目录

人脸检测与分析

调用方式

请求结构

服务地址

AI开放服务API的服务接入地址为：http://kcr.api.ksyun.com

通信协议

支持通过HTTP或HTTPS协议进行请求通信。为保证您的服务安全性，请使用HTTPS协议进行通信。

请求方式

支持GET方法发送请求，注意参数需要进行urlencode。

公共参数

公共请求header

参数名	必选	类型	说明
Authorization	是	string	必要的请求验证信息
X-Amz-Date	是	string	当前请求的时间戳，例如：20171129T114852Z
Host	否	string	kcr.api.ksyun.com

签名参数

Region：金山云机房信息，目前仅支持cn-beijing-6

Service：固定为kcr

返回结果

接口统一返回json格式数据，满足以下格式：

```
{
    "header":{
      "err_no":200,
      "err_msg":"success"
    },
    "cost":0.11,
    "request_id": "d679e27b-9f1b-44bf-b134-49a6d9f0adff",
    "request_time": 1234567890123,
    "body":{}
}
```

调用成功，err_no返回200，msg为success，否则返回对应错误码及错误信息。

cost为服务端耗时，单位秒。

每此请求会返回唯一的请求表示request_id，调用失败时提供request_id给金山云客服，方便定位问题。

request_time为每次请求时的服务器时间。

签名机制

[详见金山云签名机制文档]

人脸检测接口

输入：

GET参数部分

参数名	必选	类型	说明
Action=DetectFace	是	string	固定此值

Version=2019-12-13	是	string	API版本号，固定此值

POST请求示例

```
{
    "image_url":"xxxxx"
}
```

POST参数说明

参数	必选	类型	说明
image_url	否	string	图片url （image_url和image_data二选一）
image_data	否	string	图片内容的BASE64编码，不能超过1MB （image_url和image_data二选一）
need_attributes	否	string	是否检测人体属性，默认不检测

接口返回：


```
{
    "header": {
        "err_no": 200,
        "err_msg": "success"
    },
    "request_id": "2216eaa9-52d2-4e0f-a690-e79b10a2c2fe",
    "cost": 0.495,
    "face_num": 2,
    "face_info": [
        {
            "attributes": {
                "headwear": {
                    "rate": 0.323924720287323,
                    "name": "未戴帽子"
                },
                "gender": {
                    "rate": 0.8240132331848145,
                    "name": "女性"
                },
                "age": {
                    "rate": 0.9495382308959961,
                    "name": "青年"
                },
                "glasses": {
                    "rate": 0.9292197823524475,
                    "name": "普通眼镜"
                },
                "mask": {
                    "rate": 0.7128473520278931,
                    "name": "戴口罩"
                },
                "race": {
                    "rate": 0.4870010614395142,
                    "name": "维族"
                }
            },
            "location": {
                "top_left_x": 0.16029005560994847,
                "top_left_y": 0.2177239403843063,
                "bottom_right_y": 0.3159262808263047,
                "bottom_right_x": 0.20467229934001807,
                "rate": 0.9999997615814209
            }
        }
    ]
}
```


接口输出字段解释：

字段	类型	是否必须	说明
header.err_no	int	是	错误码，请求成功为200
header.err_msg	string	是	错误信息，请求成功为success
request_id	string	是	请求唯一标识
cost	double	是	请求耗时，单位：秒
face_num	int	是	图像中人脸个数
face_info.location	object	是	图像中人脸位置信息
face_info.location.top_left_x	double	是	人脸左上坐标的x值
face_info.location.top_left_y	double	是	人脸左上坐标的y值

face_info.location.bottom_right_x	double	是	人脸右下坐标的x值
face_info.location.bottom_right_y	double	是	人脸右下坐标的y值
face_info.location.rate	double	是	人脸置信度
face_info.attributes	object	否	人脸属性信息
face_info.attributes.gender	object	否	人脸性别
face_info.attributes.gender.name	string	否	男性、女性
face_info.attributes.gender.rate	double	否	置信度
face_info.attributes.age	object	否	人脸年龄
face_info.attributes.age.name	string	否	儿童、少年、青年、中年、老年
face_info.attributes.age.rate	double	否	置信度
face_info.attributes.headwear	object	否	戴帽子
face_info.attributes.headwear.name	string	否	未戴帽子、戴头盔、戴帽子
face_info.attributes.headwear.rate	double	否	置信度
face_info.attributes.glasses	object	否	人脸戴眼镜
face_info.attributes.glasses.name	string	否	普通眼镜、太阳镜、未戴眼镜
face_info.attributes.glasses.rate	double	否	置信度
face_info.attributes.mask	object	否	人脸戴口罩
face_info.attributes.mask.name	string	否	戴口罩、无口罩
face_info.attributes.respirator.rate	double	否	置信度
face_info.attributes.race	object	否	民族
face_info.attributes.race.name	string	否	维族、非维族
face_info.attributes.race.rate	double	否	置信度

注：为保证图片缩放后坐标点不变，此处返回坐标值均为相对图片长宽的百分比。

签名错误码

错误代码(Code)	错误消息(Message)	状态码	说明
IncompleteSignature	Date must be in ISO-8601 'basic format'. Got '%s'. See http://en.wikipedia.org/wiki/ISO_8601.	400	Date必须符合ISO_8601基本格式，参考：http://en.wikipedia.org/wiki/ISO_8601
IncompleteSignature	KSC query-string parameters must include %s. Re-examine the query-string parameters.	400	查询条件中缺少签署信息，查询条件中必须包含”X-Amz-Algorithm“、”X-Amz-Credential“、”X-Amz-SignedHeaders“、”X-Amz-Date“信息
IncompleteSignature	Unsupported ksc 'algorithm': %s.	400	只支持如下签名算法：AWS4-HMAC-SHA256
IncompleteSignature	Authorization header requires 'Credential' parameter. Authorization=%s.	400	请求Authorization header中需要包含“Credential”参数
IncompleteSignature	Credential must have exactly 5 slash-delimited elements, e.g. accesskeyid/date/region/service/aws4_request, got: %s.	400	请求Authorization header中中“Credential”至少包含5项以斜杠分隔的元素，如：keyid/date/region/service/aws4_request
IncompleteSignature	Authorization header format error.	400	请求Authorization header的格式错误
IncompleteSignature	Authorization header requires existence of either a 'X-Amz-Date' or a 'Date' header, Authorization=%s	400	请求中缺少“X-Amz-Date”或者“Date” header信息
IncompleteSignature	Authorization header requires 'Signature' parameter. Authorization=%s	400	请求Authorization header中缺少“Signature”信息
IncompleteSignature	Authorization header requires 'SignedHeaders' parameter. Authorization=%s	400	请求Authorization header中缺少“SignedHeaders”信息
MissingAuthenticationToken	Request is missing 'Host' header.	403	请求header中缺少Host
MissingAuthenticationToken	Request is missing Authentication Token.	403	请求header中缺少认证token
MissingAuthenticationToken	%s not in Http Header.	403	%s不在Http header中

SignatureDoesNotMatch	Host' must be a 'SignedHeader' in the Authorization.	403	请求的SignedHeader中必须包含Host
SignatureDoesNotMatch	Credential should be scoped with a valid terminator: 'aws4_request', not: %s.	403	请求Authorization header中的“Credential”末尾必须是“aws4_request”
SignatureDoesNotMatch	Credential should be scoped to a valid region, not:%s.	403	请求Authorization header中的“Credential”中的Region信息无效
SignatureDoesNotMatch	Credential should be scoped to correct service: %s.	403	请求Authorization header中的“Credential”中的Service信息无效
SignatureDoesNotMatch	The request signature we calculated does not match the signature you provided.	403	请求中提供的签名与实际计算结果不匹配
SignatureDoesNotMatch	Signature expired:%s.	403	签名已过期
SignatureDoesNotMatch	Date in Credential scope does not match YYYYMMDD from ISO-8601 version of date from HTTP.	403	请求Authorization header中的“Credential”中的Date应该是ISO8601基本格式，形如”YYYYMMDD“
InvalidClientTokenId	The security token included in the request is invalid.	403	请求中提供的AccessKeyId无效

