对象存储

最佳实践

产品文档

腾讯云

文档目录

最佳实践
访问控制与权限管理
ACL 访问控制实践

最近更新时间：2017-11-09 18:22:29

ACL 概述

访问控制列表（ACL）是基于资源的访问策略选项之一 ，可用来管理对存储桶和对象的访问。使用 ACL 可向其他根账户、子账户和用户组授予基本的读、写权限。

与访问策略有所不同的是，ACL 的管理权限有一些限制：

- 仅支持对腾讯云的账户赋予权限；
- 仅支持读对象、写对象、读 ACL、写 ACL 和全部权限等五个操作组；
- 不支持赋予生效条件；
- 不支持显示拒绝效力。

ACL 支持的控制粒度：

- 存储桶（Bucket）
- 对象键前缀（Prefix）
- 对象（Object）

ACL 的控制元素

当创建存储桶或对象时，其资源所属的主账户将具备对资源的全部权限，且不可修改或删除。您可以使用 ACL 赋予其他腾讯云账户的访问权限。如下提供了一个存储桶的 ACL 示例。

```
<AccessControlPolicy>
<Owner>
<ID>qcs::cam::uin/12345:uin/12345</ID>
<DisplayName>qcs::cam::uin/12345:uin/12345</DisplayName>
</Owner>
<AccessControlList>
<Grant>
<Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="RootAccount">
<ID>qcs::cam::uin/12345:uin/12345</ID>
<DisplayName>qcs::cam::uin/12345:uin/12345</DisplayName>
```

```
</Grantee>
<Permission>FULL_CONTROL</Permission>
</Grant>
<Grant>
<Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="RootAccount">
<ID>qcs::cam::uin/54321:uin/54321</ID>
<DisplayName>qcs::cam::anyone:anyone</DisplayName>
</Grantee>
<Permission>READ</Permission>
</Grant>
</AccessControlList>
</AccessControlPolicy>
```

如上 ACL 包含了识别该存储桶所有者的 Owner 元素，该存储桶所有者具备该存储桶的全部权限。同时 Grant 元素授予了匿名的读取权限，其表述形式为 `qcs::cam::anyone:anyone` 的 READ 权限。

权限被授予者

根账户

您可以对其他根账户授予用户访问权限，使用 CAM 中对委托人（principal）的定义进行授权。描述为：

```
qcs::cam::uin/1238423:uin/1238423
```

子账户

您可以对您的根账户下的子账户，或其他根账户下的子账户授权，使用 CAM 中对委托人（principal）的定义进行授权。描述为：

```
qcs::cam::uin/1238423:uin/3232
```

匿名用户

您可以对匿名用户授予访问权限，使用 CAM 中对委托人（principal）的定义进行授权。描述为：

```
qcs::cam::anyone:anyone
```

权限操作组

以下表格提供了 ACL 支持的权限操作组。

操作组	授予存储桶	授予前缀	授予对象
READ	列出和读取存储桶中的对象	列出和读取目录下的对象	读取对象

操作组	授予存储桶	授予前缀	授予对象
WRITE	创建、覆盖和删除存储桶中的任意对象	创建、覆盖和删除目录下的任意对象	覆盖和删除对象
READ_ACP	读取存储桶的 ACL	读取目录下的 ACL	读取对象的 ACL
WRITE_ACP	修改存储桶的 ACL	修改目录下的 ACL	修改对象的 ACL
FULL_CONTROL	对存储桶和对象的任何操作	对目录下的对象做任何操作	对对象执行任何操作

标准 ACL 描述

COS 支持一系列的预定义授权，称之为标准 ACL，下表列出了标准 ACL 的授权含义。

> **注意：**
> 由于主账户始终拥有 FULL_CONTROL 权限，以下表格中不再对此特别说明。

标准 ACL	含义
(空)	此为默认策略，其他人无权限，资源继承上级权限。
private	其他人没有权限。
public-read	匿名用户组具备 READ 权限。
public-read-write	匿名用户组具备 READ 和 WRITE 权限，通常不建议在存储桶赋予此权限。

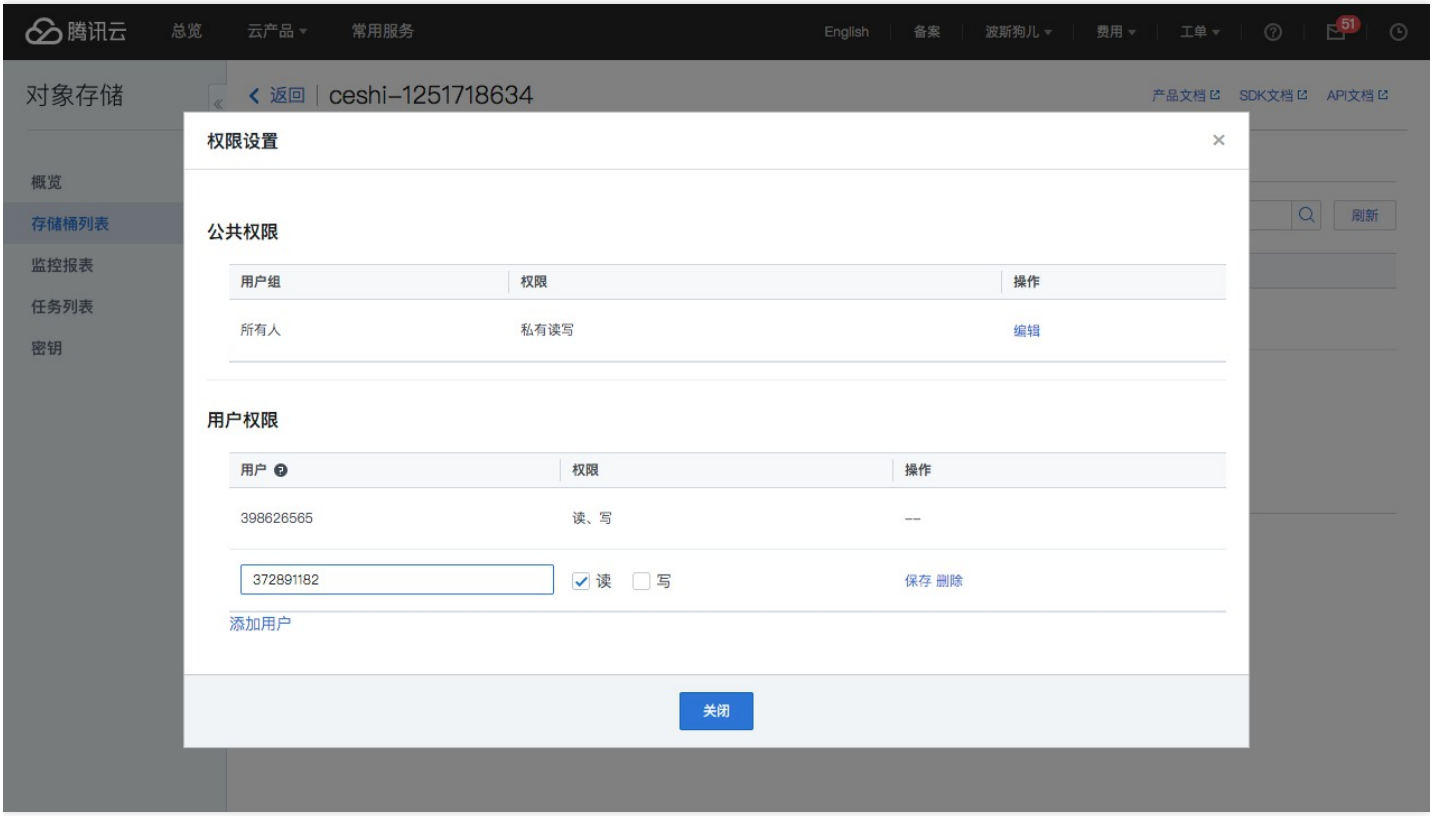
ACL 示例

为存储桶设置 ACL

以下示例对存储桶授予了另一个根账户的读取权限：

为对象设置 ACL

以下示例对某个对象授予了另一个根账户的读取权限：



访问管理实践

最近更新时间：2017-11-10 14:55:06

访问管理概述

访问管理（Cloud Access Management，CAM）是腾讯云提供的一种身份验证和授权服务，主要用于帮助客户安全管理腾讯云账户下的资源的访问权限。在授予权限时，可以对授权的对象、资源、操作进行管理，并支持设置一些策略限制。

访问管理功能

根账号资源的授权

可以将根账号的资源授权给其他人员，包括子账号或者是其他根账号，而不需要分享根账号相关的身份凭证。

精细化的权限管理

可以针对不同的资源为不同的人员授予不同的访问权限。例如可以允许某些子账号拥有某个 COS 存储桶的读权限，而另外一些子账号或者根账号可以拥有某个 COS 存储对象的写权限等。上述资源、访问权限、用户都可以批量打包。

最终一致性

CAM 目前支持腾讯云的多个地域，通过复制策略数据实现在跨地域的数据同步，虽然 CAM 策略的修改会及时提交，不过跨地域的策略同步会导致策略生效的延迟；同时 CAM 使用缓存来提高性能（目前是一分钟缓存），更新需要在缓存过期后生效。

访问管理应用场景

企业子账号权限管理

企业内不同岗位的员工需要拥有该企业云资源的最小化访问权限。

场景：某个企业拥有很多云资源，包括 CVM 、 VPC 实例、 CDN 实例、 COS 存储桶和对象等。该企业拥有很多员工，包括开发人员、测试人员、运维人员等。
部分开发人员需要拥有其所在项目相关的开发机云资源的读写权限，测试人员需要拥有其所在项目的测试机云资源的读写权限，运维人员负责机器的购买和日常运营。当企业员工职责或参与项目发生变更，将终止对应的权限。

不同企业之间的权限管理

不同企业间需要进行云资源的共享。

场景：某企业拥有很多云资源，该企业希望能专注产品研发，而将云资源运维工作授权给其他运营企业来实施。当运营企业的合同终止时，将收回对应的管理权限。

访问管理策略语法

策略（policy）由若干元素构成，用来描述授权的具体信息。核心元素包括委托人（principal）、操作（action）、资源（resource）、生效条件（condition）以及效力（effect）。

策略语法说明

- 元素仅支持小写。它们在描述上没有顺序要求。
- 对于策略没有特定条件约束的情况， condition 元素是可选项。
- 在控制台中不允许写入 principal 元素，仅支持在策略管理 API 中和策略语法相关的参数中使用 principal 。

版本 version

描述策略语法版本。目前仅允许值为 2.0。该元素是必填项。

委托人 principal

用于描述策略授权的实体。包括用户（开发商、子账号、匿名用户）、用户组，仅支持在策略管理 API 中策略语法相关的参数中使用该元素。

语句 statement

用于描述一条或多条权限的详细信息。该元素包括 action 、 resource 、 condition 、 effect 等多个其他元素的权限或权限集合。一条策略有且仅有一个 statement 元素。该元素是必填项。

操作 action

用于描述允许或拒绝的操作。操作可以是 API （以 name 前缀描述）或者功能集（一组特定的 API ，以 permid 前缀描述）。该元素是必填项。

资源 resource

用于描述授权的具体数据。资源用六段式描述。每款产品的资源定义详情会有所区别。有关如何指定资源的信息，请参阅您编写的资源声明所对应的产品文档。该元素是必填项。

生效条件 condition

用于描述策略生效的约束条件。条件包括操作符、操作键和操作值组成。条件值可包括时间、 IP 地址等信息。有些服务允许您在条件中指定其他值。该元素是非必填项。

效力 effect

用于描述声明产生的结果，包括 allow（允许）和 deny（显式拒绝）两种情况。该元素是必填项。

策略限制

限制项	限制值
一个根账号中的用户组数	20 个
一个根账号中的子用户数	1000 个
一个子用户可加入的用户组的数量	10 个
一个用户组中的子用户数	100 个
一个根账号的策略数	1000 个
关联到一个 CAM 用户、用户组的策略数	20 个
自定义策略字符数	4096 字符

策略示例

以下示例表示：
允许属于开发商 ID 1238423 下的子账号 ID 3232523 以及组 ID 18825， 在访问 IP 网段 `10.121.2.10/24` 时，对北京地域的 COS 存储桶 bucketA 和广州地域的 COS 对象 object2 ，拥有所有 COS 读 API 、写对象、以及发送消息队列的权限。

```
{
"version":"2.0",
"principal":
{"qcs":["qcs::cam::uin/1238423:uin/3232523",
"qcs::cam::uin/1238423:groupid/18825"]
},
"statement":
[
{
"effect":"allow",
"action":["name/cos:PutObject","permid/280655"],
"resource":["qcs::cos:ap-beijing:uid/1238423:prefix/bucketA/*",
"qcs::cos:ap-guangzhou:uid/1238423:prefix/bucketB/object2"],
"condition": {"ip_equal":{"qcs:ip":"10.121.2.10/24"}}
},
{
"effect":"allow",
"action":"name/cmqqueue:Sendmessages",
"resource":"*"
}
```

```
]
}
```

使用子账号对 COS 进行授权

最近更新时间：2018-05-24 17:45:57

对于腾讯云对象存储资源，不同企业之间或同企业多团队之间，需要对不同的团队或人员配置不同的访问权限。您可通过 CAM（访问管理，以下简称 CAM）进行配置，实现对不同团队或人员，以存储桶/对象粒度进行不同操作的权限设置。
总的来说，配置分为三步：创建子账号，对子账号授予权限，子账号访问。

首先，我们需要先了解几个关键概念。

名词解释

CAM 相关术语、配置详细描述请查看 CAM 概述。

根账号

根账号又被称为开发商。用户申请腾讯云账号时，系统会创建一个用于登录腾讯云服务的根账号身份。根账号是腾讯云资源使用计量计费的基本主体。
根账号默认拥有其名下所拥有的资源的完全访问权限，包括访问账单信息，修改用户密码，创建用户和用户组以及访问其他云服务资源等。默认情况下，资源只能被根账号所访问，任何其他用户访问都需要获得根账号的授权。

子账号（用户）和用户组

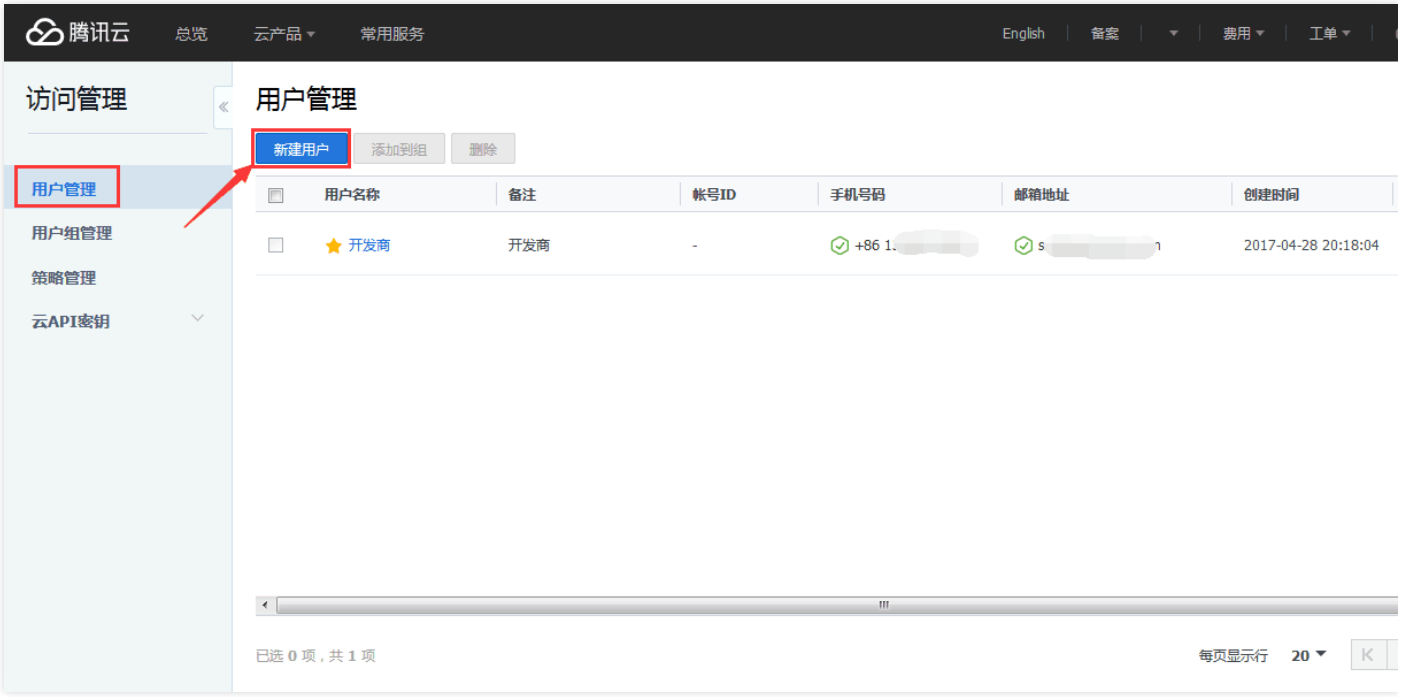
- 子账号是由根账号创建的实体，有确定的身份 ID 和身份凭证，拥有登录腾讯云控制台的权限。
- 子账号默认不拥有资源，必须由所属根账号进行授权。
  - 一个根账号可以创建多个子账号（用户）。
  - 一个子账号可以归属于多个根账号，分别协助多个根账号管理各自的云资源，但同一时刻，一个子账号只能登录到一个根账号下，管理该根账号的云资源。
- 子账号可以通过控制台切换开发商（根账号），从一个根账号切换到另外一个根账号。
  - 子账号登录控制台时，会自动切换到默认根账号上，并拥有默认根账号所授予的访问权限。
  - 切换开发商之后，子账号会拥有切换到的根账号授权的访问权限，而切换前的根账号授予的访问权限会立即失效。
- 用户组是多个相同职能的用户（子账号）的集合。您可以根据业务需求创建不同的用户组，为用户组关联适当的策略，以分配不同权限。

