**存储Proxy详细设计文档**

by 杨婷

# Proxy配置信息

## 提供对外(S3客户端)访问API接口

<https://host:port/storageproxy/v1/>........

## 基本配置

* nginx.conf中配置include storageproxy.conf
* storageproxy.conf配置业务location

location /storageproxy/v1 {

lua\_need\_request\_body on;

default\_type text/plain;

content\_by\_lua\_file $PATH/nginx/slp/storageproxy/storageproxy.lua;

}

# Proxy业务处理流程(主业务storeproxy.lua)

## 进入鉴权流程（storeproxy\_access.lua）

* **传入参数：**

http请求的method, uri, args, headers, body

* **处理流程：**
* 根据传入参数，解析并保存相关参数。
* 鉴权判断

1. 获取Authorization头

* 若没有该头，则进入2）处理；
* 有该消息头，且格式匹配Authorization: AWS {AWSAccessKeyID}:{ signature }，则协议为V2版本的S3协议，具体对请求进行身份验证的过程参考[AWS 签名版本 2](#_AWS_签名版本_2)（**签名在Authorization 消息头中部分**）。
* 有该消息头，且格式匹配Authorization: AWS4-HMAC-SHA256 Credential= AWSAccessKeyID/credential scope, SignedHeaders=SignedHeaders, Signature=signature，则协议为V4版本的S3协议，具体对请求进行身份验证的过程参考[AWS 签名版本 4](#_AWS_签名版本_4)（**签名在Authorization 消息头中部分**）。

1. 获取请求参数字段

* 在参数字段中，查找“AWSAccessKeyId=”和“Signature=”字段，若存在，则协议为V2版本的S3协议，具体对请求进行身份验证的过程参考[AWS 签名版本 2](#_AWS_签名版本_2)（**签名在查询字符串中部分**）。
* 在参数字段中，查找字段“X-Amz-Algorithm=AWS4-HMAC-SHA256”，若存在，则协议为V4版本的S3协议，具体对请求进行身份验证的过程参考[AWS 签名版本 4](#_AWS_签名版本_4)（**签名在查询字符串中部分**）。
* 若没有参数字段或上述查询匹配不成功，则进入3）处理

1. S3身份验证字段不存在，当前请求不是S3协议接口，返回**异常A**（或做其他匹配X）。

* 若鉴权分析成功，则根据鉴权计算所得signature与请求signature进行比对，不相等则鉴权失败返回**异常C**，相等则鉴权通过返回**成功D**
* **返回值：**

http状态码，处理状态码

* 返回异常A：201，处理状态码“20000000”：表明“不是期望协议类型”
* 返回异常B：201，处理状态码“20000001”：表明“鉴权分析处理失败”
* 返回异常C：404，处理状态码“20000002”：表明“鉴权失败”
* 返回异常E：201，处理状态码“20000003”：表明“根据AWSAccessKeyId无法获取用户信息”
* 返回异常F：201，处理状态码“20000004”：表明“无法获取Date和x-amz-date信息”
* 返回异常F2：201，处理状态码“20000006”：表明“鉴权失败，当前请求时间超过S3要求”
* 返回成功D：200，处理状态码“00000000”：表明“处理成功”
* 返回异常G：201，处理状态码“20000005”：表明“AWS4中Date/x-amz-date域值与Credential /X-Amz-Credential中date域不匹配”
* H， 20000007，requestbody解析失败
* 预留X
* **经过鉴权模块处理，解码处理得：**

