

# 云函数 开发者工具



腾讯云

**【 版权声明 】**

©2013–2025 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

**【 商标声明 】**

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

**【 服务声明 】**

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。

您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

**【 联系我们 】**

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

## 文档目录

### 开发者工具

Serverless Web IDE

Serverless Cloud Framework

概述

安装

权限管理

函数操作

开发调试

项目应用

支持命令列表

SCF VS Code 插件

开始使用

快速入门

安装说明

账号配置

函数操作

云端调试

插件更新日志

2.0版本升级说明

函数间调用 SDK

Node.js SDK

Python SDK

第三方工具

Malagu Framework

框架介绍

快速开始

访问数据库

# 开发者工具

## Serverless Web IDE

最近更新时间：2025-02-18 10:54:23

### 概述

Serverless Web IDE 是腾讯云 Serverless 和 CODING 基于浏览器的集成式开发环境 CloudStudio 深度合作推出的云函数在线开发 IDE，提供接近原生 IDE 的云端开发体验。

Serverless Web IDE 支持：

- 完整的函数开发、部署、测试能力。
- 终端能力，预置了常用的 pip, npm 等开发工具和云函数 SCF 已经支持的编程语言开发环境。
- 完整的 IDE 所含的基础能力，包括智能提示、代码自动补全等。
- 用户自定义 IDE 配置，在不同函数的在线开发中提供一致的 IDE 使用体验。

#### ⚠ 注意：

- 我们会为您保留 Serverless Web IDE 中的个性化配置以及代码状态，为了确保函数修改生效，请及时将修改部署到云端。
- 建议使用最新版本的 Google Chrome 浏览器以获得更好的 IDE 使用体验。

### 使用方式

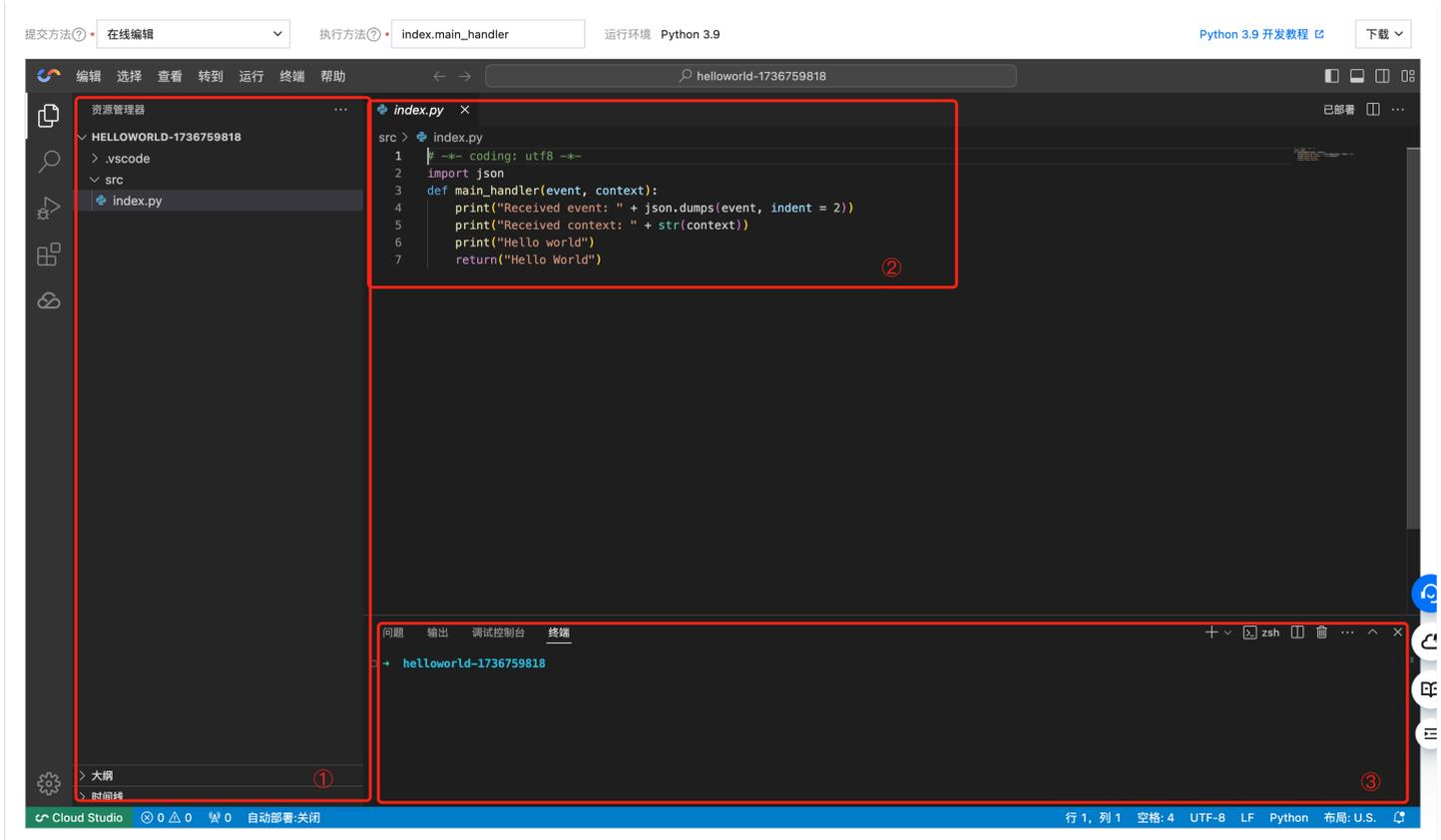
1. 登录 [Serverless 控制台](#)，在左侧选择[函数服务](#)。
2. 在函数列表中，单击函数名，进入该函数的详情页面。
3. 在函数管理页面中，选择[函数代码](#) > [在线编辑](#)，即可查看并编辑函数。

