中国移动公众服务云

【云存储】

开发手册

目 录

1. 概述 5

1.1 简介 5

1.2 特点 5

1.3 概念 5

1.3.1 容器 5

1.3.2 对象 6

1.3.3 数据中心 6

1.3.4 数据一致性 6

1.4 功能 7

1.4.1 容器的拷贝策略 7

1.4.2 访问控制列表 7

1.4.3 多版本 7

1.4.4 操作接口 7

1.5 应用编程API接口 7

1.5.1 Rest接口 7

2. 容器 8

2.1 命名规则 8

2.2 拷贝策略 8

2.3 访问控制 8

3. 对象 9

3.1 属性 9

3.1.1 对象名称 9

3.1.2 元数据 9

3.1.3 版本号 9

3.1.4 访问控制 10

3.2 多拷贝 10

3.3 多版本 10

3.4 MultiPart上传 10

3.4.1 操作用户 10

3.4.2 上传操作 11

3.5 访问期限 12

4. 访问控制列表（ACL） 13

4.1 权限定义 13

4.2 特殊用户定义 13

4.3 访问控制模式定义 14

5. 发送请求 15

5.1 账户 15

5.1.1 访问凭证 15

5.1.2 管理访问凭证 15

5.2 使用SDK 16

5.2.1 JAVA版本 16

5.2.2 C++版本 24

5.2.3 Python版本 34

5.2.4 PHP版本 41

5.3 使用REST API 48

5.3.1 认证过程 48

5.3.2 签名生成 48

5.3.3 容器操作接口描述 50

5.3.4 对象操作接口描述 55

5.3.5 MultiPart上传 61

5.3.6 统计接口描述 65

6. 运营管理平台 69

6.1 接口定义 69

6.1.1 协议形式 69

6.1.2 日期与时间表示规范 69

6.1.3 错误码 70

6.1.4 接口描述 71

6.1.5 请求字段说明 72

6.2 签约对象存储服务（CreateObsUser） 72

6.3 取消对象存储服务（DeleteObsUser） 75

6.4 变更对象存储服务（ModifyObsUser） 77

6.5 查看对象存储服务（QueryObsUser） 80

6.6 增加用户认证信息（CreateObsAccessID） 83

6.7 删除用户认证信息（DeleteObsAccessID） 85

6.8 查询用户认证信息（QueryObsAccessID） 87

# 概述

云存储开发手册首先介绍了云存储的功能及特点，其次介绍了容器、对象和ACL的概念，然后描述了如何使用现有的API来操作容器和对象，包括如何发送请求消息创建容器，如何存储和获取对象，如何设置容器和对象的访问权限。

## 简介

云存储是基于网络的分布式对象存储系统，设计的宗旨是为了给用户提供可扩展、高可靠、高性能的云存储平台，在任何时候从网络的任何地方存储和获取任何大小的数据。云存储为用户提供了简单的API接口实现上层应用。

## 特点

云存储的功能较为单一，主要专注于提供健壮和精简的基础服务，以下是云存储提供的服务：

* 创建容器：容器是数据存储的基础，存储数据首先要创建容器，容器的名称需要全局唯一。
* 在容器中存储数据：容器中存储对象的数目没有限制，每个对象的大小最大为5T(对象大小超过5G需要用MultiPart上传)。
* 下载数据：下载用户所属的数据，或者允许其他用户下载数据。
* 权限：赋予数据的读写权限给其他用户，认证机制保证数据的安全性。
* 标准的接口：使用标准的REST接口。

## 概念

### 容器

容器是对象存储的容器。每个对象都必须存储在容器中，因此上传对象之前必须要先有容器。容器的标识就是容器的名称，因此容器的命名在云存储对象

存储系统中需要唯一存在。

容器的作用有：

* 是云存储中名字空间的最高层。
* 可用于数据存储和数据传输。
* 可用于访问权限控制。
* 可以作为使用空间的统计单元。

### 对象

对象是云存储存储数据的实体，对象包含数据和元数据，元数据是由一系列name-value对构成，定义了对象的属性，其中包含了一些默认属性，例如最后一次修改时间，HTTP标准消息头ContentType等。上层应用可以在上传对象的时候提供可定义的元数据配置。

对象名在容器中是唯一的，对象的标识由容器名，对象名和版本号构成。因此任何一个云存储中的对象的访问URL为：web服务地址+容器名称+对象名称+版本号（可选）。例如：http://www.oNest.com/bucket1/pic.jpg，“bucket1”容器名称，“pic.jpg”为对象名称。

### 数据中心

云存储由多个数据中心构成，数据中心之间由高速网络相连，保证数据传输高效。云存储的数据的高可靠是通过多副本及多个数据中心备份来保证。

### 数据一致性

云存储是通过把数据备份在多个服务器上保证数据的高可靠。数据的备份是根据拷贝策略来备份的，拷贝策略定义了主要数据中心和备份数据中心，其中主要数据中心存储的是原始数据，备份数据中心存储的是副本数据。在写一个对象到云存储时，主要数据中心的数据是同步写入的，而备份数据中心的数据是异步写入的。在从云存储读一个对象的时候，首先要找到这个对象的主要数据中心，然后从主要数据中心读取。因此云存储的对象在写入后立即可见，是一个强一致性的系统。

## 功能

### 容器的拷贝策略

容器在创建的时候系统会分配一个拷贝策略，拷贝策略定义了容器的备份策略，指定了主要数据中心域名和备份的份数以及次要数据中心域名和备份的份数。拷贝策略的分配算法可以配置，例如根据用户的IP地址，容器的命名形式等方法来分配。

### 访问控制列表

参见章节4

### 多版本

参见章节3.3

### 操作接口

oNest提供REST API接口和SDK，基本操作包括:

* 容器操作 参见章节5.2和5.3
* 对象操作 参见章节5.2和5.3

## 应用编程API接口

### Rest接口

在目前的Web服务实现方案中，REST模式的Web服务相对更加简洁。REST从资源的角度来观察整个网络，分布在各处的资源由URI确定，客户端的应用通过URI来获取资源。对资源的操作包括获取、修改、创建和删除资源，这些操作正好对应HTTP协议提供的GET、POST、PUT和DELETE方法。云存储提供的就是标准的基于HTTP协议的REST接口， 云存储中的容器和对象就是两种REST资源，用户可以使用标准的HTTP请求来创建，获取，删除容器和对象。

# 容器

任何一个云存储中的对象都必须包含在一个容器中。所有对象的名字空间都基于它所属的容器。在一个容器的内部，用户可以随意命名对象的名称，只需要保证在容器内的对象名称唯一，但是容器的名称需要在整个云存储中保证唯一存在。只要用户的账户存在容器就一直存在。一个容器只能被一个用户所拥有，一个用户可以最多拥有10个容器（该值可以配置默认是10）。

## 命名规则

容器的命名需要符合以下条件：

1. 必须以字母和数字作为起始字符。
2. 长度范围为[4-64]。
3. 可以由大小写字母，数字，”\_”构成。

## 拷贝策略

容器的拷贝策略在容器的创建的时候就自动分配好了，拷贝策略定义了容器的备份策略，指定了主要数据中心域名和备份的份数以及次要数据中心域名和备份的份数。拷贝策略的分配算法可以配置，例如根据用户的IP地址，容器的命名形式等方法来分配。容器的拷贝策略决定了容器中对象的拷贝策略，拷贝策略为系统的容灾起到了很好的作用。

## 访问控制

容器的访问控制列表定义了容器能被访问的用户和用户访问的权限，用户可以随时修改容器的访问控制信息。访问控制的定义参见章节4.1。

# 对象

对象是云存储中的数据实体，对象必须存在于容器中，对象除了数据实体部分还有对象名称，版本号，元数据，访问控制列表。对象的大小最大为5T。

## 属性

### 对象名称

对象的名称只需要在容器内部保证唯一，对象名称的命名没有限制，可以包含中文字符，对象名称长度范围是[1，255]。

### 元数据

元数据是在上传对象的时候定义在HTTP消息头中的，元数据包含系统定义的元数据和用户定义的元数据。

#### 系统元数据

系统元数据包括对象创建的日期，对象的大小，数据类型等等。

#### 用户定义元数据

用户定义的元数据以name-value形式存在，并且以“x-cmcc-usermeta-”开头，以区别于系统元数据，携带在REST HTTP消息头中，元数据的长度限制小于2kb。

### 版本号

云存储支持对象的多版本存储，因此在容器内部对象名称和版本号可以唯一对应于一个对象实体。多版本参见章节3.3。

### 访问控制

对象的访问控制列表定义了对象能被访问的用户和用户访问的权限，用户可以随时修改对象的访问控制信息。访问控制的定义参见章节4.1。

## 多拷贝

容器在创建的时候就分配了拷贝策略，容器的拷贝策略决定了容器中所有对象的拷贝策略，因此对象的实体数据有多个副本，副本可以存在于一个数据中心或者多个数据中心，对象的副本数越多，可靠性越高。

## 多版本

为了防止用户的误删除操作，对象支持多个版本。对象可以回滚到任意指定版本。对象的每个版本有多个副本。同时，对象的多版本也很好地支持了对象的并发写。当两个用户同时写同一个对象时，将被系统认为是对象的两个版本，而不会因为一个用户正在写一个对象，而拒绝另一个用户的写需求。

## MultiPart上传

系统支持将对象分成多个部分，并行上传，如果对象超过了5G就必须采用MultiPart上传，上传最大大小为5T。每个部分的上传失败都只需重新上传该部分，而不用上传整个对象。对象的下载也支持多个部分并行下载。

对象的多点上传及下载大大减少了大对象上传及下载的时延，同时使得上传及下载过程中出现的故障更快的恢复。

### 操作用户

MultiPart上传的用户必须对容器有写权限，操作用户包括：

* 注册用户：在云存储注册过的，通过认证的用户。
* 匿名用户：系统中的一个特殊的用户，对应用户名” CMCC-OBS-ANONYMOUS-USERS”。 操作的HTTP头部不带Authorization。

### 上传操作

MultiPart上传操作包括：

* Init MultiPart Upload

操作用户在MultiPart上传前，需要初始化MultiPart上传，此时设置了MultiPart的对象所属人（Owner）为操作用户，并将对象上传操作信息记录在UploadingObj对象中。操作用户完成了MultiPart初始化后可以进行下面的操作。

该操作如果容器不存在返回ERR404\_No\_Such\_Container。

* Upload Part

上传Part，只有MultiPart的对象所属人可以操作。

该操作如果容器不存在返回ERR404\_No\_Such\_Container，如果UploadingObj不存在，返回ERR404\_No\_Such\_Upload。

* Complete Multipart Upload

完成Multipart上传，只有MultiPart的对象所属人可以操作。

该操作如果容器不存在返回ERR404\_No\_Such\_Container，如果UploadingObj不存在返回ERR404\_No\_Such\_Upload。

* Abort MultiPart Upload

取消Multipart上传，此操作除了MultiPart的对象所属人之外，容器的所属人也有权限操作。

该操作如果容器不存在返回ERR404\_No\_Such\_Container，如果UploadingObj不存在返回ERR404\_No\_Such\_Upload。

* List Parts

查看某个UploadingObj的Part列表，此操作除了MultiPart的对象所属人之外，容器的所属人也有权限操作。

该操作如果容器不存在返回ERR404\_No\_Such\_Container，如果UploadingObj不存在返回ERR404\_No\_Such\_Upload。