\_M.body

\_M.header

\_M.userinfo

\_M.args

## 业务流程处理（由主业务处理）

若鉴权模块返回200，则继续业务流程处理。返回其他值，根据返回值向客户端回复响应。

根据Method、URL和body匹配，决定当前待处理接口的类型，并进行相应的处理。

### Create bucket

为一个用户uid创建一个名为bucketname的bucket

* PUT bucket 有几种要求的创建格式，如下：

1). 无任何限定

PUT / HTTP/1.1

Host: colorpictures.s3.amazonaws.com

Content-Length: 0

Date: Wed, 01 Mar 2006 12:00:00 GMT

Authorization: *authorization string*

*--------------------------------------------------*

HTTP/1.1 200 OK

x-amz-id-2: YgIPIfBiKa2bj0KMg95r/0zo3emzU4dzsD4rcKCHQUAdQkf3ShJTOOpXUueF6QKo

x-amz-request-id: 236A8905248E5A01

Date: Wed, 01 Mar 2006 12:00:00 GMT

Location: /colorpictures

Content-Length: 0

Connection: close

Server: AmazonS3

*--------------------------------------------------*

2) 本期无需实现不再整理

3) 本期无需实现不再整理

* S3匹配要点：

Method：PUT

URL：/

* 操作要点

向Hbase中名为bucket表中，新增一条数据；

* 预处理
* **Header:**

读取host头的首字段作为bucketname;

读取Content-Length，若为0不限制bucket info，否则需要读取body;

判断是否有消息头“x-amz-acl”或有以“x-amz-grant-”开头的消息头；没有则按A读取body，否则按B或C读取body

* **Body：**

1. 读取body中CreateBucketConfiguration下LocationConstraint字段的value
2. 读取body中LocationConstraint字段值
3. 读取body中LocationConstraint字段值
4. 不解析body

* **其他：**

1. 构造uid\_bucketname；
2. bucket\_info组合（quota[max\_files,max\_size]，stat[cur\_files,cur\_size]）；

* 向Hbase插入数据

利用Hbase的PUT接口模板（[hbase\_put.lua](#_hbase_put)），向Hbase发起请求，接收响应(状态码、header\_json、encode(body\_json))

* 根据上述接口的执行结果，向客户端回复响应，与S3交互接口参数请参考《存储proxy接口文档》
* 分别按照解析body的类型，进行响应的回复。



* 若接收到的接口返回的响应码为200，则对header和body进行解码、校验等操作后，按照S3协议要求，返回响应的响应；
* 状态码不为200，或解码校验失败，按照S3协议要求，返回异常响应

### List bucket

列出一个用户uid下所有的bucket（bucketname+bucketinfo）

* S3匹配要点：

Method：GET；

URL：/；

* 操作要点

获取Hbase中名为bucket表中，以uid为索引的所有数据；

* 预处理

无

* 从Hbase读取数据

利用Hbase的GET接口模板（[hbase\_get.lua](#_hbase_get)），向Hbase发起请求，接收响应(状态码、header\_json、encode(body.json))

* 根据上述接口执行结果，向客户端回复响应，与S3交互接口参数请参考《存储proxy接口文档》
* 若接收到的接口返回的响应码为200，则对header和body进行解码、校验等操作后，按照S3协议要求，返回响应的响应；
* 状态码不为200，或解码校验失败，按照S3协议要求，返回异常响应

## Hbase流程控制

由于proxy用lua语言开发，且Lua对json编码支持较好，因此向hbase发送http请求时，请增加消息头Accept: application/json。

### hbase\_get

传入参数：1（目的URL），2（目的body\_json格式）

处理流程：读取hbase的地址；将传入参数作为url；使用GET方法；增加消息头；

编码body并设置；

接收请求：

若状态为nil，则报错返回httpcode,nil,nil;

否则返回httpcode, http.header, header.body

### hbase\_put

传入参数：1（目的URL），2（目的body\_json格式），3（待增加的header）

处理流程：读取hbase的地址；将传入参数作为url；使用PUT方法；增加消息头；

编码body并设置；

接收请求：

若状态为nil，则报错返回httpcode,nil,nil;

否则返回httpcode, http.header, header.body

### hbase\_conf.lua

目前设置hbase的地址；

**——后期优化可以考虑采用对redis的访问，访问hbase。**

## Ceph流程控制

本期不对object对象进行处理，暂无涉及流程。

## AWS\_ SecretAccessKey获取流程（storeproxy\_getsecret.lua）

--后期是不是考虑先去redis里读取数据？

* **传入参数：**

AWSAccessKeyId

* **处理流程：**
* 利用Hbase的GET接口模板（[hbase\_get.lua](#_hbase_get)），向Hbase发起请求，接收响应；
* 若Hbase回送的响应码为200，则解析body，获得响应内容：

uid, secretkey(计算所用AWS\_SecretAccessKey)，quota[max\_files,max\_size]，stat[cur\_files,cur\_size]

