.SECONDS);
        HostnameVerifier hostnameVerifier = NoopHostnameVerifier.INSTANCE;
        SSLConnectionSocketFactory sslSocketFactory = new SSLConnectionSocketFactory(sslContext,
            hostnameVerifier);
        Registry<ConnectionSocketFactory> socketFactoryRegistry
            = RegistryBuilder.<ConnectionSocketFactory>create().register("http",
                PlainConnectionSocketFactory.getSocketFactory()).register("https", sslSocketFactory).build();
        PoolingHttpClientConnectionManager connMgr = new
            PoolingHttpClientConnectionManager(socketFactoryRegistry);
        connMgr.setMaxTotal(200);
        connMgr.setDefaultMaxPerRoute(100);
        httpClientBuilder.setConnectionManager(connMgr);
        return httpClientBuilder;
    }

    public static String doPost(String url, String jsonStr, String token, CloseableHttpClient client) {
        HttpPost post = new HttpPost(url);
        StringEntity entity = new StringEntity(jsonStr, ContentType.APPLICATION_JSON);
        post.setEntity(entity);
        post.setHeader(HttpHeaders.CONTENT_TYPE, "application/json");
    }
}
```

```

//time unit is milliseconds
RequestConfig requestConfig =
RequestConfig.custom().setConnectTimeout(3000).setSocketTimeout(3000).build();
post.setConfig(requestConfig);
post.setHeader("X-Auth-Token", token);
HttpResponse response = null;
String result = "";
try {
    response = client.execute(post);
    HttpEntity responseBody = response.getEntity();
    if (responseBody == null) {
        System.out.println("the response body is null.");
        return result;
    } else {
        byte[] byteStream = EntityUtils.toByteArray(responseBody);
        System.arraycopy(byteStream, 0, new byte[byteStream.length], 0, byteStream.length);
        result = new String(byteStream, StandardCharsets.UTF_8);
    }
} catch (IOException e) {
    e.printStackTrace();
}
return result;
}

public static void main(String[] args) {
    CloseableHttpClient client = buildClient(TraceApacheHttpClientBuilder.create()).build();
    // endpoint和project_id需要替换成实际信息。
    String url = "https://{{endpoint}}/v1/{{project_id}}/live-detect-face";
    String jsonStr = "{ \"image_base64\": \"9j/4AAQSkZJRgABAQAgEASABIAAD...\" }";
    String token = "对应region的token";
    String result = doPost(url, jsonStr, token, client);
    System.out.println(result);
}
}

```

响应示例

状态码：200

成功响应样例

```

{
  "result":{
    "alive": true,
    "confidence": 0.823,
    "picture": "/9j/4AAQSkZJRgABAQEAYABgAAD/2w..."
  },
  "warning-list": []
}

```

状态码：400

失败响应样例

```

{
  "error_code": "FRS.0707",
  "error_msg": "Detect no face, check out your picture."
}

```

状态码

状态码请参见[状态码](#)。

错误码

错误码请参见[错误码](#)。

4.4 人脸搜索

功能介绍

人脸搜索是指在已有的人脸库中，查询与目标人脸相似的一张或者多张人脸，并返回相应的置信度。如果图片中包含多个人脸，选取图片中检测到的最大尺寸人脸作为检索的输入。

支持传入图片或者faceID进行人脸搜索。

前提条件：

请确保您已开通[人脸识别服务](#)，具体操作方法请参见[申请服务](#)。

约束限制：

- 只支持识别JPG、PNG、JPEG、BMP格式的图片。
- application/json请求的body中，请使用标准Json格式。
- Base64编码中请勿使用回车换行。
- 系统不保存用户图片。
- 图片大小**小于8MB**，由于过大图片会导致时延较长，并且图片信息量不大，建议**小于1MB**。
- 图片分辨率**小于4096*4096**，图片中人脸像素**大于80*80**，建议120*120以上。
- 为保证识别效果，人脸图片建议要求如下：
 - a. 光照大于200lux、无反光强光阴影现象。
 - b. 人脸无遮挡、整体清晰无拖尾抖动等运动模糊。
 - c. 侧脸不超过30°、俯仰角小于15°、偏转角小于15°、图片中人脸保持竖置正脸。
- 其他的约束限制信息请参见[约束与限制](#)章节。

建议：

- 由于过大图片对识别算法精度无明显提升，同时会导致时延较长，建议传入图片**小于1MB**，一般**500KB**左右足够。
- OBS上存储的图片也建议**小于1MB**。
- 图片中人脸像素建议**120*120**以上。

调试

您可以在[API Explorer](#)中调试该接口，支持自动认证鉴权。API Explorer可以自动生成SDK代码示例，并提供SDK代码示例调试功能。

URI

POST /v2/{project_id}/face-sets/{face_set_name}/search

表 4-34 路径参数

参数	是否必选	参数类型	描述
project_id	是	String	项目ID, 获取方法请参见 获取项目ID/账号名/AK/SK 。
face_set_name	是	String	人脸库名称, 字符串长度1-64。

请求参数

表 4-35 请求 Header 参数


参数	是否必选	参数类型	描述
X-Auth-Token	是	String	用户Token。 用于获取操作API的权限。获取方法请参见 认证鉴权 。
Content-Type	是	String	发送的实体的MIME类型, 参数值为“application/json”。
Enterprise-Project-Id	否	String	<p>企业项目ID。FRS支持通过企业项目管理（EPS）对不同用户组和用户的资源使用, 进行分账, 当前仅支持按需计费模式。</p> <p>获取方法: 进入“企业项目管理”页面, 单击企业名称, 在企业项目详情页获取Enterprise-Project-Id（企业项目ID）。</p>  <p>说明 创建企业项目后, 在传参时, 有以下三类场景。</p> <ul style="list-style-type: none"> 携带正确的ID, 正常使用FRS服务, 账单的企业项目会被分类到企业ID对应的企业项目中。 携带格式正确但不存在的ID, 正常使用FRS服务, 账单的企业项目会显示对应不存在的企业项目ID。 不携带ID或格式错误ID（包含特殊字符等）, 正常使用FRS服务, 账单的企业项目会被分类到"default"中。

表 4-36 请求 Body 参数

参数名	参数类型	是否必选	说明
image_url	String	与 image_file、image_base64、face_id 四选一	图片的URL路径，目前仅支持华为云上OBS的URL，使用时只需保证FRS有权限读取该OBS桶的数据。开通读取权限的操作请参见 服务授权 。
image_file	File	与 image_url、image_base64、face_id 四选一	本地图片文件，图片不能超过 8MB ，建议 小于1MB 。上传文件时，请求格式为multipart。
image_base64	String	与 image_file、image_url、face_id 四选一	图像数据，Base64编码，要求： <ul style="list-style-type: none"> Base64编码后大小不超过8MB，建议小于1MB。 图片为JPG/JPEG/BMP/PNG格式。
face_id	String	与 image_file、image_base64、image_url 四选一	导入人脸时，系统返回的人脸编号，即人脸ID。
top_n	Integer	否	返回查询到的最相似的N张人脸，N默认为10，取值范围[0,1000]。N张人脸按照置信度降序排序，置信度越大越靠前。
threshold	Double	否	人脸相似度阈值，低于这个阈值则不返回，取值范围[0,1]，一般情况下建议取值0.93，默认为0。
sort	JSONArray	否	支持字段排序，参考 sort语法 。
filter	String	否	过滤条件，参考 filter语法 。
return_fields	JSONArray	否	指定返回的自定义字段。

响应参数

状态码：200

表 4-37 响应 Body 参数

参数	参数类型	描述
faces	Array of SearchFace objects	查找的人脸集合，详见 SearchFace 。调用失败时无此字段。

表 4-38 SearchFace

参数	参数类型	描述
bounding_box	BoundingBox object	人脸在图像中的位置。
similarity	Double	人脸搜索时用于被检索的相似度。
external_fields	Object	用户添加的额外自定义字段。
external_image_id	String	人脸所在的外部图片ID。
face_id	String	人脸ID，由系统内部生成的唯一ID。

表 4-39 BoundingBox

参数	参数类型	描述
width	Integer	矩形框宽度。
top_left_y	Integer	矩形框左上角纵坐标。
top_left_x	Integer	矩形框左上角横坐标。
height	Integer	矩形框高度。

状态码： 400

表 4-40 响应 Body 参数

参数	参数类型	描述
error_code	String	调用失败时的错误码，具体请参考 错误码 。调用成功时无此字段。
error_msg	String	调用失败时的错误信息。调用成功时无此字段。

请求示例

X-Auth-Token值获取方法请参见[快速入门](#)。

- 在人脸库中，查询与目标人脸相似的一张或者多张人脸，目标人脸通过图片的base64编码传入

```
POST https://{endpoint}/v2/{project_id}/face-sets/showFaceSet/search
Request Header:
Content-Type: application/json
X-Auth-Token: MIINRwYJKoZihvcNAQcColINODCCDTQCAQExDT...

Request Body:
{
  "image_base64": "/9j/4AAQSkZJRgABAQgEASABIAAD...",
  "sort": [
    {
      "timestamp": "asc"
    }
  ],
  "return_fields": ["timestamp", "id"],
  "filter": "timestamp:12"
}
```

- 在人脸库中，查询与目标人脸相似的一张或者多张人脸，目标人脸通过图片文件传入

```
POST https://{endpoint}/v2/{project_id}/face-sets/showFaceSet/search
Request Header:
X-Auth-Token: MIINRwYJKoZihvcNAQcColINODCCDTQCAQExDT...

Request Body:
image_file: File(图片文件)
return_fields: ["timestamp", "id"]
filter: timestamp:12
```

- 在人脸库中，查询与目标人脸相似的一张或者多张人脸，目标人脸通过图片的url传入

```
POST https://{endpoint}/v2/{project_id}/face-sets/showFaceSet/search
Request Header:
Content-Type: application/json
X-Auth-Token: MIINRwYJKoZihvcNAQcColINODCCDTQCAQExDT...

Request Body:
{
  "image_url": "https://<bucket-name>.<endpoint>/<object-name>",
  "sort": [
    {
      "timestamp": "asc"
    }
  ],
  "return_fields": ["timestamp", "id"],
  "filter": "timestamp:12"
}
```

- 在人脸库中，查询与目标人脸相似的一张或者多张人脸，目标人脸通过人脸ID传入

```
POST https://{endpoint}/v2/{project_id}/face-sets/showFaceSet/search
Request Header:
Content-Type: application/json
X-Auth-Token: MIINRwYJKoZihvcNAQcColINODCCDTQCAQExDT...

Request Body:
{
  "face_id": "6KLB1Ktu",
  "sort": [
    {
      "timestamp": "asc"
    }
  ],
}
```

```
"return_fields": ["timestamp", "id"],
"filter": "timestamp:12"
}
```

- 使用Python3语言在人脸库中，查询与目标人脸相似的一张或者多张人脸
-*- coding:utf-8 -*-

```
import requests
import base64

endpoint = '开通服务所在region的人脸识别服务域名'
project_id = '开通服务所在region的用户项目ID'
token = "用户获取到的实际token值"
headers = {'Content-Type': 'application/json', 'X-Auth-Token': token}

face_set_name = 'showFaceSet'
url = "https://{endpoint}/v2/{project_id}/face-sets/{face_set_name}/search".format(
    endpoint=endpoint, project_id=project_id, face_set_name=face_set_name)
image_file_path = './data/face-demo.png'
with open(image_file_path, "rb") as bin_data:
    image_data = bin_data.read()
image_base64 = base64.b64encode(image_data).decode("utf-8")
body = {"image_base64": image_base64, "sort": [{"timestamp": "asc"}], "return_fields": ["timestamp",
"id"], "filter": "timestamp:12"}
response = requests.post(url, headers=headers, json=body, verify=False)
print(response.text)
```

- 使用Java语言在人脸库中，查询与目标人脸相似的一张或者多张人脸

```
import com.huawei.trace.http.apache.httpclient.TraceApacheHttpClientBuilder;
import org.apache.http.HttpEntity;
import org.apache.http.HttpHeaders;
import org.apache.http.HttpResponse;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.config.Registry;
import org.apache.http.config.RegistryBuilder;
import org.apache.http.conn.socket.ConnectionSocketFactory;
import org.apache.http.conn.socket.PlainConnectionSocketFactory;
import org.apache.http.conn.ssl.NoopHostnameVerifier;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import org.apache.http.entity.ContentType;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;
import org.apache.http.ssl.SSLContextBuilder;
import org.apache.http.ssl.TrustStrategy;
import org.apache.http.util.EntityUtils;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.security.KeyManagementException;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.concurrent.TimeUnit;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.SSLContext;

/**
 * 此demo仅供测试使用，强烈建议使用SDK
 * 使用前需已配置HttpClient jar包。jar包可通过下载SDK获取
 */

public class FaceSearch {
    protected static HttpClientBuilder buildClient(HttpClientBuilder httpClientBuilder) {
        SSLContext sslContext = null;
        try {
            sslContext = new SSLContextBuilder().loadTrustMaterial(null, new TrustStrategy() {
                public boolean isTrusted(X509Certificate[] arg0, String arg1) throws CertificateException {
```



```

        return true;
    }
    }).build();
} catch (NoSuchAlgorithmException e) {
    throw new RuntimeException(e.getMessage());
} catch (KeyManagementException e) {
    throw new RuntimeException(e.getMessage());
} catch (KeyStoreException e) {
    throw new RuntimeException(e.getMessage());
}
httpClientBuilder.setSSLContext(sslContext);
httpClientBuilder.setConnectionTimeToLive(30, TimeUnit.SECONDS);
HostnameVerifier hostnameVerifier = NoopHostnameVerifier.INSTANCE;
SSLConnectionSocketFactory sslSocketFactory = new SSLConnectionSocketFactory(sslContext,
hostnameVerifier);
Registry<ConnectionSocketFactory> socketFactoryRegistry
    = RegistryBuilder.<ConnectionSocketFactory>create().register("http",
    PlainConnectionSocketFactory.getSocketFactory()).register("https", sslSocketFactory).build();
PoolingHttpClientConnectionManager connMgr = new
PoolingHttpClientConnectionManager(socketFactoryRegistry);
connMgr.setMaxTotal(200);
connMgr.setDefaultMaxPerRoute(100);
httpClientBuilder.setConnectionManager(connMgr);
return httpClientBuilder;
}

public static String doPost(String url, String jsonStr, String token, CloseableHttpClient client) {
    HttpPost post = new HttpPost(url);
    StringEntity entity = new StringEntity(jsonStr, ContentType.APPLICATION_JSON);
    post.setEntity(entity);
    post.setHeader(HttpHeaders.CONTENT_TYPE, "application/json");
    //time unit is milliseconds
    RequestConfig requestConfig =
RequestConfig.custom().setConnectTimeout(3000).setSocketTimeout(3000).build();
    post.setConfig(requestConfig);
    post.setHeader("X-Auth-Token", token);
    HttpResponse response = null;
    String result = "";
    try {
        response = client.execute(post);
        HttpEntity responseBody = response.getEntity();
        if (responseBody == null) {
            System.out.println("the response body is null.");
            return result;
        } else {
            byte[] byteStream = EntityUtils.toByteArray(responseBody);
            System.arraycopy(byteStream, 0, new byte[byteStream.length], 0, byteStream.length);
            result = new String(byteStream, StandardCharsets.UTF_8);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return result;
}

public static void main(String[] args) {
    CloseableHttpClient client = buildClient(TraceApacheHttpClientBuilder.create()).build();
    // endpoint,project_id和face_set_name需要替换成实际信息。
    String url = "https://{{endpoint}}/v2/{{project_id}}/face-sets/{{face_set_name}}/search";
    String jsonStr = "{\"image_base64\": \"\\/9j\\/4AAQSkZJRgABAQgEASABIAAD...\", \"sort\":
[[{\"timestamp\": \"asc\"}], \"return_fields\": [\"timestamp\", \"id\"], \"filter\": \"timestamp:12\"}";
    String token = "对应region的token";
    String result = doPost(url, jsonStr, token, client);
    System.out.println(result);
}
}

```

响应示例

状态码：200

成功响应样例

```
{
  "faces": [
    {
      "bounding_box": {
        "width": 170,
        "top_left_y": 37,
        "top_left_x": 20,
        "height": 170
      },
      "similarity": 0.996146,
      "external_image_id": "123",
      "external_fields": {
        "id": "home",
        "timestamp": 12
      },
      "face_id": "6KLB1Ktu"
    },
    {
      "bounding_box": {
        "width": 170,
        "top_left_y": 37,
        "top_left_x": 20,
        "height": 170
      },
      "similarity": 0.996146,
      "external_image_id": "12",
      "external_fields": {
        "id": "home1",
        "timestamp": 12
      },
      "face_id": "PexOpqRj"
    }
  ]
}
```

状态码：400

失败响应样例

