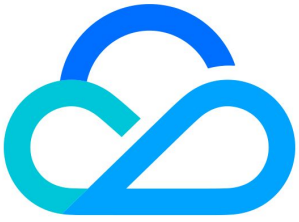
云函数

触发器



腾讯云

文档目录

触发器
触发器概述

最近更新时间：2024-07-22 11:57:53

腾讯云云函数目前支持**事件触发**与 **HTTP 请求触发**两种触发模式：

事件触发

事件触发（Event-Triggered）是典型的无服务器运行方式，核心组件是 SCF 函数和事件源。其中，事件源是发布事件（Event）的腾讯云服务或用户自定义代码，SCF 函数是事件的处理者，而函数触发器就是管理函数和事件源对应关系的集合。例如以下场景：

- **图像/视频处理**：用户上传图片时将图片切割成合适的尺寸。用户使用该应用上传照片，应用将这些用户照片存储到 COS 中并且创建每个用户照片的缩略图，并在用户页面上显示这些缩略图。本场景下，您需要选择 COS 作为事件源，在文件创建时将事件（Event）发布给 SCF 函数，事件数据提供关于存储桶和文件的所有信息。
- **数据处理**：半夜12点，分析一天所收集的数据（例如 clickstream）并生成报告。本场景下，您需要选择定时器作为事件源，在一个特定时间将事件（Event）发布给 SCF 函数。
- **自定义的应用程序**：在您的某个应用程序中调用第一个图像处理 SCF 函数，作为应用程序的一个模块。本场景下，您需要该应用程序中自行调用 Invoke API 来发布事件（Event）。

这些事件源可以是以下任意之一：

- **内部事件源**：这些是经过预配置可与 SCF 一起使用的腾讯云云服务。当您配置了这些事件源触发函数时，函数将在出现事件时被自动调用。事件源和函数的关联关系（即事件源映射）将在事件源侧维护。
- **自定义应用程序**：您可以让自定义应用程序发布事件和调用 SCF 函数。

示例1：COS发布事件并调用函数

您可以配置 COS 的事件源映射，决定 COS 在发生何种行为时触发 SCF 函数（如 PUT、DELETE 对象等）。COS 的事件源映射存储在 COS 中，使用存储桶通知功能，引导 COS 在出现特定事件类型时调用函数：

- 创建 COS 触发器。
- 用户在存储桶中创建/删除对象。
- COS 检测到对象创建/删除事件。
- COS 自动调用函数，将根据存储在 COS 配置中的事件源映射明确应该调用哪个函数。将 Bucket 及 Object 信息作为事件数据传递给函数。

示例2：定时器发布时间并调用函数

定时器的事件源映射将保存在 SCF 函数配置中，决定何时自动触发函数：

- 创建定时触发器。
- 该定时器在配置时间时自动调用函数。

示例3：自定义应用程序调用函数

如果您需要在自定义应用程序中调用某个 SCF 函数，在这种情况下您不需要配置函数触发器，也不需设置事件源映射。此时，事件源使用 Invoke API。

- 自定义应用程序使用 Invoke API 调用函数，自行传入事件数据。
- 函数接收到触发请求并执行。
- 如果使用了同步调用方式，函数将向应用程序返回结果。

> **注意：**
> 在此示例中，由于自定义应用程序和函数均为同一个用户生产的，可以指定用户凭证（APPID、SecretId 和 SecretKey）。

注意事项

1. 目前单个云函数支持的触发器相关限制，可见 配额及限制 。
2. 由于不同云服务的限制，事件源映射关系有着特定的限制。例如：对于 COS 触发器而言，同一个 COS Bucket 的相同事件（如文件上传），不能触发多个不同的函数。

HTTP 请求触发

HTTP 请求触发是云函数 Web Function 支持的特殊触发方式，原生的 HTTP 请求可以直接通过函数 URL 透传到函数环境，触发函数的运行与处理，适合 Web 服务场景开发，详细使用方式请参考 Web 函数概述 。

触发器事件消息结构汇总

最近更新时间：2024-11-07 10:23:52

本文档主要汇总所有对接云函数 SCF 的触发器事件的消息结构，触发器配置详情及限制请参见 触发器管理文档 。

> **注意：**
> 触发器传递的入参事件结构已有部分定义，可直接使用。您可以通过 java cloud event 定义 获取 Java 的库并使用，通过 go cloud event 定义 获取 Golang 的库并使用。

API 网关触发器的集成请求事件消息结构

在 API 网关触发器接收到请求时，会将类似以下 JSON 格式的事件数据发送给绑定的 SCF 函数。详情见 API 网关触发器 。

```
{
  "requestContext": {
    "serviceId": "service-f94sy04v",
    "path": "/test/{path}",
    "httpMethod": "POST",
    "requestId": "c6af9ac6-****-****-9a41-93e8deadbeef",
    "identity": {
      "secretId": "SECRET_ID" # 硬编码密钥到代码中有可能随代码泄露而暴露，为保护密钥安全，建议将密钥设置在环境变量中。
                              # 可参考 https://cloud.tencent.com/document/product/1278/85305
    },
    "sourceIp": "10.0.2.14",
    "stage": "release"
  },
  "headers": {
    "Accept-Language": "en-US,en,cn",
    "Accept": "text/html,application/xml,application/json",
    "Host": "service-3ei3tii4-251000691.ap-guangzhou.apigateway.myqloud.com",
    "User-Agent": "User Agent String"
  },
  "body": "{\"test\":\"body\"}",
  "pathParameters": {
    "path": "value"
  },
  "queryStringParameters": {
    "foo": "bar"
  },
  "headerParameters":{
    "Refer": "10.0.2.14"
  },
```

```
  "stageVariables": {
    "stage": "release"
  },
  "path": "/test/value",
  "queryString": {
    "foo" : "bar",
    "bob" : "alice"
  },
  "httpMethod": "POST"
}
```

Timer 触发器的事件消息结构

在指定时间触发调用云函数时，会将类似以下的 JSON 格式事件数据发送给绑定的 SCF 云函数。

```
{
    "Type":"Timer",
    "TriggerName":"EveryDay",
    "Time":"2019-02-21T11:49:00Z",
    "Message":"user define msg body"
}
```

COS 触发器的事件消息结构

在指定的 COS Bucket 发生对象创建或对象删除事件时，会将类似以下的 JSON 格式事件数据发送给绑定的 SCF 函数。详情见 COS 触发器 。

```
    "Records": [{
        "cos": {
            "cosSchemaVersion": "1.0",
            "cosObject": {
                "url": "http://testpic-1253970026.cos.ap-
chengdu.myqcloud.com/testfile",
                "meta": {
                    "x-cos-request-id":
"NWMxOWY4MGFfMjViMjU4NjRfMTUy********ZjM=",
                    "Content-Type": ""
                },
                "vid": "",
                "key": "/1253970026/testpic/testfile",
                "size": 1029
            },
            "cosBucket": {
                "region": "cd",
                "name": "testpic",
                "appid": "1253970026"
```

```
            },
            "cosNotificationId": "unkown"
        },
        "event": {
            "eventName": "cos:ObjectCreated:*",
            "eventVersion": "1.0",
            "eventTime": 1545205770,
            "eventSource": "qcs::cos",
            "requestParameters": {
                "requestSourceIP": "192.168.15.101",
                "requestHeaders": {
                    "Authorization": "q-sign-algorithm=******"
                }
            },
            "eventQueue":
"qcs:0:lambda:cd:appid/1253970026:default.printevent.$LATEST",
            "reservedInfo": "",
            "reqid": 179398952
        }
    }]
}
```

CKafka 触发器的事件消息结构

在指定的 CKafka Topic 接收到消息时，云函数后台的消费者模块会消费 CKafka Topic 中的消息，并将消息组装为类似以下的 JSON 格式事件，触发绑定的函数并将数据内容作为入参传递给函数。详情见 CKafka 触发器 。

```
{
  "Records": [
    {
      "Ckafka": {
        "topic": "test-topic",
        "partition":1,
        "offset":36,
        "msgKey": "None",
        "msgBody": "Hello from Ckafka!"
      }
    },
    {
      "Ckafka": {
        "topic": "test-topic",
        "partition":1,
        "offset":37,
        "msgKey": "None",
        "msgBody": "Hello from Ckafka again!"
      }
    }
```

```
  ]
}
```

CLS 触发器的事件消息结构

在指定的 CLS 触发器接收到消息时，CLS 的后台消费者模块会消费消息，并将消息组装异步调用您的函数。为保证单次触发传递数据的效率，数据字段的值是 Base64 编码的 ZIP 文档。详情见 CLS 触发器 。

```
{
  "clslogs": {
    "data":
"ewogICAgIm1lc3NhZ2VUeXBlIjogIkRBVEFfTUVTU0FHRSIsCiAgICAib3duZXIiOiAiMTIzNDU2Nzg
5MDEyIiwKICAgICJsb2dHcm91cCI6I..."
  }
}
```

在解码和解压缩后，日志数据类似以下 JSON 体，以 CLS Logs 消息数据（已解码）为例：

```
{
    "topic_id": "xxxx-xx-xx-xx-yyyyyyyy",
    "topic_name": "testname",
    "records": [{
        "timestamp": "1605578090000000",
        "content": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
    }, {
        "timestamp": "1605578090000003",
        "content": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
    }]
}
```

MPS 触发器的事件消息结构

在指定的 MPS 触发器接收到消息时，事件结构与字段以 WorkflowTask 任务为例。详情见 MPS 触发器 。示例如下：

```
{
    "EventType":"WorkflowTask",
    "WorkflowTaskEvent":{
        "TaskId":"245****654-WorkflowTask-f46dac7fe2436c47******d71946986t0",
        "Status":"FINISH",
        "ErrCode":0,
        "Message":"",
        "InputInfo":{
            "Type":"COS",
            "CosInputInfo":{
                "Bucket":"macgzptest-125****654",
```

```
            "Region":"ap-guangzhou",
            "Object":"/dianping2.mp4"
        }
    },
    "MetaData":{
        "AudioDuration":11.261677742004395,
        "AudioStreamSet":[
            {
                "Bitrate":127771,
                "Codec":"aac",
                "SamplingRate":44100
            }
        ],
        "Bitrate":2681468,
        "Container":"mov,mp4,m4a,3gp,3g2,mj2",
        "Duration":11.261677742004395,
        "Height":720,
        "Rotate":90,
        "Size":3539987,
        "VideoDuration":10.510889053344727,
        "VideoStreamSet":[
            {
                "Bitrate":2553697,
                "Codec":"h264",
                "Fps":29,
                "Height":720,
                "Width":1280
            }
        ],
        "Width":1280
    },
    "MediaProcessResultSet":[
        {
            "Type":"Transcode",
            "TranscodeTask":{
                "Status":"SUCCESS",
                "ErrCode":0,
                "Message":"SUCCESS",
                "Input":{
                    "Definition":10,
                    "WatermarkSet":[
                        {
                            "Definition":515247,
                            "TextContent":"",
                            "SvgContent":""
                        }
                    ],
```

```
                        "OutputStorage":{
                            "Type":"COS",
                            "CosOutputStorage":{
                                "Bucket":"gztest-125****654",
                                "Region":"ap-guangzhou"
                            }
                        },
                        "OutputObjectPath":"/dasda/dianping2_transcode_10",

"SegmentObjectName":"/dasda/dianping2_transcode_10_{number}",
                        "ObjectNumberFormat":{
                            "InitialValue":0,
                            "Increment":1,
                            "MinLength":1,
                            "PlaceHolder":"0"
                        }
                    },
                    "Output":{
                        "OutputStorage":{
                            "Type":"COS",
                            "CosOutputStorage":{
                                "Bucket":"gztest-125****654",
                                "Region":"ap-guangzhou"
                            }
                        },
                        "Path":"/dasda/dianping2_transcode_10.mp4",
                        "Definition":10,
                        "Bitrate":293022,
                        "Height":320,
                        "Width":180,
                        "Size":401637,
                        "Duration":11.26200008392334,
                        "Container":"mov,mp4,m4a,3gp,3g2,mj2",
                        "Md5":"31dcf904c03d0cd78346a12c25c0acc9",
                        "VideoStreamSet":[
                            {
                                "Bitrate":244608,
                                "Codec":"h264",
                                "Fps":24,
                                "Height":320,
                                "Width":180
                            }
                        ],
                        "AudioStreamSet":[
                            {
                                "Bitrate":48414,
                                "Codec":"aac",
```

```
                                "SamplingRate":44100
                            }
                        ]
                    }
                },
                "AnimatedGraphicTask":null,
                "SnapshotByTimeOffsetTask":null,
                "SampleSnapshotTask":null,
                "ImageSpriteTask":null
            },
            {
                "Type":"AnimatedGraphics",
                "TranscodeTask":null,
                "AnimatedGraphicTask":{
                    "Status":"FAIL",
                    "ErrCode":30010,
                    "Message":"TencentVodPlatErr Or Unkown",
                    "Input":{
                        "Definition":20000,
                        "StartTimeOffset":0,
                        "EndTimeOffset":600,
                        "OutputStorage":{
                            "Type":"COS",
                            "CosOutputStorage":{
                                "Bucket":"gztest-125****654",
                                "Region":"ap-guangzhou"
                            }
                        },

"OutputObjectPath":"/dasda/dianping2_animatedGraphic_20000"
                    },
                    "Output":null
                },
                "SnapshotByTimeOffsetTask":null,
                "SampleSnapshotTask":null,
                "ImageSpriteTask":null
            },
            {
                "Type":"SnapshotByTimeOffset",
                "TranscodeTask":null,
                "AnimatedGraphicTask":null,
                "SnapshotByTimeOffsetTask":{
                    "Status":"SUCCESS",
                    "ErrCode":0,
                    "Message":"SUCCESS",
                    "Input":{
                        "Definition":10,
```

```
                        "TimeOffsetSet":[

                        ],
                        "WatermarkSet":[
                            {
                                "Definition":515247,
                                "TextContent":"",
                                "SvgContent":""
                            }
                        ],
                        "OutputStorage":{
                            "Type":"COS",
                            "CosOutputStorage":{
                                "Bucket":"gztest-125****654",
                                "Region":"ap-guangzhou"
                            }
                        },

"OutputObjectPath":"/dasda/dianping2_snapshotByOffset_10_{number}",
                        "ObjectNumberFormat":{
                            "InitialValue":0,
                            "Increment":1,
                            "MinLength":1,
                            "PlaceHolder":"0"
                        }
                    },
                    "Output":{
                        "Storage":{
                            "Type":"COS",
                            "CosOutputStorage":{
                                "Bucket":"gztest-125****654",
                                "Region":"ap-guangzhou"
                            }
                        },
                        "Definition":0,
                        "PicInfoSet":[
                            {
                                "TimeOffset":0,

"Path":"/dasda/dianping2_snapshotByOffset_10_0.jpg",
                                "WaterMarkDefinition":[
                                    515247
                                ]
                            }
                        ]
                    }
                },
```

```
                "SampleSnapshotTask":null,
                "ImageSpriteTask":null
            },
            {
                "Type":"ImageSprites",
                "TranscodeTask":null,
                "AnimatedGraphicTask":null,
                "SnapshotByTimeOffsetTask":null,
                "SampleSnapshotTask":null,
                "ImageSpriteTask":{
                    "Status":"SUCCESS",
                    "ErrCode":0,
                    "Message":"SUCCESS",
                    "Input":{
                        "Definition":10,
                        "OutputStorage":{
                            "Type":"COS",
                            "CosOutputStorage":{
                                "Bucket":"gztest-125****654",
                                "Region":"ap-guangzhou"
                            }
                        },

"OutputObjectPath":"/dasda/dianping2_imageSprite_10_{number}",
                        "WebVttObjectName":"/dasda/dianping2_imageSprite_10",
                        "ObjectNumberFormat":{
                            "InitialValue":0,
                            "Increment":1,
                            "MinLength":1,
                            "PlaceHolder":"0"
                        }
                    },
                    "Output":{
                        "Storage":{
                            "Type":"COS",
                            "CosOutputStorage":{
                                "Bucket":"gztest-125****654",
                                "Region":"ap-guangzhou"
                            }
                        },
                        "Definition":10,
                        "Height":80,
                        "Width":142,
                        "TotalCount":2,
                        "ImagePathSet":[
                            "/dasda/imageSprite/dianping2_imageSprite_10_0.jpg"
                        ],
```

```
"WebVttPath":"/dasda/imageSprite/dianping2_imageSprite_10.vtt"
                    }
                }
            }
        ]
    }
}
```

CLB 触发器的事件消息结构

在 CLB 触发器接收到请求时，会将类似以下 JSON 格式的事件数据发送给绑定的 SCF 函数。详情见 CLB 触发器说明 。

```
{
  "headers": {
    "Content-type": "application/json",
    "Host": "test.clb-scf.com",
    "User-Agent": "Chrome",

    "X-Stgw-Time": "1591692977.774",
    "X-Client-Proto": "http",
    "X-Forwarded-Proto": "http",
    "X-Client-Proto-Ver": "HTTP/1.1",
    "X-Real-IP": "9.43.175.219",
    "X-Forwarded-For": "9.43.175.xx"

    "X-Vip": "121.23.21.xx",
    "X-Vport": "xx",
    "X-Uri": "/scf_location",
    "X-Method": "POST"
    "X-Real-Port": "44347",
  },
  "payload": {
    "key1": "123",
    "key2": "abc"
  },
  “isBase64Encoded”: “false”
}
```

通过事件总线触发器传递的事件结构

通过 腾讯云事件总线，可以进一步拓展函数事件触发源，通过事件总线产生的事件将以以下形式发送给云函数，其中 `"data"` 字段里内容由事件源决定，此处以 TDMQ 为例：

```
 {
  "specversion": "0",
```

```
  "id": "13a3f42d-7258-4ada-da6d-023a33******",
  "type": "connector:tdmq",
  "source": "tdmq.cloud.tencent",
  "subjuect":
"qcs::tdmq:$region:$account:topicName/$topicSets.clusterId/$topicSets.environmen
tId/$topicSets.topicName/$topicSets.subscriptionName",
  "time": "1615430559146",
  "region": "ap-guangzhou",
  "datacontenttype": "application/json;charset=utf-8",
  "data": {
            "topic":  "persistent://appid/namespace/topic-1",
          "tags": "testtopic",
            "TopicType": "0",
            "subscriptionName": "xxxxxx",
            "toTimestamp": "1603352765001",
          "partitions": "0",
            "msgId": "123345346",
            "msgBody": "Hello from TDMQ!"
  }
```

