目录

目录 1
API概览 3
人员管理相关接口 3
访问密钥管理相关接口 3
策略及授权管理相关接口 3
角色管理 3
获取角色的临时身份 3
MFA相关接口 4
项目相关接口 4
请求结构 4
请求结构 4
公共参数 4
返回结果 5
调用成功 5
调用失败 5
签名机制 6
签名过程 6
1.构造规范化请求字符串 6
2.计算签名。 6
3.将签名值作为Signature参数值添加到请求参数中。 6
签名过程示例 6
签名DEMO 7
php 7
java 8
golang 10
python3 10
KRN查询列表 10
User 11
描述 11
内容 11
Role 12
描述 12
内容 12
LoginProfile 12
描述 12
内容 12
AccessKey 12
描述 13
内容 13
AccessKeyMetadata 13
描述 13
内容 13
Policy 13
描述 13
内容 13
PolicyVersion 14
描述 14
内容 14
AttachedPolicy 14
描述 14

内容 14
PolicyUser 15
描述 15
内容 15
SummaryMap 15
描述 15
内容 15
PasswordPolicy 15
描述 15
内容 15
VirtualMFADevice 16
描述 16
内容 { 16
AssumeRoleResult 16
描述 16
AssumedRoleUser内容 16
Credentials内容 17
其他内容 17
UserSession 17
描述 17
内容 17

API概览

人员管理相关接口

接口功能	Action Name
新建子用户	CreateUser
查询子用户列表	ListUsers
更新子用户基本信息	UpdateUser
查询子用户基本信息	GetUser
删除子用户	DeleteUser
子用户修改密码	ChangePassword
更新子用户登录配置	UpdateLoginProfile
查询登录配置信息	GetLoginProfile
获取子用户ak最后使用时间	ListAllUserAccessKeys

访问密钥管理相关接口

接口功能	Action Name
生成访问密钥	CreateAccessKey
查询访问密钥列表	ListAccessKeys
更新访问密钥	UpdateAccessKey
删除访问密钥	DeleteAccessKey

策略及授权管理相关接口

接口功能	Action Name
解绑IAM用户策略	DetachUserPolicy
查询用户附加策略列表	ListAttachedUserPolicies
查询策略版本列表	ListPolicyVersions
设定默认策略版本	SetDefaultPolicyVersion
附加用户策略	AttachUserPolicy
删除策略版本内容	DeletePolicyVersion
查询策略版本内容	GetPolicyVersion
新建策略版本内容	CreatePolicyVersion
查询策略信息列表	ListPolicies
查询策略元数据信息	GetPolicy
删除策略	DeletePolicy
新建策略	CreatePolicy
查询策略关联实体列表	ListEntitiesForPolicy
更新策略描述	UpdatePolicy

角色管理

接口功能	Action Name
创建角色	CreateRole
删除角色	DeleteRole
查询角色基本信息	GetRole
查询账号的角色列表	ListRoles
附加角色访问策略	AttachRolePolicy
分离角色访问策略	DetachRolePolicy
查询角色附加的策略列表	ListAttachedRolePolicies
更新角色的信任关系	UpdateRoleTrustAccounts
更新角基本描述	UpdateRole

获取角色的临时身份

接口功能	Action Name

获取角色的临时身份 AssumeRole

MFA相关接口

接口功能	Action Name
查询虚拟设备列表	ListVirtualMFADevices
激活虚拟设备	EnableMFADevice
解绑虚拟设备	DeactivateMFADevice
获取虚拟设备	GetVirtualMFADevice

项目相关接口

接口功能	Action Name
创建项目	CreateProject
获取主账号/子用户下所有项目列表	GetAccountAllProjectList
获取项目资源列表	GetProjectInstanceList
更新实例项目	UpdateInstanceProjectId
查询项目成员列表	ListProjectMember
删除项目成员	DeleteProjectMember
添加项目成员	AddProjectMember