子账号权限配置

步骤一：创建子账号

在 CAM 控制台可创建子账号，并配置授予子账号的访问权限。具体操作如下所示：

1. 登录 CAM控制台，单击左侧菜单栏【用户管理】，单击页面按钮【新建用户】：



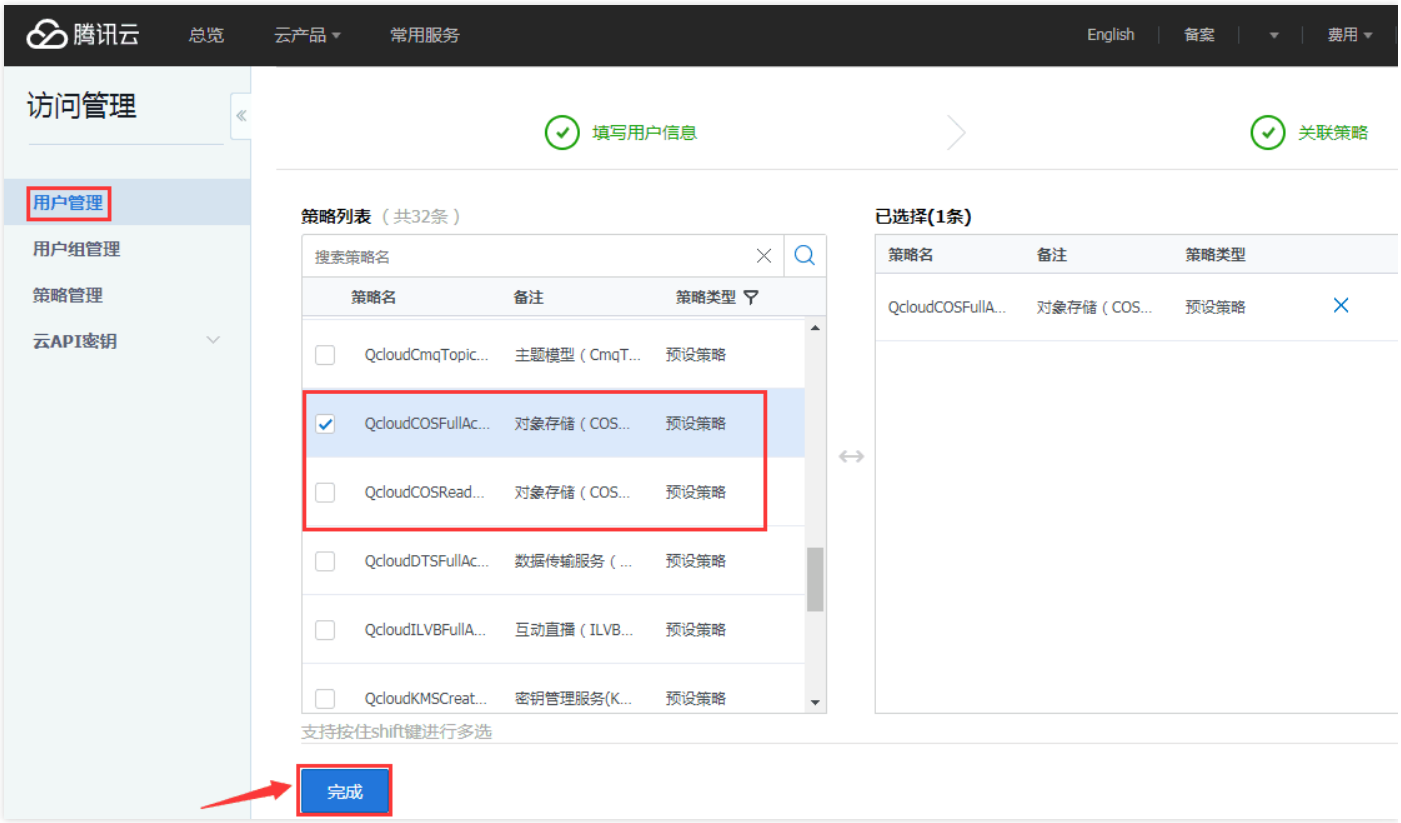
2. 按照要求填写用户相关信息。

注意：

“登录帐号”为可登录腾讯云的账号，可添加三种类型账号为子账号，添加方式如下：

- 邮箱：请输入已注册腾讯云的邮箱或对应账号 ID；
- 微信公众号：请输入其在腾讯云的账号 ID；
- QQ号：请输入QQ号或其账号 ID。

3. 根据系统提供的策略选择，可配置简单的策略，如 COS 的读写权限，只读权限等。如需配置更复杂的策略，可参考 步骤二：对子账号授予权限。



步骤二：对子账号授予权限

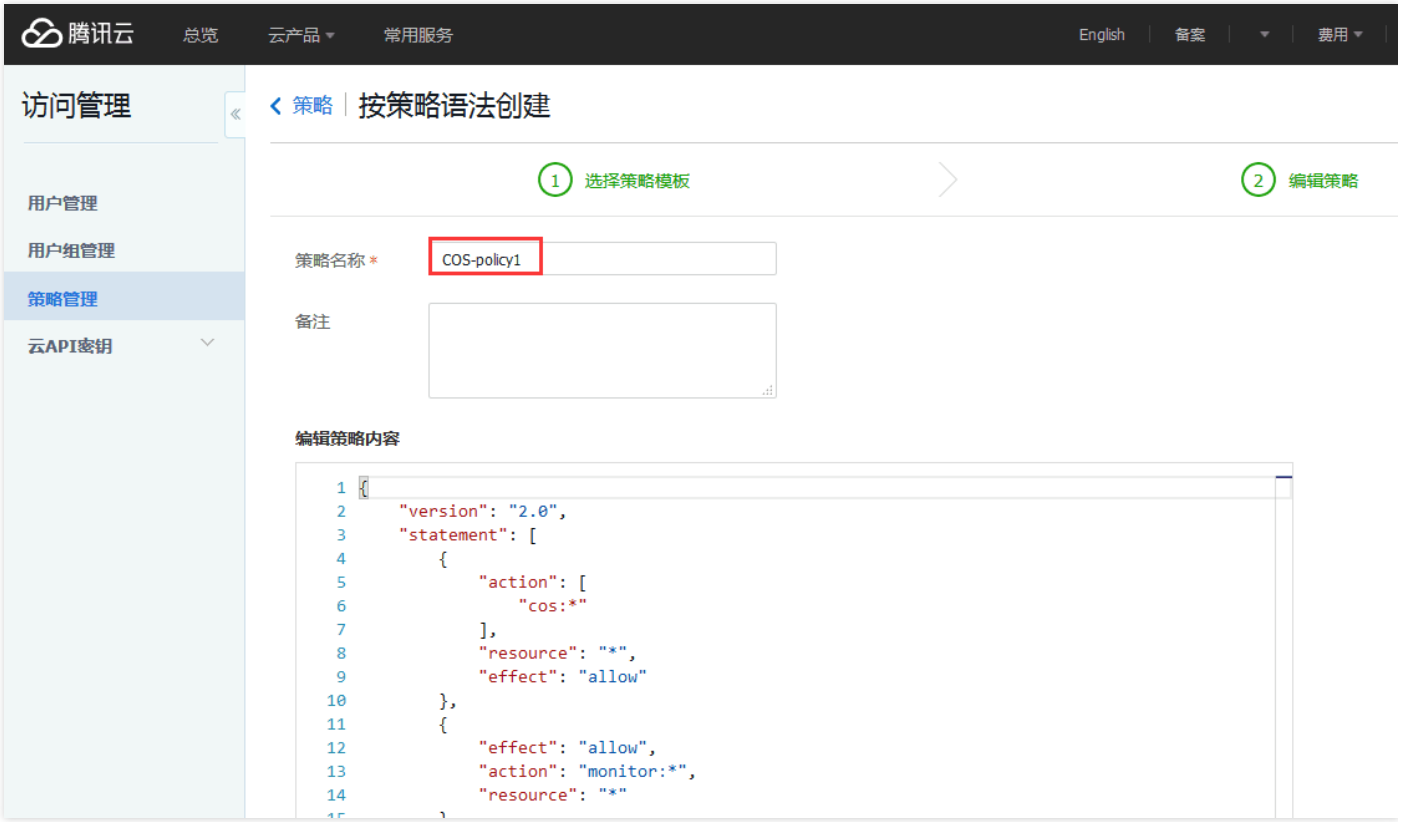
对子账号授予权限可通过 CAM，对子账号（用户）或用户组进行策略配置。

1. 登录 CAM 控制台，单击左侧菜单栏【策略管理】，单击标签页【自定义策略】，单击按钮【新建自定义策略】：

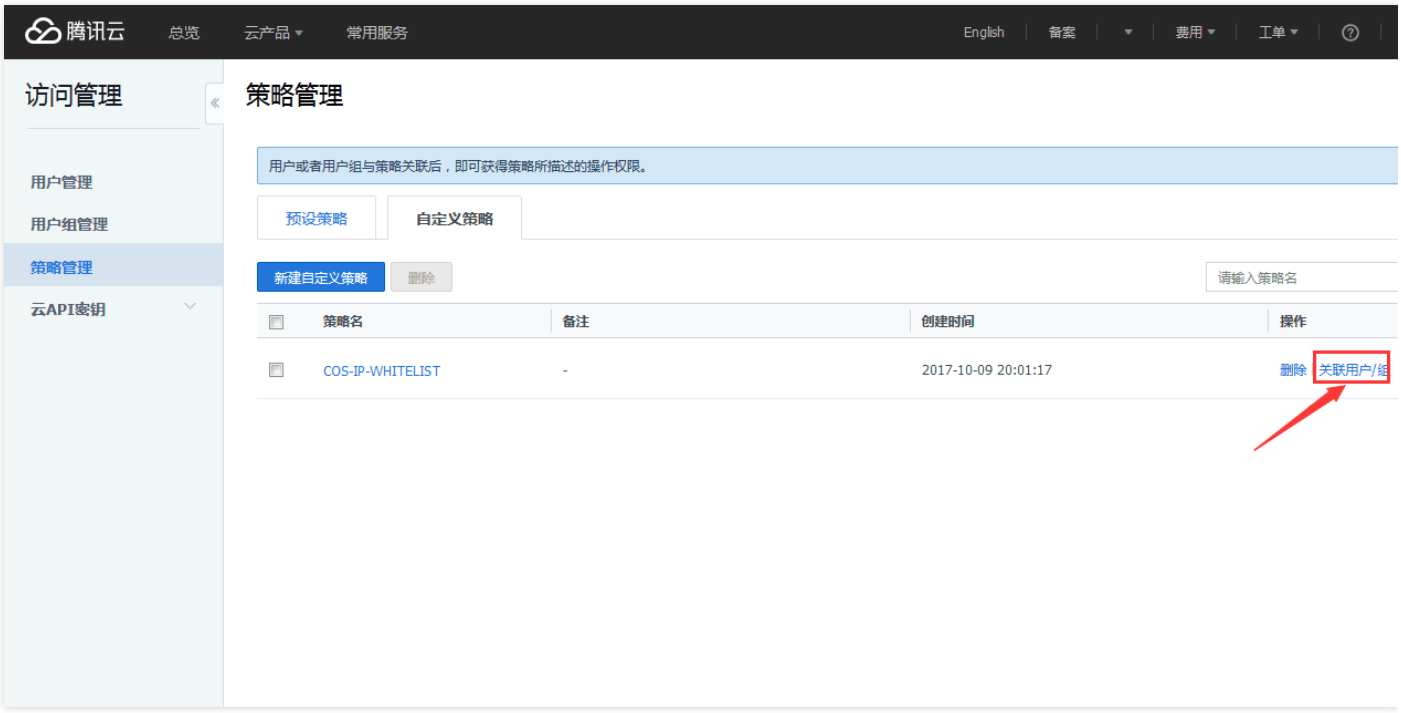
2. 选择【按策略语法创建】，进入创建页面。

3. 可供选择的模版有空白模板和已有的 COS 模板，您可以根据实际情况进行选择。

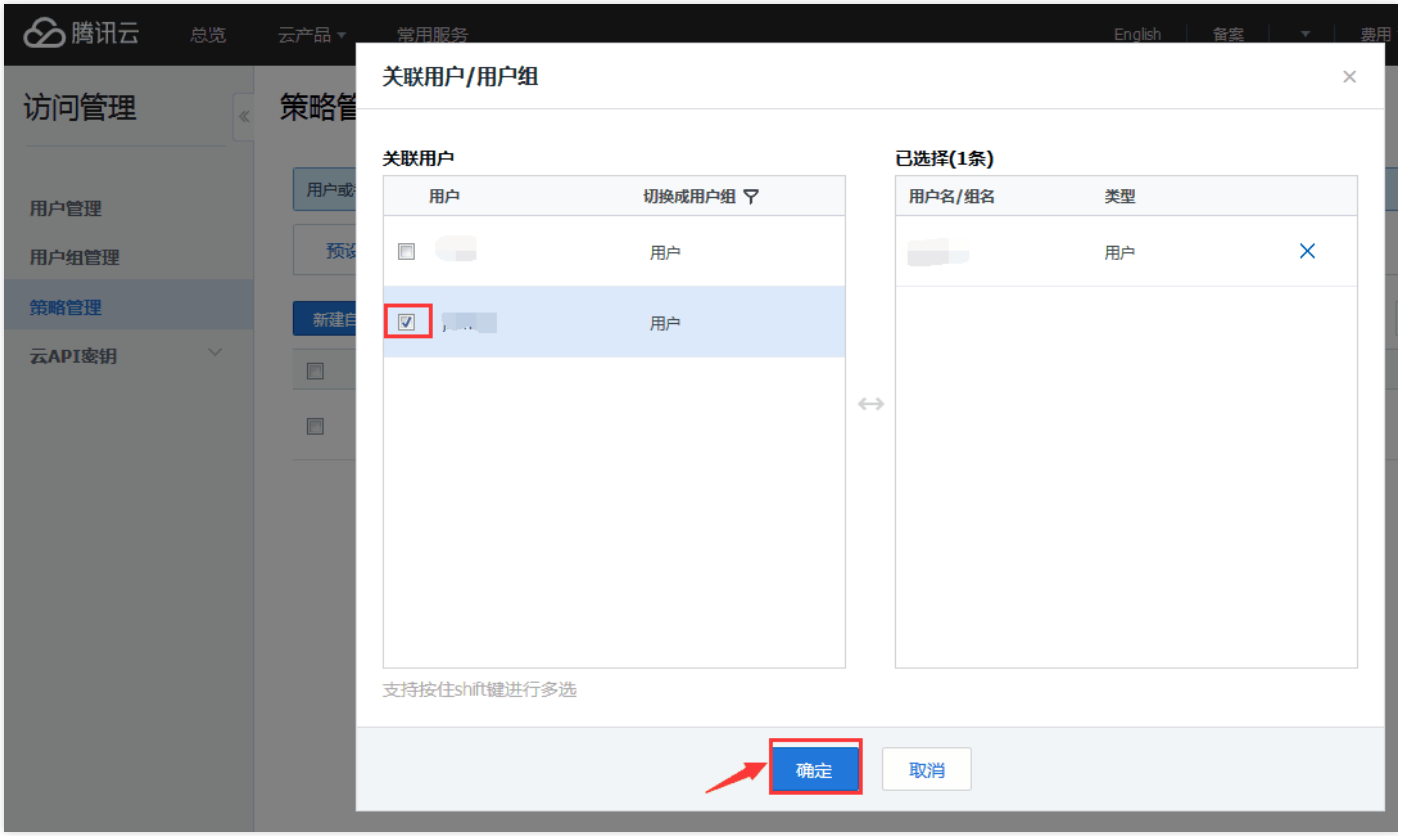
4. 编辑策略，COS 常见场景的策略请参见 CAM 产品文档 的商用案例部分。您可将策略内容复制粘贴到【编辑策略内容】输入框内。



5. 创建完成后，可将策略关联到子账户。



完成子账户授权，子账号即可根据权限范围访问 COS 资源。



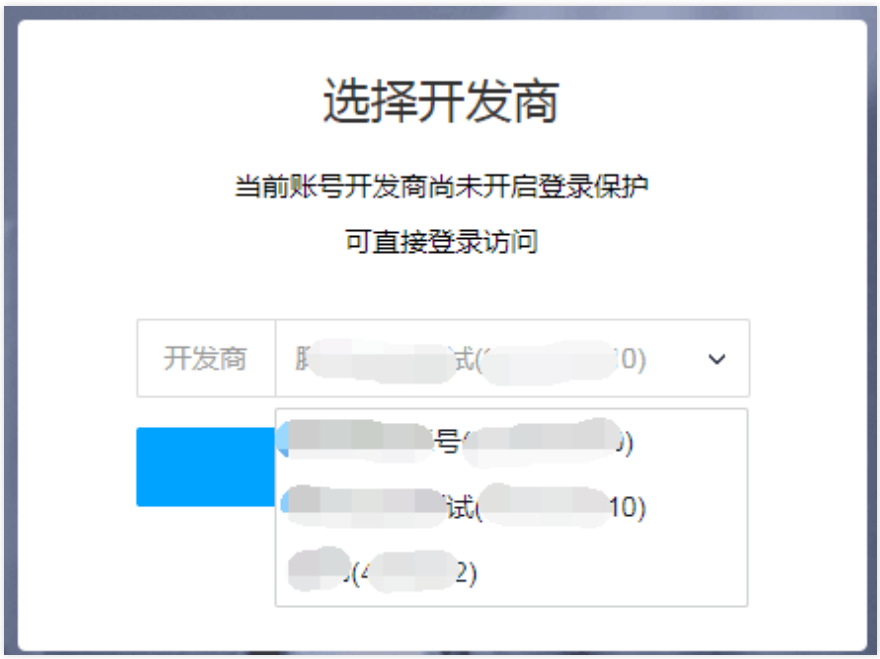
本文还通过以下策略示例说明几种典型场景，详情参见 策略示例：

- 使用 COS 对象存储时，如何给子账户配置读写权限？
- 使用 COS 对象存储时，如何给子账户配置只读权限？
- 使用 COS 对象存储时，如何仅对某 IP 段设置读写权限？

步骤三：子账号访问根账号 COS 资源

COS 访问（API/SDK）需要如下资源：APPID、SecretId、SecretKey。
当使用子账号访问 COS 资源时，需要使用根账号的 APPID，子账号的 SecretId 和 SecretKey，您可以在访问管理控制台创建子账号的 SecretId 和 SecretKey。

1. 子账号登录 访问管理控制台 时，需选择对应的根账号（开发商账号）。



2. 登录后，单击按钮【新建密钥】，即可创建子账号的 SecretID 和 SecretKey，APPID 需由根账号提供。

注意：

- 子账号需通过 XML API 或基于 XML API 的 SDK 访问 COS 资源。
- 子账号访问 COS 资源时，需要使用根账号的 APPID，子账号的 SecretId 和 SecretKey。

基于 XML 的 JAVA SDK 访问示例

以基于 XML 的 JAVA SDK 命令行为例，需填入参数如下：

```
// 1 初始化身份信息
COSCredentials cred = new BasicCOSCredentials("<主账号APPID>", "<子账号SecretId>", "<子账号SecretKey>");
```

实例如下：

```
// 1 初始化身份信息
COSCredentials cred = new BasicCOSCredentials("1250000011", "AKIDasdfmRxHPa9oLhJp9wqwraCdtzvfPasdfaqW", "e8Sdeasdfas2238ViAXjpkU6XloeN2Wpxue");
```

COSCMD 命令行工具访问示例

以 COSCMD `config` 命令行为例，需填入参数如下：

```
coscmd config -u <主账号APPID> -a <子账号SecretId> -s <子账号SecretKey> -b <主账号bucketname> -r <主账号bucketname地域>
```

实例如下：

```
coscmd config -u 1250000011 -a AKIDasdfmRxHPa9oLhJp9wqwraCdtzvfPasdfaqW -s e8Sdeasdfas2238ViAXjpkU6XloeN2Wpxue -b bucket1 -r ap-beijing
```

策略示例

在配置自定义策略时，您可将以下参考策略复制粘贴至输入框【编辑策略内容】，根据实际配置修改即可。

为子账户配置读写权限

为子账户仅配置读写权限，具体策略如下所示：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:*"
],
"resource": "*",
"effect": "allow"
},
{
"effect": "allow",
```

```
"action": "monitor:*",
"resource": "*"
}
]
}
```

为子帐户配置只读权限

为子账户仅配置只读权限，具体策略如下所示：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:List*",
"name/cos:Get*",
"name/cos:Head*",
"name/cos:OptionsObject"
],
"resource": "*",
"effect": "allow"
},
{
"effect": "allow",
"action": "monitor:*",
"resource": "*"
}
]
}
```

为子账户配置某 IP 段的读写权限

本示例中限制仅 IP 网段为 `192.168.1.0/24` 和 `192.168.2.0/24` 的地址具有读写权限，如下所示。
更丰富的生效条件填写，可查看 生效条件。

```
{
"version": "2.0",
"statement": [
{
"action": [
cos:*
],
"resource": "*",
"effect": "allow",
```

```
"condition": {
"ip_equal": {
"qcs:ip": ["192.168.1.0/24", "192.168.2.0/24"]
}
}
}
]
}
```

权限设置相关案例

最近更新时间：2017-11-10 15:00:04

更多权限设置相关案例，请参考以下文档：

授权子账号拥有该账号下 COS 资源的所有权限

授权子账号对特定目录的所有权限

授权子账号对特定目录内文件的读权限

授权子账号对特定文件的读写权限

授权子账号拥有 COS 资源的读权限

授权子账号对特定目录下所有文件的读写权限并禁止对该目录下指定文件的读写权限

授权子账号对指定前缀的文件的读写权限

授权所有用户对指定文件的读写权限

授权跨账号对指定文件的读写权限

授权跨账号的子账号对指定文件的读写权限

临时密钥生成及使用指引

最近更新时间：2018-09-07 09:45:44

获取临时密钥

CAM 的 STS 临时密钥，以云 API 的形式提供。目前云 API 提供各种语言的 SDK，用户可以使用云 API 或 SDK 来申请临时密钥三元组。

通过 API 获取

STS 云 API 接口参数

STS 云 API 的接口参数说明 如下：

字段	是否必选	类型	描述
name	是	String	存储桶名称
policy	是	String	策略语法
durationSeconds	否	Int	指定临时证书的有效期，单位：秒。 不传的话默认 1800 秒, 最长有效期: 2 小时（7200 秒）
Signature	是	String	请求签名，用来验证此次请求的合法性，需要用户根据实际的输入参数计算得出。 计算方法可参考 签名方法 章节
Timestamp	是	Int	时间戳
Nonce	是	Int	随机数
SecretId	是	String	在云 API 密钥上申请的标识身份的 SecretId，一个 SecretId 对应唯一的 SecretKey，而 SecretKey 会用来生成请求签名 Signature
Action	是	String	Action = GetFederationToken
Region	否	String	地域参数，用来标识希望操作哪个地域的实例

返回值

字段	类型	描述
credentials	Object	对象里面包含 Token，tmpSecretId，tmpSecretKey 三元组
--token	String	Token 值做鉴权时使用

字段	类型	描述
--tmpSecretId	String	tmpSecretId 签名时使用
--tmpSecretKey	String	tmpSecretKey 签名时使用
federatedUser	String	返回标识用户。 示例： qcs::sts::123456789012:federated-user/Bob
expiredTime	Int	证书无效的时间戳

访问请求示例

```
https://sts.api.qcloud.com/v2/index.php?Action=GetFederationToken&Nonce=652650920&Region=gz&RequestClient=SDK_JAVA_1.3&SecretId=SecretIDXXXXX&Signature=Bv4G9gCkDVy/lhiDHg2eOlo1PPI=&Timestamp=1494561662&name=Sevenyou&policy=eyJzdGF0ZW1lbnQiOiBbeyJhY3Rpb24iOiBbIm5hbWUvY29zOkdldE9iamVjdCIsIm5hbWUvY29zOlB1dE9iamVjdCJdLCJlZmZlY3QiOiAiYWxsb3ciLCJyZXNvdXJjZSI6WyJxY3M6OmNvczpjbi1ub3J0aDp1aWQvMTI1MjQ0ODcwMzpwcmVmaXgvLzEyNTI0NDg3MDMvcmFiYml0bGl1dGovKiJdfV0sInZlcnNpb24iOiAiMi4wIn0=
```

返回内容示例

```
{
"codeDesc": "Success",
"message": "",
"data": {
"expiredTime": 1494563462,
"credentials": {
"sessionToken": "sessionTokenXXXXX",
"tmpSecretId": "tmpSecretIdXXXXX",
"tmpSecretKey": "tmpSecretKeyXXXXX"
}
},
"code": 0
}
```

