

对象存储

(Object-Oriented Storage, OOS)

PHP SDK 开发者指南

中国电信股份有限公司

云计算分公司

目录

1 对象存储 (OOS)	1
2 使用条件.....	2
2.1 先决条件.....	2
2.2 获取 OOS 凭证.....	2
3 下载 PHP SDK.....	2
4 SDK 代码示例.....	2
4.1 Service 操作.....	2
4.1.1 GET Service (List Bucket)	2
4.1.2 GET Regions (仅 OOS 6.0 支持)	3
4.2 关于 Bucket 的操作.....	4
4.2.1 PUT Bucket.....	4
4.2.2 GET Bucket Location (仅 OOS 6.0 支持)	6
4.2.3 GET Bucket ACL.....	6
4.2.4 GET Bucket (List Objects)	7
4.2.5 DELETE Bucket.....	8
4.2.6 PUT Bucket Policy.....	8
4.2.7 GET Bucket Policy.....	10
4.2.8 DELETE Bucket Policy.....	10
4.2.9 PUT Bucket WebSite.....	11
4.2.10 GET Bucket WebSite.....	12
4.2.11 DELETE Bucket WebSite.....	12
4.2.12 List Multipart Uploads.....	13
4.2.13 PUT Bucket Logging.....	14
4.2.14 GET Bucket Logging.....	15
4.2.15 HEAD Bucket.....	15
4.2.16 PUT Bucket Lifecycle.....	16
4.2.17 GET Bucket Lifecycle.....	17
4.2.18 DELETE Bucket Lifecycle.....	17

4.2.19	PUT Bucket Accelerate.....	18
4.2.20	GET Bucket Accelerate.....	19
4.2.21	PUT Bucket cors.....	19
4.2.22	GET Bucket cors.....	20
4.2.23	DELETE Bucket cors.....	21
4.3	关于 Object 的操作.....	21
4.3.1	PUT Object.....	21
4.3.2	GET Object.....	23
4.3.3	Delete Object.....	24
4.3.4	PUT Object - Copy.....	24
4.3.5	Initial Multipart Upload.....	26
4.3.6	Upload Part.....	27
4.3.7	Complete Multipart Upload.....	29
4.3.8	Abort Multipart Upload.....	30
4.3.9	List Part.....	31
4.3.10	Copy Part.....	32
4.3.11	Delete Multiple Objects.....	32
4.3.12	生成共享链接.....	33
4.3.13	HEAD Object.....	34
4.4	关于 AccessKey 的操作.....	34
4.4.1	CreateAccessKey.....	35
4.4.2	DeleteAccessKey.....	35
4.4.3	UpdateAccessKey.....	36
4.4.4	ListAccessKey.....	36
4.5	错误处理.....	37
4.5.1	错误响应.....	37
4.5.2	错误码列表.....	38

1 对象存储 (OOS)

对象存储 (Object-Oriented Storage, OOS) 是中国电信为客户提供的一种海量、弹性、廉价、高可用的存储服务。客户只需花极少的钱就可以获得一个几乎无限的存储空间，可以随时根据需要调整对资源的占用，并只需为真正使用的资源付费。

OOS 提供了基于 Web 门户和基于 HTTP REST 接口两种访问方式，用户可以在任何地方通过互联网对数据进行管理和访问。OOS 提供的 REST 接口与 Amazon S3 兼容，因此基于 OOS 的业务可以与 Amazon S3 对接。您也可以通过 OOS 提供的 SDK 来调用 OOS 服务。


```

try {
    $bucketListInfo = $oosClient->listBuckets($options);
} catch (OosException $e) {
    $e->printException(__FUNCTION__);
    return;
}
print(__FUNCTION__ . ": OK" . "\n");
$bucketList = $bucketListInfo->getBucketList();
$owner = $bucketListInfo->getOwner();
foreach ($bucketList as $bucket) {
    print( $bucket->getName() . "\t" . $bucket->getCreatedate() . "\n");
}
}

```

4.1.2 GET Regions (仅 OOS 6.0 支持)

获取资源池中的索引位置和数据位置列表。

示例代码

```

function getRegions($oosClient)
{
    $options = array();

    try {
        $bucketRegions = $oosClient->getRegions($options);
        $metaRegions = $bucketRegions->getMetaRegions();
        $dataRegions = $bucketRegions->getDataRegions();
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        print(__FUNCTION__ . ": FAILED" . "\n");
        return;
    }

    print(__FUNCTION__ . ": OK" . "\n");
}

```

4.2 关于 Bucket 的操作

4.2.1 PUT Bucket

Put Bucket 用来创建一个 Bucket。只有已注册 OOS 账户才能创建 Bucket，匿名请求无效，创建 Bucket 的用户将是 Bucket 的拥有者。

Bucket 的命名方式如下：

- Bucket 名称必须全局唯一；
- Bucket 名称长度介于 3 到 63 字节之间；
- Bucket 名称只能由小写字母、数字、短横线 (-) 和点 (.) 组成；
- Bucket 名称可以由一个或者多个小节组成，小节之间用点 (.) 隔开，各个小节需要：
 - 必须以小写字母或者数字开始；
 - 必须以小写字母或者数字结束。
- Bucket 名称不能是 IP 地址形式（如 192.162.0.1）；
- Bucket 名称不能是一组或多组“数字.数字”的组合；
- Bucket 名称中不能包含双横线 (--)、双点 (..)、横线点 (-.) 和点横线 (.-)
- 不允许使用非法敏感字符，例如暴恐涉政相关信息等。