业务层级错误码

err_msg	err_no	说明
未开通相应的服务	411	账号未开通服务权限
请求太频繁	510	QPS达到限制

其他说明

图片大小限制为5M,长宽像素不大于5000，图片下载时间限制为2.5s内，如果下载时间超过2.5s返回下载超时；

图片像素建议不小于256*256，太小可能会影响识别效果；

图片检测接口响应时间依赖图片的下载时间，请保证所检测图片所在的存储服务的稳定，图片建议使用金山云对象存储或者cdn做缓存等；

接口regionId仅支持cn-beijing-6；

人脸比对

调用方式

请求结构

服务地址

AI开放服务API的服务接入地址为：http://kcr.api.ksyun.com

通信协议

支持通过HTTP或HTTPS协议进行请求通信。为保证您的服务安全性，请使用HTTPS协议进行通信。

请求方式

支持GET方法发送请求，注意参数需要进行urlencode。

公共参数

公共请求header

参数名	必选	类型	说明

Authorization	是	string	必要的请求验证信息
X-Amz-Date	是	string	当前请求的时间戳，例如：20171129T114852Z
Host	否	string	kcr.api.ksyun.com

签名参数

Region：金山云机房信息，目前仅支持cn-beijing-6

Service：固定为kcr

返回结果

接口统一返回json格式数据，满足以下格式：

```
{
    "header":{
      "err_no":200,
      "err_msg":"success"
    },
    "cost":0.11,
    "request_id": "d679e27b-9f1b-44bf-b134-49a6d9f0adff",
    "request_time": 1234567890123,
    "body":{}
}
```

调用成功，err_no返回200，msg为success，否则返回对应错误码及错误信息。

cost为服务端耗时，单位秒。

每此请求会返回唯一的请求表示request_id，调用失败时提供request_id给金山云客服，方便定位问题。

request_time为每次请求时的服务器时间。

签名机制

[详见金山云签名机制文档]

人脸对比接口

输入：

GET参数部分

参数名	必选	类型	说明
Action=CalculateFaceSimilarity	是	string	固定此值
Version=2019-12-13	是	string	API版本号，固定此值

POST请求示例

```
{
    "image1_url":"https://datasets.ks3-cn-beijing.ksyun.com/liandanlu/401ffac2d0b9706f7222e54833a1e7e5.jpg",
    "image2_url":"https://datasets.ks3-cn-beijing.ksyun.com/liandanlu/401ffac2d0b9706f7222e54833a1e7e5.jpg"
}
```

POST参数说明

参数	必选	类型	说明
image1_url	否	string	对比图片1的url（image1_url和image1_data二选一）
image1_data	否	string	对比图片1的内容的BASE64编码 （image1_url和image1_data二选一）
image2_url	否	string	对比图片2的url（image2_url和image2_data二选一）
image2_data	否	string	对比图片2的内容的BASE64编码 （image2_url和image2_data二选一）

注：如果使用图像base64传输，请注意整体请求不能超过1MB。

接口返回：

```
{
    "header": {
        "err_no": 200,
        "err_msg": "success"
    },
```

```
    "request_id": "090e40f5-f90c-4921-b058-30f1a7e83c3d",
    "rate": 0.82,
    "cost": 0.627,
    "img1_face_info": {
        "face_num": 1,
        "location": [
            {
                "top_left_x": 0.3674023490748368,
                "top_left_y": 0.17655514139411357,
                "bottom_right_y": 0.7315950006291683,
                "bottom_right_x": 0.6000457712216303,
                "rate": 0.9999995231628418
            }
        ]
    },
    "img2_face_info": {
        "face_num": 1,
        "location": [
            {
                "top_left_x": 0.3674023490748368,
                "top_left_y": 0.17655514139411357,
                "bottom_right_y": 0.7315950006291683,
                "bottom_right_x": 0.6000457712216303,
                "rate": 0.9999995231628418
            }
        ]
    }
}
```


接口输出字段解释：

字段	类型	是否必须	说明
header.err_no	int	是	错误码，请求成功为200
header.err_msg	string	是	错误信息，请求成功为success
request_id	string	是	请求唯一标识
cost	double	是	请求耗时，单位：秒
rate	double	是	人脸相似度（若图像中含有多张人脸，以宽度最大的人脸做为对比人脸）
img1_face_info	object	是	对比图像1的人脸识别详细信息
img2_face_info	object	是	对比图像2的人脸识别详细信息
face_num	int	是	图像中人脸个数
location	object	是	图像中人脸位置信息
location.top_left_x	double	是	人脸左上坐标的x值
location.top_left_y	double	是	人脸左上坐标的y值
location.bottom_right_x	double	是	人脸右下坐标的x值
location.bottom_right_y	double	是	人脸右下坐标的y值
location.rate	double	是	人脸置信度

注：为保证图片缩放后坐标点不变，此处返回坐标值均为相对图片长宽的百分比。

错误码说明

调用接口失败时，返回的HTTP消息体将中包含具体的错误信息，下表为错误码的具体说明，找不到错误原因时，可以联系我们，并提供Response中的RequestId/request_id，以便尽快解决问题。

签名错误码

错误代码(Code)	错误消息(Message)	状态码	说明
IncompleteSignature	Date must be in ISO-8601 'basic format'. Got '%s'. See http://en.wikipedia.org/wiki/ISO_8601.	400	Date必须符合ISO_8601基本格式，参考：http://en.wikipedia.org/wiki/ISO_8601
IncompleteSignature	KSC query-string parameters must include %s. Re-examine the query-string parameters.	400	查询条件中缺少签署信息，查询条件中必须包含”X-Amz-Algorithm“、”X-Amz-Credential“、”X-Amz-SignedHeaders“、”X-Amz-Date“信息
IncompleteSignature	Unsupported ksc 'algorithm': %s.	400	只支持如下签名算法：AWS4-HMAC-SHA256
IncompleteSignature	Authorization header requires 'Credential' parameter. Authorization=%s.	400	请求Authorization header中需要包含“Credential”参数

IncompleteSignature	Credential must have exactly 5 slash-delimited elements, e.g. accesskeyid/date/region/service/aws4_request, got: %s.	400	请求Authorization header中中“Credential”至少包含5项以斜杠分隔的元素，如：keyid/date/region/service/aws4_request
IncompleteSignature	Authorization header format error.	400	请求Authorization header的格式错误
IncompleteSignature	Authorization header requires existence of either a 'X-Amz-Date' or a 'Date' header, Authorization=%s	400	请求中缺少“X-Amz-Date”或者“Date” header信息
IncompleteSignature	Authorization header requires 'Signature' parameter. Authorization=%s	400	请求Authorization header中缺少“Signature”信息
IncompleteSignature	Authorization header requires 'SignedHeaders' parameter. Authorization=%s	400	请求Authorization header中缺少“SignedHeaders”信息
MissingAuthenticationToken	Request is missing 'Host' header.	403	请求header中缺少Host
MissingAuthenticationToken	Request is missing Authentication Token.	403	请求header中缺少认证token
MissingAuthenticationToken	%s not in Http Header.	403	%s不在Http header中
SignatureDoesNotMatch	Host' must be a 'SignedHeader' in the Authorization.	403	请求的SignedHeader中必须包含Host
SignatureDoesNotMatch	Credential should be scoped with a valid terminator: 'aws4_request', not: %s.	403	请求Authorization header中的“Credential”末尾必须是“aws4_request”
SignatureDoesNotMatch	Credential should be scoped to a valid region, not:%s.	403	请求Authorization header中的“Credential”中的Region信息无效
SignatureDoesNotMatch	Credential should be scoped to correct service: %s.	403	请求Authorization header中的“Credential”中的Service信息无效
SignatureDoesNotMatch	The request signature we calculated does not match the signature you provided.	403	请求中提供的签名与实际计算结果不匹配
SignatureDoesNotMatch	Signature expired:%s.	403	签名已过期
SignatureDoesNotMatch	Date in Credential scope does not match YYYYMMDD from ISO-8601 version of date from HTTP.	403	请求Authorization header中的“Credential”中的Date应该是ISO8601基本格式，形如”YYYYMMDD“
InvalidClientTokenId	The security token included in the request is invalid.	403	请求中提供的AccessKeyId无效

业务层级错误码

err_msg	err_no	说明
未开通相应的服务	411	账号未开通服务权限
请求太频繁	510	QPS达到限制

其他说明

图片大小限制为5M,长宽像素不大于5000，图片下载时间限制为2.5s内，如果下载时间超过2.5s返回下载超时；

图片像素建议不小于256*256，太小可能会影响识别效果；

图片检测接口响应时间依赖图片的下载时间，请保证所检测图片所在的存储服务的稳定，图片建议使用金山云对象存储或者cdn做缓存等；

接口regionId仅支持cn-beijing-6；

人脸活体

调用方式

请求结构

服务地址

AI开放服务API的服务接入地址为：http://kcr.api.ksyun.com

通信协议

支持通过HTTP或HTTPS协议进行请求通信。为保证您的服务安全性，请使用HTTPS协议进行通信。

请求方式

支持GET方法发送请求，注意参数需要进行urlencode。

公共参数

公共请求header

参数名	必选	类型	说明
Authorization	是	string	必要的请求验证信息
X-Amz-Date	是	string	当前请求的时间戳，例如：20171129T114852Z
Host	否	string	kcr.api.ksyun.com