* List Multipart uploads

查看容器下面的UploadingObj列表，此操作用户有容器的读权限即可操作。

该操作如果容器不存在返回ERR404\_No\_Such\_Container。

## 访问期限

对象通过设置访问控制列表，可以将某个用户的对象共享给其他用户。云存储Consle资源管理系统提供一个页面生成URL，该URL中携带了对象的过期时间，共享用户在过期时间内可以正常访问该对象；当超出过期时间后将会无法访问对象，这个过期时间就是对象的访问期限。如果想详细了解云存储Consle资源管理系统请参考文档《云存储Console用户使用指南》。

# 访问控制列表（ACL）

访问控制列表定义了一系列的访问控制，访问控制由权限授予人和权限组成，授予人包括匿名用户和注册用户。容器和对象都可以设置访问控制列表。

## 权限定义

系统支持的ACL宏定义如下表：

|  |  |  |
| --- | --- | --- |
| ACL宏 | 容器级访问控制功能 | 对象级访问控制功能 |
| **READ** | 允许对容器执行列出对象的操作 | 允许读取对象数据和元数据（除ACL） |
| **WRITE** | 允许创建、覆盖、删除该容器中的对象和对象元数据 | 不支持 |
| **READ\_ACP** | 只有容器和对象的拥有者有该权限，缺省隐含有该权限 | |
| **WRITE\_ACP** | 只有容器和对象的拥有者有该权限，缺省隐含有该权限 | |
| **FULL\_CONTROL** | 允许非拥有者用户拥有READ、WRITE权限，拥有者拥有全部权限 | |

例如：[UserID，READ|WRITE] 是容器ContainerName的一组访问控制定义，那么用户UserID可以列出容器ContainerName的对象列表，可以创建、覆盖、删除容器ContainerName中的对象和对象元数据。如果UserID是容器ContainerName的拥有者，UserID还可以读写容器的ACL。

## 特殊用户定义

系统包含两类特殊用户标识,普通用户不能使用：

* CMCC-OBS-ALL-REGISTED-USERS：所有已注册的CMCC对象存储业务用户
* CMCC-OBS-ANONYMOUS-USERS：匿名用户

## 访问控制模式定义

系统在执行对象或容器的PUT操作时，可以通过使用扩展头“x-cmcc-acl”的方式，同步设置对象或容器的访问控制列表，下表是根据常见的访问控制模式，封装的一些常见的访问控制模式：

|  |  |
| --- | --- |
| 访问控制模式宏 | 描述说明 |
| **default** | 拥有者具有FULL\_CONTROL的权限，容器或对象创建时，缺省权限就是**default** |
| **anonymous-read** | 拥有者具有FULL\_CONTROL权限，匿名用户拥有READ权限 |
| **anonymous-read-write** | 拥有者具有FULL\_CONTROL权限，匿名用户拥有READ | WRITE权限 |
| **obs-user-read** | 拥有者具有FULL\_CONTROL权限，任意注册用户拥有READ权限 |
| **obs-user-read-write** | 拥有者具有FULL\_CONTROL权限，任意注册用户拥有READ | WRITE权限 |
| **container-owner-read** | 对于容器操作：等同于**default**模式；  对于对象操作：对象拥有者具有FULL\_CONTROL权限，对象所在容器的拥有者，拥有READ权限； |
| **container-owner-full-control** | 对于容器操作：等同于**default**模式；  对于对象操作：对象拥有者具有FULL\_CONTROL权限，对象所在容器的拥有者，拥有FULL\_CONTROL权限； |

# 发送请求

云存储提供的服务是REST接口。用户可以使用REST API或者云存储SDK来发送请求消息到云存储。任何注册用户与云存储交互的消息都是需要认证的，认证成功对象存储系统才会接受发送请求。认证请求必须包含签名信息，签名信息是由访问凭证等信息加密生成的。如果用户使用的是云存储SDK，SDK会根据你提供的访问凭证来生成签名，如果使用的是REST接口，那么就需要自己写代码来生成签名信息。

## 账户

要发送请求消息到云存储，首先需要创建一个云存储账户，账户可以通过两种形式创建，一种是在Web页面通过邮箱来注册，注册邮箱长度范围是[3，31]，密码可以是任意字符，长度范围是[3，15]。注册流程请参考《云存储AAA使用手册》。另外一种方式是通过SOAP接口创建，用户ID可以由大小写字母，数字,“.”,“\_”，“-”，“@”构成，长度范围为[3，31]，SOAP接口参见章节6。

### 访问凭证

访问凭证由签名密钥标识和签名密钥构成。签名密钥标识由20个字符构成，例如：7ea5e2470957363f89ff，签名密钥由40个字符构成，例如：kWcrlUX5JEDGM/LtmEENI/aVmYvHNif5zB+d9+ct，签名密钥标识可以唯一确定一个账户。访问凭证的状态有激活和未激活两种状态，激活的访问凭证才是有效的，使用未激活的访问凭证不能够访问云存储。

### 管理访问凭证

用户最多可以创建十个访问凭证，最多可以拥有两个激活的访问凭证。新创建的访问凭证是未激活的，激活后的访问凭证才可以有效使用。用户可以通过菜单将访问凭证失效，用户可以将所有访问凭证都失效；用户可以删除访问凭证。如果要详细了解访问凭证的管理，请参考文档《云存储AAA使用手册》。

## 使用SDK

使用SDK的时候需要由用户来指定是内网模式还是外网模式，内网模式指客户端和云存储在相同的内网，外网模式指客户端和云存储不在同一个内网。

### JAVA版本

使用JAVA版本的SDK，首先要根据配置文件“oNestCredentials.properties”中的配置信息做初始化工作，配置文件中包含用户的访问凭证，云存储的地址信息，内外网模式等，。初始化代码示例如下：

oNestClientFactory ocf = oNestClientFactory.getInstance();

PropertiesCredentials oNestProperites = new PropertiesCredentials("oNestCredentials.properties");

ocf.setFactoryInfo(oNestProperites, oNestProperites.getoNestHostAddr());

oNest oNest;

String isIntraAddr = oNestProperites.getIsIntraAddr();

if(isIntraAddr.equals("true"))//配置为内网地址

oNest = ocf.getLocation(false,false);

else//配置为外网地址

oNest = ocf.getLocation(true,false);

初始化好的云存储对象可以对云存储进行操作。

#### 容器操作

**获取用户所有容器**

通过listBuckets()接口可以获取用户的容器列表，代码示例如下：

try {

for (Bucket bucket : oNest.listBuckets()) {

System.out.println(bucket.getName()+",create time:" + bucket.getCreationDate());

}

}catch (oNestServiceException ose) {

System.out.println("Caught an oNestServiceException.");

}catch (oNestClientException oce) {

System.out.println("Caught an oNestClientException.");

}

**创建容器**

通过createBucket()接口可以创建容器，下面的代码示例是创建容器“test\_bucket”：

try {

if(oNest.createBucket("test\_bucket") != null){

System.out.println("Creating bucket " + bucketName + " sucess!\n");

}

}

else {

System.out.println("Creating bucket " + bucketName + " failed!\n");

}catch (oNestServiceException ose) {

System.out.println("Caught an oNestServiceException.");

}catch (oNestClientException oce) {

System.out.println("Caught an oNestClientException.");

}

**删除容器**

通过deleteBucket()接口可以删除容器，删除容器前需要先删除容器下面的对象，下面的代码示例是删除容器“test\_bucket”：

try {

oNest.deleteBucket("test\_bucket");

}catch (oNestServiceException ose) {

System.out.println("Caught an oNestServiceException.");

}catch (oNestClientException oce) {

System.out.println("Caught an oNestClientException.");

}

**设置容器权限**

通过setBucketAcl()接口可以设置容器的权限，下面的代码示例是设置容器“test\_bucket”的权限为test1可读写，test2可读写、可读写ACP：

try {

AccessControlList acl = new AccessControlList();

Owner own = new Owner();

own.setId(USER\_ID);

acl.setOwner(own);

Grantee grantee1 = new CanonicalGrantee("test1");

Grantee grantee2 = new CanonicalGrantee("test2");

List<Permission> listPermission = new LinkedList<Permission>();

listPermission.add(Permission.Read);

listPermission.add(Permission.Write);

acl.grantPermission(grantee1, listPermission);

List<Permission> listPermission2 = new LinkedList<Permission>();

listPermission2.add(Permission.Read);

listPermission2.add(Permission.Write);

acl.grantPermission(grantee2, listPermission2);

oNest.setBucketAcl("test\_bucket", acl);

}catch (oNestServiceException ose) {

System.out.println("Caught an oNestServiceException.");

}catch (oNestClientException oce) {

System.out.println("Caught an oNestClientException.");

}

**获取容器权限**

通过getBucketAcl()接口可以获取容器的权限，下面的代码示例是获取容器“test\_bucket”的权限：

try {

AccessControlList acl = oNest.getBucketAcl("test\_bucket");

}catch (oNestServiceException ose) {

System.out.println("Caught an oNestServiceException.");

}catch (oNestClientException oce) {

System.out.println("Caught an oNestClientException.");

}

**获取容器中对象列表**

通过listObjects()接口可以获取容器中的对象列表，下面的代码示例是获取容器“test\_bucket”的以“My”字符串开头的对象列表：

try {

ObjectListing objectListing = oNest.listObjects(new ListObjectsRequest().

withBucketName("test\_bucket").withPrefix("My"))

}catch (oNestServiceException ose) {

System.out.println("Caught an oNestServiceException.");

}catch (oNestClientException oce) {

System.out.println("Caught an oNestClientException.");

}

#### 对象操作

**创建对象**

通过putObject()接口可以创建对象，下面的代码示例是在容器“test\_bucket”中创建对象“test\_object”：

try {

oNest.putObject(new PutObjectRequest("test\_bucket", "test\_object",

createSampleFile()));

}catch (oNestServiceException ose) {

System.out.println("Caught an oNestServiceException.");

}catch (oNestClientException oce) {

System.out.println("Caught an oNestClientException.");

}

**删除对象**

通过接口deleteObject()可以删除对象，下面的代码示例是删除容器“test\_bucket”中对象“test\_object”：

try {

oNest.deleteObject("test\_bucket", "test\_object");

}catch (oNestServiceException ose) {

System.out.println("Caught an oNestServiceException.");

}catch (oNestClientException oce) {

System.out.println("Caught an oNestClientException.");

}

**获取对象**

通过getObject()接口可以获取对象，下面的代码示例是获取容器“test\_bucket”中对象“test\_object”：

try {

oNestObject object = oNest.getObject(new GetObjectRequest("test\_bucket",

"test\_object"));

}catch (oNestServiceException ose) {

System.out.println("Caught an oNestServiceException.");

}catch (oNestClientException oce) {

System.out.println("Caught an oNestClientException.");

}

**获取对象属性**

通过getObjectMetadata()接口可以获取对象属性，下面的代码示例是获取容器“test\_bucket”中对象“test\_object”的属性：

try {

GetObjectMetadataRequest getObjectMetadataRequest = new

GetObjectMetadataRequest("test\_bucket", "test\_object");

ObjectMetadata omd = oNest.getObjectMetadata(getObjectMetadataRequest);

}catch (oNestServiceException ose) {

System.out.println("Caught an oNestServiceException.");

}catch (oNestClientException oce) {

System.out.println("Caught an oNestClientException.");

}

**设置对象属性**

通过setObjectMetadata()接口可以设置对象属性，下面代码示例是设置容器“test\_bucket”中对象“test\_object”的属性key1的值为value1，属性key2的值为value2，函数会自动在属性前面增加“x-cmcc-usermeta-”

try {

ObjectMetadata objMetadata = new ObjectMetadata();

Map<String, String> userMeta = objMetadata.getUserMetadata();

userMeta.put("key1", "value1");

userMeta.put("key2", "value2");

oNest.setObjectMetadata("test\_bucket", "test\_object", objMetadata);

