目录

介绍

分布式事务服务KDTX OpenAPI文档

本文档详细介绍了分布式事务服务KDTX 的OpenAPI的调用方式和各种相关接口。

服务信息

服务信息

- **服务地址**
  http://kdtx.api.ksyun.com
- **通讯协议**
  支持HTTP或HTTPS协议
- **请求方法**
  详见具体API接口
- **请求参数**
  每个请求都需要包含公共的请求参数和操作所特有的请求参数，详见具体API接口
- **编码方式**
  请求及结果返回都使用UTF-8字符集编码
- **返回结果**
  仅支持json返回结

Region及可用区

各个region都通过同一个endpoint来访问API服务：krds.api.ksyun.com

Region名称	类型	Region代码	可用区
华东1（上海）	VPC机房	cn-shanghai-2	cn-hanghai-2b

核心概念

计费类型

金山云分布式事务支持按日用量实时付费计费方式。在API接口中可以通过设置BillType参数来指定计费类型。

按日用量实时付费

采取按量付费的计费方式。实际事务总数等于全局事务数和分支事务数的总和。按量付费是先使用，后付费。每天按KDTX实际事务数计费。

参数名称	中文名称	类型	备注
BillType	计费方式	String	默认值：86，取值范围：86（按日计费）。

公共参数

公共参数

名称	描述	是否必须
Authorization	签名结果	是
Host	请求服务域名（用于签名计算）	是
X-Amz-Date	ISO 8601格式时间戳（用于签名计算）	是
X-KSC-REQUEST-ID	请求流水号，用于报障使用，若不提供则默认自动生成一个	否

签名机制

签名机制

发送给分布式事务服务KDTX的HTTP请求中，必须包含授权参数和其他公共参数。分布式事务服务使用用户的Access Key ID和Secret Access Key进行加密方式来验证请求者身份。Access Key ID和Secret Access Key由金山云发给用户，Access Key ID作为用户的身份标识，Secret Access Key作为用户和服务器短进行签名计算的秘钥。

签名方法

Http请求header中Authorization字段是服务的授权参数，其格式为：

```
Authorization="[HashMethod][空格]Credential=[access_key]/[scope],SignedHeaders=[signed_headers],Signature=[signature]"
```

其中：

```
[HashMethod] ="KSC4-HMAC-SHA256"
[access_key] =用户Access key ID
[scope] = [timestamp]/[region]/[service]/[req_type]
```

timestamp为yyyyMMdd格式的时间戳，region为请求服务所在区域名，service为访问的服务名，req_type为请求的类型。

```
[signed_headers]：将Headers按照name升序排列
[signed_headers] = [header_name_1];[header_name_2]....
```

签名算法： [signature]= sha256(sha256(sha256(sha256(sha256("KSC4"+sign_key,timestamp),region),service),req_type),string_to_sign)

其中：

```
 [sign_key] = 用户Secret Access Key
 [stringToSign] = "KSC4-HMAC-SHA256" + "\n" + [X-Ksc-Date] + "\n" +[scope] + "\n" + SHA-256([canonical_request])
 [canonical_request] = [HTTPRequestMethod] + "\n" + [CanonicalURI] + "\n" + [CanonicalQueryString] + "\n" + [CanonicalHeaders] +
 "\n" + [signed_headers] + "\n" + SHA-256([request_body])
 [HTTPRequestMethod] = POST或GET
 [CanonicalURI] = 请求URL中除去Endpoint之外的剩余部分。目前URL等于Endpoint，所以CanonicalURI为空
 [CanonicalQueryString] = 空
 [CanonicalHeaders]：按照[signed_headers]中的排序方式进行排序
 [CanonicalHeaders] =
 LowerCase (HeaderName1) + ':' + Trim (HeaderValue1) + "\n"+LowerCase (HeaderName2) + ':' + Trim (HeaderValue2) + "\n"+.....
..
 [request_body] = Post 请求的body部分
```

