对象存储

实践教程

产品文档

腾讯云

文档目录

实践教程

概览

最近更新时间：2024-01-06 10:47:50

对象存储（Cloud Object Storage，COS）提供了多方面的常见应用场景及其实践操作详解，包括访问控制与权限管理、性能优化、数据迁移、数据直传与备份、数据安全域名管理、大数据、无服务架构等实践场景，能够帮助您更快速、更方便地使用 COS 来实现您的多样化业务需求，具体实践请参见下表：

最佳实践	说明
访问控制与权限管理	访问控制与权限管理是腾讯云 COS 最实用的功能之一，本目录实践将帮助您了解如下： ACL 访问控制 CAM 访问管理 授权子账号访问 COS 权限设置 COS API 授权策略 使用临时密钥安全指引 临时密钥生成及使用 授权子账号拉取标签列表 条件键说明及使用 授权其他主账号下的子账号操作存储桶
性能优化	腾讯云 COS 支持性能扩展，以获取更高的请求速率。获取更高请求速率的操作步骤，请参见 请求速率与性能优化。
使用 AWS S3 SDK 访问 COS	COS 提供了 AWS S3 兼容的 API，本实践介绍了如何通过简单的配置修改，就可以使用 S3 SDK 的接口来访问 COS 上的文件。
数据容灾备份	本实践列出了如下三种备份场景，并针对这三种迁移场景分别给出适用的备份方案。 基于跨地域复制的容灾高可用架构 云上数据备份 本地数据备份
域名管理实践	介绍如何配置 HTTPS 自定义域名访问腾讯云 COS。详情请参见 配置自定义域名支持 HTTPS 访问。 介绍如何在腾讯云 COS 中配置跨域访问规则。详情请参见 设置跨域访问。 介绍如何在腾讯云 COS 中托管静态网站。详情请参见 托管静态网站。 介绍如何在腾讯云 COS 中搭建前端单页应用。详情请参见 使用 COS 静态网站功能搭建前端单页应用。
数据直传实践	本实践介绍了以下数据直传的应用场景： Web 端直传实践 小程序直传实践

	移动应用直传实践 uni-app 直传实践
数据安全	以事前防护、事中监控和事后追溯三个方面为用户介绍数据安全解决方案。详情请参见 数据安全概述。 介绍如何在腾讯云 COS 中配置防盗链，实现对访问来源的管控。详情请参见 防盗链实践。
数据检验	介绍如何在腾讯云 COS 中通过 MD5 校验的方式保证上传数据的完整性。详情请参见 MD5 校验。 介绍如何通过 CRC64 检验码来进行数据校验，详情请参见 CRC64 校验。
大数据实践	介绍如何将腾讯云 COS 作为 Druid 的 Deep storage 的操作步骤。详情请参见 使用 COS 作为 Druid 的 Deep storage。 介绍如何使用 Terraform 管理腾讯云 COS。 介绍如何使用 DataX 导入或导出腾讯云 COS。详情请参见 使用 DataX 导入或导出 COS。 介绍如何使用 CDH 配置 COSN。详情请参见 CDH 配置 COSN 指引。 介绍什么是腾讯云 COS Ranger 权限体系解决方案。详情请参见 COS Ranger 权限体系解决方案。 介绍如何使用流计算 Oceanus 接入腾讯云 COS。详情请参见 使用流计算 Oceanus 接入 COS。
在第三方应用中使用 COS	介绍如何在兼容 S3 的第三方应用中使用 COS 的通用配置，将数据保存在腾讯云 COS 上，详情请参见 在兼容 S3 的第三方应用中使用 COS 的通用配置。 介绍如何通过配置远程附件功能将论坛的附件保存在腾讯云 COS 上。 介绍如何在 WordPress 通过第三方插件将多媒体内容保存在腾讯云 COS上，详情请参见 将 WordPress 多媒体内容存储到 COS。
通过 API 进行多文件打包压缩	本文主要为用户介绍如何通过 API 进行多文件打包压缩。

访问控制与权限管理
ACL 访问控制实践

最近更新时间：2024-01-06 10:47:50

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 可向其他主账号、子账号和用户组，授予基本的读、写权限。

与访问策略有所不同的是，ACL 的管理权限有以下限制：

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

不支持显式拒绝效力

ACL 支持的控制粒度：

存储桶（Bucket）

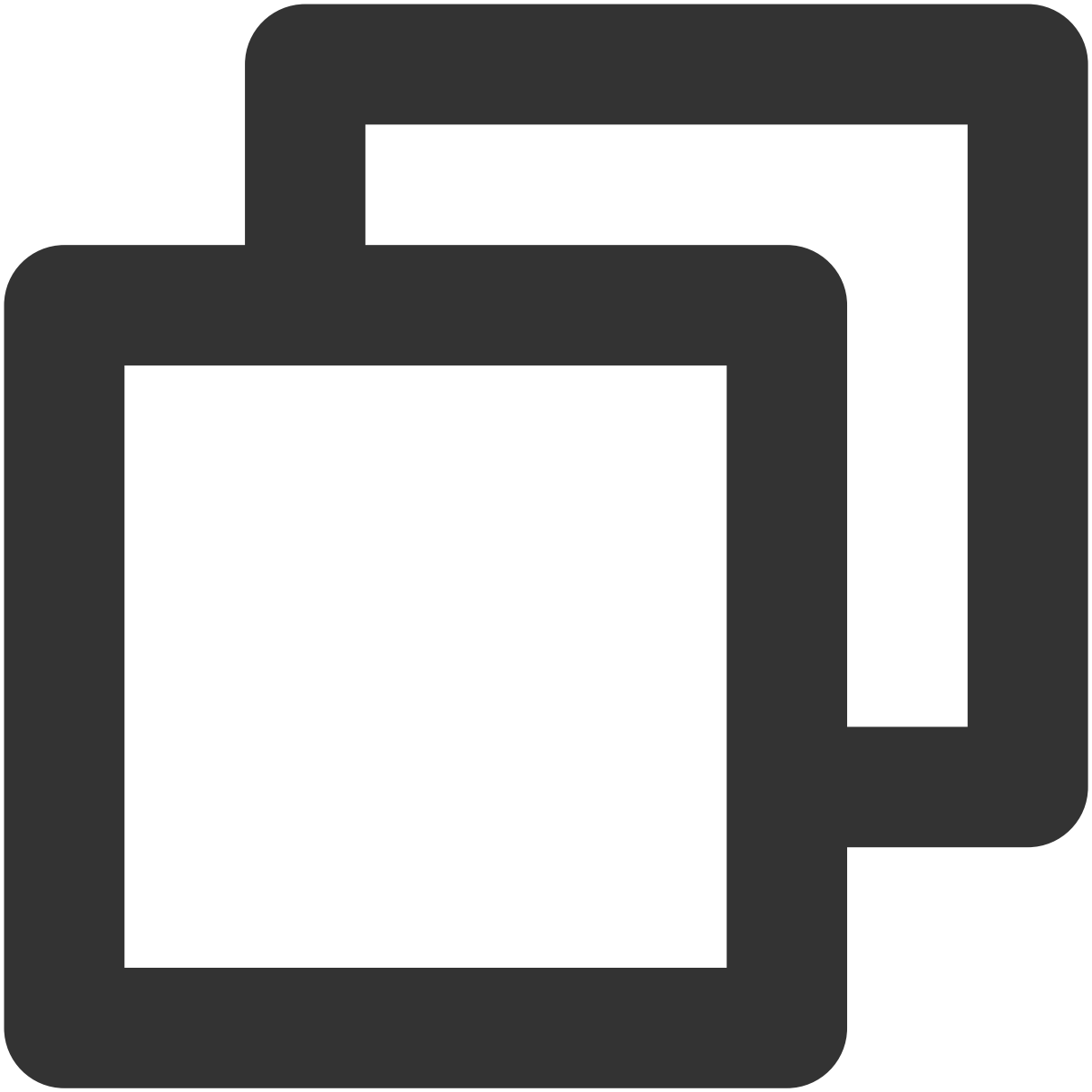
对象键前缀（Prefix）

对象（Object）

ACL 的控制元素

当创建存储桶或对象时，其资源所属的主账号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。其中的100000000001表示主账号，100000000011为主账号下的子账号，100000000002表示另一个主账号。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 `http://cam.qcloud.com/groups/global/AllUsers` 的 READ 权限。



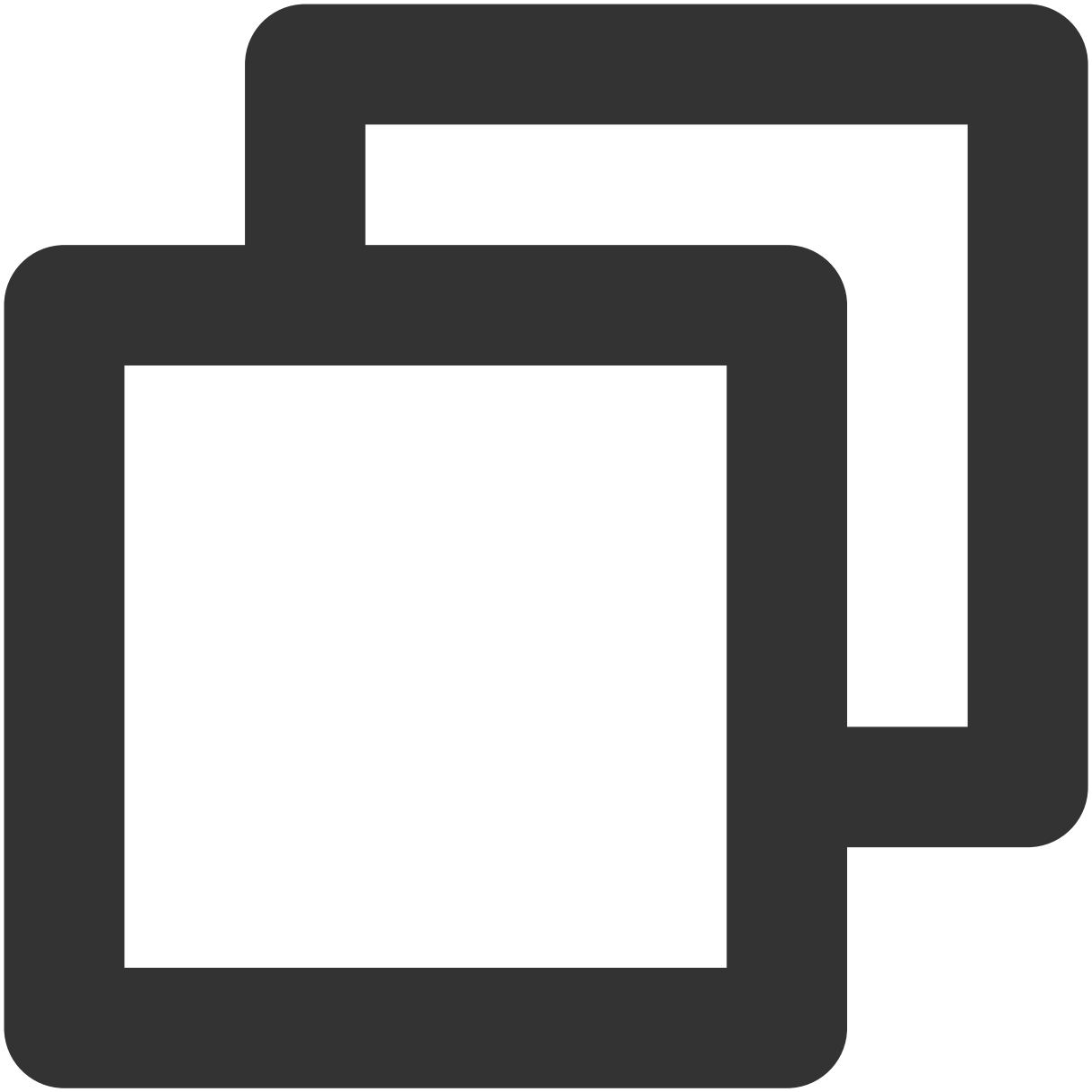
```
<AccessControlPolicy>
  <Owner>
    <ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
    <DisplayName>qcs::cam::uin/100000000001:uin/100000000001</DisplayName>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Root
        <ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
        <DisplayName>qcs::cam::uin/100000000001:uin/100000000001</DisplayName>
      </Grantee>
```

```
        <Permission>FULL_CONTROL</Permission>
      </Grant>
      <Grant>
        <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Grou
          <URI>http://cam.qcloud.com/groups/global/AllUsers</URI>
        </Grantee>
        <Permission>READ</Permission>
      </Grant>
    </AccessControlList>
  </AccessControlPolicy>
```

权限被授予者

主账号

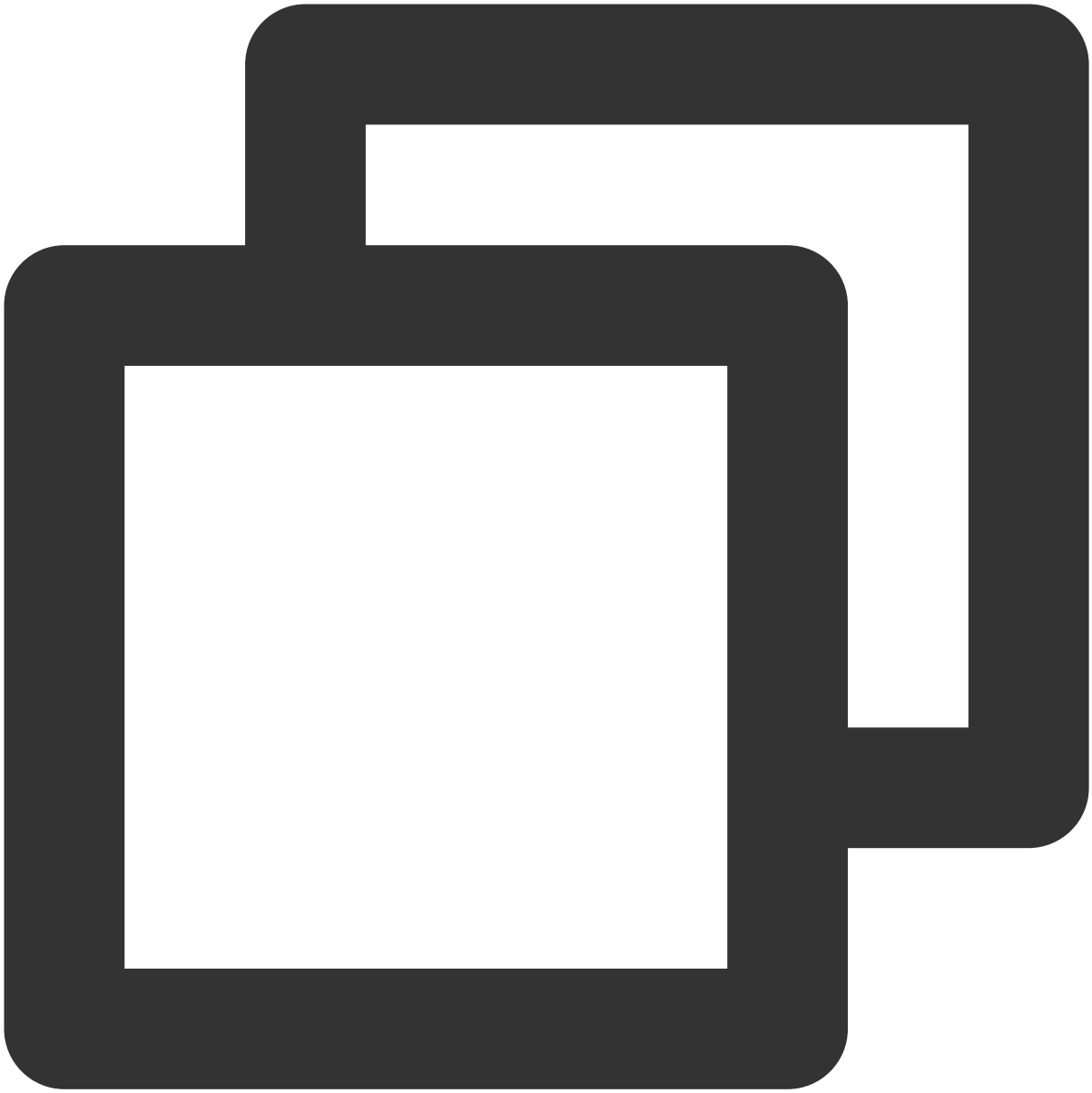
您可以对其他主账号授予用户访问权限，使用 CAM 中对委托人（principal）的定义进行授权。描述为：



```
qcs::cam::uin/100000000002:uin/100000000002
```

子账号

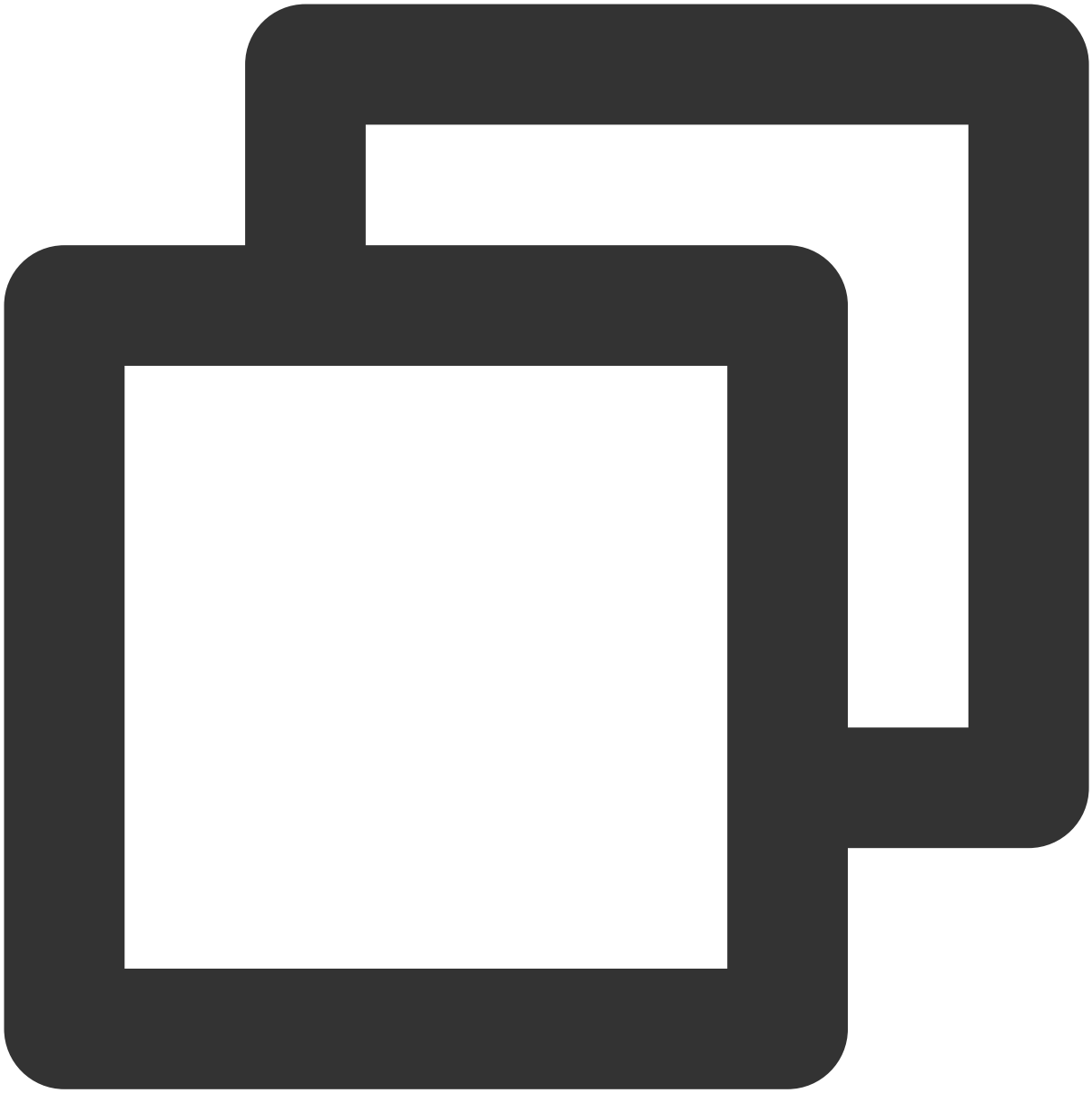
您可以对您的主账号下的子账号（如100000000011），或其他主账号下的子账号授权，使用 CAM 中对委托人（principal）的定义进行授权。描述为：



```
qcs::cam::uin/100000000001:uin/100000000011
```

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：



```
http://cam.qcloud.com/groups/global/AllUsers
```

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
WRITE	创建、覆盖和删除存储桶中的	创建、覆盖和删除目录下	不支持

	任意对象	的任意对象	
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

由于主账号始终拥有 FULL_CONTROL 权限，以下表格中不再对此特别说明。

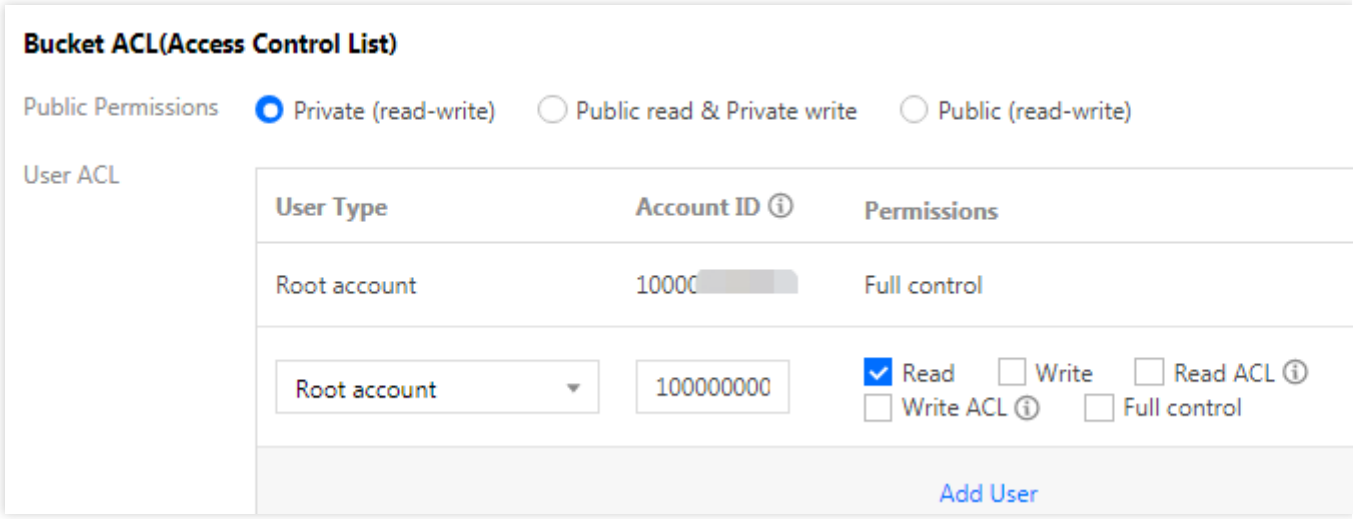
标准 ACL	含义
(空)	此为默认策略，其他人无权限，资源继承上级权限
private	其他人没有权限
public-read	匿名用户组具备 READ 权限
public-read-write	匿名用户组具备 READ 和 WRITE 权限，通常不建议在存储桶赋予此权限

使用方法

使用控制台操作 ACL

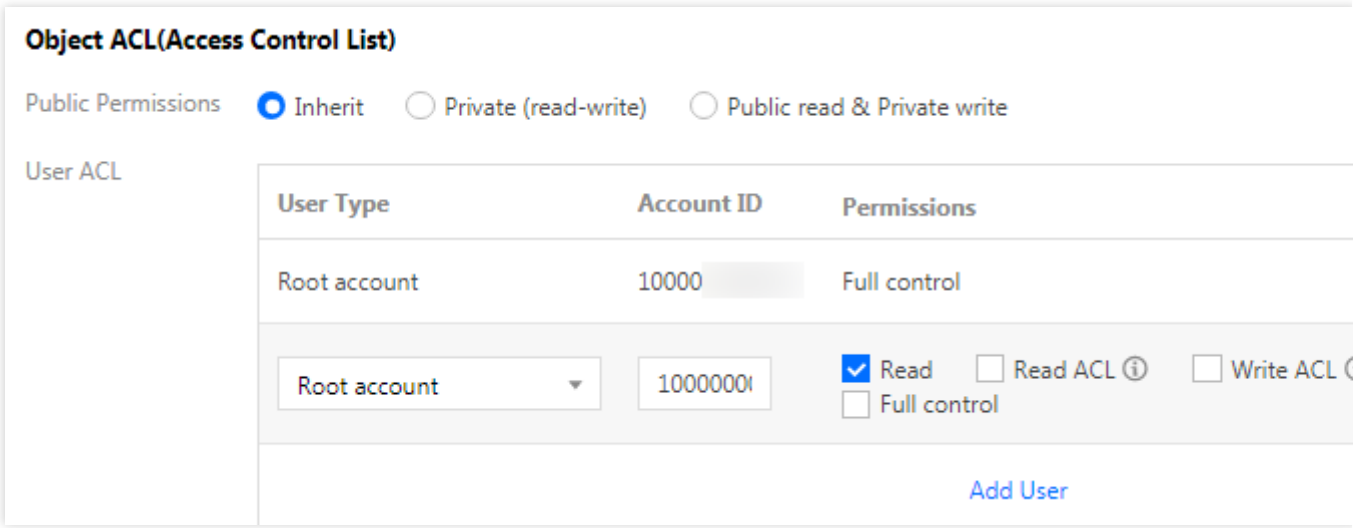
对存储桶设置 ACL

以下示例表示允许另一个主账号对某个**存储桶**有读取权限：



对对象设置 ACL

以下示例表示允许另一个主账号对某个**对象**有读取权限：



注意

如使用子账号访问存储桶或对象出现**无权限访问**的提示，请先通过主账号为子账号授权以便能够正常访问存储桶，操作步骤请参见 子账号访问存储桶列表。

使用 API 操作 ACL

存储桶 ACL

API	操作名	操作描述
PUT Bucket acl	设置存储桶 ACL	设置指定存储桶访问权限控制列表
GET Bucket acl	查询存储桶 ACL	查询存储桶的访问控制列表

对象 ACL

API	操作名	操作描述
PUT Object acl	设置对象 ACL	设置存储桶中某个对象的访问控制列表
GET Object acl	查询对象 ACL	查询对象的访问控制列表

访问管理实践

最近更新时间：2024-01-06 10:47:50

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。

功能

主账号资源的授权

可以将主账号的资源授权给其他人员，包括子账号或者是其它主账号，而不需要分享主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对象存储（Cloud Object Storage，COS）存储桶的读权限，而另外一些子账号或者主账号可以拥有某个 COS 存储对象的写权限等。上述资源、访问权限和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。

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。例如，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。例如，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。

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）。如需了解更多详情，请参见 访问策略语言概述。

说明

策略语法在描述上没有顺序要求。操作（action）元素需区分英文大小写。

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。

核心元素

核心元素	描述	是否必填
版本 version	描述策略语法版本。目前仅允许值为2.0	是
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	是
操作 action	用于描述允许或拒绝的操作。操作可以是单个 API 操作或者多个 API 操作的集合。操作（action）元素需区分英文大小写，例如 action 为 `name/cos:GetService`	是
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	是
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	否
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	是

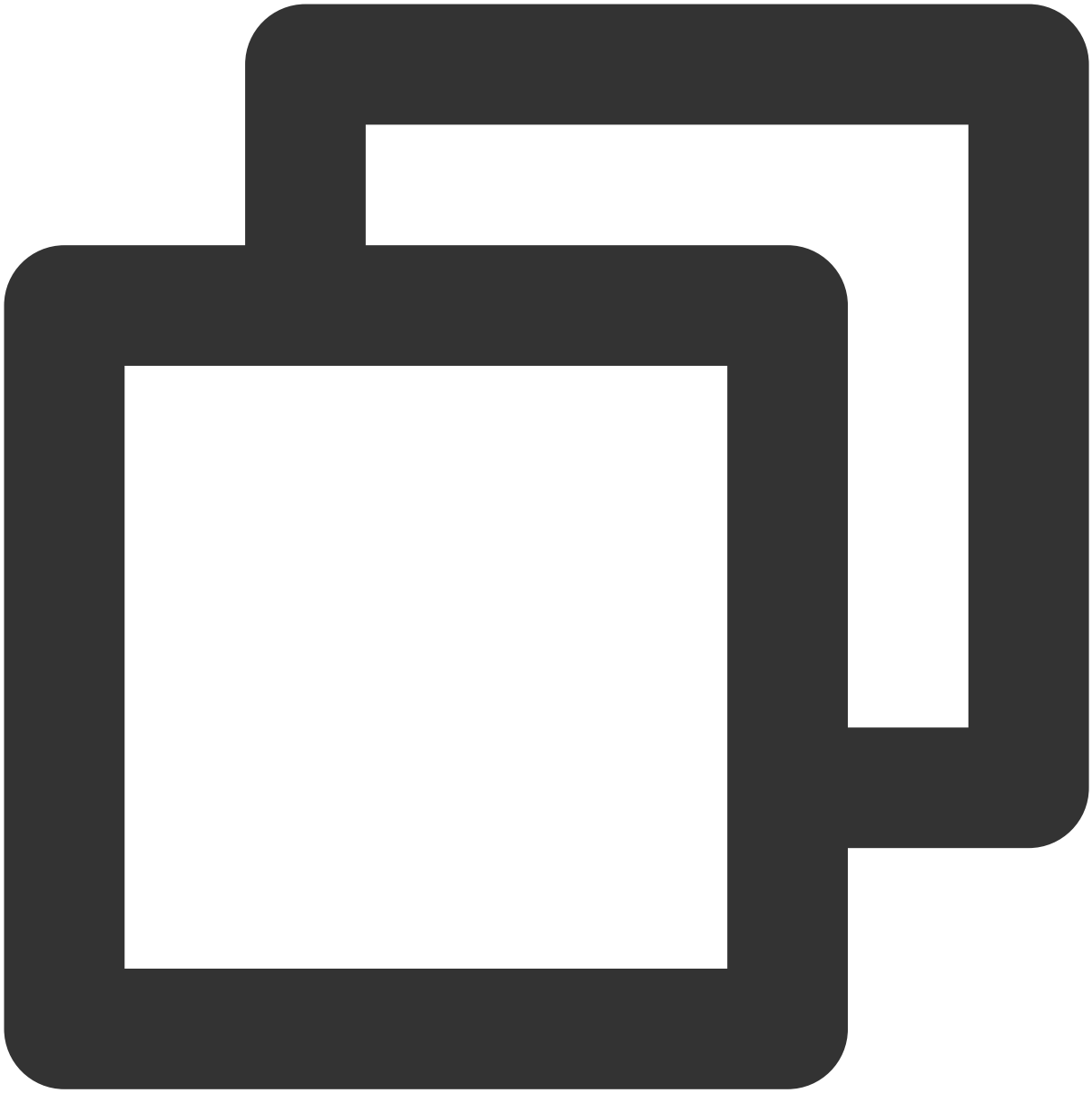
策略限制

限制项	限制值
一个主账号中的用户组数	300

一个主账号中的子账号数	1000
一个主账号中的角色数	1000
一个子账号可加入的用户组数	10
一个协作者可协作的主账号数	10
一个用户组中的子账号数	100
一个主账号可创建的自定义策略数	1500
直接关联到一个用户、用户组或角色的策略数	200
一个策略语法最大字符数	4096

策略示例

以下策略示例描述为：允许属于主账号 ID 为100000000001（APPID 为1250000000）下的子账号 ID 100000000011， 对北京地域的存储桶 examplebucket-bj 和广州地域的存储桶 examplebucket-gz 下的对象 exampleobject，在访问 IP 为 `10.*.*.10/24` 网段时，拥有**上传对象**和**下载对象**的权限。



```
{
            "version": "2.0",
            "principal": {
                "qcs": ["qcs::cam::uin/100000000001:uin/100000000011"]
            },
            "statement": [{
                "effect": "allow",
                "action": ["name/cos:PutObject", "name/cos:GetObject"],
                "resource": ["qcs::cos:ap-beijing:uid/1250000000:examplebucket-bj-1
                    "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-gz-12500000
                ],
```

```
                "condition": {
                    "ip_equal": {
                        "qcs:ip": "10.*.*.10/24"
                    }
                }
            }]
}
```

授权子账号访问 COS

最近更新时间：2024-01-06 10:47:50

概述

对于腾讯云对象存储（Cloud Object Storage，COS）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。您可通过访问管理（Cloud Access Management，CAM）对存储桶或对象设置不同的操作权限，使得不同团队或人员能够相互协作。
首先，我们需要先了解几个关键概念：主账号、子账号（用户）和用户组。CAM 的相关术语、配置详细描述请参见访问管理的 词汇表。

主账号

主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主账号身份。主账号是腾讯云资源使用计量计费的基本主体。
主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，资源只能被主账号所访问，任何其他用户访问都需要获得主账号的授权。

子账号（用户）和用户组

子账号是由主账号创建的实体，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，必须由所属主账号进行授权。
一个主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主账号，分别协助多个主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主账号下，管理该主账号的云资源。
子账号可以通过控制台切换开发商（主账号），从一个主账号切换到另外一个主账号。
子账号登录控制台时，会自动切换到默认主账号上，并拥有默认主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主账号授权的访问权限，而切换前的主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。

操作步骤

授权子账号访问 COS 分为三个步骤：创建子账号、对子账号授予权限、子账号访问 COS 资源。

步骤1：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。

2. 选择**用户 > 用户列表 > 新建用户**，进入新建用户页面。
3. 选择**自定义创建**，选择**可访问资源并接收消息**类型，单击**下一步**。
4. 按照要求填写用户相关信息。

设置用户信息：输入子用户名称，例如 Sub_user。输入子用户的邮箱，您需要为子用户添加邮箱来获取由腾讯云发出的绑定微信的邮件。

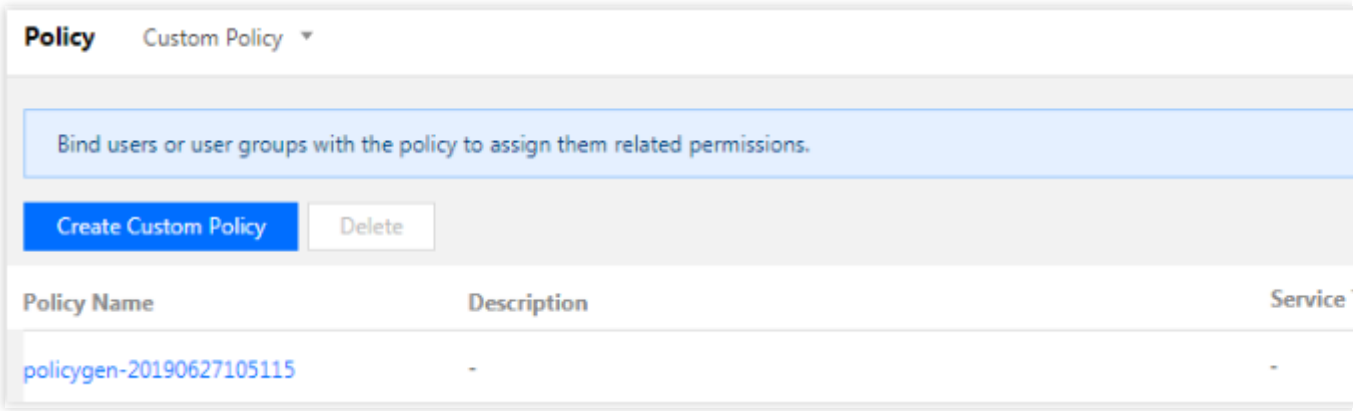
访问方式：选择编程访问和腾讯云控制台访问。其他配置可按需选择。

5. 填写用户信息完毕后，单击**下一步**，进行身份验证。
6. 身份验证完毕，设置子用户权限。根据系统提供的策略选择，可配置简单的策略，例如 COS 的存储桶列表的访问权限，只读权限等。如需配置更复杂的策略，可进行 步骤2：对子账号授予权限。
7. 设置用户标签，该项为可选项，可按需设置，单击**下一步**。
8. 确认配置信息无误后，单击**完成**即可创建子账号。

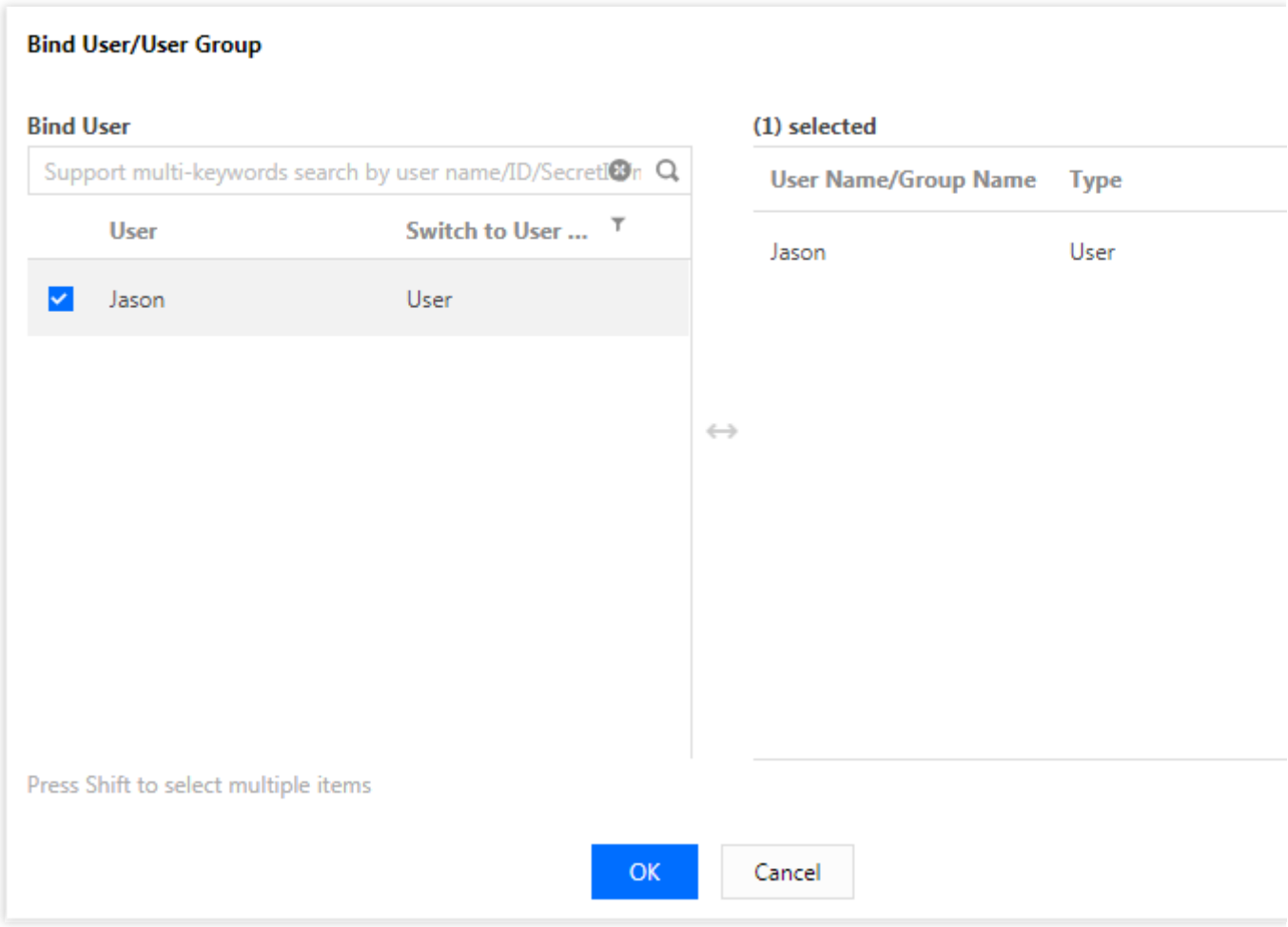
步骤2：对子账号授予权限

创建自定义策略或者选择已有策略，并将策略关联到子账号。

1. 登录 CAM 控制台。
2. 选择**策略 > 新建自定义策略 > 按策略语法创建**，进入策略创建页面。
3. 您可按照实际需求选择**空白模板**自定义授权策略，或选择与 COS 相关联的**系统模板**，单击**下一步**。
4. 输入便于您记忆的策略名称，若您选择**空白模板**，则需要输入您的策略语法，详情请参见 策略示例。您可将策略内容复制粘贴到**策略内容**编辑框内，确认输入无误后单击**完成**即可。
5. 创建完成后，将刚才已创建的策略关联到子账号。



6. 勾选子账号并单击**确定**授权后，即可使用子账号访问所限定的 COS 资源。



步骤3：子账号访问 COS 资源

根据上面步骤1设置的两种访问方式：编程访问和腾讯云控制台访问两种，介绍如下：

（1）编程访问

当使用子账号通过编程（例如 API、SDK 和工具等）访问 COS 资源时需要先获取主账号的 APPID、子账号的 SecretId 和 SecretKey 信息。您可以在访问管理控制台生成子账号的 SecretId 和 SecretKey。

1. 主账号登录 CAM 控制台 。
2. 选择**用户列表**，进入用户列表页面。
3. 单击子账号用户名称，进入子账号信息详情页。
4. 单击 **API 密钥**页签，并单击**新建密钥**为该子账号创建 SecretId 和 SecretKey。

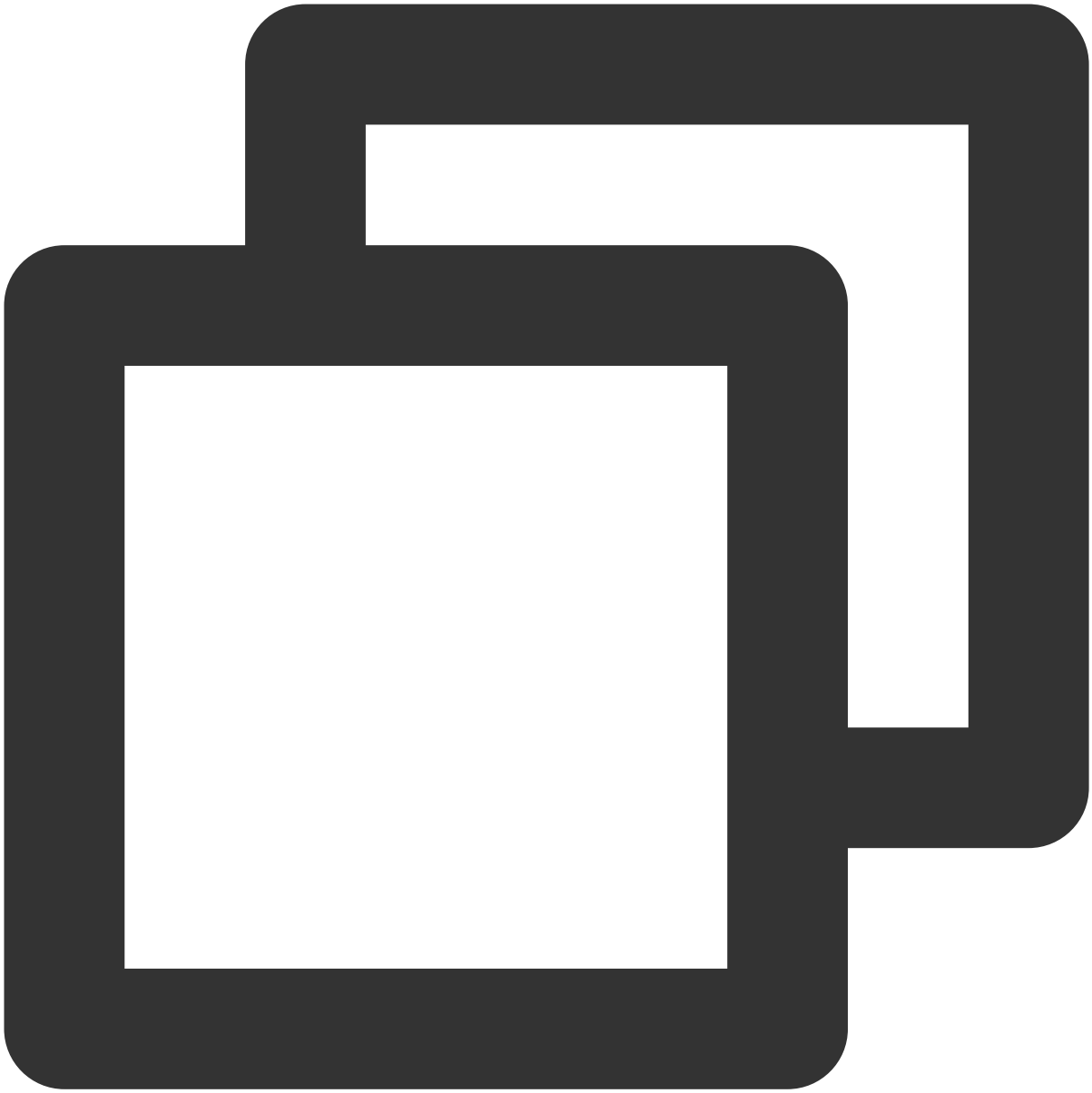
至此您就可以通过子账号的 SecretId 和 SecretKey、主账号的 APPID，访问 COS 资源。

注意

子账号需通过 XML API 或基于 XML API 的 SDK 访问 COS 资源。

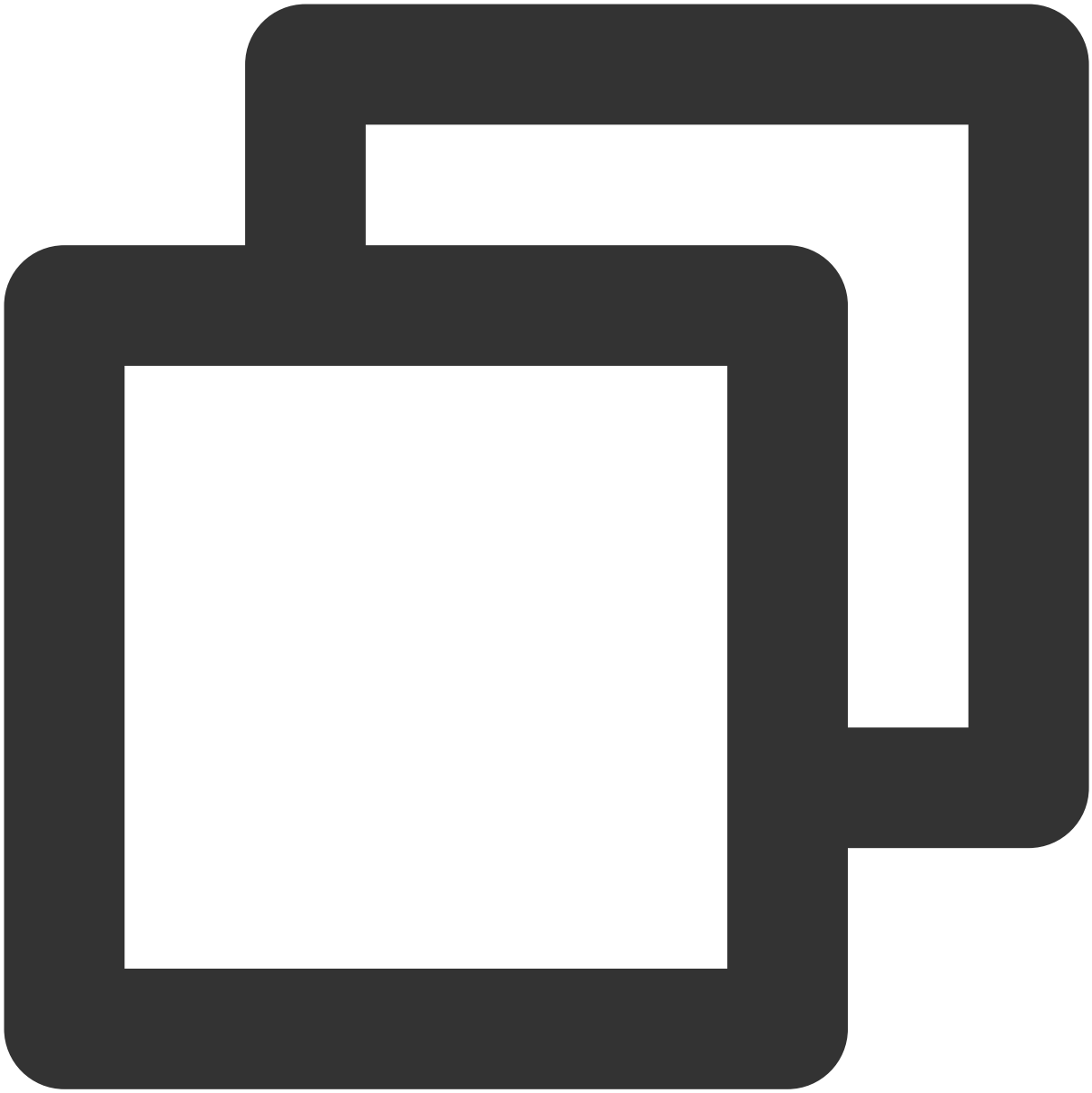
基于 XML 的 Java SDK 访问示例

以基于 XML 的 Java SDK 命令行为例，需填入参数如下：



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

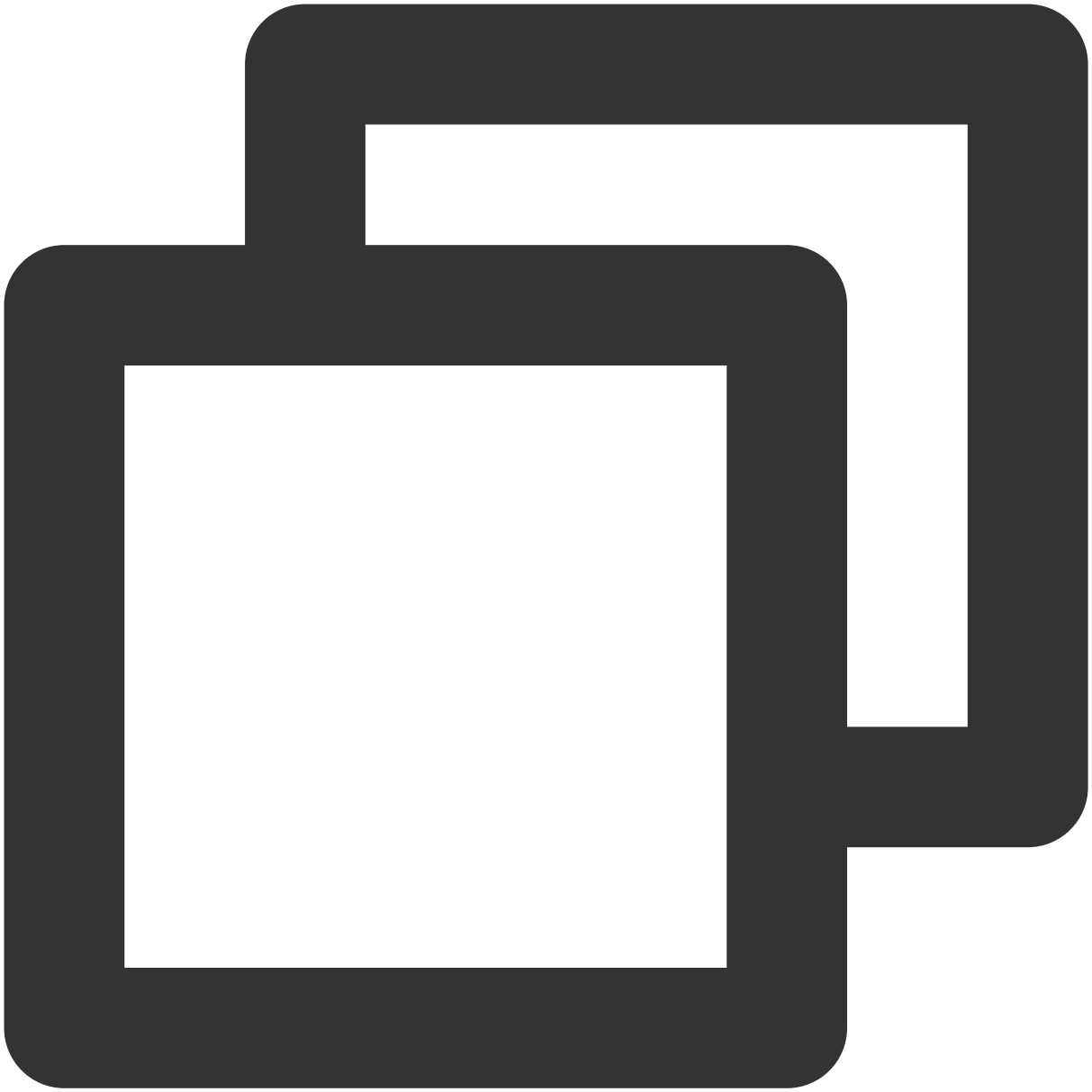
实例如下：



```
String secretId = System.getenv("secretId");//子账号的 SecretId，授权遵循最小权限指引，降
String secretKey = System.getenv("secretKey");//子账号的的 SecretKey，授权遵循最小权限指
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);

// 初始化身份信息
COSCredentials cred = new BasicCOSCredentials("<主账号 APPID>", secretId, secretKey)
```

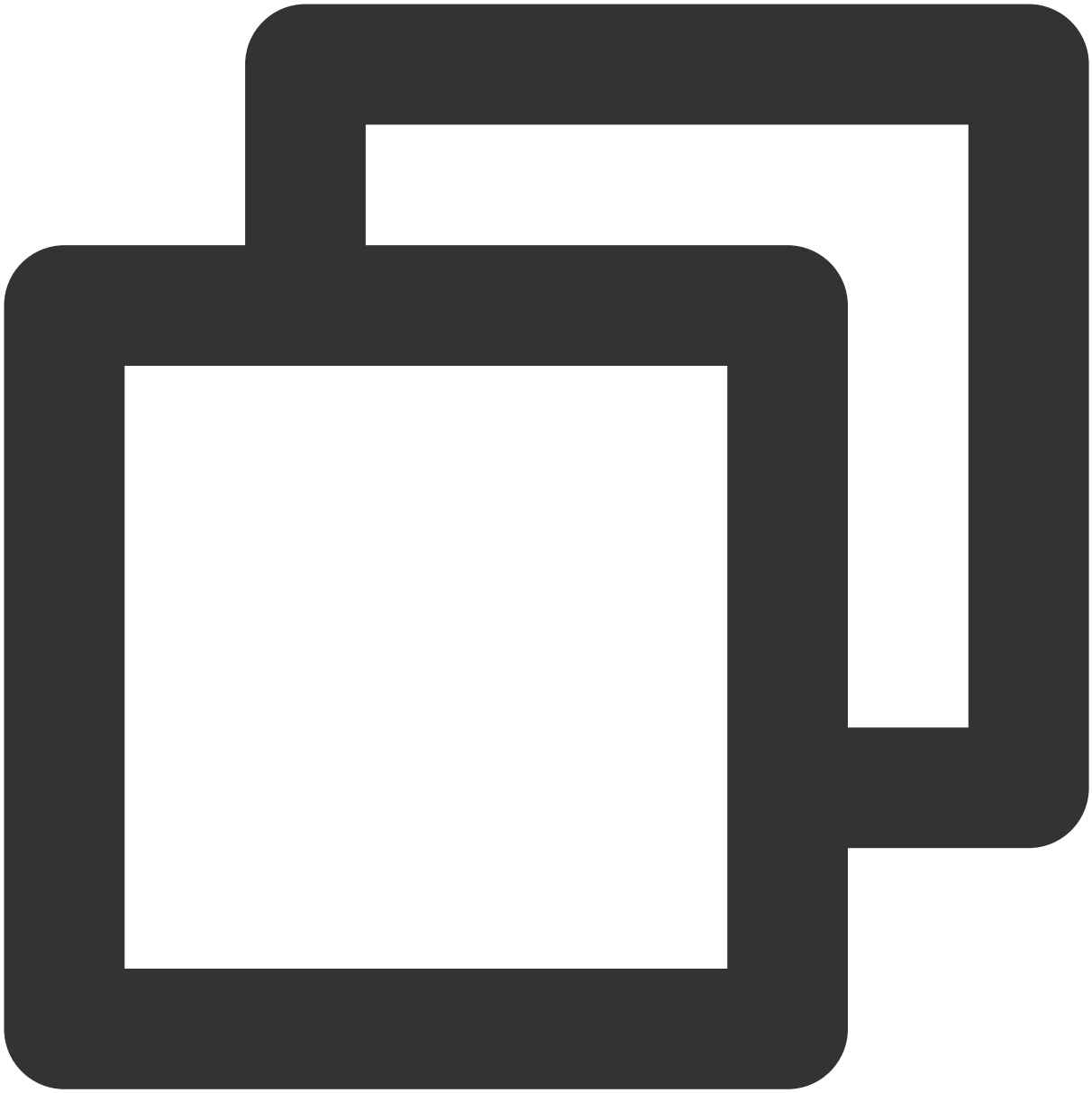
COSCMD 命令行工具访问示例

以 COSCMD 的配置命令为例，需填入的参数如下：



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

实例如下：



```
coscmd config -u 1250000000 -a AKIDasdfmRxHPa9oLhJp**** -s e8Sdeasdfas2238Vi**** -b
```

（2）腾讯云控制台

子用户被授予权限后，可在 子用户登录界面 输入主账号 ID、子用户名和子用户密码登录控制台，并在**云产品**中选择单击**对象存储**，即可访问主账号下的存储资源。

策略示例

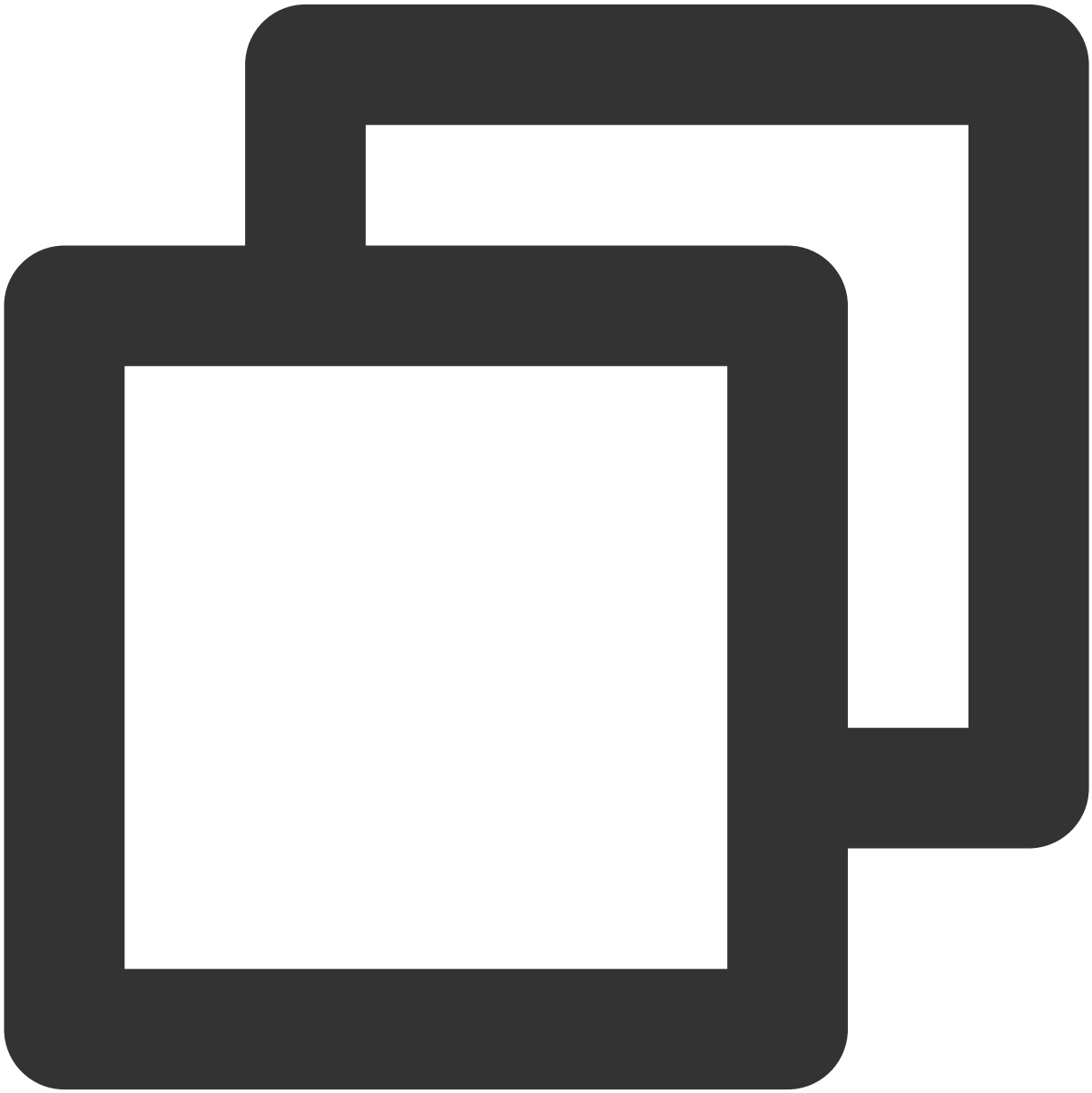
以下提供几种典型场景的策略示例，在配置自定义策略时，您可将以下参考策略复制粘贴至输入框**编辑策略内容**，根据实际配置修改即可。更多 COS 常见场景的策略语法请参见 访问策略语言概述 或 CAM 产品文档 的**商用案例**部分。

示例1：为子账户配置 COS 全读写权限

注意

该策略授权的范围较大，请谨慎配置。

具体策略如下所示：

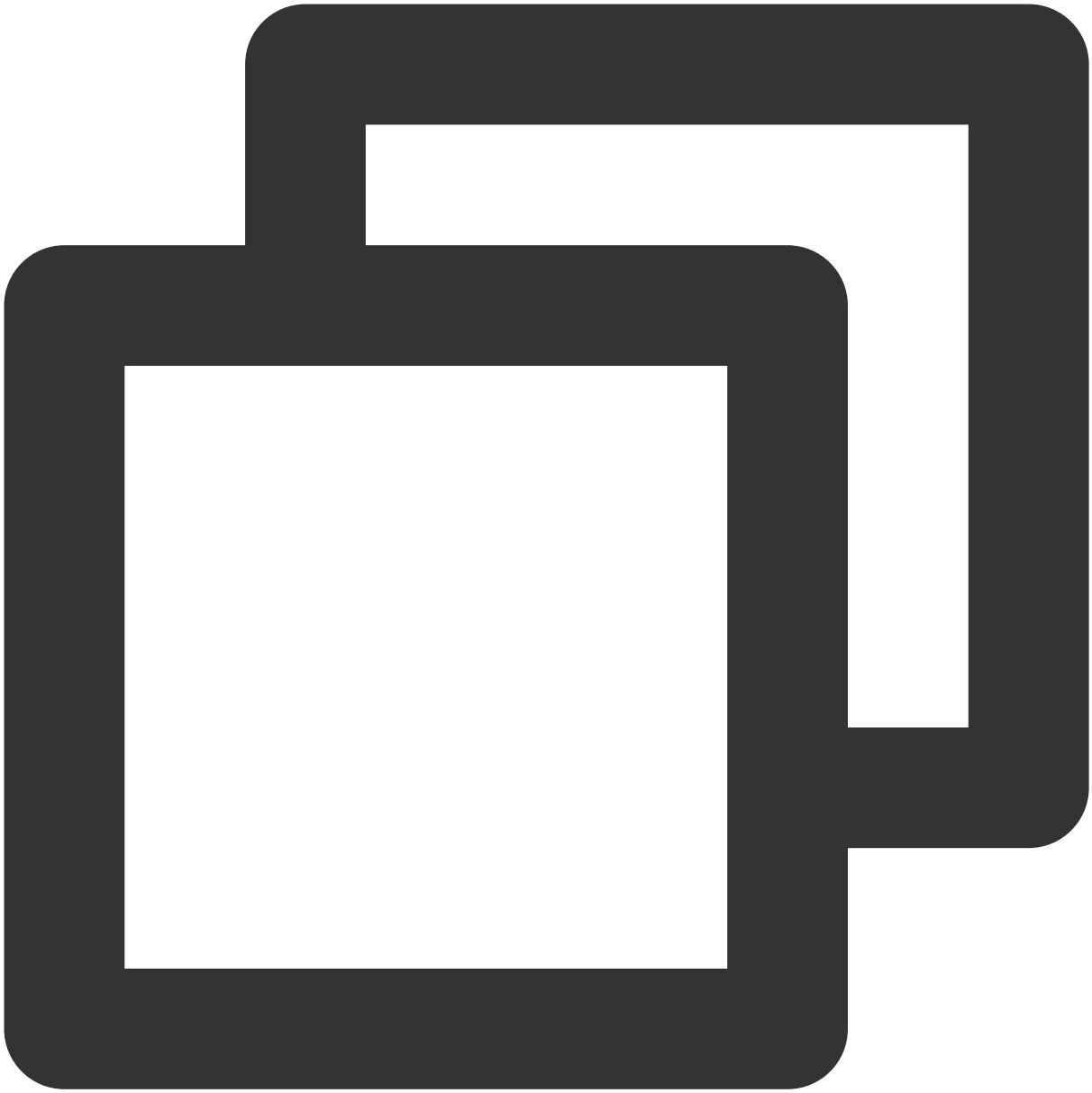


```
{
```

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

示例2：为子账户配置只读权限

为子账户仅配置只读权限，具体策略如下所示：

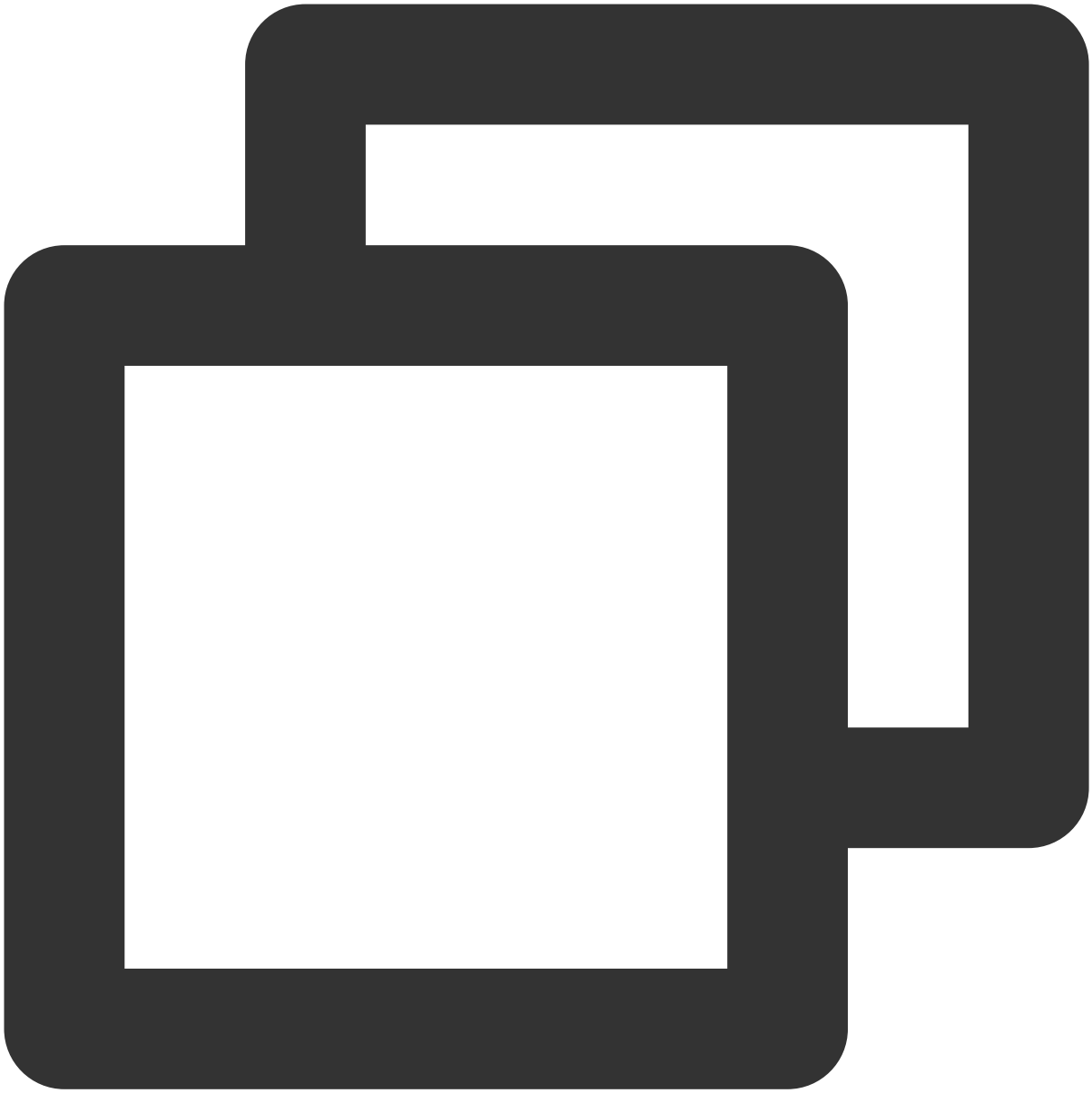


```
{
    "version": "2.0",
    "statement": [
        {
            "action": [
                "name/cos:List*",
                "name/cos:Get*",
                "name/cos:Head*",
                "name/cos:OptionsObject"
            ],
            "resource": "*",
```

```
            "effect": "allow"
        },
        {
            "effect": "allow",
            "action": "monitor:*",
            "resource": "*"
        }
    ]
}
```

示例3：为子账户配置只写权限（不含删除）

具体策略如下所示：

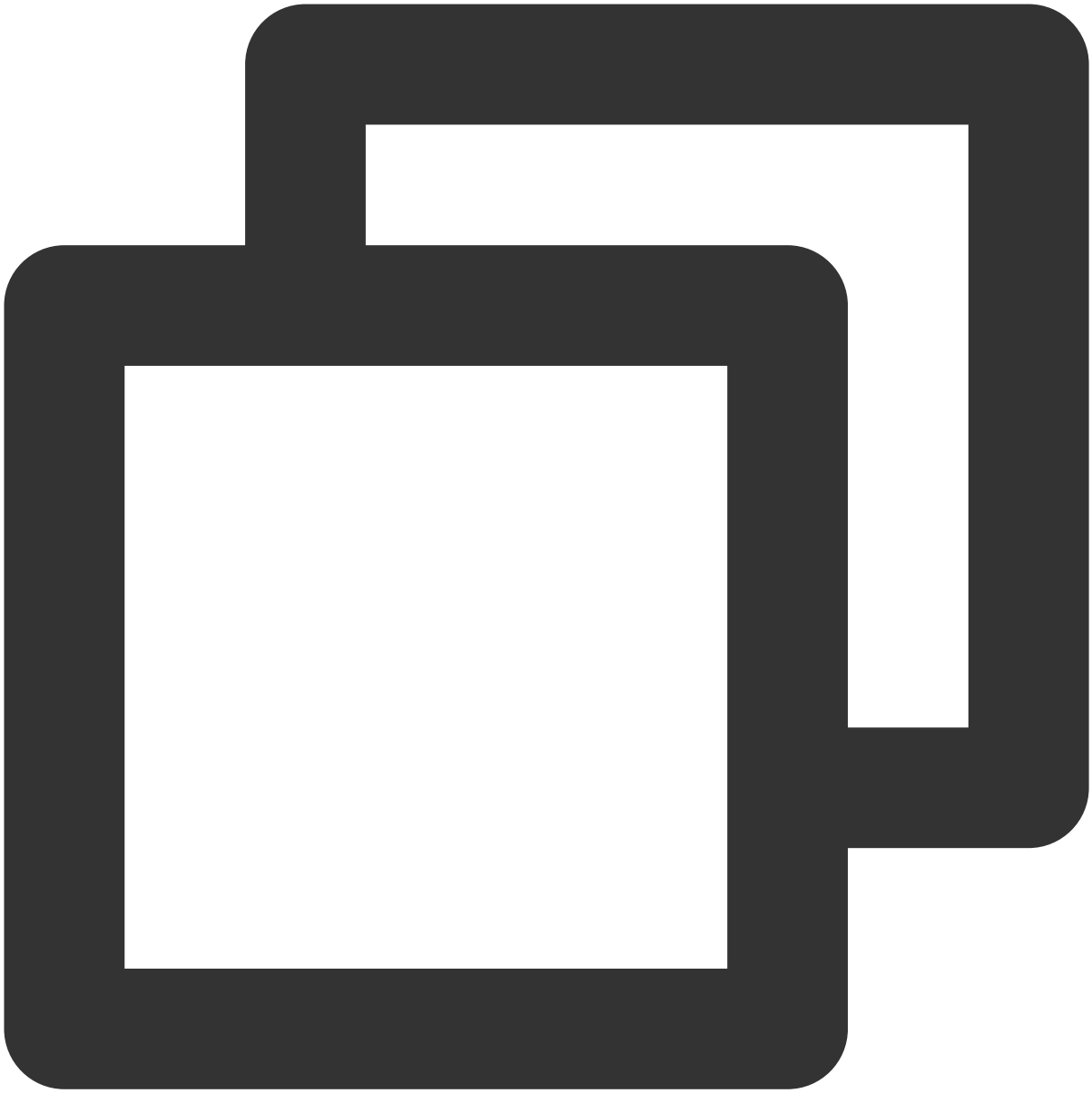


```
{
    "version": "2.0",
    "statement": [
        {
            "effect": "allow",
            "action": [
                "cos:ListParts",
                "cos:PostObject",
                "cos:PutObject*",
                "cos:InitiateMultipartUpload",
                "cos:UploadPart",
```

```
                "cos:UploadPartCopy",
                "cos:CompleteMultipartUpload",
                "cos:AbortMultipartUpload",
                "cos:ListMultipartUploads"
            ],
            "resource": "*"
        }
    ]
}
```

示例4：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，请参见 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：2024-01-06 10:47:50

通过存储桶策略（Policy）授权案例

准备工作

1. 创建存储桶

通过存储桶策略（Policy）授权仅针对特定的存储桶，因此您需要先 创建存储桶。如需针对账号维度进行授权，请参见本文的 通过访问管理（CAM）授权案例。

2. 准备被授权账号的 UIN

本文假设拥有目标存储桶的主账号 UIN 为100000000001，其下的子账号为100000000011，子账号需要被授权才能访问目标存储桶。

说明

如需查询主账号下所创建的子账号，可登录访问管理控制台，在 用户列表 中查看。

如需创建新的子账号，请参见 新建子用户 文档。

3. 打开**添加策略**对话框

进入目标存储桶的**权限管理**，选择 **Policy 权限设置 > 图形设置**，并单击打开**添加策略**对话框，随后请参见本文的授权案例进行配置。添加策略的详细操作指引，可参见 添加存储桶策略 文档。

以下列举了几种不同的授权案例，您可按照实际情况进行配置。

授权案例

案例一：授予子账号拥有特定目录的所有权限

配置信息如下：

配置项	配置值
效力	允许
用户	选择子账号，然后输入子账号的 UIN，该子账号必须为当前主账号下的子账号，例如100000000011
资源	选择指定资源路径，例如 `folder/sub-folder/*`
操作	选择所有操作

案例二：授予子账号拥有特定目录内所有文件的读权限

配置信息如下：

配置项	配置值

效力	允许
用户	选择子账号，然后输入子账号的 UIN，该子账号必须为当前主账号下的子账号，例如100000000011
资源	选择指定资源路径，例如 `folder/sub-folder/*`
操作	读操作(含列出对象列表)

案例三：授予子账号拥有特定文件的读写权限

配置信息如下：

配置项	配置值
效力	允许
用户	选择子账号，然后输入子账号的 UIN，该子账号必须为当前主账号下的子账号，例如100000000011
资源	选择指定对象键，例如 `folder/sub-folder/example.jpg`
操作	所有操作

案例四：授予子账号拥有特定目录下所有文件的读写权限，并禁止拥有该目录下指定文件的读写权限

针对此案例，我们需要添加两个策略：**允许**策略和**禁止**策略。

1. 首先添加**允许**策略，配置信息如下：

配置项	配置值
效力	允许
用户	选择子账号，然后输入子账号的 UIN，该子账号必须为当前主账号下的子账号，例如100000000011
资源	指定目录前缀，例如 `folder/sub-folder/*`
操作	所有操作

2. 随后添加**禁止**策略，配置信息如下：

配置项	配置值
效力	拒绝
用户	选择子账号，然后输入子账号的 UIN，该子账号必须为当前主账号下的子账号，例如100000000011
资源	指定需要禁止被访问的对象键，例如 `folder/sub-folder/privateobject`
操作	所有操作

案例五：授权子账号对指定前缀的文件的读写权限

配置信息如下：

配置项	配置值
效力	允许
用户	选择子账号，然后输入子账号的 UIN，该子账号必须为当前主账号下的子账号，例如100000000011
资源	指定前缀，例如 `folder/sub-folder/prefix`
操作	所有操作

通过访问管理（CAM）授权案例

用户如需针对账号维度进行授权，请参见以下文档进行配置：

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

授权子账号拥有特定目录下除指定文件之外的其他所有文件的读写权限

授权子账号对指定前缀的文件的读写权限

授权跨账号对指定文件的读写权限

COS API 授权策略使用指引

最近更新时间：2024-01-06 10:47:50

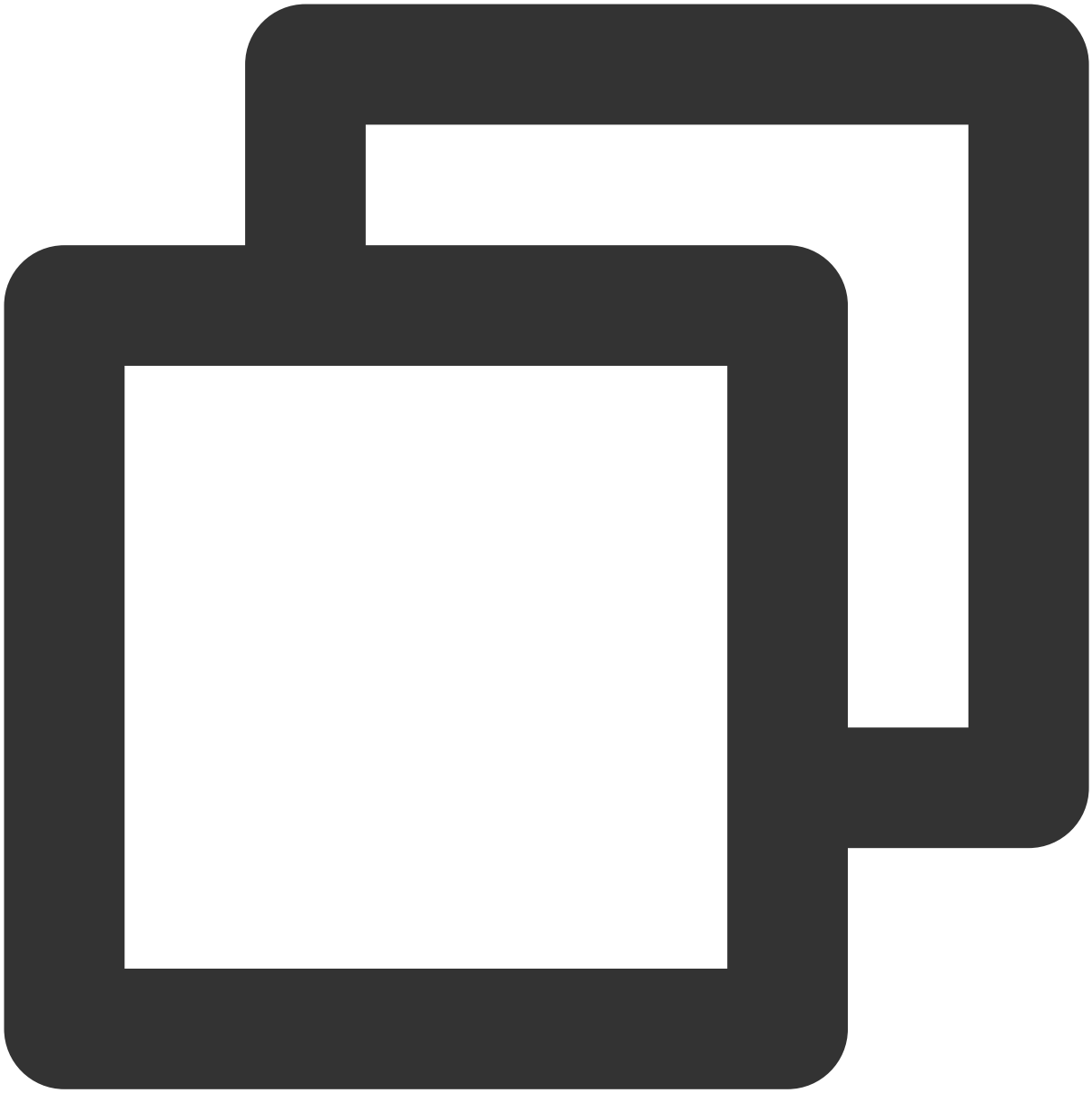
注意

在给子用户或协作者授予 API 操作权限时，请务必根据业务需要，按照最小权限原则进行授权。如果您直接授予子用户或者协作者所有资源 `(resource:*)` ，或所有操作 `(action:*)` 权限，则存在由于权限范围过大导致数据安全风险。

概述

对象存储（Cloud Object Storage，COS）使用临时密钥服务时，不同的 COS API 操作需要不同的操作权限，而且可以同时指定一个操作或一序列操作。

COS API 授权策略（policy）是一种 JSON 字符串。例如，授予 APPID 为1250000000，地域为 ap-beijing，存储桶为 examplebucket-1250000000 ，路径前缀为 doc 的上传操作（包括简单上传、表单上传、分块上传等操作）的权限，路径前缀为 doc2 的下载操作权限的策略内容如下所示：



```
{
    "version": "2.0",
    "statement": [{
            "action": [
                //简单上传操作
                "name/cos:PutObject",
                //表单上传对象
                "name/cos:PostObject",
                //分块上传：初始化分块操作
                "name/cos:InitiateMultipartUpload",
                //分块上传：List 进行中的分块上传
```

```
                "name/cos:ListMultipartUploads",
                //分块上传：List 已上传分块操作
                "name/cos:ListParts",
                //分块上传：上传分块块操作
                "name/cos:UploadPart",
                //分块上传：完成所有分块上传操作
                "name/cos:CompleteMultipartUpload",
                //取消分块上传操作
                "name/cos:AbortMultipartUpload"
            ],
            "effect": "allow",
            "resource": [
                "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
            ]
        },
        {
            "action": [
                //下载操作
                "name/cos:GetObject"
            ],
            "effect": "allow",
            "resource": [
                "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc2/*
            ]
        }
    ]
}
```

授权策略（policy）元素说明

名称	描述
version	策略语法版本，默认为2.0。
effect	有 allow （允许）和 deny （显式拒绝）两种情况。
resource	授权操作的具体数据，可以是任意资源、指定路径前缀的资源、指定绝对路径的资源或它们的组合。 **注意：** 若路径为中文，则保持中文输入即可。例如 `examplebucket-1250000000/文件夹/文件名.txt` 。
action	此处是指 COS API，根据需求指定一个或者一序列操作的组合或所有操作(`*`)，例如 action 为 `name/cos:GetService` ，**请注意区分英文大小写**。

condition	约束条件，可以不填，具体说明请参见 condition 说明。

下面列出了各 COS API 设置授权策略的示例。

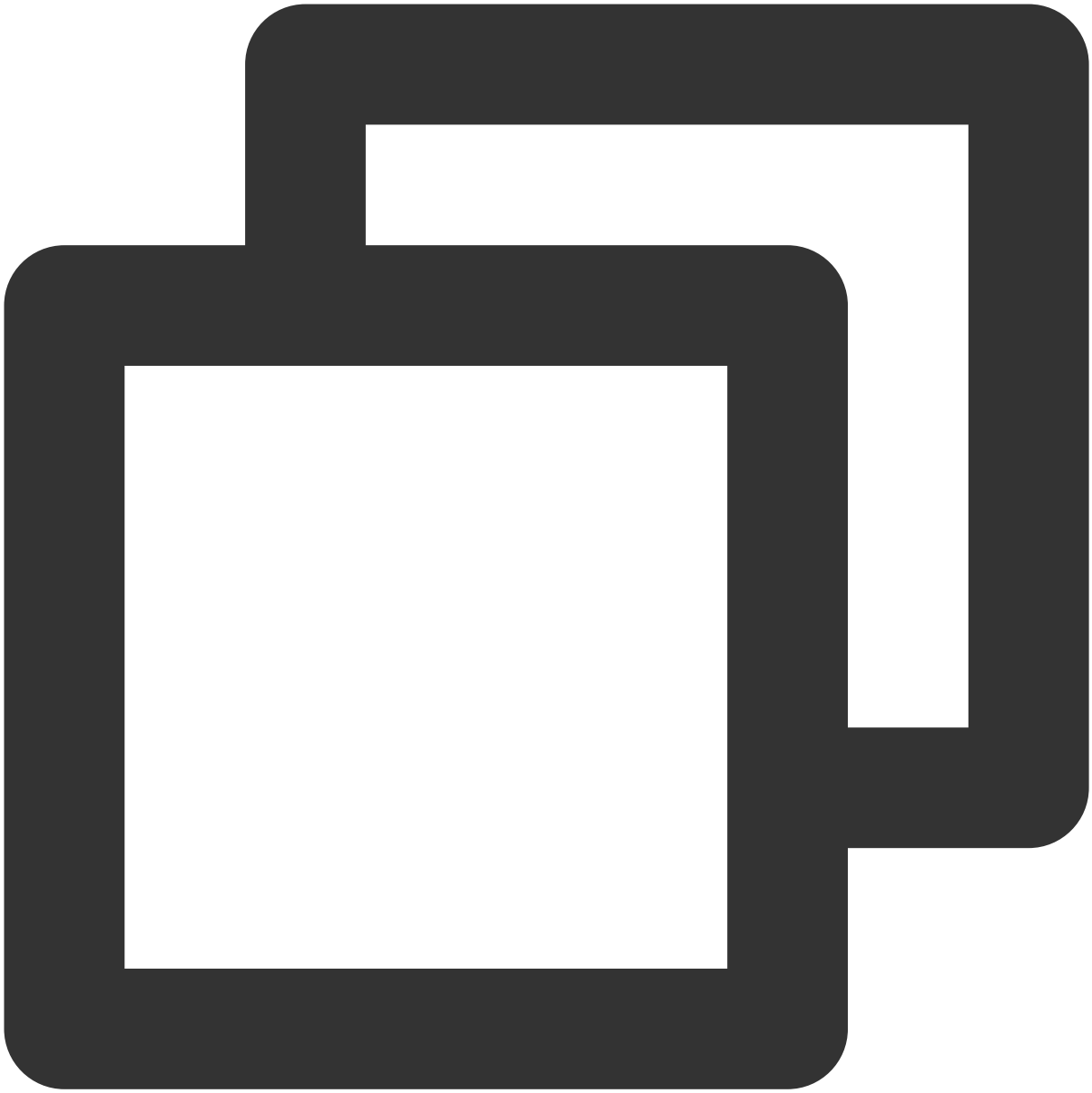
Service API

查询存储桶列表

API 接口为 GET Service，若授予其操作权限，则策略的 action 为 name/cos:GetService，resource为 `*` 。

示例

授予查询存储桶列表操作权限的策略详细内容如下：



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

```
      }
    ]
  }
```

Bucket API

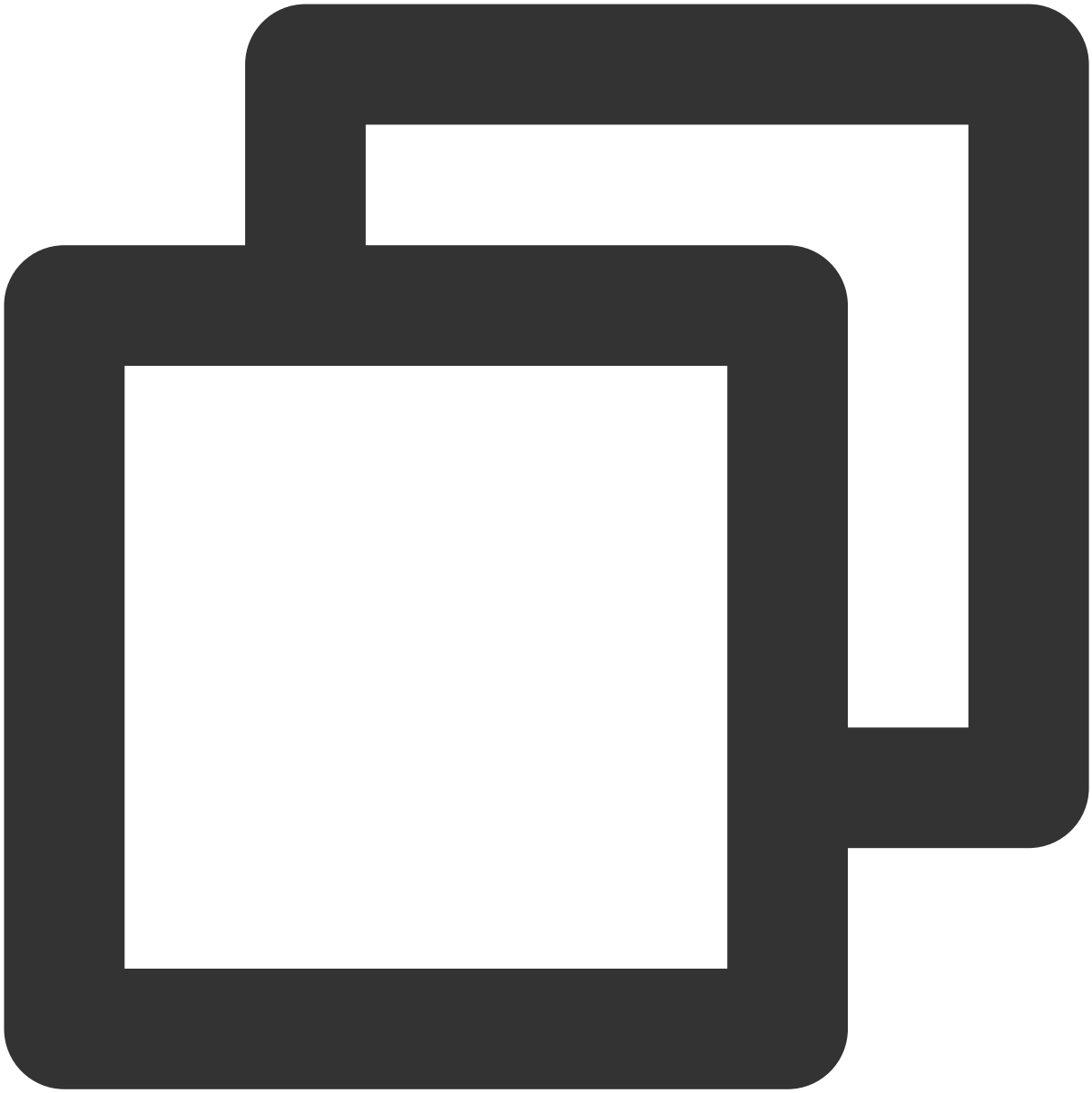
Bucket API 策略的 resource 可以归纳为以下几种情况：
操作全部地域的存储桶
则策略的 resource 为 `*`，**该策略限定的资源范围，存在由于权限范围过大导致数据安全风险，请谨慎配置**。
仅允许操作指定地域的存储桶
例如只允许操作 APPID 为1250000000，地域归属于北京（ap-beijing）的存储桶，则策略的 resource 为 `qcs::cos:ap-beijing:uid/1250000000:*` 。
仅允许操作指定地域且指定名称的存储桶
例如只可操作 APPID 为1250000000，地域为 ap-beijing 且名称为 examplebucket-1250000000 的存储桶，则策略的 resource 为 `qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/*` 。
Bucket API 策略的 action 则因操作不同而取值不同，以下列举部分 Bucket API 授权策略，其他 Bucket API 授权策略可作参照。

创建存储桶

API 接口为 PUT Bucket，若授予其操作权限，则策略的 action 为 name/cos:PutBucket。

示例

授予用户 APPID 为1250000000，创建存储桶的权限。例如创建一个地域为北京地域，存储桶名称为 examplebucket-1250000000 的存储桶，则策略详细内容如下：



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:PutBucket"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

```
        }
    ]
}
```

说明

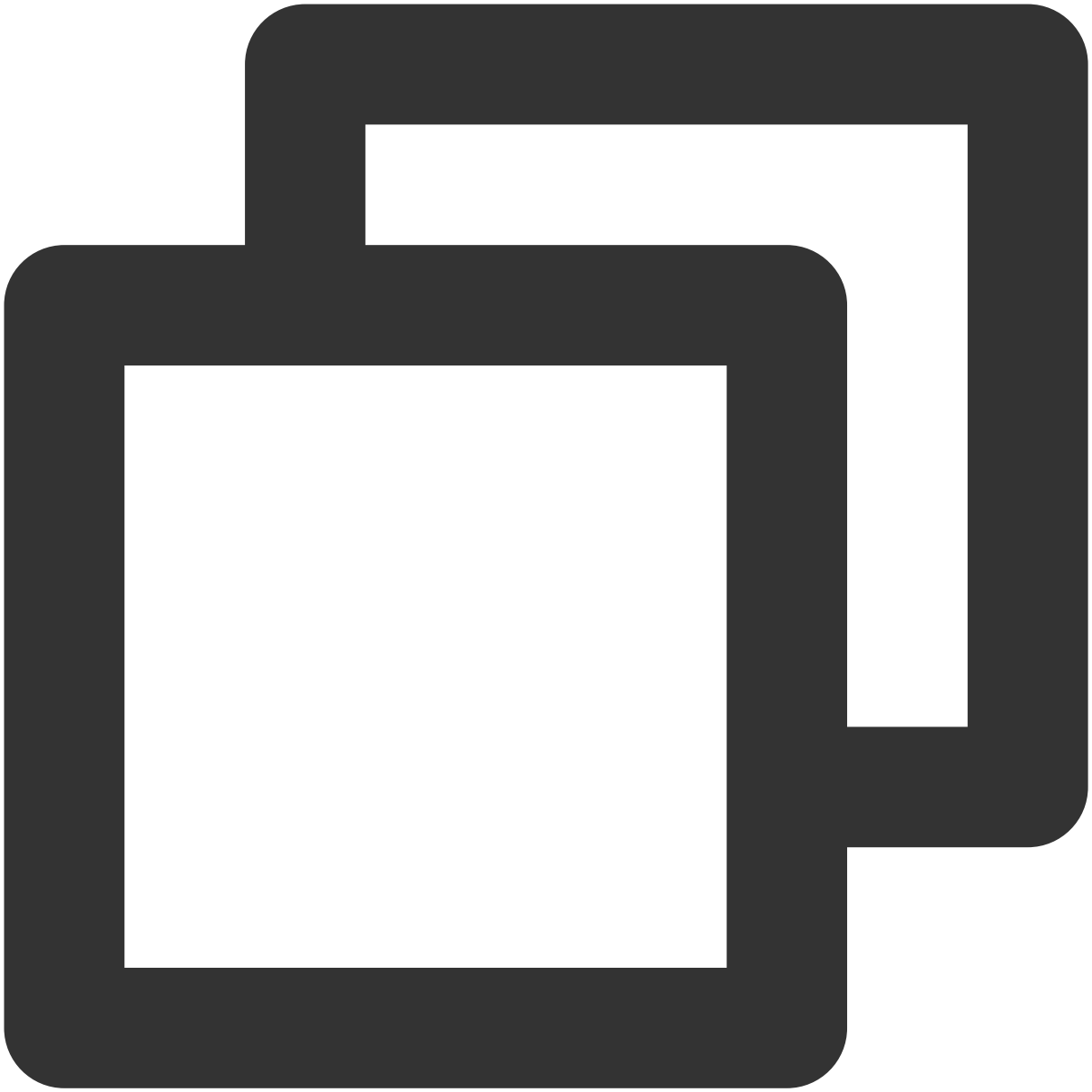
存储桶名称需符合命名规范，详情请参见 存储桶命名规范。

检索存储桶及其权限

API 接口为 HEAD Bucket，若授予其操作权限，则策略的 action 为 name/cos:HeadBucket。

示例

授予只能检索 APPID 为1250000000 ， 地域为 ap-beijing ， 存储桶为 examplebucket-1250000000 的操作权限，其策略详细内容如下：



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:HeadBucket"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

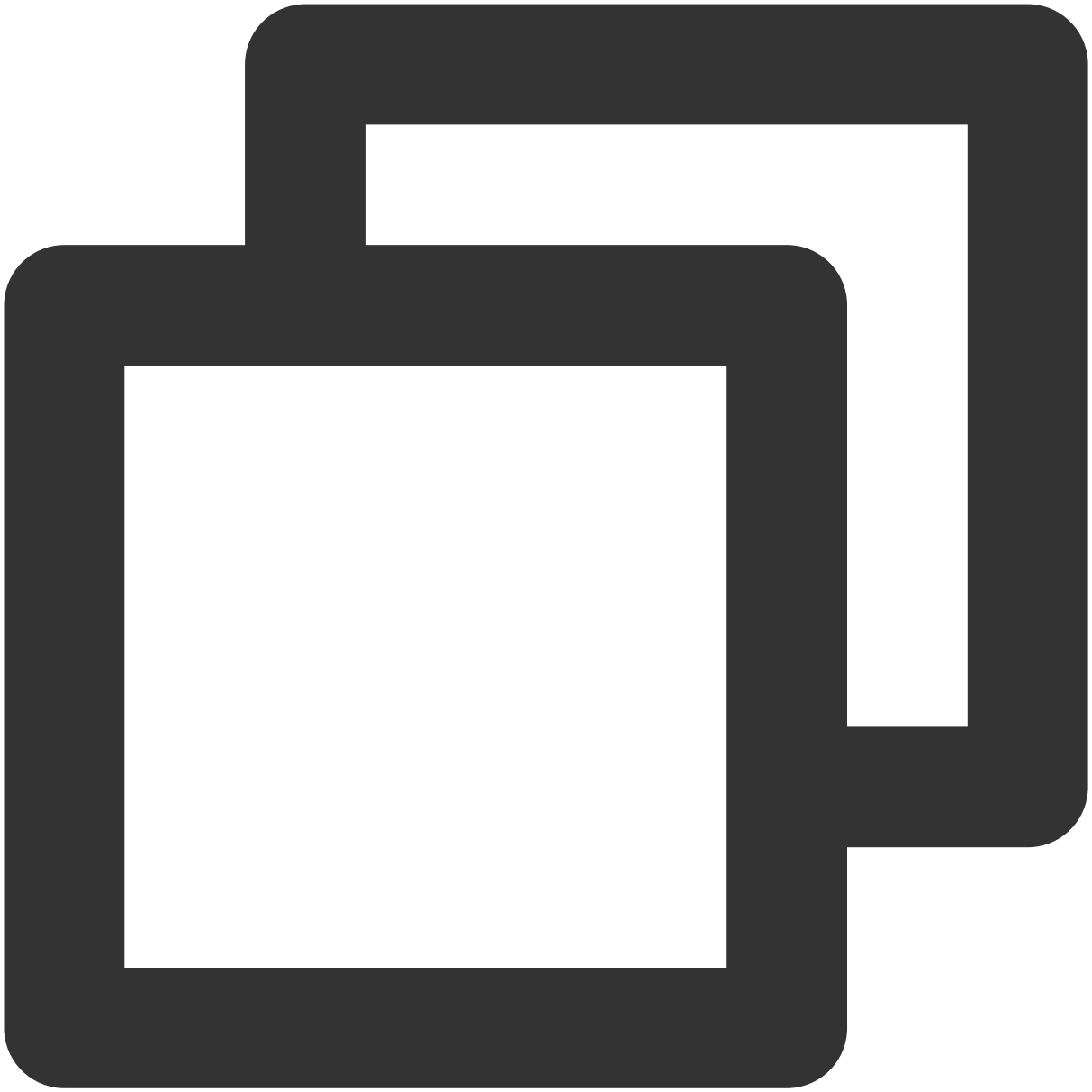
```
        }
    ]
}
```

查询对象列表

API 接口为 GET Bucket，若授予其操作权限，则策略的 action 为 name/cos:GetBucket。

示例

授予只能查询 APPID 为1250000000 ，地域为 ap-beijing，存储桶为 examplebucket-1250000000 的对象列表的操作权限，其策略详细内容如下：



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:GetBucket"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

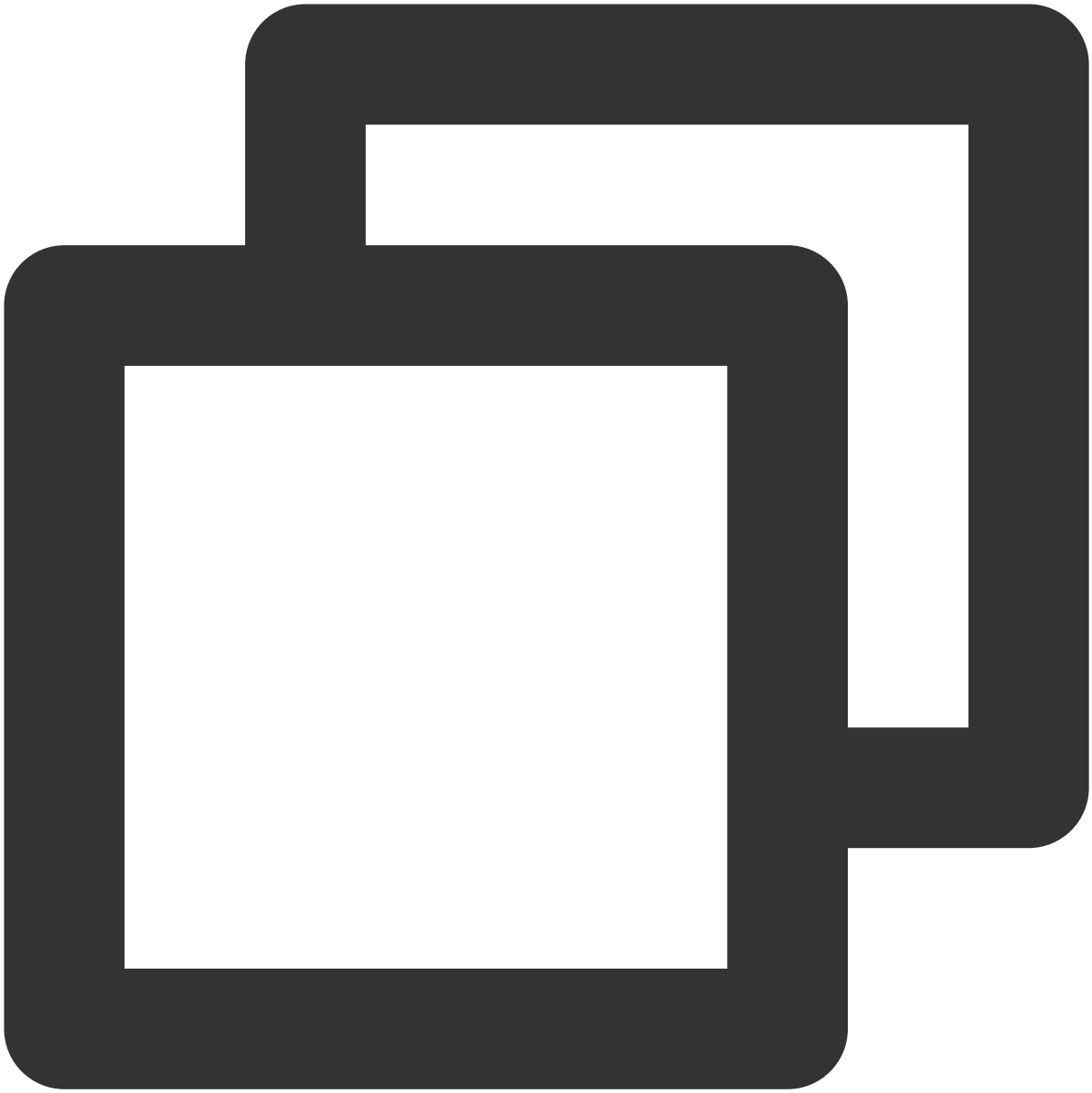
```
        }
    ]
}
```

删除存储桶

API 接口为 Delete Bucket，若授予其操作权限，则策略的 action 为 name/cos:DeleteBucket。

示例

授予只能删除 APPID 为1250000000 ，地域为 ap-beijing ，存储桶为 examplebucket-1250000000 的存储桶的操作权限，其策略详细内容如下：



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:DeleteBucket"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

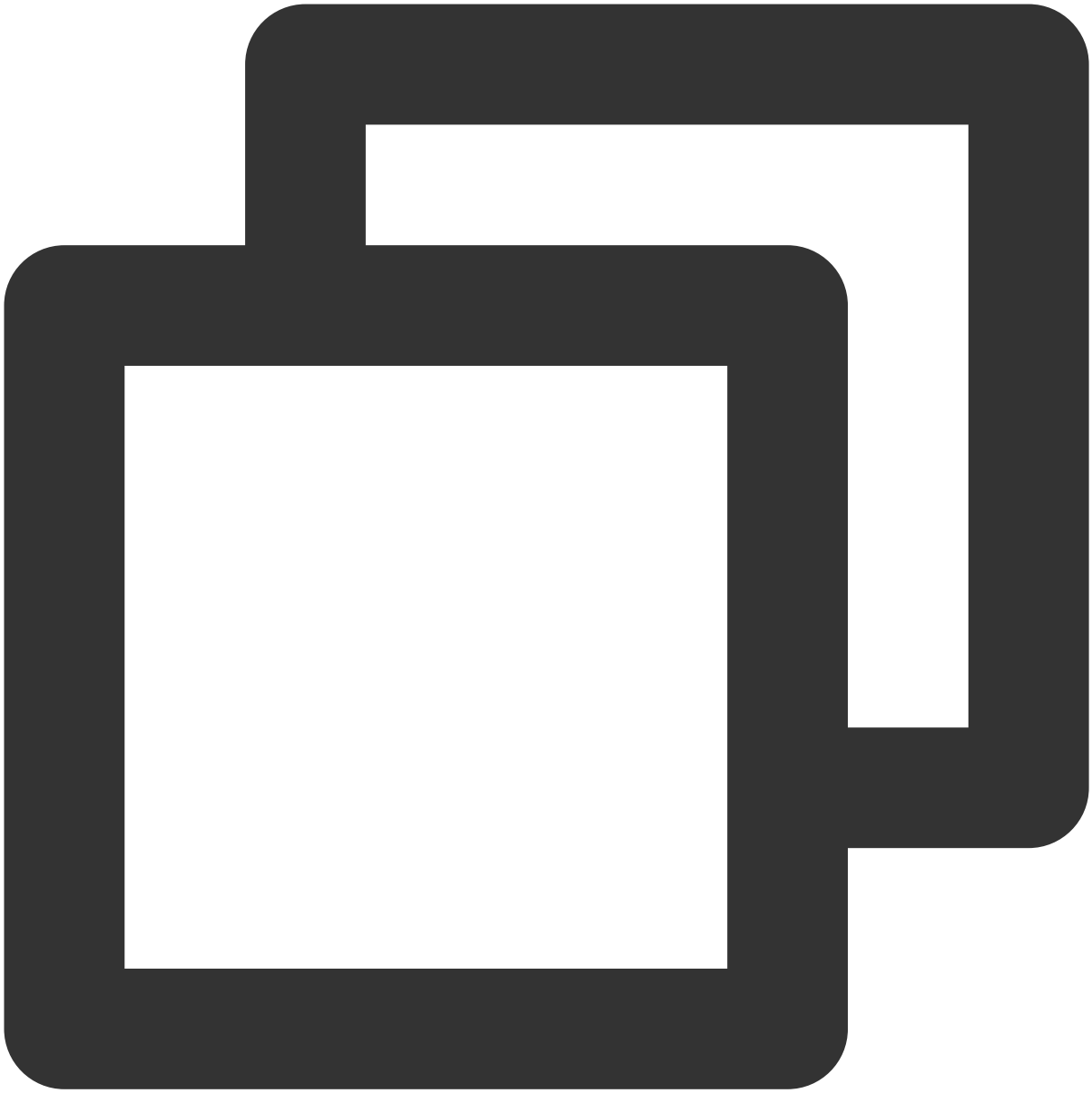
```
        }
    ]
}
```

设置存储桶 ACL

API 接口为 Put Bucket ACL，若授予其操作权限，则策略的 action 为 name/cos:PutBucketACL。

示例

授予只能设置 APPID 为1250000000 ， 地域为 ap-beijing ， 存储桶为 examplebucket-1250000000 的 ACL 的操作权限，其策略详细内容如下：



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:PutBucketACL"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

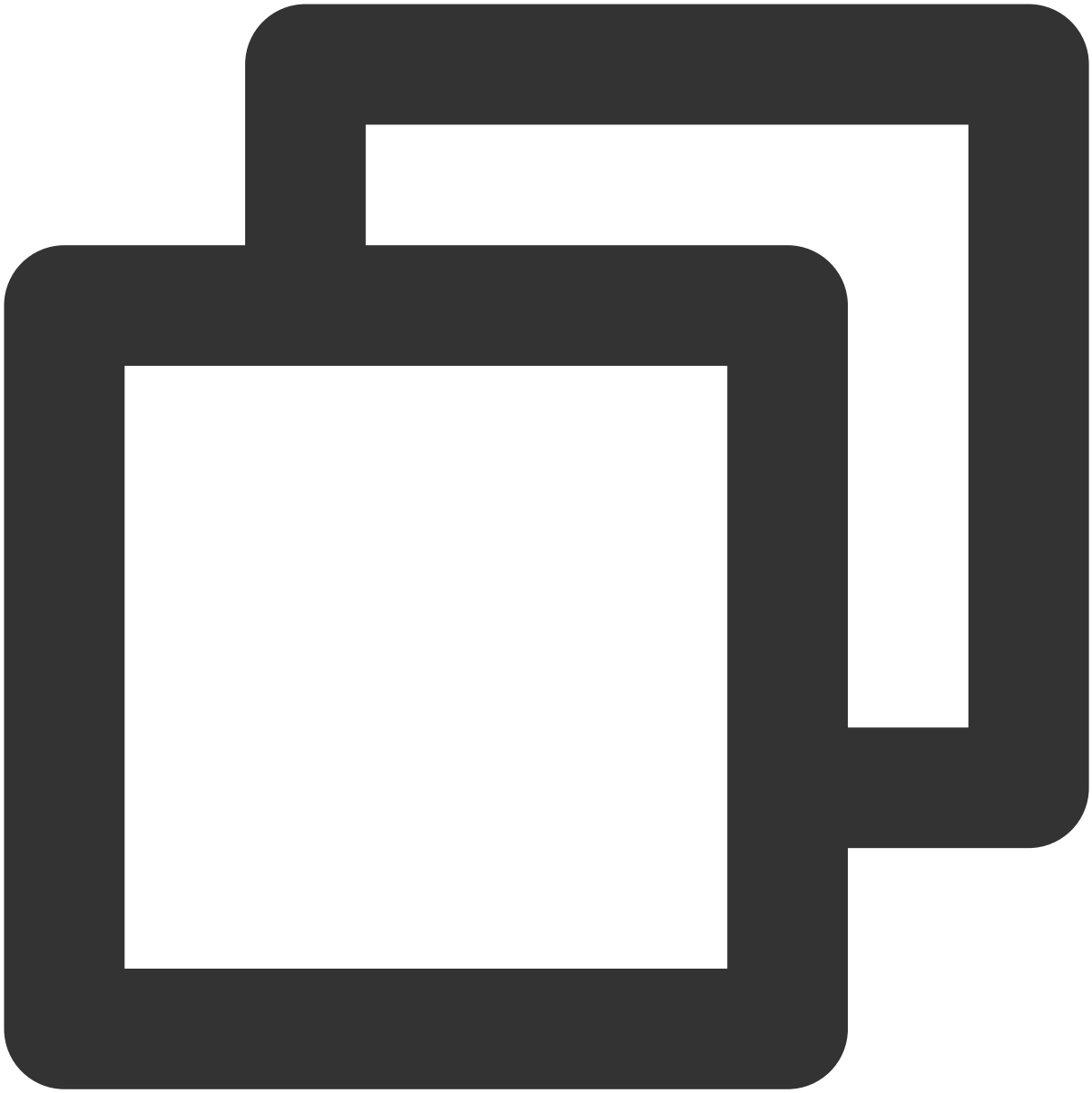
```
        }
    ]
}
```

查询存储桶 ACL

API 接口为 GET Bucket acl，若授予其操作权限，则策略的 action 为 name/cos:GetBucketACL。

示例

授予只能获取 APPID 为1250000000 ，地域为 ap-beijing ，存储桶为 examplebucket-1250000000 的 ACL 的操作权限，其策略详细内容如下：



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:GetBucketACL"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

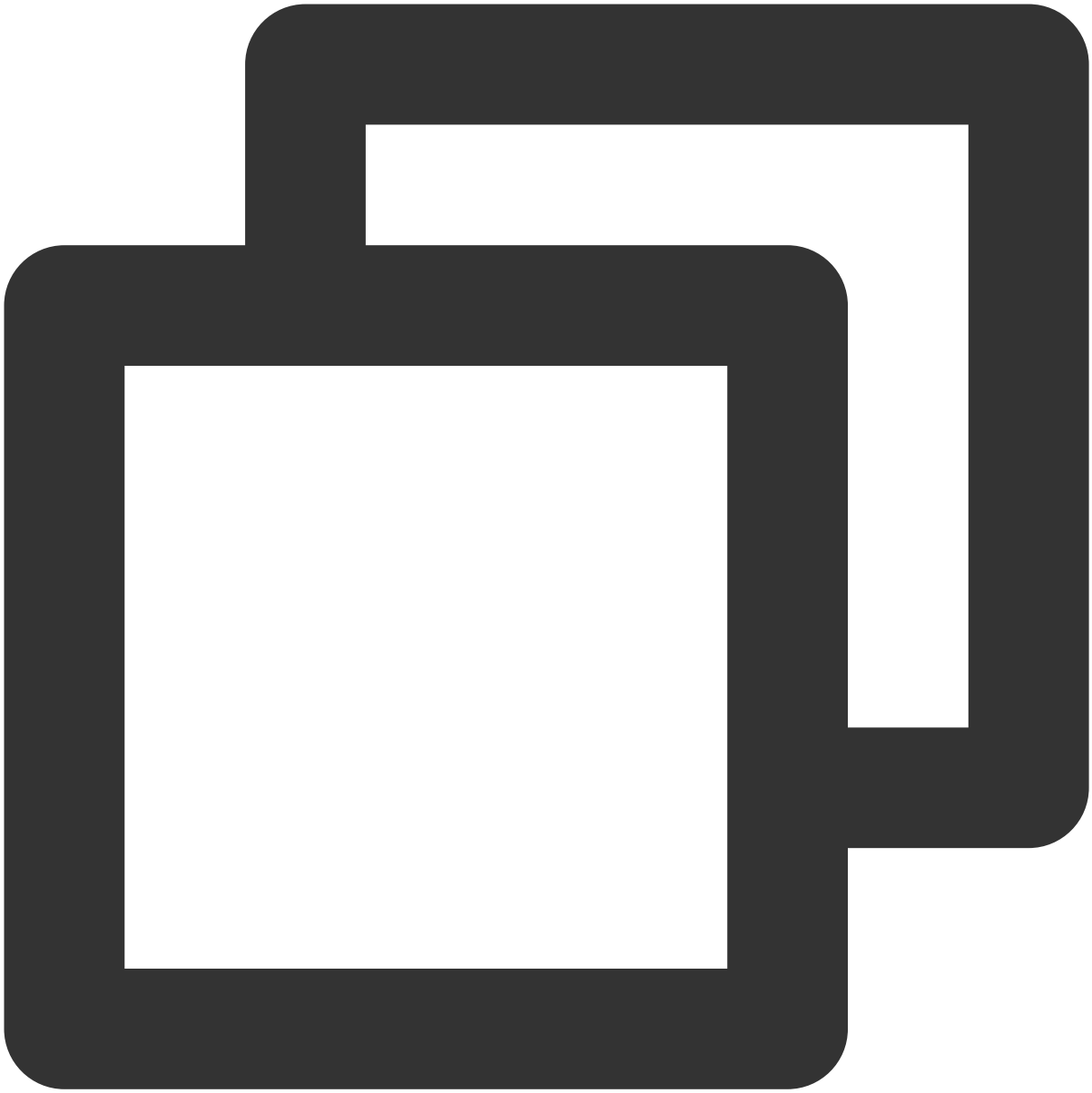
```
        }
    ]
}
```

设置跨域配置

API 接口为 PUT Bucket cors，若授予其操作权限，则策略的 action 为 name/cos:PutBucketCORS。

示例

授予只能设置 APPID 为1250000000 ， 地域为 ap-beijing ， 存储桶为 examplebucket-1250000000 的跨域配置的操作权限，其策略详细内容如下：



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:PutBucketCORS"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

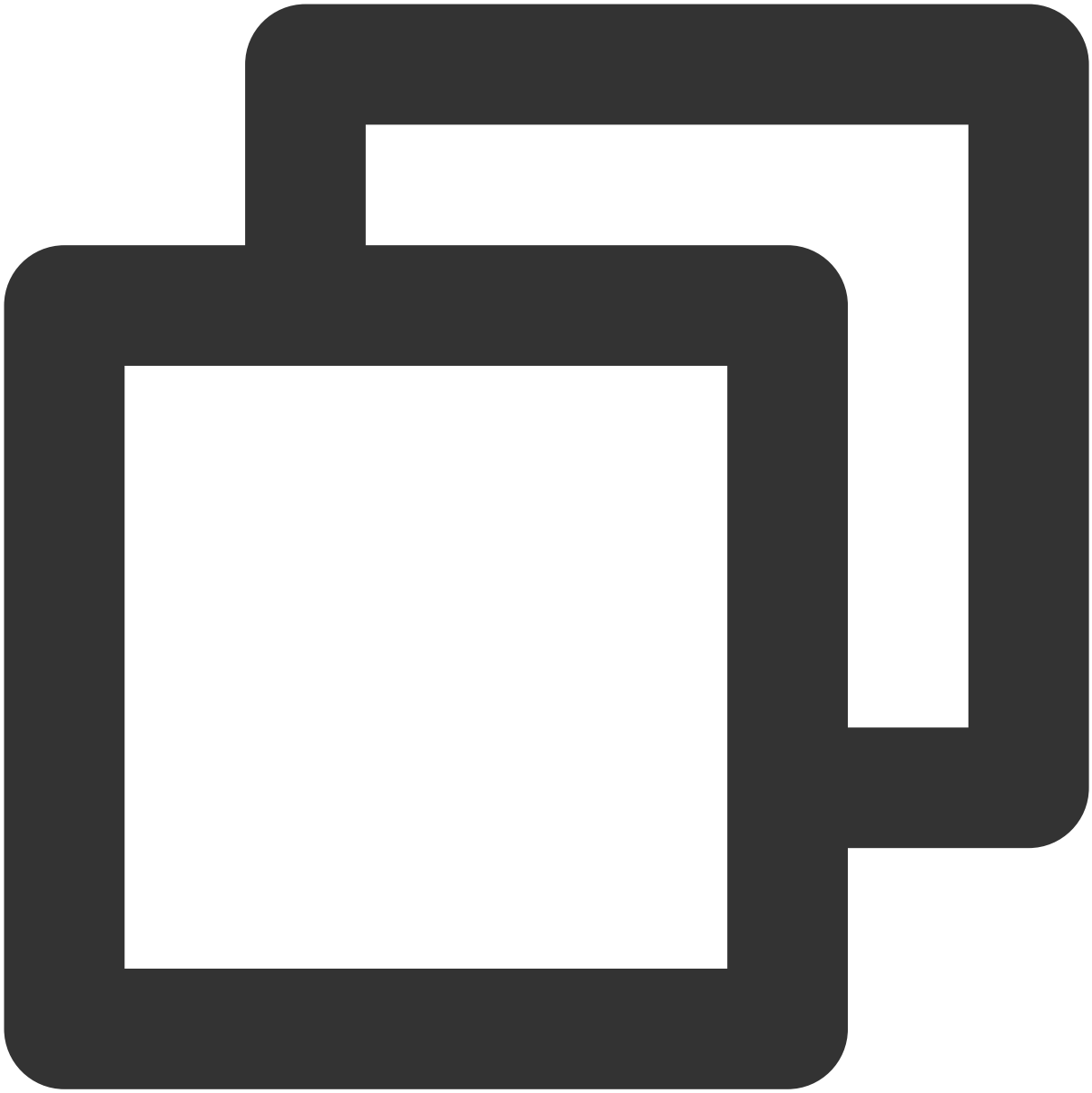
```
        }
    ]
}
```

查询跨域配置

API 接口为 GET Bucket cors，若授予其权限，则策略的 action 为 name/cos:GetBucketCORS。

示例

授予只能查询 APPID 为1250000000 ，地域为 ap-beijing ，存储桶为 examplebucket-1250000000 的跨域配置的操作权限，其策略详细内容如下：



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:GetBucketCORS"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

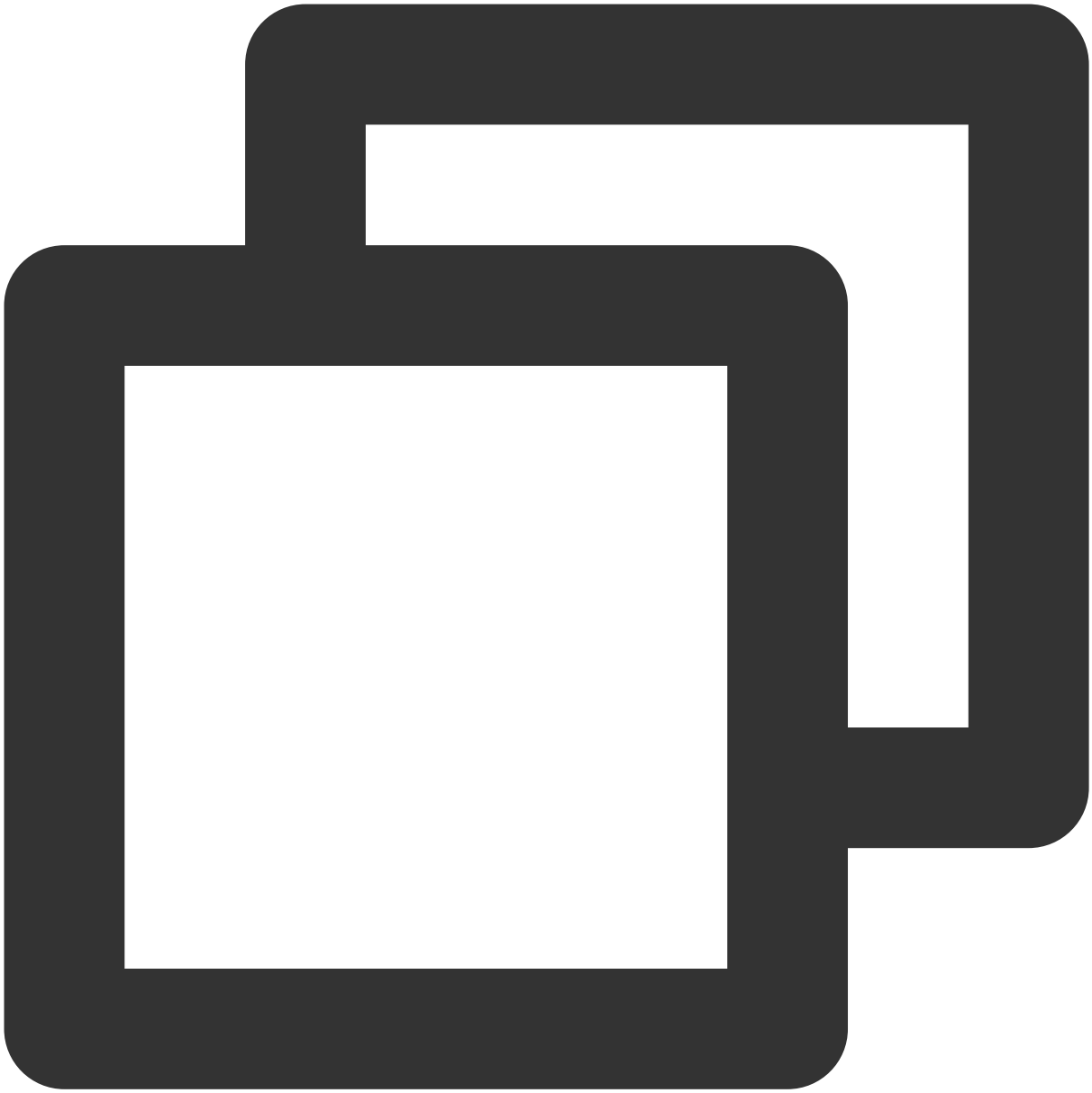
```
        }
    ]
}
```

删除跨域配置

API 接口为 DELETE Bucket cors，若授予其操作权限，则策略的 action 为 name/cos:DeleteBucketCORS。

示例

授予只能删除 APPID 为1250000000 ，地域为 ap-beijing ，存储桶为 examplebucket-1250000000 的跨域配置的操作权限，其策略详细内容如下：



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:DeleteBucketCORS"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

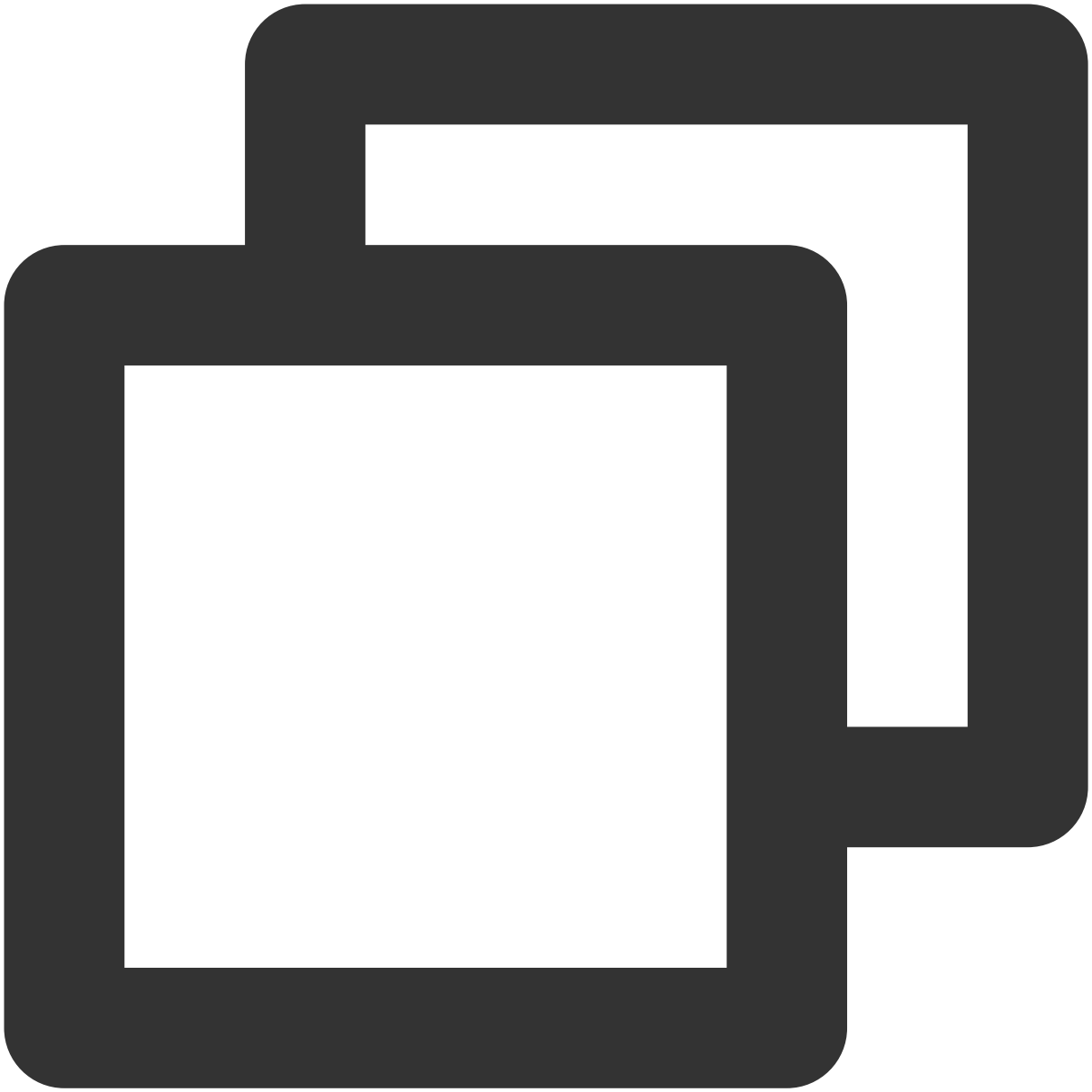
```
        }
    ]
}
```

设置生命周期

API 接口为 PUT Bucket lifecycle，若授予其操作权限，则策略的 action 为 name/cos:PutBucketLifecycle 。

示例

授予只能设置 APPID 为1250000000 ， 地域为 ap-beijing ， 存储桶为 examplebucket-1250000000 的生命周期配置的操作权限，其策略详细内容如下：



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:PutBucketLifecycle"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

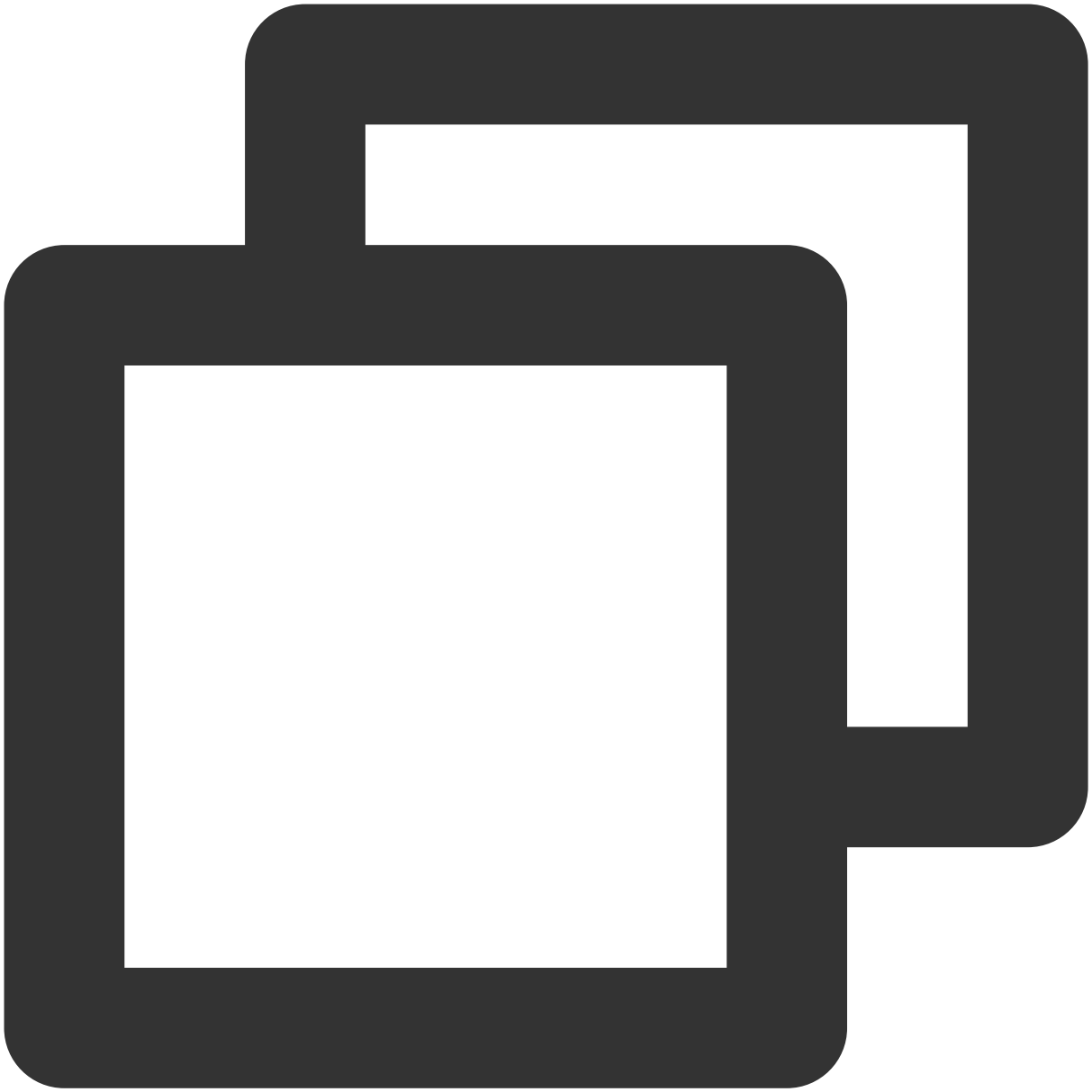
```
        }
    ]
}
```

查询生命周期

API 接口为 GET Bucket lifecycle，若授予其操作权限，则策略的 action 为 name/cos:GetBucketLifecycle。

示例

授予只能查询 APPID 为1250000000 ，地域为 ap-beijing ，存储桶为 examplebucket-1250000000 的生命周期配置的操作权限，其策略详细内容如下：



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:GetBucketLifecycle"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

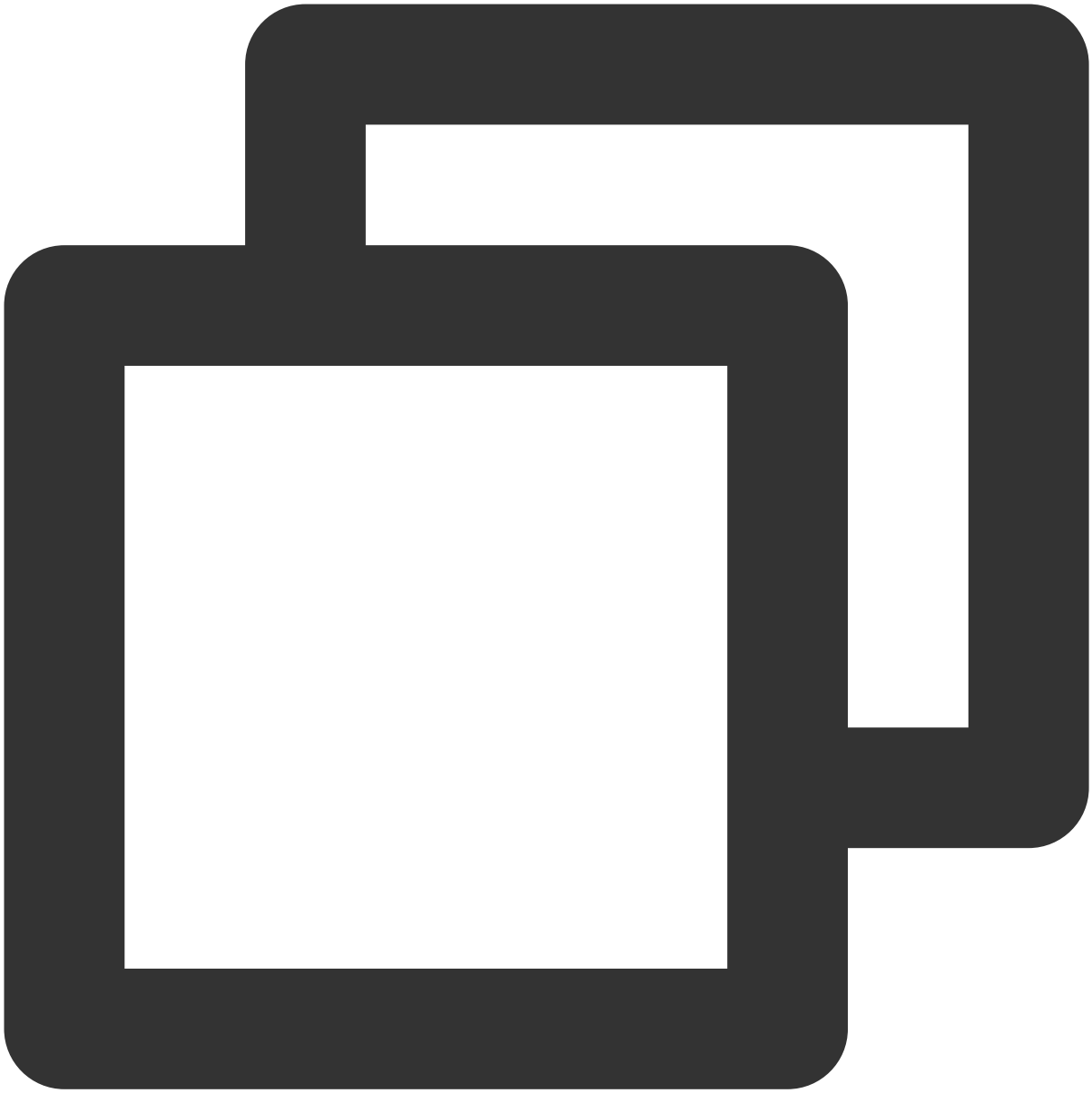
```
        }
    ]
}
```

删除生命周期

API 接口为 DELETE Bucket lifecycle，若授予其操作权限，则策略的 action 为 name/cos:DeleteBucketLifecycle。

示例

授予只能删除 APPID 为1250000000 ，地域为 ap-beijing，存储桶为 examplebucket-1250000000 的生命周期配置的操作权限，其策略详细内容如下：



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:DeleteBucketLifecycle"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

```
      }
    ]
  }
```

Object API

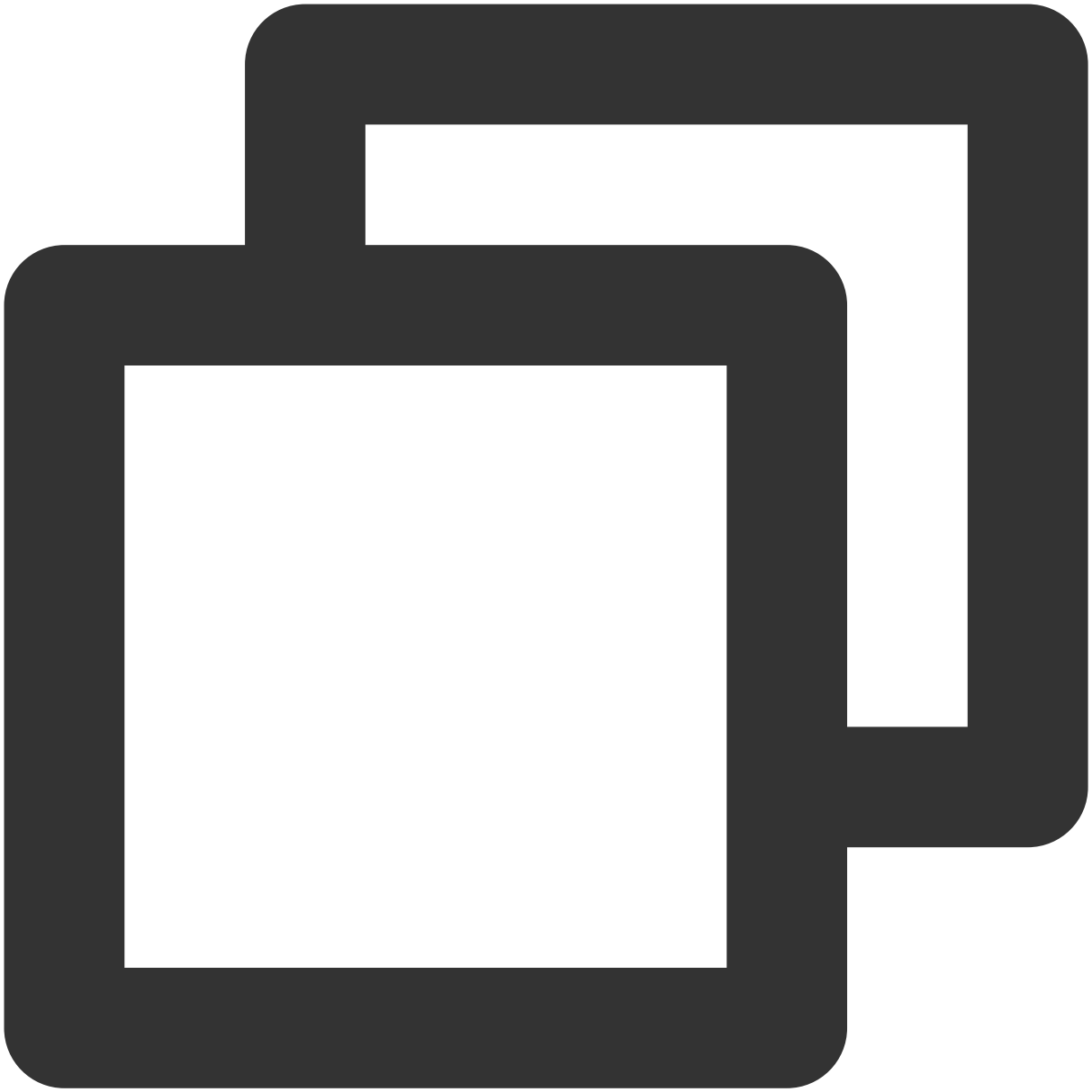
Object API 策略的 resource 可以归纳为以下几种情况：
可操作任意对象，策略的 resource 为 `*` 。
只可操作指定存储桶中的任意对象，如只可操作 APPID 为1250000000 ， 地域为 ap-beijing，且名称为 examplebucket-1250000000 的存储桶中的任意对象，则策略的 resource 为 `qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/*` 。
只可操作指定存储桶 且 指定路径前缀下的任意对象，如只可操作 APPID 为1250000000 ， 地域为 ap-beijing，存储桶为 examplebucket-1250000000，路径前缀为 doc 下的任意对象，则策略的 resource 为 `qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*` 。
只可操作指定绝对路径的对象，如只可操作 APPID 为1250000000 ， 地域为 ap-beijing，存储桶为 examplebucket-1250000000，绝对路径为 `doc/audio.mp3` 的对象，则策略的 resource 为 `qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/audio.mp3` 。
Object API 策略的 action 则因操作不同而取值不同，以下列举所有 Object API 授权策略。

简单上传对象

API 接口为 PUT Object，若授予其操作权限，则策略的 action为 name/cos:PutObject。

示例

授予只能在 APPID 为1250000000 ， 地域为 ap-beijing，存储桶为 examplebucket-1250000000，路径前缀为 doc 下进行简单上传的操作权限，其策略详细内容如下：



```
{
 "version": "2.0",
 "statement": [
   {
     "action": [
       "name/cos:PutObject"
     ],
     "effect": "allow",
     "resource": [
       "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
     ]
```

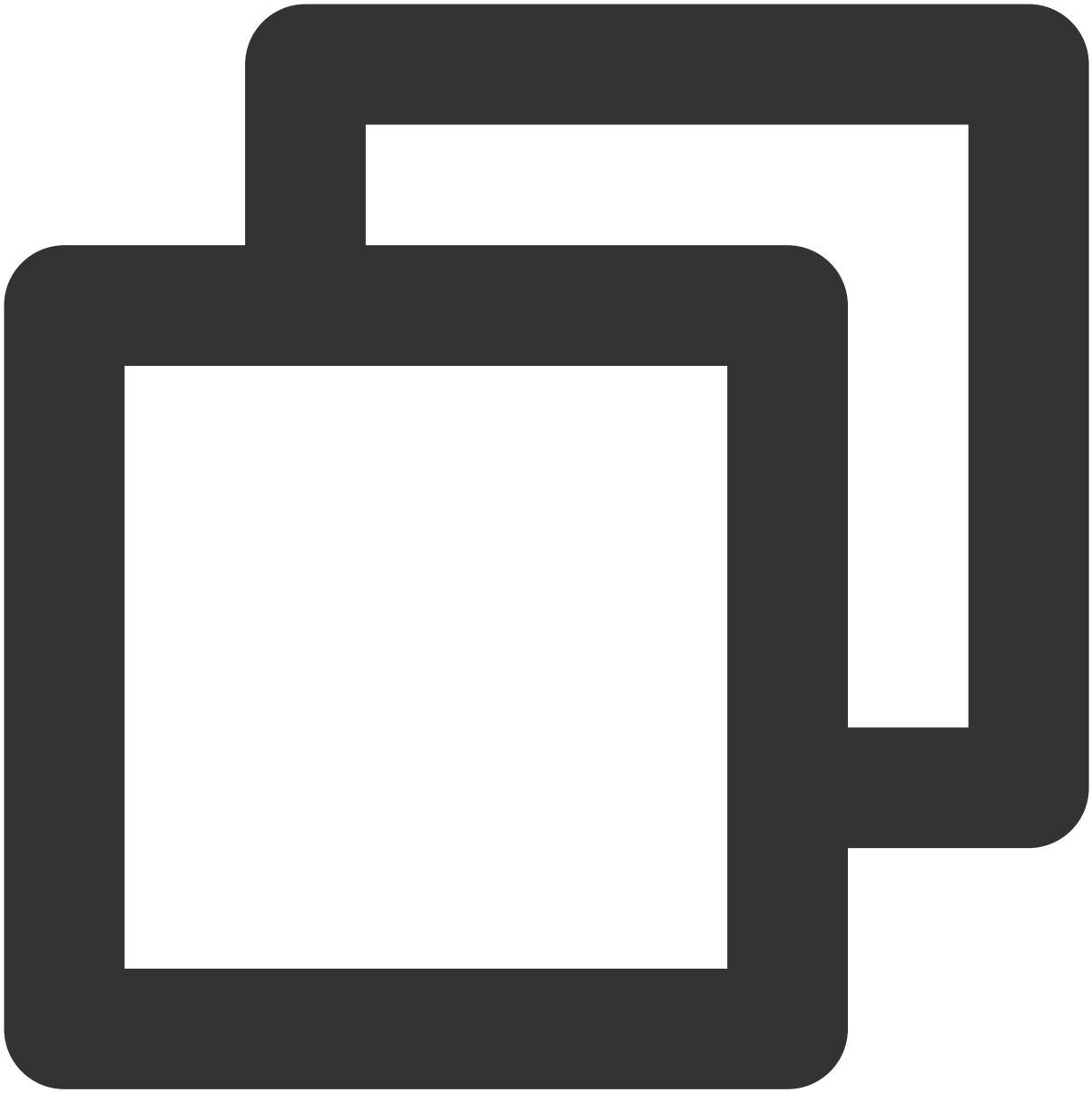
```
      }
    ]
  }
```

分块上传

分块上传包含 Initiate Multipart Upload，List Multipart Uploads，List Parts，Upload Part，Complete Multipart Upload，Abort Multipart Upload。若授予其操作权限，则策略的 action 为：`"name/cos:InitiateMultipartUpload","name/cos:ListMultipartUploads","name/cos:ListParts","name/cos:UploadPart","name/cos:CompleteMultipartUpload","name/cos:AbortMultipartUpload"` 的集合。

示例

授予只能在 APPID 为1250000000 ，地域为 ap-beijing ，存储桶为 examplebucket-1250000000，路径前缀为 doc 下进行分块上传的操作权限，其策略详细内容如下：



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:InitiateMultipartUpload",
        "name/cos:ListMultipartUploads",
        "name/cos:ListParts",
        "name/cos:UploadPart",
        "name/cos:CompleteMultipartUpload",
        "name/cos:AbortMultipartUpload"
```

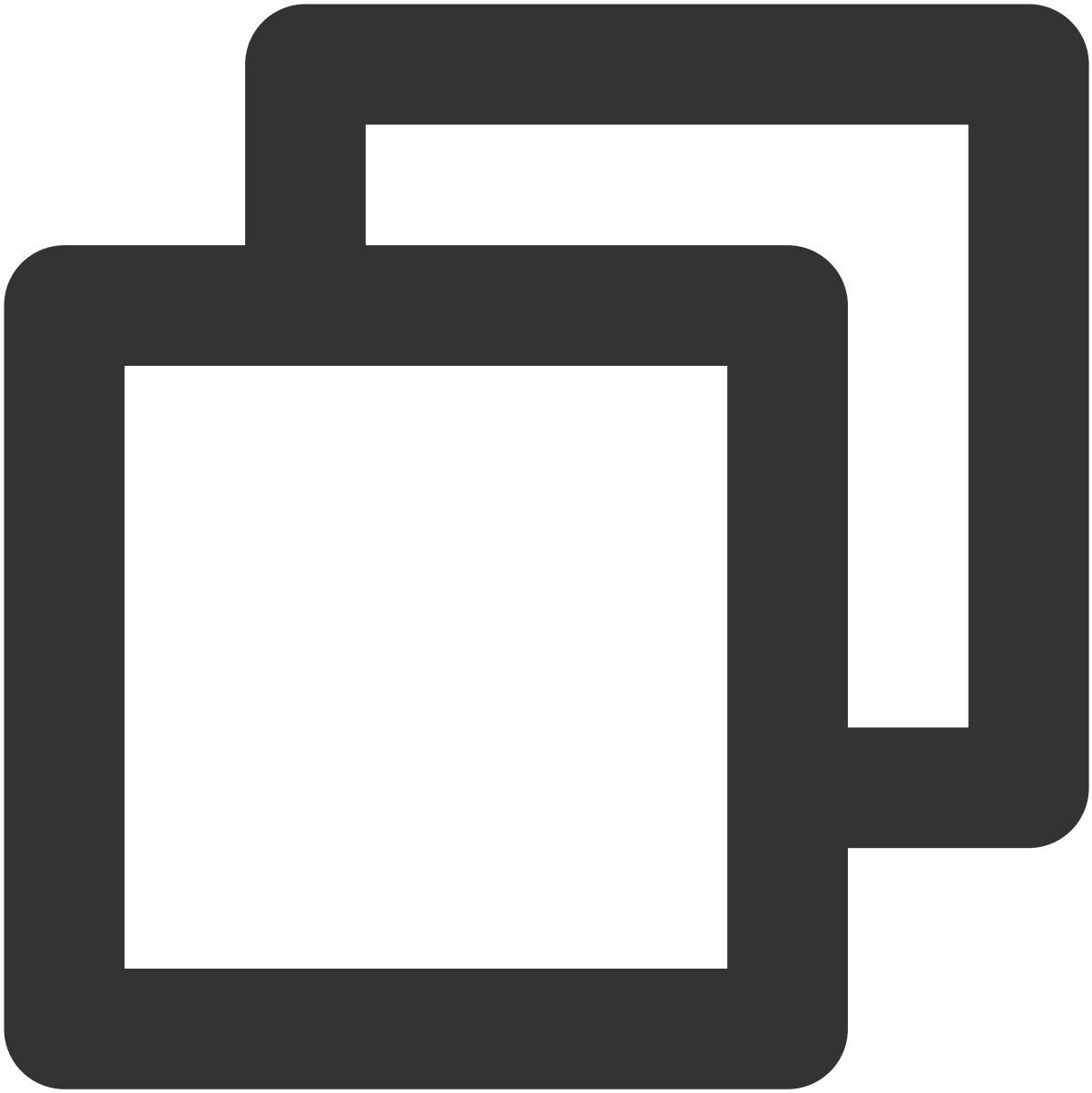
```
        ],
        "effect": "allow",
        "resource": [
          "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
        ]
      }
    ]
}
```

查询分块上传

查询存储桶中正在分块上传信息，若授予其操作权限，则策略的 action 为 name/cos:ListMultipartUploads。

示例

授予只能查询 APPID 为1250000000 ， 地域为 ap-beijing ， 存储桶为 examplebucket-1250000000 中的正在分块上传信息的操作权限，其策略详细内容如下：



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:ListMultipartUploads"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
      ]
```

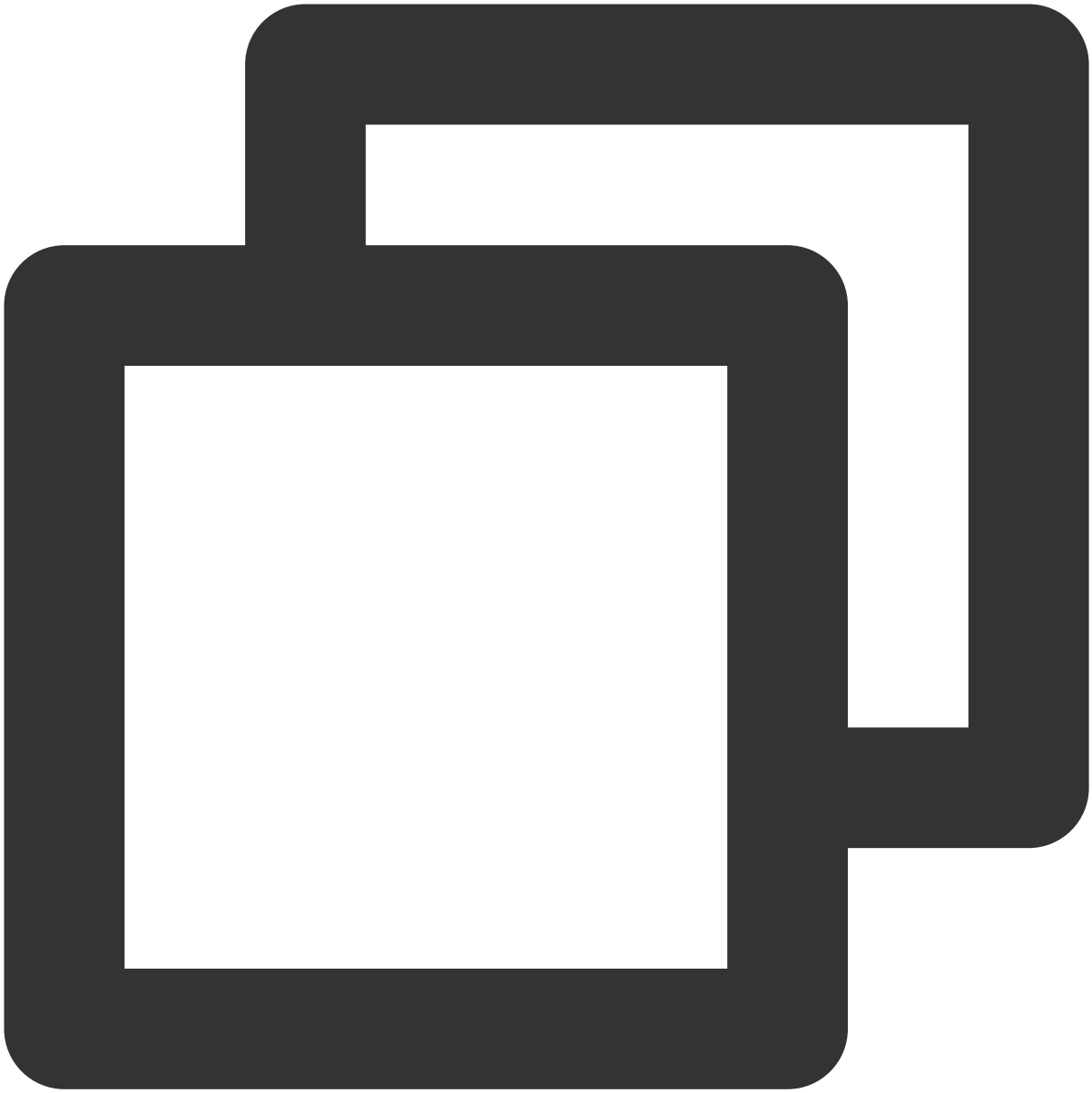
```
        }
    ]
}
```

表单上传对象

API 接口为 POST Object，若授予其操作权限，则策略的 action 为 name/cos:PostObject。

示例

授予只能在 APPID 为1250000000 ，地域为 ap-beijing ，存储桶为 examplebucket-1250000000，路径前缀为 doc 下进行 POST 上传的操作权限，其策略详细内容如下：



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:PostObject"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
```

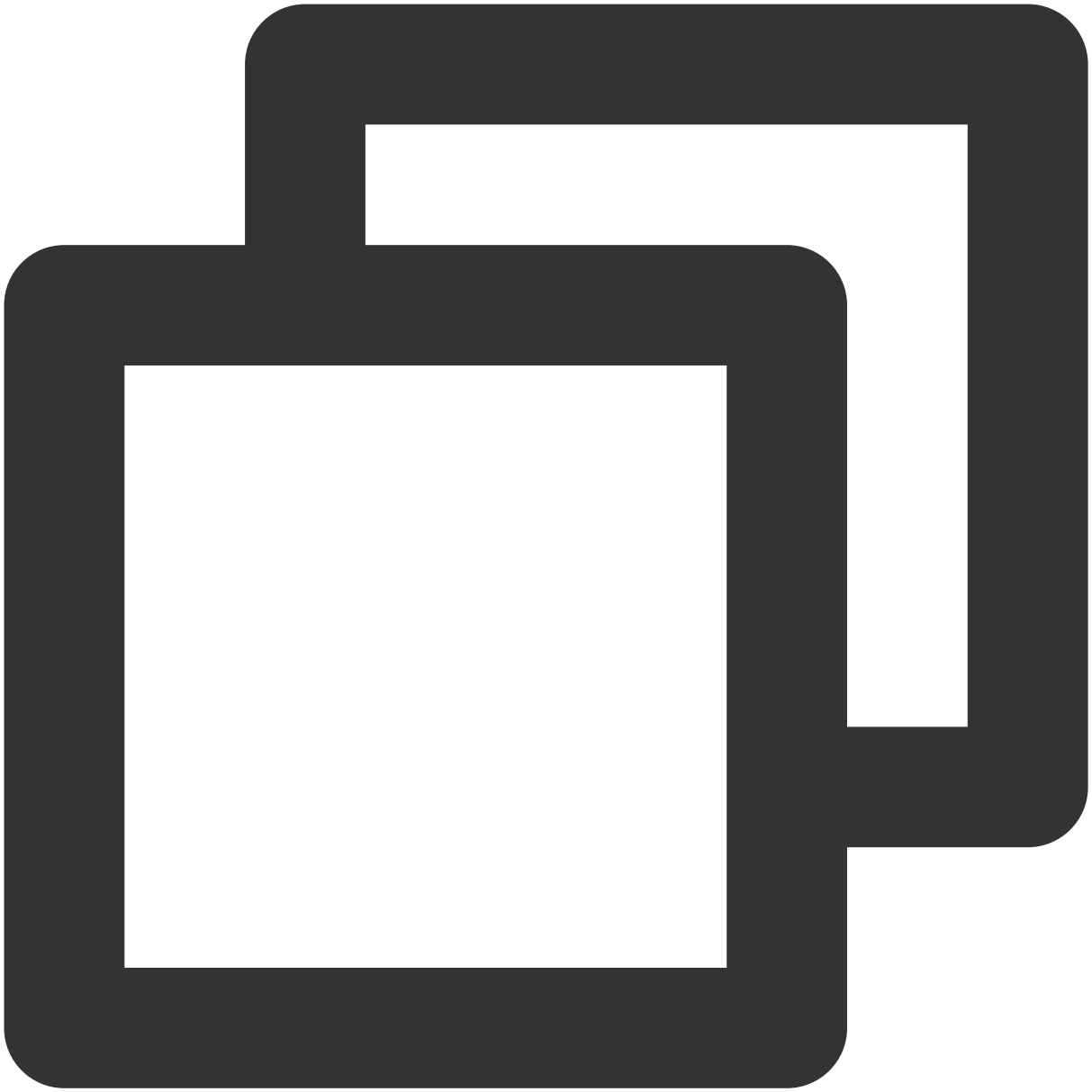
```
        }
    ]
}
```

追加上传对象

API 接口为 Append Object，若授予其操作权限，则策略的 action为 name/cos:AppendObject。

示例

授予只能在 APPID 为1250000000 ，地域为 ap-beijing，存储桶为 examplebucket-1250000000，路径前缀为 doc 下进行追加上传的操作权限，其策略详细内容如下：



```
{
 "version": "2.0",
 "statement": [
   {
     "action": [
       "name/cos:AppendObject"
     ],
     "effect": "allow",
     "resource": [
       "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
     ]
```

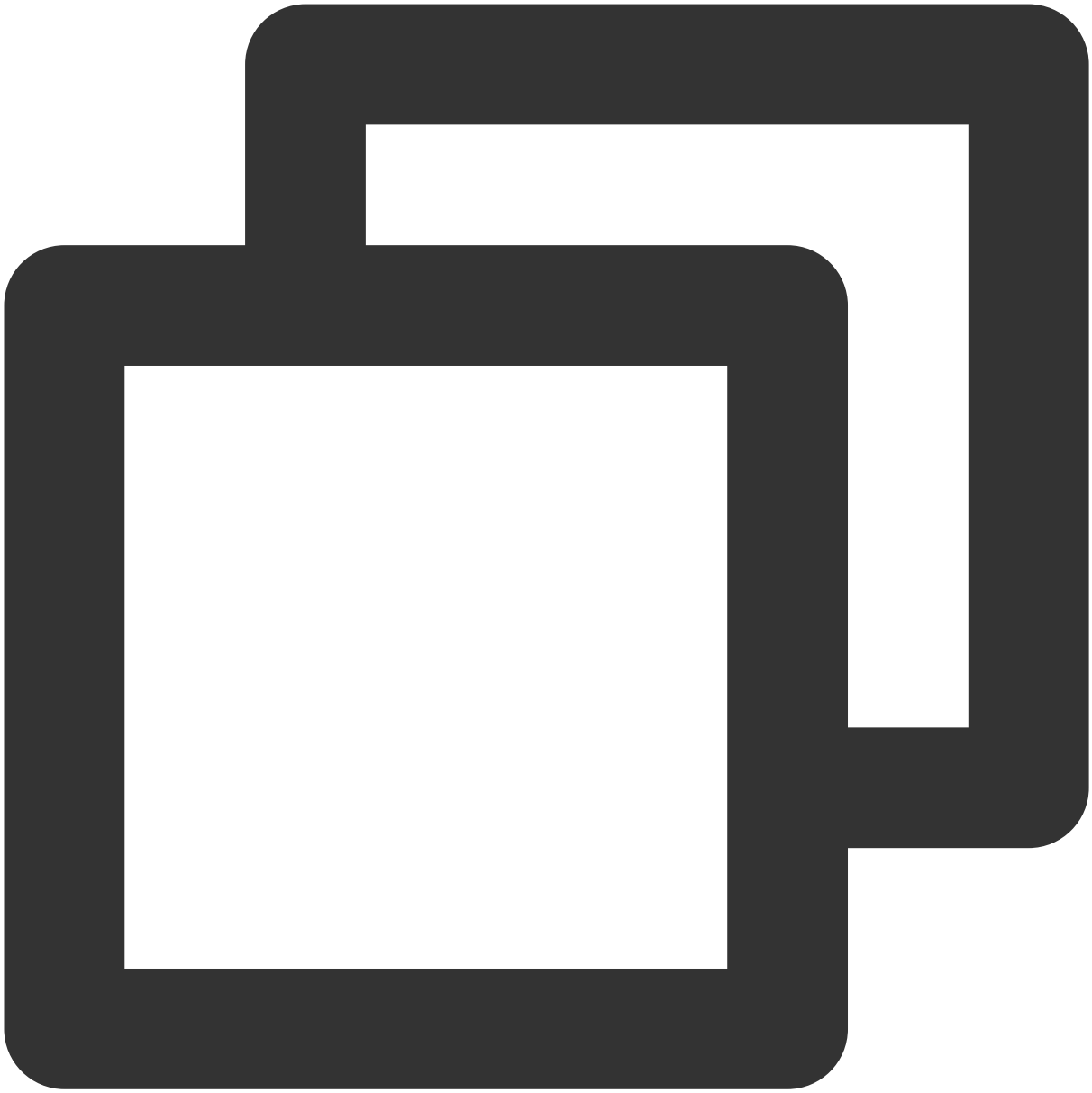
```
        }
    ]
}
```

查询对象元数据

API 接口为 HEAD Object，若授予其操作权限，则策略的 action 为 name/cos:HeadObject。

示例

授予只能查询 APPID 为1250000000 ，地域为 ap-beijing ，存储桶为 examplebucket-1250000000，路径前缀为 doc 中的对象的操作权限，其策略详细内容如下：



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:HeadObject"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
```

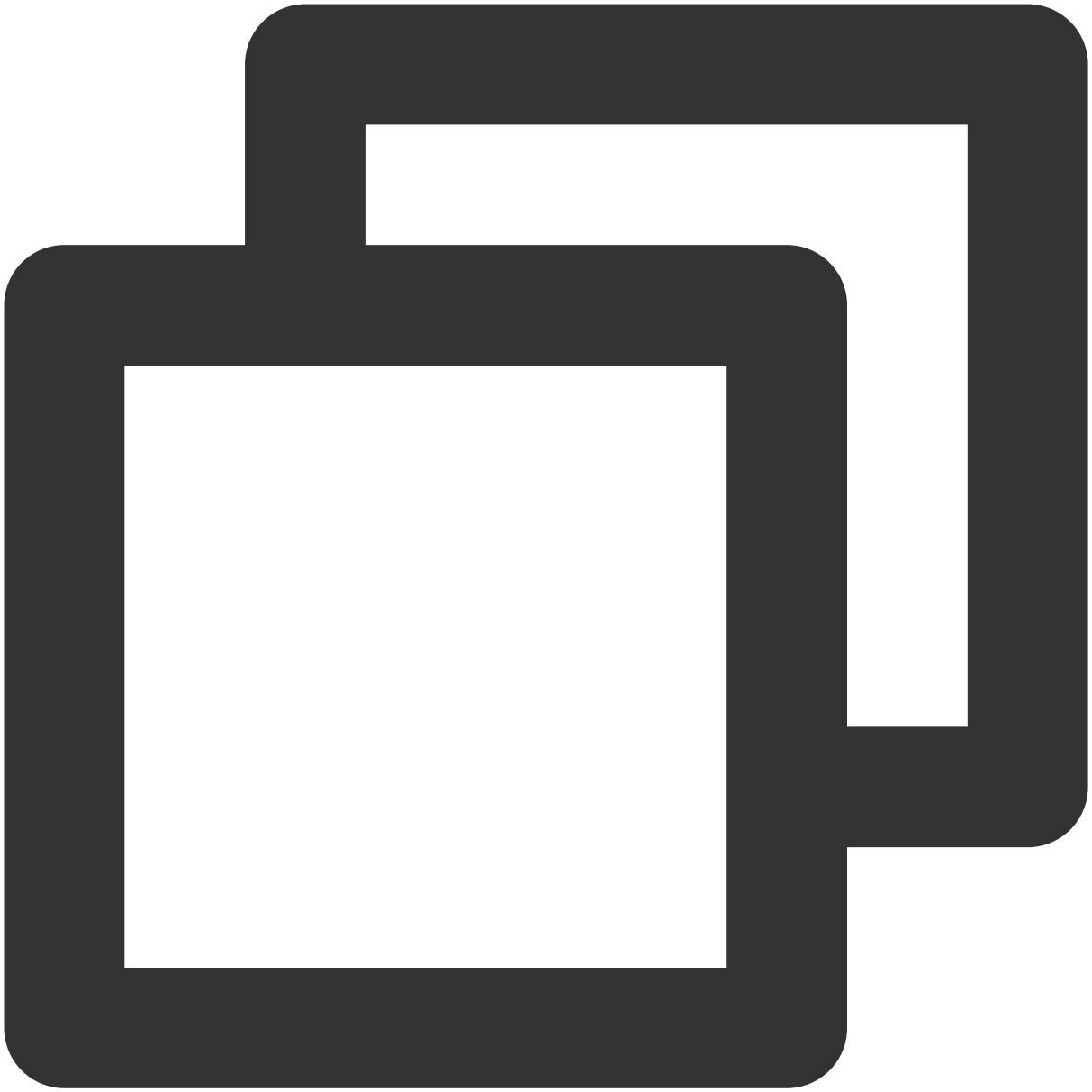
```
        }
    ]
}
```

下载对象

API 接口为 GET Object，若授予其操作权限，则策略的 action 为 name/cos:GetObject。

示例

授予只能下载 APPID 为1250000000 ，地域为 ap-beijing，存储桶为 examplebucket-1250000000，路径前缀为 doc 中的对象的操作权限，其策略详细内容如下：



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:GetObject"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
```

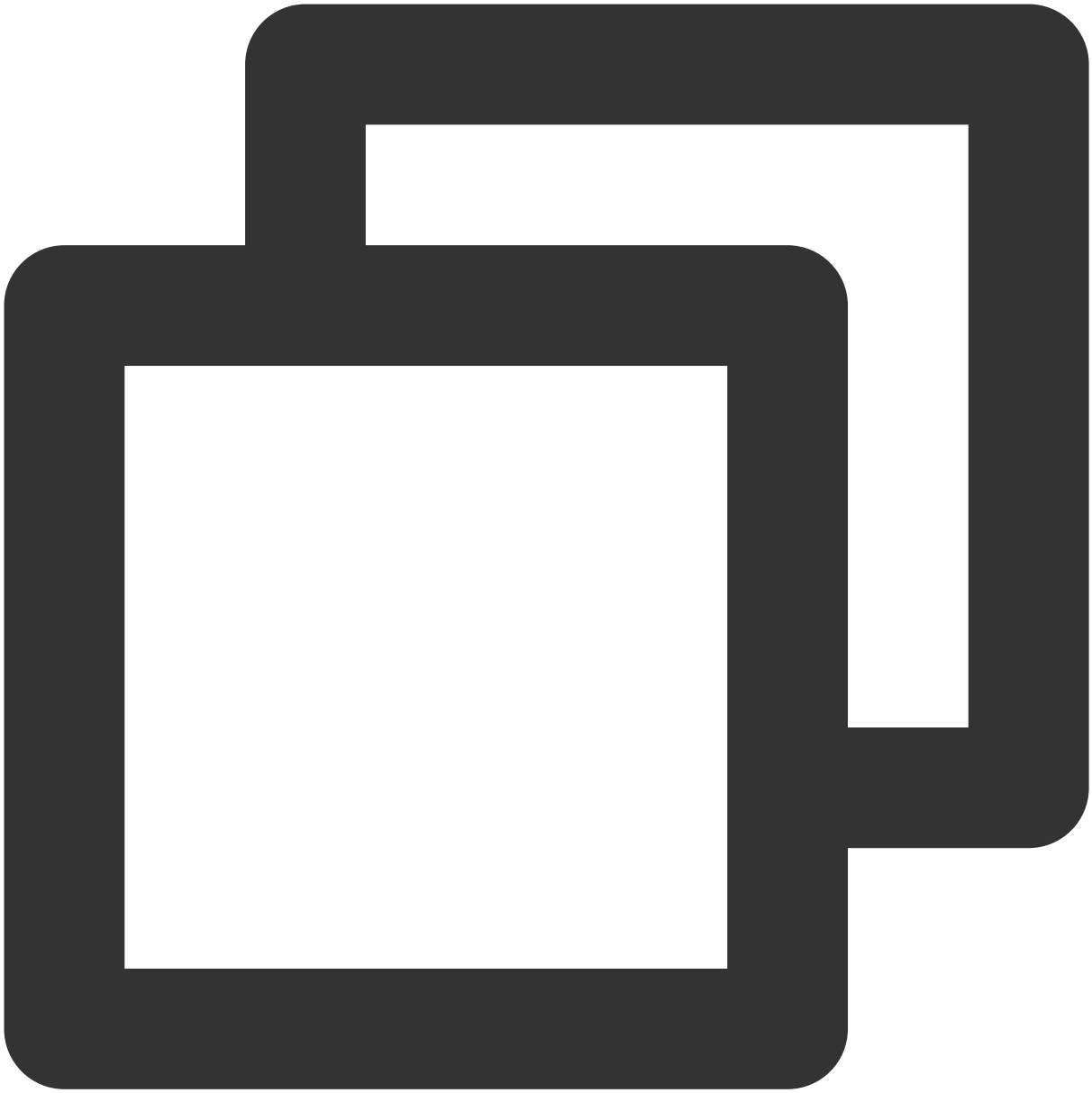
```
        }
    ]
}
```

复制对象

API 接口为 Put Object Copy，若授予其操作权限，则策略的目标对象的 action 为 name/cos:PutObject，和源对象的 action 为 name/cos:GetObject。

示例

授予在 APPID 为1250000000 ，地域为 ap-beijing，存储桶为 examplebucket-1250000000 的路径前缀为 doc 和路径前缀为 doc2 间进行分块复制的操作权限，其策略详细内容如下：



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:PutObject"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
```

```
      },
      {
        "action": [
          "name/cos:GetObject"
        ],
        "effect": "allow",
        "resource": [
          "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc2/*"
        ]
      }
    ]
  }
```

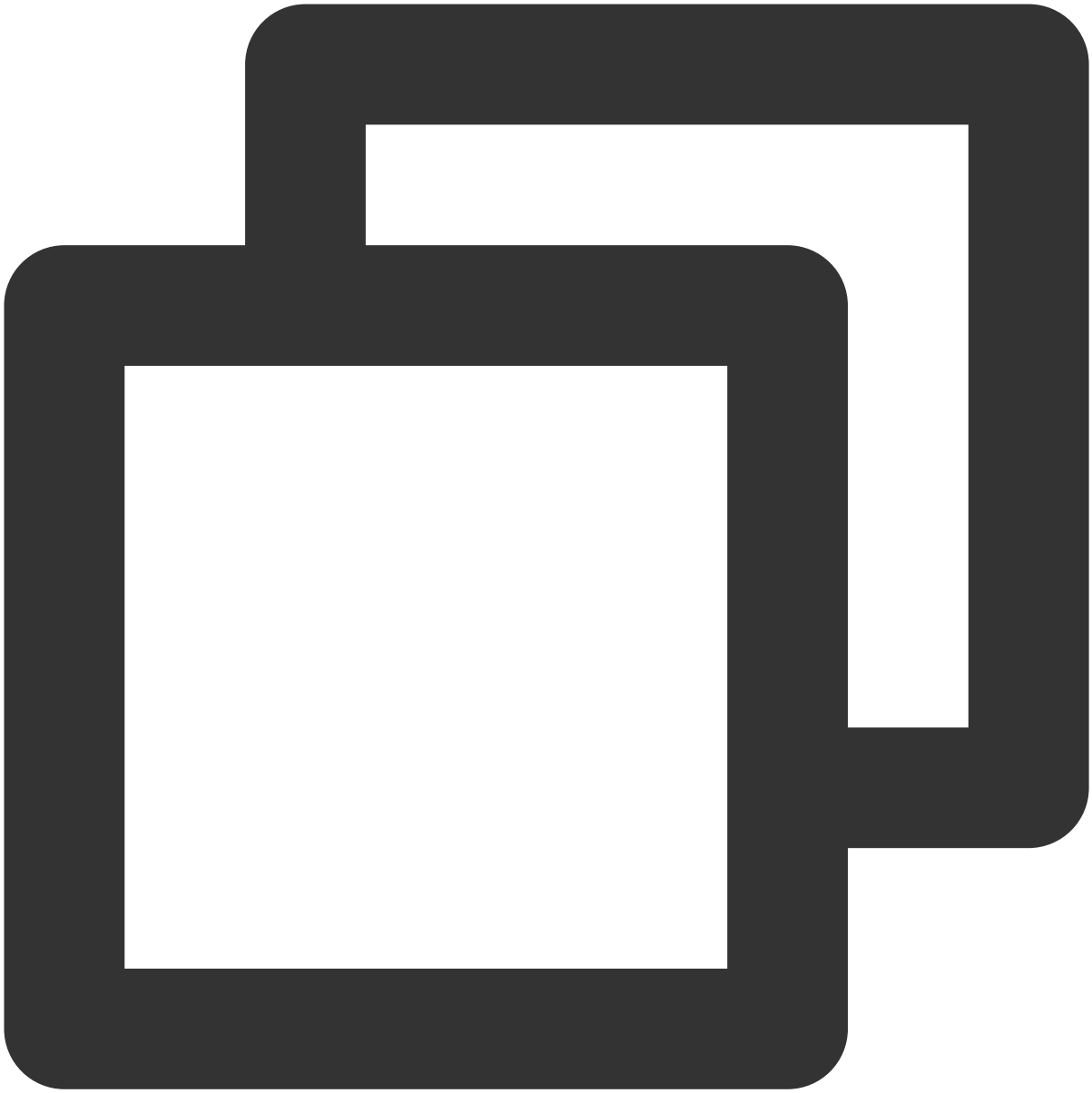
其中 `"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc2/*"` 为源对象。

复制分块

API 接口为 Upload Part - Copy，若授予其操作权限，则策略的目标对象的 action 为 action 为 `"name/cos:InitiateMultipartUpload","name/cos:ListMultipartUploads","name/cos:ListParts","name/cos:PutObject","name/cos:CompleteMultipartUpload","name/cos:AbortMultipartUpload"` 集合， 和源对象的 action 为 name/cos:GetObject。

示例

授予在 APPID 为1250000000 ，地域为 ap-beijing，存储桶为 examplebucket-1250000000 的路径前缀为 doc 和路径前缀为 doc2 间进行分块复制的操作权限，其策略详细内容如下：



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:InitiateMultipartUpload",
        "name/cos:ListMultipartUploads",
        "name/cos:ListParts",
        "name/cos:PutObject",
        "name/cos:CompleteMultipartUpload",
        "name/cos:AbortMultipartUpload"
```

```
        ],
        "effect": "allow",
        "resource": [
          "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
        ]
      },
      {
        "action": [
          "name/cos:GetObject"
        ],
        "effect": "allow",
        "resource": [
          "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc2/*"
        ]
      }
    ]
}
```

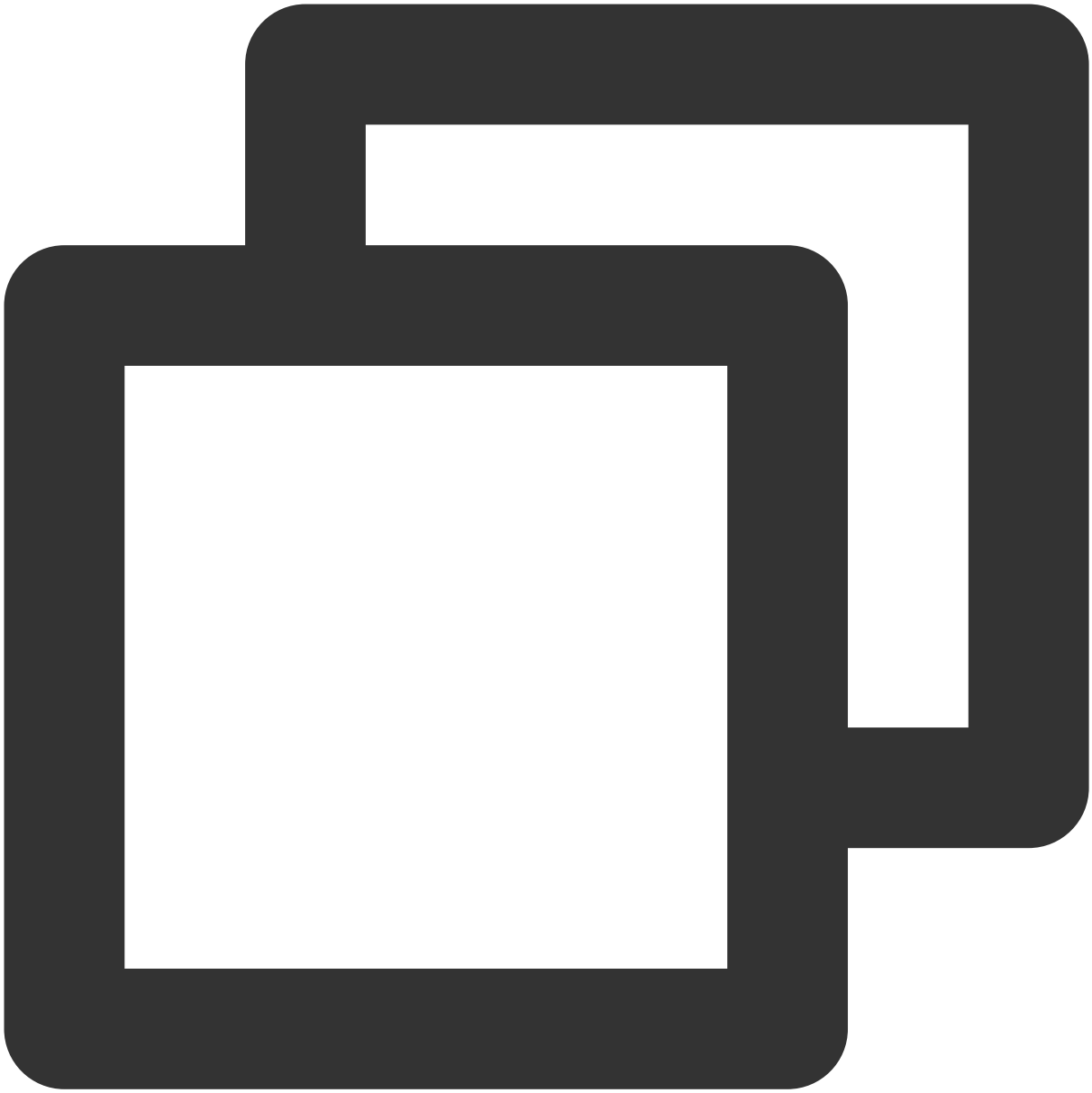
其中 `"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc2/*"` 为源对象。

设置对象 ACL

API 接口为 Put Object ACL，若授予其操作权限，则策略的 action 为 name/cos:PutObjectACL 。

示例

授予只能设置 APPID 为1250000000 ，地域为 ap-beijing ，存储桶为 examplebucket-1250000000，路径前缀为 doc 中的对象 ACL 操作权限，其策略详细内容如下：



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:PutObjectACL"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
```

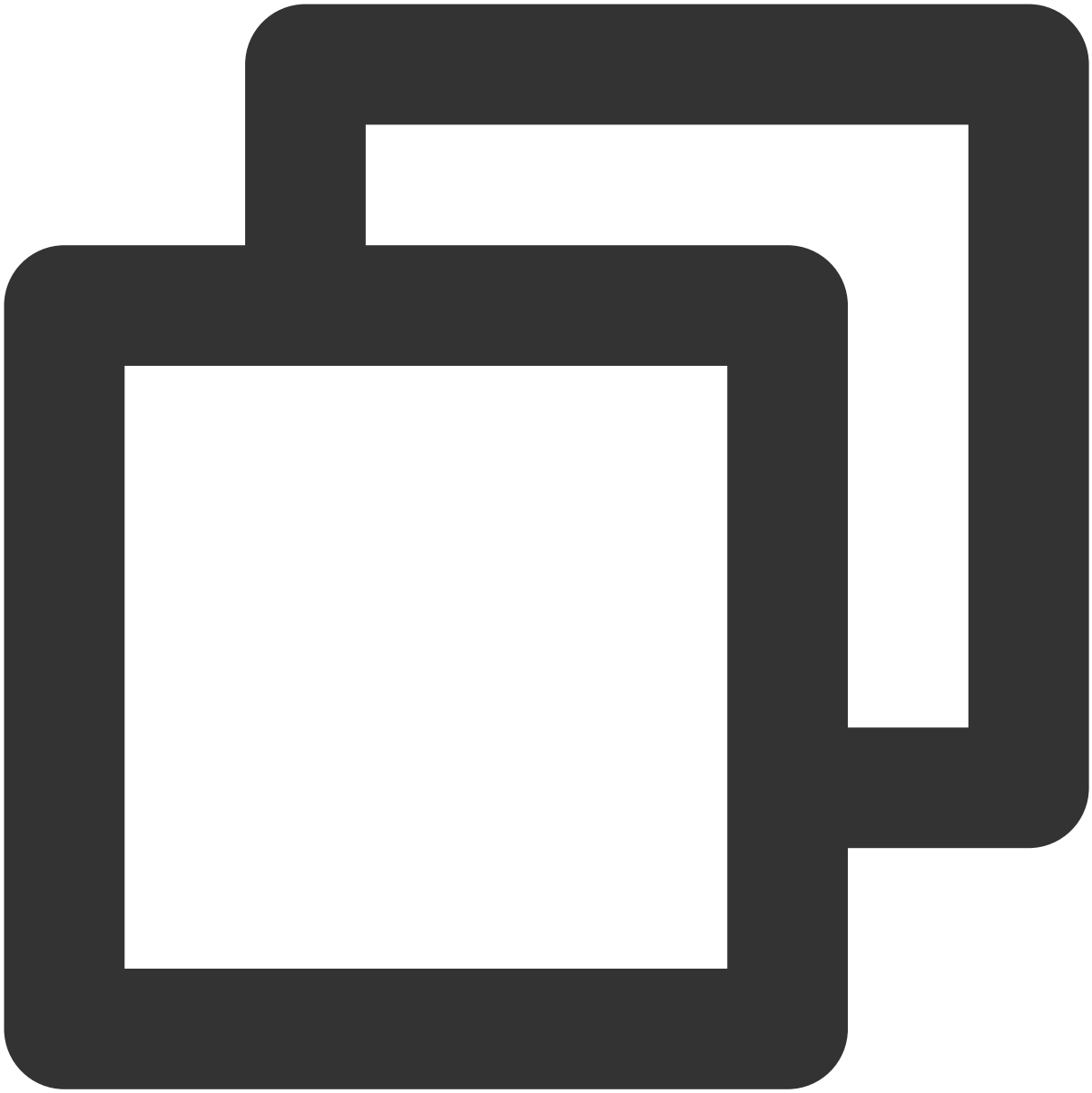
```
        }
    ]
}
```

查询对象 ACL

API 接口为 Get Object ACL，若授予其操作权限，则策略的 action 为 name/cos:GetObjectACL。

示例

授予只能查询 APPID 为1250000000 ，地域为 ap-beijing，存储桶为 examplebucket-1250000000，路径前缀为 doc 中的对象 ACL 操作权限，其策略详细内容如下：



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:GetObjectACL"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
```

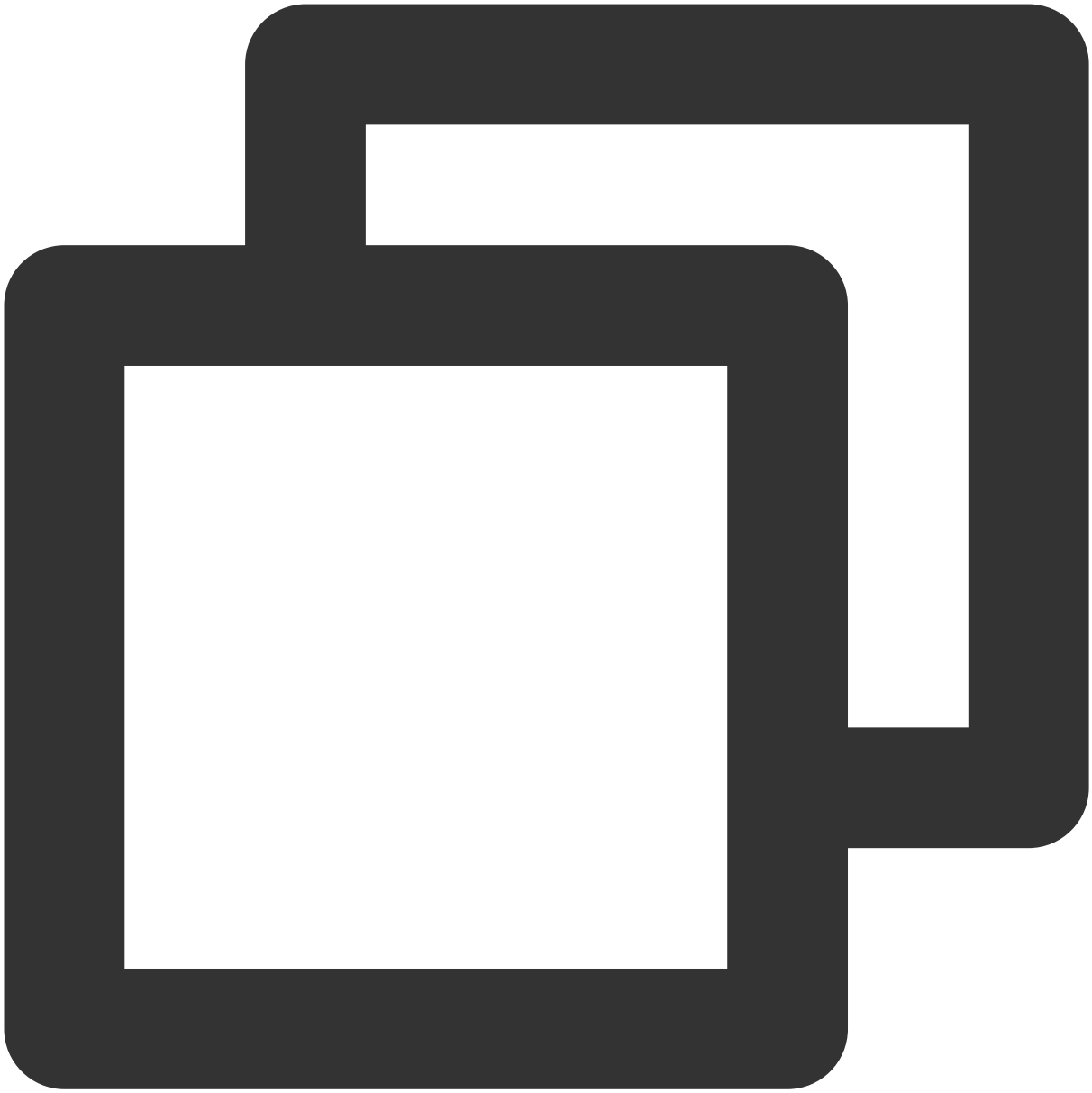
```
        }
    ]
}
```

预请求跨域配置

API 接口为 OPTIONS Object，若授予其操作权限，则策略的 action 为 name/cos:OptionsObject。

示例

授予只能在 APPID 为1250000000 ，地域为 ap-beijing，存储桶为 examplebucket-1250000000，路径前缀为 doc 下进行 Options 请求 的操作权限，其策略详细内容如下：



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:OptionsObject"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
```

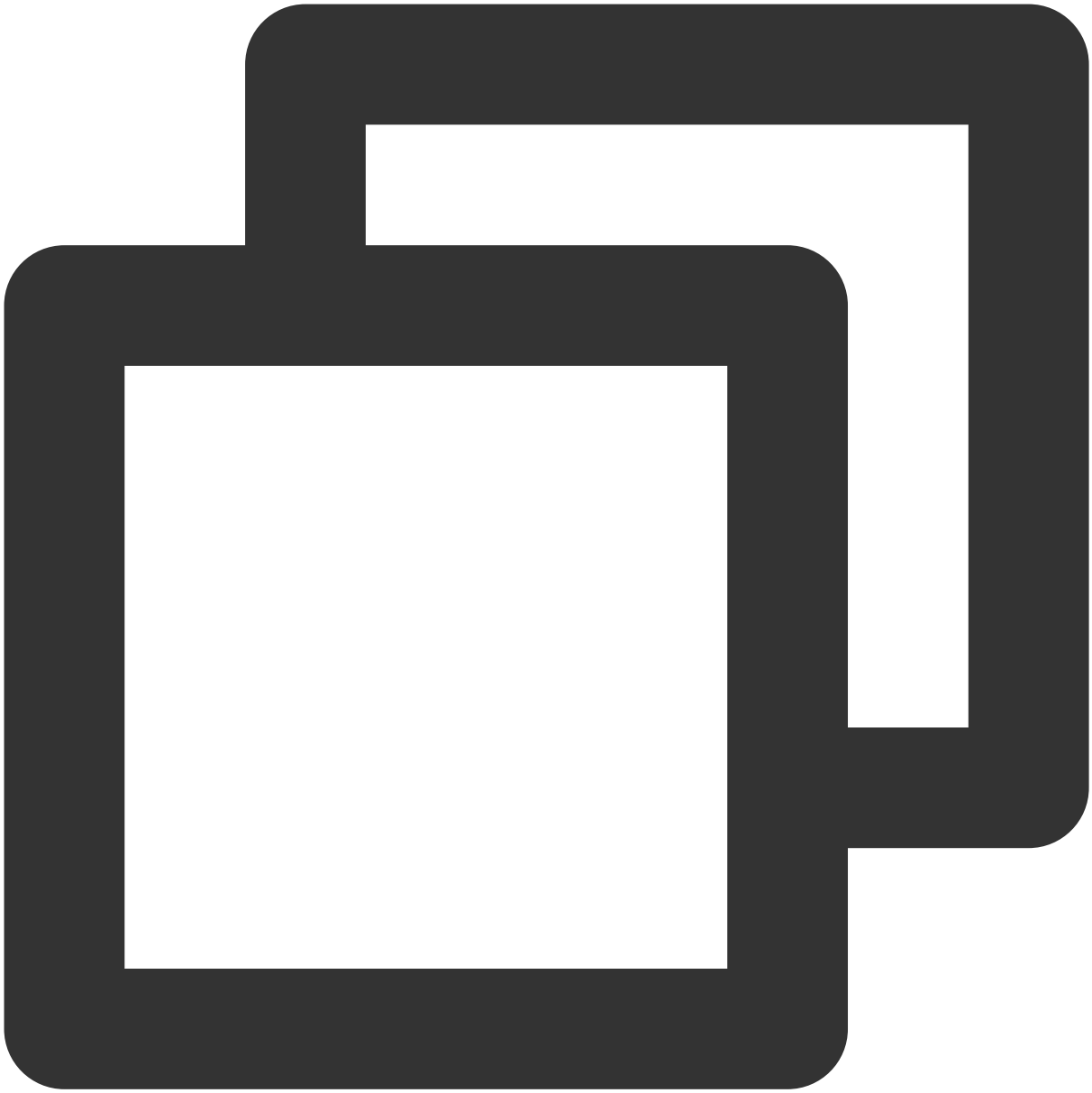
```
        }
    ]
}
```

恢复归档对象

API 接口为 Post Object Restore，若授予其操作权限，则策略的 action 为 name/cos:PostObjectRestore。

示例

授予只能在 APPID 为1250000000 ，地域为 ap-beijing，存储桶为 examplebucket-1250000000，路径前缀为 doc 下进行恢复归档的操作权限，其策略详细内容如下：



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:PostObjectRestore"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
```

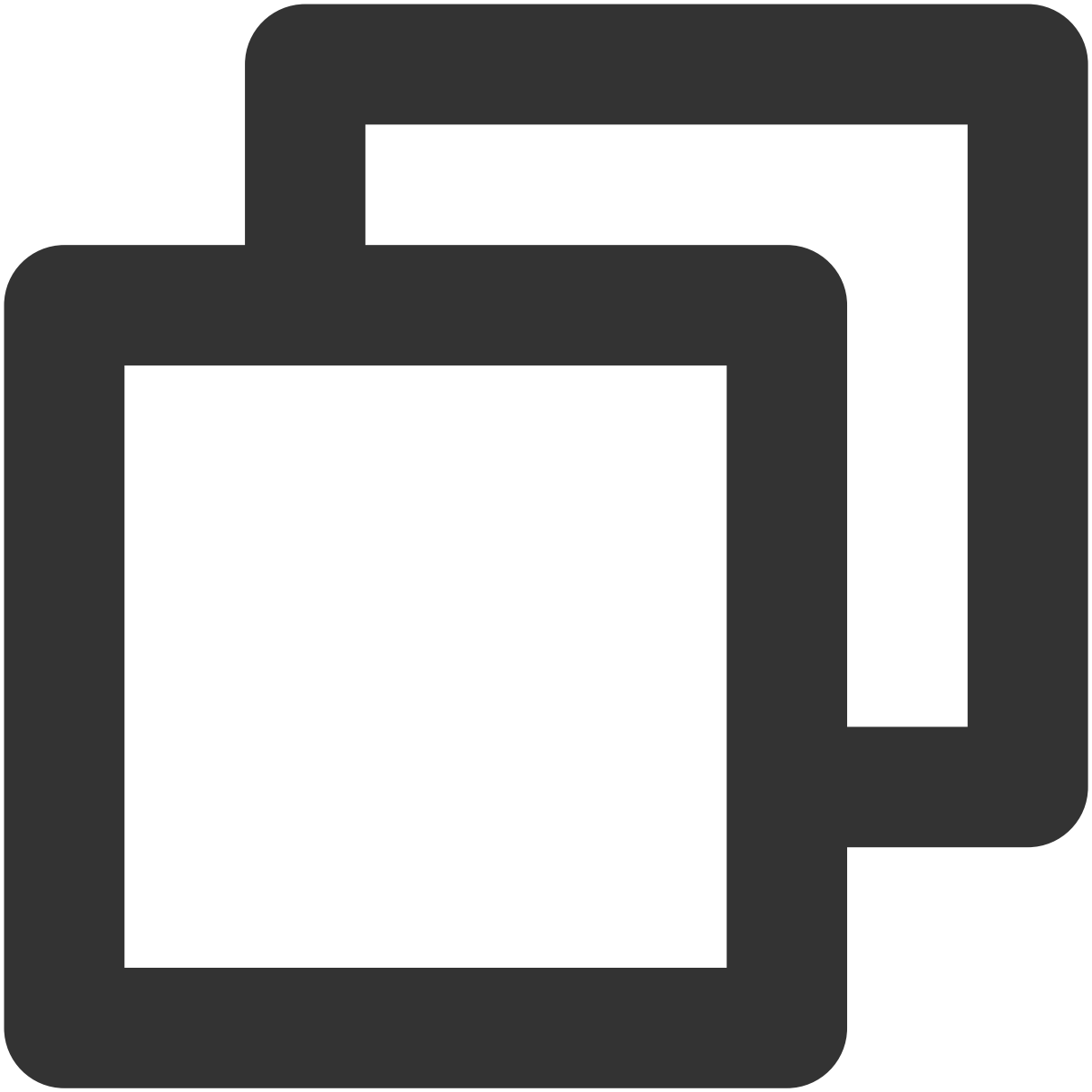
```
        }
    ]
}
```

删除单个对象

API 接口为 DELETE Object，若授予其操作权限，则策略的 action 为 name/cos:DeleteObject。

示例

授予只能删除 APPID 为1250000000 ，地域为 ap-beijing ，存储桶为 examplebucket-1250000000 中的 audio.mp3 这个对象的操作权限，其策略详细内容如下：



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:DeleteObject"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/audio.mp3"
      ]
```