* **返回值**

返回Hbase回送的响应码及对消息体解析后的json串（或异常返回nil）；

# 附录

## 对请求进行身份验证

### AWS 签名版本 2

由于S3的签名信息，可能存在于Authorization 的 HTTP 消息头，也可能作为查询字符串存在于请求URL参数种，因此鉴权分析需要分为两种处理。

#### 示例：

示例一：

Authorization: AWS AKIAIOSFODNN7EXAMPLE:frJIUN8DYpKDtOLCwo//yllqDzg=

示例二：

https://photos/puppy.jpg?AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE&Expires=1141889120&Signature=vjbyPxybdZaNmGa%2ByT272YEAiv4%3D

#### 签名在Authorization 消息头中

* 签名计算方法构成

**Authorization = "AWS" + " " + AWSAccessKeyId + ":" + Signature;**

**Signature** = Base64( HMAC-SHA1(**AWS­\_SecretAccessKey**, UTF-8-Encoding-Of( **StringToSign** ) ) );

**StringToSign** = HTTP-Verb + "\n" +

    Content-MD5 + "\n" +

    Content-Type + "\n" +

    Date + "\n" +

**CanonicalizedAmzHeaders** +

**CanonicalizedResource**;

**CanonicalizedResource** = [ "/" + Bucket ] +

    <HTTP-Request-URI, from the protocol name up to the query string> +

[ sub-resource, if present. For example "?acl", "?location", "?logging", or "?torrent"];

**CanonicalizedAmzHeaders** = <参考[CanonicalizedAmzHeaders计算方法](#CanonicalizedAmzHeaders计算方法)>

* 按照上述计算方法，对请求进行校验的步骤如下：
  1. **预处理：**
     1. 对于通过Authorization头进行身份校验的请求，其随附的客户端时间戳必须处于收到请求时的 Amazon S3 系统时间的 15 分钟之内，因此在开始校验前需要对请求时间进行校验，校验失败**返回异常F2；**
     2. 由于请求的时间可能分别由Date和x-amz-date展现，且以x-amz-date优先。因此首先读取x-amz-date头，有则作为后期处理的时间；没有则读取Date头；均没有直接**返回异常F**。

注意：当前V2版本采用x-amz-date 标头的值必须采用 RFC 2616 格式 (http://www.ietf.org/rfc/rfc2616.txt) 中的任意格式。

* 1. 获取Authorization头中的AWSAccessKeyId和Signature；
  2. 根据AWSAccessKeyId 获取AWS­\_SecretAccessKey，获取流程参考[AWS\_ SecretAccessKey获取流程](#_AWS__SecretAccessKey获取流程_1)；

（注意，与此同时可以获取到**用户的userinfo**，获取失败**返回异常E**）

* 1. 计算CanonicalizedResource；

（注意：由于S3协议提供bucket虚拟托管方案，因此计算时需要先判断bucketname所在位置。需要判断host头，如果为Host: s3.amazonaws.com，则bucketname在URL中，否则为该头值域的第一个字段。）

* 1. 计算CanonicalizedAmzHeaders；
  2. 根据HTTP的Method（表明），HTTP消息头Content-MD5、Content- Type，预处理中计算的Date的值，上述获得的CanonicalizedAmzHeaders 和CanonicalizedResource，按格式计算获取StringToSign

（注意： 对于Content-Type 或 Content-MD5 对于 PUT 请求是可选的，并且对于 GET 请求没有任何意义，因此计算时用””代替）

* 1. 根据AWS­\_SecretAccessKey和StringToSign，计算出一个signature；

（注意，上述计算时，若出现异常**返回异常B**）

* 1. 对计算出的signature与从Authorization获取的signature进行比较，相同则身份验证通过，**返回成功D**；否则验证失败，**返回异常C**。

