API 网关

开发指南

腾讯云

文档目录

开发指南
Serverless 应用中心
概述

最近更新时间：2025-04-03 16:08:02

简介

Serverless 应用中心是业界非常受欢迎的无服务器应用框架，开发者无需关心底层资源即可部署完整可用的 Serverless 应用架构。Serverless 应用中心具有资源编排、自动伸缩、事件驱动等能力，覆盖编码、调试、测试、部署等全生命周期，帮助开发者通过联动云资源，迅速构建 Serverless 应用。

Serverless Components

Serverless Components 是支持多个云资源编排和组织的场景化解决方案，主要基于用户的具体场景。例如，Express 框架支持及网站部署等。Serverless Components 可以有效简化云资源的配置和管理，将 API 网关、COS 和 CAM 等产品联动起来，用户可更关注场景和业务。

优势特性

- **简便易用**
  Serverless Components 更多的围绕客户场景进行构建，如网站、博客系统、支付服务、图像处理场景等。通过抽象底层的基础设施配置信息，开发者可以通过十分简单的配置实现场景。
- **可复用性**
  Serverless Components 可以通过非常简单的 `serverless.yml` 创建和部署，但同时也支持用十分简单的语法对 JavaScript 库 `serverless.js` 进行扩展编写和复用。
- **秒级部署**
  大多数 Serverless Components 比传统的配置工具部署快20倍左右，Components 可以通过快速的部署和远端验证，有效减少本地模拟和调试的环节。

API 网关组件

API 网关组件是 Serverless 应用中心提供的基础组件之一。该组件通过使用 Serverless Components，可以快速，方便的创建，配置和管理腾讯云的 API 网关产品。

实践教程

您可通过以下实践，开始使用 Serverless 应用中心 CLI：

项目	描述

部署静态网站托管	通过 Serverless Website 组件快速托管一个静态网站。
部署 Express.js 应用	通过 Serverless SCF 组件快速构建一个 Express.js 项目。
部署 Nuxt.js 应用	通过 Serverless 应用中心 Nuxt.js 组件，快速部署一个基于 Nuxt.js 的 SSR 项目。

安装 Serverless 应用中心

最近更新时间：2024-05-17 11:30:31

该任务指导您通过 二进制安装 或 NPM 安装 的方式，快速安装 Serverless 应用中心。

安装方式

方式一：二进制安装

如果您的本地环境没有安装 Node.js，您可以直接使用二进制的方式进行安装：

> **注意**
> 建议您使用 Node.js10.0 及以上版本，否则 Component V2 部署有可能报错。

MacOS/Linux 系统

打开命令行，输入以下命令：

```
$ curl -o- -L https://slss.io/install | bash
```

如果之前您已经安装过二进制版本，可以通过下列命令进行升级：

```
$ serverless upgrade
```

Windows 系统

Windows 系统支持通过 chocolatey 进行安装。打开命令行，输入以下命令：

```
$ choco install serverless
```

如果之前您已经安装过二进制版本，可以通过下列命令进行升级：

```
$ choco upgrade serverless
```

方式二：NPM 安装

> **注意**
> 建议您使用 Node.js10.0 及以上版本，否则 Component V2 部署有可能报错。

在命令行中运行如下命令：

```
$ npm install -g serverless
```

> **说明**
> 如 MacOS 提示无权限，则需要运行 `sudo npm install -g serverless` 进行安装。

如果之前您已经安装过 Serverless 应用中心，可以通过下列命令升级到最新版：

```
$ npm update -g serverless
```

如果您的环境中没有安装 Node.js，可以参见 Node.js 安装指南 根据您的系统环境进行安装。

查看版本信息

安装完毕后，通过运行 `serverless -v` 命令，查看 Serverless 应用中心 的版本信息：

```
$ serverless -v
```

创建及部署 API 网关服务

最近更新时间：2024-05-17 11:30:31

操作场景

该任务指导您通过 Serverless 应用中心提供的 API 网关组件快速创建与部署 API 网关服务/接口。

前提条件

已完成 安装 Serverless Cloud Framework 。

操作步骤

配置 API 网关服务

本地创建 `serverless.yml` 文件：

```
touch serverless.yml
```

在 `serverless.yml` 中进行如下配置：

```
# serverless.yml

component: apigateway # （必填）组件名称，此处为 apigateway
name: apigwDemo # （必填）实例名称
org: orgDemo # （可选）用于记录组织信息，默认值为您的腾讯云账户 appid
app: appDemo # （可选）该 next.js 应用名称
stage: dev # （可选）用于区分环境信息，默认值是 dev

inputs:
  region: ap-guangzhou
  protocols:
    - http
    - https
  serviceName: serverless
  environment: release
  endpoints:
    - path: /
      protocol: HTTP
      method: GET
      apiName: index
      function:
```

```
        functionName: myFunction
```

查看详细配置文档>>

部署 API 网关服务

参考 Serverless 应用中心 > 进阶指南 > 应用管理 中部署应用的流程，在一个 SCF 项目的目录下，执行以下命令进行扫码授权部署：

```
scf deploy
```

> **说明**
> 微信扫码授权部署有过期时间，如果想要持久授权，请参见 账号配置 。

移除已部署的服务

执行以下命令移除部署的服务：

```
scf remove
```

账号配置（可选）

当前默认支持 CLI 扫描二维码登录，如您希望配置持久的环境变量/密钥信息，也可以本地创建 `.env` 文件：

```
touch .env # 腾讯云的配置信息
```

在 `.env` 文件中配置腾讯云的 SecretId 和 SecretKey 信息并保存：

```
# .env
TENCENT_SECRET_ID=********
TENCENT_SECRET_KEY=********
```

> **说明**
> - 如果没有腾讯云账号，请先 注册新账号 。
> - 如果已有腾讯云账号，可以在 API 密钥管理 中获取 SecretId 和 SecretKey。