```
{
  "error_code": "FRS.0018",
  "error_msg": "The service inner error."
}
```

状态码

状态码请参见[状态码](#)。

错误码

错误码请参见[错误码](#)。

4.5 人脸库资源管理

4.5.1 创建人脸库

功能介绍

创建用于存储人脸特征的人脸库。您最多可以创建10个人脸库，每个人脸库最大容量为10万个人脸特征。如有更大规格的需求请联系客服。

前提条件:

请确保您已开通[人脸搜索服务](#)。

📖 说明

- 默认情况下，一个人脸库最大可支持10万个人脸特征，一个用户最多可创建10个人脸库，最多可支持10*10万（100万）个人脸特征。
- 如您的需求超出100万个人脸特征，可通过[工单](#)或者服务热线（4000-955-988或950808 转1）与我们联系，咨询具体解决方案。

调试

您可以在[API Explorer](#)中调试该接口，支持自动认证鉴权。API Explorer可以自动生成SDK代码示例，并提供SDK代码示例调试功能。

URI

POST /v2/{project_id}/face-sets

表 4-41 路径参数

参数	是否必选	参数类型	描述
project_id	是	String	项目ID，获取方法请参见 获取项目ID/账号名/AK/SK 。

请求参数

表 4-42 请求 Header 参数

参数	是否必选	参数类型	描述
X-Auth-Token	是	String	用户Token。 用于获取操作API的权限。获取方法请参见 认证鉴权 。
Content-Type	是	String	发送的实体的MIME类型，参数值为“application/json”。

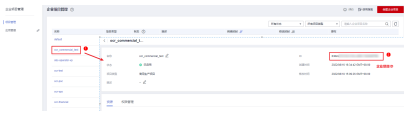
参数	是否必选	参数类型	描述
Enterprise-Project-Id	否	String	<p>企业项目ID。FRS支持通过企业项目管理（EPS）对不同用户组和用户的资源使用，进行分账，当前仅支持按需计费模式。</p> <p>获取方法：进入“企业项目管理”页面，单击企业名称，在企业项目详情页获取Enterprise-Project-Id（企业项目ID）。</p>  <p>说明 创建企业项目后，在传参时，有以下三类场景。</p> <ul style="list-style-type: none"> 携带正确的ID，正常使用FRS服务，账单的企业项目会被分类到企业ID对应的企业项目中。 携带格式正确但不存在的ID，正常使用FRS服务，账单的企业项目会显示对应不存在的企业项目ID。 不携带ID或格式错误ID（包含特殊字符等），正常使用FRS服务，账单的企业项目会被分类到"default"中。

表 4-43 请求 Body 参数

参数	是否必选	参数类型	描述
external_fields	否	Map<String, T ypeInfo>	<p>用户自定义数据，自定义字段不能以vector、bounding_box、external_image_id、face_id、create_time、_id、_all、_source等字段命名，这些字段为内置字段，自定义字段避免使用。</p> <p>自定义字段的key值长度范围为[1,36]，string类型的value长度范围为[1,256]，具体参见自定义字段。</p> <p>如果在后续操作中，使用该自定义字段向人脸库中添加人脸，需要在创建人脸库时定义该字段。</p>

参数	是否必选	参数类型	描述
face_set_name	是	String	人脸库的名称，字符串长度1-64。 建议人脸库的名称不要以下划线（_）开头，否则云监控服务会无法采集人脸数量。
face_set_capacity	否	Integer	人脸库最大的容量，填写1万整数倍的数字，例如：30000。 默认为100000，最大值为100000，可通过创建新的人脸库进行扩容，每个用户可使用10个人脸库，每个人脸库容量为10万个人脸特征。如需扩容单个人脸库规模，请联系华为云客服确认扩容规模与价格。

表 4-44 TypeInfo

参数	是否必选	参数类型	描述
type	否	String	数据类型支持string、integer、double、long等。

响应参数

状态码：200

表 4-45 响应 Body 参数

参数	参数类型	描述
face_set_info	FaceSetInfo object	人脸库信息，详见 FaceSetInfo 。调用失败时无此字段。

表 4-46 FaceSetInfo

参数	参数类型	描述
face_number	Integer	人脸库中已有的人脸特征的数量。
external_fields	Object	用户的自定义字段。
face_set_id	String	人脸库ID，随机生成的包含八个字符的字符串。

参数	参数类型	描述
face_set_name	String	人脸库名称。
create_date	String	创建时间。
face_set_capacity	Integer	人脸库最大的容量。创建人脸库时，请求参数如果不设置face_set_capacity参数，默认每个人脸库最大容量为10万个人脸特征。

状态码： 400

表 4-47 响应 Body 参数

参数	参数类型	描述
error_code	String	调用失败时的错误码，具体请参考 错误码 。调用成功时无此字段。
error_msg	String	调用失败时的错误信息。调用成功时无此字段。

请求示例

X-Auth-Token值获取方法请参见[快速入门](#)。

- 创建人脸库，并设置为可存储10万个人脸特征

```
POST https://{endpoint}/v2/{project_id}/face-sets
Request Header:
Content-Type: application/json
X-Auth-Token: MIINRwYJKoZihvcNAQcCoIIINODCCDTQCAQExDT...
```

```
Request Body:
{
  "face_set_name": "test",
  "face_set_capacity": 100000,
  "external_fields": {
    "timestamp": {
      "type": "long"
    },
    "id": {
      "type": "string"
    },
    "number": {
      "type": "integer"
    }
  }
}
```

- 使用Python3语言创建人脸库，并设置为可存储10万个人脸特征

```
# -*- coding:utf-8 -*-

import requests
import base64

endpoint = '开通服务所在region的人脸识别服务域名'
project_id = '开通服务所在region的用户项目ID'
```

```
token = '用户获取到的实际token值'  
headers = {'Content-Type': 'application/json', 'X-Auth-Token': token}  
  
url = "https://{endpoint}/v2/{project_id}/face-sets".format(endpoint=endpoint,project_id=project_id)  
body = {  
    "face_set_name": "test",  
    "face_set_capacity": 100000,  
    "external_fields" : {  
        "timestamp" : {  
            "type" : "long"  
        },  
        "id" : {  
            "type" : "string"  
        },  
        "number" : {  
            "type" : "integer"  
        }  
    }  
}  
response = requests.post(url, headers=headers, json=body, verify=False)  
print(response.text)
```

- 使用Java语言创建人脸库，并设置为可存储10万个人脸特征

```
import com.huawei.trace.http.apache.httpclient.TraceApacheHttpClientBuilder;  
import org.apache.http.HttpEntity;  
import org.apache.http.HttpHeaders;  
import org.apache.http.HttpResponse;  
import org.apache.http.client.config.RequestConfig;  
import org.apache.http.client.methods.HttpPost;  
import org.apache.http.config.Registry;  
import org.apache.http.config.RegistryBuilder;  
import org.apache.http.conn.socket.ConnectionSocketFactory;  
import org.apache.http.conn.socket.PlainConnectionSocketFactory;  
import org.apache.http.conn.ssl.NoopHostnameVerifier;  
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;  
import org.apache.http.entity.ContentType;  
import org.apache.http.entity.StringEntity;  
import org.apache.http.impl.client.CloseableHttpClient;  
import org.apache.http.impl.client.HttpClientBuilder;  
import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;  
import org.apache.http.ssl.SSLContextBuilder;  
import org.apache.http.ssl.TrustStrategy;  
import org.apache.http.util.EntityUtils;  
import java.io.IOException;  
import java.nio.charset.StandardCharsets;  
import java.security.KeyManagementException;  
import java.security.KeyStoreException;  
import java.security.NoSuchAlgorithmException;  
import java.security.cert.CertificateException;  
import java.security.cert.X509Certificate;  
import java.util.concurrent.TimeUnit;  
import javax.net.ssl.HostnameVerifier;  
import javax.net.ssl.SSLContext;  
  
/**  
 * 此demo仅供测试使用，强烈建议使用SDK  
 * 使用前需已配置HttpClient jar包。jar包可通过下载SDK获取  
 */  
  
public class CreateFaceSet {  
    protected static HttpClientBuilder buildClient(HttpClientBuilder httpClientBuilder) {  
        SSLContext sslContext = null;  
        try {  
            sslContext = new SSLContextBuilder().loadTrustMaterial(null, new TrustStrategy() {  
                public boolean isTrusted(X509Certificate[] arg0, String arg1) throws CertificateException {  
                    return true;  
                }  
            }).build();  
        } catch (NoSuchAlgorithmException e) {  
            throw new RuntimeException(e.getMessage());  
        }  
    }  
}
```

```

    } catch (KeyManagementException e) {
        throw new RuntimeException(e.getMessage());
    } catch (KeyStoreException e) {
        throw new RuntimeException(e.getMessage());
    }
    httpClientBuilder.setSSLContext(sslContext);
    httpClientBuilder.setConnectionTimeToLive(30, TimeUnit.SECONDS);
    HostnameVerifier hostnameVerifier = NoopHostnameVerifier.INSTANCE;
    SSLConnectionSocketFactory sslSocketFactory = new SSLConnectionSocketFactory(sslContext,
hostnameVerifier);
    Registry<ConnectionSocketFactory> socketFactoryRegistry
        = RegistryBuilder.<ConnectionSocketFactory>create().register("http",
        PlainConnectionSocketFactory.getSocketFactory()).register("https", sslSocketFactory).build();
    PoolingHttpClientConnectionManager connMgr = new
PoolingHttpClientConnectionManager(socketFactoryRegistry);
    connMgr.setMaxTotal(200);
    connMgr.setDefaultMaxPerRoute(100);
    httpClientBuilder.setConnectionManager(connMgr);
    return httpClientBuilder;
}

public static String doPost(String url, String jsonStr, String token, CloseableHttpClient client) {
    HttpPost post = new HttpPost(url);
    StringEntity entity = new StringEntity(jsonStr, ContentType.APPLICATION_JSON);
    post.setEntity(entity);
    post.setHeader(HttpHeaders.CONTENT_TYPE, "application/json");
    //time unit is milliseconds
    RequestConfig requestConfig =
RequestConfig.custom().setConnectTimeout(3000).setSocketTimeout(3000).build();
    post.setConfig(requestConfig);
    post.setHeader("X-Auth-Token", token);
    HttpResponse response = null;
    String result = "";
    try {
        response = client.execute(post);
        HttpEntity responseBody = response.getEntity();
        if (responseBody == null) {
            System.out.println("the response body is null.");
            return result;
        } else {
            byte[] byteStream = EntityUtils.toByteArray(responseBody);
            System.arraycopy(byteStream, 0, new byte[byteStream.length], 0, byteStream.length);
            result = new String(byteStream, StandardCharsets.UTF_8);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return result;
}

public static void main(String[] args) {
    CloseableHttpClient client = buildClient(TraceApacheHttpClientBuilder.create()).build();
    // endpoint和project_id需要替换成实际信息。
    String url = "https://{{endpoint}}/v2/{{project_id}}/face-sets";
    String jsonStr = "{ \"face_set_name\": \"test\", \"face_set_capacity\": 100000, \"external_fields\":
{ \"timestamp\": { \"type\": \"long\" }, \"id\": { \"type\": \"string\" }, \"number\": { \"type\":
\"integer\" }}}";
    String token = "对应region的token";
    String result = doPost(url, jsonStr, token, client);
    System.out.println(result);
}
}

```

响应示例

状态码：200

成功响应样例


```
{
  "face_set_info": {
    "face_number": 0,
    "face_set_id": "WYYOFIGb",
    "face_set_name": "test",
    "create_date": "2018-05-28 02:19:00",
    "face_set_capacity": 10000,
    "external_fields": {
      "timestamp": {
        "type": "long"
      },
      "id": {
        "type": "string"
      },
      "number": {
        "type": "integer"
      }
    }
  }
}
```

状态码：400

失败响应样例

```
{
  "error_code": "FRS.0002",
  "error_msg": "The authentication token is abnormal."
}
```

状态码

状态码请参见[状态码](#)。

错误码

错误码请参见[错误码](#)。

4.5.2 查询所有人脸库

功能介绍

查询当前用户所有人脸库的状态信息。

前提条件：

请确保您已开通[人脸搜索服务](#)。

调试

您可以在[API Explorer](#)中调试该接口，支持自动认证鉴权。API Explorer可以自动生成SDK代码示例，并提供SDK代码示例调试功能。

URI

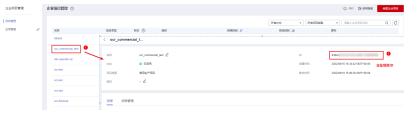
GET /v2/{project_id}/face-sets

表 4-48 路径参数

参数	是否必选	参数类型	描述
project_id	是	String	项目ID，获取方法请参见 获取项目ID/账号名/AK/SK 。

请求参数

表 4-49 请求 Header 参数

参数	是否必选	参数类型	描述
X-Auth-Token	是	String	用户Token。 用于获取操作API的权限。获取方法请参见 认证鉴权 。
Content-Type	是	String	发送的实体的MIME类型，参数值为“application/json”。
Enterprise-Project-Id	否	String	<p>企业项目ID。FRS支持通过企业项目管理（EPS）对不同用户组和用户的资源使用，进行分账，当前仅支持按需计费模式。</p> <p>获取方法：进入“企业项目管理”页面，单击企业名称，在企业项目详情页获取Enterprise-Project-Id（企业项目ID）。</p>  <p>说明 创建企业项目后，在传参时，有以下三类场景。</p> <ul style="list-style-type: none"> 携带正确的ID，正常使用FRS服务，账单的企业项目会被分类到企业ID对应的企业项目中。 携带格式正确但不存在的ID，正常使用FRS服务，账单的企业项目会显示对应不存在的企业项目ID。 不携带ID或格式错误ID（包含特殊字符等），正常使用FRS服务，账单的企业项目会被分类到"default"中。

响应参数

状态码：200

表 4-50 响应 Body 参数

参数	参数类型	描述
face_sets_info	Array of FaceSetInfo objects	人脸库信息集合，详见 FaceSetInfo 。调用失败时无此字段。

表 4-51 FaceSetInfo

参数	参数类型	描述
face_number	Integer	人脸库中已有的人脸特征的数量。
external_fields	Object	用户的自定义字段。
face_set_id	String	人脸库ID，随机生成的包含八个字符的字符串。
face_set_name	String	人脸库名称。
create_date	String	创建时间。
face_set_capacity	Integer	人脸库最大的容量。创建人脸库时，请求参数如果不设置face_set_capacity参数，默认每个人脸库最大容量为10万个人脸特征。

状态码： 400

表 4-52 响应 Body 参数

参数	参数类型	描述
error_code	String	调用失败时的错误码，具体请参考 错误码 。调用成功时无此字段。
error_msg	String	调用失败时的错误信息。调用成功时无此字段。

请求示例

X-Auth-Token值获取方法请参见[快速入门](#)。

- 查询所拥有的全部人脸库信息

```
GET https://{endpoint}/v2/{project_id}/face-sets
Request Header:
Content-Type: application/json
X-Auth-Token: MIINRwYJKoZlhvcNAQcCollINODCCDTQCAQExDT...
```
- 使用Python3语言查询所拥有的全部人脸库信息

```
# -*- coding:utf-8 -*-
import requests
```

```
import base64

endpoint = '开通服务所在region的人脸识别服务域名'
project_id = '开通服务所在region的用户项目ID'
token = '用户获取得到的实际token值'
headers = {'Content-Type': 'application/json', 'X-Auth-Token': token}

url = "https://{endpoint}/v2/{project_id}/face-sets".format(endpoint=endpoint,project_id=project_id)
response = requests.get(url, headers=headers, verify=False)
print(response.text)
```

- 使用Java语言查询所拥有的全部人脸库信息

```
import com.huawei.trace.http.apache.httpclient.TraceApacheHttpClientBuilder;
import org.apache.http.HttpEntity;
import org.apache.http.HttpHeaders;
import org.apache.http.HttpResponse;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.config.Registry;
import org.apache.http.config.RegistryBuilder;
import org.apache.http.conn.socket.ConnectionSocketFactory;
import org.apache.http.conn.socket.PlainConnectionSocketFactory;
import org.apache.http.conn.ssl.NoopHostnameVerifier;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import org.apache.http.entity.ContentType;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;
import org.apache.http.ssl.SSLContextBuilder;
import org.apache.http.ssl.TrustStrategy;
import org.apache.http.util.EntityUtils;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.security.KeyManagementException;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.concurrent.TimeUnit;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.SSLContext;

/**
 * 此demo仅供测试使用，强烈建议使用SDK
 * 使用前需已配置HttpClient jar包。jar包可通过下载SDK获取
 */

public class GetAllFaceSets {
    protected static HttpClientBuilder buildClient(HttpClientBuilder httpClientBuilder) {
        SSLContext sslContext = null;
        try {
            sslContext = new SSLContextBuilder().loadTrustMaterial(null, new TrustStrategy() {
                public boolean isTrusted(X509Certificate[] arg0, String arg1) throws CertificateException {
                    return true;
                }
            }).build();
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e.getMessage());
        } catch (KeyManagementException e) {
            throw new RuntimeException(e.getMessage());
        } catch (KeyStoreException e) {
            throw new RuntimeException(e.getMessage());
        }
        httpClientBuilder.setSSLContext(sslContext);
        httpClientBuilder.setConnectionTimeToLive(30, TimeUnit.SECONDS);
        HostnameVerifier hostnameVerifier = NoopHostnameVerifier.INSTANCE;
        SSLConnectionSocketFactory sslSocketFactory = new SSLConnectionSocketFactory(sslContext,
            hostnameVerifier);
```

```
Registry<ConnectionSocketFactory> socketFactoryRegistry
    = RegistryBuilder.<ConnectionSocketFactory>create().register("http",
        PlainConnectionSocketFactory.getSocketFactory()).register("https", sslSocketFactory).build();
PoolingHttpClientConnectionManager connMgr = new
PoolingHttpClientConnectionManager(socketFactoryRegistry);
connMgr.setMaxTotal(200);
connMgr.setDefaultMaxPerRoute(100);
httpClientBuilder.setConnectionManager(connMgr);
return httpClientBuilder;
}

public static String doGet(String url, String token, CloseableHttpClient client) {
    HttpGet get = new HttpGet(url);
    get.setHeader(HttpHeaders.CONTENT_TYPE, "application/json");
    //time unit is milliseconds
    RequestConfig requestConfig =
RequestConfig.custom().setConnectTimeout(3000).setSocketTimeout(3000).build();
    get.setConfig(requestConfig);
    get.setHeader("X-Auth-Token", token);
    HttpResponse response = null;
    String result = "";
    try {
        response = client.execute(get);
        HttpEntity responseBody = response.getEntity();
        if (responseBody == null) {
            System.out.println("the response body is null.");
            return result;
        } else {
            byte[] byteArray = EntityUtils.toByteArray(responseBody);
            System.arraycopy(byteArray, 0, new byte[byteArray.length], 0, byteArray.length);
            result = new String(byteArray, StandardCharsets.UTF_8);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return result;
}

public static void main(String[] args) {
    CloseableHttpClient client = buildClient(TraceApacheHttpClientBuilder.create()).build();
    // endpoint和project_id需要替换成实际信息。
    String url = "https://{{endpoint}}/v2/{{project_id}}/face-sets";
    String token = "对应region的token";
    String result = doGet(url, token, client);
    System.out.println(result);
}
}
```

响应示例

状态码: 200

成功响应样例

```
{
  "face_sets_info": [
    {
      "face_number": 0,
      "face_set_id": "yIXMMZTO",
      "face_set_name": "test",
      "create_date": "2018-05-11 07:49:40",
      "face_set_capacity": 10000,
      "external_fields": {
        "number": {
          "type": "integer"
        },
        "id": {
          "type": "string"
        }
      },
      "timestamp": {
```

```
    "type": "long"
  }
}
]
}
```

状态码：400

失败响应样例

```
{
  "error_code": "FRS.0002",
  "error_msg": "The authentication token is abnormal."
}
```

状态码

状态码请参见[状态码](#)。

错误码

错误码请参见[错误码](#)。

4.5.3 查询人脸库

功能介绍

查询人脸库当前的状态。

前提条件：

请确保您已开通[人脸搜索服务](#)。

调试

您可以在[API Explorer](#)中调试该接口，支持自动认证鉴权。API Explorer可以自动生成SDK代码示例，并提供SDK代码示例调试功能。

URI


GET /v2/{project_id}/face-sets/{face_set_name}

表 4-53 路径参数

参数	是否必选	参数类型	描述
project_id	是	String	项目ID，获取方法请参见 获取项目ID/账号名/AK/SK 。
face_set_name	String	是	人脸库名称。

请求参数

表 4-54 请求 Header 参数

参数	是否必选	参数类型	描述
X-Auth-Token	是	String	用户Token。 用于获取操作API的权限。获取方法请参见 认证鉴权 。
Content-Type	是	String	发送的实体的MIME类型，参数值为“application/json”。
Enterprise-Project-Id	否	String	<p>企业项目ID。FRS支持通过企业项目管理（EPS）对不同用户组和用户的资源使用，进行分账，当前仅支持按需计费模式。</p> <p>获取方法：进入“企业项目管理”页面，单击企业名称，在企业项目详情页获取Enterprise-Project-Id（企业项目ID）。</p>  <p>说明 创建企业项目后，在传参时，有以下三类场景。</p> <ul style="list-style-type: none"> 携带正确的ID，正常使用FRS服务，账单的企业项目会被分类到企业ID对应的企业项目中。 携带格式正确但不存在的ID，正常使用FRS服务，账单的企业项目会显示对应不存在的企业项目ID。 不携带ID或格式错误ID（包含特殊字符等），正常使用FRS服务，账单的企业项目会被分类到"default"中。

响应参数

状态码：200

表 4-55 响应 Body 参数

参数	参数类型	描述
face_set_info	FaceSetInfo object	人脸库信息集合，详见 FaceSetInfo 。调用失败时无此字段。

表 4-56 FaceSetInfo

参数	参数类型	描述
face_number	Integer	人脸库当中的人脸数量。
external_fields	Object	用户的自定义字段。
face_set_id	String	人脸库ID，随机生成的包含八个字符的字符串。
face_set_name	String	人脸库名称。
create_date	String	创建时间。
face_set_capacity	Integer	人脸库最大的容量。

状态码： 400

表 4-57 响应 Body 参数

参数	参数类型	描述
error_code	String	调用失败时的错误码，具体请参考 错误码 。调用成功时无此字段。
error_msg	String	调用失败时的错误信息。调用成功时无此字段。

请求示例

X-Auth-Token值获取方法请参见[快速入门](#)。

- 查询指定人脸库的信息

