数据工坊

最佳实践

文档版本	01
发布日期	2023-11-21

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

安全声明

漏洞声明

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该政策可参考华为公司官方网站的网址：https://www.huawei.com/cn/psirt/vul-response-process。
如企业客户须获取漏洞信息，请访问：https://securitybulletin.huawei.com/enterprise/cn/security-advisory。

目 录

1 发布下载 OBS 对象的算子

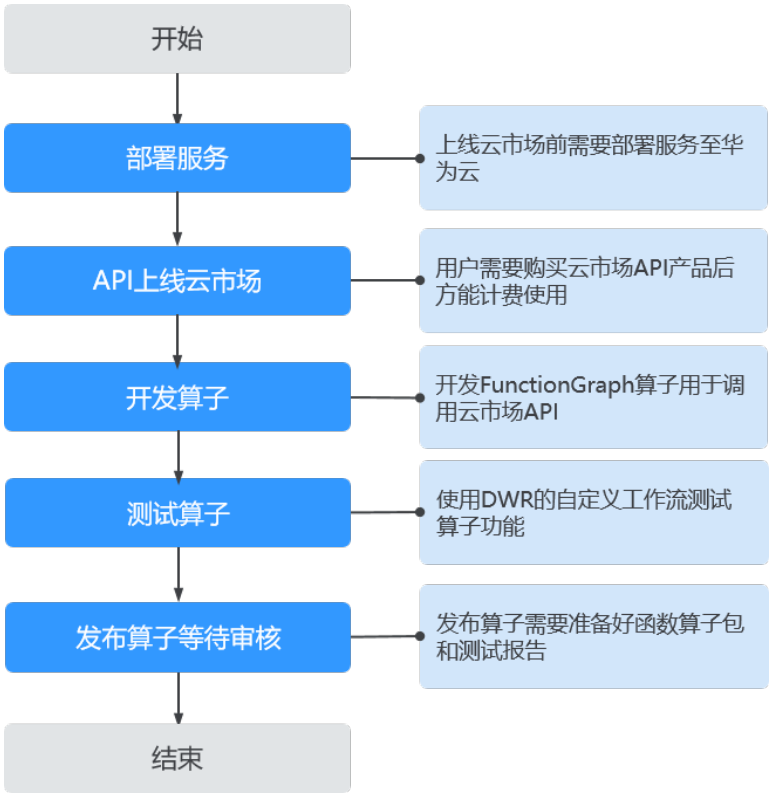
1.1 案例概述

场景介绍

DWR平台除了提供面向使用者的平台功能，同时也支持合作伙伴将自有功能包装成算子发布到DWR平台上，帮助伙伴快速搭建用户的数据处理工厂。

算子发布流程

图 1-1 算子发布流程图

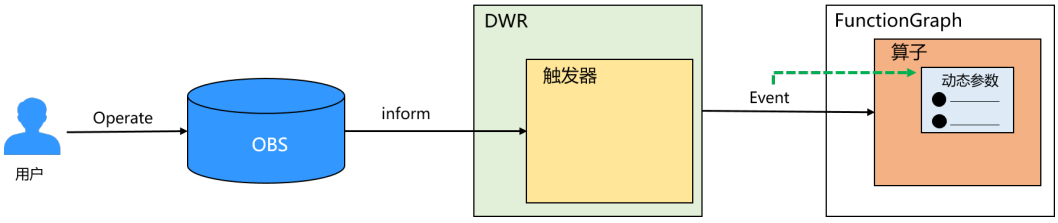


1. 部署服务：算子提供方需要将自身服务部署在华为云，同时对外提供API，方便用户进行调用。
2. API上线云市场：云市场提供API调用权限的购买入口，算子提供方将步骤1中开放的API注册到云市场中即可获取便捷的收费管理，上线指导参见**API上线云市场**。
3. 开发算子：DWR的算子执行依赖FunctionGraph服务提供执行引擎，算子提供方在发布算子前需按照**函数开发规范**开发算子。
4. 测试算子：算子提供方在发布前可通过自定义算子方式完成DWR的alpha调试，可有效降低发布后的故障率，测试结果也是算子发布时的重要审核项。
5. 发布算子等待审核：算子提供方完成步骤4中的测试后，进入到DWR控制台界面，根据提示信息提供必要的资料提交审核。

本次实践将重点描述步骤3、4、5完成算子的**开发、测试、发布**。

算子执行原理

图 1-2 算子执行原理



说明

- 动态参数：算子提供方定义的用户输入，最终由Event传递真实值。例如，云市场 appkey，算子回调函数地址。
- Event ： 算子的触发事件信息，例如OBS对象信息，桶名，动态参数。

1.2 开发算子代码

DWR算子开发运行依赖于FunctionGraph平台，本节将使用python开发一个下载OBS对象的算子，python使用详情参见**Python函数开发指南**。通过对该示例算子开发方式的说明，能够帮助算子开发者快速适应开发流程。通过本节将学到以下内容：

- 算子代码的入口结构
- 如何在算子中获得AK/SK
- 如何通过算子访问OBS

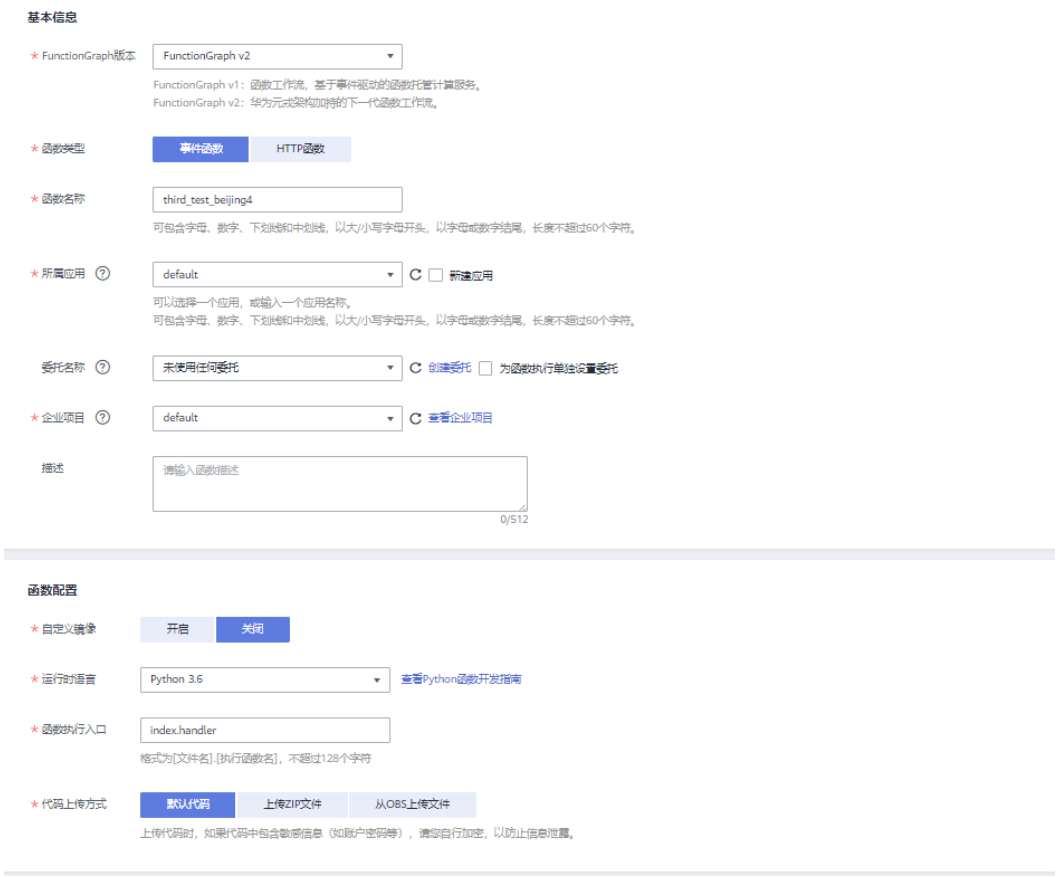
操作步骤

步骤1 登录FunctionGraph控制台创建python函数，创建详情参见**创建事件函数**。

⚠ 注意

1. 函数类型选择“事件函数”，该类型函数可以被事件触发，在DWR的使用过程中为OBS的事件，例如上传对象事件，下载对象事件等。
2. 代码上传方式选择“默认代码”。创建成功后FunctionGraph平台会自动生成一个python代码。

图 1-3 创建函数界面

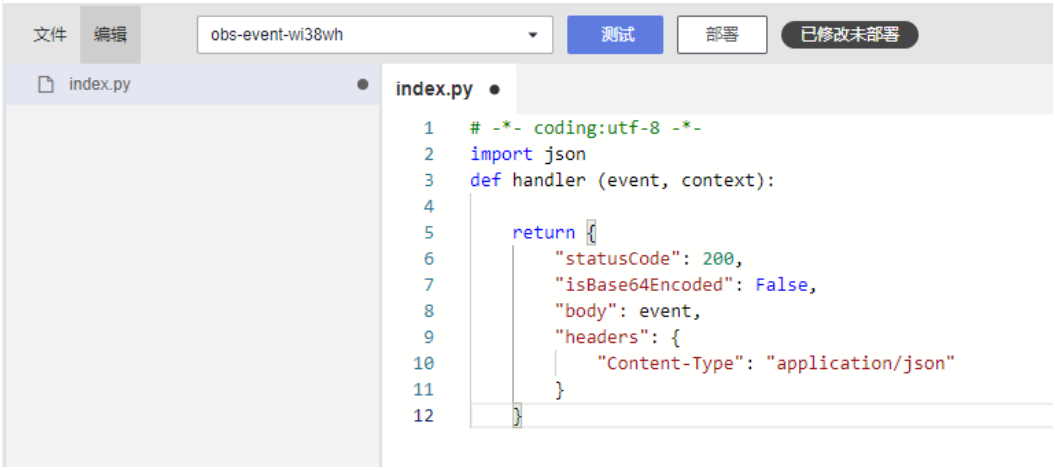


步骤2 编辑python代码。

如图1-4所示，默认代码中只有一个handler函数，入参为两个：

- event ： 触发事件信息，OBS事件触发的函数（DWR中称为算子）中event包含桶名，对象名。event也是自定义信息的承载体，DWR用户输入内容通过event中的动态参数（dynamic_source）保存。
- context ： 函数的上下文信息，例如函数执行时委托的AK/SK，Token等。更多内容可参考FunctionGraph开发文档。

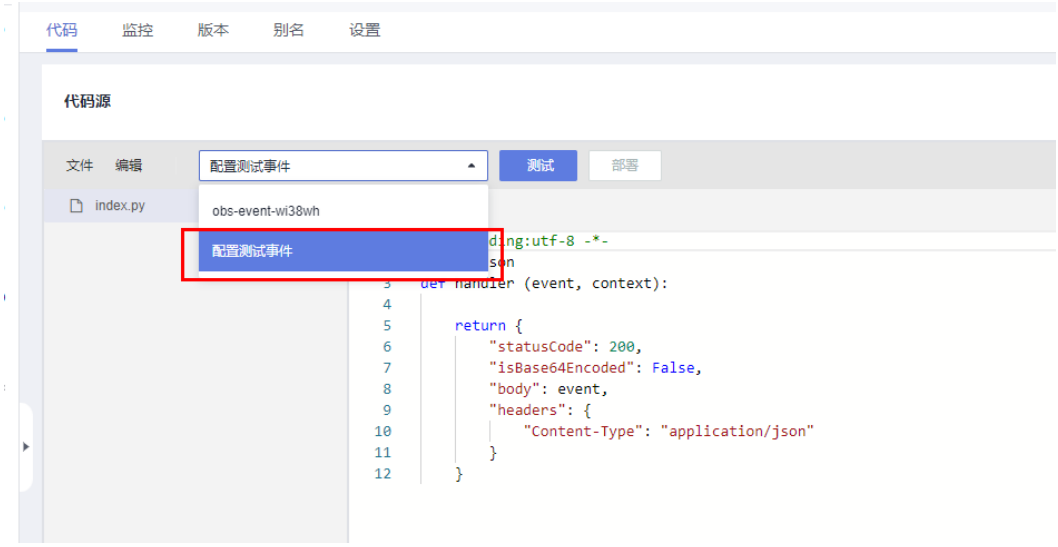
图 1-4 FunctionGraph 编辑代码界面



步骤3 测试代码

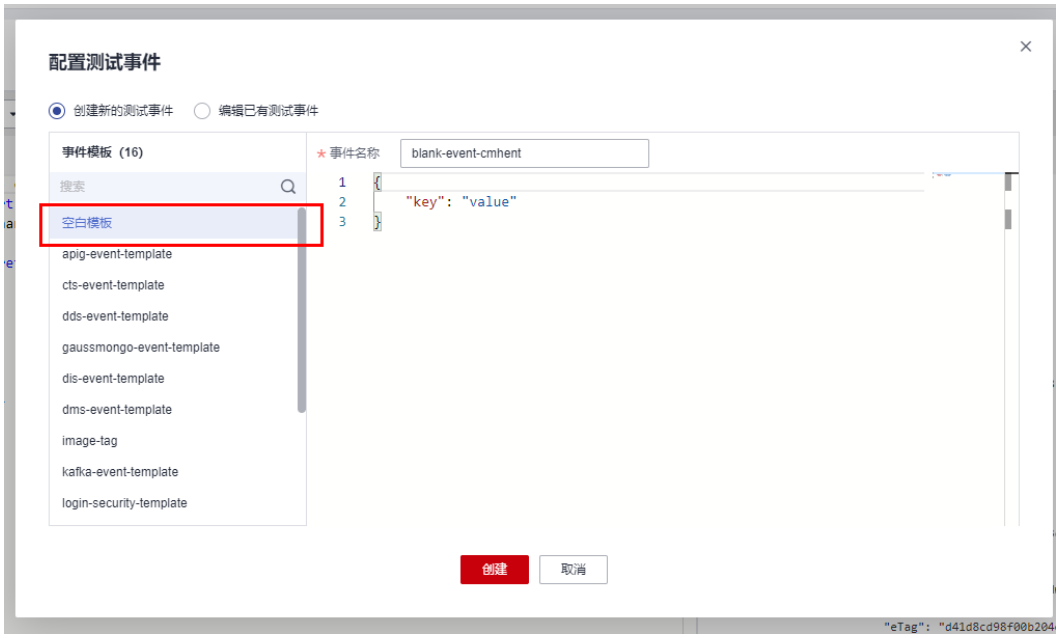
1. 测试代码需要配置测试事件，在下拉框中选择“配置测试事件”，进入配置测试事件界面。

图 1-5 配置测试事件



2. 选择“创建新的测试事件”并选择“空白模板”，将原有内容替换为以下事件(json格式)内容。

图 1-6 创建新的测试事件




```
{
  "execution_name": "84a3dd2bd67f43aa9b98cdd74604ca68",
  "graph_name": "test_workflow",
  "Records": [
    {
      "eventName": "ObjectCreated:Put",
      "eventRegion": "cn-north-4",
      "eventSource": "OBS",
      "eventTime": "2021-12-23T14:50:22.957Z",
      "eventVersion": "3.0",
      "obs": {
        "Version": "1.0",
        "bucket": {
          "bucket": "examplebucket",
          "name": "examplebucket",
          "ownerIdentity": {
            "ID": "08b4efe0fc00d3ce0f17c01b948f6e80"
          }
        },
        "configurationId": "test-trigger",
        "object": {
          "eTag": "fc85a07cff68977bf5b2108e7436ca2d",
          "key": "exampleobject.docx",
          "oldpsxpth": "",
          "sequencer": "1",
          "size": "524298",
          "versionId": "G001017DE60E176D0000401106696610null"
        }
      },
      "requestParameters": {
        "sourceIPAddress": "x.x.x.x"
      },
      "responseElements": {
        "x-obs-id-2": "",
        "x-obs-request-id": "84a3dd2bd67f43aa9b98cdd74604ca68"
      },
      "userIdentity": {
        "ID": "08b4efe0fc00d3ce0f17c01b948f6e80"
      }
    }
  ],
  "inputs": {
```

