目录

API概览

KS3请求相关

API	接口功能
请求签名V2	向KS3发起携带签名的请求
请求签名V4	向KS3发起携带V4签名的请求
公共请求头	KS3常用的请求头部
公共响应头	KS3常用的响应头部
错误码	可能出现的错误码列表

Service相关

API	接口功能
Get Service	返回请求者拥有的所有的存储空间

Bucket相关

API	接口功能
PUT Bucket	创建存储空间
GET Bucket (ListObjects)	列出指定空间中的部分或全部(上限1000)的对象
GET Location	返回用户存储空间的区域
HEAD Bucket	判断某一存储空间是否存在，以及用户对其的访问权限
DELETE Bucket	删除给定的存储空间
PUT BucketACL	设置存储空间的ACL
GET BucketACL	返回存储空间的访问权限控制列表(Access control list，简称ACL)
Put Bucket Lifecycle	为指定存储空间添加生命周期规则
Get Bucket Lifecycle	获取指定存储空间的生命周期规则配置
Delete Bucket Lifecycle	删除指定存储空间的生命周期规则
Put Bucket Policy	为指定存储空间添加空间策略(Bucket Policy)
Get Bucket Policy	获取指定存储空间的空间策略(Bucket Policy)
Delete Bucket Policy	删除指定存储空间的空间策略(Bucket Policy)
Put Bucket Replication	为源存储空间设置跨区域复制规则
Get Bucket Replication	获取源存储空间的跨区域复制规则
Delete Bucket Replication	关闭源存储空间的跨区域复制规则
GET Bucket logging	返回用户存储空间的日志记录状态
PUT Bucket logging	配置用户存储空间的日志参数、指定允许查看/修改日志参数的用户
PUT Bucket CORS	配置存储空间的CORS信息
GET Bucket CORS	获取存储空间的CORS信息
DELETE Bucket CORS	删除指定存储空间的CORS配置
OPTIONS Object	获取存储空间的CORS信息
PUT Bucket Mirror	为指定存储空间配置回源规则
GET Bucket Mirror	获取指定存储空间的回源规则
DELETE Bucket Mirror	删除指定存储空间的回源规则

Object相关

API	接口功能
PUT Object	添加一个对象到某个存储空间
PUT Object Copy	将KS3中某个对象拷贝到指定存储空间中
PUT Fetch	从第三方URL拉取文件，并且上传到KS3指定存储空间中
POST Object	使用HTML POST表单添加一个对象到某个存储空间
POST Policy	基于POST Policy方式安全上传文件
GET Object	获取对象
HEAD Object	返回对象的元数据信息
DELETE Object	删除对象
PUT Object ACL	设置对象的ACL
GET Object ACL	返回对象的ACL
Put Object Tagging	用于设置或更新对象的标签信息。
Get Object Tagging	用于获取对象的标签信息
Delete Object Tagging	删除指定对象的所有标签信息
Restore Object	解冻归档存储类型的对象

分块上传相关

API	接口功能
Initiate Multipart Upload	启动分块上传任务
Upload Part	在分块上传任务中，上传一个块
Complete Multipart Upload	完成分块上传任务，即将所有的块组装成一个对象
Abort Multipart Upload	放弃一个分块上传任务
List Parts	列出指定分块上传任务中已上传的块
List Multipart Uploads	返回存储空间下所有正在进行分块上传的任务
Upload Part Copy	通过拷贝已存在的对象的方式实现上传一个块

异步数据处理相关

API	接口功能
上传回调处理	在调用PUT Object、Complete Multipart Upload API时，携带相关的Callback参数，实现上传回调处理（Upload CallBack Processing，简称UCP）

AWS S3协议兼容介绍

API	接口功能
AWS S3协议兼容	KS3 API与AWS S3协议的兼容性对比

加密相关

API	接口功能
加密	KS3提供的两种服务器端加密的方式的使用方式

KS3计量相关

API	接口功能
KS3计量	用户可以通过该API查询KS3的使用情况

请求签名V2

KS3提供V4和V2两种签名方式，推荐使用V4签名方式，参见请求签名V4。

使用KS3 时，可通过 RESTful API 对 KS3 发起 HTTP 匿名请求或 HTTP 签名请求，对于签名请求，KS3服务器端将会对请求发起者进行身份验证。

- 匿名请求：HTTP 请求不携带任何身份标识和鉴权信息，通过 RESTful API 进行 HTTP 请求操作。当KS3接收到匿名请求时，如果发现用户请求的资源不允许匿名请求，将会返回403错误。
- 签名请求：HTTP 请求时携带签名，KS3服务器端收到消息后，进行身份验证，验证成功则接受并执行请求，否则将会返回403错误信息并丢弃此请求。 KS3提供了可视化签名工具用于生成V2签名，方便客户调试签名错误，快速定位问题。

通过HTTP请求头发送签名

用户可以在HTTP请求中增加Authorization请求头来包含签名信息。

签名计算方法

```
"Authorization: KSS" + YourAccessKey + ":" + Signature

Signature =Base64( HMAC-SHA1( YourSecretAccessKey, UTF-8-Encoding-Of( StringToSign ) ) ) ;

StringToSign = HTTP-VERB + "\n" +
Content-MD5 + "\n" +
Content-Type + "\n" +
Date + "\n"  +
CanonicalizedKssHeaders + "\n"
CanonicalizedResource;

CanonicalizedResource = [ "/" + Bucket ] +
<HTTP-Request-URI > +
[<sub-resource>];
```

说明：

- HTTP-Verb 表示请求的方法，如：GET\PUT\POST\DELETE等
- Content-MD5 表示请求内容数据的MD5值，使用Base64编码。当请求头中包含Content-MD5时，需要在StringToSign中包含Content-MD5，否则用("")替代。

  注意： Content-MD5的算法为先对数据做MD5摘要，再将MD5摘要做Base64编码，中间不需要做Hex编码。由于部分语言或工具包的MD5是默认做Hex编码的，所以当MD5算出来的结果为Hex编码时，首先需要对算出来的结果做Hex解码，然后再做Base64编码。详解RFC2616
- Content-Type 表示请求体的类型
- Date 表示此次请求操作的时间，必须为 HTTP1.1 中支持的 GMT 格式，例如：Tue, 30 Nov 2021 06:29:38 GMT。若Date时间与KS3服务端时间相差15分钟以上，则KS3将返回403错误。

  注意： 有的客户端不支持发送Date请求头。这种情况下，计算签名时需要保持Date字段的同时，在CanonicalizedKssHeaders中加入x-kss-date，格式与Date一致。当请求中包含x-kss-date头时，KS3在计算签名时会忽略Date头。
- CanonicalizedKssHeaders 表示HTTP请求中的以x-kss开头的Header组合，详见 CanonicalizedKssHeaders计算方法。
- CanonicalizedResource 表示用户访问的资源，详见CanonicalizedResource的计算方法。
- StringToSign中不包含Content-Type, Date, Content-MD5这些请求头的名字，只包含这些请求头的值。但是以“x-kss”开头的请求头的名字和值都包含在StringToSign中。
- 如果在请求中，Content-Type, Content-MD5等请求头不存在，那么该位置用空串（""）来代替。

CanonicalizedKssHeaders计算方法

所有以“x-kss-”为前缀的HTTP请求头被称为CanonicalizedKssHeaders。它的构造方法如下：

1) 将所有以“x-kss-”为前缀的HTTP请求头名字转换成小写字母。如’ X-KSS-Meta-Name: Jack’转换成’ x-kss-meta-name: Jack。

2) 将上一步得到的所有HTTP请求头按照字典序进行升序排列。

3) 如果有相同名字的请求头，则根据标准RFC 2616, 4.2章进行合并（两个值之间只用逗号分隔）。例如有两个名为’ x-kss-meta-name’ 的请求头，对应的值分别为’ fred’ 和’ barney’，则合并后为：’ x-kss-meta-name:fred,barney’ 。

4) 删除请求头和内容之间分隔符两端出现的任何空格。如’ x-kss-meta-name: fred,barney’ 转换成：’ x-kss-meta-name:fred,barney’ 。

5) 将所有请求头名称和值用’ \n’ 分隔符分隔，拼成最后的CanonicalizedKssHeaders。

注意：

- 若CanonicalizedKssHeaders为空，无需添加最后的\n。
- 如果只有一个请求头，则需要在最后添加\n，例如：x-kss-meta-yourname:Lee\n
- 如果有多个请求头，则使用使用"\n"分隔符连接在一起，并且在最后添加\n，例如：x-kss-meta-myname:Jack\nx-kss-meta-yourname:Lee\n
- 如果客户端不支持发送 Date 请求头，则在计算CanonicalizedKssHeaders时必须增加 x-kss-date 请求头。

构建CanonicalizedResource的方法

用户发送的请求中，要访问的KS3目标资源被称为CanonicalizedResource。结构如下：

```
/[BucketName/[ObjectKey[?SubResource]]]
```

- BucketName: 用户请求的Bucket名称。
- ObjectKey: 用户请求的Object名称,需要对Object名称做URL编码。
- SubResource: 用户请求的子资源。把URL参数中的"acl","lifecycle","location","logging","notification","partNumber","policy","requestPayment","torrent","uploadId","uploads","versionId","versioning","versions","website","content-type","response-content-language","response-expires","response-cache-control","response-content-disposition","response-content-encoding"筛选出来，将这些查询字符串及其请求值(不做URL编码的请求值)按照字典序，从小到大排列，以&为分隔符排列，即可得到SubResource。

CanonicalizedResource构造方法如下：

1. CanonicalizedResource="/"
2. 如果BucketName不为空,则 CanonicalizedResource = CanonicalizedResource + BucketName + “/”
3. 如果ObjectKey不为空,则 CanonicalizedResource = CanonicalizedResource + ObjectKey
4. 替换CanonicalizedResource中的双斜杠("//")为"/%2F"
5. 如果SubResource不为空，则CanonicalizedResource = CanonicalizedResource + “?” + SubResource

示例

以下是使用V2签名的示例。示例中使用的访问密钥如下：

参数	值
KSSAccessKeyId	AKLTA6qLnuowT6KzKybUQNCOTw

KSSSecretAccessKey OCd5HzFDU1YDUG6eTHASvdt1RRn5bqKNKd18JxuFrYne+bazX7gmoYUG73XjJ/d2sg==

GET Object

从examplebucket中get 对象

请求	StringToSign
GET /1.txt HTTP/1.1 Host: examplebucket.ks3-cn-beijing.ksyuncs.com Date: Tue, 30 Nov 2021 11:06:30 GMT Authorization: KSS AKLTA6qLnuowT6KzKybUQNCOTw:i+Pi Oc1sxIe6yjZwyi4/+kxmXs8=	GET\n \n \n Tue, 30 Nov 2021 11:06:30 GMT\n /examplebucket/1.txt

注意：CanonicalizedResource中包含bucket名称，但是HTTP请求URI中没有，因为bucket名称是在Host请求头中指定的。

示例代码如下：

```
import base64
import hmac
from hashlib import sha1
h = hmac.new("OCd5HzFDU1YDUG6eTHASvdt1RRn5bqKNKd18JxuFrYne+bazX7gmoYUG73XjJ/d2sg==", "GET\n\n\nTue, 30 Nov 2021 11:06:30 GMT\n/examplebucket/photos/1.jpg", sha1)
Signature = base64.encodestring(h.digest()).strip()
```

PUT Object

向examplebucket中上传一个对象

请求	StringToSign
PUT /1.txt HTTP/1.1 Content-Type: text/plain Content-Length: 10 Host: examplebucket.ks3-cn-beijing.ksyuncs.com Date: Wed, 1 Dec 2021 01:46:43 GMT Authorization: KSS AKLTA6qLnuowT6KzKybUQNCOTw:k53X 6xtOlzOz91QDYY/IA3NGVrY=	PUT\n \n text/plain\n Wed, 1 Dec 2021 01:46:43 GMT\n /examplebucket/1.txt

[data]

注意： Content-Type请求头包含在请求中，也包含在StringToSign中。但请求中没有Content-MD5请求头，所以StringToSign中是空行。

