云点播

专业版

腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

内网上传

专业版
云点播专业版简介

最近更新时间：2025-01-15 17:28:12

什么是云点播专业版?

云点播专业版（Video on Demand Professional，VOD Pro）是基于腾讯云音视频能力提供的高品质媒体服务，在标准版的基础上，支持以对象存储模式管理内容，并使用 EdgeOne 进行内容分发，为直播、教育、电商、金融、游戏等行业提供更优的用户体验。

产品功能

专业版将逐渐包含标准版的所有功能，并在此基础上增加对象存储管理模式和使用 EdgeOne 进行内容分发。云点播专业版的已上线功能请参见下方表格：

分类	功能	说明
媒体上传	多端上传	提供客户端上传（移动端、Web 端）、服务端上传（提供多语言 SDK）、控制台上传等多种上传方式。
	就近上传	可感知上传者所处的位置，分配离上传者最近的存储中心进行上传，减少传输距离，提高上传速度。同时提高稳定性，保证了成功率。
	客户端上传加速	利用腾讯云全球部署的加速网络，智能选择最优接入点和最优链路，显著提升媒体的上传速度和成功率。同时可以减少网络拥塞，降低延时，改善弱网环境下的上传质量。
存储管理	存储策略	支持根据文件修改时间、文件前缀等条件设置降冷策略，基于策略对文件执行存储降冷，以节约存储成本。
	多类型存储	支持标准存储、低频存储、归档存储和深度归档存储。可以根据文件不同的存储和使用场景，选择适合的存储方式。
	目录管理	支持创建文件目录，可以使用目录管理文件内容。
	文件管理	支持上传、下载、删除、搜索和分享文件。
分发播放	加速分发播放	使用 EdgeOne 进行内容分发，基于腾讯云遍布全球的加速节点，提供动静态数据加速、跨国加速、智能路由优化等加速能力。
	防盗链	使用 EdgeOne 访问控制提供防盗链能力，通过配置鉴权规则进行 URL 访问校验，可以有效防止站点资源被恶意盗刷。

常用工具	迁移工具	支持将第三方数据源轻松迁移至腾讯云点播专业版。

产品优势

EdgeOne 加速

- 免回源流量费用：与 EdgeOne 深度联动，通过 EdgeOne 回源云点播专业版，无须支付回源的流量费用，极大降低内容加速成本。
- 多重防盗刷保护：使用 EdgeOne 为用户提供防盗刷服务，通过 Bot 防护、流量封顶、盗刷情报库等多种防盗刷能力，全方位保护您的流量安全。

媒体上传

- 就近上传：支持在应用内创建多个存储桶，开启就近上传后，可以感知上传者所处位置，自动分配最近的存储桶上传，从而显著提升上传的速度和稳定性。
- 上传加速：通过调度优化、全球多存储园区覆盖优化、链路补充、传输优化、协议优化等多种上传加速措施，实现上传成功率达99.5%+（考虑了弱网环境和大文件上传），上传质量领先业界。

媒体存储

- 安全稳定：支持媒体文件跨多架构、多设备备份存储，提供异地容灾和用户资源隔离，数据可靠性可达99.9999999999%。
- 便捷易用：提供简洁易用的控制台，兼容 S3 对象存储协议，支持多个云厂商的迁移工具，极大降低存储迁移和接入成本。

适用场景

场景	说明
在线教育	在线教育场景下，通常存在大量的音视频教学资源，这些资源大多数是由平台上的教师录制并上传的。云点播专业版为在线教育客户提供上传加速、多端播放、版权保护、伪直播、内容审核等多种能力，帮助客户搭建更稳定安全的在线教育平台。
电商 APP	电商平台为了更好的展示商品，一般会制作推广的商品图片、视频，同时消费者也会上传商品体验反馈的视频和图片用于分享，或者商品评论来供其他消费者参考。云点播专业版为电商客户提供 UGC 加速上传、高画质低码率、多清晰度智能切换、海量存储等能力，帮助电商客户快速搭建稳定可用的电商平台。
社交短视频	在社交短视频场景下，云点播专业版提供强大的上传、存储、分发能力，同时可配套使用腾讯云短视频 SDK、腾讯云播放器等优质产品，帮助客户聚焦业务本身，快速实现高品质短视频应用。
网盘相册	云点播专业版提供大规模存储容量，可支持图片、音视频、压缩包等多种文件格式的存储，同时支持多种数据加密及安全容灾能力，为网盘相册客户提供更具性价比的数据存储

	服务。
音视频直播	音视频直播场景下，可以将直播内容实时录制存储到云点播专业版，同时提供时移回看、直播剪辑、伪直播、截图封面、智能降冷等多种直播配套能力，帮助客户搭建用户体验更优的音视频直播平台。

快速入门

最近更新时间：2025-08-05 17:39:52

本文将引导您如何将本地文件上传至云点播专业版，并获取视频的播放地址，以帮助您快速了解如何使用专业版服务。

准备工作

1. 已有可用的 EdgeOne 加速站点。EdgeOne 加速站点配置请参见 从零开始快速接入 EdgeOne 。
2. 本地有可以上传的文件，您也可以选择下载专业版示例文件 腾讯云.mp4 。

步骤一：开通云点播

1. 注册 腾讯云账号 ，并完成 实名认证 。
2. 进入 云点播控制台 ，单击**开通服务**。

> **说明：**
> 若已开通云点播服务，请直接进入下一步骤。

步骤二：创建专业版应用

1. 登录 云点播控制台 ，单击左侧导航栏**应用管理**，进入应用列表页。
2. 单击应用类型旁的提示按钮，在弹出的信息提示中，单击**立即开通专业版**。

3. 单击**创建应用**，在创建应用的弹窗中，输入应用名称，并选择应用类型为**专业版**，单击**确定**完成专业版应用的创建。

步骤三：上传文件

1. 登录 云点播控制台，单击左侧导航栏**应用管理**，进入应用列表页。
2. 找到您步骤一中创建的专业版应用，单击应用名称进入应用，默认进入存储页面。
3. 在存储页面中，单击 **default** 存储桶名称，进入存储桶。
4. 单击**上传文件**，选择本地目标文件或准备工作中下载的“腾讯云.mp4”视频，即可将文件上传至专业版应用。

步骤四：设置应用分发域名

使用预置域名分发

1. 登录 云点播控制台，单击左侧导航栏**应用管理**，进入应用列表页。
2. 找到您步骤一中创建的专业版应用，单击应用名称进入应用，然后单击**关联站点**。

3. 在关联站点的弹窗中，选择您准备工作中配置好的 EdgeOne 站点，单击**确定**，默认生成一个预置的加速域名。

使用自定义域名分发

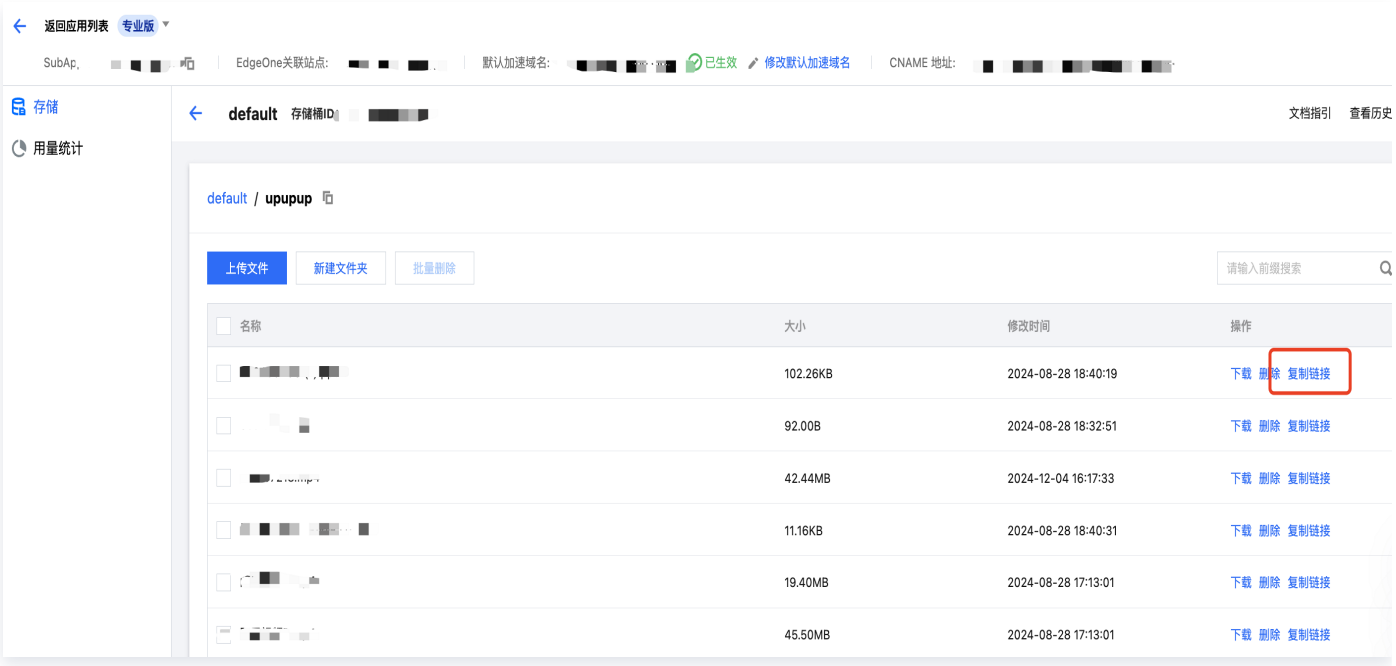
1. 在 EdgeOne 控制台 中，单击**添加域名**，源站配置选择“云点播”，同时选择步骤一中创建的专业版应用。回源范围选择“应用内的所有文件”。

2. 在 EdgeOne 中完成域名添加后，如果是 CNAME 模式接入或者 DNSPod 托管接入，域名创建完成后，EdgeOne 将为该域名分配一个 CNAME 地址，您需要完成配置 CNAME 才能使该域名的安全加速生效。配置方式请参见 修改 CNAME 解析 。如果是 NS 接入，则不需要该步骤。
3. 在 EdgeOne 中完成域名添加后，您需要进一步开启 HTTPS。配置方式请参见 通过 EdgeOne 免费证书快速实现 HTTPS 访问 。

4. 在 EdgeOne 中添加域名完成后，回到云点播控制台页面，单击顶部**修改默认域名**，选择已经配置回源到本应用域名，单击**确定**即可完成修改。

步骤五：获取文件播放地址

1. 单击目标文件或前面步骤上传的腾讯云.mp4视频后的**复制链接**。



2. 在 Web 浏览器 URL 地址栏输入刚刚已复制的 URL 地址，按下回车键，即可播放该视频。

控制台指南
存储管理
创建存储桶

最近更新时间：2025-01-10 10:59:12

专业版支持创建多个地域存储桶，从而实现视频内容就近上传及分发加速，本文将介绍如何创建存储桶。

适用场景

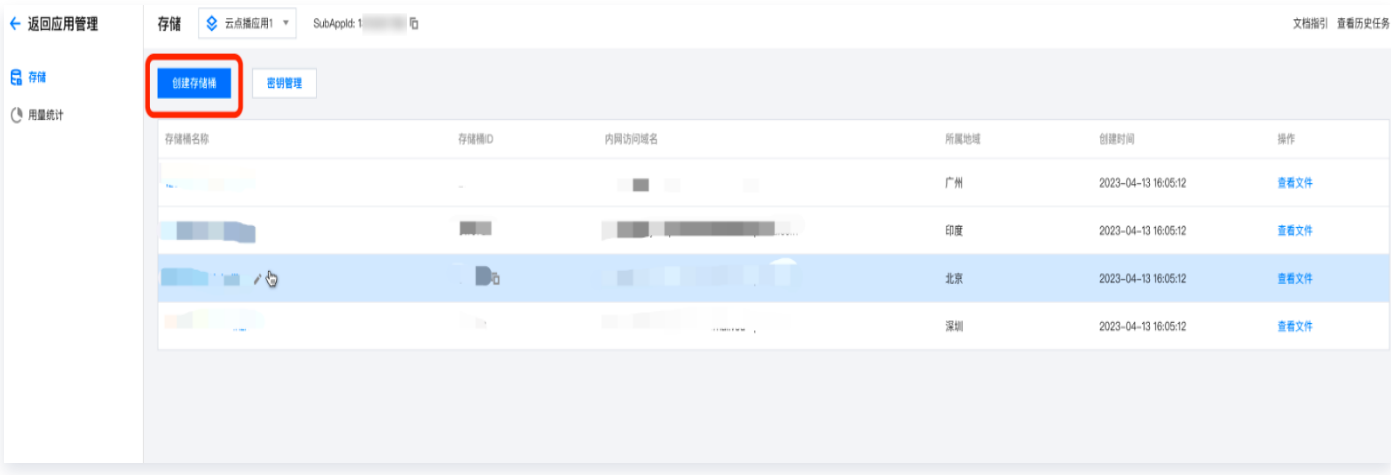
若您的用户广泛分布在多个地域，为了更好的上传体验及视频播放效果，推荐您创建多地域存储桶，以便实现就近上传及就近回源。

前提条件

- 开通云点播服务，详细请参见 购买指引 。
- 已 创建云点播专业版应用 。专业版应用目前仅对部分客户开放，如需要创建专业版应用，请联系商务。

操作步骤

1. 登录 云点播控制台 ，单击左侧导航栏**应用管理**，进入应用列表页。
2. 找到需要创建存储桶的专业版应用，单击应用名称进入应用管理页。
3. 选择左侧导航栏的**存储**，单击**创建存储桶**。



4. 在创建存储桶弹窗中，输入存储桶名称及地域信息，单击**保存**即可完成新建存储桶。

存储策略

最近更新时间：2025-06-10 16:04:01

本文档将引导您创建存储策略，通过存储策略来定期对文件转换存储类型或删除文件，来降低存储成本。

适用场景

如果您需要按照文件的修改时间或前缀，来定期将文件转为低频、归档、深度归档存储，或直接删除文件，您可以创建合适的存储策略，来自动执行对应的操作，从而降低存储成本。

前提条件

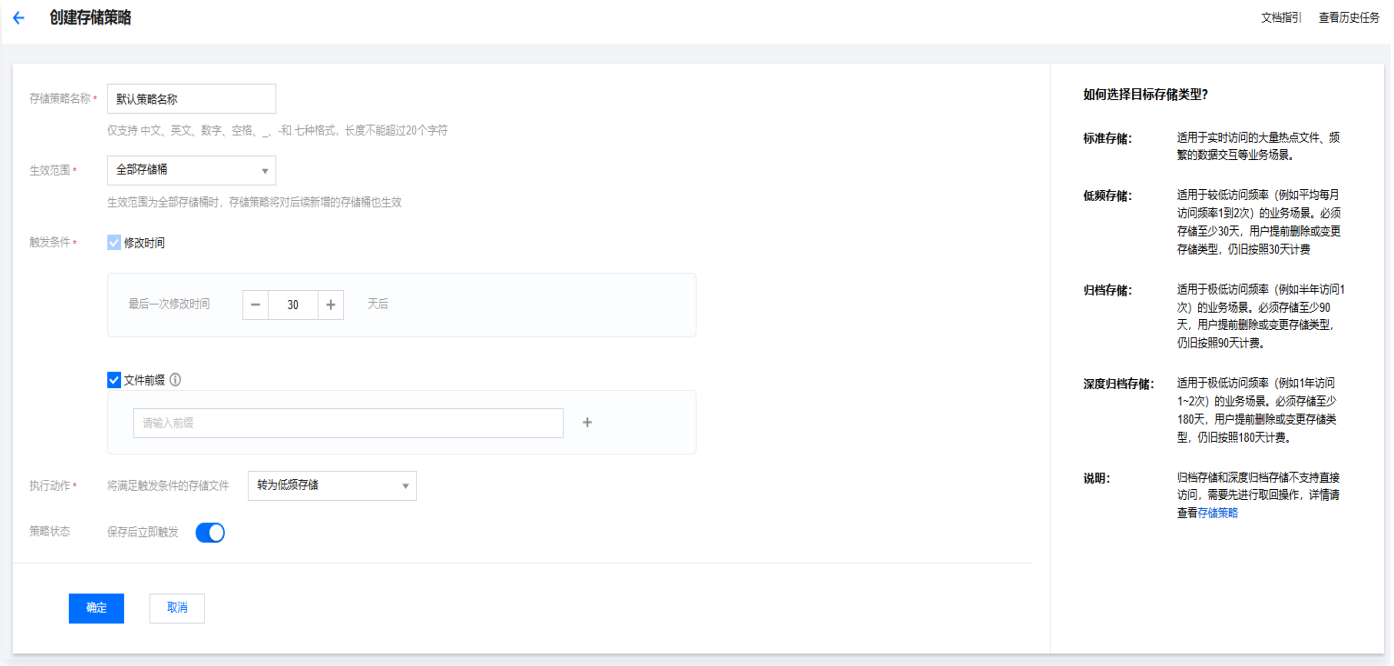
- 开通云点播服务，详细请参见 购买指引 。
- 已 创建云点播专业版应用 。专业版应用目前仅对部分客户开放，如需要创建专业版应用，请联系商务。

创建存储策略

1. 登录 云点播控制台 ，单击左侧导航栏**应用管理**，进入应用列表页。
2. 找到需要配置存储策略的专业版应用，单击应用名称进入应用存储页面。单击存储策略，进入存储策略页面。

3. 单击**创建存储策略**，进入策略新建页面。

4. 在创建存储策略配置页面，您可以根据业务情况配置合适的策略：



- 存储策略名称：存储策略名称长度不超过20个字符，仅支持中文、英文、数字、空格、_、-和. 七种格式。
- 生效范围：默认应用下所有存储桶生效。您可以通过下拉选择指定存储桶生效。如果您选择了全部存储桶生效，存储策略也会对新增存储桶生效。
- 触发条件：提供修改时间和文件前缀2个触发条件，其中修改时间最多选择3650天，文件前缀最多添加20条。
- 执行动作：默认为转为低频存储。您可以通过下拉选择转为归档存储、转为深度归档存储或删除文件。

5. 策略详情配置完成后，单击**确定**完成存储策略的创建，存储策略将在次日凌晨0点正式生效。

> **说明：**
> - 生效时间是在策略开启后（即策略状态为“已启用”）的次日凌晨0点生效。
> - 若对低频、归档和深度归档类型的媒体文件配置了降冷策略（例如：设置低频存储类型的文件沉降到归档存储或深度归档存储），则在执行降冷策略时需将数据先取回至标准存储，再进行降冷。因此会产生数据的取回费用，费用详情参见 数据取回 。

修改存储类型

最近更新时间：2025-02-12 14:51:44

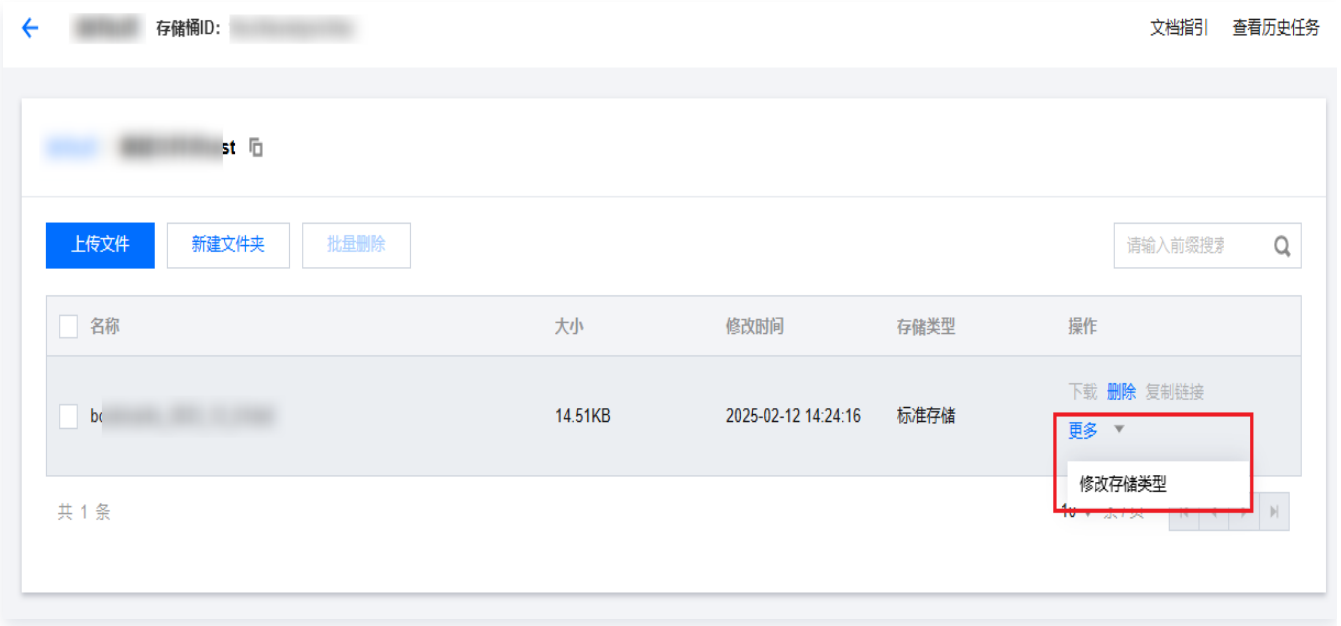
本文将介绍如何修改专业版文件的存储类型。

前提条件

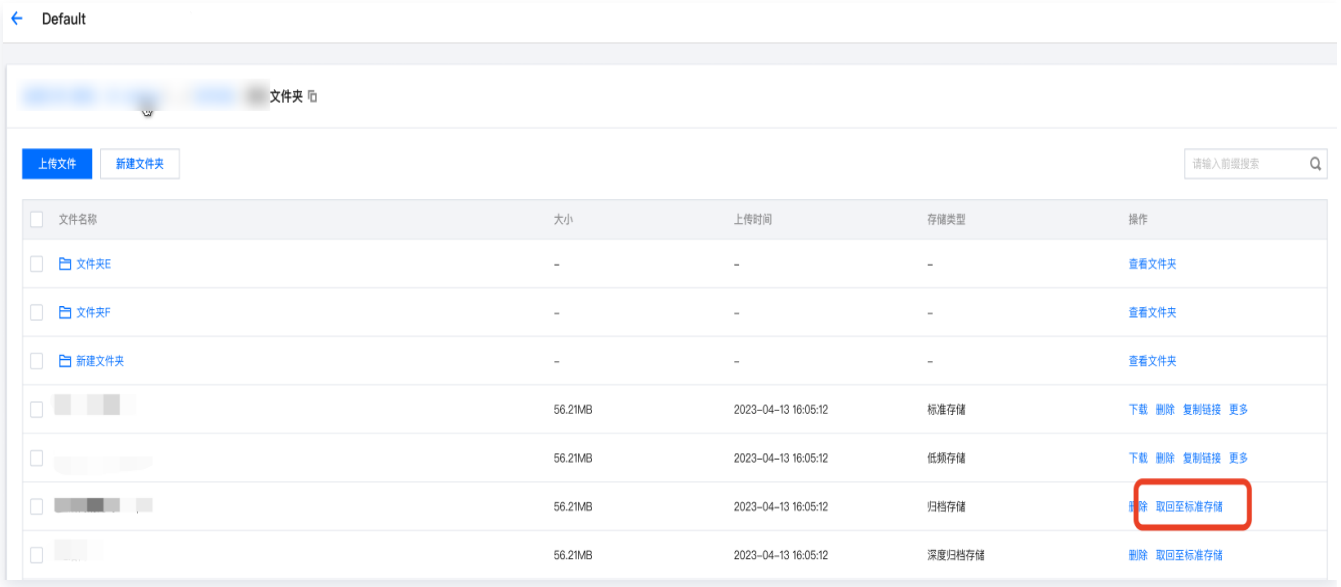
- 开通云点播服务，详情请参见 购买指引 。
- 已 创建云点播专业版应用 。专业版应用目前仅对部分客户开放，如需要创建专业版应用，请联系商务。
- 已上传文件，详情请参见 上传文件 。

操作步骤

1. 登录 云点播控制台 ，单击左侧导航栏**应用管理**，进入应用列表页。
2. 选择您需要操作的专业版应用，进入目标存储桶找到需要修改存储类型的文件。
3. 您可以根据您的诉求修改文件存储类型：
   - 对文件降冷：如果您需要将标准存储或低频存储的文件修改为归档或深度归档文件，您可以依次单击**更多** > **修改存储类型**，在修改弹窗中选择目标存储类型，单击**确定**即可完成修改。



- **临时取回文件：如果您需要短暂的将归档存储或深度归档存储文件取回，您可以单击取回至标准存储，在取回的弹窗中选择取回的有效期以及取回的模式。**



- **永久取回文件：如果您需要将归档存储或深度归档存储文件永久取回，您需要先将文件取回至标准存储，然后再依次单击更多 > 修改存储类型，选择标准存储或低频存储，即可将归档存储或深度归档存储的文件永久取回至标准存储或低频存储。**

数据取回能力

您可以直接通过数据取回能力，将归档存储/深度归档存储类型的文件变更为标准存储。对归档存储/深度归档存储文件变更存储类型和取回操作有多种模式。不同模式的最终效果一致，区别在于速度和费用（即取回费用，费用详情请参见 按量计费-数据取回 ）。

解冻模式	归档存储取回耗时	深度归档存储取回耗时
极速模式	5分钟	不支持
标准模式	5小时	24小时
批量模式	12小时	48小时

说明：
由于媒体被标识为已解冻的时间点会晚于实际解冻的时间点，因此媒体的解冻副本的可用时长会短于预期想要的时长。为了确保解冻后副本有足够的可用时长，建议在解冻深度归档存储时加多1天的有效期。

文件管理
上传文件

最近更新时间：2025-01-08 16:03:52

本文将引导您将文件通过控制台上传到专业版应用。

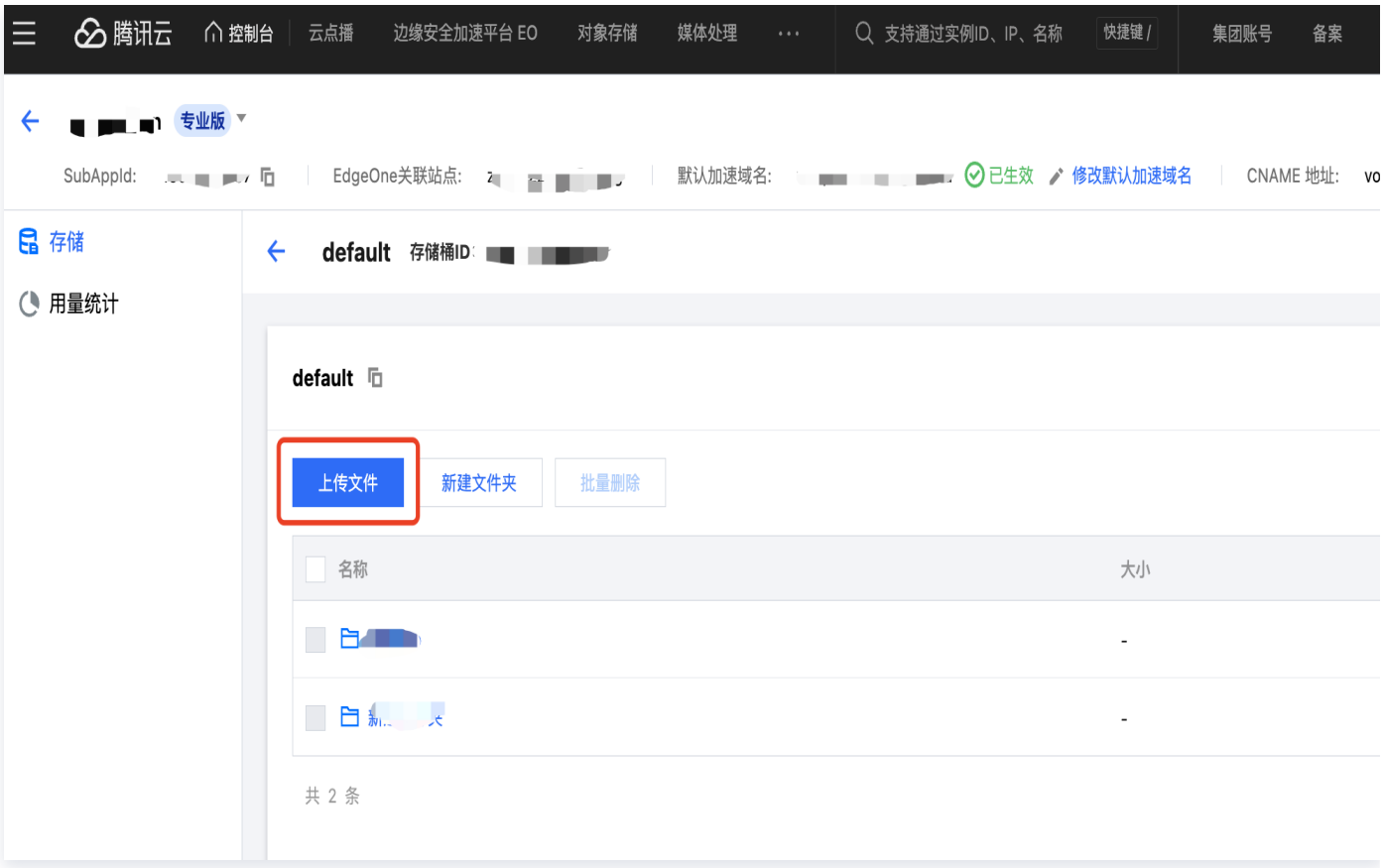
准备工作

已开通云点播，且已创建专业版应用。创建专业版应用请参考 快速入门 。

> **说明：**
> 控制台最大能够上传512GB文件。

操作步骤

1. 登录 云点播控制台 ，单击左侧导航栏**应用管理**，进入应用列表页。
2. 找到需要上传文件的应用，单击应用名称进入应用存储页面。
3. 在应用存储页面，选择需要上传文件的存储桶，单击存储桶名称进入存储桶的文件列表页面。
4. 在存储桶的文件列表页面，单击**上传文件**。



5. 在弹出的窗口中，选择单个或多个本地需要上传的文件既可开始上传文件。
6. 您可以在文件列表右上角**历史任务**中查看正在上传的文件进度，以及历史已经上传成功的文件列表。

下载文件

最近更新时间：2025-03-24 09:53:42

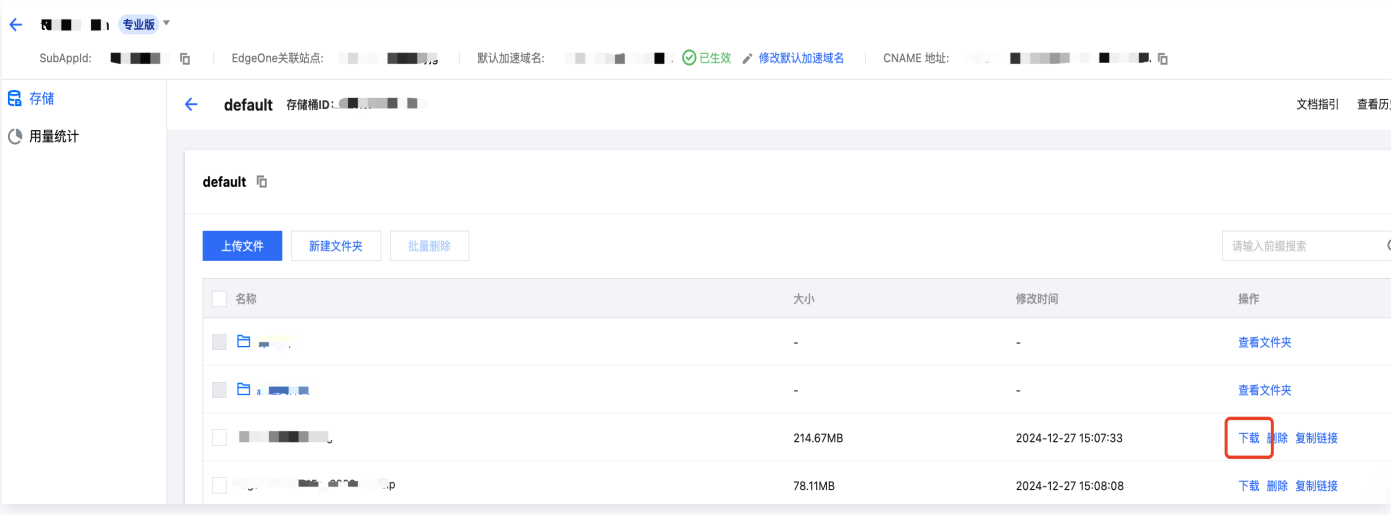
本文将引导您如何通过控制台下载专业版中的文件。

准备工作

- 已开通云点播，且已创建专业版应用。创建专业版应用请参见 快速入门 。
- 已上传文件。上传文件请参见 上传文件 。
- 已配置好默认加速域名。配置默认加速域名请参见 快速入门 。

操作步骤

1. 登录 云点播控制台 ，单击左侧导航栏**应用管理**，进入应用列表页。
2. 找到需要下载文件的应用，单击应用名称进入应用存储页面。
3. 在应用存储页面，选择需要下载文件的存储桶，单击存储桶名称进入存储桶的文件列表页面。
4. 在存储桶的文件列表页面，找到需要下载的文件，单击操作中的**下载**既可。