}catch (oNestServiceException ose) {

System.out.println("Caught an oNestServiceException.");

}catch (oNestClientException oce) {

System.out.println("Caught an oNestClientException.");

}

**获取对象权限**

通过getObjectAcl()接口可以获取对象权限，下面的代码示例是获取容器“test\_bucket”中对象“test\_object”的权限：

try {

AccessControlList obj\_acl = oNest.getObjectAcl("test\_bucket", "test\_object");

}catch (oNestServiceException ose) {

System.out.println("Caught an oNestServiceException.");

}catch (oNestClientException oce) {

System.out.println("Caught an oNestClientException.");

}

**设置对象权限**

通过setObjectAcl()接口可以设置对象权限，下面的代码示例是设置容器“test\_bucket”中对象“test\_object”的权限为test1可读、test2可写：

try {

Owner own = new Owner();

own.setId(USER\_ID);

AccessControlList obj\_acl = new AccessControlList();

obj\_acl.setOwner(own);

obj\_acl.grantPermission("test1", Permission.Read);

obj\_acl.grantPermission("test2", Permission.Write);

oNest.setObjectAcl("test\_bucket", "test\_object", obj\_acl);

}catch (oNestServiceException ose) {

System.out.println("Caught an oNestServiceException.");

}catch (oNestClientException oce) {

System.out.println("Caught an oNestClientException.");

}

### C++版本

使用C++版本的SDK，首先要根据配置文件“config.xml”中的配置信息做初始化工作，配置文件中包含用户的访问凭证，云存储的地址信息，内外网模式等，。初始化代码示例如下：

Credentials cred(accesskey,secreykey);

oNestClient ->setCredentials(cred);

Config config;

if(!config.init("../conf/config.xml"))

return false;

oNestClient \* oNestClient = new oNestClient();

oNestClient->init(config);

初始化好的oNestClient指针可以对oNest进行操作。

#### 容器操作

**获取用户所有容器**

通过getService接口可以获取用户的容器列表，代码示例如下：

try{

ListAllMyBucketResult result = oNestClient->getService();

}

catch(Error & error)

{

LogInfo("Error code:%s ,message:%s", error.getCode().c\_str()

,error.getMessage().c\_str());

throw error;

}

**创建容器**

通过putBucket()接口可以创建容器，创建容器前可以用setAclist()设置访问控制模式，下面的代码示例是创建容器“test\_bucket”，并设置访问控制模式为默认方式：

try{

Bucket bucket;

bucket.setBucketName("test\_bucket");

AccessControlList acllist;

acllist.setSpecAcl("default");// 设置访问控制模式为默认方式

bucket.setAclist(acllist);

oNestClient->putBucket(bucket);

}

catch(Error & error)

{

LogInfo("Error code:%s ,message:%s", error.getCode().c\_str()

,error.getMessage().c\_str());

throw error;

}

**删除容器**

通过delBucket()接口可以删除容器，删除容器前需要先删除容器下面的对象，下面的代码示例是删除容器“test\_bucket”：

try{

oNestClient->delBucket("test\_bucket");

}

catch(Error & error)

{

LogError("Error code:%s ,message:%s", error.getCode().c\_str()

,error.getMessage().c\_str());

throw error;

}

**设置容器权限**

通过putBucketAcl()接口可以设置容器的权限，下面的代码示例是设置容器“test\_bucket”的权限为test1可读写、可写ACP，test2可读写：

try{

Bucket bucket;

bucket.setBucketName("test\_bucket");

AccessControlList acl;

acl.setAclEnty("test1",AclType::READ|AclType::WRITE);

acl.setAclEnty("test2",AclType::READ|AclType::WRITE);

bucket.setAclist(acl);

oNestClient->putBucketAcl(bucket);

}

catch(Error & error)

{

LogError("Error code:%s ,message:%s", error.getCode().c\_str()

,error.getMessage().c\_str());

throw error;

}

**获取容器权限**

通过getBucketAcl()接口可以获取容器的权限，下面的代码示例是获取容器“test\_bucket”的权限：

try{

AccessControlList acl= oNestClient->getBucketAcl("test\_bucket");

}

catch(Error & error)

{

LogError("Error code:%s ,message:%s", error.getCode().c\_str()

,error.getMessage().c\_str());

throw error;

}

**获取容器中对象列表**

通过getBucket()接口可以获取容器中的对象列表，下面的代码示例是获取容器“test\_bucket”的对象列表：

try{

ListBucketResult result = oNestClient->getBucket("test\_bucket");

}

catch(Error & error)

{

LogError("Error code:%s ,message:%s", error.getCode().c\_str()

,error.getMessage().c\_str());

throw error;

}

#### 对象操作

**创建对象**

通过putObject()接口可以创建对象，下面的代码示例是在容器“test\_bucket”中创建对象“test\_object”：

Object obj("test\_object");

obj.setBucketName("test\_bucket");

FILE \* file = fopen("/test.log","rb");

if(file == NULL)

{

LogError("file（%s） is not exist...\n",up\_file.c\_str());

return;

}

FileData fildata(file);

obj.setData(&fildata);

try{

oNestClient->putObject(obj);

fclose(file);

}

catch(Error & error)

{

fclose(file);

LogError("Error code:%s ,message:%s", error.getCode().c\_str()

,error.getMessage().c\_str());

throw error;

}

**删除对象**

通过接口delObject()可以删除对象，下面的代码示例是删除容器“test\_bucket”中对象“test\_object”：

try{

Object obj("test\_object");

obj.setBucketName("test\_bucket");

oNestClient->delObject(obj);

}

catch(Error & error)

{

LogError("Error code:%s ,message:%s", error.getCode().c\_str()

,error.getMessage().c\_str());

throw error;

}

**获取对象**

通过getObject()接口可以获取对象，下面的代码示例是获取容器“test\_bucket”中对象“test\_object”：

try{

Object obj("test\_object");

obj.setBucketName("test\_bucket");

FILE \* file = fopen(down\_file.c\_str(),"wb+");

if(file == NULL)

return true;

FileData fildata(file);

obj.setData(&fildata);

oNestClient->getObject(obj);

fclose(file);

}

catch(Error & error)

{

LogError("Error code:%s ,message:%s", error.getCode().c\_str()

,error.getMessage().c\_str());

throw error;

}

**获取对象属性**

通过getObjectMeta()接口可以获取对象属性，下面的代码示例是获取容器“test\_bucket”中对象“test\_object”的属性：

try{

Object obj("test\_object");

obj.setBucketName("test\_bucket");

oNestClient->getObjectMeta(obj);

}

catch(Error & error)

{

LogError("Error code:%s ,message:%s", error.getCode().c\_str()

,error.getMessage().c\_str());

throw error;

}

**设置对象属性**

通过putObjectMeta()接口可以设置对象属性，下面代码示例是设置容器“test\_bucket”中对象“test\_object”的属性metakey1的值为metavalue1, 属性metakey2的值为metavalue2，函数会自动在属性前面增加“x-cmcc-usermeta-”。

try{

Object obj("test\_object");

obj.setBucketName("test\_bucket");

Meta meta;

meta.setMetaEntry("metakey1","metavalue1");

meta.setMetaEntry("metakey2","metavalue2");

obj.setMeta(meta);

oNestClient->putObjectMeta(obj);

}

catch(Error & error)

{

LogError("Error code:%s ,message:%s", error.getCode().c\_str()

,error.getMessage().c\_str());

throw error;

}

**获取对象权限**

通过getObjectAcl()接口可以获取对象权限，下面的代码示例是获取容器“test\_bucket”中对象“test\_object”的权限：

try{

Object obj("test\_object");

obj.setBucketName("test\_bucket");

AccessControlList acl = oNestClient->getObjectAcl(obj);

}

catch(Error & error)

{

LogError("Error code:%s ,message:%s"

, error.getCode().c\_str(),error.getMessage().c\_str());

throw error;

}

**设置对象权限**

通过putObjectAcl()接口可以设置对象权限，下面的代码示例是设置容器“test\_bucket”中对象“test\_object”的权限为test1可读，test2可写：

try{

Object obj("test\_object");

obj.setBucketName("test\_bucket");

AccessControlList acl;

acl.setOwner("otest");

acl.setSpecAcl(AclType::DEFAULT);

acl.setAclEnty("test1",AclType::READ);

acl.setAclEnty("test2",AclType::WRITE);

obj.setAclist(acl);

oNestClient->putObjectAcl(obj);

}

catch(Error & error)

{

LogError("Error code:%s ,message:%s"

, error.getCode().c\_str(),error.getMessage().c\_str());

throw error;

}

### Python版本

使用Python版本的SDK，首先要做初始化工作，包含用户的访问凭证，oNest的地址信息，内外网模式等。初始化代码示例如下，“CMCC”是协议版本，“fa2cce67eeded9c81526”是签名密钥标识，“5ebe7f43fdcd5f03b0d66c7a21053f7e9e9da005”是签名密钥，“False”指不采用Https方式，“10.24.1.45:18080”是对象存储系统的入口地址，“1”表示采用内网通讯模式:

authinfo = oNest\_common.AuthInfo('CMCC', 'fa2cce67eeded9c81526',

'5ebe7f43fdcd5f03b0d66c7a21053f7e9e9da005', False, '10.24.1.45:18080', 1);

self.oNest = oNest\_client.oNestClient(authinfo)

初始化好的云存储对象就可以操作云存储。

#### 容器操作

**获取用户所有容器**

通过list\_buckets\_of\_user()接口可以获取用户的容器列表，代码示例如下：

response = self.oNest.list\_buckets\_of\_user()

if response == None:

print 'list\_buckets\_of\_user\_test failed...'

else:

print response.body

print 'owner: %s' % response.owner

print 'buckets num: %d' % len(response.entries)

for item in response.entries:

print ' container\_name: %s' % item.container\_name

print ' create\_date: %s' % item.create\_date

**创建容器**

通过create\_bucket()接口可以创建容器，下面的代码示例是创建容器“test\_bucket”：

if self.oNest.create\_bucket('test\_bucket') == True:

print 'create\_bucket ok...'

else:

print 'create\_bucket failed...'

**删除容器**

通过delete\_bucket()接口可以删除容器，删除容器前需要先删除容器下面的对象，下面的代码示例是删除容器“test\_bucket”：

if self.oNest.delete\_bucket('test\_bucket') == True:

print 'delete\_bucket ok...'

else:

print 'delete\_bucket failed...'

**设置容器权限**

通过put\_bucket\_acl()接口可以设置容器的权限，下面的代码示例是设置容器“test\_bucket”的权限为用户test可读写：

bucket\_acl = '<AccessControlPolicy>\

<Owner>\

<UserID>admin</UserID>\

</Owner>\

<AccessControlList>\

<Grant>\

<GranteeID>\

<ID>test</ID>\

</GranteeID>\

<Permission>READ|WRITE</Permission>\

</Grant>\

</AccessControlList>\

</AccessControlPolicy>'

if self.oNest.put\_bucket\_acl(‘test\_bucket’, bucket\_acl) == True:

print 'put\_bucket\_acl ok...'

else:

print 'put\_bucket\_acl failed...'