导入 API
定义 API

最近更新时间：2020-08-11 17:50:12

通过 OpenAPI 规范能够定义标准的 RESTful 风格的 API。一般来说，OpenAPI 文档有三个必须的对象：

- openapi：OpenAPI 规范的版本号。
- info：API 的元数据。
- paths：API 的请求路径与操作。

本文将分对象介绍 OpenAPI 规范与 API 网关的映射关系。

openapi

API 网关支持 3.0.0 版本的 OpenAPI 规范。

info

OpenAPI 和 API 网关 info 对象的映射关系如下表：

OpenAPI 对象	数据类型	OpenAPI 对象说明	API 网关对象
info.title	String	API 所属服务的名称	未使用
info.description	String	API 所属服务的描述	未使用
info.version	String	版本号	未使用

paths

OpenAPI 和 API 网关 paths 对象的映射关系如下表：

OpenAPI 对象	数据类型	OpenAPI 对象说明	API 网关对象
paths.path	Object	API 请求路径	API 前端请求路径
operation.operationId	String	API 名称	API 名称
operation.description	String	API 描述	未使用
operation.parameters	Object	API 请求参数	API 请求参数

operation.responses	String	API 响应	未使用

说明

- OpenAPI 3.0.0 规范的对象定义请参见 OpenAPI Specification 。
- 导入 API 功能的操作流程请参见 导入 API 。
- 导入 API 的完整示例请参见 导入 API 示例 。

导入 API 示例

最近更新时间：2024-11-05 14:59:02

本文介绍不同 API 后端导入 API 示例。YAML 格式示例如下。

后端为 Mock 类型

```
openapi: 3.0.0
info:
  title: test
  version: 1.0.1
paths:
  /:
    get:
      operationId: test
      responses:
        '200':
          description: The list of possible responsesas they are
returned from executing
            this operation.
      x-apigw-api-business-type: NORMAL
      x-apigw-api-type: NORMAL
      x-apigw-backend:
        MockReturnHttpHeaders: []
        MockReturnHttpStatusCode: 200
        ServiceMockReturnMessage: success
        ServiceType: MOCK
      x-apigw-cors: false
      x-apigw-protocol: HTTP
      x-apigw-service-timeout: 15
```

后端为 Proxy 类型

```
openapi: 3.0.0
info:
  title: testa
  version: 1.0.1
paths:
  /proxy:
    get:
      operationId: test
```

```
      responses:
        '200':
          description: The list of possible responsesas they are
returned from executing
            this operation.
      x-apigw-api-business-type: NORMAL
      x-apigw-api-type: NORMAL
      x-apigw-backend:
        ServiceConfig:
          Method: GET
          Path: /
          Url: http://cloud.tencent.com
        ServiceType: HTTP
      x-apigw-cors: false
      x-apigw-protocol: HTTP
      x-apigw-service-timeout: 15
```

后端为 VPC 内网服务类型

```
openapi: 3.0.0
info:
  title: test
  version: 1.0.1
paths:
  /:
    get:
      operationId: test
      responses:
        '200':
          description: The list of possible responsesas they are
returned from executing
            this operation.
      x-apigw-api-business-type: NORMAL
      x-apigw-api-type: NORMAL
      x-apigw-backend:
        ServiceConfig:
          Method: GET
          Path: /
          Product: clb
          UniqVpcId: vpc-xxxxxx
          Url: http://172.x.x.x:8xxx
        ServiceType: HTTP
      x-apigw-const-paramters:
```

```
      - DefaultValue: xxx
        Desc: "xxxx后端Host"
        Name: Host
        Position: HEADER
      x-apigw-cors: false
      x-apigw-protocol: HTTP
      x-apigw-service-timeout: 15
```

后端为 SCF Event 类型

```
openapi: 3.0.0
info:
  title: testa
  version: 1.0.1
paths:
  /scf:
    get:
      operationId: test
      responses:
        '200':
          description: The list of possible responsesas they are
returned from executing
            this operation.
      x-apigw-api-business-type: NORMAL
      x-apigw-api-type: NORMAL
      x-apigw-backend:
        IsBase64Encoded: false
        ServiceScfFunctionName: APIGWCustomRespDemo-xxxxx
        ServiceScfFunctionNamespace: default
        ServiceScfFunctionQualifier: $DEFAULT
        ServiceScfFunctionType: EVENT
        ServiceScfIsIntegratedResponse: false
        ServiceType: SCF
      x-apigw-cors: false
      x-apigw-protocol: HTTP
      x-apigw-service-timeout: 15
```

后端为 SCF Web 函数类型

```
openapi: 3.0.0
info:
  title: testa
```

```
  version: 1.0.1
paths:
  /scf:
    get:
      operationId: test
      responses:
        '200':
          description: The list of possible responsesas they are
returned from executing
            this operation.
      x-apigw-api-business-type: NORMAL
      x-apigw-api-type: NORMAL
      x-apigw-backend:
        IsBase64Encoded: false
        ServiceScfFunctionName: flask_demo-xxxxxxxxx
        ServiceScfFunctionNamespace: default
        ServiceScfFunctionQualifier: $DEFAULT
        ServiceScfFunctionType: HTTP
        ServiceScfIsIntegratedResponse: false
        ServiceType: SCF
      x-apigw-cors: false
      x-apigw-protocol: HTTP
      x-apigw-service-timeout: 15
```

后端为 COS 类型

```
openapi: 3.0.0
info:
  title: test
  version: 1.0.1
paths:
  /cos:
    get:
      operationId: test
      responses:
        '200':
          description: The list of possible responsesas they are
returned from executing
            this operation.
      x-apigw-api-business-type: NORMAL
      x-apigw-api-type: NORMAL
      x-apigw-backend:
        ServiceConfig:
```

```
          CosConfig:
            Action: GetObject
            Authorization: true
            BucketName: xxxxxxx
            PathMatchMode: FullPath
          Path: /
        ServiceType: COS
      x-apigw-cors: false
      x-apigw-protocol: HTTP
      x-apigw-service-timeout: 15
```