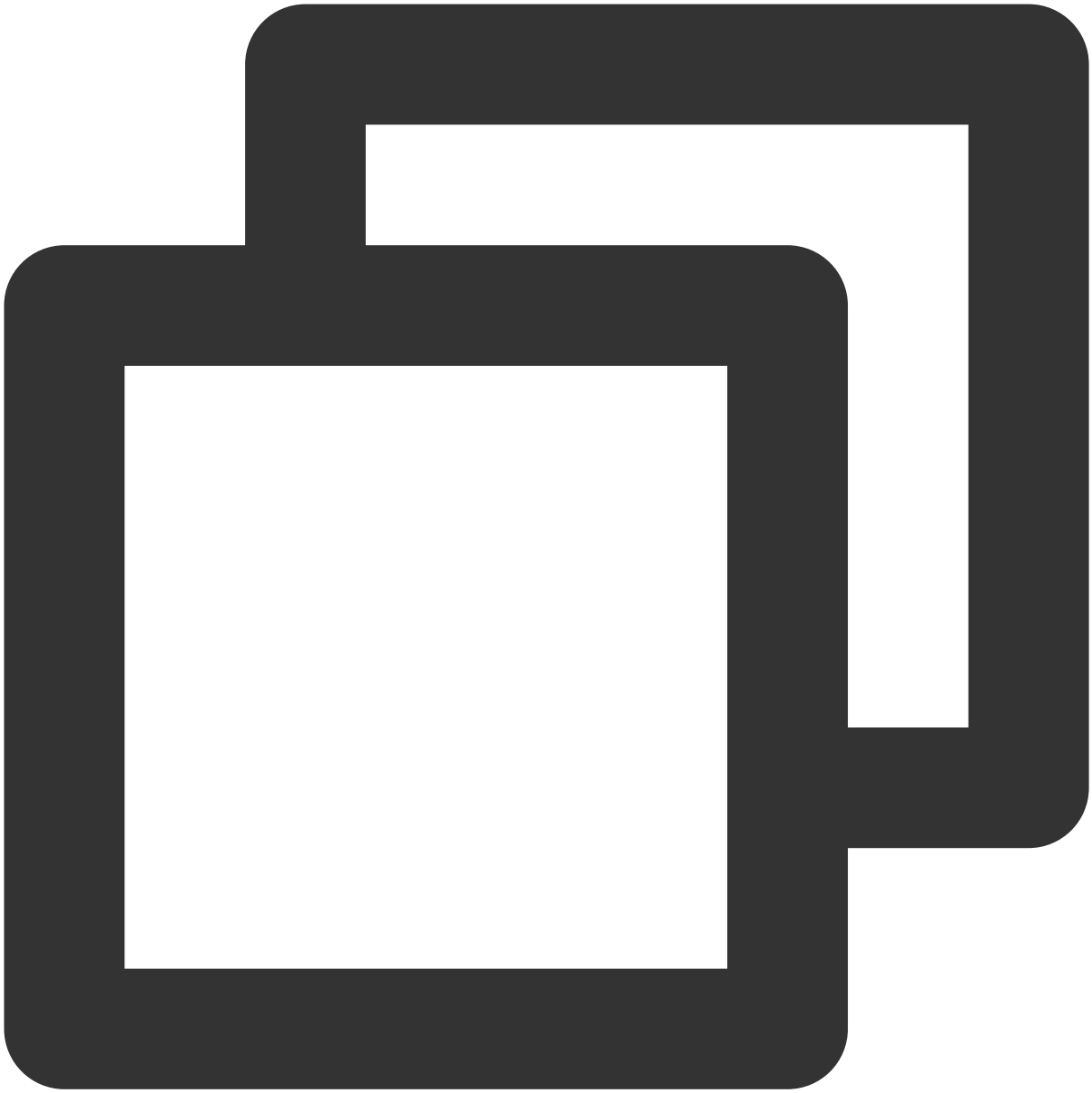
```
        }
    ]
}
```

删除多个对象

API 接口为 DELETE Multiple Objects，若授予其操作权限，则策略的 `action` 为 `name/cos:DeleteObject` 。

示例

授予只能批量删除 APPID 为1250000000 ，地域为 ap-beijing，存储桶为 examplebucket-1250000000 中的 audio.mp3 和 video.mp4 两个对象的操作权限，其策略详细内容如下：



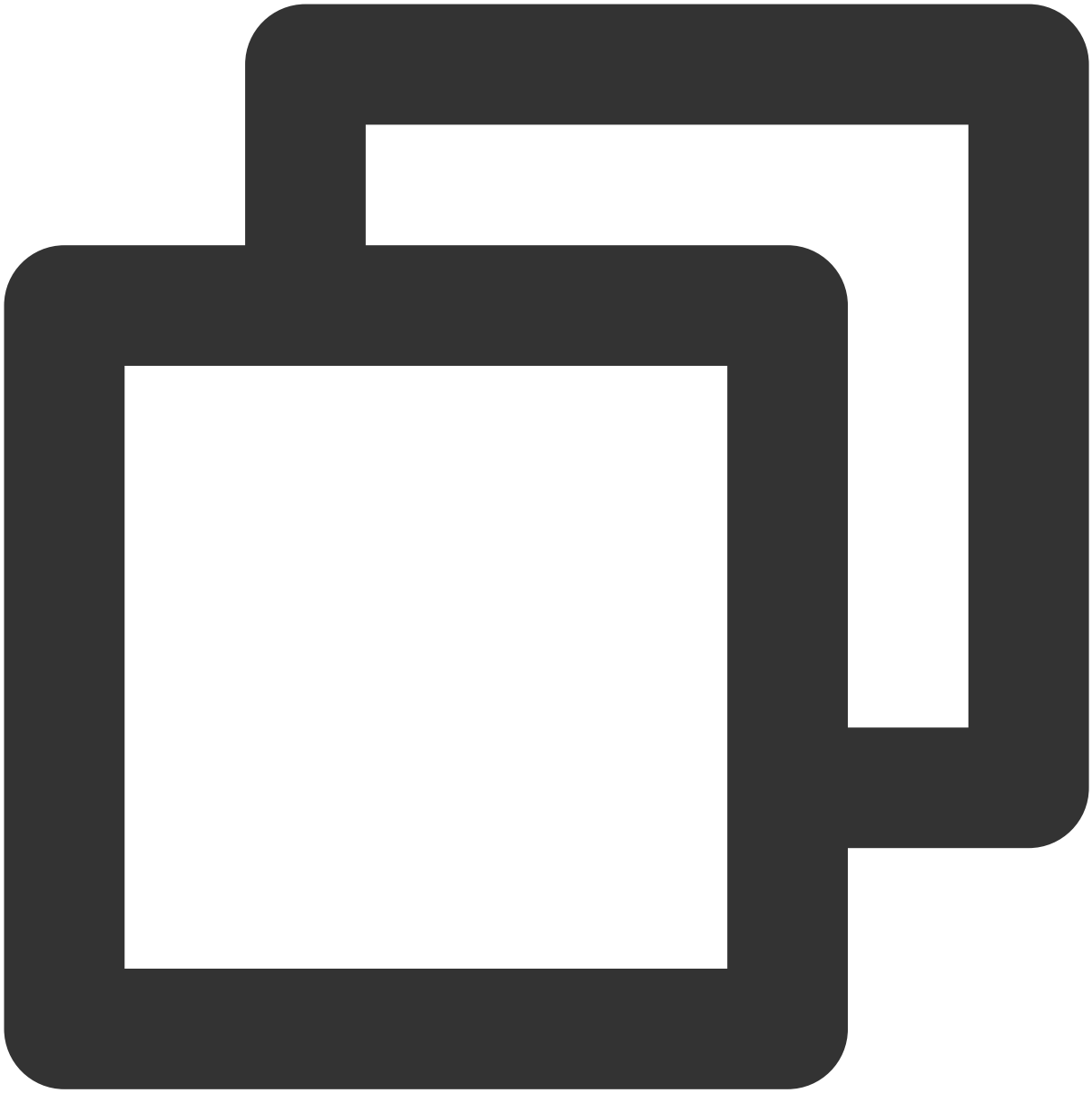
```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "name/cos:DeleteObject"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/audio.mp3",
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/video.mp4"
```

```
        ]
      }
    ]
}
```

常见场景授权策略

授予所有资源完全读写权限

授予所有资源完全读写权限，其策略详细内容如下：

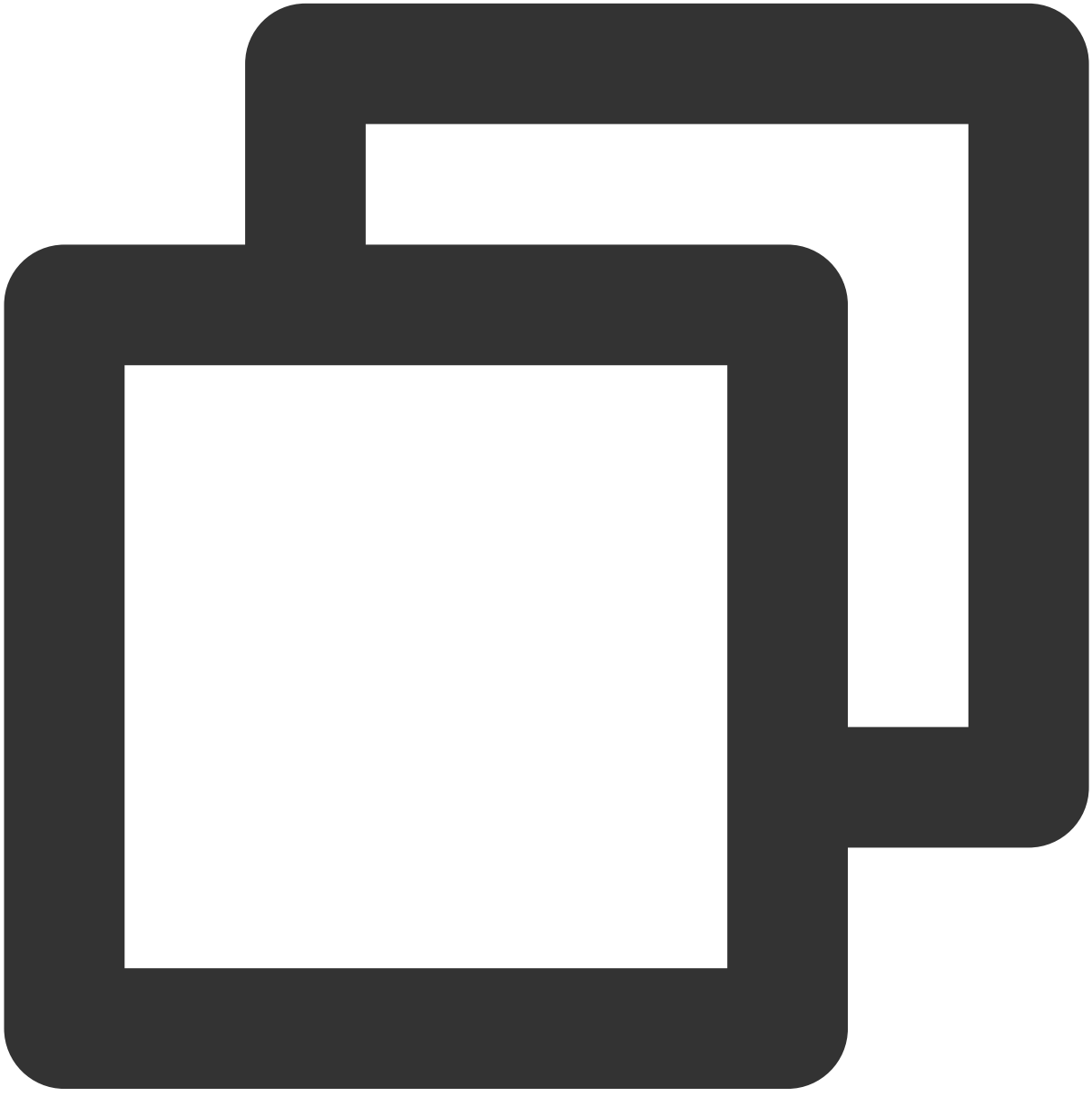


```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "*"
      ],
      "effect": "allow",
      "resource": [
        "*"
      ]
```

```
      }
    ]
}
```

授予所有资源只读权限

授予所有资源只读权限，其策略详细内容如下：

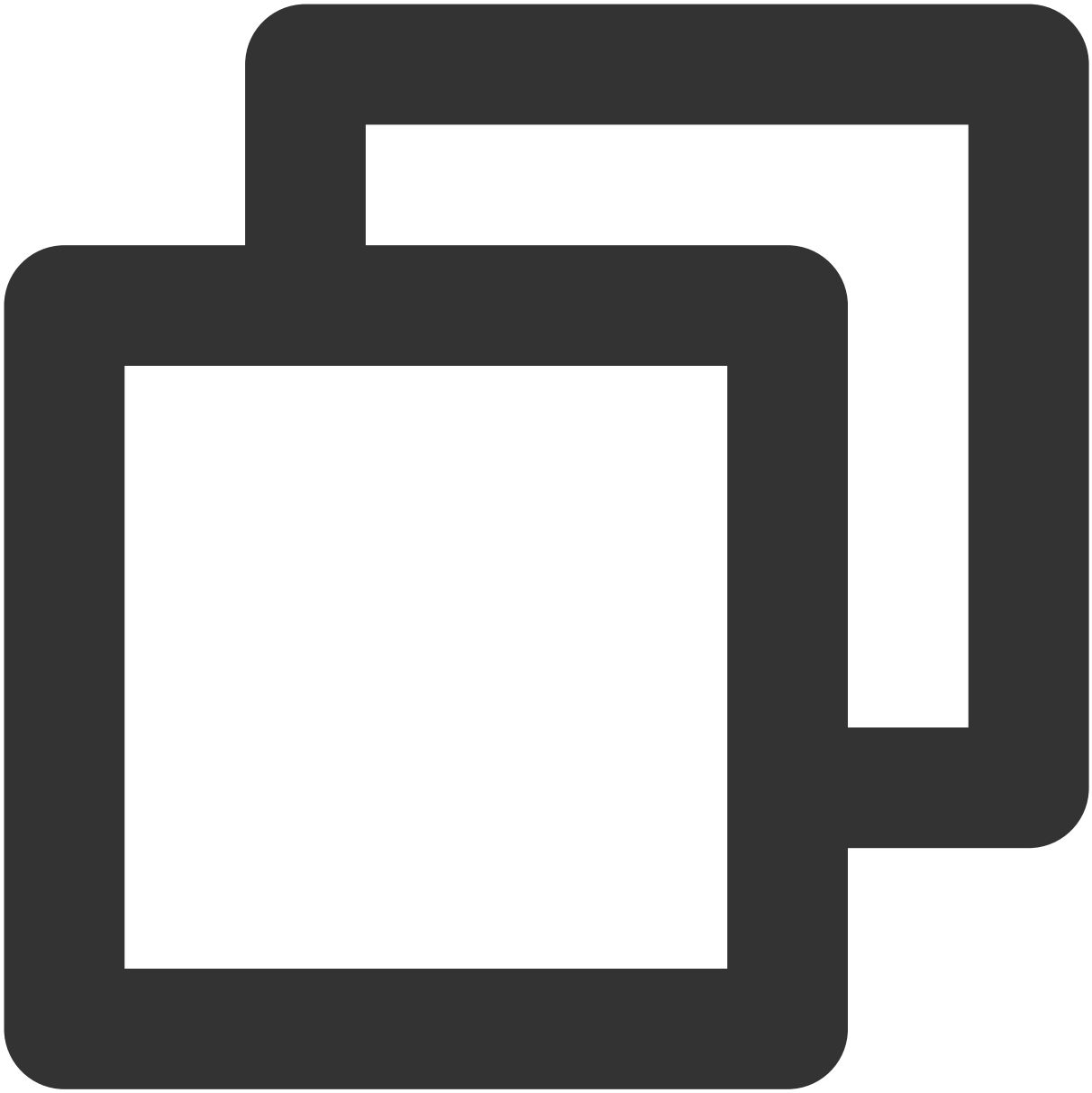


```
{
  "version": "2.0",
  "statement": [
```

```
    {
      "action": [
        "name/cos:HeadObject",
        "name/cos:GetObject",
        "name/cos:GetBucket",
        "name/cos:OptionsObject"
      ],
      "effect": "allow",
      "resource": [
        "*"
      ]
    }
  ]
}
```

授予指定路径前缀的读写操作

授予用户只能访问存储桶 examplebucket-1250000000 中路径前缀为 doc 下的文件，且无法操作其它路径文件的操作权限，该策略详细内容如下：



```
{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "*"
      ],
      "effect": "allow",
      "resource": [
        "qcs::cos:ap-shanghai:uid/1250000000:examplebucket-1250000000/doc/*"
      ]
```

```
        }
    ]
}
```

用于前端直传 COS 的临时密钥安全指引

最近更新时间：2024-01-06 10:47:50

简介

在移动应用和 Web 应用中，您可以通过 iOS/Android/JavaScript SDK 在前端直接向对象存储（Cloud Object Storage，COS）发起请求，此时数据的上传和下载可以不经过您的后端服务器，既节约了您后端服务器的带宽和负载，还可以充分利用 COS 的带宽和全球加速等能力，提升您的应用体验。
在实际应用中，您需要使用临时密钥作为前端的 COS 请求签名，以防止永久密钥的泄漏及越权访问等问题。然而，即便是使用临时密钥，如果您在生成临时密钥时指定了过大的权限或路径，那么同样有可能发生越权访问等问题，将对您的应用带来一定的风险，本文重点介绍了部分风险案例，以及您应当遵守的安全规范，以便让您的应用能够安全的使用 COS。

前提条件

本文假定您已经充分理解临时密钥的相关概念，能够生成及使用临时密钥来请求 COS。有关临时密钥的生成及使用指引，请参见 临时密钥生成及使用指引 文档。

注意

使用临时密钥授权访问时，请务必根据业务需要，按照最小权限原则进行授权。如果您直接授予所有资源（`resource:*`），或者所有操作（`action:*`）权限，则存在由于权限范围过大导致的数据安全风险。

反面案例与安全规范

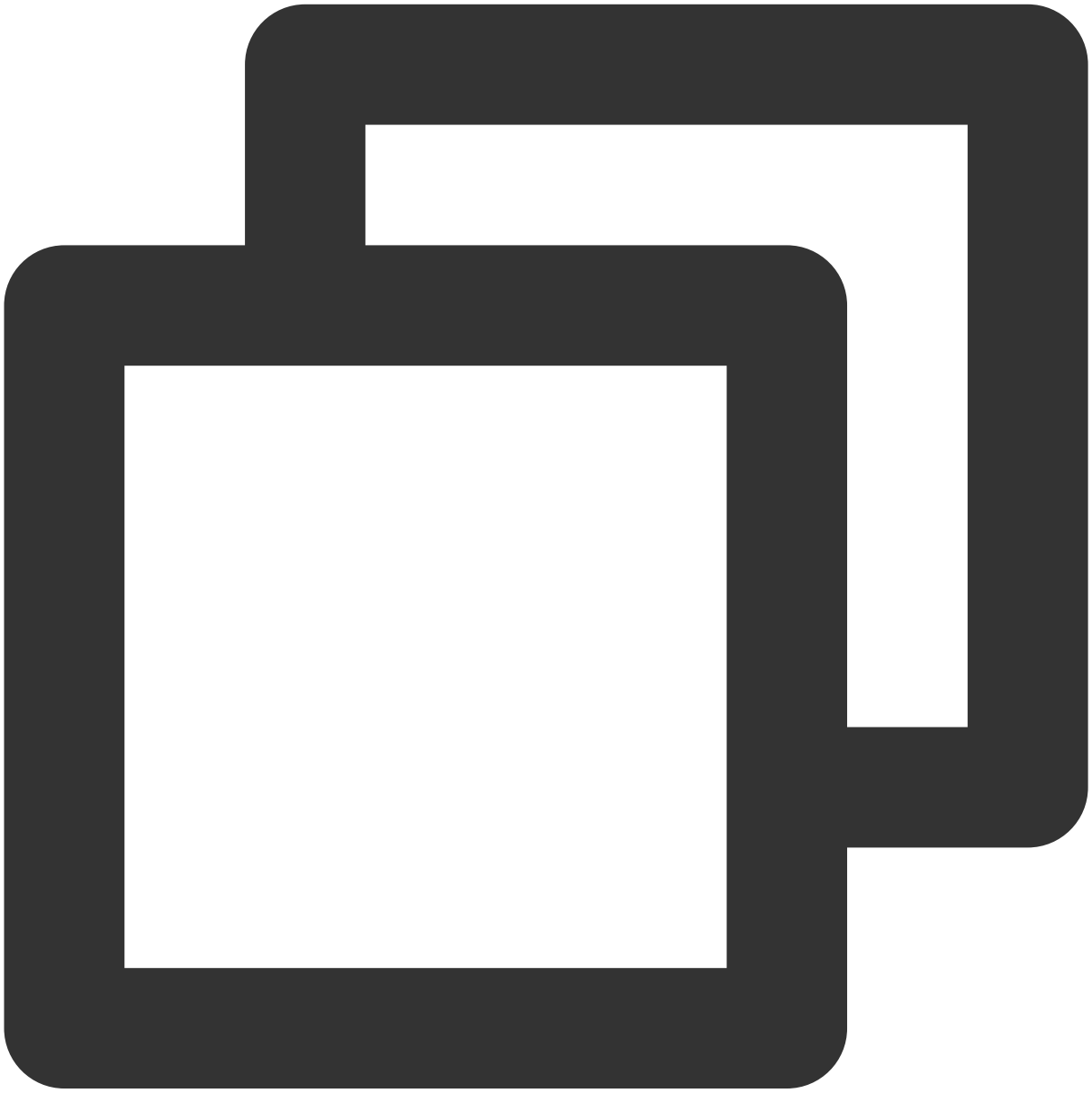
反面案例一：资源（resource）超范围限定

应用 A 在注册用户上传头像中使用到了 COS，每个注册用户的头像拥有固定的对象键 `app/avatar/<Username>.jpg`，同时还会包含头像的不同尺寸，对应的对象键分别为 `app/avatar/<Username>_m.jpg` 和 `app/avatar/<Username>_s.jpg`，后端为了方便使用，在生成临时密钥时直接将 resource 指定为 `qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/avatar/*`，此时恶意用户通过网络抓包等手段获取到生成的临时密钥后，可以覆盖上传任何用户的头像，产生越权访问，用户的合法头像数据被覆盖导致丢失。

安全规范

resource 代表临时密钥所允许访问的资源路径，此时需要充分考虑该路径所覆盖的最终用户，原则上 resource 所指定的资源要求仅能被单一用户所使用。此案例中指定的 `qcs::cos:<Region>:uid/<APPID>:<BucketName-`

`APPID>/app/avatar/*` 显然会覆盖所有用户，因此存在安全漏洞。
在该案例中，可以考虑将用户的头像路径修改为 `app/avatar/<Username>/<size>.jpg` ，此时可以将 resource 指定为 `qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/avatar/<Username>/*` 来满足规范要求；此外，resource 字段支持以数组的形式传入多个值。因此，您也可以显式指定多个 resource 值来完全限定用户有权限访问的最终资源路径，例如：



```
"resource": [
    "qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/avatar/<Username>.jpg",
    "qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/avatar/<Username>_m.jpg",
    "qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/avatar/<Username>_s.jpg"
```

```
]
```

反面案例二：操作（action）超范围限定

应用 B 提供一个公共的照片墙功能，所有照片均存放置在 `app/photos/*` 下面，客户端同时需要列出对象（GET Bucket）和下载对象（GET Object）操作，后端为了方便使用，在生成临时密钥时直接将 action 指定为了 `name/cos:*`，此时恶意用户通过网络抓包等手段获取到生成的临时密钥后，可以对该资源路径下的任何对象执行所有对象操作（例如上传和删除等操作），产生越权访问，导致数据丢失，影响线上业务。

安全规范

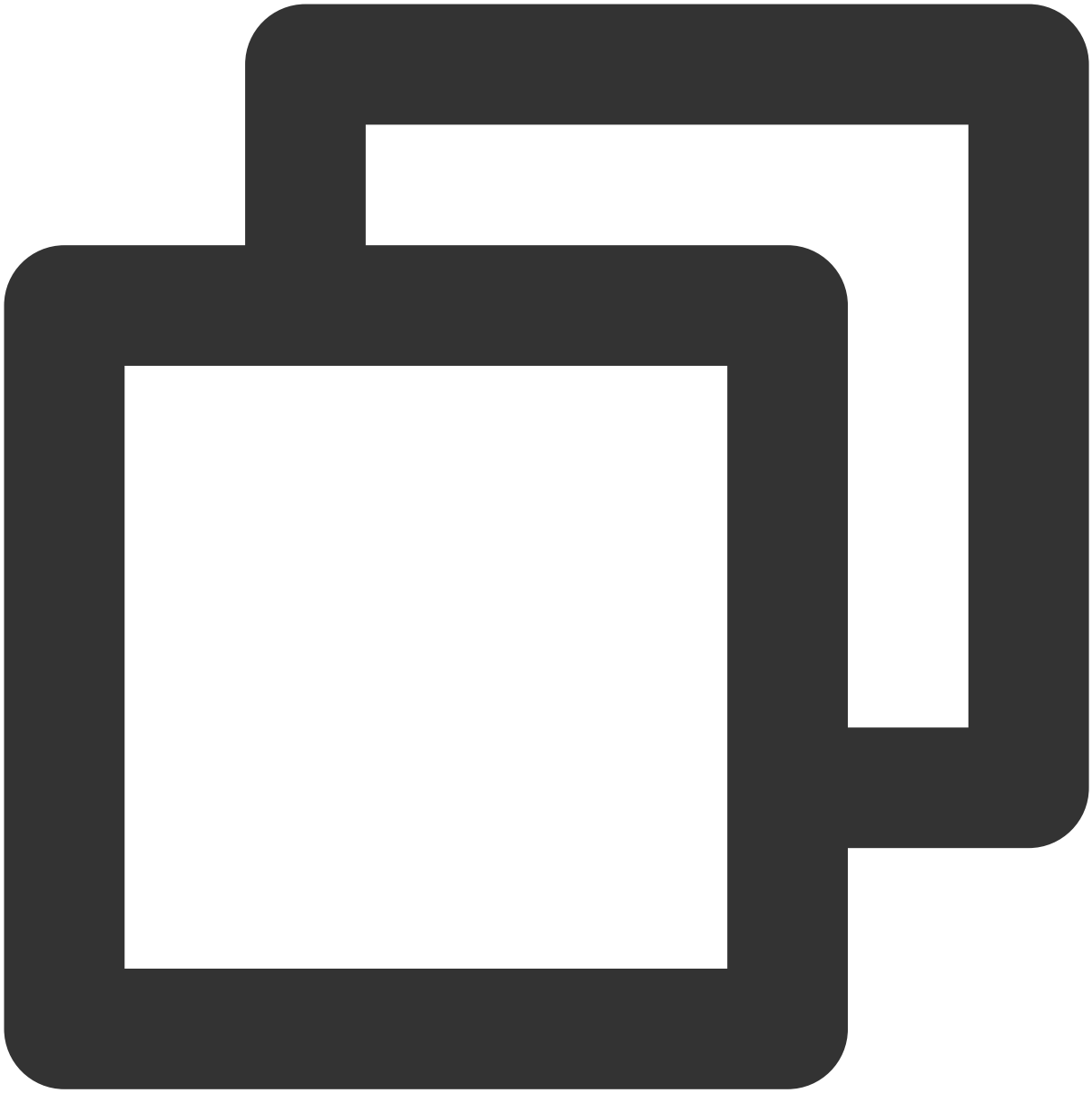
action 代表临时密钥所允许请求的操作，原则上不允许使用 `name/cos:*` 等允许所有操作的临时密钥下发至前端，必须明确列出所有需要用到的操作，同时如果各操作所需的资源路径不同，则需要**操作**与**资源**路径单独匹配，而不应合并处理。
在该案例中，应当使用 `"action": [ "name/cos:GetBucket", "name/cos:GetObject" ]` 指明具体的操作。授权操作指引请参见 COS API 授权策略使用指引。

反面案例三：资源与操作超范围限定

应用 C 提供一个管理工具，允许用户列出并下载所有人的文件（ `app/files/*` ），但只能上传和删除个人目录下的文件（ `app/files/<Username>/*` ），后端为了方便使用，在生成临时密钥时将2种权限，共4种操作（action）混合在一起。2种权限对应的资源路径也混合在一起，此时的临时密钥将具备资源路径中指定的更大权限，即用户可以列出、下载、上传和删除所有人的文件，恶意用户可以据此篡改或删除他人的文件，产生越权访问，用户的合法数据将暴露在风险之中。

安全规范

对于多个 action 与 resource 的组合，不应简单的将它们分别合并，而应当通过多个 statement 的形式组合在一起，避免简单的分别合并导致权限扩大化。
在该案例中，应当使用



```
"statement": [
    {
        "effect": "allow",
        "action": [
            "name/cos:GetBucket",
            "name/cos:GetObject"
        ],
        "resource": "qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/files/*"
    },
    {
        "effect": "allow",
```

```
            "action": [
                "name/cos:PutObject",
                "name/cos:DeleteObject"
            ],
            "resource": "qcs::cos:<Region>:uid/<APPID>:<BucketName-APPID>/app/files/<Us
        }
    ]
```

反面案例四：越权获取临时密钥

应用 D 提供一个论坛应用，论坛的帖子附件保存在 COS 上，该论坛应用分为不同的用户级别，只有发帖数达到一定阈值的活跃用户才能看到某个版面内的帖子和附件，对于一些公共版面，所有用户均可以查看其中的帖子和附件。
在使用 COS 时，后端生成临时密钥的接口会根据前端传入的版面 ID，生成允许下载对应版面附件的临时密钥。但在实现过程中，后端没有判断具体请求的用户是否有权限访问指定的版面 ID，导致任何人均可请求该接口获取可访问私密版面附件的临时密钥，产生越权访问，需要被限制访问的资源未能被有效限制，导致数据意外泄漏。

安全规范

对于获取临时密钥的接口，除了要保证获取到的临时密钥本身的权限在预期范围中，还应判断实际的业务场景，即正确的权限是否被正确的用户所请求，避免低权限用户拿到高权限的临时密钥。
在本案例中，后端接口还应该判断当前请求用户是否具备访问特定版面的权限，如无权访问则不允许返回临时密钥。

总结

以上几个案例说明了在预期外扩大临时密钥的权限可能导致的安全风险。由于前端直传 COS 时，恶意用户比较容易获取到临时密钥内容，因此该场景相对于通过后端访问 COS 来说，需要开发者们更加注重权限的控制。
本文提到的安全规范即最小权限原则，在实际应用中，根据 action 和 resource 可以枚举出所有可能的权限，例如3种 action 和2个 resource，可以计算出3 x 2=6个被允许访问的资源和对应的操作，您可以据此评估每一种情况是否符合预期，如超出预期权限范围，则应当考虑通过枚举多个 statement 的方式拆分权限。
此外，用于生成临时密钥的接口本身也要充分考虑身份认证和鉴权处理，只有获取临时密钥的行为是安全的，这样得到的临时密钥才能确保真正安全。安全链条上，不能有任何一处疏漏！

临时密钥生成及使用指引

最近更新时间：2024-01-06 10:47:50

注意

使用临时密钥授权访问时，请务必根据业务需要，按照最小权限原则进行授权。如果您直接授予所有资源 `(resource:*)`，或者所有操作 `(action:*)` 权限，则存在由于权限范围过大导致的数据安全风险。

您在申请临时密钥时，如果指定了权限范围，那么申请到的临时密钥也只能在权限范围内进行操作。例如您在申请临时密钥时，指定了可以往存储桶 examplebucket-1-1250000000 上传文件的权限范围，那么申请到的密钥**不能**将文件上传到 examplebucket-2-1250000000，也**不能**从 examplebucket-1-1250000000 中下载文件。

临时密钥

临时密钥（临时访问凭证） 是通过 CAM 云 API 提供的接口，获取到权限受限的密钥。

COS API 可以使用临时密钥计算签名，用于发起 COS API 请求。

COS API 请求使用临时密钥计算签名时，需要用到获取临时密钥接口返回信息中的三个字段，如下：

TmpSecretId

TmpSecretKey

Token

使用临时密钥的优势

Web、iOS、Android 使用 COS 时，通过固定密钥计算签名方式不能有效地控制权限，同时把永久密钥放到客户端代码中有极大的泄露风险。如若通过临时密钥方式，则可以方便、有效地解决权限控制问题。

例如，在申请临时密钥过程中，可以通过设置权限策略 policy 字段，限制操作和资源，将权限限制在指定的范围内。

有关 COS API 授权策略，请参见：

COS API 临时密钥授权策略指引

常见场景的临时密钥权限策略示例

获取临时密钥

获取临时密钥，可以通过提供的 COS STS SDK 方式获取，也可以直接请求 STS 云 API 的方式获取。

注意

举例使用的是 Java SDK，需要在 GitHub 上获取 SDK 代码（版本号）。若提示找不到对应 SDK 版本号，请确认是否在 GitHub 上获取到对应版本的 SDK。

COS STS SDK

COS 针对 STS 提供了 SDK 和样例，目前已有 Java、Nodejs、PHP、Python、Go 等多种语言的样例。具体内容请参见 COS STS SDK。各个 SDK 的使用说明请参见 Github 上的 README 和样例。各语言 GitHub 地址如下表格所示：

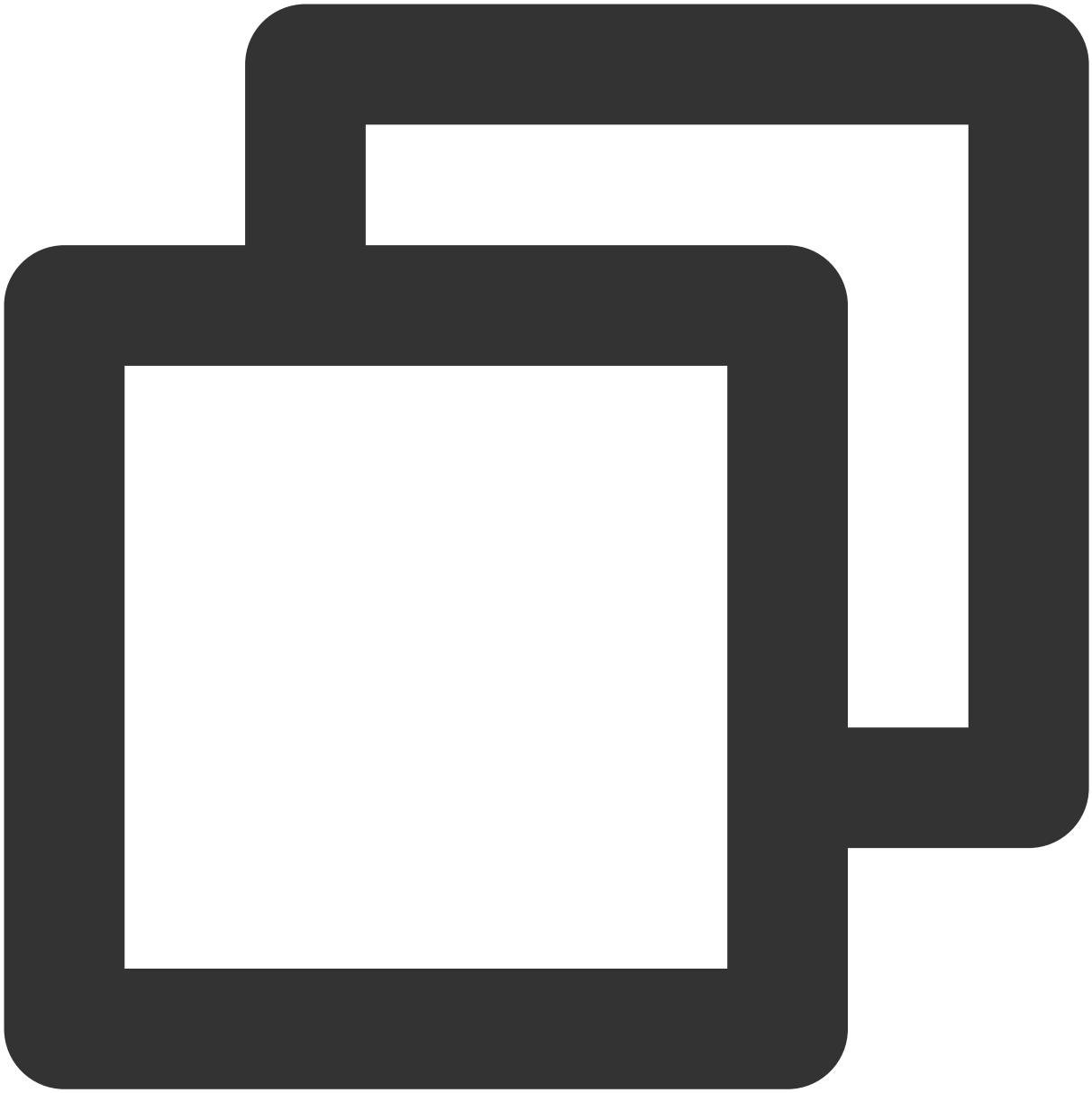
语言类型	代码安装地址	代码示例地址
Java	安装地址	示例地址
.NET	安装地址	示例地址
Go	安装地址	示例地址
NodeJS	安装地址	示例地址
PHP	安装地址	示例地址
Python	安装地址	示例地址

注意

STS SDK 为了屏蔽 STS 接口本身版本间的差异性，返回参数结构不一定与 STS 接口完全一致，详情请参见 Java SDK 文档。

假设您使用的是 Java SDK，请先下载 Java SDK，然后运行如下获取临时密钥示例：

代码示例



```
// 根据 github 提供的 maven 集成方法导入 java sts sdk，使用 3.1.0 及更高版本
public class Demo {
    public static void main(String[] args) {
        TreeMap<String, Object> config = new TreeMap<String, Object>();

        try {
            //这里的 SecretId 和 SecretKey 代表了用于申请临时密钥的永久身份（主账号、子账号等
            String secretId = System.getenv("secretId");//用户的 SecretId，建议使用子账
             String secretKey = System.getenv("secretKey");//用户的 SecretKey，建议使用
            // 替换为您的云 api 密钥 SecretId
            config.put("secretId", secretId);
```

```
        // 替换为您的云 api 密钥 SecretKey
        config.put("secretKey", secretKey);

        // 设置域名:
        // 如果您使用了腾讯云 cvm，可以设置内部域名
        //config.put("host", "sts.internal.tencentcloudapi.com");

        // 临时密钥有效时长，单位是秒，默认 1800 秒，目前主账号最长 2 小时（即 7200 秒），
        config.put("durationSeconds", 1800);

        // 换成您的 bucket
        config.put("bucket", "examplebucket-1250000000");
        // 换成 bucket 所在地区
        config.put("region", "ap-guangzhou");

        // 这里改成允许的路径前缀，可以根据自己网站的用户登录态判断允许上传的具体路径
        // 列举几种典型的前缀授权场景：
        // 1、允许访问所有对象："*"
        // 2、允许访问指定的对象："a/a1.txt", "b/b1.txt"
        // 3、允许访问指定前缀的对象："a*", "a/*", "b/*"
        // 如果填写了“*”，将允许用户访问所有资源；除非业务需要，否则请按照最小权限原则授予用
        config.put("allowPrefixes", new String[] {
                "exampleobject",
                "exampleobject2"
        });

        // 密钥的权限列表。必须在这里指定本次临时密钥所需要的权限。
        // 简单上传、表单上传和分块上传需要以下的权限，其他权限列表请参见 https://www.ten
        String[] allowActions = new String[] {
                 // 简单上传
                "name/cos:PutObject",
                // 表单上传、小程序上传
                "name/cos:PostObject",
                // 分块上传
                "name/cos:InitiateMultipartUpload",
                "name/cos:ListMultipartUploads",
                "name/cos:ListParts",
                "name/cos:UploadPart",
                "name/cos:CompleteMultipartUpload"
        };
        config.put("allowActions", allowActions);
            /**
         * 设置condition（如有需要）
         //# 临时密钥生效条件，关于condition的详细设置规则和COS支持的condition类型可以参
         final String raw_policy = "{\\n" +
         "  \\"version\\":\\"2.0\\",\\n" +
         "  \\"statement\\":[\\n" +
```

```
                "    {\\n" +
                "      \\"effect\\":\\"allow\\",\\n" +
                "      \\"action\\":[\\n" +
                "          \\"name/cos:PutObject\\",\\n" +
                "          \\"name/cos:PostObject\\",\\n" +
                "          \\"name/cos:InitiateMultipartUpload\\",\\n" +
                "          \\"name/cos:ListMultipartUploads\\",\\n" +
                "          \\"name/cos:ListParts\\",\\n" +
                "          \\"name/cos:UploadPart\\",\\n" +
                "          \\"name/cos:CompleteMultipartUpload\\"\\n" +
                "        ],\\n" +
                "      \\"resource\\":[\\n" +
                "          \\"qcs::cos:ap-shanghai:uid/1250000000:examplebucket-125000
                "      ],\\n" +
                "      \\"condition\\": {\\n" +
                "        \\"ip_equal\\": {\\n" +
                "            \\"qcs:ip\\": [\\n" +
                "                \\"192.168.1.0/24\\",\\n" +
                "                \\"101.226.100.185\\",\\n" +
                "                \\"101.226.100.186\\"\\n" +
                "            ]\\n" +
                "        }\\n" +
                "      }\\n" +
                "    }\\n" +
                "  ]\\n" +
                "}";

                config.put("policy", raw_policy);
                */


            Response response = CosStsClient.getCredential(config);
            System.out.println(response.credentials.tmpSecretId);
            System.out.println(response.credentials.tmpSecretKey);
            System.out.println(response.credentials.sessionToken);
        } catch (Exception e) {
            e.printStackTrace();
            throw new IllegalArgumentException("no valid secret !");
        }
    }
}
```

常见问题和解答

JSONObject 包冲突导致 NoSuchMethodError

使用3.1.0及以后的版本。

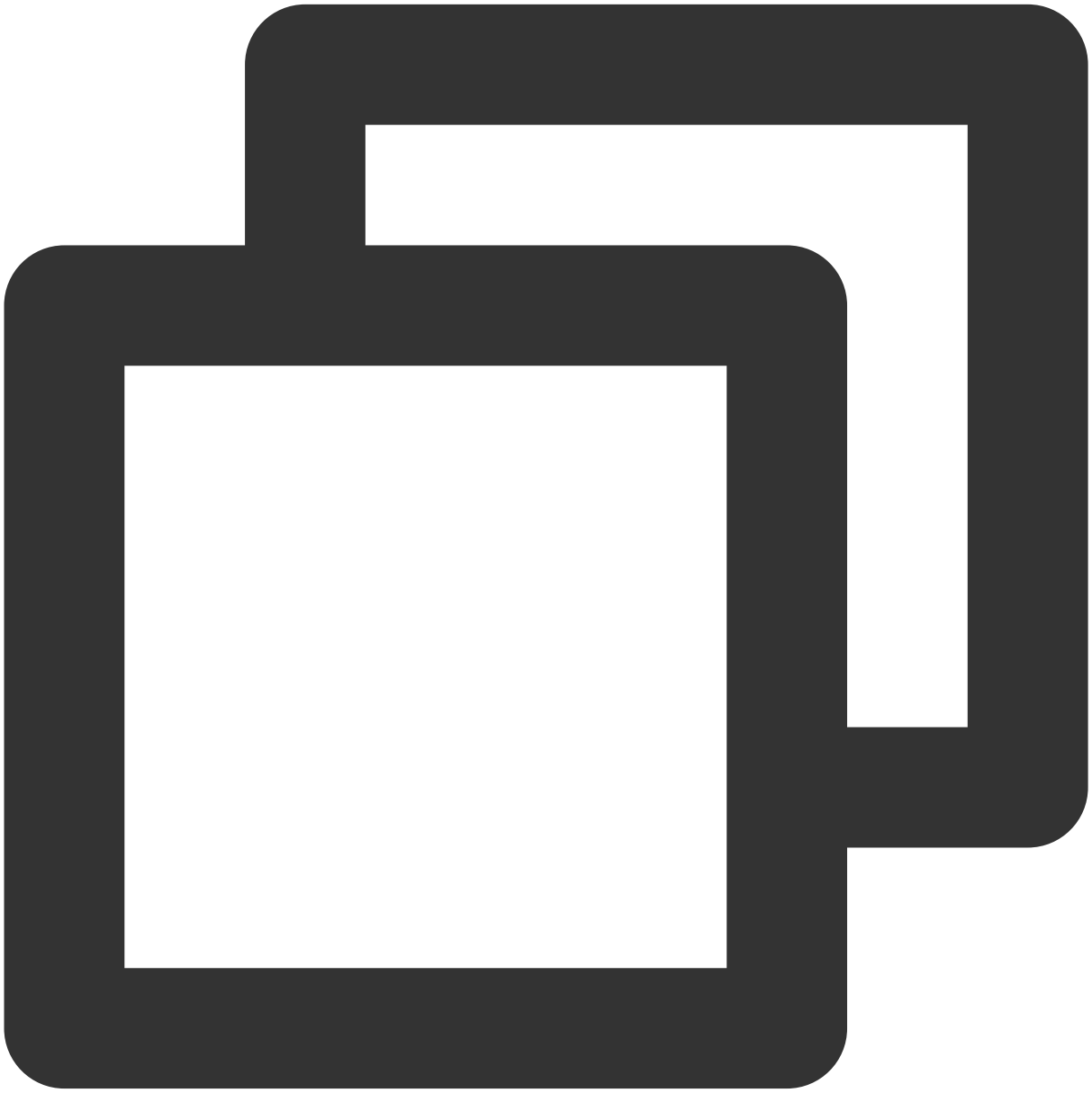
使用临时密钥访问 COS

COS API 使用临时密钥访问 COS 服务时，通过 x-cos-security-token 字段传递临时 sessionToken，通过临时 SecretId 和 SecretKey 计算签名。

以 COS Java SDK 为例，使用临时密钥访问 COS 示例如下：

说明

运行如下示例前，请前往 Github 项目 获取 Java SDK 安装包。



```
// 根据 github 提供的 maven 集成方式导入 cos xml java sdk
import com.qcloud.cos.*;
import com.qcloud.cos.auth.*;
```

```
import com.qcloud.cos.exception.*;
import com.qcloud.cos.model.*;
import com.qcloud.cos.region.*;
public class Demo {
    public static void main(String[] args) throws Exception {

        // 用户基本信息
        String tmpSecretId = "COS_SECRETID";    // 替换为 STS 接口返回给您的临时 SecretI
        String tmpSecretKey = "COS_SECRETKEY";  // 替换为 STS 接口返回给您的临时 Secret
        String sessionToken = "Token";  // 替换为 STS 接口返回给您的临时 Token

        // 1 初始化用户身份信息(secretId, secretKey)
        COSCredentials cred = new BasicCOSCredentials(tmpSecretId, tmpSecretKey);
        // 2 设置 bucket 区域,详情请参见 COS 地域 https://cloud.tencent.com/document/pr
        ClientConfig clientConfig = new ClientConfig(new Region("ap-guangzhou"));
        // 3 生成 cos 客户端
        COSClient cosclient = new COSClient(cred, clientConfig);
        // bucket 名需包含 appid
        String bucketName = "examplebucket-1250000000";

        String key = "exampleobject";
        // 上传 object, 建议 20M 以下的文件使用该接口
        File localFile = new File("src/test/resources/text.txt");
        PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, l

        // 设置 x-cos-security-token header 字段
        ObjectMetadata objectMetadata = new ObjectMetadata();
        objectMetadata.setSecurityToken(sessionToken);
        putObjectRequest.setMetadata(objectMetadata);

        try {
            PutObjectResult putObjectResult = cosclient.putObject(putObjectRequest)
            // 成功：putobjectResult 会返回文件的 etag
            String etag = putObjectResult.getETag();
        } catch (CosServiceException e) {
            //失败, 抛出 CosServiceException
            e.printStackTrace();
        } catch (CosClientException e) {
            //失败, 抛出 CosClientException
            e.printStackTrace();
        }

        // 关闭客户端
        cosclient.shutdown();

    }
}
```

授权子账号按照存储桶标签拉取存储桶列表

最近更新时间：2024-01-06 10:47:50

简介

对象存储（Cloud Object Storage，COS）控制台、API 提供了按存储桶标签筛选存储桶列表的功能，该功能通过标签授权实现。

授权步骤

1. 使用主账号 Owner 登录到 访问管理 控制台，进入到策略配置页面。
2. 根据以下步骤，授权子账号 SubUser 子账号访问具有指定标签的存储桶，可通过**策略生成器**或**策略语法**实现。

通过策略生成器

通过策略语法

1. 进入 访问管理 策略配置页面。
2. 单击**新建自定义策略 > 按策略生成器创建**。
3. 进入授权配置页面，配置信息如下：

效果：选择为允许，默认不变。

服务：选择对象存储。

操作：选择**读操作 > GetService(拉取存储桶列表)**。

资源：选择**全部资源**。

条件：单击**添加其他条件**。进入侧窗配置，配置信息如下：

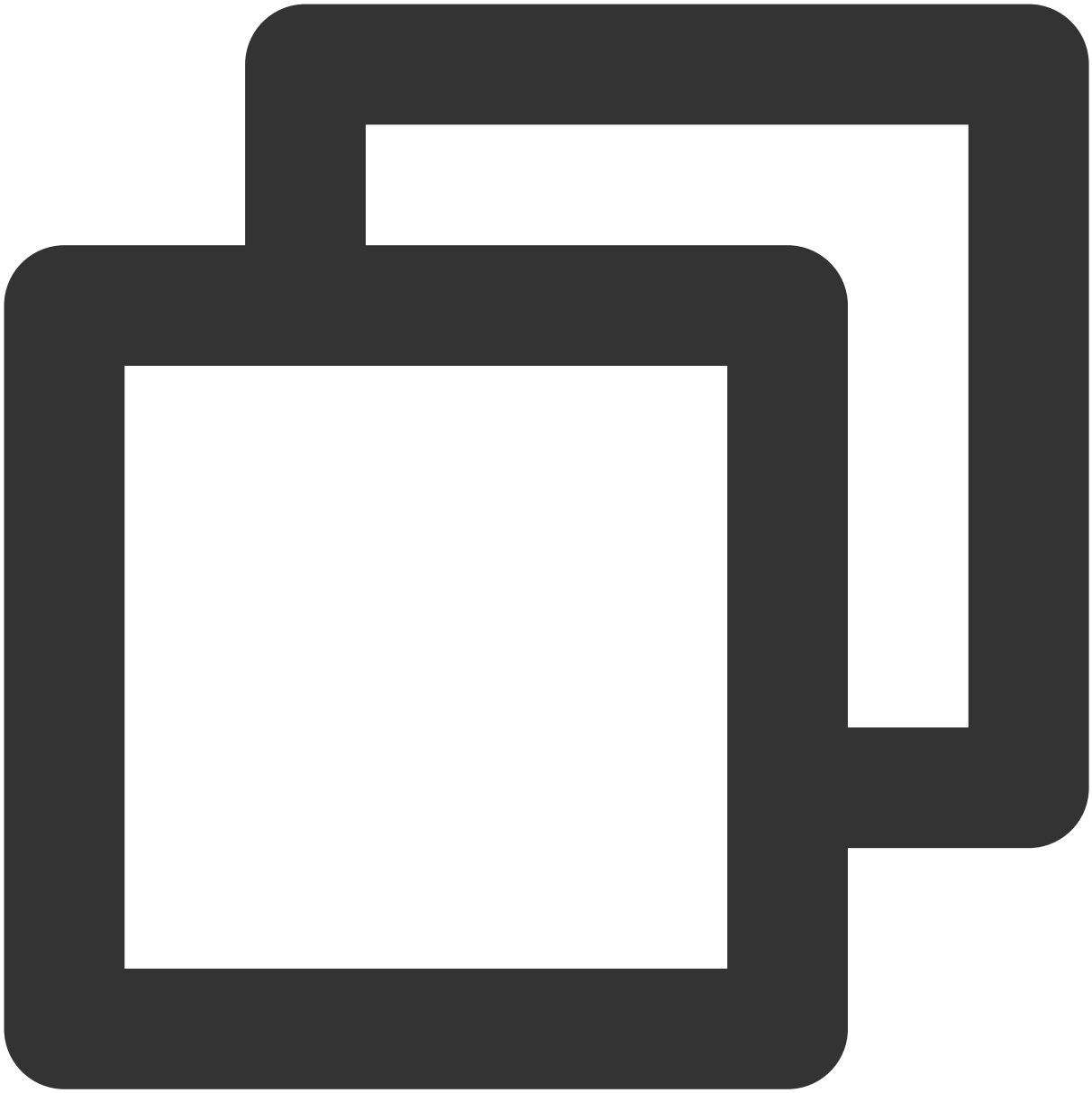
条件键：选择 `qcs:resource_tag` 。

运算符：选择 `string_equal` 。

条件值：按照 `key&val` 格式输入标签，将 key 替换为标签键，value 替换为标签值。

4. 单击**下一步**，输入策略名称。
5. 单击**完成**，即可完成创建。

1. 进入 访问管理 策略配置页面。
2. 单击**新建自定义策略 > 按策略语法创建**。
3. 选择空白模板创建，单击**下一步**。
4. 按照如下策略格式进行输入。其中，需要将 key 和 value 分别替换为指定的标签键和标签值。



```
{
 "version": "2.0",
 "statement": [
     {
         "effect": "allow",
         "action": [
             "name/cos:GetService"
         ],
         "resource": "*",
         "condition": {
             "for_any_value:string_equal": {
```

```
                "qcs:resource_tag": [
                    "key&value"
                ]
            }
        }
    }
  ]
}
```

5. 单击**完成**，即可完成创建。

3. 将策略关联子账号 SubUser。在策略页面，找到步骤2创建的策略，在其右侧单击**关联用户/组/角色**。

4. 在弹窗中，勾选子账号 SubUser，并单击**确定**，即可将子账号 SubUser 关联至该策略。

控制台查看

1. 通过子账号 SubUser 登录 COS 控制台。

2. 在存储桶列表页面，将**自动展示该子账号有权限访问的存储桶列表**。

经过上述步骤，即完成了子账号授权访问包含指定标签（key，value）的存储桶。

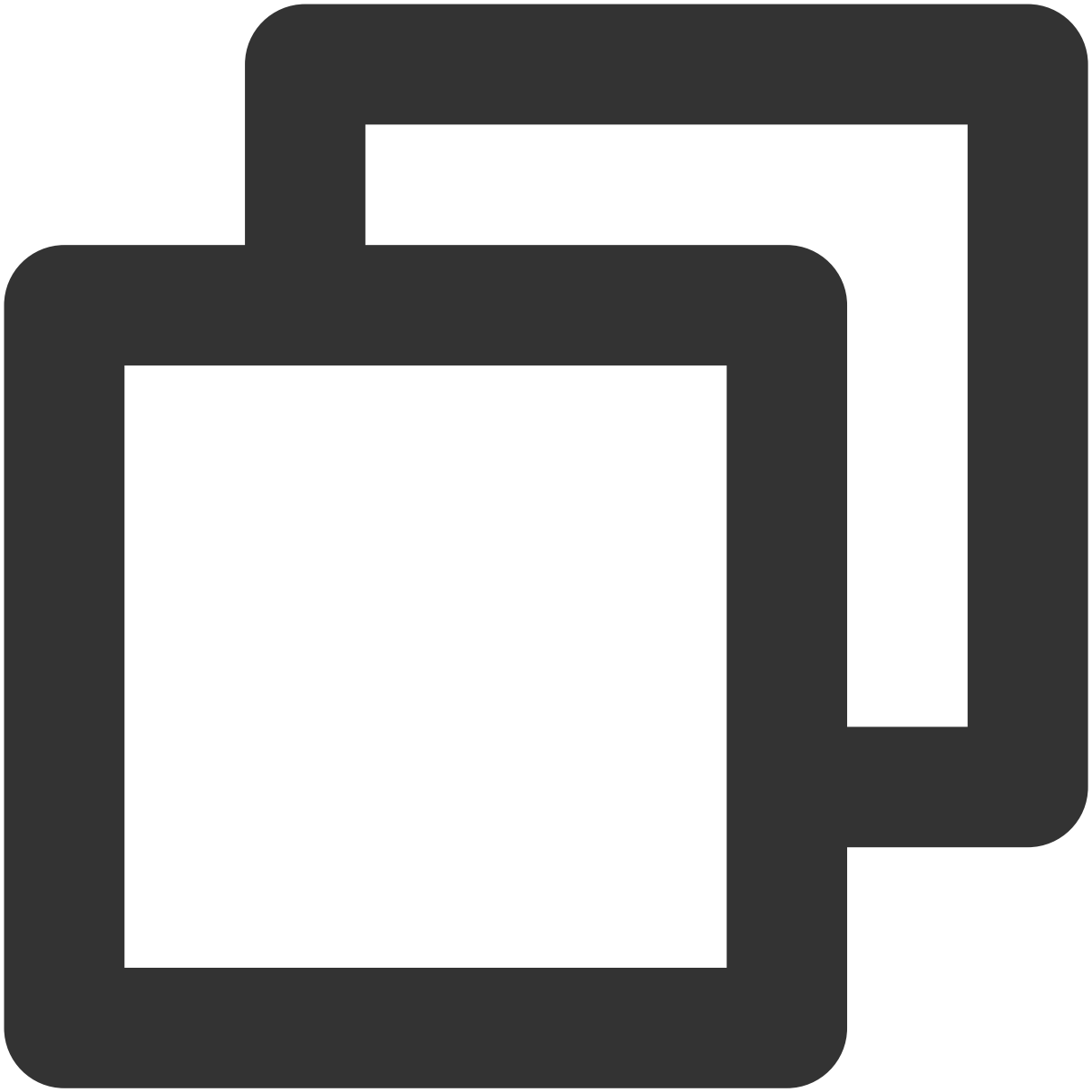
接口调用

注意

与控制台不同，调用 GetService API 接口不支持自动展示子账号有权限访问的存储桶列表，必须传入标签参数。GetService 接口当前仅支持传入一个标签。

1. 使用子账号 SubUser 的密钥发起请求。

2. 调用 GetService 接口，传入标签过滤参数，例如(key,value)。请求示例如下，详情可参见 GET Service（List Buckets）。



```
GET /?tagkey=key1&tagvalue=value1 HTTP/1.1
Date: Fri, 24 May 2019 11:59:51 GMT
Authorization: Auth String
```

条件键说明及使用示例

最近更新时间：2024-01-06 10:47:50

在使用访问策略授予权限时，您可以指定策略的 生效条件，例如限制用户访问来源、上传文件的存储类型等。
本文档为您提供了在存储桶策略中使用对象存储（Cloud Object Storage，COS）条件键的常用示例，您可以在 生效条件 文档中查看 COS 支持的全部条件键和适用请求。

说明

当您使用条件键编写策略时，请务必遵循最小权限原则，仅为适用请求（action）添加相应的条件键，避免在指定操作（action）时使用通配符“*”，导致请求失败，关于条件键的介绍，可参见 生效条件 文档。
当您使用访问管理 CAM 控制台创建策略时，请注意语法格式，version、principal、statement、effect、action、resource、condition 语法元素需保持首字母大写或者全小写。

条件键使用示例

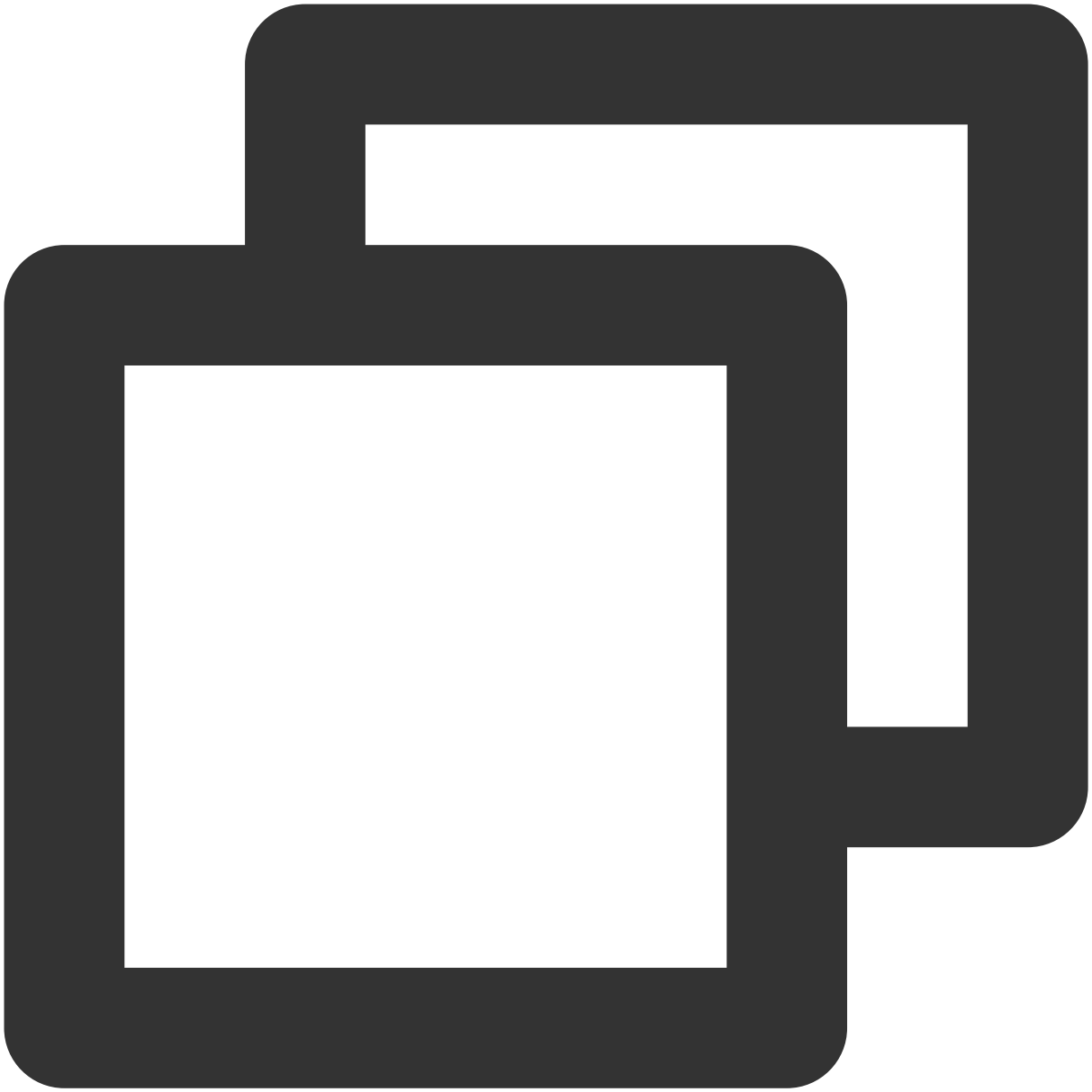
限制用户访问 IP（qcs:ip）

条件键 qcs:ip

使用条件键 `qcs:ip` 限制用户访问 IP，适用于所有请求。

示例：只允许指定 IP 来源的用户访问

以下策略示例描述为：允许属于主账号 ID 为100000000001（APPID 为1250000000）下的子账号 ID 100000000002，对北京地域的存储桶 examplebucket-bj 和广州地域的存储桶 examplebucket-gz 下的对象 exampleobject，在访问 IP 在 `192.168.1.0/24` 网段和 IP 为 `101.226.100.185` 或 `101.226.100.186` 时，拥有上传对象和下载对象的权限。



```
{
    "version": "2.0",
    "principal": {
        "qcs": [
            "qcs::cam::uin/100000000001:uin/100000000002"
        ]
    },
    "statement": [
        {
            "effect": "allow",
            "action": [
```

```
                "name/cos:PutObject",
                "name/cos:GetObject"
            ],
            "resource": [
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-gz-1250000000/e
            ],
            "condition": {
                "ip_equal": {
                    "qcs:ip": [
                        "192.168.1.0/24",
                        "101.226.100.185",
                        "101.226.100.186"
                    ]
                }
            }
        }
    ]
}
```

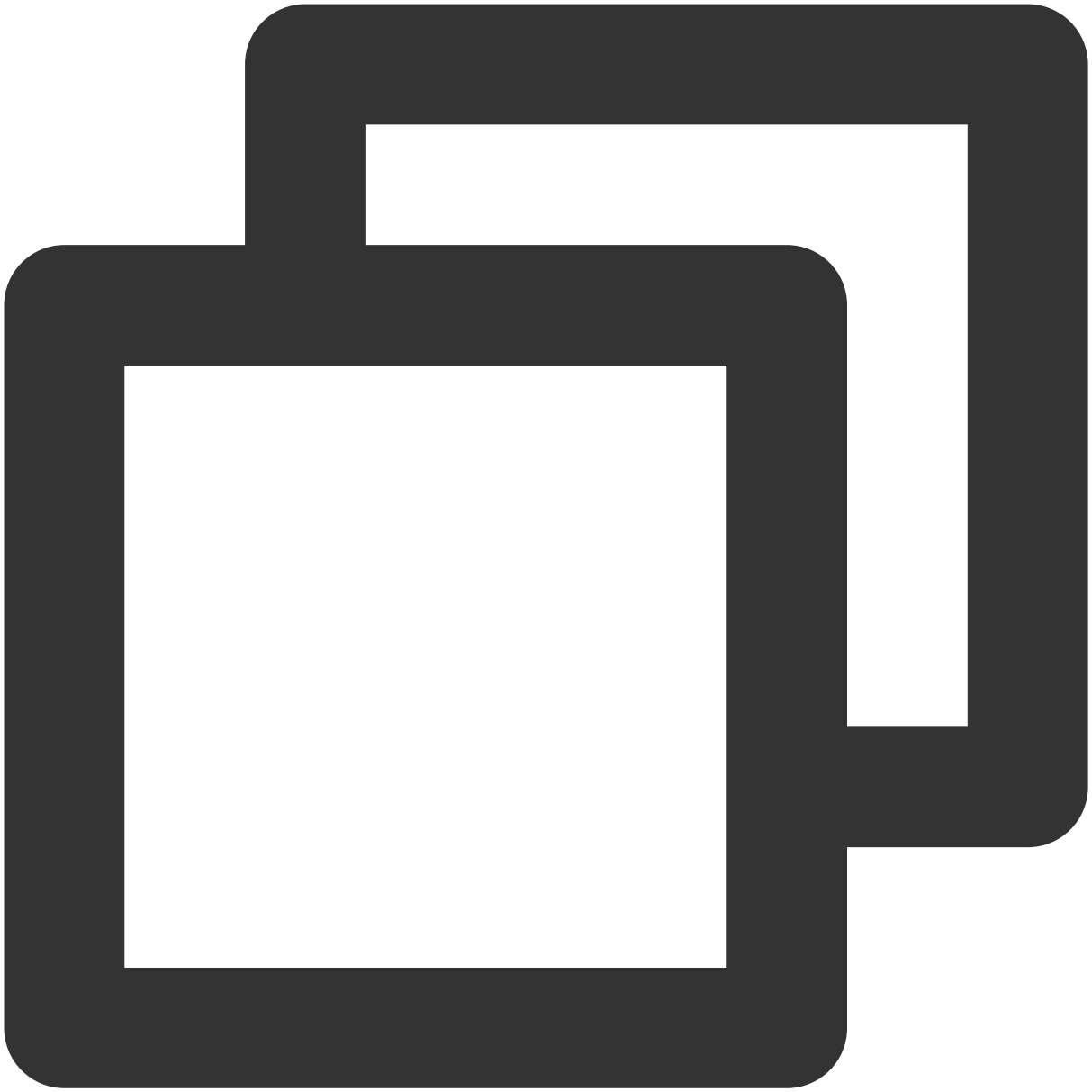
限制 vpcid（vpc:requester_vpc）

条件键 vpc:requester_vpc

使用条件键 `vpc:requester_vpc` 限制用户访问的 vpcid，关于 vpcid 的更多介绍，请参见腾讯云产品 私有网络。

示例：限制 vpcid 为 aqp5jrc1

以下策略示例描述为：允许属于主账号 ID 为100000000001（APPID 为1250000000）下的子账号 ID 100000000002访问存储桶 examplebucket-1250000000，在 vpcid 为 aqp5jrc1 时请求获得授权。



```
{
  "statement": [
    {
      "action": [
        "name/cos:*"
      ],
      "condition": {
        "string_equal": {
          "vpc:requester_vpc": [
            "vpc-aqp5jrc1"
          ]
```

```
        }
      },
      "effect": "allow",
      "principal": {
        "qcs": [
          "qcs::cam::uin/100000000001:uin/100000000002"
        ]
      },
      "resource": [
        "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/*"
      ]
    }
  ],
  "version": "2.0"
}
```

只允许访问对象的最新版本或者指定版本（cos:versionid）

请求参数 versionid

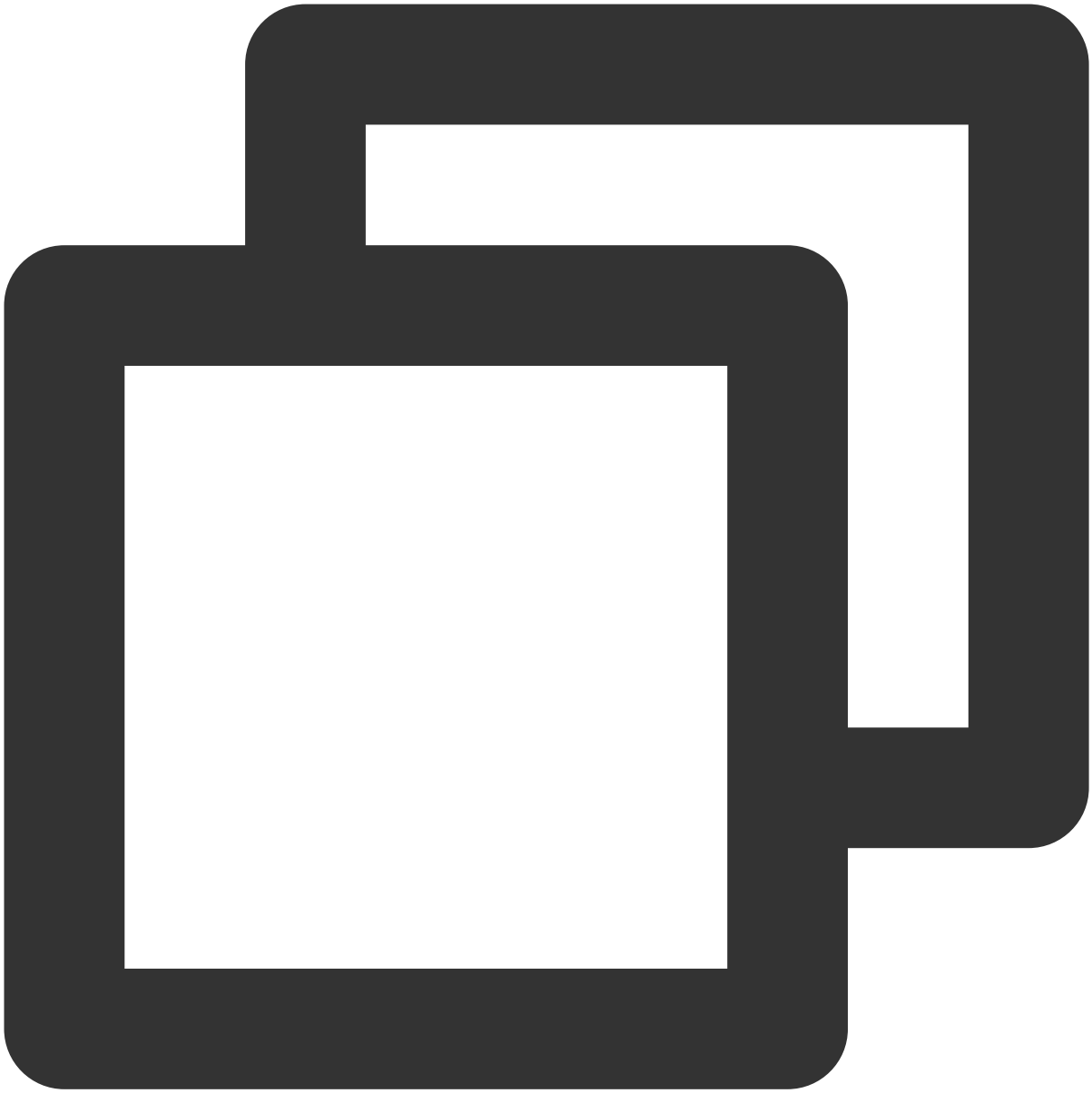
请求参数 `versionid` 表示对象的版本号，关于版本控制相关内容可查看 版本控制概述。您可以在下载对象（GetObject）、删除对象（DeleteObject）时使用请求参数 `versionid` 指定需要操作的对象版本。
不带 `versionid` 请求参数时，请求默认作用于对象的最新版本。
`versionid` 请求参数为一个空字符串时，等同于不带 `versionid` 请求参数时。
`versionid` 请求参数为字符串 `"null"` 的情况。对于一个存储桶在开启版本控制之前上传的对象，开启版本控制后，这批对象的版本号统一是字符串 `"null"` 。

条件键 cos:versionid

条件键 `cos:versionid` 用于限制请求参数 `versionid` 。

示例1：只允许用户获取指定版本号的对象

假设主账号（uin:100000000001）拥有存储桶 examplebucket-1250000000，需要向其子用户（uin:100000000002）进行授权，仅允许子用户获取指定版本号的对象。
采用以下存储桶策略后，子用户（uin:100000000002）发起下载对象请求时，只有在携带了 versionid 参数，且 versionid 的值为版本号 “MTg0NDUxNTc1NjIzMTQ1MDAwODg” 时，请求才会成功。



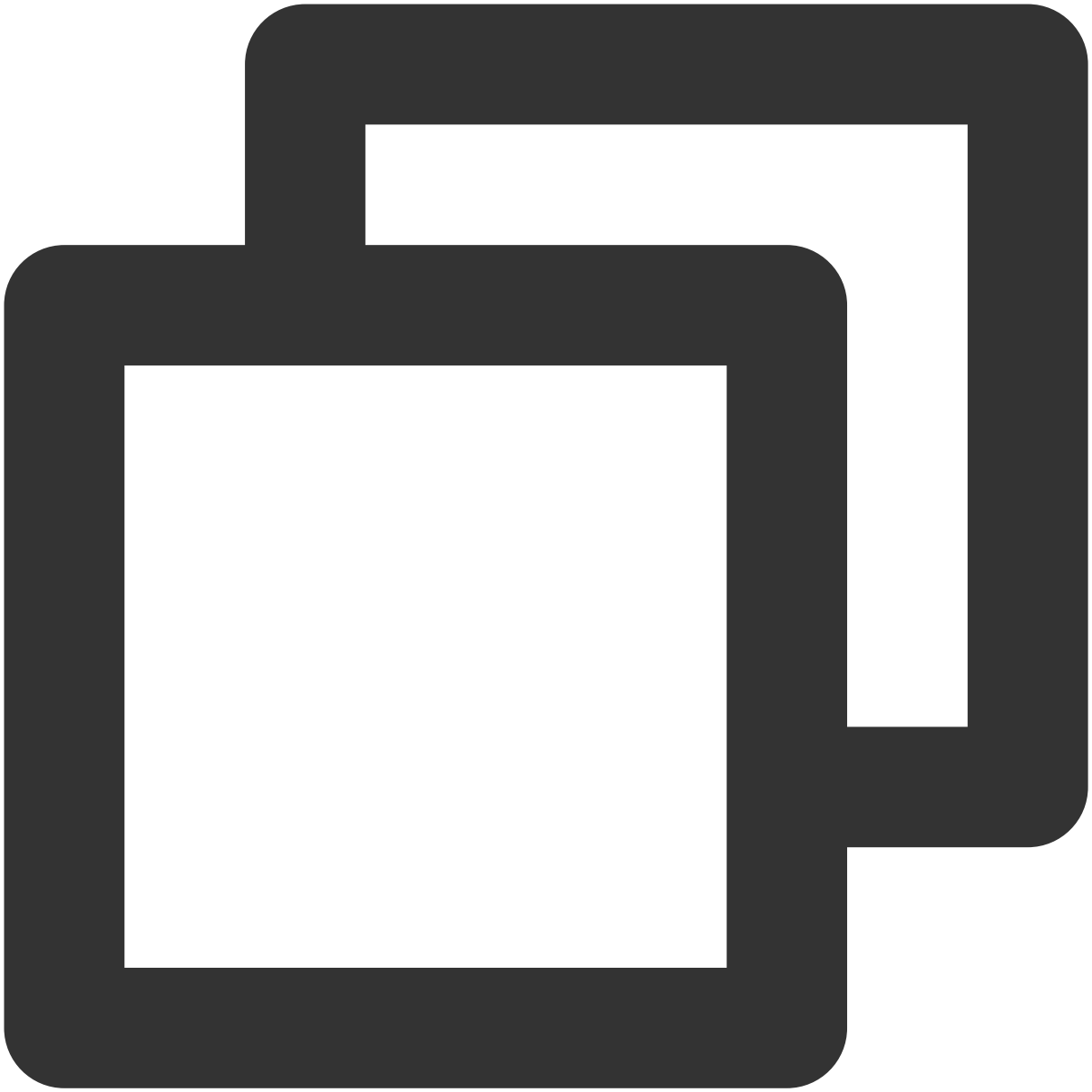
```
{
    "version":"2.0",
    "statement":[
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"allow",
            "action":[
```

```
                "name/cos:GetObject"
            ],
            "condition":{
                "string_equal":{
                    "cos:versionid":"MTg0NDUxNTc1NjIzMTQ1MDAwODg"
                }
            },
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ]
        }
    ]
}
```

添加显式拒绝

当使用上述策略为子用户授予权限时，由于子用户可能通过其他途径获得没有任何条件的相同权限，范围更大的授权策略生效。例如，若子用户处于某个用户组中，主账户为用户组赋予了 GetObject 权限而没有附加任何条件，以上策略对版本号的限制将不起作用。

为了应对这种情况，您可以在上述策略的基础上，通过添加显式拒绝的策略（deny）来完成更严格的权限限制。下面这个 deny 策略的含义是，子用户发起下载对象请求时，若没有携带 versionid 参数，或 versionid 的版本号不是“MTg0NDUxNTc1NjIzMTQ1MDAwODg”，这个请求将被拒绝。由于 deny 的优先级高于其他策略，因此添加显式拒绝可以最高程度的避免权限漏洞。



```
{
    "statement":[
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"allow",
            "action":[
                "name/cos:GetObject"
```

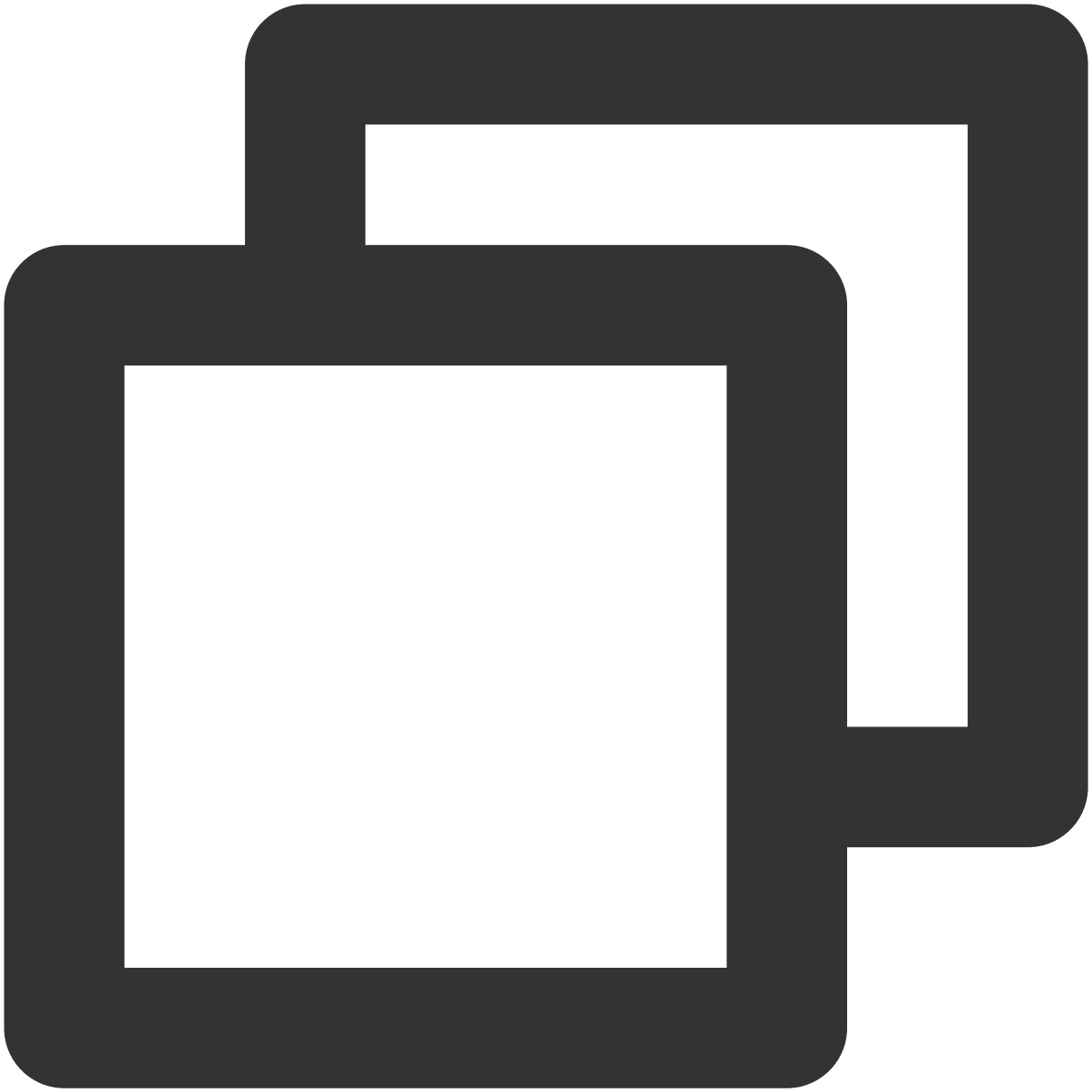
```
            ],
            "condition":{
                "string_equal":{
                    "cos:versionid":"MTg0NDUxNTc1NjIzMTQ1MDAwODg"
                }
            },
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ]
        },
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"deny",
            "action":[
                "name/cos:GetObject"
            ],
            "condition":{
                "string_not_equal_if_exist":{
                    "cos:versionid":"MTg0NDUxNTc1NjIzMTQ1MDAwODg"
                }
            },
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ]
        }
    ],
    "version":"2.0"
}
```

示例2：只允许用户获取最新版本的对象

假设主账号（uin:100000000001）拥有存储桶 examplebucket-1250000000，需要限制其子用户（uin:100000000002）只能获取最新版本的对象。

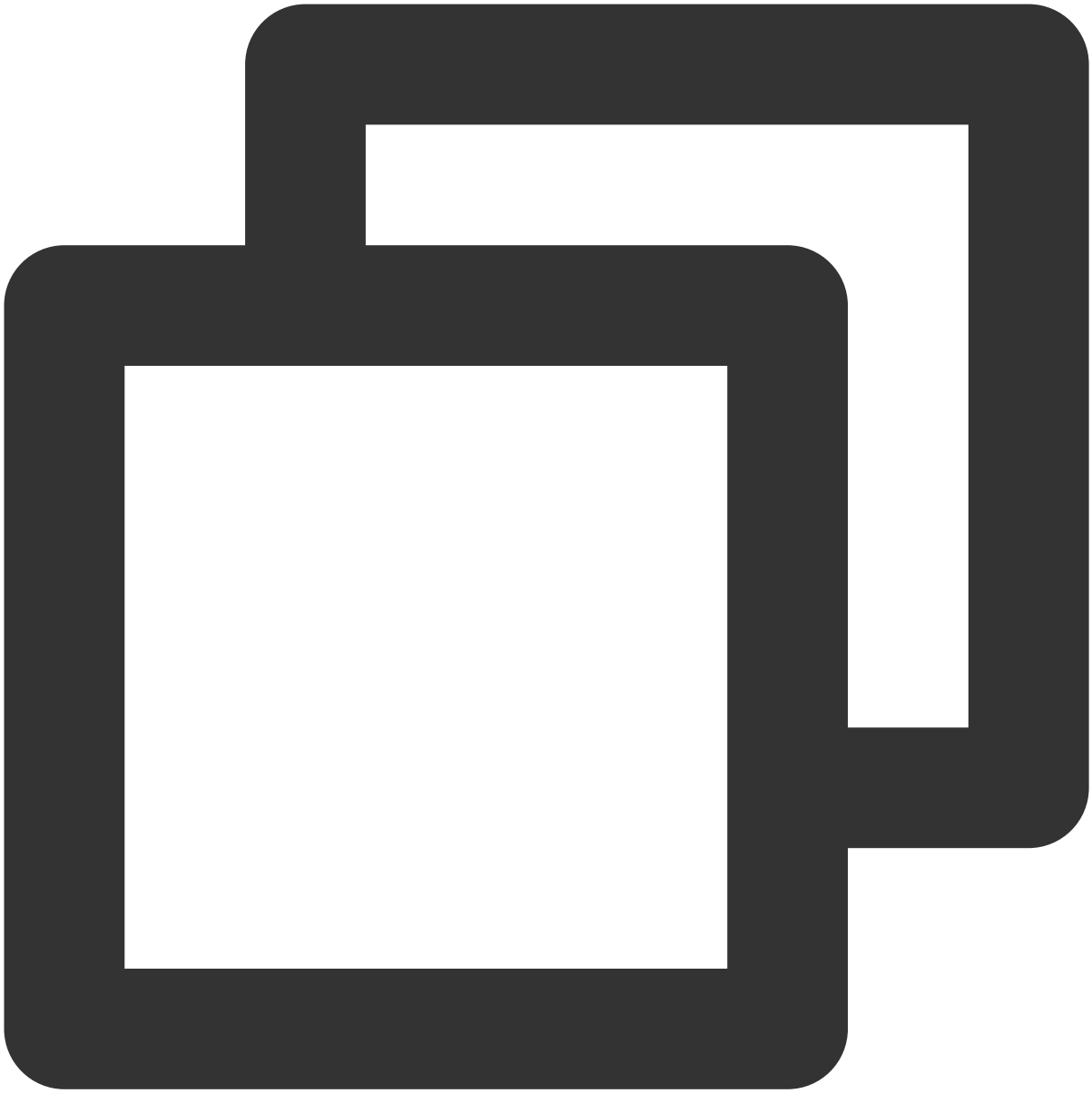
由于在不携带请求参数 `versionid` 或 `versionid` 为空字符串时，GetObject 默认获取最新版本的对象。因此，我们可以在条件中使用 string_equal_if_exsit：

1. 若不携带 versionid，默认按照 true 处理，命中 allow 条件，请求将被 allow。
2. 若请求参数 versionid 为空，即 `“”`，同样会命中 allow 策略，只对获取最新版本的对象的请求进行授权。



```
"condition": {
    "string_equal_if_exist": {
        "cos:versionid": ""
    }
}
```

添加显式拒绝后，完整的存储桶策略如下所示：



```
{
 "statement":[
     {
         "principal":{
             "qcs":[
                 "qcs::cam::uin/100000000001:uin/100000000002"
             ]
         },
         "effect":"allow",
         "action":[
             "name/cos:GetObject"
```

```
            ],
            "condition":{
                "string_equal_if_exist":{
                    "cos:versionid":""
                }
            },
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ]
        },
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"deny",
            "action":[
                "name/cos:GetObject"
            ],
            "condition":{
                "string_not_equal":{
                    "cos:versionid":""
                }
            },
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ]
        }
    ],
    "version":"2.0"
}
```

示例3：不允许用户删除开启版本控制前上传的对象

在开启版本控制前，您的存储桶中可能已上传了一部分对象，这些对象的版本号为“null”。有时候，您需要对这些对象开启额外的保护，例如，禁止用户对这些对象进行永久删除操作，也就是拒绝带版本号的删除操作。

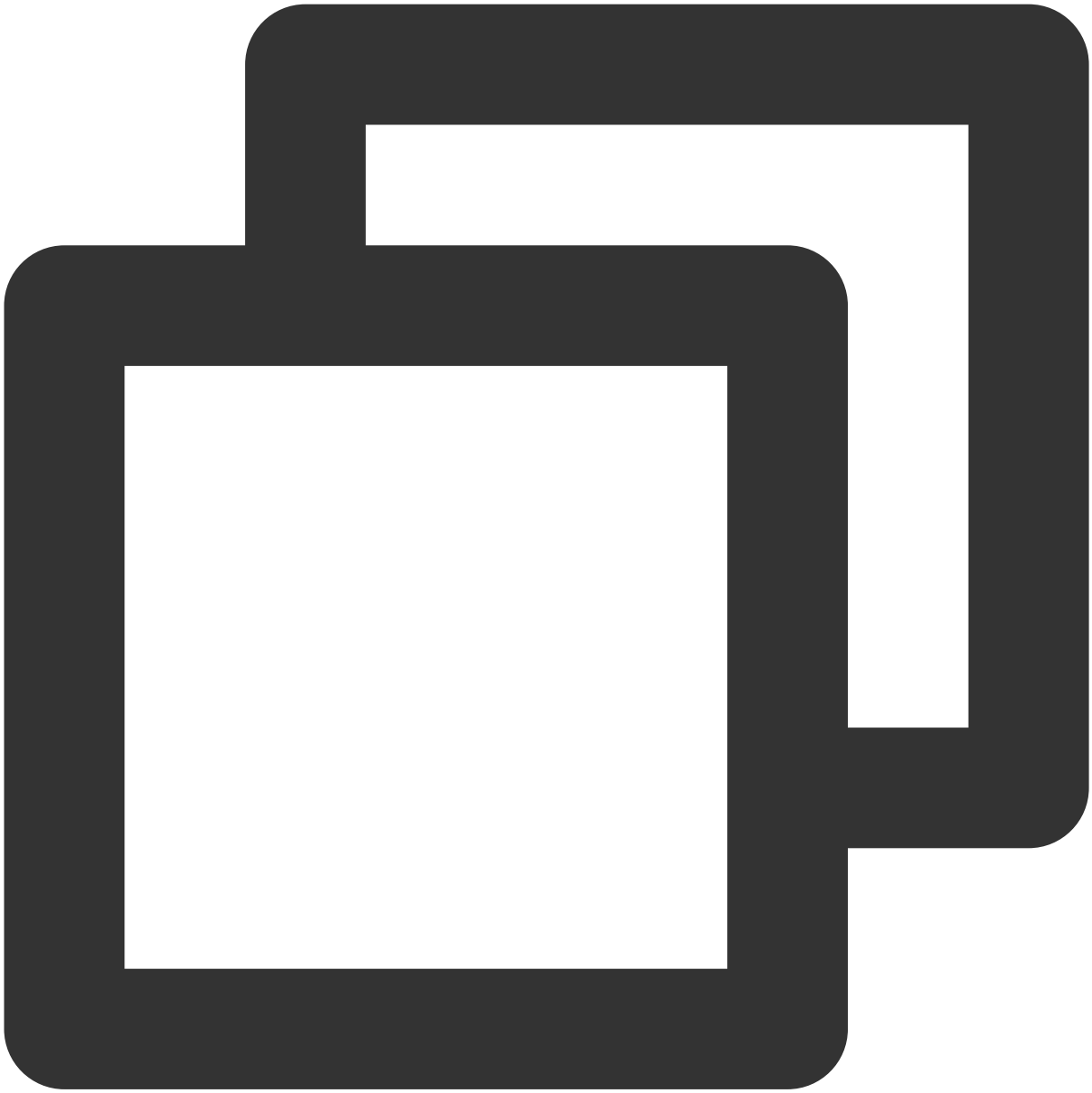
下面这个存储桶策略示例，包含了两个策略：

1. 授权子用户使用 DeleteObject 请求删除存储桶中的对象。

2. 对 DeleteObject 请求的生效条件做了限制。当 DeleteObject 请求携带了请求参数 versionid，且 versionid 为"null"时，拒绝这个 DeleteObject 请求。

由此，若存储桶 examplebucket-1250000000中先上传了对象 A，随后存储桶开启了版本控制，此时对象 A 的版本号为字符串"null"。

添加了这个存储桶策略后，对象 A 将被保护起来。对于子用户针对对象 A 发起的 DeleteObject 请求，若请求不带版本号，由于开启了版本控制，对象 A 本身不会被永久删除，而只会被打上删除标记；若请求携带了 A 的版本号"null"，这个请求将被拒绝，保护对象 A 不被永久删除。



```
{
    "statement":[
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
```

```
            },
            "effect":"allow",
            "action":[
                "name/cos:DeleteObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ]
        },
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"deny",
            "action":[
                "name/cos:DeleteObject"
            ],
            "condition":{
                "string_equal":{
                    "cos:versionid":"null"
                }
            },
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ]
        }
    ],
    "version":"2.0"
}
```

限制上传文件的大小（cos:content-length）

请求头部 Content-Length

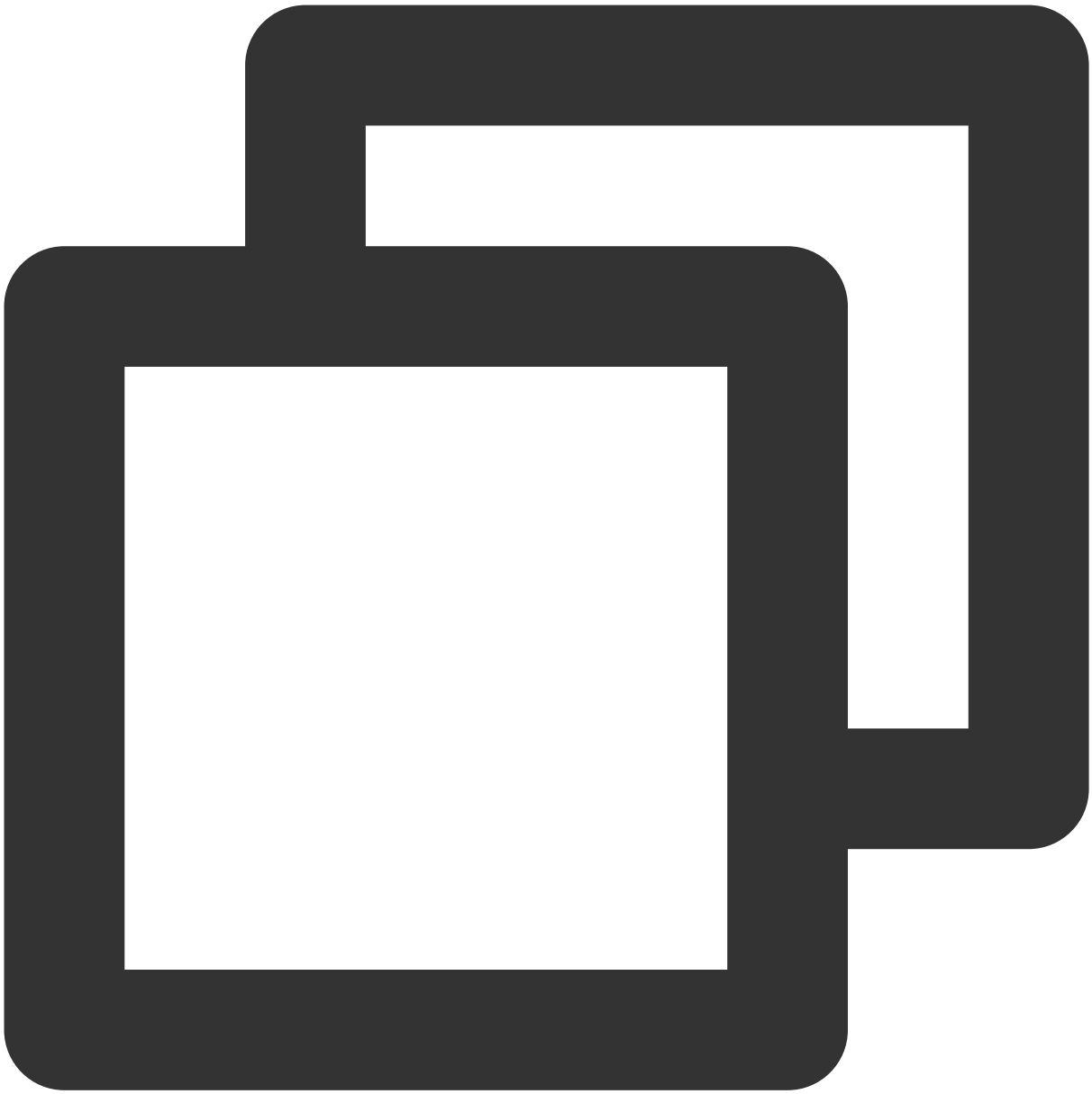
RFC 2616中定义的 HTTP 请求内容长度（字节），在 PUT 和 POST 请求中经常使用。详情见请求头部列表

条件键 cos:content-length

上传对象时，可以通过条件键 `cos:content-length` 限制请求头部 `Content-Length`，进而限制上传对象的文件大小，以方便您更加灵活管理存储空间，避免上传过大、过小文件浪费存储空间与网络带宽。
在下面两个示例中，假设主账号（uin:100000000001）拥有存储桶 examplebucket-1250000000，可以通过 `cos:content-length` 条件键限制子用户（uin:100000000002）上传请求的 Content-Length 头的大小。

示例1: 限制请求头部 Content-Length 的最大值

限制 PutObject 和 PostObject 上传请求必须携带 Content-Length 头部，且这个头部的值不得大于10字节。

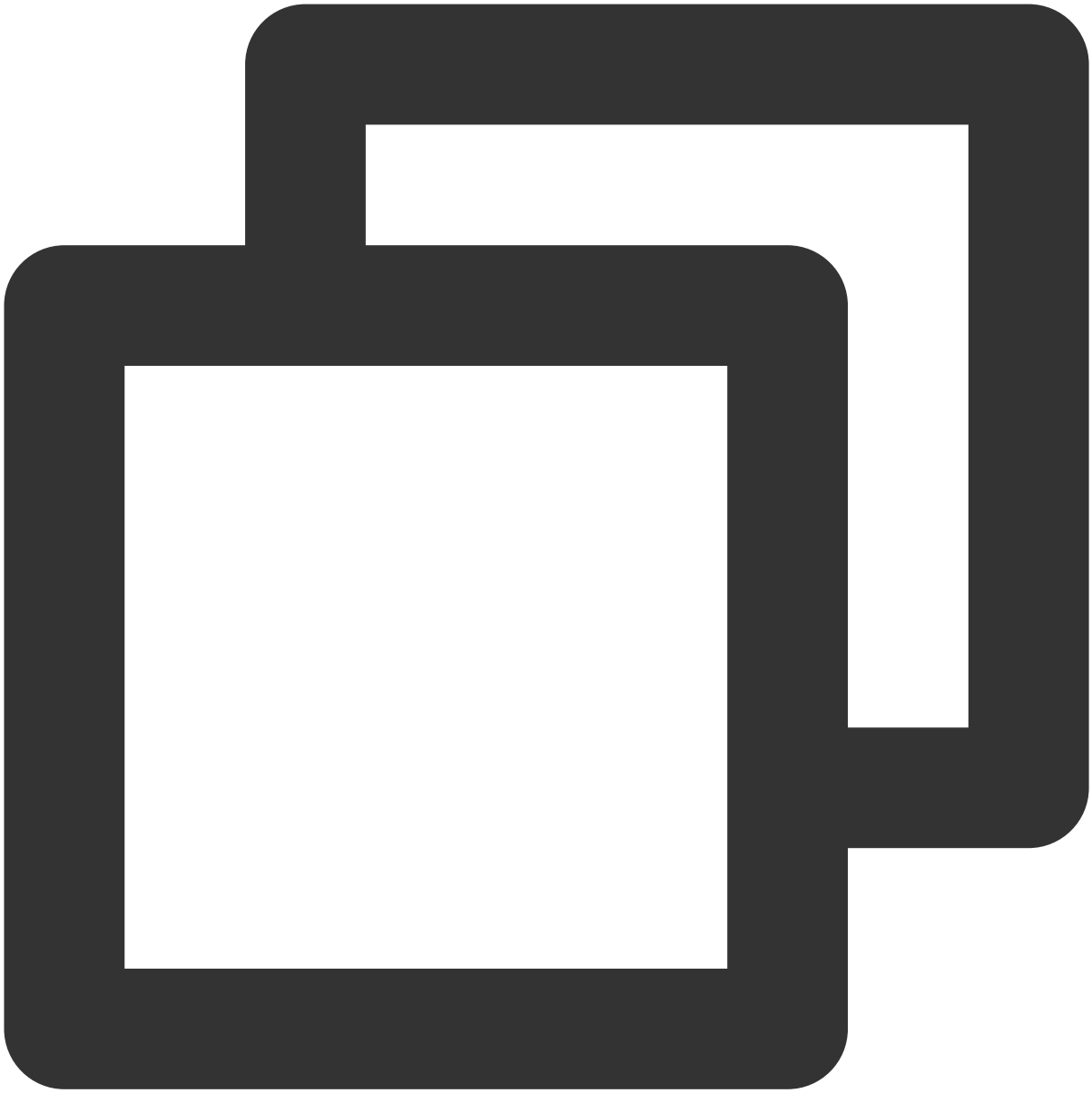


```
{
    "version":"2.0",
    "statement":[
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
```

```
            "effect":"allow",
            "action":[
                "name/cos:PutObject",
                "name/cos:PostObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "numeric_less_than_equal":{
                    "cos:content-length":10
                }
            }
        },
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"deny",
            "action":[
                "name/cos:PutObject",
                "name/cos:PostObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "numeric_greater_than_if_exist":{
                    "cos:content-length":10
                }
            }
        }
    ]
}
```

示例2: 限制请求头部 Content-Length 的最小值

限制 PutObject 和 PostObject 上传请求的必须携带 Content-Length 头部，且 Content-Length 的值不得小于2字节。



```
{
    "version":"2.0",
    "statement":[
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"allow",
            "action":[
```

```
                "name/cos:PutObject",
                "name/cos:PostObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "numeric_greater_than_equal":{
                    "cos:content-length":2
                }
            }
        },
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"deny",
            "action":[
                "name/cos:PutObject",
                "name/cos:PostObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "numeric_less_than_if_exist":{
                    "cos:content-length":2
                }
            }
        }
    ]
}
```

限制上传文件的类型（cos:content-type）

请求头部 Content-Type

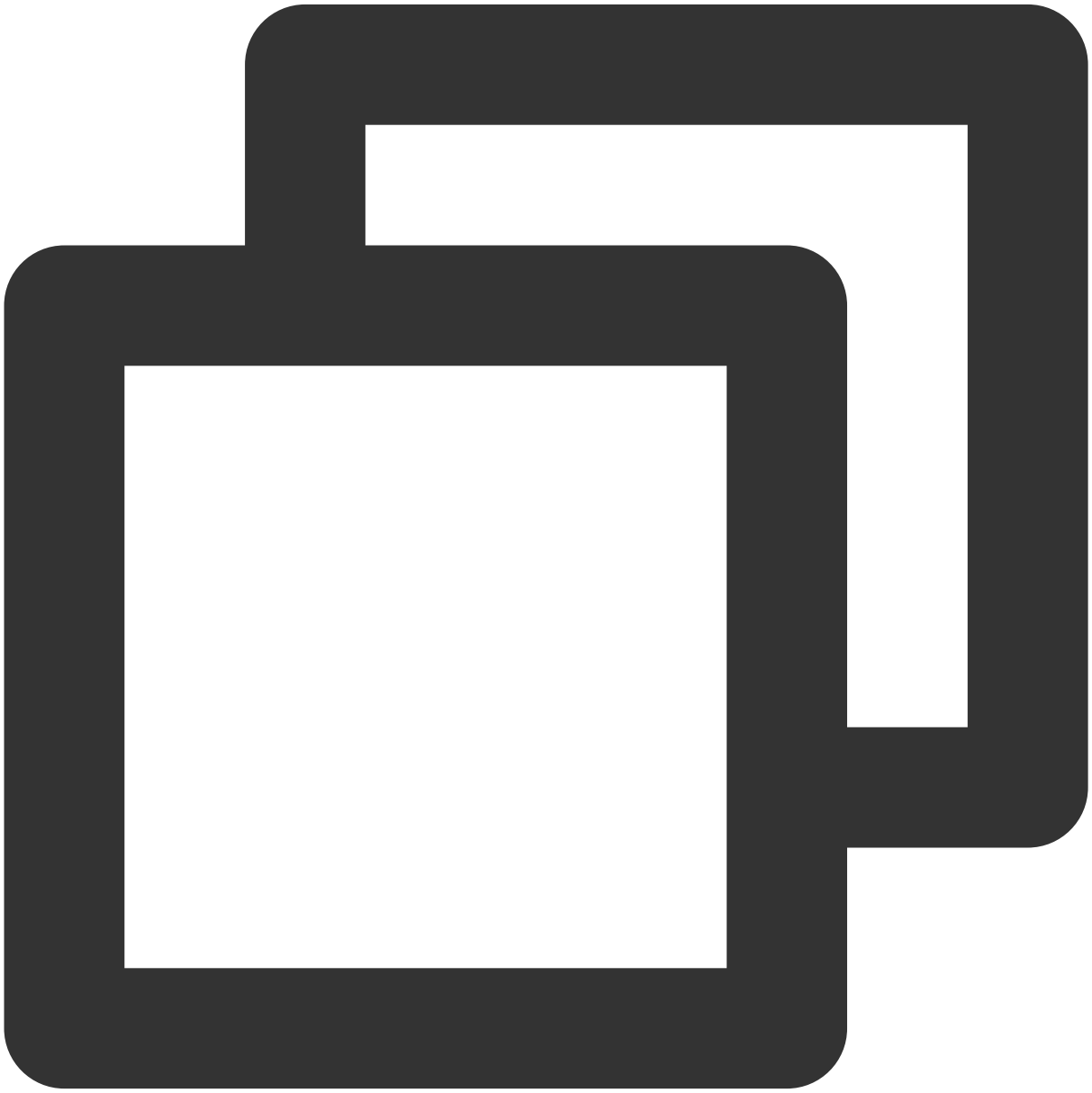
RFC 2616中定义的 HTTP 请求内容类型（MIME），例如 `application/xml` 或 `image/jpeg`，详情请参见请求头部列表。

条件键 cos:content-type

使用条件键 `cos:content-type` 可以对请求的 `Content-Type` 头部进行限制。

示例1: 限定上传对象（PutObject）的 Content-Type 必须为 “image/jpeg”

假设主账号（uin:100000000001）拥有存储桶 examplebucket-1250000000，可以通过 `cos:content-type` 条件键限制子用户（uin:100000000002）上传请求的 Content-Type 头的具体内容。
下面这个存储桶策略的含义是：限制使用 PutObject 上传对象必须携带 Content-Type 头部，且 Content-Type 的值为“image/jpeg”。
需要注意的是，string_equal 要求请求必须携带 Content-Type 头部，且 Content-Type 的值必须与规定值完全一致。在实际请求中，您需要**明确指定请求的 Content-Type 头部**。否则，当您的请求不携带 Content-Type 头部时，请求将会失败；此外，当您使用某些工具发起请求，并未明确指定 Content-Type 时，工具可能会为您自动添加不符合预期的 Content-Type 头部，也可能导致请求失败。



```
{
```

```
    "version":"2.0",
    "statement":[
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"allow",
            "action":[
                "name/cos:PutObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "string_equal":{
                    "cos:content-type":"image/jpeg"
                }
            }
        },
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"deny",
            "action":[
                "name/cos:PutObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "string_not_equal_if_exist":{
                    "cos:content-type":"image/jpeg"
                }
            }
        }
    ]
}
```

限制下载请求返回的文件类型（cos:response-content-type）

请求参数 response-content-type

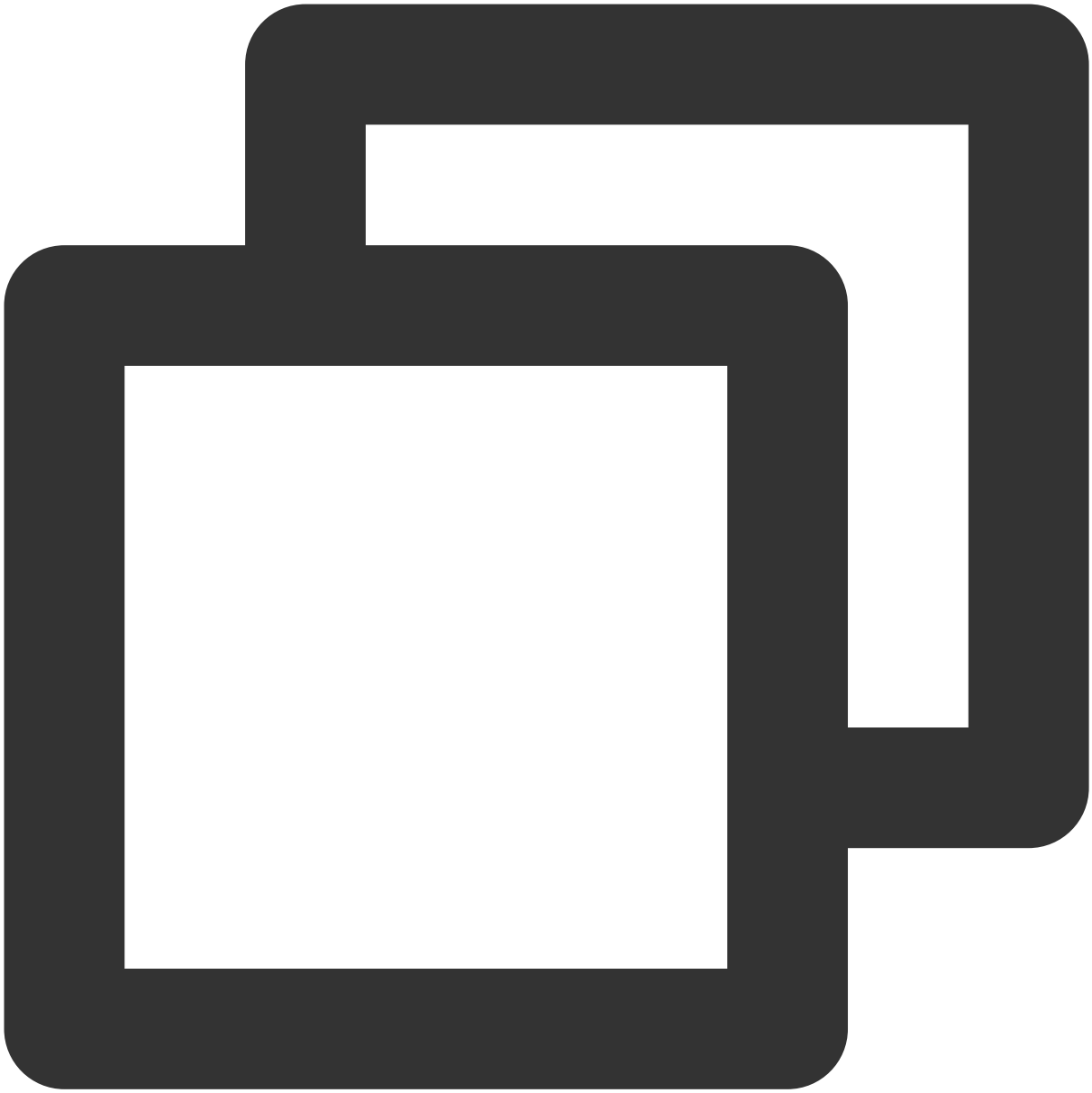
GetObject 接口支持加入请求参数 `response-content-type`，用于设置响应中 Content-Type 头部的值。

条件键 cos:response-content-type

使用条件键 `cos:response-content-type`，您可以对请求的是否必须携带请求参数 `response-content-type` 参数值做限制。

示例1：限制 Get Object 的请求参数 response-content-type 必须为 “image/jpeg”

假设主账号（uin:100000000001）拥有存储桶 examplebucket-1250000000，以下存储桶策略的含义是，限制子用户（uin:100000000002）的 Get Object 请求必须携带请求参数 response-content-type，且请求参数值必须为“image/jpeg”,由于 `response-content-type` 是请求参数，发起请求时需要经过 urlencode，即 `response-content-type=image%2Fjpeg` ,所以在设置 Policy 时，“image/jpeg”也需要经过 urlencode 填写"image%2Fjpeg"。



```
{
    "version":"2.0",
    "statement":[
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"allow",
            "action":[
```

```
                "name/cos:GetObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "string_equal":{
                    "cos:response-content-type":"image%2Fjpeg"
                }
            }
        },
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"deny",
            "action":[
                "name/cos:GetObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "string_not_equal_if_exist":{
                    "cos:response-content-type":"image%2Fjpeg"
                }
            }
        }
    ]
}
```

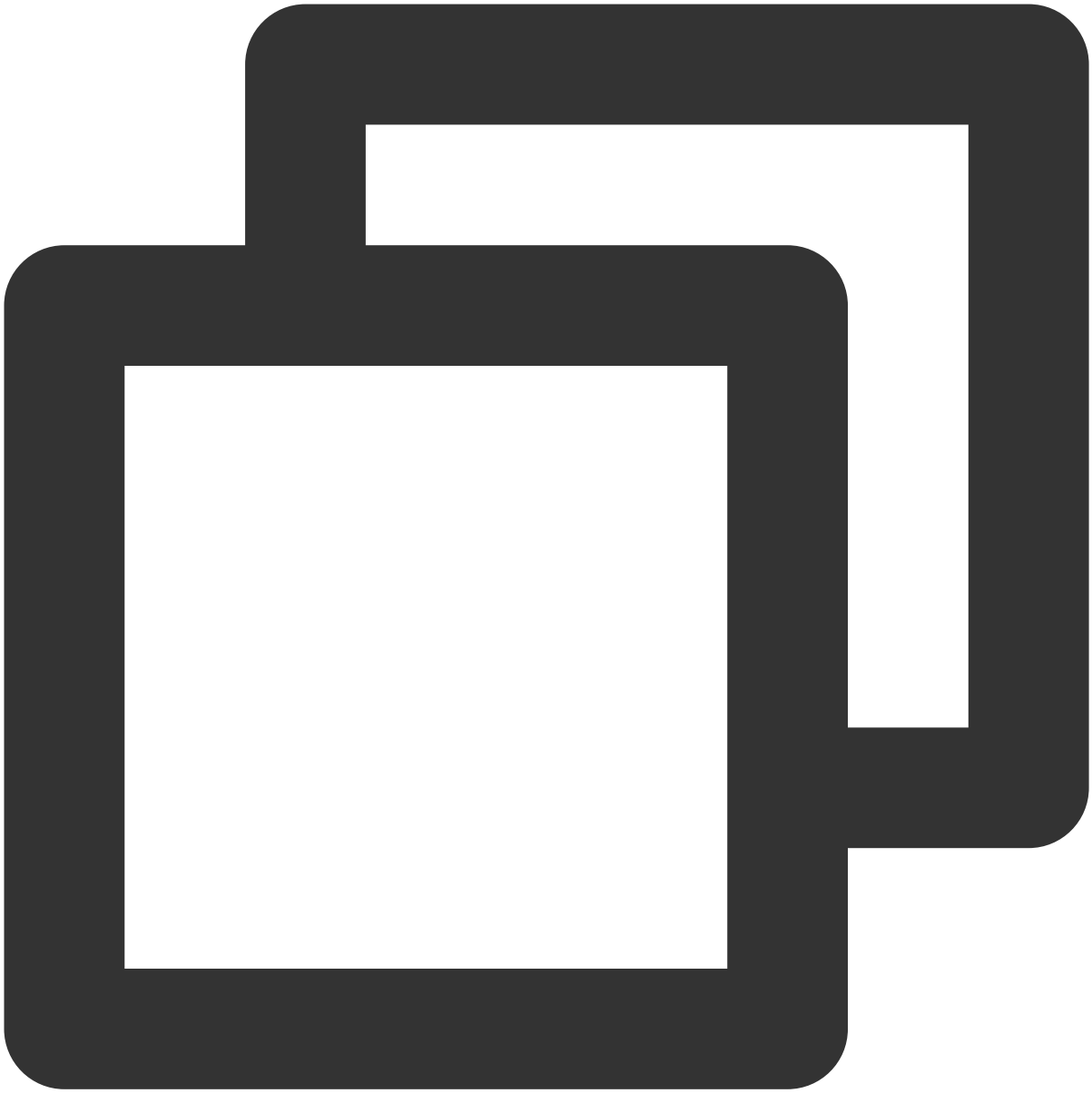
只允许使用了 HTTPS 协议的请求通过（cos:secure-transport）

条件键 cos:secure-transport

您可以使用条件键 `cos:secure-transport` 限制请求必须使用 HTTPS 协议

示例1：下载请求需要使用 HTTPS 协议

假设主账号（uin:100000000001）拥有存储桶 examplebucket-1250000000，以下存储桶策略的含义表示仅对由子用户（uin:100000000002）使用了 HTTPS 协议的 GetObject 请求进行授权。

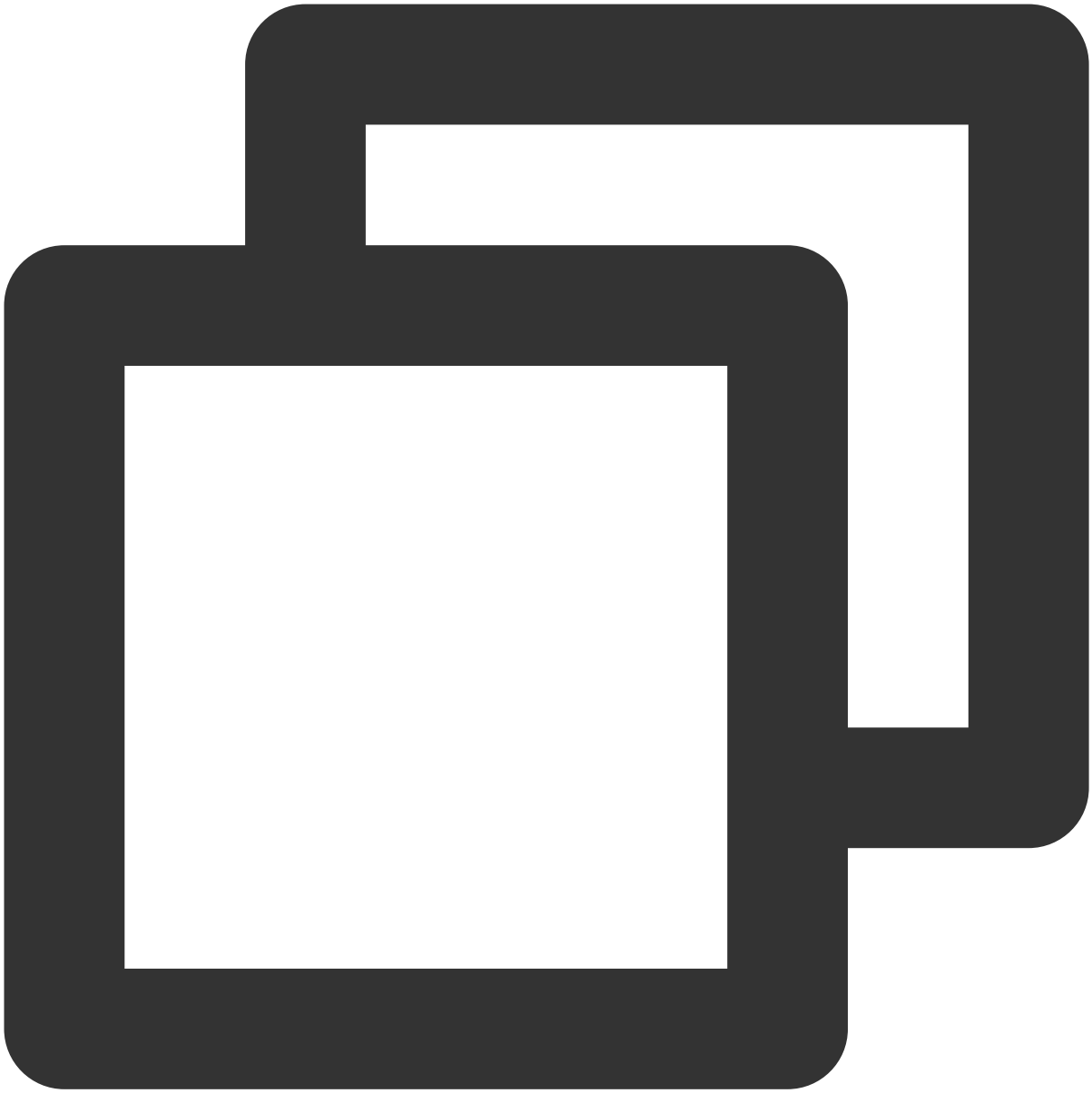


```
{
    "version":"2.0",
    "statement":[
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"allow",
            "action":[
```

```
                "name/cos:GetObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "bool_equal":{
                    "cos:secure-transport":"true"
                }
            }
        }
    ]
}
```

示例2：拒绝任何不使用 HTTPS 协议的请求

假设主账号（uin:100000000001）拥有存储桶 examplebucket-1250000000，以下存储桶策略的含义表示拒绝子用户（uin:100000000002）任何不使用 HTTPS 协议的请求。



```
{
    "version":"2.0",
    "statement":[
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"deny",
            "action":[
```

```
                "*"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "bool_equal":{
                    "cos:secure-transport":"false"
                }
            }
        }
    ]
}
```

只允许设置指定的存储类型（cos:x-cos-storage-class）

请求头部 x-cos-storage-class

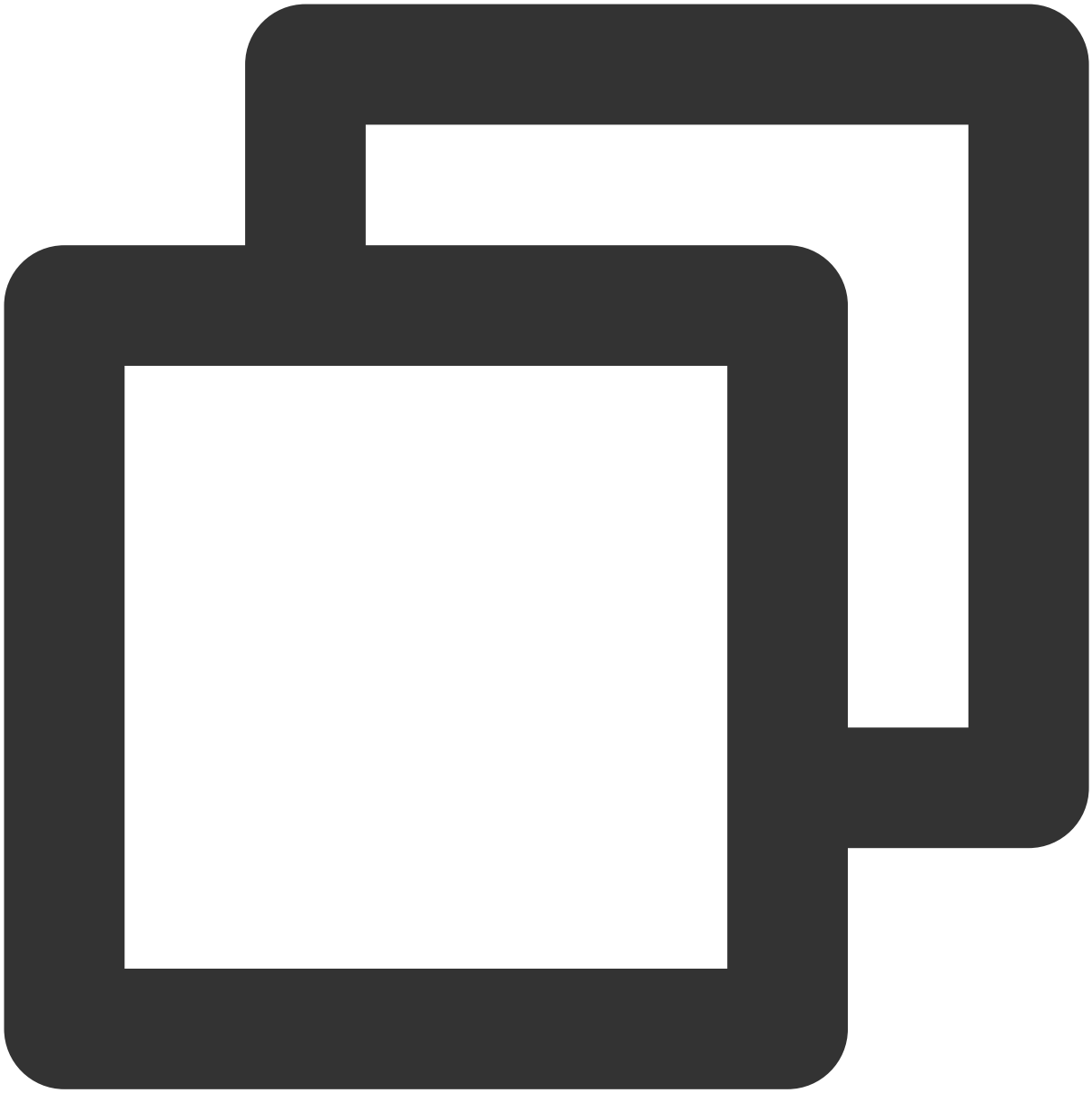
用户可以通过请求头部 `x-cos-storage-class` 在上传对象时指定存储类型或修改对象的存储类型。

条件键 cos:x-cos-storage-class

您可以通过条件键 `cos:x-cos-storage-class` 限制请求头部 `x-cos-storage-class`，进而限制可能修改存储类型的请求。
COS 的存储类型字段包括：`STANDARD`、`MAZ_STANDARD`，`STANDARD_IA`、`MAZ_STANDARD_IA`、`INTELLIGENT_TIERING`、`MAZ_INTELLIGENT_TIERING`、`ARCHIVE`、`DEEP_ARCHIVE`。

示例1：要求 PutObject 时必须将存储类型设置为标准类型

假设主账号（uin:100000000001）拥有存储桶 examplebucket-1250000000，通过存储桶策略策略，限制子账户（uin:100000000002）的 PutObject 请求必须携带 x-cos-storage-class 头部，并且头部值为 `STANDARD`。



```
{
    "version":"2.0",
    "statement":[
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"allow",
            "action":[
```

```
                "name/cos:PutObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "string_equal":{
                    "cos:x-cos-storage-class":"STANDARD"
                }
            }
        },
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"deny",
            "action":[
                "name/cos:PutObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "string_not_equal_if_exist":{
                    "cos:x-cos-storage-class":"STANDARD"
                }
            }
        }
    ]
}
```

只允许设置指定的存储桶/对象 ACL（cos:x-cos-acl）

请求头部 x-cos-acl

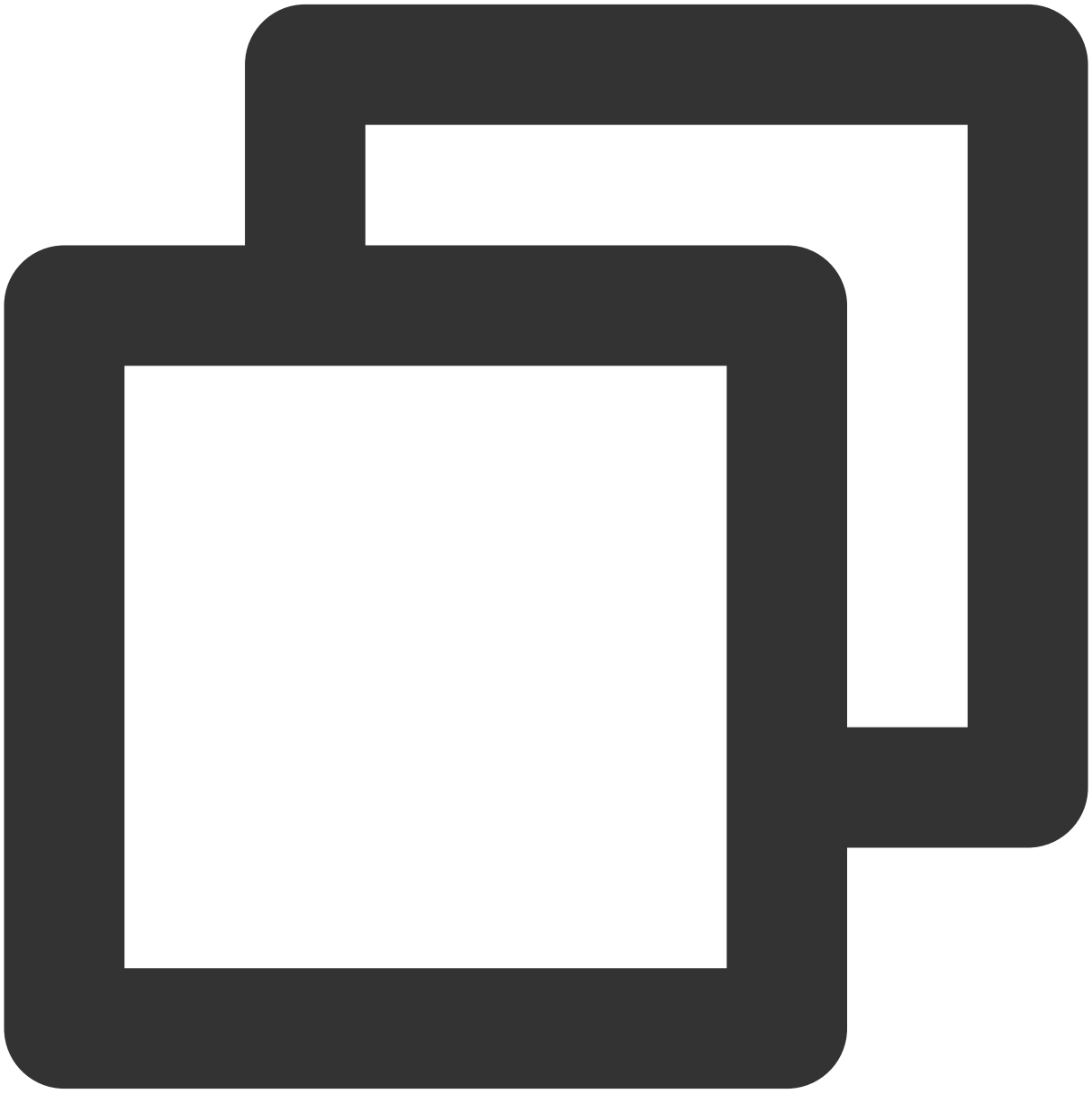
使用请求头部 `x-cos-acl` ，您可以在上传对象、创建存储桶时指定访问控制列表（ACL），或修改对象、存储桶 ACL；关于 ACL 的相关介绍可参见 ACL概述。
存储桶的预设 ACL： `private` 、 `public-read` 、 `public-read-write` 、 `authenticated-read` 。
对象的预设 ACL： `default` 、 `private` 、 `public-read` 、 `authenticated-read` 、 `bucket-owner-read` 、 `bucket-owner-full-control` 。

条件键 cos:x-cos-acl

您可以通过条件键 `cos:x-cos-acl` 限制请求的头部 `x-cos-acl`，进而限制可能修改对象或存储桶 ACL 的请求。

示例1：PutObject 时必须同时将对象的 ACL 设置为私有的

假设主账号（uin:100000000001）拥有存储桶 examplebucket-1250000000，需要限制子用户（uin:100000000002）只能上传私有对象。以下策略要求发起 PutObject 请求必须携带 x-cos-acl 头部，并且头部值为 `private` 。



```
{
    "version":"2.0",
```

```
    "statement":[
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"allow",
            "action":[
                "name/cos:PutObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "string_equal":{
                    "cos:x-cos-acl":"private"
                }
            }
        },
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"deny",
            "action":[
                "name/cos:PutObject"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "string_not_equal_if_exist":{
                    "cos:x-cos-acl":"private"
                }
            }
        }
    ]
}
```

只允许列出指定目录下的对象（cos:prefix）

条件键 cos:prefix

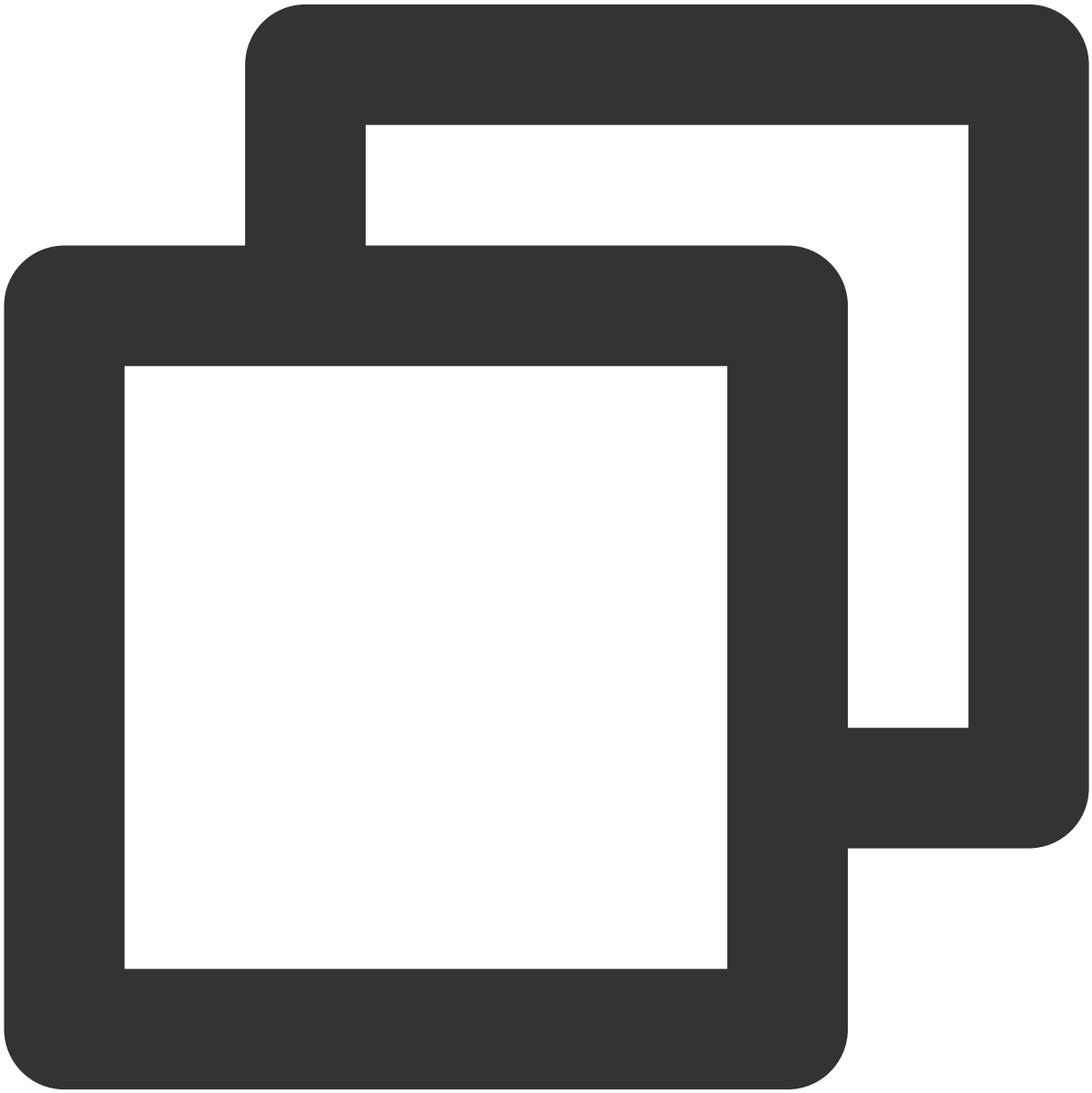
您可以通过条件键 `cos:prefix` 限制请求参数 prefix。

注意

prefix 的值若为特殊字符（中文、 `/` 等），写入存储桶策略前需要先经过 urlencode。

示例1：只允许列出存储桶指定目录下的对象

假设主账号（uin:100000000001）拥有存储桶 examplebucket-1250000000，需要限制子用户（uin:100000000002）只可列出存储桶 folder1 目录下的对象。以下存储桶策略规定，子用户发起 GetBucket 请求必须携带 prefix 参数，且值为“folder1”。



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

```
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"allow",
            "action":[
                "name/cos:GetBucket"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "string_equal":{
                    "cos:prefix":"folder1"
                }
            }
        },
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"deny",
            "action":[
                "name/cos:GetBucket"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "string_equal_if_exist":{
                    "cos:prefix":"folder1"
                }
            }
        }
    ],
    "version":"2.0"
}
```

只允许使用指定版本的 TLS 协议（cos:tls-version）

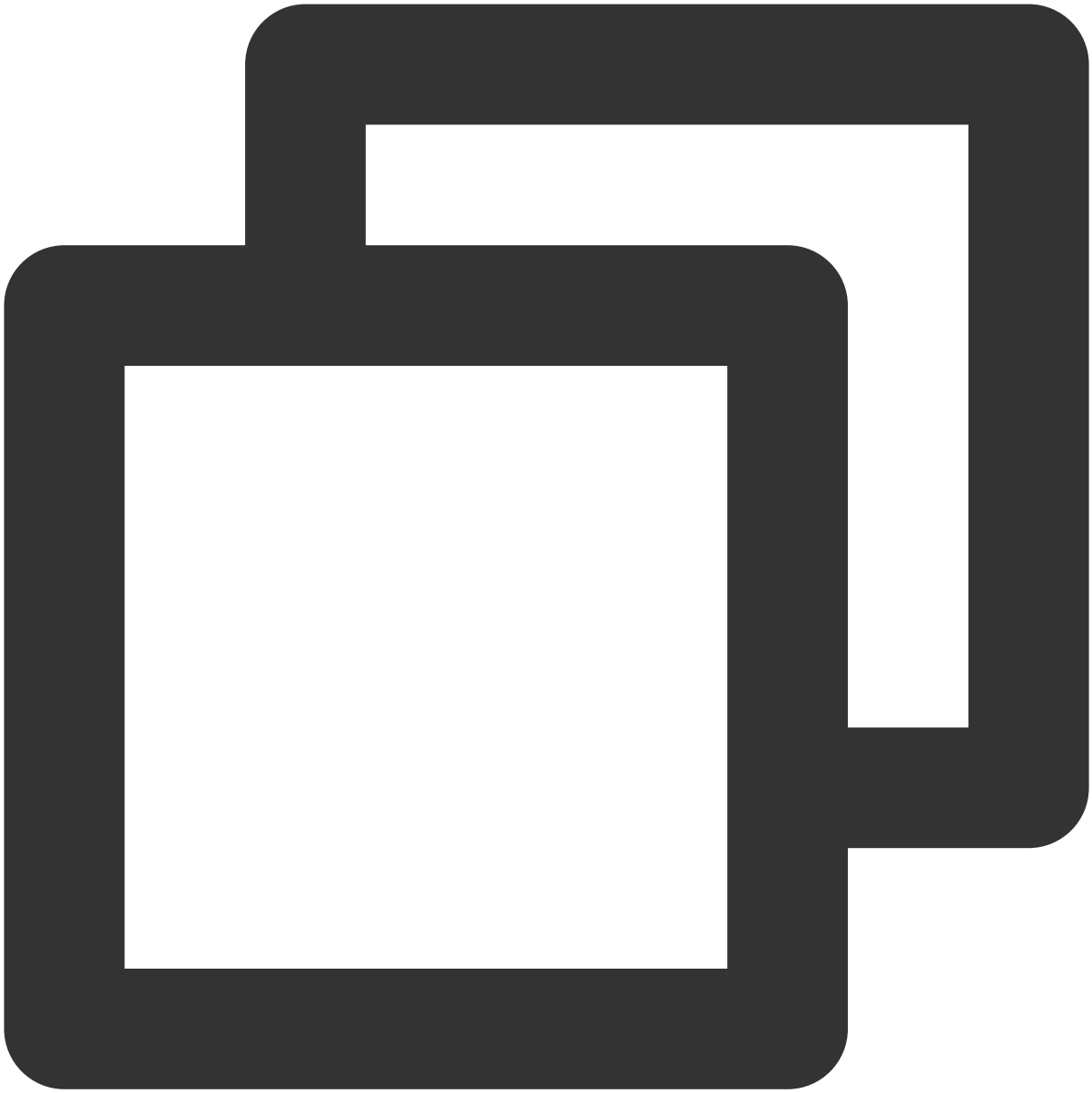
条件键 cos:tls-version

您可以通过条件键 `cos:tls-version` 限制 HTTPS 请求的 TLS 版本，该条件键为 Numric 类型，支持输入浮点数，例如 1.0、1.1、1.2 等。

示例1：仅对 TLS 协议版本为1.2的 HTTPS 请求进行授权

请求场景	预期
HTTPS 请求，TLS 版本为1.0	403，失败
HTTPS 请求，TLS 版本为1.2	200，成功

策略示例如下：



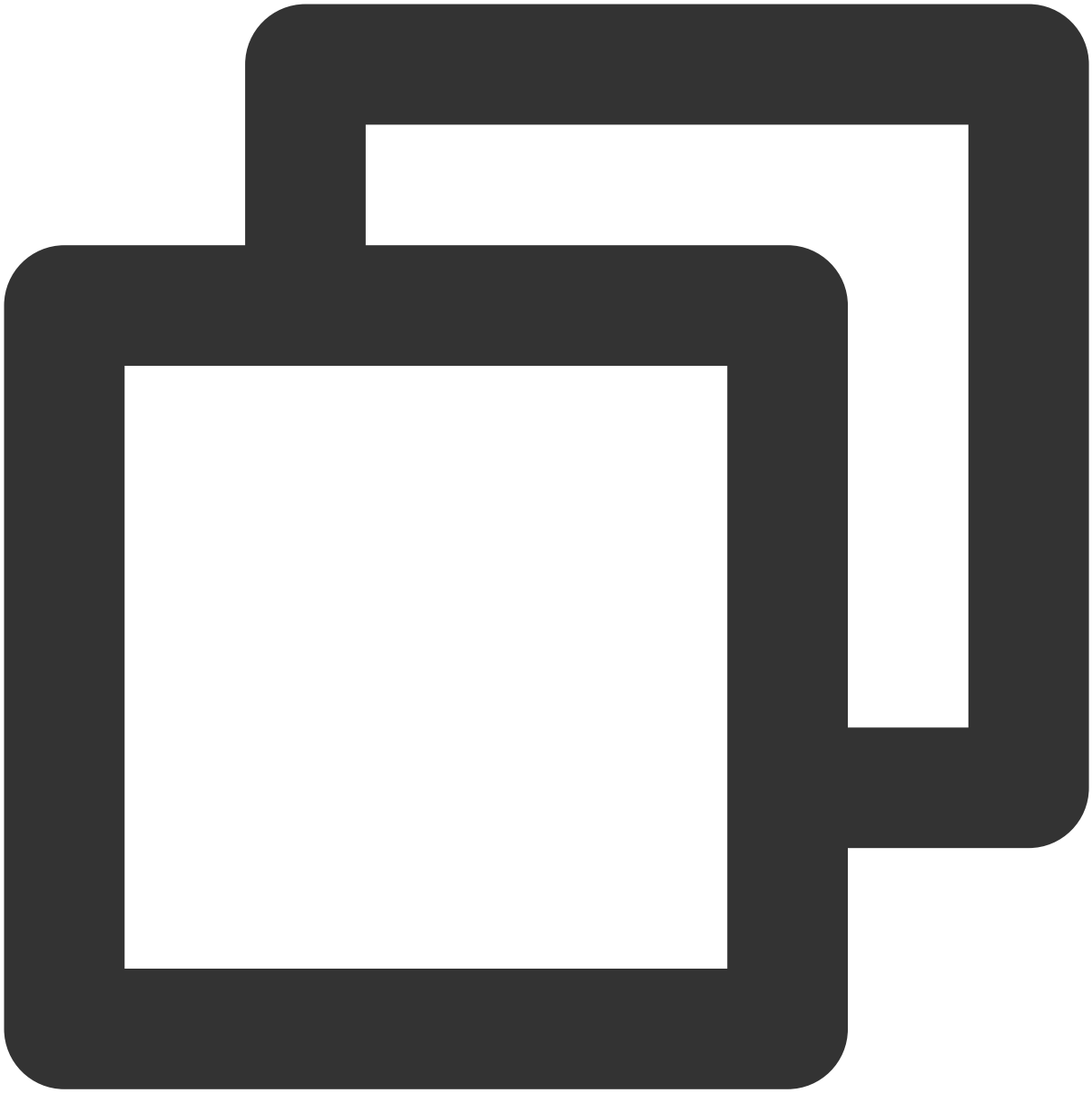
```
{
    "version":"2.0",
    "statement":[
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"allow",
            "action":[
```

```
                "*"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "numeric_equal":{
                    "cos:tls-version":1.2
                }
            }
        }
    ]
}
```

示例2：拒绝 TLS 协议版本小于1.2的 HTTPS 请求

请求场景	预期
HTTPS 请求，TLS 版本为1.0	403，失败
HTTPS 请求，TLS 版本为1.2	200，成功

策略示例如下：



```
{
    "version":"2.0",
    "statement":[
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"allow",
            "action":[
```

```
                "*"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "numeric_greater_than_equal":{
                    "cos:tls-version":1.2
                }
            }
        },
        {
            "principal":{
                "qcs":[
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect":"deny",
            "action":[
                "*"
            ],
            "resource":[
                "qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
            ],
            "condition":{
                "numeric_less_than_if_exist":{
                    "cos:tls-version":1.2
                }
            }
        }
    ]
}
```

创建存储桶时强制设置指定存储桶标签（qcs:request_tag）

说明

条件键 request_tag 仅适用于 PutBucket、PutBucketTagging 操作。GetService、PutObject、PutObjectTagging 等操作不支持本条件键。

条件键 qcs:request_tag

您可以通过条件键 `qcs:request_tag` 限制用户发起请求 PutBucket、PutBucketTagging 必须携带指定的存储桶标签。

示例：限制用户创建存储桶时必须携带指定的存储桶标签

许多用户会通过存储桶标签管理存储桶，下面的策略示例为：限制用户只有在创建存储桶时，必须设置指定的存储桶标签 `<a,b>` 和 `<c,d>`，才能获得授权。

存储桶标签可以设置多个，存储桶标签键值、标签数量不同都会作为不同集合。假设用户携带的多个参数值为集合 A，条件规定的多个参数值为集合 B。在使用该条件键，可以通过限定词 for_any_value、for_all_value 的不同组合表示不同的含义。

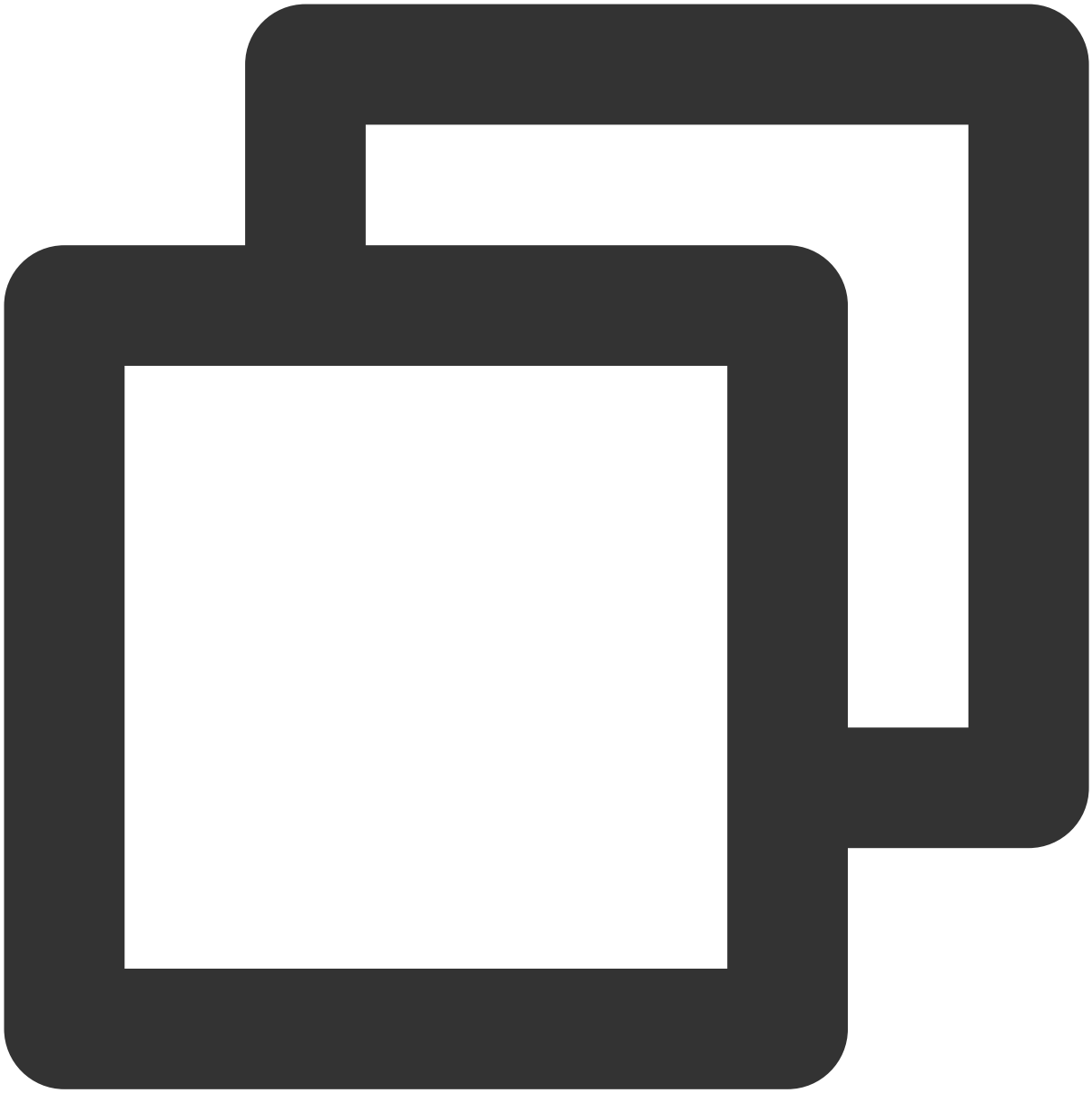
`for_any_value:string_equal` 表示 A 和 B 存在交集时生效。

`for_all_value:string_equal` 表示 A 是 B 的子集时生效。

当使用 `for_any_value:string_equal` 时，对应的策略和请求表现如下：

请求场景	预期
PutBucket，请求头部 `x-cos-tagging: a=b&c=d`	200，成功
PutBucket，请求头部 `x-cos-tagging: a=b`	200，成功
PutBucket，请求头部 `x-cos-tagging: a=b&c=d&e=f`	200，成功

策略示例如下：



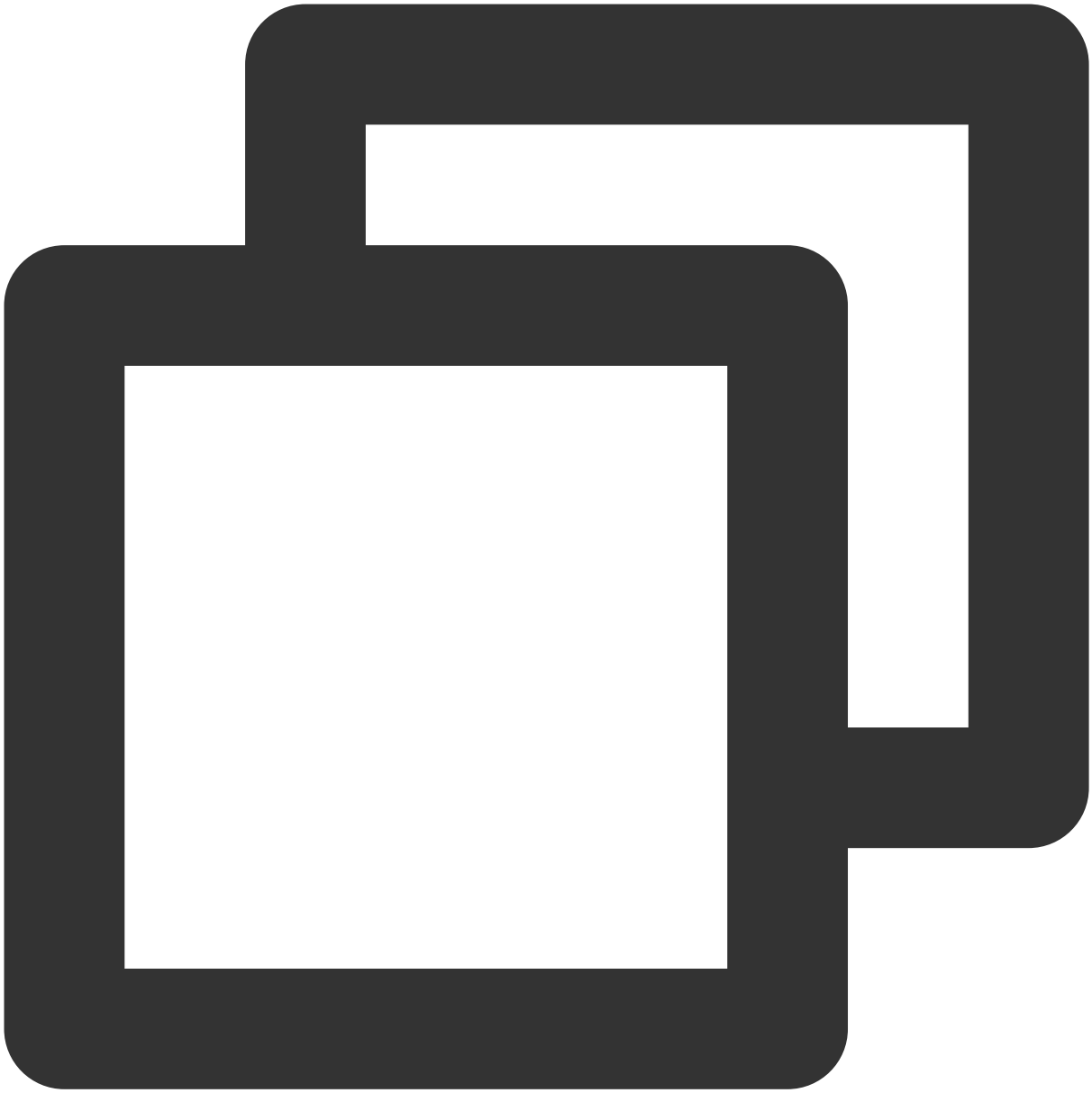
```
{
    "version": "2.0",
    "statement": [
        {
            "principal": {
                "qcs": [
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect": "allow",
            "action": [
```

```
                "name/cos:PutBucket"
            ],
            "resource": "*",
            "condition": {
                "for_any_value:string_equal": {
                    "qcs:request_tag": [
                        "a&b",
                        "c&d"
                    ]
                }
            }
        }
    ]
}
```

当使用 `for_all_value:string_equal` 时，对应的策略和请求表现如下：

请求场景	预期
PutBucket，请求头部 `x-cos-tagging: a=b&c=d`	200，成功
PutBucket，请求头部 `x-cos-tagging: a=b`	200，成功
PutBucket，请求头部 `x-cos-tagging: a=b&c=d&e=f`	403，失败

策略示例如下：



```
{
    "version": "2.0",
    "statement": [
        {
            "principal": {
                "qcs": [
                    "qcs::cam::uin/100000000001:uin/100000000002"
                ]
            },
            "effect": "allow",
            "action": [
```

```
                "name/cos:PutBucket"
            ],
            "resource": "*",
            "condition": {
                "for_all_value:string_equal": {
                    "qcs:request_tag": [
                        "a&b",
                        "c&d"
                    ]
                }
            }
        }
    ]
}
```

授予其他主账号下的子账号操作名下存储桶的权限

最近更新时间：2024-06-03 11:10:25

操作场景

当前主账号 A（APPID 为1250000000）名下有存储桶 `examplebucket1-1250000000` 和 `examplebucket2-1250000000`，另有一主账号 B 及其子账号 B0，B0 由于业务需要，希望能够操作账号 A 名下的两个存储桶，下面将为您介绍如何进行相关的授权操作。

操作步骤

授予主账号 B 操作 A 名下存储桶的权限

1. 使用主账号 A 登录 对象存储控制台。
2. 单击**存储桶列表**，找到需要授权的存储桶，单击其名称进入存储桶详情页。
3. 在左侧导航栏中，单击**权限管理**，进入该存储桶的权限管理页。
4. 找到 **Policy 权限设置**配置项，单击**添加策略**，选择自定义策略，单击**下一步**。
5. 填写如下表单项：

策略ID：可不填。
效力：允许。
用户：单击添加用户，用户类型选择主账号，账号 ID 填写主账号 B 的 UIN，例如100000000002。
资源：根据需要选择，默认为整个存储桶。
资源路径：仅指定资源时需要填写，根据需要填写。
操作：单击添加操作，选择所有操作，如仅需授权部分操作，也可以选择一个或多个实际需要的操作。
条件：根据需要填写，如不需要可留空。

6. 单击**完成**，此时主账号 B 将拥有操作该存储桶的相关权限。
7. 如需授权其他存储桶，可重复上述步骤。

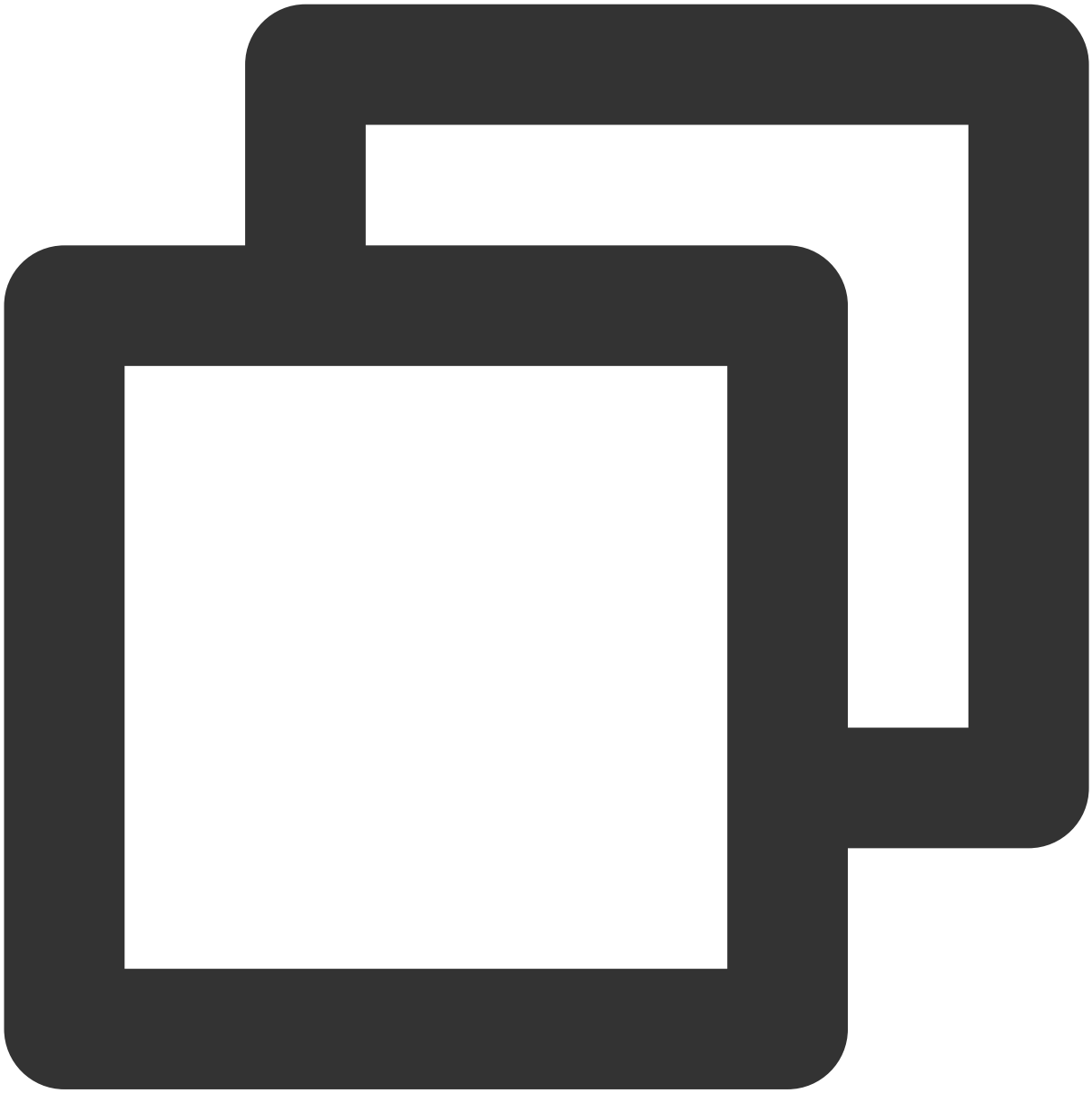
授予子账号 B0 操作 A 名下存储桶的权限

1. 使用主账号 B 登录访问管理 CAM 控制台的 策略 页面。
2. 选择**新建自定义策略 > 按策略语法创建**，随后选择空白模板，单击**下一步**。

说明：
主账号 B 给子账号 B0 授权，只能通过自定义策略授权，不支持通过预设策略授权。

3. 填写如下表单：

策略名称：自行定义一个不重复且有意义的策略名称，例如 cos-child-account。
备注：可选，自行编写。
策略内容：



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

```
                "qcs::cos::uid/1250000000:examplebucket1-1250000000/*",
                "qcs::cos::uid/1250000000:examplebucket2-1250000000/*"
            ]
    }
  ]
}
```

以上策略表示将主账号 B 有权操作的 A 名下的存储桶均授权给子账号 B0。其中，`uid/1250000000` 中的1250000000为主账号 A 的 APPID，`examplebucket1-1250000000` 和 `examplebucket2-1250000000` 为主账号 B 有权操作的 A 名下的存储桶。

4. 单击**完成**，完成策略的创建。
5. 在**策略**列表中找到刚才已创建的策略，并单击右侧的**关联用户/组/角色**。
6. 在弹窗中，勾选子账号 B0，单击**确定**。
7. 完成授权操作，即可使用子账号 B0 的密钥，测试操作 A 名下的存储桶。

性能优化
请求速率与性能优化

最近更新时间：2024-01-06 10:47:50

注意

当前 COS 已经通过底层索引打散机制实现高 QPS，如果有需要更高性能的 QPS 需求，可以 联系我们。在日常组织文件过程中，我们仍然推荐您遵循本文档的要求，避免过于集中的索引存储方式。

简介

本文探讨请求速率性能优化在腾讯云对象存储（Cloud Object Storage，COS）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30000个 PUT 请求，或者每秒30000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。

说明

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 COS 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，详情请参见 域名管理。如果预计存储桶的综合请求速率会超过每秒30000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。

说明

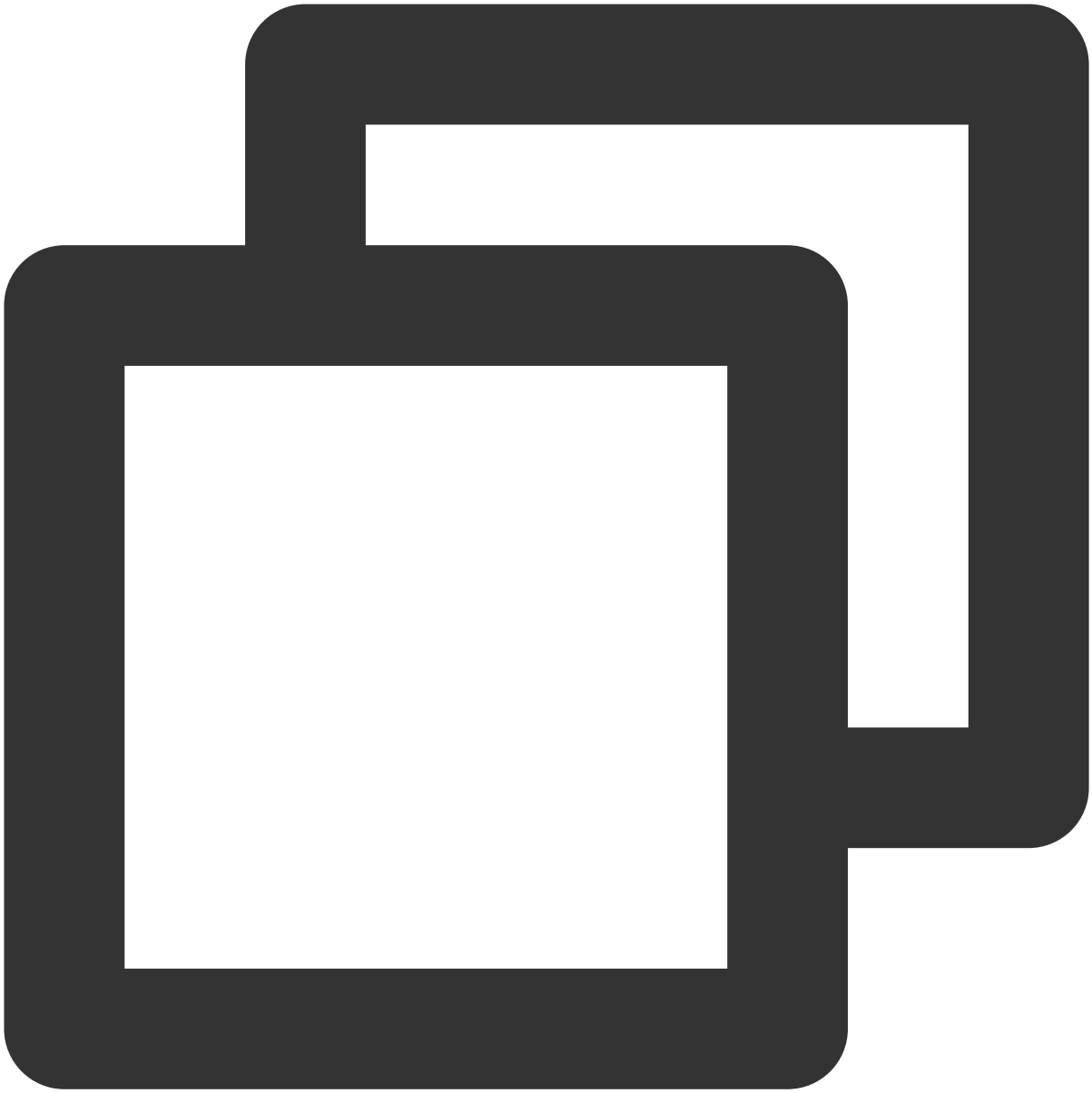
如果您的混合请求负载只是偶尔达到每秒30000个，并在突发时不超过每秒30000个，您可无需遵循本指南。

实践步骤

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 COS 对 Object 键值的存储方法。

腾讯云在 COS 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 `examplebucket-1250000000.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
...
```

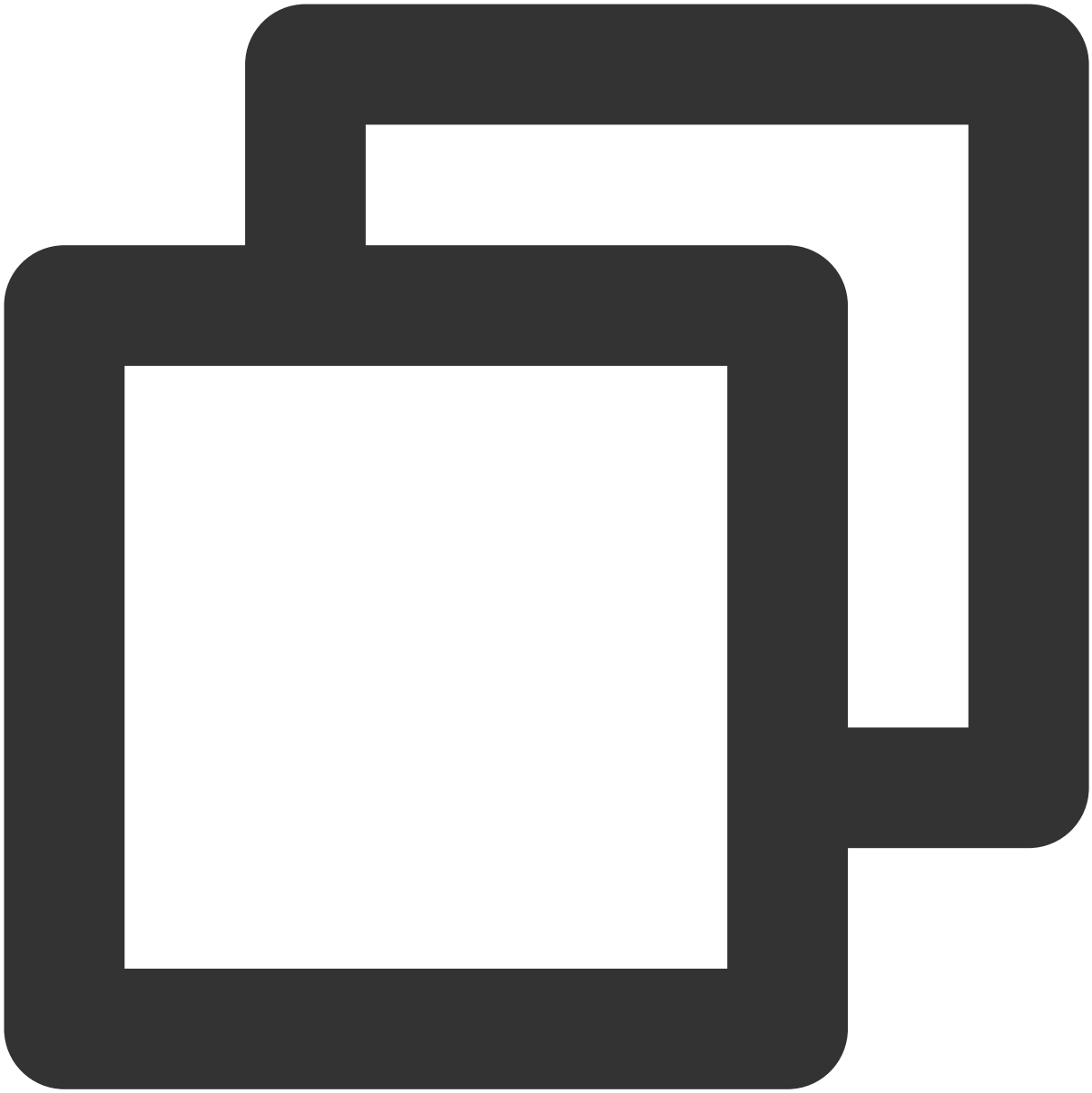
如果您的业务典型负载超过每秒30000个请求，则应避免使用上述案例中的顺序键值。当您的业务必须使用顺序编号或日期时间等字符作为对象键时，您可以使用一些方法向对象键名称添加随机前缀，即可实现在多个索引分区实现键值管理，提升集中负载的性能。以下提供了一些在键值中增加随机性的方法。

注意

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30000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。



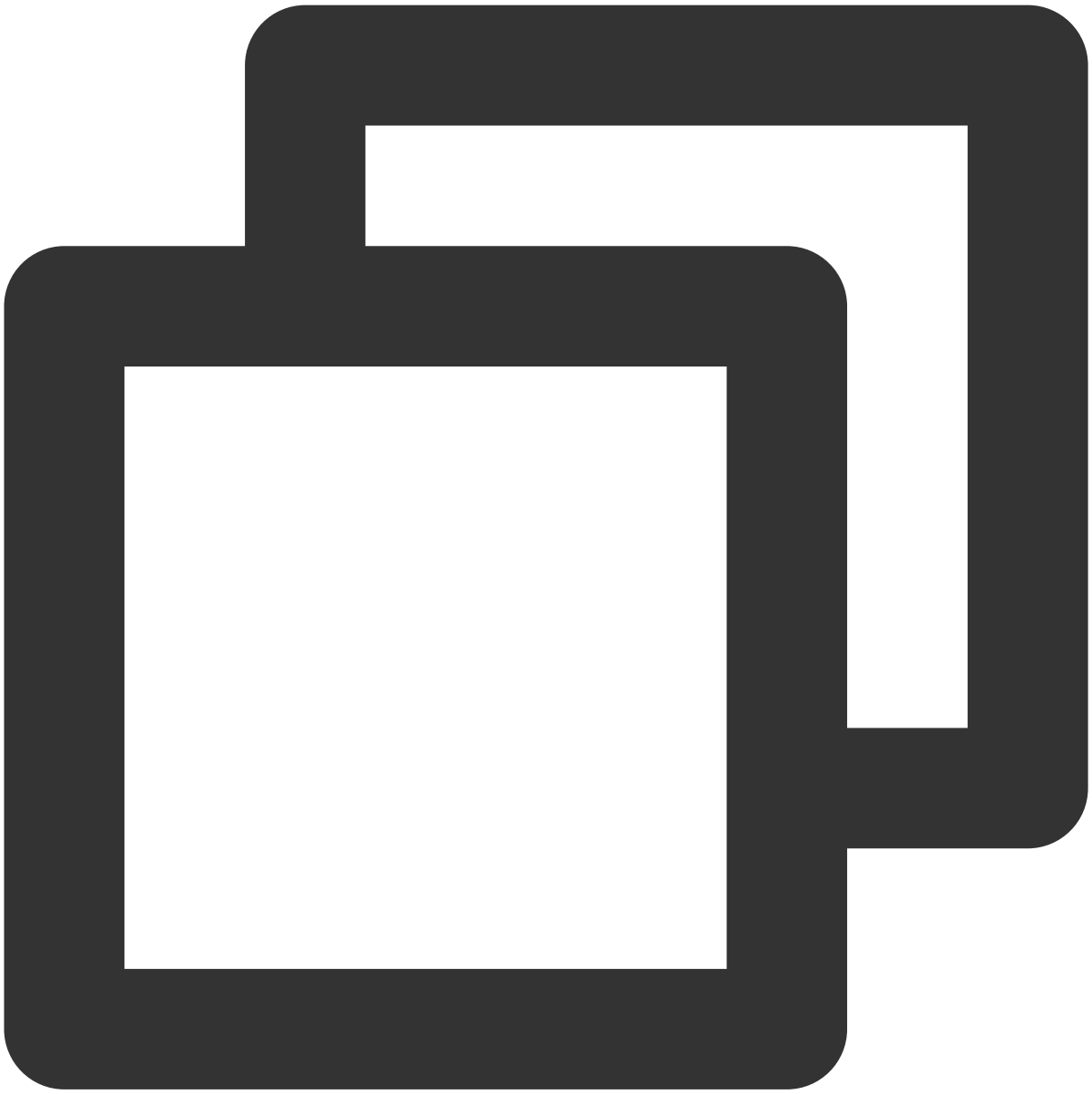
```
faf1-20170701/log120000.tar.gz
e073-20170701/log120500.tar.gz
333c-20170701/log121000.tar.gz
2c32-20170701/log121500.tar.gz
```

注意

由于腾讯云 COS 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。

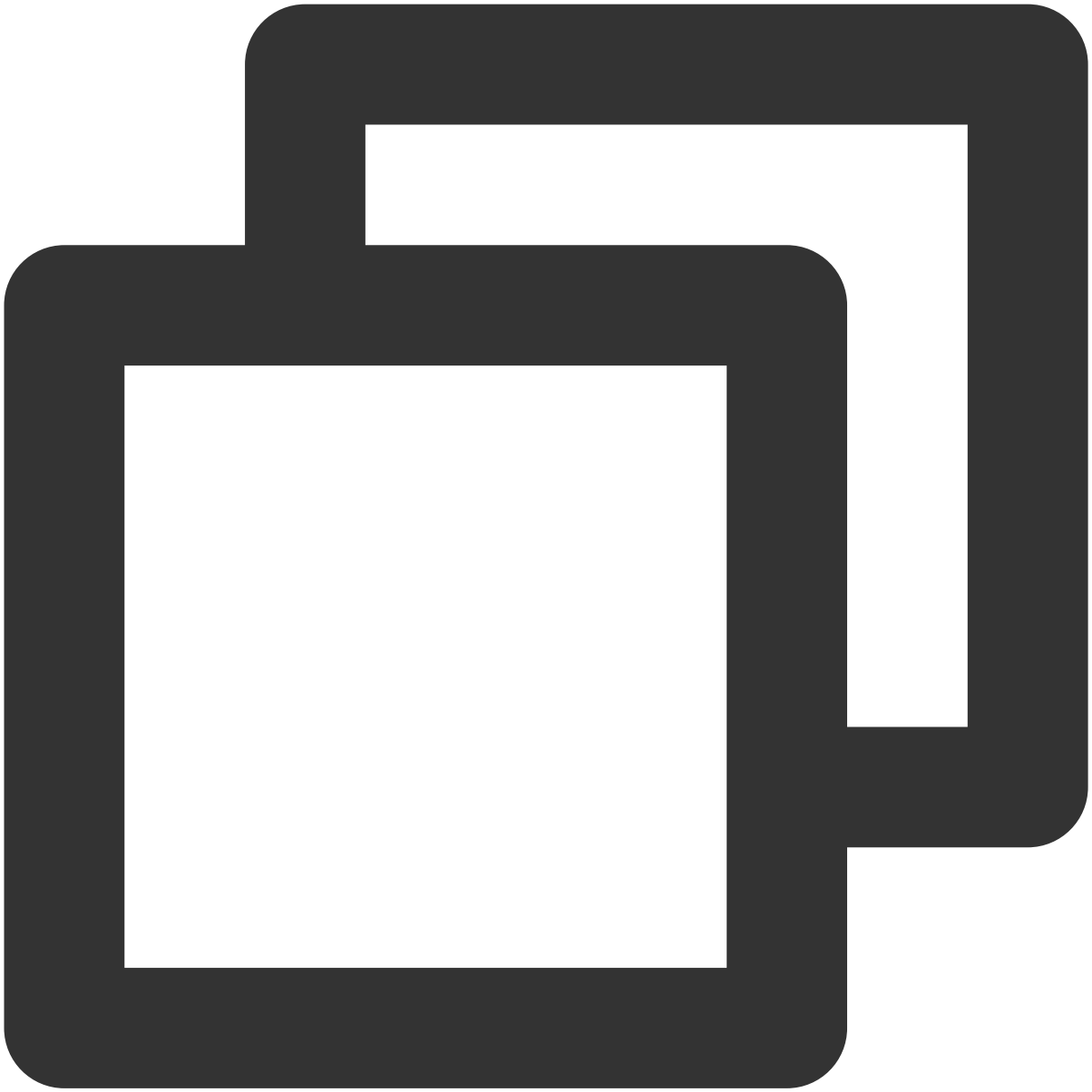
添加枚举值前缀

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30000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。

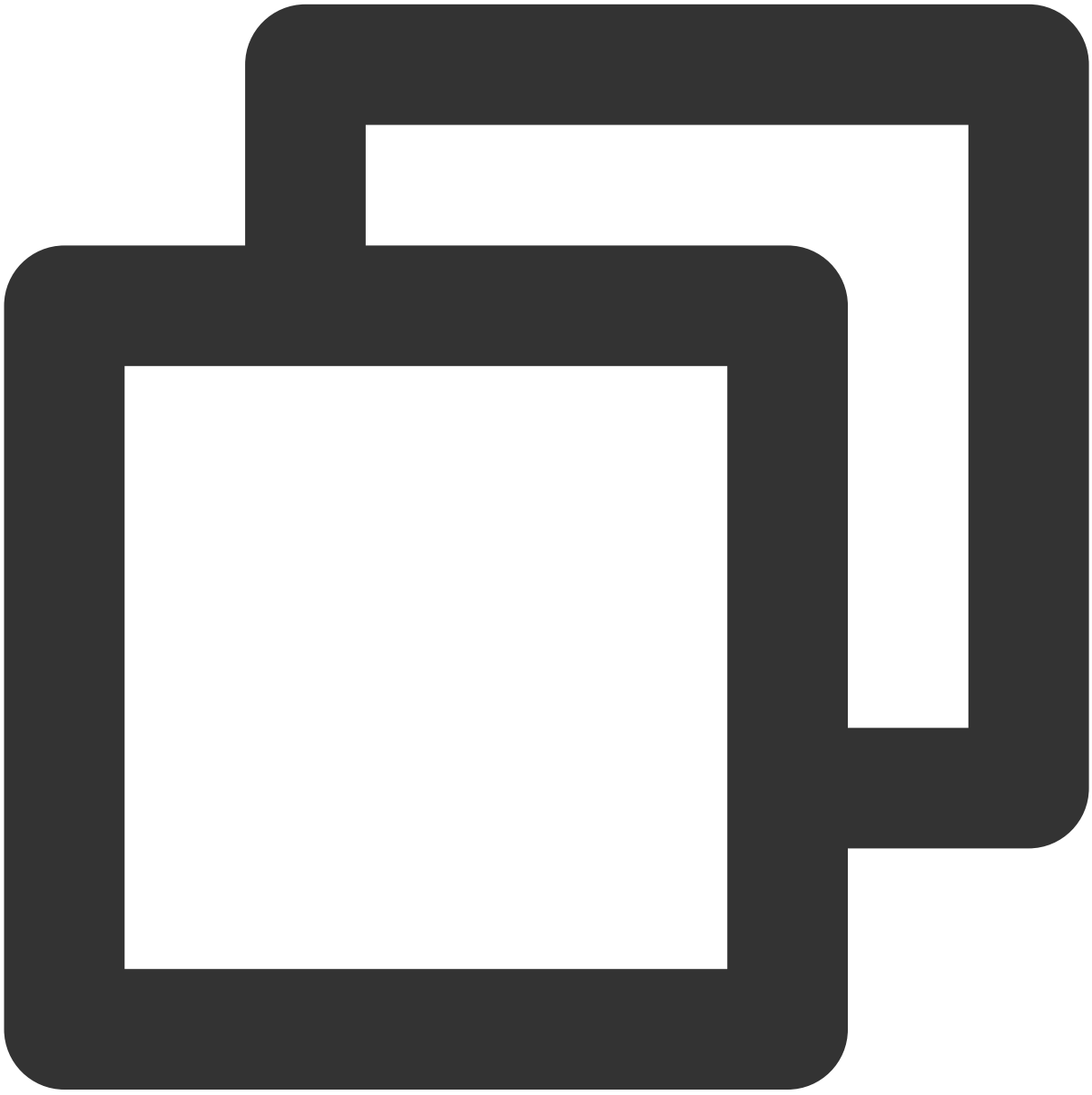


```
logs/faf1-20170701/log120000.tar.gzlogs/
e073-20170701/log120500.tar.gz
logs/333c-20170701/log121000.tar.gz
...
images/0165-image001/indexpage1.jpg
images/a349-image002/indexpage2.jpg
images/ac00-image003/indexpage3.jpg
```

```
...
```

反转键值名称字符串

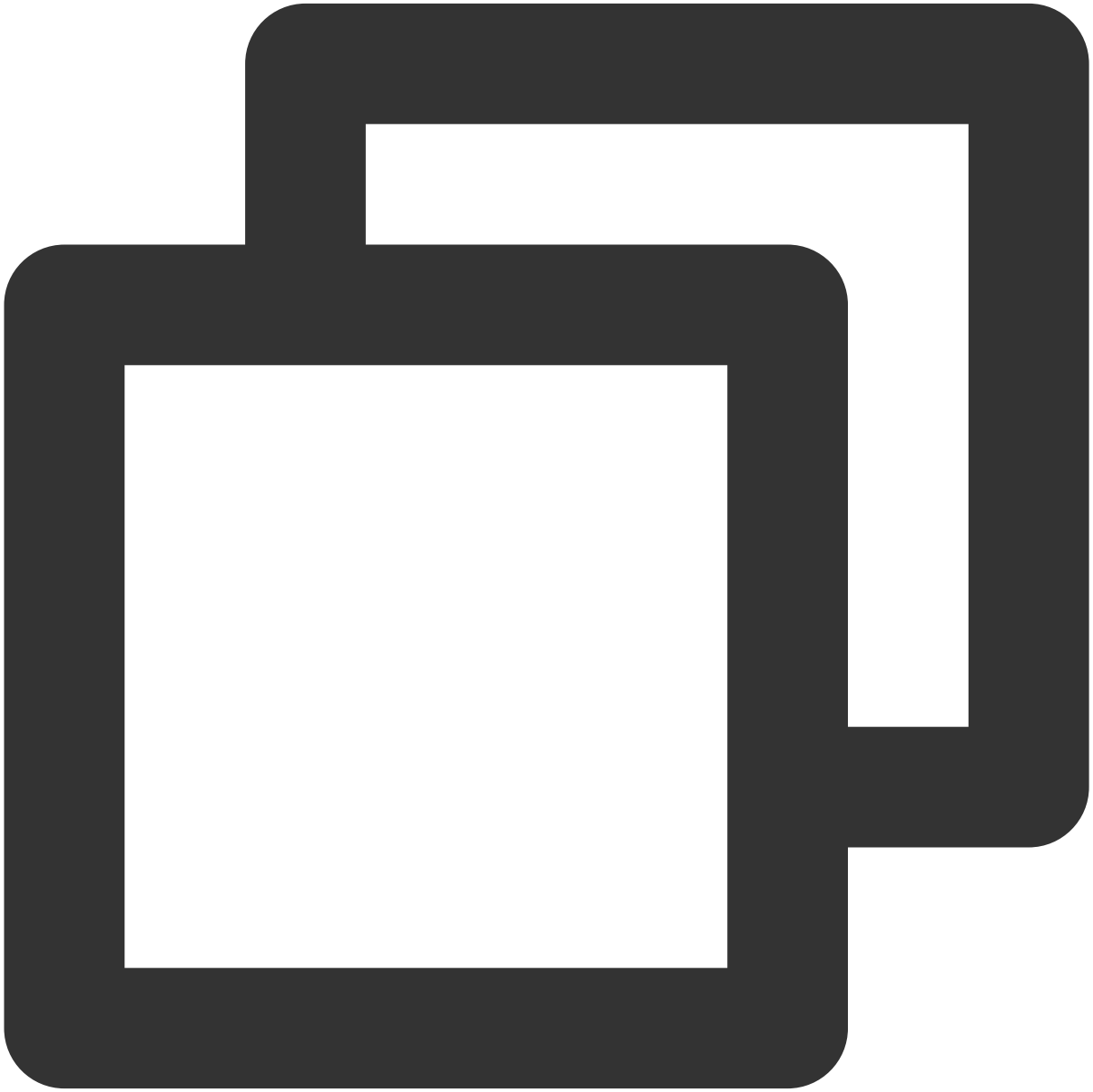
当您的业务不得不使用连续递增的 ID 或日期，或需要一次性上传大量的连续前缀对象时，如下述典型用法：



```
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，详情请参见 域名管理 文档。

COS 压测指南

最近更新时间：2024-01-06 10:47:50

COSBench 简介

COSBench 是一款由 Intel 开源，用于对象存储的压测工具。腾讯云对象存储（Cloud Object Storage，COS）作为兼容 S3 协议的对象存储系统，可使用该工具进行读写性能压测。

系统环境

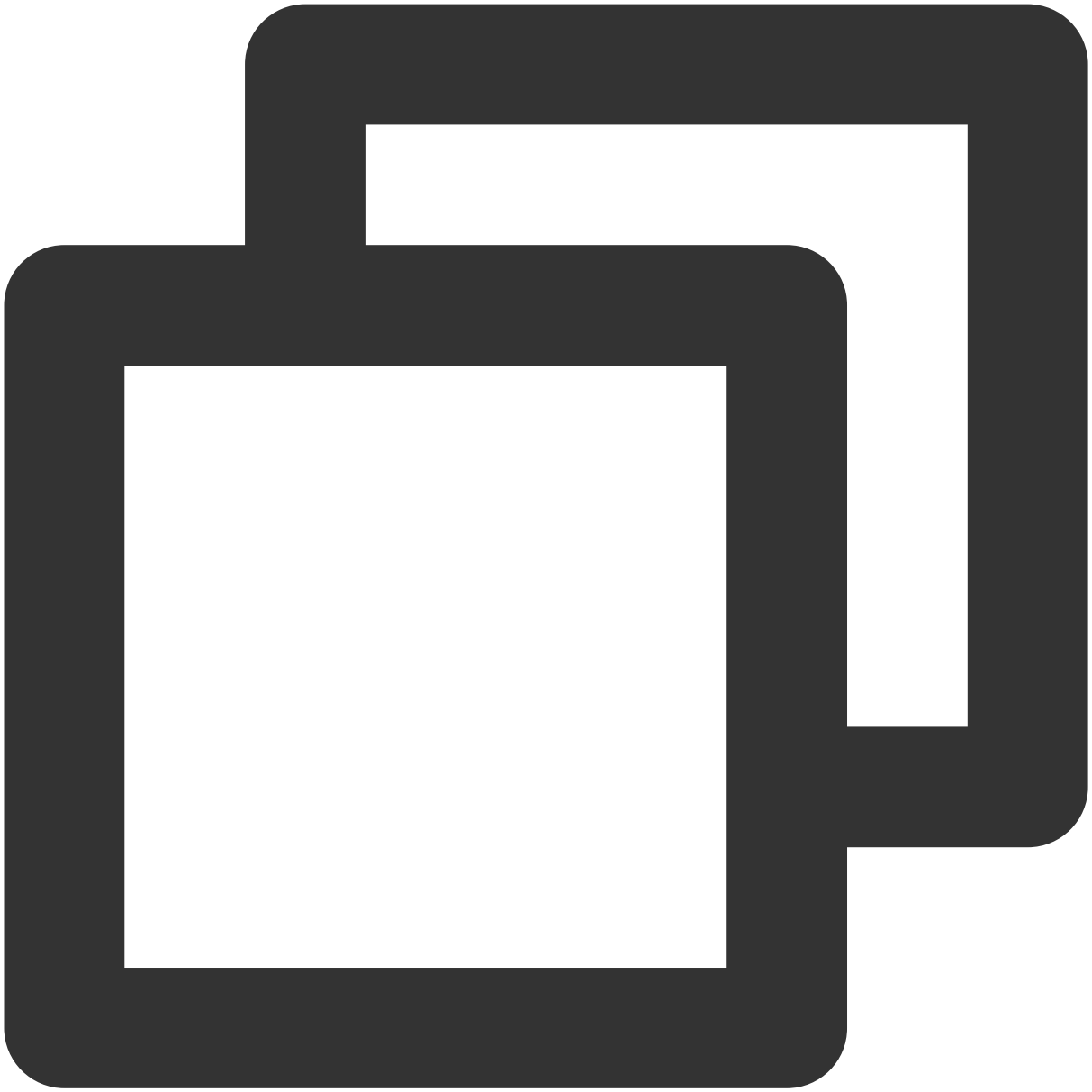
工具推荐运行在 CentOS 7.0 及其以上版本，ubuntu 环境可能存在预期外的问题。

影响性能的因素

机器核心数：机器核心数较少，开启的 worker 数目较多，容易在上下文切换上产生大量的开销，建议采用32或64核进行压测。
机器网卡：机器流出的流量受网卡限制，大文件的流量压力测试，建议采用万兆以上的网卡。
请求网络链路：外网链路质量不一，同时外网下载会产生外网下行流量费用，建议同地域使用内网访问。
测试时间：性能测试时，建议测试时间适当延长，获取一个较为稳定的数值。
测试环境：程序运行的 JDK 版本，同样也会影响性能。例如测试 HTTPS，低版本客户端的加密算法存在 GCM BUG，随机数发生器可能存在锁等问题。

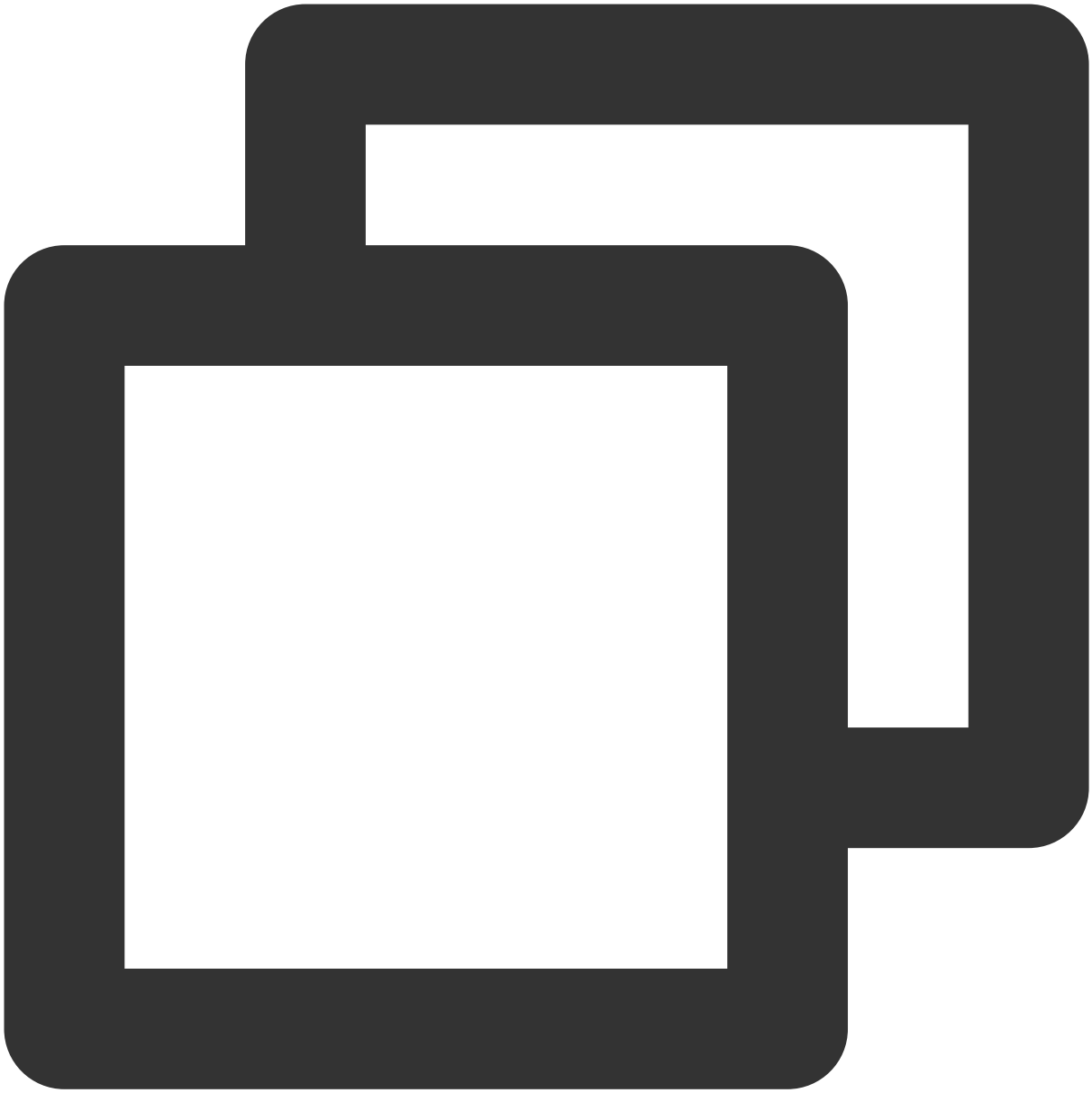
COSBench 实践步骤

1. 从 COSBench GitHub 网站 下载 COSBench 0.4.2.c4.zip 压缩包，并在服务器上进行解压。
2. 安装 COSBench 的依赖库，执行如下命令。
对于 centos 系统，执行如下命令安装依赖：



```
sudo yum install nmap-ncat java curl java-1.8.0-openjdk-devel -y
```

对于 ubuntu 系统，执行如下命令安装依赖



```
sudo apt install nmap openjdk-8-jdk
```

3. 编辑 s3-config-sample.xml 文件并添加任务配置信息，任务配置主要包含如下五个阶段：

3.1 init 阶段：创建存储桶。

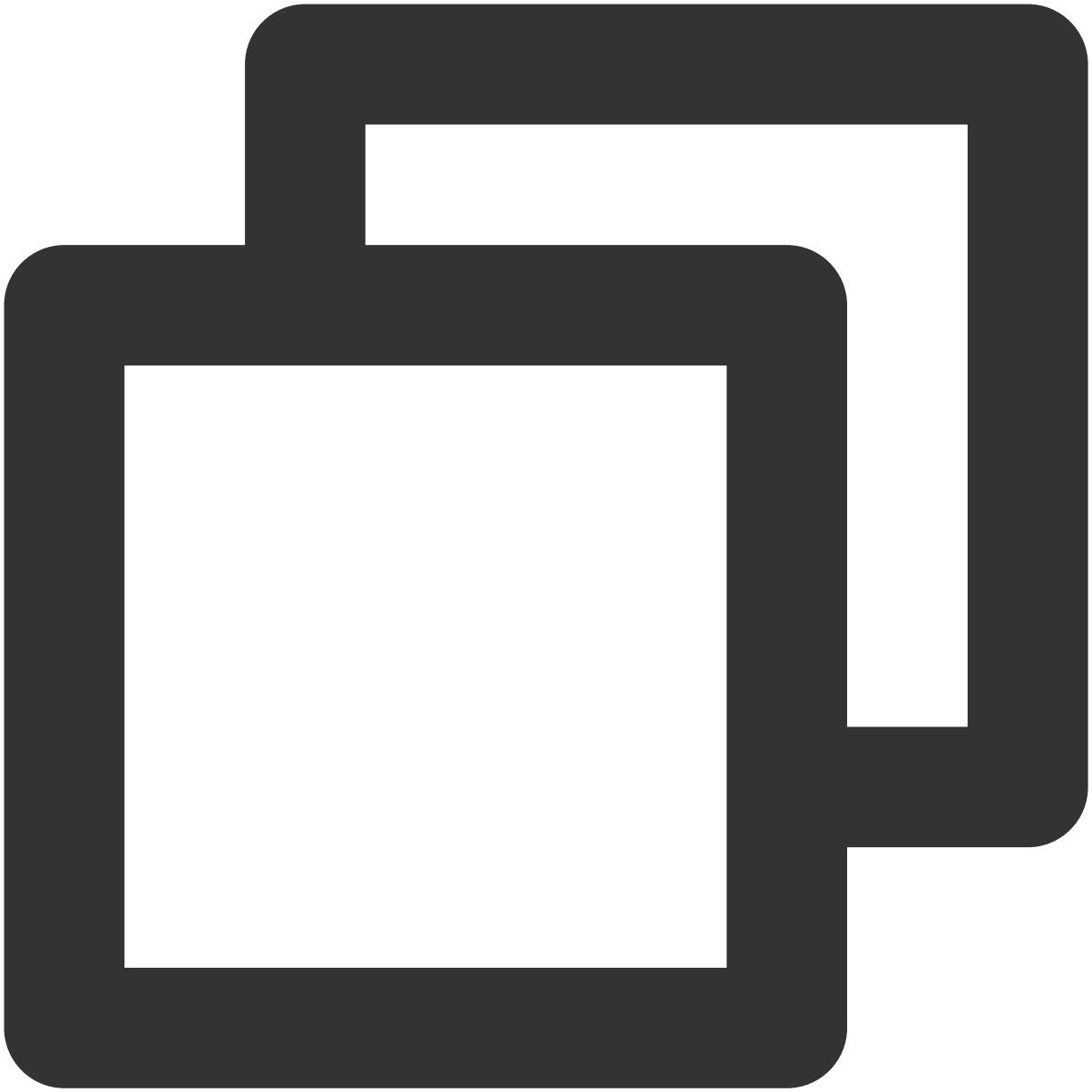
3.2 prepare 阶段：worker 线程，PUT 上传指定大小的对象，用于 main 阶段读取。