删除文件

最近更新时间：2025-04-25 09:53:01

本文将引导您如何通过控制台删除专业版中的文件。

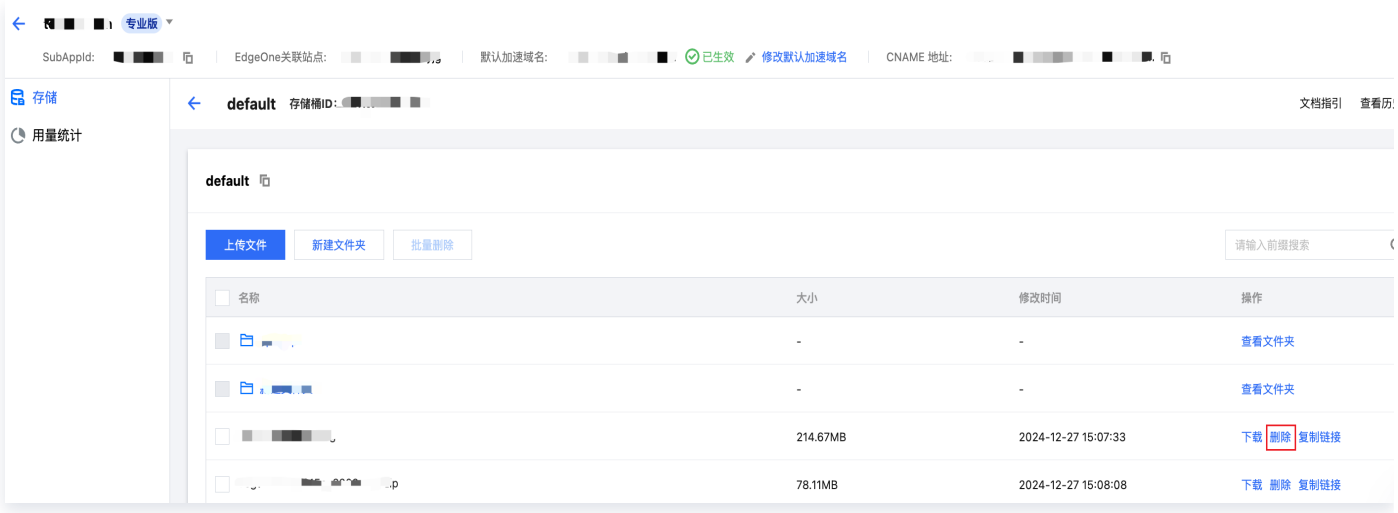
准备工作

- 已开通云点播，且已创建专业版应用。创建专业版应用请参见 快速入门 。
- 已上传文件。上传文件请参见 上传文件 。

操作步骤

删除单个文件

1. 登录 云点播控制台 ，单击左侧导航栏**应用管理**，进入应用列表页。
2. 找到需要删除文件的应用，单击应用名称进入应用存储页面。
3. 在应用存储页面，选择需要删除文件的存储桶，单击存储桶名称进入存储桶的文件列表页面。
4. 在存储桶的文件列表页面，找到需要删除的文件，单击操作中的**删除**。



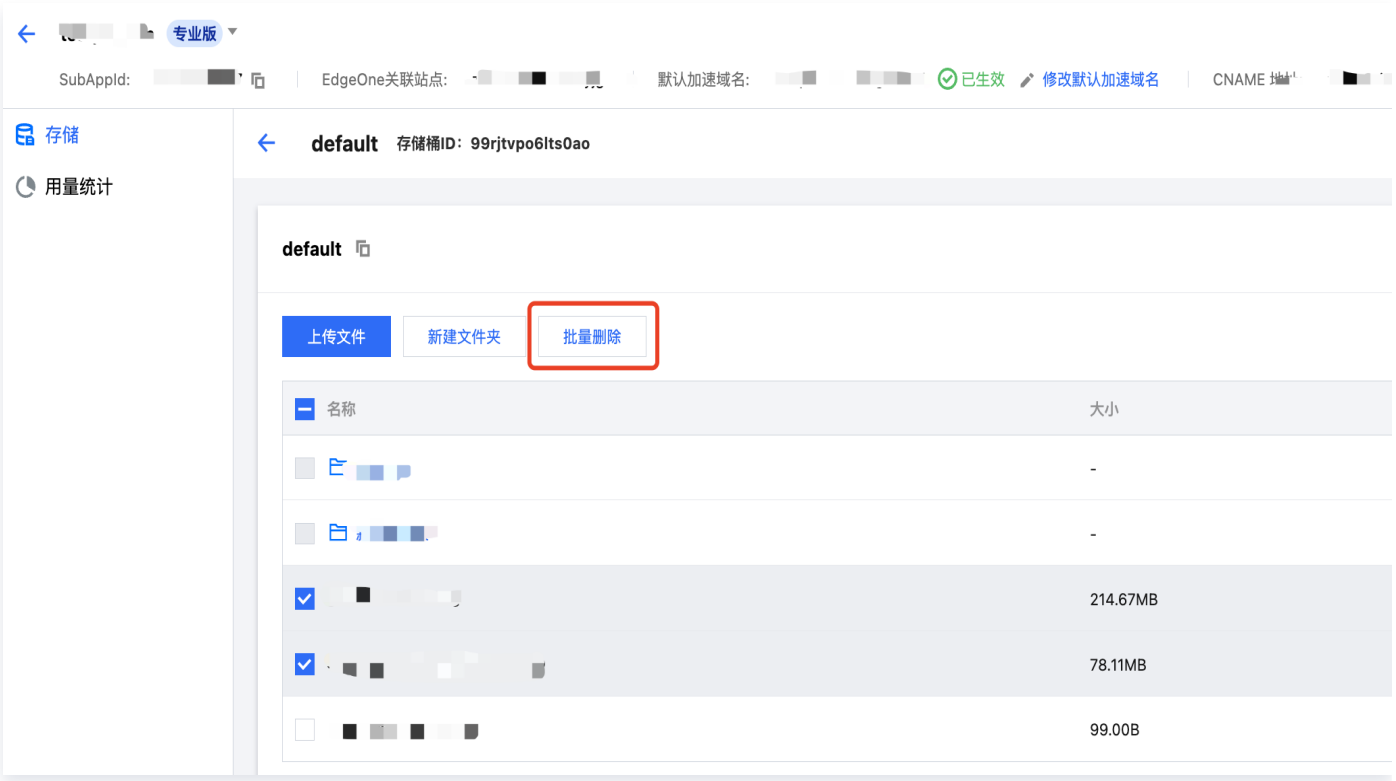
5. 在弹出的确认弹窗中，单击**确定**既可删除文件。

> **说明：**
> 文件将被从腾讯云彻底删除，无法恢复，也无法通过 CDN 节点访问该文件。

批量删除文件

1. 登录 云点播控制台 ，单击左侧导航栏**应用管理**，进入应用列表页。
2. 找到需要删除文件的应用，单击应用名称进入应用存储页面。
3. 在应用存储页面，选择需要删除文件的存储桶，单击存储桶名称进入存储桶的文件列表页面。

4. 在存储桶的文件列表页面，勾选需要删除的文件，单击文件列表上方的**批量删除**。



5. 在弹出的确认弹窗中，单击**确定**既可批量删除文件。

> **说明：**
> 文件将被从腾讯云彻底删除，无法恢复，也无法通过 CDN 节点访问该文件。

搜索文件

最近更新时间：2025-03-27 10:47:22

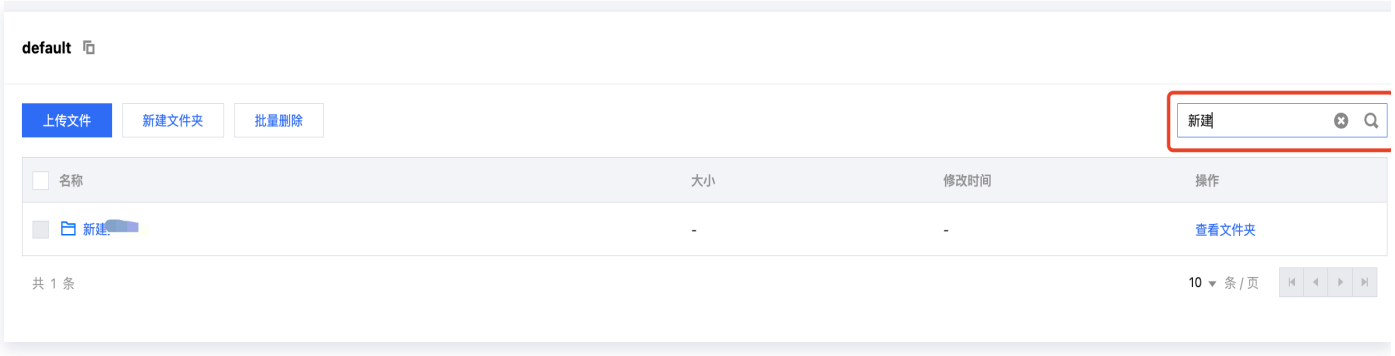
本文将引导您如何通过控制台搜索专业版中的文件。

准备工作

- 已开通云点播，且已创建专业版应用。创建专业版应用请参见 快速入门 。
- 已上传文件。上传文件请参见 上传文件 。

操作步骤

1. 登录 云点播控制台 ，单击左侧导航栏**应用管理**，进入应用列表页。
2. 找到需要搜索文件的应用，单击应用名称进入应用存储页面。
3. 在应用存储页面，选择需要搜索文件的存储桶，单击存储桶名称进入存储桶的文件列表页面。
4. 在存储桶的文件列表页面右上方搜索框中，输入文件前缀，按回车键既可查询具有相同前缀的文件列表。



5. 如需取消搜索，您可以清除搜索框中的内容，再次按回车键即可取消搜索展示全部文件。

分享文件

最近更新时间：2025-04-27 11:58:32

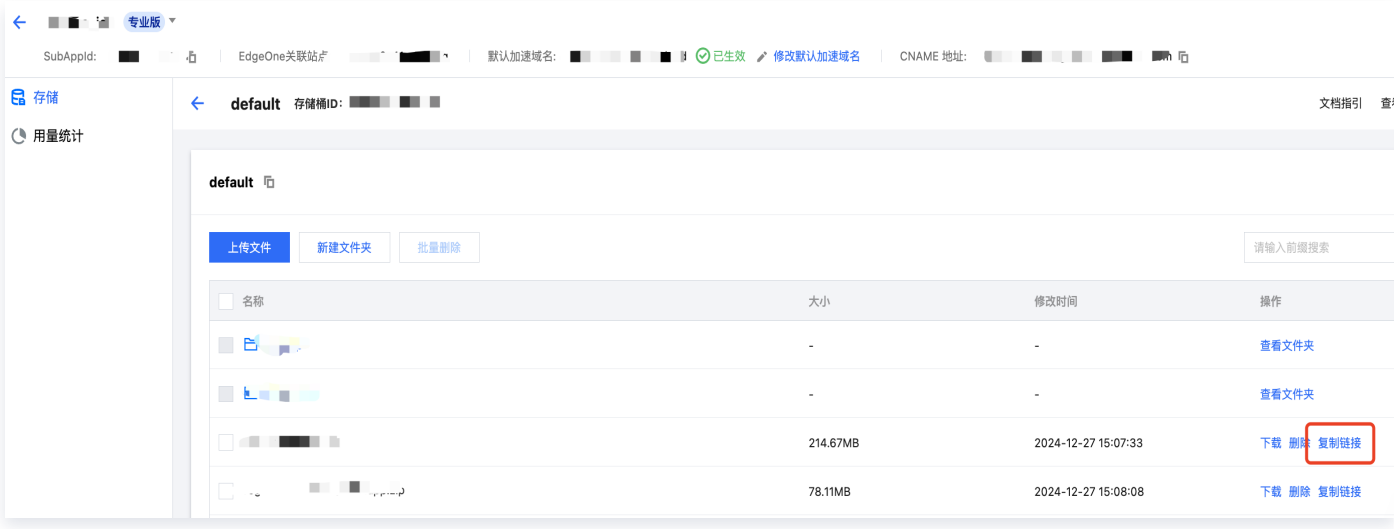
本文将引导您如何通过控制台分享专业版中的文件。

准备工作

- 已开通云点播，且已创建专业版应用。创建专业版应用请参见 快速入门 。
- 已上传文件。上传文件请参见 上传文件 。
- 已配置好默认加速域名。配置默认加速域名请参见 快速入门 。

操作步骤

1. 登录 云点播控制台 ，单击左侧导航栏**应用管理**，进入应用列表页。
2. 找到需要分享文件的应用，单击应用名称进入应用存储页面。
3. 在应用存储页面，选择需要分享文件的存储桶，单击存储桶名称进入存储桶的文件列表页面。
4. 在存储桶的文件列表页面，找到需要分享的文件，单击操作中的**复制链接**既可将文件复制后分享给其他用户。



创建文件夹

最近更新时间：2025-03-24 09:53:42

本文将指导您如何通过控制台创建文件夹，以便在专业版中对文件进行分类管理。

准备工作

已开通云点播，且已创建专业版应用。创建专业版应用请参见 快速入门 。

操作步骤

1. 登录 云点播控制台 ，单击左侧导航栏**应用管理**，进入应用列表页。
2. 找到需要创建文件夹的应用，单击应用名称进入应用存储页面。
3. 在应用存储页面，选择需要创建文件夹的存储桶，单击存储桶名称进入存储桶的文件列表页面。
4. 在存储桶的文件列表页面，单击**新建文件夹**。

5. 在弹出的弹窗中，输入文件夹名称，单击**确定**既可完成文件夹的新建。

   文件夹命名规则如下：
   - 可用数字、中英文和可见字符的组合，长度最大为100个字符。
   - 不允许文件夹名为空。

> **说明：**
> 文件夹不支持重名，同一路径下，同名文件夹只能存在一个。

密钥管理

最近更新时间：2025-01-08 16:03:52

本文将引导您如何创建专业版应用密钥，以便通过内网使用该密钥访问存储桶中的文件。

适用场景

如果您的服务部署在腾讯云服务器（Cloud Virtual Machine，CVM）上，您可以通过密钥和内网访问域名来访问文件。如果是同一地域内的访问，可以实现内网访问，此时上传和下载文件均产生内网流量，不会产生流量费用。

操作步骤

1. 登录 云点播控制台，单击左侧导航栏进入**应用管理**，进入应用列表页。
2. 转到需要管理密钥的应用，单击应用名称进入应用管理页。
3. 选择左侧导航栏的**存储**，单击**密钥管理**。

4. 在密钥管理页面中，单击**创建密钥**，确认保存密钥信息即可完成密钥的创建。

说明：

- 云点播专业版应用最多可创建2个密钥。
- 为降低密钥泄露风险，密钥仅在创建时提供 AccessKey 信息，后续不可再进行查询，请及时保存好密钥信息。

事件通知

最近更新时间：2025-06-24 10:37:31

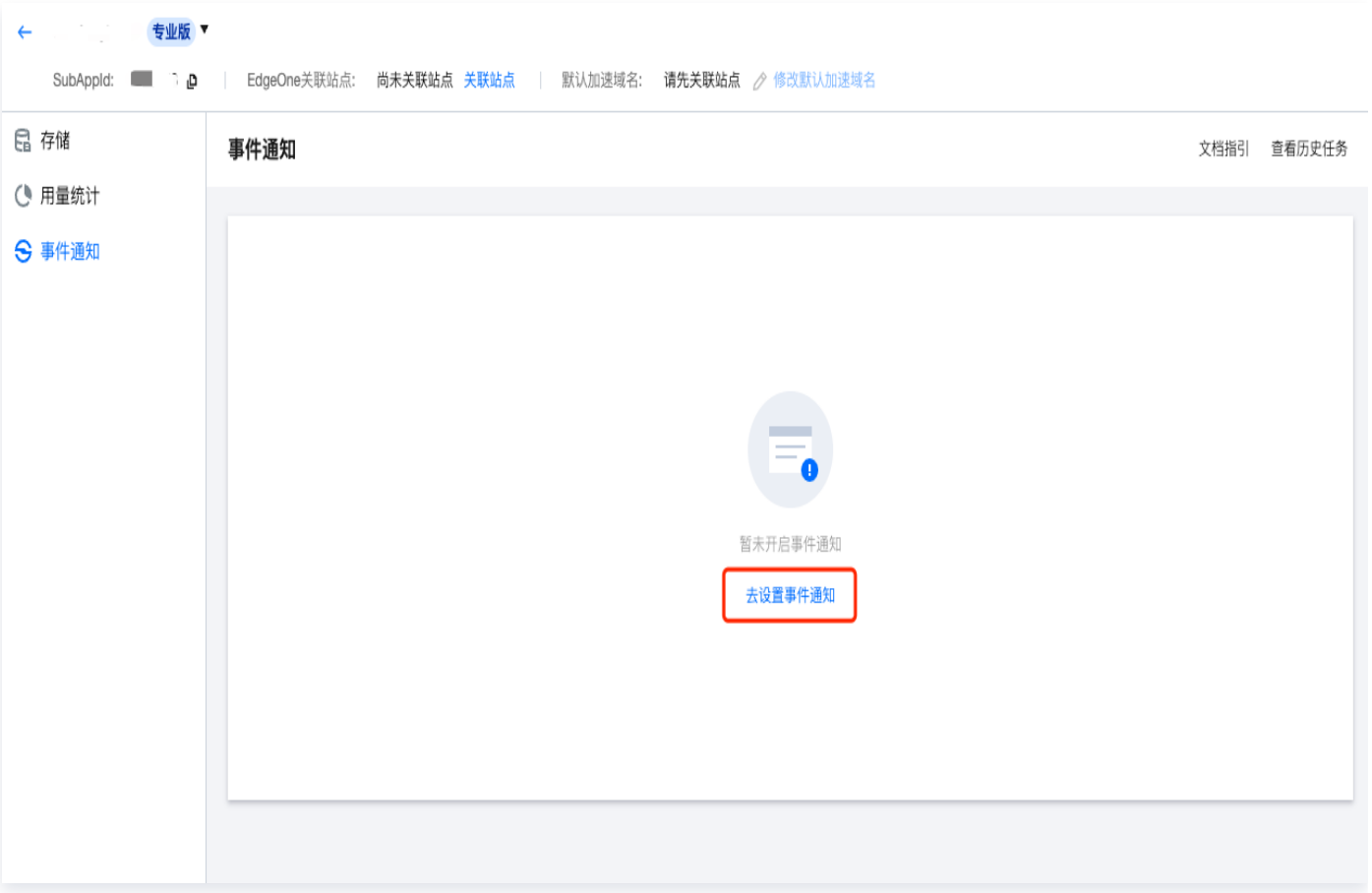
本文档将引导您如何启用专业版事件通知，以便在文件相关处理完成后及时收到通知。

适用场景

您可以根据实际需求设置事件通知 URL，在文件相关处理完成后，系统将向该 URL 发送 HTTP 请求，请求体中包含任务的结果状态。

操作步骤

1. 登录 云点播控制台 ，单击左侧导航栏进入**应用管理**，进入应用列表页。
2. 找到您需要设置的专业版应用，单击应用名称进入应用管理页。
3. 在左侧导航栏选择**事件通知**，单击**去设置事件通知**，进入“设置事件通知”页面。



4. 在设置页面中根据实际需求配置以下参数：
   - 事件通知方式：目前仅支持 HTTP 回调。
   - 回调 URL：根据实际需求设置接收回调的 App 后台地即可。
   - 通知事件：目前仅支持文件上传完成事件通知。

迁移工具
全量迁移工具
AWS S3国际站迁移

最近更新时间：2025-02-05 11:25:32

本文将引导您如何将 AWS S3的文件，通过云点播迁移工具，迁移到云点播专业版。

准备工作

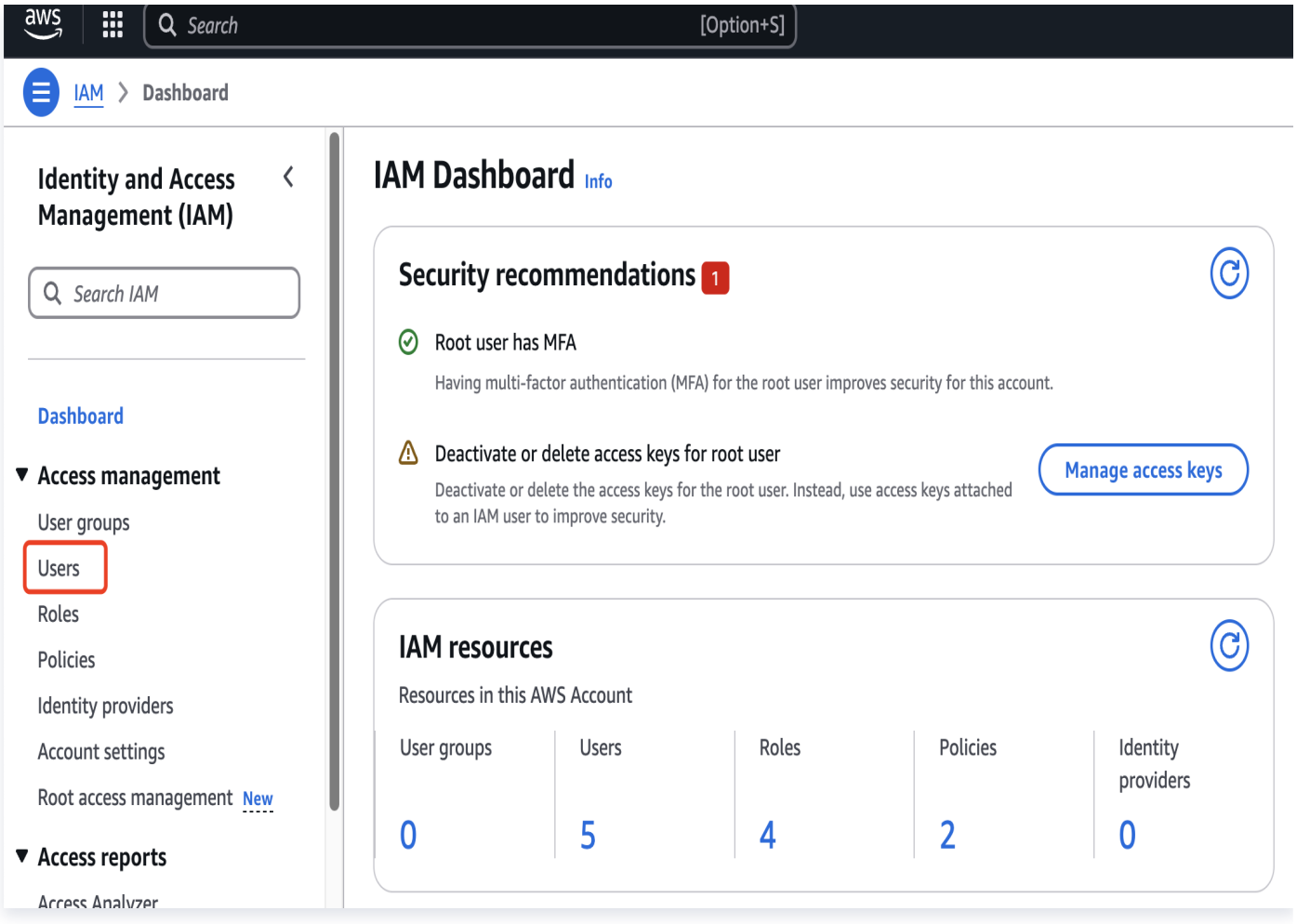
1. 已有可用的专业版应用，创建专业版应用请参考 快速入门 。
2. 已有可用的 AWS S3存储桶。

操作步骤

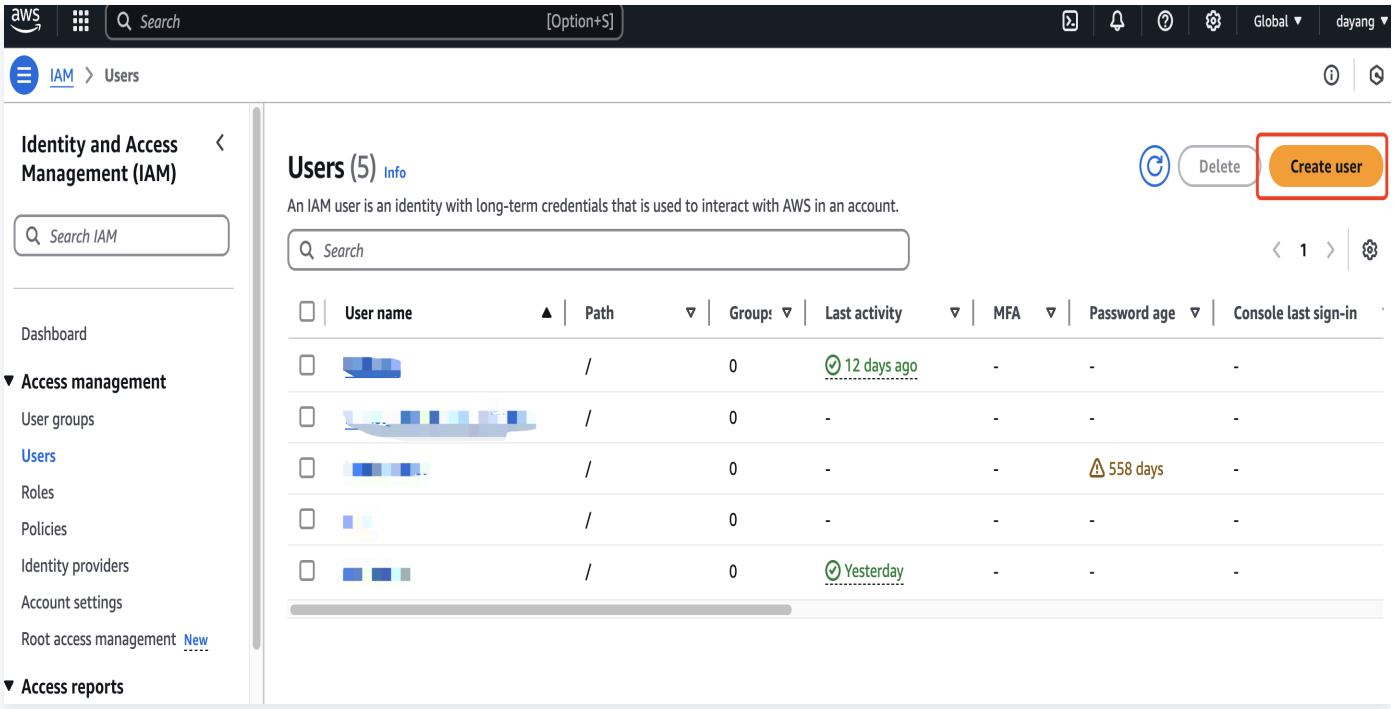
步骤一：创建 AWS IAM 账号并授予相关权限

> **说明：**
> 以下内容仅供参考，具体操作以 AWS S3实际展示及提供的服务内容为准。

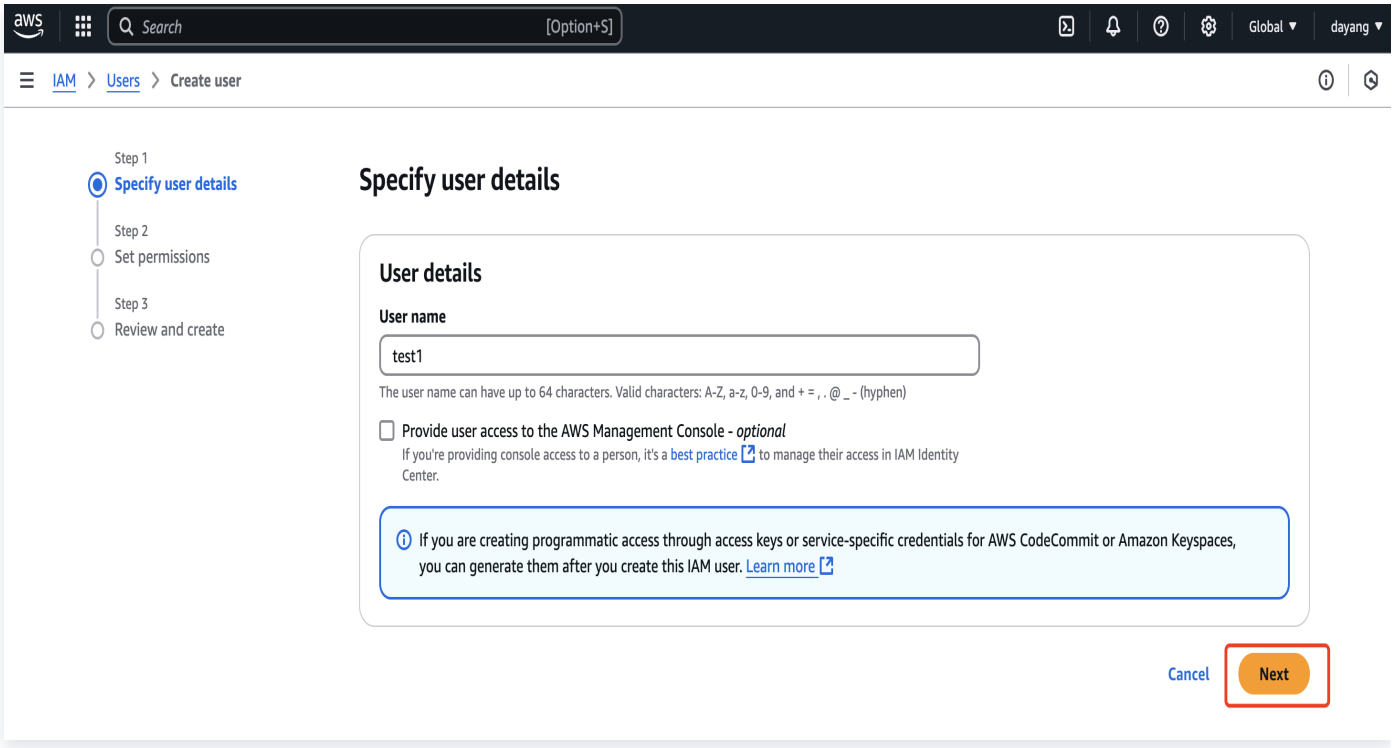
1. 登录 AWS 控制台，进入 Identity and Access Management (IAM) 界面，单击 **Users**，进入用户管理页面。



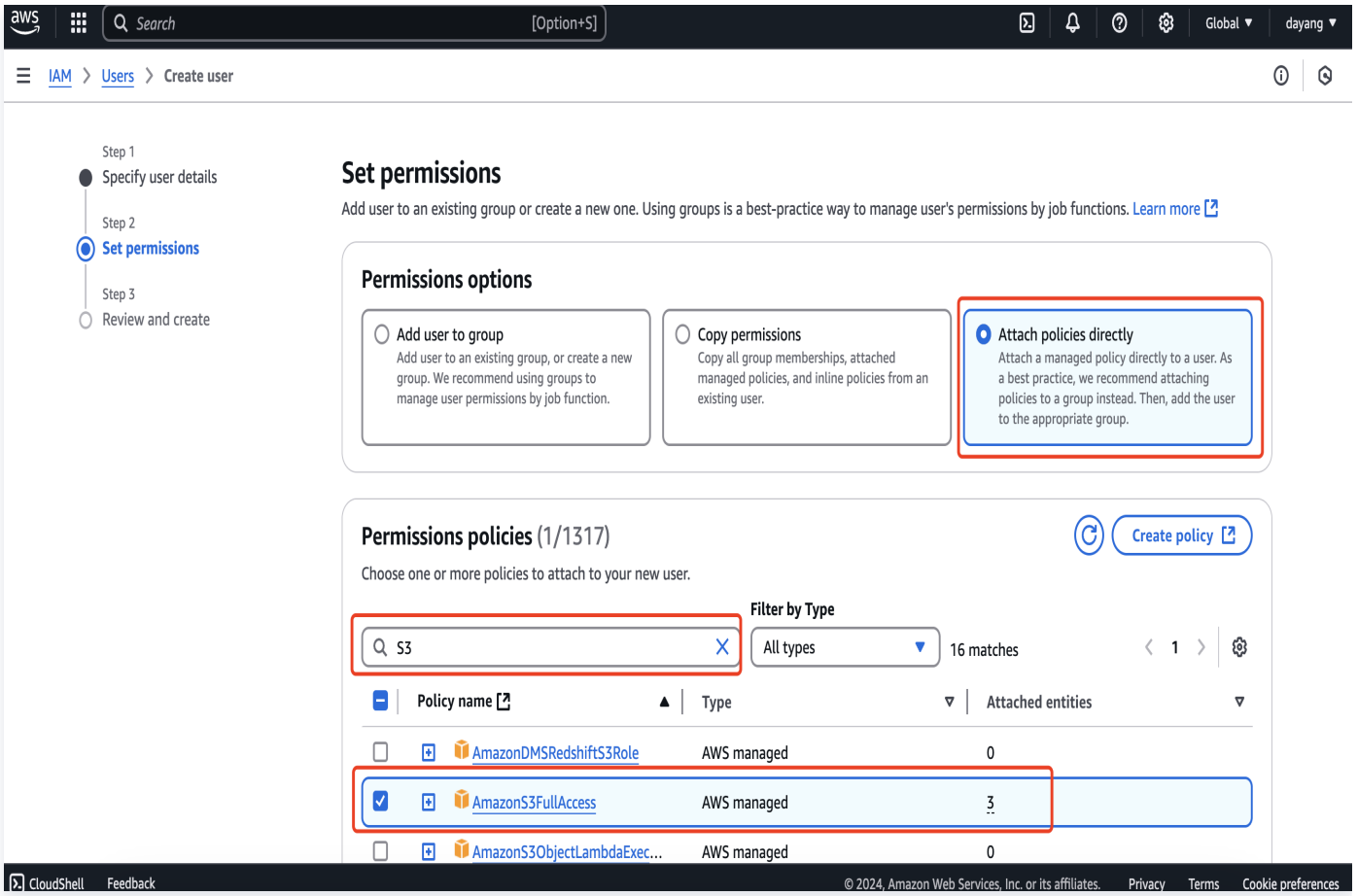
2. 进入用户管理页面后，单击 **Create user**。



3. 输入用户的名称，单击 **Next**。



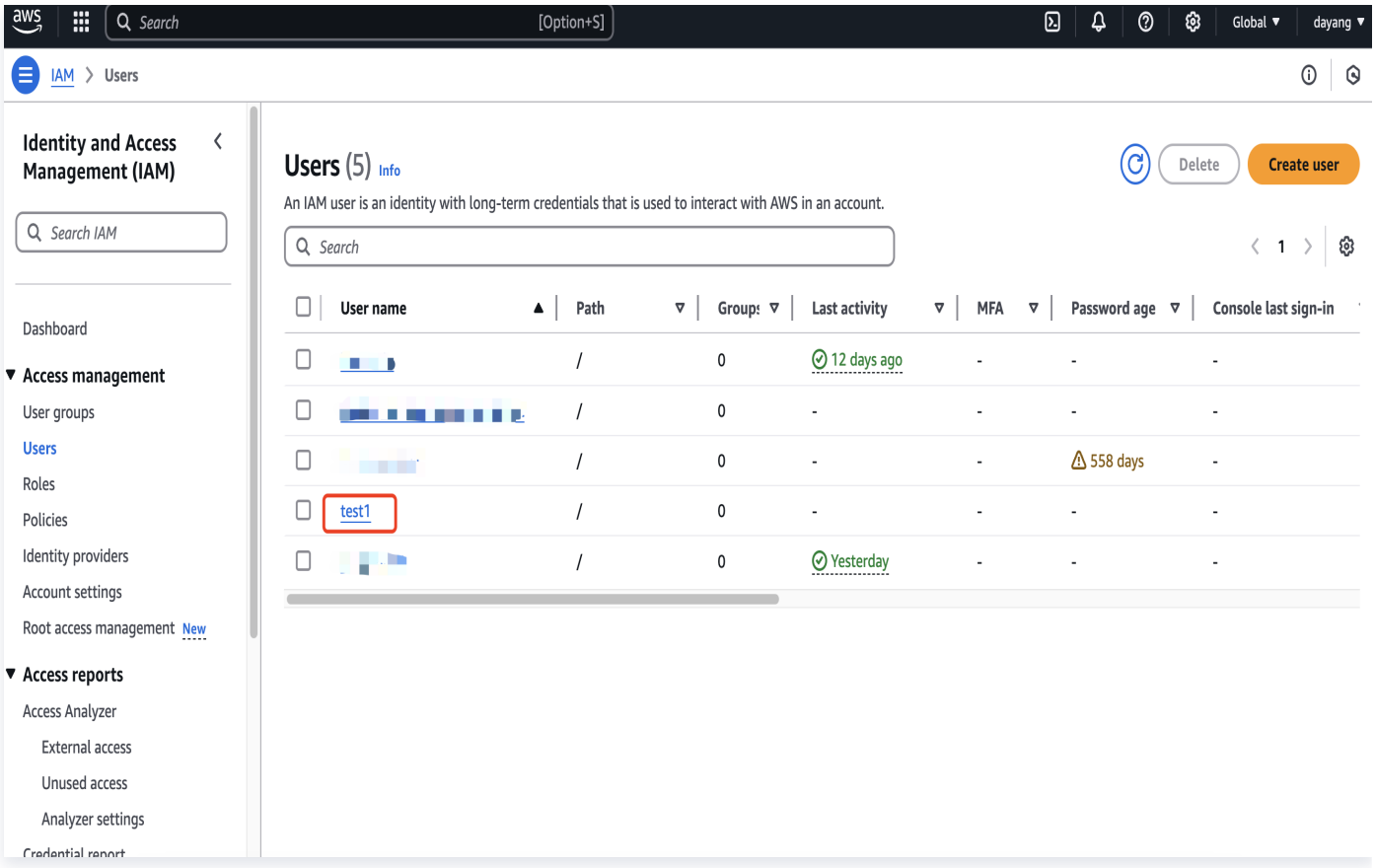
4. 在权限设置页面，单击 **Attach policies directly** 选项，在搜索栏中输入“S3”，在检索结果中选择 AmazonS3FullAccess 权限，单击 **Next** 并完成用户的创建。