示例代码

OOS 5.0 PUT Bucket 示例：

```
function createBucket($oosClient, $bucket)
{
    try {
        $options = array();
        //本接口只能在版本5的资源池上使用，最后一个参数必须是null
        $oosClient->createBucket($bucket,
OosClient::OOS_ACL_TYPE_PUBLIC_READ_WRITE,$options,null);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
}
```

OOS 6.0 PUT Bucket 示例:

```

function createBucketWithLocation($oosClient, $bucket)
{
    try {
        $options = array();
        //当只有一个元数据和数据域时候，不能设置
metadataLocationConstraint 和 dataLocationConstraint,目前可用的有
ZhengZhou|ShenYang|ChengDu|WuLuMuQi|LanZhou|QingDao|GuiYang|LaSa|W
uHu|WuHan|ShenZhen
        //但是要根据用户权限调用 Service.getRegions 查看可用的资源池
        $metadataLocationConstraint = new
MetadataLocationConstraint("ZhengZhou");
        $type = "Specified";
        $locationList = array();
        $locationList["ShenYang"] = "ShenYang";
        $locationList["ChengDu"] = "ChengDu";
        $scheduleStrategy = "NotAllowed";
        $dataLocationConstraint = new
DataLocation($type,$locationList,$scheduleStrategy);
        $bucketDetailInfo = new BucketDetailInfo();
        $bucketDetailInfo->setMetaLocation($metadataLocationConstraint);
        $bucketDetailInfo->setDataLocation($dataLocationConstraint);

        //本接口只能在版本 6 的资源池上使用，必须提供 BucketDetailInfo
的对象
        $oosClient->createBucket($bucket,
OosClient::OOS_ACL_TYPE_PUBLIC_READ_WRITE,$options,$bucketDetailIn
fo);

```



```
} catch (OosException $e) {  
    $e->printException(__FUNCTION__);  
    return;  
}  
print(__FUNCTION__ . ": OK" . "\n");  
}
```

4.2.2 GET Bucket Location (仅 OOS 6.0 支持)

GET Bucket Location 操作用来获取 Bucket 的索引位置和数据位置，只有 Bucket 的所有者才能执行此操作。

示例代码

```
function getBucketLocation($oosClient, $bucket)  
{  
    try {  
        $options = array();  
        $bucketDetailInfo = $oosClient->getBucketLocation($bucket,$options);  
    } catch (OosException $e) {  
        $e->printException(__FUNCTION__);  
        return;  
    }  
    print(__FUNCTION__ . ": OK" . "\n");  
}
```

4.2.3 GET Bucket ACL

GET Bucket ACL 操作用来获取 Bucket 的 ACL 信息，用户必须对该 Bucket 有读权限。

示例代码

```
function getBucketAcl($oosClient, $bucket)
```

```

{
    try {
        $options = array();

        $bucketAcl = $oosClient->getBucketAcl($bucket,$options);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
}

```

4.2.4 GET Bucket (List Objects)

GET Bucket 操作返回 Bucket 中部分或者全部（最多 1000）的 object 信息。用户可以在请求元素中设置选择条件来获取 Bucket 中 object 的子集。

要执行该操作，需要对操作的 Bucket 拥有读权限。

示例代码

```

function listObjects($oosClient,$bucket)
{
    $listObjectsResult = null;
    try {
        $options = array();
        $options["max-keys"] = 40;
        $options["prefix"] = "doc/";
        $options["delimiter"] = "/";
        $options["marker"] = "doc/F";

        $listObjectsResult = $oosClient->listObjects($bucket,$options);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
}

```

```
$bucketName = $listObjectsResult->getBucketName();
}
```

4.2.5 DELETE Bucket

该操作用来执行删除 Bucket 的操作，但要求所有 Bucket 中的 object 都必须被删除。

示例代码

```
function deleteBucket($oosClient, $bucket)
{
    try {
        $options = array();

        $oosClient->deleteBucket($bucket,$options);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
}
```

4.2.6 PUT Bucket Policy

在 PUT 操作的 url 中加上 Policy，可以进行添加或修改 Policy 的操作。如果 Bucket 已经存在了 Policy，此操作会替换原有 Policy。只有 Bucket 的 owner 才能执行此操作，否则会返回 403 AccessDenied 错误。

示例代码

```
function putBucketPolicy($oosClient, $bucket)
{
    $options = array();

    $textPolicy = <<<< BPLY
    {
```

```

        'Version': '2012-10-17',
        'Id': 'http referer policy example',
        'Statement': [
            {
                'Sid': 'Allow get requests referred by
www.mysite.com ',
                'Effect': 'Allow',
                'Principal': {
                    'AWS': [
                        '*'
                    ]
                },
                'Action': 's3:*',
                'Resource': 'arn:aws:s3:::testphp2/*',
                'Condition': {
                    'StringLike': {
                        'aws:Referer': [
                            'http://www.mysite.com/*'
                        ]
                    }
                }
            }
        ]
    }
}

BPLY;

try {
    $oosClient->putBucketPolicy($bucket, $textPolicy,$options);
} catch (OosException $e) {
    $e->printException(__FUNCTION__);
    return;
}
print(__FUNCTION__ . ": OK" . "\n");
}

```

4.2.7 GET Bucket Policy

在 GET 操作的 url 中加上 Policy，可以获得指定 Bucket 的 Policy。只有 Bucket 的 owner 才能执行此操作，否则会返回 403 AccessDenied 错误。如果 Bucket 没有 Policy，返回 404, NoSuchPolicy 错误。