**获取容器权限**

通过get\_bucket\_acl()接口可以获取容器的权限，下面的代码示例是获取容器“test\_bucket”的权限：

response = self.oNest.get\_bucket\_acl('test\_bucket')

if response == None:

print 'get\_bucket\_acl\_test failed...'

else:

response.body

print 'owner: %s' % response.owner

print 'acls num: %d' % len(response.entries)

for item in response.entries:

print ' grantee: %s' % item.grantee

print ' permission: %s' % item.permission

**获取容器中对象列表**

通过list\_objects\_of\_bucket()接口可以获取容器中的对象列表，下面的代码示例是获取容器“test\_bucket”的对象列表：

response = self.oNest.list\_objects\_of\_bucket('test\_bucket')

if response == None:

print 'list\_objects\_of\_bucket\_test failed...'

else:

print response.body

print 'container\_name: %s' % response.container\_name

print 'prefix: %s' % response.prefix

print 'max\_results: %s' % response.max\_results

print 'marker: %s' % response.marker

print 'is\_truncated: %s' % response.is\_truncated

print 'objects num: %d' % len(response.entries)

for item in response.entries:

print ' object\_uri: %s' % item.object\_uri

print ' ctime: %s' % item.ctime

print ' size: %s' % item.size

#### 对象操作

**创建对象**

通过put\_object()接口可以创建对象，下面的代码示例是在容器“test\_bucket”中创建对象“test\_object”，其内容来自于“F:\\Dick\\test.txt”：

if self.oNest.put\_object('test\_bucket', 'test\_object', 'F:\\Dick\\test.txt') == True:

print 'put\_object ok...'

else:

print 'put\_object failed...'

**删除对象**

通过接口delete\_object()可以删除对象，下面的代码示例是删除容器“test\_bucket”中对象“test\_object”：

if self.oNest.delete\_object('test\_bucket', 'test\_object') == True:

print 'delete\_object ok...'

else:

print 'delete\_object failed...'

**获取对象**

通过get\_object\_data()接口可以获取对象数据，下面的代码示例是获取容器“test\_bucket”中对象“test\_object”的数据：

data = self.oNest.get\_object\_data('test\_bucket', 'test\_object')

if data == None:

print 'get\_object\_data\_test failed...'

else:

print data

**获取对象属性**

通过get\_object\_meta()接口可以获取对象属性，下面的代码示例是获取容器“test\_bucket”中对象“test\_object”的属性：

resp = self.oNest.get\_object\_meta('test\_bucket', 'test\_object')

if resp == None:

print 'get\_object\_meta\_test failed...'

else:

print resp

**设置对象属性**

通过put\_object\_meta()接口可以设置对象属性，下面代码示例是设置容器“test\_bucket”中对象“test\_object”的属性key1的值为value1，属性key2的值为value2，python接口需要用户在属性前面加上“x-cmcc-usermeta-meta-”。

object\_meta = {'x-cmcc-usermeta-metakey1': 'value1', 'x-cmcc-usermeta-metakey2':'value2'}

if self.oNest.put\_object\_meta('test\_bucket', OBJECT\_KEY, object\_meta) == True:

print 'put\_object\_meta ok...'

else:

print 'put\_object\_meta failed...'

**获取对象权限**

通过get\_object\_acl()接口可以获取对象权限，下面的代码示例是获取容器“test\_bucket”中对象“test\_object”的权限：

response = self.oNest.get\_object\_acl('test\_bucket', 'test\_object')

if response == None:

print 'get\_object\_acl\_test failed...'

else:

response.body

print 'owner: %s' % response.owner

print 'acls num: %d' % len(response.entries)

for item in response.entries:

print ' grantee: %s' % item.grantee

print ' permission: %s' % item.permission

**设置对象权限**

通过put\_object\_acl()接口可以设置对象权限，下面的代码示例是设置容器“test\_bucket”中对象“test\_object”的权限为test可读：

object\_acl = '<AccessControlPolicy>\

<Owner>\

<UserID>admin</UserID>\

</Owner>\

<AccessControlList>\

<Grant>\

<GranteeID>\

<ID>test</ID>\

</GranteeID>\

<Permission>READ</Permission>\

</Grant>\

</AccessControlList>\

</AccessControlPolicy>'

if self.oNest.put\_object\_acl('test\_bucket', 'test\_object', object\_acl) == True:

print 'put\_object\_acl ok...'

else:

print 'put\_object\_acl failed...'

### PHP版本

#### 容器操作

使用PHP版本的SDK，首先要做初始化工作，包含用户的访问凭证，云存储的地址信息，内外网模式等。初始化代码示例如下:

public $key = '3016128d2411248396d6' ;

public $secret\_key = '0b706dbcfc850ec526a38872fa9b9da618df435c';

public $OwnerID = 'yierhe@gmail.com';

public $hostname = '10.24.1.45:48284';

$oNest = new CmccoNest();

初始化好的云存储对象可以对云存储进行操作。

**获取用户所有容器**

通过list\_buckets()接口可以获取用户的容器列表，代码示例如下：

$response=$oNest->list\_buckets();

$response\_status=(int)$response->status;

if($response\_status !== 200){

echo ‘error’;

}else{

echo ‘success’;

}

**创建容器**

通过create\_bucket()接口可以创建容器，下面的代码示例是创建容器“test\_bucket”：

$response=$oNest->create\_bucket(’test\_bucket’);

$response\_status=(int)$response->status;

if($response\_status !== 201{

echo ‘error’;

}else{

echo ‘success’;

}

**删除容器**

通过delete\_bucket()接口可以删除容器，删除容器前需要先删除容器下面的对象，下面的代码示例是删除容器“test\_bucket”：

$response=$oNest->delete\_bucket(’test\_bucket’);

$response\_status=(int)$response->status;

if($response\_status !== 204){

echo ‘error’;

}else{

echo ‘success’;

}

**设置容器权限**

通过set\_bucket\_acl()接口可以设置容器的权限，下面的代码示例是设置容器“test\_bucket”的访问控制权限为test1可读，test2可读：

$optt=array(array( 'id' => 'test1','permission' => 'READ'),

array( 'id' => 'test2','permission' => 'READ'));

$response=$oNest->set\_bucket\_acl(’test\_bucket’, $optt);

$response\_status=(int)$response->status;

if($response\_status !== 200){

echo ‘error’;

}else{

echo ‘success’;

}

**获取容器权限**

通过get\_bucket\_acl()接口可以获取容器的权限，下面的代码示例是获取容器“test\_bucket”的权限：

$response=$oNest->get\_bucket\_acl(’test\_bucket’);

$response\_status=(int)$response->status;

if($response\_status !== 200){

echo ‘error’;

}else{

echo ‘success’;

}

**获取容器中对象列表**

通过list\_objects()接口可以获取容器中的对象列表，下面的代码示例是获取容器“test\_bucket”的以“My”字符串开头的对象列表，最多返回50条记录：

$optt = array('prefix' => 'My', 'maxResults' => 50 );

$response=$oNest->list\_objects(’test\_bucket’,$optt);

$response\_status=(int)$response->status;

if($response\_status !== 200){

echo ‘error’;

}else{

echo ‘success’;

}

#### 对象操作

**创建对象**

通过create\_object()接口可以创建对象，下面的代码示例是在容器“test\_bucket”中创建对象“test\_object”：

$response=$oNest->create\_object(’test\_bucket’, ’test\_object’);

$response\_status=(int)$response->status;

if($response\_status !== 201){

echo ‘error’;

}else{

echo ‘success’;

}

**删除对象**

通过接口delete\_object()可以删除对象，下面的代码示例是删除容器“test\_bucket”中对象“test\_object”：

$response=$oNest->delete\_object(’test\_bucket’, ’test\_object’);

$response\_status=(int)$response->status;

if($response\_status !== 204){

echo ‘error’;

}else{

echo ‘success’;

}

**获取对象**

通过get\_object()接口可以获取对象，下面的代码示例是获取容器“test\_bucket”中对象“test\_object”：

$response=$oNest->get\_object(’test\_bucket’, ’test\_object’);

$response\_status=(int)$response->status;

if($response\_status !== 200){

echo ‘error’;

}else{

echo ‘success’;

}

**获取对象属性**

通过get\_object\_metadata()接口可以获取对象属性，下面的代码示例是获取容器“test\_bucket”中对象“test\_object”的属性：

$response=$oNest->get\_object\_metadata(’test\_bucket’, ’test\_object’);

$response\_status=(int)$response->status;

if($response\_status !== 200){

echo ‘error’;

}else{

echo ‘success’;

}

**设置对象属性**

通过set\_object\_metadata()接口可以设置对象属性，下面代码示例是设置容器“test\_bucket”中对象“test\_object”的属性为“metakey1- metavalue1，metakey2- metavalue2”：

$optt['headers']['x-cmcc-usermeta-metakey1'] = 'metavalue1';

$optt['headers']['x-cmcc-usermeta-metakey2'] = 'metavalue2';

$response=$oNest->set\_object\_metadata(’test\_bucket’, ’test\_object’,$optt);

$response\_status=(int)$response->status;

if($response\_status !== 200){

echo ‘error’;

}else{

echo ‘success’;

}

**获取对象权限**

通过get\_object\_acl()接口可以获取对象权限，下面的代码示例是获取容器“test\_bucket”中对象“test\_object”的权限：

$response=$oNest->get\_object\_acl(’test\_bucket’, ’test\_object’);

$response\_status=(int)$response->status;

if($response\_status !== 200){

echo ‘error’;

}else{

echo ‘success’;

}

**设置对象权限**

通过set\_object\_acl()接口可以设置对象权限，下面的代码示例是设置容器“test\_bucket”中对象“test\_object”的权限为test1可读，test2可读：

$optt = array(array( 'id' => 'test1',

'permission' => 'READ'),

array( 'id' => 'test2',

'permission' => 'READ'));

$response=$oNest->set\_object\_acl(’test\_bucket’, ’test\_object’，$optt);

$response\_status=(int)$response->status;

if($response\_status !== 200){

echo ‘error’;

}else{

echo ‘success’;

}

## 使用REST API

### 认证过程

发送到云存储的请求必须包含认证信息，认证信息中必须提供以下几个参数：

* 签名密钥标识：用户接入时的访问凭证，此信息是用户注册后系统分配给用户的。
* 签名：是用签名密钥加密，请求时按一定的算法计算产生。
* 日期：GMT时间。

云存储接受到请求消息后，首先比较消息头中的日期和系统的日期，如果差值在有效范围内（24小时以内）才认为是有效消息，然后查找签名密钥标识是否存在，检查签名密钥标识的状态是否有效(，再查找签名密钥标识对应的用户是否有效。如果都有效，最后按照相同的算法计算签名，如果计算出的签名和消息头中携带的签名相匹配就认为认证成功。

### 签名生成

#### 生成算法

签名生成算法如下：

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

其中：

Base64参考 RFC 2405规范。

HMAC-SHA1参考RFC 2104规范，需要两个参数，AccessSecretKey和StringToSign。

StringToSign = HTTP-Verb +"\n"+

Content-MD5 +”\n”+

Content-Type +"\n"+

Date +"\n"+

CanonicalizedHeaders + CanonicalizedResource;

其中：

* HTTP-Verb: 即为http请求操作的VERB，如: "PUT"，，"GET","DELETE"等字符串
* Content-MD5：HTTP消息头域，没有该域填空
* Content-Type：HTTP消息头的文档类型域，没有该域填空
* Date：消息发送的时间，采用GMT格式
* CanonicalizedHeaders: Http扩展头信息集合
* CanonicalizedResource : Http-Verb操作的目标资源URI

扩展信息部分：

* 当使用扩展时间戳（x-cmcc-date）时，扩展时间戳替代HTTP请求中的 Date 域参与计算签名，StringToSign 字串中Date取空字符串。
* CanonicalizedHeaders构建方法为：

1. 将所有的扩展头名称转换为小写字母，如X-CMCC-Date，转化为x-cmcc-date
2. 清除每个扩展头的空格和换行符
3. 在所有的扩展头后面插入换行符（U+000A）
4. 按字典序排列拼接所有的扩展头形成CanonicalizedHeaders