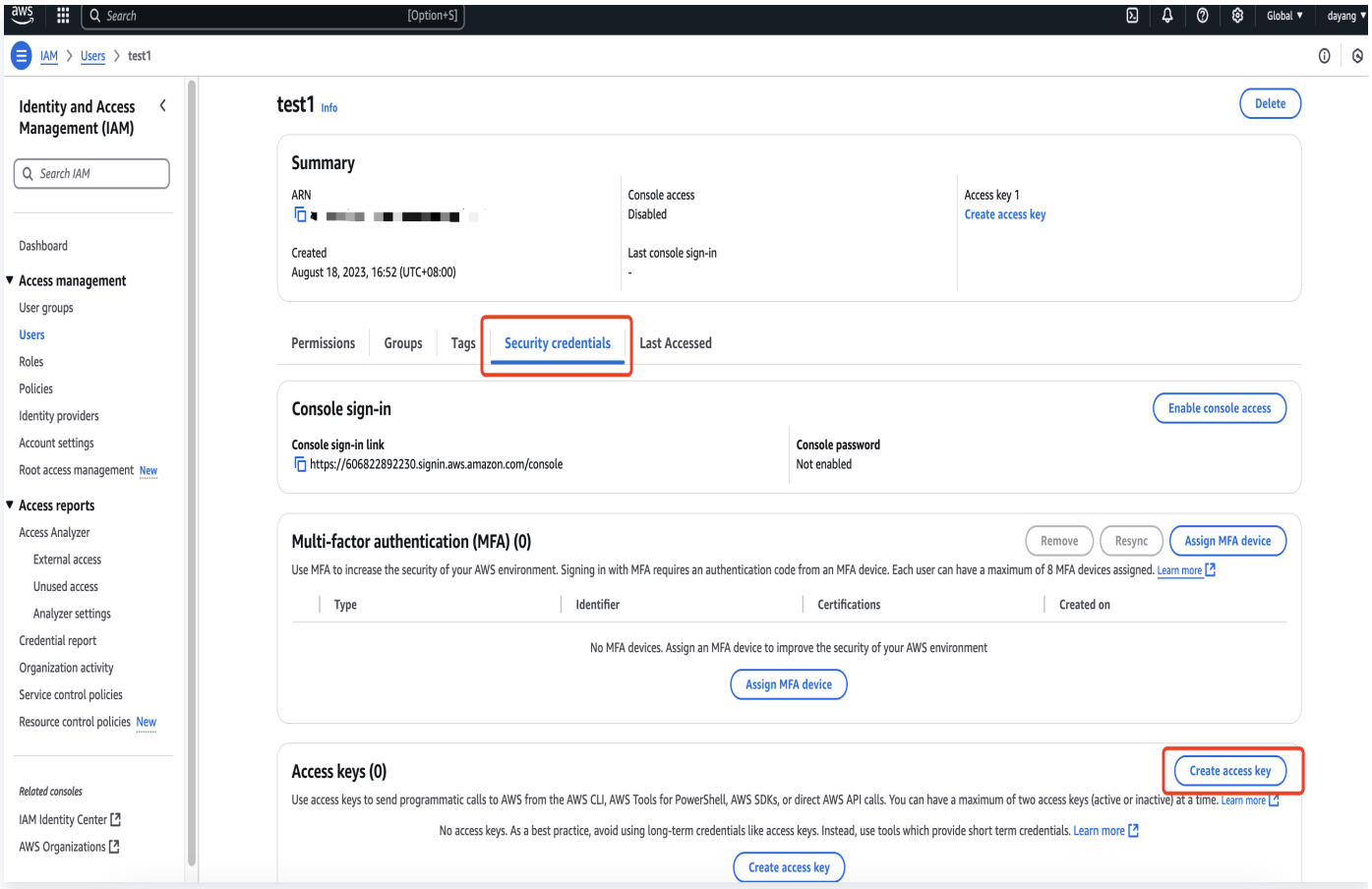
步骤二：创建 AWS IAM 用户访问密钥

> **说明：**
> 以下内容仅供参考，具体操作以 AWS S3实际展示及提供的服务内容为准。

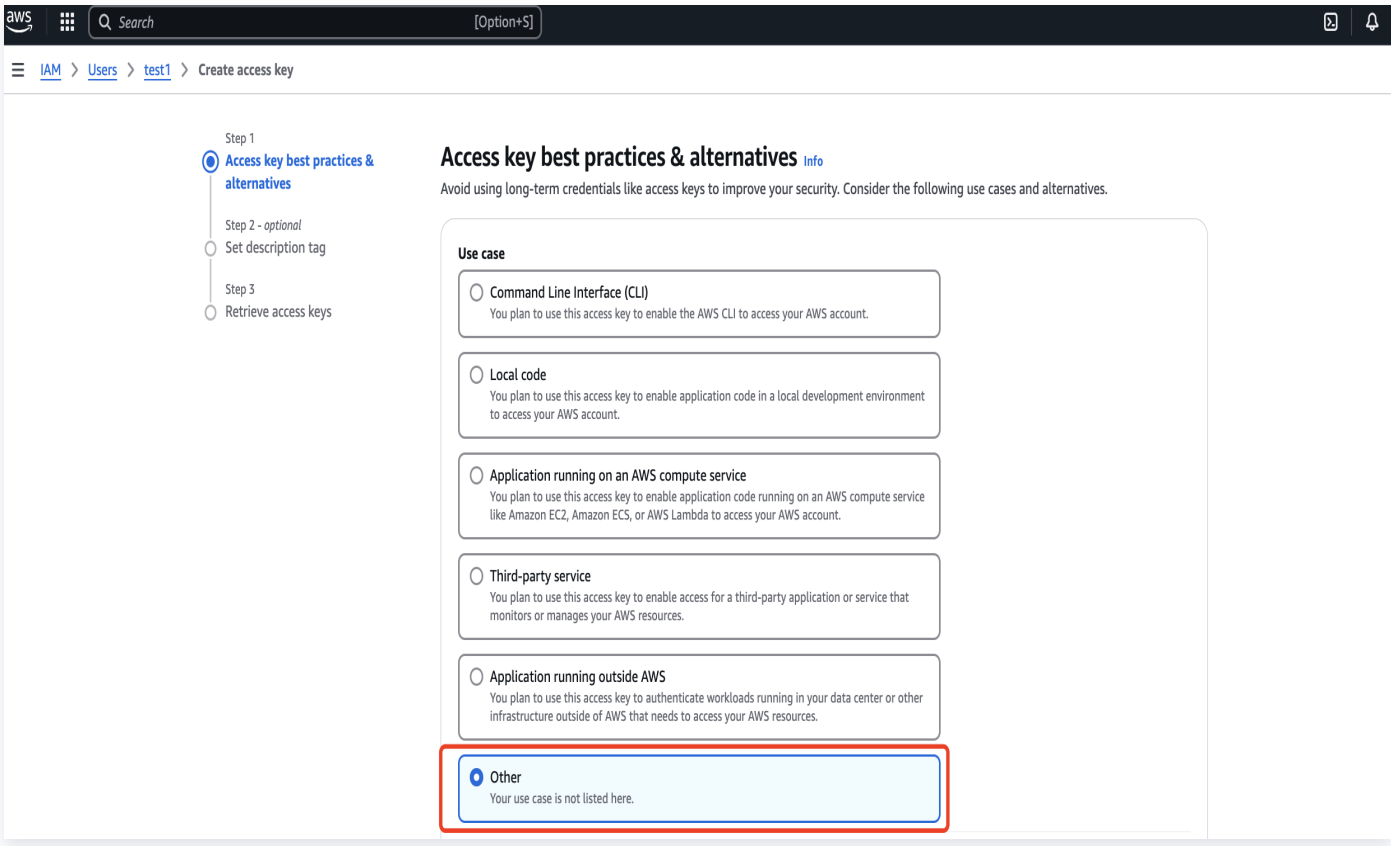
1. AWS IAM 用户创建完成后，单击用户名称。



2. 选择 **Security credentials > Access keys**，然后单击 **Create access key**。



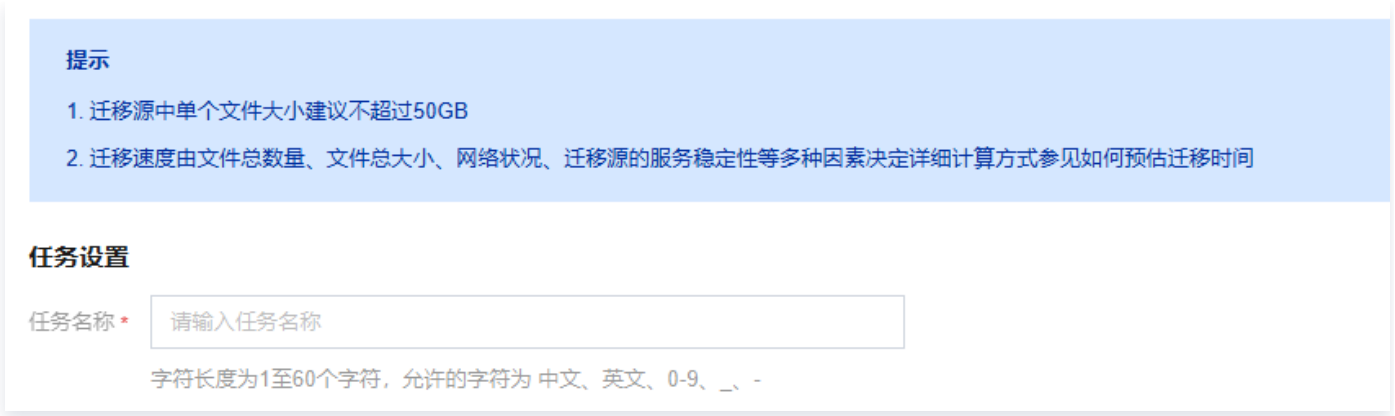
3. 选择 **Other**，单击 **Next** 创建。注意保存好 Access key ID 和 Secret access key。



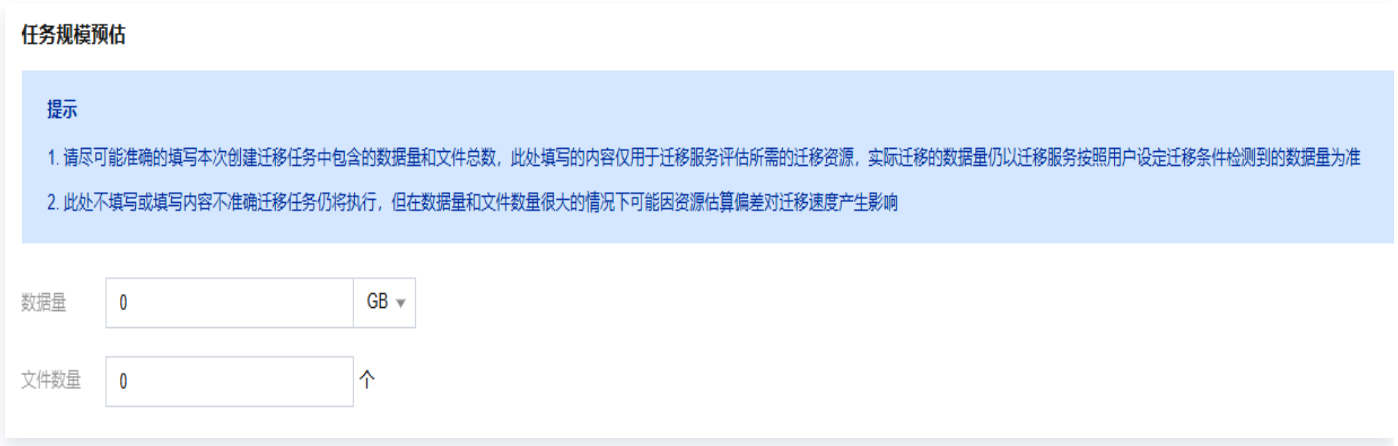
步骤三：创建迁移任务

1. 登录 云点播控制台，单击左侧导航栏**迁移工具**，进入迁移任务列表，并单击**新建任务**，进行迁移参数的设置。

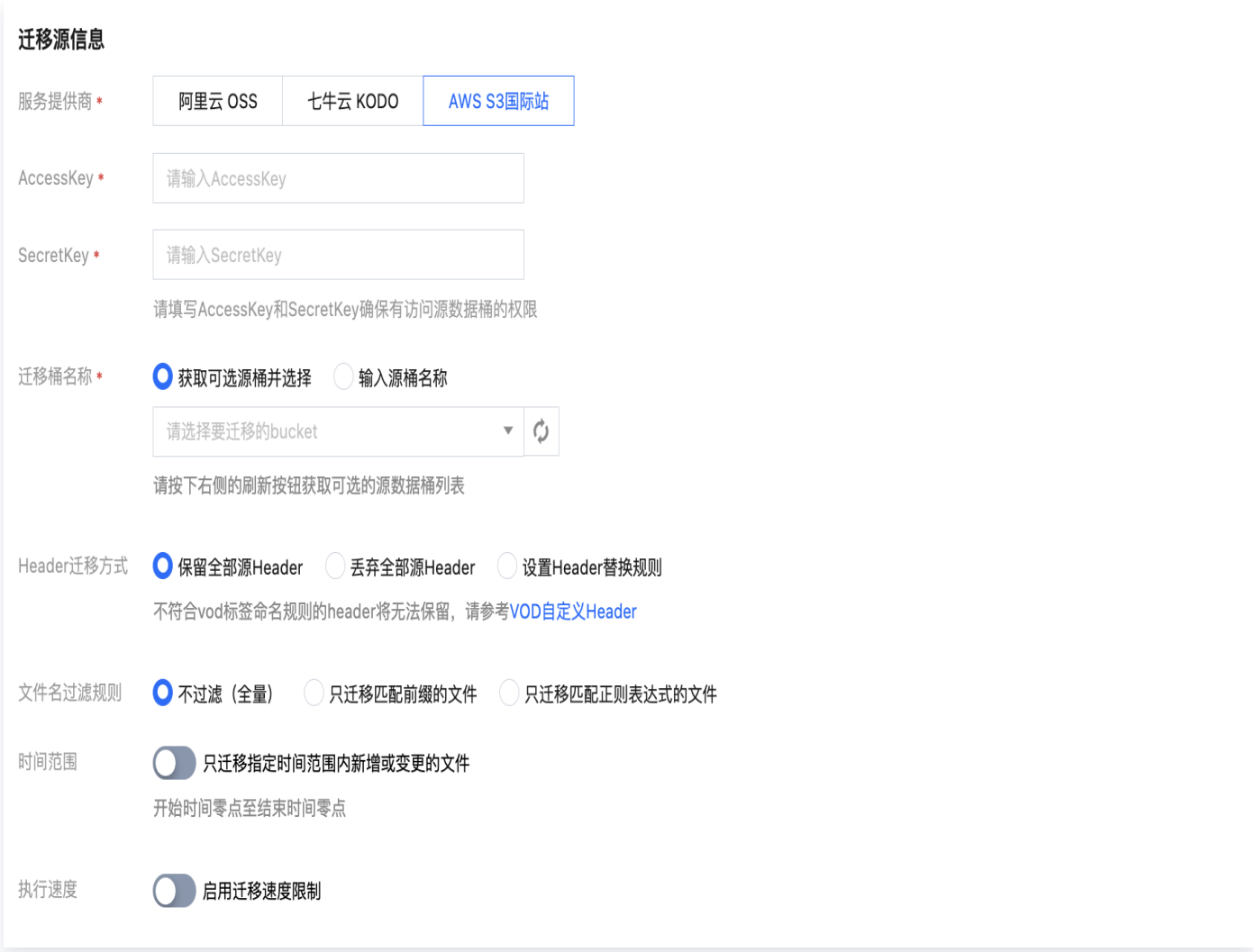
2. 设置迁移任务名称。任务名称长度为1至60个字符，允许的字符为中文、英文、0-9、_、-。



3. 预估任务规模。请准确的填写任务规模，以便我们更好的准备相关资源，非必填。



4. 设置要迁移的文件来源。



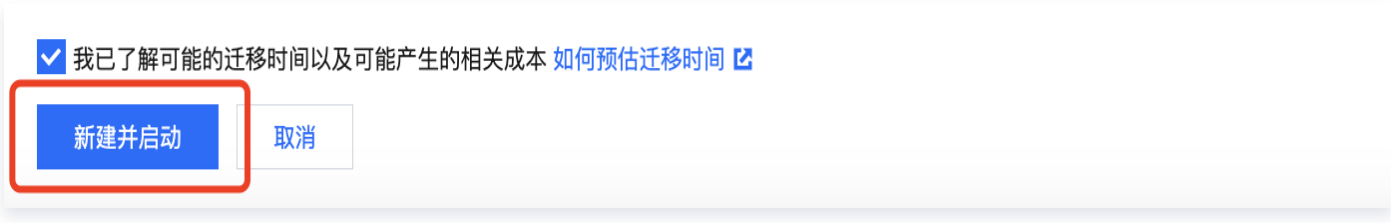
- **服务提供商**：此处迁移源服务提供商应选择 AWS S3。
- **AccessKey、SecretKey**：文本框中输入先前新建用于迁移的 AWS IAM 用户账号的 AccessKeyID 和 AccessKeySecret。
- **迁移桶名称**：填入密钥后，单击“迁移桶名称”下拉框右侧的**刷新**，即可获取源对象存储桶列表。也可以选择手动输入源桶名称。
- **Header迁移方式**：如果源桶中的文件设定了 Header/Tag 并且需要在迁移后保留，请选择保留或设置替换规则。
- **文件名过滤规则**：选择对指定桶中的全部文件进行迁移，或仅迁移指定前缀的文件。
- **时间范围**：开启时间范围，只迁移指定时间范围内新增或变更的文件。
- **执行速度、限速方式**：各公有云厂商的对象存储都有速度限制。为确保业务稳定，请在迁移前与源厂商确认并设置最高迁移可用 Mbps。

5. 选择要迁移到的目标位置。

- **服务提供商**：默认为腾讯云VOD – 专业版。
- **迁移目标应用**：选择您要迁移到的目标应用。如果您有刚创建的应用，您可以单击应用选择框右侧的**刷新**按钮，既可获得最新的专业版应用列表。
- **迁移目标存储桶**：选择目标应用下的具体存储桶。
- **保存路径**：指定迁移到目标桶的指定目录。
  - 保存到根目录： 直接将源桶中的文件按原始相对路径保存到目标桶的根目录。
  - 保存到指定目录：将源桶中的文件保持原始相对路径保存到指定目录中。例如：源桶中的文件 `/testa.txt` ， `/dir/testb.txt` 两个文件，文本框中填写“migration”，那么迁移后这两个文件在目标桶中的路径为： `/migration/testa.txt` ， `/migration/dir/testb.txt` 。
- **同名文件**：指定同名文件的处理方式。

> **注意：**
> - 同名文件选择**覆盖**，则迁移时会直接覆盖掉同名文件。
> - 若同名文件选择**跳过**，会基于文件最后修改时间 LastModified 判断，即：
>   - 如果源地址中文件的 LastModified 晚于或者等于目的地址中文件的 LastModified，则执行覆盖。
>   - 如果源地址中文件的 LastModified 早于目的地址中文件的 LastModified，则执行跳过。
> - 若在迁移过程中对象（文件）内容有变化，需要进行二次迁移。

6. 单击新建并启动，即可启动迁移任务。迁移预估时间可参见文档 预估文件迁移时间 。



步骤四：查看迁移状态和进度

在文件迁移工具主界面中，可以查看所有文件迁移任务的状态和进度：

- “任务完成”状态，绿色是任务完成并且所有文件都迁移成功，黄色是迁移任务完成但部分文件迁移失败。
- 单击“重试失败任务”链接后，该任务中失败的文件将会重试迁移，已经成功迁移的文件不会重传。
- 单击“导出”链接可以导出迁移过程中失败的文件列表。

阿里云 OSS 迁移

最近更新时间：2025-02-05 11:25:32

本文将引导您如何将阿里云 OSS 的文件，通过云点播迁移工具，迁移到云点播专业版。

准备工作

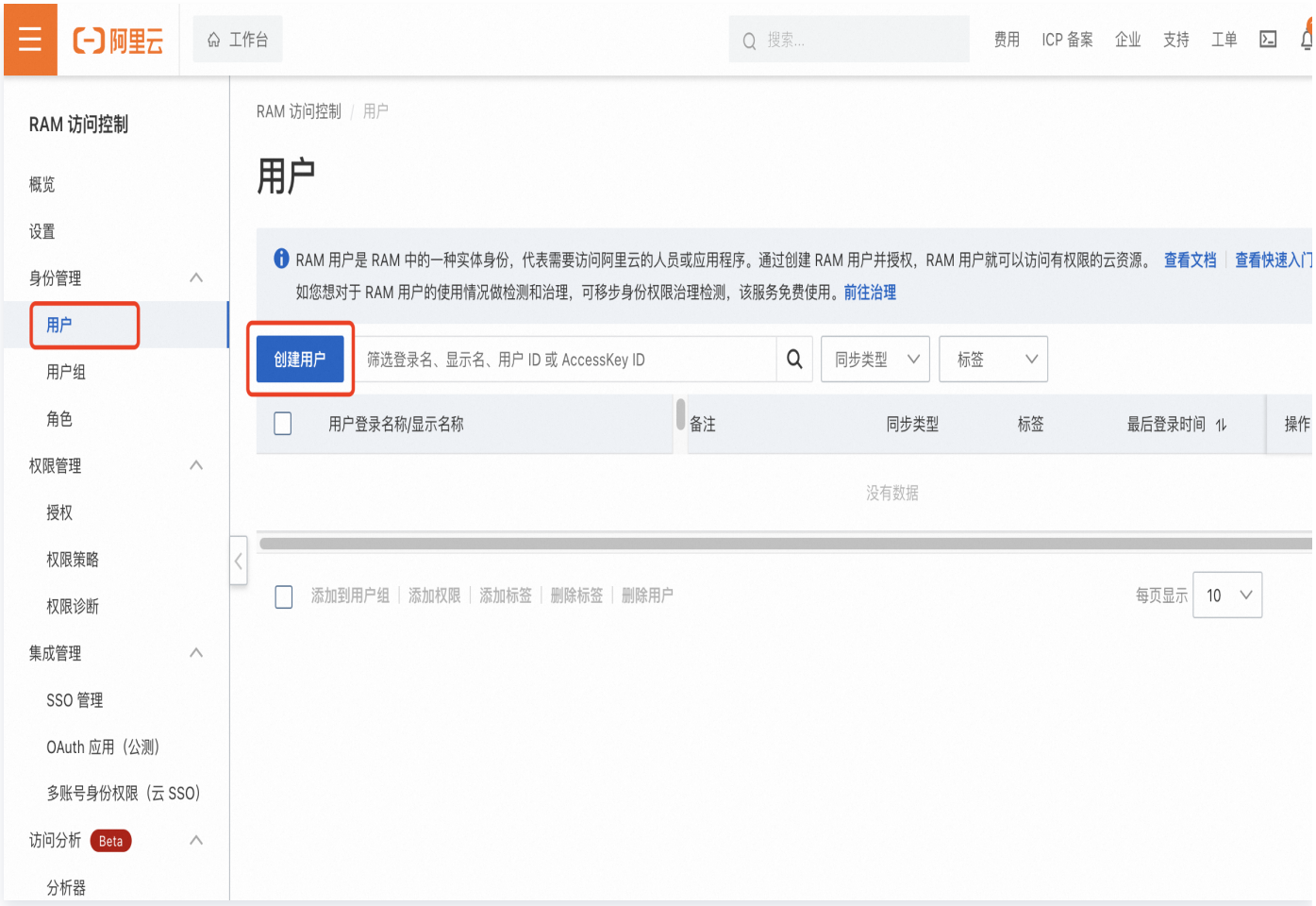
1. 已有可用的专业版应用，创建专业版应用请参考 快速入门 。
2. 已有可用的阿里云 OSS 存储桶。

操作步骤

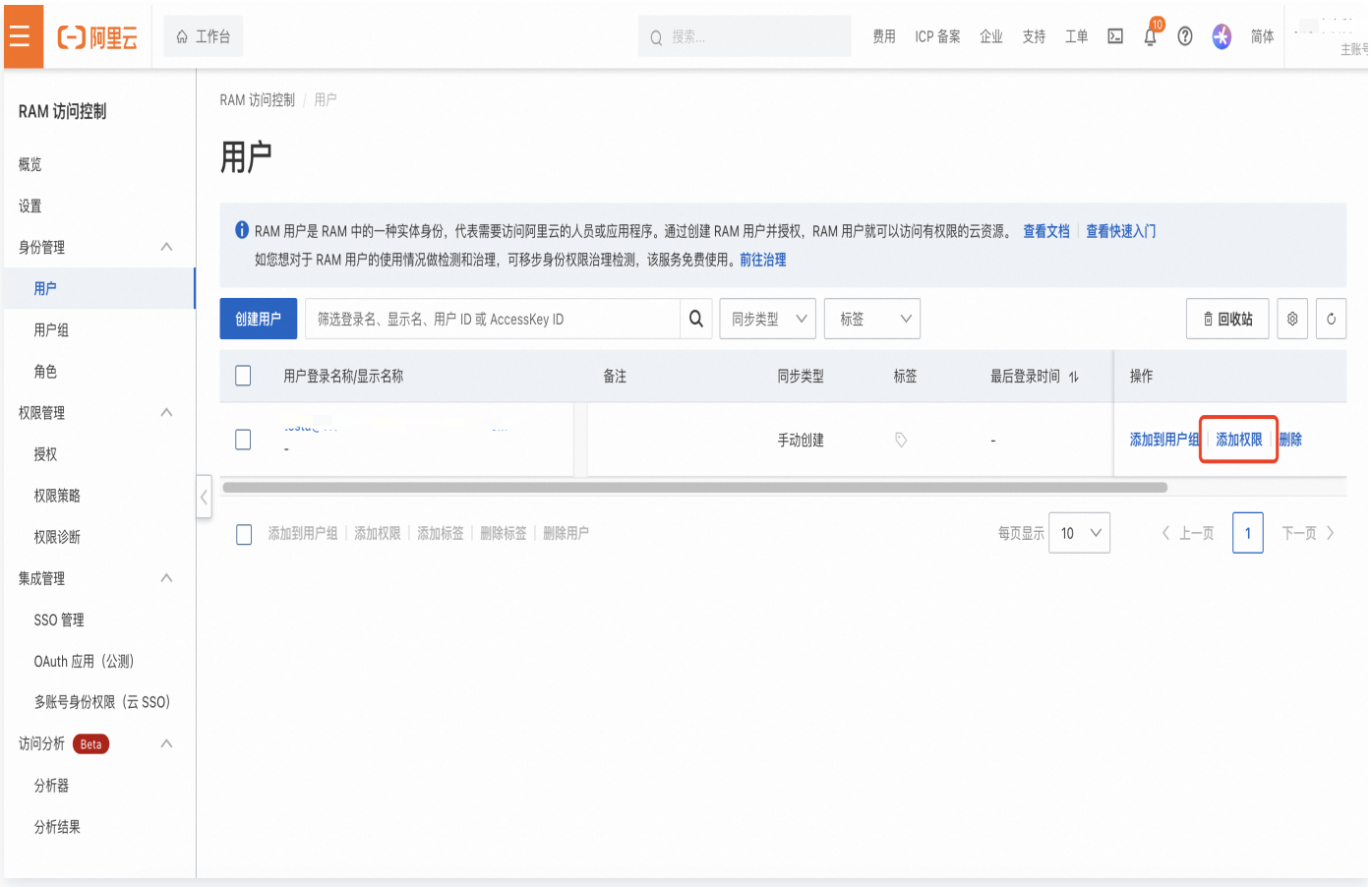
步骤一：创建阿里云 RAM 用户并授予相关权限

> **说明：**
> 以下内容仅供参考，具体操作以阿里云实际展示及提供的服务内容为准。

1. 登录阿里云控制台，进入 RAM 访问控制页面，进入**身份管理** > **用户**页面，并单击**创建用户**。



2. 输入用户的名称，勾选控制台密码登录和编程访问，单击确定。
3. 保存生成的账号、密码、AccessKeyID 和 AccessKeySecret。
4. 用户创建完成后，回到用户列表页面，单击添加权限。