API 网关触发器
API 网关触发器概述（即将下线）

最近更新时间：2024-08-20 15:55:41

> **注意：**
> 由于 API 网关产品计划于2025年6月30日停止服务，云函数 SCF & Serverless 应用的 API 网关触发器将进行如下调整，具体内容请参考：云函数 SCF 与 Serverless 应用平台 API 网关触发器下线通知 。如果短期您仍需要新建 API 网关触发器，请提交工单至 API 网关团队申请开白。

API 网关触发器简介

您可以通过编写 SCF 云函数来实现 Web 后端服务，并通过 API 网关对外提供服务。API 网关会将请求内容以参数形式传递给函数，并将函数返回作为响应返回给请求方。API 网关触发器同时支持事件函数与 Web 函数触发，本文仅介绍**事件函数**触发的请求方式，Web 函数触发请参见 Web 函数触发器管理 。
API 网关触发器具有以下特点：

- **Push 模型**
  API 网关在接收到 API 请求后，如果 API 在网关上的后端配置了对接云函数，该函数将会被触发运行。同时 API 网关会将 API 请求的相关信息以 event 入参的形式发送给被触发的函数。API 请求的相关信息包含了例如具体接收到请求的服务和 API 规则、请求的实际路径、方法、请求的 path、header、query 等内容。
- **同步调用**
  API 网关以同步调用的方式来调用函数，会在 API 网关中配置的超时时间未到前等待函数返回。有关调用类型的更多信息，请参见 调用类型 。

API 网关触发器配置

API 网关触发器分别支持在 云函数控制台 或在 API 网关控制台 中进行配置，云函数使用的 API 网关触发器默认 QPS 上限为500，您可以在 API 网关控制台-服务 调整 QPS 配置。

云函数控制台

在**云函数控制台**中，支持在触发方式中添加 API 网关触发器。支持选取已有 API 服务或新建 API 服务。支持请求方法（目前支持 ANY、GET、HEAD、POST、PUT、DELETE 六种方法的请求）、发布环境（测试、预发布及发布环境）及鉴权方式（API 网关密钥对）的定义。详情见 触发器管理 。

API 网关控制台

在 **API 网关控制台**中配置 API 规则时，后端配置可选 Cloud Function，且在选择 Cloud Function 后，即可选择与 API 服务相同地域的云函数。在 API 网关控制台上，可以配置及管理更高阶的 API 服务，如限流计划、黑白名单等。

在 API 网关配置对接云函数时，也需要配置超时时间。API 网关中的请求超时时间和云函数的运行超时时间，两者分别生效。超时规则如下：

- **API 网关超时时间 > 云函数超时时间**
  云函数超时先生效，API 请求响应为 `200 HTTP code`，但返回内容为云函数超时报错内容。
- **API 网关超时时间 < 云函数超时时间**
  API 网关超时先生效，API 请求响应为 `5xx HTTP code`，标识请求超时。

API 网关触发器绑定限制

API 网关中，一条 API 规则仅能绑定一个云函数，但一个云函数可以被多个 API 规则绑定为后端。您可以在 API 网关控制台 创建一个包含不同路径的 API 并将后端指向同一个函数。相同路径、相同请求方法及不同发布环境的 API 被视为同一个 API，无法重复绑定。
目前 API 网关触发器仅支持同地域云函数绑定，例如广州地区创建的云函数，仅支持被广州地区创建的 API 服务中规则所绑定和触发。如果您想要使用特定地域的 API 网关配置来触发云函数，可以通过在对应地域下创建函数来实现。

请求与响应

针对 API 网关发送到云函数的请求处理方式，和云函数响应给 API 网关的返回值处理方式，称为请求方法和响应方法。请求方法和响应方法规划和实现的分别有透传方式和集成方式。

集成请求与透传请求

集成请求，是指 API 网关会将 HTTP 请求内容，转换为请求数据结构；请求数据结构作为函数的 event 输入参数，传递给函数并进行处理。具体请求数据结构说明如下。
透传请求请参考 Web 函数触发器管理 。

> **注意：**
> 当您需要将图片或文件通过 API 网关传入云函数时，需要将图片或文件进行 Base64 编码。如果上传的文件在 Base64 编码后的大小超过6MB，建议您通过客户端先将文件上传至 对象存储 COS，再将 Object 地址传递给云函数，由云函数从 COS 拉取文件，以完成大文件的上传。

API 网关触发器的集成请求事件消息结构

在 API 网关触发器接收到请求时，会将类似以下 JSON 格式的事件数据发送给绑定的云函数。

```
{
  "requestContext": {
    "serviceId": "service-f94sy04v",
    "path": "/test/{path}",
    "httpMethod": "POST",
    "requestId": "c6af9ac6-7b61-11e6-9a41-93e8deadbeef",
    "identity": {
      "secretId": "abdcdxxxxxxxsdfs"
    },
    "sourceIp": "10.0.2.14",
```

```
    "stage": "release"
  },
  "headers": {
    "accept-Language": "en-US,en,cn",
    "accept": "text/html,application/xml,application/json",
    "host": "service-3ei3tii4-251000691.ap-guangzhou.apigateway.myqloud.com",
    "user-Agent": "User Agent String"
  },
  "body": "{\"test\":\"body\"}",
  "pathParameters": {
    "path": "value"
  },
  "queryStringParameters": {
    "foo": "bar"
  },
  "headerParameters":{
    "Refer": "10.0.2.14"
  },
  "stageVariables": {
    "stage": "release"
  },
  "path": "/test/value",
  "queryString": {
    "foo" : "bar",
    "bob" : "alice"
  },
  "httpMethod": "POST"
}
```

数据结构内容详细说明如下：

结构名	内容
requestContext	请求来源的 API 网关的配置信息、请求标识、认证信息、来源信息。其中： • serviceId，path，httpMethod 指向 API 网关的服务 ID、API 的路径和方法。 • stage 指向请求来源 API 所在的环境。 • requestId 标识当前这次请求的唯一 ID。 • identity 标识用户的认证方法和认证的信息。 • sourceIp 标识请求来源 IP。
path	记录实际请求的完整 Path 信息。
httpMethod	记录实际请求的 HTTP 方法。
queryString	记录实际请求的完整 Query 内容。
body	记录实际请求转换为 String 字符串后的内容。

headers	记录实际请求的完整 Header 内容。
pathParameters	记录在 API 网关中配置过的 Path 参数以及实际取值。
queryStringParameters	记录在 API 网关中配置过的 Query 参数以及实际取值。
headerParameters	记录在 API 网关中配置过的 Header 参数以及实际取值。

注意：

- 在 API 网关迭代过程中， requestContext 内的内容可能会增加更多。目前会保证数据结构内容仅增加，不删除，不对已有结构进行破坏。
- 实际请求时的参数数据可能会在多个位置出现，可根据业务需求选择使用。

集成响应与透传响应

集成响应，是指 API 网关会将云函数的返回内容进行解析，并根据解析内容构造 HTTP 响应。通过使用集成响应，可以通过代码自主控制响应的状态码、headers、body 内容，可以实现自定义格式的内容响应，例如响应 XML、HTML、JSON 甚至 JS 内容。在使用集成响应时，需要按照 API 网关触发器的集成响应返回数据结构，才可以被 API 网关成功解析，否则会出现

```
{"errno":403,"error":"Invalid scf response format. please check your scf response format."}
```

错误信息。

透传响应，是指 API 网关将云函数的返回内容直接传递给 API 请求方。通常这种响应的数据格式直接确定为 JSON 格式，状态码根据函数执行的状态定义，函数执行成功即为 200 状态码。通过透传响应，用户可以自行获取到 JSON 格式后在调用位置解析结构，获取结构内的内容。

注意：

- 如果当前通过 API 网关控制台配置的 API 网关触发器，处理响应的方式默认为透传响应。如需开启集成响应，请在 API 配置中的后端配置位置，勾选**启用集成响应**，并在代码中按如下说明的数据结构返回内容。
- 如果当前通过云函数控制台配置的 API 网关触发器，默认已开启集成响应功能，请注意返回数据的格式。
- 启用集成响应后，若同时启用异步执行，函数将返回 404 状态码。建议避免同时启用这两项配置。

API 网关触发器的集成响应返回数据结构

在 API 网关设置为集成响应时，需要将包含以下 JSON 格式的数据结构返回给 API 网关。

```
{
    "isBase64Encoded": false,
    "statusCode": 200,
    "headers": {"Content-Type":"text/html"},
    "body": "<html><body><h1>Heading</h1><p>Paragraph.</p></body></html>"
}
```

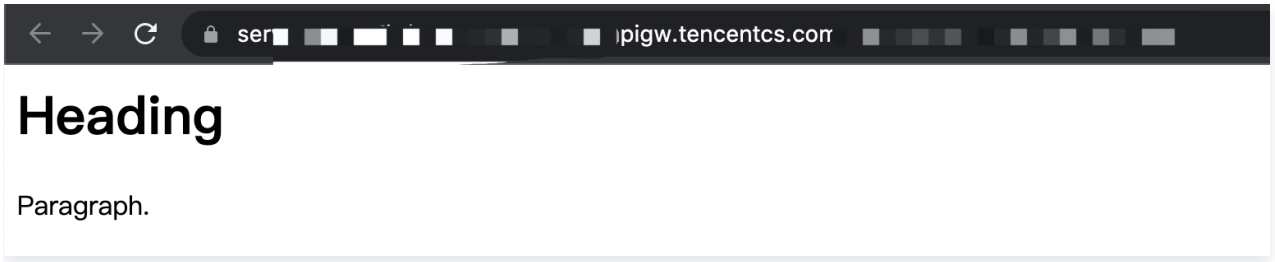
数据结构内容详细说明如下：

结构名	内容
isBase64Encoded	指明 body 内的内容是否为 Base64 编码后的二进制内容，取值需要为 JSON 格式的 true 或 false。不同语言的 true 和 false 规范不同，请根据所使用的语言对应进行调整。
statusCode	HTTP 返回的状态码，取值需要为 Integer 值。
headers	HTTP 返回的头部内容，取值需要为多个 key-value 对象，或 `key:[value,value]` 对象。其中 key、value 均为字符串。headers 请求头暂不支持 Location key。
body	HTTP 返回的 body 内容。

以 Python 3.6 为例，示例代码如下：

```
# -*- coding: utf8 -*-
import json
def main_handler(event, context):
    return {
        "isBase64Encoded": False,
        "statusCode": 200,
        "headers": {"Content-Type":"text/html"},
        "body": "<html><body><h1>Heading</h1><p>Paragraph.</p></body></html>"
    }
```

通过 API 网关触发函数，返回的结果如下：



在需要返回 key 相同的多个 headers 时，可以使用字符串数组的方式描述不同 value，例如：

```
{
    "isBase64Encoded": false,
    "statusCode": 200,
    "headers": {"Content-Type":"text/html","Key":["value1","value2","value3"]},
    "body": "<html><body><h1>Heading</h1><p>Paragraph.</p></body></html>"
}
```

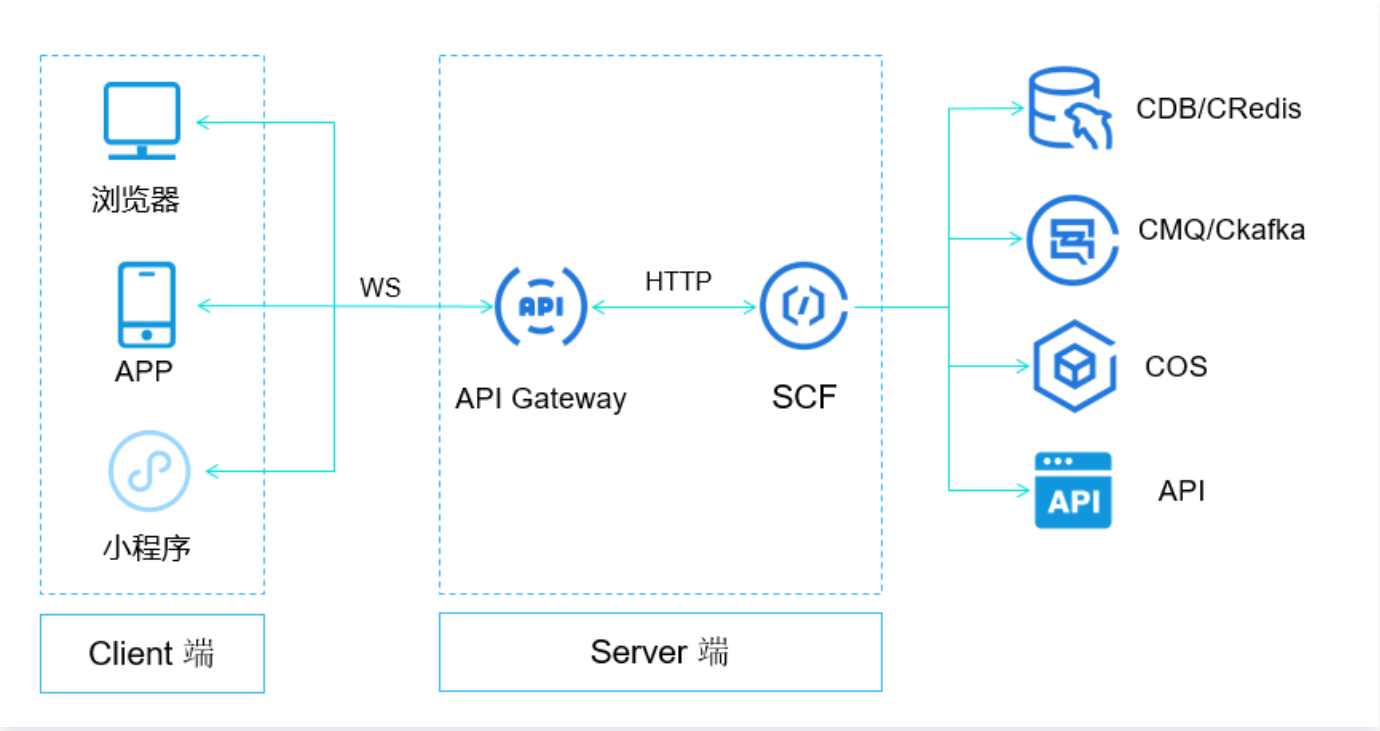
Websocket 原理介绍

最近更新时间：2022-07-26 10:25:49

> **说明**
> 本文介绍**事件函数**支持 WebSocket 的解决方案，目前 **Web 函数**已经支持原生 WebSocket 协议，详情请参见 WebSocket 协议支持 。

实现原理

WebSocket 协议是基于 TCP 的一种新的网络协议。它实现了浏览器与服务器全双工（full-duplex）通信，即允许服务器主动发送信息给客户端。WebSocket 在服务端有数据推送需求时，可以主动发送数据至客户端。而原有 HTTP 协议的服务端对于需推送的数据，仅能通过轮询或 long poll 的方式来让客户端获得。
由于云函数是无状态且以触发式运行，即在有事件到来时才会被触发，因此，为了实现 WebSocket，云函数与 API 网关相结合，通过 API 网关承接及保持与客户端的连接。您可以认为 API 网关与 SCF 一起实现了服务端。当客户端有消息发出时，会先传递给 API 网关，再由 API 网关触发云函数执行。当服务端云函数要向客户端发送消息时，会先由云函数将消息 POST 到 API 网关的反向推送链接，再由 API 网关向客户端完成消息的推送。具体的实现架构如下：



对于 WebSocket 的整个生命周期，主要由以下几个事件组成：

- 连接建立：客户端向服务端请求建立连接并完成连接建立。
- 数据上行：客户端通过已经建立的连接向服务端发送数据。
- 数据下行：服务端通过已经建立的连接向客户端发送数据。
- 客户端断开：客户端要求断开已经建立的连接。
- 服务端断开：服务端要求断开已经建立的连接。

对于 WebSocket 整个生命周期的事件，云函数和 API 网关的处理过程如下：

- 连接建立：客户端与 API 网关建立 WebSocket 连接，API 网关将连接建立事件发送给 SCF。
- 数据上行：客户端通过 WebSocket 发送数据，API 网关将数据转发送给 SCF。
- 数据下行：SCF 通过向 API 网关指定的推送地址发送请求，API 网关收到后会将数据通过 WebSocket 发送给客户端。
- 客户端断开：客户端请求断开连接，API 网关将连接断开事件发送给 SCF。
- 服务端断开：SCF 通过向 API 网关指定的推送地址发送断开请求，API 网关收到后断开 WebSocket 连接。

因此，API 网关与 SCF 之间的交互，需要由3类云函数来承载：

- 注册函数：在客户端发起和 API 网关之间建立 WebSocket 连接时触发该函数，通知 SCF WebSocket 连接的 secConnectionID。通常会在该函数记录 secConnectionID 到持久存储中，用于后续数据的反向推送。
- 清理函数：在客户端主动发起 WebSocket 连接中断请求时触发该函数，通知 SCF 准备断开连接的 secConnectionID。通常会在该函数清理持久存储中记录的该 secConnectionID。
- 传输函数：在客户端通过 WebSocket 连接发送数据时触发该函数，告知 SCF 连接的 secConnectionID 以及发送的数据。通常会在该函数处理业务数据。例如，是否将数据推送给持久存储中的其他 secConnectionID。

> **注意**
> 当您需要主动给某个 secConnectionID 推送数据或主动断开某个 secConnectionID 时，均需要用到 API 网关的反向推送地址。

数据结构

连接建立

1. 当客户端发起 WebSocket 建立连接的请求时，API 网关会将约定好的 JSON 数据结构封装在 Body 中，并以 HTTP POST 方法发送给注册函数。您可以从函数的 event 中获取，请求的 Body 示例如下：

```
{
  "requestContext": {
    "serviceName": "testsvc",
    "path": "/test/{testvar}",
    "httpMethod": "GET",
    "requestId": "c6af9ac6-7b61-11e6-9a41-93e8deadbeef",
    "identity": {
      "secretId": "abdcdxxxxxxxsdfs"
    },
    "sourceIp": "10.0.2.14",
    "stage": "prod",
    "websocketEnable":true
  },
  "websocket":{
    "action":"connecting",
    "secConnectionID":"xawexasdfewezdfsdfeasdfffa==",
    "secWebSocketProtocol":"chat,binary",
    "secWebSocketExtensions":"extension1,extension2"
```

```
    }
}
```

