

智能鉴黄

API 文档

产品文档



腾讯云

【版权声明】

©2013-2019 腾讯云版权所有

本文档著作权归腾讯云单独所有，未经腾讯云事先书面许可，任何主体不得以任何形式复制、修改、抄袭、传播全部或部分本文档内容。

【商标声明】

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。

【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或模式的承诺或保证。

文档目录

API 文档

更新历史

简介

API 概览

调用方式

请求结构

公共参数

接口鉴权 v3

接口鉴权

返回结果

图片内容审核相关接口

图像内容审核

数据结构

错误码

智能鉴黄 API 2017

智能鉴黄 API

鉴权签名

错误码说明

API 文档

更新历史

最近更新时间：2019-04-26 15:01:36

第 1 次发布

发布时间：2019-04-26 14:53:40

本次发布包含了以下内容：

改善已有的文档。

新增接口：

- [ImageModeration](#)

新增数据结构：

- [Candidate](#)
- [DisgustResult](#)
- [FaceRect](#)
- [FaceResult](#)
- [PoliticsResult](#)
- [PornResult](#)
- [TerrorismResult](#)

简介

最近更新时间：2019-07-24 15:39:46

智能鉴黄 API 升级到 **3.0 版本**。全新的 API 接口文档更加规范和全面，统一的参数风格和公共错误码，统一的 SDK/CLI 版本与 API 文档严格一致，给您带来简单快捷的使用体验。支持全地域就近接入让您更快连接腾讯云产品。

智能内容审核产品介绍

产品概述

智能鉴黄现已全面升级至“智能内容审核（TI Content Moderation，简称 TICM）”。

智能内容审核基于腾讯海量数据资源和深度学习技术，支持准确识别图片、视频等多媒体内容中的色情、暴力恐怖、政治敏感等违规风险，解放审核人力，保护业务健康发展。

图像内容审核产品功能

色情识别

通过学习和分析图像的肤色、姿态和场景等，智能识别图片和视频中的色情和性感内容，准确率高达 99.95%，有效规避运营风险。

暴恐识别

暴恐识别模型更加严格，支持智能识别图像中的恐怖组织头目、暴恐场景、旗帜标识等违禁内容，保护业务远离涉暴涉恐风险。

政治敏感识别

支持政治人物、敏感人物和涉政敏感场景等识别，有效降低业务违规和侵权风险。

API 概览

最近更新时间：2019-04-26 15:01:36

图片内容审核相关接口

接口名称	接口功能
ImageModeration	图像内容审核

调用方式

请求结构

最近更新时间：2019-08-14 19:18:17

1. 服务地址

API 支持就近地域接入，本产品就近地域接入域名为 `ticm.tencentcloudapi.com`，也支持指定地域域名访问，例如广州地域的域名为 `ticm.ap-guangzhou.tencentcloudapi.com`。

推荐使用就近地域接入域名。根据调用接口时客户端所在位置，会自动解析到**最近**的某个具体地域的服务器。例如在广州发起请求，会自动解析到广州的服务器，效果和指定 `ticm.ap-guangzhou.tencentcloudapi.com` 是一致的。

注意：对时延敏感的业务，建议指定带地域的域名。

目前支持的域名列表为：

接入地域	域名
就近地域接入（推荐，只支持非金融区）	<code>ticm.tencentcloudapi.com</code>
华南地区(广州)	<code>ticm.ap-guangzhou.tencentcloudapi.com</code>
华东地区(上海)	<code>ticm.ap-shanghai.tencentcloudapi.com</code>
华北地区(北京)	<code>ticm.ap-beijing.tencentcloudapi.com</code>
西南地区(成都)	<code>ticm.ap-chengdu.tencentcloudapi.com</code>
西南地区(重庆)	<code>ticm.ap-chongqing.tencentcloudapi.com</code>
港澳台地区(中国香港)	<code>ticm.ap-hongkong.tencentcloudapi.com</code>
亚太东南(新加坡)	<code>ticm.ap-singapore.tencentcloudapi.com</code>
亚太东南(曼谷)	<code>ticm.ap-bangkok.tencentcloudapi.com</code>
亚太南部(孟买)	<code>ticm.ap-mumbai.tencentcloudapi.com</code>
亚太东北(首尔)	<code>ticm.ap-seoul.tencentcloudapi.com</code>
亚太东北(东京)	<code>ticm.ap-tokyo.tencentcloudapi.com</code>
美国东部(弗吉尼亚)	<code>ticm.na-ashburn.tencentcloudapi.com</code>
美国西部(硅谷)	<code>ticm.na-siliconvalley.tencentcloudapi.com</code>
北美地区(多伦多)	<code>ticm.na-toronto.tencentcloudapi.com</code>
欧洲地区(法兰克福)	<code>ticm.eu-frankfurt.tencentcloudapi.com</code>
欧洲地区(莫斯科)	<code>ticm.eu-moscow.tencentcloudapi.com</code>

注意：由于金融区和非金融区是隔离不互通的，因此当访问金融区服务时（公共参数 Region 为金融区地域），需要同时指定带金融区地域的域名，最好和 Region 的地域保持一致。

金融区接入地域	金融区域名
华东地区(上海金融)	ticm.ap-shanghai-fsi.tencentcloudapi.com
华南地区(深圳金融)	ticm.ap-shenzhen-fsi.tencentcloudapi.com

2. 通信协议

腾讯云 API 的所有接口均通过 HTTPS 进行通信，提供高安全性的通信通道。

3. 请求方法

支持的 HTTP 请求方法:

- POST (推荐)
- GET

POST 请求支持的 Content-Type 类型：

- application/json (推荐)，必须使用 TC3-HMAC-SHA256 签名方法。
- application/x-www-form-urlencoded，必须使用 HmacSHA1 或 HmacSHA256 签名方法。
- multipart/form-data (仅部分接口支持)，必须使用 TC3-HMAC-SHA256 签名方法。

GET 请求的请求包大小不得超过32KB。POST 请求使用签名方法为 HmacSHA1、HmacSHA256 时不得超过1MB。POST 请求使用签名方法为 TC3-HMAC-SHA256 时支持10MB。

4. 字符编码

均使用 UTF-8 编码。

公共参数

最近更新时间：2019-08-14 19:18:17

公共参数是用于标识用户和接口鉴权目的的参数，如非必要，在每个接口单独的接口文档中不再对这些参数进行说明，但每次请求均需要携带这些参数，才能正常发起请求。

签名方法 v3

使用 TC3-HMAC-SHA256 签名方法时，公共参数需要统一放到 HTTP Header 请求头部中，如下：

参数名称	类型	必选	描述
X-TC-Action	String	是	操作的接口名称。取值参考接口文档中输入参数公共参数 Action 的说明。例如云服务器的查询实例列表接口，取值为 DescribeInstances。
X-TC-Region	String	是	地域参数，用来标识希望操作哪个地域的数据。接口接受的地域取值参考接口文档中输入参数公共参数 Region 的说明。注意：某些接口不需要传递该参数，接口文档中会对此特别说明，此时即使传递该参数也不会生效。
X-TC-Timestamp	Integer	是	当前 UNIX 时间戳，可记录发起 API 请求的时间。例如 1529223702。注意：如果与服务器时间相差超过5分钟，会引起签名过期错误。
X-TC-Version	String	是	操作的 API 的版本。取值参考接口文档中输入公共参数 Version 的说明。例如云服务器的版本 2017-03-12。
Authorization	String	是	HTTP 标准身份认证头部字段，例如： TC3-HMAC-SHA256 Credential=AKIDEXAMPLE/Date/service/tc3_request, SignedHeaders=content-type;host, Signature=fe5f80f77d5fa3beca038a248ff027d0445342fe2855ddc963176630326f1024 其中， - TC3-HMAC-SHA256：签名方法，目前固定取该值； - Credential：签名凭证，AKIDEXAMPLE 是 SecretId；Date 是 UTC 标准时间的日期，取值需要和公共参数 X-TC-Timestamp 换算的 UTC 标准时间日期一致；service 为产品名，通常为域名前缀，例如域名 cvm.tencentcloudapi.com 意味着产品名是 cvm。本产品取值为 tcm； - SignedHeaders：参与签名计算的头部信息，content-type 和 host 为必选头部； - Signature：签名摘要。
X-TC-Token	String	否	临时证书所用的 Token，需要结合临时密钥一起使用。临时密钥和 Token 需要到访问管理服务调用接口获取。长期密钥不需要 Token。

假设用户想要查询广州地域的云服务器实例列表，则其请求结构按照请求 URL、请求头部、请求体示例如下：

HTTP GET 请求结构示例：

```
https://cvm.tencentcloudapi.com/?Limit=10&Offset=0
```

```
Authorization: TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5yKBZQpn74WfkmLPx3EXAMPLE/2018-10-09/cvm/tc3_request, SignedHeaders=content-type;host, Signature=5da7a33f6993f0614b047e5df4582db9e9bf4672ba50567dba16c6ccf174c474
```

```
Content-Type: application/x-www-form-urlencoded
```

```
Host: cvm.tencentcloudapi.com
X-TC-Action: DescribeInstances
X-TC-Version: 2017-03-12
X-TC-Timestamp: 1539084154
X-TC-Region: ap-guangzhou
```

HTTP POST (application/json) 请求结构示例：

```
https://cvm.tencentcloudapi.com/

Authorization: TC3-HMAC-SHA256 Credential=AKIDEXAMPLE/2018-05-30/cvm/tc3_request, SignedHeaders=content-type;host, Signature=582c400e06b5924a6f2b5d7d672d79c15b13162d9279b0855cfba6789a8edb4c
Content-Type: application/json
Host: cvm.tencentcloudapi.com
X-TC-Action: DescribeInstances
X-TC-Version: 2017-03-12
X-TC-Timestamp: 1527672334
X-TC-Region: ap-guangzhou

{"Offset":0,"Limit":10}
```

HTTP POST (multipart/form-data) 请求结构示例 (仅特定的接口支持)：

```
https://cvm.tencentcloudapi.com/

Authorization: TC3-HMAC-SHA256 Credential=AKIDEXAMPLE/2018-05-30/cvm/tc3_request, SignedHeaders=content-type;host, Signature=582c400e06b5924a6f2b5d7d672d79c15b13162d9279b0855cfba6789a8edb4c
Content-Type: multipart/form-data; boundary=58731222010402
Host: cvm.tencentcloudapi.com
X-TC-Action: DescribeInstances
X-TC-Version: 2017-03-12
X-TC-Timestamp: 1527672334
X-TC-Region: ap-guangzhou

--58731222010402
Content-Disposition: form-data; name="Offset"

0
--58731222010402
Content-Disposition: form-data; name="Limit"

10
--58731222010402--
```