```
GET https://{endpoint}/v2/{project_id}/face-sets/showFaceSet
Request Header:
Content-Type: application/json
X-Auth-Token: MIINRwYJKoZlHvcNAQcColINODCCDTQCAQExDT...
```

- 使用Python3语言查询指定人脸库的信息

```
# -*- coding:utf-8 -*-

import requests
import base64

endpoint = '开通服务所在region的人脸识别服务域名'
project_id = '开通服务所在region的用户项目ID'
token = '用户获取得到的实际token值'
headers = {'Content-Type': 'application/json', 'X-Auth-Token': token}

face_set_name = 'showFaceSet'
url = "https://{endpoint}/v2/{project_id}/face-sets/{face_set_name}".format(
    endpoint=endpoint,project_id=project_id,face_set_name=face_set_name)
response = requests.get(url, headers=headers, verify=False)
print(response.text)
```


- 使用Java语言查询指定人脸库的信息

```
import com.huawei.trace.http.apache.httpclient.TraceApacheHttpClientBuilder;
import org.apache.http.HttpEntity;
import org.apache.http.HttpHeaders;
import org.apache.http.HttpResponse;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.config.Registry;
import org.apache.http.config.RegistryBuilder;
import org.apache.http.conn.socket.ConnectionSocketFactory;
import org.apache.http.conn.socket.PlainConnectionSocketFactory;
import org.apache.http.conn.ssl.NoopHostnameVerifier;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;
import org.apache.http.ssl.SSLContextBuilder;
import org.apache.http.ssl.TrustStrategy;
import org.apache.http.util.EntityUtils;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.security.KeyManagementException;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.concurrent.TimeUnit;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.SSLContext;

/**
 * 此demo仅供测试使用，强烈建议使用SDK
 * 使用前需已配置HttpClient jar包。jar包可通过下载SDK获取
 */

public class GetFaceSetByFaceSetName {
    protected static HttpClientBuilder buildClient(HttpClientBuilder httpClientBuilder) {
        SSLContext sslContext = null;
        try {
            sslContext = new SSLContextBuilder().loadTrustMaterial(null, new TrustStrategy() {
                public boolean isTrusted(X509Certificate[] arg0, String arg1) throws CertificateException {
                    return true;
                }
            }).build();
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e.getMessage());
        } catch (KeyManagementException e) {
            throw new RuntimeException(e.getMessage());
        } catch (KeyStoreException e) {
            throw new RuntimeException(e.getMessage());
        }
        httpClientBuilder.setSSLContext(sslContext);
        httpClientBuilder.setConnectionTimeToLive(30, TimeUnit.SECONDS);
        HostnameVerifier hostnameVerifier = NoopHostnameVerifier.INSTANCE;
        SSLConnectionSocketFactory sslSocketFactory = new SSLConnectionSocketFactory(sslContext,
        hostnameVerifier);
        Registry<ConnectionSocketFactory> socketFactoryRegistry
            = RegistryBuilder.<ConnectionSocketFactory>create().register("http",
            PlainConnectionSocketFactory.getSocketFactory()).register("https", sslSocketFactory).build();
        PoolingHttpClientConnectionManager connMgr = new
        PoolingHttpClientConnectionManager(socketFactoryRegistry);
        connMgr.setMaxTotal(200);
        connMgr.setDefaultMaxPerRoute(100);
        httpClientBuilder.setConnectionManager(connMgr);
        return httpClientBuilder;
    }

    public static String doGet(String url, String token, CloseableHttpClient client) {
        HttpGet get = new HttpGet(url);
```

```
get.setHeader(HttpHeaders.CONTENT_TYPE, "application/json");
//time unit is milliseconds
RequestConfig requestConfig =
RequestConfig.custom().setConnectTimeout(3000).setSocketTimeout(3000).build();
get.setConfig(requestConfig);
get.setHeader("X-Auth-Token", token);
HttpResponse response = null;
String result = "";
try {
    response = client.execute(get);
    HttpEntity responseBody = response.getEntity();
    if (responseBody == null) {
        System.out.println("the response body is null.");
        return result;
    } else {
        byte[] byteStream = EntityUtils.toByteArray(responseBody);
        System.arraycopy(byteStream, 0, new byte[byteStream.length], 0, byteStream.length);
        result = new String(byteStream, StandardCharsets.UTF_8);
    }
} catch (IOException e) {
    e.printStackTrace();
}
return result;
}

public static void main(String[] args) {
    CloseableHttpClient client = buildClient(TraceApacheHttpClientBuilder.create()).build();
    // endpoint,project_id和face_set_name需要替换成实际信息。
    String url = "https://{{endpoint}}/v2/{{project_id}}/face-sets/{{face_set_name}}";
    String token = "对应region的token";
    String result = doGet(url, token, client);
    System.out.println(result);
}
}
```

响应示例

状态码：200

成功响应样例

```
{
  "face_set_info": {
    "face_number": 94,
    "face_set_id": "T785tx1N",
    "face_set_name": "showFaceSet",
    "create_date": "2018-05-10 01:44:39",
    "face_set_capacity": 10000,
    "external_fields": {
      "number": {
        "type": "integer"
      },
      "id": {
        "type": "string"
      },
      "timestamp": {
        "type": "long"
      }
    }
  }
}
```

状态码：400

失败响应样例

```
{
  "error_code": "FRS.0202",
  "error_msg": "The service has been freeze."
}
```

状态码

状态码请参见[状态码](#)。

错误码

错误码请参见[错误码](#)。

4.5.4 删除人脸库

功能介绍

删除人脸库以及其中所有的人脸。人脸库数据为用户隐私数据，该数据无备份，删除时请谨慎操作。

前提条件：

请确保您已开通[人脸搜索服务](#)。

调试

您可以在[API Explorer](#)中调试该接口，支持自动认证鉴权。API Explorer可以自动生成SDK代码示例，并提供SDK代码示例调试功能。

URI

DELETE /v2/{project_id}/face-sets/{face_set_name}

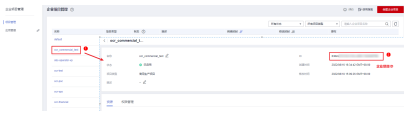
表 4-58 路径参数

参数	是否必选	参数类型	描述
project_id	是	String	项目ID，获取方法请参见 获取项目ID/账号名/AK/SK 。
face_set_name	是	String	人脸库名称。

请求参数

表 4-59 请求 Header 参数

参数	是否必选	参数类型	描述
X-Auth-Token	是	String	用户Token。 用于获取操作API的权限。获取方法请参见 认证鉴权 。
Content-Type	是	String	发送的实体的MIME类型，参数值为“application/json”。

参数	是否必选	参数类型	描述
Enterprise-Project-Id	否	String	<p>企业项目ID。FRS支持通过企业项目管理（EPS）对不同用户组和用户的资源使用，进行分账，当前仅支持按需计费模式。</p> <p>获取方法：进入“企业项目管理”页面，单击企业项目名称，在企业项目详情页获取Enterprise-Project-Id（企业项目ID）。</p>  <p>说明 创建企业项目后，在传参时，有以下三类场景。</p> <ul style="list-style-type: none"> 携带正确的ID，正常使用FRS服务，账单的企业项目会被分类到企业ID对应的企业项目中。 携带格式正确但不存在的ID，正常使用FRS服务，账单的企业项目会显示对应不存在的企业项目ID。 不携带ID或格式错误ID（包含特殊字符等），正常使用FRS服务，账单的企业项目会被分类到"default"中。

响应参数

状态码：200

表 4-60 响应 Body 参数

参数	参数类型	描述
face_set_name	String	人脸库名称。调用失败时无此字段。

状态码：400

表 4-61 响应 Body 参数

参数	参数类型	描述
error_code	String	调用失败时的错误码，具体请参考 错误码 。调用成功时无此字段。

参数	参数类型	描述
error_msg	String	调用失败时的错误信息。调用成功时无此字段。

请求示例

X-Auth-Token值获取方法请参见[快速入门](#)。

- 删除指定的人脸库

```
DELETE https://{endpoint}/v2/{project_id}/face-sets/showFaceSet
Request Header:
Content-Type: application/json
X-Auth-Token: MIINRwYJKoZlHvcNAQcColINODCCDTQCAQExDT...
```

- 使用Python3语言删除指定的人脸库

```
# -*- coding:utf-8 -*-

import requests
import base64

endpoint = '开通服务所在region的人脸识别服务域名'
project_id = '开通服务所在region的用户项目ID'
token = '用户获取得到的实际token值'
headers = {'Content-Type': 'application/json', 'X-Auth-Token': token}

face_set_name = 'showFaceSet'
url = "https://{endpoint}/v2/{project_id}/face-sets/{face_set_name}".format(endpoint=endpoint,project_id=project_id,face_set_name=face_set_name)
response = requests.delete(url, headers=headers, verify=False)
print(response.text)
```

- 使用Java语言删除指定的人脸库

```
import com.huawei.trace.http.apache.httpclient.TraceApacheHttpClientBuilder;
import org.apache.http.HttpEntity;
import org.apache.http.HttpHeaders;
import org.apache.http.HttpResponse;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.client.methods.HttpDelete;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.config.Registry;
import org.apache.http.config.RegistryBuilder;
import org.apache.http.conn.socket.ConnectionSocketFactory;
import org.apache.http.conn.socket.PlainConnectionSocketFactory;
import org.apache.http.conn.ssl.NoopHostnameVerifier;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;
import org.apache.http.ssl.SSLContextBuilder;
import org.apache.http.ssl.TrustStrategy;
import org.apache.http.util.EntityUtils;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.security.KeyManagementException;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.concurrent.TimeUnit;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.SSLContext;

/**
 * 此demo仅供测试使用，强烈建议使用SDK
 * 使用前需已配置HttpClient jar包。jar包可通过下载SDK获取
```

```
*/
public class DeleteFaceSetByFaceSetName {
    protected static HttpClientBuilder buildClient(HttpClientBuilder httpClientBuilder) {
        SSLContext sslContext = null;
        try {
            sslContext = new SSLContextBuilder().loadTrustMaterial(null, new TrustStrategy() {
                public boolean isTrusted(X509Certificate[] arg0, String arg1) throws CertificateException {
                    return true;
                }
            }).build();
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e.getMessage());
        } catch (KeyManagementException e) {
            throw new RuntimeException(e.getMessage());
        } catch (KeyStoreException e) {
            throw new RuntimeException(e.getMessage());
        }
        httpClientBuilder.setSSLContext(sslContext);
        httpClientBuilder.setConnectionTimeToLive(30, TimeUnit.SECONDS);
        HostnameVerifier hostnameVerifier = NoopHostnameVerifier.INSTANCE;
        SSLConnectionSocketFactory sslSocketFactory = new SSLConnectionSocketFactory(sslContext,
            hostnameVerifier);
        Registry<ConnectionSocketFactory> socketFactoryRegistry
            = RegistryBuilder.<ConnectionSocketFactory>create().register("http",
                PlainConnectionSocketFactory.getSocketFactory()).register("https", sslSocketFactory).build();
        PoolingHttpClientConnectionManager connMgr = new
        PoolingHttpClientConnectionManager(socketFactoryRegistry);
        connMgr.setMaxTotal(200);
        connMgr.setDefaultMaxPerRoute(100);
        httpClientBuilder.setConnectionManager(connMgr);
        return httpClientBuilder;
    }

    public static String doDelete(String url, String token, CloseableHttpClient client) {
        HttpDelete delete = new HttpDelete(url);
        delete.setHeader(HttpHeaders.CONTENT_TYPE, "application/json");
        //time unit is milliseconds
        RequestConfig requestConfig =
        RequestConfig.custom().setConnectTimeout(3000).setSocketTimeout(3000).build();
        delete.setConfig(requestConfig);
        delete.setHeader("X-Auth-Token", token);
        HttpResponse response = null;
        String result = "";
        try {
            response = client.execute(delete);
            HttpEntity responseBody = response.getEntity();
            if (responseBody == null) {
                System.out.println("the response body is null.");
                return result;
            } else {
                byte[] byteStream = EntityUtils.toByteArray(responseBody);
                System.arraycopy(byteStream, 0, new byte[byteStream.length], 0, byteStream.length);
                result = new String(byteStream, StandardCharsets.UTF_8);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return result;
    }

    public static void main(String[] args) {
        CloseableHttpClient client = buildClient(TraceApacheHttpClientBuilder.create()).build();
        // endpoint,project_id和face_set_name需要替换成实际信息。
        String url = "https://{{endpoint}}/v2/{{project_id}}/face-sets/{{face_set_name}}";
        String token = "对应region的token";
        String result = doDelete(url, token, client);
        System.out.println(result);
    }
}
```

```
}  
}
```

响应示例

状态码：200

成功响应样例

```
{  
  "face_set_name": "showFaceSet"  
}
```

状态码：400

失败响应样例

```
{  
  "error_code": "FRS.0002",  
  "error_msg": "The authentication token is abnormal."  
}
```

状态码

状态码请参见[状态码](#)。

错误码

错误码请参见[错误码](#)。

4.6 人脸资源管理

4.6.1 添加人脸

功能介绍

添加人脸到人脸库中。将单张图片中的人脸添加至人脸库中，支持添加最大人脸或所有人脸。

前提条件：

请确保您已开通[人脸搜索服务](#)。

约束限制：

- 只支持识别JPG、PNG、JPEG、BMP格式的图片。
- application/json请求的body中，请使用标准Json格式。
- Base64编码中请勿使用回车换行。
- 系统不保存用户图片。
- 图片大小小于8MB，由于过大图片会导致时延较长，并且图片信息量不大，建议小于1MB。
- 图片分辨率小于4096*2160，图片中人脸像素大于80*80，建议120*120以上。
- 为保证识别效果，人脸图片建议要求如下：
 - a. 光照大于200lux、无反光强光阴影现象。

- b. 人脸无遮挡、整体清晰无拖尾抖动等运动模糊。
- c. 侧脸不超过30°、俯仰角小于15°、偏转角小于15°、图片中人脸保持竖置正脸。
- 其他的约束限制信息请参见[约束与限制](#)章节。

建议:

- 由于过大图片对识别算法精度无明显提升，同时会导致时延较长，建议传入图片小于1MB，一般500KB左右足够。
- OBS上存储的图片也建议小于1MB。

调试

您可以在[API Explorer](#)中调试该接口，支持自动认证鉴权。API Explorer可以自动生成SDK代码示例，并提供SDK代码示例调试功能。

URI

POST /v2/{project_id}/face-sets/{face_set_name}/faces

表 4-62 路径参数

参数	是否必选	参数类型	描述
project_id	是	String	项目ID，获取方法请参见 获取项目ID/账号名/AK/SK 。
face_set_name	是	String	人脸库名称，字符串长度1-64。

请求参数

表 4-63 请求 Header 参数

参数	是否必选	参数类型	描述
X-Auth-Token	是	String	用户Token。 用于获取操作API的权限。获取方法请参见 认证鉴权 。
Content-Type	是	String	发送的实体的MIME类型，参数值为“application/json”。

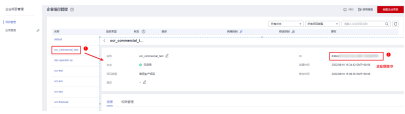
参数	是否必选	参数类型	描述
Enterprise-Project-Id	否	String	<p>企业项目ID。FRS支持通过企业项目管理（EPS）对不同用户组和用户的资源使用，进行分账，当前仅支持按需计费模式。</p> <p>获取方法：进入“企业项目管理”页面，单击企业名称，在企业项目详情页获取Enterprise-Project-Id（企业项目ID）。</p>  <p>说明 创建企业项目后，在传参时，有以下三类场景。</p> <ul style="list-style-type: none"> 携带正确的ID，正常使用FRS服务，账单的企业项目会被分类到企业ID对应的企业项目中。 携带格式正确但不存在的ID，正常使用FRS服务，账单的企业项目会显示对应不存在的企业项目ID。 不携带ID或格式错误ID（包含特殊字符等），正常使用FRS服务，账单的企业项目会被分类到"default"中。

表 4-64 请求 Body 参数

参数名	参数类型	是否必选	说明
image_url	String	与 image_file、image_base64三选一	图片的URL路径，目前仅支持华为云上OBS的URL，使用时只需保证FRS有权限读取该OBS桶的数据。开通读取权限的操作请参见 服务授权 。
image_file	File	与 image_url、image_base64三选一	本地图片文件，图片不能超过 8MB ，建议 小于1MB 。上传文件时，请求格式为multipart。
image_base64	String	与 image_file、image_url三选一	图像数据，Base64编码，要求： <ul style="list-style-type: none"> Base64编码后大小不超过8MB，建议小于1MB。 图片为JPG/JPEG/BMP/PNG格式。

参数名	参数类型	是否必选	说明
external_image_id	String	否	用户指定的图片外部ID，与当前图像绑定。用户没提供，系统会生成一个。 该ID长度范围为1~36位，可以包含字母、数字、中划线或者下划线，不包含其他的特殊字符。
external_fields	Object	否	根据用户自定义数据类型，填入相应的数值。 需在创建人脸库时定义external_fields字段，才可以在添加人脸时使用该字段，参考 自定义字段 。
single	boolean	否	是否将图片中的最大人脸添加至人脸库。可选值包括： <ul style="list-style-type: none"> • true: 传入的单张图片中如果包含多张人脸，则只将最大人脸添加到人脸库中。 • false: 默认为false。传入的单张图片中如果包含多张人脸，则将所有人脸添加至人脸库中。

响应参数

状态码：200

表 4-65 响应 Body 参数

参数	参数类型	描述
face_set_id	String	人脸库ID。调用失败时无此字段。
face_set_name	String	人脸库名称。调用失败时无此字段。
faces	Array of FaceSetFace objects	人脸库当中的人脸结构，详见 FaceSetFace 。调用失败时无此字段。

表 4-66 FaceSetFace

参数	参数类型	描述
bounding_box	BoundingBox object	人脸在图像中的位置。BoundingBox结构见 BoundingBox 。

参数	参数类型	描述
external_fields	Object	用户添加的额外字段。
external_image_id	String	人脸所在的外部图片ID。
face_id	String	人脸ID，由系统内部生成的唯一ID。

表 4-67 BoundingBox

参数	参数类型	描述
width	Integer	矩形框宽度。
top_left_y	Integer	矩形框左上角纵坐标。
top_left_x	Integer	矩形框左上角横坐标。
height	Integer	矩形框高度。

状态码： 400

表 4-68 响应 Body 参数

参数	参数类型	描述
error_code	String	调用失败时的错误码，具体请参考 错误码 。调用成功时无此字段。
error_msg	String	调用失败时的错误信息。调用成功时无此字段。

请求示例

X-Auth-Token值获取方法请参见[快速入门](#)。

- 向人脸库中添加人脸，人脸通过图片base64编码传入
 POST https://{endpoint}/v2/{project_id}/face-sets/showFaceSet/faces
 Request Header:
 Content-Type: application/json
 X-Auth-Token: MIINRwYJKoZihvcNAQcColINODCCDTQCAQExDT...
 Request Body:

```
{
  "image_base64": "/9j/4AAQSkZJRgABAgEASABIAAD",
  "external_image_id": "imageID",
  "external_fields": {
    "timestamp": 12,
    "id": "home"
  }
}
```

- 向人脸库中添加人脸，人脸通过图片文件传入

POST https://{endpoint}/v2/{project_id}/face-sets/showFaceSet/faces

Request Header:
X-Auth-Token: MIINRwYJKoZIhvcNAQcColINODCCDTQCAQExDT...

Request Body:
image_file: File(图片文件)
external_image_id: imageID
external_fields: {"timestamp": 12,"id": "home"}
- 向人脸库中添加人脸，人脸通过图片url传入

POST https://{endpoint}/v2/{project_id}/face-sets/showFaceSet/faces

Request Header:
Content-Type: application/json
X-Auth-Token: MIINRwYJKoZIhvcNAQcColINODCCDTQCAQExDT...

Request Body:
{
 "image_url":"https://<bucket-name>.<endpoint>/<object-name>",
 "external_image_id":"imageID",
 "external_fields": {
 "timestamp": 12,
 "id": "home"
 }
}
- 使用Python3语言向人脸库中添加人脸，人脸通过图片文件传入

```
# -*- coding:utf-8 -*-

import requests
import base64

endpoint = '开通服务所在region的人脸识别服务域名'
project_id = '开通服务所在region的用户项目ID'
token = '用户获取得到的实际token值'
headers = {'Content-Type': 'application/json', 'X-Auth-Token': token}

face_set_name = 'showFaceSet'
url = "https://{endpoint}/v2/{project_id}/face-sets/{face_set_name}/faces".format(
    endpoint=endpoint,project_id=project_id,face_set_name=face_set_name)
imagepath = r'./data/face-demo.png'
with open(imagepath, "rb") as bin_data:
    image_data = bin_data.read()
image_base64 = base64.b64encode(image_data).decode("utf-8")
body = {
    "image_base64": image_base64,
    "external_image_id": "imageID",
    "external_fields": {
        "timestamp": 12,
        "id": "home"
    }
}

response = requests.post(url, headers=headers, json=body, verify=False)
print(response.text)
```
- 使用Java语言向人脸库中添加人脸，人脸通过图片base64编码传入

```
import com.huawei.trace.http.apache.httpclient.TraceApacheHttpClientBuilder;
import org.apache.http.HttpEntity;
import org.apache.http.HttpHeaders;
import org.apache.http.HttpResponse;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.config.Registry;
import org.apache.http.config.RegistryBuilder;
import org.apache.http.conn.socket.ConnectionSocketFactory;
import org.apache.http.conn.socket.PlainConnectionSocketFactory;
import org.apache.http.conn.ssl.NoopHostnameVerifier;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import org.apache.http.entity.ContentType;
import org.apache.http.entity.StringEntity;
```

```
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;
import org.apache.http.ssl.SSLContextBuilder;
import org.apache.http.ssl.TrustStrategy;
import org.apache.http.util.EntityUtils;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.security.KeyManagementException;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.concurrent.TimeUnit;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.SSLContext;

/**
 * 此demo仅供测试使用，强烈建议使用SDK
 * 使用前需已配置HttpClient jar包。jar包可通过下载SDK获取
 */

public class AddFace {
    protected static HttpClientBuilder buildClient(HttpClientBuilder httpClientBuilder) {
        SSLContext sslContext = null;
        try {
            sslContext = new SSLContextBuilder().loadTrustMaterial(null, new TrustStrategy() {
                public boolean isTrusted(X509Certificate[] arg0, String arg1) throws CertificateException {
                    return true;
                }
            }).build();
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e.getMessage());
        } catch (KeyManagementException e) {
            throw new RuntimeException(e.getMessage());
        } catch (KeyStoreException e) {
            throw new RuntimeException(e.getMessage());
        }
        httpClientBuilder.setSSLContext(sslContext);
        httpClientBuilder.setConnectionTimeToLive(30, TimeUnit.SECONDS);
        HostnameVerifier hostnameVerifier = NoopHostnameVerifier.INSTANCE;
        SSLConnectionSocketFactory sslSocketFactory = new SSLConnectionSocketFactory(sslContext,
            hostnameVerifier);
        Registry<ConnectionSocketFactory> socketFactoryRegistry
            = RegistryBuilder.<ConnectionSocketFactory>create().register("http",
                PlainConnectionSocketFactory.getSocketFactory()).register("https", sslSocketFactory).build();
        PoolingHttpClientConnectionManager connMgr = new
        PoolingHttpClientConnectionManager(socketFactoryRegistry);
        connMgr.setMaxTotal(200);
        connMgr.setDefaultMaxPerRoute(100);
        httpClientBuilder.setConnectionManager(connMgr);
        return httpClientBuilder;
    }

    public static String doPost(String url, String jsonStr, String token, CloseableHttpClient client) {
        HttpPost post = new HttpPost(url);
        StringEntity entity = new StringEntity(jsonStr, ContentType.APPLICATION_JSON);
        post.setEntity(entity);
        post.setHeader(HttpHeaders.CONTENT_TYPE, "application/json");
        //time unit is milliseconds
        RequestConfig requestConfig =
        RequestConfig.custom().setConnectTimeout(3000).setSocketTimeout(3000).build();
        post.setConfig(requestConfig);
        post.setHeader("X-Auth-Token", token);
        HttpResponse response = null;
        String result = "";
        try {
            response = client.execute(post);
            HttpEntity responseBody = response.getEntity();
        }
    }
}
```

```

        if (responseBody == null) {
            System.out.println("the response body is null.");
            return result;
        } else {
            byte[] byteStream = EntityUtils.toByteArray(responseBody);
            System.arraycopy(byteStream, 0, new byte[byteStream.length], 0, byteStream.length);
            result = new String(byteStream, StandardCharsets.UTF_8);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    }
    return result;
}

public static void main(String[] args) {
    CloseableHttpClient client = buildClient(TraceApacheHttpClientBuilder.create()).build();
    // endpoint,project_id和face_set_name需要替换成实际信息。
    String url = "https://{{endpoint}}/v2/{{project_id}}/face-sets/{{face_set_name}}/faces";
    String jsonStr = "{\"image_base64\": \"9j/4AAQSkZJRgABAgEASABIAAD...\"}";
    String token = "对应region的token";
    String result = doPost(url, jsonStr, token, client);
    System.out.println(result);
}
}
}

```

响应示例

状态码：200

成功响应样例

```

{
  "face_set_id": "T785tx1N",
  "face_set_name": "showFaceSet",
  "faces": [
    {
      "bounding_box": {
        "width": 63,
        "top_left_y": 100,
        "top_left_x": 221,
        "height": 63
      },
      "external_image_id": "Xr0phyap",
      "external_fields": {
        "timestamp": 12,
        "id": "home"
      },
      "face_id": "JLa9hYLI"
    }
  ]
}

```

状态码：400

失败响应样例

```

{
  "error_code": "FRS.0404",
  "error_msg": "Detect no face, can not add it to face set."
}

```

状态码

状态码请参见[状态码](#)。

错误码

错误码请参见[错误码](#)。

4.6.2 查询人脸

功能介绍

查询指定人脸库中人脸信息。

前提条件：

请确保您已开通[人脸搜索服务](#)。

调试

您可以在[API Explorer](#)中调试该接口，支持自动认证鉴权。API Explorer可以自动生成SDK代码示例，并提供SDK代码示例调试功能。

URI

GET /v2/{project_id}/face-sets/{face_set_name}/faces?offset=xxx&limit=xxx

或者

GET /v2/{project_id}/face-sets/{face_set_name}/faces?face_id={face_id}

表 4-69 路径参数

参数	是否必选	参数类型	描述
project_id	是	String	项目ID，获取方法请参见 获取项目ID/账号名/AK/SK 。
face_set_name	是	String	人脸库名称。

表 4-70 Query 参数

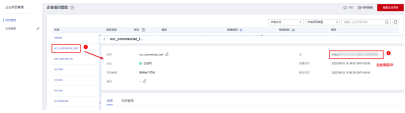
参数	是否必选	参数类型	描述
offset	否，与face_id二选一	Integer	查询的起始位置，取值范围为非负整数，默认为0。
limit	否，与face_id二选一	Integer	单次查询总量，默认为5。offset+limit值不能超过10000。
face_id	否，与offset二选一	String	人脸ID。

📖 说明

offset+limit值不能超过10000。人脸识别系统非存储系统，暂时不支持数据遍历操作，用户可以自行保存face_id，根据face_id查询导入的人脸信息。

请求参数

表 4-71 请求 Header 参数

参数	是否必选	参数类型	描述
X-Auth-Token	是	String	用户Token。 用于获取操作API的权限。获取方法请参见 认证鉴权 。
Content-Type	是	String	发送的实体的MIME类型，参数值为“application/json”。
Enterprise-Project-Id	否	String	<p>企业项目ID。FRS支持通过企业项目管理（EPS）对不同用户组和用户的资源使用，进行分账，当前仅支持按需计费模式。</p> <p>获取方法：进入“企业项目管理”页面，单击企业名称，在企业项目详情页获取Enterprise-Project-Id（企业项目ID）。</p>  <p>说明 创建企业项目后，在传参时，有以下三类场景。</p> <ul style="list-style-type: none"> 携带正确的ID，正常使用FRS服务，账单的企业项目会被分类到企业ID对应的企业项目中。 携带格式正确但不存在的ID，正常使用FRS服务，账单的企业项目会显示对应不存在的企业项目ID。 不携带ID或格式错误ID（包含特殊字符等），正常使用FRS服务，账单的企业项目会被分类到"default"中。

响应参数

状态码：200

表 4-72 响应 Body 参数

参数	参数类型	描述
face_set_id	String	人脸库ID，随机生成的包含八个字符的字符串。调用失败时无此字段。

参数	参数类型	描述
face_set_name	String	人脸库名称。调用失败时无此字段。
faces	Array of FaceSetFace objects	人脸库当中的人脸结构，详见 FaceSetFace 。调用失败时无此字段。

表 4-73 FaceSetFace

参数	参数类型	描述
bounding_box	BoundingBox object	人脸在图像中的位置。BoundingBox结构见 BoundingBox 。
external_fields	Object	用户添加的额外字段。
external_image_id	String	人脸所在的外部图片ID。
face_id	String	人脸ID，由系统内部生成的唯一ID。

表 4-74 BoundingBox

参数	参数类型	描述
width	Integer	矩形框宽度。
top_left_y	Integer	矩形框左上角纵坐标。
top_left_x	Integer	矩形框左上角横坐标。
height	Integer	矩形框高度。

状态码： 400

表 4-75 响应 Body 参数

参数	参数类型	描述
error_code	String	调用失败时的错误码，具体请参考 错误码 。调用成功时无此字段。
error_msg	String	调用失败时的错误信息。调用成功时无此字段。

请求示例

X-Auth-Token值获取方法请参见[快速入门](#)。

- 在人脸库中查询某一人脸特征信息

```
GET https://{endpoint}/v2/{project_id}/face-sets/showFaceSet/faces?offset=0&limit=1
Request Header:
Content-Type: application/json
X-Auth-Token: MIINRwYJKoZIhvcNAQcCollINODCCDTQCAQExDT...
```

- 使用Python3语言在人脸库中查询某一人脸特征信息

```
# -*- coding:utf-8 -*-

import requests
import base64

endpoint = '开通服务所在region的人脸识别服务域名'
project_id = '开通服务所在region的用户项目ID'
token = '用户获取得到的实际token值'
headers = {'Content-Type': 'application/json', 'X-Auth-Token': token}

face_set_name = 'showFaceSet'
url = "https://{endpoint}/v2/{project_id}/face-sets/{face_set_name}/faces?offset=0&limit=1".format(
    endpoint=endpoint, project_id=project_id, face_set_name=face_set_name)
response = requests.get(url, headers=headers, verify=False)
print(response.text)
```

- 使用Java语言在人脸库中查询某一人脸特征信息

```
import com.huawei.trace.http.apache.httpclient.TraceApacheHttpClientBuilder;
import org.apache.http.HttpEntity;
import org.apache.http.HttpHeaders;
import org.apache.http.HttpResponse;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.config.Registry;
import org.apache.http.config.RegistryBuilder;
import org.apache.http.conn.socket.ConnectionSocketFactory;
import org.apache.http.conn.socket.PlainConnectionSocketFactory;
import org.apache.http.conn.ssl.NoopHostnameVerifier;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;
import org.apache.http.ssl.SSLContextBuilder;
import org.apache.http.ssl.TrustStrategy;
import org.apache.http.util.EntityUtils;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.security.KeyManagementException;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.concurrent.TimeUnit;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.SSLContext;

/**
 * 此demo仅供测试使用，强烈建议使用SDK
 * 使用前需已配置HttpClient jar包。jar包可通过下载SDK获取
 */

public class GetFaceByFaceId {
    protected static HttpClientBuilder buildClient(HttpClientBuilder httpClientBuilder) {
        SSLContext sslContext = null;
        try {
            sslContext = new SSLContextBuilder().loadTrustMaterial(null, new TrustStrategy() {
                public boolean isTrusted(X509Certificate[] arg0, String arg1) throws CertificateException {
                    return true;
                }
            });
        } catch (KeyManagementException | KeyStoreException | NoSuchAlgorithmException | CertificateException | IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
    }
    }).build();
} catch (NoSuchAlgorithmException e) {
    throw new RuntimeException(e.getMessage());
} catch (KeyManagementException e) {
    throw new RuntimeException(e.getMessage());
} catch (KeyStoreException e) {
    throw new RuntimeException(e.getMessage());
}
httpClientBuilder.setSSLContext(sslContext);
httpClientBuilder.setConnectionTimeToLive(30, TimeUnit.SECONDS);
HostnameVerifier hostnameVerifier = NoopHostnameVerifier.INSTANCE;
SSLConnectionSocketFactory sslSocketFactory = new SSLConnectionSocketFactory(sslContext,
hostnameVerifier);
Registry<ConnectionSocketFactory> socketFactoryRegistry
    = RegistryBuilder.<ConnectionSocketFactory>create().register("http",
    PlainConnectionSocketFactory.getSocketFactory()).register("https", sslSocketFactory).build();
PoolingHttpClientConnectionManager connMgr = new
PoolingHttpClientConnectionManager(socketFactoryRegistry);
connMgr.setMaxTotal(200);
connMgr.setDefaultMaxPerRoute(100);
httpClientBuilder.setConnectionManager(connMgr);
return httpClientBuilder;
}

public static String doGet(String url, String token, CloseableHttpClient client) {
    HttpGet get = new HttpGet(url);
    get.setHeader(HttpHeaders.CONTENT_TYPE, "application/json");
    //time unit is milliseconds
    RequestConfig requestConfig =
RequestConfig.custom().setConnectTimeout(3000).setSocketTimeout(3000).build();
    get.setConfig(requestConfig);
    get.setHeader("X-Auth-Token", token);
    HttpResponse response = null;
    String result = "";
    try {
        response = client.execute(get);
        HttpEntity responseBody = response.getEntity();
        if (responseBody == null) {
            System.out.println("the response body is null.");
            return result;
        } else {
            byte[] byteStream = EntityUtils.toByteArray(responseBody);
            System.arraycopy(byteStream, 0, new byte[byteStream.length], 0, byteStream.length);
            result = new String(byteStream, StandardCharsets.UTF_8);
        }
    }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return result;
}

public static void main(String[] args) {
    CloseableHttpClient client = buildClient(TraceApacheHttpClientBuilder.create()).build();
    // endpoint,project_id,face_set_name和face_id需要替换成实际信息。
    String url = "https://{{endpoint}}/v2/{{project_id}}/face-sets/{{face_set_name}}/faces?
face_id={{face_id}}";
    String token = "对应region的token";
    String result = doGet(url, token, client);
    System.out.println(result);
}
}
```

响应示例

状态码：200

成功响应样例

```
{
  "face_set_id": "T785tx1N",
  "face_set_name": "showFaceSet",
  "faces": [
    {
      "bounding_box": {
        "width": 63,
        "top_left_y": 100,
        "top_left_x": 221,
        "height": 63
      },
      "external_image_id": "alzRAa58",
      "face_id": "cFydu4d2",
      "external_fields": {
        "number": 122,
        "id": "home",
        "timestamp": 12
      }
    }
  ]
}
```

状态码：400

失败响应样例

```
{
  "error_code": "FRS.0002",
  "error_msg": "The authentication token is abnormal."
}
```

状态码

状态码请参见[状态码](#)。

错误码

错误码请参见[错误码](#)。

4.6.3 更新人脸

功能介绍

根据人脸ID (face_id) 更新单张人脸信息。

前提条件：

请确保您已开通[人脸搜索服务](#)。

说明

application/json请求的body中，请使用标准Json格式。

调试

您可以在[API Explorer](#)中调试该接口，支持自动认证鉴权。API Explorer可以自动生成SDK代码示例，并提供SDK代码示例调试功能。

URI

PUT /v2/{project_id}/face-sets/{face_set_name}/faces

表 4-76 路径参数

参数	是否必选	参数类型	描述
project_id	是	String	项目ID，获取方法请参见 获取项目ID/账号名/AK/SK 。
face_set_name	是	String	人脸库名称。

请求参数

表 4-77 请求 Header 参数


参数	是否必选	参数类型	描述
X-Auth-Token	是	String	用户Token。 用于获取操作API的权限。获取方法请参见 认证鉴权 。
Content-Type	是	String	发送的实体的MIME类型，参数值为“application/json”。
Enterprise-Project-Id	否	String	<p>企业项目ID。FRS支持通过企业项目管理（EPS）对不同用户组和用户的资源使用，进行分账，当前仅支持按需计费模式。</p> <p>获取方法：进入“企业项目管理”页面，单击企业名称，在企业项目详情页获取Enterprise-Project-Id（企业项目ID）。</p>  <p>说明 创建企业项目后，在传参时，有以下三类场景。</p> <ul style="list-style-type: none"> 携带正确的ID，正常使用FRS服务，账单的企业项目会被分类到企业ID对应的企业项目中。 携带格式正确但不存在的ID，正常使用FRS服务，账单的企业项目会显示对应不存在的企业项目ID。 携带正确的ID，正常使用FRS服务，账单的企业项目会被分类到企业ID对应的企业项目中。

表 4-78 请求 Body 参数

参数名	参数类型	是否必选	说明
face_id	String	是	人脸ID，由系统内部生成的唯一ID。
external_image_id	String	否	用户指定的图片外部ID，与当前图像绑定。用户不设置该参数时，系统会自动生成一个。该ID长度范围为[1,36]，可以包含字母、数字、中划线或者下划线，不包含其他的特殊字符。 这里是待修改的参数，external_image_id和external_fields至少选一个。
external_fields	Object	否	自定义字段的key值长度范围为[1,36]，string类型的value长度范围为[1,256]，具体参见 自定义字段 。 这里是待修改的参数，external_image_id和external_fields至少选一个。

响应参数

状态码： 200

表 4-79 响应 Body 参数

参数	参数类型	描述
face_number	Integer	更新的人脸数量。调用失败时无此字段。
face_set_id	String	人脸库ID。调用失败时无此字段。
face_set_name	String	人脸库名称。调用失败时无此字段。

状态码： 400

表 4-80 响应 Body 参数

参数	参数类型	描述
error_code	String	调用失败时的错误码，具体请参考 错误码 。调用成功时无此字段。
error_msg	String	调用失败时的错误信息。调用成功时无此字段。

请求示例

X-Auth-Token值获取方法请参见[快速入门](#)。

- 更新人脸库中某一人脸的特征信息

```
PUT https://{endpoint}/v2/{project_id}/face-sets/showFaceSet/faces
Request Header:
Content-Type: application/json
X-Auth-Token: MIINRwYJKoZihvcNAQcColINODCCDTQCAQExDT...