Python Client Demo

```
import hashlib
import hmac
import urllib

import datetime
import requests

class RequestComposer:
    def __init__(self, ak, sk, service, request_parameters, method, content_hash=False):
        self.ak = ak
        self.sk = sk
        self.service = service
        self.content_hash = content_hash
        self.header_name_authorization = 'Authorization'
        self.header_name_host = 'Host'
        self.header_name_content_sha256 = 'X-Amz-Content-Sha256'
        self.header_name_date = 'X-Amz-Date'
        self.hash_keyword = 'AWS4'
        self.hash_method = 'AWS4-HMAC-SHA256'
        self.request_parameters = request_parameters
        self.method = method

    @staticmethod
    def get_canonical_headers(headers):
        canonical = []

        for header in headers:
            c_name = header.lower().strip()
            raw_value = str(headers[header])
            if '"' in raw_value:
                c_value = raw_value.strip()
            else:
                c_value = ' '.join(raw_value.strip().split())
            canonical.append('%s:%s' % (c_name, c_value))
        return '\n'.join(sorted(canonical)) + '\n'

    @staticmethod
    def get_signed_headers(headers):
        l = ['%s' % n.lower().strip() for n in headers]
        l = sorted(l)
```

```
        return ';'.join(l)

    def get_headers(self, host, region, payload, additional_signing_headers=None):
        if additional_signing_headers is None:
            additional_signing_headers = {}
        if self.ak is None or self.sk is None:
            return None

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

        # ************* TASK 1: CREATE A CANONICAL REQUEST *************
        # http://docs.aws.amazon.com/general/latest/gr/sigv4-create-canonical-request.html

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

        # 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, request
        # parameters are passed in the body of the request and the query string
        # is blank.
        canonical_querystring = self.request_parameters

        # Step 4: Create the canonical headers. Header names and values
        # must be trimmed and lowercase, and sorted in ASCII order.
        # Note that there is a trailing \n.
        headers_to_sign = {
            self.header_name_host: host,  # sign indispensable
            self.header_name_date: amz_date,  # sign indispensable
        }

        if self.content_hash:
            headers_to_sign[self.header_name_content_sha256] = hashlib.sha256(payload).hexdigest()
        # additional headers can be added to signing process
        for h in additional_signing_headers:
            if h not in headers_to_sign:
                headers_to_sign[h] = additional_signing_headers[h]

        canonical_headers = self.get_canonical_headers(headers_to_sign)

        # 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 include those that you want to be included in the
        # hash of the request. "Host" and "x-amz-date" are always required.
        # For DynamoDB, content-type and x-amz-target are also required.
        signed_headers = self.get_signed_headers(headers_to_sign)

        # Step 6: Create payload hash. In this example, the payload (body of
        # the request) contains the request parameters.
        payload_hash = hashlib.sha256(payload).hexdigest()

        # Step 7: Combine elements to create 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 = self.hash_method
        credential_scope = date_stamp + '/' + region + '/' + self.service + '/' + 'aws4_request'
        string_to_sign = algorithm + '\n' + amz_date + '\n' + credential_scope + '\n' + hashlib.sha256(
            canonical_request).hexdigest()

        # ************* TASK 3: CALCULATE THE SIGNATURE *************
        # Create the signing key using the function defined above.
        signing_key = self.getSignatureKey(self.sk, date_stamp, region, self.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 *************
        # Put the signature information in a header named Authorization.
        authorization_header = algorithm + ' ' + 'Credential=' + self.ak + '/' + credential_scope + ', ' + \
            'SignedHeaders=' + signed_headers + ', ' + 'Signature=' + signature
        # For DynamoDB, the request can include any headers, but MUST include "host", "x-amz-date",
        # "x-amz-target", "content-type", and "Authorization". Except for the authorization
        # header, the headers must be included in the canonical_headers and signed_headers values, as
```

```
        # noted earlier. Order here is not significant.
        # # Python note: The 'host' header is added automatically by the Python 'requests' library.
        headers = {}
        for h in headers_to_sign:
            headers[h] = headers_to_sign[h]
            headers[self.header_name_authorization] = authorization_header

        return headers

    # Key derivation functions. See:
    @staticmethod
    def sign(key, msg):
        return hmac.new(key, msg.encode("utf-8"), hashlib.sha256).digest()

    def getSignatureKey(self, key, date_stamp, region_name, service_name):
        kDate = self.sign((self.hash_keyword + key).encode('utf-8'), date_stamp)
        kRegion = self.sign(kDate, region_name)
        kService = self.sign(kRegion, service_name)
        kSigning = self.sign(kService, 'aws4_request')
        return kSigning

class Client(object):
    def __init__(self, host, ak=None, sk=None, service=None, version=None, region=None):
        self.service = service
        self.host = host
        self.ak = ak
        self.sk = sk
        self.region = region
        self.version = version

    def call(self, target, dict_request_parameters=None, headers=None):
        # usiing kop authentication
        if headers is None:
            headers = {}
        url = 'http://%s?' % self.host
        if isinstance(dict_request_parameters, str):
            dict_request_parameters = eval(dict_request_parameters)
        dict_request_parameters['Version'] = self.version

        request_parameters = "Action=%s&" % target + '&'.join(
            ["%s=%s" % (k, urllib.quote(str(dict_request_parameters[k]), ''))
             for k in sorted(dict_request_parameters.keys())])
        url += request_parameters

        composer = RequestComposer(ak=self.ak, sk=self.sk, service=self.service,
                                   request_parameters=request_parameters, method='GET')
        headers = composer.get_headers(host=self.host, region=self.region,
                                       payload="", additional_signing_headers=headers)
        r = requests.get(url, headers=headers)
        return r

AK = 'YOUR ACCESS KEY'
SK = 'YOUR SECRET KEY'
VERSION = "2016-07-01"
SERVICE = "krds"
REGION = "cn-beijing-6"
HOST = "krds.%s.api.ksyun.com" % REGION

client = Client(HOST, AK, SK, SERVICE, VERSION, REGION)
additional_headers = {'Accept': 'application/json'}
result = client.call("DescribeDBEngineVersions", {'Engine': 'MySQL', 'EngineVersion': '5.5'}, additional_headers)
print result.json()
```