#### 实例

以REST接口为例：

|  |  |
| --- | --- |
| 请求消息头 | 计算签名的字符串 |
| PUT /mycontainer/baby.jpg HTTP/1.1  Content-Type: image/jpeg  Content-Length: 43567  Host: obs.chinamobile.com  Date: Sun, 6 Feb 2011 21:30:00 +0000  Authorization:CMCC TIKVJ979BgZHT9826KJP:lkgsdrili87vkjn9o8dvL9SsO6Ak | PUT\n  \n  image/jpeg\n  Sun, 6 Feb 2010 21:30:00 +0000\n  \n  /mycontainer/baby.jpg |
| PUT /mycontainer/finance.txt HTTP/1.1  Host: obs.chinamobile.com  Date: Mon, 7 Feb 2011 08:30:00 +0000  content-type: text/plain  Content-MD5: jn9o098hBGZHT94BqNli87vk  x-cmcc-usermeta-metakey1:high  x-cmcc-usermeta-metakey2:none  Content-Length: 914087  Authorization: CMCC TIKVJ979BgZHT9826KJP:lkgsdrili87vkjn9o8dvL9SsO6Ak | PUT\n  jn9o098hBGZHT94BqNli87vk\n  text/plain\n  Mon, 7 Feb 2010 21:30:00 +0000\n  x-cmcc-usermeta-metakey1:high\n  x-cmcc-usermeta-metakey2:none\n  /mycontainer/finance.txt |
| Get /mycontainer/baby.jpg  Host: obs.chinamobile.com  Date: Sun, 6 Feb 2011 21:30:00 +0000  x-cmcc-date: Sun, 6 Feb 2011 13:30:00 +0000  Authorization: CMCC TIKVJ979BgZHT9826KJP:xXjDGYUmKxnwqr5KXNPGldn5LbA= | Get\n  \n  \n  x-cmcc-date: Sun, 6 Feb 2011 13:30:00 +0000\n  /mycontainer/baby.jpg |

### 容器操作接口描述

用户发起容器操作前必须要有合法的签名密钥标识和签名密钥。

#### 获取用户所有容器

获取用户的所有容器，包括每个容器的相关信息。操作方法采用GET方法，操作目标资源为“/”，下面是一个获取用户容器列表的请求消息，返回的容器列表以字典序排序。

GET */* HTTP/1.1

Host: *obs.chinamobile.com*

Date: *Wed, 08 Aug 2012 07:35:00 GMT*

Authorization: *CMCC bf2529f47e655e31365a:gy6GnBjw9/+3N9sfCI/K90G5RCo=*

对象存储系统通过签名密钥标识可以获取用户ID，请求消息认证通过后对象存储系统将该用户ID下面的容器列表返回给用户。

#### 创建容器

创建的容器必须符合容器的命名规则，并全局唯一，容器的命名规则参见2.1章节。操作方法采用PUT方法，操作目标资源为“/ContainerName”，其中ContainerName为容器名称。创建一个名称为ContainerName的容器的请求消息如下：

PUT */ContainerName* HTTP/1.1

Host: *obs.chinamobile.com*

Date: *Wed, 08 Aug 2012 07:35:00 GMT*

Authorization: *CMCC bf2529f47e655e31365a:gy6GnBjw9/+3N9sfCI/K90G5RCo=*

创建容器的时候可以同时设置容器的访问控制模式，通过消息头x-cmcc-acl添加，下面的请求消息将容器ContainerName的访问控制模式设置为匿名用户可读。常见的访问控制模式访问控制模式参见4.3章节。

PUT */ContainerName* HTTP/1.1

Host: *obs.chinamobile.com*

Date: *Wed, 08 Aug 2012 07:35:00 GMT*

Authorization: *CMCC bf2529f47e655e31365a:gy6GnBjw9/+3N9sfCI/K90G5RCo=*

"x-cmcc-acl":"anonymous-read"

用户创建桶的最大个数由配置文件中参数决定，配置文件中该参数一般默认是10。

#### 删除容器

删除容器前需要确认容器为空，如果容器中还存在对象删除会失败，且只有容器所属人有权限删除容器。操作方法采用DELETE方法，操作目标资源为“/ContainerName”，其中ContainerName为容器名称。删除容器ContainerName的请求消息如下：

DELETE */ContainerName* HTTP/1.1

Host: *obs.chinamobile.com*

Date: *Wed, 08 Aug 2012 07:35:00 GMT*

Authorization: *CMCC bf2529f47e655e31365a:gy6GnBjw9/+3N9sfCI/K90G5RCo=*

#### 设置容器权限

用户设置容器的访问控制列表（ACL）。只有对容器有WRITE\_ACP访问权限的用户才能进行该操作（目前仅支持桶的owner有READ\_ACP,WRITE\_ACP权限）。操作方法采用PUT操作，操作目标资源由“/ContainerName?acl”组成的URI表示，其中“ContainerName”为容器名称，“acl”指明本操作目标为访问控制列表。下面是设置容器ContainerName访问控制列表的请求消息，消息中需包含容器的拥有者，访问控制列表中可以设置多个访问用户，每个访问用户权限定义包含用户ID和用户权限。可以对系统中存在的用户（包括匿名用户X-CMCC-ANONYMOUS-USERS和注册用户X-CMCC-REGISTED-USERS）赋权限：READ, WRITE, READ|WRITE；ACl文件大小限制为2kb，;桶的owner可以对自己赋权限，比如赋WRITE权限，但对owner的权限操作不保存在获取容器权限的权限列表中；Owner、 grantee 和permission 都是必选项，且判断owner信息正确与否，判断grantee是否存在，判断permission取值是否符合定义值。可以同时对一个用户put多个权限，将以列表中最后一个权限为准。权限定义参见4.1章节。

PUT */ContainerName?acl* HTTP/1.1

Host: *obs.chinamobile.com*

Date: *Wed, 08 Aug 2012 07:35:00 GMT*

Authorization: *CMCC bf2529f47e655e31365a:gy6GnBjw9/+3N9sfCI/K90G5RCo=*

<AccessControlPolicy>

<Owner>

<UserID>owner</UserID>

</Owner>

<AccessControlList>

<Grant>

<GranteeID>

<ID>ID</ID>

</GranteeID>

<Permission>Permission</Permission>

</Grant>

</AccessControlList>

</AccessControlPolicy>

#### 获取容器权限

用户获取容器的访问权限控制列表（ACL）。只有对容器有READ\_ACP访问权限的用户才能获取结果（目前仅支持桶的owner有READ\_ACP,WRITE\_ACP权限）。操作方法采用GET操作，操作目标资源由“/ContainerName?acl”组成的URI表示，其中“ContainerName”为容器名称，“acl”指明本操作的目标为访问控制列表。下面是一个获取容器ContainerName访问控制列表的请求消息：

GET */ContainerName?acl* HTTP/1.1

Host: *obs.chinamobile.com*

Date: *Wed, 08 Aug 2012 07:35:00 GMT*

Authorization: *CMCC bf2529f47e655e31365a:gy6GnBjw9/+3N9sfCI/K90G5RCo=*

#### 获取容器中对象列表

GetBucketAcl结果与最后一次PutBucketAcl的值一致，如果没有设置其他用户权限，则响应消息里仅显示owner信息。

对桶有读权限的用户获取容器中对象列表，并可通过输入参数（支持marker，prefix，maxResults为过滤参数）对结果集合进行筛选。此操作每次返回指定数量的以字典序排序的对象结果集，但最多不能超过200条(返回的最多条数可通过配置文件来配置，默认是200条)。只有对容器有读访问权限的用户才能获取结果。操作方法采用GET操作，操作目标资源为“/ContainerName”，其中ContainerName为容器名称。输入参数包括prefix，delimiter，marker和maxResult，这几个参数都是可选参数。

* prefix：此参数规定了返回结果集中对象URI的前缀
* maxResults：此参数规定了返回结果集的对象数量上限。如果满足参数的结果集对象数量超过maxResults值，超出的部分将会被截断，同时在返回体中设置截断标签isTruncated为true。如果maxResults取值为0或者负数返回全部对象列表。
* marker：此参数指定了对象URI的起始值（marker类似一个基于字符串类型的游标）。对于一次不能返回所有符合查询条件的请求（由于受到了maxResults的限制），客户端可以通过设置marker为上次调用请求的最后一个对象ID值，使得该次调用请求从上次调用请求的最后一个对象ID值的下一个位置进行继传。
* delimiter：此参数为分隔符，返回结果集以delimiter字符作为终止符来过滤。结果集查找中没有找到delimiter的对象作为对象列表返回，找到delimiter的对象，将对象的前缀提取出来作为CommonPrefix来返回。

下面是一个获取容器ContainerName对象列表的请求消息：

GET */ContainerName*?prefix=*prefix*&maxResults=*maxResults*&marker=*marker* &delimiter=/HTTP/1.1

Host: *obs.chinamobile.com*

Date: *Wed, 08 Aug 2012 07:35:00 GMT*

Authorization: *CMCC bf2529f47e655e31365a:gy6GnBjw9/+3N9sfCI/K90G5RCo=*

容器如果能被匿名用户读访问，则也能被注册用户访问。匿名用户访问注册用户有读权限的容器会失败。

### 对象操作接口描述

用户发起对象操作前必须要有合法的签名密钥标识和签名密钥。

#### 创建对象

根据指定的URI创建一个对象，对象存储系统中的对象必须属于一个特定的容器，用户创建对象前必须对该对象所属容器有写权限。对象有key，数据和元数据信息，对象的key在桶中是唯一的（多次上传同名的对象，后来上传的会覆盖之前上传的）。创建对象操作方法为PUT方法，操作目标资源为“/ContainerName/ObjectName”，其中ContainerName为容器名称，ObjectName为对象名称。下面是在容器ContainerName中创建对象ObjectName的请求消息：

PUT */ContainerName/ObjectName* HTTP/1.1

Host: *obs.chinamobile.com*

Date: *Wed, 08 Aug 2012 07:35:00 GMT*

Authorization: *CMCC bf2529f47e655e31365a:gy6GnBjw9/+3N9sfCI/K90G5RCo=*

创建对象的时候可以同时设置对象的访问控制模式，通过消息头x-cmcc-acl添加，常见的访问控制模式的定义参见4.3章节。下面的请求消息是创建对象ObjectName时，同时设置对象的访问控制模式为匿名用户可读。

PUT */ContainerName/ObjectName* HTTP/1.1

Host: *obs.chinamobile.com*

Date: *Wed, 08 Aug 2012 07:35:00 GMT*

Authorization: *CMCC bf2529f47e655e31365a:gy6GnBjw9/+3N9sfCI/K90G5RCo=*

"x-cmcc-acl":"anonymous-read"

创建对象的时候还可以同时设置对象的元数据，用户定义元数据参见3.1.2.2章节。下面的请求消息是创建对象ObjectName时，同时设置了一组元数据，元数据名称为key1，值为value1。

PUT */ContainerName/ObjectName* HTTP/1.1

Host: *obs.chinamobile.com*

Date: *Wed, 08 Aug 2012 07:35:00 GMT*

Authorization: *CMCC bf2529f47e655e31365a:gy6GnBjw9/+3N9sfCI/K90G5RCo=*

" x-cmcc-usermeta-key1 ":"value1"

通过该操作上传对象大小限制为5GB，超过5GB需要用MultiPart上传，元数据的大小限制为2KB。

#### 删除对象

删除指定的URI的对象，用户删除对象前必须对该对象所属的容器具有写权限。操作方法采用DELETE操作，操作目标资源为“/ContainerName/ObjectName”，其中ContainerName为容器名称，ObjectName为对象名称。下面是删除容器ContainerName中对象ObjectName的请求消息：