签名参数

Region：金山云机房信息，目前仅支持cn-beijing-6

Service：固定为kcr

返回结果

接口统一返回json格式数据，满足以下格式：

```
{
    "header":{
      "err_no":200,
      "err_msg":"success"
    },
    "cost":0.11,
    "request_id": "d679e27b-9f1b-44bf-b134-49a6d9f0adff",
    "request_time": 1234567890123,
    "body":{}
}
```

调用成功，err_no返回200，msg为success，否则返回对应错误码及错误信息。

cost为服务端耗时，单位秒。

每此请求会返回唯一的请求表示request_id，调用失败时提供request_id给金山云客服，方便定位问题。

request_time为每次请求时的服务器时间。

签名机制

[详见金山云签名机制文档]

活体检测

注意事项

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体
- **Base64编码**：图片的base64编码是指将图片数据编码成一串字符串，使用该字符串代替图像地址，首先得到图片的二进制，然后用Base64格式编码即可。注意：图片的base64编码是不包含图片头的，比如data:image/jpg;base64
- **图片格式**：现支持PNG、JPG、JPEG、BMP，不支持GIF图片

HTTP请求方式

post

输入

GET参数部分

参数名	必选	类型	说明
Action=FaceVerify	是	string	固定此值
Version=2020-07-16	是	string	API版本号，固定此值

Body参数

```
{"images":[
      {"image_url":"http://www.baidu.com/image/QKA666.jpg",
        "face_field":"quality",
        "option":"COMMON"}
      ]
}
```

或

```
{"images":[
      {"image_data":"9jjiuuyu86sadw00...",
        "face_field":"quality",
        "option":"COMMON"}
      ]
}
```

Body参数说明

参数	必选	类型	说明
image_url	可选	string	图片URL(**image_url和image_data必须二选一**，图片大小不超过2M)
image_data	可选	string	Base64编码后的图片，需urlencode，编码后的图片大小不超过2M
face_field	可选	string	当前只支持固定值quality，可不指定该参数，接口将默认采用quality
option	可选	string	场景信息，程序会视不同的场景选用相对应的模型，当前支持的场景有COMMON(通用场景)，GATE(闸机场景)，默认使用COMMON

输出

接口返回示例：

```
{
    "header": {
        "err_no": 200,
        "err_msg": "success"
    },
    "cost": 3.29,
    "result": {
        "thresholds": {
            "frr_1e-4": 0.05,
            "frr_1e-3": 0.3,
            "frr_1e-2": 0.9
        },
        "face_liveness": 1.0,
        "face_list": [
                {
                "location": {
                    "left": 222.23,
                    "top": 414.4,
                    "width": 320.0,
                    "height": 353.0,
                    "rotation": 1
                },
                "angle": {
                    "yam": 0.0,
                    "pitch": -0.1,
                    "roll": -0.44
                },
                "quality": {
                    "blur": 0.0,
                    "illumination": 88.0,
                    "completeness": 1
                },
                "face_token": "8d362ef6b11de723358893ebb770061f",
                "face_probability": 1.0
            }
```

```
        ]
    },
    "request_id": "\"123\""
}
```

接口输出字段说明

字段	类型	是否必须	说明
header	object	是	返回头信息
+err_no	int	是	错误码，请求成功为200，请求错误时会返回400(参数错误)或500(服务内部错误)
+err_msg	string	是	错误信息，请求成功为success；当请求错误时，如果是服务内部错误，会统一返回"服务器内部错误"；如果是参数错误，则会返回具体的参数错误信息
request_id	string	是	请求唯一标识
cost	double	是	请求耗时，单位：秒
result	object	是	识别结果，请求失败时该参数为null
+face_liveness	float	是	活体分数值
+thresholds	object	是	由服务端返回最新的阈值数据（随着模型的优化，阈值可能会变化），将此参数与返回的face_liveness进行比较，可以作为活体判断的依据。误识率越低，准确率越高，相应的拒绝率也越高
++frr_1e-4	float	是	万分之一误识率的阈值
++frr_1e-3	float	是	千分之一误识率的阈值
++frr_1e-2	float	是	百分之一误识率的阈值
+face_list	array	是	图片的详细信息描述
++face_token	doube	是	人脸图片的唯一标识
++location	object	是	人脸在图片中的位置
+++left	double	是	人脸区域离左边界的距离
+++top	double	是	人脸区域离上边界的距离
+++width	double	是	人脸区域的宽度
+++height	double	是	人脸区域的高度
+++rotation	int64	是	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
++face_probability	double	是	人脸置信度，范围【0~1】，代表这是一张人脸的概率，0最小、1最大
++angle	object	是	人脸旋转角度参数
+++yam	double	是	三维旋转之左右旋转角[-90(左), 90(右)]
+++pitch	double	是	三维旋转之俯仰角度[-90(上), 90(下)]
+++roll	double	是	平面内旋转角[-180(逆时针), 180(顺时针)]
++quality	object	是	人脸质量信息
+++blur	double	是	人脸模糊程度，范围[0~1]，0表示清晰，1表示模糊
+++illumination	double	是	取值范围在[0~255], 表示脸部区域的光照程度 越大表示光照越好
+++completeness	int64	是	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内

活体阈值参考

活体检测face_liveness的判断阈值选择，可参考下表：

阈值（Threshold）	误拒率（FRR）	通过率（TAR）	攻击拒绝率（TRR））
0.05	0.01%	99.99%	97.75%
0.1	0.05%	99.95%	98.33%
0.3 （**推荐**）	0.1%	99.9%	98.82%
0.5	0.5%	99.5%	99.67%
0.9	1%	99%	99.77%

概念介绍：

- **拒绝率（TRR）**：比如99%，指100次作弊假体识别，会有99次被拒绝。
- **误拒率（FRR）**：比如1%，指1000次正常活体识别，会有10次因为活体分数低于阈值被错误拒绝。

错误码说明

调用接口失败时，返回的HTTP消息体将中包含具体的错误信息，下表为错误码的具体说明，找不到错误原因时，可以联系我们，并提供Response中的RequestId/request_id，以便尽快解决问题。

签名错误码

错误代码(Code)	错误消息(Message)	状态码	说明
IncompleteSignature	Date must be in ISO-8601 'basic format'. Got '%s'. See http://en.wikipedia.org/wiki/ISO_8601.	400	Date必须符合ISO_8601基本格式，参考：http://en.wikipedia.org/wiki/ISO_8601
IncompleteSignature	KSC query-string parameters must include %s. Re-examine the query-string parameters.	400	查询条件中缺少签署信息，查询条件中必须包含”X-Amz-Algorithm“、”X-Amz-Credential“、”X-Amz-SignedHeaders“、”X-Amz-Date“信息
IncompleteSignature	Unsupported ksc 'algorithm': %s.	400	只支持如下签名算法：AWS4-HMAC-SHA256
IncompleteSignature	Authorization header requires 'Credential' parameter. Authorization=%s.	400	请求Authorization header中需要包含“Credential”参数
IncompleteSignature	Credential must have exactly 5 slash-delimited elements, e.g. accesskeyid/date/region/service/aws4_request, got: %s.	400	请求Authorization header中中“Credential”至少包含5项以斜杠分隔的元素，如：keyid/date/region/service/aws4_request
IncompleteSignature	Authorization header format error.	400	请求Authorization header的格式错误
IncompleteSignature	Authorization header requires existence of either a 'X-Amz-Date' or a 'Date' header, Authorization=%s	400	请求中缺少“X-Amz-Date”或者“Date” header信息
IncompleteSignature	Authorization header requires 'Signature' parameter. Authorization=%s	400	请求Authorization header中缺少“Signature”信息
IncompleteSignature	Authorization header requires 'SignedHeaders' parameter. Authorization=%s	400	请求Authorization header中缺少“SignedHeaders”信息
MissingAuthenticationToken	Request is missing 'Host' header.	403	请求header中缺少Host
MissingAuthenticationToken	Request is missing Authentication Token.	403	请求header中缺少认证token
MissingAuthenticationToken	%s not in Http Header.	403	%s不在Http header中
SignatureDoesNotMatch	Host' must be a 'SignedHeader' in the Authorization.	403	请求的SignedHeader中必须包含Host
SignatureDoesNotMatch	Credential should be scoped with a valid terminator: 'aws4_request', not: %s.	403	请求Authorization header中的“Credential”末尾必须是“aws4_request”
SignatureDoesNotMatch	Credential should be scoped to a valid region, not:%s.	403	请求Authorization header中的“Credential”中的Region信息无效
SignatureDoesNotMatch	Credential should be scoped to correct service: %s.	403	请求Authorization header中的“Credential”中的Service信息无效
SignatureDoesNotMatch	The request signature we calculated does not match the signature you provided.	403	请求中提供的签名与实际计算结果不匹配
SignatureDoesNotMatch	Signature expired:%s.	403	签名已过期