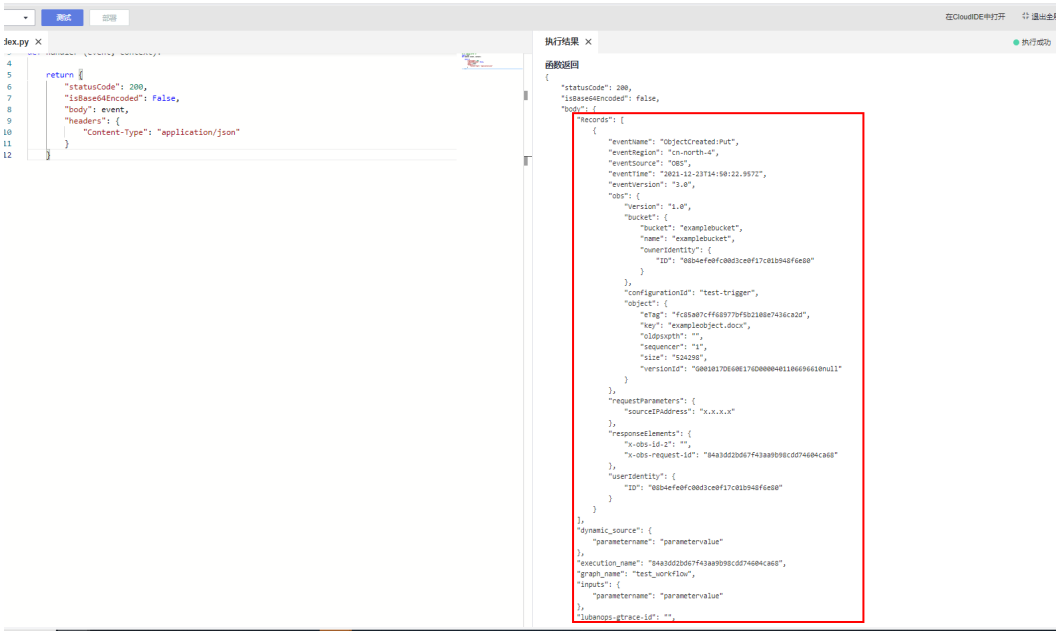


```
        "parametername": "parametervalue"
    },
    "dynamic_source": {
        "parametername": "parametervalue",
    "bucketname": "inputBucketName",
        "object": "inputObject"
    }
}
```

3. 点击“保存”后，即可点击“测试”按钮查看运行结果。

   可以在“执行结果”中看到保存的事件信息完整的打印在了参数body中，此函数已具备接收OBS事件的能力，可作为算子在DWR中运行。接下来我们将在其中添加“下载OBS对象功能”让函数功能更加丰富。

图 1-7 测试结果



步骤4 添加OBS下载逻辑（涉及AK/SK获取）

下载OBS的方式参考**下载对象**，示例采用**二进制下载**方式，主要代码如下所示。

如**图1-8**所示，执行结果中已经获取到目标对象的大小 size: 292685。

```
# -*- coding:utf-8 -*-
import json
# 引入模块
from obs import ObsClient
import os

def handler (event, context):
    # 获取AK/SK
    ak = context.getAccessKey()
    sk = context.getSecretKey()
    regionid = os.getenv('region')
    # dynamic_source是event中一个特殊结构体，通过它算子提供方可以获取用户在DWR中配置的参数
    bucketname = event["dynamic_source"]["bucketname"]
    objectName = event["dynamic_source"]["object"]
    print("ak : {}, sk :  {}".format(ak, sk)  )

    print("bucket name in dynamic_source is  : {}".format(bucketname)  )
    # 创建ObsClient实例
    obsClient = ObsClient(
        access_key_id=ak,
```



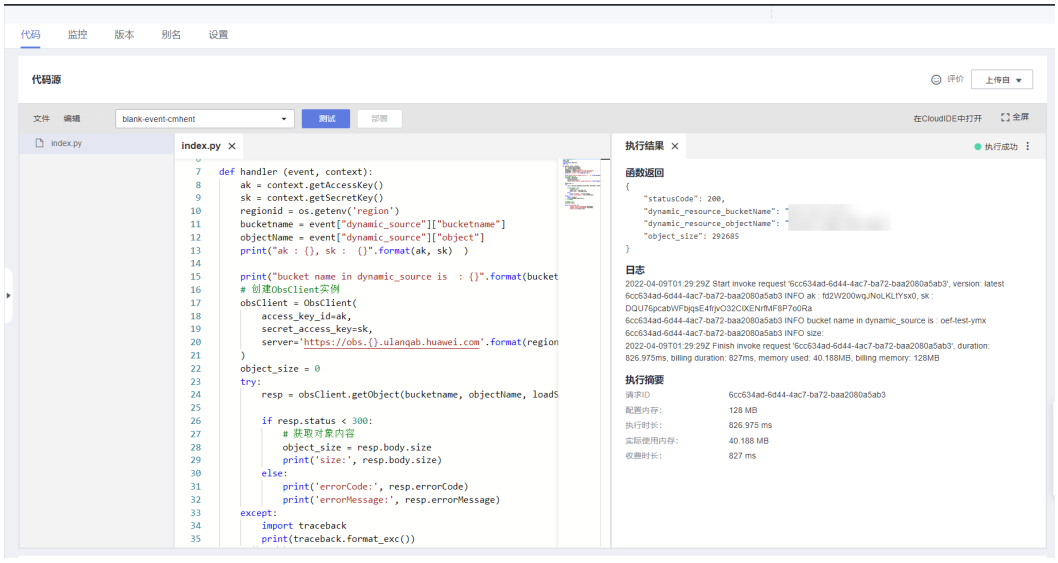
```
    secret_access_key=sk,
    server='https://obs.{}.huawei.com'.format(regionid)
)
object_size = 0
try:
    resp = obsClient.getObject(bucketname, objectName, loadStreamInMemory=True)

    if resp.status < 300:
        # 获取对象内容
        object_size = resp.body.size
        print('size:', resp.body.size)
    else:
        print('errorCode:', resp.errorCode)
        print('errorMessage:', resp.errorMessage)
except:
    import traceback
    print(traceback.format_exc())
# 使用访问OBS

# 关闭obsClient
obsClient.close()

return {"statusCode":200,
        "dynamic_resource_bucketName":bucketname,
        "dynamic_resource_objectName":objectName,
        "object_size":object_size}
```

图 1-8 OBS 下载对象流程执行结果



说明

1. AK/SK通过context 上下文对象保存，需要注意的是这里的AK/SK属于函数配置中的**委托**，故AK/SK的权限与该委托中的**授权范围相同**，如果需访问OBS，用户需要对委托进行授权并配置到FunctionGraph的函数配置项中。
2. 桶名以及对象名通过event进行获取。如图1-2所示，桶名、对象等信息保存在event中，代码中dynamic_source是event中一个特殊结构体，通过它算子提供方可以获取用户在DWR中配置的参数。这也是算子提供方获取用户配置参数的主要方式，后文将对该参数进行更加详细的介绍。bucketname以及object的值根据测试需要，点击函数测试参数配置。
3. obsclient形参中的server参数为OBS的endpoint，详情参见**地区和终端节点**。 不同的region拥有不同的endpoint，其主要区别在于regionid，其值通过配置环境变量完成。
4. 代码中最后一个return语句为必选代码块，工作流的执行过程中要求算子的返回结果为一个json体（对json中的内容无要求），否则将会发生执行异常。

----结束

1.3 测试算子功能

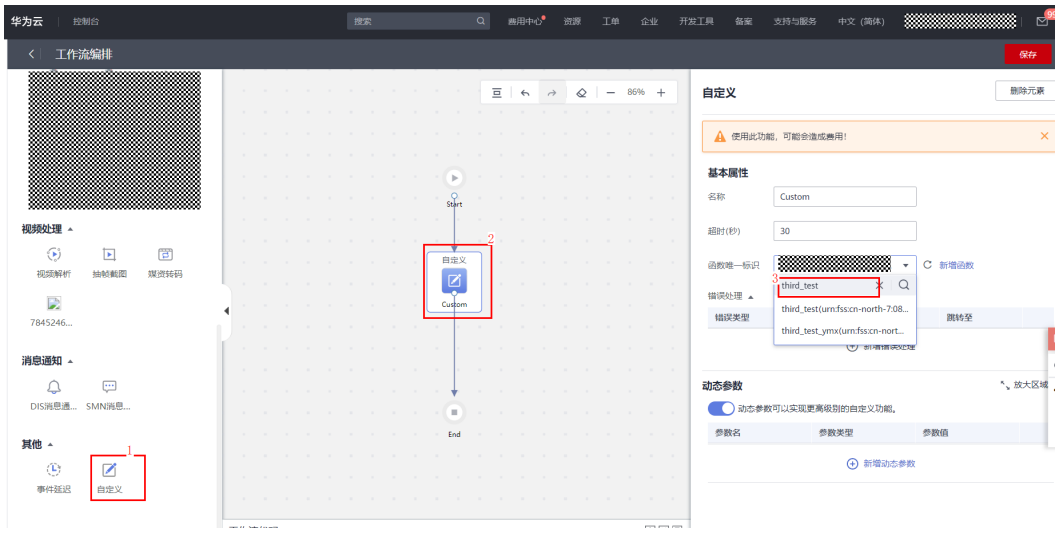
当算子功能代码开发完毕后，可以在DWR工作流中使用或进行上线前的测试。本节主要介绍如何使用DWR的自定义工作流来测试算子的功能。

测试算子功能

步骤1 登录DWR控制台， 在工作流页面点击“创建工作流”进入工作流编排界面。

1. 将左侧“自定义”算子拖拽至编排区域。
2. 鼠标单击各流程图标下方的小圆圈并长按拖拽，将工作流完整串联起来。
3. 在函数唯一标识中搜索上文中创建的测试算子。

图 1-9 编排自定义工作流



步骤2 配置DWR动态参数(dynamic_source)

上文算子测试时我们通过dynamic_source获取了两个参数值bucketnamel以及object。在工作流中，需要指明dynamic_source，并配置两个参数，否则算子将无法找到数值内容。点击“新增动态参数”，并添加动态参数，配置结果如图1-10所示。

配置完成后，点击**保存**按钮输入工作流名称，点击确认，成功保存后界面会自动跳转到工作流列表界面。

图 1-10 配置动态参数例图

步骤3 配置OBS桶触发器

根据图1-2所示，工作流的执行时机由触发器决定，当前DWR仅支持OBS的触发器

在工作流列表中找到刚才创建的工作流，点击**“创建事件触发器”**，如图1-11所示，在弹出窗口中根据输入项名称完成触发器的配置。

图 1-11 创建触发器

说明

- 前缀与后缀配合使用可以使触发器监听桶中不同目录中的内容，例如图1-11中触发器将监听testdir目录中以.png结尾的文件。
- 事件源触发器分两大类：
  - ObjectCreated：监听包括上传，更新，复制等操作。
  - ObjectRemoved：监听删除对象操作。
- 事件源存储桶：触发器监听桶，只有当其中对象内容发生变化时才能触发工作流执行。

步骤4 上传对象触发工作流执行

完成触发器配置后，工作流已经跟触发器关联，事件源存储桶中监听对象的变化将会触发工作流的执行，这一步我们将上传符合监听事件的对象。

在事件源存储桶中创建一个目录testdir 并上传.png结尾的文件至testdir目录下。

回到DWR工作流列表界面，点击**“运行记录”**查看工作流执行结果。

图 1-12 查看工作流运行结果

输出值与测试算子的输出结果相同，表示算子已经完成测试，具备发布条件。具备发布条更重要的是动态参数的存在让用户拥有了与算子交互的接口。

说明

动态参数中的桶名称、对象是由用户输入的参数，而触发器中的桶名称、对象则是面向触发器使用，二者不可混淆。

----结束

1.4 发布算子

当完成算子开发、算子在工作流中的测试后，即可在DWR上将算子发布到算子市场。

操作步骤

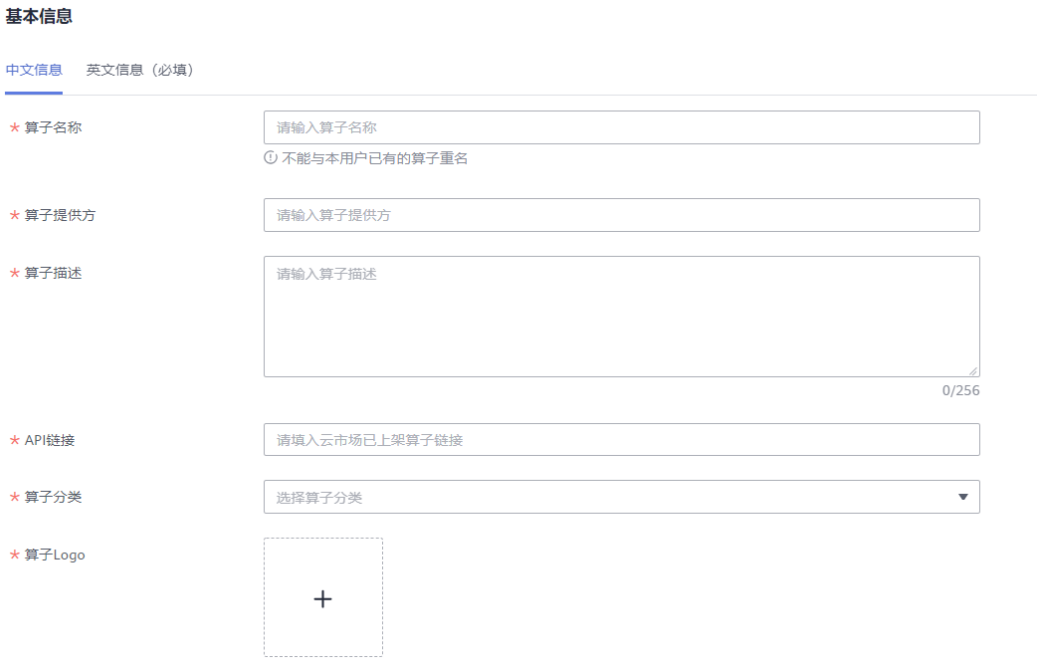
步骤1 在DWR控制台的“发布算子”页面，单击界面右上角的“发布公共算子”。

步骤2 配置算子基本信息。

表 1-1 配置基本信息

参数	说明
算子名称	不能与本用户已有的算子重名。
算子提供方	-
算子描述	-
API链接	填写华为云云市场已上架的算子链接。
算子分类	根据算子市场上提供的分类进行选择。
算子logo	支持主流图片格式

图 1-13 算子基本信息



步骤3 配置算子参数。

Inputs参数:

```
[
  {    //算子所在工作流输入列表
      "parameter_name": "bucketname",
      "parameter_value": "",
      "value_type": "",
      "default": "",
      "type": "string",
      "label": "Body",
      "constraints": {
          "regex": ".*"
      },
      "invisible": false,
      "description": "目标文件所在桶"
  },
  {
      "parameter_name": "object",
      "parameter_value": "",
      "value_type": "",
      "default": "",
      "type": "string",
      "label": "Body",
      "constraints": {
          "regex": ".*"
      },
      "invisible": false,
      "description": "下载文件名"
  }
]
```