示例代码

```
function getBucketPolicy($oosClient, $bucket)
{
    $policyJson = null;
    try {
        $policyJson = $oosClient->getBucketPolicy($bucket);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
    print("Policy:" . $policyJson . "\n");
}
```

4.2.8 DELETE Bucket Policy

在 DELETE 操作的 url 中加上 policy，可以删除指定 Bucket 的 Policy。只有 Bucket 的 owner 才能执行此操作，否则会返回 403 AccessDenied 错误。如果 Bucket 没有 Policy，返回 204 NoContent。

示例代码

```
function deleteBucketPolicy($oosClient, $bucket)
{
    try {
        $oosClient->deleteBucketPolicy($bucket);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
}
```

```
print(__FUNCTION__ . ": OK" . "\n");
}
```

4.2.9 PUT Bucket WebSite

在 PUT 操作的 url 中加上 website，可以设置 website 配置。如果 Bucket 已经存在了 website，此操作会替换原有 website。只有 Bucket 的 owner 才能执行此操作，否则会返回 403 AccessDenied 错误。

WebSite 功能可以让用户将静态网站存放到 OOS 上。对于已经设置了 WebSite 的 Bucket，当用户访问 `http://bucketName.oos-website-cn.oos.ctyunapi.cn` 时，会跳转到用户指定的主页，当出现 4XX 错误时，会跳转到用户指定的出错页面。

如果想通过自有域名的形式（例如 `http://yourdomain.com/login.html`）而非通过第三方域名的形式（例如 `http://yourdomain.com.oos.ctyunapi.cn/login.html`）访问，可以创建一个名为“yourdomain.com”的 bucket，并在域名管理系统中将“yourdomain.com”增加一个别名记录“oos.ctyunapi.cn”。

示例代码

```
function putBucketWebsite($oosClient, $bucket)
{
    $websiteConfig = new WebsiteConfig("index.html", "error.html");
    try {
        $oosClient->putBucketWebsite($bucket, $websiteConfig);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
}
```

4.2.10 GET Bucket WebSite

在 GET 操作的 url 中加上 website, 可以获得指定 Bucket 的 website。只有 Bucket 的 owner 才能执行此操作, 否则会返回 403 AccessDenied 错误。

示例代码

```
function getBucketWebsite($oosClient, $bucket)
{
    $websiteConfig = null;
    try {
        $websiteConfig = $oosClient->getBucketWebsite($bucket);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
    print($websiteConfig->serializeToXml() . "\n");
}
```

4.2.11 DELETE Bucket WebSite

在 DELETE 操作的 url 中加上 website, 可以删除指定 bucket 的 website。只有 Bucket 的 owner 才能执行此操作, 否则会返回 403 AccessDenied 错误。如果 Bucket 没有 website, 返回 200 OK。

示例代码

```
function deleteBucketWebsite($oosClient, $bucket)
{
    try {
        $oosClient->deleteBucketWebsite($bucket);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
}
```

```
}

```

4.2.12 List Multipart Uploads

该接口用于列出所有已经通过 Initiate Multipart Upload 请求初始化，但未完成或未终止的分片上传过程。

响应中最多返回 1000 个分片上传过程的信息，它既是响应能返回的最大分片上传过程数目，也是请求的默认值。用户也可以通过设置 max-uploads 参数来限制响应中的分片上传过程数目。如果当前的分片上传过程数超出了这个值，则响应中会包含一个值为 true 的 IsTruncated 元素。如果用户要列出多于这个值的分片上传过程信息，则需要继续调用 List Multipart Uploads 请求，并在请求中设置 key-marker 和 upload-id-marker 参数。

在响应体中，分片上传过程的信息通过 key 来排序。如果用户的应用程序中启动了多个使用同一 key 对象开头的分片上传过程，那么响应体中分片上传过程首先是通过 key 来排序，在相同 key 的分片上传内部则是按上传启动的起始时间的升序来进行排列。

示例代码

```
function listMultipartUploads($oosClient, $bucket)
{
    $options = array(
        'delimiter' => '/',
        'max-uploads' => 100,
        // 'key-marker' => "",
        // 'prefix' => "",
        // 'upload-id-marker' => ""
    );
    try {
        $listMultipartUploadInfo =
$oosClient->listMultipartUploads($bucket, $options);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
    }
    return;
}
```



```

    }
    printf(__FUNCTION__ . ": listMultipartUploads OK\n");
    $listUploadInfo = $listMultipartUploadInfo->getUploads();
    var_dump($listUploadInfo);
}

```

4.2.13 PUT Bucket Logging

在 PUT 操作的 url 中加上 logging，可以进行添加/修改/删除 logging 的操作。如果 Bucket 已经存在了 logging，此操作会替换原有 logging。只有 Bucket 的 owner 才能执行此操作，否则会返回 403 AccessDenied 错误。

示例代码

```

function putBucketLogging($oosClient, $bucket)
{
    $option = array();
    // Access logs are stored in the same bucket.
    $targetBucket = $bucket;
    $targetPrefix = "access.log";

    try {
        $oosClient->putBucketLogging($bucket, $targetBucket,
        $targetPrefix, $option);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
}

```

4.2.14 GET Bucket Logging

在 GET 操作的 url 中加上 logging，可以获得指定 Bucket 的 logging。只有 Bucket 的 owner 才能执行此操作，否则会返回 403 AccessDenied 错误。

示例代码

```
function getBucketLogging($oosClient, $bucket)
{
    $loggingConfig = null;
    $options = array();
    try {
        $loggingConfig = $oosClient->getBucketLogging($bucket,
$options);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
    print($loggingConfig->serializeToXml() . "\n");
}
```

