智能创作

开发文档

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。

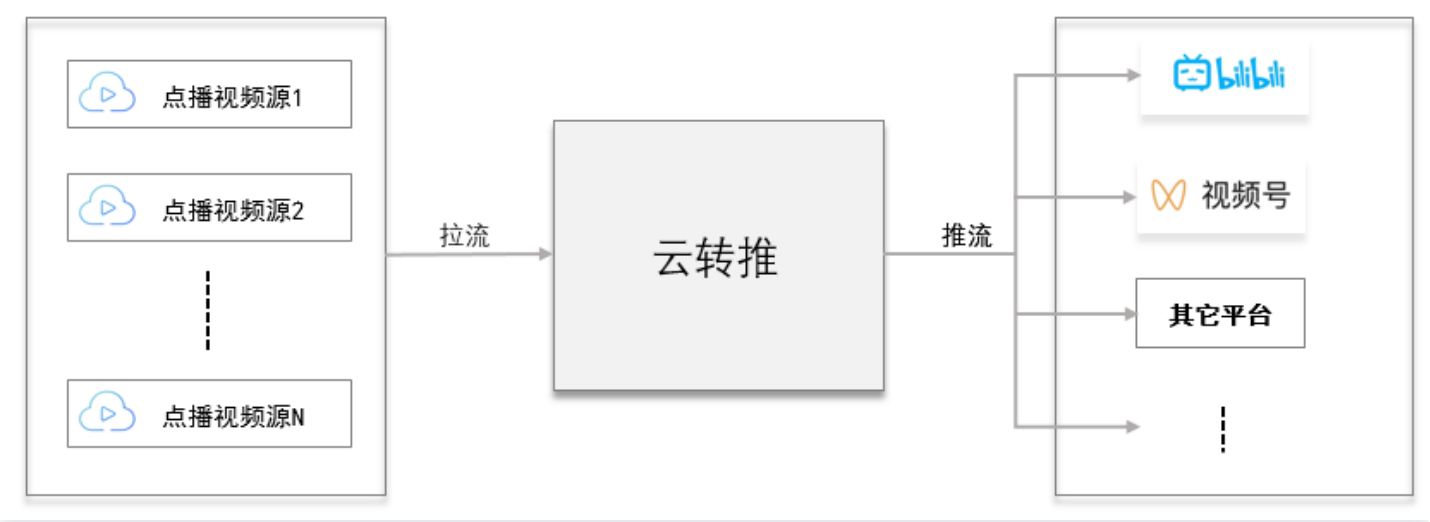
文档目录

开发文档
最佳实践
伪直播（点播转直播）

最近更新时间：2024-12-04 11:57:21

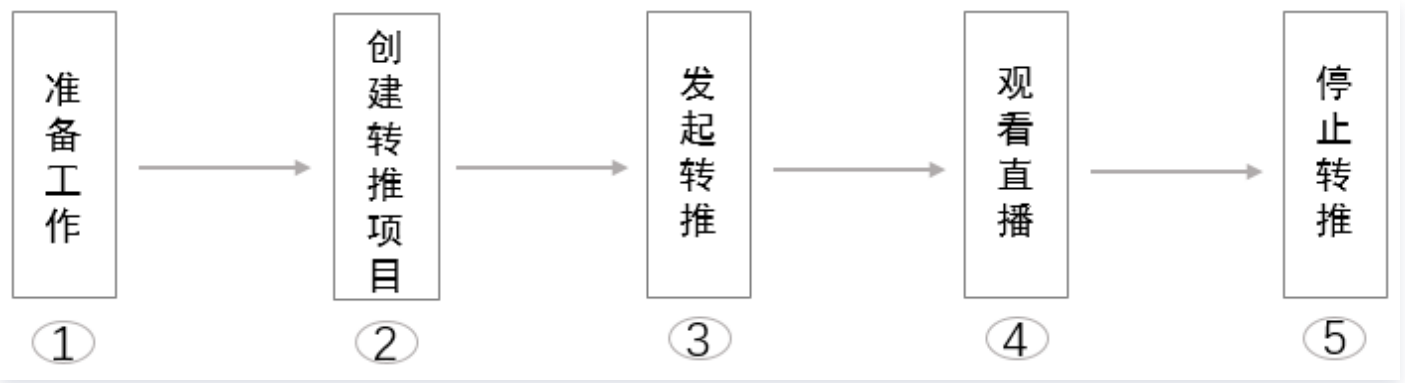
功能概述

伪直播(点播转直播)是将点播视频通过**云转推**功能转化为直播流，并将其分发到不同的直播平台，支持多个点播视频循环播放、单次播放等功能。可应用于在线教育、活动直播、电竞赛事等将录播视频转为直播的场景。整个方案的简要技术架构为：



实现步骤概要

使用云转推实现伪直播(点播转直播)功能，只需要按如下几步操作即可完成：



步骤1：准备工作

开通云点播

请参考 云点播 – 快速入门 – 步骤1 开通云点播服务。

创建智能创作平台

请参考 智能创作 – 控制台指南 – 创建平台 创建平台。

获取 API 密钥

请求云 API 需要使用到开发者的 API 密钥（即 SecretId 和 SecretKey）。如果还未创建过密钥，请参见 创建密钥文档 生成新的 API 密钥。

> **说明**
> 如果已创建过密钥，请参见 查看密钥文档 获取 API 密钥。

准备点播视频源

请自行准备点播视频源。为方便测试，这里准备了三个测试视频，可直接测试使用：

- 测试视频1：
  `http://1810000000.vod2.myqcloud.com/b64e98acvodcq1810000000/c20be67852858908095998 44312/f0.mp4`
- 测试视频2：
  `http://1810000000.vod2.myqcloud.com/b64e98acvodcq1810000000/f697474752858908100079 30755/f0.mp4`
- 测试视频3：
  `http://1810000000.vod2.myqcloud.com/b64e98acvodcq1810000000/edeb768352858908100143 30867/f0.mp4`

> **注意**
> 点播输入源必须符合伪直播场景要求，否则会出现转推异常，详情请参见 点播视频源预处理最佳实践 。

准备直播推流及播放域名

请参考 云直播 – 标准直播 – 快速入门 准备直播推流域名及播放域名，如果您已经有直播推流域名及播放域名可忽略。文中后续以如下直播推流和播放域名作为示例：

- 推流域名1：`rtmp://push.example.com/cme/live1`
- 播放域名1：`https://play.example.com/cme/live1.flv`
- 推流域名2：`rtmp://push.example.com/cme/live2`
- 播放域名2：`https://play.example.com/cme/live2.flv`

> **注意**

> 这里推流域名及播放域名仅为示例，不可推流和播放。线上使用时，请换成您系统使用的直播推流域名及播放域名。

步骤2：创建云转推项目

创建云转推项目 API 的接口为 创建项目 。为了简化操作，建议您使用 API Explorer 创建项目。本文中使用的点播视频源以 准备工作-准备点播视频源 准备的视频源为例，可以直接使用该视频测试。这里通过一个接口调用示例说明关键参数设置：

```
https://cme.tencentcloudapi.com/?Action=CreateProject
&Platform=test
&Category=STREAM_CONNECT
&Name=云转推项目
&Owner.Id=user_id_700099304
&Owner.Type=PERSON
&StreamConnectProjectInput.MainInput.InputType=VodPull
&StreamConnectProjectInput.MainInput.VodPullInputInfo.inputUrls.0=http://
1810000000.vod2.myqcloud.com/b64e98acvodcq1810000000/c20be678528589080959
9844312/f0.mp4
&StreamConnectProjectInput.MainInput.VodPullInputInfo.inputUrls.1=http://
1810000000.vod2.myqcloud.com/b64e98acvodcq1810000000/f6974747528589081000
7930755/f0.mp4
&StreamConnectProjectInput.MainInput.VodPullInputInfo.inputUrls.2=http://
1810000000.vod2.myqcloud.com/b64e98acvodcq1810000000/edeb7683528589081001
4330867/f0.mp4
&StreamConnectProjectInput.Outputs.0.Id=1
&StreamConnectProjectInput.Outputs.0.Name=推流1
&StreamConnectProjectInput.Outputs.0.Type=URL
&StreamConnectProjectInput.Outputs.0.PushUrl=rtmp://push.example.myqcloud
.com/cme/live1
&StreamConnectProjectInput.Outputs.1.Id=2
&StreamConnectProjectInput.Outputs.1.Name=推流2
&StreamConnectProjectInput.Outputs.1.Type=URL
&StreamConnectProjectInput.Outputs.1.PushUrl=rtmp://push.example.myqcloud
.com/cme/live2
&<公共请求参数>
```

下面基于智能创作的 Java SDK 提供一个简单的示例：

```
import com.tencentcloudapi.common.Credential;
import com.tencentcloudapi.common.profile.ClientProfile;
import com.tencentcloudapi.common.profile.HttpProfile;
```

```
import com.tencentcloudapi.common.exception.TencentCloudSDKException;
import com.tencentcloudapi.cme.v20191029.CmeClient;
import com.tencentcloudapi.cme.v20191029.models.*;

public class CreateProject
{
    public static void main(String [] args) {
        try{
            // 需要传入准备工作中获取的 SecretId，SecretKey，此处还需注意密钥对的
保密
            Credential cred = new Credential("SecretId", "SecretKey");
            HttpProfile httpProfile = new HttpProfile();
            httpProfile.setEndpoint("cme.tencentcloudapi.com");
            ClientProfile clientProfile = new ClientProfile();
            clientProfile.setHttpProfile(httpProfile);
            CmeClient client = new CmeClient(cred, "", clientProfile);

            // 以下为关键参数部分
            CreateProjectRequest req = new CreateProjectRequest();
            // 平台，填写准备工作中创建的平台
            req.setPlatform("test");
            req.setName("云转推测试");

            // 项目归属，目前仅支持归属个人的项目，请参考注意事项部分
            Entity entity1 = new Entity();
            entity1.setType("PERSON");
            entity1.setId("user_id_700099304");
            req.setOwner(entity1);

            // 项目类型，固定为 STREAM_CONNECT
            req.setCategory("STREAM_CONNECT");
            req.setDescription("云转推测试项目");
            StreamConnectProjectInput streamConnectProjectInput1 = new
StreamConnectProjectInput();
            StreamInputInfo streamInputInfo1 = new StreamInputInfo();
            // 视频源类型，固定为 VodPull
            streamInputInfo1.setInputType("VodPull");
            VodPullInputInfo vodPullInputInfo1 = new VodPullInputInfo();

            // 点播视频源列表，将按顺序播放
            String[] inputUrls1 = {

"http://1810000000.vod2.myqcloud.com/b64e98acvodcq1810000000/bec21e445285
890810010546287/f0.mp4",
```

```
"http://1810000000.vod2.myqcloud.com/b64e98acvodcq1810000000/f69747475285
890810007930755/f0.mp4",

"http://1810000000.vod2.myqcloud.com/b64e98acvodcq1810000000/edeb76835285
890810014330867/f0.mp4"
            };
            vodPullInputInfo1.setInputUrls(inputUrls1);

            // 循环次数，注意：目前支持播放一次、指定播放次数及无限循环播放。填0为播
放一次
            vodPullInputInfo1.setLoopTimes(0L);
            streamInputInfo1.setVodPullInputInfo(vodPullInputInfo1);
            streamConnectProjectInput1.setMainInput(streamInputInfo1);

            // 设置输出
            StreamConnectOutput[] streamConnectOutputs1 = new
StreamConnectOutput[2];
            StreamConnectOutput streamConnectOutput1 = new
StreamConnectOutput();
            streamConnectOutput1.setId("1");
            streamConnectOutput1.setName("转推流1");
            streamConnectOutput1.setType("URL");

streamConnectOutput1.setPushUrl("rtmp://push.example.myqcloud.com/cme/liv
e1");
            streamConnectOutputs1[0] = streamConnectOutput1;

            StreamConnectOutput streamConnectOutput2 = new
StreamConnectOutput();
            streamConnectOutput2.setId("2");
            streamConnectOutput2.setName("转推流2");
            streamConnectOutput2.setType("URL");

streamConnectOutput2.setPushUrl("rtmp://push.example.myqcloud.com/cme/liv
e2");
            streamConnectOutputs1[1] = streamConnectOutput2;

            streamConnectProjectInput1.setOutputs(streamConnectOutputs1);
            req.setStreamConnectProjectInput(streamConnectProjectInput1);

            // 返回的 resp 是一个 CreateProjectResponse 的实例，与请求对象对应
            CreateProjectResponse resp = client.CreateProject(req);
```

```
            // 输出 json 格式的字符串回包，可拿到创建项目返回的项目 Id
            System.out.println(CreateProjectResponse.toJsonString(resp));
        } catch (TencentCloudSDKException e) {
            System.out.println(e.toString());
        }
    }
}
```

> ⚠ 注意
>
> - 关于归属 Owner 参数，请参考文档 账号 部分。
> - 本示例没有添加备输入源，添加方式与主输入源一致。添加备输入源是为防止主输入源出现异常时切换到备输入源转推。
> - 创建项目成功后，可拿到云转推的项目 ID，后续操作步骤都以项目 ID：`cmepid_6107a30cd488c00001b1ab8b` 为例。

步骤3：发起转推

发起转推 API 接口为 操作云转推项目 。为了简化操作，建议您使用 API Export 发起转推。这里通过一个接口调用示例说明关键参数设置：

```
https://cme.tencentcloudapi.com/?Action=HandleStreamConnectProject
&Platform=test
&ProjectId=cmepid_6107a30cd488c00001b1ab8b
&Operation=Start
&<公共请求参数>
```

下面基于智能创作的 Java SDK 提供一个简单的示例：

```
import com.tencentcloudapi.common.Credential;
import com.tencentcloudapi.common.profile.ClientProfile;
import com.tencentcloudapi.common.profile.HttpProfile;
import com.tencentcloudapi.common.exception.TencentCloudSDKException;
import com.tencentcloudapi.cme.v20191029.CmeClient;
import com.tencentcloudapi.cme.v20191029.models.*;

public class HandleStreamConnectProject
{
    public static void main(String [] args) {
        try{
            // 需要传入准备工作中获取的 SecretId，SecretKey，此处还需注意密钥对的保密
```

```
            Credential cred = new Credential("SecretId", "SecretKey");
            HttpProfile httpProfile = new HttpProfile();
            httpProfile.setEndpoint("cme.tencentcloudapi.com");
            ClientProfile clientProfile = new ClientProfile();
            clientProfile.setHttpProfile(httpProfile);
            CmeClient client = new CmeClient(cred, "", clientProfile);

            // 以下为关键参数
            HandleStreamConnectProjectRequest req = new
HandleStreamConnectProjectRequest();

            // 平台，填写准备工作中创建的平台
            req.setPlatform("test");
            // 项目 Id，填步骤2 创建项目时返回的项目 Id
            req.setProjectId("cmepid_6107a30cd488c00001b1ab8b");
            // 操作参数，固定为 "Start"
            req.setOperation("Start");

            // 返回的 resp 是一个 HandleStreamConnectProjectResponse 的实
例，与请求对象对应
            HandleStreamConnectProjectResponse resp =
client.HandleStreamConnectProject(req);

            // 输出 json 格式的字符串回包

System.out.println(HandleStreamConnectProjectResponse.toJsonString(resp))
;
        } catch (TencentCloudSDKException e) {
            System.out.println(e.toString());
        }
    }
}
```

步骤4：观看直播

发起转推完成后，可以使用 VLC 等播放器观看直播(测试播放域名为：`https://play.example.myqcloud.com/cme/live1.flv` 或者 `https://play.example.myqcloud.com/cme/live2.flv`)，将可看到直播画面。

步骤5：停止转推

停止转推接口为 操作云转推项目 。为了简化操作，建议您使用 API Export 停止转推。这里通过一个接口调用示例说明关键参数设置：

```
https://cme.tencentcloudapi.com/?Action=HandleStreamConnectProject
&Platform=test
&ProjectId=cmepid_6107a30cd488c00001b1ab8b
&Operation=Stop
&<公共请求参数>
```

下面基于智能创作的 Java SDK 提供一个简单的示例：

```
import com.tencentcloudapi.common.Credential;
import com.tencentcloudapi.common.profile.ClientProfile;
import com.tencentcloudapi.common.profile.HttpProfile;
import com.tencentcloudapi.common.exception.TencentCloudSDKException;
import com.tencentcloudapi.cme.v20191029.CmeClient;
import com.tencentcloudapi.cme.v20191029.models.*;

public class HandleStreamConnectProject
{
    public static void main(String [] args) {
        try{
            // 需要传入准备工作中获取的 SecretId，SecretKey，此处还需注意密钥对的
保密
            Credential cred = new Credential("SecretId", "SecretKey");
            HttpProfile httpProfile = new HttpProfile();
            httpProfile.setEndpoint("cme.tencentcloudapi.com");
            ClientProfile clientProfile = new ClientProfile();
            clientProfile.setHttpProfile(httpProfile);
            CmeClient client = new CmeClient(cred, "", clientProfile);


            // 以下为关键参数
            HandleStreamConnectProjectRequest req = new
HandleStreamConnectProjectRequest();
            // 平台，填写准备工作中创建的平台
            req.setPlatform("test");
            // 项目 Id，填步骤2 创建项目时返回的项目 Id
            req.setProjectId("cmepid_6107a30cd488c00001b1ab8b");
            // 操作参数，固定为 "Stop"
            req.setOperation("Stop");

            // 返回的 resp 是一个 HandleStreamConnectProjectResponse 的实
例，与请求对象对应
```

```
            HandleStreamConnectProjectResponse resp =
client.HandleStreamConnectProject(req);

            // 输出 json 格式的字符串回包

System.out.println(HandleStreamConnectProjectResponse.toJsonString(resp))
;
        } catch (TencentCloudSDKException e) {
            System.out.println(e.toString());
        }
    }
}
```

点播视频源预处理最佳实践

实际推流过程中，如果点播输入源质量不可控，会造成转推后直播画面异常(花屏、倍速播放、卡顿等)，常见的异常有：

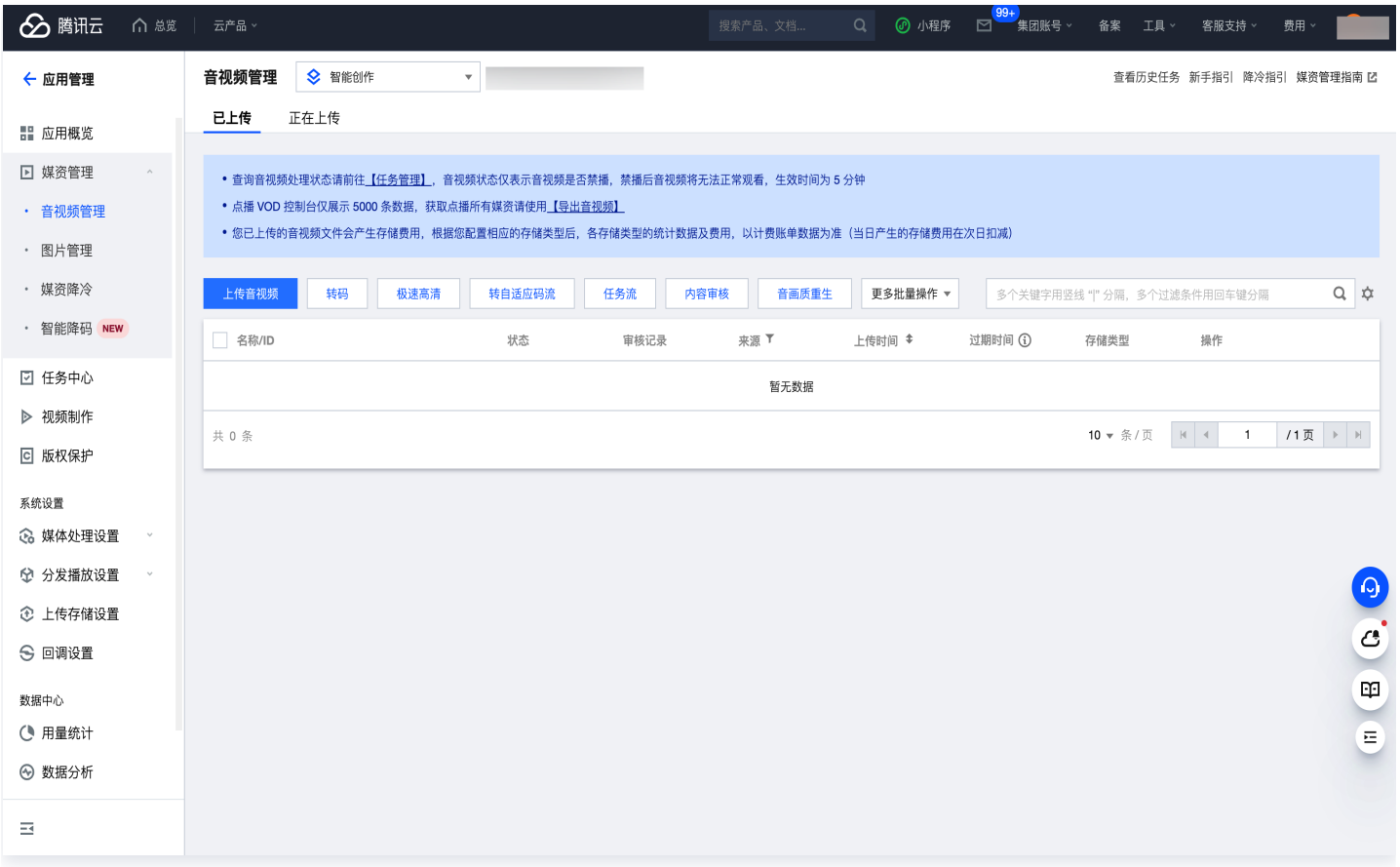
- 码率过高(超过 4M)，转推为直播流后，播放端可能因为网络带宽不够而出现播放卡顿。
- 输入源文件不在腾讯云点播存储，转推时无法走内网拉流，因外网异常出现拉流失败或者卡顿。
- 视频源封装格式异常(目前仅支持音频编码格式：LC-AAC, 视频编码格式：H264/HEVC)，造成转推画面倍速播放、卡顿。
- 输入源为多个点播视频源时，帧率和分辨率均不同，转推成直播流时，会造成直播画面花屏。
- 视频源为 FLV 封装格式，文件大于 2G，或者视频源为 hls 封装格式但 TS 分片数大于3000，可能会影响转推，造成转推后直播画面卡顿。
- 视频源音/视频 DTS 时间戳非顺序递增，出现回退或跳跃，会造成转推后直播画面卡顿。

为解决上述问题，需要将源视频上传到腾讯云点播走内网拉流，同时对源视频进行转码(也称之为“清洗”)。下面以使用云点播控制台及使用云点播 API 上传视频并清洗为例说明简要操作步骤。