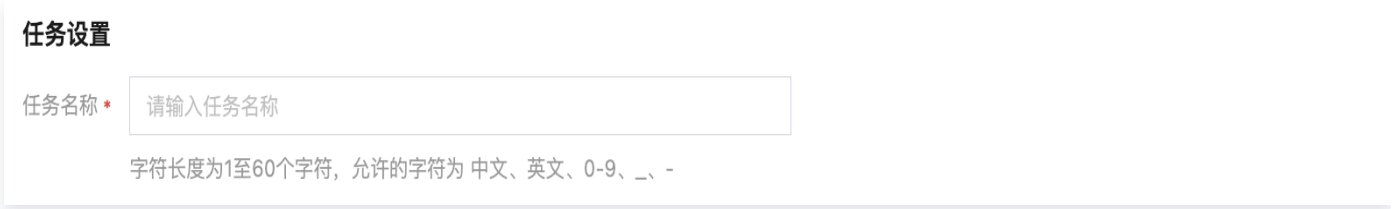


5. 在新增授权页面的权限策略中，输入"OSS"检索策略，并勾选 AliyunOSSFullAccess，单击**确认新增授权**。

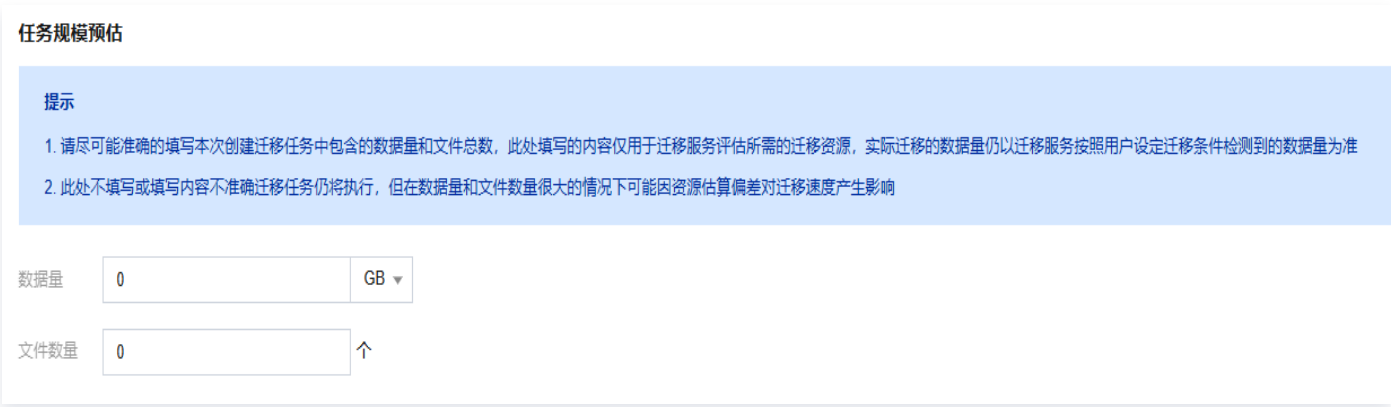
步骤二：创建迁移任务

1. 登录 云点播控制台，单击左侧导航栏**迁移工具**，进入迁移任务列表，并单击**新建任务**，进行迁移参数的设置。

2. 设置迁移任务名称。任务名称长度为1至60个字符，允许的字符为中文、英文、0-9、_、-。



3. 预估任务规模。请准确的填写任务规模，以便我们更好的准备相关资源，非必填。



4. 设置要迁移的文件来源。

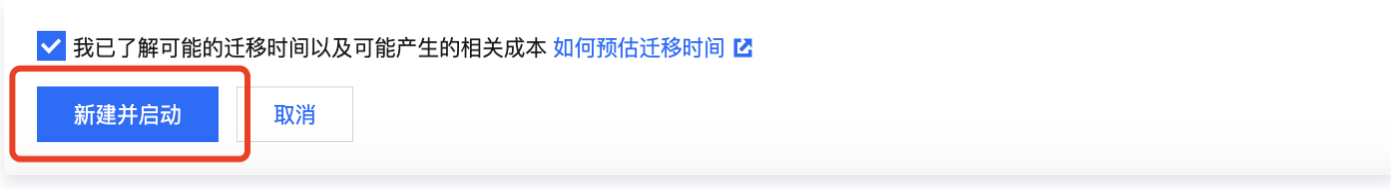
- **服务提供商**：此处迁移源服务提供商应选择阿里云 OSS。
- **AccessKey、SecretKey**：文本框中输入先前新建用于迁移的阿里云 RAM 用户账号的 AccessKeyID 和 AccessKeySecret。
- **迁移桶名称**：填入密钥后，单击“迁移桶名称”下拉框右侧的**刷新**按钮，即可获取源对象存储桶列表。也可以选择手动输入源桶名称。
- **Header迁移方式**：如果源桶中的文件设定了 Header/Tag 并且需要在迁移后保留，请选择保留或设置替换规则。
- **文件名过滤规则**：选择对指定桶中的全部文件进行迁移，或仅迁移指定前缀的文件。
- **时间范围**：开启时间范围，只迁移指定时间范围内新增或变更的文件。
- **执行速度、限速方式**：各公有云厂商的对象存储都有速度限制。为确保业务稳定，请在迁移前与源厂商确认并设置最高迁移可用 Mbps。

5. 选择要迁移到的目标位置。

- **服务提供商**：默认为腾讯云VOD – 专业版。
- **迁移目标应用**：选择您要迁移到的目标应用，如果您有刚创建的应用，您可以单击应用选择框右侧的**刷新**，既可获得最新的专业版应用列表。
- **迁移目标存储桶**：选择目标应用下的具体存储桶。
- **保存路径**：指定迁移到目标桶的指定目录。
  - 保存到根目录： 直接将源桶中的文件按原始相对路径保存到目标桶的根目录。
  - 保存到指定目录：将源桶中的文件保持原始相对路径保存到指定目录中。例如：源桶中的文件 `/testa.txt` ， `/dir/testb.txt` 两个文件，文本框中填写“migration”，那么迁移后这两个文件在目标桶中的路径为： `/migration/testa.txt` ， `/migration/dir/testb.txt` 。
- **同名文件**：指定同名文件的处理方式。

> ⚠ **注意：**
>
> - 同名文件选择**覆盖**，则迁移时会直接覆盖掉同名文件。
> - 若同名文件选择**跳过**，会基于文件最后修改时间 LastModified 判断，即：
>   - 如果源地址中文件的 LastModified 晚于或者等于目的地址中文件的 LastModified，则执行覆盖。
>   - 如果源地址中文件的 LastModified 早于目的地址中文件的 LastModified，则执行跳过。
> - 若在迁移过程中对象（文件）内容有变化，需要进行二次迁移。

6. 单击新建并启动，即可启动迁移任务。迁移预估时间可参见文档 预估文件迁移时间 。



步骤三：查看迁移状态和进度

在文件迁移工具主界面中，可以查看所有文件迁移任务的状态和进度：

- “任务完成”状态，绿色是任务完成并且所有文件都迁移成功，黄色是迁移任务完成但部分文件迁移失败。
- 单击“重试失败任务”链接后，该任务中失败的文件将会重试迁移，已经成功迁移的文件不会重传。
- 单击“导出”链接可以导出迁移过程中失败的文件列表。

七牛云 KODO 迁移

最近更新时间：2025-02-05 11:25:32

本文将引导您如何将七牛云 KODO 的文件，通过云点播迁移工具，迁移到云点播专业版。

准备工作

1. 已有可用的专业版应用，创建专业版应用请参考 快速入门 。
2. 已有可用的七牛云 KODO 存储桶。
3. 如您需要使用 CDN 加速域名来下载文件，请保证待迁移的七牛云 KODO 存储桶，已配置 CDN 加速域名。如您需要使用 S3 域名下载文件，则无需配置 CDN 加速域名。

操作步骤

步骤一：获取七牛云密钥

> **说明：**
> 以下内容仅供参考，具体操作以七牛云实际展示及提供的服务内容为准。

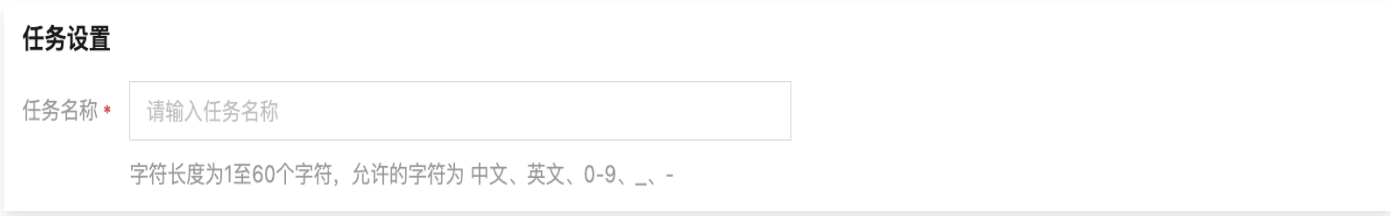
1. 登录七牛云控制台，进入密钥管理页面，单击 SK 右侧的隐藏。

2. 身份验证通过后，保存七牛云的 AK、SK 信息。

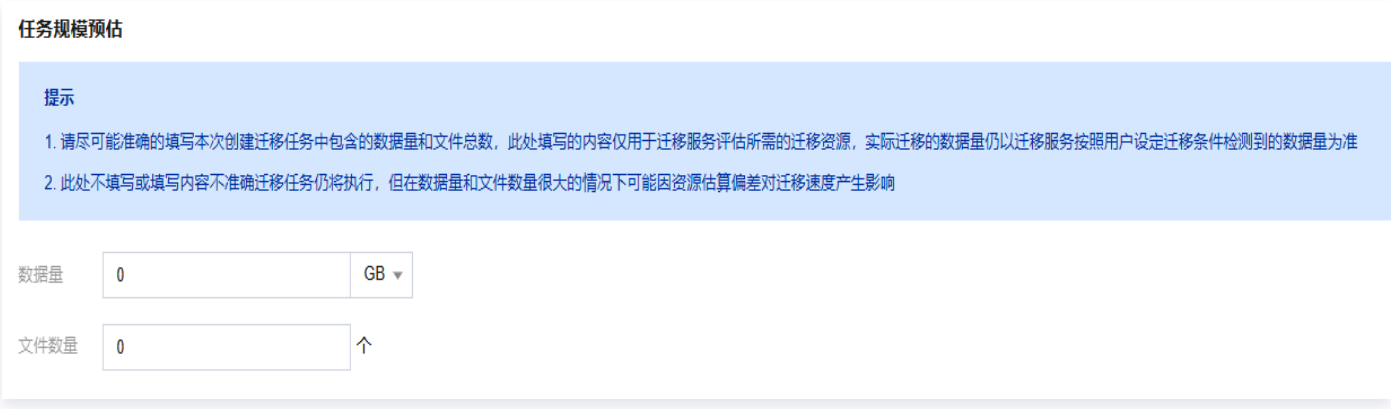
步骤二：创建迁移任务

1. 登录 云点播控制台 ，单击左侧导航栏**迁移工具**，进入迁移任务列表，并单击**新建任务**，进行迁移参数的设置。

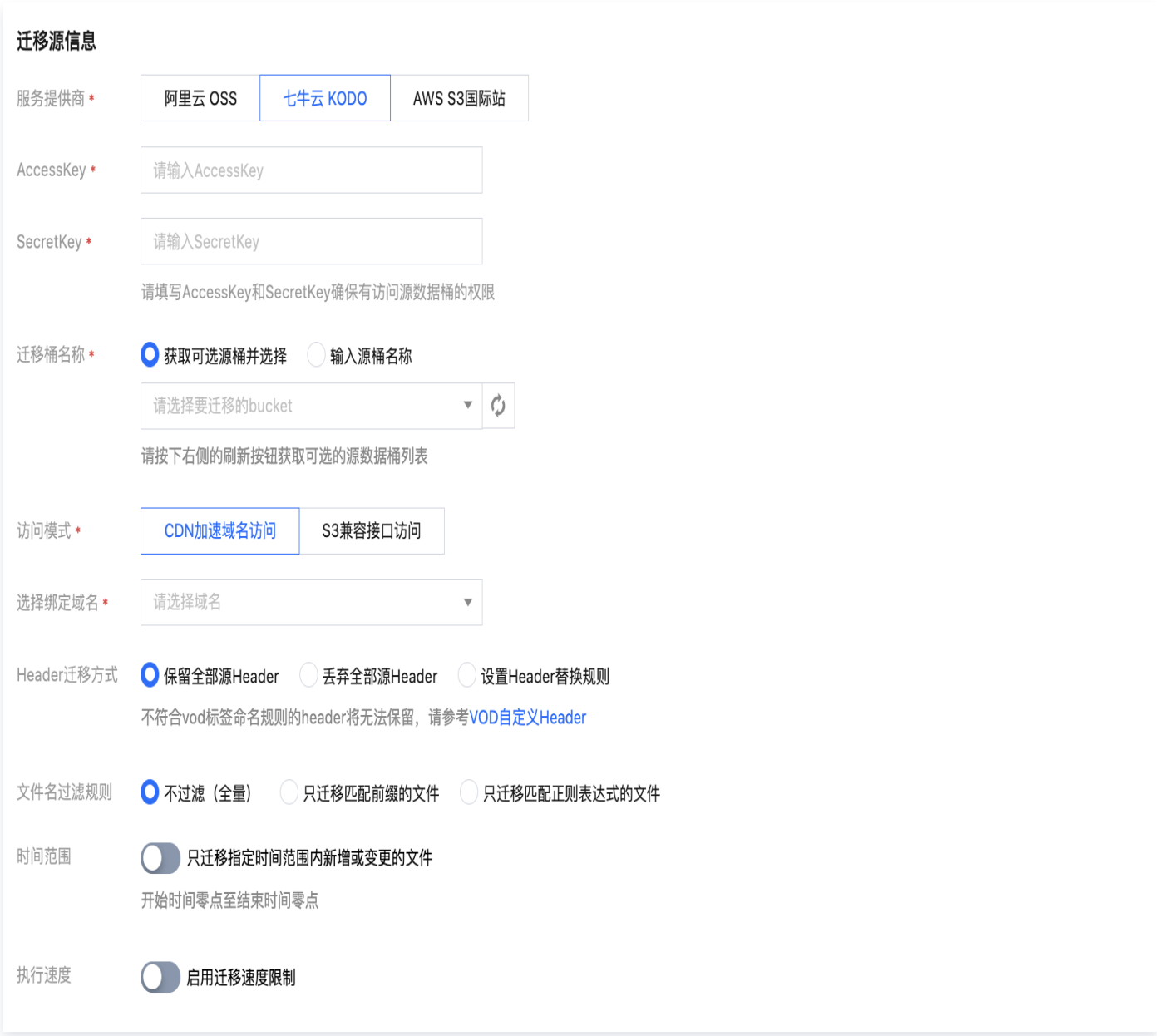
2. 设置迁移任务名称。任务名称长度为1至60个字符，允许的字符为中文、英文、0-9、_、-。



3. 预估任务规模。请准确的填写任务规模，以便我们更好的准备相关资源，非必填。



4. 设置要迁移的文件来源。



- **服务提供商**：此处迁移源服务提供商应选择七牛云 KODO。
- **AccessKey、SecretKey**：文本框中输入步骤一中保存的七牛云 AK、SK。
- **迁移桶名称**：填入密钥后，单击“迁移桶名称”下拉框右侧的**刷新**，即可获取源对象存储桶列表。也可以选择手动输入源桶名称。
- **访问模式**：
  - CDN 加速域名访问：上述迁移桶确认后，选择该项可以获取在七牛控制台中对应存储桶所绑定的 CDN 加速域名列表，在下拉列表中选择域名即可，任务运行会通过 CDN 加速域名访问待迁移文件。
  - S3兼容接口访问：通过七牛控制台获取到 S3兼容访问接口的空间域名，然后填入输入框，任务运行会通过空间域名直接访问待迁移文件。

- **Header迁移方式**：如果源桶中的文件设定了 Header/Tag 并且需要在迁移后保留，请选择保留或设置替换规则。
- **文件名过滤规则**：选择对指定桶中的全部文件进行迁移，或仅迁移指定前缀的文件。
- **时间范围**：开启时间范围，只迁移指定时间范围内新增或变更的文件。
- **执行速度、限速方式**：各公有云厂商的对象存储都有速度限制。为确保业务稳定，请在迁移前与源厂商确认并设置最高迁移可用 Mbps。

5. 选择要迁移到的目标位置。

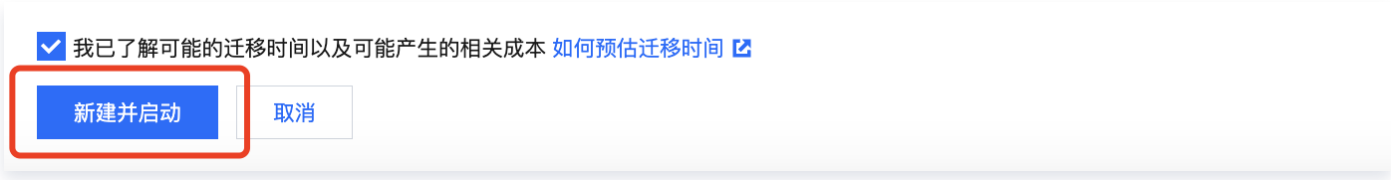
- **服务提供商**：默认为腾讯云VOD – 专业版。
- **迁移目标应用**：选择您要迁移到的目标应用，如果您有刚创建的应用，您可以单击应用选择框右侧的**刷新**，既可获得最新的专业版应用列表。
- **迁移目标存储桶**：选择目标应用下的具体存储桶。
- **保存路径**：指定迁移到目标桶的指定目录。
  - 保存到根目录： 直接将源桶中的文件按原始相对路径保存到目标桶的根目录。
  - 保存到指定目录：将源桶中的文件保持原始相对路径保存到指定目录中。例如：源桶中的文件 `/testa.txt` ， `/dir/testb.txt` 两个文件，文本框中填写“migration”，那么迁移后这两个文件在目标桶中的路径为： `/migration/testa.txt` ， `/migration/dir/testb.txt` 。
- **同名文件**：指定同名文件的处理方式。

注意：

- 同名文件选择**覆盖**，则迁移时会直接覆盖掉同名文件。
- 若同名文件选择**跳过**，会基于文件最后修改时间 LastModified 判断，即：
  - 如果源地址中文件的 LastModified 晚于或者等于目的地址中文件的 LastModified，则执行覆盖。

- 如果源地址中文件的 LastModified 早于目的地址中文件的 LastModified，则执行跳过。
- 若在迁移过程中对象（文件）内容有变化，需要进行二次迁移。

6. 单击新建并启动，即可启动迁移任务。迁移预估时间可参见文档 预估文件迁移时间 。



步骤三：查看迁移状态和进度

在文件迁移工具主界面中，可以查看所有文件迁移任务的状态和进度：

- “任务完成”状态，绿色是任务完成并且所有文件都迁移成功，黄色是迁移任务完成但部分文件迁移失败。
- 单击“重试失败任务”链接后，该任务中失败的文件将会重试迁移，已经成功迁移的文件不会重传。
- 单击“导出”链接可以导出迁移过程中失败的文件列表。

增量迁移工具

最近更新时间：2025-03-18 11:04:12

本文将指引您如何配置增量迁移策略。配置增量迁移策略后，当请求的文件在云点播中不存在时，云点播将从您配置的迁移源地址中拉取文件，返回给请求方的同时将该文件写入到云点播中。增量迁移策略适用于热文件迁移等逐步迁移的场景，您可以按照自身实际需要进行设置。

准备工作

1. 已有可用的专业版应用，创建专业版应用请参考 快速入门 。
2. 已有可用的迁移源站。

操作步骤

1. 登录 云点播控制台 ，单击左侧导航栏**迁移工具**，选择**增量迁移工具**，并单击**新建策略**，进行迁移参数的设置。

2. 设置迁移任务名称。任务名称长度为1至100个字符，允许的字符为中文、英文、0-9、-。

3. 设置要迁移的文件来源。

- 迁移源名称：暂时仅支持选择 HTTP源，对象存储源将在后续支持。
- 迁移源地址：只需填入域名或 IP 地址，支持域名或 IP 地址后面添加端口号。无需加上前缀 http:// 或 https://。
- HTTP Header传递规则：可配置需要透传给迁移源站的请求中带有的头部信息。
  - 透传指定 HTTP Header：指定允许传递的 Header，最多可添加 10 个。
  - 禁止透传所有 HTTP Header：禁止透传所有参数。
- 新增 HTTP Header：指定 HTTP Header 的 Key 和 Value，指定后，会在 Header 中添加该 Key 和 Value，并透传到迁移源站，最多可添加10个。

4. 设置要迁移的目标信息。

- 迁移目标应用：选择您需要迁移到的目标应用，仅支持选择专业版应用。如果您有刚创建的应用，您可以单击应用选择框右侧的刷新按钮，即可获得最新的专业版应用列表。

- 迁移目标存储桶：选择目标应用下的具体存储桶，单个存储桶仅支持设置1条增量迁移策略。

5. 单击**确认**，即可完成增量迁移策略的配置。

相关文档

- 如果您需要定时清理增量迁移的文件，您可以参考 存储策略 配置定时清理策略。
- 如果您需要一次性迁移完所有文件，您可以参考 全量迁移工具 配置全量迁移策略。

开发指南
媒体存储
基本概念

最近更新时间：2025-01-08 16:03:52

本文将介绍云点播专业版的一些基本概念，包含存储桶、对象相关内容。

存储桶

存储桶是云点播专业版应用下的存储容器，单个应用可创建多个地域的存储桶，是对象的载体，无容量上限。

存储桶命名规范

存储桶名称支持用户自定义，命名规范如下：

- 仅支持小写英文字母和数字，即[a-z，0-9]、中划线“-”及其组合。
- 存储桶命名不能以“-”开头或结尾。
- 名称长度不超过40个字符。

存储桶所属地域

存储桶所属地域是指存储桶数据实际存放的地理位置。云点播专业版应用可以创建多个地域的存储桶。您可以选择在离您的业务最近的地域上创建存储桶，以达到低延迟、低成本以及满足合规性要求的目的。

> ⚠ **注意：**
> 云点播专业版每个应用在单地域仅支持创建一个存储桶，且存储桶地域选择后将不允许更改，如需在单个应用内创建多个相同地域的存储桶，请 联系我们 。

存储桶内网访问域名

云点播专业版应用为每个存储桶提供一个内网访问域名。通过这个域名，您可以通过内网访问存储桶中的对象。如果为同地域访问，则可实现内网访问，此时上传和下载均产生内网流量，不会产生流量费用。请参见 使用方法 。

对象

对象是云点播专业版的基本单元，可以是任何格式的数据（视频、图片、文本等）。存储桶是对象的载体，对象都存储在存储桶中。

对象键

对象键是对象在存储桶内的唯一标识，可以理解为文件路径，例如 2025/01/01/hello.txt。

对象键命名规范

- 可以使用任何 UTF-8 字符，为了确保名称与其他应用程序的最大兼容性，推荐使用英文大小写字母、数字，即[a-z，A-Z，0-9]及其组合。
- 编码长度最大为850个字节。
- 不允许以正斜线 / 或者反斜线 \ 开头。
- 不允许携带以下字符串：%0a、%0d。
- 不支持 ASCII 控制字符中的字符上(↑)，字符下(↓)，字符右(→)，字符左(←)，分别对应 CAN(24)，EM(25)，SUB(26)，ESC(27)。
- 对于特殊字符，例如*、%等字符，尽量避免直接作为文件名使用。

支持的存储操作

最近更新时间：2025-03-24 09:53:42

概述

云点播专业版应用支持以对象存储模式管理媒体内容，客户在专业版应用中创建了存储桶后，可以按 AWS S3 兼容的方式对存储桶中的媒体内容进行管理。
本文介绍专业版应用支持的存储操作以及这些操作可使用的访问方式。以 AWS S3 的存储操作为对照，通过表格来列举已经实现或者产品功能上已经覆盖的 S3 操作，并详细列举对操作中具体特性的支持情况。

对象级操作

已支持的对象级操作如下表：

操作名	S3 API	特性	访问方式
获取对象元数据	HeadObject	**请求头：** 已支持 If 匹配专用头部： • If-Match • If-Modified-Since • If-None-Match • If-Unmodified-Since	• 存储桶公网域名（S3 兼容） • 存储桶内网域名（S3 兼容）
获取对象列表	ListObjects	**URL 参数：** 已支持如下： • delimiter • encoding-type • marker • max-keys • prefix	• 腾讯云控制台 • 存储桶公网域名（S3 兼容） • 存储桶内网域名（S3 兼容）
	ListObjectsV2	**URL 参数：** 已支持如下： • list-type • continuation-token • delimiter • encoding-type • max-keys • prefix • start-after	

下载对象	GetObject	无	• 腾讯云控制台 • 存储桶内网域名（S3 兼容）

上传单个对象	PutObject	**请求头：**
		已支持如下：
		• Cache-Control
		• Content-Disposition
		• Content-Encoding
		• Content-Length
		• Content-MD5
		• Content-Type
		• Expires
		• x-amz-storage-class：
		  ○ STANDARD
		  ○ STANDARD_IA
			• 腾讯云控制台
			• 存储桶公网域名（S3 兼容）
			• 存储桶内网域名（S3 兼容）
			• 应用上传域名（S3 兼容）

删除单个对象	DeleteObject	无	• 腾讯云控制台 • 存储桶公网域名（S3 兼容） • 存储桶内网域名（S3 兼容）
删除多个对象	DeleteObjects	无	• 腾讯云控制台 • 存储桶公网域名（S3 兼容） • 存储桶内网域名（S3 兼容）
获取分片上传事件列表	ListMultipartUploads	**URL 参数：** 已支持如下： • delimiter • encoding-type • key-marker • max-uploads • prefix • upload-id-marker	• 腾讯云控制台 • 存储桶公网域名（S3 兼容） • 存储桶内网域名（S3 兼容） • 应用上传域名（S3 兼容）
创建分片上传	CreateMultipartUpload	**请求头：** 已支持如下： • Cache-Control • Content-Disposition • Content-Encoding • Content-Type • Expires • x-amz-storage-class：	• 腾讯云控制台 • 存储桶公网域名（S3 兼容） • 存储桶内网域名（S3 兼容） • 应用上传域名（S3 兼容）

		○ STANDARD ○ STANDARD_IA	
完成分片上传	CompleteMultipartUpload	**URL 参数:** 已支持：uploadId **请求 Body:** 已支持如下: • ETag • PartNumber	• 腾讯云控制台 • 存储桶公网域名（S3 兼容） • 存储桶内网域名（S3 兼容） • 应用上传域名（S3 兼容）
终止分片上传	AbortMultipartUpload	**URL 参数:** 已支持：uploadId	• 腾讯云控制台 • 存储桶公网域名（S3 兼容） • 存储桶内网域名（S3 兼容） • 应用上传域名（S3 兼容）
上传分片	UploadPart	**请求头:** 已支持如下: • Content-Length • Content-MD5 **URL 参数:** 已支持如下: • partNumber • uploadId	• 腾讯云控制台 • 存储桶公网域名（S3 兼容） • 存储桶内网域名（S3 兼容） • 应用上传域名（S3 兼容）
获取指定上传事件中已上传的分片列表	ListParts	**URL 参数:** 已支持如下: • max-parts • part-number-marker • uploadId	• 腾讯云控制台 • 存储桶公网域名（S3 兼容） • 存储桶内网域名（S3 兼容） • 应用上传域名（S3 兼容）
复制单个对象	CopyObject	**请求头:** 已支持如下: • Cache-Control • Content-Disposition • Content-Encoding • Content-Type • Expires • x-amz-copy-source • x-amz-metadata-directive • x-amz-storage-class ○ STANDARD	• 腾讯云控制台 • 存储桶公网域名（S3 兼容） • 存储桶内网域名（S3 兼容）

		○ STANDARD_IA ○ ARCHIVE ○ DEEP_ARCHIVE 已支持 If 匹配专用头部： • x-amz-copy-source-if-match • x-amz-copy-source-if-modified-since • x-amz-copy-source-if-none-match • x-amz-copy-source-if-unmodified-since	
复制分片	UploadPartCopy	**请求头：** 已支持如下： • x-amz-copy-source • x-amz-copy-source-range 已支持 If 匹配专用头部： • x-amz-copy-source-if-match • x-amz-copy-source-if-modified-since • x-amz-copy-source-if-none-match • x-amz-copy-source-if-unmodified-since **URL 参数：** 已支持如下： • partNumber • uploadId	• 腾讯云控制台 • 存储桶公网域名（S3 兼容） • 存储桶内网域名（S3 兼容）
恢复归档或深度归档对象	Restore Object	**请求 Body：** 已支持如下： • Days	• 腾讯云控制台 • 存储桶公网域名（S3 兼容） • 存储桶内网域名（S3 兼容）

存储桶级操作

> **说明：**
> 云点播专业版在应用级别提供存储桶操作，功能上可覆盖常用的 S3 存储桶操作，但是不提供 S3 兼容的 API。

已支持的存储桶级操作如下：

操作名	S3 API	特性	访问方式
创建存储桶	CreateBucket	已支持：指定存储地域	腾讯云控制台
获取存储桶列表	ListBuckets	无	腾讯云控制台
创建存储桶生命周期	PutBucketLifecycleConfiguration	已支持：到期降冷、到期删除	腾讯云控制台
查询存储桶生命周期	GetBucketLifecycleConfiguration	无	腾讯云控制台
删除存储桶生命周期	DeleteBucketLifecycle	无	腾讯云控制台

存储访问方式

最近更新时间：2025-03-14 11:57:23

本文主要介绍专业版应用支持的存储访问方式和这些访问方式的使用方法。

> **说明：**
> 控制台访问方式请参见 控制台指南，腾讯云 API3.0 访问方式请参见 服务端 API 文档，这两种方式本文不再重复介绍。

访问方式

专业版应用支持的存储访问方式如下：

类型	访问方式	访问域名	作用
预置域名	存储桶公网域名	`[BucketId].vodpro.[存储地域].eovod.com`	用于公网对媒体文件进行增删改查。
	存储桶内网域名	`[BucketId].vodsrc-internal.[存储地域].eovod.com`	用于在腾讯云内网同地域对媒体文件进行增删查改，支持下载且不产生流量费用。
自定义域名	EdgeOne 加速分发域名	客户自定义	用于媒体文件的分发播放。

使用方法

存储桶公网域名

专业版应用为每个存储桶预置了一个公网域名，客户可使用该域名对存储桶内的媒体文件进行增删改查。具体支持的操作清单请参见文档 支持的存储操作 。

> **注意：**
> - 存储桶公网域名有一定的生效时间，一般在存储桶创建后 30 ~ 120 分钟生效。
> - 存储桶公网域名不支持下载文件，如需下载或浏览文件内容请使用 Edgeone 加速域名 。

准备工作

1. 创建应用和存储桶。

   在 云点播控制台 创建好专业版应用和存储桶，具体操作步骤参见 快速入门 和 创建存储桶 文档。

2. 获取密钥对。

从专业版应用的密钥管理中，获取密钥对 `AccessKeyId` 和 `SecretAccessKey`，具体操作步骤参见 密钥管理 文档。

使用 S3 SDK 访问存储桶

下面介绍在常用的编程语言中，如何进行适配以访问专业版应用的存储桶。
假设存储地域为 `ap-guangzhou`，存储桶 ID 为 `bucketid1`，常用语言初始化 S3 客户端并获取对象元数据的代码实现如下所示：

GO

```
import (
    "context"
    "errors"
    "fmt"
    "net/url"
    "os"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/credentials"
    "github.com/aws/aws-sdk-go-v2/service/s3"
    smep "github.com/aws/smithy-go/endpoints"
    "github.com/aws/smithy-go/logging"
)

type customEndpointResolverV2 struct {
}

// ResolveEndpoint 自定义接入点
func (r *customEndpointResolverV2) ResolveEndpoint(ctx context.Context,
    params s3.EndpointParameters) (smep.Endpoint, error) {
    if params.Bucket == nil || params.Region == nil {
        return smep.Endpoint{}, errors.New("invalid endpoint param")
    }
    return smep.Endpoint{
        URI: url.URL{
            Scheme: "https",
            Host:   fmt.Sprintf("%s.vodpro.%s.eovod.com",
*params.Bucket, *params.Region),
        },
    }, nil
}
```

```go
func main() {
    // 初始化客户端
    s3cli := s3.New(s3.Options{
        Credentials: credentials.NewStaticCredentialsProvider(
            "AccessKeyId",     // 填入密钥对中的 AccessKeyId
            "SecretAccessKey", // 填入密钥对中的 SecretAccessKey
            ""),
        EndpointResolverV2: new(customEndpointResolverV2),         // 自定义访问域名
        UsePathStyle:       false,                                 // 请求禁用 path style
        Logger:             logging.NewStandardLogger(os.Stdout), // 日志打印到标准输出流
        ClientLogMode:      aws.LogRequest | aws.LogResponse,      // 打印请求头和响应头
        Region:             "ap-guangzhou",                        // 存储地域
    })
    // 获取媒体文件的元数据
    _, _ = s3cli.HeadObject(context.TODO(), &s3.HeadObjectInput{
        Bucket: aws.String("bucketid1"),  // Bucket 设置为点播专业版应用中的存储桶 ID
        Key:    aws.String("a/b/c.jpeg"), // 媒体文件在存储桶中的路径
    })
}
```

Java

```java
import software.amazon.awssdk.auth.credentials.AwsBasicCredentials;
import
software.amazon.awssdk.auth.credentials.StaticCredentialsProvider;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.HeadObjectRequest;
import software.amazon.awssdk.services.s3.model.HeadObjectResponse;

import java.net.URI;

public class Main {
    // 自定义端点解析器
    public static void main(String[] args) {
        // 初始化 S3 客户端
```

```
        S3Client s3Client = S3Client.builder()

.credentialsProvider(StaticCredentialsProvider.create(AwsBasicCredential
s.create(
                        "AccessKeyId",     // 填入密钥对中的 AccessKeyId
                        "SecretAccessKey" // 填入密钥对中的 SecretAccessKey
                )))
                .endpointOverride(URI.create("https://vodpro.ap-
guangzhou.eovod.com"))
                .region(Region.of("ap-guangzhou"))
                .build();

        // 获取媒体文件的元数据
        try {
            HeadObjectRequest headObjectRequest =
HeadObjectRequest.builder()
                    .bucket("bucketid1")     // Bucket 设置为点播专业版应用中
的存储桶 ID
                    .key("a/b/c.jpeg")       // 媒体文件在存储桶中的路径
                    .build();
            HeadObjectResponse headObjectResponse =
s3Client.headObject(headObjectRequest);
            System.out.println("Content Length: " +
headObjectResponse.contentLength());
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            s3Client.close();
        }
    }
}
```