3.3 main 阶段：worker 线程混合读写对象，运行指定时长。

3.4 cleanup 阶段，删除生成的对象。

3.5 dispose 阶段：删除存储桶。

示例配置如下：



```
<?xml version="1.0" encoding="UTF-8" ?>
<workload name="s3-50M-sample" description="sample benchmark for s3">

  <storage type="s3" config="accesskey=AKIDHZRLB9Ibhdp7Y7gyQq6BOk1997xxxxxx;secretk

  <workflow>

    <workstage name="init">
      <work type="init" workers="10" config="cprefix=examplebucket;csuffix=-1250000
    </workstage>
```

```
    <workstage name="prepare">
      <work type="prepare" workers="100" config="cprefix=examplebucket;csuffix=-125
    </workstage>

    <workstage name="main">
      <work name="main" workers="100" runtime="300">
        <operation type="read" ratio="50" config="cprefix=examplebucket;csuffix=-12
        <operation type="write" ratio="50" config="cprefix=examplebucket;csuffix=-1
      </work>
    </workstage>

    <workstage name="cleanup">
      <work type="cleanup" workers="10" config="cprefix=examplebucket;csuffix=-1250
    </workstage>

    <workstage name="dispose">
      <work type="dispose" workers="10" config="cprefix=examplebucket;csuffix=-1250
    </workstage>

  </workflow>

</workload>
```

参数说明

参数	描述
accesskey、secretkey	密钥信息，建议使用子账号密钥，授权遵循 最小权限指引，降低使用风险。子账号密钥获取可参考 子账号访问密钥管理
cprefix	存储桶名称前缀，例如 examplebucket
containers	为存储桶名称数值区间，最后的存储桶名称由 cprefix 和 containers 组成，例如：examplebucket1，examplebucket2
csuffix	用户的 APPID，需注意 APPID 前面带上符号 -，例如 -1250000000
runtime	压测运行时间
ratio	读和写的比例
workers	压测线程数

4. 编辑 cosbench-start.sh 文件，在 Java 启动行添加如下参数，关闭 s3 的 md5 校验功能：```plaintext-Dcom.amazonaws.services.s3.disableGetObjectMD5Validation=true```5. 启动 cosbench 服务。 - 对于 centos 系统，执行以下命令：```plaintextsudo bash start-all.sh``` - 对于 ubuntu 系统，执行以下命令：```plaintextsudo bash start-driver.sh

&sudo bash start-controller.sh &```6. 执行以下命令提交任务。```plaintextsudo bash cli.sh submit conf/s3-config-sample.xml```并通过该网址 `http://ip:19088/controller/index.html`（ip 替换为用户的压测机器 IP）查看执行状态：此时可以看到五个执行阶段，如下图所示：7. 下面展示的是所属地域为北京地域、32核、内网带宽为17Gbps 的 CVM 进行上传和下载性能测试，包括以下2个阶段： 1. prepare 阶段：100个 worker 线程，上传1000个50MB 对象。 2. main 阶段：100个 worker 线程混合读写对象，运行300秒。
经过以上阶段1和阶段2的性能压测，结果如下：

Basic Info

ID: w27 **Name**: s3-50M-sample **Current State**: finished

Submitted At: 2019-3-20 10:39:53 **Started At**: 2019-3-20 10:39:53 **Stopped At**: 2019-3-20 10:50:29

more info

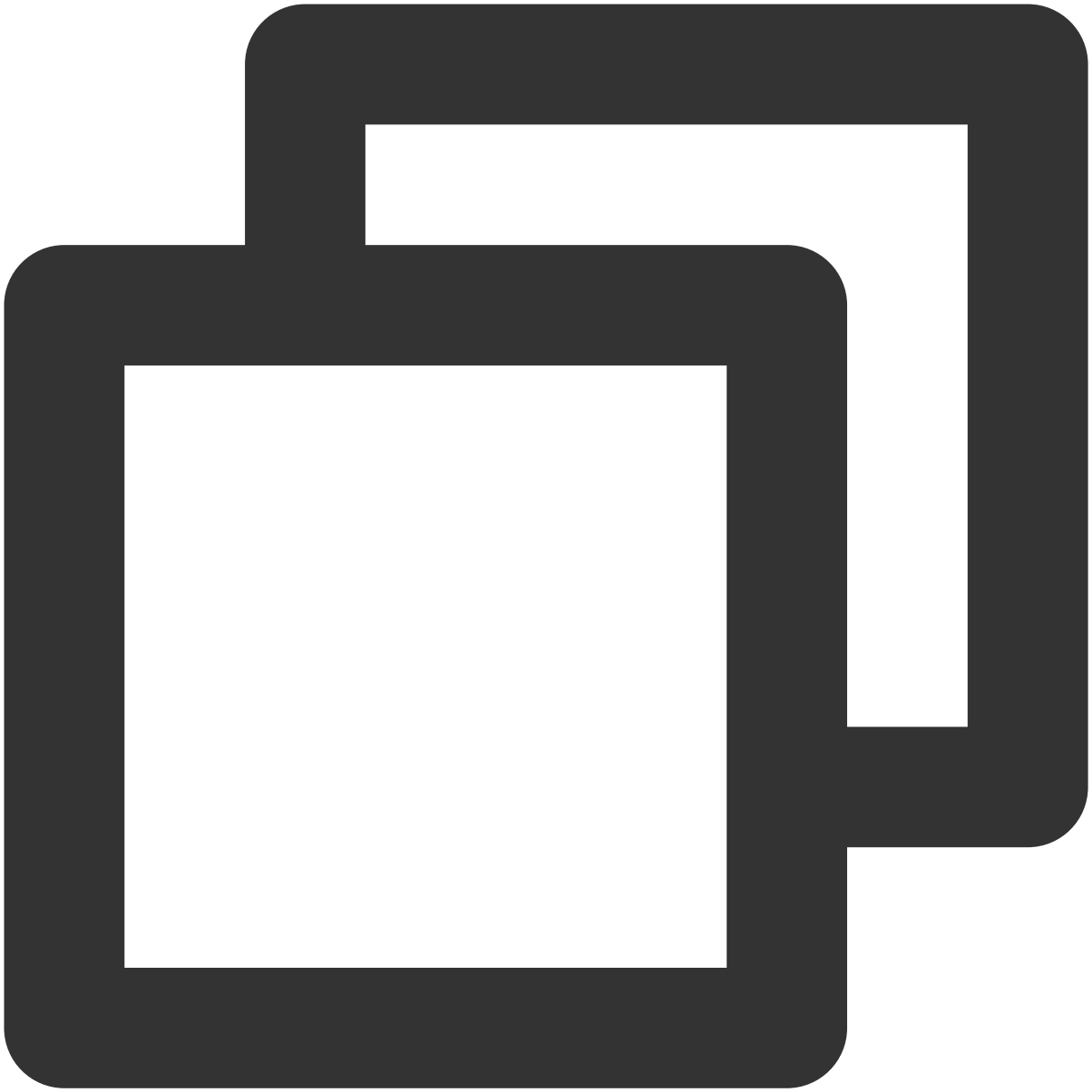
Final Result

General Report

Op-Type	Op-Count	Byte-Count	Avg-ResTime	Avg-ProcTime	Throughput	Bandwidth	Succ-Ratio
op1: init -write	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A
op1: prepare - write	10 kops	500 GB	2460.65 ms	181.83 ms	41.27 op/s	2.06 GB/S	100%
op1: read	10.22 kops	510.75 GB	2012.74 ms	118.33 ms	34.15 op/s	1.71 GB/S	100%
op2: write	10.28 kops	514.2 GB	908.98 ms	317.71 ms	34.38 op/s	1.72 GB/S	100%
op1: cleanup - delete	20 kops	0 B	14.19 ms	14.19 ms	704.74 op/s	0 B/S	100%
op1: dispose - delete	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A

show peformance details

8. 执行以下命令，停止测试服务。



```
sudo bash stop-all.sh
```

数据迁移
本地数据迁移至 COS

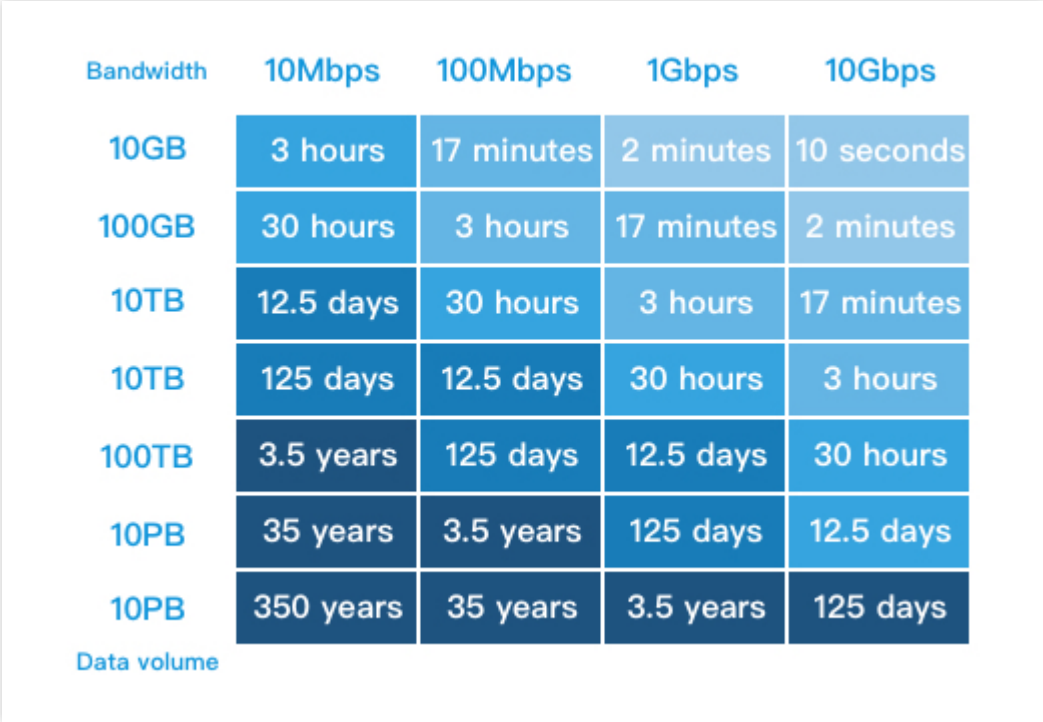
最近更新时间：2024-01-06 10:47:50

实践场景

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。

迁移方式	说明
COS Migration（线上迁移）	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。
云数据迁移 CDM（线下迁移）	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。

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。



说明：

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

迁移实践

COS Migration

迁移操作步骤如下：

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，请参见 COS Migration 工具 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。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。上述调整可在配置文件 config.ini 中 `[common]` 分节，修改如下参数进行调整：

参数名称	参数说明
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。

bigFileExecutorNum	大文件并发度，默认值为8。如果是通过外网来连接 COS，且带宽较小，请减小该并发度。
smallFileExecutorNum	小文件并发度，默认值为64。如果是通过外网来连接 COS，且带宽较小，请减小该并发度。
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，休眠状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。

2. 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。

云数据迁移 CDM

迁移操作步骤如下：
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。
详情请参见 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把数据迁移至腾迅云对象存储 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台 CDM 设备。
多台设备通过不同网络接口连接多台 CDM 设备。

第三方云存储数据迁移至 COS

最近更新时间：2024-01-06 10:47:50

实践背景

对于使用第三方云平台存储的用户，对象存储（Cloud Object Storage，COS）可以帮助用户将第三方云平台的存储数据快速迁移至 COS。

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
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

该平台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。

迁移实践

迁移服务平台 MSP

迁移服务平台（Migration Service Platform，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”能够帮助用户将数据从各类公有云或数据源站中迁移至 COS。

迁移操作步骤如下：

1. 登录 迁移服务平台控制台。
2. 在左导航栏中单击**对象存储迁移**，进入对象存储迁移页面。
3. 单击**新建任务**，新建迁移任务并配置任务信息。
4. 启动任务。

具体操作可参见以下迁移教程：

阿里云 OSS 迁移

华为云 OBS 迁移

七牛云 KODO 迁移

UCLOUD UFile 迁移

金山云 KS3 迁移

百度云 BOS 迁移

AWS S3 迁移

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。

以 URL 作为源地址的数据迁移至 COS

最近更新时间：2024-01-06 10:47:50

实践背景

对于用户想要使用 URL 列表作为数据源地址进行数据迁移。对象存储 COS 支持以下迁移方式：

迁移方式	说明
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。
COS 回源	COS 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。
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。

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 回源 的迁移方式。COS 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。

说明：

目前 COS 暂不支持含有鉴权信息的 URL 数据进行迁移。

迁移实践

迁移服务平台 MSP

迁移操作步骤如下：

1. 登录 迁移服务平台 MSP。
1. 单击左侧菜单栏中的【迁移工具】，找到“文件迁移工具”入口，并单击【立即使用】。
2. 新建迁移任务并配置任务信息。
3. 启动任务。

具体操作可参见 URL 列表迁移。

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差异，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 。

COS 回源

迁移操作步骤如下：

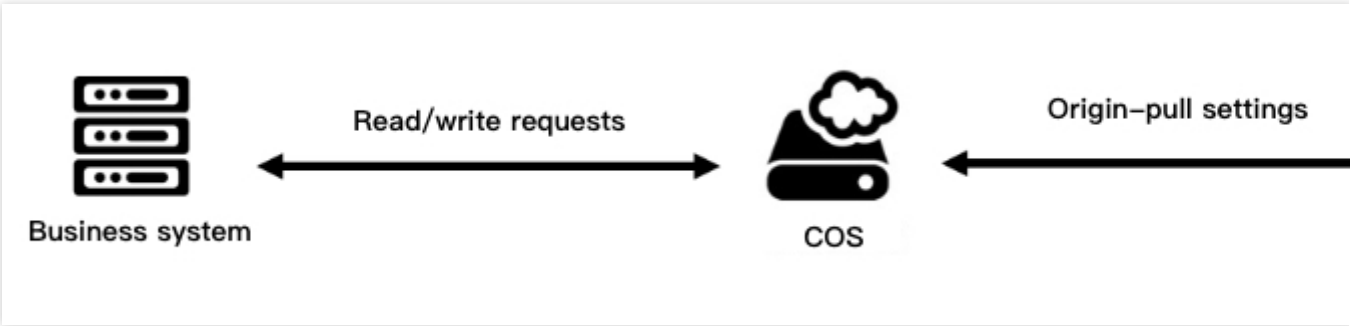
1. 进入对象存储控制台，在待迁移数据的目标存储桶中开启回源设置。
2. 配置存储桶回源地址，然后保存。
3. 将业务系统的读写请求转到腾讯云对象存储 COS 上。

具体操作请参见 设置回源 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。

COS Migration

迁移操作步骤如下：

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，请参见 COS Migration 工具 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。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。上述调整可在配置文件 config.ini 中 `[common]` 分节，修改如下参数进行调整：

参数名称	参数说明
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。
bigFileExecutorNum	大文件并发度，默认值为8。如果是通过外网来连接 COS，且带宽较小，请减小该并发度。
smallFileExecutorNum	小文件并发度，默认值为64。如果是通过外网来连接 COS，且带宽较小，请减小该并发度。
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，休眠状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。

2. 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。

COS 之间数据迁移

最近更新时间：2024-01-06 10:47:50

实践背景

对于正在使用腾讯云对象存储 COS 的用户，如果需要将一个存储桶的数据迁移至同样是腾讯云对象存储 COS 中，我们建议您使用以下迁移方式：
线上迁移：迁移服务平台 MSP
线上迁移：存储桶复制

迁移实践

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。
迁移操作步骤如下：
1. 登录 迁移服务平台 MSP。
2. 在左导航栏中单击【对象存储迁移】，进入对象存储迁移页面。
3. 单击【新建任务】，新建迁移任务并配置任务信息。
4. 启动任务。
具体操作可参见 腾讯云 COS 间迁移。
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差异，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 。

存储桶复制

存储桶复制是腾讯云对象存储 COS 针对存储桶的一项配置，通过配置存储桶复制规则，可以在不同地域的存储桶中自动、异步地复制**增量对象**。启用存储桶复制后，COS 将精确复制源存储桶中的对象内容（例如对象元数据和版本 ID 等）到目标存储桶中，复制的对象副本拥有完全一致的属性信息。此外，源存储桶中对于对象的操作，如上传对象、删除对象等操作，也将被复制到目标存储桶中。具体介绍和操作请参见 存储桶复制。

Hadoop 文件系统与 COS 之间的数据迁移

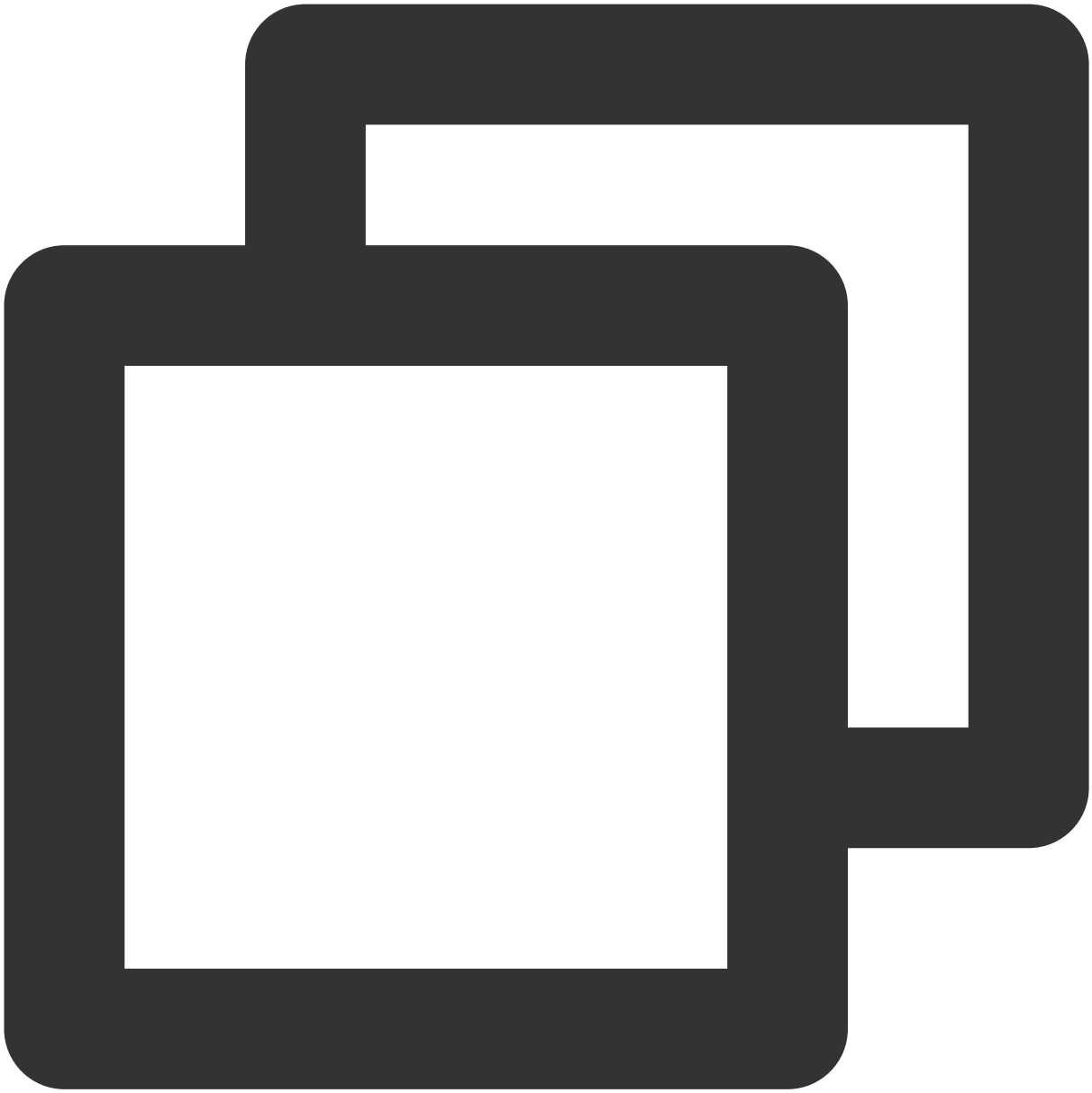
最近更新时间：2024-01-06 10:47:50

简介

Hadoop Distcp（Distributed copy）主要是用于 Hadoop 文件系统内部或之间进行大规模数据复制的工具，它基于 Map/Reduce 实现文件分发、错误处理以及最终的报告生成。由于利用了 Map/Reduce 的并行处理能力，每个 Map 任务负责完成源路径中部分文件的复制，因此它可以充分利用集群资源来快速完成集群或 Hadoop 文件系统之间的大规模数据迁移。
由于 Hadoop-COS 实现了 Hadoop 文件系统的语义，因此利用 Hadoop Distcp 工具可以方便地在对象存储（Cloud Object Storage，COS）与其他 Hadoop 文件系统之间进行双向的数据迁移，本文就以 HDFS 为例，介绍 Hadoop 文件系统与 COS 之间利用 Hadoop Distcp 工具完成数据迁移的方式。

前提条件

1. Hadoop 集群中已经安装 Hadoop-COS 插件，并且正确配置了 COS 的访问密钥等。可使用如下 Hadoop 命令检查 COS 访问是否正常：



```
hadoop fs -ls cosn://examplebucket-1250000000/
```

如果能够正确地列出 COS Bucket 中的文件列表，则表示 Hadoop-COS 安装和配置正确，可以进行以下实践步骤。

2. COS 的访问账户必须要具备读写 COS 存储桶中目标路径的权限。

注意

您可以按需授予子账号读写 COS 存储桶内资源的权限，建议按照 最小权限原则 和 子用户授权指南 进行授权，以下几种是常见预设策略：

DataFullControl：数据全读写权限，包含读、写、列出文件列表以及删除操作，建议谨慎授予。

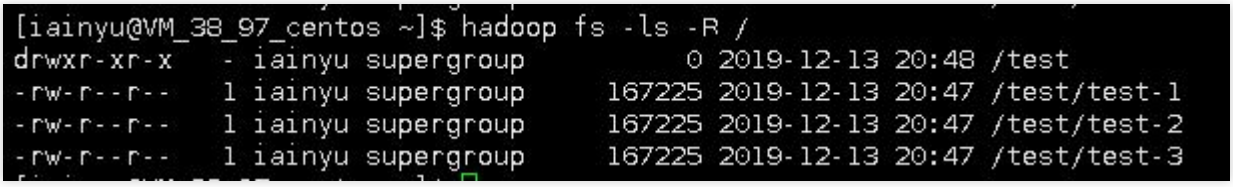
QcloudCOSDataReadOnly：数据只读权限。

QcloudCOSDataWriteOnly：数据只写权限。
如果需要使用自定义监控能力，需要授权腾讯云可观测平台指标上报和读取接口操作权限，请谨慎授予 QcloudMonitorFullAccess或者按需授予 腾讯云可观测平台接口 权限 。

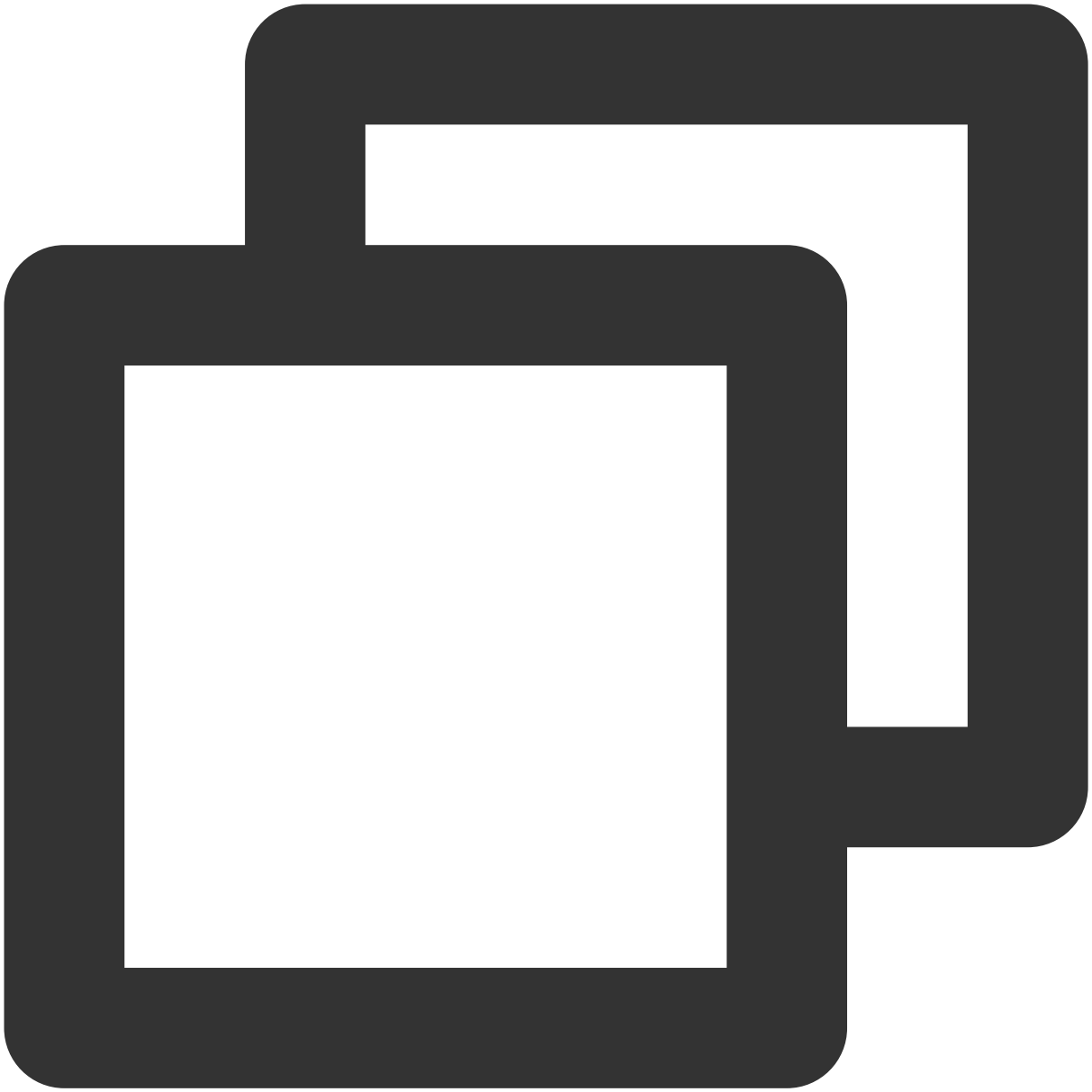
实践步骤

将 HDFS 中的数据复制到 COS 的存储桶中

通过 Hadoop Distcp 将本地 HDFS 集群中 `/test` 目录下的文件迁移到 COS 的 hdfs-test-1250000000 存储桶中。

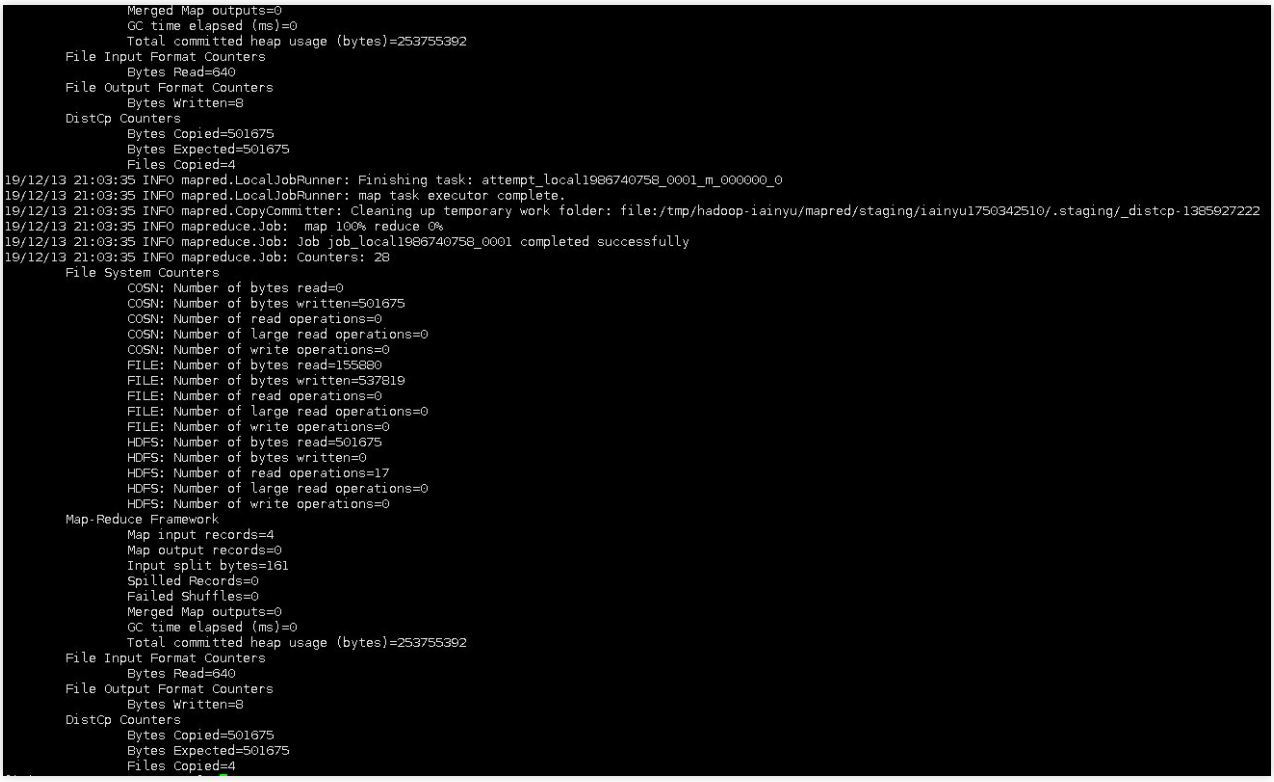


1. 执行如下命令启动迁移：

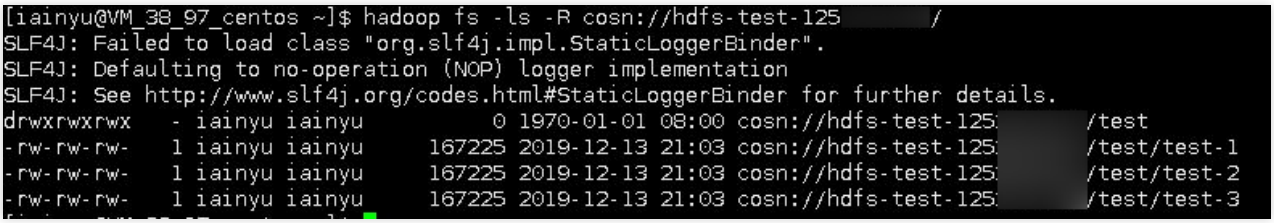


```
hadoop distcp hdfs://10.0.0.3:9000/test cosn://hdfs-test-1250000000/
```

Hadoop Distcp 会启动 MapReduce 作业来执行文件复制任务，完成后会输出简单报表信息，如下图所示：

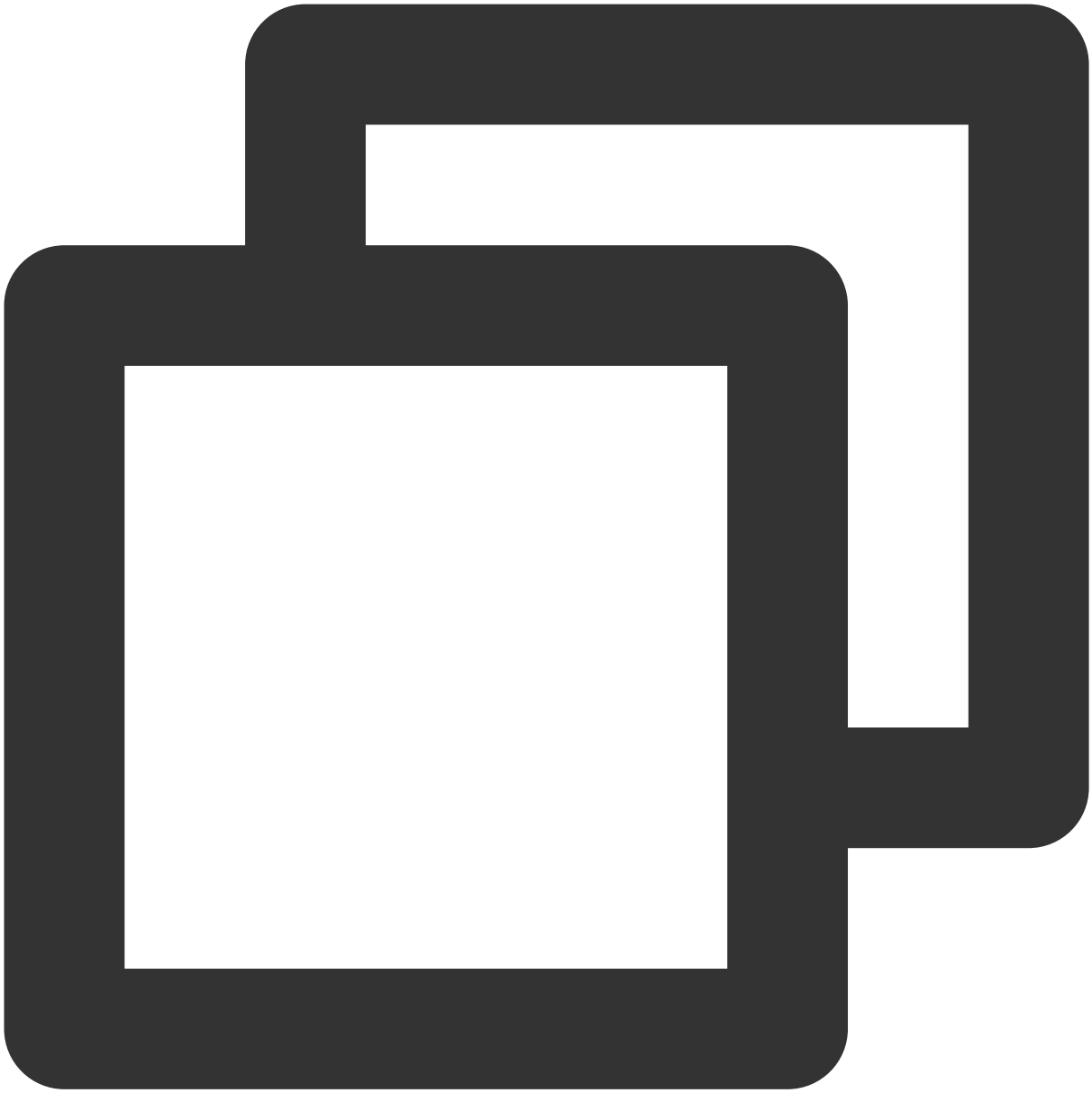


2. 执行 `hadoop fs -ls -R cosn://hdfs-test-1250000000/` 命令可以列出刚才已迁移到存储桶 hdfs-test-1250000000 的目录和文件。



将 COS 中存储桶的文件复制到本地 HDFS 集群

Hadoop Distcp 是一个支持不同集群和文件系统之间复制数据的工具，因此，将 COS 存储桶中的对象路径作为源路径，HDFS 的文件路径作为目标路径即可将 COS 中的数据文件复制到本地 HDFS：

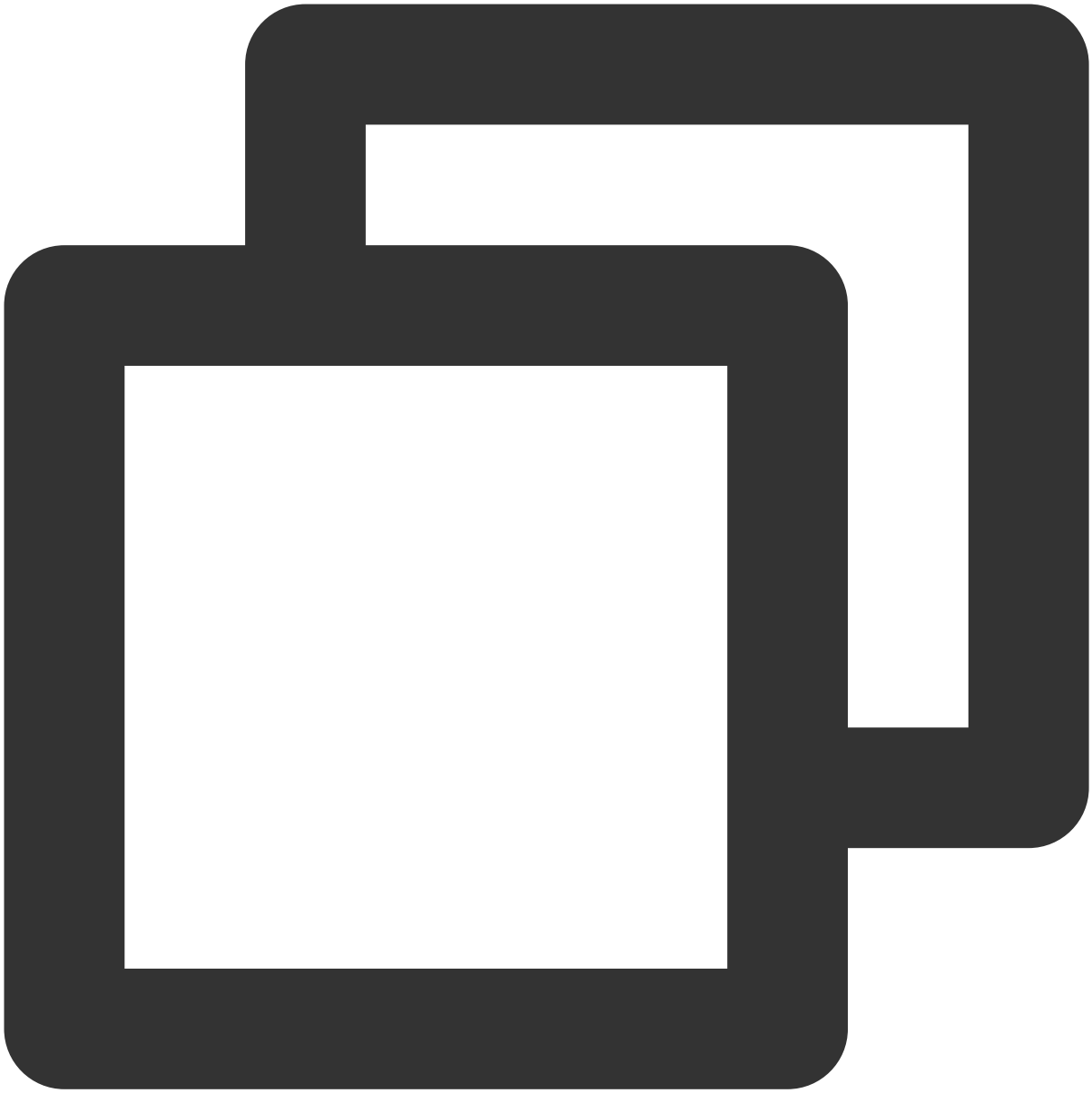


```
hadoop distcp cosn://hdfs-test-1250000000/test hdfs://10.0.0.3:9000/
```

指定配置 Distcp 命令行参数进行 HDFS 和 COS 之间的数据迁移

说明

该命令行配置支持双向操作，可支持 HDFS 数据迁移到 COS，也可以将 COS 数据迁移到 HDFS。
用户可直接配置以下命令：



```
hadoop distcp -Dfs.cosn.impl=org.apache.hadoop.fs.CosFileSystem -Dfs.cosn.bucket.re
```

参数说明如下：

Dfs.cosn.impl：始终配置为 org.apache.hadoop.fs.CosFileSystem。

Dfs.cosn.bucket.region：填写存储桶所在地域，可在 COS 控制台存储桶列表中查看。

Dfs.cosn.userinfo.secretId：填写存储桶拥有者账号下的 SecretId，可前往 云 API 密钥 中获取。

Dfs.cosn.userinfo.secretKey：填写存储桶拥有者账号下的 secretKey，可前往 云 API 密钥 中获取。

libjars：指定 Hadoop-COS jar 包位置。Hadoop-COS jar 包可前往 Github 仓库 中的 dep 目录下进行下载。

说明

其他参数请参考 Hadoop 工具 文档。

Hadoop distcp 的扩展参数

Hadoop distcp 工具支持丰富的运行参数。例如，可以通过 `-m` 来指定最大用于并行复制的 Map 任务数目，`-bandwidth` 来限制每个 map 所使用的最大带宽等。具体可参考 Apache Hadoop distcp 工具的官方文档：DistCp Guide。

使用 AWS S3 SDK 访问 COS

最近更新时间：2024-01-06 10:47:50

简介

对象存储（Cloud Object Storage，COS）提供了 AWS S3 兼容的 API，因此当您的数据从 S3 迁移到 COS 之后，只需要进行简单的配置修改，即可让您的客户端应用轻松兼容 COS 服务。本文主要介绍不同开发平台的 S3 SDK 的适配步骤。在完成添加适配步骤后，您就可以使用 S3 SDK 的接口来访问 COS 上的文件了。

准备工作

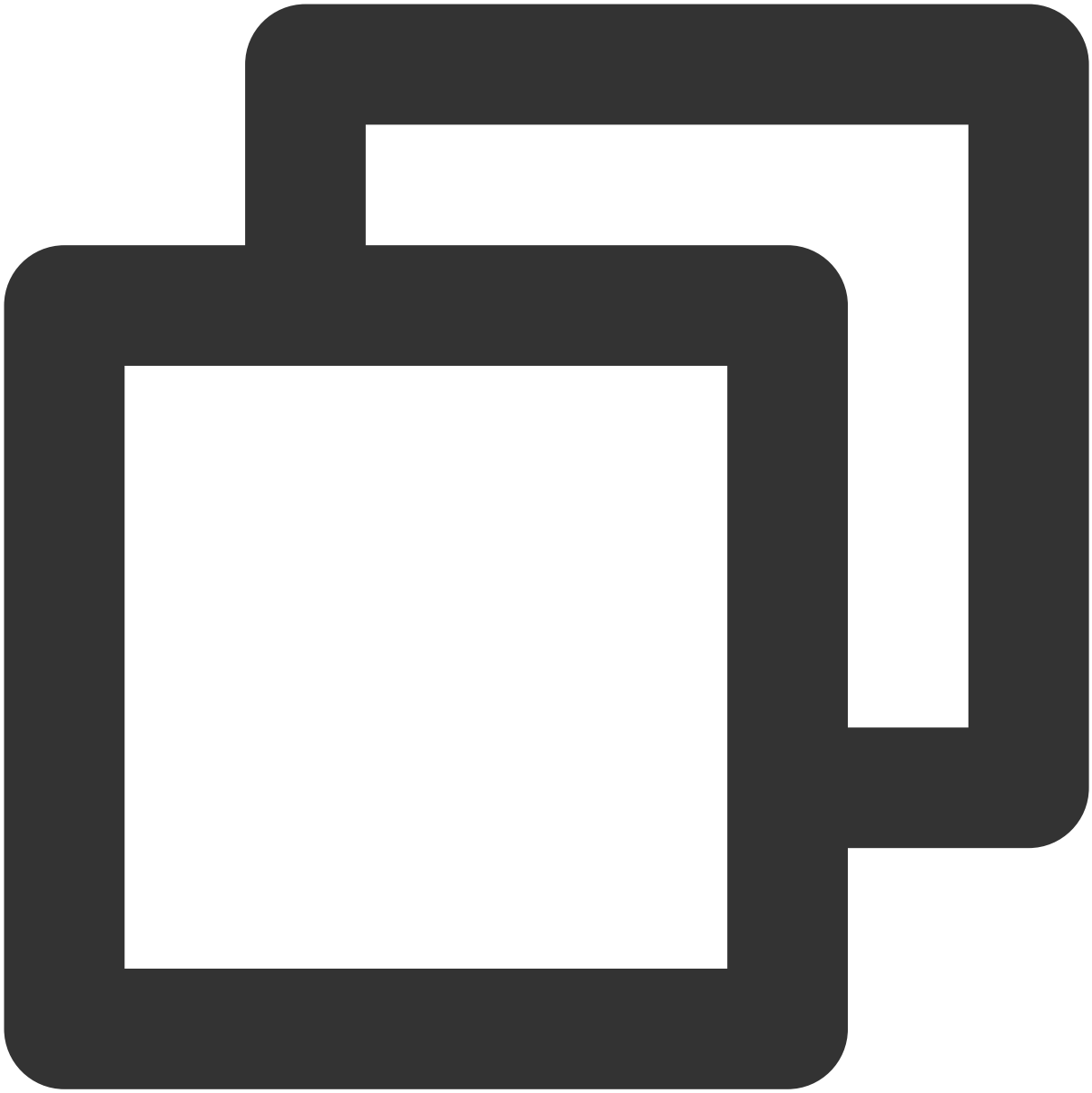
1. 您已 注册腾讯云账号，并且从 访问管理控制台 上获取了腾讯云密钥 SecretID 与 SecretKey。
2. 您已有一个集成了 S3 SDK，并能正常运行的客户端应用。

Android

下面以 AWS Android SDK 2.14.2 版本为例，介绍如何适配以便访问 COS 服务。对于终端访问 COS，将永久密钥放到客户端代码中有极大的泄露风险，我们建议您接入 STS 服务获取临时密钥，详情请参见 临时密钥生成及使用指引。

初始化

初始化实例时，您需要设置临时密钥提供者和 Endpoint，以存储桶所在地域是 `ap-guangzhou` 为例：



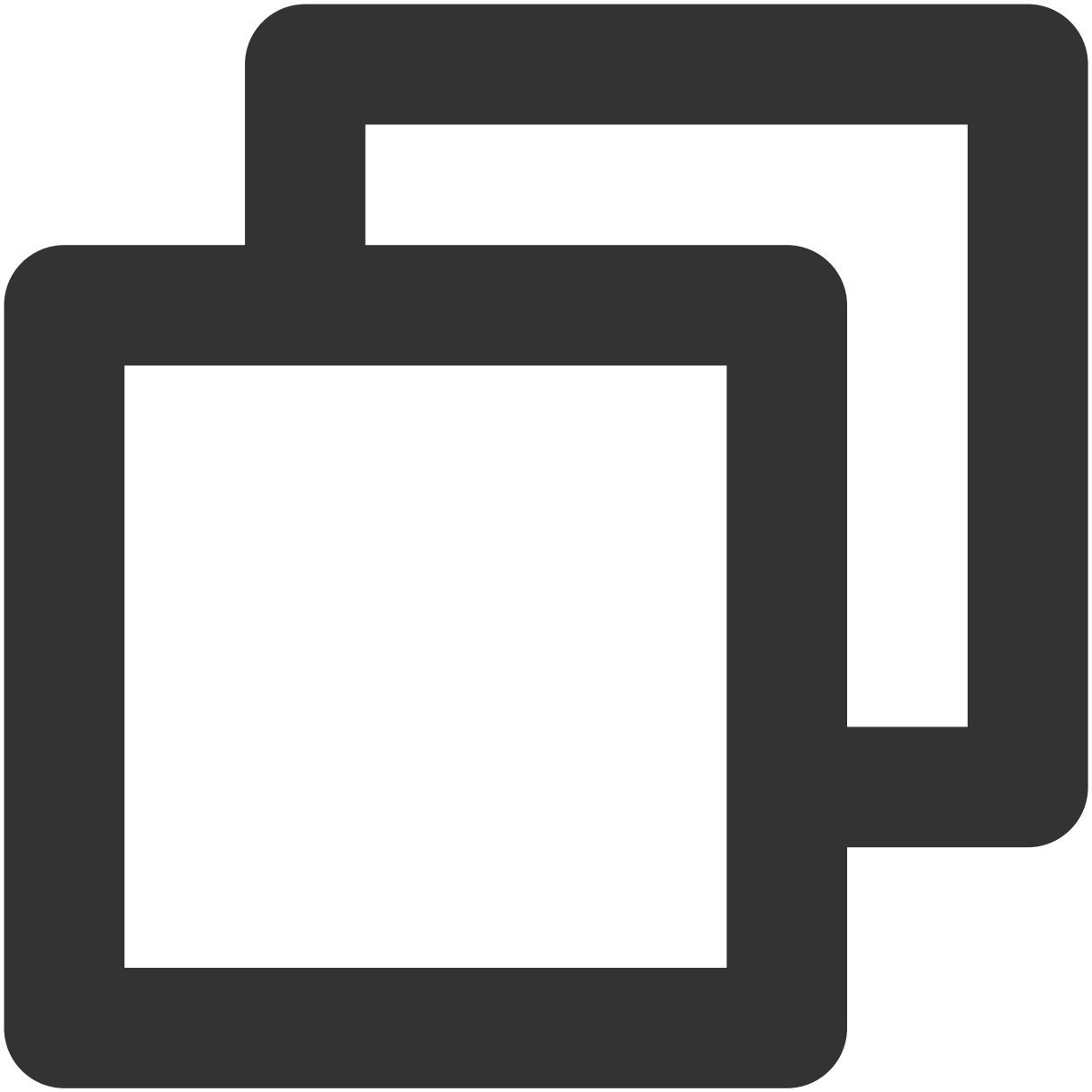
```
AmazonS3Client s3 = new AmazonS3Client(new AWSCredentialsProvider() {
    @Override
    public AWSCredentials getCredentials() {
         // 这里后台请求 STS 得到临时密钥信息
        return new BasicSessionCredentials(
                "<TempSecretID>", "<TempSecretKey>", "<STSSessionToken>"
        );
    }

    @Override
    public void refresh() {
```

```
            //
        }
});

s3.setEndpoint("cos.ap-guangzhou.myqcloud.com");
```

iOS

以 AWS iOS SDK 2.10.2 版本为例，介绍如何适配以便访问 COS 服务。对于终端访问 COS，将永久密钥放到客户端代码中有极大的泄露风险，我们建议您接入 STS 服务获取临时密钥，详情请参见 临时密钥生成及使用指引。

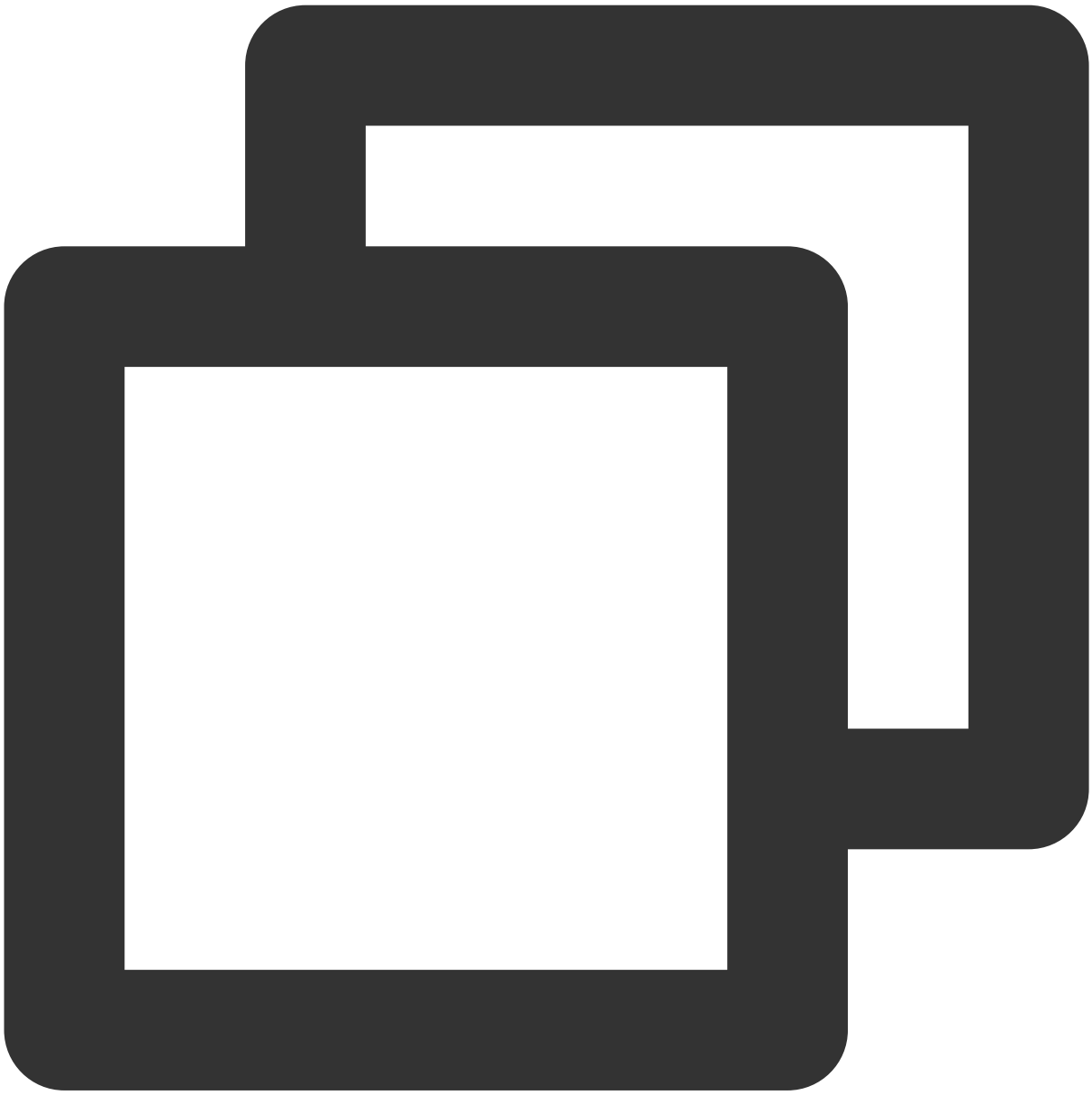
1. 实现 AWSCredentialsProvider 协议



```
-(AWSTask<AWSCredentials *> *)credentials{
    // 这里后台请求 STS 得到临时密钥信息
    AWSCredentials *credential = [[AWSCredentials alloc]initWithAccessKey:@"<TempSe

    return [AWSTask taskWithResult:credential];

}

- (void)invalidateCachedTemporaryCredentials{

}
```

2. 提供临时密钥提供者和 Endpoint

以存储桶所在地域是 `ap-guangzhou` 为例：



```
NSURL* bucketURL = [NSURL URLWithString:@"https://cos.ap-guangzhou.myqcloud.com"];

AWSEndpoint* endpoint = [[AWSEndpoint alloc] initWithRegion:AWSRegionUnknown servic
AWSServiceConfiguration* configuration = [[AWSServiceConfiguration alloc]
    initWithRegion:AWSRegionUSEast2 endpoint:endpoint
    credentialsProvider:[MyCredentialProvider new]]; // MyCredentialProvider 实现了
```

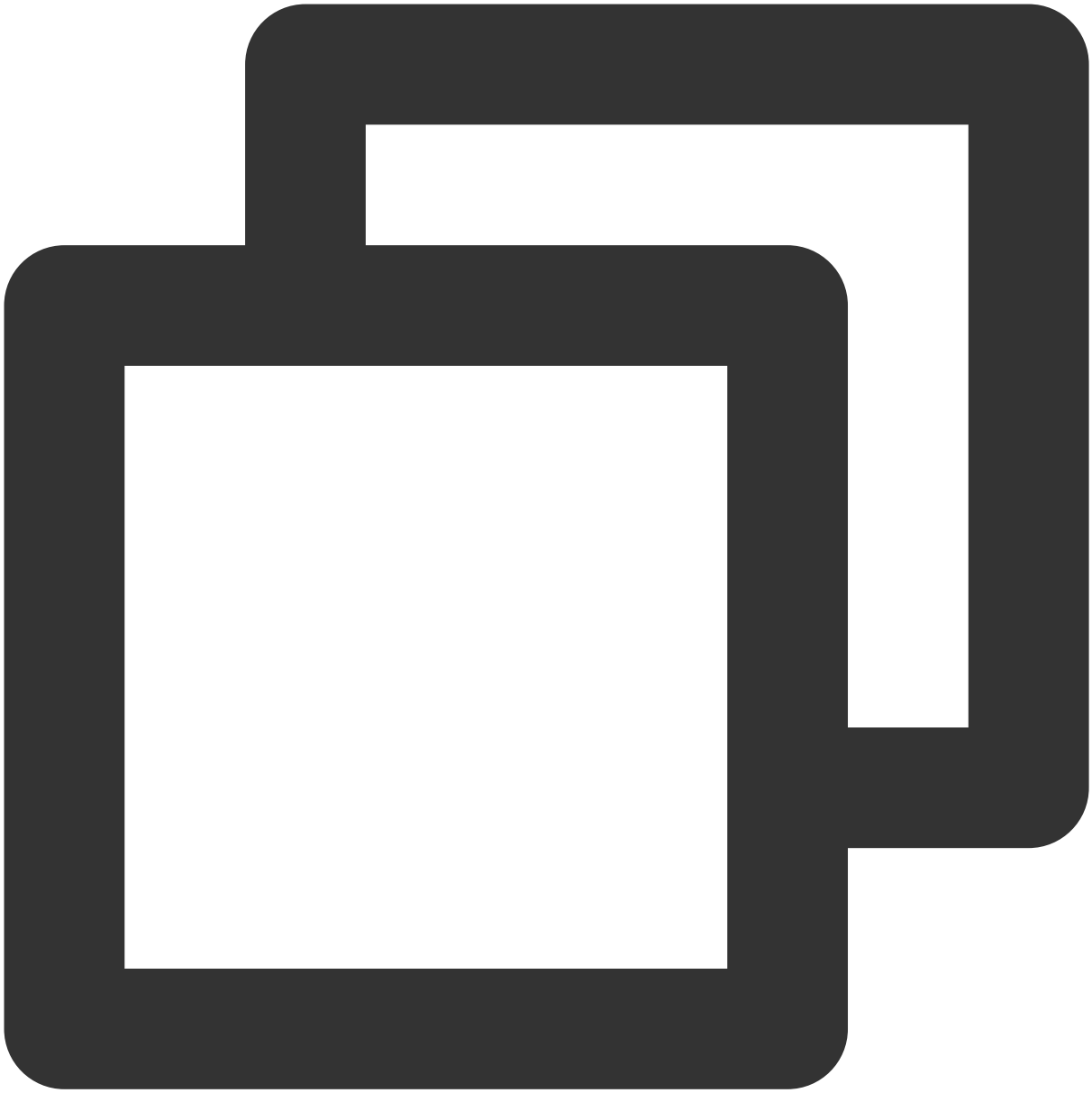
```
[[AWSServiceManager defaultServiceManager] setDefaultServiceConfiguration:configura
```

Node.js

下面以 AWS JS SDK 2.509.0 版本为例，介绍如何适配以便访问 COS 服务。

初始化

初始化实例时设置腾讯云密钥和 Endpoint，以存储桶所在地域是 `ap-guangzhou` 为例，代码示例如下：



```
var AWS = require('aws-sdk');

AWS.config.update({
    accessKeyId: "COS_SECRETID",
    secretAccessKey: "COS_SECRETKEY",
    region: "ap-guangzhou",
    endpoint: 'https://cos.ap-guangzhou.myqcloud.com',
});

s3 = new AWS.S3({apiVersion: '2006-03-01'});
```

Java

下面以 AWS Java SDK 1.11.609 版本为例，介绍如何适配以便访问 COS 服务。

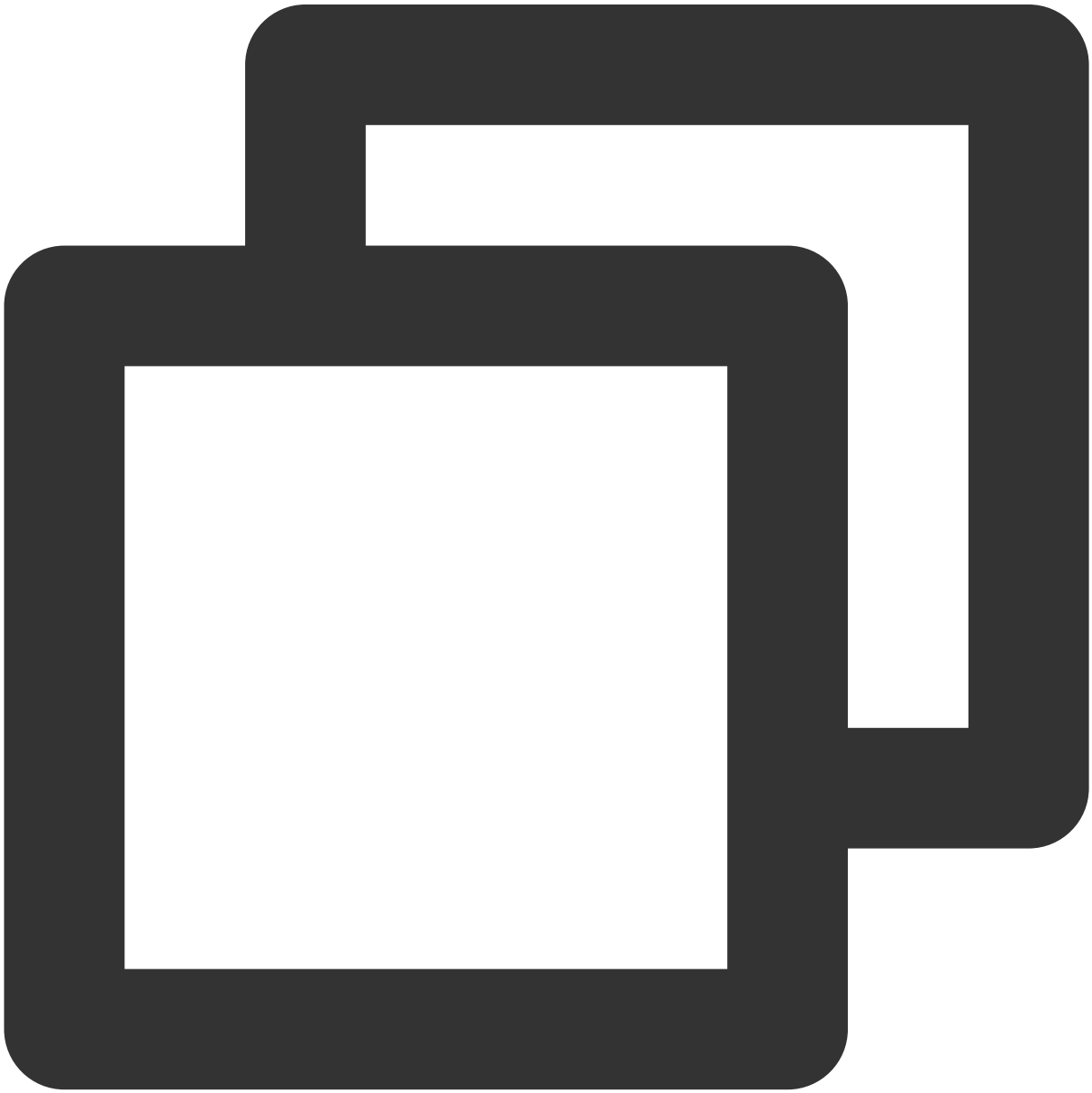
1. 修改 AWS 配置和证书文件

说明

下面以 Linux 为例，修改 AWS 配置和证书文件。

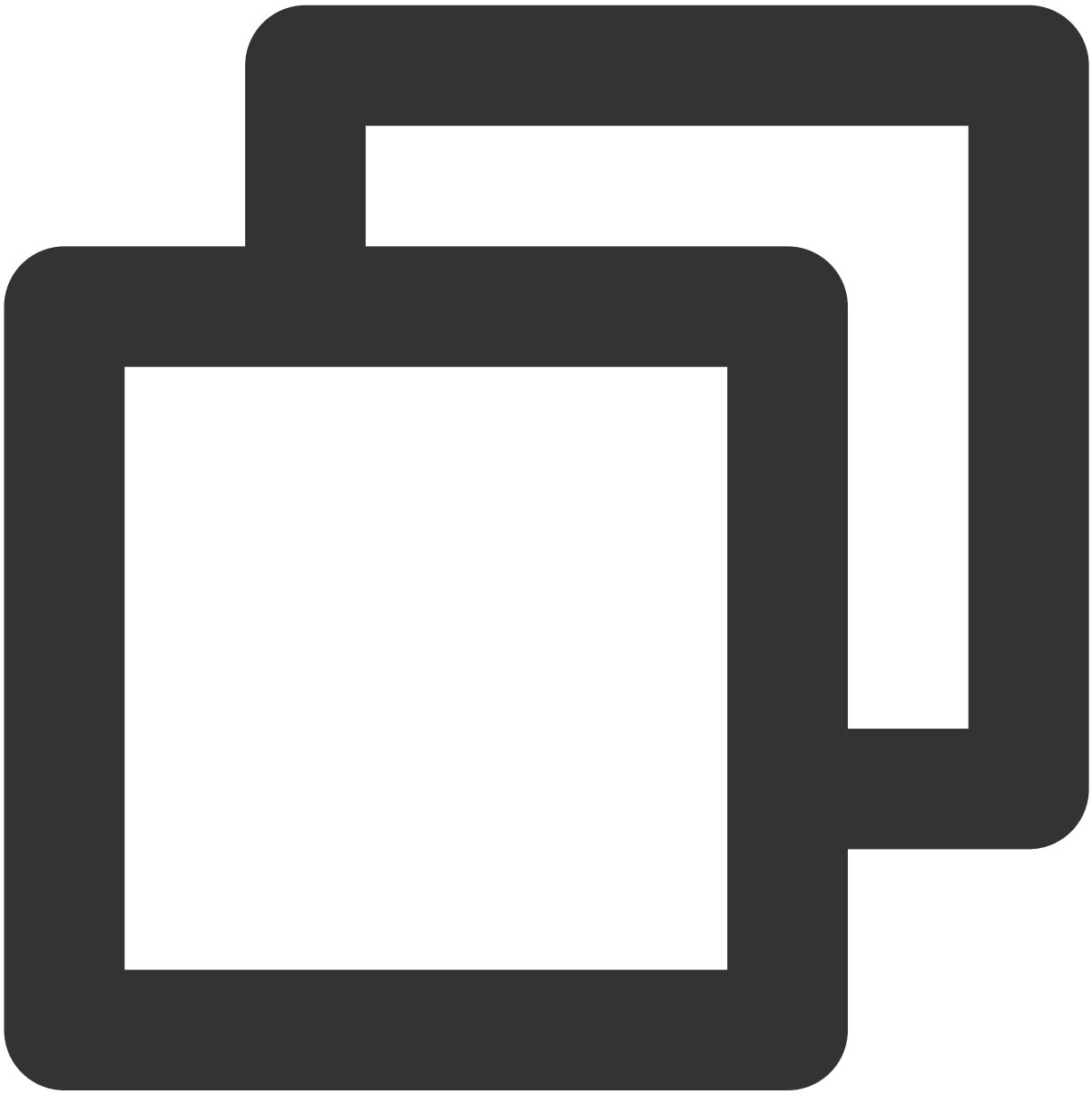
AWS SDK 的默认配置文件通常在用户目录下，可以参考 配置和证书文件。

在配置文件（文件位置是 `~/.aws/config` ）中添加以下配置信息：



```
[default]
s3 =
addressing_style = virtual
```

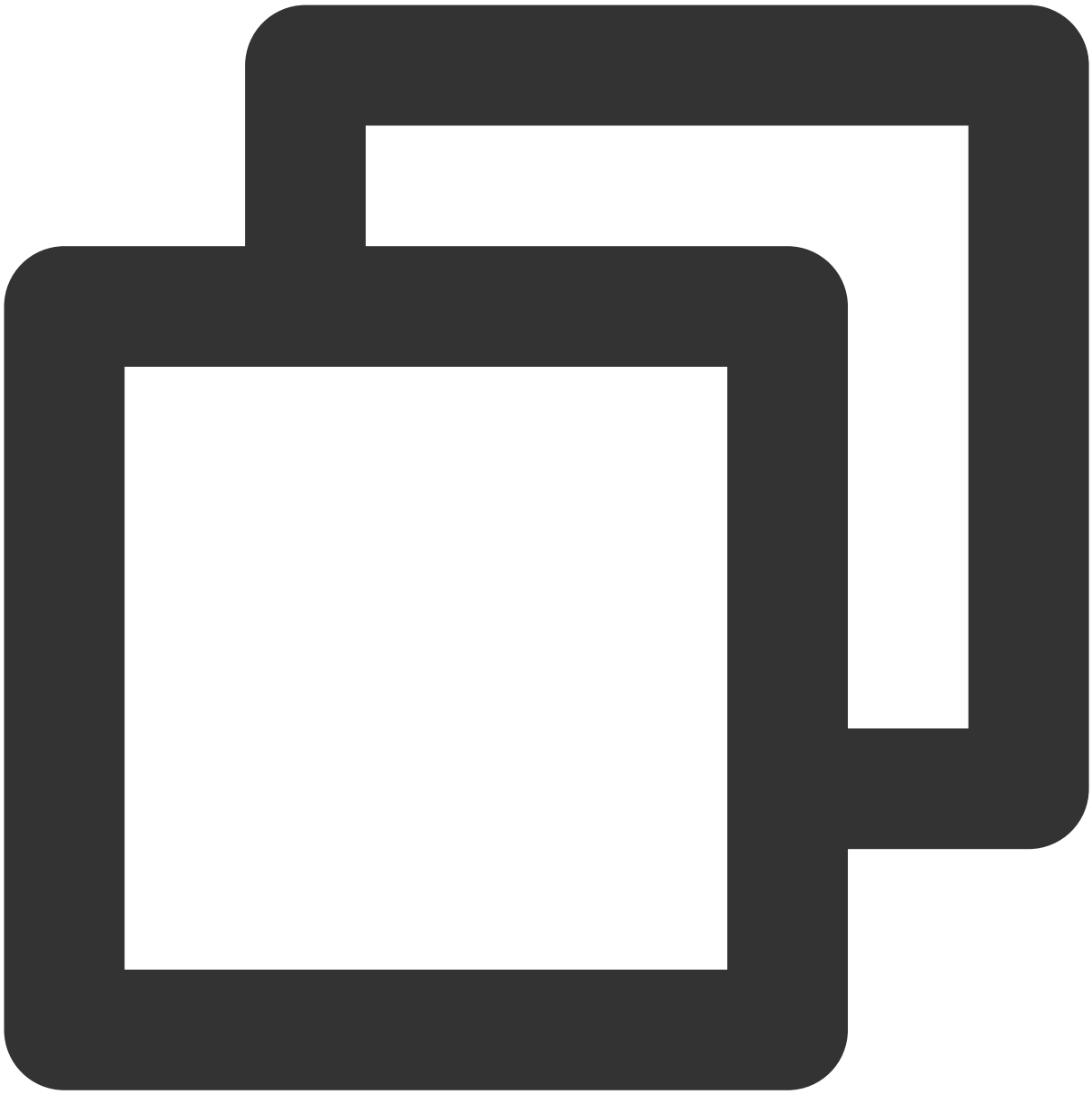
在证书文件（文件位置是 `~/.aws/credentials` ）中配置腾讯云的密钥：



```
[default]
aws_access_key_id = [COS_SECRETID]
aws_secret_access_key = [COS_SECRETKEY]
```

2. 代码中设置 Endpoint

以存储桶所在地域是 `ap-guangzhou` 为例，代码示例如下：



```
AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
    .withEndpointConfiguration(new AwsClientBuilder.EndpointConfiguration(
            "http://cos.ap-guangzhou.myqcloud.com",
            "ap-guangzhou"))
    .build();
```

Python

下面以 AWS Python SDK 1.9.205 版本为例，介绍如何适配以便访问 COS 服务。

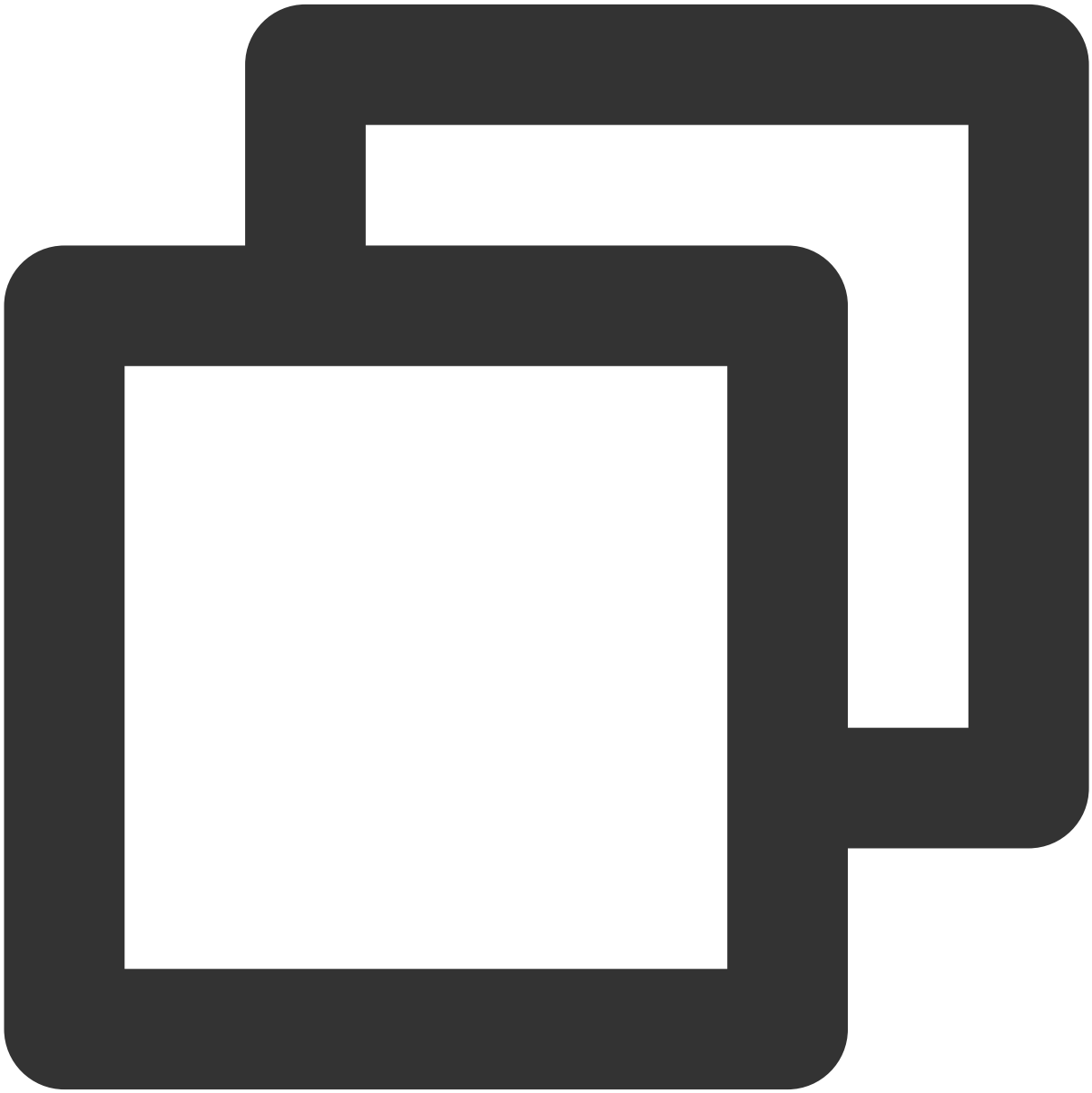
1. 修改 AWS 配置和证书文件

说明

下面以 Linux 为例，修改 AWS 配置和证书文件。

AWS SDK 的默认配置文件通常在用户目录下，可以参考 配置和证书文件。

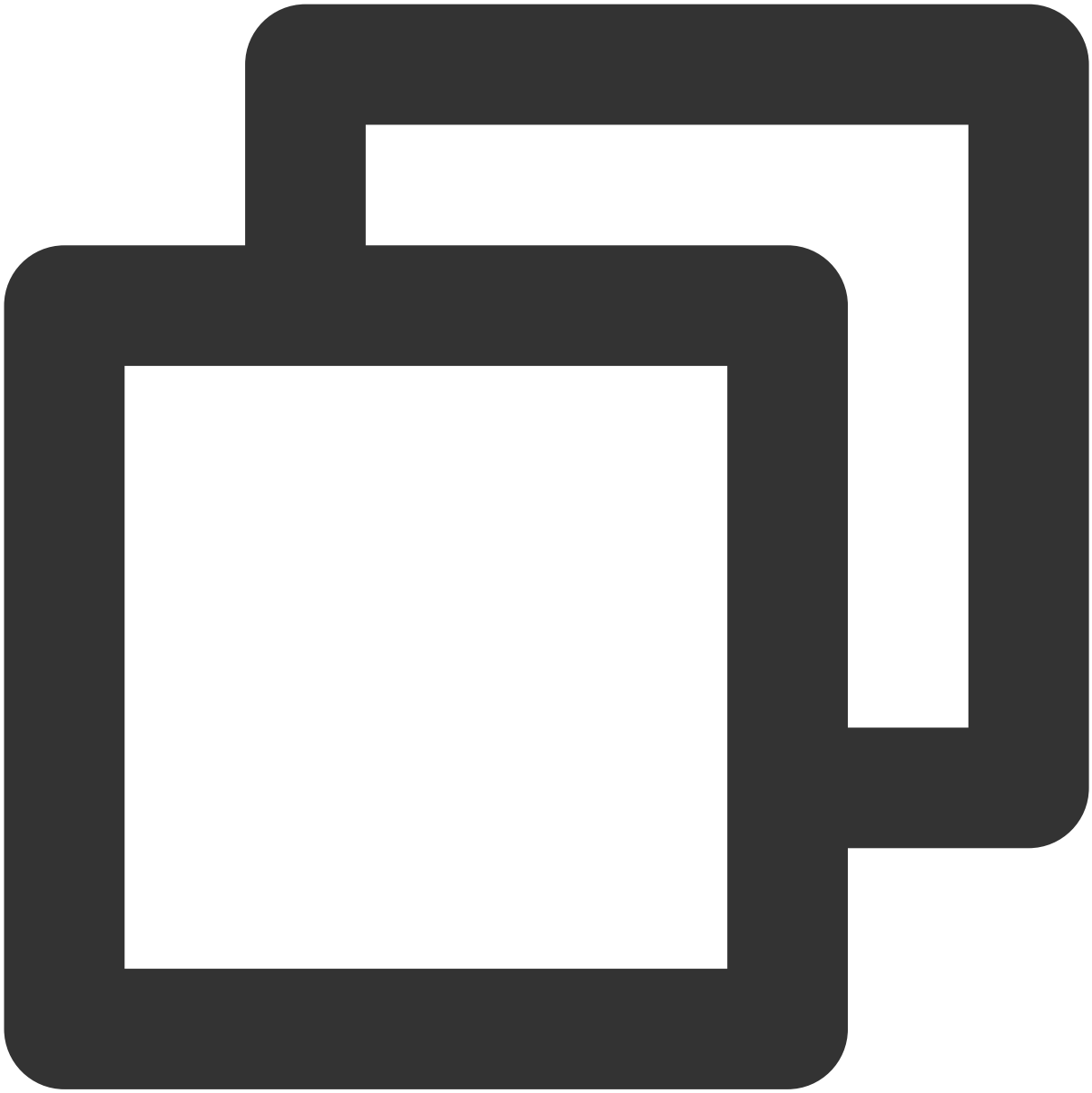
在配置文件（文件位置是 `~/.aws/config` ） 中添加以下配置：



```
[default]
s3 =
```

```
    signature_version = s3
    addressing_style = virtual
```

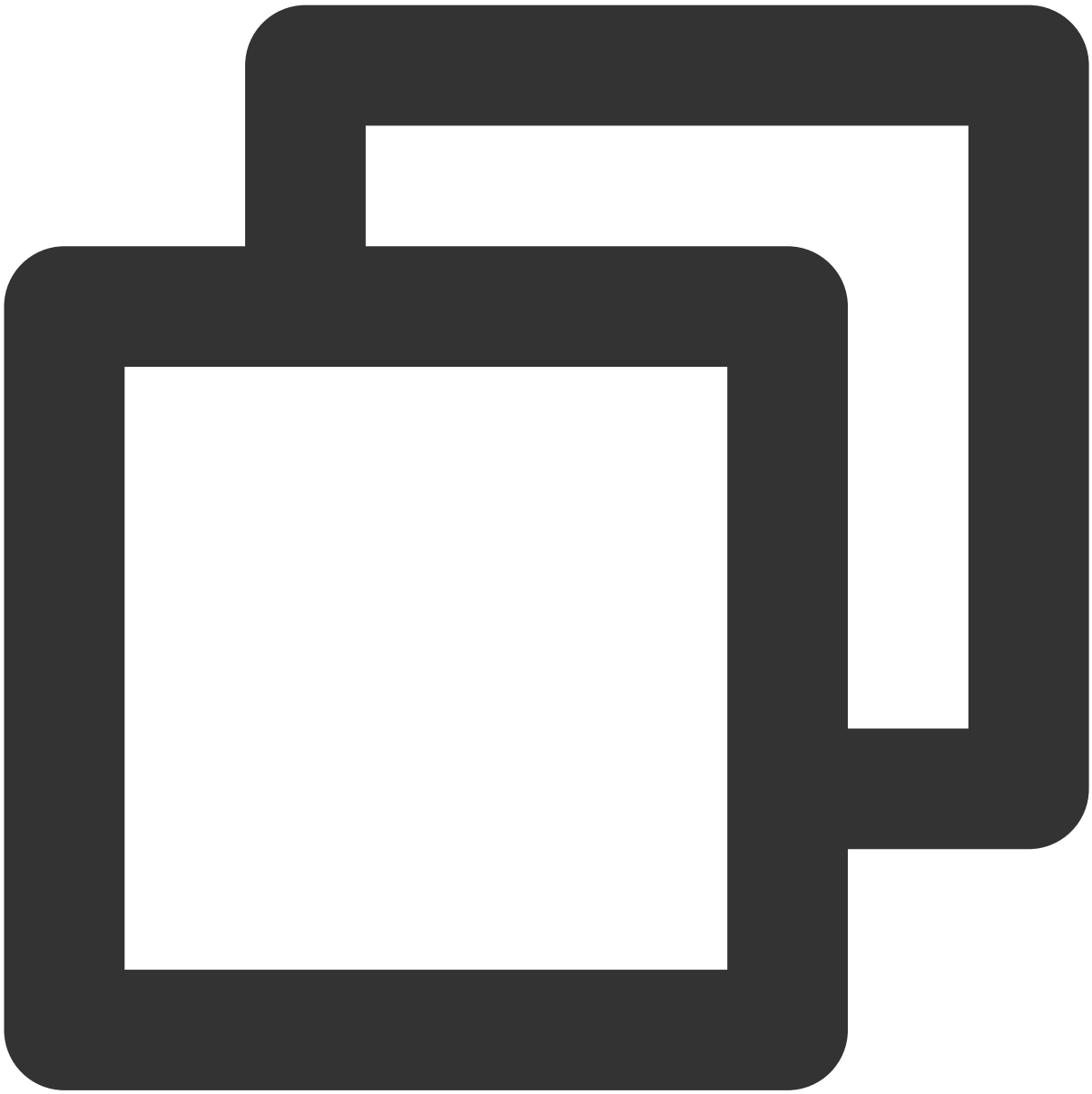
在证书文件（文件位置是 `~/.aws/credentials` ）中配置腾讯云的密钥：



```
[default]
aws_access_key_id = [COS_SECRETID]
aws_secret_access_key = [COS_SECRETKEY]
```

2. 代码中设置 Endpoint

以存储桶所在地域是 `ap-guangzhou` 为例：



```
client = boto3.client('s3', endpoint_url='https://cos.ap-guangzhou.myqcloud.com')
```

PHP

下面以 AWS PHP SDK 3.109.3 版本为例，介绍如何适配以便访问 COS 服务。

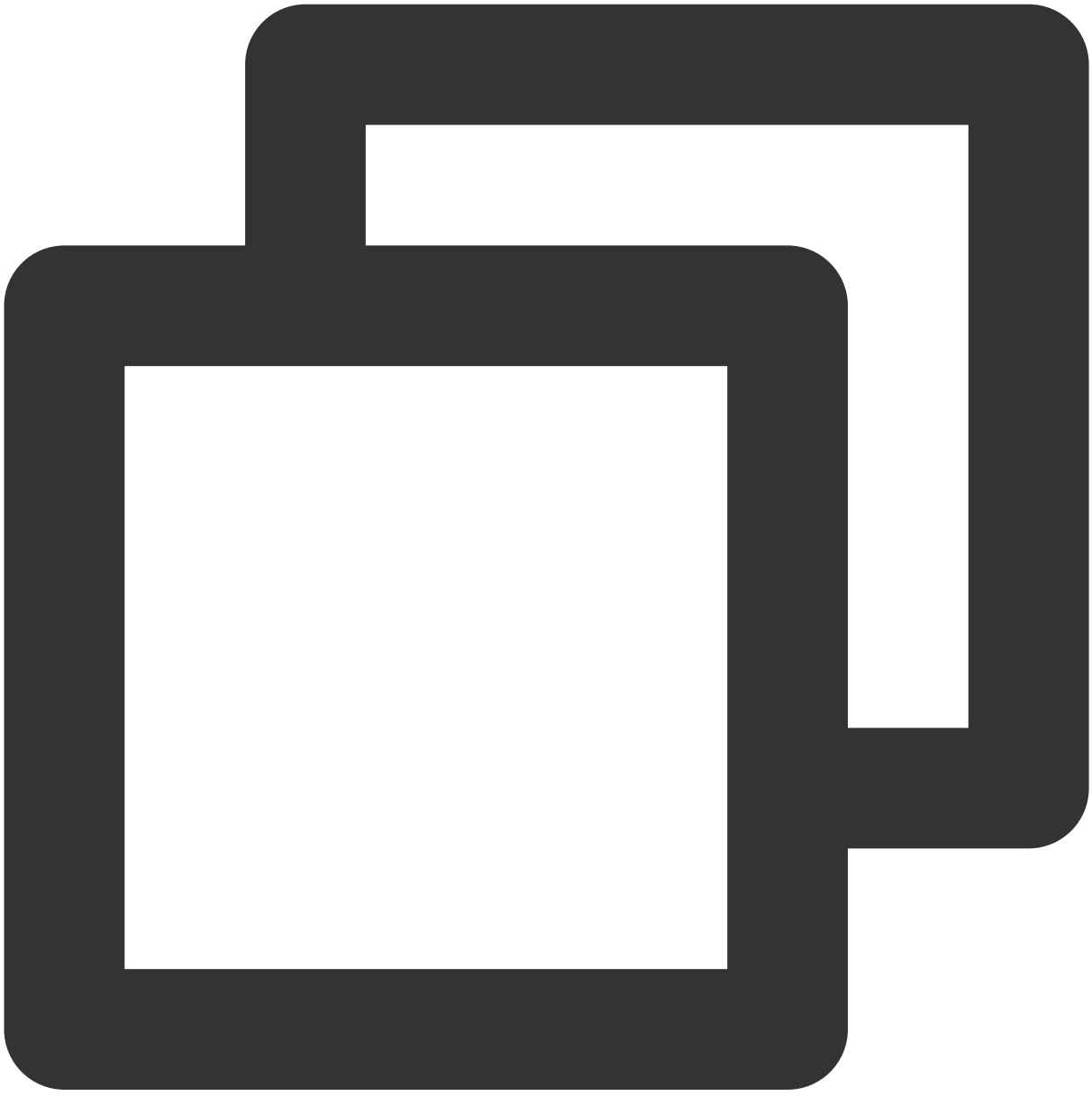
1. 修改 AWS 配置和证书文件

说明

下面以 Linux 为例，修改 AWS 配置和证书文件。

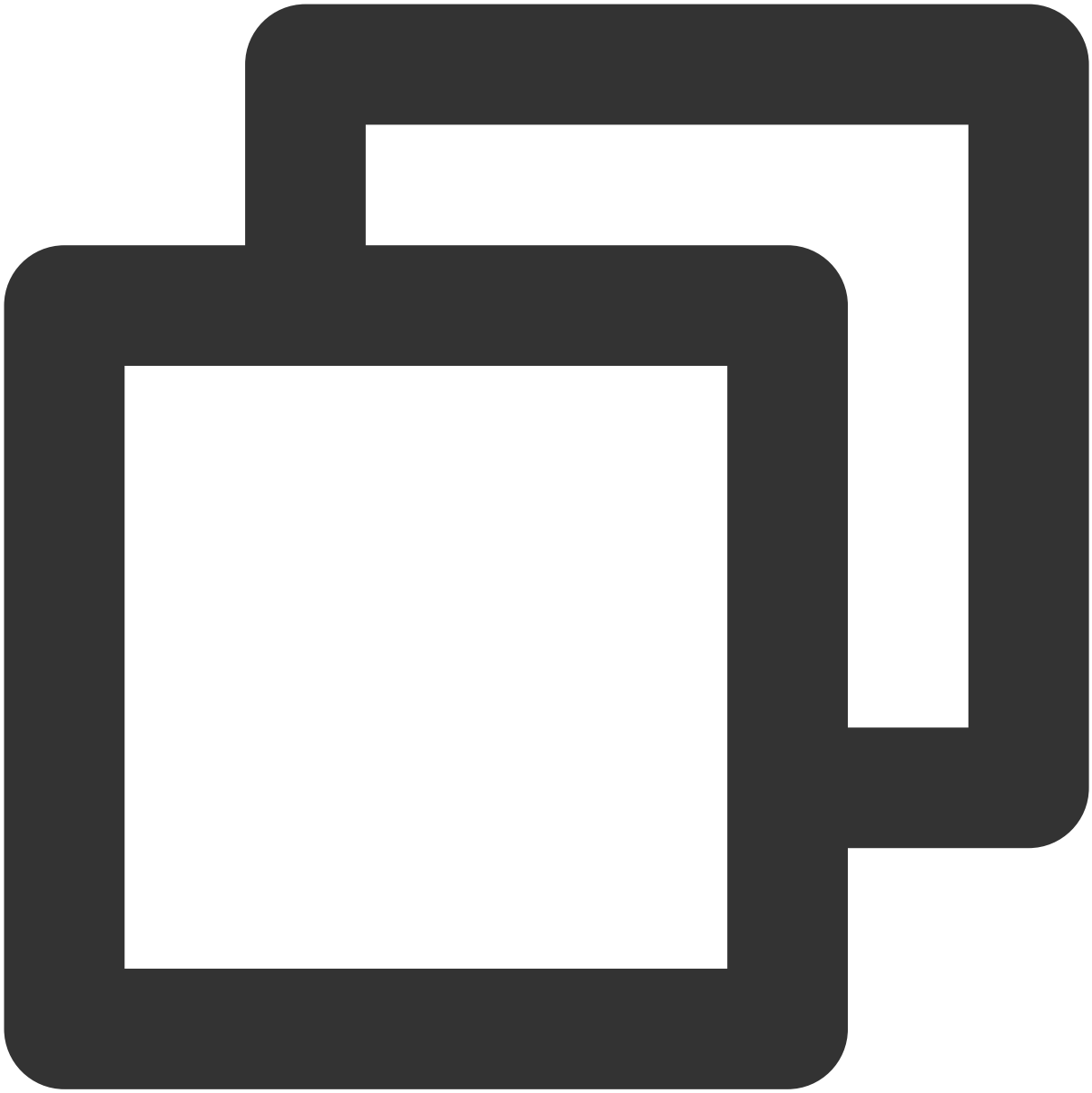
AWS SDK 的默认配置文件通常在用户目录下，可以参考 配置和证书文件。

在配置文件（文件位置是 `~/.aws/config` ） 中添加以下配置：



```
[default]
s3 =
addressing_style = virtual
```

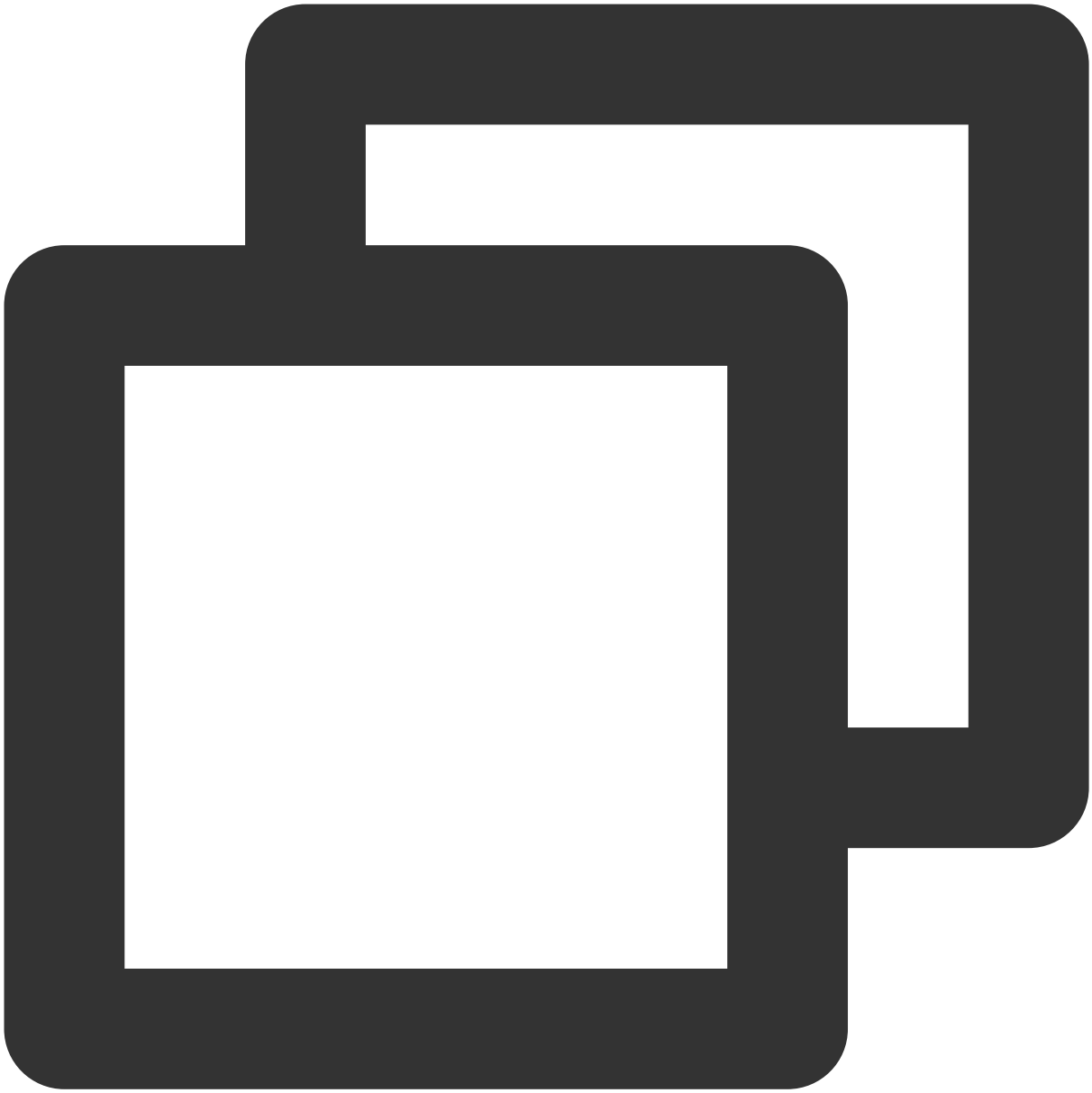
在证书文件（文件位置是 `~/.aws/credentials` ）中配置腾讯云的密钥：



```
[default]
aws_access_key_id = [COS_SECRETID]
aws_secret_access_key = [COS_SECRETKEY]
```

2. 代码中设置 Endpoint

以存储桶所在地域是 `ap-guangzhou` 为例：



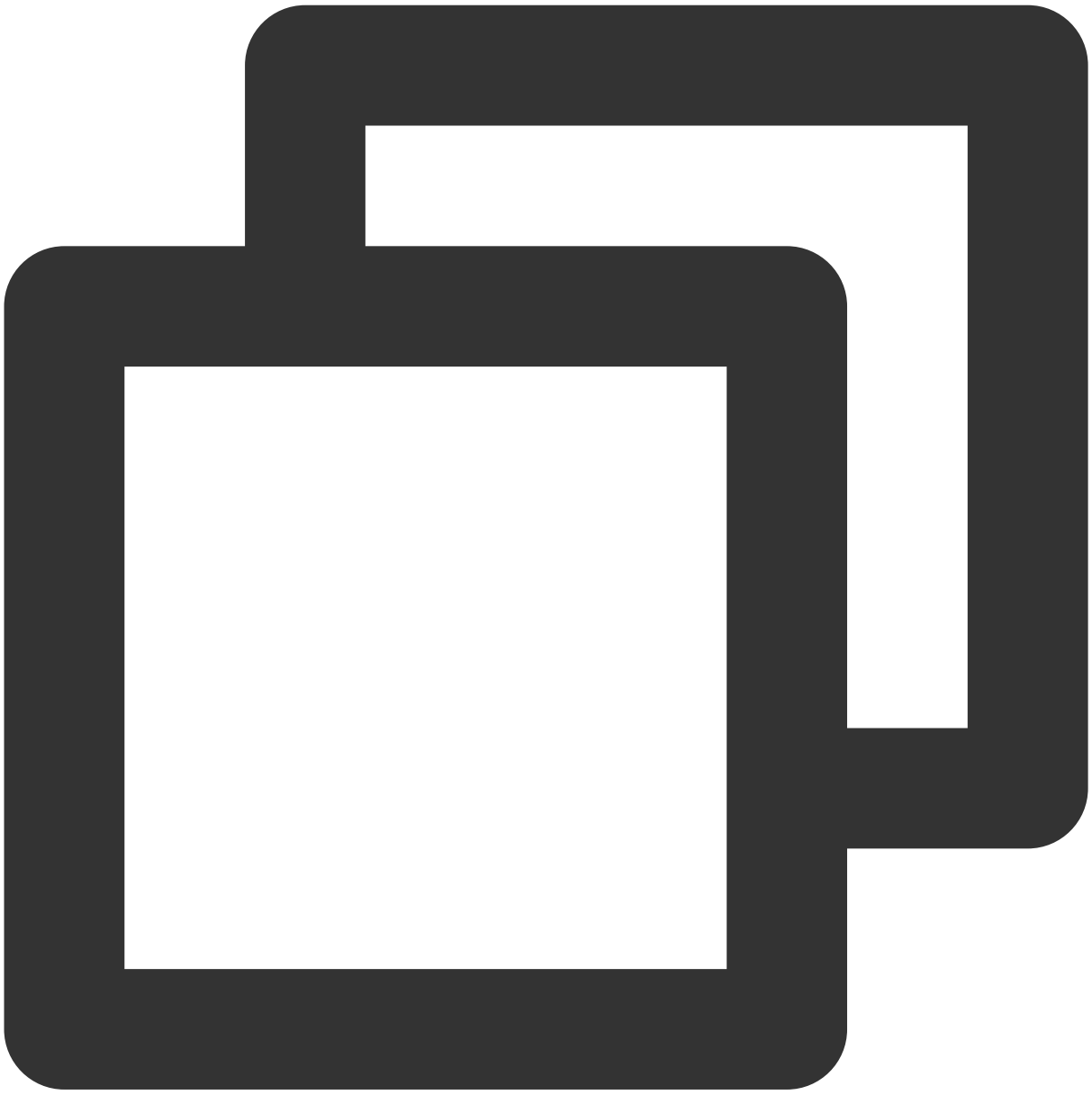
```
$S3Client = new S3Client([
  'region'          => 'ap-guangzhou',
  'version'         => '2006-03-01',
  'endpoint'        => 'https://cos.ap-guangzhou.myqcloud.com'
]);
```

.NET

下面以 AWS .NET SDK 3.3.104.12 版本为例，介绍如何适配以便访问 COS 服务。

初始化

初始化实例时设置腾讯云密钥和 Endpoint，以存储桶所在地域是 `ap-guangzhou` 为例：



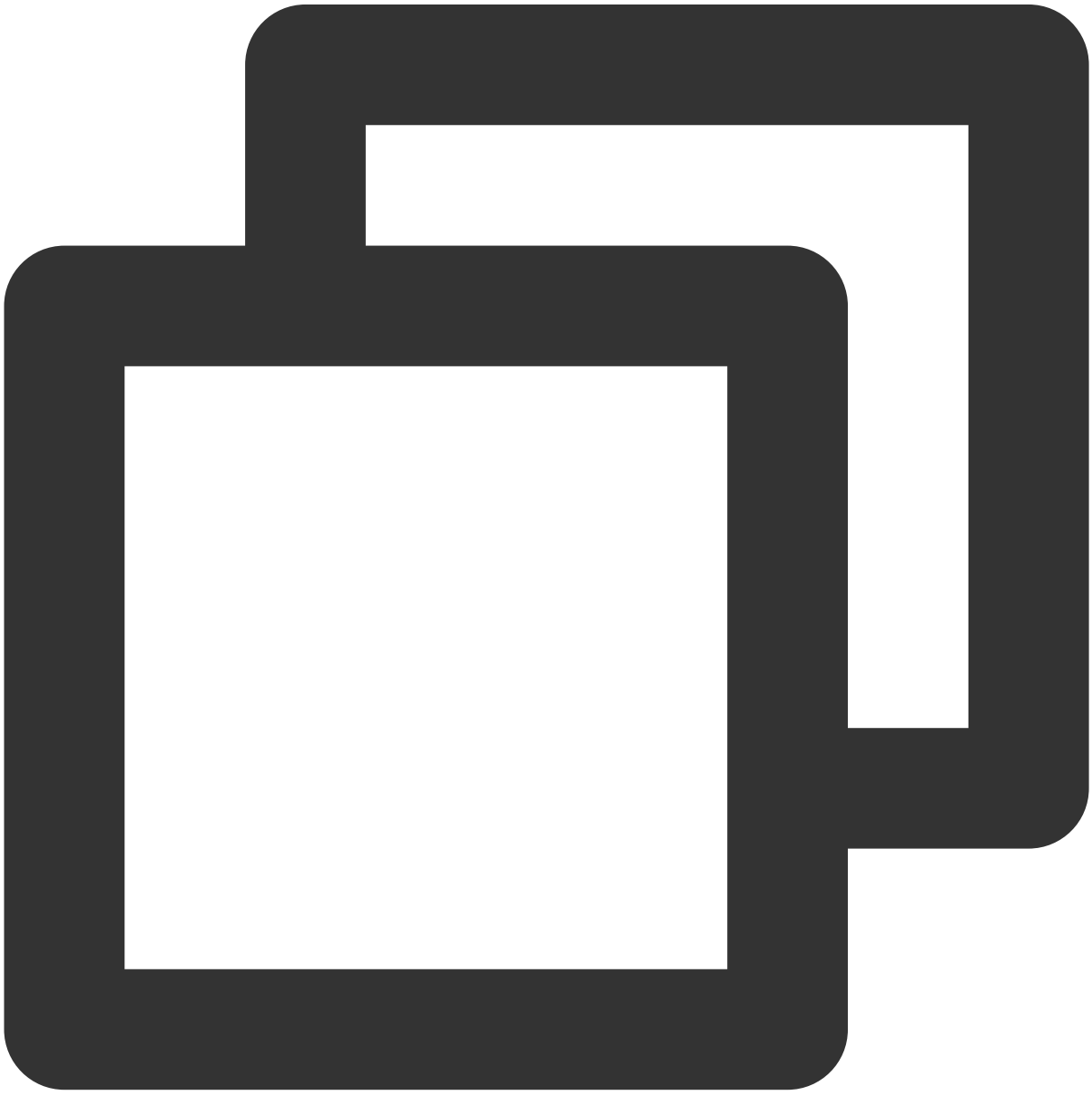
```
string sAccessKeyId = "COS_SECRETID";
string sAccessKeySecret = "COS_SECRETKEY";
string region = "ap-guangzhou";

var config = new AmazonS3Config() { ServiceURL = "https://cos." + region + ".myqclo
var client = new AmazonS3Client(sAccessKeyId, sAccessKeySecret, config);
```

Go

下面以 AWS Go SDK 1.21.9 版本为例，介绍如何适配以便访问 COS 服务。

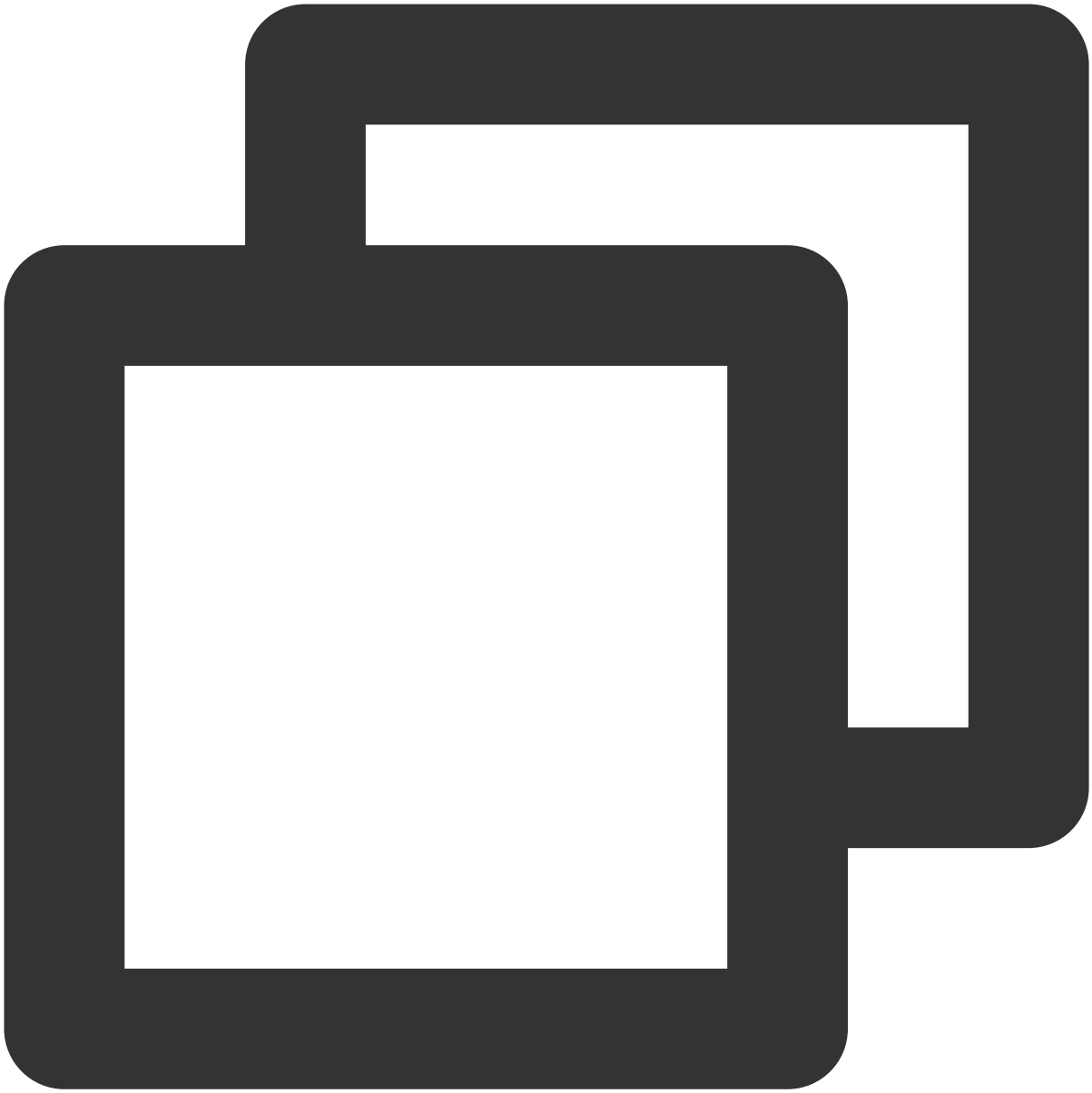
1. 根据密钥创建 session

以存储桶所在地域是 `ap-guangzhou` 为例：



```
func newSession() (*session.Session, error) {
    creds := credentials.NewStaticCredentials("COS_SECRETID", "COS_SECRETKEY", "")
    region := "ap-guangzhou"
    endpoint := "http://cos.ap-guangzhou.myqcloud.com"
    config := &aws.Config{
        Region:           aws.String(region),
        Endpoint:         &endpoint,
        S3ForcePathStyle: aws.Bool(true),
        Credentials:      creds,
        // DisableSSL:       &disableSSL,
    }
    return session.NewSession(config)
}
```

2. 根据 session 创建 server 发起请求



```
sess, _ := newSession()
service := s3.New(sess)

// 以上传文件为例
fp, _ := os.Open("yourLocalFilePath")
defer fp.Close()

ctx, cancel := context.WithTimeout(context.Background(), time.Duration(30)*time.Sec
defer cancel()

service.PutObjectWithContext(ctx, &s3.PutObjectInput{
```

```
    Bucket: aws.String("examplebucket-1250000000"),
    Key:    aws.String("exampleobject"),
    Body:   fp,
})
```

C++

下面以 AWS C++ SDK 1.7.68 版本为例，介绍如何适配以便访问 COS 服务。

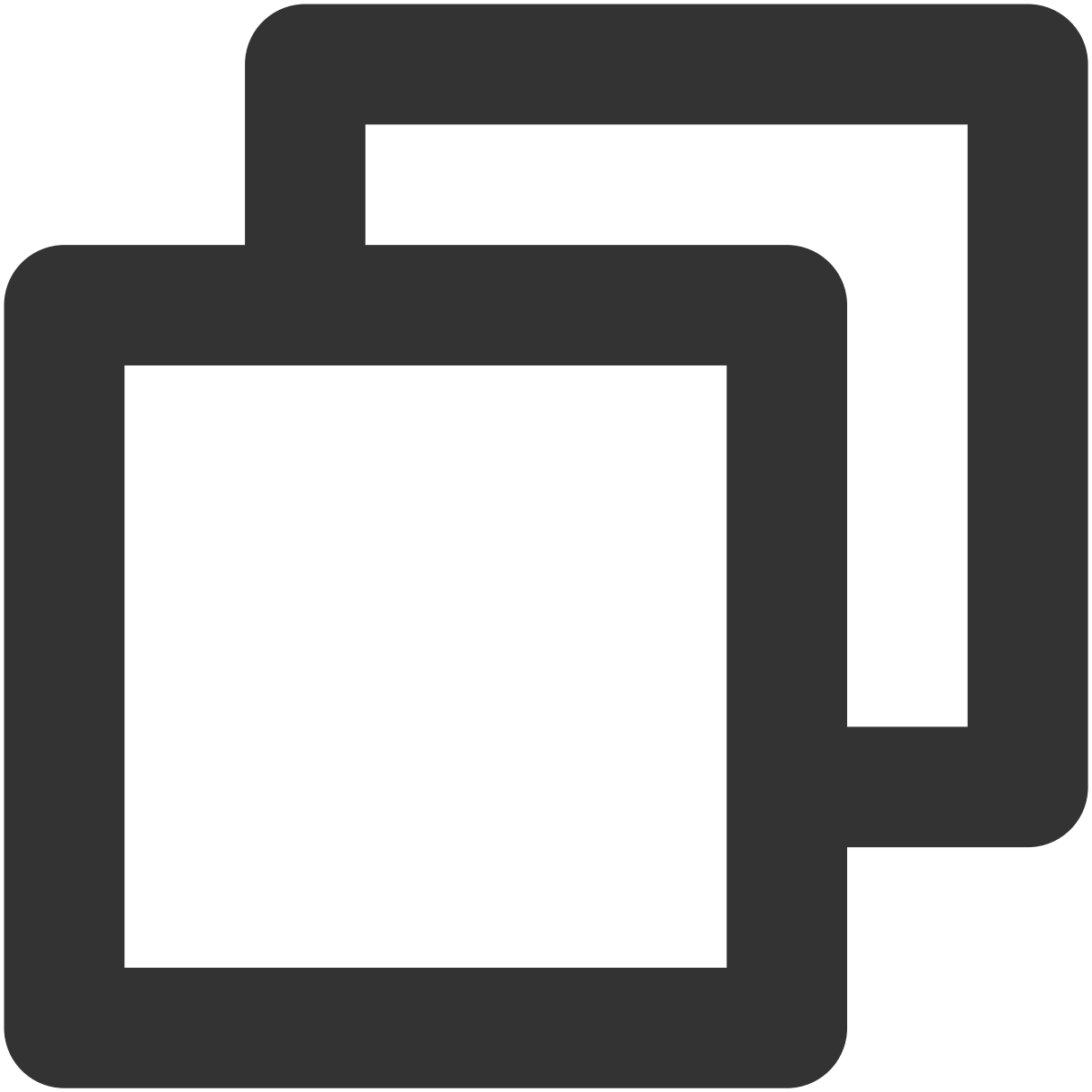
1. 修改 AWS 配置和证书文件

说明

下面以 Linux 为例，修改 AWS 配置和证书文件。

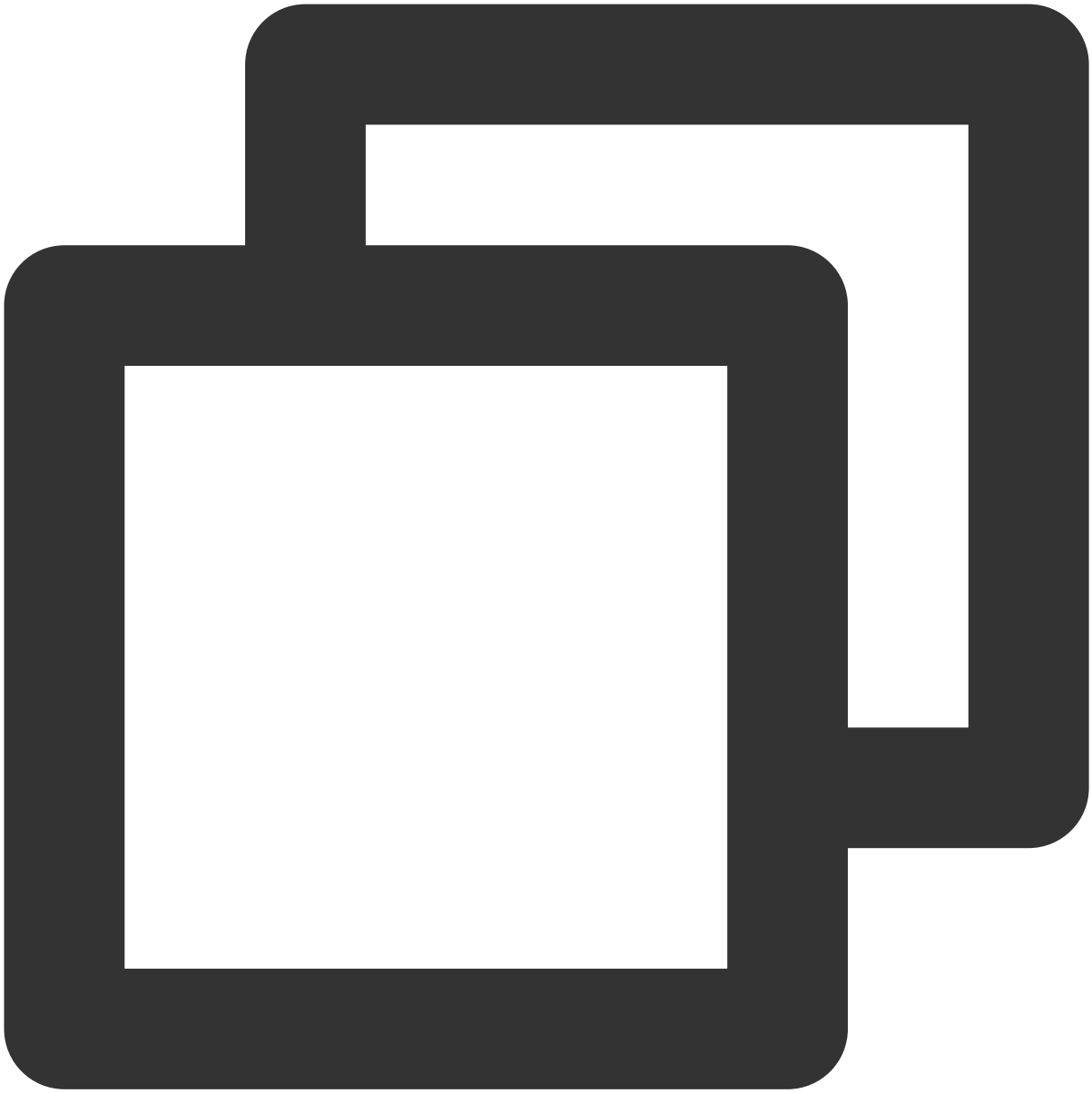
AWS SDK 的默认配置文件通常在用户目录下，可以参考 配置和证书文件。

在配置文件（文件位置是 `~/.aws/config` ） 中添加以下配置：



```
[default]
s3 =
addressing_style = virtual
```

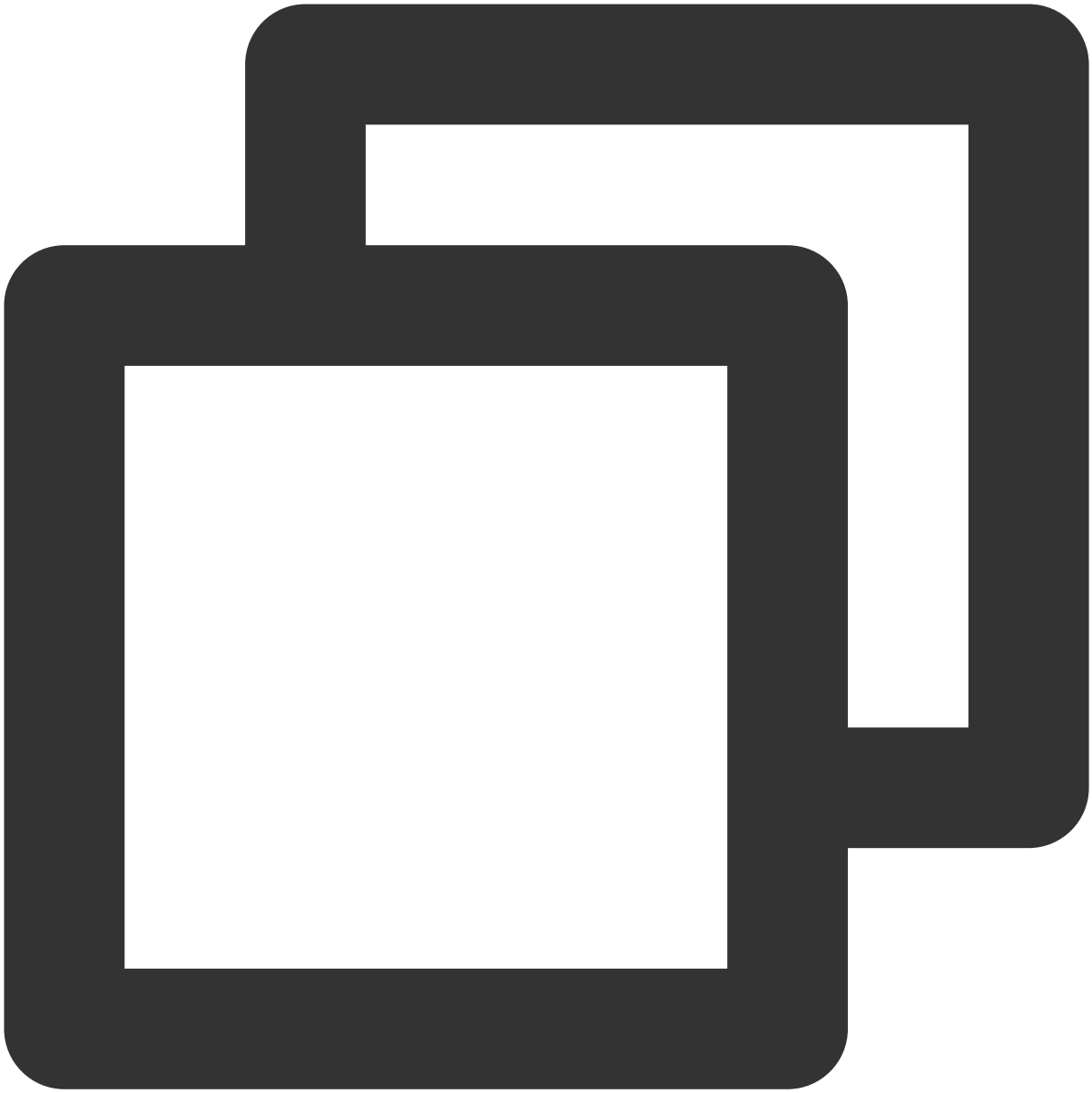
在证书文件（文件位置是 `~/.aws/credentials` ）中配置腾讯云的密钥：



```
[default]
aws_access_key_id = [COS_SECRETID]
aws_secret_access_key = [COS_SECRETKEY]
```

2. 代码中设置 Endpoint

以存储桶所在地域是 `ap-guangzhou` 为例，代码示例如下：



```
Aws::Client::ClientConfiguration awsCC;
awsCC.scheme = Aws::Http::Scheme::HTTP;
awsCC.region = "ap-guangzhou";
awsCC.endpointOverride = "cos.ap-guangzhou.myqcloud.com";
Aws::S3::S3Client s3_client(awsCC);
```

数据容灾备份
基于存储桶复制的容灾高可用架构

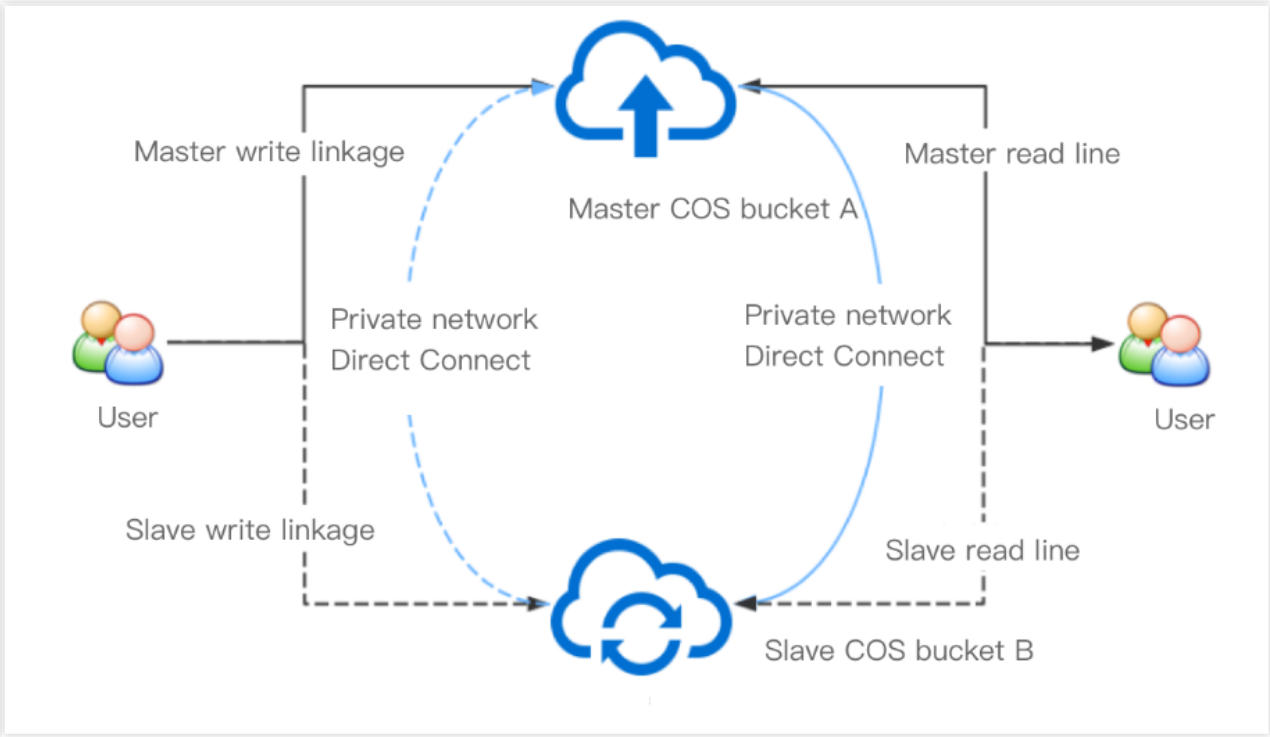
最近更新时间：2024-01-06 10:47:50

简介

腾讯云对象存储 COS 为客户提供了99.95%的可用性和99.999999999%的可靠性。然而，由于自然灾害、光纤故障等诸多不可控因素的存在，云上数据的可用性和可靠性均无法达到100%，同时，部分行业由于业务的特殊性，例如金融行业，需要保证业务高可用和高可靠性。
为了实现企业业务的连续性和稳定性，满足企业对高可用和高可靠性的需求，腾讯云对象存储提供了基于存储桶复制功能的数据容灾高可用方案。我们建议企业用云时，根据业务需要对云上数据进行容灾、备份，保障业务持续稳定运行。
本文主要介绍两个方面，首先介绍一种基于存储桶复制的云上业务主备切换的容灾方案，另一方面进一步介绍一种基于存储桶复制的高可用方案，通过存储桶复制、回源和 SCF、CDN 等多种产品和功能实现业务高可用。

基于存储桶复制的容灾备份方案

容灾需要满足三个要素：冗余（Redundance）、远距离（Remote）和数据全备份（Replication）。
冗余：即数据冗余，要求数据需要同时备份到另一个可用系统中。
远距离：指的是备份数据存储在相隔较远的另一个地域，因为灾害往往具有地理上的连续性，只有充分长的距离才能保障冗余数据的可用。
数据全备份：指的是备份数据零丢失。
COS 的存储桶复制功能可以实现增量数据的跨地域同步，用户上传的数据，根据其文件大小和地域距离远近，可以在几秒到几十分钟内拷贝到另一地域的存储桶中。基于存储桶复制，可以实现数据的异地冗余备份，从而实现业务容灾。有关存储桶复制的介绍，可参见 存储桶复制概述。开启存储桶复制需要先开启版本控制功能，有关版本控制的介绍可参见 版本控制概述。
基于存储桶复制的容灾备份架构示意图如下：



在这一架构下，客户的存储桶 A 和存储桶 B 互为主备。假设企业客户的数据存储在存储桶 A 上，另一地域的存储桶 B 是备用存储桶。该企业为了保障业务连续性和稳定性，为存储桶 A 和存储桶 B 分别配置了存储桶复制规则。在存储桶复制规则生效的情况下，存储桶 A 的增量数据会自动复制到存储桶 B 中，存储桶 B 的增量数据同样会自动复制到存储桶 A 中。

注意

存储桶 A 中的增量数据复制到存储桶 B 后，虽然是存储桶 B 中的增量数据，但不会再被复制到存储桶 A 中。

正常情况下，企业的主读写请求链路均指向存储桶 A，所有增量数据将被自动增量同步复制到存储桶 B 中作为备份数据。客户侧可以在上传或者下载程序中加入网络质量检测的模块，在检测到主存储桶 A 宕机时，迅速将读写请求链路切换到备存储桶 B 中。

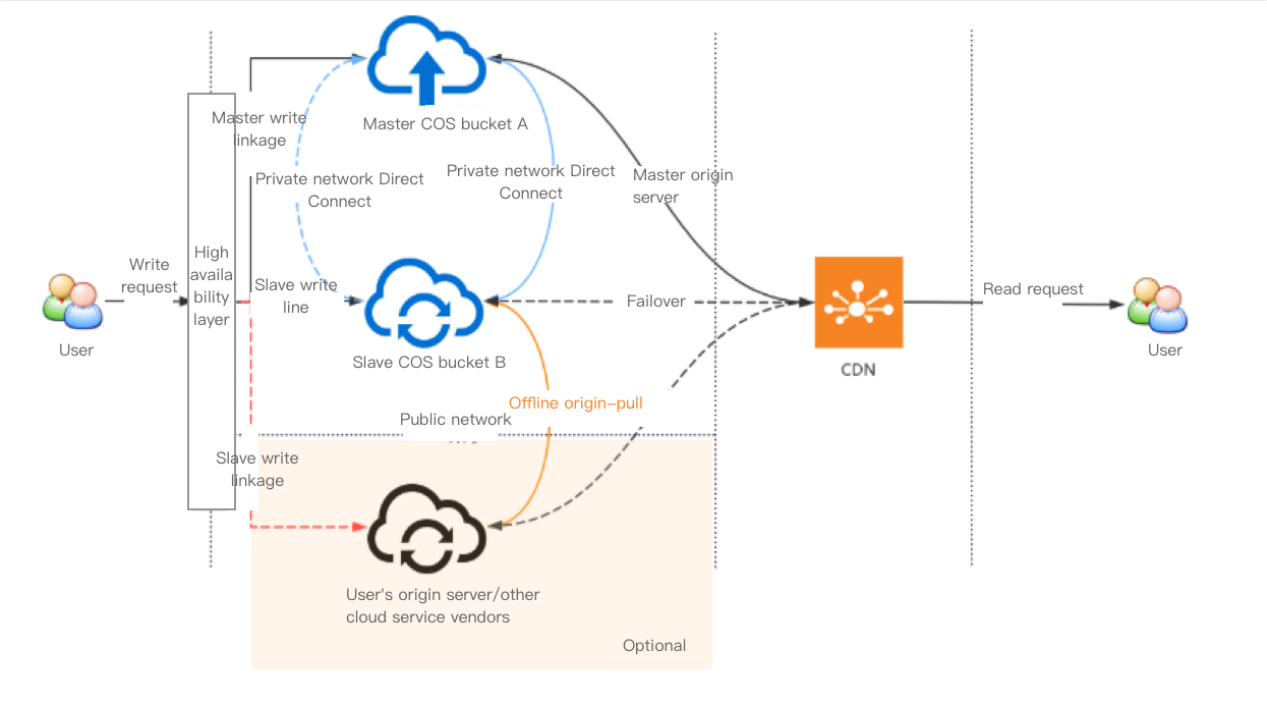
注意

网络质量检测可以基于腾讯云云函数 SCF 实现，通过对云函数代码进行修改，将拨测地址修改为主备存储桶的域名，同时根据业务需求将告警代码片段修改为其他业务所需的措施。

基于存储桶复制的高可用方案

上文介绍了一种基于存储桶复制的容灾备份方案，该方案能够利用云上已有产品和功能实现数据备份和容灾切换的工作。但真实业务运行状态可能复杂多样，上述的容灾备份方案未必能保障业务的高可用。因此，本小节提出一种基于存储桶复制的高可用方案，通过存储桶复制、回源和 SCF、CDN 等多种产品和功能实现业务高可用。

基于存储桶复制的业务高可用架构示意图如下：



这一架构主要分为以下几个层次：

高可用层：集成网络检测和业务调度，根据链路的连通率等指标进行链路切换，用户可以根据 SCF 实现（可参照上一小节介绍），也可以根据业务需求在客户端自行实现。

存储层：一般情况下由 COS 不同地域的存储桶组成；用户也可以通过 设置回源策略，**引入自有源站或者其他云厂商**上的存储桶，进一步保障数据的一致性。

CDN 层：通过腾讯云 CDN 海量边缘节点提供就近访问的功能，可以避免用户直接访问源站数据，保证源站数据安全。

这一架构保障业务高可用的方式阐述如下：

1. 正常情况下，企业的主写请求链路均指向存储桶 A，所有增量数据将被自动同步复制到存储桶 B 中作为备份数据。

2. 当主存储桶 A 的链路不通时（例如拨测质量下降或者检测到上传失败），则客户端可以将写请求链路切换至主存储桶 B，此时所有增量数据同样将被自动同步复制到存储桶 A 中。

3. 客户还可以选择在自有源站或者其他云厂商上先备份一份冗余数据，同时给存储桶 B 配置回源策略。假设在极端情况下，主存储桶 A 和 B 链路都同时无法连通，那么在上传数据到存储桶 B 失败的情况下，存储桶 B 可以从源站拉取数据。

注意

全量冗余备份数据成本较高，客户也可以选择只冗余备份热数据（例如仅数小时内上传的文件），以减少数据存储成本。

如果您选择了源站作为高可用架构中的一部分，那么您在设计该架构时请注意评估源站带宽以及其限制可能带来的影响。

4. 从存储桶读取数据可以通过直接访问存储桶实现，也可以为存储桶 绑定 CDN 加速域名，通过腾讯云 CDN 边缘节点为客户提供就近访问的能力。如果您的业务数据涉及到内容分发场景，或者不希望您的用户直接访问到您的存储桶，我们推荐您搭配使用 腾讯云 CDN。

说明

如果您直接从存储桶中读取数据，那么需要客户端支持 follow HTTP 302。

腾讯云 CDN 提供近千个边缘节点，能够为用户提供邻近的访问节点，提高读取速率。您可以为 CDN 绑定多个源站互为主备，保障高可用，可参见 源站配置 进行配置。

如果您希望尽可能保障源站安全，设置源站私有读写并开启 CDN 回源鉴权，能够让您的用户能匿名访问缓存在 CDN 边缘节点上的数据，同时保障源站数据安全。

参考文档

以下文档可能为您实现容灾高可用架构提供帮助：

版本控制概述

存储桶复制概述

设置回源

CDN 加速配置

源站配置

域名接入

云上数据备份

最近更新时间：2024-01-06 10:47:50

简介

本文主要为用户概括介绍通过哪些方式可以将存在云端的数据进行备份。腾讯云对象存储 COS 为用户提供了以下三种方式，方便用户对存放在 COS 中的数据进行备份：
基于跨地域复制的容灾备份方案
批量数据复制方案（COS 批量处理功能）
基于数据迁移工具 MSP 的迁移备份方案

基于跨地域复制的容灾备份方案

跨地域复制功能是针对存储桶的一项配置，通过配置跨地域复制规则，可以在不同存储地域的存储桶中自动、异步地复制**增量对象**。启用跨地域复制后，COS 将精确复制源存储桶中的对象内容（例如对象元数据和版本 ID 等）到目标存储桶中，复制的对象副本拥有完全一致的属性信息。
具体可参见：基于跨地域复制的容灾高可用架构。

适用场景

异地容灾：COS 为对象数据提供了11个9的可用性，但仍然存在各种不可抗因素如战争、自然灾害等因素导致数据丢失。如果您无法忍受因数据丢失带来的损失，希望显式地在不同地域维护一份数据副本，那么您可以通过跨地域复制实现数据的异地容灾，当某个数据中心因为不可抗因素损毁时，另一个地域的数据中心仍然可以提供副本数据以供您使用。
合规性要求：COS 默认在物理盘中为数据提供多副本和纠删码等方式保障数据的可用性，但某些行业中可能存在合规性要求，规定您需要在不同的存储地域间保存数据副本。因此启用跨地域复制，可以实现在不同存储地域间复制数据以满足这些合规性要求。
减少访问延迟：当您的客户在不同地理位置访问对象时，您可以通过跨地域复制，在与客户地理位置最近的可用存储地域中维护对象副本，最大限度上缩短客户的访问延迟，有利于提高您的产品体验。
操作原因：如果您在两个不同地域中均具有计算集群，且这些计算集群需要处理同一套数据，则您可以通过跨地域复制在这两个不同的地域中维护对象副本。
数据迁移与备份：您可以根据业务发展需要，将业务数据从一个可用地域复制到另一个可用地域，实现数据迁移和数据备份。

使用方法

请参见：设置跨地域复制 控制台文档。

批量数据复制方案

COS批量处理功能为用户提供了对象级别的大规模批量复制操作。

适用场景

可能由于某些业务原因，您需要将已有的一个存储桶中的所有数据进行备份至另一个存储桶中，以保证数据的可用性和可靠性。

使用方法

请参见：批量处理 控制台文档。

使用说明

批量复制对象操作需要结合清单功能使用，有关清单的说明可参见 清单功能概述。您可以将指定对象从源存储桶中批量复制到目标存储桶中。目标存储桶所属地域可以相同或者不同。批量复制对象操作支持自定义配置与 PUT Object - copy 相关的参数，这些配置信息可以影响到副本的元数据信息或者存储类型等信息。有关 PUT Object-copy 的更多介绍，请参见 PUT Object - copy。

说明：

所有待处理对象均需处于同一存储桶中。

在一项批处理任务中，仅可配置一个目标存储桶。

您需要被授予从源存储桶中读取对象的权限以及往目标存储桶中写入对象的权限。

待处理的对象不得超过5GB。

不支持检查 ETags 和使用用户自定义密钥的服务端加密。

如果目标存储桶中未开启版本控制，且存在同名对象文件，COS 将执行覆盖对象文件的操作。

数据迁移备份方案

腾讯云迁移服务平台 MSP 为用户提供方便快捷的资源迁移方案，用户建立迁移任务后，还可以通过迁移服务控制台查看迁移进度、迁移报告等信息。具体操作请参见：腾讯云 COS 间迁移。

本地数据备份

最近更新时间：2024-01-06 10:47:50

简介

本文主要为用户概括介绍通过哪些方式可以将本地数据进行备份上云，腾讯云对象存储 COS 为用户提供了以下三种备份方式，方便用户将本地的数据备份至 COS 存储桶中：

COSBrowser 的文件同步备份

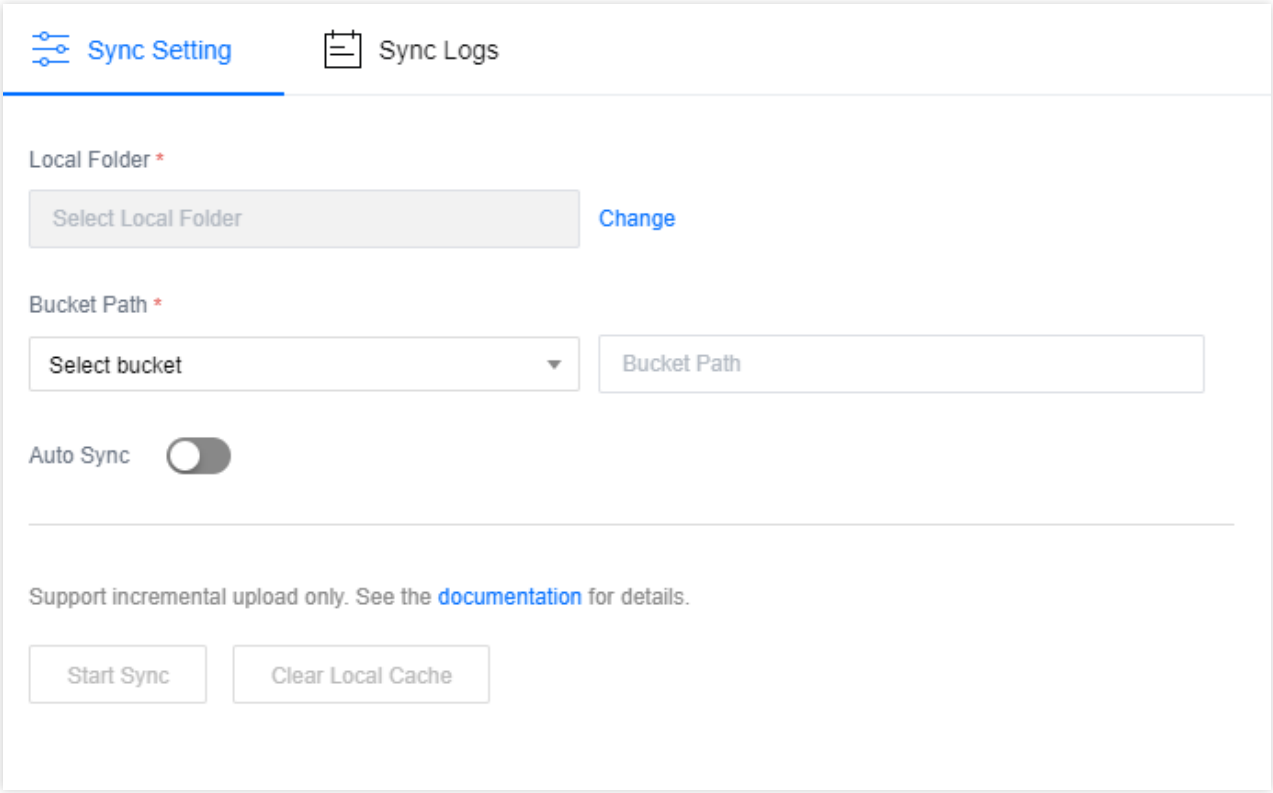
COS Migration 线上迁移备份

CDM 线下迁移备份

文件同步备份

COSBrowser 是腾讯云对象存储推出的用于管理云上文件的可视化界面工具，让您可以使用更简单的交互轻松实现对 COS 资源的查看、传输和管理。COSBrowser 有桌面端和移动端两种，详情可参见 COSBrowser 简介。

COSBrowser 桌面端集成了文件同步功能，可以通过关联本地文件夹与存储桶，实现本地文件实时同步上传至云端。



使用方法

请参见：文件同步使用说明。

线上迁移方案

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。

适用场景

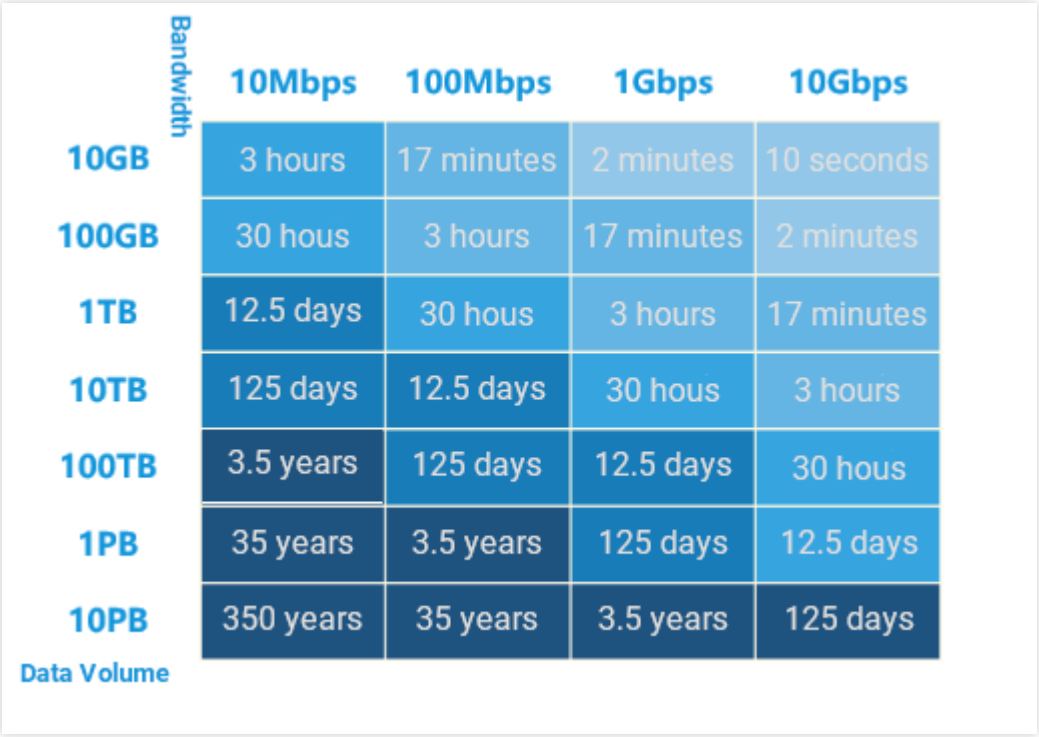
对于拥有本地 IDC 的用户，COS Migration 帮助用户将本地 IDC 的海量数据快速迁移至对象存储 COS。

使用方法

请参见：COS Migration。

线下迁移方案

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。
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。



Data Volume \ Bandwidth	10Mbps	100Mbps	1Gbps	10Gbps
10GB	3 hours	17 minutes	2 minutes	10 seconds
100GB	30 hous	3 hours	17 minutes	2 minutes
1TB	12.5 days	30 hous	3 hours	17 minutes
10TB	125 days	12.5 days	30 hous	3 hours
100TB	3.5 years	125 days	12.5 days	30 hous
1PB	35 years	3.5 years	125 days	12.5 days
10PB	350 years	35 years	3.5 years	125 days

使用方法

请参见：云数据迁移 CDM。

域名管理实践
存储桶切换自定义域名

最近更新时间：2024-07-05 18:37:28

背景

为保证整体服务的安全性和稳定性，2024年1月1日后创建的存储桶，若使用 COS 默认域名访问对象，任意类型文件不支持预览，apk/ipa 类型文件不支持下载。详情可参考 COS 存储桶域名使用安全管理通知。

2024年1月1日后创建的存储桶，若您希望通过浏览器直接预览文件或下载存储桶内的 apk/ipa 类型对象，建议用户使用自定义域名访问对象。2024年1月1日前创建的存储桶，存储桶默认域名的预览、下载行为不受影响，但为了获得更好的服务稳定性，建议用户优先使用自定义域名。

本文介绍如何为存储桶配置自定义域名，从访问存储桶默认域名切换为访问自定义域名。

第一步：域名注册与备案

首先，用户需要准备一个已备案的自定义域名。
域名注册：您如果没有自定义域名，可前往 域名注册 购买域名。
域名备案：如果您的自定义域名用于配置到中国大陆地域的存储桶，则必须经过备案。

第二步：为存储桶配置自定义域名

1. 准备好自定义域名后， 登录 对象存储控制台，进入存储桶列表，选择您需要配置的存储桶。
2. 进入存储桶详情页面，选择**域名与传输管理 > 自定义源站域名**。
3. 单击**添加域名**，配置域名信息：
域名：输入准备好的自定义域名。
源站类型：分为以下几种。
默认源站：如果您希望将自定义域名用作默认源站，请选择默认源站。
静态网站源站：如果您希望将自定义域名用作静态网站，请先为存储桶开启静态网站功能，然后选择静态网站源站。
全球加速源站：如果您希望将自定义域名用作全球加速，请先为存储桶开启全球加速功能，然后选择全球加速源站。

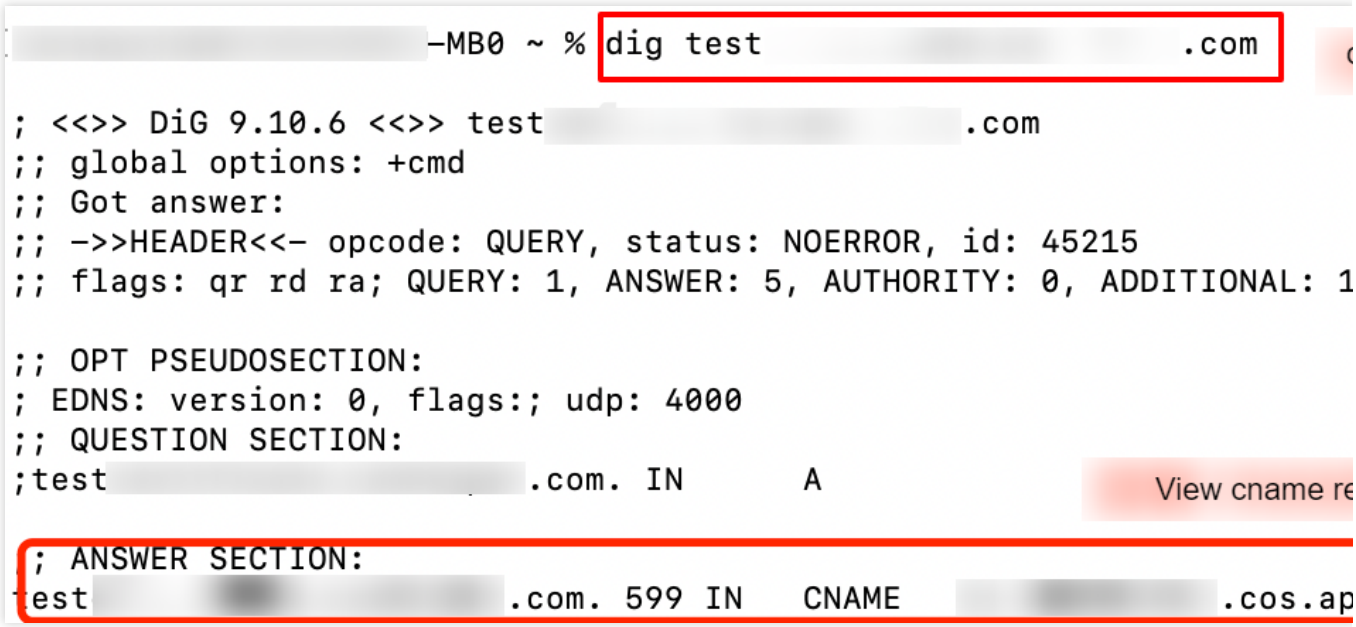
4. 配置 HTTPS 证书。如果需要使用 HTTPS 协议访问，需要为自定义域名配置证书。
如果您需要使用自有证书，需要将证书内容和私钥内容粘贴到指定输入框。
如果您使用的是腾讯云申请的证书，可直接在弹窗中选择当前账号下已有的腾讯云证书。
5. 自定义域名配置完成后，记录 CNAME 一栏的信息（例如 bucket-1250000000.cos.ap-beijing.myqcloud.com），用于后续配置域名解析。

第三步：配置域名解析

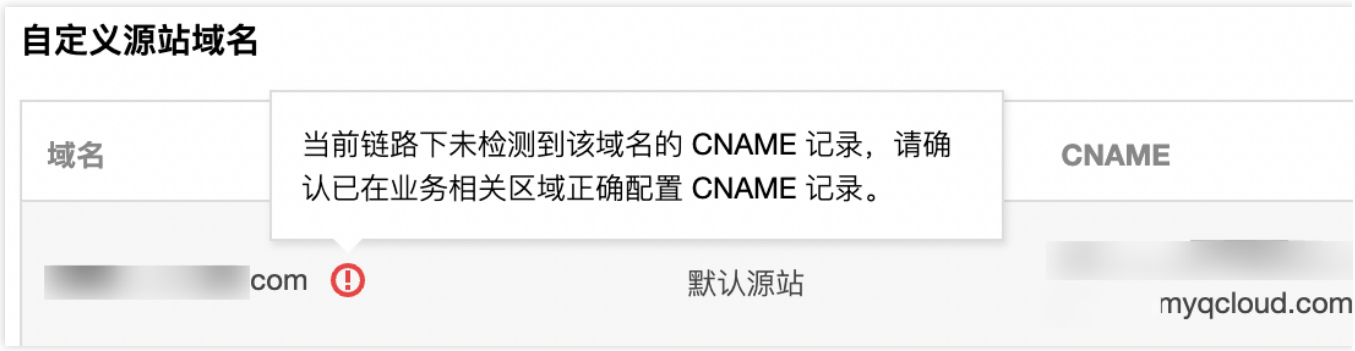
腾讯云域名

如果域名 DNS 厂商是腾讯云，可以前往 云解析 DNS 控制台，配置 CNAME 解析记录。

1. 前往 云解析 DNS 控制台，找到对应的域名，单击**解析**按钮。
2. 单击**快速添加解析**，为该域名添加一条解析记录。
3. 在弹窗中，选择**网站解析**，网站地址选择**域名映射(CNAME)**。输入第二步记录的 CNAME 信息输入，例如 bucket-1250000000.cos.ap-beijing.myqcloud.com。
4. 解析记录需要一定时间生效，可以通过 dig 命令或 对象存储控制台 查看解析是否成功更生效。验证方法如下：
在命令行窗口输入命令：`dig mydomain.com`，查看 CNAME 记录是否正确生效。（使用时请将 `mydomain.com` 替换为自定义域名）



登录 对象存储控制台，查看存储桶自定义域名，如果域名的 CNAME 没有成功生效，会出现相应提示。



其他厂商域名

如果域名 DNS 厂商不是腾讯云，需要前往相应的 DNS 服务配置 CNAME 解析记录。

第四步：访问自定义域名

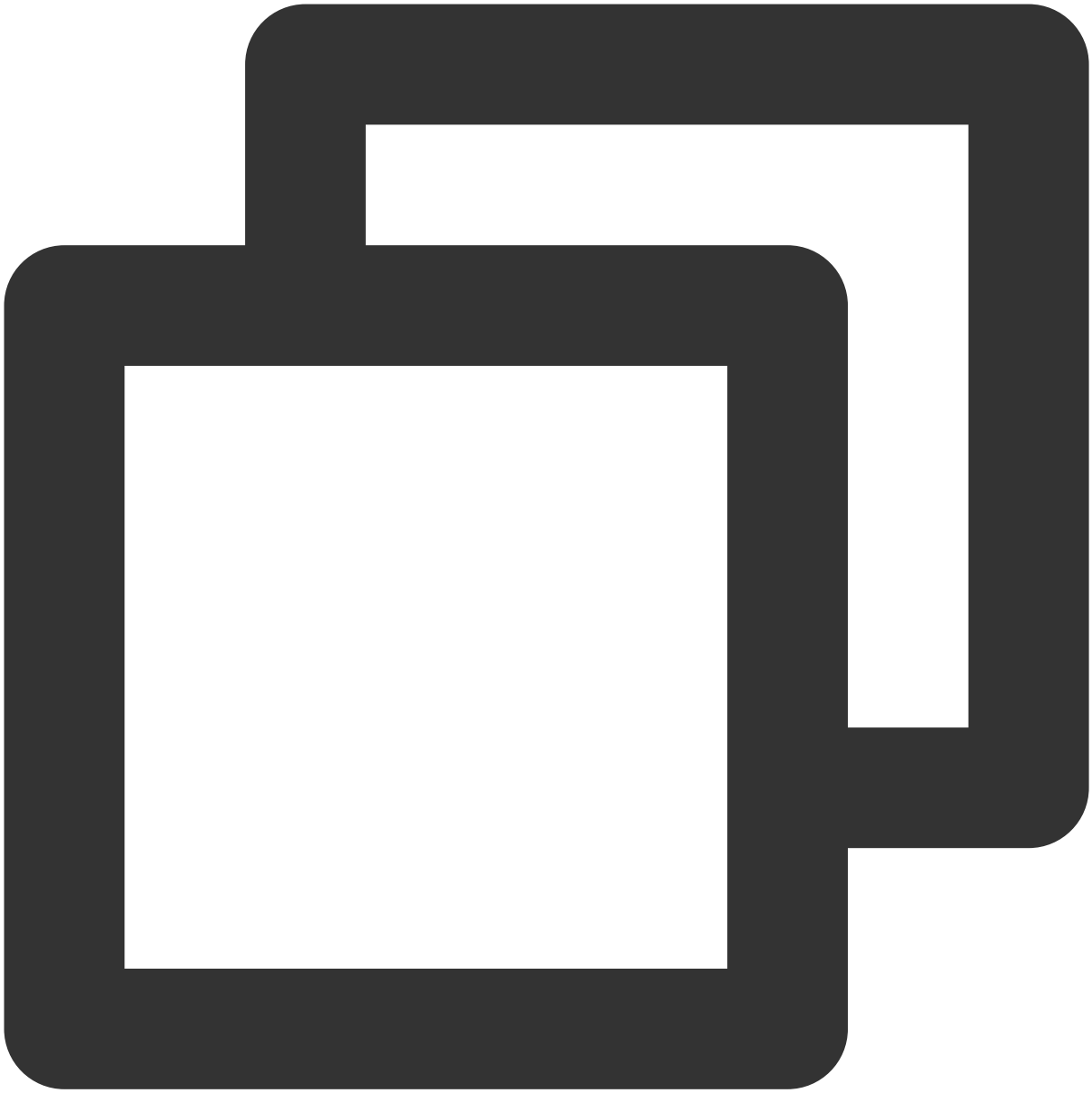
经过上述步骤，完成了自定义域名的配置。下面说明如何使用自定义域名访问 COS。

查看对象访问链接

1. 登录 对象存储控制台 ，找到已配置自定义域名的存储桶，并单击进入**文件列表**。选择一个对象，进入对象详情。操作指引请参见 查看对象信息。
2. 切换指定域名为**自定义源站域名**，下方的对象地址、临时链接会相应切换为自定义域名的链接，访问公有读对象可以使用对象地址（无签名），访问私有读对象可以使用临时链接（有签名）。

API 访问切换自定义域名

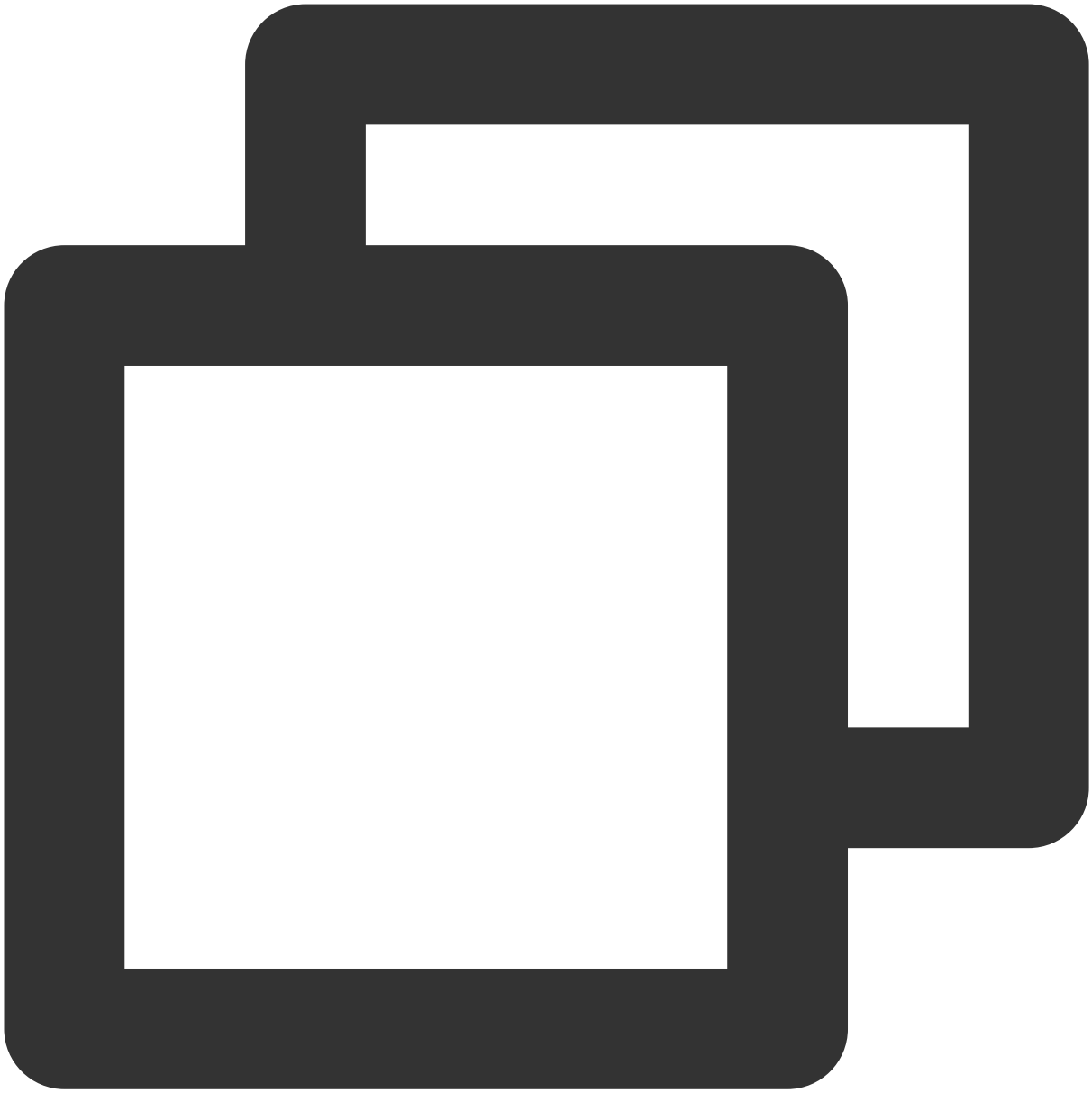
对于直接使用 API 访问 COS 的用户，只需要在访问时将请求 Host 修改为自定义域名即可。



```
GET /<ObjectKey> HTTP/1.1
Host: <BucketName-APPID>.cos.<Region>.myqcloud.com          #  替换为用户的自定义域名
Date: GMT Date
Authorization: Auth String
```

SDK 访问切换为自定义域名

对于使用 SDK 的用户，只需要在初始化 Client 时将 domain 参数设置为自定义域名即可。以 Python SDK 为例，代码示例如下。



```
domain = 'user-define.example.com' # 用户自定义域名
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=t
client = CosS3Client(config)
```

各语言 COS SDK 切换自定义域名的代码示例可参考以下文档：

Go SDK

JAVA SDK

Python SDK

PHP SDK

Android SDK
iOS SDK
JS SDK
Node.js SDK
.NET SDK
C++ SDK
C 语言 SDK
小程序SDK

配置自定义域名支持 HTTPS 访问

最近更新时间：2024-01-06 10:54:03

简介

用户可通过自有域名（自定义域名，例如 `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

操作步骤

开启 CDN 加速

步骤1：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，详细操作指引请参见 开启自定义 CDN 加速域名 文档。

步骤2：配置 HTTPS 访问

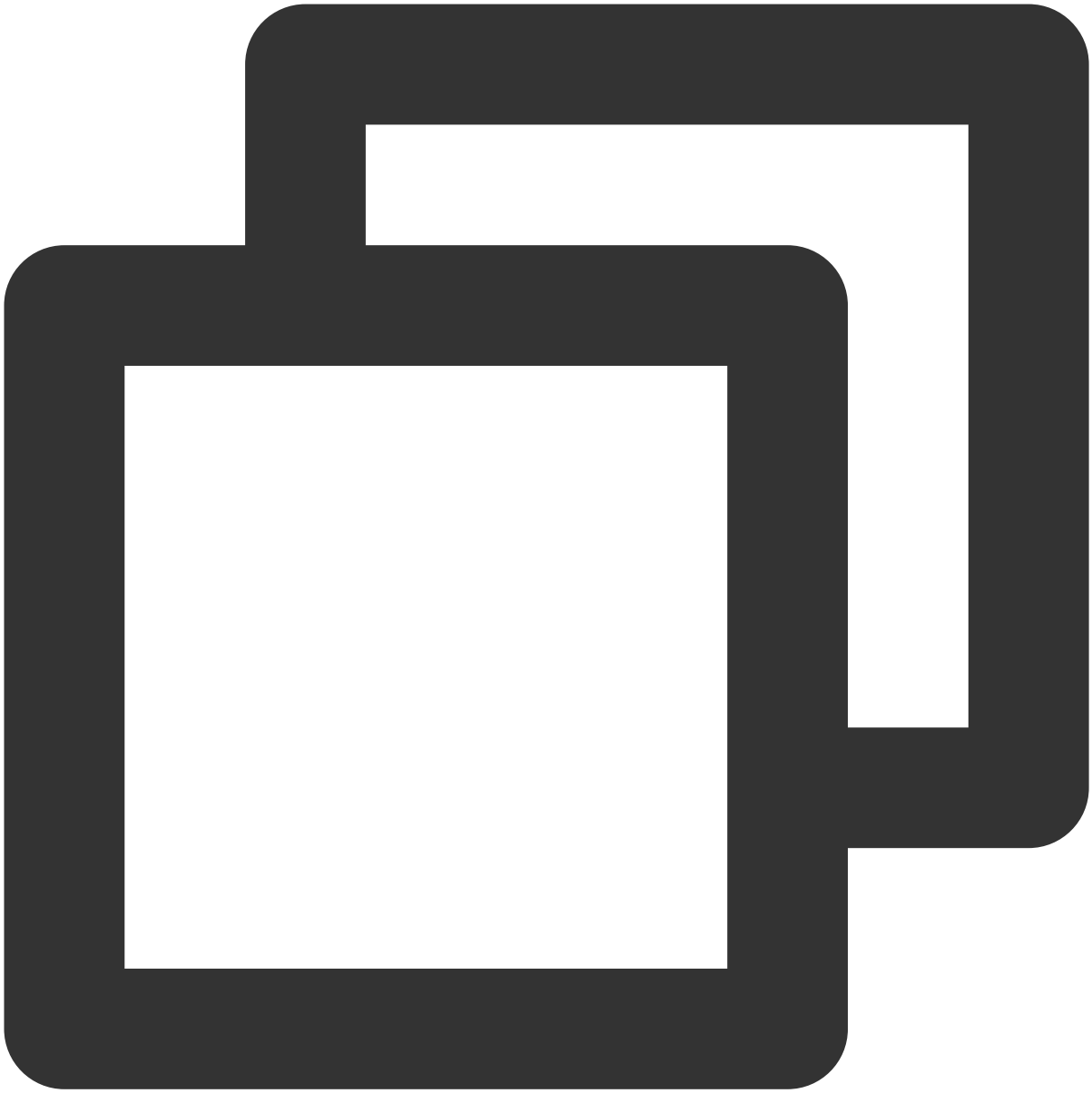
在 CDN 控制台 进行 HTTPS 配置，详细操作指引请参见 HTTPS 加速配置指南。

关闭 CDN 加速

本章节主要以示例的形式介绍在对象存储（Cloud Object Storage，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` 访问所属地域为广州、名称为 testhttps-1250000000 的存储桶，具体操作步骤如下：

步骤1：绑定自定义域名

COS 国内公有云地域、新加坡地域已支持托管自定义源站域名的 HTTPS 证书，您可以通过控制台为已添加的自定义源站域名绑定证书，详见 方式一。如果您的域名还没有 HTTPS 证书，可点击 申请腾讯云证书。
对于其他海外地域暂不支持 HTTPS 证书托管，若需要使用 HTTPS 证书，可参考 方式二。
方式一：通过 COS 控制台绑定自定义源站域名
将存储桶 testhttps-1250000000 绑定到域名 `https://test.cos.com` ，关闭 CDN 加速，详细操作指引请参见 开启自定义源站域名 文档。
方式二：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）：

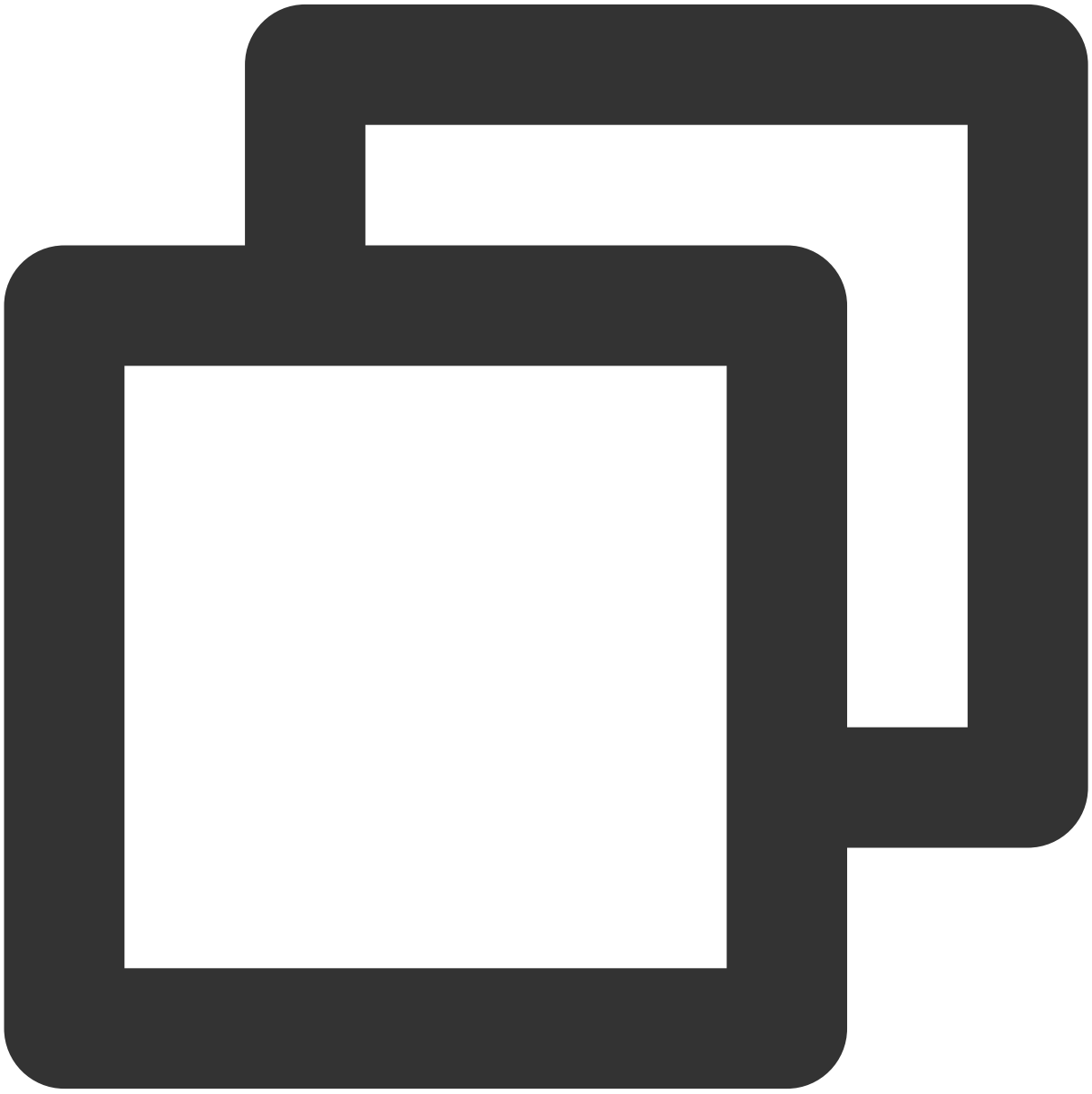


```
server {
  listen        443;
  server_name  test.cos.com ;

  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-1250000000.cos.ap-guangzhou.myqcloud.com; //配置存
    }
}
```

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，请前往 腾讯云 SSL 证书 页面进行申请。

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;
```

步骤2：解析域名到服务器

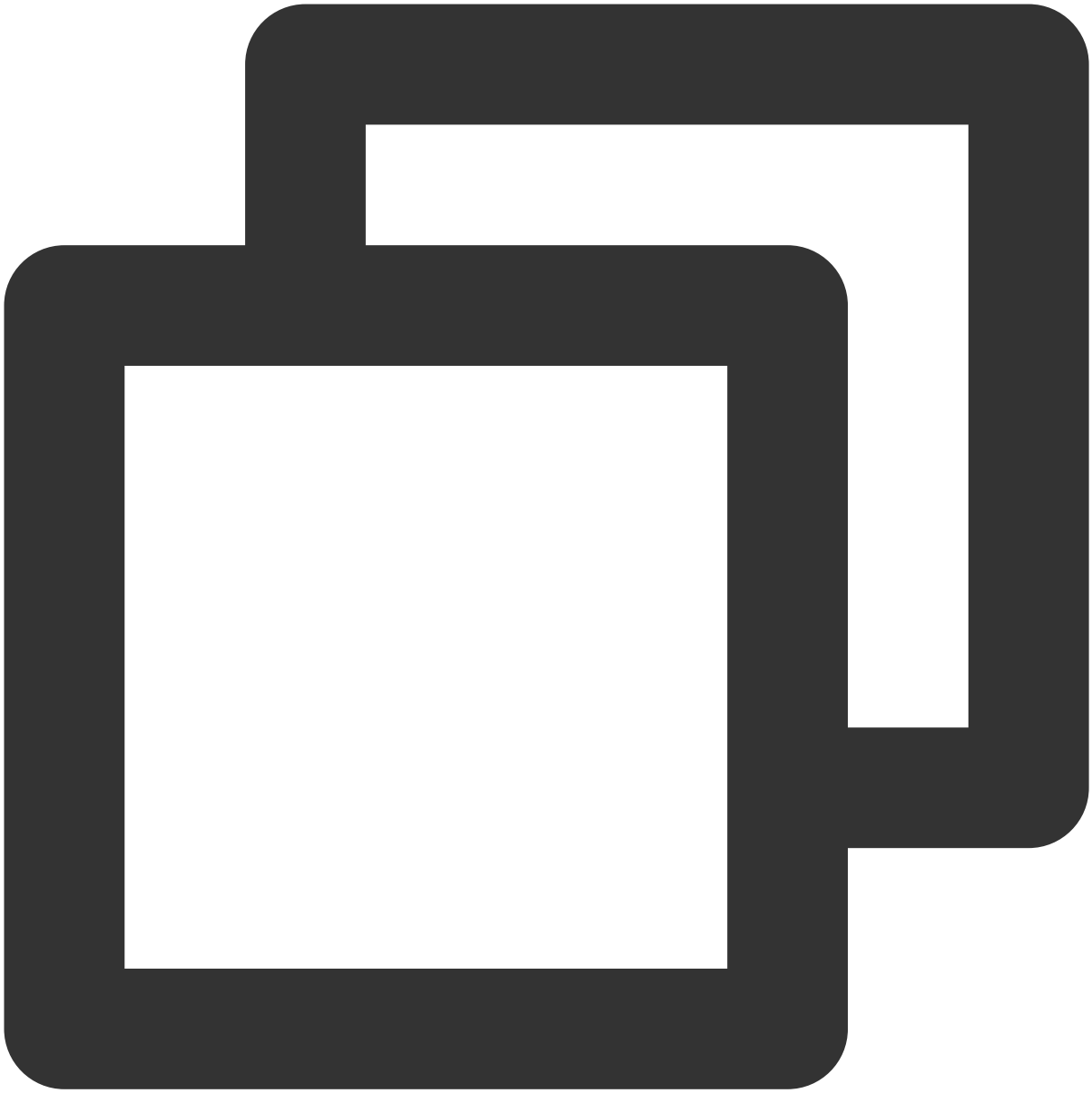
在您域名的 DNS 解析服务商处解析您的域名。

步骤3：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，需要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。详细操作指引请参见 设置静态网站。

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。



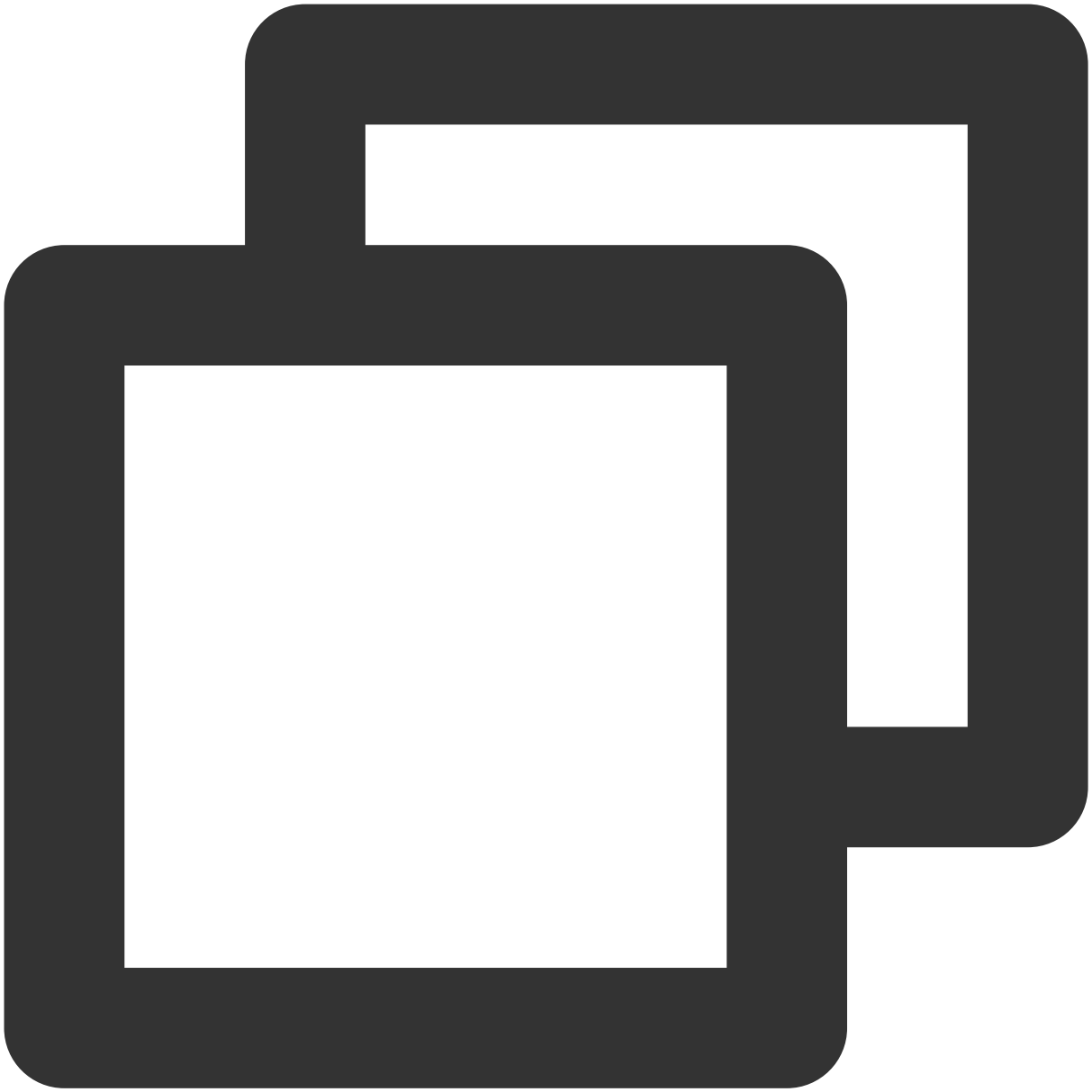
```
proxy_set_header Host $http_host;
```

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：

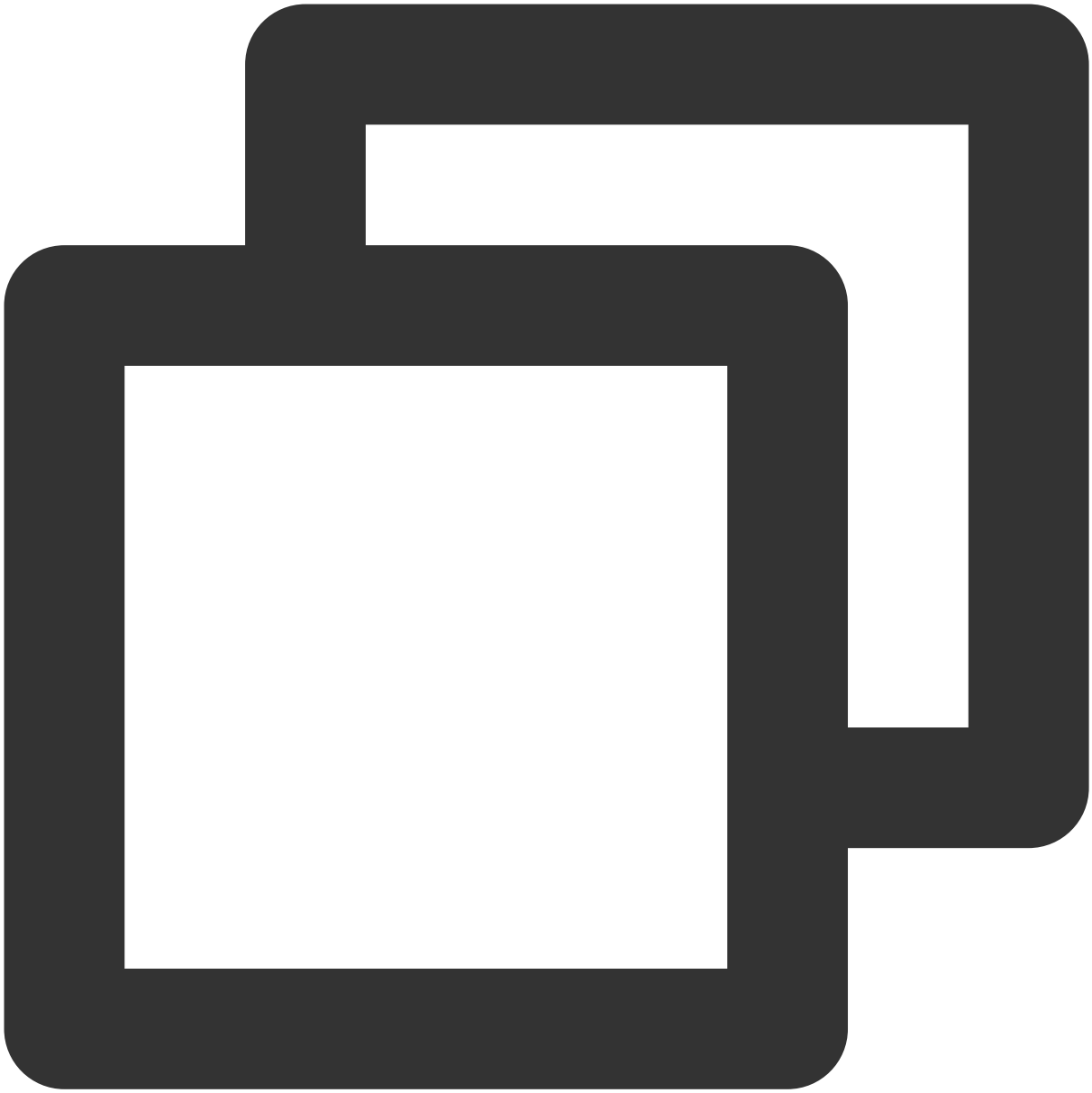
1.1 登录 对象存储控制台， 开启防盗链设置功能，选择白名单。详细操作指引请参见 设置防盗链。

1.2 在 Nginx 配置文件中，增加一行信息并重启 Nginx，刷新浏览器缓存。



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

1.3 设置完成后，直接打开文件将提示报错 `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"
}
```

设置跨域访问

最近更新时间：2024-01-06 10:54:03

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，但用户无感知。

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，即可跨源通信。

CORS 主要使用场景

用户在使用浏览器的情况下会使用到 CORS，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 的主要应用是实现在浏览器端使用 AJAX 直接访问 COS 的数据或上传、下载数据，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

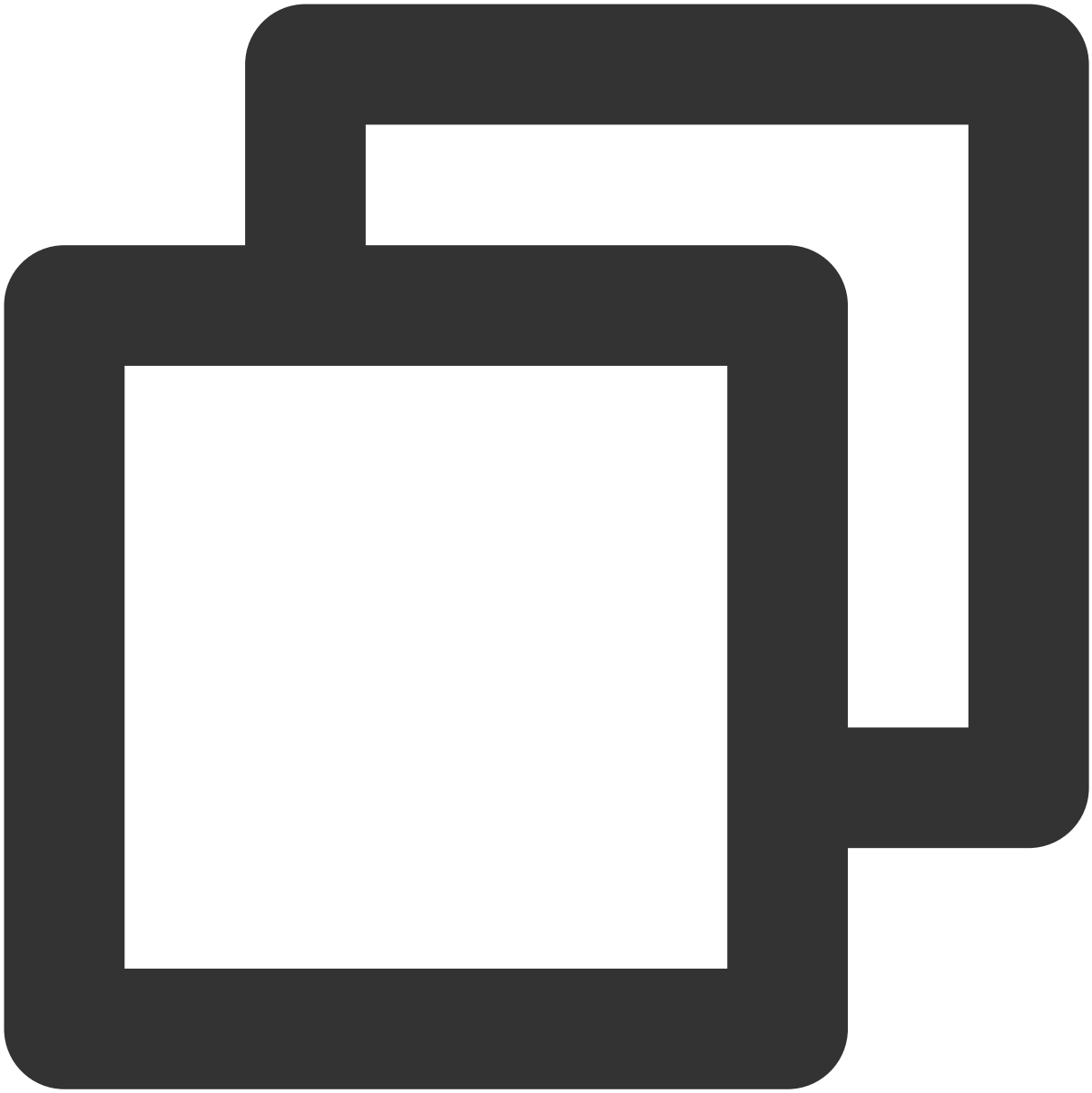
COS 支持对 CORS 规则的配置，从而根据需求允许或者拒绝相应的跨域请求。该 CORS 规则配置属于存储桶级别。
CORS 请求的通过与否和 COS 的身份验证等是完全独立的，即 COS 的 CORS 规则仅仅是用来决定是否附加 CORS 相关的 Header 的一个规则。是否拦截该请求完全由浏览器决定。
目前 COS 的所有 Object 相关接口都提供了 CORS 相关的验证，另外 Multipart 相关的接口目前也已经完全支持 CORS 验证。
说明
当同一个浏览器上的分别来源于 `www.a.com` 和 `www.b.com` 的两个页面同时请求同一跨域资源时，若 `www.a.com` 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。

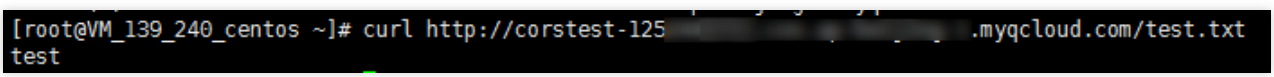
准备工作

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.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.myqcloud.com/test.txt
```

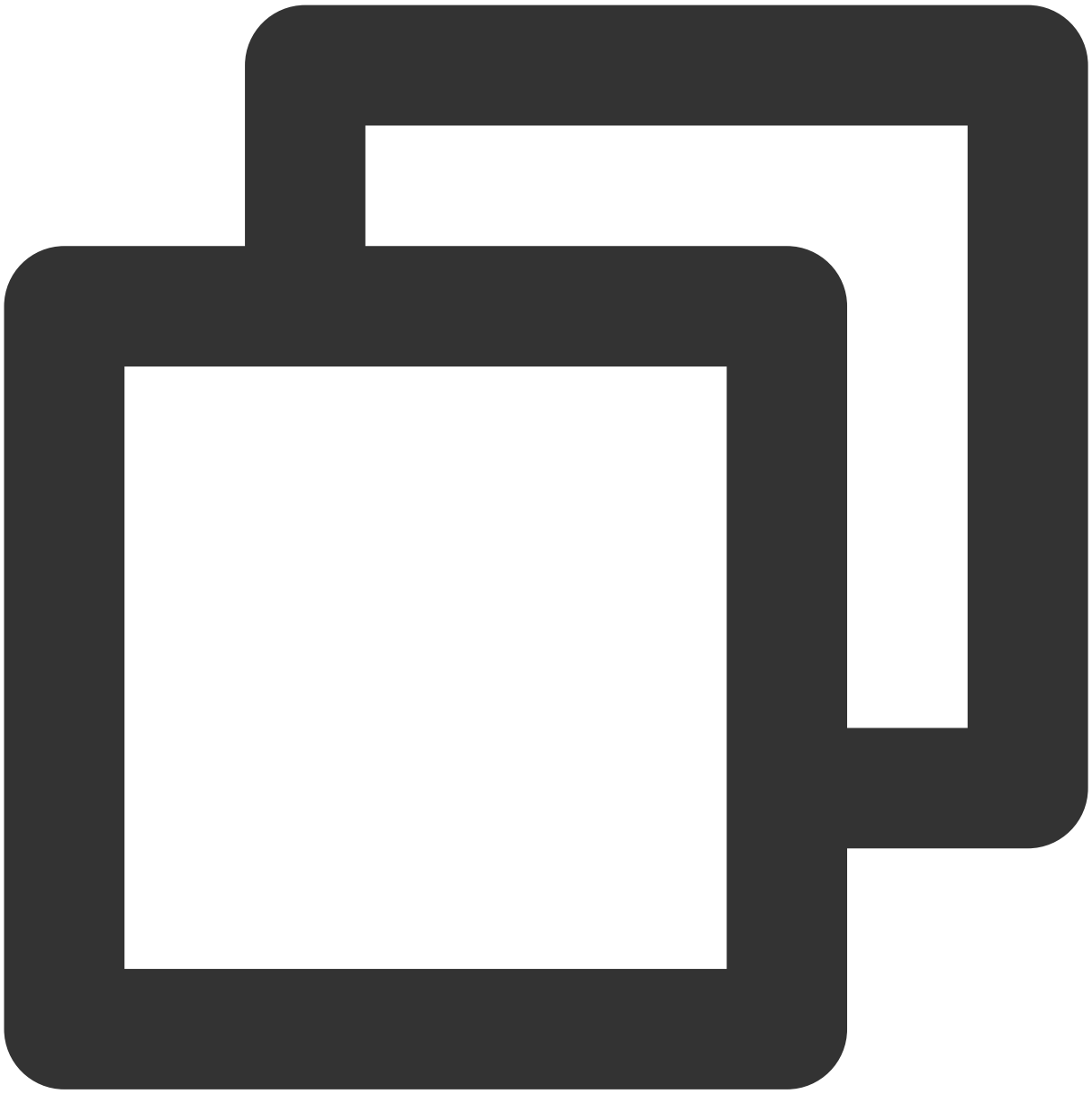
返回 test.txt 文件的内容：test。表示该文档可以正常访问。



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。

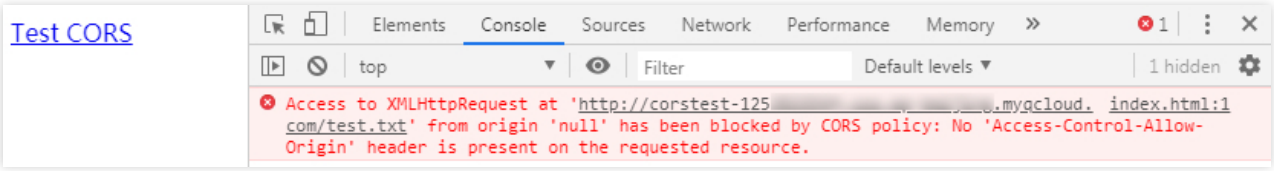
1. 写一个简单的 HTML 文件，将以下代码复制到本地保存成 HTML 文件后使用浏览器打开。因为没有设置自定义的头，因此该请求不需要预检。



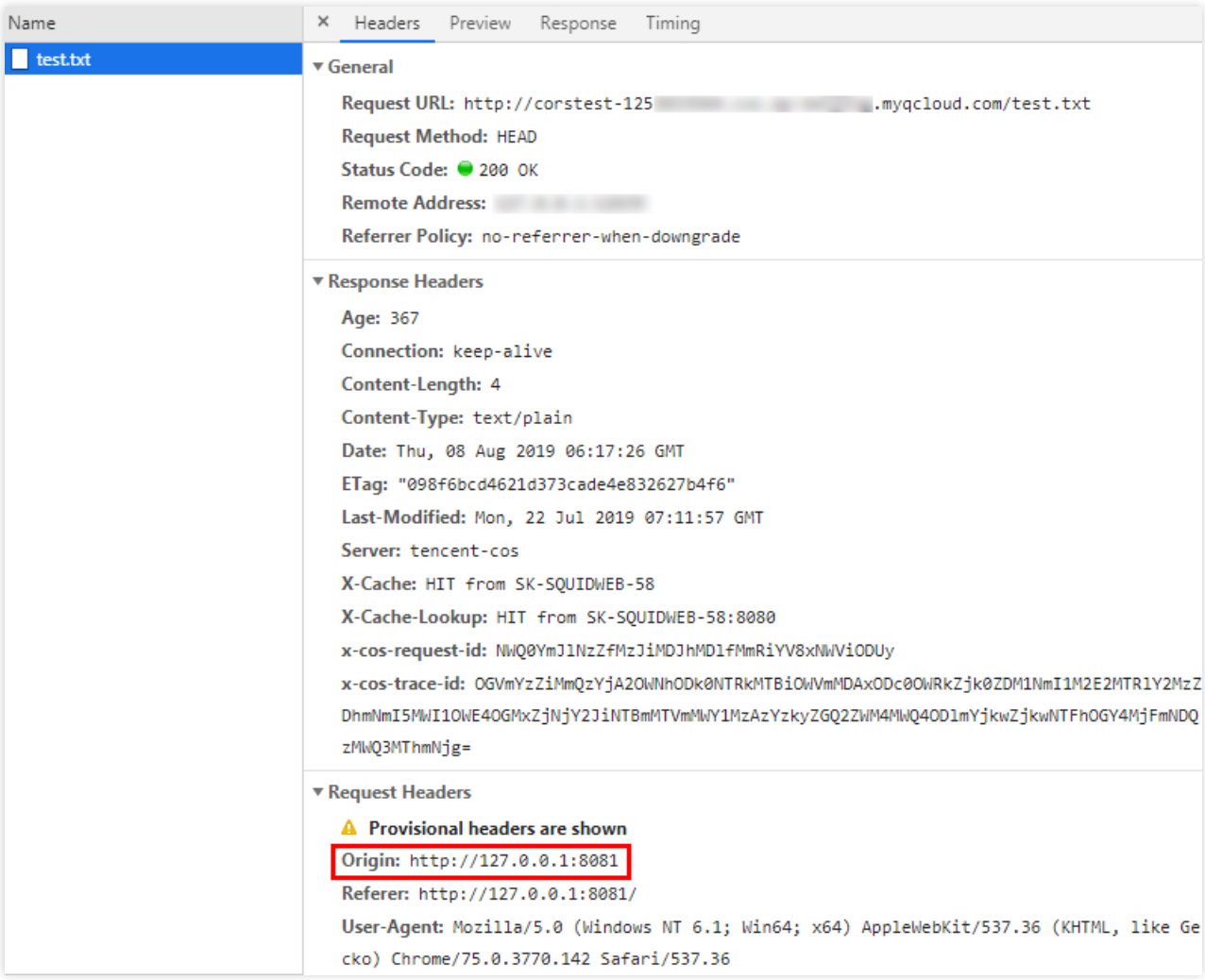
```
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
</head>
<body>
<a href="javascript:test()">Test CORS</a>
<script>
```