TSF 微服务 API 类型

```
openapi: 3.0.0
info:
  title: SCF_API_SERVICE
  version: 1.0.1
paths:
  /:
    get:
      operationId: test
      parameters:
      - description: ""
        in: header
        name: X-MicroService-Name
        required: true
        schema:
          type: string
      - description: ""
        in: header
        name: X-NameSpace-Code
        required: true
        schema:
          type: string
      responses:
        '200':
          description: The list of possible responsesas they are
returned from executing
            this operation.
      x-apigw-api-business-type: NORMAL
      x-apigw-api-type: TSF
      x-apigw-backend:
        MicroServices:
```

```
        - ClusterId: cluster-xxxxxx
          MicroServiceName: provider-demo
          NamespaceId: namespace-xxxxx
        ServiceConfig:
          Path: /
        ServiceTsfHealthCheckConf:
          ErrorThresholdPercentage: 50
          IsHealthCheck: true
          RequestVolumeThreshold: 20
          SleepWindowInMilliseconds: 5000
        ServiceTsfLoadBalanceConf:
          IsLoadBalance: true
          Method: RoundRobinRule
          SessionStickRequired: false
          SessionStickTimeout: 0
        ServiceType: TSF
      x-apigw-cors: false
      x-apigw-protocol: HTTP
      x-apigw-service-timeout: 15
```

JSON 格式导入 API

以 Mock 后端为例，JSON 格式参考如下。其余类型可参考 YAML 格式。另外，导入数据中如有 x-apigw-backend 则不需要 servers。

```
info:
  title: TEST
  version: 1.0.1
openapi: 3.0.0
paths:
  /mock:
    get:
      description: importMockAPI
      operationId: Mock API
      responses:
        '200':
          description: >-
            The list of possible responsesas they are returned from
executing
            this operation.
      x-apigw-api-business-type: NORMAL
      x-apigw-api-type: NORMAL
      x-apigw-backend:
```

```
      MockReturnHttpHeaders: []
      MockReturnHttpStatusCode: 200
      ServiceMockReturnMessage: Import Mock API Test
      ServiceType: MOCK
    x-apigw-cors: false
    x-apigw-protocol: HTTP
    x-apigw-service-timeout: 15
```

签名工具

最近更新时间：2023-06-26 10:11:02

操作场景

API 网关签名工具是腾讯云 API 网关为用户提供的 Web 工具，可用于生成密钥对认证 API 的请求签名。您可以通过 API 网关控制台，在工具页面上填入指定的参数，以生成请求签名。

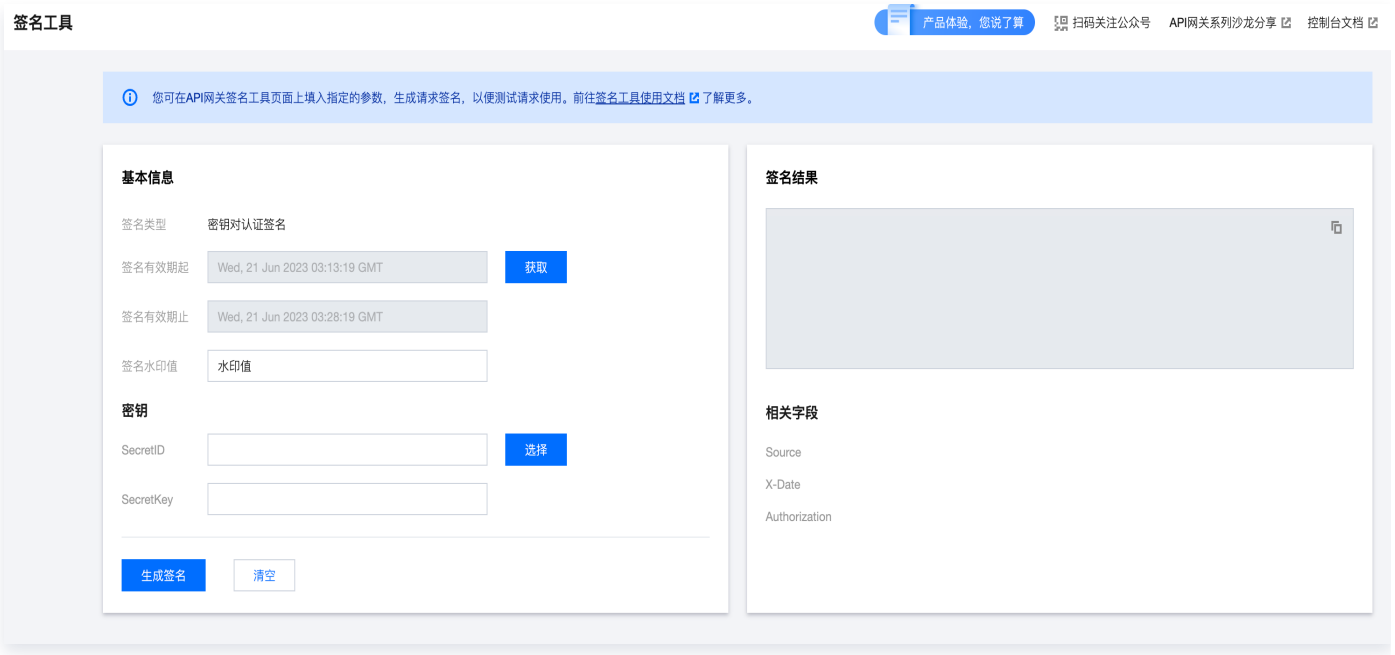
前提条件

已下载并安装 Postman 插件（ 下载地址 ）

操作步骤

步骤1：填写签名基本配置信息

1. 登录 API 网关控制台 。
2. 在左侧导航栏单击**工具** > **签名工具**，进入 API 网关签名工具页面。
3. 在**基本信息**栏中，单击**获取**，自动填充签名有效时间。
4. 输入签名水印值，可填写任意值。