C++

```
#include <aws/core/Aws.h>
#include <aws/core/auth/AWSCredentials.h>
#include <aws/s3/S3Client.h>
#include <aws/s3/model/HeadObjectRequest.h>
#include <iostream>
#include <string>

int main() {
```

```
    // 初始化 AWS SDK
    const Aws::SDKOptions options;
    InitAPI(options);
    // 自定义访问域名
    Aws::Client::ClientConfiguration clientConfig;
    clientConfig.scheme = Aws::Http::Scheme::HTTPS;
    clientConfig.endpointOverride = "vodpro.ap-guangzhou.eovod.com";
    // 创建 S3 客户端
    const Aws::S3::S3Client s3Client(
        Aws::Auth::AWSCredentials(
            "AccessKeyId", // 填入密钥对中的 AccessKeyId
            "SecretAccessKey" // 填入密钥对中的 SecretAccessKey
        ),
        nullptr, clientConfig);
    // 创建 HeadObject 请求
    Aws::S3::Model::HeadObjectRequest request;
    request.SetBucket("bucketid1"); // Bucket 设置为点播专业版应用中的存储桶
ID
    request.SetKey("a/b/c.jpeg"); // 媒体文件在存储桶中的路径
    // 发送请求
    auto outcome = s3Client.HeadObject(request);
    if (outcome.IsSuccess()) {
        std::cout << "Head object succeeded!" << std::endl;
        const auto &object = outcome.GetResult();
        std::cout << "Content Length: " << object.GetContentLength() <<
std::endl;
    } else {
        const auto error = outcome.GetError();
        std::cout << "Error: " << error.GetMessage() << std::endl;
    }
    // 清理 AWS SDK
    Aws::ShutdownAPI(options);
    return 0;
}
```

Python

```
import boto3
from botocore.config import Config
from botocore.exceptions import ClientError
```

```
# 创建 S3 客户端
s3_client = boto3.client(
    's3',
    aws_access_key_id='AccessKeyId',                          # 填入密钥对中的
AccessKeyId
    aws_secret_access_key='SecretAccessKey',                  # 填入密钥对中的
SecretAccessKey
    endpoint_url='https://vodpro.ap-guangzhou.eovod.com',     # 自定义访问域名
    config=Config(s3={'addressing_style': 'virtual'}),        # 使用 virtual
hosted-style
)

try:
    # 获取媒体文件的元数据
    response = s3_client.head_object(
        Bucket="bucketid1", # Bucket 设置为点播专业版应用中的存储桶 ID
        Key="a/b/c.jpeg"    # 媒体文件在存储桶中的路径
    )
    print(response)
except ClientError as e:
    print(f"Error: {e}")
```

存储桶内网域名

专业版应用为每个存储桶预置了一个内网域名，客户可使用该域名在腾讯云相同地域的服务器通过内网对存储桶内的媒体文件进行增删改查，支持文件下载且不会产生流量费用，具体支持的操作清单请参见 支持的存储操作 。

> **注意：**
> - 存储桶内网域名有一定的生效时间，一般在存储桶创建后 30 ~ 120 分钟生效。
> - 内网域名必须在与存储桶相同地域的腾讯云服务器上才能访问，否则会得到 404 响应码。可以尝试在服务器上使用 `nslookups` 等指令进行域名解析，若得到形如 `10.*.*.*` 、`100.*.*.*` 或 `169.254.*.*` 等形式的 IP，则一般可以进行内网访问。

准备工作

1. 创建应用和存储桶。

   在 云点播控制台 创建好专业版应用和存储桶，具体操作步骤参见 快速入门 和 创建存储桶 文档。
2. 获取密钥对。

   从专业版应用的密钥管理中，获取密钥对 `AccessKeyId` 和 `SecretAccessKey` ，具体操作步骤参见 密钥管理 文档。
3. 准备服务器。

准备好与存储桶相同地域的腾讯云服务器，如 云服务器 、轻量应用服务器 、容器服务 等。

使用 S3 SDK 访问存储桶

下面介绍在常用的编程语言中，如何进行适配以通过内网访问专业版应用的存储桶。
假设存储地域为 `ap-guangzhou` ，存储桶 ID 为 `bucketid1` ，常用语言初始化 S3 客户端并获取对象元数据的代码实现如下所示：

GO

```
import (
    "context"
    "errors"
    "fmt"
    "net/url"
    "os"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/credentials"
    "github.com/aws/aws-sdk-go-v2/service/s3"
    smep "github.com/aws/smithy-go/endpoints"
    "github.com/aws/smithy-go/logging"
)

type customEndpointResolverV2 struct {
}

// ResolveEndpoint 自定义接入点
func (r *customEndpointResolverV2) ResolveEndpoint(ctx context.Context,
    params s3.EndpointParameters) (smep.Endpoint, error) {
    if params.Bucket == nil || params.Region == nil {
        return smep.Endpoint{}, errors.New("invalid endpoint param")
    }
    return smep.Endpoint{
        URI: url.URL{
            Scheme: "https",
            Host:   fmt.Sprintf("%s.vodsrc-internal.%s.eovod.com",
*params.Bucket, *params.Region),
        },
    }, nil
}

func main() {
```

```
    // 初始化客户端
    s3cli := s3.New(s3.Options{
        Credentials: credentials.NewStaticCredentialsProvider(
            "AccessKeyId",     // 填入密钥对中的 AccessKeyId
            "SecretAccessKey", // 填入密钥对中的 SecretAccessKey
            ""),
        EndpointResolverV2: new(customEndpointResolverV2),         // 自定义访问域名
        UsePathStyle:       false,                                 // 请求禁用 path style
        Logger:             logging.NewStandardLogger(os.Stdout), // 日志打印到标准输出流
        ClientLogMode:      aws.LogRequest | aws.LogResponse,      // 打印请求头和响应头
        Region:             "ap-guangzhou",                        // 存储地域
    })
    // 获取媒体文件的元数据
    _, _ = s3cli.HeadObject(context.TODO(), &s3.HeadObjectInput{
        Bucket: aws.String("bucketid1"),  // Bucket 设置为点播专业版应用中的存储桶 ID
        Key:    aws.String("a/b/c.jpeg"), // 媒体文件在存储桶中的路径
    })
}
```

Java

```
import software.amazon.awssdk.auth.credentials.AwsBasicCredentials;
import
software.amazon.awssdk.auth.credentials.StaticCredentialsProvider;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.HeadObjectRequest;
import software.amazon.awssdk.services.s3.model.HeadObjectResponse;

import java.net.URI;

public class Main {
    // 自定义端点解析器
    public static void main(String[] args) {
        // 初始化 S3 客户端
        S3Client s3Client = S3Client.builder()
```

```
.credentialsProvider(StaticCredentialsProvider.create(AwsBasicCredential
s.create(
                        "AccessKeyId",     // 填入密钥对中的 AccessKeyId
                        "SecretAccessKey" // 填入密钥对中的 SecretAccessKey
                )))
                .endpointOverride(URI.create("https://vodsrc-
internal.ap-guangzhou.eovod.com"))
                .region(Region.of("ap-guangzhou"))
                .build();

        // 获取媒体文件的元数据
        try {
            HeadObjectRequest headObjectRequest =
HeadObjectRequest.builder()
                    .bucket("bucketid1")     // Bucket 设置为点播专业版应用中
的存储桶 ID
                    .key("a/b/c.jpeg")       // 媒体文件在存储桶中的路径
                    .build();
            HeadObjectResponse headObjectResponse =
s3Client.headObject(headObjectRequest);
            System.out.println("Content Length: " +
headObjectResponse.contentLength());
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            s3Client.close();
        }
    }
}
```

C++

```
#include <aws/core/Aws.h>
#include <aws/core/auth/AWSCredentials.h>
#include <aws/s3/S3Client.h>
#include <aws/s3/model/HeadObjectRequest.h>
#include <iostream>
#include <string>

int main() {
    // 初始化 AWS SDK
```

```
    const Aws::SDKOptions options;
    InitAPI(options);
    // 自定义访问域名
    Aws::Client::ClientConfiguration clientConfig;
    clientConfig.scheme = Aws::Http::Scheme::HTTPS;
    clientConfig.endpointOverride = "vodsrc-internal.ap-
guangzhou.eovod.com";
    // 创建 S3 客户端
    const Aws::S3::S3Client s3Client(
        Aws::Auth::AWSCredentials(
            "AccessKeyId", // 填入密钥对中的 AccessKeyId
            "SecretAccessKey" // 填入秘钥对中的 SecretAccessKey
        ),
        nullptr, clientConfig);
    // 创建 HeadObject 请求
    Aws::S3::Model::HeadObjectRequest request;
    request.SetBucket("bucketid1"); // Bucket 设置为点播专业版应用中的存储桶
ID
    request.SetKey("a/b/c.jpeg"); // 媒体文件在存储桶中的路径
    // 发送请求
    auto outcome = s3Client.HeadObject(request);
    if (outcome.IsSuccess()) {
        std::cout << "Head object succeeded!" << std::endl;
        const auto &object = outcome.GetResult();
        std::cout << "Content Length: " << object.GetContentLength() <<
std::endl;
    } else {
        const auto error = outcome.GetError();
        std::cout << "Error: " << error.GetMessage() << std::endl;
    }
    // 清理 AWS SDK
    Aws::ShutdownAPI(options);
    return 0;
}
```

Python

```
import boto3
from botocore.config import Config
from botocore.exceptions import ClientError
```

```
# 创建 S3 客户端
s3_client = boto3.client(
    's3',
    aws_access_key_id='AccessKeyId',                                    # 填入
密钥对中的 AccessKeyId
    aws_secret_access_key='SecretAccessKey',                            # 填入
秘钥对中的 SecretAccessKey
    endpoint_url='https://vodsrc-internal.ap-guangzhou.eovod.com', # 自定
义访问域名
    config=Config(s3={'addressing_style': 'virtual'}),      # 使用 virtual
hosted-style
)

try:
    # 获取媒体文件的元数据
    response = s3_client.head_object(
        Bucket="bucketid1", # Bucket 设置为点播专业版应用中的存储桶 ID
        Key="a/b/c.jpeg"     # 媒体文件在存储桶中的路径
    )
    print(response)
except ClientError as e:
    print(f"Error: {e}")
```

EdgeOne 加速域名

专业版应用支持使用腾讯云 EdgeOne 进行媒体内容的加速分发。客户可以在 EdgeOne 中将域名源站配置为云点播专业版应用，即可使用自定义域名通过 EdgeOne 强大的安全加速网络，迅捷地下载或者播放媒体文件。
在 EdgeOne 中为专业版应用添加加速域名的操作方法请参见 EdgeOne 文档 VOD 源站指引 和 添加加速域名。

存储策略

最近更新时间：2025-03-14 11:57:23

概述

如果您需要定期调整指定文件的 存储类型 或者删除文件来降低成本，云点播专业版提供了存储策略能力（对应 AWS S3 的生命周期能力）。

配置元素

一个存储策略包含以下元素：存储策略名称、生效范围、触发条件、执行动作。

存储策略名称

仅支持中文、英文、数字、空格、_（下划线）、-（连字符）和 .（小数点），最长 20 个字符。

生效范围

存储策略当前支持**指定存储桶集合**和**全部存储桶**两类生效范围。

- 指定存储桶集合：存储策略仅生效于指定存储桶。
- 全部存储桶：应用内全部存储桶均生效，包括后续新增存储桶。

> ⚠ **注意：**
> 如果存储策略的生效范围是**全部存储桶**，则该存储策略会**自动对应用内新增的存储桶生效**。

触发条件

存储策略会对生效范围内，符合触发条件的所有文件进行操作。触发条件支持基于文件最后的修改时间和 文件键 前缀进行设置。

执行动作

存储策略支持的执行动作有：转为低频存储、转为归档存储、转为深度归档存储和删除文件。

修改存储类型

转为低频存储、转为归档存储、转为深度归档存储都属于修改文件的 存储类型 。
修改存储类型是单向的，只允许修改方向为标准存储 > 低频存储 > 归档存储 > 深度归档存储，也支持跳级沉降（例如标准存储 > 归档存储），不支持逆向。

> ⚠ **注意：**
> - 如果有多个修改存储类型的存储策略对同一个文件生效，会优先执行修改至最冷存储类型的存储策略。例如，存储策略 A 配置了文件修改 30 天后转为低频存储，存储策略 B 配置了文件修改 30 天后转为深

度归档存储，都命中了 hello.txt，则存储策略 B 生效。

- 强烈建议不要配置包含冲突触发条件的存储策略，防止预期外的行为和计费表现。
- 存储策略不会修改小于 64 KB 的文件的存储类型。
- 修改存储类型不会改变文件的修改时间。

删除文件

存储策略删除文件时，会先把文件加入异步的删除队列，因此存储策略的执行时间和文件最终的删除时间会有延迟。

注意：

- 如果有多个删除文件的存储策略对同一个文件生效，会优先执行过期时间最短的存储策略。例如存储策略 A 配置了文件修改 7 天后删除，存储策略 B 配置了文件修改 30 天后删除，都命中了 hello.txt，则 hello.txt 会在修改时间 7 天后就被删除，即存储策略 A 生效。
- 如果删除文件的存储策略和修改存储类型的存储策略对同一个文件生效，删除文件的优先级高于修改存储类型。例如存储策略 A 配置了文件修改 30 天后转为低频存储，存储策略 B 配置了文件修改 30 天后删除，都命中了 hello.txt，则 hello.txt 在修改时间 30 天后会被直接删除，即存储策略 B 生效。
- 强烈建议不要配置包含冲突触发条件的存储策略，防止预期外的行为和计费表现。

使用方法

存储策略相关操作仅支持在控制台进行，详见 控制台指南 。

上传
上传综述

最近更新时间：2025-06-13 16:30:52

您可以上传任何类型的文件到云点播专业版（以下简称专业版）支持地域的存储桶。文件完成上传后，专业版将以对象的方式管理文件。
专业版应用支持多地域的存储桶，且每个存储桶可管理的对象数量无上限。文件上传到云点播专业版后，您可以基于对象做处理并使用您的 EdgeOne 域名进行全球加速分发。
专业版兼容 AWS S3 上传协议，您可以使用 S3 SDK 上传文件到专业版存储桶。使用专业版上传具备以下优势：

- EdgeOne 全球上传加速：控制台、服务端和客户端上传均默认使用 EdgeOne 上传加速。
- 应用级就近上传：应用级上传域名支持就近上传至您专业版应用下的存储桶。

上传方式

云点播支持以下几种上传方式：

上传方式	说明
控制台上传	通过云点播专业版控制台将本地文件上传到专业版存储。 • 适用场景：开发者直接管理少量文件。 • 主要优点：方便快捷、无技术门槛。 • 大小限制：单个对象最大512GB。

服务端上传	开发者通过 API 或 SDK 方式将存储在服务器中的文件上传到专业版存储。
	• 适用场景：自动化、系统化的生产环境。
	• 主要优点：全球上传加速。
	• 大小限制：
	  ○ 简单上传：单个对象最大5GB；
	  ○ 分块上传：单个对象最大48.82TB，块大小为1MB – 5GB，最后一个块可小于1MB，分块数1 – 10000来上传。

客户端上传	终端通过 API 或 SDK 方式将客户端本地文件上传到专业版存储。
	• 适用场景：UGC、PGC 等。
	• 主要优点：全球上传加速、应用内就近上传。
	• 大小限制：
	  ○ 简单上传：单个对象最大5GB；
	  ○ 分块上传：单个对象最大48.82TB，块大小为1MB – 5GB，最后一个块可小于1MB，分块数1 – 10000来上传。

内网上传	开发者可通过腾讯云资源内网上传文件至同地域的专业版存储。

- 适用场景：内网上传拥有同地域腾讯云计算资源，可直接通过内网管理文件。
- 主要优点：内网安全访问、上传和分发均无流量费用。
- 大小限制：
  - 简单上传：单个对象最大5GB；
  - 分块上传：单个对象最大48.82TB，块大小为1MB – 5GB，最后一个块可小于1MB，分块数1 – 10000来上传。

服务端上传
服务端上传指引

最近更新时间：2025-06-13 16:30:52

本文主要提供专业版应用的服务端上传指引。

适用场景

开发者通过 API 或 SDK 方式将存储在其后台服务器中的文件上传到专业版存储，本文将介绍如何使用 AWS S3 SDK 上传文件。

上传方式

服务端需通过专业版预置域名上传，支持方式如下：

上传方式	上传域名	上传凭证	支持操作
存储桶预置域名	`[BucketId].vodpro.[存储地域].eovod.com`	永久密钥（推荐）	上传至指定存储桶
		临时凭证	上传至指定存储桶
应用预置域名	`[SubAppId].vodpro-upload.com`	临时凭证	• 上传至指定存储桶 • 就近上传至应用内某区域的存储桶

存储桶预置域名

专业版为每个存储桶预置了一个公网访问域名，可使用该域名上传文件至指定存储桶。

使用永久密钥通过 AWS S3 SDK 上传文件

下文介绍在常用的编程语言中，如何使用永久密钥上传文件至专业版应用的存储桶。
假设专业版应用 ID 为 `1234567890` ，存储桶的存储地域为 `ap-guangzhou` ，存储桶 ID 为 `bucketid1` 。
本示例使用的存储桶公网访问域名为： `bucketid1.vodpro.ap-guangzhou.eovod.com` 。

准备工作

1. 创建专业版应用和存储桶

   在 云点播控制台 创建好专业版应用和存储桶，具体操作步骤参见 快速入门 和 创建存储桶 文档。
2. 获取应用级别永久密钥对

   从专业版应用的密钥管理中，获取密钥对 `AccessKeyId` 和 `SecretAccessKey` ，具体操作步骤参见 密钥管理 文档。

上传示例

常用语言初始化 S3 客户端并上传文件，代码实现如下所示：

GO

```
// Package main
// 本示例基于 AWS SDK for Go v2 service/s3 v1.72.3 实现。
// AWS SDK for Go v2 service/s3 v1.73.0 及以后版本需禁用 S3 的默认完整性保护。
// 具体参考: https://github.com/aws/aws-sdk-go-v2/discussions/2960
package main

import (
    "context"
    "errors"
    "fmt"
    "log"
    "net/url"
    "os"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/credentials"
    "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
    "github.com/aws/aws-sdk-go-v2/service/s3"
    smep "github.com/aws/smithy-go/endpoints"
    "github.com/aws/smithy-go/logging"
)


// customEndpointResolverV2 自定义接入点
type customEndpointResolverV2 struct{}

// ResolveEndpoint 自定义接入点
func (r *customEndpointResolverV2) ResolveEndpoint(ctx context.Context,
    params s3.EndpointParameters) (smep.Endpoint, error) {
    if params.Bucket == nil || params.Region == nil {
        return smep.Endpoint{}, errors.New("invalid endpoint param")
    }
    return smep.Endpoint{
        URI: url.URL{
            Scheme: "https",
            Host:   fmt.Sprintf("%s.vodpro.ap-guangzhou.eovod.com",
*params.Bucket),
```

```
        },
    }, nil
}

func main() {
    // new s3 client
    s3cli := s3.New(s3.Options{
        Credentials: credentials.NewStaticCredentialsProvider(
            "AccessKeyId",     // 填入密钥对中的 AccessKeyId
            "SecretAccessKey", // 填入密钥对中的 SecretAccessKey
            ""),               // 永久密钥无需填充 SessionToken
        EndpointResolverV2: new(customEndpointResolverV2),         // 自定义接入点
        UsePathStyle:       false,                                 // 请求禁用 path style
        Logger:             logging.NewStandardLogger(os.Stdout), // 日志打印到标准输出流
        ClientLogMode:      aws.LogRequest | aws.LogResponse,      // 打印请求头和响应头
        Region:             "auto",                                // 固定填写 auto
    })

    // 打开本地文件
    file, err := os.Open("demo.mp4") // 本地文件路径
    if err != nil {
        log.Fatalf("failed to open file: %v", err)
        return
    }
    defer file.Close()

    // 上传文件到 S3
    // 创建上传器
    uploader := manager.NewUploader(s3cli, func(u *manager.Uploader) {
        u.PartSize = 10 * 1024 * 1024 // 设置分片大小为 10MB
        u.Concurrency = 5             // 设置并发上传的分片数
    })
    // 上传文件
    result, err := uploader.Upload(context.TODO(), &s3.PutObjectInput{
        Bucket: aws.String("bucketid1"),        // Bucket 设置为点播专业版应用中的存储桶 ID
        Key:    aws.String("upload/demo.mp4"), // 媒体文件在存储桶中的路径
        Body:   file,
```

```
    })

    if err != nil {
        log.Fatalf("failed to upload file: %v", err)
        return
    }
    log.Printf("etag: %s", *result.ETag) // 获取上传文件 etag
}O
```

Java

```
// 本示例基于 AWS SDK for Java v2 service/s3 v2.31.35 实现。
// AWS SDK for Java 2.30.0 及以后版本需禁用 S3 的默认完整性保护。
// 具体参考: https://github.com/aws/aws-sdk-java-v2/issues/5801
import software.amazon.awssdk.auth.credentials.AwsBasicCredentials;
import
software.amazon.awssdk.auth.credentials.StaticCredentialsProvider;
import software.amazon.awssdk.http.nio.netty.NettyNioAsyncHttpClient;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3AsyncClient;
import software.amazon.awssdk.services.s3.S3Configuration;
import software.amazon.awssdk.transfer.s3.S3TransferManager;
import software.amazon.awssdk.transfer.s3.model.UploadFileRequest;
import
software.amazon.awssdk.transfer.s3.progress.LoggingTransferListener;
import software.amazon.awssdk.core.checksums.RequestChecksumCalculation;
import software.amazon.awssdk.core.checksums.ResponseChecksumValidation;

import java.io.File;
import java.net.URI;
import java.util.concurrent.CompletableFuture;

public class Main {
    public static void main(String[] args) {
        S3AsyncClient s3AsyncClient = null;
        S3TransferManager transferManager = null;

        try {
            // 专业版应用永久密钥 AccessKeyId
            AwsBasicCredentials credentials =
AwsBasicCredentials.create(
                    "AccessKeyId", // 专业版应用永久密钥 AccessKeyId
```

```
                    "SecretAccessKey" // 专业版应用永久密钥 SecretAccessKey
            );
            String region = "ap-guangzhou"; // 专业版应用存储桶地域
            String bucketId = "bucketid1"; // 专业版应用存储桶ID
            String filePath = "demo.mp4"; // 本地文件路径
            String key = "upload/demo.mp4"; // 文件在存储桶中的路径
            String endpointUrl =
String.format("https://vodpro.%s.eovod.com", region); // 桶级别预置域名端点

            // 使用自定义端点构建S3异步客户端
            s3AsyncClient = S3AsyncClient.builder()

.credentialsProvider(StaticCredentialsProvider.create(credentials)) // 设
置凭证提供者
                    .endpointOverride(URI.create(endpointUrl)) // 设置端点
                    .region(Region.of("auto")) // 固定填写 auto

.httpClient(NettyNioAsyncHttpClient.builder().build()) // 设置HTTP客户端
                    .serviceConfiguration(S3Configuration.builder()
                            .pathStyleAccessEnabled(false) // 设置路径样式
访问，桶级别需明确禁用 PathStyle 模式
                            .build())

.requestChecksumCalculation(RequestChecksumCalculation.WHEN_REQUIRED)

.responseChecksumValidation(ResponseChecksumValidation.WHEN_REQUIRED)
                    .build();

            // 使用S3异步客户端创建S3TransferManager
            transferManager = S3TransferManager.builder()
                    .s3Client(s3AsyncClient)
                    .build();

            File fileToUpload = new File(filePath);

            // 创建上传请求并添加传输监听器用于进度跟踪
            UploadFileRequest uploadRequest =
UploadFileRequest.builder()
                    .putObjectRequest(builder -> builder
                            .bucket(bucketId)
                            .key(key))

.addTransferListener(LoggingTransferListener.create())
```

```
                    .source(fileToUpload)
                    .build();

            // 开始上传
            CompletableFuture<Void> future =
transferManager.uploadFile(uploadRequest)
                    .completionFuture()
                    .thenAccept(response -> {
                        System.out.println("上传成功! ETag: " +
response.response().eTag());
                    });

            // 等待上传完成
            future.join();

            System.out.println("上传任务完成");
        } catch (Exception e) {
            System.err.println("上传失败: " + e.getMessage());
            e.printStackTrace();
        } finally {
            // 资源释放
            try {
                if (transferManager != null) {
                    transferManager.close();
                }
                if (s3AsyncClient != null) {
                    s3AsyncClient.close();
                }
                System.out.println("资源已释放");
            } catch (Exception e) {
                System.err.println("关闭资源时发生错误: " +
e.getMessage());
            }
        }
    }
}
```

C++

```
// 本示例基于 AWS SDK for C++ v1.11.560 实现。
// AWS SDK for C++ v1.11.486 及以后版本需禁用 S3 的默认完整性保护。
// 具体参考: https://github.com/aws/aws-sdk-cpp/issues/3253
```

```
#include <aws/core/Aws.h>
#include <aws/core/auth/AWSCredentials.h>
#include <aws/core/client/ClientConfiguration.h>
#include <aws/s3/S3Client.h>
#include <aws/transfer/TransferManager.h>
#include <iostream>

int main()
{
    // AWS SDK 初始化
    Aws::SDKOptions options;
    options.loggingOptions.logLevel =
Aws::Utils::Logging::LogLevel::Info;
    Aws::InitAPI(options);

    {
        // 定义认证和端点信息
        const Aws::String accessKeyId = "AccessKeyId";          // 填入点播
专业版密钥对中的 AccessKeyId
        const Aws::String secretAccessKey = "SecretAccessKey"; // 填入点播
专业版密钥对中的 SecretAccessKey
        const Aws::String region = "ap-guangzhou";              // 存储桶所
在地域
        const Aws::String bucketId = "bucketid1";               // 点播专业
版应用中的存储桶 ID
        const Aws::String keyName = "upload/demo.mp4";          // 文件在存
储桶中的路径
        const Aws::String filePath = "demo.mp4";                // 本地文件
路径

        // 配置客户端
        Aws::Client::ClientConfiguration clientConfig;
        clientConfig.region = "auto";
// 固定填写 auto
        clientConfig.scheme = Aws::Http::Scheme::HTTPS;
// 协议使用 HTTPS
        clientConfig.enableEndpointDiscovery = false;
// 禁用 endpoint discovery
        clientConfig.verifySSL = true;
// 启用 SSL 证书验证
        clientConfig.endpointOverride = "vodpro." + region +
".eovod.com"; // 桶级别预置域名
        // 禁用完整性校验
```

```
        clientConfig.checksumConfig.requestChecksumCalculation =
Aws::Client::RequestChecksumCalculation::WHEN_REQUIRED;
        clientConfig.checksumConfig.responseChecksumValidation =
Aws::Client::ResponseChecksumValidation::WHEN_REQUIRED;

        // 设置认证信息
        Aws::Auth::AWSCredentials credentials(accessKeyId,
secretAccessKey); // 填入点播专业版永久密钥

        // 创建 S3 客户端
        auto s3Client = Aws::MakeShared<Aws::S3::S3Client>(
            "S3Client",
            credentials,
            clientConfig,
            Aws::Client::AWSAuthV4Signer::PayloadSigningPolicy::Never,
// 禁用 payload signing
            true
// 桶级别预置域名需明确使用 VirtualAddressing
        );

        // 创建线程池执行器
        auto executor =
Aws::MakeShared<Aws::Utils::Threading::PooledThreadExecutor>("executor",
10);

        // 配置 TransferManager
        Aws::Transfer::TransferManagerConfiguration
transferConfig(executor.get());
        transferConfig.s3Client = s3Client;

        // 设置分片大小为 10MB
        transferConfig.bufferSize = 10 * 1024 * 1024;

        // 创建 TransferManager
        auto transferManager =
Aws::Transfer::TransferManager::Create(transferConfig);

        std::cout << "Starting file upload: " << filePath << " to " <<
bucketId << "/" << keyName << std::endl;

        // 执行上传
        auto uploadHandle = transferManager->UploadFile(
            filePath,                               // 本地文件路径
```

```
            bucketId,                                   // 存储桶ID
            keyName,                                    // 对象键（存储路径）
            "application/octet-stream",                 // 内容类型
            Aws::Map<Aws::String, Aws::String>() // 元数据
        );

        // 等待上传完成
        uploadHandle->WaitUntilFinished();

        // 检查上传状态
        if (uploadHandle->GetStatus() ==
Aws::Transfer::TransferStatus::COMPLETED)
        {
            std::cout << "File upload completed successfully!" <<
std::endl;
        }
        else
        {
            auto lastError = uploadHandle->GetLastError();
            std::cerr << "File upload failed: " <<
lastError.GetMessage() << std::endl;
        }
    }

    // 清理SDK资源
    Aws::ShutdownAPI(options);
    return 0;
}
```

Python

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
本示例基于 AWS SDK for Python v1.38.7 实现。
Boto3 的 AWS SDK for Python v1.36.0 及以后版本需禁用 S3 的默认完整性保护。
具体参考：https://github.com/boto/boto3/issues/4392
"""

import boto3
from botocore.config import Config
```

```python
from botocore.exceptions import ClientError

# 常量定义
REGION = "ap-guangzhou"  # 存储桶地域
BUCKET_ID = "bucketid1"  # 填入点播专业版应用中的存储桶 ID
FILE_PATH = "demo.mp4"  # 本地文件路径
OBJECT_KEY = "upload/demo.mp4"  # 文件在存储桶中的路径

# 创建 S3 客户端
s3_client = boto3.client(
    "s3",
    aws_access_key_id="AccessKeyId",  # 填入密钥对中的 AccessKeyId
    aws_secret_access_key="SecretAccessKey",  # 填入密钥对中的
SecretAccessKey
    endpoint_url=f"https://vodpro.{REGION}.eovod.com",  # 桶级别预置上传域
名
    region_name="auto",  # 固定填写 auto
    config=Config(
        s3={"addressing_style": "virtual"},  # 使用 virtual hosted-style
        request_checksum_calculation="when_required",  # Boto3 的 AWS SDK
for Python v1.36.0 及以后版本需禁用 S3 的默认完整性保护。
        response_checksum_validation="when_required",  # Boto3 的 AWS SDK
for Python v1.36.0 及以后版本需禁用 S3 的默认完整性保护。
    ),
)

try:
    # 上传文件
    response = s3_client.upload_file(
        Bucket=BUCKET_ID,  # 填入点播专业版应用中的存储桶 ID
        Key=OBJECT_KEY,  # 文件在存储桶中的路径
        Filename=FILE_PATH,  # 本地文件路径
    )
    print(response)
except ClientError as e:
    print(f"Error: {e}")
```

⚠ 注意：

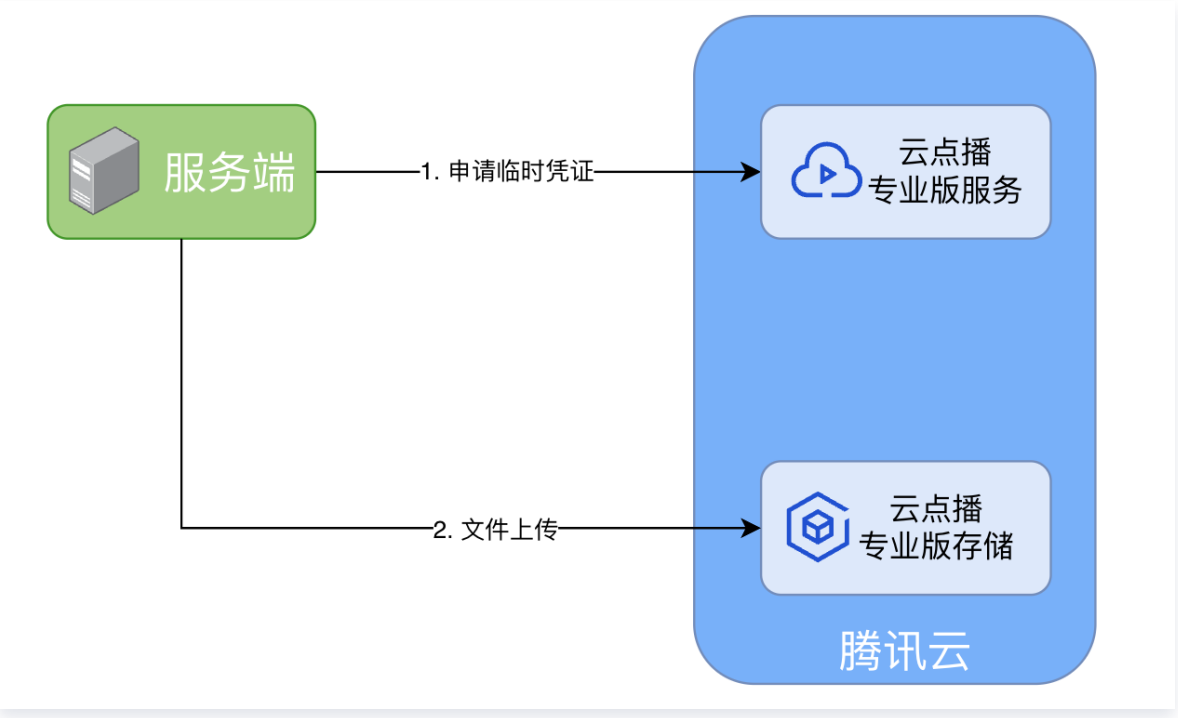
- 使用 AWS SDK 请确认数据完整性保护特性，详情请参考 AWS SDK 的相关文档 Data Integrity Protections for Amazon S3。
- 使用存储桶预置公网域名，路径必须为 VirtualStyle 格式。

使用临时凭证通过 AWS S3 SDK 上传文件

下面介绍在常用的编程语言中，如何使用临时凭证上传文件至专业版应用的存储桶。
假设专业版应用 ID 为 `1234567890` ，存储桶的存储地域为 `ap-guangzhou` ，存储桶 ID 为 `bucketid1` 。
本示例使用的存储桶访问域名为： `bucketid1.vodpro.ap-guangzhou.eovod.com` 。

上传步骤

临时凭证通常用于对密钥安全性要求较高的场景。服务端使用临时凭证上传文件至专业版存储示意图如下：



1. 申请上传临时凭证：服务端调用云点播专业版服务的 创建应用存储临时访问凭证 接口。
2. 上传文件：服务端使用临时凭证上传文件至云点播专业版存储。

准备工作

1. 创建专业版应用和存储桶

   在 云点播控制台 创建好专业版应用和存储桶，具体操作步骤参见 快速入门 和 创建存储桶 文档。
2. 获取腾讯云账号永久密钥对

   2.1 登录腾讯云控制台，选择访问管理 > 访问密钥 > API 密钥管理，进入“API 密钥管理” 页面。

   2.2 获取云 API 密钥。如果您尚未创建密钥，则单击新建密钥即可创建一对 `SecretId` 和 `SecretKey` 。

上传文件

申请上传临时凭证

申请 `bucketid1` 存储桶上传 `upload/demo.mp4` 的临时凭证。

GO