签名方法 v1

使用 HmacSHA1 和 HmacSHA256 签名方法时，公共参数需要统一放到请求串中，如下

参数名称	类型	必选	描述
------	----	----	----

参数名称	类型	必选	描述
Action	String	是	操作的接口名称。取值参考接口文档中输入参数公共参数 Action 的说明。例如云服务器的查询实例列表接口，取值为 DescribeInstances。
Region	String	是	地域参数，用来标识希望操作哪个地域的数据。接口接受的地域取值参考接口文档中输入参数公共参数 Region 的说明。注意：某些接口不需要传递该参数，接口文档中会对此特别说明，此时即使传递该参数也不会生效。
Timestamp	Integer	是	当前 UNIX 时间戳，可记录发起 API 请求的时间。例如1529223702，如果与当前时间相差过大，会引起签名过期错误。
Nonce	Integer	是	随机正整数，与 Timestamp 联合起来，用于防止重放攻击。
SecretId	String	是	在 云API密钥 上申请的标识身份的 SecretId，一个 SecretId 对应唯一的 SecretKey，而 SecretKey 会用来生成请求签名 Signature。
Signature	String	是	请求签名，用来验证此次请求的合法性，需要用户根据实际的输入参数计算得出。具体计算方法参见接口鉴权文档。
Version	String	是	操作的 API 的版本。取值参考接口文档中输入公共参数 Version 的说明。例如云服务器的版本 2017-03-12。
SignatureMethod	String	否	签名方式，目前支持 HmacSHA256 和 HmacSHA1。只有指定此参数为 HmacSHA256 时，才使用 HmacSHA256 算法验证签名，其他情况均使用 HmacSHA1 验证签名。
Token	String	否	临时证书所用的 Token，需要结合临时密钥一起使用。临时密钥和 Token 需要到访问管理服务调用接口获取。长期密钥不需要 Token。

假设用户想要查询广州地域的云服务器实例列表，其请求结构按照请求 URL、请求头部、请求体示例如下：

HTTP GET 请求结构示例：

```
https://cvm.tencentcloudapi.com/?Action=DescribeInstances&Version=2017-03-12&SignatureMethod=HmacSHA256&Timestamp=1527672334&Signature=37ac2f4fde00b0ac9bd9eadeb459b1bbee224158d66e7ae5fcadb70b2d181d02&Region=ap-guangzhou&Nonce=23823223&SecretId=AKIDEXAMPLE
```

```
Host: cvm.tencentcloudapi.com
Content-Type: application/x-www-form-urlencoded
```

HTTP POST 请求结构示例：

```
https://cvm.tencentcloudapi.com/
```

```
Host: cvm.tencentcloudapi.com
Content-Type: application/x-www-form-urlencoded
```

```
Action=DescribeInstances&Version=2017-03-12&SignatureMethod=HmacSHA256&Timestamp=1527672334&Signature=37ac2f4fde00b0ac9bd9eadeb459b1bbee224158d66e7ae5fcadb70b2d181d02&Region=ap-guangzhou&Nonce=23823223&SecretId=AKIDEXAMPLE
```

地域列表

本产品所有接口 Region 字段的可选值如下表所示。如果接口不支持该表中的所有地域，则会在接口文档中单独说明。

区域	取值
华北地区(北京)	ap-beijing
华南地区(广州)	ap-guangzhou
港澳台地区(中国香港)	ap-hongkong
华东地区(上海)	ap-shanghai
北美地区(多伦多)	na-toronto

接口鉴权 v3

最近更新时间：2019-08-14 19:18:17

腾讯云 API 会对每个请求进行身份验证，用户需要使用安全凭证，经过特定的步骤对请求进行签名（Signature），每个请求都需要在公共请求参数中指定该签名结果并以指定的方式和格式发送请求。

申请安全凭证

本文使用的安全凭证为密钥，密钥包括 SecretId 和 SecretKey。每个用户最多可以拥有两对密钥。

- SecretId：用于标识 API 调用者身份，可以简单类比为用户名。
- SecretKey：用于验证 API 调用者的身份，可以简单类比为密码。
- **用户必须严格保管安全凭证，避免泄露，否则将危及财产安全。如已泄漏，请立刻禁用该安全凭证。**

申请安全凭证的具体步骤如下：

1. 登录 [腾讯云管理中心控制台](#)。
2. 前往 [云API密钥](#) 的控制台页面。
3. 在 [云API密钥](#) 页面，单击【新建】即可以创建一对密钥。

使用开发者资源

腾讯云 API 配套了 7 种常见的编程语言 SDK，均已开源，包括 [Python](#)、[Java](#)、[PHP](#)、[Go](#)、[NodeJS](#)、[.NET](#)、[C++](#)。同时还提供了 [API Explorer](#) 供用户在线调用、签名验证、生成 SDK 代码。如果在签名时遇到困难，请善加利用这些资源。

TC3-HMAC-SHA256 签名方法

TC3-HMAC-SHA256 签名方法相比以前的 HmacSHA1 和 HmacSHA256 签名方法，功能上覆盖了以前的签名方法，而且更安全，支持更大的请求，支持 json 格式，性能有一定提升，建议使用该签名方法计算签名。

云 API 支持 GET 和 POST 请求。对于 GET 方法，只支持 Content-Type: application/x-www-form-urlencoded 协议格式。对于 POST 方法，目前支持 Content-Type: application/json 以及 Content-Type: multipart/form-data 两种协议格式，json 格式绝大多数接口均支持，multipart 格式只有特定接口支持，此时该接口不能使用 json 格式调用，参考具体业务接口文档说明。推荐使用 POST 请求，因为两者的结果并无差异，但 GET 请求只支持 32 KB 以内的请求包。

下面以云服务器查询广州实例列表作为例子，分步骤介绍签名的计算过程。我们选择该接口是因为：

1. 云服务器默认已开通，该接口很常用；
2. 该接口是只读的，不会改变现有资源的状态；
3. 接口覆盖的参数种类较全，可以演示包含数据结构的数组如何使用。

在示例中，不论公共参数或者接口的参数，我们尽量选择容易犯错的情况。在实际调用接口时，请根据实际情况来，每个接口的参数并不相同，不要照抄这个例子的参数和值。

假设用户的 SecretId 和 SecretKey 分别是：AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE 和 Gu5t9xGARNpq86cd98joQYCN3EXAMPLE。用户想查看广州云服务器名为“未命名”的主机状态，只返回一条数据。则请求可能为：

```
curl -X POST https://cvm.tencentcloudapi.com \
-H "Authorization: TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE/2019-02-25/cvm/tc3_request, SignedHeaders=content-type;host, Signature=72e494ea809ad7a8c8f7a4507b9bddcbaa8e581f516e8da2f66e2c5a96525168" \
-H "Content-Type: application/json; charset=utf-8" \
-H "Host: cvm.tencentcloudapi.com" \
-H "X-TC-Action: DescribeInstances" \
-H "X-TC-Timestamp: 1551113065" \
-H "X-TC-Version: 2017-03-12" \
-H "X-TC-Region: ap-guangzhou" \
-d '{"Limit": 1, "Filters": [{"Values": ["\u672a\u547d\u540d"], "Name": "instance-name"}]}'
```

下面详细解释签名计算过程。

1. 拼接规范请求串

按如下伪代码格式拼接规范请求串（CanonicalRequest）：

```
CanonicalRequest =
HTTPRequestMethod + '\n' +
CanonicalURI + '\n' +
CanonicalQueryString + '\n' +
CanonicalHeaders + '\n' +
SignedHeaders + '\n' +
HashedRequestPayload
```

字段名称	解释
HTTPRequestMethod	HTTP 请求方法（GET、POST）。此示例取值为 POST。
CanonicalURI	URI 参数，API 3.0 固定为正斜杠（/）。
CanonicalQueryString	发起 HTTP 请求 URL 中的查询字符串，对于 POST 请求，固定为空字符串""，对于 GET 请求，则为 URL 中问号（?）后面的字符串内容，例如：Limit=10&Offset=0。 注意：CanonicalQueryString 需要经过 URL 编码。
CanonicalHeaders	参与签名的头部信息，至少包含 host 和 content-type 两个头部，也可加入自定义的头部参与签名以提高自身请求的唯一性和安全性。 拼接规则： 1. 头部 key 和 value 统一转成小写，并去掉首尾空格，按照 key:value\n 格式拼接； 2. 多个头部，按照头部 key（小写）的 ASCII 升序进行拼接。 此示例计算结果是 content-type:application/json; charset=utf-8\nhost:cvm.tencentcloudapi.com\n。 注意：content-type 必须和实际发送的相符合，有些编程语言网络库即使未指定也会自动添加 charset 值，如果签名时和发送时不一致，服务器会返回签名校验失败。

字段名称	解释
SignedHeaders	参与签名的头部信息，说明此次请求有哪些头部参与了签名，和 CanonicalHeaders 包含的头部内容是一一对应的。content-type 和 host 为必选头部。 拼接规则： 1. 头部 key 统一转成小写； 2. 多个头部 key（小写）按照 ASCII 升序进行拼接，并且以分号（;）分隔。 此示例为 content-type;host
HashedRequestPayload	请求正文（payload，即 body，此示例为 {"Limit": 1, "Filters": [{"Values": ["\u672a\u547d\u540d"], "Name": "instance-name"}]}）的哈希值，计算伪代码为 Lowercase(HexEncode(Hash.SHA256(RequestPayload)))，即对 HTTP 请求正文做 SHA256 哈希，然后十六进制编码，最后编码串转换成小写字母。对于 GET 请求，RequestPayload 固定为空字符串。此示例计算结果是 35e9c5b0e3ae67532d3c9f17ead6c90222632e5b1ff7f6e89887f1398934f064。

根据以上规则，示例中得到的规范请求串如下：

```
POST
/

content-type:application/json; charset=utf-8
host:cvm.tencentcloudapi.com

content-type;host
35e9c5b0e3ae67532d3c9f17ead6c90222632e5b1ff7f6e89887f1398934f064
```

2. 拼接待签名字符串

按如下格式拼接待签名字符串：

```
StringToSign =
Algorithm + \n +
RequestTimestamp + \n +
CredentialScope + \n +
HashedCanonicalRequest
```

字段名称	解释
Algorithm	签名算法，目前固定为 TC3-HMAC-SHA256。
RequestTimestamp	请求时间戳，即请求头部的公共参数 X-TC-Timestamp 取值，取当前时间 UNIX 时间戳，精确到秒。此示例取值为 1551113065。
CredentialScope	凭证范围，格式为 Date/service/tc3_request，包含日期、所请求的服务和终止字符串（tc3_request）。 Date 为 UTC 标准时间的日期，取值需要和公共参数 X-TC-Timestamp 换算的 UTC 标准时间日期一致 ；service 为产品名，必须与调用的产品域名一致。此示例计算结果是 2019-02-25/cvm/tc3_request。

字段名称	解释
HashedCanonicalRequest	前述步骤拼接所得规范请求串的哈希值，计算伪代码为 <code>Lowercase(HexEncode(Hash.SHA256(CanonicalRequest)))</code> 。此示例计算结果是 <code>5ffe6a04c0664d6b969fab9a13bdab201d63ee709638e2749d62a09ca18d7031</code> 。