通过 SDK 获取

以云 API 提供的 Java SDK 为例：

```
import com.qcloud.Utilities.Json.JSONObject;

public class Demo {
public static void main(String[] args) {
```

```
/* 如果是循环调用下面举例的接口，需要从此处开始您的循环语句。切记！ */
TreeMap<String, Object> config = new TreeMap<String, Object>();
config.put("SecretId", "SecretIdAAAA");
config.put("SecretKey", "SecretKeyZZZZ");

/* 请求方法类型 POST、GET */
config.put("RequestMethod", "GET");

/* 区域参数，可选: gz: 广州; sh: 上海; hk: 香港; ca: 北美; 等。 */
config.put("DefaultRegion", "gz");

QcloudApiModuleCenter module = new QcloudApiModuleCenter(new Sts(),
config);

TreeMap<String, Object> params = new TreeMap<String, Object>();
/* 将需要输入的参数都放入 params 里面，必选参数是必填的。 */
/* DescribeInstances 接口的部分可选参数如下 */
params.put("name", "sevenyou");
String policy = "{\"statement\": [{\"action\": [\"name/cos:GetObject\",\"name/cos:PutObject\"],\"effect
\": \"allow\",\"resource\":[\"qcs::cos:ap-beijing:uid/12345678910:prefix//12345678910/sevenyou/*
\"]}],\"version\": \"2.0\"}";
params.put("policy", policy);

/* 在这里指定所要用的签名算法，不指定默认为 HmacSHA1*/
//params.put("SignatureMethod", "HmacSHA256");

/* generateUrl 方法生成请求串, 可用于调试使用 */
System.out.println(module.generateUrl("GetFederationToken", params));
String result = null;
try {
/* call 方法正式向指定的接口名发送请求，并把请求参数 params 传入，返回即是接口的请求结果。 */
result = module.call("GetFederationToken", params);
JSONObject json_result = new JSONObject(result);
System.out.println(json_result);
} catch (Exception e) {
System.out.println("error..." + e.getMessage());
}
}
}
```

申请临时三元组时，需要描述策略 Policy，示例中以 IP 做限制的 Policy 可能如下：

```
{
"statement": [
{
"action": [
```

```
"name/cos:GetObject",
"name/cos:HeadObject"
],
"condition": {
"ip_equal": {
"qcs:ip": [
"101.226.226.185/32"
]
}
},
"effect": "allow",
"resource": [
"qcs::cos:cn-east:uid/12345678910:prefix//12345678910/sevenyou/*"
]
}
],
"version": "2.0"
}
```

Policy 描述请参考 策略语法。

> **注意：**
> resource 字段中的 prefix 后跟 // 。uid后面跟的是appid，而不是UIN。

使用临时密钥访问 COS

COS 访问通过 x-cos-security-token 字段来传递临时 Token，而临时 SecretId 和 SecretKey 则用来生成密钥，以 Jave SDK 为例使用临时密钥访问 COS，示例如下：
从 Github 下载：Java SDK

> **注意：**
> 下列代码，需要加入 x-cos-security-token 字段，传递 STS 的临时密钥。

```
public class Demo {
public static void main(String[] args) throws Exception {
// 用户基本信息
String appid = "12345678910";
String secret_id = "TmpSecretId";
String secret_key = "TmpSecretKey";
```

```
String sessionToken = "TmpToken";

// 设置密钥
COSCredentials cred = new BasicCOSCredentials(appid, secret_id, secret_key);
// 设置区域, 这里设置为华北
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing");
// 生成 cos 客户端对象
COSClient cosClient = new COSClient(cred, clientConfig);

// 创建 bucket
// bucket 数量上限 200 个, bucket 是重操作, 一般不建议创建如此多的 bucket
// 重复创建同名 bucket 会报错
String bucketName = "rabbitliutj";
// 上传 object, 建议 20M 以下的文件使用该接口
File localFile = new File("D:\\test\\rabbit_test.txt");
String key = "/upload_single_demo.txt";
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, localFile);
ObjectMetadata objectMetadata = new ObjectMetadata();
objectMetadata.setSecurityToken (sessionToken);
putObjectRequest.setMetadata(objectMetadata);
PutObjectResult putObjectResult = cosClient.putObject(putObjectRequest);
System.out.println(putObjectResult);

// 关闭客户端 (关闭后台线程)
cosClient.shutdown();
}
}
```

请求速率与性能优化

最近更新时间：2017-12-27 21:23:08

本文探讨请求速率性能优化在腾讯云 COS 对象存储上的最佳实践。

腾讯云对象存储提供的典型工作负载能力为每秒 100 个 PUT/LIST/DELETE 等请求，或者每秒 400 个 GET 请求。如果您的工作负载超过了上述能力，建议您遵循本指南实现请求速率的性能扩展和优化。

> **注意：**
> 请求负载指的是每秒发起的请求数量，非并发连接数。即您在保持数千连接数的同时，仍可以在 1s 时间内发送数百个新连接请求。

腾讯云对象存储支持性能扩展，以支持更高的请求速率。如果您的请求是高 GET 请求负载，建议您搭配腾讯云 CDN 产品进行使用，请参考有关 COS 搭配 CDN 的最佳实践。如果预计存储桶的请求速率会超过每秒 500 个 PUT/LIST/DELETE 请求，建议您 联系我们，以便于为工作负载做好准备，避免遇到请求的限制。

> **注意：**
> 如果您的混合请求负载只是偶尔达到每秒 500 个，并在突发时不超过每秒 800 个，您可无需遵循本指南。

混合请求负载

当需要上传大量对象的时候，您选择的对象键可能会引发性能问题，以下将简述腾讯云对象存储对 Object 键值的存储方法。

腾讯云在对象存储的每一个服务地域都维护了存储桶 (Bucket) 和对象 (Object) 的键值作为索引，对象键以 UTF-8 二进制顺序保存在索引的多个分区中。对于大量的键值，例如使用时间戳或者字母顺序可能会耗尽键值所在分区的读写性能，以存储桶路径 `testbucket-123454321.cos.ap-beijing.myqcloud.com` 为例，以下列出了可能会耗尽索引性能的一些案例：

```
20170701/log120000.tar.gz
20170701/log120500.tar.gz
20170701/log121000.tar.gz
20170701/log121500.tar.gz
...
image001/indexpage1.jpg
image002/indexpage2.jpg
```

```
image003/indexpage3.jpg
...
```

如果您的业务典型负载超过每秒 500 个请求，则应避免使用上述案例中的顺序键值。当您的业务必须使用顺序编号或日期时间等字符作为对象键时，您可以使用一些方法向对象键名称添加随机前缀，即可实现在多个索引分区实现键值管理，提升集中负载的性能。以下提供了一些在键值中增加随机性的方法。

以下提供的所有方法，均是有可能提升单个存储桶访问性能的方法，如果您的业务典型负载超过每秒 500 个请求，在执行以下方法的同时，您仍需 联系我们 以便于为您的业务负载提前做好准备。

添加十六进制哈希前缀

最直接的增加对象键随机性的方式，就是在对象键名称的最前面添加哈希字符串作为前缀，例如可以在上传对象时，对路径键值进行 SHA1 或 MD5 哈希计算，并选取几位字符作为前缀添加到键值名称，通常 2~4 位的字符哈希前缀可以满足需要。

```
faf1-20170701/log120000.tar.gz
e073-20170701/log120500.tar.gz
333c-20170701/log121000.tar.gz
2c32-20170701/log121500.tar.gz
```

> **注意：**
> 由于腾讯云对象存储的键值索引是以 UTF-8 二进制顺序作为索引的，因此在执行列出对象（GET Bucket）操作时，您可能需要发起 65536 次列出对象操作，才能得到原来完整的 20170701 前缀结构。

添加枚举值前缀

如果您仍想要保留对象键的检索易用性，您可以针对您的文件类型枚举出一些前缀，实现对象分组，相同枚举值的前缀将共享所在索引分区的性能。

```
logs/20170701/log120000.tar.gz
logs/20170701/log120500.tar.gz
logs/20170701/log121000.tar.gz
...
images/image001/indexpage1.jpg
images/image002/indexpage2.jpg
images/image003/indexpage3.jpg
...
```

如果您在相同枚举前缀下，仍然有较高的访问负载，持续超过每秒 500 个请求，您可以参照上述添加十六进制哈希前缀的方式，在枚举值后继续添加哈希前缀执行多个索引分区，以实现更高性能的读写。

```
logs/faf1-20170701/log120000.tar.gz
logs/e073-20170701/log120500.tar.gz
logs/333c-20170701/log121000.tar.gz
...
images/0165-image001/indexpage1.jpg
images/a349-image002/indexpage2.jpg
images/ac00-image003/indexpage3.jpg
...
```

反转键值名称字符串

当您的业务不得不使用连续递增的 ID 或日期，或需要一次性上传大量的连续前缀对象时，如下述典型用法：

```
20170701/log0701A.tar.gz
20170701/log0701B.tar.gz
20170702/log0702A.tar.gz
20170702/log0702B.tar.gz
...
id16777216/album/hongkong/img20170701121314.jpg
id16777216/music/artist/tony/anythinggoes.mp3
id16777217/video/record20170701121314.mov
id16777218/live/show/date/20170701121314.mp4
...
```

如上的键值命名方式极易耗尽 `2017` 与 `id` 为前缀的键值所在的索引分区，此时将键值前缀的一部分反转后，则实现了一定的随机性。

```
10707102/log0701A.tar.gz
10707102/log0701B.tar.gz
20707102/log0702A.tar.gz
20707102/log0702B.tar.gz
...
61277761di/album/hongkong/img20170701121314.jpg
61277761di/music/artist/tony/anythinggoes.mp3
71277761di/video/record20170701121314.mov
81277761di/live/show/date/20170701121314.mp4
...
```

高 GET 请求负载

如果主要的业务负载是 GET 请求（即下载请求）为主，除了建议遵守上述准则外，还建议您搭配腾讯云 CDN 服务使用。

腾讯云 CDN 利用遍布全国和全球的边缘加速节点，向用户分发内容时可降低延时并提高速度，对于热点文件可以支持预拉热的方式进行缓存，从而减少回源到 COS 的 GET 请求数。请参考有关 COS 搭配 CDN 的最佳实践。

配置自定义域名支持 HTTPS 访问

最近更新时间：2018-05-24 17:36:26

用户可通过自有域名（自定义域名，如 test.cos.com ）访问存储桶（Bucket）下的对象（Object）。具体操作指引如下：

- 开启 CDN 加速时配置自定义域名支持 HTTPS 访问
- 关闭 CDN 加速时配置自定义域名支持 HTTPS 访问

开启 CDN 加速

一、绑定自定义域名

将存储桶绑定到您的自有域名，开启 CDN 加速。操作指引参考 域名管理--自定义域名。

二、配置 HTTPS 访问

在 CDN 控制台进行 HTTPS 配置，操作指引参考 HTTPS 配置。

关闭 CDN 加速

本章节主要以示例的形式介绍在 COS 中通过反向代理配置自定义域名（关闭 CDN 加速）支持 https 访问的操作步骤。本示例将实现不开启 CDN 加速的情况下，直接通过自定义域名 https://test.cos.com 访问所属地域为华南、名称为 testhttps-12345678 的存储桶，具体操作步骤如下：

一、绑定自定义域名

将存储桶 testhttps 绑定到域名 https://test.cos.com ，关闭 CDN 加速。操作指引参考 域名管理--自定义域名。

二、为域名配置反向代理

在服务器上为域名 https://test.cos.com 配置反向代理。具体配置参考如下（以下 Nginx 配置仅供参考）：

```
server {
listen 443;
server_name test.cos.com ;

ssl on;
ssl_certificate /usr/local/nginx/conf/server.crt;
ssl_certificate_key /usr/local/nginx/conf/server.key;

error_log logs/test.cos.com.error_log;
```

```
access_log logs/test.cos.com.access_log;
location / {
root /data/www/;
proxy_pass http://testhttps-12345678.cos.ap-guangzhou.myqcloud.com; //配置存储桶（Bucket）的默认下载域名
}

}
```

其中 server.crt; 、 server.key 是您的自有（自定义）域名的 HTTPS 证书。若您的域名还没有 HTTPS 证书 ，可以在 腾讯云 SSL 证书 页面申请。
若暂时没有证书，可以删除以下配置信息，但访问时会出现告警，单击继续即可访问：

```
ssl on;
ssl_certificate /usr/local/nginx/conf/server.crt;
ssl_certificate_key /usr/local/nginx/conf/server.key;
```

三、解析域名到服务器

在您域名的 DNS 解析服务商处解析您的域名。若您使用的是腾讯云云解析，请前往 云解析控制台，将域名 test.cos.com 解析到步骤二中的服务器的 IP 上，指引参考 域名解析。

进阶配置

通过浏览器直接打开网页

在配置好自定义域名支持 HTTPS 访问后，就可以通过您的域名下载存储桶（Bucket）中的对象（Object）了。若根据业务需要，想直接在浏览器中访问网页、图片等，可通过静态网站功能实现。操作指引参考 静态网站设置。

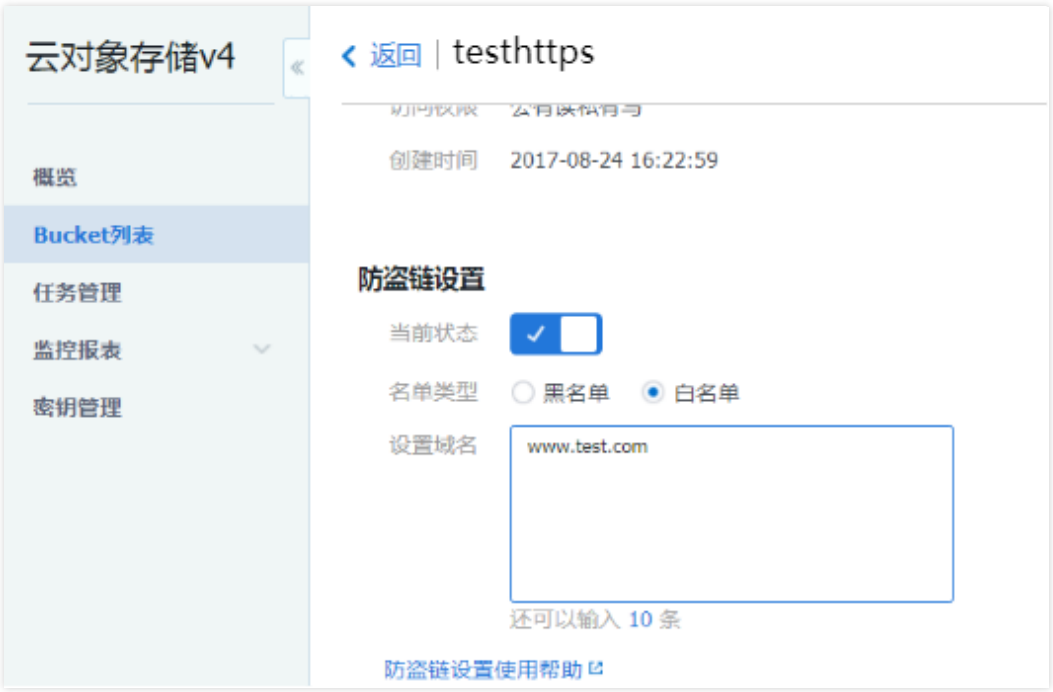
配置完成后，在 Nginx 配置中增加一行信息，重启 Nginx，刷新浏览器缓存即可。

```
proxy_set_header Host $http_host;
```

配置 refer 防盗链

若存储桶（Bucket）是公有的，会有被盗链的风险。用户可以通过防盗链设置，开启 Referer 白名单，防止被恶意盗链。具体操作步骤如下：

1. 在COS 控制台 开启防盗链设置功能，选择白名单。操作指引参考 防盗链设置



2. 在 Nginx 配置中增加一行信息，再重启 Nginx，刷新浏览器缓存。

```
proxy_set_header Referer www.test.com;
```

3. 设置完成后，直接打开文件会报错： errorcode：-46616 ；错误提示：未命中 refer 白名单。但是通过代理访问自定义域名，可以正常打开网页。

```
{
    errorcode: -46616,
    errormsg: "not hit white refer, retcode:-46616"
}
```

Web 端直传实践

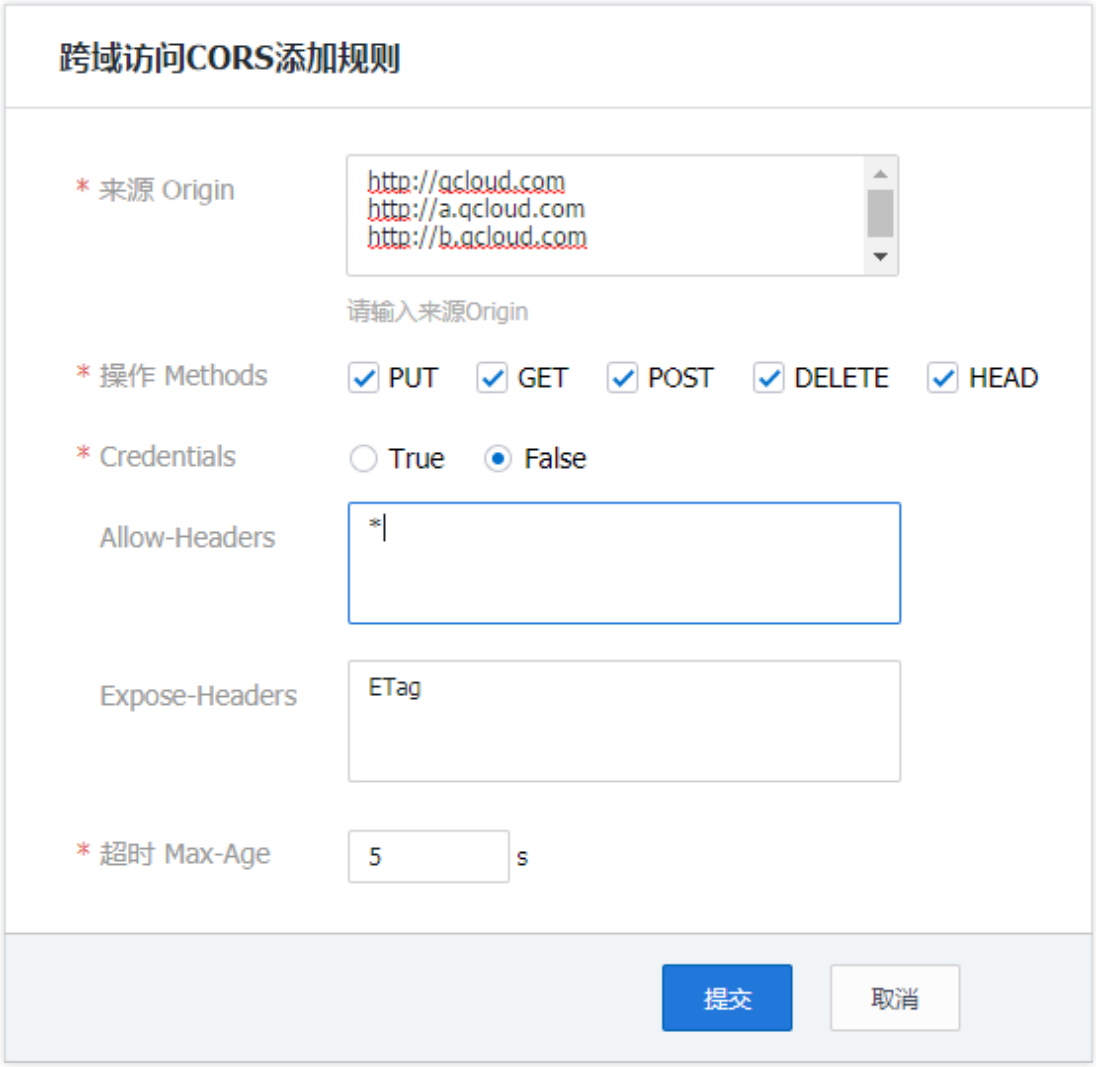
最近更新时间：2018-05-15 10:02:01

本文档介绍如何不依赖 SDK，用简单的代码，在网页（Web 端）直传文件到 COS 的存储桶。

> 本文档内容基于 XML API 。

一、前期准备

1. 登录 COS 控制台 并创建存储桶，得到 Bucket（存储桶名称） 和 Region（地域名称）。
2. 登录 控制台密钥管理 获取您的项目 SecretId 和 SecretKey。
3. 在 COS 控制台，进入新建的存储桶，单击【基础配置】，配置 CORS 规则，配置示例如下图：



二、计算签名

签名计算放在前端会暴露 SecretKey，因此我们把签名计算过程放在后端实现，前端通过 AJAX 向后端获取签名结果，正式部署时请在后端加一层您的网站本身的权限检验。
指引参考 PHP 和 Node.js 的签名示例，其他语言请参照对应的 XML SDK 文档。