使用点播控制台上传并清洗源视频

步骤1：上传视频

将点播视频源上传腾讯云点播，转推时可以走内网拉流，避免拉流时网络抖动等影响。可参考 云点播 - 快速入门 - 步骤2 上传一个测试视频，单击 此处 查看本测试视频，如下图所示：

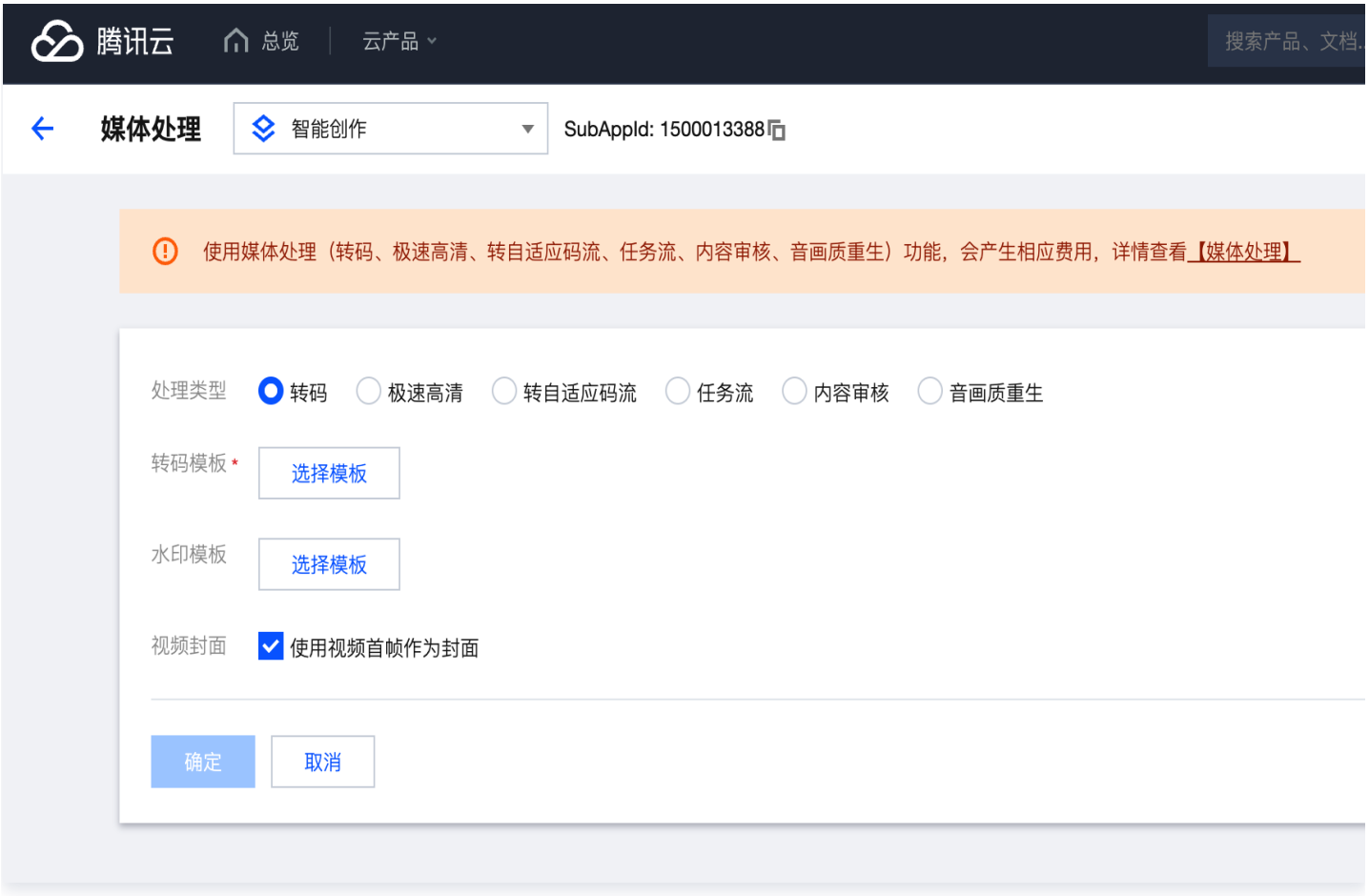


> **说明**
>
> 建议使用较短的视频文件进行测试（例如时长为几十秒的视频），避免转码过程耗时太长。

步骤2：发起视频清洗

1. 在云点播控制台 视频管理 页面勾选新上传的测试视频，然后单击**视频处理**：

2. 在弹框中，处理类型选择**转码**，然后单击**转码模板**：



3. 选择所需的转码模板，目前系统已经预置好了视频源清洗转码模板 `VodToLiveAdaptiveTranscode`（模板 ID 80000），然后单击**确定**。

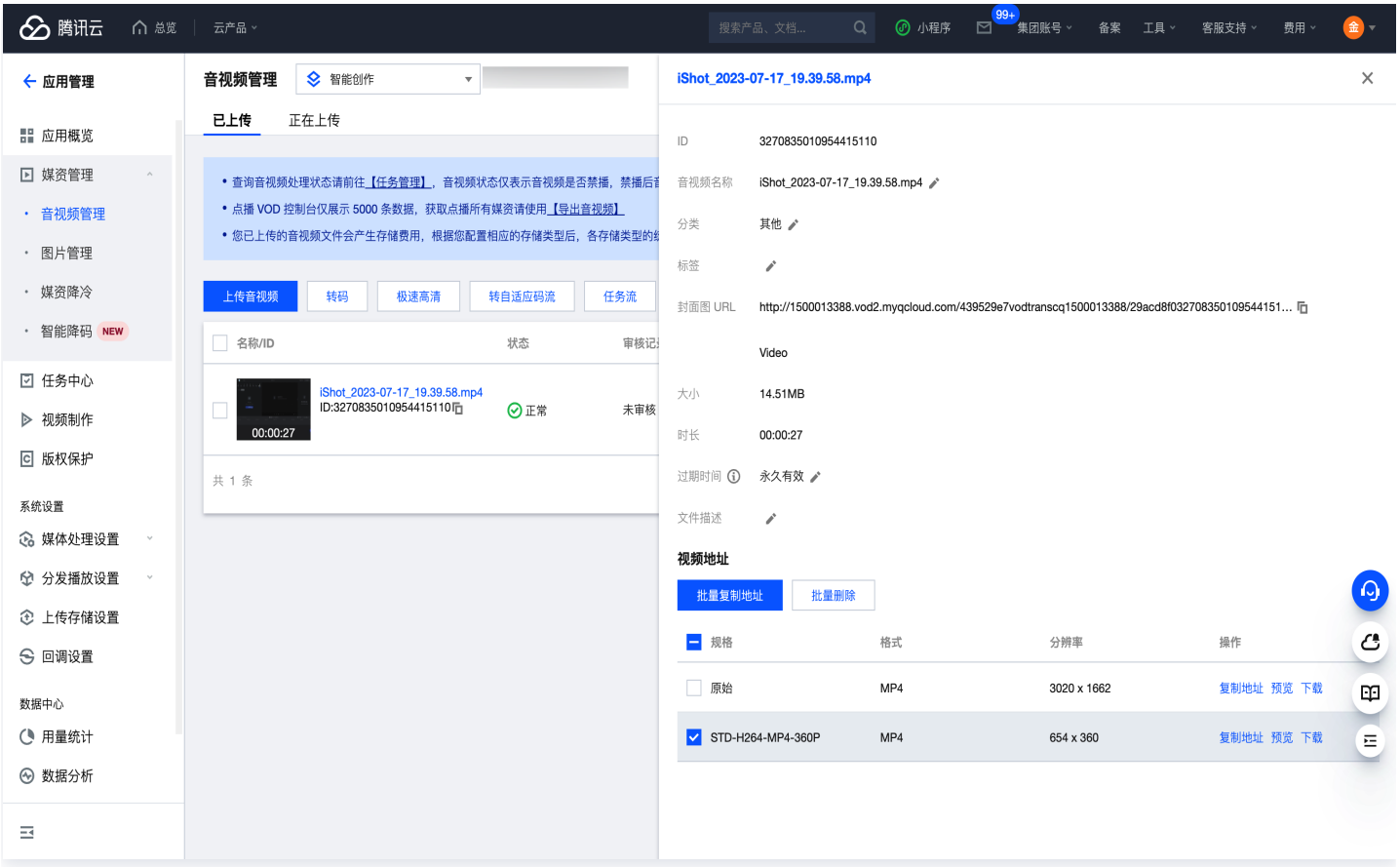
4. 单击**确定**，发起转码：

5. 在**任务管理**的列表中可见当前转码任务的状态为“处理中”，表示视频正在转码：

步骤3：查看转码结果

1. 进入控制台 任务管理，等待列表中的测试视频状态变为“正常”（表示转码已完成）。单击测试视频右侧的**管理**，进入任务详情页面：

2. 在**基本信息**标签页下的**标准转码列表** 中，转出了 `VodToLiveAdaptiveTranscode` 规格的视频。可以单击**复制地址**复制转码视频的 URL，然后作为云转推的点播输入源 URL。



调用云 API 上传并清洗源视频

实际使用过程中，需要调用腾讯云点播 API 来进行点播视频上传及清洗，下面简要介绍操作步骤。

1. **视频上传到云点播：**
   将点播视频上传到腾讯云点播，转推时可以走内网拉流，避免拉流时网络抖动影响。腾讯云点播提供了多种方式将视频上传到云点播，请参见 云点播-最佳实践-媒体上传 。
2. **点播视频进行转码清洗：**
   为保证转推输入点播视频源符合要求，建议使用云点播的转码功能进行视频清洗，然后使用转码后的视频源进行转推。点播侧提供了完整的转码方案，具体接入方法请参见 云点播-最佳实践-如何对视频进行转码 。
   关于转码模板参数配置问题，目前系统已经配置好了转码模板 ID： `80000` ，请直接使用。该转码模板配置如下：

配置项	配置内容
封装格式	MP4
视频编码格式	H264
宽高	同源
帧率	同源（但最低不低于 25fps，不高于 60fps）
音频编码格式	AAC
声道数	2
音频采样率	44100Hz
音视频码率	根据目标宽高自适应，但最大不高于9M

注意

- 因 UGC 生产的视频源质量不可控，强烈建议按上述方案将视频上传到腾讯云点播并发起清洗。
- 为简化接口调用，云点播侧可以在上传视频后，指定视频处理任务流进行转码清洗。具体实现方案请参见 上传视频后自动转码 ，云点播侧已配置好了一个任务流 `VodToLivePreProcessProcedure` ，请直接使用。

多平台同步直播

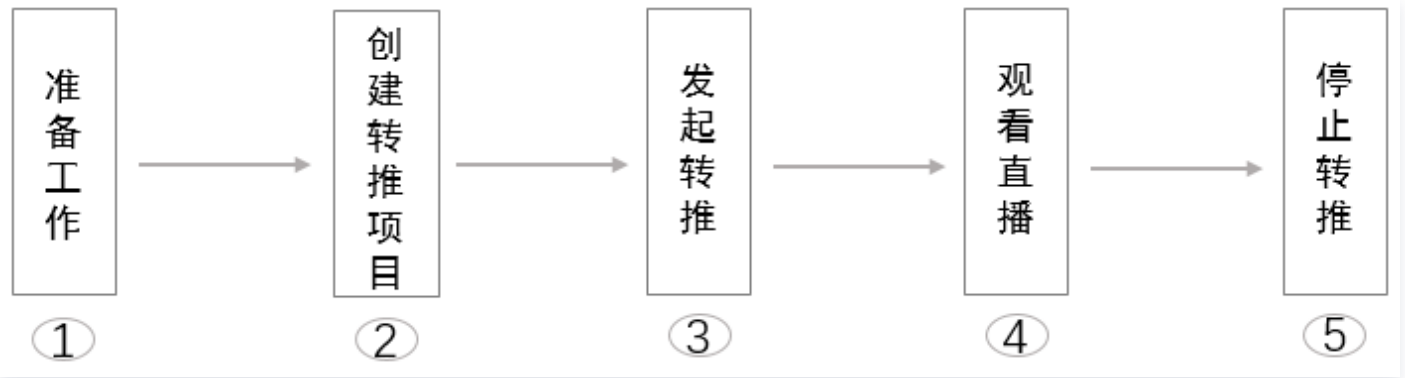
最近更新时间：2023-09-15 17:53:22

功能概述

多平台直播是通过**云转推**将一路视频流转推到视频号、微视、快手等多个第三方厂商同步开播或者转推到同一个直播厂商的多路流同步直播。整个方案的简要技术架构为：

实现步骤概要

使用云转推实现多平台直播功能，只需要按如下几步操作即可完成：



步骤1：准备工作

创建智能创作平台

请参考 智能创作 – 控制台指南 – 创建平台 创建平台。

获取 API 密钥

请求云 API 需要使用到开发者的 API 密钥（即 SecretId 和 SecretKey）。如果还未创建过密钥，请参见 创建密钥文档 生成新的 API 密钥；如果已创建过密钥，请参见 查看密钥文档 获取 API 密钥。

准备直播源

请自行准备需要转推的直播源 URL，文中以测试直播流 URL：`https://play.example.com/cme/src_live.flv` 为例。

> **注意**
> - 实际使用中，请替换直播源 URL 为您需要转推的直播源 URL，并保证已经推流。
> - 如果要实现直接推流模式，可以在创建转推项目完成后拿到推流地址，直接推流即可。
> - 直播源音/视频 DTS 时间戳需要顺序递增，不要出现回退或跳跃。

准备直播推流及播放域名

请参考 云直播 – 标准直播 – 快速入门 准备直播推流域名及播放域名，如果您已经有直播推流域名及播放域名可忽略。文中后续以如下直播推流和播放域名作为示例：

- 推流域名1：`rtmp://push.example.myqcloud.com/cme/live1`
- 播放域名1：`https://play.example.myqcloud.com/cme/live1.flv`
- 推流域名2：`rtmp://push.example.myqcloud.com/cme/live2`
- 播放域名2：`https://play.example.myqcloud.com/cme/live2.flv`

> **注意**
> 这里推流域名和播放域名仅为示例，不可推流及播放。线上使用时，请换成您系统使用的直播推流域名和播放域名。

步骤2：创建云转推项目

创建项目的云 API 接口为 创建项目 。为了简化操作，建议您使用 API Export 创建项目。本文中使用的直播源以测试直播源 URL：`https://play.example.com/cme/src_live.flv` 为例。这里通过一个接口调用示例说明关键参数设置：

```
https://cme.tencentcloudapi.com/?Action=CreateProject
&Platform=test
&Category=STREAM_CONNECT
&Name=stream_connect
&Owner.Id=user_id_109993839
&Owner.Type=PERSON
&StreamConnectProjectInput.MainInput.InputType=LivePull
&StreamConnectProjectInput.MainInput.LivePullInputInfo.InputUrl=https://p
lay.example.com/cme/src_live.flv
```

```
&StreamConnectProjectInput.Outputs.0.Id=1
&StreamConnectProjectInput.Outputs.0.Name=推流1
&StreamConnectProjectInput.Outputs.0.Type=URL
&StreamConnectProjectInput.Outputs.0.PushUrl=rtmp://push.example.com/cme/
live1
&StreamConnectProjectInput.Outputs.1.Id=2
&StreamConnectProjectInput.Outputs.1.Name=推流2
&StreamConnectProjectInput.Outputs.1.Type=URL
&StreamConnectProjectInput.Outputs.1.PushUrl=rtmp://push.example.com/cme/
live2
&<公共请求参数>
```

下面基于智能创作的 Java SDK 提供一个简单的示例：

```
import com.tencentcloudapi.common.Credential;
import com.tencentcloudapi.common.profile.ClientProfile;
import com.tencentcloudapi.common.profile.HttpProfile;
import com.tencentcloudapi.common.exception.TencentCloudSDKException;
import com.tencentcloudapi.cme.v20191029.CmeClient;
import com.tencentcloudapi.cme.v20191029.models.*;

public class CreateProject
{
    public static void main(String [] args) {
        try{
            // 需要传入准备工作中获取的 SecretId，SecretKey，此处还需注意密钥对的
保密
            Credential cred = new Credential("SecretId", "SecretKey");
            HttpProfile httpProfile = new HttpProfile();
            httpProfile.setEndpoint("cme.tencentcloudapi.com");
            ClientProfile clientProfile = new ClientProfile();
            clientProfile.setHttpProfile(httpProfile);
            CmeClient client = new CmeClient(cred, "", clientProfile);

            // 以下为关键参数部分
            CreateProjectRequest req = new CreateProjectRequest();
            // 平台，填写准备工作中创建的平台
            req.setPlatform("test");
            req.setName("直播多平台分发");

            // 项目归属，目前仅支持归属个人的项目，请参考注意事项部分
            Entity entity1 = new Entity();
            entity1.setType("PERSON");
```

```
            entity1.setId("user_id_109993839");
            req.setOwner(entity1);

            // 项目类型，固定为 STREAM_CONNECT
            req.setCategory("STREAM_CONNECT");
            req.setDescription("多平台分发测试");

            StreamConnectProjectInput streamConnectProjectInput1 = new
StreamConnectProjectInput();
            StreamInputInfo streamInputInfo1 = new StreamInputInfo();
            // 视频源类型，固定为 LivePull
            streamInputInfo1.setInputType("LivePull");
            LivePullInputInfo livePullInputInfo1 = new
LivePullInputInfo();
            // 直播视频源 URL

livePullInputInfo1.setInputUrl("https://play.example.com/cme/src_live.flv
");
            streamInputInfo1.setLivePullInputInfo(livePullInputInfo1);
            streamConnectProjectInput1.setMainInput(streamInputInfo1);

            // 设置输出
            StreamConnectOutput[] streamConnectOutputs1 = new
StreamConnectOutput[2];
            StreamConnectOutput streamConnectOutput1 = new
StreamConnectOutput();
            streamConnectOutput1.setId("1");
            streamConnectOutput1.setName("推流1");
            streamConnectOutput1.setType("URL");

streamConnectOutput1.setPushUrl("rtmp://push.example.com/cme/live1");
            streamConnectOutputs1[0] = streamConnectOutput1;

            StreamConnectOutput streamConnectOutput2 = new
StreamConnectOutput();
            streamConnectOutput2.setId("2");
            streamConnectOutput2.setName("推流2");
            streamConnectOutput2.setType("URL");

streamConnectOutput2.setPushUrl("rtmp://push.example.com/cme/live2");
            streamConnectOutputs1[1] = streamConnectOutput2;

            streamConnectProjectInput1.setOutputs(streamConnectOutputs1);
            req.setStreamConnectProjectInput(streamConnectProjectInput1);
```

```
            // 返回的 resp 是一个 CreateProjectResponse 的实例，与请求对象对应
            CreateProjectResponse resp = client.CreateProject(req);
            // 输出 json 格式的字符串回包，可拿到创建项目返回的项目 Id
            System.out.println(CreateProjectResponse.toJsonString(resp));
        } catch (TencentCloudSDKException e) {
            System.out.println(e.toString());
        }
    }
}
```

> **注意**
>
> - 本示例没有添加备输入源，添加方式与主输入源一致。添加备输入源为防止主输入源出现异常时切换到备输入源进行转推。
> - 创建项目成功后，可拿到云转推的项目 Id，后续步骤以 `cmepid_7107a50cd488c00003c1be9p` 为例。

步骤3：发起转推

发起转推 API 接口为 操作云转推项目 。为了简化操作，建议您使用 API Export 发起转推。这里通过一个接口调用示例说明关键参数设置：

```
https://cme.tencentcloudapi.com/?Action=HandleStreamConnectProject
&Platform=test
&ProjectId=cmepid_7107a50cd488c00003c1be9p
&Operation=Start
&<公共请求参数>
```

下面基于智能创作的 Java SDK 提供一个简单的示例：

```
import com.tencentcloudapi.common.Credential;
import com.tencentcloudapi.common.profile.ClientProfile;
import com.tencentcloudapi.common.profile.HttpProfile;
import com.tencentcloudapi.common.exception.TencentCloudSDKException;
import com.tencentcloudapi.cme.v20191029.CmeClient;
import com.tencentcloudapi.cme.v20191029.models.*;

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

```
            // 需要传入准备工作中获取的 SecretId，SecretKey，此处还需注意密钥对的
保密
            Credential cred = new Credential("SecretId", "SecretKey");
            HttpProfile httpProfile = new HttpProfile();
            httpProfile.setEndpoint("cme.tencentcloudapi.com");
            ClientProfile clientProfile = new ClientProfile();
            clientProfile.setHttpProfile(httpProfile);
            CmeClient client = new CmeClient(cred, "", clientProfile);

            // 以下为关键参数
            HandleStreamConnectProjectRequest req = new
HandleStreamConnectProjectRequest();

            // 平台，填写准备工作中创建的平台
            req.setPlatform("test");
            // 项目 Id，填步骤2 创建项目时返回的项目 Id
            req.setProjectId("cmepid_7107a50cd488c00003c1be9p");
            // 操作参数，固定为 "Start"
            req.setOperation("Start");

            // 返回的 resp 是一个 HandleStreamConnectProjectResponse 的实
例，与请求对象对应
            HandleStreamConnectProjectResponse resp =
client.HandleStreamConnectProject(req);

            // 输出 json 格式的字符串回包

System.out.println(HandleStreamConnectProjectResponse.toJsonString(resp))
;
        } catch (TencentCloudSDKException e) {
            System.out.println(e.toString());
        }
    }
}
```

步骤4：观看直播

发起转推完成后，可以使用 VLC 等播放器观看直播(测试时播放域名为：`https://play.example.com/cme/live1.flv` 或者 `https://play.example.com/cme/live2.flv`)，将可看到直播画面。

步骤5：停止转推

转推结束后，需要停止转推任务。停止转推接口为 操作云转推项目 。为了简化操作，建议您使用 API Explorer 停止转推。这里通过一个接口调用示例说明关键参数设置：

```
https://cme.tencentcloudapi.com/?Action=HandleStreamConnectProject
&Platform=test
&ProjectId=cmepid_7107a50cd488c00003c1be9p
&Operation=Stop
&<公共请求参数>
```

下面基于智能创作的 Java SDK 提供一个简单的示例：

```
import com.tencentcloudapi.common.Credential;
import com.tencentcloudapi.common.profile.ClientProfile;
import com.tencentcloudapi.common.profile.HttpProfile;
import com.tencentcloudapi.common.exception.TencentCloudSDKException;
import com.tencentcloudapi.cme.v20191029.CmeClient;
import com.tencentcloudapi.cme.v20191029.models.*;

public class HandleStreamConnectProject
{
    public static void main(String [] args) {
        try{
            // 需要传入准备工作中获取的 SecretId，SecretKey，此处还需注意密钥对的
保密
            Credential cred = new Credential("SecretId", "SecretKey");
            HttpProfile httpProfile = new HttpProfile();
            httpProfile.setEndpoint("cme.tencentcloudapi.com");
            ClientProfile clientProfile = new ClientProfile();
            clientProfile.setHttpProfile(httpProfile);
            CmeClient client = new CmeClient(cred, "", clientProfile);