SignatureDoesNotMatch	Date in Credential scope does not match YYYYMMDD from ISO-8601 version of date from HTTP.	403	请求Authorization header中的“Credential”中的Date应该是ISO8601基本格式，形如”YYYYMMDD“
InvalidClientTokenId	The security token included in the request is invalid.	403	请求中提供的AccessKeyId无效

业务层级错误码

err_msg	err_no	说明
未开通相应的服务	411	账号未开通服务权限
请求太频繁	510	QPS达到限制

其他说明

图片大小限制为5M,长宽像素不大于5000，图片下载时间限制为2.5s内，如果下载时间超过2.5s返回下载超时；

图片像素建议不小于256*256，太小可能会影响识别效果；

图片检测接口响应时间依赖图片的下载时间，请保证所检测图片所在的存储服务的稳定，图片建议使用金山云对象存储或者cdn做缓存等；

接口regionId仅支持cn-beijing-6；

- **通过率（TAR）**：比如99%，指100次正常活体识别，会有99次因为活体分数高于阈值而通过。
- **阈值（Threshold）**：高于此数值，则可判断为活体，推荐0.3。

身份核验

调用方式

请求结构

服务地址

AI开放服务API的服务接入地址为：http://kcr.api.ksyun.com

通信协议

支持通过HTTP或HTTPS协议进行请求通信。为保证您的服务安全性，请使用HTTPS协议进行通信。

请求方式

支持GET方法发送请求，注意参数需要进行urlencode。

公共参数

公共请求header

参数名	必选	类型	说明
Authorization	是	string	必要的请求验证信息
X-Amz-Date	是	string	当前请求的时间戳，例如：20171129T114852Z
Host	否	string	kcr.api.ksyun.com

签名参数

Region：金山云机房信息，目前仅支持cn-beijing-6

Service：固定为kcr

返回结果

接口统一返回json格式数据，满足以下格式：

```
{
    "header":{
      "err_no":200,
```

```
    "err_msg":"success"
  },
  "cost":0.11,
  "request_id": "d679e27b-9f1b-44bf-b134-49a6d9f0adff",
  "request_time": 1234567890123,
  "body":{}
}
```

调用成功，err_no返回200，msg为success，否则返回对应错误码及错误信息。

cost为服务端耗时，单位秒。

每此请求会返回唯一的请求表示request_id，调用失败时提供request_id给金山云客服，方便定位问题。

request_time为每次请求时的服务器时间。

签名机制

[详见金山云签名机制文档]

身份核验

业务能力

- **质量检测（可选）**：判断图片中是否包含人脸，以及判断人脸的姿态、遮挡、模糊、光照等是否符合识别条件，**通过参数quality_control设置是否进行检测**；
- **活体检测（可选）**：判断图片中的人脸是否为二次翻拍（如用户A用手机拍摄了一张包含人脸的图片1，用户B翻拍了图片1得到了图片2，并用图片2伪造成用户A去进行识别操作），**通过参数liveness_control设置是否进行检测**；
- **公安验证（必选）**：根据姓名和身份证号，从公安系统调取公民身份证小图，将指定的人脸图片与证件小图进行对比得到比对分数，基于对比分数判断是否为同一人。由于公安系统小图具有最权威的身份证明作用，对用户本人的验证结果可信度也最高。
- 上述三项能力为顺序串行验证，接口默认返回公安身份对比分值，质量检测和活体检测为可选项。如果选择了这两项，则验证顺序为人脸质量检测->活体检测->公安身份验证。可以根据业务场景，选择这两项中的某一项或都不选择。
- 如选择了前两个环节，则有任意一个条件不通过，则整个请求流程终止，并返回具体的错误信息。

推荐阈值

- 此接口使用的对比算法，使用了针对带水纹证件照的专项模处理，可保证将水纹信息的影响降到尽可能低。
- 如果比对成功，最终返回的有效数据为一个**对比分值**，在0~1之间，您可以设定具体的阈值来判断是否验证通过，推荐阈值为0.8，对应的误识率为万分之一。

  注意事项：

  - **请求体格式化**：Content-Type为application/json，通过json格式化请求体
  - **Base64编码**：图片的base64编码是指将图片数据编码成一串字符串，使用该字符串代替图像地址，首先得到图片的二进制，然后用Base64格式编码即可。注意：图片的base64编码是不包含图片头的，比如data:image/jpg;base64
  - **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**

HTTP请求方式

post

请求参数

GET参数部分

参数名	必选	类型	说明
Action=PersonVerify	是	string	固定此值
Version=2020-07-16	是	string	API版本号，固定此值

Body参数格式

```
{
 "image_url":"https://www.zhibo8.cc/123.jpg",
 "id_card_number":"320103200001267529",
 "name":"窃.格瓦拉",
 "quality_control":"NORMAL",
 "liveness_control":"NORMAL"
```

```
}
```

或

```
{
 "image_data":"adersfdfsafgf23...",
 "id_card_number":"320103200001267529",
 "name":"窃.格瓦拉",
 "quality_control":"NORMAL",
 "liveness_control":"NORMAL"
}
```

参数	必选	类型	说明
image_url	否	string	图片URL(**image_url和image_data必须二选一**，图片大小不超过2M)
image_data	否	string	Base64编码后的图片数据，编码后的图片大小不超过2M，图片尺寸不超过1920*1080
id_card_number	是	string	身份证号码
name	是	string	姓名（注：需要是UTF-8编码的中文）
quality_control	否	string	图片质量控制，对应业务能力中的"质量检测" NONE: 不进行控制 LOW:较低的质量要求 NORMAL: 一般的质量要求 HIGH: 较高的质量要求 **默认 NONE**
liveness_control	否	string	活体检测控制 ，对应业务能力中的"活体检测" NONE: 不进行控制 LOW:较低的活体要求(高通过率 低攻击拒绝率) NORMAL: 一般的活体要求(平衡的攻击拒绝率, 通过率) HIGH: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**

输出

接口返回示例

```
{
    "header": {
        "err_no": 200,
        "err_msg": "success"
    },
    "cost": 1.092,
    "result": {
        "score": 94.503
    },
    "request_id": "\"123\""
}
```

参数	必选	类型	说明
header	是	object	返回头信息
header.err_no	是	int	错误码，请求成功为200，请求错误时会返回400(参数错误)或500(服务内部错误)
header.err_msg	是	string	错误信息，请求成功为success；当请求错误时，如果是服务内部错误，会统一返回"服务器内部错误"；如果是参数错误，则会返回具体的参数错误信息
cost	是	double	请求耗时，单位：秒
result	是	object	验证结果
result.score	是	float	与公安小图相似度可能性，用于验证生活照与公安小图是否为同一人，有正常分数时为[0~100]，推荐阈值80，超过即判断为同一人
request_id	是	string	请求唯一标识

错误码说明

调用接口失败时，返回的HTTP消息体将中包含具体的错误信息，下表为错误码的具体说明，找不到错误原因时，可以联系我们，并提供Response中的RequestId/request_id，以便尽快解决问题。