#### 签名在查询字符串中

* 签名计算方法构成

Signature = URL-Encode( Base64( HMAC-SHA1(**AWSAccessKeyId**, UTF-8-Encoding-Of( **StringToSign** ) ) ) );

**StringToSign** = HTTP-VERB + "\n" +

Content-MD5 + "\n" +

Content-Type + "\n" +

**Expires + "\n" +**

CanonicalizedAmzHeaders +

CanonicalizedResource;

* 按照上述计算方法，对请求进行校验的步骤参考对比“签名在Authorization消息头中”算法，变更如下：
  1. **预处理：**
     1. 无需再判断、计算Date和x-amz-date，直接获取消息参数中“Expires”字段的值，并赋值给Date
  2. 获取消息参数中的AWSAccessKeyId和Signature字段的值；

其余计算参考[签名在Authorization 消息头中](#_签名在Authorization_消息头中)对请求进行校验的步骤的c)-h)阶段

### AWS 签名版本 4

#### 示例：

示例一：

Authorization: AWS4-HMAC-SHA256 Credential=AKIDEXAMPLE/20150830/us-east-1/iam/aws4\_request, SignedHeaders=content-type;host;x-amz-date, Signature=5d672d79c15b13162d9279b0855cfba6789a8edb4c82c400e06b5924a6f2b5d7

示例二：

https://iam.amazonaws.com?Action=ListUsers&Version=2010-05-08&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIDEXAMPLE%2F20150830%2Fus-east-1%2Fiam%2Faws4\_request&X-Amz-Date=20150830T123600Z&X-Amz-Expires=60&X-Amz-SignedHeaders=content-type%3Bhost&X-Amz-Signature=37ac2f4fde00b0ac9bd9eadeb459b1bbee224158d66e7ae5fcadb70b2d181d02

#### 签名在Authorization 消息头中

* 签名计算方法构成

**Authorization: AWS4-HMAC-SHA256 Credential=AWSAccessKeyId/credential scope, SignedHeaders=SignedHeaders, Signature=signature**

**signature** = HexEncode(*HMAC*(**Derived-Signing-Key**, **String­\_To\_Sign**))

**Derived-Signing-Key** = HMAC(***kService***, "aws4\_request")

**kService** = HMAC(***kRegion***, *Service*)

**kRegion** = HMAC(***kDate***, *Region*)

**kDate** = HMAC("AWS4" + ***kSecret***, *Date*)

**kSecret** = **AWS\_SecretAccessKey**

**String­\_To\_Sign** =

Algorithm + '\n' +

RequestDate + '\n' +

**CredentialScope** + '\n' +

*HexEncode*(*Hash*(**CanonicalRequest**))

**CanonicalRequest** =

HTTPRequestMethod + '\n' +

*CanonicalURI* + '\n' +

*CanonicalQueryString* + '\n' +

*CanonicalHeaders* + '\n' +

**SignedHeaders** + '\n' +

HexEncode(Hash(*RequestPayload*)

上述描述：

* **Hash** 表示生成消息摘要的函数，通常是 SHA-256。
* **HexEncode** 表示以小写字母形式返回摘要的 base-16 编码的函数。例如，HexEncode("m") 返回值 6d 而不是 6D。输入的每一个字节都必须表示为两个十六进制字符。
* **HMAC(key, data)** 表示以二进制格式返回输出的 HMAC-SHA256 函数
* **Date、Service、Region**均可从Authorization头中credential scope中获取。
* **CanonicalURI**（规范URI）是 URI 的绝对路径部分的 URI 编码版本，从 HTTP 主机到开始查询字符串参数（如果有）的问号字符（“?”）。

注意：如果绝对路径为空，则使用正斜杠 (/)

* **CanonicalQueryString**（规范查询字符串），构造请参考[V4版本CanonicalQueryString计算方法](#_V4版本CanonicalQueryString计算方法)