List Objects

列出examplebucket中的对象。

请求	StringToSign
GET /?prefix=1&max-keys=50 HTTP/1.1 Host: examplebucket.ks3-cn-beijing.ksyuncs.com Date: Wed, 1 Dec 2021 01:51:57 GMT Authorization: KSS AKLTA6qLnuowT6KzKybUQNCOTw:VpjI PQFR7PuTYnbZ1Xp/BrEgBSw=	GET\n \n \n Wed, 1 Dec 2021 01:51:57 GMT\n /examplebucket/

注意：CanonicalizedResource的结尾要有斜杠/，查询字符串为空。

获取ACL

获取examplebucket的访问控制权限配置信息。

请求	StringToSign
GET /?acl HTTP/1.1 Host: examplebucket.ks3-cn-beijing.ksyuncs.com Date: Wed, 1 Dec 2021 01:56:35 GMT Authorization: KSS AKLTA6qLnuowT6KzKybUQNCOTw:97pp TrAzwsJn5vYwCHajNWnq7Mw=	GET\n \n \n Wed, 1 Dec 2021 01:56:35 GMT\n /examplebucket/?acl

注意：在CanonicalizedResource中包含子资源查询字符串参数。

Delete Object

从examplebucket中删除对象。bucket在Path中指定，并使用x-kss-date请求头。

请求	StringToSign
DELETE /examplebucket/1.txt HTTP/1.1 Host: ks3-cn-beijing.ksyuncs.com Date: Wed, 1 Dec 2021 03:39:18 GMT x-kss-date: Wed, 1 Dec 2021 03:39:18 GMT Authorization: KSS AKLTA6qLnuowT6KzKybUQNCOTw:jUOK m9QlcWxLiR9BNw13+FlHKuw=	DELETE\n \n \n Wed, 1 Dec 2021 03:39:18 GMT\n x-kss-date:Wed, 1 Dec 2021 03:39:18 GMT\n /examplebucket/1.txt

注意：此请求使用x-kss-date请求头来替代Date请求头。

使用自定义元数据上传对象

下面的例子在上传对象时，指定了自定义的元数据。

请求	StringToSign
PUT /1.txt HTTP/1.1 Host: examplebucket.ks3-cn-beijing.ksyuncs.com Date: Wed, 1 Dec 2021 06:26:05 GMT X-Kss-Acl: public-read Content-Type: text/plain Content-MD5: u7iq5XwQTNpAyThDrV5tuA== X-Kss-Meta-key1: value1 X-Kss-Meta-key2: value2 X-Kss-Meta-key2: value3 Content-Disposition: attachment Content-Length: 10 Authorization: KSS AKLTA6qLnuowT6KzKybUQNCOTw:vK9N g6vkG6bJWk3HDYby6QOOeBw=	PUT\n u7iq5XwQTNpAyThDrV5tuA== text/plain\n Wed, 1 Dec 2021 06:26:05 GMT\n x-kss-acl:public-read\n x-kss-meta-key1:value1\n x-kss-meta-key2:value2\n /examplebucket/1.txt

[data]

注意：“x-kss-”请求头被排序，并转换为小写字符。只有Content-Type, Content-MD5请求头被加入到了StringToSign中，但是其他的Content-*请求头没有被加入。

List Buckets

请求	StringToSign

```
GET / HTTP/1.1                                  GET\n
Host: ks3-cn-beijing.ksyuncs.com                \n
Date: Wed, 1 Dec 2021 06:29:04 GMT              \n
Authorization: KSS AKLTA6qLnuowT6KzKybUQNCOTw:G8TT Wed, 1 Dec 2021 06:29:04 GMT\n
lgydlSkLIgSyG6kYP+IcF+A=                        /
```

对象名被编码

请求	StringToSign
PUT /%E6%B5%8B%E8%AF%95.txt HTTP/1.1 Host: examplebucket.ks3-cn-beijing.ksyuncs.com Date: Wed, 1 Dec 2021 06:32:40 GMT Authorization: KSS 7799e793ce4624ee7e5a:dxhSBHoI6e VSPcXJqEghlUzZMnY=	PUT\n \n text/plain\n Wed, 1 Dec 2021 06:32:40 GMT\n /examplebucket/%E6%B5%8B%E8%AF%95.txt

注意：StringToSign中的对象名称被URL编码。

通过 URL QueryString 发送签名

除了将签名信息加入请求头外，用户还可以将签名信息加入到URL查询参数中。这样就可以通过浏览器直接访问对象数据。

注意：使用URL签名方式时，有将您授权的数据在过期时间内暴露在互联网上的风险，建议您预先评估后再使用。

URL中包含签名的示例如下：

```
http://examplebucket.ks3-cn-beijing.ksyuncs.com/1.txt?KSSAccessKeyId=AKLTA6qLnuowT6KzKybUQNCOTw&Expires=1638345010&Signature=0INTzi%2FDcz2sjL6O6LCncO0U05E%3D
```

注意：在URL中实现签名，必须至少包含Signature，Expires，KSSAccessKeyId三个参数。

请求参数

请求参数名称	示例中的值	描述
KSSAccessKeyId	AKLTA6qLnuowT6KzKybUQNCOTw	用户的AccessKeyId，获取方式参见获取AK/SK。
Expires	1638345010	签名过期时间，是UNIX时间，即自1970 年 1 月 1 日 00:00:00 UTC 以来的秒数。 如果KS3接收到URL请求的时间晚于签名中包含的Expires参数，则返回请求超时的错误码。
Signature	0INTzi%2FDcz2sjL6O6LCncO0U05E%3D	计算后的签名

URL中包含签名的方法和请求头中包含签名的算法基本一样，主要区别如下：

1) 通过URL包含签名时，之前的Date参数换成Expires参数。

2) 不支持同时在URL和Head中包含签名。

3) 如果传入的Signature，Expires，KSSAccessKeyId出现不止一次，以首次为准。

4) 请求先验证请求时间是否晚于Expires时间：

- 如果请求时间晚于Expires时间，则返回错误响应码；
- 如果请求时间不晚于Expires时间，则验证签名。

与AWS S3的兼容性

KS3同时支持KS3 V2和AWS V2签名。

- 当通过HTTP请求头发送签名时，如果Authorization的值以KSS开头，那么根据KS3 V2进行签名，将x-kss开头的请求头加入签名计算；如果Authorization的值以AWS开头，那么根据AWS V2签名，将x-amz开头的请求头加入签名计算。
- 当通过URL请求参数发送签名时，如果参数为KSSAccessKeyId，那么根据KS3 V2进行签名；如果参数为AWSAccessKeyId，那么根据AWS V2进行签名。

请求签名V4

使用对象存储服务（KS3） 时，可通过 RESTful API 对 KS3 发起 HTTP 匿名请求或 HTTP 签名请求，对于签名请求，KS3服务器端将会对请求发起者进行身份验证。

- 匿名请求：HTTP 请求不携带任何身份标识和鉴权信息，通过 RESTful API 进行 HTTP 请求操作。
- 签名请求：HTTP 请求时携带签名，KS3服务器端收到消息后，进行身份验证，验证成功则可接受并执行请求，否则将会返回错误信息并丢弃此请求。

KS3提供V4和V2两种签名方式，推荐使用V4签名方式。KS3 V4签名兼容 AWS Signature Version 4，签名方法参见Authenticating Requests - AWS Signature Version 4。

注意：本文描述的是HTTP RESTful API 请求签名方式，如果您使用 SDK 进行开发，SDK中会包含签名计算方法，您无需单独计算签名。仅在您希望通过原始 API 进行开发时，需要根据本文所描述步骤进行签名计算。

V4签名原理

创建V4签名的步骤简述如下，后面将详述各个步骤的操作方法。

1. 创建规范请求。
2. 利用规范请求和其他信息创建待签名的字符串（StringToSign）。
3. 利用 SecretKey 生成签名密钥，然后将该签名密钥与在上一步中创建的字符串结合使用来创建签名。
4. 将生成的签名添加到 HTTP 请求头中或者作为参数添加到 URL 中。

认证方式

用户可以通过以下几种方式来发送签名信息：

- 通过 HTTP Authorization 请求头发送签名请求，这是验证KS3请求的最常用方法。所有KS3 API操作（Post Object操作除外）都需要此请求头。
- 通过 URL QueryString 发送签名
- 在HTTP Post 表单中包含签名

通过HTTP Authorization 请求头发送签名

概述

Authorization请求头示例如下（增加换行是为了方便阅读，实际为空字符串）：

```
Authorization: KSS4-HMAC-SHA256
Credential=AKLTYS28X1sBSXCMRPMxU77t7A/20211129/SHANGHAI/ks3/kss4_request,
SignedHeaders=content-type;host;user-agent;x-kss-content-sha256;x-kss-date,  Signature=6d9df5717ebb7cdeae7d08d817b80d6a90ca32320b07e46f30c64136eade9d48
```

组成字段说明如下：

字段名称	描述
KSS4-HMAC-SHA256	用于签名的算法，固定值。

Credential	用户的AccessKeyId和范围信息，范围信息包括请求日期、区域、服务、终止字符串kss4_request，格式如下： \/\/\/\/kss4_request 其中： date格式为YYYYMMDD。 region：参见Endpoint与Region的对应关系 中的Region英文名称 service：固定值为ks3
SignedHeaders	已签名请求头的列表。该列表只需包含请求头名字，用分号分隔，必须全部小写，并按字符顺序对其进行排序，示例如下：host;range;x-amz-date
Signature	计算出的256位签名信息，以64个小写十六进制字符串形式表示。

有两种签名计算方式：

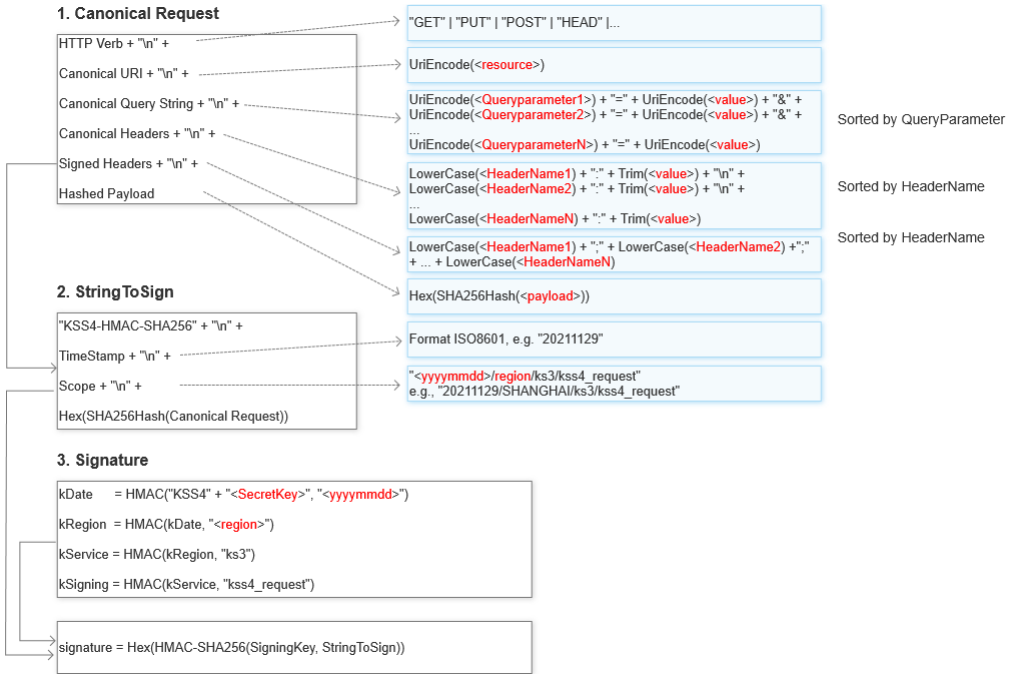
1）签名负载方式 - 用户可以选择计算整个负载（即请求体）的checksum，并将其包含在签名计算中。这种方式提高了安全性，但用户需要读取两次负载或将其缓冲在内存中。我们建议用户使用包含负载checksum的签名方式，以增强安全性。