动态参数（dynamic_source）:

```
{
  "bucketname": {
      "get_input": "$.inputs.bucketname"
  },
  "object": {
```



```
      "get_input": "$.inputs.object"
   }
}
```

权限1.1版本:

```
[
   {
      "action": [
         "obs:bucket:HeadBucket",
         "obs:bucket:ListBucketMultipartUploads",
         "obs:object:AbortMultipartUpload",
         "obs:object:PutObject",
         "obs:object:GetObject"
      ],
      "resource": []
   }
]
```

步骤4 单击右下角的“提交审核”。

审核通过后，算子将发布至算子市场。您可以过滤第三方的算子提供方，查看您发布的算子。

----结束

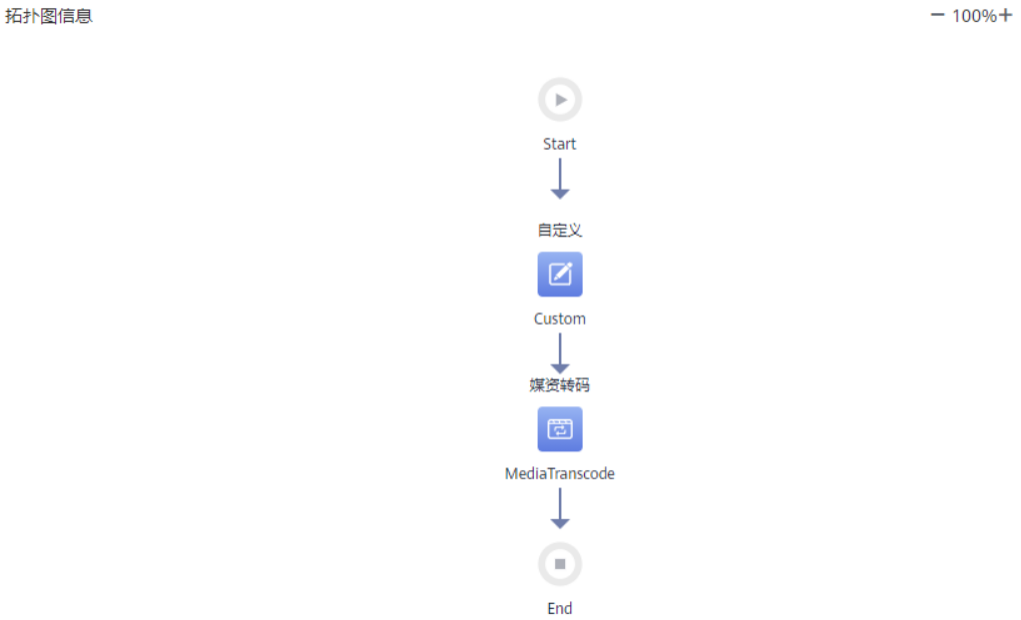
2 创建视频转码工作流

场景介绍

针对使用DWR做视频转码的场景，如果需要**每个对象转码任务的参数不同**，可通过给对象增加对象元数据（例如：x-obs-meta-transcode-commands: base64(commands)），再使用DWR集成自定义函数的方式来实现视频转码。

具体的工作流视图如图2-1所示。

图 2-1 工作流视图



资源和成本

表 2-1 资源和成本规划

资源	资源说明	数量	每月费用
OBS	算子请求OBS API。	1	通过算子对数据进行处理，都会涉及到对OBS API的调用，每调用一次API都计算一次请求次数。对象存储服务OBS会根据调用API的请求次数进行费用收取，收取详情参见**OBS请求费用说明**。
FunctionGraph函数	算子使用FunctionGraph函数工作流。	1	通过算子对数据进行处理，会使用到函数工作流的资源，比如算子执行时长，函数工作流会根据资源使用情况进行收费，收费详情参见**函数工作流计费说明**。
视频转码	新建转码任务可以将视频进行转码，并在转码过程中压制水印、视频截图等。视频转码前需要配置转码模板。 待转码的音视频需要存储在与媒体处理服务同区域的OBS桶中，且该OBS桶已授权。	1	由媒体处理服务MPC进行收费，详情查看**计费说明**。

操作步骤

步骤1 创建解析对象元数据并封装MPC转码任务的函数。

在FunctionGraph创建函数，选择“Python 2.7”运行时语言，并为函数配置具有访问OBS权限的委托，最后配置函数环境变量：region_id=cn-north-4。创建函数请参见**《函数工作流服务用户指南》**。

具体代码如下：

```
# -*- coding:utf-8 -*-
import urllib
import os
from obs import ObsClient  # Require public dependency:esdk_obs_python-3.x
import base64


class CreateTranscodingReq:
    def __init__(self, input=None, output=None, trans_template_id=None, av_parameters=None,
output_filenames=None,
                 user_data=None, watermarks=None, thumbnail=None, priority=None, subtitle=None,
encryption=None,
                 crop=None, audio_track=None, multi_audio=None, video_process=None, audio_process=None):
        if input is not None:
            self.input = input
        self.output = output
        if trans_template_id is not None:
            self.trans_template_id = trans_template_id
```



```
        if av_parameters is not None:
            self.av_parameters = av_parameters
        if output_filenames is not None:
            self.output_filenames = output_filenames
        if user_data is not None:
            self.user_data = user_data
        if watermarks is not None:
            self.watermarks = watermarks
        if thumbnail is not None:
            self.thumbnail = thumbnail
        if priority is not None:
            self.priority = priority
        if subtitle is not None:
            self.subtitle = subtitle
        if encryption is not None:
            self.encryption = encryption
        if crop is not None:
            self.crop = crop
        if audio_track is not None:
            self.audio_track = audio_track
        if multi_audio is not None:
            self.multi_audio = multi_audio
        if video_process is not None:
            self.video_process = video_process
        if audio_process is not None:
            self.audio_process = audio_process


def handler(event, context):
    # 获取上传桶、对象信息
    bucketName = event['Records'][0]['obs']['bucket']['name']
    objectKey = urllib.unquote(event['Records'][0]['obs']['object']['key'])  # a/b/c
    prefix = os.path.basename(objectKey)
    obsServer = 'obs.cn-north-7.ulanqab.huawei.com'
    # 使用obs sdk
    obsClient = newObsClient(context, obsServer)
    # 获取对象元数据
    resp = obsClient.getObjectMetadata(bucketName, objectKey)
    if resp.status < 300:
        print('headers: {}'.format(resp.header))
    else:
        print('errorCode: %s, errorMessage: %s', resp.errorCode, resp.errorMessage)
        return "ERROR"
    # 获取转码命令元数据
    command_base64 = dict(resp.header)["transcode-command"]
    print("command base64 {}".format(command_base64))
    if not command_base64:
        return "Error"
    # base64反解析
    command = base64.b64decode(command_base64)
    print("command :{}".format(command))
    # 解析转码命令
    command_map = trans(command)
    # 封装MPC转码任务参数
    av_parameters = []
    watermarks = []
    if "avthumb" in command_map:
        av_param = {}
        _type = 4
        if command_map["avthumb"] == "mp4":
            _type = 4
            av_param["video"] = {
                "output_policy": "transcode",
                "codec": 1,
                "bitrate": 40,
                "max_iframes_interval": 5,
            }
        elif command_map["avthumb"] == "mp3":
            _type = 5
```



```
            av_param["audio"] = {}
        av_param["common"] = {
            "pack_type": _type,
            "PVC": False,
            "hls_interval": 2,
            "dash_interval": 2,
        }
        av_parameters.append(av_param)
    # 文字水印参数封装
    if "wmText" in command_map:
        watermarks.append({
            "text_context": command_map["wmText"],
            "text_watermark": {
                "dx": command_map["wmOffsetX"],
                "dy": command_map["wmOffsetY"],
                "font_size": command_map["wmFontSize"],
            }
        })
    # 图片水印参数封装
    if "wmImage" in command_map:
        watermarks.append({
            "input": {
                "location": context.getUserData('region_id'),
                "bucket": bucketName,
                "object": objectKey
            },
            "image_watermark": {
                "dx": command_map["wmOffsetX"],
                "dy": command_map["wmOffsetY"],
            }
        })
    transcode_req = CreateTranscodingReq(input={
        "location": context.getUserData('region_id'),
        "bucket": bucketName,
        "object": objectKey
    }, output={
        "location": context.getUserData('region_id'),
        "bucket": bucketName,
        "object": prefix,
        "file_name": "demo.mp4"

    }, av_parameters=av_parameters,
        watermarks=watermarks)
    event["dynamic_source"] = {
        "transcodes": [
            transcode_req.__dict__
        ]
    }
    return event


def newObsClient(context, obsServer):
    ak = context.getAccessKey()
    sk = context.getSecretKey()
    return ObsClient(access_key_id=ak, secret_access_key=sk, server=obsServer)


def trans(command):
    commands = str(command).split("/")
    ret = {}
    key = ''
    for i in range(0, len(commands)):
        if i % 2 == 0:
            key = commands[i]
            continue
        ret[key] = commands[i]
    return ret
```

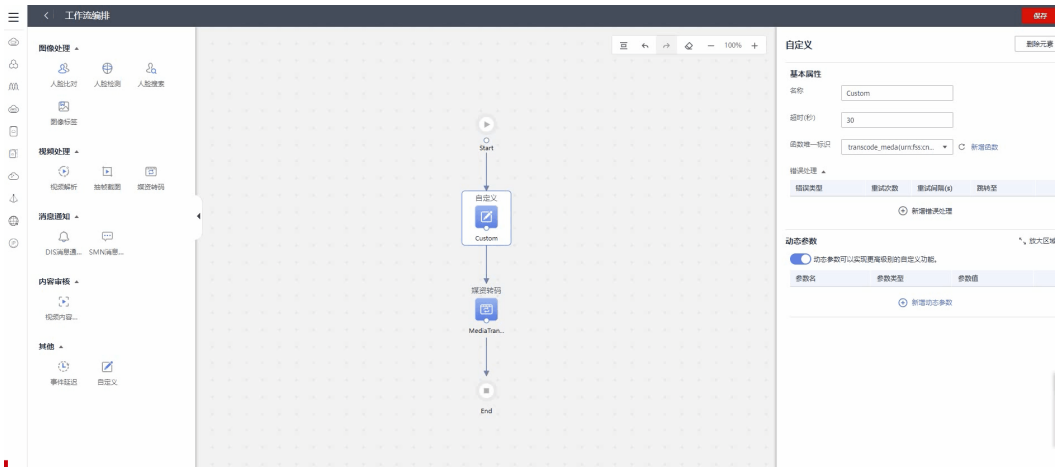
步骤2 创建工作流。

关联**步骤1**的自定义函数和“媒资转码”算子。拓扑图如**图2-2**所示。

须知

需要关闭自定义算子和“媒资转码”算子的动态参数开关。

图 2-2 创建工作流



步骤3 配置触发器。

触发器关联到需要做转码的桶，并根据业务需要指定对象前、后缀。

填写配置信息如下：

触发器名称：“test-trans”

关联工作流：“trans-teat-meta”

事件源存储桶：“dataplus-test”

事件源类型：“ObjectCreated”

前缀：“perfix”

后缀：“suffix”

步骤4 上传对象。

上传对象时带上自定义对象元数据，具体代码如下：

```
# -*- coding:utf-8 -*-
from obs import ObsClient
import base64

if __name__ == '__main__':
    # 上传后使用DWR做转码的参数
    command = "avthumb/mp4/wmText/dGVzdF90ZXh0/wmGravityText/SouthEast/wmFontSize/20/
wmOffsetX/10/wmOffsetY/38"
    client = ObsClient(
        access_key_id="XXX",
        secret_access_key="XXX",
        server="https://obs.cn-north-4.myhuaweicloud.com"
    )
    # 上传对象，并配置转码对象命令元数据
    resp = client.putObject("dataplus-test", "test.sh", "content", metadata={
```



```
    "x-obs-meta-transcode-command": base64.b64encode(command)
})
print resp
```

----结束

3 抽帧截图（官方算子）

3.1 方案概述

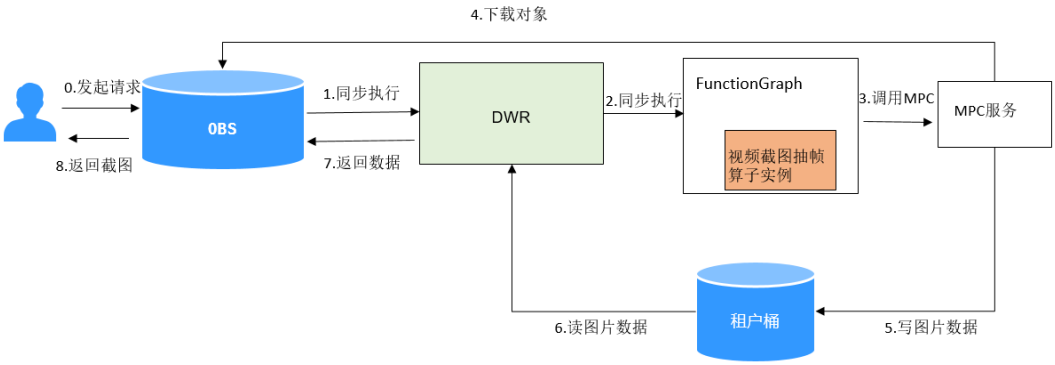
应用场景

需要对视频抽帧截图时，在不编写额外代码的情况下使用抽帧截图算子对视频进行指定时间点截帧。

约束与限制

该抽帧截图算子目前暂不支持中文对象。

方案架构



1. 用户使用对象域名向OBS发送请求
2. DWR收到请求同步执行位于FunctionGraph的视频截图抽帧算子实例，调用MPC服务
3. MPC服务完成抽帧截图，将结果写回租户桶
4. DWR读取租户桶并将结果返回给用户