注意：

1. Date 必须从时间戳 X-TC-Timestamp 计算得到，且时区为 UTC+0。如果加入系统本地时区信息，例如东八区，将导致白天和晚上调用成功，但是凌晨时调用必定失败。假设时间戳为 1551113065，在东八区的时间是 2019-02-26 00:44:25，但是计算得到的 Date 取 UTC+0 的日期应为 2019-02-25，而不是 2019-02-26。
2. Timestamp 必须是当前系统时间，且需确保系统时间和标准时间是同步的，如果相差超过五分钟则必定失败。如果长时间不和标准时间同步，可能导致运行一段时间后，请求必定失败，返回签名过期错误。

根据以上规则，示例中得到的待签名字符串如下：

```
TC3-HMAC-SHA256
1551113065
2019-02-25/cvm/tc3_request
5ffe6a04c0664d6b969fab9a13bdab201d63ee709638e2749d62a09ca18d7031
```

3. 计算签名

1) 计算派生签名密钥，伪代码如下：

```
SecretKey = "Gu5t9xGARNpq86cd98joQYCN3EXAMPLE"
SecretDate = HMAC_SHA256("TC3" + SecretKey, Date)
SecretService = HMAC_SHA256(SecretDate, Service)
SecretSigning = HMAC_SHA256(SecretService, "tc3_request")
```

字段名称	解释
SecretKey	原始的 SecretKey，即 <code>Gu5t9xGARNpq86cd98joQYCN3EXAMPLE</code> 。
Date	即 Credential 中的 Date 字段信息。此示例取值为 <code>2019-02-25</code> 。
Service	即 Credential 中的 Service 字段信息。此示例取值为 <code>cvm</code> 。

2) 计算签名，伪代码如下：

```
Signature = HexEncode(HMAC_SHA256(SecretSigning, StringToSign))
```

4. 拼接 Authorization

按如下格式拼接 Authorization：

```
Authorization =
Algorithm + ' ' +
```

```
'Credential=' + SecretId + '/' + CredentialScope + ',' +  
'SignedHeaders=' + SignedHeaders + ',' +  
'Signature=' + Signature
```

字段名称	解释
Algorithm	签名方法，固定为 TC3-HMAC-SHA256。
SecretId	密钥对中的 SecretId，即 AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE。
CredentialScope	见上文，凭证范围。此示例计算结果是 2019-02-25/cvm/tc3_request。
SignedHeaders	见上文，参与签名的头部信息。此示例取值为 content-type;host。
Signature	签名值。此示例计算结果是 72e494ea809ad7a8c8f7a4507b9bddcbaa8e581f516e8da2f66e2c5a96525168。

根据以上规则，示例中得到的值为：

```
TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE/2019-02-25/cvm/tc3_request, SignedHeaders=content-type;host, Signature=72e494ea809ad7a8c8f7a4507b9bddcbaa8e581f516e8da2f66e2c5a96525168
```

最终完整的调用信息如下：

```
POST https://cvm.tencentcloudapi.com/  
Authorization: TC3-HMAC-SHA256 Credential=AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE/2019-02-25/cvm/tc3_request, SignedHeaders=content-type;host, Signature=72e494ea809ad7a8c8f7a4507b9bddcbaa8e581f516e8da2f66e2c5a96525168  
Content-Type: application/json; charset=utf-8  
Host: cvm.tencentcloudapi.com  
X-TC-Action: DescribeInstances  
X-TC-Version: 2017-03-12  
X-TC-Timestamp: 1551113065  
X-TC-Region: ap-guangzhou  
  
{ "Limit": 1, "Filters": [{"Values": ["\u672a\u547d\u540d"], "Name": "instance-name"}] }
```

5. 签名演示

Java

```
import java.nio.charset.Charset;  
import java.nio.charset.StandardCharsets;  
import java.security.MessageDigest;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
import java.util.TimeZone;  
import java.util.TreeMap;  
import javax.crypto.Mac;  
import javax.crypto.spec.SecretKeySpec;  
import javax.xml.bind.DatatypeConverter;  
  
public class TencentCloudAPITC3Demo {
```

```
private final static Charset UTF8 = StandardCharsets.UTF_8;
private final static String SECRET_ID = "AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE";
private final static String SECRET_KEY = "Gu5t9xGARNpq86cd98joQYCN3EXAMPLE";
private final static String CT_JSON = "application/json; charset=utf-8";

public static byte[] hmac256(byte[] key, String msg) throws Exception {
    Mac mac = Mac.getInstance("HmacSHA256");
    SecretKeySpec secretKeySpec = new SecretKeySpec(key, mac.getAlgorithm());
    mac.init(secretKeySpec);
    return mac.doFinal(msg.getBytes(UTF8));
}

public static String sha256Hex(String s) throws Exception {
    MessageDigest md = MessageDigest.getInstance("SHA-256");
    byte[] d = md.digest(s.getBytes(UTF8));
    return DatatypeConverter.printHexBinary(d).toLowerCase();
}

public static void main(String[] args) throws Exception {
    String service = "cvm";
    String host = "cvm.tencentcloudapi.com";
    String region = "ap-guangzhou";
    String action = "DescribeInstances";
    String version = "2017-03-12";
    String algorithm = "TC3-HMAC-SHA256";
    String timestamp = "1551113065";
    //String timestamp = String.valueOf(System.currentTimeMillis() / 1000);
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
    // 注意时区，否则容易出错
    sdf.setTimeZone(TimeZone.getTimeZone("UTC"));
    String date = sdf.format(new Date(Long.valueOf(timestamp + "000")));

    // ***** 步骤 1：拼接规范请求串 *****
    String httpRequestMethod = "POST";
    String canonicalUri = "/";
    String canonicalQueryString = "";
    String canonicalHeaders = "content-type:application/json; charset=utf-8\n" + "host:" + host + "\n";
    String signedHeaders = "content-type;host";

    String payload = "{\"Limit\": 1, \"Filters\": [{\"Values\": [\"\\u672a\\u547d\\u540d\"], \"Name\": \"instance-name\"}] }";
    String hashedRequestPayload = sha256Hex(payload);
    String canonicalRequest = httpRequestMethod + "\n" + canonicalUri + "\n" + canonicalQueryString + "\n"
        + canonicalHeaders + "\n" + signedHeaders + "\n" + hashedRequestPayload;
    System.out.println(canonicalRequest);

    // ***** 步骤 2：拼接待签名字符串 *****
    String credentialScope = date + "/" + service + "/" + "tc3_request";
    String hashedCanonicalRequest = sha256Hex(canonicalRequest);
    String stringToSign = algorithm + "\n" + timestamp + "\n" + credentialScope + "\n" + hashedCanonicalRequest;
    System.out.println(stringToSign);

    // ***** 步骤 3：计算签名 *****
    byte[] secretDate = hmac256(("TC3" + SECRET_KEY).getBytes(UTF8), date);
    byte[] secretService = hmac256(secretDate, service);
```

```
byte[] secretSigning = hmac256(secretService, "tc3_request");
String signature = DatatypeConverter.printHexBinary(hmac256(secretSigning, stringToSign)).toLowerCase();
System.out.println(signature);

// ***** 步骤 4 : 拼接 Authorization *****
String authorization = algorithm + " " + "Credential=" + SECRET_ID + "/" + credentialScope + ", "
+ "SignedHeaders=" + signedHeaders + ", " + "Signature=" + signature;
System.out.println(authorization);

TreeMap<String, String> headers = new TreeMap<String, String>();
headers.put("Authorization", authorization);
headers.put("Content-Type", CT_JSON);
headers.put("Host", host);
headers.put("X-TC-Action", action);
headers.put("X-TC-Timestamp", timestamp);
headers.put("X-TC-Version", version);
headers.put("X-TC-Region", region);

StringBuilder sb = new StringBuilder();
sb.append("curl -X POST https://").append(host)
.append(" -H \"Authorization: ").append(authorization).append("\")")
.append(" -H \"Content-Type: application/json; charset=utf-8\"")
.append(" -H \"Host: ").append(host).append("\")")
.append(" -H \"X-TC-Action: ").append(action).append("\")")
.append(" -H \"X-TC-Timestamp: ").append(timestamp).append("\")")
.append(" -H \"X-TC-Version: ").append(version).append("\")")
.append(" -H \"X-TC-Region: ").append(region).append("\")")
.append(" -d ").append(payload).append("");
System.out.println(sb.toString());
}
}
```

Python

```
# -*- coding: utf-8 -*-
import hashlib, hmac, json, os, sys, time
from datetime import datetime

# 密钥参数
secret_id = "AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE"
secret_key = "Gu5t9xGARNpq86cd98joQYCN3EXAMPLE"

service = "cvm"
host = "cvm.tencentcloudapi.com"
endpoint = "https://" + host
region = "ap-guangzhou"
action = "DescribeInstances"
version = "2017-03-12"
algorithm = "TC3-HMAC-SHA256"
#timestamp = int(time.time())
timestamp = 1551113065
date = datetime.utcfromtimestamp(timestamp).strftime("%Y-%m-%d")
params = {"Limit": 1, "Filters": [{"Name": "instance-name", "Values": [u"未命名"]}]}
```

```
# ***** 步骤 1：拼接规范请求串 *****
http_request_method = "POST"
canonical_uri = "/"
canonical_querystring = ""
ct = "application/json; charset=utf-8"
payload = json.dumps(params)
canonical_headers = "content-type:%s\nhost:%s\n" % (ct, host)
signed_headers = "content-type;host"
hashed_request_payload = hashlib.sha256(payload.encode("utf-8")).hexdigest()
canonical_request = (http_request_method + "\n" +
canonical_uri + "\n" +
canonical_querystring + "\n" +
canonical_headers + "\n" +
signed_headers + "\n" +
hashed_request_payload)
print(canonical_request)

# ***** 步骤 2：拼接待签名字符串 *****
credential_scope = date + "/" + service + "/" + "tc3_request"
hashed_canonical_request = hashlib.sha256(canonical_request.encode("utf-8")).hexdigest()
string_to_sign = (algorithm + "\n" +
str(timestamp) + "\n" +
credential_scope + "\n" +
hashed_canonical_request)
print(string_to_sign)

# ***** 步骤 3：计算签名 *****
# 计算签名摘要函数
def sign(key, msg):
return hmac.new(key, msg.encode("utf-8"), hashlib.sha256).digest()
secret_date = sign(("TC3" + secret_key).encode("utf-8"), date)
secret_service = sign(secret_date, service)
secret_signing = sign(secret_service, "tc3_request")
signature = hmac.new(secret_signing, string_to_sign.encode("utf-8"), hashlib.sha256).hexdigest()
print(signature)

# ***** 步骤 4：拼接 Authorization *****
authorization = (algorithm + " " +
"Credential=" + secret_id + "/" + credential_scope + ", " +
"SignedHeaders=" + signed_headers + ", " +
"Signature=" + signature)
print(authorization)

print('curl -X POST ' + endpoint
+ ' -H "Authorization: ' + authorization + '"
+ ' -H "Content-Type: application/json; charset=utf-8"
+ ' -H "Host: ' + host + '"
+ ' -H "X-TC-Action: ' + action + '"
+ ' -H "X-TC-Timestamp: ' + str(timestamp) + '"
+ ' -H "X-TC-Version: ' + version + '"
+ ' -H "X-TC-Region: ' + region + '"
+ " -d '" + payload + "'")
```

