目录

公共参数

公共参数是每个OpenAPI接口都需要使用的请求参数。支持GET和POST两种HTTP方法。

- GET请求：放在url的query。面。例如：{sevice}.api.ksyun.com?{业务参数}&{公共参数}。
- POST请求：放在http body里面. 例如：{业务参数}&{公共参数}。

名称	类型	是否必须参数	描述
Accesskey	String	是	用户在控制台创建的Accesskey，获取方式请参考获取AK/SK
Service	String	是	服务名称，参考请求结构章节说明
Action	String	是	操作接口名，与调用的具体openAPI相关
Version	String	是	接口版本号，与具体的操作接口有关
Timestamp	String	是	时间，UTC格式，例如：2019-08-13T17:18:36Z
SignatureVersion	String	是	签名版本号，固定值：1.0
SignatureMethod	String	是	签名算法，固定值：HMAC-SHA256
Signature	String	是	签名，具体请查看签名机制
Region	String	否	区域，不传默认cn-beijing-6。不同服务支持的region不同，参考请求结构章节说明
SecurityToken	String	否	安全令牌，在使用临时AK/SK需要传该字段，如果使用GET方法，需要对该字段进行urlencode 获取方式参考文档获取角色的临时身份
DryRun	Boolean	否	检查当前调用者是否有权限执行相关操作，而不是真的调用执行相关操作
Format	String	否	指定响应格式，固定值：json

签名机制

1.构造规范化请求字符串

第一步：请求参数排序

- 请求参数包括公共参数和业务参数，不包含公共参数Signature。
- 排序规则以参数名按照字典排序。

第二步：请求参数编码 使用UTF-8字符集按照RFC3986规则编码请求参数和参数取值，编码规则如下：

- 对于字符A ~ Z、a ~ z、0 ~ 9以及字符-、_、.和~不编码。
- 对于其他字符编码成%XY的格式，其中XY是字符对应ASCII码的16进制(大写)。

  不同语言实现的URLEncode方式不同，为了得到RFC3986规则的编码，分别说明如下：

  - 如果您使用的是Java中的java.net.URLEncoder，可以先用URLEncoder.encode方法编码，随后将编码后的字符中加号（+）替换为%20、星号（*）替换为%2A、%7E替换为波浪号（~）即可；
  - 如果您使用的是golang中的net/url，可以用url.Values.Encode方法编码，随后将编码后的字符中加号（+）替换为%20即可；
  - 如果您使用的是php中的rawurlencode方法，不需要替换；
  - 如果您使用的是python3中的urllib.request，可以用urllib.request.quote方法，需要指定字符串中的波浪号（~）不编码，例如urllib.request.quote(str, '~')，str为待编码字符串；

第三步：请求参数拼接成CanonicalizedQueryString 每对URLEncode后的参数名称和参数值，用=进行连接。每对之间使用&进行连接。得到规范化请求字符串CanonicalizedQueryString。

2.计算签名。

sign = hash_hmac(‘sha256’, CanonicalizedQueryString, sk)

sign值为签名算法返回的16进制格式小写字符串

签名样例：

5346bfebeb3f2162e2459e09a52b640584e5f1aa5012b15c6b85388680d4663e 计算签名时使用的sk为Accesskey对应的秘钥，使用的哈希算法是：HMAC-SHA256。

3.将签名值作为Signature参数值添加到请求参数中。

签名过程示例

该样例以iam服务的CreateUser为例，点击查看该API文档。

1、构造规范化请求字符串：CanonicalizedQueryString

```
//请求参数数组:
array (
    'Accesskey'=> 'AKLTXQVFOpOmS6aahIrD5r0B3Q', //公共参数
    'Service'=> 'iam', //公共参数
    'Action'=> 'CreateUser', //公共参数
    'Version'=> '2015-11-01', //公共参数
    'Timestamp'=> '2021-08-12T02:47:36Z', //公共参数
    'SignatureVersion'=> '1.0', //公共参数
    'SignatureMethod'=> 'HMAC-SHA256', //公共参数
    'UserName' => 'Ttest', //业务参数
    'RealName' => '周四测试', //业务参数
    'Email' => 'zsce@kkingsoft.com', //业务参数
    'Remark' => '~ce shi*%#|+', //业务参数
)
```

第一步：请求参数排序

```
//排序结果如下:
array (
    'Accesskey' => 'AKLTXQVFOpOmS6aahIrD5r0B3Q',
    'Action' => 'CreateUser',
    'Email' => 'zsce@kkingsoft.com',
    'RealName' => '周四测试',
    'Remark' => '~ce shi*%#|+',
    'Service' => 'iam',
    'SignatureMethod' => 'HMAC-SHA256',
    'SignatureVersion' => '1.0',
    'Timestamp' => '2021-08-12T02:47:36Z',
    'UserName' => 'Ttest',
    'Version' => '2015-11-01',
)
```