三、前端上传

方案 A：使用 AJAX 上传

AJAX 上传需要浏览器支持基本的 HTML5 特性，当前方案使用的是 XML API 的 PutObject 接口，操作指引：

1. 按照 步骤一、前期准备 的步骤，准备好存储桶。
2. 创建 test.html 文件，修改下方代码的 Bucket 和 Region，复制到 test.html 文件。
3. 部署好后端的签名服务，并修改 test.html 里的签名服务地址。
4. 把 test.html 放在 Web 服务器下，然后在浏览器访问页面，测试文件上传功能。

```
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Ajax Put 上传</title>
<style>
h1, h2 {
font-weight: normal;
}

#msg {
margin-top: 10px;
}
</style>
</head>
<body>

<h1>Ajax Put 上传</h1>

<input id="fileSelector" type="file">
<input id="submitBtn" type="submit">

<div id="msg"></div>

<script>
```

```
(function () {
// 请求用到的参数
var Bucket = 'test-1250000000';
var Region = 'ap-guangzhou';
var protocol = location.protocol === 'https:' ? 'https:' : 'http:';
var prefix = protocol + '//' + Bucket + '.cos.' + Region + '.myqcloud.com/';

// 计算签名
var getAuthorization = function (options, callback) {
var method = (options.Method || 'get').toLowerCase();
var key = options.Key || '';
var pathname = key.indexOf('/') === 0 ? key : '/' + key;
// var url = 'http://127.0.0.1:3000/sts-auth' +
var url = '../server/sts-auth.php' +
'?method=' + method +
'&pathname=' + encodeURIComponent(pathname);
var xhr = new XMLHttpRequest();
xhr.open('GET', url, true);
xhr.onload = function (e) {
var AuthData;
try {
AuthData = JSON.parse(xhr.responseText)
} catch (e) {}
if (AuthData && AuthData.Authorization) {
callback(null, {
Authorization: AuthData.Authorization,
XCosSecurityToken: AuthData.XCosSecurityToken,
});
} else {
console.error(AuthData);
callback('获取签名出错');
}
};
xhr.onerror = function (e) {
callback('获取签名出错');
};
xhr.send();
};

// 上传文件
var uploadFile = function (file, callback) {
var Key = 'dir/' + file.name; // 这里指定上传目录和文件名
getAuthorization({Method: 'PUT', Key: Key}, function (err, info) {

if (err) {
alert(err);
```

```
return;
}

var auth = info.Authorization;
var XCosSecurityToken = info.XCosSecurityToken;
var url = prefix + Key;
var xhr = new XMLHttpRequest();
xhr.open('PUT', url, true);
xhr.setRequestHeader('Authorization', auth);
XCosSecurityToken && xhr.setRequestHeader('x-cos-security-token', XCosSecurityToken);
xhr.onload = function () {
if (xhr.status === 200 || xhr.status === 206) {
var ETag = xhr.getResponseHeader('etag');
callback(null, {url: url, ETag: ETag});
} else {
callback('文件 ' + Key + ' 上传失败，状态码：' + xhr.status);
}
};
xhr.onerror = function () {
callback('文件 ' + Key + ' 上传失败，请检查是否没配置 CORS 跨域规则');
};
xhr.send(file);
});
};

// 监听表单提交
document.getElementById('submitBtn').onclick = function (e) {
var file = document.getElementById('fileSelector').files[0];
if (!file) {
document.getElementById('msg').innerText = '未选择上传文件';
return;
}
file && uploadFile(file, function (err, data) {
console.log(err || data);
document.getElementById('msg').innerText = err ? err : ('上传成功，ETag=' + data.ETag);
});
};
})();
</script>

</body>
</html>
```

执行效果如下图：

方案 B：使用 Form 表单上传

Form 表单上传支持低版本的浏览器的上传（如 IE8），当前方案使用的是 XML API 的 PostObject 接口。操作指引：

1. 按照 步骤一、前期准备 的步骤，准备好存储桶。
2. 创建 test.html 文件，修改下方代码的 Bucket 和 Region，复制到 test.html 文件。
3. 部署好后端的签名服务，并修改 test.html 里的签名服务地址。
4. 在 test.html 同一个目录下创建一个空的 empty.html ，用于上传成功时跳转回来。
5. 把 test.html 和 empty.html 放在 Web 服务器下，然后在浏览器访问页面，测试文件上传功能。

```
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Form 表单简单上传</title>
<style>h1, h2 {font-weight: normal;}#msg {margin-top:10px;}</style>
</head>
<body>

<h1>Form 表单简单上传（兼容 IE8）</h1>
<div>最低兼容到 ie6 上传，不支持 onprogress</div>

<form id="form" target="submitTarget" action="" method="post" enctype="multipart/form-data" acc
ept="*/*">
<input id="name" name="name" type="hidden" value="">
<input name="success_action_status" type="hidden" value="200">
<input id="success_action_redirect" name="success_action_redirect" type="hidden" value="">
<input id="key" name="key" type="hidden" value="">
```

```
<input id="Signature" name="Signature" type="hidden" value="">
<input name="Content-Type" type="hidden" value="">
<input id="x-cos-security-token" name="x-cos-security-token" type="hidden" value="">
<input id="fileSelector" name="file" type="file">
<input id="submitBtn" type="button" value="提交">
</form>
<iframe id="submitTarget" name="submitTarget" style="display:none;" frameborder="0"></iframe>

<div id="msg"></div>

<script>
(function () {

// 请求用到的参数
var Bucket = 'test-1250000000';
var Region = 'ap-guangzhou';
var protocol = location.protocol === 'https:' ? 'https:' : 'http:';
var prefix = protocol + '//' + Bucket + '.cos.' + Region + '.myqcloud.com/';
var form = document.getElementById('form');
form.action = prefix;

// 计算签名
var getAuthorization = function (options, callback) {
var method = (options.Method || 'get').toLowerCase();
// var url = 'http://127.0.0.1:3000/sts-auth' +
var url = '../server/sts-auth.php' +
'?method=' + method +
'&pathname=' + encodeURIComponent('/');
var xhr = new XMLHttpRequest();
xhr.open('GET', url, true);
xhr.onreadystatechange = function (e) {
if (xhr.readyState === 4) {
if (xhr.status === 200) {
var AuthData;
try {
AuthData = (new Function('return ' + xhr.responseText))();
} catch (e) {}
if (AuthData && AuthData.Authorization) {
callback(null, {
Authorization: AuthData.Authorization,
XCosSecurityToken: AuthData.XCosSecurityToken,
});
} else {
console.error(AuthData);
callback('获取签名出错');
}
```

```
} else {
callback('获取签名出错');
}
}
};
xhr.send();
};

// 监听上传完成
var Key;
var submitTarget = document.getElementById('submitTarget');
var showMessage = function (err, data) {
console.log(err || data);
document.getElementById('msg').innerText = err ? err : ('上传成功，ETag=' + data.ETag);
};
submitTarget.onload = function () {
var search;
try {
search = submitTarget.contentWindow.location.search.substr(1);
} catch (e) {
showMessage('文件 ' + Key + ' 上传失败');
}
if (search) {
var items = search.split('&');
var i, arr, data = {};
for (i = 0; i < items.length; i++) {
arr = items[i].split('=');
data[arr[0]] = decodeURIComponent(arr[1] || '');
}
showMessage(null, {url: prefix + Key, ETag: data.etag});
} else {
}
};

// 发起上传
document.getElementById('submitBtn').onclick = function (e) {
var filePath = document.getElementById('fileSelector').value;
if (!filePath) {
document.getElementById('msg').innerText = '未选择上传文件';
return;
}
Key = 'dir/' + filePath.match(/[\\\/]?([^\\\/]+)$/)[1]; // 这里指定上传目录和文件名
getAuthorization({Method: 'POST', Key: Key}, function (err, AuthData) {
// 在当前目录下放一个空的 empty.html 以便让接口上传完成跳转回来
document.getElementById('success_action_redirect').value = location.href.substr(0, location.href.lastInd
exOf('/') + 1) + 'empty.html';
```

```
document.getElementById('key').value = Key;
document.getElementById('Signature').value = AuthData.Authorization;
document.getElementById('x-cos-security-token').value = AuthData.XCosSecurityToken || '';
form.submit();
});
};
})();
</script>

</body>
</html>
```

执行效果如下图：

相关文档

若您有更丰富的接口调用需求，请参考以下 JavaScript SDK 文档：

- JavaScript SDK
- JavaScript SDK（历史版本 API）

设置跨域访问

最近更新时间：2018-08-28 14:30:25

同源策略

同源策略限制从一个源加载的文档或脚本与来自另一个源的资源进行交互的方式，是用于隔离潜在恶意文件的关键安全机制。同协议、同域名（或IP）、以及同端口视为同一个域，一个域内的脚本仅仅具有本域内的权限，即本域脚本只能读写本域内的资源，而无法访问其它域的资源。这种安全限制称为同源策略。

同源的定义

两个页面的协议、域名和端口（若指定了端口）相同，则视为同源。如下表给出了相对 `http://www.example.com/dir/page.html` 的同源检测示例：

URL	结果	原因
`http://www.example.com/dir2/other.html`	成功	协议、域名、端口相同
`http://www.example.com/dir/inner/another.html`	成功	协议、域名、端口相同
`https://www.example.com/secure.html`	失败	协议不同（HTTPS）
`http://www.example.com:81/dir/etc.html`	失败	端口不同（81）
`http://news.example.com/dir/other.html`	失败	域名不同

跨域访问

跨域资源共享（Cross-Origin Resource Sharing，CORS）机制，我们简称为跨域访问，允许 Web 应用服务器进行跨域访问控制，从而使跨域数据传输得以安全进行。CORS 需要浏览器和服务器同时支持。目前，所有浏览器都支持该功能，IE 浏览器要求版本 IE10 或以上。

整个 CORS 通信过程，都是浏览器自动完成，不需要用户参与。对于开发者来说，CORS 通信与同源的 AJAX 通信没有差别，代码完全一样。浏览器一旦发现 AJAX 请求跨源，就会自动添加一些附加的头信息，有时还会多出一次附加的请求，但用户不会有感觉。

因此，实现 CORS 通信的关键是服务器。只要服务器实现了 CORS 接口，就可以跨源通信。

CORS 主要使用场景

CORS 使用一定是在使用浏览器的情况下，因为控制访问权限的是浏览器而非服务器。因此使用其它客户端的时候无需关心任何跨域问题。

CORS 的主要应用是实现无论上传或者下载，在浏览器端使用 AJAX 直接访问 COS 的数据，而无需通过用户本身的应用服务器中转。对于同时使用 COS 和使用 AJAX 技术的网站，建议使用 CORS 来实现与 COS 的直接通信。

COS 对 CORS 的支持

COS 支持对 CORS 规则的配置，从而根据需求允许或者拒绝相应的跨域请求。该 CORS 规则配置属于存储桶级别。
CORS 请求的通过与否和 COS 的身份验证等是完全独立的，即 COS 的 CORS 规则仅仅是用来决定是否附加 CORS 相关的 Header 的一个规则。是否拦截该请求完全由浏览器决定。
目前 COS 的所有 Object 相关接口都提供了 CORS 相关的验证，另外 Multipart 相关的接口目前也已经完全支持 CORS 验证。

> 当运行在同一个浏览器上的分别来源于 www.a.com 和 www.b.com 的两个页面同时请求同一跨域资源时，若 www.a.com 的请求先到达服务器，服务器会将资源带上 Access-Control-Allow-Origin 的 Header，并返回给 www.a.com 的用户。这时 www.b.com 又发起了请求，浏览器会将 Cache 的上一次请求响应返回给用户，Header 的内容和 CORS 的要求不匹配，就会导致 www.b.com 请求失败。

CORS 设置示例

以下简单示例介绍使用 AJAX 从 COS 获取数据的配置步骤。示例中使用的存储桶（Bucket）权限设置为公有（Public），访问权限为私有的存储桶（Bucket）只需要在请求中附加签名，其余配置相同。
以下示例中使用的存储桶名为 corstest，存储桶访问权限为公有读私有写。

设置 CORS 之前

1. **确认文件可正常访问**
   上传一个 test.txt 的文本文档到 corstest。test.txt 的访问地址为 http://corstest-125xxxxxxx.cos.ap-beijing-1.myqcloud.com/test.txt 。
   使用 curl 直接访问该文本文档，请将以下地址替换为您的文件地址：

   ```
   curl http://corstest-125xxxxxxx.cos.ap-beijing-1.myqcloud.com/test.txt
   ```

   返回 test.txt 文件的内容：test。表示该文档可以正常访问。

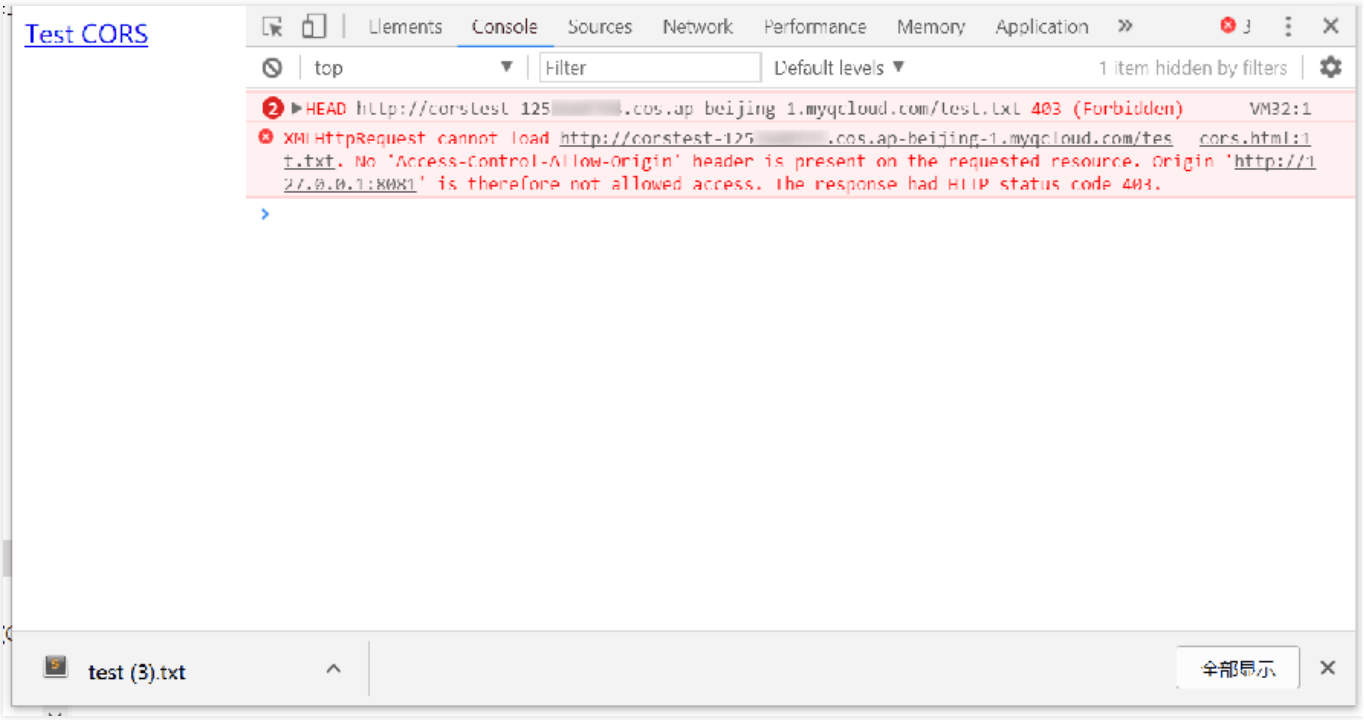
   ```
   [root@VM_139_240_centos ~]# curl http://corstest-125[redacted].cos.ap-beijing-1.myqcloud.com/test.txt
   test
   ```

2. **使用 AJAX 技术访问文件**
   我们现在尝试使用 AJAX 技术来直接访问该 test.txt 文件。

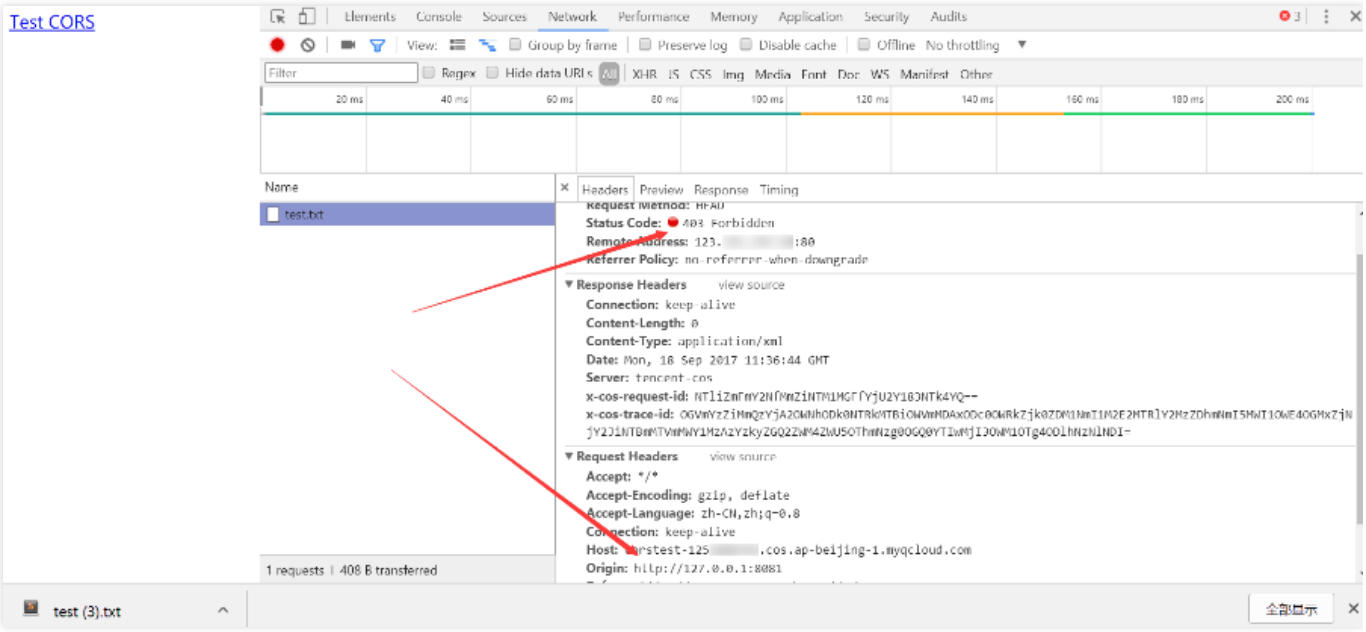
i. 写一个简单的 HTML 文件，将以下代码复制到本地保存成 HTML 文件后使用浏览器打开。因为没有设置自定义的头，因此该请求不需要预检。

```
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
</head>
<body>
<a href="javascript:test()">Test CORS</a>
<script>
function test() {
var url = 'http://corstest-125xxxxxxx.cos.ap-beijing-1.myqcloud.com/test.txt';
var xhr = new XMLHttpRequest();
xhr.open('HEAD', url);
xhr.onload = function () {
var headers = xhr.getAllResponseHeaders().replace(/\r\n/g, '\n');
alert('request success, CORS allow.\n' +
'url: ' + url + '\n' +
'status: ' + xhr.status + '\n' +
'headers:\n' + headers);
};
xhr.onerror = function () {
alert('request error, maybe CORS error.');
};
xhr.send();
}
</script>
</body>
</html>
```

ii. 在浏览器中打开该 HTML 文件，单击【Test CORS】的按钮发送请求后，出现以下错误，错误提示：无权限访问，原因是没有找到 Access-Control-Allow-Origin 这个 Header。显然，这是因为服务器没有配置 CORS。



iii. 访问失败，再进入 Header 界面检查原因，可见浏览器发送了带 Origin 的 Request，因此是一个跨域请求。



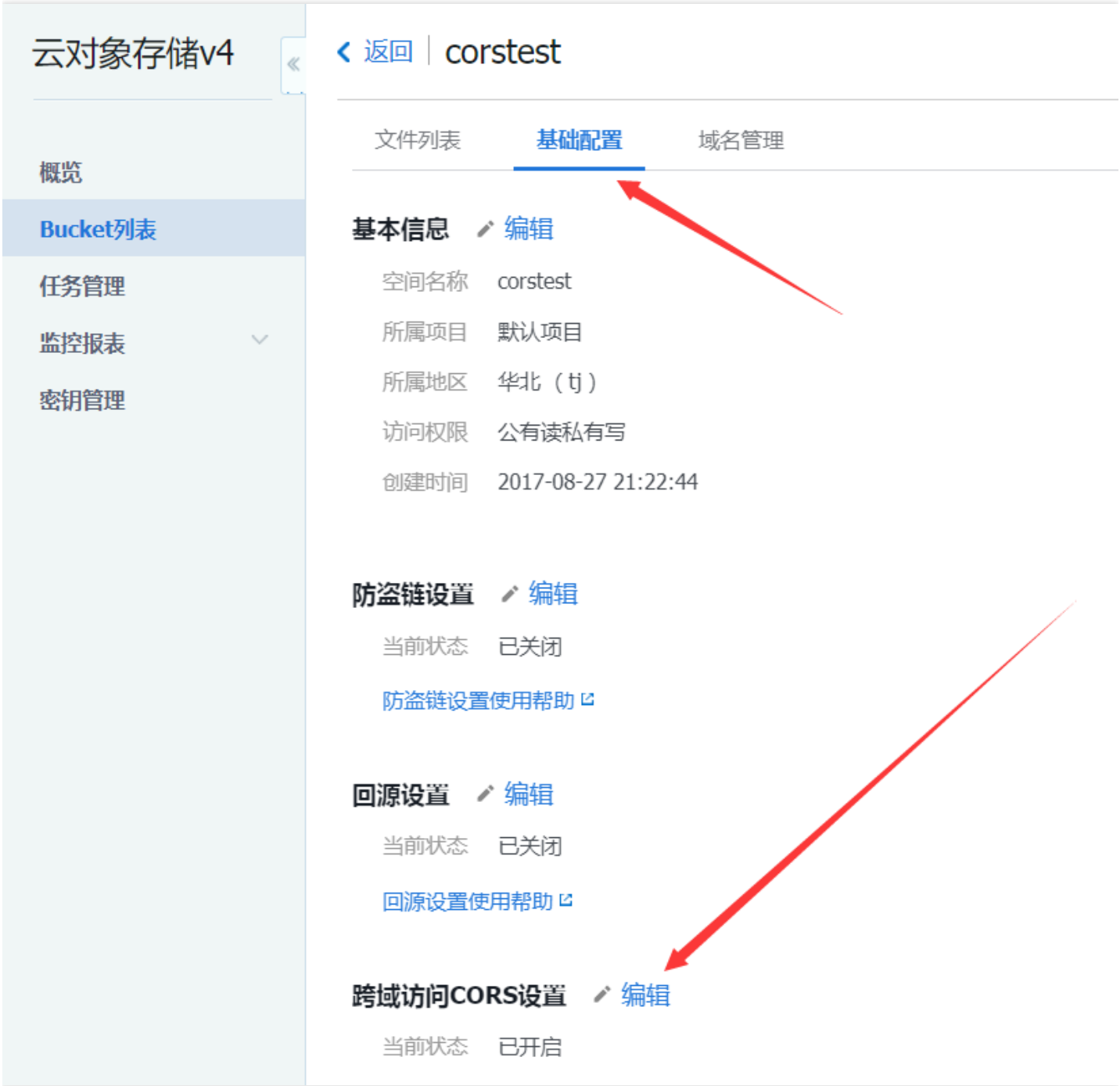
我们将网页搭建在服务器上，地址为 http://127.0.0.1:8081 ，所以 Origin 是 http://127.0.0.1:8081 。