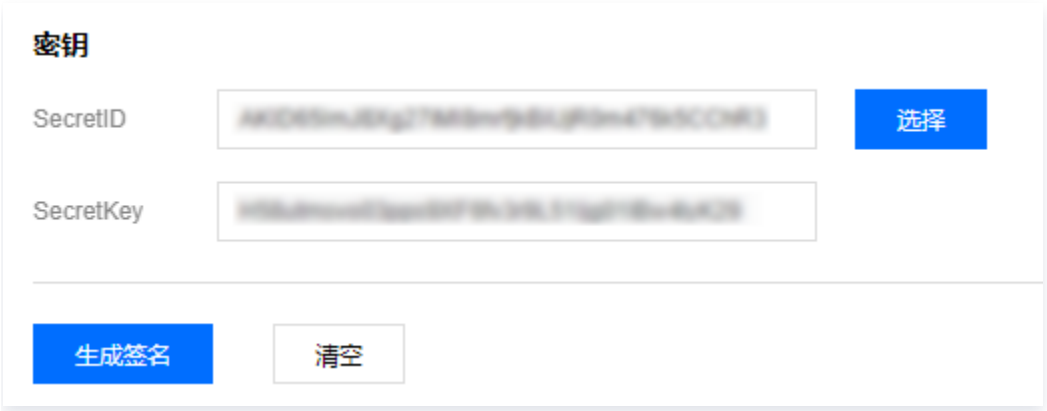


步骤2：输入 API 密钥

在密钥栏中，输入您的 API 密钥信息：

方式一：单击**选择**，选择您账户下已创建的密钥。

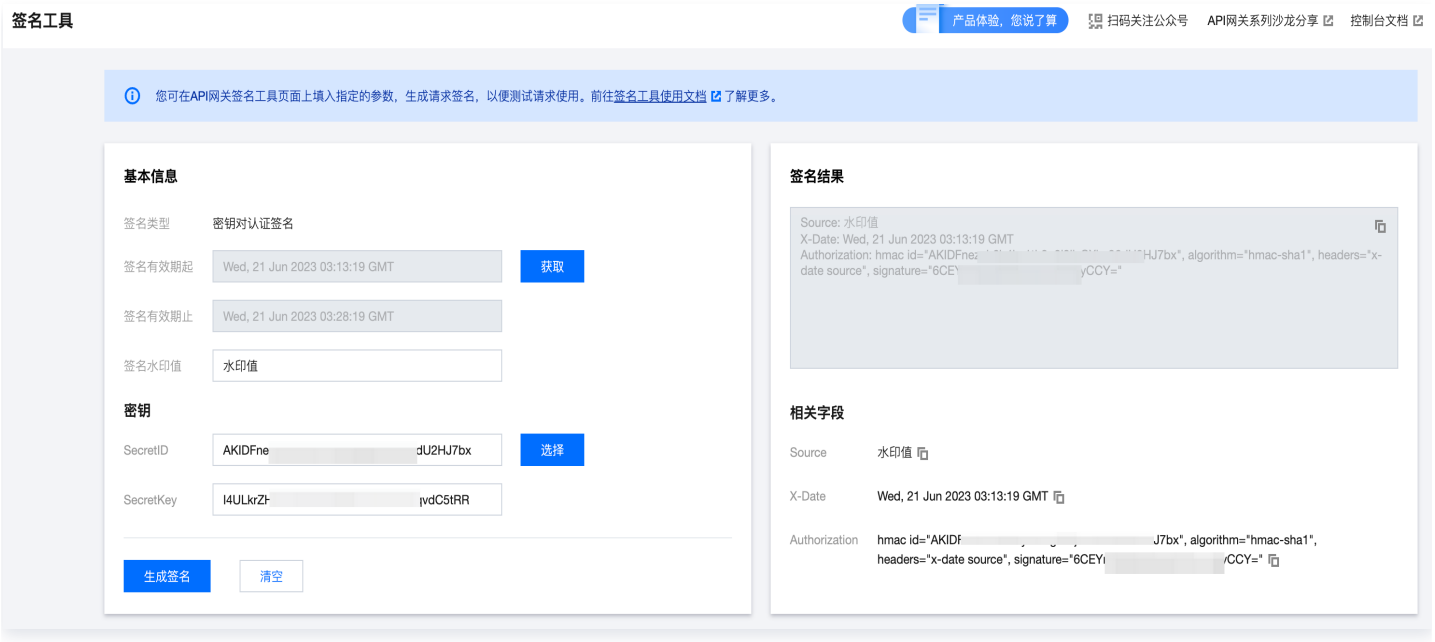
方式二：直接输入密钥。
填写时，请确保该信息的准确性。填写错误将导致您的签名被视为无效签名。