请求结构

请求结构

客户调用金山云**访问控制**服务的openAPI接口是通过向指定服务地址发送请求，并按照openAPI文档说明在请求中添加相应的公共参数和接口参数来完成的。

访问控制openAPI的请求结构组成如下：

1. 服务地址

   访问控制的服务接入地址为：iam.api.ksyun.com

2. 通信协议

   支持通过 HTTP 或 HTTPS 两种方式进行请求通信，推荐使用安全性更高的 HTTPS方式发送请求。

3. 请求方法

   访问控制的openAPI同时支持GET和POST请求，推荐使用**GET**请求方式。

4. 请求参数

   金山云openAPI请求包含两类参数：**公共请求参数**和**业务请求参数**。其中，公共请求参数是每个接口都要用到的请求参数，具体可参见公共参数小节；业务请求参数是各个接口所特有的，具体见各个接口的“请求参数”描述。

5. 字符编码

   请求及返回结果都使用UTF-8字符集进行编码。

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： iam.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
Service	String	是	服务名称，提供open-api的服务英文简称，访问控制服务固定值iam
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，访问控制服务接口当前只支持一个版本，即2015-11-01
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，默认cn-beijing-6。不同服务支持的region不同，访问控制服务使用的region为cn-beijing-6
SecurityToken	String	否	安全令牌，在使用临时ak(STS 角色扮演获取的ak/sk),需要传该字段，如果使用GET方法，需要对该字段进行urlencode
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

示例

```
https://iam.api.ksyun.com/?
Action=ListUsers
&Accesskey=AKLTAGFB7-e_Qtest
&Service=iam
&Version=2015-11-01
&Timestamp=2020-08-05T07:16:22Z
&SignatureVersion=1.0
&SignatureMethod=HMAC-SHA256
&Signature=6f00100528da6630b786a9326813916c4b6e77dgagda2d4e4a1b05c7e
&业务参数
```

返回结果

调用金山云的openAPI服务，调用成功，返回的HTTP状态码（Status）为200；调用失败，返回4xx 或5xx的HTTP状态码（Status）。

金山云的**访问控制**服务的调用返回的数据格式支持xml和json两种，默认返回xml格式，可通过设置HTTP Header Accept=application/json来改变返回数据格式。

调用成功

- xml格式示例

```
<!--结果的根结点-->
<接口名称+Response>
    <ResponseMetadata>
        <RequestId>4C467B38-3910-447D-87BC-AC049166F216</RequestId>
    </ResponseMetadata>
    <!--返回结果数据-->
</接口名称+Response>
```

- json格式示例

```
{
    "RequestId": "4C467B38-3910-447D-87BC-AC049166F216"
    /*返回结果数据*/
}
```

调用失败

调用接口失败，不会返回结果数据；HTTP请求返回一个4xx或5xx的HTTP状态码，返回的HTTP消息体中包含具体的错误代码(code)及错误信息(message)；与调用成功一样还包含请求ID(RequestId)。

- json格式示例

```
{
    "RequestId": "68093a99-2f63-4f39-8f70-3047ab8ecb5b",
    "Error": {
        "Type": "Sender",
        "Code": "InvalidParameterValue",
        "Message": "An invalid or out-of-range value was supplied for the input parameter PathPrefix."
    }
}
```

- 当请求出错时建议可以先根据该错误码在公共错误码(所有业务都可能出现的错误码为公共错误码)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。

- 如果您找不到错误原因时，可以联系金山云客服，并提供RequestId，以便我们尽快帮您解决问题。

签名机制

签名过程

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=AKLTXQVFOpOmS6aahIrD5r0B3Q' \
--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'
```

签名DEMO

php

```
//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'=> 'AKLTXQVF0pOmS6aahIrD5rOB3Q',
 '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

```
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
    }
}
```

golang

```
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", "AKLTXQVFOpOmS6aahIrD5r0B3Q")
    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

}
```

python3

```
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' : 'AKLTXQVFOpOmS6aahIrD5r0B3Q',
    '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
```