设置 CORS

确定访问不成功的原因之后，可以通过设置存储桶相关的 CORS 来解决以上问题。COS 控制台可以进行 CORS 设置，本示例使用控制台来完成 CORS 的设置。若您的 CORS 设置不是特别复杂，也建议使用控制台来完成 CORS 设置。

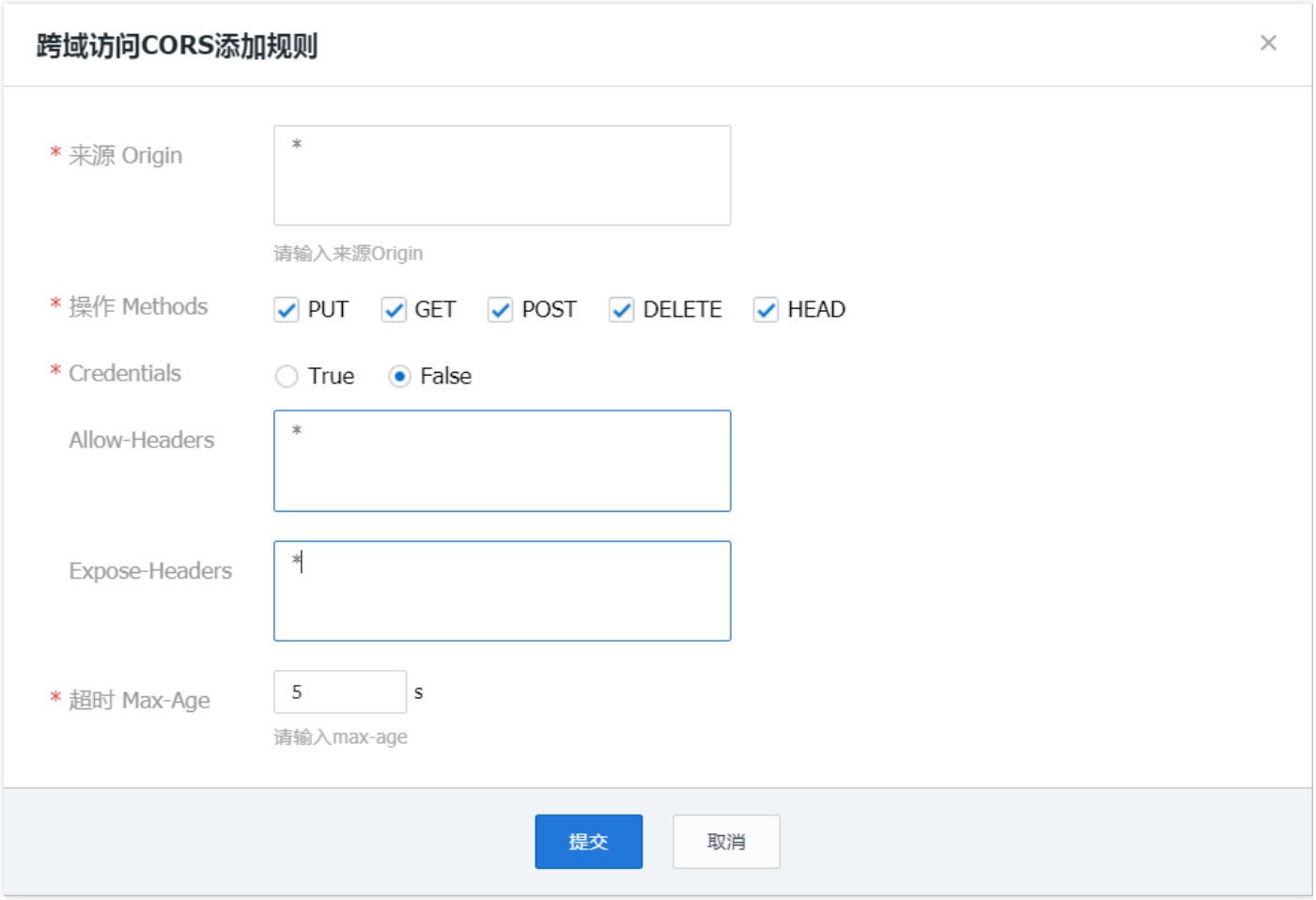
I. 登录 COS 控制台，进入配置页面。

单击 COS 控制台 登录，单击【Bucket 列表】，然后进入相关的存储桶（Bukect，如 corstest），单击【基础配置】，即可找到跨域访问 CORS 设置项。



II. 开启并设置 CORS 规则

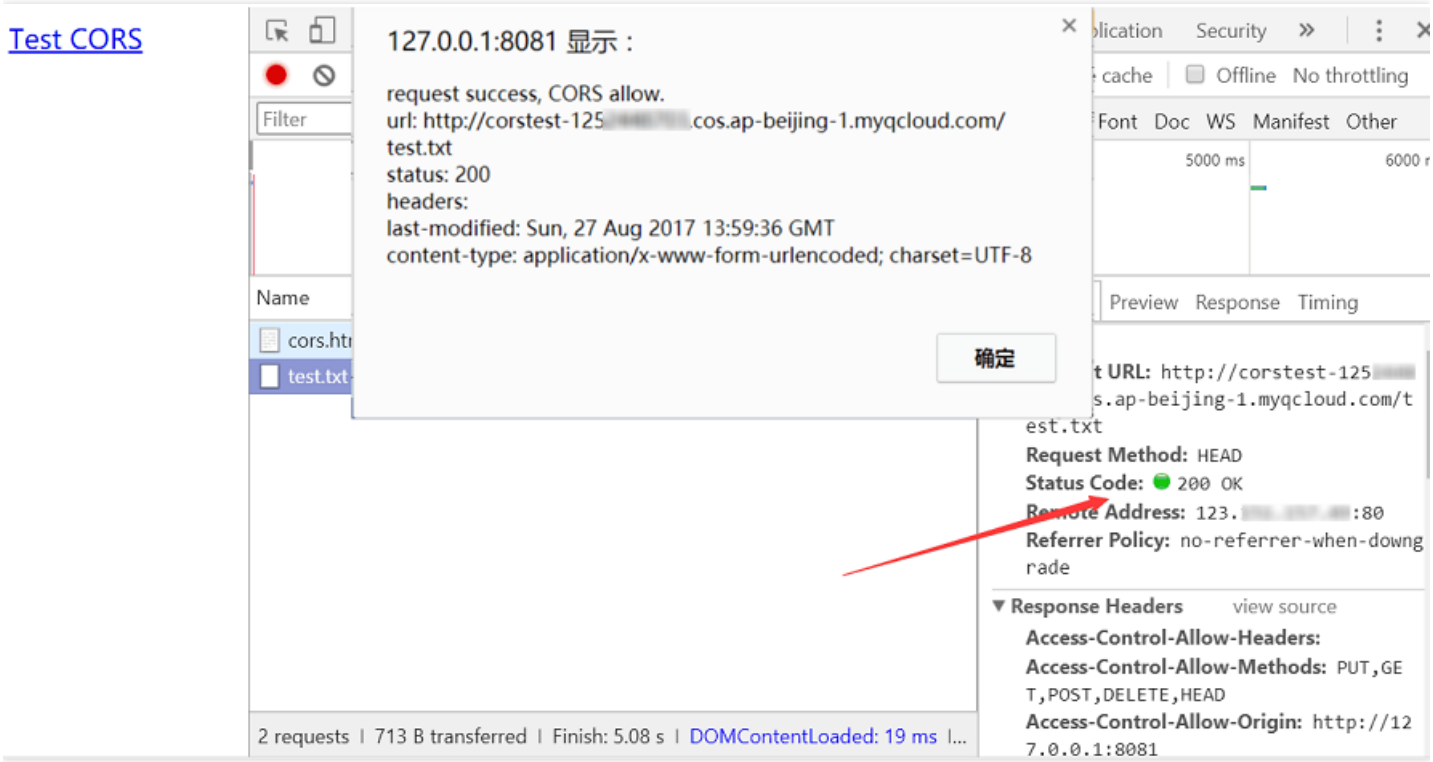
1. 单击【编辑】，开启 CORS 状态，单击【+新增规则】，出现设置页面，添加第一条规则，使用最宽松的配置如下：



注意：
CORS 设置是由一条一条规则组成的，会从第一条开始逐条匹配，以最早匹配上的规则为准。

验证结果

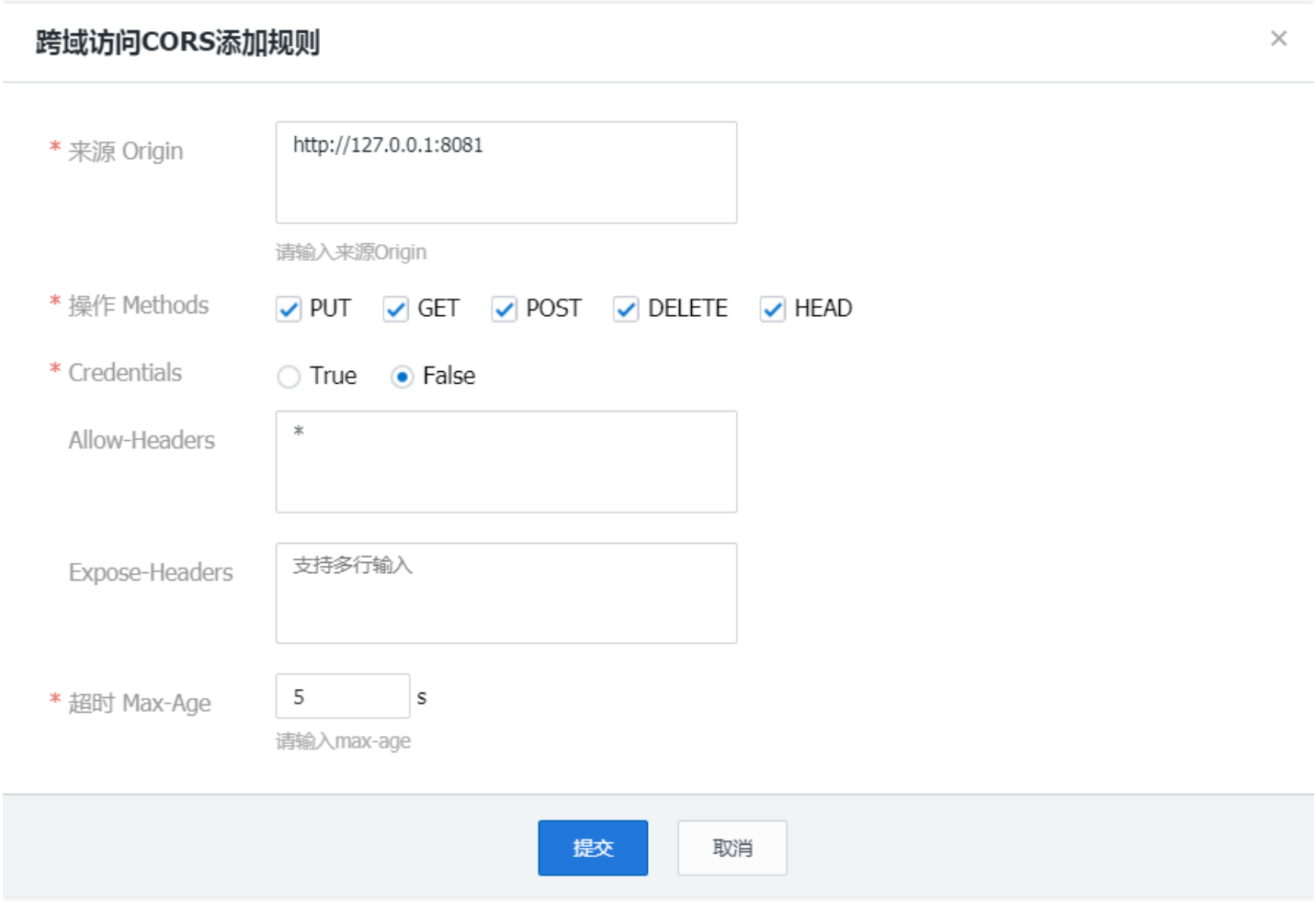
配置完成后，重新尝试访问 test.txt 文本文件。结果如下，可以正常访问请求了。



故障排除及建议

若想要排除跨域带来的访问问题，可以将 CORS 设置为以上最宽松的配置，该配置允许所有的跨域请求。该配置下依然出错，则表明错误出现在其他部分而不是 CORS。

除了最宽松的配置之外，您还可以配置更精细的控制机制来实现针对性的控制。比如对于本示例可以使用如下最小的配置匹配成功：



因此对于大部分场景来说，推荐您根据自己的使用场景来使用最小的配置以保证安全性。

CORS 配置项说明

CORS 配置有以下几项：

来源 Origin

允许跨域请求的来源。

- 可以同时指定多个来源,每行只能填写一个。
- 配置支持 `*` ，表示全部域名都允许，不推荐。
- 支持单个具体域名，形如 `http://www.abc.com` 。
- 支持二级泛域名，形如 `http://*.abc.com` ,但是每行只能有一个 `*` 号。
- 注意不要遗漏协议名 HTTP 或 HTTPS，若端口不是默认的 80，还需要带上端口。

操作 Methods

枚举允许的跨域请求方法（一个或者多个）。
例如：GET、PUT、POST、DELETE、HEAD。

Allow-Header

允许的跨域请求 Header。

- 可以同时指定多个来源,每行只能填写一个。
- Header 容易遗漏，没有特殊需求的情况下，建议设置为 * ，表示允许所有。
- 大小写不敏感。
- 在 Access-Control-Request-Headers 中指定的每个 Header，都必须在 Allowed-Header 中有对应项。

Expose-Header

暴露给浏览器的 Header 列表，即用户从应用程序中访问的响应头（例如一个 Javascript 的 XMLHttpRequest 对象）。

- 具体的配置需要根据应用的需求确定，默认推荐填写 Etag。
- 不允许使用通配符，大小写不敏感，每行只能填写一个。

超时 Max-Age

浏览器对特定资源的预取请求（OPTIONS请求）返回结果的缓存时间，单位为秒。没有特殊情况时可以设置稍大一点，如 60 秒。
该项是可选配置项。

防盗链

最近更新时间：2018-06-20 11:16:23

功能介绍

腾讯云对象存储支持防盗链配置，建议您通过控制台的防盗链设置黑/白名单，来进行安全防护。

盗链案例

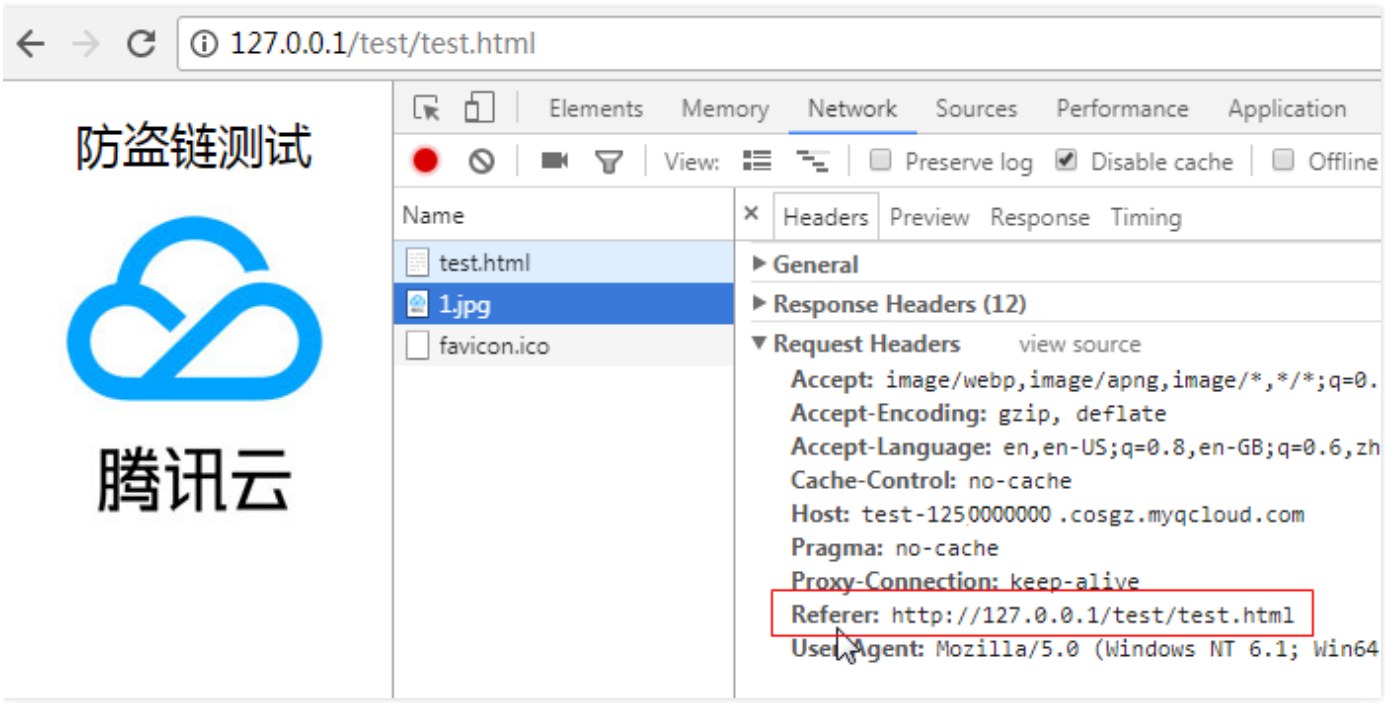
用户 A 在 COS 上传了图片资源 `1.jpg` ，得到可访问链接 `http://test-1250000000.cosgz.myqcloud.com/1.jpg` ，并且在他自己的网页 `http://a.com/a.html` 嵌入了该图片，能正常访问。
用户 B 在 `a.com` 上看到了图片，也在他自己的网页 `http://b.com/b.html` 嵌入了 `1.jpg` 的链接，B 的网页也能正常显示图片。
以上案例中，A 的图片资源 `1.jpg` 就被 B 盗链了。此时 A 在不知情的情况下，COS 上的资源持续被 B 网页正常使用，A 负担了额外的流量费用，造成了费用损失。

防盗链判断原理

防盗链是通过请求 Header 里的 Referer 来判断的：

- Referer 是 Header 的一部分，当浏览器向 Web 服务器发送请求的时候，一般会带上 Referer，告诉服务器该请求是从哪个页面链接过来的，服务器就可以禁止或允许某些来源的网站访问资源。
- 如果直接在浏览器直接打开文件链接 `http://test-1250000000.cosgz.myqcloud.com/1.jpg` ，请求 Header 里不会带有 Referer。

例如，下图是在 http://127.0.0.1/test/test.html 嵌入了 1.jpg ，访问 test.html 时就带有 Referer 指向访问来源：



控制台设置说明

设置步骤

1. 登录 对象存储控制台，进入左侧菜单栏【Bucket 列表】，选择需要设置防盗链的存储桶，进入存储桶。

2. 在存储桶详情页，单击【基础配置】，找到防盗链设置，单击【编辑】按钮进入可编辑状态。

3. 确认当前状态为开启，选择名单类型（黑名单或白名单），设置好相应域名，设置完成单击【保存】即可。用户设置防盗链状态为开启后，必须填入相应的域名。

设置规则说明

- 名单类型黑、白名单二选一：
  - 黑名单：限制名单内的域名访问存储桶的默认访问地址，若名单内的域名访问存储桶的默认访问地址，则返回403。
  - 白名单：限制名单外的域名访问存储桶的默认访问地址，若名单外的域名访问存储桶的默认访问地址，则返回403。
- 配置规则示例：
  - 支持带端口的域名和 IP，如 `test.com:8080` 、 `10.10.10.10:8080` 等地址。
  - 配置 `test.com` ，可命中如 `test.com/123` 、 `test.com.cn` 等以 `test.com` 为前缀的地址。
  - 配置 `test.com` ，可命中如 `https://test.com` 和 `http://test.com` 为前缀的地址。
  - 配置 `test.com` ，可命中它的带端口域名 `test.com:8080` 。
  - 配置 `test.com:8080` ，不会命中域名 `test.com` 。
  - 配置 `*.test.com` ，可限制它的二级、三级域名 `test.com` 、 `b.test.com` 、 `a.b.test.com` 。
- 设置域名支持最多十条域名且为前缀匹配；支持域名、IP 和通配符 `*` 等形式的地址；一个地址占一行，多个地址请换行。