DELETE */ContainerName/ObjectName* HTTP/1.1

Host: *obs.chinamobile.com*

Date: *Wed, 08 Aug 2012 07:35:00 GMT*

Authorization: *CMCC bf2529f47e655e31365a:gy6GnBjw9/+3N9sfCI/K90G5RCo=*

对桶有WRITE权限的用户可以删除该桶里的对象，包括该用户上传的和别的用户上传的对象；

#### 获取对象

根据指定的URI读取一个对象的数据，用户获取对象前必须对该对象具有读权限。操作方法采用GET方法，操作目标资源为“/ContainerName/ObjectName”，其中ContainerName为容器名称，ObjectName为对象名称。响应消息中返回对象数据长度，类型，具体数据。该操作支持参数If-Match/If-None-Match,If-Modified-Since/If-Unmodified-Since。对象的Etag如果与If-Match取值相同，则读取对象成功，如果不同返回”Precondition-Failed”;对象的修改时间如果大于If-Modified-Since则读取对象成功，反之则返回失败。下面是获取容器ContainerName中对象ObjectName的请求消息：

GET */ContainerName/ObjectName* HTTP/1.1

Host: *obs.chinamobile.com*

Date: *Wed, 08 Aug 2012 07:35:00 GMT*

Authorization: *CMCC bf2529f47e655e31365a:gy6GnBjw9/+3N9sfCI/K90G5RCo=*

Range:bytes=start-end

Range表示获取对象的字节大小范围，需要满足start>=0, end>=start,当end超过对象的长度大小时返回对象的最大长度。

#### 获取对象属性

根据指定的URI读取某个对象的属性值，获取对象属性前必须对该对象具有读权限。操作方法有两种，一种是采用GET方法，操作目标资源为“/ContainerName/ObjectName?meta”。另外一种是采用HEAD方法，操作目标资源为“/ContainerName/ObjectName”。其中ContainerName为容器名称，ObjectName为对象名称，meta表示对象属性。下面是获取容器ContainerName中对象ObjectName属性的请求消息，有两种方法：

方法一：使用GET方法。

GET */ContainerName/ObjectName?meta* HTTP/1.1

Host: *obs.chinamobile.com*

Date: *Wed, 08 Aug 2012 07:35:00 GMT*

Authorization: *CMCC bf2529f47e655e31365a:gy6GnBjw9/+3N9sfCI/K90G5RCo=*

方法二：使用HEAD方法。

HEAD */ContainerName/ObjectName* HTTP/1.1

Host: *obs.chinamobile.com*

Date: *Wed, 08 Aug 2012 07:35:00 GMT*

Authorization: *CMCC bf2529f47e655e31365a:gy6GnBjw9/+3N9sfCI/K90G5RCo=*

#### 设置对象属性

根据指定的URI设置该对象的用户自定义属性值，设置对象属性前必须并对该对象的容器具有写权限。操作方法为PUT方法，操作目标资源为“/ContainerName/ObjectName?meta”，其中ContainerName为容器名称，ObjectName为对象名称，meta表示对象属性。下面是设置容器ContainerName中对象ObjectName属性的请求消息，消息中设置了一组元数据，元数据名称为key1，值为value1。用户可通过消息头x-cmcc-usermeta-\*方式写入对象元数据，也可在上传对象的时候写入元数据；关键字需严格匹配，否则自动过滤掉；Meta值最大为2kb。

PUT */ContainerName/ObjectName?meta* HTTP/1.1

Host: *obs.chinamobile.com*

Date: *Wed, 08 Aug 2012 07:35:00 GMT*

Authorization: *CMCC bf2529f47e655e31365a:gy6GnBjw9/+3N9sfCI/K90G5RCo=*

" x-cmcc-usermeta-key1 ":"value1"

1. 对桶有WRITE权限的用户可以桶里的对象进行PutObjectMeta（写元数据）操作
2. 用户可通过消息头x-cmcc-usermeta-\*方式写入对象元数据，也可在上传对象的时候写入元数据；关键字需严格匹配，否则自动过滤掉。
3. Meta值最大为2kb。

#### 获取对象权限

获取对象访问控制列表的操作，返回信息包含指定对象的访问控制列表信息。用户必须拥有对指定对象READ\_ACP的权限（目前仅支持对象的owner有READ\_ACP,WRITE\_ACP的权限），才能执行获取对象ACL的操作。操作方法采用GET操作，操作目标资源为“/ContainerName/ObjectName?acl”，其中ContainerName为容器名称，ObjectName为对象名称，acl表示访问控制列表。下面是获取容器ContainerName中对象ObjectName的访问控制列表的请求消息：

GET */ContainertName/ObjectName?acl* HTTP/1.1

Host: *obs.chinamobile.com*

Date: *Wed, 08 Aug 2012 07:35:00 GMT*

Authorization: *CMCC bf2529f47e655e31365a:gy6GnBjw9/+3N9sfCI/K90G5RCo=*

1. 该操作读取的acl列表与之前PutObjectACL里的一致
2. Content-Length是指acl文件的长度

#### 设置对象权限

设置对象访问控制列表的操作。用户必须拥有对指定对象WRITE\_ACP的权限，才能执行设置对象ACL的操作（目前仅支持对象的owner有READ\_ACP,WRITE\_ACP权限）。。操作方法采用PUT操作，操作目标资源为“/ContainerName/ObjectName?acl”，其中ContainerName为容器名称，ObjectName为对象名称，acl表示访问控制列表。下面是设置容器ContainerName中对象ObjectName的访问控制列表的请求消息，消息中需包含对象的拥有者，访问控制列表中可以设置多个访问用户，每个访问用户权限定义包含用户ID和用户权限。权限定义参见4.1章节。可参照PutBucketAcl操作。

1. 目前仅支持对象owner对object进行写acl列表操作
2. 可以对系统里存在的用户进行赋权限操作。Acl文件要按照特定的格式，其中的每一项都是必选项，信息不能有误，acl内容大小限制为2kb。
3. 可以对单个用户或者整个用户组赋权限：所有注册用户或者所有匿名用户 CMCC-OBS-ALL-REGISTED-USERS，CMCC-OBS-ANONYMOUS-USERS。

PUT*/Container/ObjectName?acl* HTTP/1.1

Host: *obs.chinamobile.com*

Date: *Wed, 08 Aug 2012 07:35:00 GMT*

Authorization: *CMCC bf2529f47e655e31365a:gy6GnBjw9/+3N9sfCI/K90G5RCo=*

<AccessControlPolicy>

<Owner>

<UserID>ID</UserID>

</Owner>

<AccessControlList>

<Grant>

<GranteeID>

<ID>ID</ID>

</GranteeID>

<Permission>Permission</Permission>

</Grant>

</AccessControlList>

</AccessControlPolicy>

### MultiPart上传

oNest支持将对象分成多个部分，并行上传。如果对象超过了5G就必须采用MultiPart上传，上传最大大小为5T。

#### 初始化MultiPart上传

用户使用MultiPart上传前需要先初始化MultiPart，操作方法采用POST方法，操作目标资源为“/bucketname/objectname？uploads”，其中“bucketname”是容器名称，“objectname”是对象名称，输入参数uploads表示MultiPart上传。用户需要对容器“bucketname”具有写权限，下面是一个初始化MulitPart上传的操作请求。

POST /bucketname/objectname?uploads HTTP/1.1

Host: obs.chinamobile.com

Date: Wed, 12 Oct 2009 17:50:00 GMT

Authorization: CMCC 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=

Content-Type: text/plain

Content-Length: 0

初始化成功，对象存储系统会在响应消息中返回给用户一个uploadID。

#### 上传Part数据

根据指定的URI 进行part 数据上传，操作方法采用POST方法，操作目标资源为“/bucketname/objectname”，其中“bucketname”是容器名称，“objectname”是对象名称，参数“uploadId”是是在初始化MultPart上传获取到的，参数“partNumber”指上传Part的序列号。用户需要是进行part 上传初始化的用户，下面是上传两个Part数据的操作请求，第一个Part的序列号是1，第二个Part的序列号是2。

PUT /bucketname/objectname? uploadId=asdfadfasdfdasdfd&partNumber=1 HTTP/1.1

Host: obs.chinamobile.com

Date: Wed, 12 Oct 2009 17:50:00 GMT

Authorization: CMCC 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=

Content-Type: text/plain

Content-Length: 11434

[11434 bytes of object data]

PUT /bucketname/objectname? uploadId=asdfadfasdfdasdfd&partNumber=2 HTTP/1.1

Host: obs.chinamobile.com

Date: Wed, 12 Oct 2009 17:50:00 GMT

Authorization: CMCC 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=

Content-Type: text/plain

Content-Length: 11434

[11434 bytes of object data]

请求消息的body部分是Part的数据。

#### 完成MultiPart上传

根据指定的URI完成指定对象的多part 上传操作，操作方法采用POST方法，操作目标资源为“/bucketname/objectname”，其中“bucketname”是容器名称，“objectname”是对象名称，参数“uploadId”是在初始化MultPart上传获取到的。用户需要是进行part上传初始化的用户，下面是完成MultiPart上传的操作请求，消息中指定完成了序列号为1和序列号为2的两个Part。

POST /bucketname/objectname? uploadId=asdfadfasdfdasdfd HTTP/1.1

Host: obs.chinamobile.com

Date: Wed, 12 Oct 2009 17:50:00 GMT

Authorization: CMCC 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=

Content-Type: text/plain

Content-Length: xxx

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<CompleteMultipartUpload>

<Part>

<PartNumber>1</PartNumber>

<ETag>b2ad2b9d86219e71598f7768de524423</ETag>

</Part>

<Part>

<PartNumber>2</PartNumber>

<ETag>9a274b15f3b4723dd24ae9b7a1e570f3</ETag>

</Part>

</CompleteMultipartUpload>

#### 取消MultiPart上传

根据指定的URI取消指定对象的多part 上传操作，操作方法采用DELETE方法，操作目标资源为“/bucketname/objectname”，其中“bucketname”是容器名称，“objectname”是对象名称，参数“uploadId”是在初始化MultPart上传获取到的。用户需要是进行part上传初始化的用户或者容器的所有者，下面是取消MultiPart上传的操作请求。

DELETE /bucketname/objectname? uploadId=asdfadfasdfdasdfd HTTP/1.1

Host: obs.chinamobile.com

Date: Wed, 12 Oct 2009 17:50:00 GMT

Authorization: CMCC 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=

Content-Type: text/plain

Content-Length: 0

#### 列出MultiPart上传对象的Part列表

根据指定的URI 列出使用MultiPart Upload模式上传，且未完成的指定Object的part列表，操作方法采用GET方法，操作目标资源为“/bucketname/objectname”，其中“bucketname”是容器名称，“objectname”是对象名称，参数“uploadId”是在初始化MultPart上传获取到的，参数“part-number-marker”是Part的起始值，参数“max\_parts”是返回结果集的part数量上限。用户需要是进行part 上传初始化的用户或者容器的所有者，下面是列出MultiPart上传对象的Part列表的操作请求。

GET /bucketname/objectname? uploadId=asdfadfasdfdasdfd

&part-number-marker=someMarker&max\_parts =num HTTP/1.1

Host: obs.chinamobile.com

Date: Wed, 12 Oct 2009 17:50:00 GMT

Authorization: CMCC 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=

Content-Type: text/plain

Content-Length: xxx

#### 获取容器中未上传完对象列表

根据指定的URI获取容器中使用MultiPart Upload模式上传，且未完成的Object列表。操作方法采用GET方法，操作目标资源为“/bucketname”，其中“bucketname”是容器名称，参数uploads表示MultiPart上传，参数“key-marker”指定了对象URI的起始值，参数“upload-id-marker” 指定了对象UploadID的起始值，参数“delimiter”是分隔符，参数“prefix=”规定了返回结果集中对象URI的前缀，参数“max-uploads”规定了返回结果集的对象数量上限。下面是获取容器“bucketname”未上传完对象列表的操作请求：