错误码

错误码

OK

成功

- HttpCode：200

INVALID_ACTION

操作无效，请检查操作是否符合规范

- HttpCode：400

INVALID_PARAMETER_COMBINATION

参数绑定错误

- HttpCode：400

INVALID_PARAMETER_VALUE

参数值必要或者参数值格式不正确

- HttpCode：400

MISSING_PARAMETER

参数必填

- HttpCode：400

NOT_FOUND

资源不存在，或者已被删除

- HttpCode：404

MALFORMED_QUERY_STRING

请求包含语法错误

- HttpCode：404

INTERNAL_FAILURE

服务器内部错误，联系技术支持解决

- HttpCode：500

SERVICE_UNAVAILABLE

服务暂不可用，请稍后重试

- HttpCode：503

BACKING_UP

实例备份中，无法进行其他操作，请稍后重试

- HttpCode：403

INSTANCE_TASK

实例正在任务中，任务中包括重启中、主从切换中等

- HttpCode：409

OPERATION_DENIED_CREATE_BACKUP

一个实例至多支持5个手动备份

- HttpCode：400

INVALID_PACKAGES

不支持购买非标套餐配置，请联系客服人员

- HttpCode：400

查看监控数据

接口	method	url
GET	/kdtx	

输入项中文名称	字段名	数据类型	输入属性	取值范围及备注
Action	Action	String	必输	InspectDistributeTransactionGroupsMonitor
查询开始时间	StartTime	long		
查询结束时间	EndTime	long		
GroupId	long			

header中文名	字段名	输入属性	取值范围
区域	X-KSC-REGION		
账户名称	X-KSC-ACCOUNT-ID		

输出项中文名称	字段名	数据类型	输出属性	取值范围及备注
msg	msg	String		
status	status	int		
data	data	对象	timestamp：时间戳；globalTransCount：时间戳对应的全局事务数；branchTransCount：时间戳对应的分支事务数；	[{ "timestamp": 1561608838000, "globalTxCount": 10, "branchTxCount": 30 }, { "timestamp": 1561608838000, "globalTxCount": 10, "branchTxCount": 30 }]