```
function test() {
    var url = 'http://corstest-125xxxxxxx.cos.ap-beijing.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>
```

2. 在浏览器中打开该 HTML 文件，单击 **Test CORS** 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。



3. 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。



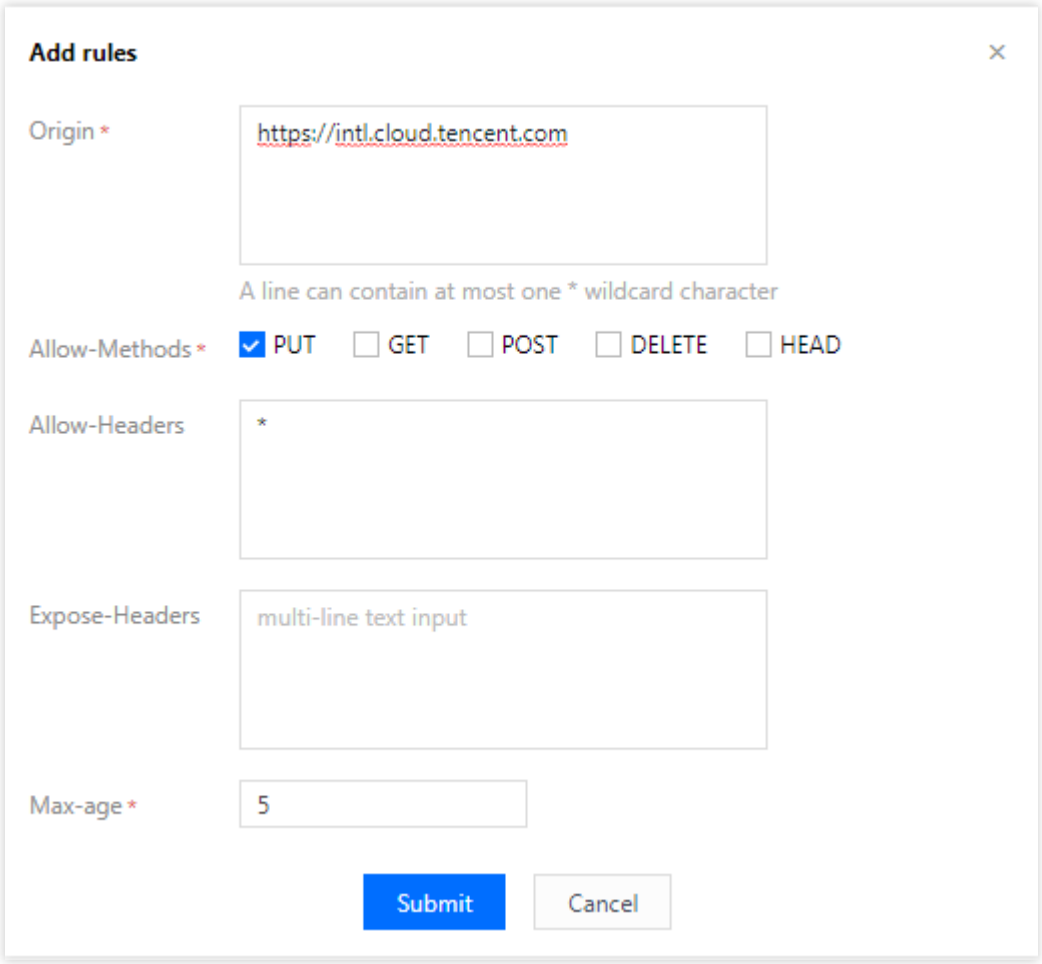
说明

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。

1. 登录 COS 控制台 ，单击**存储桶列表**，进入相关的存储桶，单击**安全管理**页签，下拉页面即可找到“跨域访问 CORS 设置”。
2. 单击**添加规则**，添加第一条规则，使用最宽松的配置如下：

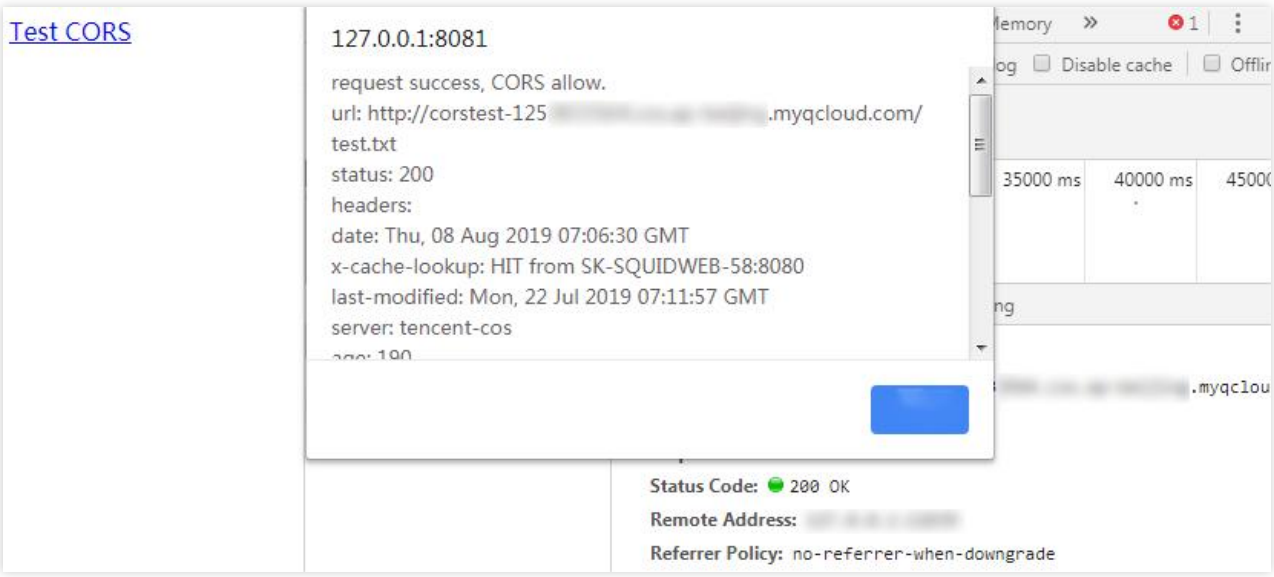


注意

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

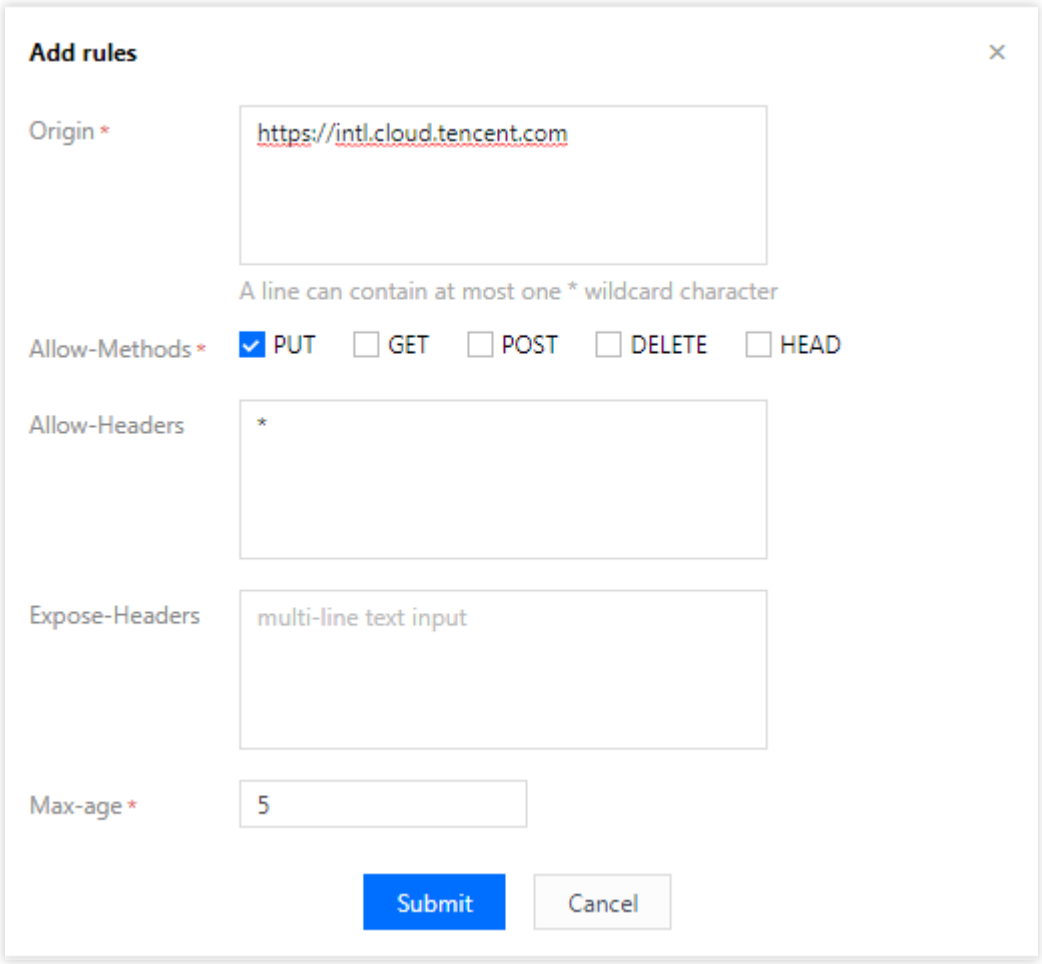
配置完成后，重新尝试访问 test.txt 文本文件。结果如下，可以正常访问请求。



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。例如，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，例如60秒，该项是可选配置项。

使用 COS 静态网站功能搭建前端单页应用

最近更新时间：2024-01-06 10:54:03

什么是单页应用？

单页应用（single-page application，SPA）是一种网络应用程序或网站的模型，它通过动态重写当前页面与用户进行交互，而非传统的从服务器重新加载整个新页面。这种方法避免了页面之间切换打断用户体验，使应用程序更像一个桌面应用程序。在单页应用中，所有必要的代码（HTML、JavaScript 和 CSS）都通过单个页面的加载而检索，或者根据需要（通常是为响应用户操作）动态装载适当的资源并添加到页面。
目前在前端开发领域，常见的单页应用开发框架有 React、Vue、Angular 等。
本文将使用目前较为热门的2个框架，一步步教您使用**腾讯云对象存储（Cloud Object Storage，COS）**提供的**静态网站**功能快速搭建一个线上可用的单页应用，并提供一些常见问题的解决方案。

准备工作

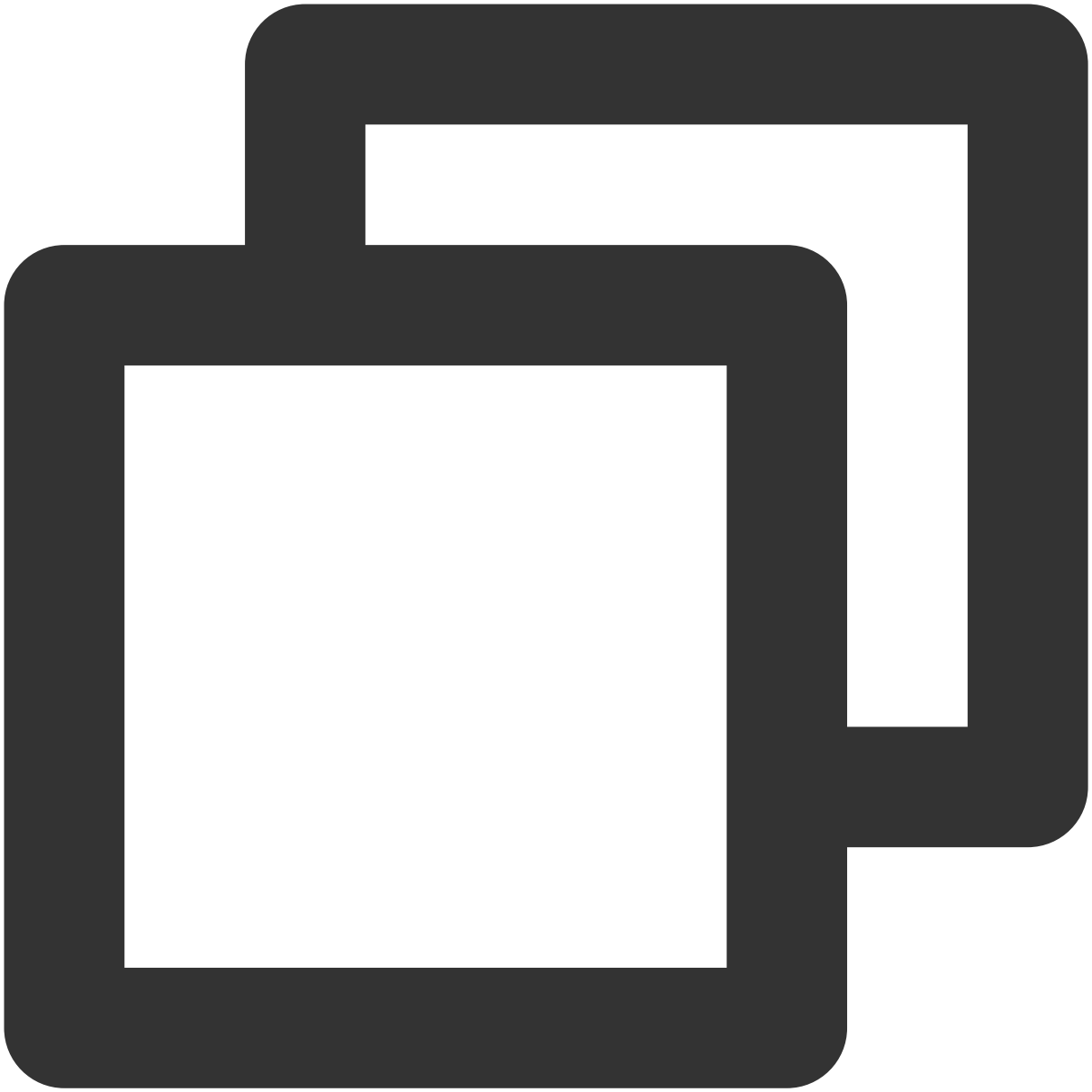
1. 安装 Node.js环境。
2. 注册腾讯云账户，并完成实名认证，确保能正常登录 腾讯云 COS 控制台。
3. 创建一个存储桶（为了方便测试可将权限设置为**公有读私有写**）。

编写前端代码

注意
如果已经自行实现了代码，可直接跳至 开启存储桶静态网站配置 步骤查看。

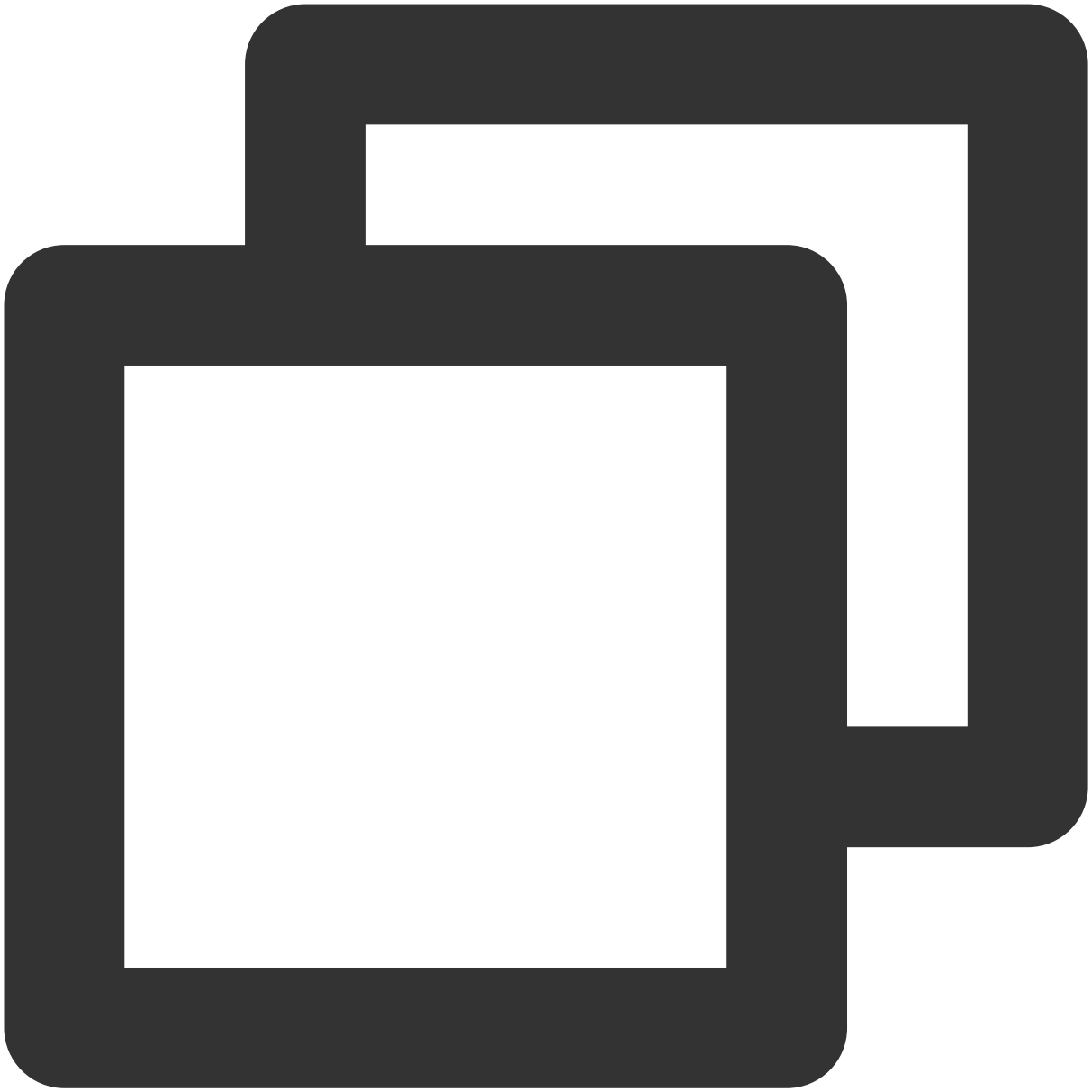
使用 Vue 快速搭建一个单页应用

1. 执行如下命令，安装 vue-cli：



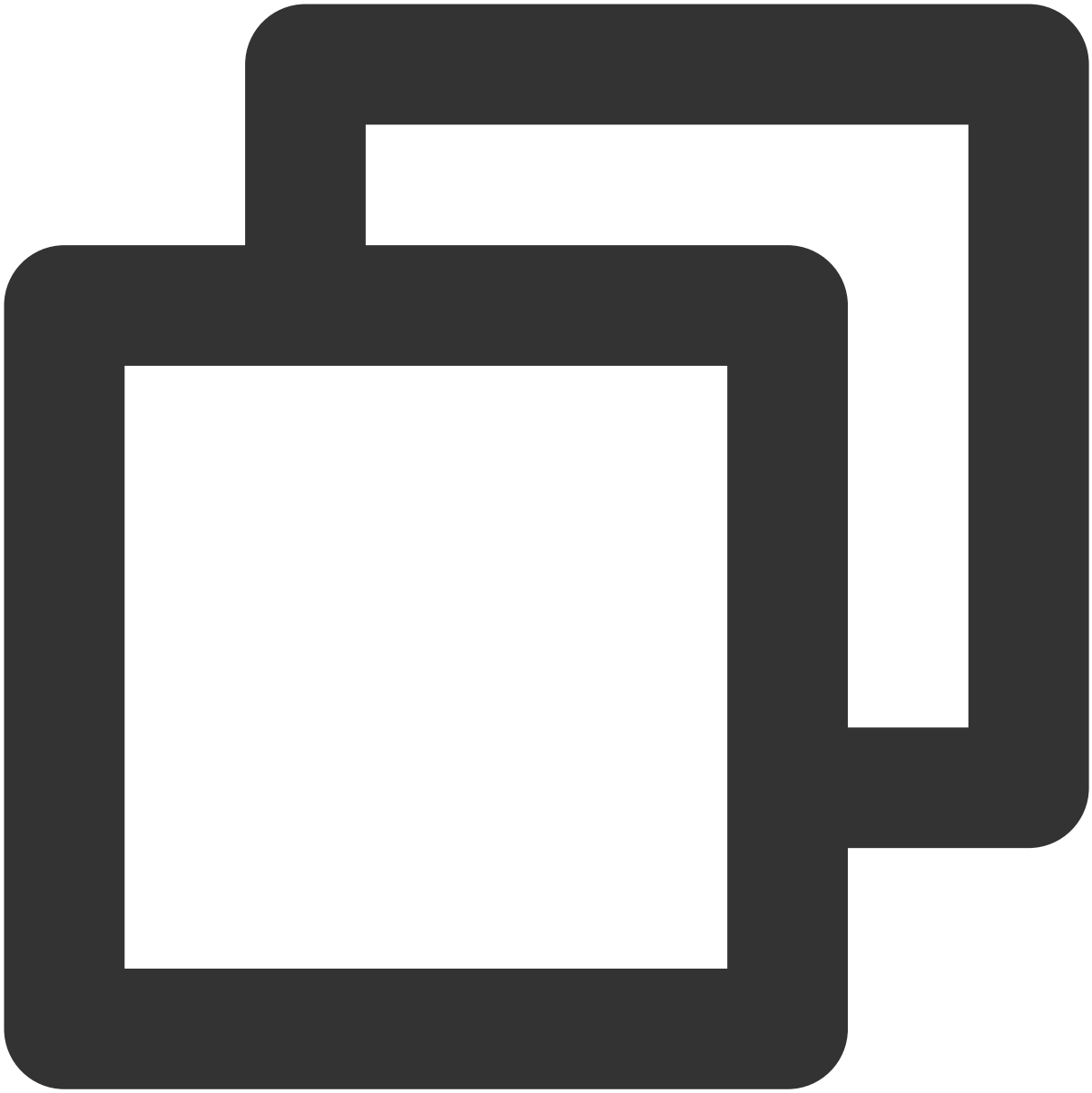
```
npm install -g @vue/cli
```

2. 执行如下命令，使用 vue-cli 快速生成一个 vue 项目，可参见 官方文档。



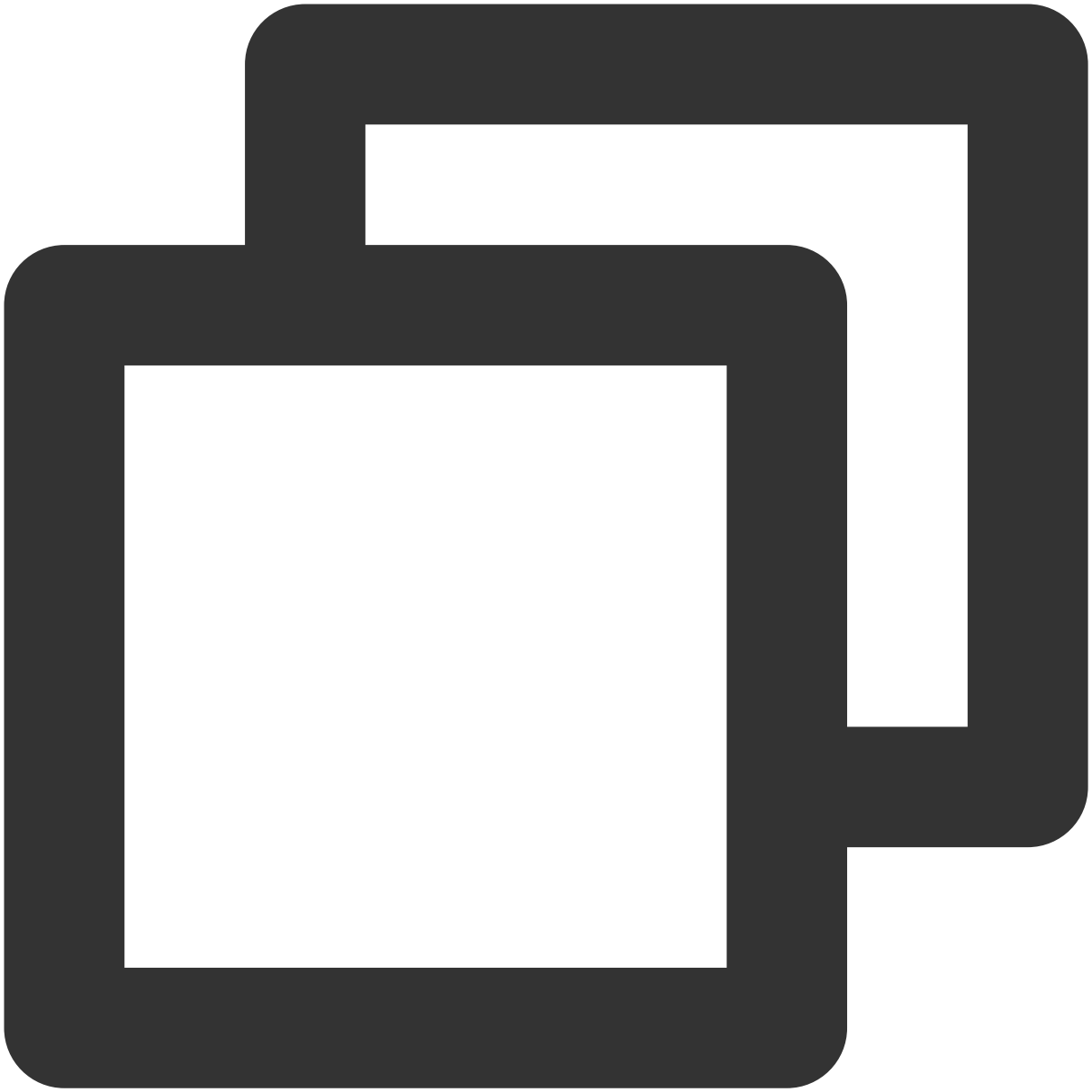
```
vue create vue-spa
```

3. 执行如下命令，在项目根目录下安装 vue-router：



```
npm install vue-router -S   (Vue2.x)
```

或者



```
npm install vue-router@4 -S   (Vue3.x)
```

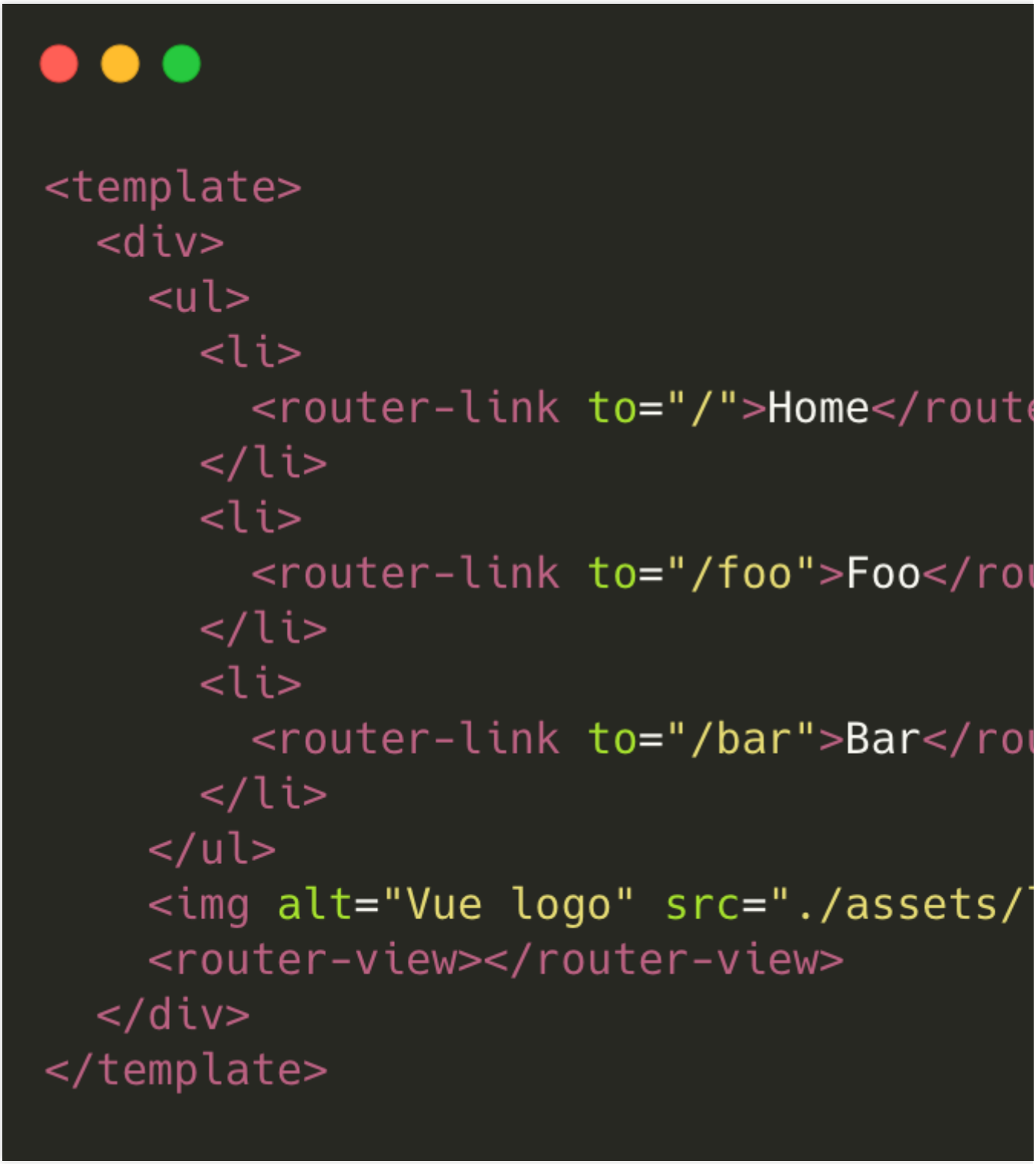
4. 修改项目里的 main.js 和 App.vue 文件。

main.js 如下图：

```
import { createApp } from 'vue'
import { createRouter, createWebHistory }
import App from './App.vue'
import Home from './components/Home.vue'
import Foo from './components/Foo.vue'
import Bar from './components/Bar.vue'
import Default from './components/404.vue'

const routes = [
    { path: '/', component: Home },
    { path: '/foo', component: Foo },
    { path: '/bar', component: Bar },
    { path: '/:w+', component: Default }
]

const router = createRouter({
    history: createWebHistory(),
    routes
})

const app = createApp(App)
app.use(router)
app.mount('#app')
```

</br>

App.vue 主要修改组件的 template，如下图：

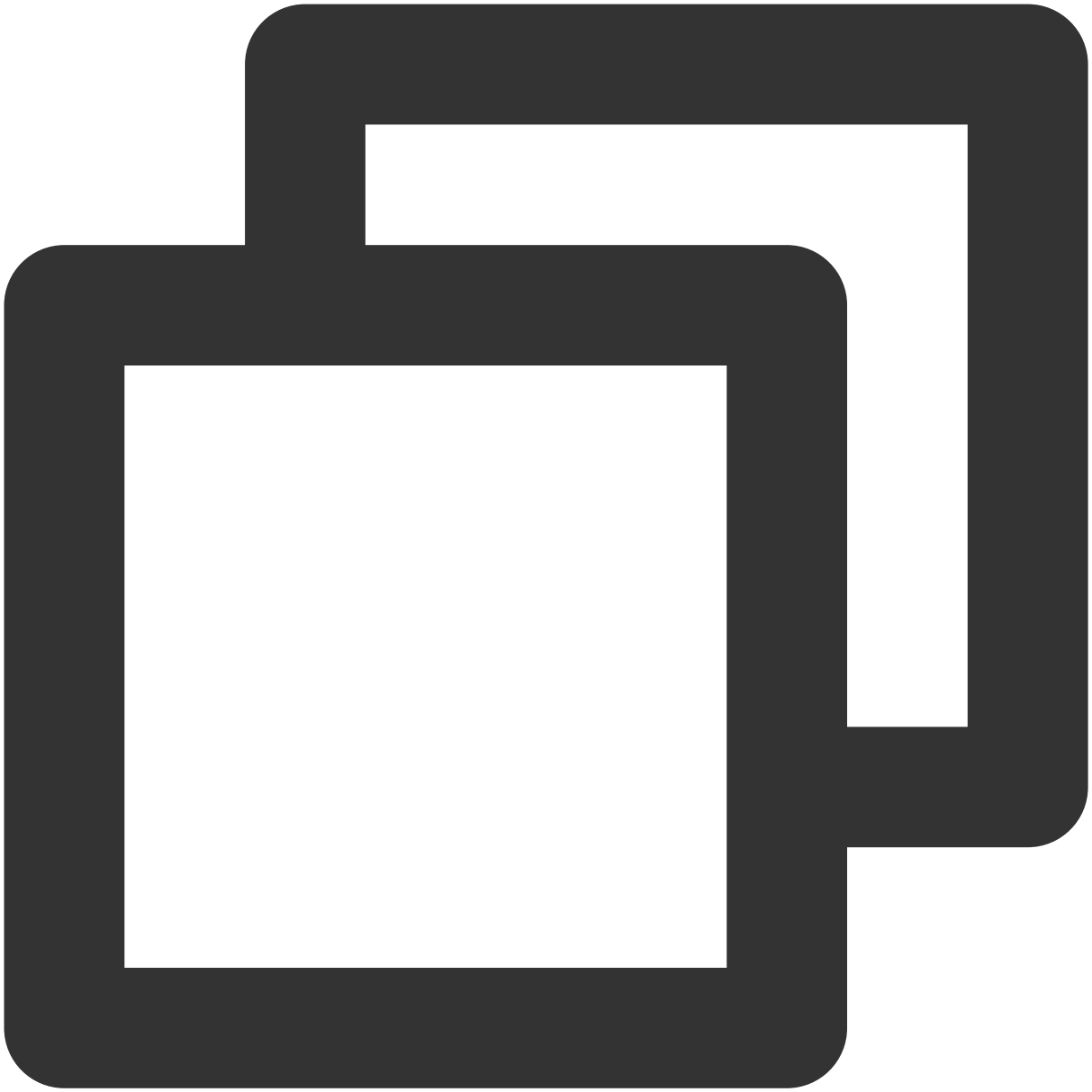


</br>

说明

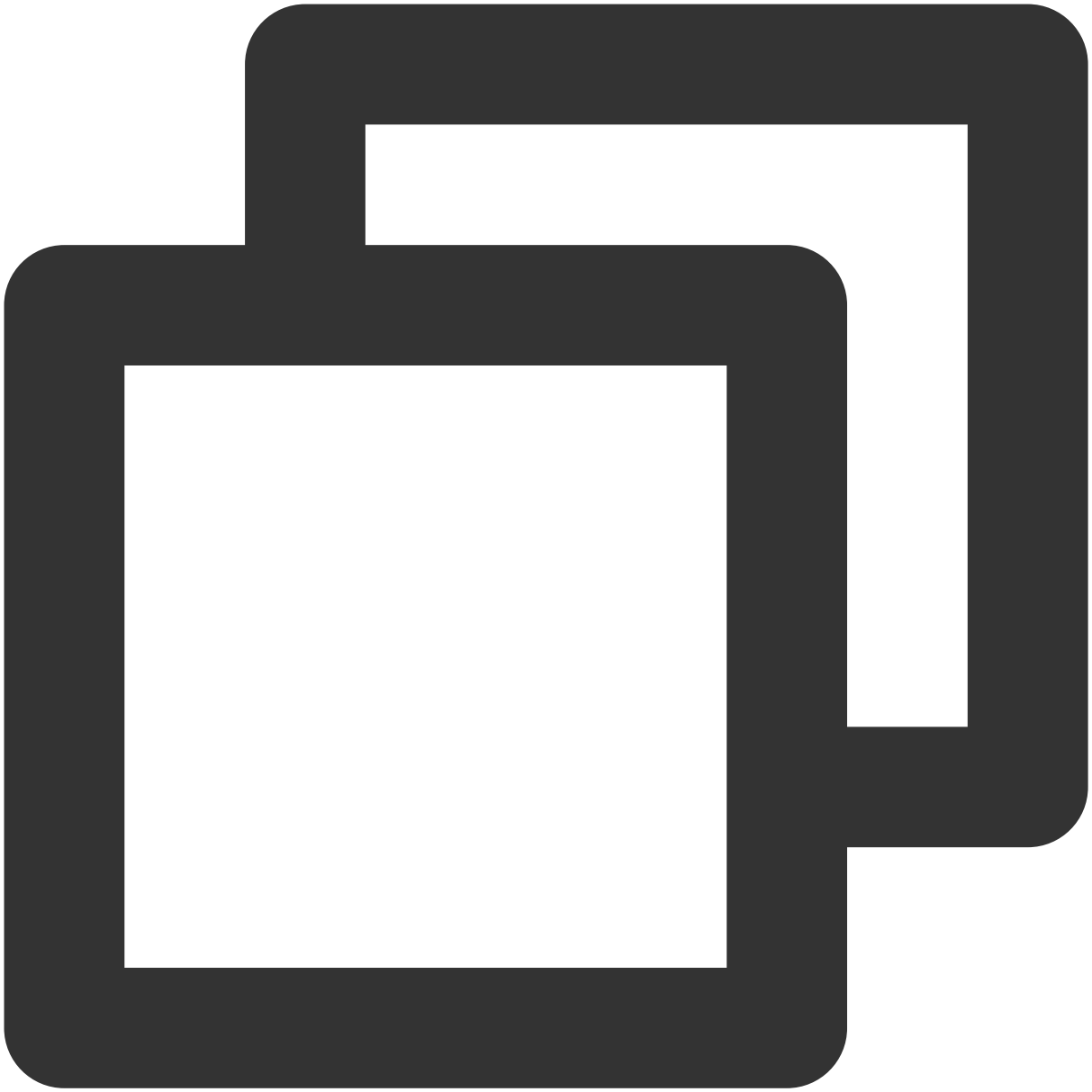
由于篇幅有限，这里仅节选部分关键代码，完整代码可 单击此处 下载。

5. 修改代码后，执行如下命令，进行本地预览。



```
npm run serve
```

6. 调试并预览检查无误后，执行如下命令，打包生产环境代码。

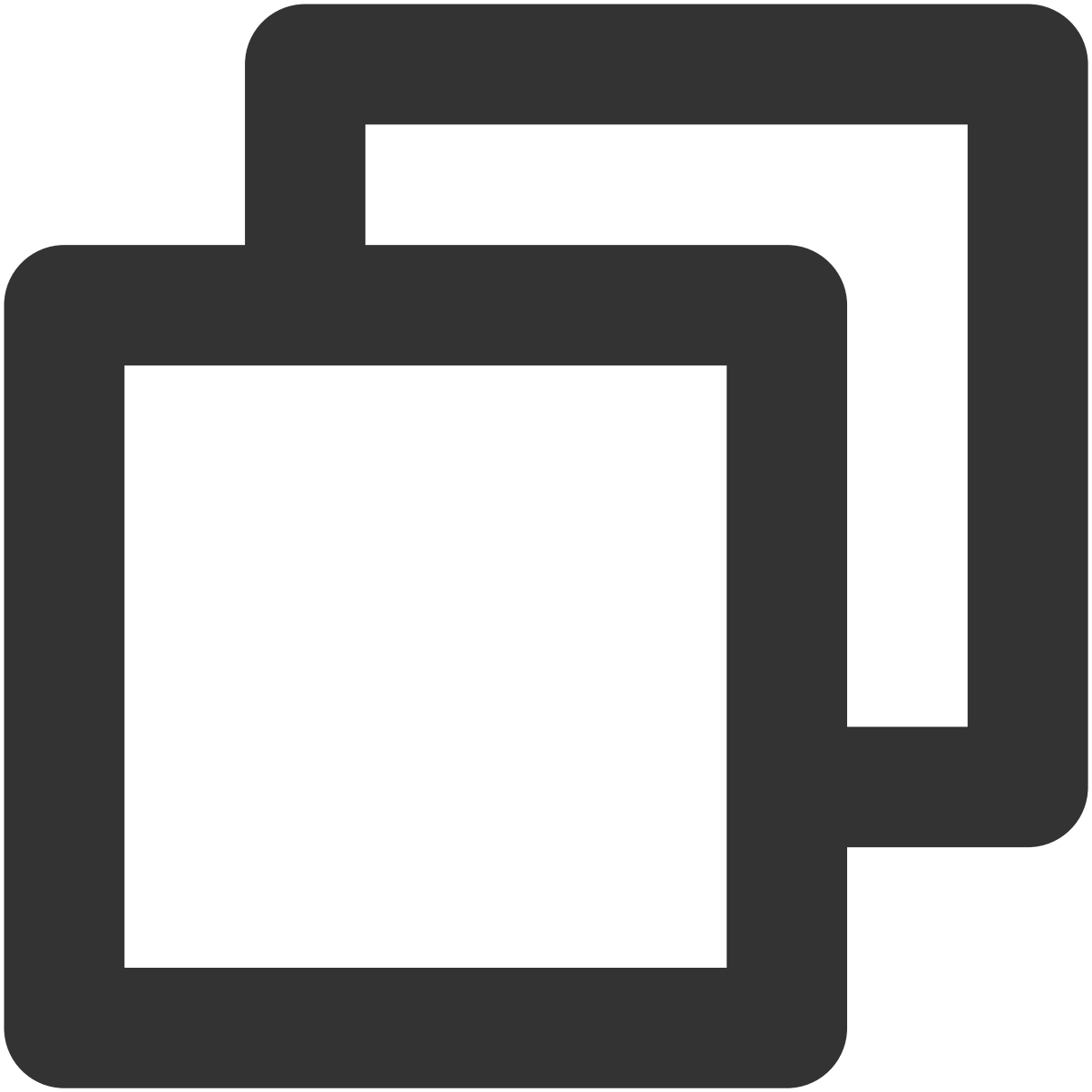


```
npm run build
```

此时，将会在项目根目录下生成 dist 目录。至此，Vue 的程序代码已经准备完毕。

使用 React 快速搭建一个单页应用

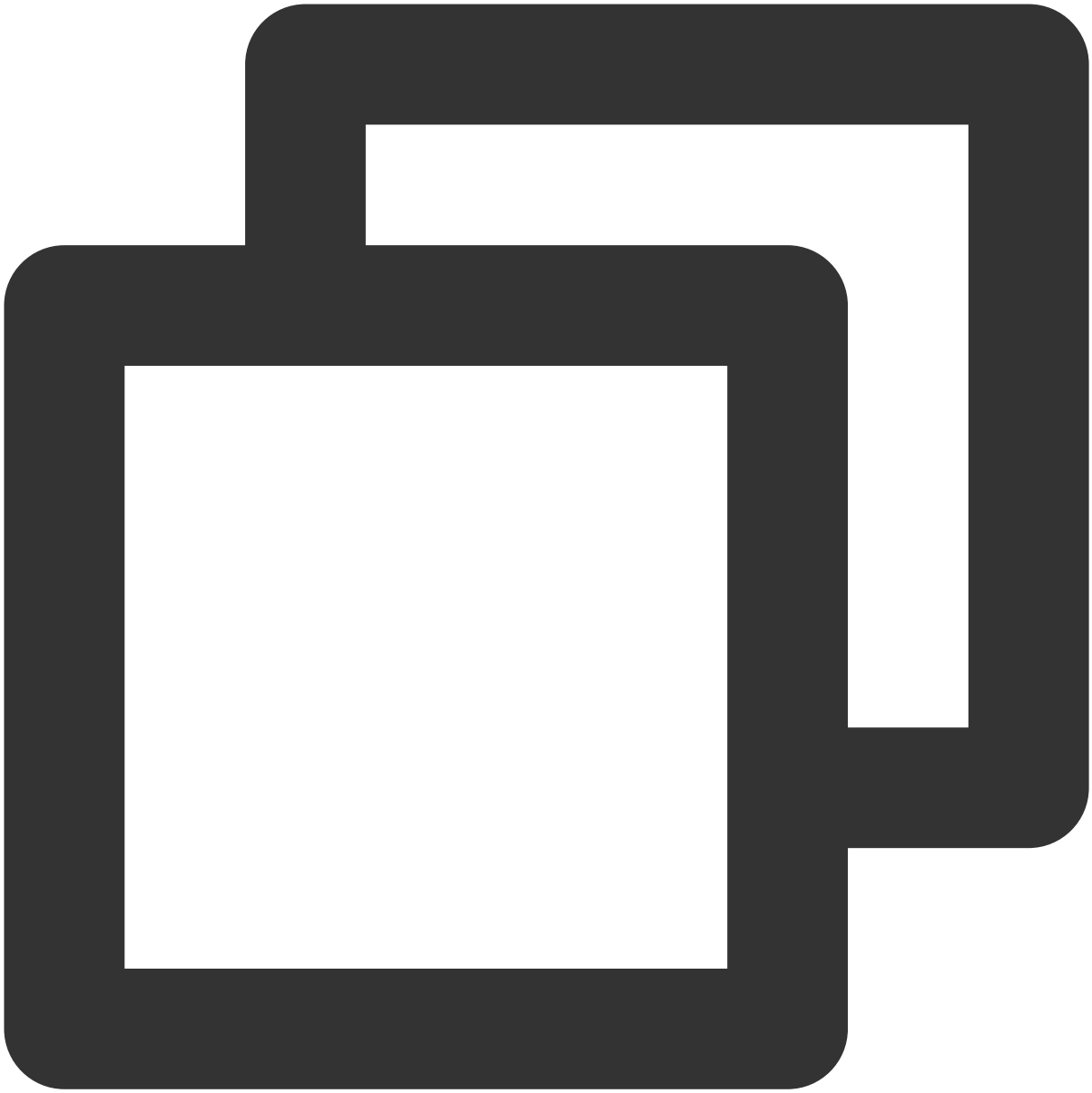
1. 执行如下命令，安装 create-react-app：



```
npm install -g create-react-app
```

2. 使用 create-react-app 快速生成一个 react 项目，可参考 官方文档。

3. 执行如下命令，在项目根目录下安装 react-router-dom：



```
npm install react-router-dom -S
```

4. 修改项目里的 App.js 文件。

```
import React from 'react';
import { BrowserRouter as Router, Switch, Route, Link }
```

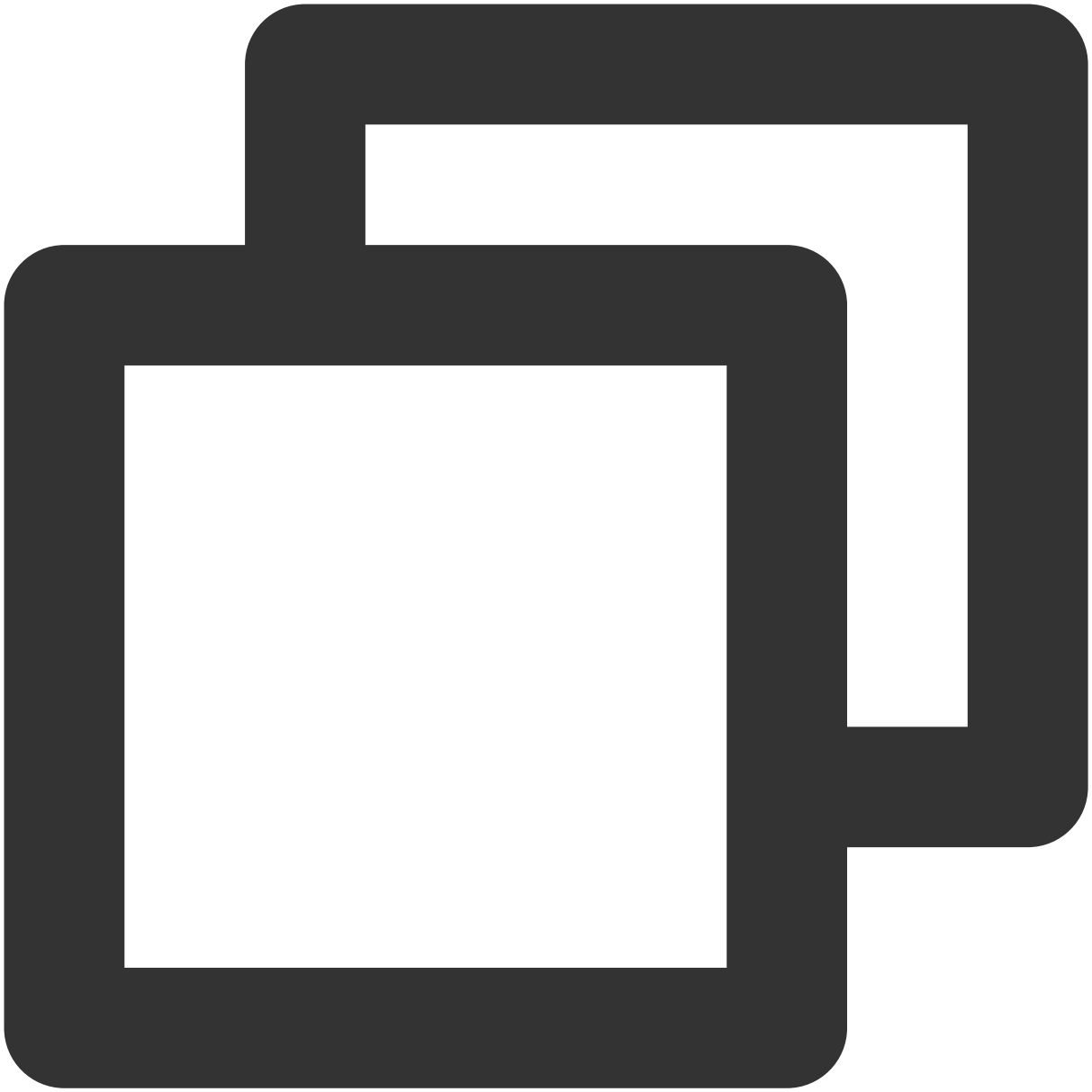
```
import './App.css';

function Home() {
  return <h2>Home</h2>;
}

function About() {
  return <h2>About</h2>;
}

function NoMatch() {
  return <h2>404 Page</h2>
}

function App() {
  return (
    <Router>
      <div className="App">
        <nav>
          <ul>
            <li>
              <Link to="/">Home</Link>
            </li>
            <li>
              <Link to="/about">About</Link>
            </li>
          </ul>
        </nav>
        <Switch>
          <Route exact path="/">
            <Home />
          </Route>
          <Route path="/about">
            <About />
          </Route>
          <Route path="*">
            <NoMatch />
          </Route>
        </Switch>
```

```
        </div>
      </Router>
    );
}

export default App;
```

说明

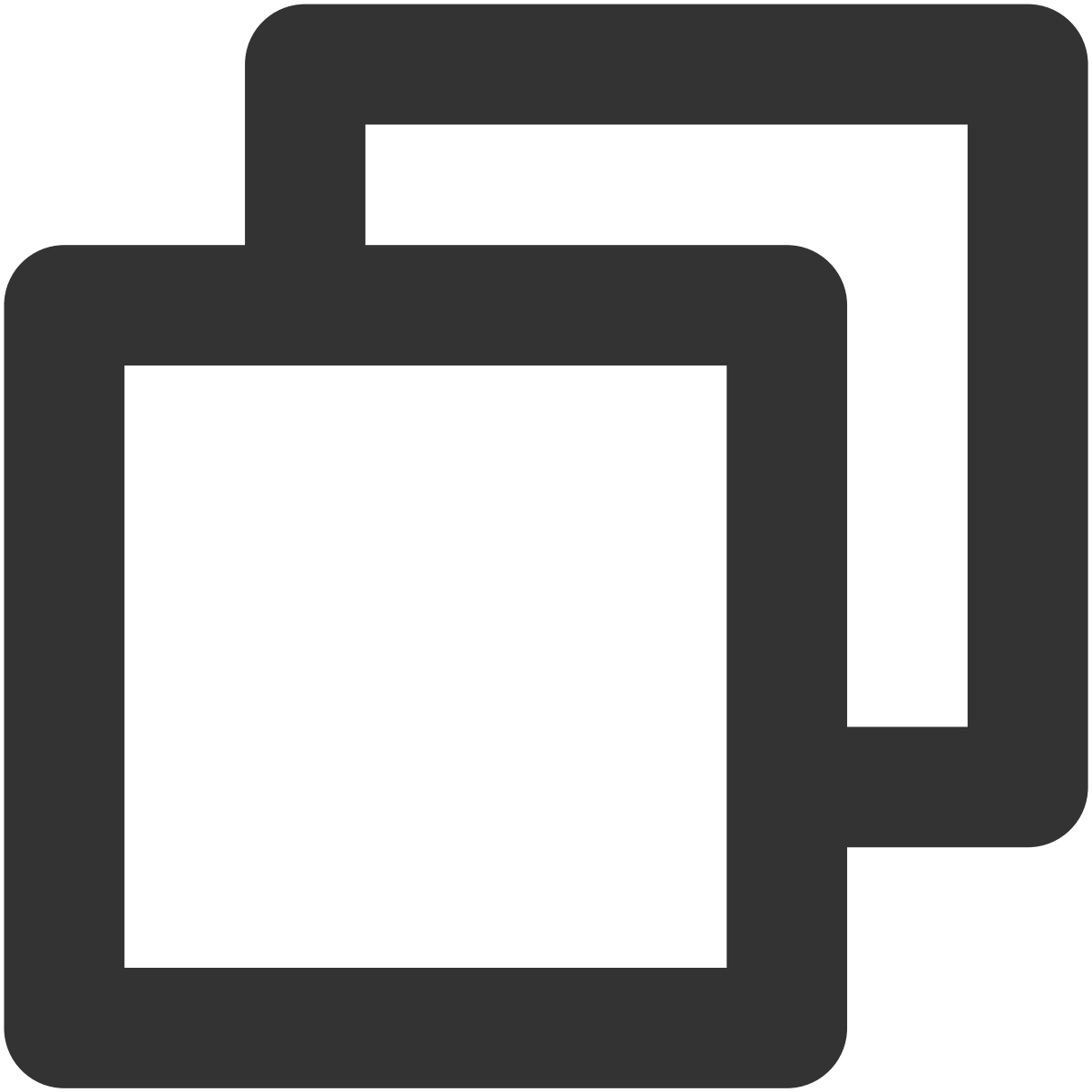
由于篇幅有限，这里仅节选部分关键代码，完整代码可 单击此处 下载。

5. 修改代码后，执行如下命令，进行本地预览。



```
npm run start
```

6. 调试并预览检查无误后，执行如下命令，打包生产环境代码。



```
npm run build
```

此时将会在项目根目录下生成 build 目录。至此，React 的程序代码已经准备完毕。

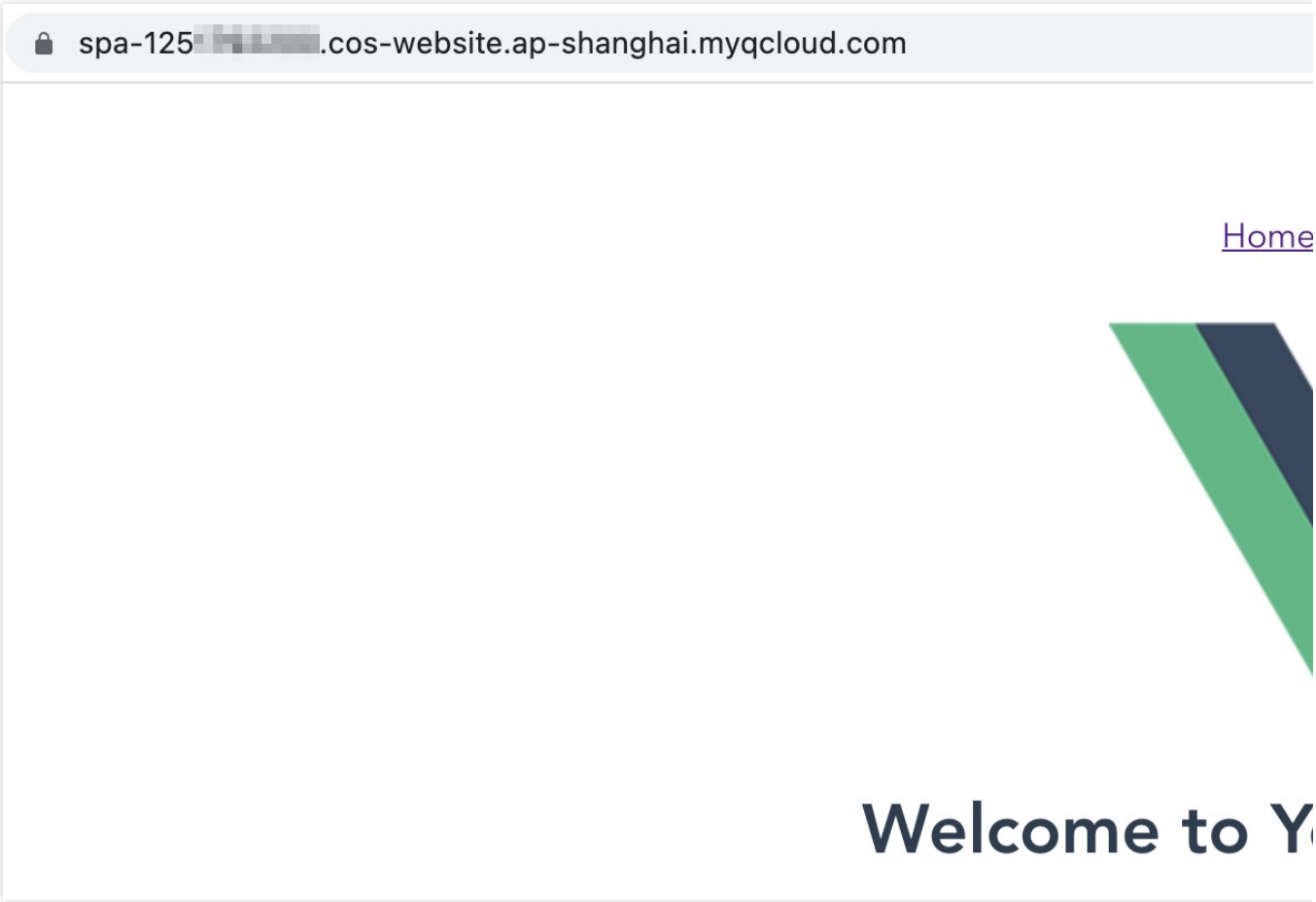
开启存储桶静态网站配置

1. 进入已创建存储桶的详情页面，并在左侧导航栏中，单击**基础配置 > 静态网站**。
2. 在静态网站管理页面，参考下图，进行配置。操作详情请参见 设置静态网站。

部署至 COS

1. 找到之前已经配置了静态网站的存储桶，进入文件列表页面。
2. 将编译目录（Vue 默认为 dist 目录，react 默认为 build 目录）下的所有文件上传至存储桶的根目录下。操作详情请参见 上传对象。
3. 访问存储桶的静态网站域名（如下图中的访问节点）。

即可看到已经部署完成的应用主页，这里以 Vue 应用举例。



4. 尝试切换路由（Home、Foo、Bar），并刷新页面，验证是否符合预期（即在路由下刷新不会出现404报错）。

常见问题

我不想使用静态网站的默认域名怎么办？可以使用我自己的域名吗？

除了上面使用的默认静态网站域名，COS 还支持设置自定义 CDN 加速域名、自定义源站域名。具体可参考 域名管理概述 文档进行配置，配置成功后即可使用自己想要的域名来访问应用。

请注意，如果您选择配置 CDN 加速域名，请关注 CDN 节点缓存过期配置，以便获取更新后的数据。

部署好应用之后，切换路由能成功渲染，但页面一刷新就出现404报错，是什么原因？

原因可能是配置静态网站的时候，缺失配置或错误配置了**错误文档**导致，请再次回顾本文开头提供的标准配置截图，可以看到错误文档和索引文档均配置为 `index.html` 。
由于单页应用的特性，我们需要保证在任何情况下都要成功访问到应用入口（一般为 `index.html` ），这样才能触发后续路由的一系列内部逻辑。

切换路由之后，页面能正常显示，但 HTTP 状态码依然为404，怎样才能正常返回200？

这里原因是配置静态网站的时候，缺少了配置**错误文档响应码**导致，可参考开头的配置截图，将其配置为200即可解决。

配置了错误文档之后，访问错误的路径还需要展示404的功能，应该如何处理？

这里推荐在前端代码里实现404逻辑，在路由配置最底部设置一个底层的匹配规则，当前面所有规则都匹配失败的时候则渲染一个404组件，组件内容可根据自行需求来设计实现。具体可参考本文提供的代码 demo 里的路由配置的最后一个配置。

访问页面出现 403 Access Denied 报错是什么原因？

原因可能是存储桶的权限设置了**私有读写**，可以修改为**公有读私有写**解决。
另外，如果使用 CDN 加速域名访问**私有读写**的存储桶，还需注意开启**回源鉴权**配置，这样才能授权 CDN 服务正常访问到 COS 下的资源。

图片处理实践
图文混合水印实战

最近更新时间：2024-01-06 10:47:50

操作场景

数据万象基础图片处理的水印功能支持在图片上添加 图片水印 或 文字水印。但在现实的业务场景中，经常出现一张图片上既有固定的 Logo 水印，也有动态变化的文字水印（例如用户名）的情况。针对上述场景，我们提供了如下方案供您参考，您可结合实际业务场景，选择合适的接入方案。

方案优势对比

方案	优点	缺点
方案一	接入简单，实现方便。	水印尺寸无法跟随图片大小进行动态变换，无法进行平铺操作。
方案二	在图片尺寸多变的场景下，会达到比方案一更好的自适应效果。	接入流程更为繁琐，且会收取两次处理费用。

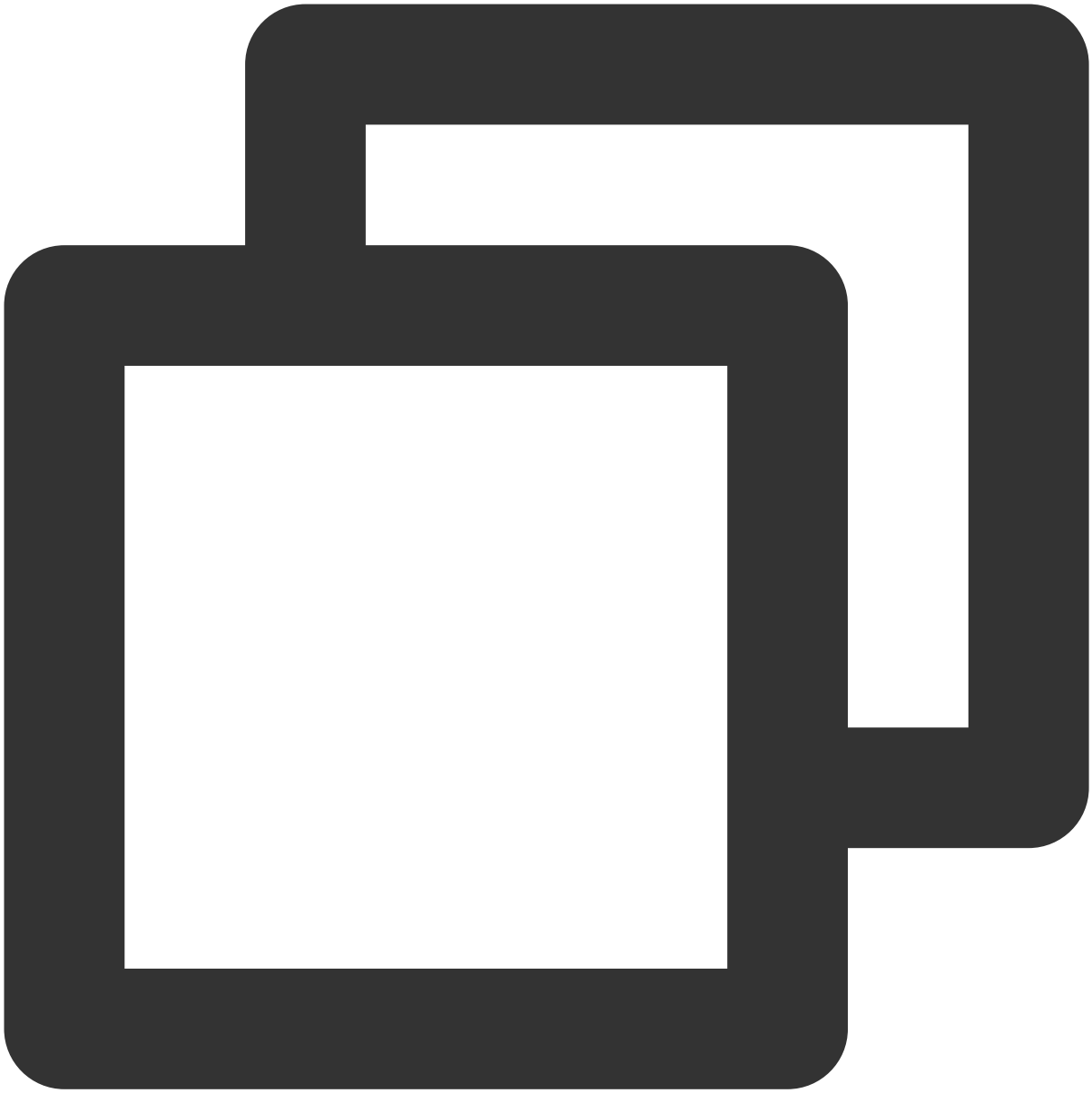
操作方案

方案一：使用管道操作符实现一条 URL 同时打上两种水印

数据万象的基础图片处理功能支持使用 管道操作符 "|"实现一次请求、多次处理，且只计算一次处理和流量费用。使用这种方式可大幅减少重复请求带来的时延和额外费用。

操作步骤

1. 参见 图片水印 API 文档，定义图片水印的参数。
如您不熟悉 API 参数，可参考 样式管理 文档，通过控制台新增样式，生成参数。
处理参数如下：



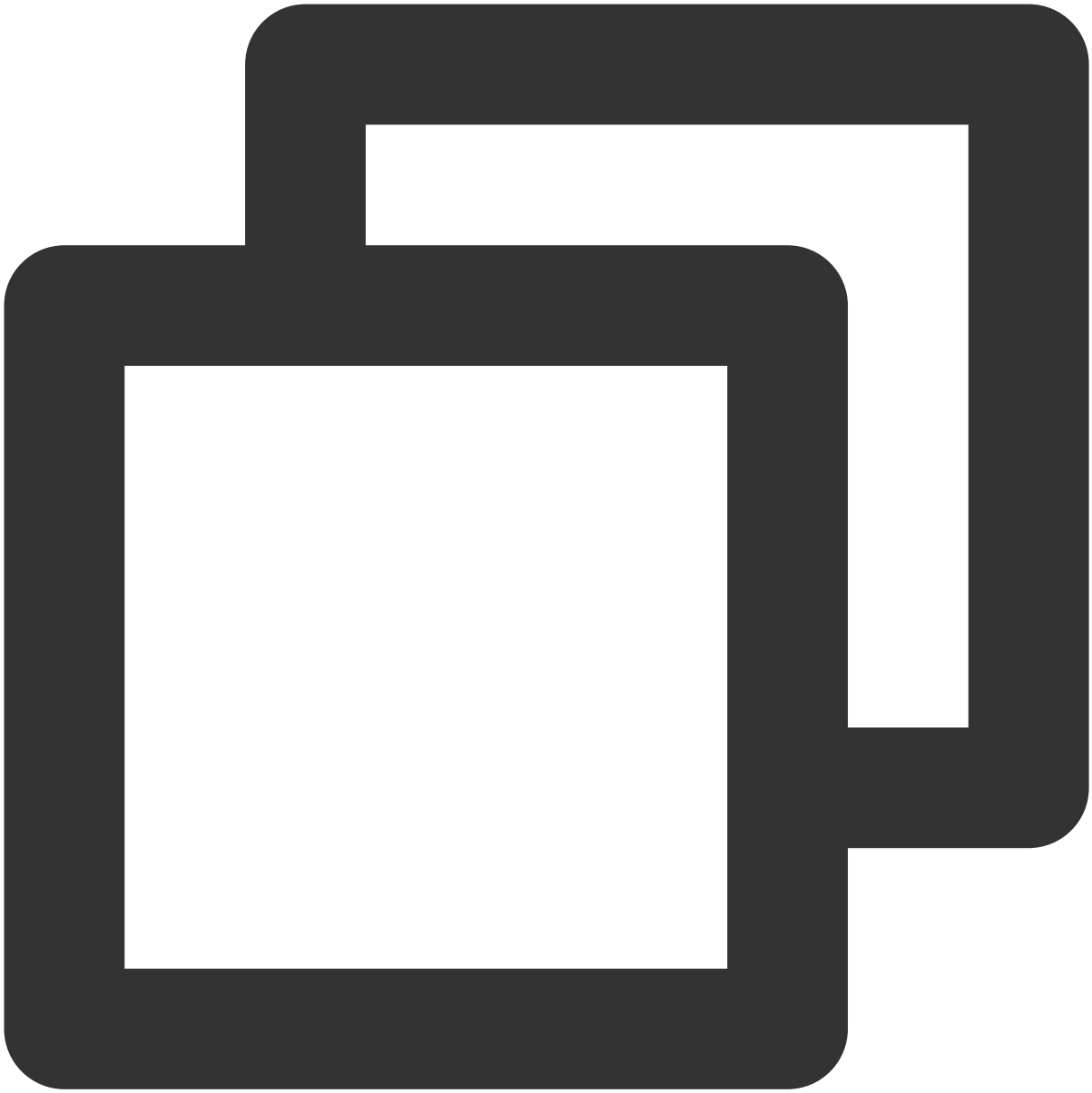
```
watermark/1/image/aHR0cDovL2V4YW1wbGVzLTEyNTgxMjU2MzguY29zLmFwLWd1YW5nemhvdS5teXFjb
```

说明

此处的编码 `aHR0cDovL2V4YW1wbGVzLTEyNTgxMjU2MzguY29zLmFwLWd1YW5nemhvdS5teXFjbG91ZC5jb20vbG9nby5wbmc` 为水印图片链接（即存储在对象存储 bucket 上的图片链接）经过 URL 安全的 base64 编码后生成。

2. 参见 文字水印 API 文档，定义文字水印的参数。

如您不熟悉 API 参数，可参考 样式管理 通过控制台新增样式，生成参数。

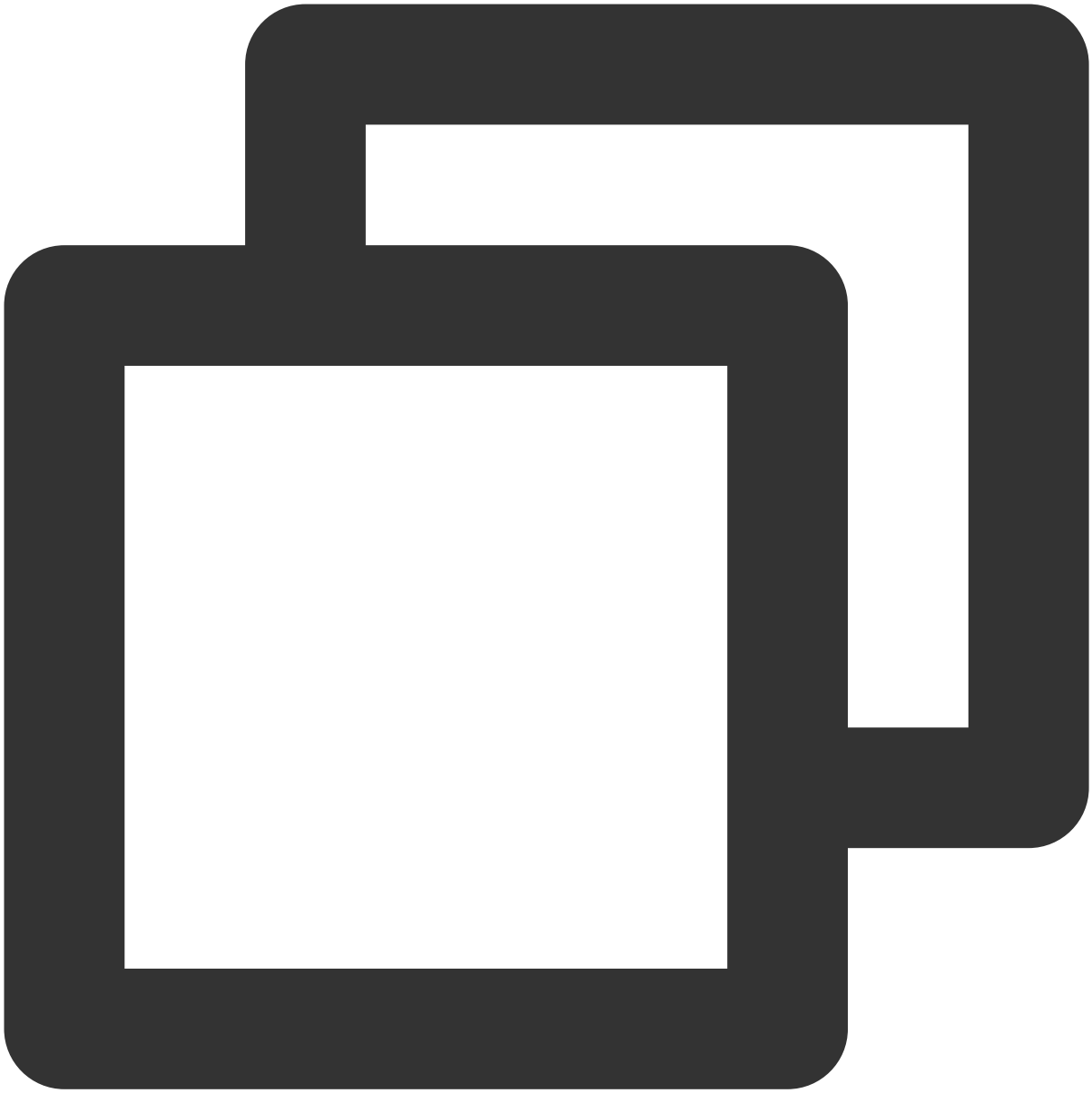


```
watermark/2/text/VUlOOiAxMjM0NTY3OA/font/SGVsdmV0aWNhLmRmb250/fontsize/36/fill/IzAw
```

说明

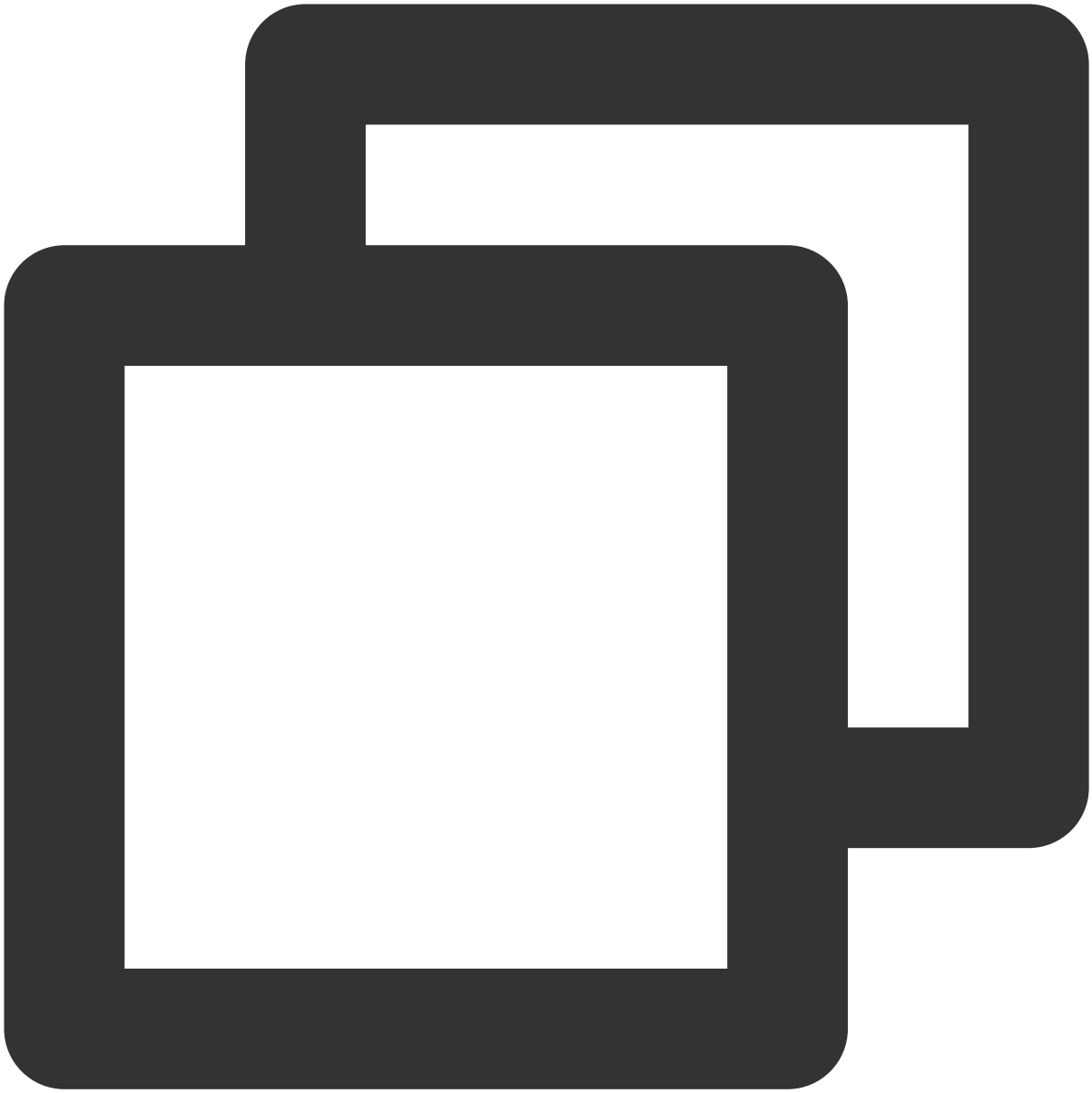
此处编码 `VUlOOiAxMjM0NTY3OA` 为文字信息 `UIN: 12345678` 经过 URL 安全的 base64 编码后生成。

3. 使用管道操作符拼接图片水印和文字水印的两段参数：



```
watermark/2/text/VUlOOiAxMjM0NTY3OA/font/SGVsdmV0aWNhLmRmb250/fontsize/36/fill/IzAw
```

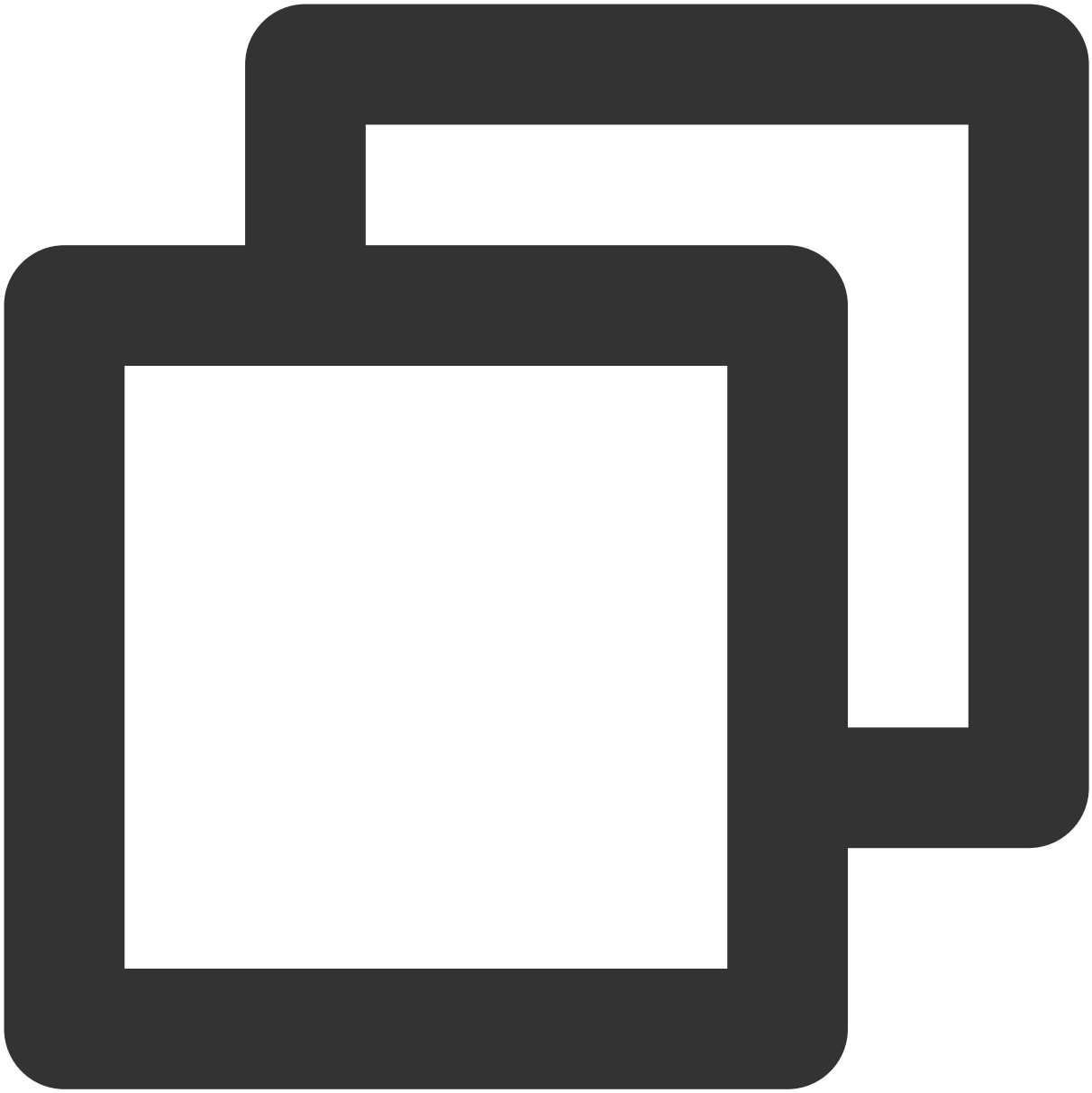
4. 将拼接的参数拼接至图片的下载链接后方：



```
https://examples-1258125638.cos.ap-guangzhou.myqcloud.com/preview.png?watermark/2/t
```

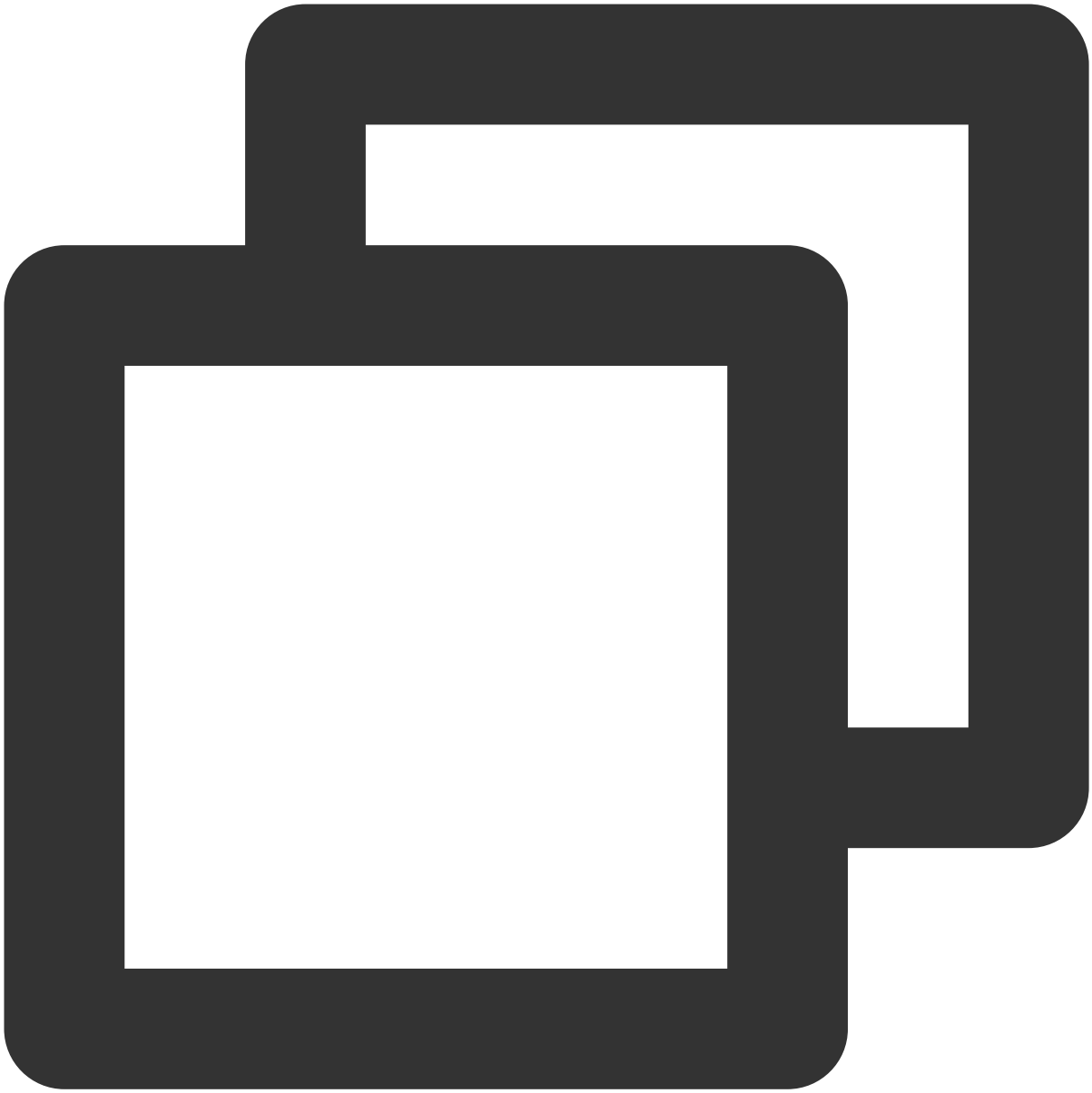
为了缩短 URL 长度，我们可以参考 样式管理 文档，在控制台将图片水印（保持不变）的部分新增为样式 `watermark1` ：

这样，链接即可缩短为：



```
https://examples-1258125638.cos.ap-guangzhou.myqcloud.com/preview.png/watermark1?wa
```

之后需要更改文字内容时，您只需将链接中 `VUlOOiAxMjM0NTY3OA` 部分替换为更新后的 base64 编码即可实现。例如 `UIN: 88888888` 的编码为 `VUlOOiA4ODg4ODg4OA`，此时只需将链接更改为如下内容，即可实现文字内容的替换。



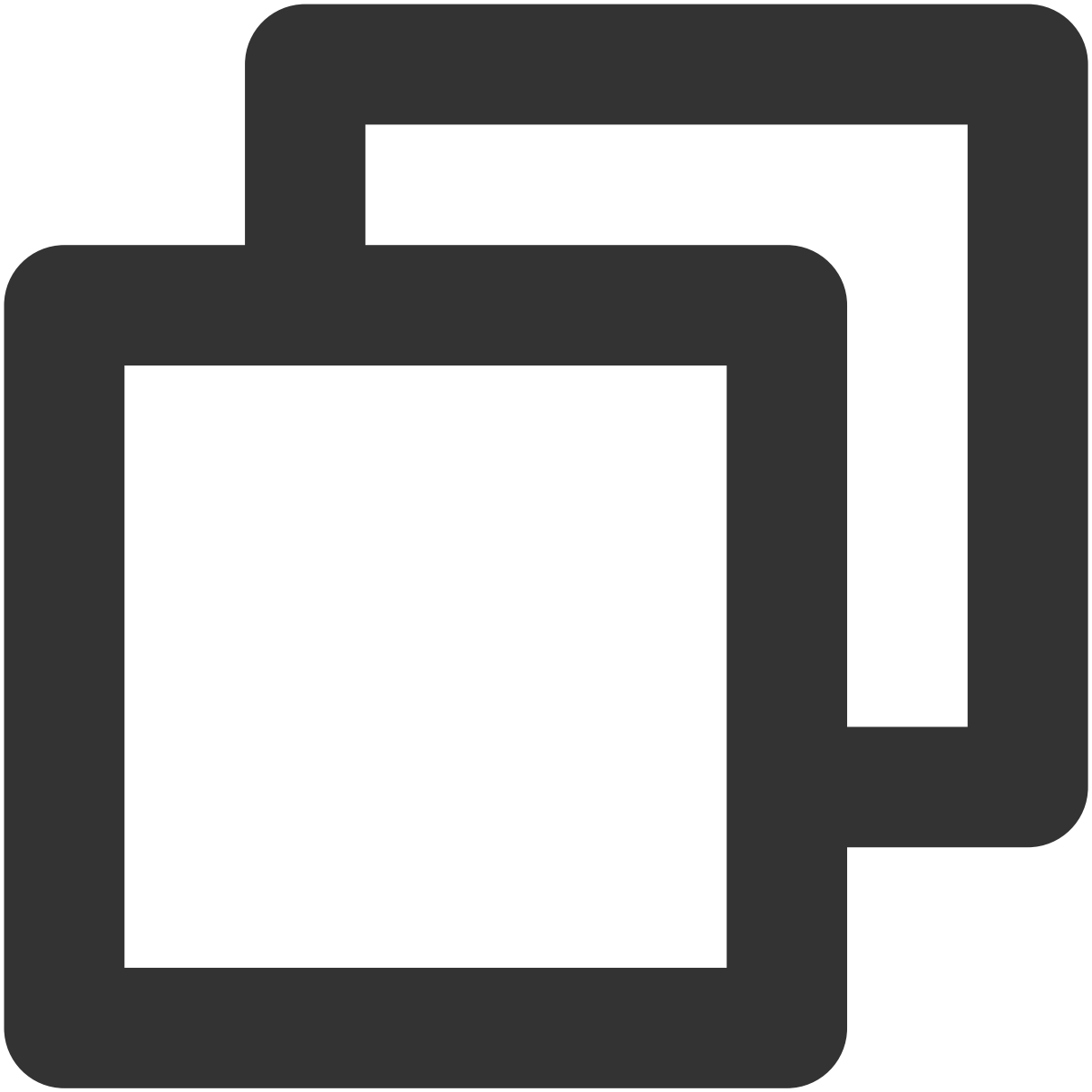
```
https://examples-1258125638.cos.ap-guangzhou.myqcloud.com/preview.png/watermark1?wa
```

方案二：将文字和图片水印打印到透明图片上，再将其作为图片水印

1. 准备一张400px * 400px尺寸的透明 PNG 图片，上传至存储桶中。详情可参见 上传文件 文档。

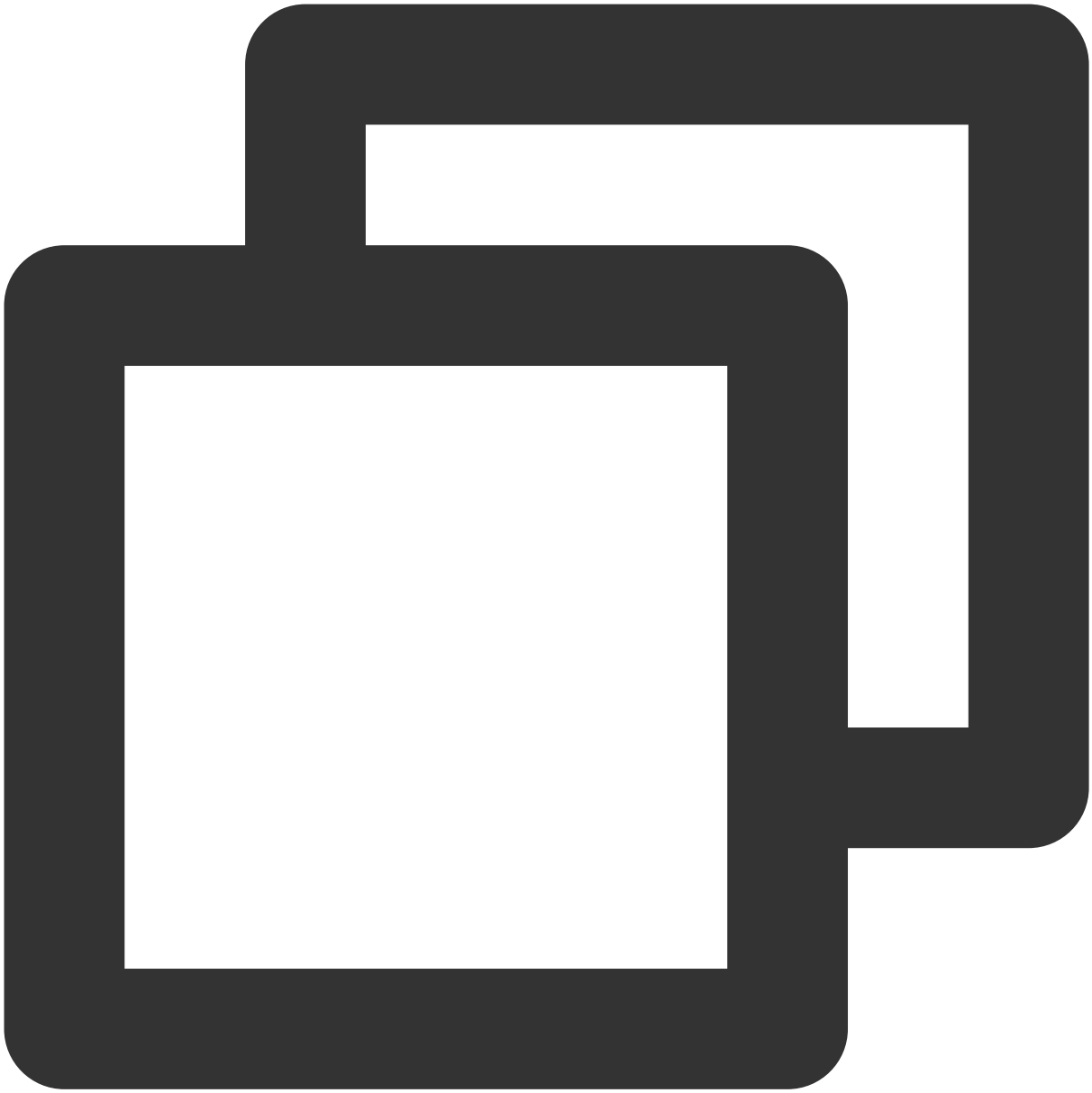
例如：`https://examples-1258125638.cos.ap-guangzhou.myqcloud.com/transparent.png`

2. 参见**方案一的步骤1 - 步骤3**，生成图片水印和文字水印两段参数，并将其拼接。
3. 将拼接的参数拼接至透明 PNG 图片的下载链接后方：



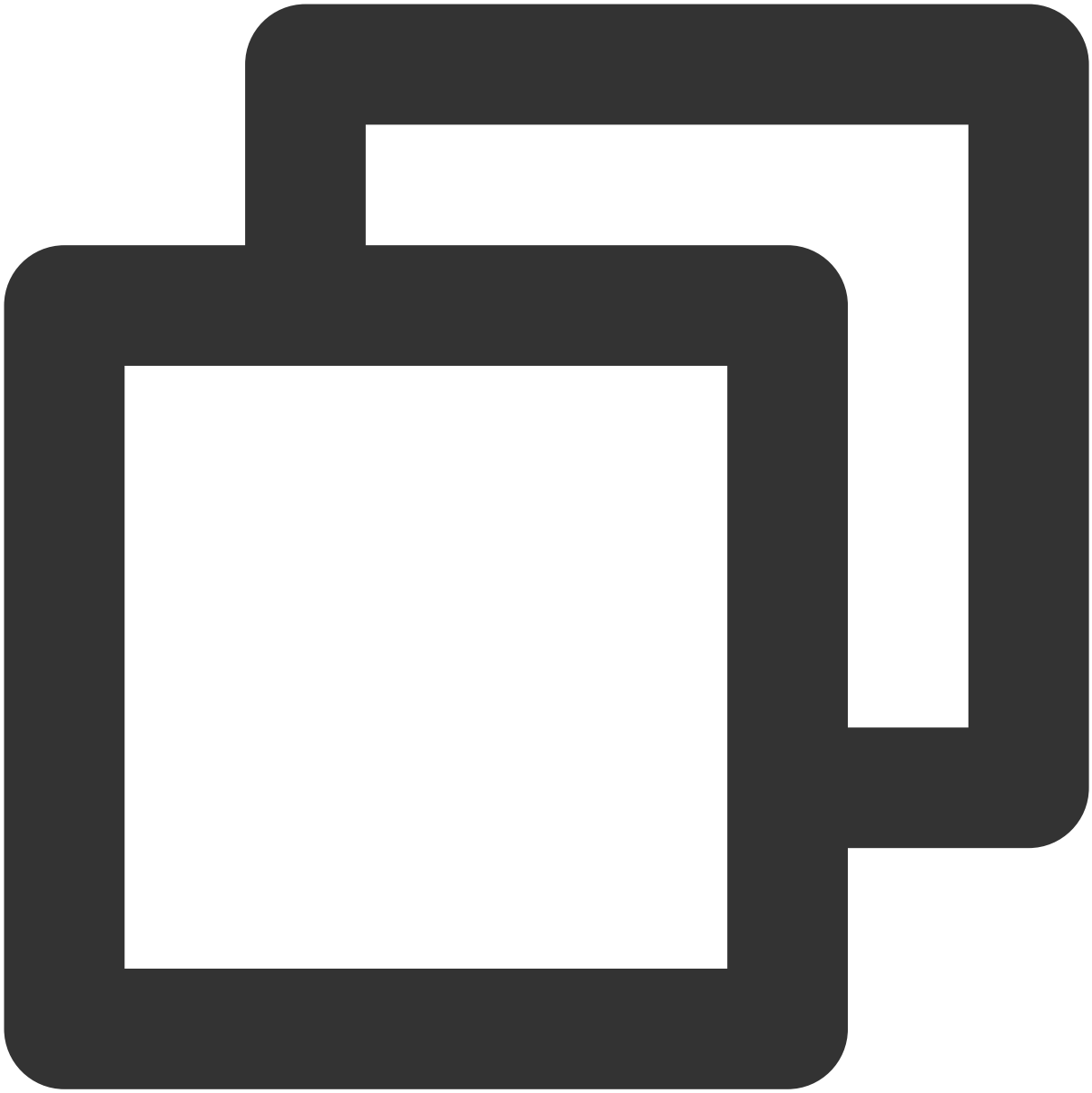
```
https://examples-1258125638.cos.ap-guangzhou.myqcloud.com/transparent.png?watermark
```

4. 将这张透明图片作为水印图片，对原图进行打水印操作即可：



```
https://examples-1258125638.cos.ap-guangzhou.myqcloud.com/preview.png?watermark/1/i
```

您还可以通过 `scatype` 参数使水印图尺寸根据图片大小等比例缩放，并通过 `batch` 参数设置平铺。



```
https://examples-1258125638.cos.ap-guangzhou.myqcloud.com/preview.png?watermark/1/i
```

COS 音视频播放器实践
COS 音视频播放器概述

最近更新时间：2024-01-06 10:54:03

本文主要介绍 COS 音视频云端处理与端侧播放如何进行实际应用，文中的实践案例涵盖音视频处理所支持的协议、功能以及如何播放 COS 音视频文件的操作指引，并结合 腾讯云数据万象（CI） 丰富的音视频处理能力，为您提供更多的产品功能使用思路并获得更好的播放性能体验。

协议支持

音视频协议	URL 地址格式	PC 浏览器	移动端浏览器
MP3	https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp3	支持	支持
MP4	https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4	支持	支持
HLS (M3U8)	https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8	支持	支持
FLV	https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.flv	支持	支持
DASH	https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mpd	支持	支持

注意

HLS、FLV、DASH 视频在部分浏览器环境中播放需要依赖 Media Source Extensions。

功能支持

功能	TCPlayer 播放器	DPlayer 播放器	Videojs 播放器
播放 MP4 格式视频	查看详情	查看详情	查看详情
播放 HLS 格式视频	查看详情	查看详情	查看详情

播放 FLV 格式视频	查看详情	查看详情	查看详情
播放 DASH 格式视频	查看详情	查看详情	查看详情
播放 PM3U8 (私有 M3U8) 视频	查看详情	查看详情	查看详情
设置封面图	查看详情	查看详情	查看详情
设置 HLS 标准加密	查看详情	查看详情	查看详情
切换清晰度	查看详情	查看详情	-
设置动态水印	查看详情	-	-
设置左上角 LOGO	-	查看详情	-
设置进度预览图	查看详情	-	-
设置字幕	查看详情	-	-
设置多语言	查看详情	-	-
设置贴片广告	查看详情	-	-

说明

播放器兼容常见的浏览器，播放器内部会自动区分平台，并使用最优的播放方案。例如在 Chrome 等现代浏览器中优先使用 HTML5 技术实现视频播放，而手机浏览器上会使用 HTML5 技术或者浏览器内核能力实现视频播放。

使用指引

使用 TCPlayer 播放 COS 视频文件
使用 DPlayer 播放 COS 视频文件
使用 VideojsPlayer 播放 COS 视频文件

使用 TCPlayer 播放 COS 视频文件

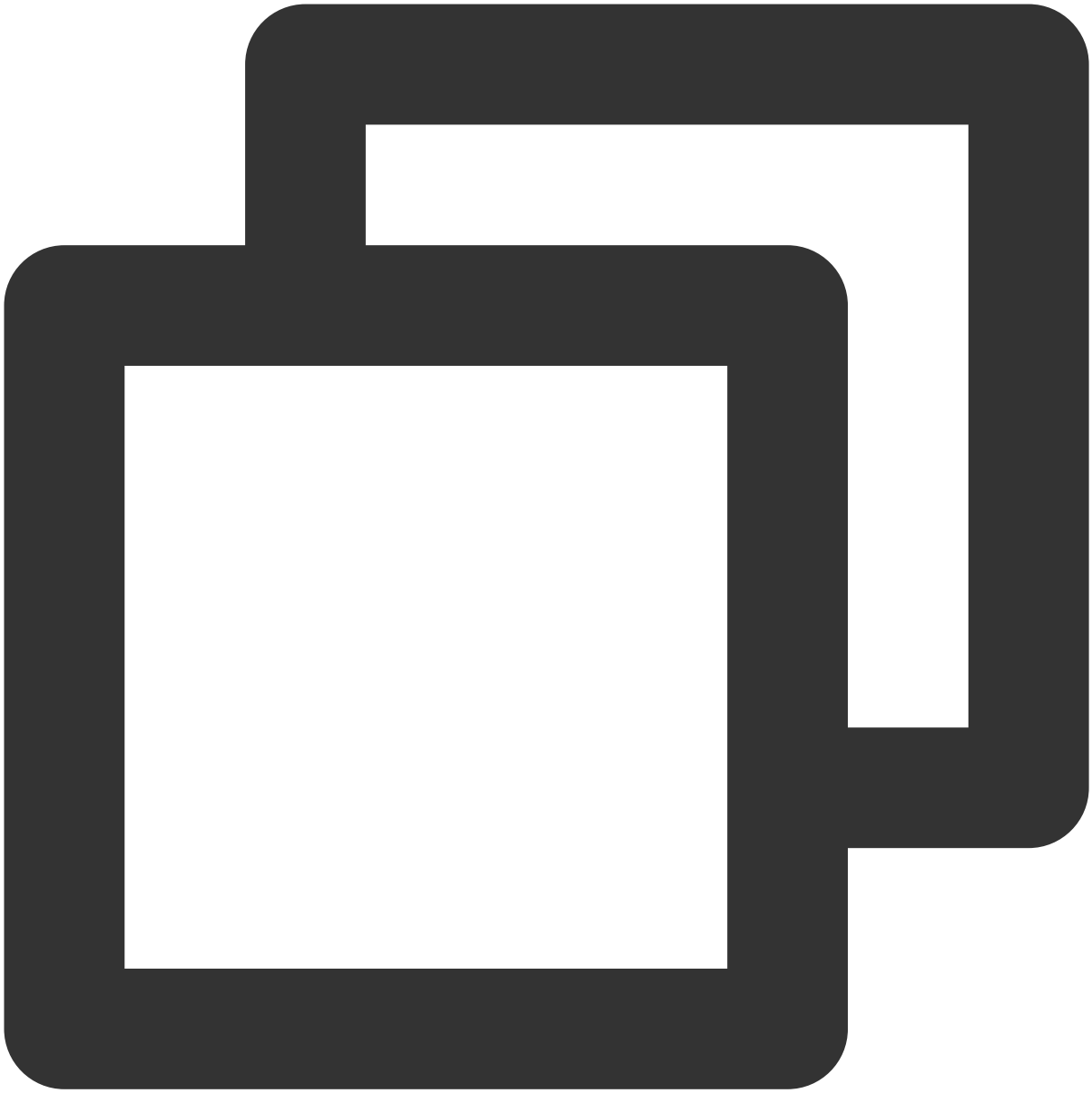
最近更新时间：2024-01-06 10:54:03

简介

本文将介绍如何使用音视频终端 SDK（腾讯云视立方）集成的 TCPlayer并结合 腾讯云数据万象(CI) 所提供的丰富的音视频能力，实现在 Web 浏览器播放 COS 视频文件。

集成指引

步骤1：在页面中引入播放器样式文件及脚本文件



```
<!--播放器样式文件-->
<link href="https://web.sdk.qcloud.com/player/tcplayer/release/v4.2.1/tcplayer.min.
<!--播放器脚本文件-->
<script src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.0/tcplayer.v4.
```

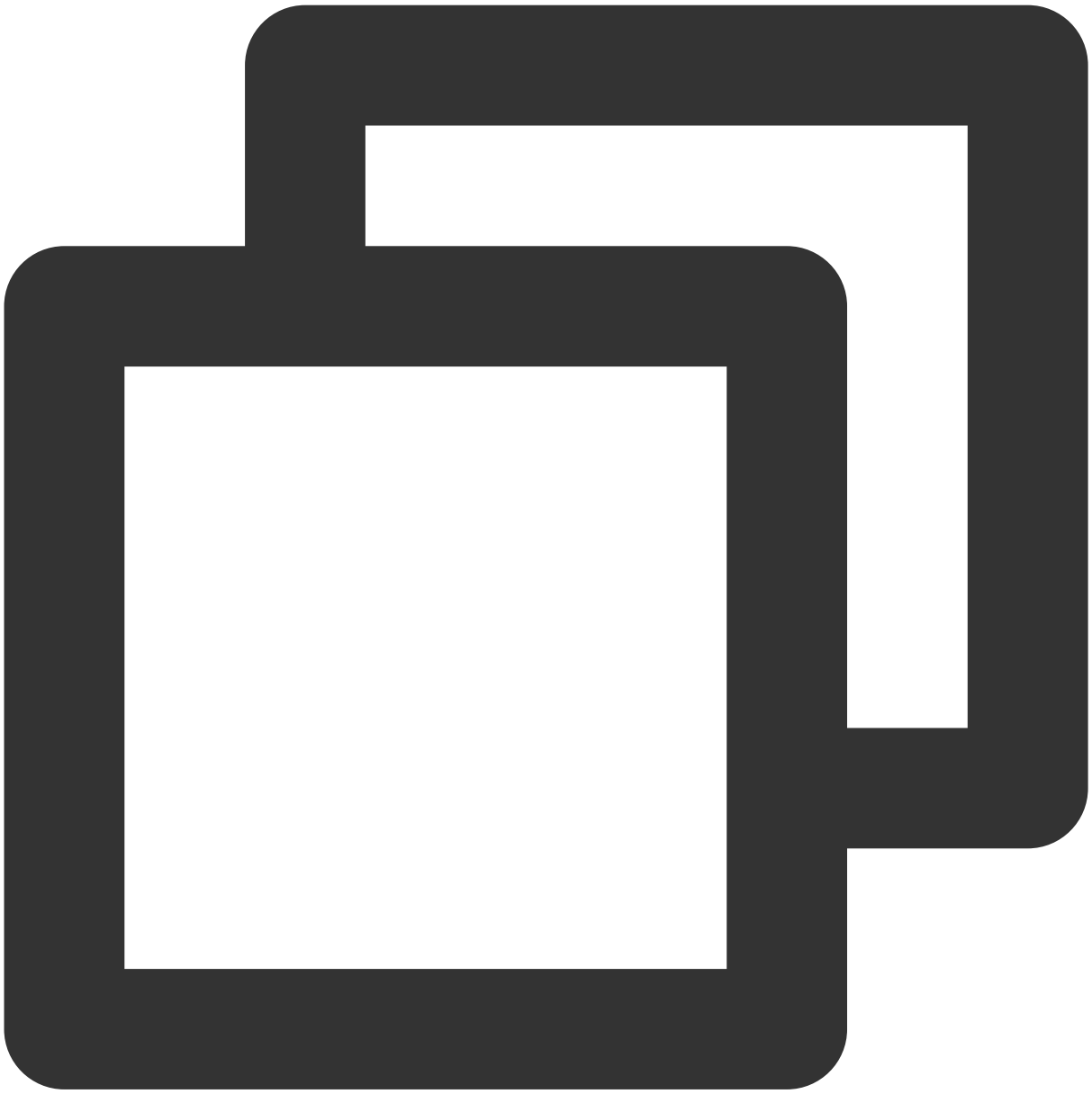
说明

建议在正式使用播放器 SDK 时，自行部署以上相关静态资源，单击下载播放器资源。

部署解压后的文件夹，不能调整文件夹里面的目录，避免资源互相引用异常。

步骤2：设置播放器容器节点

在需要展示播放器的页面位置加入播放器容器。例如，在 index.html 中加入如下代码（容器 ID 以及宽高都可以自定义）。



```
<video id="player-container-id" width="414" height="270" preload="auto" playsinline
</video>
```

说明

播放器容器必须为 `<video>` 标签。

示例中的 `player-container-id` 为播放器容器的 ID，可自行设置。

播放器容器区域的尺寸，建议通过 CSS 进行设置，通过 CSS 设置比属性设置更灵活，可以实现例如铺满全屏、容器自适应等效果。

示例中的 `preload` 属性规定是否在页面加载后载入视频，通常为了更快的播放视频，会设置为 `auto`，其他可选值：`meta`（当页面加载后只载入元数据），`none`（当页面加载后不载入视频），移动端由于系统限制不会自动加载视频。

`playsinline` 和 `webkit-playsinline` 这几个属性是为了在标准移动端浏览器不劫持视频播放的情况下实现行内播放，此处仅作示例，请按需使用。

设置 `x5-playsinline` 属性在 TBS 内核会使用 X5 UI 的播放器。

步骤3：获取视频文件对象地址

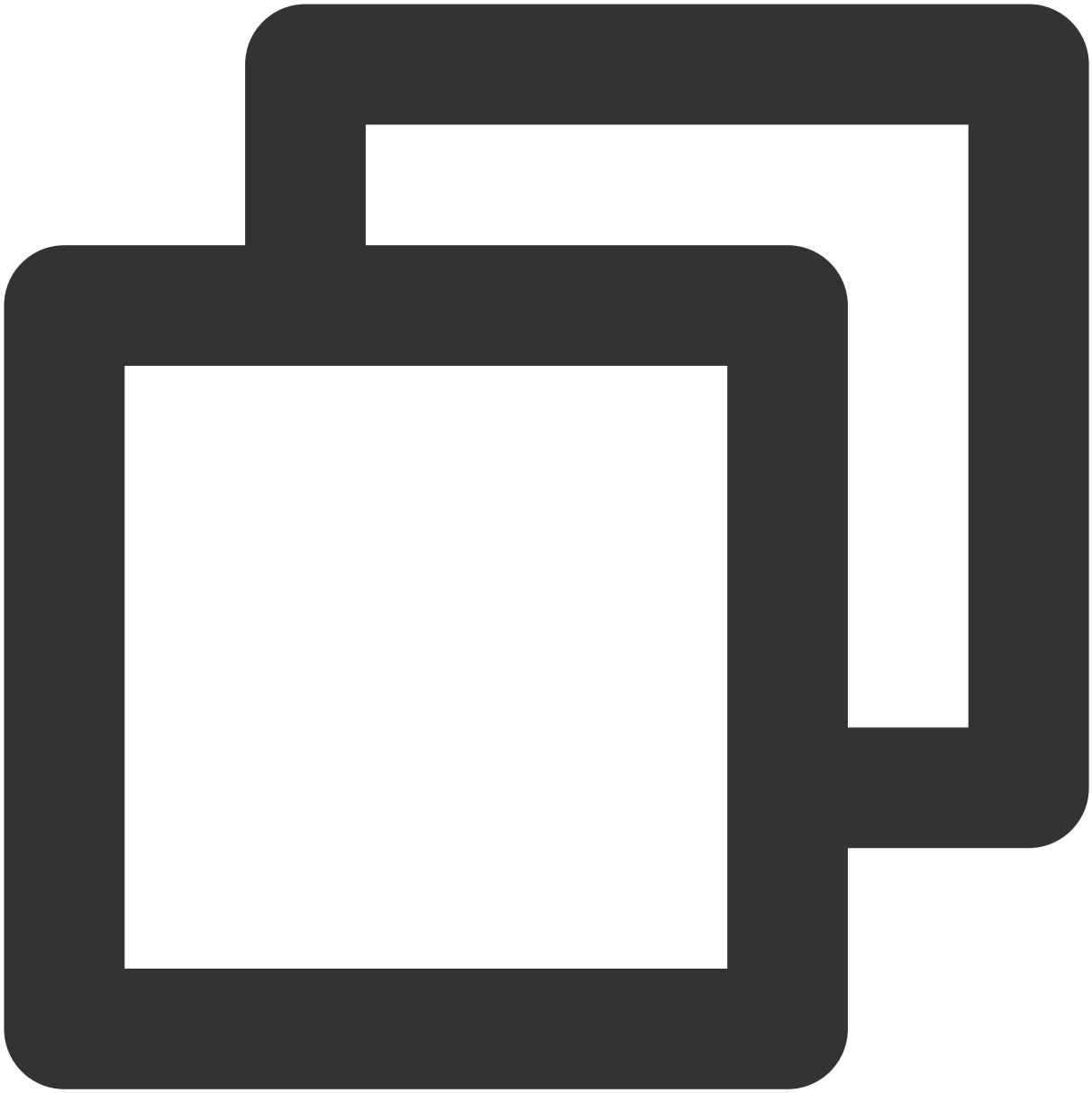
1. 创建一个存储桶。
2. 上传视频文件。
3. 获取视频文件对象地址，格式为：`https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.<视频格式>`。

说明

若存在跨域问题，则需要进行存储桶跨域访问 CORS 设置，详情请参见 设置跨域访问。

若存储桶为私有读写，则对象地址需要携带签名，详情请参见 请求签名。

步骤4：初始化播放器，并传入 COS 视频文件对象地址 URL



```
var player = TCPlayer("player-container-id", {}); // player-container-id 为播放器容器
player.src("https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4"); // COS
```

功能指引

播放不同格式的视频文件

1. 获取 COS 存储桶上的视频文件对象地址。

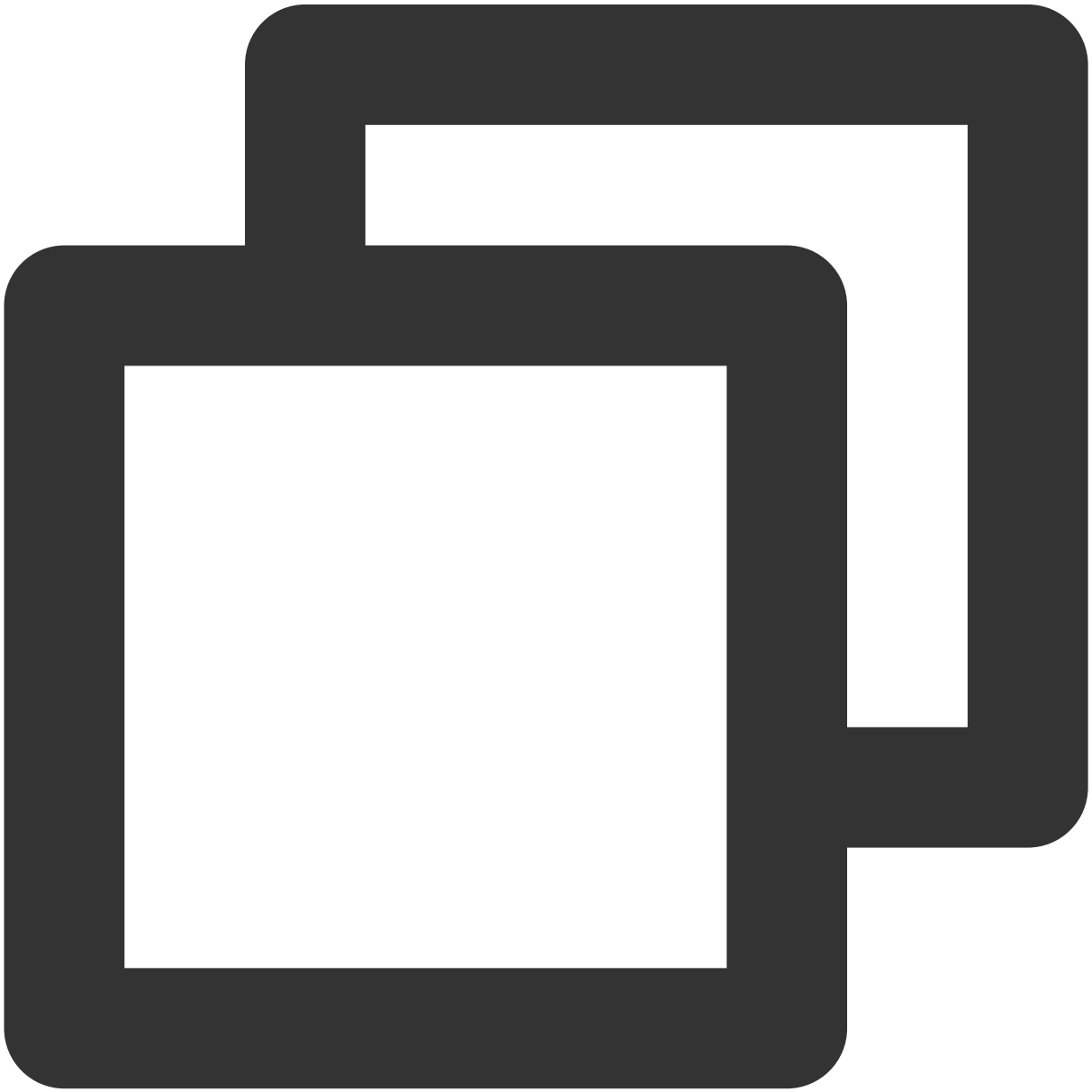
说明

未经转码的源视频在播放时有可能出现不兼容的情况，建议您使用转码后的视频进行播放，通过数据万象 音视频转码处理，获取不同格式视频文件。

2. 针对不同的视频格式，为了保证多浏览器的兼容性，需要引入对应的依赖。

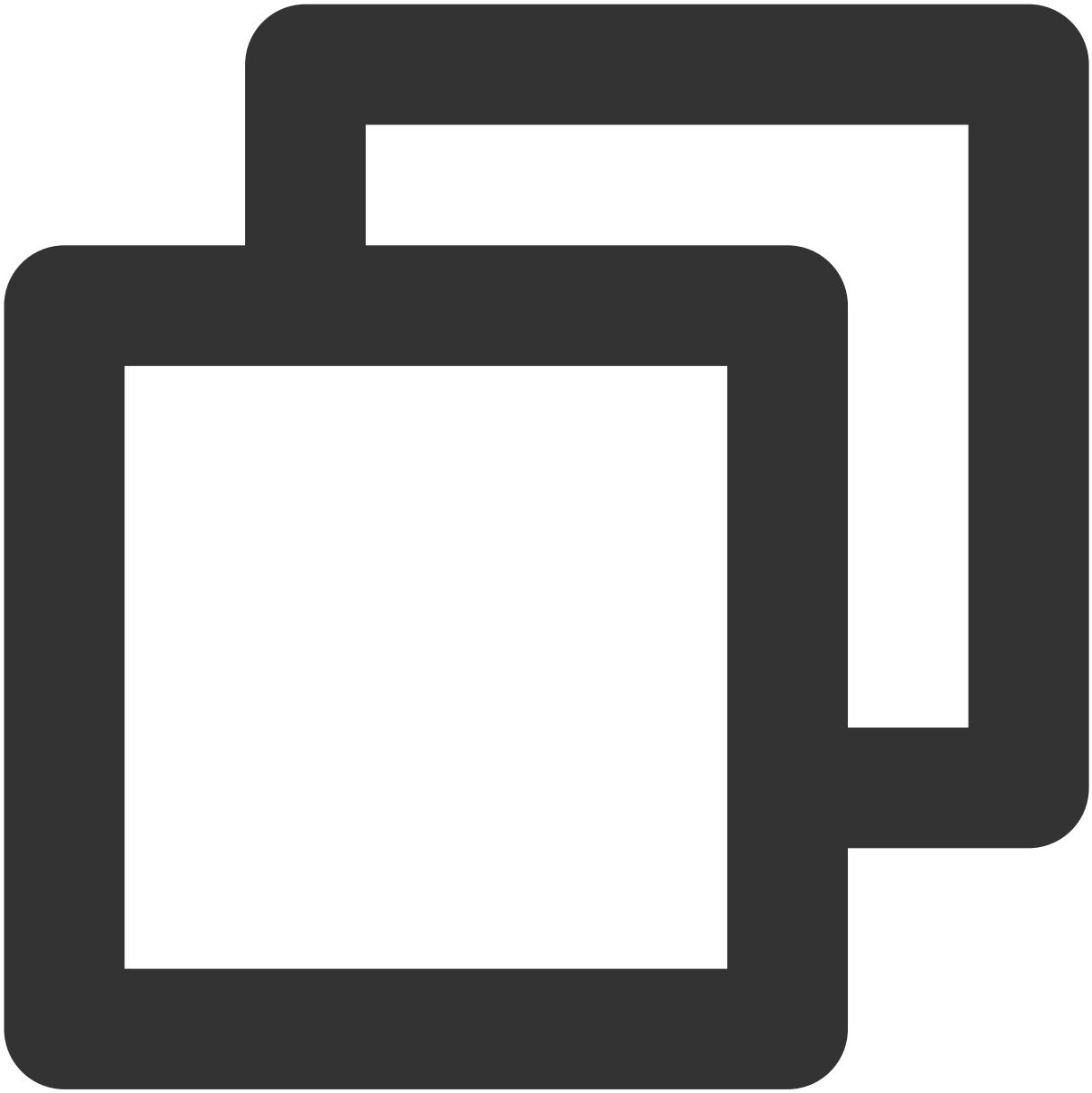
MP4：无需引入其他依赖。

HLS：如果需要在 Chrome 和 Firefox 等现代浏览器中通过 H5 播放 HLS 格式的视频，需要在 tcplayer.min.js 之前引入 hls.min.js。



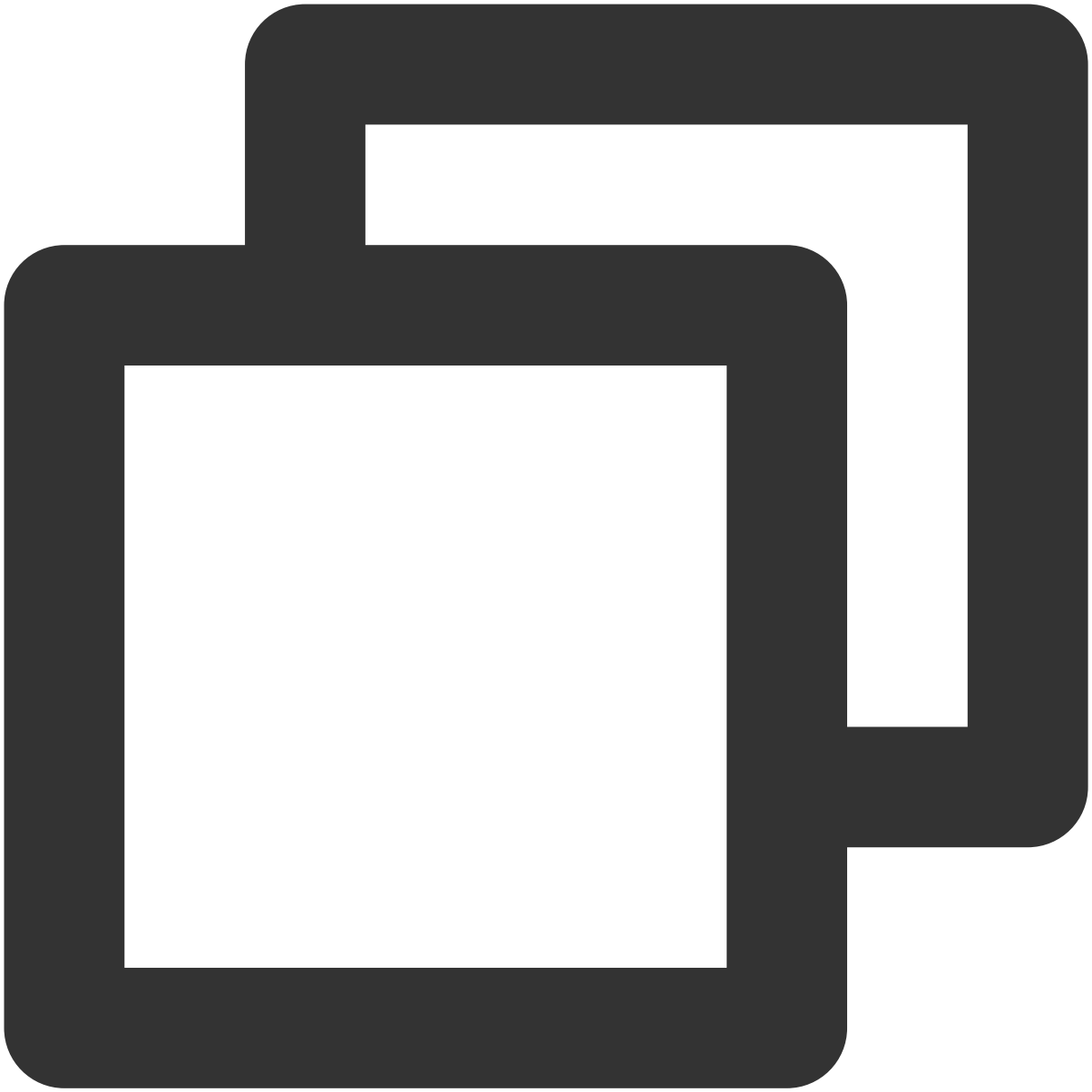
```
<script src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.2.1/libs/hls.m
```

FLV：如果需要在 Chrome 和 Firefox 等现代浏览器中通过 H5 播放 FLV 格式的视频，需要在 tcplayer.min.js 之前引入 flv.min.js。



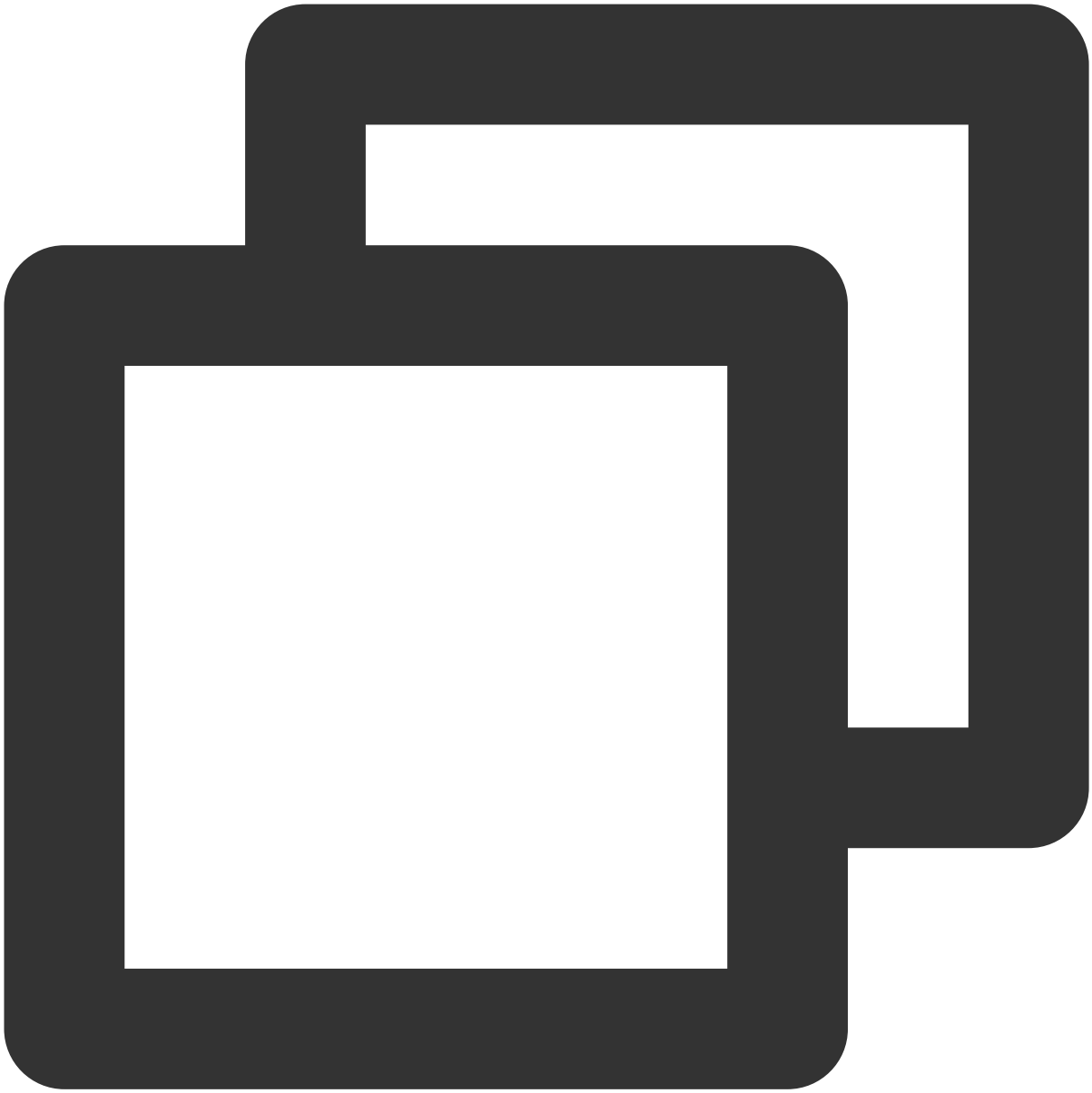
```
<script src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.2/libs/flv.m
```

DASH：DASH 视频需要加载 dash.all.min.js 文件。



```
<script src="https://cos-video-1258344699.cos.ap-guangzhou.myqcloud.com/lib/dash.
```

3. 初始化播放器并传入对象地址。



```
var player = TCPlayer("player-container-id", {}); // player-container-id 为播放器容器
player.src("https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4"); // COS
```

获取示例代码：

播放 MP4 示例代码

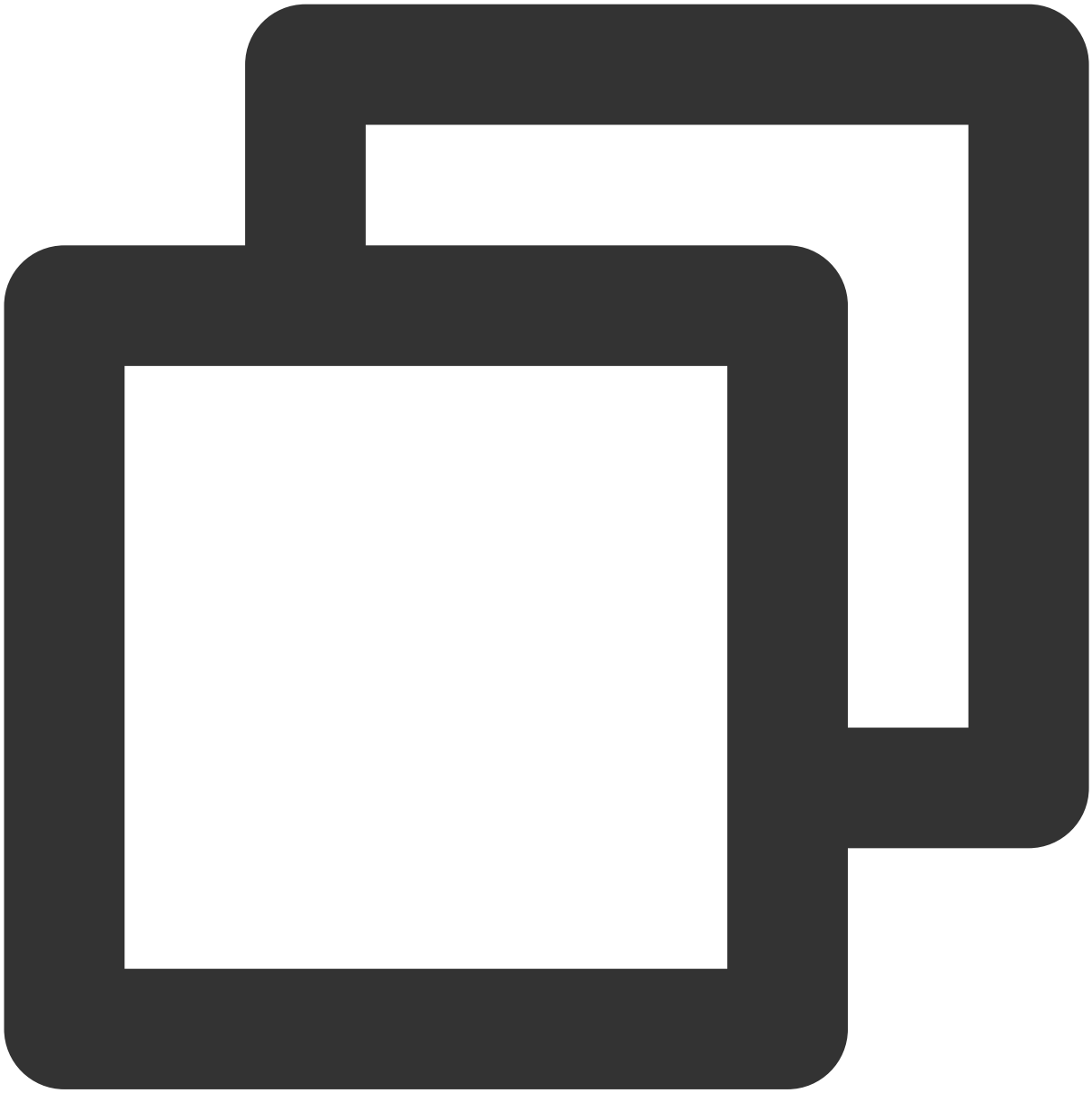
播放 FLV 示例代码

播放 HLS 示例代码

播放 DASH 示例代码

播放 PM3U8 视频

PM3U8 是指私有的 M3U8 视频文件，COS 提供用于获取私有 M3U8 TS 资源的下载授权API，可参见 私有 M3U8 接口。



```
var player = TCPlayer("player-container-id", {
    poster: "https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8?ci-proce
});
```

获取示例代码：

播放 PM3U8 示例代码

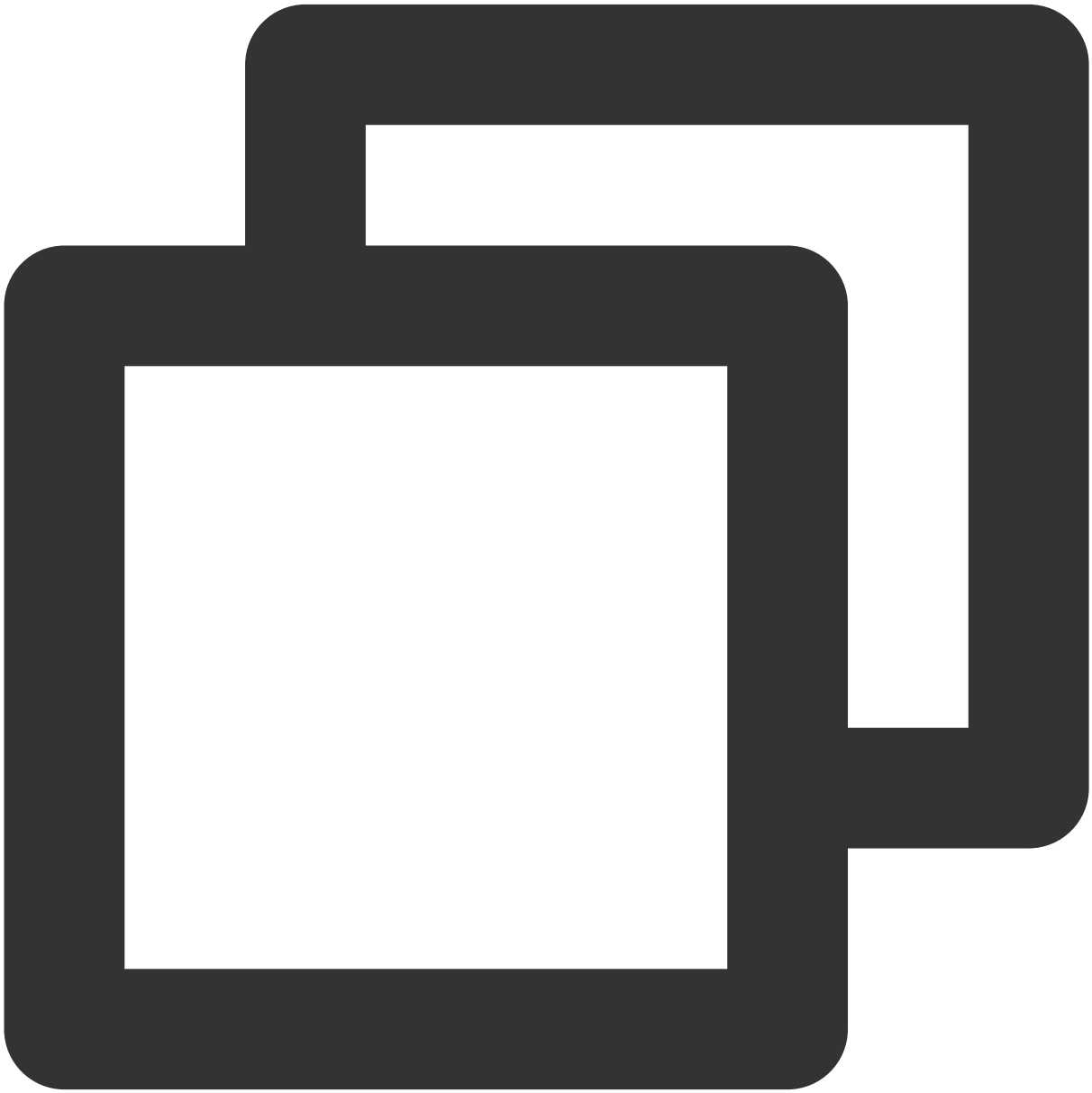
设置封面图

1. 获取 COS 存储桶上的封面图对象地址。

注意

通过数据万象 智能封面 能力，提取最优帧生成截图作为封面，可提升内容吸引力。

2. 设置封面图。



```
var player = TCPlayer("player-container-id", {
 poster: "https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.png"
```

```
});
```

获取示例代码：

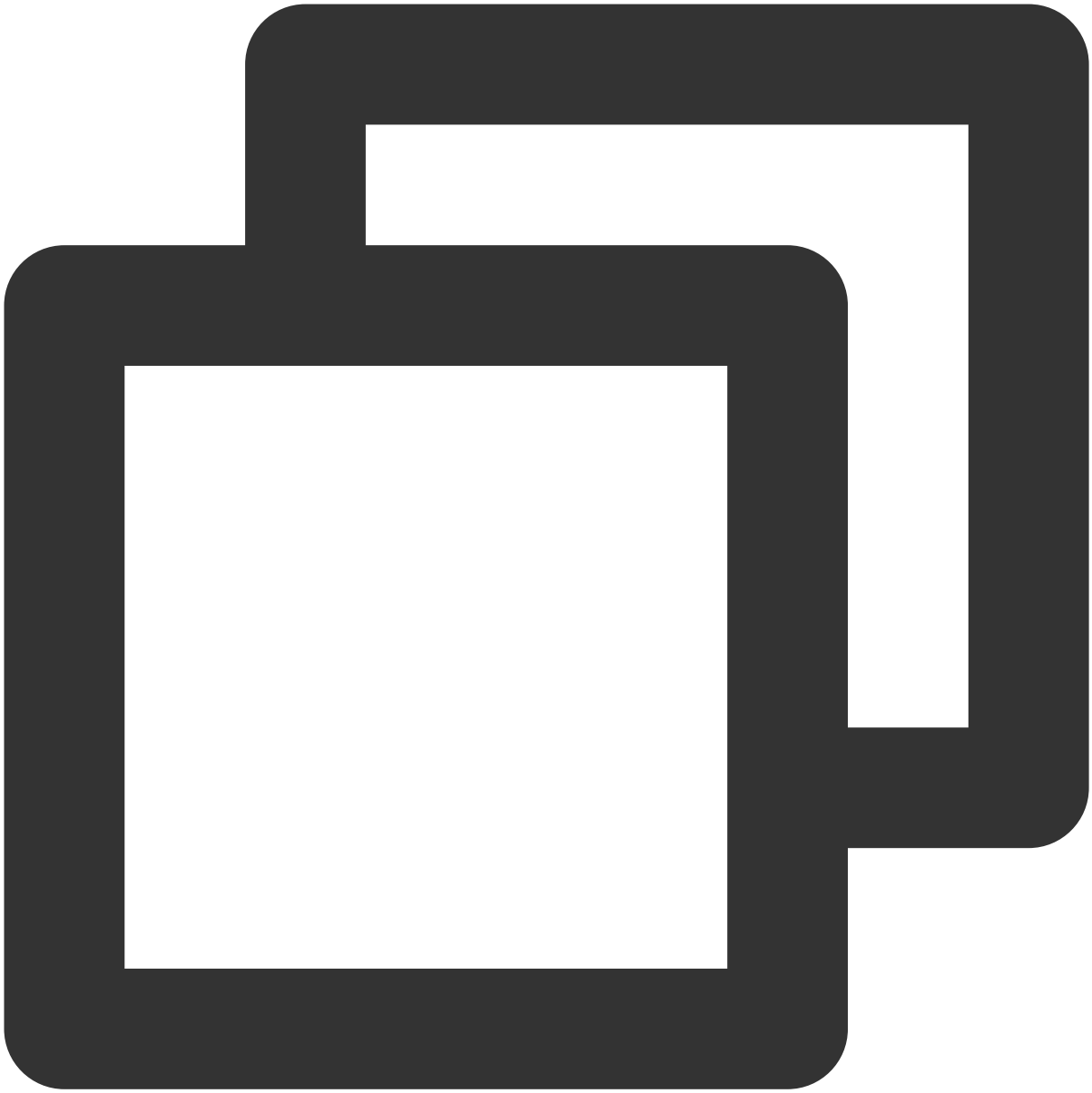
设置封面图示例代码

播放 HLS 加密视频

为了保障视频内容安全，防止视频被非法下载和传播，数据万象提供了对 HLS 视频内容进行加密的功能，拥有相比于私有读文件更高的安全级别。加密后的视频，无法分发给无访问权限的用户观看。即使视频被下载到本地，视频本身也是被加密的，无法恶意二次分发，从而保障您的视频版权不受到非法侵犯。

操作步骤如下：

1. 参见 播放 HLS 加密视频 流程，生成加密视频。
2. 初始化播放器并传入视频对象地址。



```
var player = TCPlayer("player-container-id", {}); // player-container-id 为播放器容器
player.src("https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8"); // hls
```

获取示例代码：

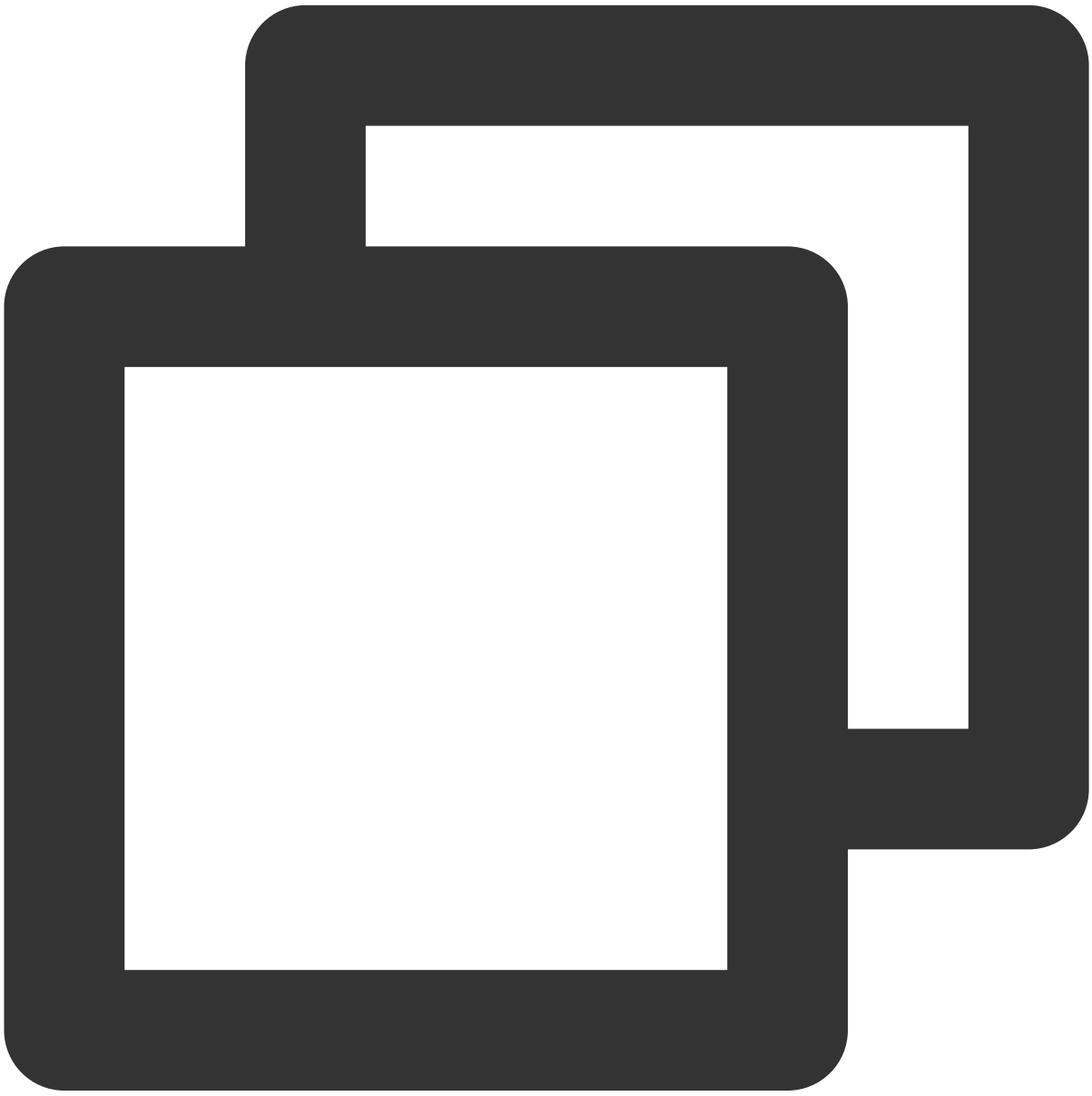
播放 HLS 加密视频示例代码

切换清晰度

数据万象 自适应码流 功能，可以将视频文件转码并打包生成自适应码流输出文件，帮助用户在不同网络情况下快速分发视频内容，播放器能够根据当前带宽，动态选择最合适的码率播放。

操作步骤如下：

1. 通过 数据万象 自适应码流 功能，生成多码率自适应的 HLS 或 DASH 目标文件。
2. 初始化播放器并传入视频对象地址。



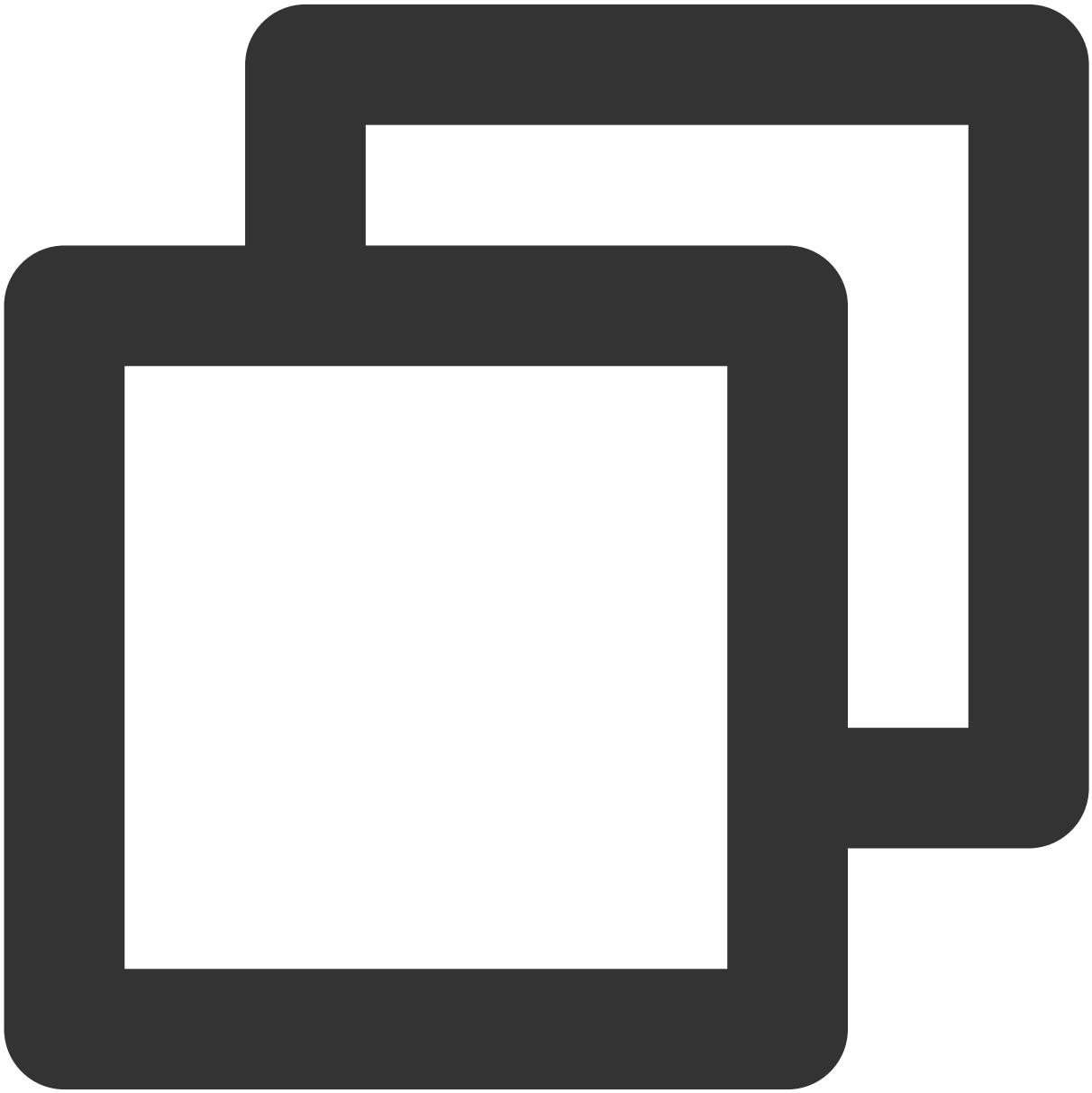
```
var player = TCPlayer("player-container-id", {}); // player-container-id 为播放器容器
player.src("https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8"); // 多码
```

获取示例代码：

切换清晰度示例代码

设置动态水印

播放器支持为视频添加位置与速度产生变换的水印。在使用动态水印功能时，播放器对象的引用不能暴露到全局环境，否则动态水印可以轻易去除，数据万象也支持在云端对视频进行添加动态水印等操作，详情可参见水印模板接口。



```
var player = TCPlayer("player-container-id", {
    plugins:{
```

```
        DynamicWatermark: {
          speed: 0.2, // 速度
          content: "腾讯云数据万象 CI", // 文案
          opacity: 0.7 // 透明度
        }
      }
    });
```

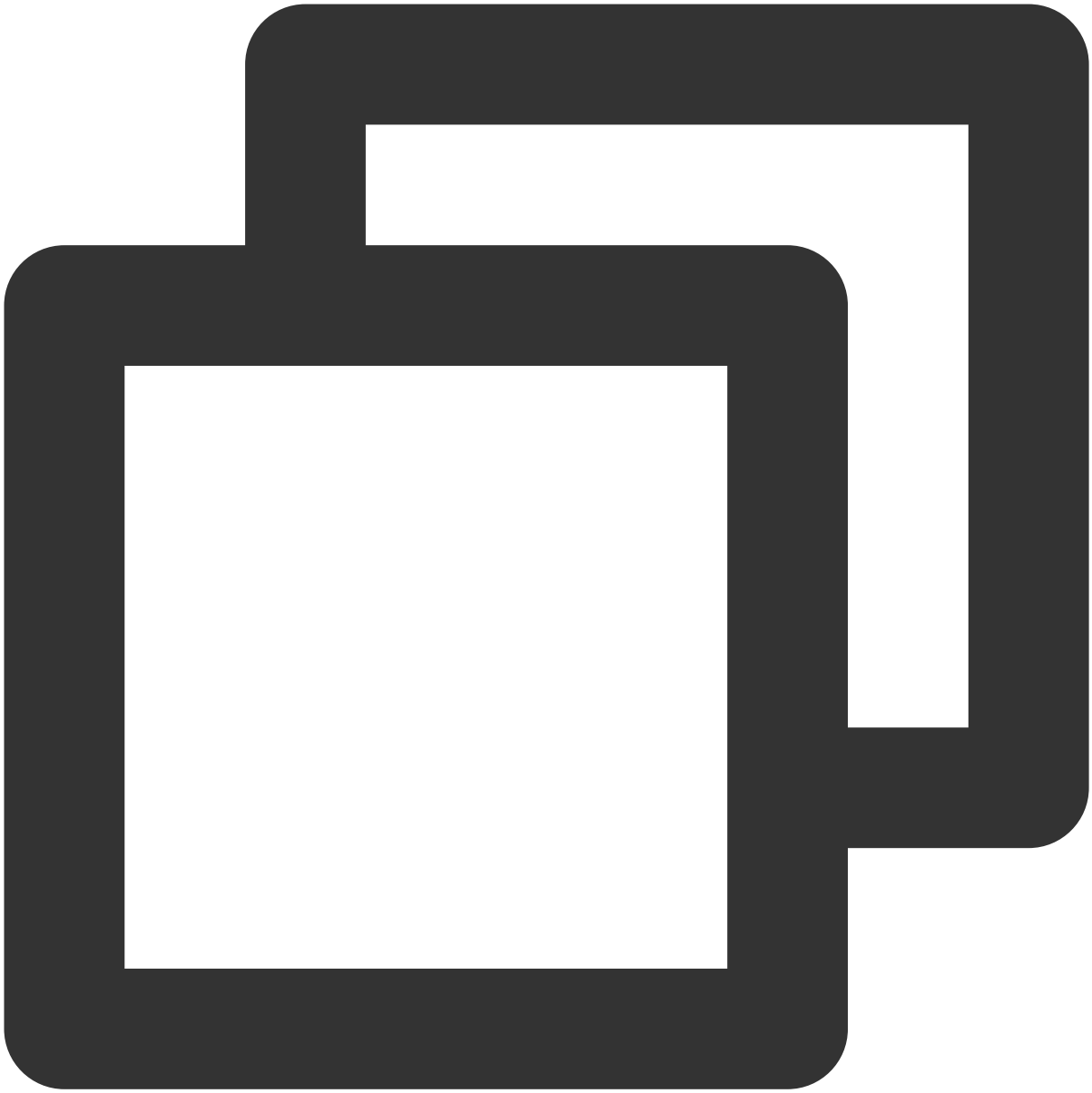
获取示例代码：

设置动态水印

设置贴片广告

操作步骤如下：

1. 准备视频广告封面图以及广告链接。
2. 初始化播放器，设置广告封面图和链接，并设置广告节点。



```
var PosterImage = TCPlayer.getComponent('PosterImage');
PosterImage.prototype.handleClick = function () {
 window.open('https://www.tencentcloud.com/products/ci'); // 设置广告链接
};

var player = TCPlayer('player-container-id', {
 poster: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx..png', // 广告封面
});
player.src('https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4');

var adTextNode = document.createElement('span');
```

```
adTextNode.className = 'ad-text-node';
adTextNode.innerHTML = '广告';

var adCloseIconNode = document.createElement('i');
adCloseIconNode.className = 'ad-close-icon-node';
adCloseIconNode.onclick = function (e) {
  e.stopPropagation();
  player.posterImage.hide();
};

player.posterImage.el_.appendChild(adTextNode);
player.posterImage.el_.appendChild(adCloseIconNode);
```

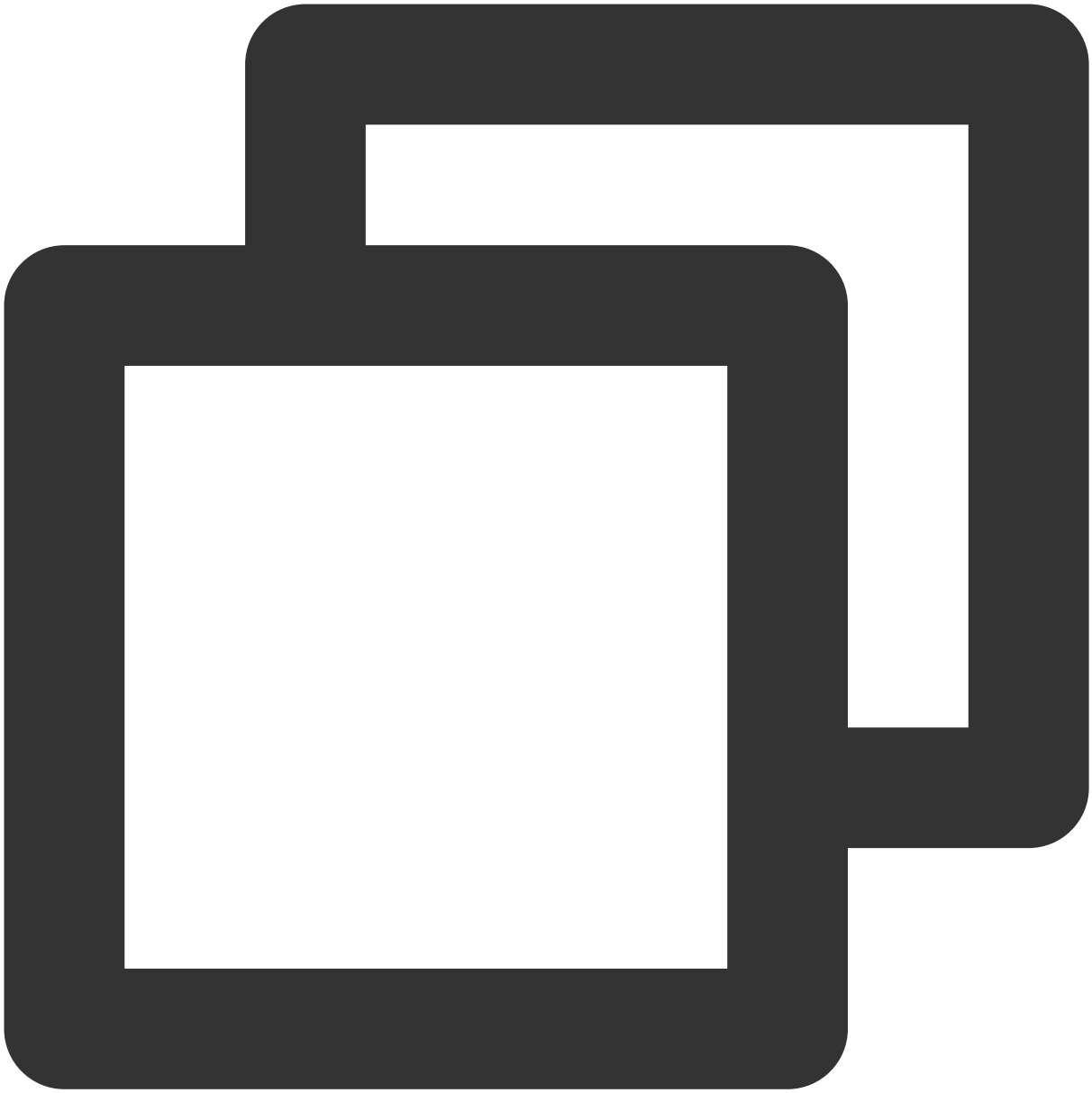
获取示例代码：

设置贴片广告示例代码

设置视频进度图

操作步骤如下：

1. 通过数据万象 视频截帧 并生成雪碧图。
2. 获取步骤1生成的雪碧图和 VTT（雪碧图位置描述文件） 对象地址。
3. 初始化播放器，并设置视频地址和 VTT 文件。



```
var player = TCPlayer('player-container-id', {
  plugins: {
 VttThumbnail: {
   vttUrl: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.vtt' // 进度图
 },
  },
});
player.src('https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4');
```

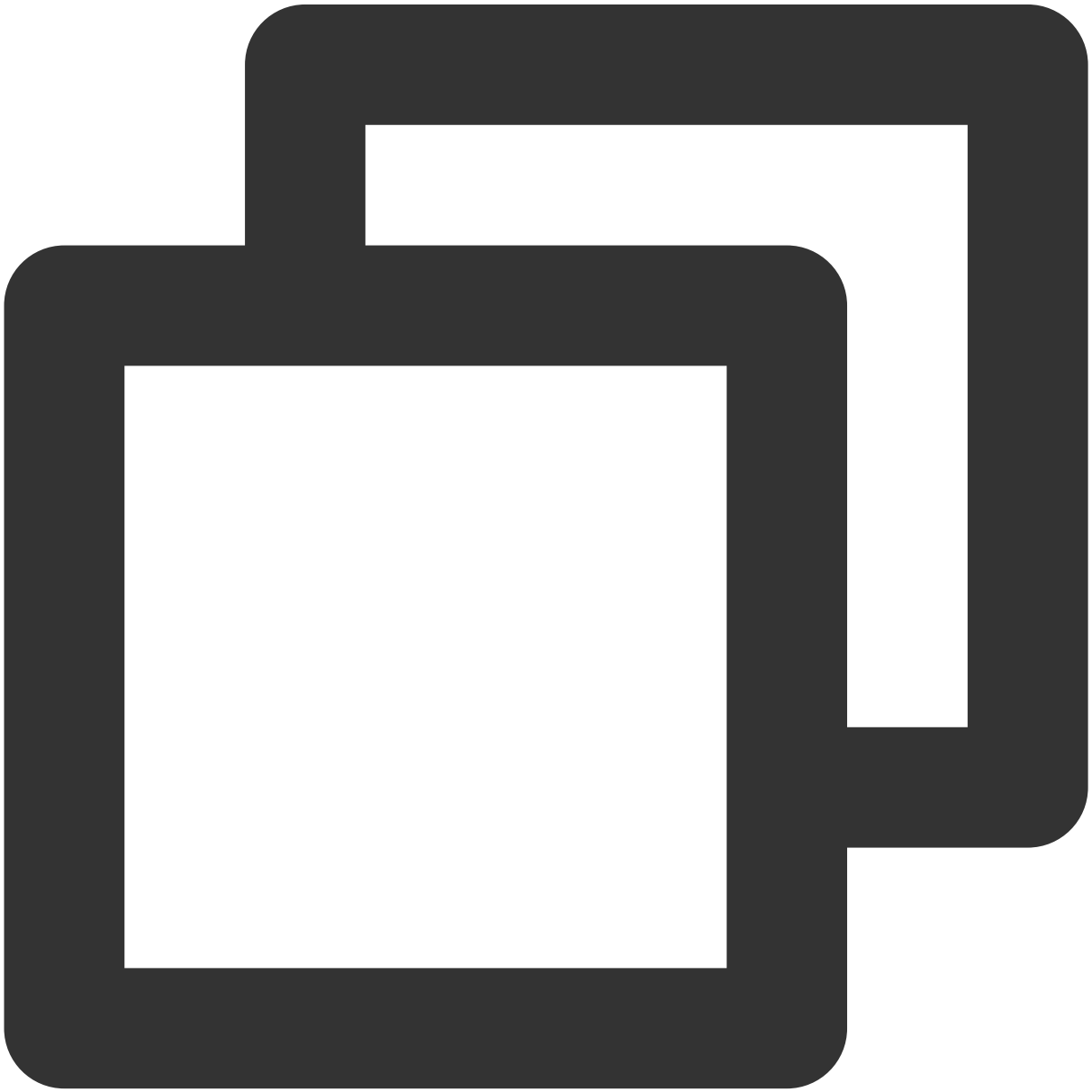
获取示例代码：

设置视频进度图示例代码

设置视频字幕

操作步骤如下：

1. 通过数据万象语音识别 并生成字幕文件。
2. 获取步骤1生成的字幕 SRT 文件对象地址。
3. 初始化播放器，并设置视频地址和字幕 SRT 文件。



```
var player = TCPlayer('player-container-id', {});
```

```
player.src('https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4');
player.on('ready', function () {
  // 添加字幕文件
  var subTrack = player.addRemoteTextTrack({
 src: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.srt', // 字幕文件
 kind: 'subtitles',
 srclang: 'zh-cn',
 label: '中文',
 default: 'true',
  }, true);
});
```

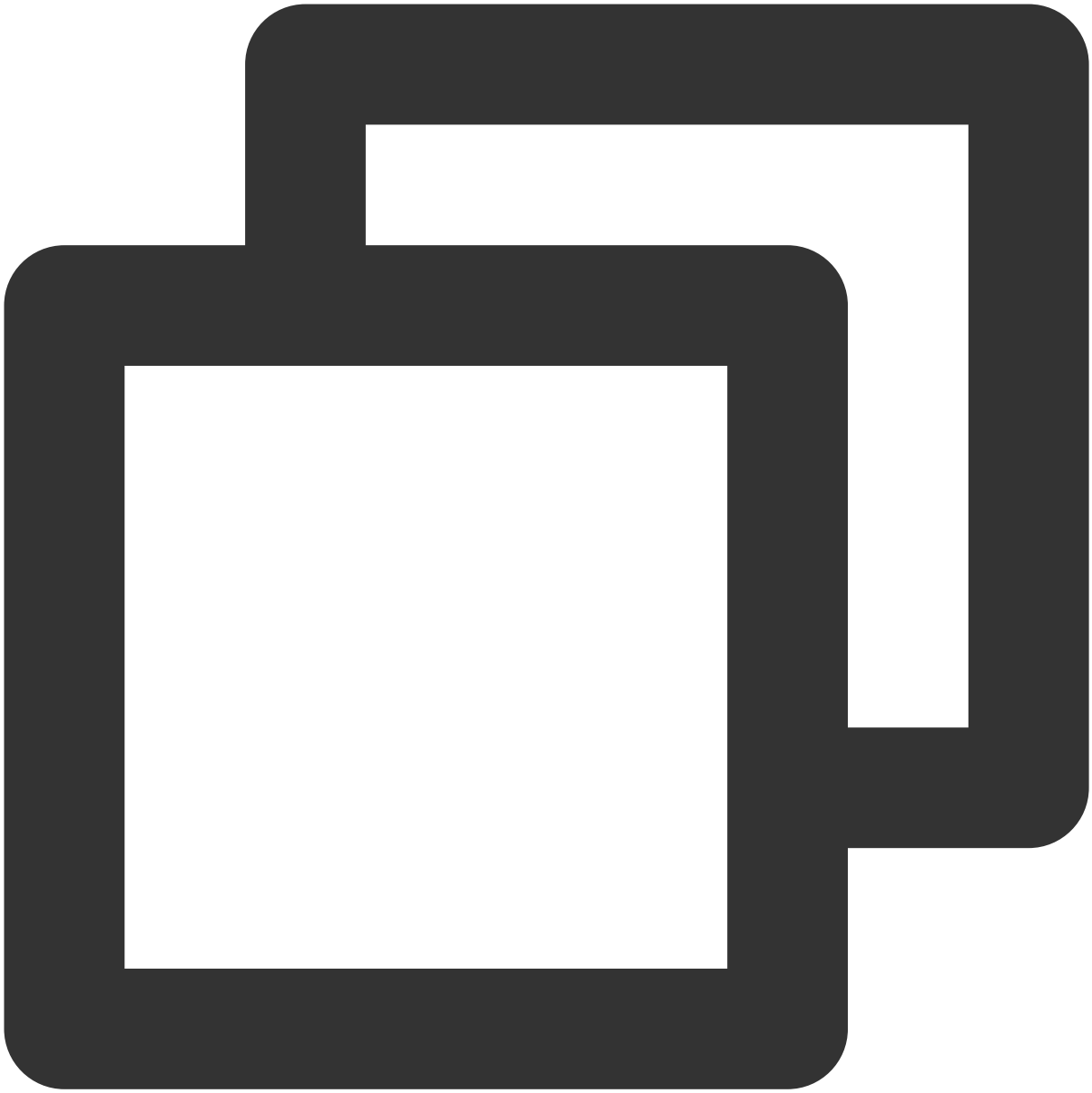
获取示例代码：

设置视频字幕示例代码

设置视频多语言字幕

操作步骤如下：

1. 通过数据万象语音识别生成字幕文件，并同时翻译成多种语言。
2. 获取步骤1生成的多语言字幕 SRT 文件对象地址。
3. 初始化播放器，并设置视频地址和多语言字幕 SRT 文件。



```
var player = TCPlayer('player-container-id', {});
player.src('https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4');
player.on('ready', function () {
  // 设置中文字幕
  var subTrack = player.addRemoteTextTrack({
 src: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/zh.srt', // 字幕文件
 kind: 'subtitles',
 srclang: 'zh-cn',
 label: '中文',
 default: 'true',
  }, true);
```

```
    // 设置英文字幕
    var subTrack = player.addRemoteTextTrack({
  src: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/en.srt', // 字幕文件
  kind: 'subtitles',
  srclang: 'en',
  label: '英文',
  default: 'false',
    }, true);
});
```

获取示例代码：

设置视频多语言字幕示例代码

使用 DPlayer 播放 COS 视频文件

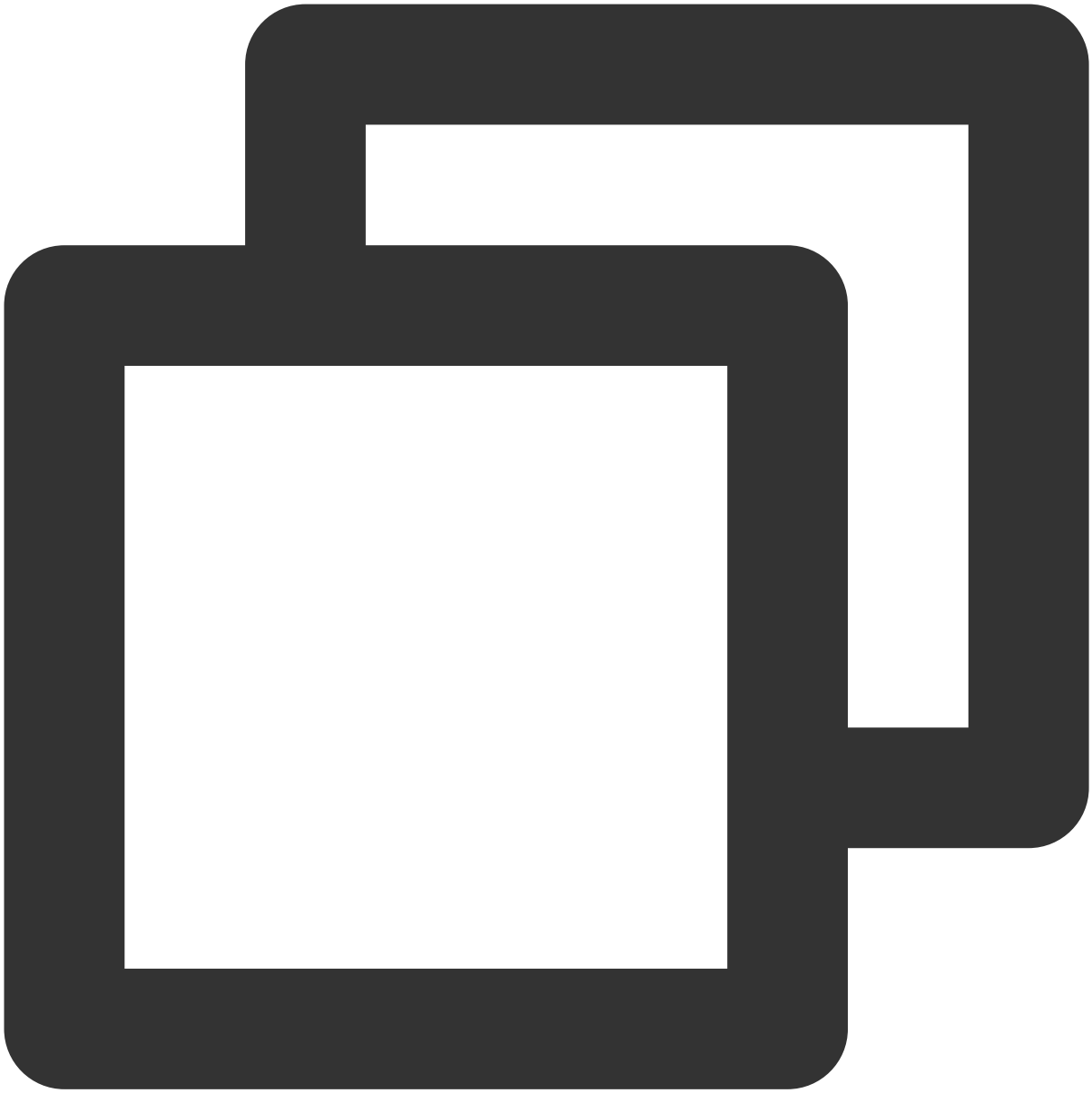
最近更新时间：2024-01-06 10:54:03

简介

本文将介绍如何使用 DPlayer 并结合 腾讯云数据万象（CI） 所提供的丰富的音视频能力，实现在 Web 浏览器播放 COS 视频文件。

集成指引

步骤1：在页面中引入播放器脚本文件以及按需引入部分依赖文件



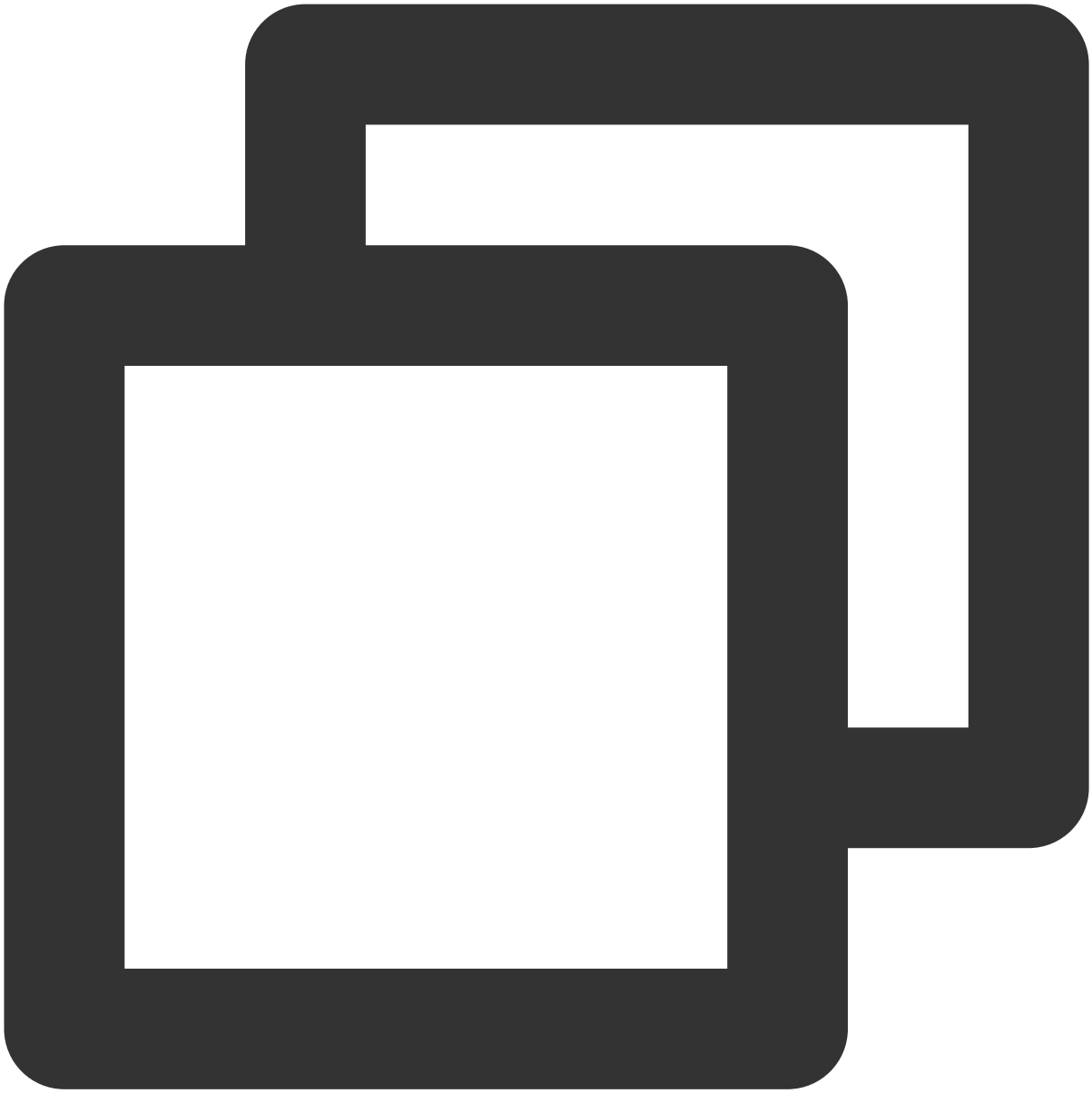
```
<!-- 播放器脚本文件 -->
<script src="https://cdn.jsdelivr.net/npm/dplayer@1.26.0/dist/DPlayer.min.js"></scr
```

说明

建议在正式使用播放器时，自行部署以上相关静态资源。

步骤2：设置播放器容器节点

在需要展示播放器的页面位置加入播放器容器。例如，在 index.html 中加入如下代码（容器 ID 以及宽高都可以自定义）。



```
<div id="dplayer" style="width: 100%; height: 100%"></div>
```

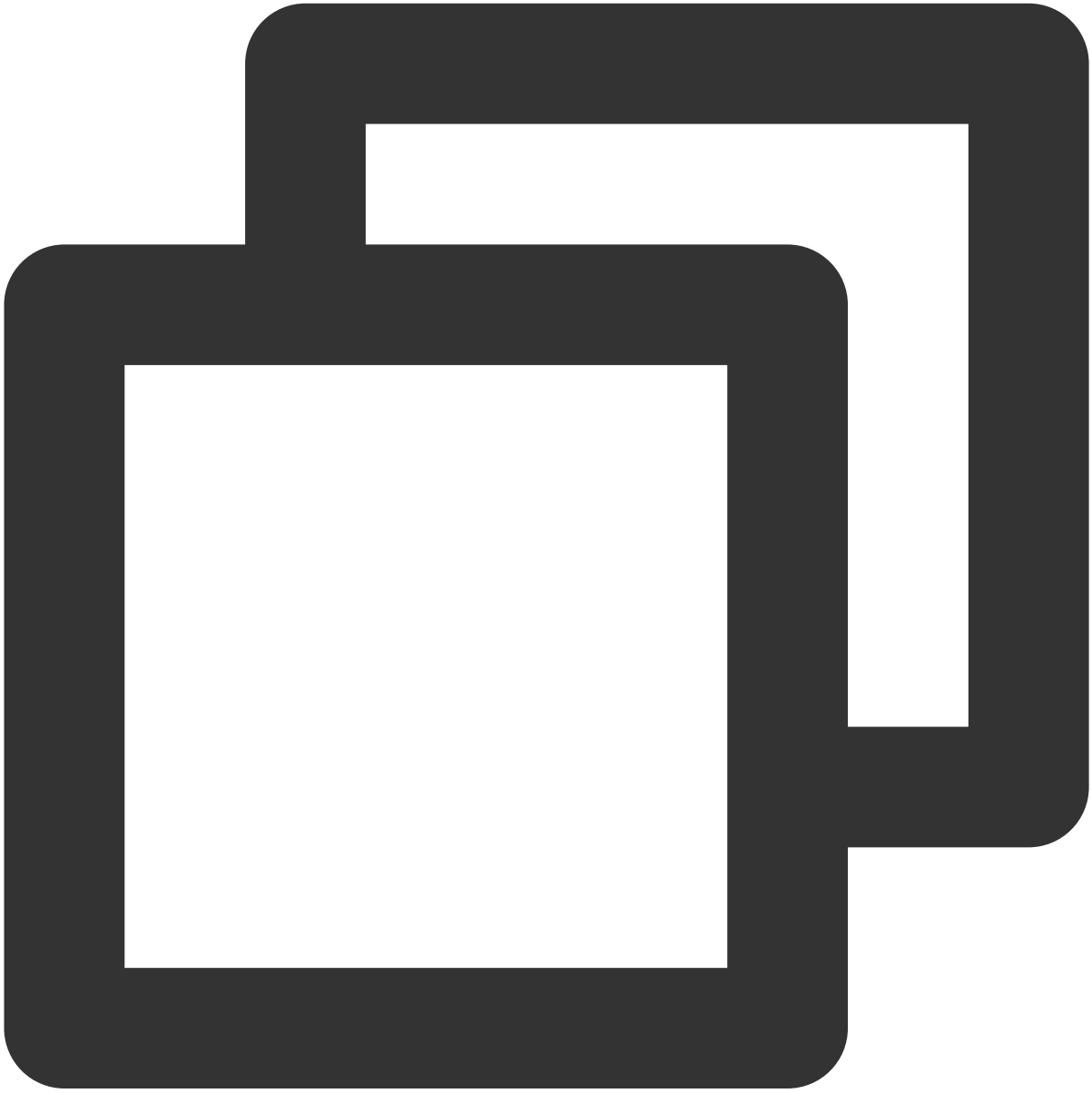
步骤3：获取视频文件对象地址

1. 创建一个存储桶
2. 上传视频文件
3. 获取视频文件对象地址，格式为：`https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.<视频格式>`。

说明

若存在跨域问题，则需要进行存储桶跨域访问 CORS 设置，详情请参见 设置跨域访问。
若存储桶为私有读写，则对象地址需要携带签名，详情请参见 请求签名。

步骤4：初始化播放器，并传入 COS 视频文件对象地址 URL



```
const dp = new DPlayer({
  container: document.getElementById('dplayer'),
  video: {
      url: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4', // COS 视
  },
});
```

功能指引

播放不同格式的视频文件

1. 获取 COS 存储桶上的视频文件对象地址。

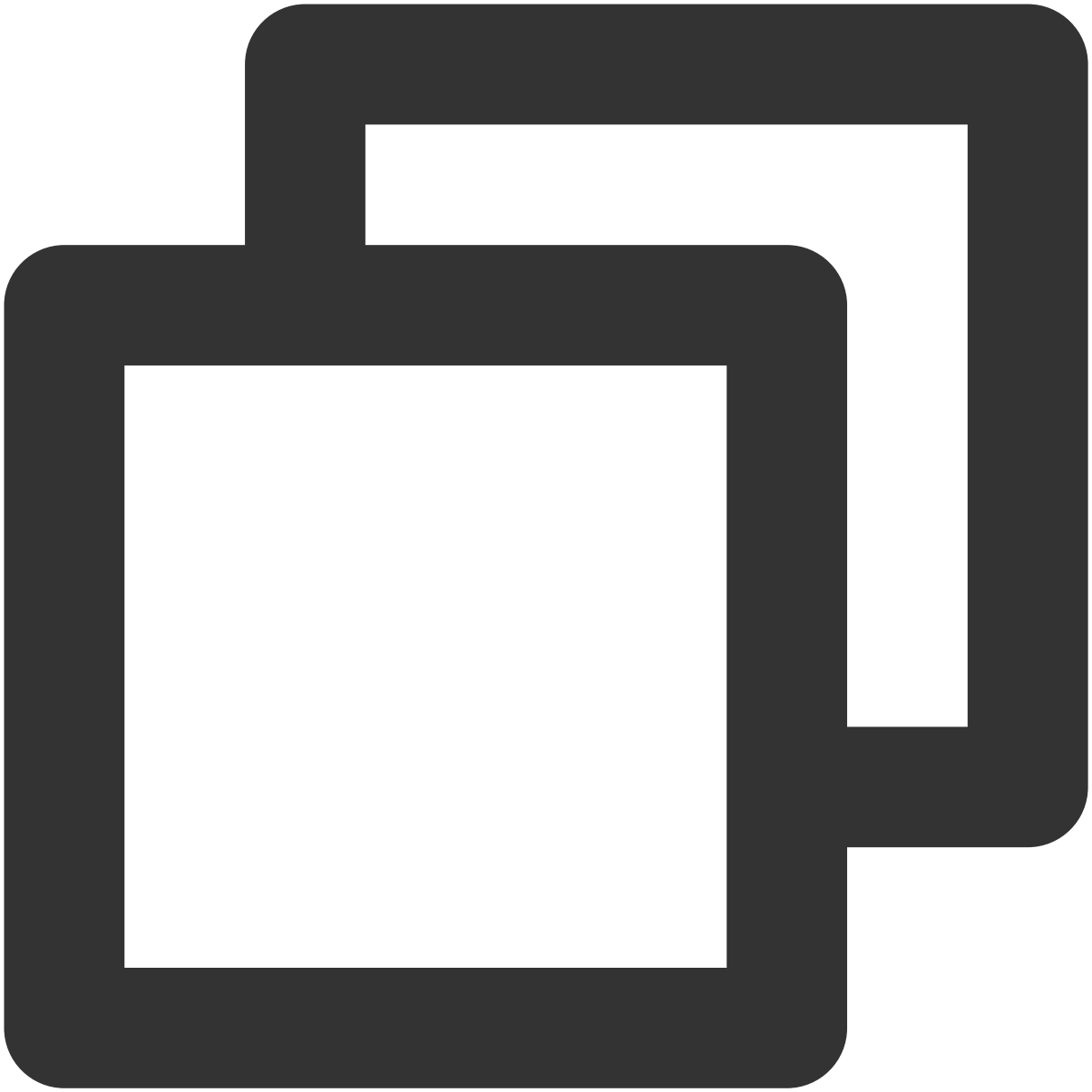
说明

未经转码的源视频在播放时有可能出现不兼容的情况，建议您使用转码后的视频进行播放，通过数据万象 音视频转码处理，获取不同格式视频文件。

2. 针对不同的视频格式，为了保证多浏览器的兼容性，需要引入对应的依赖。

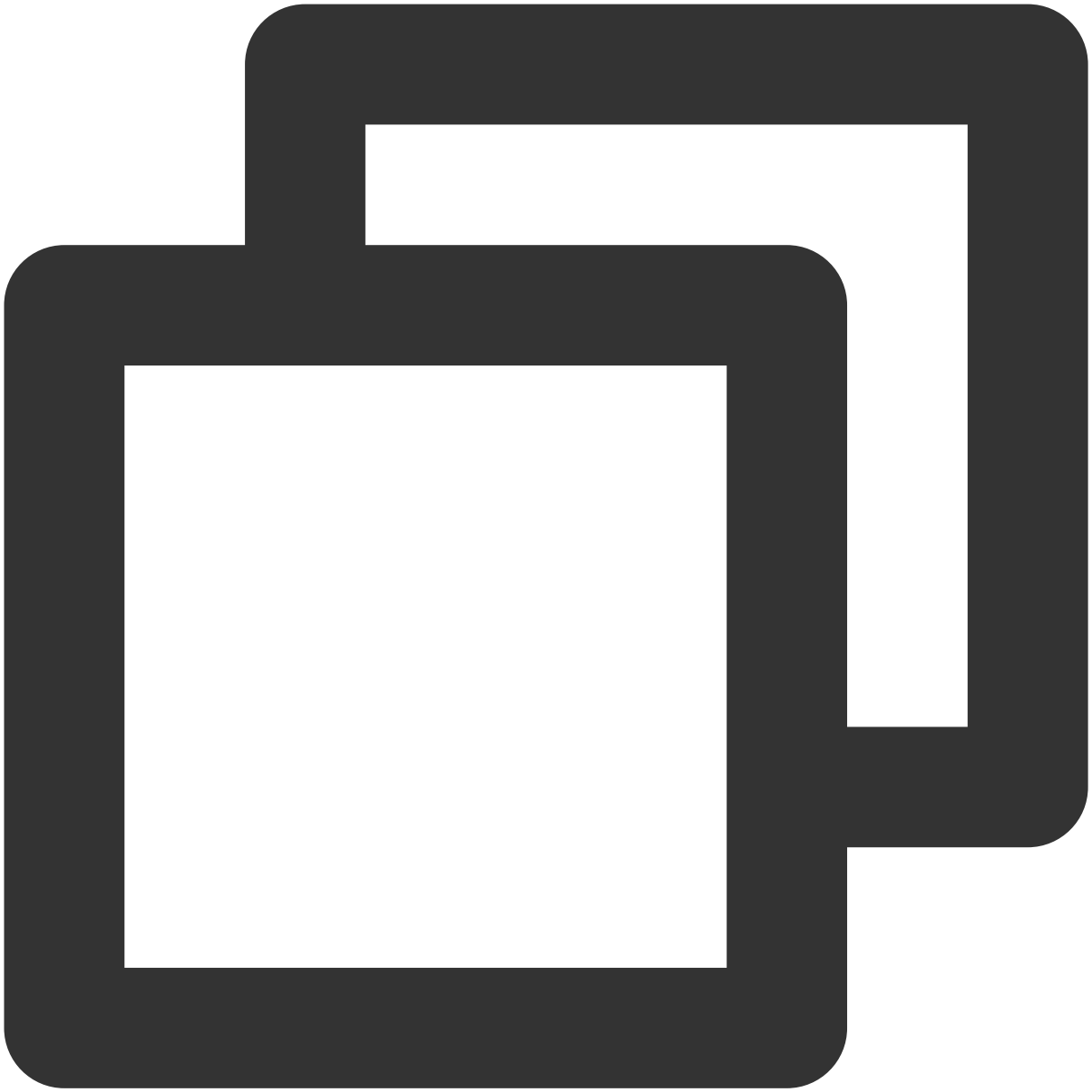
MP4：无需引入其他依赖。

HLS：如果需要在 Chrome 和 Firefox 等现代浏览器中通过 H5 播放 HLS 格式的视频，需要在 tcplayer.min.js 之前引入 hls.min.js。



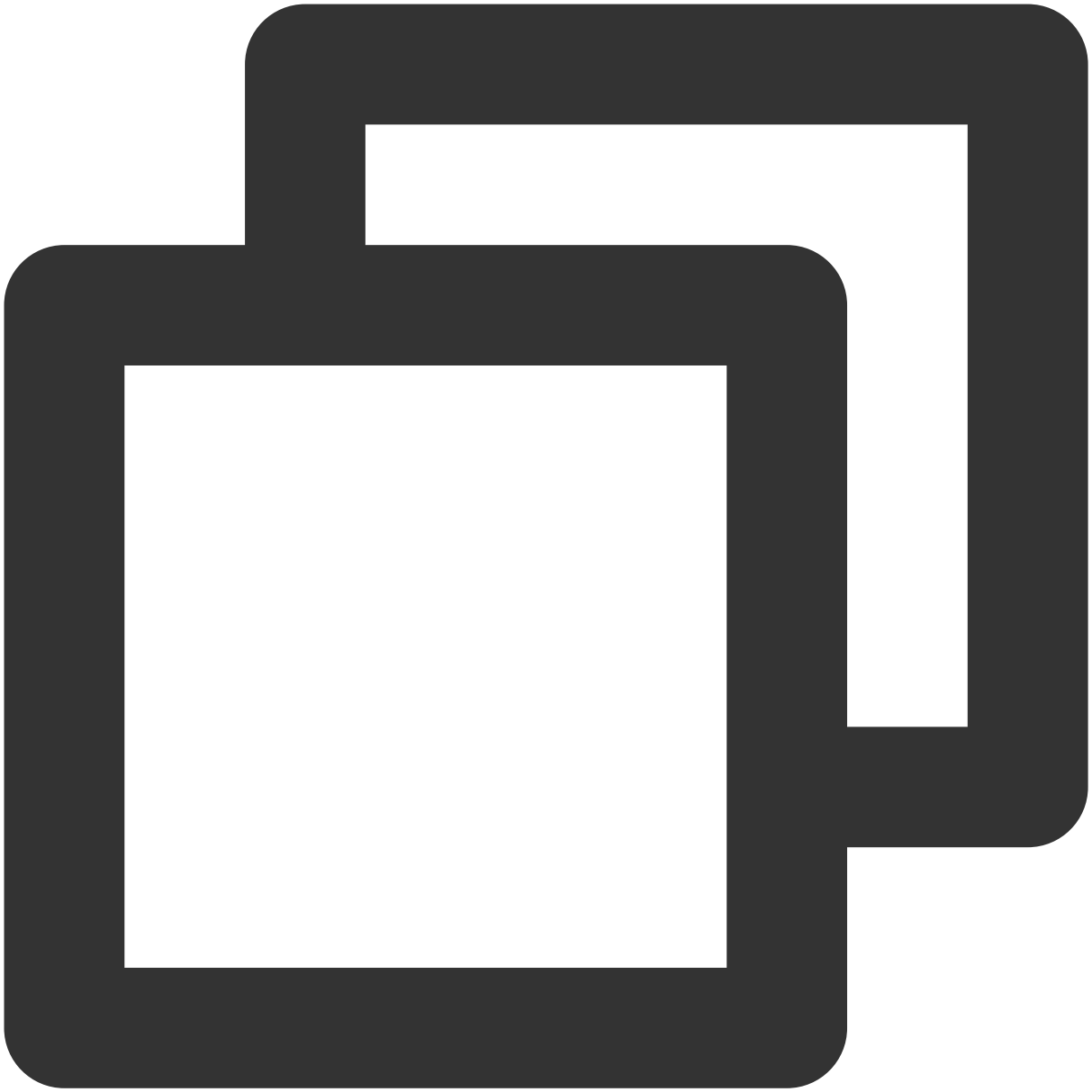
```
<script src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.2.1/libs/hls.m
```

FLV：如果需要在 Chrome 和 Firefox 等现代浏览器中通过 H5 播放 FLV 格式的视频，需要在 tcplayer.min.js 之前引入 flv.min.js。



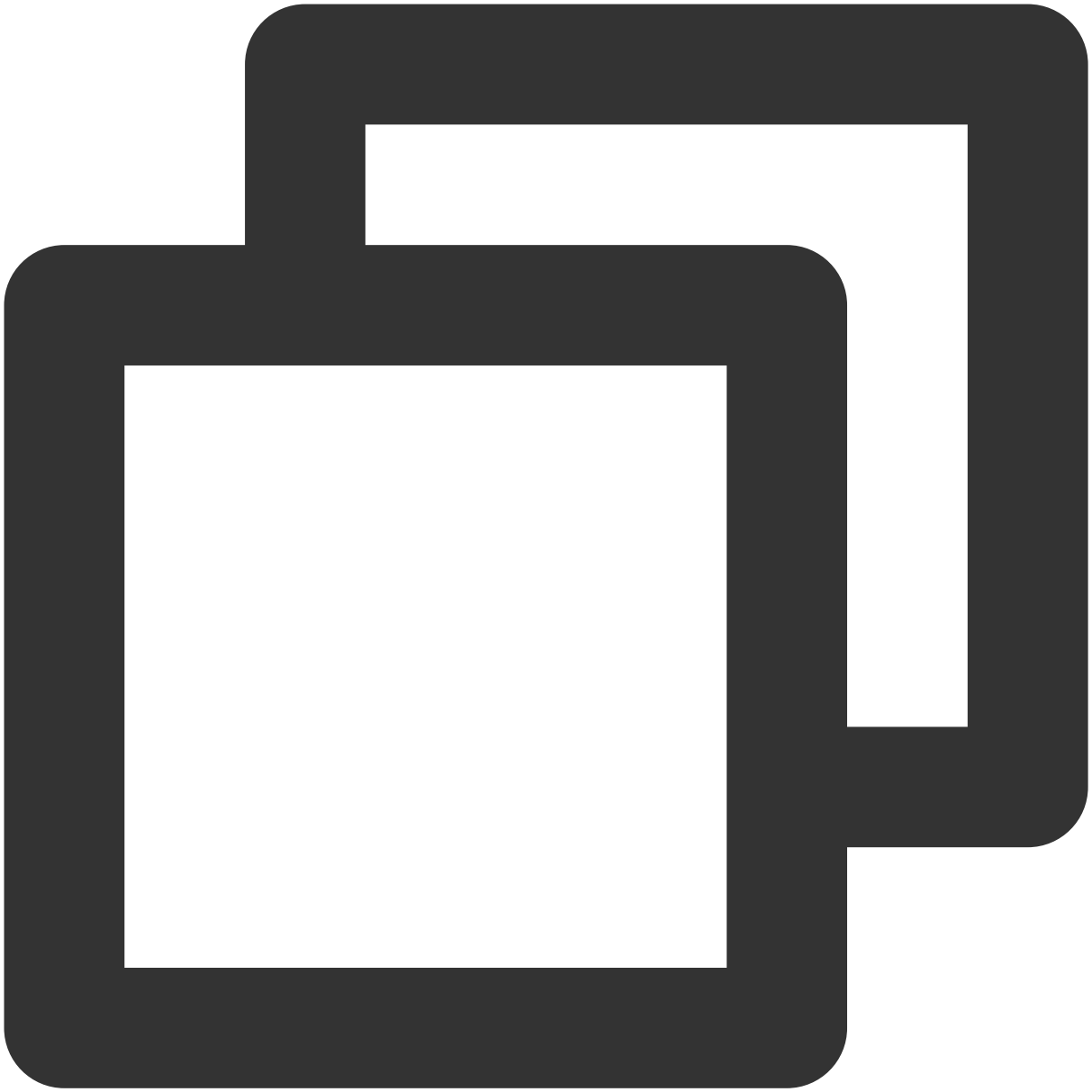
```
<script src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.2/libs/flv.m
```

DASH：DASH 视频需要加载 dash.all.min.js 文件。



```
<script src="https://cos-video-1258344699.cos.ap-guangzhou.myqcloud.com/lib/dash.
```

3. 初始化播放器并传入对象地址。