注意：如果请求不包括查询字符串，请使用空字符串（实际上是空白行）。

* **CanonicalHeaders**（规范标头），构造请参考[V4版本CanonicalHeaders计算方法](#_V4版本CanonicalHeaders计算方法)
* **RequestPayload**，是使用 SHA256 的哈希（摘要）函数以基于 HTTP 或 HTTPS 请求正文中的负载，创建以小写十六进制字符串形式表示的哈希值。

注意：如果负载为空，则使用空字符串作为哈希函数的输入。

* 按照上述计算方法，对请求进行校验的步骤如下：

1. **预处理：**
   * 1. 对于通过Authorization头进行身份校验的请求，S3 v4版本要求，请求必须在请求中的时间戳的 5 分钟内到达 AWS，因此在开始校验前需要对请求时间进行校验；
     2. 由于请求的时间可能分别由Date和x-amz-date展现，且以x-amz-date优先。因此首先读取x-amz-date头，有则作为后期处理的时间；没有则读取Date头；均没有直接**返回异常F**。
     3. 注意V4版本中对x-amz-date的格式要求是使用 ISO8601 基本格式 YYYYMMDD'T'HHMMSS'Z'。
2. 获取Authorization头中的Signature、credential scope, SignedHeaders，继而获取AWSAccessKeyId/Date/Rejion/ Service；

（注意1：credential scope包括五部分AWSAccessKeyId/Date/Rejion/ Service/aws4\_request；

注意2：该Date（格式为 YYYYMMDD）是否与预处理所得Date的前半部分相同，不同需要报错，返回异常G）

1. 根据AWSAccessKeyId 获取**AWS­\_SecretAccessKey**，获取流程参考[AWS\_ SecretAccessKey获取流程](#_AWS__SecretAccessKey获取流程_1)；

（注意：与此同时可以获取到**用户的userinfo**，获取失败**返回异常E**）

1. 根据HTTP的请求Method，计算所得CanonicalURI、CanonicalQueryString 、CanonicalHeaders参数，从Authorization获取已签名的标头SignedHeaders的值，计算所得的RequestPayload负载，按要求的格式组合为单个字符串，即为所求规范请求**CanonicalRequest；**
2. 计算待签字符串**String\_To\_Sign**；
3. 首先使用对负载进行哈希处理时所使用的相同算法（SHA-256）来创建规范请求的摘要，最终以小写十六进制字符字符串形式表示。
4. 其次，将Algorithm “AWS4-HMAC-SHA256”，预处理时获得的请求日期值RequestDate，从Authorization获得的凭证范围值CredentialScope，及i中计算所得的摘要，按要求格式组合成单个字符串即为所求。
5. 根据HTTP的Method，HTTP消息头Content-MD5、Content- Type，预处理中计算的Date的值，上述获得的CanonicalizedAmzHeaders 和CanonicalizedResource，按格式计算获取**StringToSign**

（注意： 对于Content-Type 或 Content-MD5 对于 PUT 请求是可选的，并且对于 GET 请求没有任何意义，因此计算时用””代替）

1. 按要求的格式计算派生的签名密钥**Derived-Signing-Key**。
2. 根据上述计算所得String\_To\_Sign和Derived-Signing-Key，计算出一个**signature**；

（注意：上述计算时，若出现异常**返回异常B**）

1. 对计算出的signature与从Authorization获取的signature进行比较，相同则身份验证通过，**返回成功D**；否则验证失败，**返回异常C**。

#### 签名在查询字符串中

* 签名计算方法构成

querystring = Action=action

+ &X-Amz-Algorithm= **AWS4-HMAC-SHA256**

+ &X-Amz-Credential= urlencode(**AWSAccessKeyId** + '/' + **credential\_scope**)

+ &X-Amz-Date=date

+ &X-Amz-Expires=timeout interval

+ &X-Amz-SignedHeaders=**signed\_headers**

+ &X-Amz-Signature=**signature**

* 按照上述计算方法，对请求进行校验的步骤参考对比“签名在Authorization消息头中”算法，变更如下：

1. **预处理：**
2. Date取消息参数中x-amz-date字段的值

（注意：仍要拿前半部分与credential\_scope字段中Date域进行比对）

1. 与V2版本比较， V4版本保留了x-amz-date字段，且以此作为Date时间，并X-Amz-Expires作为请求的操作的最短可行时间
2. Signature、credential scope, SignedHeaders从消息参数中的对应字段中获取