数据结构内容详细说明如下：

结构名	内容
requestContext	请求来源的 API 网关的配置信息、请求标识、认证信息、来源信息。其中包括： serviceName，path，httpMethod：指向 API 网关的服务、API 的路径和方法。 stage：指向请求来源 API 所在的环境。 requestId：标识当前这次请求的唯一 ID。 identity：标识用户的认证方法和认证的信息。 sourceIp：标识请求来源 IP。
websocket	建立连接的详细信息。其中包括： action：指本次请求的动作。 secConnectionID：字符串，即标识 WebSocket 连接的 ID。原始长度为128Bit，是经过 base64 编码后的字符串，共32个字符。 secWebSocketProtocol：字符串，可选字段。代表子协议列表。如果原始请求有该字段内容将传给云函数，否则该字段不出现。 secWebSocketExtensions：字符串，可选字段。代表扩展列表。如果原始请求有该字段内容将传给云函数，否则该字段不出现。

注意：
在 API 网关迭代过程中，requestContext 中的内容可能会大量增加。目前只保证数据结构内容仅增加，不删除，且不对已有结构进行破坏。

2. 当注册函数收到连接建立的请求后，需要在函数处理结束时，将是否同意建立连接的响应消息返回至 API 网关中。响应 Body 要求为 JSON 格式，其示例如下：

```
{
    "errNo":0,
    "errMsg":"ok",
    "websocket":{
      "action":"connecting",
      "secConnectionID":"xawexasdfewezdfsdfeasdfffa==",
      "secWebSocketProtocol":"chat,binary",
      "secWebSocketExtensions":"extension1,extension2"
    }
}
```

数据结构内容详细说明如下：

结构名	内容

errNo	整型，必选项。响应错误码。errNo 为0时，表示握手成功，同意连接建立。
errMsg	字符串，必选项。错误原因。errNo 为非0时，表示生效。
websocket	连接建立的详细信息。其中： • action：指本次请求的动作。 • secConnectionID：字符串，是标识 WebSocket 连接的 ID，原始长度为128Bit，是经过 base64 编码后的字符串，共32个字符。 • secWebSocketProtocol：字符串，可选字段。为单个子协议的值。如果原始请求有该字段内容，API 网关会透传到客户端。 • secWebSocketExtensions：字符串，可选字段。为单个扩展的值。如果原始请求有该字段内容，API 网关会透传到客户端。

注意：
- SCF 请求超时默认认为连接建立失败。
- 当 API 网关收到云函数的响应消息后，优先判断 HTTP 响应码。如果响应码为200，则解析响应 Body。如果响应码为非200，则认为 SCF 出现故障，拒绝建立连接。

数据传输

上行数据传输

传输请求

当客户端通过 WebSocket 发送数据时，API 网关会把约定好的 JSON 数据结构封装在 Body 中，并以 HTTP POST 方法发送给传输函数。您可以从函数的 event 中获取，请求的 Body 示例如下：

```
{
  "websocket":{
    "action":"data send",
    "secConnectionID":"xawexasdfewezdfsdfeasdfffa==",
    "dataType":"text",
    "data":"xxx"
  }
}
```

数据结构内容详细说明如下：

参数	内容
websocket	数据传输的详细信息。
action	本次请求的动作，本文以 “data send” 为例。
secConnectionID	字符串，是标识 WebSocket 连接的 ID。原始长度为128Bit，是经过 base64 编码后的字符串，共32个字符。

dataType	传输数据的类型。 “binary”：表示二进制。 “text”：表示文本。
data	传输的数据。如果 “dataType” 是 “binary”，则为 base64 编码后的二进制流；如果 “dataType” 是 “text”，则为字符串。

传输响应

在传输函数运行结束后，会向 API 网关返回 HTTP 响应，API 网关会根据响应码做出相应的动作：

- 如果响应码为200，表示函数运行成功。
- 如果响应码为非200，表示系统故障，API 网关会主动给客户端发 FIN 包。

> **注意**
> API 网关不会处理响应 Body 中的内容。

下行数据回调

回调请求

当云函数需要向客户端推送数据或主动断开连接时，可以发起 Request 请求，把数据封装在 Body 中，以 POST 方法发送给 API 网关的反向推向地址。请求 Body 要求为 JSON 格式，其示例如下：

```
{
  "websocket":{
    "action":"data send", //向客户端发送数据
    "secConnectionID":"xawexasdfewezdfsdfeasdfffa==",
    "dataType":"text",
    "data":"xxx"
  }
}
```

```
{
  "websocket":{
    "action":"closing", //发送断开连接请求
    "secConnectionID":"xawexasdfewezdfsdfeasdfffa=="
  }
}
```

数据结构内容详细说明如下：

字段	内容
websocket	数据传输的详细信息。
action	本次请求的动作，支持内容为 “data send”、“closing” 两种：

	“data send”：为向客户端发送数据。 “closing”：为向客户端发起连接断开请求，可以不包含 "dataType" 和 "data" 内容。
secConnectionID	字符串，是标识 websocket 连接的 ID，原始长度为 128bit，是经过 base64 编码后的字符串，共32个字符。
dataType	传输数据的类型，一共两种： “binary”：表示二进制。 “text”：表示文本。
data	传输的数据： 如果 “dataType” 是 “binary”，则为 base64 编码后的二进制流。 如果 “dataType” 是 “text”，则为字符串。

回调响应

在回调结束后，可根据 API 网关的响应码判断回调的结果：

- 如果响应码为200，表示回调成功。
- 如果响应码为非200，表示系统故障，此时 API 网关会主动向客户端发 FIN 包。

同时，在响应结果中可以拿到为 JSON 格式的响应 Body，示例如下：

```
{
    "errNo":0,
    "errMsg":"ok"
}
```

数据结构内容详细说明如下：

字段	内容
errNo	整数，响应错误码。如果为0表示成功。
errMsg	字符串，错误原因。

连接清理

客户端主动断开连接

注销请求

当客户端主动发起 WebSocket 建立断开的请求时，API 网关会把约定好的 JSON 数据结构封装在 Body 中，并以 HTTP POST 方法发送给清理函数。您可以从函数的 event 中获取，请求的 Body 示例如下：

```
{
  "websocket":{
    "action":"closing",
    "secConnectionID":"xawexasdfewezdfsdfeasdfffa=="
  }
```

```
}
```

数据结构内容详细说明如下：

字段	内容
websocket	连接断开的详细信息。
action	本次请求的动作，此处为 "closing"。
secConnectionID	字符串。 是标识 WebSocket 连接的 ID。原始长度为128Bit，是经过 base64 编码后的字符串，共32个字符。

> **注意**
> 在清理函数中，您可以从 event 中获取 secConnectionID，并在永久存储（如数据库）中删除该 ID。

注销响应

在清理函数运行结束后，会向 API 网关返回 HTTP 响应，API 网关会根据响应码做出相应的动作：

- 如果响应码为200，表示函数运行成功。
- 如果响应码为非200，表示系统故障。

> **注意**
> API 网关不会处理响应 Body 中的内容。

服务端主动断开连接

参考 下行数据回调，在函数中发起 Request 请求，将以下数据结构封装在 Body 中，并以 POST 方法发送给 API 网关的反向推向地址。

```
{
  "websocket":{
    "action":"closing", //发送断开连接请求
    "secConnectionID":"xawexasdfewezdfsdfeasdfffa=="
  }
}
```

> **注意**
> 当主动断开客户端的链接时，您需要先获取客户端 WebSocket 的 secConnectionID，并将其填写在数据结构中；再在永久存储（如数据库）中删除该 ID。

使用方法

最近更新时间：2021-11-01 14:37:24

> **说明**
> 本文介绍**事件函数**支持 WebSocket 的解决方案，目前 **Web 函数**已经支持原生 WebSocket 协议，详情请参见 WebSocket 协议支持 。

在 原理介绍 章节中，提到需要3类云函数来承载与 API 网关之间的交互：

- 注册函数：在客户端发起和 API 网关之间建立 WebSocket 连接时触发该函数，通知 SCF WebSocket 连接的 secConnectionID。通常会在该函数记录 secConnectionID 到持久存储中，用于后续数据的反向推送。
- 清理函数：在客户端主动发起 WebSocket 连接中断请求时触发该函数，通知 SCF 准备断开连接的 secConnectionID。通常会在该函数清理持久存储中记录的该 secConnectionID。
- 传输函数：在客户端通过 WebSocket 连接发送数据时触发该函数，告知 SCF 连接的 secConnectionID 以及发送的数据。通常会在该函数处理业务数据。例如，是否将数据推送给持久存储中的其他 secConnectionID。

> **注意**
> 当您需要主动给某个 secConnectionID 推送数据或主动断开某个 secConnectionID 时，均需要用到 API 网关的反向推送地址。

本文档以 Python2.7 为例，介绍各类函数 main_handler 的写法。

函数代码示例

注册函数

```
# -*- coding: utf8 -*-
import json
import requests
def main_handler(event, context):
    print('Start Register function')
    print("event is %s"%event)
    retmsg = {}
    global connectionID
    if 'requestContext' not in event.keys():
        return {"errNo":101, "errMsg":"not found request context"}
    if 'websocket' not in event.keys():
        return {"errNo":102, "errMsg":"not found websocket"}
    connectionID = event['websocket']['secConnectionID']
    retmsg['errNo'] = 0
    retmsg['errMsg'] = "ok"
    retmsg['websocket'] = {
            "action":"connecting",
            "secConnectionID":connectionID
```

```
        }
    if "secWebSocketProtocol" in event['websocket'].keys():
        retmsg['websocket']['secWebSocketProtocol'] = event['websocket']
['secWebSocketProtocol']
    if "secWebSocketExtensions" in event['websocket'].keys():
        ext = event['websocket']['secWebSocketExtensions']
        retext = []
        exts = ext.split(";")
        print(exts)
        for e in exts:
            e = e.strip(" ")
            if e == "permessage-deflate":
                #retext.append(e)
                pass
            if e == "client_max_window_bits":
                #retext.append(e+"=15")
                pass
        retmsg['websocket']['secWebSocketExtensions'] = ";".join(retext)
    print("connecting \n connection id:%s"%event['websocket']
['secConnectionID'])
    print(retmsg)
    return retmsg
```

注意

在本函数中，您可以自行扩充其他业务逻辑。例如，将 secConnectionID 保存到 TencentDB 中，创建并关联聊天室等。

传输函数

```
# -*- coding: utf8 -*-
import json
import requests
g_connectionID = 'xxxx'  #转发消息到某个特定的 websocket 连接
sendbackHost = "http://set-7og8wn64.cb-beijing.apigateway.tencentyun.com/api-
xxxx" #API 网关的反向推送地址，在下一步 API 创建好后才能拿到
#主动向 Client 端推送消息
def send(connectionID,data):
    retmsg = {}
    retmsg['websocket'] = {}
    retmsg['websocket']['action'] = "data send"
    retmsg['websocket']['secConnectionID'] = connectionID
    retmsg['websocket']['dataType'] = 'text'
    retmsg['websocket']['data'] = json.dumps(data)
    print("send msg is %s"%retmsg)
    r = requests.post(sendbackHost, json=retmsg)
```

```
def main_handler(event, context):
    print('Start Transmission function')
    print("event is %s"%event)
    if 'websocket' not in event.keys():
        return {"errNo":102, "errMsg":"not found web socket"}
    for k in event['websocket'].keys():
        print(k+":"+event['websocket'][k])
    # 发送内容给某个客户端
    #connectionID = event['websocket']['secConnectionID']
    data = event['websocket']['data']
    send(g_connectionID,data)
    return event
```

注意

- 在本函数中，您可以自行扩充其他业务逻辑。例如，将本次获取的数据转发给其他保存在 TencentDB 中的 secConnectionID。
- 在 API 网关的 API 详情中，可以获取 API 网关的反向推送地址。具体操作参看 配置 API 网关 。

清理函数

```
import json
import requests
g_connectionID = 'xxxx'  #转发消息到某个特定的 websocket 连接
sendbackHost = "http://set-7og8wn64.cb-beijing.apigateway.tencentyun.com/api-xxxx" #API 网关的反向推送地址，在下一步 API 创建好后才能拿到
#主动发送断开信息
def close(connectionID):
    retmsg = {}
    retmsg['websocket'] = {}
    retmsg['websocket']['action'] = "closing"
    retmsg['websocket']['secConnectionID'] = connectionID
    r = requests.post(sendbackHost, json=retmsg)
    return retmsg
def main_handler(event, context):
    print('Start Delete function')
    print("event is %s"%event)
    if 'websocket' not in event.keys():
        return {"errNo":102, "errMsg":"not found web socket"}
    for k in event['websocket'].keys():
        print(k+":"+event['websocket'][k])
    #close(g_connectionID)
    return event
```

注意

> 在本函数中，您可以自行扩充其他业务逻辑。例如，将本次断开的 secConnectionID 从 TencentDB 中移除，或强制某个 secConnectionID 的 Client 下线。

您可前往 基于 Websocket 搭建匿名聊天室，通过实践了解云函数及 API 网关的创建及使用方法。

COS 触发器
COS 触发器说明

最近更新时间：2024-11-14 09:08:42

用户可以编写 SCF 函数来处理 COS Bucket 中的对象创建和对象删除事件。COS 可将事件发布给 SCF 函数并将事件数据作为参数来调用该函数。用户可以在 COS Bucket 中添加存储桶通知配置，该配置可标识触发函数的事件类型和希望调用的函数名称等信息。
COS 触发器具有以下特点：

- **Push 模型**
  COS 会监控指定的 Bucket 动作（事件类型）并调用相关函数，将事件数据推送给 SCF 函数。在推模型中使用 Bucket 通知来保存 COS 的事件源映射。
- **异步调用**
  COS 始终使用异步调用类型来调用函数，结果不会返回给调用方。有关调用类型的更多信息，请参见 调用类型 。

COS 触发器属性

- **COS Bucket（必选）**：配置的 COS Bucket，仅支持选择同地域下的 COS 存储桶。
- **事件类型（必选）**：支持 “文件上传” 和 “文件删除”、以及更细粒度的上传和删除事件，具体事件类型见下表。事件类型决定了触发器何时触发云函数，例如选择 “文件上传” 时，会在该 COS Bucket 中有文件上传时触发该函数。

事件类型	描述
cos:ObjectCreated:*	以下提到的所有上传事件均可触发云函数。
cos:ObjectCreated:Put	使用 Put Object 接口创建文件时触发云函数。
cos:ObjectCreated:Post	使用 Post Object 接口创建文件时触发云函数。
cos:ObjectCreated:Copy	使用 Put Object – Copy 接口创建文件时触发云函数。
cos:ObjectCreated:Append	使用 APPEND Object 接口创建文件时触发云函数。（此接口 COS 侧已下线，不推荐使用）
cos:ObjectCreated:CompleteMultipartUpload	使用 CompleteMultipartUpload 接口创建文件时触发云函数。
cos:ObjectCreated:Origin	通过 COS 回源 创建对象时触发云函数。
cos:ObjectCreated:Replication	通过跨区域复制创建对象时触发云函数。
cos:ObjectRemove:*	以下提到的所有删除事件均可触发云函数。
cos:ObjectRemove:Delete	在未开启版本管理的 Bucket 下使用 Delete Object 接口删除的 Object，或者使用 versionid 删除指定版本的 Object 时触发云函数。

cos:ObjectRemove:DeleteMarkerCreated	在开启或者暂停版本管理的 Bucket 下使用 Delete Object 接口删除的 Object 时触发云函数。
cos:ObjectRestore:Post	创建了归档恢复的任务时触发云函数。
cos:ObjectRestore:Completed	完成归档恢复任务时触发云函数。
cos:TaskComplete:LifecycleCompleted	完成生命周期配置时触发云函数。
cos:TaskComplete:BatchTaskCompleted	完成批量处理时触发云函数。
cos:InventoryReportCreated:Put	完成清单文件生成时触发云函数。
cos:ObjectAcl:*	全部对象中任何一个被设置访问控制事件时触发云函数。
cos:ObjectAcl:Put	设置某个对象的访问控制事件时触发云函数。
cos:ObjectTagging:*	以下提到的所有对象标签事件均会触发云函数。
cos:ObjectTagging:Put	设置对象标签事件时触发云函数。
cos:ObjectTagging:Delete	删除对象标签事件时触发云函数。

- **前缀过滤（可选）**：前缀过滤通常用于过滤指定目录下的文件事件。例如，前缀过滤为 `test/`，则仅 `test/` 目录下的文件事件才可以触发函数，`hello/` 目录下的文件事件不触发函数。
- **后缀过滤（可选）**：后缀过滤通常用于过滤指定类型或后缀的文件事件。例如，后缀过滤为 `.jpg`，则仅 `.jpg` 结尾的文件的事件才可以触发函数，`.png` 结尾的文件事件不触发函数。

COS 触发器使用限制

为了避免 COS 的事件生产投递出现错误，COS 针对每个 Bucket 的每个事件（如文件上传/文件删除等）和前后缀过滤的组合，限制同一组规则只能绑定一个可触发的函数。因此，在创建 COS 触发器时，请不要针对同一个 COS Bucket 配置相同的规则。例如，您可以为函数 A 配置 test Bucket 的 “Created: * ” 事件触发（未配置过滤规则），那么该 test Bucket 的上传事件不能再绑定到其他函数，这些事件包含（Created:Put、Created:Post 等），但是您可以为函数 B 配置 test Bucket 的 “ObjectRemove” 事件触发。
当使用前后缀过滤规则时，为了保证同一个 Bucket 触发事件的唯一性，需要注意同一 Bucket 无法使用重叠前缀、重叠后缀或前缀和后缀的重叠组合为相同的事件类型定义筛选规则。例如，当您给函数 A 配置了 test Bucket 的 “Created: *” 和前缀过滤为 “Log” 的事件触发，那么该 test Bucket 下就不能再创建 “Created: *” 和前缀过滤为 “Log” 的事件触发。
目前 COS 触发器仅支持同地域 COS Bucket 事件触发，即广州区创建的 SCF 函数，在配置 COS 触发器时，仅支持选择广州区（华南）的 COS Bucket。如果您想要使用特定地域的 COS Bucket 事件来触发 SCF 函数，可以通过在对应地域下创建函数来实现。
COS 触发器有 SCF 侧和 COS 侧两个维度限制：