签名失败

存在以下签名失败的错误码，请根据实际情况处理。

错误码	错误描述
AuthFailure.SignatureExpire	签名过期。Timestamp 与服务器接收到请求的时间相差不得超过五分钟。
AuthFailure.SecretIdNotFound	密钥不存在。请到控制台查看密钥是否被禁用，是否少复制了字符或者多了字符。
AuthFailure.SignatureFailure	签名错误。可能是签名计算错误，或者签名与实际发送的内容不符合，也有可能是密钥 SecretKey 错误导致的。
AuthFailure.TokenFailure	临时证书 Token 错误。
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）。

接口鉴权

最近更新时间：2019-08-14 19:18:17

腾讯云 API 会对每个访问请求进行身份验证，即每个请求都需要在公共请求参数中包含签名信息（Signature）以验证请求者身份。签名信息由安全凭证生成，安全凭证包括 SecretId 和 SecretKey；若用户还没有安全凭证，请前往 [云API密钥页面](#) 申请，否则无法调用云 API 接口。

1. 申请安全凭证

在第一次使用云 API 之前，请前往 [云 API 密钥页面](#) 申请安全凭证。安全凭证包括 SecretId 和 SecretKey：

- SecretId 用于标识 API 调用者身份
- SecretKey 用于加密签名字符串和服务器端验证签名字符串的密钥。
- **用户必须严格保管安全凭证，避免泄露。**

申请安全凭证的具体步骤如下：

1. 登录 [腾讯云管理中心控制台](#)。
2. 前往 [云 API 密钥](#) 的控制台页面
3. 在 [云 API 密钥](#) 页面，单击【新建】即可以创建一对 SecretId/SecretKey。

注意：每个账号最多可以拥有两对 SecretId/SecretKey。

2. 生成签名串

有了安全凭证 SecretId 和 SecretKey 后，就可以生成签名串了。以下是生成签名串的详细过程：

假设用户的 SecretId 和 SecretKey 分别是：

- SecretId: AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE
- SecretKey: Gu5t9xGARNpq86cd98joQYCN3EXAMPLE

注意：这里只是示例，请根据用户实际申请的 SecretId 和 SecretKey 进行后续操作！

以云服务器查看实例列表(DescribeInstances)请求为例，当用户调用这一接口时，其请求参数可能如下：

参数名称	中文	参数值
Action	方法名	DescribeInstances
SecretId	密钥 ID	AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE
Timestamp	当前时间戳	1465185768
Nonce	随机正整数	11886
Region	实例所在区域	ap-guangzhou
InstanceId.0	待查询的实例 ID	ins-09dx96dg

参数名称	中文	参数值
Offset	偏移量	0
Limit	最大允许输出	20
Version	接口版本号	2017-03-12

2.1. 对参数排序

首先对所有请求参数按参数名的字典序（ASCII 码）升序排序。注意：1）只按参数名进行排序，参数值保持对应即可，不参与比大小；2）按 ASCII 码比大小，如 InstanceIds.2 要排在 InstanceIds.12 后面，不是按字母表，也不是按数值。用户可以借助编程语言中的相关排序函数来实现这一功能，如 PHP 中的 ksort 函数。上述示例参数的排序结果如下：

```
{
  'Action': 'DescribeInstances',
  'InstanceIds.0': 'ins-09dx96dg',
  'Limit': 20,
  'Nonce': 11886,
  'Offset': 0,
  'Region': 'ap-guangzhou',
  'SecretId': 'AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE',
  'Timestamp': 1465185768,
  'Version': '2017-03-12',
}
```

使用其它程序设计语言开发时，可对上面示例中的参数进行排序，得到的结果一致即可。

2.2. 拼接请求字符串

此步骤生成请求字符串。将把上一步排序好的请求参数格式化成“参数名称=参数值”的形式，如对 Action 参数，其参数名称为 "Action"，参数值为 "DescribeInstances"，因此格式化后就为 Action=DescribeInstances。注意：“参数值”为原始值而非 url 编码后的值。

然后将格式化后的各个参数用 "&" 拼接在一起，最终生成的请求字符串为：

```
Action=DescribeInstances&InstanceIds.0=ins-09dx96dg&Limit=20&Nonce=11886&Offset=0&Region=ap-guangzhou&SecretId=AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE&Timestamp=1465185768&Version=2017-03-12
```

2.3. 拼接签名原字符串

此步骤生成签名原字符串。签名原字符串由以下几个参数构成：

1. 请求方法: 支持 POST 和 GET 方式，这里使用 GET 请求，注意方法为全大写。
2. 请求主机: 查看实例列表(DescribeInstances)的请求域名为：cvm.tencentcloudapi.com。实际的请求域名根据接口所属模块的不同而不同，详见各接口说明。
3. 请求路径: 当前版本云API的请求路径固定为 /。
4. 请求字符串: 即上一步生成的请求字符串。

签名原文串的拼接规则为：请求方法 + 请求主机 + 请求路径 + ? + 请求字符串。

示例的拼接结果为：

```
GETcvm.tencentcloudapi.com/?Action=DescribeInstances&InstanceId=ins-09dx96dg&Limit=20&Nonce=11886&Offset=0&Region=ap-guangzhou&SecretId=AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE&Timestamp=1465185768&Version=2017-03-12
```

2.4. 生成签名串

此步骤生成签名串。首先使用 HMAC-SHA1 算法对上一步中获得的**签名原文字符串**进行签名，然后将生成的签名串使用 Base64 进行编码，即可获得最终的签名串。

具体代码如下，以 PHP 语言为例：

```
$secretKey = 'Gu5t9xGARNpq86cd98joQYCN3EXAMPLE';
$srcStr = 'GETcvm.tencentcloudapi.com/?Action=DescribeInstances&InstanceId=ins-09dx96dg&Limit=20&Nonce=11886&Offset=0&Region=ap-guangzhou&SecretId=AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE&Timestamp=1465185768&Version=2017-03-12';
$signStr = base64_encode(hash_hmac('sha1', $srcStr, $secretKey, true));
echo $signStr;
```

最终得到的签名串为：

```
EliP9YW3pW28FpsEdkXt/+WcGel=
```

使用其它程序设计语言开发时，可用上面示例中的原文进行签名验证，得到的签名串与例子中的一致即可。

3. 签名串编码

生成的签名串并不能直接作为请求参数，需要对其进行 URL 编码。

如上一步生成的签名串为 EliP9YW3pW28FpsEdkXt/+WcGel=，最终得到的签名串请求参数（Signature）为：EliP9YW3pW28FpsEdkXt/+WcGel=，它将用于生成最终的请求 URL。

注意：如果用户的请求方法是 GET，或者请求方法为 POST 同时 Content-Type 为 application/x-www-form-urlencoded，则发送请求时所有请求参数的值均需要做 URL 编码，参数键和=符号不需要编码。非 ASCII 字符在 URL 编码前需要先用 UTF-8 进行编码。

注意：有些编程语言的网路库会自动为所有参数进行 urlencode，在这种情况下，就不需要对签名串进行 URL 编码了，否则两次 URL 编码会导致签名失败。

注意：其他参数值也需要进行编码，编码采用 RFC 3986。使用 %XY 对特殊字符例如汉字进行百分比编码，其中“X”和“Y”为十六进制字符（0-9 和大写字母 A-F），使用小写将引发错误。

4. 签名失败

根据实际情况，存在以下签名失败的错误码，请根据实际情况处理。

错误代码	错误描述
AuthFailure.SignatureExpire	签名过期
AuthFailure.SecretIdNotFound	密钥不存在

错误代码	错误描述
AuthFailure.SignatureFailure	签名错误
AuthFailure.TokenFailure	token 错误
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）

5. 签名演示

在实际调用 API 3.0 时，推荐使用配套的腾讯云 SDK 3.0，SDK 封装了签名的过程，开发时只关注产品提供的具体接口即可。详细信息参见 [SDK 中心](#)。当前支持的编程语言有：

- [Python](#)
- [Java](#)
- [PHP](#)
- [Go](#)
- [JavaScript](#)
- [.NET](#)

为了更清楚的解释签名过程，下面以实际编程语言为例，将上述的签名过程具体实现。请求的域名、调用的接口和参数的取值都以上述签名过程为准，代码只为解释签名过程，并不具备通用性，实际开发请尽量使用 SDK。

最终输出的 url 可能为：`https://cvm.tencentcloudapi.com/?Action=DescribeInstances&InstanceId=ins-09dx96dg&Limit=20&Nonce=11886&Offset=0&Region=ap-guangzhou&SecretId=AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE&Signature=Elip9YW3pW28FpsEdkXt/+WcGel=&Timestamp=1465185768&Version=2017-03-12`。

注意：由于示例中的密钥是虚构的，时间戳也不是系统当前时间，因此如果将此 url 在浏览器中打开或者用 curl 等命令调用时会返回鉴权错误：签名过期。为了得到一个可以正常返回的 url，需要修改示例中的 SecretId 和 SecretKey 为真实的密钥，并使用系统当前时间戳作为 Timestamp。

注意：在下面的示例中，不同编程语言，甚至同一语言每次执行得到的 url 可能都有所不同，表现为参数的顺序不同，但这并不影响正确性。只要所有参数都在，且签名计算正确即可。