GET /bucketname?uploads&key-marker=somemarker&upload-id-marker= somemarker&delimiter=/&prefix=europe/max-uploads=num HTTP/1.1

Host: obs.chinamobile.com

Date: Wed, 12 Oct 2009 17:50:00 GMT

Authorization: CMCC 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=

Content-Type: text/plain

Content-Length: xxx

### 统计接口描述

#### 统计规则

**规则一**

成功的操作，是都进行记录的。失败时在什么情况下进行记录，可以转换成操作进行到哪一步之后进行记录，下面进行说明。

统计涉及到的操作有Putobject、Getobject、Delobject、Complete Multipart Upload。

操作的一般性顺序：

1, 用户认证。

2、检查bucket是否存在，以及用户是否挂起。

3、检查object或者uploadingObj 是否存在（Putobject 不需检查）。

4、检查用户是否有操作权限。

由于统计要填入有意义的BucketName objectName UserName。

如果要形成一条有意义的记录，至少需要知道操作中有价值的bucket 和object。

所以规则如下：

Putobject时，向一个不存在的bucket 进行put 不做记录。Bucket确认存在后发生错误，进行统计。

Getobject 时，get 一个不存在的object 不做记录。Bucket和object 都确定存在后发生错误，进行记录。

Delobject 时，删除一个不存在的object 不做记录。Bucket和object 都确定存在后发生错误，进行记录。

Complete Multipart Upload 时，如果不存在一个对应的uploadingObj ，不做记录。Bucket和uploadingObj都确定存在后发生错误，进行记录。

注：由于认证是在bucket 检查之前。所以认证错误是不记录的。

**规则二**

成功则都进行记录，失败只记录产生了流量的操作。

Putobject 失败时，查看上传流量的大小。如果大于0 进行记录。

Getobject 失败时，查看下载流量的大小，如果大于0，进行记录。

Delobject失败时，不记录。

Complete Multipart Upload 失败时，不记录。

**其他规则**

UserName

现在Log 中的owner 都是设置bucket Owner。流量都是记录在bucket Owner 下。

ObjectSize

Log 中的ObjectSize 在成功时，记录对象的实际大小。失败时记录传输的数据大小。

#### 获取用户空间统计

获取指定用户的空间统计，包括用户的使用空间，用户的容器个数和对象个数。操作方法采用GET方法，操作目标资源为“/rest/”，参数userstat表示用户空间统计，参数user指定用户ID，下面是获取用户空间统计的请求消息：

GET /rest/?userstat&user=UserID HTTP/1.1

Host: *obs.chinamobile.com*

Date: *Wed, 08 Aug 2012 07:35:00 GMT*

Authorization: *CMCC bf2529f47e655e31365a:gy6GnBjw9/+3N9sfCI/K90G5RCo=*

#### 获取容器统计

获取指定容器的统计，包括容器的使用空间，容器内的对象个数。操作方法采用GET方法，操作目标资源为“/rest/”，参数bucketstat表示容器统计，参数bucket指定容器名称，下面是获取用容器统计的请求消息：

GET /rest/?bucketstat&bucket=ContainerName HTTP/1.1

Host: *obs.chinamobile.com*

Date: *Wed, 08 Aug 2012 07:35:00 GMT*

Authorization: *CMCC bf2529f47e655e31365a:gy6GnBjw9/+3N9sfCI/K90G5RCo=*

# 运营管理平台

云存储提供运营管理平台，采用soap协议通讯，支持用户的管理和用户的访问凭证管理。

## 接口定义

### 协议形式

通信双方采用一问一答的通讯机制，即一次请求对应于一次应答。

一般情况，通信双方之间的消息发送后等待T秒后未收到响应，应立即重发，再连续发送N-1次后仍未得到响应则停发。现阶段建议取值为：T=60秒，N=3。

### 日期与时间表示规范

1. 日期与时间

|  |  |
| --- | --- |
| 年 | yyyy |
| 月 | mm |
| 日 | dd |
| 时 | hh(24小时制) |
| 分 | nn |
| 秒 | ss |

1. 日期时间分隔符定义

|  |  |
| --- | --- |
| 日期分隔符 | - |
| 时间分隔符 | : |
| 日期与时间分隔符 | 一个半角空格 |

1. 示例

例如：2005年12月9日 21时01分39秒表示为： 2005-12-09 21:01:39

### 错误码

|  |  |  |
| --- | --- | --- |
| Code | Result Code | 描述 |
| 0 | OK | 操作成功 |
| 500 | Internal Error | 服务器内部错误 |
| 501 | Authentication Failure | 认证失败 |
| 502 | User Exists | 用户已经存在 |
| 503 | Access Key Exists | 签名密钥标识已经存在 |
| 504 | User Not Exists | 用户不存在 |
| 505 | User Disabled | 用户暂停 |
| 507 | UserID Exists | 用户ID已经存在 |
| 520 | Invalid Capacity | 容量配额无效 |
| 521 | Invalid User ID | 用户ID无效 |
| 522 | Invalid Access Key | 签名密钥标识无效 |
| 523 | Invalid Secret Key | 签名密钥无效 |
| 524 | Invalid Status | 用户状态无效 |
| 525 | Invalid ModifyFlag | 修改标识无效 |
| 526 | Invalid MonitorFlag | 监控标识无效 |
| 553 | Execeed AccessKey Max Number | 签名密钥超过最大个数 |

### 接口描述

**SOAP- Common Elements请求头**

鉴权访问参数包括四部分内容： Timestamp、TransactionID:

|  |  |  |  |
| --- | --- | --- | --- |
| 结构 | 参数名 | 参数类型 | 说明 |
| Authorization | Timestamp | String | 系统当前的时间戳，时间格式见5.1.2节,接受端N分钟内判断此请求有效性。超过N分钟则认为无效。N可配置。 |
| TransactionID | String | 资源管理平台使用被访问方提供的SOAP协议，每次发起请求必须参数保证消息唯一。  TransactionID分成三部分，第一部分长为10位，表示资源管理平台编码，第二部分长为14位即为Timestamp；第三部分长为8位，由0开始，循环进位，直到进位满了之后再清零，重新开始计数。接收方在返回应答时，应答的序列号必须和相对应的请求的序列号相同。序列号用来唯一标示一个消息。 |

**SOAP- Common Elements响应头**

|  |  |  |  |
| --- | --- | --- | --- |
| 结构 | 参数名 | 参数类型 | 说明 |
| Response | TransactionID | String | 资源管理平台使用被访问方提供的SOAP协议，每次发起请求必须参数保证消息唯一。TransactionID分成三部分，第一部分长为10位，表示资源管理平台编码，第二部分长为14位即为Timestamp；第三部分长为8位，由0开始，循环进位，直到进位满了之后再清零，重新开始计数。接收方在返回应答时，应答的序列号必须和相对应的请求的序列号相同。序列号用来唯一标示一个消息。 |
| ResultCode | String | 请求返回码 |

### 请求字段说明

UserID：可以由大小写字母，数字,“.”,“\_”，“-”，“@”构成，长度范围为[3，31]。

AccessID：可以由大小写字母，数字,“\_”,“-”构成，长度范围为[3，128]。

AccessSecretKey：可以由大小写字母，数字,“\_”,“-”构成，长度范围为[3，128]。

Capacity范围：[0，8796093022207L]

其他字段在下面各请求消息中有说明。

## 签约对象存储服务（CreateObsUser）

**接口功能**

该接口用于为用户签约对象存储服务。

**接口说明**

|  |  |  |
| --- | --- | --- |
| 消息名 | 消息类型 | 消息方向 |
| CreateObsUserReq | Request | 资源池系统🡪基于对象的云存储 |
| CreateObsUserResp | Response | 基于对象的云存储🡪资源池系统 |

CreateObsUserReq

| 名称 | 类型 | 描述 | 必须/可选 |
| --- | --- | --- | --- |
| UserID | 字符串 | 用户的标识 | 必选 |
| Status | 字符串 | 用户状态：  enable – 正常；  disable – 暂停 | 必须 |
| AccessID | 字符串 | 签名密钥标识 | 必选 |
| AccessSecretKey | 字符串 | 签名的密钥 | 必选 |
| Capacity | 数字 | 容量配额，单位为MByte | 必须 |
| MonitorFlag | BOOL | 是否对该用户打开监控功能，不携带缺省为FALSE | 必选 |

CreateObsUserResp

| 名称 | 类型 | 描述 | 必须/可选 |
| --- | --- | --- | --- |
| Code | 数字 | 操作返回码 | 必选 |

**接口消息**

请求消息CreateObsUserReq

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ones="http://www.oNest.com/oNest">

<soapenv:Header/>

<soapenv:Body>

<ones:CreateObsUserReq>

<ones:auth>

<ones:timestamp>2012-11-28 16:35:41</ones:timestamp>

<ones:transactionID>00000000001333185600000000000112</ones:transactionID>

</ones:auth>

<ones:reqUserCreate>

<ones:userID>wy1128</ones:userID>

<ones:status>enable</ones:status>

<ones:accessID>wy1</ones:accessID>

<ones:accessSecretKey>123</ones:accessSecretKey>

<ones:capacity>100</ones:capacity>

<ones:monitorFlag>true</ones:monitorFlag>

</ones:reqUserCreate>

</ones:CreateObsUserReq>

</soapenv:Body>

</soapenv:Envelope>

响应消息CreateObsUserResp

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

<soap:Body>

<CreateObsUserResp xmlns="http://www.oNest.com/oNest">

<transactionID>00000000001333185600000000000112</transactionID>

<resultCode>OK</resultCode>

<respUserCreate>

<code>0</code>

</respUserCreate>

</CreateObsUserResp>

</soap:Body>

</soap:Envelope>

## 取消对象存储服务（DeleteObsUser）

**接口功能**

该接口用于为用户取消已申请到的对象存储服务。即根据用户ID删除用户，当用户删除后，该用户的所有AccessID一并删除。

**接口说明**

|  |  |  |
| --- | --- | --- |
| 消息名 | 消息类型 | 消息方向 |
| DeleteObsUserReq | Request | 资源池系统🡪基于对象的oNest |
| DeleteObsUserResp | Response | 基于对象的oNest🡪资源池系统 |

DeleteObsUserReq

| 名称 | 类型 | 描述 | 必须/可选 |
| --- | --- | --- | --- |
| UserID | 字符串 | 用户的标识 | 必选 |

DeleteObsUserResp

| 名称 | 类型 | 描述 | 必须/可选 |
| --- | --- | --- | --- |
| Code | 数字 | 操作返回码 | 必选 |

**接口消息**

请求消息DeleteObsUserReq

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ones="http://www.oNest.com/oNest">