- SCF 侧限制：云函数单函数最多可关联10个 COS 触发器。
- COS 侧限制：单个 COS 存储桶最多支持关联10个触发器。

COS 触发器的事件消息结构

在指定的 COS Bucket 发生对象创建或对象删除事件时，会将类似以下的 JSON 格式事件数据发送给绑定的 SCF 函数。


```
{
    "Records": [{
        "cos": {
            "cosSchemaVersion": "1.0",
            "cosObject": {
                "url": "http://testpic-1253970026.cos.ap-
chengdu.myqcloud.com/testfile",
                "meta": {
                    "x-cos-request-id":
"NWMxOWY4MGFfMjViMjU4NjRfMTUyMVxxxxxxxxx=",
                    "Content-Type": "",
                    "x-cos-meta-mykey": "myvalue"
                },
                "vid": "",
                "key": "/1253970026/testpic/testfile",
                "size": 1029
            },
            "cosBucket": {
                "region": "cd",
                "name": "testpic",
                "appid": "1253970026"
            },
            "cosNotificationId": "unkown"
        },
        "event": {
            "eventName": "cos:ObjectCreated:*",
            "eventVersion": "1.0",
            "eventTime": 1545205770,
            "eventSource": "qcs::cos",
            "requestParameters": {
                "requestSourceIP": "192.168.15.101",
                "requestHeaders": {
                    "Authorization": "q-sign-algorithm=sha1&q-
ak=xxxxxxxxxxxxxx&q-sign-time=1545205709;1545215769&q-key-
time=1545205709;1545215769&q-header-list=host;x-cos-storage-class&q-url-param-
list=&q-signature=xxxxxxxxxxxxxx"
                }
            },
            "eventQueue":
"qcs:0:scf:cd:appid/1253970026:default.printevent.$LATEST",
            "reservedInfo": "",
            "reqid": 179398952
        }
```

```
    }]
}
```

数据结构内容详细说明如下：

结构名	内容
Records	列表结构，可能有多条消息合并在列表中。
event	记录事件信息，包括事件版本、事件源、事件名称、时间、队列信息、请求参数、请求 ID。
cos	记录事件对应的 COS 信息。
cosBucket	记录具体事件发生的 Bucket，包含 Bucket 名称、地域、所属用户 APPID。APPID 可前往 账号信息 页面获取。
cosObject	记录具体事件发生的对象，包含对象文件路径、大小、自定义元数据、访问 URL。

相关示例

以下为 Java 语言的 COS 触发器示例，您可参考示例进行使用：

```
https://github.com/tencentyun/scf-demo-
java/blob/master/src/main/java/example/Cos.java
```

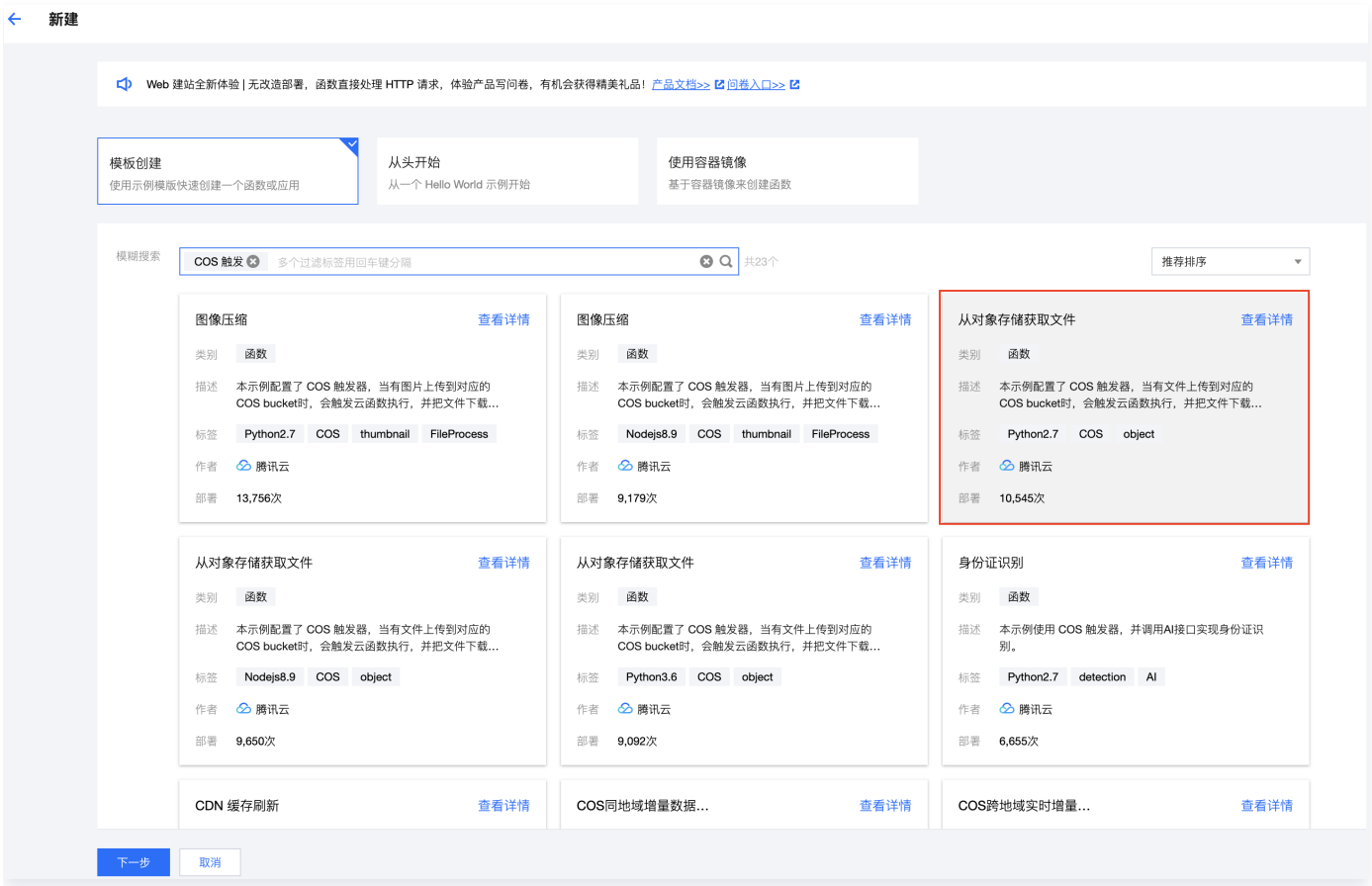
使用方法

最近更新时间：2022-12-23 14:33:16

本篇文档将为您指导，如何创建 COS 触发器并完成函数的调用。

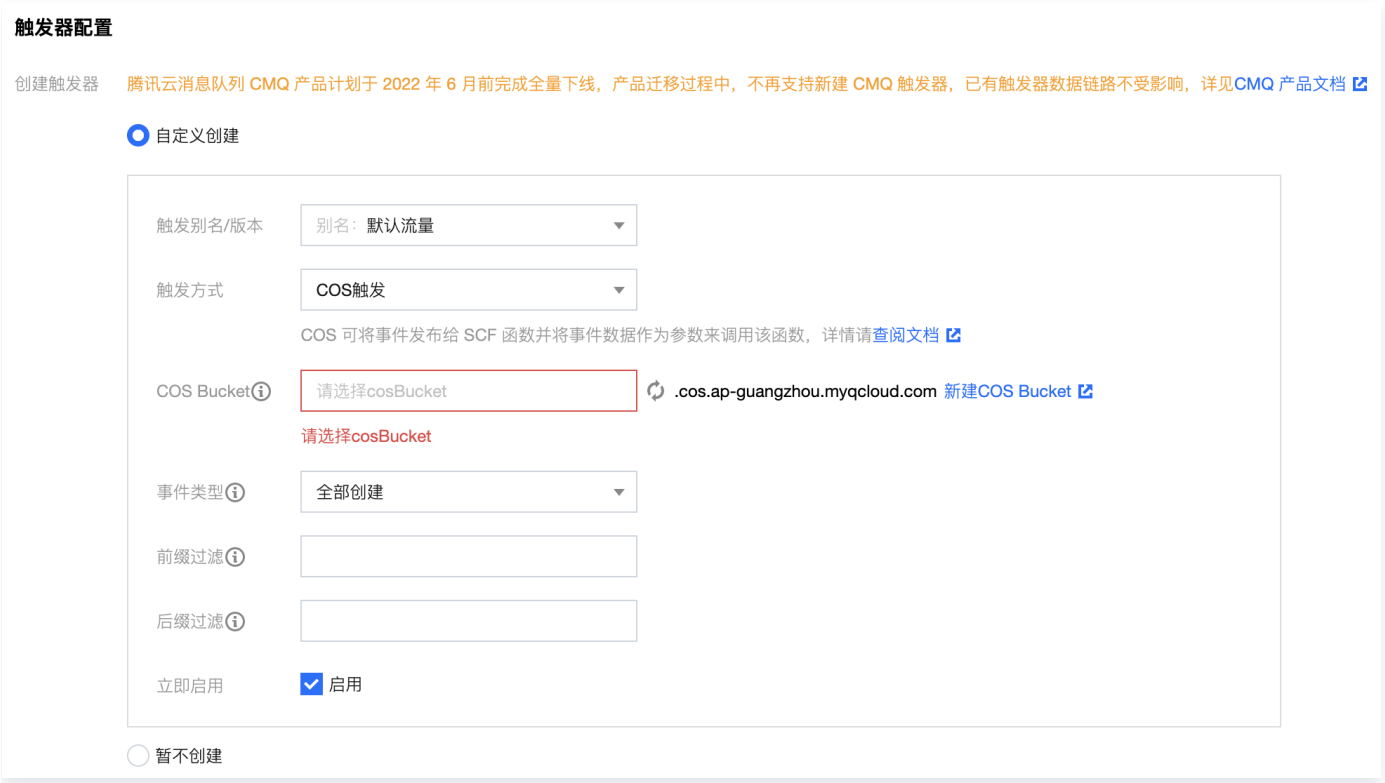
步骤1：创建函数

登录 Serverless控制台，在新建函数页面，完成您的函数代码上传与部署。详情可参见 使用控制台创建一个事件函数。此处以 COS 示例模板为例，创建函数项目，模板默认创建流程中，直接配置触发器，实际创建中，您也可以创建完成后再进行配置，此处以创建完成后配置为例进行说明：



步骤2：配置触发器

选择COS 触发器后，按照指引，配置指定 Bucket、触发事件类型等信息，即可完成触发器创建：



步骤3：管理触发器

创建完成后，在“触发器管理”页面可以看到创建的触发器信息。

CLS 触发器
CLS 触发器说明

最近更新时间：2023-08-16 20:14:41

用户可以编写云函数 SCF 来处理 CLS 日志服务中采集到的日志，通过将采集到的日志作为参数传递来调用 SCF 云函数，函数代码可以对其进行数据加工处理、分析或将其转储到其他云产品。
CLS 触发器具有以下特点：

- **Push 模型**
  CLS 会监控指定的日志主题，在一定时间内聚合并调用相关函数，将事件数据推送给 SCF 函数。
- **异步调用**
  CLS 始终使用异步调用类型来调用函数，结果不会返回给调用方。有关调用类型的更多信息，请参阅 调用类型 。

CLS 触发器属性

- **日志集**：可配置连接的 CLS 日志集，仅支持选择同地域下的日志集。
- **日志主题**：可配置连接的 CLS 日志主题，日志主题是管理配置日志服务触发器的最小单元。
- **最长等待时间**：单次事件拉取的最长等待时间。

CLS 触发器消费及消息传递

CLS 后台的消费模块在消费到消息后，会根据**最长等待时间**、**投递过程**及**消息体大小**等信息，组合为事件结构并发起函数调用（异步调用）。相关限制说明如下：

- **最长等待时间**
  目前云函数后台的消费模块的范围在3 - 300s，避免时延过长才进行消费。例如，最长等待时间设置为60s，则消费模板会60s聚合一次日志数据投递至云函数。
- **异步调用的事件大小限制**
  128KB，详情请参见 限制说明 。如果日志主题的消息较大，例如消息体在最长等待时间内通过后台组件压缩后依然达到128KB以上，由于异步调用的128KB限制，传递给云函数的事件结构中系统会依次截取通过 Gzip 压缩后达到128KB的消息体并投递，而不使用用户配置的最长等待时间的聚合。
- **投递过程**
  如果投递过程中发生错误且是不可重试的错误（例如 AccessDeniedException 或 ResourceNotFoundException），则 CLS 触发器会中断重试传输逻辑。

> **说明**
> - 在消息传递过程中，每次组合的时间聚合不一定相同，即每个事件结构内的消息个数在1 - **配置的最长等待时间**之间。如果配置的最长等待时间过大，有可能出现事件结构内的聚合时间始终不会达到最大聚合时间的情况。
> - 在云函数中获取到事件内容后，可选择循环处理的方式，确保每一条消息都得到处理，而不是假定每次传递的消息时间均是恒定的。

CLS 触发器的事件消息结构

在指定的 CLS 触发器接收到消息时，CLS 的后台消费者模块会消费消息，并将消息组装异步调用您的函数。为保证单次触发传递数据的效率，数据字段的值是 Base64 编码的 Gzip 文档。

```
{
  "clslogs": {
    "data":
"ewogICAgIm1lc3NhZ2VUeXBlIjogIkRBVEFfTUVTU0FHRSIsCiAgICAib3duZXIiOiAiMTIzNDU2Nzg
5MDEyIiwKICAgICJsb2dHcm91cCI6I..."
  }
}
```

在解码和解压缩后，日志数据类似以下 JSON 体，以 CLS Logs 消息数据（已解码）为例：

```
{
    "topic_id": "xxxx-xx-xx-xx-yyyyyyyy",
    "topic_name": "testname",
    "records": [{
        "timestamp": "1605578090000000",
        "content": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
    }, {
        "timestamp": "1605578090000000",
        "content": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
    }]
}
```

数据结构内容详细说明如下：

结构名	内容
topic_id	日志主题的 ID
topic_name	日志主题的名称
timestamp	日志生产时间，微秒级时间戳
content	日志内容

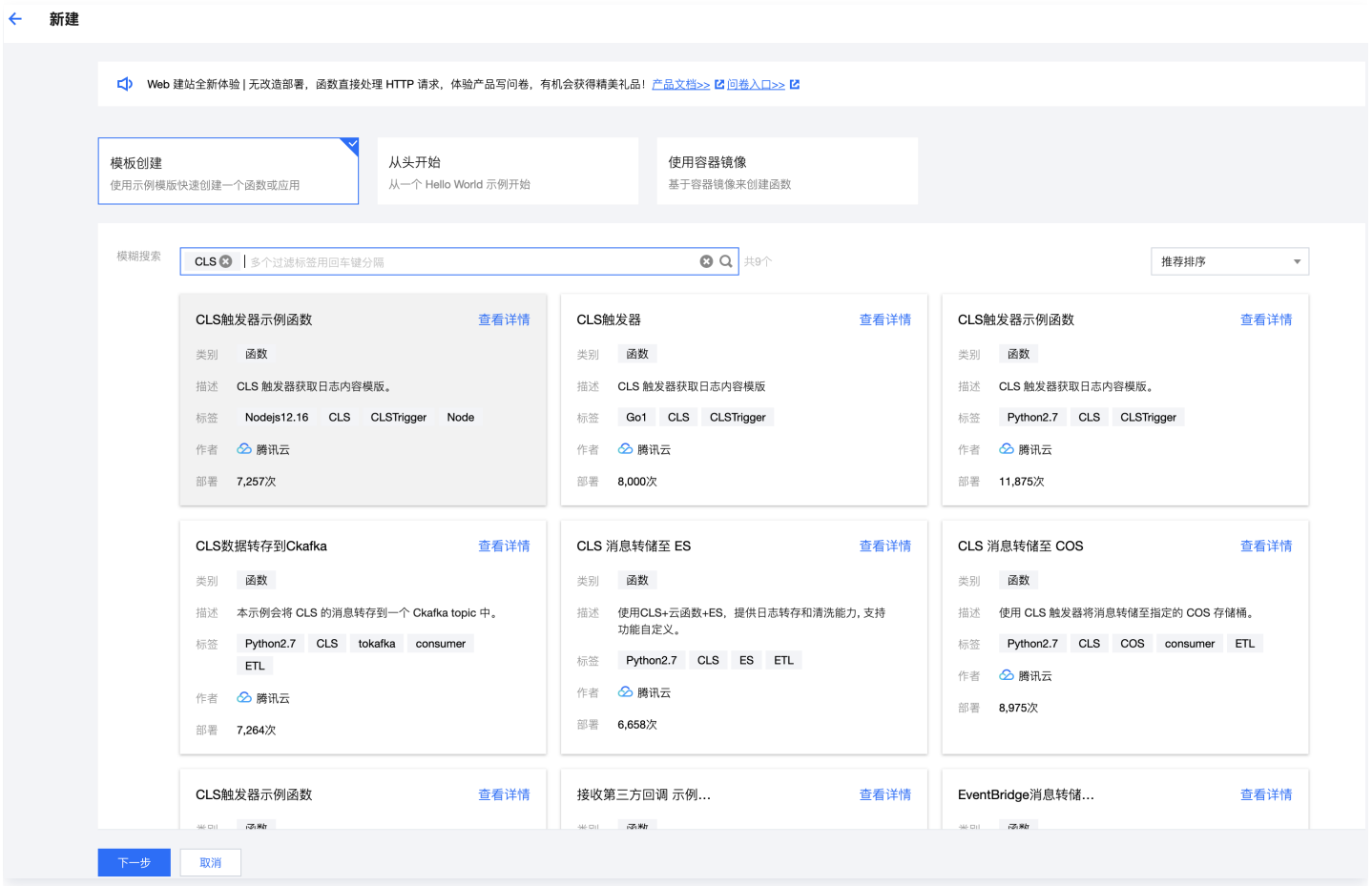
使用方法

最近更新时间：2022-12-26 11:02:26

本篇文档将为您指导，如何创建 CLS 触发器并完成函数的调用。

步骤1：创建函数

登录 Serverless控制台，在新建函数页面，完成您的函数代码上传与部署。详情可参见 使用控制台创建一个事件函数。此处以 CLS 示例模板为例，创建函数项目，模板默认创建流程中，直接配置触发器。在实际创建过程中，您也可以创建完成后再进行配置，此处以创建完成后配置触发器为例进行说明：



步骤2：配置触发器

选择CLS 触发器后，按照指引，配置指定日志集、日志主题等信息，即可完成触发器创建：

步骤3：管理触发器

创建完成后，在“触发器管理”页面可以看到创建的触发器信息，并完成触发器的开启、关闭等操作。

定时触发器
定时触发器说明

最近更新时间：2023-08-24 18:16:53

用户可以编写 SCF 函数来处理定时任务（支持秒级触发）。定时器会在指定时间自动触发 SCF 函数。定时触发器具有以下特点：

- **Push 模型**：定时器指定时间到达时直接调用相关函数的 Invoke 接口来触发函数。该事件源映射关系保存在 SCF 函数中。
- **异步调用**：定时器始终使用异步调用类型来调用函数，结果不会返回给调用方。有关调用类型的更多信息，请参阅 调用类型 。