查看事务分组列表

接口	method	url
GET	/kdtx	

输入项中文名称	字段名	数据类型	输入属性	取值范围及备注
Action	Action	String	必输	InspectDistributeTransactionGroups
请求页码	Page			默认为第一页
每页展示数量	Size			默认每页10条记录

header中文名	字段名	输入属性	取值范围
账户名称	X-KSC-ACCOUNT-ID		
区域	X-KSC-REGION		

输出项中文名称	字段名	数据类型	输出属性	取值范围及备注
msg	msg	String		
status	status	int		
page	page	int		
size	size	int		
totalPage	totalPage	int		
totalCount	totalCount	int		
data	data	对象	groupId：记录ID，唯一标识一条记录；region：region名称；groupName：事务分组名称；groupCount：事务分组数；globalTransCount：全局事务总数；branchTransCount：分支事务总数；createTime：事务分组创建时间；	[{ "groupId": 1, "region": "cn-beijing-1", "groupName": "order-kdtx-service-group", "groupCount": 1, "globalTxCount": 20, "branchTxCount": 30, "createTime": 1561608838000, "token": "dfaslwerwekj334234jljadf" }, { "groupId": 2, "region": "cn-beijing-1", "groupName": "order-kdtx-service-group", "groupCount": 1, "globalTxCount": 20, "branchTxCount": 30, "createTime": 1561608838000, "token": "dfaslwerwekj334234jljadf" }]

查询分支事务

接口	method	url
GET	/kdtx	

输入项中文名称	字段名	数据类型	输入属性	取值范围及备注
Action	Action	String	必输	InspectBranchDistributeTransactions
请求页码	Page			默认为第一页

每页展示数量	Size			默认每页10条记录
事务状态	State	string	选填	running/commit/rollback/error
全局事务ID	Xid	string	选填	
分支事务ID	BranchId	string	选填	
事务分组ID	GroupId	long		

header中文名	字段名	输入属性	取值范围
账户名称	X-KSC-ACCOUNT-ID		
区域	X-KSC-REGION		

输出项中文名称	字段名	数据类型	输出属性	取值范围及备注
msg	msg	String		
status	status	int		
page	page	int		
size	size	int		
totalPage	totalPage	int		
totalCount	totalCount	int		
data	data	对象	id：记录ID，唯一标识一条记录；xid：全局事务ID；branchId：分支事务ID； state：分支事务状态：commit/rollback/running/error；stateText：分支服务状态展示文本：已提交、已回滚、运行中、异常；lockKey：全局锁标识；resourceId：资源ID；clientId：客户端ID；branchType：分支事务类型：0自动事务，1手动事务；branchTypeText：分支事务类型展示文本；msg：如果为异常状态，则记录异常信息；region：地域标识；	[{ "id": 1, "xid": "23429-adfas-2134-eqwrq", "branchId": "23429-adfas-2134-eqwrq", "state": "error", "stateText": "异常", "lockKey": "poc_account:100", "resourceId": "jdbc:mysql://product:112.23.12.23:8992", "clientId": "product:112.23.12.23:8992", "branchType": 0, "branchTypeText": "AT", "msg": "rollback timeout", "region": "cn-beijing-1" }, { "id": 2, "xid": "23429-adfas-2134-eqwrq", "branchId": "23429-adfas-2134-eqwrq", "state": "commit", "stateText": "已提交", "lockKey": "poc_account:100", "resourceId": "jdbc:mysql://product:112.23.12.23:8992", "clientId": "product:112.23.12.23:8992", "branchType": 1, "branchTypeText": "MT", "msg": "success", "region": "cn-beijing-1" }]