KRN查询列表

资源名称	英文名称	资源KRN
实例	instance	krn:ksc:kec:*region*:*account-id*:instance/*instance-id*
云主机镜像	image	krn:ksc:kec:*region*::image/*image-id*
卷	volume	krn:ksc:kec:*region*:*account-id*:volume/*volume-id*
快照	snapshot	krn:ksc:kec:*region*:*account-id*:snapshot/*snapshot-id*
安全组	security-group	krn:ksc:vpc:*region*:*account-id*:security-group/*security-group-id*
子网	subnet	krn:ksc:vpc:*region*:*account-id*:subnet/*subnet-id*
网络接口	network-interface	krn:ksc:vpc:*region*:*account-id*:network-interface/*network-interface-id*
虚拟私有网络	vpc	krn:ksc:vpc:*region*:*account-id*:vpc/*vpc-id*
网络ACL	network-acl	krn:ksc:vpc:*region*:*account-id*:network-acl/*network-acl-id*
路由	route	krn:ksc:vpc:*region*:*account-id*:route/*route-id*
本地地址转换	nat	krn:ksc:vpc:*region*:*account-id*:nat/*nat-id*
隧道网关	tunnel	krn:ksc:vpc:*region*:*account-id*:tunnel/*tunnel-id*
对等连接	vpc-peering-connection	krn:ksc:vpc:*region*:*account-id*:vpc-peering-connection/*vpc-peering-connection-id*
负载均衡器	loadbalancer	krn:ksc:slb:*region*:*account-id*:loadbalancer/*load-balancer-id*
监听器	listener	krn:ksc:slb:*region*:*account-id*:listener/*listener-id*
主账户	root	krn:ksc:iam::*account-id*:root
用户	user	krn:ksc:iam::*account-id*:user/*user-name*
角色	role	krn:ksc:iam::*account-id*:role/*role-name*
assumeRole	assumeRole	krn:ksc:sts::*account-id*:assumed-role/*role-name*/*role-session-name*
策略	policy	krn:ksc:iam::*account-id* | ksc:policy/*policy-name*
云物理主机镜像	image	krn:ksc:epc:*region*::image/*image-id*
云物理主机实例	epc-host	krn:ksc:epc:*region*:*account-id*:epc-host/*epc-host-id*
域名	domain	krn:ksc:cdn::*account-id*:domain/*domain-id*
CDN类型	cdn-type	krn:ksc:cdn::*account-id*:cdn-type/*cdn-type-name*
网络ACL规则	network-acl-entry	krn:ksc:vpc:*region*:*account-id*:network-acl-entry/*network-acl-entry-id*
安全组规则	security-group-entry	krn:ksc:vpc:*region*:*account-id*:security-group-entry/*security-group-entry-id*
存储空间	bucket	krn:ksc:ks3:::bucket
对象（文件）	object	krn:ksc:ks3:::bucket/objectKey(详细示例请参见：https://docs.ksyun.com/documents/5178)

User

描述

用户数据类型，包含IAM用户实体的相关信息。 在如下操作中作为响应元素被使用：

- CreateUser
- GetUser
- ListUsers

内容

名称	类型	必须	长度限制(字符)	参数格式	描述
Krn	String	是	最短是20,最长2048	格式：krn:ksc:iam::*account-id*:user/*user-name*	用户的唯一标识，使用KRN
CreateDate	DateTime	是	20	按照ISO8601标准，使用UTC时间，格式为"YYYY-MM-DDThh:mm:ssZ"例如，2014-05-26T12:00:00.000Z （为北京时间2014年5月26日20点0分0秒0毫秒）	用户的创建时间
Path	String	是	最短1,最长512	/	路径
UserId	String	是	22字符	[a-zA-Z0-9-_]+	用户唯一标识

UserName	String	是	最短1,最长64	[\w+=,.@-]+	新创建的IAM用户的用户名
RealName	String	是	最短2,最长128	[\u4e00-\u9fff]+	IAM用户的真实姓名

Role

描述

角色数据类型，包含角色的相关信息。 在如下操作中作为响应元素被使用：

- CreateRole
- GetRole
- ListRoles

内容

名称	类型	必须	长度限制(字符)	参数格式	描述
Krn	String	是	最短是25,最长2048	格式：krn:ksc:iam::*account-id*:role/*role-name*	角色的唯一标识
CreateDate	DateTime	是	20	按照ISO8601标准，使用UTC时间，格式为"YYYY-MM-DDThh:mm:ssZ"例如，2014-05-26T12:00:00Z（为北京时间2014年5月26日20点0分0秒）	创建时间
Path	String	是	最短1,最长512	\/ or \/[:graph:]+\/	路径
RoleId	String	是	22字符	[a-zA-Z0-9-_]+	Role唯一标识
RoleName	String	是	最短1,最长64	[\w+=,.@-]+	角色名
TrustedAccounts	String	是	最短1,最长2048	[\w\s:"*{}[]/$-]+	描述角色的信任关系的策略文档