            // 以下为关键参数
            HandleStreamConnectProjectRequest req = new
HandleStreamConnectProjectRequest();
            // 平台，填写准备工作中创建的平台
            req.setPlatform("test");
            // 项目 Id，填步骤2 创建项目时返回的项目 Id
            req.setProjectId("cmepid_7107a50cd488c00003c1be9p");
            // 操作参数，固定为 "Stop"
            req.setOperation("Stop");

            // 返回的 resp 是一个 HandleStreamConnectProjectResponse 的实
例，与请求对象对应
```

```
            HandleStreamConnectProjectResponse resp =
client.HandleStreamConnectProject(req);

            // 输出 json 格式的字符串回包

System.out.println(HandleStreamConnectProjectResponse.toJsonString(resp))
;
        } catch (TencentCloudSDKException e) {
            System.out.println(e.toString());
        }
    }
}
```

使用视频编辑模板导出视频

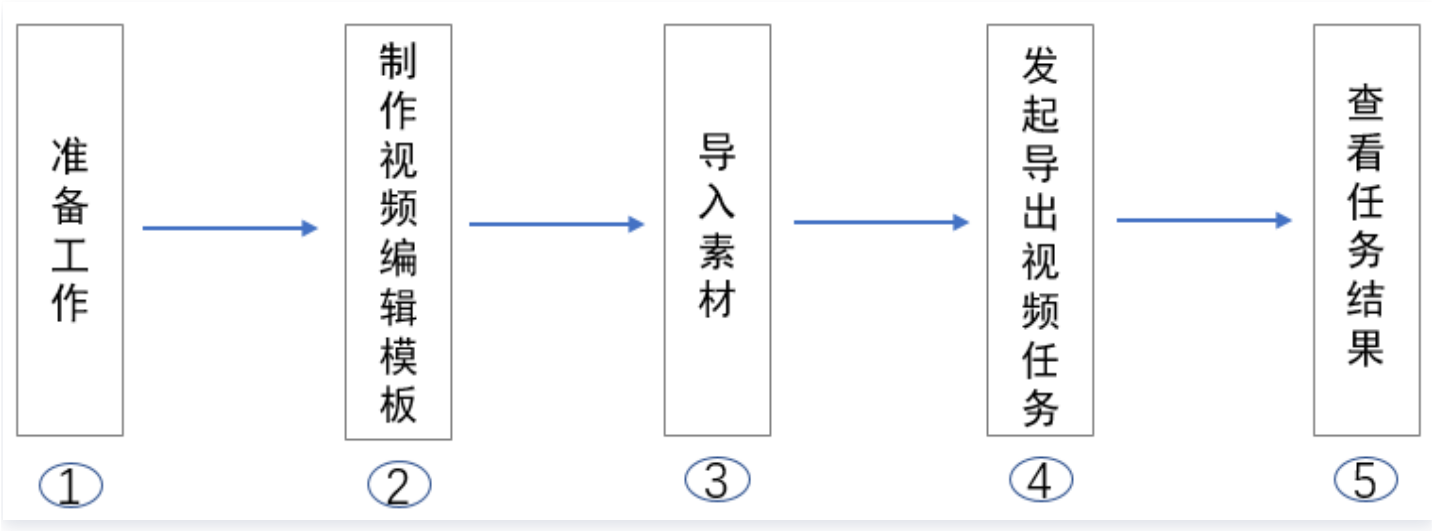
最近更新时间：2022-09-08 18:17:41

功能概述

智能创作视频剪辑支持通过视频编辑模板导出视频，只需要替换视频编辑模板中的部分媒体即可生产出效果与视频编辑模板类似的视频，可用于直接套用模板进行批量视频生产的场景。

实现步骤概要

使用视频编辑模板导出视频，只需要按如下几步操作即可完成：



步骤1：准备工作

创建智能创作平台

请参见 管理端使用指南 – 创建标准型平台 创建平台。

> ⚠ 注意
> 使用视频编辑模板功能，需要创建**标准型平台**。

获取 API 密钥

请求云 API 需要使用到开发者的 API 密钥（即 SecretId 和 SecretKey）。如果还未创建过密钥，请参见 创建密钥文档 生成新的 API 密钥；如果已创建过密钥，请参见 查看密钥文档 获取 API 密钥。

步骤2：制作视频编辑模板

参考 用户端使用指南-视频剪辑-模板制作 制件视频编辑模板。制作完成后，可以拿到**视频编辑模板 ID**，将在接口调用中使用。

> **注意**
> 后续步骤以视频编辑模板 ID：`5fc07c745d6cbe00018f1830@Public@CME` 为例，此模板为智能创作提供的公共模板，可直接使用。

步骤3：导入素材

使用视频编辑模板导出视频，需要替换模板中的部分媒体，这里需要先将媒体导入到智能创作媒资库。导入媒体的接口为 导入媒体 。为了简化操作，建议您使用 API Explorer 导入媒体。这里通过一个接口调用示例说明关键参数设置：

```
https://cme.tencentcloudapi.com/?Action=ImportMaterial
&Platform=test
&VodFileId=5285890796182734267
&Owner.Id=user_100920093
&Owner.Type=PERSON
&Name=测试媒体1
&<公共请求参数>
```

下面基于智能创作的 Java SDK 提供一个简单的示例：

```
import com.tencentcloudapi.common.Credential;
import com.tencentcloudapi.common.profile.ClientProfile;
import com.tencentcloudapi.common.profile.HttpProfile;
import com.tencentcloudapi.common.exception.TencentCloudSDKException;
import com.tencentcloudapi.cme.v20191029.CmeClient;
import com.tencentcloudapi.cme.v20191029.models.*;

public class ImportMaterial
{
    public static void main(String [] args) {
        try{
            // 需要传入准备工作中获取的 SecretId，SecretKey，此处还需注意密钥对的
保密
            Credential cred = new Credential("SecretId", "SecretKey");
            HttpProfile httpProfile = new HttpProfile();
            httpProfile.setEndpoint("cme.tencentcloudapi.com");
            ClientProfile clientProfile = new ClientProfile();
            clientProfile.setHttpProfile(httpProfile);
            CmeClient client = new CmeClient(cred, "", clientProfile);
```

```
            // 以下为关键参数部分
            ImportMaterialRequest req = new ImportMaterialRequest();
            // 平台，填准备工作中创建的平台 ID
            req.setPlatform("test");
            // 视频来源，推荐使用 VOD，先将视频上传到腾讯云点播
            req.setSourceType("VOD");
            Entity entity1 = new Entity();
            entity1.setType("PERSON");
            entity1.setId("user_100920093");
            req.setOwner(entity1);
            // 媒体名称
            req.setName("测试媒体1");
            // 腾讯云点播 FileId，使用上传到点播的视频 FileId
            req.setVodFileId("5285890796182734267");
            // 媒体文件路径，使用已经存在的媒体文件路径
            req.setClassPath("/媒体");
            // 媒体预处理参数，如果不需要做编辑预览，截取封面和雪碧图，此参数可不填
            req.setPreProcessDefinition(10L);
            // 返回的 resp 是一个 ImportMaterialResponse 的实例，与请求对象对应
            ImportMaterialResponse resp = client.ImportMaterial(req);
            // 输出 json 格式的字符串回包，任务 Id 可从此处得到

System.out.println(ImportMaterialResponse.toJsonString(resp));
        } catch (TencentCloudSDKException e) {
            System.out.println(e.toString());
        }
    }
}
```

注意

- 关于归属 Owner 参数，请参考文档 账号 部分。
- 如果替换的媒体为图片，目前可支持图片 URL 替换模板中的图片媒体，可忽略本步骤。
- 导入媒体，需要将媒体(无论是视频、音频或是图片)上传到云点播，获取到云点播 FileId。腾讯云点播提供了多种方式将视频上传到云点播，请参见 云点播-最佳实践-媒体上传 。
- 媒体上传，使用的点播子应用必须与创建智能创作平台使用的点播子应用一致，否则会返回点播文件不存在的错误。
- 发起导入媒体后，系统会返回媒体 ID，可用于替换模板中使用的媒体，后续步骤以媒体 ID：`84a9b60cd409c2203c1be93` 为例。

步骤4：导出视频

使用视频编辑模板导出视频接口为 使用视频编辑模板导出视频 。为了简化操作，建议您使用 API Explorer 发起导出视频。这里通过一个接口调用示例说明关键参数设置：

```
https://cme.tencentcloudapi.com/?Action=ExportVideoByTemplate
&Platform=test
&TemplateId=5fc07c745d6cbe00018f1830@Public@CME
&SlotReplacements.0.Id=1
&SlotReplacements.0.ReplacementType=VIDEO
&SlotReplacements.0.MediaReplacementInfo.MaterialId=5fc4ad7f69ec740001790
72345
&SlotReplacements.0.MediaReplacementInfo.StartTimeOffset=0
&Definition=10
&ExportDestination=VOD
&VODExportInfo.Name=测试视频
&VODExportInfo.ClassId=10000
&<公共请求参数>
```

下面基于智能创作的 Java SDK 提供一个简单的示例：

```
import com.tencentcloudapi.common.Credential;
import com.tencentcloudapi.common.profile.ClientProfile;
import com.tencentcloudapi.common.profile.HttpProfile;
import com.tencentcloudapi.common.exception.TencentCloudSDKException;
import com.tencentcloudapi.cme.v20191029.CmeClient;
import com.tencentcloudapi.cme.v20191029.models.*;

public class ExportVideoByTemplate
{
    public static void main(String [] args) {
        try{
            // 需要传入准备工作中获取的 SecretId，SecretKey，此处还需注意密钥对的
保密
            Credential cred = new Credential("SecretId", "SecretKey");
            HttpProfile httpProfile = new HttpProfile();
            httpProfile.setEndpoint("cme.tencentcloudapi.com");
            ClientProfile clientProfile = new ClientProfile();
            clientProfile.setHttpProfile(httpProfile);
            CmeClient client = new CmeClient(cred, "", clientProfile);
            // 以下为关键参数部分
            ExportVideoByTemplateRequest req = new
ExportVideoByTemplateRequest();
            // 平台，填准备工作中创建的平台 ID
            req.setPlatform("test");
```

```
            // 视频编辑模板 ID，填第二步制作视频编辑模板获取到的视频编辑模板 ID
            req.setTemplateId("5fc07c745d6cbe00018f1830@Public@CME");

            // 模板媒体替换信息
            SlotReplacementInfo[] slotReplacementInfos1 = new
SlotReplacementInfo[1];
            SlotReplacementInfo slotReplacementInfo1 = new
SlotReplacementInfo();
            // 卡槽 Id，此处为制作视频编辑模板设置的替换卡槽 ID，唯一确认需要替换的
媒体
            slotReplacementInfo1.setId(0L);
            // 替换类型，此处必须与视频编辑模板中原有的媒体类型一致
            slotReplacementInfo1.setReplacementType("VIDEO");
            // 替换的媒体信息
            MediaReplacementInfo mediaReplacementInfo1 = new
MediaReplacementInfo();
            // 媒体的媒体类型，目前支持云剪素材(CMEMaterialId)，如果替换的媒体为图
片，可填 'ImageUrl'
            mediaReplacementInfo1.setMediaType("CMEMaterialId");

mediaReplacementInfo1.setMaterialId("84a9b60cd409c2203c1be93");
            // 替换的媒体开始时间，对于视频和音频有效，即替换的媒体以此开始时间使用，
单位为秒
            mediaReplacementInfo1.setStartTimeOffset(0F);

slotReplacementInfo1.setMediaReplacementInfo(mediaReplacementInfo1);

            slotReplacementInfos1[0] = slotReplacementInfo1;

            req.setSlotReplacements(slotReplacementInfos1);
            // 导出模板 ID，用于指定导出的视频的参数，例如分辨率等。
            req.setDefinition(10L);
            // 视频导出目标，可是直接导出到腾讯云点播，也可以导出到云剪媒资。
            req.setExportDestination("VOD");
            VODExportInfo vODExportInfo1 = new VODExportInfo();
            vODExportInfo1.setName("模板导出视频1");
            vODExportInfo1.setClassId(5L);
            req.setVODExportInfo(vODExportInfo1);

            // 操作者 UID，必须保证该 UID 对模板及替换的素材有使用权限。
            req.setOperator("user_100920093");

            // 返回的 resp 是一个 ExportVideoByTemplateResponse 的实例，与请
求对象对应
```

```
            ExportVideoByTemplateResponse resp =
client.ExportVideoByTemplate(req);
            // 输出 json 格式的字符串回包

System.out.println(ExportVideoByTemplateResponse.toJsonString(resp));
        } catch (TencentCloudSDKException e) {
            System.out.println(e.toString());
        }
    }
}
```

注意

- 发起导出视频任务后，系统会生成一个任务 ID，后续操作步骤以 `1250000000-CME-32f87637061242508eb10d6e4ecd0cd68` 为例。
- 本例中是将视频直接导出到云点播，在云剪的媒资中无法查询到该导出的视频。如需要导出视频到云剪媒资库，请导出目标参数 `ExportDestination` 必须设置为 `CME` 并填写 CMEExportInfo 参数即可。
- 替换模板中的媒体，需要**类型一致**，即如果视频编辑模板中原有的媒体是视频，则一定要用视频替换。

步骤5：查看导出结果

发起导出任务后，需要调用接口查看任务处理结果。查看任务结果的接口为 获取任务详情 。为了简化操作，建议您使用 API Export 查看处理结果。这里通过一个接口调用示例说明关键参数设置：

```
https://cme.tencentcloudapi.com/?Action=DescribeTaskDetail
&Platform=test
&TaskId=1250000000-CME-32f87637061242508eb10d6e4ecd0cd68
&<公共请求参数>
```

下面基于智能创作的 Java SDK 提供一个简单的示例：

```
import com.tencentcloudapi.common.Credential;
import com.tencentcloudapi.common.profile.ClientProfile;
import com.tencentcloudapi.common.profile.HttpProfile;
import com.tencentcloudapi.common.exception.TencentCloudSDKException;
import com.tencentcloudapi.cme.v20191029.CmeClient;
import com.tencentcloudapi.cme.v20191029.models.*;

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

```
        try{
            // 需要传入准备工作中获取的 SecretId，SecretKey，此处还需注意密钥对的
保密
            Credential cred = new Credential("SecretId", "SecretKey");
            HttpProfile httpProfile = new HttpProfile();
            httpProfile.setEndpoint("cme.tencentcloudapi.com");
            ClientProfile clientProfile = new ClientProfile();
            clientProfile.setHttpProfile(httpProfile);
            CmeClient client = new CmeClient(cred, "", clientProfile);
            // 以下为关键参数部分
            DescribeTaskDetailRequest req = new
DescribeTaskDetailRequest();
            // 平台，填准备工作中创建的平台 ID
            req.setPlatform("test");
            // 任务 ID，填第四步导出视频生成的任务 ID
            req.setTaskId("1250000000-CME-
32f87637061242508eb10d6e4ecd0cd68");
            // 返回的resp是一个DescribeTaskDetailResponse的实例，与请求对象对应
            DescribeTaskDetailResponse resp =
client.DescribeTaskDetail(req);
            // 输出json格式的字符串回包，可获取到任务结果

System.out.println(DescribeTaskDetailResponse.toJsonString(resp));
        } catch (TencentCloudSDKException e) {
            System.out.println(e.toString());
        }
    }
}
```

注意

- 发起导出视频任务后，因系统合成任务需要异步处理，所以不能立马返回处理结果。需要轮询直至任务完成（也就是任务的状态为 `SUCCESS` ）时才能获取到导出结果。
- 导出任务成功后，通过查询任务详情可获取到导出视频的播放 URL，腾讯云点播 FileId 等信息。

如何接收事件通知

最近更新时间：2023-11-30 10:23:32

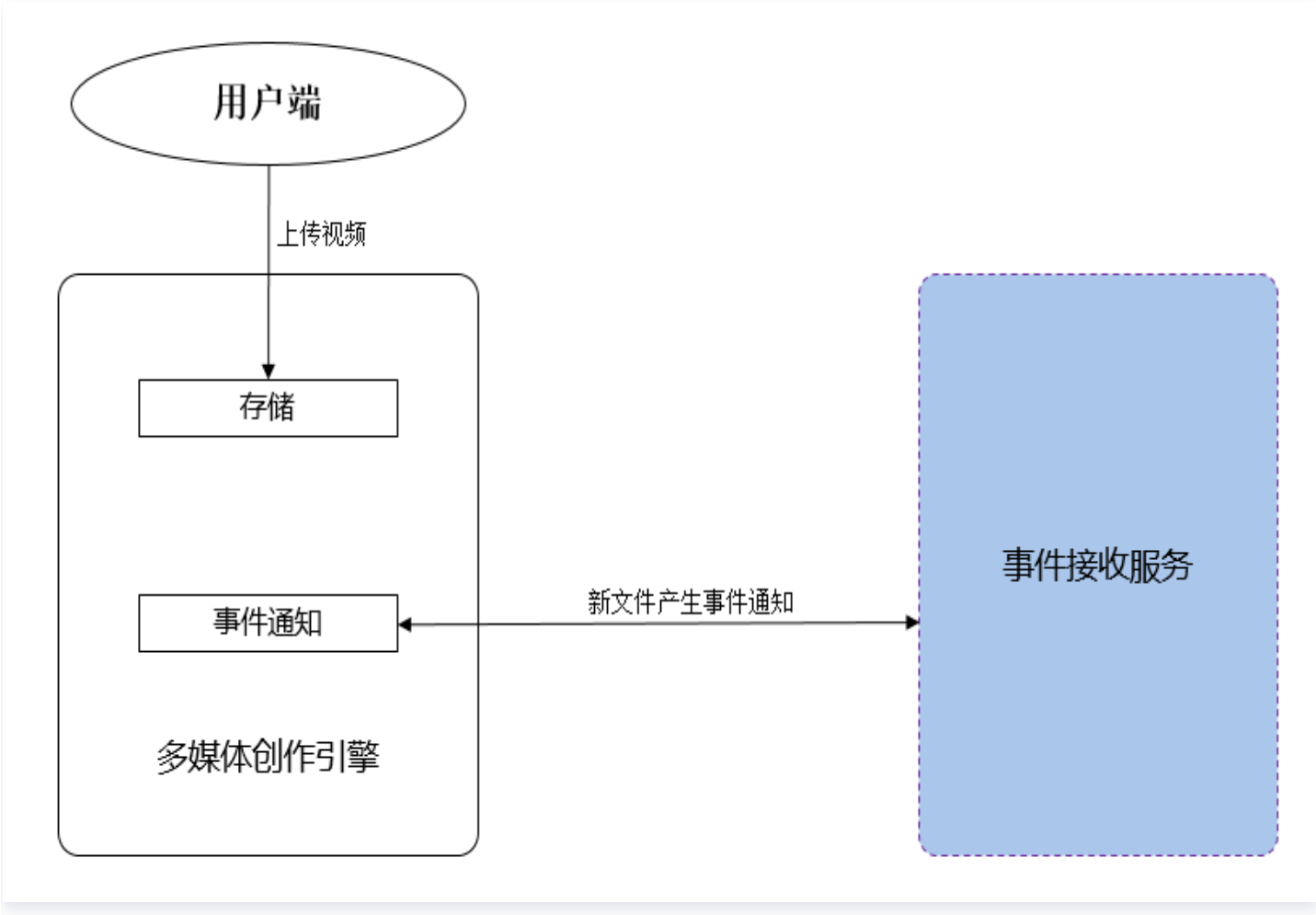
使用须知

功能介绍

本文以在用户端上传一个视频为例，向开发者展示智能创作 事件通知机制 的使用方法。

架构和流程

开发者搭建一个 HTTP 服务，接收来自智能创作的事件通知请求。本文档以该服务通过对 Storage.NewFileCreated 的处理，实现获取用户在用户端上传视频的媒体 ID 及云点播 FileId 等其它信息为例，来介绍事件通知服务的实现方法。
系统主要由三个部分组成：用户端、事件接收服务、智能创作，其中事件接收服务是本文档需要实现及部署的服务。
其架构如下图所示：



具体业务流程为：

1. 用户端上传一个视频到智能创作。
2. 智能创作后台发起 `Storage.NewFileCreated` 事件通知请求给事件接收服务。
3. 事件接收服务解析事件通知内容，将新文件的媒体 ID 及云点播 FileId 打印到日志中。

接口约定

本教程中约定，事件接收服务暴露的接口形式如下：

- 事件接收服务的 URL 地址为 `https://api.example.com/callback` 。
- 智能创作以 HTTP POST 方式发起事件通知，通过 BODY 传递事件通知内容。

准备工作

案例中所用到的素材

视频：
观看视频

创建智能创作平台

请参见 创建标准型平台 创建一个智能创作平台。

> **注意**
> 若已创建智能创作平台，请忽略此步骤。

快速部署事件通知接收服务

步骤1：部署后端服务框架

本教程基于 Node.js 以及 koa 框架进行开发（您也可以替换成您熟悉的编程语言与框架）。也可以使用腾讯云轻量应用服务器（Lighthouse）或无服务器云函数（SCF）快速部署，请参见：

- Lighthouse：搭建 Node.js 开发环境 。
- SCF：快速部署 Koa 框架 。

事件通知服务的主框架代码如下：

```
const Koa = require('koa')
const Router = require('koa-router')
const bodyparser = require('koa-bodyparser');

const app = new Koa()
const router = new Router()

// 示例请求路由代码，后续会将其用正式路由替换
router.post('/hello', (ctx, next) => {
```

```
    ctx.body = 'world'
})

app.use(bodyparser());
app.use(router.routes())
app.listen(80)
```

部署服务，并访问 `http://api.example.com/hello`，接口将返回文本 `world`。

步骤2：开发事件接收服务接口(Callback)

主要流程如下：

1. 接收请求，从 BODY 中解析出新文件的媒体 ID 及云点播 FileId。
2. 在日志中输出媒体 ID 及云点播 FileId。
3. 组装应答。

在 App 的路由部分增加如下代码，来支持 `Callback` 路径的路由：

```
const fileCreatedEventType = "Storage.NewFileCreated"

router.post('/Callback', async cxt => {
    //获取post传递的数据
    const body = cxt.request.body
    const eventtype = body.EventType
    if (eventtype === null) {
        console.log("invalid EventType")
        ctx.body = {
             Code: "Failed",
        }
        return
    }

    switch (eventtype) {
        case fileCreatedEventType:
            const event = body.StorageNewFileCreatedEvent
            console.log("MaterialId: " + event.MaterialId)
            console.log("FileId: " + event.FileId)
        default:
            console.log("EventType: " + eventtype)
    }

    ctx.body = {
         Code: "Success",
    }
```

```
});
```