第二步：请求参数编码

```
//编码结果如下:
array (
    'Accesskey' => 'AKLTXQVFOpOmS6aahIrD5r0B3Q',
    'Action' => 'CreateUser',
    'Email' => 'zsce%40kkingsoft.com',
    'RealName' => '%E5%91%A8%E5%9B%9B%E6%B5%8B%E8%AF%95',
    'Remark' => '~ce%20shi%2A%25%23%7C%2B',
    'Service' => 'iam',
    'SignatureMethod' => 'HMAC-SHA256',
    'SignatureVersion' => '1.0',
    'Timestamp' => '2021-08-12T02%3A47%3A36Z',
    'UserName' => 'Ttest',
    'Version' => '2015-11-01',
)
```

第三步：请求参数拼接成CanonicalizedQueryString

```
//拼接结果如下
Accesskey=AKLTXQVFOpOmS6aahIrD5r0B3Q&Action=CreateUser&Email=zsce%40kkingsoft.com&RealName=%E5%91%A8%E5%9B%9B%E6%B5%8B%E8%AF%95&Remark=~ce%20shi%2A%25%23%7C%2B&Service=iam&SignatureMethod=HMAC-SHA256&SignatureVersion=1.0&Timestamp=2021-08-12T02%3A47%3A36Z&UserName=Ttest&Version=2015-11-01
```

2、计算签名

```
sk = 'OMovU5PTLh6y9E9Ioe3K411jt99VqyQSBXgAcDYlo49R3lvUIzb6e/efZCFDmtFlzw=='
CanonicalizedQueryString = "Accesskey=AKLTXQVFOpOmS6aahIrD5r0B3Q&Action=CreateUser&Email=zsce%40kkingsoft.com&RealName=%E5%91%A8%E5%9B%9B%E6%B5%8B%E8%AF%95&Remark=~ce%20shi%2A%25%23%7C%2B&Service=iam&SignatureMethod=HMAC-SHA256&SignatureVersion=1.0&Timestamp=2021-08-12T02%3A47%3A36Z&UserName=Ttest&Version=2015-11-01"
//sign = hash_hmac('sha256', CanonicalizedQueryString, sk)
sign: fc9088ab845949dac4040be9b7ce7859068b5c21d4c400fec8ee0cefb777f659
```

3、将签名值作为Signature参数值添加到请求参数中

```
//请求示例:
```

```
curl -s -L -X POST 'iam.api.ksyun.com' \
-H 'Accept: application/json' \
-H 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'Accesskey=AKLTXQVFOpOmS6aahIrD5rOB3Q' \
--data-urlencode 'Service=iam' \
--data-urlencode 'Action=CreateUser' \
--data-urlencode 'Version=2015-11-01' \
--data-urlencode 'Timestamp=2021-08-12T02:47:36Z' \
--data-urlencode 'SignatureVersion=1.0' \
--data-urlencode 'SignatureMethod=HMAC-SHA256' \
--data-urlencode 'UserName=Ttest' \
--data-urlencode 'RealName=周四测试' \
--data-urlencode 'Email=zsce@kkingsoft.com' \
--data-urlencode 'Remark=~ce shi*%#|+' \
--data-urlencode 'Signature=fc9088ab845949dac4040be9b7ce7859068b5c21d4c400fec8ee0cefb777f659'
```

PHP

PHP签名DEMO如下：

```
//使用时把sign方法拷贝到代码工程内即可
function sign($params, $secret_key)
{
    ksort($params, SORT_STRING);

    $str_encode = '';
    foreach($params as $k=>$v) {
        $str_encode .= rawurlencode($k).'='.rawurlencode($v).'&';
    }
    $str_encode = substr($str_encode, 0, -1);

    return hash_hmac("sha256", $str_encode, $secret_key);
}

$arr =[
 'Accesskey'=> 'AKLTXQVFOpOmS6aahIrD5rOB3Q',
 'Service'=> 'iam',
 'Action'=> 'CreateUser',
 'Version'=> '2015-11-01',
 'Timestamp'=> '2021-08-12T02:47:36Z',
 'SignatureVersion'=> '1.0',
 'SignatureMethod'=> 'HMAC-SHA256',
 'UserName' => 'Ttest',
 'RealName' => '周四测试',
 'Email' => 'zsce@kkingsoft.com',
 'Remark' => '~ce shi*%#|+',
 ];
$sk = 'OMovU5PTLh6y9E9Ioe3K411jt99VqyQSBXgAcDYlo49R3lvUIzb6e/efZCFDmtFlzw==';
$signature = sign($arr, $sk);

var_dump($signature); //fc9088ab845949dac4040be9b7ce7859068b5c21d4c400fec8ee0cefb777f659
```