步骤3：生成签名

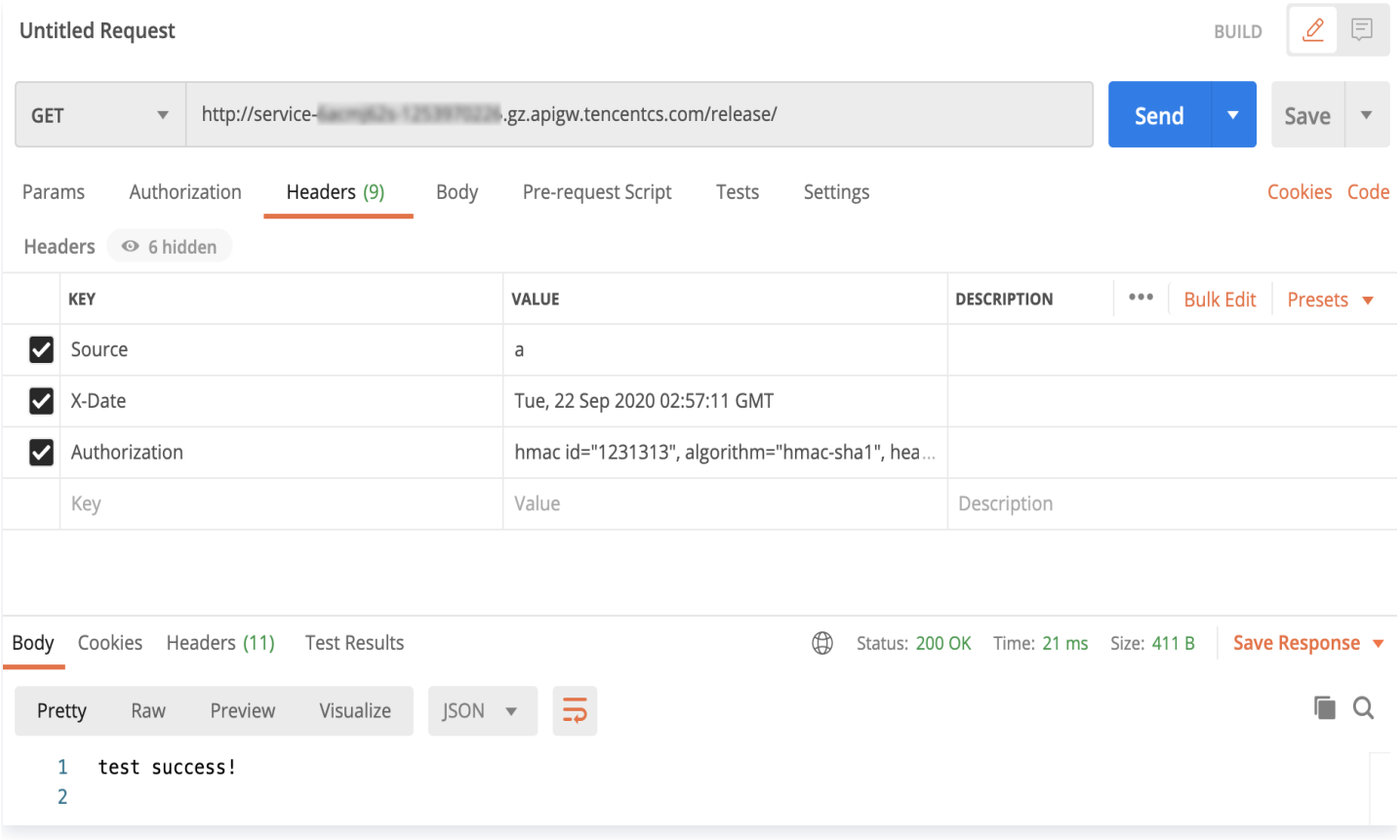
单击**生成签名**，即可在右侧展示区中查看请求签名结果。主要参数介绍如下：

- Source：签名水印值。
- X-Date：格林威治时间（GMT）格式的 HTTP 请求构造时间，X-Date 里的时间和当前时间的差值不能超过15分钟。
- Authorization：签名内容。



步骤4：在 Postman 中发起调用

打开 Postman，将 Source、X-Date、Authorization 三个参数填写在 Header 中，并填写 API 请求地址（API 网关提供的默认域名/环境/API 路径）、API 请求参数等信息，单击 **Send**，即可发起对密钥对认证 API 的调用。



注意事项

- 由于服务端会校验时间，所以您必须在签名有效期内发起调用，否则将调用失败。
- 本签名工具只校验不合法参数，不识别和提示您填写的错误参数。
- 密钥对认证的相关介绍请参见 密钥对认证 。

快速删除服务工具

最近更新时间：2024-05-17 11:30:32

操作场景

为了保护用户业务，避免因服务误删除导致的业务损失，如果服务中有 API 或未下线的服务，则此服务不能直接删除。用户必须依次进行下线服务、查询该服务的 API 列表、批量删除服务下 API、删除服务等步骤后才能删除服务，非常繁琐。
针对此问题，腾讯云 API 网关团队和腾讯云 Serverless 应用中 团队推出了 Serverless Cleaner，用于快速删除 API 网关服务。

操作步骤

1. 前往 下载地址，下载 Serverless Cleaner。
2. 使用命令行工具进入 Serverless Cleaner 所在目录。
3. 在命令行工具中输入如下命令，安装依赖文件。

```
$ npm install
```

4. 安装依赖文件完成后，进入 Serverless Cleaner 所在目录，复制 config.example.js 并重命名为 config.js。
5. 根据需要删除服务的信息，修改 config.js 的内容。

```
module.exports = {
 tencent: {
     // 资源所在地域，如果您未修改，可以在运行时输入
     region: 'ap-guangzhou',
     // 身份认证
     credentials: {
         SecretId: '',
         SecretKey: '',
     },
     apigwOptions: {
         // 不会被删除的服务的 ID，在您使用  Serverless Cleaner 之前，请将您重要的服务 ID 输入到这里，避免影响业务
         exclude: [],
         // 将被删除的服务的 ID，优先级高于 exclude。如果 exclude 与 include 中存在同一个服务 ID，该服务将被删除
         include: [],
     },
```

```
      scfOptions: {
          // 删除 SCF 资源时需要使用，仅清除 API 网关服务时无需填写
          exclude: [],
          include: [],
      },
  },
};
```