<soapenv:Header/>

<soapenv:Body>

<ones:DeleteObsUserReq>

<ones:auth>

<ones:timestamp>2012-11-28 17:45:41</ones:timestamp>

<ones:transactionID>00000000001333185600000000000113</ones:transactionID>

</ones:auth>

<ones:reqUserDel>

<ones:userID>wy1128</ones:userID>

</ones:reqUserDel>

</ones:DeleteObsUserReq>

</soapenv:Body>

</soapenv:Envelope>

响应消息DeleteObsUserResp

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

<soap:Body>

<DeleteObsUserResp xmlns="http://www.oNest.com/oNest">

<transactionID>00000000001333185600000000000113</transactionID>

<resultCode>OK</resultCode>

<respUserDelete>

<code>0</code>

</respUserDelete>

</DeleteObsUserResp>

</soap:Body>

</soap:Envelope>

## 变更对象存储服务（ModifyObsUser）

**接口功能**

该接口用于为用户变更已申请到的对象存储服务。可以变更用户的存储服务状态，配额等参数。根据ModifyFlag标识项取值决定修改哪个参数，其他参数项取值忽略不计，仅修改对应参数项。比如修改status时，status就是必选项，取值必须符合要求；其他项取值就忽略不计，可以不带这些参数。更改项的标识值可以叠加，比如修改状态对应”1”, 修改配额对应”2”,如果取值”3”, 那么就是修改这两项。

**接口说明**

|  |  |  |
| --- | --- | --- |
| 消息名 | 消息类型 | 消息方向 |
| ModifyObsUserReq | Request | 资源池系统🡪基于对象的oNest |
| ModifyObsUserResp | Response | 基于对象的oNest🡪资源池系统 |

ModifyObsUserReq

| 名称 | 类型 | 描述 | 必须/可选 |
| --- | --- | --- | --- |
| UserID | 字符串 | 用户的标识 | 必选 |
| ModifyFlag | 数字 | 本次操作变更字段的标识：  1： Status；  2： Capacity；  4：MonitorFlag； | 必选 |
| Status | 字符串 | 用户状态：  enable – 正常；  disable – 暂停 | 可选 |
| Capacity | 数字 | 容量配额，单位为MByte | 可选 |
| MonitorFlag | BOOL | 是否对该用户打开监控功能 | 可选 |

ModifyObsUserResp

| 名称 | 类型 | 描述 | 必须/可选 |
| --- | --- | --- | --- |
| Code | 数字 | 操作返回码 | 必选 |

**接口消息**

请求消息ModifyObsUserReq

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ones="http://www.oNest.com/oNest">

<soapenv:Header/>

<soapenv:Body>

<ones:ModifyObsUserReq>

<ones:auth>

<ones:timestamp>2012-11-28 17:45:56</ones:timestamp>

<ones:transactionID>00000000001333185600000000000013</ones:transactionID>

</ones:auth>

<ones:reqUserMod>

<ones:userID>gl\_test1</ones:userID>

<ones:modifyFlag>3</ones:modifyFlag>

<ones:status>disable</ones:status>

<ones:capacity>11</ones:capacity>

<ones:monitorFlag>true</ones:monitorFlag>

</ones:reqUserMod>

</ones:ModifyObsUserReq>

</soapenv:Body>

</soapenv:Envelope>

响应消息ModifyObsUserResp

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

<soap:Body>

<ModifyObsUserResp xmlns="http://www.oNest.com/oNest">

<transactionID>00000000001333185600000000000013</transactionID>

<resultCode>OK</resultCode>

<respUserModify>

<code>0</code>

</respUserModify>

</ModifyObsUserResp>

</soap:Body>

</soap:Envelope>

## 查看对象存储服务（QueryObsUser）

**接口功能**

该接口用于查看用户已申请到的对象存储服务。

**接口说明**

|  |  |  |
| --- | --- | --- |
| 消息名 | 消息类型 | 消息方向 |
| QueryObsUserReq | Request | 资源池系统🡪基于对象的oNest |
| QueryObsUserResp | Response | 基于对象的oNest🡪资源池系统 |

QueryObsUserReq

| 名称 | 类型 | 描述 | 必须/可选 |
| --- | --- | --- | --- |
| UserID | 字符串 | 用户的标识 | 必选 |

QueryObsUserResp

| 名称 | 类型 | 描述 | 必须/可选 |
| --- | --- | --- | --- |
| Code | 数字 | 操作返回码 | 必选 |
| Status | 字符串 | 用户状态：  enable – 正常；  disable – 暂停 | 必须 |
| Capacity | 数字 | 容量配额，单位为MByte | 必须 |
| UsedCapacity | 数字 | 已使用容量，单位为MByte | 必选 |
| CreateObsUserTime | 字符串 | 用户签约对象存储服务的时间，GMT格式 | 必选 |
| MonitorFlag | BOOL | 是否对该用户打开监控功能 | 必选 |

**接口消息**

请求消息QueryObsUserReq

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ones="http://www.oNest.com/oNest">

<soapenv:Header/>

<soapenv:Body>

<ones:QueryObsUserReq>

<ones:auth>

<ones:timestamp>2012-11-28 17:50:56</ones:timestamp>

<ones:transactionID>00000000001333185600000000000013</ones:transactionID>

</ones:auth>

<ones:reqUserQuery>

<ones:userID>gl\_test1</ones:userID>

</ones:reqUserQuery>

</ones:QueryObsUserReq>

</soapenv:Body>

</soapenv:Envelope>

返回消息QueryObsUserResp

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

<soap:Body>

<QueryObsUserResp xmlns="http://www.oNest.com/oNest">

<transactionID>00000000001333185600000000000013</transactionID>

<resultCode>OK</resultCode>

<respUserQuery>

<code>0</code>

<capacity>11</capacity>

<createObsUserTime>2012-11-28 02:40:29</createObsUserTime>

<monitorFlag>true</monitorFlag>

<status>disable</status>

<usedCapacity>0</usedCapacity>

</respUserQuery>

</QueryObsUserResp>

</soap:Body>

</soap:Envelope>

## 增加用户认证信息（CreateObsAccessID）

**接口功能**

该接口用于增加用户认证信息。

AccessID不能重复，AccessID为整个系统所有用户共用，比如ID1为用户test1的AccessID，那么别的用户（或者test1自己）再创建AccessID为ID1时，就会提示该AccessID已存在；AccessSecretKey可以重复。

**接口说明**

|  |  |  |
| --- | --- | --- |
| 消息名 | 消息类型 | 消息方向 |
| CreateObsAccessIDReq | Request | 资源池系统🡪基于对象的oNest |
| CreateObsAccessIDResp | Response | 基于对象的oNest🡪资源池系统 |

CreateObsAccessIDReq

| 名称 | 类型 | 描述 | 必须/可选 |
| --- | --- | --- | --- |
| UserID | 字符串 | 用户的标识 | 必选 |
| AccessID | 字符串 | 签名密钥标识 | 必选 |
| AccessSecretKey | 字符串 | 签名密钥 | 必选 |

CreateObsAccessIDResp

| 名称 | 类型 | 描述 | 必须/可选 |
| --- | --- | --- | --- |
| Code | 数字 | 操作返回码 | 必选 |

**接口消息**

请求消息CreateObsAccessIDReq

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ones="http://www.oNest.com/oNest">

<soapenv:Header/>

<soapenv:Body>

<ones:CreateObsAccessIDReq>

<ones:auth>

<ones:timestamp>2012-11-28 17:57:56</ones:timestamp>

<ones:transactionID>00000000001333185600000000000013</ones:transactionID>

</ones:auth>

<ones:reqAccessIDCreate>

<ones:userID>gl\_test1</ones:userID>

<ones:accessID>IDg1</ones:accessID>

<ones:accessSecretKey>1234</ones:accessSecretKey>

</ones:reqAccessIDCreate>

</ones:CreateObsAccessIDReq>

</soapenv:Body>

</soapenv:Envelope>

响应消息请求消息CreateObsAccessIDResp

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

<soap:Body>

<CreateObsAccessIDResp xmlns="http://www.oNest.com/oNest">

<transactionID>00000000001333185600000000000013</transactionID>

<resultCode>OK</resultCode>

<respAccessIDCreate>

<code>0</code>

</respAccessIDCreate>

</CreateObsAccessIDResp>

</soap:Body>

</soap:Envelope>

## 删除用户认证信息（DeleteObsAccessID）

**接口功能**

该接口用于删除用户认证信息。

**接口说明**

|  |  |  |
| --- | --- | --- |
| 消息名 | 消息类型 | 消息方向 |
| DeleteObsAccessIDReq | Request | 资源池系统🡪基于对象的oNest |
| DeleteObsAccessIDResp | Response | 基于对象的oNest🡪资源池系统 |

DeleteObsAccessIDReq

| 名称 | 类型 | 描述 | 必须/可选 |
| --- | --- | --- | --- |
| UserID | 字符串 | 用户的标识 | 必选 |
| AccessID | 字符串 | 签名密钥标识 | 必选 |

DeleteObsAccessIDResp

| 名称 | 类型 | 描述 | 必须/可选 |
| --- | --- | --- | --- |
| Code | 数字 | 操作返回码 | 必选 |

**接口消息**

请求消息DeleteObsAccessIDReq

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ones="http://www.oNest.com/oNest">

<soapenv:Header/>

<soapenv:Body>

<ones:DeleteObsAccessIDReq>

<ones:auth>

<ones:timestamp>2012-11-28 17:57:56</ones:timestamp>

<ones:transactionID>00000000001333185600000000000013</ones:transactionID>

</ones:auth>

<ones:reqAccessIDDel>

<ones:userID>gl\_test1</ones:userID>

<ones:accessID>IDg1</ones:accessID>

</ones:reqAccessIDDel>

</ones:DeleteObsAccessIDReq>

</soapenv:Body>

</soapenv:Envelope>

响应消息DeleteObsAccessIDResp

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

<soap:Body>

<DeleteObsAccessIDResp xmlns="http://www.oNest.com/oNest">

<transactionID>00000000001333185600000000000013</transactionID>

<resultCode>OK</resultCode>

<respAccessIDDelete>

<code>0</code>

</respAccessIDDelete>

</DeleteObsAccessIDResp>

</soap:Body>

</soap:Envelope>

## 查询用户认证信息（QueryObsAccessID）

**接口功能**

该接口用于查询用户认证信息。

返回AccessID列表，按照AccessID的hash值排序。

**接口说明**

|  |  |  |
| --- | --- | --- |
| 消息名 | 消息类型 | 消息方向 |
| QueryObsAccessIDReq | Request | 资源池系统🡪基于对象的oNest |
| QueryObsAccessIDResp | Response | 基于对象的oNest🡪资源池系统 |

QueryObsAccessIDReq

| 名称 | 类型 | 描述 | 必须/可选 |
| --- | --- | --- | --- |
| UserID | 字符串 | 用户的标识 | 必选 |

QueryObsAccessIDResp

| 名称 | 类型 | 描述 | 必须/可选 |
| --- | --- | --- | --- |
| Code | 数字 | 操作返回码 | 必选 |
| AccessID | 字符串 | 签名密钥标识（可能是多个，以列表形式返回） | 必须 |

**接口消息**

请求消息QueryObsAccessIDReq

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ones="http://www.oNest.com/oNest">

<soapenv:Header/>

<soapenv:Body>

<ones:QueryObsAccessIDReq>

<ones:auth>

<ones:timestamp>2012-11-28 17:57:56</ones:timestamp>

<ones:transactionID>00000000001333185600000000000013</ones:transactionID>

</ones:auth>

<ones:reqAccessIDQuery>

<ones:userID>gl\_test1</ones:userID>

</ones:reqAccessIDQuery>

</ones:QueryObsAccessIDReq>

</soapenv:Body>

</soapenv:Envelope>

响应消息QueryObsAccessIDResp

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

<soap:Body>

<QueryObsAccessIDResp xmlns="http://www.oNest.com/oNest">

<transactionID>00000000001333185600000000000013</transactionID>

<resultCode>OK</resultCode>

<respAccessIDQuery>

<code>0</code>

<accessID>IDg1</accessID>

<accessID>cf3178c34157319e71af</accessID>

</respAccessIDQuery>

</QueryObsAccessIDResp>

</soap:Body>

</soap:Envelope>