2）无签名负载方式 - 在签名计算中不包括负载的checksum。

上述两种方式，都必须携带x-kss-content-sha256请求头，如果选择签名负载方式，请将x-kss-content-sha256请求头的值设置为负载的checksum值，否则将值设置为文本字符串 UNSIGNED-PAYLOAD。

签名过程

签名流程图



下表描述了图中显示的方法，用户需要为这些函数实现代码。

功能	描述
Lowercase()	将字符串转换为小写。
Hex()	小写16进制编码。
SHA256Hash()	安全散列算法（SHA）加密散列函数。
HMAC-SHA256()	使用签名密钥，根据SHA256算法计算出的签名值。
Trim()	删除任何前导或尾随空格。
UriEncode()	URI编码每个字节。UriEncode（）必须强制执行以下规则： URI编码除了下面字符之外的每个字节：'A' - 'Z', 'a' - 'z', '0' - '9', '-', '.', '_'和'~'。 空格字符是保留字符，必须编码为“%20”（而不是“+”）。 每个URI编码字节由'%'和两位十六进制值组成。 十六进制值中的字母必须为大写，例如“%1A”。 除了对象名之外，对正斜杠字符'/'进行编码。 例如，如果对象名称为photos/Jan/sample.jpg，则不对名称中的正斜杠进行编码。 重要建议用户编写自己的自定义UriEncode函数，以确保您的编码可以正常工作。

注意：以下代码是Java中的示例UriEncode（）函数。

```
StringBuilder result = new StringBuilder();
for (int i = 0; i < input.length(); i++) {
char ch = input.charAt(i);
if ((ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z') || (ch >= '0' && ch <= '9') || ch == '_' || ch == '-' || ch == '~' || ch == '.') {
result.append(ch);
} else if (ch == '/') {
result.append(encodeSlash ? "%2F" : ch);
} else {
result.append(toHexUTF8(ch));
}
}
return result.toString();
}
```

创建规范请求

将请求的内容（包括请求方法、URI、请求头等）组织为标准规范格式。伪代码如下：

```
CanonicalRequest =  HTTPRequestMethod + '\n' +  CanonicalURI  + '\n' +  CanonicalQueryString + '\n' +  CanonicalHeaders + '\n' +  SignedHeaders + '\n' +  HexEncode(Hash(RequestPayload))
```

HTTPMethod

- HTTP方法，例如GET，PUT，HEAD和DELETE等。

CanonicalURI

- URI的绝对路径，以域名后面的“/”开头，直到字符串的末尾或者问号字符（'?'）截止(不包含?及后面的查询参数)。例如/examplepath/myphoto.jpg。
- 对于查询bucket 列表接口，请求资源为'/'
- 若BucketName作为Host一部分，文件名字为空时，请求资源为'/'
- 对请求资源进行urlencode，'/'不做encode

CanonicalQueryString

CanonicalQueryString是URI编码后的查询字符串参数。用户需要单独对参数名称和值进行URI编码。并需要按参数名称的字母顺序，对参数进行排序，排序在编码后进行。具有重复名称的参数应按值进行排序。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。

以下URI示例中的查询字符串是 prefix=somePrefix&marker=someMarker&max-keys=20：

```
http://examplebucket.ks3-cn-beijing.ksyuncs.com/?prefix=somePrefix&marker=someMarker&max-keys=20
```

CanonicalQueryString的构造方式如下（为了便于阅读，添加了换行符）：

```
UriEncode("marker")+"="+UriEncode("someMarker")+"&"+
UriEncode("max-keys")+"="+UriEncode("20")  + "&" +
UriEncode("prefix")+"="+UriEncode("somePrefix")
```

当请求的目标是子资源时，相应的查询参数的值设置为空字符串（“”）。例如，下面的请求用于设置bucket的ACL权限：

```
http://examplebucket.ks3-cn-beijing.ksyuncs.com/?acl
```

在这种情况下，CanonicalQueryString为：

```
UriEncode("acl") + "=" +  ""
```

如果URI中不包含“?”，则请求中不存在查询字符串，此时将CanonicalQueryString设置为空字符串（“”），但仍需要包含“\ n”。

CanonicalHeaders

CanonicalHeaders是请求头的列表。请求头名称转成小写，请求头名称和值之间用冒号（":"）分隔，各行之间用换行符（“\ n”）分隔，并对请求头名称按字母顺序进行排序，示例如下：

```
Lowercase(<HeaderName1>)+":"+Trim(<value>)+"\n"
Lowercase(<HeaderName2>)+":"+Trim(<value>)+"\n"
...
Lowercase(<HeaderNameN>)+":"+Trim(<value>)+"\n"
```

CanonicalHeaders列表必须包括以下内容：

- HTTP Host请求头：
  - 如使用Virtual Hosted-Style访问方式（例如：https://bucket-name.ks3-cn-region.ksyuncs.com/object-name），host需要包含bucket名称（host:bucket-name.ks3-cn-region.ksyuncs.com）。
  - 如使用Path-Style访问方式（例如：https://ks3-cn-region.ksyuncs.com/bucket-name/object-name），host不需要包含bucket名称（host:ks3-cn-region.ksyuncs.com）。
- 如果存在Content-Type请求头，则必须将其添加到 CanonicalHeaders列表中。
- 所有x-kss-开头的请求头
- 对于临时授权，需要添加x-kss-security-token
- x-kss-content-sha256请求头

除了上述列出的必须增加的头，只计算SignedHeaders中包含的头。注意，Authorization不参与签名内容计算。

以下是CanonicalHeaders 的示例，请求头名称为小写并已排序。

```
host:bucket-name.ks3-cn-beijing.ksyuncs.com
x-kss-content-sha256:e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
x-kss-date:20211130T070023Z
```

SignedHeaders

为防止用户请求头被篡改，将参与签名运算的请求头加入到SignedHeaders中。SignedHeaders是按字母顺序排序的，以分号分隔的小写请求头名称列表。列表中的请求头与用户在CanonicalHeaders字符串中包含的请求头相同。

格式为：

```
Lowercase(<HeaderName1>)+";"+Lowercase(<HeaderName2>)+";"+...+Lowercase(<HeaderNameN>)
```

例如，对于前面的示例，SignedHeaders的值为：

```
host;x-kss-content-sha256;x-kss-date
```

> 注意：时间戳（x-kss-date或Date请求头）15分钟内有效，没有权限的用户可通过截获已签名的请求，并篡改SignedHeaders中没有包含的部分来伪造请求，所以为确保您的数据安全，建议您签名所有请求头和请求体。

HashedPayload

HashedPayload是请求体的SHA256哈希的十六进制值。

如果请求中没有请求体，则计算空字符串的哈希值，如下所示：

```
Hex(SHA256Hash(""))
```

哈希返回以下值：

e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855

例如，当用户使用PUT请求上传对象时，用户可以在请求体中提供对象数据。使用GET请求检索对象时，没有请求体，所以计算空字符串的哈希。

x-kss-content-sha256 请求头是必须的，取值范围如下：

值	描述
HexEncode(Sha256Hash(RequestPayload))	请求体经过哈希处理的以小写十六进制字符串表示形式
UNSIGNED-PAYLOAD	忽略请求正文的哈希校验
STREAMING-KSS4-HMAC-SHA256-PAYLOAD	chunk传输校验，暂不支持

计算方法为：

```
HashedPayload = Lowercase(HexEncode(SHA256Hash(requestPayload)))
```

使用 SHA256 哈希函数对请求体创建哈希值。V4签名不需要您使用特定字符编码来对负载中的文本进行编码。

如果负载为空，则使用空字符串作为哈希函数的输入。

创建待签名字符串

待签名的字符串格式如下：

```
"KSS4-HMAC-SHA256" + "\n" + <RequestDateTime> + "\n" + <CredentialScope> + "\n" + Hex(SHA256Hash(<CanonicalRequest>))
```

- 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 KSS4-HMAC-SHA256。
- 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-kss-date 标头中指定的。此值必须与您在前面所有步骤中使用的值匹配。
- 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（“kss4_request”）。区域和服务名称字符串必须采用 UTF-8 编码。即格式为：

```
date.Format(<YYYYMMDD>) + "/" + <region> + "/" + <service> + "/kss4_request"
```

- 日期必须为 YYYYMMDD 格式。请注意，日期不包括时间值。
- 请确保您指定的区域是您将请求发送到的目标区域。 KS3的region 示例：如 中国北京 为 BEIJING ，其余对应关系请参阅Region英文名称。
- service为ks3。

计算签名

使用 secretKey作为初始哈希操作的密钥，对请求日期、区域和服务执行一系列加密哈希操作（HMAC 操作），从而生成签名密钥。伪代码如下：

```
kSecret = your Secret Key
kDate = HMAC("KSS4" + kSecret, Date)
kRegion = HMAC(kDate, Region)
kService = HMAC(kRegion, Service)
SigningKey = HMAC(kService, "kss4_request")
```

最终签名是使用签名密钥作为密钥，对待签名字符串计算得到HMAC-SHA256哈希值。伪代码如下：

```
HMAC-SHA256(SigningKey, StringToSign)
```

请注意，哈希过程中所使用的日期的格式为 YYYYMMDD（例如，20150830），不包括时间。

确保以正确的顺序为您要使用的编程语言指定 HMAC 参数。在此示例中，密钥是第1个参数，数据（消息）是第2个参数，但您使用的函数可能以不同顺序指定密钥和数据。

使用摘要（二进制格式）来派生密钥。大多数语言都有用来计算二进制格式哈希（通常称为摘要）或十六进制编码哈希（称为十六进制摘要）的函数。派生密钥需要使用二进制格式摘要。

示例：

```
HMAC(HMAC(HMAC(HMAC("KSS4" + kSecret,"20211129"),"BEIJING"),"ks3"),"kss4_request")
```

将签名信息添加到请求头

在计算签名后，将其添加到请求的 HTTP 请求头Authorization中，伪代码如下：

```
Authorization: <algorithm> Credential=<ak>/<credential_scope>, SignedHeaders=<SignedHeaders>, Signature=<signature>
```

示例

以下是使用V4签名的示例。示例中使用的访问密钥如下：

参数	值
KSSAccessKeyId	AKLTA6qLnuowT6KzKybUQNCOTw
KSSSecretAccessKey	OCd5HzFDU1YDUG6eTHASvdt1RRn5bqKNKd18JxuFrYne+bazX7gmoYUG73XjJ/d2sg==

Bucket名称：examplebucket。访问的域名是ks3-cn-beijing.ksyuncs.com, region是BEIJING

示例：GET对象

从存储桶 examplebucket中获取对象1.txt的前5个字节。请求如下：

```
GET /1.txt HTTP/1.1
x-kss-content-sha256: e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
Authorization: SignatureToBeCalculated
x-kss-date: 20211130T061712Z
Range: bytes=0-4
Host: examplebucket.ks3-cn-beijing.ksyuncs.com
```

由于此GET请求不提供任何请求体内容，因此该 x-amz-content-sha256请求头的值是空请求体的哈希值。以下步骤显示Authorization请求头的计算的方法。

1)StringToSign

a.创建规范请求

```
GET
/1.txt

host:examplebucket.ks3-cn-beijing.ksyuncs.com
range:bytes=0-4
x-kss-content-sha256:e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
x-kss-date:20211130T062035Z
host;range;x-kss-content-sha256;x-kss-date
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

其中，最后一行是空请求体的hash值。第三行是空，因为此请求不包含请求参数。

b.待签名字符串

```
KSS4-HMAC-SHA256
20211130T062035Z
20211130/BEIJING/ks3/kss4_request
e124a1d2400e6c08fdfc78c02a62f8a8900d67d577ffedc1820347794a106dfe
```

2)生成签名密钥

```
signing key = HMAC-SHA256(HMAC-SHA256(HMAC-SHA256(HMAC-SHA256("KSS4" + "<YourSecretAccessKey>","20211130"),"BEIJING"),"ks3"),"kss4_request")
```

3)计算后的签名

```
0b6e5f3e77ca9e0201c4033916a796c232ebe244c2a42f23493d7aba45217f09
```

4)Authorization请求头