JAVA

java签名DEMO如下：

```
package com.ksyun.util;

import org.apache.commons.codec.digest.HmacAlgorithms;
import org.apache.commons.codec.digest.HmacUtils;

import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;
import java.nio.charset.StandardCharsets;
import java.util.HashMap;
import java.util.Map;
import java.util.TreeMap;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/***
    使用时，把测试相关方法删掉。测试方法包括：main，test1，test2
    把这个类拷贝到自己的工程内
    签名调用：SignatureUtils.signature(params, secretKey)
**/
/**
```

```
 * @author chenxushao@kingsoft.com
 */
public class SignatureUtils {

    private static final Pattern ENCODED_CHARACTERS_PATTERN;

    static {
        StringBuilder pattern = new StringBuilder();
        pattern.append(Pattern.quote("+"))
            .append("|")
            .append(Pattern.quote("*"))
            .append("|")
            .append(Pattern.quote("%7E"))
            .append("|")
            .append(Pattern.quote("%2F"));
        ENCODED_CHARACTERS_PATTERN = Pattern.compile(pattern.toString());
    }

    /**
     * 注意空格( )会编码成+号，所以+最后需要替换成%20（%20为空格）
     * 在URLEncode后需对三种字符替换：加号（+）替换成 %20、星号（*）替换成 %2A、 %7E 替换回波浪号（~）
     */
    private static String urlEncode(final String value, final boolean path) {
        if (value == null) {
            return "";
        }

        try {

            String encoded = URLEncoder.encode(value, StandardCharsets.UTF_8.name());
            Matcher matcher = ENCODED_CHARACTERS_PATTERN.matcher(encoded);
            StringBuffer buffer = new StringBuffer(encoded.length());

            while (matcher.find()) {
                String replacement = matcher.group(0);
                if ("+".equals(replacement)) {
                    replacement = "%20";
                } else if ("*".equals(replacement)) {
                    replacement = "%2A";
                } else if ("%7E".equals(replacement)) {
                    replacement = "~";
                } else if (path && "%2F".equals(replacement)) {
                    replacement = "/";
                }

                matcher.appendReplacement(buffer, replacement);
            }
            matcher.appendTail(buffer);
            return buffer.toString();
        } catch (UnsupportedEncodingException ex) {
            throw new RuntimeException(ex);
        }
    }

    private static String getCanonicalizedQueryString(Map<String, Object> params) {
        final Map<String, String> tmap = new TreeMap<>();
        for (Map.Entry<String, Object> entry : params.entrySet()) {
            String key = entry.getKey();
            Object value = entry.getValue();
            String encodedParamName = urlEncode(key, false);
            String encodedValues = urlEncode(value.toString(), false);
            tmap.put(encodedParamName, encodedValues);
        }

        final StringBuilder sb = new StringBuilder();
        for (Map.Entry<String, String> entry : tmap.entrySet()) {
            if (sb.length() > 0) {
                sb.append("&");
            }
            sb.append(entry.getKey()).append('=').append(entry.getValue());
        }
        return sb.toString();
    }

    public static String signature(Map<String, Object> params, String secretKey) {
        String canonicalizedQueryString = getCanonicalizedQueryString(params);
        return new HmacUtils(HmacAlgorithms.HMAC_SHA_256, secretKey).hmacHex(canonicalizedQueryString);
    }

    public static void test2() {
        String accesskey = "AKLTXQVFOpOmS6aahIrD5rOB3Q";
        String secretKey = "OMovU5PTLh6y9E9Ioe3K411jt99VqyQSBXgAcDYlo49R3lvUIzb6e/efZCFDmtFlzw==";

        Map<String, Object> params = new HashMap<>();
```

```
        params.put("Accesskey", accesskey);//公共参数
        params.put("Service", "iam");//公共参数
        params.put("Action", "CreateUser");//公共参数
        params.put("Version", "2015-11-01");//公共参数
        params.put("Timestamp", "2021-08-12T02:47:36Z");//公共参数
        params.put("SignatureVersion", "1.0");//公共参数
        params.put("SignatureMethod", "HMAC-SHA256");//公共参数
        params.put("UserName", "Ttest"); //api特有参数
        params.put("RealName", "周四测试");//api特有参数
        params.put("Email", "zsce@kkingsoft.com");//api特有参数
        params.put("Remark", "~ce shi*%#|+");//api特有参数

        System.out.println(signature(params, secretKey));
    }

    public static void test1() {
        String accesskey = "AKLTXQVFOpOmS6aahIrD5rOB3Q";
        String secretKey = "OMovU5PTLh6y9E9Ioe3K411jt99VqyQSBXgAcDYlo49R3lvUIzb6e/efZCFDmtFlzw==";

        Map<String, Object> params = new HashMap<>();
        params.put("Accesskey", accesskey); //公共参数
        params.put("Service", "iam"); //公共参数
        params.put("Action", "GetUser"); //公共参数
        params.put("Version", "2015-11-01"); //公共参数
        params.put("Timestamp", "2021-08-06T07:45:36Z"); //公共参数
        params.put("SignatureVersion", "1.0"); //公共参数
        params.put("SignatureMethod", "HMAC-SHA256"); //公共参数
        params.put("UserName", "freestest"); //api特有参数

        System.out.println(signature(params, secretKey));
    }

    public static void main(String[] args) {
        test1();//签名结果：9294d873d0f921bed24b6089708b66fbdfc4a6ea0eb30ad21e73ce603b82fbb7
        test2();//签名结果：fc9088ab845949dac4040be9b7ce7859068b5c21d4c400fec8ee0cefb777f659
    }
}
```