4.2.15 HEAD Bucket

此操作用于判断 Bucket 是否存在，而且用户是否有权限访问。如果 Bucket 存在，而且用户有权限访问时，此操作返回 200 OK。否则，返回 404 不存在，或者 403 没有权限。

示例代码

```
function headBucket($oosClient, $bucket)
{
    try {
        $res = $oosClient->headBucket($bucket);
    } catch (OosException $e) {
        printf(__FUNCTION__ . ": FAILED\n");
        printf($e->getMessage() . "\n");
        return;
    }
}
```

```

    }
    if ($res === true) {
        print(__FUNCTION__ . ": OK" . "\n");
    } else {
        print(__FUNCTION__ . ": FAILED" . "\n");
    }
}

```

4.2.16 PUT Bucket Lifecycle

存储在 OOS 中的对象有时需要有生命周期。比如，用户可能上传了一些周期性的日志文件到 Bucket 中，一段时间后，用户可能不需要这些日志对象了。通过 PUT Bucket lifecycle 可以指定生命周期。

示例代码

```

function putBucketLifecycle($oosClient, $bucket)
{
    $lifecycleConfig = new LifecycleConfig();
    $actions = array();
    $actions[] = new
LifecycleAction(OosClient::OOS_LIFECYCLE_EXPIRATION,
OosClient::OOS_LIFECYCLE_TIMING_DAYS, 3);
    $lifecycleRule = new LifecycleRule("delete obsoleted files",
"obsoleted/", "Enabled", $actions);
    $lifecycleConfig->addRule($lifecycleRule);
    $actions = array();
    $actions[] = new
LifecycleAction(OosClient::OOS_LIFECYCLE_EXPIRATION,
OosClient::OOS_LIFECYCLE_TIMING_DATE, '2022-10-12T00:00:00.000Z');
    $lifecycleRule = new LifecycleRule("delete temporary files",
"temporary/", "Enabled", $actions);
    $lifecycleConfig->addRule($lifecycleRule);
    try {
        $oosClient->putBucketLifecycle($bucket, $lifecycleConfig);
    }
}

```

```

    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
}

```

4.2.17 GET Bucket Lifecycle

此接口用于返回配置的 Bucket 生命周期。

示例代码

```

function getBucketLifecycle($oosClient, $bucket)
{
    $lifecycleConfig = null;
    try {
        $lifecycleConfig = $oosClient->getBucketLifecycle($bucket);
        $lifecycleConfig->getRules();
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
    print($lifecycleConfig->serializeToXml() . "\n");
}

```

4.2.18 DELETE Bucket Lifecycle

此接口用于删除配置的 Bucket 生命周期，OOS 将会删除指定 bucket 的所有生命周期配置规则。用户的对象将永远不会到期，OOS 也不会再自动删除对象。

示例代码

```

function deleteBucketLifecycle($oosClient, $bucket)
{

```

```

try {
    $oosClient->deleteBucketLifecycle($bucket);
} catch (OosException $e) {
    $e->printException(__FUNCTION__);
    return;
}
print(__FUNCTION__ . ": OK" . "\n");
}

```

4.2.19 PUT Bucket Accelerate

在 PUT 操作的 url 中加上 accelerate，可以进行添加或修改 CDN IP 白名单的操作。如果 Bucket 已经配置了 CDN 加速，此操作会替换原有配置。只有 Bucket 的 owner 才能执行此操作，否则会返回 403 AccessDenied 错误。一个 Bucket 最多配置 5 个 IP 白名单地址段。

示例代码

```

function putBucketAccelerate($oosClient, $bucket)
{
    $AccelerateConfig = new AccelerateConfig();
    $AccelerateConfig->setStatus("Enabled");
    $AccelerateConfig->addIPWhiteLists("36.111.88.0/24");
    $AccelerateConfig->addIPWhiteLists("114.80.1.136");
    try {
        $oosClient->putBucketAccelerate($bucket, $AccelerateConfig);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
}

```

4.2.20 GET Bucket Accelerate

在 GET 操作的 url 中加上 accelerate, 可以获得指定 Bucket 的 cdn 配置信息。只有 Bucket 的 owner 才能执行此操作, 否则会返回 403 AccessDenied 错误。

如果 Bucket 没有配置过 CDN 加速, 那么将不会返回状态信息。

示例代码

```
function getBucketAccelerate($oosClient, $bucket)
{
    try {
        $accelerateConfig = $oosClient->getBucketAccelerate($bucket);

    } catch (OosException $e) {
        printf(__FUNCTION__ . ": FAILED\n");
        printf($e->getMessage() . "\n");
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
    print($accelerateConfig->serializeToXml() . "\n");
    //$accelerateConfig->getRefererList();
}
```

4.2.21 PUT Bucket cors

跨域资源共享 (Cross-Origin Resource Sharing, CORS) 定义了客户端 Web 应用程序在一个域中与另一个域中的资源进行交互的方式, 是浏览器出于安全考虑而设置的一个限制, 即同源策略。例如, 当来自于 A 网站的页面中的 JavaScript 代码希望访问 B 网站的时候, 浏览器会拒绝该访问, 因为 A、B 两个网站是属于不同的域。

通过 CORS, 客户可以构建丰富的客户端 Web 应用程序, 同时可以选择性地允许跨域访问 OOS 资源。

示例代码

```

function putBucketCors($oosClient, $bucket)
{
    $corsConfig = new CorsConfig();
    $rule = new CorsRule();
    $rule->addAllowedHeader("x-amz-header");
    $rule->addAllowedOrigin("http://www.b.com");
    $rule->addAllowedMethod("POST");
    $rule->setMaxAgeSeconds(10);
    $corsConfig->addRule($rule);

    try {
        $oosClient->putBucketCors($bucket, $corsConfig);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
}