签名错误码

错误代码(Code)	错误消息(Message)	状态码	说明

IncompleteSignature	Date must be in ISO-8601 'basic format'. Got '%s'. See http://en.wikipedia.org/wiki/ISO_8601.	400	Date必须符合ISO_8601基本格式，参考：http://en.wikipedia.org/wiki/ISO_8601
IncompleteSignature	KSC query-string parameters must include %s. Re-examine the query-string parameters.	400	查询条件中缺少签署信息，查询条件中必须包含”X-Amz-Algorithm“、”X-Amz-Credential“、”X-Amz-SignedHeaders“、”X-Amz-Date“信息
IncompleteSignature	Unsupported ksc 'algorithm': %s.	400	只支持如下签名算法：AWS4-HMAC-SHA256
IncompleteSignature	Authorization header requires 'Credential' parameter. Authorization=%s.	400	请求Authorization header中需要包含“Credential”参数
IncompleteSignature	Credential must have exactly 5 slash-delimited elements, e.g. accesskeyid/date/region/service/aws4_request, got: %s.	400	请求Authorization header中中“Credential”至少包含5项以斜杠分隔的元素，如：keyid/date/region/service/aws4_request
IncompleteSignature	Authorization header format error.	400	请求Authorization header的格式错误
IncompleteSignature	Authorization header requires existence of either a 'X-Amz-Date' or a 'Date' header, Authorization=%s	400	请求中缺少“X-Amz-Date”或者“Date” header信息
IncompleteSignature	Authorization header requires 'Signature' parameter. Authorization=%s	400	请求Authorization header中缺少“Signature”信息
IncompleteSignature	Authorization header requires 'SignedHeaders' parameter. Authorization=%s	400	请求Authorization header中缺少“SignedHeaders”信息
MissingAuthenticationToken	Request is missing 'Host' header.	403	请求header中缺少Host
MissingAuthenticationToken	Request is missing Authentication Token.	403	请求header中缺少认证token
MissingAuthenticationToken	%s not in Http Header.	403	%s不在Http header中
SignatureDoesNotMatch	Host' must be a 'SignedHeader' in the Authorization.	403	请求的SignedHeader中必须包含Host
SignatureDoesNotMatch	Credential should be scoped with a valid terminator: 'aws4_request', not: %s.	403	请求Authorization header中的“Credential”末尾必须是“aws4_request”
SignatureDoesNotMatch	Credential should be scoped to a valid region, not:%s.	403	请求Authorization header中的“Credential”中的Region信息无效
SignatureDoesNotMatch	Credential should be scoped to correct service: %s.	403	请求Authorization header中的“Credential”中的Service信息无效
SignatureDoesNotMatch	The request signature we calculated does not match the signature you provided.	403	请求中提供的签名与实际计算结果不匹配
SignatureDoesNotMatch	Signature expired:%s.	403	签名已过期
SignatureDoesNotMatch	Date in Credential scope does not match YYYYMMDD from ISO-8601 version of date from HTTP.	403	请求Authorization header中的“Credential”中的Date应该是ISO8601基本格式，形如”YYYYMMDD“
InvalidClientTokenId	The security token included in the request is invalid.	403	请求中提供的AccessKeyId无效

业务层级错误码

err_msg	err_no	说明
未开通相应的服务	411	账号未开通服务权限
请求太频繁	510	QPS达到限制

其他说明

图片大小限制为5M,长宽像素不大于5000，图片下载时间限制为2.5s内，如果下载时间超过2.5s返回下载超时；

图片像素建议不小于256*256，太小可能会影响识别效果；

图片检测接口响应时间依赖图片的下载时间，请保证所检测图片所在的存储服务的稳定，图片建议使用金山云对象存储或者cdn做缓存等；

接口regionId仅支持cn-beijing-6；

人体识别API文档

即将上线，敬请期待

金山云签名机制文档

目录

签名机制

第一次使用金山云openAPI之前，用户需要登录金山云控制台获取账户秘钥。

账户秘钥主要有两部分组成：

- Access Key：简称AK，账户秘钥的唯一标示，可以在公网进行传递。
- Secret Key：简称SK，账户秘钥的加密秘钥，使用加密秘钥对请求进行加密。SK切记不要在公网传递。

以图像识别服务为例，以下是调用图像识别服务的原始的请求：

```
GET /?Action=ClassifyTerrorismImage&Version=2017-11-07&image_url=https://ks3-cn-beijing.ksyun.com/imgdb/vision.jpg HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: kir.api.ksyun.com
X-Amz-Date: 20171129T100303Z
```

您可使用 `Authorization` header将身份验证信息添加到请求中，`Authorization` header包含以下信息：

- 用于签名的算法 (AWS4-HMAC-SHA256)
- 凭证范围（包含您的访问密钥 ID）
- 参与签名计算的header列表
- 计算签名。该签名基于您的请求信息，由您使用金山云Secret Key生成。该签名用于向 金山云 确认您的身份。

以下是添加了签名信息的请求：

```
GET /?Action=ClassifyTerrorismImage&Version=2017-11-07&image_url=https://ks3-cn-beijing.ksyun.com/imgdb/vision.jpg HTTP/1.1

Authorization: AWS4-HMAC-SHA256 Credential=AKLTfERrGQtUQNiiirSQFW7BzQ/20171129/cn-beijing-6/kir/aws4_request, SignedHeaders=host
;x-amz-date, Signature=dd6a401930a04a78c1c1d374bd63f89a960355aef346d70b4b7d185c48a2ffe9

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

Host: kir.api.ksyun.com

X-Amz-Date: 20171129T100303Z
```

签名过程

1. 创建规范请求

规范请求伪码：

```
CanonicalRequest =
  HTTPRequestMethod + '\n' +
  CanonicalURI + '\n' +
  CanonicalQueryString + '\n' +
```

```
CanonicalHeaders + '\n' +
SignedHeaders + '\n' +
HexEncode(Hash(RequestPayload))
```

- HTTPRequestMethod是 HTTP 请求方法（GET、PUT、POST 等），后跟换行符，此处是GET
- CanonicalURI是指规范化后的HTTP的URI绝对路径，如果绝对路径为空，则使用正斜杠/
- CanonicalQueryString是指规范化的参数列表，如果不包括query string，使用空字符代替，此处是Action=ClassifyTerrorismImage&Version=2017-11-07&image_url=http://kvision.qyvideo.net/static/vision.jpg

  要构建规范查询字符串，请完成以下步骤：

  1. 按字符代码点以升序顺序对参数名称进行排序。例如，以大写字母 F 开头的参数名称排在以小写字母 b 开头的参数名称之前。
  2. 根据以下规则对每个参数名称和值进行 URI 编码：
     - 请勿对 RFC 3986 定义的任何非预留字符进行 URI 编码，这些字符包括：A-Z、a-z、0-9、连字符 (-)、下划线 (_)、句点 (.) 和波浪符 (~)。
     - 使用 %XY 对所有其他字符进行百分比编码，其中“X”和“Y”为十六进制字符（0-9 和大写字母 A-F）。例如，空格字符必须编码为 %20（不像某些编码方案那样使用“+”），扩展 UTF-8 字符必须采用格式 %XY%ZA%BC。
  3. 以排序后的列表中第一个参数名称开头，构造规范查询字符串。
  4. 对于每个参数，追加 URI 编码的参数名称，后跟等号字符 (=)，再接 URI 编码的参数值。对没有值的参数使用空字符串。
  5. 在每个参数值后追加与字符 (&)，列表中最后一个值除外。

- CanonicalHeaderss是指规范化后的HTTP header部分，此处是

```
content-type:application/x-www-form-urlencoded + '\n' +
host:kir.api.ksyun.com + '\n' +
x-amz-date:20171129T100303Z + '\n'
```

  规范化过程如下：

  1. 必须包含host header
  2. 将所有header名转换为小写形式
  3. 以header名称进行字母排序
  4. 将第3部的结果用换行符\n进行连接，最后一个header后也需要换行符

- SignedHeaders是指参加签名的header名称列表，此处为content-type;host;x-amz-date
- RequestPayload为HTTP的请求body，如果为空那么就是空字符串

2. 创建待签字符串

   1. 以算法名称开头，后跟换行符。该值是您用于计算规范请求摘要的哈希算法。对于 SHA256，算法是 AWS4-HMAC-SHA256。
   2. 追加请求日期值，后跟换行符。该日期是使用 ISO8601 基本格式以 YYYYMMDD'T'HHMMSS'Z' 格式在 x-amz-dateheader中指定的。此值必须与您在前面所有步骤中使用的值匹配。
   3. 追加凭证范围值，后跟换行符。此值是一个字符串，包含日期、目标区域、所请求的服务和小写字符形式的终止字符串（“aws4_request”）。区域和服务名称字符串必须采用 UTF-8 编码，日期必须为 YYYYMMDD 格式。请注意，日期不包括时间值。
   4. 追加CanonicalRequest的hash(sha256)结果，此处是

```
AWS4-HMAC-SHA256
20171129T100303Z
20171129/cn-beijing-6/kir/aws4_request
65e3a839cb411c645880bb07d3ff3e0a3a1ea4696d0c2e2b9de507bf63e79883
```