定时触发器属性

- 定时器名称（必选）：最大支持60个字符，支持 `a-z` ， `A-Z` ， `0-9` ， `-` 和 `_` 。必须以字母开头，且一个函数下不支持同名的多个定时触发器。
- 触发周期（必选）：指定的函数触发时间。用户可以使用控制台上的默认值，或选择自定义标准的 Cron 表达式来决定何时触发函数。有关 Cron 表达式的更多信息，请参考下面的内容。
- 入参（可选）：最大支持4KB的字符串，可以在入口函数的 “event” 参数中获取。

Cron 表达式

定时触发器现已推出秒级触发功能，创建定时触发器时，用户能够使用标准的 Cron 表达式的形式自定义何时触发。

Cron 表达式语法

Cron 表达式有七个必需字段，按空格分隔。

第一位	第二位	第三位	第四位	第五位	第六位	第七位
秒	分钟	小时	日	月	星期	年

其中，每个字段都有相应的取值范围：

字段	值	通配符	特殊符号
秒	0 – 59的整数	, – * /	无
分钟	0 – 59的整数	, – * /	无
小时	0 – 23的整数	, – * /	无
日	1 – 31的整数（需要考虑月的天数）	, – * /	? L W
月	1 – 12的整数或 JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC	, – * /	无

星期	0 - 6的整数或 SUN,MON,TUE,WED,THU,FRI,SAT。其中0指星期日，1指星期一，以此类推。	, - * /	? L #
年	1970 - 2099的整数	, - * /	无

通配符

通配符	含义
,（逗号）	代表取用逗号隔开的字符的并集。例如：在“小时”字段中 1,2,3 表示1点、2点和3点。
-（破折号）	包含指定范围的所有值。例如：在“日”字段中，1 - 15包含指定月份的1号到15号。
*（星号）	表示所有值。在“小时”字段中，* 表示每个小时。
/（正斜杠）	指定增量。在“分钟”字段中，输入1/10以指定从第一分钟开始的每隔十分钟重复。例如，第11分钟、第21分钟和第31分钟，以此类推。

特殊符号

特殊符号	含义
?	只可使用在“日”字段、“星期”字段，表示不指定具体某日或星期几。例如：“日”字段指定为2，“星期”字段为“?”时，表示只判断日期为2日，而不指定2日为星期几。
L	只可使用在“日”字段、“星期”字段，表示“最后”。 在“日”字段，表示月份的最后一天；在“星期”字段，表示最后一个星期几，限定范围为0-6的整数，例如：5L 表示最后一个星期五。
W	只可使用在“日”字段，表示距离某日最接近的工作日（星期一到星期五）。例如：10W 表示离10日最接近的工作日，配合“月”字段，可表示 某月中最接近某日的工作日；“日”字段中，“L”与“W”组合为“LW”使用时表示月份的最后一个工作日。
#	只可使用在“星期”字段，且“#”前后必须给定数字，表示第几个星期几。“#”前的数字表示星期几，范围为0-6的整数；“#”后的数字表示第几个，范围为1-5的整数。例如：2#3 表示第三个星期二。

注意事项

1. 在 Cron 表达式中的“日”和“星期”字段同时指定具体值时，两者为“或”关系，即两者的条件分别均生效。
2. 使用“W”符号时，指定某日与其最近的工作日为同一月，不会跨月。如指定“1W”，即距离1日最接近的工作日，当1日为周六时，与1日最接近的工作日为本月3日（周一），而不是上月最后一日（周五）。

示例

下面展示了一些 Cron 表达式和相关含义的示例：

表达式	相关含义

`*/5 * * * * * *`	表示每5秒触发一次
`0 15 10 1 * ? *`	表示在每月的1日的上午10:15触发
`0 15 10 ? * MON-FRI *`	表示在周一到周五每天上午10:15触发
`0 0 10,14,16 * * * *`	表示在每天上午10点，下午2点，4点触发
`0 */30 9-17 * * * *`	表示在每天上午9点到下午5点每半小时触发
`0 0 12 ? * WED *`	表示在每个星期三中午12点触发
`0 0 0 L * * *`	表示在每月的最后一天的零时触发
`0 0 0 * 5 4L *`	表示在5月的最后一个周四的零时触发
`0 0 0 12W 6 * *`	表示在6月中最接近12日的工作日的零时触发
`0 0 0 ? * 2#3 *`	表示在每月的第三个星期二的零时触发
`0 0 0 LW * ? *`	表示在每月的最后一个工作日的零时触发

定时触发器入参说明

定时触发器在触发函数时，会把如下的数据结构封装在 event 里传给云函数。同时，定时触发器支持自定义传入 Message，缺省为空。

```
{
    "Type":"Timer",
    "TriggerName":"EveryDay",
    "Time":"2019-02-21T11:49:00Z",
    "Message":"user define msg body"
}
```

字段	含义
Type	触发器的类型，值为 Timer。
TriggerName	定时触发器的名称。最大支持60个字符，支持 `a-z` ， `A-Z` ， `0-9` ， `-` 和 `_` 。必须以字母开头，且一个函数下不支持同名的多个定时触发器。
Time	触发器创建时间，0时区。
Message	字符串类型。

使用方法

最近更新时间：2022-12-22 17:43:19

本篇文档将为您指导，如何创建定时触发器并完成函数的调用。

步骤1：创建函数

登录 Serverless控制台，在新建函数页面，完成您的函数代码上传与部署。详情可参见 使用控制台创建一个事件函数。
本文以定时事件示例模板为例，创建函数项目，模板默认创建流程中，直接配置触发器。如下图所示：

实际创建中，您也可以在项目创建完成后再进行触发器配置，操作详情见 创建触发器。

步骤2：配置触发器

在触发器配置步骤中，触发方式选择**定时触发**后，配置定时任务名称、触发周期等信息，即可完成触发器创建：

步骤3：管理触发器

创建完成后，在“触发管理”页面可以看到创建的触发器信息，并进行触发器的开启与关闭。

CKafka 触发器
CKafka 触发器说明

最近更新时间：2025-02-07 14:25:02

用户可以编写云函数来处理 CKafka 中收取到的消息。云函数后台模块可以作为消费者消费 CKafka 中的消息，并将消息传递给云函数。
CKafka 触发器具有以下特点：

- **Pull 模型**：云函数的后台模块作为消费者，连接 CKafka 实例并消费消息。在后台模块获取到消息后，会将消息封装到数据结构中并调用指定的函数，将消息数据传递给云函数。
- **同步调用**： CKafka 触发器使用同步调用类型来调用函数。有关调用类型的更多信息，请参见 调用类型 。

> **说明：**
> - 对于运行错误（含用户代码错误和 Runtime 错误），CKafka 触发器会按照您配置的重试次数进行重试。默认重试10000次。
> - 对于系统错误，CKafka 触发器会采用指数退避的方式持续重试，直至成功为止。

CKafka 触发器属性

- **CKafka 实例**：配置连接的 CKafka 实例，仅支持选择同地域下的实例。对于配置了 ACL 访问策略的 Ckafka 实例，请选择加密访问，详情请参见 配置 ACL 策略 。
- **加密信息**：选择加密访问时，需输入Ckafka实例中应用ACL策略所配置的具备访问权限的账号及密码。
- **Topic**：支持在 CKafka 实例中已经创建的 Topic。
- **Consumer Group**：支持选择 Ckafka 侧已存在的消费组或新建消费组，并支持新建消费组时，用户自定义消费组名称。
- **最大批量消息数**：在拉取并批量投递给当前云函数时的最大消息数，目前支持最高配置为10000。结合消息大小、写入速度等因素影响，每次触发云函数并投递的消息数量不一定能达到最大值，而是处在1 - 最大消息数之间的一个变动值。
- **起始位置**：触发器消费消息的起始位置，默认从最新位置开始消费。支持最新、最开始、按指定时间点三种配置；选择已有消费组时，为保证消费连续性，请确认 Ckafka 侧该消费组位置已设置为“从最新位置开始消费”，否则可能导致 Offset 重置失败（Consumer Group 为 Stable 状态时，不支持重置消费组），详情请参见 设置 Offset 。
- **重试次数**：函数发生运行错误（含用户代码错误和 Runtime 错误）时的最大重试次数。
- **最长等待时间**：单次触发的最长等待时间。举例：用户配置了最大批量消息数为1000，最长等待时间为60秒。假设10秒后，云函数已经采集了1000条消息，则直接触发函数执行；假设过了60秒，云函数只采集到50条消息，也会触发函数执行。

> **注意：**
> 1. 当前已创建的 CKafka 触发器仅支持对“最大批量消息数”、“重试次数”、“最长等待时间”三个配置项进行编辑。不支持修改“加密访问”状态及“加密信息”。

2. 当使用存量消费组时，函数平台将启动消费组件，自动创建新的消费者。**强烈建议在 CKafka 触发器创建完成后，停止原有的消费实例。**否则，可能由于原消费者和新消费者所使用的 SDK 不一致，导致消费组一直处于 Rebalance 状态，无法正常消费。

CKafka 消费及消息传递

由于 CKafka 消息无主动推送能力，需要消费方通过拉取的方式，拉取到消息并进行消费。因此，在配置 CKafka 触发器后，云函数后台会通过启动 CKafka 消费模块，作为消费者，并在 CKafka 中创立独立的消费组进行消费。
云函数后台的消费模块在消费到消息后，会根据一定的**超时时间**、**累积消息数量大小**及**最大批量消息数**等信息，组合为事件结构并发起函数调用（同步调用）。相关限制说明如下：

- **超时时间**：目前云函数后台的消费模块的超时时间为60秒，避免时延过长才进行消费。例如，Ckafka Topic 的消息写入很少，消费模块在60秒内没有凑够最大批量消息数的消息，则依然会发起函数调用。
- **同步调用的事件大小限制**：6MB，详情请参见 限制说明 。如果 Ckafka Topic 的消息很大，例如单条消息就已经达到6MB，那么由于同步调用的6MB限制，所以传递给云函数的事件结构中只会有一条消息，而不是用户配置的最大消息个数。
- **最大批量消息数**：同 CKafka 触发器属性，由用户设置，目前支持最高配置为10000。
- **消费机制**：函数平台**支持 at least once（至少消费1次）**，不保证 exactly once（即不能保证数据完全不重复消费）。

云函数后台的消费模块会循环这个过程，且会保证消息消费的顺序性，即前一批消息消费完（同步调用），再进行下一批消息的消费。

说明：

- 在这个过程中，每次组合的消息数量不一定相同，即每个事件结构内的消息个数在**1 – 配置的最大消息个数**之间。如果配置的最大消息数过大，有可能出现事件结构内的消息个数始终不会达到最大消息数的情况。
- 在云函数中获取到事件内容后，可选择循环处理的方式，确保每一条消息都得到处理，而不应假定每次传递的消息个数均是恒定的。

CKafka 触发器的事件消息结构

在指定的 CKafka Topic 接收到消息时，云函数的后台消费者模块会消费到消息，并将消息组装为类似以下的 JSON 格式事件，触发绑定的函数并将数据内容作为入参传递给函数。

```
{
  "Records": [
    {
      "Ckafka": {
        "topic": "test-topic",
        "Partition":1,
        "offset":36,
        "msgKey": "None",
        "msgBody": "Hello from Ckafka!"
      }
    },
    {
```

```
        "Ckafka": {
          "topic": "test-topic",
          "Partition":1,
          "offset":37,
          "msgKey": "None",
          "msgBody": "Hello from Ckafka again!"
        }
      }
  ]
}
```

数据结构内容详细说明如下：

结构名	内容
Records	列表结构，可能有多条消息合并在列表中
Ckafka	标识事件来源为 CKafka
topic	消息来源 Topic
partition	消息来源的分区 ID
offset	消费偏移编号
msgKey	消息 key
msgBody	消息内容

常见问题

CKafka 消息堆积了很多该如何处理?

在您配置 CKafka 触发器后，云函数后台会通过启动 CKafka 消费模块作为消费者，在 CKafka 中创立独立的消费组进行消费，且消费模块的数量等于 Ckafka Topic 的分区（partition）数量。
如果堆积了很多 Ckafka 消息，则需要提升消费能力。提升消费能力有以下方法：

- 增加 Ckafka Topic 的分区数。云函数的消费能力正比于分区数量，云函数后台的 CKafka 消费模块会自动匹配 Ckafka Topic 分区数，即可以通过增加分区来提升消费能力。
- 优化云函数的运行时间。云函数的运行时间越短，消费能力就越强。若云函数的运行时间变长（例如，云函数内需要写 DB，DB 的响应变慢），则消费速度就会下降。

使用方法

最近更新时间：2022-07-13 09:59:25

本篇文档将为您指导，如何创建 Ckafka 触发器并完成函数的调用。

步骤1：创建函数

1. 登录 Serverless 控制台 。
2. 在新建函数页面，选择使用**模板创建**来新建函数，在搜索框里筛选 `Ckafka` ，选择“Ckafka 消息转储至 COS”。如下图所示：

3. 单击**下一步**。

步骤2：配置触发器

在函数配置页面，填写函数基础配置。

1. 在“触发器配置”中，选择使用**自定义创建**来新建触发器。如下图所示：

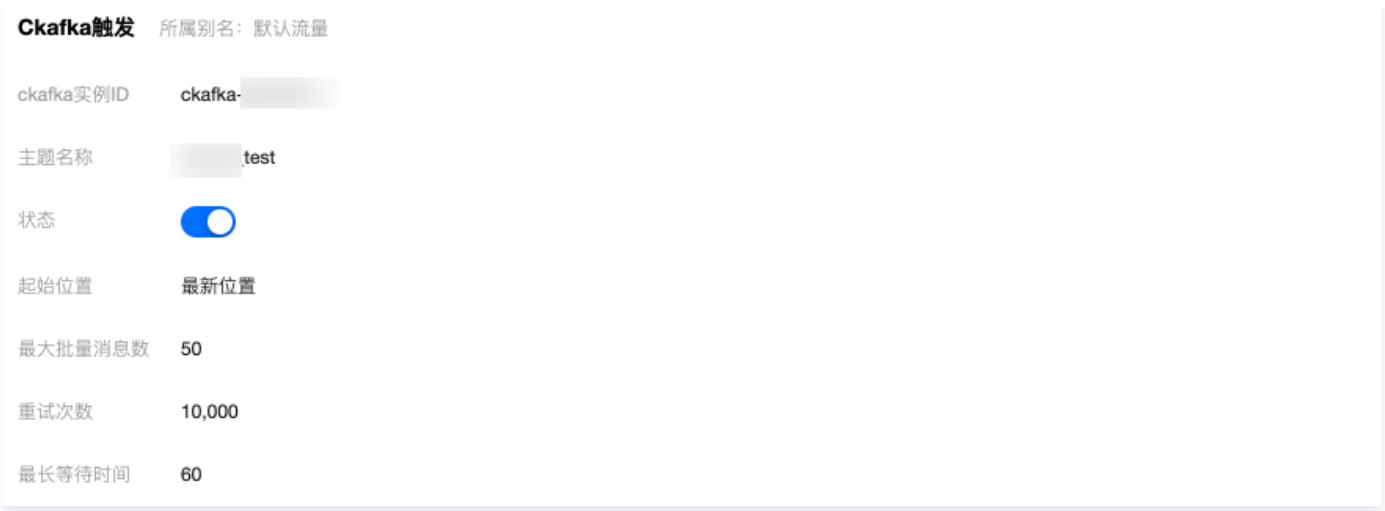
选择 **Ckafka 触发器**后，按照指引，配置消息来源的 Ckafka 实例的名称、主题等信息。您也可以选择 新建Ckafka 。

> **注意**
> 请保证您的函数与 Ckafka 在相同 VPC 下。

2. 单击**完成**。

步骤3：管理触发器

函数创建完成后，前往函数详情页，您可以在“触发管理”中查看已创建的触发器信息。您还可以开启或关闭触发器。



Apache Kafka 触发器
Apache Kafka 触发器说明

最近更新时间：2025-02-07 14:25:02

云函数 SCF 现已支持将 Apache Kafka 作为事件触发源，实现 Kafka 消息的批量消费和处理。

Apache Kafka 简介

Apache Kafka 是一个开源事件流平台，支持数据管道和流分析等工作负载。云函数 SCF 支持将业务函数与自建的 Apache Kafka 集群结合使用，支持的 Kafka 集群包括跨地域的 CKafka 集群、其他云厂商上的 Kafka 集群或通过 Confluent Cloud 等程序管理的类 Kafka 集群（如 Azure 的 EventHub）。
云函数 SCF 支持基于 Kafka 协议框架的事件源，支持批量消费，并可通过最大批量消息数、最长等待时间、重试次数等参数控制批处理行为。

自建 Apache Kafka 触发器特点

自建 Apache Kafka 触发器具有以下特点：

- **Pull 模型**：云函数的后台模块作为消费者，连接 Kafka 实例并消费消息。在后台模块获取到消息后，会将消息封装到数据结构中并调用指定的函数，将消息数据传递给云函数。
- **同步调用**：自建 Apache Kafka 触发器使用同步调用类型来调用函数。有关调用类型的更多信息，请参见 调用类型 。

> **说明：**
> - 对于运行错误（含用户代码错误和 Runtime 错误），自建 Apache Kafka 触发器会按照您配置的重试次数进行重试。默认重试10000次。
> - 对于系统错误，自建 Apache Kafka 触发器会采用指数退避的方式持续重试，直至成功为止。

自建 Apache Kafka 触发器属性

- **触发器名称**：支持2~60个字符，支持 `a-z` ， `A-Z` ， `0-9` ， `-` 和 `_` 。必须以字母开头，以数字或字母结尾，且一个函数下不支持同名的多个定时触发器。
- **Bootstrap Servers**：配置需连接消费的自建 Apache Kafka 实例地址，支持多个 Bootstrap Servers，支持 `IP+端口` 或 `域名+端口` 。
- **Topic**：输入已创建的 Apache Kafka 实例的Topic。
- **Consumer Group**：选择已创建的 Apache Kafka 实例的 Consumer Group。如果指定的消费组不存在，函数将自动创建新消费组。推荐使用独立消费组，不和已有的业务混用，以免影响已有的消费收发。
- **安全协议**：Apache Kafka 实例所使用的安全协议，当前支持 `PLAINTEXT` 、 `SASL_SSL` 、 `SASL_PLAINTEXT` 。
- **身份验证机制**：Apache Kafka 实例所使用的身份验证机制，当前支持 `无` 、 `PLAIN` 、 `SCRAM-SHA-256` 、 `SCRAM-SHA-256` ，若您的实例未设置身份验证，可选择 `无` 。
- **用户名及密码**：选择了身份验证机制的情况下，需输入允许访问该实例的用户名及密码。
- **最大批量消息数**：拉取并批量投递给云函数时的最大消息数，目前支持最高配置为10000。结合消息大小、写入速度等因素影响，每次触发云函数并投递的消息数量不一定能达到最大值，而是处在1 - 最大消息数之间的一个变动值。