```

4.2.22 GET Bucket cors

返回 Bucket 的跨域配置信息。只有 Bucket 的 owner 才能执行此操作，否则会返回 403 AccessDenied 错误。

示例代码

```

function getBucketCors($oosClient, $bucket)
{
    $corsConfig = null;
    try {
        $corsConfig = $oosClient->getBucketCors($bucket);
    } catch (OosException $e) {
        printf(__FUNCTION__ . ": FAILED\n");
        printf($e->getMessage() . "\n");
        return;
    }
}

```

```
    }  
    print(__FUNCTION__ . ": OK" . "\n");  
    print($corsConfig->serializeToXml() . "\n");  
}
```

4.2.23 DELETE Bucket cors

删除 Bucket 的跨域配置信息。只有 Bucket 的 owner 才能执行此操作，否则返回 403 AccessDenied 错误。

示例代码

```
function deleteBucketCors($oosClient, $bucket)  
{  
    try {  
        $oosClient->deleteBucketCors($bucket);  
    } catch (OosException $e) {  
        printf(__FUNCTION__ . ": FAILED\n");  
        printf($e->getMessage() . "\n");  
        return;  
    }  
  
    print(__FUNCTION__ . ": OK" . "\n");  
}
```

4.3 关于 Object 的操作

4.3.1 PUT Object

Put Object 用来向指定 bucket 中添加一个对象，要求发送请求者对该 Bucket 有写权限，用户必须添加完整的对象。

示例代码

OOS 5.0 示例

```
function putObject($oosClient, $bucket)
{
    $object = "object.php";
    $content = file_get_contents(__FILE__);
    $options = array();
    $xHeaders = array();
    $xHeaders["Cache-Control"] = "yes";
    $xHeaders["x-amz-meta-aaa"] = "aaa";
    $xHeaders["x-amz-storage-class"] = "STANDARD";

    $options[OosUtil::OOS_HEADERS] = $xHeaders;
    try {
        $oosClient->putObject($bucket, $object, $content, $options,null);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
}
```

OOS 6.0 示例

```
function putObjectToLcation($oosClient, $bucket)
{
    $object = "object.php";
    $content = file_get_contents(__FILE__);
    $options = array();
    $xHeaders = array();
    $xHeaders["Cache-Control"] = "yes";
    $xHeaders["x-amz-meta-aaa"] = "aaa";
    $xHeaders["x-amz-storage-class"] = "STANDARD";

    $options[OosUtil::OOS_HEADERS] = $xHeaders;
    try {
```

```

        $type = "Specified";
        $locationList = array();
        $locationList["ShenYang"] = "ShenYang";
        $locationList["ChengDu"] = "ChengDu";
        $scheduleStrategy = "NotAllowed";
        $dataLocation = new
DataLocation($type,$locationList,$scheduleStrategy);

        $oosClient->putObject($bucket, $object, $content,
$options,$dataLocation);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
}

```

4.3.2 GET Object

GET Object 操作用来检索在 OOS 中的对象信息。执行 GET 操作，用户必须对 object 所在的 bucket 有读权限。如果 Bucket 是 public read 的权限，匿名用户也可以通过非授权的方式进行读操作。

示例代码

```

function getObject($oosClient, $bucket)
{
    //$object = "object.php";
    $object = "liveplus/object3.php";
    $options = array();
    $xHeaders = array();
    $xHeaders["Range"] = "bytes=1-100";
    $options[OosUtil::OOS_HEADERS] = $xHeaders;
    try {
        $objectInfo = $oosClient->getObject($bucket, $object, $options);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
    }
}

```

```

        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
}

```

4.3.3 Delete Object

Delete Object 操作移除指定的对象，要求用户要对对象所在的 Bucket 拥有写权限。

示例代码

```

function deleteObject($oosClient, $bucket)
{
    $object = "oos-php-sdk-test/upload-test-object-name.txt";
    try {
        $oosClient->deleteObject($bucket, $object);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
}

```

4.3.4 PUT Object - Copy

通过 PUT Object- Copy 操作创建一个存储在 OOS 里的对象的拷贝。PUT Object -Copy 操作类似于执行一个 GET 然后在执行一次 PUT。增加请求头，x-awz-copy-source，使用 PUT 操作将源对象存入指定 bucket。要执行拷贝请求，用户需要对源对象有读权限，对目标 Bucket 有写权限。

示例代码

OOS 5.0 示例

```

function copyObject($oosClient, $bucket)

```

```
{
    $fromBucket = $bucket;
    $fromObject = "oos-php-sdk-test/upload-test-object-name.txt";
    $toBucket = $bucket;
    $toObject = $fromObject . '.copy';
    $options = array();
    try {
        $oosClient->copyObject($fromBucket, $fromObject, $toBucket,
            $toObject, $options);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
    }
    return;
}
print(__FUNCTION__ . ": OK" . "\n");
}
```

OOS 6.0 示例

```
function copyObject($oosClient, $bucket)
{
    $fromBucket = $bucket;
    $fromObject = "oos-php-sdk-test/upload-test-object-name.txt";
    $toBucket = $bucket;
    $toObject = $fromObject . '.copy';

    $options = array();

    try {
        $type = "Specified";
        $locationList = array();
        $locationList["ShenYang"] = "ShenYang";
        $locationList["ChengDu"] = "ChengDu";
        $scheduleStrategy = "NotAllowed";
        $dataLocation = new
        DataLocation($type,$locationList,$scheduleStrategy);
    }
}
```

```

        $copyPartInfo= $oosClient->copyObject($fromBucket,
$fromObject, $toBucket, $toObject, $options,$dataLocation);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
    $copyPartInfo->getETag();
}