```go
// Package main
package main

import (
    "context"
    "encoding/json"
    "fmt"
    "log"
    "net/url"

    "github.com/tencentcloud/tencentcloud-sdk-go/tencentcloud/common"
    "github.com/tencentcloud/tencentcloud-sdk-
go/tencentcloud/common/profile"
    vod20240718 "github.com/tencentcloud/tencentcloud-sdk-
go/tencentcloud/vod/v20240718"
)

const (
    appId    = 251000000          // 腾讯云账号 APPID
    subAppId = 1234567890         // 云点播专业版应用 APPID
    bucketId = "bucketid1"        // 云点播专业版应用存储桶的 ID
    fileKey  = "upload/demo.mp4" // 上传到存储后的文件 KEY，包含完整路径和文件
名，例如 upload/demo.mp4
)

// createStorageCredentialsPolicy 创建存储凭证策略
type createStorageCredentialsPolicy struct {
    Statement []policyStatement `json:"statement"`
    Version   string            `json:"version"`
}

// policyStatement 策略语句
type policyStatement struct {
    Action   []string `json:"action"`
    Effect   string   `json:"effect"`
    Resource []string `json:"resource"`
}

// cred 临时凭证
type cred struct {
    AccessKeyId     string
```

```
    SecretAccessKey string
    SessionToken    string
}

// getCredential 获取临时凭证
func getCredential(context.Context) (*cred, error) {
    // 1. 获取临时凭证
    // 1.1 初始化调用腾讯云 API 对象
    credential := common.NewCredential(
        "SecretId",  // 腾讯云账号 SecretId
        "SecretKey", // 腾讯云账号 SecretKey
    )
    prof := profile.NewClientProfile()
    vodClient, err := vod20240718.NewClient(credential, "ap-guangzhou",
prof)
    if err != nil {
        log.Fatalf("create VOD client fail: %+v", err)
        return nil, fmt.Errorf("create VOD client fail: %w", err)
    }

    // 1.2 构造申请上传临时凭证请求
    policy := createStorageCredentialsPolicy{
        Statement: []policyStatement{{
            Action: []string{ // 当前仅支持如下 action，可以只申请其中一部分
                "name/vod:PutObject",
                "name/vod:ListParts",
                "name/vod:PostObject",
                "name/vod:CreateMultipartUpload",
                "name/vod:UploadPart",
                "name/vod:CompleteMultipartUpload",
                "name/vod:AbortMultipartUpload",
                "name/vod:ListMultipartUploads",
            },
            Effect: "allow",
            Resource: []string{
                fmt.Sprintf("qcs::vod:%s:uid/%d:prefix//%d/%s/%s",
                    "ap-guangzhou", // 存储桶所在地域
                    appId,          // 腾讯云账号 APPID
                    subAppId,       // 云点播专业版应用 APPID
                    bucketId,       // 云点播专业版应用存储桶的 ID
                    fileKey,        // 上传到存储后的文件 KEY，包含完整路径和
文件名，例如 upload/demo.mp4
                ),
```

```
            },
        }},
        Version: "2.0",
    }
    req := vod20240718.NewCreateStorageCredentialsRequest()
    req.SubAppId = common.Uint64Ptr(subAppId)
    policyStr, _ := json.Marshal(policy)
    req.Policy = common.StringPtr(url.QueryEscape(string(policyStr)))

    // 1.3 申请上传临时凭证
    resp, err := vodClient.CreateStorageCredentials(req)
    if err != nil {
        log.Fatalf("create storage credentials fail: %+v", err)
        return nil, fmt.Errorf("create storage credentials fail: %w",
err)
    }
    log.Printf("create storage credentials success: %+v", resp)
    creds := resp.Response.Credentials
    return &cred{
        AccessKeyId:     *creds.AccessKeyId,
        SecretAccessKey: *creds.SecretAccessKey,
        SessionToken:    *creds.SessionToken,
    }, nil
}
```

Java

```
import com.tencentcloudapi.common.Credential;
import com.tencentcloudapi.common.exception.TencentCloudSDKException;
import com.tencentcloudapi.common.profile.ClientProfile;
import com.tencentcloudapi.vod.v20240718.VodClient;
import
com.tencentcloudapi.vod.v20240718.models.CreateStorageCredentialsRequest
;
import
com.tencentcloudapi.vod.v20240718.models.CreateStorageCredentialsRespons
e;

import org.json.JSONArray;
import org.json.JSONObject;

import java.net.URLEncoder;
```

```
import java.nio.charset.StandardCharsets;

/**
 * 凭证辅助类，负责获取临时存储凭证
 */
public class CredentialHelper {
    // 常量定义
    private static final long APP_ID = 251000000; // 腾讯云账号 APPID
    private static final long SUB_APP_ID = 1234567890; // 云点播专业版应用
APPID
    private static final String BUCKET_ID = "bucketid1"; // 云点播专业版应
用存储桶的 ID
    private static final String FILE_KEY = "upload/demo.mp4"; // 上传到存
储后的文件 KEY
    private static final String REGION = "ap-guangzhou"; // 存储桶所在地域

    /**
     * 凭证对象，存储临时凭证信息
     */
    public static class Cred {
        private final String accessKeyId;
        private final String secretAccessKey;
        private final String sessionToken;

        public Cred(String accessKeyId, String secretAccessKey, String
sessionToken) {
            this.accessKeyId = accessKeyId;
            this.secretAccessKey = secretAccessKey;
            this.sessionToken = sessionToken;
        }

        public String getAccessKeyId() {
            return accessKeyId;
        }

        public String getSecretAccessKey() {
            return secretAccessKey;
        }

        public String getSessionToken() {
            return sessionToken;
        }
    }
```

```
    /**
     * 获取临时凭证
     *
     * @return 临时凭证对象
     * @throws Exception 如果获取凭证失败
     */
    public static Cred getCredential() throws Exception {
        try {
            // 1. 初始化腾讯云 API 客户端
            Credential credential = new Credential("SecretId",
"SecretKey"); // 腾讯云账号 SecretId 和 SecretKey
            ClientProfile clientProfile = new ClientProfile(); // 客户端配
置
            VodClient vodClient = new VodClient(credential, "ap-
guangzhou", clientProfile); // 创建 VodClient 对象

            // 2. 构造并编码策略
            String policyJson = createPolicyJson();
            String encodedPolicy = URLEncoder.encode(policyJson,
StandardCharsets.UTF_8.name());

            // 3. 创建并发送请求
            CreateStorageCredentialsRequest req = new
CreateStorageCredentialsRequest();
            req.setSubAppId(SUB_APP_ID); // 云点播专业版应用 APPID
            req.setPolicy(encodedPolicy); // 策略

            // 4. 获取响应并返回凭证
            CreateStorageCredentialsResponse resp =
vodClient.CreateStorageCredentials(req);

            return new Cred(
                    resp.getCredentials().getAccessKeyId(),
                    resp.getCredentials().getSecretAccessKey(),
                    resp.getCredentials().getSessionToken());
        } catch (TencentCloudSDKException e) {
            System.err.println("获取存储凭证失败: " + e.getMessage());
            throw new Exception("获取存储凭证失败", e);
        }
    }

    /**
```

```
     * 创建策略JSON字符串，使用 org.json 库
     *
     * @return 策略JSON字符串
     */
    private static String createPolicyJson() {
        // 构建资源路径
        String resource = String.format(
                "qcs::vod:%s:uid/%d:prefix//%d/%s/%s",
                REGION,
                APP_ID,
                SUB_APP_ID,
                BUCKET_ID,
                FILE_KEY);

        // 构建操作列表
        String[] actions = {
                "name/vod:PutObject",
                "name/vod:ListParts",
                "name/vod:PostObject",
                "name/vod:CreateMultipartUpload",
                "name/vod:UploadPart",
                "name/vod:CompleteMultipartUpload",
                "name/vod:AbortMultipartUpload",
                "name/vod:ListMultipartUploads"
        };

        // 使用 JSONObject 构建 JSON
        JSONObject policy = new JSONObject();
        policy.put("version", "2.0");

        JSONArray statements = new JSONArray();
        JSONObject statement = new JSONObject();

        JSONArray actionArray = new JSONArray();
        for (String action : actions) {
            actionArray.put(action);
        }
        statement.put("action", actionArray);

        statement.put("effect", "allow");

        JSONArray resources = new JSONArray();
        resources.put(resource);
```

```
        statement.put("resource", resources);

        statements.put(statement);
        policy.put("statement", statements);

        return policy.toString();
    }
}
```

C++

```
#include <tencentcloud/core/TencentCloud.h>
#include <tencentcloud/core/profile/ClientProfile.h>
#include <tencentcloud/core/profile/HttpProfile.h>
#include <tencentcloud/core/Credential.h>
#include <tencentcloud/vod/v20240718/VodClient.h>
#include
<tencentcloud/vod/v20240718/model/CreateStorageCredentialsRequest.h>
#include
<tencentcloud/vod/v20240718/model/CreateStorageCredentialsResponse.h>
#include <string>
#include <sstream>
#include <iomanip>
#include <iostream>
#include <nlohmann/json.hpp>

using json = nlohmann::json;

const uint64_t APP_ID = 251000000;                // 腾讯云账号 APPID
const uint64_t SUB_APP_ID = 1234567890;           // 云点播专业版应用 APPID
const std::string BUCKET_ID = "bucketid1";        // 云点播专业版应用存储桶
的 ID
const std::string REGION = "ap-guangzhou";        // 云点播专业版应用存储桶
地域
const std::string OBJECT_KEY = "upload/demo.mp4"; // 申请权限的存储文件 KEY

// 凭证结构体
struct Credential
{
    std::string accessKeyId;
    std::string secretAccessKey;
    std::string sessionToken;
```

```
};

// URL 编码函数
std::string UrlEncode(const std::string &value)
{
    std::ostringstream escaped;
    escaped.fill('0');
    escaped << std::hex;

    for (char c : value)
    {
        if (isalnum(c) || c == '-' || c == '_' || c == '.' || c == '~')
        {
            escaped << c;
        }
        else
        {
            escaped << std::uppercase;
            escaped << '%' << std::setw(2) << int((unsigned char)c);
            escaped << std::nouppercase;
        }
    }

    return escaped.str();
}

// 获取临时凭证
Credential GetCredential()
{
    // 初始化腾讯云 SDK
    TencentCloud::InitAPI();

    // 创建 VOD 客户端
    TencentCloud::Credential credential("SecretId", "SecretKey"); // 填入
腾讯云账号 SecretId 和 SecretKey
    TencentCloud::HttpProfile httpProfile;
    TencentCloud::ClientProfile clientProfile;
    clientProfile.SetHttpProfile(httpProfile);

    TencentCloud::Vod::V20240718::VodClient client(credential, "ap-
guangzhou", clientProfile);

    // 构建策略
```

```
    json policy_json = {
        {"statement", {{{"action", {"name/vod:PutObject",
"name/vod:ListParts", "name/vod:PostObject",
"name/vod:CreateMultipartUpload", "name/vod:UploadPart",
"name/vod:CompleteMultipartUpload", "name/vod:AbortMultipartUpload",
"name/vod:ListMultipartUploads"}}, {"effect", "allow"}, {"resource",
{"qcs::vod:" + REGION + ":uid/" + std::to_string(APP_ID) + ":prefix//" +
std::to_string(SUB_APP_ID) + "/" + BUCKET_ID + "/" + OBJECT_KEY}}}}},
        {"version", "2.0"}};
    std::string policy = policy_json.dump();

    // 创建请求对象
    TencentCloud::Vod::V20240718::Model::CreateStorageCredentialsRequest
req;
    req.SetSubAppId(SUB_APP_ID);
    req.SetPolicy(UrlEncode(policy));

    // 发送请求
    auto outcome = client.CreateStorageCredentials(req);
    if (!outcome.IsSuccess())
    {
        std::cerr << "Failed to get storage credentials: " <<
outcome.GetError().GetErrorMessage() << std::endl;
        TencentCloud::ShutdownAPI();
        exit(1);
    }

    // 提取凭证
    auto response = outcome.GetResult();
    auto creds = response.GetCredentials();

    Credential result;
    result.accessKeyId = creds.GetAccessKeyId();
    result.secretAccessKey = creds.GetSecretAccessKey();
    result.sessionToken = creds.GetSessionToken();

    // 清理腾讯云 SDK
    TencentCloud::ShutdownAPI();

    return result;
}
```

Python

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import json
import urllib.parse
from typing import NamedTuple

from tencentcloud.common import credential
from tencentcloud.common.profile import client_profile
from tencentcloud.vod.v20240718 import vod_client, models

# 常量定义
APP_ID = 251000000  # 腾讯云账号 APPID
SUB_APP_ID = 1234567890  # 云点播专业版应用 APPID
BUCKET_ID = "bucketid1"  # 云点播专业版应用存储桶的 ID
OBJECT_KEY = "upload/demo.mp4"  # 上传到存储后的文件 KEY
REGION = "ap-guangzhou"  # 地域


class Credential(NamedTuple):
    """临时凭证"""

    access_key_id: str
    secret_access_key: str
    session_token: str


def get_credential() -> Credential:
    """获取临时凭证"""
    # 1. 初始化调用腾讯云 API 对象
    cred = credential.Credential(
        "SecretId",  # 腾讯云账号 SecretId
        "SecretKey",  # 腾讯云账号 SecretKey
    )
    prof = client_profile.ClientProfile()
    vod_cli = vod_client.VodClient(cred, "ap-guangzhou", prof)

    # 2. 构造申请上传临时凭证请求
    policy = {
        "statement": [
            {
```

```
                "action": [
                    "name/vod:PutObject",
                    "name/vod:ListParts",
                    "name/vod:PostObject",
                    "name/vod:CreateMultipartUpload",
                    "name/vod:UploadPart",
                    "name/vod:CompleteMultipartUpload",
                    "name/vod:AbortMultipartUpload",
                    "name/vod:ListMultipartUploads",
                ],
                "effect": "allow",
                "resource": [f"qcs::vod:
{REGION}:uid/{APP_ID}:prefix//{SUB_APP_ID}/{BUCKET_ID}/{OBJECT_KEY}"],
            }
        ],
        "version": "2.0",
    }

    req = models.CreateStorageCredentialsRequest()
    req.SubAppId = SUB_APP_ID
    req.Policy = urllib.parse.quote(json.dumps(policy))

    # 3. 申请上传临时凭证
    resp = vod_cli.CreateStorageCredentials(req)
    creds = resp.Credentials
    return Credential(
        access_key_id=creds.AccessKeyId,
        secret_access_key=creds.SecretAccessKey,
        session_token=creds.SessionToken,
    )
```

文件上传

常用语言初始化 S3 客户端并上传文件至存储桶，代码实现如下所示：

GO

```
// Package main
// 本示例基于 AWS SDK for Go v2 service/s3 v1.72.3 实现。
// AWS SDK for Go v2 service/s3 v1.73.0 及以后版本需禁用 S3 的默认完整性保护。
// 具体参考: https://github.com/aws/aws-sdk-go-v2/discussions/2960
package main
```

```go
import (
    "context"
    "errors"
    "fmt"
    "log"
    "net/url"
    "os"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/credentials"
    "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
    "github.com/aws/aws-sdk-go-v2/service/s3"
    smep "github.com/aws/smithy-go/endpoints"
    "github.com/aws/smithy-go/logging"
)

type customEndpointResolverV2 struct {
}

// ResolveEndpoint 自定义接入点
func (r *customEndpointResolverV2) ResolveEndpoint(ctx context.Context,
    params s3.EndpointParameters) (smep.Endpoint, error) {
    if params.Bucket == nil || params.Region == nil {
        return smep.Endpoint{}, errors.New("invalid endpoint param")
    }
    return smep.Endpoint{
        URI: url.URL{
            Scheme: "https",
            Host: fmt.Sprintf(
                "%s.vodpro.ap-guangzhou.eovod.com",
                *params.Bucket, // 填入点播专业版应用中的存储桶 ID
            ),
        },
    }, nil
}

func main() {
    // 1. 获取临时凭证
    cred, err := getCredential(context.Background())
    if err != nil {
        log.Fatalf("get credential fail: %v", err)
        return
    }
```

```
    // 2. 使用临时凭证上传文件
    // 2.1 创建 s3 client
    s3cli := s3.New(s3.Options{
        Credentials: credentials.NewStaticCredentialsProvider(
            cred.AccessKeyId,     // 填入临时凭证中的 AccessKeyId
            cred.SecretAccessKey, // 填入临时凭证中的 SecretAccessKey
            cred.SessionToken,    // 填入临时凭证中的 SessionToken
        ),
        EndpointResolverV2: new(customEndpointResolverV2),         // 自定
义接入点
        UsePathStyle:       false,                                 // 请求
禁用 path style
        Logger:             logging.NewStandardLogger(os.Stdout), // 日志
打印到标准输出流
        ClientLogMode:      aws.LogRequest | aws.LogResponse,      // 打印
请求头和响应头
        Region:             "auto",                                // 固定
填写 auto
    })

    // 2.2 打开本地文件
    file, err := os.Open("demo.mp4") // 本地文件路径
    if err != nil {
        log.Fatalf("failed to open file: %v", err)
        return
    }
    defer file.Close()

    // 2.3 上传文件到 S3
    // 2.3.1 创建上传器
    uploader := manager.NewUploader(s3cli, func(u *manager.Uploader) {
        u.PartSize = 10 * 1024 * 1024 // 设置分片大小为 10MB
        u.Concurrency = 5             // 设置并发上传的分片数
    })
    // 2.3.2 上传文件
    result, err := uploader.Upload(context.TODO(), &s3.PutObjectInput{
        Bucket: aws.String(bucketId), // Bucket 设置为点播专业版应用中的存储
桶 ID
        Key:    aws.String(fileKey),  // 媒体文件在存储桶中的路径
        Body:   file,
    })
```

```
    if err != nil {
        log.Fatalf("failed to upload file: %v", err)
        return
    }
    log.Printf("etag: %s", *result.ETag) // 获取上传文件 etag
}
```

Java

```
// 本示例基于 AWS SDK for Java v2 service/s3 v2.31.35 实现。
// AWS SDK for Java 2.30.0 及以后版本需禁用 S3 的默认完整性保护。
// 具体参考: https://github.com/aws/aws-sdk-java-v2/issues/5801
import software.amazon.awssdk.auth.credentials.AwsBasicCredentials;
import software.amazon.awssdk.auth.credentials.AwsCredentials;
import software.amazon.awssdk.auth.credentials.AwsSessionCredentials;
import
software.amazon.awssdk.auth.credentials.StaticCredentialsProvider;
import software.amazon.awssdk.http.nio.netty.NettyNioAsyncHttpClient;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3AsyncClient;
import software.amazon.awssdk.services.s3.S3Configuration;
import software.amazon.awssdk.transfer.s3.S3TransferManager;
import software.amazon.awssdk.transfer.s3.model.UploadFileRequest;
import
software.amazon.awssdk.transfer.s3.progress.LoggingTransferListener;
import software.amazon.awssdk.core.checksums.RequestChecksumCalculation;
import software.amazon.awssdk.core.checksums.ResponseChecksumValidation;

import java.io.File;
import java.net.URI;
import java.util.concurrent.CompletableFuture;

public class Main {
    public static void main(String[] args) {
        S3AsyncClient s3AsyncClient = null;
        S3TransferManager transferManager = null;

        try {
            // 1. 获取临时凭证
            CredentialHelper.Cred cred =
CredentialHelper.getCredential();
```

```
            System.out.println("获取临时凭证成功, AccessKeyId: " +
cred.getAccessKeyId());

            // 2. 创建会话凭证（包含临时 token）
            AwsSessionCredentials sessionCredentials =
AwsSessionCredentials.create(
                    cred.getAccessKeyId(), // 临时凭证 AccessKeyId
                    cred.getSecretAccessKey(), // 临时凭证 SecretAccessKey
                    cred.getSessionToken() // 临时凭证 SessionToken
            );

            String region = "ap-guangzhou"; // 专业版应用存储桶地域
            String bucketId = "bucketid1"; // 专业版应用存储桶ID
            String filePath = "demo.mp4"; // 本地文件路径
            String key = "upload/demo.mp4"; // 文件在存储桶中的路径
            String endpointUrl =
String.format("https://vodpro.%s.eovod.com", region); // 桶级别预置域名端点

            // 3. 使用自定义端点构建S3异步客户端
            s3AsyncClient = S3AsyncClient.builder()

.credentialsProvider(StaticCredentialsProvider.create(sessionCredentials
)) // 设置凭证提供者
                    .endpointOverride(URI.create(endpointUrl)) // 设置端点
                    .region(Region.of("auto")) // 设置区域 auto 自动选择

.httpClient(NettyNioAsyncHttpClient.builder().build()) // 设置HTTP客户端
                    .serviceConfiguration(S3Configuration.builder()
                            .pathStyleAccessEnabled(false) // 设置路径样式
访问, 桶级别需明确禁用 PathStyle 模式
                            .build())

.requestChecksumCalculation(RequestChecksumCalculation.WHEN_REQUIRED)

.responseChecksumValidation(ResponseChecksumValidation.WHEN_REQUIRED)
                    .build();

            // 4. 使用S3异步客户端创建S3TransferManager
            transferManager = S3TransferManager.builder()
                    .s3Client(s3AsyncClient)
                    .build();

            // 5. 准备上传文件
```

```
            File fileToUpload = new File(filePath);
            if (!fileToUpload.exists()) {
                throw new RuntimeException("文件不存在: " + filePath);
            }

            // 6. 创建上传请求并添加传输监听器用于进度跟踪
            UploadFileRequest uploadRequest =
UploadFileRequest.builder()
                    .putObjectRequest(builder -> builder
                            .bucket(bucketId)
                            .key(key))

.addTransferListener(LoggingTransferListener.create())
                    .source(fileToUpload)
                    .build();

            // 7. 开始上传
            System.out.println("开始上传文件...");
            CompletableFuture<Void> future =
transferManager.uploadFile(uploadRequest)
                    .completionFuture()
                    .thenAccept(response -> {
                        System.out.println("上传成功! ETag: " +
response.response().eTag());
                    });

            // 8. 等待上传完成
            future.join();

            System.out.println("上传任务完成");
        } catch (Exception e) {
            System.err.println("上传失败: " + e.getMessage());
            e.printStackTrace();
        } finally {
            // 资源释放
            try {
                if (transferManager != null) {
                    transferManager.close();
                }
                if (s3AsyncClient != null) {
                    s3AsyncClient.close();
                }
                System.out.println("资源已释放");
```

```
            } catch (Exception e) {
                System.err.println("关闭资源时发生错误: " +
e.getMessage());
            }
        }
    }
}
```

C++

```
// 本示例基于 AWS SDK for C++ v1.11.560 实现。
// AWS SDK for C++ v1.11.486 及以后版本需禁用 S3 的默认完整性保护。
// 具体参考: https://github.com/aws/aws-sdk-cpp/issues/3253
#include <aws/core/Aws.h>
#include <aws/core/auth/AWSCredentials.h>
#include <aws/core/client/ClientConfiguration.h>
#include <aws/s3/S3Client.h>
#include <aws/transfer/TransferManager.h>
#include <iostream>
#include <string>

// 引用 get_cred.cpp 中定义的凭证结构体
struct Credential
{
    std::string accessKeyId;
    std::string secretAccessKey;
    std::string sessionToken;
};

// 引用 get_cred.cpp 中定义的函数
extern Credential GetCredential();

int main()
{
    // AWS SDK 初始化
    Aws::SDKOptions options;
    options.loggingOptions.logLevel =
Aws::Utils::Logging::LogLevel::Info;
    Aws::InitAPI(options);

    {
        // 获取临时凭证
```

```
        std::cout << "Getting temporary credentials..." << std::endl;
        Credential cred = GetCredential();
        std::cout << "Successfully obtained temporary credentials,
access key Id: " << cred.accessKeyId << std::endl;

        // 定义端点信息
        const Aws::String region = "ap-guangzhou";      // 云点播专业版应用
存储桶所在地域
        const Aws::String bucketId = "bucketid1";       // 云点播专业版应用
存储桶的 ID
        const Aws::String keyName = "upload/demo.mp4"; // 文件在存储桶中的
路径
        const Aws::String filePath = "demo.mp4";        // 本地文件路径

        // 配置客户端
        Aws::Client::ClientConfiguration clientConfig;
        clientConfig.region = "auto";
// 固定填写 auto
        clientConfig.scheme = Aws::Http::Scheme::HTTPS;
// 协议使用 HTTPS
        clientConfig.enableEndpointDiscovery = false;
// 禁用 endpoint discovery
        clientConfig.verifySSL = true;
// 启用 SSL 证书验证
        clientConfig.endpointOverride = "vodpro." + region +
".eovod.com"; // 桶级别预置域名
        // 禁用完整性校验
        clientConfig.checksumConfig.requestChecksumCalculation =
Aws::Client::RequestChecksumCalculation::WHEN_REQUIRED;
        clientConfig.checksumConfig.responseChecksumValidation =
Aws::Client::ResponseChecksumValidation::WHEN_REQUIRED;

        // 设置认证信息（使用临时凭证）
        Aws::Auth::AWSCredentials credentials(
            cred.accessKeyId.c_str(),
            cred.secretAccessKey.c_str(),
            cred.sessionToken.c_str());

        // 创建 S3 客户端
        auto s3Client = Aws::MakeShared<Aws::S3::S3Client>(
            "S3Client",
            credentials,
            clientConfig,
```

```
            Aws::Client::AWSAuthV4Signer::PayloadSigningPolicy::Never,
// 禁用 payload signing
            true
// 桶级别预置域名需明确使用 VirtualAddressing
        );

        // 创建线程池执行器
        auto executor =
Aws::MakeShared<Aws::Utils::Threading::PooledThreadExecutor>("executor",
10);

        // 配置 TransferManager
        Aws::Transfer::TransferManagerConfiguration
transferConfig(executor.get());
        transferConfig.s3Client = s3Client;

        // 设置分片大小为 10MB
        transferConfig.bufferSize = 10 * 1024 * 1024;

        // 创建 TransferManager
        auto transferManager =
Aws::Transfer::TransferManager::Create(transferConfig);

        std::cout << "Starting file upload: " << filePath << " to " <<
bucketId << "/" << keyName << std::endl;

        // 执行上传
        auto uploadHandle = transferManager->UploadFile(
            filePath,                                  // 本地文件路径
            bucketId,                                  // 存储桶ID
            keyName,                                   // 对象键（存储路径）
            "application/octet-stream",                // 内容类型
            Aws::Map<Aws::String, Aws::String>() // 元数据
        );

        // 等待上传完成
        uploadHandle->WaitUntilFinished();

        // 检查上传状态
        if (uploadHandle->GetStatus() ==
Aws::Transfer::TransferStatus::COMPLETED)
        {
```

```
            std::cout << "File upload completed successfully!" <<
std::endl;
        }
        else
        {
            auto lastError = uploadHandle->GetLastError();
            std::cerr << "File upload failed: " <<
lastError.GetMessage() << std::endl;
        }
    }

    // 清理SDK资源
    Aws::ShutdownAPI(options);
    return 0;
}
```

Python

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
本示例基于 AWS SDK for Python v1.38.7 实现。
Boto3 的 AWS SDK for Python v1.36.0 及以后版本需禁用 S3 的默认完整性保护。
具体参考：https://github.com/boto/boto3/issues/4392
"""

import boto3
from botocore.config import Config
from botocore.exceptions import ClientError

from get_cred import get_credential, REGION, BUCKET_ID, OBJECT_KEY

# 常量定义
FILE_PATH = "demo.mp4"

try:
    # 1. 获取临时凭证
    cred = get_credential()

    # 2. 创建 S3 客户端
    s3_client = boto3.client(
```

```
        "s3",
        aws_access_key_id=cred.access_key_id,  # 临时凭证的 AccessKeyId
        aws_secret_access_key=cred.secret_access_key,  # 临时凭证的
SecretAccessKey
        aws_session_token=cred.session_token,  # 临时凭证的 SessionToken
        endpoint_url=f"https://vodpro.{REGION}.eovod.com",  # 桶级别预置上
传域名
        region_name="auto",  # 固定填写 auto
        config=Config(
            s3={"addressing_style": "virtual"},  # 使用 virtual hosted-
style
            request_checksum_calculation="when_required",  # Boto3 的 AWS
SDK for Python v1.36.0 及以后版本需禁用 S3 的默认完整性保护。
            response_checksum_validation="when_required",  # Boto3 的 AWS
SDK for Python v1.36.0 及以后版本需禁用 S3 的默认完整性保护。
        ),
    )

    # 3. 上传文件
    response = s3_client.upload_file(
        Bucket=BUCKET_ID,  # 填入点播专业版应用中的存储桶 ID
        Key=OBJECT_KEY,  # 文件在存储桶中的路径
        Filename=FILE_PATH,  # 本地文件路径
    )
    print(response)
except ClientError as e:
    print(f"Error: {e}")
```

注意：

- 使用 AWS SDK 请确认数据完整性保护特性，详情请参考 AWS SDK 的相关文档 Data Integrity Protections for Amazon S3。
- 使用存储桶预置公网域名，路径必须为 VirtualStyle 格式。

应用预置域名

专业版为每个应用预置了一个公网上传加速域名，可使用该域名上传至应用内所有存储桶。

使用临时凭证通过 AWS S3 SDK 上传文件

下面介绍在常用的编程语言中，如何进行适配以上传文件至专业版应用的存储桶。
假设专业版应用 ID 为 `1234567890` ，存储桶的存储地域为 `ap-guangzhou` ，存储桶 ID 为 `bucketid1` 。示例使用的应用上传加速域名为： `1234567890.vodpro-upload.com` 。

准备工作

1. 创建专业版应用和存储桶

   在 云点播控制台 创建好专业版应用和存储桶，具体操作步骤参见 快速入门 和 创建存储桶 文档。

2. 获取腾讯云账号永久密钥对

   2.1 登录腾讯云控制台，选择访问管理 > 访问密钥 > API 密钥管理，进入“API 密钥管理” 页面。

   2.2 获取云 API 密钥。如果您尚未创建密钥，则单击新建密钥即可创建一对 `SecretId` 和 `SecretKey`。

上传示例

申请上传临时凭证

GO

```
// Package main
package main

import (
    "context"
    "encoding/json"
    "fmt"
    "log"
    "net/url"

    "github.com/tencentcloud/tencentcloud-sdk-go/tencentcloud/common"
    "github.com/tencentcloud/tencentcloud-sdk-
go/tencentcloud/common/profile"
    vod20240718 "github.com/tencentcloud/tencentcloud-sdk-
go/tencentcloud/vod/v20240718"
)

const (
    appId    = 251000000          // 腾讯云账号 APPID
    subAppId = 1234567890         // 云点播专业版应用 APPID
    fileKey  = "upload/demo.mp4" // 申请权限的存储文件 KEY，包含完整路径和文件
名，例如 upload/demo.mp4
    bucketId = "auto"             // 云点播专业版应用存储桶的 ID，auto 表示自动
就近选择存储桶
    region   = "auto"             // 云点播专业版应用存储桶地域，auto 表示自动就
近选择地域
)

// createStorageCredentialsPolicy 创建存储凭证策略
```

```
type createStorageCredentialsPolicy struct {
    Statement []policyStatement `json:"statement"`
    Version   string            `json:"version"`
}

// policyStatement 策略语句
type policyStatement struct {
    Action   []string `json:"action"`
    Effect   string   `json:"effect"`
    Resource []string `json:"resource"`
}

// cred 临时凭证
type cred struct {
    AccessKeyId     string
    SecretAccessKey string
    SessionToken    string
}

// getCredential 获取临时凭证
func getCredential(context.Context) (*cred, error) {
    // 1. 获取临时凭证
    // 1.1 初始化调用腾讯云 API 对象
    credential := common.NewCredential(
        "SecretId",  // 腾讯云账号 SecretId
        "SecretKey", // 腾讯云账号 SecretKey
    )
    prof := profile.NewClientProfile()
    vodClient, err := vod20240718.NewClient(credential, "ap-guangzhou",
prof)
    if err != nil {
        log.Fatalf("create VOD client fail: %+v", err)
        return nil, fmt.Errorf("create VOD client fail: %w", err)
    }

    // 1.2 构造申请上传临时凭证请求
    policy := createStorageCredentialsPolicy{
        Statement: []policyStatement{{
            Action: []string{ // 当前仅支持如下 action，可以只申请其中一部分
                "name/vod:PutObject",
                "name/vod:ListParts",
                "name/vod:PostObject",
                "name/vod:InitiateMultipartUpload",
```

```
                "name/vod:UploadPart",
                "name/vod:CompleteMultipartUpload",
                "name/vod:AbortMultipartUpload",
                "name/vod:ListMultipartUploads",
            },
            Effect: "allow",
            Resource: []string{
                fmt.Sprintf("qcs::vod:%s:uid/%d:prefix//%d/%s/%s",
                    region,    // 就近地域上传
                    appId,     // 腾讯云账号 APPID
                    subAppId, // 云点播专业版应用 APPID
                    bucketId, // 就近地域的存储桶
                    fileKey,   // 上传到存储后的文件 KEY，包含完整路径和文件名，
例如 upload/demo.mp4
                ),
            },
        }},
        Version: "2.0",
    }
    req := vod20240718.NewCreateStorageCredentialsRequest()
    req.SubAppId = common.Uint64Ptr(subAppId)
    policyStr, _ := json.Marshal(policy)
    req.Policy = common.StringPtr(url.QueryEscape(string(policyStr)))