```
const dp = new DPlayer({
  container: document.getElementById('dplayer'),
  video: {
   url: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4', // COS 视频对
  },
});
```

获取示例代码：

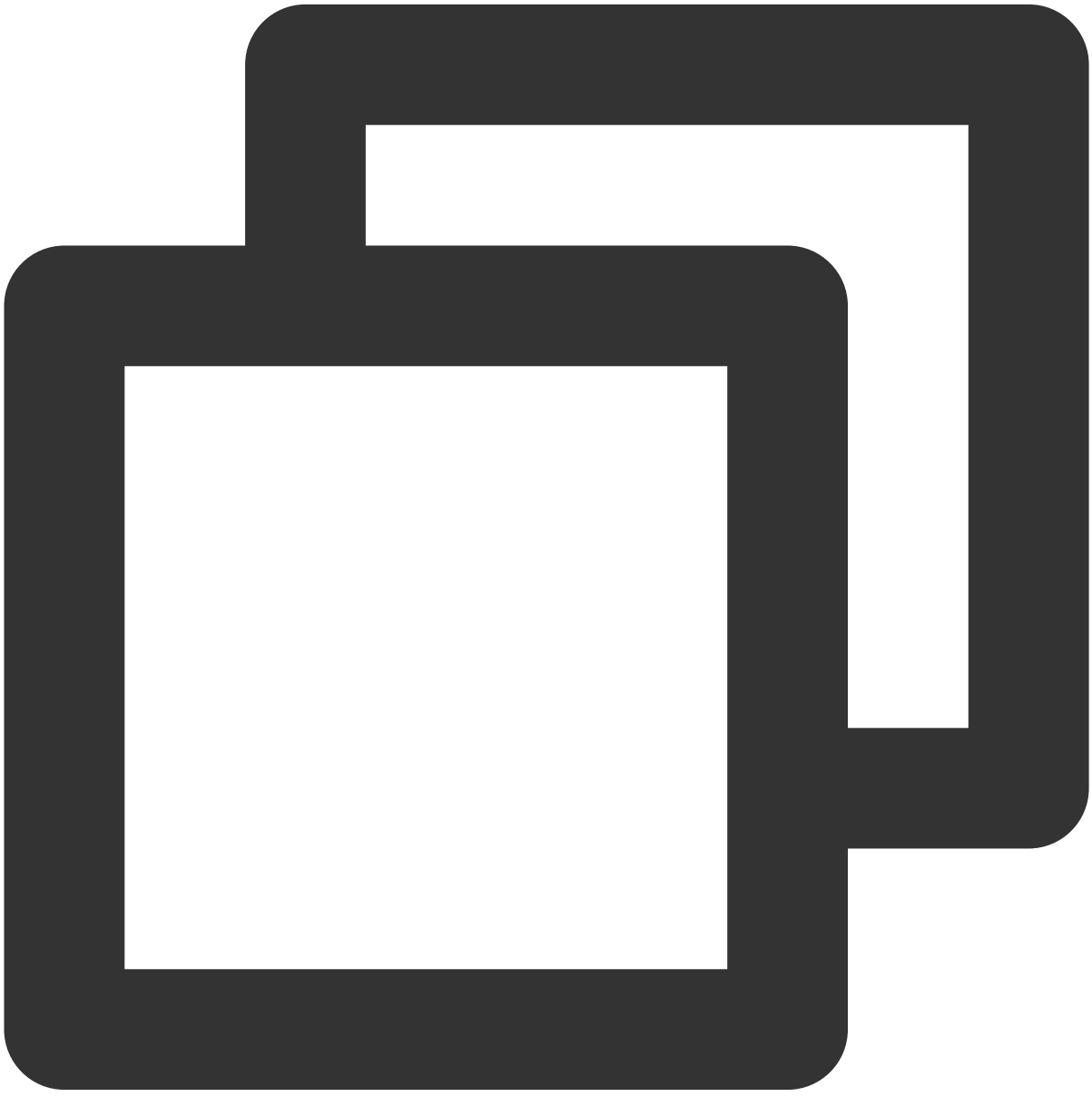
播放 MP4 示例代码

播放 FLV 示例代码

播放 HLS 示例代码
播放 DASH 示例代码

播放 PM3U8 视频

PM3U8 是指私有的 M3U8 视频文件，COS 提供用于获取私有 M3U8 TS 资源的下载授权 API，可参见 私有 M3U8 接口。



```
const dp = new DPlayer({
  container: document.getElementById('dplayer'),
```

```
    // 关于 pm3u8 详情请查看相关文档：https://www.tencentcloud.com/document/product/436/
    video: {
      url: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8?ci-process
    }
  });
```

获取示例代码：

播放 PM3U8 示例代码

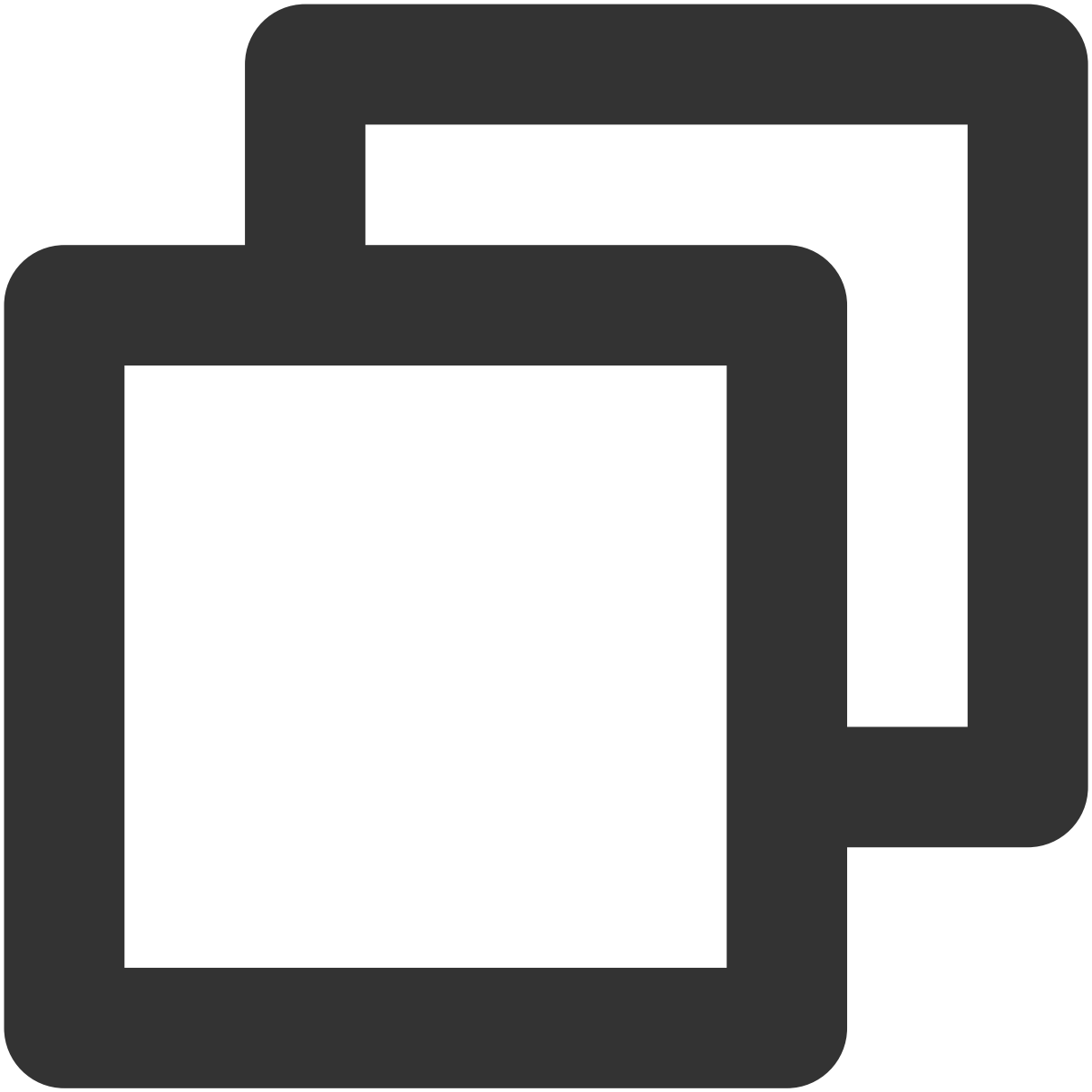
设置封面图

1. 获取 COS 存储桶上的封面图对象地址。

注意

通过数据万象 智能封面 能力，提取最优帧生成截图作为封面，可提升内容吸引力。

2. 初始化播放器并设置封面图。



```
const dp = new DPlayer({
  container: document.getElementById('dplayer'),
  video: {
 url: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4',
 pic: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.png',
  },
});
```

获取示例代码：

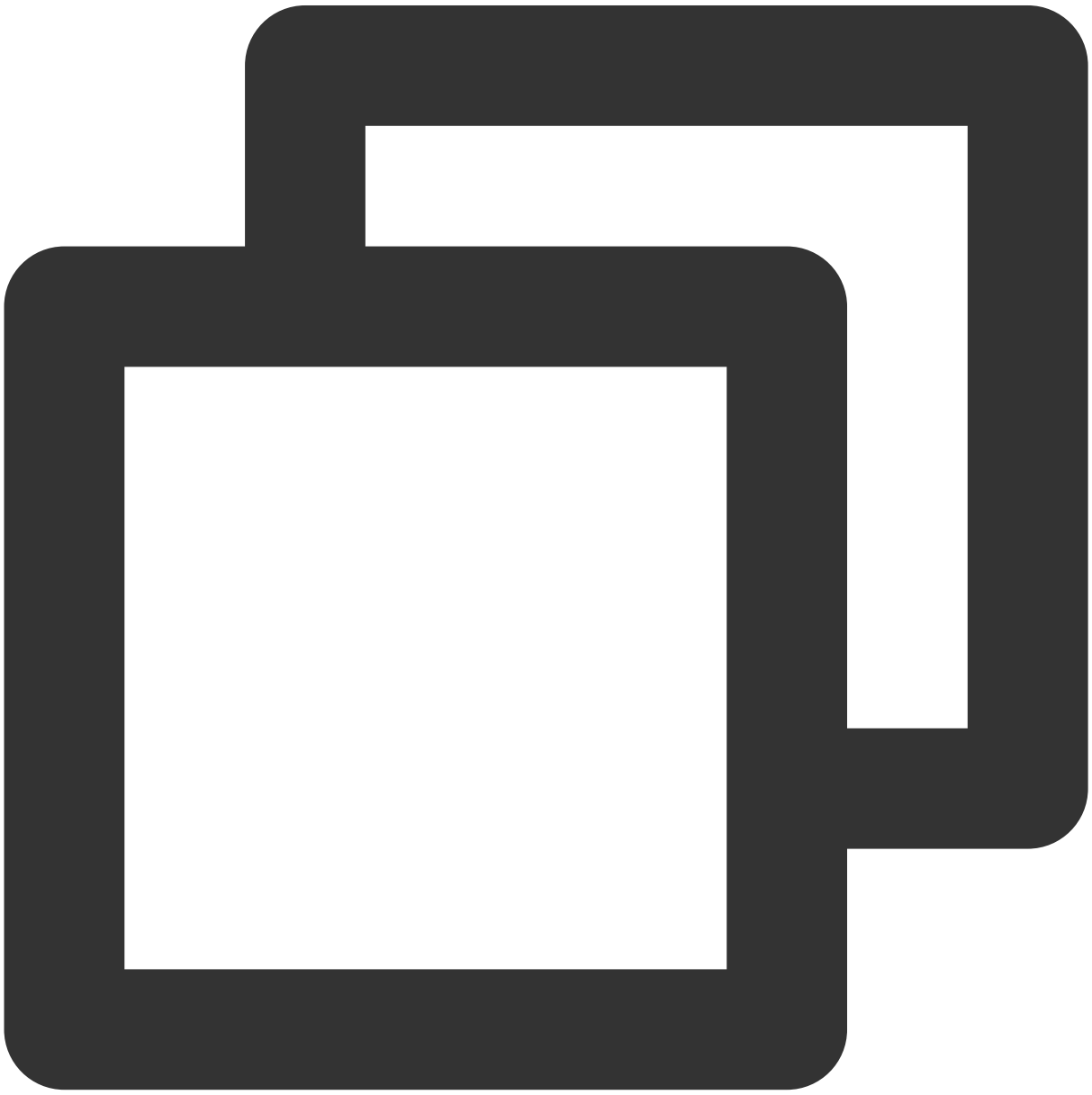
设置封面图示例代码

播放 HLS 加密视频

为了保障视频内容安全，防止视频被非法下载和传播，数据万象提供了对 HLS 视频内容进行加密的功能，拥有相比于私有读文件更高的安全级别。加密后的视频，无法分发给无访问权限的用户观看。即使视频被下载到本地，视频本身也是被加密的，无法恶意二次分发，从而保障您的视频版权不受到非法侵犯。

操作步骤如下：

1. 参见 播放 HLS 加密视频 和 COS 音视频实践 | 给你的视频加把锁 流程，生成加密视频。
2. 初始化播放器并传入视频对象地址。



```
 const dp = new DPlayer({
container: document.getElementById('dplayer'),
video: {
  url: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8' // 加密视频地址
}
 });
```

获取示例代码：

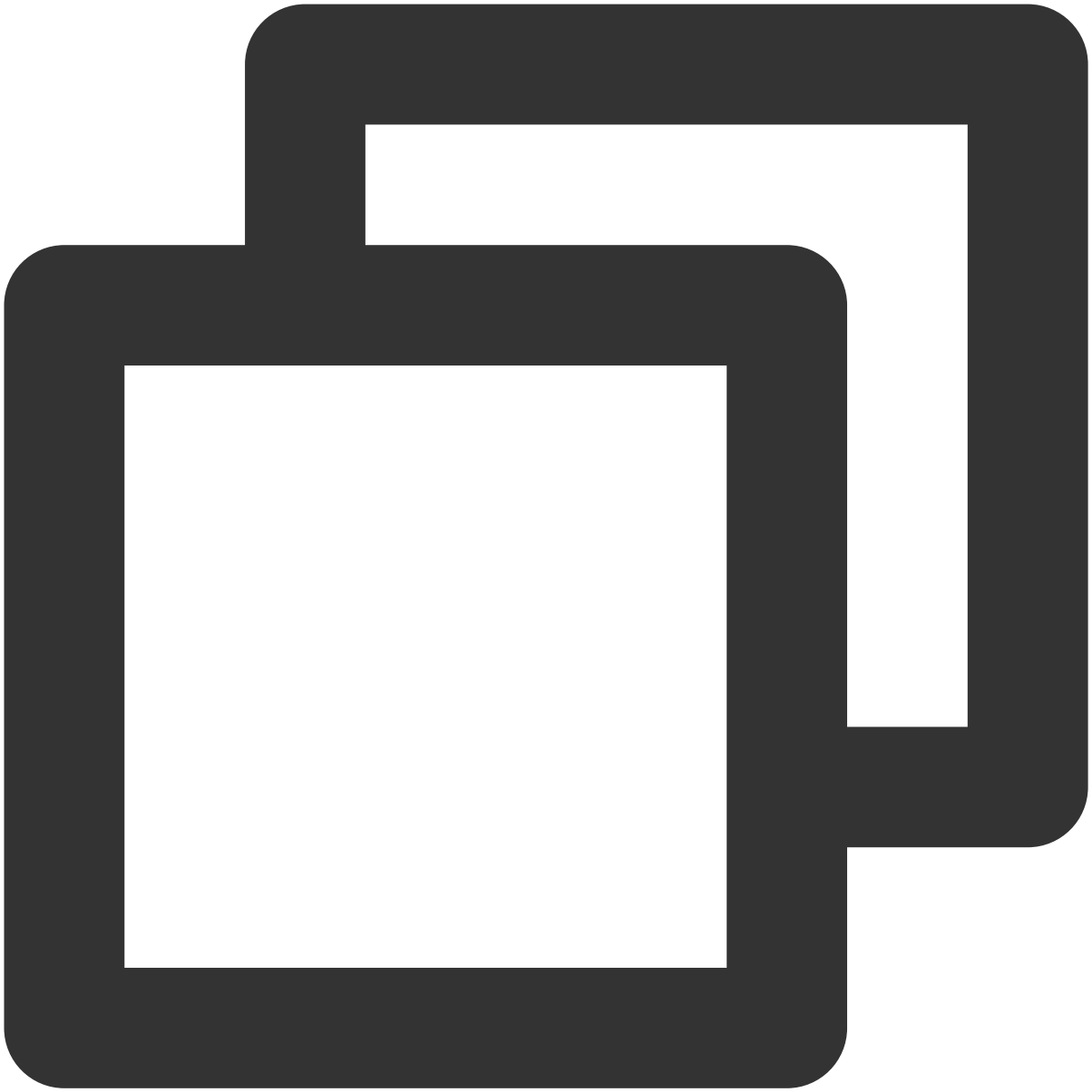
播放 HLS 加密视频示例代码

切换多清晰度

数据万象 自适应码流 功能，可以将视频文件转码并打包生成自适应码流输出文件，帮助用户在不同网络情况下快速分发视频内容，播放器能够根据当前带宽，动态选择最合适的码率播放，详情可参见 COS 音视频实践 | 数据工作流助你播放多清晰度视频。

操作步骤如下：

1. 通过 数据万象 自适应码流 功能，生成多码率自适应的 HLS 或 DASH 目标文件。
2. 初始化播放器并传入视频对象地址。



```
const dp = new DPlayer({
  container: document.getElementById('dplayer'),
  video: {
   url: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8', //  多码率的
  },
});
```

获取示例代码：

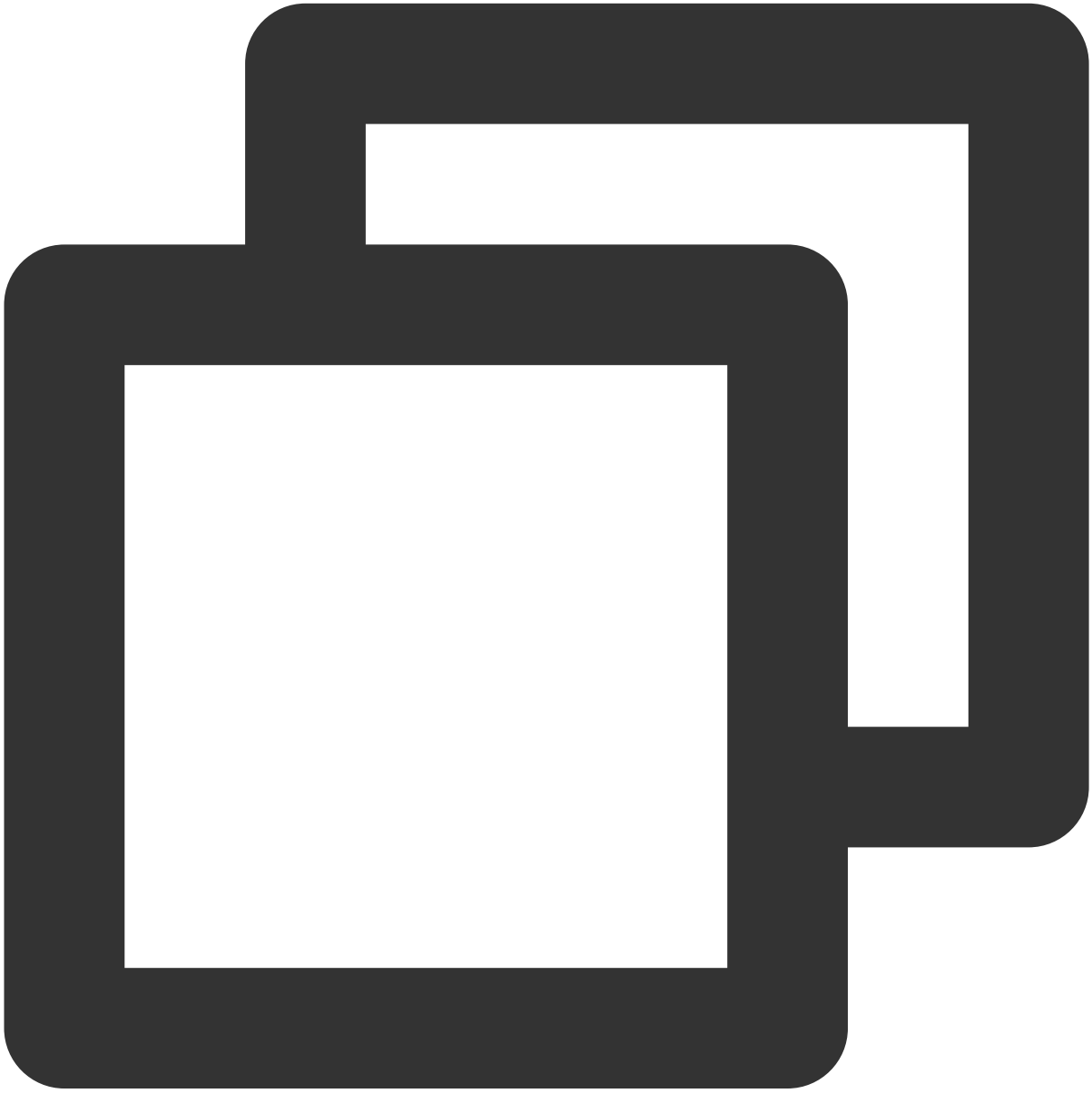
切换清晰度示例代码

设置左上角 LOGO

播放器支持在左上角设置 LOGO。

操作步骤如下：

1. 获取 COS 存储桶上的 LOGO 图标对象地址。
2. 初始化播放器并设置 LOGO 图标。



```
const dp = new DPlayer({
 container: document.getElementById('dplayer'),
  video: {
   url: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4',
```

```
    },
   logo: 'https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.svg'
  });
```

获取示例代码：

设置左上角 LOGO 示例代码

使用 VideojsPlayer 播放 COS 视频文件

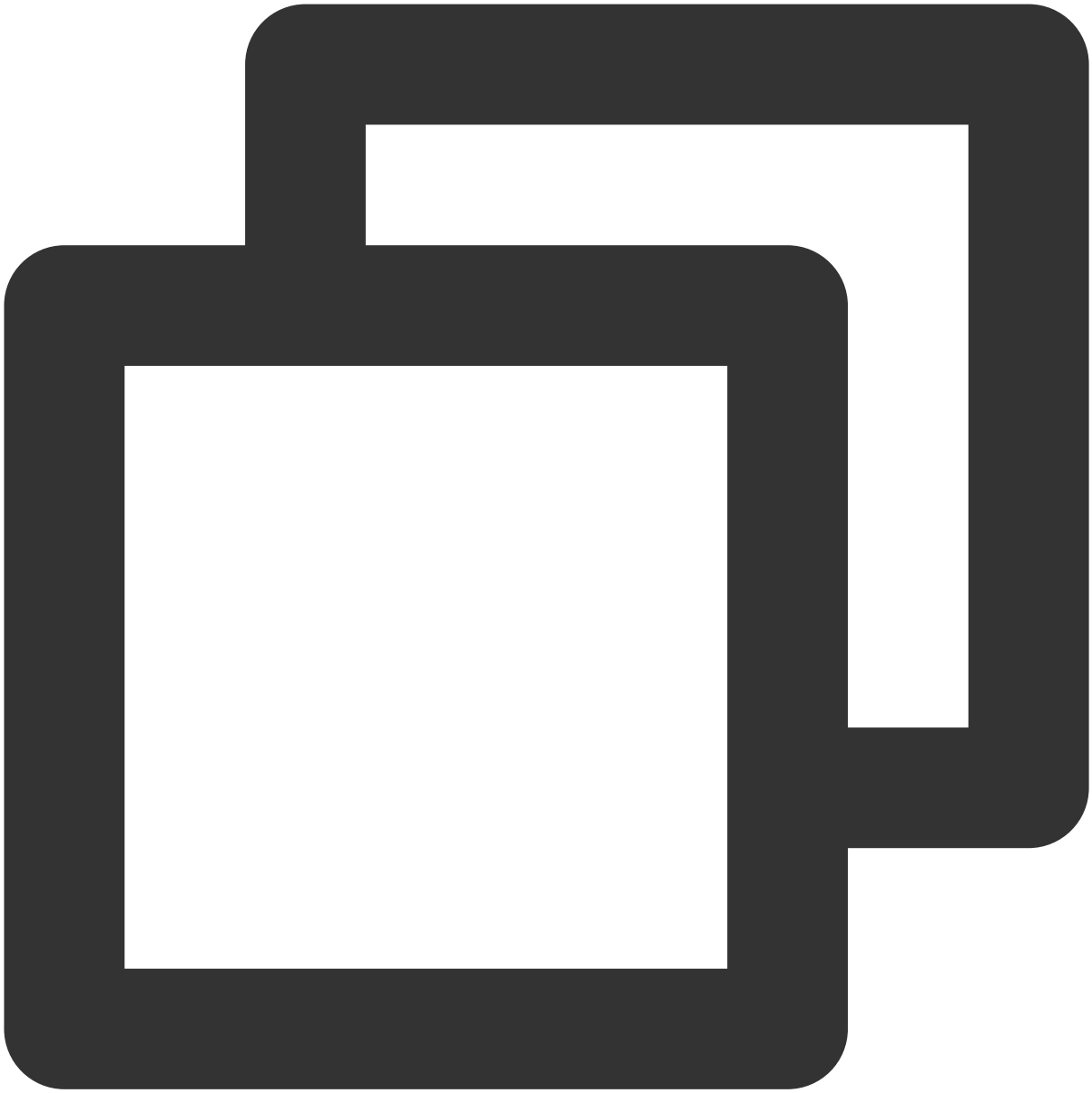
最近更新时间：2024-01-06 10:54:03

简介

本文将介绍如何使用 VideojsPlayer 并结合 腾讯云数据万象(CI) 所提供的丰富的音视频能力，实现在 Web 浏览器播放 COS 视频文件。

集成指引

步骤1：在页面中引入播放器样式文件及脚本文件



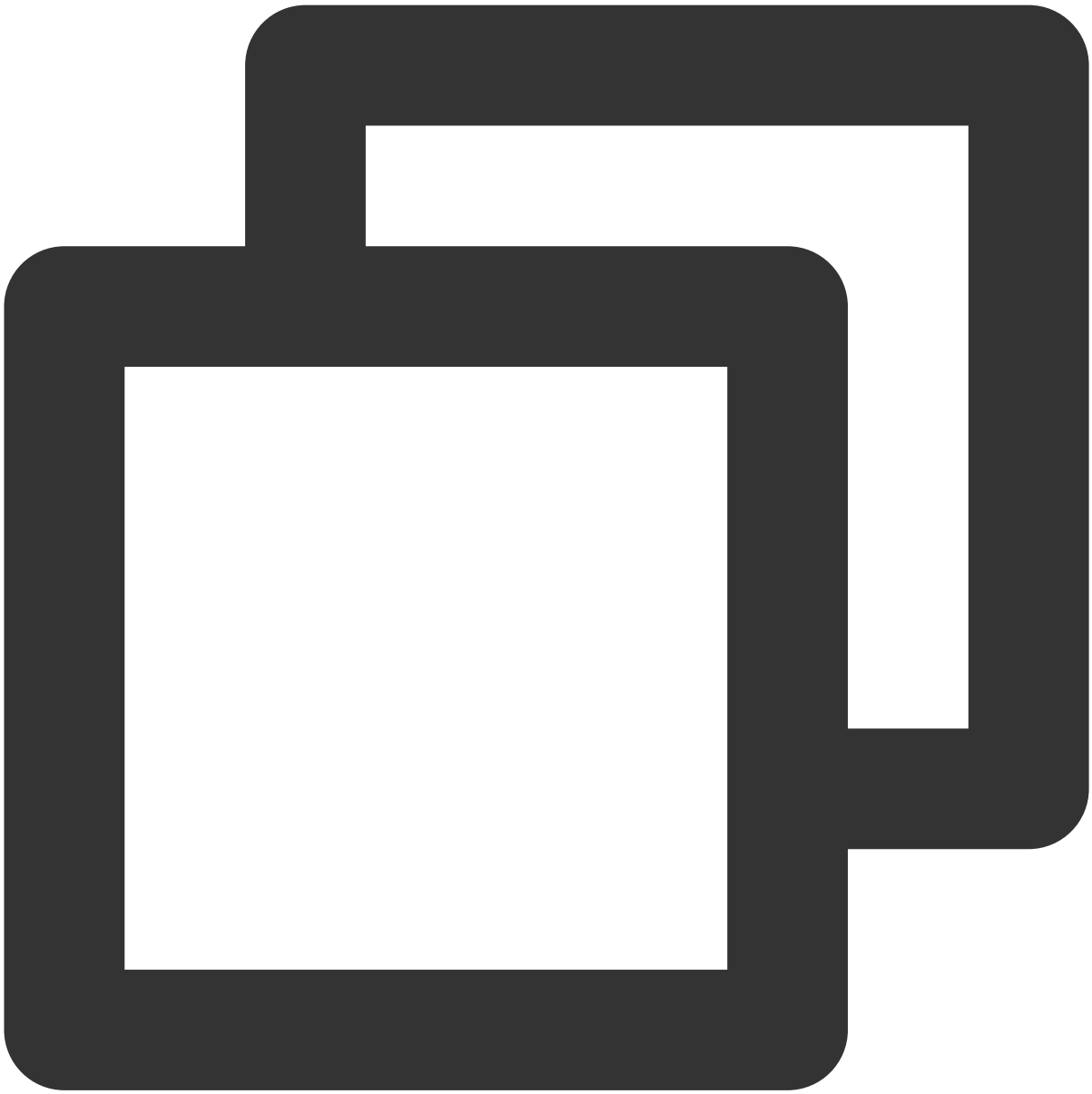
```
<!-- 播放器样式文件 -->
<link href="https://vjs.zencdn.net/7.19.2/video-js.css" rel="stylesheet" />
<!-- 播放器脚本文件 -->
<script src="https://vjs.zencdn.net/7.19.2/video.min.js"></script>
```

说明

建议在正式使用播放器时，自行部署以上相关静态资源。

步骤2：设置播放器容器节点

在需要展示播放器的页面位置加入播放器容器。例如，在 index.html 中加入如下代码（容器 ID 以及宽高都可以自定义）。



```
<video
  id="my-video"
  class="video-js"
  controls
  preload="auto"
  width="100%"
  height="100%"
  data-setup="{}"
```

```
></video>
```

步骤3：获取视频文件对象地址

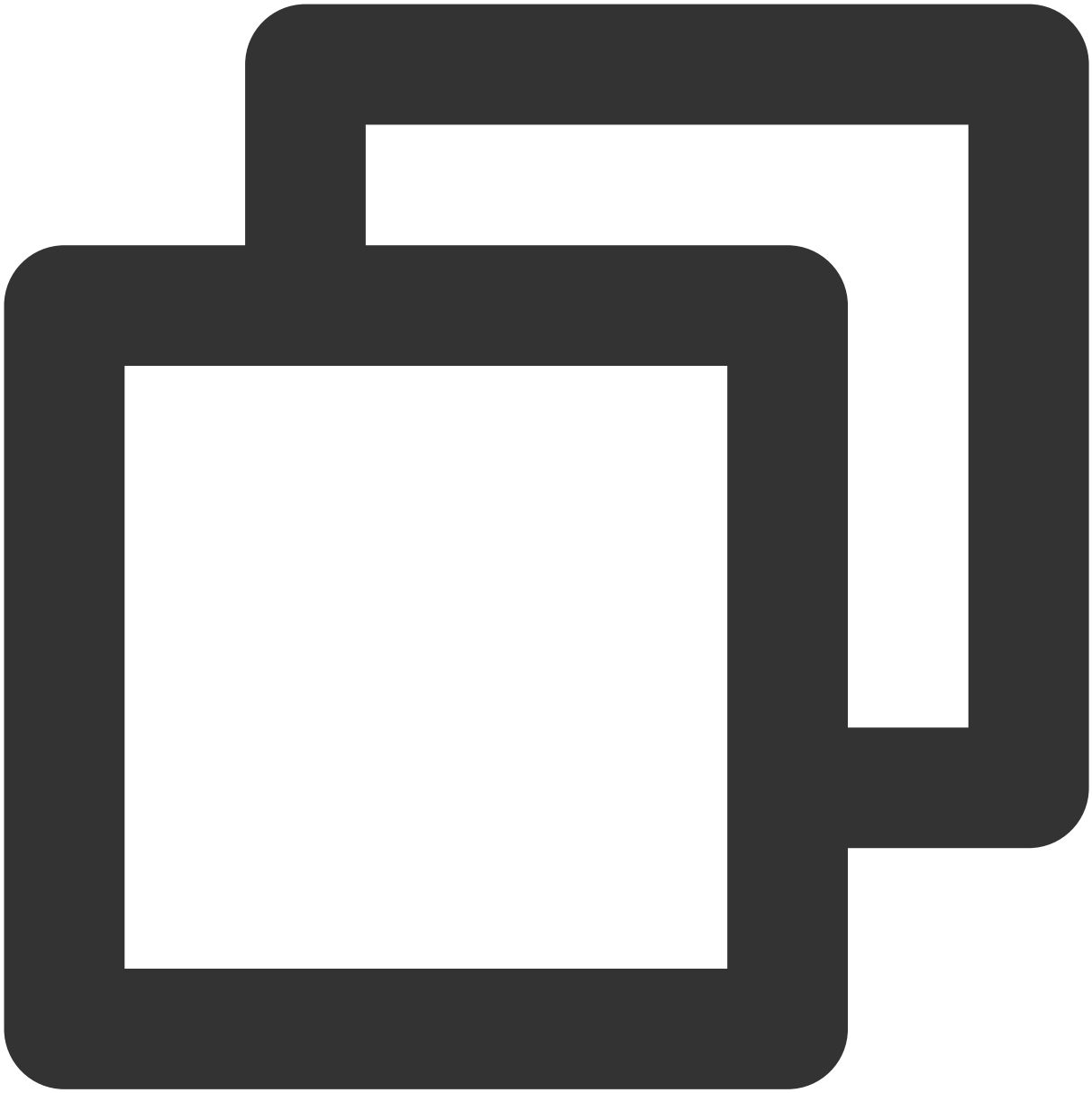
1. 创建一个存储桶。
2. 上传视频文件。
3. 获取视频文件对象地址，格式为 `https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.<视频格式>` 。

说明

若存在跨域问题，则需要进行存储桶跨域访问 CORS 设置，详情请参见 设置跨域访问。

若存储桶为私有读写，则对象地址需要携带签名，详情请参见 请求签名。

步骤4：在播放器容器内设置视频地址，传入 COS 视频文件对象地址 URL



```
<video
  id="my-video"
  class="video-js"
  controls
  preload="auto"
  width="100%"
  height="100%"
  data-setup="{}"
>
  <source
    src="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4"
```

```
    type="video/mp4"
  />
</video>
```

功能指引

播放不同格式的视频文件

1. 获取 COS 存储桶上的视频文件对象地址。

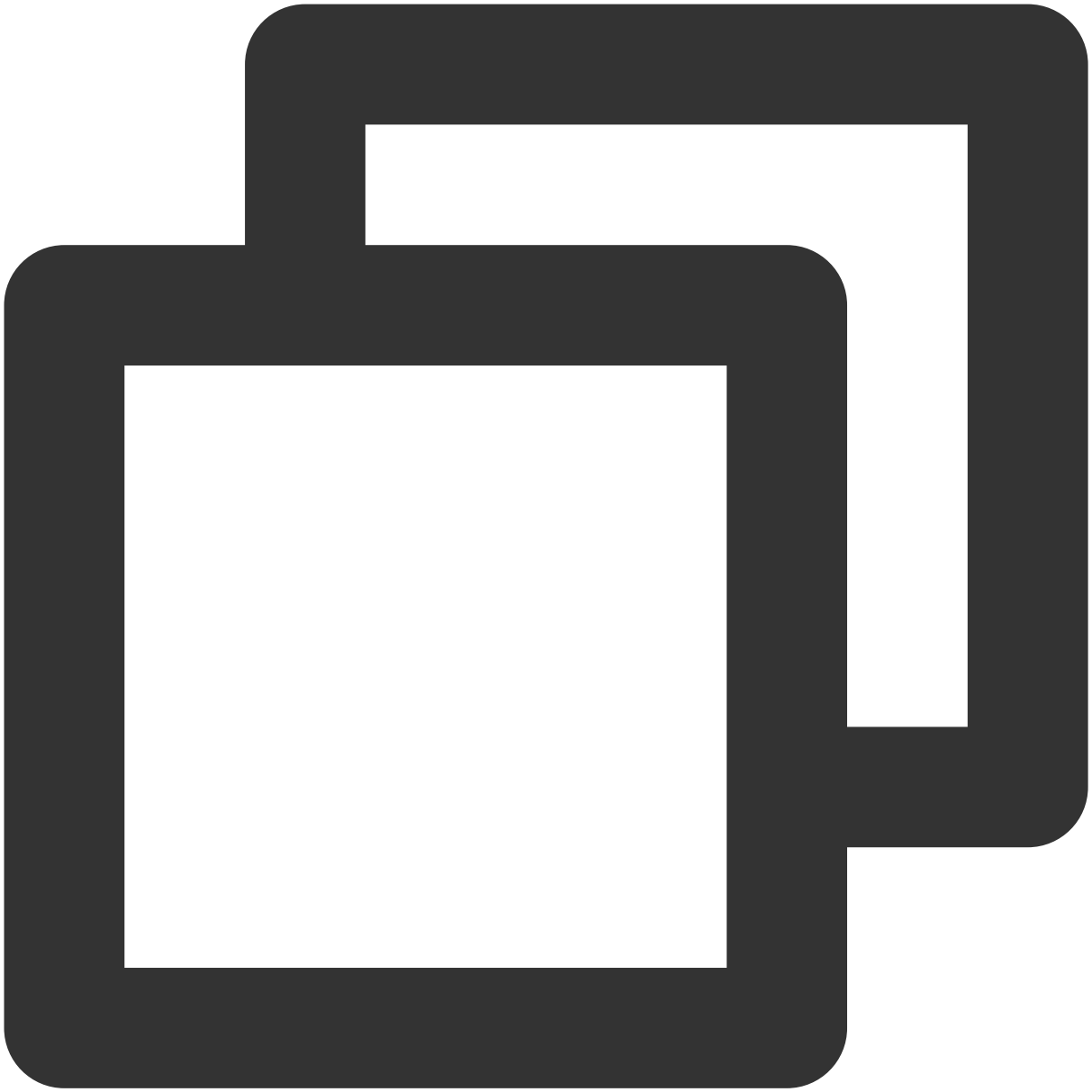
说明

未经转码的源视频在播放时有可能出现不兼容的情况，建议您使用转码后的视频进行播放，通过数据万象 音视频转码处理，获取不同格式视频文件。

2. 针对不同的视频格式，为了保证多浏览器的兼容性，需要引入对应的依赖。

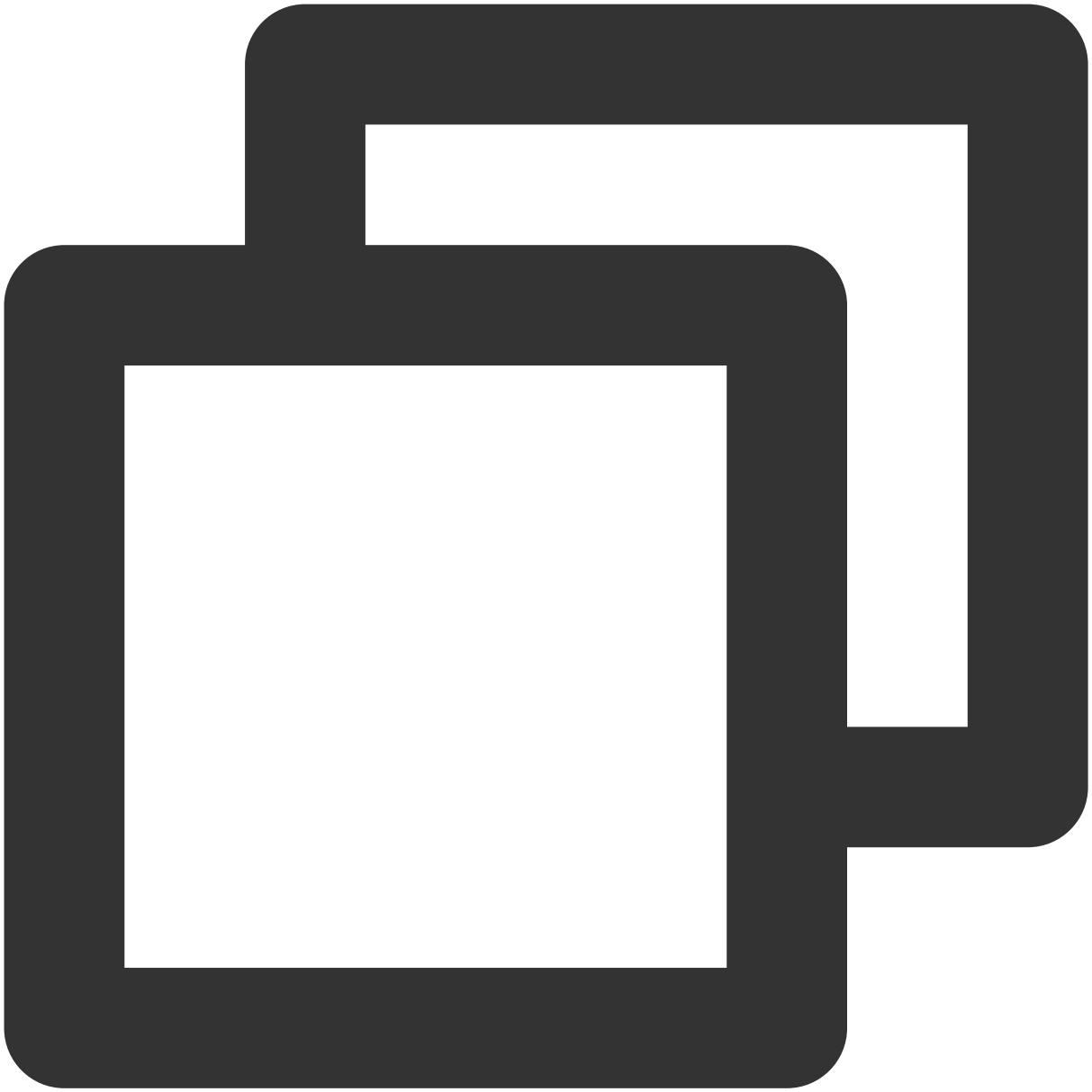
MP4：无需引入其他依赖。

HLS：如果需要在 Chrome 和 Firefox 等现代浏览器中通过 H5 播放 HLS 格式的视频，需要在 tcplayer.min.js 之前引入 hls.min.js。



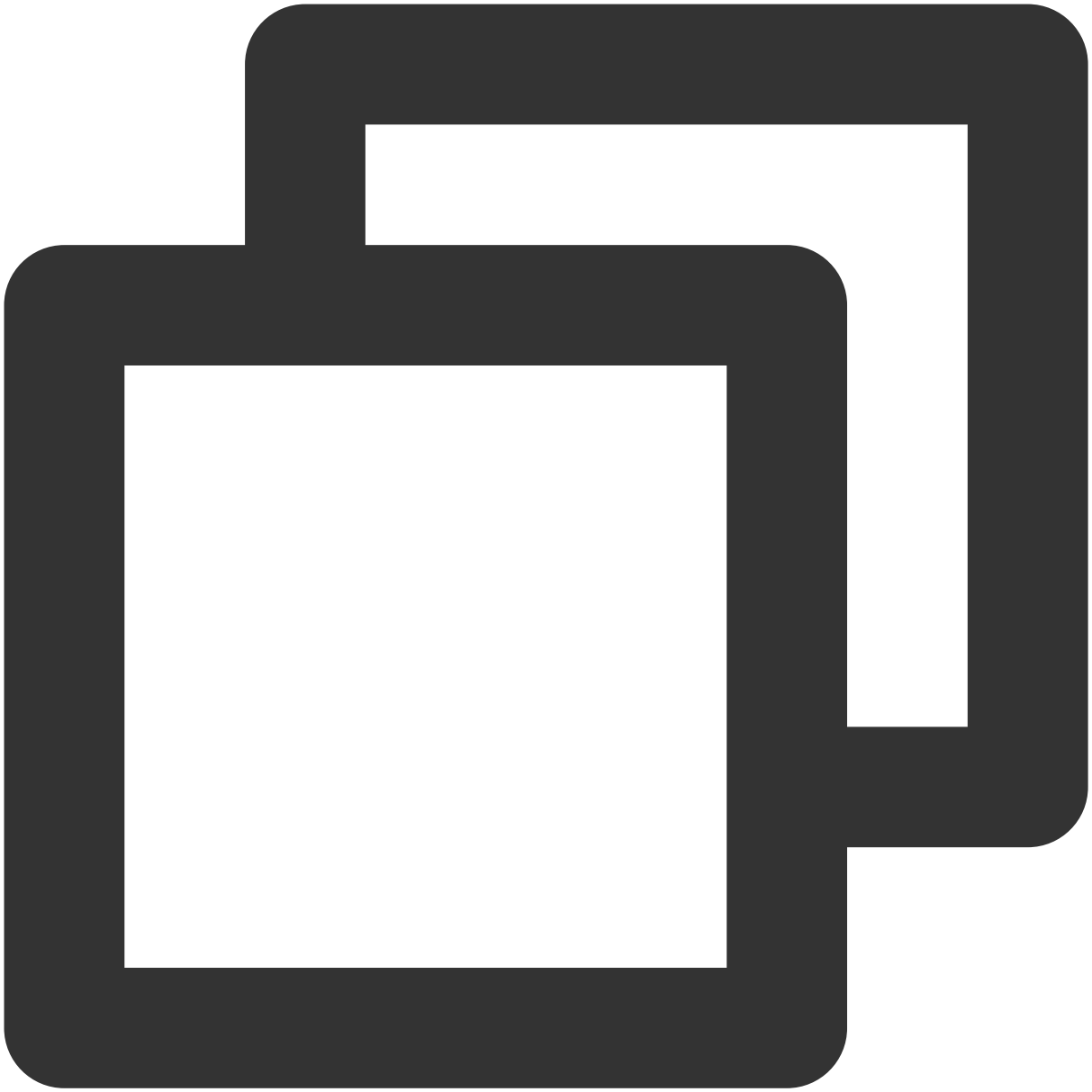
```
<script src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.2.1/libs/hls.m
```

FLV：如果需要在 Chrome 和 Firefox 等现代浏览器中通过 H5 播放 FLV 格式的视频，需要在 tcplayer.min.js 之前引入 flv.min.js。



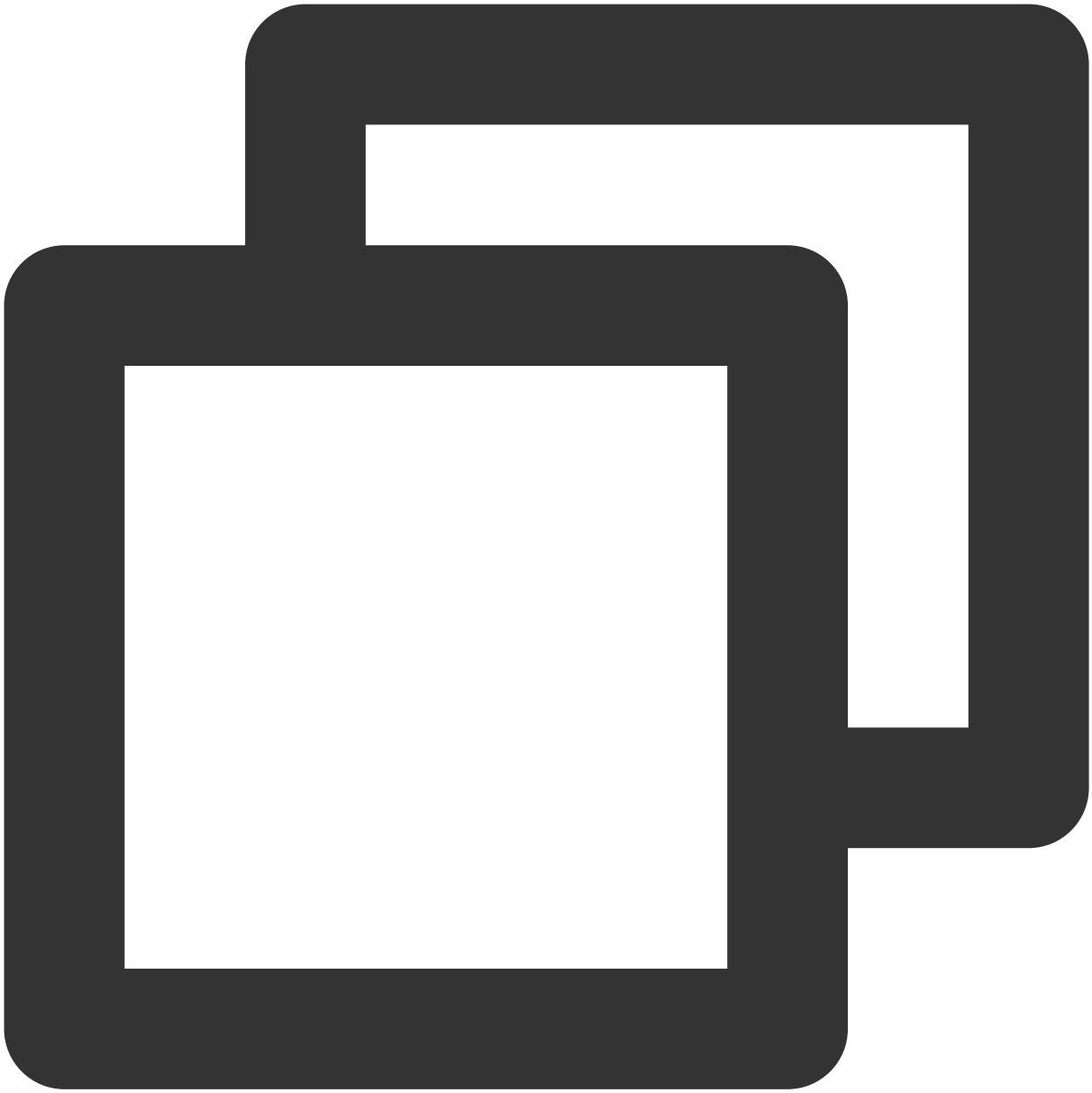
```
<script src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.2/libs/flv.m
```

DASH：DASH 视频需要加载 dash.all.min.js 文件。



```
<script src="https://cos-video-1258344699.cos.ap-guangzhou.myqcloud.com/lib/dash.
```

3. 初始化播放器并传入对象地址。



```
<!-- MP4 -->
<source
  src="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4"
  type="video/mp4"
/>

<!-- HLS -->
<source
  src="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8"
  type="application/x-mpegURL"
/>
```

```
<!-- FLV -->
<source
  src="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.flv"
  type="video/x-flv"
/>

<!-- DASH -->
<source
  src="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mpd"
  type="application/dash+xml"
/>
```

获取示例代码：

播放 MP4 示例代码

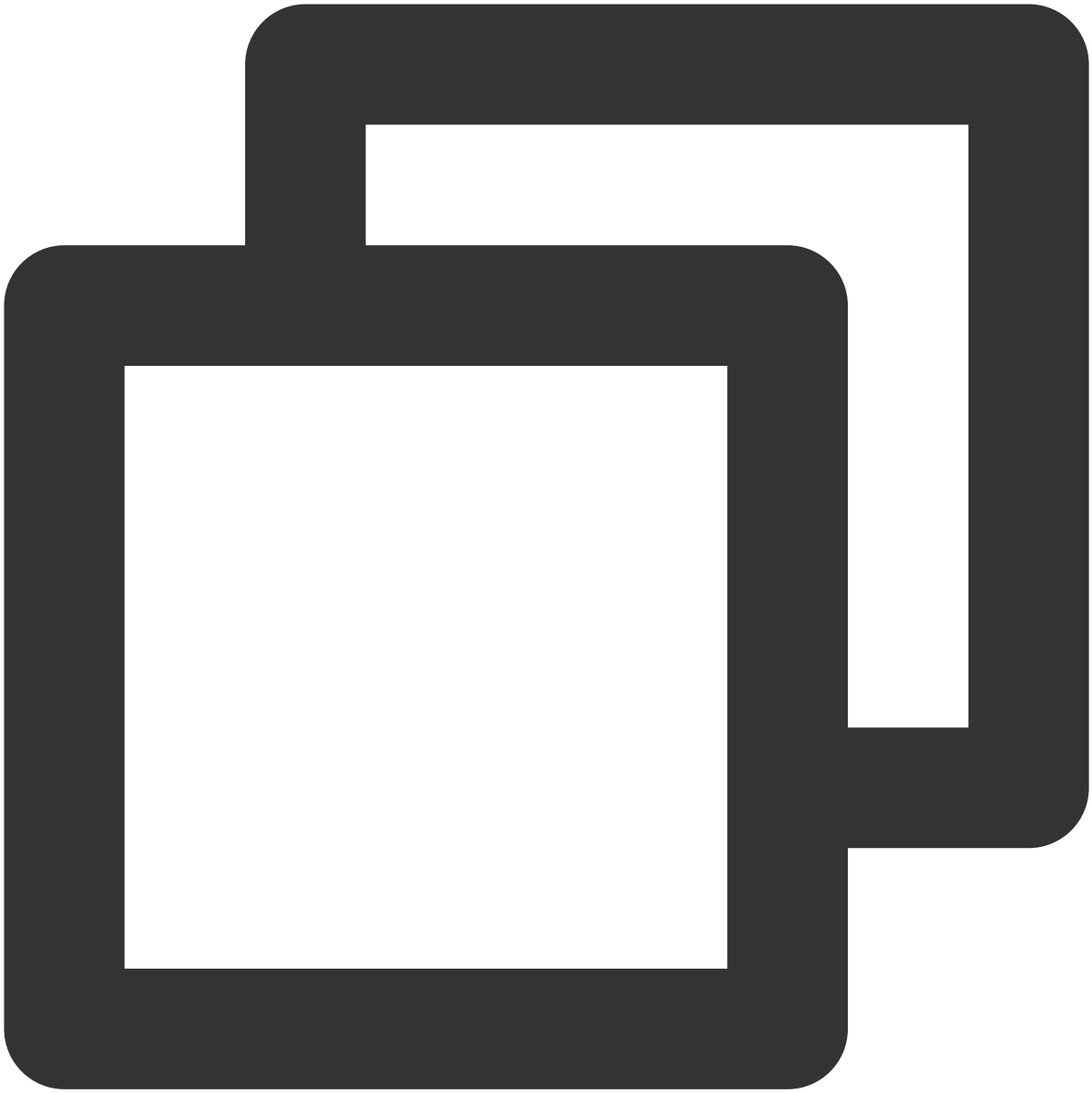
播放 FLV 示例代码

播放 HLS 示例代码

播放 DASH 示例代码

播放 PM3U8 视频

PM3U8 是指私有的 M3U8 视频文件，COS 提供用于获取私有 M3U8 TS 资源的下载授权API，可参见 私有 M3U8 接口。



```
<source
  src="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8?ci-process=pm3
  type="application/x-mpegURL"
/>
```

获取示例代码：

播放 PM3U8 示例代码

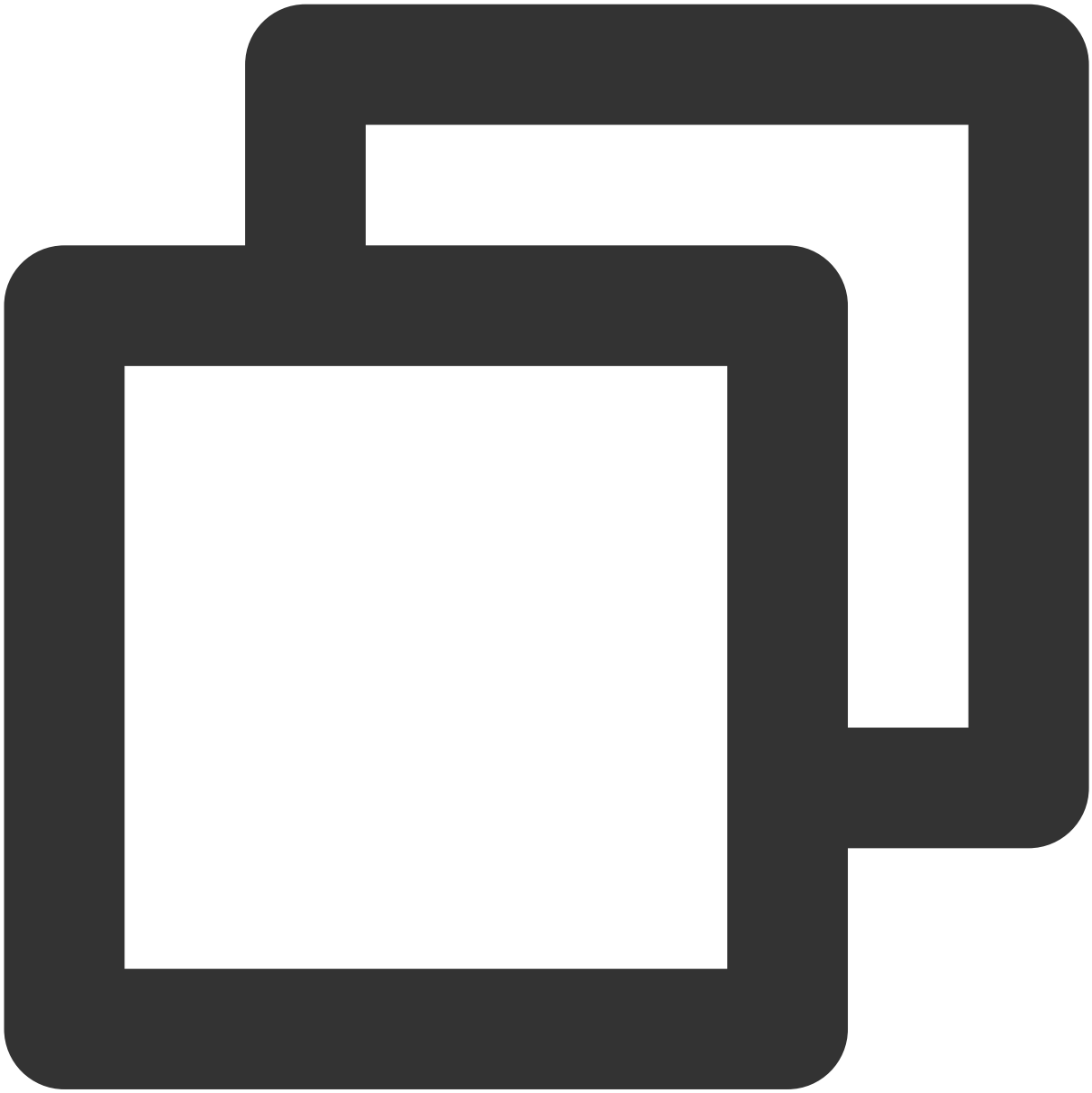
设置封面图

1. 获取 COS 存储桶上的封面图对象地址。

注意

通过数据万象 智能封面 能力，提取最优帧生成截图作为封面，可提升内容吸引力。

2. 初始化播放器并设置封面图。



```
<video
  id="my-video"
  class="video-js"
  controls
  preload="auto"
  width="100%"
```

```
    height="100%"
    data-setup="{}"
    poster="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/poster.png"
  >
    <source
      src="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.mp4"
      type="video/mp4"
    />
  </video>
```

获取示例代码：

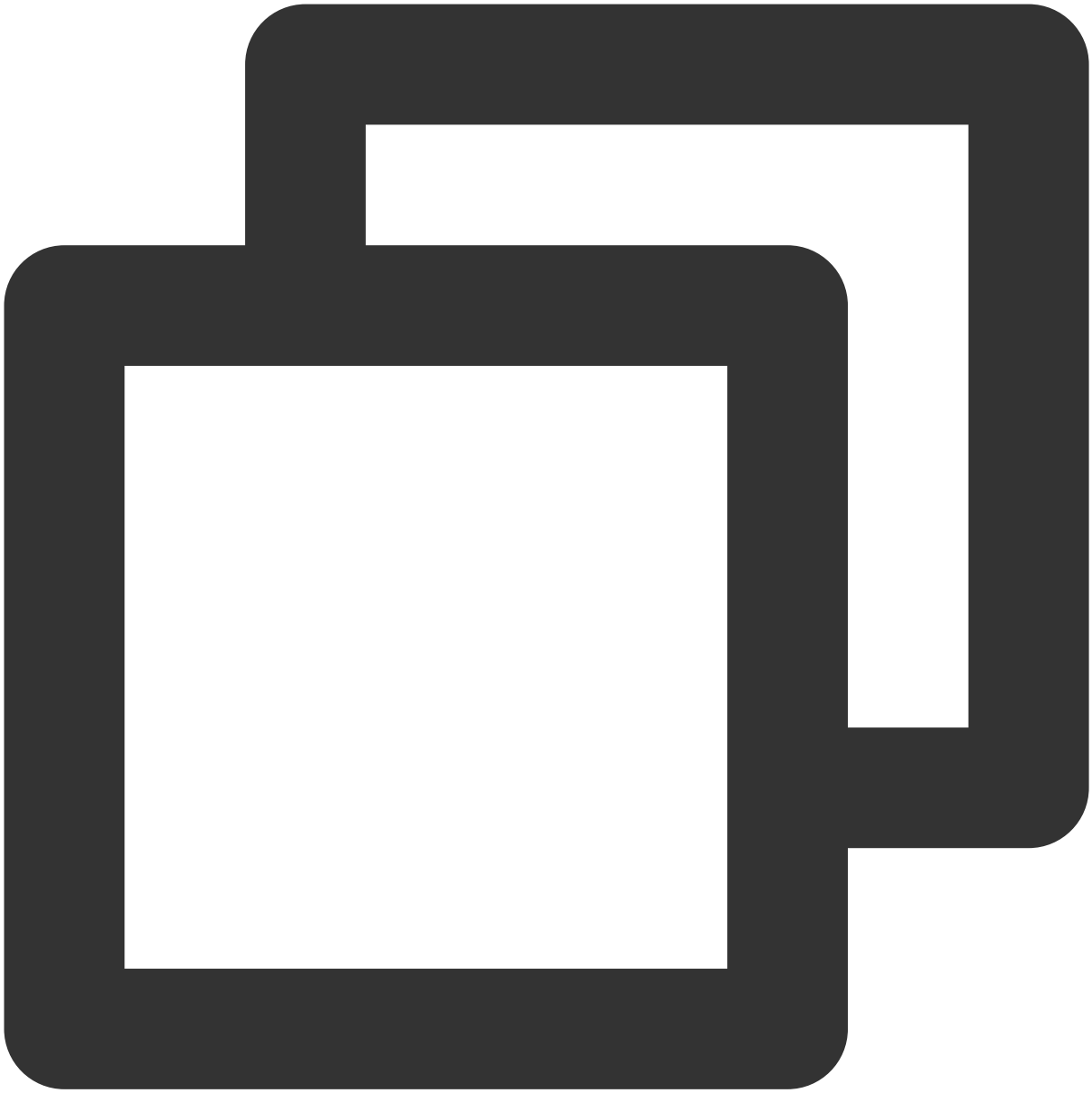
设置封面图示例代码

播放 HLS 加密视频

为了保障视频内容安全，防止视频被非法下载和传播，数据万象提供了对 HLS 视频内容进行加密的功能，拥有相比于私有读文件更高的安全级别。加密后的视频，无法分发给无访问权限的用户观看。即使视频被下载到本地，视频本身也是被加密的，无法恶意二次分发，从而保障您的视频版权不受到非法侵犯。

操作步骤如下：

1. 参见 播放 HLS 加密视频 和 COS 音视频实践 | 给你的视频加把锁 流程，生成加密视频。
2. 初始化播放器并传入视频对象地址。



```
<source
  src="https://<BucketName-APPID>.cos.<Region>.myqcloud.com/xxx.m3u8"
  type="application/x-mpegURL"
/>
```

获取示例代码：

播放 HLS 加密视频示例代码

工作流实践
使用自定义函数管理 COS 文件

最近更新时间：2024-01-06 10:54:03

概述

对象存储（Cloud Object Storage，COS）工作流提供了一系列针对音视频、图片等媒体文件的处理能力，用户可以按照自身需求，灵活快速地搭建媒体处理流水线。随着越来越多用户接入 COS 工作流，一些定制化需求也被提上议程。为了保障灵活性，COS 工作流推出了自定义云函数功能，支持用户在工作流中配置云函数节点，在云函数中实现定制化逻辑。
为了进一步降低用户的使用门槛，COS 工作流整理了常用的云函数功能模板，并将其创建流程集成至节点的配置步骤中，方便用户对处理前的源文件、处理后的产物文件进行后续加工。现已支持修改对象属性、移动对象，删除对象等基本操作。

应用场景

在媒体处理后，对源文件进行移动、沉降等处理，降低存储成本。
在媒体处理后，对产物文件进行标签设置，修改头部等处理，方便业务使用。

方案优势

开箱即用：无需开发函数逻辑，无需关注复杂的部署流程，简易配置即可使用。
配置灵活：可以按需配置常用功能节点，分别针对源文件和产物文件进行不同的业务操作。
拓展方便：支持用户修改云函数逻辑，以满足更多定制化需求。

操作步骤

1. 登录 对象存储控制台。
2. 在左侧导航栏中，单击**存储桶列表**，进入存储桶列表页面。
3. 单击待操作的存储桶，进入存储桶详情页面。
4. 在左侧导航栏中，选择**数据工作流 > 工作流**，单击**创建工作流**。
5. 在创建工作流页面，配置业务需要的媒体处理节点，例如“音视频转码”节点，详情可参阅工作流。
6. 在创建工作流页面，添加**自定义函数**节点，选择您需要的常用功能函数。

如果您尚未创建此类函数，则单击**创建函数**。

创建常用功能函数的步骤如下：

1. 填写函数基础配置。输入函数名称前缀，勾选**授权SCF服务**，单击**下一步**。
2. 填写属性配置。按业务需求设置存储类型，自定义头部等，单击**下一步**。
3. 勾选处理对象，可以只针对工作流源文件执行该操作。
4. 单击**确认**。
5. COS 工作流封装了函数创建，版本发布，别名切流等过程，请等待其创建完成。
6. 创建完成后，选中刚刚创建的函数实例，单击**确定**。
7. 单击**保存**当前工作流即可。

操作验证

1. 登录 对象存储控制台。
2. 在左侧导航栏中，单击**存储桶列表**，进入存储桶列表页面。
3. 单击待操作的存储桶，进入存储桶详情页面。
4. 在左侧导航栏中，选择**数据工作流 > 工作流**，进入工作流管理页面。
5. 找到刚创建的工作流，单击启用，并前往指定存储桶路径上传媒体文件，等待工作流执行。
6. 待工作流运行结束后，即：

可以看到媒体处理成功，产物已生成。

源文件已成功设置了存储类型和自定义头部。

数据直传
Web 端直传实践

最近更新时间：2024-01-06 10:54:03

简介

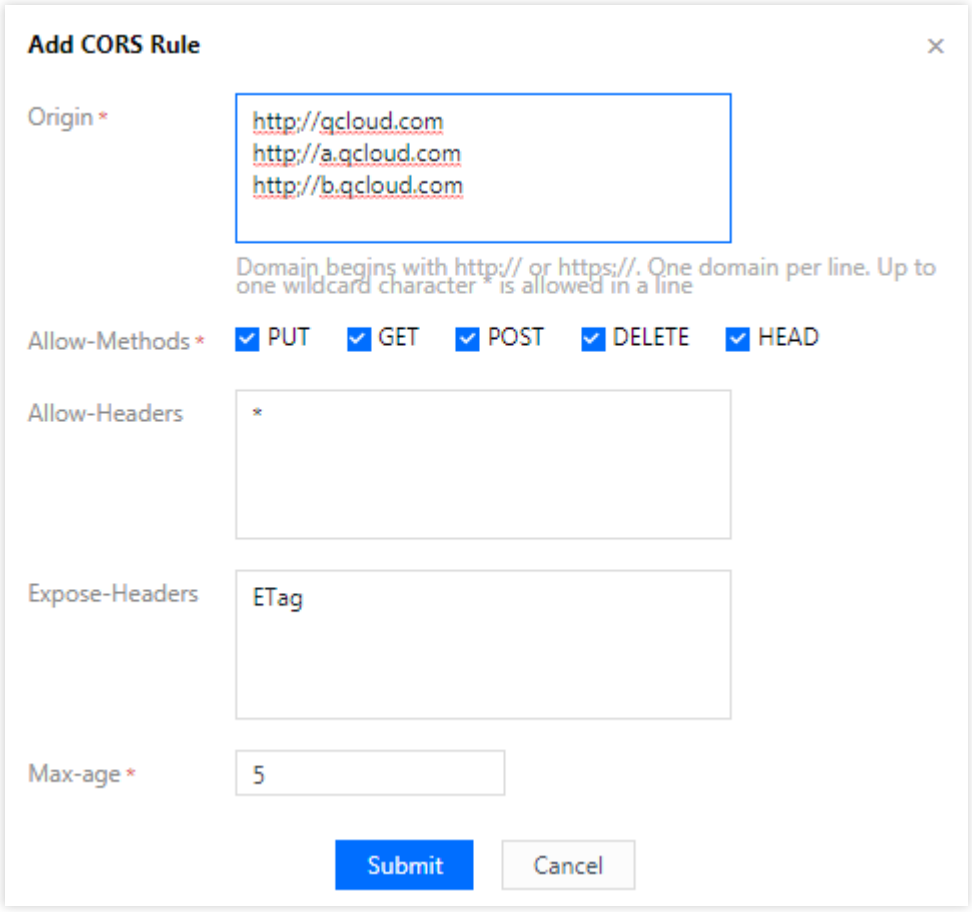
本文档介绍如何不依赖 SDK，用简单的代码，在网页（Web 端）直传文件到对象存储（Cloud Object Storage，COS）的存储桶。

注意

本文档内容基于 XML 版本的 API。

前提条件

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），详情请参见 创建存储桶 文档。
2. 进入存储桶详情页，单击**安全管理**页签。下拉页面找到**跨域访问CORS设置**配置项，单击**添加规则**，配置示例如下图，详情请参见 设置跨域访问 文档。



3. 登录 访问管理控制台， 获取您的项目 SecretId 和 SecretKey。

实践步骤

注意

正式部署时服务端请加一层您的网站本身的权限检验。

获取临时密钥和计算签名

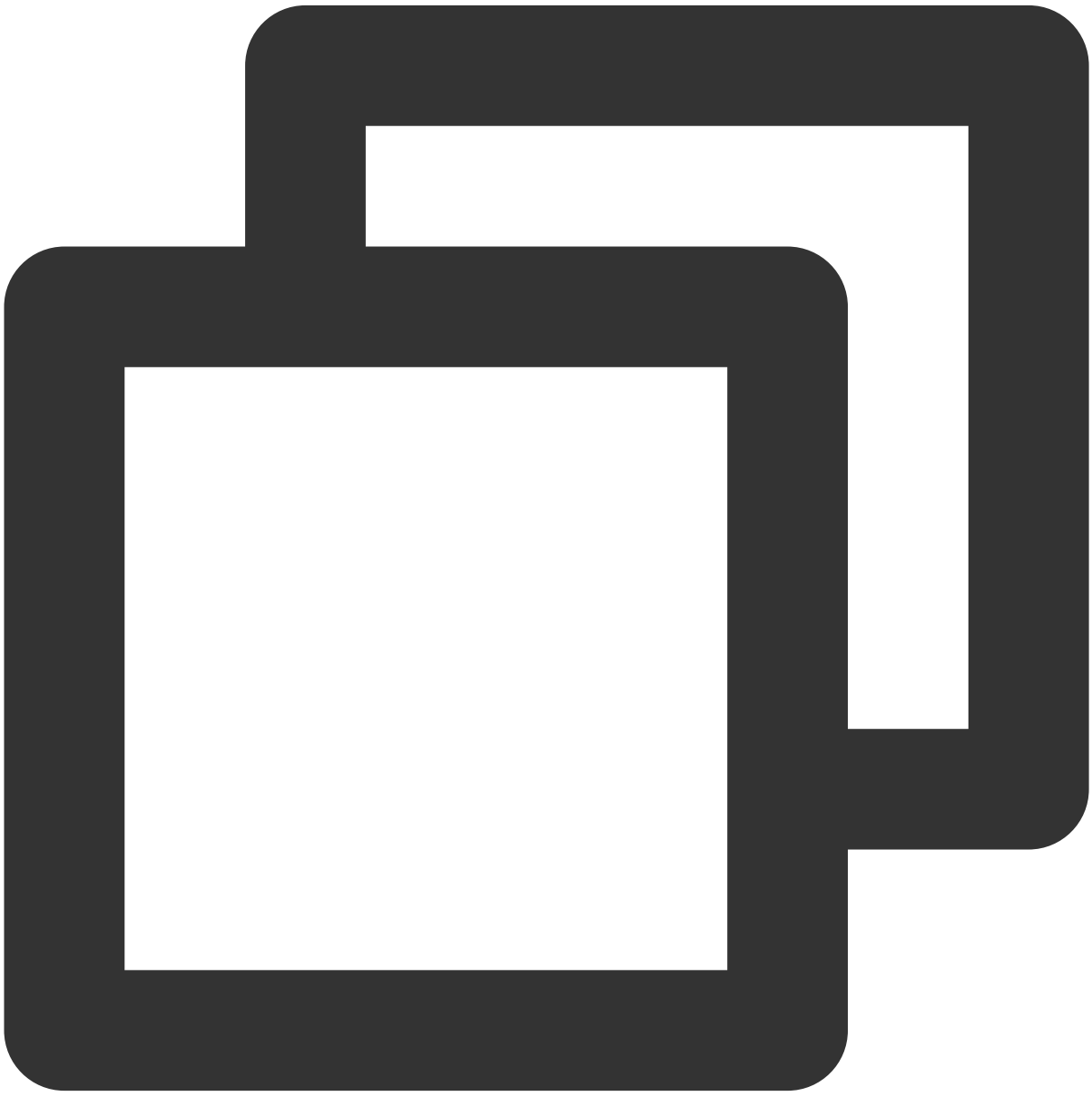
出于安全考虑，签名使用临时密钥，服务端搭建临时密钥服务，可参考 PHP 示例、Nodejs 示例。
如有其他语言或自行实现可以参考以下流程：
1. 向服务端获取临时密钥，服务端首先使用固定密钥 SecretId、SecretKey 向 STS 服务获取临时密钥，得到临时密钥 tmpSecretId、tmpSecretKey、sessionToken，详情请参考 临时密钥生成及使用指引 或 cos-sts-sdk 文档。
2. 前端通过 tmpSecretId、tmpSecretKey，以及 method、pathname 计算签名，可参考下文使用 cos-auth.js 来计算签名，如果业务需要也可以放在后端计算签名。
3. 如果使用 PutObject 接口上传文件，将计算得到的签名和 sessionToken，分别放到发请求时 header 的 authorization 和 x-cos-security-token 字段里。
如果使用 PostObject 接口上传文件，则将计算得到的签名和 sessionToken，分别放到发请求时表单的 Signature 和 x-cos-security-token 字段里。

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 PUT Object 文档，操作指引如下：

1. 按照 前提条件 的步骤，准备存储桶的相关配置。
2. 创建 `test.html` 文件，修改下方代码的 Bucket 和 Region，并复制到 `test.html` 文件。
3. 部署后端的签名服务，并修改 `test.html` 里的签名服务地址。
4. 将 `test.html` 放在 Web 服务器下，并通过浏览器访问页面，测试文件上传功能。



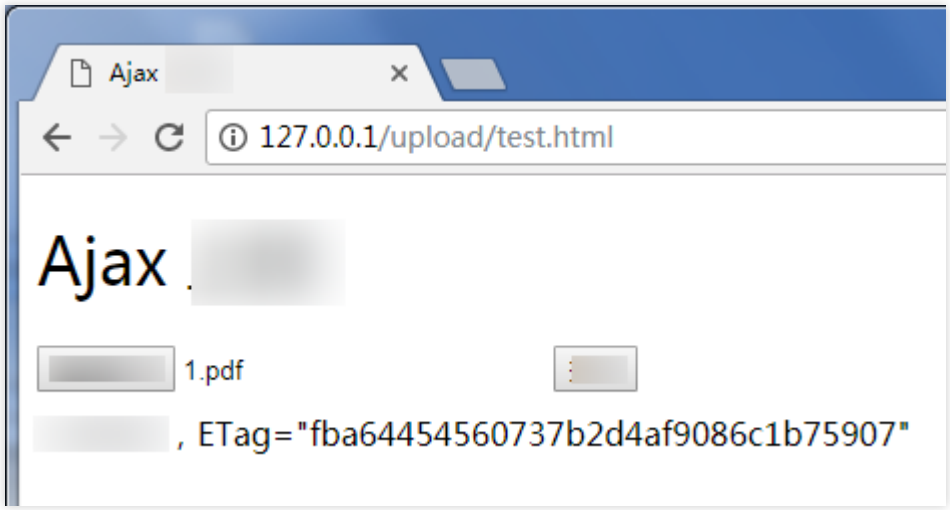
```
<!doctype html>
```

```
<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 src="https://unpkg.com/cos-js-sdk-v5/demo/common/cos-auth.min.js"></script>
<script>
    (function () {
        // 请求用到的参数
        var Bucket = 'examplebucket-1250000000';
        var Region = 'ap-guangzhou';
        var protocol = location.protocol === 'https:' ? 'https:' : 'http:';
        var prefix = protocol + '//' + Bucket + '.cos.' + Region + '.myqcloud.com/'

        // 对更多字符编码的 url encode 格式
        var camSafeUrlEncode = function (str) {
            return encodeURIComponent(str)
                .replace(/!/g, '%21')
                .replace(/'/g, '%27')
                .replace(/\(/g, '%28')
                .replace(/\)/g, '%29')
                .replace(/\*/g, '%2A');
        };

        // 计算签名
        var getAuthorization = function (options, callback) {
            // var url = 'http://127.0.0.1:3000/sts-auth' +
            var url = '../server/sts.php';
            var xhr = new XMLHttpRequest();
```

```
            xhr.open('GET', url, true);
            xhr.onload = function (e) {
                var credentials;
                try {
                    credentials = (new Function('return ' + xhr.responseText))().cr
                } catch (e) {}
                if (credentials) {
                    callback(null, {
                        SecurityToken: credentials.sessionToken,
                        Authorization: CosAuth({
                            SecretId: credentials.tmpSecretId,
                            SecretKey: credentials.tmpSecretKey,
                            Method: options.Method,
                            Pathname: options.Pathname,
                        })
                    });
                } else {
                    console.error(xhr.responseText);
                    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', Pathname: '/' + Key}, function (err, i

                if (err) {
                    alert(err);
                    return;
                }

                var auth = info.Authorization;
                var SecurityToken = info.SecurityToken;
                var url = prefix + camSafeUrlEncode(Key).replace(/%2F/g, '/');
                var xhr = new XMLHttpRequest();
                xhr.open('PUT', url, true);
                xhr.setRequestHeader('Authorization', auth);
                SecurityToken && xhr.setRequestHeader('x-cos-security-token', Secur
                xhr.upload.onprogress = function (e) {
                    console.log('上传进度 ' + (Math.round(e.loaded / e.total * 10000
                };
```

```
                xhr.onload = function () {
                    if (/^2\\d\\d$/.test('' + xhr.status)) {
                        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，ET
            });
        };
    })();
</script>

</body>
</html>
```

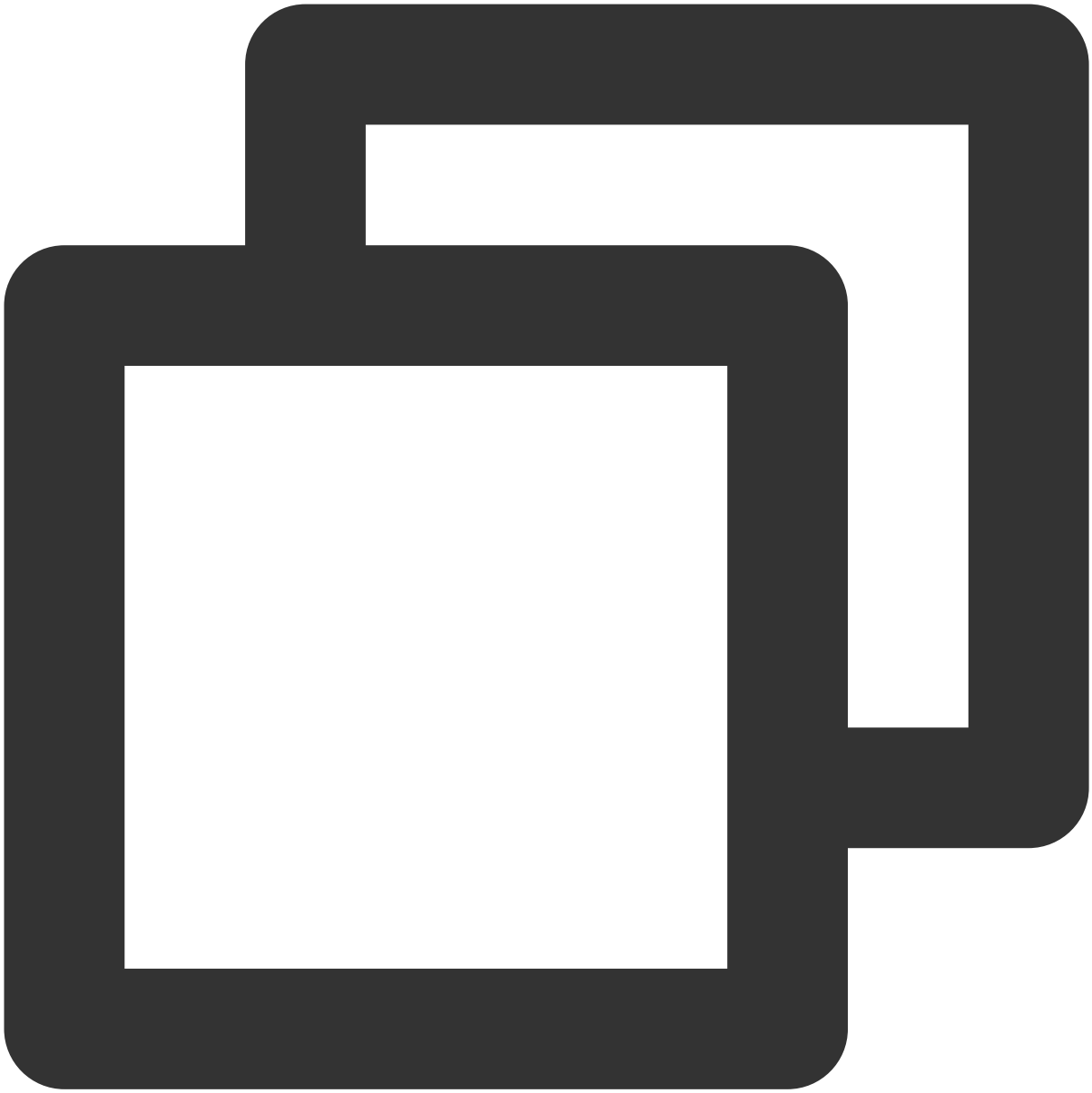
执行效果如下图：



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 Post Object 接口。操作指引：

1. 按照 前提条件 的步骤，准备存储桶。
2. 创建 `test.html` 文件，修改下方代码的 Bucket 和 Region，并复制到 `test.html` 文件。
3. 部署后端的签名服务，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. 将 `test.html` 和 `empty.html` 放在 Web 服务器下，并通过浏览器访问页面，测试文件上传功能。



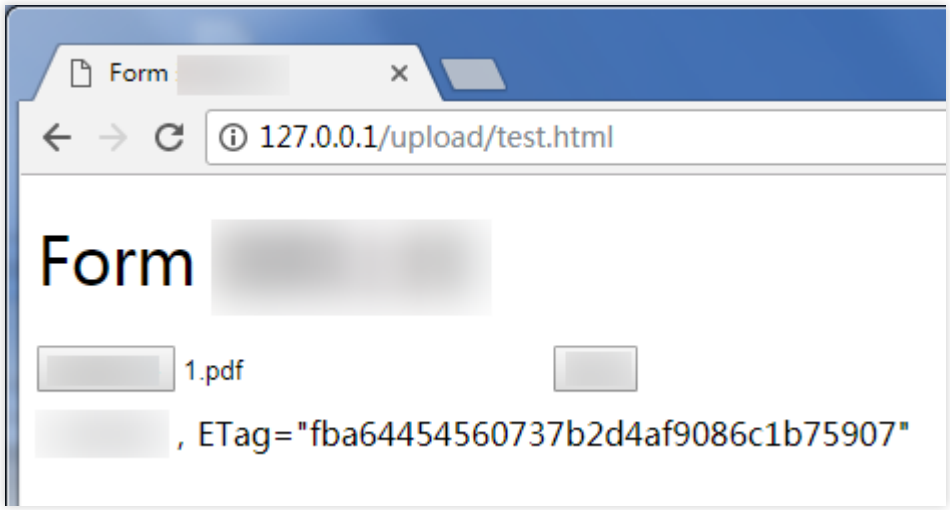
```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Form 表单简单上传</title>
    <style>h1, h2 {font-weight: normal;}#msg {margin-top:10px;}</style>
</head>
<body>

<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/fo
    <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
    <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" valu

    <!-- file 字段放在表单最后，避免文件内容过长影响签名判断和鉴权 -->
    <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"

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

<script src="https://unpkg.com/cos-js-sdk-v5/demo/common/cos-auth.min.js"></script>
<script>
    (function () {

        // 请求用到的参数
        var Bucket = 'examplebucket-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;

        // 对更多字符编码的 url encode 格式
        var camSafeUrlEncode = function (str) {
            return encodeURIComponent(str)
                .replace(/!/g, '%21')
                .replace(/'/g, '%27')
                .replace(/\(/g, '%28')
                .replace(/\)/g, '%29')
                .replace(/\*/g, '%2A');
        };

        // 计算签名
        var getAuthorization = function (options, callback) {
            // var url = 'http://127.0.0.1:3000/sts' +
            var url = '../server/sts.php';
            var xhr = new XMLHttpRequest();
            xhr.open('GET', url, true);
            xhr.onreadystatechange = function (e) {
```

```
            if (xhr.readyState === 4) {
                if (/^2\\d\\d$/.test('' + xhr.status)) {
                    var credentials;
                    try {
                        credentials = (new Function('return ' + xhr.responseTex
                    } catch (e) {}
                    if (credentials) {
                        callback(null, {
                            SecurityToken: credentials.sessionToken,
                            Authorization: CosAuth({
                                SecretId: credentials.tmpSecretId,
                                SecretKey: credentials.tmpSecretKey,
                                Method: options.Method,
                                Pathname: options.Pathname,
                            })
                        });
                    } else {
                        console.error(xhr.responseText);
                        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='
    };
    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 + camSafeUrlEncode(Key).replace(/%2F
            } 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
            getAuthorization({Method: 'POST', Pathname: '/'}, function (err, AuthDa
                // 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
                document.getElementById('key').value = Key;
                document.getElementById('Signature').value = AuthData.Authorization
                document.getElementById('x-cos-security-token').value = AuthData.Se
                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

小程序直传实践

最近更新时间：2024-01-06 10:54:03

简介

本文档介绍如何不依赖 SDK，用简单的代码，在小程序直传文件到对象存储（Cloud Object Storage，COS）的存储桶。

注意

本文档内容基于 XML 版本的 API。

前期条件

1. 登录 对象存储控制台 ，创建存储桶，设置 BucketName（存储桶名称） 和 Region（地域名称），详情请参见 创建存储桶 文档。
2. 登录 访问管理控制台，进入 API 密钥管理页面，获取您的项目 SecretId 和 SecretKey。
3. 配置小程序域名白名单

小程序里请求 COS 需要登录到微信公众平台，在“开发”->“开发设置”中，配置域名白名单。SDK 用到了两个接口：wx.uploadFile 和 wx.request。

cos.postObject 使用 wx.uploadFile 发送请求。

其他方法使用 wx.request 发送请求。

两者都需要在对应白名单里，配置 COS 域名。白名单里配置的域名格式有两种：

如果只用到一个存储桶，可以配置 Bucket 域名作为白名单域名，例如 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com` 。

如果用到多个存储桶，可以选择后缀式请求 COS，把 bucket 放在 pathname 里请求，这种方式需要配置地域域名作为白名单，例如 `cos.ap-guangzhou.myqcloud.com` 。具体可参考下文代码中的注释。

方案说明

执行过程

1. 在前端选择文件，前端将后缀发送给服务端。
2. 服务端根据后缀，生成带时间的随机 COS 文件路径，并计算对应的签名，返回 URL 和签名信息给前端。
3. 前端使用 PUT 或 POST 请求，直传文件到 COS。

方案优势

权限安全：使用服务端签名可以有效限定安全的权限范围，只能用于上传指定的一个文件路径。
路径安全：由服务端决定随机的 COS 文件路径，可以有效避免已有文件被覆盖的问题和安全风险。

后缀式请求

COS API 一般的请求格式都类似 `POST http://examplebucket-1250000000.cos.ap-beijing.myqcloud.com/` ，请求的域名是存储桶域名。这样如果在小程序里用到多个存储桶，则需要配置这个存储桶域名作为白名单域名。解决方法如下：
COS 提供了后缀式请求格式 `POST http://cos.ap-beijing.myqcloud.com/examplebucket-1250000000/` ，请求的域名是地域域名，存储桶名称放在请求的路径里。在小程序里用到同一个地域多个存储桶，只需要配置一个域名 `cos.ap-beijing.myqcloud.com` 作为白名单域名。
后缀式请求格式需要注意，签名时使用的路径要用以存储桶名称作为前缀的路径，例如 `/examplebucket-1250000000/` 。

实践步骤

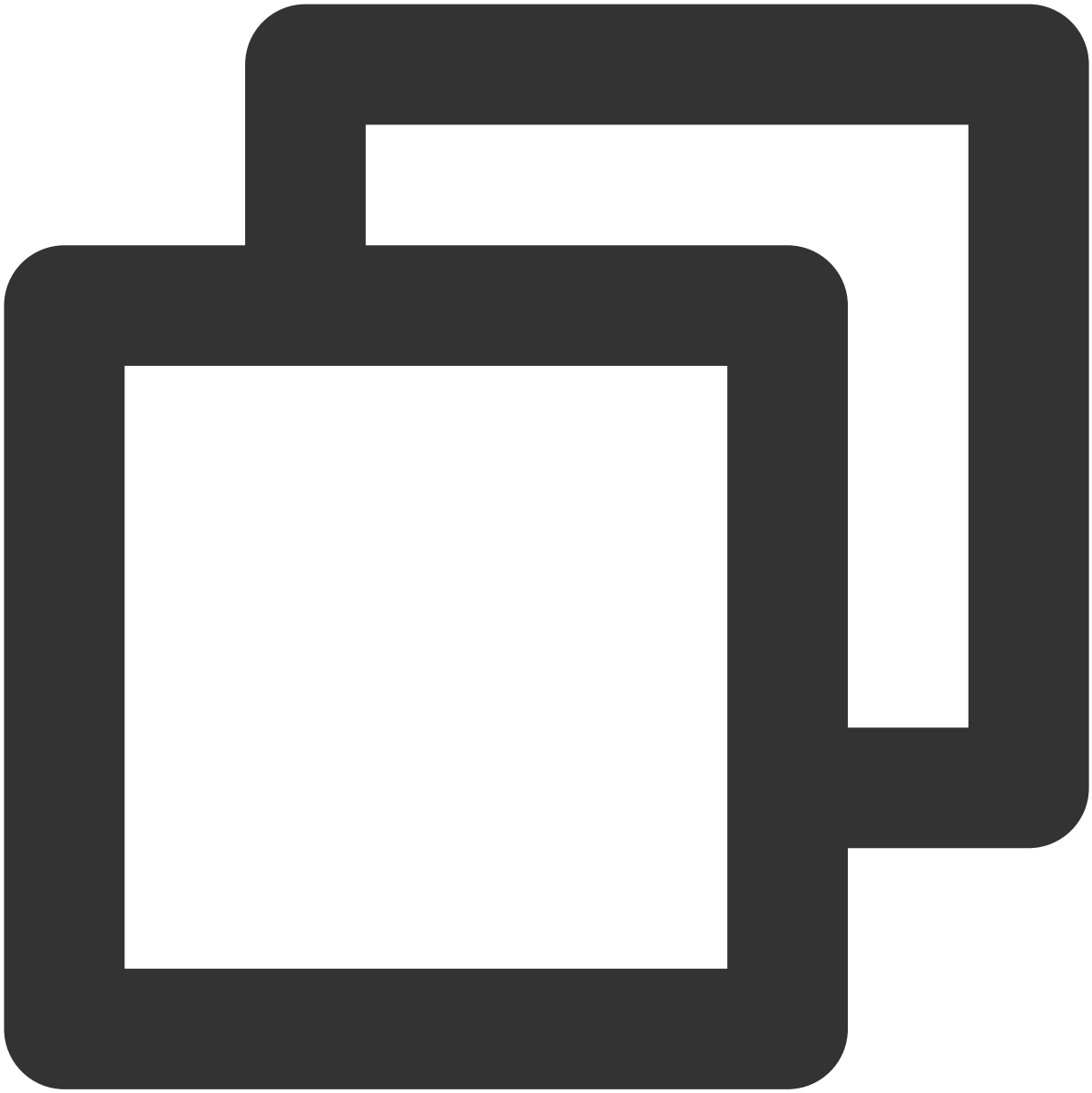
配置服务端实现签名

注意
正式部署时服务端请加一层您的网站本身的权限检验。
如何计算签名可参考文档 请求签名。
服务端使用 Nodejs 计算签名代码可参考 Nodejs 示例。

小程序上传示例

以下代码同时举例了 PUT Object 接口（推荐使用）和POST Object 接口，操作指引如下：

使用 POST 上传



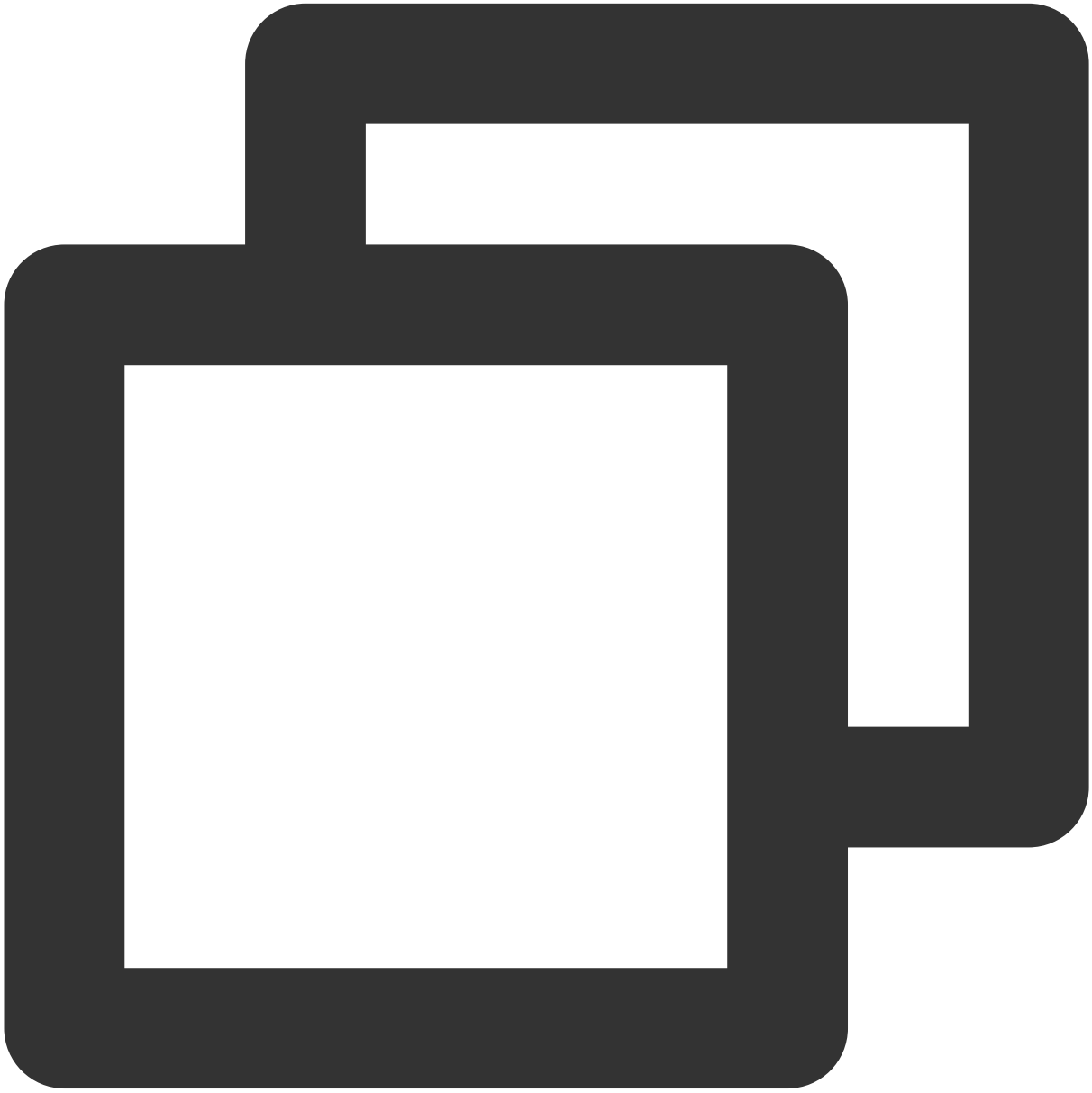
```
var uploadFile = function () {
  // 对更多字符编码的 url encode 格式
  var camSafeUrlEncode = function (str) {
    return encodeURIComponent(str)
      .replace(/!/g, '%21')
      .replace(/'/g, '%27')
      .replace(/\(/g, '%28')
      .replace(/\)/g, '%29')
      .replace(/\*/g, '%2A');
  };
```

```
// 获取签名
var getAuthorization = function (options, callback) {
  wx.request({
    method: 'GET',
    // 替换为自己服务端地址 获取post上传签名
    url: 'http://127.0.0.1:3000/post-policy?ext=' + options.ext,
    dataType: 'json',
    success: function (result) {
      var data = result.data;
      if (data) {
        callback(data);
      } else {
        wx.showModal({
          title: '临时密钥获取失败',
          content: JSON.stringify(data),
          showCancel: false,
        });
      }
    },
    error: function (err) {
      wx.showModal({
        title: '临时密钥获取失败',
        content: JSON.stringify(err),
        showCancel: false,
      });
    },
  });
};

var postFile = function ({ prefix, filePath, key, formData }) {
  var requestTask = wx.uploadFile({
    url: prefix,
    name: 'file',
    filePath: filePath,
    formData: formData,
    success: function (res) {
      var url = prefix + '/' + camSafeUrlEncode(key).replace(/%2F/g, '/');
      if (res.statusCode === 200) {
        wx.showModal({ title: '上传成功', content: url, showCancel: false });
      } else {
        wx.showModal({
          title: '上传失败',
          content: JSON.stringify(res),
          showCancel: false,
        });
      }
      console.log(res.header['x-cos-request-id']);
```

```
        console.log(res.statusCode);
        console.log(url);
      },
      fail: function (res) {
        wx.showModal({
          title: '上传失败',
          content: JSON.stringify(res),
          showCancel: false,
        });
      },
    });
    requestTask.onProgressUpdate(function (res) {
      console.log('正在进度:', res);
    });
  };

  // 上传文件
  var uploadFile = function (filePath) {
    var extIndex = filePath.lastIndexOf('.');
    var fileExt = extIndex >= -1 ? filePath.substr(extIndex + 1) : '';
    getAuthorization({ ext: fileExt }, function (AuthData) {
      // 请求用到的参数
      var prefix = 'https://' + AuthData.cosHost;
      var key = AuthData.cosKey; // 让服务端来决定文件名更安全
      var formData = {
        key: key,
        success_action_status: 200,
        'Content-Type': '',
        'q-sign-algorithm': AuthData.qSignAlgorithm,
        'q-ak': AuthData.qAk,
        'q-key-time': AuthData.qKeyTime,
        'q-signature': AuthData.qSignature,
        policy: AuthData.policy,
      };
      if (AuthData.securityToken)
        formData['x-cos-security-token'] = AuthData.securityToken;
      postFile({ prefix, filePath, key, formData });
    });
  };

  // 选择文件
  wx.chooseMedia({
    count: 1, // 默认9
    sizeType: ['original'], // 可以指定是原图还是压缩图，这里默认用原图
    sourceType: ['album', 'camera'], // 可以指定来源是相册还是相机，默认二者都有
    success: function (res) {
      uploadFile(res.tempFiles[0].tempFilePath);
```

```
    },
  });
};
```

使用 PUT 上传



```
var uploadFile = function () {
  // 对更多字符编码的 url encode 格式
  var camSafeUrlEncode = function (str) {
    return encodeURIComponent(str)
      .replace(/!/g, '%21')
```

```
      .replace(/'/g, '%27')
      .replace(/\(/g, '%28')
      .replace(/\)/g, '%29')
      .replace(/\*/g, '%2A');
  };

  // 获取签名
  var getAuthorization = function (options, callback) {
    wx.request({
      method: 'GET',
      // 替换为自己服务端地址 获取put上传签名
      url: 'http://127.0.0.1:3000/put-sign?ext=' + options.ext,
      dataType: 'json',
      success: function (result) {
        var data = result.data;
        if (data) {
          callback(data);
        } else {
          wx.showModal({
            title: '临时密钥获取失败',
            content: JSON.stringify(data),
            showCancel: false,
          });
        }
      },
      error: function (err) {
        wx.showModal({
          title: '临时密钥获取失败',
          content: JSON.stringify(err),
          showCancel: false,
        });
      },
    });
  };

  var putFile = function ({ prefix, filePath, key, AuthData }) {
    // put上传需要读取文件的真实内容来上传
    const wxfs = wx.getFileSystemManager();
    wxfs.readFile({
      filePath: filePath,
      success: function (fileRes) {
        var requestTask = wx.request({
          url: prefix + '/' + key,
          method: 'PUT',
          header: {
            Authorization: AuthData.authorization,
            'x-cos-security-token': AuthData.securityToken,
```

```
          },
          data: fileRes.data,
          success: function success(res) {
            var url = prefix + '/' + camSafeUrlEncode(key).replace(/%2F/g, '/');
            if (res.statusCode === 200) {
              wx.showModal({
                title: '上传成功',
                content: url,
                showCancel: false,
              });
            } else {
              wx.showModal({
                title: '上传失败',
                content: JSON.stringify(res),
                showCancel: false,
              });
            }
            console.log(res.statusCode);
            console.log(url);
          },
          fail: function fail(res) {
            wx.showModal({
              title: '上传失败',
              content: JSON.stringify(res),
              showCancel: false,
            });
          },
        });
        requestTask.onProgressUpdate(function (res) {
          console.log('正在进度:', res);
        });
      },
    });
};

// 上传文件
var uploadFile = function (filePath) {
  var extIndex = filePath.lastIndexOf('.');
  var fileExt = extIndex >= -1 ? filePath.substr(extIndex + 1) : '';
  getAuthorization({ ext: fileExt }, function (AuthData) {
    const prefix = 'https://' + AuthData.cosHost;
    const key = AuthData.cosKey;
    putFile({ prefix, filePath, key, AuthData });
  });
};

// 选择文件
```

```
  wx.chooseMedia({
    count: 1, // 默认9
    sizeType: ['original'], // 可以指定是原图还是压缩图，这里默认用原图
    sourceType: ['album', 'camera'], // 可以指定来源是相册还是相机，默认二者都有
    success: function (res) {
      uploadFile(res.tempFiles[0].tempFilePath);
    },
  });
};
```

相关文档

如需使用小程序 SDK，请参见小程序 SDK 快速入门 文档。

移动应用直传实践

最近更新时间：2024-01-06 10:54:03

简介

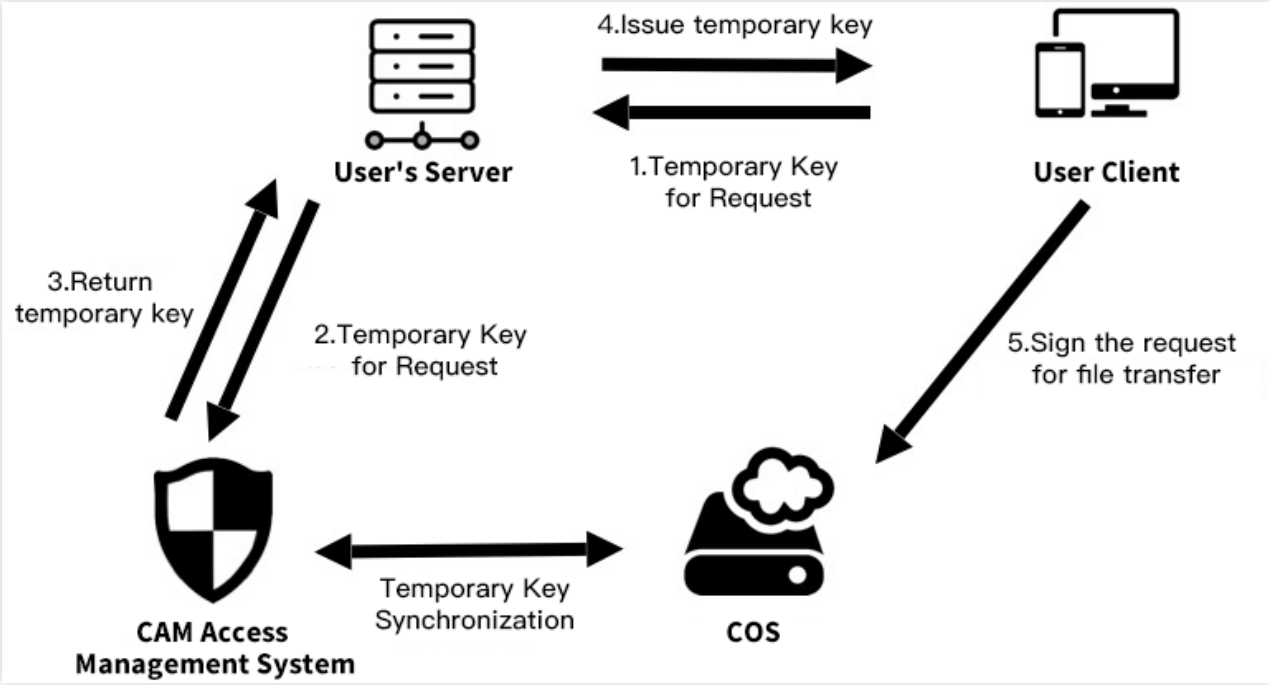
本文主要介绍基于腾讯云对象存储 COS，如何快速实现一个移动应用的文件直传功能。您的服务器上只需要生成和管理访问密钥，无需关心细节，文件数据都存放在腾讯云 COS 上。

架构说明

对于客户端应用，把永久密钥放到客户端代码中，这既容易泄露您的密钥信息，也不便于控制用户访问权限。我们建议通过在请求时携带临时密钥，您可以临时授权您的 App 访问您的存储资源，而不会泄露您的永久密钥，密钥的有效期由您指定，过期后自动失效。
COS 移动端 SDK（Android/IOS）均很好的支持了通过临时密钥来授权请求，您只需要在后台搭建一个临时密钥的服务，即可无缝给终端 COS 请求进行授权。

架构图

整体架构图如下所示：



其中：
用户客户端：即用户手机 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。
用户服务端：用户自己的后台服务器，这里用于获取临时密钥，并返回给应用 App。

前提条件

1. 创建存储桶。
在 COS 控制台 上创建存储桶。按照您的需求，请将存储桶权限设置为私有读写或者公有读私有写。详细操作步骤请参见 创建存储桶 和 设置访问权限 。
2. 获取永久密钥。
临时密钥需要通过永久密钥生成。请前往 API 密钥管理 获取 SecretId、SecretKey，前往 账号信息 获取 APPID。

实践步骤

搭建临时密钥服务

出于安全考虑，签名使用临时密钥，需要服务端搭建临时密钥服务，并提供 API 接口给客户端使用。具体搭建步骤请参见 临时密钥生成及使用指引。

注意

正式部署时服务端请加一层您的网站本身的权限检验。

选择合适的权限

按照最小权限原则，建议您按照自己的需求，通过 Policy 控制临时密钥的权限范围。服务器下发一个完全读写权限的密钥，一旦被攻击，可能导致其他用户的数据泄露。具体的配置方法请参见 临时密钥生成及使用指引。

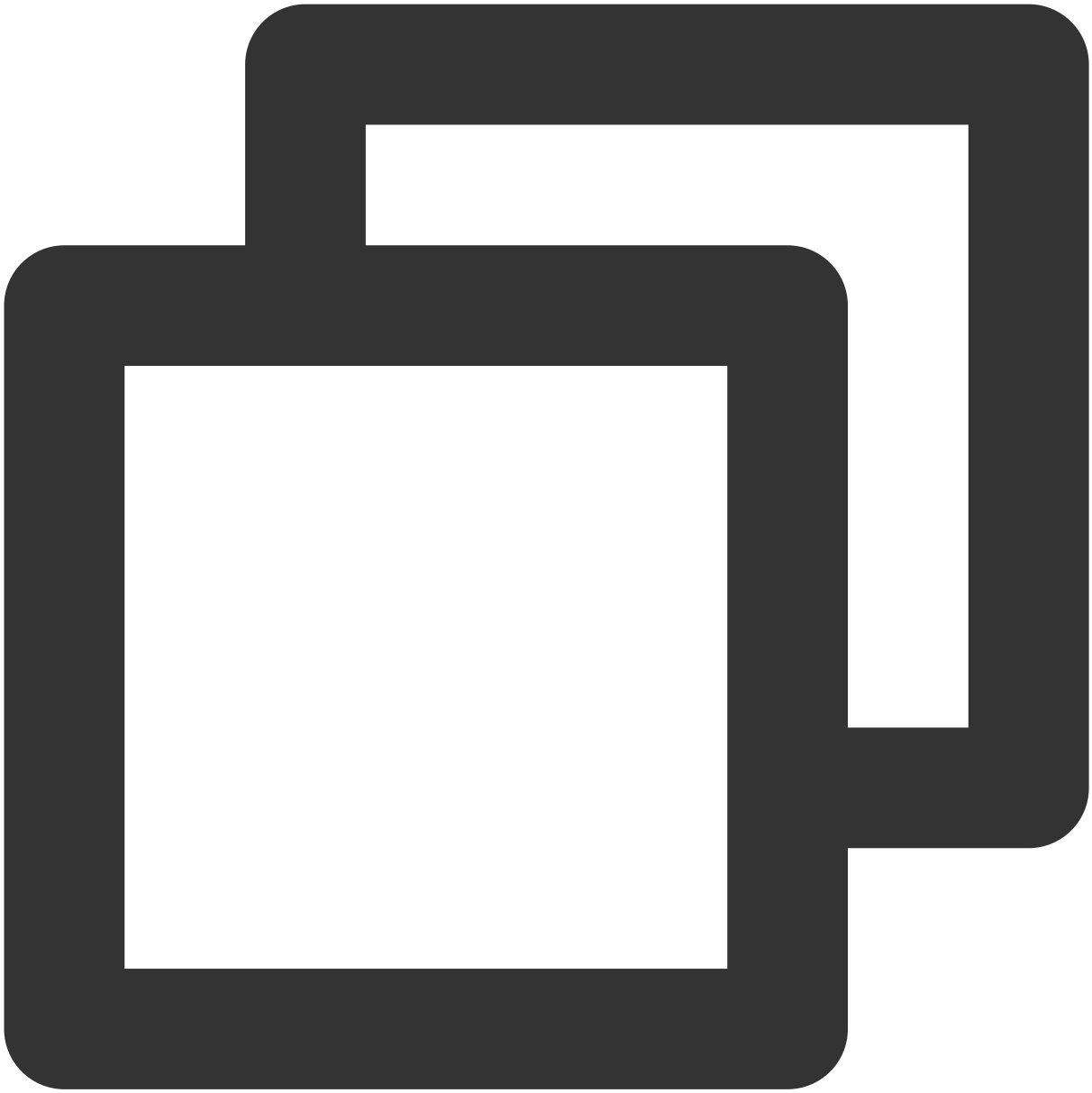
SDK 接入授权服务

Android

搭建好临时密钥服务后，您需要将 SDK 接入到授权服务上，SDK 会负责控制请求的并发数，也会将有效的密钥缓存在本地，并在密钥失效之后重新再次请求，您无需管理获取的密钥。

标准响应体授权

如果您直接将 STS SDK 中得到的 JSON 数据作为临时密钥服务的响应体（cossign 即采用了这种方式），那么您可以使用如下代码来创建 COS SDK 中的授权类：



```
import android.content.Context;
import com.tencent.cos.xml.*;
import com.tencent.qcloud.core.auth.*;
import com.tencent.qcloud.core.common.*;
import com.tencent.qcloud.core.http.*;
import java.net.*;


Context context = ...;
CosXmlServiceConfig cosXmlServiceConfig = ...;
```

```
/**
 * 获取授权服务的 url 地址
 */
URL url = null; // 后台授权服务的 url 地址
try {
    url = new URL("your_auth_server_url");
} catch (MalformedURLException e) {
    e.printStackTrace();
}

/**
 * 初始化 {@link QCloudCredentialProvider} 对象，来给 SDK 提供临时密钥。
 */
QCloudCredentialProvider credentialProvider = new SessionCredentialProvider(new Htt
                .url(url)
                .method("GET")
                .build());

CosXmlService cosXmlService = new CosXmlService(context, cosXmlServiceConfig, crede
```

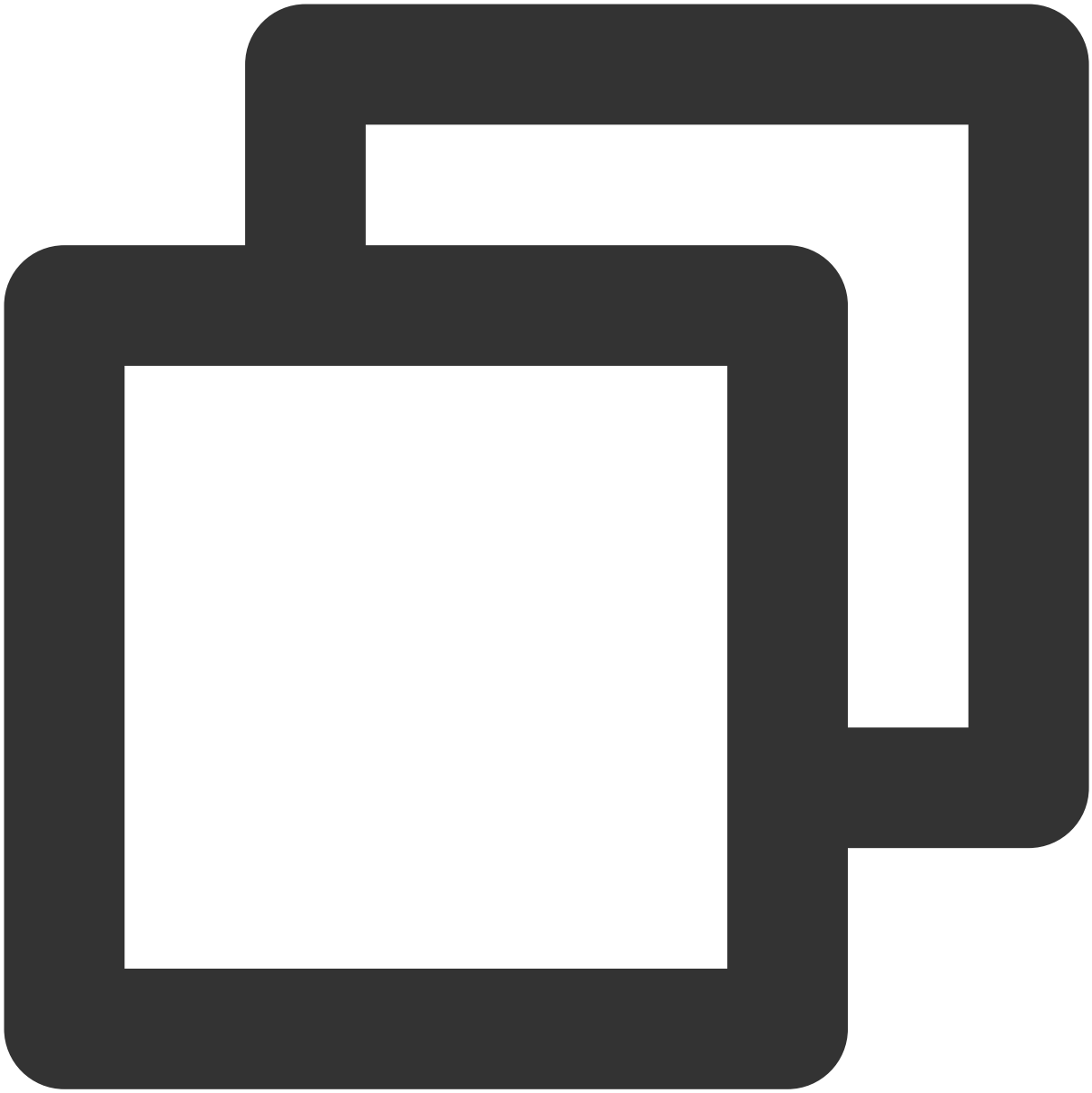
说明

这种方式下签名的开始时间为手机本地时间，因此如果手机本地时间偏差较大（十分钟以上），可能会导致签名出错，这种情况建议使用下述的自定义响应体授权。

自定义响应体授权

如果您想获得更大的灵活性，例如自定义临时密钥服务的 HTTP 响应体，给终端返回服务器时间作为签名的开始时间，用来避免由于用户手机本地时间偏差过大导致的签名不正确，或者使用其他的协议来进行终端和服务端之间的通信，那么您可以继承 BasicLifecycleCredentialProvider 类，并实现其 fetchNewCredentials()：

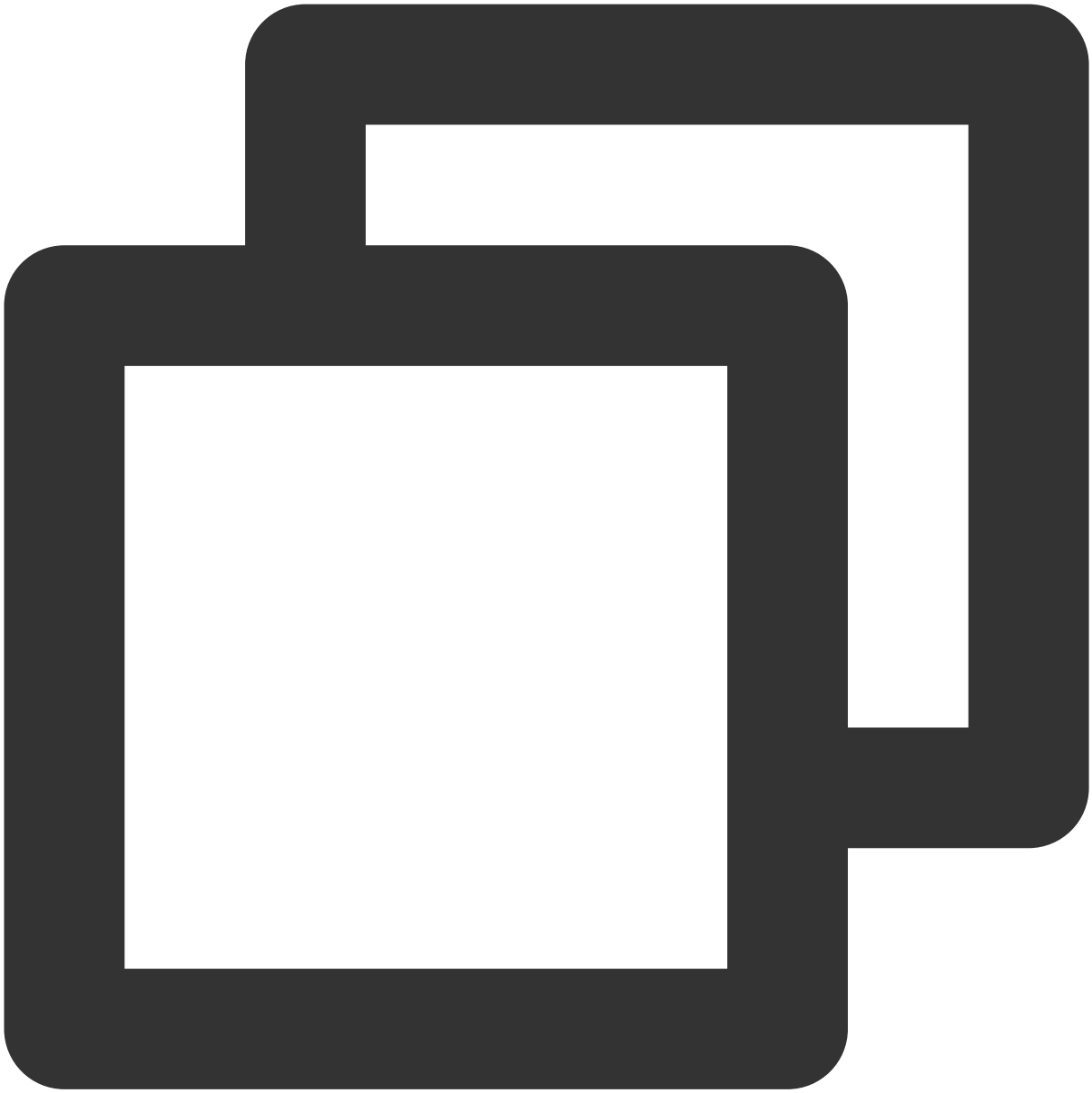
请先定义一个 MyCredentialProvider 类：



```
import android.content.Context;
import com.tencent.cos.xml.*;
import com.tencent.qcloud.core.auth.*;
import com.tencent.qcloud.core.common.*;
import com.tencent.qcloud.core.http.*;
import java.net.*;


public class MyCredentialProvider extends BasicLifecycleCredentialProvider {

    @Override
```

```
    protected QCloudLifecycleCredentials fetchNewCredentials() throws QCloudClientE

        // 首先从您的临时密钥服务器获取包含了签名信息的响应
        ....

        // 然后解析响应，获取密钥信息
        String tmpSecretId = ...;
        String tmpSecretKey = ...;
        String sessionToken = ...;
        long expiredTime = ...;

        // 返回服务器时间作为签名的起始时间
        long beginTime = ...;

        // todo something you want

        // 最后返回临时密钥信息对象
        return new SessionQCloudCredentials(tmpSecretId, tmpSecretKey, sessionToken
    }
}
```

利用您定义的 MyCredentialProvider 实例来授权请求：



```
import android.content.Context;
import com.tencent.cos.xml.*;
import com.tencent.qcloud.core.auth.*;
import com.tencent.qcloud.core.common.*;
import com.tencent.qcloud.core.http.*;
import java.net.*;

Context context = ...;
CosXmlServiceConfig cosXmlServiceConfig = ...;

/**
```

```
 * 初始化 {@link QCloudCredentialProvider} 对象，来给 SDK 提供临时密钥。
 */
QCloudCredentialProvider credentialProvider = new MyCredentialProvider();

CosXmlService cosXmlService = new CosXmlService(context, cosXmlServiceConfig, crede
```

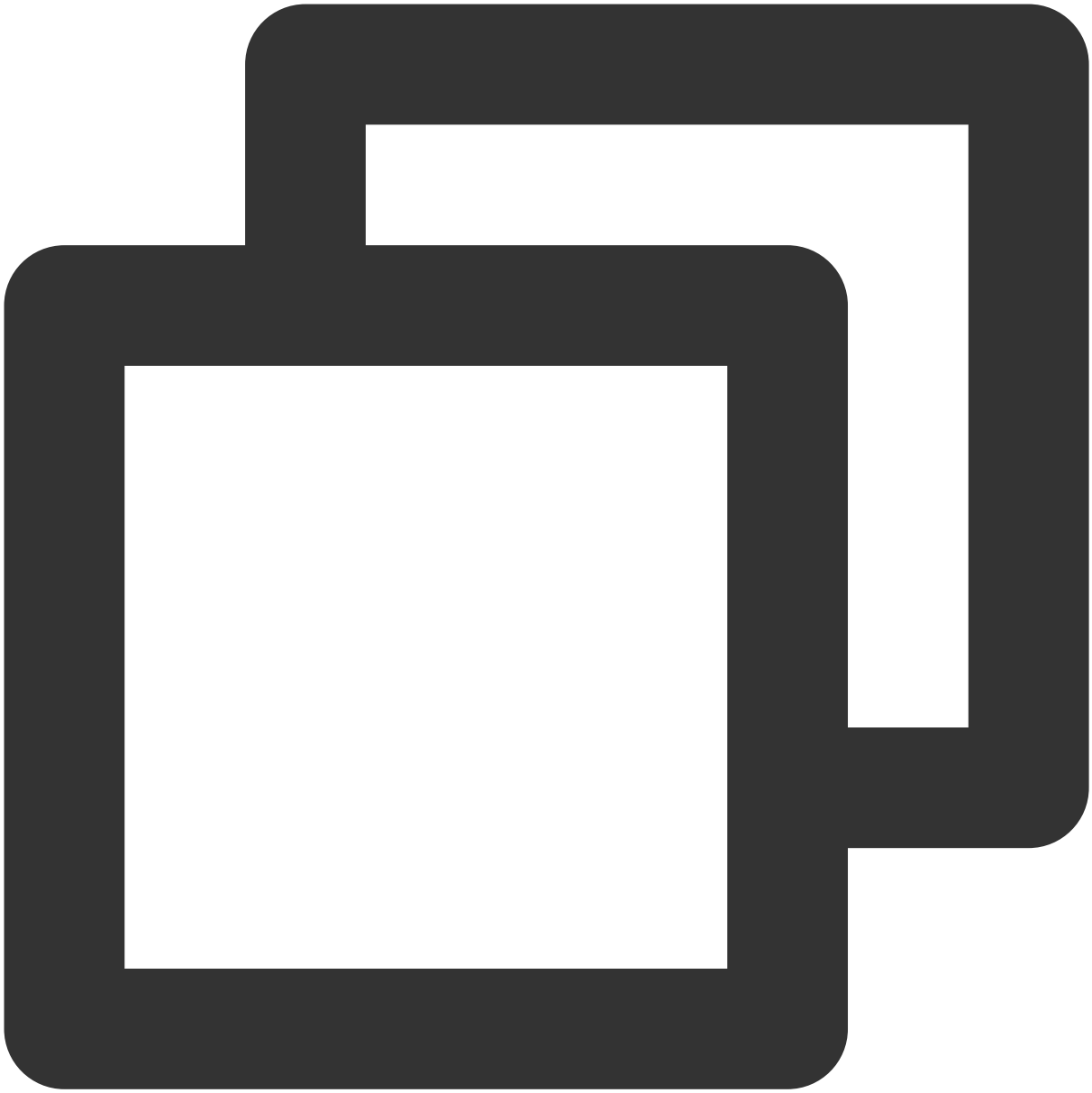
完整的示例代码请参见 Android COS Transfer Practice。

更多关于 Android 如何向 COS 上传和下载文件，请参见 Android SDK 快速入门。

iOS

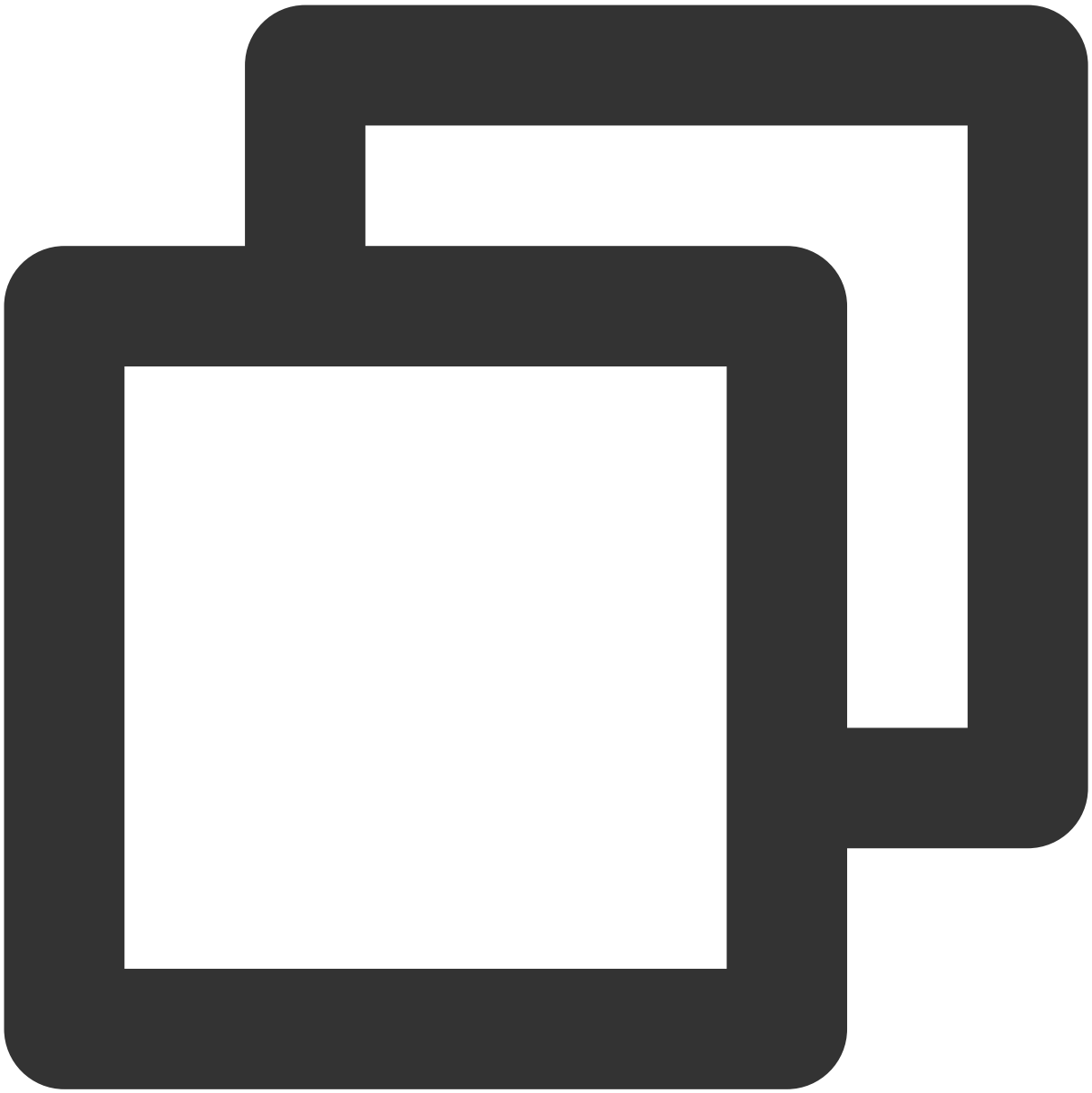
我们提供了 QCloudCredentailFenceQueue 来方便地获取和管理临时签名。QCloudCredentailFenceQueue 提供了栅栏机制，也就是说您使用 QCloudCredentailFenceQueue 获取签名的话，所有需要获取签名的请求会等待签名完成后再执行，免去了自己管理异步过程。

使用 QCloudCredentailFenceQueue，我们需要先生成一个实例。



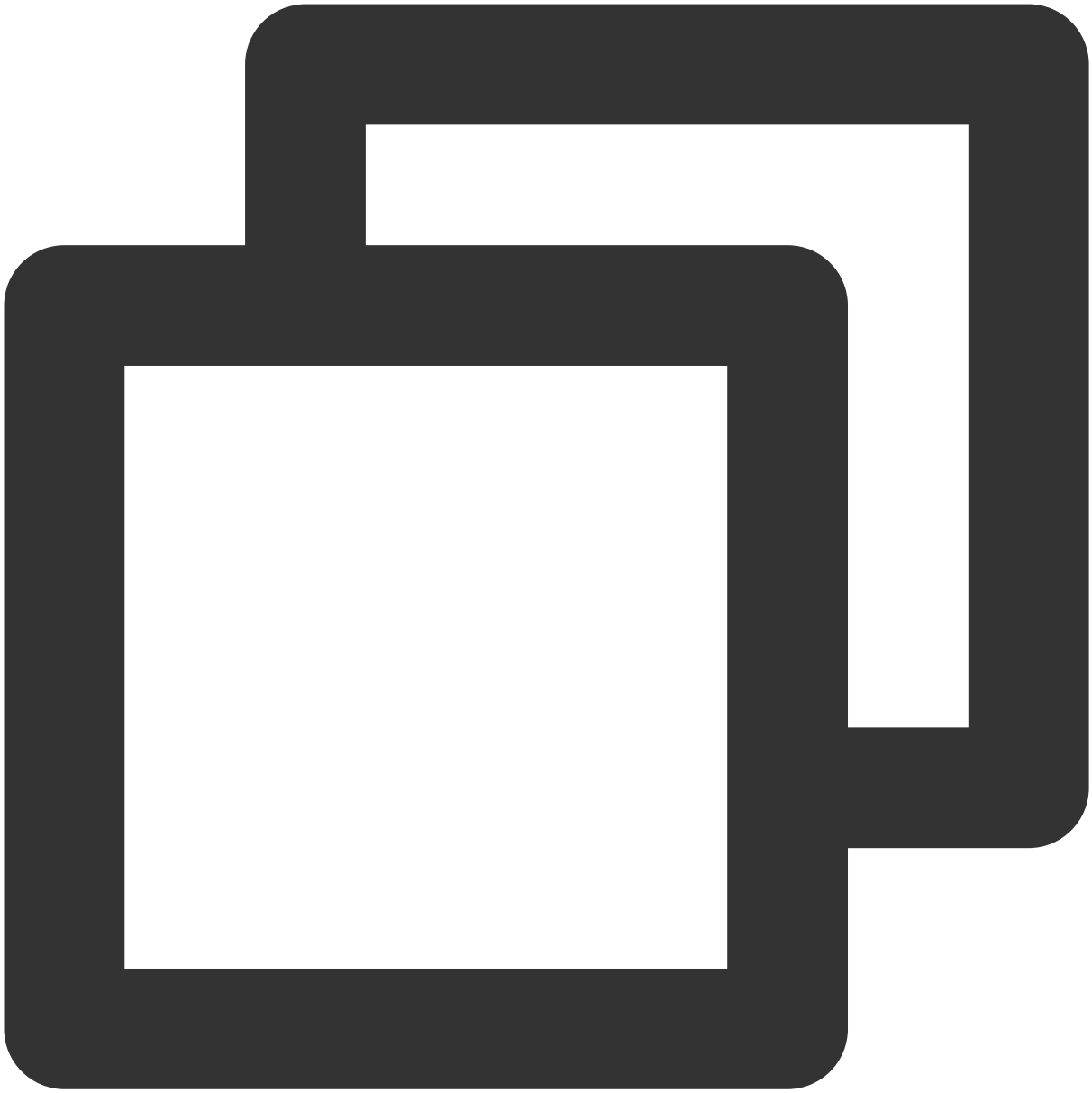
```
 //AppDelegate.m
//AppDelegate需遵循QCloudCredentailFenceQueueDelegate协议
//
- (BOOL)application:(UIApplication * )application didFinishLaunchingWithOptions:(NS
    // init step
    self.credentialFenceQueue = [QCloudCredentailFenceQueue new];
    self.credentialFenceQueue.delegate = self;
    return YES;
}
```

然后调用 QCloudCredentailFenceQueue 的类需要遵循 QCloudCredentailFenceQueueDelegate 并实现协议内定义的方法：



```
- (void) fenceQueue:(QCloudCredentailFenceQueue * )queue requestCreatorWithContinue
```

当通过 QCloudCredentailFenceQueue 去获取签名时，所有需要签名的 SDK 里的请求都会等待该协议定义的方法内拿到了签名所需的参数并生成有效的签名后执行。请看以下示例：



```
- (void)fenceQueue:(QCloudCredentailFenceQueue *)queue requestCreatorWithContinue:(
    QCloudHTTPRequest* request = [QCloudHTTPRequest new];
    request.requestData.serverURL = @“your sign service url”;//请求的URL

    [request setConfigureBlock:^(QCloudRequestSerializer *requestSerializer, QCloud
        requestSerializer.serializerBlocks = @[QCloudURLFuseWithURLEncodeParamters]
        responseSerializer.serializerBlocks = @[QCloudAcceptRespnseCodeBlock([NSSet
                                              QCloudResponseJSONSerilizerBlock];/
    }];

    [request setFinishBlock:^(id response, NSError *error) {
```

```
        if (error) {
            error = [NSError errorWithDomain:@"com.tac.test" code:-1111 userInfo:@{
            continueBlock(nil, error);
        } else {
            QCloudCredential* crendential = [[QCloudCredential alloc] init];
            crendential.secretID = response[@"data"][@"credentials"][@"tmpSecretId"
            crendential.secretKey = response[@"data"][@"credentials"][@"tmpSecretKe
            credential.startDate =[NSDate dateWithTimeIntervalSince1970:@"返回的服务器
            crendential.experationDate = [NSDate dateWithTimeIntervalSinceNow:[resp
            crendential.token = response[@"data"][@"credentials"][@"sessionToken"];
            QCloudAuthentationV5Creator* creator = [[QCloudAuthentationV5Creator al
            continueBlock(creator, nil);

        }
    }];
    [[QCloudHTTPSessionManager shareClient] performRequest:request];
}
```

更多关于 iOS 如何向 COS 上传和下载文件，请参见 iOS SDK 快速入门。

体验

Android

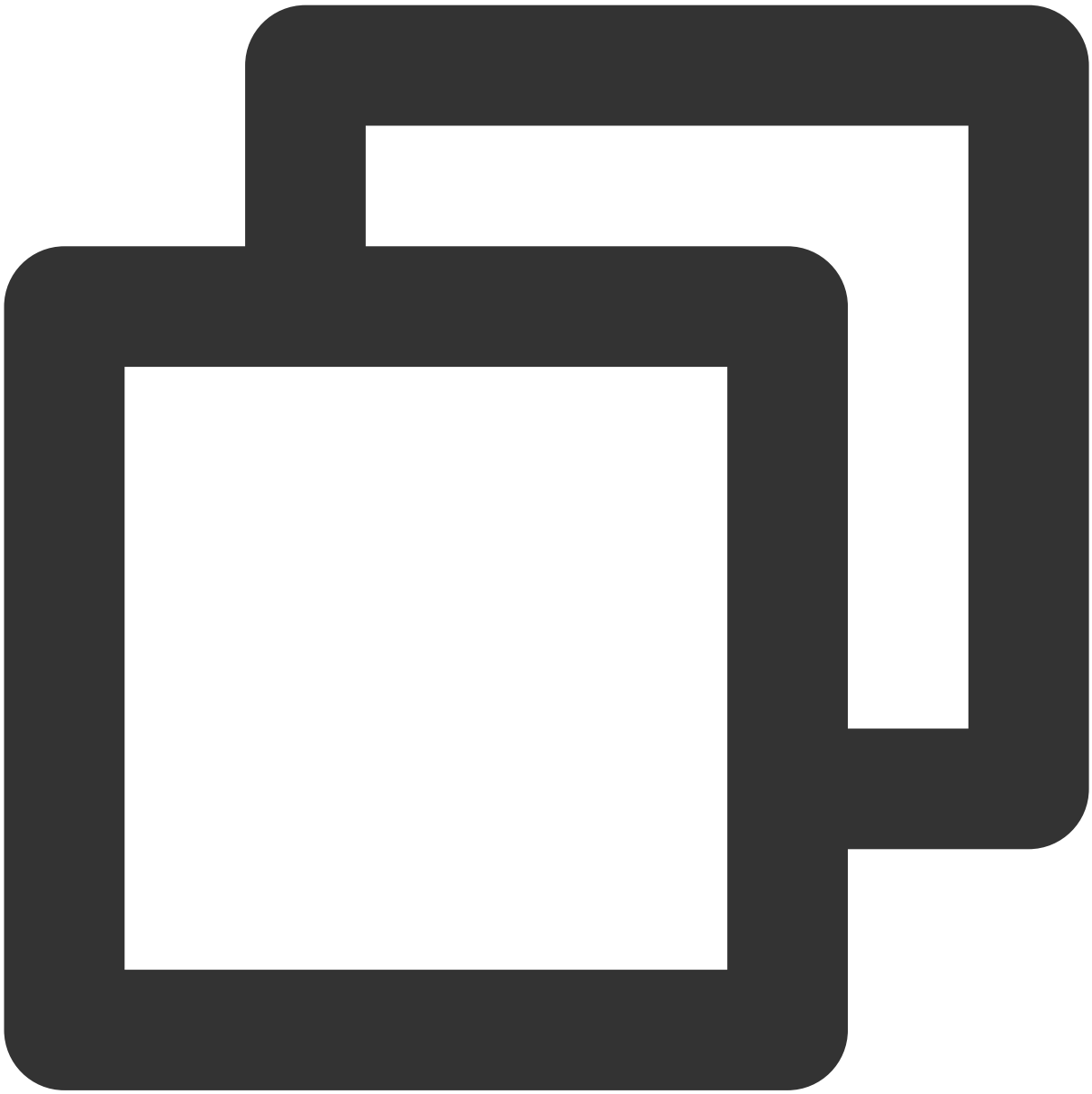
您可以单击 这里，或者使用 Android 手机浏览器 App，扫描下方二维码下载体验 demo：

完整的代码请参见：COS Android Demo。

iOS

iOS 的完整示例工程请参见 COS iOS Demo。

修改 QCloudCOSXMLDemo/QCloudCOSXMLDemo/key.json 文件，填入 APPID，secretID，secretKey 参数值，然后执行以下命令：



```
pod install
```

说明

APPID，secretID，secretKey 可前往 API 密钥管理 页面获取。

执行命令之后，打开 QCloudCOSXMLDemo.xcworkspace 即可进入 Demo 体验。

uni-app 直传实践

最近更新时间：2024-01-06 10:54:03

简介

本文档介绍如何不依赖 SDK，使用简单的代码，在 uni-app 直传文件到对象存储（Cloud Object Storage，COS）的存储桶。

说明

本文档内容基于 XML API 的 PostObject 接口。

方案说明

执行过程

1. 在前端选择文件，前端将后缀发送给服务端。
2. 服务端根据后缀，生成带时间的随机 COS 文件路径，并计算对应的 PostObject policy 签名，返回 URL 和签名信息给前端。
3. 前端调用 PostObject 接口，直传文件到 COS。

方案优势

权限安全：使用 PostObject policy 签名可以有效限定安全的权限范围，只能用于上传指定的一个文件路径。
路径安全：由服务端决定随机的 COS 文件路径，可以有效避免已有文件被覆盖的问题和安全风险。
兼容多端：使用 uni-app 提供的选文件、上传接口，可以一份代码多端可兼容（Web/小程序/App）。

前提条件

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），详情请参见 创建存储桶 文档。
2. 登录 访问管理控制台， 获取您的项目 SecretId 和 SecretKey。
3. 进入刚创建的存储桶详情页面，在**安全管理 > 跨域访问CORS设置**页面，单击**添加规则**。配置示例如下图，详情请参见 设置跨域访问 文档。

实践步骤

注意

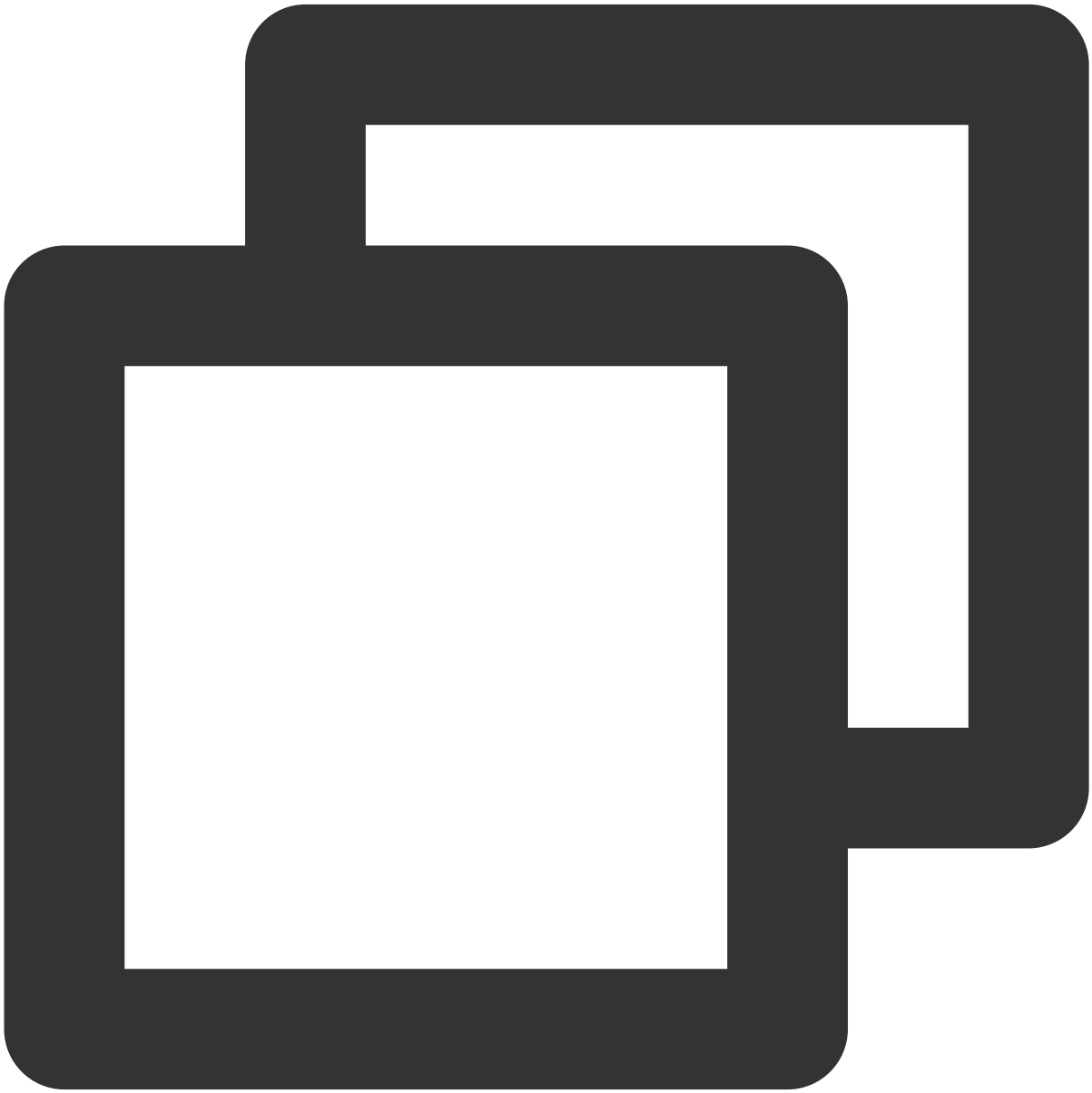
正式部署前，建议您在服务端加一层对网站本身的权限检验。

前端上传

1. 参考 post-policy 示例 实现一个服务端接口，用于生成随机文件路径、计算签名，并返回给前端。
2. 使用 HBuilderX 默认模板创建 uni-app 应用。

创建后，该应用为一个基于 Vue 的项目。

3. 复制以下代码替换 pages/index/index.vue 文件内容，并修改调用的 post-policy 接口链接，将其指向自己的服务端地址（即步骤1的服务端接口）。



```
<template>
```

```
<view class="content">
   <button type="default" @click="selectUpload">选择文件上传</button>
   <image v-if="fileUrl" class="image" :src="fileUrl"></image>
</view>
</template>

<script>
export default {
   data() {
      return {
         title: 'Hello',
         fileUrl: ''
      };
   },
   onLoad() {

   },
   methods: {
      selectUpload() {

         var vm = this;

         // 对更多字符编码的 url encode 格式
         var camSafeUrlEncode = function (str) {
            return encodeURIComponent(str)
                    .replace(/!/g, '%21')
                    .replace(/'/g, '%27')
                    .replace(/\(/g, '%28')
                    .replace(/\)/g, '%29')
                    .replace(/\*/g, '%2A');
         };

         // 获取上传路径、上传凭证L
         var getUploadInfo = function (extName, callback) {
            // 传入文件后缀，让后端生成随机的 COS 对象路径，并返回上传域名、PostObject 接口要
            // 参考服务端示例：https://github.com/tencentyun/cos-demo/tree/main/server
            uni.request({
               url: 'http://127.0.0.1:3000/post-policy?ext=' + extName,
               success: (res) => {
                  callback && callback(null, res.data.data);
               },
               error(err) {
                  callback && callback(err);
               },
            });
         };
```

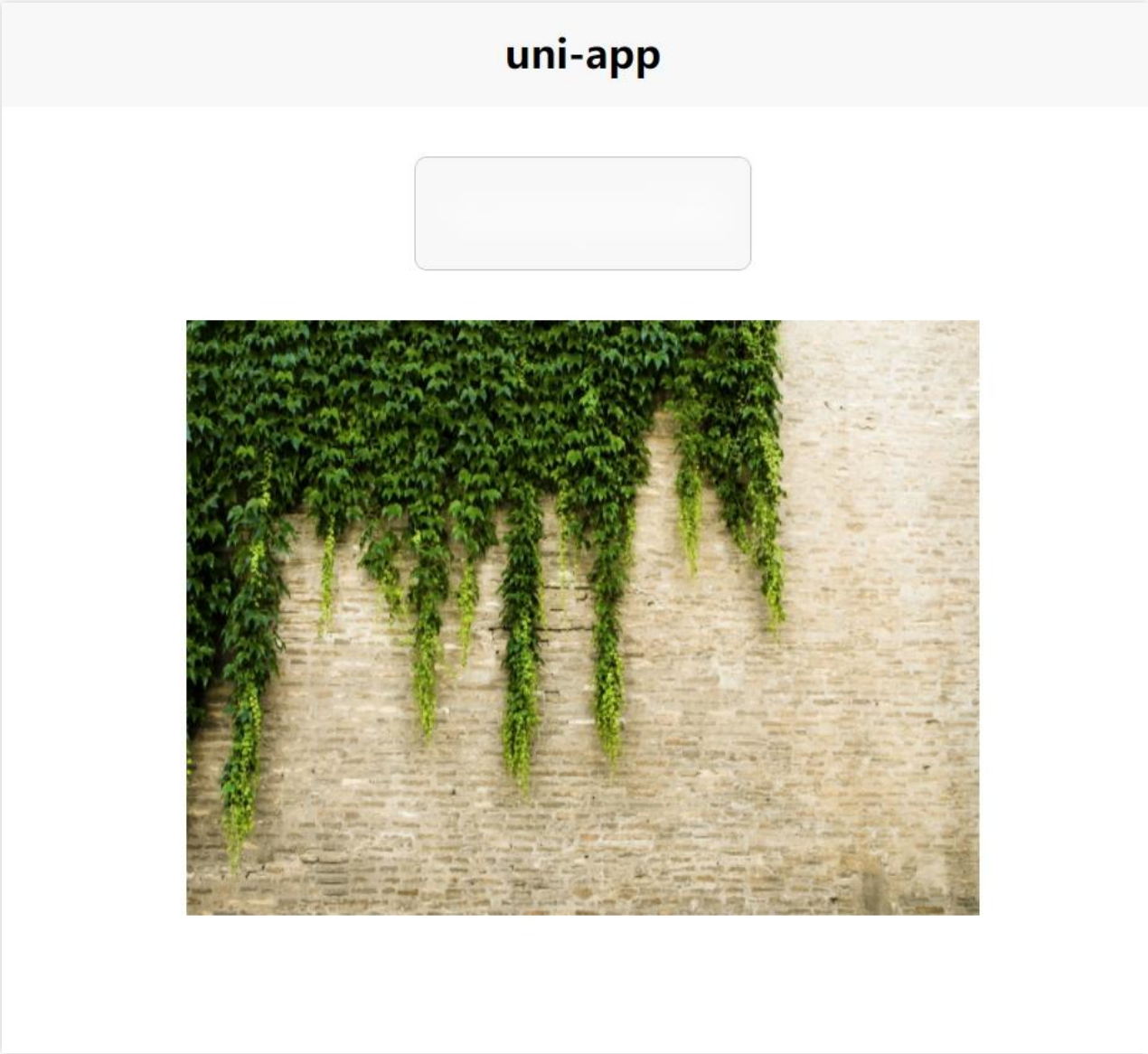
```
// 发起上传请求，上传使用 PostObject 接口，使用 policy 签名保护
// 接口文档：https://www.tencentcloud.com/document/product/436/14690#.E7.AD.
var uploadFile = function (opt, callback) {
    var formData = {
        key: opt.cosKey,
        policy: opt.policy, // 这个传 policy 的 base64 字符串
        success_action_status: 200,
        'q-sign-algorithm': opt.qSignAlgorithm,
        'q-ak': opt.qAk,
        'q-key-time': opt.qKeyTime,
        'q-signature': opt.qSignature,
    };
    // 如果服务端用了临时密钥计算，需要传 x-cos-security-token
    if (opt.securityToken) formData['x-cos-security-token'] = opt.securityT
    uni.uploadFile({
        url: 'https://' + opt.cosHost, //仅为示例，非真实的接口地址
        filePath: opt.filePath,
        name: 'file',
        formData: formData,
        success: (res) => {
            if (![200, 204].includes(res.statusCode)) return callback && call
            var fileUrl = 'https://' + opt.cosHost + '/' + camSafeUrlEncode(o
            callback && callback(null, fileUrl);
        },
        error(err) {
            callback && callback(err);
        },
    });
};

// 选择文件
uni.chooseImage({
    success: (chooseImageRes) => {
        var file = chooseImageRes.tempFiles[0];
        if (!file) return;
        // 获取要上传的本地文件路径
        var filePath = chooseImageRes.tempFilePaths[0];
        // 获取上传的文件后缀，然后后端生成随机 COS 路径地址
        var fileName = file.name;
        var lastIndex = fileName.lastIndexOf('.');
        var extName = lastIndex > -1 ? fileName.slice(lastIndex + 1) : '';
        // 获取预上传用的域名、路径、凭证
        getUploadInfo(extName, function (err, info) {
            // 上传文件
            info.filePath = filePath;
            uploadFile(info, function (err, fileUrl) {
                vm.fileUrl = fileUrl;
```

```
                    });
                });
            }
        });
    },
  }
}
</script>

<style>
.content {
   padding: 20px 0;
   display: flex;
   flex-direction: column;
   align-items: center;
   justify-content: center;
}

.image {
   margin-top: 20px;
   margin-left: auto;
   margin-right: auto;
}
</style>
```

4. 在 HBuilderX 上，选择**运行 > 运行到浏览器 > Chrome**，即可在浏览器选择文件进行上传。

执行效果如下图所示：

直传效果：



Demo 代码地址

示例 Demo 下载地址：upload-demo

内容审核实践
内容分发网络（CDN）场景

最近更新时间：2024-01-06 10:54:03

简介

内容审核功能提供了自动冻结能力，可以将违规文件自动进行冻结处理。但由于冻结只会处理对象存储（Cloud Object Storage，COS）源站的数据，对于使用了 CDN 场景的用户，CDN 上的缓存无法在第一时间处理。
本文提供了一种通过云函数和 API 网关处理的方式，帮助用户解决 CDN 缓存无法及时冻结的问题。

操作步骤

1. 登录 腾讯云云函数控制台，在**函数服务**页面，单击**新建**创建云函数。
2. 选择**从头开始**，并填写以下基础配置：
函数类型：选择**事件函数**。
函数名称：自定义一个函数名称。
地域：选择您使用了审核功能的存储桶所在的地域。
运行环境 选择**Python2.7**。
3. 函数代码按下述配置：
提交方法：选择**本地上传 zip 包**或者**通过 cos 上传 zip 包**，zip 包可 单击此处 进行下载。
执行方法：填 index.main_handler。
4. 单击**高级配置**，配置**环境配置**，其中除环境变量外，其余配置可自行按需修改。
资源类型：CPU
内存：512MB
初始化超时时间：65
执行超时时间：30
环境变量：
CI_AUDITING_CALLBACK：回调地址，这里填原本在内容审核的回调设置中的地址，设置回调地址则会进行回调，不设置不回调。
CDN_URL：CDN 地址，必选，设置 CDN 地址以刷新 CDN 缓存。
REGION：地域，必选，设置为 bucket 所在地域。
BUCKET_ID：存储桶 ID（即存储桶名称），必选，设置为存储桶 ID。存储桶 ID 用于查询图片样式，填写错误会导致样式查询出错。

IMAGE_STYLE_SEPARATORS：图片的样式分隔符，如果需要为图片刷新样式，则需要设置这个变量，多个分割符则连续写，不需要隔开。

CDN_REFRESH_TYPE：图片对象缓存刷新方式，默认为按 url 刷新，仅刷新样式，如设置为**path**，则会刷新按图片处理参数访问的缓存。注意，path 方式刷新会去除包含此文件名的更长的文件名的文件，请谨慎使用。

以上环境变量请确认正确填写，以确保缓存刷新符合预期，示例如下：

5. 权限配置勾选**运行角色**，单击**新建运行角色**，跳转到**新建自定义角色**页面。
6. 角色载体选择**云函数(scf)，单击下一步**。
7. 配置角色策略，选择 **QcloudCDNFullAccess** 和 **QcloudCIReadOnlyAccess**，单击**下一步**。
8. 为角色命名后，单击**完成**。
9. 自定义角色创建完毕后，回到云函数创建页面，刷新角色下拉表，选择刚才新建的角色。
10. 上述步骤完成后，单击**完成**，创建云函数。
11. 再进入 API 网关控制台，开通 API 网关服务。
12. 您可以参考 创建后端对接云函数 SCF 的 API 完成创建。
13. 在发布环境选择**发布**，然后单击**发布服务**。
14. 在数据万象**内容审核**页面，为图片或其他目标类型设置回调参数，回调 URL 设置为上一步创建的 API 网关的地址。
15. 回调设置完成后，CDN 上的资源将会自动根据审核回调结果进行缓存刷新。

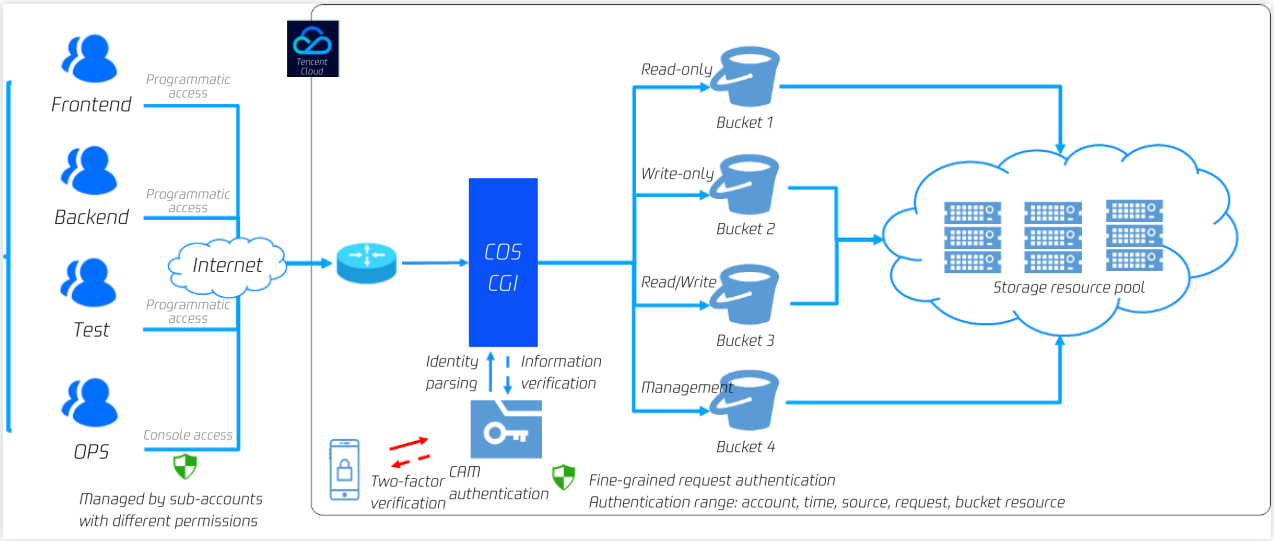
数据安全
数据安全概述

最近更新时间：2024-01-06 10:54:03

事前防护手段

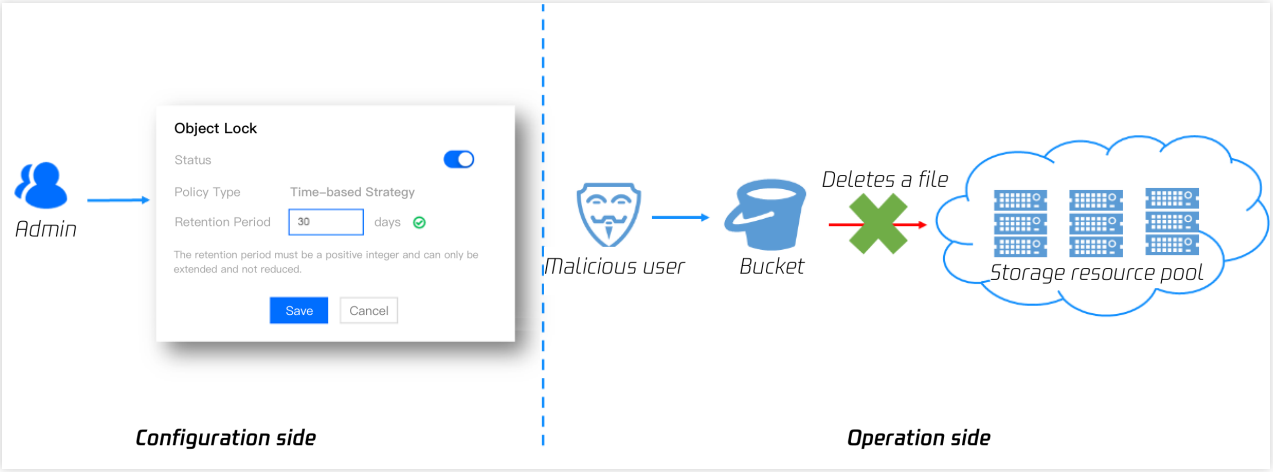
1. 权限隔离

对上云企业来说，账号安全和资源合理授权是构筑立体防护体系的第一道门锁。云上资源管理的授权应该规避如下风险：
使用腾讯云主账号进行日常操作；
为员工建立子账号，但授权过大；
对高权限子账号用户和高危操作没有访问条件控制；
没有定期审计用户的权限和登录信息；
缺乏权限的管理制度和流程。
腾讯云 CAM 通过账户分级、权限分级等多种措施保障权限清晰、安全可控。账户分级方面，主账号可以为所有合法的 CAM 用户，包括子账号、协作者等，授予编程访问和控制台访问等不同的访问权限；权限分级方面，则通过服务级、接口级、资源级等不同级别的授权，授权 CAM 用户**可以在哪种条件下，通过哪种方式对哪些资源进行哪种操作**。
例如，可以在主账号里创建子账号，给子账号分配主账号下资源的管理权限，而不需要分享主账号的相关的身份凭证。另外，可以针对不同的资源，授权给不同的人员拥有不同的访问权限。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主账号可以拥有某个 COS 对象的写权限等。这里的资源、访问权限、用户都可以批量打包，从而做到精细化的权限管理。
对于一些高危操作（如删除数据）的权限，也可剖离出来进行授权，仅允许用户在控制台进行操作，同时通过开启 MFA 校验来进行二次认证。开启 MFA 校验后，用户在执行此类高危操作时会触发短信校验码进行校验。



2. 对象锁定

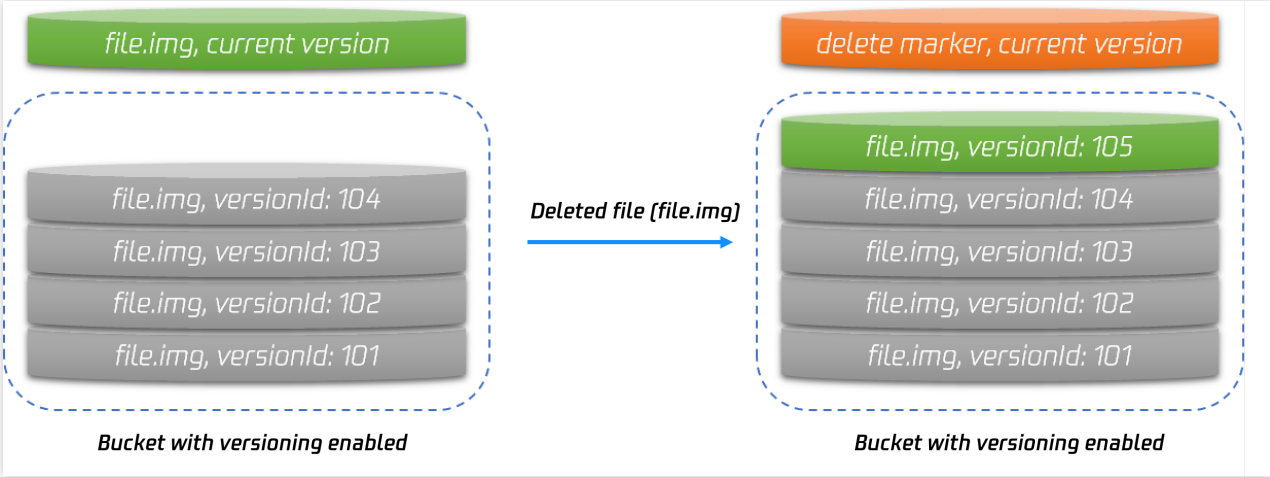
对于一些核心敏感数据，如金融交易数据、医疗影像数据等，可通过对象锁定功能来防止文件在上传之后被删除或者篡改。配置对象锁定功能后，在配置的有效期内，存储桶内的所有数据将处于只读状态，不可覆盖写或者删除；此项操作对包括主账号在内所有 CAM 用户及匿名用户生效。



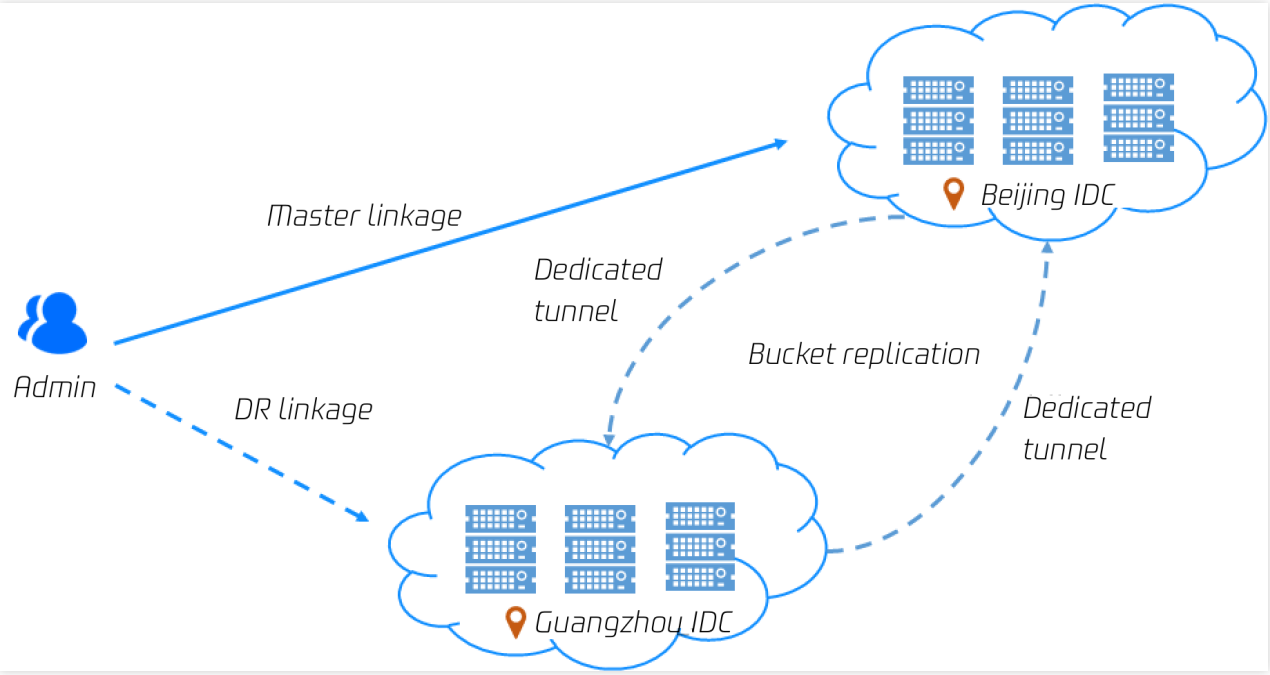
3. 数据灾备

腾讯云对象存储提供了包括数据加密、版本控制、存储桶复制和生命周期等多种功能在内的数据管理能力支持。敏感文件可通过加密功能保障数据读写安全；通过版本控制和存储桶复制实现异地容灾，进一步保证数据持久性，确保数据误删或者被恶意删除时可从备份站点恢复数据；通过生命周期进行数据沉降和删除，减少数据存储成本。

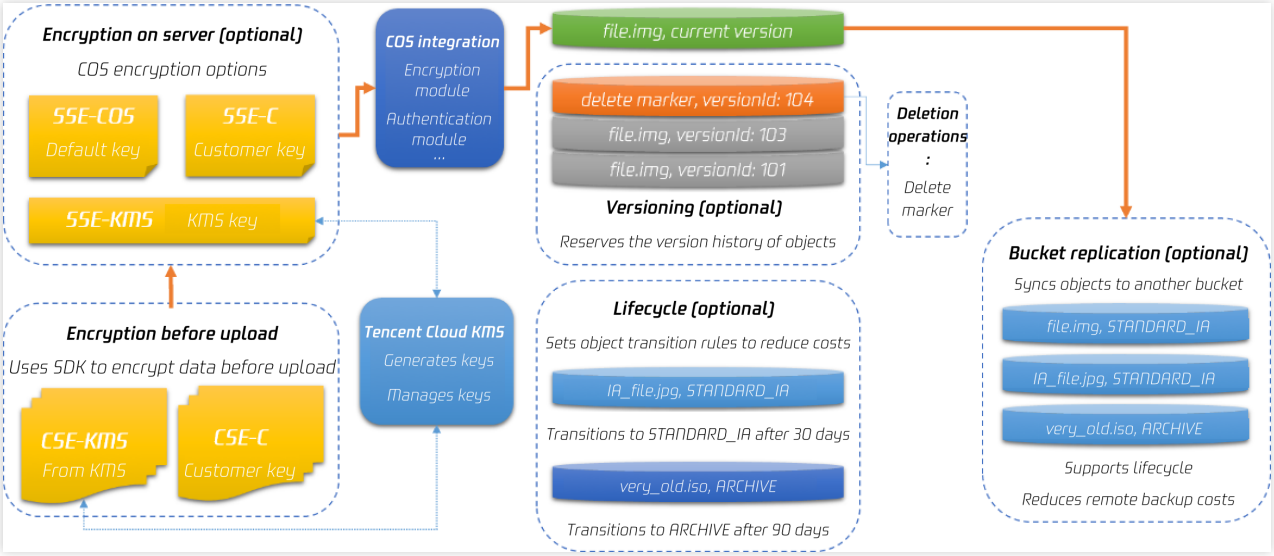
版本控制功能可以保障用户的文件不会被覆盖写或者删除。在开启版本控制配置后，所有同名文件的写操作都会视同新增不同版本的同名文件，删除操作等同于新增一项删除标记；可通过指定版本号访问过去任意版本的数据，可实现数据的回滚操作，解决数据误删和覆盖的风险。



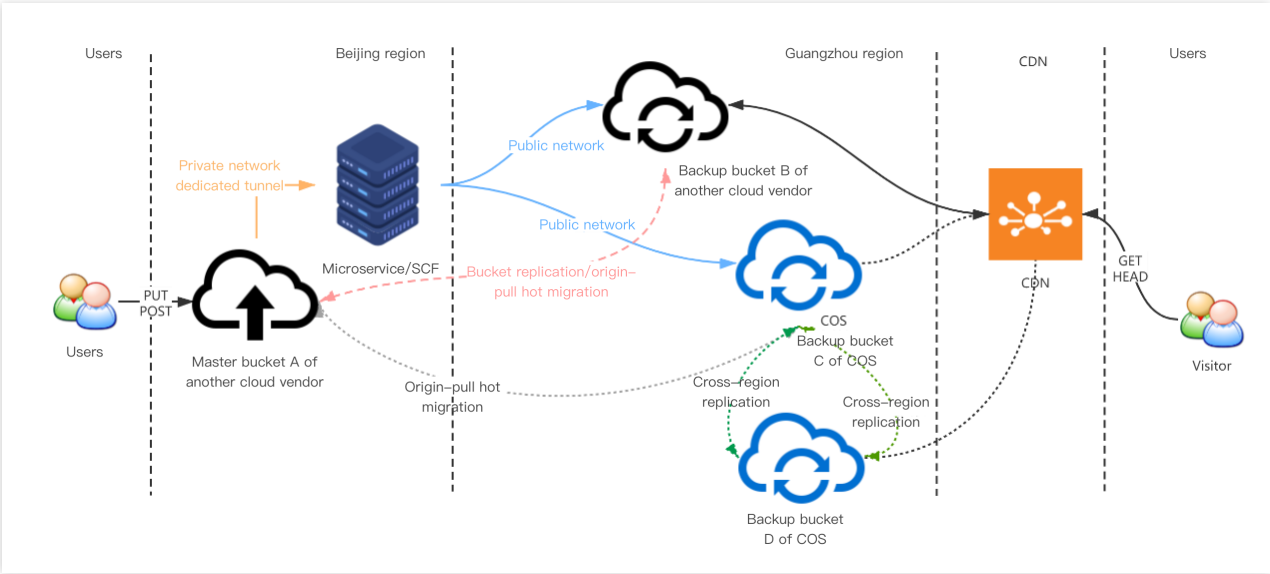
此外，对象存储还提供了存储桶复制的功能，帮助用户将所有增量文件通过专线复制到其他城市的数据中心，实现异地容灾的作用。当主存储桶中的数据被删除时，可从备份存储桶中通过批量拷贝的方式恢复数据。



考虑到版本控制和存储桶复制功能都可能造成文件数增加，用户也可以通过生命周期功能将一些备份数据沉降至低频或者归档存储等更便宜的存储类型，从而实现低成本冷备。综合数据加密、版本控制、存储桶复制和生命周期功能，腾讯云对象存储对外提供的完整冷备方案如下图所示。



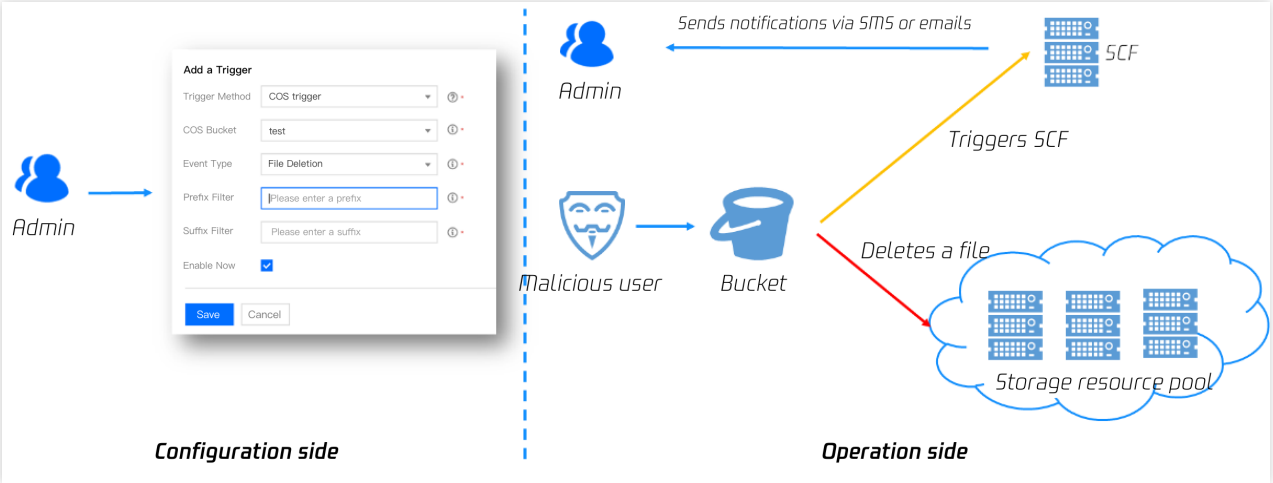
对于一些数据主要存储在其他云厂商，且对数据持久性要求苛刻的客户，COS 也提供基于云函数的多云灾备方案。首先数据存储在其他云厂商上（如 AWS 或者 OSS），客户可通过云函数触发数据同步或者存储桶复制实现异地容灾，保障数据持久性；同时，通过云函数触发数据迁移，将核心数据备份到腾讯云的对象存储服务上，并通过腾讯云的存储桶复制功能，实现异地灾备；最后，通过腾讯云的权限管控，管理 COS 的数据访问权限，保障极端情况下数据可从腾讯云 COS 上恢复数据。



事中监控手段

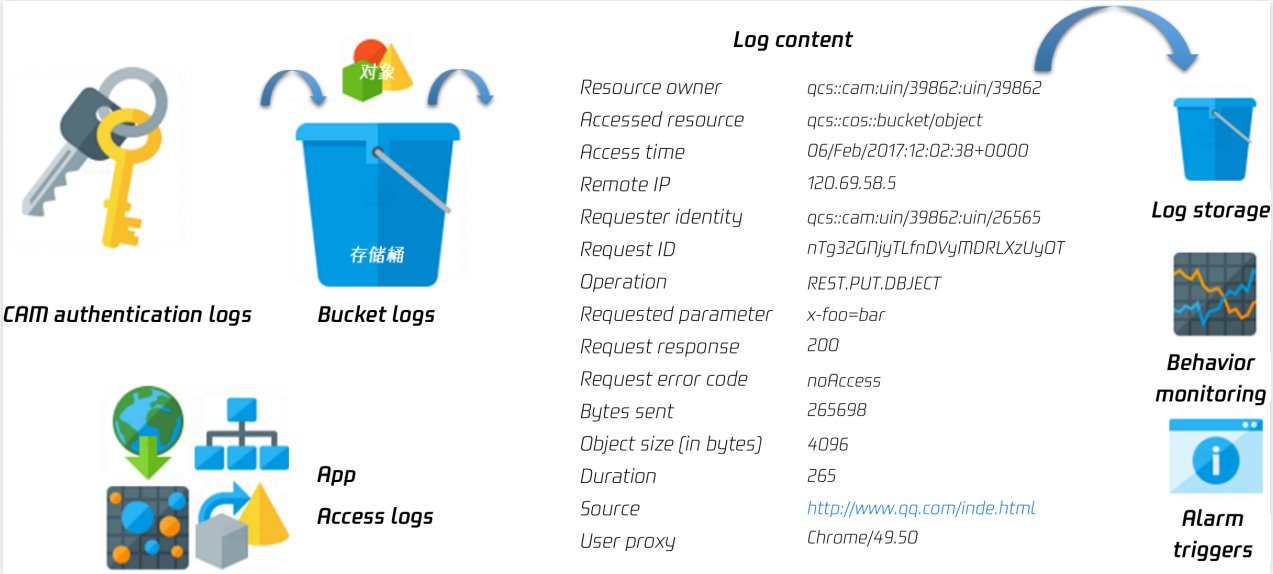
腾讯云对象存储基于云函数提供了事件通知功能。对于删除文件这类高危操作，可以通过 SCF 针对 DeleteObject 这类高危操作配置云函数，在高危操作行为发生时，即刻发送通知到邮件或者手机上，确保可以及时发现高危行为，

并采取手段中止。



事后追溯手段

腾讯云对象存储为用户提供了多渠道低门槛的日志监控和审计功能。对于存储桶的用户访问日志，如删除文件（DeleteObject）、覆盖写文件（PutObjectCopy）、修改文件权限（PutObjectACL）等操作，均可通过存储桶访问日志功能进行排查，删除操作等高危行为可追溯可查证；对于存储桶配置管理行为，如删除存储桶（DeleteBucket）、修改存储桶访问控制列表（PutBucketACL）、修改存储桶策略（PutBucketPolicy）等操作，可通过云审计日志进行排查，权限配置修改等行为也可追溯查证。



防盗刷指引

最近更新时间：2024-07-17 16:05:29

前言

近年来，越来越多的用户在搭建网站或图床时将图片视频等资源上传到对象存储 COS，提升了访问稳定性的同时减轻了服务器的存储空间压力，但随之而来的流量盗刷、图片盗链问题也困扰着不少开发者，一旦存储空间被恶意访问，会产生高额的流量费用，产生不必要的纠纷。这类问题实际上可以通过多种手段来防护，本文将主要介绍一些常见的防护手段，帮助开发者合理配置存储桶，建立安全机制，降低因类似问题带来的大额资金损失的风险。COS 控制台支持盗刷风险检测，请参见 盗刷风险检测 。

防护方案

盗刷有很多防护方式，这里将按配置难易度和门槛来区分，开发者可根据实际情况和需求来选择。

基础防护方案

修改存储桶访问权限

访问权限是存储桶最核心和敏感的配置之一，据不完全统计，绝大部分被盗刷的原因都是因为用户设置了**公有读**权限，由于公有读权限可不带签名直接匿名访问，这给了黑产和攻击者可乘之机。将存储桶改为私有读写可大大降低盗刷风险，拿不到密钥就无法计算签名，访问会被拒绝。
如果业务没有特别需求，这里建议您平时尽量配置**私有读写**权限。COS 控制台目前有两个地方可以设置存储桶权限：
创建存储桶弹窗

存储桶详情页-权限管理

开启存储桶防盗链

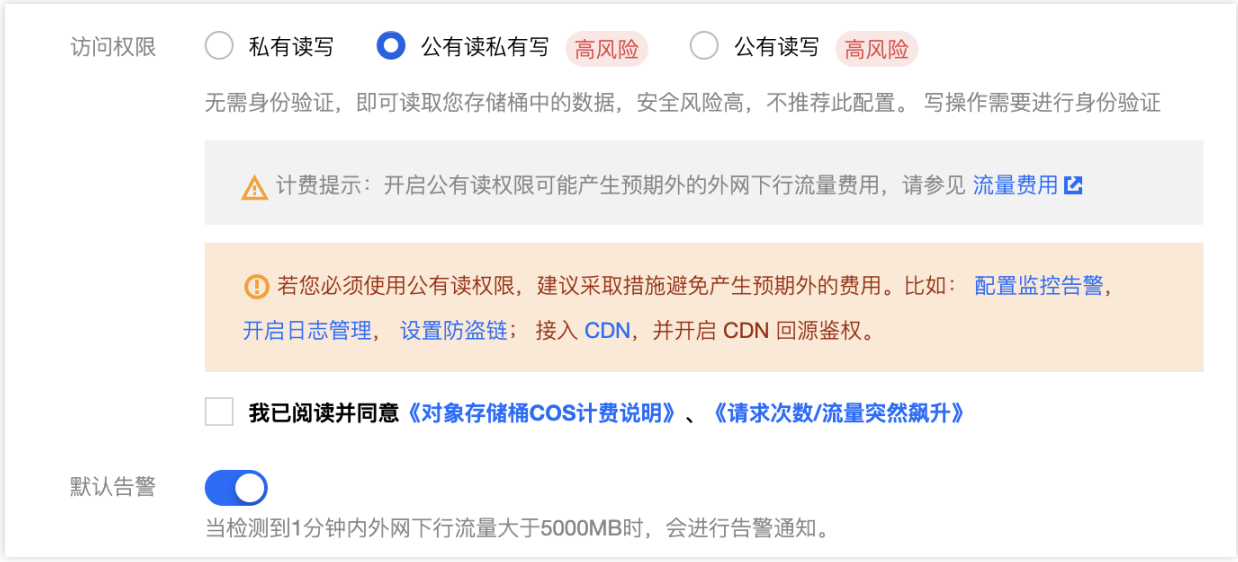
防盗链也是最常见的防护手段之一，其原理是通过 HTTP 的 Referer 头部进行判断校验，用户可通过配置白名单或黑名单，来允许或者禁止某些访问来源。

在 **COS 控制台-存储桶详情页-安全管理**可以找到防盗链设置：

这里建议您空 referer 配置为拒绝，黑白名单可根据业务实际情况来选择，如果是固定在某些域名下访问可设置白名单，如已经发现有恶意访问并明确知道访问域名或 IP，可手动设置黑名单来拦截。如需了解更多防盗链的使用技巧，可参考 防盗链实践。

配置云监控告警

配置日志管理能帮助定位和分析盗刷的来源，提供了丰富的字段可查询，但缺点是需要开发者主动去关注日志。如需更加及时的发现盗刷问题，可以配置云监控告警。目前 COS 控制台创建存储桶时，就支持同时配置告警：



如果默认告警的策略不能满足要求，也可以手动去 腾讯云可观测平台 创建自定义的告警策略：

进入**告警配置-告警策略**，点击**新建策略**，在云产品监控下找到对象存储，然后在告警对象的实例 ID 里找到需要配置的存储桶，就可以自己指定触发条件了，所有 COS 存储桶的数据监控页面里展示的指标，这里都支持配置。

开启日志管理

前面讲防盗链黑名单的时候提到过“如发现有恶意访问域名或 IP”，这里就需要我们开启存储桶的日志管理，访问日志里会记录每一次请求的各种字段，帮助我们快速定位到访问来源。
在 **COS 控制台-存储桶详情页-日志管理**可以找到日志存储，开启后即可在存储桶指定的路径前缀下找到访问日志。

日志文件里的字段可以参考 日志管理概述 文档，下面介绍一些常用的可分析字段：

userSecretKeyId：可以确定请求是通过哪个密钥 KeyId 进行的访问，如确定不是业务自己发起的请求，则很可能是密钥已经泄漏了，可前往 腾讯云CAM控制台 禁用不安全的密钥。

referer：即防盗链设置里用于判断的条件，如发现不认识的 referer，可能是被其他网站盗链，可配置防盗链-黑名单限制该 referer 访问，可参考1.2 开启存储桶防盗链。

remoteIp：可以确定访问来源 IP，如发现是不可信任的 IP，可前往**存储桶详情页-权限管理-Policy权限设置，**点击**添加策略，**配置 Policy 禁止该 IP 访问存储桶，示例如下：

进阶防护方案

使用自定义 CDN 加速域名

腾讯云 CDN 也提供了很多配置项来进行防护，也能有效抵御盗刷。
如我们的业务使用了自定义 CDN 域名，可以前往 腾讯云CDN控制台 **-域名详情-访问控制页面**进行配置，主要常用到的一些配置有：
防盗链配置：

CDN鉴权配置：

IP 黑白名单配置：

UA 黑白名单配置：

还有一些更为复杂的配置项，如 IP 访问限频配置、下行限速配置等，可查阅 CDN 控制台相关文档 来使用。

SCF+云监控+COS API实现自动封堵

在基础防护里，我们提到了可以配置云监控告警来及时发现流量异常情况，但有些时候可能会有信息遗漏或人在外不方便及时处理的情况，这里可以通过自动化脚本代码调用 API 来实现一个简单的自动化逻辑。
从云监控获取到外网下行流量指标（InternetTraffic）异常 -> 自动调用 COS PutBucketAcl 修改存储桶权限为私有读写。主要涉及以下2个 API 接口，这里提供了在线调用示例供调试参考：
云监控的 GetMonitorData - 调用示例
COS 的 PutBucketAcl - 调用示例

相关建议

以上为本文提供的一些盗刷问题的常见防护手段，除此之外，我们在日常使用 COS 时也需要注意一些小细节，这里提供一些额外的使用建议，也能一定程度上降低盗刷风险：
避免在对外开源的代码里使用明文的 API 访问密钥，避免密钥泄露引发安全风险，建议参考**最小权限原则**来使用。
避免配置跨域规则时允许所有来源（即Origin: *）访问，尽量设置明确的来源。
上传大量文件时，避免使用规律过于简单的顺序前缀（如数字序号，时间戳等），这样可能会导致攻击者更容易遍历到存储桶下的文件。

防盗链实践

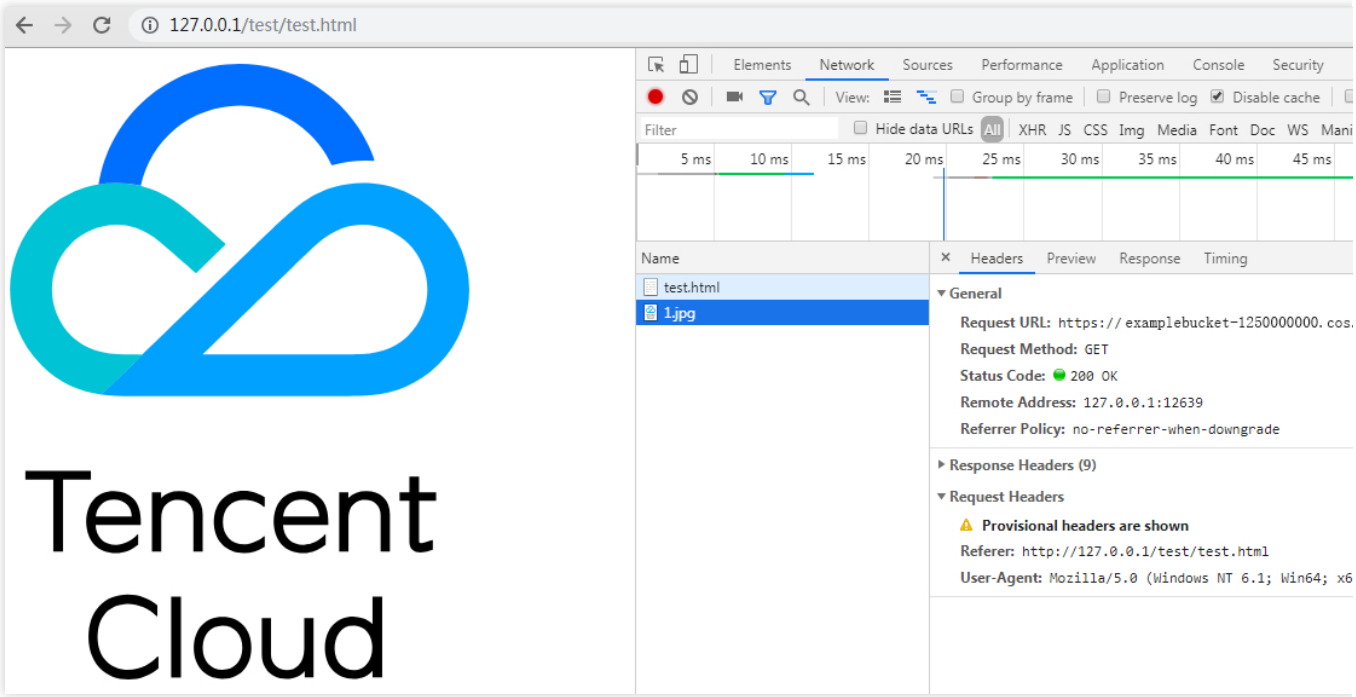
最近更新时间：2024-01-06 10:54:03

简介

腾讯云对象存储支持防盗链配置，用户可以对存储桶设置防盗链功能，该功能可以实现对访问来源设置黑、白名单，避免资源被盗用。本文为您详细介绍如何为存储桶配置防盗链，防止资源被盗用。

防盗链判断原理

防盗链是通过请求 Header 里的 Referer 地址来进行判断：
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 `https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/1.jpg`，请求 Header 里不会带有 Referer。
例如，下图是在 `https://127.0.0.1/test/test.html` 嵌入了图片 `1.jpg`，访问 `https://127.0.0.1/test/test.html` 时就带有 Referer 指向访问来源：

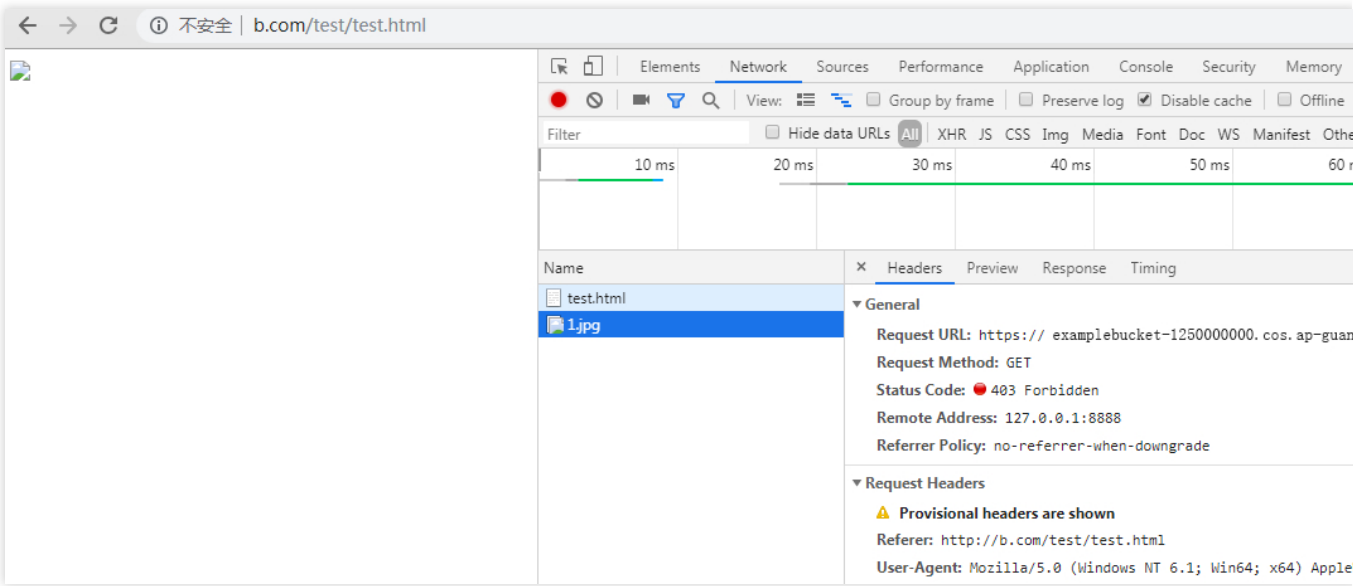


盗链案例分析

用户 A 在 COS 上传了图片资源 `1.jpg` ，得到图片的可访问链接为 `https://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/1.jpg` 。
用户 A 将该图片嵌入到自己的网页 `https://example.com/index.html` 上，图片能正常访问。
用户 B 在用户 A 网页上看到了该图片，决定将该图片嵌入在他自己的网页 `https://b.com/test/test.html` 上，此时用户 B 的网页也能正常显示该图片。
以上案例中，用户 A 的图片资源 `1.jpg` 就被用户 B 盗链了。此时用户 A 在不知情的情况下，COS 上的资源持续被用户 B 网页正常使用，用户 A 负担了额外的流量费用，造成了费用损失。

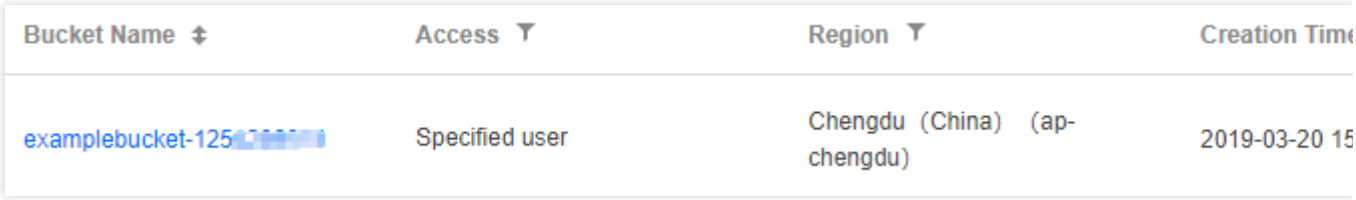
解决方式

根据以上 盗链案例分析 ，用户 A 可以通过防盗链设置防止用户 B 盗链图片，具体方法如下：
1. 用户 A 给存储桶 examplebucket-1250000000 设置防盗链规则，有两种方式可以防止用户 B 盗链：
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 `*.example.com` 并保存生效。
2. 开启了防盗链配置之后：
访问 `https://example.com/index.html` 图片显示正常。
访问 `https://b.com/test/test.html` 图片无法显示，表现如下图。

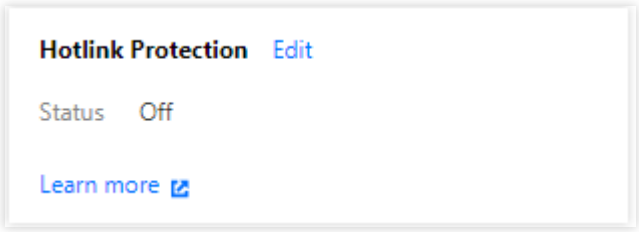


具体步骤

1. 登录 对象存储控制台，在左侧导航栏中单击【存储桶列表】，进入存储桶列表页。
2. 选择需要设置防盗链的存储桶，进入存储桶。



3. 在左侧菜单栏中单击【安全管理】>【防盗链设置】，进入存储桶防盗链设置的配置页。
4. 在“防盗链设置”栏中，单击【编辑】，进入编辑状态。



5. 开启防盗链，并配置名单类型和域名，此处选择开启方式二，详细说明如下：

类型：有黑、白名单两种：

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。

Referer ：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。配置规则说明和示例如下：

支持带端口的域名和 IP，如 `example.com:8080` 、 `10.10.10.10:8080` 等地址。

配置 `example.com` ，可命中如 `example.com/123` 等以 `example.com` 为前缀的地址。

配置 `example.com` ，可命中如 `https://example.com` 和 `http://example.com` 为前缀的地址。

配置 `example.com` ，可命中它的带端口域名 `example.com:8080` 。

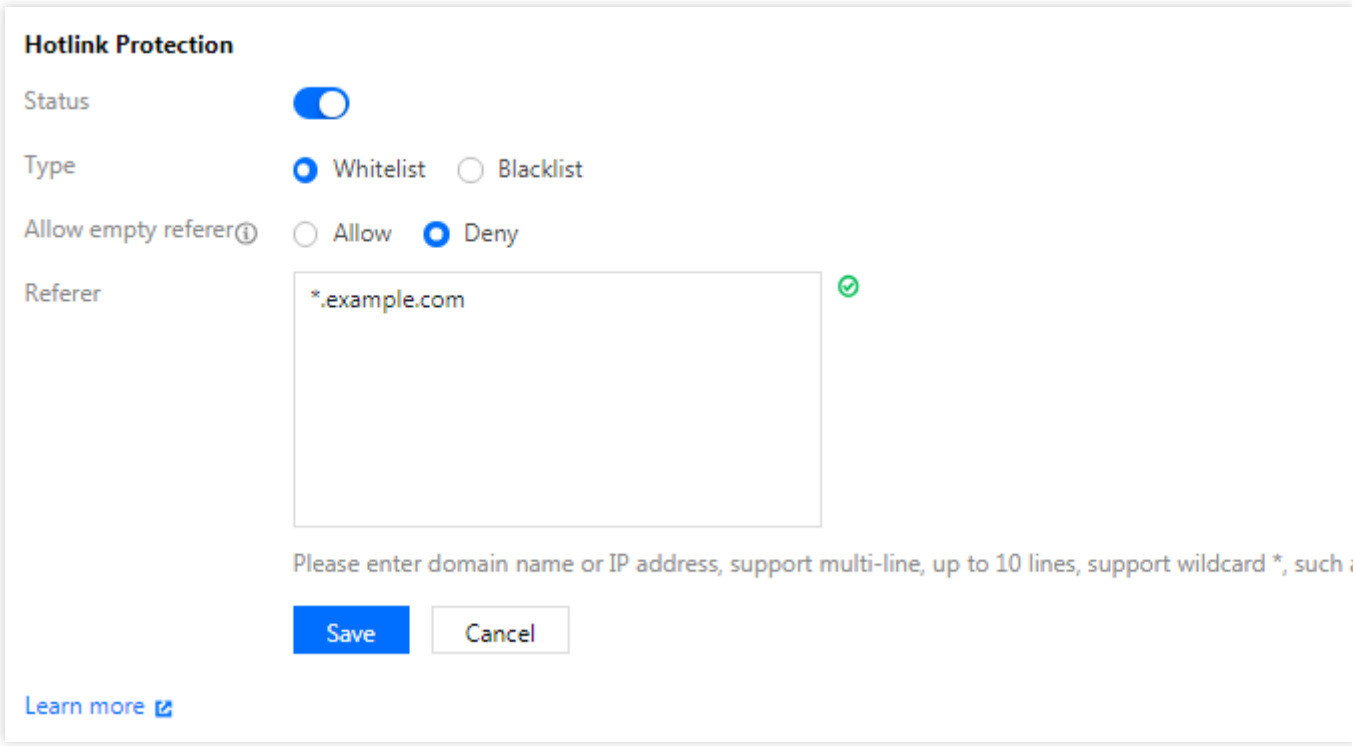
配置 `example.com:8080` ，不会命中域名 `example.com` 。