```
Authorization: KSS4-HMAC-SHA256
Credential=AKLTA6qLnuowT6KzKybUQNCOTw/20211130/BEIJING/ks3/kss4_request,
SignedHeaders=host;range;x-kss-content-sha256;x-kss-date,
Signature=0b6e5f3e77ca9e0201c4033916a796c232ebe244c2a42f23493d7aba45217f09
```

示例：PUT对象

在存储桶中上传对象1.txt。

```
PUT /1.txt  HTTP/1.1
x-kss-content-sha256: 7509e5bda0c762d2bac7f90d758b5b2263fa01ccbc542ab5e3df163be08e6ca9
Authorization: SignatureToBeCalculated
x-kss-date: 20211130T062938Z
x-kss-storage-class: STANDARD
Host: examplebucket.ks3-cn-beijing.ksyuncs.com
Content-Length: 12

hello world!
```

以下步骤显示Authorization请求头的计算的方法。

1)StringToSign

a.创建规范请求

```
PUT
/1.txt

content-length:12
host:examplebucket.ks3-cn-beijing.ksyuncs.com
x-kss-content-sha256:7509e5bda0c762d2bac7f90d758b5b2263fa01ccbc542ab5e3df163be08e6ca9
x-kss-date:20211130T062938Z
x-kss-storage-class:STANDARD

content-length;host;x-kss-content-sha256;x-kss-date;x-kss-storage-class
7509e5bda0c762d2bac7f90d758b5b2263fa01ccbc542ab5e3df163be08e6ca9
```

其中，第三行是空，因为此请求不包含请求参数。最后一行是请求体的hash值，它应该与x-kss-content-sha256 请求头的值相同。

b.待签名字符串

```
KSS4-HMAC-SHA256
20211130T062938Z
20211130/BEIJING/ks3/kss4_request
35bc694c8cc1176f94aa68fcb2ccc01303d8190c4de88f76c5989cbfaecdb626
```

2)生成签名密钥

```
signing key =  HMAC-SHA256(HMAC-SHA256(HMAC-SHA256(HMAC-SHA256("KSS4" + "<YourSecretAccessKey>","20211130"),"BEIJING"),"ks3"),"kss4_request")
```

3)计算后的签名

```
87e3404b5aa78b92f1453ee16a9274c52e42b414eab576e8d25c212bb53dc0b0
```

4)Authorization请求头

```
Authorization: KSS4-HMAC-SHA256
Credential=AKLTA6qLnuowT6KzKybUQNCOTw/20211130/BEIJING/ks3/kss4_request,
SignedHeaders=content-length;host;x-kss-content-sha256;x-kss-date;x-kss-storage-class,
Signature=87e3404b5aa78b92f1453ee16a9274c52e42b414eab576e8d25c212bb53dc0b0
```

示例：列出存储空间中的对象

列出存储空间 examplebucket中的对象，prefix设置为“1”，最多返回2个对象。请求如下：

```
GET /?max-keys=2&prefix=1  HTTP/1.1
x-kss-content-sha256: e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855Authorization: SignatureToBeCalculated
x-amz-date: 20211130T063717Z
Host: examplebucket.ks3-cn-beijing.ksyuncs.com
```

以下步骤显示Authorization请求头的计算的方法。

1)StringToSign

a.创建规范请求

```
GET
/
max-keys=2&prefix=1
host:examplebucket.ks3-cn-beijing.ksyuncs.com
x-kss-content-sha256:e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
x-kss-date:20211130T063717Z

host;x-kss-content-sha256;x-kss-date
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

其中，最后一行是空请求体的hash值。

b.待签名字符串

```
KSS4-HMAC-SHA256
20211130T063717Z
20211130/BEIJING/ks3/kss4_request
ec5654b7a599933116a221760119535b4c75552ec6c629d69580c826a3f77e76
```

2)生成签名密钥

```
signing key = HMAC-SHA256(HMAC-SHA256(HMAC-SHA256(HMAC-SHA256("KSS4" + "<YourSecretAccessKey>","20211130"),"BEIJING"),"ks3"),"kss4_request")
```

3)计算后的签名

```
2db9781b81a2b21852964b2dec0b07f58d0d1355fdedb27a9513294cb5776f9b
```

4)Authorization请求头

```
Authorization: KSS4-HMAC-SHA256
Credential=AKLTA6qLnuowT6KzKybUQNCOTw/20211130/BEIJING/ks3/kss4_request,
SignedHeaders=host;x-kss-content-sha256;x-kss-date,
Signature=2db9781b81a2b21852964b2dec0b07f58d0d1355fdedb27a9513294cb5776f9b
```

通过查询参数发送签名

概述

用户可以使用查询字符串参数来发送签名信息，示例如下。

```
https://examplebucket.ks3-cn-beijing.ksyuncs.com/test.txt
?X-Kss-Algorithm=KSS4-HMAC-SHA256
&X-Kss-Credential=<your-access-key-id>/<yyyymmdd>/<region>/ks3/kss4_request
&X-Kss-Date=20211129T065239Z
&X-Kss-Expires=86400
&X-Kss-SignedHeaders=host
&X-Kss-Signature=<signature-value>
```

在示例URL中，请注意以下事项：

- 添加换行符是为了便于阅读。
- 该X-Kss-CredentialURL中的值仅显示可读性“/”字符。在实际中，它应编码为%2F。例如：

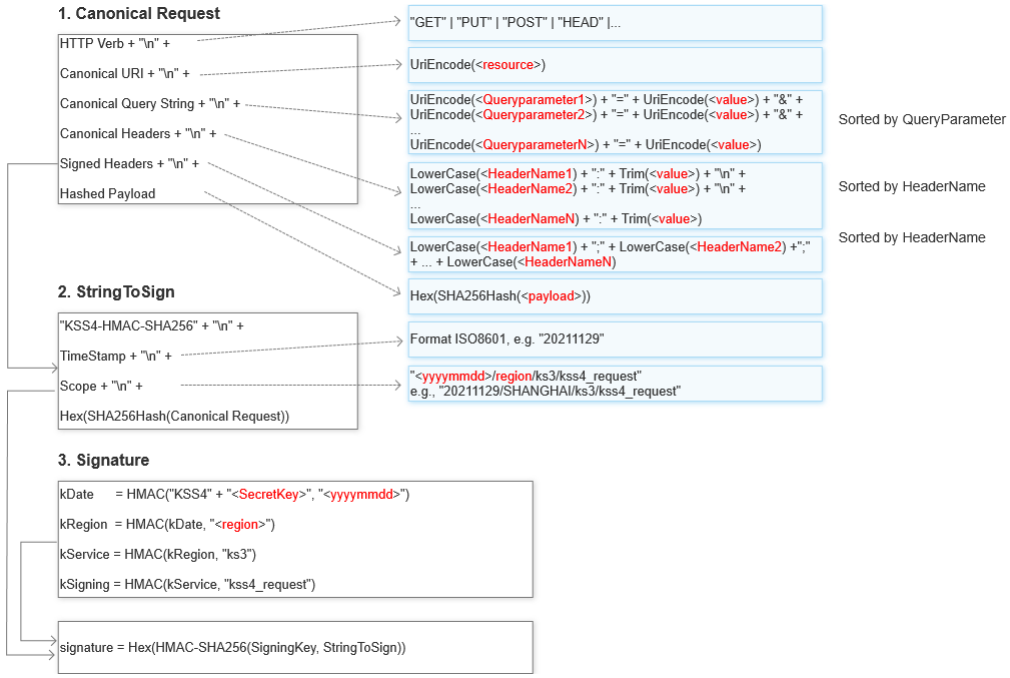
  ```
  &X-Kss-Credential=<your-access-key-id>%2F20211129%2F<region>%2Fks3%2Fkss4_request
  ```

  各参数含义说明如下。

参数名称	描述
X-Kss-Algorithm	签名算法:KSS4-HMAC-SHA256
X-Kss-Credential	用户的accessKeyId和范围信息，范围信息包括请求日期、区域、服务、终止字符串kss4_request，格式如下：\/\/\/\/kss4_request其中：date格式为YYYYMMDD。region：参阅Region英文名称 service：为ks3
X-Kss-Date	日期和时间格式必须遵循ISO 8601标准，并且必须使用“yyyyMMddTHHmmssZ”格式进行格式化。例如，如果日期和时间是“08/01/2018 15：32：41.982-700”，则必须首先将其转换为UTC（协调世界时），然后转换为“20180801T083241Z”。请求的时间戳不能大于（服务端时间戳+15分钟）
X-Kss-Expires	过期时间，单位秒。时间范围为1-604800（7天）。例如，86400（表示24小时）。请求时间戳+过期时间应大于服务端时间戳。
X-Kss-SignedHeaders	列出用于计算签名的标头。签名计算中需要以下标头： HTTP Host请求头。 x-kss-*请求头。
X-Kss-Signature	签名结果

签名过程

签名过程如下图所示。



使用参数的签名过程与使用请求头的签名过程类似，说明如下：

- 由于创建签名URL的时候，并不知道请求体的内容，所以设置常量UNSIGNED-PAYLOAD。
- 规范查询字符串（Canonical Query String）必须包括除了X-Kss-Signature之外的所有查询参数。
- 规范请求头（Canonical Headers）必须包括HTTP Host请求头和x-kss-开头的请求头，这些请求头都将参与签名计算。

示例

通过创建签名URL的方式，与其他人共享examplebucket中1.txt对象，过期时间设置为7天（604800秒）。以GET请求为例：

```
GET
http://examplebucket.ks3-cn-beijing.ksyuncs.com/1.txt
?X-Kss-Algorithm=KSS4-HMAC-SHA256
&X-Kss-Credential=AKLTA6qLnuowT6KzKybUQNCOTw%2F20211130%2FBEIJING%2Fks3%2Fkss4_request
&X-Kss-Date=20211130T075703Z
&X-Kss-Expires=604800
&X-Kss-SignedHeaders=host
&X-Kss-Signature=<signature-value>
```

以下步骤首先说明如何计算签名和构建预签名URL。示例中使用的访问密钥如下：

参数	值
KSSAccessKeyId	AKLTA6qLnuowT6KzKybUQNCOTw
KSSSecretAccessKey	OCd5HzFDU1YDUG6eTHASvdt1RRn5bqKNKd18JxuFrYne+bazX7gmoYUG73XjJ/d2sg==

1)StringToSign

a. 创建规范请求（以GET请求为例）

```
GET
/1.txt
X-Kss-Algorithm=KSS4-HMAC-SHA256&X-Kss-Credential=AKLTA6qLnuowT6KzKybUQNCOTw%2F20211130%2FBEIJING%2Fks3%2Fkss4_request&X-Kss-Date=20211130T075703Z&X-Kss-Expires=604800&X-Kss-SignedHeaders=host
host:examplebucket.ks3-cn-beijing.ksyuncs.com

host
UNSIGNED-PAYLOAD
```

b. 待签名字符串

```
KSS4-HMAC-SHA256
20211130T075703Z
20211130/BEIJING/ks3/kss4_request
19469bd87d923505aa26d4596f44ffc24b0a1bc65c2a15c149bfd31621d06488
```

2)生成签名密钥

```
signing key =  HMAC-SHA256(HMAC-SHA256(HMAC-SHA256(HMAC-SHA256("KSS4" + "<YourSecretAccessKey>","20211130"),"BEIJING"),"ks3"),"kss4_request")
```

3)计算后的签名

```
f6c0682252a278ca84ea2f4acbff6cefe15d9529b3ef678ee3d0ec452c697b00
```

4)预签名URL

```
http://examplebucket.ks3-cn-beijing.ksyuncs.com/1.txt?X-Kss-Algorithm=KSS4-HMAC-SHA256&X-Kss-Credential=AKLTA6qLnuowT6KzKybUQNCOTw%2F20211130%2FBEIJING%2Fks3%2Fkss4_request&X-Kss-Date=20211130T075703Z&X-Kss-Expires=604800&X-Kss-SignedHeaders=host&X-Kss-Signature=f6c0682252a278ca84ea2f4acbff6cefe15d9529b3ef678ee3d0ec452c697b00
```