注意：以下代码仅适用于 API 3.0，不能直接用于其他的签名流程，即使是旧版的 API，由于存在细节差异也会导致签名计算错误，请以对应的实际文档为准。

Java

```
import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;
import java.util.Random;
import java.util.TreeMap;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import javax.xml.bind.DatatypeConverter;

public class TencentCloudAPIDemo {
    private final static String CHARSET = "UTF-8";
```

```
public static String sign(String s, String key, String method) throws Exception {
    Mac mac = Mac.getInstance(method);
    SecretKeySpec secretKeySpec = new SecretKeySpec(key.getBytes(CHARSET), mac.getAlgorithm());
    mac.init(secretKeySpec);
    byte[] hash = mac.doFinal(s.getBytes(CHARSET));
    return DatatypeConverter.printBase64Binary(hash);
}

public static String getStringToSign(TreeMap<String, Object> params) {
    StringBuilder s2s = new StringBuilder("GETcvm.tencentcloudapi.com/?");
    // 签名时要求对参数进行字典排序，此处用TreeMap保证顺序
    for (String k : params.keySet()) {
        s2s.append(k).append("=").append(params.get(k).toString()).append("&");
    }
    return s2s.toString().substring(0, s2s.length() - 1);
}

public static String getUrl(TreeMap<String, Object> params) throws UnsupportedOperationException {
    StringBuilder url = new StringBuilder("https://cvm.tencentcloudapi.com/?");
    // 实际请求的url中对参数顺序没有要求
    for (String k : params.keySet()) {
        // 需要对请求串进行urlencode，由于key都是英文字母，故此处仅对其value进行urlencode
        url.append(k).append("=").append(URLEncoder.encode(params.get(k).toString(), CHARSET)).append("&");
    }
    return url.toString().substring(0, url.length() - 1);
}

public static void main(String[] args) throws Exception {
    TreeMap<String, Object> params = new TreeMap<String, Object>(); // TreeMap可以自动排序
    // 实际调用时应当使用随机数，例如：params.put("Nonce", new Random().nextInt(java.lang.Integer.MAX_VALUE));
    params.put("Nonce", 11886); // 公共参数
    // 实际调用时应当使用系统当前时间，例如：params.put("Timestamp", System.currentTimeMillis() / 1000);
    params.put("Timestamp", 1465185768); // 公共参数
    params.put("SecretId", "AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE"); // 公共参数
    params.put("Action", "DescribeInstances"); // 公共参数
    params.put("Version", "2017-03-12"); // 公共参数
    params.put("Region", "ap-guangzhou"); // 公共参数
    params.put("Limit", 20); // 业务参数
    params.put("Offset", 0); // 业务参数
    params.put("InstanceId.0", "ins-09dx96dg"); // 业务参数
    params.put("Signature", sign(getStringToSign(params), "Gu5t9xGARNpq86cd98joQYCN3EXAMPLE", "HmacSHA1")); // 公共参数
    System.out.println(getUrl(params));
}
}
```

Python

注意：如果是在 Python 2 环境中运行，需要先安装 requests 依赖包：`pip install requests`。

```
# -*- coding: utf8 -*-
import base64
```

```
import hashlib
import hmac
import time

import requests

secret_id = "AKIDz8krbsJ5yKBZQpn74WFkmLPx3EXAMPLE"
secret_key = "Gu5t9xGARNpq86cd98joQYCN3EXAMPLE"

def get_string_to_sign(method, endpoint, params):
    s = method + endpoint + "/"
    query_str = "&".join("%s=%s" % (k, params[k]) for k in sorted(params))
    return s + query_str

def sign_str(key, s, method):
    hmac_str = hmac.new(key.encode("utf8"), s.encode("utf8"), method).digest()
    return base64.b64encode(hmac_str)

if __name__ == '__main__':
    endpoint = "cvm.tencentcloudapi.com"
    data = {
        'Action': 'DescribeInstances',
        'InstanceId.0': 'ins-09dx96dg',
        'Limit': 20,
        'Nonce': 11886,
        'Offset': 0,
        'Region': 'ap-guangzhou',
        'SecretId': secret_id,
        'Timestamp': 1465185768, # int(time.time())
        'Version': '2017-03-12'
    }
    s = get_string_to_sign("GET", endpoint, data)
    data["Signature"] = sign_str(secret_key, s, hashlib.sha1)
    print(data["Signature"])
    # 此处会实际调用，成功后可能产生计费
    # resp = requests.get("https://" + endpoint, params=data)
    # print(resp.url)
```

返回结果

最近更新时间：2019-08-14 19:18:18

正确返回结果

以云服务器的接口查看实例状态列表 (DescribeInstancesStatus) 2017-03-12 版本为例，若调用成功，其可能的返回如下为：

```
{
  "Response": {
    "TotalCount": 0,
    "InstanceStatusSet": [],
    "RequestId": "b5b41468-520d-4192-b42f-595cc34b6c1c"
  }
}
```

- Response 及其内部的 RequestId 是固定的字段，无论请求成功与否，只要 API 处理了，则必定会返回。
- RequestId 用于一个 API 请求的唯一标识，如果 API 出现异常，可以联系我们，并提供该 ID 来解决问题。
- 除了固定的字段外，其余均为具体接口定义的字段，不同的接口所返回的字段参见接口文档中的定义。此例中的 TotalCount 和 InstanceStatusSet 均为 DescribeInstancesStatus 接口定义的字段，由于调用请求的用户暂时还没有云服务器实例，因此 TotalCount 在此情况下的返回值为 0，InstanceStatusSet 列表为空。

错误返回结果

若调用失败，其返回值示例如下为：

```
{
  "Response": {
    "Error": {
      "Code": "AuthFailure.SignatureFailure",
      "Message": "The provided credentials could not be validated. Please check your signature is correct."
    },
    "RequestId": "ed93f3cb-f35e-473f-b9f3-0d451b8b79c6"
  }
}
```

- Error 的出现代表着该请求调用失败。Error 字段连同其内部的 Code 和 Message 字段在调用失败时是必定返回的。
- Code 表示具体出错的错误码，当请求出错时可以先根据该错误码在公共错误码和当前接口对应的错误码列表里面查找对应原因和解决方案。
- Message 显示出了这个错误发生的具体原因，随着业务发展或体验优化，此文本可能会经常保持变更或更新，用户不应依赖这个返回值。
- RequestId 用于一个 API 请求的唯一标识，如果 API 出现异常，可以联系我们，并提供该 ID 来解决问题。

公共错误码

返回结果中如果存在 Error 字段，则表示调用 API 接口失败。Error 中的 Code 字段表示错误码，所有业务都可能出现的错误码为公共错误码，下表列出了公共错误码。

错误码	错误描述
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）。
AuthFailure.MFAFailure	MFA 错误。
AuthFailure.SecretIdNotFound	密钥不存在。
AuthFailure.SignatureExpire	签名过期。
AuthFailure.SignatureFailure	签名错误。
AuthFailure.TokenFailure	token 错误。
AuthFailure.UnauthorizedOperation	请求未 CAM 授权。
DryRunOperation	DryRun 操作，代表请求将会是成功的，只是多传了 DryRun 参数。
FailedOperation	操作失败。
InternalError	内部错误。
InvalidAction	接口不存在。
InvalidParameter	参数错误。
InvalidParameterValue	参数取值错误。
LimitExceeded	超过配额限制。
MissingParameter	缺少参数错误。
NoSuchVersion	接口版本不存在。
RequestLimitExceeded	请求的次数超过了频率限制。
ResourceInUse	资源被占用。
ResourceInsufficient	资源不足。
ResourceNotFound	资源不存在。
ResourceUnavailable	资源不可用。
UnauthorizedOperation	未授权操作。
UnknownParameter	未知参数错误。
UnsupportedOperation	操作不支持。
UnsupportedProtocol	HTTPS 请求方法错误，只支持 GET 和 POST 请求。
UnsupportedRegion	接口不支持所传地域。

图片内容审核相关接口

图像内容审核

最近更新时间：2019-07-24 15:39:47

1. 接口描述

接口请求域名：tictm.tencentcloudapi.com。

本接口提供多种维度的图像审核能力，支持色情和性感内容识别，政治人物和涉政敏感场景识别，以及暴恐人物、场景、旗帜标识等违禁内容的识别。

默认接口请求频率限制：50次/秒。

2. 输入参数

以下请求参数列表仅列出了接口请求参数和部分公共参数，完整公共参数列表见 [公共请求参数](#)。

参数名称	必选	类型	描述
Action	是	String	公共参数，本接口取值：ImageModeration
Version	是	String	公共参数，本接口取值：2018-11-27
Region	是	String	公共参数，详见产品支持的 地域列表 。
Scenes.N	是	Array of String	本次调用支持的识别场景，可选值如下： 1. PORN，即色情识别 2. TERRORISM，即暴恐识别 3. POLITICS，即政治敏感识别 支持多场景（Scenes）一起检测。例如，使用 Scenes=["PORN", "TERRORISM"]，即对一张图片同时进行色情识别和暴恐识别。
ImageUrl	否	String	图片URL地址。 图片限制： • 图片格式：PNG、JPG、JPEG。 • 图片大小：所下载图片经Base64编码后不超过4M。图片下载时间不超过3秒。 • 图片像素：大于50*50像素，否则影响识别效果； • 长宽比：长边：短边<5； 接口响应时间会受到图片下载时间的影响，建议使用更可靠的存储服务，推荐将图片存储在腾讯云COS。
Config	否	String	预留字段，后期用于展示更多识别信息。
Extra	否	String	透传字段，透传简单信息。
ImageBase64	否	String	图片经过base64编码的内容。最大不超过4M。与ImageUrl同时存在时优先使用ImageUrl字段。

3. 输出参数

参数名称	类型	描述
Suggestion	String	识别场景的审核结论： PASS：正常 REVIEW：疑似 BLOCK：违规
PornResult	PornResult	色情识别结果。 注意：此字段可能返回 null，表示取不到有效值。
TerrorismResult	TerrorismResult	暴恐识别结果。 注意：此字段可能返回 null，表示取不到有效值。
PoliticsResult	PoliticsResult	政治敏感识别结果。 注意：此字段可能返回 null，表示取不到有效值。
Extra	String	透传字段，透传简单信息。
DisgustResult	DisgustResult	恶心内容识别结果。 注意：此字段可能返回 null，表示取不到有效值。
RequestId	String	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。

4. 示例

示例1 多个识别场景调用都成功

例：同时调用色情识别、暴恐识别和政治敏感识别，返回结果均为成功。

输入示例

```
https://ticm.tencentcloudapi.com/?Action=ImageModeration
&ImageUrl=https://xx/a.jpg
&Config= ""
&Scenes.0=PORN
&Scenes.1=POLITICS
&Scenes.2=TERRORISM
&<公共请求参数>
```

输出示例

```
{
  "Response": {
    "Suggestion": "REVIEW",
    "PornResult": {
      "Code": 0,
      "Msg": "OK",
      "Suggestion": "PASS",
      "Confidence": 83,
      "AdvancedInfo": "",

```

```
"Type": "LABEL"
},
"PoliticsResult": {
  "Code": 0,
  "Msg": "OK",
  "Suggestion": "PASS",
  "Confidence": 8,
  "FaceResults": [
    {
      "FaceRect": {
        "Height": 221,
        "Width": 221,
        "X": 98,
        "Y": 115
      },
      "Candidates": [
        {
          "Name": "本拉登",
          "Confidence": 8
        }
      ]
    }
  ],
  "Type": "FACE"
},
"TerrorismResult": {
  "Code": 0,
  "Msg": "OK",
  "Suggestion": "REVIEW",
  "Confidence": 42,
  "AdvancedInfo": "",
  "FaceResults": [],
  "Type": "LABEL"
},
"Extra": "",
"RequestId": "02f1733c-1bfe-49ed-9e72-8ecd6ba058dd"
}
```

示例2 部分识别场景调用都成功

例：只调用其中的色情识别，返回结果为成功。

输入示例

```
https://ticm.tencentcloudapi.com/?Action=ImageModeration
&ImageUrl=https://porn.jpg
&Config= ""
&Scenes.0=PORN
&<公共请求参数>
```