- **起始位置**：触发器消费消息的起始位置，当前支持从最新位置开始消费。
- **重试次数**：函数发生运行错误（含用户代码错误和 Runtime 错误）时的最大重试次数。
- **最长等待时间**：单次触发的最长等待时间。示例：用户配置了最大批量消息数为1000，最长等待时间为60秒。假设10秒后，云函数已经采集了1000条消息，则直接触发函数执行；假设过了60秒，云函数只采集到50条消息，也会触发函数执行。

> **注意：**
> - 当前已创建的自建 Apache Kafka 触发器仅支持对“最大批量消息数”、“重试次数”、“最长等待时间”三个配置项进行编辑。
> - 如需在专区（金融专区、自动驾驶专区等）地域使用此能力，请 提交工单 与我们联系以获取支持。其他地域当前仅支持对通过公网访问的 Kafka 实例进行消费，如您的 Kafka 实例需在 VPC 环境中使用，则暂不适用此能力。

自建 Apache Kafka 消费及消息传递

由于自建 Apache Kafka 消息无主动推送能力，需要消费方通过拉取方式进行消费。因此，在配置自建 Apache Kafka 触发器后，云函数后台会通过启动自建 Apache Kafka 消费模块，作为消费者，并在自建 Apache Kafka 中创立独立的消费组进行消费。
云函数后台的消费模块在消费到消息后，会根据一定的**超时时间**、**累积消息数量大小**及**最大批量消息数**等信息，组合为事件结构并发起函数调用（同步调用）。相关限制说明如下：

- **超时时间**：目前云函数后台的消费模块的超时时间为60秒，避免时延过长才进行消费。例如，Topic 的消息写入很少，消费模块在60秒内没有凑够最大批量消息数的消息，则依然会发起函数调用。
- **同步调用的事件大小限制**：6MB，详情请参见 限制说明 。如果 Topic 的消息很大，例如单条消息就已经达到6MB，由于同步调用的6MB限制，传递给云函数的事件结构中只会有一条消息，而不是用户配置的最大消息个数。
- **最大批量消息数**：同自建 Apache Kafka 触发器属性，由用户设置，目前支持最高配置为10000。
- **消费机制**：函数平台**支持 at least once（至少消费1次）**，不保证 exactly once（即不能保证数据完全不重复消费）。

云函数后台的消费模块会循环这个过程，且会保证消息消费的顺序性，即前一批消息消费完（同步调用），再进行下一批消息的消费。

> **说明：**
> - 在这个过程中，每次组合的消息数量不一定相同，即每个事件结构内的消息个数在**1 – 配置的最大消息个数**之间。如果配置的最大消息数过大，有可能出现事件结构内的消息个数始终不会达到最大消息数的情况。
> - 在云函数中获取到事件内容后，可选择循环处理的方式，确保每一条消息都得到处理，而不应假定每次传递的消息个数均是恒定的。
> - 云函数会使用标准 Kafka 协议获取您所指定的 Topic 下的分区数，同时后台消费模块自动创建相同数量的消费者，若未获取到对应的分区数，将默认创建20个消费者。

常见问题

自建 Apache Kafka 消息堆积了很多该如何处理?

在您配置自建 Apache Kafka 触发器后，云函数后台会通过启动消费模块作为消费者，在自建 Apache Kafka 中创立独立的消费组进行消费，且消费模块的数量等于 Topic 的分区（partition）数量。如果堆积了很多消息，则需要提升消费能力。
提升消费能力的方法如下：
优化云函数的运行时间。云函数的运行时间越短，消费能力就越强。若云函数的运行时间变长（例如，云函数内需要写 DB，DB 的响应变慢），则消费速度就会下降。

使用方法

最近更新时间：2024-12-02 21:19:43

本文档以使用自建 Apache Kafka 触发器跨地域消费 CKafka 集群为例，指导您如何创建自建 Apache Kafka 触发器并完成函数的调用。

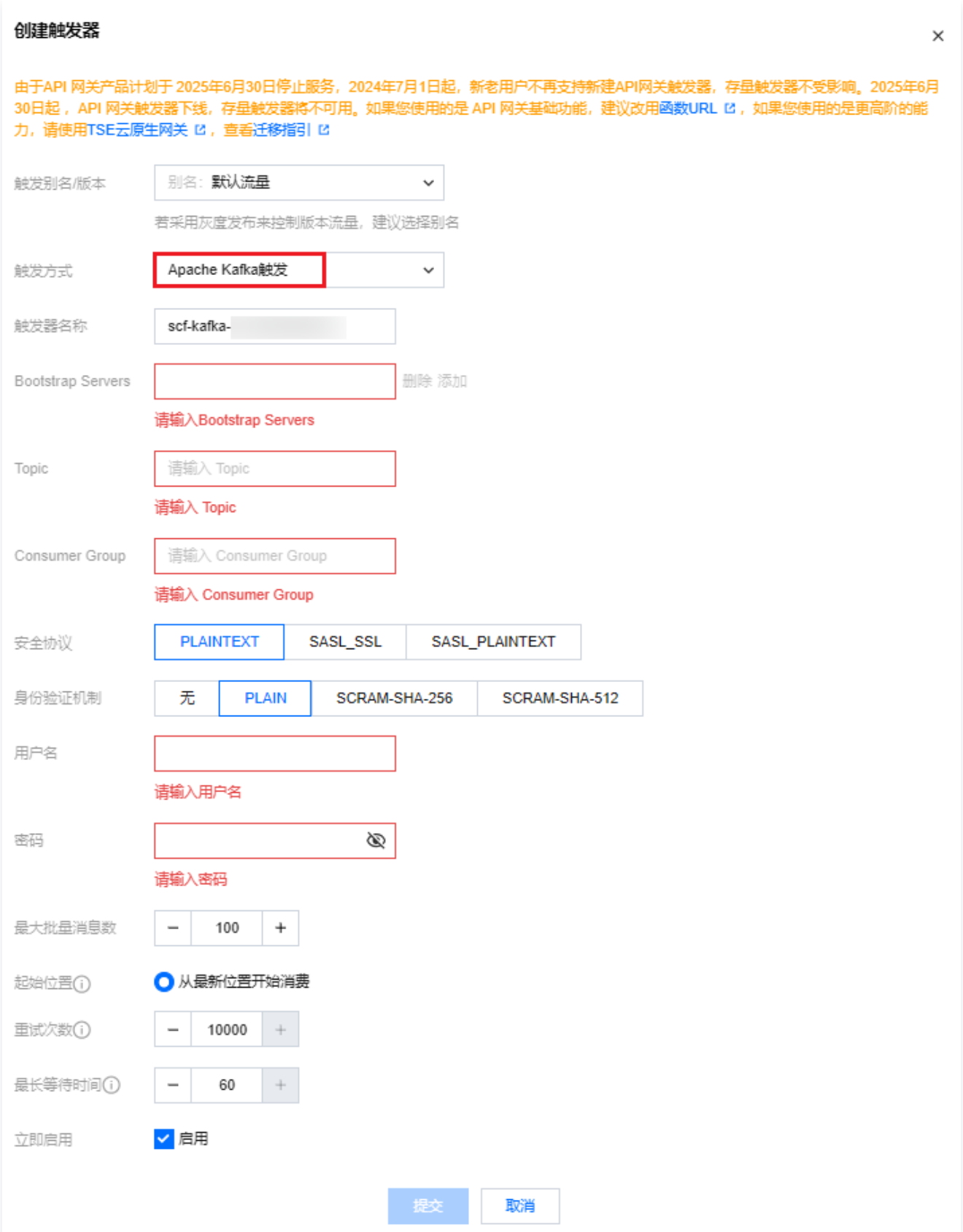
前提条件

- 已完成函数创建。
- 已完成 Kafka 集群及 Topic 的创建。

操作步骤

步骤1：创建 Apache Kafka 触发器

1. 登录 Serverless 控制台，单击左侧导航栏的**函数服务**。
2. 在主界面上方选择函数所在地域和命名空间，单击列表中的函数名称，进入函数详情页面。
3. 在左侧导航中选择**触发管理**，单击**创建触发器**。
4. 在创建触发器面板，选择 **Apache Kafka 触发**，并填写触发器相关信息。如下图所示：



配置项	操作	本文示例
触发版本/别名	默认值为 Default，也可以切换至函数已发布的别名或	Default

	其他版本。	
触发方式	Apache Kafka 触发。	Apache Kafka 触发
触发器名称	填写自定义的触发器名称。	scf-kafka-1728981649432
Bootstrap Servers	所需访问 Kafka 实例的主机和端口地址，允许创建多个。	11.135.x.x:7661
Topic	选择已创建的 Kafka 实例的 Topic。	test1015
Consumer Group	所需使用的消费组名称，若主题下已存在此消费组则使用此消费组继续消费，不存在时进行新建。	此处使用新建消费组，test1015
安全协议	Kafka 实例所应用的安全协议。取值说明如下： • PLAINTEXT • SASL_SSL • SASL_PLAINTEXT	SASL_PLAINTEXT
身份验证机制	此 Kafka 实例所使用的身份验证机制。取值说明如下： • 无 • PLAIN • SCRAM-SHA-256 • SCRAM-SHA-512	PLAIN
用户名	身份验证机制中需通过用户名、密码信息验证时，需要配置 Apache Kafka 用户名用于身份验证。	admin
密码	身份验证机制中需通过用户名、密码信息验证时，需要配置 Apache Kafka 密码用于身份验证。	******
最大批量消息数	在拉取并批量投递给当前云函数时的最大消息数，目前支持最高配置为10000。结合消息大小、写入速度等因素影响，每次触发云函数并投递的消息数量不一定能达到最大值，而是处在1 - 最大消息数之间的一个变动值。	1
起始位置	选择消息的消费位点，当前支持从最新位置开始消费。	最新位置
重试次数	函数发生运行错误（含用户代码错误和 Runtime 错误）时的最大重试次数，最高支持配置10000。	1
最长等待时间	单次触发的最长等待时间。示例：用户配置了最大批量消息数为1000，最长等待时间为60秒。假设10秒后，云函数已经采集了1000条消息，则直接触发函数执行；假设过了60秒，云函数只采集到50条消息，也会触发函数执行。	1
触发器启用状态	创建触发器后是否立即启用。默认勾选启用触发器，即	启用触发器

	创建触发器后立即启用触发器。	

5. 单击**提交**。

步骤2：检查触发器状态

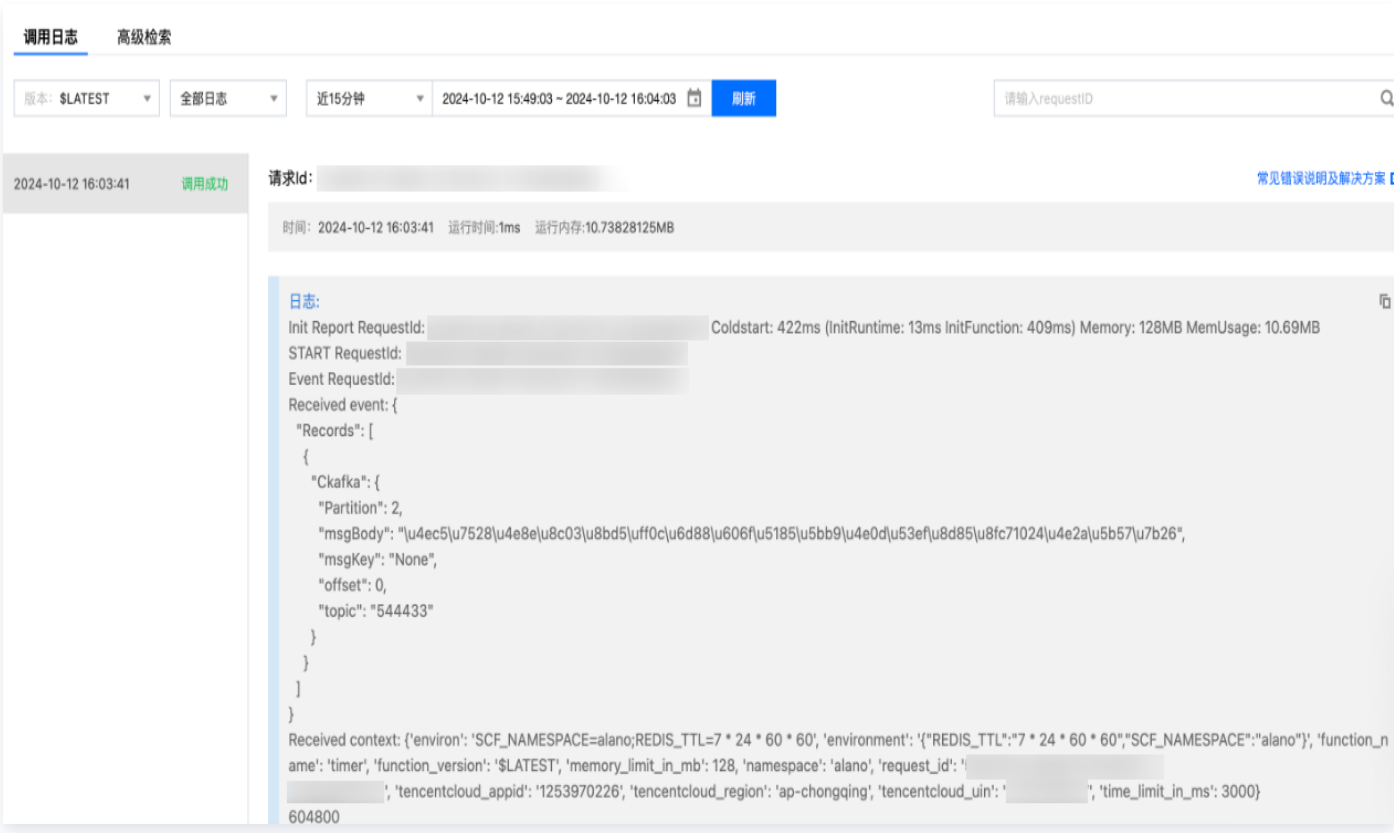
触发器创建成功后，在触发管理页面，检查触发器状态。如下图所示：

系统会自动新建消费组并绑定好订阅关系。如下图所示：

步骤3：Kafka 消息消费及测试

1. 投递消息到 Kafka 中。如下图所示：

2. 在函数详情页面，选择日志查询页签，并查询函数执行记录，此时可看到已成功消费并触发函数运行。



MQTT 触发器

最近更新时间：2025-06-18 15:57:32

您可以通过编写云函数来处理 MQTT 中收取到的消息。云函数后台模块可以作为消费者消费 MQTT 中的消息，并将消息传递给云函数。
MQTT 触发器具有以下特点：

- **Pull 模型**：云函数的后台模块作为消费者，连接 MQTT 实例并消费消息。后台模块获取到消息后，会将消息封装到数据结构中并调用指定的函数，将消息数据传递给云函数。
- **同步调用**：MQTT 触发器使用同步调用类型来调用函数。更多信息请参见 调用类型 。

> **注意：**
> 由于当前消息队列 MQTT 版产品仅在部分地域开放，因此不同地域下的函数对 MQTT 触发器的开放程度存在差异，地域下是否支持创建 MQTT 触发器以控制台实际展示为准。

MQTT 触发器属性

- **触发器名称**：最大支持60个字符，支持 `a-z` ， `A-Z` ， `0-9` ， `-` 和 `_` 。必须以字母开头，且一个函数下不支持同名的多个 MQTT 触发器。
- **MQTT 实例**：配置连接的 MQTT 实例，仅支持选择同地域下的实例。
- **订阅主题**：提供手动选择 Topic 及自定义 Topic Filter 两种方式声明需订阅 MQTT 主题。
- **手动选择 Topic 及子 Topic**：选择所选实例下，已创建的 Topic 信息，仅支持自建 Topic，无法订阅 MQTT 消息队列内置 Topic。
- **自定义 Topic Filter**：支持订阅自建 Topic 及 MQTT 消息队列内置 Topic，可使用 “+”、“#” 通配符表达更多语义。
- **消费方式**：顺序消费或非顺序消费，当前仅支持非顺序消费。
- **加密信息**：需配置在 MQTT 实例 ACL 策略中具备访问权限的账号和密码。
- **开启 Base64 标准编码**：会自动将您的消息内容进行 Base64 编码处理。
- **最大批量消息数**：在拉取并批量投递给当前云函数时的最大消息数，默认为1，最高配置为10000。结合消息大小、写入速度等因素影响，每次触发云函数并投递的消息数量不一定能达到最大值，而是处在1 – 最大消息数之间的一个变动值。
- **重试次数**：函数发生运行错误（含用户代码错误和 Runtime 错误）时的最大重试次数，默认为3。
- **最长等待时间**：单次触发的最长等待时间。例如：用户配置了最大批量消息数为1000，最长等待时间为60秒。假设10秒后，云函数已经采集了1000条消息，则直接触发函数执行；假设过了60秒，云函数只采集到50条消息，也会触发函数执行。

MQTT 消费及消息传递

由于 MQTT 消息无主动推送能力，需要消费方通过拉取的方式，拉取到消息并进行消费。因此，在配置 MQTT 触发器后，云函数后台会通过启动 MQTT 消费模块，作为消费者，并在 MQTT 中创立消费客户端进行消费。
云函数后台的消费模块在消费到消息后，会根据一定的**累积消息数量大小**及**最大批量消息数**等信息，组合为事件结构并发起函数调用（同步调用）。相关限制说明如下：

- **同步调用的事件大小限制**：6MB，详情请参见 限制说明 。如果 Topic 的消息很大，例如单条消息就已经达到6MB，那么由于同步调用的6MB限制，传递给云函数的事件结构中只会有一条消息，而不是用户配置的最大消息个数。
- **最大批量消息数**：用户可配置，目前支持最高配置为10000。