在表单中包含签名

对于Post Object请求，HTML 表单中必须包含 Policy 和 Signature 信息。 计算 Signature 的具体流程为：

1. 创建一个 UTF-8 编码的 Policy文档，如何构建policy ，详见Policy构建方法。
2. 将 Policy 进行 base64 编码，其值即为 Policy 表单域中填入的值，将该值作为签名的字符串(StringToSign)。
3. 签名Signature =HMAC-SHA256(SigningKey, StringToSign)

表单项	说明
policy	表单上传安全策略
x-kss-algorithm	签名算法：KSS4-HMAC-SHA256
x-kss-credential	签名范围：格式为\/\/\/\/kss4_request
x-kss-date	请求时间:时间格式为 yyyyMMddTHHmmssZ （X-Kss-Date<=now+15m） 请求的时间戳不能大于（服务器时间戳+15m）
x-kss-signature	签名结果（计算签名时queryString不包含X-Kss-Signature）

其它说明

日期

您在凭证范围中使用的日期必须与您的请求的日期匹配。您可以用多种方法将日期包括在请求中。您可以使用 date 请求头或 x-kss-date请求头，或者将 x-kss-date作为查询参数包含在内。

时间戳必须采用 UTC 表示，并具有以下 ISO 8601 格式：YYYYMMDD’T’HHMMSS’Z’。例如，20150830T123600Z 是有效时间戳。请勿在时间戳中包含毫秒。

KS3先检查时间戳的x-kss-date请求头或参数，如果无法找到 x-kss-date，则将寻找 date 请求头。 检查八位数字字符串形式的凭证范围，表示请求的年 (YYYY)、月 (MM) 和日 (DD)。例如，如果 x-kss-date 标头值为 20111015T080000Z，并且凭证范围的日期部分为 20111015，则允许身份验证过程继续执行。如果日期不匹配，则拒绝请求，即使时间戳距离凭证范围中的日期仅有数秒之差也是如此。例如，将拒绝其x-kss-date请求头值为 20151014T235959Z 且凭证范围包括日期 20151015 的请求。

URLencode方法

- 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 (.) 和波形符 (~)
- 空格字符必须编码为 %20（不像某些编码方案那样使用“+”）
- 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），扩展 UTF-8 字符必须采用格式 %XY%ZA%BC
- 除了对象名称之外，对前斜杠字符‘/’进行编码

与AWS S3的兼容性

KS3同时支持KS3 V4和AWS V4签名， 如果签名算法为KSS4-HMAC-SHA256，则根据KS3 V4进行签名，如果签名算法为AWS4-HMAC-SHA256，则根据AWS V4签名。两种签名方式对比如下：

项目	S3	KS3
请求头前缀	x-amz-	x-kss-
queryString前缀	X-Amz-	X-Kss-
协议	AWS4	KSS4 下角标
服务名称	s3	ks3
固定值	aws4_request	kss4_request

公共请求头

下表列出了KS3中常用的请求头部。

名称	描述
Authorization	必要的请求验证信息。匿名用户不需要。
Content-Length	请求体的长度信息（不包含头部），以字节为单位。
Content-Type	请求体中资源的类型。例如： text/plain 。
Content-MD5	请求对象的MD5摘要信息。
Date	当前请求的时间和日期。格式如: Wed, 01 Mar 2006 12:00:00 GMT。
Expect	当用户应用使用100-continue，认证通过才会发送请求体。如果头部信息认证无效，则不会发送请求内容。 有效值: 100-continue
Host	路径样式的请求，其值为：ks3-cn-beijing.ksyuncs.com。 虚拟样式的请求中，样式为：BucketName.ks3-cn-beijing.ksyuncs.com。

公共响应头

下表列出了KS3中常用的响应头部。

名称	描述
Content-Length	响应体的字节数。 类型: String 默认值: 无
Content-Type	内容的 MIME 类型。例如：Content-Type: text/html; charset=utf-8 。 类型: String 默认值: 无
Connection	指定服务器的连接的开放和关闭。 类型: Enum 有效值: keep-alive | close 默认值: 无
Date	响应时的时间和日期，格式如：Wed, 01 Mar 2014 12:00:00 GMT 。 类型: String 默认值: 无
ETag	响应的对象的实体标签，与对象内容有关，与名称无关。 类型: String
Server	响应服务器名称。 类型: String 默认值: Tengine
x-kss-request-id	由KS3指定的唯一值，可用于解决KS3出现的问题。 类型: String 默认值: 无

Service相关

GET Service

描述

此GET操作将返回给发出请求的验证用户一个包含其所有bucket的列表。
你需要使用KS3颁发的AccessKey来验证请求。匿名请求不会返回bucket的列表，并且你也无法得到不属于你的bucket的列表。

请求

简单请求语法

```
GET / HTTP/1.1
Host: {endpoint}
Date: {date}
Authorization: {SignatureValue}
```

指定项目组时请求语法

```
GET  /?projectIds={projectId} HTTP/1.1
Host: {endpoint}
Date: {date}
Authorization: {SignatureValue}
```

说明：

- 如果需要返回项目ID为2345的项目下的bucket列表，请求为/?projectIds=2345
- 如需返回多个项目下的bucket列表，例如需返回项目ID为1234、3456、3344下所有的bucket列表，多个项目ID需以英文半角逗号分隔，请求为/?projectIds=1234,3456,3344
- 如果没有projectIds参数，或者projectIds参数为空，将返回所有项目下的bucket列表。

请求参数

该接口不使用请求参数。

请求头部

该接口只使用常用请求头部。获取更多信息，请点击常用请求头部。

响应

响应内容

名称	描述
ListAllMyBucketsResult	响应信息容器 类型：Container 子节点：Owner, Buckets 父节点：无
Owner	包含bucket拥有者信息的容器 类型：Container 父节点：ListAllMyBucketsResult
ID	Bucket拥有者的用户ID 类型：String 父节点：ListAllMyBucketsResult.Owner
DisplayName	Bucket拥有者的名称 类型：String 父节点：ListAllMyBucketsResult.Owner
Buckets	包含一个或多个bucket的容器 类型：Container 子节点：Bucket 父节点：ListAllMyBucketsResult
Bucket	包含bucket信息的容器 类型：Container 子节点：Name, CreationDate,Type,Region 父节点：ListAllMyBucketsResult.Buckets
Name	Bucket的名字 类型：String 父节点：ListAllMyBucketsResult.Buckets.Bucket
CreationDate	bucket的创建日期 类型：date(yyyy-mm-ddThh:mm:ss.timezone, e.g., 2009-02-03T16:45:09.000Z) 父节点：ListAllMyBucketsResult.Buckets.Bucket
Type	Bucket的类型，NORMAL为非归档类型，ARCHIVE为归档类型 类型：String 父节点：ListAllMyBucketsResult.Buckets.Bucket
Region	Bucket所在区域 类型：String 父节点：ListAllMyBucketsResult.Buckets.Bucket

特殊错误

该接口不返回任何特殊错误。

示例

简单请求示例

```
GET / HTTP/1.1
Host: ks3-cn-beijing.ksyuncs.com
Date: Wed, 01 Jan  2014 12:00:00 GMT
Authorization: authorization string
```

响应示例

```
<?xml version="1.0" encoding="UTF-8"?>
<ListAllMyBucketsResult>
  <Owner>
    <ID>bcaf1ffd86f461ca5fb16fd081034f</ID>
    <DisplayName>webfile</DisplayName>
  </Owner>
  <Buckets>
    <Bucket>
      <Name>quotes</Name>
      <CreationDate>2014-01-01T16:45:09.000Z</CreationDate>
      <Type>NORMAL</Type>
      <Region>SHANGHAI</Region>
    </Bucket>
    <Bucket>
      <Name>samples</Name>
      <CreationDate>2014-01-01T16:41:58.000Z</CreationDate>
      <Type>ARCHIVE</Type>
      <Region>BEIJING</Region>
    </Bucket>
  </Buckets>
</ListAllMyBucketsResult>
```

指定项目组时请求示例

```
GET /?projectIds=0 HTTP/1.1
Host: ks3-cn-beijing.ksyuncs.com
Date: Wed, 01 Jan  2014 12:00:00 GMT
Authorization: authorization string
```

响应示例

```
<?xml version="1.0" encoding="UTF-8"?>
<ListAllMyBucketsResult>
    <Owner>
        <ID>bcaf1ffd86f461ca5fb16fd081034f</ID>
        <DisplayName>webfile</DisplayName>
    </Owner>
    <Buckets>
        <Bucket>
            <Name>quotes</Name>
            <CreationDate>2014-01-01T16:45:09.000Z</CreationDate>
            <Type>NORMAL</Type>
            <Region>SHANGHAI</Region>
        </Bucket>
    </Buckets>
</ListAllMyBucketsResult>
```

基本操作

访问控制权限（ACL）

生命周期（Lifecycle）

空间策略（Policy）

复制（Replication）

日志（Logging）

跨域资源共享（CORS）

回源（Mirror）

基本操作

分块上传

接口描述

当文件较大时，可以选择分块上传。把大文件进行切割上传到服务器。 分块上传分为三步：

1. Initiate Multipart Upload 初始化分块上传
2. Upload Part 上传文件块
3. Complete Multipart Upload 完成分块上传

上传中，你可以使用Abort Multipart Upload取消上传，或者List Parts查看上传的分块。或者List Multipart Uploads查看当前的bucket下有多少个uploadid。

注意：KS3最多支持10000个文件块

接口包括

- Initiate Multipart Upload 初始化分块上传
- Upload Part 上传文件块
- Complete Multipart Upload 完成分块上传
- Abort Multipart Upload 取消分块上传
- List Parts 查看已上传的块
- List Multipart Uploads 查看bucket下的分块上传
- Upload Part Copy 拷贝为块

访问控制权限（ACL）

对象标签（Tagging）

存储类型

描述

- KS3标准存储适合频繁访问、有热点存在的各类音视频、图片、网站静态资源的数据，较低的延迟和较高的吞吐量性能的数据。
- KS3低频存储用于保存不频繁访问但在需要时也要求快速访问的数据。
- KS3归档存储（ACHIVE)适合需要长期保存（建议3个月以上）的归档数据，在存储周期内很少被访问，数据进入到可读取状态需要1分钟到10分钟的解冻时间。适合需要长期保存的档案数据、医疗影像、科学资料、影视素材。

用户可以在上传文件过程中，通过HTTP头指定存储类型；相应的，在下载文件过程中我们会通过HTTP响应头指示文件的存储类型。除此之外，在用户枚举文件时也会返回文件的存储类型信息。

您想采取某个特定存储类型来存储文件，您可以通过以下方式：

- 利用上传接口，指定请求头x-kss-storage-class为STANDARD/ STANDARD_IA/ARCHIVE。
- 创建指定类型Bucket，上传Object时无需指定存储类型，即可按照默认Bucket的存储类型。
- 通过生命周期转化，设定存储类型转化规则，按照规则系统自动到期转化为归档存储类型

上传接口

Put Object 上传文件

用户上传过程中，可以通过设置 x-kss-storage-class 头来指定存储类型。有效值为：STANDARD、STANDARD_IA、ARCHIVE。对于无效的存储类型，KS3会拒绝本次请求。

上传文件时，当不指定请求头x-kss-storage-class时，如果Bucket是归档类型，Object自动为归档类型，如果Bucket是非归档类型，Object自动为标准类型；如果指定Object 的x-kss-storageClass则以用户指定为准，例如在非归档存储空间中，指定Object 的x-kss-storageClass为ARCHIVE时，则该文件为归档存储类型。

bucket类型	x-kss-storage-class请求头	文件类型
归档存储类型	不上传	归档类型
归档存储类型	ARCHIVE	归档类型
归档存储类型	STANDAND	标准类型
归档存储类型	STANDARD_IA	低频类型
非归档存储类型	不上传	标准类型
非归档存储类型	STANDAND	标准类型
非归档存储类型	STANDARD_IA	低频类型
非归档存储类型	ARCHIVE	归档类型

请求语法：

```
PUT /{ObjectKey} HTTP/1.1
Host: {BucketName}.{endpoint}
Date: {date}
Authorization: {SignatureValue}
x-kss-storage-class: {StorageClass}
```