输出示例

```
{
  "Response": {
    "Suggestion": "PASS",
    "PornResult": {
      "Code": 0,
      "Msg": "OK",
      "Suggestion": "PASS",
      "Confidence": 0,
      "AdvancedInfo": "",
      "Type": "LABEL"
    },
    "PoliticsResult": null,
    "TerrorismResult": null,
    "Extra": "",
    "RequestId": "5a645a08-2e2b-4979-9815-1f2c00c2304e"
  }
}
```

示例3 部分识别场景调用失败

例：只调用其中的色情识别，返回结果为失败，Code 不为0。

输入示例

```
https://ticm.tencentcloudapi.com/?Action=ImageModeration
&ImageUrl=https://porn.jpg
&Config= ""
&Scenes.0=PORN
&<公共请求参数>
```

输出示例

```
{
  "Response": {
    "Suggestion": "",
    "PornResult": {
      "Code": -1,
      "Msg": "内部错误",
      "Suggestion": "",
      "Confidence": 0,
      "AdvancedInfo": "",
      "Type": "LABEL"
    },
    "PoliticsResult": null,
    "TerrorismResult": null,
    "Extra": "",
    "RequestId": "3854392b-19d4-4cae-8933-b010cd087c84"
  }
}
```

示例4 部分识别场景调用且部分成功

例：同时调用色情识别和暴恐识别，其中色情识别的结果为失败（Code非0），暴恐识别的结果为成功。

输入示例

```
https://ticm.tencentcloudapi.com/?Action=ImageModeration
&ImageUrl=https://fake.jpg
&Config= ""
&Scenes.0=PORN
&Scenes.0=TERRORISM
&<公共请求参数>
```

输出示例

```
{
  "Response": {
    "Suggestion": "REVIEW",
    "PornResult": {
      "Code": -1,
      "Msg": "内部错误",
      "Suggestion": "",
      "Confidence": 0,
      "AdvancedInfo": "",
      "Type": "LABEL"
    },
    "TerrorismResult": {
      "Code": 0,
      "Msg": "OK",
      "Suggestion": "REVIEW",
      "Confidence": 44,
      "AdvancedInfo": "",
      "Type": "LABEL",
      "FaceResults": []
    },
    "PoliticsResult": null,
    "Extra": "",
    "RequestId": "547d2427-2f82-4d8d-99e0-f2a504619661"
  }
}
```

示例5 整体结果失败

例：调用部分识别场景或者全部场景，由于下载等原因，导致整体结果失败。

输入示例

```
https://ticm.tencentcloudapi.com/?Action=ImageModeration
&ImageUrl=https://error.jpg
&Config= ""
&Scenes.0=PORN
&Scenes.1=POLITICS
&Scenes.2=TERRORISM
&<公共请求参数>
```

输出示例

```
{
  "Response": {
    "Error": {
      "Code": "FailedOperation.DownloadError",
      "Message": "下载失败"
    },
    "RequestId": "5f77ef5e-f381-49af-89bf-e558b25b3c15"
  }
}
```

5. 开发者资源

API Explorer

该工具提供了在线调用、签名验证、SDK 代码生成和快速检索接口等能力，能显著降低使用云 API 的难度，推荐使用。

- [API 3.0 Explorer](#)

SDK

云 API 3.0 提供了配套的开发工具集（SDK），支持多种编程语言，能更方便的调用 API。

- [Tencent Cloud SDK 3.0 for Python](#)
- [Tencent Cloud SDK 3.0 for Java](#)
- [Tencent Cloud SDK 3.0 for PHP](#)
- [Tencent Cloud SDK 3.0 for Go](#)
- [Tencent Cloud SDK 3.0 for NodeJS](#)
- [Tencent Cloud SDK 3.0 for .NET](#)

命令行工具

- [Tencent Cloud CLI 3.0](#)

6. 错误码

以下仅列出了接口业务逻辑相关的错误码，其他错误码详见 [公共错误码](#)。

错误码	描述
FailedOperation.DownloadError	文件下载失败。
FailedOperation.InvokeChargeError	调用计费返回失败。
FailedOperation.UnKnowError	未知错误。
FailedOperation.UnOpenError	服务未开通。
InvalidParameterValue.InvalidParameterValueLimit	参数值错误。

错误码	描述
LimitExceeded.TooLargeFileError	文件内容太大。
ResourcesSoldOut.ChargeStatusException	计费状态异常。

数据结构

最近更新时间：2019-04-26 15:01:36

Candidate

识别出人脸对应的候选人。

被如下接口引用：ImageModeration。

名称	类型	描述
Name	String	识别出人脸对应的候选人数组。当前返回相似度最高的候选人。
Confidence	Integer	相似度，0-100之间。

DisgustResult

恶心识别结果。

被如下接口引用：ImageModeration。

名称	类型	描述
Code	Integer	该识别场景的错误码： 0表示成功， -1表示系统错误， -2表示引擎错误。
Msg	String	错误码描述信息。
Suggestion	String	识别场景的审核结论： PASS：正常 REVIEW：疑似 BLOCK：违规
Confidence	Integer	图像恶心的分数，0-100之间，分数越高恶心几率越大。

FaceRect

识别出的人脸在图片中的位置。

被如下接口引用：ImageModeration。

名称	类型	描述
X	Integer	人脸区域左上角横坐标。
Y	Integer	人脸区域左上角纵坐标。

名称	类型	描述
Width	Integer	人脸区域宽度。
Height	Integer	人脸区域高度。

FaceResult

人脸识别结果。

被如下接口引用：ImageModeration。

名称	类型	描述
FaceRect	FaceRect	检测出的人脸框位置。
Candidates	Array of Candidate	候选人列表。当前返回相似度最高的候选人。

PoliticsResult

政治敏感识别结果。

被如下接口引用：ImageModeration。

名称	类型	描述
Code	Integer	该识别场景的错误码： 0表示成功， -1表示系统错误， -2表示引擎错误， -1400表示图片解码失败， -1401表示图片不符合规范。
Msg	String	错误码描述信息。
Suggestion	String	识别场景的审核结论： PASS：正常 REVIEW：疑似 BLOCK：违规
Confidence	Integer	图像涉政的分数，0-100之间，分数越高涉政几率越大。 Type为DNA时： 0到75，Suggestion建议为PASS 75到90，Suggestion建议为REVIEW 90到100，Suggestion建议为BLOCK Type为FACE时： 0到55，Suggestion建议为PASS 55到60，Suggestion建议为REVIEW 60到100，Suggestion建议为BLOCK

名称	类型	描述
FaceResults	Array of FaceResult	Type取值为'FACE'时，人脸识别的结果列表。基于图片中实际检测到的人脸数，返回数组最大值不超过5个。
Type	String	取值'DNA' 或'FACE'。DNA表示结论和置信度来自图像指纹，FACE表示结论和置信度来自人脸识别。
AdvancedInfo	String	鉴政识别返回的详细标签后期开放。

PornResult

色情识别结果。

被如下接口引用：ImageModeration。

名称	类型	描述
Code	Integer	该识别场景的错误码： 0表示成功， -1表示系统错误， -2表示引擎错误， -1400表示图片解码失败。
Msg	String	错误码描述信息。
Suggestion	String	识别场景的审核结论： PASS：正常 REVIEW：疑似 BLOCK：违规
Confidence	Integer	算法对于Suggestion的置信度，0-100之间，值越高，表示对于Suggestion越确定。
AdvancedInfo	String	预留字段，后期用于展示更多识别信息。
Type	String	取值'LABEL'，LABEL表示结论和置信度来自标签分类。

TerrorismResult

暴恐识别结果。

被如下接口引用：ImageModeration。

名称	类型	描述
Code	Integer	该识别场景的错误码： 0表示成功， -1表示系统错误， -2表示引擎错误， -1400表示图片解码失败。
Msg	String	错误码描述信息。

名称	类型	描述
Suggestion	String	识别场景的审核结论： PASS：正常 REVIEW：疑似 BLOCK：违规
Confidence	Integer	图像涉恐的分数，0-100之间，分数越高涉恐几率越大。 Type为LABEL时： 0到86，Suggestion建议为PASS 86到91，Suggestion建议为REVIEW 91到100，Suggestion建议为BLOCK Type为FACE时： 0到55，Suggestion建议为PASS 55到60，Suggestion建议为REVIEW 60到100，Suggestion建议为BLOCK
FaceResults	Array of FaceResult	Type取值为'FACE'时，人脸识别的结果列表。基于图片中实际检测到的人脸数，返回数组最大值不超过5个。
AdvancedInfo	String	暴恐识别返回的详细标签后期开放。
Type	String	取值'LABEL' 或'FACE'，LABEL表示结论和置信度来自标签分类，FACE表示结论和置信度来自人脸识别。

错误码

最近更新时间：2019-07-24 15:39:47

功能说明

如果返回结果中存在 Error 字段，则表示调用 API 接口失败。例如：

```
{
  "Response": {
    "Error": {
      "Code": "AuthFailure.SignatureFailure",
      "Message": "The provided credentials could not be validated. Please check your signature is correct."
    },
    "RequestId": "ed93f3cb-f35e-473f-b9f3-0d451b8b79c6"
  }
}
```

Error 中的 Code 表示错误码，Message 表示该错误的具体信息。

错误码列表

公共错误码

错误码	说明
AuthFailure.InvalidSecretId	密钥非法（不是云 API 密钥类型）。
AuthFailure.MFAFailure	MFA 错误。
AuthFailure.SecretIdNotFound	密钥不存在。请在控制台检查密钥是否已被删除或者禁用，如状态正常，请检查密钥是否填写正确，注意前后不得有空格。
AuthFailure.SignatureExpire	签名过期。Timestamp 和服务器时间相差不得超过五分钟，请检查本地时间是否和标准时间同步。
AuthFailure.SignatureFailure	签名错误。签名计算错误，请对照调用方式中的接口鉴权文档检查签名计算过程。
AuthFailure.TokenFailure	token 错误。
AuthFailure.UnauthorizedOperation	请求未授权。请参考 CAM 文档对鉴权的说明。
DryRunOperation	DryRun 操作，代表请求将会是成功的，只是多传了 DryRun 参数。
FailedOperation	操作失败。
InternalError	内部错误。
InvalidAction	接口不存在。
InvalidParameter	参数错误。

错误码	说明
InvalidParameterValue	参数取值错误。
LimitExceeded	超过配额限制。
MissingParameter	缺少参数错误。
NoSuchVersion	接口版本不存在。
RequestLimitExceeded	请求的次数超过了频率限制。
ResourceInUse	资源被占用。
ResourceInsufficient	资源不足。
ResourceNotFound	资源不存在。
ResourceUnavailable	资源不可用。
UnauthorizedOperation	未授权操作。
UnknownParameter	未知参数错误。
UnsupportedOperation	操作不支持。
UnsupportedProtocol	HTTP(S)请求协议错误，只支持 GET 和 POST 请求。
UnsupportedRegion	接口不支持所传地域。