MPS 触发器

最近更新时间：2023-03-16 16:45:18

视频处理（Media Processing Service，MPS）是针对海量多媒体数据，提供的云端转码和音视频处理服务。您可以编写云函数来处理 MPS 中的回调信息，通过接收相关回调帮助转储、投递和处理视频任务中的相关事件与后续内容。
MPS 触发器具有以下特点：

- **Push 模型**：MPS 触发器会监听视频处理的回调信息，并通过单次触发的方式将事件数据推送至 SCF 函数。
- **异步调用**：MSP 触发器始终使用异步调用类型来调用函数，结果不会返回给调用方。有关调用类型的更多信息，请参见 调用类型 。

MPS 触发器属性

- **事件类型**：MPS 触发器以账号维度的事件类型推送 Event 事件，目前支持工作流任务（WorkflowTask）和视频编辑任务（EditMediaTask）两种事件类型触发。
- **事件处理**：MPS 触发器以服务维度产生的事件作为事件源，不区分地域、资源等属性。每个账号全地域只能创建一个 MPS 触发器。如需多个函数并行处理任务，请参见 函数间调用 SDK 。

MPS 触发器的事件消息结构

在指定的 MPS 触发器接收到消息时，事件结构与字段以 WorkflowTask 为例，示例如下：

```
{
    "EventType":"WorkflowTask",
    "WorkflowTaskEvent":{
        "TaskId":"245****654-WorkflowTask-f46dac7fe2436c47******d71946986t0",
        "Status":"FINISH",
        "ErrCode":0,
        "Message":"",
        "InputInfo":{
            "Type":"COS",
            "CosInputInfo":{
                "Bucket":"macgzptest-125****654",
                "Region":"ap-guangzhou",
                "Object":"/dianping2.mp4"
            }
        },
        "MetaData":{
            "AudioDuration":11.261677742004395,
            "AudioStreamSet":[
                {
                    "Bitrate":127771,
                    "Codec":"aac",
                    "SamplingRate":44100
                }
            ],
```

```
            "Bitrate":2681468,
            "Container":"mov,mp4,m4a,3gp,3g2,mj2",
            "Duration":11.261677742004395,
            "Height":720,
            "Rotate":90,
            "Size":3539987,
            "VideoDuration":10.510889053344727,
            "VideoStreamSet":[
                {
                    "Bitrate":2553697,
                    "Codec":"h264",
                    "Fps":29,
                    "Height":720,
                    "Width":1280
                }
            ],
            "Width":1280
        },
        "MediaProcessResultSet":[
            {
                "Type":"Transcode",
                "TranscodeTask":{
                    "Status":"SUCCESS",
                    "ErrCode":0,
                    "Message":"SUCCESS",
                    "Input":{
                        "Definition":10,
                        "WatermarkSet":[
                            {
                                "Definition":515247,
                                "TextContent":"",
                                "SvgContent":""
                            }
                        ],
                        "OutputStorage":{
                            "Type":"COS",
                            "CosOutputStorage":{
                                "Bucket":"gztest-125****654",
                                "Region":"ap-guangzhou"
                            }
                        },
                        "OutputObjectPath":"/dasda/dianping2_transcode_10",

"SegmentObjectName":"/dasda/dianping2_transcode_10_{number}",
                        "ObjectNumberFormat":{
                            "InitialValue":0,
                            "Increment":1,
```

```
                        "MinLength":1,
                        "PlaceHolder":"0"
                    }
                },
                "Output":{
                    "OutputStorage":{
                        "Type":"COS",
                        "CosOutputStorage":{
                            "Bucket":"gztest-125****654",
                            "Region":"ap-guangzhou"
                        }
                    },
                    "Path":"/dasda/dianping2_transcode_10.mp4",
                    "Definition":10,
                    "Bitrate":293022,
                    "Height":320,
                    "Width":180,
                    "Size":401637,
                    "Duration":11.26200008392334,
                    "Container":"mov,mp4,m4a,3gp,3g2,mj2",
                    "Md5":"31dcf904c03d0cd78346a12c25c0acc9",
                    "VideoStreamSet":[
                        {
                            "Bitrate":244608,
                            "Codec":"h264",
                            "Fps":24,
                            "Height":320,
                            "Width":180
                        }
                    ],
                    "AudioStreamSet":[
                        {
                            "Bitrate":48414,
                            "Codec":"aac",
                            "SamplingRate":44100
                        }
                    ]
                }
            },
            "AnimatedGraphicTask":null,
            "SnapshotByTimeOffsetTask":null,
            "SampleSnapshotTask":null,
            "ImageSpriteTask":null
        },
        {
            "Type":"AnimatedGraphics",
            "TranscodeTask":null,
```

```
                "AnimatedGraphicTask":{
                    "Status":"FAIL",
                    "ErrCode":30010,
                    "Message":"TencentVodPlatErr Or Unkown",
                    "Input":{
                        "Definition":20000,
                        "StartTimeOffset":0,
                        "EndTimeOffset":600,
                        "OutputStorage":{
                            "Type":"COS",
                            "CosOutputStorage":{
                                "Bucket":"gztest-125****654",
                                "Region":"ap-guangzhou"
                            }
                        },

"OutputObjectPath":"/dasda/dianping2_animatedGraphic_20000"
                    },
                    "Output":null
                },
                "SnapshotByTimeOffsetTask":null,
                "SampleSnapshotTask":null,
                "ImageSpriteTask":null
            },
            {
                "Type":"SnapshotByTimeOffset",
                "TranscodeTask":null,
                "AnimatedGraphicTask":null,
                "SnapshotByTimeOffsetTask":{
                    "Status":"SUCCESS",
                    "ErrCode":0,
                    "Message":"SUCCESS",
                    "Input":{
                        "Definition":10,
                        "TimeOffsetSet":[

                        ],
                        "WatermarkSet":[
                            {
                                "Definition":515247,
                                "TextContent":"",
                                "SvgContent":""
                            }
                        ],
                        "OutputStorage":{
                            "Type":"COS",
                            "CosOutputStorage":{
```

```
                                "Bucket":"gztest-125****654",
                                "Region":"ap-guangzhou"
                            }
                        },

"OutputObjectPath":"/dasda/dianping2_snapshotByOffset_10_{number}",
                        "ObjectNumberFormat":{
                            "InitialValue":0,
                            "Increment":1,
                            "MinLength":1,
                            "PlaceHolder":"0"
                        }
                    },
                    "Output":{
                        "Storage":{
                            "Type":"COS",
                            "CosOutputStorage":{
                                "Bucket":"gztest-125****654",
                                "Region":"ap-guangzhou"
                            }
                        },
                        "Definition":0,
                        "PicInfoSet":[
                            {
                                "TimeOffset":0,

"Path":"/dasda/dianping2_snapshotByOffset_10_0.jpg",
                                "WaterMarkDefinition":[
                                    515247
                                ]
                            }
                        ]
                    }
                },
                "SampleSnapshotTask":null,
                "ImageSpriteTask":null
            },
            {
                "Type":"ImageSprites",
                "TranscodeTask":null,
                "AnimatedGraphicTask":null,
                "SnapshotByTimeOffsetTask":null,
                "SampleSnapshotTask":null,
                "ImageSpriteTask":{
                    "Status":"SUCCESS",
                    "ErrCode":0,
                    "Message":"SUCCESS",
```

```
                    "Input":{
                        "Definition":10,
                        "OutputStorage":{
                            "Type":"COS",
                            "CosOutputStorage":{
                                "Bucket":"gztest-125****654",
                                "Region":"ap-guangzhou"
                            }
                        },

"OutputObjectPath":"/dasda/dianping2_imageSprite_10_{number}",
                        "WebVttObjectName":"/dasda/dianping2_imageSprite_10",
                        "ObjectNumberFormat":{
                            "InitialValue":0,
                            "Increment":1,
                            "MinLength":1,
                            "PlaceHolder":"0"
                        }
                    },
                    "Output":{
                        "Storage":{
                            "Type":"COS",
                            "CosOutputStorage":{
                                "Bucket":"gztest-125****654",
                                "Region":"ap-guangzhou"
                            }
                        },
                        "Definition":10,
                        "Height":80,
                        "Width":142,
                        "TotalCount":2,
                        "ImagePathSet":[
                            "/dasda/imageSprite/dianping2_imageSprite_10_0.jpg"
                        ],

"WebVttPath":"/dasda/imageSprite/dianping2_imageSprite_10.vtt"
                    }
                }
            }
        ]
    }
}
```

WorkflowTask 事件

WorkflowTask 事件消息体详细字段如下：

```
{
    "EventType":"WorkflowTask",
    "WorkflowTaskEvent":{
        // WorkflowTaskEvent 字段
     }
}
```

WorkflowTask 数据结构及字段内容详细说明：

名称	类型	描述
TaskId	String	视频处理任务 ID。
Status	String	任务流状态，取值如下： • PROCESSING：处理中。 • FINISH：已完成。
ErrCode	Integer	已弃用，请使用各个具体任务的 ErrCode。
Message	String	已弃用，请使用各个具体任务的 Message。
InputInfo	MediaInputInfo	视频处理的目标文件信息。注意：此字段可能返回 null，表示取不到有效值。
MetaData	MediaMetaData	原始视频的元信息。注意：此字段可能返回 null，表示取不到有效值。
MediaProcessResultSet	Array of MediaProcessTaskResult	视频处理任务的执行状态与结果。
AiContentReviewResultSet	Array of AiContentReviewResult	视频内容审核任务的执行状态与结果。
AiAnalysisResultSet	Array of AiAnalysisResult	视频内容分析任务的执行状态与结果。
AiRecognitionResultSet	Array of AiRecognitionResult	视频内容识别任务的执行状态与结果。

EditMediaTask 事件

EditMediaTask 事件消息体详细字段如下：

```
{
    "EventType":"EditMediaTask",
    "EditMediaTaskEvent":{
        // EditMediaTask 字段
```

```
        }
}
```

EditMediaTask 数据结构及字段内容详细说明：

名称	类型	描述
TaskId	String	任务 ID。
Status	String	任务状态，取值如下： • PROCESSING：处理中。 • FINISH：已完成。
ErrCode	Integer	错误码0：成功；其他值：失败。
Message	String	错误信息。
Input	EditMediaTaskInput	视频编辑任务的输入。
Output	EditMediaTaskOutput	视频编辑任务的输出。注意：此字段可能返回 null，表示取不到有效值。

CLB 触发器说明

最近更新时间：2025-06-30 18:34:52

您可以通过编写云函数 SCF 来实现 Web 后端服务，并通过负载均衡 CLB 对外提供服务。负载均衡 CLB 触发器会将请求内容以参数形式传递给函数，并将函数返回作为响应返回给请求方。
CLB 触发器具有以下特点：

- **Push 模型**
  负载均衡 CLB 监听器在接收到 CLB 侧发出的请求后，如果 CLB 在后端配置了对接的云函数，该函数将会被触发运行。同时 CLB 会将请求的相关信息以 event 入参的形式发送给被触发的函数。相关信息包含了具体接收到请求的方法、请求的 path、header、query 等内容。
- **同步调用**
  CLB 触发器通过同步调用的方式来调用函数。有关调用类型的更多信息，请参阅 调用类型 。

> **说明：**
> CLB 账户分为标准账户类型和传统账户类型。传统账户类型不支持绑定 SCF，建议升级为标准账户类型。详情可参见 账户类型升级说明 。

CLB 触发器配置

CLB 触发器支持在 Serverless 控制台 或在 负载均衡控制台 中进行配置。

云函数控制台

在**云函数控制台**中，支持 在触发方式中添加 CLB 负载均衡触发器 、支持选取已有 CLB 负载均衡或新建主机路由规则、支持配置 URL 请求路径。

负载均衡控制台

在**负载均衡控制台**中配置路由规则时，后端配置可选 Cloud Function，且在选择 Cloud Function 后，即可选择与 CLB 负载均衡相同地域的云函数。在负载均衡控制台上，可以配置及管理更高阶的 CLB 负载均衡服务，例如 WAF 防护、SNI 多域名证书，弹性网卡等能力。

CLB 触发器绑定限制

- CLB 负载均衡中，一条 CLB 负载均衡路径规则仅能绑定一个云函数，但一个云函数可以被多个 CLB 负载均衡规则绑定为后端。相同路径、相同监听器及主机被视为同一个规则，无法重复绑定。
- 目前 CLB 负载均衡触发器仅支持同地域云函数绑定。例如，广州地区创建的云函数，仅支持被广州地区创建的 CLB 负载均衡中规则所绑定和触发。

请求与响应

CLB 负载均衡发送到云函数的请求处理方式，和云函数响应给 CLB 负载均衡的返回值处理方式，称为请求方法和响应方法。请求方法和响应方法都为 CLB 触发器自动处理。CLB 触发器触发云函数时，必须按照响应方法返回数据结构。

> **注意：**
> `X-Vip`、`X-Vport`、`X-Uri`、`X-Method`、`X-Real-Port` 字段必须先在 CLB 控制台进行自定义配置后才可以进行传递，自定义配置可参考 CLB 产品文档。

CLB 触发器的集成请求事件消息结构

在 CLB 负载均衡触发器接收到请求时，会将类似以下 JSON 格式的事件数据发送给绑定的云函数。

> **注意：**
> 在 CLB 触发场景下，所有的请求和响应由于需要以 JSON 方式传递，对于一些图片、文件等数据，直接放入 JSON 会导致不可见字符丢失，需要进行 Base64 编码，此处规定如下：
> - 如果 Content-type 为 text/*、application/json、application/javascript、application/xml，CLB 不会对 body 内容进行转码。
> - 其他类型一律进行 Base64 转码再转发。

```
{
  "headers": {
    "Content-type": "application/json",
    "Host": "test.clb-scf.com",
    "User-Agent": "Chrome",

    "X-Stgw-Time": "1591692977.774",
    "X-Client-Proto": "http",
    "X-Forwarded-Proto": "http",
    "X-Client-Proto-Ver": "HTTP/1.1",
    "X-Real-IP": "9.43.175.219",
    "X-Forwarded-For": "9.43.175.xx",

    "X-Vip": "121.23.21.xx",
    "X-Vport": "xx",
    "X-Uri": "/scf_location",
    "X-Method": "POST",
    "X-Real-Port": "44347",
  },
  "payload": "{\"key1\": \"123\", \"key2\": \"abc\"}"
}
```

数据结构内容详细说明如下：

结构名	内容

X-Stgw-Time	请求发起的时间戳
X-Forwarded-Proto	请求的 scheme 结构体
X-Client-Proto-Ver	协议类型
X-Real-IP	客户端 IP 地址
X-Forwarded-For	经过的代理 IP 地址
X-Real-Port	记录在 CLB 中配置过的 Path 参数以及实际取值。（可选，CLB 个性化配置）
X-Vip	CLB 负载均衡的 VIP 地址（可选，CLB 个性化配置）
X-Vport	CLB 负载均衡的 Vport（可选，CLB 个性化配置）
X-Url	请求 CLB 负载均衡的 PATH（可选，CLB 个性化配置）
X-Method	请求 CLB 负载均衡的 method（可选，CLB 个性化配置）

注意：

- 在 CLB 负载均衡迭代过程中，内容可能会增加更多。目前会保证数据结构内容仅增加，不删除，不对已有结构进行破坏。
- 实际请求时的参数数据可能会在多个位置出现，可根据业务需求选择使用。

集成响应

集成响应，是指 CLB 负载均衡会将云函数的返回内容进行解析，并根据解析内容构造 HTTP 响应。通过使用集成响应，可以通过代码自主控制响应的状态码、headers、body 内容，可以实现自定义格式的内容响应，例如响应 XML、HTML、JSON 甚至 JS 内容。在使用集成响应时，需要按照 CLB 负载均衡触发器的集成响应返回数据结构，才可以被成功解析，否则会出现 `Invalid scf response. expected scf response valid JSON.` 错误信息。

CLB 触发器的集成响应返回数据结构

在 CLB 负载均衡设置为集成响应时，需要返回类似如下内容的数据结构。

```
{
    "isBase64Encoded": false,
    "statusCode": 200,
    "headers": {"Content-Type":"text/html"},
    "body": "<html><body><h1>Heading</h1><p>Paragraph.</p></body></html>"
}
```

数据结构内容详细说明如下：

结构名	内容
isBase64Encoded	指明 body 内的内容是否为 Base64 编码后的二进制内容，取值需要为 JSON 格式的 true 或 false。
statusCode	HTTP 返回的状态码，取值需要为 Integer 值。
headers	HTTP 返回的头部内容，取值需要为多个 key-value 对象，或 `key:[value,value]` 对象。其中 key、value 均为字符串。
body	HTTP 返回的 body 内容。

在需要返回 key 相同的多个 headers 时，可以使用字符串数组的方式描述不同 value，例如：

```
{
    "isBase64Encoded": false,
    "statusCode": 200,
    "headers": {"Content-Type":"text/html","Key":["value1","value2","value3"]},
    "body": "<html><body><h1>Heading</h1><p>Paragraph.</p></body></html>"
}
```

云 API 触发器

最近更新时间：2019-12-04 17:58:17

概述

用户可以编写 SCF 函数来处理自身业务逻辑，并通过 SCF 暴露的管理接口来触发云函数。管理接口在腾讯云中统一称为云 API。通过使用 SCF 云 API 中的 Invoke 接口，用户可以按需触发调用云函数。
详细的云 API 接口调用方法可见 运行函数 API 文档。云 API 触发器具有以下特点：

- **调用模式可选**：可根据 InvocationType 参数，自行定义同步或异步触发方法。
- **自定义事件**：可根据 ClientContext 参数，自行定义触发云函数的事件或数据内容，内容需要以 JSON 格式编码。

云 API 调用

通过云 API 触发云函数，需要：

1. 按 API 接口进行鉴权；
2. 填写公共参数；
3. 根据 返回结果 解析返回结果。

另外，为了避免自行构造请求内容及解析内容，用户也可以直接使用云 API SDK 触发云函数。云 API SDK 提供了 Python，PHP，Java，GO，.NET，Node.js 语言的实现，各语言 SDK 的具体使用方式可见 腾讯云 SDK 中心 。