> **注意**
> 智能创作只判断 HTTP 返回码，如果是 200 即认为成功，否则会重试回调三次。

步骤3：配置事件通知地址

配置事件通知地址请参阅 开通事件回调 提工单，由智能创作工作人员配置。

步骤4：测试事件接收服务

按照 网盘-上传综述 指引的方式上传视频。等待视频上传成功后，登录到事件通知服务，应该可以看到日志中打印出媒体 Id 及云点播 FileId。

> **注意**
> 目前事件通知是以智能创作 平台 为单位配置，上传视频所使用的用户端平台必须与事件通知配置的平台一致，否则无法接收到事件通知。

开发指南
基本概念与模型
平台

最近更新时间：2023-02-17 09:59:13

平台（Platform）包含了一组构建视频制作应用的完整资源集合，包括 账号 、团队 、媒资 、项目 、任务 等。不同平台之间的资源完全隔离。每个平台必须绑定腾讯 云点播（VOD）产品的一个 子应用（或者主应用），并将 媒体存储与处理 托管给后者。

主要特性

- **资源隔离：平台与平台之间的智能创作资源相互隔离，即资源只能归属于一个平台，只能被该平台的用户使用。**
- **独立数据统计：为每一个平台生成单独的数据统计信息，例如视频编辑总时长、用户登录 DAU、媒资总使用量等。**

使用限制

- 不支持修改平台的英文名称。
- 暂不支持删除平台。

应用场景

多部门或多业务隔离

某企业基于腾讯云开发自有产品，其中 A、B 部门都需要使用智能创作的能力。这两个业务需要相互隔离，但出于财务管理的考虑，该企业无法为 A 和 B 部门分别开通一个独立的腾讯云账号。这时就可以使用智能创作的平台特性，为 A 和 B 部门各分配一个平台。

区分正式环境和测试环境

开发者想对某些智能创作的特性进行测试，又担心会影响线上运营。开发者可以为正式环境和测试环境各开通一个平台，新特性先在测试环境进行验证，确认无误后再变更线上环境。

操作指南

控制台相关操作

目前仅支持在控制台上对平台进行管理，具体请参见 创建平台 。

服务端 API

具体请参见 获取平台列表 。

项目

最近更新时间：2022-09-08 18:18:01

项目是承载具体媒体制作工作的载体，例如视频剪辑项目。项目通常是由一组媒体资源作为输入，经过一定的处理之后，再输出目标媒体内容。

> **说明**
> 目前支持 Web 在线视频编辑项目，详情请参见 Web 在线视频编辑集成 。

操作指南

操作	服务端 API	客户端 API
创建项目	CreateProject	N/A
获取项目列表	DescribeProjects	/Project/DescribeProjects
修改项目	ModifyProject	/Project/ModifyProject
删除项目	DeleteProject	/Project/DeleteProject

任务

最近更新时间：2022-09-08 18:18:07

任务是执行耗时较长的**服务端**媒体处理工作，例如媒体转码、媒体合成等。因其执行时间较长，故而无法采用一般的「请求 - 应答」模式完成处理。对于此类服务，一般的流程是：

1. 创建任务，返回任务 ID。
2. 开发者可以根据任务 ID 查询任务状态。

> **注意**
> 目前 智能创作 尚未提供任务状态变更通知能力。

操作指南

操作	服务端 API
获取任务详情	DescribeTaskDetail
获取任务列表	DescribeTasks

账号

最近更新时间：2023-05-22 18:03:38

智能创作的账号是一个应用中用户的唯一标识。智能创作提供用户级别的资源管理以及操作管控体系，因此也需要通过账号来对平台中的用户进行区分。智能创作的账号体系具备如下特点：

特点	说明
单平台内唯一	用户标识在平台内唯一，因此不建议开发者系统将多个用户映射到同一个用户标识内，否则无法区分实际的身份。而两个平台可以存在相同的用户标识，他们分别属于不同的平台，互不影响。
完全自定义	开发者可以自由定义用户的标识，既可以复用 App 已有的账号体系，也可以根据需要独立生成。
无需事先同步	开发者无需实现将已有 App 中的账号同步至智能创作，便可直接使用账号 ID。例如，在调用服务端 API 创建项目时，可以直接指定任意用户作为项目的所有者，而无需关心该用户 ID 是否在系统中使用过。在添加团队成员时，智能创作 也不会对目标成员 ID 做出任何限制。

团队

最近更新时间：2022-06-09 15:11:42

团队（Team）是由多个用户组成的实体。团队中的用户存在如下几种角色：

- **团队所有者（Owner）：** 对团队具有高级别的操作权限，包括增删团队管理员、任命管理员、增删团队成员等。一个团队只允许有一个所有者。
- **团队管理员（Admin）：** 辅助团队所有者进行团队管理的用户，具有增删成员等权限。一个团队中允许存在多个管理员。
- **团队成员（Member）：** 可获取团队信息，对团队资源具备受限的操作权限。

典型应用场景

团队主要是配合 媒资 及 授权体系 来使用，具体表现在：

- 每个团队可以拥有独立的媒体资源，不同的团队角色对团队媒资具有不同级别的访问权限。具体参见 团队媒资 。
- 可以将团队的部分或全部资源授权给其他团队或者非团队成员，使得后者能够以特定的权限访问本团队的资源。具体参见 媒资授权 。
- 某些场景下，App 需要搭建面向所有用户的公共素材库。可以创建一个团队专门用来管理公共素材，将 App 的运营人员添加到团队中。然后将该团队的所有素材授权给 App 内的所有用户。

操作指南

操作	服务端 API	客户端 API
创建团队	CreateTeam	N/A
获取团队信息	DescribeTeams	/Team/DescribeTeams
获取用户所加入的团队	DescribeJoinTeams	/Team/DescribeJoinTeams
修改团队信息	ModifyTeam	N/A
删除团队	DeleteTeam	N/A
添加团队成员	AddTeamMember	N/A
修改团队成员	ModifyTeamMember	N/A
删除团队成员	DeleteTeamMembers	N/A
获取团队成员信息	DescribeTeamMembers	N/A

媒资管理
媒资管理综述

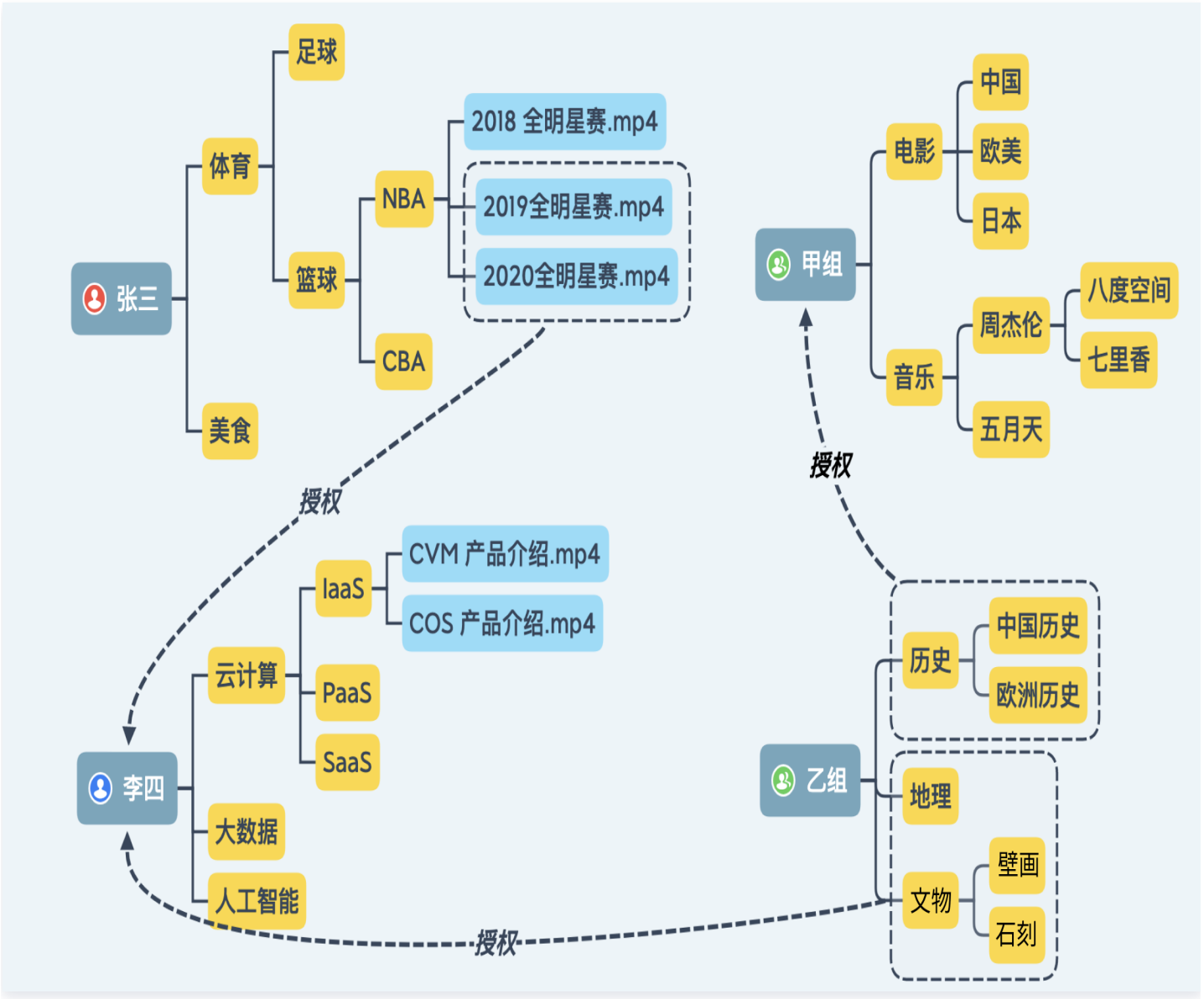
最近更新时间：2023-05-22 18:03:38

智能创作提供了一组具备完整功能的媒体资源管理能力，开发者可以基于这套能力实现个人及团队的媒体资源管理。具体能力包括：

能力项	能力介绍
多种媒体格式	支持多种类型的媒体，包括视频、音频、图片、模板等。
多种导入来源	支持将腾讯云点播（VOD）、对象存储（COS）中的视频、音频、图片等文件托管到媒资库。
树形分类管理	支持为每个用户、团队维护独立的媒资分类树。每个用户或团队的分类体系完全独立，并可以借助于媒体授权能力实现跨用户（或团队）媒资访问。
链接	支持创建到特定媒体或者分类的链接，类似于 Windows 的快捷方式或者 Unix 文件系统的软连接。
媒体搜索	支持依照媒体的名称、类型、标签等信息进行模糊搜索。
媒体标签	支持为每个媒体自由打上标签；也支持全局预设标签体系，并为每个媒体打上全局预设标签。
媒体授权	用户可以将自身特定分类或媒体授权给其他用户（或团队），从而使后者具备对特定媒资的访问权限。团队所有者或管理员亦可将团队资源授权给其他用户（或团队）。
共享空间	用户可在共享空间内看到其被哪些用户（或团队）授予了哪些资源的访问权限。
支持与项目打通	在线视频编辑项目、导播、转推项目均支持以媒资库中的素材作为输入，项目的输出亦可落地到媒资库中。

典型案例

下图为一个典型的媒资案例。



案例媒资	案例讲解
个人媒资	两个用户，张三和李四，分别有自己独立的媒资库。
团队媒资	两个团队，甲组和乙组，也分别有各自团队专属的媒资库。
媒体授权	张三将 `2019全明星赛.mp4` 、 `2020全明星赛.mp4` 两个媒体授权给李四，因此李四具备对此二媒体的访问权限。 乙组将 `地理` 、 `文物` 两个分类授权给李四，因此李四也具备对此二分类的访问权限。 乙组将 `历史` 分类授权给甲组，因此甲组 所有成员 均具备该分类的访问权限。
共享空间	李四得到张三、乙组的授权，因此其共享空间中包括张三、乙组。其中来自张三的共享根为 `2019全明星赛.mp4` ， `2020全明星赛.mp4` 两个媒体 ，来自乙组的共享根为 `地理` 、

	文物 两个分类。 甲组得到了乙组的授权，因此其 所有成员 的共享空间中都包括乙组，共享根为 历史 分类。

媒体存储与处理

最近更新时间：2023-04-24 10:59:26

媒体类型

智能创作媒资库支持如视频、音频、图片三种基础媒体类型，同时还支持链接等特殊媒体类型。具体参见 媒体类型 。

- **基础媒体类型**会占据存储空间。智能创作不提供存储服务，仅支持将已经存在于特定存储平台的基础类型媒体**导入**到媒资库中，具体参见 存储空间 以及 媒体导入 。
- **特殊媒体类型**无存储空间消耗，可以直接创建，无需从已有存储服务中导入。

存储空间

智能创作媒资库不提供媒体存储能力，也不会产生任何与媒体存储相关的计费。因此，对于视频、音频、图片等基础媒体类型，添加媒体的唯一方式是将已经存储在特定服务中的媒体导入到媒资库中。
智能创作支持的导入来源包括：

- 位于腾讯云点播（VOD）中的媒体文件。
- 位于腾讯云对象存储（COS）中的媒体文件。
- 位于开发者自有存储中的媒体文件。

为保证智能创作媒资服务的基本可用性，在创建平台时，需要绑定同 APPID 下 VOD 产品的某一子应用（或主应用）作为媒资库的默认存储载体。具体绑定方法参见 创建平台 。

媒体导入

智能创作支持的媒资导入方式包括：

- 将媒体文件导入到用户或团队的媒资库。
- 直接将媒体文件导入到项目中，此时媒体本身归属于项目，而不会出现在项目所有者的媒资库中。

对于被导入的媒体：

- 被导入的媒体是对当前平台所绑定的 VOD 子应用（或主应用）中某个 FileID 的引用，不会产生具体的物理复制。
- 智能创作媒资库中，对某一媒体文件的复制，仅仅是增加对 VOD 子应用（或主应用）中目标 FileID 的引用，不会产生物理复制。
- 在智能创作中将被导入的媒体删除，被删除的仅仅是对 VOD 中目标 FileID 的引用，不会连带删除 VOD 中被绑定的 FileID。
- 如果智能创作某一媒体所绑定 VOD FileID 被删除，该删除操作将同步到智能创作，智能创作会静默将其所关联的媒体全部删除，以避免无效引用的出现。

导入腾讯云 VOD 中的 FileID

仅支持将平台所绑定的 VOD 子应用（或主应用）中的 FileID 导入媒资库中。

操作	服务端 API
向媒资库中导入媒体	ImportMaterial
在项目中导入媒体	ImportMediaToProject

注意

为避免最终客户端用户将 VOD 中任意 FileID 导入媒资库，产生不必要的越权问题，故而不提供客户端媒资导入接口。

导入其他存储服务中的媒体

为保证整个产品的体验流畅度，降低接入难度，如需要对非腾讯云 VOD 或 COS 的资源引入媒资库并进行编辑，强烈建议开发者将文件迁移到腾讯云 VOD。将其他存储服务中的媒体迁移到 VOD 参见：

- VOD 服务端上传。
- VOD 离线拉取 API（PullUpload）。
- VOD 源站迁移工具。

说明

如媒资迁移难度较大，或因合规等要求无法迁移，请联系商务进行特殊支持。

媒体处理

对基础媒体的处理

智能创作支持对已经导入媒资库的视频、音频、图片等媒体文件进行处理，例如转码、语音识别等。智能创作对基础类型媒体的处理能力全部依赖 VOD 产品，也不会产生媒体处理相关的费用。

使用示例：

使用 Web 在线编辑项目对一个 FLV 视频进行编辑。为保证前端可预览，需要对 FLV 文件进行适当转码。此时智能创作将直接调用 VOD 相关接口对该媒体所绑定的 VOD FileID 进行转码，所产生的转码费用会计入 VOD 产品中，智能创作产品不会因本次转码产生计费。

基础媒资管理

最近更新时间：2024-12-11 16:09:53

智能创作支持为每个用户维护独立的个人媒体库，也支持为每个团队维护独立的公共媒体库。每个用户/团队的媒资库可以看做是独立的，类似于网盘的云上文件系统。

个人媒资

归属于某个用户的媒资库，在不进行授权的情况下，个人媒资库只有用户本人可访问。
用户对其个人媒资具备最高操作权限，包括：

操作类型	操作权限
媒资操作	导入、读取、使用（添加到项目）、修改、删除媒资。
分类操作	创建、读取、修改、删除分类。

团队媒资

归属于某个 团队 的媒资库。默认情况下，团队中的不同角色对团队媒资具有不同级别的操作权限：

角色类型	角色权限
团队所有者（Owner）	对团队媒资具备最高操作权限，包括媒资导入、读取、使用、修改、删除等，分类创建、读取、修改、删除等。
团队管理员（Admin）	与团队所有者权限相同（仅在媒资管理方面）。
团队成员（Member）	仅具备团队媒资以及分类的读取、使用权限，不具备增加、删除、修改能力。

分类管理

媒资库支持为每个用户、团队维护独立的分类树，类似于文件系统的目录。

- 每个用户/团队都有独立的根分类。
- 每个分类下可以继续添加子分类，最多支持 10 级子分类嵌套。
- 同一分类的各个子分类不允许重名。
- 任意媒资文件**能且只能**归属于其所有者的**某一个**分类之下。

移动分类

分类管理没有提供修改分类的接口，开发者可以通过移动分类来实现分类的重命名及修改父分类：

移动分类	分类示例
重命名	将分类 /素材/视频/NBA 移动到 /素材/视频/篮球，若 /素材/视频/篮球 分类不存在，则 NBA 分类被重命名为 篮球，分类下的素材归属到 篮球 分类。
修改父分类	将分类 /素材/视频/NBA 移动到 /素材/视频/篮球，若 /素材/视频/篮球 分类已经存在，则 NBA 分类会被移动到 篮球 分类下，最终分类下的素材及子分类归属到新分类 /素材/视频/篮球/NBA 下。

说明

移动分类操作的语义与 UNIX Shell 脚本中 `mv` 命令的语义一致。

删除分类

只允许删除空分类。对于待删除的分类，要求：

- 没有素材归属在目标分类下。
- 目标分类没有子分类。

预置分类

智能创作为每个用户和团队都预置了如下一组目录，如下图所示。可以直接将媒体导入或移动到该目录下，而无需提前创建。

```
/
├── 媒资
├── 素材
│   ├── 视频
│   ├── 音频
│   ├── 图片
├── 效果素材
│   ├── 文字
│   ├── 图文
│   ├── 字幕
│   ├── 贴图
│   ├── 转场
│   ├── 背景音乐
│   ├── 滤镜
│   ├── 音效
│   ├── 动效
│   ├── 边框
│   ├── 风格
├── 模板
├── 成片
```

```
├── 收藏
├── 发布通道
```

> **注意**
> 预置目录无法修改与删除。

分类相关操作

操作	服务端 API	客户端 API
创建分类	CreateClass	/Material/CreateClass
浏览分类	ListMedia	/Material/ListMedia
移动分类	MoveClass	/Material/ModifyClass
删除分类	DeleteClass	/Material/DeleteClass

媒体管理

媒体类型

媒资库支持如下两种类型：

媒体类型	媒体说明	媒体格式
基础媒体类型	基础媒体类型包括如下几种。每种类型支持的格式与 VOD 产品相同。参见 VOD 媒体类型。	视频、音频、图片。
特殊媒体类型	到某个用户/团队的某个媒体或者分类的引用，参见 链接。	链接。

媒体搜索

媒资库支持如下几种类型的媒体搜索能力：

- 依照媒资 ID 精确获取媒体信息。
- 依照分类树遍历媒资文件。
- 依照名称、简介、标签、创建时间等信息模糊搜索媒体文件。

> **注意：**
> 用户除了可以访问自身以及其所加入团队的媒资库，还可以访问其他用户或其尚未加入的团队的媒体资源，但前提是得到必要的授权。具体参见 媒体授权 。

媒体管理相关操作

操作	服务端 API	客户端 API
获取媒体详情	DescribeMaterials	/Material/DescribeMaterials
修改媒体信息	ModifyMaterial	/Material/ModifyMaterial
搜索媒体	SearchMaterial	/Material/SearchMaterial
浏览某一分类下的媒体资源	ListMedia	/Material/ListMedia
平铺浏览某一分类下的媒体资源	FlattenListMedia	N/A

链接

链接是到某个用户/团队的某个媒体或者分类的引用，类似于 Windows 的快捷方式或者 Unix 文件系统的软连接。其目的在于简化访问路径。

- 链接本身与视频音频等基础媒体一样，是媒资的一种，其必须归属于某个用户/团队的分类树之下。
- 链接可以指向任意用户/团队的分类或媒体，而不仅限于链接本身所归属的分类树内的分类或媒体。
- 不允许创建指向链接的链接。

链接类型

链接类型可分为以下两种：

链接类型	具体示例
指向媒体的链接	Bob 的媒资库中存在一个音频媒体，其路径为 /音乐/华语歌手/七里香。 Alice 在自己媒资库的 /music 分类下创建指向 Bob 媒资库中 /音乐/华语歌手/七里香 的链接，其名称为 qilixiang。 如果授权得当，则 Bob 可以通过自身媒资库中的 /music/qilixiang 链接访问 Bob 媒资库中的 /音乐/华语歌手/七里香 音频文件。
指向分类的链接	Team1 下存在 /体育/篮球 这一分类。 Alice 在自己的 /sport 分类下创建指向 Team1 /体育/篮球 的链接，链接的名称为 basketball。 如果授权得当，则 Alice 可以通过自身分类树下的 /sport/basketball 链接访问 Team1 的 /体育/篮球 分类。

链接相关操作

操作	服务端 API
创建链接	CreateLink

标签

与分类相比，标签是一种更加灵活的媒体资源管理方式，具体表现在：

- 同一个媒体文件可以被打上多个标签。
- 多个媒体文件可以拥有相同的标签。

智能创作媒资库提供自有标签和预置标签两种形式。自由标签最为灵活，预置标签更加统一。单个媒体文件可以同时被打上自由标签以及预置标签，两者互不干涉。

自由标签

媒资的所有者（或具备修改权限的用户）可以将任意文本作为标签打在该媒资上。单个媒体文件的标签不会重复。

预置标签

预置标签的主要目的是规范整个平台内的标签体系，不允许用户随意设置标签内容，从而将用户可以设置的标签限定在某一范围内。
预置标签具有以下特点：

- 与自由标签相比，预置标签是由 `标签 ID + 标签名称` 组成。
- 在整个平台内，每个预置标签都具备唯一 ID，多个标签的名称可以相同。
- 标签可以具备层级结构，最多支持三级。