注意：

- endpoint与Region对应关系

- SignatureValue签名算法

SDK示例代码：

```
PutObjectRequest request = new PutObjectRequest(bucketName, key, file);
request.setStorageClass(StorageClass.StandardInfrequentAccess);
PutObjectResult result = client.putObject(request); // 假设已经合理初始化Ks3Client
```

Post Object 上传文件

用户可以在表单域中通过设置 x-kss-storage-class 域来指定存储类型。规则和Put Object上传文件一致。

Initiate Multipart Upload 初始化分块上传

用户可以在初始化分块上传时通过 x-kss-storage-class 请求头设置文件的存储类型。有效值为： STANDARD 、STANDARD_IA 、ARCHIVE。规则和Put Object上传文件一致。对于无效的存储类型，KS3会拒绝本次请求。

请求语法：

```
POST /{ObjectKey}?uploads HTTP/1.1
Host: {BucketName}.{endpoint}
Date: {date}
Authorization: {SignatureValue}
x-kss-storage-class: {StorageClass}
```

SDK示例代码：

```
InitiateMultipartUploadRequest request = new InitiateMultipartUploadRequest(bucketName, key);
request.setStorageClass(StorageClass.StandardInfrequentAccess);
InitiateMultipartUploadResult result = client.initiateMultipartUpload(request); // 假设已经合理初始化Ks3Client
```

后续的所有分块上传操作(Upload Part、Complete、Abort)都不再接受 x-kss-storage-class 请求头。

Put Object Copy 复制文件

对于复制文件，KS3支持重新指定存储类型。若不指定目的文件的存储类型，则按照目标存储的默认存储类型保存。如果复制源文件是归档存储文件，成功复制的前提是，源文件必须完成解冻，为可读状态。

Upload Part Copy 复制分块

对于复制分块，KS3目前不支持重新指定存储类型。目的文件的存储类型必须与源文件存储类型一致。 对于复制分块，KS3会忽略 x-kss-storage-class 请求头。 如果源文件存储类型与目的文件存储类型不一致，会报错。提示 InvalidStorageClass。

下载接口

Get Object 下载文件

用户下载文件时，如果文件是低频存储文件，则在响应头中会出现 x-kss-storage-class 头；标准存储文件下载时没有这个响应头；归档文件必须完成解冻，才能正常下载。

```
GetObjectRequest request = new GetObjectRequest(bucketName, key);
GetObjectResult result = client.getObject(request);
Ks3Object object = result.getObject();
object.getObjectContent();
```

Head Object 查看文件元信息

Java SDK中，可以通过获取元信息来查看存储类型，示例代码如下：

```
HeadObjectRequest request = new HeadObjectRequest(bucketName, key);
HeadObjectResult result =client.headObject(request);
String storageClass = result.getObjectMetadata().getStorageClass();
```

对于标准存储文件，getStorageClass 返回 null；对于低频存储文件，getStorageClass 返回 STANDARD_IA；对于归档存储文件，getStorageClass 返回 ARCHIVE。

枚举接口

Get Bucket 枚举bucket下的所有文件

用户获取某个Bucket下文件时，KS3会为每个文件返回其存储类型。

SDK示例代码：

```
ListObjectsRequest request = new ListObjectsRequest(bucketName);
ObjectListing listing = client.listObjects(request);
for (Ks3ObjectSummary summary : listing.getObjectSummaries()) {
    String storageClas = summary.getStorageClass();
}
```

对于标准存储文件，getStorageClass 返回 STANDARD；对于低频存储文件，getStorageClass 返回 STANDARD_IA；对于归档存储文件，getStorageClass 返回 ARCHIVE。

List Multipart Uploads 查看bucket下的分块上传

用户在查看当前bucket下所有的分块上传请求时，KS3会为每个分块上传返回其存储类型。

SDK示例代码：

```
ListMultipartUploadsRequest request = new ListMultipartUploadsRequest(bucketName);
ListMultipartUploadsResult result = client.listMultipartUploads(request);
for (MultiPartUploadInfo uploadInfo : result.getUploads()) {
    String storageClass = uploadInfo.getStorageClass();
}
```

对于标准存储文件，getStorageClass 返回 STANDARD；对于低频存储文件，getStorageClass 返回 STANDARD_IA；对于归档存储文件，getStorageClass 返回 ARCHIVE。

List Parts 查看已上传的块

用户在列举某个分块上传已经上传的块时，KS3会返回当前这次分块上传的存储类型。

SDK示例代码：

```
ListPartsRequest reqest = new ListPartsRequest(bucketName, key, result.getUploadId());
ListPartsResult result = client.listParts(reqest);
String storageClass = result.getStorageClass();
```

对于标准存储文件，getStorageClass 返回 STANDARD；对于低频存储文件，getStorageClass 返回 STANDARD_IA；对于归档存储文件，getStorageClass 返回 ARCHIVE。

上传回调处理

支持上传回调处理的接口有PUT Object、COMPLETE MULTIPART UPLOAD、POST Object

1.PUT Object、COMPLETE MULTIPART UPLOAD

描述

在调用Put Object，Complete Multipart Upload接口时，可携带相关的Callback参数，实现上传回调处理（Upload CallBack Processing，UCP）。

客户端将文件及相关Callback请求头上传到KS3后，KS3会通过POST方法向用户的回调地址POST一段json数据，此json数据为客户端传入的x-kss-callbackbody请求头填充变量后转化得来。用户服务端正确处理回调后返回{"result":true}，如果回调处理错误返回{"result":false}则表示文件上传失败，如果用户服务端没有返回任何消息，则进入超时等待，默认等待时间为3秒，重试两次。只有在KS3服务端收到{"result":true}时，文件才真正上传成功。
支持的头部参数里面加入x-kss-callbackauth 值为”1”时表示开启回调鉴权功能，具体详见第三部分“回调鉴权”。

请求接口

上传回调请求保持原有接口不变，参考Put Object,Complete Multipart Upload。

请求参数

该接口不使用请求参数。

请求头部

名称	描述	是否必选
x-kss-callbackurl	支持http和https,接收回调的服务器地址	是
x-kss-callbackbody	回调参数支持自定义参数返回、常量和魔法变量，自定义变量通过Header传回，例如objectKey=${key}&etag=${etag}&uid=123 回调参数的value值不能包含“=”和“&”	是
x-kss-callbackauth	值为”1”时表示开启回调鉴权功能。回调鉴权功能具体详见第三部分“回调鉴权”	否
kss-location	自定义头以kss-开始	自定义

魔法变量

参数	说明	备注
bucket	文件上传的Bucket	Utf-8编码
key	文件的名称	Utf-8编码
etag	文件Md5值经过base64处理	
objectSize	文件大小	以字节标识
mimeType	文件类型	
createTime	文件创建时间	Unix时间戳表示，1420629372，确切到秒

回调响应

回调结果

应用服务器返回响应给KS3，返回的回调内容需要包含：

- 指定的回调内容：{"result":"true"}或{"result":"false"}；如返回的内容不包含该内容，则回调失败（同时上传也会失败）
- 其他回调内容：Body体大小不能超过1MB，内容需为JSON格式；如Body体大小超过1MB，则回调失败（同时上传也会失败）
- 例如返回的回调请求为：

```
    HTTP/1.0 200 OK
    Server: BaseHTTP/0.3 Python/2.7.6
    Date: Mon, 14 Sep 2021 12:37:27 GMT
    Content-Type: application/json

{ "result":"true", "a":"b" }

####  上传结果：增加回调内容
回调成功后，KS3将回调应用服务器返回的内容返回给用户；如回调失败，将返回400  CallbackFail
```language
HTTP/1.1 200 OK
Date: Mon, 14 Sep 2015 12:37:27 GMT
Content-Type: application/json
Content-Length: 120
Connection: close
ETag: "D8E8FCA2DC0F896FD7CB4CB0031BA249"
Server: KS3

{
"result":"true",
"a":"b"
}
```

特殊错误

回调失败，将返回400 CallbackFail

2.POST Object

描述

采用POST回调时，需要在请求后面加入一个postcallback的参数，如POST /?postcallback HTTP/1.1，表示这是一个POST回调请求。当加入postcallback参数时，表单项中必须同时存在回调相关的参数，否则请求会判定不合法，具体与回调服务器交互过程与PUT回调一致。其它语法和普通POST请求一样，参考Post Object。

请求接口

上传回调请求保持原有接口不变，参考Post Object。

请求参数

参数	描述	是否必选
postcallback	表示这是一个POST回调请求	是

表单项

名称	描述	是否必选
x-kss-callbackurl	支持http(s)	是
x-kss-callbackbody	回调参数支持自定义参数返回、常量和魔法变量，自定义变量通过Header传回，例如objectKey=${key}&etag=${etag}&uid=123 回调参数的value值不能包含“=”和“&”	是
x-kss-callbackauth	值为”1”时表示开启回调鉴权功能。回调鉴权功能具体详见第三部分“回调鉴权”	否
kss-location	自定义头以kss-开始	自定义

魔法变量

参数	说明	备注
bucket	文件上传的Bucket	Utf-8编码
key	文件的名称	Utf-8编码
etag	文件Md5值经过base64处理	
```

objectSize	文件大小	以字节标识
mimeType	文件类型	
createTime	文件创建时间	Unix时间戳表示，1420629372，确切到秒

回调响应

响应内容

回调响应内容与Put Object回调内容及方式一致。

特殊错误

调用POST回调方法时，如果未传入回调必填的表单项，请求将返回400；如果传入了回调相关的表单项，但是调用的是普通（非回调）POST方法时，请求也会返回400。

回调鉴权

如果回调时用户开启了鉴权字段x-kss-callbackauth(该字段在PUT和Complete Multipart Upload请求中写在header里，在POST请求中写在表单项里)，则KS3向用户的回调地址返回的header中，就会包含Authorization和x-kss-date字段内容：

```
Authorization: ks3cbauth AKLTxxxxx5IxxxpA7xxxxxx:FIexxyY3jGeO7JxxxxxW5rBejKM=
  x-kss-date：1590132647609
```

其中ks3cbauth为固定值，AKLT开头到冒号前内容为用户此条POST请求使用的AK，冒号后面的部分是用此AK、对应SK和服务端指定时间（此处x-kss-date为回调发起时间即KS3服务器的时间）算出的签名串signature，算签名的方法和发送请求的v2签名计算方法类似，但简化一些：

```
signString = "ksscallback " + {timestamp} //形如"ksscallback 1590132647609"，中间有个空格，ksscallback是个固定值
  signature = Base64(HMAC-SHA1(YourSecretKey, UTF-8-Encoding-Of(signString)));
```

AWS S3协议兼容

目前KS3 API接口覆盖了AWS S3协议大部分的基本功能，并在持续扩充新特性。除少量AWS S3特殊特性接口未覆盖外，用户采用AWS S3协议的软件、服务基本上可以做到无缝迁移。

详细的接口兼容性情况见下表，供有需求的用户参考。

	Desciption	Name	Amazon	Kingsoft
Service Operation	获取所有bucket信息	GET Service	√	√
Bucket Operation	Bucket基本操作	DELETE Bucket	√	√
		GET Bucket (ListObjects)	√	√
		HEAD Bucket	√	√
		PUT Bucket	√	√
	Bucket cors相关操作	DELETE Bucket cors	√	√
		GET Bucket cors	√	√
		PUT Bucket cors	√	√
	Bucket lifecycle相关操作	DELETE Bucket lifecycle	√	部分兼容，除碎片相关外
		GET Bucket lifecycle	√	
		PUT Bucket lifecycle	√	
	Bucket policy相关操作	DELETE Bucket policy	√	部分兼容，除condition相关字段外
		GET Bucket policy	√	
		PUT Bucket policy	√	
	Bucket tagging相关操作	DELETE Bucket tagging	√	×
		GET Bucket tagging	√	
		PUT Bucket tagging	√	
	Bucket website相关操作	DELETE Bucket website	√	×
		GET Bucket website	√	
		PUT Bucket website	√	
	Bucket logging相关操作	GET Bucket logging	√	√
		PUT Bucket logging	√	√
	Bucket notification相关操作	GET Bucket notification	√	×
		PUT Bucket notification	√	
	Bucket versioning相关操作	GET Bucket versioning	√	×
		GET Bucket Object versions	√	
		PUT Bucket versioning	√	
	Bucket acl相关操作	PUT Bucket acl	√	√
		GET Bucket acl	√	√