#### ⚠ 注意：

Java、Go 运行时暂不支持在线编辑，仅支持上传已经开发完成编译打包后的 ZIP 包或二进制文件。SCF 环境暂不提供 Java、Go 的编译能力。具体请参见 [Golang 部署指南](#)、[Java 部署指南](#)。

### 概览图

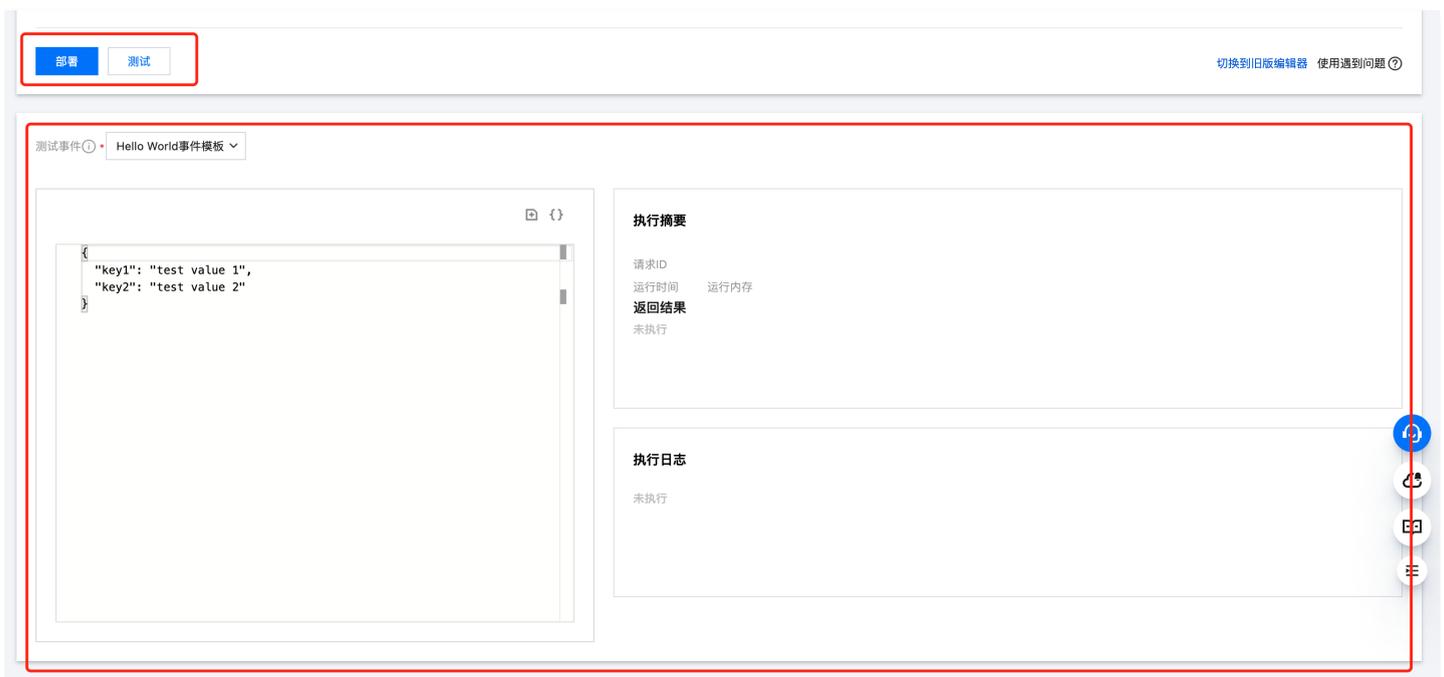
本文将 Serverless Web IDE 工具整体页面从左至右顺序依次介绍。如下图所示：