6. 在命令行工具中输入如下命令，进行服务清除。

```
$ npm run clean
```

上传文件

最近更新时间：2023-06-26 09:44:12

操作场景

该任务指导您通过 API 网关上传文件到后端服务，按后端类型不同和文件大小不同可分为四种情况，您需要根据自身业务的情况合理选择。

上传流程

- 如果是后端对接 SCF 的 API，上传 Base64 编码后小于 6MB 的文件，建议您通过客户端对文件进行 Base64 编码，将编码后的内容传递给 SCF，再由 SCF 进行 Base64 解码，以完成小文件的上传。
- 如果是后端对接 SCF 的 API，上传 Base64 编码后大于 6MB 的文件，建议您通过客户端先将文件上传至 COS，将 Object 地址传递给 SCF，再由 SCF 从 COS 拉取文件，以完成大文件的上传。
- 如果是后端对接其他服务的 API，上传小于 16MB 的文件，您只需要不填写参数，保持透传即可。
- 如果是后端对接其他服务的 API，上传大于 16MB 的文件，建议您通过客户端先将文件上传至 COS，将 Object 地址传递给后端服务，再由后端服务从 COS 拉取文件，以完成大文件的上传。

Serverless 多文件上传处理

最近更新时间：2023-06-21 17:47:04

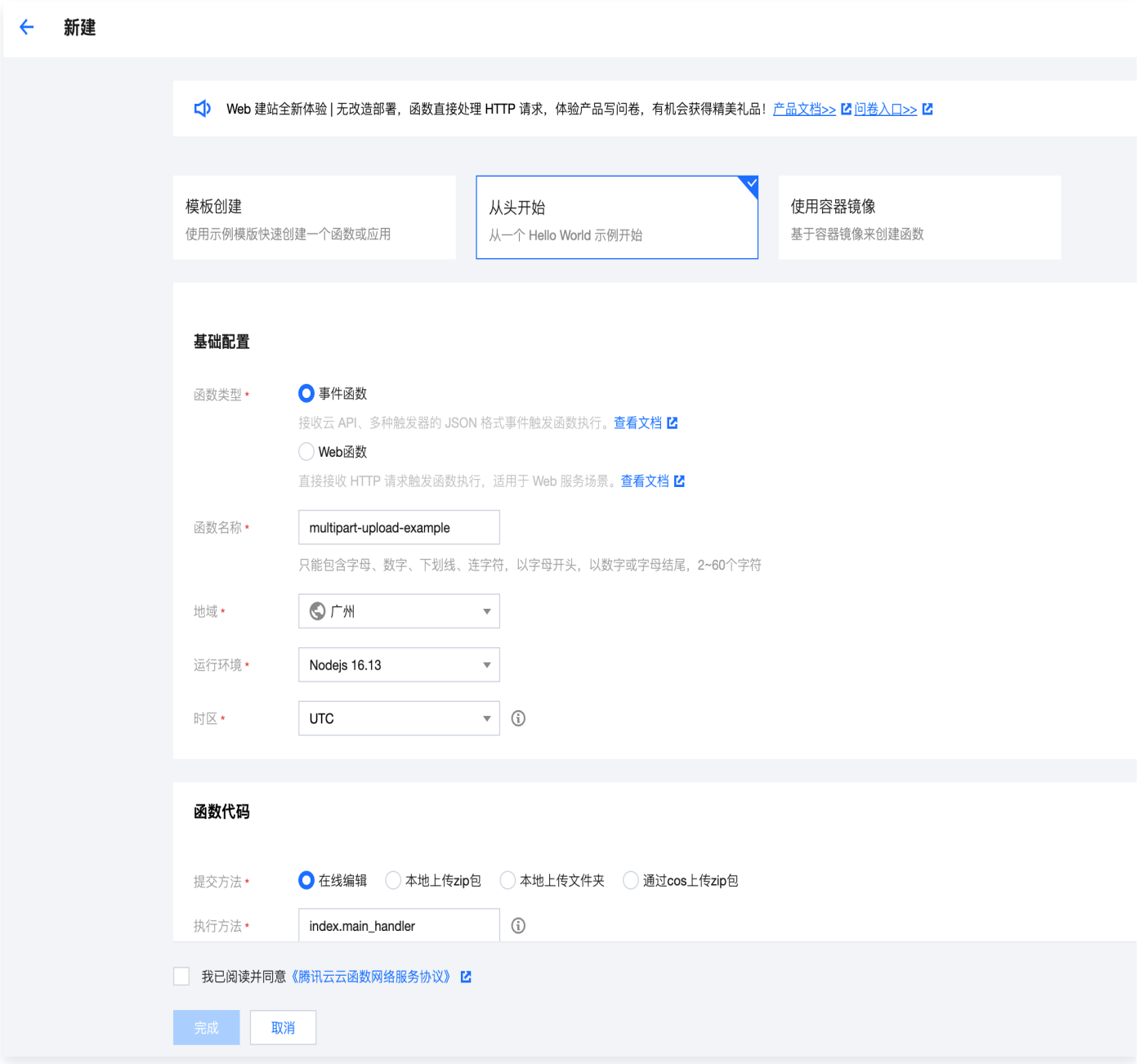
操作场景

通过腾讯云 Serverless 处理 multipart/form-data 多文件上传的 HTTP 请求，原理上需要利用 API 网关的 Base64 编码能力，将原始 HTTP 请求中的 multipart 字节流编码为字符串，以便将 HTTP Event 序列化，传入云函数 SCF 进行处理。
云函数将 API 网关传来 event 中的 body 获取并解码 Base64 后，生成的字节流则与普通 HTTP 请求中的无异，正常处理即可。在 Node.JS 中，我们可以利用 `busboy` 等库进行处理。