方案优势

无需额外代码编写，快速构建视频抽帧截图应用。

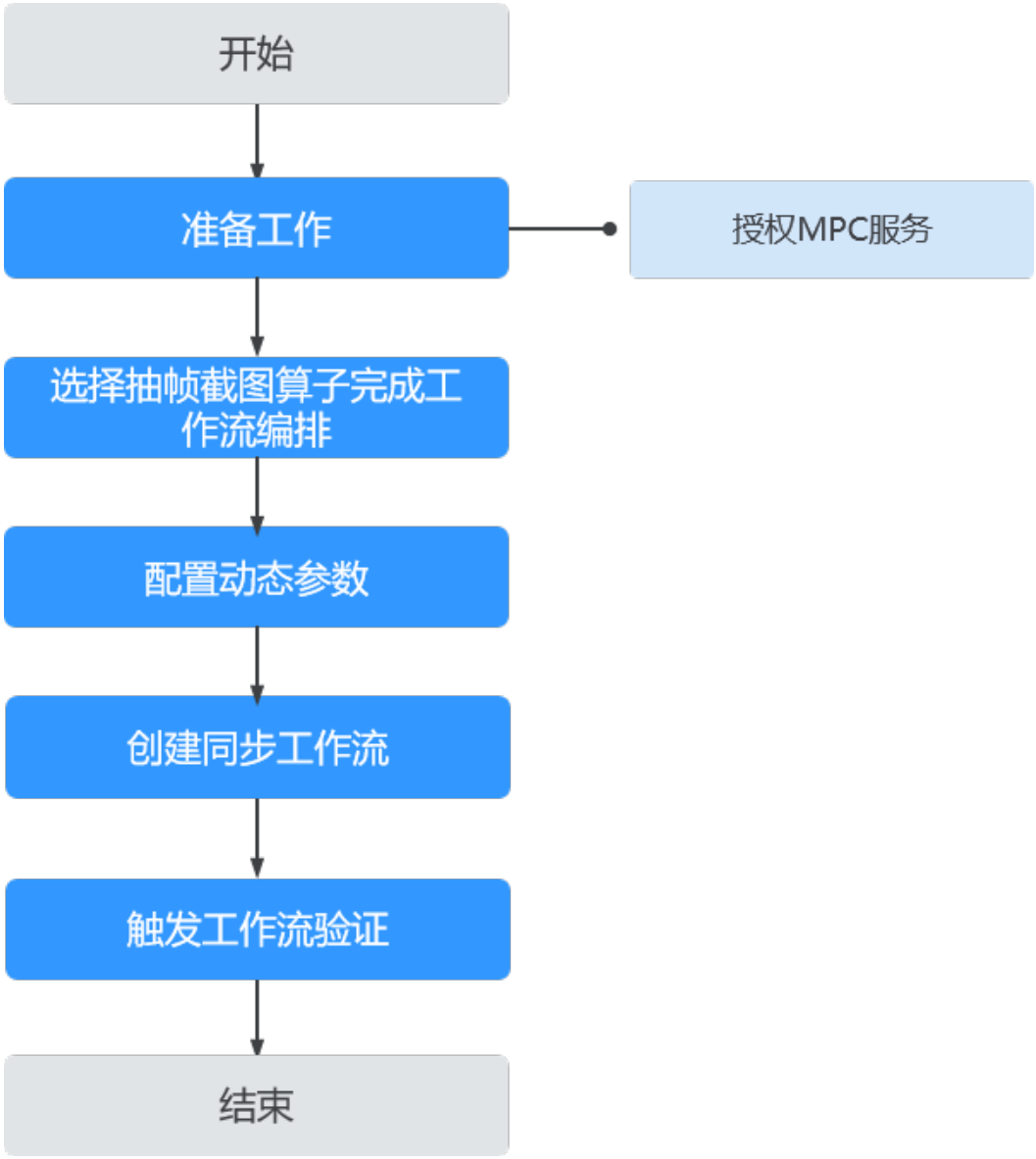
3.2 资源和成本

表 3-1 资源和成本规划

资源	资源说明	数量	每月费用
OBS	算子请求OBS API。	1	通过算子对数据进行处理，都会涉及到对OBS API的调用，每调用一次API都计算一次请求次数。对象存储服务OBS会根据调用API的请求次数进行费用收取，收取详情参见**OBS请求费用说明**。
FunctionGraph函数	算子使用FunctionGraph函数工作流。	1	通过算子对数据进行处理，会使用到函数工作流的资源，比如算子执行时长，函数工作流会根据资源使用情况进行收费，收费详情参见**函数工作流计费说明**。
抽帧截图算子	使用抽帧截图算子快速构建视频抽帧截图应用。	1	由媒体处理服务MPC进行收费，详情查看**计费说明**。

3.3 操作流程

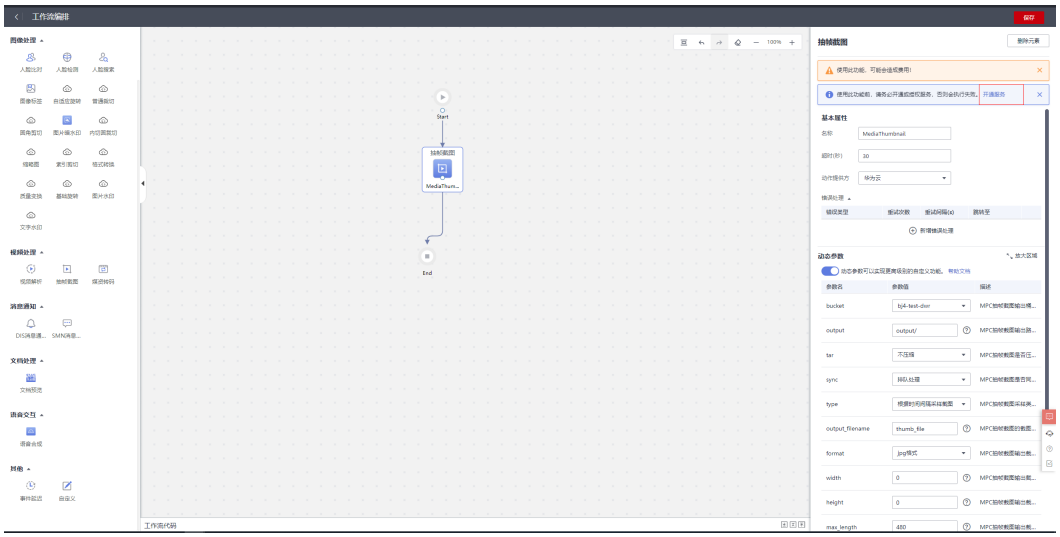
图 3-1 操作流程



3.4 实施步骤

准备工作

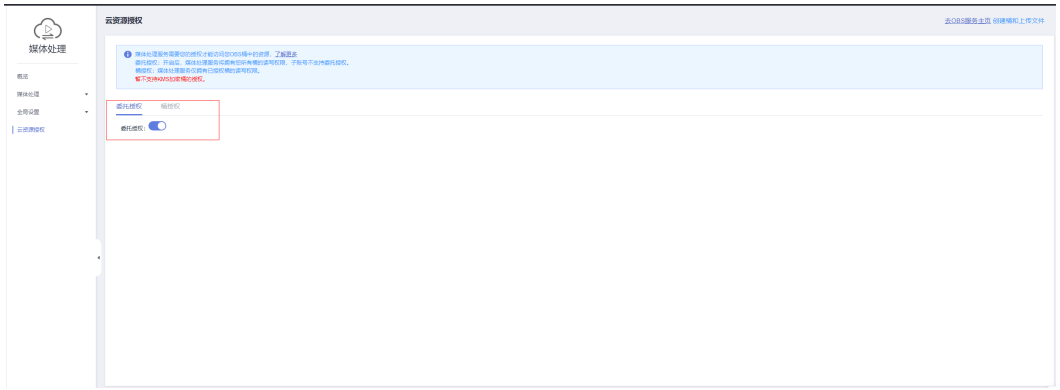
1. 点击开通mpc服务授权。



2. 选择打开**委托授权**或**桶授权**。

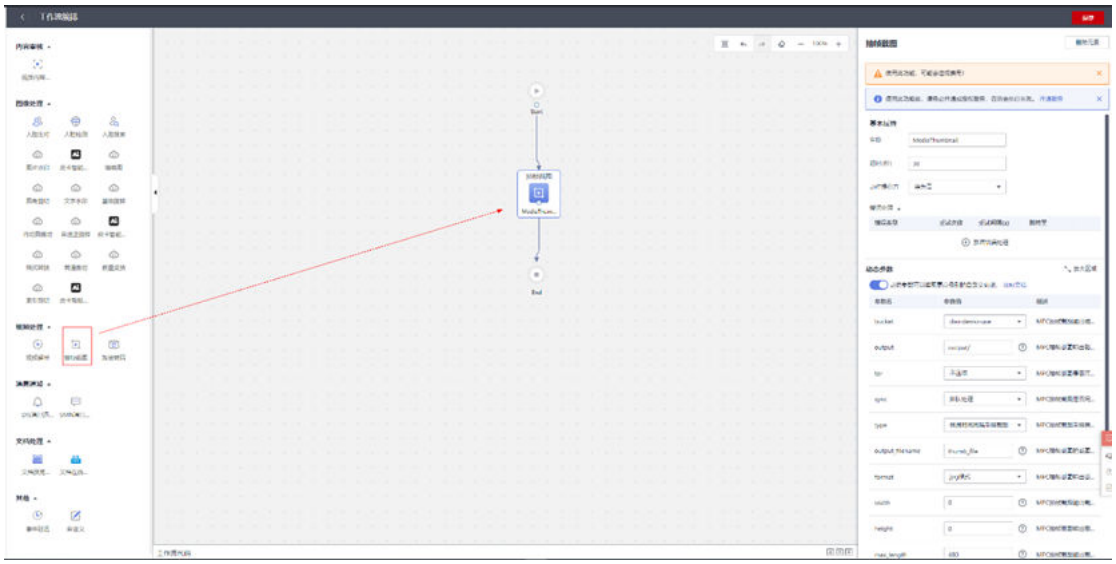
   委托授权：开启后，媒体处理服务将拥有您所有桶的读写权限，子账号不支持委托权限。

   桶授权：开启后，媒体处理服务仅拥有已授权桶的读写权限。

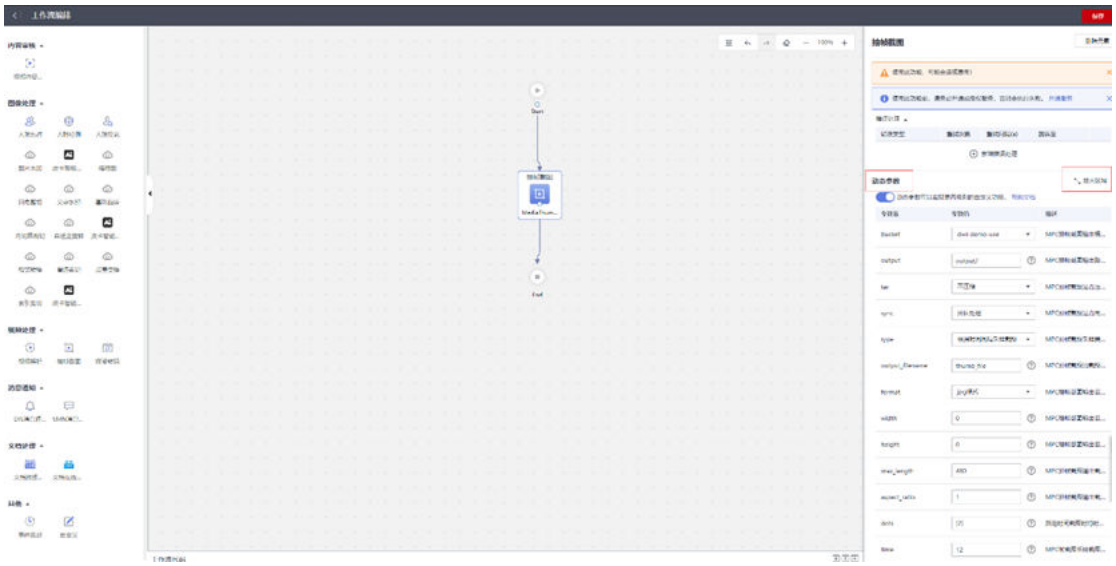


操作步骤

步骤1 登录DWR控制台，在“工作流”页面单击右上角“创建工作流”，进入创建工作流界面，选择抽帧截图算子进行工作流编排。



步骤2 点击右侧算子详情配置动态参数。



步骤3 按照指定时间点截图并输出最长边尺寸为480的棘突，参数修改如下：

bucket: 抽帧截图结果输出的桶名，如图所示填为test-workflow-sh1

output: 抽帧截图结果在输出桶下的输出路径如图中所示输出图片的路径为桶tes-workflow-sh1的output目录

tar: 抽帧截图图片是否需要压缩， 选择否

sync: 是否同步处理图片，目前只有按照时间点截图支持同步

type: 抽帧截图的采样类型，选择按照时间点截图

format:输出截图格式，目前只支持jpg格式

max_length: 截图最长边的尺寸，填入480

width：输出截图的宽度，填入0

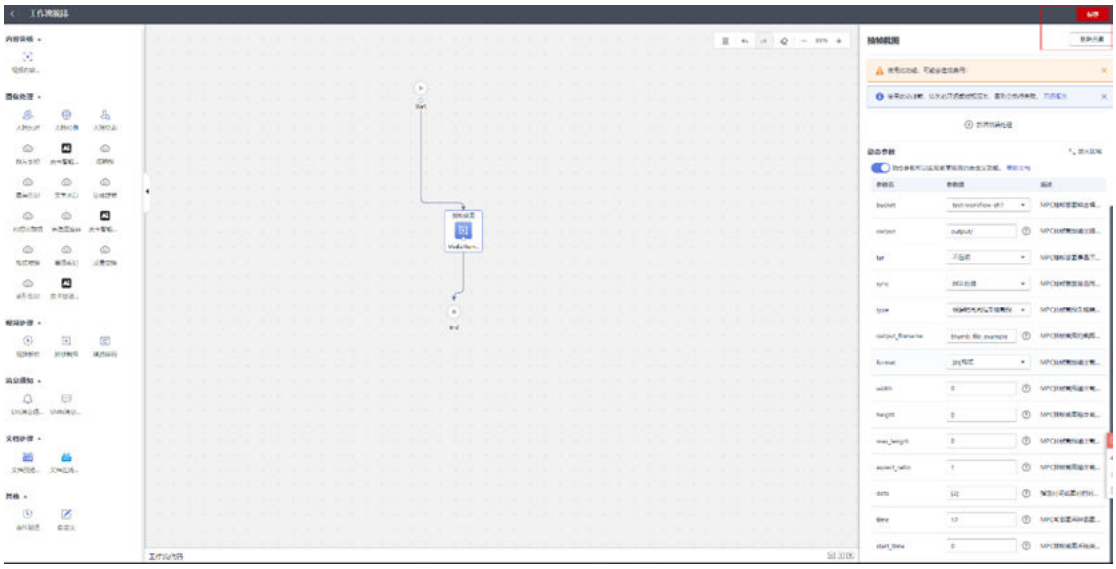
height：输出截图的高度，填入0

(当有Width和Height均不为0时，按照width和height限制输出图片的尺寸.