3. 创建签名秘钥

   伪码如下：

```
kSecret = your secret access key
kDate = HMAC("AWS4" + kSecret, Date)
kRegion = HMAC(kDate, Region)
kService = HMAC(kRegion, Service)
kSigning = HMAC(kService, "aws4_request")
```

4. 计算签名

```
signature = HexEncode(HMAC(kSigning, step 2's result))
```

5. 将签名添加到header Authorization，此处是

```
Authorization: AWS4-HMAC-SHA256 Credential=AKLTfERrGQtUQNiiirSQFW7BzQ/20171129/cn-beijing-6/kir/aws4_request, SignedHeaders=host;x-amz-date, Signature=dd6a401930a04a78c1c1d374bd63f89a960355aef346d70b4b7d185c48a2ffe9
```

OpenAPI调用方式

每个API均会提供以下参数，将SDK中相应值进行替换即可

参数名	必选	类型	说明
Action	是	string	API名称
Version	是	string	API版本号
Service	是	string	服务名，对多种Action进行逻辑分组

python实现

```
# -*- coding: utf-8 -*-
# Summary: OpenAPI
# Author: Hongchen Dou
# Time-stamp: Mon Dec  4 20:06:38 2017
import datetime
import hashlib
import hmac
import json
import sys
import requests

if sys.version_info < (3, 0):
    import urllib
else:
    from urllib import parse

def _sign(key, msg):
    return hmac.new(key, msg.encode('utf-8'), hashlib.sha256).digest()

def get_signature_key(key, dateStamp, regionName, serviceName):
    kDate = _sign(('AWS4' + key).encode('utf-8'), dateStamp)
    kRegion = _sign(kDate, regionName)
    kService = _sign(kRegion, serviceName)
    kSigning = _sign(kService, 'aws4_request')
    return kSigning

def api_request(action, version, access_key, secret_key, host='kir.api.ksyun.com', method='POST', service='kir', req_params=None,
                post_data=None):
    """
    openapi 请求
    :param host: 主机名
    :param method: 'GET'/ 'POST'
    :param service: 取值'kir'
    :param action: ClassifyImage
    :param version: API版本号
    :param access_key:
    :param secret_key:
    :param req_params: 请求参数. eg: {"image_url":"http://pica.nipic.com/2008-01-06/20081611025280_2.jpg"}
    :param post_data: json请求体，参考api文档
    """

    # ************* REQUEST VALUES *************
    region = 'cn-beijing-6'
    endpoint = 'http://{}'.format(host)
    if req_params:
        if sys.version_info < (3, 0):
            request_parameters = 'Action={}&Version={}&'.format(action, version) + urllib.urlencode(req_params)
        else:
            request_parameters = 'Action={}&Version={}&'.format(action, version) + parse.urlencode(req_params)
    else:
        request_parameters = 'Action={}&Version={}'.format(action, version)

    if access_key is None or secret_key is None:
        print('No access key is available.')
        return (400, None)

    # Create a date for headers and the credential string
    t = datetime.datetime.utcnow()
    amzdate = t.strftime('%Y%m%dT%H%M%SZ')
    datestamp = t.strftime('%Y%m%d') # Date w/o time, used in credential scope
    #amzdate = '20171129T100303Z'
    #datestamp = '20171129'

    # ************* TASK 1: CREATE A CANONICAL REQUEST *************
```

```
    # Step 1 is to define the verb (GET, POST, etc.)--already done.

    # Step 2: Create canonical URI--the part of the URI from domain to query
    # string (use '/' if no path)
    canonical_uri = '/'

    # Step 3: Create the canonical query string. In this example (a GET request),
    # request parameters are in the query string. Query string values must
    # be URL-encoded (space=%20). The parameters must be sorted by name.
    # For this example, the query string is pre-formatted in the request_parameters variable.
    canonical_querystring = request_parameters

    # Step 4: Create the canonical headers and signed headers. Header names
    # must be trimmed and lowercase, and sorted in code point order from
    # low to high. Note that there is a trailing \n.
    canonical_headers = 'content-type:application/json' + '\n' + 'host:' + host + '\n' + 'x-amz-date:' + amzdate + '\n'

    # Step 5: Create the list of signed headers. This lists the headers
    # in the canonical_headers list, delimited with ";" and in alpha order.
    # Note: The request can include any headers; canonical_headers and
    # signed_headers lists those that you want to be included in the
    # hash of the request. "Host" and "x-amz-date" are always required.
    signed_headers = 'content-type;host;x-amz-date'

    # Step 6: Create payload hash (hash of the request body content). For GET
    # requests, the payload is an empty string ("").
    if post_data is None:
        payload_hash = hashlib.sha256(''.encode('utf-8')).hexdigest()
    else:
        payload_hash = hashlib.sha256(post_data.encode('utf-8')).hexdigest()

    # Step 7: Combine elements to create canonical request
    canonical_request = method + '\n' + canonical_uri + '\n' + canonical_querystring + '\n' + canonical_headers + '\n' + signed_h
eaders + '\n' + payload_hash

    # ************* TASK 2: CREATE THE STRING TO SIGN*************
    # Match the algorithm to the hashing algorithm you use, either SHA-1 or
    # SHA-256 (recommended)
    algorithm = 'AWS4-HMAC-SHA256'
    credential_scope = datestamp + '/' + region + '/' + service + '/' + 'aws4_request'
    string_to_sign = algorithm + '\n' + amzdate + '\n' + credential_scope + '\n' + hashlib.sha256(canonical_request.encode('utf-8'
)).hexdigest()

    # ************* TASK 3: CALCULATE THE SIGNATURE *************
    # Create the signing key using the function defined above.
    signing_key = get_signature_key(secret_key, datestamp, region, service)

    # Sign the string_to_sign using the signing_key
    signature = hmac.new(signing_key, (string_to_sign).encode('utf-8'), hashlib.sha256).hexdigest()

    # ************* TASK 4: ADD SIGNING INFORMATION TO THE REQUEST *************
    # The signing information can be either in a query string value or in
    # a header named Authorization. This code shows how to use a header.
    # Create authorization header and add to request headers
    authorization_header = algorithm + ' ' + 'Credential=' + access_key + '/' + credential_scope + ', ' +  'SignedHeaders=' + si
gned_headers + ', ' + 'Signature=' + signature

    # The request can include any headers, but MUST include "host", "x-amz-date",
    # and (for this scenario) "Authorization". "host" and "x-amz-date" must
    # be included in the canonical_headers and signed_headers, as noted
    # earlier. Order here is not significant.
    # Python note: The 'host' header is added automatically by the Python 'requests' library.
    headers = {'Content-Type': 'application/json', 'X-Amz-Date': amzdate, 'Authorization':authorization_header}

    # ************* SEND THE REQUEST *************
    request_url = endpoint + '?' + canonical_querystring

    # print '\nBEGIN REQUEST++++++++++++++++++++++++++++++++++++'
    # print 'Request URL = ' + request_url
    r = requests.request(method, request_url, headers=headers, data=post_data)
    return (r.status_code, r.text)

if __name__ == "__main__":
    post_data = {
        "guard_id":"1547778774476511751",
        "image_url": "https://ks3-cn-beijing.ksyun.com/imgdb/chenshuibian.jpeg"
    }

    ak = "ak from www.ksyun.com"
    sk = "sk from www.ksyun.com"
    code, text = api_request(
        host='kir.api.ksyun.com',
        method='POST',
        service='kir',
```

```
        action='ClassifyImageGuard',
        version='2019-01-18',
        access_key=ak,
        secret_key=sk,
        post_data=json.dumps(post_data)
    )
    print(text)
```

java实现

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.HttpURLConnection;
import java.net.URL;
import java.security.MessageDigest;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.TimeZone;

public class KingeyeSdk {

    public static void main(String[] args) throws Exception {
        String access_key = "ak from www.ksyun.com";
        String secret_key = "sk from www.ksyun.com";

        String postData = "{\n    \"guard_id\": \"1547778774476511751\",\n    \"image_url\": \"https://ks3-cn-beijing.ksyun.com/im
gdb/chenshuibian.jpeg\"\n}";

        System.out.println(request("kir.api.ksyun.com", "POST", "kir",
                "ClassifyImageGuard", "2019-01-18", access_key, secret_key, postData));