配置 `*.example.com` ，可限制它的二级、三级域名 `example.com` 、 `b.example.com` 、 `a.b.example.com` 。

注意

用户设置防盗链状态为**开启**后，必须填入相应的域名。

6. 配置完成之后，单击【保存】即可。



常见问题

关于防盗链的相关的疑问，可前往 COS 常见问题中的 数据安全 文档寻求解答。

数据校验
MD5 校验

最近更新时间：2024-01-06 10:54:03

简介

数据在客户端和服务器间传输时可能会出现错误，COS 可以通过 MD5 校验的方式保证上传数据的完整性，只有当 COS 服务器接收到的数据 MD5 校验值与用户设置的 MD5 校验值一致时，数据才可上传成功。
COS 里每个对象对应一个 ETag，ETag 是对象被创建时对象内容的信息标识，但 ETag 不一定等同于对象内容的 MD5 校验值，因此不能通过 ETag 来校验下载对象与原对象是否一致，但用户可使用自定义对象元数据（x-cos-meta-*）来实现下载对象与原对象的一致性校验。

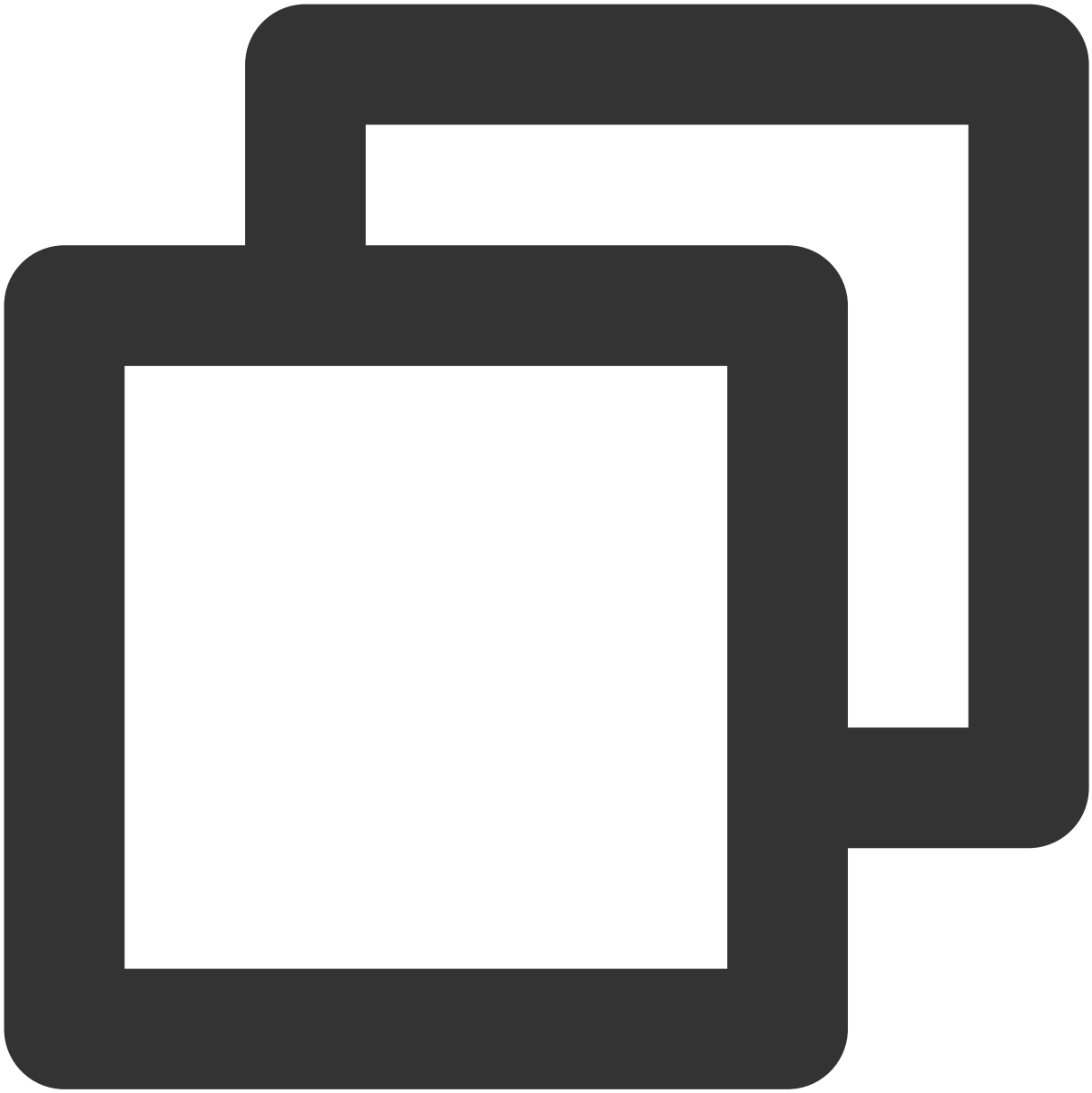
数据校验方式

校验上传对象
如果用户需要校验上传到 COS 的对象与本地对象是否一致，可以在上传时将 HTTP 请求的 Content-MD5 字段设置为经过 Base64 编码的对象内容 MD5 校验值，此时 COS 服务器将校验用户上传的对象，只有当 COS 服务器接收到的对象 MD5 校验值与用户设置的 Content-MD5 一致时，对象才可上传成功。
校验下载对象
如果用户需要校验下载对象与原对象是否一致，可在对象上传时使用校验算法计算对象的校验值，通过自定义元数据设置对象的校验值，在下载对象后，用户重新计算对象的校验值，并与该自定义元数据进行比较验证。在这种方式下，用户可自主选择校验算法，但对于同一个对象而言，其上传和下载时所使用的校验算法应保持一致。

API 接口示例

简单上传请求

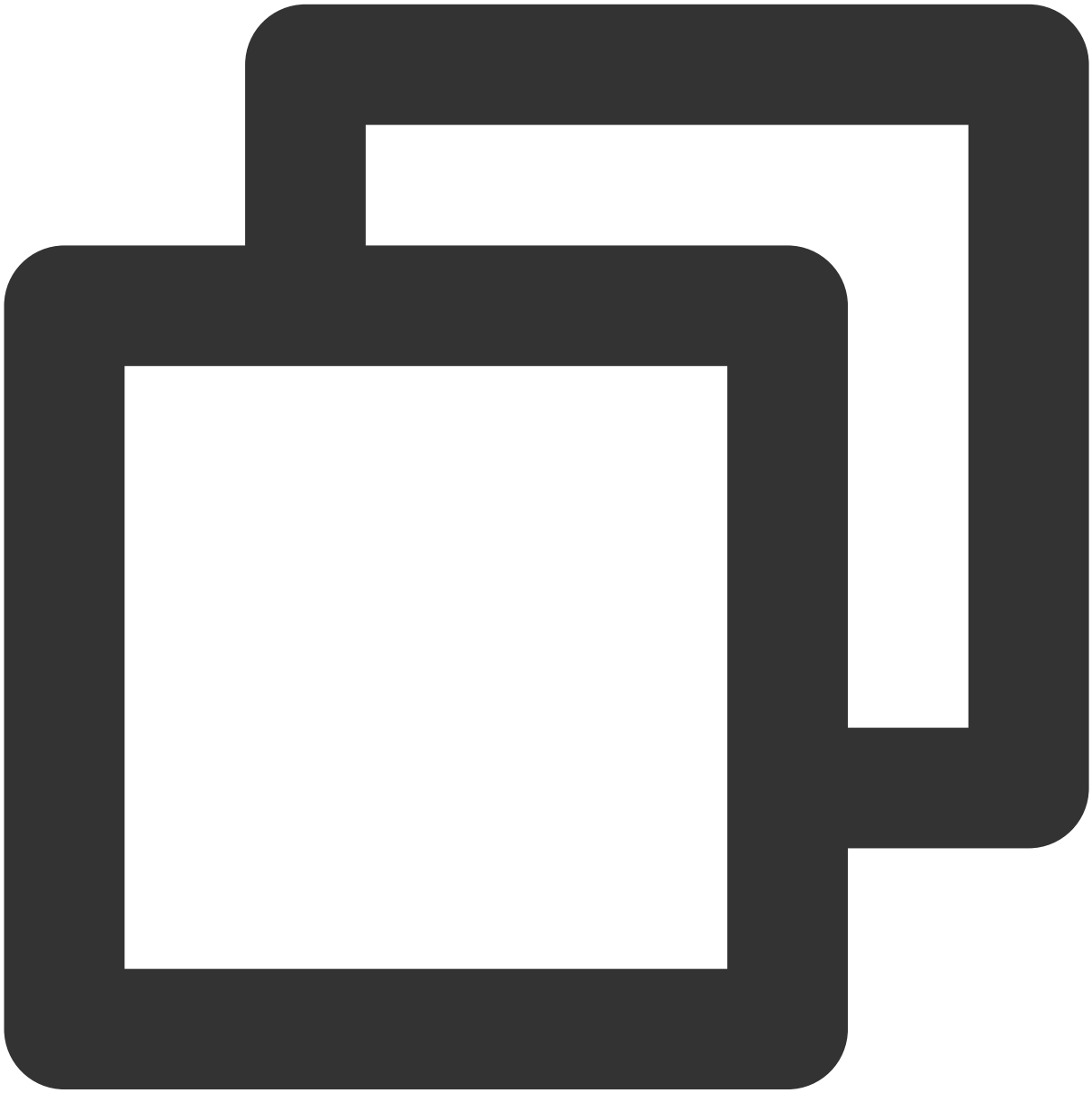
下面为用户上传对象的请求示例。上传对象时，设置 Content-MD5 为经过 Base64 编码的对象内容 MD5 校验值，此时只有当 COS 服务器接收到的对象 MD5 校验值与用户设置的 Content-MD5 一致时，对象才可上传成功，并且将自定义元数据 x-cos-meta-md5 设置为对象的校验值。
说明
示例演示是通过 MD5 校验算法得到对象的校验值，用户可自主选择其他的校验算法。



```
PUT /exampleobject HTTP/1.1
Host: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
Date: Fri, 21 Jun 2019 09:24:28 GMT
Content-Type: image/jpeg
Content-Length: 13
Content-MD5: ti4QvKtVqIJAvZxDbP/c+Q==
Authorization: q-sign-algorithm=sha1&q-ak=AKID8A0fBVtYFrNm02oY1g1JQQF0c3JO****&q-si
x-cos-meta-md5: b62e10bcab55a88240bd9c436cffdcf9
Connection: close

[Object Content]
```

分块上传请求

下面为初始化分块上传的请求示例。在上传对象分块时，用户可通过初始化分块上传来设置对象的自定义元数据，这里将自定义元数据 x-cos-meta-md5 设置为对象的校验值。



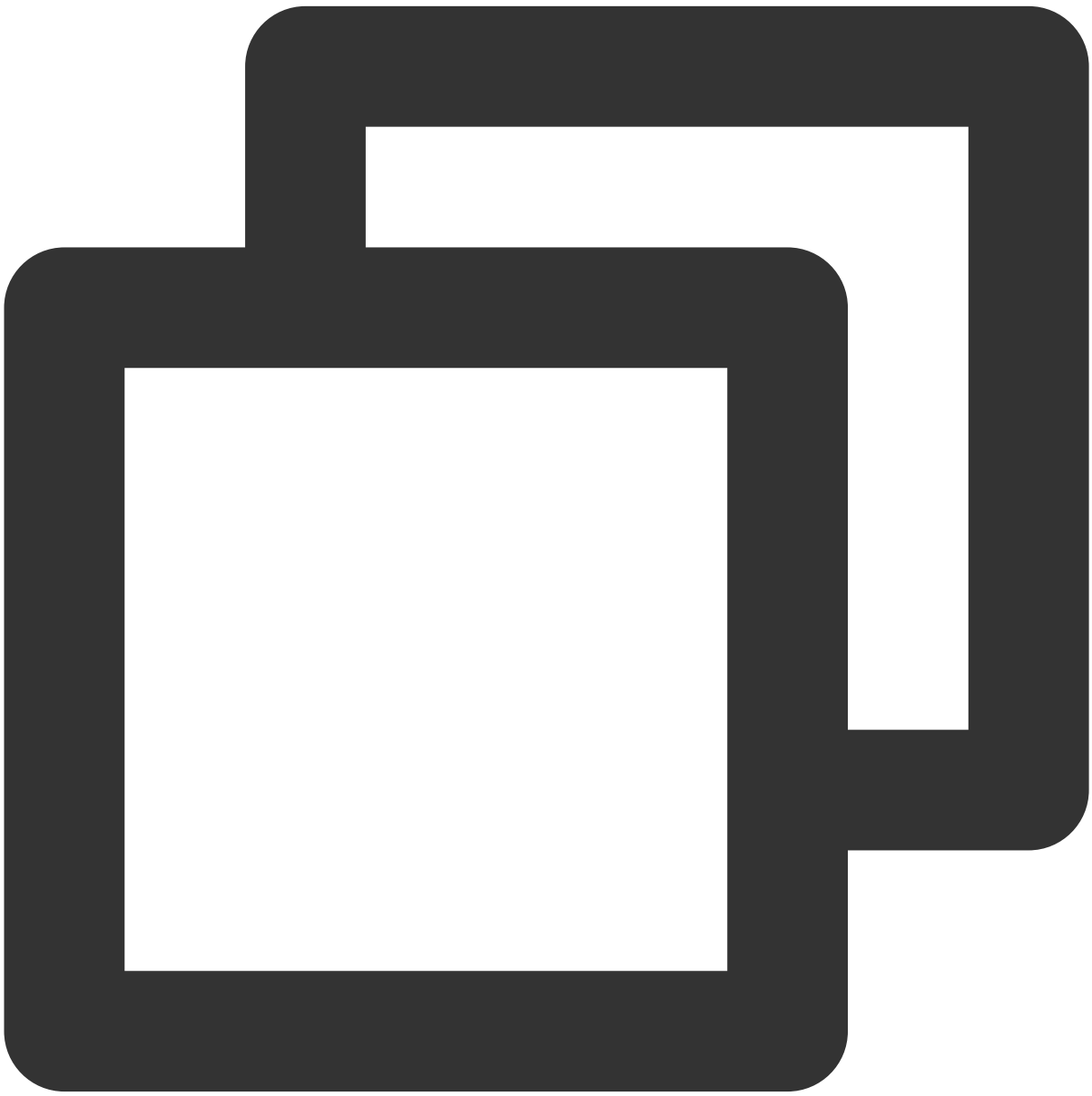
```
POST /exampleobject?uploads HTTP/1.1
Host: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
Date: Fri, 21 Jun 2019 09:45:12 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKID8A0fBVtYFrNm02oY1g1JQQF0c3JO****&q-si
x-cos-meta-md5: b62e10bcab55a88240bd9c436cffdcf9
```

注意

对于分块上传的文件，COS 只会校验每个分块的 MD5 值，而不会计算合并后完整文件的 MD5 值。

对象下载响应

下面为用户发出下载对象请求后得到的响应示例。从响应中用户可以得到对象的自定义元数据 x-cos-meta-md5，用户可通过重新计算对象的校验值，与该自定义元数据进行比较，从而验证下载对象和原对象是否一致。



```
HTTP/1.1 200 OK
Content-Type: application/octet-stream
Content-Length: 13
```

```
Connection: close
Accept-Ranges: bytes
Cache-Control: max-age=86400
Content-Disposition: attachment; filename=example.jpg
Date: Thu, 04 Jul 2019 11:33:00 GMT
ETag: "b62e10bcab55a88240bd9c436cffdcf9"
Last-Modified: Thu, 04 Jul 2019 11:32:55 GMT
Server: tencent-cos
x-cos-request-id: NWQxZGUzZWNfNjI4NWQ2NF9lMWYyXzk1NjFj****
x-cos-meta-md5: b62e10bcab55a88240bd9c436cffdcf9

[Object Content]
```

SDK 示例

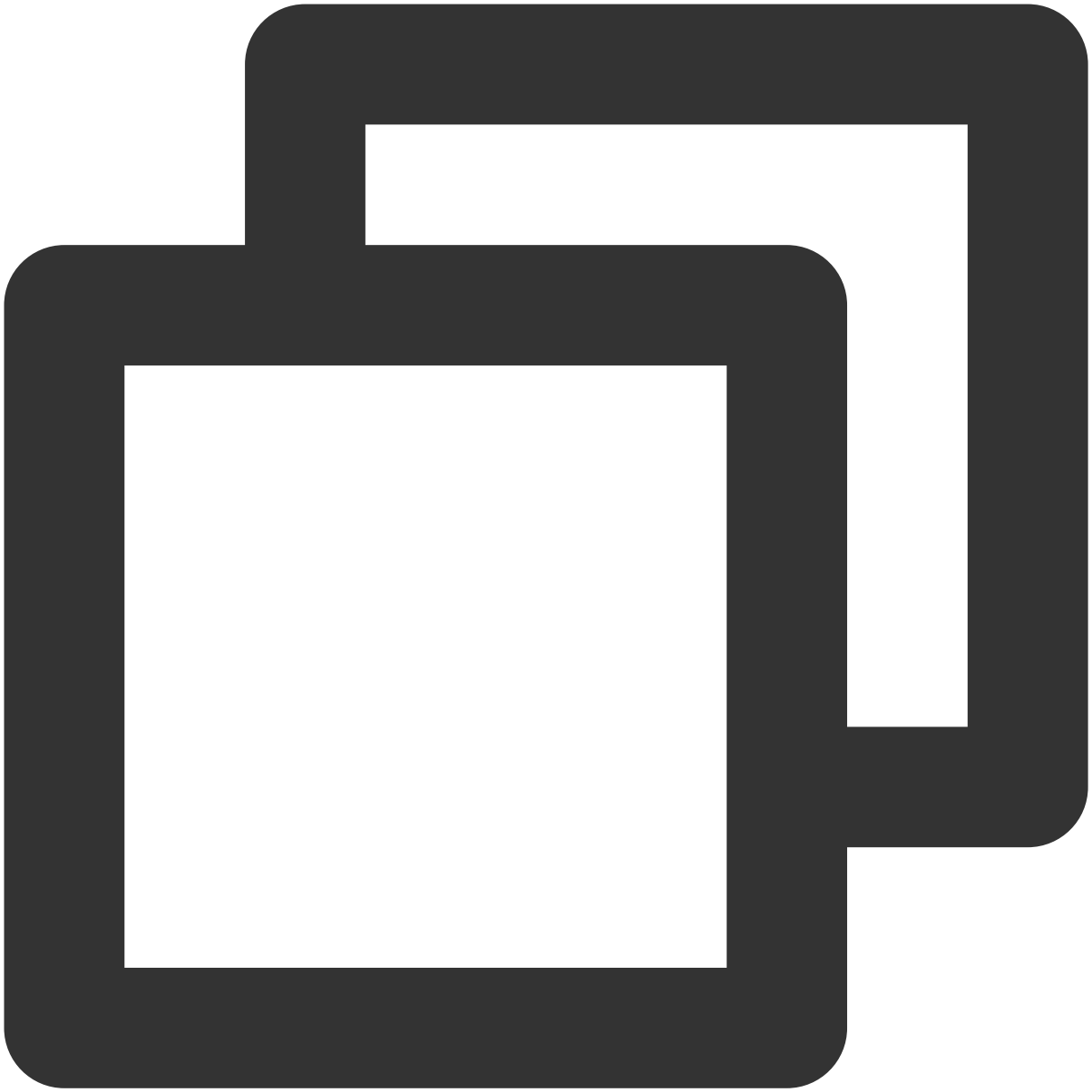
下面以 Python SDK 为例演示如何校验对象，完整的示例代码如下。

说明

代码基于 Python 2.7，其中 Python SDK 详细使用方式，请参见 Python SDK 对象操作 文档。

1. 初始化配置

设置用户属性，包括 SecretId、SecretKey 和 Region，并创建客户端对象。



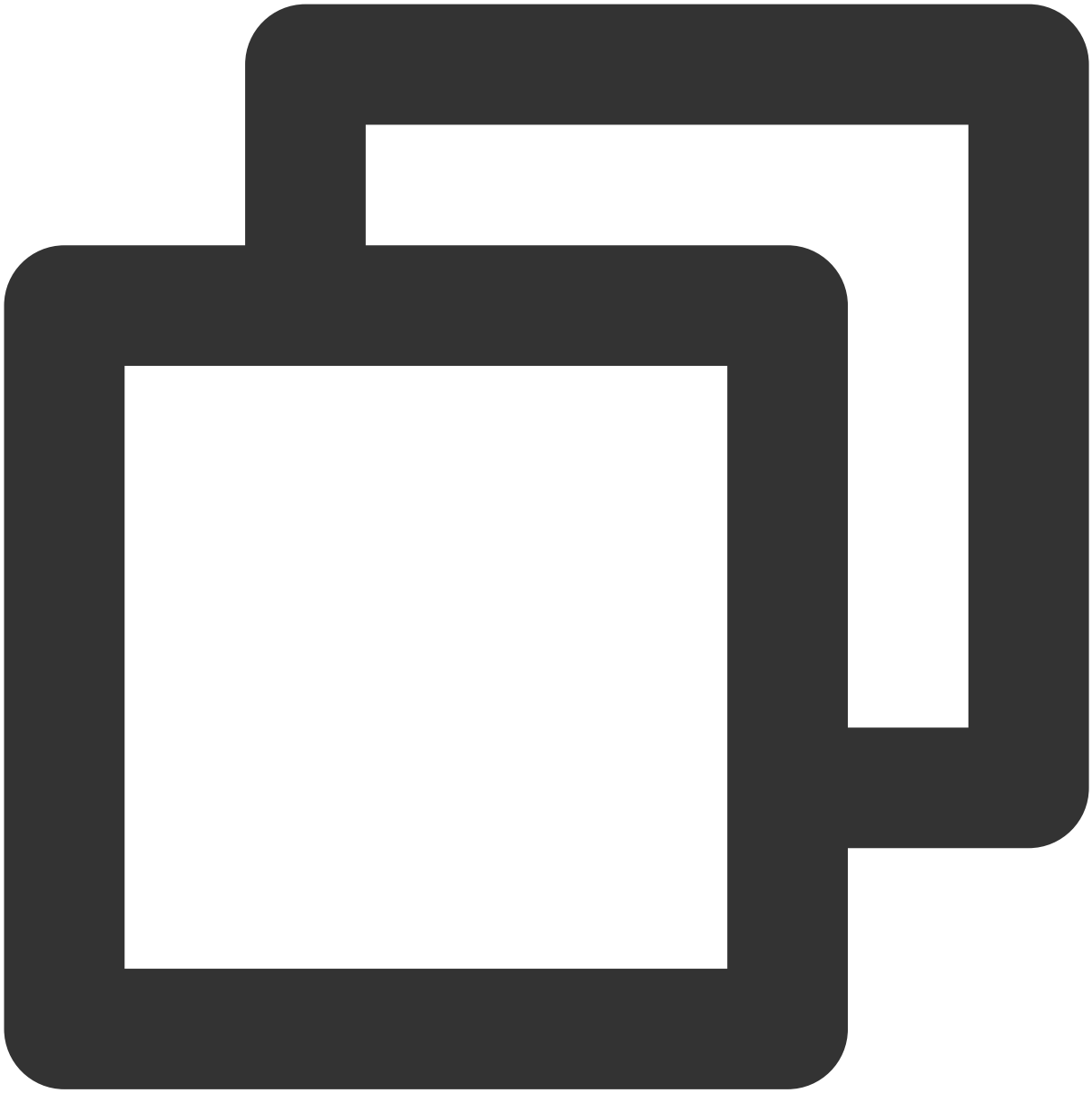
```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
from qcloud_cos import CosServiceError
from qcloud_cos import CosClientError
import sys
import os
import logging
import hashlib

logging.basicConfig(level=logging.INFO, stream=sys.stdout)
```

```
# 设置用户属性，包括 SecretId，SecretKey，Region
# APPID 已在配置中移除，请在参数 Bucket 中带上 APPID。Bucket 由 BucketName-APPID 组成。
secret_id = os.environ['COS_SECRET_ID']     # 用户的 SecretId，建议使用子账号密钥，授权遵
secret_key = os.environ['COS_SECRET_KEY']   # 用户的 SecretKey，建议使用子账号密钥，授权
region = 'ap-beijing'      # 替换为您的 Region，这里以北京为例
token = None               # 临时密钥的 Token，临时密钥生成和使用指引参见 https://cloud.te
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=t
client = CosS3Client(config)
```

2. 校验简单上传对象

（1）计算对象的校验值

通过 MD5 校验算法得到对象的校验值，用户可自主选择其他的校验算法。

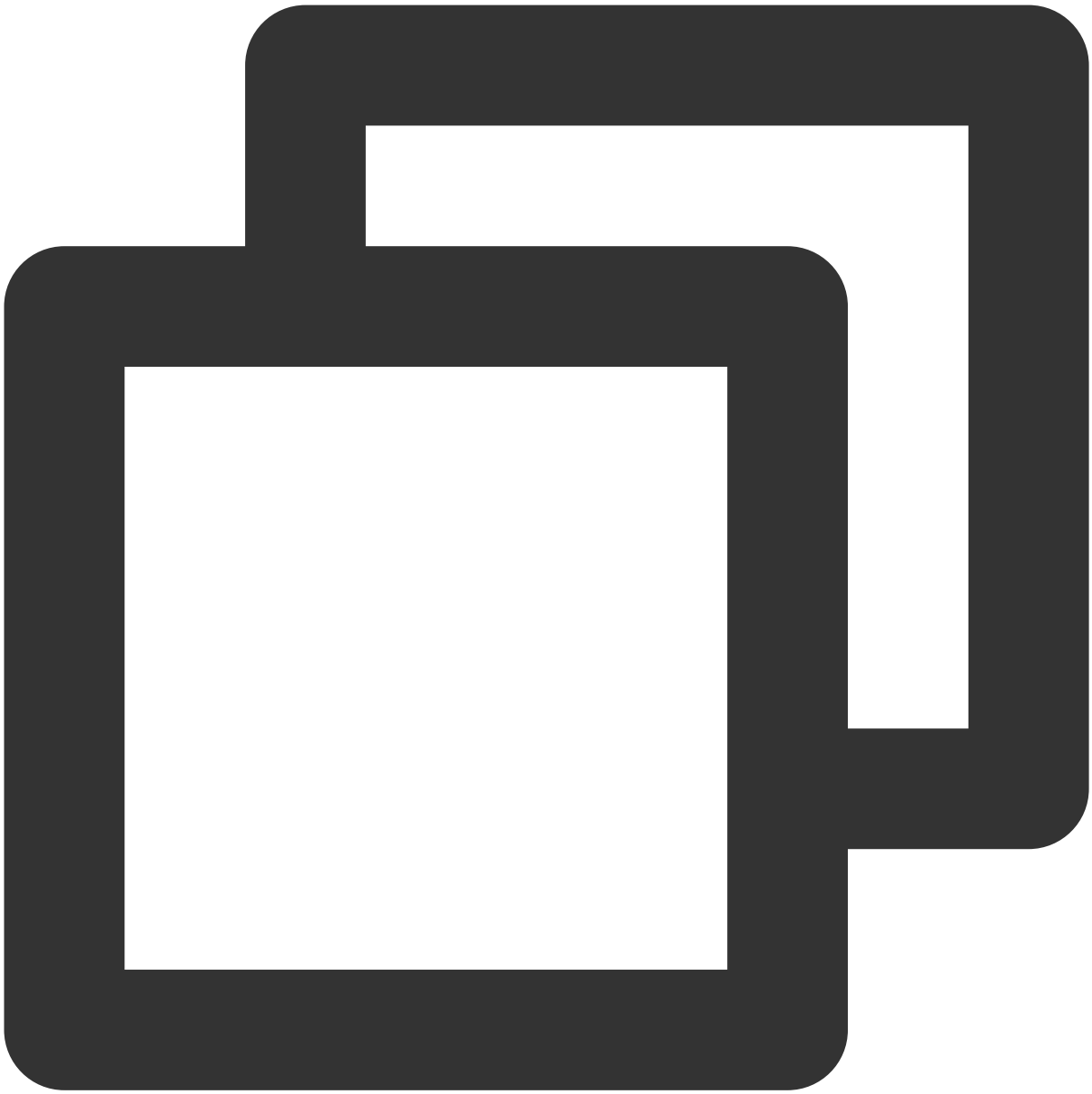


```
object_body = 'hello cos'
#得到对象的 md5 校验值
md5 = hashlib.md5()
md5.update(object_body)
md5_str = md5.hexdigest()
```

（2）简单上传对象

代码中 EnableMD5=True 表示开启对象上传的 MD5 校验，Python SDK 会计算 Content-MD5，打开后将增加上传耗时，当 COS 服务器接收到的对象 MD5 校验值与 Content-MD5 一致时，对象才可上传成功。

x-cos-meta-md5 为用户自定义的参数（自定义参数名称格式为 x-cos-meta-*），这里参数表示了对象的 MD5 校验值。

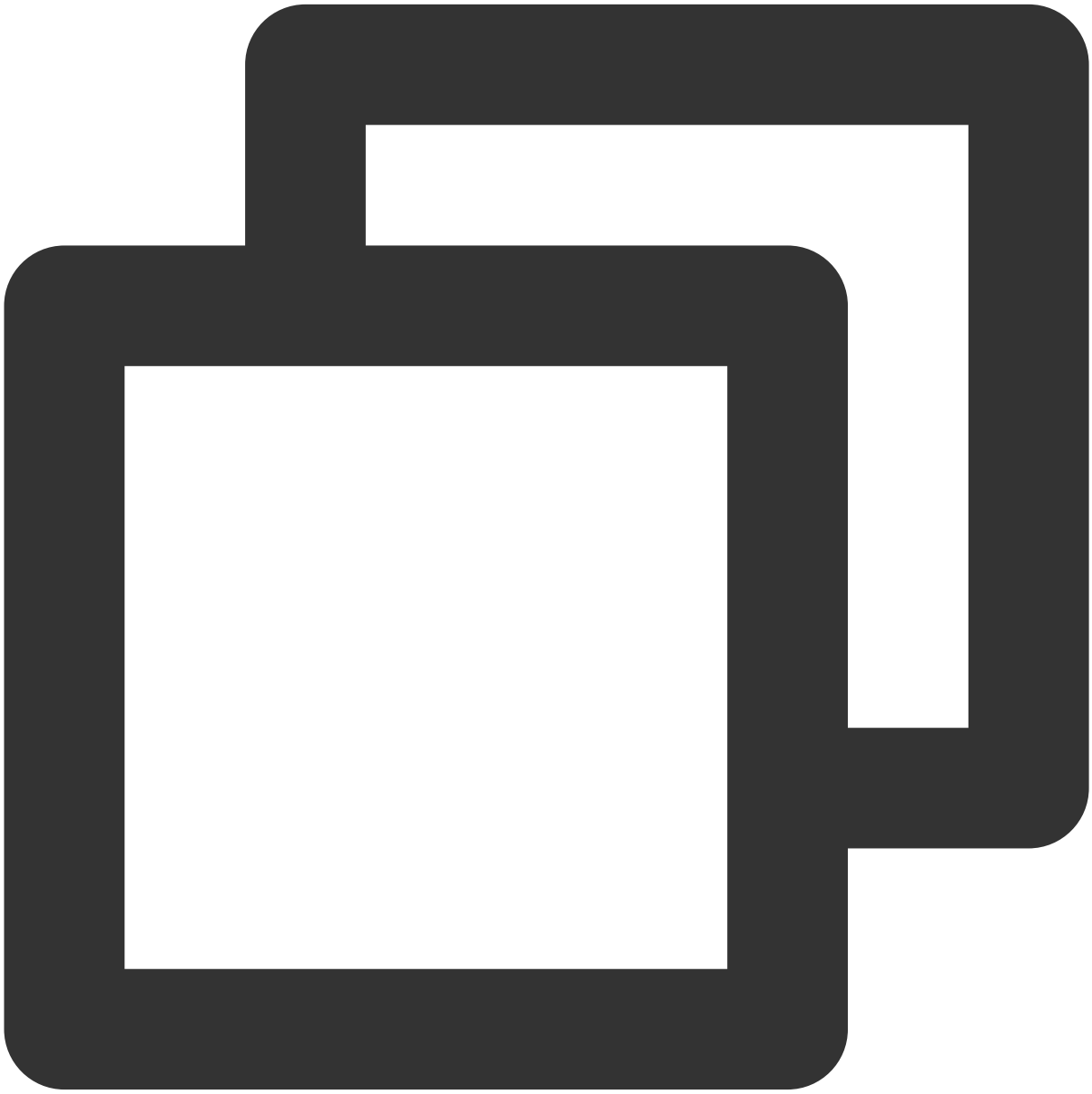


```
#简单上传对象，并打开 MD5 校验
response = client.put_object(
    Bucket='examplebucket-1250000000',       #替换为您的 Bucket 名称，examplebucket 是一
    Body='hello cos',                   #上传的对象内容
    Key='example-object-1',             #替换为您上传的对象 Key 值
    EnableMD5=True,                     #开启上传的 MD5 校验
    Metadata={                          #设置自定义参数，将对象的 MD5 校验值作为参数值存入 C
        'x-cos-meta-md5' : md5_str
```

```
    }
)
print 'ETag: ' + response['ETag']      # Object 的 Etag 值
```

（3）下载对象

下载对象并得到用户自定义参数。

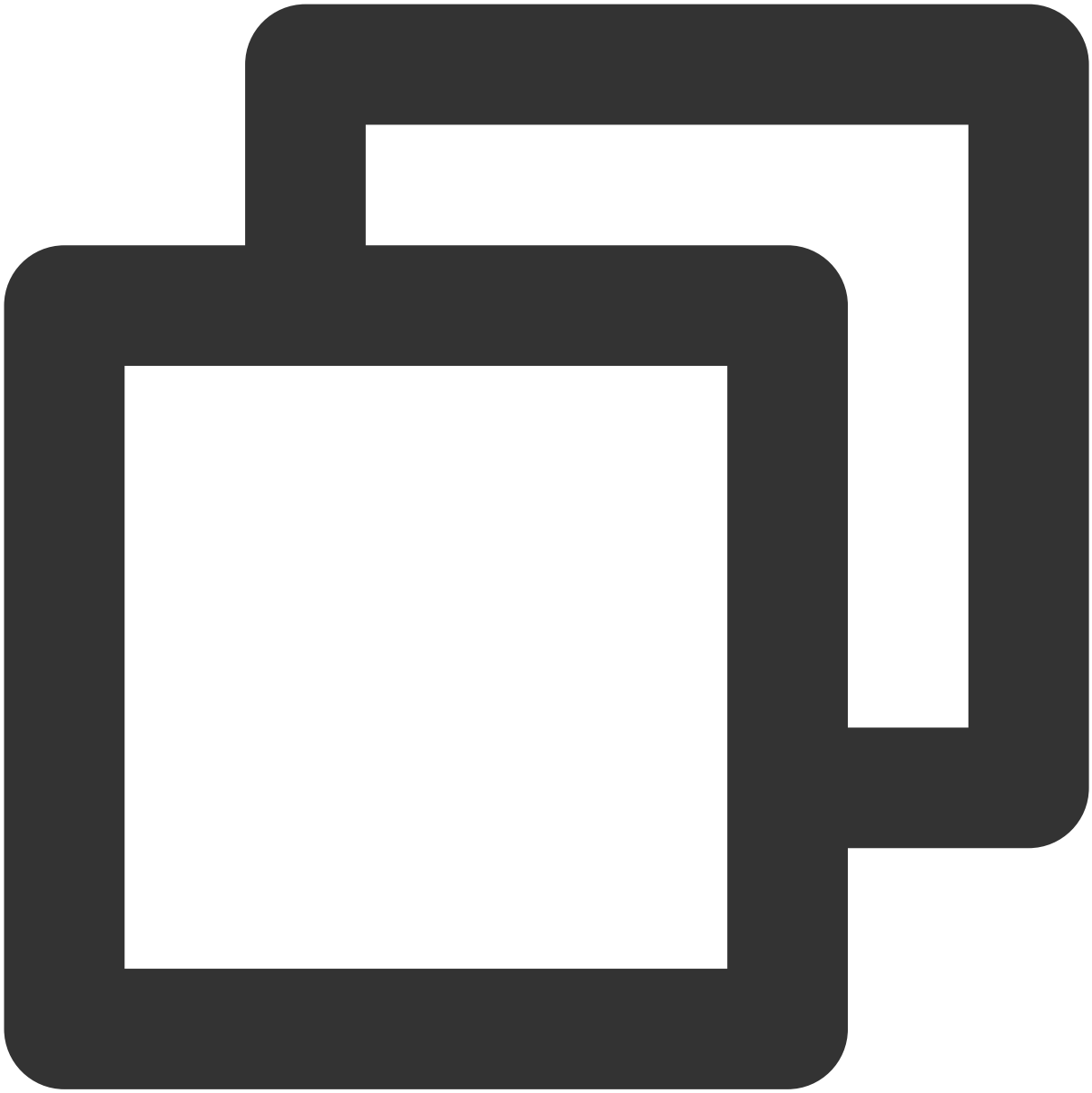


```
#下载对象
response = client.get_object(
    Bucket='examplebucket-1250000000',       #替换为您的 Bucket 名称，examplebucket 是-
```

```
    Key='example-object-1'                    #下载对象的 Key 值
)
fp = response['Body'].get_raw_stream()
download_object = fp.read()                   #得到对象内容
print "get object body: " + download_object
print 'ETag: ' + response['ETag']
print 'x-cos-meta-md5: ' + response['x-cos-meta-md5']    #得到用户自定义参数 x-cos-meta
```

（4） 校验对象

成功下载对象后，用户重新计算对象的校验值（校验算法应与上传对象时保持一致），并与用户自定义参数 x-cos-meta-md5 进行比较，验证下载的对象与上传对象内容是否一致。

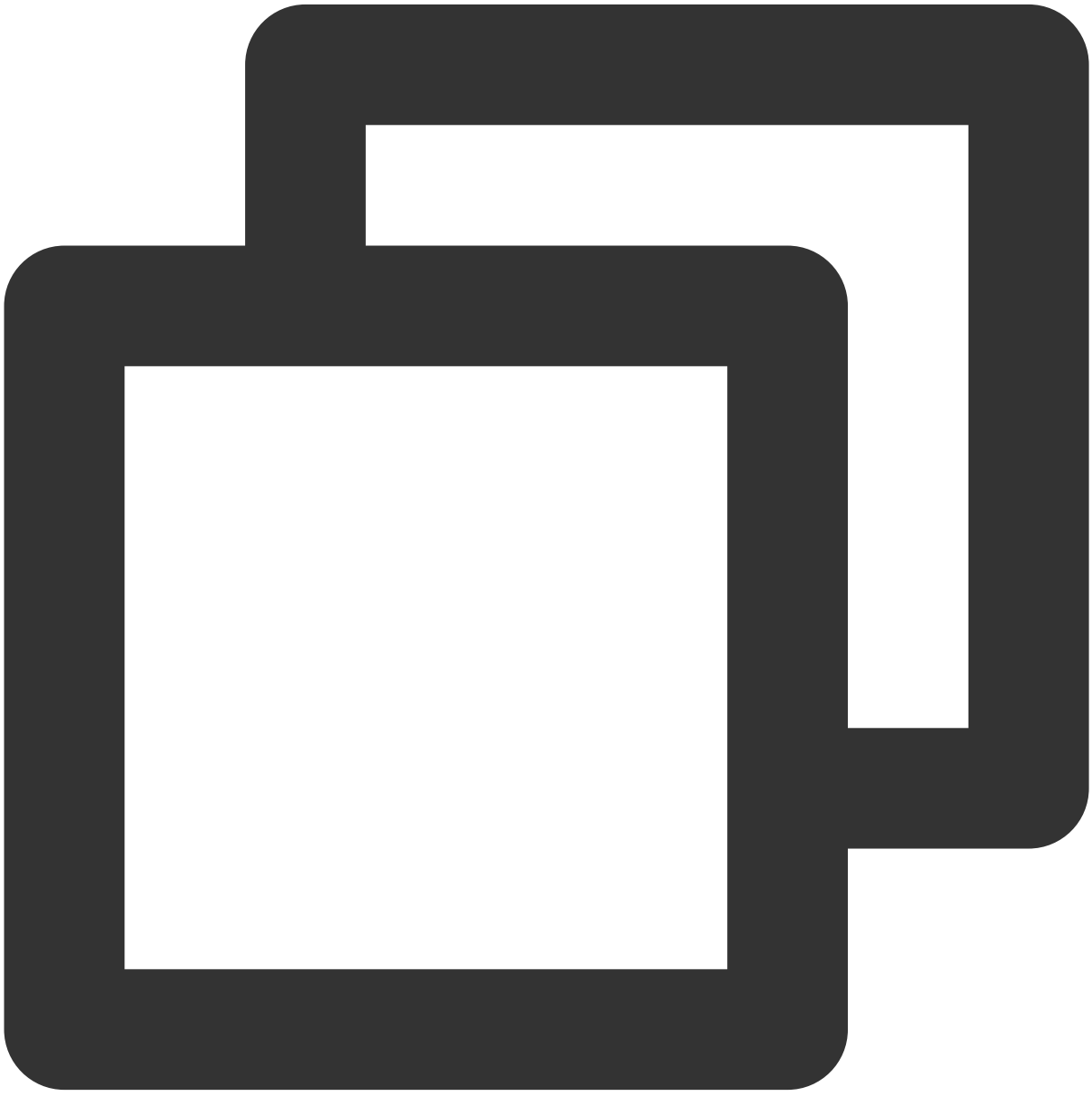


```
#计算下载对象的 MD5 校验值
md5 = hashlib.md5()
md5.update(download_object)
md5_str = md5.hexdigest()
print 'download object md5: ' + md5_str

#通过下载对象的 MD5 校验值与上传对象的 MD5 校验值比较，从而验证对象的一致性
if md5_str == response['x-cos-meta-md5']:
    print 'MD5 check OK'
else:
    print 'MD5 check FAIL'
```

3. 校验分块上传对象

（1）计算对象的校验值

模拟对象分块，并计算整个对象的校验值，下面通过 MD5 校验算法得到对象的校验值，用户可自主选择其他的校验算法。

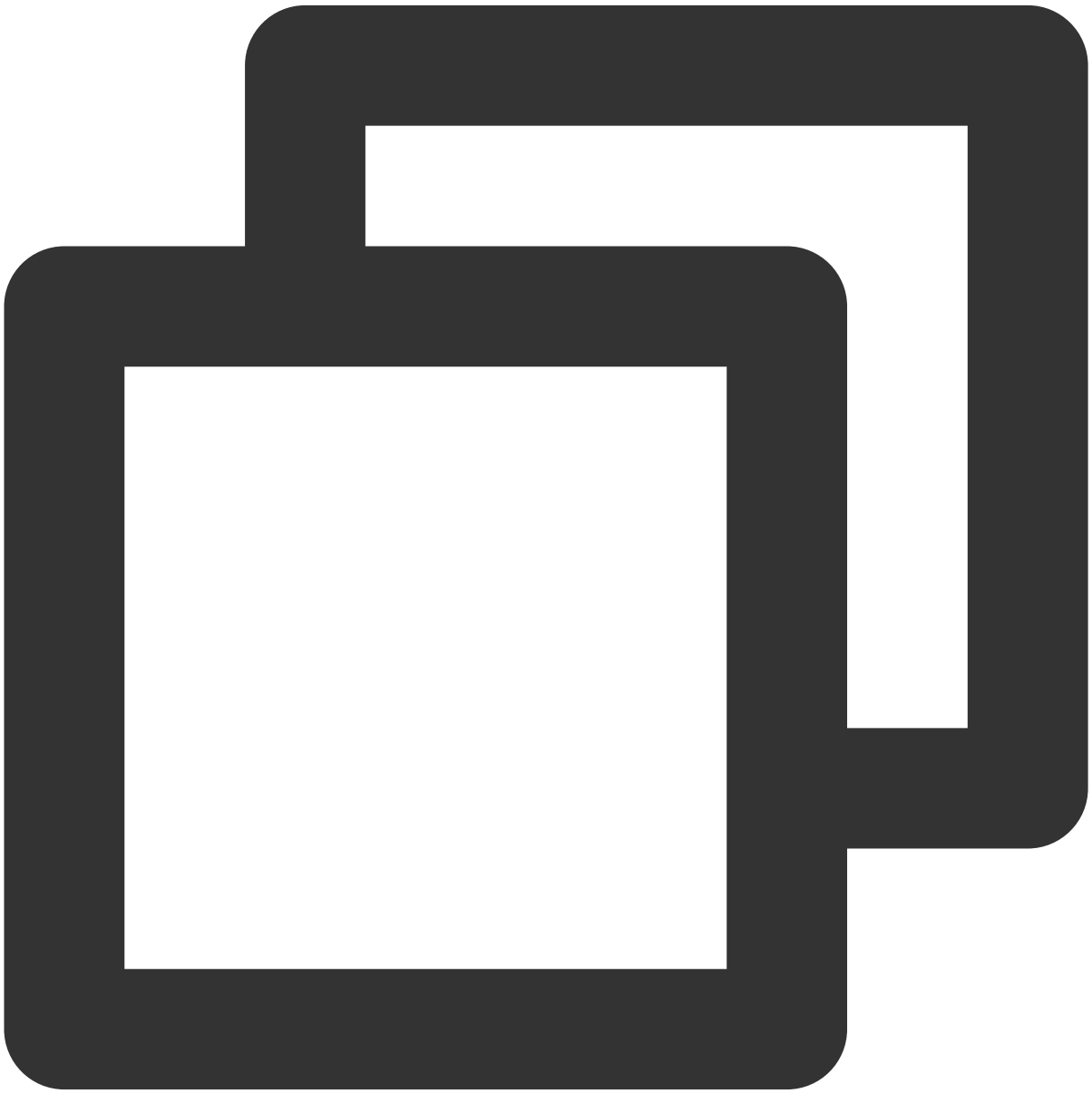


```
OBJECT_PART_SIZE = 1024 * 1024       #模拟每个分块的大小
OBJECT_TOTAL_SIZE = OBJECT_PART_SIZE * 1 + 123       #对象的总大小
object_body = '1' * OBJECT_TOTAL_SIZE        #对象内容
```

```
#计算整个对象内容的 MD5 校验值
md5 = hashlib.md5()
md5.update(object_body)
md5_str = md5.hexdigest()
```

（2）初始化分块上传

在初始化分块上传时，设置自定义参数 x-cos-meta-md5，将整个对象的 MD5 校验值作为参数内容。

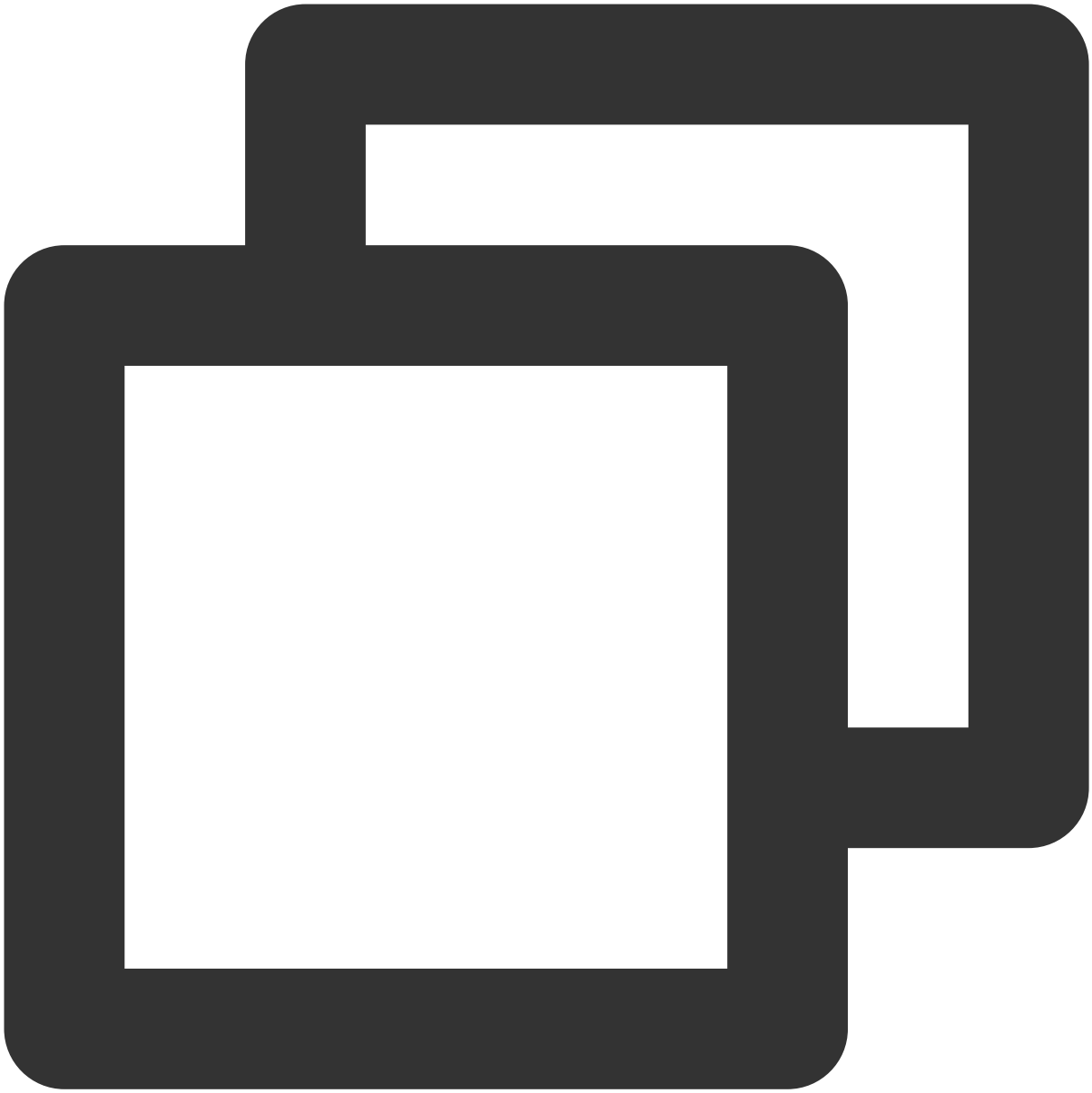


```
#初始化分块上传
```

```
response = client.create_multipart_upload(
    Bucket='examplebucket-1250000000',   #替换为您的 Bucket 名称，examplebucket 是一个举
    Key='exampleobject-2',               #替换为您上传的对象 Key 值
    StorageClass='STANDARD',             #对象的存储级别
    Metadata={
        'x-cos-meta-md5' : md5_str       #设置自定义参数，设置为 MD5 校验值
    }
)
#获取分块上传的 UploadId
upload_id = response['UploadId']
```

（3）分块上传对象

分块上传对象，通过将对象切分成多个块进行上传，最多支持10000分块，每个分块大小为1MB - 5GB，最后一个分块可以小于1MB。上传分块时，需要设置每个分块的 PartNumber（编号）。EnableMD5=True 为打开分块校验，打开后将增加上传耗时，此时 Python SDK 会计算每个分块的 Content-MD5，只有当 COS 服务器接收到的对象 MD5 校验值与 Content-MD5 一致时，分块才可上传成功。上传成功后，返回每个分块的 ETag。

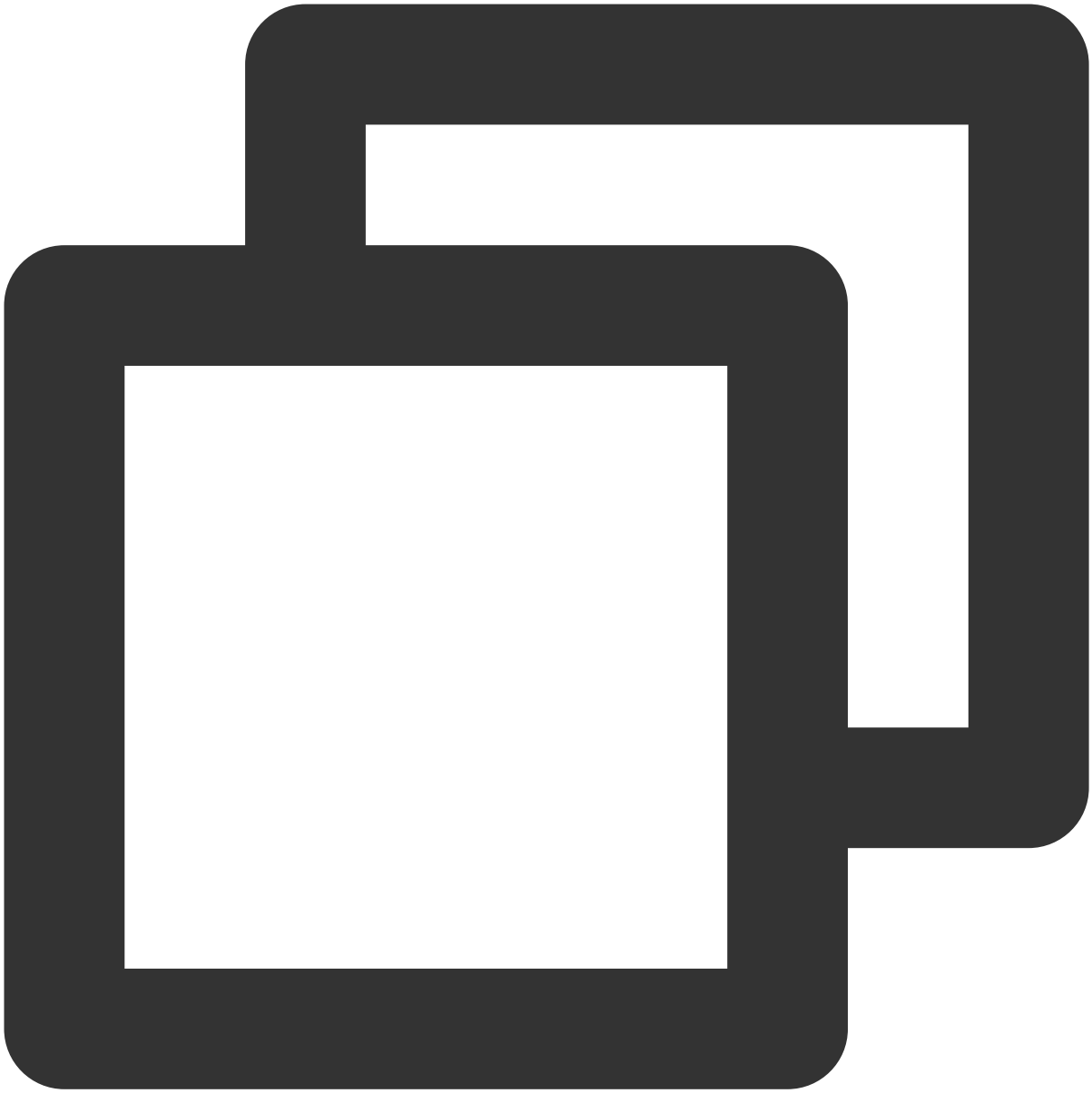


```
#分块上传对象，每个分块大小为 OBJECT_PART_SIZE，最后一个分块可能不足 OBJECT_PART_SIZE
part_list = list()
position = 0
left_size = OBJECT_TOTAL_SIZE
part_number = 0
while left_size > 0:
    part_number += 1
    if left_size >= OBJECT_PART_SIZE:
        body = object_body[position:position+OBJECT_PART_SIZE]
    else:
        body = object_body[position:]
```

```
    position += OBJECT_PART_SIZE
    left_size -= OBJECT_PART_SIZE

    #上传分块
    response = client.upload_part(
        Bucket='examplebucket-1250000000',   #替换为您的 Bucket 名称，examplebucket 是一
        Key='exampleobject-2',               #对象的 Key 值
        Body=body,
        PartNumber=part_number,
        UploadId=upload_id,
        EnableMD5=True        #打开分块校验，COS 服务器会对每个分块进行 MD5 校验。
    )
    etag = response['ETag']      #ETag 表示每个分块的 MD5 值
    part_list.append({'ETag' : etag, 'PartNumber' : part_number})
    print etag + ', ' + str(part_number)
```

（4）完成分块上传

在所有分块上传完成后，需要进行完成分块上传操作。每个分块的 ETag 和 PartNumber 需要一一对应，COS 服务器用于校验块的准确性。完成分块上传后，返回的 ETag 表示合并后对象的唯一标签值，而不是整个对象内容的 MD5 校验值，因此下载对象时，可通过自定义参数来进行校验。

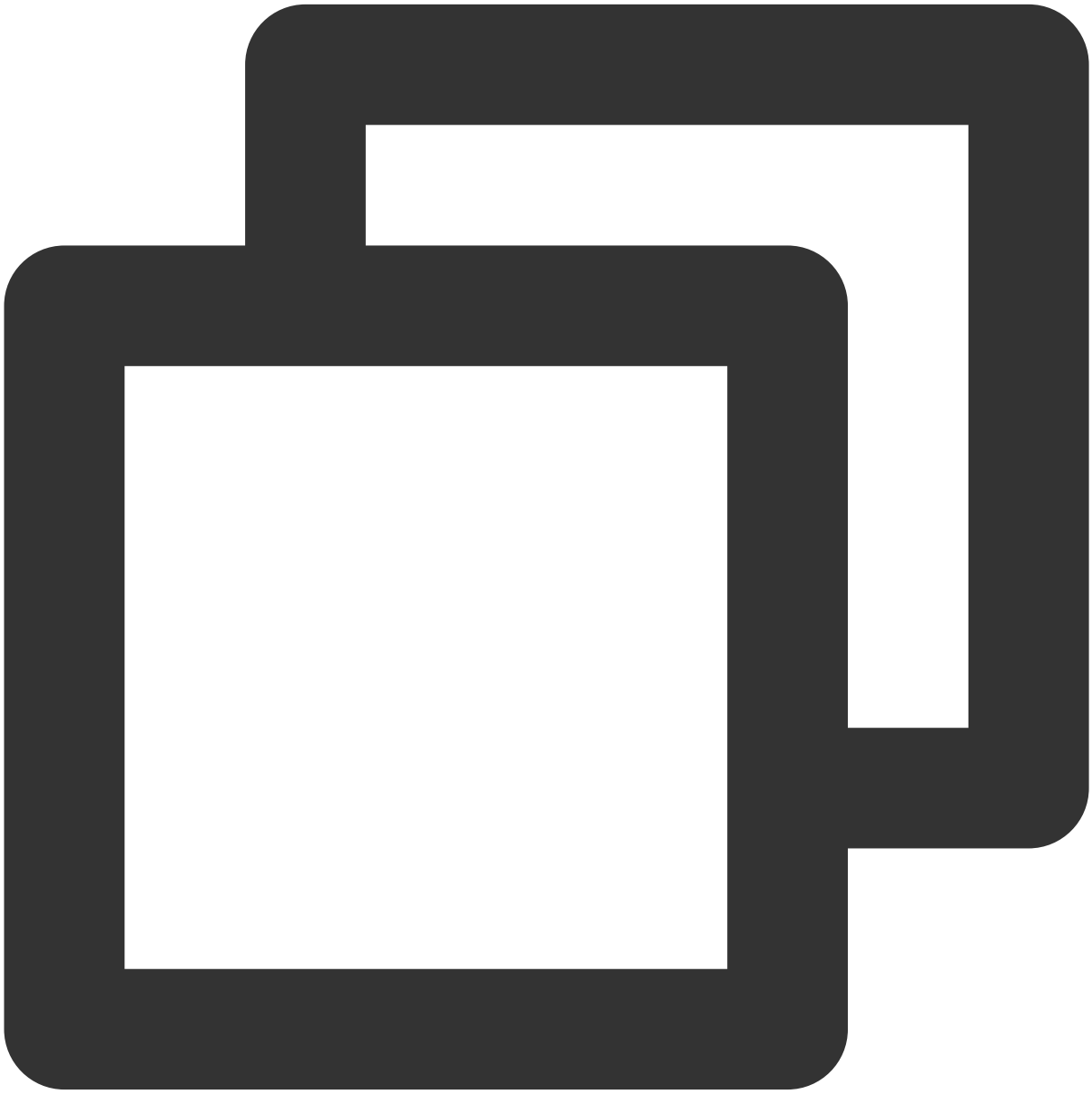


```
#完成分块上传
response = client.complete_multipart_upload(
    Bucket='examplebucket-1250000000',  #替换为您的 Bucket 名称，examplebucket 是一个举
    Key='exampleobject-2',              #对象的 Key 值
    UploadId=upload_id,
    MultipartUpload={       #要求每个分块的 ETag 和 PartNumber 一一对应
        'Part' : part_list
    },
)

#ETag 表示合并后对象的唯一标签值，该值不是对象内容的 MD5 校验值，仅用于检查对象唯一性。
```

```
print "ETag: " + response['ETag']
print "Location: " + response['Location']    #URL地址
print "Key: " + response['Key']
```

（5）下载对象

下载对象并得到用户自定义参数。

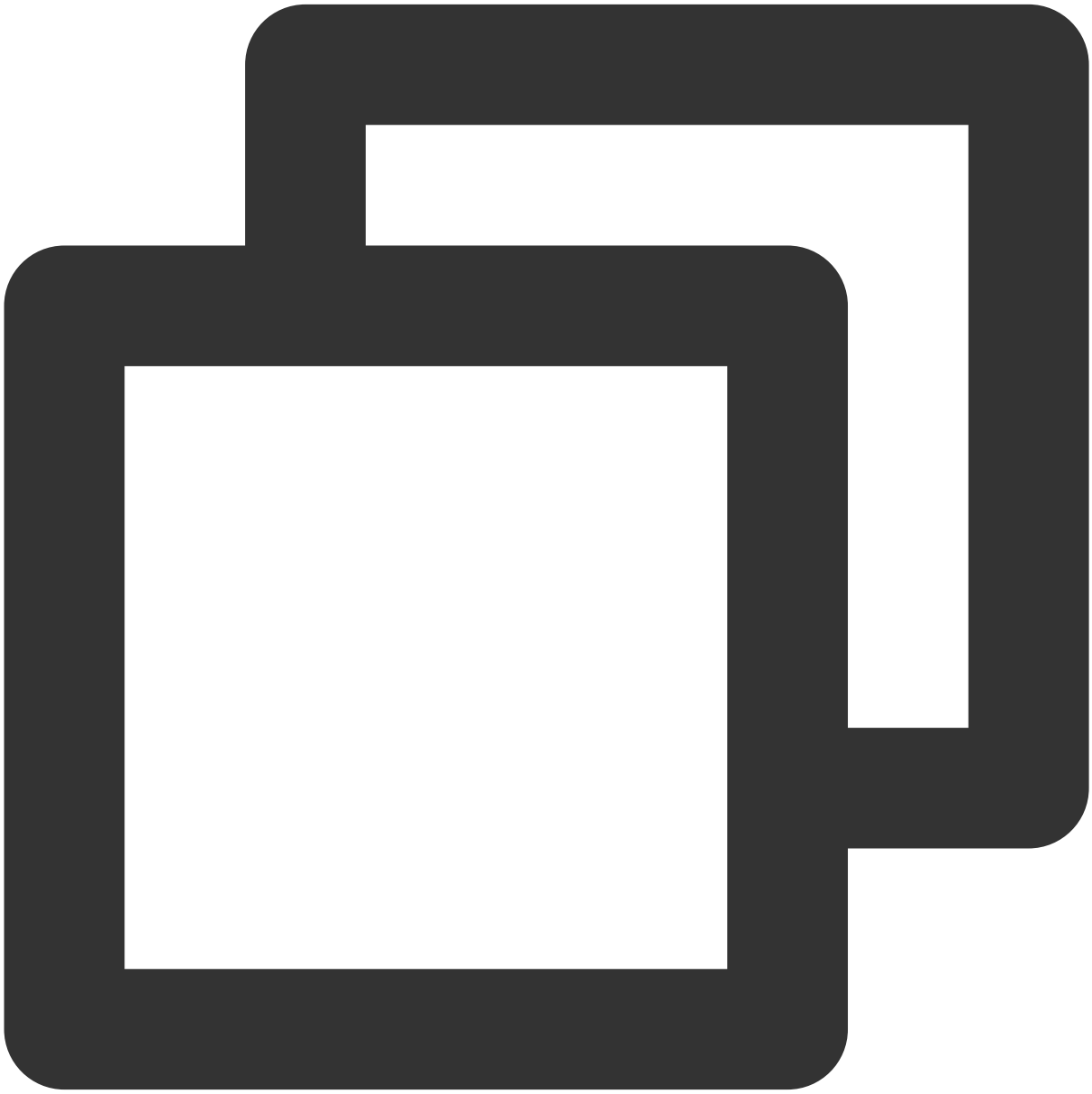


```
# 下载对象
response = client.get_object(
    Bucket='examplebucket-1250000000',  #替换为您的 Bucket 名称，examplebucket 是一个举
```

```
    Key='exampleobject-2'                #对象的 Key 值
)
print 'ETag: ' + response['ETag']                              #对象的 ETag 不是对象内容的 MD
print 'x-cos-meta-md5: ' + response['x-cos-meta-md5']          #得到用户自定义参数 x-cos-met
```

（6）校验对象

成功下载对象后，用户重新计算对象的 MD5 校验值，并与用户自定义参数 x-cos-meta-md5 进行比较，验证下载的对象与上传对象内容是否一致。



```
#计算下载对象的 MD5 校验值
```

```
fp = response['Body'].get_raw_stream()
DEFAULT_CHUNK_SIZE = 1024*1024
md5 = hashlib.md5()
chunk = fp.read(DEFAULT_CHUNK_SIZE)
while chunk:
    md5.update(chunk)
    chunk = fp.read(DEFAULT_CHUNK_SIZE)
md5_str = md5.hexdigest()
print 'download object md5: ' + md5_str

#通过下载对象的 MD5 校验值与上传对象的 MD5 校验值比较，从而验证对象的一致性
if md5_str == response['x-cos-meta-md5']:
    print 'MD5 check OK'
else:
    print 'MD5 check FAIL'
```

CRC64 校验

最近更新时间：2024-01-06 10:54:03

简介

数据在客户端和服务器间传输时可能会出现错误，COS 除了可以通过 MD5 和自定义属性 验证数据完整性外，还可以通过 CRC64 检验码来进行数据校验。
COS 会对新上传的对象进行 CRC64 计算，并将结果作为对象的属性进行存储，随后在返回的响应头部中携带 x-cos-hash-crc64ecma，该头部表示上传对象的 CRC64 值， 根据ECMA-182标准计算得到。对于 CRC64 特性上线前就已经存在于 COS 的对象，COS 不会对其计算 CRC64 值，所以获取此类对象时不会返回其 CRC64 值。

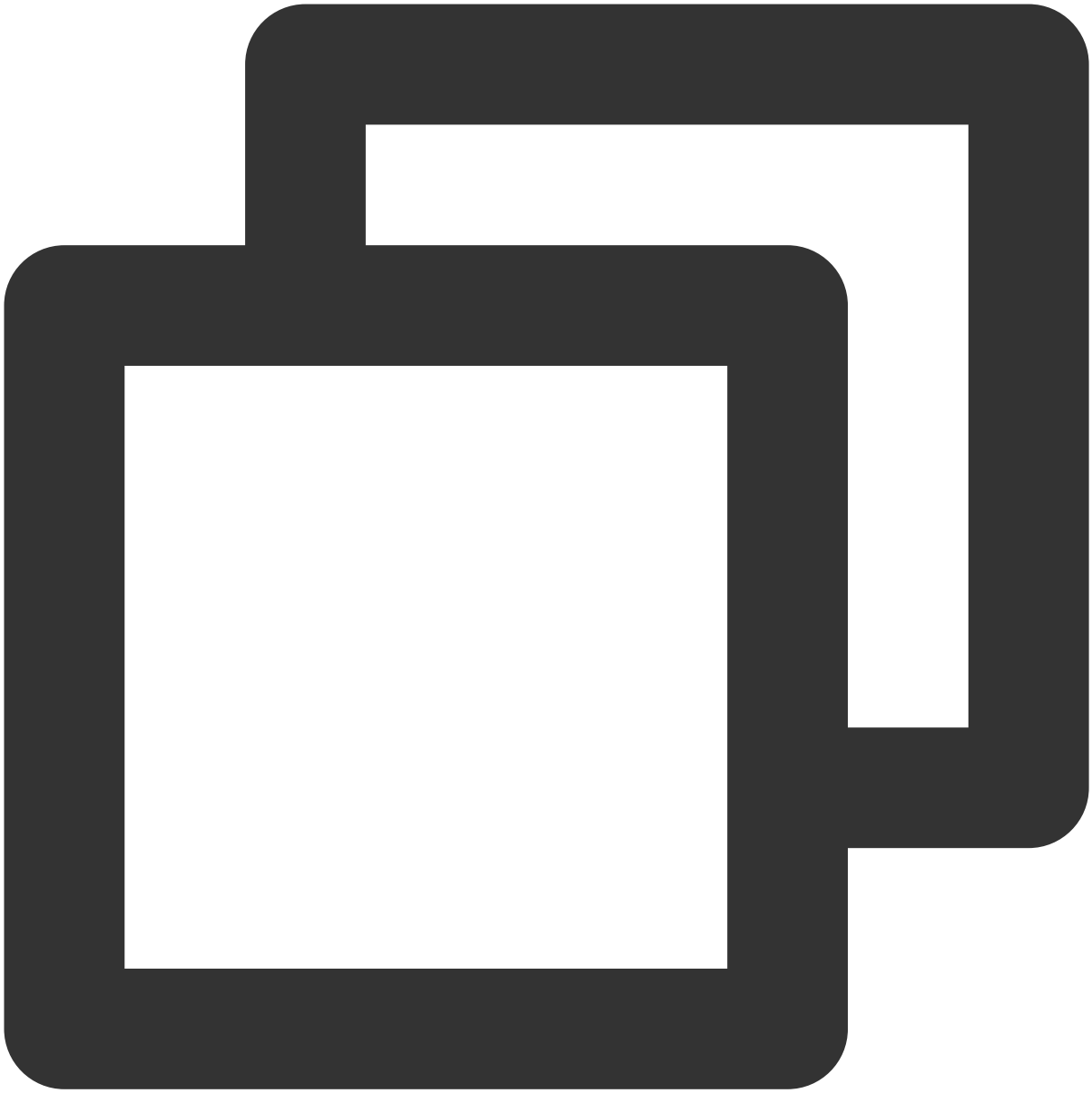
操作说明

目前支持 CRC64 的 API 如下：
简单上传接口
PUT Object 和 POST Object：用户可在返回的响应头中获得文件 CRC64 校验值。
分块上传接口
Upload Part：用户可以根据 COS 返回的 CRC64 值与本地计算的数值进行比较验证。
Complete Multipart Upload：如果每个分块都有 CRC64 属性，则会返回整个对象的 CRC64 值，如果某些分块不具备 CRC64 值，则不返回。
执行 Upload Part - Copy 时，会返回对应的 CRC64 值。
执行 PUT Object - Copy 时，如果源对象存在 CRC64 值，则返回 CRC64，否则不返回。
执行 HEAD Object 和 GET Object 时，如果对象存在 CRC64，则返回。用户可以根据 COS 返回的 CRC64 值和本地计算的 CRC64 进行比较验证。

API 接口示例

分块上传响应

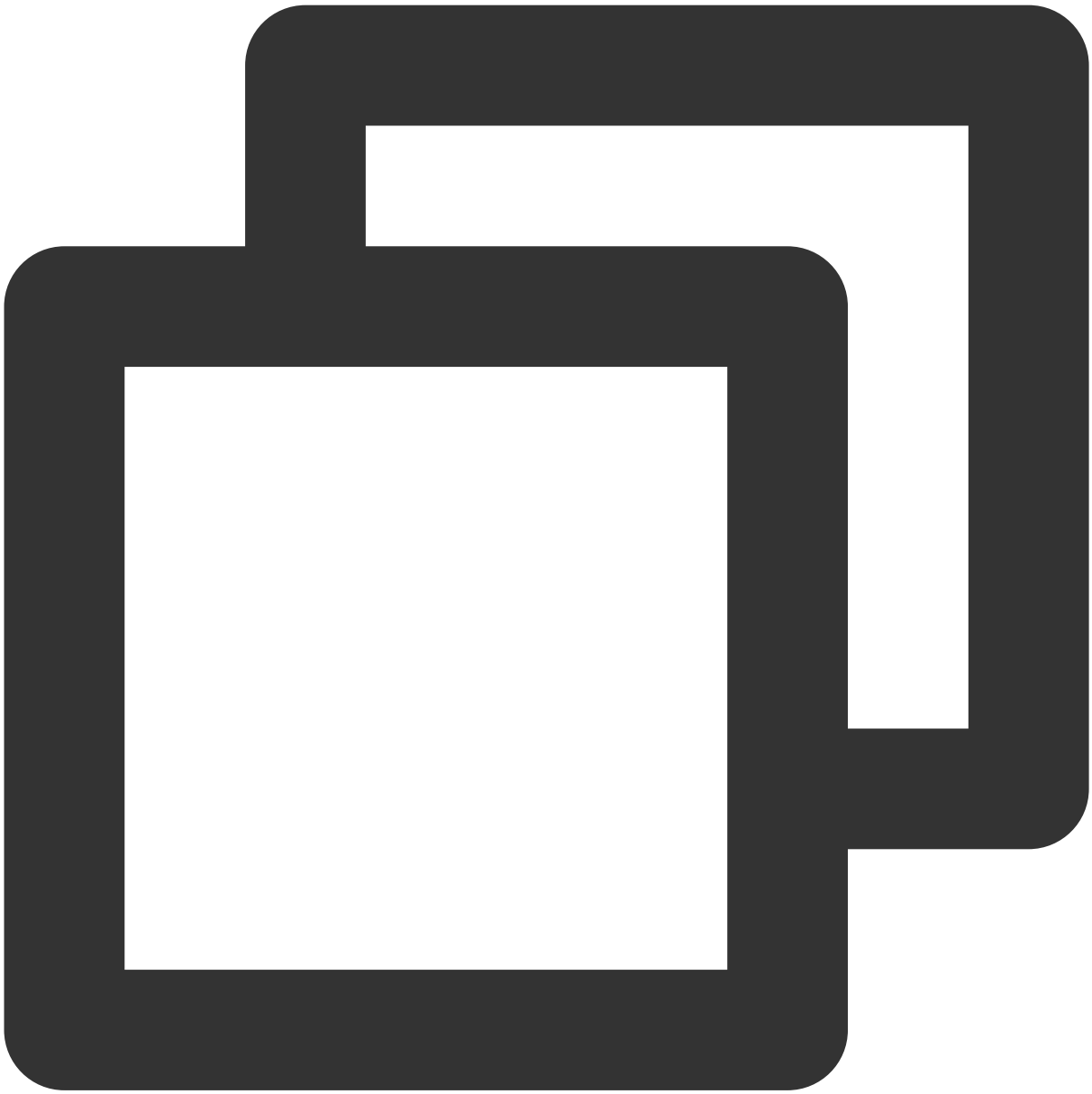
下面为用户发出 Upload Part 请求后得到的响应示例。x-cos-hash-crc64ecma 头部表示分块的 CRC64 值，用户可以通过该值与本地计算的 CRC64 值进行比较，从而校验分块完整性。



```
HTTP/1.1 200 OK
content-length: 0
connection: close
date: Thu, 05 Dec 2019 01:58:03 GMT
etag: "358e8c8b1bfa35ee3bd44cb3d2cc416b"
server: tencent-cos
x-cos-hash-crc64ecma: 15060521397700495958
x-cos-request-id: NWRlODY0MmJfMjBiNDU4NjRfNjkyZl80ZjZi****
```

完成分块上传响应

下面为用户发出 Complete Multipart Upload 请求后得到的响应示例。x-cos-hash-crc64ecma 头部表示整个对象的 CRC64 值，用户可以通过该值与本地计算的 CRC64 值进行比较，从而校验对象完整性。



```
HTTP/1.1 200 OK
content-type: application/xml
transfer-encoding: chunked
connection: close
date: Thu, 05 Dec 2019 02:01:17 GMT
server: tencent-cos
x-cos-hash-crc64ecma: 15060521397700495958
x-cos-request-id: NWRlODY0ZWRfMjNiMjU4NjRfOGQ4Ml81MDEw****
```

```
[Object Content]
```

SDK 示例

Python SDK

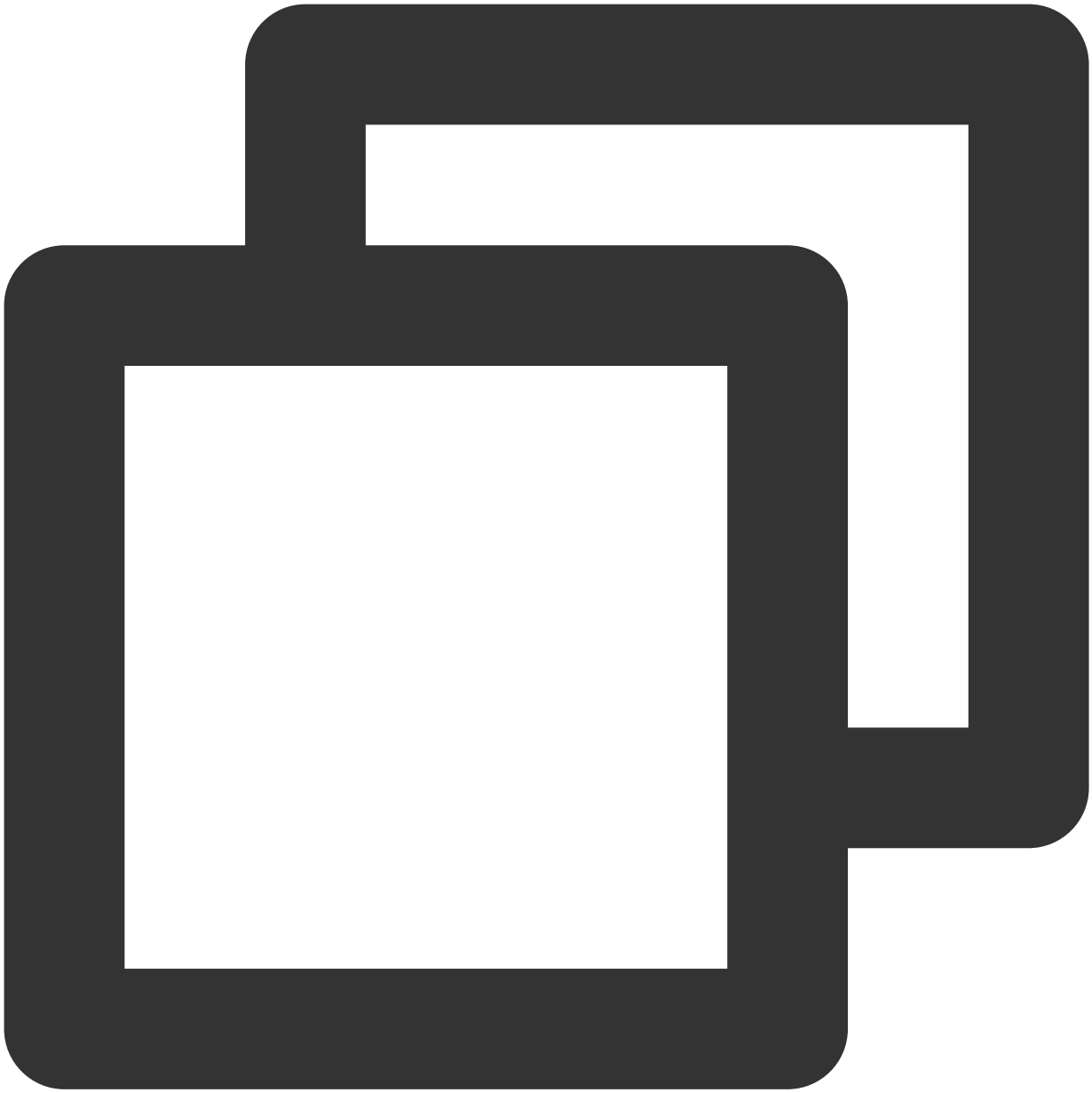
下面以 Python SDK 为例演示如何校验对象，完整的示例代码如下。

说明

代码基于 Python 2.7，其中 Python SDK 详细使用方式，请参见 Python SDK 的 对象操作 文档。

1. 初始化配置

设置用户属性，包括 SecretId、SecretKey 和 Region，并创建客户端对象。

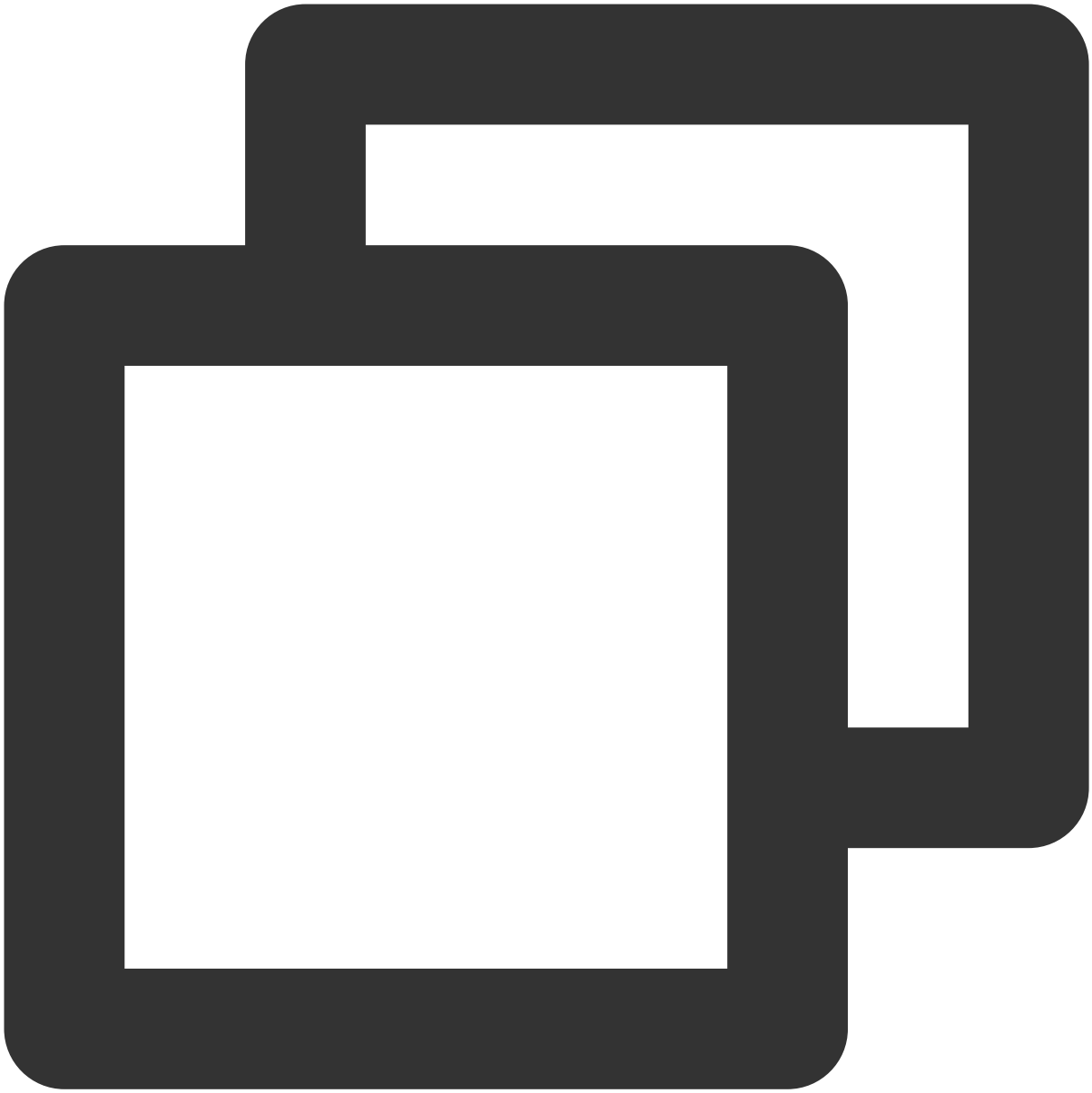


```
# -*- coding=utf-8
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
from qcloud_cos import CosServiceError
from qcloud_cos import CosClientError
import sys
import logging
import hashlib
import crcmod

logging.basicConfig(level=logging.INFO, stream=sys.stdout)
```

```
# 设置用户属性，包括 SecretId，SecretKey，Region
# APPID 已在配置中移除，请在参数 Bucket 中带上 APPID。Bucket 由 BucketName-APPID 组成
secret_id = COS_SECRETID            # 替换为您的 SecretId 信息
secret_key = COS_SECRETKEY          # 替换为您的 SecretKey 信息
region = 'ap-beijing'       # 替换为您的 Region，这里以北京为例
token = None                # 使用临时密钥需要传入 Token，默认为空，可不填
config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key, Token=t
client = CosS3Client(config)
```

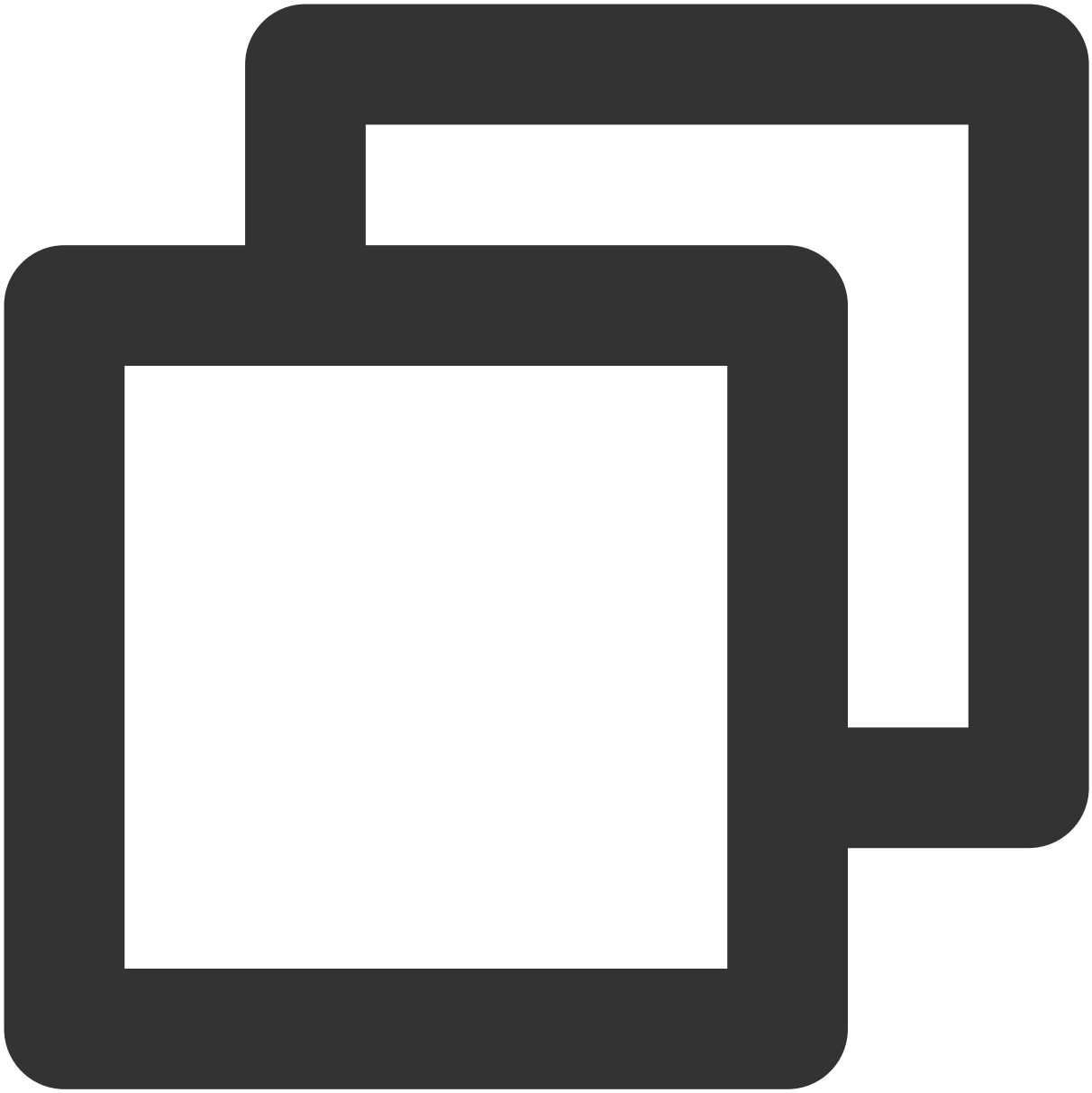
2. 计算对象的校验值

模拟对象分块，并计算整个对象的 CRC64 校验值。



```
OBJECT_PART_SIZE = 1024 * 1024       #模拟每个分块的大小
OBJECT_TOTAL_SIZE = OBJECT_PART_SIZE * 1 + 123       #对象的总大小
object_body = '1' * OBJECT_TOTAL_SIZE        #对象内容

#计算整个对象 crc64 校验值
c64 = crcmod.mkCrcFun(0x142F0E1EBA9EA3693, initCrc=0, xorOut=0xffffffffffffffff, re
local_crc64 =str(c64(object_body))
```

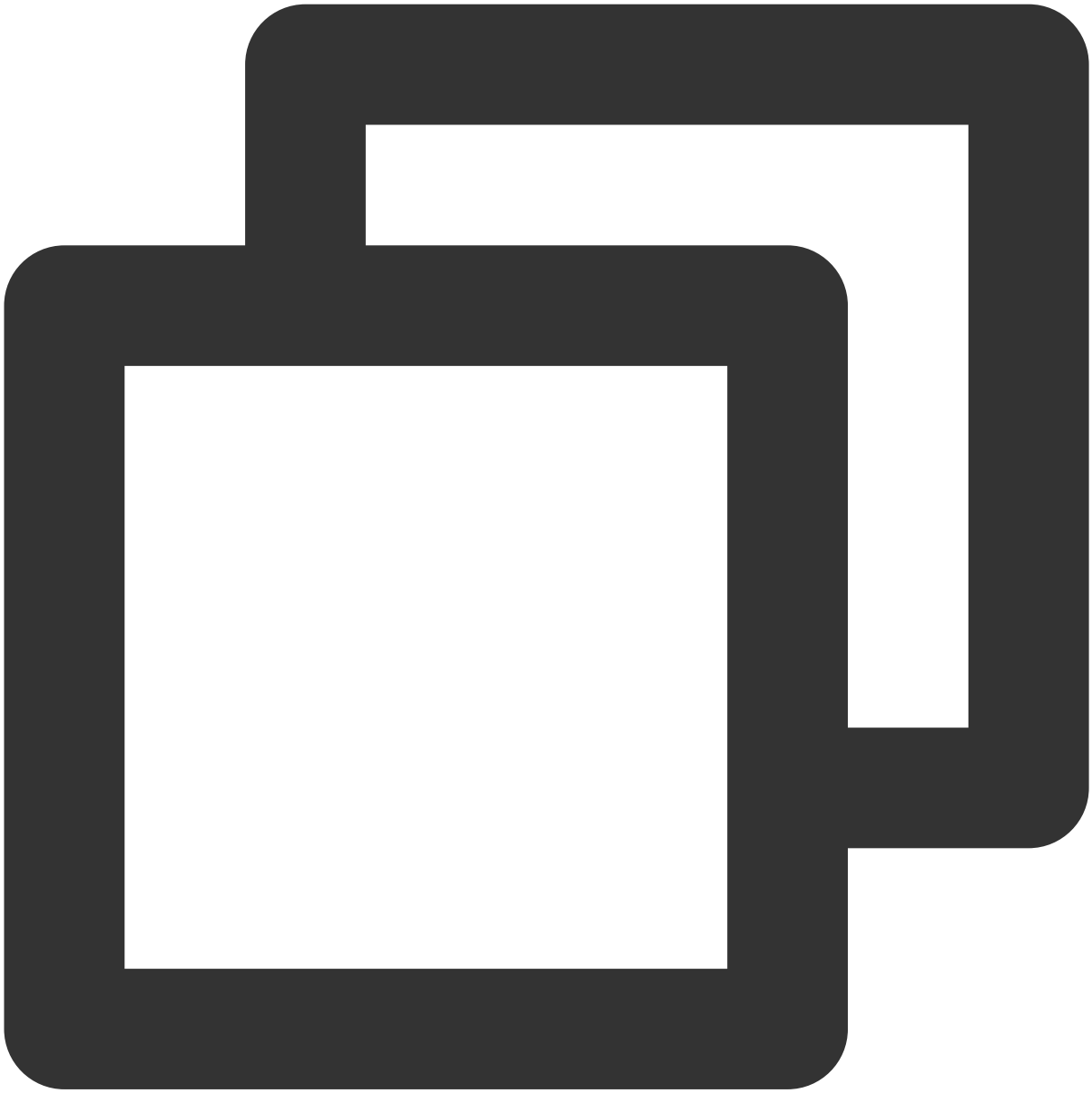
3. 初始化分块上传



```
#初始化分块上传
response = client.create_multipart_upload(
    Bucket='examplebucket-1250000000',   #替换为您的 Bucket 名称，examplebucket 是一个举
    Key='exampleobject',                 #替换为您上传的对象 Key 值
    StorageClass='STANDARD',             #对象的存储类型
)
#获取分块上传的 UploadId
upload_id = response['UploadId']
```

4. 分块上传对象

分块上传对象，通过将对象切分成多个块进行上传，最多支持10000分块，每个分块大小为1MB - 5GB，最后一个分块可以小于1MB。上传分块时，需要设置每个分块的 PartNumber（编号），并计算每个分块的 CRC64 值，在分块上传成功后，可以通过返回的 CRC64 值与本地计算的数值进行校验。

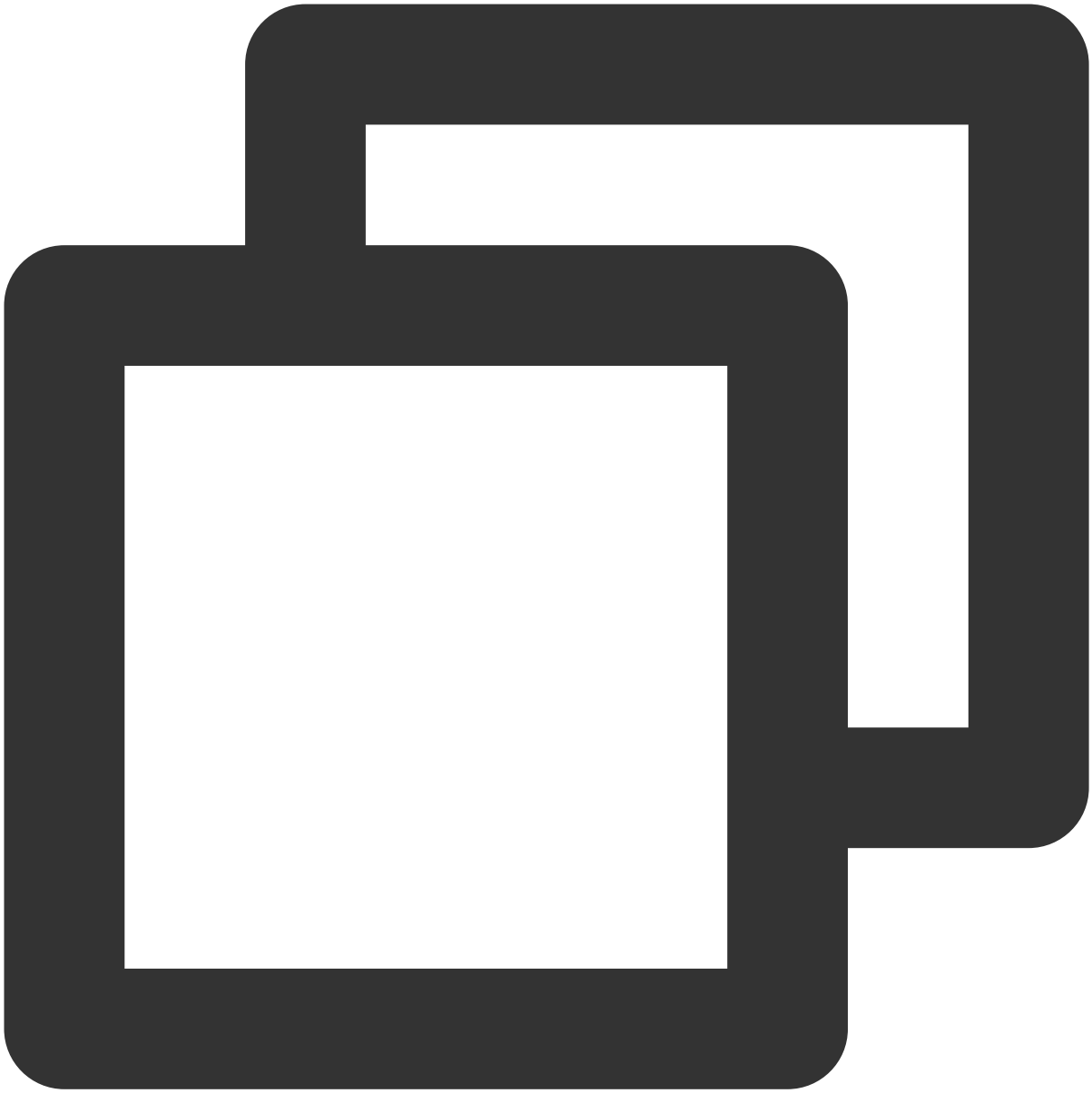


```
#分块上传对象，每个分块大小为 OBJECT_PART_SIZE，最后一个分块可能不足 OBJECT_PART_SIZE
part_list = list()
position = 0
left_size = OBJECT_TOTAL_SIZE
part_number = 0
while left_size > 0:
    part_number += 1
```

```
    if left_size >= OBJECT_PART_SIZE:
        body = object_body[position:position+OBJECT_PART_SIZE]
    else:
        body = object_body[position:]
    position += OBJECT_PART_SIZE
    left_size -= OBJECT_PART_SIZE
    local_part_crc64 = str(c64(body))    #本地计算 CRC64

    response = client.upload_part(
        Bucket='examplebucket-1250000000',
        Key='exampleobject',
        Body=body,
        PartNumber=part_number,
        UploadId=upload_id,
    )
    part_crc_64 = response['x-cos-hash-crc64ecma']       # 服务器返回的 CRC64
    if local_part_crc64 != part_crc_64:          # 数据检验
        print 'crc64 check FAIL'
        exit(-1)
    etag = response['ETag']
    part_list.append({'ETag' : etag, 'PartNumber' : part_number})
```

5. 完成分块上传

在所有分块上传完成后，需要进行完成分块上传操作。可以通过 COS 返回的 CRC64 和本地对象的 CRC64 进行比较验证。



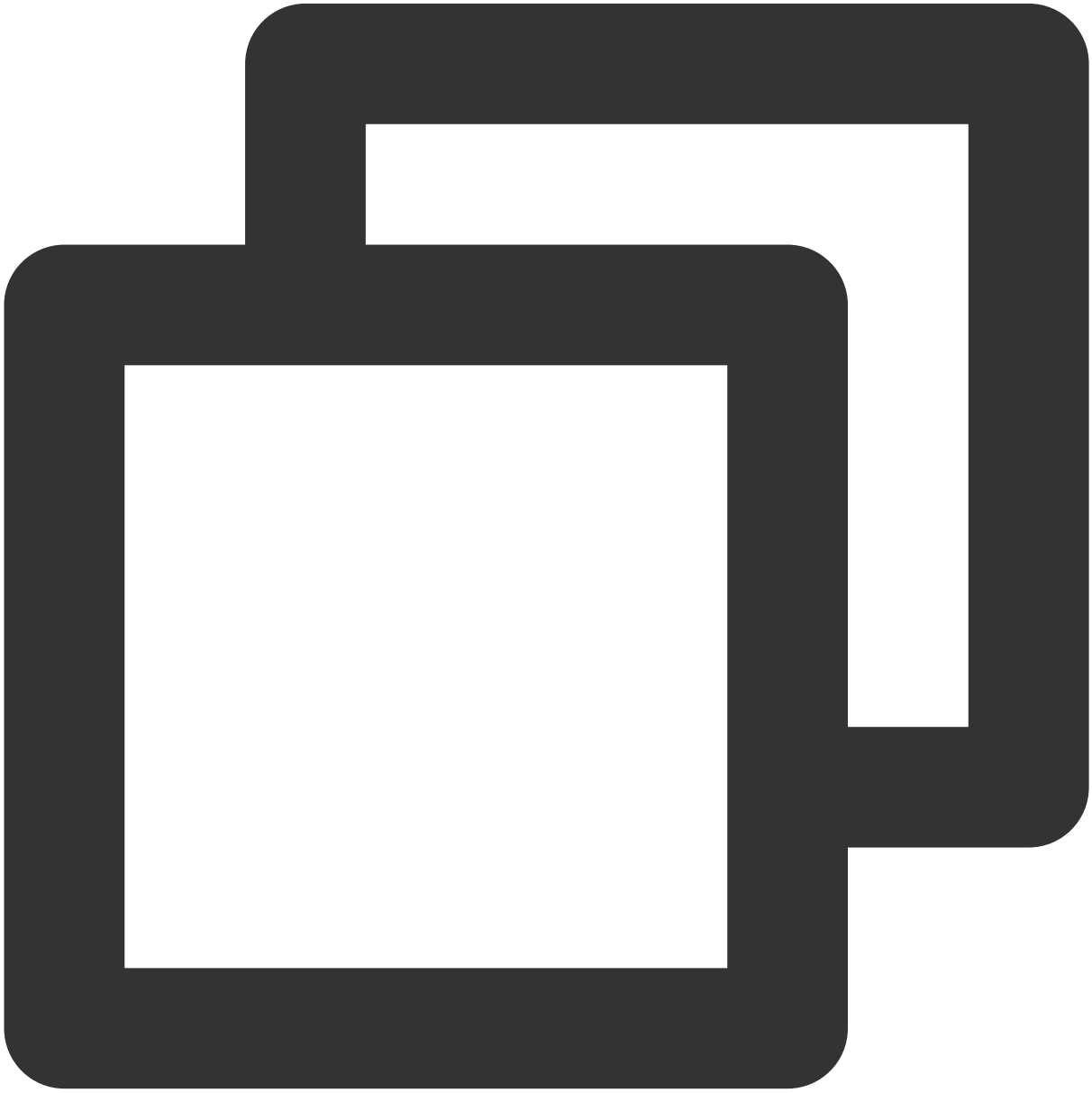
```
#完成分块上传
response = client.complete_multipart_upload(
    Bucket='examplebucket-1250000000',  #替换为您的 Bucket 名称，examplebucket 是一个举
    Key='exampleobject',                #对象的 Key 值
    UploadId=upload_id,
    MultipartUpload={                               #要求每个分块的 ETag 和 PartNumber 一一
        'Part' : part_list
    },
)
crc64ecma = response['x-cos-hash-crc64ecma']
if crc64ecma != local_crc64:                        # 数据检验
```

```
    print 'check crc64 Failed'
    exit(-1)
```

Java SDK

上传对象，推荐使用 Java SDK 的高级接口，参见 Java SDK 对象操作。

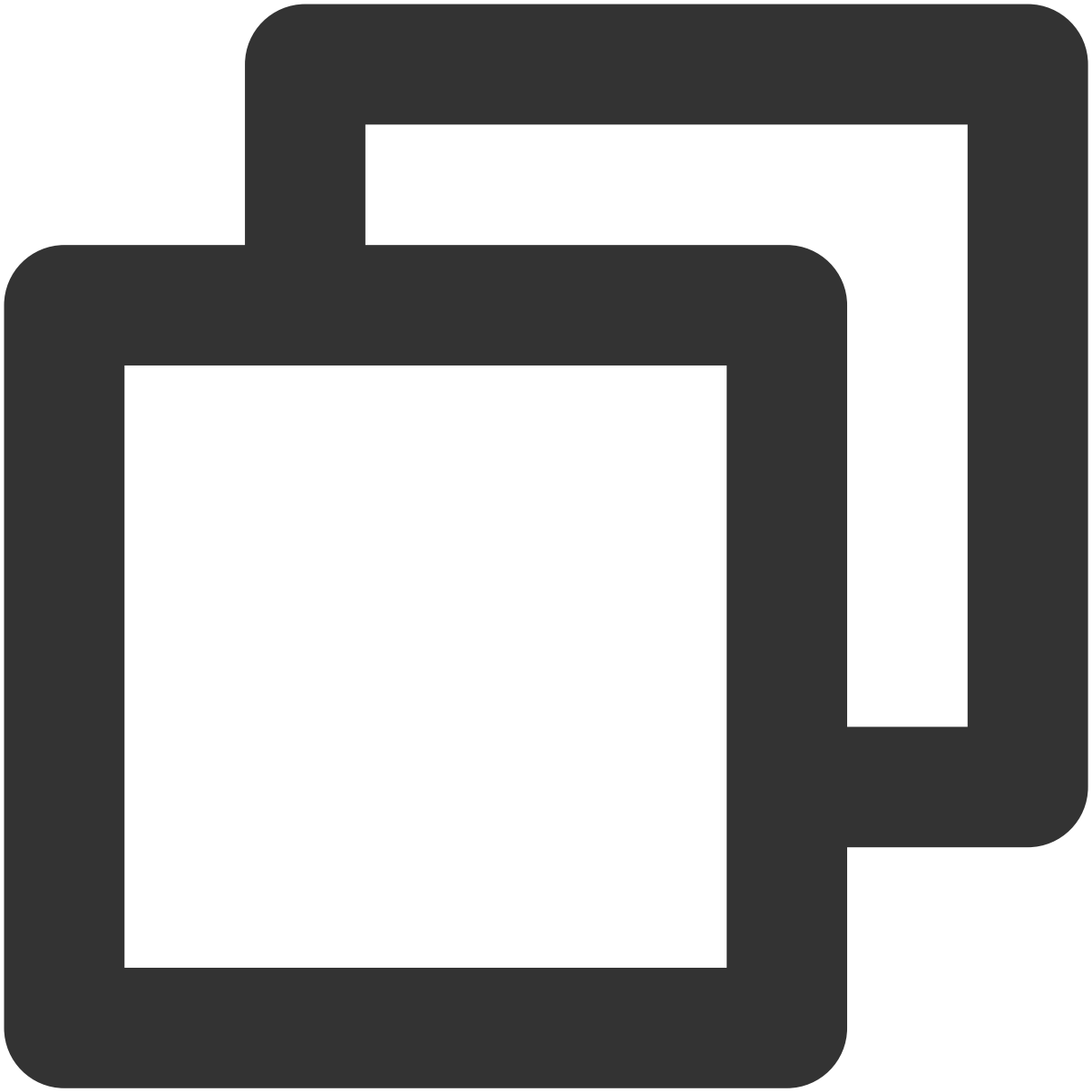
如何在本地计算文件的 crc64



```
String calculateCrc64(File localFile) throws IOException {
    CRC64 crc64 = new CRC64();
```

```
    try (FileInputStream stream = new FileInputStream(localFile)) {
        byte[] b = new byte[1024 * 1024];
        while (true) {
            final int read = stream.read(b);
            if (read <= 0) {
                break;
            }
            crc64.update(b, read);
        }
    }

    return Long.toUnsignedString(crc64.getValue());
}
```

如何获得 COS 上文件的 crc64 值, 并与本地文件做校验



```
// COSClient 的创建参考：[快速入门](https://www.tencentcloud.com/document/product/436/1
ObjectMetadata cosMeta = COSClient().getObjectMetadata(bucketName, cosFilePath);
String cosCrc64 = cosMeta.getCrc64Ecma();
String localCrc64 = calculateCrc64(localFile);

if (cosCrc64.equals(localCrc64)) {
    System.out.println("ok");
} else {
    System.out.println("fail");
}
```

大数据实践
使用 COS 作为 Druid 的 Deep storage

最近更新时间：2024-01-06 10:54:03

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。

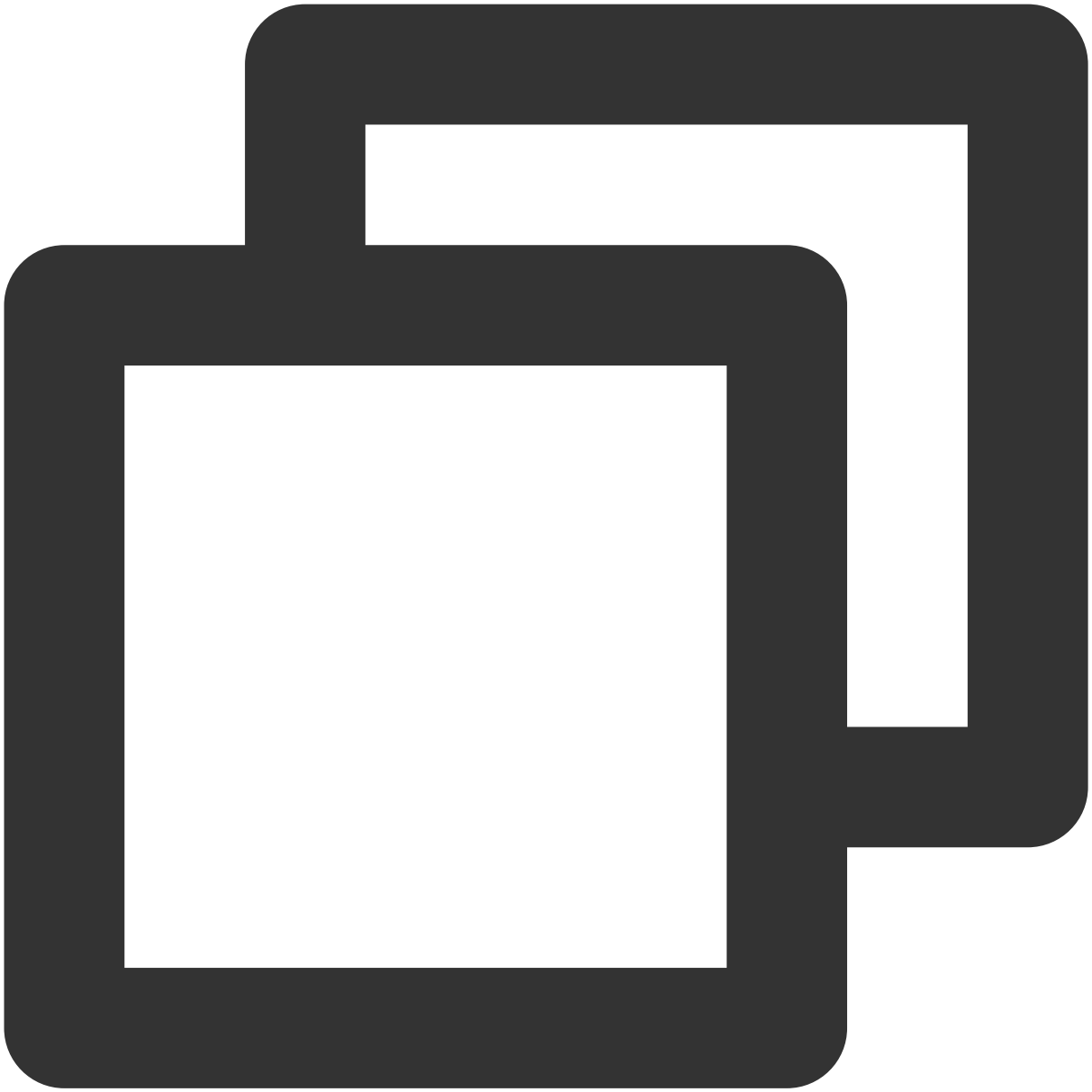
安装 hadoop-cos

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.x.x.jar 以及 cos_hadoop_api-5.2.6.jar 拷贝到 druid 安装路径的 extensions/druid-hdfs-storage 以及 hadoop-dependencies/hadoop-client/2.x.x。

使用方法

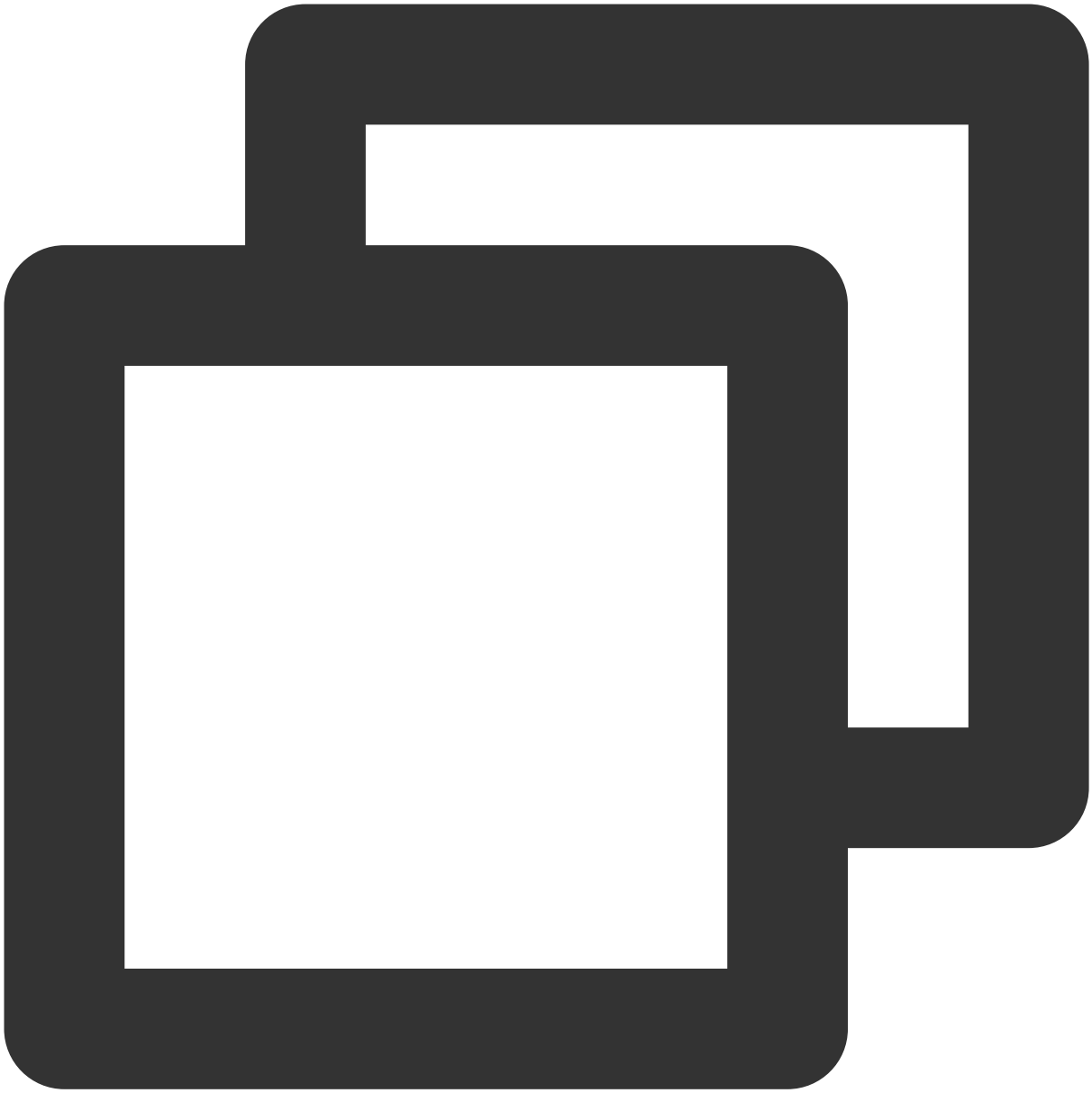
配置修改

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.tmp.dir</name>
        <value>/tmp/hadoop_cos</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。

使用 DataX 导入或导出 COS

最近更新时间：2024-01-06 10:54:03

环境依赖

HADOOP-COS 与对应版本的 cos_api-bundle。
DataX 版本：DataX-3.0。

下载与安装

获取 HADOOP-COS

在官方 Github 上下载 HADOOP-COS 与对应版本的 cos_api-bundle。

获取 DataX 软件包

在官方 Github 上下载 DataX。

安装 HADOOP-COS

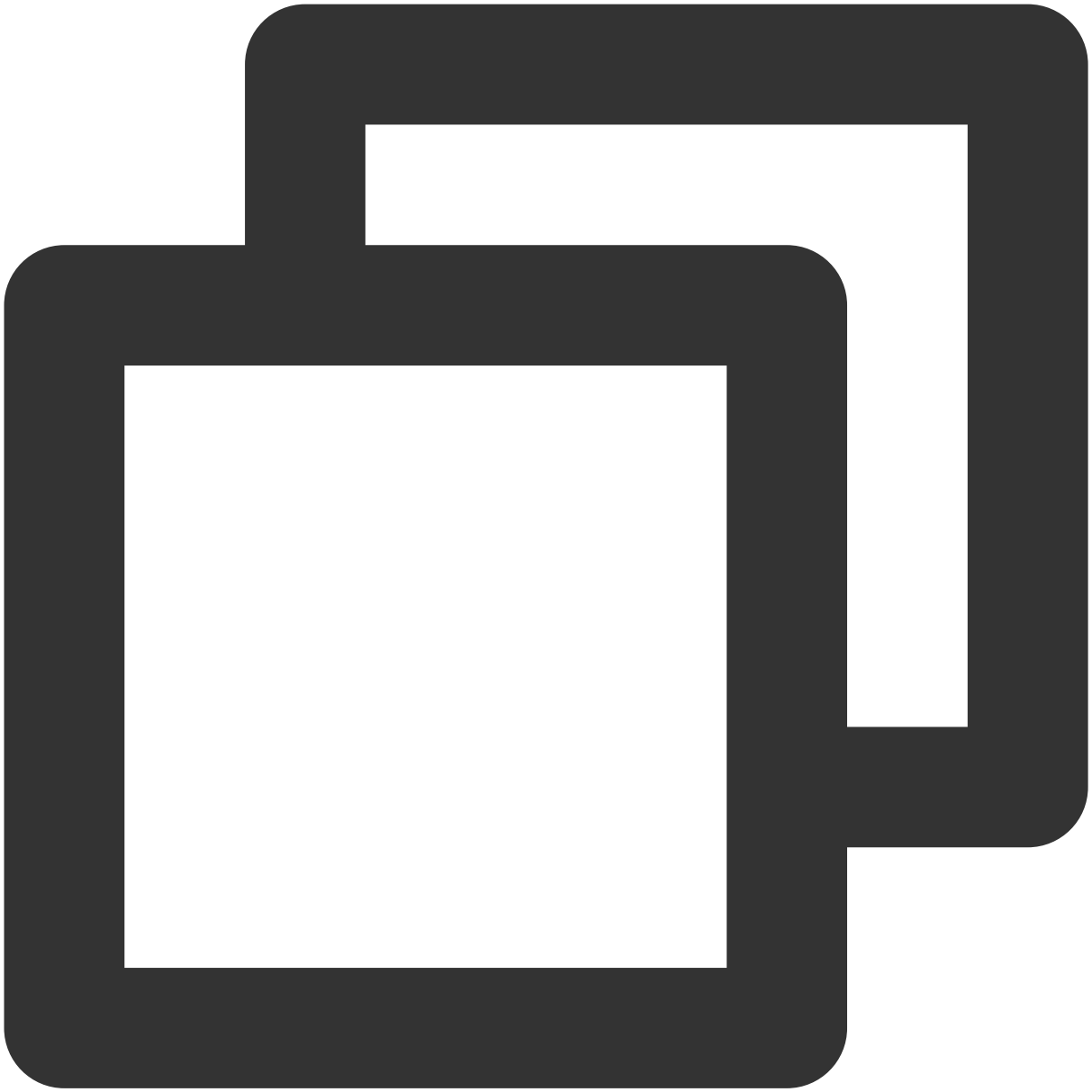
下载 HADOOP-COS 后，将 `hadoop-cos-2.x.x-${version}.jar` 和 `cos_api-bundle-${version}.jar` 拷贝到 Datax 解压路径 `plugin/reader/hdfsreader/libs/` 以及 `plugin/writer/hdfswriter/libs/` 下。

使用方法

DataX 配置

修改 datax.py 脚本

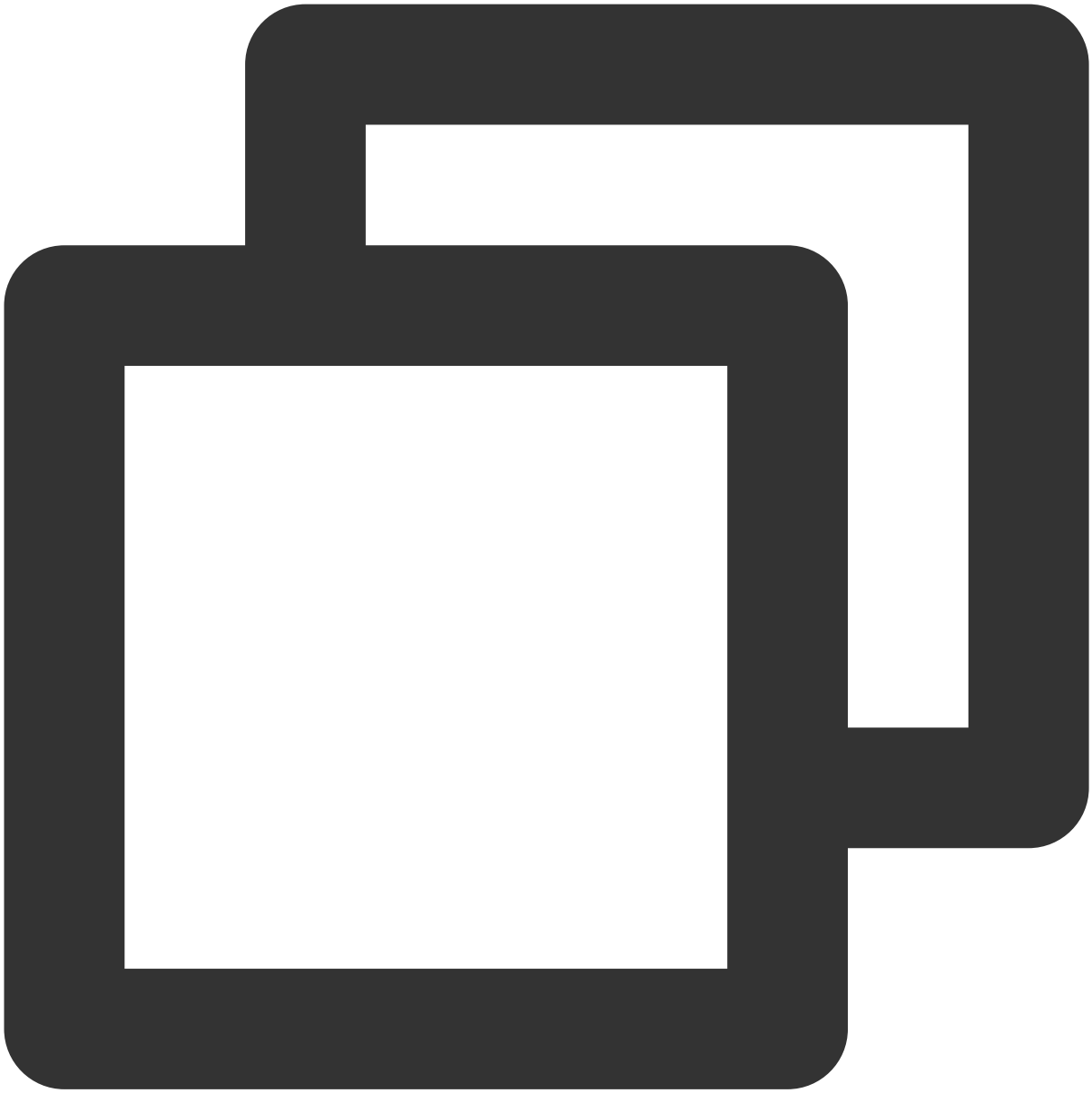
打开 DataX 解压目录下的 bin/datax.py 脚本，修改脚本中的 CLASS_PATH 变量为如下：



```
CLASS_PATH = ("%s/lib/*:%s/plugin/reader/hdfsreader/libs/*:%s/plugin/writer/hdfswri
```

在配置 JSON 文件里配置 hdfsreader 和 hdfswriter

示例 JSON 如下：



```
{
    "job": {
        "setting": {
            "speed": {
                "byte": 10485760
            },
            "errorLimit": {
                "record": 0,
                "percentage": 0.02
            }
        },
```

```
        "content": [{
            "reader": {
                "name": "hdfsreader",
                "parameter": {
                    "path": "testfile",
                    "defaultFS": "cosn://examplebucket-1250000000/",
                    "column": ["*"],
                    "fileType": "text",
                    "encoding": "UTF-8",
                    "hadoopConfig": {
                        "fs.cosn.impl": "org.apache.hadoop.fs.CosFileSystem",
                        "fs.cosn.userinfo.region": "ap-beijing",
                        "fs.cosn.tmp.dir": "/tmp/hadoop_cos",
                        "fs.cosn.userinfo.secretId": "COS_SECRETID",
                        "fs.cosn.userinfo.secretKey": "COS_SECRETKEY"
                    },
                    "fieldDelimiter": ","
                }
            },
            "writer": {
                "name": "hdfswriter",
                "parameter": {
                    "path": "/user/hadoop/",
                    "fileName": "testfile1",
                    "defaultFS": "cosn://examplebucket-1250000000/",
                    "column": [{
                            "name": "col",
                            "type": "string"
                        },
                        {
                            "name": "col1",
                            "type": "string"
                        },
                        {
                            "name": "col2",
                            "type": "string"
                        }
                    ],
                    "fileType": "text",
                    "encoding": "UTF-8",
                    "hadoopConfig": {
                        "fs.cosn.impl": "org.apache.hadoop.fs.CosFileSystem",
                        "fs.cosn.userinfo.region": "ap-beijing",
                        "fs.cosn.tmp.dir": "/tmp/hadoop_cos",
                        "fs.cosn.userinfo.secretId": "COS_SECRETID",
                        "fs.cosn.userinfo.secretKey": "COS_SECRETKEY"
                    },
```

```
                    "fieldDelimiter": ":",
                    "writeMode": "append"
                }
            }
        }]
    }
}
```

配置说明如下：

hadoopConfig 配置为 cosn 所需要的配置。

defaultFS 填写为 cosn 的路径，例如 `cosn://examplebucket-1250000000/` 。

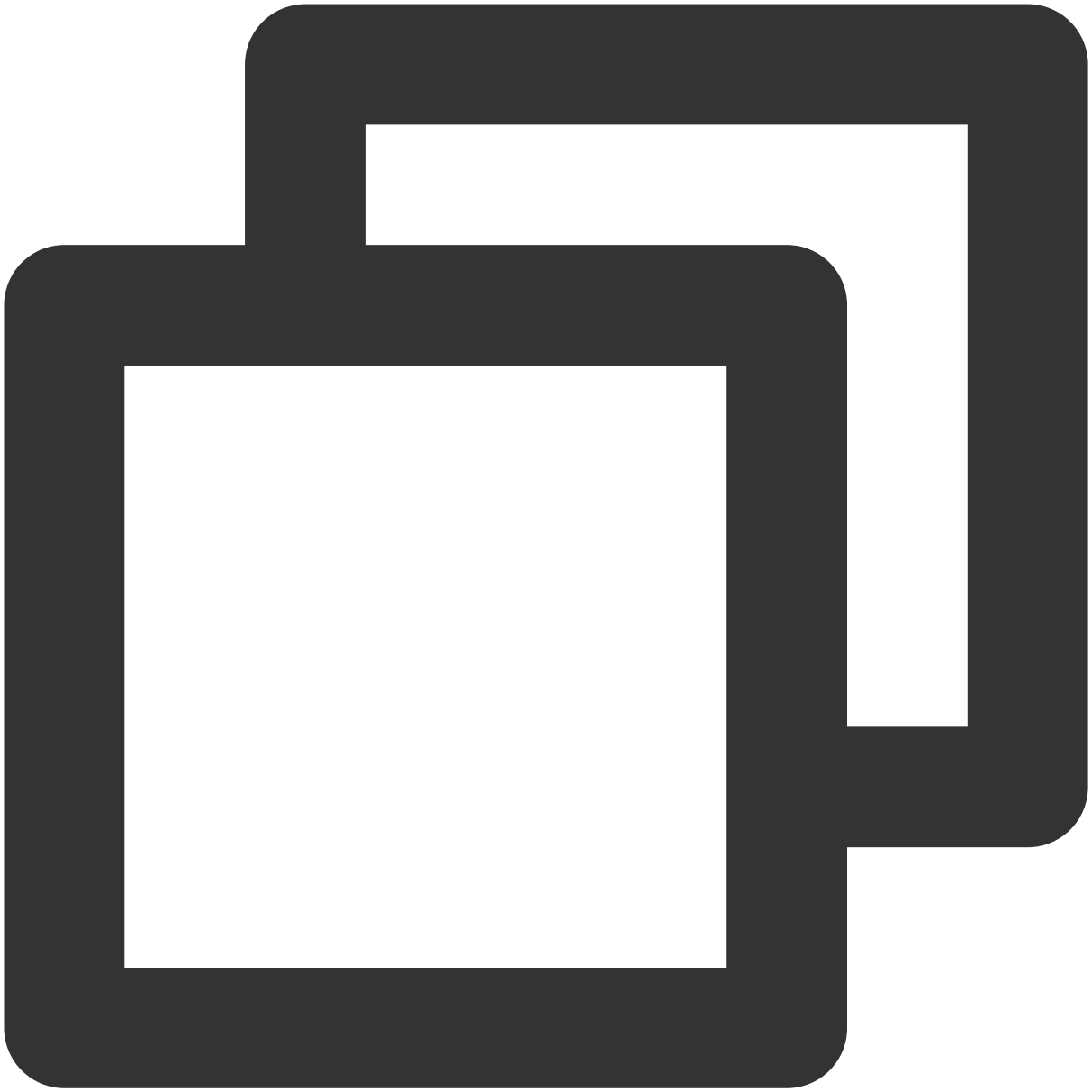
fs.cosn.userinfo.region 修改为存储桶所在的地域，例如 `ap-beijing` ，详情请参见 地域和访问域名。

COS_SECRETID 和 COS_SECRETKEY 修改为 COS 密钥。

其他配置同 hdfs 配置项即可。

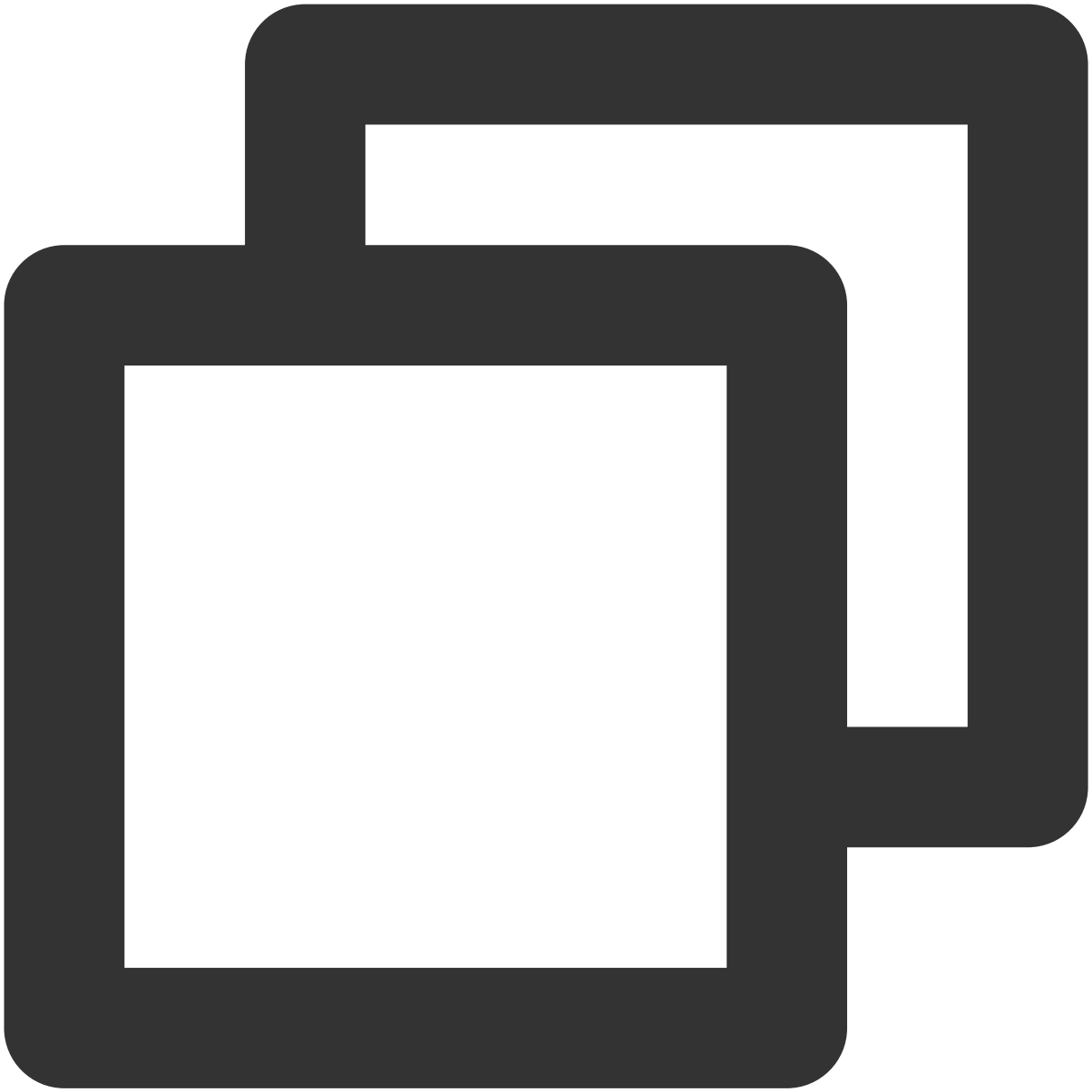
执行数据迁移

将配置文件保存为 hdfs_job.json，存放到 job 目录下，执行以下命令行：



```
bin/datax.py job/hdfs_job.json
```

观察屏幕正常输出如下：



```
2020-03-09 16:49:59.543 [job-0] INFO  JobContainer -
         [total cpu info] =>
                averageCpu                     | maxDeltaCpu                    | m
                -1.00%                         | -1.00%                         | -


         [total gc info] =>
                 NAME                 | totalGCCount       | maxDeltaGCCount    | m
                 PS MarkSweep         | 1                  | 1                  | 1
                 PS Scavenge          | 1                  | 1                  | 1
```

```
2020-03-09 16:49:59.543 [job-0] INFO  JobContainer - PerfTrace not enable!
2020-03-09 16:49:59.543 [job-0] INFO  StandAloneJobContainerCommunicator - Total 2
2020-03-09 16:49:59.544 [job-0] INFO  JobContainer -
任务启动时刻                    : 2020-03-09 16:49:48
任务结束时刻                    : 2020-03-09 16:49:59
任务总计耗时                    :                 11s
任务平均流量                    :               3B/s
记录写入速度                    :              0rec/s
读出记录总数                    :                   2
读写失败总数                    :                   0
```

CDH 配置 COSN 指引

最近更新时间：2024-01-06 10:54:03

简介

CDH（Cloudera's Distribution, including Apache Hadoop）是业界流行的 Hadoop 发行版本。本文指导如何在 CDH 环境下使用 COSN 存储服务，以实现大数据计算与存储分离，提供灵活及低成本的大数据解决方案。

说明

COSN 是 Hadoop-COS 文件系统的简称。

COSN 大数据组件支持情况如下：

组件名称	COSN 大数据组件支持情况	服务组件是否需要重启
Yarn	支持	重启 NodeManager
Hive	支持	重启 HiveServer 和 HiveMetastore
Spark	支持	重启 NodeManager
Sqoop	支持	重启 NodeManager
Presto	支持	重启 HiveServer 和 HiveMetastore 以及 Presto
Flink	支持	否
Impala	支持	否
EMR	支持	否
自建组件	后续支持	无
HBase	不推荐	无

版本依赖

本文依赖的组件版本如下：

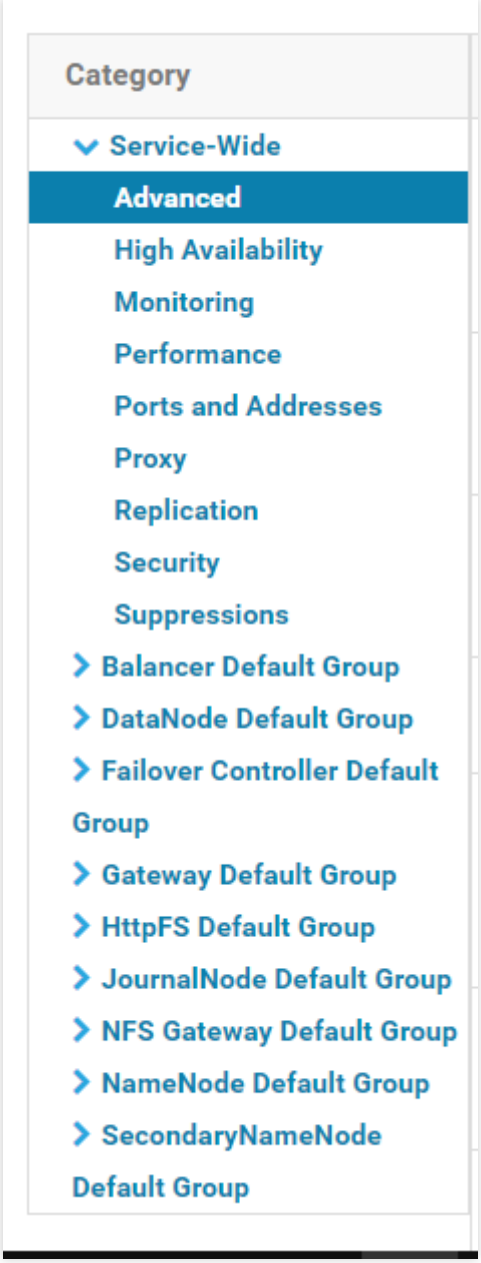
CDH 5.16.1

Hadoop 2.6.0

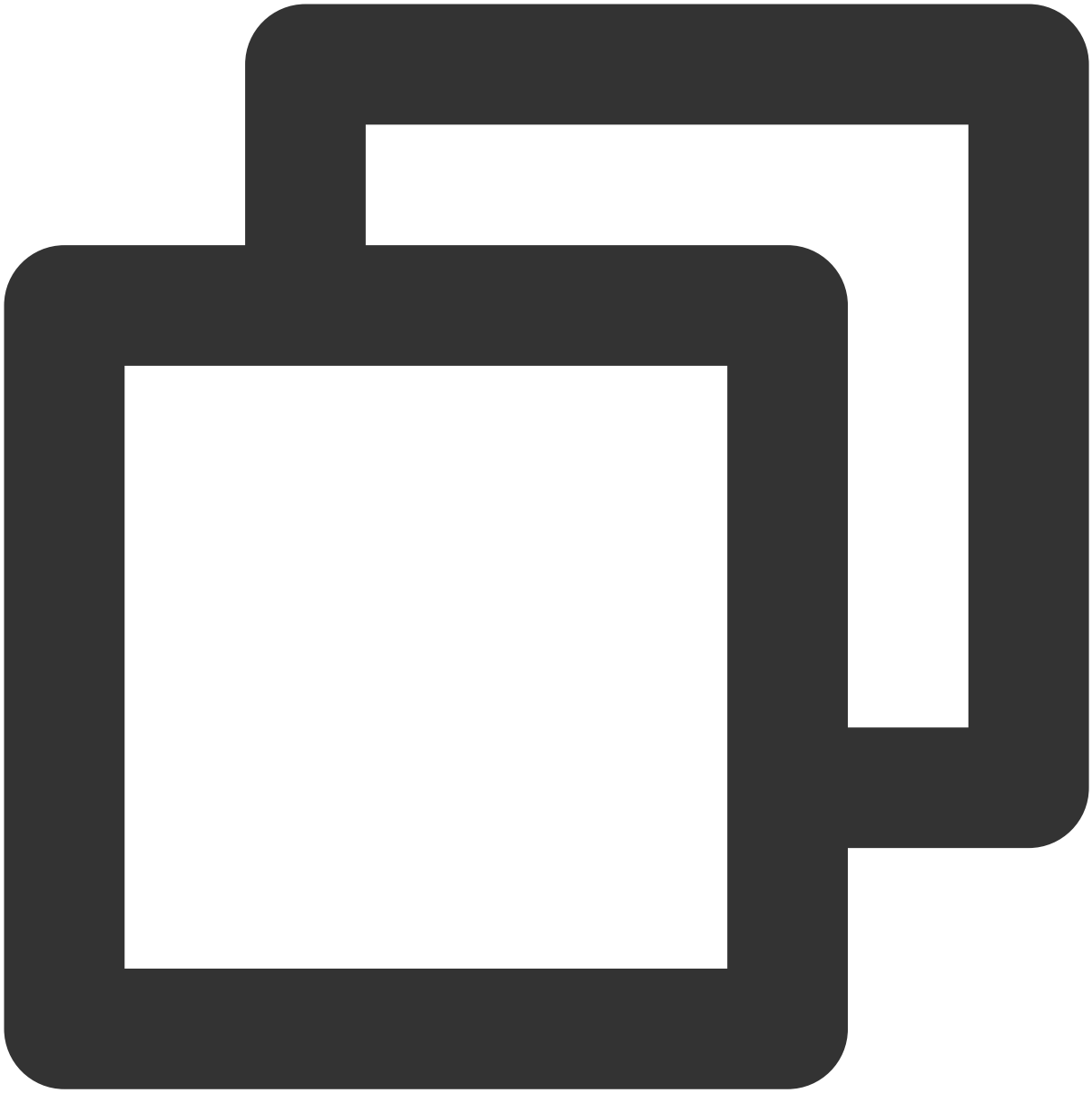
使用方法

存储环境配置

1. 登录 CDH 管理页面。
2. 在系统主页，选择**配置** > **服务范围** > **高级**，进入高级配置代码段页面，如下图所示：



3. 在 Cluster-wide Advanced Configuration Snippet(Safety Valve) for core-site.xml 的代码框中，填入 COSN 配置。



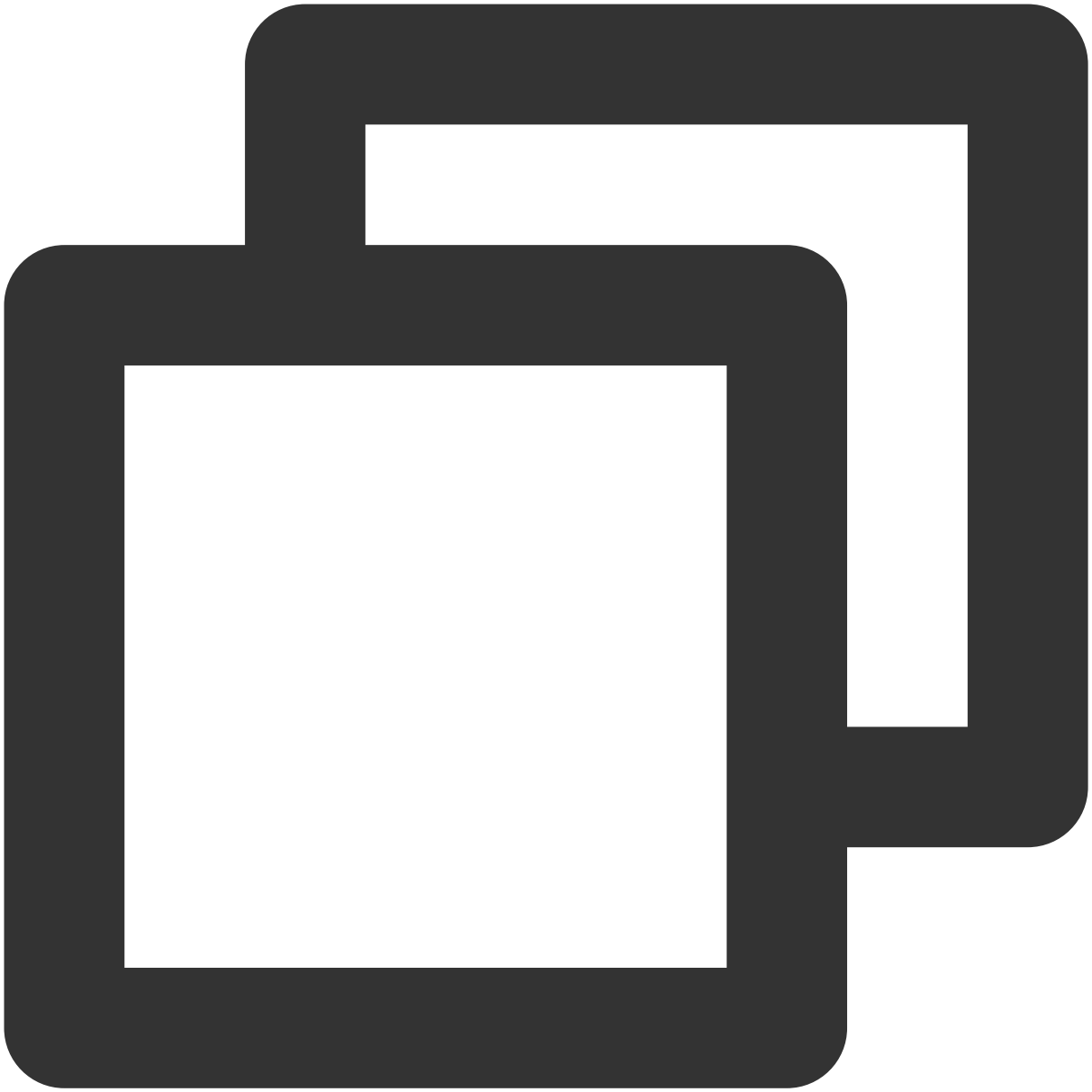
```
<property>
<name>fs.cosn.userinfo.secretId</name>
<value>AK***</value>
</property>
<property>
<name>fs.cosn.userinfo.secretKey</name>
<value></value>
</property>
<property>
<name>fs.cosn.impl</name>
<value>org.apache.hadoop.fs.CosFileSystem</value>
```

```
</property>
<property>
<name>fs.AbstractFileSystem.cosn.impl</name>
<value>org.apache.hadoop.fs.CosN</value>
</property>
<property>
<name>fs.cosn.bucket.region</name>
<value>ap-shanghai</value>
</property>
```

以下为必选的 COSN 配置项（需添加到 core-site.xml 中），COSN 其他配置可参见 Hadoop 工具 文档。

COSN 配置项	值	含义
fs.cosn.userinfo.secretId	AKxxxx	账户的 API 密钥信息
fs.cosn.userinfo.secretKey	Wpxxxx	账户的 API 密钥信息
fs.cosn.bucket.region	ap-shanghai	用户存储桶所在地域
fs.cosn.impl	org.apache.hadoop.fs.CosFileSystem	cosn 对 FileSystem 的实现类，固定为 org.apache.hadoop.fs.CosFileSysten
fs.AbstractFileSystem.cosn.impl	org.apache.hadoop.fs.CosN	cosn 对 AbstractFileSystem 的实现类，固定为 org.apache.hadoop.fs.CosN

4. 对 HDFS 服务进行操作，单击部署客户端配置，此时以上 core-site.xml 配置会更新到集群里的机器上。
5. 将 COSN 最新的 SDK 包，放置到 CDH HDFS 服务的 jar 包路径下，请根据实际值进行替换，示例如下：



```
cp hadoop-cos-2.7.3-shaded.jar /opt/cloudera/parcels/CDH-5.16.1-1.cdh5.16.1.p0.3/li
```

注意

在集群中的每台机器都需要在相同的位置放置 SDK 包。

数据迁移

使用 Hadoop Distcp 工具将 CDH HDFS 数据迁移到 COSN，详情请参见 Hadoop 文件系统与 COS 之间的数据迁移。

大数据套件使用 COSN

1. MapReduce

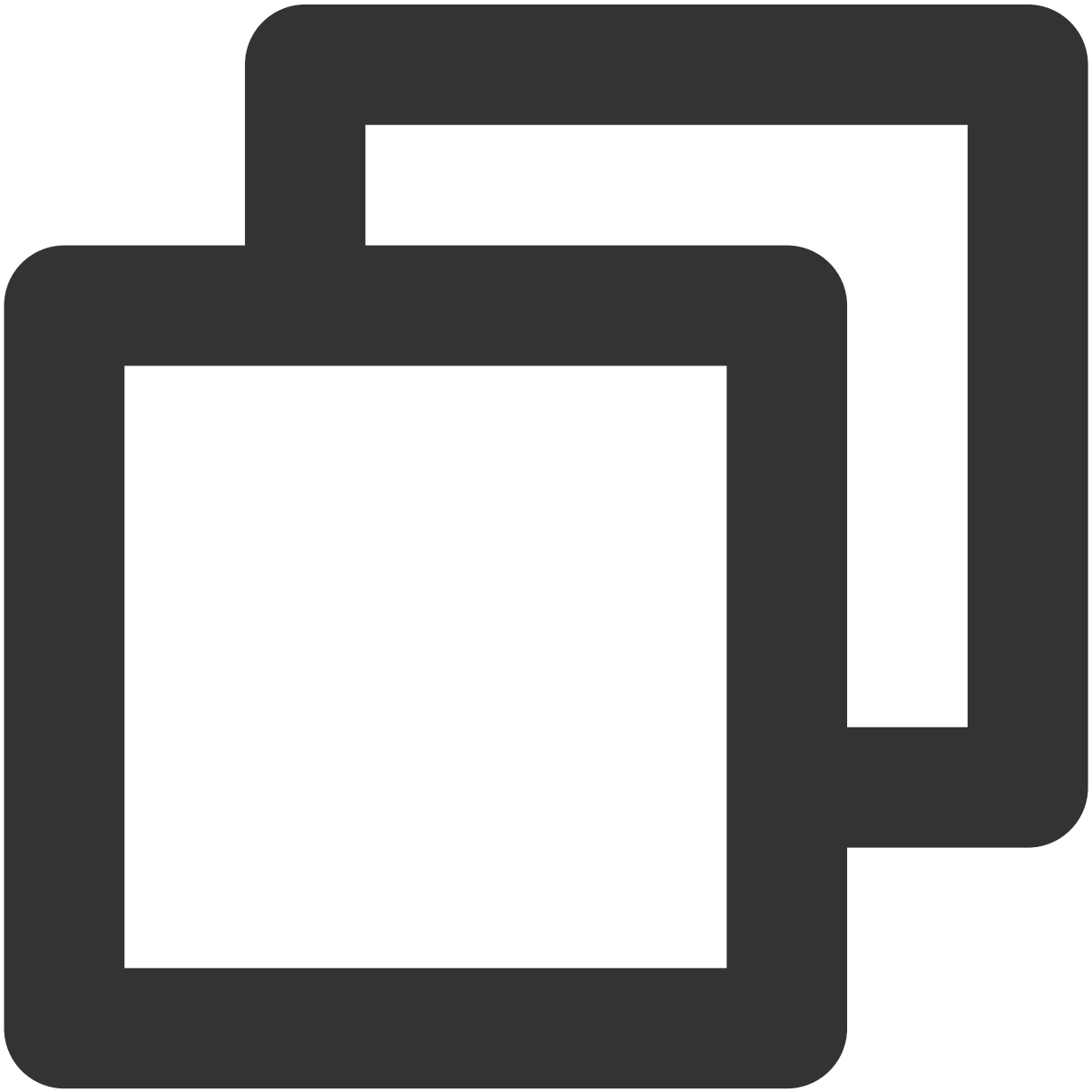
操作步骤

（1）按照 数据迁移 章节，配置好 HDFS 的相关配置，并将 COSN 的 SDK jar 包，放置到 HDFS 相应的目录。

（2）在 CDH 系统主页，找到 YARN，重启 NodeManager 服务（TeraGen 命令可以不用重启，但是 TeraSort 由于业务内部逻辑，需要重启 NodeManger，建议都统一重启 NodeManager 服务）。

示例

下面以 Hadoop 标准测试中的 TeraGen 和 TeraSort 为例：



```
hadoop jar ./hadoop-mapreduce-examples-2.7.3.jar teragen  -Dmapred.job.maps=500  -D

hadoop jar ./hadoop-mapreduce-examples-2.7.3.jar terasort -Dmapred.max.split.size=1
```

说明

`cosn://    schema` 后面请替换为用户大数据业务的存储桶路径。

2. Hive

2.1 MR 引擎

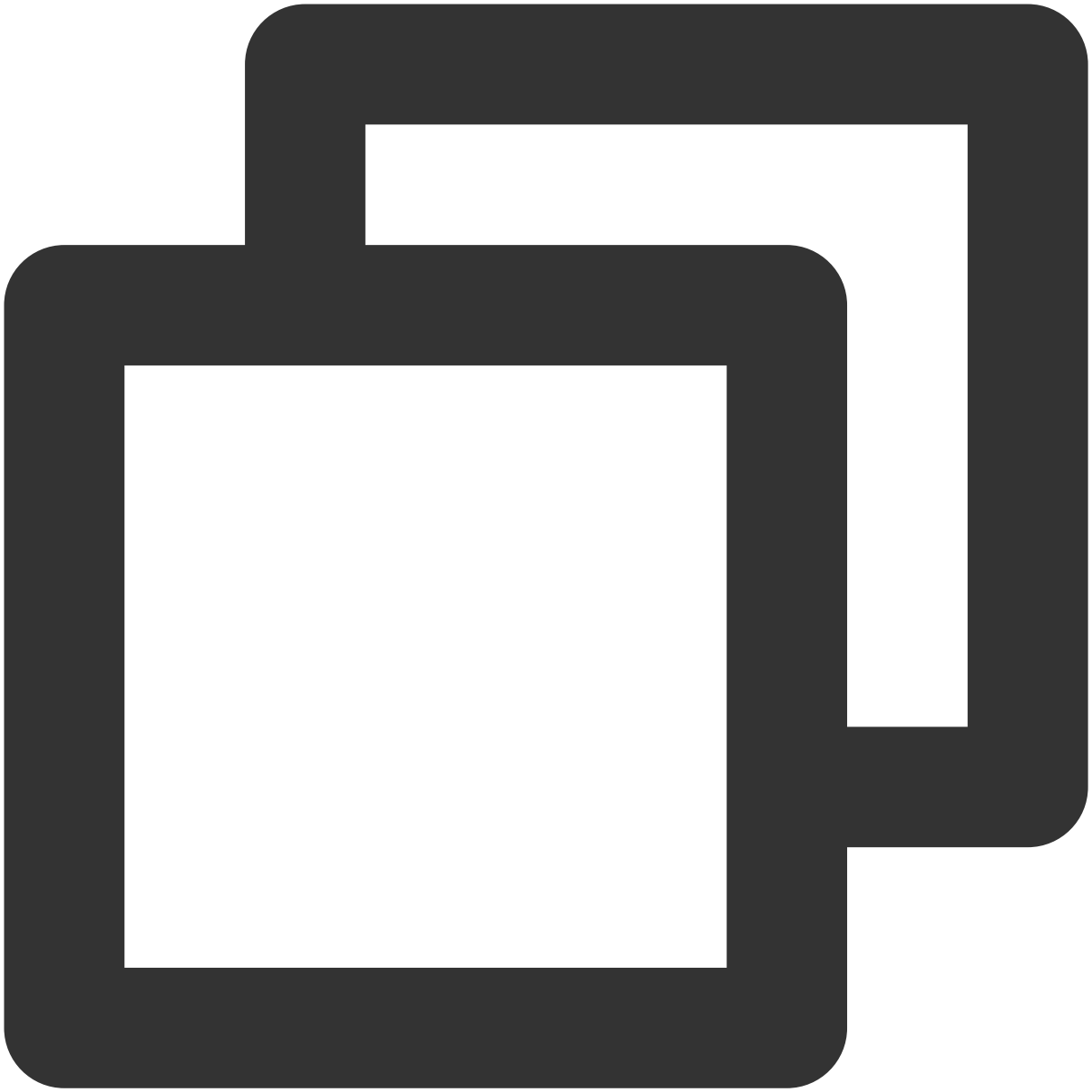
操作步骤

（1）按照 数据迁移 章节，配置好 HDFS 的相关配置，并且将 COSN 的 SDK jar 包，放置到 HDFS 相应的目录。

（2）在 CDH 主页面，找到 HIVE 服务， 重启 Hiveserver2 及 HiverMetastore 角色。

示例

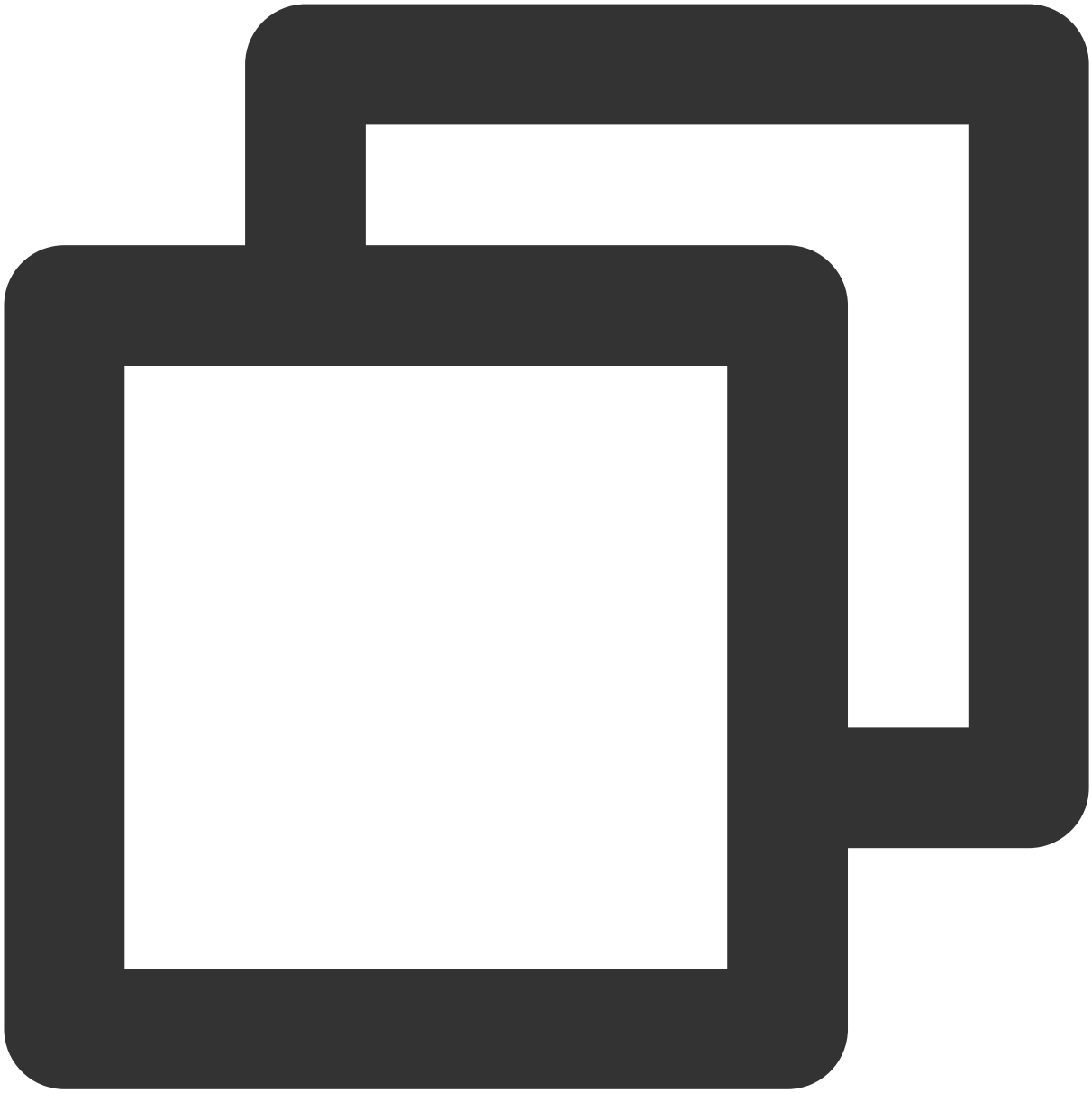
某用户的真实业务查询，例如执行 Hive 命令行，创建一个 Location，作为在 CHDFS 上的分区表：



```
CREATE TABLE `report.report_o2o_pid_credit_detail_grant_daily`(
  `cal_dt` string,
  `change_time` string,
  `merchant_id` bigint,
  `store_id` bigint,
  `store_name` string,
  `wid` string,
  `member_id` bigint,
  `meber_card` string,
  `nickname` string,
  `name` string,
```

```
  `gender` string,
  `birthday` string,
  `city` string,
  `mobile` string,
  `credit_grant` bigint,
  `change_reason` string,
  `available_point` bigint,
  `date_time` string,
  `channel_type` bigint,
  `point_flow_id` bigint)
PARTITIONED BY (
  `topicdate` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.orc.OrcSerde'
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcInputFormat'
    OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat'
LOCATION
  'cosn://examplebucket-1250000000/user/hive/warehouse/report.db/report_o2o_pid_cre
TBLPROPERTIES (
  'last_modified_by'='work',
  'last_modified_time'='1589310646',
  'transient_lastDdlTime'='1589310646')
```

执行 sql 查询：



```
select count(1) from report.report_o2o_pid_credit_detail_grant_daily;
```

观察结果如下：

```
hive> select count(1) from report.report_o2o_pid_credit_detail_grant_daily;
Query ID = root_20200713121818_3a911a0c-2496-4e7e-b59d-b2a26266a6ab
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1594351711155_0014, Tracking URL = http://hadoop01:8088/prox
y/application_1594351711155_0014/
Kill Command = /opt/cloudera/parcels/CDH-5.16.1-1.cdh5.16.1.p0.3/lib/hadoop/bin
/hadoop job  -kill job_1594351711155_0014
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2020-07-13 12:18:19,189 Stage-1 map = 0%,  reduce = 0%
2020-07-13 12:18:25,391 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 8.89 s
ec
2020-07-13 12:18:30,544 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 10.8
 sec
MapReduce Total cumulative CPU time: 10 seconds 800 msec
Ended Job = job_1594351711155_0014
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 10.8 sec   HDFS Read: 17383
HDFS Write: 6 SUCCESS
Total MapReduce CPU Time Spent: 10 seconds 800 msec
OK
27677
Time taken: 19.128 seconds, Fetched: 1 row(s)
```

2.2 Tez 引擎

Tez 引擎需要将 COSN 的 jar 包导入到 Tez 的压缩包内，下面以 apache-tez.0.8.5 为例进行说明：

操作步骤

（1）找到 CDH 集群安装的 tez 包，然后解压，例如/usr/local/service/tez/tez-0.8.5.tar.gz。

（2）将 COSN 的 jar 包放置到解压后的目录下，然后重新压缩输出一个压缩包。

（3）将新的压缩包上传到 tez.lib.uris 指定的路径下（如果之前存在路径则直接替换即可）。

（4）在 CDH 主页面，找到 HIVE，重启 hiveserver 和 hivemetastore。

3. Spark

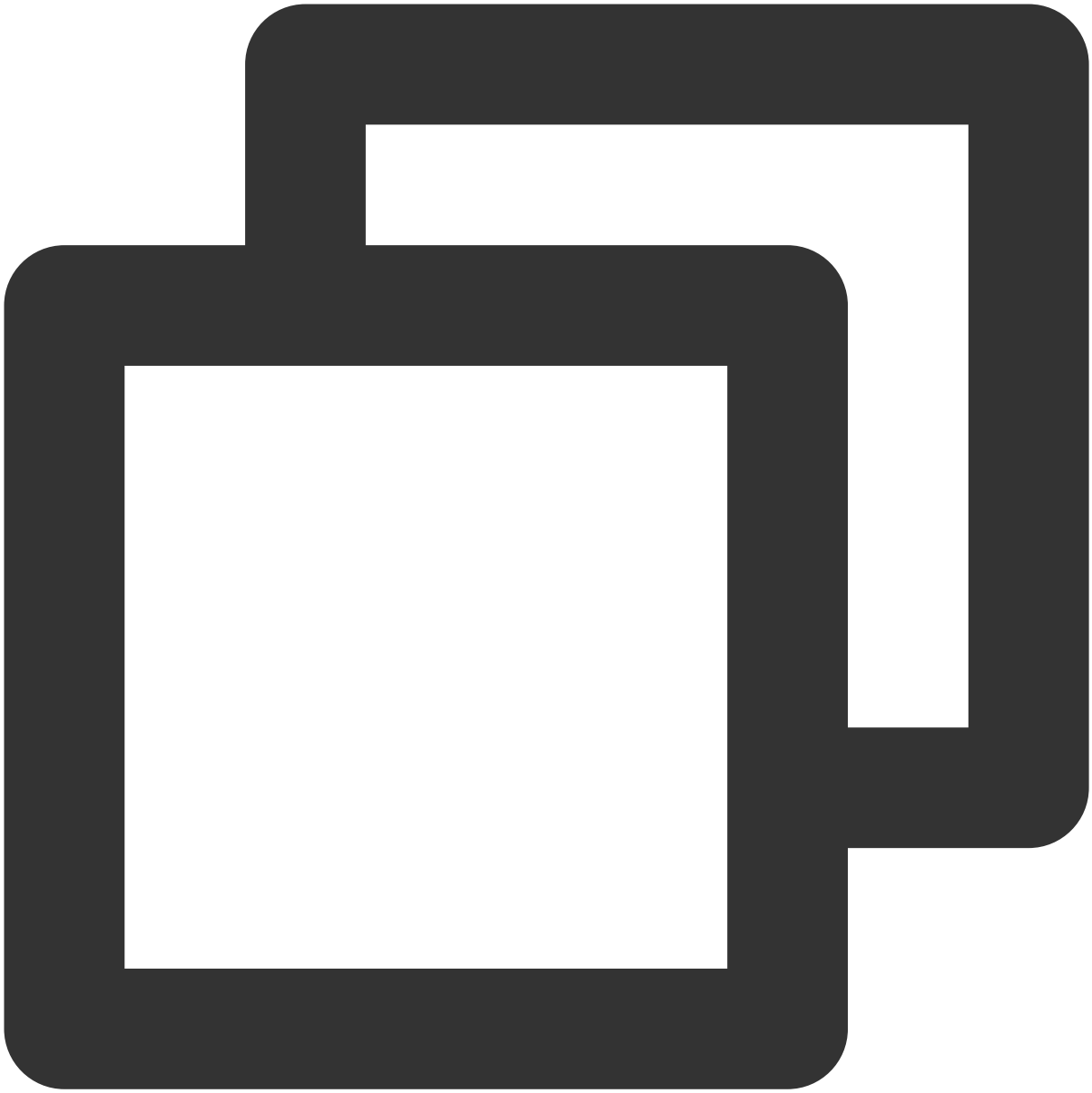
操作步骤

（1）按照 数据迁移 章节，配置好 HDFS 的相关配置，并且将 COSN 的 SDK jar 包，放置到 HDFS 相应的目录。

（2）重启 NodeManager 服务。

示例

以 COSN 进行 Spark example word count 测试为例。



```
spark-submit  --class org.apache.spark.examples.JavaWordCount --executor-memory 4g
```

执行结果如下：

```
20/07/17 20:39:03 INFO cluster.YarnScheduler: Removed TaskSet 1.0, whose tasks have al
20/07/17 20:39:03 INFO scheduler.DAGScheduler: ResultStage 1 (collect at JavaWordCount
20/07/17 20:39:03 INFO scheduler.DAGScheduler: Job 0 finished: collect at JavaWordCoun
And: 4
day.: 1
God: 4
Let: 1
light:: 1
it: 1
light,: 1
evening: 1
was: 2
that: 1
light.: 1
be: 1
Day,: 1
said,: 1
darkness.: 1
he: 1
first: 1
there: 2
light: 2
Night.: 1
morning: 1
darkness: 1
were: 1
saw: 1
divided: 1
and: 4
good:: 1
from: 1
called: 2
the: 8
20/07/17 20:39:03 INFO ui.SparkUI: Stopped Spark web UI at http://10.0.0.42:4040
```

4. Sqoop

操作步骤

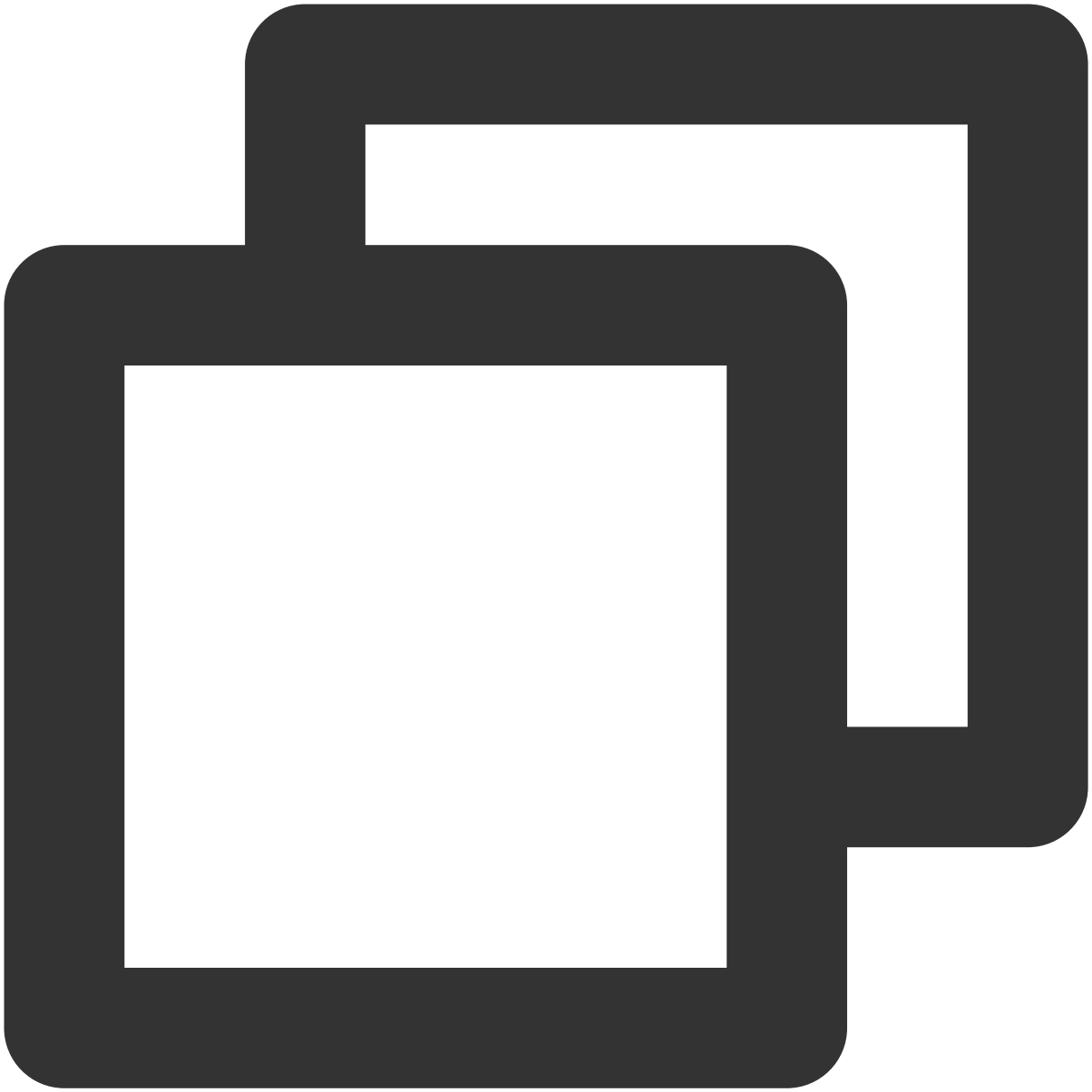
（1）按照 数据迁移 章节，配置好 HDFS 的相关配置，并且将 COSN 的 SDK jar 包，放置到 HDFS 相应的目录。

（2）COSN 的 SDK jar 包还需要放到 sqoop 目录下（例如/opt/cloudera/parcels/CDH-5.16.1-1.cdh5.16.1.p0.3/lib/sqoop/）。

（3）重启 NodeManager 服务。

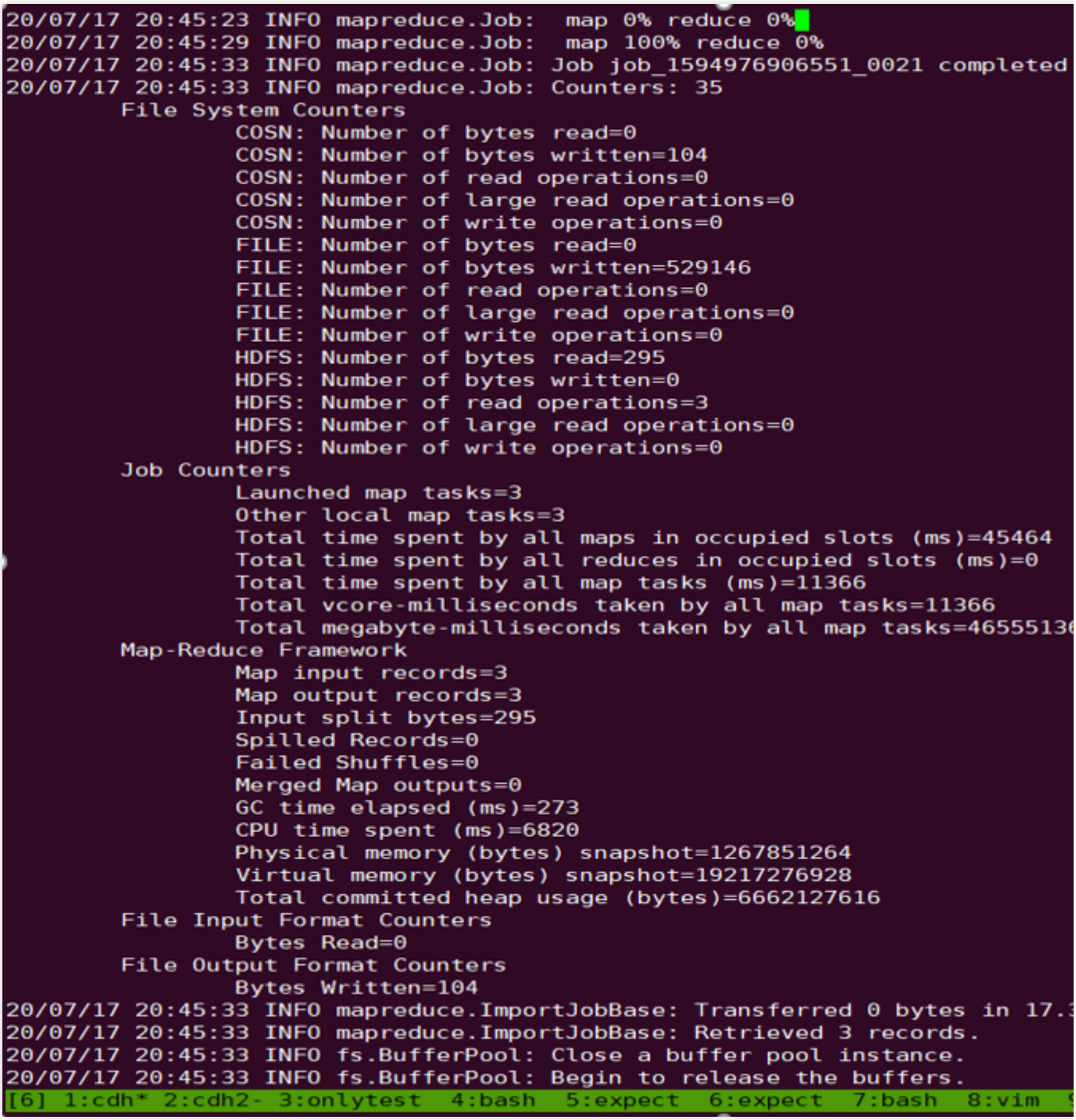
示例

以导出 MYSQL 表到 COSN 为例，可参考 关系型数据库和 HDFS 的导入导出 文档进行测试。



```
sqoop import --connect "jdbc:mysql://IP:PORT/mysql" --table sqoop_test --username r
```

执行结果如下：



5. Presto

操作步骤

（1）按照 数据迁移 章节，配置好 HDFS 的相关配置，并且将 COSN 的 SDK jar 包，放置到 HDFS 相应的目录。

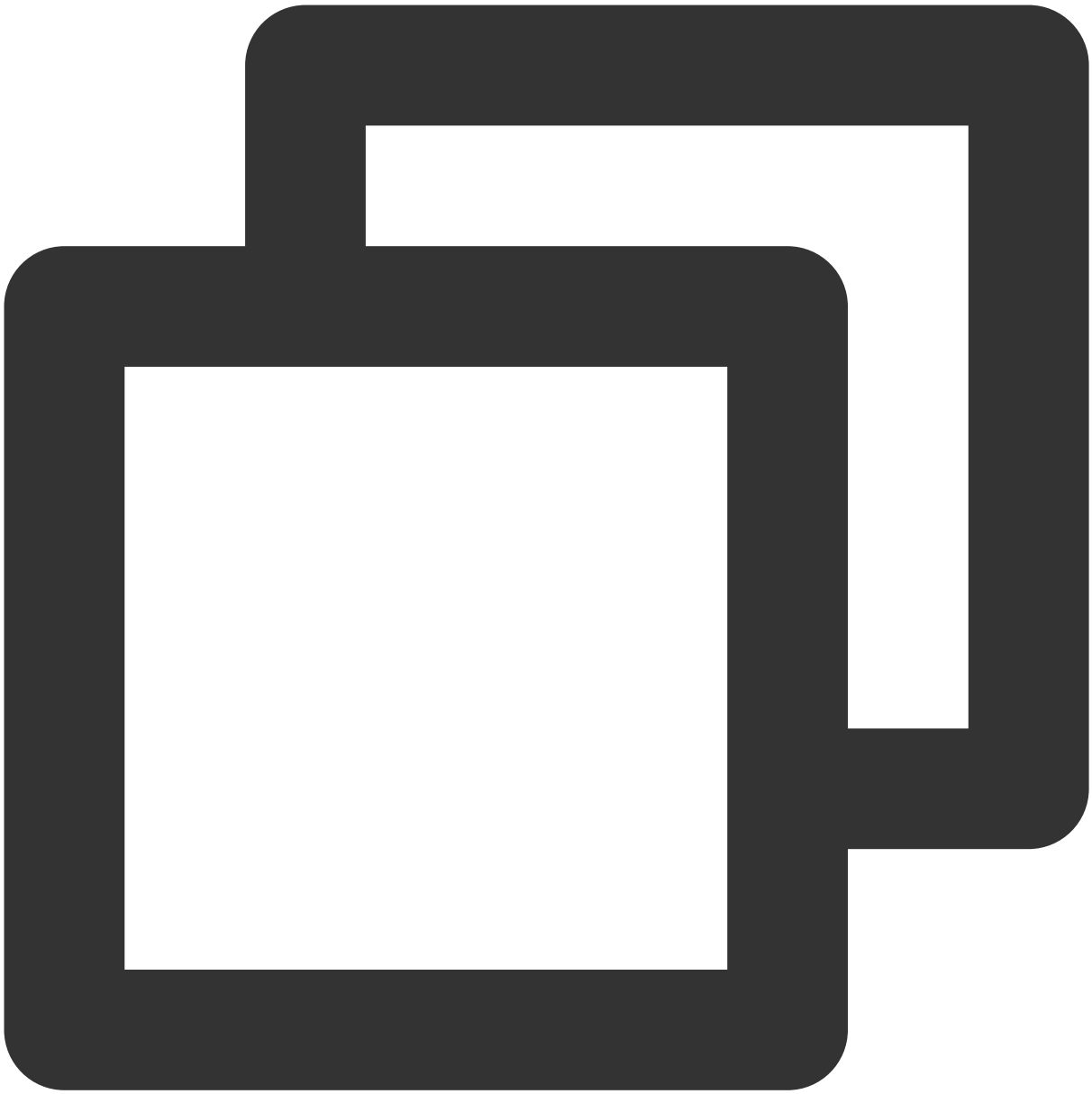
（2）COSN 的 SDK jar 包还需要放到 presto 目录下（例如/usr/local/services/cos_presto/plugin/hive-hadoop2）。

（3）由于 presto 不会加载 hadoop common 下的 gson-2...jar，需将 gson-2...jar 也放到 presto 目录下（例如 /usr/local/services/cos_presto/plugin/hive-hadoop2，仅 CHDFS 依赖 gson）。

（4）重启 HiveServer、HiveMetaStore 和 Presto 服务。

示例

以 HIVE 创建 Location 为 COSN 的表查询为例：

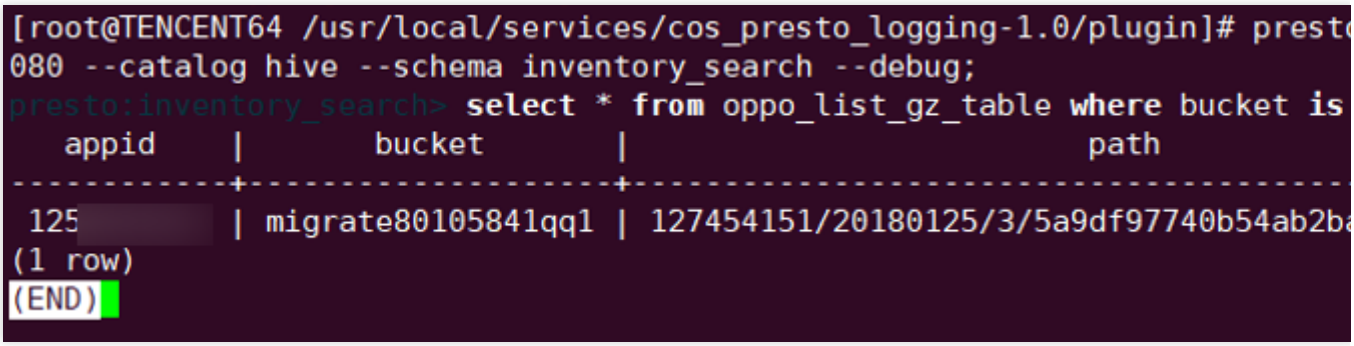


```
select * from cosn_test_table where bucket is not null limit 1;
```

说明

cosn_test_table 为 location 是 cosn scheme 的表。

查询结果如下：


[root@TENCENT64 /usr/local/services/cos_presto_logging-1.0/plugin]# presto
080 --catalog hive --schema inventory_search --debug;
presto:inventory_search> select * from oppo_list_gz_table where bucket is
appid | bucket | path
125 | migrate80105841qq1 | 127454151/20180125/3/5a9df97740b54ab2b
(1 row)
(END)

COS Ranger 权限体系解决方案

最近更新时间：2024-01-06 10:54:03

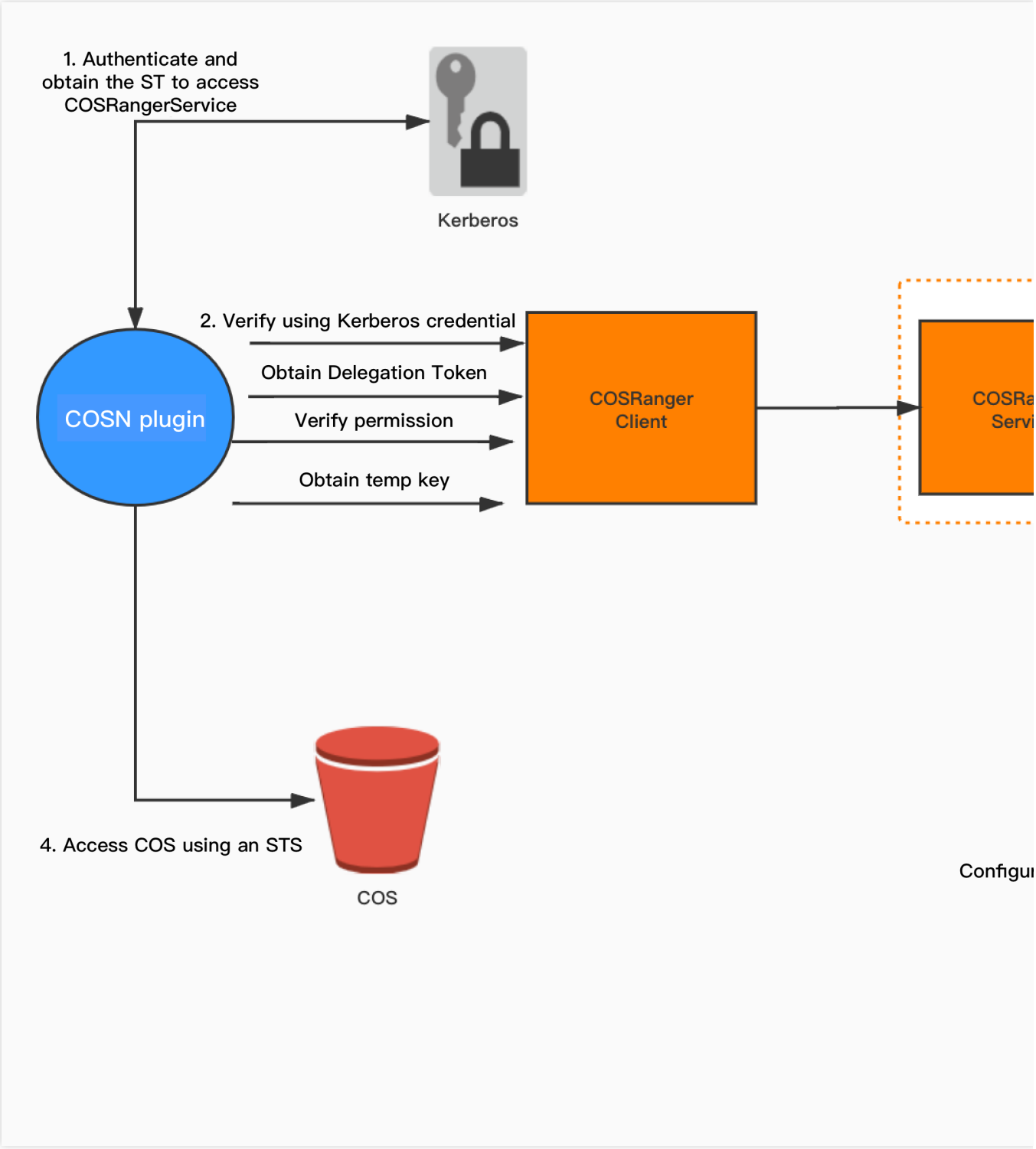
背景

Hadoop Ranger 权限体系是大数据场景下的权限解决方案。用户使用存算分离后，将数据托管在对象存储（Cloud Object Storage，COS）上。COS 使用的是腾讯云访问管理（Cloud Access Management，CAM）权限体系，无论是用户身份，权限策略等，都与本地 Hadoop Ranger 体系不同。为维持客户的使用习惯，我们提供 COS 的 Ranger 接入解决方案。

优势

细粒度的权限控制，兼容 Hadoop 权限逻辑，用户统一管理大数据组件与云端托管存储的权限。
插件侧无需在 core-site 中设置密钥，密钥统一在 COS Ranger Service 中设置，避免明文密钥的泄露。

解决方案架构



Hadoop 权限体系中, 认证由 Kerberos 提供，授权鉴权由 Ranger 负责。在此基础上，我们提供以下组件，来支持 COS 的 Ranger 权限方案。

1. COS Ranger Plugin：提供 Ranger 服务端的服务定义插件。它们提供了 Ranger 侧的 COS 服务描述，包括权限种类，必要参数定义（例如 COS 的 bucket 参数和 region 参数）。部署了该插件后，用户即可在 Ranger 的控制页面上，填写相应的权限策略。

2. COS Ranger Service：该服务集成了 Ranger 的客户端，周期性从 Ranger 服务端同步权限策略，在收到客户的鉴权请求后，在本地进行权限校验。 同时它提供了 Hadoop 中 DelegationToken 相关的生成，续租等接口，所有的接

口都是通过 Hadoop IPC 定义。
3. COS Ranger Client：COSN 插件对其进行动态加载，把权限校验的请求转发给 COS Ranger Service。

部署环境

Hadoop 环境。
ZooKeeper、Ranger、Kerberos 服务（如果有认证需求，则部署）。
说明
以上服务由于是成熟的开源组件，因此客户可自行安装。

部署组件

部署组件请按照 CHDFS Ranger Plugin、COS Ranger Service、COS Ranger Client、COSN 次序进行。
部署 COS Ranger Plugin
部署 COS Ranger Service
部署 COS Ranger Client
部署 COSN
COS-Ranger-Plugin 拓展了 Ranger Admin 控制台上的服务种类，用户可在 Ranger 控制台上，设置和 COS 相关的操作权限。

代码地址

可前往 Github 的 ranger-plugin 目录下获取。

版本

V1.0 版本及以上。

部署步骤

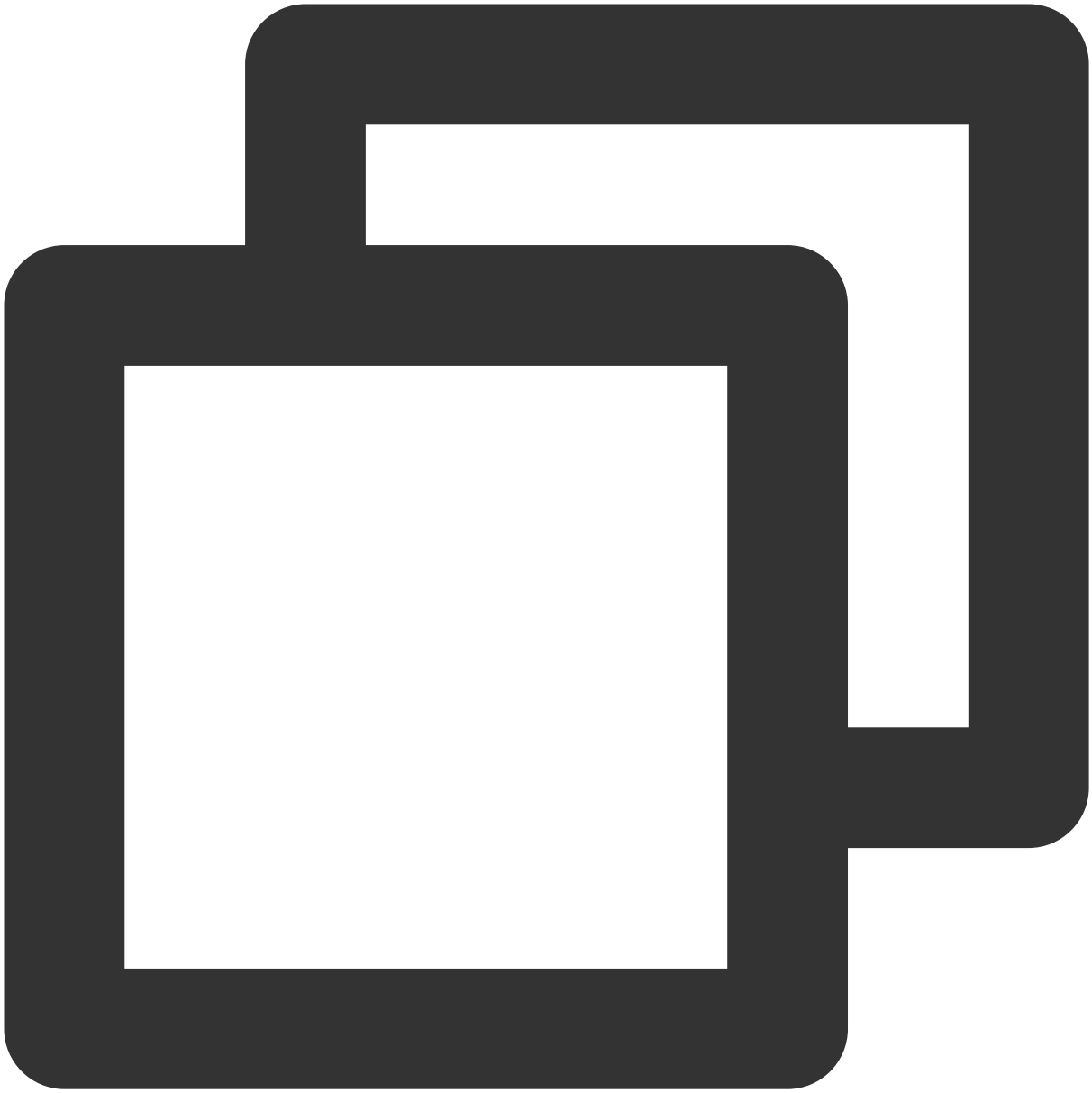
1. 在 Ranger 的服务定义目录下新建 COS 目录（注意，目录权限需要保证至少有 x 与 r 权限）。
a. 腾讯云的 EMR 环境，路径是 ranger/ews/webapp/WEB-INF/classes/ranger-plugins。
b. 自建的 hadoop 环境，可以通过在 ranger 目录下查找 hdfs 等已经接入到 ranger 服务的组件，查找目录位置。

```
[hadoop@10 ranger-plugins]$ ls -l
total 68
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 atl
drwxr-xr-x 2 hadoop hadoop 4096 Dec 15 10:57 cho
drwxr-xr-x 2 hadoop hadoop 4096 Dec 15 10:57 cos
drwxr-xr-x 2 hadoop hadoop 4096 Feb 25  2020 hba
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 hdf
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 hiv
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 kaf
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 kms
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 knc
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 kyl
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 nif
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 nif
drwxr-xr-x 2 hadoop hadoop 4096 Aug  5 20:48 pre
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 sol
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 sqo
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 sto
drwxr-xr-x 2 hadoop hadoop 4096 Feb 19  2020 yar
```

2. 在 COS 目录下，放入 cos-chdfs-ranger-plugin-xxx.jar。（注意 jar 包至少有 r 权限）。同时需要放入 cos-ranger.json 文件，可前往 Github 获取。