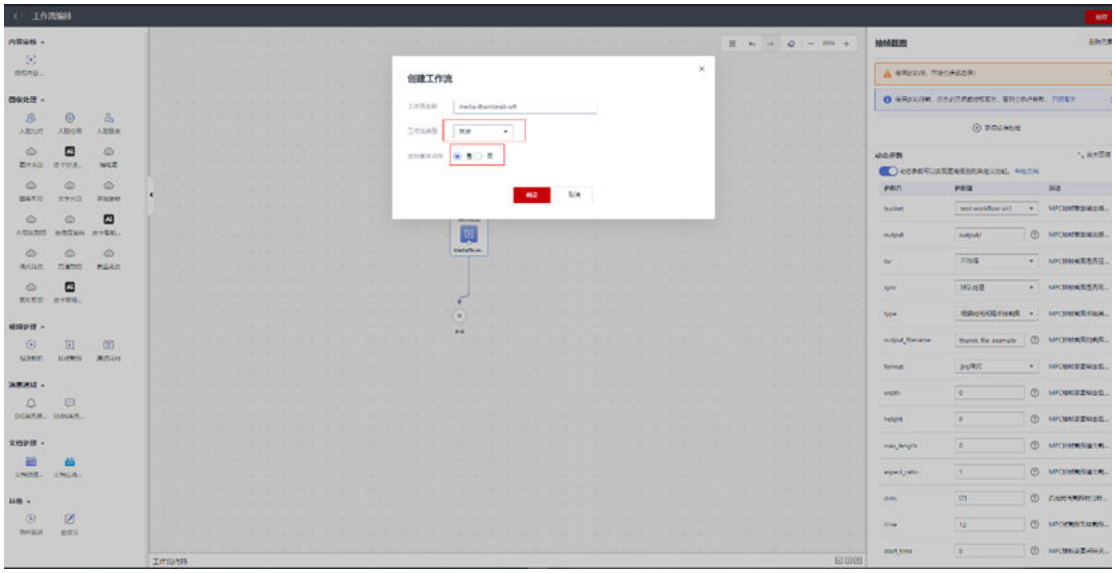
当width和height为0，Max_length不为0时，按照max_length计算得出图片尺寸)

dots: 指定时间截图时的时间点数组，填入[2]代表截取视频的第2秒。如果需要第0s首帧则填入[0]

步骤4 配置完成后，点击保存。



步骤5 选择创建同步工作流，可选择支持匿名访问。

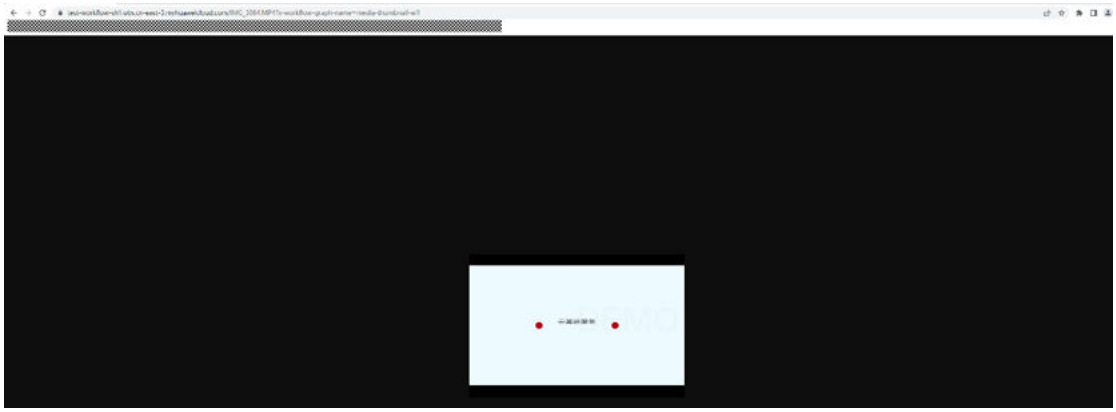


步骤6 创建成功，同步触发工作流。

请确保创建的工作流，触发的桶和获取结果写入的桶在同一个region

使用方式为: <对象url>?x-workflow-graph-name={工作流名称}

使用参数：x-workflow-graph-name后跟上工作流名称，即可同步触发工作流.如图所示为返回的截帧结果



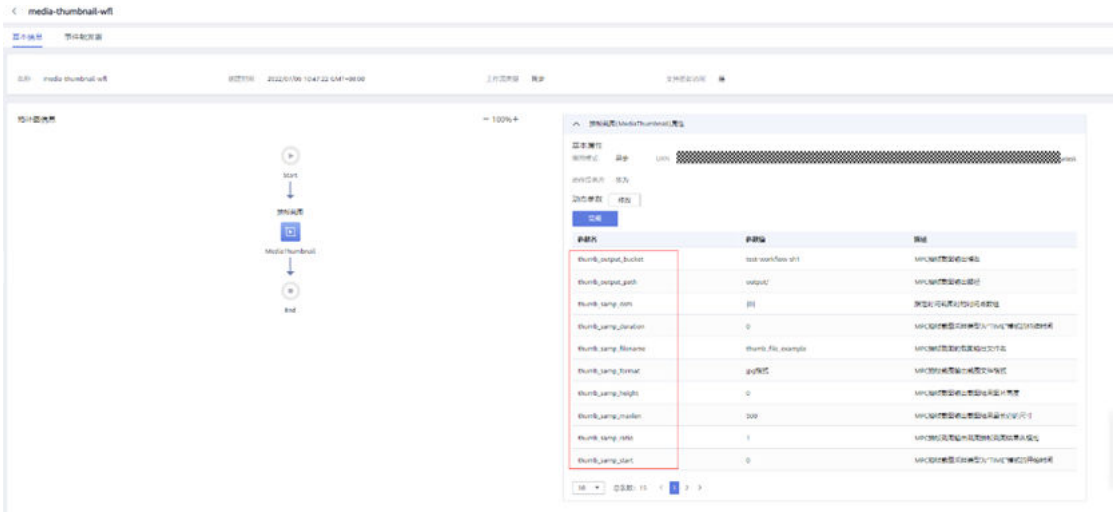
当需要对工作流参数做修改时，可以选择两种方式。以修改截图最长边的尺寸到500，采样点到11s为例：

1. 直接在url后对参数修改

   在工作流名称后使用’/’加上参数名，再使用’_’连接加上指定参数值．不通参数间使用’,’分隔

   <对象url>?x-workflow-graph-name={工作流名称}/<参数名>_{参数值}

   对应的参数名请点击工作流详情后查看:

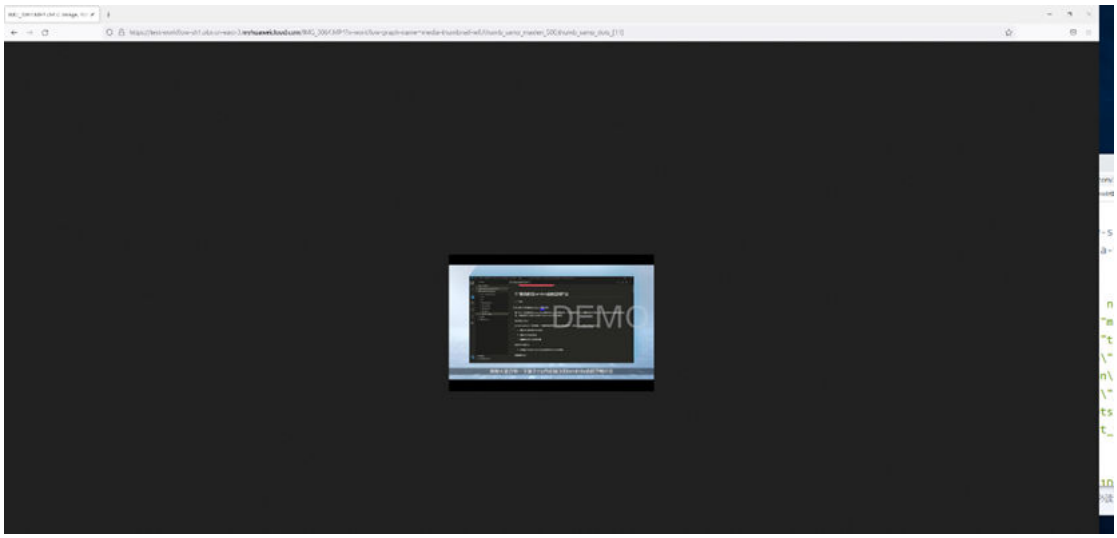


   触发工作流:

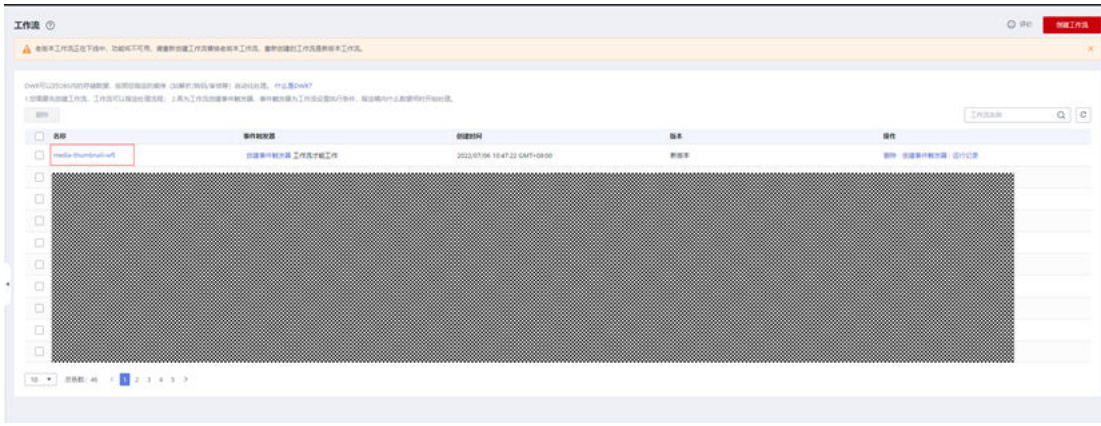
   https://test-workflow-sh1.obs.cn-east-3.myhuaweicloud.com/[redacted].MP4?x-workflow-graph-name=media-thumbnail-wfl/thumb%5Fsamp%5Fmaxlen_500,thumb%5Fsamp%5Fdots_[11]

   参数名

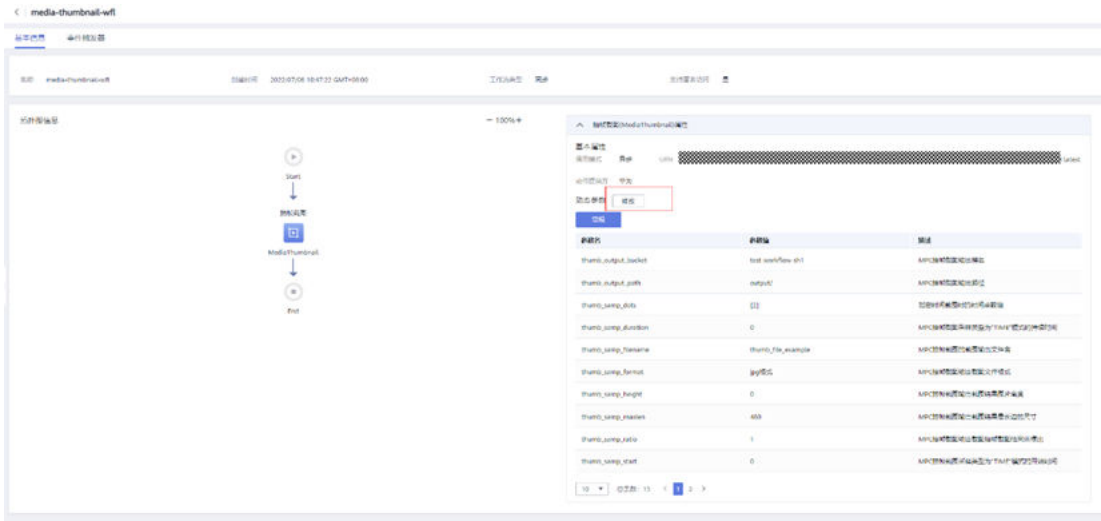
   注意:当参数名有下划线时，需将下划线转成%5F



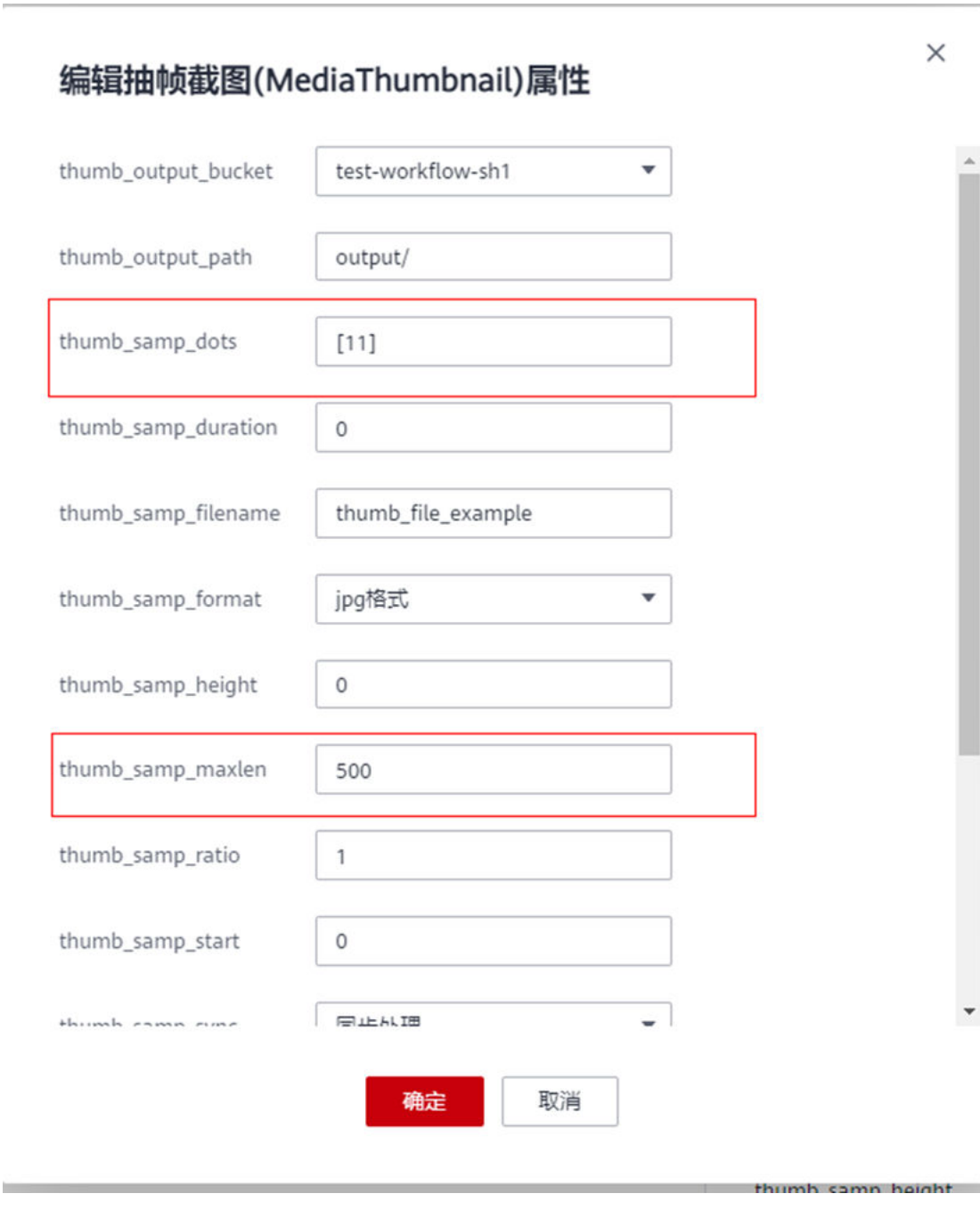
2. 直接在DWR工作流界面修改
   a. 进入DWR工作流界面并点击进入工作流详情