事件总线触发器
事件总线触发器说明

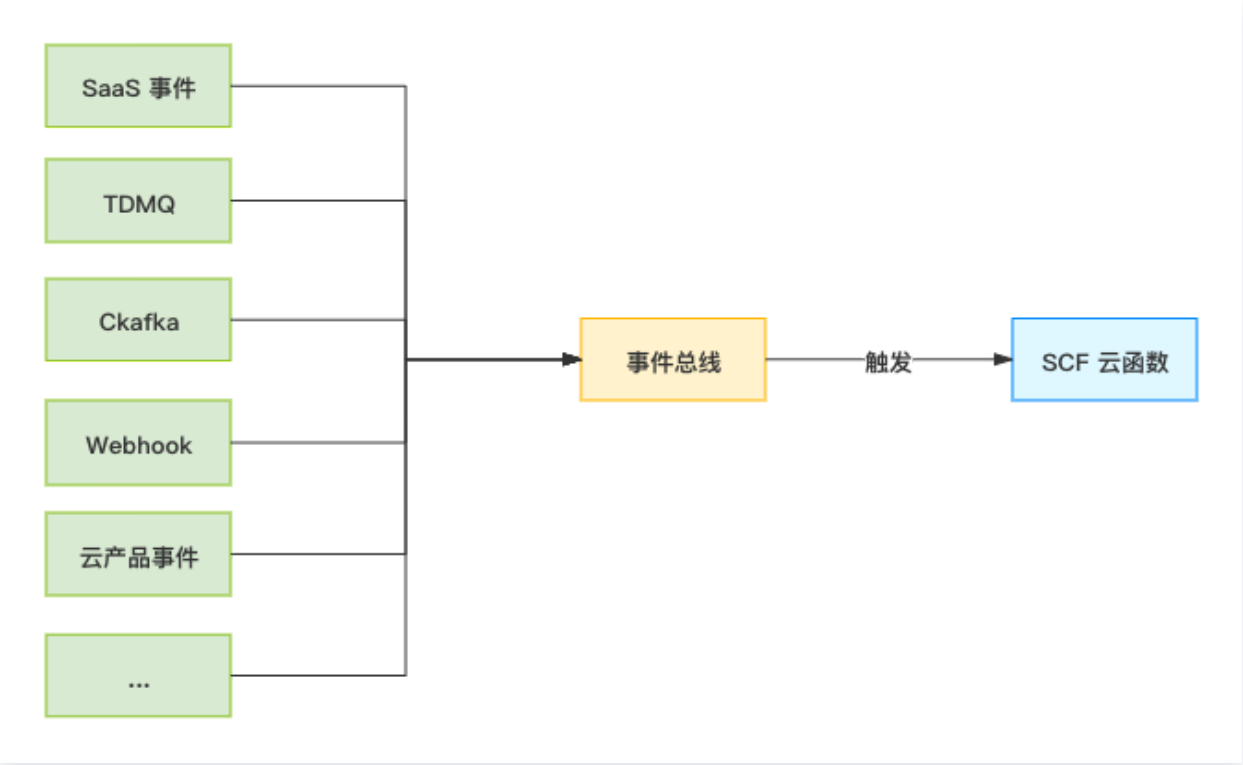
最近更新时间：2021-07-27 17:38:15

产品介绍

腾讯云事件总线（EventBridge）是一款安全、稳定、高效的无服务器事件管理平台。事件中心的事件总线可以接收来自您自己的应用程序、软件即服务（SaaS）和腾讯云服务的实时事件及相关数据流，并将事件、数据路由到，触发云函数完成事件的处理。

触发方式

事件总线提供了云上事件源的通用接入方式，云服务事件或产品事件通过事件总线进行匹配后，可以直接发送到指定目标函数，完成函数的触发和事件的消费。



事件总线触发器具有以下特点：

- **Push 模型**
  事件总线会将收到的产品事件主动推送至云函数，完成函数的触发与调用。
- **异步调用**
  事件总线触发器使用异步触发类型来调用函数。有关调用类型的更多信息，请参见 调用类型 。

> **说明**
> - 对于运行错误（含用户代码错误和 Runtime 错误），事件总线触发器默认重试两次。
> - 对于系统错误，事件总线触发器会采用指数退避的方式持续重试六小时。

应用场景

- **通用触发场景**
  事件总线拓展云函数的触发场景，可以广泛用于事件驱动架构中，以使用事件触发服务在应用之间进行通信。
- **云原生驱动场景**
  事件总线 + 云函数提供了事件完全的生命周期与完全云原生的事件驱动方案，帮助您快速架构云原生事件驱动（EDA），联动更多应用场景。
- **状态变化通知**
  事件总线可以作为中心接收所有应用的状态变化，然后将这些应用状态变化分别路由到对应的云函数，完成事件处理、存储、告警等更多能力。

支持事件源

- 消息队列 TDMQ。详情可参见 TDMQ 触发 。
- 消息队列 Ckafka。详情可见 Ckafka 触发 。
- SaaS 事件投递（企业集成服务 EIS 提供支持）。详情可见 SaaS 触发 。

事件消费

对于投递目标为云函数时，事件总线支持批量投递，您可以根据实际业务需求，选择投递方式。
批量投递参数说明：

- **最长等待时间：** 单次触发的最长等待时间，最长等待时间范围（1-60s），默认为 1。
- **最大批量消息数：** 在拉取并批量投递给当前云函数时的最大消息数，目前支持最高配置为10000。结合消息大小、写入速度等因素影响，每次触发云函数并投递的消息数量不一定能达到最大值，而是处在1 - 最大消息数之间的一个变动值。

> **注意**
> 开启批量投递功能后，事件将以数组形式投递，请注意事件消费端的格式适配。

未开启批量投递事件格式：

```
{
    "specversion": "1.0.2",
    "id": "13a3f42d-7258-4ada-da6d-023a33******",
    "type": "connector:apigw",
    "source": "apigw.cloud.tencent",
    "subjuect": "qcs::apigw:ap-guangzhou:uid1250000000/appidxxx:Serverid/Appid",
    "time": "1615430559146",
    "region": "ap-guangzhou",
    "datacontenttype": "application/json;charset=utf-8",
    "data":{
            $data_value
        }
}
```

开启批量投递，数组模式：

```
{
   "EventList":[
      {
        "specversion": "1.0.2",
        "id": "13a3f42d-7258-4ada-da6d-023a33******",
        "type": "connector:apigw",
        "source": "apigw.cloud.tencent",
        "subjuect": "qcs::apigw:ap-
guangzhou:uid1250000000/appidxxx:Serverid/Appid",
        "time": "1615430559146",
        "region": "ap-guangzhou",
        "datacontenttype": "application/json;charset=utf-8",
        "data":{
              $data_value
         }
      },
      {
        "specversion": "1.0.2",
        "id": "13a3f42d-7258-4ada-da6d-023a33******",
        "type": "connector:apigw",
        "source": "apigw.cloud.tencent",
        "subjuect": "qcs::apigw:ap-
guangzhou:uid1250000000/appidxxx:Serverid/Appid",
        "time": "1615430559146",
        "region": "ap-guangzhou",
        "datacontenttype": "application/json;charset=utf-8",
        "data":{
              $data_value
         }
      }
```

```
    ]
}
```

TDMQ 触发

最近更新时间：2022-12-22 17:26:38

通过 EventBridge 事件总线，用户可以编写云函数来处理 TDMQ 消息队列中收取到的消息。云函数后台模块可以作为消费者消费 TDMQ 中的消息，并将消息传递给云函数，本篇文档将为您指导，云函数如何通过 EventBridge 事件总线触发器，接收并消费来自 TDMQ 的产品事件。

创建步骤

步骤1：创建函数

登录 Serverless 控制台，在新建函数页面，完成您的函数代码上传与部署。详情可参见 使用控制台创建一个事件函数 。

> **注意**
> 目前 TDMQ 只支持北京、上海、广州地域。

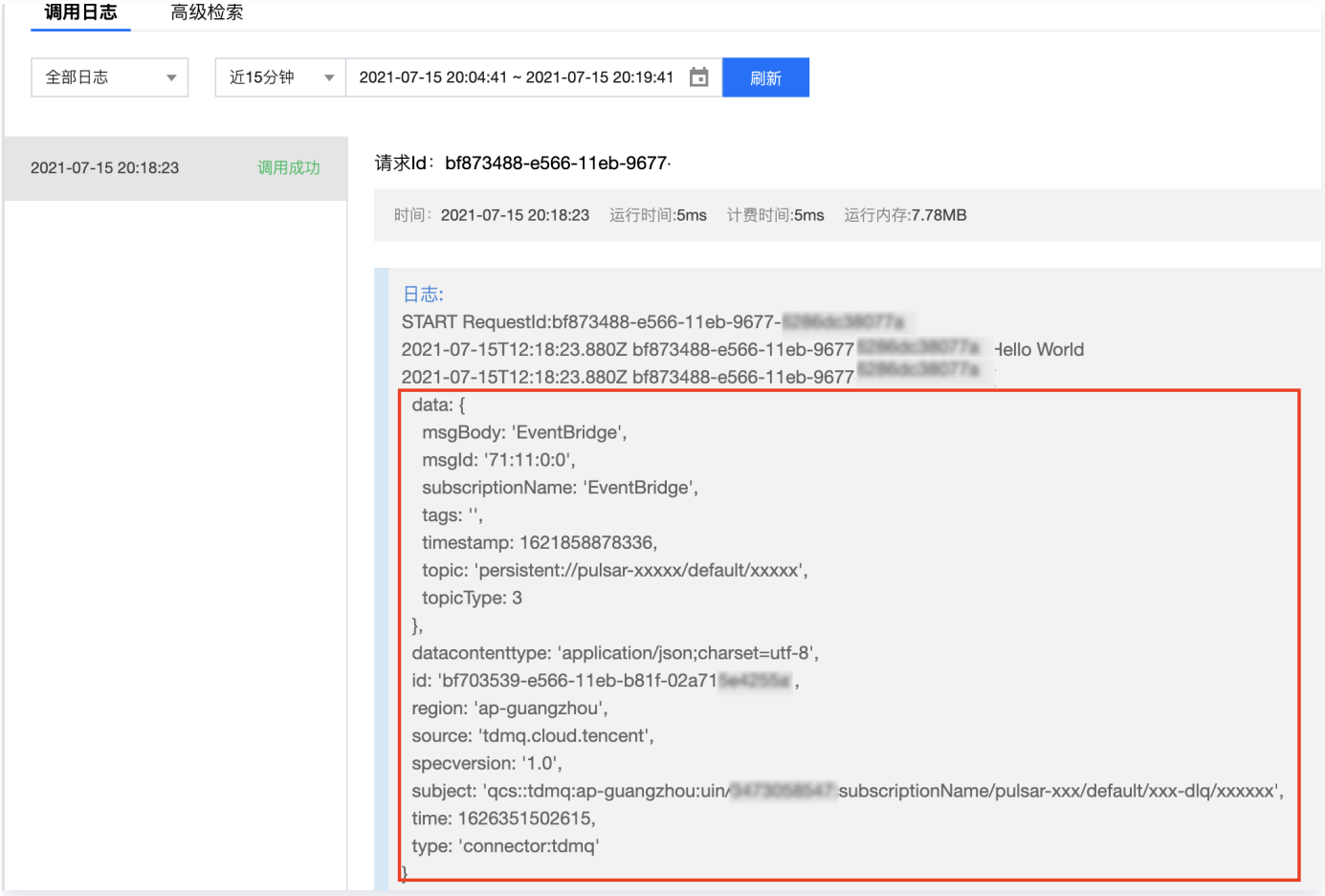
步骤2：配置触发器

在配置触发器步骤，选择 **TDMQ Pulsar消息队列触发**后，按照指引，依次选择您的 TDMQ 集群、主题等信息，指定触发事件源，消费位置：

步骤3：管理触发器

创建完成后，在“触发器管理”页面可以看到创建的触发器信息，点击进入事件总线控制台，即可完成事件集、事件源等信息管理，详情请参考 事件总线产品文档 。

给指定 TDMQ 消息队列发送信息，即可看到函数被正常调用：



事件结构

```
{
    {
     "specversion": "0",
     "id": "13a3f42d-7258-4ada-da6d-023a333b4662",
     "type": "connector:tdmq",
     "source": "tdmq.cloud.tencent",
     "subjuect":
"qcs::tdmq:$region:$account:topicName/$topicSets.clusterId/$topicSets.environmen
tId/$topicSets.topicName/$topicSets.subscriptionName",
     "time": "1615430559146",
     "region": "ap-guangzhou",
     "datacontenttype": "application/json;charset=utf-8",
     "data": {
                    "topic":  "persistent://appid/namespace/topic-1",
            "tags": "testtopic",
                    "TopicType": 0,
                    "subscriptionName": "xxxxxx",
                    "toTimestamp": "1603352765001",
            "partitions": "0",
```

```
                    "msgId": "123345346",
                    "msgBody": "Hello from TDMQ!"
    }
}
```

参数说明如下：

参数	描述
topic	Topic 完整路径 `persistent://appid/namespace/topic-1` 。
subscriptionName	订阅名称。
timestamp	时间戳，精确到毫秒。
tags	TDMQ 标签。
msgId	TDMQ 消息 ID。
msgBody	TDMQ 消息体。
topictype	topic 类型描述： 0：普通消息。 1：全局顺序消息。 2：局部顺序消息。 3：重试队列。 4：死信队列。

触发器配置描述

最近更新时间：2025-06-13 15:09:02

当您调用触发器接口 设置函数触发方式（CreateTrigger）时，对应的 `TriggerDesc` 参数为触发器描述，您可参考本文进行使用。

定时触发器

该参数直接填写 Cron 表达式，相关内容请参考 Cron 表达式 。

TriggerDesc 示例

每五分钟触发一次

```
0 */5 * * * * *
```

API 网关触发器

名称	类型	必选	描述
api	ApigwApi	否	创建 API 网关的 API 配置。
service	ApigwService	否	创建 API 网关的 Service 配置。
release	ApigwRelease	否	创建 API 网关后，发布的环境。

ApigwApi

名称	类型	必选	描述
authRequired	String	否	是否需要鉴权，可选 TRUE 或者 FALSE，默认为 FALSE。
requestConfig	ApigwApiRequestConfig	否	请求后端 API 的配置。
isIntegratedResponse	String	否	是否使用集成响应，可选 TRUE 或者 FALSE，默认为 FALSE。
IsBase64Encoded	String	否	是否打开 Base64 编码，可选 TRUE 或者 FALSE，默认为 FALSE。

ApigwApiRequestConfig

名称	类型	必选	描述
method	String	否	请求后端 API 的 method 配置，必须是 `ANY` 、 `GET` 、 `HEAD`

			、`POST`、`PUT`、`DELETE` 中的一种。

ApigwService

名称	类型	必选	描述
serviceId	String	否	Apigw Service ID（不传入则新建一个 Service）。

ApigwRelease

名称	类型	必选	描述
environmentName	String	是	发布的环境，填写 `release`、`test` 或 `prepub`，不填写默认为 `release`。

TriggerDesc 示例

```
{
  "api":{
    "authRequired":"FALSE",
    "requestConfig":{
      "method":"ANY"
    },
    "isIntegratedResponse":"FALSE"
  },
  "service":{
    "serviceName":"SCF_API_SERVICE"
  },
  "release":{
    "environmentName":"release"
  }
}
```

Ckafka 触发器

名称	类型	必选	描述
maxMsgNum	String	是	5秒内每汇聚 maxMsgNum 条 Ckafka 消息，则触发一次函数调用。
offset	String	是	offset 为开始消费 Ckafka 消息的位置，目前支持 `latest`、`earliest` 和 `毫秒级时间戳` 三种方式。
retry	String	是	当函数报错时最大重试次数。

timeOut	String	是	单次触发的最长等待时间，可选时间范围1~60。
consumerGroupName	String	否	指定消费组名称，可实现使用已有CKafka消费组 或 自定义消费组名称的效果，为空时将按照默认命名规则创建消费组。
easConfig	String	否	开启加密访问时，需传入用户名、密码。

TriggerDesc 示例

```
{
  "TriggerDesc": {
    "maxMsgNum": 100,
    "offset": "latest",
    "retry": 10000,
    "timeOut": 60,
    "consumerGroupName": "scf_ckafka-w4pn52ndtopic-bf9fom5klam-phiy2wrg1",
    "easConfig": {
      "userName": "test3423",
      "password": "cccc34567"
    }
  }
}
```

```
{
  "TriggerDesc": {
    "maxMsgNum": 100,
    "offset": "latest",
    "retry": 10000,
    "timeOut": 1,
    "consumerGroupName":null,
    "easConfig": {
      "userName": null,
      "password": null
    }
  }
}
```

API 请求说明

使用 API 请求创建 Ckafka 触发器时，TriggerName 字段需定义为要转储的 Ckafka instanceId，topicName 信息。

`[instanceId]-[topicName]`，请求示例如下：

```
TriggerName: "ckafka-8tfxzia3-test"
```

COS 触发器

名称	类型	必选	描述
event	String	是	COS 的事件类型
filter	CosFilter	是	COS 文件名的过滤规则。

CosFilter

名称	类型	必选	描述
Prefix	String	否	过滤文件的前缀规则。
Suffix	String	否	过滤文件的后缀规则，必须以 `.` 开头。

COS 事件冲突规则

- 核心思想：一个事件最多触发一次函数调用。如果一个事件有其他产品绑定，该事件也不可再绑定至函数。
- 最多设置一个前缀过滤及一个后缀过滤。
- 若已设置 `cos:ObjectCreated:*` 事件，且没有设置前后缀，则后续再绑定任何以 `cos:ObjectCreated` 作为开头的事件都会失败。
- 前缀及后缀均匹配才有效，且当前缀和后缀都冲突的时候，后面的绑定会失败。

TriggerDesc 示例

```
{"event":"cos:ObjectCreated:*","filter":{"Prefix":"","Suffix":""}}
```

> **注意：**
> 在 `TriggerDesc` 中作为触发器描述时，JSON 字符串须是连续且中间不能有空格的字符串。

API 请求说明

使用 API 请求创建 COS 触发器，TriggerName 字段需定义为要转储的 COS 存储桶 XML API 访问域名，示例如下：

```
TriggerName: "xxx.cos.ap-guangzhou.myqcloud.com"
```

> **注意：**
> 访问域名请在对象存储控制台**存储桶列表 > 基础配置 > 基本信息**中查看。

CMQ 触发器

名称	类型	必选	描述
filterType	String	否	消息过滤类型，`1` 为标签类型，`2` 为路由匹配类型。
filterKey	String	否	当 `filterType` 为 `1` 时表示消息过滤标签，当 `filterType` 为 `2` 时表示 Binding Key。

TriggerDesc 示例

```
{"filterType":1,"filterKey":["test"]}
```

```
{"filterType":2,"filterKey":["#test"]}
```

API 请求说明

使用 API 请求创建 CMQ 触发器，TriggerName 字段需定义为 CMQ Topic，示例如下：

```
TriggerName: "Tabortest"
```