查询全局事务

接口	method	url
GET	/kdtx	

输入项中文名称	字段名	数据类型	输入属性	取值范围及备注
Action	Action	String	必输	InspectGlobalDistributeTransactions
请求页码	Page			默认为第一页
每页展示数量	Size			默认每页10条记录
事务状态	State	string	选填	running/commit/rollback/error
全局事务ID	Xid	string	选填	
全局事务名称	Name	string	选填	
应用ID	AppId	string		
groupId	GroupId	long		

header中文名	字段名	输入属性	取值范围
账户名称	X-KSC-ACCOUNT-ID		
区域	X-KSC-REGION		

输出项中文名称	字段名	数据类型	输出属性	取值范围及备注
msg	msg	String		
status	status	int		
page	page	int		
size	size	int		
totalPage	totalPage	int		
totalCount	totalCount	int		

data	data	对象	xid：全局事务ID； name：全局事务名称；state：全局事务状态；stateText：分支服务状态展示文本：已提交、已回滚、运行中、异常；commit/rollback/running/error；appId：应用ID；startTime：事务开始时间；timeout：事务超时时间；msg：如果为异常状态，则记录异常信息；	[{ "id": 1, "xid": "23429-adfas-2134-eqwrq", "name": "payTheBill(com.ksyun.demo.order.controller.vo.PayReq)", "state": "error", "stateText": "异常", "appId": "order", "startTime": 1561608838000, "timeout": 6000, "msg": "rollback timeout", "region": "cn-beijing-1" }, { "id": 2, "xid": "23429-adfas-2134-eqwrq", "name": "payTheBill(com.ksyun.demo.order.controller.vo.PayReq)", "state": "committed", "stateText": "已提交", "appId": "order", "startTime": 1561608838000, "timeout": 6000, "msg": "success", "region": "cn-beijing-1" }]

查询指定用户全局事务概览信息（概览页）

接口	method	url
GET	/kdtx	

输入项中文名称	字段名	数据类型	输入属性	取值范围及备注
动作名称	Action	String	必输	Resume

header中文名	字段名	输入属性	取值范围
账户名称	X-KSC-ACCOUNT-ID		

输出项中文名称	字段名	数据类型	输出属性	取值范围及备注
status	status	int		
msg	msg	String		
data	data	对象	groupCount：事务分组总数；globalTransCount：全局事务总数；branchTransCount：分支事务总数；regions：分region统计信息；region：region名称；groupCount：事务分组数；globalTrans：全局事务统计；branchTrans：分支事务统计；total：事务总数；committed：已提交事务数；rollbacked：已回滚事务数；running：运行中事务数；error：异常事务数；	{ "groupCount": 2, "globalTxCount": 10, "branchTxCount": 30, "regions": [{ "region": "华北1（北京）", "groupCount": 1, "globalTx": { "total": 5, "committed": 3, "rollbacked": 1, "running": 1, "error": 0 }, "branchTx": { "total": 5, "committed": 3, "rollbacked": 1, "running": 1, "error": 0 } }] }

新建事务分组

接口	method	url
GET	/kdtx	

输入项中文名称	字段名	数据类型	输入属性	取值范围及备注
Action	Action	String	必输	CreateDistributeTransactionGroup
分组名称	GroupName			
计费类型	BillType			
ProductWhat				1.正式 2.试用

header中文名	字段名	输入属性	取值范围
账户名称	X-KSC-ACCOUNT-ID		
区域	X-KSC-REGION		

输出项中文名称	字段名	数据类型	输出属性	取值范围及备注
msg	msg	String		
status	status	int		
data	data	对象	groupId：分组ID； groupName：分组名称；region：名称；chargeType：计费类型，预留字段，给1即可；	{ "groupCount": 2, "globalTxCount": 10, "branchTxCount": 30, "regions": [{ "region": "华北1（北京）", "groupCount": 1, "globalTx": { "total": 5, "committed": 3, "rollbacked": 1, "running": 1, "error": 0 }, "branchTx": { "total": 5, "committed": 3, "rollbacked": 1, "running": 1, "error": 0 } }, { "region": "北京5区", "groupCount": 1, "globalTx": { "total": 5, "committed": 3, "rollbacked": 1, "running": 1, "error": 0 }, "branchTx": { "total": 5, "committed": 3, "rollbacked": 1, "running": 1, "error": 0 } }] }