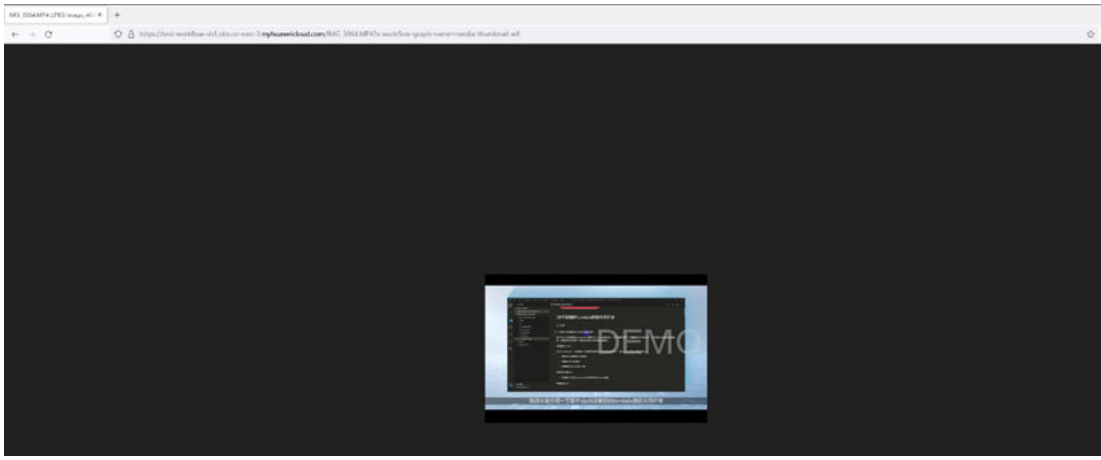
   b. 点击修改对参数进行修改



   修改thumb_samp_dots为[11]，thumb_samp_maxlen为500，点击确定。



再次同步触发工作流，获得结果。



----结束

4 抽帧截图（自定义算子）

4.1 方案概述

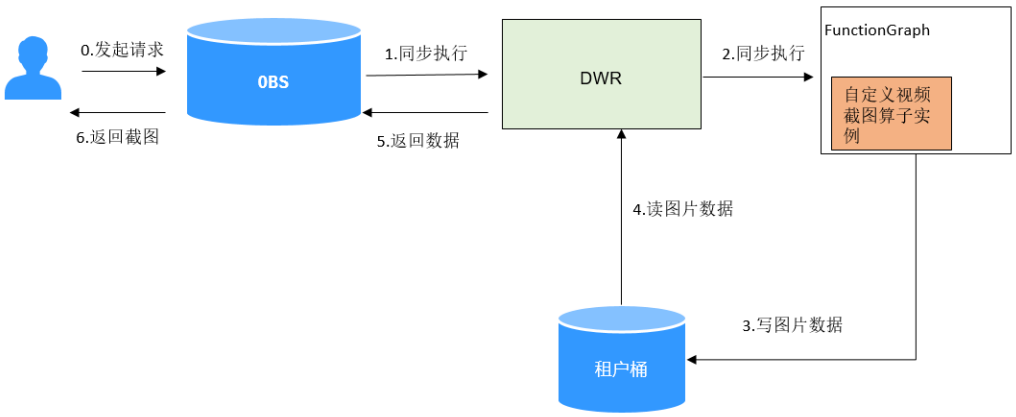
应用场景

需要对视频抽帧截图时，在不编写额外代码的情况下使用抽帧截图算子对视频进行指定时间点截帧。

约束与限制

该抽帧截图算子目前暂不支持中文对象。

方案架构



方案优势

用户自定义算子，无需依赖额外服务，功能更灵活。

4.2 资源和成本

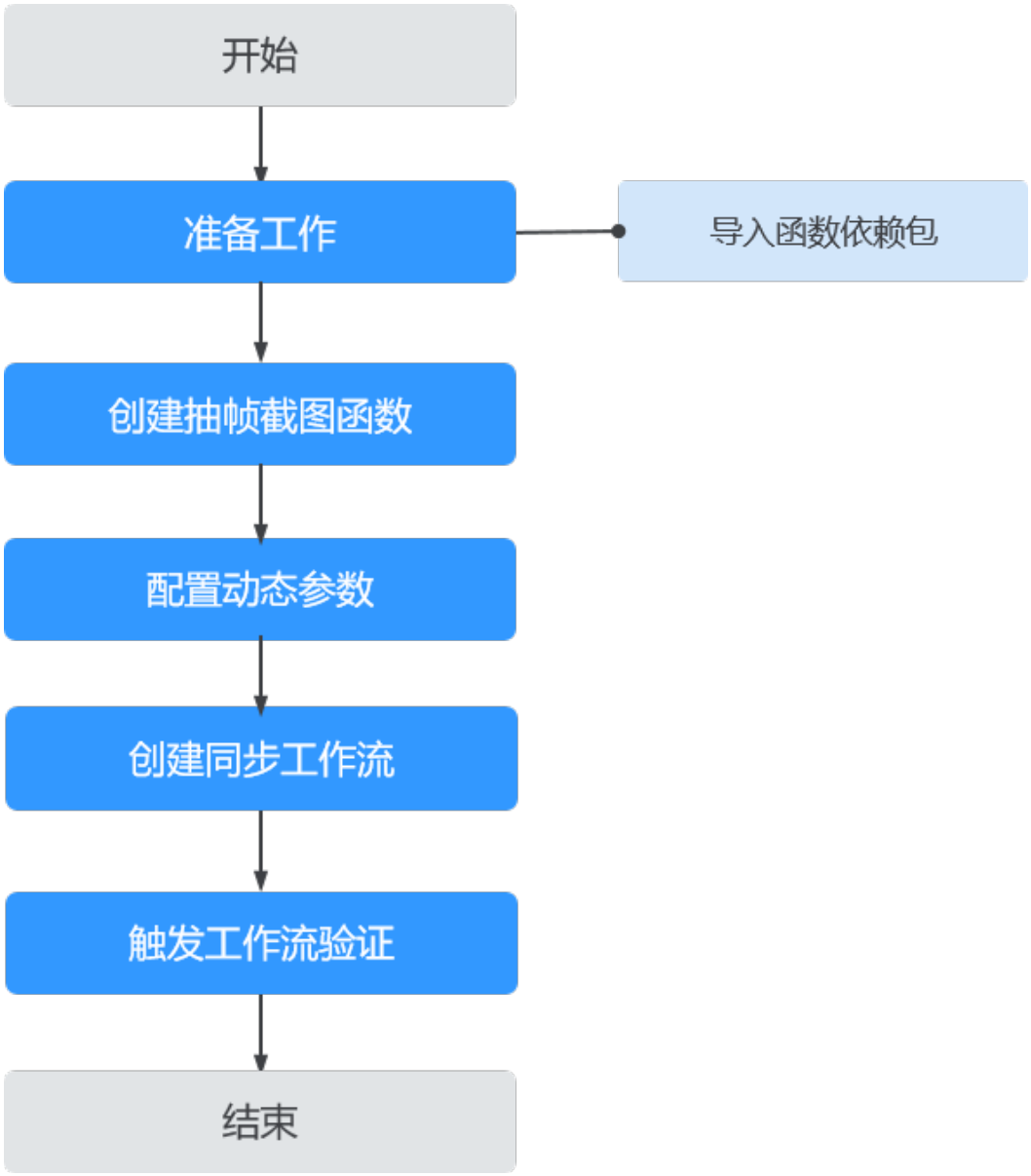
表 4-1 资源和成本规划

资源	资源说明	数量	每月费用
OBS	算子请求OBS API。	1	通过算子对数据进行处理，都会涉及到对OBS API的调用，每调用一次API都计算一次请求次数。对象存储服务OBS会根据调用API的请求次数进行费用收取，收取详情参见**OBS请求费用说明**。
FunctionGraph函数	算子使用FunctionGraph函数工作流。	1	通过算子对数据进行处理，可能会使用到函数工作流的资源，比如算子执行时长，函数工作流会根据资源使用情况进行收费，收费详情参见**函数工作流计费说明**。

4.3 操作流程

您首先需要在FunctionGraph导入自定义算子需要的函数依赖包。然后需要在FunctionGraph创建自定义抽帧截图函数。之后需要在DWR工作流编排界面完成配置动态参数，创建同步工作流。最后触发工作流进行验证。

图 4-1 操作流程



4.4 实施步骤

前提条件

步骤1 创建依赖包。

1. 登录FunctionGraph控制台，在依赖包管理页面点击“创建依赖包”。

2. 填写相关参数，完成依赖包创建。

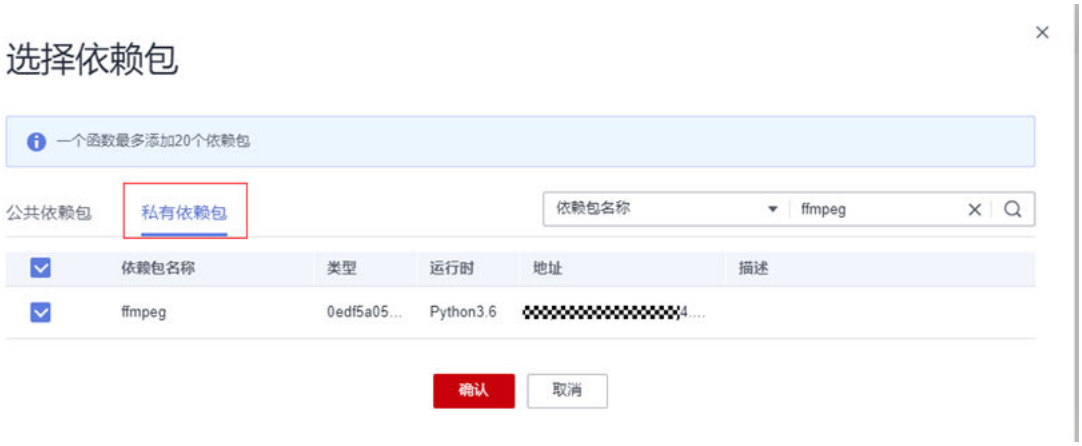
步骤2 在“函数列表”页面单击“创建函数”完成函数创建。

其中委托需要创建能够授权工作流FunctionGraph访问OBS的权限，创建方式参考**创建委托**。

步骤3 函数创建成功后进入函数详情页添加依赖包。

1. 在“代码依赖包”模块点击“添加依赖包”。

2. 选择私有依赖包，添加步骤**步骤1**中创建好的依赖包



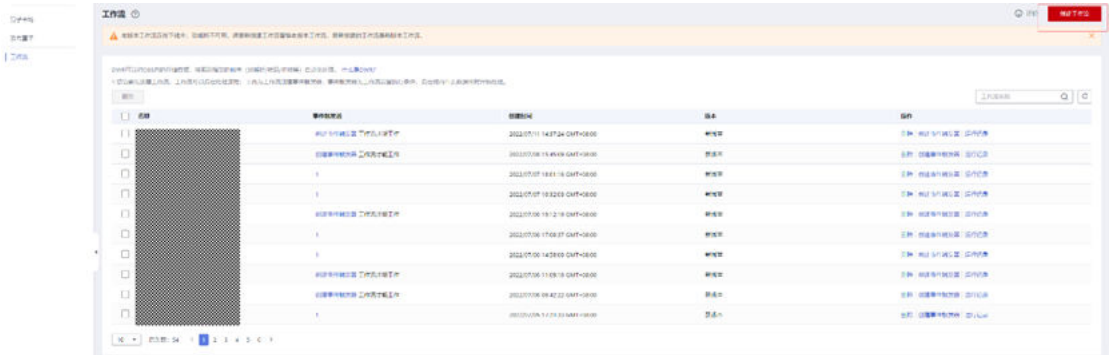
步骤4 依赖添加成功后，在FunctionGraph控制台添加算子代码并保存部署。参考示例如下：

```
# -*- coding:utf-8 -*-
import json
# 引入模块
from obs import ObsClient
import os
import time
import urllib.parse
```



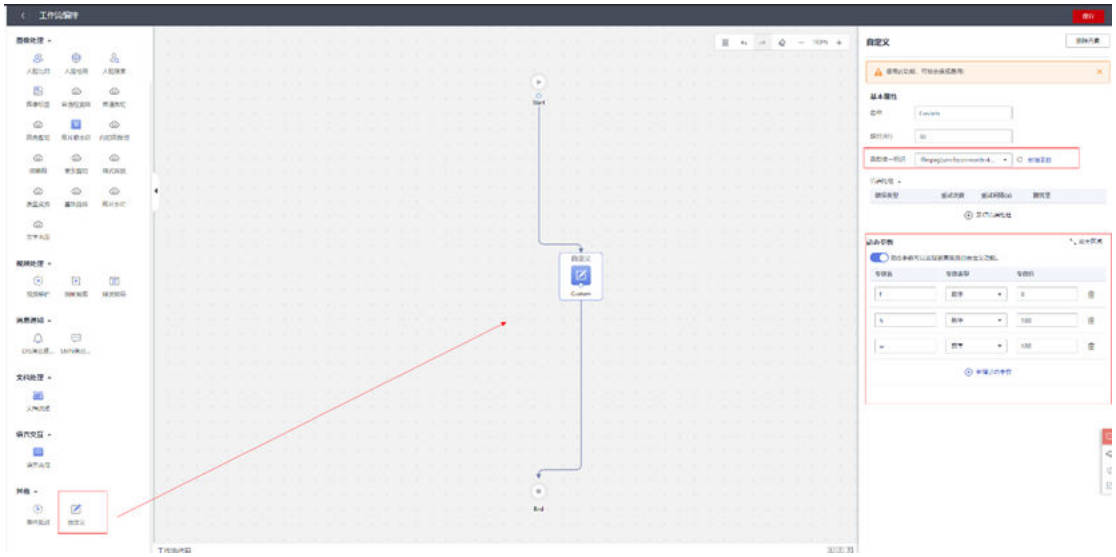
```
def handler(event, context):
    print(context)
    print(event)
    obsClient = ObsClient(
        access_key_id=context.getAccessKey(),
        secret_access_key=context.getSecretKey(),
        server='https://obs.cn-north-4.myhuaweicloud.com' #OBS的endpoint，不同region的OBS Endpoint不同
    )
    bucketName = event['Records'][0]['obs']['bucket']['name']
    objectKey = urllib.parse.unquote(event['Records'][0]['obs']['object']['key'])  # a/b/c
    frame = event["dynamic_source"]["f"] #截取的秒数
    height = event["dynamic_source"]["h"]
    width = event["dynamic_source"]["w"]
    # 使用访问OBS
    start = time.time()
    resp = obsClient.getObject(bucketName, objectKey, "/tmp/" + objectKey)
    end = time.time() - start
    # print(resp)
    print("download test.jpg run :" + str(end - start))
    start = time.time()
    cmd = "/opt/function/code/ffmpeg -i /tmp/" + objectKey + " -ss {f} -vframes 1 -vf scale={w}:{h} -y /tmp/
{f}.jpg".format(
        f=frame, w=width, h=height)
    print(cmd)
    os.system(cmd)
    dsObjectKey = "{}.jpg".format(frame)
    start = time.time()
    out_obj = "output/{obj}/{file}".format(obj=objectKey, file=dsObjectKey)
    resp = obsClient.putFile(bucketName, out_obj,
                             "/tmp/" + dsObjectKey)
    end = time.time() - start
    print(" upload run :" + str(end - start))
    # 关闭obsClient
    obsClient.close()
    tasks = [
        {
            "output": {
                "bucket": bucketName,
                "object": out_obj,
                "location": "cn-north-4"
            }
        }
    ]
    event["dynamic_source"] = {
        "tasks": tasks
    }
    return event
```