操作步骤

步骤1：创建云函数

1. 登录 云函数控制台 。
2. 在**函数服务**页面，单击**新建**，选择从头开始，运行环境选择 `Node.js` 。
   创建时，具体参数如下：



3. 单击**完成**，完成云函数的创建。

步骤2：编写代码并部署

1. 云函数创建完成后，可以参考以下示例代码编写处理 multipart/form-data 的具体逻辑。

```
// handler.js
"use strict";
const stream = require("stream");
const Busboy = require("busboy");
```

```
/** 处理用户上传 （POST） */
const handlePost = (event) => {
  return new Promise((resolve, reject) => {
    const busboy = new Busboy({ headers: event.headers });
    let html = "";
    /** 接受到文件 */
    busboy.on("file", (fieldname, file, filename, encoding, mimetype)
=> {
      let buf = Buffer.alloc(0);
      console.log({ fieldname });
      /** 接受到文件的数据块，拼接出完整的 buffer */
      file.on("data", function (data) {
        buf = Buffer.concat([buf, data]);
      });
      /** 文件的数据块接受完毕，生成 DOM 字符串 */
      file.on("end", function () {
        const imgBase64 = buf.toString("base64");
        html += `<img src="data:${mimetype};base64, ${imgBase64}" />`;
      });
    });
    /** multipart/form-data 接受完毕，构造并返回生成的 html */
    busboy.on("finish", function () {
      console.log({ msg: "Parse form complete!", html });
      resolve({
        statusCode: 200,
        headers: {
          "content-type": "text/html",
        },
        body: html,
      });
    });

    /**
     * busboy 需要 stream pipe 的方式来进行处理，
     * 我们将 body 解码为 buffer后，
     * 转换为 stream，最终 pipe 给 busbody
     */
    const bodyBuf = Buffer.from(event.body, "base64");
    var bufferStream = new stream.PassThrough();
    bufferStream.end(bodyBuf);
    bufferStream.pipe(busboy);
  });
};
```

```
/** 返回静态文件 */
const handleGet = (event) => {
  const html = `<html><head></head><body>
    <form method="POST" enctype="multipart/form-data">
    <input type="file" name="image-1" accept="image/*"><br />
     <input type="file" name="image-2" accept="image/*"><br />
     <input type="submit">
    </form>
    </body></html>`;
  console.log({ msg: "Get form complete!", html });
  return {
    statusCode: 200,
    headers: {
      "content-type": "text/html",
    },
    body: html,
  };
};

/** 云函数入口函数 */
exports.main_handler = async (event, context) => {
  const method = event.httpMethod;
  /** 当请求为 POST 请求时，我们处理用户的 multipart/form-data，并生成展示上传结果的页面 */
  if (method === "POST") {
    return handlePost(event);
  }
  /** 当请求为 GET 请求时，我们返回上传文件的页面 */
  if (method === "GET") {
    return handleGet(event);
  }
};
```

2. 编写代码后，您也可以为云函数安装运行时需要的依赖。例如，利用 busboy 进行 multipart/form-data 数据的解码。

> **注意**
> 依赖要安装在 src 文件夹下。

```
[root@ws-lgivut-0 src]# npm i busboy
npm WARN saveError ENOENT: no such file or directory, open '/usr/local/var/
npm notice created a lockfile as package-lock.json. You should commit this
npm WARN enoent ENOENT: no such file or directory, open '/usr/local/var/fur
npm WARN src No description
npm WARN src No repository field.
npm WARN src No README data
npm WARN src No license field.

+ busboy@0.3.1
added 3 packages from 1 contributor and audited 3 packages in 0.386s
found 0 vulnerabilities
```