应用场景示例：

- 某电竞场景，需要为比赛视频打标签，但用户打标签过于随意，不便于全局管理。例如对于和平精英这一款游戏的相关内容，用户可能设置的标签多种多样，例如 `和平精英` 、 `吃鸡` 、 `绝地求生` 、 `PUBG` 、 `PUBG-Mobile` 等。
- 针对上述情况，可以使用预置标签。App 后台创建全局标签，名称为 `和平精英` ，其父标签为 `比赛录像` 。
- 不允许用户设置自由标签，从而限制用户只能为视频打上名为 `和平精英` 的预置标签。
- App 后台管理员可以修改全局标签的名称，例如将 `和平精英` 这个预置标签的名称改为 `吃鸡` 。此时，所有被打上该标签的媒体，外显标签名称都得到了更新。

媒资授权与共享空间

最近更新时间：2024-08-14 18:37:41

媒资授权

默认情况下，用户只能访问自身或者其所加入团队的媒资库，而不能访问其他用户或者其未加入团队的媒资库。要突破这一限制，就必须引入授权体系。

授权操作由如下元素构成：

元素	说明
授权者（Authorizer）	用户可以将其自身媒资库中的分类或者媒体授权给其他用户或团队，此时授权者为该用户。团队的 Owner 或者 Admin 可以将团队资源授权给其他用户或团队，此时授权者为该团队（注意此时授权者不等于操作者）。
被授权者（Authorizee）	被授予权限的用户或者团队。当被授权者为团队时，相当于对该团队的所有成员进行授权（如某一成员退出团队则不再享受授权）。
资源（Resource）	授权者分类树下的某一个媒体，或者某一个分类。
权限（Permission）	系统预设四种权限，详情请参见 预设权限；开发者亦可自定义权限。

> **注意：**
> 授权不具备传递性，即：用户不能将其他用户授权给自己的媒体进一步授权给第三方。

预设权限

以下是预设权限参数解读：

参数	说明
R（可读）	被授权者可以读取被授权媒体的基本信息，浏览被授权分类，但不能将媒体添加到自己的媒资库，也不能使用该媒体（例如将其添加到在线编辑项目中）。
X（可用）	被授权者可以读取被授权媒体的基本信息，浏览被授权分类，可以使用该媒体（例如将其添加到在线编辑项目中），但不能将其复制到自己的媒资库中。当授权被取消之后，被授权者将无法再使用该素材。
C（可复制）	被授权者可以将该素材复制到自身媒资库中，从而创建一份该素材的副本，并以任意方式操作该副本。当授权被取消之后，被授权者对素材副本的操作权限不受影响。

W（可修改）	被授权者可以修改、删除得到授权的媒体或分类。

操作指南

操作	服务端 API
对资源进行授权	GrantResourceAuthorization
获取资源授权列表	DescribeResourceAuthorization
撤销授权	RevokeResourceAuthorization

共享空间

当用户被授予某些资源的访问权限时，其需要一个统一的入口来列出这些资源。共享空间（SharedSpace）正是为了解决这一问题而设计的，它的目的是让用户可以获知哪些用户或团队对该用户授予了哪些资源。用户在其共享空间中可以看到：

- 哪些用户或团队对**该用户**存在资源授权。
- 哪些用户或团队对**该用户所在的团队**存在资源授权。

每一个授权操作都将产生一个**共享根**。如果被授权资源为单个媒体，则共享根为媒体文件本身。如果被授权资源是一个分类，则共享根为该分类，被授权者可以遍历该分类及其子分类下的所有媒体。

操作指南

操作	服务端 API
获取共享空间	DescribeSharedSpace

播放器与合成协议
播放器

最近更新时间：2023-11-15 20:48:01

介绍

Web Iframe 接入 可以不用关心播放器细节，如果业务需要自行实现剪辑交互，可以使用我们的播放器进行轨道数据播放。

- `window.YJPlayer` 是用于预览 视频合成协议数据 的 Web 播放器构造函数。
- `window.YJPlayer.Helper` 提供辅助函数协助组装 合成协议数据 。

快速开始

```
<html>
  <head>
    <!--cdn形式引入我们的sdk--->

    <!-- window.YJPlayer -->
    <script src="https://vs-cdn.tencent-cloud.com/sdk/yj-
player@2.0.3.js"></script>
  </head>
  <body>
    <div id="container"></div>
  </body>
  <script>
    /**
     *    @auth 登录方法，保证可以调用CME，WebAPI。
     *    @param sign {{string}} 签名串，请参见 生成客户端访问签名（Signature）
https://cloud.tencent.com/document/product/1156/50898。
     *    注意：这里只能使用 action = Login 的签名串
     **/

    YJPlayer.Helper.CMEUtils.auth({
      sign: "your_sign",
    })
      .then(() => {
        console.log("登录成功");
        /**
         * 创建一个轨道数据
         */
        const videoTrack = YJPlayer.Helper.Track.create({
```

```
          type: "video",
        });

        console.log(videoTrack);

        /***
         * 从媒资库获取一个视频类型素材。
         * 使用append助手方法往轨道添加元素。
         */

YJPlayer.Helper.CMEUtils.createTrackItem(["video_asset_id"]).then(
          (videoItem) => {
            YJPlayer.Helper.Track.append(videoTrack, videoItem);

            videoItem.duration = 8000; //调整播放时长为8秒
            const data = [videoTrack];
            let player = new YJPlayer({
              mode: "preview",
              container: document.getElementById("container"),
              aspectRatio: "16:9",
              data: { tracks: data },
            });
            player.play();
          }
        );
      })
      .catch((err) => {
        console.error("登录异常");
      });
  </script>
</html>
```

播放器

构造函数

参数	描述	类型	是否必填
mode	播放器模式	edit | preview	是
aspectRatio	播放器舞台纵横比	默认值：`16:9` | `9:16`，对应的分辨率为 `960 * 540` 以及 `540 * 960`	是

container	`yj-player` 实例所挂载的 `dom` 容器	HTMLElement	是
data	剪辑协议数据	{ tracks: [] }	否

```
const player = new YJPlayer({
  mode: "preiew",
  container: document.getElementById("container"),
  aspectRatio: "16:9",
  data: { tracks: [] },
});
```

属性

所有的都是只读属性。

参数	描述	类型
width	播放器舞台区域宽度	number
height	播放器舞台区域高度	number
ended	播放器播放结束态标志	boolean
aspectRatio	播放器舞台纵横比	`16:9` | `9:16`
currentTime	当前播放画面帧的时间戳，单位为毫秒	number
totalTime	可播放画面帧的总时长，单位为毫秒	number

方法

方法名	参数	说明
play	无	播放器启动。
setResolution	无	设置播放器舞台比例
pause	无	播放器暂停
seek	(frame: number)	播放器跳到指定帧
stop	无	播放器结束，并 seek 到最后一帧
toStart	无	播放器跳到片头

toEnd	无	播放器跳到片尾
forward	(seconds: number)	视频快进指定秒数
backward	(seconds: number)	视频后退指定秒数
clear	无	重置播放器数据
updateData	(fustionData: { tracks: [] })	更新轨道协议数据，详见轨道协议部分

```
/**
* 初次自行设置比例和舞台像素，使用时，传入指定比例
* 注意，后台导出的舞台预览仅支持以下比例尺寸.
*{
  '16:9': {
    width: 960,
    height: 540,
  },
  '9:16': {
    width: 540,
    height: 960,
  },
  '1:1': {
    width: 540,
    height: 540,
  },
  '2:1': {
    width: 1080,
    height: 540,
  },
  '6:7': {
    width: 540,
    height: 630,
  },
  '7:6': {
    width: 630,
    height: 540,
  },
  '3:4': {
    width: 540,
    height: 720,
  },
  '4:3': {
    width: 720,
```

```
    height: 540,
  },
}
*
*/

YJPlayer.setResolution("16:9", {
  width: 960,
  height: 540,
});
const player = new YJPlayer({
  aspectRatio: "16:9",
});
```

事件

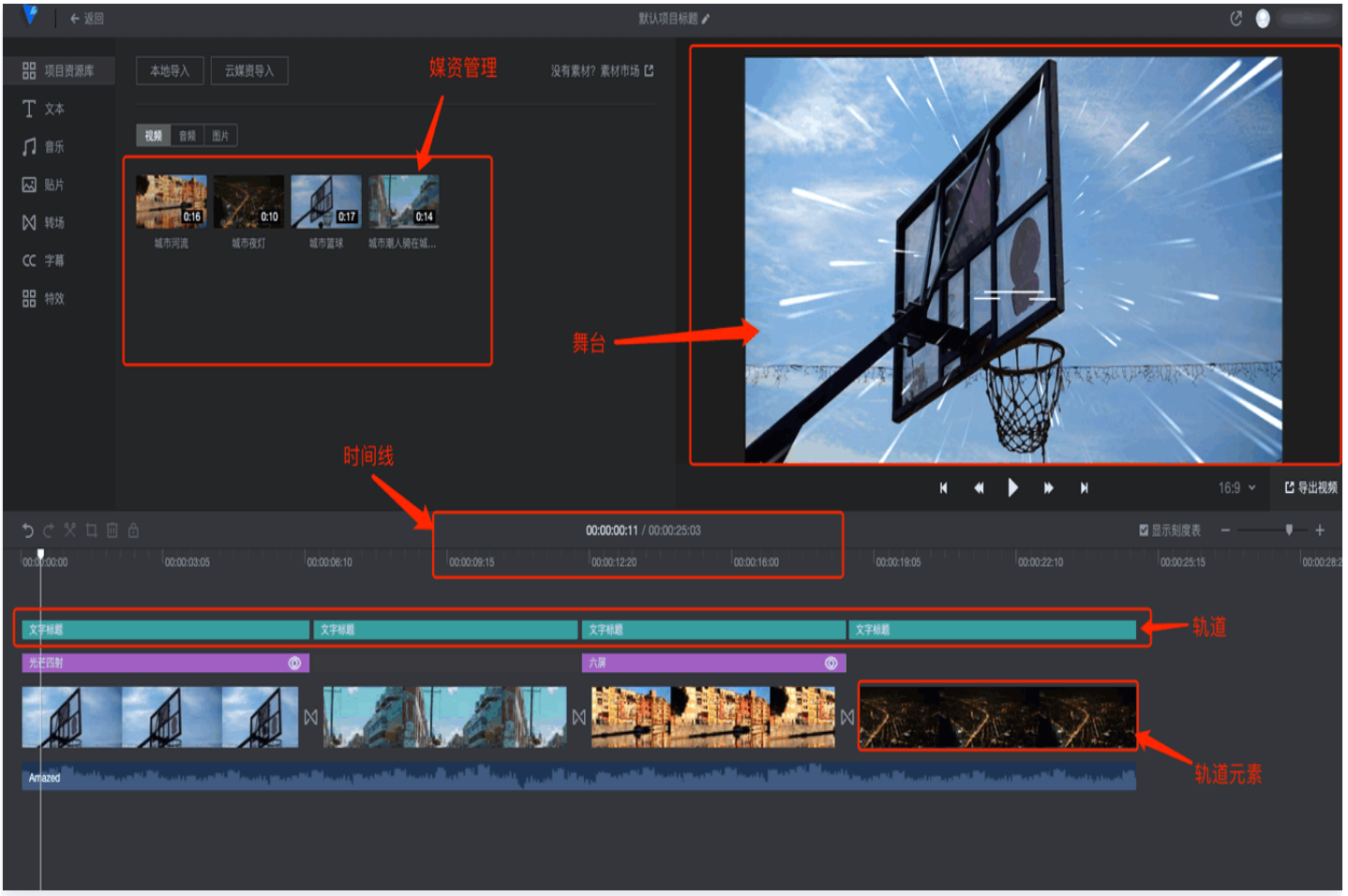
事件名	参数	说明
stateChange	(state: playing | pause)	播放器状态改变的事件回调
timeUpdate	(frame: number)	播放器时间更新的事件回调
player:playing	无	播放器开始播放时的事件回调
player:pause	无	播放器暂停播放时的事件回调
player:ended	无	播放器播放结束时的事件回调
player:error	无	播放器加载失败时的事件回调

```
// 所有事件可直接通过应用实例本身进行监听
// 播放状态切换
player.on("stateChange", (res) => {
  // do something...
});
```

合成协议

最近更新时间：2025-06-11 11:34:21

合成协议可视化编辑在页面上的体现为红框所示部分，进一步体验请登录 腾讯制作云 。



> **注意：**
> 合成协议用于描述导出合成视频原始资源的组织形式，**如果输入协议异常，会引起导出失败等预期外行为。**

示例合成协议


```
[
  {
    "id": "9e841f80-371f-46c7-ae15-70d2a3d7846f",
    "items": [
      {
        "asset_id": "3819215537437086O3@Public@CME",
        "contents": {
          "text": "猫头鹰"
        },
        "duration": 3000,
```

```
        "id": "2f6897a3-6491-4ac5-8d20-83a32e3ff2fe",
        "start_time": 0,
        "text_style_id": "38192155374370 8603@Public@CME",
        "type": "subtitle"
      },
      {
        "asset_id": "38192155374370 8603@Public@CME",
        "contents": {
          "text": "松鼠"
        },
        "duration": 3000,
        "id": "4a2ecfb5-498d-4218-9ba6-917fe70353d0",
        "is_new": true,
        "start_time": 3000,
        "text_style_id": "38192155374370 8603@Public@CME",
        "type": "subtitle"
      }
    ],
    "styles": [
      {
        "id": "38192155374370 8603@Public@CME",
        "text_style": {
          "align": "center",
          "bold": 0,
          "font": "SimHei",
          "font_color": "#ffffff",
          "font_size": 60,
          "height": 110,
          "italic": 0
        }
      }
    ],
    "type": "subtitle"
  },
  {
    "id": "738157fa-8ae5-4ae0-8ead-b5e8d4ef70d7",
    "items": [
      {
        "asset_id": "5fc609c6b41f1d0001beab46@Public@CME",
        "duration": 3000,
        "filter_asset_id": "",
        "height": 541,
        "id": "83069edb-99c3-4a29-8446-6d4feba86832",
        "operations": [
```

```
        {
          "params": {
            "angle": 0
          },
          "type": "image_rotate"
        }
      ],
      "position": {
        "x": 480,
        "y": 270
      },
      "section": {
        "from": 0,
        "to": 3000
      },
      "start_time": 0,
      "type": "image",
      "width": 773
    },
    {
      "asset_id": "5fc609ceb41f1d0001beab95@Public@CME",
      "duration": 3000,
      "filter_asset_id": "",
      "height": 540,
      "id": "3ae2a8d4-d326-4367-9712-922580506a51",
      "operations": [
        {
          "params": {
            "angle": 0
          },
          "type": "image_rotate"
        }
      ],
      "position": {
        "x": 480,
        "y": 270
      },
      "section": {
        "from": 0,
        "to": 3000
      },
      "start_time": 3000,
      "type": "image",
      "width": 811
```

```
      }
    ],
    "type": "video"
  }
]
```

协议主要概念

- **舞台：**合成视频的可视区域，可视区域包含两个要素，宽高比以及分辨率，目前支持两种比例，`16:9` 与 `9:16`，`SDK` 默认舞台分辨率为 960 * 540。

  > **注意**
  > 导出时会使用原素材重新合成，所以导出分辨率和预览分辨率无关，可导出分辨率有三种，具体请参见 导出详情 。

- **剪辑时间线：**所有在舞台上的资源以及相关内容，都在同一时间线上播放。此时间线与资源本身无关。同一轨道上的时间线不可重叠。
- **轨道(Track)：**决定舞台展示层级的容器，顺序字段用数字表示，从小到大，表示从顶层到底层，顶层值为 0，参见 轨道 。
- **轨道元素(TrackItem)：**决定展示内容的容器，只能承载媒资，参见 轨道元素 。
- **媒资管理：**整合到制作云系统的基本媒体资源，如何导入媒资，具体请参见 导入素材库 。

基本概念

```
// 轨道元素
const trackItem = {
  id: "45ea83ad-a770-4e4a-a9b2-93cdefa4d369",
  start_time: 1440, // 剪辑时间线上的开始时间
  duration: 7880, // 剪辑时间线上的持续时间
  type: "video", // 素材类型
  section: {
    // 素材时间戳截取，素材有自身时间线时，才有此属性
    from: 0,
    to: 7880,
  },
  asset_id: "5f6464c6ccbc8d0001fc308b", // 素材 id
  width: 540,
  height: 304,
  position: {
    x: 270,
    y: 480,
  },
```

```
  operations: [
    // 对该元素的操作处理，这里是指的旋转，具体请参见 多媒体处理部分
    {
      type: "image_rotate",
      params: {
        angle: 0,
      },
    },
  ],
};

// 轨道
const track = {
  id: "beb45741-c9f0-45b7-a06c-d33b579208b8",
  type: "video",
  order: 0, // 轨道层级
  items: [trackItem], // 轨道上的轨道元素
};

// 后台导出协议：https://cloud.tencent.com/document/product/1156/44159
const TrackData = JSON.stringify([track]);
```

说明

注意上述轨道协议用于后台导出，不能直接通过 `yjplayer.updateData({ tracks })` 更新到播放器，前端通过合成协议助手拼装轨道数据。

轨道

轨道一共有六种类型：`title`、`frame`、`subtitle`、`image`、`audio`、`video`，每种轨道能容纳的素材对应如下：

轨道	容纳素材	中文描述
title	title	文本
subtitle	subtitle	字幕
image	sequence	序列帧（此类素材只能从智能创作素材制作工具产出）
frame	frame,shader	特效
audio	audio	音频

video	video/image/transition	视频/图片/转场

素材类型协议描述请参见 轨道元素 。
决定展示内容的容器，只能导入我们的素材库，请参见 导入素材库 。

轨道元素

主要支持以下类型素材：

名称	说明
video	视频。
audio	音频。
image	图片。
title	作为内容填充的文本文字。
subtitle	字幕，语音内容的文字展示形式。
frame	特效。
transition	转场。

video

该种类的素材我们提供了更丰富的处理形式，具体请参见 多媒体处理 ，元素示例如下：

```
const videoItem = {
  id: "45ea83ad-a770-4e4a-a9b2-93cdefa4d369",
  start_time: 1440, //剪辑时间线上的开始时间
  duration: 7880, //剪辑时间线上的持续时间
  type: "video", //素材类型
  section: {
    // 素材时间戳截取，素材有自身时间线时，才有此属性。
    from: 0,
    to: 7880,
  },
  asset_id: "5f6464c6ccbc8d0001fc308b",
  width: 540,
  height: 304, // 前端预览暂不支持通过 height 调整视频大小
  position: {
    x: 270,
    y: 480,
```

```
  },
  operations: [
    //对该元素的操作处理，这里是指的旋转，具体请参见 多媒体处理章节。
    {
      type: "image_rotate",
      params: {
        angle: 0,
      },
    },
  ],
};
```

image

该种类的素材我们提供了更丰富的处理形式，具体请参见 多媒体处理，图片除开没有自身时间线外，与视频类型素材基本一致，示例如下：

```
const imageItem = {
  id: "45ea83ad-a770-4e4a-a9b2-93cdefa4d369",
  start_time: 1440, //剪辑时间线上的开始时间
  duration: 7880, //剪辑时间线上的持续时间
  type: "image", //素材类型
  asset_id: "5f6464c6ccbc8d0001fc308b",
  width: 540,
  height: 304, // 前端预览暂不支持通过 height 调整图片大小
  position: {
    x: 270,
    y: 480,
  },
  operations: [
    //详情参见多媒体处理章节。
    {
      type: "image_rotate",
      params: {
        angle: 0,
      },
    },
  ],
};
```

audio

该种类的素材我们提供了更丰富的处理形式，具体请参见 多媒体处理，示例如下：

```
const audioItem = {
  id: "3259bbc5-c581-43eb-96a3-1fac66c46aa8",
  type: "audio",
  order: 0,
  items: [
    {
      id: "c6424c76-0320-4f92-9607-d2a53748fb14",
      start_time: 0,
      duration: 60080,
      type: "audio",
      asset_id: "28192155374371990 2@Public@CME",
      section: {
        from: 0,
        to: 60080,
      },
      operations: [], //详情参见多媒体处理章节。
    },
  ],
};
```

title

作为内容填充的文本文字，内容填充的文字可以调整样式，示例数据如下：

```
const titleItem = {
  id: "f83a90eb-a9d9-4634-81d4-c7dc130afaec",
  start_time: 80,
  duration: 3000,
  type: "advanced_title", // 固定文字类型不可变更。
  content: {
    template_id: "yj_templ_title_text", //固定模板ID,不可变更
    params: {
      text: "文字标题", //文字内容
      text_style: {
        //具体属性可参考下面的列表
        font_size: 60,
        font_color: "#ffffff",
        align: "center",
        height: 220,
        bold: 0,
        italic: 0,
      },
    },
```

```
  },
  position: {
    x: 270,
    y: 480,
  },
  width: 240, // 前端预览暂不支持通过 width 调整文字宽度
  height: 69, // 前端预览暂不支持通过 height 调整文字高度
  transition: [
    //动画效果
    {
      start_time: 0, //动画效果起始时间
      duration: 500, //动画效果持续时间
      type: "in", //动画类型入场
      name: "fade",
    },
    {
      start_time: 2500,
      duration: 500,
      type: "out", //动画类型出场
      name: "move", //
    },
  ],
  // 动画效果，用于前端预览,组装复杂协议建议使用我们提供的sdk
  transitionFE: {
    start: {
      name: "FadeIn", //动画类型入场
      duration: 500, //动画效果持续时间
    },
    end: {
      name: "MoveOut", //动画类型出场
      duration: 500,
    },
  },
};
```

说明

文字 `text_style` 可支持文字的 通用属性设置 。自由文字动画效果，暂时只支持一个入场一个出场。

名称	描述	出入场限制
fade	淡入	in，out

move	左移入	in，out
scale	放大	in，out
bounce	弹跳	in

subtitle

字幕类型基本示例如下，字幕类型素材的文字样式轨道通用，所以文字样式是挂载在轨道数据结构：

```
const subTitleItem = {
  id: "3d343d8d-f950-40ea-aa50-e4e0c16cf2f2",
  start_time: 480,
  duration: 3000,
  type: "subtitle",
  style_id: "38192155374370820 6@Public@CME",
  text: "字幕文字1",
  asset_id: "38192155374370820 6@Public@CME",
};

/*****
 *  字幕样式对象
 * **/
const styleItem = {
  id: "38192155374370820 6@Public@CME",
  content: {
    template_id: "yj_templ_subtitle_common",
    params: {
      font_size: 60,
      font_color: "#ffffff",
      align: "center",
      height: 110,
      bold: 0,
      italic: 0,
      background_color: "#000000",
      background_alpha: 100,
      font: "SimHei",
      margin_bottom: 125,
      font_align: "right",
      font_align_margin: 30,
      bottom_color: "#3A1616",
      bottom_alpha: 100,
      border_color: "#FFFFFF",
      border_width: 2,
```