Go

Go签名DEMO如下：

```
package main

import (
    "crypto/hmac"
    "crypto/sha256"
    "encoding/hex"
    "fmt"
    "net/url"
    "strings"
)
//使用时把sign方法拷贝到代码工程内即可
func sign(params url.Values, sk string) string {
    strEncode := params.Encode()
    strEncode = strings.Replace(strEncode, "+", "%20", -1)
    h := hmac.New(sha256.New, []byte(sk))
    h.Write([]byte(strEncode))
    return hex.EncodeToString(h.Sum(nil))
}

func main() {
    params := url.Values{}
    params.Add("Accesskey", "AKLTXQVFOpOmS6aahIrD5rOB3Q")
    params.Add("Service", "iam")
    params.Add("Action", "CreateUser")
    params.Add("Version", "2015-11-01")
    params.Add("Timestamp", "2021-08-12T02:47:36Z") //通过time.Now().UTC().Format("2006-01-02T15:04:05Z")实现
    params.Add("SignatureVersion", "1.0")
    params.Add("SignatureMethod", "HMAC-SHA256")
    params.Add("UserName", "Ttest")
    params.Add("RealName", "周四测试")
    params.Add("Email", "zsce@kkingsoft.com")
    params.Add("Remark", "~ce shi*%#|+")

    sk := "OMovU5PTLh6y9E9Ioe3K411jt99VqyQSBXgAcDYlo49R3lvUIzb6e/efZCFDmtFlzw=="
    signature := sign(params, sk)

    fmt.Println(signature) // fc9088ab845949dac4040be9b7ce7859068b5c21d4c400fec8ee0cefb777f659

}
```

Python

python3 签名DEMO如下：

```
import sys, os, base64, datetime, hashlib, hmac
import urllib.request

# 使用时把sign方法拷贝到代码工程内即可
def sign(params, secret_key):
    str_encode = ''
    param_keys = sorted(params.keys())
    for key in param_keys:
        str_encode += urllib.request.quote(key, '~') + '=' + urllib.request.quote(str(params[key]), '~') + '&'

    return hmac.new(bytes(secret_key, 'utf-8'), bytes(str_encode[:-1], 'utf-8'), hashlib.sha256).hexdigest()

arr = {
    'Accesskey' : 'AKLTXQVFOpOmS6aahIrD5rOB3Q',
    'Service' : 'iam',
    'Action' : 'CreateUser',
    'Version' : '2015-11-01',
    'Timestamp' : '2021-08-12T02:47:36Z', # 使用如下的方式产生UTC格式的时间，datetime.datetime.utcnow()
    'SignatureVersion' : '1.0',
    'SignatureMethod' : 'HMAC-SHA256',
    'UserName' : 'Ttest',
    'RealName' : '周四测试',
    'Email' : 'zsce@kkingsoft.com',
    'Remark' : '~ce shi*%#|+'
}
sk = 'OMovU5PTLh6y9E9Ioe3K411jt99VqyQSBXgAcDYlo49R3lvUIzb6e/efZCFDmtFlzw=='
signature = sign(arr, sk)

print(signature) #结果：fc9088ab845949dac4040be9b7ce7859068b5c21d4c400fec8ee0cefb777f659
```