设置示例

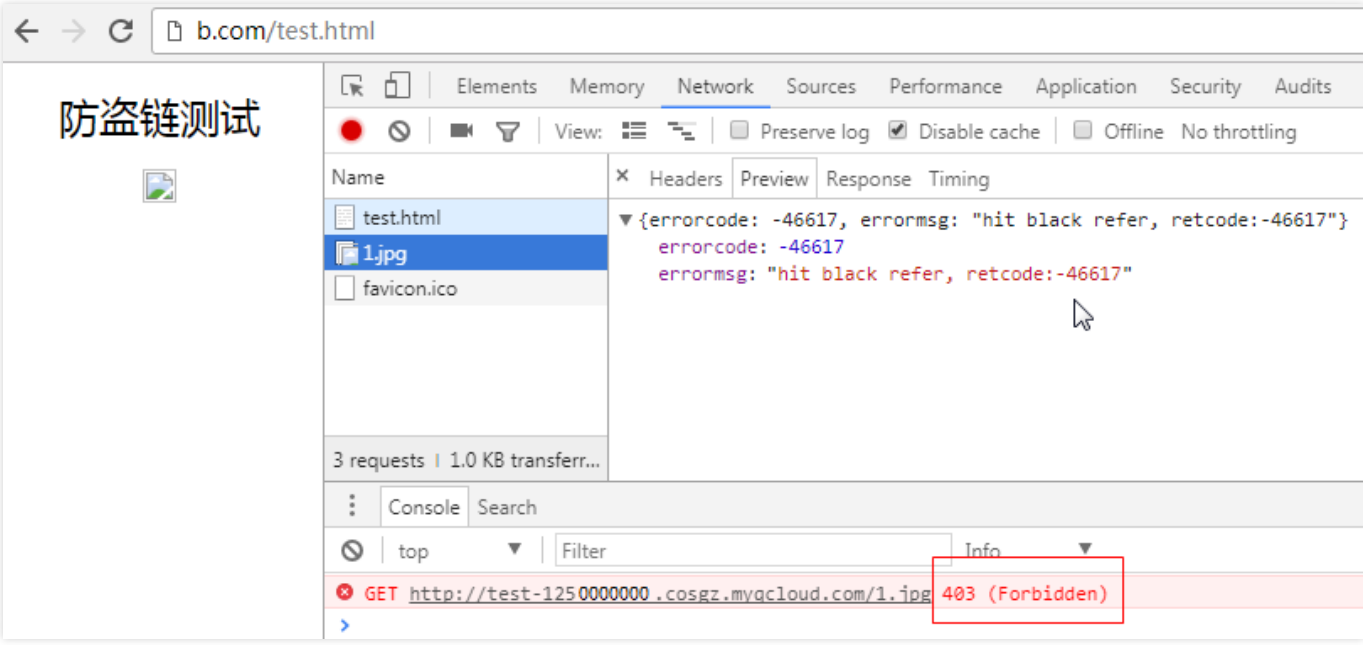
我们使用上文中的 盗链案例 来举例，介绍用户 A 如何通过防盗链设置防止用户 B 盗链图片：

1. 用户 A 给存储桶 test 设置了防盗链规则，有两种方式可以防止 `b.com` 盗链，您可以根据您的实际情况选择：

- 开启方式一：配置黑名单模式，域名设置填入 *.b.com 并保存生效。
- 开启方式二：配置白名单模式，域名设置填入 *.a.com 并保存生效。

2. 开启了防盗链配置之后：

- 访问 http://a.com/a.html 图片显示正常。
- 访问 http://b.com/b.html 图片无法显示，表现如下图。



常见问题

存储桶打开了 CDN 加速，并使用 CDN 域名访问资源，防盗链配置会不生效？

> 由于域名是 CDN 域名，CDN 会有缓存，导致表现不稳定，需要到 CDN 控制台 配置防盗链。

能否设置白名单允许访问，并且浏览器单独打开链接也允许访问？

> 浏览器单独打开链接时，Referer 为空，当前暂不支持额外配置 Referer。

设置了存储桶 test 的防盗链白名单，允许 a.com 访问，但是 a.com 下的网页播放器却不能播放存储桶 test 下的视频文件？

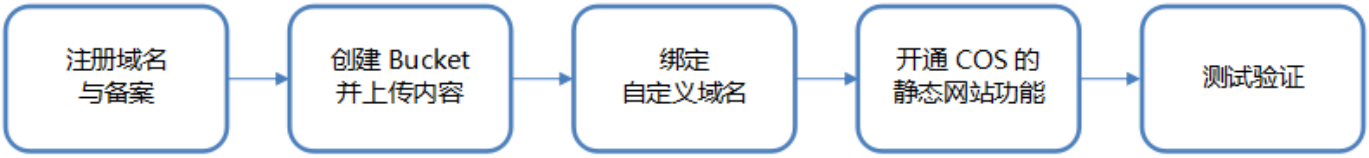
网页中使用 Windows Media Player、Flash Player 等播放器播放视频链接时，在请求里的 Referer 为空，导致没命中白名单，建议换成黑名单的形式配置，让 Referer 为空时也能正常访问。

托管静态网站

最近更新时间：2018-05-24 17:35:54

在此实践中，用户可以在腾讯云对象存储（以下简称 COS）上托管静态网站，访客可以通过自定义域名（例如 www.example.com ）访问托管的静态网站。无论您是想在 COS 上托管已有静态网站还是从零开始建站，此实践可帮助您在 COS 上托管静态网站。以下是具体步骤：

步骤预览



事前准备

以下是实践过程中，将会用到的相关服务：

域名注册：如果还未注册域名，则需要先注册一个域名，例如 www.example.com 。可通过腾讯云 域名注册 申请域名。通常，只需少量费用，即可拥有一个域名。

COS：您将使用 COS 创建存储桶 ，配置权限以允许每个人查看内容，然后上传网页内容。

内容分发网络：内容分发网络（以下简称 CDN）和云解析服务将共同作用，使域名和网站内容绑定，同时为静态网站加速，降低访问延迟，提高可用性。

云解析：利用云解析，可将域名和网站内容绑定在一起，实现使用自定义域名访问静态网站的目的。

> 本指南中的所有步骤都使用 `www.example.com` 作为示例域名。实际操作中请使用您的自有域名替换此域名。

一、注册域名与备案

域名注册是在互联网上建立任何服务的基础。注册域名之后，还需要进行备案，网站才能正常访问。请根据您的具体情况进行操作：

- 已注册域名并备案，可跳过本步骤，进行 步骤二。
- 已注册域名但未备案，请进行 域名备案。
- 未注册域名，请先 注册域名，再进行 域名备案。

二、创建存储桶并上传内容

在完成域名注册及备案后，您需要在 COS 控制台中执行以下任务，以创建和配置网站内容：
2.1 为您的网站内容创建存储桶。
2.2 配置存储桶并上传内容。

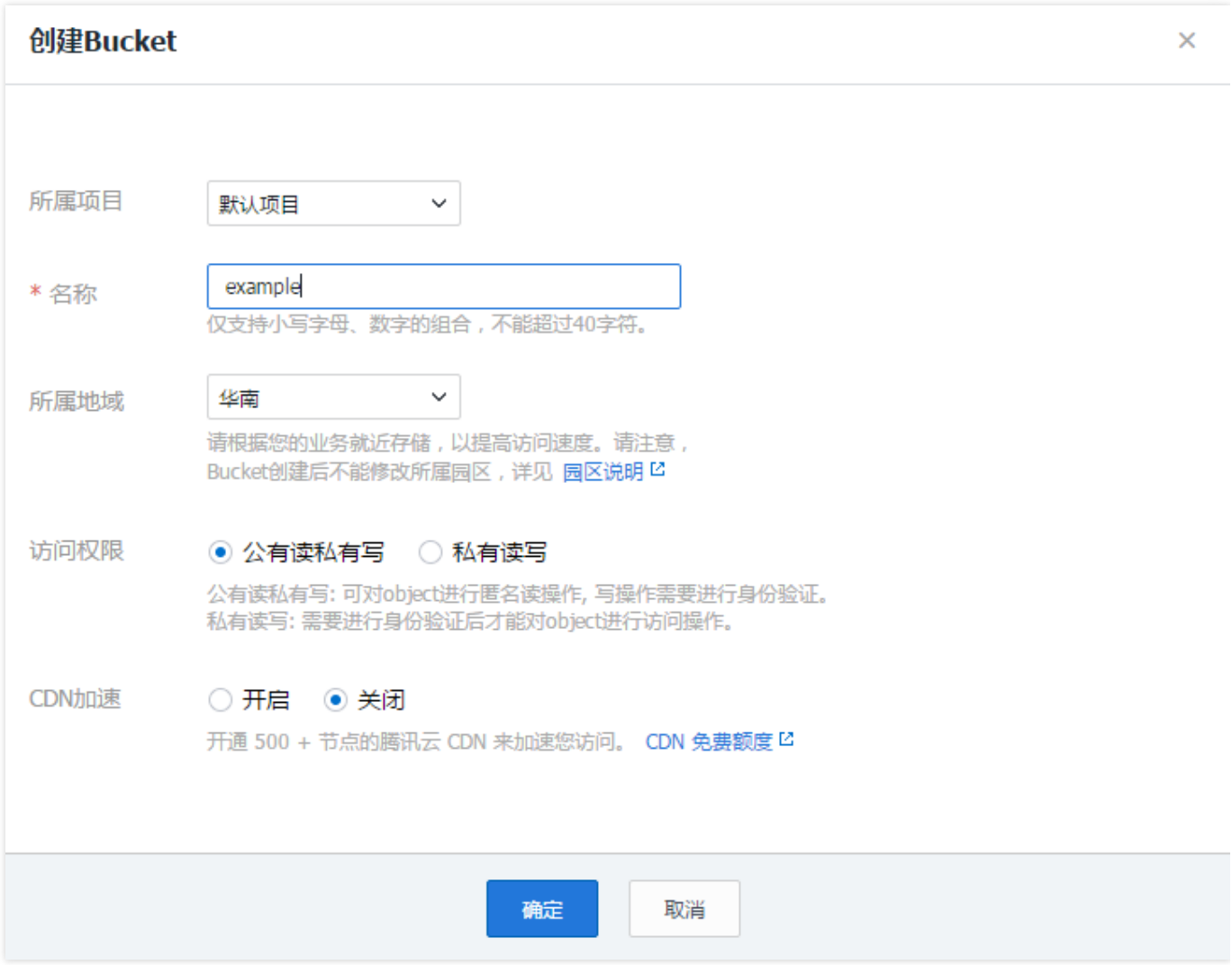
2.1 创建存储桶

请使用腾讯云账号登录 COS 控制台，为您的网站创建相应的存储桶。存储桶在 COS 中用于存储数据，您可以将网站内容存储在一个存储桶中。可通过 COS 概览页或 Bucket 列表快速创建存储桶，详细设置可参考 创建存储桶：

- **通过概览页**

  i. 登录对象存储控制台后，当您首次创建存储桶时，请单击概览页上的【创建 Bucket】，弹出创建 Bucket 对话框。

  ii. 请填写存储桶名称（如 example），选定存储桶所属地域（请参阅 可用地域），单击【确定】即可快速创建一个存储桶。

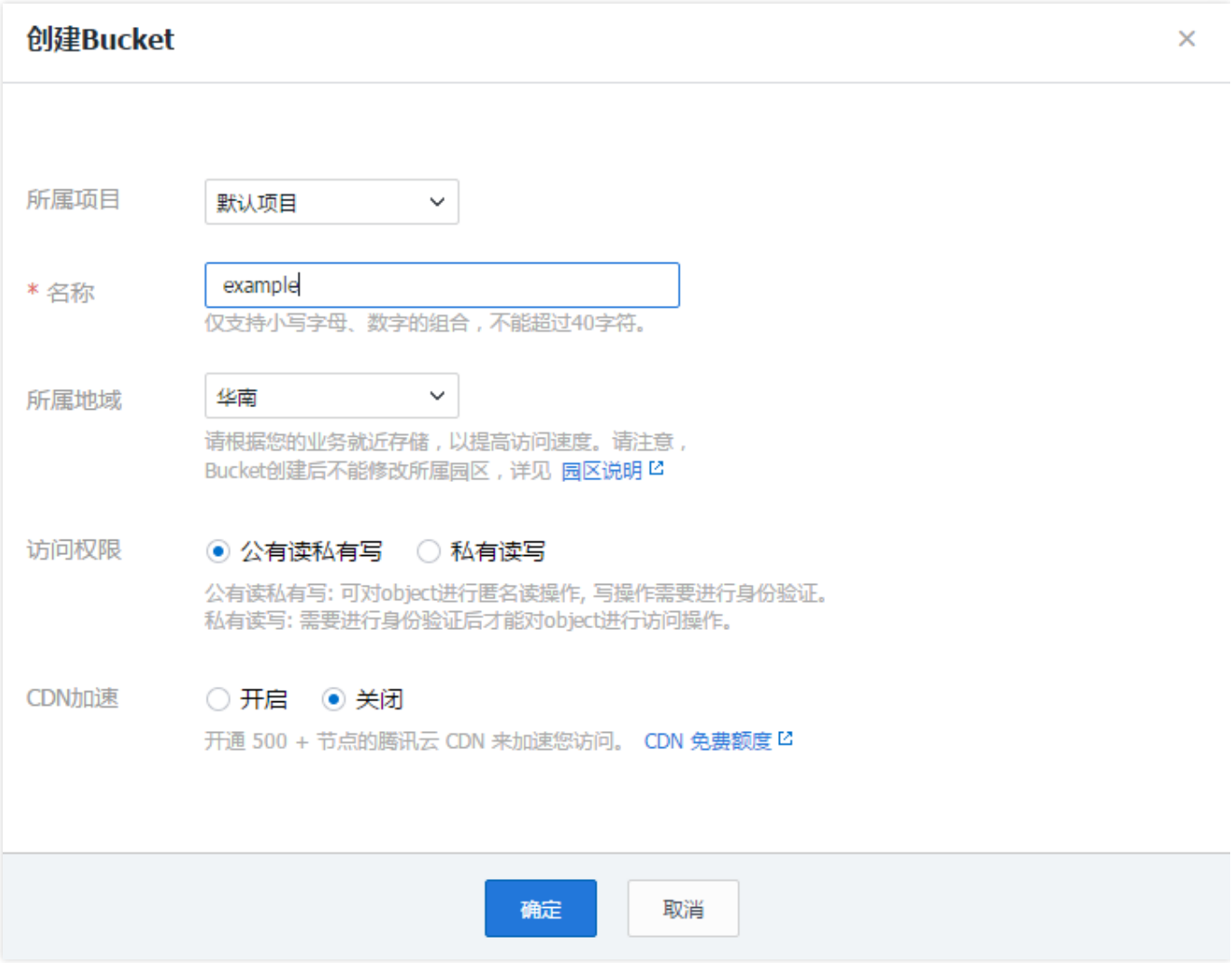


- **通过 Bucket 列表**

  i. 登录对象存储控制台后，请单击左侧导航【 Bucket 列表】，进入 Bucket 列表。

  ii. 单击【+ 创建 Bucket】，弹出创建 Bucket 对话框。

iii. 请填写存储桶名称（如 example），选定存储桶所属地域（请参阅 可用地域），单击【确定】即可快速创建一个存储桶。



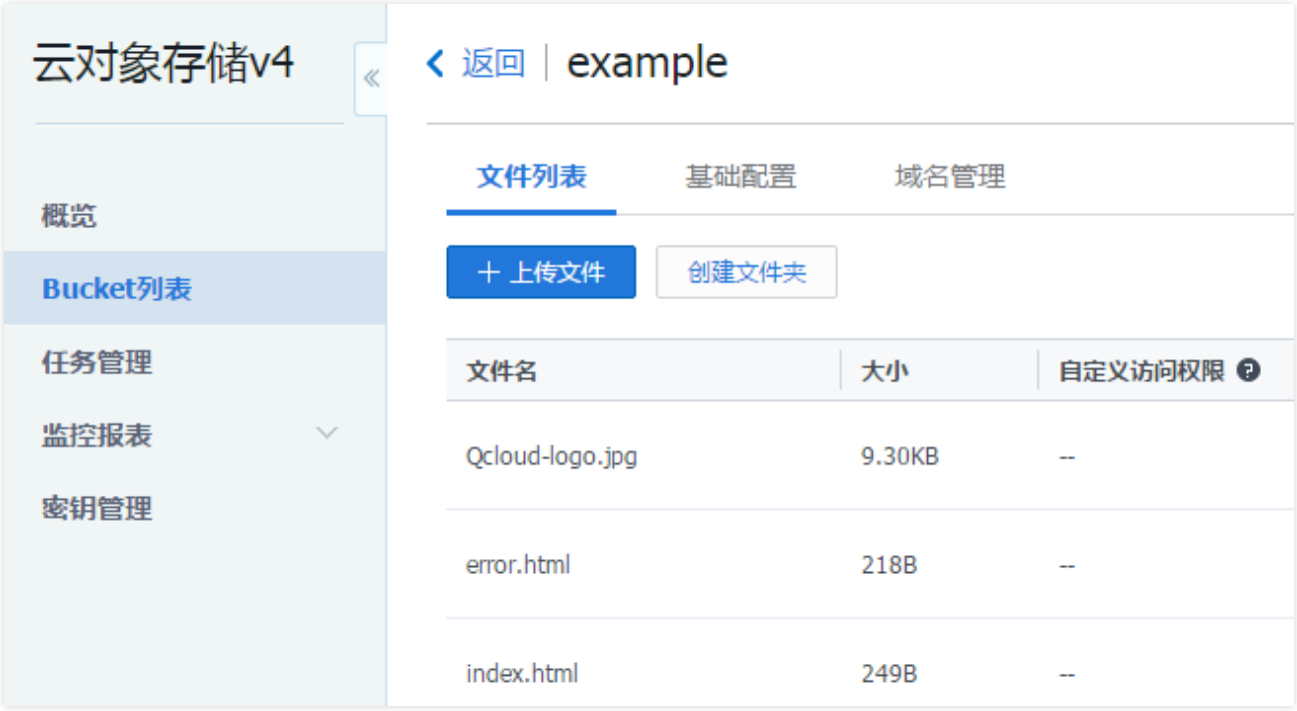
2.2 配置存储桶并上传内容

1. 将存储桶的访问权限设置为**公有读私有写**，使网站内容可被公开访问。
   i. 在 COS 控制台，单击已创建好的存储桶。
   ii. 进入存储桶后，单击【基础配置】>基本信息的【编辑】按钮。

iii. 修改存储桶的访问权限为公有读私有写，保存即可。

2. 将您的网站内容上传到已创建好的存储桶。

单击【文件列表】，在文件列表页面上传网站内容。详细说明请参考 上传对象。



私有读写：只有该存储桶的创建者及有授权的账号才对该存储桶中的对象有读写权限，其他任何人对该存储桶中的对象都没有读写权限。推荐使用。
共有读私有写：任何人（包括匿名访问者）都对该存储桶中的对象有读权限，但只有存储桶创建者及有授权的账号才对该存储桶中的对象有写权限。
共有读写：任何人（包括匿名访问者）都对该存储桶中的对象有读权限和写权限，不推荐使用。

在存储桶中托管的内容可以是文本文件、照片、视频——任何您想要托管的内容。如果还未构建网站，则只需为此实践创建一个文件。
例如，您可使用以下 HTML 创建文件，并将其上传到存储桶。网站主页的文件名通常为 index.html。在后续步骤中，您将提供此文件作为网站的索引文档。

```
<!DOCTYPE html>
<html>
<head>
<title>Hello COS!</title>
<meta charset="utf-8">
</head>
<body>
<p>欢迎使用&nbspCOS&nbsp的静态网站功能。 </p>
<p>这是首页！ </p>
</body>
</html>
```

开启静态网站功能后，当用户访问任何不带文件指向的一级目录时，COS 默认优先匹配对应存储桶目录下 index.html，其次为 index.htm，若无此文件，则返回 404。

三、绑定自定义域名

用户只有绑定自定义域名并开启静态网站功能后，才可以直接在浏览器中打开资源。使用默认提供的域名（CDN 加速域名和 COS 默认域名）访问资源时将始终弹出下载框。

可设置自定义域名直接指向存储桶，并开通静态网站功能，达到通过浏览器直接访问网站的目的（存储桶中的内容）。同时为降低网站访问延迟，提高可用性。在此实践中，在绑定自定义域名的同时，为自定义域名开启 CDN 加速（不开启 CDN 加速的配置请参考 域名管理-配置自定义域名），使网站访客获取更好的浏览体验。