业务错误码

错误码	说明
FailedOperation.DownloadError	文件下载失败。
FailedOperation.InvokeChargeError	调用计费返回失败。
FailedOperation.UnKnowError	未知错误。
FailedOperation.UnOpenError	服务未开通。
InvalidParameterValue.InvalidParameterValueLimit	参数值错误。
LimitExceeded.TooLargeFileError	文件内容太大。
ResourcesSoldOut.ChargeStatusException	计费状态异常。

智能鉴黄 API 2017

智能鉴黄 API

最近更新时间：2019-04-26 10:05:59

接口描述

接口请求域名：https://recognition.image.myqcloud.com/detection/porn_detect

本接口（porn_detect）用于对用户上传的图片进行鉴黄处理。

说明：

- 本接口支持 HTTPS 协议，如果您现在使用的是 HTTP 协议，为了保障您的数据安全，请切换至 HTTPS。
- 开发者无需注册账号，即可在 [用户体验平台](#) 体验智能鉴黄效果。

请求头 header

所有请求都要求含有以下头部信息：

参数	必选	值	描述
host	是	recognition.image.myqcloud.com	腾讯云图片鉴黄服务器域名
content-length	否	包体总长度	整个请求包体内容的总长度，单位：字节（byte）
content-type	是	application/json 或 multipart/form-data	根据不同接口选择，每个请求最多支持 20 张 url 或图片： 1. 使用图片 url，选择 application/json； 2. 使用图片文件，选择 multipart/form-data。
authorization	是	鉴权签名	用于 鉴权 的签名

注意：

如选择 multipart/form-data，请使用 HTTP 框架/库推荐的方式设置请求的 contenttype，不推荐直接调用 setheader 等方法设置，否则可能导致 boundary 缺失引起请求失败。

使用图片 url

输入参数

使用 application/json 格式：

参数	必选	类型	说明
----	----	----	----

参数	必选	类型	说明
appid	是	String	接入项目的唯一标识，可在 账号信息 或 云 API 密钥 中查看。
url_list	是	String 数组	图片 url 列表

输出参数

接口返回的 result_list 为 JSON 数组，数组的每个元素如下：

字段	类型	说明
code	Int	错误码，0为成功。
message	String	服务器返回的信息。
url	String	当前图片的 url。
data	Object	具体查询数据，具体见下表。

data 字段具体内容：

字段	类型	说明
result	Int	供参考的识别结果：0为正常，1为黄图，2为疑似图片。
confidence	Double	识别为黄图的置信度，分值 0-100；是 normal_score、hot_score、porn_score 的综合评分。
normal_score	Double	图片为正常图片的评分。
hot_score	Double	图片为性感图片的评分。
porn_score	Double	图片为色情图片的评分。
forbid_status	Int	封禁状态，0表示正常，1表示图片已被封禁（只有存储在腾讯云的图片才会被封禁）。

① 说明：

- 当 result=0时，表明图片为正常图片。
- 当 result=1时，表明该图片是系统判定为违禁涉黄的图片，如果存储在腾讯云则会直接被封禁掉。
- 当 result=2 时，表明该图片是疑似图片($83 \leq \text{confidence} < 91$)，即为黄图的可能性很大。

示例

输入示例：

POST /detection/porn_detect HTTP/1.1

Authorization: FCHXddYbhZEBfTeZ0j8mn9Og16JhPTEwMDAwMzc5Jms9QUtJRGVVRZDBrRU1yM2J4ZjhRckJiUkp3Sk5zbTN3V1IEeHN1JnQ9MTQ2ODQ3NDY2MiZyPTU3MiZ1PTAmYj10ZXN0YnVja2V0JmU9MTQ3MTA2NjY2Mg==

Host: recognition.image.myqcloud.com

Content-Length: 238

Content-Type: "application/json"

```
{
  "appid": 10000379,
  "url_list": [
    "http://www.bz55.com/uploads/allimg/140805/1-140P5162300-50.jpg",
    "http://img.taopic.com/uploads/allimg/130716/318769-130G60P30462.jpg"
  ]
}
```

输出示例：

```
{
  "result_list": [
    {
      "code": 0,
      "message": "success",
      "url": "http://www.bz55.com/uploads/allimg/140805/1-140P5162300-50.jpg",
      "data": {
        "result": 0,
        "forbid_status": 0,
        "confidence": 12.509,
        "hot_score": 87.293,
        "normal_score": 12.707,
        "porn_score": 0.0
      }
    },
    {
      "code": 0,
      "message": "success",
      "url": "http://img.taopic.com/uploads/allimg/130716/318769-130G60P30462.jpg",
      "data": {
        "result": 0,
        "forbid_status": 0,
        "confidence": 14.913,
        "hot_score": 99.997,
        "normal_score": 0.003,
        "porn_score": 0.0
      }
    }
  ]
}
```

使用图片文件

输入参数

使用 multipart/form-data 格式：

参数	必选	类型	说明
appid	是	uint	业务 ID。

参数	必选	类型	说明
image	是	image/jpeg	图片文件，每个请求最多支持20个。参数名须为 "image[0]"、"image[1]"等，每张图片需指定 filename。

输出参数

接口返回的 result_list 为 json 数组，数组的每个元素如下：

字段	类型	说明
code	Int	服务器错误码，0为成功。
message	String	服务器返回的信息。
filename	String	当前图片的 filename，与请求包中 filename 一致。
data	Object	具体查询数据，内容见下表。

data 字段具体内容：

字段	类型	说明
result	Int	供参考的识别结果：0为正常，1为黄图，2为疑似图片。
confidence	Double	识别为黄图的置信度，分值 0-100；是 normal_score、hot_score、porn_score 的综合评分。
normal_score	Double	图片为正常图片的评分。
hot_score	Double	图片为性感图片的评分。
porn_score	Double	图片为色情图片的评分。
forbid_status	Int	封禁状态，0表示正常，1表示图片已被封禁（只有存储在腾讯云的图片才会被封禁）。

① 说明：

- 当 result=0时，表示该图片为正常图片。
- 当 result=1时，表示该图片是系统判定的涉黄违禁的图片，如果存储在腾讯云则会直接被封禁掉。
- 当 result=2时，表示该图片是疑似图片($83 \leq \text{confidence} < 91$)，即为黄图的可能性很大。

示例

输入示例：

```
POST /detection/porn_detect HTTP/1.1
Content-Type:multipart/form-data;boundary=-----acebdf13572468
Authorization:FCHXddYbhZEBfTeZ0j8mn9Og16JhPTEwMDAwMzc5Jms9QUtJRGVRZDBrRU1yM2J4ZjhRckJiUkp3Sk5zbTN
3V1IEeHN1JnQ9MTQ2ODQ3NDY2MiZyPTU3MiZ1PTAmYj10ZXN0YnVja2V0JmU9MTQ3MTA2NjY2Mg==
Host: recognition.image.myqcloud.com
Content-Length: 61478

-----acebdf13572468
```

```
Content-Disposition: form-data; name="appid";

10000379
-----acebdf13572468
Content-Disposition: form-data; name="image[0]"; filename="1.jpg"
Content-Type: image/jpeg

<@INCLUDE *D:\185839ggh0oedgnog04g0b.jpg.thumb.jpg*@>
-----acebdf13572468
Content-Disposition: form-data; name="image[1]"; filename="2.jpg"
Content-Type: image/jpeg

<@INCLUDE *D:\200132svnmybmhbmmbmga.jpg.thumb.jpg*@>
-----acebdf13572468
```

输出示例：

```
{
  "result_list": [
    {
      "code": 0,
      "message": "success",
      "filename": "1.jpg",
      "data": {
        "result": 1,
        "forbid_status": 0,
        "confidence": 96.853,
        "hot_score": 0.0,
        "normal_score": 0.0,
        "porn_score": 100.0
      }
    },
    {
      "code": 0,
      "message": "success",
      "filename": "2.jpg",
      "data": {
        "result": 0,
        "forbid_status": 0,
        "confidence": 41.815,
        "hot_score": 19.417,
        "normal_score": 0.077,
        "porn_score": 80.506
      }
    }
  ]
}
```

错误码

错误码	含义
-----	----

错误码	含义
3	错误的请求；其中 message:account abnormal,errorno is:2 为账号欠费停服
4	签名为空
5	签名串错误
6	APPID/存储桶/ url 不匹配
7	签名编码失败（内部错误）
8	签名解码失败（内部错误）
9	签名过期
10	APPID 不存在
11	SecretId 不存在
12	APPID 不匹配
13	重放攻击
14	签名失败
15	操作太频繁，触发频控
16	内部错误
17	未知错误
200	内部打包失败
201	内部解包失败
202	内部链接失败
203	内部处理超时
-1300	图片为空
-1308	url 图片下载失败
-1400	非法的图片格式
-1403	图片下载失败
-1404	图片无法识别
-1505	url 格式不对
-1506	图片下载超时
-1507	无法访问 url 对应的图片服务器
-5062	url 对应的图片已被标注为不良图片，无法访问（专指存储于腾讯云的图片）

更多其他 API 错误码请查看 [错误码说明](#)。

鉴权签名

最近更新时间：2019-08-07 10:07:50

[拼接签名串](#)

操作

签名与鉴权

人脸支付通过签名来验证请求的合法性。开发者将签名授权给客户端，使其具备上传下载及管理指定资源的能力。

签名分为两种：

- 多次有效签名：签名中绑定或者不绑定文件 fileid，需要设置大于当前时间的有效期，最长可设置三个月，在此期间内签名可多次使用。
- 单次有效签名：签名中绑定文件 fileid，有效期必须设置为 0，此签名只可使用一次，且只能应用于被绑定的文件。

具体应用参见 [签名适用场景](#)。

签名算法

获取签名所需信息

生成签名所需信息必须使用主账号的，包括 App ID、Secret ID 和 Secret Key。

注意：

- 如果您已使用过 [API 密钥](#)，或在2018年4月1日后接入人脸支付，请使用 [API 密钥](#)。
- 如果您已使用过100、101等开头的项目ID，可以继续使用 [项目密钥](#)，但建议使用 [API 密钥](#)，2018年4月1日后创建的项目ID，不再支持使用 [项目密钥](#)。
- 目前仅支持使用主账号的 Secret ID 和 Secret Key，暂不支持子账号的使用，计划后续实现。

拼接签名串

拼接多次有效签名串：

`a=[appid]&b=[bucket]&k=[SecretID]&e=[expiredTime]&t=[currentTime]&r=[rand]&f=[fileid]`

拼接单次有效签名串：

`a=[appid]&b=[bucket]&k=[SecretID]&e=[expiredTime]&t=[currentTime]&r=[rand]&f=[fileid]`