3. 重启 Ranger 服务。

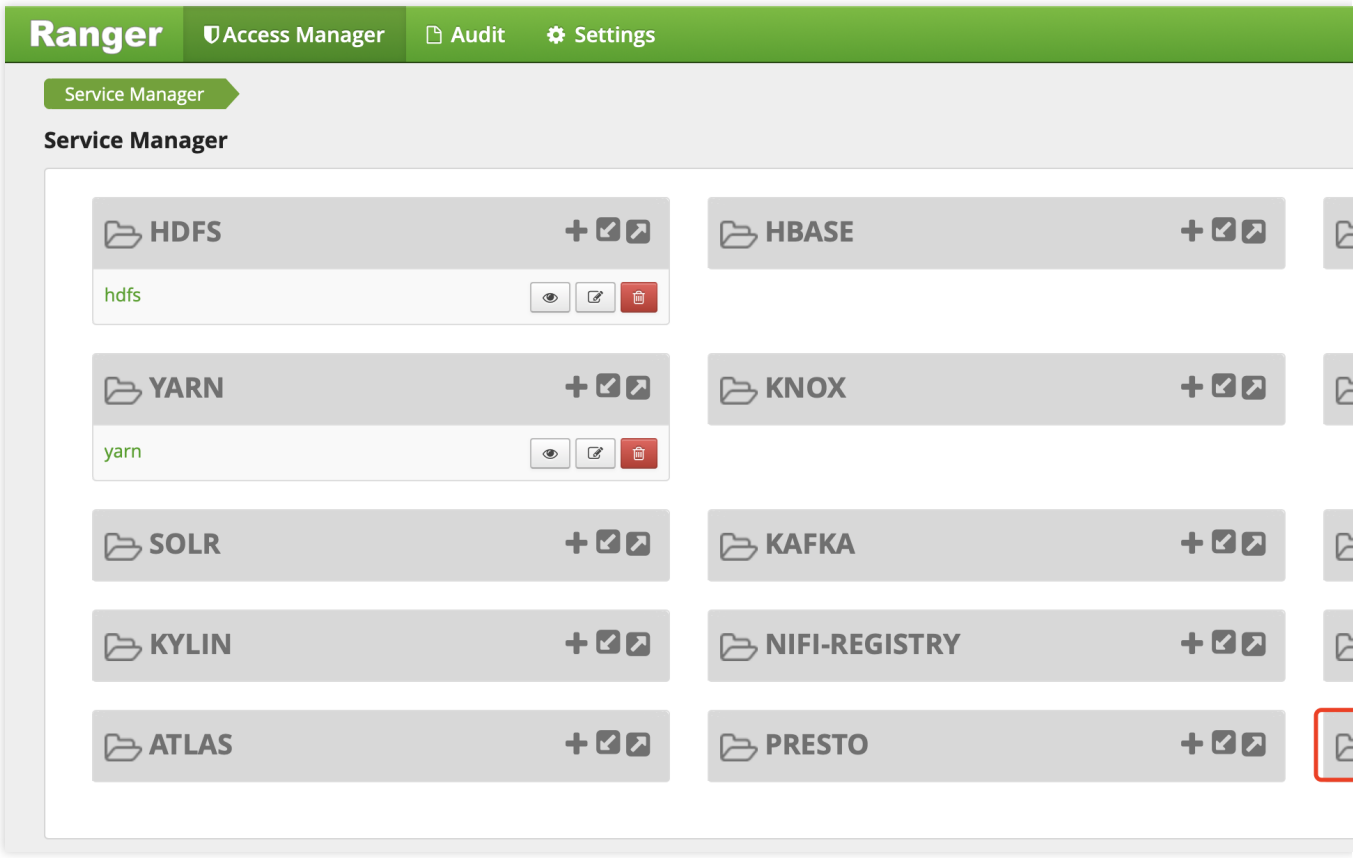
4. 在 Ranger 上注册 COS Service。可参考如下命令：



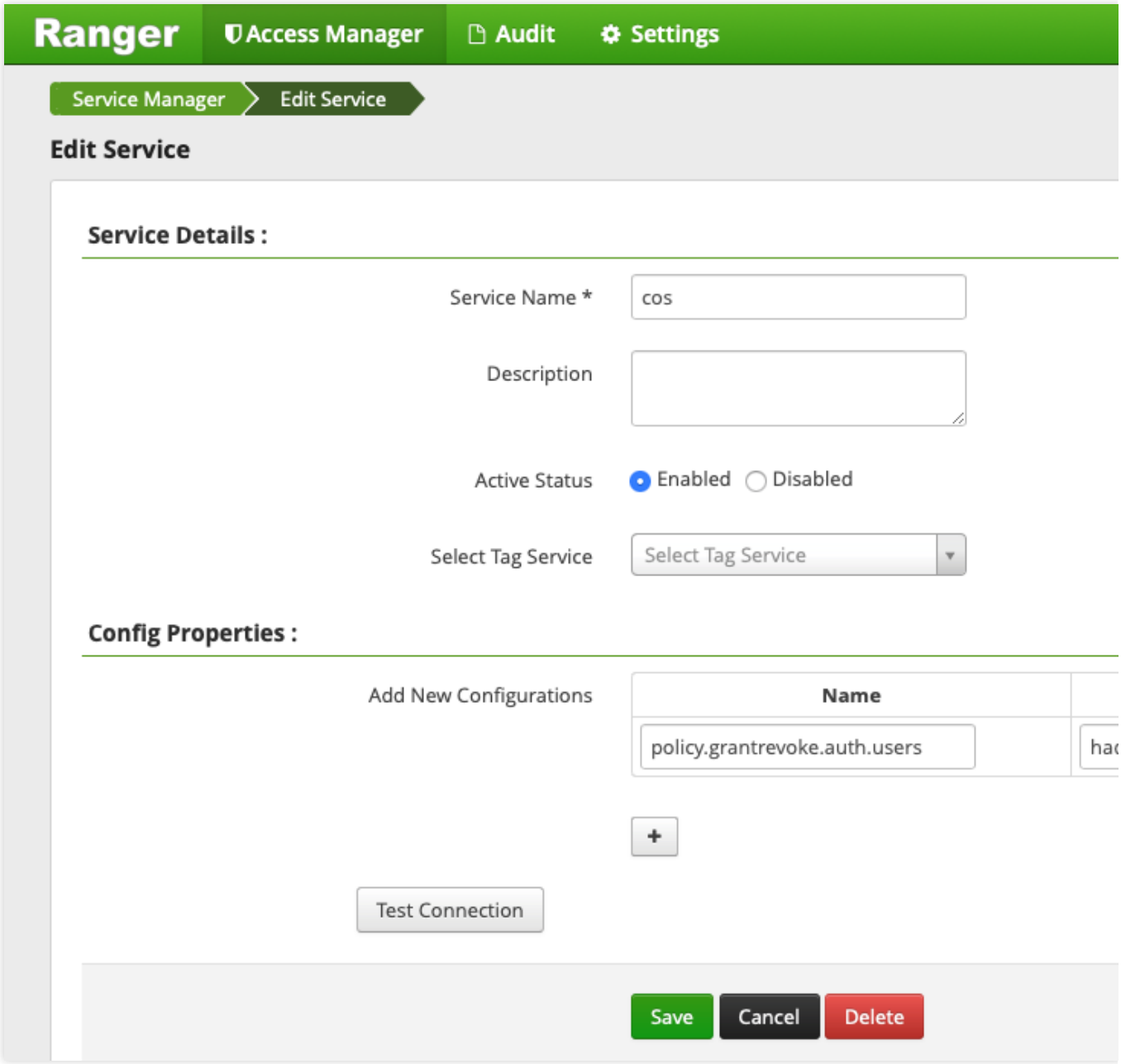
```
##生成服务，需传入 Ranger 管理员账号密码，以及 Ranger 服务的地址。
##对于腾讯云 EMR 集群，管理员用户是 root，密码是构建 emr 集群时设置的 root 密码，ranger 服务的
adminUser=root
##构建 EMR 集群时设置的密码，也是 ranger 服务 web 页面的登录密码
adminPasswd=xxxxxx
##如果 ranger 服务有多个 master 节点，任选一个 master 即可
rangerServerAddr=10.0.0.1:6080
##命令行中 -d 指定步骤 2 中的 json 文件
curl -v -u${adminUser}:${adminPasswd} -X POST -H "Accept:application/json" -H "Cont
##如果要删除刚定义的服务，则传入刚刚创建服务时，返回的服务 ID
serviceId=102
```

```
curl -v -u${adminUser}:${adminPasswd} -X DELETE -H "Accept:application/json" -H "Co
```

5. 创建服务成功后，可在 Ranger 控制台看到 COS 服务。如下所示：

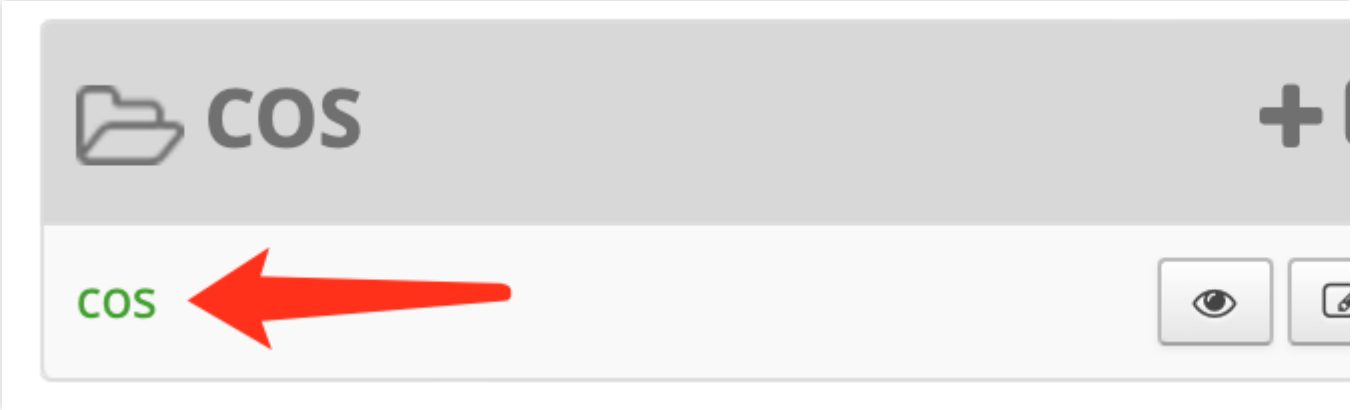


6. 在 COS 服务侧单击 +，定义新服务实例，服务实例名可自定义，例如 `cos` 或者 `cos_test` ，服务的配置如下所示。

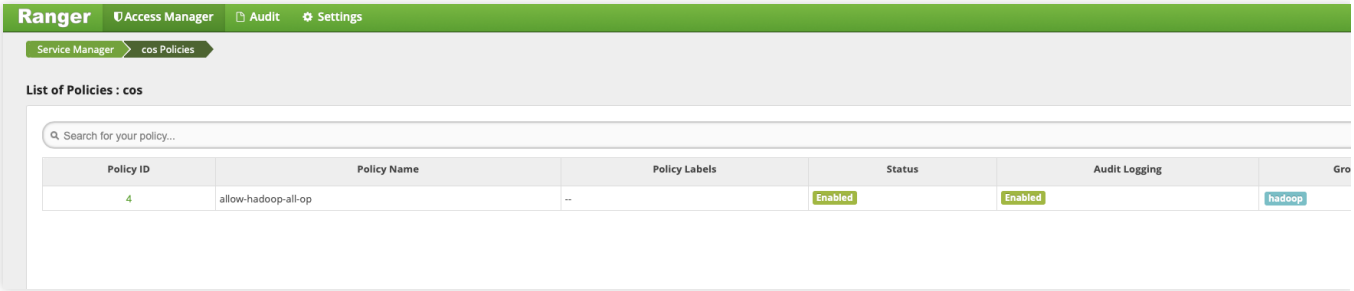


其中 policy.grantrevoke.auth.users 需设置后续启动 COS Ranger Service 服务的用户名（即允许拉取权限策略的用户）。通常建议设置成 hadoop，后续 COS Ranger Service 可使用此用户名进行启动。

7. 单击新生成的 COS 服务实例。



添加 policy，如下所示：



8. 在跳转界面中，配置以下参数，说明如下：

bucket：存储桶名称，例如 examplebucket-1250000000，可登录 COS 控制台 查看。

path：COS 对象路径。注意 COS 的对象路径不以/开始。

include：表示设置的权限适用于 path 本身，还是除了 path 以外的其他路径。

recursive：表示权限不仅适用于 path，还适用于 path 路径下的子成员（即递归子成员）。通常用于 path 设置为目录的情况。

user/group：用户名和用户组。这里是或的关系，即用户名或者用户组满足其中一个，即可拥有对应的操作权限。

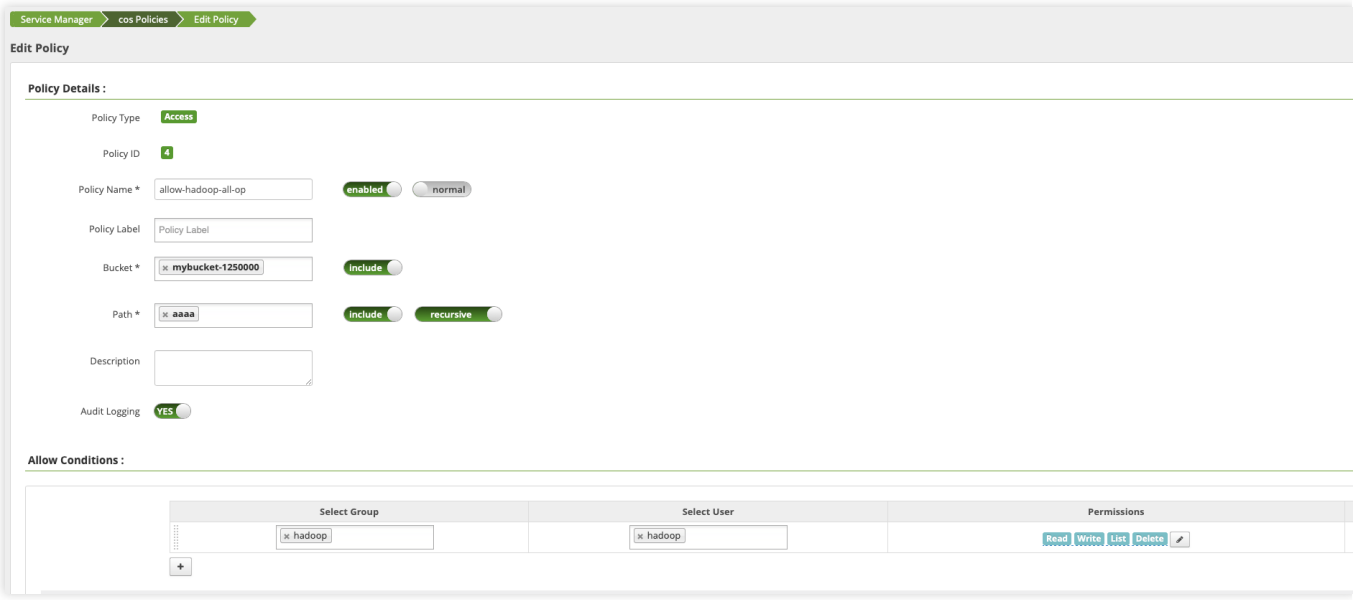
Permissions：

Read：读操作。对应于对象存储里面的 GET、HEAD 类操作，包括下载对象、查询对象元数据等。

Write：写操作,。对应于对象存储里面的 PUT 类等修改操作，例如上传对象。

Delete：删除操作。 对应于对象存储里删除 Object。对于 Hadoop 的 Rename 操作，需要有对原路径的删除操作权限，对新路径的写入操作权限。

List：遍历权限。对应于对象存储里面的 List Object。



COS Ranger Service 是整个权限体系的核心，负责集成 ranger 的客户端，接收 ranger client 的鉴权请求，token 生成续租请求和临时密钥生成请求。同时也是敏感信息（腾讯云密钥信息）所在的区域，通常部署在堡垒机器上，只允许集群管理员操作，查看配置等。
COS Ranger Service 支持一主多备的 HA 部署，DelegationToken 状态持久化到 HDFS。通过 ZK 抢锁决定 Leader 身份。获取 Leader 身份的服务会把地址写入 ZK，以便 COS Ranger Client 进行路由寻址。

代码地址

可前往 Github 的 cos-ranger-server 目录下获取。

版本

V5.1.2版本及以上。

部署步骤

1. 将 COS Ranger Service 服务代码拷贝到集群的几台机器上，生产环境建议至少两台机器（一主一备）。因为涉及到敏感信息，建议是堡垒机或者权限严格管控的机器。
2. 修改 cos-ranger.xml 文件中的相关配置，其中必须修改的配置项如下所示。配置项说明请参见文件中的注释说明（配置文件可前往 Github 的 cos-ranger-service/conf 目录下获取）。
qcloud.object.storage.rpc.address
qcloud.object.storage.status.port
qcloud.object.storage.enable.cos.ranger
qcloud.object.storage.zk.address （zk 地址，cos ranger service 启动后注册到 zk 上）
qcloud.object.storage.cos.secret.id
qcloud.object.storage.cos.secret.key
3. 修改 ranger-cos-security.xml 文件中的相关配置。其中必须修改的配置项有如下所示。配置项说明请参见文件中的注释说明（配置文件可前往 Github 的 cos-ranger-service/conf 目录下获取）。

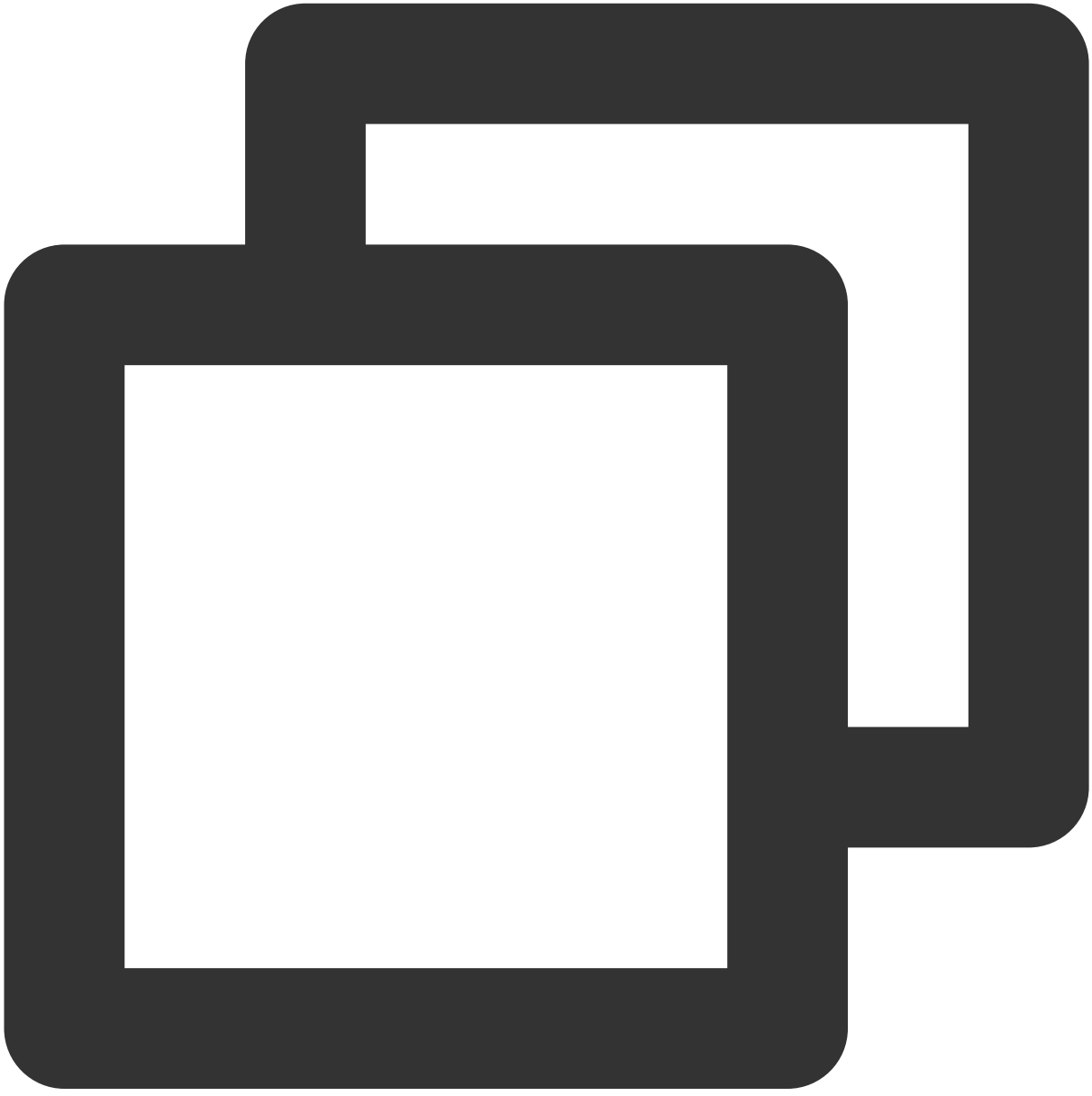
ranger.plugin.cos.policy.cache.dir

ranger.plugin.cos.policy.rest.url

ranger.plugin.cos.service.name

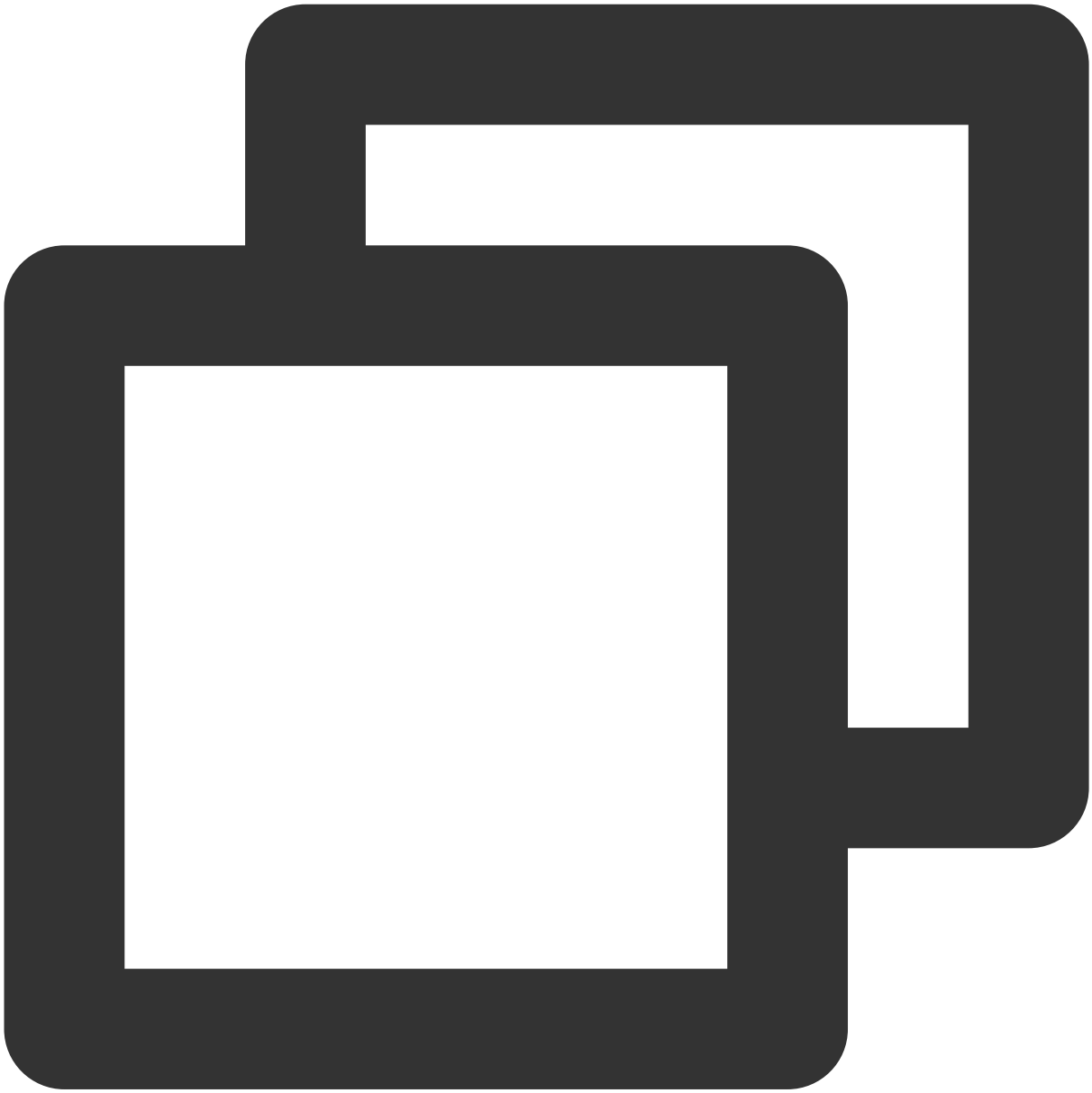
4. 修改 start_rpc_server.sh 中 hadoop_conf_path 和 java.library.path 的配置。这两个配置分别指向 hadoop 配置文件所在的目录（例如 core-site.xml、hdfs-site.xml）以及 hadoop native lib 路径。

5. 执行如下命令启动服务。



```
chmod +x start_rpc_server.sh
nohup ./start_rpc_server.sh &> nohup.txt &
```

6. 如果启动失败，查看 log 下 error 日志是否有错误信息。
7. COS Ranger Service 支持展示 HTTP 端口状态（端口名为 qcloud.object.storage.status.port，默认值为9998）。用户可通过以下命令获取状态信息（例如是否包含 leader、鉴权数量统计等）



```
# 请将下面的10.xx.xx.xxx替换为部署 ranger service 的机器 IP
# port 9998 设置为 qcloud.object.storage.status.port 配置值
curl -v http://10.xx.xx.xxx:9998/status
```

如果只部署了一个 cos ranger service 节点，会在上述接口响应中看到当前节点成为 leader。

如果部署了多个 cos ranger service 节点，会在上述接口响应中看到其他节点成为 leader，完成全部节点重启后，会看到最早完成重启的节点成为 leader。
COS Ranger Client 由 hadoop cosn 插件动态加载，并代理访问 COS Ranger Service 的相关请求。例如获取临时密钥、获取 token、鉴权操作等。

代码地址

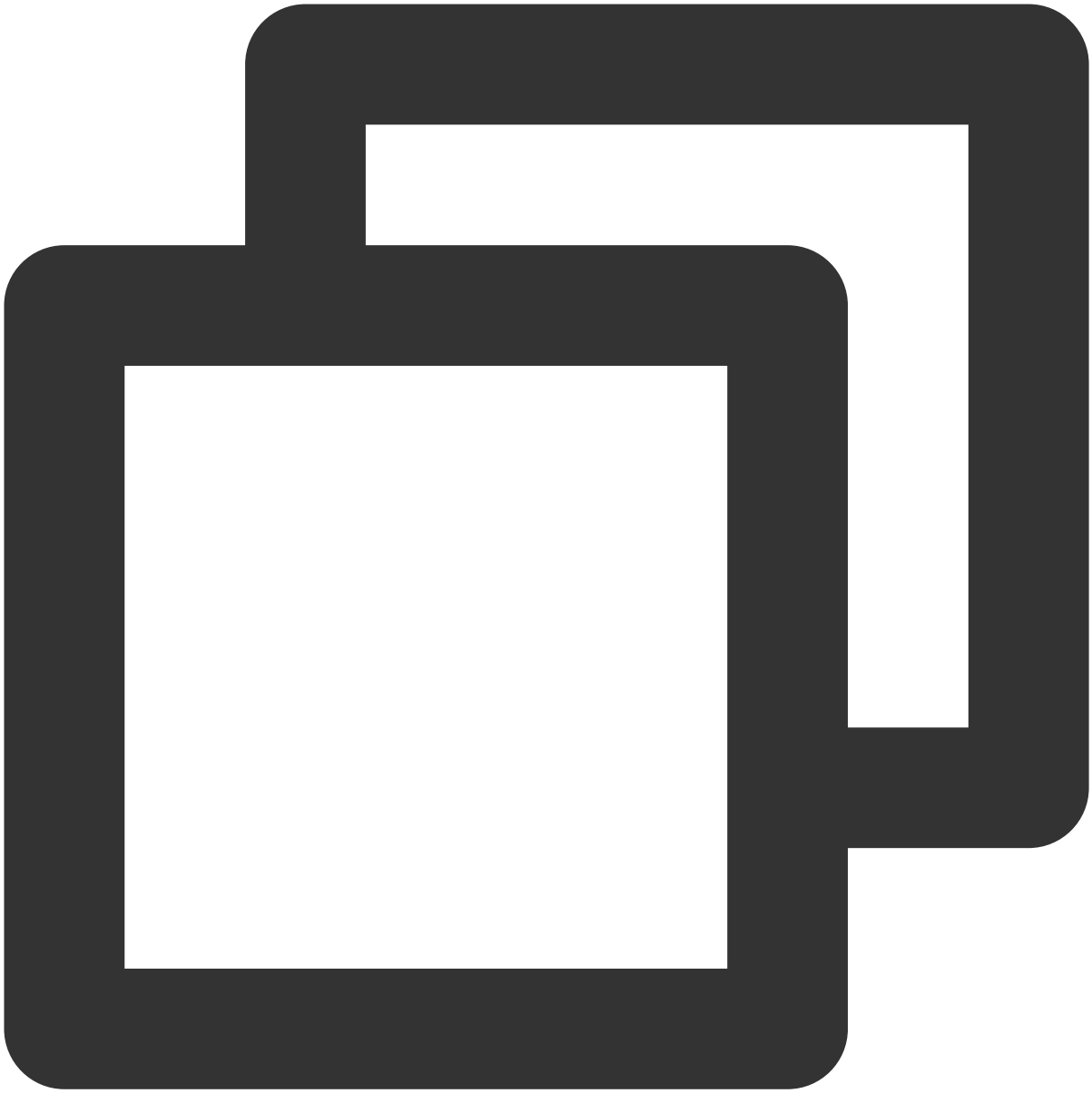
可前往 Github 的 cos-ranger-client 和 cosn-ranger-interface 目录下获取。

版本

cos-ranger-client 要求V5.0 版本及以上。cosn-ranger-interface 要求 v1.0.4版本及以上。

部署方式

1. 将 cos-ranger-client jar 包和cosn-ranger-interface jar 包拷贝到与 COSN 同一目录下通常在/usr/local/service/hadoop/share/hadoop/common/lib/目录下；请选择拷贝与自身 hadoop 大版本一致的 jar 包，最后确保 jar 包有可读权限。
2. 在 core-site.xml 添加如下配置项：



```
<configuration>
            <!--******必须配置********-->
            <!-- 上一步部署的 cos ranger server 的地址 -->
            <property>
                <name>qcloud.object.storage.ranger.service.address</name>
                <value>10.0.0.8:9999,10.0.0.10:9999</value>
            </property>

            <!--***可选配置****-->
            <!-- 设置 cos ranger service 端用的 kerberos 凭据，参考 cos ranger service
```

```
            <property>
                      <name>qcloud.object.storage.kerberos.principal</name>
                      <value>hadoop/_HOST@EMR-XXXX</value>
            </property>

          <!--***可选配置****-->
          <!-- zk 上记录的 ranger server 的 IP 地址路径, 这里使用了默认值, 配置必须与 COS Ra
           <property>
                     <name>qcloud.object.storage.zk.leader.ip.path</name>
                     <value>/ranger_qcloud_object_storage_leader_ip</value>
           </property>
          <!-- zk 上记录的 cos ranger service follower 的 IP 地址路径,这里使用了默认值 必须
           <property>
                     <name>qcloud.object.storage.zk.follower.ip.path</name>
                     <value>/ranger_qcloud_object_storage_follower_ip</value>
           </property>
</configuration>
```

版本

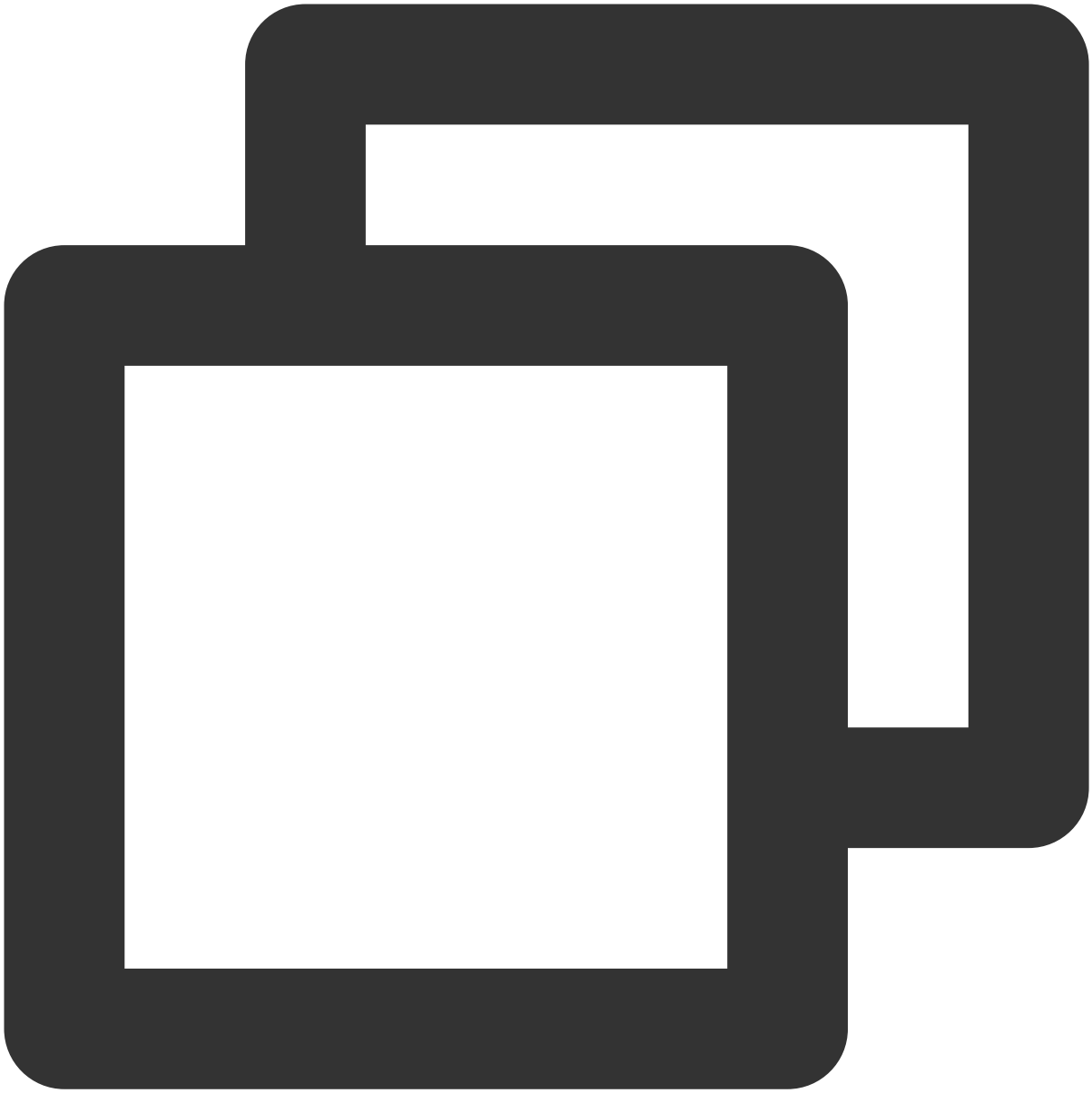
V8.0.1版本及以上。

部署方式

部署 COSN 插件方法请参考 Hadoop 工具 文档，但需注意以下几点：

1. 使用 ranger 后，fs.cosn.userinfo.secretId 和 fs.cosn.userinfo.secretKey 密钥信息不需要配置。COSN 插件后续通过 COSRangerService 获取临时密钥。

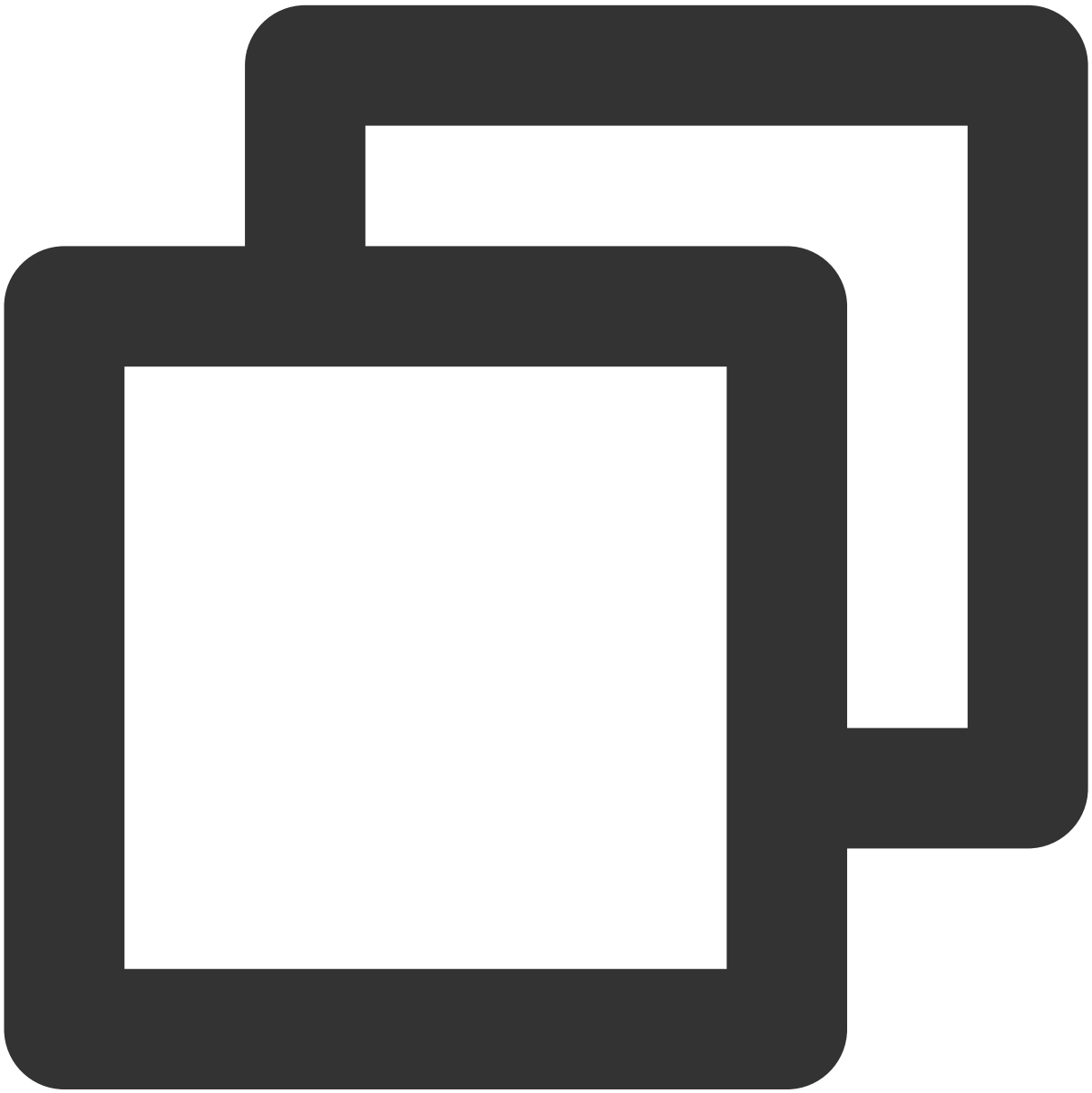
2. fs.cosn.credentials.provider 需设置为 org.apache.hadoop.fs.auth.RangerCredentialsProvider 才可通过 Ranger 进行认证鉴权。如下所示：



```
<property>
         <name>fs.cosn.credentials.provider</name>
         <value>org.apache.hadoop.fs.auth.RangerCredentialsProvider</value>
</property>
```

验证

1. 使用 hadoop cmd 执行访问 COSN 的相关操作。查看当前用户执行的操作是否符合主账号的权限设置预期，示例如下所示：



```
#将bucket，路径等替换为主账号的实际信息。
hadoop fs -ls cosn://examplebucket-1250000000/doc
hadoop fs -put ./xxx.txt cosn://examplebucket-1250000000/doc/
hadoop fs -get cosn://examplebucket-1250000000/doc/exampleobject.txt
hadoop fs -rm cosn://examplebucket-1250000000/doc/exampleobject.txt
```

2. 使用 MR Job 进行验证，验证前需重启相关的服务，例如 Yarn、Hive 等。

常见问题

kerberos 是否必须安装？

Kerberos 满足认证的需求，如果所在的集群，用户都是可信的，例如仅内部使用的集群。若用户仅进行鉴权操作，为了避免无权限的客户误操作，那么可以不安装 Kerberos，只使用 ranger 进行鉴权。同时 Kerberos 会引入一些性能损耗。请客户综合自己的安全需求与性能需求进行考量。如果需要认证，开启 Kerberos 后，需要设置 COS Ranger Service 和 COS Ranger Client 相关的配置项。

如果开启了 Ranger，但未配置任何 Policy，或者未匹配到任何 Policy，会如何操作？

如果未匹配上任何 policy，会默认拒绝该操作。

配置 COS Ranger Service 侧的密钥可以是子账号？

可以是子账号，但是必须拥有被操作 bucket 的相应权限，才能生成临时密钥给到 COSN 插件，进行相应的操作。通常建议这里设置的密钥拥有对该 bucket 的所有权限。

临时密钥需如何更新，每次访问 COS 前都需要从 COS Ranger Service 侧获取?

临时密钥是 cache 在 COSN 插件侧，并周期性进行异步更新。

在 ranger 页面更改了 Policy 未生效怎么办？

请修改 ranger-cos-security.xml 文件的配置项：ranger.plugin.cos.policy.pollIntervalMs，调小该配置项（单位为毫秒），然后重启 COS Ranger Service 服务。Policy 相关测试结束后，建议修改回原来值（时间间隔太小导致轮询频率高，从而导致 CPU 利用率高企）。

使用流计算 Oceanus 接入 COS

最近更新时间：2024-01-06 10:54:03

Oceanus 简介

流计算 Oceanus是大数据生态体系的实时化分析利器。只需几分钟，您就可以轻松构建网站点击流分析、电商精准推荐、物联网 IoT 等应用。流计算基于 Apache Flink 构建，提供全托管的云上服务，您无须关注基础设施的运维，并能便捷对接云上数据源，获得完善的配套支持。
流计算 Oceanus 提供了便捷的控制台环境，方便用户编写 SQL 分析语句或者上传运行自定义 JAR 包，支持作业运维管理。基于 Flink 技术，流计算可以在 PB 级数据集上支持亚秒级的处理延时。
目前 Oceanus 使用的是独享集群模式，用户可以在自己的集群中运行各类作业，并进行相关资源管理。本文将为您详细介绍如何使用 Oceanus 对接对象存储（Cloud Object Storage，COS）。

准备工作

创建 Oceanus 集群

登录 Oceanus 控制台，创建一个 Oceanus 集群。

创建 COS 存储桶

1. 登录 COS 控制台。
2. 在左侧导航栏中，单击**存储桶列表**。
3. 单击**创建存储桶**，创建一个存储桶。具体可参见 创建存储桶 文档。

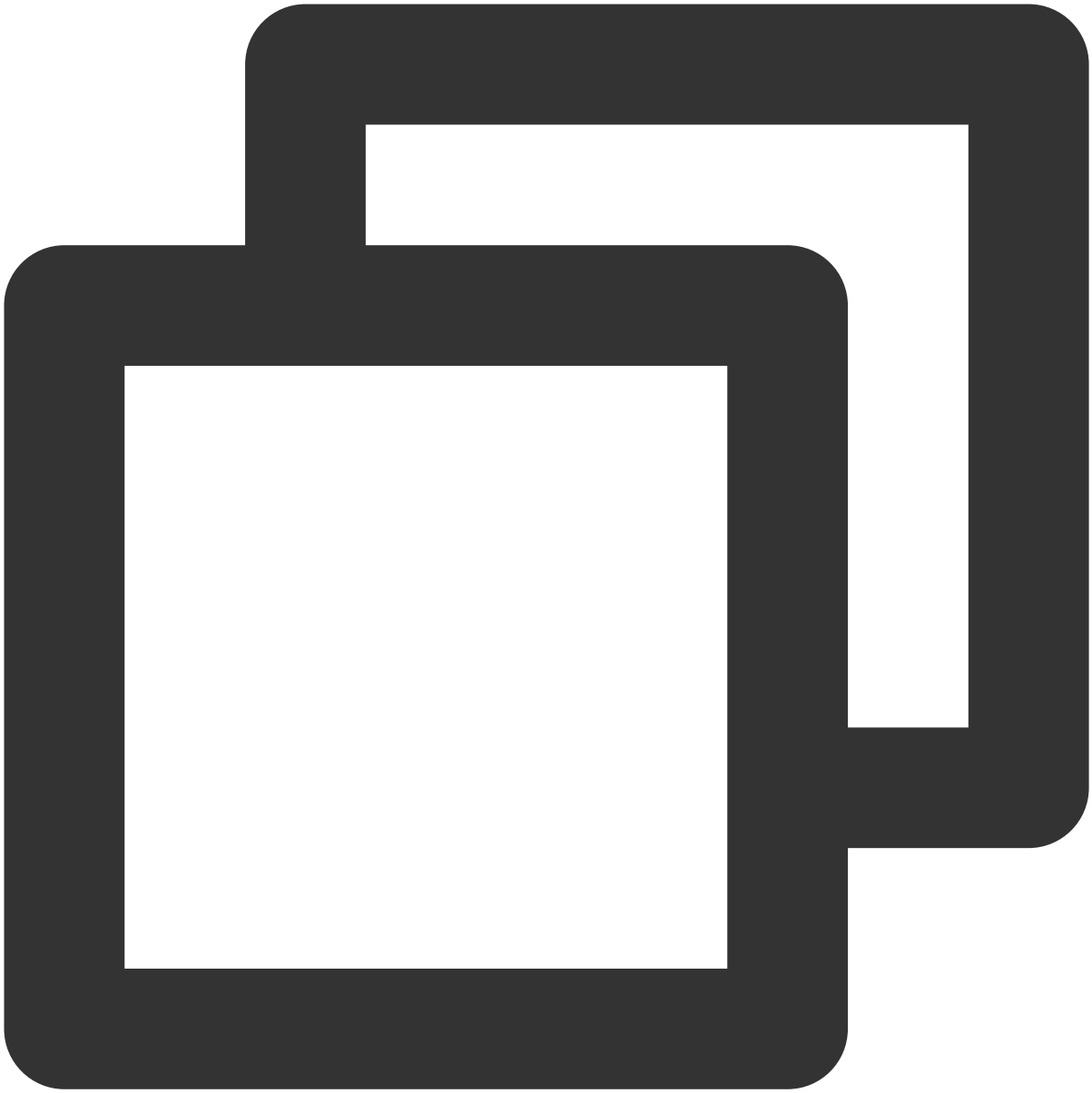
说明

当写入 COS 时，Oceanus 作业所运行的地域必须和 COS 在同一个地域。

实践步骤

前往 Oceanus 控制台，创建一个 SQL 作业，集群选择与 COS 在相同地域的集群。

1. 创建 Source



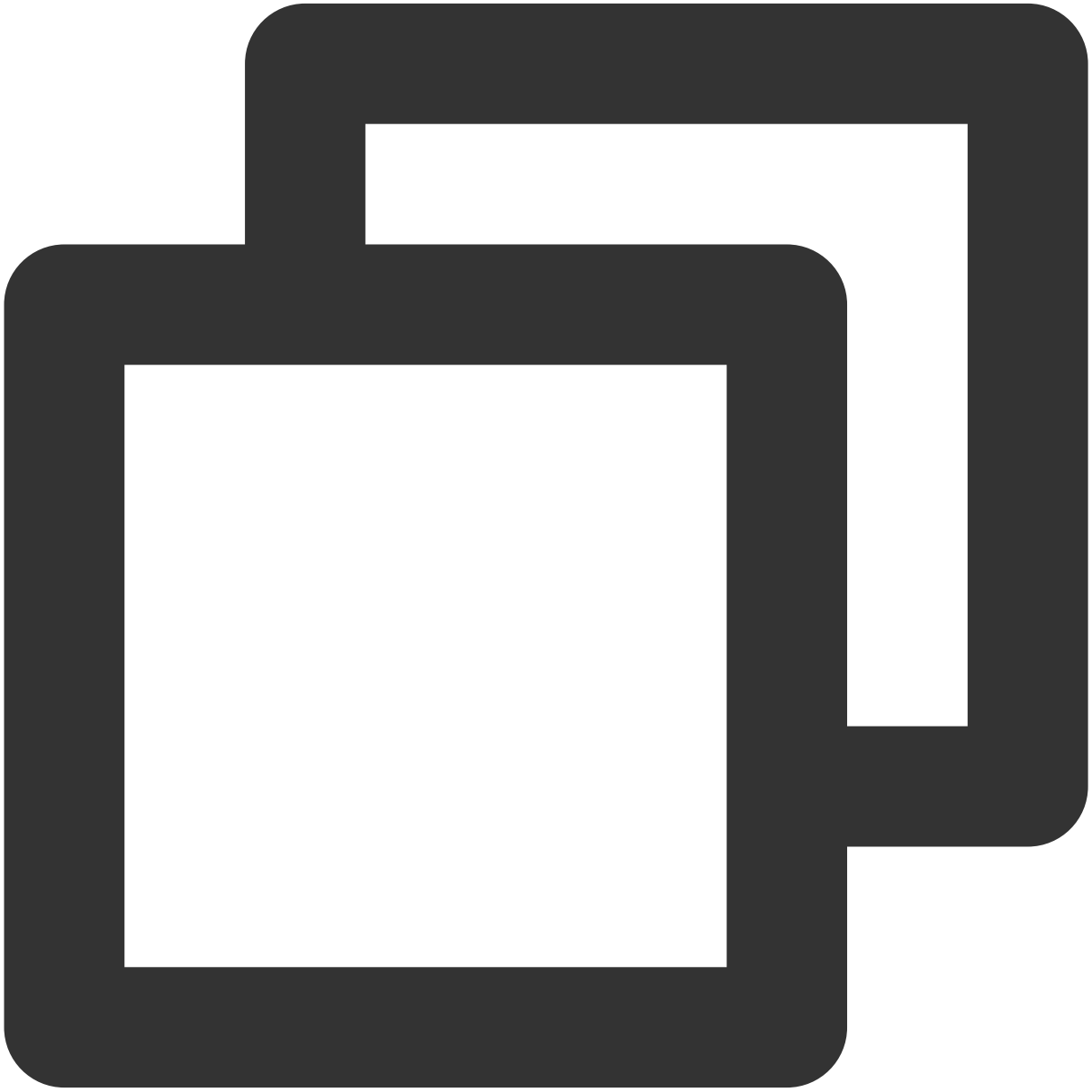
```
CREATE TABLE `random_source` (
  f_sequence INT,
  f_random INT,
  f_random_str VARCHAR
  ) WITH (
  'connector' = 'datagen',
  'rows-per-second'='10',                    -- 每秒产生的数据条数
  'fields.f_sequence.kind'='random',         -- 随机数
  'fields.f_sequence.min'='1',               -- 随机数的最小值
  'fields.f_sequence.max'='10',              -- 随机数的最大值
  'fields.f_random.kind'='random',           -- 随机数
```

```
    'fields.f_random.min'='1',                  -- 随机数的最小值
    'fields.f_random.max'='100',                -- 随机数的最大值
    'fields.f_random_str.length'='10'           -- 随机字符串的长度
);
```

说明

此处选用内置 connector `datagen`，请根据实际业务需求选择相应数据源。

2. 创建 Sink



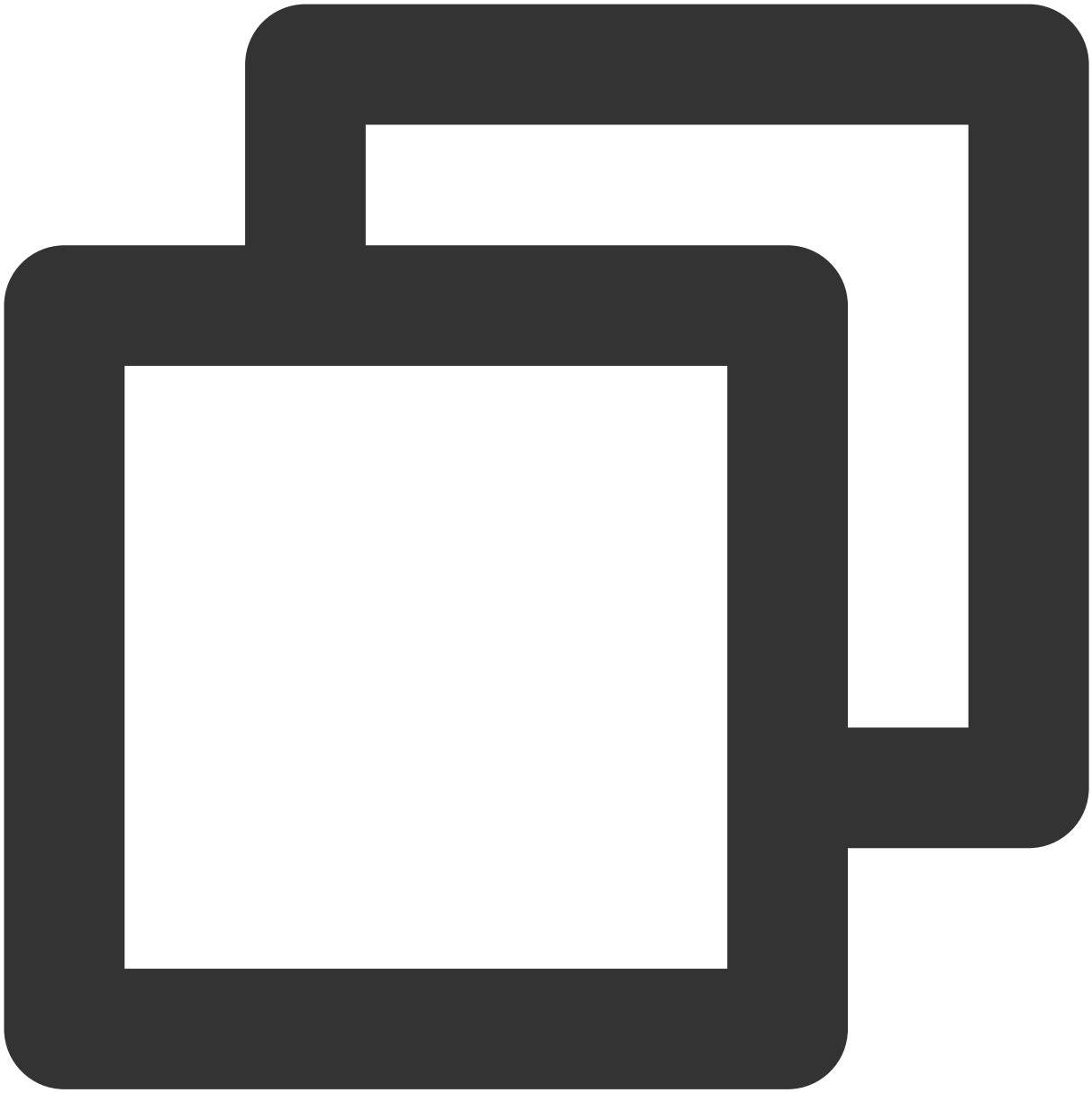
```
-- 请将<存储桶名称>和<文件夹名称>替换成您实际的存储桶名称和文件夹名称
```

```
CREATE TABLE `cos_sink` (
  f_sequence INT,
  f_random INT,
  f_random_str VARCHAR
) PARTITIONED BY (f_sequence) WITH (
    'connector' = 'filesystem',
    'path'='cosn://<存储桶名称>/<文件夹名称>/',                    --- 数据写入的目录路径
    'format' = 'json',                                          --- 数据写入的格式
    'sink.rolling-policy.file-size' = '128MB',                  --- 文件最大的大小
    'sink.rolling-policy.rollover-interval' = '30 min',         --- 文件最大写入时间
    'sink.partition-commit.delay' = '1 s',                      --- 分区提交延迟
    'sink.partition-commit.policy.kind' = 'success-file'        --- 分区提交方式
);
```

说明

更多 Sink 的 WITH 参数，请参见Filesystem (HDFS/COS)文档。

3. 业务逻辑



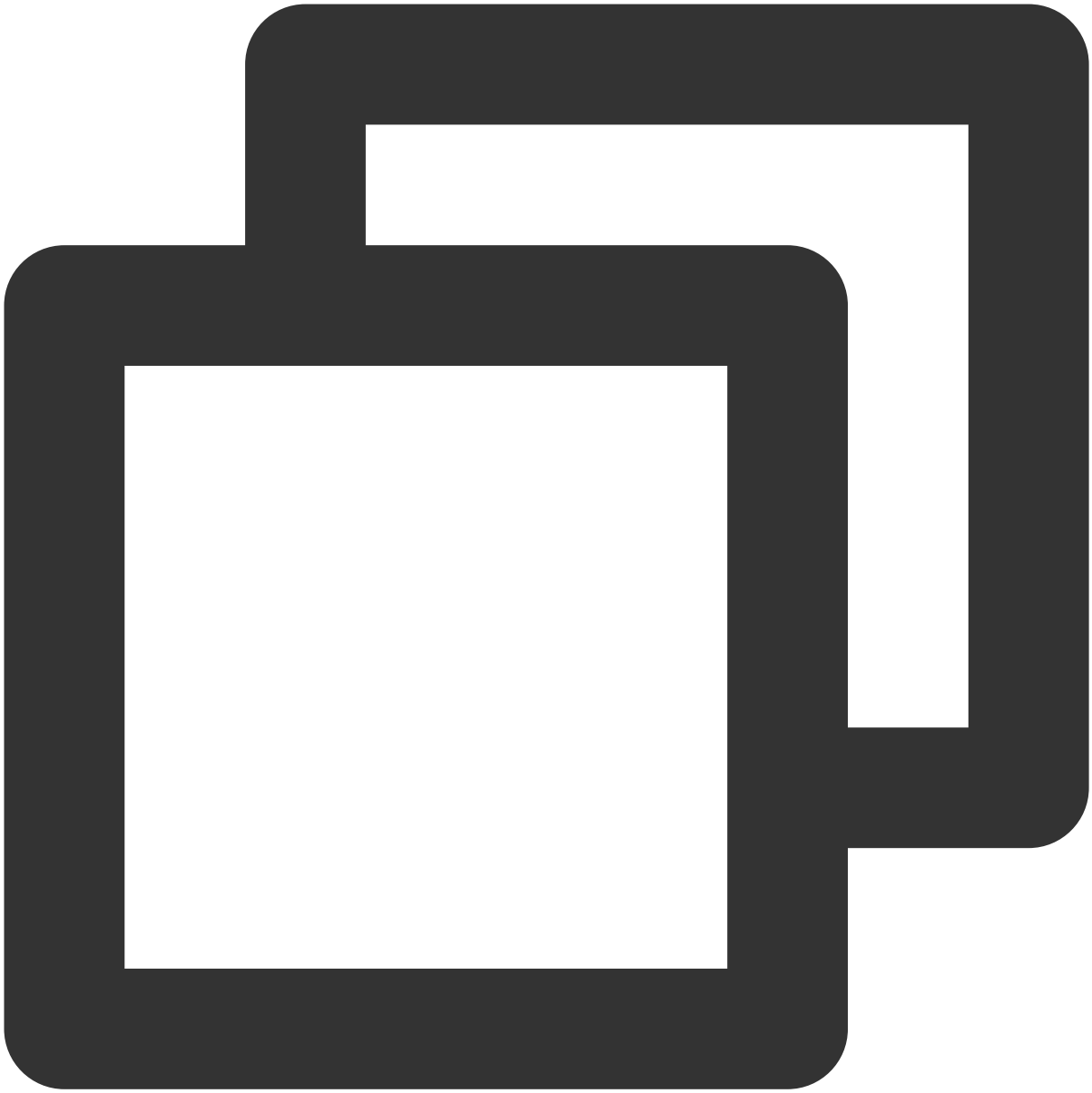
```
INSERT INTO `cos_sink`
SELECT * FROM `random_source`;
```

注意

此处只做展示，无实际业务目的。

4. 作业参数设置

在**内置 Connector**选择 `flink-connector-cos` ，在**高级参数**中对 COS 的地址进行如下配置：



```
fs.AbstractFileSystem.cosn.impl: org.apache.hadoop.fs.CosN
fs.cosn.impl: org.apache.hadoop.fs.CosFileSystem
fs.cosn.credentials.provider: org.apache.flink.fs.cos.OceanusCOSCredentialsProvider
fs.cosn.bucket.region: <COS 所在地域>
fs.cosn.userinfo.appid: <COS 所属用户的 appid>
```

作业配置说明如下：

请将 `<COS 所在地域>` 替换为您实际的 COS 地域，例如：ap-guangzhou。

请将 `<COS 所属用户的 appid>` 替换为您实际的 APPID，具体请进入 账号中心 查看。

说明

具体的作业参数设置请参见Filesystem (HDFS/COS) 文档。

5. 启动作业

依次单击**保存 > 语法检查 > 发布草稿**，等待 SQL 作业启动后，即可前往相应 COS 目录中查看写入数据。

在第三方应用中使用 COS
在兼容 S3 的第三方应用中使用 COS 的通用配置

最近更新时间：2024-01-06 10:54:03

Amazon Simple Storage Service（Amazon S3，下文简称 S3）是 AWS 最早推出的云服务之一，经过多年的发展，S3 协议在对象存储行业事实上已经成为标准。腾讯云对象存储（Cloud Object Storage，COS）提供了兼容 S3 的实现方案，因此您可以在大部分兼容 S3 应用中直接使用 COS 服务。本文将重点介绍如何将此类应用配置为使用 COS 服务。

准备工作

确认应用是否可以使用 COS 服务

如果您在应用的说明中看到类似 `S3 Compatible` 字样，那么大多数情况可以使用 COS 服务。如果您在实际使用过程中发现应用的某些功能无法正常使用，您可以 联系我们。联系时，请说明您是从该文档中看到的指引，并提供相关应用的名称和截图等信息，以便我们可以更快的帮您解决问题。
如果您的应用只说明支持 `Amazon S3`，这表明该应用可以使用 S3 服务，但能否使用 COS 服务，还需要在相关的配置中进一步尝试，本文也会在后续的配置说明中做进一步的说明。

准备 COS 服务

步骤1：注册腾讯云账号

（如果已在腾讯云注册，可忽略此步骤。）

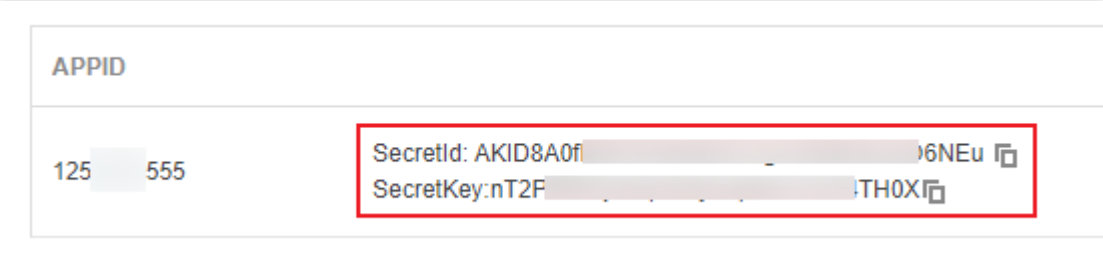
步骤2：完成实名认证

（如果已完成，可忽略此步骤。）
详细认证流程，请参见 实名认证介绍。

步骤3：开通 COS 服务

步骤4：准备 APPID 和访问密钥

在访问管理控制台的 API 密钥管理 页面中获取并记录 **APPID**、**SecretId** 和 **SecretKey**。



步骤5：创建存储桶

参见 创建存储桶 创建一个存储桶。
部分应用内置创建存储桶的过程，如果您希望由应用去创建存储桶，您可以忽略此步骤。

在应用中配置 COS 服务

基本配置

大部分应用在配置使用的存储服务时，都有类似的配置项，下面列举这些配置项的常见名称及相关说明：

说明

如果您在配置过程中有任何疑问，也可以 联系我们。联系时，请说明您是从该文档中看到的指引，并提供相关应用的名称和截图等信息，以便我们可以更快的帮您解决问题。

配置项的常见名称	相关说明
提供商/服务提供商/存储服务提供商/Service Provider/Storage Provider/Provider 等	这里主要是选择应用应使用哪种存储，可能存在以下几种情况： 如果该选项中有类似 S3 兼容存储/S3 Compatible等字样的选项，那么优先使用这个选项。 如果只有 amazon web services/AWS/Amazon S3 等字样，那么先使用这个选项，但是在后面的配置中需留意我们的进一步说明。 如果没有类似选项，但是在应用的说明中有提到支持 S3 服务或 S3 兼容服务，那么您可以继续后面的配置，但同样需要留意我们的进一步说明。 如果是其他情况，很抱歉，该应用可能不能使用 COS 服务。
服务端点/服务地址/服务 URL/Endpoint/Custom Endpoint/Server URL 等	这里用于填写 S3 兼容服务的服务地址，在使用 COS 服务时，这里填写 COS 的服务地址，形式为： cos.<Region>.myqcloud.com 或 https://cos.<Region>.myqcloud.com 。

	是否需要填写 https:// ，根据具体的应用有所不同，您可以自行尝试。其中 <Region> 代表 COS 的可用地域。 在应用中，您只能在服务地址中指定的地域创建或选择存储桶。 例如您的存储桶在广州地域，那么服务地址应当配置为 cos.ap-guangzhou.myqcloud.com，如果您配置成其他地域，那么在应用中您无法找到广州地域下的存储桶。 如果应用的服务提供商中只能选择Amazon S3，并且服务端点是可以配置的，那么您可以将服务端点修改为前述的cos.<Region>.myqcloud.com或https://cos.<Region>.myqcloud.com。 如果服务端点是不可配置的或没有服务端点配置项，那么您的应用不能使用 COS 服务。
Access Key/Access Key ID 等	这里填写 步骤4 中记录的 SecretId。
Secret Key/Secret/Secret Access Key 等	这里填写 步骤4 中记录的 SecretKey。
地域/Region 等	选择默认、自动、Auto 或 Automatic。
存储桶/Bucket 等	选择或输入现有的存储桶名称，格式为<BucketName-APPID>，例如examplebucket-1250000000，其中 BucketName 为 步骤5 中创建存储桶时填写的存储桶名称，APPID 为 步骤4 中记录的 APPID。如上文所描述，这里的存储桶将限定在服务地址所指定的地域中，其他地域的存储桶将不会被列出或无法正常使用。如果您需要创建新的存储桶，那么新创建的存储桶名字也需要符合前面所讲的 <BucketName-APPID> 格式，否则就无法正常创建存储桶。

其他项与高级配置说明

部分应用除了上述基本配置外，还有一些其他项与高级配置，下面将提供部分 COS 的功能说明，以便您更好的在应用中使用 COS 服务。

服务端口与协议

COS 服务支持 HTTP 协议和 HTTPS 协议，均使用协议默认的80和443端口，基于安全考虑，我们建议您优先通过 HTTPS 协议使用 COS 服务。

Path-Style 与 Virtual Hosted-Style

COS 同时支持两种使用风格。

AWS V2 签名与 AWS V4 签名

COS 同时支持两种签名格式。

结语

COS 不保证与 S3 的完全兼容，如果您在应用中使用 COS 服务时遇到任何问题，您可以 联系我们。联系时，请说明您是从该文档中看到的指引，并提供相关应用的名称和截图等信息，以便我们可以更快的帮您解决问题。

将 WordPress 远程附件存储到 COS

最近更新时间：2024-01-06 10:54:03

简介

WordPress 是使用 PHP 语言开发的博客平台，用户可以在支持 PHP 和 MySQL 数据库的服务器上架设属于自己的网站，也可以把 WordPress 当作一个内容管理系统（CMS）来使用。
WordPress 功能强大、扩展性强，这主要得益于其插件众多，易于扩充功能，基本上一个完整网站该有的功能，通过其第三方插件都能实现所有功能。
这篇文章我们来介绍一下如何使用插件实现远程附件功能，将 WordPress 的媒体库附件存储在腾讯云 对象存储（Cloud Object Storage，COS） 上。
COS 具有高扩展性、低成本、可靠和安全等特点，将媒体库附件保存在 COS 上有以下好处：
附件将拥有更高的可靠性。
您的服务器无需为附件准备额外的存储空间。
用户查看图片附件时将直连 COS 服务器，不占用您服务器的下行带宽/流量，用户访问速度更快。
可配合腾讯云 内容分发网络（Content Delivery Network，CDN） 进一步提升用户查看图片附件的速度，优化网站访问速度。

前提条件

1. 已有 COS 存储桶。如无，可参见 创建存储桶 操作指引。
2. 已创建服务器。例如云服务器（Cloud Virtual Machine，CVM）。相关指引可参见 CVM 产品文档。

实践步骤

Wordpress 部署

基于腾讯云服务器来快速搭建 WordPress，有两种方式：通过镜像部署和手动部署。如果您对业务网站有较高的扩展性需求，可手动搭建，详情请参见以下指引：
手动搭建 WordPress 个人站点（Linux）
手动搭建 WordPress 个人站点（Windows）
这里介绍通过镜像部署 WordPress，镜像部署简便快捷。操作步骤如下：
1. 通过镜像部署 WordPress。
1.1 登录 云服务器控制台，单击实例管理页面的**新建**。
1.2 根据页面提示选择机型，并在**实例配置 > 镜像**中单击**镜像市场**，选择**从镜像市场选择**。

1.3 在“镜像市场”弹窗中，选择**基础软件**，输入 **wordpress** 进行搜索。
1.4 按需选择镜像，以选择 **WordPress博客程序_v5.5.3(CentOS | LAMP)** 为例，单击**免费使用**。
1.5 完成选购后，登录 CVM 控制台，为刚创建的实例关联安全组，需添加放通80端口的入站规则。
2. 在实例的管理页面，复制该云服务器实例的**公网 IP**，在本地浏览器中访问地址 `http://公网 IP/wp-admin` ，开始安装 WordPress 网站：
2.1 选择 Wordpress 语言后，单击 **Continue**。
2.2 按需输入 WordPress 站点标题、管理员用户名、管理员密码及电子邮件。
2.3 单击**安装WordPress**。
2.4 单击**登录**。
3. 将 WordPress 升级到新版本6.0.2。
单击仪表盘左侧菜单，进入“更新”菜单，更新到新版本6.0.2。

创建 COS 存储桶

1. 创建一个**公有读私有写**的存储桶，存储桶的地域建议与运行 WordPress 博客平台的 CVM 的地域相同，创建详情请参见 创建存储桶 文档。
2. 在**存储桶列表**中找到刚才创建的存储桶，并单击其存储桶名称，进入存储桶页面。
3. 在左侧导航栏中，单击**概览**，查看访问域名并记录。

安装并配置插件

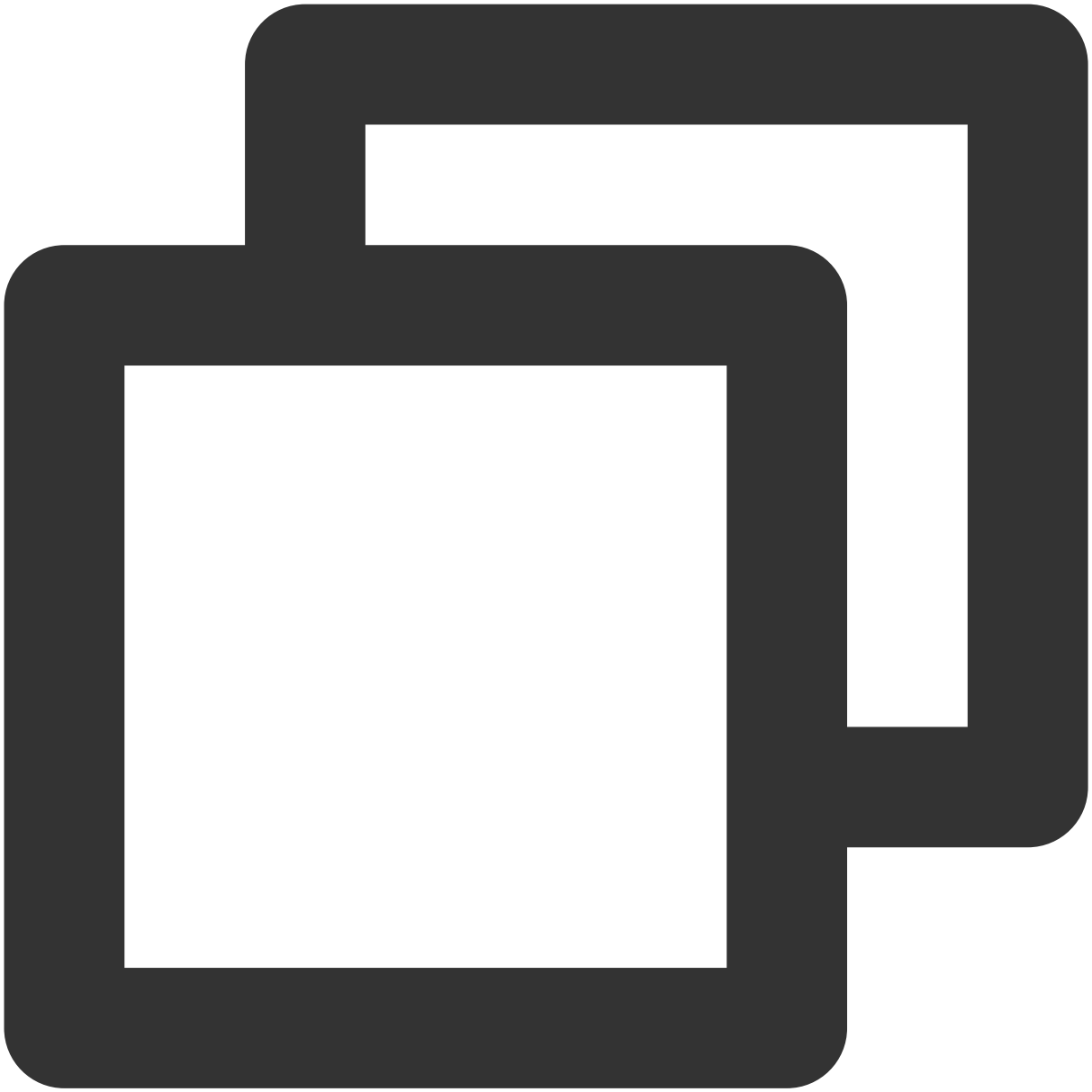
安装插件的方式包括在插件库中安装和源码安装。

在插件库中安装（推荐使用）

在 WordPress 后台，单击**插件**，直接搜索 **tencentcloud-cos** 插件，单击**立即安装**即可。

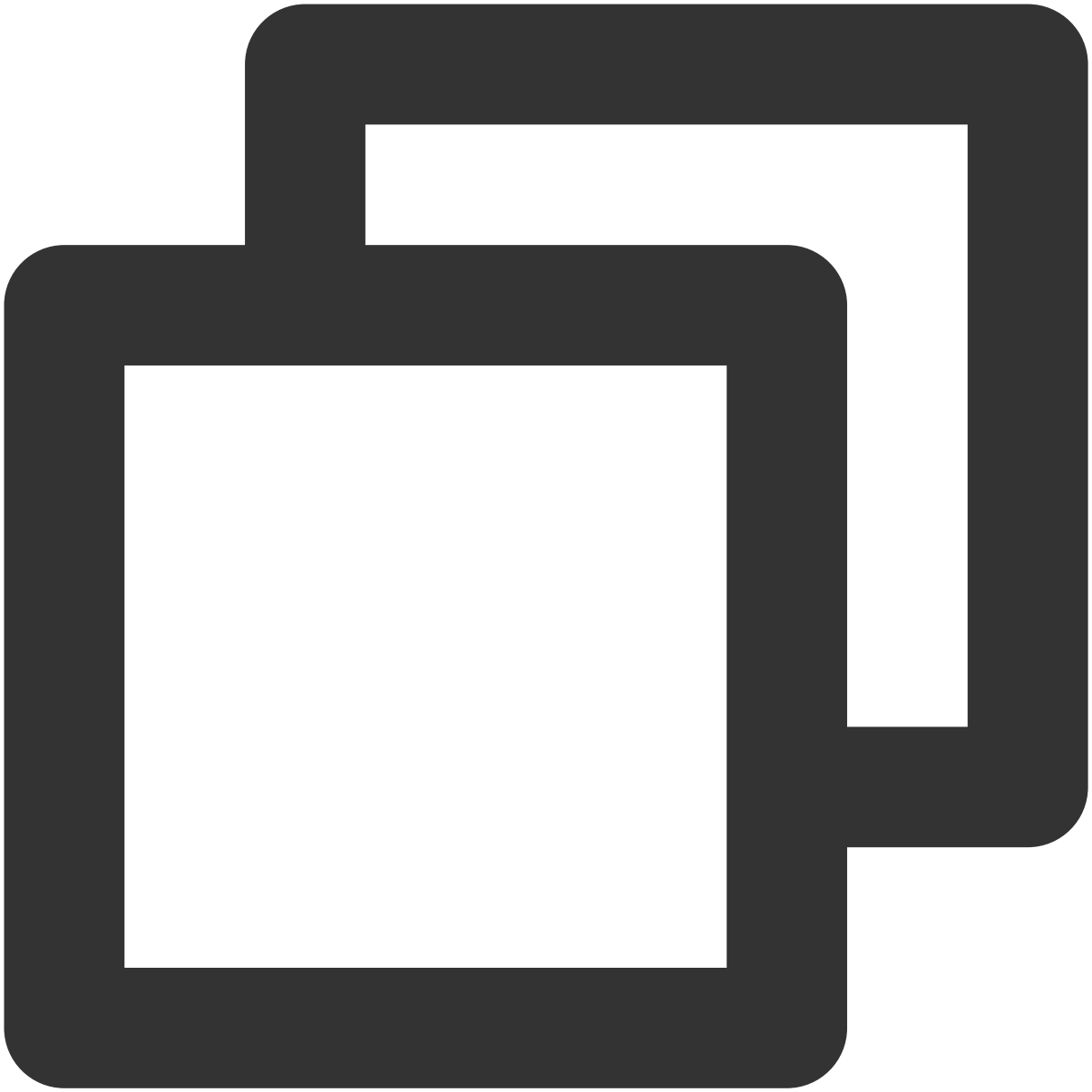
源码安装

首先下载插件源码，然后将插件源码上传到 WordPress 插件目录 `wp-content/plugins` ，最后在后台启用。
下面以 Ubuntu 为例安装插件：
1. 进入 wp-content 的父目录：



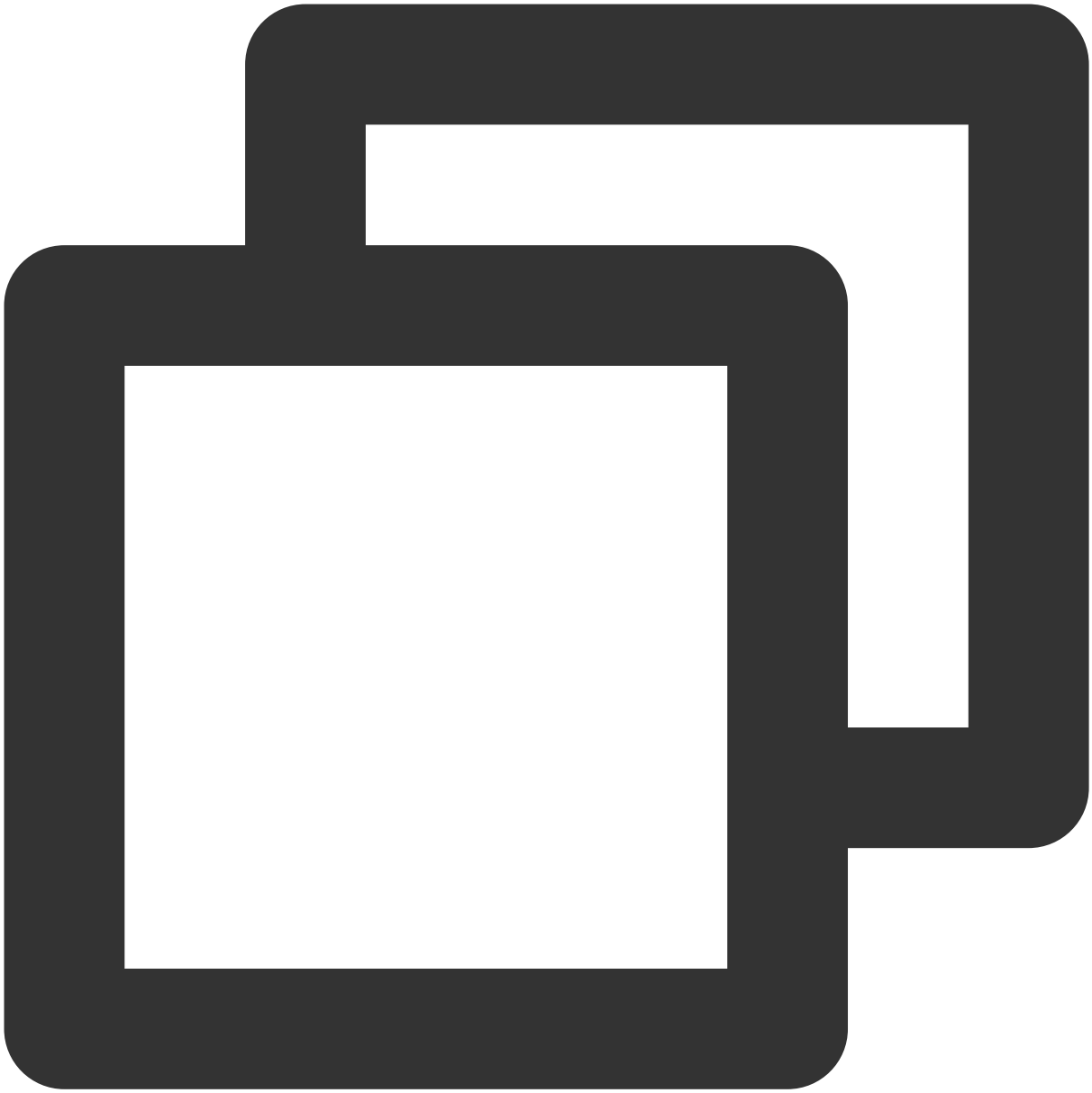
```
cd /var/www/html
```

2. 添加权限：



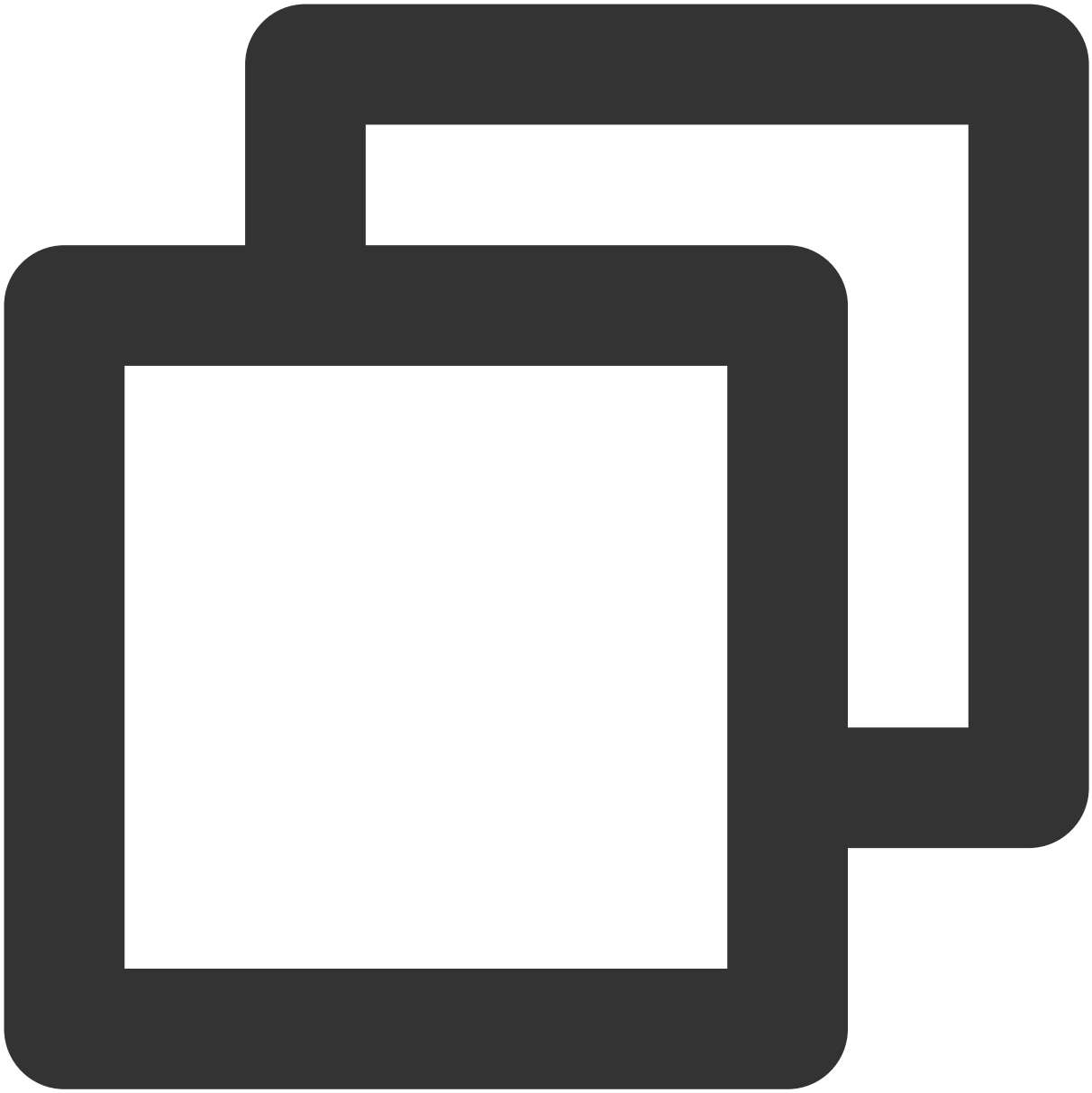
```
chmod -R 777 wp-content
```

3. 创建插件目录：



```
cd wp-content/plugins/
mkdir tencent-cloud-cos
cd tencent-cloud-cos
```

4. 下载插件到插件目录：



```
wget https://cos5.cloud.tencent.com/cosbrowser/code/tencent-cloud-cos.zip
unzip tencent-cloud-cos.zip
rm tencent-cloud-cos.zip -f
```

5. 单击“插件”左侧菜单，即可看到该插件，单击启用该插件。

配置插件

在插件 tencent-cloud-cos 配置 COS 存储桶信息：

1. 单击“设置”按钮来配置插件 tencent-cloud-cos。
2. 在页面中配置 COS 的相关信息，配置说明见下表：

配置项	配置值
SecretId、SecretKey	访问密钥信息，可前往 云 API 密钥 中创建和获取
所属地域	创建存储桶时所选择的地域
空间名称	创建存储桶时自定义的存储桶名称，例如examplebucket-1250000000
访问域名	指 COS 的默认存储桶域名，用户在创建存储桶时，由系统根据存储桶名称和地域自动生成。不同地域的存储桶有不同的默认域名。您可以前往 对象存储控制台 ，在存储桶的概览 > 域名信息中查看
自动重命名	文件上传到 COS 后自动重命名，避免与已有同名文件相冲突，可按照指定格式重命名
不在本地保存	开启后，不会在本地保留源文件
保留远程文件	开启后，当删除文件，将只删除本地文件副本，仍然保留远程 COS 桶中的文件副本，可方便找回
禁止缩略图	开启后不会上传对应的缩略图文件
数据万象	开启数据万象服务，可对图片进行编辑、压缩、转换格式、添加水印等操作，详情可参见 数据万象产品介绍
文件审核	开启文件审核，可对图片、视频、音频、文本、文档、网页等多媒体内容进行安全智能审核服务，可帮助用户有效识别色情低俗、违法违规、恶心反感等违禁内容，规避运营风险。
文档预览	开启文档预览，可将文件转码为图片、PDF 或 HTML5页面，解决文档内容的页面展示问题，详情可参见 文档预览概述
调试	记录错误、异常和警告信息

3. 配置完成后，单击**保存配置**即可。

验证 Wordpress 附件存储到 COS

在 Wordpress 创建一篇带图片的文章，查看图片是否保存在 COS。

1. 创建一篇带图片的文章。在 Wordpress 仪表盘，单击“文章”左侧菜单。
2. 对 Wordpress 默认生成的”世界，您好！“文章进行编辑。
3. 单击右边 “＋”按钮。
4. 然后选择上传一张图片。

5. 上传完成后，查看已上传图片的 URL，确认图片地址为 COS 的地址，例如 `https://wd-125000000.cos.ap-nanjing.myqcloud.com/2022/10/立夏-1200x675.jpeg`，格式为：`https://<BucketName-APPID>.cos.<Region>.myqcloud.com/<ObjectKey>`，表示图片已上传到 COS 存储桶。
6. 登录 COS 控制台，在 COS 存储桶里能看到您刚才上传的图片。

说明

以上测试成功后，接下来如果需要同步旧资源到 COS 存储桶中（可使用 COSCMD 工具 或者 COS Migration 工具），**否则后台无法正常预览旧资源**。同步完成以后，可以开启回源设置，可参考下文的 设置回源。

扩展

1. 使用 CDN 加速访问

存储桶如果需要配置 CDN 加速，可参见 CDN 加速配置 文档。在插件设置中将 URL 前缀修改为默认 CDN 加速域名或自定义加速域名即可。

2. 替换数据库中的资源地址

如果不是新创建的站点，数据库当中必定是旧的资源链接地址，我们需要将资源地址进行替换，插件提供了替换功能，请在首次替换前记得备份。

旧域名填写原资源域名，例如 `https://example.com/`

新域名填写现在的资源域名，例如 `https://img.example.com/`

3. 设置跨域访问

在文章中引用对应的资源链接，控制台会提示跨域的错误 `No 'Access-Control-Allow-Origin' header is present on the requested resource`。原因是没有添加 header。您需要在跨域访问 CORS 设置中添加 HTTP Header 配置。下面提供两种途径进行配置：

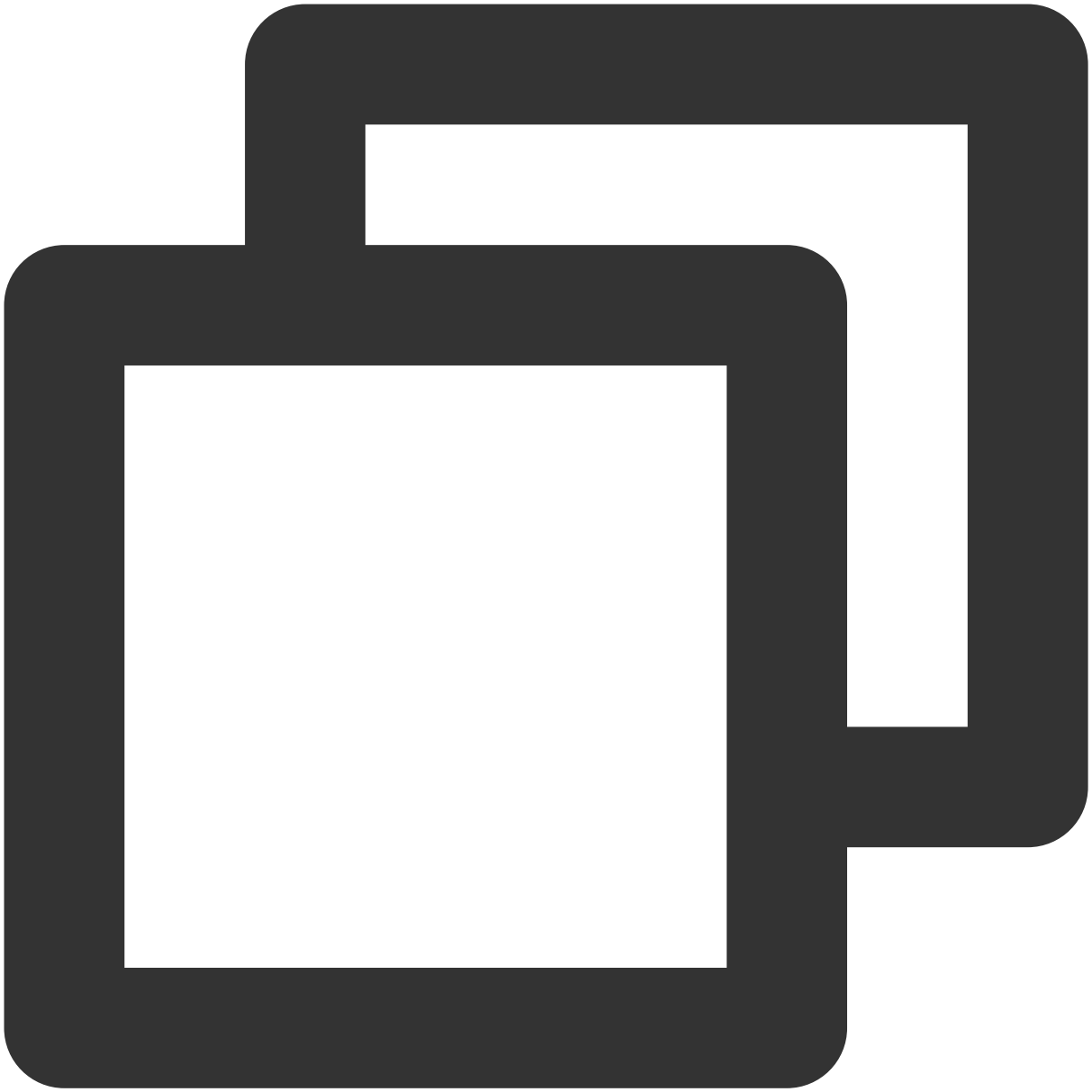
在 COS 控制台上配置

说明

关于跨域配置操作步骤，请参见 设置跨域访问 文档。

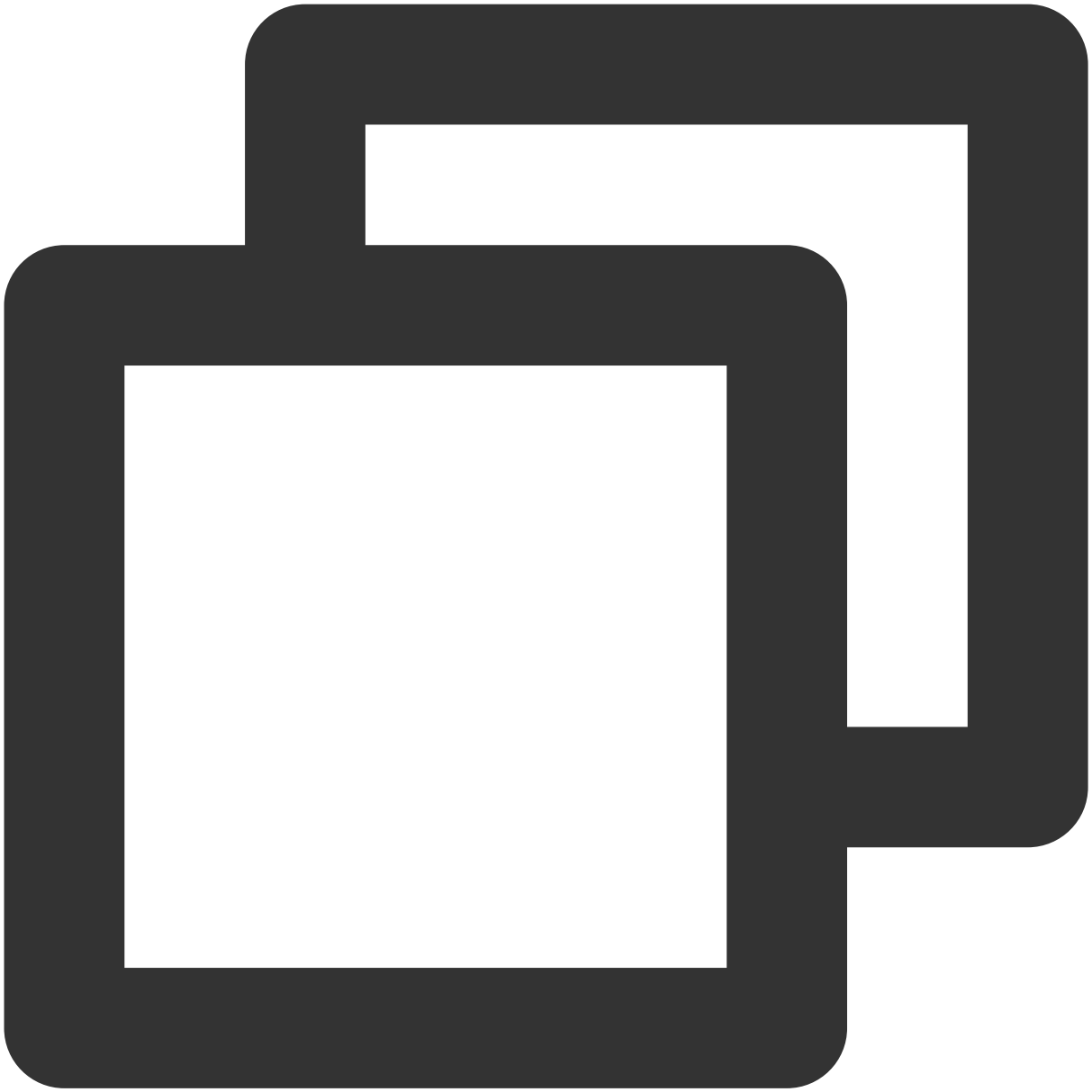
在 CDN 控制台上配置

如允许所有域名，则配置如下：



```
Access-Control-Allow-Origin: *
```

只允许您个人的域名访问，则配置如下：



```
Access-Control-Allow-Origin: https://example.com
```

4. 设置回源

如果您不在 WordPress 后台媒体库中上传资源，建议开启回源设置，详细步骤请参见 设置回源 文档。

开启回源设置后，当客户端首次访问 COS 源文件时，COS 发现无法命中对象，对客户端返回 302 HTTP 状态码并跳转至回源地址所对应的地址，此时对象由源站提供给客户端，从而保证访问。同时 COS 从源站复制该文件并保存至存储桶对应的目录中；第二次访问时，COS 直接命中对象并返回给客户端。

将 Ghost 博客应用中的附件存储到 COS

最近更新时间：2024-01-06 10:54:03

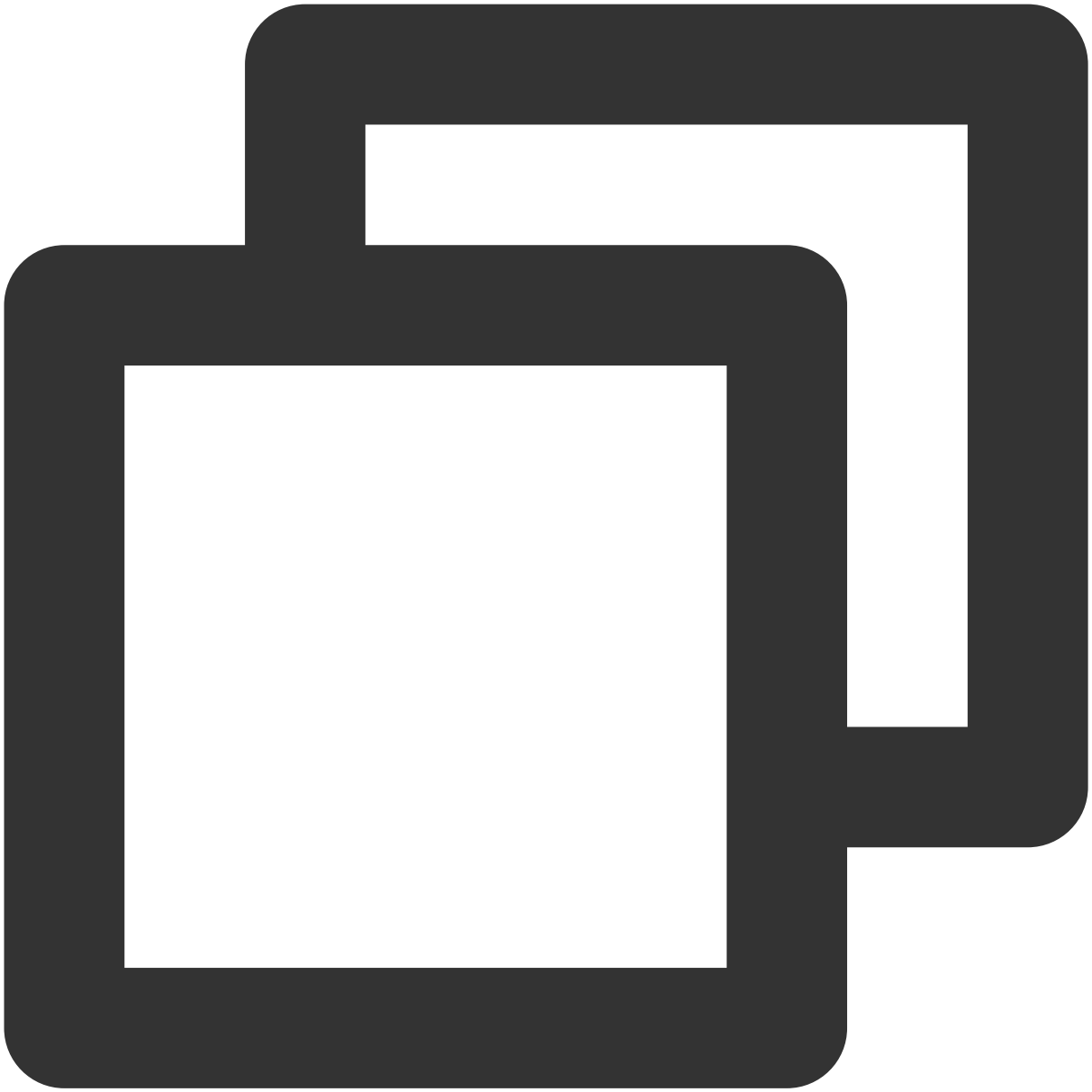
简介

Ghost 是一个基于 Node.js 快速搭建博客类网站的框架，开发者可通过 Ghost 官方 cli 工具一键生成自己的个人网站，并支持部署到云服务器和 Docker 上。
作为一个博客类网站，上传附件是必不可少的功能，Ghost 默认会将附件存储在本地，本文将介绍如何通过插件将附件保存在 腾讯云对象存储（Cloud Object Storage，COS） 上，将论坛附件保存在 COS 上有以下好处：
附件将拥有更高的可靠性。
您的服务器无需为论坛附件准备额外的存储空间。
用户查看图片附件时将直连 COS 服务器，不占用您服务器的下行带宽/流量，用户访问速度更快。
可配合 腾讯云内容分发网络（Content Delivery Network，CDN） 进一步提升论坛用户查看图片附件的速度。

准备工作

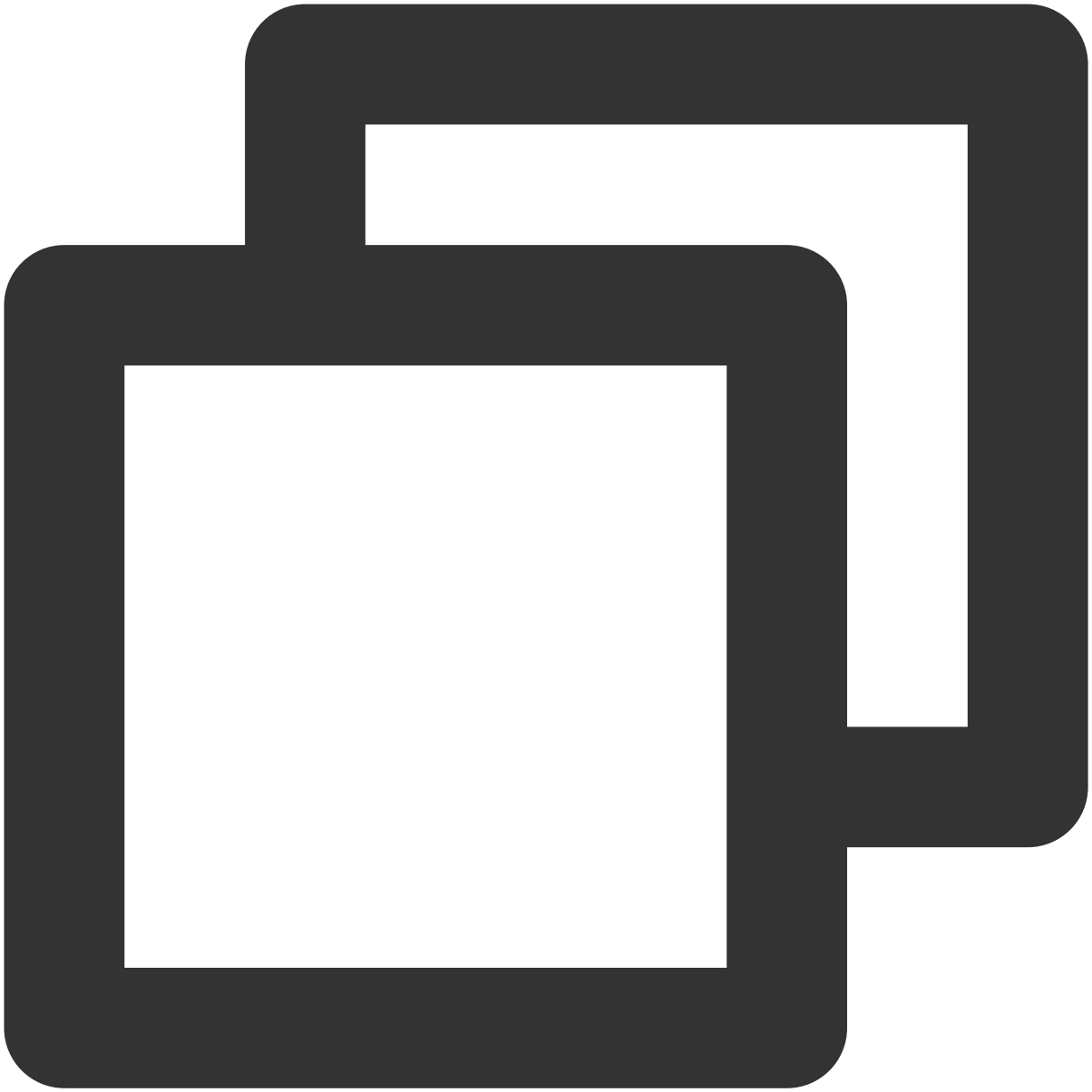
搭建 Ghost 网站

1. 安装 Node.js 环境。
2. 安装 ghost-cli。



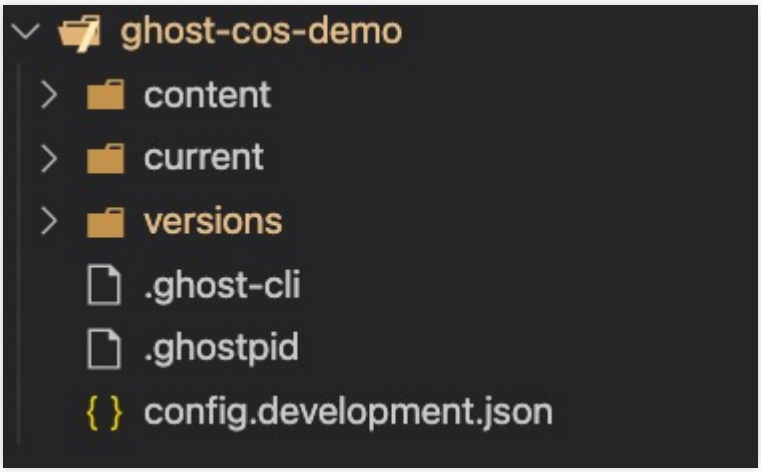
```
npm install ghost-cli@latest -g
```

3. 创建一个项目，在该项目的根目录下执行命令：

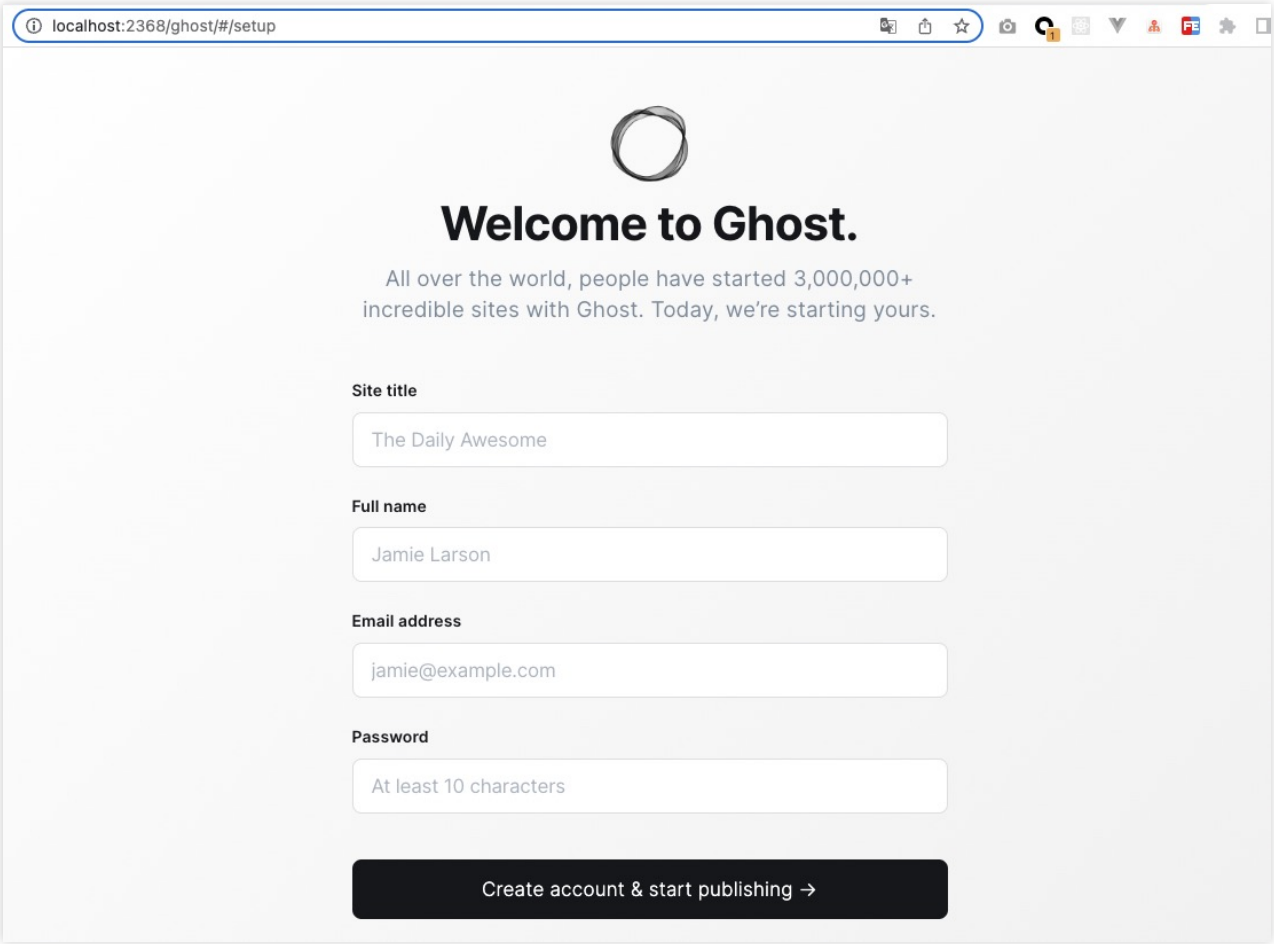


```
ghost install local
```

创建成功后的项目结构如下图所示：



4. 打开浏览器，进入 localhost:2368，出现注册页面，注册后进入管理后台。



创建 COS 存储桶

1. 在 COS 控制台 创建一个访问权限为**公有读私有写**的存储桶，操作指引可参见 创建存储桶。

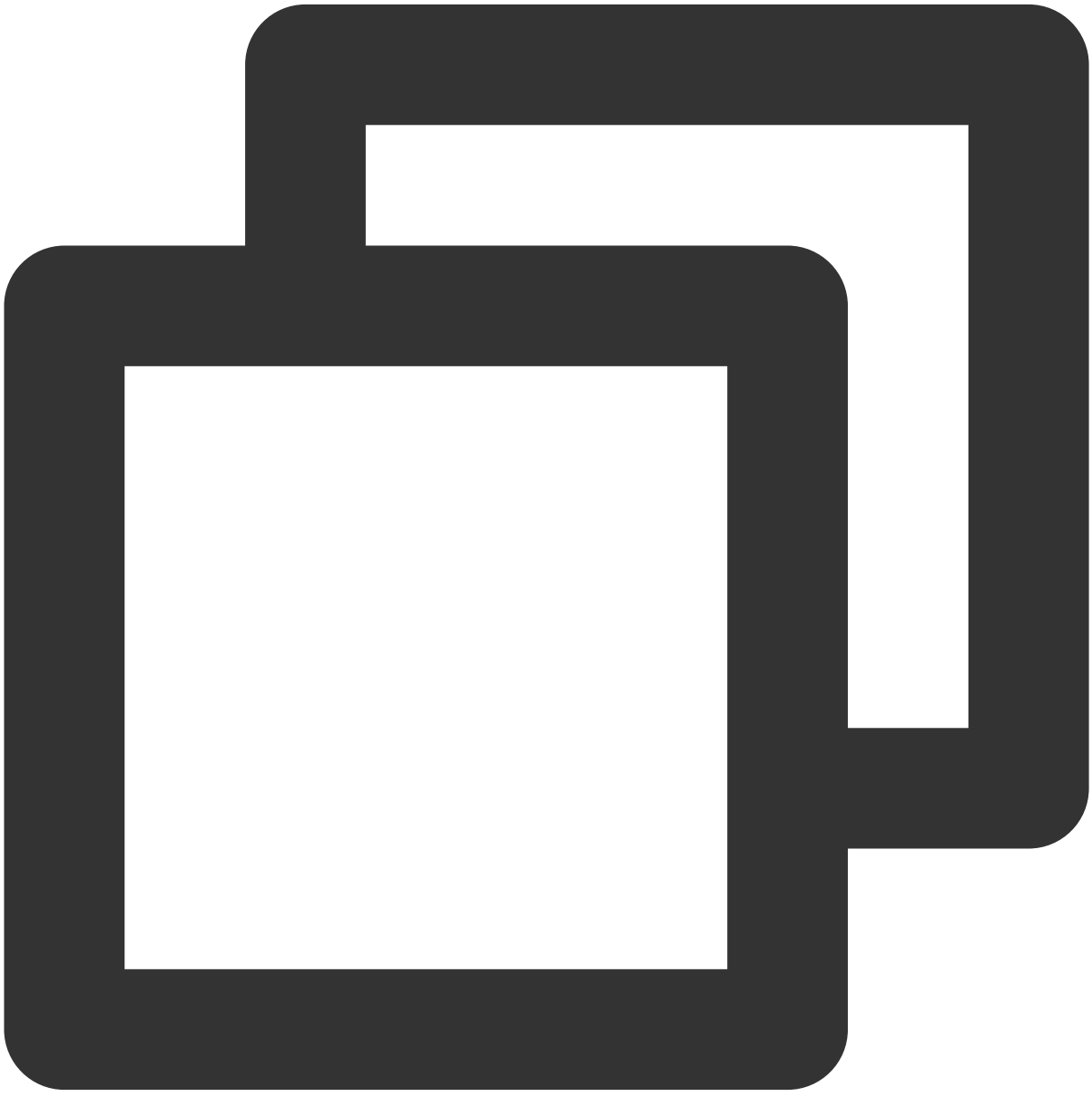
2. 单击**安全管理 > 跨域访问 CORS 设置**，添加一行跨域设置，为方便调试可使用以下配置，操作指引可参见 设置跨域访问。

将 Ghost 关联到 COS存储桶

注意

建议使用子账号密钥，授权遵循 最小权限指引，降低使用风险。子账号密钥获取可参考 子账号访问密钥管理。

1. 修改 Ghost 项目根目录下的 config.development.json 配置文件，添加如下配置：

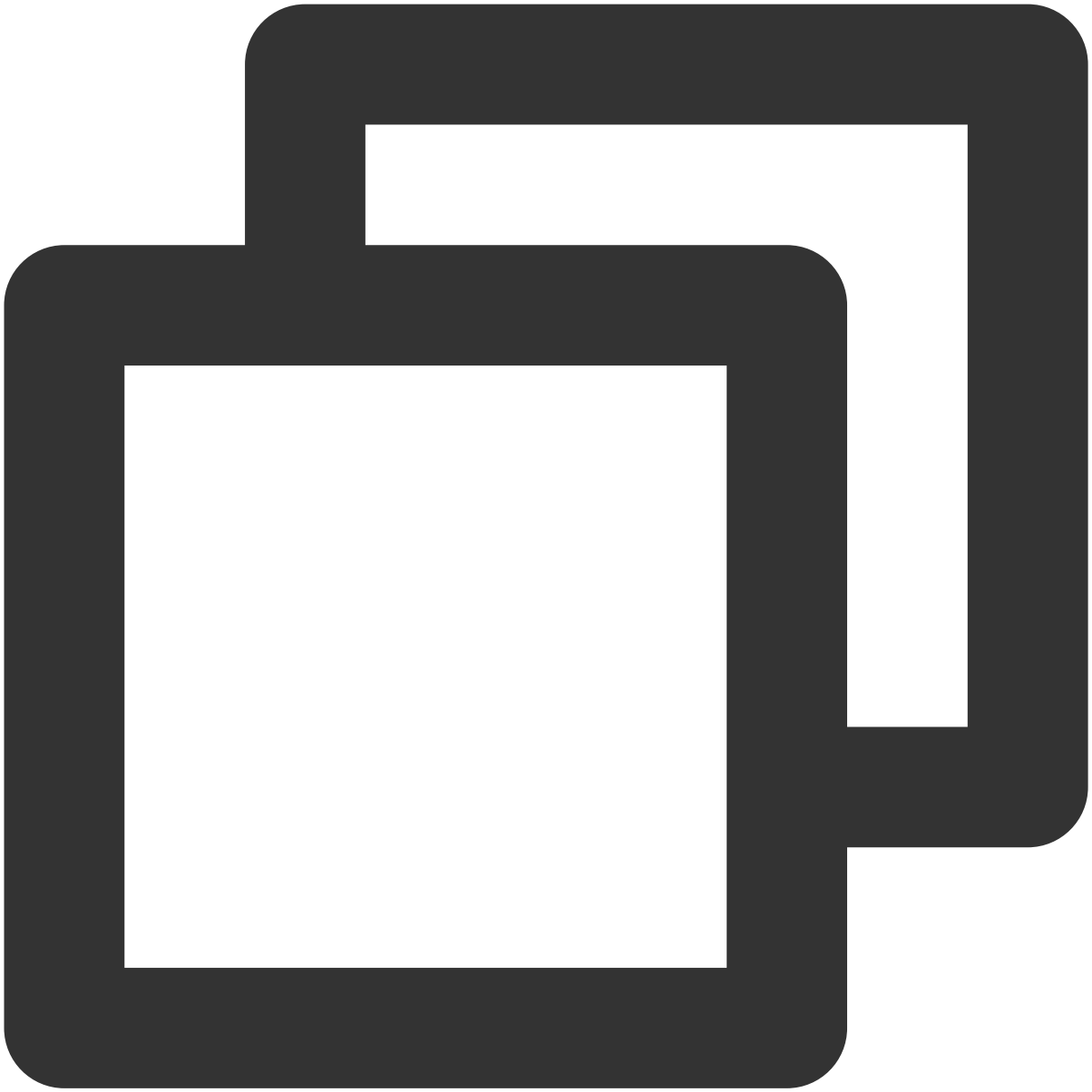


```
 "storage": {
"active": "ghost-cos-store",
"ghost-cos-store": {
  "BasePath": "ghost/", // 可修改为自己的目录名，不填写则默认根目录
  "SecretId": "AKID*************",
  "SecretKey": "***************",
  "Bucket": "xxx-125********",
  "Region": "**-*******"
}
 }
```

参数说明如下：

配置项	配置值
BasePath	文件所存储的 COS 路径，可自行修改，不填写则默认根目录
SecretId	访问密钥信息，可前往 云 API 密钥 中创建和获取
SecretKey	访问密钥信息，可前往 云 API 密钥 中创建和获取。
Bucket	创建存储桶时自定义的名称，例如 examplebucket-1250000000。
Region	创建存储桶时所选择的地域。

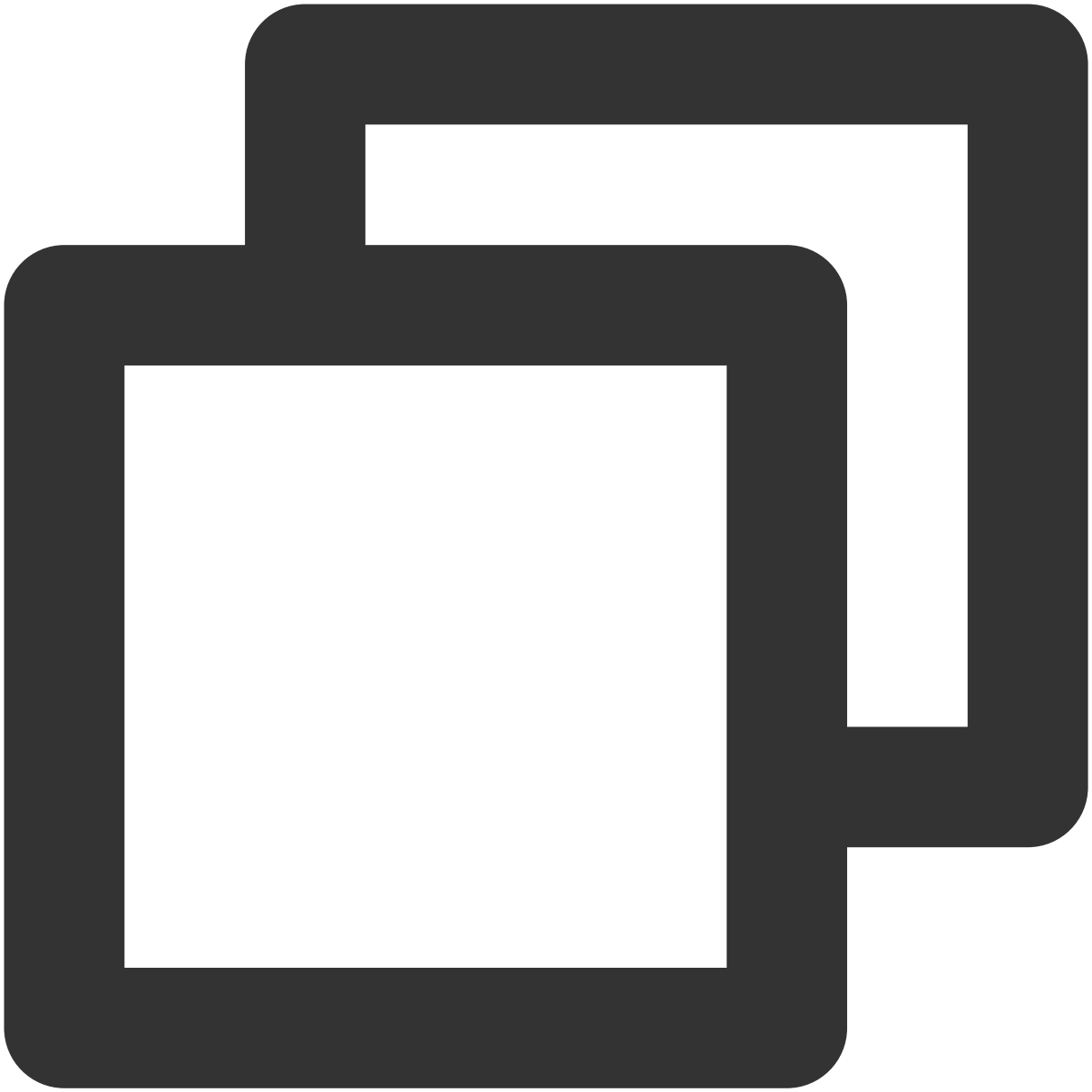
2. 创建自定义存储目录，在项目根目录下执行：



```
mkdir -p content/adapters/storage
```

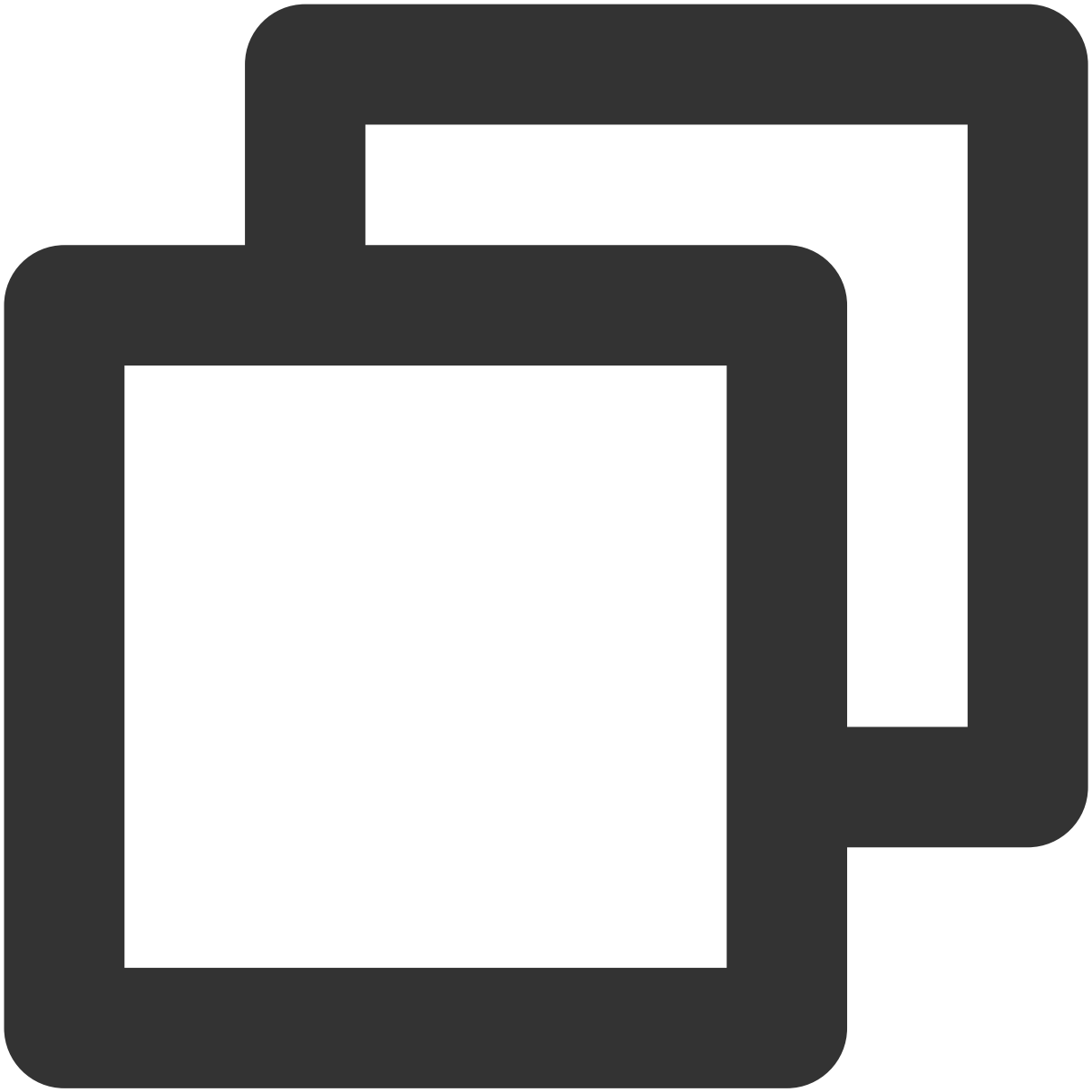
3. 安装腾讯云官方提供的 ghost-cos-store 插件。

3.1 通过 npm 安装。



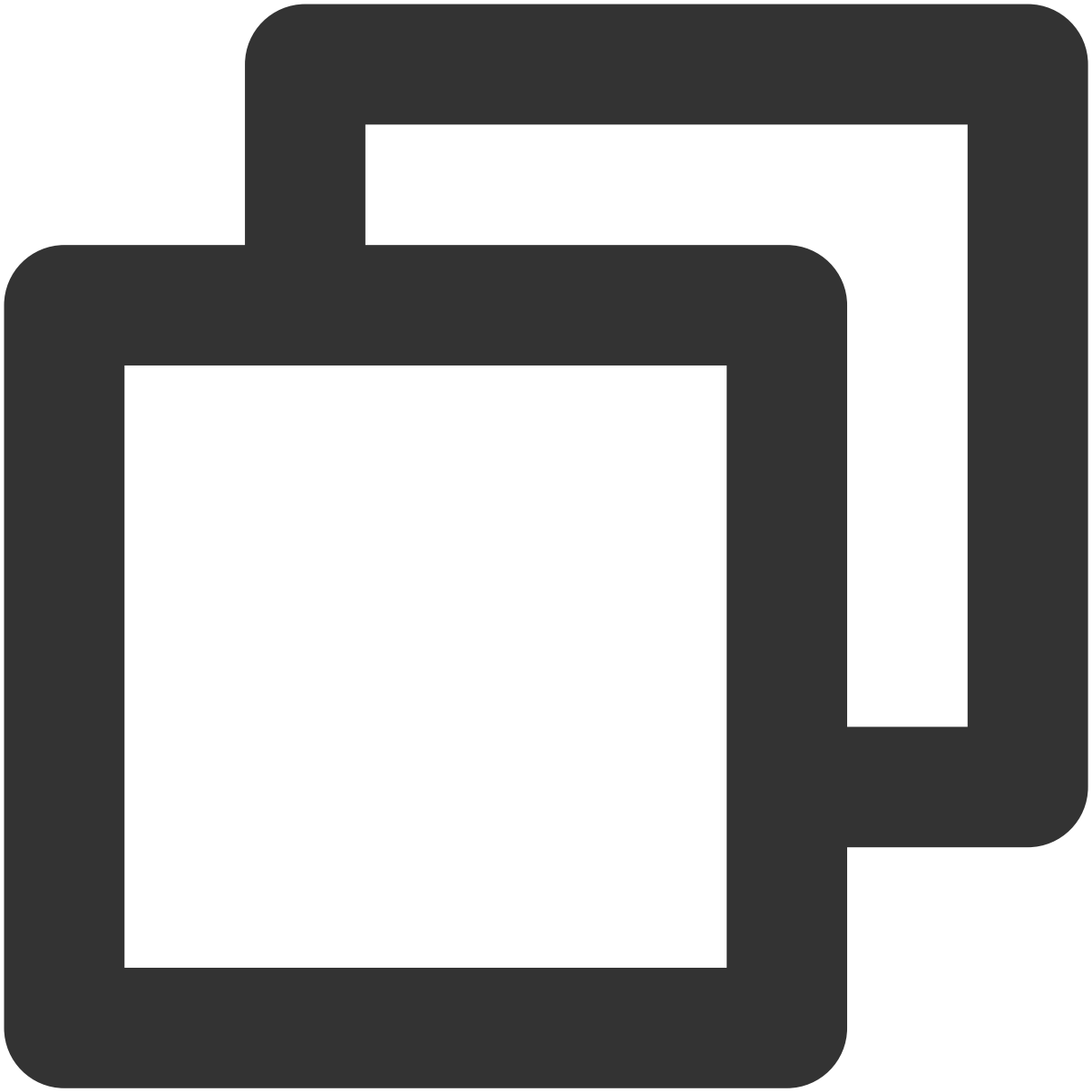
```
npm install ghost-cos-store
```

3.2 在 storage 目录下创建 ghost-cos-store.js 文件，内容如下：



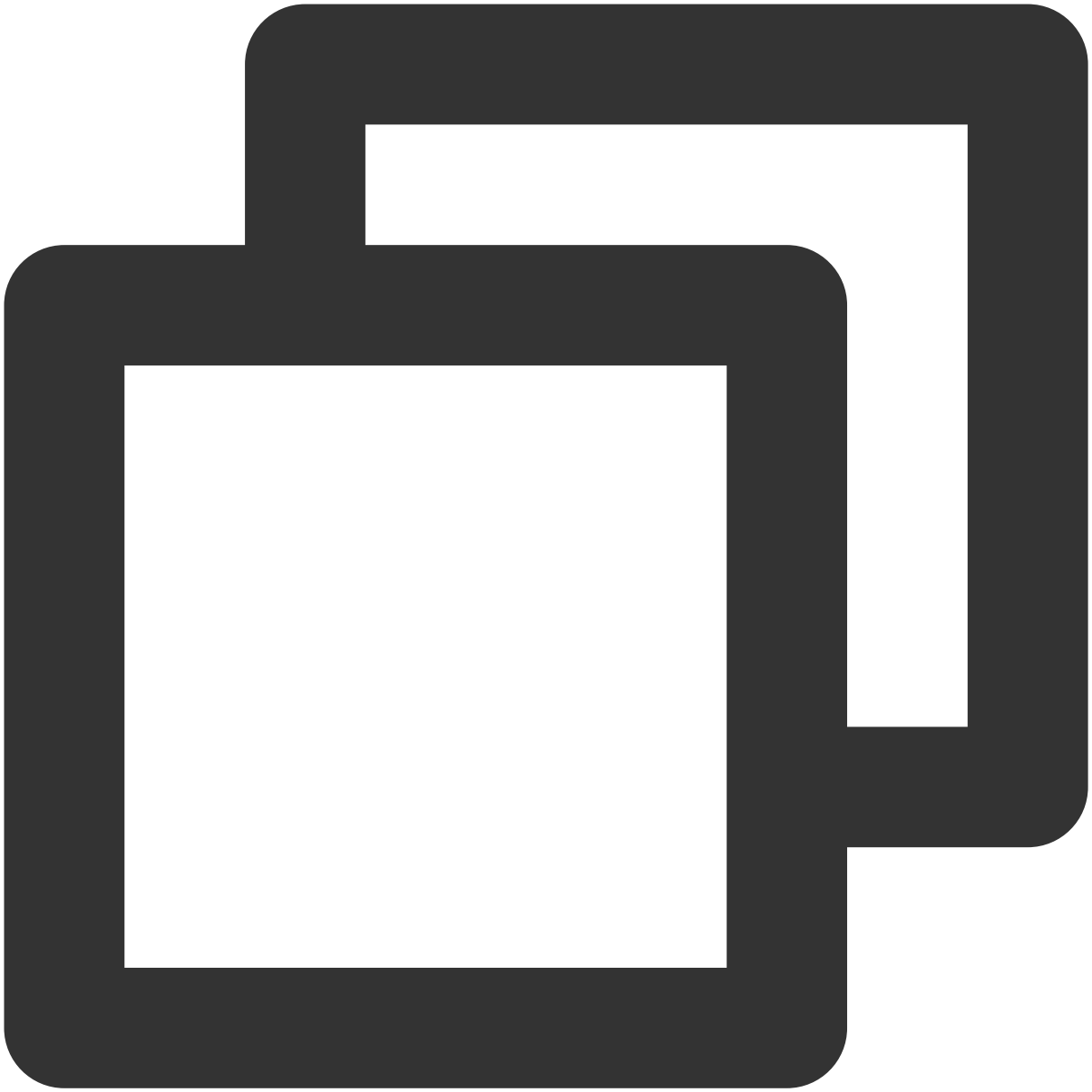
```
//  content/adapters/storage/ghost-cos-store.js
module.exports = require('ghost-cos-store');
```

3.3 通过 git clone 安装。



```
cd content/adapters/storage
git clone https://github.com/tencentyun/ghost-cos-store.git
cd ghost-cos-store
npm i
```

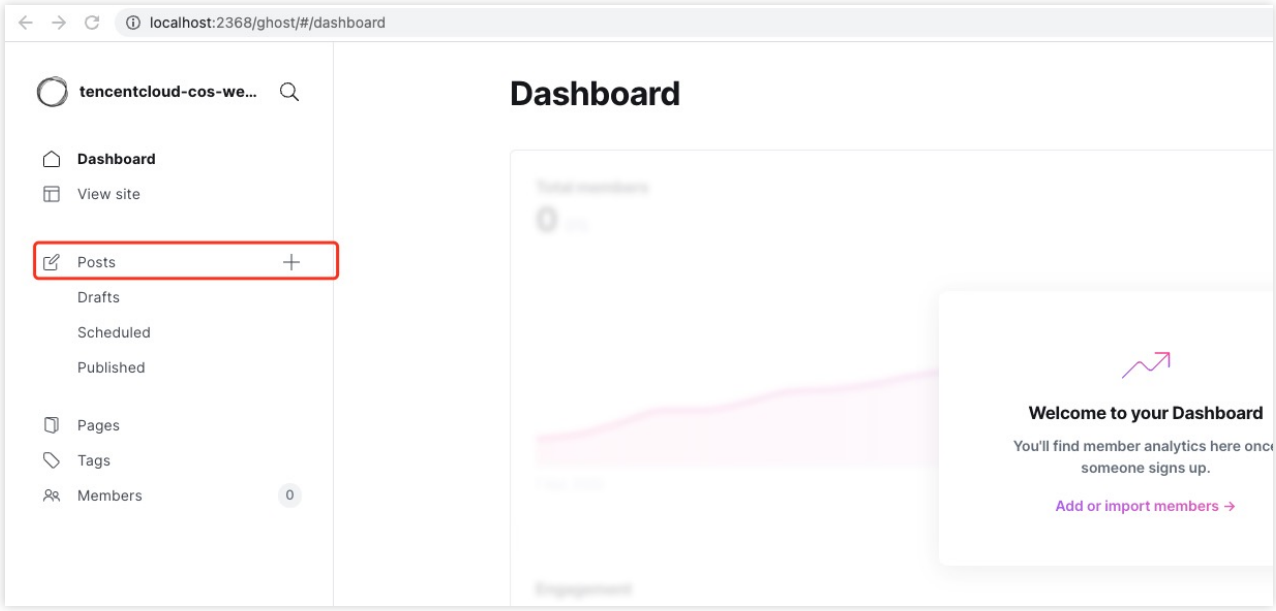
3.4 安装完成后，需要重启 Ghost。



```
ghost restart
```

发文并进行上传测试

1. 进入 Ghost 管理后台，单击发表一篇文章。



2. 单击上传图片，在浏览器抓包可以看到 upload 请求成功，并返回了图片对应的 COS 链接。

将个人计算机中的文件备份到 COS

最近更新时间：2024-01-06 10:54:03

前言

数据无价，相信很多人都深有体会。数码照片、电子文档、工作产出、游戏存档，哪一样丢失都很头疼。除了硬盘故障导致的文件丢失，人为的误操作、计算机宕机或软件崩溃导致的单一文件丢失，以及被要求“回滚版本”却发现没有保存历史版本的尴尬，都是工作和生活中令人头疼不已的问题。因此，备份的重要性，毋庸置疑。

说起备份，很多人想到的就是使用移动硬盘或者在局域网内搭建 NAS 存储，然后将文件往里面上传就行了。但实际上真的这么简单吗？

备份，其实是一个系统工程。除了将文件复制到备份媒介上，还需要验证备份内容的准确性。而复制与验证这两项工作，还需要定期去执行，这样在发生文件丢失时，才能最大限度挽回损失。此外，备份媒介也是需要去维护的，需要及时将损坏的硬盘进行替换。

综上所述，那么有没有简单的办法可以保证文件的安全呢？答案是肯定的。

随着云服务的发展，我们有可靠的企业级云存储服务，腾讯云对象存储（Cloud Object Storage，COS）就是这样一类服务；随着国家提速降费的号召，宽带越来越快，而且越来越便宜，让我们将文件备份上云成为现实。接下来，我们就需要一款软件，打通计算机中的文件和云存储，将我们的文件定期自动备份到云上，并定期验证备份文件的准确性。

软件介绍

Arq® Backup 是一款支持 Windows 和 macOS 系统的商业备份软件，该软件在系统后台运行，根据配置每隔一段时间自动备份指定的目录，同时软件会保留每个时间点备份的文件，因此可以轻松的找到某个文件的历史版本。此外，每个时间点的备份只会备份有差异的文件，对于不同路径的重复文件也只备份一次，使备份体积尽可能小，备份速度尽可能快。在将备份文件传输到网络之前，软件会基于用户输入的密码对备份文件进行加密，保证其在网络传输过程中或在云端存储中都不会被盗用，保证用户敏感数据的安全性。

Arq® Backup 商业授权为49.99美元每个用户，用户购买后可以在单台计算机上使用，同时软件提供30天免费使用，可以试用后再购买。

说明

Arq® Backup 软件目前暂时没有简体中文版，软件的下载、购买和相关说明均可在该软件 官方网站 内查看。

准备腾讯云对象存储

说明

若您目前已经在使用 COS，请忽略1 - 2步骤。

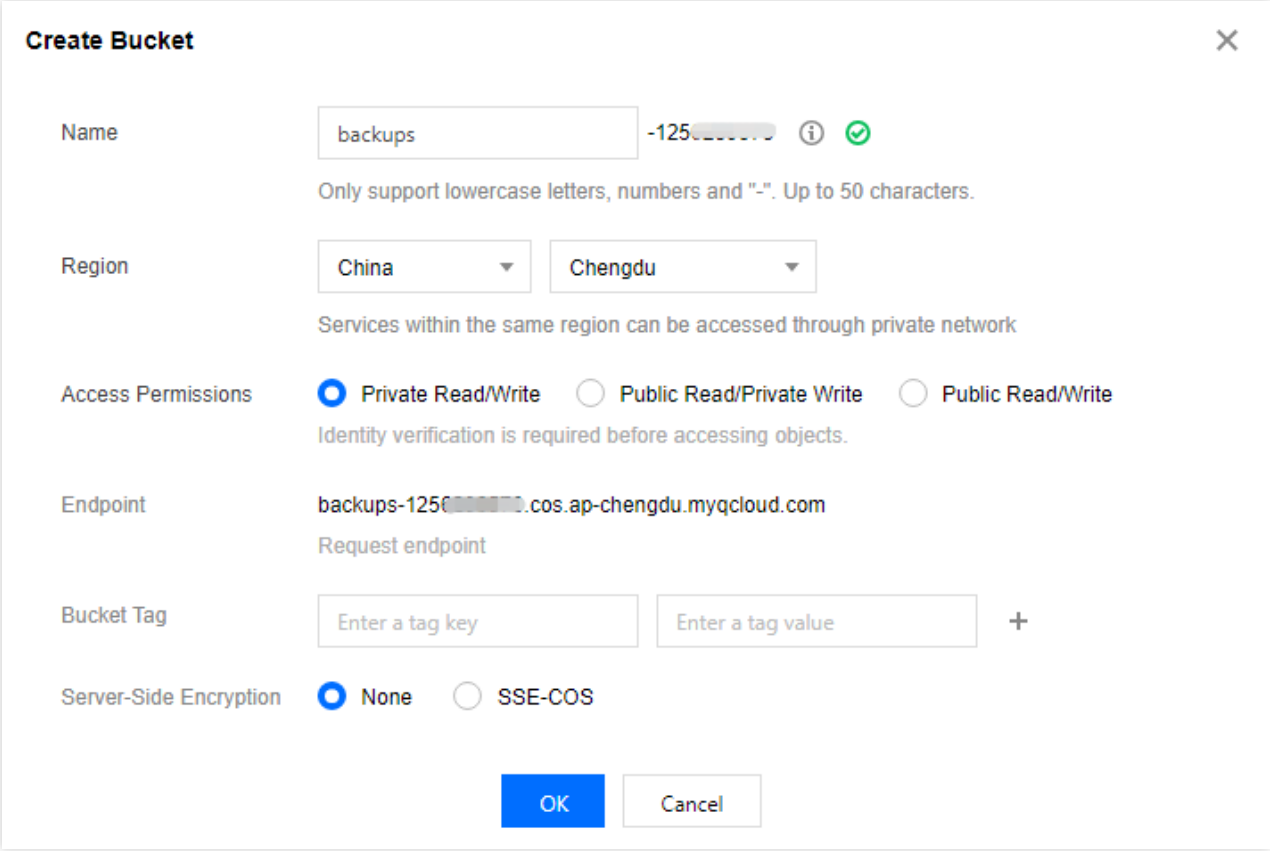
1. 注册腾讯云账号 并完成 实名认证，未进行实名认证的用户无法购买中国境内的资源。

2. 登录 对象存储 COS 控制台，按照提示开通 COS。

3. 在对象存储 COS 控制台中，单击左侧导航栏的**存储桶列表**，然后单击**创建存储桶**，开始创建存储桶：

名称：存储桶名称，例如 “backups”。

所属地域：可以根据您所在地就近选择，但是请不要选择金融地域，目前我们对于西南地区有价格上的优惠，因此也可以选择“成都”或“重庆”享受更优惠的价格。

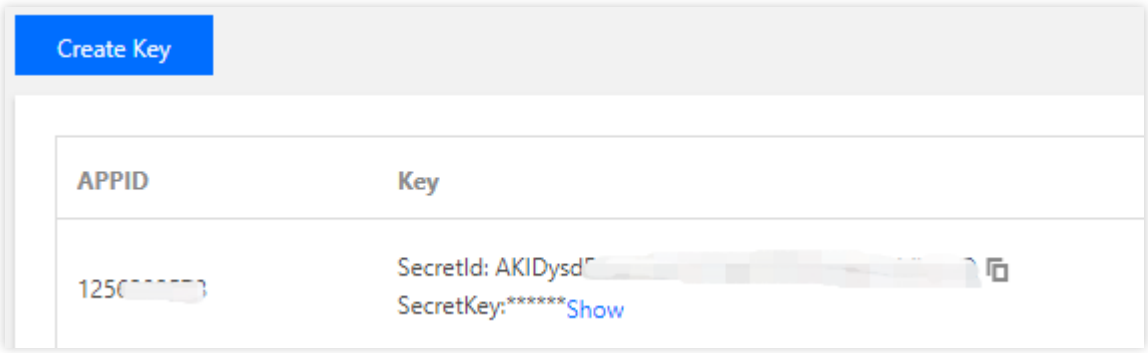


其他配置项保持默认，将**请求域名**地址复制保存，然后单击**确定**完成创建。

说明

创建存储桶的详细步骤，请参见 创建存储桶。

4. 登录 API 密钥管理控制台 ，创建并记录密钥信息 SecretId 和 SecretKey。



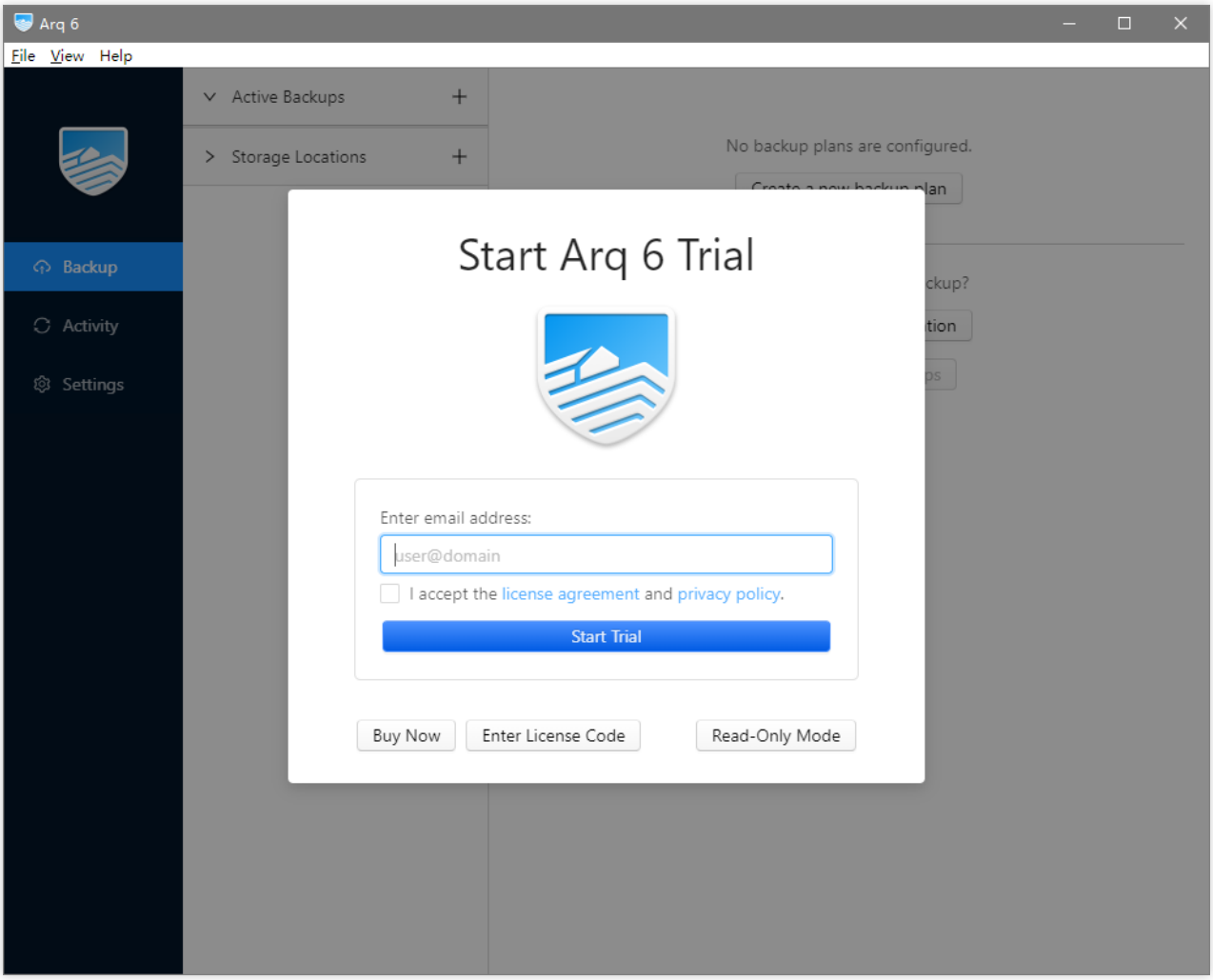
安装并配置 Arq® Backup

说明

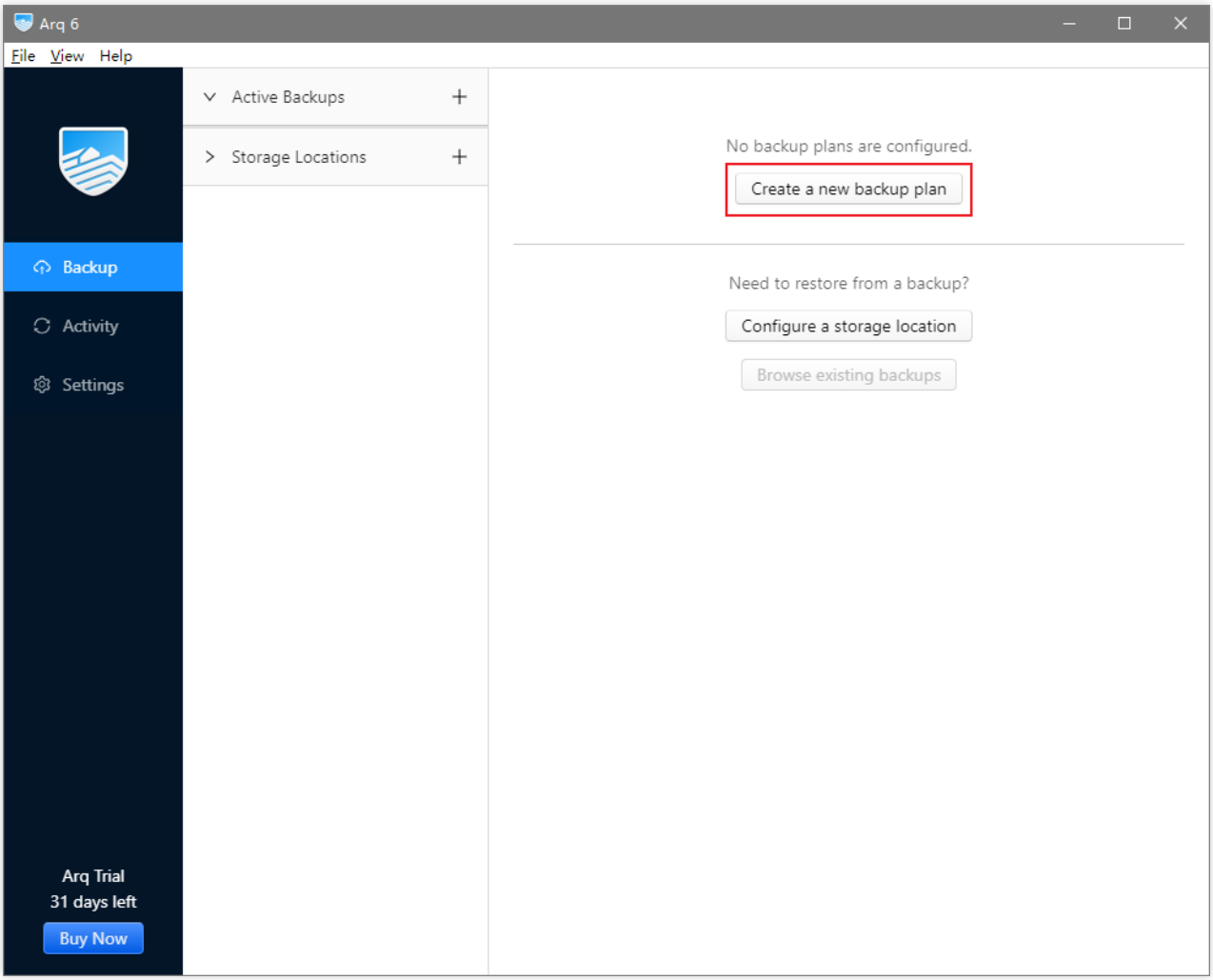
本文以 Windows 的 Arq® Backup 6.2.11版本为例。

1. 从 Arq® Backup 官网 下载软件。

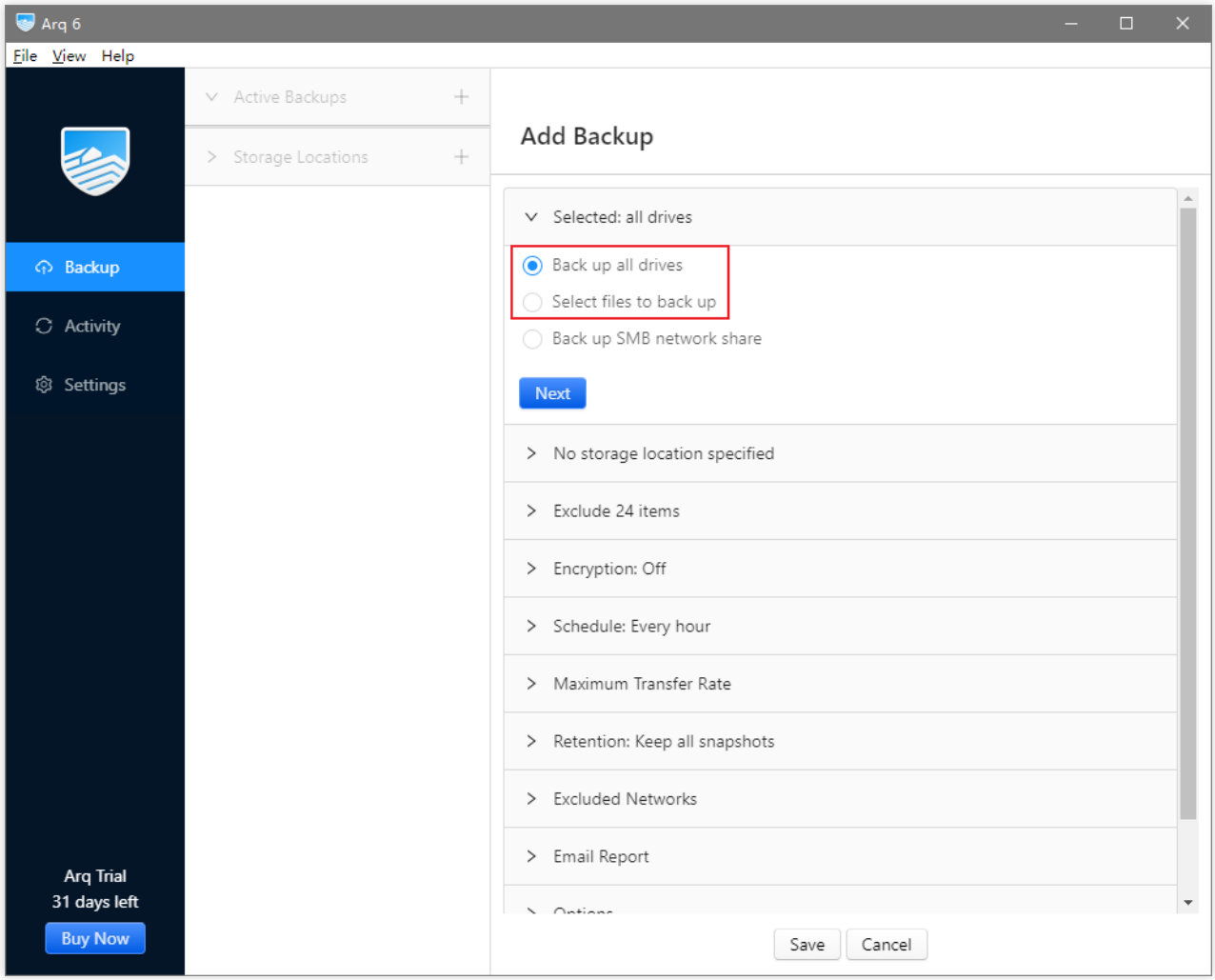
2. 按提示完成软件安装，安装完成后软件会自动启动，首次启动时会提示登录，此时输入邮箱地址并单击 **Start Trial**。



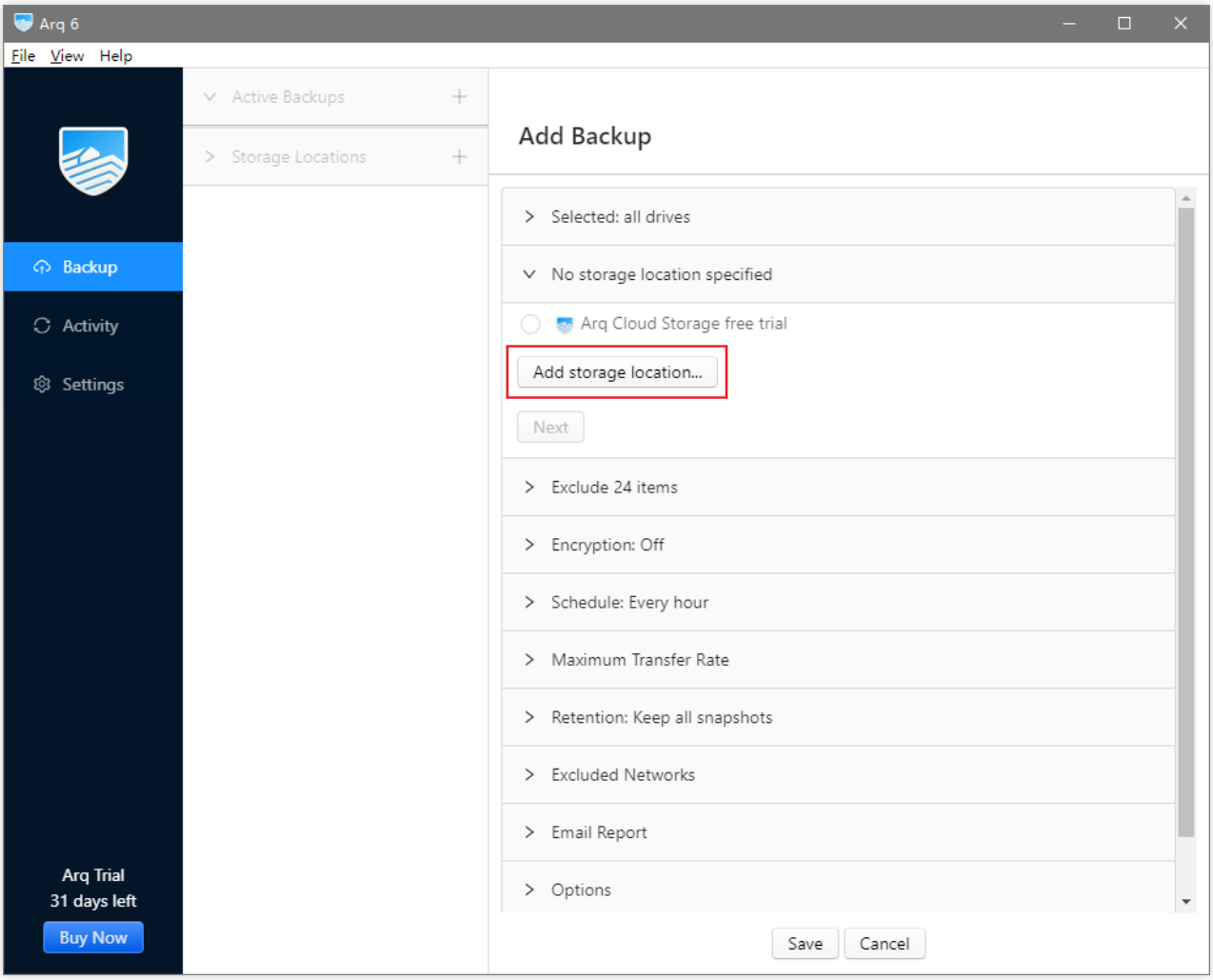
3. 在 **Backup** 界面中单击 **Create a new backup plan**，添加备份计划。



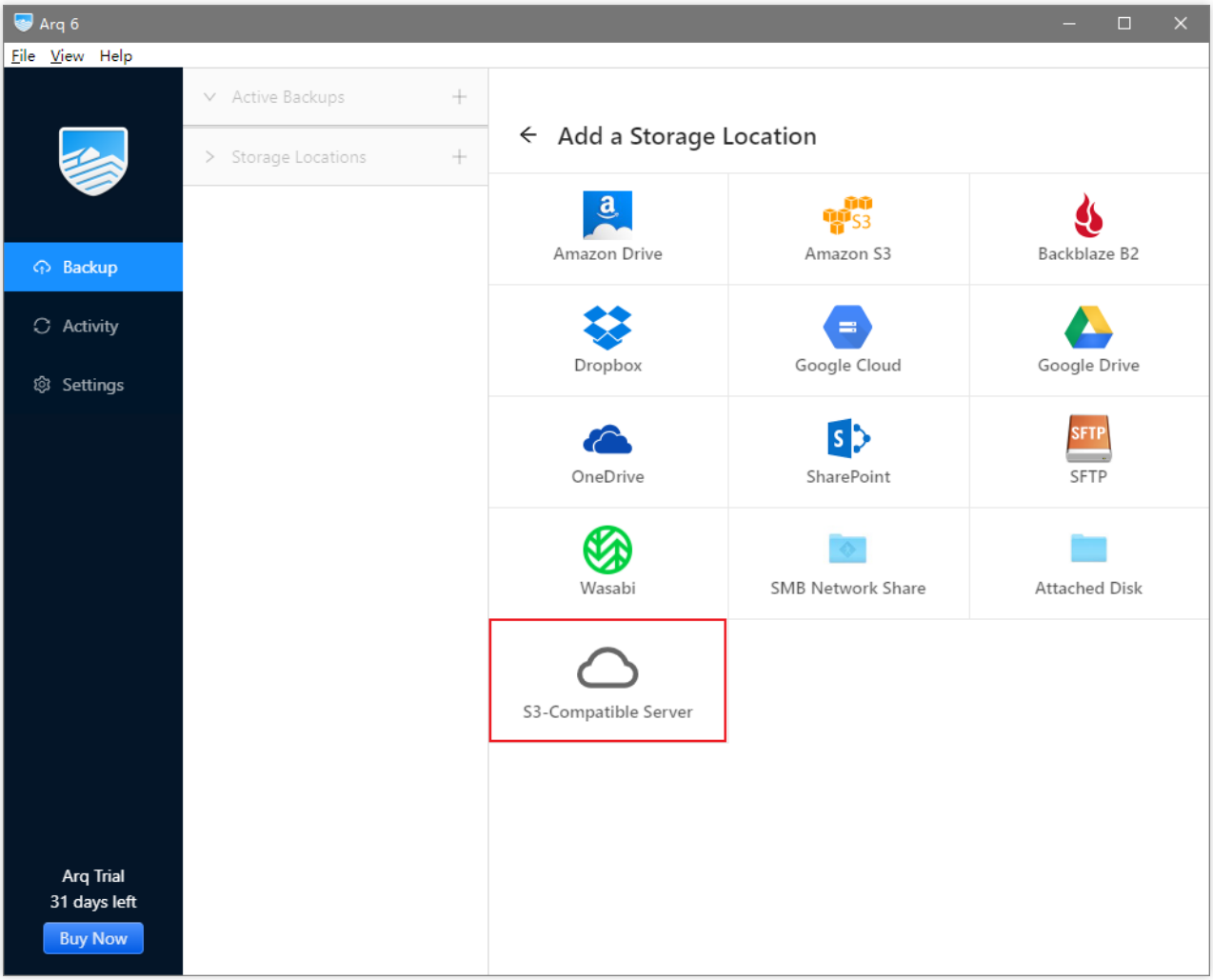
4. 在跳转界面，选择需要备份的目录，可以选择所有硬盘或指定目录。



5. 单击 **Add storage location**，添加备份存储位置，如下图所示。



6. 此处我们选择 **S3-Compatible Server**。

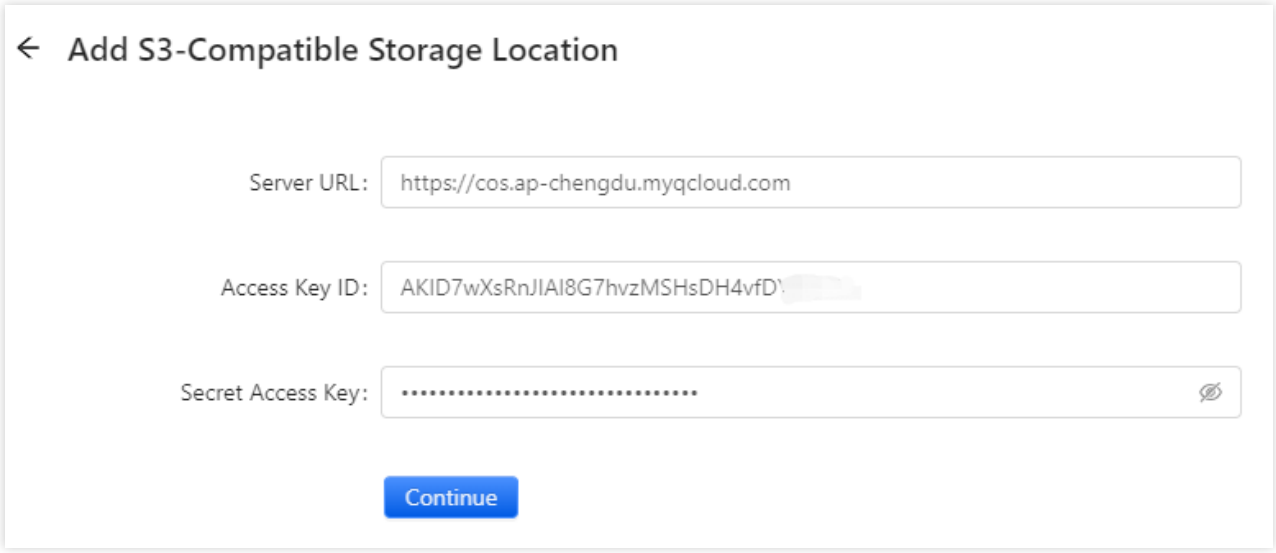


7. 在跳转界面中按照以下说明进行配置。配置完毕后，单击 **Continue**。

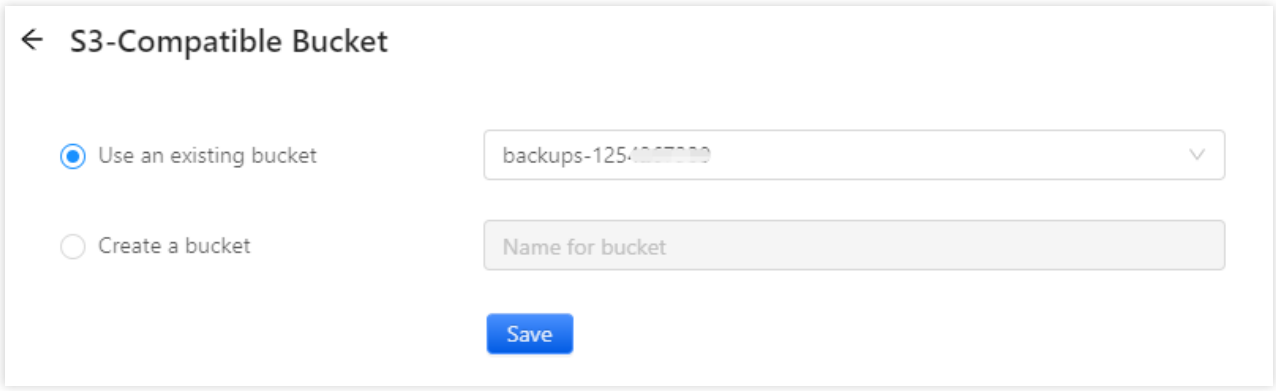
Server URL：输入上文记录的请求域名中，从 `cos` 开始的部分，并在前面加上 `https://`，例如 `https://cos.ap-chengdu.myqcloud.com`，请注意这里不包含存储桶名称。

Access Key ID：上文记录的密钥信息中的 SecretId。

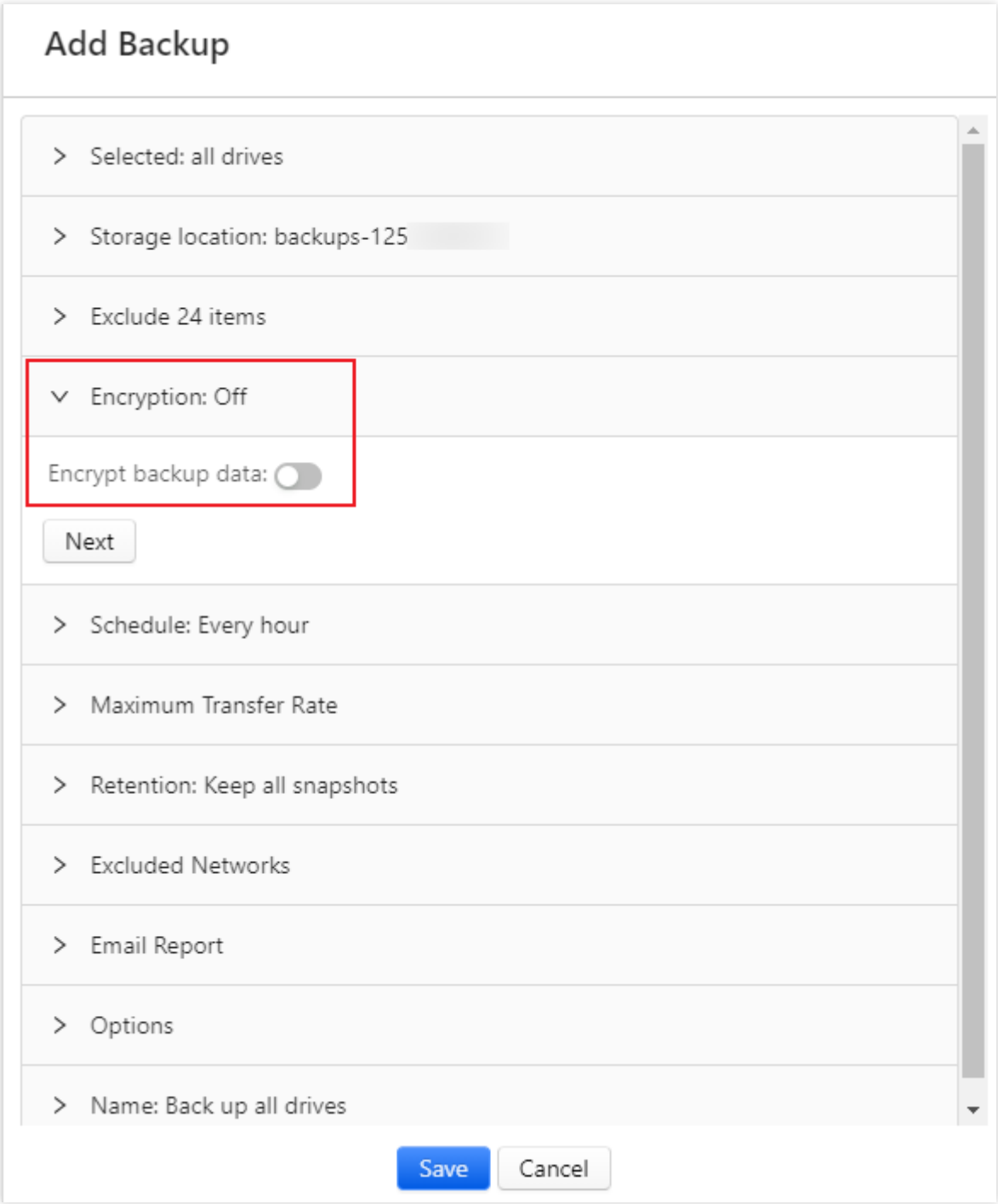
Secret Access Key：上文记录的密钥信息中的 SecretKey。



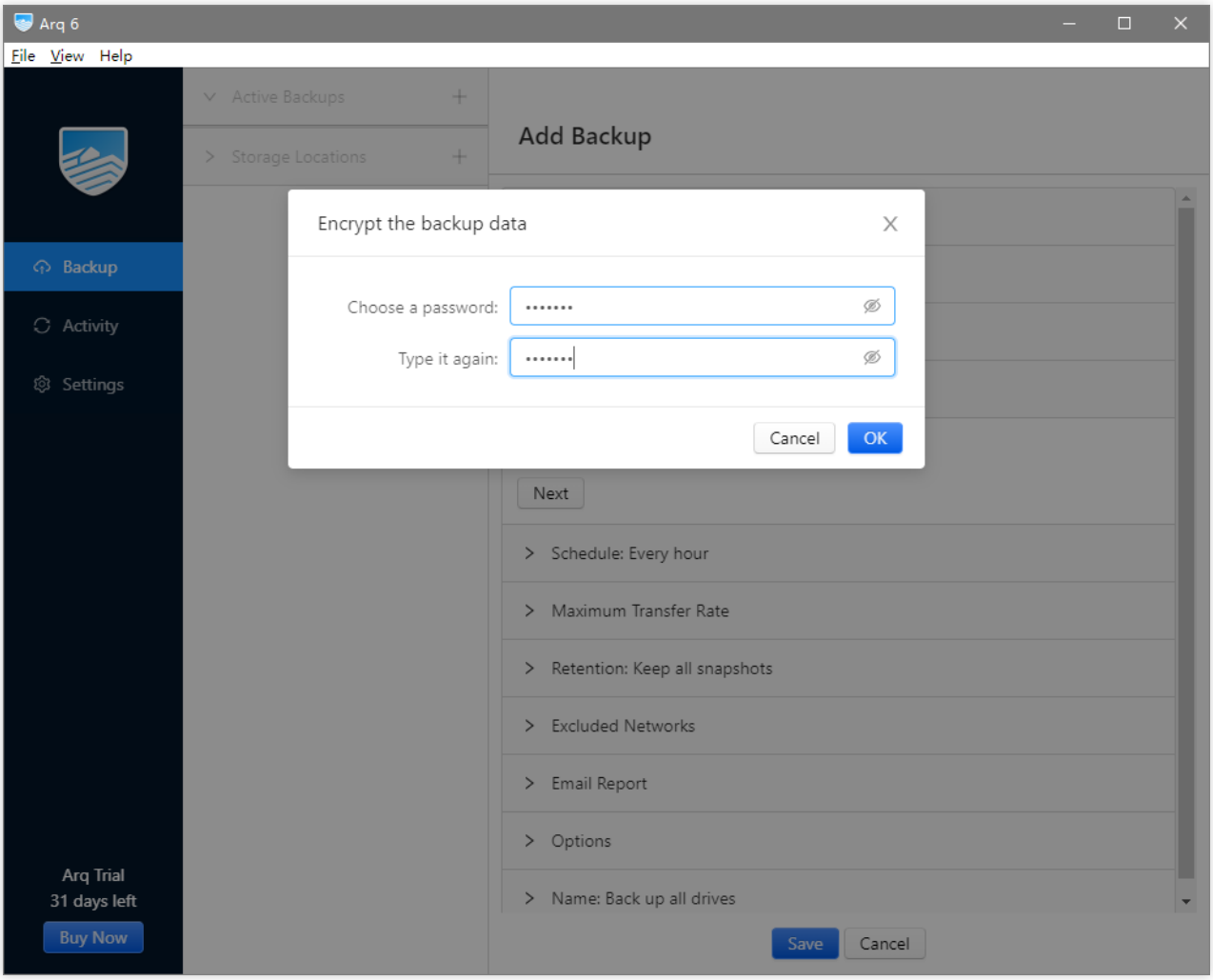
8. 在随后的界面中选择 **Use an existing bucket**，并选择上文创建的存储桶，例如 **backups-1250000000**，然后单击 **Save**。



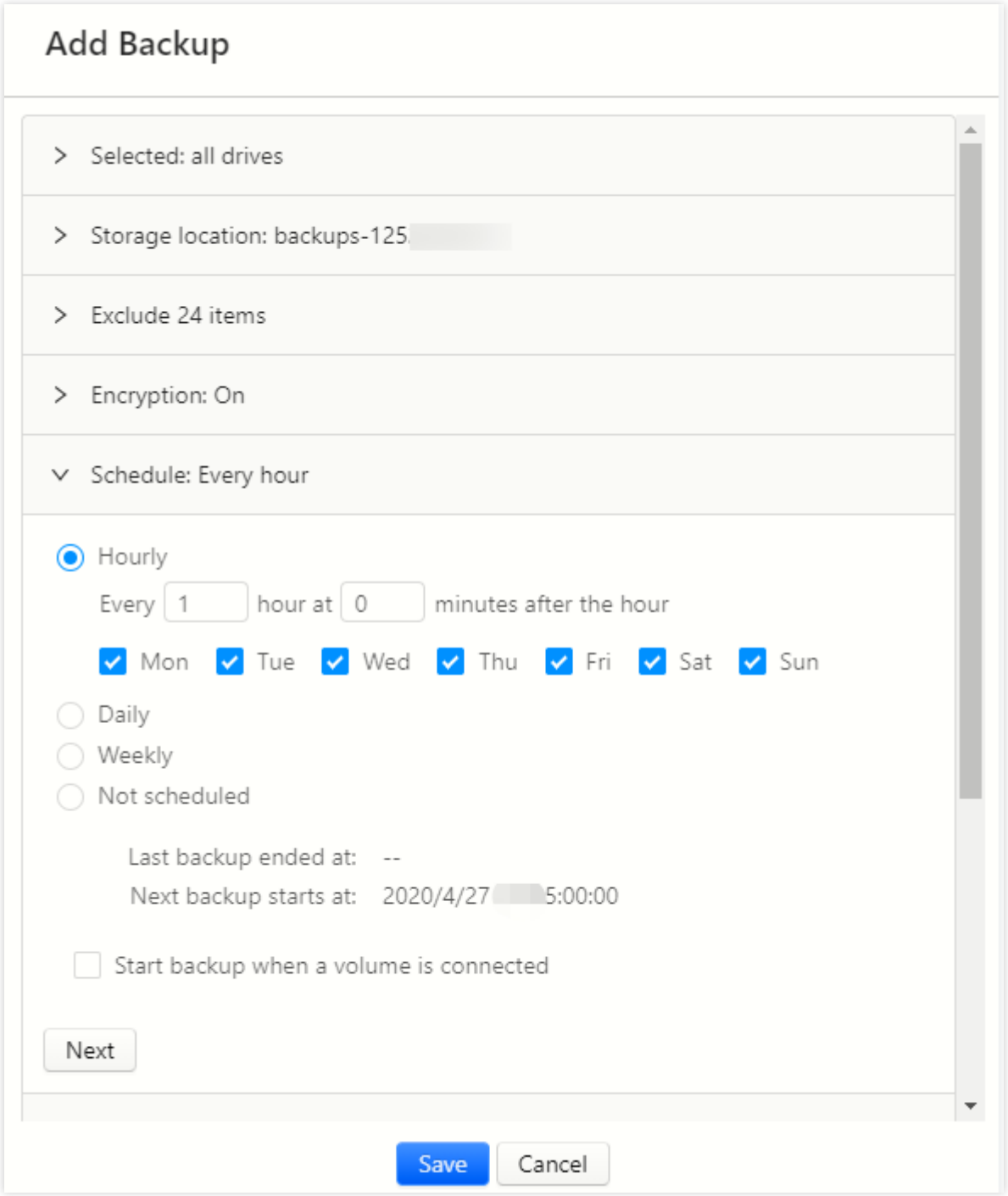
9. （可选）选择是否加密备份数据，此处我们选择**开启**按钮。



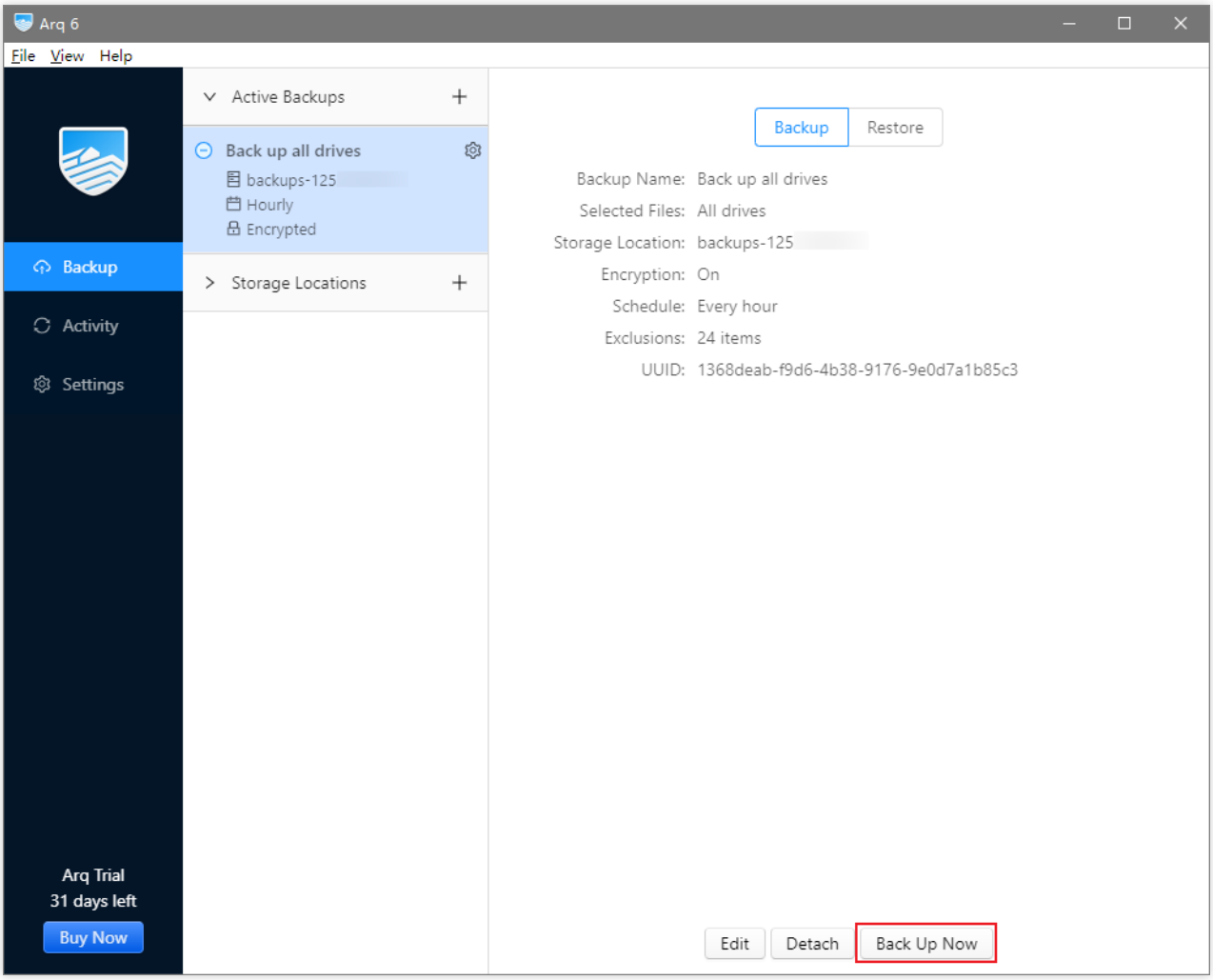
10. 在弹窗中设置用于加密的密码。输入两次用于加密备份文件的密码，并单击 **OK**。**注意请牢记备份密码，否则将无法从备份恢复文件！**


Arq 6
File View Help
Active Backups
Storage Locations
Backup
Activity
Settings
Arq Trial
31 days left
Buy Now
Add Backup
Encrypt the backup data
Choose a password:
Type it again:
Cancel
OK
Next
Schedule: Every hour
Maximum Transfer Rate
Retention: Keep all snapshots
Excluded Networks
Email Report
Options
Name: Back up all drives
Save
Cancel

11. （可选）设置备份周期。

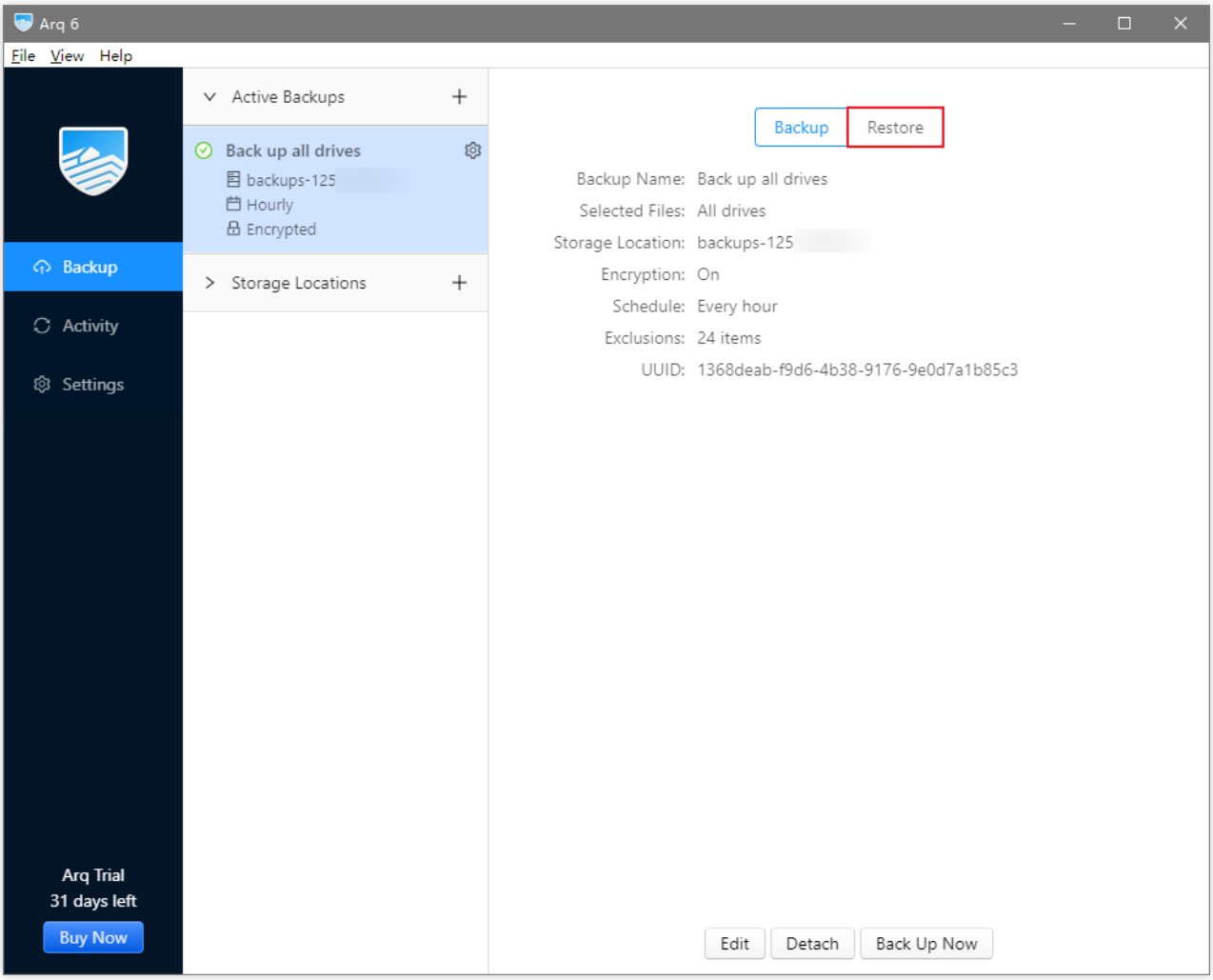


12. 单击**Save**保存设置，然后单击**Back Up Now**开始备份。

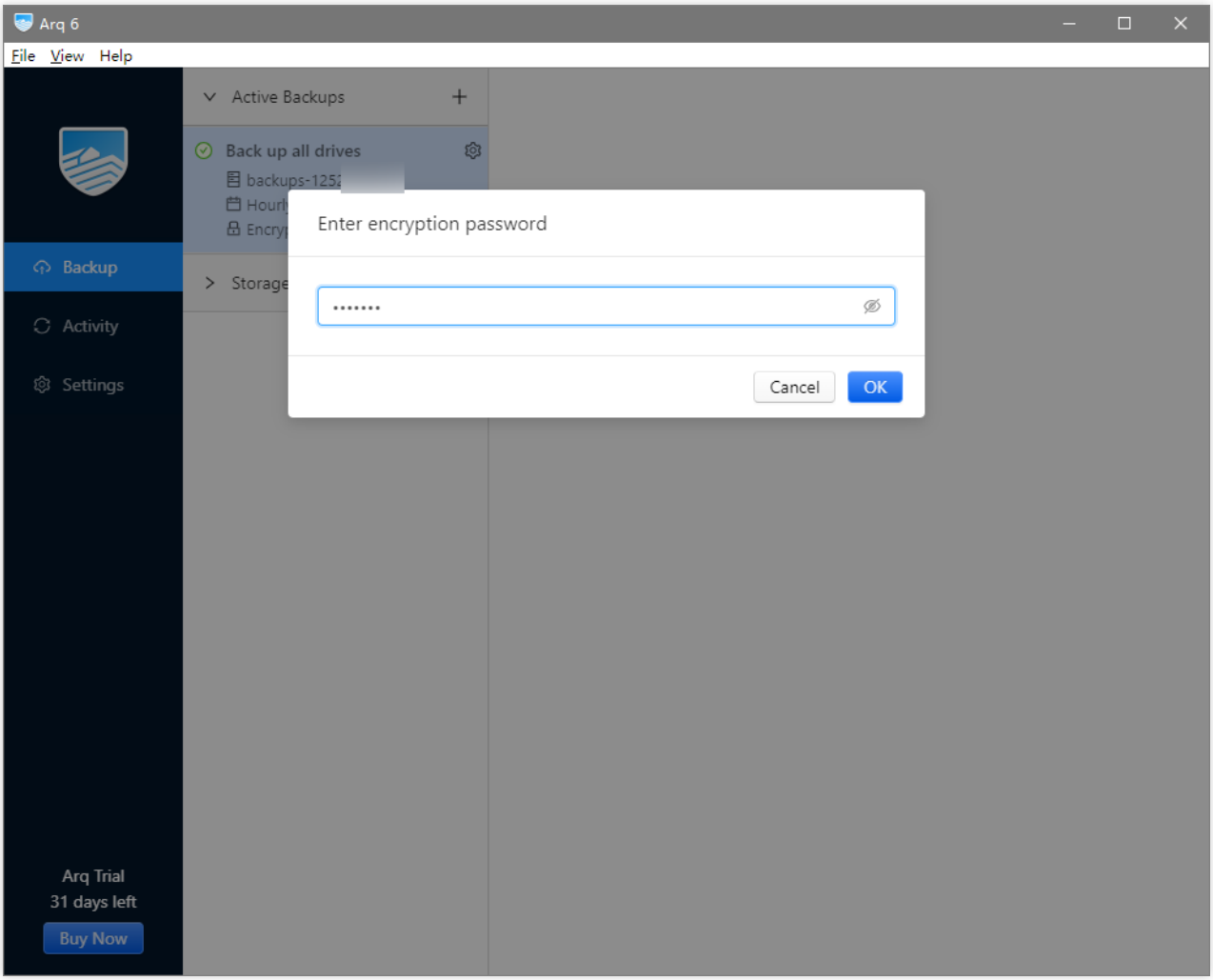


从备份中恢复文件

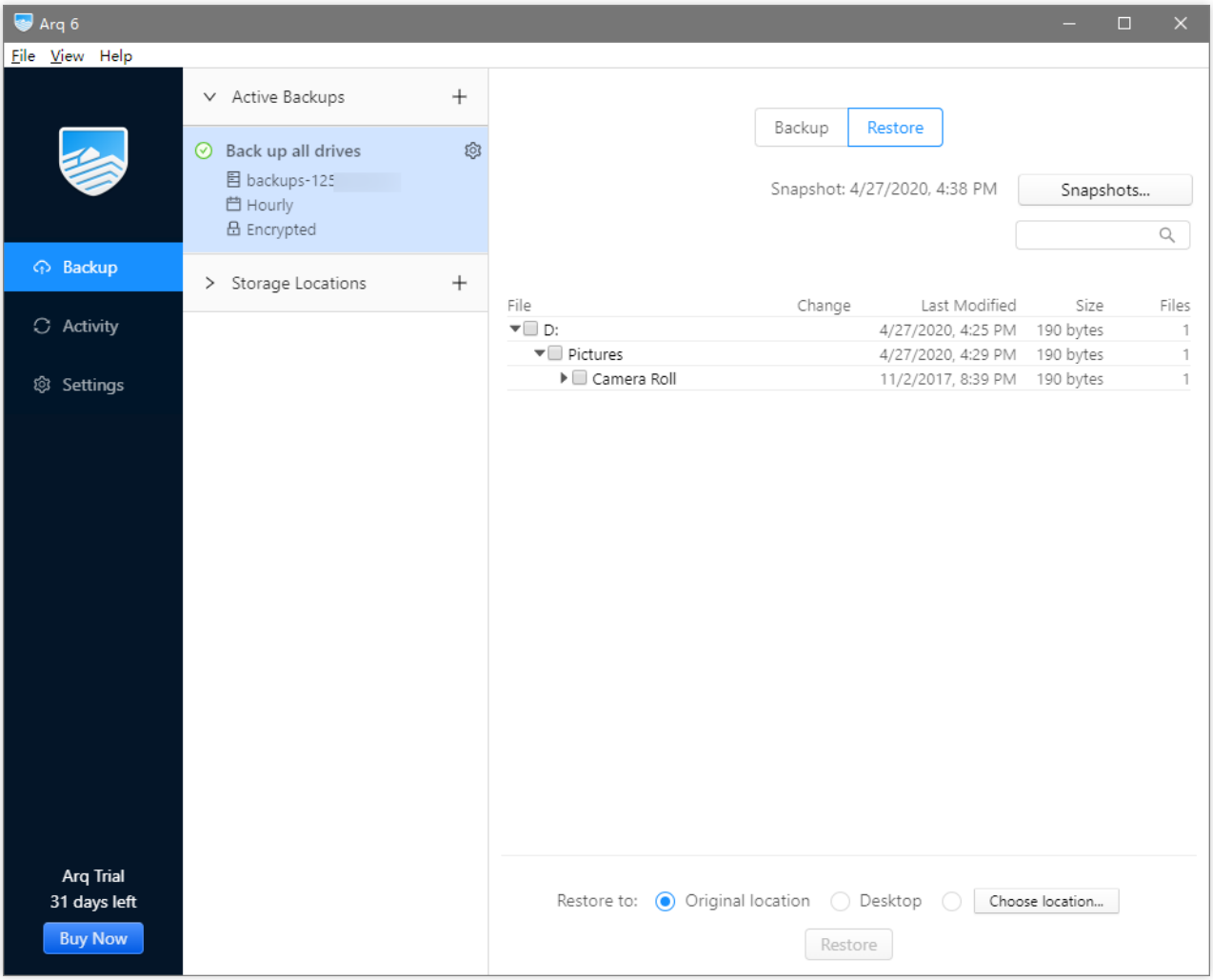
1. 在主界面左侧 **Backup** 列表中，单击 **Restore**。