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

> **说明**
> 字幕文字的 `params` 支持 文字通用设置 。

frame

> **注意**
> 特效元素，只支持固定模板，和固定操作。特效必须覆盖其它素材，所以轨道层级处于顶层位置。更多内容请参见 image-glshader 。

示例：

```
const item = {
  id: "7ae45f5d-f351-4351-b501-64f36ab74f2d",
  width: 960,
  height: 540, // 前端预览暂不支持通过 height 调整大小
  start_time: 0,
  duration: 3000,
  type: "frame",
  operations: [
    {
      type: "image_glshader",
      params: {
        name: "LightCircle", //可替换模板
      },
    },
  ],
  shaderName: "LightCircle", //预览模板
  controlVisible: true, // 前端预览控制是否可见
};
```

transition

> **注意**
> 转场元素，只支持固定模板。转场元素必须在两个 video 元素之间。转场会叠加两个元素之间时间，转场是平均过渡，所以对转场前后元素有时间要求。推荐使用我们的 SDK 做转场添加操作。具体请参见 典型案例 。

代码示例：

```
const tracks = [
  {
    id: "fdc7bba6-fed6-4fb2-8fe8-e471f599de8b",
    type: "video",
    order: 0,
    items: [
      {
        id: "11da8fa9-3303-4dd9-9895-c112254ad274",
        start_time: 0,
        duration: 6620,
        type: "video",
        section: {
          from: 0,
          to: 6620,
        },
        asset_id: "5f3de82664ac1d000117a8e5",
        filter_asset_id: "",
        width: 960,
        height: 540,
        position: {
          x: 480,
          y: 270,
        },
        operations: [
          {
            type: "image_rotate",
            params: {
              angle: 0,
            },
          },
        ],
      },
      /**
       *这是转场元素，转场时长3秒，注意前后视频元素与转场元素的时间覆盖。
       **/
      {
        id: "184b06ab-e406-4bc7-b29d-d50c803321cd",
        asset_id: "3819215537437092570Public@CME",
        start_time: 3620,
        duration: 3000,
        type: "transition",
```

```
      prev_item_id: "11da8fa9-3303-4dd9-9895-c112254ad274",
      next_item_id: "c3d59391-0e37-4e98-a03a-944fba0670a4",
      operations: [
        {
          type: "trans_image_glt",
          params: {
            name: "doorway", //转场效果名字
          },
        },
        {
          type: "trans_audio_fade_inout",
          params: {},
        },
      ],
      record: {
        prev_item_init: 5120,
        next_item_init: 5120,
        next_item_start_time: 5120,
      },
    },
    {
      id: "c3d59391-0e37-4e98-a03a-944fba0670a4",
      start_time: 3620,
      duration: 6380,
      type: "video",
      section: {
        from: 3620,
        to: 10000,
      },
      asset_id: "5f3de82664ac1d000117a8e5",
      filter_asset_id: "",
      width: 960,
      height: 540,
      position: {
        x: 480,
        y: 270,
      },
      operations: [
        {
          type: "image_rotate",
          params: {
            angle: 0,
          },
        },
```

```
          ],
        },
      ],
    },
];
```

转场效果列表

素材 ID	名称	中文描述
617281ea02af87115081286b@Public@CME	crosshatch	交叉网格
617281ea02af87115081286a@Public@CME	colorphase	色彩溶解 2
617281ea02af871150812869@Public@CME	circle	椭圆遮罩
617281ea02af871150812868@Public@CME	cannabisleaf	枫叶遮罩
617281ea02af871150812867@Public@CME	burn	燃烧转场
617281ea02af871150812866@Public@CME	angular	逆时扫描
617281ea02af871150812865@Public@CME	WaterDrop	水滴溶解
617281ea02af871150812864@Public@CME	StereoViewer	分身融合
617281ea02af871150812863@Public@CME	SimpleZoom	拉近变焦
617281ea02af871150812862@Public@CME	GridFlip	网格翻转
617281ea02af871150812861@Public@CME	GlitchDisplace	水银溶解
617281ea02af871150812860@Public@CME	CrossZoom	交叉变焦
617281ea02af87115081285f@Public@CME	ColourDistance	色彩溶解 1
617281ea02af87115081285e@Public@CME	Bounce	向下跳动
617281ea02af87115081285d@Public@CME	Directional	向下平移
617281ea02af87115081285b@Public@CME	fadecolor	黑色转场
617281ea02af87115081285a@Public@CME	LinearBlur	交叉溶解
600590f402af8747f6eb32e8@Public@CME	ZoomInCircles	水波纹
600590f402af8747f6eb32e7@Public@CME	wipeUp	向上收起
600590f402af8747f6eb32e6@Public@CME	wipeRight	向右收起

617281ea02af87115081287f@Public@CME	windowslice	窗口切片
617281ea02af87115081287e@Public@CME	wind	线条向右
617281ea02af87115081287d@Public@CME	undulatingBurnOut	椭圆遮罩 2
617281ea02af87115081287c@Public@CME	squeeze	挤压
617281ea02af87115081287b@Public@CME	squareswire	矩形溶解 2
617281ea02af87115081287a@Public@CME	rotate_scale_fade	风车旋转
617281ea02af871150812879@Public@CME	ripple	波纹溶解
617281ea02af871150812878@Public@CME	randomsquares	矩形溶解 1
617281ea02af871150812877@Public@CME	pixelize	像素溶解
617281ea02af871150812876@Public@CME	pinwheel	风车遮罩
617281ea02af871150812875@Public@CME	perlin	花边遮罩
617281ea02af871150812874@Public@CME	multiply_blend	多重混合
617281ea02af871150812873@Public@CME	morph	缓慢抖动
617281ea02af871150812872@Public@CME	luminance_melt	线条向下
617281ea02af871150812871@Public@CME	kaleidoscope	万花筒
617281ea02af871150812870@Public@CME	hexagonalize	六边形网格
617281ea02af87115081286f@Public@CME	flyeye	水波溶解
617281ea02af87115081286e@Public@CME	fadegrayscale	缓慢渐变
617281ea02af87115081286d@Public@CME	displacement	左上擦除
617281ea02af87115081286c@Public@CME	directionalwipe	右下擦除
600590f402af8747f6eb32e5@Public@CME	wipeLeft	向左收起
600590f402af8747f6eb32e4@Public@CME	wipeDown	向下收起
600590f402af8747f6eb32e3@Public@CME	windowblinds	百叶窗
600590f402af8747f6eb32e2@Public@CME	Swirl	螺旋
600590f402af8747f6eb32e1@Public@CME	swap	放大切换

600590f402af8747f6eb32e0@Public@CME	Radial	雷达扫描
600590f402af8747f6eb32df@Public@CME	PolkaDotsCurtain	弧形扩散
600590f402af8747f6eb32de@Public@CME	polar_function	椭圆扩散
600590f402af8747f6eb32dd@Public@CME	DoomScreenTransition	幕布
600590f402af8747f6eb32dc@Public@CME	cube	立方体
600590f402af8747f6eb32db@Public@CME	circleopen	椭圆聚拢
600590f402af8747f6eb32da@Public@CME	ButterflyWaveScrawler	晃动
381921553743709260@Public@CME	InvertedPageCurl	翻页
381921553743709259@Public@CME	heart	心形
381921553743709256@Public@CME	crosswarp	交叉扭曲
381921553743709258@Public@CME	GlitchMemories	抖动
381921553743709253@Public@CME	Dreamy	波浪
381921553743709257@Public@CME	doorway	门廊
381921553743709255@Public@CME	DreamyZoom	梦幻变焦
381921553743709249@Public@CME	BowTieHorizontal	水平蝴蝶结
381921553743709252@Public@CME	Mosaic	九宫格
381921553743709251@Public@CME	CircleCrop	圆环聚拢
381921553743709250@Public@CME	BowTieVertical	垂直蝴蝶结
381921553743708553@Public@CME	fade	淡入淡出

元素类型及轨道关系

资源类型	元素子类型	所在轨道类型
自由文字	advanced_title	title
音频	audio	audio
马赛克	frame	frame
图片	image	video

视频	video	video
转场	transition	video
字幕	subtitle	subtitle
风格	frame	frame，subType="shader"

轨道元素必要属性

```
const trackItem = {
  id: "61afa77e-8c1f-4fcb-8255-f4d8f4617d0b", // 元素ID,单个剪辑协议内唯一，如果使用我们的sdk，可以不关注此id的的生产。
  start_time: 880, // 剪辑时间线上的开始时间。
  type: "video", // 素材类型.表明当前素材类型,以决定如何处理.
  duration: 3000, // 播放持续时间。
  asset_id: "7311950889467332 55", // 媒资id。
};
```

轨道元素位置计算

默认的舞台分辨率为 `960 * 540` ，采用直角坐标系确定元素位置，以左上角为原点(0,0)。整个舞台位于坐标系第一象限内。元素的 `position` 属性默认描述元素中心点。例如一张 `100*100` 的图片紧贴原点完整放置在舞台上，那么它的 `position` 值为(50,50)。

属性值	类型	说明
x	number	横坐标。
y	number	纵坐标。

轨道元素截取

> **说明**
> 仅在自身有时间线的元素上存在，例如 音频，视频。

属性值	类型	说明
from	number	片段起始时间戳，单位为毫秒 ms。
to	number	片段末尾时间戳，单位为毫秒 ms。

轨道元素通用数据

文字通用属性

> **说明**
> 适用于 `title` 的 `text_style` 属性，以及 `subtitle` 的 `params` 属性。

属性值	类型	说明
height	number	高度，仅字幕可用。（**必要**）
font_size	number	字体大小。（**必要**）
font_color	string	字体颜色，16 进制 RGB，如 `#ffffff` 。(**必要**)
font	string	字体名
font_alpha	number	背景透明度，范围[0, 100]。
font_align	string	水平对齐方式：center（默认）、left、right。
font_uline	number	下划线宽度。
font_align_margin	number	大于等于 0，font_align=left 表示左边距，font_align=right 表示右边距，其他无效。
bold	number	字体加粗，默认 0（不加粗）、1（加粗）。仅可用于黑体、宋体和楷体。字幕（subtitle）使用 bold 属性，内容填充文字（advanced_title）使用 font_bold。
italic	number	字体倾斜，默认 0（正常）、1（斜体），字幕（subtitle）使用 italic 属性，内容填充文字（advanced_title）使用 font_italic。
border_width	number	边框宽度，取值只支持 0 和 1。
border_color	string	边框颜色，border_width 不为 0 时有效。
border_alpha	number	边框透明度，border_width 不为 0 时有效，取值只支持 0 和 100。
shadow_color	string	文字阴影，仅内容填充文字（advanced_title）可用。

shadow_alpha	number	背景透明度，范围[0, 100]，shadow_color 存在时有效，仅内容填充文字（advanced_title）可用。
bottom_color	string	文字底色，仅字幕（subtitle）可用。
bottom_alpha	number	文字底色透明度，范围[0, 100]，bottom_color 存在时有效，仅字幕（subtitle）可用。
background_color	string	全屏背景颜色，16 进制 RGB，如 `#000000`。
background_alpha	number	全屏背景透明度，范围[0, 100]，background_color 存在时有效。
margin_bottom	number	仅字幕（subtitle）可用，默认为 0。

多媒体处理

> **说明**
> 该属性允许调整素材的一些基础信息，通过 `operations` 字段进行设置，完整能力如下列表，Y 表示支持此操作，N 表示不支持。

operation	video	image	audio	transition	frame	备注
image_mirror	支持	支持	不支持	不支持	不支持	图像镜像。
image_rotate	支持	支持	不支持	不支持	不支持	图像旋转。
image_filter_normal	支持	支持	不支持	不支持	不支持	普通图像滤镜。
image_filter_lut	支持	支持	不支持	不支持	不支持	Lut 滤镜模板。
image_transparent	支持	支持	不支持	不支持	不支持	图像透明度。
image_crop	支持	支持	不支持	不支持	不支持	图像剪切，**不可单独预览**。

image_space	支持	支持	不支持	不支持	不支持	图像缩放，**不可单独预览**。
image_mosaic	不支持	不支持	不支持	不支持	支持	马赛克。
image_glshader	不支持	不支持	不支持	不支持	支持	特效。
image_lens_stretch	支持	支持	不支持	不支持	支持	运动效果。
audio_volumes	支持	不支持	支持	不支持	不支持	音量调整。

基础示例：

```
/* 图像（图片、视频图像） 相关的操作 */
const imageMirrorOp = {
  type: "image_mirror", //  【必选】【string】  operation 类型，
  params: {
    left_right: 1, //  【必选】【int】  取值：1-左右镜像
    up_down: 1, //  【必选】【int】  取值：1-上下镜像
  },
};
//  图像旋转
const imageRotateOp = {
  type: "image_rotate", //  【必选】【string】  operation 类型，
  params: {
    angle: 90, //  【必选】【int】  旋转角度
  },
};
// 图像简单滤镜
const imageFilterNormalOp = {
  type: "image_filter_normal",
  params: {
    contrast: 100, //  【必选】【int】  对比度，取值范围 [-100, 100]，0 表示不处理
    brightness: 50, //  【必选】【int】  亮度，取值范围 [-100, 100]，0 表示不处理
    saturation: -20, //  【必选】【int】  饱和度，取值范围 [-100, 100]，0 表示不处理
  },
};
```

```
// 图像透明
const imageTransparentOp = {
  type: "image_transparent",
  params: {
    alpha: 0, //  【必选】【int】  透明度，取值范围[0,100]，0表示透明
  },
};
// 图像裁剪
const imageCropOp = {
  type: "image_crop",
  params: {
    x: 0, //  【必选】【int】  裁剪起始点(相对于item的)
    y: 0, //  【必选】【int】  裁剪起始点(相对于item的)
    width: 0, //  【必选】【int】  裁剪宽度
    height: 0, //  【必选】【int】  裁剪高度
  },
};
// 图像马赛克
const imageMosaicOp = {
  type: "image_mosaic",
  params: {
    name: "vague", //  【必选】【string】  马赛克类型，目前只支持模糊(vague)、方
块(mosaic)
    x: 0, //  【必选】【int】  马赛克起始点(相对于item的)
    y: 0, //  【必选】【int】  马赛克起始点(相对于item的)
    width: 0, //  【必选】【int】  马赛克宽度
    height: 0, //  【必选】【int】  马赛克高度
    degree: 100, //  【可选】【int】  马赛克程度，范围[0,100]
  },
};

// 调整大小和位置

const imageSpaceOp = {
  type: "image_space",
  params: {
    x: 0, //  【可选】【int】  存在表示新的中心点x
    y: 0, //  【可选】【int】  存在表示新的中心点y
    width: 0, //  【可选】【int】  存在且大于0，表示新的宽度
    height: 0, //  【可选】【int】  存在且大于0，表示新的高度
  },
};
// 图像opengl shader效果
```

```
const imageGlshaderOp = {
  type: "image_glshader"
  params: {
    name: "xxx", //  【必选】【string】  效果名称
  },
};

// 音量调整
const audioOp =   {
  type: "audio_volumes",
  params: {
    "all": 52  // 【必选】【number】  音量大小，值域0-100
  }
};
```

- **crop**

  裁剪功能预览的流程如下：

  1.1 将待裁剪视频按照满屏的方式放置到舞台内。

  1.2 在满屏基础上设置裁剪区域。

  1.3 还原视频位置信息。

  > **说明**
  > 裁剪数据设置流程复杂，如需设置也可以使用我们 SDK 提供的 `crop` 方法。

  示例：

```
const item = {
id: "0bfdd748-b548-42f4-9711-09ae5c534061",
start_time: 0,
duration: 63560,
type: "video",
section: {
  from: 0,
  to: 63560,
},
asset_id: "5f6464c6ccbc8d0001fc308b",
width: 960, //视频原始宽高
height: 254, //视频原始宽高
position: {
  //元素原本位置信息
  x: 480,
```

```
    y: 270,
  },
  operations: [
    {
      /**
   * 对视频重设，让其满屏
   **/
      type: "image_space",
      params: {
   x: 480,
   y: 270,
   width: 960,
   height: 540,
      },
      from: "crop_start", //表示裁剪开始
    },
    {
      /**
   * 设置裁剪区域，裁剪区域是一个矩形，
   * x,y为裁剪位置
   * width,height表示裁剪区域
   **/
      type: "image_crop",
      params: {
   x: 376,
   y: 28,
   width: 223,
   height: 59,
      },
    },
    {
      /**
   * 还原元素位置和大小，这个示例里裁剪开始前与裁剪结束后位置是一样的。
   **/
      type: "image_space",
      params: {
   x: 480,
   y: 270,
   width: 960,
   height: 254,
      },
      from: "crop_end", //裁剪结束
    },
  ],
```

```
  sizeControl: 0,
  };
```

- **image-mosaic**

> **说明**
> 特效类轨道必须是覆盖在普通视频轨道之上。

示例：

```
/***
 * 必须使用特效类轨道。
 ***/
const trackItem = {
  id: "adfe3eab-db8f-4b4c-96e8-730bf3a9226d",
  type: "frame",
  order: 0,
  items: [],
};

const item = {
  id: "0c5e990d-bdf9-47f6-b1a5-ac7639060e4f",
  asset_id: "fe-731195088946733254",
  start_time: 0,
  duration: 3000,
  type: "frame",
  operations: [
    {
      type: "image_mosaic", //类型固定为 image_mosaic
      params: {
        name: "mosaic", //名字固定为 image_mosaic
        degree: 20,
        x: 311,
        y: 40,
        width: 320,
        height: 180,
      },
    },
  ],
};

/**
 * 放入轨道
```

```
 * */
track.items.push(trackItem);

/**
 * 最终数据结构
 * */
const data = [trackItem];
```

- **image-filter-lut**

  协议示例大致如下：

```
const item = {
id: "7edc76a8-1a28-4546-b0ca-fd7403660ca1",
start_time: 0,
duration: 63560,
type: "video",
section: {
  from: 0,
  to: 63560,
},
asset_id: "5f6464c6ccbc8d0001fc308b",
width: 960,
height: 540,
position: {
  x: 480,
  y: 270,
},
operations: [
  {
    type: "image_rotate",
    params: {
      angle: 0,
    },
  },
  {
    /**
     * 注意：这里要使用我们的协议工具辅助完成拼装。
     * 详情请看下面
     ***/
    type: "image_filter_lut",
    params: {
      name: "yj_templ_lut_24", //修改此模板ID即可变更效果
      image_url:
```

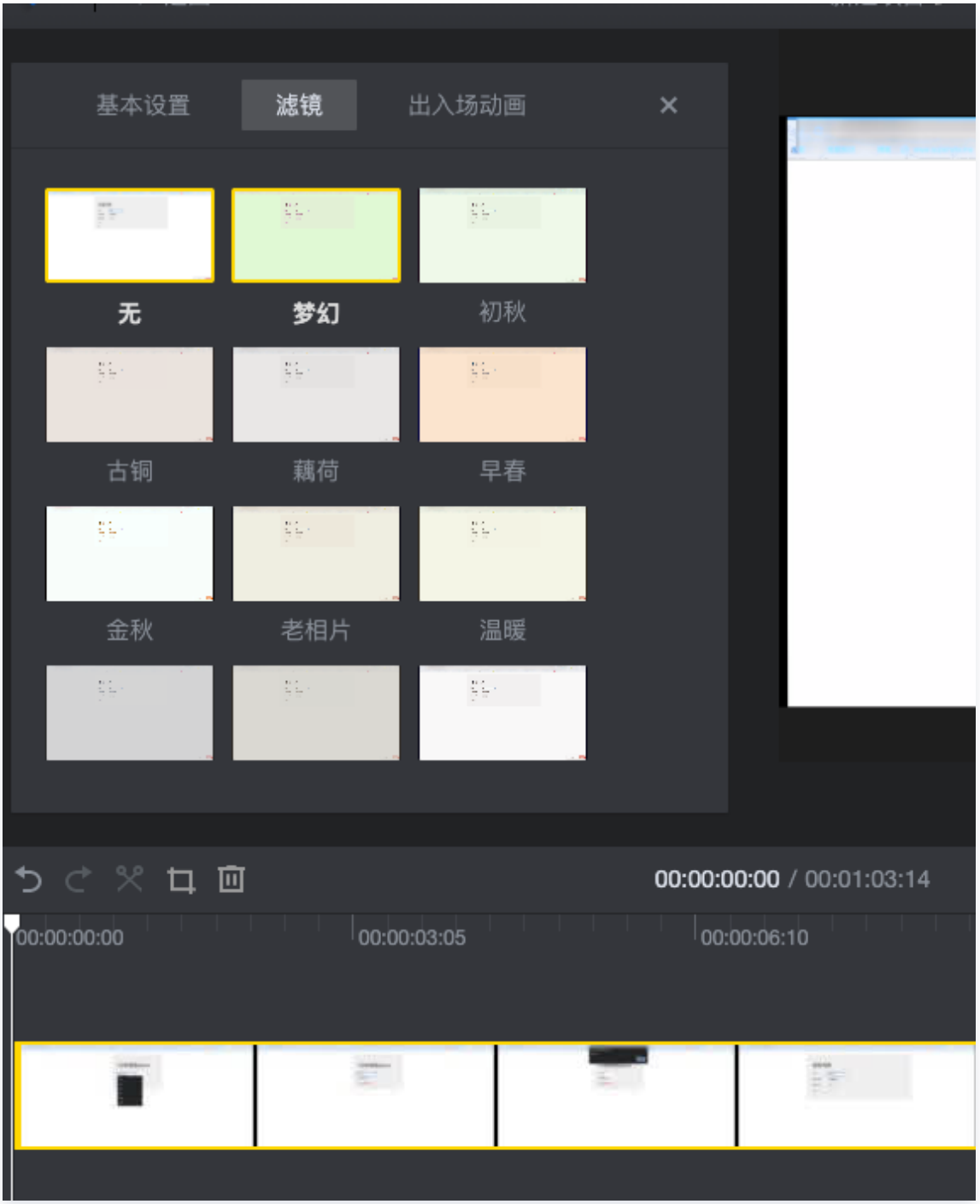
```
"https://1810000000.vod2.myqcloud.com/b64e98acvodcq1810000000/thumbnail
/filter/yj_templ_lut_24.png",
    },
  },
],
};
```