```

4.3.5 Initial Multipart Upload

本接口初始化一个分片上传（Multipart Upload）操作，并返回一个上传 ID，此 ID 用来将此次分片上传操作中上传的所有片段合并成一个对象。用户在执行每一次子上传请求（见 Upload Part）时都应该指定该 ID。用户也可以在表示整个分片上传完成的最后一个请求中指定该 ID。或者在用户放弃该分片上传操作时指定该 ID。

示例代码

OOS5.0 示例：

```

function initiateMultipartUpload($oosClient, $bucket)
{
    $object = "seL4.zip";
    $options = array();

    try {
        $uploadInfo = $oosClient->initiateMultipartUpload($bucket,
$object, $options);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
    print("uploadId=" . $uploadInfo->getUploadId() . " . \n");
}

```

OOS 6.0 示例

```
function initiateMultipartUpload($oosClient, $bucket)
{
    $subject = "mutipart.txt";
    $options = array();
    $xheaders = array();
    $xheaders["Cache-Control"] = "yes";
    $xheaders["x-amz-meta-aaa"] = "aaa";
    $xheaders["x-amz-storage-class"] = "STANDARD";

    $options[OosUtil::OOS_HEADERS] = $xheaders;
    try {
        $stype = "Specified";
        $locationList = array();
        $locationList["ShenYang"] = "ShenYang";
        $locationList["ChengDu"] = "ChengDu";
        $scheduleStrategy = "NotAllowed";
        $dataLocation = new
DataLocation($stype,$locationList,$scheduleStrategy);
        $uploadInfo = $oosClient->initiateMultipartUpload($bucket,
Subject, $options, $dataLocation);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
    print("uploadId=" . $uploadInfo->getUploadId() . " . \n");
    print("getKey=" . $uploadInfo->getKey() . " . \n");
    print("getBucket=" . $uploadInfo->getBucket() . " . \n");
}
```

4.3.6 Upload Part

Upload Part 用于实现分片上传操作中片段的上传。

OOS SDK PHP 开发者指南

在上传任何一个分片之前，必须执行 Initial Multipart Upload 操作来初始化分片上传操作，初始化成功后，OOS 会返回一个上传 ID，这是一个唯一的标识，用户必须在调用 Upload Part 接口时加入该 ID。

分片号 PartNumber 可以唯一标识一个片段并且定义该分片在对象中的位置，范围从 1 到 10000。如果用户用之前上传过的片段的分片号来上传新的分片，之前的分片将会被覆盖。

除了最后一个分片外，所有分片的大小都应该不小于 5M，最后一个分片的大小不受限制。

为了确保数据不会由于网络传输而毁坏，需要在每个分片上传请求中指定 Content-MD5 头，OOS 通过提供的 Content-MD5 值来检查数据的完整性，如果不匹配，则会返回一个错误信息。

示例代码

响应中包含 Etag 头，用户需要在最后发送完成分片上传过程请求的时候包含该 Etag 值。

```
function uploadPart($oosClient, $bucket,$uploadId)
{
    $subject = "seL4.zip";
    $file = "D:\\TMP\\seL4.zip";
    $options = array();
    $options["fileUpload"] = OosUtil::encodePath($file);
    $options["partNumber"] = 1;

    try {
        $etag =
$oosClient->uploadPart($bucket,$subject,$uploadId,$options);
        print("\n===etag: " . (string)$etag);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
}
```

4.3.7 Complete Multipart Upload

Complete Multipart Upload 通过合并之前的上传片段来完成一次分片上传过程。

用户首先初始化分片上传过程，然后通过 Upload Part 接口上传所有分片。在成功将一次分片上传过程的所有相关片段上传之后，调用这个接口来结束分片上传过程。当收到这个请求的时候，OOS 会以分片号升序排列的方式将所有片段依次拼接来创建一个新的对象。在这个 Complete Multipart Upload 请求中，用户需要提供一个片段列表。同时，必须确保这个片段列表中的所有片段必须是已经上传完成的，Complete Multipart Upload 操作会将片段列表中提供的片段拼接起来。对片段列表中的每个片段，需要提供该片段上传完成时返回的 ETag 头的值和对应的分片号。

处理一次 Complete Multipart Upload 请求可能需要花费几分钟时间。OOS 在处理这个请求之前会发送一个值为 200 响应头。在处理这个请求的过程中，OOS 每隔一段时间就会发送一个空格字符来防止连接超时。因为一个请求在初始的 200 响应已经发出之后仍可能失败，用户需要检查响应体内容以判断请求是否成功。

注意，如果 Complete Multipart Upload 请求失败，应用程序会尝试重新发送该请求。

由于 Complete Multipart Upload 请求可能需要花费几分钟时间，所以 OOS 提供了不合并片段也可以读取 Object 内容的功能。在没有调用 Complete Multipart Upload 接口合并片段时，也可以通过调用 Get Object 接口来获取文件内容，OOS 会根据最近一次创建的 uploadId，以分片号升序的方式顺序读取片段内容，返回给客户端。但此时不能返回整个文件的 ETag 值。