LoginProfile

描述

登录配置数据类型，包含IAM用户实体的登录配置信息，包括用户名、密码创建时间和是否下次登录设置新密码。 在如下操作中作为响应元素被使用：

- CreateLoginProfile
- GetLoginProfile

注：用户的登录密码属于登录配置的组成元素，需要被安全可靠保存，但不会任何方式显示给用户。

内容

名称	类型	必须	长度限制(字符)	参数格式	描述
CreateDate	DateTime	是	20	按照ISO8601标准，使用UTC时间，格式为"YYYY-MM-DDThh:mm:ssZ"例如，2014-05-26T12:00:00.000Z （为北京时间2014年5月26日20点0分0秒0毫秒）	登录配置信息的创建时间
PasswordResetRequired	Boolean	是		true/false	用户下次成功登录后是否需要设置新密码,默认false，即不设定
UserName	String	是	最短1,最长64	[\w+=,.@-]+	登录配置所属的IAM用户的用户名

AccessKey

描述

访问密钥数据类型，包含访问密钥的相关信息。 在如下操作中作为响应元素被使用：

- CreateAccessKey

内容

名称	类型	必须	长度限制(字符)	参数格式	描述
CreateDate	DateTime	是	20	按照ISO8601标准，使用UTC时间，格式为"YYYY-MM-DDThh:mm:ssZ"例如，2014-05-26T12:00:00.000Z （为北京时间2014年5月26日20点0分0秒0毫秒）	访问密钥的创建时间
AccessKeyId	String	是	最短20,最长32	AKLT[a-zA-Z0-9-_]+	访问密钥ID
SecretAccessKey	String	是	68	[a-zA-Z0-9/+]+==	私有访问密钥
Status	String	是	6或8	Active/Inactive	访问密钥的状态，Active为可用，Inactive为不可用
UserName	String	否	最短1,最长64	[\w+=,.@-]+	如果是IAM用户的访问密钥，该域显示其用户名；如果是主账户的访问密钥，且已经设置别名，则该域显示别名信息，否则不显示

AccessKeyMetadata

描述

访问密钥元数据数据类型，不包含私有访问密钥的访问密钥相关信息。 在如下操作中作为响应元素被使用：

- ListAccessKeys

内容

名称	类型	必须	长度限制(字符)	参数格式	描述
CreateDate	DateTime	是	20	按照ISO8601标准，使用UTC时间，格式为"YYYY-MM-DDThh:mm:ssZ"例如，2014-05-26T12:00:00.000Z （为北京时间2014年5月26日20点0分0秒0毫秒）	访问密钥的创建时间
AccessKeyId	String	是	最短20,最长32	AKLT[a-zA-Z0-9-_]+	访问密钥ID
Status	String	是	6或8	Active/Inactive	访问密钥的状态，Active为可用，Inactive为不可用
UserName	String	否	最短1,最长64	[\w+=,.@-]+	如果是IAM用户的访问密钥，该域显示其用户名；如果是主账户的访问密钥，且已经设置别名，则该域显示别名信息，否则不显示

Policy

描述

策略数据类型，包含策略实体的元数据信息。 在如下操作中作为响应元素被使用：

- CreatePolicy
- GetPolicy
- ListPolicies

内容

名称	类型	必须	长度限制(字符)	参数格式	描述
Krn	String	是	最短是25,最长2048	格式：krn:ksc:iam::*account-id*:policy/*policy-name*	策略的唯一标识，使用KRN
CreateDate	DateTime	是	20	按照ISO8601标准，使用UTC时间，格式为"YYYY-MM-DDThh:mm:ssZ"例如，2014-05-26T12:00:00.000Z （为北京时间2014年5月26日20点0分0秒0毫秒）	策略的创建时间，即第一个策略版本的创建时间
UpdateDate	DateTime	是	20	按照ISO8601标准，使用UTC时间，格式为"YYYY-MM-DDThh:mm:ssZ"例如，2014-05-26T12:00:00.000Z （为北京时间2014年5月26日20点0分0秒0毫秒）	策略上次更新时间，如果只有一个策略版本，更新时间与创建时间一致，否则更新时间是最后一个策略版本的创建时间
Path	String	是	最短1,最长512	/	策略的路径
PolicyId	String	是	22字符	[a-zA-Z0-9-_]+	策略唯一标识
PolicyName	String	是	最短1,最长128	[\w+=,.@-]+	策略名称
DefaultVersionId	String	是	最短2	v[1-9][0-9]*	策略的默认版本Id
AttachmentCount	Integer	是		数值	附加该策略的实体数量，目前只有一种实体IAM用户
Description	String	否	0-1000		策略的友好描述信息，GetPolicy操作返回该元素，CreatePolicy和ListPolicies操作不返回该元素