步骤5 函数保存成功后，进入到数工坊DWR控制台工作流界面，点击“创建工作流”。



步骤6 在工作流编排界面选择自定义算子并完成连线。其中基本属性部分的函数唯一标识选择步骤**步骤4**创建好的自定义函数。

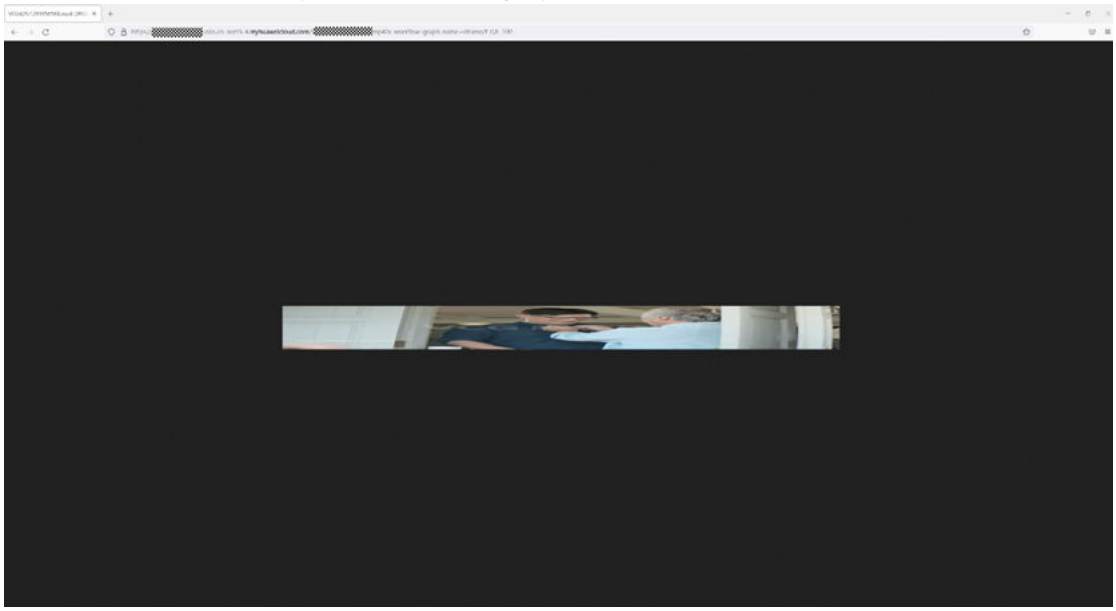
加入动态参数f（截图秒数frame），h（输出图片高度height），w（输出图片宽度width）。



步骤7 点击保存工作流。可选择同步并支持匿名访问

步骤8 保存成功后通过url方式触发工作流。其中可以指定参数f（截图秒数frame）,h（输出图片高度）获得截帧结果。

https://test-workflow-bj4.obs.cn-north-4.myhuaweicloud.com/VCG42N1291956568.mp4?x-workflow-graph-name=vframe/f_0,h_100



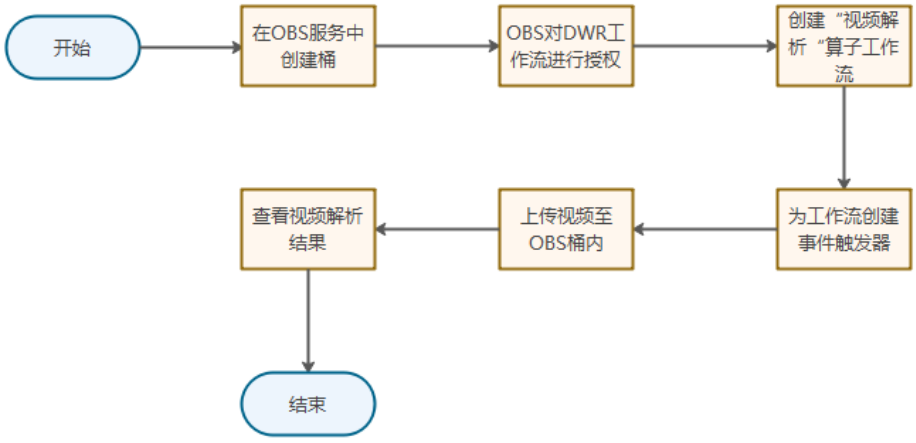
----结束

5 视频解析

场景介绍

解析视频的元数据信息，包括：视频时长、视频格式、码率等。用于视频播放时获取时长，分辨率展示 等等。

流程一览



操作步骤

1. 在OBS服务中创建桶

   在OBS服务控制台创建两个桶，一个用于上传待处理的数据对象，一个用于存储处理后的数据对象。桶的详细介绍参见**创建桶**。

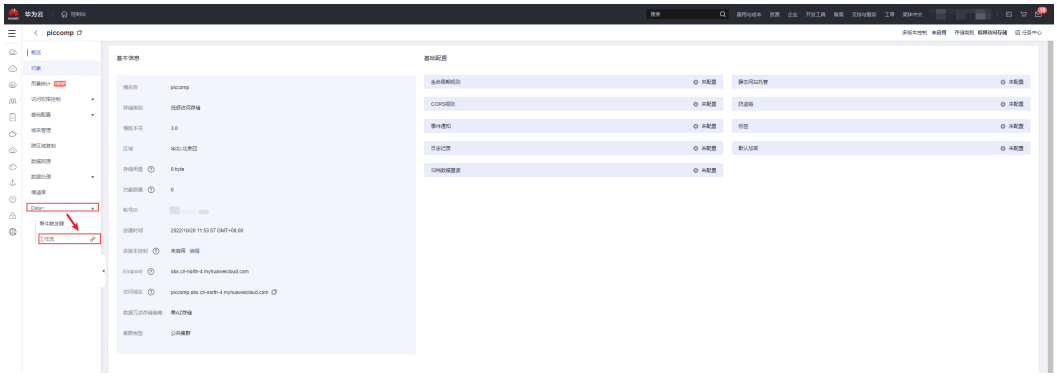
   数据输入桶：“piccomp”

   数据处理后输出桶：“piccomp-output”

   说明

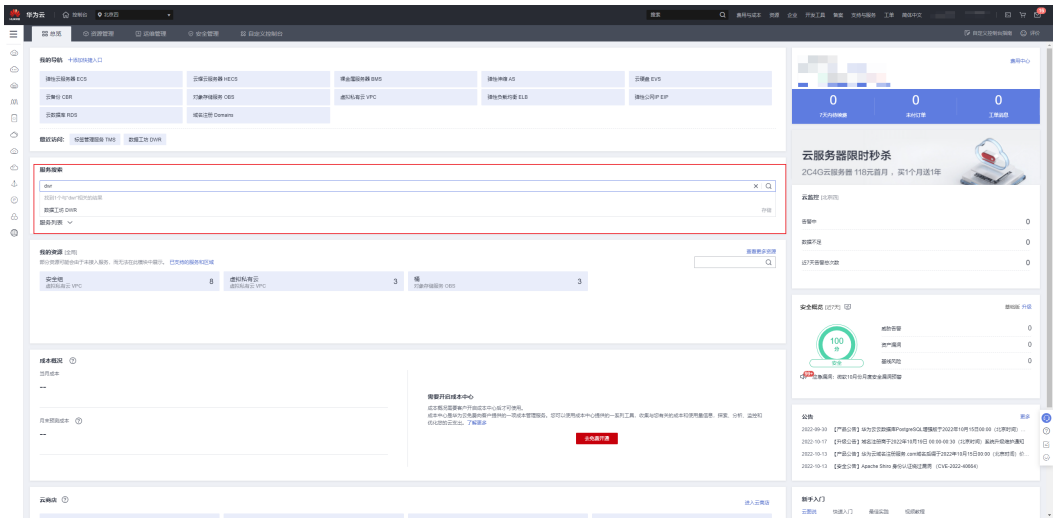
   这里创建两个桶是为了防止无限循环。因为处理后的数据如果放在源桶，又跟事件触发器执行的条件匹配，会继续触发工作流执行。详细介绍参见**了解更多无限循环原因**。

2. OBS对DWR工作流进行授权。以使得DWR可以对OBS内的存储数据，按照您指定的顺序（如解析/转码/审核等）自动化处理。



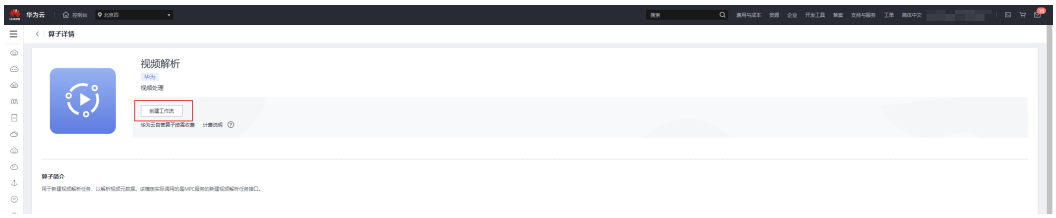
3. 进入“视频解析“算子工作流

   a. 打开控制台，在“服务搜索“框内搜索“DWR/数据工坊“并选择加载建议信息



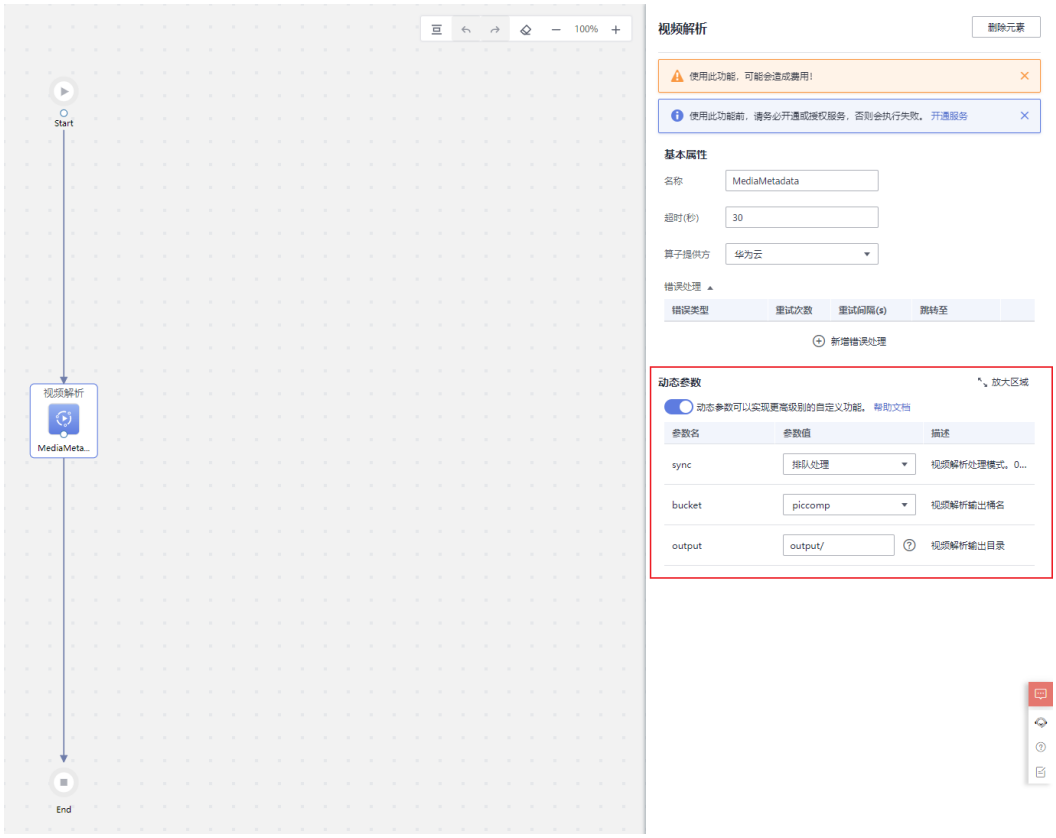
   b. 选择“视频解析“算子

   c. 单击“创建工作流“



4. 在DWR服务中创建工作流
   a. 将“视频解析”算子拖拽至编排区域进行串联编排，并填写相关属性配置信息，如图5-1所示。
      - bucket为视频处理后的输出桶，这里设置为“piccomp-output”。
   b. 保存并填写工作流名称“MediaMetadata”。

图 5-1 编排工作流

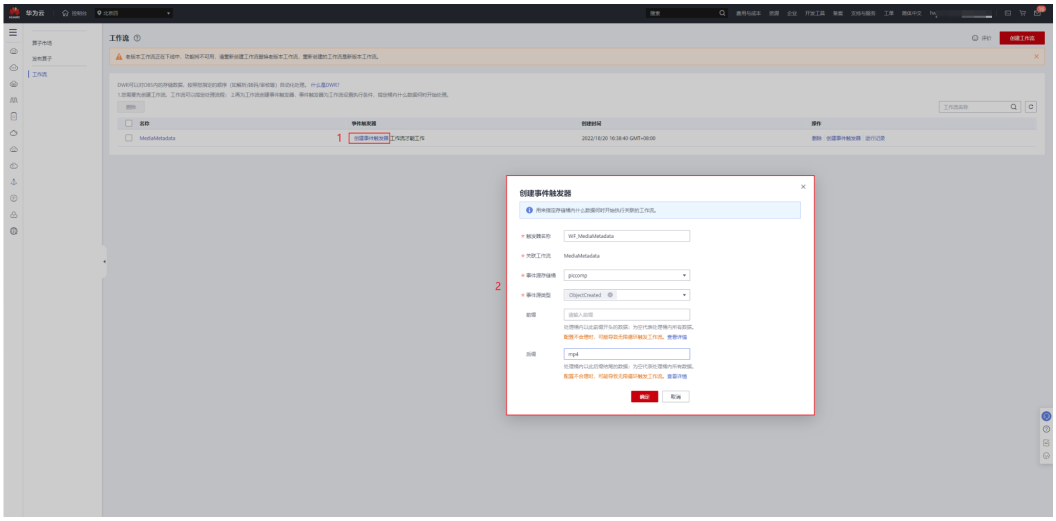


5. 在DWR服务中创建事件触发器。
   在工作流列表中，单击工作流“MediaMetadata”操作列的“创建事件触发器”进行触发器的创建，如图5-2所示。
   事件源存储桶选择“piccomp”。

说明

触发器执行存在一定的延时，大概5分钟生效。

图 5-2 创建事件触发器



6. 上传文件触发工作流执行

   登录OBS服务控制台，进入桶“piccomp”对象列表页面上传视频，如图5-3所示。

   上传视频至piccomp桶后OBS会自动生成事件触发工作流运行，进行视频解析。

图 5-3 上传视频

7. 查看视频解析结果

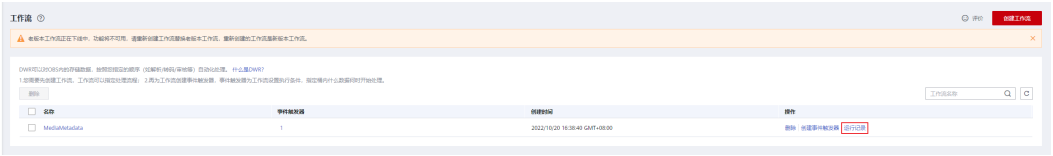
   解析后的视频参数存放在piccomp-output桶的output文件夹中，如图5-4所示。原视频的元数据信息如图5-5所示。

图 5-4 解析后结果文件

图 5-5 视频解析结果内容


```
{
    "size": 181398241,
    "duration_ms": 137.0,
    "duration": 137,
    "format": "MP4",
    "bitrate": 10584942,
    "video": [
        {
            "width": 1920,
            "height": 1080,
            "bitrate": 10026,
            "bitrate_bps": 10267567,
            "frame_rate": 25,
            "codec": "H.264",
            "peak_bitrate": 0
        }
    ],
    "audio": [
        {
            "codec": "AAC",
            "sample": 48000,
            "channels": 2,
            "sky_switch": 0,
            "bitrate": 309,
            "bitrate_bps": 317375,
            "file_size": 0,
            "embed_video": 0
        }
    ]
}
```


8. （可选）查看运行记录
   a. 在DWR控制台左侧导航栏选择“工作流”，进入“工作流”页面
   b. 在工作流列表中，单击工作流“MediaMetadata”操作列的“运行记录”可以查看函数流的执行记录。详细介绍参见**函数流执行历史管理**。