示例代码

```
function completeMultipartUpload($oosClient, $bucket,$uploadId)
{
    $object = "seL4.zip";
```



```

$uploadParts = array();

$uploadParts[] = array(
    'PartNumber' => 1,
    'ETag' => "706010df87ac70109e9aec2048398440"
);

try {
    $completeMultipartUploadInfo =
$soosClient->completeMultipartUpload($bucket,$object,$uploadId,$uploadParts);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
    $completeMultipartUploadInfo->getKey();
}

```

4.3.8 Abort Multipart Upload

该接口用于终止一次分片上传操作。分片上传操作被终止后，用户不能再通过上传 ID 上传其它片段，之前已上传完成的片段所占用的存储空间将被释放。如果此时任何片段正在上传，该上传过程可能会也可能不会成功。所以，为了释放所有片段所占用的存储空间，可能需要多次终止分片上传操作。

示例代码

```

function abortMultipartUpload($soosClient, $bucket,$uploadId)
{
    $object = "seL4.zip";
    try {
        $soosClient->abortMultipartUpload($bucket,$object,$uploadId);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
    }
}

```

```

        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
    print("uploadId=" . "$uploadId\n");
}

```

4.3.9 List Part

List Part 操作用于列出一个分片上传过程中已经上传完成的所有片段。

List Part 必须包含一个通过 Initial Multipart Upload 操作获取的上传 ID。该请求最多返回 1000 个上传片段信息，默认返回的片段数是 1000。用户可以通过指定 `max-parts` 参数来指定一次请求返回的片段数。如果用户的分片上传过程超过 1000 个片段，响应中的 `IsTruncated` 字段的值则被设置成 `true`，并且指定一个 `NextPartNumberMarker` 元素。用户可以在下一个连续 List Part 请求中加入 `part-number-marker` 参数，并把它设置成上一个请求返回的 `NextPartNumberMarker` 值。

示例代码

```

function listParts($oosClient, $bucket,$object,$uploadId)
{
    $options = array(
        'max-parts' => 1000,
        'part-number-marker' => 2,
    );
    try {
        $listPartsInfo = $oosClient->listParts($bucket, $object, $uploadId,
$options);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    printf(__FUNCTION__ . ": listParts OK\n");
}

```

```

    $listPart = $listPartsInfo->getListPart();
    var_dump($listPart);
}

```

4.3.10 Copy Part

可以将已经存在的 object 作为分段上传的片段，拷贝生成一个新的片段。需要指定请求头 `x-amz-copy-source` 来定义拷贝源。如果想拷贝源 object 中的一部分，可以加请求头 `x-amz-copy-source-range`。

示例代码

```

function copyObjectPart($oosClient, $fromBucket, $fromObject, $toBucket,
    $toObject,$toPartNumber,$toUploadId)
{
    $options = array();
    try {
        $copyPartInfo = $oosClient->copyObjectPart($fromBucket,
            $fromObject, $toBucket, $toObject,
                $toPartNumber,$toUploadId,$options);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
}

```

4.3.11 Delete Multiple Objects

Delete Multiple Object 功能支持用一个 HTTP 请求删除一个 bucket 中的多个 object。如果你知道你想删除的 object 名字，此功能可以批量删除这些 object，而不用发送多个单独的删除请求。

批量删除请求包含一个不超过 1000 个 object 的 XML 列表。在这个 xml 中，你需要指定要删除的 object 的名字。对于每个 Object，OOS 都会返回删除的结果，成功或者失败。注意，如果请求中的 object 不存在，那么 OOS 也会返回删除成功。

批量删除功能支持两种格式的响应，全面信息和简明信息。默认情况下，OOS 在响应中会显示全面信息，即包含每个 object 的删除结果。在简明信息模式下，OOS 只返回删除出错的 object 的结果。对于成功删除的 object，在响应中将不返回任何信息。

最后，批量删除功能必须使用 Content-MD5 请求头，OOS 使用此头来保证请求体在传输过程中没有被修改。

示例代码

```
function deleteObjects($oosClient, $bucket)
{
    $objects = array();
    $objects[] = "mpu/multipart-test.txt";
    $objects[] = "mpu/multipart-test.txt.copied";
    try {
        $oosClient->deleteObjects($bucket, $objects);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
}
```

4.3.12 生成共享链接

对于私有或只读 Bucket，可以通过生成 Object 的共享链接的方式，将 Object 分享给其他人，同时可以在链接中设置限速以对下载速度进行控制。

示例代码

```
function getSignedUrlForGettingObject($oosClient, $bucket)
{
    $object = "object.php";
    $timeout = 3600 * 24;
    //限速 KB
    $options[OosClient::OOS_LIMITRATE] = "1024";
    try {
```

```

        $signedUrl = $oosClient->signUrl($bucket, $object,
$timeout,OosClient::OOS_HTTP_GET,$options);
    } catch (OosException $e) {
        printf(__FUNCTION__ . ": FAILED\n");
        printf($e->getMessage() . "\n");
        return;
    }

    print(__FUNCTION__ . ": signedUrl: " . $signedUrl . "\n");
}

```

4.3.13 HEAD Object

Head 操作用于获取对象的元数据信息，而不返回数据本身。当只希望获取对象的属性信息时，可以使用此操作。

示例代码

```

function headObject($oosClient, $bucket)
{
    $object = "object.php";
    try {
        $objectInfo = $oosClient->headObject($bucket, $object);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
    print($objectInfo->getETag());
}