	Bucket requestPayment相关操作	GET Bucket requestPayment	√	×
		PUT Bucket requestPayment	√	
	枚举该Bucket下的所有分块上传	List Multi Part Uploads	√	√
	删除Object	DELETE Object	√	√
	删除多个Object	Delete Multiple Objects	√	×
	下载Object	GET Object	√	√
	获取Object ACL	GET Object ACL	√	√
	获取Object BT 种子	GET Object torrent	√	×
	获取Object 元信息	HEAD Object	√	√
	Object对HTML5浏览器跨域支持	OPTIONS Object	√	×
	浏览器表单上传Object	POST Object	√	√
	Amazon Glacier存储恢复	POST Object restore	√	×
	上传Object	PUT Object	√	√
Object Operation	设置Object ACL	PUT Object acl	√	√
	复制Object	PUT Object - Copy	√	√
	分块上传相关操作	Initiate Multipart Upload	√	√
		Upload Part	√	√
		Upload Part - Copy	√	×
		Complete Multipart Upload	√	√
		Abort Multipart Upload	√	√
		List Parts	√	√
	对象标签相关操作	Delete Object Tagging	√	√
		Get Object Tagging	√	√
		Put Object Tagging	√	√
Image Thumbnail			×	√

加密

KS3服务端加密使用指南

目录

1. 概述

服务器端加密关乎静态数据加密，即 KS3 在将您的数据写入数据中心内的磁盘时会在对象级别上加密这些数据，并在您访问这些数据时为您解密这些数据。只要您验证了您的请求并且拥有访问权限，您访问加密和未加密数据元的方式就没有区别。例如，如果您使用预签名的 URL 来共享您的对象，那么对于加密和解密对象，该 URL 的工作方式是相同的。

2. 使用方式

KS3目前支持两种方式管理加密密钥：

- 使用具有 KS3 托管密钥的服务器端加密 (SSE-S3) - 利用多因素强加密来使用不同密钥加密每个对象。作为额外的保护，它将使用定期轮换的主密钥加密密钥本身。KS3 服务器端加密使用256 位高级加密标准 (AES-256)来加密您的数据。
- 使用具有客户提供的密钥的服务器端加密 (SSE-C) - 您管理数据的加密/解密、加密密钥和相关工具。

3. API支持

3.1 使用具有 KS3 托管密钥的服务器端加密 (SSE-S3)

- PUT 操作
- POST 操作
- Initiate Multipart Upload 操作
- Upload Part 操作
- Complete Multipart Upload 操作
- COPY 操作
- GET 操作
- HEAD 操作

PUT 操作

请求头

名称	描述
x-kss-server-side-encryption	如果请求中包含此头，服务端将对数据进行加密处理，合法值：AES256

响应头

名称	描述
x-kss-server-side-encryption	如果请求头里包含该值，则响应头将包含该值

错误返回

类型	描述
x-kss-server-side -encryption	如果x-kss-server-side-encryption请求头的值不是AES256

POST 操作

表单项

如果用户需要服务器使用默认加密，需要以下表单项

名称	描述
x-kss-server-side-encryption	如果存储 object 时使用了服务端加密，则需要配置该表单项，值为使用的加密算法，目前支持AES256。 类型：String

错误返回

类型	描述
加密类型无效	报400错误，x-kss-server-side-encryption表单项的值不是AES256

Initiate Multipart Upload 操作

请求头

名称	描述
x-kss-server-side-encryption	如果请求中包含此头，服务端将生成加密密钥，并在分块上传时使用该密钥进行加密，合法值：AES256

响应头

名称	描述
x-kss-server-side-encryption	* 如果请求头里包含该值，则响应头将包含该值

错误返回

类型	描述
加密类型无效	报400错误，x-kss-server-side-encryption请求头的值不是AES256
加密类型无效	报400错误，x-kss-server-side-encryption请求头为空

Upload Part 操作

响应头

名称	描述
x-kss-server-side-encryption	如果数据通过KS3 托管密钥的服务器端加密，则响应头将包含该值

错误返回

类型	描述
非法参数	如果提供x-kss-server-side-encryption请求头，将报400错误，提示参数非法

Complete Multipart Upload 操作

响应头

名称	描述
x-kss-server-side-encryption	如果数据通过KS3 托管密钥的服务器端加密，则响应头将包含该值

错误返回

类型	描述
非法参数	如果提供x-kss-server-side-encryption请求头，将报400错误，提示参数非法

COPY 操作

请求头

名称	描述
x-kss-server-side-encryption	合法值：AES256
x-kss-copy-source-server-side-encryption-customer-algorithm	如果被拷贝对象为客户端提供密钥加密方式，需提供该请求头
x-kss-copy-source-server-side-encryption-customer-key	如果被拷贝对象为客户端提供密钥加密方式，需提供该请求头
x-kss-copy-source-server-side-encryption-customer-key-MD5	如果被拷贝对象为客户端提供密钥加密方式，需提供该请求头

注：x-kss-copy-source-server-side-encryption-customer-algorithm 、x-kss-copy-source-server-side-encryption-customer-key 、x-kss-copy-source-server-side-encryption-customer-key-MD5此三个请求头必须同时存在

响应头

名称	描述
x-kss-server-side-encryption	如果请求头里包含该值，则响应头将包含该值

错误返回

类型	描述
加密类型无效	400错误，x-kss-server-side-encryption请求头的值不是AES256
MD5值错误	400错误，x-kss-copy-source-server-side-encryption-customer-key-MD5 不是 x-kss-copy-source-server-side-encryption-customer-key的MD5值
加密算法错误	400错误，x-kss-copy-source-server-side-encryption-customer-algorithm不是合法的AES256
非法参数	400错误，x-kss-copy-source-server-side-encryption-customer-algorithm 、x-kss-copy-source-server-side-encryption-customer-key 、x-kss-copy-source-server-side-encryption-customer-key-MD5 3个请求头未同时存在

GET 操作

响应头

名称	描述
x-kss-server-side-encryption	如果数据通过KS3 托管密钥的服务器端加密，则响应头将包含该值

错误返回

类型	描述
非法参数	如果提供x-kss-server-side-encryption请求头，将报400错误，提示参数非法

HEAD 操作

响应头

名称	描述
x-kss-server-side-encryption	如果数据通过KS3 托管密钥的服务器端加密，则响应头将包含该值

错误返回

类型	描述
非法参数	如果提供x-kss-server-side-encryption请求头，将报400错误，提示参数非法

3.2 通过使用客户提供的加密密钥的服务器端加密 (SSE-C) 保护数据

PUT 操作

请求头

名称	描述
x-kss-server-side-encryption-customer-algorithm	客户端提供的加密算法，合法值：AES256
x-kss-server-side-encryption-customer-key	客户端提供的加密密钥
x-kss-server-side-encryption-customer-key-MD5	客户端提供的通过BASE64编码的通过128位MD5加密的密钥的MD5值

注：此三个请求头必须同时存在

响应头

名称	描述
x-kss-server-side-encryption-customer-algorithm	如果请求头里包含该值，则响应头将包含该值
x-kss-server-side-encryption-customer-key-MD5	如果请求头里包含该值，则响应头将包含该值

错误返回

类型	描述
MD5值错误	400错误，xx-kss-server-side-encryption-customer-key-MD5 不是 x-kss-server-side-encryption-customer-key 的MD5值
加密算法错误	400错误，x-kss-server-side-encryption-customer-algorithm不是合法的AES256
非法参数	400错误，上述3个请求头未同时存在

POST 操作

请求头

名称	描述
x-kss-server-side-encryption-customer-algorithm	客户端提供的加密算法，合法值：AES256
x-kss-server-side-encryption-customer-key	客户端提供的加密密钥
x-kss-server-side-encryption-customer-key-MD5	客户端提供的通过BASE64编码的通过128位MD5加密的密钥的MD5值

注：此三个请求头必须同时存在

响应头

名称	描述
x-kss-server-side -encryption -customer-algorithm	如果请求头里包含该值，则响应头将包含该值
x-kss-server-side-encryption-customer-key-MD5	如果请求头里包含该值，则响应头将包含该值

错误返回

类型	描述
MD5值错误	400错误，x-kss-server-side-encryption-customer-key-MD5 不是 x-kss-server-side-encryption-customer-key 的MD5值
加密算法错误	400错误，x-kss-server-side -encryption -customer-algorithm 不是合法的AES256
非法参数	400错误，上述3个请求头未同时存在

Initiate Multipart Upload 操作

请求头

名称	描述

x-kss-server-side-encryption-customer-algorithm	客户端提供的加密算法，合法值：AES256
x-kss-server-side-encryption-customer-key	客户端提供的加密密钥
x-kss-server-side-encryption-customer-key-MD5	客户端提供的通过BASE64编码的通过128位MD5加密的密钥的MD5值

注：此三个请求头必须同时存在

响应头

名称	描述
x-kss-server-side -encryption -customer-algorithm	如果请求头里包含该值，则响应头将包含该值
x-kss-server-side-encryption-customer-key-MD5	如果请求头里包含该值，则响应头将包含该值

错误返回

类型	描述
MD5值错误	400错误，x-kss-server-side-encryption-customer-key-MD5 不是 x-kss-server-side-encryption-customer-key 的MD5值
加密算法错误	400错误，x-kss-server-side -encryption -customer-algorithm不是合法的AES256
非法参数	400错误，上述3个请求头未同时存在

Upload Part 操作

请求头

名称	描述
x-kss-server-side -encryption -customer-algorithm	客户端提供的加密算法，合法值：AES256
x-kss-server-side-encryption-customer-key	客户端提供的加密密钥
x-kss-server-side-encryption-customer-key-MD5	客户端提供的通过BASE64编码的通过128位MD5加密的密钥的MD5值

注：此三个请求头必须同时存在

响应头

名称	描述
x-kss-server-side -encryption -customer-algorithm	如果请求头里包含该值，则响应头将包含该值
x-kss-server-side-encryption-customer-key-MD5	如果请求头里包含该值，则响应头将包含该值

错误返回

类型	描述
MD5值错误	400错误，x-kss-server-side-encryption-customer-key-MD5 不是 x-kss-server-side-encryption-customer-key 的MD5值
加密算法错误	400错误，x-kss-server-side-encryption-customer-algorithm不是合法的AES256
非法参数	400错误，上述3个请求头未同时存在
缺少客户密钥	400错误，数据通过客户端提供密钥加密，但是请求中未提供客户密钥
MD5值不一致	400错误，获取数据时提供的密钥的MD5和存储时不一致

Complete Multipart Upload 操作

响应头

名称	描述
x-kss-server-side-encryption-customer-algorithm	如果请求头里包含该值，则响应头将包含该值
x-kss-server-side-encryption-customer-key-MD5	如果请求头里包含该值，则响应头将包含该值

错误返回

类型	描述
非法参数	400错误，不应该携带客户端提供密钥进行加密的请求头，如果携带，提示参数非法

COPY 操作

请求头

名称	描述
xx-kss-server-side-encryption-customer-algorithm	客户端提供的加密算法，合法值：AES256
x-kss-server-side-encryption-customer-key	客户端提供的加密密钥
x-kss-server-side-encryption-customer-key-MD5	客户端提供的通过BASE64编码的通过128位MD5加密的密钥的MD5值
x-kss-copy-source-server-side-encryption-customer-algorithm	如果被拷贝对象为客户端提供密钥加密方式，需提供该请求头
x-kss-copy-source-server-side-encryption-customer-key	如果被拷贝对象为客户端提供密钥加密方式，需提供该请求头
x-kss-copy-source-server-side-encryption-customer-key-MD5	如果被拷贝对象为客户端提供密钥加密方式，需提供该请求头

注： x-kss-server...三个请求头以及x-kss-copy-source...三个请求头必须同时存在

响应头

名称	描述
x-kss-server-side-encryption-customer-algorithm	如果请求头里包含该值，则响应头将包含该值
x-kss-server-side-encryption-customer-key-MD5	如果请求头里包含该值，则响应头将包含该值