PolicyVersion

描述

策略版本数据类型，包含策略版本的信息。 在如下操作中作为响应元素被使用：

- CreatePolicyVersion
- GetPolicyVersion
- ListPolicyVersions

内容

名称	类型	必须	长度限制(字符)	参数格式	描述
CreateDate	DateTime	是	20	按照ISO8601标准，使用UTC时间，格式为"YYYY-MM-DDThh:mm:ssZ"例如，2014-05-26T12:00:00.000Z （为北京时间2014年5月26日20点0分0秒0毫秒）	策略版本的创建时间
Document	String	否	最短1,最长5K	[\w\s:"*{}[]/$-]+	策略版本的策略文档内容，只有调用GetPolicyVersion接口时显示该元素
VersionId	String	是	最短2	v[1-9][0-9]*	策略版本Id
IsDefaultVersion	Boolean	是			用于指定当前策略版本是否为默认策略版本

AttachedPolicy

描述

附加策略数据类型，包含附加策略实体的元数据信息。 在如下操作中作为响应元素被使用：

- ListAttachedUserPolicies

内容

名称	类型	必须	长度限制(字符)	参数格式	描述
PolicyKrn	String	是	最短是25,最长2048	格式：krn:ksc:iam::*account-id*:policy/*policy-name*	策略的唯一标识，使用KRN
PolicyName	String	是	最短1,最长128	[\w+=,.@-]+	策略名称

PolicyUser

描述

策略关联用户数据类型，包含指定策略的关联用户实体的元数据信息。 在如下操作中作为响应元素被使用：

- ListEntitiesForPolicy

内容

名称	类型	必须	长度限制(字符)	参数格式	描述
UserName	String	是	最短1,最长64	[\w+=,.@-]+	策略关联用户名

SummaryMap

描述

概要映射表数据类型，包含IAM实体的使用和配额信息。 在如下操作中作为响应元素被使用：

- GetAccountSummary

内容

名称	类型	必须	长度限制(字符)	参数格式	描述
AccessKeysPerUserQuota	Integer	是	无	数字	每个IAM用户的访问密钥配额，2
AccountAccessKeysPresent	Integer	是	无	0或1	账户当前是否有访问密钥，有是1，否则是0
AttachedPoliciesPerUserQuota	Integer	是	无	数字	每个IAM用户可附加的策略数量配额，5
Policies	Integer	是	无	数字	账户当前自定义策略的数量
PoliciesQuota	Integer	是	无	数字	账户的自定义策略数量配额，50
PolicySizeQuota	Integer	是	无	数字	自定义策略的字符数配额，不包括空白字符，2048
PolicyVersionsInUse	Integer	是	无	数字	账户当前附加到IAM实体的策略数量
PolicyVersionsInUseQuota	Integer	是	无	数字	账户中可附加到IAM实体的策略数量配额，500
Users	Integer	是	无	数字	账户当前IAM用户数量
UsersQuota	Integer	是	无	数字	账户的IAM用户数量配额，100
VersionsPerPolicyQuota	Integer	是	无	数字	自定义策略的版本数量配额，5

PasswordPolicy

描述

密码策略数据类型，包含IAM用户密码策略的详细信息。 在如下操作中作为响应元素被使用：

- GetAccountPasswordPolicy

内容

名称	类型	必须	长度限制(字符)	参数格式	描述

AllowUsersToChangePassword	Boolean	是	无	无	是否允许全部IAM用户在控制台修改自己的密码
ExpirePasswords	Boolean	是	无	无	是否允许密码过期，如果MaxPasswordAge大于0则取值true，否则false
HardExpiry	Boolean	是	无	无	是否阻止IAM用户在密码过期后修改自己的密码
MaxPasswordAge	Integer	否	最小值1，最大值1095	无	IAM用户密码**有效天数**，默认0则不显示
MinimumPasswordLength	Integer	是	最小值8，最大值128	无	IAM用户密码**最小长度**
PasswordReusePrevention	Integer	否	最小值1，最大值24	无	阻止IAM用户使用的历史密码数量，默认0则不显示
RequireLowercaseCharacters	Boolean	是	无	无	是否强制要求IAM用户密码包含一个**小写字母**
RequireNumbers	Boolean	否	无	无	是否强制要求IAM用户密码包含一个**数字**
RequireSymbols	Boolean	否	无	无	是否强制要求IAM用户密码包含一个后面的**符号**（！@ # $ % ^ & * ( ) _ + - = [ ] { } ’）
RequireUppercaseCharacters	Boolean	否	无	无	是否强制要求IAM用户密码包含一个**大写字母**