Request Body:
{
  "face_id": "iexEBb6t",
  "external_image_id": "imageID",
  "external_fields": {
    "timestamp": 12,
    "id": "300018629384756"
  }
}
```

- 使用Python3语言更新人脸库中某一人脸的特征信息

```
# -*- coding:utf-8 -*-

import requests
import base64

endpoint = '开通服务所在region的人脸识别服务域名'
project_id = '开通服务所在region的用户项目ID'
token = '用户获取得到的实际token值'
headers = {'Content-Type': 'application/json', 'X-Auth-Token': token}

face_set_name = 'showFaceSet'
url = "https://{endpoint}/v2/{project_id}/face-sets/{face_set_name}/faces".format(
    endpoint=endpoint, project_id=project_id, face_set_name=face_set_name)
body = {
  "face_id": "iexEBb6t",
  "external_image_id": "imageID",
  "external_fields": {
    "timestamp": 12,
    "id": "300018629384756"
  }
}
response = requests.put(url, headers=headers, json=body, verify=False)
print(response.text)
```

- 使用Java语言更新人脸库中某一人脸的特征信息

```
import com.huawei.trace.http.apache.httpclient.TraceApacheHttpClientBuilder;
import org.apache.http.apache.http.HttpEntity;
import org.apache.http.HttpHeaders;
import org.apache.http.HttpResponse;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.client.methods.HttpPut;
import org.apache.http.config.Registry;
import org.apache.http.config.RegistryBuilder;
import org.apache.http.conn.socket.ConnectionSocketFactory;
import org.apache.http.conn.socket.PlainConnectionSocketFactory;
import org.apache.http.conn.ssl.NoopHostnameVerifier;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import org.apache.http.entity.ContentType;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;
import org.apache.http.ssl.SSLContextBuilder;
import org.apache.http.ssl.TrustStrategy;
import org.apache.http.util.EntityUtils;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.security.KeyManagementException;
```

```
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.concurrent.TimeUnit;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.SSLContext;

/**
 * 此demo仅供测试使用，强烈建议使用SDK
 * 使用前需已配置HttpClient jar包。jar包可通过下载SDK获取
 */

public class UpdateFaceByFaceId {
    protected static HttpClientBuilder buildClient(HttpClientBuilder httpClientBuilder) {
        SSLContext sslContext = null;
        try {
            sslContext = new SSLContextBuilder().loadTrustMaterial(null, new TrustStrategy() {
                public boolean isTrusted(X509Certificate[] arg0, String arg1) throws CertificateException {
                    return true;
                }
            }).build();
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e.getMessage());
        } catch (KeyManagementException e) {
            throw new RuntimeException(e.getMessage());
        } catch (KeyStoreException e) {
            throw new RuntimeException(e.getMessage());
        }
        httpClientBuilder.setSSLContext(sslContext);
        httpClientBuilder.setConnectionTimeToLive(30, TimeUnit.SECONDS);
        HostnameVerifier hostnameVerifier = NoopHostnameVerifier.INSTANCE;
        SSLConnectionSocketFactory sslSocketFactory = new SSLConnectionSocketFactory(sslContext,
        hostnameVerifier);
        Registry<ConnectionSocketFactory> socketFactoryRegistry
            = RegistryBuilder.<ConnectionSocketFactory>create().register("http",
            PlainConnectionSocketFactory.getSocketFactory()).register("https", sslSocketFactory).build();
        PoolingHttpClientConnectionManager connMgr = new
        PoolingHttpClientConnectionManager(socketFactoryRegistry);
        connMgr.setMaxTotal(200);
        connMgr.setDefaultMaxPerRoute(100);
        httpClientBuilder.setConnectionManager(connMgr);
        return httpClientBuilder;
    }

    public static String doPut(String url, String jsonStr, String token, CloseableHttpClient client) {
        HttpPut put = new HttpPut(url);
        put.setHeader(HttpHeaders.CONTENT_TYPE, "application/json");
        StringEntity entity = new StringEntity(jsonStr, ContentType.APPLICATION_JSON);
        put.setEntity(entity);
        put.setHeader(HttpHeaders.CONTENT_TYPE, "application/json");
        //time unit is milliseconds
        RequestConfig requestConfig =
        RequestConfig.custom().setConnectTimeout(3000).setSocketTimeout(3000).build();
        put.setConfig(requestConfig);
        put.setHeader("X-Auth-Token", token);
        HttpResponse response = null;
        String result = "";
        try {
            response = client.execute(put);
            HttpEntity responseBody = response.getEntity();
            if (responseBody == null) {
                System.out.println("the response body is null.");
                return result;
            } else {
                byte[] byteStream = EntityUtils.toByteArray(responseBody);
                System.arraycopy(byteStream, 0, new byte[byteStream.length], 0, byteStream.length);
                result = new String(byteStream, StandardCharsets.UTF_8);
            }
        }
    }
}
```



```
    } catch (IOException e) {
        e.printStackTrace();
    }
    return result;
}

public static void main(String[] args) {
    CloseableHttpClient client = buildClient(TraceApacheHttpClientBuilder.create()).build();
    // endpoint,project_id和face_set_name需要替换成实际信息。
    String url = "https://{{endpoint}}/v2/{{project_id}}/face-sets/{{face_set_name}}/faces";
    String token = "对应region的token";
    String jsonStr = "{ \"face_id\": \"iexEBb6t\", \"external_image_id\": \"imageID\", \"external_fields\": { \"timestamp\": 12, \"id\": \"300018629384756\" }}";
    String result = doPut(url, jsonStr,token, client);
    System.out.println(result);
}
}
```

响应示例

状态码：200

成功响应样例

```
{
  "face_number": 1,
  "face_set_id": "T785tx1N",
  "face_set_name": "showFaceSet"
}
```

状态码：400

失败响应样例

```
{
  "error_code": "FRS.0303",
  "error_msg": "The face id is not exist, checkout your input."
}
```

状态码

状态码请参见[状态码](#)。

错误码

错误码请参见[错误码](#)。

4.6.4 删除人脸

功能介绍

根据指定字段删除人脸库中人脸，删除后人脸库容量会相应的释放。

前提条件：

请确保您已开通[人脸搜索服务](#)。

调试

您可以在[API Explorer](#)中调试该接口，支持自动认证鉴权。API Explorer可以自动生成SDK代码示例，并提供SDK代码示例调试功能。

URI

DELETE /v2/{project_id}/face-sets/{face_set_name}/faces?field_name=field_value

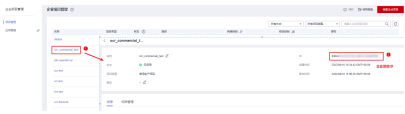
表 4-81 路径参数

参数	是否必选	参数类型	描述
project_id	是	String	项目ID，获取方法请参见 获取项目ID/账号名/AK/SK 。
face_set_name	是	String	人脸库名称。
field_name	是	String	按条件删除的字段名，支持固定字段（external_image_id和face_id），以及用户的 自定义字段 （不支持空字符串和null值删除）。

请求参数

表 4-82 请求 Header 参数

参数	是否必选	参数类型	描述
X-Auth-Token	是	String	用户Token。 用于获取操作API的权限。获取方法请参见 认证鉴权 。
Content-Type	是	String	发送的实体的MIME类型，参数值为“application/json”。

参数	是否必选	参数类型	描述
Enterprise-Project-Id	否	String	<p>企业项目ID。FRS支持通过企业项目管理（EPS）对不同用户组和用户的资源使用，进行分账，当前仅支持按需计费模式。</p> <p>获取方法：进入“企业项目管理”页面，单击企业名称，在企业项目详情页获取Enterprise-Project-Id（企业项目ID）。</p>  <p>说明 创建企业项目后，在传参时，有以下三类场景。</p> <ul style="list-style-type: none"> 携带正确的ID，正常使用FRS服务，账单的企业项目会被分类到企业ID对应的企业项目中。 携带格式正确但不存在的ID，正常使用FRS服务，账单的企业项目会显示对应不存在的企业项目ID。 不携带ID或格式错误ID（包含特殊字符等），正常使用FRS服务，账单的企业项目会被分类到"default"中。

响应参数

状态码：200

表 4-83 响应 Body 参数

参数	参数类型	描述
face_number	Integer	删除的人脸数量。调用失败时无此字段。
face_set_id	String	人脸库ID。调用失败时无此字段。
face_set_name	String	人脸库名称。调用失败时无此字段。

状态码：400

表 4-84 响应 Body 参数

参数	参数类型	描述
error_code	String	调用失败时的错误码，具体请参考 错误码 。调用成功时无此字段。
error_msg	String	调用失败时的错误信息。调用成功时无此字段。

请求示例

X-Auth-Token值获取方法请参见[快速入门](#)。

- 使用external_image_id（人脸所在的外部图片ID）删除人脸库中的某一人脸特征

DELETE https://{endpoint}/v2/{project_id}/face-sets/showFaceSet/faces?external_image_id=imageID

Request Header:

Content-Type: application/json

X-Auth-Token: MIINRwYJKoZihvcNAQcCoIINODCCDTQCAQExDT...
- 使用face_id（人脸ID）删除人脸库中的某一人脸特征

DELETE https://{endpoint}/v2/{project_id}/face-sets/showFaceSet/faces?face_id=faceID

Request Header:

Content-Type: application/json

X-Auth-Token: MIINRwYJKoZihvcNAQcCoIINODCCDTQCAQExDT...
- 使用用户自定义的人脸特征标识字段，删除人脸库中的某一人脸特征

DELETE https://{endpoint}/v2/{project_id}/face-sets/showFaceSet/faces?id=home

Request Header:

Content-Type: application/json

X-Auth-Token: MIINRwYJKoZihvcNAQcCoIINODCCDTQCAQExDT...
- 使用Python3语言通过face_id（人脸ID）删除人脸库中的某一人脸特征

```
# -*- coding:utf-8 -*-

import requests
import base64

endpoint = '开通服务所在region的人脸识别服务域名'
project_id = '开通服务所在region的用户项目ID'
token = '用户获取得到的实际token值'
headers = {'Content-Type': 'application/json', 'X-Auth-Token': token}

face_set_name = 'showFaceSet'
face_id = 'T785tx1N'
url = "https://{endpoint}/v2/{project_id}/face-sets/{face_set_name}/faces?id={face_id}".format(
    endpoint=endpoint, project_id=project_id, face_set_name=face_set_name, face_id=face_id)
response = requests.delete(url, headers=headers, verify=False)
print(response.text)
```
- 使用Java语言通过face_id（人脸ID）删除人脸库中的某一人脸特征

```
import com.huawei.trace.http.apache.httpclient.TraceApacheHttpClientBuilder;
import org.apache.http.HttpEntity;
import org.apache.http.HttpHeaders;
import org.apache.http.HttpResponse;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.client.methods.HttpDelete;
import org.apache.http.config.Registry;
import org.apache.http.config.RegistryBuilder;
import org.apache.http.conn.socket.ConnectionSocketFactory;
import org.apache.http.conn.socket.PlainConnectionSocketFactory;
import org.apache.http.conn.ssl.NoopHostnameVerifier;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;
import org.apache.http.ssl.SSLContextBuilder;
```

```
import org.apache.http.ssl.TrustStrategy;
import org.apache.http.util.EntityUtils;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.security.KeyManagementException;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.concurrent.TimeUnit;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.SSLContext;

/**
 * 此demo仅供测试使用，强烈建议使用SDK
 * 使用前需已配置HttpClient jar包。jar包可通过下载SDK获取
 */

public class DeleteFaceByFacelId {
    protected static HttpClientBuilder buildClient(HttpClientBuilder httpClientBuilder) {
        SSLContext sslContext = null;
        try {
            sslContext = new SSLContextBuilder().loadTrustMaterial(null, new TrustStrategy() {
                public boolean isTrusted(X509Certificate[] arg0, String arg1) throws CertificateException {
                    return true;
                }
            }).build();
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e.getMessage());
        } catch (KeyManagementException e) {
            throw new RuntimeException(e.getMessage());
        } catch (KeyStoreException e) {
            throw new RuntimeException(e.getMessage());
        }
        httpClientBuilder.setSSLContext(sslContext);
        httpClientBuilder.setConnectionTimeToLive(30, TimeUnit.SECONDS);
        HostnameVerifier hostnameVerifier = NoopHostnameVerifier.INSTANCE;
        SSLConnectionSocketFactory sslSocketFactory = new SSLConnectionSocketFactory(sslContext,
hostnameVerifier);
        Registry<ConnectionSocketFactory> socketFactoryRegistry
            = RegistryBuilder.<ConnectionSocketFactory>create().register("http",
                PlainConnectionSocketFactory.getSocketFactory()).register("https", sslSocketFactory).build();
        PoolingHttpClientConnectionManager connMgr = new
PoolingHttpClientConnectionManager(socketFactoryRegistry);
        connMgr.setMaxTotal(200);
        connMgr.setDefaultMaxPerRoute(100);
        httpClientBuilder.setConnectionManager(connMgr);
        return httpClientBuilder;
    }

    public static String doDelete(String url, String token, CloseableHttpClient client) {
        HttpDelete delete = new HttpDelete(url);
        delete.setHeader(HttpHeaders.CONTENT_TYPE, "application/json");
        //time unit is milliseconds
        RequestConfig requestConfig =
RequestConfig.custom().setConnectTimeout(3000).setSocketTimeout(3000).build();
        delete.setConfig(requestConfig);
        delete.setHeader("X-Auth-Token", token);
        HttpResponse response = null;
        String result = "";
        try {
            response = client.execute(delete);
            HttpEntity responseBody = response.getEntity();
            if (responseBody == null) {
                System.out.println("the response body is null.");
                return result;
            } else {
                byte[] byteStream = EntityUtils.toByteArray(responseBody);
                System.arraycopy(byteStream, 0, new byte[byteStream.length], 0, byteStream.length);
            }
        }
    }
}
```

```

        result = new String(byteStream, StandardCharsets.UTF_8);
    }
} catch (IOException e) {
    e.printStackTrace();
}
return result;
}

public static void main(String[] args) {
    CloseableHttpClient client = buildClient(TraceApacheHttpClientBuilder.create()).build();
    // endpoint,project_id,face_set_name和face_id需要替换成实际信息。
    String url = "https://{{endpoint}}/v2/{{project_id}}/face-sets/{{face_set_name}}/faces?
face_id={{face_id}}";
    String token = "对应region的token";
    String result = doDelete(url, token, client);
    System.out.println(result);
}
}
}

```

响应示例

状态码：200

成功响应样例

```

{
  "face_number": 1,
  "face_set_id": "T785tx1N",
  "face_set_name": "showFaceSet"
}

```

状态码：400

失败响应样例

```

{
  "error_code": "FRS.0402",
  "error_msg": "External id is not exist, can not delete face"
}

```

4.6.5 批量删除人脸

功能介绍

自定义筛选条件，批量删除人脸库中的符合指定条件的多张人脸。

前提条件：

请确保您已开通[人脸搜索服务](#)。

调试

您可以在[API Explorer](#)中调试该接口，支持自动认证鉴权。API Explorer可以自动生成 SDK代码示例，并提供SDK代码示例调试功能。

URI

DELETE /v2/{project_id}/face-sets/{face_set_name}/faces/batch

表 4-85 路径参数

参数	是否必选	参数类型	描述
project_id	是	String	项目ID，获取方法请参见 获取项目ID/账号名/AK/SK 。
face_set_name	是	String	人脸库名称。

请求参数

表 4-86 请求 Header 参数


参数	是否必选	参数类型	描述
X-Auth-Token	是	String	用户Token。 用于获取操作API的权限。获取方法请参见 认证鉴权 。
Content-Type	是	String	发送的实体的MIME类型，参数值为“application/json”。
Enterprise-Project-Id	否	String	<p>企业项目ID。FRS支持通过企业项目管理（EPS）对不同用户组和用户的资源使用，进行分账，当前仅支持按需计费模式。</p> <p>获取方法：进入“企业项目管理”页面，单击企业名称，在企业项目详情页获取Enterprise-Project-Id（企业项目ID）。</p>  <p>说明 创建企业项目后，在传参时，有以下三类场景。</p> <ul style="list-style-type: none"> 携带正确的ID，正常使用FRS服务，账单的企业项目会被分类到企业ID对应的企业项目中。 携带格式正确但不存在的ID，正常使用FRS服务，账单的企业项目会显示对应不存在的企业项目ID。 不携带ID或格式错误ID（包含特殊字符等），正常使用FRS服务，账单的企业项目会被分类到"default"中。

表 4-87 请求 Body 参数

参数名	参数类型	是否必选	说明
filter	String	是	过滤条件，具体参见 filter语法 。

响应参数

状态码：200

表 4-88 响应 Body 参数

参数	参数类型	描述
face_number	Integer	删除的人脸数量。调用失败时无此字段。
face_set_id	String	人脸库ID。调用失败时无此字段。
face_set_name	String	人脸库名称。调用失败时无此字段。

状态码：400

表 4-89 响应 Body 参数

参数	参数类型	描述
error_code	String	调用失败时的错误码，具体请参考 错误码 。调用成功时无此字段。
error_msg	String	调用失败时的错误信息。调用成功时无此字段。

请求示例

X-Auth-Token值获取方法请参见[快速入门](#)。

- 通过预设的filter筛选条件，批量删除人脸库中的多个人脸特征
 DELETE https://{endpoint}/v2/{project_id}/face-sets/showFaceSet/faces/batch
 Request Header:
 Content-Type: application/json
 X-Auth-Token: MIINRwYJKoZlhvcNAQcColINODCCDTQCAQExDT...
 Request Body:

```
{
  "filter" : "age:[20 TO 30]"
}
```
- 使用Python3语言通过预设的filter筛选条件，批量删除人脸库中的多个人脸特征

```
# -*- coding:utf-8 -*-

import requests
import base64

endpoint = '开通服务所在region的人脸识别服务域名'
```



```
project_id = '开通服务所在region的用户项目ID'
token = '用户获取得到的实际token值'
headers = {'Content-Type': 'application/json', 'X-Auth-Token': token}

face_set_name = 'showFaceSet'
url = "https://{endpoint}/v2/{project_id}/face-sets/{face_set_name}/faces/batch".format(
    endpoint=endpoint, project_id=project_id, face_set_name=face_set_name)
body = {"filter" : "age:[20 TO 30]"}
response = requests.delete(url, headers=headers, json=body, verify=False)
print(response.text)
```

- 使用Java语言通过预设的filter筛选条件，批量删除人脸库中的多个人脸特征

```
import com.huawei.trace.http.apache.httpclient.TraceApacheHttpClientBuilder;
import org.apache.http.HttpEntity;
import org.apache.http.HttpHeaders;
import org.apache.http.HttpResponse;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.client.methods.HttpEntityEnclosingRequestBase;
import org.apache.http.config.Registry;
import org.apache.http.config.RegistryBuilder;
import org.apache.http.conn.socket.ConnectionSocketFactory;
import org.apache.http.conn.socket.PlainConnectionSocketFactory;
import org.apache.http.conn.ssl.NoopHostnameVerifier;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import org.apache.http.entity.ContentType;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;
import org.apache.http.ssl.SSLContextBuilder;
import org.apache.http.ssl.TrustStrategy;
import org.apache.http.util.EntityUtils;
import java.io.IOException;
import java.net.URI;
import java.nio.charset.StandardCharsets;
import java.security.KeyManagementException;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.concurrent.TimeUnit;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.SSLContext;

/**
 * 此demo仅供测试使用，强烈建议使用SDK
 * 使用前需已配置HttpClient jar包。jar包可通过下载SDK获取
 */

public class DeleteFaceBath {
    protected static HttpClientBuilder buildClient(HttpClientBuilder httpClientBuilder) {
        SSLContext sslContext = null;
        try {
            sslContext = new SSLContextBuilder().loadTrustMaterial(null, new TrustStrategy() {
                public boolean isTrusted(X509Certificate[] arg0, String arg1) throws CertificateException {
                    return true;
                }
            }).build();
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e.getMessage());
        } catch (KeyManagementException e) {
            throw new RuntimeException(e.getMessage());
        } catch (KeyStoreException e) {
            throw new RuntimeException(e.getMessage());
        }
        httpClientBuilder.setSSLContext(sslContext);
        httpClientBuilder.setConnectionTimeToLive(30, TimeUnit.SECONDS);
        HostnameVerifier hostnameVerifier = NoopHostnameVerifier.INSTANCE;
        SSLConnectionSocketFactory sslSocketFactory = new SSLConnectionSocketFactory(sslContext,
            hostnameVerifier);
```

```

Registry<ConnectionSocketFactory> socketFactoryRegistry
    = RegistryBuilder.<ConnectionSocketFactory>create().register("http",
        PlainConnectionSocketFactory.getSocketFactory()).register("https", sslSocketFactory).build();
PoolingHttpClientConnectionManager connMgr = new
PoolingHttpClientConnectionManager(socketFactoryRegistry);
connMgr.setMaxTotal(200);
connMgr.setDefaultMaxPerRoute(100);
httpClientBuilder.setConnectionManager(connMgr);
return httpClientBuilder;
}

public static String doDelete(String url, String jsonStr, String token, CloseableHttpClient client) {
    HttpDeleteWithBody delete = new HttpDeleteWithBody(url);
    delete.setHeader(HttpHeaders.CONTENT_TYPE, "application/json");
    StringEntity entity = new StringEntity(jsonStr, ContentType.APPLICATION_JSON);
    delete.setEntity(entity);
    delete.setHeader(HttpHeaders.CONTENT_TYPE, "application/json");
    //time unit is milliseconds
    RequestConfig requestConfig =
RequestConfig.custom().setConnectTimeout(3000).setSocketTimeout(3000).build();
    delete.setConfig(requestConfig);
    delete.setHeader("X-Auth-Token", token);
    HttpResponse response = null;
    String result = "";
    try {
        response = client.execute(delete);
        HttpEntity responseBody = response.getEntity();
        if (responseBody == null) {
            System.out.println("the response body is null.");
            return result;
        } else {
            byte[] byteStream = EntityUtils.toByteArray(responseBody);
            System.arraycopy(byteStream, 0, new byte[byteStream.length], 0, byteStream.length);
            result = new String(byteStream, StandardCharsets.UTF_8);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return result;
}

public static void main(String[] args) {
    CloseableHttpClient client = buildClient(TraceApacheHttpClientBuilder.create()).build();
    // endpoint,project_id和face_set_name需要替换成实际信息。
    String url = "https://{{endpoint}}/v2/{{project_id}}/face-sets/{{face_set_name}}/faces/batch";
    String jsonStr = "{\"filter\": \"age:[20 TO 30]\"}";
    String token = "对应region的token";
    String result = doDelete(url, jsonStr, token, client);
    System.out.println(result);
}
}

class HttpDeleteWithBody extends HttpEntityEnclosingRequestBase {
    public static final String METHOD_NAME = "DELETE";

    @Override
    public String getMethod() {
        return METHOD_NAME;
    }

    public HttpDeleteWithBody(final String uri) {
        super();
        setURI(URI.create(uri));
    }

    public HttpDeleteWithBody(final URI uri) {
        super();
        setURI(uri);
    }
}

```

```
public HttpDeleteWithBody() {  
    super();  
}
```

响应示例

状态码：200

成功响应样例

```
{  
  "face_number": 1,  
  "face_set_id": "T785tx1N",  
  "face_set_name": "showFaceSet"  
}
```

状态码：400

失败响应样例

```
{  
  "error_code": "FRS.0407",  
  "error_msg": "All the data not suitable, no data to be deleted."  
}
```

状态码

状态码请参见[状态码](#)。

错误码

错误码请参见[错误码](#)。

5 公共数据结构

5.1 自定义字段

数据类型

支持String, Integer, Float, Double, Boolean, Long。

📖 说明

1. external_image_id, bounding_box, similarity, face_id, create_time, vector, _id, _all, _source为内置字段, 自定义字段避免使用。
2. 自定义字段总个数不能超过10个, key的长度范围为[1,36], 可以由数字、字母、下划线和中划线组成。
3. String类型的value长度范围为[1,256], 可以由数字、字母、下划线和中划线组成。
4. 字段重复定义会被覆盖。
5. 不支持数值类型添加后缀的输入方式, 如1.0f, 100L, 1.0d等。

语法逻辑

"external_fields"以JSON形式定义, 名称和类型对应。

```
Request Body:
{
  "face_set_name": "test",
  "face_set_capacity": "100000",
  "external_fields": {
    "location": {
      "type": "long"
    },
    "timestamp": {
      "type": "integer"
    },
    "male": {
      "type": "boolean"
    },
    "title": {
      "type": "string"
    },
    "weight": {
      "type": "double"
    }
  }
}
```

```
    "score": {  
      "type": "float"  
    }  
  }  
}
```

5.2 sort 语法

json array形式，排序只支持数值类型，字段重复定义会被覆盖。

- 例1：单个排序字段

```
"sort": [  
  {  
    "location": "desc"  
  }  
]
```

- 例2：多个排序字段

```
"sort": [  
  {  
    "timestamp": "desc"  
  },  
  {  
    "rowkey": "asc"  
  }  
]
```

5.3 filter 语法

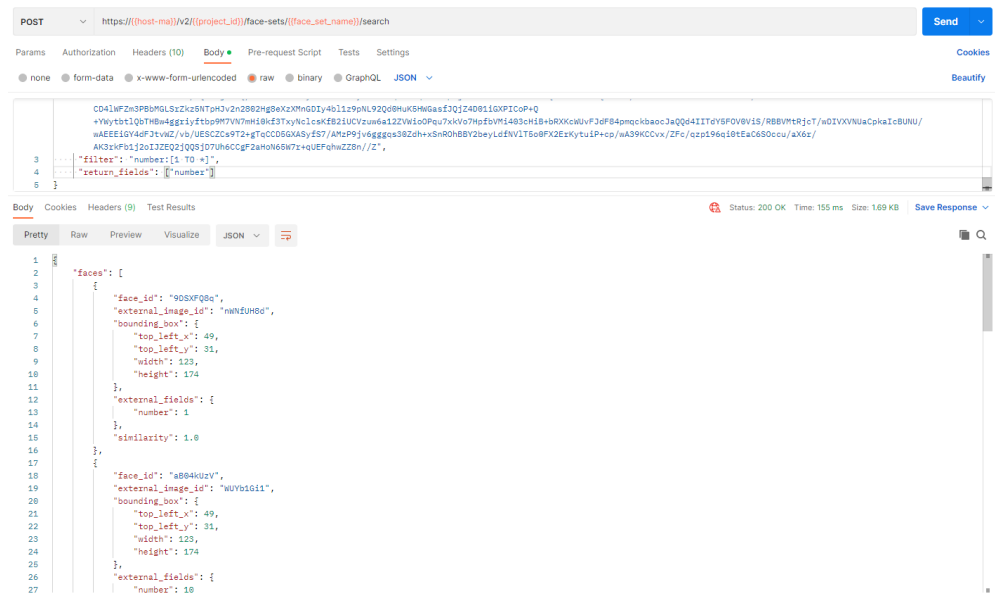
数值类型

1. 数值类型的范围查询，[]包含两端点值，{}不包含两端点值。

```
rowkey:[1 TO *] 大于等于1  
rowkey:[* TO 1] 小于等于1  
rowkey:[1 TO 10] 1到10
```

Request Body示例：

```
{  
  "image_base64": "/9j/4AAQSkZJRgABAQAAQABAAD...",  
  "filter": "number:[1 TO *]"  
}
```

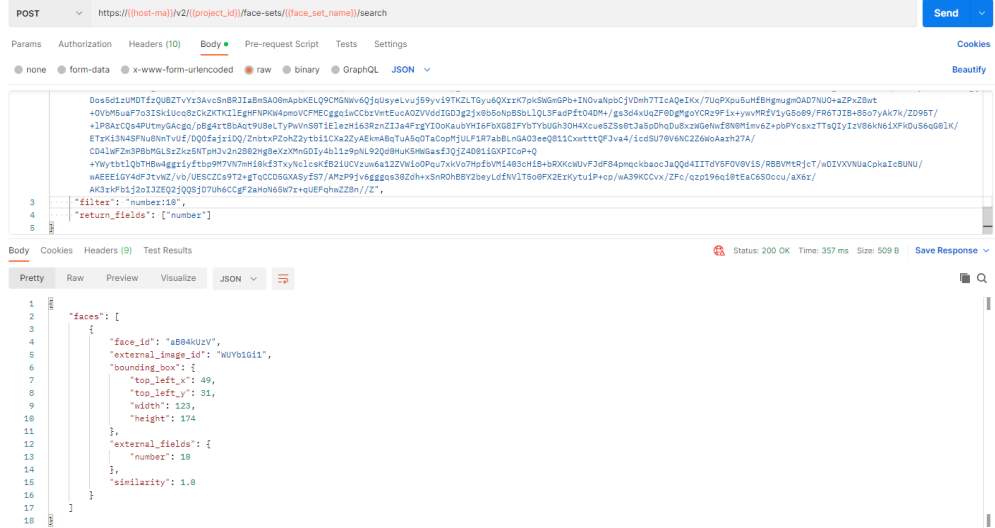


2. 单值。

rowkey:1 值为1

Request Body示例:

```
{
  "image_base64": "/9j/4AAQSkZJRgABAQAAQABAAD...",
  "filter": "number:10"
}
```

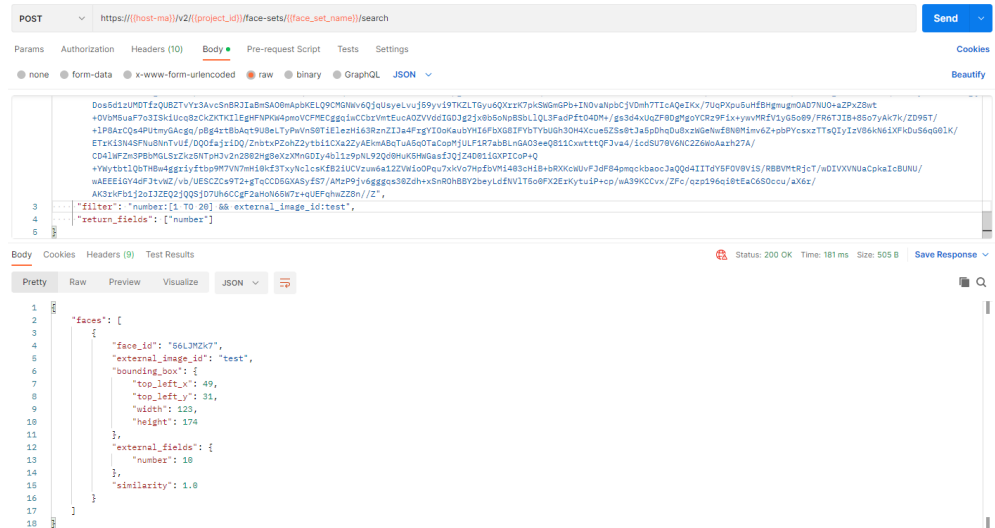


3. 多条件, 使用()区分优先级。

rowkey:[1 TO *] && externalImageID:1
(rowkey:[1 TO *] && externalImageID:1) || timestamp:1000
rowkey:[1 TO *] && (externalImageID:1 || timestamp:1000)

Request Body示例:

```
{
  "image_base64": "/9j/4AAQSkZJRgABAQAAQABAAD...",
  "filter": "number:[1 TO 20] && external_image_id:test"
}
```

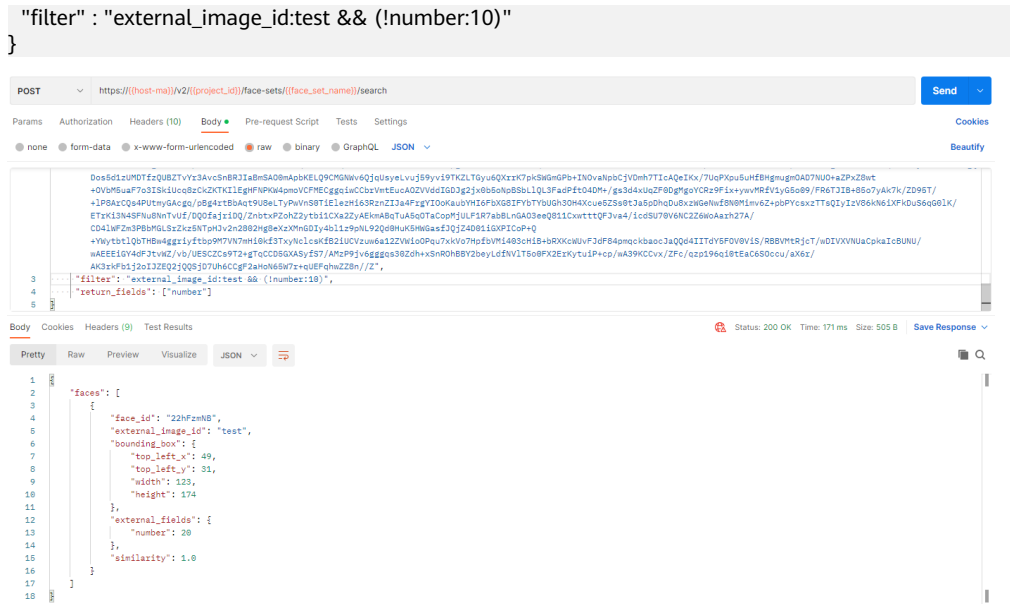


4. 非, 需要在非语句前后加括号。

externalImageID:1 && (!rowkey:2)

Request Body示例:

```
{
  "image_base64": "/9j/4AAQSkZJRgABAQAAQABAAD...",
  "filter": "number:[1 TO 20] && external_image_id:test",
  "rowkey": 1
}
```



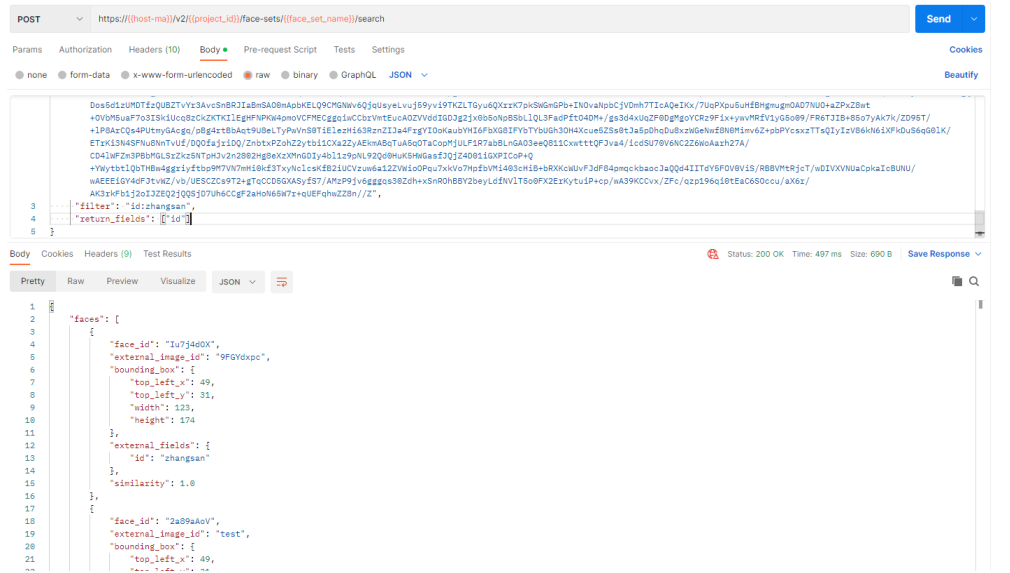
字符串类型

1. 单值

title:quick

Request Body示例:

```
{
  "image_base64": "/9j/4AAQSkZJRgABAQAAQABAAD...",
  "filter": "id:zhngsan"
}
```

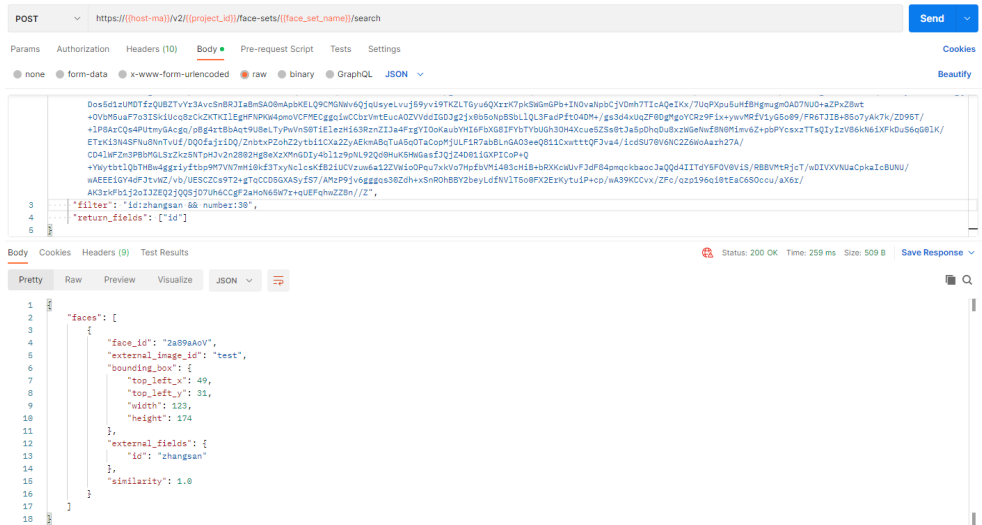


2. 多条件

title:quick && color:brown

Request Body示例:

```
{
  "image_base64": "/9j/4AAQSkZJRgABAQAAQABAAD...",
  "filter": "id:zhngsan && number:30"
}
```



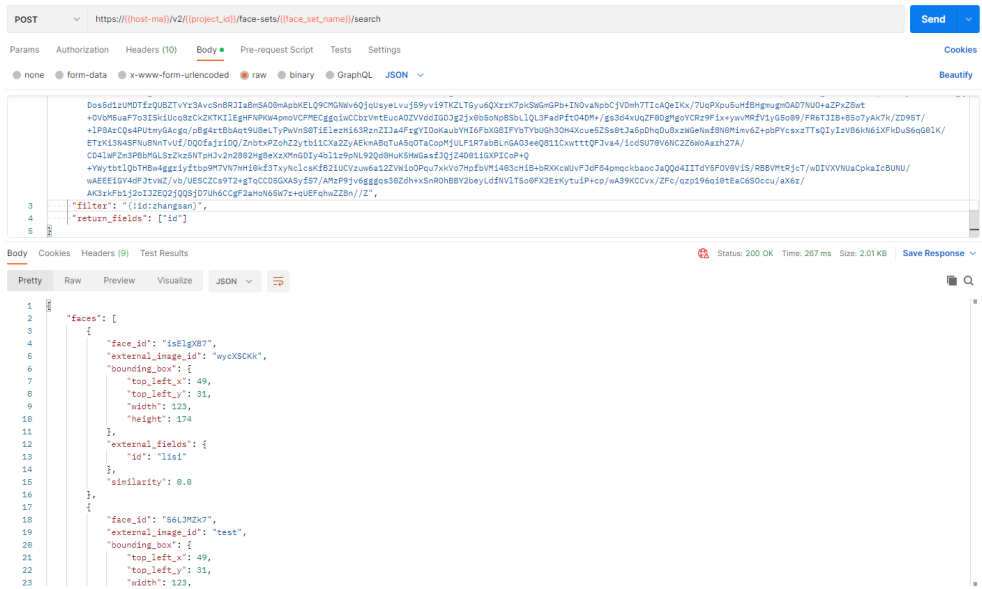
3. 非，使用括号分隔 (!color:blue)

Request Body示例:

```

{
  "image_base64": "/9j/4AAQSkZJRgABAQAAQABAAD...",
  "filter": "!(id:zhangsan)"
}

```



boolean 类型

单值

male:true

说明

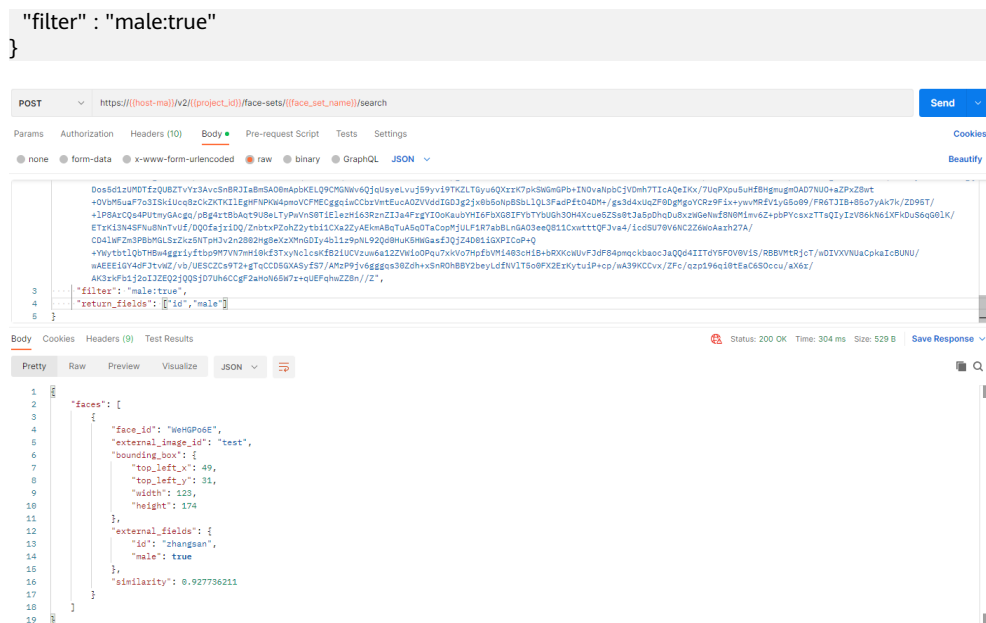
1. 逻辑条件过多，请使用()来区分优先级。
2. 不支持空字符串，null值搜索。

Request Body示例:

```

{
  "image_base64": "/9j/4AAQSkZJRgABAQAAQABAAD...",
  "filter": "!(id:zhangsan)"
}

```

5.4 消息对象结构

5.4.1 AllParam

功能介绍

涉及到的所有参数类型的详细说明。

参数说明

表 5-1 结果格式说明表

名称	类型	说明
project_id	String	项目ID，获取方法请参见 获取项目ID/账号名/AK/SK 。
image_url	String	图片的URL路径，目前仅支持华为云上OBS的URL，且人脸识别服务有权限读取该OBS桶的数据。开通读取权限的操作请参见 服务授权 。
image_file	File	本地图片文件，图片不能超过8MB，建议小于 1MB 。上传文件时，请求格式为multipart。
image_base64	String	图像数据，Base64编码，要求： <ul style="list-style-type: none"> Base64编码后大小不超过8MB，建议。 图片为JPG/JPEG/BMP/PNG格式。
similarity	Double	人脸相似度，1表示最大，0表示最小，值越大表示越相似。一般情况下超过0.93即可认为是同一个人。

名称	类型	说明
face_set_name	String	人脸库名称，1位到64位之间，可以包含字母、数字、中划线或者下划线，不能包含其他的特殊字符。
face_set_capacity	Integer	人脸库最大的容量，填写1万整数倍的数字，例如30000。默认为100000，最大值为100000，可通过创建新的人脸库进行扩容，每个用户可使用10个人脸库，每个人脸库容量为10万个人脸特征。如需扩容单个人脸库规模，请联系华为云客服确认扩容规模与价格。
face_id	String	导入人脸时，系统返回的人脸编号，为8个随机生成的大小写字母组成。
external_image_id	String	用户指定的图片外部ID，与当前图像绑定。用户没提供，系统会生成一个。该ID长度范围为1~36位，可以包含字母、数字、中划线或者下划线，不包含其他的特殊字符。
external_fields	Json	根据用户自定义数据类型，填入相应的数值。创建人脸库时，定义该字段。具体参见 自定义字段 。
top_n	Integer	返回查询到最相似的N张人脸，N默认为10。如果返回前5个，则该变量N的值为5。取值范围1~1000。
threshold	Double	人脸相似度阈值，低于这个阈值则不返回，取值范围[0,1]，一般情况下建议取值0.93，默认为0。
offset	Integer	从第几条数据读起，默认为0。
limit	Integer	读取多少条，默认为5。
video_url	String	视频的URL路径，目前仅支持华为云上OBS的URL，且人脸识别服务有权限读取该OBS桶的数据。开通读取权限的操作请参见 服务授权 。 视频要求： <ul style="list-style-type: none"> ● 视频Base64编码后大小不超过8MB。 ● 限制视频时长1~15秒。 ● 建议帧率10fps~30fps。 ● 封装格式：mp4、avi、flv、webm、asf、mov。 ● 视频编码格式：h261、h263、h264、hevc、vc1、vp8、vp9、wmv3。

名称	类型	说明
video_file	File	本地视频文件。上传文件时，请求格式为 multipart。视频要求： <ul style="list-style-type: none"> ● 视频文件大小不超过8MB，建议客户端压缩到200KB~2MB。 ● 限制视频时长1~15秒。 ● 建议帧率10fps~30fps。 ● 封装格式：mp4、avi、flv、webm、asf、mov。 ● 视频编码格式：h261、h263、h264、hevc、vc1、vp8、vp9、wmv3。
video_base64	String	视频数据，Base64编码，要求： <ul style="list-style-type: none"> ● Base64编码后大小不超过8MB，建议客户端压缩到200KB~2MB。 ● 限制视频时长1~15秒。 ● 建议帧率10fps~30fps。 ● 封装格式：mp4、avi、flv、webm、asf、mov。 ● 视频编码格式：h261、h263、h264、hevc、vc1、vp8、vp9、wmv3。
actions	String	动作代码顺序列表，英文逗号(,)分隔。建议单动作，目前支持的动作有： <ul style="list-style-type: none"> ● 1: 左摇头 ● 2: 右摇头 ● 3: 点头 ● 4: 嘴部动作
action_time	String	该参数为动作时间数组拼接的字符串，数组的长度和actions的数量一致，每一项代表了对应次序动作的起始时间和结束时间，单位为距视频开始的毫秒数。
error_code	String	调用失败时的错误码。
error_msg	String	调用失败时的错误信息。
attributes	String	是否返回人脸属性，希望获取的属性列表，多个属性用逗号隔开。目前支持的属性有： <ul style="list-style-type: none"> ● 0: 人脸姿态 ● 2: 年龄 ● 3: 人脸关键点 ● 4: 装束(帽子、眼镜) ● 5: 笑脸

5.4.2 DetectFace

功能介绍

人脸检测以及人脸比对返回的人脸结构。

参数说明

表 5-2 结构格式说明表

名称	类型	说明
bounding_box	BoundingBox object	人脸在图像中的位置。 BoundingBox结构见 BoundingBox 。
landmark	Landmark object	人脸关键点位置，人脸比对没有此值。 Landmark结构见 Landmark 。

5.4.3 Landmark

功能介绍

人脸关键点结构。

参数说明

表 5-3 结构格式说明表

名称	类型	说明
nose_contour	List [Point]	鼻子轮廓，Point为轮廓坐标值。
mouth_contour	List [Point]	嘴巴轮廓，Point为轮廓坐标值。
eyebrow_contour	List [Point]	眉毛轮廓，Point为轮廓坐标值。
eyes_contour	List [Point]	眼睛轮廓，Point为轮廓坐标值。
face_contour	List [Point]	人脸轮廓，Point为轮廓坐标值。

5.4.4 Attributes

功能介绍

人脸属性结构。

参数说明

表 5-4 结构格式说明表

名称	类型	说明
age	Integer	年龄。
dress	List of strings	包含glass和hat两个属性结果。
glass	String	是否戴眼镜： <ul style="list-style-type: none"> • yes: 戴眼镜 • none: 未戴眼镜 • unknown: 未知
hat	String	是否戴帽子： <ul style="list-style-type: none"> • yes: 戴帽子 • none: 未戴帽子 • unknown: 未知
mask	String	是否戴口罩： <ul style="list-style-type: none"> • yes: 戴口罩 • none: 未戴口罩 • unknown: 未知
hair	String	发型： <ul style="list-style-type: none"> • long: 长发 • short: 短发 • unknown: 未知
beard	String	胡须： <ul style="list-style-type: none"> • yes: 有胡须 • none: 无胡须 • unknown: 未知
phototype	String	图片类型： <ul style="list-style-type: none"> • idcard: 证件照 • monitor: 摄像头监控 • internet photo: 网络图片
smile	String	笑脸。
quality	FaceQuality	图片中人脸的遮挡度、模糊度、光照强度、姿态角度。
expression	FaceExpression	人脸表情，包括中性、高兴、害怕、惊讶、伤心、生气、厌恶。

名称	类型	说明
face_angle	Integer	人脸图片旋转角（顺时针偏转角度），支持0°、90°、180°和270°图片旋转。
dress	List of strings	包含glass和hat两个属性结果。
glass	String	是否戴眼镜： <ul style="list-style-type: none"> • yes: 戴眼镜 • dark: 戴墨镜 • none: 未戴眼镜 • unknown: 未知
hat	String	是否戴帽子： <ul style="list-style-type: none"> • yes: 戴帽子 • none: 未戴帽子 • unknown: 未知
headpose	List of doubles	人脸轮廓坐标值。
pitch_angle	Double	围绕X轴旋转，俯仰角，范围[-180,180]。
roll_angle	Double	围绕Z轴旋转，翻滚角，范围[-180,180]。
yaw_angle	Double	围绕Y轴旋转，偏航角，范围[-180,180]。

5.4.5 FaceQuality

功能介绍

人脸质量结构。

参数说明

表 5-5 结构格式说明表

名称	类型	说明
total_score	Double	人脸质量总分，取值范围[0-1]，分值越大质量越高。
blur	Double	模糊度，取值范围[0-1]，分值越大模糊问题越严重。
pose	Double	姿态，取值范围[0-1]，分值越大姿态问题越严重。

名称	类型	说明
occlusion	Double	遮挡, 取值范围[0-1], 分值越大遮挡问题越严重。
illumination	Double	光照, 取值范围[0-1], 分值越大光照问题越严重。

5.4.6 FaceExpression

功能介绍

人脸表情结构。

参数说明

表 5-6 结构格式说明表

名称	类型	说明
type	String	人脸表情类型 <ul style="list-style-type: none"> • neutral: 中性 • happy: 高兴 • fear: 害怕 • surprise: 惊讶 • sad: 伤心 • angry: 生气 • disgust: 厌恶 • unknown: 图片质量问题导致未识别
probability	Double	表情置信度, 取值范围[0-1]。

5.4.7 FaceSetFace

功能介绍

人脸库当中的人脸结构。

参数说明

表 5-7 结构格式说明表

名称	类型	说明
bounding_box	BoundingBox object	人脸在图像中的位置。 BoundingBox结构见 BoundingBox 。
face_id	String	人脸ID，由系统内部生成的唯一ID。
external_image_id	String	人脸所在的外部图片ID。
external_fields	Json	用户添加的额外字段。

5.4.8 SearchFace

功能介绍

人脸搜索返回的人脸结构。

参数说明

表 5-8 结构格式说明表

名称	类型	说明
bounding_box	BoundingBox object	人脸在图像中的位置。 BoundingBox结构见 BoundingBox 。
face_id	String	人脸ID，由系统内部生成的唯一ID。
external_image_id	String	人脸所在的外部图片ID。
similarity	Double	人脸搜索时用于被检索的相似度。
external_fields	Json	用户添加的额外自定义字段。

5.4.9 FaceSetInfo

功能介绍

人脸库的基本信息。

参数说明

表 5-9 结构格式说明表

名称	类型	说明
face_set_name	String	人脸库名称。
face_set_id	String	人脸库ID，随机生成的包含八个字符的字符串。
create_date	String	创建时间。
face_set_capacity	Integer	人脸库最大的容量。
face_number	Integer	人脸库当中的人脸数量。
external_fields	Json	用户的自定义字段。

5.4.10 BoundingBox

功能介绍

人脸在图像中的位置，坐标系的原点 (0,0) 在左上角。

参数说明

表 5-10 结构格式说明表

名称	类型	说明
top_left_x	Integer	矩形框左上角横坐标。
top_left_y	Integer	矩形框左上角纵坐标。
width	Integer	矩形框宽度。
height	Integer	矩形框高度。

5.4.11 VideoDetectResult

功能介绍

视频活体检测结果结构体。

参数说明

表 5-11 结构格式说明表

名称	类型	说明
alive	Boolean	是否是活体。
picture	String	检测出最大人脸的图片base64。
actions	List	动作列表。详细参数请参见 表 字段要素说明 。

表 5-12 actions 字段要素说明

名称	类型	说明
action	Integer	动作编号，取值范围：[1,2,3,4]，其中： <ul style="list-style-type: none"> • 1：左摇头 • 2：右摇头 • 3：点头 • 4：嘴部动作
confidence	Double	系统对检测对象是否为活体的判定置信程度，取值范围为0到1。置信度越高，表示系统越确定检测对象是一个真实的活体。可结合alive参数进行判断，alive返回true表示为活体。

5.4.12 LivelessDetectResult

功能介绍

静默活体检测结果结构体。

结构格式说明

表 5-13 结构格式说明表

名称	类型	说明
alive	Boolean	是否是活体。
picture	String	检测出最大人脸的图片base64字符串。

名称	类型	说明
confidence	Double	系统对检测对象是否为活体的判定置信程度，取值范围为0到1。置信度越高，表示系统越确定检测对象是一个真实的活体。可结合alive参数进行判断，alive返回true表示为活体。

5.4.13 ServiceInfo

功能介绍

记录子服务信息。

参数说明

表 5-14 结构格式说明表

名称	类型	说明
subscribe_status	Boolean	是否开通该子服务。
create_time	String	开通该子服务时间。

5.4.14 WarningList

功能介绍

视频活体检测警告信息。

参数说明

表 5-15 结构格式说明

名称	类型	说明
warningCode	Integer	警告ID。
warningMsg	String	警告消息。

表 5-16 错误提示

warningCode	warningMsg
1	人脸没有朝向前方。

warningCode	warningMsg
2	视频没有超过1秒。
3	视频超过15秒。
4	两个人脸。
5	没有人脸。
6	动作幅度太小。
7	视频质量差或者视频拍摄不是真人。
8	选择不出优选图片。
101	整体人脸质量过低。
102	人脸模糊。
103	人脸姿态太大。
104	人脸有遮挡。
105	图片太暗，光照不够。
106	图片中包含多张人脸。

6 高频报错处理办法

操作场景

本节内容介绍了调用API时，常见的报错及处理办法。

表 6-1 调用 API 高频报错处理办法

常见报错	处理办法
FRS.0020子服务未开通	开通该服务，检查开通服务的区域（或账号）与调用服务的区域（或账号）是否一致，检查API的URL是否拼写正确。
APIG.0101	检查请求地址或者URL是否填写正确。
账密报错The username or password is wrong	请确认近期华为云账号是否有升级为华为账号。如果进行过升级，建议您创建一个IAM账户，使用该账户获取Token。

FRS.0020 子服务未开通

在调用人脸识别接口时，报错“FRS.0020”，子服务未开通。该报错表示您的子服务没有开通，或url填写错误。

- 如未开通服务请登录**人脸识别服务控制台**人脸识别服务控制台选择“开通服务”。
- 如已经开通服务，请检查调用接口的请求地址是否正确。

您还可以通过这个视频教程了解如何构造请求调用API：<https://bbs.huaweicloud.com/videos/102987>。

图 6-1 检查区域信息



APIG.0101

在调用人脸识别接口时，报错“APIG.0101”，“The API does not exist or has not been published in the environment”。该报错表示调用接口时使用的请求地址或者URL不存在或没有发布。

参考API使用说明获取正确的请求样例，并检查请求样例中需要修改的参数“{endpoint}”、{project_id}，是否按规范进行替换。同时检查url中填写的endpoint区域时否和开通服务区域一致。

1. 参考API使用说明获取正确的请求样例。
查询服务状态
`https://{endpoint}/v1/{project_id}/subscribe`
2. 将请求样例中的“{endpoint}”进行替换。人脸识别服务的请求地请参考[终端节点](#)章节进行获取。
例如，如果您的服务部署在华北-北京四区域，则该区域的endpoint为“face.cn-north-4.myhuaweicloud.com”。
3. 将请求样例中的“{project_id}”进行替换。项目ID的获取方法请参见[获取项目ID](#)。

URL请参考[构造请求](#)构造请求章节。

账密报错 The username or password is wrong

获取Token时出现“The username or password is wrong.”。

请确认近期华为云账号是否有升级为华为账号。当前，如果您通过华为账号入口登录华为云账号，就会指引升级。华为云账号若已升级为华为账号，将不支持获取账号Token。

建议您创建一个IAM账户，使用该账户获取Token。

详细处理步骤请参见[如何处理账密报错](#)。

7 附录

7.1 状态码

表 7-1 状态码

状态码	编码	说明
100	Continue	继续请求。 这个临时响应用来通知客户端，它的部分请求已经被服务器接收，且仍未被拒绝。
101	Switching Protocols	切换协议。只能切换到更高级的协议。 例如，切换到HTTP的新版本协议。
200	OK	服务器已成功处理了请求。
201	Created	创建类的请求完全成功。
202	Accepted	已经接受请求，但未处理完成。
203	Non-Authoritative Information	非授权信息，请求成功。
204	NoContent	请求完全成功，同时HTTP响应不包含响应体。 在响应OPTIONS方法的HTTP请求时返回此状态码。
205	Reset Content	重置内容，服务器处理成功。
206	Partial Content	服务器成功处理了部分GET请求。
300	Multiple Choices	多种选择。请求的资源可包括多个位置，相应可返回一个资源特征与地址的列表用于用户终端（例如：浏览器）选择。
301	Moved Permanently	永久移动，请求的资源已被永久的移动到新的URI，返回信息会包括新的URI。

状态码	编码	说明
302	Found	资源被临时移动。
303	See Other	查看其它地址。 使用GET和POST请求查看。
304	Not Modified	所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。
305	Use Proxy	所请求的资源必须通过代理访问。
306	Unused	已经被废弃的HTTP状态码。
400	BadRequest	非法请求。 建议直接修改该请求，不要重试该请求。
401	Unauthorized	在客户端提供认证信息后，返回该状态码，表明服务端指出客户端所提供的认证信息不正确或非法。
402	Payment Required	保留请求。
403	Forbidden	请求被拒绝访问。 返回该状态码，表明请求能够到达服务端，且服务端能够理解用户请求，但是拒绝做更多的事情，因为该请求被设置为拒绝访问，建议直接修改该请求，不要重试该请求。
404	NotFound	所请求的资源不存在。 建议直接修改该请求，不要重试该请求。
405	MethodNotAllowed	请求中带有该资源不支持的方法。 建议直接修改该请求，不要重试该请求。
406	Not Acceptable	服务器无法根据客户端请求的内容特性完成请求。
407	Proxy Authentication Required	请求要求代理的身份认证，与401类似，但请求者应当使用代理进行授权。
408	Request Time-out	服务器等候请求时发生超时。 客户端可以随时再次提交该请求而无需进行任何更改。
409	Conflict	服务器在完成请求时发生冲突。 返回该状态码，表明客户端尝试创建的资源已经存在，或者由于冲突请求的更新操作不能被完成。
410	Gone	客户端请求的资源已经不存在。 返回该状态码，表明请求的资源已被永久删除。
411	Length Required	服务器无法处理客户端发送的不带Content-Length的请求信息。

状态码	编码	说明
412	Precondition Failed	未满足前提条件，服务器未满足请求者在请求中设置的其中一个前提条件。
413	Request Entity Too Large	由于请求的实体过大，服务器无法处理，因此拒绝请求。为防止客户端的连续请求，服务器可能会关闭连接。如果只是服务器暂时无法处理，则会包含一个Retry-After的响应信息。
414	Request-URI Too Large	请求的URI过长（URI通常为网址），服务器无法处理。
415	Unsupported Media Type	服务器无法处理请求附带的媒体格式。
416	Requested range not satisfiable	客户端请求的范围无效。
417	Expectation Failed	服务器无法满足Expect的请求头信息。
422	Unprocessable Entity	请求格式正确，但是由于含有语义错误，无法响应。
429	TooManyRequests	表明请求超出了客户端访问频率的限制或者服务端接收到多于它能处理的请求。建议客户端读取相应的Retry-After首部，然后等待该首部指出的时间后再重试。
500	InternalServerError	表明服务端能被请求访问到，但是不能理解用户的请求。
501	Not Implemented	服务器不支持请求的功能，无法完成请求。
502	Bad Gateway	充当网关或代理的服务器，从远端服务器接收到了一个无效的请求。
503	ServiceUnavailable	被请求的服务无效。 建议直接修改该请求，不要重试该请求。
504	ServerTimeout	请求在给定的时间内无法完成。客户端仅在为请求指定超时（Timeout）参数时会得到该响应。
505	HTTP Version not supported	服务器不支持请求的HTTP协议的版本，无法完成处理。

7.2 错误码

调用API出错后，将不会返回结果数据。调用方可根据每个API对应的错误码来定位错误原因。当调用出错时，HTTP请求返回一个4xx或5xx的HTTP状态码。返回的消息体中是具体的错误代码及错误信息。在调用方找不到错误原因时，可以联系客服，并提供错误码，以便我们尽快帮您解决问题。

- 异常响应样例