其余计算参考[签名在Authorization 消息头中](#_签名在Authorization_消息头中_1)对请求进行校验的步骤的c)-h)阶段

## 状态码列表

状态码“00000000”：表明“处理成功”

状态码“20000000”：表明“不是期望协议类型”

状态码“20000001”：表明“鉴权分析处理失败”

状态码“20000002”：表明“鉴权失败”

状态码“20000003”：表明“根据AWSAccessKeyId无法获取用户信息”

状态码“20000004”：表明“无法获取Date和x-amz-date信息”

状态码“20000005”：表明“AWS4中Date/x-amz-date域值与Credential /X-Amz-Credential中date域不匹配”

## V2版本CanonicalizedAmzHeaders计算方法（storeproxy\_connicalheader\_v2.lua）

* 规范化步骤

1. 选择所有以“x-amz-”开头的 HTTP 请求头，并将每个 HTTP头名称转换为小写。

例如，“X-Amz-Date”改为“x-amz-date”。

1. 根据标头名称，按字典顺序排列标头集。
2. 将相同名称的标头字段合并为一个“header-name:comma-separated-value-list”对，各值之间不留空格。

例如，可以将元数据标头“x-amz-meta-username: fred”和“x-amz-meta-username: barney”合并为单个标头“x-amz-meta-username: fred,barney”。

1. 通过将折叠空格（包括换行符）替换为单个空格，“展开”跨多个行的长标头
2. 删除标头中冒号周围的空格。

例如，标头“x-amz-meta-username: fred,barney”改为“x-amz-meta-username:fred,barney”。

1. 请向生成的列表中的每个标准化标头附加换行字符 (U+000A)并连接所有的标头规范化为单个字符串，形成最终 CanonicalizedResource 元素。

## V4版本CanonicalQueryString计算方法（storeproxy\_connicalquerystr\_v4.lua）

要构建规范查询字符串，请完成以下步骤：

1. 按字符代码以升序（ASCII 顺序）对参数名称进行排序。例如，以大写字母 F（ASCII 代码 70）开头的参数名称排在以小写字母 b（ASCII 代码 98）开头的参数名称之前。
2. 根据以下规则对每个参数名称和值进行 URI 编码：
   * 请勿对 [RFC 3986](http://tools.ietf.org/html/rfc3986) 定义的任何非预留字符进行 URI 编码，这些字符包括：A-Z、a-z、0-9、连字符 (-)、下划线 (\_)、句点 (.) 和波浪符 ( ~ )。
   * 使用 %XY 对所有其他字符进行百分比编码，其中“X”和“Y”为十六进制字符（0-9 和大写字母 A-F）。例如，空格字符必须编码为 %20（不像某些编码方案那样使用“+”），扩展 UTF-8 字符必须采用格式 %XY%ZA%BC。
3. 以排序后的列表中第一个参数名称开头，构造规范查询字符串。
4. 对于每个参数，追加 URI 编码的参数名称，后跟字符“=”（ASCII 代码 61），再接 URI 编码的参数值。对没有值的参数使用空字符串。
5. 在每个参数值后追加字符“&”（ASCII 代码 38），列表中最后一个值除外。

查询 API 的一种方案是将所有请求参数放入查询字符串中。例如，对 Amazon S3 这样做可以创建预签名 URL。在这种情况下，规范查询字符串不能只包含请求的参数，还必须包含在签名流程中要使用的参数 - 哈希算法、凭证范围、日期和已签名标头参数。