3.1 域名添加

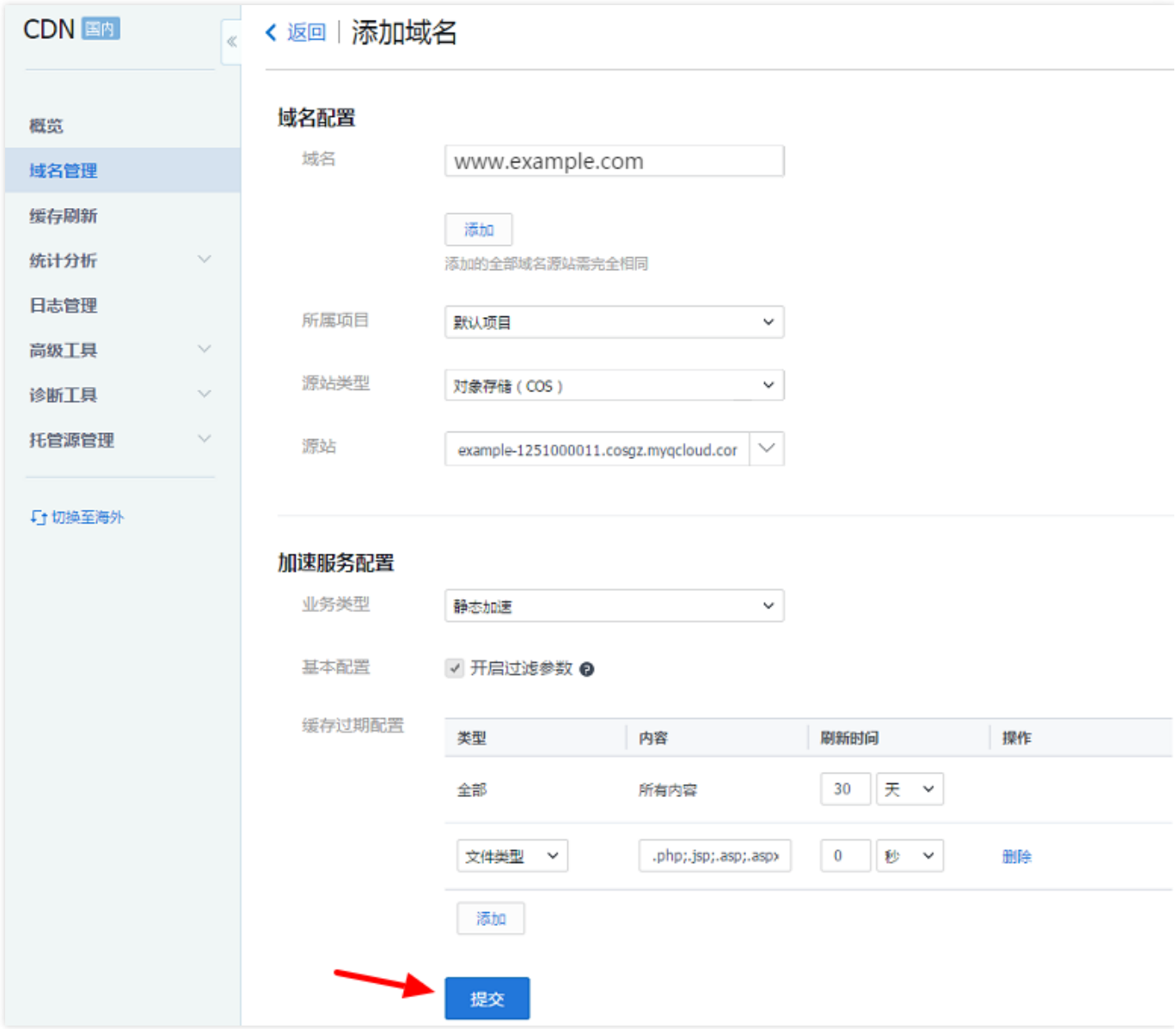
在进行自定义域名添加时，有两种途径供您选择：

- 通过 CDN 控制台添加
- 通过 COS 控制台添加

如果您想在添加自定义域名的同时，进行 CDN 高级管理和配置，可优先选择 CDN 控制台添加域名。本实践中不涉及 CDN 高级管理和配置，如需帮助，请参考 CDN 配置管理。
如果您只需要先添加自定义域名，不进行其他配置，COS 控制台添加将节省时间。

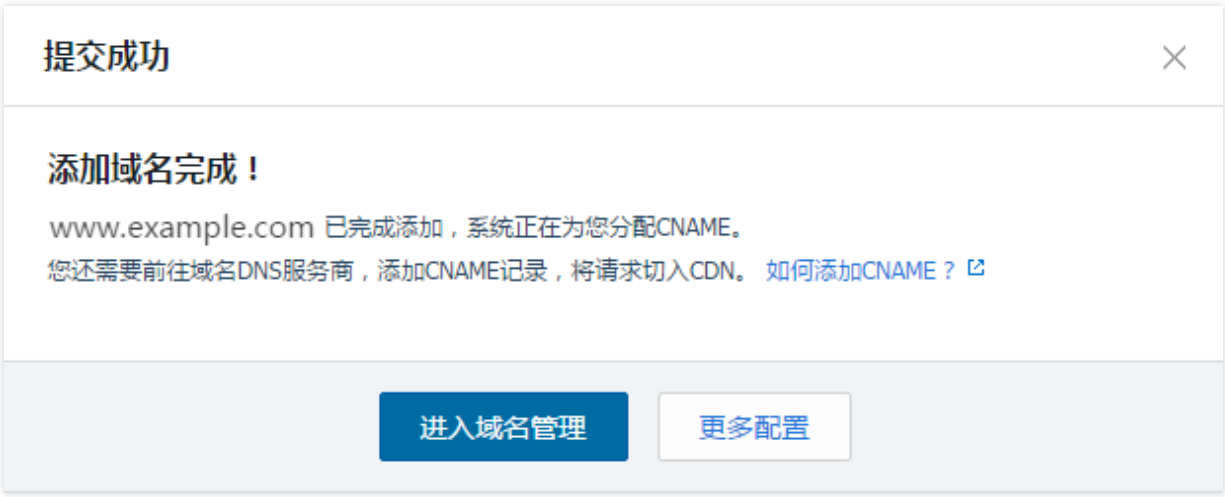
- **通过 CDN 控制台添加**
  i. 请登录 CDN 控制台，从左侧导航进入域名管理页面。
  ii. 单击【+添加域名】，进入域名配置界面。

iii. 请输入自有域名，源站类型选择**对象存储（COS）**，并为源站选择托管网站内容的对应存储桶默认域名。业务类型选择静态加速，其他保持默认配置，提交即可。


CDN 国内
概览
域名管理
缓存刷新
统计分析
日志管理
高级工具
诊断工具
托管源管理
切换至海外
返回 | 添加域名
域名配置
域名
www.example.com
添加
添加的全部域名源站需完全相同
所属项目
默认项目
源站类型
对象存储（COS）
源站
example-1251000011.cosgz.myqcloud.cor
加速服务配置
业务类型
静态加速
基本配置
开启过滤参数
缓存过期配置
类型
内容
刷新时间
操作
全部
所有内容
30
天
文件类型
.php;.jsp;.asp;.asp›
0
秒
删除
添加
提交

iv. 域名添加完成。

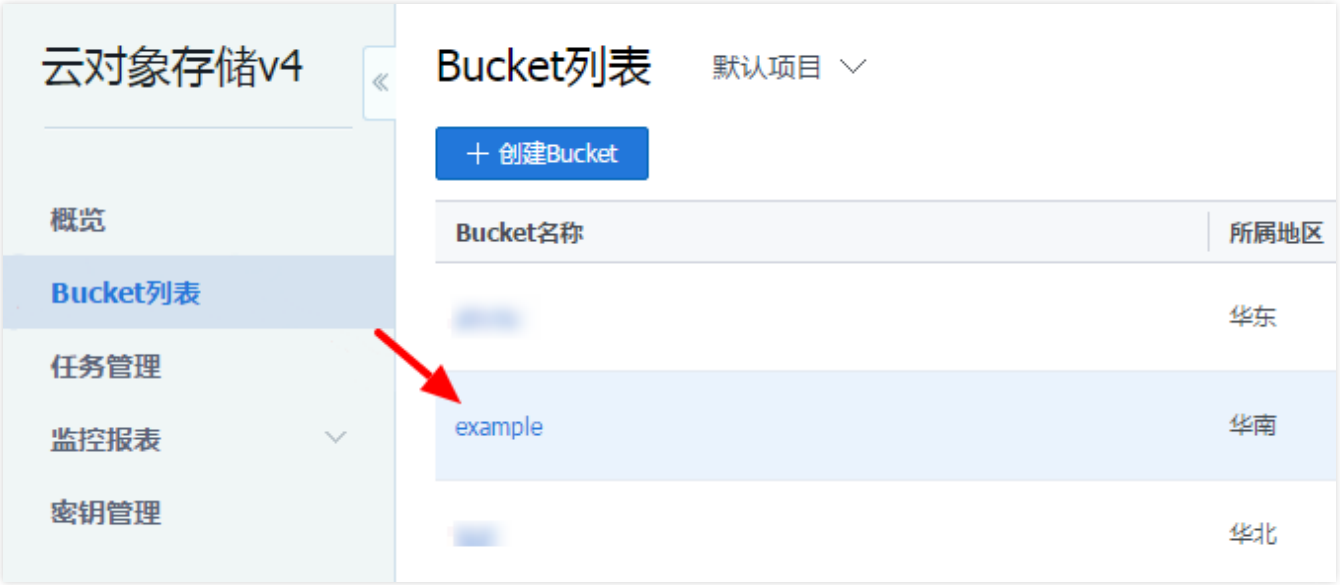
a. 请关闭弹窗，耐心等待域名配置下发至全网节点（约 15 分钟）。



b. 获取系统分配的 CNAME 记录，再进行步骤 3.2。

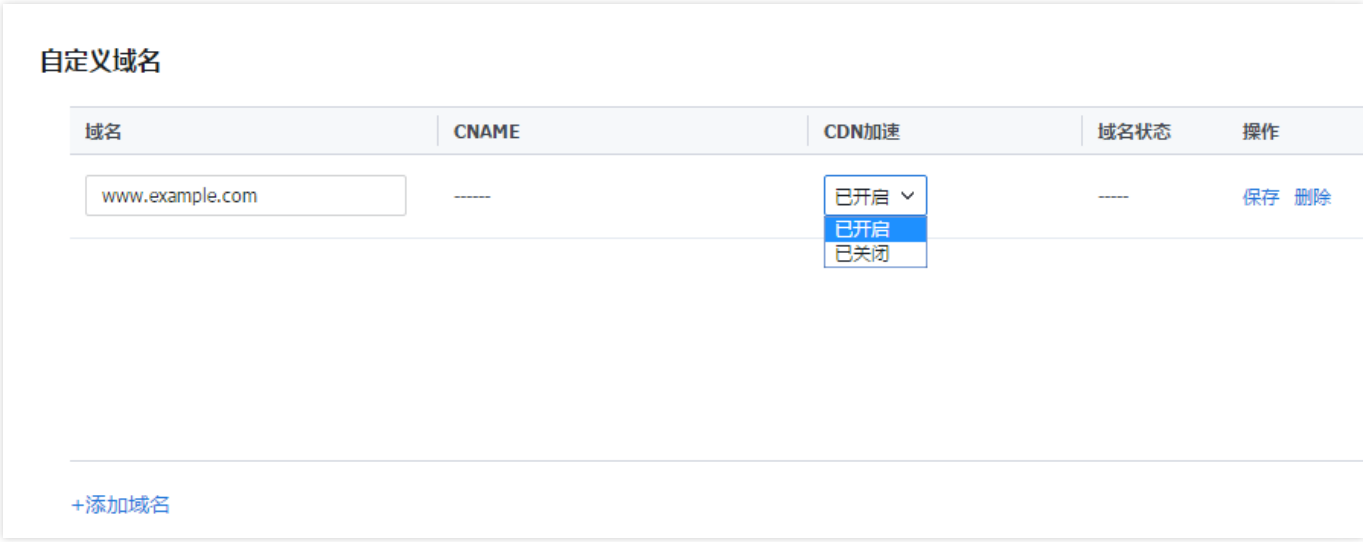
- **通过 COS 控制台添加**

  i. 登录 COS 控制台 ，进入左侧菜单栏【 Bucket 列表】，单击存储网站内容的存储桶（如 example），进入存储桶。



  ii. 单击【域名管理】，进入域名管理页面，单击自定义域名下的【+ 添加域名】按钮，进入可配置状态。

iii. 输入待绑定的自定义域名（如 www.example.com ），选择开启 CDN 加速，单击【保存】即可完成添加。

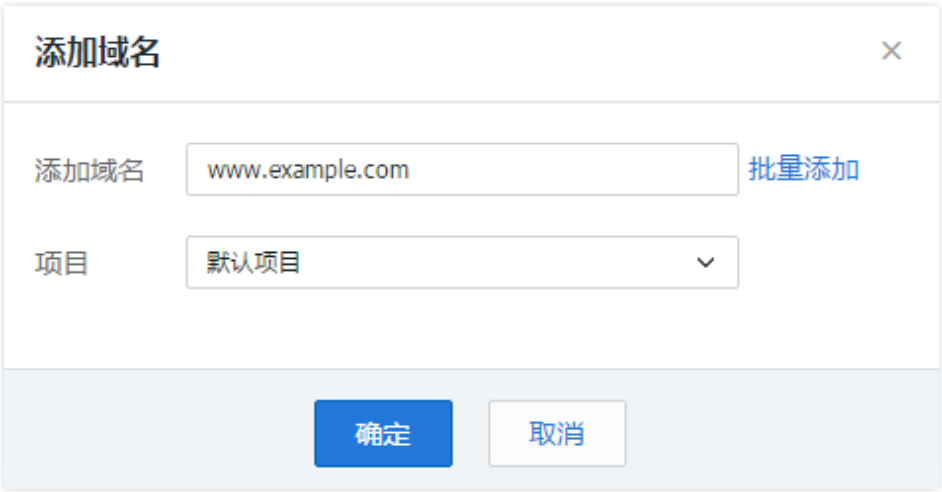


iv. 请稍等几分钟，等待域名上线。获取对应的 CNAME 记录，再进行步骤 3.2。

3.2 域名解析

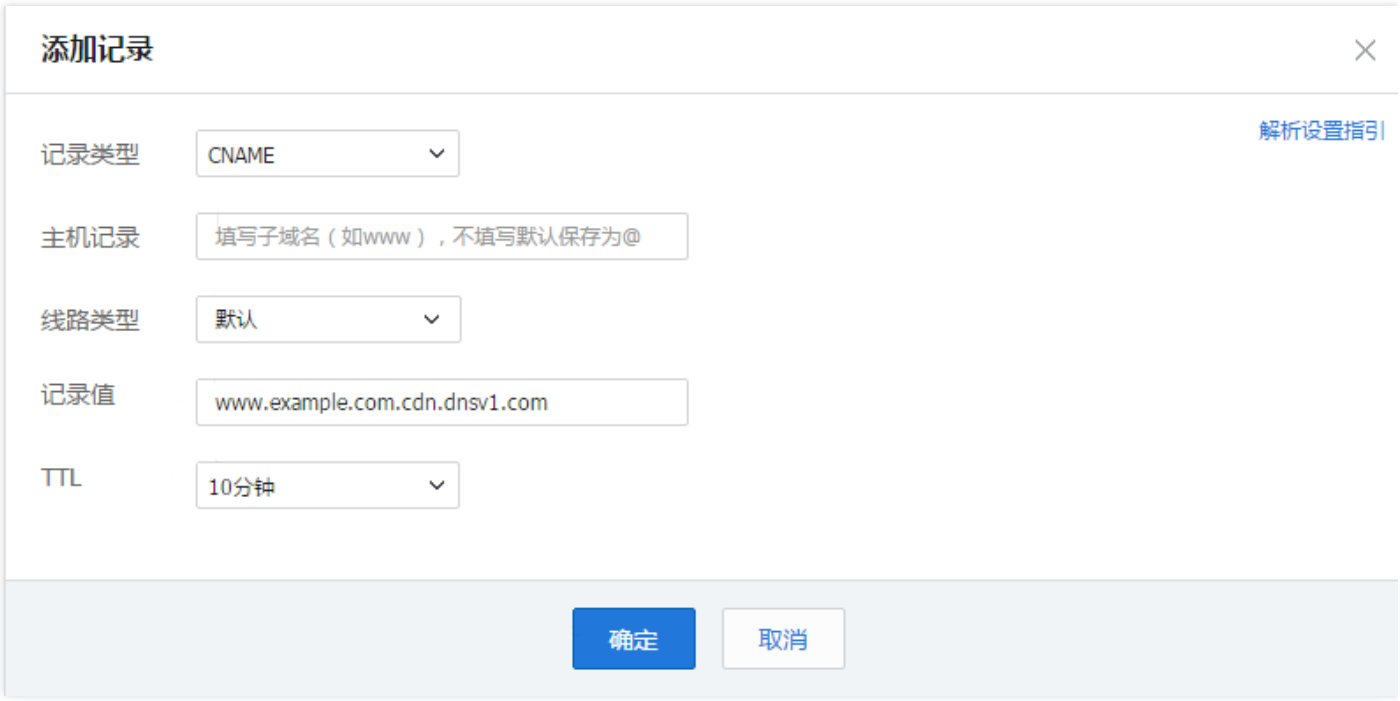
1. 添加自定义域名后，还需进行域名解析。请登录 域名管理控制台，单击左侧菜单栏【云解析】>【一级域名】，进入一级域名菜单。单击【+添加域名】，弹出添加域名对话框。

2. 输入自定义域名，选择所属项目，单击【确定】保存即可。



3. 域名添加成功后，单击域名，进入解析记录管理页面。单击【+ 添加记录】，弹出添加记录对话框。

4. 记录类型选择 CNAME，主机记录留空，线路类型选择默认，填入 步骤 3 获取的 CNAME 记录，TTL 保持默认，单击【确定】保存即可。完成解析添加后，大约需 15 分钟左右生效，请耐心等待。



四、设置静态网站托管

将网站内容与自定义域名绑定之后，需要开启 COS 的静态网站功能，才能通过浏览器直接访问网站内容。具体步骤如下：

1. 登录 COS 控制台 ，进入左侧菜单栏【 Bucket 列表】，单击存储网站内容的存储桶（如 example），进入存储桶。

2. 单击【基础配置】，找到静态网站设置，单击【编辑】按钮，进入可编辑状态。

3. 修改当前状态为开启，开启 Index 索引，开启指定的 Http 状态码并设置指向文件（可选），设置完成单击【保存】即可。

静态网站设置的具体配置及相关参数说明，请参考 静态网站设置。

五、测试验证

在完成上述步骤后，可通过在浏览器地址栏输入网站域名进行访问，来验证实践结果，以 `www.example.com` 为例：

- `http://www.example.com` ——返回名为 example 的存储桶中的索引页面（index.html）。
- `http://www.example.com/test.html`（不存在的文件）——返回名为 example 的存储桶中的 404 文档（error.html）。

在某些情况下，您可能需要清除缓存才能看到预期结果。

快速搭建移动应用传输服务

最近更新时间：2018-05-21 17:05:01

背景

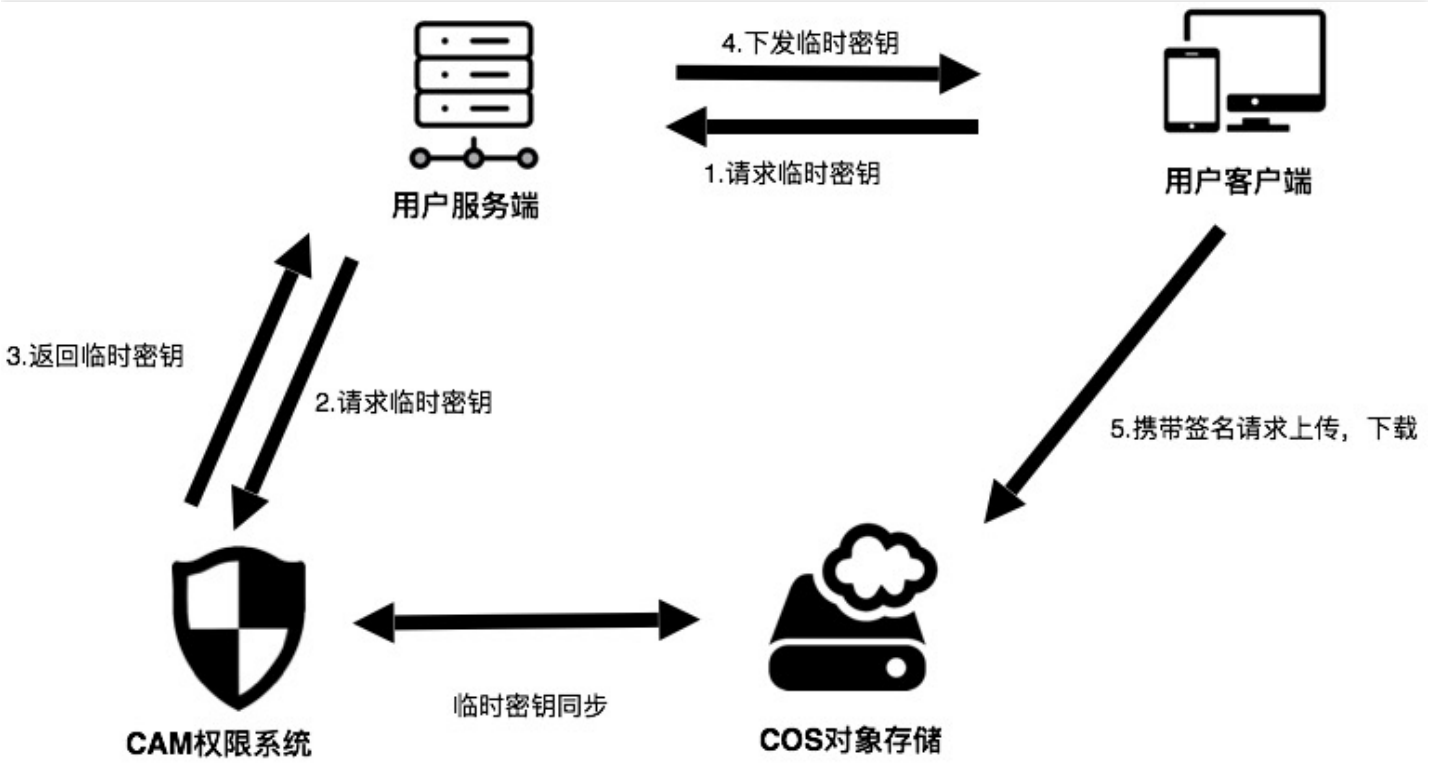
移动互联网时代，App 作为移动互联网服务的基础设施，往往需要上传和下载大量的数据，数据的安全性和可靠性尤为重要。现在开发者可以将数据存储相关的问题交给 腾讯云 COS 服务，而只需要关心自己应用的业务逻辑即可，可减少很多工作量，提升开发效率。