错误返回

类型	描述
MD5值错误	400错误，x-kss(-copy-source)-server-...-MD5 不是 x-kss(-copy-source)-server-...-key 的MD5值
加密算法错误	400错误，x-kss(-copy-source)-server-side -...-algorithm不是合法的AES256
非法参数	400错误，上述3个请求头未同时存在
缺少客户密钥	400错误，数据通过客户端提供密钥加密，但是请求中未提供客户密钥
MD5值不一致	400错误，获取数据时提供的密钥的MD5和存储时不一致

GET 操作

请求头

名称	描述

x-kss-server-side-encryption-customer-algorithm	客户端提供的加密算法，合法值：AES256
x-kss-server-side-encryption-customer-key	客户端提供的加密密钥
x-kss-server-side-encryption-customer-key-MD5	客户端提供的通过BASE64编码的通过128位MD5加密的密钥的MD5值

注：此三个请求头必须同时存在

响应头

名称	描述
x-kss-server-side-encryption-customer-algorithm	如果请求数据通过客户提供的加密密钥的服务器端加密，则响应头将包含该值
x-kss-server-side-encryption-customer-key-MD5	如果请求数据通过客户提供的加密密钥的服务器端加密，则响应头将包含该值

错误返回

类型	描述
MD5值错误	400错误，x-kss-server-...-MD5 不是 x-kss-server-...-key 的MD5值
加密算法错误	400错误，x-kss-server-side -...-algorithm不是合法的AES256
非法参数	400错误，上述3个请求头未同时存在
缺少客户密钥	400错误，数据通过客户端提供密钥加密，但是请求中未提供客户密钥
MD5值不一致	400错误，获取数据时提供的密钥的MD5和存储时不一致

HEAD 操作

请求头

名称	描述
x-kss-server-side-encryption-customer-algorithm	客户端提供的加密算法，合法值：AES256
x-kss-server-side-encryption-customer-key	客户端提供的加密密钥
x-kss-server-side-encryption-customer-key-MD5	客户端提供的通过BASE64编码的通过128位MD5加密的密钥的MD5值

注：此三个请求头必须同时存在

响应头

名称	描述
x-kss-server-side-encryption-customer-algorithm	如果请求数据通过客户提供的加密密钥的服务器端加密，则响应头将包含该值
x-kss-server-side-encryption-customer-key-MD5	如果请求数据通过客户提供的加密密钥的服务器端加密，则响应头将包含该值

错误返回

类型	描述
MD5值错误	400错误，x-kss-server-...-MD5 不是 x-kss-server-...-key 的MD5值
加密算法错误	400错误，x-kss-server-side -...-algorithm不是合法的AES256
非法参数	400错误，上述3个请求头未同时存在
缺少客户密钥	400错误，数据通过客户端提供密钥加密，但是请求中未提供客户密钥
MD5值不一致	400错误，获取数据时提供的密钥的MD5和存储时不一致

KS3计量

KS3为用户提供计量API，方便用户查询KS3的使用情况，包括容量、流量、请求次数、带宽、数据取回量、对象标签数等。

访问方式

计量API访问地址为：ks3bill.api.ksyun.com，支持HTTP和HTTPS访问。 主账号和具有对象存储计量权限的子用户，可以访问计量API。计量API支持GET方法发送请求。

公共请求参数

参数名称	类型	必填	描述
Service	String	是	服务名称，固定值：ks3bill
Action	String	是	操作接口名，固定值：QueryKs3Data
Version	String	是	接口版本号，固定值：v1
Signature	String	是	计量API使用V4签名方式，具体参见签名机制

请求参数

参数名称	类型	必填	描述
StartTime	String	是	查询用量开始时间：yyyyMMddHHmm 例如：202111230000
EndTime	String	是	查询用量结束时间（与开始时间同月，不支持跨月查询）：yyyyMMddHHmm 例如：202111240000
DateType	String	是	支持按天粒度查询，固定值：Day
Bucketname	String	是	存储空间名称，最多支持同时查询5个存储桶的用量明细，以逗号分隔

Ks3Product	String	否	可以查询单个计费项，也可查询多个计费项，以逗号分隔，如果不填，则查询除带宽外的所有计费项 存储量：DataSize 外网流量：NetworkFlow CDN回源流量：CDNFlow 跨区域复制流量：ReplicationFlow GET类请求次数：RequestsGet PUT类请求次数：RequestsPut 数据取回量：RestoreSize 对象标签梳理：TagNum 下行带宽（不区分外网和CDN）：BandwidthDown 外网下行带宽：NetBandwidthDown CDN下行带宽：CDNBandwidthDown

响应参数

参数名称	描述	类型
Data	包含一个或多个bucket的容器 子节点：Buckets	container
Buckets	包含bucket信息的容器 子节点：Name,Type,DataSize,PutRequest,GetRequest,NetworkFlow,CDNFlow,ReplicationFlow,StandardIAData,ArchiveData,TagNum 父节点：Data	container
Name	Bucket的名称 父节点：Data.Buckets	String
StartTime	数据开始时间 父节点：Data.Buckets	String
EndTime	数据结束时间 父节点：Data.Buckets	String
StandardDataSize	Bucket的标准存储量，单位是Bytes 父节点：Data.Buckets	String
StandardIADataSize	Bucket的低频存储量，单位是Bytes 父节点：Data.Buckets	String
ArchiveDataSize	Bucket的归档存储量，单位是Bytes 父节点：Data.Buckets	String
StandardPutRequest	标准存储的PUT请求次数 父节点：Data.Buckets	String
StandardIAPutRequest	低频存储的PUT请求次数 父节点：Data.Buckets	String
ArchivePutRequest	归档存储的PUT请求次数 父节点：Data.Buckets	String
StandardGetRequest	标准存储的GET请求次数 父节点：Data.Buckets	String
StandardIAGetRequest	低频存储的GET请求次数 父节点：Data.Buckets	String
ArchiveGetRequest	归档存储的GET请求次数 父节点：Data.Buckets	String
NetworkFlow	外网下行流量，单位是Bytes 父节点：Data.Buckets	String
CDNFlow	CDN回源流量，单位是Bytes 父节点：Data.Buckets	String
ReplicationFlow	跨区域复制流量，单位是Bytes 父节点：Data.Buckets	String
BandwidthDown	外网下行带宽，不区分外网下行和CDN下行，单位是bps 父节点：Data.Buckets	String
OuterBandwidthDown	外网下行带宽，单位是bps 父节点：Data.Buckets	String
CDNBandwidthDown	CDN下行带宽，单位是bps 父节点：Data.Buckets	String
StandardIAData	低频存储数据取回量，单位是Bytes 父节点：Data.Buckets	String
ArchiveData	归档存储解冻数据量，单位是Bytes 父节点：Data.Buckets	String
TagNum	对象标签的数量 父节点：Data.Buckets	String
RequestId	由KS3指定的唯一值，可用于解决KS3出现的问题	String

请求示例

```
GET /?Action=QueryKs3Data&StartTime=202111220000&EndTime=202111222300&DateType=Day&Bucketname=sh-cm5&Ks3Product=DataSize,NetworkFlow,CDNFlow,ReplicationFlow,RequestsGet,RequestsPut,RestoreSize,TagNum,BandwidthDown,NetBandwidthDown,CDNBandwidthDown&Service=ks3bill&Version=v1 HTTP/1.1
Host: ks3bill.api.ksyun.com
X-Amz-Date:  20211124T070940Z
Authorization: {SignatureValue}
```

响应示例


```
{
    "Code": "OK",
    "Message": "OK",
    "Data": {
        "Buckets": [
            {
                "Name": "sh-cm5",
                "StartTime": "2021-11-22 00:00:00",
                "EndTime": "2021-11-22 23:59:59",
                "StandardDataSize": "1000",
                "StandardIADataSize": "2000",
                "ArchiveDataSize": "3000",
                "StandardPutRequest": "10",
                "StandardIAPutRequest": "20",
                "ArchivePutRequest": "30",
                "StandardGetRequest": "40",
                "StandardIAGetRequest": "50",
```

```
                "ArchiveGetRequest": "60",
                "NetworkFlow": "7000",
                "CDNFlow": "8000",
                "BandwidthDown": [
                    {
                        "2021-11-22 08:00:00": "500"
                    }
                ],
                "CDNBandwidthDown": [
                    {
                    }
                ],
                "OuterBandwidthDown": [
                    {
                        "2021-11-22 08:00:00": "600"
                    }
                ],
                "ReplicationFlow": "9000",
                "StandardIAData": "1000",
                "ArchiveData": "2000",
                "TagNum": "10"
            }
        ]
    },
    "RequestId": "5d52bb03-e25c-4c8f-baa8-2b1c569058b4"
}
```


配置权限

主账号或者具有权限的子用户可以调用计量API，查询计量相关信息。

子用户配置权限的方法如下：

- 登录金山云控制台首页，点击右上角用户名称下方的**身份与控制**。
- 为子用户添加策略。点击左侧**访问控制-人员管理-子用户**，选择要授权的子用户，点击**添加权限**，将系统策略“KS3BillFullAccess“授权给子用户。

错误说明

错误码	错误信息	状态码	描述
SignatureDoesNotMatch	The Signature is not matched.	400	签名不匹配
InvalidArgument	Invalid length of StartTime or EndTime.	400	无效的日期格式
InvalidArgument	Invalid StartTime.	400	StartTime不合法
InvalidArgument	Invalid EndTime.	400	EndTime不合法
InvalidArgument	StartTime and EndTime should be in the same month.	400	StartTime和EndTime应在同一个月
InvalidArgument	The EndTime should be later than the StartTime.	400	结束时间应晚于开始时间
InvalidArgument	The number of specified buckets can not exceed 5.	400	最多支持查询5个桶的用量明细
InvalidArgument	Invalid DateType.	400	DateType不合法
InvalidArgument	Invalid Ks3Product:< Ks3Product >.	400	Ks3Product不合法
InvalidAccessKey	Invalid Accesskey.	400	无效的AccessKey

错误码

message	HTTP Status	描述
AccessDenied	403	拒绝访问
BadDigest	400	MD5值错误
BucketAlreadyExists	409	Bucket已经存在
BucketAlreadyOwnedByYou	409	用户已经是Bucket的拥有者
BucketNotEmpty	409	Bucket不为空
InternalError	500	内部错误
InvalidAccessKey	403	无效的AccessKey
InvalidACLString	400	ACL配置无效
InvalidAuthorizationString	400	无效的验证字符串
InvalidBucketName	400	无效的Bucket名称
InvalidDateFormat	400	无效的日期格式
InvalidDigest	400	无效的MD5值
InvalidEncryptionAlgorithm	400	无效的指定加密算法
InvalidHostHeader	400	无效的头信息
InvalidParameter	400	无效的参数
InvalidPath	400	无效的路径
InvalidQueryString	400	无效的请求字符串
InvalidRange	416	无效的range
KeyTooLong	400	Key太长
MetadataTooLarge	400	metadata过大
MethodNotAllowed	405	不支持的方法
MissingDateHeader	400	头信息中缺少data
MissingHostHeader	400	头信息中缺少host
NoSuchBucket	404	该Bucket不存在
NoSuchKey	404	该Key不存在
NotImplemented	501	无法处理的方法
RequestTimeTooSkewed	403	发起请求的时间和服务器时间超出15分钟
SignatureDoesNotMatch	403	签名不匹配
TooManyBuckets	400	用户的Bucket数目超过限制
URLExpired	403	url过期
BadParams	400	参数错误
ImageTypeNotSupport	400	图片类型不支持
MissingFormArgs	400	没有上传Policy
ContentRangeError	400	Range错误
ContentLengthOutOfRange	400	上传文件内容大于range
PolicyError	400	Policy错误
ExpirationError	400	Policy中没有expiration
FormUnmatchPolicy	400	表单中的内容和policy不匹配
RequestEntityTooLarge	413	上传文件的大小不符合限制
Request Timeout	408	服务器等候请求时发生超时
Precondition Failed	412	调用GetObject接口时， object 的 ETag(entity tag)与指定值不一致