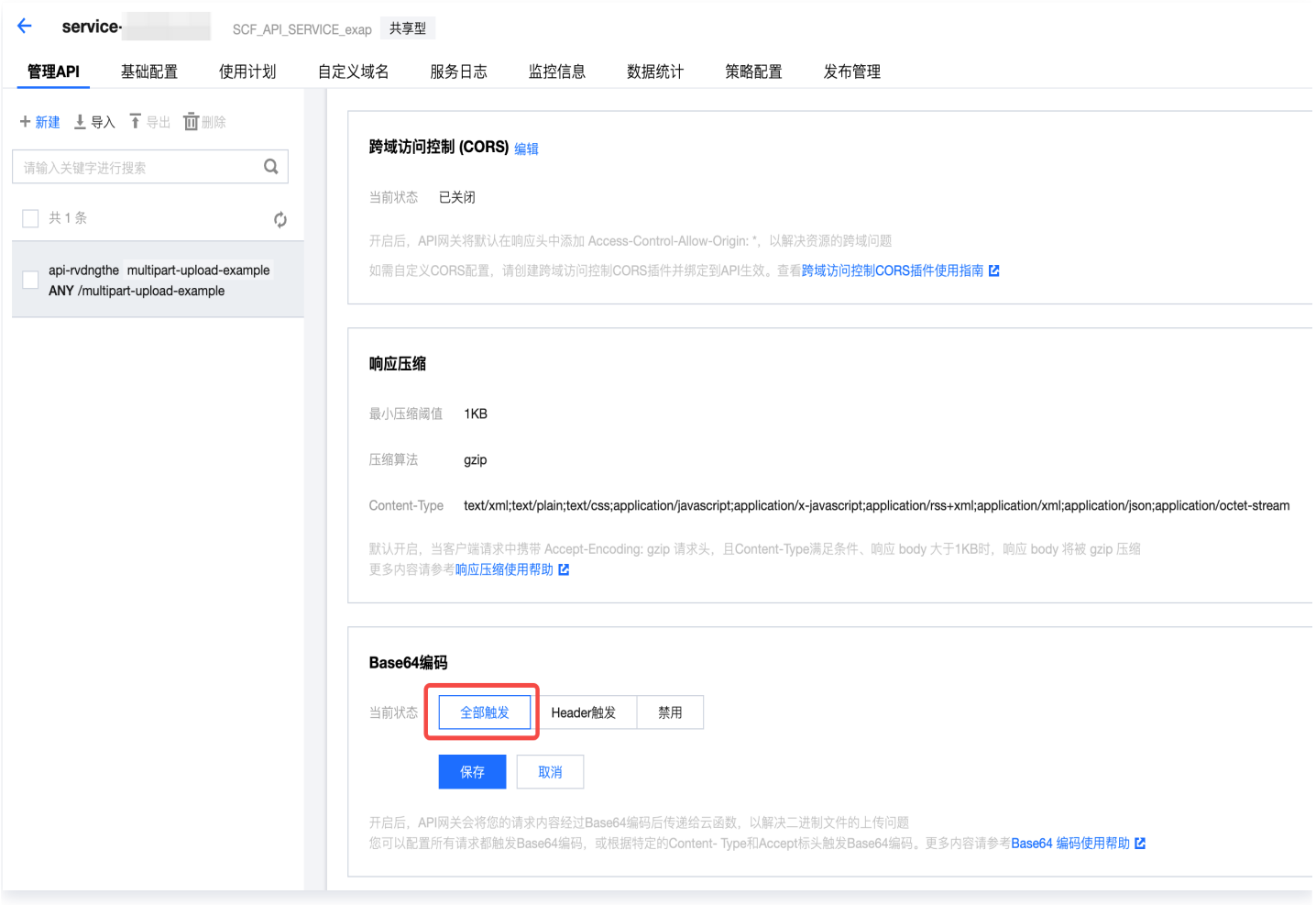
3. 单击**部署**，完成云函数的部署。

步骤3：绑定 API 网关触发器

在云函数的触发管理中，我们需要为云函数绑定 API 网关触发器，才能够处理用户具体的 HTTP 请求，具体的绑定方式和配置如下图：

这个时候，如果访问 API 网关绑定的链接，会发现虽然静态页面能够工作，但是上传图片后，页面没有展示正确的结果。这是因为默认情况下，API 网关没有开启 base64 编码功能，multipart formdata 被错误编码为字符串传

入 handler 函数，busboy 自然无法进行解码。
因此，我们需要进入 API 网关，找到绑定的 API点开详情，在API的基础配置中打开 base64 编码。



打开并发布服务后，我们的服务就可以正常工作了。

示例 Demo

您可访问腾讯云 Serverless 官方搭建好的 示例 Demo 来查看文件上传的效果。

API 网关传递给后端的结构体

最近更新时间：2024-11-05 14:59:02

操作场景

客户端请求 API 网关，API 网关会将客户端请求内容处理后再传递给后端，此时 API 网关传递给后端的结构体一般是固定的。
本文记录了对接不同后端类型时 API 网关传递给后端的结构体，以便您开发排障使用。

后端对接公网 URL/IP 和对接 VPC 内资源的结构体

```
GET / HTTP/1.1
Host: 10.0.0.0
Connection: keep-alive
X-Client-Proto: http
X-Client-Proto-Ver: HTTP/1.1
X-Forwarded-For: 100.100.10.1
X-Real-IP: 100.100.10.1
User-Agent: curl/7.29.0
Accept: /
x-b3-traceid: 12345678906f******7cd8db0fe843dc
X-Api-RequestId: 12345678906f******7cd8db0fe843dc
X-Api-Scheme: http
```

数据结构内容详细说明如下：

名称	内容
Host	用于指定虚拟主机
Connection	用于决定当前的事务完成后，是否会关闭网络连接
X-Client-Proto	记录客户端请求协议
X-Client-Proto-Ver	记录客户端请求协议版本
X-Forwarded-For	记录从客户端发起请求后客户端真实 IP 地址
X-Real-IP	记录每一层代理的IP，非客户端请求的真实IP
User-Agent	记录客户端请求工具

Accept	代表客户端希望接受的数据类型
x-b3-traceid	记录本次请求的 RequestId，等值于 X-Api-RequestId，用于适配 API 网关内部模块
X-Api-RequestId	记录本次请求的 RequestId
X-Api-Scheme	记录客户端请求协议，等值于 X-Client-Proto，用于适配 API 网关内部模块

注意：
由于不同 Content-Type 的请求 body 结构不同，且客户端请求可能添加自定义 Header，本文仅列出了经过 API 网关后的固定请求 Header。

后端对接 SCF 的结构体


```
{
    "body": "{key:value}",
    "requestContext": {
        "httpMethod": "ANY",
        "serviceId": "service-dlhbuxqh",
        "path": "/scf/{pathParam}",
        "sourceIp": "14.17.22.36",
        "identity": {},
        "stage": "release"
    },
    "queryStringParameters": {
        "queryParam": ""
    },
    "headers": {
        "content-length": "11",
        "x-b3-traceid": "12345678906f******7cd8db0fe843dc",
        "x-qualifier": "$DEFAULT",
        "accept": "/",
        "user-agent": "curl/7.69.1",
        "host": "service-abcdefgh-1234567890.gz.apigw.tencentcs.com",
        "requestsource": "APIGW",
        "x-api-scheme": "http",
        "x-api-requestid": " 12345678906f******7cd8db0fe843dc",
        "content-type": "application/x-www-form-urlencoded"
    },
```

```
    "pathParameters": {
        "pathParam": "mypath"
    },
    "queryString": {
        "a": "1",
        "b": "2"
    },
    "httpMethod": "POST",
    "headerParameters": {
        "headerparam": ""
    },
    "path": "/scf/mypath",
    "isBase64Encoded": "true",
}
```

数据结构内容详细说明如下：

名称	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。 • 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。
isBase64Encoded	记录在请求体是否被进行了 Base64 编码，取值为 true 或 false。