    }
    public static String request(
            String host,
            String method,
            String service,
            String action,
            String version,
            String access_key,
             String secret_key,
             String postData) throws Exception {
        String contenttype = "application/json";
        String region = "cn-beijing-6";
        String endpoint = "http://" + host;
        String request_parameters = "Action=" + action + "&Version=" + version;

        Date t = new Date();
        SimpleDateFormat timeFormater = new SimpleDateFormat("yyyyMMdd'T'HHmmss'Z'");
        timeFormater.setTimeZone(TimeZone.getTimeZone("UTC"));
        String amzdate = timeFormater.format(t);
//        String amzdate = "20171211T123249Z";

        SimpleDateFormat dateFormater = new SimpleDateFormat("yyyyMMdd");
        timeFormater.setTimeZone(TimeZone.getTimeZone("UTC"));
        String datestamp = dateFormater.format(t);
//        String datestamp = "20171211";

        String canonical_uri = "/";
        String canonical_querystring = request_parameters;
        String canonical_headers = "content-type:" + contenttype + "\n" + "host:" + host + "\n" + "x-amz-date:" + amzdate + "\n";
        String signed_headers = "content-type;host;x-amz-date";

        String payload_hash = toHex(hash(postData));

        String canonical_request = method + '\n' + canonical_uri + '\n' + canonical_querystring + '\n' + canonical_headers + '\n'
 + signed_headers + '\n' + payload_hash;

        String algorithm = "AWS4-HMAC-SHA256";
        String credential_scope = datestamp + '/' + region + '/' + service + '/' + "aws4_request";
        String string_to_sign = algorithm + '\n' +  amzdate + '\n' +  credential_scope + '\n' +  toHex(hash(canonical_request));
        byte[] signing_key = getSignatureKey(secret_key, datestamp, region, service);

        String signature = toHex(hmacSHA256(string_to_sign, signing_key));

        String authorization_header = algorithm + ' ' + "Credential=" + access_key + '/' + credential_scope + ", " +  "SignedHea
ders=" + signed_headers + ", " + "Signature=" + signature;

        String request_url = endpoint + '?' + canonical_querystring;

        HttpURLConnection connection = (HttpURLConnection)new URL(request_url).openConnection();
```

```
        connection.setRequestMethod("POST");
        connection.setRequestProperty("x-amz-date", amzdate);
        connection.setRequestProperty("Authorization", authorization_header);
        connection.setRequestProperty("Content-Type", contenttype);
        connection.setDoOutput(true);

        OutputStreamWriter wr = new OutputStreamWriter(connection.getOutputStream());
        wr.write(postData);
        wr.flush();

        try {
            int code = connection.getResponseCode();
            if (code == HttpURLConnection.HTTP_OK) {
                BufferedReader reader = new BufferedReader(new InputStreamReader(connection.getInputStream(), "UTF-8"));

                String content = "";
                String line = null;
                while ((line = reader.readLine()) != null) {
                    content += line;
                }
                reader.close();
                return content;
            } else {
                System.out.println(connection.getResponseMessage() + connection.getResponseCode());
            }
        } catch (Exception e) {

        }
        return null;
    }
    public static String toHex(byte[] src) {
        StringBuilder stringBuilder = new StringBuilder("");
        if (src == null || src.length <= 0) {
            return null;
        }
        for (int i = 0; i < src.length; i++) {
            int v = src[i] & 0xFF;
            String hv = Integer.toHexString(v);
            if (hv.length() < 2) {
                stringBuilder.append(0);
            }
            stringBuilder.append(hv);
        }
        return stringBuilder.toString();
    }
    static byte[] hash(String text) throws Exception {
        if (text == null)
            text = "";
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        md.update(text.getBytes("UTF8"));
        return md.digest();
    }

    static byte[] hmacSHA256(String data, byte[] key) throws Exception {
        String algorithm="HmacSHA256";
        Mac mac = Mac.getInstance(algorithm);
        mac.init(new SecretKeySpec(key, algorithm));
        return mac.doFinal(data.getBytes("UTF8"));
    }

    static byte[] getSignatureKey(String key, String dateStamp, String regionName, String serviceName) throws Exception {
        byte[] kSecret = ("AWS4" + key).getBytes("UTF8");
        byte[] kDate = hmacSHA256(dateStamp, kSecret);
        byte[] kRegion = hmacSHA256(regionName, kDate);
        byte[] kService = hmacSHA256(serviceName, kRegion);
        byte[] kSigning = hmacSHA256("aws4_request", kService);
        return kSigning;
    }
}
```

php实现

```
<?php

$access_key = 'ak from www.ksyun.com';
$secret_key = 'sk from www.ksyun.com';

$post_data = '{
    "guard_id": "154778774476511751",
    "image_url": "https://ks3-cn-beijing.ksyun.com/imgdb/chenshuibian.jpeg"
}';

$method = "POST";
```

```
$service = "kir";
$host = "kir.api.ksyun.com";
$region = 'cn-beijing-6';
$endpoint = 'http://' . $host;
$action = 'ClassifyImageGuard';
$version = '2019-01-18';
$contenttype = 'application/json';
$request_parameters = 'Action=' . $action . '&Version=' . $version;

function sign($key, $msg) {
    return hash_hmac("sha256", $msg,  $key, true);
}
function getSignatureKey($key, $dateStamp, $regionName, $serviceName) {
    $kDate = sign('AWS4' . $key, $dateStamp);
    $kRegion = sign($kDate, $regionName);
    $kService = sign($kRegion, $serviceName);
    $kSigning = sign($kService, 'aws4_request');
    return $kSigning;
}

date_default_timezone_set('UTC');
$time = time() ;
$amzdate = date("Ymd",$time)."T".date("His",$time) . "Z";
$datestamp = date("Ymd",$time);
#$amzdate = '20180202T035703Z';
#$datestamp = '20180202';

$canonical_uri = '/';

$canonical_querystring = $request_parameters;

$canonical_headers = 'content-type:' . $contenttype . "\n" . 'host:' . $host . "\n" . 'x-amz-date:' . $amzdate . "\n";

$signed_headers = 'content-type;host;x-amz-date';

$payload_hash = hash("sha256", $post_data);

echo 'payload_hash:' . $payload_hash. "\n";

$canonical_request = $method . "\n" . $canonical_uri . "\n" . $canonical_querystring . "\n" . $canonical_headers . "\n" . $signed_h
eaders . "\n" . $payload_hash;

echo 'canonical_request:' . $canonical_request  . "\n";

$algorithm = 'AWS4-HMAC-SHA256';
$credential_scope = $datestamp . '/' . $region . '/' . $service . '/' . 'aws4_request';
$string_to_sign = $algorithm . "\n" .  $amzdate . "\n" .  $credential_scope . "\n" . hash("sha256", $canonical_request);

echo 'string_to_sign:' . $string_to_sign . "\n";

$signing_key = getSignatureKey($secret_key, $datestamp, $region, $service);

#echo $secret_key . "\n" . $datestamp . "\n" . $region . "\n" .  $service;
#echo $signing_key . "\n";

$signature = hash_hmac("sha256", $string_to_sign, $signing_key);

echo 'signature:' . $signature . "\n";

#echo "sign:" . $signature . "\n";
$authorization_header = $algorithm . ' ' . 'Credential=' . $access_key . '/' . $credential_scope . ', ' .  'SignedHeaders=' . $sig
ned_headers . ', ' . 'Signature=' . $signature;

echo 'authorization_header:' . $authorization_header . "\n";

$request_url = $endpoint . '?' . $canonical_querystring;

$ch = curl_init($request_url);
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_POSTFIELDS, $post_data);
curl_setopt($ch, CURLOPT_HTTPHEADER, array("x-amz-date:$amzdate", "Authorization:$authorization_header", "Content-Type:$contenttyp
e"));

echo "\n\n-------------result--------------\n\n";

$result = json_decode(curl_exec($ch));

var_dump($result);

?>
```

golang实现

```
package main

import (
    "fmt"
    "crypto/hmac"
    "crypto/sha256"
    "encoding/json"
    "net/url"
    "time"
    "net/http"
    "strings"
    "io/ioutil"
)

func _sign(key []byte, msg string) []byte {
    h := hmac.New(sha256.New, key)
    h.Write([]byte(msg))
    return h.Sum(nil)
}

func get_signature_key(key string, dateStamp string, regionName string, serviceName string) []byte {
    kDate := _sign([]byte("AWS4" + key), dateStamp)
    kRegion := _sign(kDate, regionName)
    kService := _sign(kRegion, serviceName)
    kSigning := _sign(kService, "aws4_request")
    return kSigning
}

func api_request(action, version, access_key, secret_key, host, method, service string,
    req_params map[string]string, post_data map[string]interface{}) (int, string) {

    // ************* REQUEST VALUES *************
    region := "cn-beijing-6"
    endpoint := "http://" + host
    request_parameters := "Action=" + action + "&Version=" + version
    if req_params != nil {
        params := url.Values{}
        for key := range req_params {
            params.Add(key, req_params[key])
        }
        request_parameters +=  "&" + params.Encode()
    }
    if access_key == "" || secret_key == "" {
        fmt.Println("No access key is available.")
        return 400, ""
    }
    t := time.Now().UTC()
    amzdate := t.Format("20060102T150405Z")
    datestamp := t.Format("20060102")

    // ************* TASK 1: CREATE A CANONICAL REQUEST *************
    // Step 1 is to define the verb (GET, POST, etc.)--already done.