VirtualMFADevice

描述

虚拟多因素认证数据类型，包含虚拟多因素认证设备的相关信息。

在如下操作中作为响应元素被使用：

- CreateVirtualMFADevice

内容 {

名称	类型	必须	长度限制(字符)	参数格式	描述
SerialNumber	String	是	20	[\w+=/:,.@-]+	设备序列号
Base32StringSeed	String	否	最短20,最长32	Base64-encoded binary data	多因素认证设备密钥
QRCodePNG	String	否	68	Base64-encoded binary data	密钥二维码PNG
EnableDate	DateTime	否	20	按照ISO8601标准，使用UTC时间，格式为"YYYY-MM-DDThh:mm:ssZ"例如，2014-05-26T12:00:00Z（为北京时间2014年5月26日20点0分0秒）	虚拟多因素认证设备激活日期
User	User数据类型	否	最短1,最长64	[\w+=,.@-]+	绑定用户的基本信息

AssumeRoleResult

描述

访问密钥数据类型，包含临时访问密钥的相关信息。 在如下操作中作为响应元素被使用：

- AssumeRole

AssumedRoleUser内容

名称	类型	必须	长度限制(字符)	参数格式	描述
Arn	String	是	min:20,max:2048	krn:ksc:sts::*account-id*:assumed-role/*role-name/role-session-name*	
AssumedRoleId	String	是	min:2,max:96	[\w+=,.@:-]*+	包含roleId 和 role_session_name,用冒号: 做分隔

Credentials内容

名称	类型	必须	长度限制(字符)	参数格式	描述
AccessKeyId	String	是	min:20,max:32	AKRT[a-zA-Z0-9-_]+	
Expiration	Timestamp	是			当前临时证书的过期时间
SecretAccessKey	String	是		[a-zA-Z0-9/+]+==	
SecurityToken	String	是			会话session

其他内容

名称	类型	必须	长度限制(字符)	参数格式	描述
PackedPolicySize	String	是	min:0		返回0

- JSON示例


```
{
"AssumeRoleResult": {
    "Credentials": {
        "SecretAccessKey": "xxxx==",
        "Expiration": "2018-06-19T10:32:50Z",
        "AccessKeyId": "AKTRxxxx",
        "SecurityToken": "V1KYiiBxLGb0U2tI3hBLVNL7jqlnfj+6NDubuGqhTjSKzUxMPQoQF9LWBCN/JoTVaJL4AoL2xB7v7MR0ibed+7+KQ95IrocgRpF0wotCW+UiBbcgCGlUZlvgJgY4aXj/u9SNCu0TITgZFFFmCJBND2iagzkiCiKzIvFHuPesYrcBMN3Na2IdtTIEtMO4GPr4d3PzMYE6cuSs900EHCsiQ05e8kfyYNGGHUj7RykEzA711HuNV7m8U00JncNGAmbeXMaXp0enyTWkzJJYN+FLT5BoXFJ3RWCYIl9byd8Ub87ZT5TXSggIJ+C+GfhbuatH7Gk5a68bCnoMETUZMQDDA1lg=="
    },
    "AssumedRoleUser": {
        "Krn": "krn:ksc:sts::xxx:assumed-role/xxx/thisisatest",
        "AssumedRoleId": "xxx:thisisatest"
    }
},
"RequestId": "93e14aef-442f-455f-bbcc-fecec40fe364"
}
```


UserSession

描述

用户数据类型，包含IAM用户实体的相关信息。 在如下操作中作为响应元素被使用：

- GetUserSession

内容

名称	类型	必须	长度限制(字符)	参数格式	描述
Url	String	是	最短是20,最长2048	-	获取Session校验的地址
Expiration	String	是	20个字符	Y-m-d\TH:i:s\Z	连接的失效时间