> **注意**
>
> 滤镜素材使用请参见 典型案例–使用公共滤镜 。

```
/**
 * 所有的滤镜素材
 **/
const tplMap = [
  { Name: "氰版照", MaterialId: "5fd329e702af8755735859a1@Public@CME" },
  { Name: "胶片", MaterialId: "5fd329e702af87557358599f@Public@CME" },
  { Name: "魅影", MaterialId: "5fd329e702af87557358599e@Public@CME" },
  { Name: "冷酷", MaterialId: "5fd329e702af87557358599d@Public@CME" },
  { Name: "美式", MaterialId: "5fd329e702af87557358599c@Public@CME" },
  { Name: "天王星", MaterialId: "5fd329e702af8755735859a4@Public@CME" },
  { Name: "柯达", MaterialId: "5fd329e702af87557358599b@Public@CME" },
  { Name: "蜡纸底片", MaterialId: "5fd329e702af8755735859a5@Public@CME"
},
  { Name: "凹版照相", MaterialId: "5fd329e702af8755735859a2@Public@CME"
},
  { Name: "树脂", MaterialId: "5fd329e702af8755735859a3@Public@CME" },
  { Name: "橙色天空", MaterialId: "5fd329e702af875573585996@Public@CME"
},
  { Name: "中性", MaterialId: "5fd329e702af875573585995@Public@CME" },
  { Name: "幻紫", MaterialId: "5fd329e702af875573585994@Public@CME" },
  { Name: "午后", MaterialId: "5fd329e702af87557358599a@Public@CME" },
  { Name: "假日", MaterialId: "5fd329e702af875573585992@Public@CME" },
  { Name: "黄昏", MaterialId: "5fd329e702af875573585999@Public@CME" },
  { Name: "胡桃木", MaterialId: "5fd329e702af875573585991@Public@CME" },
  { Name: "红杉", MaterialId: "5fd329e702af875573585998@Public@CME" },
  { Name: "水墨", MaterialId: "5fd329e702af875573585993@Public@CME" },
  { Name: "卷曲粉红", MaterialId: "5fd329e702af875573585997@Public@CME"
},
  { Name: "青山", MaterialId: "5fd329e702af87557358598c@Public@CME" },
  { Name: "朋克", MaterialId: "5fd329e702af87557358598b@Public@CME" },
  { Name: "青色", MaterialId: "5fd329e702af87557358598a@Public@CME" },
```

```
    { Name: "城镇", MaterialId: "5fd329e702af875573585989@Public@CME" },
    { Name: "小镇", MaterialId: "5fd329e702af875573585988@Public@CME" },
    { Name: "珠光蓝", MaterialId: "5fd329e702af875573585987@Public@CME" },
    { Name: "巧克力", MaterialId: "5fd329e702af875573585990@Public@CME" },
    { Name: "自由蓝", MaterialId: "5fd329e702af87557358598f@Public@CME" },
    { Name: "都市", MaterialId: "5fd329e702af87557358598e@Public@CME" },
    { Name: "雪域森林", MaterialId: "5fd329e702af87557358598d@Public@CME"
  },
    { Name: "橘光", MaterialId: "5fd329e702af875573585982@Public@CME" },
    { Name: "蒙层", MaterialId: "5fd329e702af875573585981@Public@CME" },
    { Name: "明亮", MaterialId: "5fd329e702af875573585985@Public@CME" },
    { Name: "洛莫彩色", MaterialId: "5fd329e702af875573585980@Public@CME"
  },
    { Name: "文艺", MaterialId: "5fd329e702af875573585984@Public@CME" },
    { Name: "藏青", MaterialId: "5fd329e702af87557358597e@Public@CME" },
    { Name: "录影带", MaterialId: "5fd329e702af87557358597f@Public@CME" },
    { Name: "梦想家", MaterialId: "5fd329e702af87557358597c@Public@CME" },
    { Name: "暗调", MaterialId: "5fd329e702af875573585986@Public@CME" },
    { Name: "青柠", MaterialId: "5fd329e702af875573585983@Public@CME" },
    { Name: "暗黑", MaterialId: "5fd329e702af875573585976@Public@CME" },
    { Name: "初秋", MaterialId: "281921553743709738@Public@CME" },
    { Name: "傍晚", MaterialId: "281921553743709742@Public@CME" },
    { Name: "温暖", MaterialId: "281921553743709741@Public@CME" },
    { Name: "冷光", MaterialId: "281921553743709740@Public@CME" },
    { Name: "梦幻", MaterialId: "281921553743709737@Public@CME" },
    { Name: "标准深", MaterialId: "5fd329e702af87557358597b@Public@CME" },
    { Name: "倒叙", MaterialId: "5fd329e702af87557358597a@Public@CME" },
    { Name: "仲夏", MaterialId: "5fd329e702af875573585979@Public@CME" },
    { Name: "青春", MaterialId: "5fd329e702af875573585978@Public@CME" },
    { Name: "老相片", MaterialId: "281921553743709733@Public@CME" },
    { Name: "宁静", MaterialId: "381921553743709127@Public@CME" },
    { Name: "古铜", MaterialId: "281921553743709739@Public@CME" },
    { Name: "清晨", MaterialId: "381921553743709125@Public@CME" },
    { Name: "藕荷", MaterialId: "281921553743709735@Public@CME" },
    { Name: "自然", MaterialId: "381921553743709128@Public@CME" },
    { Name: "早春", MaterialId: "281921553743709736@Public@CME" },
    { Name: "薄暮", MaterialId: "381921553743709126@Public@CME" },
    { Name: "金秋", MaterialId: "281921553743709734@Public@CME" },
    { Name: "晨光", MaterialId: "381921553743709129@Public@CME" },
];
```

具体效果可登录 腾讯制作云 进行查看，如下图：



- **image-filter-normal**

  示例：

```
const item = {
id: "7edc76a8-1a28-4546-b0ca-fd7403660ca1",
start_time: 0,
duration: 63560,
type: "video",
section: {
```

```
  from: 0,
  to: 63560,
},
asset_id: "5f6464c6ccbc8d0001fc308b",
width: 960,
height: 540,
position: {
  x: 480,
  y: 270,
},
operations: [
  {
    type: "image_rotate",
    params: {
      angle: 0,
    },
  },
  {
    type: "image_filter_normal", //修改params值即可调整对应效果
    params: {
      contrast: 100, //对比度
      brightness: 0, //明亮度
      saturation: 0, //饱和度
    },
  },
],
};
```

- **image-glshader**

  基础示例：

```
/**
 * 特效视频只能放到特效轨道上.注意特效必须覆盖其它素材。
 * **/
const track = {
  id: "61eed235-a61a-4ae0-baaf-49dda661c962",
  type: "frame",
  order: 1,
  items: [],
  subType: "shader",
};

/****
```

```
 * 特效元素
 * **/
const trackItem = {
  id: "03816c76-4ba0-48bd-b7c7-289f45002652",
  width: 960, //宽高取值与舞台保持一致
  height: 540, //宽高取值与舞台保持一致
  asset_id: "5f0aba91cb093b30566c5da9@Public@CME",
  start_time: 0,
  duration: 3000,
  type: "frame",
  operations: [
    /**
     * 如果是组合特效，则有两个操作对象
     * */
    {
      type: "image_glshader",
      params: {
        name: "LightCircle", //导出替换值
      },
    // 注：组合特效需要注意顺序，顺序改变导出效果会有变化
    // {
    //   type: "image_glshader",
    //   params: {
    //     name: "ScBlurThreeY", //导出替换值
    //   },
    // },
    // {
    //   type: "image_glshader",
    //   params: {
    //     name: "ScBlurThreeX", //导出替换值
    //   },
    // },
  ],
  /***
   * 由于部分滤镜特效是由多种操作组合成，例如注释内的高斯模糊效果，
   * 所以预览值和导出值有可能不一样。预览值表示可以在预览组件上看的效果。
   * **/
  shaderName: "LightCircle", //预览值
  // shaderName: "ScBlurThree" // 模糊
};

track.items.push(trackItem)

/**
```

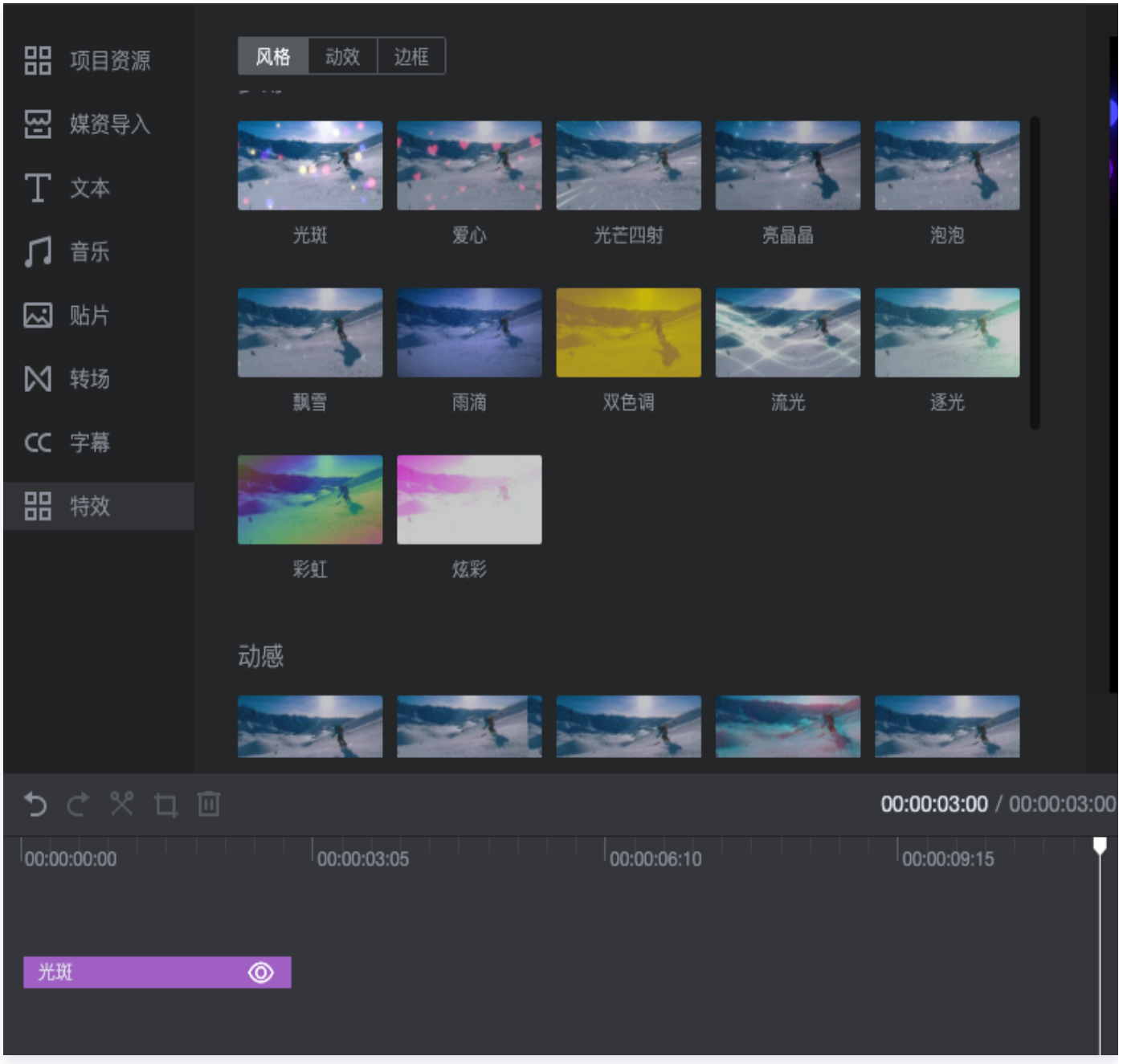
```
 * 最终数据结构
 * */
const data = [trackItem]
```

所有的特效描述：

预览名称	描述	组合
LightCircle	光斑	LightCircle
Heart	爱心	Heart
Shining	光芒四射	Shining
Blink	亮晶晶	Blink
Bubble	泡泡	Bubble
Snow	飘雪	Snow
Rain	雨滴	Rain
Duotone	双色调	Duotone
FlowingLight	流光	FlowingLight
ChasingLight	逐光	ChasingLight
Rainbow	彩虹	Rainbow
Multicoloured	炫彩	Multicoloured
Shake	抖动	Shake
Swing	摇晃	Swing
SoulOut	灵魂	SoulOut
Hallucination	幻觉	Hallucination
ShineWhite	闪白	ShineWhite
Glitch	故障	Glitch
OldVideo	老电影	OldVideo
Mirror	镜像	Mirror
ScBlurThree	模糊分屏	ScBlurThreeY、ScBlurThreeX

ScGrayThree	黑白三屏	ScGrayThree
ScTwo	两屏	ScTwo
ScThree	三屏	ScThree
ScFour	四屏	ScFour
ScSix	六屏	ScSix
ScNine	九屏	ScNine

具体效果可登录 腾讯制作云 进行查看，如下图：



- **image_lens_stretch**
  - **格式描述：**

```
const op = {
type: "image_lens_stretch",
transformName: "xxx", //【必选】【string】
params: {
  name: "xxx", //  【必选】【string】
  start_time: 0, //  【可选】【int】  存在且大于等于0，相对于item时间，效果开始时间，默认0
  duration: 0, //  【必选】【int】  存在且大于0，效果持续时间
  from_center: [], //  【可选】【float 数组】  初始状态中心点，size=2，[x, y]，取值>=0，默认0.5
  from_scale: [], //  【可选】【float 数组】  初始状态大小，  size=2，[width, height]，取值>=0，默认1
  to_center: [], //  【可选】【float 数组】  目标状态中心点，size=2，[x, y]，取值>=0，默认0.5
  to_scale: [], //  【可选】【float 数组】  目标状态大小，  size=2，[width, height]，取值>=0，默认1
},
};
// transformName-params.name

// 通用运动动画
// ZoomOut-Stretch 推远
// ZoomIn-Stretch 拉近
// MoveLeft-Transform 左移
// MoveRight-Transform 右移
// MoveDown-Transform 上移
// MoveUp-Transform 下移

// 文字动画
// FeatherRight-FeatherRight 羽化向右展开
// FeatherLeft-FeatherLeft 羽化向左展开
// Typewriter-Typewriter 打字机
// WipeRight-WipeRight 向右擦除
// WipeLeft-WipeLeft 向左擦除
```

  - **基础示例：**

```
/****
```

```
 * 运动元素
 * 这里以图片运动为例
 * **/
const trackItem = [
  {
    id: "5e08d7bb-a3df-4721-a9dc-2d7cf1ffb7fe",
    type: "video",
    order: 0,
    items: [
      {
        id: "94ea0aea-ad23-4e1a-aae1-52f1f8bc5805",
        start_time: 0,
        duration: 8080,
        type: "image",
        asset_id: "5f3cca3bb439500001b2edd8",
        filter_asset_id: "",
        operations: [
          {
            type: "image_lens_stretch",
            transformName: "ZoomOut",
            params: {
              name: "Stretch",
              duration: 8100,
              start_time: 0,
              from_scale: [2.5, 2.5],
              to_scale: [1.1, 1.1],
              from_center: [0.5, 0.5],
              to_center: [0.5, 0.5],
            },
          },
          {
            type: "image_rotate",
            params: {
              angle: 0,
            },
          },
        ],
        width: 673,
        height: 540,
        position: {
          x: 480,
          y: 270,
        },
        section: {
```

```
            from: 0,
            to: 8080,
          },
          transitionFE: {
            start: {
              name: "",
              duration: 0,
            },
            end: {
              name: "",
              duration: 0,
            },
          },
          transition: [],
        },
      ],
    },
];

/**
 * 最终数据结构
 * */
const data = [trackItem];
```

合成协议助手

CMEUtils

`YJPlayer.Helper.CMEUtils` 协助用户处理媒资素材。

auth

`YJPlayer.Helper.CMEUtils.auth`

在远端登录，以便获取资源，再次调用可以新增登录角色。

> **注意**
> 没有登录则以下部分接口无法使用，登录后再调用以下方法。

- **输入参数**

参数	描述	类型	必填
sign	身份验证签名，详情请参见 签名概述	string	是

- **返回值**

```
type result = Promise<{
code:string
}>
```

logout

登出

- **输入参数**

  空。

- **返回值**

  空。

createTrackItem

创建 `轨道元素(trackItem)` 数据，可以直接添加到 `轨道(track)` 中使用。

- **输入参数**

参数	描述	类型	必填
materialIds	媒资 ID	string[]	是

- **返回值**
- 返回一个基础的视频合成协议元素，详情请参见 轨道元素 。

```
type result = Promise<{
id: string;
type: materialType;
start_time: number;
duration: number;
asset_id: string;
}>;
```

- **示例**

```
/**
 *     @auth 登录方法，保证可以调用CME，WebAPI。
 *     @param sign {{string}} 签名串。
 **/
YJPlayer.Helper.CMEUtils.auth({
  sign: "your_sign",
```

```
})
  .then(() => {
    console.log("登录成功");
  })
  .catch((err) => {
    console.error("登录异常");
  });

/**
 *  @createTrackItem 创建轨道元素协议数据。
 *  @param asset_ids {{string[]}} 媒资ID数组
 *  @param options {{object}} 可选参数
 * */
YJPlayer.Helper.CMEUtils.createTrackItem(["materialId1",
"materialId2"], {})
  .then((data) => {
    console.log("创建成功", data);
  })
  .catch((err) => {
    console.log("创建失败", err);
  });
```

Track & TrackItem

`YJPlayer.Helper.Track` 和 `YJPlayer.Helper.TrackItem` 两种操作场景，包含方法如下：

操作场景	命名空间	描述
Track 操作场景	create	创建一个包含基础协议格式的轨道数据
	append	添加轨道元素方法
	sort	按照我们预设层级逻辑为轨道重新排列
	addMosaic	添加马赛克
	addSubtitle	添加字幕
TrackItem 操作场景	create	创建非素材类型的轨道元素
	crop	对素材进行裁剪

Track 操作场景

- **create**
  - **输入参数**

参数	描述	类型	必填
options	轨道设置，具体类型请参见 轨道描述	{type:'video'/'audio'/....}	是
index	轨道层级	number	否，默认从 0 开始自增。注：轨道有层级顺序，参考 sort 说明。

- **返回值**

  合成协议的基础请参见 轨道数据 。

```
interface Track {
  id: string;
  type: string;
  order: number;
  items: any[];
}

type result = Promise<Track>;
```

- **示例**

```
const imageTrack = YJPlayer.Helper.Track.create({
  type: "image",
});
console.log(imageTrack);
```

- **append**
  - **输入参数**

参数	描述	类型	必填
materialId	媒资 ID	string	是
trackItem	轨道元素	object	是
insertIndex	插入位置，默认插入队尾	number	否

- **返回值**

  已填充元素的轨道数据。

```
type result = Track;
```

- **示例：**

```
const imageTrack = YJPlayer.Helper.Track.create({
type: "image",
});
const item = {};
YJPlayer.Helper.CMEUtils.createTrackItem(["materialId"],
{}).then((trackItem) => {
Track.append(imageTrack, trackItem);
console.log(imageTrack);
});
```

- **sort**
- **轨道排序方法，会将传入轨道设置到正确的层级上，由于展示问题，部分轨道应该被摆到合适的位置才会有效果，例如字幕应该是处于靠前位置的轨道。以下是轨道权重，如并列放置则可以随意调整层级：**

```
- title
- frame
- subtitle
- image/audio/video
```

- **输入参数**

参数	描述	类型	必填
tracks	包含 轨道 数据的数组	object	是

- **返回值**

  已排序的轨道数组。

```
type result = Track[];
```

- **示例**

```
const imageTrack = YJPlayer.Helper.Track.create({
```

```
  type: "image",
});
const videoTrack = YJPlayer.Helper.Track.create({
  type: "video",
});
const subtitleTrack = YJPlayer.Helper.Track.create({
  type: "subtitle",
});

const sortedData = YJPlayer.Helper.Track.sort([
  imageTrack,
  videoTrack,
  subtitleTrack,
]);

/**
 * subtitle会被排到最前面
 * */
console.log(sortedData);
```

- **addMosaic**

  给合成协议添加马赛克。

  - **输入参数**

参数	描述	类型	必填
params	左上角开始坐标位置(x,y)，裁剪矩行区域 (width,height)，{x,y,width,height}	object	是
fusionData	包含 轨道 的数组	object	是

  - **返回值**

    已经添加过马赛克轨道的数据协议。

```
type result = Track[];
```

  - **示例**

```
/**
 * 先获取一段已有的合成数据。
 */
const fusiontData = [
```

```
  /*
   * 一段轨道数据
   **/
];
/**
 * 从视频的左上角位置开始，位于(10,10)，
 * 宽高为(50,50)的部分打上马赛克。
 */
const result = YJPlayer.Helper.Track.addMosaic(
  {
    x: 10,
    y: 10,
    width: 50,
    height: 50,
  },
  fusiontData
);
console.log(result);
```

- **addSubtitle**

  添加字幕轨道功能。

  - **输入参数**

参数	描述	类型	必填
fusionData	包含 轨道 的数组	object	是
param	起始时间，持续时间，文字内容 {start_time，duration，text}	object	是
styles	文字样式 {font_size,font_color,align,italic,background_color,background_alpha,height}，请参见 示例	object	否

  - **返回值**

    已经添加过字幕轨道的数据协议。

```
type result = Track[];
```

  - **示例**

```
/**
 * 先获取一段已有的合成数据。
 */
const fusiontData = [
  /*
   * 一段轨道数据
   **/
];
/***
 *
 * 视频第一秒开始，字幕持续1秒，字幕内容是“这是字幕1”，
 * 以此类推
 */
const subtitles = [
  {
    duration: 1000, //持续时间 ，单位ms
    text: "这是字幕1",
    start_time: 1000, //开始时间点.单位ms
  },
  {
    duration: 1000,
    text: "这是字幕2",
    start_time: 3000,
  },
];
const result = YJPlayer.Helper.Track.addSubtitle(fusiontData,
subtitles, {
  font_size: 12, //文字大小 ，单位px
  font_color: "#cb0000", // 文字颜色
  align: "center", //"center" | "left" | "right"s
  italic: false, // 文字斜体
  background_color: "#000", //字幕背景
  background_alpha: 50, //背景透明度0-100
  height: 100, //高度 单位像素
});
console.log(result);
```

TrackItem 操作场景

- **create**
  创建非素材类型的轨道元素，需要传入相关参数。
  - **输入参数**

参数	描述	类型	必填
type	元素类型	string	是
material_type	元素的子类型（同一个元素类型有不同的子类型）	string	是
start_time	剪辑时间线上的开始时间	number	是
duration	剪辑时间线上的持续时间	number	是
width	元素在舞台中展示的宽度	number	是
height	元素在舞台中展示的高度	number	是
position	元素在舞台中的位置	object { x: number; y: number;}	是
section	元素片段时间线	object { from: number; to: number;}	音频、视频、特效等元素必填
text	元素文本内容	string	字幕和文本元素必填
content	元素辅助性元数据内容	object	文本和自由文字元素必填
style_id	字幕样式 ID	string	仅字幕元素必填
asset_url	元素媒资 URL	string	非文本和自由文字元素必填
thumbnail_url	元素媒资 URL，提供裁剪尺寸时可用于内部优化	string	否
asset_id	元素媒资 ID	string	否