下面的示例说明包含身份验证信息的查询字符串。该示例中为便于阅读添加了换行符，但在您的代码中，规范查询字符串必须是连续的一行文本。

**查询字符串中的身份验证参数示例**

Action=ListUsers&

Version=2010-05-08&

X-Amz-Algorithm=AWS4-HMAC-SHA256&

X-Amz-Credential=AKIDEXAMPLE%2F20150830%2Fus-east-1%2Fiam%2Faws4\_request&

X-Amz-Date=20150830T123600Z&

X-Amz-SignedHeaders=content-type%3Bhost%3Bx-amz-date

有关身份验证参数的更多信息，请参阅 [任务 2：创建签名版本 4 的待签字符串](http://docs.aws.amazon.com/zh_cn/general/latest/gr/sigv4-create-string-to-sign.html)。

**Note**

您可以使用 AWS Security Token Service (AWS STS) 提供的临时安全凭证对请求进行签名。其过程与使用长期凭证相同，但在您添加签名信息到查询字符串时，您必须为安全令牌添加额外的查询参数。参数名称为 X-Amz-Security-Token，参数值为 URI 编码的会话令牌（在您获取临时安全凭证时从 AWS STS 收到的字符串）。

对于一些服务，您必须在规范（签名）查询字符串中包括 X-Amz-Security-Token 查询参数。对于其他服务，计算签名之后，您需要在末尾添加 X-Amz-Security-Token 参数。有关详细信息，请参阅该服务的 API 参考文档。

## V4版本CanonicalHeaders计算方法（storeproxy\_connicalheader\_v4.lua）

规范标头包括您要包含在签名请求中的所有 HTTP 标头的列表。

您必须至少包含 host 标头。标准标头（如 content-type）是可选的。不同的服务可能需要其他标头。

**规范标头示例**

content-type:application/x-www-form-urlencoded; charset=utf-8\n

host:iam.amazonaws.com\n

x-amz-date:20150830T123600Z\n

要创建规范标头列表，请将所有标头名称转换为小写形式并删除前导空格和尾随空格。将标头值中的连续空格转换为单个空格。

以下伪代码描述如何构造规范标头列表：

CanonicalHeaders =

*CanonicalHeadersEntry0* + *CanonicalHeadersEntry1* + ... + *CanonicalHeadersEntryN*

CanonicalHeadersEntry =

Lowercase(*HeaderName*) + ':' + Trimall(*HeaderValue*) + '\n'

Lowercase 表示将所有字符转换为小写字母的函数。Trimall 函数删除值前后的多余空格并将连续空格转换为单个空格。

通过按字符代码对（小写）标头排序，然后对标头名称进行迭代操作，来构建规范标头列表。根据以下规则构造每个标头：

* 追加小写标头名称，后跟冒号。
* 追加该标头的值的逗号分隔列表。请勿对有多个值的标头进行值排序。
* 追加一个新行（“\n”）。

下列示例对更复杂的一组标头及其规范形式进行比较：

**原始标头**

Host:iam.amazonaws.com\n

Content-Type:application/x-www-form-urlencoded; charset=utf-8\n

My-header1:    a   b   c \n

X-Amz-Date:20150830T123600Z\n

My-Header2:    "a   b   c" \n

**规范形式**

content-type:application/x-www-form-urlencoded; charset=utf-8\n

host:iam.amazonaws.com\n

my-header1:a b c\n

my-header2:"a b c"\n

x-amz-date:20150830T123600Z\n

**Note**

每个标头都后跟换行符，这意味着完整列表以换行符结束。

对于规范形式，进行了下列更改：

* 标头名称已转换为小写字符。
* 标头已按字符代码进行排序。
* 已从 my-header1 和 my-header2 值中删除前导空格和尾随空格。
* 对于 my-header1 和 my-header2 值，已将 a b c 中的连续空格转换为单个空格。

**Note**

您可以使用 AWS Security Token Service (AWS STS) 提供的临时安全凭证对请求进行签名。该过程与使用长期凭证相同，但在 Authorization 标头中包括签名信息时，必须为安全令牌添加额外的 HTTP 标头。标头名称为 X-Amz-Security-Token，标头值是会话令牌（在您获取临时安全凭证时从 AWS STS 收到的字符串）。