    // 1.3 申请上传临时凭证
    resp, err := vodClient.CreateStorageCredentials(req)
    if err != nil {
        log.Fatalf("create storage credentials fail: %+v", err)
        return nil, fmt.Errorf("create storage credentials fail: %w",
err)
    }
    log.Printf("create storage credentials success: %+v", resp)
    creds := resp.Response.Credentials
    return &cred{
        AccessKeyId:     *creds.AccessKeyId,
        SecretAccessKey: *creds.SecretAccessKey,
        SessionToken:    *creds.SessionToken,
    }, nil
}
```

Java

```
import com.tencentcloudapi.common.Credential;
import com.tencentcloudapi.common.exception.TencentCloudSDKException;
import com.tencentcloudapi.common.profile.ClientProfile;
import com.tencentcloudapi.vod.v20240718.VodClient;
import
com.tencentcloudapi.vod.v20240718.models.CreateStorageCredentialsRequest
;
import
com.tencentcloudapi.vod.v20240718.models.CreateStorageCredentialsRespons
e;

import org.json.JSONArray;
import org.json.JSONObject;

import java.net.URLEncoder;
import java.nio.charset.StandardCharsets;

/**
 * 凭证辅助类，负责获取临时存储凭证
 */
public class CredentialHelper {
    // 常量定义
    private static final long APP_ID = 251000000; // 腾讯云账号 APPID
    private static final long SUB_APP_ID = 1234567890; // 云点播专业版应用
APPID
    private static final String BUCKET_ID = "auto"; // 云点播专业版应用存储
桶的 ID，auto 表示自动就近选择存储桶
    private static final String FILE_KEY = "upload/demo.mp4"; // 申请权限
的存储文件 KEY，包含完整路径和文件名，例如 upload/demo.mp4
    private static final String REGION = "auto"; // 云点播专业版应用存储桶地
域，auto 表示自动就近选择地域

    /**
     * 凭证对象，存储临时凭证信息
     */
    public static class Cred {
        private final String accessKeyId;
        private final String secretAccessKey;
        private final String sessionToken;

        public Cred(String accessKeyId, String secretAccessKey, String
sessionToken) {
            this.accessKeyId = accessKeyId;
```

```
            this.secretAccessKey = secretAccessKey;
            this.sessionToken = sessionToken;
        }

        public String getAccessKeyId() {
            return accessKeyId;
        }

        public String getSecretAccessKey() {
            return secretAccessKey;
        }

        public String getSessionToken() {
            return sessionToken;
        }
    }

    /**
     * 获取临时凭证
     *
     * @return 临时凭证对象
     * @throws Exception 如果获取凭证失败
     */
    public static Cred getCredential() throws Exception {
        try {
            // 1. 初始化腾讯云 API 客户端
            Credential credential = new Credential("SecretId",
"SecretKey"); // 腾讯云账号 SecretId 和 SecretKey
            ClientProfile clientProfile = new ClientProfile(); // 客户端配
置
            VodClient vodClient = new VodClient(credential, "ap-
guangzhou", clientProfile); // 创建 VodClient 对象

            // 2. 构造并编码策略
            String policyJson = createPolicyJson();
            String encodedPolicy = URLEncoder.encode(policyJson,
StandardCharsets.UTF_8.name());

            // 3. 创建并发送请求
            CreateStorageCredentialsRequest req = new
CreateStorageCredentialsRequest();
            req.setSubAppId(SUB_APP_ID); // 云点播专业版应用 APPID
            req.setPolicy(encodedPolicy); // 策略
```

```
            // 4. 获取响应并返回凭证
            CreateStorageCredentialsResponse resp =
vodClient.CreateStorageCredentials(req);

            return new Cred(
                    resp.getCredentials().getAccessKeyId(),
                    resp.getCredentials().getSecretAccessKey(),
                    resp.getCredentials().getSessionToken());
        } catch (TencentCloudSDKException e) {
            System.err.println("获取存储凭证失败: " + e.getMessage());
            throw new Exception("获取存储凭证失败", e);
        }
    }

    /**
     * 创建策略JSON字符串，使用 org.json 库
     *
     * @return 策略JSON字符串
     */
    private static String createPolicyJson() {
        // 构建资源路径
        String resource = String.format(
                "qcs::vod:%s:uid/%d:prefix//%d/%s/%s",
                REGION,
                APP_ID,
                SUB_APP_ID,
                BUCKET_ID,
                FILE_KEY);

        // 构建操作列表
        String[] actions = {
                "name/vod:PutObject",
                "name/vod:ListParts",
                "name/vod:PostObject",
                "name/vod:CreateMultipartUpload",
                "name/vod:UploadPart",
                "name/vod:CompleteMultipartUpload",
                "name/vod:AbortMultipartUpload",
                "name/vod:ListMultipartUploads"
        };

        // 使用 JSONObject 构建 JSON
```

```
        JSONObject policy = new JSONObject();
        policy.put("version", "2.0");

        JSONArray statements = new JSONArray();
        JSONObject statement = new JSONObject();

        JSONArray actionArray = new JSONArray();
        for (String action : actions) {
            actionArray.put(action);
        }
        statement.put("action", actionArray);

        statement.put("effect", "allow");

        JSONArray resources = new JSONArray();
        resources.put(resource);
        statement.put("resource", resources);

        statements.put(statement);
        policy.put("statement", statements);

        return policy.toString();
    }
}
```

C++

```
#include <tencentcloud/core/TencentCloud.h>
#include <tencentcloud/core/profile/ClientProfile.h>
#include <tencentcloud/core/profile/HttpProfile.h>
#include <tencentcloud/core/Credential.h>
#include <tencentcloud/vod/v20240718/VodClient.h>
#include
<tencentcloud/vod/v20240718/model/CreateStorageCredentialsRequest.h>
#include
<tencentcloud/vod/v20240718/model/CreateStorageCredentialsResponse.h>
#include <string>
#include <sstream>
#include <iomanip>
#include <iostream>
#include <nlohmann/json.hpp>
```

```
using json = nlohmann::json;

// 常量定义
const uint64_t APP_ID = 251000000;                    // 腾讯云账号 APPID
const uint64_t SUB_APP_ID = 1234567890;               // 云点播专业版应用 APPID
const std::string BUCKET_ID = "auto";                 // 云点播专业版应用存储桶
的 ID，auto 表示自动就近选择存储桶
const std::string REGION = "auto";                    // 云点播专业版应用存储桶
地域，auto 表示自动就近选择地域
const std::string OBJECT_KEY = "upload/demo.mp4"; // 申请权限的存储文件
KEY，包含完整路径和文件名，例如 upload/demo.mp4

// 凭证结构体
struct Credential
{
    std::string accessKeyId;
    std::string secretAccessKey;
    std::string sessionToken;
};

// URL 编码函数
std::string UrlEncode(const std::string &value)
{
    std::ostringstream escaped;
    escaped.fill('0');
    escaped << std::hex;

    for (char c : value)
    {
        if (isalnum(c) || c == '-' || c == '_' || c == '.' || c == '~')
        {
            escaped << c;
        }
        else
        {
            escaped << std::uppercase;
            escaped << '%' << std::setw(2) << int((unsigned char)c);
            escaped << std::nouppercase;
        }
    }

    return escaped.str();
}
```

```
// 获取临时凭证
Credential GetCredential()
{
    // 初始化腾讯云 SDK
    TencentCloud::InitAPI();

    // 创建 VOD 客户端
    TencentCloud::Credential credential("SecretId", "SecretKey"); // 填入
腾讯云账号 SecretId 和 SecretKey
    TencentCloud::HttpProfile httpProfile;
    TencentCloud::ClientProfile clientProfile;
    clientProfile.SetHttpProfile(httpProfile);

    TencentCloud::Vod::V20240718::VodClient client(credential, "ap-
guangzhou", clientProfile);

    // 构建策略
    json policy_json = {
        {"statement", {{{"action", {"name/vod:PutObject",
"name/vod:ListParts", "name/vod:PostObject",
"name/vod:CreateMultipartUpload", "name/vod:UploadPart",
"name/vod:CompleteMultipartUpload", "name/vod:AbortMultipartUpload",
"name/vod:ListMultipartUploads"}}, {"effect", "allow"}, {"resource",
{"qcs::vod:" + REGION + ":uid/" + std::to_string(APP_ID) + ":prefix//" +
std::to_string(SUB_APP_ID) + "/" + BUCKET_ID + "/" + OBJECT_KEY}}}}},
        {"version", "2.0"}};
    std::string policy = policy_json.dump();

    // 创建请求对象
    TencentCloud::Vod::V20240718::Model::CreateStorageCredentialsRequest
req;
    req.SetSubAppId(SUB_APP_ID);
    req.SetPolicy(UrlEncode(policy));

    // 发送请求
    auto outcome = client.CreateStorageCredentials(req);
    if (!outcome.IsSuccess())
    {
        std::cerr << "Failed to get storage credentials: " <<
outcome.GetError().GetErrorMessage() << std::endl;
        TencentCloud::ShutdownAPI();
        exit(1);
```

```
    }

    // 提取凭证
    auto response = outcome.GetResult();
    auto creds = response.GetCredentials();

    Credential result;
    result.accessKeyId = creds.GetAccessKeyId();
    result.secretAccessKey = creds.GetSecretAccessKey();
    result.sessionToken = creds.GetSessionToken();

    // 清理腾讯云 SDK
    TencentCloud::ShutdownAPI();

    return result;
}
```

Python

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import json
import urllib.parse
from typing import NamedTuple

from tencentcloud.common import credential
from tencentcloud.common.profile import client_profile
from tencentcloud.vod.v20240718 import vod_client, models

# 常量定义
APP_ID = 251000000  # 腾讯云账号 APPID
SUB_APP_ID = 1234567890  # 云点播专业版应用 APPID
BUCKET_ID = "auto"  # 云点播专业版应用存储桶的 ID，auto 表示自动就近选择存储桶
OBJECT_KEY = "upload/demo.mp4"  # 上传到存储后的文件 KEY
REGION = "auto"  # 地域 auto 表示自动就近选择地域


class Credential(NamedTuple):
    """临时凭证"""

    access_key_id: str
```

```
    secret_access_key: str
    session_token: str


def get_credential() -> Credential:
    """获取临时凭证"""
    # 1. 初始化调用腾讯云 API 对象
    cred = credential.Credential(
        "SecretId",  # 腾讯云账号 SecretId
        "SecretKey",  # 腾讯云账号 SecretKey
    )
    prof = client_profile.ClientProfile()
    vod_cli = vod_client.VodClient(cred, "ap-guangzhou", prof)

    # 2. 构造申请上传临时凭证请求
    policy = {
        "statement": [
            {
                "action": [
                    "name/vod:PutObject",
                    "name/vod:ListParts",
                    "name/vod:PostObject",
                    "name/vod:CreateMultipartUpload",
                    "name/vod:UploadPart",
                    "name/vod:CompleteMultipartUpload",
                    "name/vod:AbortMultipartUpload",
                    "name/vod:ListMultipartUploads",
                ],
                "effect": "allow",
                "resource": [f"qcs::vod:
{REGION}:uid/{APP_ID}:prefix//{SUB_APP_ID}/{BUCKET_ID}/{OBJECT_KEY}"],
            }
        ],
        "version": "2.0",
    }

    req = models.CreateStorageCredentialsRequest()
    req.SubAppId = SUB_APP_ID
    req.Policy = urllib.parse.quote(json.dumps(policy))

    # 3. 申请上传临时凭证
    resp = vod_cli.CreateStorageCredentials(req)
    creds = resp.Credentials
```

```
    return Credential(
        access_key_id=creds.AccessKeyId,
        secret_access_key=creds.SecretAccessKey,
        session_token=creds.SessionToken,
    )
```

上传文件

常用语言初始化 S3 客户端并上传文件，代码实现如下所示：

GO

```go
// Package main
// 本示例基于 AWS SDK for Go v2 service/s3 v1.72.3 实现。
// AWS SDK for Go v2 service/s3 v1.73.0 及以后版本需禁用 S3 的默认完整性保护。
// 具体参考: https://github.com/aws/aws-sdk-go-v2/discussions/2960
package main

import (
    "context"
    "errors"
    "fmt"
    "log"
    "net/url"
    "os"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/credentials"
    "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
    "github.com/aws/aws-sdk-go-v2/service/s3"
    smep "github.com/aws/smithy-go/endpoints"
    "github.com/aws/smithy-go/logging"
)

type customEndpointResolverV2 struct {
}

// ResolveEndpoint 自定义接入点
func (r *customEndpointResolverV2) ResolveEndpoint(ctx context.Context,
    params s3.EndpointParameters) (smep.Endpoint, error) {
    if params.Bucket == nil || params.Region == nil {
        return smep.Endpoint{}, errors.New("invalid endpoint param")
    }
```

```
    return smep.Endpoint{
        URI: url.URL{
            Scheme: "https",
            Host: fmt.Sprintf(
                "%d.vodpro-upload.com",
                subAppId, // 子账号 Id
            ),
        },
    }, nil
}

func main() {
    // 1. 获取临时凭证
    cred, err := getCredential(context.Background())
    if err != nil {
        log.Fatalf("get credential fail: %v", err)
        return
    }

    // 2. 使用临时凭证上传文件
    // 2.1 创建 s3 client
    s3cli := s3.New(s3.Options{
        Credentials: credentials.NewStaticCredentialsProvider(
            cred.AccessKeyId,
            cred.SecretAccessKey,
            cred.SessionToken,
        ),
        EndpointResolverV2: new(customEndpointResolverV2),          // 自定
义接入点
        UsePathStyle:       true,                                   // 应用
级别域名需明确请求启用 path style 模式
        Logger:             logging.NewStandardLogger(os.Stdout), // 日志
打印到标准输出流
        ClientLogMode:      aws.LogRequest | aws.LogResponse,       // 打印
请求头和响应头
        Region:             "auto",                                 // 固定
填写 "auto"
    })

    // 2.2 打开本地文件
    file, err := os.Open("demo.mp4") // 本地文件路径
    if err != nil {
        log.Fatalf("failed to open file: %v", err)
```

```
        return
    }
    defer file.Close()

    // 2.3 上传文件到 S3
    // 2.3.1 创建上传器
    uploader := manager.NewUploader(s3cli, func(u *manager.Uploader) {
        u.PartSize = 10 * 1024 * 1024 // 设置分片大小为 10MB
        u.Concurrency = 5             // 设置并发上传的分片数
    })
    // 应用级别上传上传路径为 bucketId/fileKey
    // S3 SDK Uploader 不支持 path style 模式，此处需拼接后作为上传 key
    path, err := url.JoinPath(bucketId, fileKey)
    if err != nil {
        log.Fatalf("failed to join path: %v", err)
        return
    }
    // 2.3.2 上传文件
    result, err := uploader.Upload(context.TODO(), &s3.PutObjectInput{
        Bucket: aws.String(bucketId), // Bucket 设置为点播专业版应用中的存储
桶 ID
        // 文件在存储桶中的路径
        // 应用级别上传上传路径为 bucketId/fileKey
        // S3 SDK Uploader 不支持 path style 模式，此处需拼接后作为上传 key
        Key:  aws.String(path),
        Body: file,
    })

    if err != nil {
        log.Fatalf("failed to upload file: %v", err)
        return
    }
    log.Printf("etag: %s", *result.ETag) // 获取上传文件 etag
}
```

Java

```
// 本示例基于 AWS SDK for Java v2 service/s3 v2.31.35 实现。
// AWS SDK for Java 2.30.0 及以后版本需禁用 S3 的默认完整性保护。
// 具体参考: https://github.com/aws/aws-sdk-java-v2/issues/5801
import software.amazon.awssdk.auth.credentials.AwsBasicCredentials;
import software.amazon.awssdk.auth.credentials.AwsCredentials;
```

```
import software.amazon.awssdk.auth.credentials.AwsSessionCredentials;
import
software.amazon.awssdk.auth.credentials.StaticCredentialsProvider;
import software.amazon.awssdk.http.nio.netty.NettyNioAsyncHttpClient;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3AsyncClient;
import software.amazon.awssdk.services.s3.S3Configuration;
import software.amazon.awssdk.transfer.s3.S3TransferManager;
import software.amazon.awssdk.transfer.s3.model.UploadFileRequest;
import
software.amazon.awssdk.transfer.s3.progress.LoggingTransferListener;
import software.amazon.awssdk.core.checksums.RequestChecksumCalculation;
import software.amazon.awssdk.core.checksums.ResponseChecksumValidation;

import java.io.File;
import java.net.URI;
import java.util.concurrent.CompletableFuture;

public class Main {
    public static void main(String[] args) {
        S3AsyncClient s3AsyncClient = null;
        S3TransferManager transferManager = null;

        try {
            // 1. 获取临时凭证
            CredentialHelper.Cred cred =
CredentialHelper.getCredential();
            System.out.println("获取临时凭证成功, AccessKeyId: " +
cred.getAccessKeyId());

            // 2. 创建会话凭证（包含临时 token）
            AwsSessionCredentials sessionCredentials =
AwsSessionCredentials.create(
                    cred.getAccessKeyId(), // 临时凭证 AccessKeyId
                    cred.getSecretAccessKey(), // 临时凭证 SecretAccessKey
                    cred.getSessionToken() // 临时凭证 SessionToken
            );

            long subAppId = 1234567890; // 云点播专业版应用 APPID
            String bucketId = "auto"; // 云点播专业版应用存储桶的 ID，auto 表
示自动就近选择存储桶
            String filePath = "demo.mp4"; // 本地文件路径
```

```
            String key = "upload/demo.mp4"; // 申请权限的存储文件 KEY，包含完
整路径和文件名，例如 upload/demo.mp4
            String endpointUrl = String.format("https://%d.vodpro-
upload.com", subAppId); // 应用级别的预置域名端点

            // 3. 使用自定义端点构建S3异步客户端
            s3AsyncClient = S3AsyncClient.builder()

.credentialsProvider(StaticCredentialsProvider.create(sessionCredentials
)) // 设置凭证提供者
                    .endpointOverride(URI.create(endpointUrl)) // 设置端点
                    .region(Region.of("auto")) // 固定填写 auto

.httpClient(NettyNioAsyncHttpClient.builder().build()) // 设置HTTP客户端
                    .serviceConfiguration(S3Configuration.builder()
                            .pathStyleAccessEnabled(true) // 设置路径样式访
问，应用级别需明确使用 PathStyle 模式
                            .build())

.requestChecksumCalculation(RequestChecksumCalculation.WHEN_REQUIRED)

.responseChecksumValidation(ResponseChecksumValidation.WHEN_REQUIRED)
                    .build();

            // 4. 使用S3异步客户端创建S3TransferManager
            transferManager = S3TransferManager.builder()
                    .s3Client(s3AsyncClient)
                    .build();

            // 5. 准备上传文件
            File fileToUpload = new File(filePath);
            if (!fileToUpload.exists()) {
                throw new RuntimeException("文件不存在: " + filePath);
            }

            // 6. 创建上传请求并添加传输监听器用于进度跟踪
            UploadFileRequest uploadRequest =
UploadFileRequest.builder()
                    .putObjectRequest(builder -> builder
                            .bucket(bucketId)
                            .key(key))

.addTransferListener(LoggingTransferListener.create())
```

```
                    .source(fileToUpload)
                    .build();

            // 7. 开始上传
            System.out.println("开始上传文件...");
            CompletableFuture<Void> future =
transferManager.uploadFile(uploadRequest)
                    .completionFuture()
                    .thenAccept(response -> {
                        System.out.println("上传成功! ETag: " +
response.response().eTag());
                    });

            // 8. 等待上传完成
            future.join();

            System.out.println("上传任务完成");
        } catch (Exception e) {
            System.err.println("上传失败: " + e.getMessage());
            e.printStackTrace();
        } finally {
            // 资源释放
            try {
                if (transferManager != null) {
                    transferManager.close();
                }
                if (s3AsyncClient != null) {
                    s3AsyncClient.close();
                }
                System.out.println("资源已释放");
            } catch (Exception e) {
                System.err.println("关闭资源时发生错误: " +
e.getMessage());
            }
        }
    }
}
```

C++

```
// 本示例基于 AWS SDK for C++ v1.11.560 实现。
// AWS SDK for C++ v1.11.486 及以后版本需禁用 S3 的默认完整性保护。
```

```
// 具体参考: https://github.com/aws/aws-sdk-cpp/issues/3253
#include <aws/core/Aws.h>
#include <aws/core/auth/AWSCredentials.h>
#include <aws/core/client/ClientConfiguration.h>
#include <aws/s3/S3Client.h>
#include <aws/transfer/TransferManager.h>
#include <iostream>
#include <string>

// 引用 get_cred.cpp 中定义的凭证结构体
struct Credential
{
    std::string accessKeyId;
    std::string secretAccessKey;
    std::string sessionToken;
};

// 引用 get_cred.cpp 中定义的函数
extern Credential GetCredential();

int main()
{
    // AWS SDK 初始化
    Aws::SDKOptions options;
    options.loggingOptions.logLevel =
Aws::Utils::Logging::LogLevel::Info;
    Aws::InitAPI(options);

    {
        // 获取临时凭证
        std::cout << "Getting temporary credentials..." << std::endl;
        Credential cred = GetCredential();
        std::cout << "Successfully obtained temporary credentials,
access key Id: " << cred.accessKeyId << std::endl;
        // 定义常量
        const uint64_t subAppId = 1234567890;           // 云点播专业版应用
SubAppId
        const Aws::String bucketId = "auto";            // 云点播专业版应用
存储桶的 ID，auto 表示自动就近选择存储桶
        const Aws::String keyName = "upload/demo.mp4"; // 文件在存储桶中的
路径，包含完整路径和文件名，例如 upload/demo.mp4
        const Aws::String filePath = "demo.mp4";        // 本地文件路径
```

```
        // 配置客户端
        Aws::Client::ClientConfiguration clientConfig;
        clientConfig.region = "auto";
// 固定填写 auto
        clientConfig.scheme = Aws::Http::Scheme::HTTPS;
// 协议使用 HTTPS
        clientConfig.enableEndpointDiscovery = false;
// 禁用 endpoint discovery
        clientConfig.verifySSL = true;
// 启用 SSL 证书验证
        clientConfig.endpointOverride = std::to_string(subAppId) +
".vodpro-upload.com"; // 应用级别预置域名
        // 禁用完整性校验
        clientConfig.checksumConfig.requestChecksumCalculation =
Aws::Client::RequestChecksumCalculation::WHEN_REQUIRED;
        clientConfig.checksumConfig.responseChecksumValidation =
Aws::Client::ResponseChecksumValidation::WHEN_REQUIRED;

        // 设置认证信息（使用临时凭证）
        Aws::Auth::AWSCredentials credentials(
            cred.accessKeyId.c_str(),
            cred.secretAccessKey.c_str(),
            cred.sessionToken.c_str());

        // 创建 S3 客户端
        auto s3Client = Aws::MakeShared<Aws::S3::S3Client>(
            "S3Client",
            credentials,
            clientConfig,
            Aws::Client::AWSAuthV4Signer::PayloadSigningPolicy::Never,
// 禁用 payload signing
            false
// 应用级别预置域名需明确使用 PathStyle 格式
        );

        // 创建线程池执行器
        auto executor =
Aws::MakeShared<Aws::Utils::Threading::PooledThreadExecutor>("executor",
10);

        // 配置 TransferManager
        Aws::Transfer::TransferManagerConfiguration
transferConfig(executor.get());
```

```
        transferConfig.s3Client = s3Client;

        // 设置分片大小为 10MB
        transferConfig.bufferSize = 10 * 1024 * 1024;

        // 创建 TransferManager
        auto transferManager =
Aws::Transfer::TransferManager::Create(transferConfig);

        std::cout << "Starting file upload: " << filePath << " to " <<
bucketId << "/" << keyName << std::endl;

        // 执行上传
        auto uploadHandle = transferManager->UploadFile(
            filePath,                                   // 本地文件路径
            bucketId,                                   // 存储桶ID
            keyName,                                    // 对象键（存储路径）
            "application/octet-stream",                 // 内容类型
            Aws::Map<Aws::String, Aws::String>() // 元数据
        );

        // 等待上传完成
        uploadHandle->WaitUntilFinished();

        // 检查上传状态
        if (uploadHandle->GetStatus() ==
Aws::Transfer::TransferStatus::COMPLETED)
        {
            std::cout << "File upload completed successfully!" <<
std::endl;
        }
        else
        {
            auto lastError = uploadHandle->GetLastError();
            std::cerr << "File upload failed: " <<
lastError.GetMessage() << std::endl;
        }
    }

    // 清理SDK资源
    Aws::ShutdownAPI(options);
    return 0;
}
```

Python

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
本示例基于 AWS SDK for Python v1.38.7 实现。
Boto3 的 AWS SDK for Python v1.36.0 及以后版本需禁用 S3 的默认完整性保护。
具体参考：https://github.com/boto/boto3/issues/4392
"""

import boto3
from botocore.config import Config
from botocore.exceptions import ClientError

from get_cred import get_credential, SUB_APP_ID, BUCKET_ID, OBJECT_KEY

# 常量定义
FILE_PATH = "demo.mp4"


try:
    # 1. 获取临时凭证
    cred = get_credential()

    # 2. 创建 S3 客户端
    s3_client = boto3.client(
        "s3",
        aws_access_key_id=cred.access_key_id,  # 临时凭证的 AccessKeyId
        aws_secret_access_key=cred.secret_access_key,  # 临时凭证的
SecretAccessKey
        aws_session_token=cred.session_token,  # 临时凭证的 SessionToken
        endpoint_url=f"https://{SUB_APP_ID}.vodpro-upload.com",  # 应用级
别预置上传域名
        region_name="auto",  # 固定填写 auto
        config=Config(
            s3={"addressing_style": "path"},  # 使用 path-style
            request_checksum_calculation="when_required",  # Boto3 的 AWS
SDK for Python v1.36.0 及以后版本需禁用 S3 的默认完整性保护。
            response_checksum_validation="when_required",  # Boto3 的 AWS
SDK for Python v1.36.0 及以后版本需禁用 S3 的默认完整性保护。
        ),
    )
```

```
    # 3. 上传文件
    response = s3_client.upload_file(
        Bucket=BUCKET_ID,  # 填入点播专业版应用中的存储桶 ID
        Key=OBJECT_KEY,  # 文件在存储桶中的路径
        Filename=FILE_PATH,  # 本地文件路径
    )
    print(response)
except ClientError as e:
    print(f"Error: {e}")
```

⚠ 注意：

- 使用 AWS SDK 请确认数据完整性保护特性，详情请参考 AWS SDK 的相关文档 Data Integrity Protections for Amazon S3。
- 使用应用级别上传加速域名，必须使用 PathStyle 格式。

客户端上传
客户端上传指引

最近更新时间：2025-06-13 16:30:52

本文主要提供专业版应用客户端上传指引。

适用场景

终端用户将客户端本地视频上传到云点播，适用于 UGC、PGC 等场景。

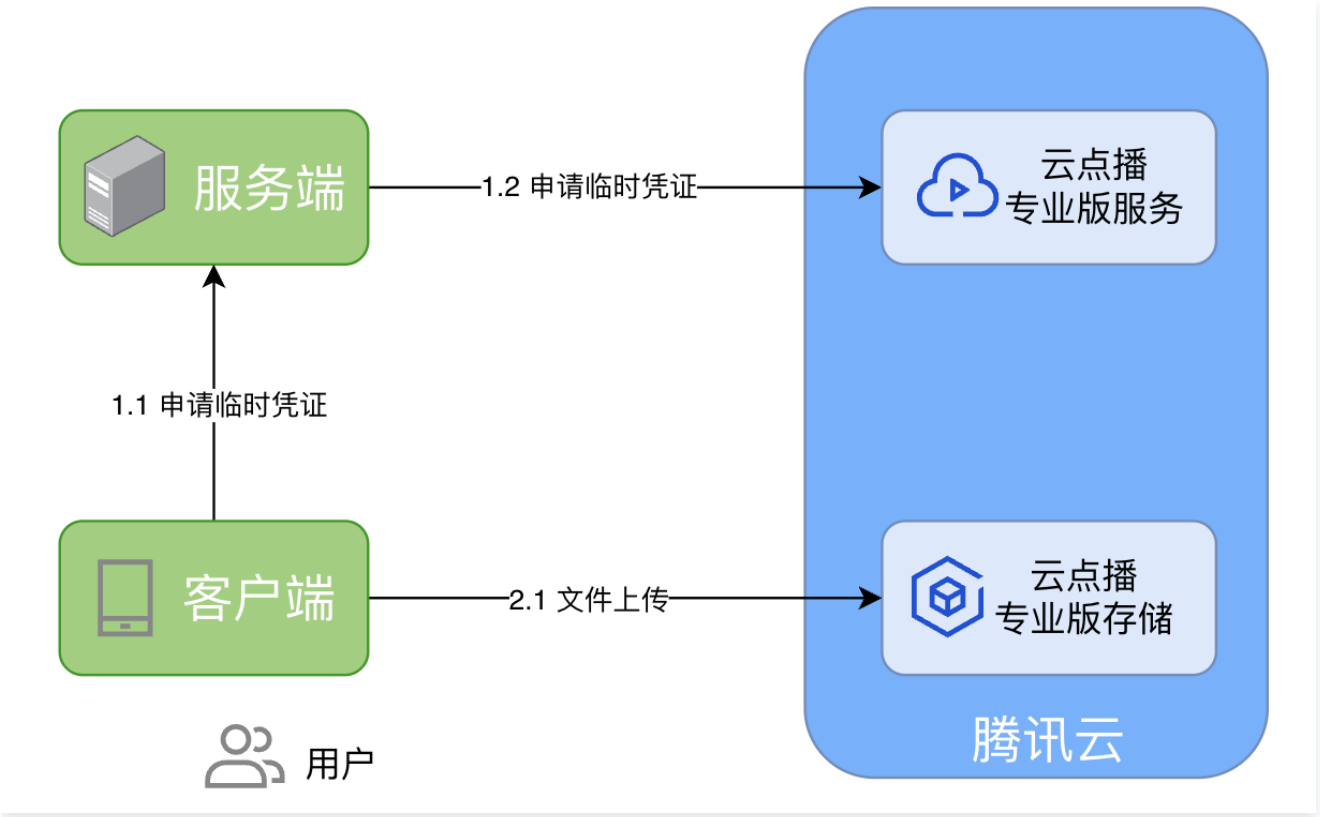
上传方式

客户端对密钥安全性要求较高，我们推荐通过专业版预置域名使用临时凭证上传，支持方式如下：

上传方式	上传域名	上传凭证	支持操作
存储桶预置域名	`[BucketId].vodpro.[存储地域].eovod.com`	临时凭证	上传至指定存储桶
应用预置域名	`[SubAppId].vodpro-upload.com`	临时凭证	• 上传至指定存储桶 • 就近上传至应用内某区域的存储桶

使用临时凭证上传文件

临时凭证通常用于对密钥安全性要求较高的场景。客户端使用临时凭证上传文件至专业版存储整体流程如下图：



1. 申请上传临时凭证

   1.1 客户端向 App 后台申请上传临时凭证。

   1.2 服务端调用云点播专业版服务 申请上传临时凭证 。

2. 上传文件

   客户端使用临时凭证上传文件至云点播专业版存储。

准备工作

1. 创建专业版应用和存储桶

   在 云点播控制台 创建好专业版应用和存储桶，具体操作步骤参见 快速入门 和 创建存储桶 文档。

2. 获取腾讯云账号永久密钥对

   2.1 登录腾讯云控制台，选择访问管理 > 访问密钥 > API 密钥管理，进入“API 密钥管理” 页面。

   2.2 获取云 API 密钥。如果您尚未创建密钥，则单击新建密钥即可创建一对 `SecretId` 和 `SecretKey` 。

服务端申请上传临时凭证

客户服务端使用 `SecretId` 和 `SecretKey` 调用 申请上传临时凭证 接口，获取到上传临时凭证后派发给客户端使用。
假设专业版应用 ID 为 `1234567890` ，存储桶的存储地域为 `ap-guangzhou` ，存储桶 ID 为 `bucketid1` 。
服务端申请 `bucketid1` 存储桶上传 `upload/demo.mp4` 的临时凭证，实现如下：

GO

```
// Package main
package main

import (
    "context"
    "encoding/json"
    "fmt"
    "log"
    "net/url"

    "github.com/tencentcloud/tencentcloud-sdk-go/tencentcloud/common"
    "github.com/tencentcloud/tencentcloud-sdk-
go/tencentcloud/common/profile"
    vod20240718 "github.com/tencentcloud/tencentcloud-sdk-
go/tencentcloud/vod/v20240718"
)

const (
    appId    = 251000000          // 腾讯云账号 APPID
    subAppId = 1234567890         // 云点播专业版应用 APPID
    bucketId = "bucketid1"        // 云点播专业版应用存储桶的 ID
    fileKey  = "upload/demo.mp4" // 上传到存储后的文件 KEY，包含完整路径和文件
名，例如 upload/demo.mp4
)

// createStorageCredentialsPolicy 创建存储凭证策略
type createStorageCredentialsPolicy struct {
    Statement []policyStatement `json:"statement"`
    Version   string            `json:"version"`
}

// policyStatement 策略语句
type policyStatement struct {
    Action   []string `json:"action"`
    Effect   string   `json:"effect"`
    Resource []string `json:"resource"`
}

// cred 临时凭证
type cred struct {
    AccessKeyId     string
    SecretAccessKey string
    SessionToken    string
```

```
}

// getCredential 获取临时凭证
func getCredential(context.Context) (*cred, error) {
    // 1. 获取临时凭证
    // 1.1 初始化调用腾讯云 API 对象
    credential := common.NewCredential(
        "SecretId",  // 腾讯云账号 SecretId
        "SecretKey", // 腾讯云账号 SecretKey
    )
    prof := profile.NewClientProfile()
    vodClient, err := vod20240718.NewClient(credential, "ap-guangzhou",
prof)
    if err != nil {
        log.Fatalf("create VOD client fail: %+v", err)
        return nil, fmt.Errorf("create VOD client fail: %w", err)
    }