```
{
  "error_code": "FRS.0202",
  "error_msg": " The service has been freeze."
}
```

● 参数说明

参数	是否必选	类型	说明
error_code	否	String	错误码。
error_msg	否	String	错误消息。

● 错误码说明

错误码类别	状态码	错误码	说明	处理措施
服务类公共错误 (错误码区间: 1 ~ 99)	403	FRS.0002	鉴权token失败, 用户token错误或者已过期。	请参考 认证鉴权 章节, 重新获取您的token信息, 并使用获取的token来调用人脸识别服务的接口。
	400	FRS.0010	缺少请求头或者请求头为空。	请参考 构造请求 章节, 检查公共消息头是否配置。
	400	FRS.0011	缺少参数。	请检查请求参数必填参数是否遗漏。
	400	FRS.0012	请求参数格式不正确。	请检查参数输入的格式是否合法。 <ul style="list-style-type: none"> 检查必填参数是否填写完整。 检查输入数据规格是否符合约束限制。
	400	FRS.0013	输入的人脸图片尺寸过大或过小。	请上传符合规范的人脸图片。 人脸图片分辨率需小于4096*4096, 图片中人脸像素大于80*80, 建议120*120以上。
	400	FRS.0014	输入的不是json格式。	请检查body体的json格式是否合法。


错误码类别	状态码	错误码	说明	处理措施
	400	FRS.0015	图片base64解析错误。	<ul style="list-style-type: none"> 使用Base64图像数据时，注意编码格式要与API文档中的请求示例保持一致。图片转base64方法可参考如何获取图片的base64编码 使用本地图片文件时，请检查文件是否为图片，且图片类型满足约束。 使用图片URL地址时，请检查URL是否为OBS地址，且地址为图片文件的路径。
	400	FRS.0016	上传的文件格式不支持。	请参考 约束与限制 章节，上传支持识别的文件格式。
	400	FRS.0017	上传的body体超出允许的范围。	<p>上传的body体超出允许的范围，请检查图片和分辨率的大小是否在允许的范围。</p> <p>请参考约束与限制章节，使用符合规定的图片。</p>
	500	FRS.0018	服务内部错误。	请联系客服提供支持。
	400	FRS.0019	服务未开通。	请登录 人脸识别管理控制台 ，根据业务需求，选择需要开通的服务。
	400	FRS.0020	子服务未开通。	<ul style="list-style-type: none"> 请登录人脸识别管理控制台，根据业务需求，选择需要开通的子服务。 如已开通服务，请检查开通服务的区域（或账号）与调用服务的区域（或账号）是否一致；若一致请检查API的URL是否正确。
	400	FRS.0021	无效OBS的url。	请参考请求样例，检查OBS的url是否正确。

错误码类别	状态码	错误码	说明	处理措施
	400	FRS.0023	OBS上的文件大小超出范围。	请参考 约束与限制 章节，检查OBS中文件的大小是否符合接口要求
	400	FRS.0024	OBS上的文件不存在。	请检查OBS上的文件是否存在。
	400	FRS.0027	OBS未授权。	请登录 人脸识别管理控制台 ，点击右上角“服务授权”，完成OBS授权。
	403	FRS.0028	Project ID跟Token不匹配。	请检查获取Token的Project ID是否和请求url上的Project ID一致。
	400	FRS.0032	排序字段以及类型不支持，只支持数字类型。	排序字段以及类型不支持，只支持数字类型。请参考 sort语法 章节查看示例。
	429	FRS.0033	当前请求数太多，超过流控限制。	建议控制请求策略或者进行重试，建议扩容以增加QPS配额。
	400	FRS.0035	下载地址不合法。	请参考请求样例，检查OBS的url是否正确。
	400	FRS.0036	从url获取文件失败。	请使用OBS提供的url上传图片，并检查url是否正确。
服务管理错误 (错误码区间: 201 ~ 300)	400	FRS.0201	此用户已经注册了。	此用户已经注册了。
	400	FRS.0202	服务已经被冻结。	请联系客服提供支持。
	400	FRS.0203	更新用户信息失败。	请检查输入的用户信息。
人脸库资源错误 (错误码区间: 301 ~ 400)	400	FRS.0301	无法创建已经存在的人脸库。	请检查输入的人脸库名称，创建的人脸库已存在，请更换人脸库名称后重新创建。
	400	FRS.0302	人脸库不存在。	请检查输入的人脸库是否存在。
	400	FRS.0304	未检测到人脸，无法搜索。	请参考 约束与限制 章节，上传符合规范的人脸图片。

错误码类别	状态码	错误码	说明	处理措施
	400	FRS.0305	超过最大的人脸库数量，无法创建新的人脸库。	请检查人脸库数量是否超出限制联系客服提供支持。
	403	FRS.0306	超过人脸库容量限制，无法增加更多的人脸。	请新增人脸库，重新添加人脸到新的人脸库中。
人脸资源错误 (错误码区间: 401~500)	400	FRS.0401	找不到对应类型的值，无法删除。	请参考人脸资源管理接口说明。
	400	FRS.0402	输入的字段不存在。	请参考人脸资源管理接口说明。
	400	FRS.0403	Face ID不存在。	请检查输入的Face ID是否存在。
	400	FRS.0404	未检测到人脸，无法添加。	请参考 约束与限制 章节，上传符合规范的人脸图片。
	400	FRS.0405	未定义的自定义字段，无法添加。	请检查添加的自定义字段是否存在，创建自定义字段后再进行添加。
	400	FRS.0406	导入的数据类型与定义不匹配。	请检查导入的数据类型是否与定义的数据类型一致。
	400	FRS.0407	批量删除未找到匹配数据。	请检查需要删除的数据信息是否存在。
人脸比对错误 (错误码区间: 501~600)	400	FRS.0501	照片未检测到人脸。	请参考 约束与限制 章节，上传符合规范的人脸图片。
人脸搜索错误码 (错误码区间: 29~31)	400	FRS.0029	过滤格式错误。	请参考人脸搜索 请求参数 章节，检查输入格式是否合法。
	400	FRS.0030	返回字段未定义。	请参考人脸搜索 请求参数 章节，检查return_fields是否定义。
	400	FRS.0031	排序字段不在返回字段中。	请参考人脸搜索 请求参数 章节，检查排序字段是否定义。

错误码类别	状态码	错误码	说明	处理措施
活体检测错误 (错误码区间: 701 ~ 800)	400	FRS.070 1	视频解析错误。	请检查视频是否有损坏。
	400	FRS.070 2	不支持的动作。	请参考 动作活体检测 的请求参数, 检查输入的动作是否存在。
	400	FRS.070 3	Action time非法。	请检查输入是否合法。
	400	FRS.070 4	传入的action数量超过10个。	传入的action数量超过10个。
	400	FRS.070 5	匹配模式不支持。	请参考 约束与限制 章节, 检查输入是否合法。
	400	FRS.070 6	视频时长不支持, 限制[1-15]秒。	视频时长超过限制[1-15]秒, 请使用符合时长要求的视频。
	400	FRS.070 7	照片未检测到人脸。	请参考 约束与限制 章节, 检查照片中是否有人脸或者人脸是否满足约束限制。
	400	FRS.070 8	检查输入的人脸图片质量是否满足要求。	请参考 约束与限制 章节, 检查输入的人脸图片质量是否满足要求。
	400	FRS.075 1	视频数据解析失败。	请参考 约束与限制 章节, 检查输入的视频是否满足要求。
	400	FRS.070 9	检查输入的人脸图片包含多张人脸。	请检查输入的人脸图片是否包含多张人脸, 静默活体检测仅支持单人脸照片。

- 网关错误码

错误码类别	状态码	错误码	说明	处理措施
网关类错误 (错误码区间: 1 ~ 400)	404	APIG.0101	访问的API不存在或尚未在环境中发布。	<ul style="list-style-type: none"> 请检查API的URL是否拼写正确，例如，URL中是否缺少project_id。 检查URI中的区域信息是否和调用服务配置的区域是否对应。  HTTP请求方法（POST，GET等）是否正确。 具体的URI信息请在各接口的API页面查看。
	413/504	APIG.0201	请求体超过范围或者后端超时。	请检查请求的大小是否合法，请联系客服检查后端服务是否运行正常。
	401	APIG.0301	IAM身份验证信息不正确： <ul style="list-style-type: none"> decrypt token fail: token解析失败。 token expires: token过期。 verify ak-sk signature fail: AK/SK认证失败。 	<ul style="list-style-type: none"> token解析失败，请检查获取token的方法，请求体信息是否正确；检查获取token的环境与调用的环境是否一致。 token超时（token expires），请重新获取token，使用不过期的token。 请检查AK/SK是否正确（AK对应的SK错误，不匹配；AK/SK中多填了空格）。 AK/SK频繁出现鉴权出错，连续错误5次以上，被锁定5分钟（5分钟内，则一直认为其是异常的鉴权请求），5分钟后解锁重新认证。 检查账号权限，是否欠费，被冻结等。

错误码类别	状态码	错误码	说明	处理措施
	401	APIG.0307	Token需要更新	<ul style="list-style-type: none"> Token有效期为24小时，请使用重新获取Token调用API。 检查接口URL中的终端节点是否填写正确。部署在不同区域的服务，不可以跨区域调用。如果调用了不同区域的接口，导致判断为Token失效，显示APIG.0307错误码。

当您调用API时，如果遇到“APIGW”开头的错误码，请参见[API网关错误码](#)进行处理。

7.3 获取项目 ID/账号名/AK/SK

7.3.1 获取项目 ID/账号名

从控制台获取项目 ID/账号名

1. 登录[管理控制台](#)。
2. 鼠标移动至用户名，在下拉列表中单击“我的凭证”。
3. 在“我的凭证”页面，可以查看用户名、账号名，在项目列表中查看项目ID。

图 7-1 查看项目 ID



调用 API 获取项目 ID

项目ID还可以通过调用[查询指定条件下的项目信息](#)API获取。

获取项目ID的接口为“GET https://{Endpoint}/v3/projects”，其中{Endpoint}为IAM的终端节点，可以从[地区和终端节点](#)获取。选中“Headers”配置项，添加“KEY”为“X-Auth-Token”，“VALUE”为获取的Token值，详细的接口认证鉴权请参见[认证鉴权](#)。

响应示例如下，例如人脸识别服务部署的区域为“cn-north-1”，响应消息体中查找“name”为“cn-north-1”，其中projects下的“id”即为项目ID。


```
{
  "projects": [
    {
      "domain_id": "65382450e8f64ac0870cd180d14e684b",
      "is_domain": false,
      "parent_id": "65382450e8f64ac0870cd180d14e684b",
      "name": "cn-north-1",
      "description": "",
      "links": {
        "next": null,
        "previous": null,
        "self": "https://www.example.com/v3/projects/a4a5d4098fb4474fa22cd05f897d6b99"
      },
      "id": "a4a5d4098fb4474fa22cd05f897d6b99",
      "enabled": true
    }
  ],
  "links": {
    "next": null,
    "previous": null,
    "self": "https://www.example.com/v3/projects"
  }
}
```

7.3.2 获取账号 ID

在调用接口的时候，部分URL中需要填入账号ID（domain-id），所以需要先在管理控制台上获取到账号ID。账号ID获取步骤如下：

1. 注册并登录管理控制台。
2. 单击用户名，在下拉列表中单击“我的凭证”。
在“我的凭证”页面的项目列表中查看账号ID。

图 7-2 查看账号 ID



7.3.3 获取 AK/SK

1. 登录**人脸识别管理控制台**。
2. 单击页面右上角的用户名，并选择“我的凭证”。
进入“我的凭证”页面。
3. 单击“访问密钥”页签下的“新增访问密钥”。
弹出“新增访问密钥”对话框。
4. 输入“登录密码”，如果绑定了手机或者邮箱，还需要获取验证码并进行验证。
验证成功后，弹出访问密钥下载对话框。
5. 单击“确定”，并根据提示下载保存访问密钥。
6. 如果已生成过AK/SK，找到原来已下载的AK/SK文件，文件名一般为：
credentials.csv。