    // Step 2: Create canonical URI--the part of the URI from domain to query
    // string (use '/' if no path)
    canonical_uri := "/"

    // Step 3: Create the canonical query string. In this example (a GET request),
    // request parameters are in the query string. Query string values must
    // be URL-encoded (space=%20). The parameters must be sorted by name.
    // For this example, the query string is pre-formatted in the
    // request_parameters variable.
    canonical_querystring := request_parameters

    // Step 4: Create the canonical headers and signed headers. Header names
    // must be trimmed and lowercase, and sorted in code point order from
    // low to high. Note that there is a trailing \n.
    canonical_headers := "content-type:application/json" + "\n" + "host:" + host + "\n" + "x-amz-date:" + amzdate + "\n"

    // Step 5: Create the list of signed headers. This lists the headers
    // in the canonical_headers list, delimited with ";" and in alpha order.
    // Note: The request can include any headers; canonical_headers and
    // signed_headers lists those that you want to be included in the
    // hash of the request. "Host" and "x-amz-date" are always required.
    signed_headers := "content-type;host;x-amz-date"

    // Step 6: Create payload hash (hash of the request body content). For GET
    // requests, the payload is an empty string ("").
    h := sha256.New()
    var json_data string
    if post_data == nil {
        h.Write([]byte(""))
    } else {
        data, _ := json.Marshal(post_data)
```

```
        json_data = string(data)
        //fmt.Println(string(json_data))
        //data := "{\"business\": [\"porn\"], \"image_urls\": [\"https://ks3-cn-beijing.ksyun.com/imgdb/chenshuibian.jpeg\"]}"
        //fmt.Println(data)
        h.Write([]byte(json_data))
    }
    payload_hash := fmt.Sprintf("%x", h.Sum(nil))
    //fmt.Printf("%x\n", payload_hash)

    // Step 7: Combine elements to create canonical request
    canonical_request := method + "\n" + canonical_uri + "\n" + canonical_querystring + "\n" + canonical_headers + "\n" + signed_
headers + "\n" + payload_hash

    // ************* TASK 2: CREATE THE STRING TO SIGN*************
    // Match the algorithm to the hashing algorithm you use, either SHA-1 or
    // SHA-256 (recommended)
    algorithm := "AWS4-HMAC-SHA256"
    credential_scope := datestamp + "/" + region + "/" + service + "/" + "aws4_request"
    ha := sha256.New()
    ha.Write([]byte(canonical_request))
    hash := fmt.Sprintf("%x", ha.Sum(nil))
    string_to_sign := algorithm + "\n" + amzdate + "\n" + credential_scope + "\n" + hash

    // ************* TASK 3: CALCULATE THE SIGNATURE *************
    // Create the signing key using the function defined above.
    signing_key := get_signature_key(secret_key, datestamp, region, service)

    // Sign the string_to_sign using the signing_key
    s := hmac.New(sha256.New, signing_key)
    s.Write([]byte(string_to_sign))
    signature := fmt.Sprintf("%x", s.Sum(nil))

    // ************* TASK 4: ADD SIGNING INFORMATION TO THE REQUEST ***********
    // The signing information can be either in a query string value or in
    // a header named Authorization. This code shows how to use a header.
    // Create authorization header and add to request headers
    authorization_header := algorithm + " " + "Credential=" + access_key + "/" + credential_scope + ", " + "SignedHeaders=" + si
gned_headers + ", " + "Signature=" + signature

    // ************* SEND THE REQUEST *************
    request_url := endpoint + "?" + canonical_querystring
    client := &http.Client{}
    req, _ := http.NewRequest(method, request_url, strings.NewReader(json_data))
    req.Header.Set("Content-Type", "application/json")
    req.Header.Set("X-Amz-Date", amzdate)
    req.Header.Set("Authorization", authorization_header)
    resp, _ := client.Do(req)
    defer resp.Body.Close()
    body, _ := ioutil.ReadAll(resp.Body)
    code := resp.StatusCode

    return code, string(body)
}

func main() {
    post_data := map[string]interface{}{
        "guard_id": "154778774476511751",
        "image_url": "https://ks3-cn-beijing.ksyun.com/imgdb/chenshuibian.jpeg",
    }

    ak := "ak from www.ksyun.com"
    sk := "sk from www.ksyun.com"

    code, text := api_request(
        "ClassifyImageGuard",
        "2019-01-18",
        ak,
        sk,
        "kir.api.ksyun.com",
        "POST",
        "kir",
        nil,
        post_data,
    )
    fmt.Println(code)
    fmt.Println(text)
}
```

签名错误码

错误代码(Code)	错误消息(Message)	状态码	说明
IncompleteSignature	Date must be in ISO-8601 'basic format'. Got '%s'. See http://en.wikipedia.org/wiki/ISO_8601.	400	Date必须符合ISO_8601基本格式，参考：http://en.wikipedia.org/wiki/ISO_8601
IncompleteSignature	KSC query-string parameters must include %s. Re-examine the query-string parameters.	400	查询条件中缺少签署信息，查询条件中必须包含”X-Amz-Algorithm“、”X-Amz-Credential“、”X-Amz-SignedHeaders“、”X-Amz-Date“信息
IncompleteSignature	Unsupported ksc 'algorithm': %s.	400	只支持如下签名算法：AWS4-HMAC-SHA256
IncompleteSignature	Authorization header requires 'Credential' parameter. Authorization=%s.	400	请求Authorization header中需要包含“Credential”参数
IncompleteSignature	Credential must have exactly 5 slash-delimited elements, e.g. accesskeyid/date/region/service/aws4_request, got: %s.	400	请求Authorization header中中“Credential”至少包含5项以斜杠分隔的元素，如：keyid/date/region/service/aws4_request
IncompleteSignature	Authorization header format error.	400	请求Authorization header的格式错误
IncompleteSignature	Authorization header requires existence of either a 'X-Amz-Date' or a 'Date' header, Authorization=%s	400	请求中缺少“X-Amz-Date”或者“Date” header信息
IncompleteSignature	Authorization header requires 'Signature' parameter. Authorization=%s	400	请求Authorization header中缺少“Signature”信息
IncompleteSignature	Authorization header requires 'SignedHeaders' parameter. Authorization=%s	400	请求Authorization header中缺少“SignedHeaders”信息
MissingAuthenticationToken	Request is missing 'Host' header.	403	请求header中缺少Host
MissingAuthenticationToken	Request is missing Authentication Token.	403	请求header中缺少认证token
MissingAuthenticationToken	%s not in Http Header.	403	%s不在Http header中
SignatureDoesNotMatch	Host' must be a 'SignedHeader' in the Authorization.	403	请求的SignedHeader中必须包含Host
SignatureDoesNotMatch	Credential should be scoped with a valid terminator: 'aws4_request', not: %s.	403	请求Authorization header中的“Credential”末尾必须是“aws4_request”
SignatureDoesNotMatch	Credential should be scoped to a valid region, not:%s.	403	请求Authorization header中的“Credential”中的Region信息无效
SignatureDoesNotMatch	Credential should be scoped to correct service: %s.	403	请求Authorization header中的“Credential”中的Service信息无效
SignatureDoesNotMatch	The request signature we calculated does not match the signature you provided.	403	请求中提供的签名与实际计算结果不匹配
SignatureDoesNotMatch	Signature expired:%s.	403	签名已过期
SignatureDoesNotMatch	Date in Credential scope does not match YYYYMMDD from ISO-8601 version of date from HTTP.	403	请求Authorization header中的“Credential”中的Date应该是ISO8601基本格式，形如”YYYYMMDD“
InvalidClientTokenId	The security token included in the request is invalid.	403	请求中提供的AccessKeyId无效