本文主要介绍如何快速搭建一个基于 COS 的应用传输服务，在腾讯云 COS 上实现应用数据的上传下载，在您的服务器上只需要部署您自己的业务、生成和管理临时密钥。

其中服务器端获取临时密钥的SDK可以从qcloud-cos-sts-sdk下载，目前支持JAVA、Node、Python版本

架构

整体架构图如下所示：



其中：

- 应用 APP：即用户手机上的 App。
- COS：腾讯云对象存储，负责存储 App 上传的数据。
- CAM：腾讯云访问管理，用于生成 COS 的临时密钥。
- 应用服务器：用户自己的后台服务器，这里用于向 CAM 请求临时密钥，并返回给应用 App。

步骤

1. 应用向用户的应用服务器发送一个获取临时密钥的请求。
   用户 App 在向 COS 发送请求时，必须通过密钥对请求进行签名（具体签名算法）。直接将用户的永久密钥放在终端上会存在很大的泄密风险，因此必须将您的永久密钥放在您的应用服务器上，然后由应用服务器向腾讯云 CAM 申请临时密钥，并最终返回给终端对请求进行签名。

2. 应用服务器收到请求后，向 CAM 申请临时密钥。
   我们认为用户的应用服务器是可信任的，可以直接配置用户的永久密钥用于向 CAM 申请临时密钥。同时，在服务端您可以配置临时密钥的有效期以及策略（Policy）。

3. CAM 返回临时密钥给应用服务器。

4. 应用服务器返回临时密钥给应用 App。
   我们对数据的返回格式没有做具体的要求，这里建议直接返回 CAM 的 JSON 数据即可。

5. 应用 App 向 COS 发送请求（如上传文件等）。
   应用 App 获取到临时密钥后，可以通过我们的 SDK 很容易地对请求进行签名，来使用我们的 COS 服务。

示例

搭建一个应用服务器

本示例采用 Python 编写，您可以 单击下载。

1. **配置参数**
   下载并解压好的文件夹根目录下有 config.py 目录，可以来配置以下参数：

```
class Config(object):

def __init__(self):
pass

# 用户昵称，非必选
NAME = "WANG"
# 策略
POLICY = COMMON_POLICY
# 临时证书有效期
DURATION_SECOND = 1800
# 用户的secret id
SECRET_ID = "您的 secret id"
# 用户的secret key
SECRET_KEY = "您的 secret key"
```

   示例中提供的是一个拥有账户下所有资源读写权限的策略，如果您想配置更细化的策略，请参考 CAM COS 相关案例。

2. **解析返回的 JSON 数据**

```
{
"code":0,"message":"","codeDesc":"Success",
"data":
{
"credentials":
```

```
{
"sessionToken":"42f8151428b3960b1226f421b8f271c6242ad02c3",
"tmpSecretId":"AKIDtd9QSGWBIDuMaYFp57tSmrhJgohLtvpT",
"tmpSecretKey":"ZfV5PVLvFLCvPefPt76qKYXIo56tSmrg"
},
"expiredTime":1508400619
}
}
```

应用服务器搭建临时密钥服务

下面以腾讯云 CVM Ubuntu 实例为例，详细介绍如何搭建一个简单的临时密钥服务。

1. **登录腾讯云 CVM**
   购买 CVM 实例后，可直接用浏览器在控制台登录，或者使用终端直接 SSH 登录。

2. **Ubuntu 环境下配置 Python 开发环境**

```
sudo apt-get update
sudo apt-get install python-dev python-pip python-virtualenv
```

3. **下载源码并解压**

```
wget http://rickenwang-1253653367.cosgz.myqcloud.com/resources/signer-release.zip
unzip signer-release.zip
```

> 下载后直接解压，解压后需要修改根目录下的 `secret_key` 和 `secret_id` 配置信息，具体值可以在 云 API 密钥控制台 上查询。

4. **激活 virtualenv 环境并进行配置**

   i. 激活 virtualenv 环境：

```
cd 到flask项目目录
virtualenv venv
source venv/bin/activate
```

   ii. 配置环境：

```
pip install gunicorn
pip install flask
pip install flask-script
```

5. **使用 supervisor 管理服务**

i. 安装 supervisor 并打开配置文件

```
pip install supervisor
echo_supervisord_conf > supervisor.conf
vi supervisor.conf
```

ii. 在配置文件 supervisor.conf 的底部添加以下内容：

```
[program:manage]
command=./venv/bin/gunicorn -w4 -b0.0.0.0:5000 manage:app
directory=./
startsecs=0
stopwaitsecs=0
autostart=false
autorestart=false
stdout_logfile=./log/gunicorn.log
stderr_logfile=./log/gunicorn.err
```

6. **启动服务并测试**

使用如下命令启动服务：

```
supervisord -c supervisor.conf
supervisorctl -c supervisor.conf start manage
```

应用服务器启动后，直接用浏览器访问 http://ip:5000/sign 地址，若返回临时密钥 JSON 字符串，则说明应用服务器临时密钥服务正常运行。

> **注意：**
> 浏览器访问时 IP 请修改为您服务器的 IP 或者域名，默认的端口是 5000，您可自行在 supervisor.conf 文件中修改。

应用 App 接入应用服务器临时密钥服务

若您对腾讯云 COS 服务已有一些基本的了解，那么您可以很容易地使用我们 COS XML 终端 SDK，并将您的应用服务器的临时密钥服务接入进来，下面以 Android SDK 为例说明，总共分为三步：

> **注意：**
> 使用我们的 SDK 前，您需要了解一些基本的名词术语，如 APPID、Bucket、SecretId、SecretKey 等，术语说明参见 API 简介-术语信息。

1. 定义 `MySessionCredentialProvider` 类继承抽象类 `BasicLifecycleCredentialProvider` ，并实现其 fetchNewCredentials() 方法。

```
public class MySessionCredentialProvider extends BasicLifecycleCredentialProvider {

@Override
protected QCloudLifecycleCredentials fetchNewCredentials() throws QCloudClientException {

SessionQCloudCredentials credentials;
String tempSecretId = "";
String tempSecretKey = "";
String sessionToken = "";
long expireTime;

// 1、访问应用服务器获取临时密钥 API，获取临时密钥 JSON 字符串
// ...


// 2、解析 JSON 字符串，初始化 tempSecretId, tempSecretKey, sessionToken, expiredTime 四个参数
// ...


// 3、返回 SessionQCloudCredentials
return new SessionQCloudCredential(tempSecretId, tempSecreKey, sessionToken, expireTime);

}

}
```

2. 用 `MySessionCredentialProvider` 初始化 `CosXmlService` 。

```
// 您的账户信息
String appid = "对象存储 的服务APPID";
String region = "存储桶 所在的地域";
```

```
// 创建 CosXmlServiceConfig 对象
CosXmlServiceConfig serviceConfig = new CosXmlServiceConfig.Builder()
.setAppidAndRegion(appid, region)
.build();

// 初始化 CosXmlService
CosXmlService cosXmlService = new CosXmlService(getApplicationContext(), cosXmlServiceConfig,
new MySessionCredentialProvider());
```

3. 通过 CosXmlService 对象的各种接口访问 COS 服务，下面以简单上传为例：

```
String bucket = "存储桶名称";
String cosPath = "远端路径，即存储到 COS 上的绝对路径"; //格式如 cosPath = "/test.txt";
String srcPath = "本地文件的绝对路径"; // 如 srcPath = Environment.getExternalStorageDirectory().getPa
th() + "/test.txt";
long signDuration = 600; //签名的有效期，单位为秒

PutObjectRequest putObjectRequest = new PutObjectRequest(bucket, cosPath, srcPath);

putObjectRequest.setSign(signDuration,null,null);

/*
* 设置进度显示
* 实现 QCloudProgressListener.onProgress(long progress, long max) 方法，
* progress 已上传的大小， max 表示文件的总大小
*/
putObjectRequest.setProgressListener(new QCloudProgressListener() {

@Override
public void onProgress(long progress, long max) {
float result = (float) (progress * 100.0/max);
Log.w("TEST","progress =" + (long)result + "%");
}
});

//使用同步方法上传
try {
PutObjectResult putObjectResult = cosXmlService.putObject(putObjectRequest);
Log.w("TEST","success: " + putObjectResult.accessUrl);

} catch (CosXmlClientException e) {
//抛出异常
Log.w("TEST","CosXmlClientException =" + e.toString());
} catch (CosXmlServiceException e) {
```

```
//抛出异常
Log.w("TEST","CosXmlServiceException =" + e.toString());
}
```

相关文档

更详细的移动端 XML SDK 使用说明请参见：

- COS XML Android SDK
- COS XML iOS SDK

使用 COS 作为 Druid 的 Deep storage

最近更新时间：2018-08-07 10:00:20

环境依赖

HADOOP-COS 与 Hadoop-COS-Java-SDK（包含在 HADOOP-COS 的 dep 目录下）

Druid 版本：Druid-0.12.1

下载与安装

获取 hadoop-cos

在官方 Github 上下载 HADOOP-COS。

安装 hadoop-cos

Druid 使用 COS 作为 Deep Storage 需要借助 Druid-hdfs-extension 实现：
下载 HADOOP-COS 后，将 dep 目录下的 hadoop-cos-2.7.3.jar 以及 cos_hadoop_api-5.2.5.jar 拷贝到 druid 安装路径的 extensions/druid-hdfs-storage 以及 hadoop-dependencies/hadoop-client/2.7.3。

使用方法

配置修改

首先，修改 druid 安装路径的 conf/druid/_common/common.runtime.properties 文件，将 hdfs 的extension 加入到 druid.extensions.loadList 中，同时指定 hdfs 为 druid 的 deep storage，而路径则填写为 cosn 的路径：

```
properties
druid.extensions.loadList=["druid-hdfs-storage"]
druid.storage.type=hdfs
druid.storage.storageDirectory=cosn://bucket-appid/<druid-path>
```

然后，在 conf/druid/_common/ 这个目录下新建一个 hdfs 的配置文件 hdfs-site.xml，填入 COS 的密钥信息等：

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
```

```
You may obtain a copy of the License at
http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->
<!-- Put site-specific property overrides in this file. -->
<configuration>
<property>
<name>fs.cosn.userinfo.secretId</name>
<value>xxxxxxxxxxxxxxxxxxxxxxxx</value>
</property>
<property>
<name>fs.cosn.userinfo.secretKey</name>
<value>xxxxxxxxxxxxxxx</value>
</property>
<property>
<name>fs.cosn.impl</name>
<value>org.apache.hadoop.fs.CosFileSystem</value>
</property>
<property>
<name>fs.cosn.userinfo.region</name>
<value>ap-xxxx</value>
</property>
<property>
<name>fs.cosn.buffer.dir</name>
<value>/tmp</value>
</property>
</configuration>
```

上述配置的支持项与 HADOOP-COS 官网文档描述完全一致，请查阅 HADOOP-COS 官方文档。

开始使用

最后依次启动 druid 的进程后，Druid 的数据就可以加载到 COS 中。

高效安全地迁移数据至对象存储 COS

最近更新时间：2018-08-31 17:14:48

概述

对象存储 COS 支持多种数据迁移方式，用户可以按照不同的迁移场景选择不同的数据迁移方式。本文档总结归纳了几种典型的迁移场景所适用的迁移方式，以及对应的迁移操作步骤和操作技巧，帮助用户高效、安全地将数据迁移到对象存储 COS。

迁移类型	迁移方式
线上迁移	COS Migration ，迁移服务平台 MSP，COS 镜像回源
线下迁移	云数据迁移 CDM

迁移场景

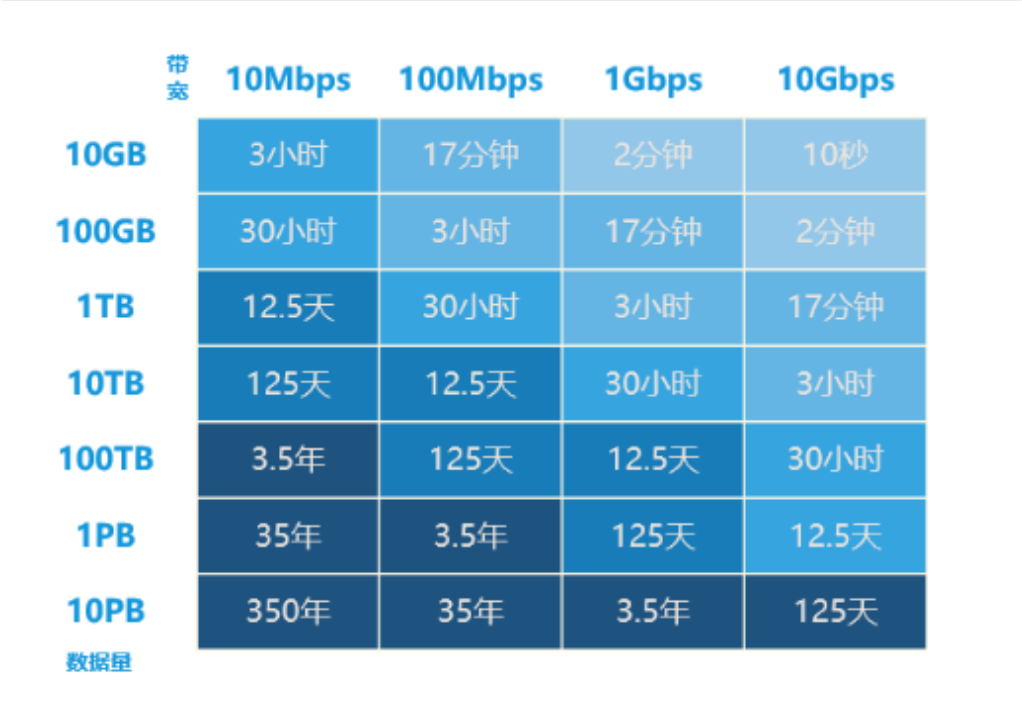
1. 本地数据迁移至对象存储 COS

对于拥有本地 IDC 的用户，对象存储 COS 在不同迁移类型上支持以下几种迁移方式，帮助用户将本地 IDC 的海量数据快速迁移至对象存储 COS。

- 线上迁移：COS Migration
- 线下迁移：云数据迁移 CDM

用户可依据数据迁移量、IDC 出口带宽、IDC 空闲机位资源、可接受的迁移完成时间等因素来考虑如何选择迁移方式。下图展示的是使用线上迁移时预估的时间消耗，可以看出，若此次迁移周期超过 10 天的时间或者迁移数据量超

过 50TB，我们建议您选择 云数据迁移 CDM 进行线下迁移。否则，请选择线上迁移。



带宽 / 数据量	10Mbps	100Mbps	1Gbps	10Gbps
10GB	3小时	17分钟	2分钟	10秒
100GB	30小时	3小时	17分钟	2分钟
1TB	12.5天	30小时	3小时	17分钟
10TB	125天	12.5天	30小时	3小时
100TB	3.5年	125天	12.5天	30小时
1PB	35年	3.5年	125天	12.5天
10PB	350年	35年	3.5年	125天

注意：
1MB 以下的小文件数量较多、磁盘 IO 性能不足等等也会影响到数据的迁移进度。

2. 第三方云存储上的数据迁移至对象存储 COS

对于使用第三方云存储的用户，想要实现跨云平台进行数据迁移，对象存储 COS 支持有以下两种迁移方式可供选择：

- 线上迁移：COS Migration、迁移服务平台 MSP

这两种迁移方式都支持查看数据迁移进度、文件一致性校验、失败重传、断点续传等功能，能够满足用户数据基本的迁移需求。但这两种迁移方式在交互形式和功能特点等方面又有所差别，如下表所示。用户可根据以下差异性对比表，选择最适合的一种方式进行数据迁移。值得注意的是，使用迁移服务平台 MSP 进行数据迁移需要申请使用权限。

迁移方式	交互形式	区分大小文件的阈值	迁移并发度	HTTPS 安全传输
COS Migration	修改配置文件，非可视化操作	可自定义调整	针对大文件小文件分别定义并发度	可选择是否开启，关闭有利于加快迁移速度
迁移服务平台 MSP	可视化页面操作	采用默认设置	全局统一	开启

3. 数据以 URL 作为源地址迁移至对象存储 COS

对于用户想要使用 URL 列表作为数据源地址进行数据迁移。对象 COS 支持有以下 3 种方式：

- 线上迁移：COS Migration、迁移服务平台 MSP、COS 镜像回源

COS Migration 和迁移服务平台 MSP 的选择可参考上面场景 2 的介绍部分。在数据源站迁移上云的过程中，如果用户只希望把数据源站中的热数据全部迁移至云端，而冷数据保留在源站，可以采取 COS 镜像回源 的迁移方式。COS 镜像回源可以实现将有读写请求访问的热数据迁移至对象存储 COS，自动对低频访问的业务数据进行冷热分层。

4. 对象存储 COS 之间的数据迁移

对于正在使用对象存储 COS 的用户，如需要将一个存储桶的数据迁移至同个账号下的另一个存储桶（或者跨账号跨地域的存储桶），可以使用对象存储 COS 提供的跨区域复制功能进行数据迁移，目前该功能还没正式上线，敬请期待。

迁移实践

COS Migration

COS Migration 是一个集成了 COS 数据迁移功能的一体化工具。用户只需要通过简单的配置操作，便可将数据快速迁移至 COS 中。

迁移操作有以下几个步骤：

1. 安装 Java 环境
2. 安装 COS Migration 工具
3. 修改配置文件
4. 启动工具

具体的操作方法，请参阅 COS Migration 文档。

操作技巧

下面介绍如何配置 COS Migration 能最大程度提高迁移速度：

1. 根据自身网络环境调整区分大小文件的阈值和迁移并发度，实现大文件分块，小文件并发传输的最佳迁移方式。
   在配置文件 `config.ini` 中 `[common]` 分节，修改 `smallFileThreshold` 小文件阈值参数，大于等于这个阈值使用分块上传 ,默认设置为5MB。分别修改大文件并发度 `bigFileExecutorNum` 和小文件并发度 `smallFileExecutorNum` ，如果是通过外网来连接 COS，且带宽较小，请减小该并发度。小文件与大文件的并发度分别默认为 64 和 8。

2. 调整工具执行时间和设立带宽限制，保证自身业务运行不受迁移数据带宽占用影响。
   在配置文件 `config.ini` 中 `[common]` 分节，修改参数 `executeTimeWindow` ，该参数定义迁移工具每天执行

的时间段 ，其他时间则会进入休眠状态，休眠态暂停迁移并会保留迁移进度, 直到下一个时间窗口自动继续执行。

3. 采用分布式并行传输可以进一步加快迁移速度。用户可以考虑使用多台机器安装 COS Migration 并分别执行不同源数据的迁移任务。

迁移服务平台 MSP

迁移服务平台 MSP 是集成了多种迁移工具，并且提供可视化界面的平台，能够帮助用户轻松监控和管理大规模的数据迁移任务。其中"文件迁移工具"能帮助用户将数据从各类公有云和数据源站中迁移至对象存储 COS。

迁移操作有以下几个步骤：

1. 通过控制台导航栏中的【迁移工具】找到"文件迁移工具" 入口并启用。
2. 新建迁移任务并配置任务信息。
3. 启动任务。

具体操作请查阅 迁移工具 文档。

操作技巧

在进行数据迁移过程中，数据源的读取速度会因为不同的网络环境而有所不同，但客户根据实际状况在"新建文件迁移任务"时选择较高的QPS并发度，有助于提高迁移速度 。

云数据迁移 CDM

云数据迁移 CDM 是利用腾讯云提供的离线迁移专用设备，帮助用户将本地数据迁移至云端的一种迁移方式，可解决本地数据中心通过网络传输迁移云端时间长、成本高、安全性低的问题。

迁移操作主要有以下几个步骤：

1. 前往云数据迁移 CDM 控制台提交申请。
2. 申请审核通过后，用户等待签收设备。
3. 收到设备后，按照迁移设备手册把数据拷贝至设备。
4. 完成数据拷贝后，在控制台提交回寄申请并等待腾讯云把数据迁往对象存储 COS。

详情请参考 云数据迁移 CDM 产品文档。

操作技巧

下面介绍如何高效安全通过离线迁移迁移数据至腾迅云对象存储 COS。

1. 在 IDC 配置 10Gbps 的网络环境，为避免本地数据环境成为传输瓶颈，您可配备高性能的挂载点机器，最大程度加速拷贝。
2. 适用 CDM 传输数据最快的方式是并行传输数据，用户通过监控设备的 CPU 和内存使用率，如果当前迁移速度低于预期，可以选择以下并行传输方式。
   - 多台设备通过不同网络接口连接同一个 CDM 设备。

- 多台设备通过不同网络接口连接多台 CDM 设备。

COS 镜像回源

COS 镜像回源是把数据源站中有读写访问请求的数据自动迁移至腾讯云的对象存储 COS。此迁移方式不仅可以帮助用户快速对数据进行冷热分层，还能加快业务系统中热数据的读写访问速度。

迁移操作主要有以下几个步骤：

1. 进入对象存储的控制台，在待迁移数据的目的 bucket 中开启回源设置。
2. 配置 bucket 回源地址，然后保存。
3. 将业务系统的读写请求转到腾讯云对象存储 COS 上。

具体操作请参考 回源设置 控制台指南文档。

操作技巧

以下步骤可以完成源数据的冷热分离，热数据无缝迁移到腾讯云对象存储 COS，以便加快热数据的读取请求速度。

1. 将业务系统的读写请求切换到 COS 上，在 COS 控制台打开回源设置功能，回源地址为数据源站，此时系统结构如下图所示。

2. 一段时间后，冷数据仍然在源站，但热数据已经迁移至腾讯云对象存储 COS。迁移过程中业务系统不受影响。