- **返回值**

  携带裁剪操作数据的轨道元素对象，完整数据结构请参见 合成协议 。

```
/**
```

```
 *  @create 创建轨道元素协议数据。
 *  @param options {{object}} 可选参数
 * */
const trackItem = YJPlayer.Helper.TrackItem.create({
  type: "video",
  material_type: "video",
  start_time: 0,
  duration: 10000,
  width: 960,
  height: 540,
  position: {
    x: 480,
    y: 270,
  },
  section: {
    from: 0,
    to: 10000,
  },
  asset_url: "https://xxx.xxx.mp4",
});
```

- **crop**

  裁剪视频元素,需要传入舞台相关参数。

  - **输入参数**

参数	描述	类型	必填
trackItem	轨道元素	object	是
options	裁剪区域中心点位置，裁剪大小	object: { x:number , y:number, width:number, height:number }	是

  - **返回值**

    携带裁剪操作数据的轨道元素对象，完整数据结构请参见 轨道元素 。

```
type result = {
id: string;
asset_id: string;
// ....
operation: any[];
};
```

- **示例**

```
/**
 * 裁剪的必须是视频元素
 *  @createTrackItem 创建轨道元素协议数据。
 *  @param asset_id {{string}} 媒资ID
 *  @param options {{object}} 可选参数
 * */
YJPlayer.Helper.CMEUtils.createTrackItem(["materialId"], {})
  .then((data) => {
    YJPlayer.Helper.TrackItem.crop(data, {
      crop: {
        x: 20,
        y: 20,
        width: 100,
        height: 100,
      },
      stage,
    });
    console.log("裁剪完成", data);
  })
  .catch((err) => {
    console.log("数据拉取失败", err);
  });
```

典型案例

多个视频首尾衔接

```
/**
 * 传入轨道元素数组，返回一段将传入轨道元素内容首位衔接的完整轨道数据。主要有以下步骤：
 * 1. 创建一个轨道。
 * 2. 元素添加到轨道上，默认是首尾衔接填充轨道内容。
 * 3. 得到一个可以播放的视频合成协议。
 */
async function demo1() {
  const trackItems = await YJPlayer.Helper.CMEUtils.createTrackItem([
    "materialId1",
    "materialId2",
    "materialId3",
  ]);
  const track = YJPlayer.Helper.Track.create({
    type: "video",
```

```
  });
  trackItems.forEach((element) => {
    YJPlayer.Helper.Track.append(track, element);
  });
  console.log("track:", track);
  const resultData = [track];
  console.log(resultData);
  return resultData;
}
```

裁剪视频

```
/**
 * 裁剪视频步骤：
 * 1. 创建一个轨道。
 * 2. 得到一个视频/图片轨道元素。对其使用裁剪方法。
 * 3. 裁剪后的元素添加到轨道内。
 * 4. 得到一个可以播放的视频合成协议。
 */
async function demo2() {
  const trackItems = await YJPlayer.Helper.CMEUtils.createTrackItem([
    "materialId1",
    "materialId2",
    "materialId3",
  ]);
  const track = YJPlayer.Helper.Track.create({
    type: "video",
  });
  /**
   * 在宽高为960,540的舞台上
   * 从视频的左上角位置(100,100)
   * 截取宽50 ,高50的画面
   */
  const cropedElement = YJPlayer.Helper.TrackItem.crop(trackItems[0], {
    rect: {
      x: 100,
      y: 100,
      width: 50,
      height: 50,
    },
    stage: {
      width: 960,
      height: 540,
```

```
    },
  });
  YJPlayer.Helper.Track.append(track, cropedElement);
  const resultData = [track];
  console.log(resultData);
  return resultData;
}
```

使用公共滤镜

```
/**
 * 对一个已有的视频/图片轨道元素应用一个lut滤镜素材:
 * 1. 通过素材ID创建视频/图片元素
 * 2. 应用lut滤镜效果
 * 3. 创建轨道，并装填轨道
 * 4. 得到一个可预览协议
 **/
async function demo0() {
  const trackItems = await YJPlayer.Helper.CMEUtils.createTrackItem([
    "materialId1",
    "materialId2",
    "materialId3",
  ]);
  for (const i = 0; i < trackItems.length; ++i) {
    /**
     * 胶片滤镜应用，胶片滤镜ID 为 '5fd329e702af8755735859a1@Public@CME'
     * 全部滤镜查看请到滤镜部分查看
     ***/
    await YJPlayer.Helper.TrackItem.lut(
      trackItems[i],
      "5fd329e702af87557358599f@Public@CME"
    );
  }

  /*
    设置滤镜强度，通过 filter_strength 字段
    https://vs-cdn.tencent-cloud.com/sdk/yj-player-1.4.10.js 版本以上支持
    {
      "type": "image_filter_lut",
      "params": {
        "name": "yj_templ_lut_87",
        "image_url":
"https://1810000000.vod2.myqcloud.com/b64e98acvodcq1810000000/thumbnail/f
```

```
ilter/yj_templ_lut_87.png",
        "filter_strength": 100
      }
    }
  */

  const track = YJPlayer.Helper.Track.create({
    type: "video",
  });
  trackItems.forEach((element) => {
    YJPlayer.Helper.Track.append(track, element);
  });
  console.log("track:", track);
  const resultData = [track];
  console.log(resultData);
  return resultData;
}
```

视频添加转场

```
/***
 * 添加视频转场：
 * 1. 获取一段已有的视频合成协议。
 * 2. 向其中添加一个视频转场。
 * 3. 得到一个已经处理好的视频合成协议。
 */
async function demo3() {
  /**
   * 先获取一段已有的合成数据。
   */
  const fusionData = demo1();
  /**
   * 创建一个公共的视频转场元素，这里使用的是九宫格效果。
   * 查阅转场元素涉及相关素材ID请到转场部分查看。
   *
   */
  const transition = await YJPlayer.Helper.CMEUtils.createTrackItem([
    "38192155374370925 2@Public@CME",
  ]);
  /**
   * 将转场添加到视频轨道的第一个视频片段后。
   */
  YJPlayer.Helper.Track.append(fusionData[0], transition[0], 1);
```

```
  return fusionData;
}
```

视频打马赛克

```
/***
 * 对合成视频部分打马赛克。
 * 1. 获得一个合成协议。
 * 2. 调用轨道场景内添加马赛克方法。
 * 3. 得到一个已经处理好的视频合成协议。
 */
async function demo3() {
  /**
   * 先获取一段已有的合成数据。
   */
  const fusionData = demo1();
  /**
   * 从视频时间线起始点打上马赛克。
   * 坐标位于(10,10)的位置
   * 持续时长为5s
   * 马赛克宽高为(200,200)
   */
  const result = YJPlayer.Helper.Track.addMosaic(
    fusionData,
    {
      start_time: 0,
      duration: 5000,
      degree: 50,
      x: 10,
      y: 10,
      width: 200,
      height: 200,
    },
    0 // 0为马赛克效果，1为高斯模糊效果
  );
  console.log(result);
  return result;
}
```

设置默认字幕

```
/***
```

```
 * 添加默认字幕：
 * 1. 获取一段已有的视频合成协议。
 * 2. 调用轨道场景内添加字幕方法。
 * 3. 得到一个已经处理好的视频合成协议。
 */
async function demo4() {
  const trackItems = await YJPlayer.Helper.CMEUtils.createTrackItem([
    "materialId1",
    "materialId2",
    "materialId3",
  ]);
  /**
   * 先获取一段已有的合成数据。
   */
  const fusionData = demo1(trackItems);

  /***
   *
   * 视频第一秒开始，字幕持续1秒，字幕内容是“这是字幕1”，
   * 以此类推
   */
  const subtitles = [
    {
      duration: 1000, //持续时间 ，单位ms
      text: "这是字幕1",
      start_time: 1000, //开始时间点.单位ms
    },
    {
      duration: 1000,
      text: "这是字幕2",
      start_time: 3000,
    },
  ];
  const result = YJPlayer.Helper.Track.addSubtitle(fusionData,
subtitles);
  console.log(result);
  return result;
}
```

序列帧-贴片/序列帧-文字/序列帧-特效

```
/**
 * 1. 创建一个对应类型轨道
```

```
 * 2. 通过指定素材ID创建轨道元素，【注意】此功能对象不可进行序列化，对象中包含BLOB类型数据
 * 3. 装填轨道
 * 4. 得到一个可预览协议
 **/
/**
 * 序列帧-贴片
 */
async function demo9() {
  // image-sequence
  const track = YJPlayer.Helper.Track.create({
    type: "image",
  });

  const trackItem = await YJPlayer.Helper.TrackItem.createSequence(
    "608a7497c1af830001c9292e@Public@CME", //素材ID
    yjPlayer // yjPlayer 实例，参考【播放器】部分
    // 可选参数
    // {
    //   width?: number
    //   height?: number
    //   position?: {
    //     x: number
    //     y: number
    //   }
    //   start_time?: number
    //   duration?: number
    // }
  );

  YJPlayer.Helper.Track.append(track, trackItem);
  const resultData = [track];
  return resultData;
}

/**
 * 序列帧-文字
 */
async function demo10() {
  // title-sequence_title
  const track = YJPlayer.Helper.Track.create({
    type: "title",
  });
  const trackItem = await YJPlayer.Helper.TrackItem.createSequence(
```

```
    "608bc4d689ea7200013ff244@Public@CME", //素材ID
    yjPlayer //yjPlayer的实例
    // {
    //   width?: number
    //   height?: number
    //   position?: {
    //     x: number
    //     y: number
    //   }
    //   start_time?: number
    //   duration?: number
    //   content?: 文字可用的配置项，参考 advanced_title
    // }
  );
  YJPlayer.Helper.Track.append(track, trackItem);
  const resultData = [track];
  return resultData;
}

/**
 * 序列帧-特效
 */
async function demo11() {
  // frame-sequence_effect
  // subType: shader

  const track = YJPlayer.Helper.Track.create({
    type: "shader",
  });
  const trackItem = await YJPlayer.Helper.TrackItem.createSequence(
    "60ebfe3852840e0001b4836c@Public@CME", //素材ID
    yjPlayer //yjPlayer的实例
    // {
    //   start_time?: number
    //   duration?: number
    // }
  );
  YJPlayer.Helper.Track.append(track, trackItem);
  const resultData = [track];
  return resultData;
}
```

事件通知
事件通知综述

最近更新时间：2023-07-12 14:23:22

在智能创作用户端或者使用服务端 API 进行视频上传、媒体移动、媒体删除等操作，都被称为一个事件。事件执行完成后，会立即通知 App 服务操作的执行结果，即事件通知。

回调方式

目前智能创作只支持 App 服务被动接收事件通知的模式。配置回调 URL 及通知事件类型后，智能创作会在事件完成后，向回调 URL 发起回调。
智能创作发起的回调的形式是 HTTP 请求，请求体为 JSON 格式，请求方法为 POST，内容包含 EventContent 结构。
以 **新文件产生事件通知** 为例，回调中的 EventType 参数为 `Storage.NewFileCreated`，事件内容为 StorageNewFileCreatedEvent 。完整的事件回调请求示例如下：

```
POST /callback HTTP1.1
Accept: */*
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Cache-Control: no-cache
Connection: close
Content-Length: 415
Content-Type: application/json
Host: api.example.com

{
    "EventType":"Storage.NewFileCreated",
    "Operator":"user_id_129883000303003329",
    "StorageNewFileCreatedEvent":{
        "FileId":"528589*****73533167",
        "MaterialId":"5fdafac******0001c82b2e",
        "OperationType":"Upload",
        "Owner":{
            "Id":"user_id_129883000303003329",
            "Type":"PESRON"
        },
        "ClassPath":"/媒资",
        "TaskId":"",
        "SourceContext":""
    }
```

```
}
```

事件类型

智能创作支持以下几种事件通知：

归类	事件通知	事件触发时机
媒体变更类	新文件产生	新文件产生事件，触发的场景包括：上传，在用户端主动上传媒体文件到媒资库拉取上传，用户在网盘使用媒体转拉功能上传媒体编辑合成直播流剪辑直播流录制
	媒体导入	将媒体导入到个人、团队或者项目中
	媒体添加	媒体添加事件，触发的场景有：将媒资库中已有的媒体文件添加到个人/团队媒资库，或者项目中，包括创建链接复制媒体（仅限复制媒体，复制目录导致的媒体增加不会产生该事件）
	媒体移动	移动媒体（仅限媒体移动，移动目录导致的媒体移动不会产生该事件）
	媒体修改	媒体修改，仅限修改媒体的名称，预置标签及自定义标签产生该事件
	媒体删除	媒体删除事件，触发的场景包括：删除媒资库中已有的媒体文件删除项目等导致的媒体文件删除
分类变更类	分类创建	新建分类
	分类移动	移动分类到另一个分类下
	分类删除	删除分类路径
项目状态变更类	云推项目状态变更	云转推项目状态变更，目前仅支持转推开始及转推结束通知
	导播台项目状态变更	导播台项目状态变更通知，触发的场景有：导播台启动导播台停止导播台 PVW 开启导播台 PGM 开启，输出推流开始导播台 PVW 停止导播台 PGM 停止，输出推流结束导播台被回收，需要恢复才能使用
视频导出类	视频导出完成	视频导出完成通知，触发的场景有：导出视频编辑项目、使用视频合成协议导出视频、使用视频剪辑模板导出视频、直播剪辑

开通事件回调

目前还无法自助开通事件回调。如需开通，请准备好事件通知地址 URL 及事件通知类型提 工单，由智能创作工作人员配置开通。

事件类型

最近更新时间：2022-09-08 18:19:05

本文档介绍智能创作目前支持的通知事件。

> **注意**
> 如业务配置了事件回调通知，用户操作触发了对应类型的事件后，回调接收服务会接收到来自智能创作的 HTTP 请求，请求采用 POST 方法，请求内容在 BODY 中。

媒体变更类事件

新文件产生事件

- **事件名称**：Storage.NewFileCreated
- **事件说明**：当业务配置了该类型的事件通知，如果在智能创作有新文件产生，业务后台即可收到该事件通知。事件通知内容为 StorageNewFileCreatedEvent 结构。事件内容如下所示（省略了值为 null 的字段）：

```
{
  "EventType":"Storage.NewFileCreated",
  "Operator":"user_id_12988300030300329",
  "StorageNewFileCreatedEvent":{
      "FileId":"528589*****73533167",
      "MaterialId":"5fdafac******0001c82b2e",
      "OperationType":"Upload",
      "Owner":{
          "Id":"user_id_12988300030300329",
          "Type":"PESRON"
      },
      "ClassPath":"/媒资",
      "TaskId":"",
      "SourceContext":""
  }
}
```

媒体导入事件

- **事件名称**：Material.Imported
- **事件说明**：当业务配置了该类型的事件通知，用户在智能创作导入媒体，业务后台即可收到该事件通知。事件通知内容为 MaterialImportedEvent 结构。事件内容如下所示（省略了值为 null 的字段）：

```
{
  "EventType":"Material.Imported",
  "Operator":"user_id_12988300030300329",
  "MaterialImportedEvent":{
      "MediaInfoSet":[
          {
              "FileId":"5285890*****3533169",
              "MaterialId":"5fdafaca******001c82b2e"
          }
      ],
      "Owner":{
          "Id":"user_id_12988300030300329",
          "Type":"PESRON"
      },
      "ClassPath":"/媒资"
  }
}
```

媒体添加事件

- **事件名称**：Material.Added
- **事件说明**：当业务配置了该类型的事件通知，用户在智能创作将媒体添加到个人目录、添加到项目中时，业务后台即可收到该事件通知。事件通知内容为 MaterialAddedEvent 结构。事件内容如下所示（省略了值为 null 的字段）：

```
{
  "EventType":"Material.Added",
  "Operator":"user_id_12988300030300329",
  "MaterialAddedEvent":{
      "MaterialIdSet":[
          "5fdafaca*******001c82b2e"
      ],
      "Owner":{
          "Id":"user_id_12988300030300329",
          "Type":"PESRON"
      },
      "ClassPath":"/媒资/视频"
  }
}
```

媒体移动事件

- **事件名称**：Material.Moved
- **事件说明**：当业务配置了该类型的事件通知，用户在智能创作将媒体移动到其它分类下，业务后台即可收到该事件通知。事件通知内容为 MaterialMovedEvent 结构。事件内容如下所示（省略了值为 null 的字段）：

```
{
  "EventType":"Material.Moved",
  "Operator":"user_id_12988300030300329",
  "MaterialMovedEvent":{
      "MaterialIdSet":[
          "5fdafac******d0001c82b2e"
      ],
      "SourceOwner":{
          "Id":"user_id_12988300030300329",
          "Type":"PESRON"
      },
      "SourceClassPath":"/媒资",
      "DestinationOwner":{
          "Id":"user_id_12988300030300330",
          "Type":"PESRON"
      },
      "DestinationClassPath":"/媒资/视频"
  }
}
```

媒体更新事件

- **事件名称**：Material.Modified
- **事件说明**：当业务配置了该类型的事件通知，用户在智能创作更新媒体名称、预置标签或者自定义标签时，业务后台即可收到该事件通知。事件通知内容为 MaterialModifiedEvent 结构。事件内容如下所示（省略了值为 null 的字段）：

```
{
  "EventType":"Material.Modified",
  "Operator":"user_id_12988300030300329",
  "MaterialModifiedEvent":{
      "MaterialId":"5fdafaca*******001c82b2e",
      "Name":"修改后的名称",
      "PresetTagIdSet":[

      ],
      "TagSet":[
          "风景",
```

```
            "城市"
        ]
    }
}
```

媒体删除事件

- **事件名称**：Material.Deleted
- **事件说明**：当业务配置了该类型的事件通知，用户在智能创作删除媒体时，业务后台即可收到该事件通知。事件通知内容为 MaterialDeletedEvent 结构。事件内容如下所示（省略了值为 null 的字段）：

```
{
  "EventType":"Material.Deleted",
  "Operator":"user_id_12988300030300329",
  "MaterialDeletedEvent":{
      "MaterialIdSet":["5fdafaca*******001c82b2e"]
  }
}
```

分类变更类事件

分类创建事件

- **事件名称**：Class.Created
- **事件说明**：当业务配置了该类型的事件通知，用户在智能创作创建分类时，业务后台即可收到该事件通知。事件通知内容为 ClassCreatedEvent 结构。事件内容如下所示（省略了值为 null 的字段）：

```
{
  "EventType":"Class.Created",
  "Operator":"user_id_12988300030300329",
  "ClassCreatedEvent":{
      "Owner":{
          "Id":"user_id_12988300030300329",
          "Type":"PERESON"
      },
      "ClassPath":"/媒体/测试"
  }
}
```

分类移动事件

- **事件名称**：Class.moved

- **事件说明**：当业务配置了该类型的事件通知，如果用户在智能创作移动分类时，业务后台即可收到该事件通知。事件通知内容为 ClassMovedEvent 结构。事件内容如下所示（省略了值为 null 的字段）：

```
{
  "EventType":"Class.moved",
  "Operator":"user_id_12988300030300329",
  "ClassMovedEvent":{
      "SourceOwner":{
          "Id":"user_id_12988300030300329",
          "Type":"PERESON"
      },
      "SourceClassPathSet":[
          "/媒体/视频/测试1",
          "/媒体/视频/测试2"
      ],
      "DestinationOwner":{
          "Id":"user_id_12988300030300329",
          "Type":"PERESON"
      },
      "DestinationClassPath":"/媒体/赛事"
  }
}
```

分类删除件

- **事件名称**：Class.Deleted
- **事件说明**：当业务配置了该类型的事件通知，用户在智能创作删除分类时，业务后台即可收到该事件通知。事件通知内容为 ClassDeletedEvent 结构。事件内容如下所示（省略了值为 null 的字段）：

```
{
  "EventType":"Class.Deleted",
  "Operator":"user_id_12988300030300329",
  "ClassDeletedEvent":{
      "Owner":{
          "Id":"user_id_12988300030300329",
          "Type":"PERESON"
      },
      "SourceClassPathSet":[
          "/媒体/视频/测试1",
          "/媒体/视频/测试2"
      ]
  }
```

```
}
```

项目状态变更类事件

云转推项目状态变更事件

- **事件名称**：Project.StreamConnect.StatusChanged
- **事件说明**：当业务配置了该类型的事件通知，云转推项目状态发生变更（包括开始转推及关闭转推）时，业务后台即可收到该事件通知。事件通知内容为 ProjectStreamConnectStatusChangedEvent 结构。事件内容如下所示（省略了值为 null 的字段）：

```
{
  "EventType":"Project.StreamConnect.StatusChanged",
  "Operator":"user_id_12988300030300329",
  "ProjectStreamConnectStatusChangedEvent":{
      "ProjectId":"cmepid_6fdafaca*******0001c82b23",
      "Status":"Working"
  }
}
```

导播台项目状态变更事件

- **事件名称**：Project.Switcher.StatusChanged
- **事件说明**：当业务配置了该类型的事件通知，导播台项目状态发生变更（包括启动导播台、关闭导播台等）时，业务后台即可收到该事件通知。事件通知内容为 ProjectSwitcherStatusChangedEvent 结构。事件内容如下所示（省略了值为 null 的字段）：

```
{
  "EventType":"Project.Switcher.StatusChanged",
  "Operator":"user_id_12988300030300329",
  "ProjectSwitcherStatusChangedEvent": {
      "ProjectId":"cmepid_78dbfacb*******022c89b46",
      "Status":"Started"
  }
}
```

视频导出完成事件

- **事件名称**：Task.VideoExportCompleted
- **事件说明**：当业务配置了该类型的事件通知，视频导出完成（包括导出视频编辑项目、使用视频合成协议导出视频等）时，业务后台即可收到该事件通知。事件通知内容为 VideoExportCompletedEvent 结构。事件内容如

下所示（省略了值为 null 的字段）：


```
{
  "EventType":"Task.VideoExportCompleted",
  "Operator":"user_id_12988300030300329",
  "VideoExportCompletedEvent":{
    "TaskId":"1*********bec5a75e20t",
    "Status":"SUCCESS",
    "ErrCode":0,
    "ErrMsg":"",
    "Output":{
      "MaterialId":"626101****01772235",
      "VodFileId":"38770****198482",
      "URL":"http://15000***/***/cec9182372270.mp4",

"CoverURL":"http://1500***/***4/coverBySnapshot/coverBySnapshot_10_0.jp
g",
      "MetaData":{
        "Size":5224677,
        "Container":"mov,mp4,m4a,3gp,3g2,mj2",
        "Bitrate":2112687,
        "Height":1080,
        "Width":1920,
        "Duration":19.784000396728516,
        "Rotate":0,
        "VideoStreamInfoSet":[
          {
            "Bitrate":2025016,
            "Height":1080,
            "Width":1920,
            "Codec":"h264",
            "Fps":25
          }
        ],
        "AudioStreamInfoSet":[
          {
            "Bitrate":83050,
            "SamplingRate":44100,
            "Codec":"aac"
          }
        ]
      }
    }
```

```
    }
}
```