6 媒资转码

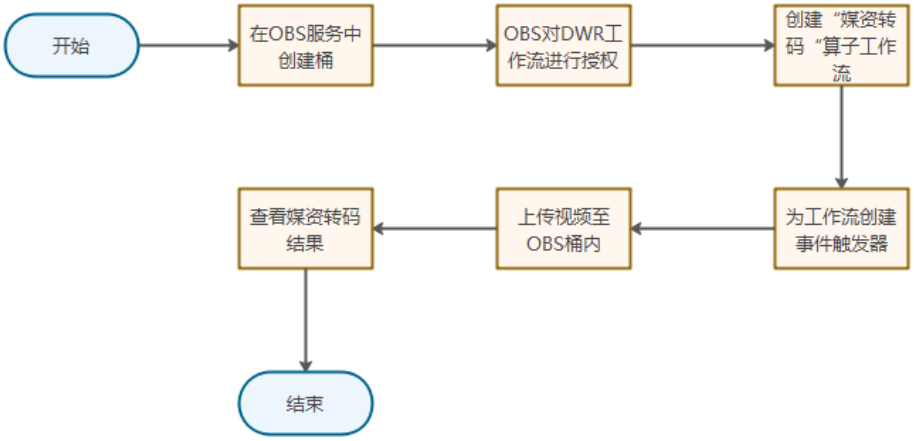
场景介绍

执行MPC服务的预置转码模板“DASH_H.265_4K_低码_1入9出”，将片源转为4K、2K、1080等码率的视频。该模板实际调用的是MPC服务的新建转码任务接口。

约束与限制

媒资转码算子不支持同步返回，如您配置了同步工作流执行媒资转码算子，则会采用异步方式执行，且返回值会格式异常。

流程一览



操作步骤

1. 在OBS服务中创建桶

   在OBS服务控制台创建两个桶，一个用于上传待处理的数据对象，一个用于存储处理后的数据对象。桶的详细介绍参见**创建桶**。

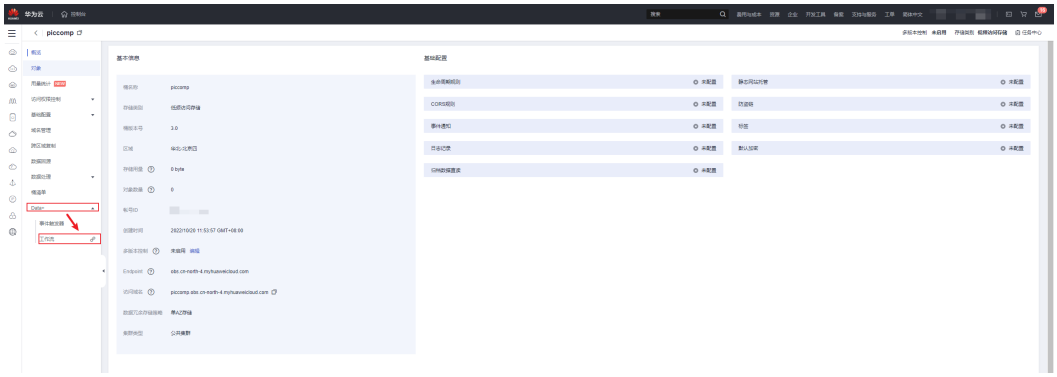
   数据输入桶：“piccomp”

   数据处理后输出桶：“piccomp-output”

说明

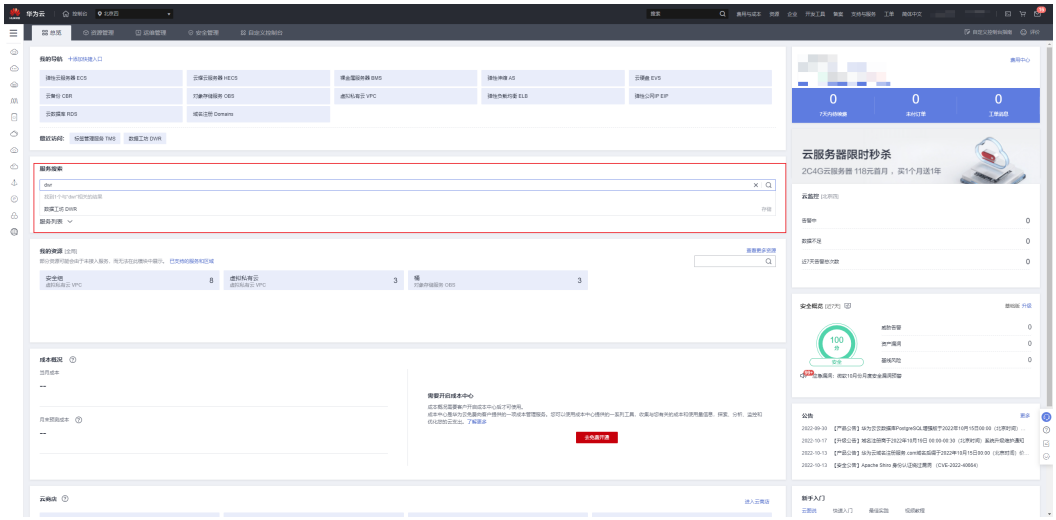
这里创建两个桶是为了防止无限循环。因为处理后的数据如果放在源桶，又跟事件触发器执行的条件匹配，会继续触发工作流执行。详细介绍参见**了解更多无限循环原因**。

2. OBS对DWR工作流进行授权。以使得DWR可以对OBS内的存储数据，按照您指定的顺序（如解析/转码/审核等）自动化处理。



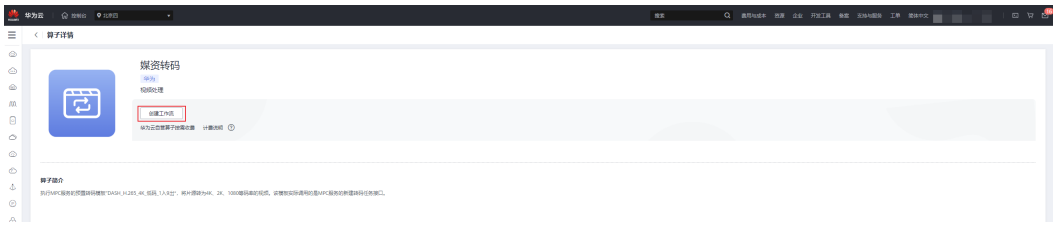
3. 进入“媒资转码“算子工作流

   a. 打开控制台，在“服务搜索“框内搜索“DWR/数据工坊“并选择加载建议信息



   b. 选择“媒资转码“算子

   c. 单击“创建工作流“



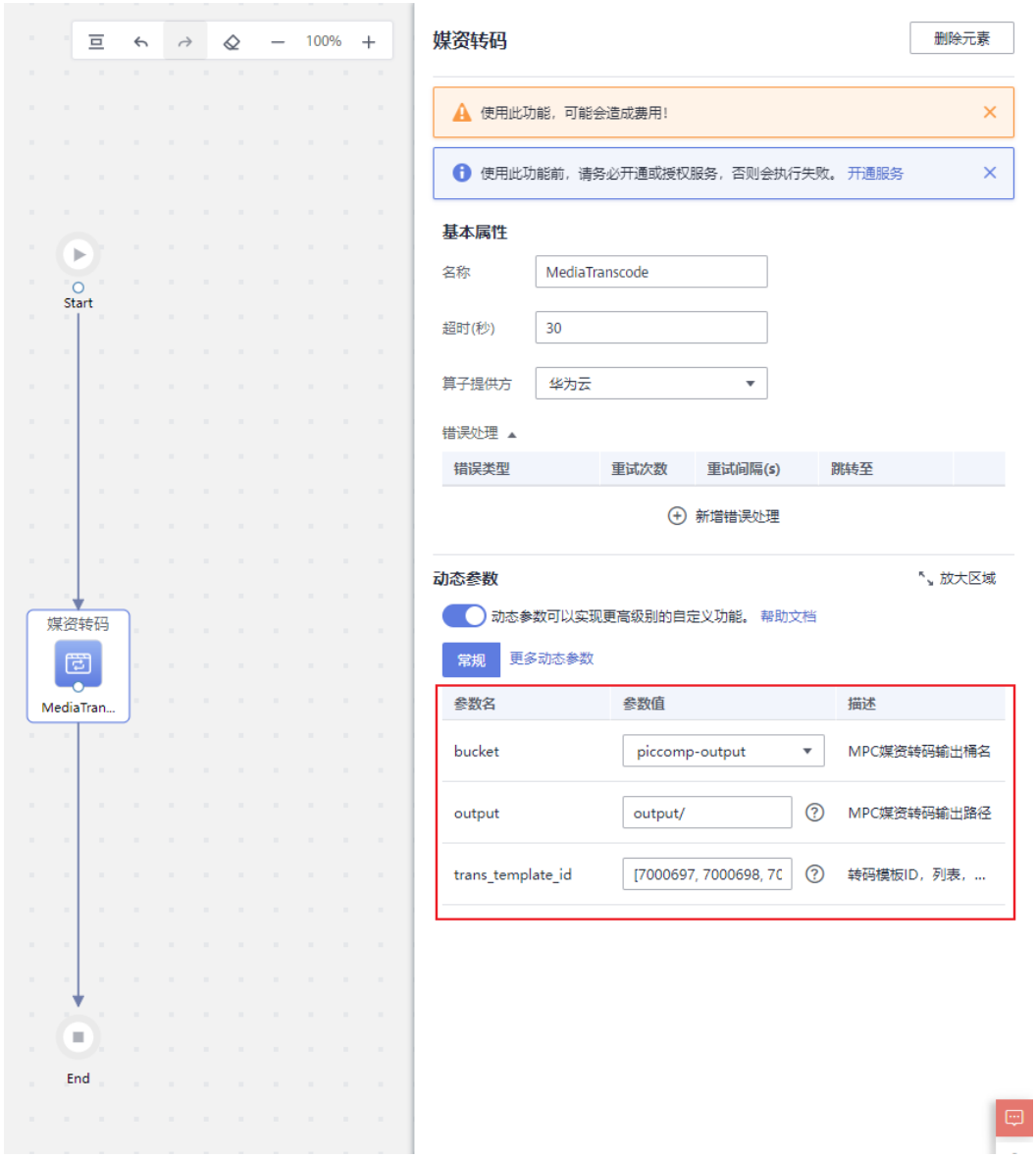
4. 在DWR服务中创建工作流

   a. 将“媒资转码”算子拖拽至编排区域进行串联编排，并填写相关属性配置信息，如图6-1所示。

      - bucket为视频处理后的输出桶，这里设置为“piccomp-output”。

   b. 保存并填写工作流名称“MediaTranscode”。编排工作流

图 6-1 编排工作流



5. 在DWR服务中创建事件触发器。

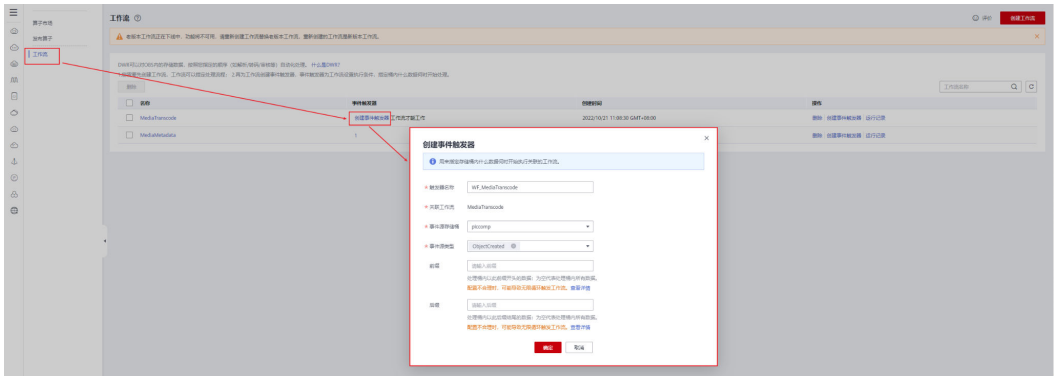
在工作流列表中，单击工作流“MediaTranscode”操作列的“创建事件触发器”进行触发器的创建，如图6-2所示。

事件源存储桶选择“piccomp”。

说明

触发器执行存在一定的延时，大概5分钟生效。

图 6-2 创建事件触发器



6. 上传文件触发工作流执行

登录OBS服务控制台，进入桶“piccomp”对象列表页面上传视频，如图6-3所示。

上传视频至piccomp桶后OBS会自动生成事件触发工作流运行，进行媒资转码。

图 6-3 上传视频

7. 查看媒资转码结果

转码后的视频存放在piccomp-output桶的output文件夹中，如图6-4所示。媒资转码工作流运行记录信息如图6-5所示。

图 6-4 媒资转码后结果文件

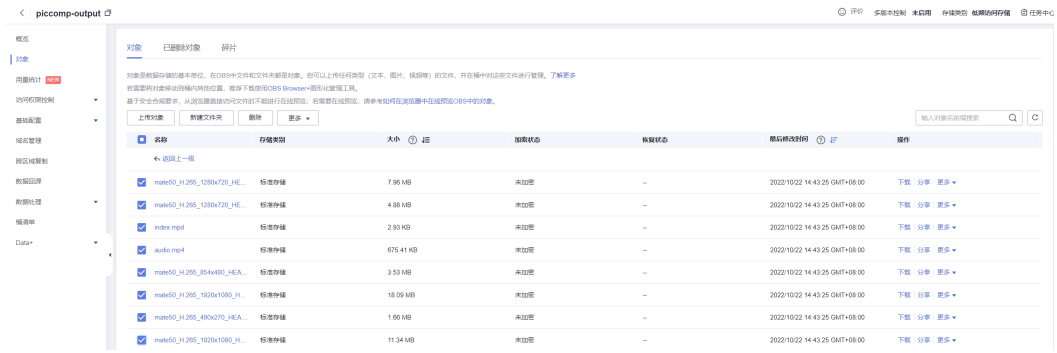
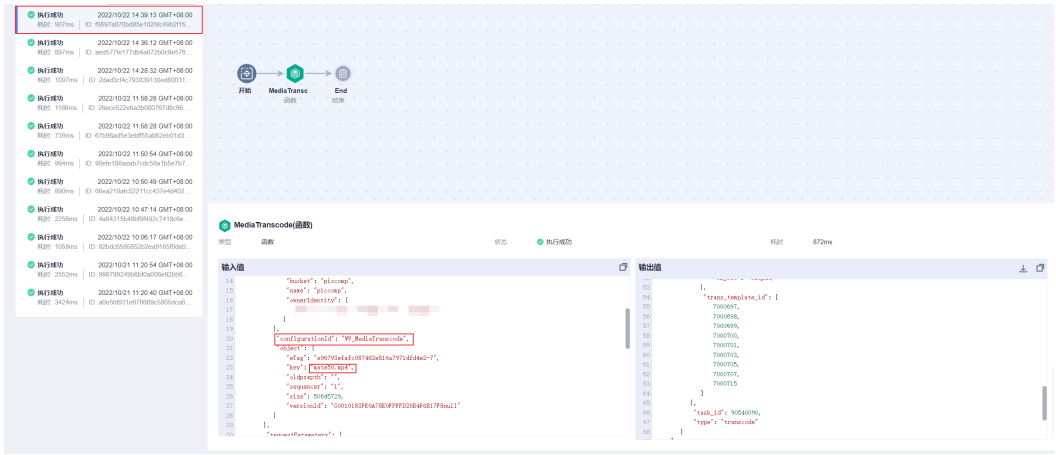


图 6-5 媒资转码工作流运行记录



8. （可选）查看运行记录

   a. 在DWR控制台左侧导航栏选择“工作流”，进入“工作流”页面

   b. 在工作流列表中，单击工作流“MediaTranscode”操作列的“运行记录”可以查看函数流的执行记录。详细介绍参见**函数流执行历史管理**。