    // 1.2 构造申请上传临时凭证请求
    policy := createStorageCredentialsPolicy{
        Statement: []policyStatement{{
            Action: []string{ // 当前仅支持如下 action，可以只申请其中一部分
                "name/vod:PutObject",
                "name/vod:ListParts",
                "name/vod:PostObject",
                "name/vod:CreateMultipartUpload",
                "name/vod:UploadPart",
                "name/vod:CompleteMultipartUpload",
                "name/vod:AbortMultipartUpload",
                "name/vod:ListMultipartUploads",
            },
            Effect: "allow",
            Resource: []string{
                fmt.Sprintf("qcs::vod:%s:uid/%d:prefix//%d/%s/%s",
                    "ap-guangzhou", // 存储桶所在地域
                    appId,          // 腾讯云账号 APPID
                    subAppId,       // 云点播专业版应用 APPID
                    bucketId,       // 云点播专业版应用存储桶的 ID
                    fileKey,        // 上传到存储后的文件 KEY，包含完整路径和
文件名，例如 upload/demo.mp4
                ),
            },
        }},
```

```
        Version: "2.0",
    }
    req := vod20240718.NewCreateStorageCredentialsRequest()
    req.SubAppId = common.Uint64Ptr(subAppId)
    policyStr, _ := json.Marshal(policy)
    req.Policy = common.StringPtr(url.QueryEscape(string(policyStr)))

    // 1.3 申请上传临时凭证
    resp, err := vodClient.CreateStorageCredentials(req)
    if err != nil {
        log.Fatalf("create storage credentials fail: %+v", err)
        return nil, fmt.Errorf("create storage credentials fail: %w",
err)
    }
    log.Printf("create storage credentials success: %+v", resp)
    creds := resp.Response.Credentials
    return &cred{
        AccessKeyId:     *creds.AccessKeyId,
        SecretAccessKey: *creds.SecretAccessKey,
        SessionToken:    *creds.SessionToken,
    }, nil
}
```

Java

```
import com.tencentcloudapi.common.Credential;
import com.tencentcloudapi.common.exception.TencentCloudSDKException;
import com.tencentcloudapi.common.profile.ClientProfile;
import com.tencentcloudapi.vod.v20240718.VodClient;
import
com.tencentcloudapi.vod.v20240718.models.CreateStorageCredentialsRequest
;
import
com.tencentcloudapi.vod.v20240718.models.CreateStorageCredentialsRespons
e;

import org.json.JSONArray;
import org.json.JSONObject;

import java.net.URLEncoder;
import java.nio.charset.StandardCharsets;
```

```
/**
 * 凭证辅助类，负责获取临时存储凭证
 */
public class CredentialHelper {
    // 常量定义
    private static final long APP_ID = 251000000; // 腾讯云账号 APPID
    private static final long SUB_APP_ID = 1234567890; // 云点播专业版应用
APPID
    private static final String BUCKET_ID = "bucketid1"; // 云点播专业版应
用存储桶的 ID
    private static final String FILE_KEY = "upload/demo.mp4"; // 上传到存
储后的文件 KEY
    private static final String REGION = "ap-guangzhou"; // 存储桶所在地域

    /**
     * 凭证对象，存储临时凭证信息
     */
    public static class Cred {
        private final String accessKeyId;
        private final String secretAccessKey;
        private final String sessionToken;

        public Cred(String accessKeyId, String secretAccessKey, String
sessionToken) {
            this.accessKeyId = accessKeyId;
            this.secretAccessKey = secretAccessKey;
            this.sessionToken = sessionToken;
        }

        public String getAccessKeyId() {
            return accessKeyId;
        }

        public String getSecretAccessKey() {
            return secretAccessKey;
        }

        public String getSessionToken() {
            return sessionToken;
        }
    }

    /**
```

```
     * 获取临时凭证
     *
     * @return 临时凭证对象
     * @throws Exception 如果获取凭证失败
     */
    public static Cred getCredential() throws Exception {
        try {
            // 1. 初始化腾讯云 API 客户端
            Credential credential = new Credential("SecretId",
"SecretKey"); // 腾讯云账号 SecretId 和 SecretKey
            ClientProfile clientProfile = new ClientProfile(); // 客户端配
置
            VodClient vodClient = new VodClient(credential, "ap-
guangzhou", clientProfile); // 创建 VodClient 对象

            // 2. 构造并编码策略
            String policyJson = createPolicyJson();
            String encodedPolicy = URLEncoder.encode(policyJson,
StandardCharsets.UTF_8.name());

            // 3. 创建并发送请求
            CreateStorageCredentialsRequest req = new
CreateStorageCredentialsRequest();
            req.setSubAppId(SUB_APP_ID); // 云点播专业版应用 APPID
            req.setPolicy(encodedPolicy); // 策略

            // 4. 获取响应并返回凭证
            CreateStorageCredentialsResponse resp =
vodClient.CreateStorageCredentials(req);

            return new Cred(
                    resp.getCredentials().getAccessKeyId(),
                    resp.getCredentials().getSecretAccessKey(),
                    resp.getCredentials().getSessionToken());
        } catch (TencentCloudSDKException e) {
            System.err.println("获取存储凭证失败: " + e.getMessage());
            throw new Exception("获取存储凭证失败", e);
        }
    }

    /**
     * 创建策略JSON字符串，使用 org.json 库
     *
```

```
     * @return 策略JSON字符串
     */
    private static String createPolicyJson() {
        // 构建资源路径
        String resource = String.format(
                "qcs::vod:%s:uid/%d:prefix//%d/%s/%s",
                REGION,
                APP_ID,
                SUB_APP_ID,
                BUCKET_ID,
                FILE_KEY);

        // 构建操作列表
        String[] actions = {
                "name/vod:PutObject",
                "name/vod:ListParts",
                "name/vod:PostObject",
                "name/vod:CreateMultipartUpload",
                "name/vod:UploadPart",
                "name/vod:CompleteMultipartUpload",
                "name/vod:AbortMultipartUpload",
                "name/vod:ListMultipartUploads"
        };

        // 使用 JSONObject 构建 JSON
        JSONObject policy = new JSONObject();
        policy.put("version", "2.0");

        JSONArray statements = new JSONArray();
        JSONObject statement = new JSONObject();

        JSONArray actionArray = new JSONArray();
        for (String action : actions) {
            actionArray.put(action);
        }
        statement.put("action", actionArray);

        statement.put("effect", "allow");

        JSONArray resources = new JSONArray();
        resources.put(resource);
        statement.put("resource", resources);
```

```
        statements.put(statement);
        policy.put("statement", statements);

        return policy.toString();
    }
}
```

C++

```
#include <tencentcloud/core/TencentCloud.h>
#include <tencentcloud/core/profile/ClientProfile.h>
#include <tencentcloud/core/profile/HttpProfile.h>
#include <tencentcloud/core/Credential.h>
#include <tencentcloud/vod/v20240718/VodClient.h>
#include
<tencentcloud/vod/v20240718/model/CreateStorageCredentialsRequest.h>
#include
<tencentcloud/vod/v20240718/model/CreateStorageCredentialsResponse.h>
#include <string>
#include <sstream>
#include <iomanip>
#include <iostream>
#include <nlohmann/json.hpp>

using json = nlohmann::json;

const uint64_t APP_ID = 251000000;                // 腾讯云账号 APPID
const uint64_t SUB_APP_ID = 1234567890;           // 云点播专业版应用 APPID
const std::string BUCKET_ID = "bucketid1";        // 云点播专业版应用存储桶
的 ID
const std::string REGION = "ap-guangzhou";        // 云点播专业版应用存储桶
地域
const std::string OBJECT_KEY = "upload/demo.mp4"; // 申请权限的存储文件 KEY

// 凭证结构体
struct Credential
{
    std::string accessKeyId;
    std::string secretAccessKey;
    std::string sessionToken;
};
```

```
// URL 编码函数
std::string UrlEncode(const std::string &value)
{
    std::ostringstream escaped;
    escaped.fill('0');
    escaped << std::hex;

    for (char c : value)
    {
        if (isalnum(c) || c == '-' || c == '_' || c == '.' || c == '~')
        {
            escaped << c;
        }
        else
        {
            escaped << std::uppercase;
            escaped << '%' << std::setw(2) << int((unsigned char)c);
            escaped << std::nouppercase;
        }
    }

    return escaped.str();
}

// 获取临时凭证
Credential GetCredential()
{
    // 初始化腾讯云 SDK
    TencentCloud::InitAPI();

    // 创建 VOD 客户端
    TencentCloud::Credential credential("SecretId", "SecretKey"); // 填入
腾讯云账号 SecretId 和 SecretKey
    TencentCloud::HttpProfile httpProfile;
    TencentCloud::ClientProfile clientProfile;
    clientProfile.SetHttpProfile(httpProfile);

    TencentCloud::Vod::V20240718::VodClient client(credential, "ap-
guangzhou", clientProfile);

    // 构建策略
    json policy_json = {
```

```
        {"statement", {{{"action", {"name/vod:PutObject",
"name/vod:ListParts", "name/vod:PostObject",
"name/vod:CreateMultipartUpload", "name/vod:UploadPart",
"name/vod:CompleteMultipartUpload", "name/vod:AbortMultipartUpload",
"name/vod:ListMultipartUploads"}}, {"effect", "allow"}, {"resource",
{"qcs::vod:" + REGION + ":uid/" + std::to_string(APP_ID) + ":prefix//" +
std::to_string(SUB_APP_ID) + "/" + BUCKET_ID + "/" + OBJECT_KEY}}}}},
        {"version", "2.0"}};
    std::string policy = policy_json.dump();

    // 创建请求对象
    TencentCloud::Vod::V20240718::Model::CreateStorageCredentialsRequest
req;
    req.SetSubAppId(SUB_APP_ID);
    req.SetPolicy(UrlEncode(policy));

    // 发送请求
    auto outcome = client.CreateStorageCredentials(req);
    if (!outcome.IsSuccess())
    {
        std::cerr << "Failed to get storage credentials: " <<
outcome.GetError().GetErrorMessage() << std::endl;
        TencentCloud::ShutdownAPI();
        exit(1);
    }

    // 提取凭证
    auto response = outcome.GetResult();
    auto creds = response.GetCredentials();

    Credential result;
    result.accessKeyId = creds.GetAccessKeyId();
    result.secretAccessKey = creds.GetSecretAccessKey();
    result.sessionToken = creds.GetSessionToken();

    // 清理腾讯云 SDK
    TencentCloud::ShutdownAPI();

    return result;
}
```

Python

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import json
import urllib.parse
from typing import NamedTuple

from tencentcloud.common import credential
from tencentcloud.common.profile import client_profile
from tencentcloud.vod.v20240718 import vod_client, models

# 常量定义
APP_ID = 251000000  # 腾讯云账号 APPID
SUB_APP_ID = 1234567890  # 云点播专业版应用 APPID
BUCKET_ID = "bucketid1"  # 云点播专业版应用存储桶的 ID
OBJECT_KEY = "upload/demo.mp4"  # 上传到存储后的文件 KEY
REGION = "ap-guangzhou"  # 地域


class Credential(NamedTuple):
    """临时凭证"""

    access_key_id: str
    secret_access_key: str
    session_token: str


def get_credential() -> Credential:
    """获取临时凭证"""
    # 1. 初始化调用腾讯云 API 对象
    cred = credential.Credential(
        "SecretId",  # 腾讯云账号 SecretId
        "SecretKey",  # 腾讯云账号 SecretKey
    )
    prof = client_profile.ClientProfile()
    vod_cli = vod_client.VodClient(cred, "ap-guangzhou", prof)

    # 2. 构造申请上传临时凭证请求
    policy = {
        "statement": [
            {
                "action": [
                    "name/vod:PutObject",
```

```
                    "name/vod:ListParts",
                    "name/vod:PostObject",
                    "name/vod:CreateMultipartUpload",
                    "name/vod:UploadPart",
                    "name/vod:CompleteMultipartUpload",
                    "name/vod:AbortMultipartUpload",
                    "name/vod:ListMultipartUploads",
                ],
                "effect": "allow",
                "resource": [f"qcs::vod:
{REGION}:uid/{APP_ID}:prefix//{SUB_APP_ID}/{BUCKET_ID}/{OBJECT_KEY}"],
            }
        ],
        "version": "2.0",
    }

    req = models.CreateStorageCredentialsRequest()
    req.SubAppId = SUB_APP_ID
    req.Policy = urllib.parse.quote(json.dumps(policy))

    # 3. 申请上传临时凭证
    resp = vod_cli.CreateStorageCredentials(req)
    creds = resp.Credentials
    return Credential(
        access_key_id=creds.AccessKeyId,
        secret_access_key=creds.SecretAccessKey,
        session_token=creds.SessionToken,
    )
```

使用 SDK 上传文件

下面介绍在常用的客户端 SDK 中，如何进行适配以上传文件至专业版应用的存储桶。

域名

专业版为每个应用预置了一个上传加速域名，应用下所有存储桶均可使用该域名进行上传。
示例使用的应用上传加速域名均为：`1234567890.vodpro-upload.com`。

使用 VOD SDK 上传文件

App 客户端使用获取到的临时凭证，将文件上传至云点播专业版存储，详情参见：Flutter 上传 SDK。

Flutter 上传 SDK

最近更新时间：2025-06-13 16:30:53

本示例基于 VOD 专业版 Flutter SDK v1.0.2，可 单击下载 上传 Demo 及源码。

环境准备

- Flutter SDK 版本：⩾ 3.3.0。
- **在 iOS 和 Android 端上传媒体文件时，如果业务需要从相册或存储中获取文件，则需自行配置并申请相册访问权限和存储访问权限等。**

Demo 中以第三方插件访问为示例，配置了所需的权限，客户可根据业务场景自行调整。

```
<uses-permission android:name="android.permission.INTERNET"/>
```

依赖组件

将 `voduploadadv` 项目，复制到本地，使用相对目录进行依赖。
这里以复制到项目的平级目录为例，在项目 pubspec.yaml 根据 `voduploadadv` 的相对路径进行依赖。

```
  voduploadadv:
    path: ../voduploadadv/
```

添加完成后，执行如下命令刷新 pub 依赖。

```
flutter clean
flutter pub get
```

配置鉴权

上传过程中，需要对您客户端的身份进行鉴权，此时需要您配置鉴权回调，在 SDK 上传过程中需要拉取鉴权的时候，会调用您配置的回调，来获取 `secretId`，`secretKey`，`token`，`expiredTime` 等信息。

> **注意：**
> 该配置为必须调用项，如果不配置，会导致无法上传。

使用示例如下：

```
TxVodUploadPlugin.instance.initWithScopeLimitCredential((stsCredentialSc
opes) async {
```

```
  // 建议调用服务器端获取鉴权信息
  Map<String, dynamic> jsonMap = jsonDecode(testRsCre);
  Map<String, dynamic> resJsonMap = jsonMap["Response"];
  Map<String, dynamic> credentialsJsonMap = resJsonMap["Credentials"];
  int expiredTimeStamp =
DateTime.parse(credentialsJsonMap['Expiration']).millisecondsSinceEpoch;
  return FTXSessionQCloudCredentials(
      secretId: credentialsJsonMap["AccessKeyId"],
      secretKey: credentialsJsonMap["SecretAccessKey"],
      token: credentialsJsonMap["SessionToken"],
      expiredTime: expiredTimeStamp);
});
```

在获取鉴权的时候，强烈建议您从服务器获取临时的鉴权凭证，防止鉴权凭证泄露。

其中 `stsCredentialScopes` 包含 `action`，`region`，`bucket`，`prefix` 几个字段。

- region 默认为空字符串。
- bucket 为您传入的 bucketId，auto 模式下则为 auto。
- prefix 前缀目录，默认为空字符串。
- action 则可能会包含如下字段：

```
//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"
```

如果您针对不同的上传应用有不同的鉴权凭证，可以根据以上信息进行区分。

创建上传对象

创建 `TXUploader` 的时候，有两种模式，分别是 `subApp` 和 `bucket`，分别对应自动模式和指定桶模式。

两种模式分别对应构造方法为 `TXUploader.bucketMode` 和 `TXUploader.autoMode` 。
此外还需要传入您的子应用 id，示例代码如下：

```
_uploader = TXUploader.bucketMode(0000000000, "xxxxxxxxxxxx");
```

设置上传回调

创建完成之后，如果需要对上传状态进行监控，需要设置回调。**回调均为可选实现，业务可根据自身需求进行实现。**
设置回调示例代码如下：

```
 _uploader?.setUploadCallback = FTXUploadCallback(
  successCallBack: (header, result) async {

   // 上传成功回调，此处可从 result 中获取上传完成后的文件 url，
result.accessUrl

    }, failCallBack: (clientException, serviceException) async {

    // 上传失败回调，可在此处处理错误信息

    }, progressCallBack: (complete, target) {

    // 上传进度回调，complete 为已上传的字节数量，target为总数量

    },stateCallBack:(state) {

    // 上传状态变更回调

    }
    , startUploadCallback: (bucket, cosKey, uploadId) async {

    // 上传开始回调，此处可以获得 uploadId，可以进行存储，后续如果文件上传到一半，可
以传入 uploadId 进行续传

    });
```

开始上传

根据业务需要，**可选是否设置上传配置。**

```
UploadConfig uploadConfig = UploadConfig();
```

```
uploadConfig.sliceSizeForUpload = 1024 * 1024 * 2; // 默认设置分片大小2M，可
根据项目需要来调整
uploadConfig.isHttps = true; // 是否开启 https，默认开启
uploadConfig.enableVerification = true; // 是否开启分片校验，默认开启
```

配置上传参数

上传参数 UploadParams 参数如下

参数	含义
localFilePath	本地文件路径，必传。
fileKey	上传文件键值，为上传到服务器的文件名，必传。
uploadId	上传 uploadId，可不传，传入则会续传之前该 uploadId 没有上传的部分。
uploadConfig	上传配置，可不传。

配置完成后，可以开始上传，示例如下：

```
int? code = await _uploader?.upload(params);
if (code == TXUploadCode.TX_UPLOAD_OK) {
  showResult("正在上传");
} else {
  showResult("上传 code$code");
}
```

上传 code 的返回值：

- `TXUploadCode.TX_UPLOAD_OK` **代表调用成功。**
- `TXUploadCode.TX_UPLOAD_BUSY` **代表该** `TXUploader` **已经在处理上传，拒绝再次调用。**
- `TX_FILE_NOT_FOUND` **代表传入的文件路径不存在。**

内网上传

最近更新时间：2025-08-04 21:19:11

本文主要提供专业版应用内网上传指引。

适用场景

开发者可通过腾讯云资源内网上传文件至同地域的专业版存储。
云点播专业版支持通过腾讯云内网访问存储文件。客户可使用腾讯云资源内网访问相同地域的云点播专业版存储；内网访问时，上传和下载文件均不会产生流量费用。

上传方式

内网上传支持方式如下：

上传方式	上传域名	上传凭证	支持操作
存储桶内网预置域名	`[BucketId].vodsrc-internal.[存储地域].eovod.com`	永久密钥	上传至指定存储桶

存储桶内网预置域名

专业版应用为每个存储桶预置了一个内网域名，客户可使用该域名在腾讯云服务器通过内网上传文件至相同地域的存储桶。

使用永久密钥通过 AWS S3 SDK 上传文件

下面介绍在常用的编程语言中，如何进行适配以上传文件至专业版应用的存储桶。
假设专业版应用 ID 为 `1234567890` ，存储桶的存储地域为 `ap-guangzhou` ，存储桶 ID 为 `bucketid1` 。
本实例使用的存储桶内网访问域名为： `bucketid1.vodsrc-internal.ap-guangzhou.eovod.com` 。

准备工作

1. 创建专业版应用和存储桶

   在 云点播控制台 创建好专业版应用和存储桶，具体操作步骤参见 快速入门 和 创建存储桶 文档。
2. 获取应用级别永久密钥对

   从专业版应用的密钥管理中，获取密钥对 `AccessKeyId` 和 `SecretAccessKey` ，具体操作步骤参见 密钥管理 文档。
3. 验证当前是否是内网访问

   内网域名必须在与存储桶相同地域的腾讯云服务器上才能访问，否则会得到404响应码。可以尝试在服务器上使用 `nslookups` 等指令进行域名解析，若得到形如 `10.*.*.*` 、 `100.*.*.*` 或 `169.254.*.*` 等形式的IP，则一般可以进行内网访问。

上传示例

常用语言初始化 S3 客户端并上传文件，代码实现如下所示：

GO

```
// Package main
// 本示例基于 AWS SDK for Go v2 service/s3 v1.72.3 实现。
// AWS SDK for Go v2 service/s3 v1.73.0 及以后版本需禁用 S3 的默认完整性保护。
// 具体参考: https://github.com/aws/aws-sdk-go-v2/discussions/2960
package main

import (
    "context"
    "errors"
    "fmt"
    "log"
    "net/url"
    "os"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/credentials"
    "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
    "github.com/aws/aws-sdk-go-v2/service/s3"
    smep "github.com/aws/smithy-go/endpoints"
    "github.com/aws/smithy-go/logging"
)


// customEndpointResolverV2 自定义接入点
type customEndpointResolverV2 struct{}

// ResolveEndpoint 自定义接入点
func (r *customEndpointResolverV2) ResolveEndpoint(ctx context.Context,
    params s3.EndpointParameters) (smep.Endpoint, error) {
    if params.Bucket == nil || params.Region == nil {
        return smep.Endpoint{}, errors.New("invalid endpoint param")
    }
    return smep.Endpoint{
        URI: url.URL{
            Scheme: "http",
            Host:   fmt.Sprintf("%s.vodsrc-internal.ap-
guangzhou.eovod.com", *params.Bucket),
```

```
        },
    }, nil
}

func main() {
    // new s3 client
    s3cli := s3.New(s3.Options{
        Credentials: credentials.NewStaticCredentialsProvider(
            "AccessKeyId",     // 填入密钥对中的 AccessKeyId
            "SecretAccessKey", // 填入密钥对中的 SecretAccessKey
            ""),               // 永久密钥无需填充 SessionToken
        EndpointResolverV2: new(customEndpointResolverV2),         // 自定义接入点
        UsePathStyle:       false,                                 // 请求禁用 path style
        Logger:             logging.NewStandardLogger(os.Stdout), // 日志打印到标准输出流
        ClientLogMode:      aws.LogRequest | aws.LogResponse,      // 打印请求头和响应头
        Region:             "auto",                                // 固定填写 auto
    })

    // 打开本地文件
    file, err := os.Open("demo.mp4") // 本地文件路径
    if err != nil {
        log.Fatalf("failed to open file: %v", err)
        return
    }
    defer file.Close()

    // 上传文件到 S3
    // 创建上传器
    uploader := manager.NewUploader(s3cli, func(u *manager.Uploader) {
        u.PartSize = 10 * 1024 * 1024 // 设置分片大小为 10MB
        u.Concurrency = 5             // 设置并发上传的分片数
    })
    // 上传文件
    result, err := uploader.Upload(context.TODO(), &s3.PutObjectInput{
        Bucket: aws.String("bucketid1"),        // Bucket 设置为点播专业版应用中的存储桶 ID
        Key:    aws.String("upload/demo.mp4"), // 媒体文件在存储桶中的路径
        Body:   file,
```

```
    })

    if err != nil {
        log.Fatalf("failed to upload file: %v", err)
        return
    }
    log.Printf("etag: %s", *result.ETag) // 获取上传文件 etag
}
```

Java

```
// 本示例基于 AWS SDK for Java v2 service/s3 v2.31.35 实现。
// AWS SDK for Java 2.30.0 及以后版本需禁用 S3 的默认完整性保护。
// 具体参考: https://github.com/aws/aws-sdk-java-v2/issues/5801
import software.amazon.awssdk.auth.credentials.AwsBasicCredentials;
import
software.amazon.awssdk.auth.credentials.StaticCredentialsProvider;
import software.amazon.awssdk.http.nio.netty.NettyNioAsyncHttpClient;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3AsyncClient;
import software.amazon.awssdk.services.s3.S3Configuration;
import software.amazon.awssdk.transfer.s3.S3TransferManager;
import software.amazon.awssdk.transfer.s3.model.UploadFileRequest;
import
software.amazon.awssdk.transfer.s3.progress.LoggingTransferListener;
import software.amazon.awssdk.core.checksums.RequestChecksumCalculation;
import software.amazon.awssdk.core.checksums.ResponseChecksumValidation;

import java.io.File;
import java.net.URI;
import java.util.concurrent.CompletableFuture;

public class Main {
    public static void main(String[] args) {
        S3AsyncClient s3AsyncClient = null;
        S3TransferManager transferManager = null;

        try {
            // 专业版应用永久密钥 AccessKeyId
            AwsBasicCredentials credentials =
AwsBasicCredentials.create(
                    "AccessKeyId", // 专业版应用永久密钥 AccessKeyId
```

```
                    "SecretAccessKey" // 专业版应用永久密钥 SecretAccessKey
            );
            String region = "ap-guangzhou"; // 专业版应用存储桶地域
            String bucketId = "bucketid1"; // 专业版应用存储桶ID
            String filePath = "demo.mp4"; // 本地文件路径
            String key = "upload/demo.mp4"; // 文件在存储桶中的路径
            String endpointUrl = String.format("http://vodsrc-
internal.%s.eovod.com", region); // 桶级别预置域名端点

            // 使用自定义端点构建S3异步客户端
            s3AsyncClient = S3AsyncClient.builder()

.credentialsProvider(StaticCredentialsProvider.create(credentials)) // 设
置凭证提供者
                    .endpointOverride(URI.create(endpointUrl)) // 设置端点
                    .region(Region.of("auto")) // 设置区域 auto 自动选择

.httpClient(NettyNioAsyncHttpClient.builder().build()) // 设置HTTP客户端
                    .serviceConfiguration(S3Configuration.builder()
                            .pathStyleAccessEnabled(false) // 设置路径样式
访问，桶级别需明确禁用 PathStyle 模式
                            .build())

.requestChecksumCalculation(RequestChecksumCalculation.WHEN_REQUIRED)

.responseChecksumValidation(ResponseChecksumValidation.WHEN_REQUIRED)
                    .build();

            // 使用S3异步客户端创建S3TransferManager
            transferManager = S3TransferManager.builder()
                    .s3Client(s3AsyncClient)
                    .build();

            File fileToUpload = new File(filePath);

            // 创建上传请求并添加传输监听器用于进度跟踪
            UploadFileRequest uploadRequest =
UploadFileRequest.builder()
                    .putObjectRequest(builder -> builder
                            .bucket(bucketId)
                            .key(key))

.addTransferListener(LoggingTransferListener.create())
```

```
                    .source(fileToUpload)
                    .build();

            // 开始上传
            CompletableFuture<Void> future =
transferManager.uploadFile(uploadRequest)
                    .completionFuture()
                    .thenAccept(response -> {
                        System.out.println("上传成功! ETag: " +
response.response().eTag());
                    });

            // 等待上传完成
            future.join();

            System.out.println("上传任务完成");
        } catch (Exception e) {
            System.err.println("上传失败: " + e.getMessage());
            e.printStackTrace();
        } finally {
            // 资源释放
            try {
                if (transferManager != null) {
                    transferManager.close();
                }
                if (s3AsyncClient != null) {
                    s3AsyncClient.close();
                }
                System.out.println("资源已释放");
            } catch (Exception e) {
                System.err.println("关闭资源时发生错误: " +
e.getMessage());
            }
        }
    }
}
```

C++

```
// 本示例基于 AWS SDK for C++ v1.11.560 实现。
// AWS SDK for C++ v1.11.486 及以后版本需禁用 S3 的默认完整性保护。
// 具体参考: https://github.com/aws/aws-sdk-cpp/issues/3253
```

```
#include <aws/core/Aws.h>
#include <aws/core/auth/AWSCredentials.h>
#include <aws/core/client/ClientConfiguration.h>
#include <aws/s3/S3Client.h>
#include <aws/transfer/TransferManager.h>
#include <iostream>

int main()
{
    // AWS SDK 初始化
    Aws::SDKOptions options;
    options.loggingOptions.logLevel =
Aws::Utils::Logging::LogLevel::Info;
    Aws::InitAPI(options);

    {
        // 定义认证和端点信息
        const Aws::String accessKeyId = "AccessKeyId";          // 填入点播
专业版密钥对中的 AccessKeyId
        const Aws::String secretAccessKey = "SecretAccessKey"; // 填入点播
专业版密钥对中的 SecretAccessKey
        const Aws::String region = "ap-guangzhou";              // 存储桶所
在地域
        const Aws::String bucketId = "bucketid1";               // 点播专业
版应用中的存储桶 ID
        const Aws::String keyName = "upload/demo.mp4";          // 文件在存
储桶中的路径
        const Aws::String filePath = "demo.mp4";                // 本地文件
路径

        // 配置客户端
        Aws::Client::ClientConfiguration clientConfig;
        clientConfig.region = "auto";
// 固定填写 auto
        clientConfig.scheme = Aws::Http::Scheme::HTTP;
// 协议使用 HTTP
        clientConfig.enableEndpointDiscovery = false;
// 禁用 endpoint discovery
        clientConfig.verifySSL = true;
// 启用 SSL 证书验证
        clientConfig.endpointOverride = "vodsrc-internal." + region +
".eovod.com"; // 桶级别预置域名
        // 禁用完整性校验
```

```
        clientConfig.checksumConfig.requestChecksumCalculation =
Aws::Client::RequestChecksumCalculation::WHEN_REQUIRED;
        clientConfig.checksumConfig.responseChecksumValidation =
Aws::Client::ResponseChecksumValidation::WHEN_REQUIRED;

        // 设置认证信息
        Aws::Auth::AWSCredentials credentials(accessKeyId,
secretAccessKey); // 填入点播专业版永久密钥

        // 创建 S3 客户端
        auto s3Client = Aws::MakeShared<Aws::S3::S3Client>(
            "S3Client",
            credentials,
            clientConfig,
            Aws::Client::AWSAuthV4Signer::PayloadSigningPolicy::Never,
// 禁用 payload signing
            true
// 桶级别预置域名需明确使用 VirtualAddressing
        );

        // 创建线程池执行器
        auto executor =
Aws::MakeShared<Aws::Utils::Threading::PooledThreadExecutor>("executor",
10);

        // 配置 TransferManager
        Aws::Transfer::TransferManagerConfiguration
transferConfig(executor.get());
        transferConfig.s3Client = s3Client;

        // 设置分片大小为 10MB
        transferConfig.bufferSize = 10 * 1024 * 1024;

        // 创建 TransferManager
        auto transferManager =
Aws::Transfer::TransferManager::Create(transferConfig);

        std::cout << "Starting file upload: " << filePath << " to " <<
bucketId << "/" << keyName << std::endl;

        // 执行上传
        auto uploadHandle = transferManager->UploadFile(
            filePath,                               // 本地文件路径
```

```
            bucketId,                                   // 存储桶ID
            keyName,                                    // 对象键（存储路径）
            "application/octet-stream",                 // 内容类型
            Aws::Map<Aws::String, Aws::String>() // 元数据
        );

        // 等待上传完成
        uploadHandle->WaitUntilFinished();

        // 检查上传状态
        if (uploadHandle->GetStatus() ==
Aws::Transfer::TransferStatus::COMPLETED)
        {
            std::cout << "File upload completed successfully!" <<
std::endl;
        }
        else
        {
            auto lastError = uploadHandle->GetLastError();
            std::cerr << "File upload failed: " <<
lastError.GetMessage() << std::endl;
        }
    }

    // 清理SDK资源
    Aws::ShutdownAPI(options);
    return 0;
}
```

Python

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
本示例基于 AWS SDK for Python v1.38.7 实现。
Boto3 的 AWS SDK for Python v1.36.0 及以后版本需禁用 S3 的默认完整性保护。
具体参考：https://github.com/boto/boto3/issues/4392
"""

import boto3
from botocore.config import Config
```

```python
from botocore.exceptions import ClientError

# 常量定义
REGION = "ap-guangzhou"  # 存储桶地域
BUCKET_ID = "bucketid1"  # 填入点播专业版应用中的存储桶 ID
FILE_PATH = "demo.mp4"  # 本地文件路径
OBJECT_KEY = "upload/demo.mp4"  # 文件在存储桶中的路径

# 创建 S3 客户端
s3_client = boto3.client(
    "s3",
    aws_access_key_id="AccessKeyId",  # 填入密钥对中的 AccessKeyId
    aws_secret_access_key="SecretAccessKey",  # 填入密钥对中的
SecretAccessKey
    endpoint_url=f"http://vodsrc-internal.{REGION}.eovod.com",  # 桶级别
预置上传域名
    config=Config(
        s3={"addressing_style": "virtual"},  # 使用 virtual hosted-style
        request_checksum_calculation="when_required",  # Boto3 的 AWS SDK
for Python v1.36.0 及以后版本需禁用 S3 的默认完整性保护。
        response_checksum_validation="when_required",  # Boto3 的 AWS SDK
for Python v1.36.0 及以后版本需禁用 S3 的默认完整性保护。
    ),
)

try:
    # 上传文件
    response = s3_client.upload_file(
        Bucket=BUCKET_ID,  # 填入点播专业版应用中的存储桶 ID
        Key=OBJECT_KEY,  # 文件在存储桶中的路径
        Filename=FILE_PATH,  # 本地文件路径
    )
    print(response)
except ClientError as e:
    print(f"Error: {e}")
```

⚠ 注意：

- 使用 AWS SDK 请确认数据完整性保护特性，详情请参考 AWS SDK 的相关文档 Data Integrity Protections for Amazon S3。
- 使用存储桶预置公网域名，路径必须为 VirtualStyle 格式。