注意：

- 多次有效签名串中 fileid 为可选参数。

- fileid 为空，表示不绑定资源，例如上传签名和下载签名。
- fileid 不为空，表示绑定资源，例如绑定资源的下载。

签名串中各字段含义如下：

字段	解释
a	开发者的 AppID，接入人脸支付时由系统生成。
b	Bucket，图片资源的组织管理单元，历史遗留字段，可不填。
k	Secret ID。
e	签名的有效期，是一个符合 UNIX Epoch 时间戳规范的数值，单位为秒；单次签名时，e 必须设置为 0。
t	当前时间戳，是一个符合 UNIX Epoch 时间戳规范的数值，单位为秒，多次签名时，e 应大于 t。
r	随机串，无符号 10 进制整数，用户需自行生成，最长 10 位。
f	资源存储的唯一标识，单次签名必填；多次签名选填，如填写则会验证与当前操作的文件路径是否一致。

注意：

- 拼接单次有效签名串时，有效期e必须设置为 0，以保证此签名只能针对固定资源使用一次。
- 删除和复制文件必须使用单次有效签名，上传必须使用多次有效签名。
- 具体应用参见 [签名适用场景](#)。

生成签名

1. 使用 HMAC-SHA1 算法对请求进行加密（SHA1算法加密后的输出必须是原始的二进制数据，否则签名失败）。
2. 对 original 使用 HMAC-SHA1 算法进行签名，然后将 original 附加到签名结果的末尾，再进行 Base64 编码，得到最终的 sign。
3. 生成签名的公式如下：

$\text{SignTmp} = \text{HMAC-SHA1}(\text{SecretKey}, \text{original})$

$\text{Sign} = \text{Base64}(\text{SignTmp}.\text{original})$

注意：

- 此处使用的是标准的 Base64 编码，不是 urlsafe 的 Base64 编码。
- SecretKey 为 API 密钥，original 为 [拼接签名串](#) 节中拼接好的签名串。

PHP 签名示例

本节介绍生成签名的算法实例，实例中使用 PHP 语言，如果开发者使用其他与开发，请使用对应的算法。

获取签名所需信息

获取得到的签名所需信息如下：

AppID : YOUR AppID_ID

Bucket : tencentyun (可不填)

Secret ID : YOUR SECRET_ID

Secret Key : YOUR SECRET_KEY

拼接签名串

```
$appid = "YOUR APPID_ID";
$bucket = "tencentyun";
$secret_id = "YOUR SECRET_ID";
$secret_key = "YOUR SECRET_KEY";
$expired = time() + 2592000;
$onceExpired = 0;
$current = time();
$rdm = rand();
$userid = "0";
$fileid = "tencentyunSignTest";

$srcStr = 'a='.$appid.'&b='.$bucket.'&k='.$secret_id.'&e='.$expired.'&t='.$current.'&r='.$rdm.'&f=';

$srcWithFile = 'a='.$appid.'&b='.$bucket.'&k='.$secret_id.'&e='.$expired.'&t='.$current.'&r='.$rdm.'&f='.$fileid;

$srcStrOnce = 'a='.$appid.'&b='.$bucket.'&k='.$secret_id.'&e='.$onceExpired.'&t='.$current.'&r='.$rdm
.&f='.$fileid;
```

生成签名

SHA1 算法加密后的输出必须是原始的二进制数据，否则签名失败：

```
$signStr = base64_encode(hash_hmac('SHA1', $srcStr, $secret_key, true).$srcStr);

$srcWithFile = base64_encode(hash_hmac('SHA1', $srcWithFile, $secret_key, true).$srcWithFile);

$signStrOnce = base64_encode(hash_hmac('SHA1', $srcStrOnce, $secret_key, true).$srcStrOnce);

echo $signStr."\n";

echo $srcWithFile ."\n";

echo $signStrOnce ."\n";
```

Java 签名示例

```
/*
 * Copyright 2017, Tencent Inc
 * All rights reserved.
 *
 * Created on 2017年9月12日
 */
```

```
package sign;

import java.util.Base64;
import java.util.Random;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

public class Sign {

    /**
     * 生成 Authorization 签名字段
     *
     * @param appId
     * @param secretId
     * @param secretKey
     * @param bucketName
     * @param expired
     * @return
     * @throws Exception
     */
    public static String appSign(long appId, String secretId, String secretKey, String bucketName,
    long expired) throws Exception {
        long now = System.currentTimeMillis() / 1000;
        int rdm = Math.abs(new Random().nextInt());
        String plainText = String.format("a=%d&b=%s&k=%s&t=%d&e=%d&r=%d", appId, bucketName,
        secretId, now, now + expired, rdm);
        byte[] hmacDigest = HmacSha1(plainText, secretKey);
        byte[] signContent = new byte[hmacDigest.length + plainText.getBytes().length];
        System.arraycopy(hmacDigest, 0, signContent, 0, hmacDigest.length);
        System.arraycopy(plainText.getBytes(), 0, signContent, hmacDigest.length,
        plainText.getBytes().length);
        return Base64Encode(signContent);
    }

    /**
     * 生成 base64 编码
     *
     * @param binaryData
     * @return
     */
    public static String Base64Encode(byte[] binaryData) {
        String encodedstr = Base64.getEncoder().encodeToString(binaryData);
        return encodedstr;
    }

    /**
     * 生成 hmacsha1 签名
     *
     * @param binaryData
     * @param key
     * @return
     * @throws Exception
     */
}
```

```

public static byte[] HmacSha1(byte[] binaryData, String key) throws Exception {
    Mac mac = Mac.getInstance("HmacSHA1");
    SecretKeySpec secretKey = new SecretKeySpec(key.getBytes(), "HmacSHA1");
    mac.init(secretKey);
    byte[] HmacSha1Digest = mac.doFinal(binaryData);
    return HmacSha1Digest;
}

/**
 * 生成 hmacsha1 签名
 *
 * @param plainText
 * @param key
 * @return
 * @throws Exception
 */
public static byte[] HmacSha1(String plainText, String key) throws Exception {
    return HmacSha1(plainText.getBytes(), key);
}
}

```

Node JS 签名示例

```

var crypto = require('crypto');

var secretId = 'YOUR_SECRET_ID',
    secretKey = 'YOUR_SECRET_KEY',
    appId = 'APPID',
    pexpired = 86400,
    userid = 0;

var now = parseInt(Date.now() / 1000),
    rdm = parseInt(Math.random() * Math.pow(2, 32)),
    plainText = 'a=' + appId + '&k=' + secretId + '&e=' + (now+peexpired) + '&t=' + now + '&r=' + rdm + userid + '&f=',
    data = new Buffer(plainText,'utf8'),
    res = crypto.createHmac('sha1',secretKey).update(data).digest(),
    bin = Buffer.concat([res,data]);

var sign = bin.toString('base64');

```

C++ 签名示例

```

//g++ -g sign_sample.cpp -o sign -lcrypto

#include <stdio.h>
#include <stdlib.h> /* srand, rand */
#include <time.h> /* time */

```

```
#include <openssl/hmac.h>
#include <openssl/pem.h>
#include <openssl/bio.h>
#include <openssl/evp.h>
#include <string>
#include <vector>
#include <sstream>

#define HMAC_LENGTH 20

std::vector<unsigned char> hmac_sha1(std::string& data, std::string& key)
{
    unsigned char* result;
    unsigned int len = HMAC_LENGTH;

    result = (unsigned char*)malloc(sizeof(char) * len);

    HMAC_CTX ctx;
    HMAC_CTX_init(&ctx);

    HMAC_Init_ex(&ctx, key.c_str(), key.length(), EVP_sha1(), NULL);
    HMAC_Update(&ctx, (unsigned char*)data.c_str(), data.length());
    HMAC_Final(&ctx, result, &len);
    HMAC_CTX_cleanup(&ctx);

    std::vector<unsigned char> sha1;
    for (int i = 0; i < len; i++){
        sha1.push_back(result[i]);
    }

    free(result);
    return sha1;
}

std::string base64_encode(const std::string& src){
    BUF_MEM * bptr = NULL;
    BIO* b64 = BIO_new(BIO_f_base64());
    BIO_set_flags(b64, BIO_FLAGS_BASE64_NO_NL);
    BIO* bmem = BIO_new(BIO_s_mem());
    if(NULL == b64 || NULL == bmem){
        return "";
    }

    bmem = BIO_push(b64, bmem);
    int ret = BIO_write(bmem, src.data(), src.length());
    if(ret <= 0){
        return "";
    }

    ret = BIO_flush(bmem);
    BIO_get_mem_ptr(bmem, &bptr);
    std::string res(bptr->data, bptr->length);
    BIO_free_all(bmem);
    return res;
}
```

```
}

int main() {
    std::string appid = "1000001";
    std::string secret_id = "YOUR SECRETID";
    std::string secret_key = "YOUR SECRETKEY";
    time_t now = time(NULL);
    long expired = (long)now + 2592000;
    long onceExpired = 0;
    long current = (long)now;
    int rdm = rand();
    std::string userid = "0";

    std::stringstream raw_stream;
    raw_stream << "a=" << appid << "&k=" << secret_id << "&e=" << expired << "&t=" << current
    << "&r=" << rdm;
    std::string raw = raw_stream.str();

    std::vector<unsigned char> sha1 = hmac_sha1(raw, secret_key);

    std::stringstream data_stream;
    for (int i = 0; i != sha1.size(); i++)
        data_stream << sha1[i];
    data_stream << raw;
    std::string data = data_stream.str();

    std::string sign = base64_encode(data);

    printf("%s\n", sign.c_str());

    return 0;
}
```

签名适用场景

签名的适用场景有如下限制：

场景	适用签名
智能鉴黄	多次有效签名
图片标签	多次有效签名
OCR识别	多次有效签名
人脸识别	多次有效签名
人脸核身	多次有效签名
人脸融合	多次有效签名

错误码说明

最近更新时间：2019-08-07 09:58:44

人脸错误码说明

错误码	含义
3	错误的请求；其中 message:account abnormal,errorno is:2 为账号欠费停服
4	签名为空
5	签名串错误
6	签名中的 appid/bucket 与操作目标不匹配
9	签名过期
10	appid 不存在
11	secretid 不存在
12	appid 和 secretid 不匹配
13	重放攻击
14	签名校验失败
15	操作太频繁，触发频控
16	Bucket 不存在
21	无效参数
23	请求包体过大
107	鉴权服务内部错误
108	鉴权服务不可用
213	内部错误
-1101	人脸检测失败
-1102	图片解码失败
-1103	特征处理失败
-1104	提取轮廓错误
-1105	提取性别错误
-1106	提取表情错误
-1107	提取年龄错误

错误码	含义
-1108	提取姿态错误
-1109	提取眼镜错误
-1200	特征存储错误
-1300	图片为空
-1301	参数为空
-1302	个体已存在
-1303	个体不存在
-1304	参数过长
-1305	人脸不存在
-1306	组不存在
-1307	组列表不存在
-1308	url 图片下载失败
-1309	人脸个数超过限制
-1310	个体个数超过限制
-1311	组个数超过限制
-1312	对个体添加了几乎相同的人脸
-1400	非法的图片格式
-1403	图片下载失败