```

4.4 关于 AccessKey 的操作

OOS 的 IAM 服务端地址参见《开发者文档》Endpoint 列表。

4.4.1 CreateAccessKey

创建一对普通的 AccessKey 和 SecretKey，默认的状态是 Active。只有主 key 才能执行此操作。

为保证账户的安全，SecretKey 只在创建的时候会被显示。请把 key 保存起来，比如保存到一个文本文件中。如果 SecretKey 丢失了，你可以删除 AccessKey，并创建一对新的 key。默认情况下，每个账号最多创建 10 个 AccessKey。

示例代码

```
function createAccessKey($oosIamClient)
{
    try {
        $options = array();
        $options["bucket"] = "";
        $options["region"] = 'cn';
        $options["OOS_SERVICE"] = 'sts';

        $keyInfo = $oosIamClient->CreateAccessKey($options);
        $keyInfo->getAccessKeyId();
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
}
```

4.4.2 DeleteAccessKey

删除一对普通的 AccessKey 和 SecretKey。只有主 key 才能执行此操作。

示例代码

```
function deleteAccessKey($oosIamClient,$accessKeyId)
{
    try {
```

```

        $data = $soosIamClient->DeleteAccessKey($accessKeyId);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }

    print(__FUNCTION__ . ": OK" . "\n");
    print("\nResponse json data is :" . $data . "\n");
}

```

4.4.3 UpdateAccessKey

更新普通的 AccessKey 的状态，或将普通 key 设置成为主 key，反之亦然。只有主 key 才能执行此操作。

示例代码

```

function updateAccessKey($soosIamClient,$accessKeyId,$status,$isPrimary)
{
    try {
        $data =
$soosIamClient->UpdateAccessKey($accessKeyId,$status,$isPrimary);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }
    print(__FUNCTION__ . ": OK" . "\n");
    print("\nResponse json data is :" . $data . "\n");
}

```

4.4.4 ListAccessKey

ListAccessKey 列出账号下的主 key 和普通 key。只有主 key 才能执行此操作。可以通过 MaxItems 参数指定返回的结果数量，默认返回 100 个 key。可以通过 Marker 参数设置返回的起始位置，该参数可以从前一次请求的响应体中获得。为保证账号安全，ListAccessKey 操作时，不会返回 SecretKey。

示例代码

```
function listAccessKey($oosIamClient,$MaxItems,$Marker)
{
    $options = array();
    $options["bucket"] = "";
    $options["region"] = 'cn';
    $options["OOS_SERVICE"] = 'sts';

    $keyList = array();
    try {
        if($MaxItems<=0)
            $MaxItems = 100;

        $keyList =
$oosIamClient->ListAccessKey($MaxItems,$Marker,$options);
    } catch (OosException $e) {
        $e->printException(__FUNCTION__);
        return;
    }

    foreach ($keyList->getKeyList() as $keyOne) {
        print("ListAccessKey" . "\t accessKeyId:" .
$keyOne->getAccessKeyId()
        . "\t status:" . $keyOne->getStatus()
        . "\t isPrimary:" . $keyOne->getIsPrimary()
        . "\n");
    }
    print(__FUNCTION__ . ": OK" . "\n");
}
```

4.5 错误处理

4.5.1 错误响应

当请求出错时，OOS 返回以下格式的错误响应信息。


```

<?xml version="1.0" encoding="UTF-8"?>
<Error>
  <Code>NoSuchKey</Code>
  <Message>The resource you requested does not exist</Message>
  <Resource>/mybucket/myfoto.jpg</Resource>
  <RequestId>4442587FB7D0A2F9</RequestId>
</Error>

```

响应元素说明如下。

名称	描述
Code	错误码是描述出错信息的字符串。详细的错误码信息可以参见后面的列表。
Message	错误信息的简单描述。
RequestId	请求的 ID。
Resource	出错相关的 bucket 或 object 信息。

4.5.2 错误码列表

错误码	描述	HTTP 状态码
NotVerify	用户未激活	403 Forbidden
InvalidArgument	无效的请求参数	400
BucketAlreadyExists	请求的 Bucket 的 name 已经不可用，请选择两个不同的名字尝试。	409 Conflict
BucketNotEmpty	试图删除的 Bucket 非空	409 Conflict
InvalidAccessKeyId	使用的 OOS ID 不存在	403 Forbidden
InvalidBucketName	指定的 bucket 无效	400 Bad Request
InvalidURI	无法解析指定的 URI	400 Bad Request
MethodNotAllowed	指定的方法不允许操作该资源	405 Method Not Allowed
MissingContentLength	在 HTTP 头中必须提供 ContentLength	411 length required
RequestTimeTooSkewed	请求的时间和服务器时间相差 15 分钟以上	403 Bad Request
NoSuchBucket	指定的 bucket 不存在	404 Not Found
NoSuchKey	指定对象的 Key 不存在	404 Not Found
SignatureDoesNotMatch	我们计算出的请求中的签名与用户提供的签名不一致。检查 OOS 账户安全	403 Forbidden
TooManyBuckets	Bucket 个数超出限制	400 Bad Request

OOS SDK PHP 开发者指南

PutObjectTooFast	并发写同一个对象或分段片段时冲突	400 Bad Request
MalformedXML	XML 格式不合法	400 Bad Request
CanNotModifyMetadataLocation	索引位置不允许修改	400 Bad Request
InvalidLocationConstraint	用户提供的配置索引位置和数据位置的 xml 格式不合法	400 Bad Request
InvalidRequest	IP 白名单不合法。	400 Bad Request
InvalidRequest	IP 白名单个数超过限制。	400 Bad Request
Internal Error	OOS 服务错误，请重试	500 InternalServerError
SlowDown	OOS 服务繁忙，请重试	503 Service Unavailable