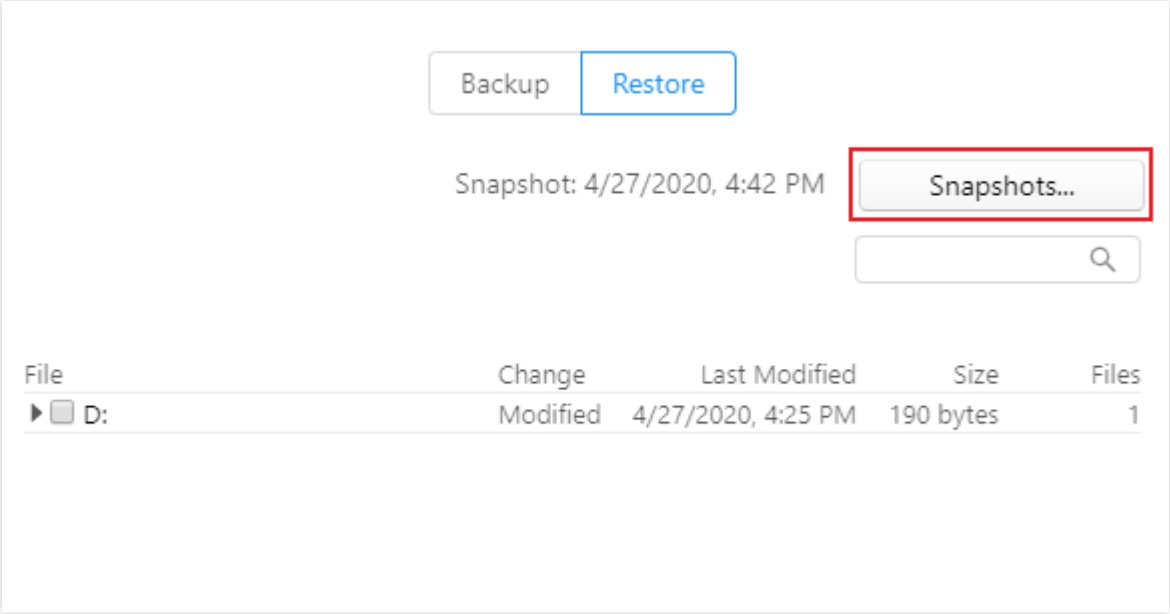
2. 如果按照上面第9步设置了加密备份数据，则需要输入密码。



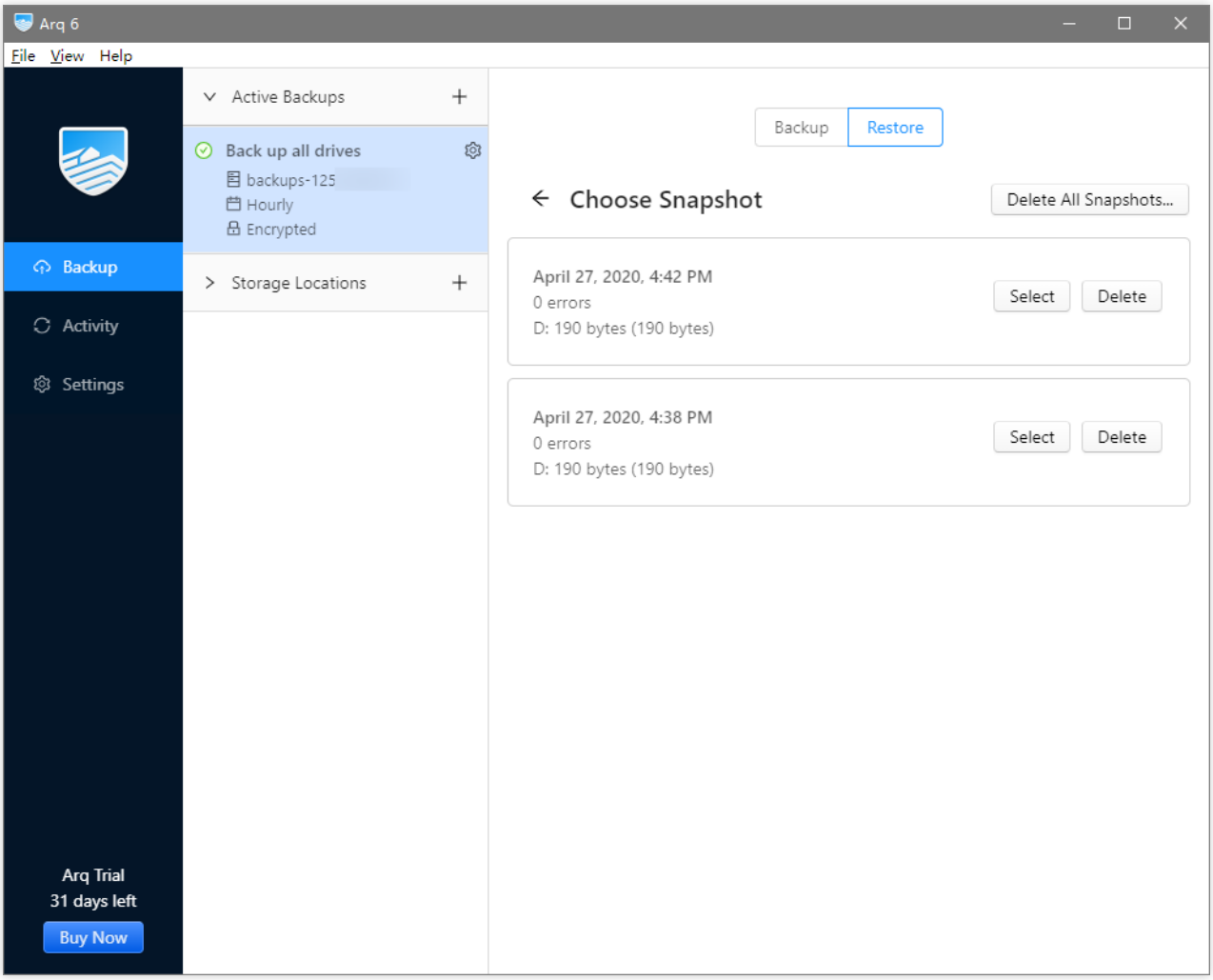
3. 选择要恢复的目录或文件，以及保存恢复目录或文件的位置，单击 **Restore** 开始恢复。



4. 恢复操作默认是从最新的备份中恢复，如果有需要，可以从快照中找到历史版本的备份，并从历史版本的备份中恢复。单击 **Snapshots** 查看历史快照。


Backup
Restore
Snapshot: 4/27/2020, 4:42 PM
Snapshots...
File
Change
Last Modified
Size
Files
D:
Modified
4/27/2020, 4:25 PM
190 bytes
1

5. 选择历史快照。



6. 选择要恢复的历史目录或文件，以及保存恢复目录或文件的位置，单击 **Restore** 开始恢复。
7. 等待界面提示恢复完成，即可到刚才指定的目录中查看恢复的文件。

使用 NextCloud + COS 搭建个人网盘

最近更新时间：2024-01-06 10:54:03

前言

NextCloud 是一款用于创建网络硬盘的开源客户端和服务器软件。每个人都可以借助该软件自行搭建私人的网盘服务。
NextCloud 的服务端采用 PHP 编写，底层存储默认保存在服务器的本地硬盘中。通过修改 NextCloud 配置，可以使用腾讯云对象存储（Cloud Object Storage，COS）作为底层存储，享受对象存储带来的更低廉的存储成本、更高的可靠性和容灾能力，以及无限的存储空间。
本文将介绍 NextCloud 服务端所依赖的环境，并分析对比本地存储与对象存储的区别，最后将讲解实战搭建个人网盘。

注意

将现有 NextCloud 服务端实例从本地存储更改为使用腾讯云对象存储可能导致已有的文件不可见。如需修改已有实例的存储方式，建议依照本教程搭建全新的 NextCloud 服务端并配置使用腾讯云对象存储，随后将旧实例的数据迁移至新实例。

NextCloud 服务端环境简介

NextCloud 服务端采用 PHP 编写，数据库可以使用 SQLite、MySQL、MariaDB 或 PostgreSQL，其中 SQLite 由于性能上的限制，通常不建议在实际应用中使用。虽然 PHP、MySQL 及相关的服务器软件都有 Windows 版本，但是根据 NextCloud 社区的反馈，在 Windows 下运行 NextCloud 服务端会存在文字编码等问题，因此官方宣称不支持在 Windows 下部署 NextCloud 服务端。

服务器配置

腾讯云云服务器（Cloud Virtual Machine，CVM）目前有多个实例族，每个实例族中又分为多个子类型。不同的实例族有不同的侧重点，例如大内存或高 IO 等。NextCloud 定位于个人、家庭或中小企业用户，对各项硬件资源需求都不高，因此选择各项资源均衡的标准型即可满足需求。对于子类型，通常来说最新的子类型将拥有更高的性价比，一般情况下选择最新的子类型即可。
在确定实例族和子类型后，还会面临具体的 vCPU 与内存规格选择。此外，不同的 vCPU 与内存规格还有对应的内网带宽和网络收发包。PHP 可以使用 OPcache 提升性能，而 NextCloud 服务端支持使用 APCu 内存缓存进一步提升性能，因此在规格的选择上，建议选择较大的内存。
由于云服务器在购买后支持配置的调整，我们可以先购买配置比较低的规格，例如1核 vCPU 与4GB内存，在完成搭建并实际上线使用后，根据用户数、文件数以及云服务器的相关监控数据在判断是否需要提高规格提升性能。如果

您预期在家庭或中小企业等多用户场景下使用，那么建议选购2核8GB到4核16GB的配置，以提供足够的性能满足多用户的使用。

服务器操作系统

主流的 Linux 发行版都可以很好的支持 NextCloud 服务端运行，除了不同系统在安装软件包时使用的命令（即包管理工具）有所差别外，其余的配置工作没有区别。

说明

本文以 CentOS 7.7操作系统的云服务器为例进行演示。

数据库

如上文所述，在实际应用中通常使用 MySQL 搭配 PHP 使用，而 MariaDB 是 MySQL 的“复刻”版本，与 MySQL 保持高度的兼容，因此 MySQL 5.7+或 MariaDB 10.2+均可以很好的配合 NextCloud 服务端使用。

腾讯云提供托管的云数据库 MySQL 和云数据库 MariaDB，相对于在云服务器上自建数据库，云数据库默认采用一主一备的高可用模式，具有更高的可靠性，且提供自动备份等方便的运维操作，因此强烈建议在实际应用中使用云数据库。

说明

本文以云数据库 MySQL 5.7版本为例进行演示。

Web 服务器及 PHP 运行时

NextCloud 服务端通过 `.htaccess` 指定了部分配置，因此使用 Apache 服务器软件时可直接使用 NextCloud 服务端自带的配置项。Nginx 是近些年发展较快的 Web 服务器软件，相对 Apache 具有安装配置简单、资源占用少、负载能力更强的优点，通过将 NextCloud 服务端中的 `.htaccess` 配置转写为 Nginx 的配置，亦可很好的支持 NextCloud 服务端的运行，本文将使用 Nginx 服务器软件，并提供完整的 Nginx 配置示例可供参考。

PHP 运行时目前已经发展到 PHP 7，主要维护的版本包括7.2、7.3和7.4，这3个版本均支持 NextCloud 服务端，我们使用最新的7.4即可。此外，NextCloud 还依赖 PHP 的部分扩展模块，下文将详细介绍具体的扩展模块要求。

腾讯云网络环境

腾讯云目前提供基础网络和私有网络（VPC）环境。基础网络是腾讯云上所有用户的公共网络资源池，所有云服务器的内网 IP 地址都由腾讯云统一分配，无法自定义网段划分、IP 地址。私有网络是用户在腾讯云上建立的一块逻辑隔离的网络空间，在私有网络内，用户可以自由定义网段划分、IP 地址和路由策略。目前基础网络由于资源紧缺且无法扩增等功能，新注册账号及部分新建可用区均不再支持基础网络，因此本文将以私有网络为例进行后续的演示。

说明

有关私有网络的进一步介绍，请参见 私有网络产品概述。

云硬盘与对象存储的对比

在云服务器中，云硬盘（Cloud Block Storage，CBS）将以云服务器中的本地硬盘的形式挂载在操作系统中，NextCloud 默认使用文件系统存储网盘数据，因此可以直接将 NextCloud 的数据存储在操作系统中的云硬盘。与云硬盘相比，使用对象存储的优势从如下几个维度进行讲解。

应用场景

云硬盘

云硬盘属于块存储，可直接挂载到云服务器操作系统中作为硬盘使用，通常情况被操作系统独占，即只能挂载在一台云服务器中，但其拥有较高的读写性能，适用于高 IO 低延时且不需要与其他云服务器共享的场景。

对象存储

对象存储以 HTTP 协议对外提供读写接口，需要通过编程的方式访问对象存储的存储对象（文件）。对象存储使用对象键（Key，可以理解为文件路径）作为索引，无存储容量的限制。由于使用网络传输，在速度和延时上相对较大，但因为操作是对象级别，因此一个软件完成一个对象的操作后，另一个软件即可马上操作同一对象，适用于对性能要求不高、需要低成本大容量存储或有共享访问需求的场景。由于网盘应用本身通过网络传输，对延时的要求不高，且从网盘客户端到网盘服务端再到对象存储的链路中，影响速度与时延的因素主要在于客户端所处的网络环境，而对象存储本身不限速，因此对象存储更适合搭配网盘应用。

维护

云硬盘

云硬盘为固定容量，可通过控制台或云 API 扩容，扩容后还需要在操作系统中扩展分区，且在扩展分区时有一定的分区异常风险，有一定的维护成本。

对象存储

对象存储按需使用，不限制总容量，也不限制对象数（文件数），完全无需维护。

数据安全

云硬盘和对象存储均使用多副本等手段保证数据的可靠性。

搭建 NextCloud 服务端运行环境

准备 NextCloud 服务端依赖的云产品

云服务器

入门操作请参见 云服务器 CVM 快速入门。

云数据库 MySQL

入门操作请参见 云数据库 MySQL 快速入门。

对象存储

1. 打开并登录 对象存储控制台（首次使用需先开通对象存储服务），进入**存储桶列表**，单击**创建存储桶**，根据下表说明进行配置：

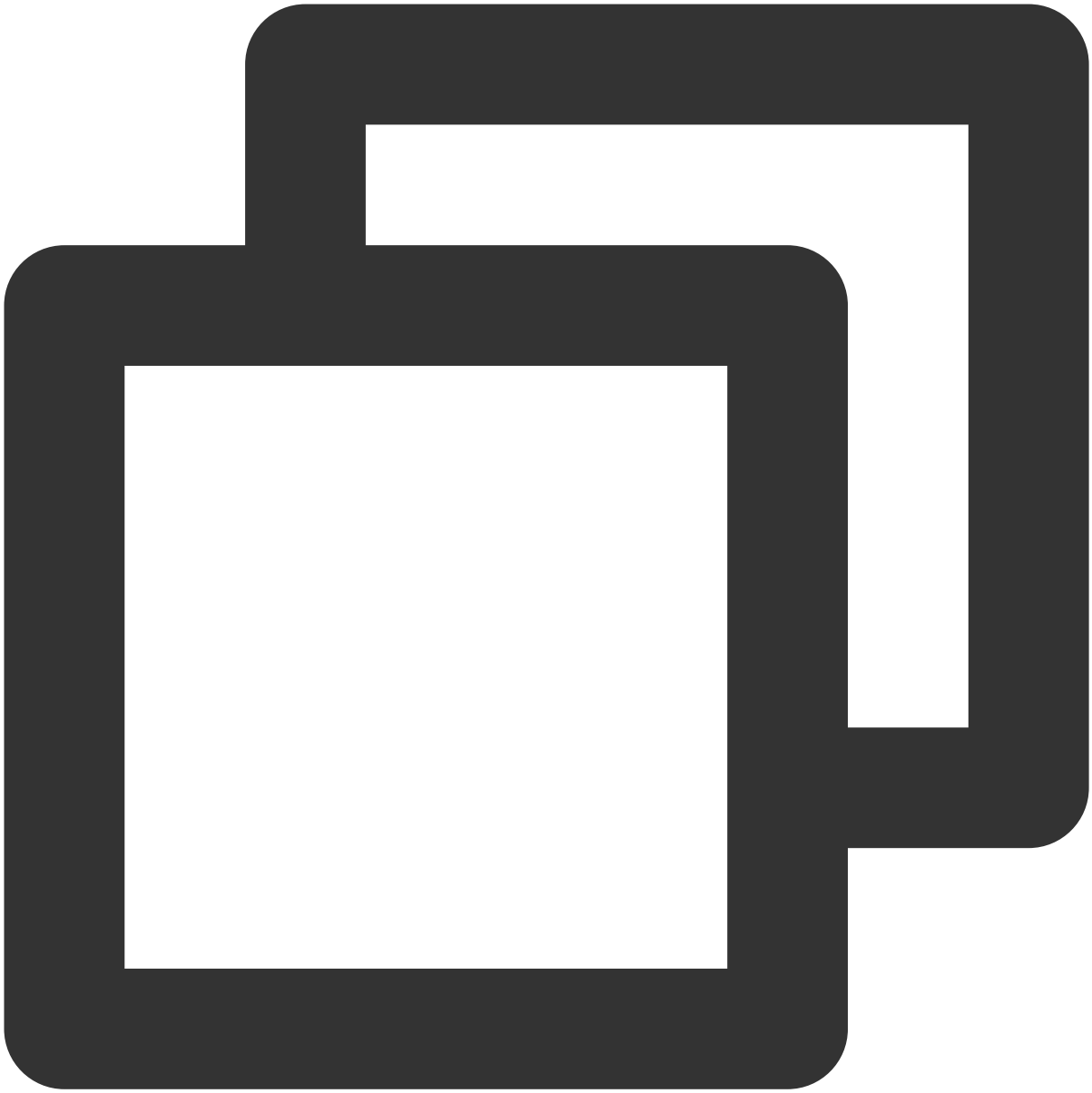
配置项	值
名称	输入一个自定义的存储桶名称，例如 nextcloud。注意，该名称确定后将不允许更改
所属地域	与所购 CVM 所属地域保持一致
其他	保持默认

2. 完成以上配置后，单击**确定**完成创建。

安装、配置 Web 服务器和 PHP 进行时

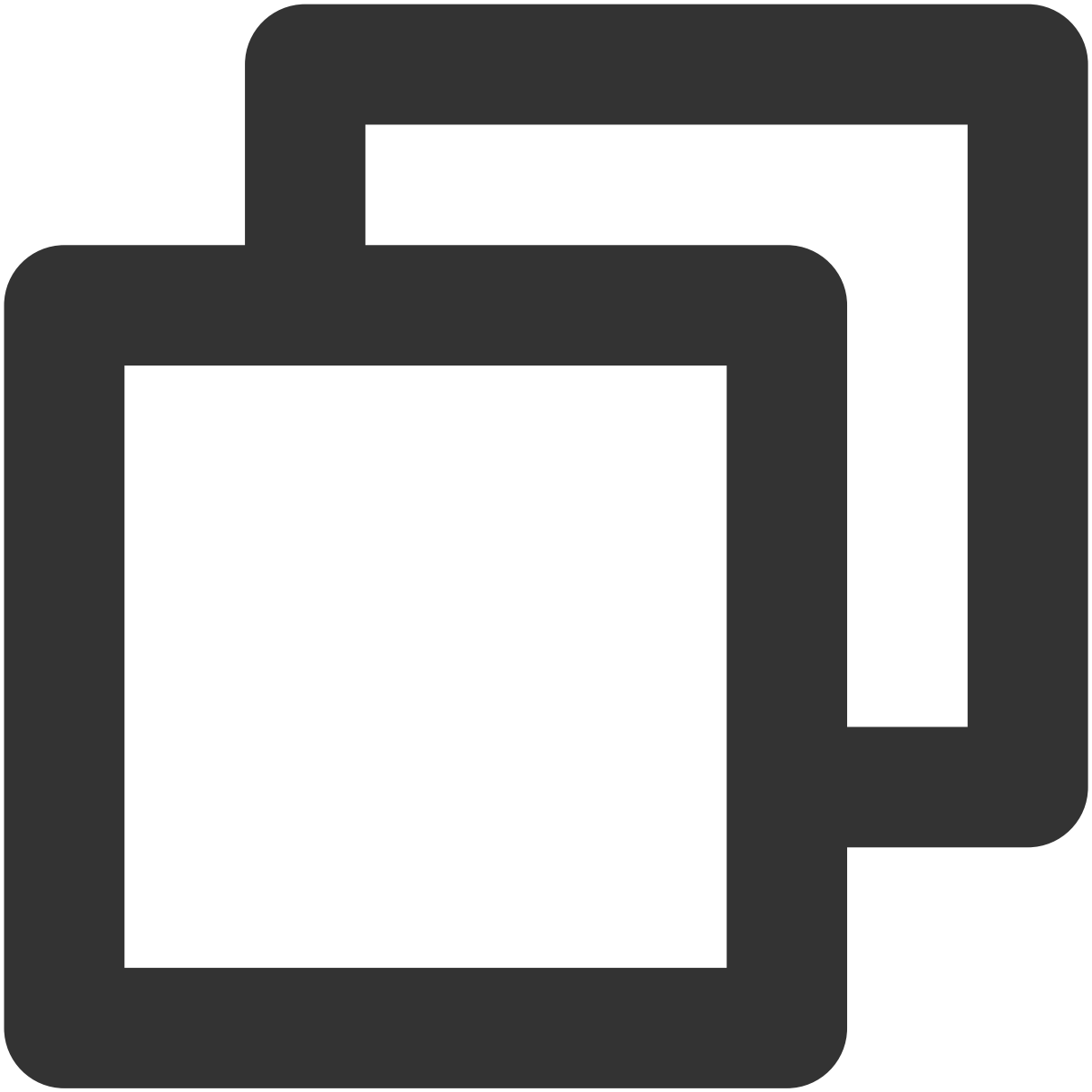
安装 Nginx

1. 使用 SSH 工具登录到新购服务器。
2. 执行下述命令安装 Nginx：



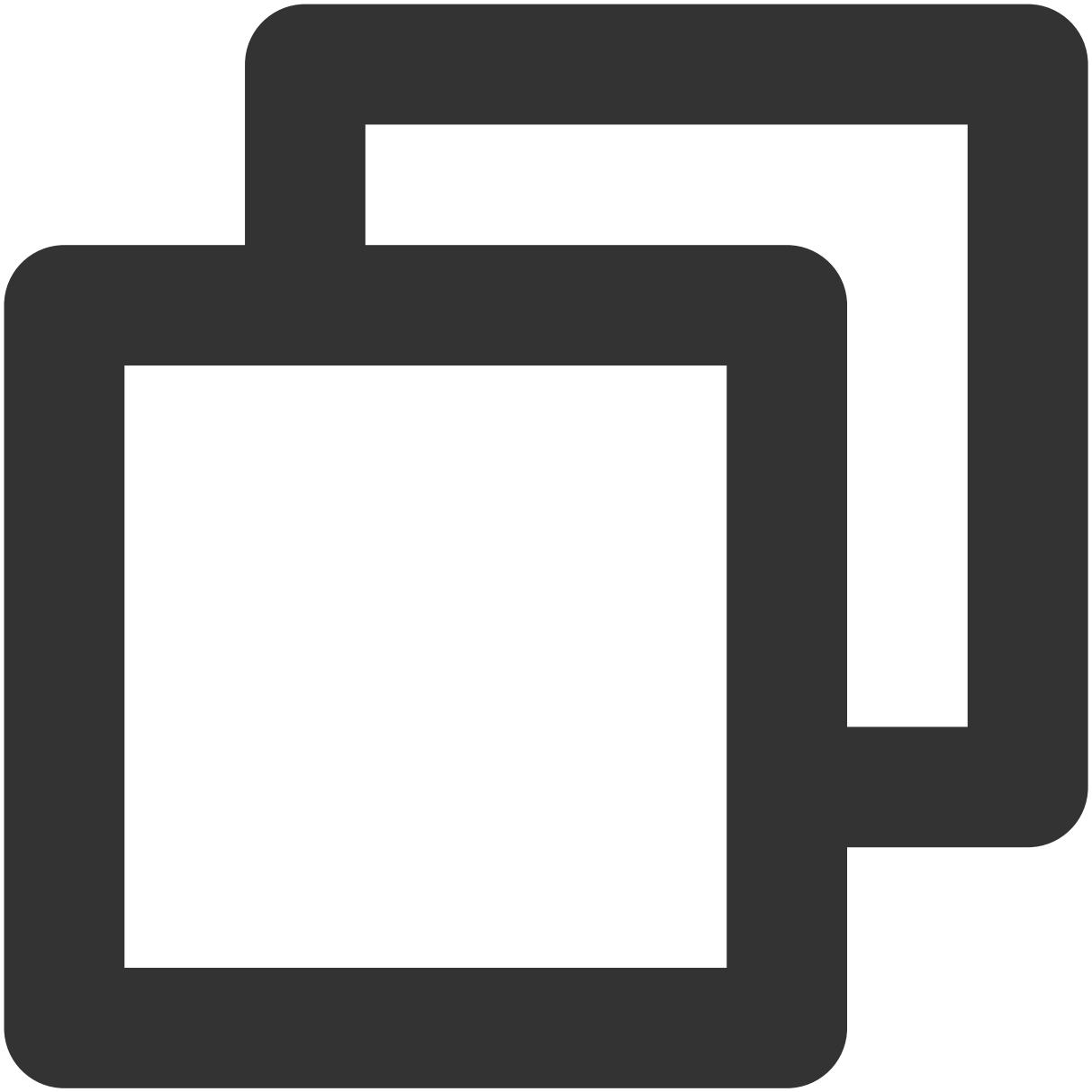
```
yum install nginx
```

当提示以下信息时，按 `Y` 并回车确认安装（下同）。



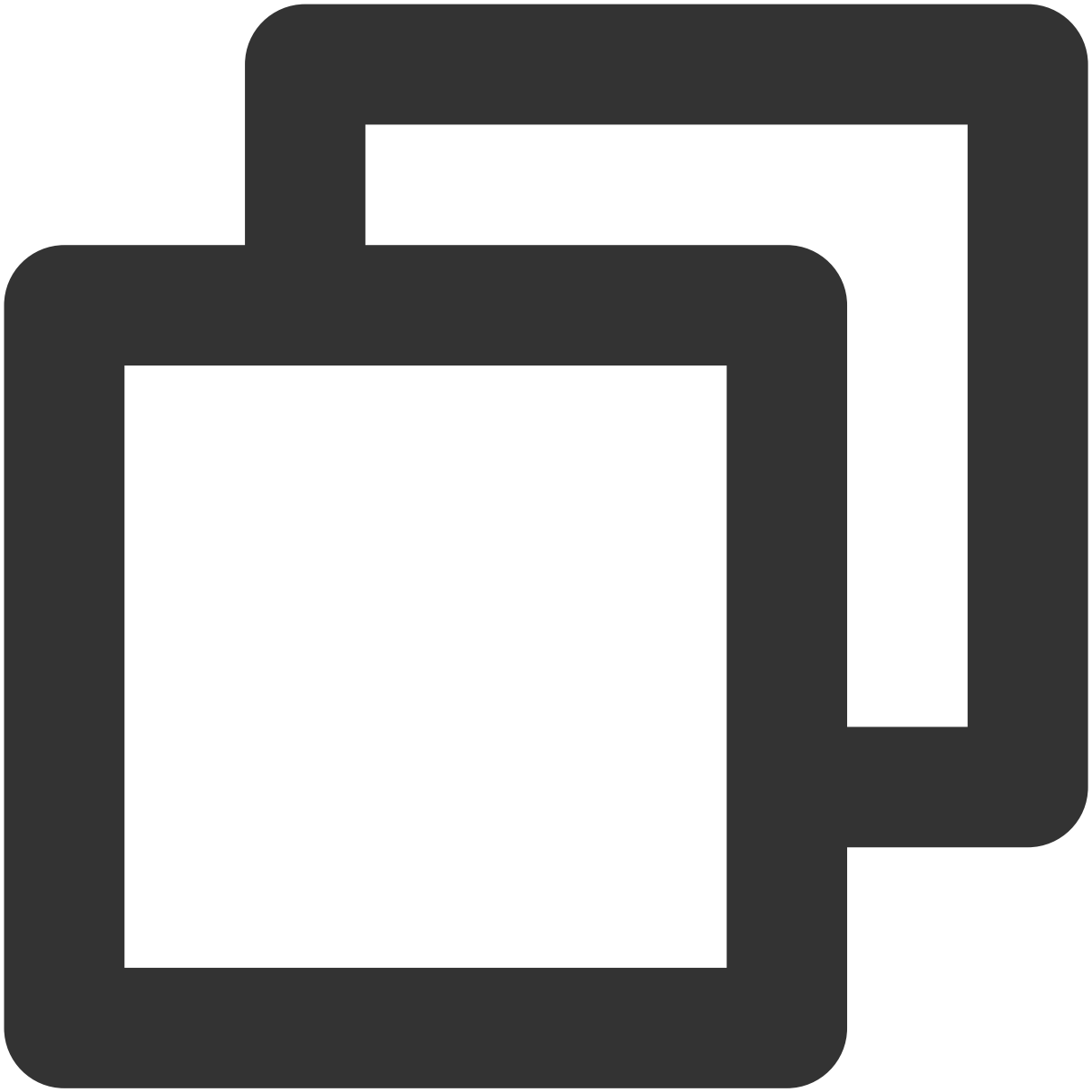
```
Is this ok [y/d/N]:
```

3. 当出现以下信息时，表示安装完成。



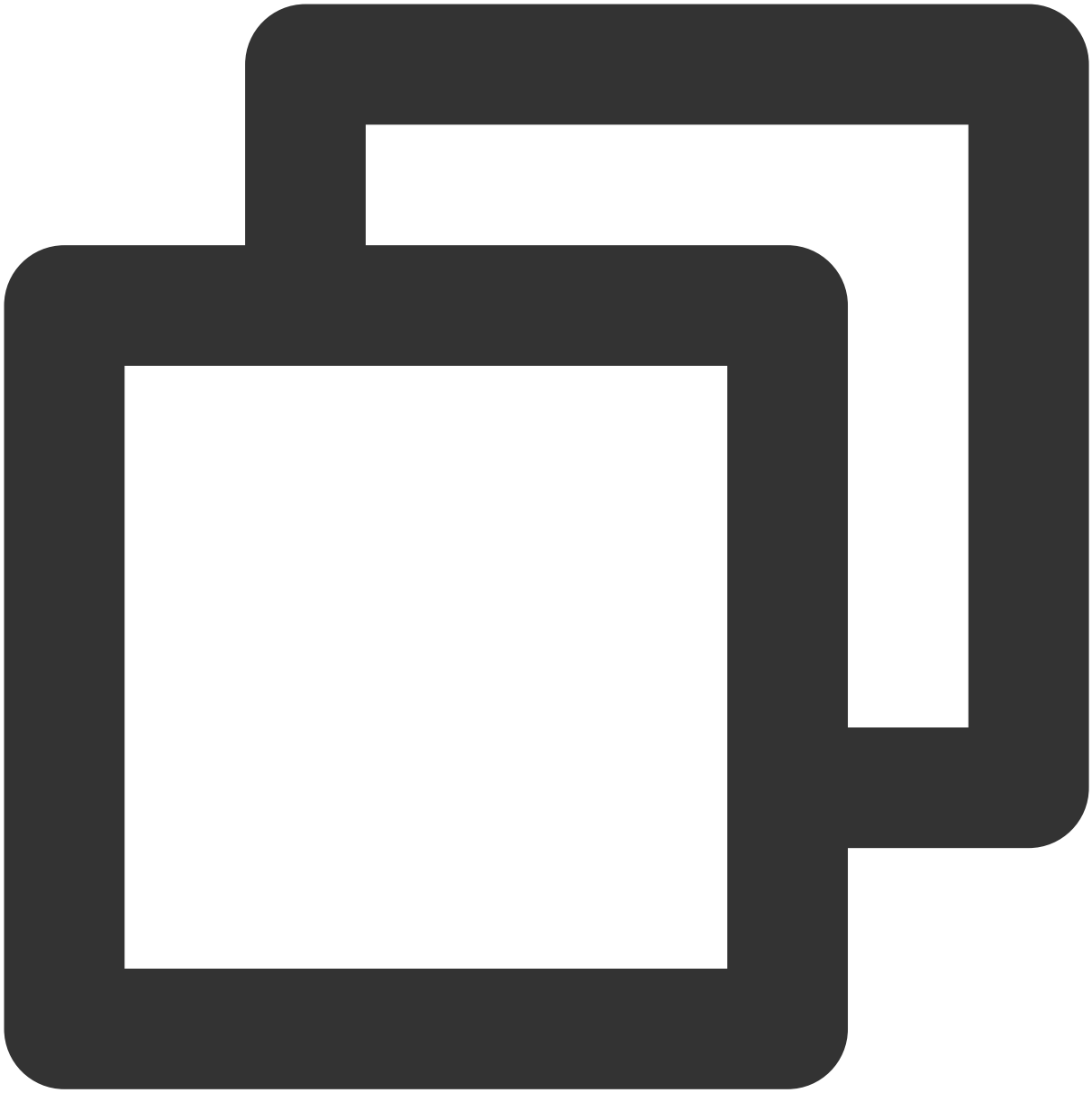
```
Complete!
[root@VM-0-10-centos ~]#
```

4. 接下来执行以下命令，验证是否可以正常查看 Nginx 版本。



```
nginx -v
```

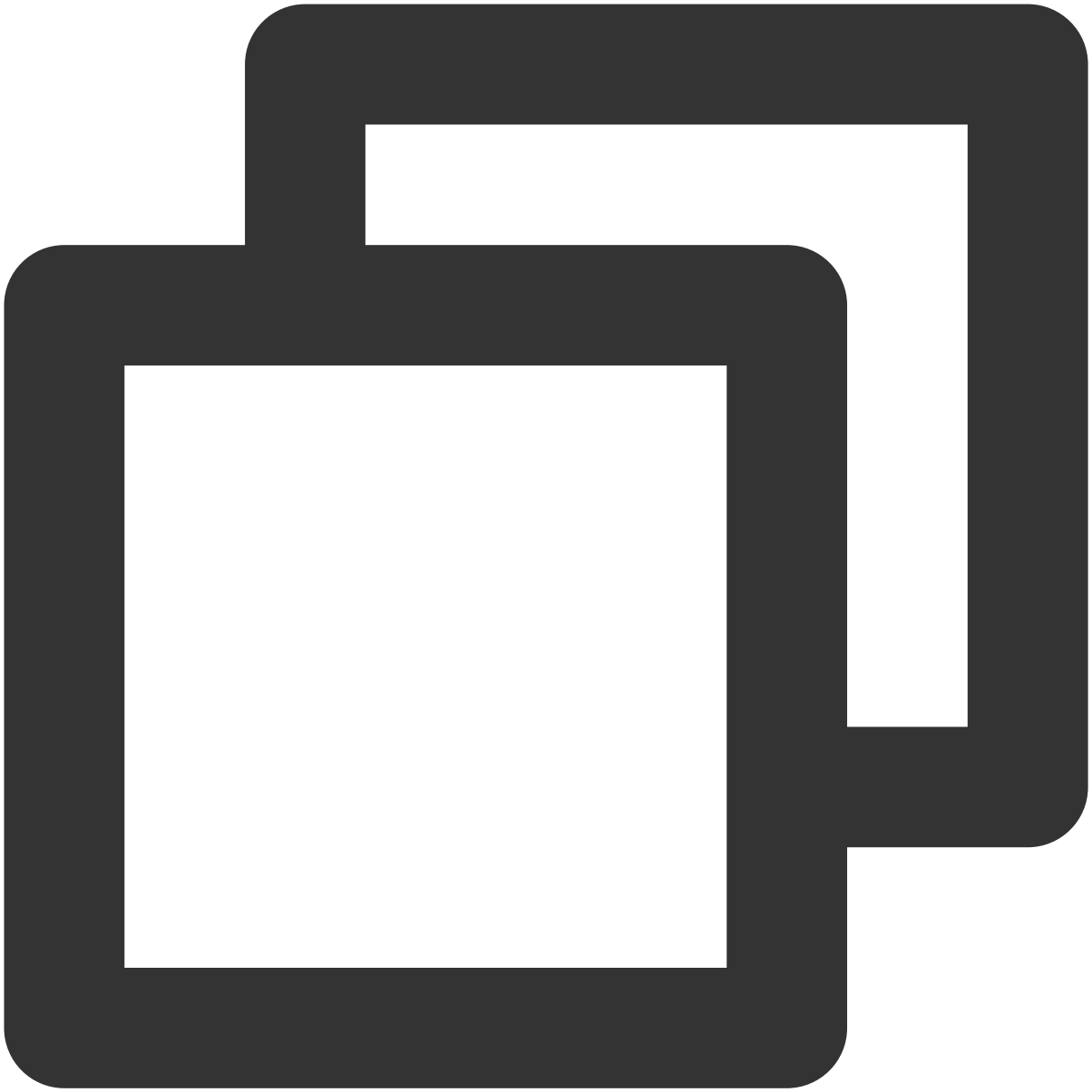
假如出现以下信息，则验证安装完成。



```
nginx version: nginx/1.16.1
```

安装 PHP

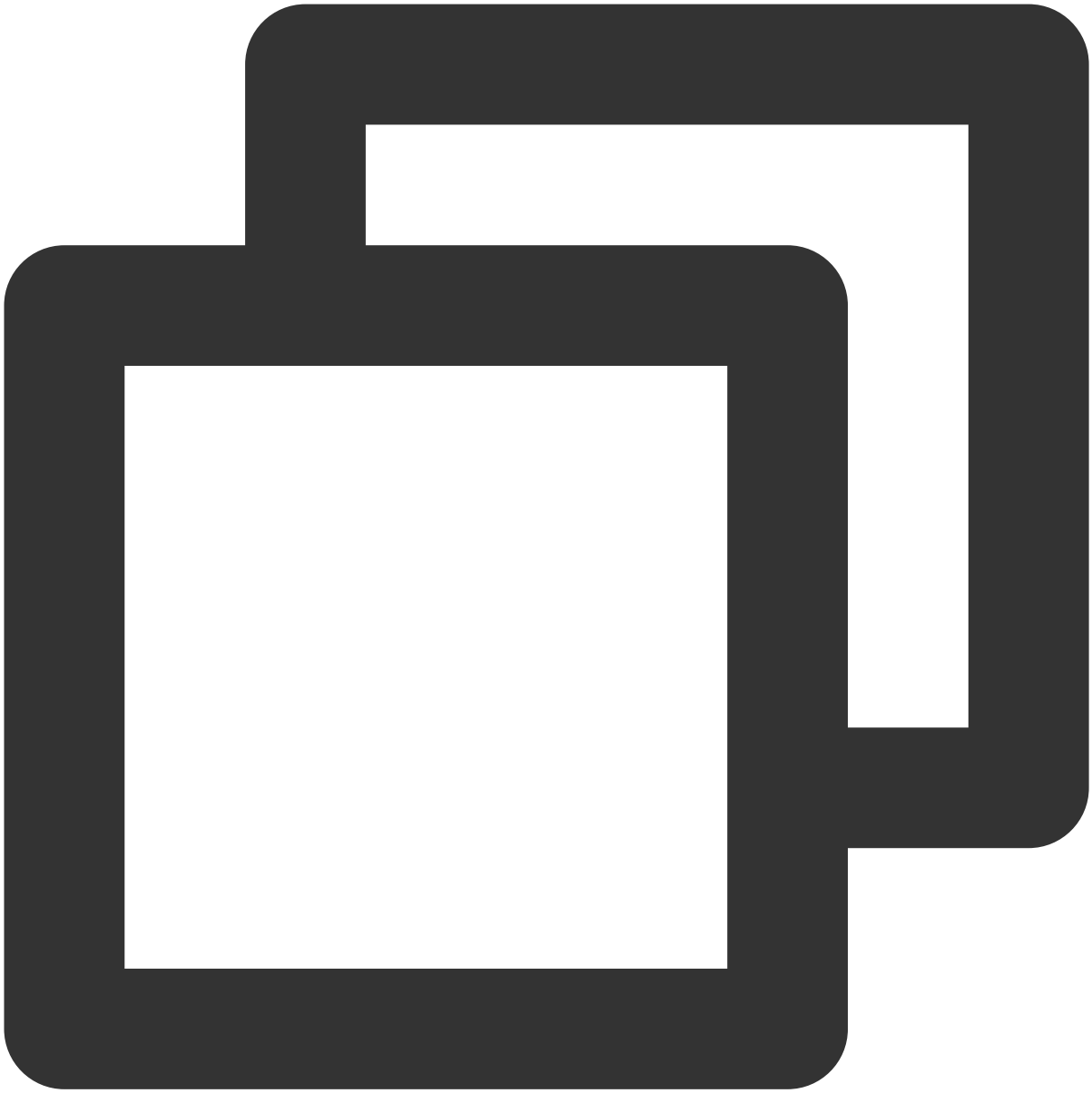
1. 使用 SSH 工具登录到新购服务器。
2. 执行下述命令开始安装 PHP 7.4：



```
yum install epel-release yum-utils
```

3. 依次执行下述命令：

命令1：

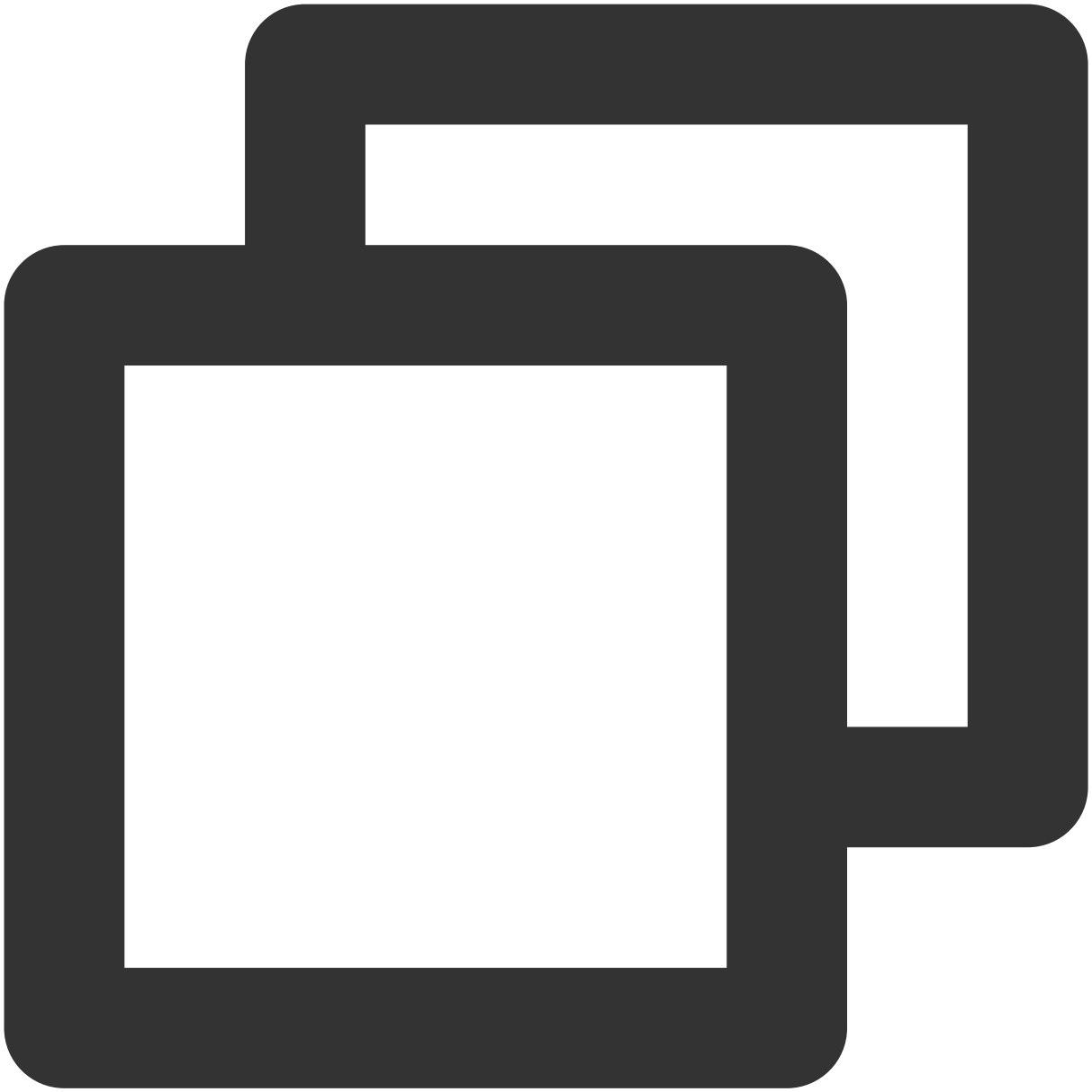


```
yum install http://rpms.remirepo.net/enterprise/remi-release-7.rpm
```

说明

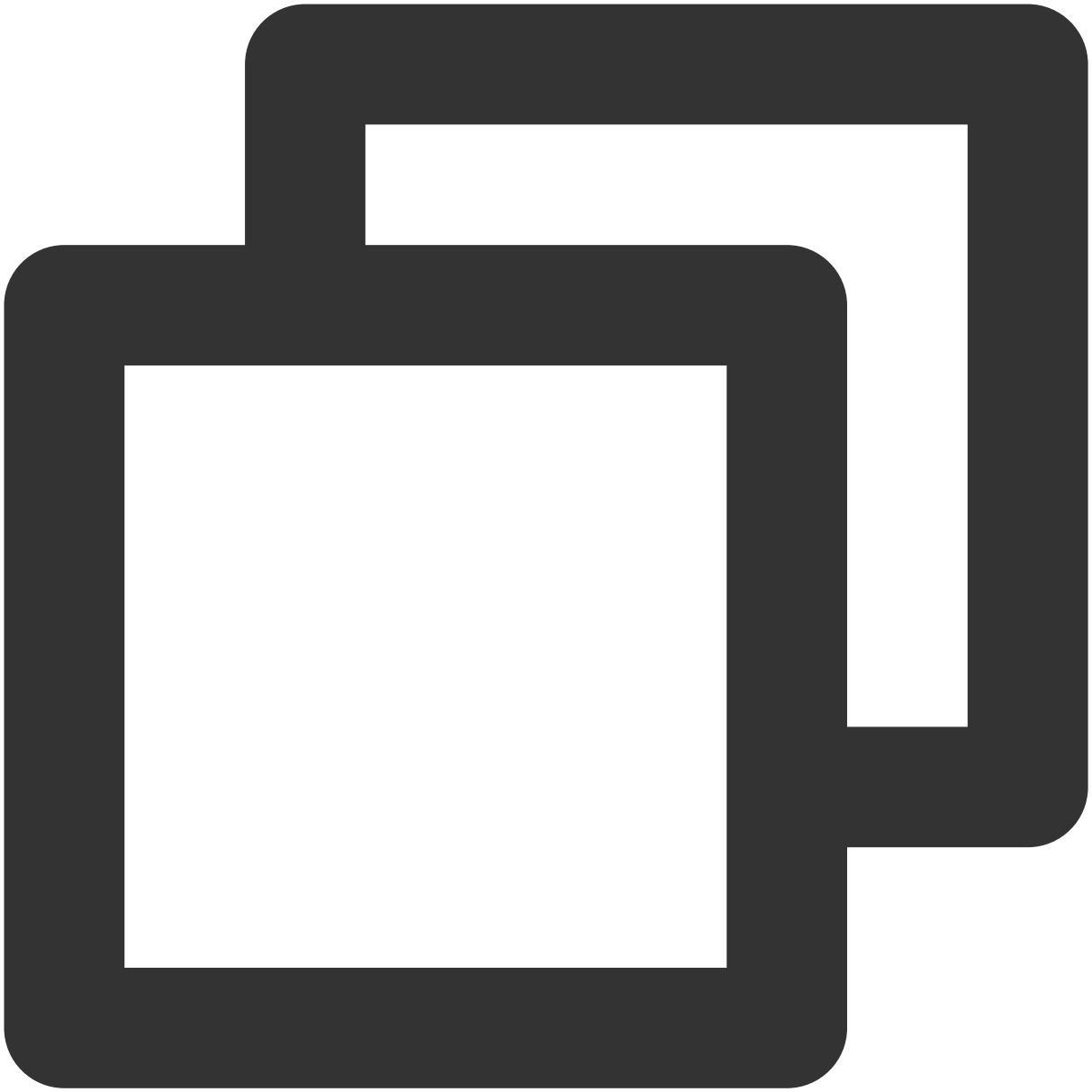
执行上述命令时如遇速度过慢、进度长时间不动，可以按 `Ctrl-C` 取消并重新执行该条命令（下同）。

命令2：



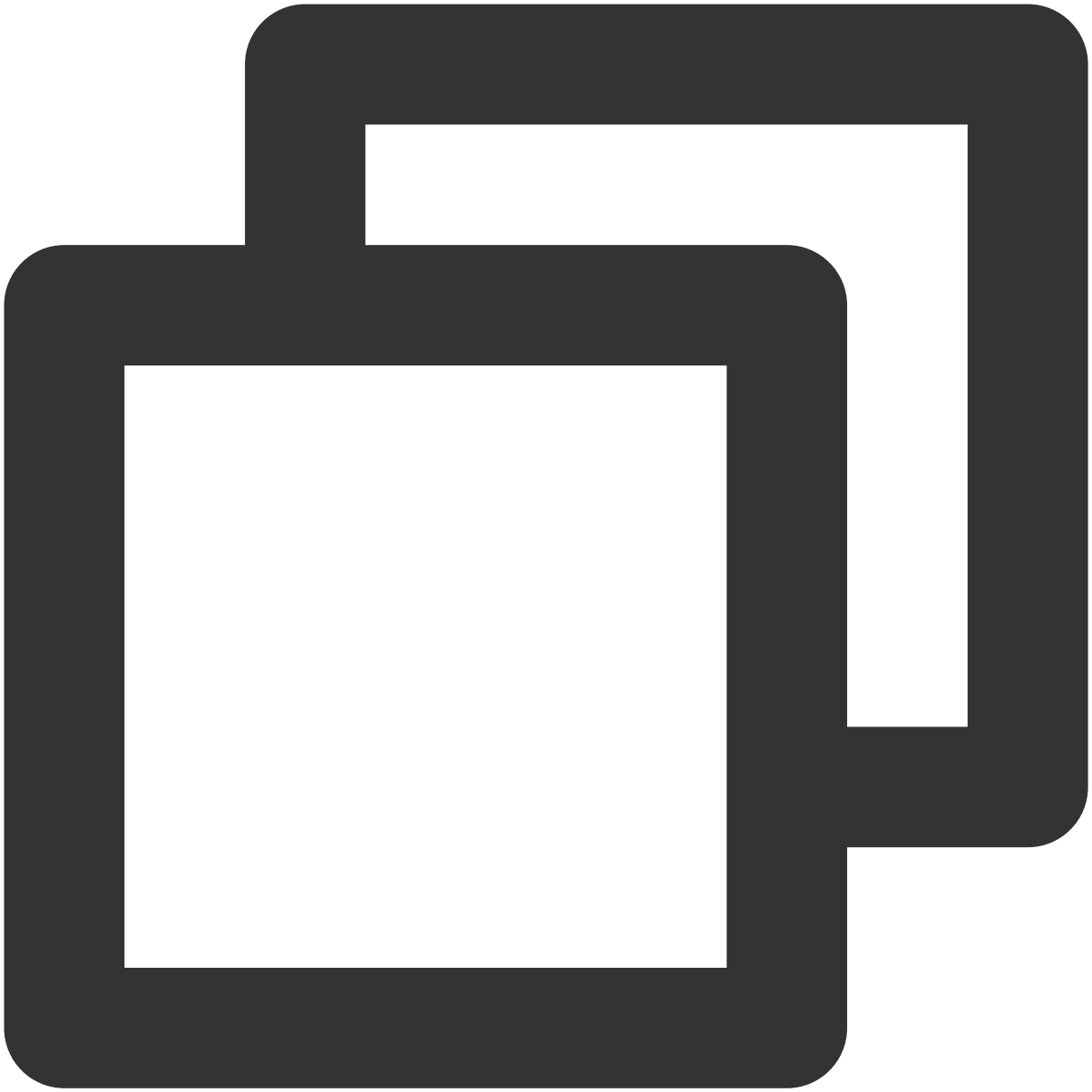
```
yum-config-manager --enable remi-php74
```

命令3：



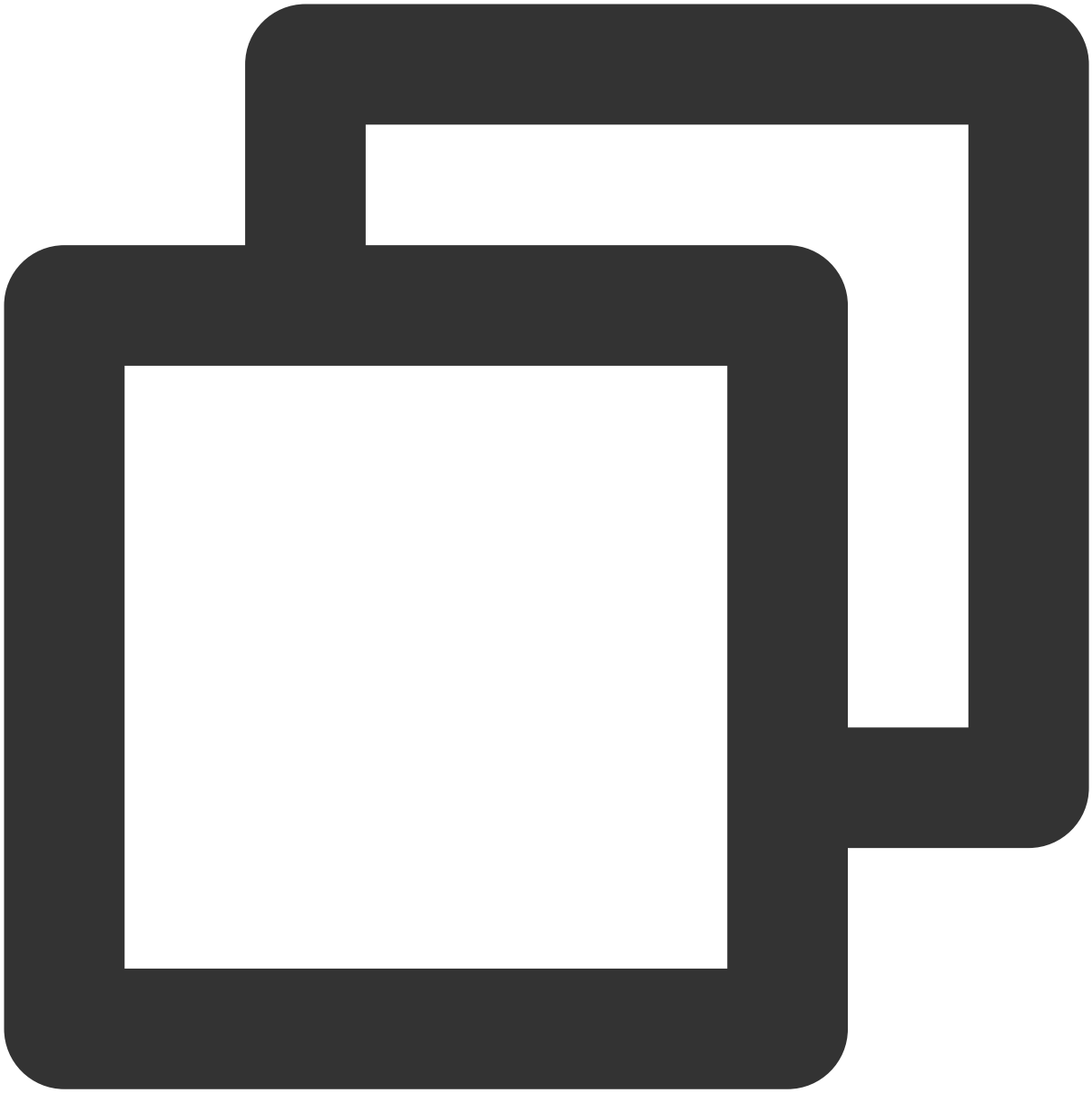
```
yum install php php-fpm
```

4. 安装完成后，执行以下命令，验证是否可以正常查看 PHP 版本。



```
php -v
```

假如出现以下信息，则验证安装完成。



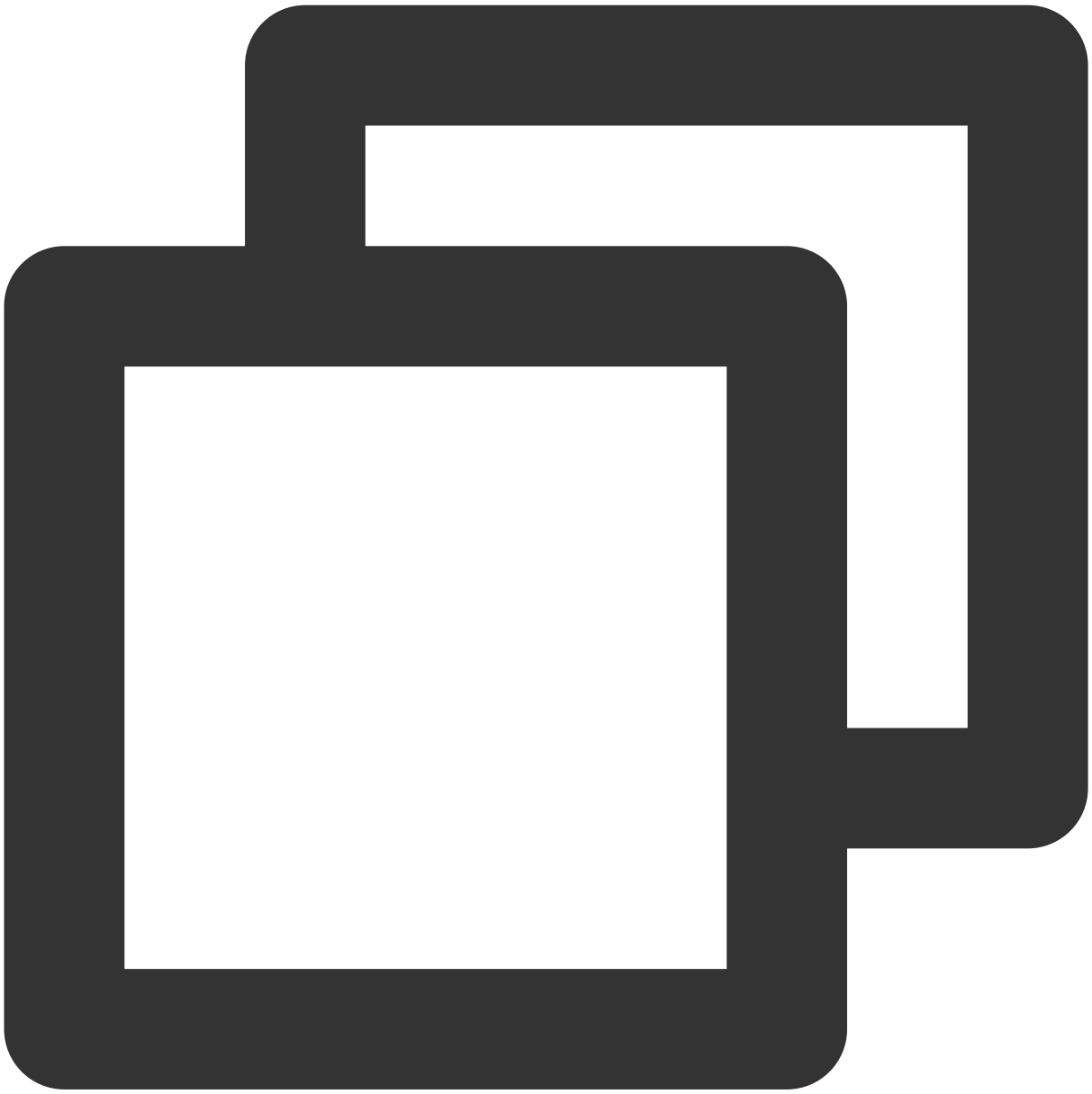
```
PHP 7.4.8 (cli) (built: Jul 9 2020 08:57:23) ( NTS )
Copyright (c) The PHP Group
Zend Engine v3.4.0, Copyright (c) Zend Technologies
```

安装 PHP 模块

除了基本的 PHP 外，NextCloud 还依赖其他 PHP 模块来实现部分功能。有关 NextCloud 依赖的详细模块信息，可参阅 NextCloud 官方文档。

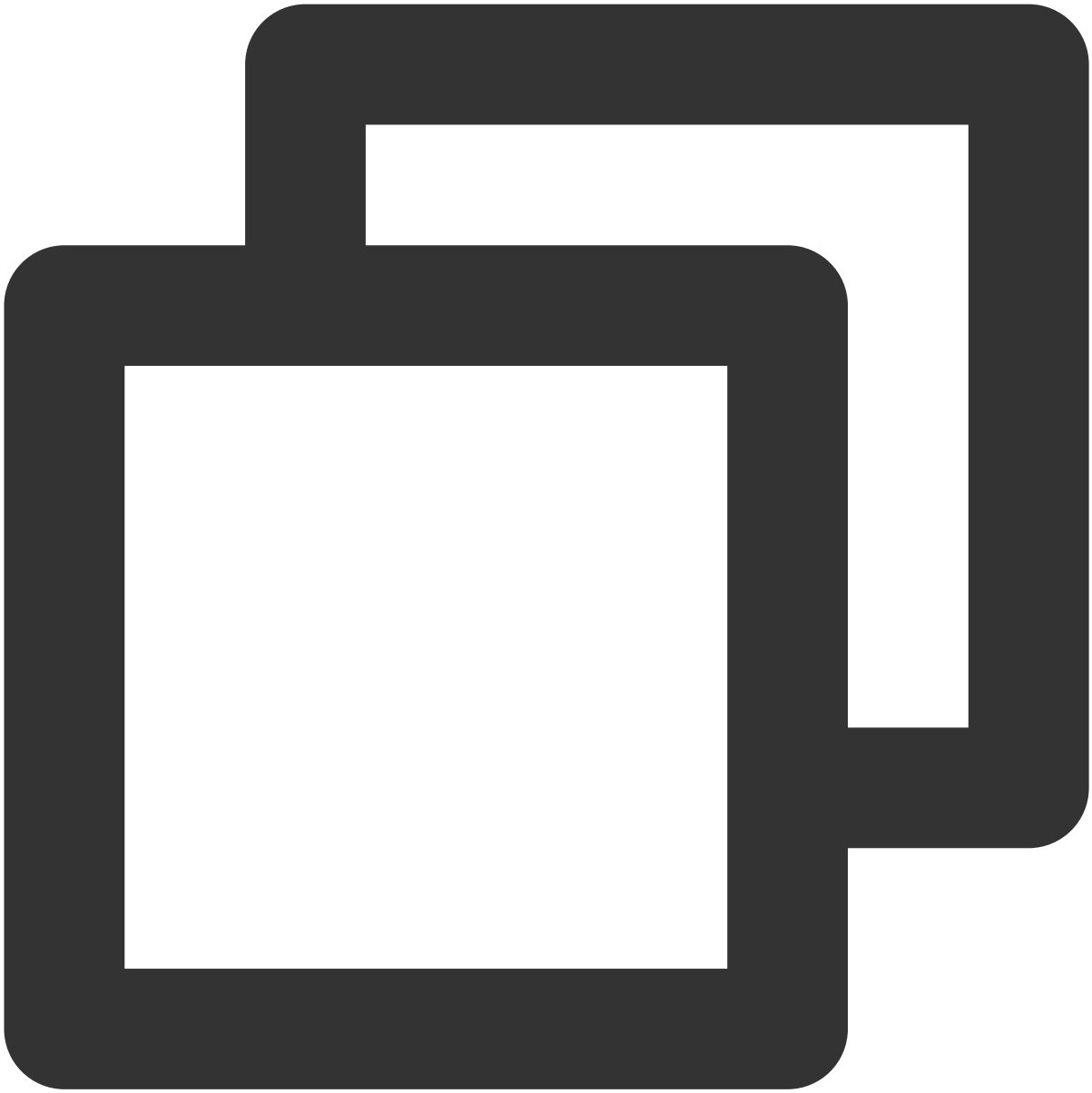
本教程中将安装 NextCloud 必选的 PHP 模块，如果您计划后续使用 NextCloud 的其他可选功能，请留意并自行安装所依赖的其他 PHP 模块。

1. 使用 SSH 工具登录到新购服务器。
2. 执行下述命令安装 PHP 模块：



```
yum install php-xml php-gd php-mbstring php-mysqlnd php-intl php-zip
```

3. 安装完成后，执行以下命令查看已安装的 PHP 模块。



```
php -m
```

4. 如果还需要安装其他模块，重复执行 `yum install <php-module-name>` 即可。

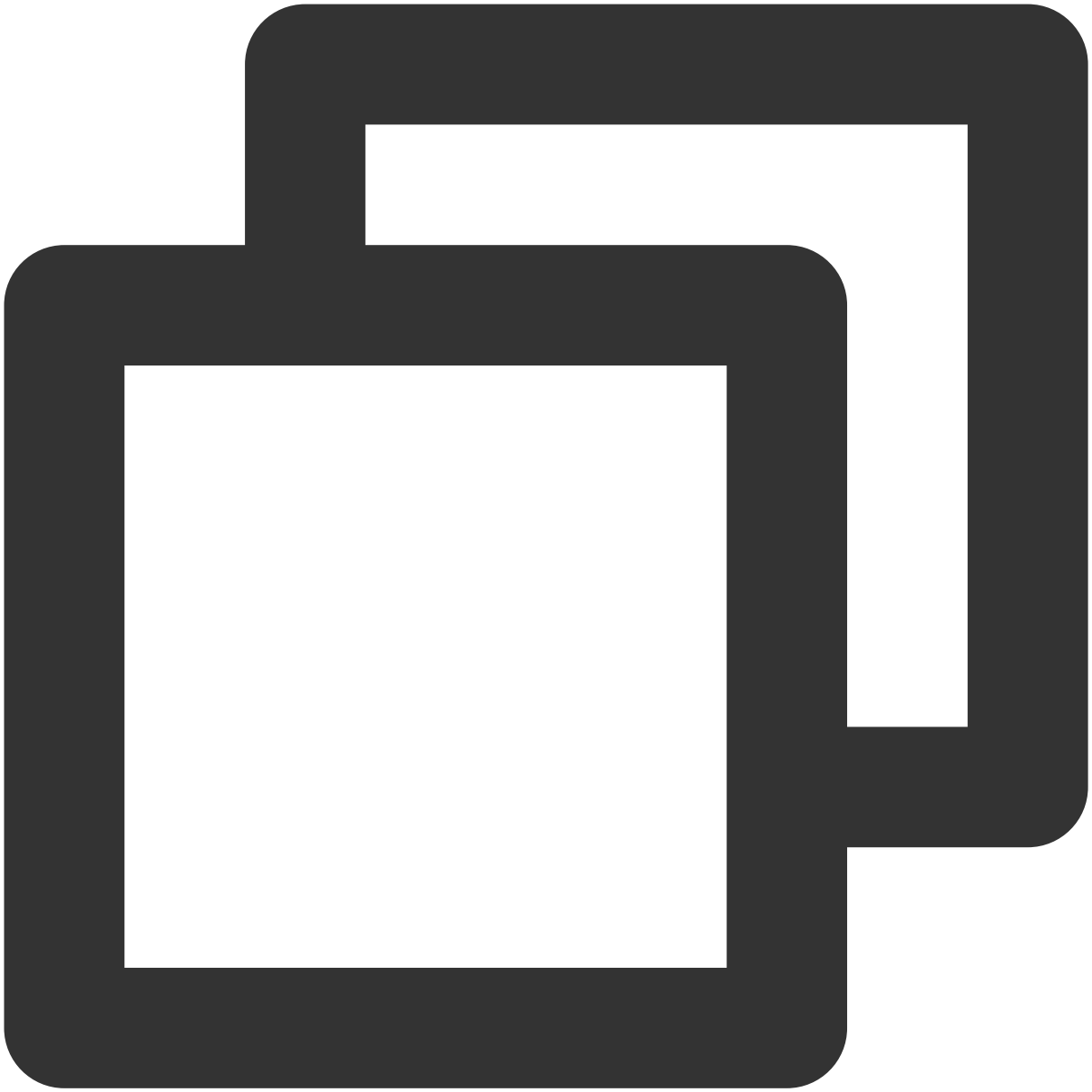
上传并解压 NextCloud 服务端代码

1. 在 NextCloud 官网 下载 NextCloud 服务端最新版安装包，并上传至服务器 `/var/www/` 目录下，您可以通过以下方法上传：
1. 使用 `wget` 命令直接在服务器上下载安装包，例如：进入 `/var/www/` 目录后，执行命令 `wget https://download.nextcloud.com/server/releases/nextcloud-19.0.1.zip` 。

2. 下载到本地计算机上，然后通过 SFTP 或 SCP 等软件将安装包上传至 `/var/www/` 目录。
3. 下载到本地计算机上，使用 lrzsz 上传，方法是：
3.1 使用 SSH 工具登录到新购服务器。
3.2 执行 `yum install lrzsz` 安装 lrzsz。
3.3 执行 `cd /var/www/` 进入目标目录。
3.4 执行 `rz -bye`，随后在 SSH 工具中选择下载到本地的 NextCloud 服务端安装包（依据 SSH 工具的不同，此处操作将不尽相同）。
4. 使用 SSH 工具登录到新购服务器。
5. 执行 `unzip nextcloud-<version>.zip` 解压安装包，例如 `unzip nextcloud-19.0.1.zip`。

配置 PHP

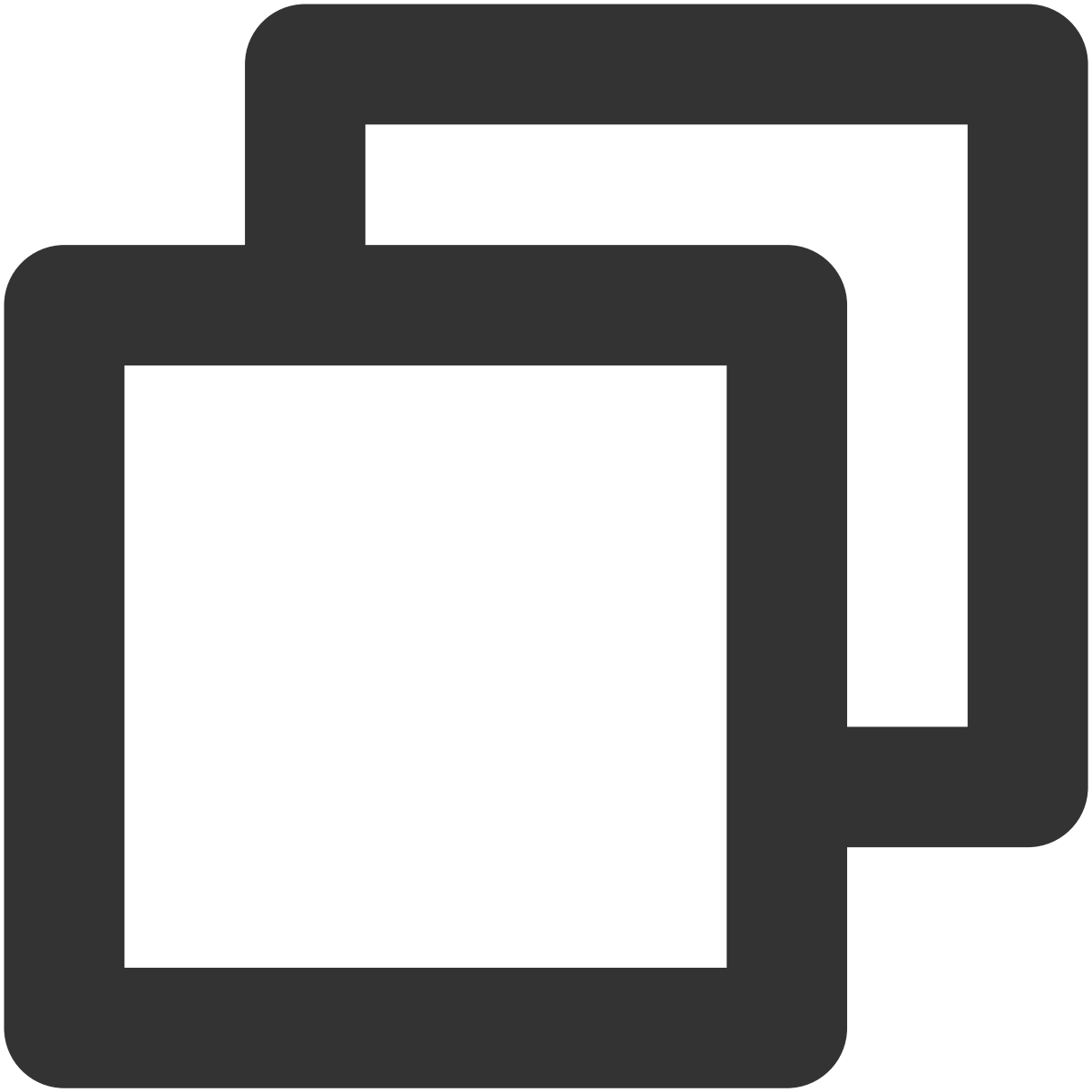
1. 使用 SSH 工具登录到新购服务器。
2. 执行 `vim /etc/php-fpm.d/www.conf` 打开 PHP-FPM 的配置文件，并依次修改配置项（关于 vim 的具体使用请参阅相关资料，您也可以使用其他方式修改该配置文件）。
1. 将 `user = apache` 修改为 `user = nginx`。
2. 将 `group = apache` 修改为 `group = nginx`。
3. 修改完成后，输入 `:wq` 保存文件并退出（有关 vim 的详细操作指引，请参阅相关文档）。
4. 执行下述命令修改目录所有者，使 PHP 能够适配 Nginx 使用：



```
chown -R nginx:nginx /var/lib/php
```

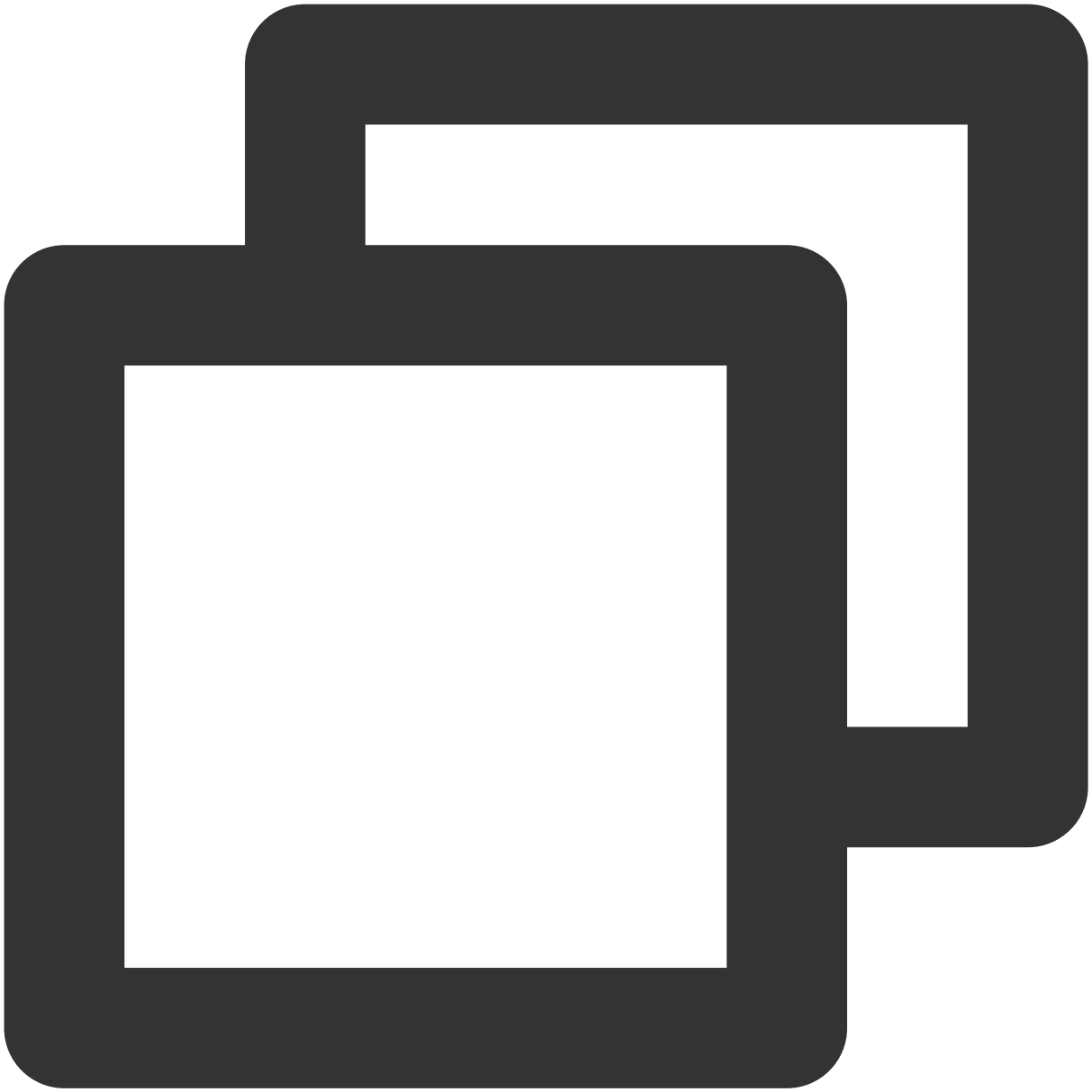
5. 依次执行下述命令，并启动 PHP-FPM 服务：

命令1：



```
systemctl enable php-fpm    # 命令1
```

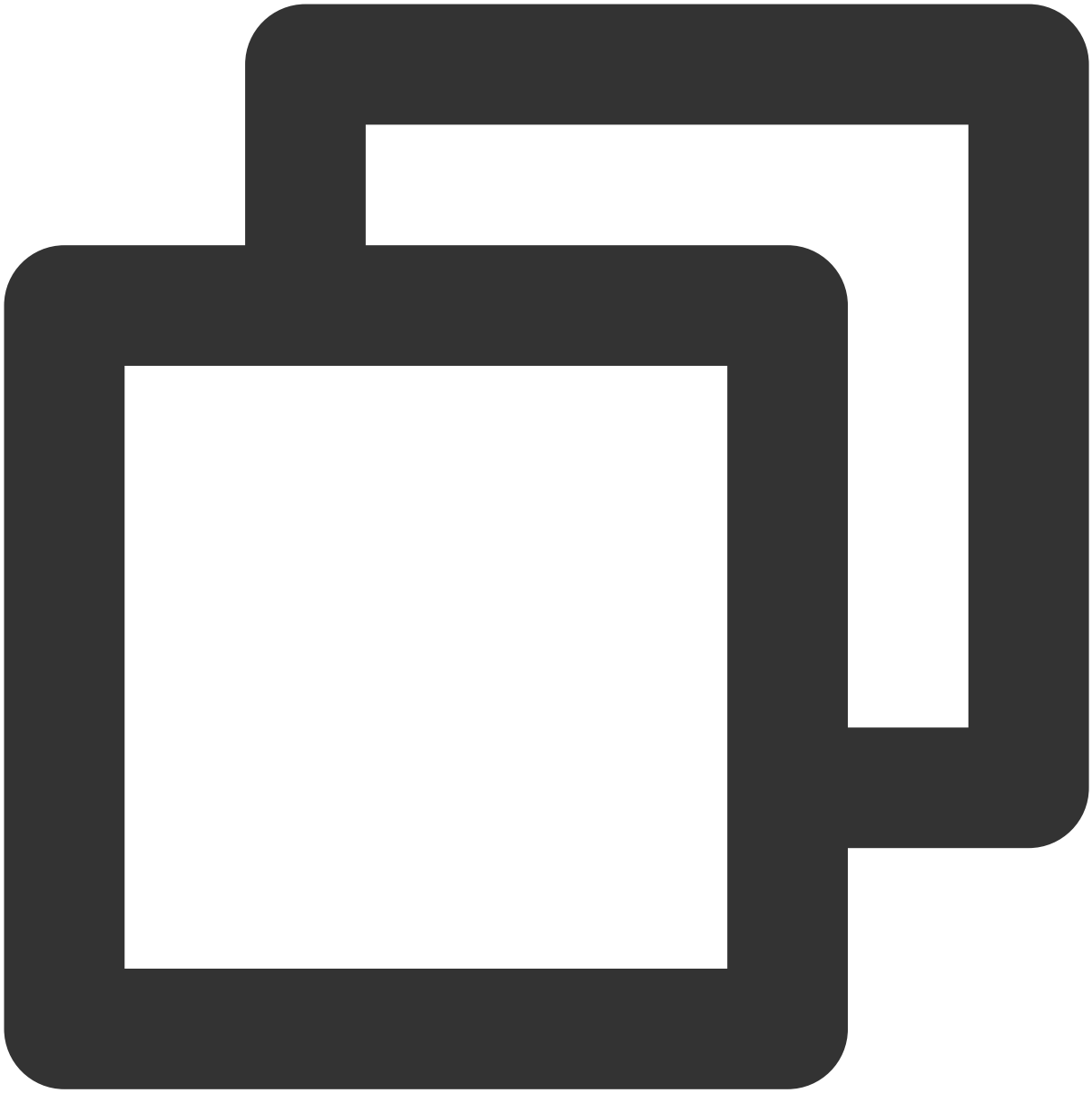
命令2：



```
systemctl start php-fpm   # 命令2
```

配置 Nginx

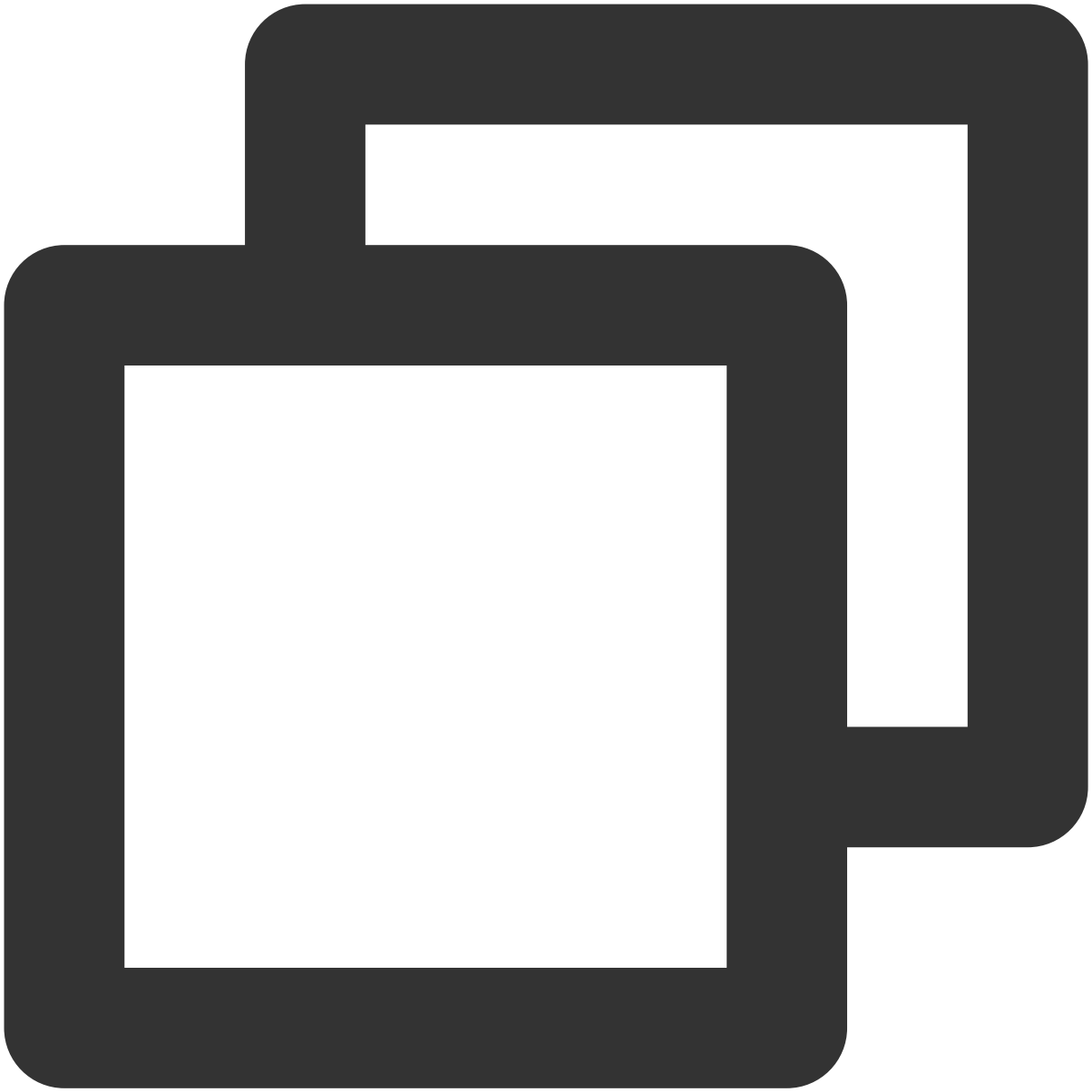
1. 使用 SSH 工具登录到新购服务器。
2. 执行下述命令修改网站目录所有者：



```
chown -R nginx:nginx /var/www
```

3. 备份当前的 Nginx 配置文件 `/etc/nginx/nginx.conf`，您可以：

1. 执行 `cp /etc/nginx/nginx.conf ~/nginx.conf.bak` 将当前配置文件备份到家（HOME）目录。
2. 使用 SFTP 或 SCP 等软件将当前配置文件下载到本地计算机。
3. 将 `/etc/nginx/nginx.conf` 修改或替换为如下内容：



```
# For more information on configuration, see:
#   * Official English Documentation: http://nginx.org/en/docs/
#   * Official Russian Documentation: http://nginx.org/ru/docs/

user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log;
pid /run/nginx.pid;

# Load dynamic modules. See /usr/share/doc/nginx/README.dynamic.
include /usr/share/nginx/modules/*.conf;
```

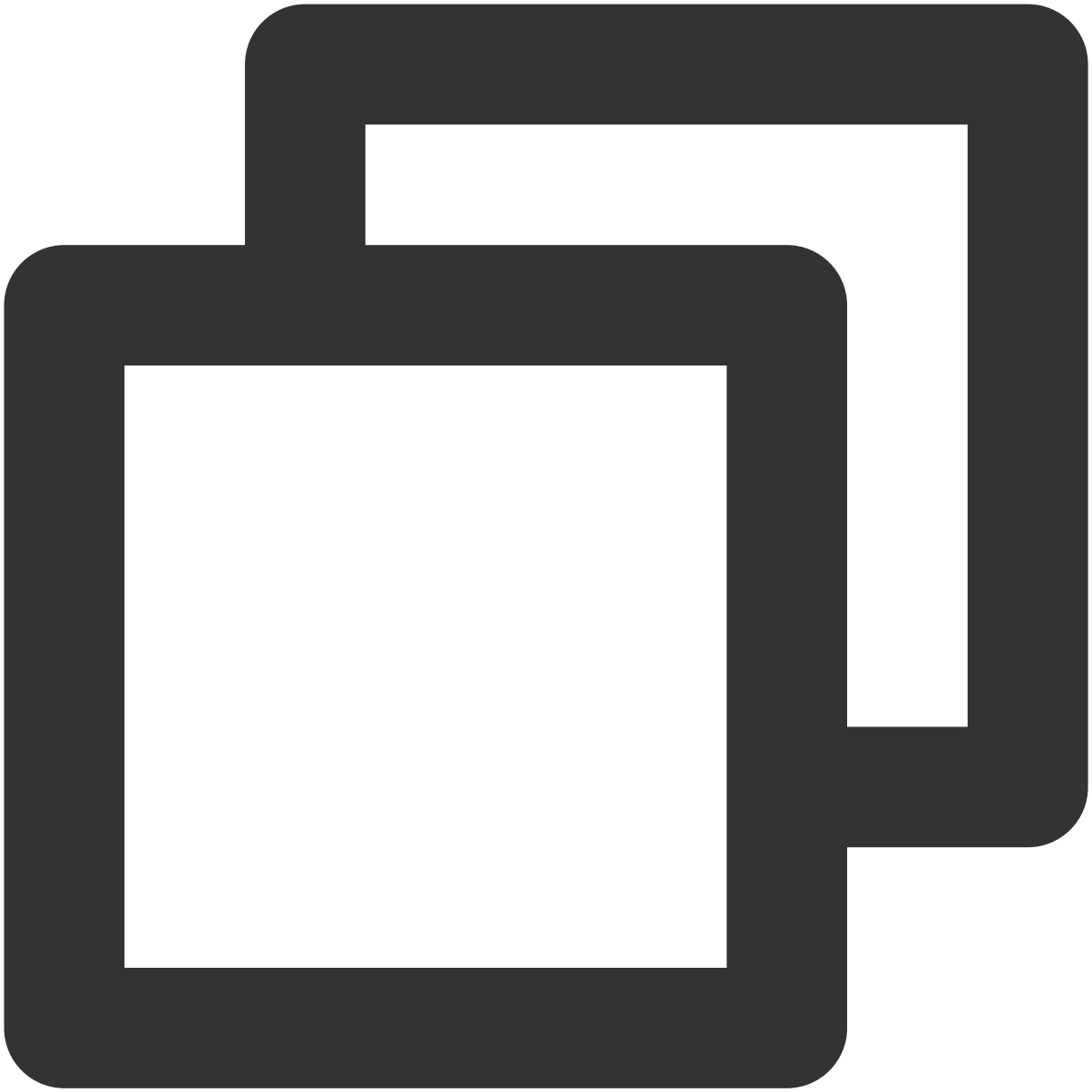
```
events {
    worker_connections 1024;
}

http {
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile            on;
    tcp_nopush          on;
    tcp_nodelay         on;
    keepalive_timeout   65;
    types_hash_max_size 2048;

    include             /etc/nginx/mime.types;
    default_type        application/octet-stream;

    # Load modular configuration files from the /etc/nginx/conf.d directory.
    # See http://nginx.org/en/docs/ngx_core_module.html#include
    # for more information.
    include /etc/nginx/conf.d/*.conf;

    server {
        listen       80 default_server;
        listen       [::]:80 default_server;
        server_name  _;
        root         /var/www/nextcloud;

        add_header Referrer-Policy "no-referrer" always;
        add_header X-Content-Type-Options "nosniff" always;
        add_header X-Download-Options "noopen" always;
        add_header X-Frame-Options "SAMEORIGIN" always;
        add_header X-Permitted-Cross-Domain-Policies "none" always;
        add_header X-Robots-Tag "none" always;
        add_header X-XSS-Protection "1; mode=block" always;

        client_max_body_size 512M;
        fastcgi_buffers 64 4K;

        gzip on;
        gzip_vary on;
        gzip_comp_level 4;
        gzip_min_length 256;
```

```
        gzip_proxied expired no-cache no-store private no_last_modified no_etag aut
        gzip_types application/atom+xml application/javascript application/json app

        # Load configuration files for the default server block.
        include /etc/nginx/default.d/*.conf;

        location / {
            try_files $uri $uri/ =404;
            index index.php;
        }

        location ~ ^\\/(?:build|tests|config|lib|3rdparty|templates|data)\\/ {
            deny all;
        }
        location ~ ^\\/(?:\\.|autotest|occ|issue|indie|db_|console) {
            deny all;
        }

        location ~ ^\\/(?:index|remote|public|cron|core\\/ajax\\/update|status|ocs\
            fastcgi_split_path_info ^(.+?\\.php)(\\/.*|)$;
            set $path_info $fastcgi_path_info;
            try_files $fastcgi_script_name =404;
            include fastcgi_params;
            fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
            fastcgi_param PATH_INFO $path_info;
            fastcgi_param modHeadersAvailable true;
            fastcgi_pass 127.0.0.1:9000;
            fastcgi_intercept_errors on;
            fastcgi_request_buffering off;
        }

        location ~ ^\\/(?:updater|oc[ms]-provider)(?:$|\\/) {
            try_files $uri/ =404;
            index index.php;
        }

        location ~ \\.(css|js|svg|gif)$ {
            add_header Cache-Control "max-age=15778463";
        }

        location ~ \\.woff2?$ {
            add_header Cache-Control "max-age=604800";
        }
    }

# Settings for a TLS enabled server.
#
```

```
#    server {
#        listen       443 ssl http2 default_server;
#        listen       [::]:443 ssl http2 default_server;
#        server_name  _;
#        root         /usr/share/nginx/html;
#
#        ssl_certificate "/etc/pki/nginx/server.crt";
#        ssl_certificate_key "/etc/pki/nginx/private/server.key";
#        ssl_session_cache shared:SSL:1m;
#        ssl_session_timeout  10m;
#        ssl_ciphers HIGH:!aNULL:!MD5;
#        ssl_prefer_server_ciphers on;
#
#        # Load configuration files for the default server block.
#        include /etc/nginx/default.d/*.conf;
#
#        location / {
#        }
#
#        error_page 404 /404.html;
#            location = /40x.html {
#        }
#
#        error_page 500 502 503 504 /50x.html;
#            location = /50x.html {
#        }
#    }

}
```

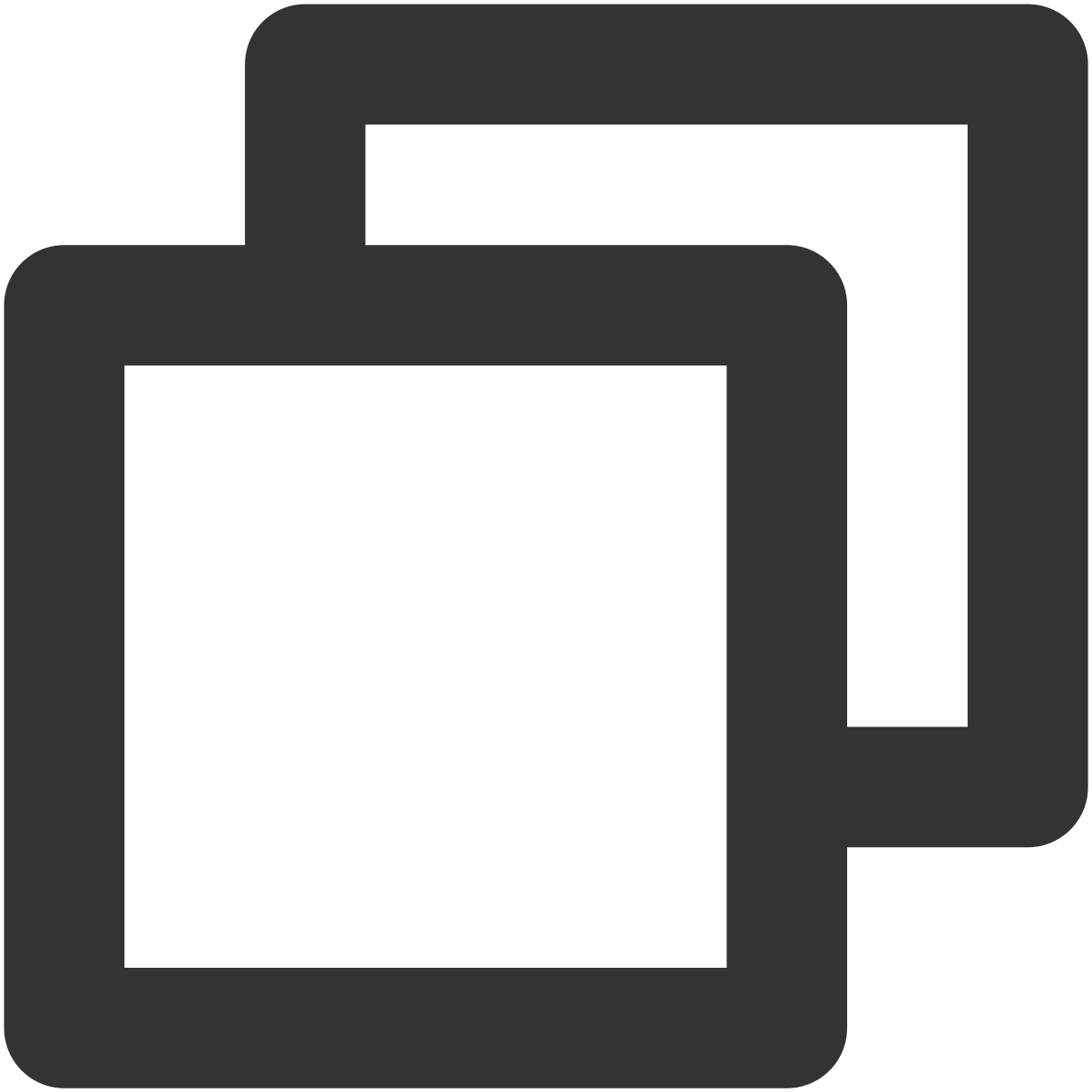
4. 依次执行下述命令，并启动 Nginx 服务：

命令1：



```
systemctl enable nginx
```

命令2：



```
systemctl start nginx
```

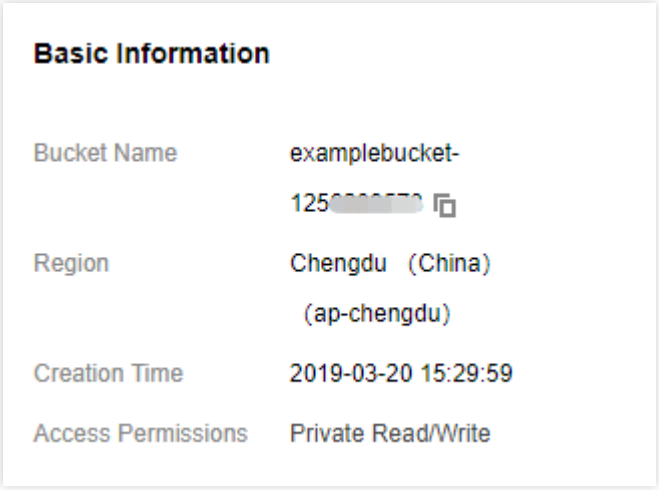
配置 NextCloud 服务端使用对象存储

获取对象存储相关信息

1. 登录 对象存储控制台。
2. 找到此前创建的存储桶，并单击存储桶名称。

examplebucket-125[redacted]	Specified user	Chengdu（China） (ap-chengdu)	2019-03-20 15:29:59

3. 在左侧导航栏中，选择**概览**页签，记录**基本信息**中的**存储桶名称**和**所属地域**中的英文部分。

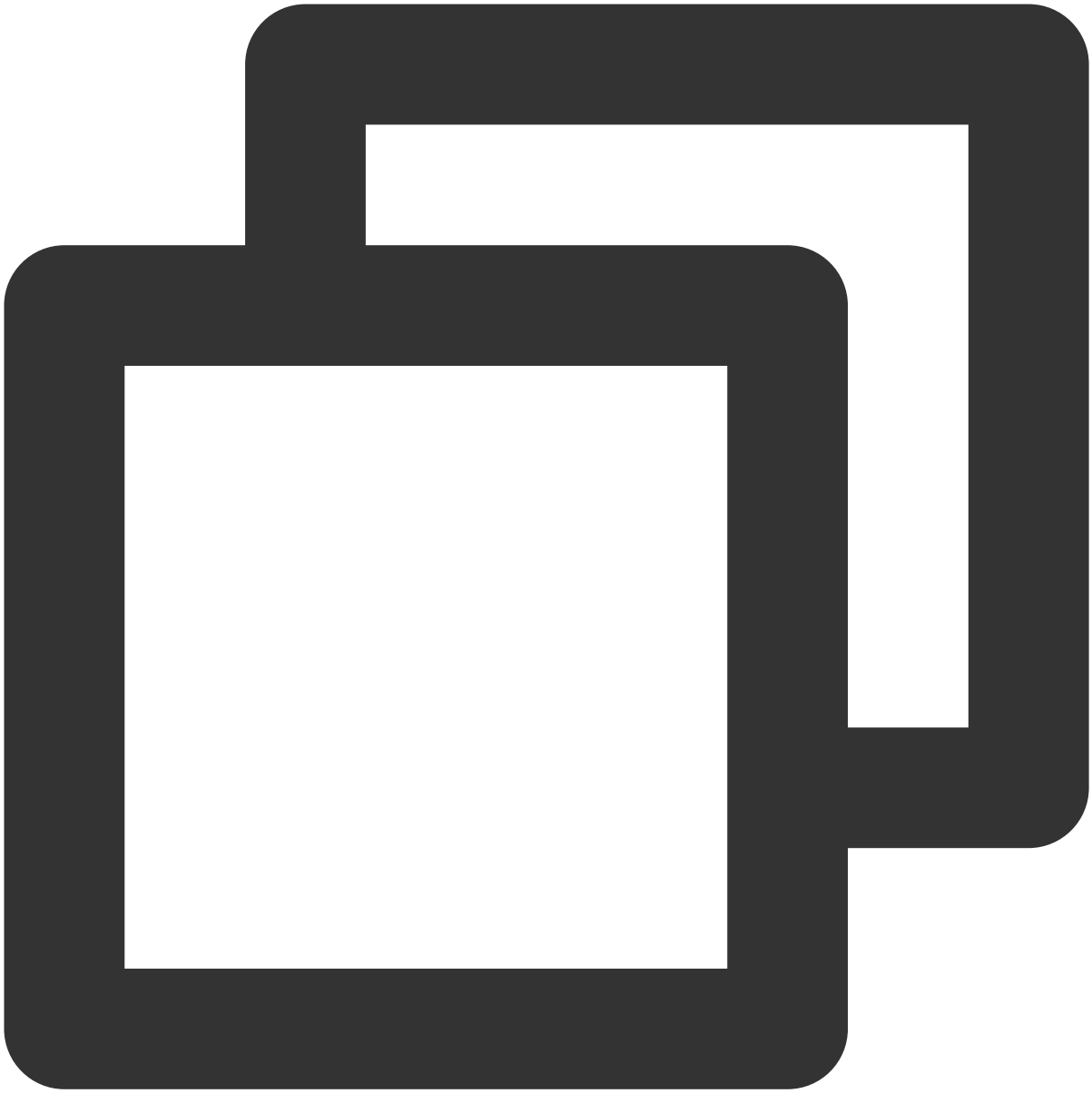


获取 API 密钥

建议使用子账号密钥，授权遵循 最小权限指引，降低使用风险，子账号密钥获取可参考 子账号访问密钥管理。

修改 NextCloud 服务端配置文件

1. 使用文本编辑工具创建 `config.php` ，输入如下内容并根据注释修改相关的值：



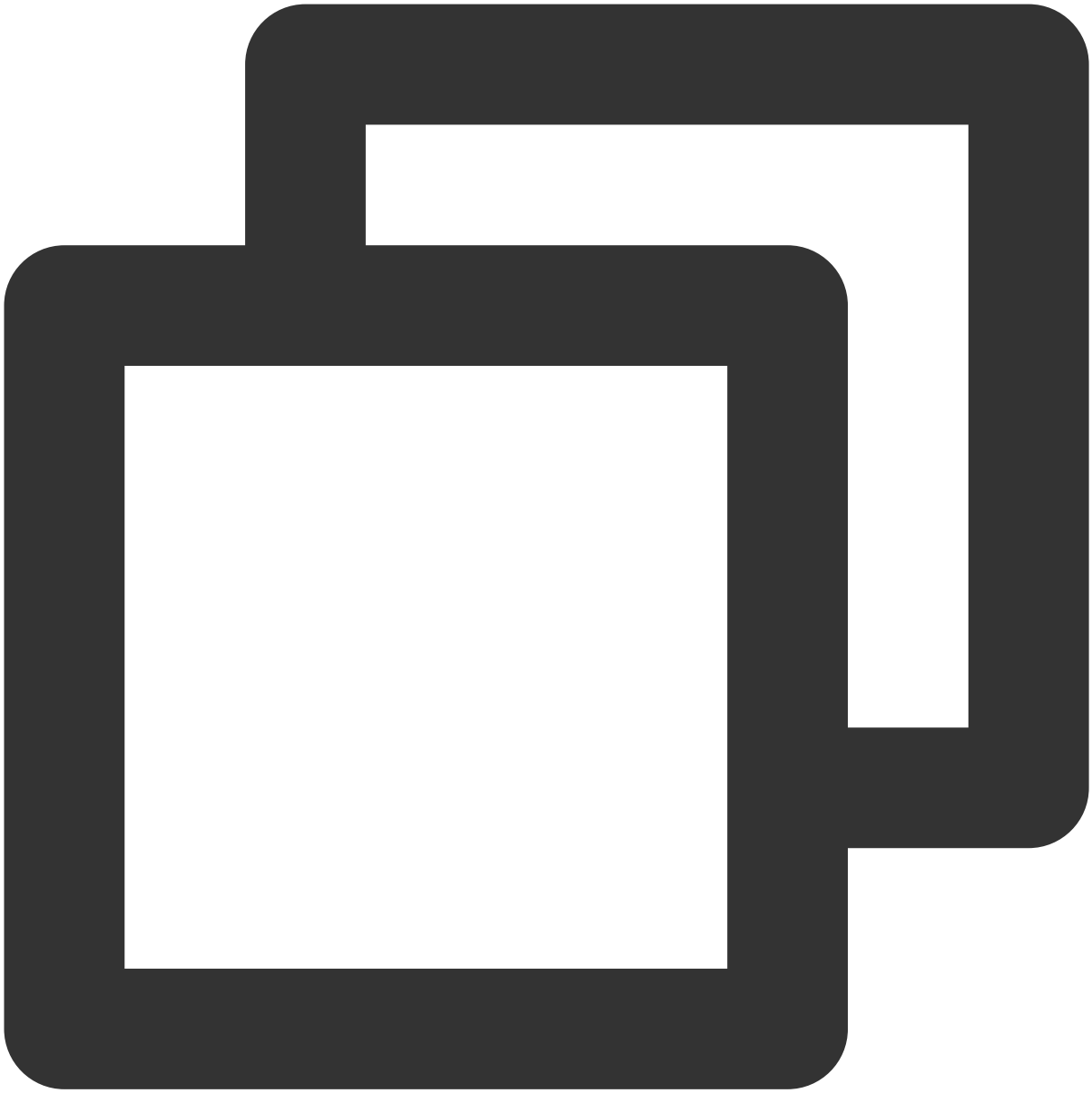
```
<?php
$CONFIG = array(
  'objectstore' => array(
 'class' => '\\\\OC\\\\Files\\\\ObjectStore\\\\S3',
 'arguments' => array(
   'bucket' => 'nextcloud-1250000000', // 存储桶名称（空间名称）
   'autocreate' => false,
   'key'  => 'AKIDxxxxxxxx', // 替换为用户的 SecretId
   'secret' => 'xxxxxxxxxxxx', // 替换为用户的 SecretKey
   'hostname' => 'cos.<Region>.myqcloud.com', // 将 <Region> 修改为所属地域，如 ap-shar
   'use_ssl' => true,
```

```
  ),
   ),
);
```

如下图所示：

2. 将该文件保存并上传到 `/var/www/nextcloud/config/` 目录下（保持文件名为 `config.php` ），您可以通过 SFTP 或 SCP 软件上传文件，也可以通过 `rz -bye` 命令上传。

3. 执行下述命令修改配置文件的所有者：



```
chown nginx:nginx /var/www/nextcloud/config/config.php
```

配置域名

若您计划使用自己的域名而不是 IP 地址访问您的 NextCloud 服务端，您可参考各域名注册商的说明文档，注册新域名并将域名解析到您 CVM 的 IP 地址并完成备案。

由于 NextCloud 服务端在安装过程中会记录安装时使用的域名或 IP 地址，因此建议您在开始安装前完成域名的注册、解析和备案，并使用域名访问 NextCloud 服务端的安全界面。
如果您在完成 NextCloud 服务端后需要更换域名或 IP 地址，您可以自行修改 `/var/www/nextcloud/config/config.php` 配置文件中的 `trusted_domains`，详情请参阅 NextCloud 官方文档。

安装 NextCloud 服务端

1. 使用浏览器访问 NextCloud 服务端，创建并牢记管理员用户名和密码。
2. 展开**存储与数据库**，根据下表说明进行配置：

配置项	值
数据名录	/var/www/nextcloud/data（保持默认）
配置数据库	MySQL/MariaDB
数据库用户	root
数据库密码	初始化云数据库 MySQL 时填写的 root 密码
数据库名	nextcloud（或其他未被使用的数据库名）
数据库主机（默认显示为 localhost）	云数据库 MySQL 的内网地址

3. 单击**安装完成**，等待 NextCloud 服务端完成安装。
4. 如果安装过程中出现504 Gateway Timeout 等错误信息，可直接刷新重试。
5. 安装完成后，使用管理员账号登录 NextCloud 服务端即可开始使用网页版 NextCloud。

NextCloud 服务端调优

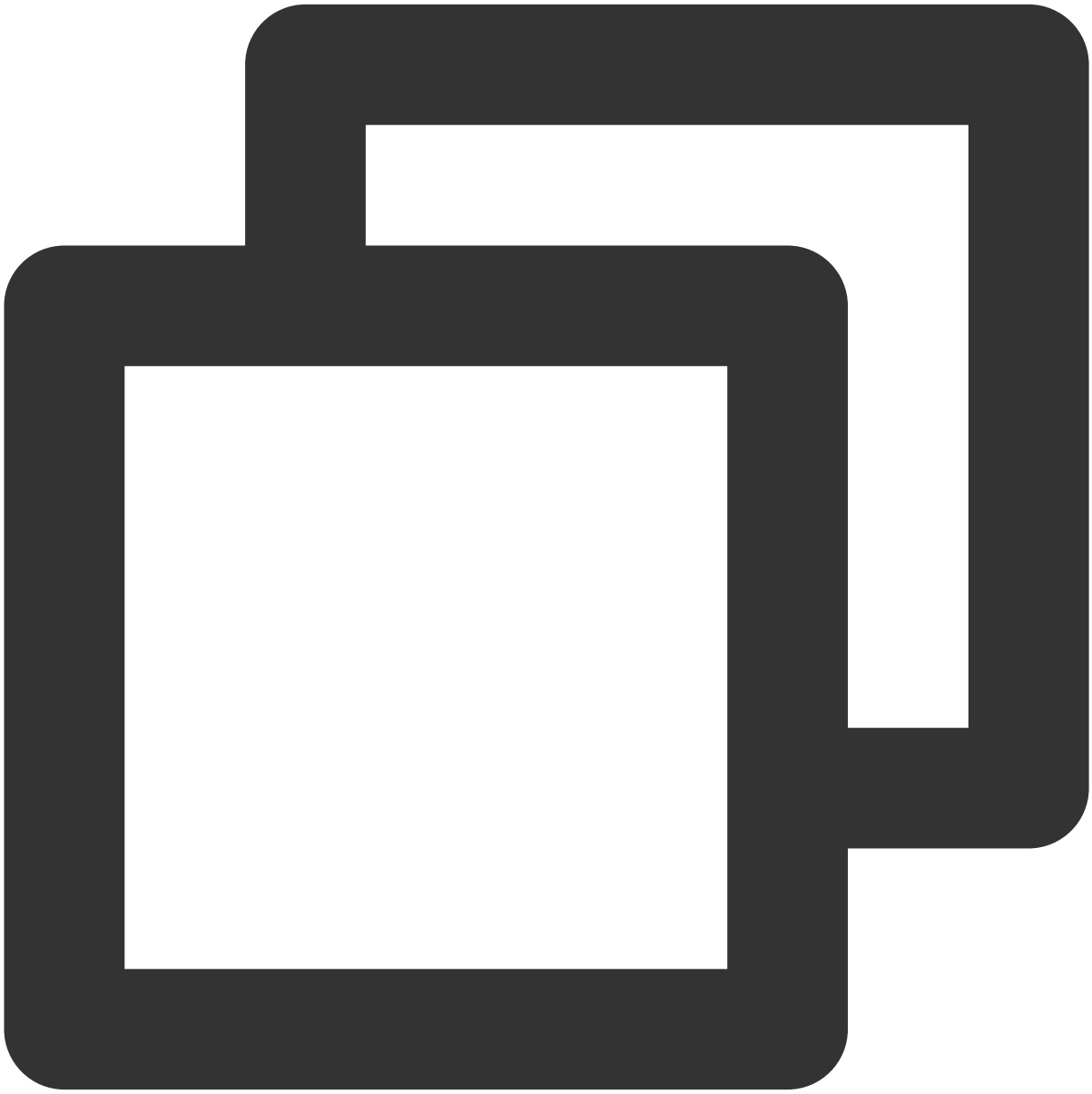
后台任务

NextCloud 服务端有时需要在无需用户交互时执行一些后台任务，例如数据库清理等操作。PHP 的运行特性限制了基于 PHP 的程序无法内部维持一个独立的工作进程或线程，因此类似后台任务的场景需要由外部主动调用对应的 PHP 程序来执行。
NextCloud 服务端提供3种后台任务的调用方式，默认是通过网页端的登录用户，在浏览器自动发起 AJAX 请求来唤起服务端的后台任务执行，这种方式严重依赖用户的登录情况，如果没有用户登录，那么这些后台任务将无法执行，因此可靠性最低。

为了避免基于 AJAX 的后台任务可靠性低的问题，我们推荐使用 Linux 下的 cron 来配置后台任务，Linux 下的 cron 可以精确的控制任务被唤起的时间，例如每5分钟（分钟数为5的整数倍）或每个整点的第10分钟等等，可定义的时间粒度包括分钟、小时、一个月中的某天、月份和星期几，一些操作系统还支持秒和年，因此具备高度灵活性。有关 cron 的相关说明和配置，您可参考相关资料。

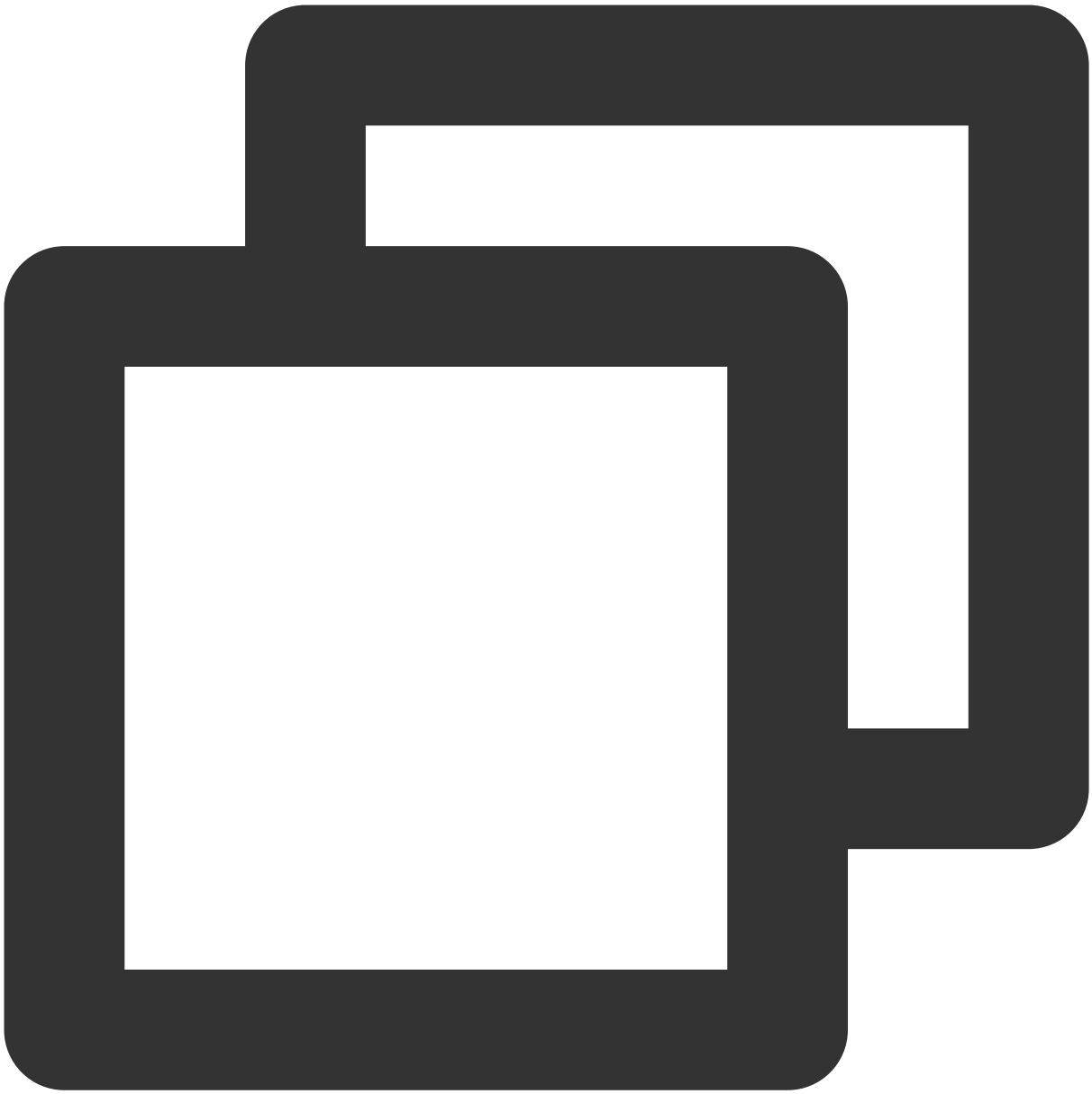
下面将介绍下如何配置 cron 来满足 NextCloud 服务端的后台任务：

1. 使用 SSH 工具登录到新购服务器。
2. 执行下述命令安装 PHP 模块：



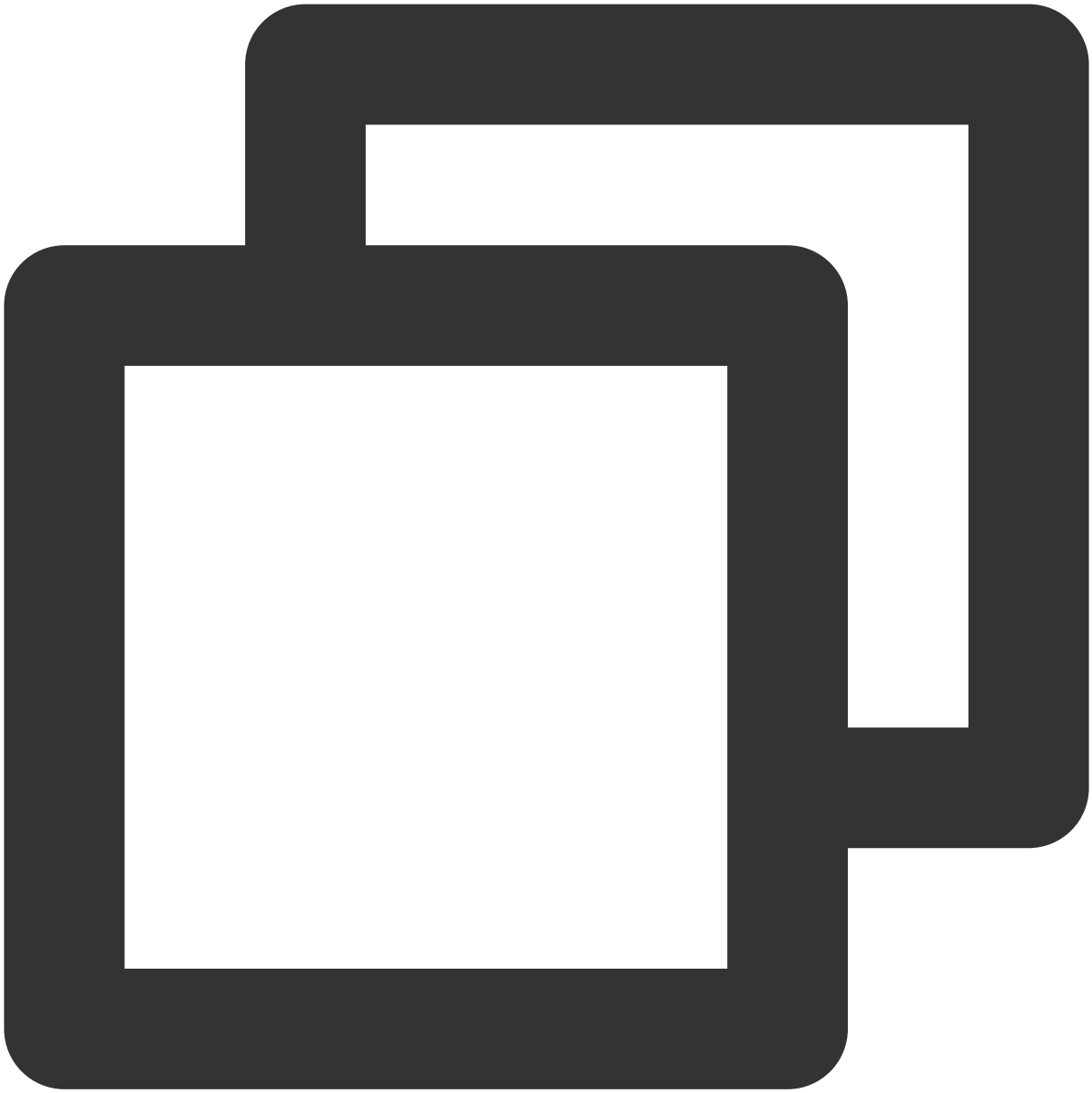
```
yum install php-posix
```

3. 执行下述命令打开或创建用于 nginx 账户的 cron 配置：



```
crontab -u nginx -e
```

4. 接下来的编辑界面为 vi/vim，按 `i` 键进入编辑模式，插入一行，内容如下：



```
*/5 * * * * php -f /var/www/nextcloud/cron.php
```

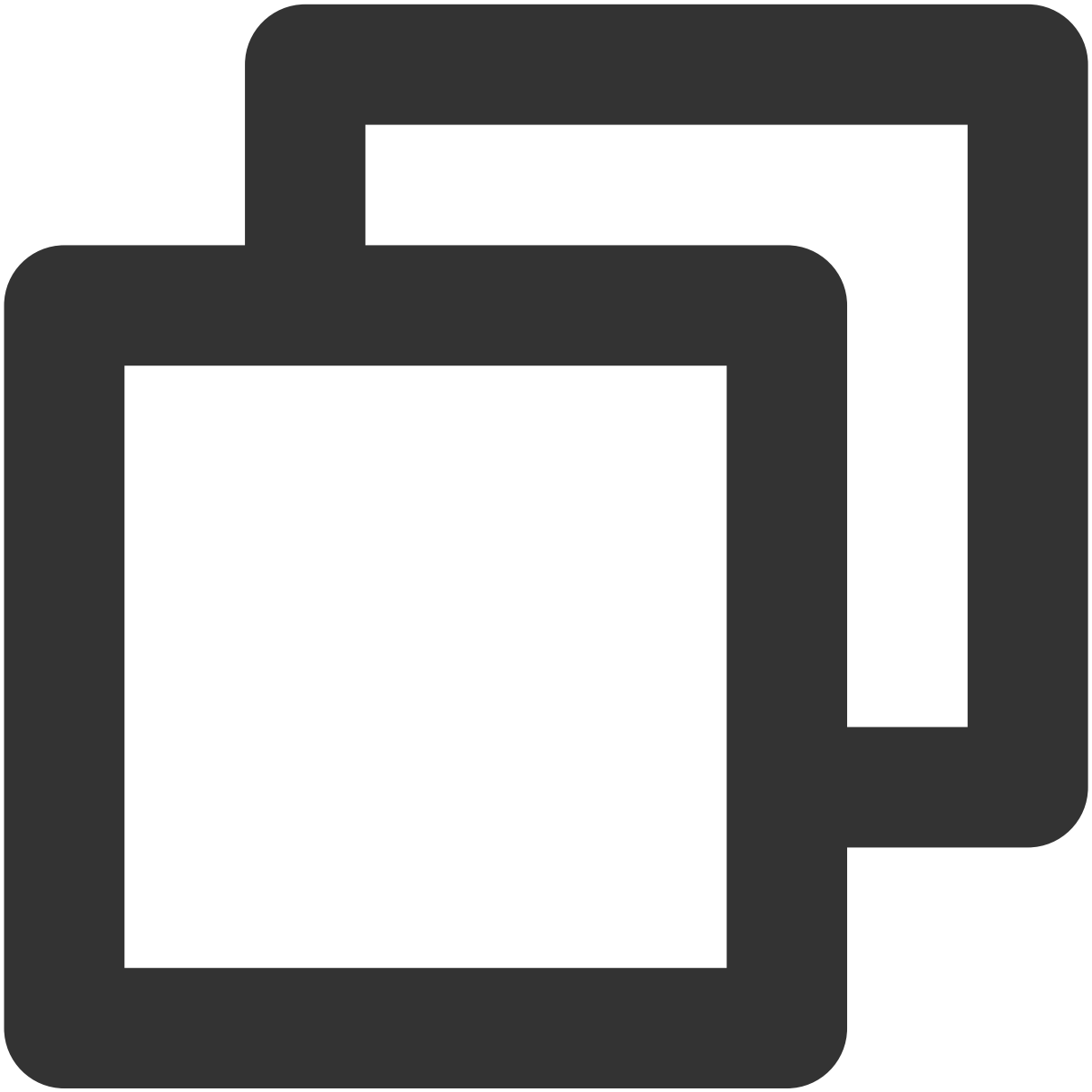
随后按 `ESC` 退出编辑模式，输入 `:wq` 保存退出（有关 vi/vim 的详细操作指引，请参阅相关文档）。

上述配置使用了 NextCloud 官方推荐的每5分钟执行一次（分钟数是5的整数倍）。待5分钟后后台任务执行完成，可以打开浏览器登录 NextCloud 服务端，单击右上角用户名首字母图标，进入**设置**，在左侧菜单进入**基本设置**，可以看到**后台任务**处默认选中了 Cron。

内存缓存

PHP 可以使用 OPcache 提升性能，NextCloud 服务端也支持使用 APCu 内存缓存进一步提升性能，下面将介绍下相关的操作流程：

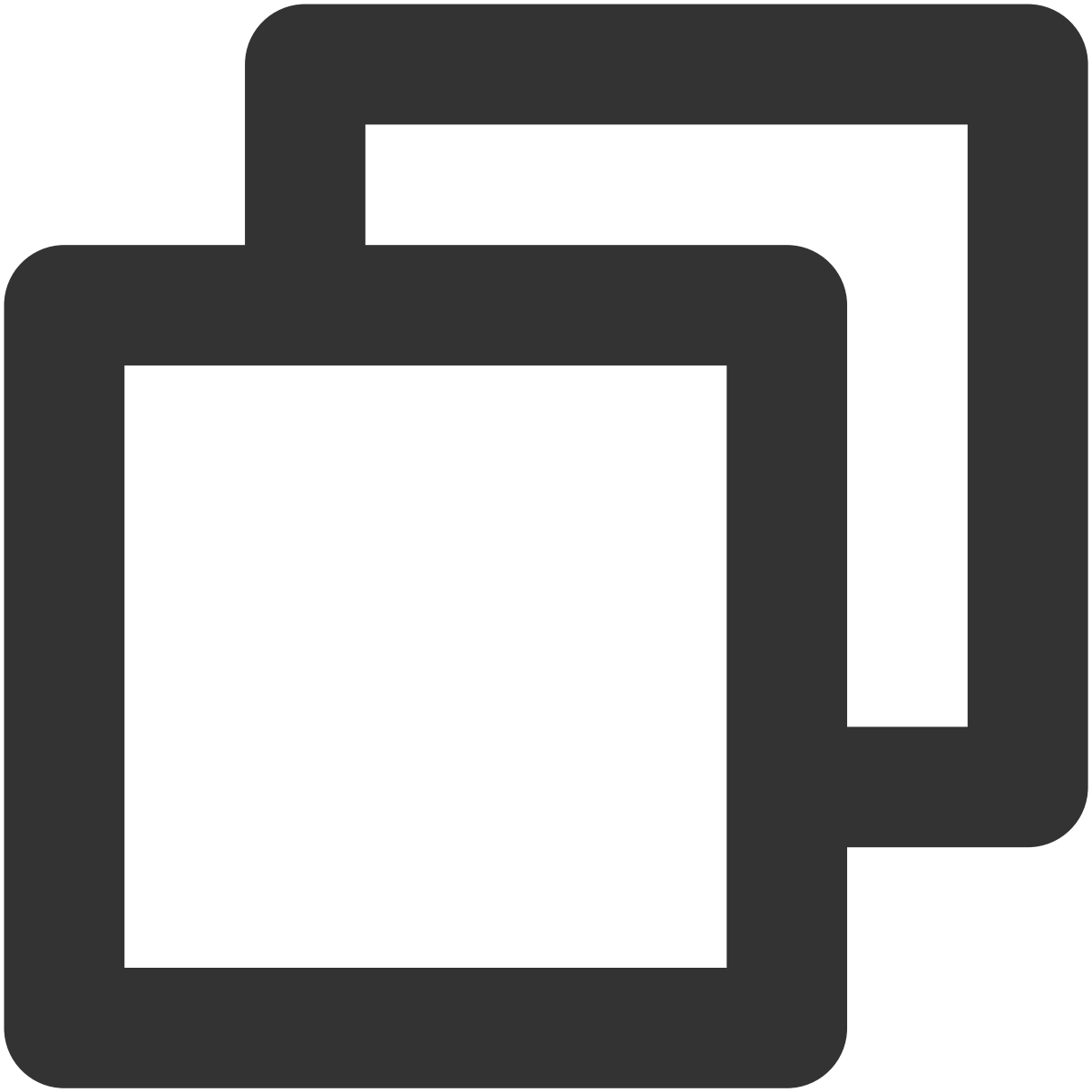
1. 使用 SSH 工具登录到新购服务器。
2. 执行下述命令安装 PHP 模块：



```
yum install php-pecl-apcu
```

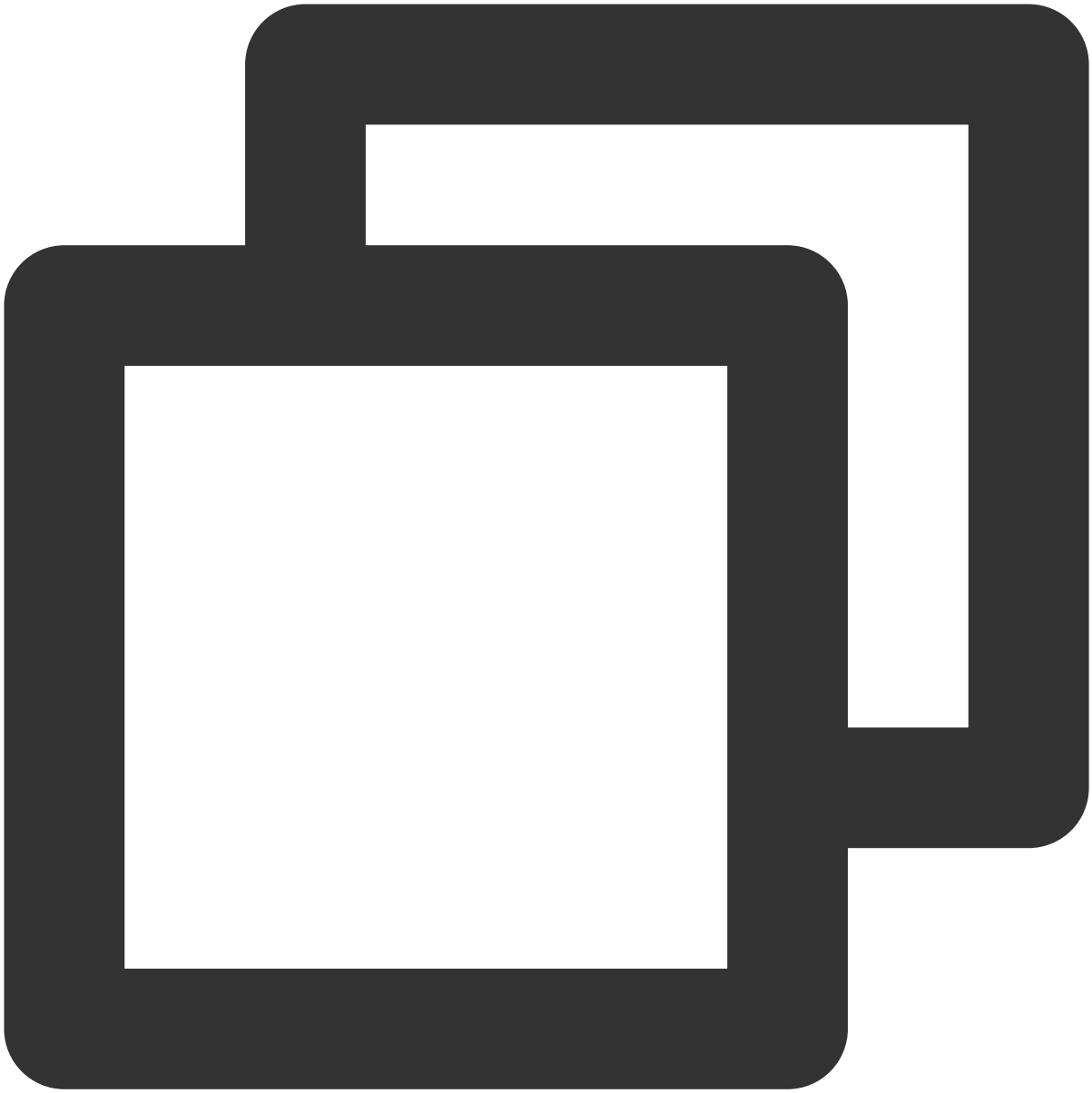
3. 依次执行下述命令重启 Nginx 和 PHP-FPM：

命令1：



```
systemctl restart nginx
```

命令2：



```
systemctl restart php-fpm
```

4. 执行 `vim /var/www/nextcloud/config/config.php`，打开 NextCloud 服务端的配置文件，在 `$CONFIG = array (` 中添加一行：`'memcache.local' => '\\OC\\Memcache\\APCu',`，随后保存文件并退出。

```
[root@VM-0-10-centos config]# vim /var/www/nextcloud/config/config.
<?php
$CONFIG = array (
  'objectstore' =>
  array (
    'class' => '\\OC\\Files\\ObjectStore\\S3',
    'arguments' =>
    array (
      'bucket' => 'nextcloud-125     555',
      'autocreate' => false,
      'key' => 'AKID                                  6NEu',
      'secret' => 'nT2P                               TH0X',
      'hostname' => 'cos.ap-shanghai.myqcloud.com',
      'use_ssl' => true,
    ),
  ),
  'instanceid' => '          ',
  'passwordsalt' => '                                ',
  'secret' => '                                                                ',
  'trusted_domains' =>
  array (
    0 => '                ',
  ),
  'datadirectory' => '/var/www/nextcloud/data',
  'dbtype' => 'mysql',
  'version' => '19.0.1.1',
  'overwrite.cli.url' => 'http://                ',
  'dbname' => 'nextcloud',
  'dbhost' => '172.17.0.13:3306',
  'dbport' => '',
  'dbtableprefix' => 'oc_',
  'mysql.utf8mb4' => true,
  'dbuser' => 'oc_          ',
  'dbpassword' => '                                ',
  'installed' => true,
  'memcache.local' => '\OC\Memcache\APCu',
);
```

5. 如果配置了 cron 来优化后台任务，那么还需要修改 PHP 的 apc 配置：执行 `vim /etc/php.d/40-apcu.ini`，打开 PHP APCu 的配置文件，将 `;apc.enable_cli=0` 修改为 `apc.enable_cli=1`（请注意需要同时去掉前面分号），保存退出。如果路径 `/etc/php.d/40-apcu.ini` 不存在，那么请自行在 `/etc/php.d/` 目录下查找并编辑有 apc 或 apcu 字样的 `.ini` 配置文件。

```
[root@VM-0-10-centos data]# vim /etc/php.d/40-apcu.ini
; Enable APCu extension module
extension = apcu.so

;       This can be set to 0 to disable APCu
apc.enabled=1

;       Setting this enables APCu for the CLI version of PHP
;       (Mostly for testing and debugging).
apc.enable_cli=1
```

配置客户端访问

NextCloud 官方提供桌面同步客户端和移动客户端，可在 NextCloud 官网或各大应用商店下载。在配置 NextCloud 时需输入 NextCloud 的服务端地址（域名或 IP），随后输入自己的用户名和密码并登录，即可开始使用客户端。

将 COS 作为本地磁盘挂载到 Windows 服务器

最近更新时间：2024-01-06 10:54:03

操作场景

在 Windows 系统上操作腾讯云对象存储（Cloud Object Storage，COS），目前主要的实现方式还是通过 API、COSBrowser、COSCMD 工具。
而对于喜欢使用 Windows 服务器的用户，使用 COSBrowser 工具大多只能当做网盘，对服务器上的直接使用程序或者操作并不友好。本文将介绍如何使存储价格低廉的对象存储挂载到 Windows 服务器上映射为本地磁盘。

说明

本案例实践适用于 Windows 7 / Windows Server 2012 / 2016 / 2019 / 2022系统。

操作步骤

下载与安装

本案例实践使用到以下三种软件，您可选择安装适用于自己所使用系统的软件版本：
1. 前往 Github 下载 Winfsp。
本实践下载的版本为 winfsp-1.12.22301，下载完成后，按步骤默认安装即可。

说明

Windows Server 2012 R2 不适用于 Winfsp 1.12.22242版本，可适用于 Winfsp 1.11.22176版本。

2. 前往 Git 官网 或者 Github 下载 Git 工具。
本实践下载的版本为 Git-2.38.1-64-bit，下载完成后，按步骤默认安装即可。
3. 前往 Rclone 官网 或者 Github 下载 Rclone 工具。
本实践下载的版本是 rclone-v1.60.1-windows-amd64，该软件无需安装，下载后，只需解压到任意一个英文目录下即可（如果解压到的路径含有中文将有可能会报错）。本实践案例路径举例为 E:\\AutoRclone。

说明

Github 下载速度可能比较慢甚至打不开，可自行在其他官方渠道进行下载。

配置 Rclone

注意

以下配置步骤以 rclone-v1.60.1-windows-amd64 版本为例，其他版本的配置过程可能存在一定差异，请注意相应调整。

1. 打开任意文件夹，并在左侧导航目录下找到 **此电脑**，单击右键选择 **属性 > 高级系统设置 > 环境变量 > 系统变量 > Path**，单击**新建**。
2. 在弹出的窗口中，填写 Rclone 解压后的路径（E:\\AutoRclone），单击 **确定**。
3. 打开 Windows Powershell，输入 `rclone --version` 命令，按 **Enter**，查看 Rclone 是否成功安装。
4. 确认 Rclone 安装成功后，在 Windows Powershell 中，输入 `rclone config` 命令，按 **Enter**。
5. 在 Windows Powershell 中，输入 **n** ，按 **Enter**，新建一个 New remote。
6. 在 Windows Powershell 中，输入该磁盘的名称，例如 myCOS，按 **Enter**。
7. 在显示的选项中，选择包含 “Tencent COS” 的选项，即输入 **5**，按 **Enter**。

```
Option Storage.
Type of storage to configure.
Choose a number from below, or type in your own value.
 1 / 1Fichier
   \ (fichier)
 2 / Akamai NetStorage
   \ (netstorage)
 3 / Alias for an existing remote
   \ (alias)
 4 / Amazon Drive
   \ (amazon cloud drive)
 5 / Amazon S3 Compliant Storage Providers including AWS, Alibaba, Ceph, China Mobile, Cloudflar
ean, Dreamhost, Huawei OBS, IBM COS, IDrive e2, IONOS Cloud, Lyve Cloud, Minio, Netease, RackCor
tackPath, Storj, Tencent COS, Qiniu and Wasabi
   \ (s3)
 6 / Backblaze B2
   \ (b2)
```

8. 在显示的选项中，选择包含 “TencentCOS” 的选项，输入 **21**，按 **Enter**。

```
20 / Storj (S3 Compatible Gateway)
   \ (Storj)
21 / Tencent Cloud Object Storage (COS)
   \ (TencentCOS)
22 / Wasabi Object Storage
   \ (Wasabi)
23 / Qiniu Object Storage (Kodo)
   \ (Qiniu)
24 / Any other S3 compatible provider
   \ (Other)
provider> 21
```

9. 执行到 `env_auth>` 时，按 Enter。
10. 执行到 `access_key_id>` 时，输入腾讯云 COS 的访问密钥 SecretId，按 **Enter**。

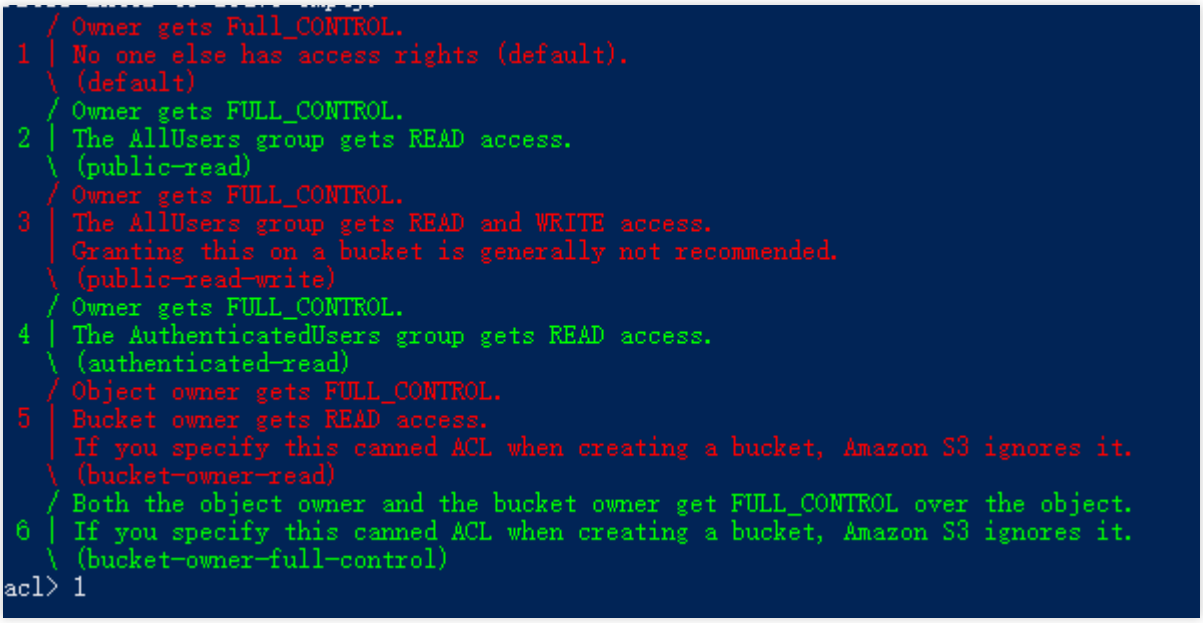
说明

此处建议使用子账号权限，可前往 API 密钥管理 查看自己的 SecretId 和 SecretKey。

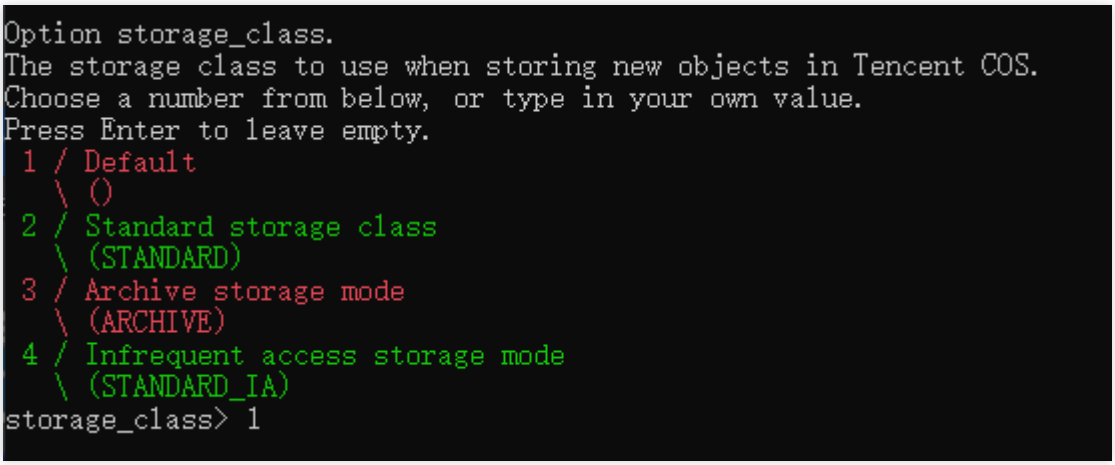
11. 执行到 `secret_access_key>` 时，输入腾讯云 COS 的访问密钥 SecretKey，按 **Enter**。
12. 根据显示的腾讯云各地域的网关地址，查看存储桶的所属地域，选择对应的地域。

本实践以广州为例，选择 `cos.ap-guangzhou.myqcloud.com` ，输入**4**，按 **Enter**。

13. 在显示的腾讯云 COS 的权限类型中，根据实际需求选择 default、public-read 等。此处选择的权限类型为对象权限类型，仅针对新上传的文件有效。本实践以 default 为例，输入**1**，按 **Enter**。



14. 在显示的腾讯云对象存储的存储类型中，您可根据实际需求选择以何种存储类型将文件上传到 COS。本实践以 Default 为例，输入**1**，按 **Enter**。



Default 表示默认

Standard storage class 表示标准存储（STANDARD）

Archive storage mode 表示归档存储（ARCHIVE）

Infrequent access storage mode 表示低频存储（STANDARD_IA）

说明

如需设置智能分层存储或者深度归档存储类型，请采用 **修改配置文件** 的方式，在配置文件中，将 storage_class 的值设置为 INTELLIGENT_TIERING 或 DEEP_ARCHIVE 即可。关于存储类型的更多介绍，请参见 存储类型概述。

15. 执行到 `Edit advanced config? (y/n)` 时，按 **Enter**。

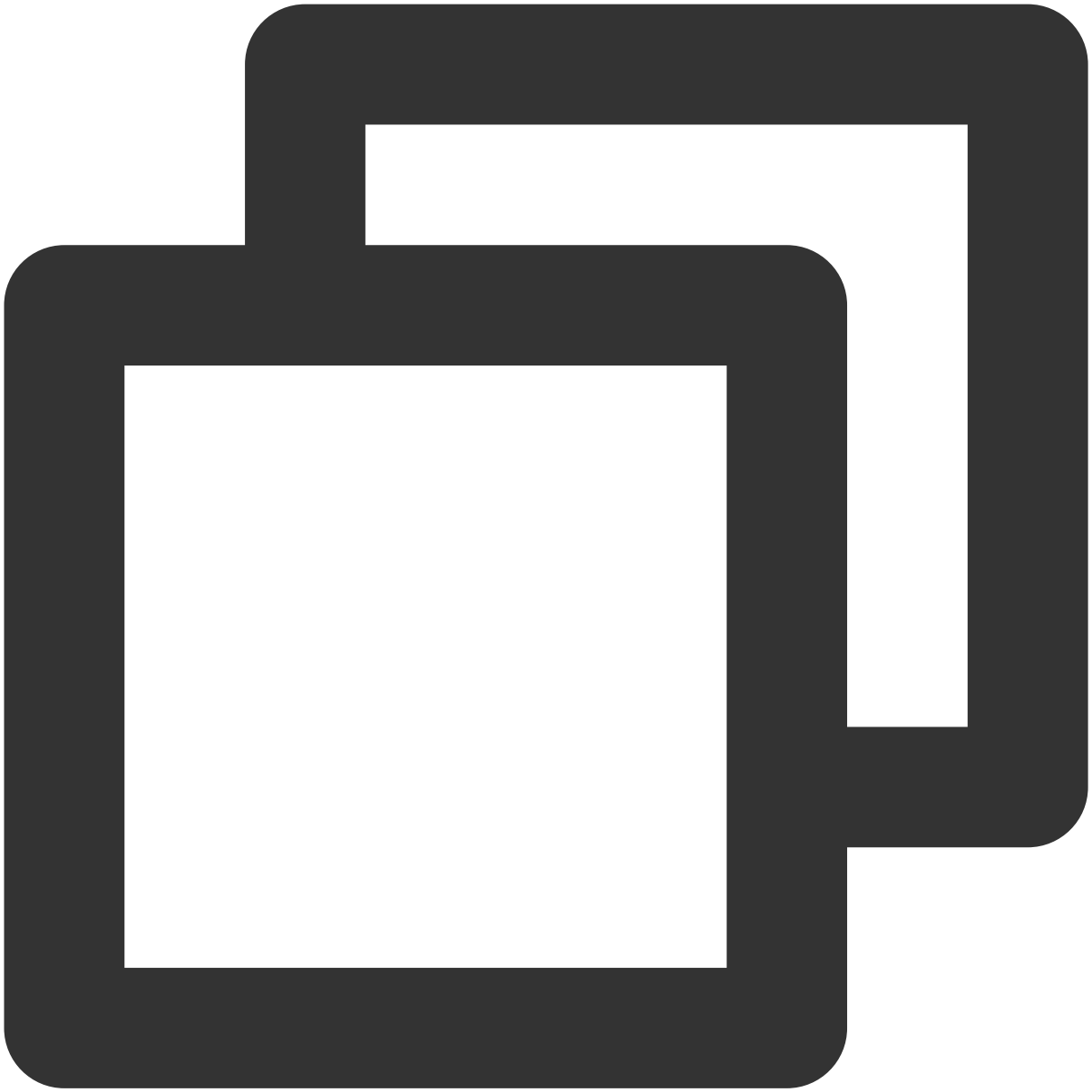
16. 确认信息无误后，按 **Enter**。
17. 输入 **q**，完成配置。

修改配置文件

以上步骤配置完成后，将会生成一个名称为 rclone.conf 的配置文件，一般位于 `C:\\Users\\用户名\\AppData\\Roaming\\rclone` 文件夹下。如果您想要修改 rclone 的配置，可直接对其进行修改。如若未找到该配置文件，可在命令窗口执行 `rclone config file` 命令查看其配置文件。

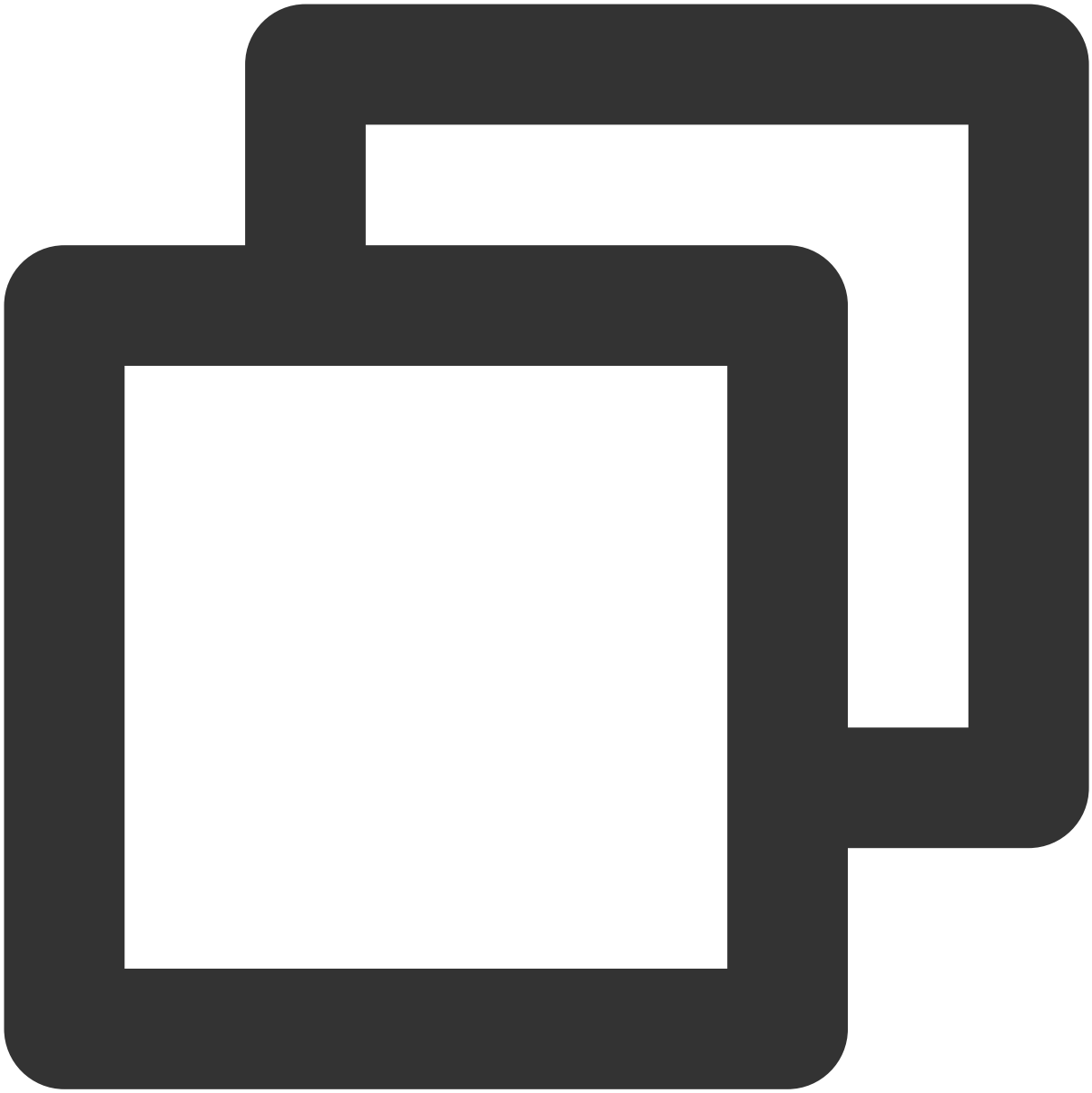
挂载 COS 为本地磁盘

1. 打开已安装的 Git Bash，并输入执行命令。此处提供了两种使用场景（二选一），您可根据实际需求选择其中一种。
如果映射为局域网共享驱动器（推荐），则执行命令如下



```
rclone mount myCOS:/ Y: --fuse-flag --VolumePrefix=\\server\\share --cache-dir E:\\
```

如果映射为局域网共享驱动器（推荐），则执行命令如下：



```
rclone mount myCOS:/ Y: --cache-dir E:\\temp --vfs-cache-mode writes &
```

myCOS：替换为用户自定义的磁盘名称。

Y：替换为您想要挂载后，硬盘的盘符名称即可，请不要与本地的 C、D、E 盘等重复。

E:\\temp 为本地缓存目录，可自行设置。注意：需确保用户拥有目录权限。

当出现提示 “The service rclone has been started” 则说明挂载成功。

2. 输入 **exit**，退出终端。

3. 在本地计算机的**我的电脑**中，即可找到一个名为 myCOS(Y:) 的磁盘。

打开该磁盘，即可查看包含您整个广州地域的所有存储桶名称。此时，您可以进行上传、下载、新建和删除等本地

磁盘的常用操作。

注意

在操作当中如遇报错，请在 git bash 软件中查看详细报错信息。

在挂载磁盘中，若对存储桶进行删除操作，无论存储桶是否存在文件，都将会被删除，请谨慎操作。

若您对挂载磁盘中的存储桶名称进行更改，会导致 COS 存储桶名称发生改变，请谨慎操作。

设置开机自启动挂载硬盘

由于如上操作在电脑重启后，映射的磁盘将会消失，需要再次手工操作。因此，我们可以设置自启动装置，让服务器每次重启后都自动挂载磁盘。

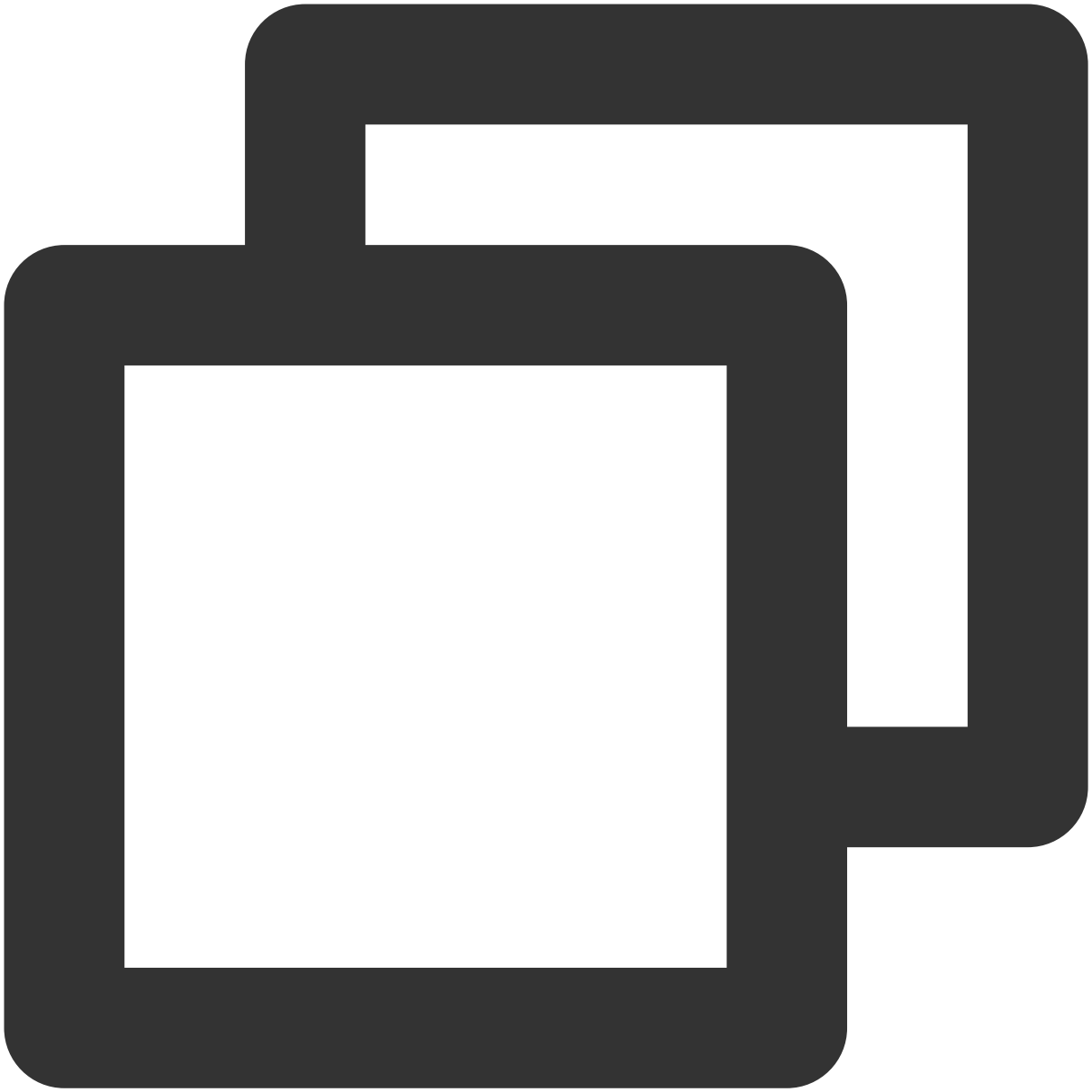
1. 在 Rclone 安装目录 E:\\AutoRclone 下，分别新建 startup_rclone.vbs 和 startup_rclone.bat 文件。

说明

Powershell 创建文本文件时需要注意编码，否则生成的.bat、 .vbs 等文本文件无法执行。

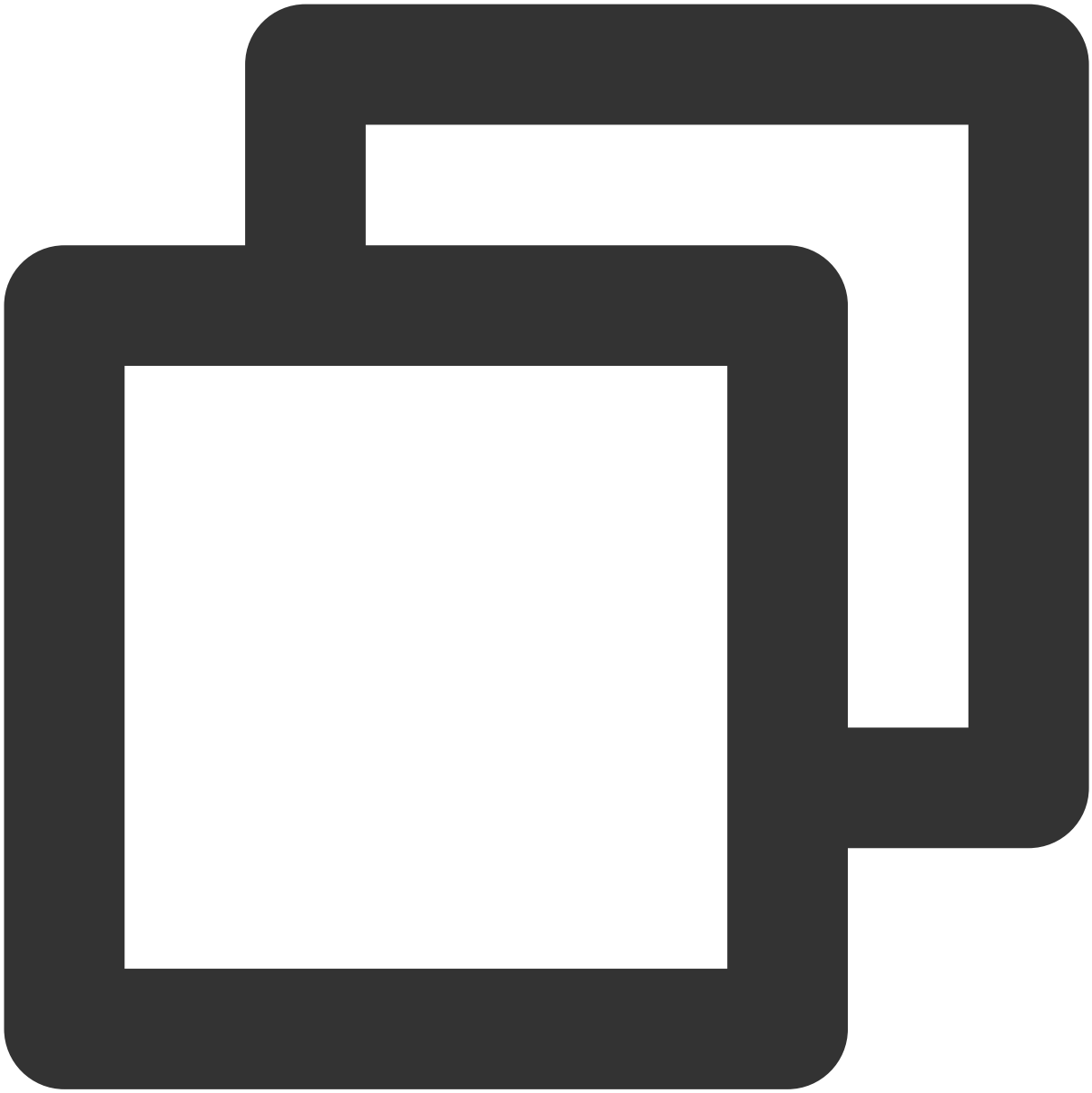
2. 在 startup_rclone.bat 中，写入如下挂载命令：

如果映射为局域网共享驱动器，输入如下命令：



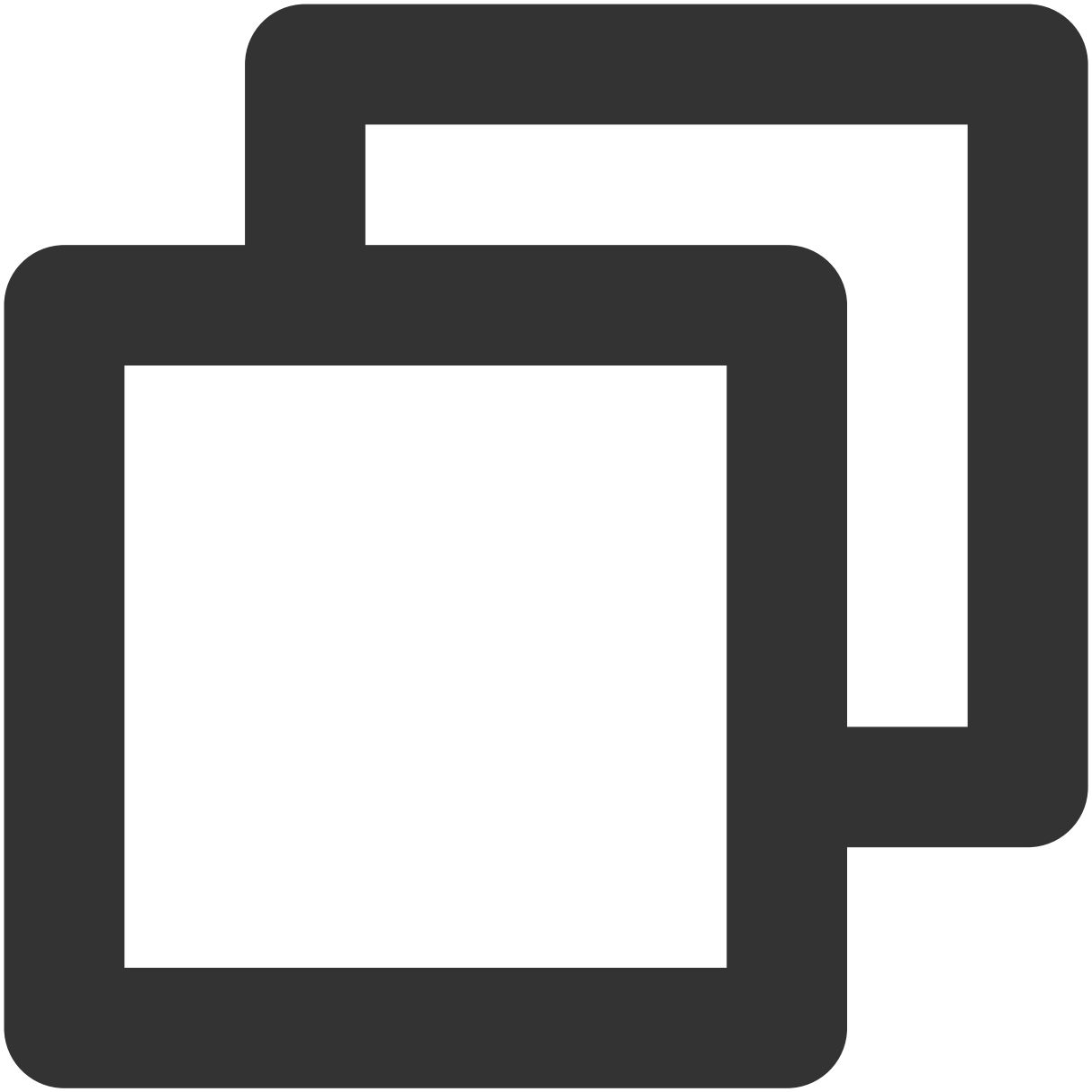
```
rclone mount myCOS:/ Y: --fuse-flag --VolumePrefix=\\server\\share --cache-dir E:\\
```

如果映射为本地磁盘，输入如下命令：



```
rclone mount myCOS:/ Y: --cache-dir E:\\temp --vfs-cache-mode writes &
```

3. 在 startup_rclone.vbs 中，写入如下代码：



```
CreateObject("WScript.Shell").Run "cmd /c E:\\AutoRclone\\startup_rclone.bat",0
```

注意

请将代码中的路径修改为您实际的路径。

4. 将 startup_rclone.vbs 文件剪切到 %USERPROFILE%\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Startup 文件夹下。

5. 重启服务器。

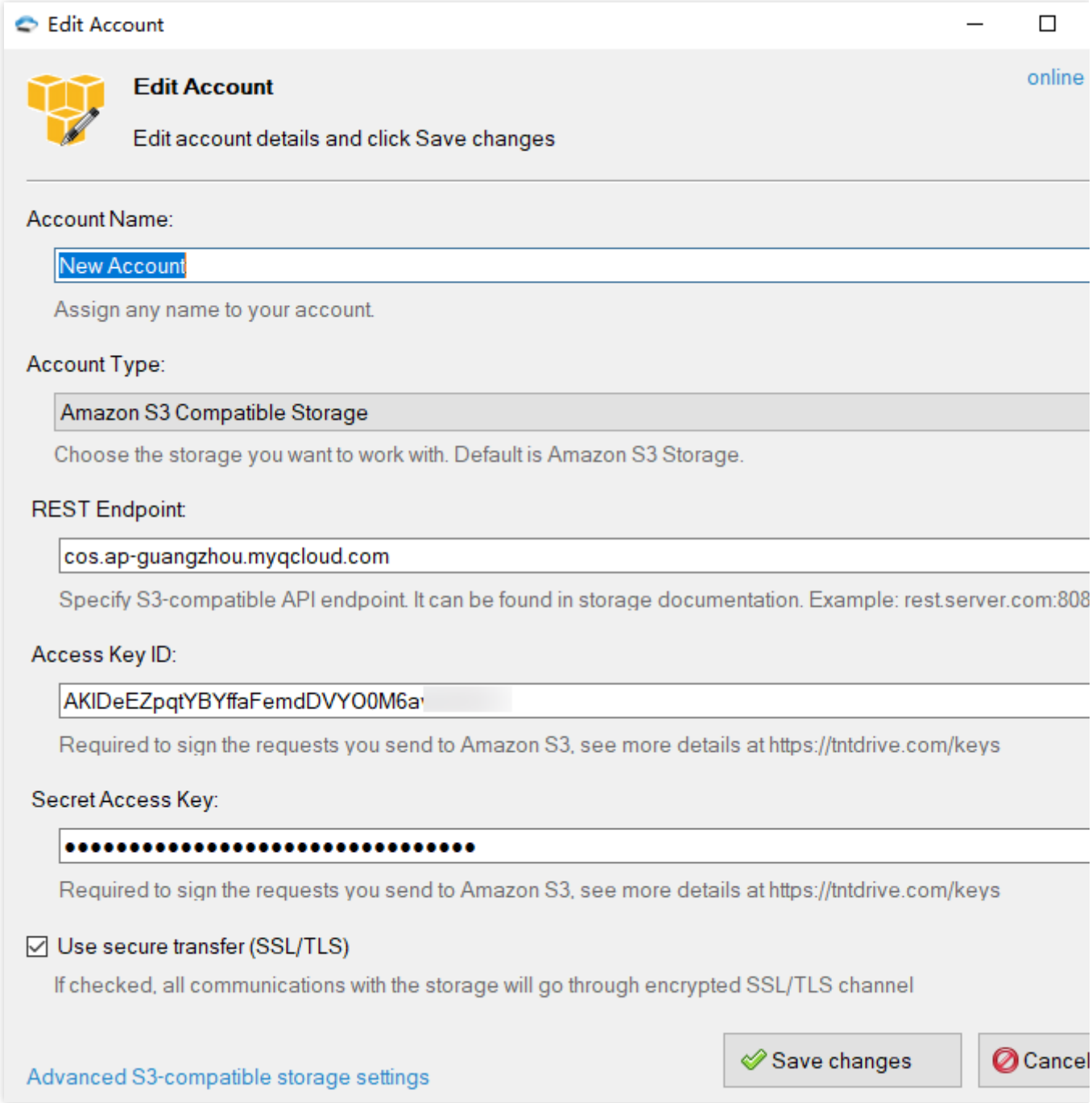
说明

自动挂载配置后并重启服务器，通常情况下需要等待十几秒才能看到挂载成功。

相关操作

您也可以通过使用第三方商业收费工具，将 COS 挂载到 Windows 服务器上映射为本地磁盘。如下操作以 TntDrive 工具为例。

1. 下载和安装 TntDrive。
2. 打开 TntDrive，单击 **Account > Add New Account**，创建一个用户账号。



主要参数信息如下：

Account Name：自定义账号名称。

Account Type：由于 COS 兼容 S3，因此该处可选择 **Amazon S3 Compatible Storage**。

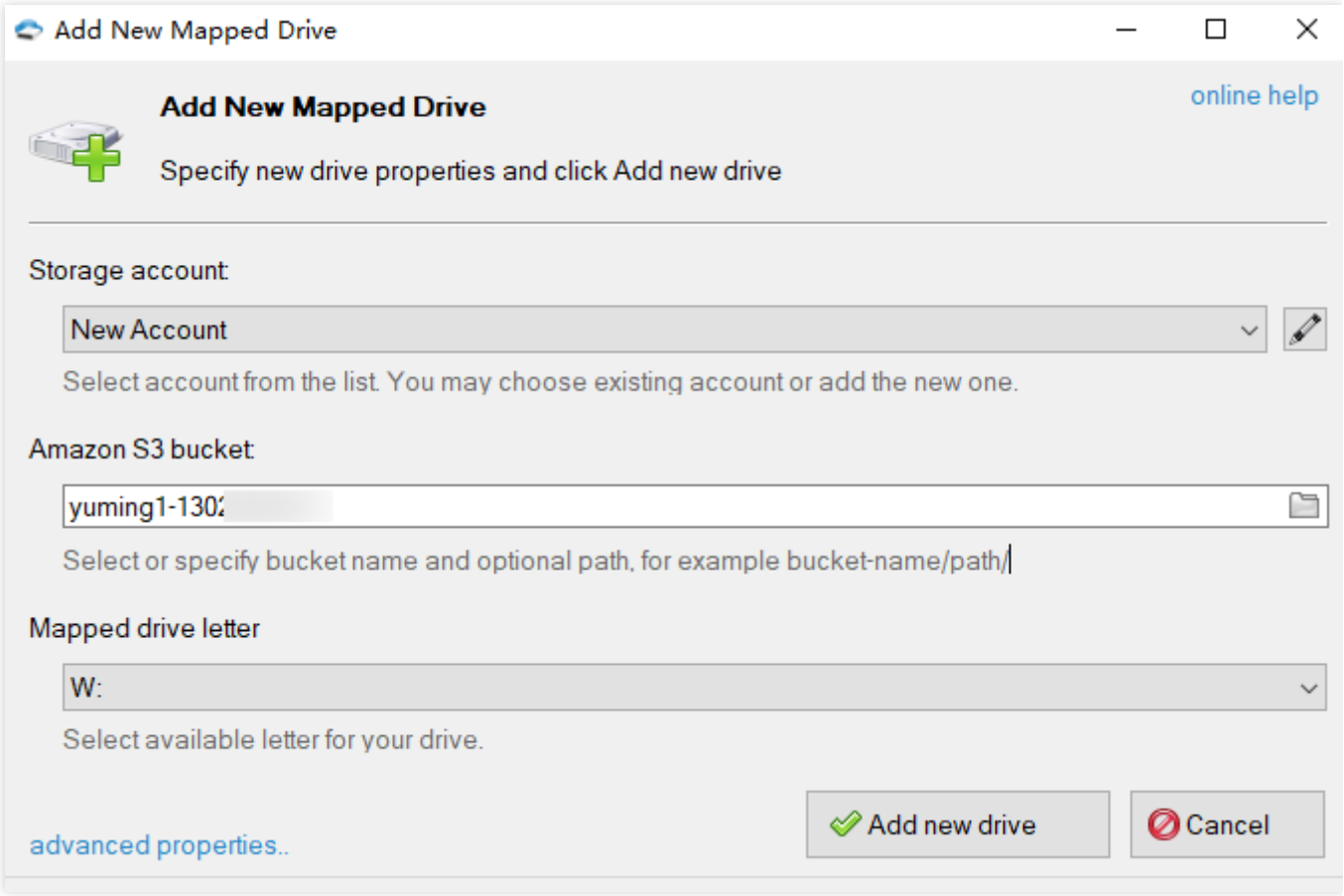
REST Endpoint：填写存储桶所在的地域，例如存储桶位于广州地域，则填 cos.ap-guangzhou.myqcloud.com。

Access Key ID：填写 SecretId。可在 API 密钥管理 页面中创建和获取。

Secret Access Key：填写 SecretKey。

3. 单击 **Add new account**。

4. 在 TntDrive 界面，单击 **Add New Mapped Drives**，创建一个 Mapped Drives。



主要参数信息如下：

Amazon S3 Bucket：输入存储桶路径，或选择存储桶名称。可单击右侧按钮选择存储桶。该处展示的是步骤2设置的广州地域下的存储桶。（一个存储桶独立映射为一个磁盘）。

Mapped drives letter：设置磁盘的盘符名称，请不要与本地的 C、D、E 盘等重复。

5. 确认以上信息单击 **Add new drive**。

6. 在本地计算机的 **我的电脑** 中，即可找到该磁盘。如果想把所有的存储桶都映射到 Windows 服务器中，请重复以上步骤。

使用 PicGo+Typora+COS 搭建图床服务

最近更新时间：2024-01-06 10:54:03

简介

图床服务提供图片存储、图片加工处理、图片全网分发等功能，为全球无数的博客网站和社区论坛提供了后端图片服务支撑。开发者们可以使用腾讯云**对象存储（Cloud Object Storage，COS）**搭建图床服务，COS 是腾讯云提供的一种存储海量文件的分布式存储服务，提供了更丰富的功能、更优越的性能、更高的可靠性保障。

COS 用于图床场景的优势有：

低成本：存储单价低，按量付费，用多少算多少。

不限速：上传下载不限速，不再长时间等待 loading，访问质量也更好。

高可用：有高等级的 SLA 可用性保障，存储的数据有高达99.9999999999%的持久性保障。

容量无限：文件分布式存储，支持海量文件，容量按需使用。

实践场景

场景1：新增图片使用 COS 搭建图床服务

本场景使用到以下工具：

PicGo：一款支持多种云存储配置、快捷生成图片链接的工具。

Typora：一款轻量级 Markdown 编辑器，支持多种输出格式，支持将本地图片一键上传至图床。

操作步骤

1. 安装 PicGo 并设置腾讯云 COS 服务相关参数。

说明

本次实践使用的是 PicGo 2.3.1版本，其他版本的配置过程可能存在一定差异，请注意相应调整。

在 PicGo 官网 下载和安装 PicGo 后，在图床设置里找到**腾讯云 COS**，并配置以下相关参数项：

COS 版本：选择 COS v5。

设定 SecretId：开发者拥有的项目身份识别 ID，用于身份认证，可在 API 密钥管理 页面中创建和获取。

设定 SecretKey：开发者拥有的项目身份密钥，可在 API 密钥管理 页面获取。

设定 Bucket：存储桶，COS 中用于存储数据的容器。有关存储桶的进一步说明，请参见 存储桶概述 文档。

设定 AppId：开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源，可在 API 密钥管理 页面获取。

设定存储区域：存储桶所属地域信息，枚举值可参见 可用地域 文档，例如 ap-beijing、ap-hongkong、eu-frankfurt 等。

设定存储路径：图片存放到 COS 存储桶中的路径。

设定自定义域名：可选，若您为上方的存储空间配置了自定义源站域名，则可填写。相关介绍可参见 开启自定义源站域名。
设定网址后缀：通过在网址后缀添加 COS 数据处理参数实现图片压缩、裁剪、格式转换等操作，相关介绍可参见 图片处理。
2. 设置 typora（可选）。
说明
如果您的编辑需求不是 Markdown 场景，可以忽略此步骤，仅使用上一步安装的 PicGo 作为图床工具。
设置指引如下：
1. 在 typora 的偏好设置的**图像**中，进行如下配置：
在**插入图片时**，选择**上传图片**。
在**上传服务设定**，选择 **PicGo(app)**，并设置刚才安装的 PicGo.exe 位置。
2. 重启 typora，使设置生效。
3. 进入 typora 编辑器区域，直接拖放或粘贴图片，即可上传图片并自动替换为 COS 文件链接。（如果粘贴后没有自动替换为 COS 链接，可以检查 PicGo 中的 server 设置是否已打开）。

场景2：将原图床仓库图片快速迁移到腾讯云 COS

以某图床服务举例，您可以找到本地图床文件夹，或从线上下载完整文件夹，并将文件夹中所有图片转存到 COS 存储桶。最后再统一替换链接域名即可恢复网站。

操作步骤

步骤1：下载原图床服务的图片

登录原图床网站页面，下载此前已上传的图片文件夹。

步骤2：创建 COS 存储桶并设置防盗链

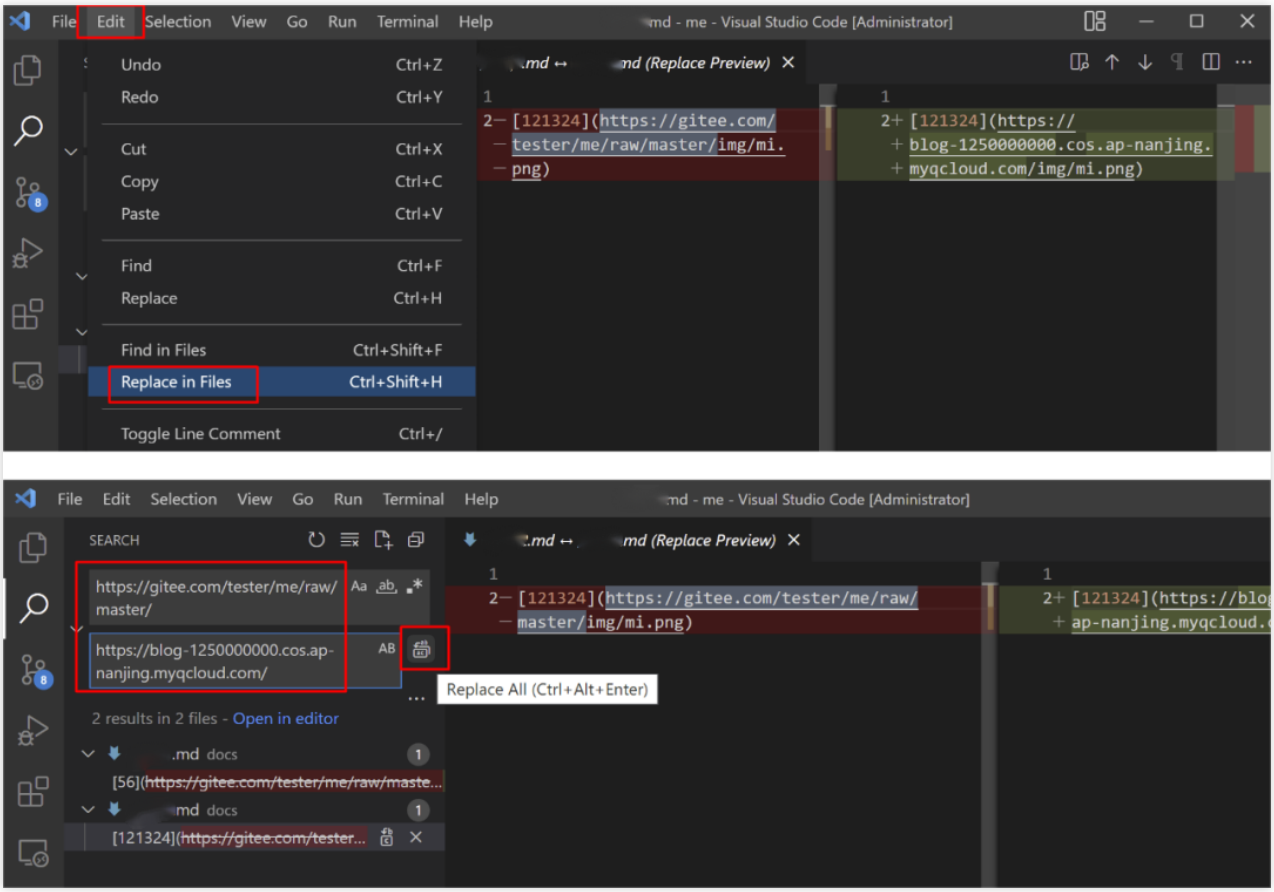
1. 注册腾讯云账号，创建一个访问权限为**公有读私有写**的存储桶，操作指引请参见 创建存储桶。
2. 创建存储桶后，在存储桶里打开防盗链设置，避免图片被盗刷，操作指引请参见 设置防盗链。

步骤3：上传文件夹到存储桶

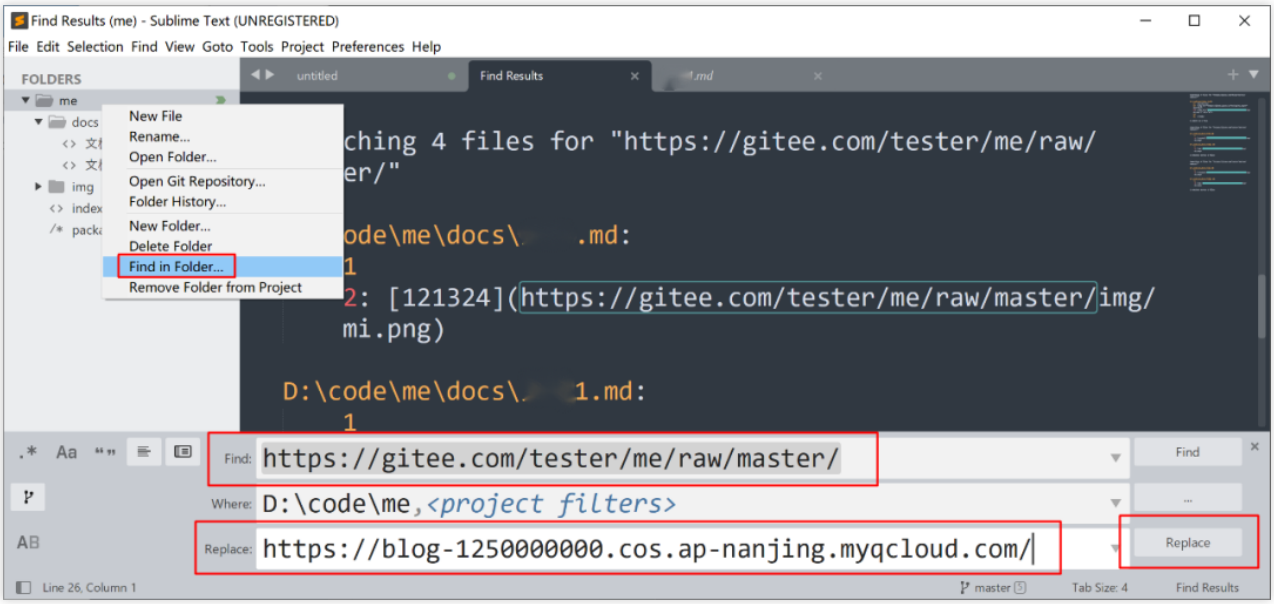
在刚才已创建的存储桶里，单击上传文件夹，将刚才准备好的图片文件夹，上传到 COS 存储桶。
说明
如果图片数量较多，还可以使用 COSBrowser 客户端 快速上传图片。

步骤4：全局替换链接域名

在 COS 控制台存储桶概览页，复制存储桶默认域名（也可以绑定自定义 CDN 加速域名）。使用常用代码编辑器，对项目全局搜索替换失效链接前缀为 COS 存储桶默认域名。
说明
关于默认域名的说明，请参见 地域和访问域名。
vscode 搜索替换示例：



sublime text 搜索替换示例：



通过 CloudBerry Explorer 管理 COS 资源

最近更新时间：2024-01-06 10:54:03

简介

CloudBerry Explorer 是一款可用于管理对象存储（Cloud Object Storage，COS）的客户端工具。通过 CloudBerry Explorer 可实现将 COS 挂载在 Windows 等操作系统上，方便用户访问、移动和管理 COS 文件。

支持系统

支持 Windows、macOS 系统。

下载地址

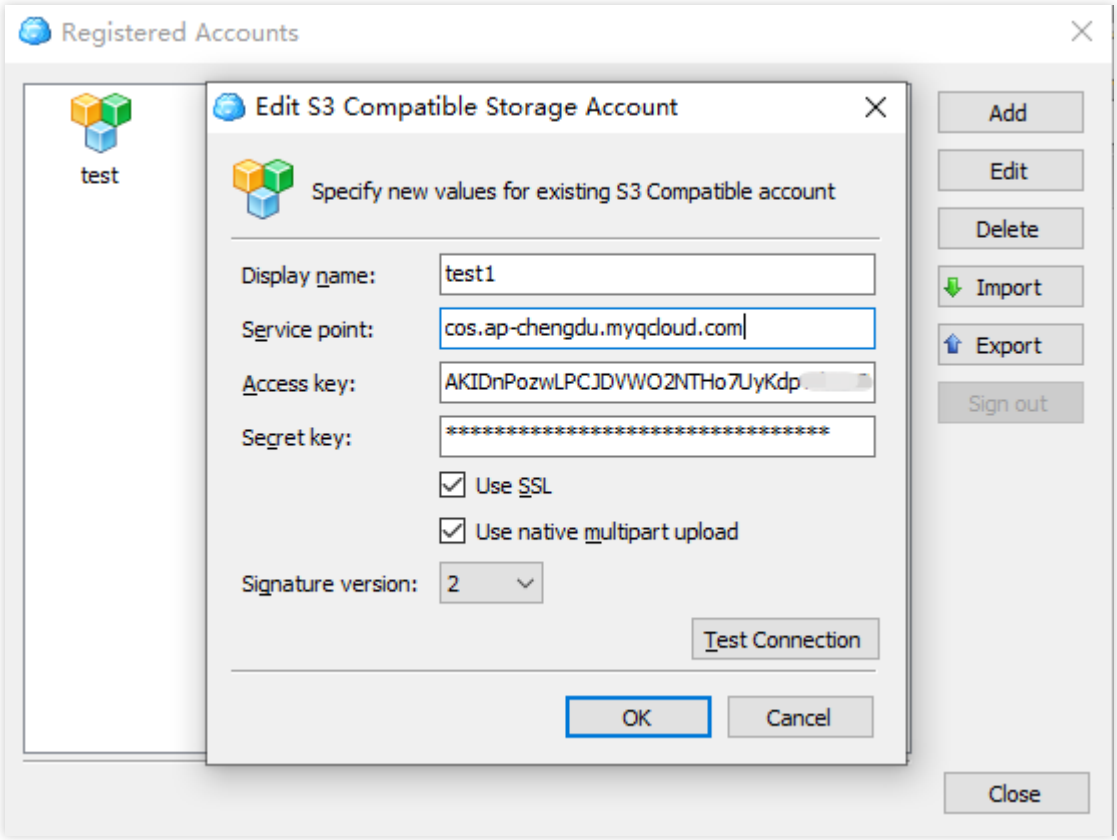
前往 CloudBerry 官方下载。

安装和配置

注意

以下配置步骤以 CloudBerry Explorer Windows v6.3版本为例，其他版本的配置过程可能存在一定差异，请注意相应调整。

1. 双击安装包，按照提示完成安装。
2. 打开工具，选择并双击 S3 Compatible。
3. 在弹窗中配置以下信息，单击 Test Connection 显示连接成功即可。



配置项说明如下：

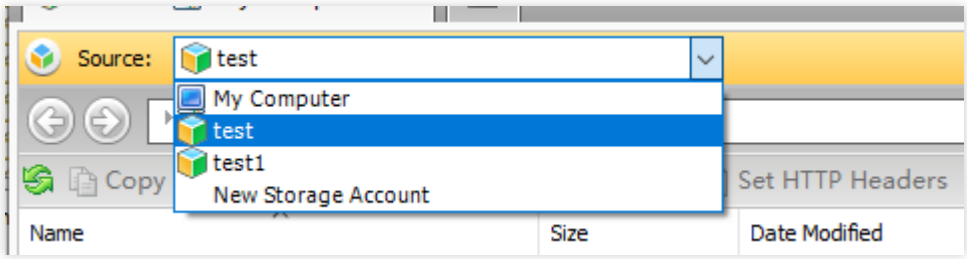
Display name：输入自定义用户名。

Service point：格式为 `cos.<Region>.myqcloud.com`，例如访问成都地域的存储桶，则输入 `cos.ap-chengdu.myqcloud.com`，适用的地域简称（region）请参见 地域和访问域名。

Access key：输入访问密钥信息 SecretId，可前往 云 API 密钥 中创建和获取。

Secret key：输入访问密钥信息 Secretkey，可前往 云 API 密钥 中创建和获取。

4. 账户信息添加完成后，在 Source 中选择之前设置的用户名，可查看该用户名下的存储桶列表，即表示已配置完成。



管理 COS 文件

查询存储桶列表

在 Source 中选择之前设置的用户名，可查看该用户名下的存储桶列表。

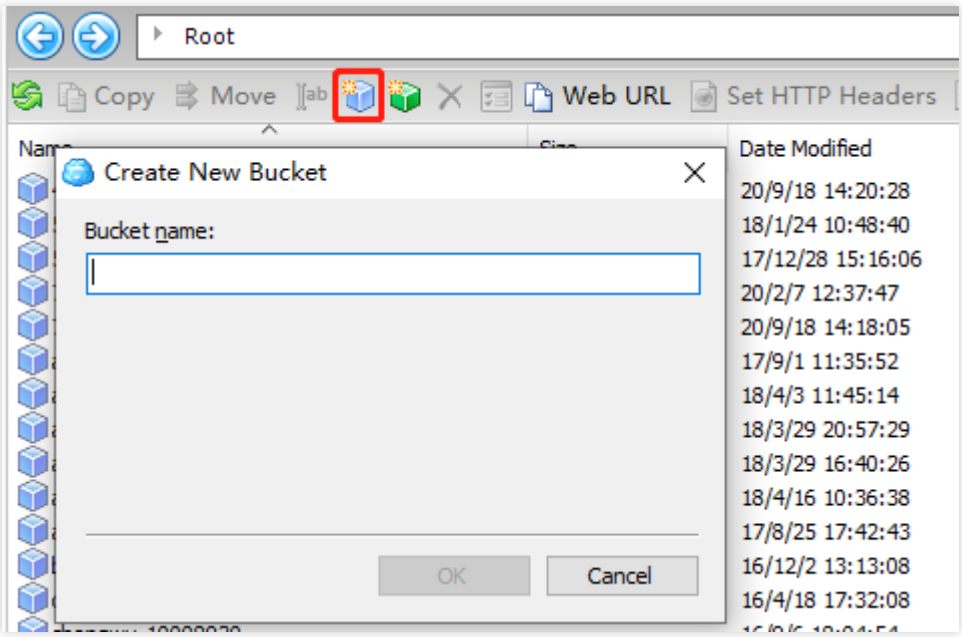
注意

只能查看 Service point 配置的地域所对应的存储桶。如需查看其它地域的存储桶，可单击 **File > Edit Accounts**，并选择用户名，修改 Service point 参数为其他地域即可。

创建存储桶

单击图中的图标，在弹窗中输入完整的存储桶名称，例如 examplebucket-1250000000。输入无误后，单击 **OK** 即可创建完成。

关于存储桶的命名规范，请参见 存储桶命名规范。

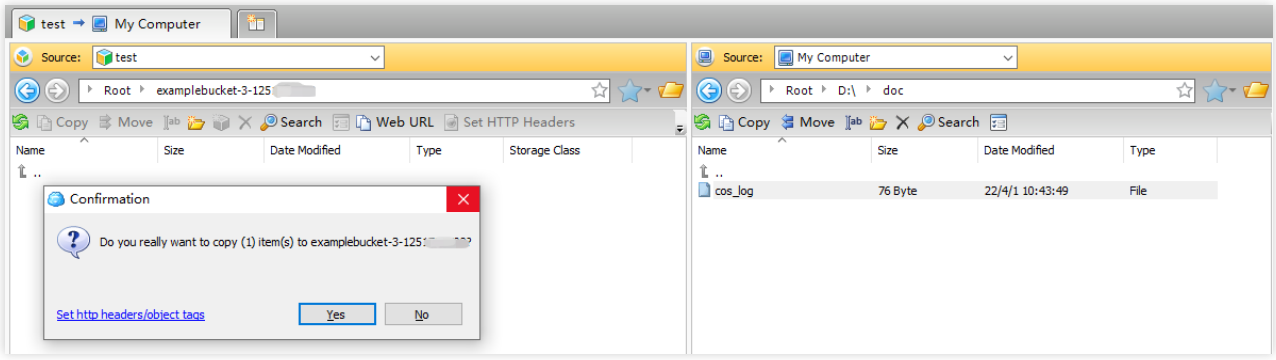


删除存储桶

在存储桶列表中选择需删除存储桶，右键单击 **Delete** 即可删除。

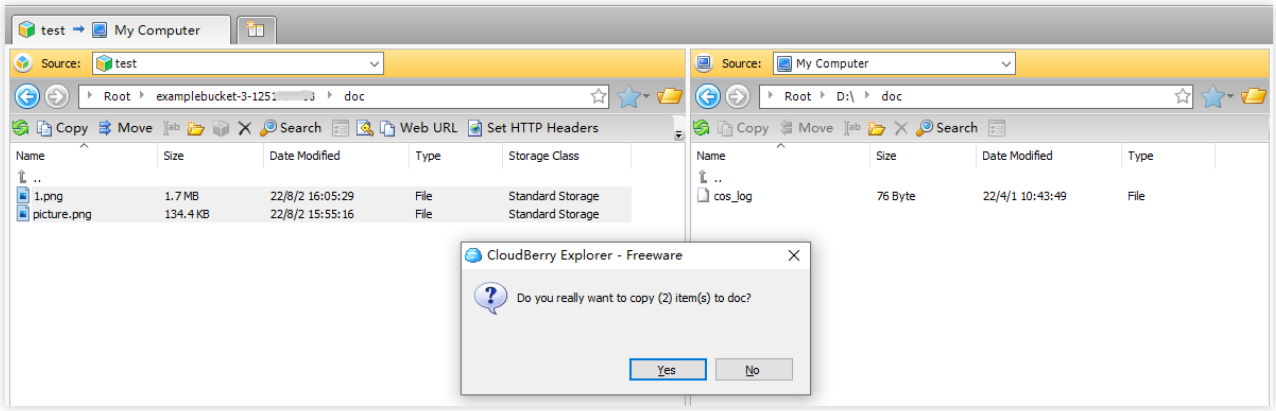
上传对象

在存储桶列表中选择需上传对象的存储桶或路径，然后在本地计算机中选择需上传的对象，并将其拖拽到左侧窗口中，即可完成上传操作。



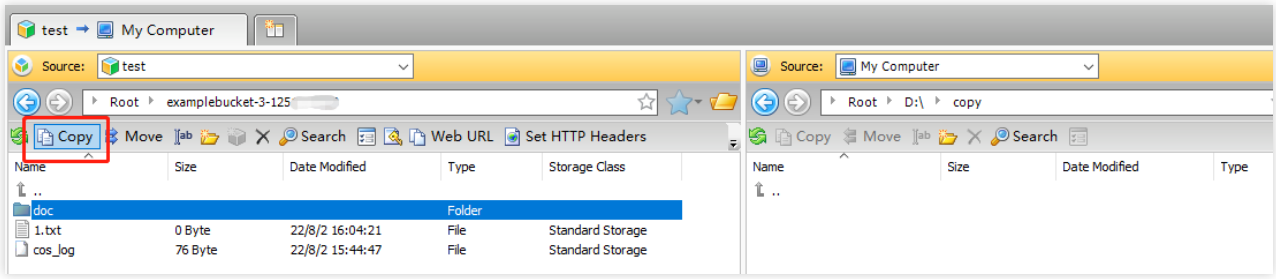
下载对象

在左侧窗口中选择需下载的对象，然后将对象拖拽到右侧本地计算机的文件夹中，即可完成下载操作。



复制对象

在工具的右侧窗口中，选择对象被复制后的目标路径，然后在左侧窗口中选择需复制的对象，右键单击 **Copy**，确认弹窗信息，即可完成对象复制操作。



重命名对象

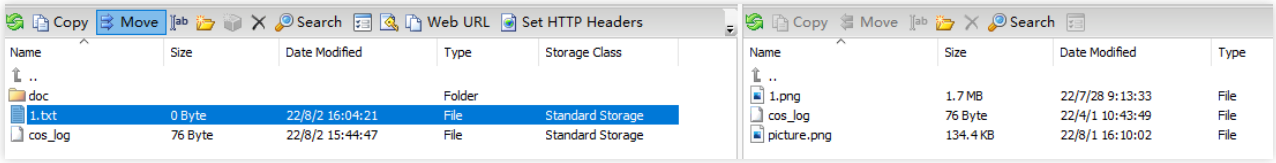
在存储桶中找到需重命名的对象，并右键单击 **Rename**，输入新名称即可。

删除对象

在存储桶中找到需删除的对象，并右键单击 **Delete**，即可删除对象。

移动对象

在工具的右侧窗口中，选择对象被移动后的目标路径，然后在左侧窗口中选择需移动的对象，并右键单击 **Move**，确认弹窗信息，即可完成对象移动操作。



其他功能

除了以上功能，CloudBerry Explorer 还支持其他功能，例如设置对象 ACL、查看对象元数据、自定义 Headers、获取对象 URL 等。用户可根据实际需求进行操作。