1. 资源管理器
2. 文件编辑区
3. 命令行终端

### 函数操作

在 Serverless Web IDE 中，可以完成函数代码编辑、部署、测试全流程操作。您可在下方选择测试模板，单击测试，函数将根据所选模板执行函数运行，并输出结果及日志。如下图所示：



### 函数部署

Serverless Web IDE 提供手动部署和自动部署两种函数部署方式，支持在线安装依赖。

- **部署方式:**

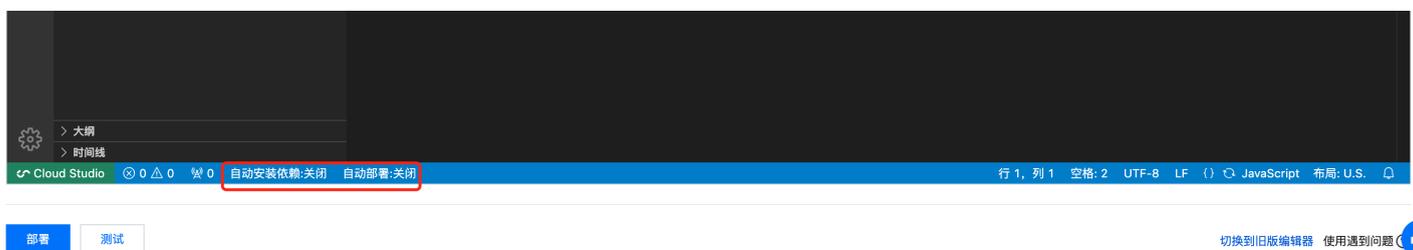
- **手动部署:** 手动部署模式下，您可以通过单击 IDE 右上角部署按钮触发函数部署到云端。
- **自动部署:** 自动部署模式下，保存 (ctrl + s 或 command + s) 即可触发函数部署到云端。

- **在线安装依赖:** 目前只支持 Node.js 运行环境，在线安装依赖开启后，在函数部署时会根据 package.json 中的配置自动安装依赖，详情可参考 [在线依赖安装](#)。

**⚠ 注意:**

- 函数的根目录为 /src，部署操作默认将 /src 目录下的文件打包上传。请将需要部署到云端的文件放在 /src 目录下。
- 自动部署模式下保存即触发函数部署到云端，不建议在有流量的函数上开启。

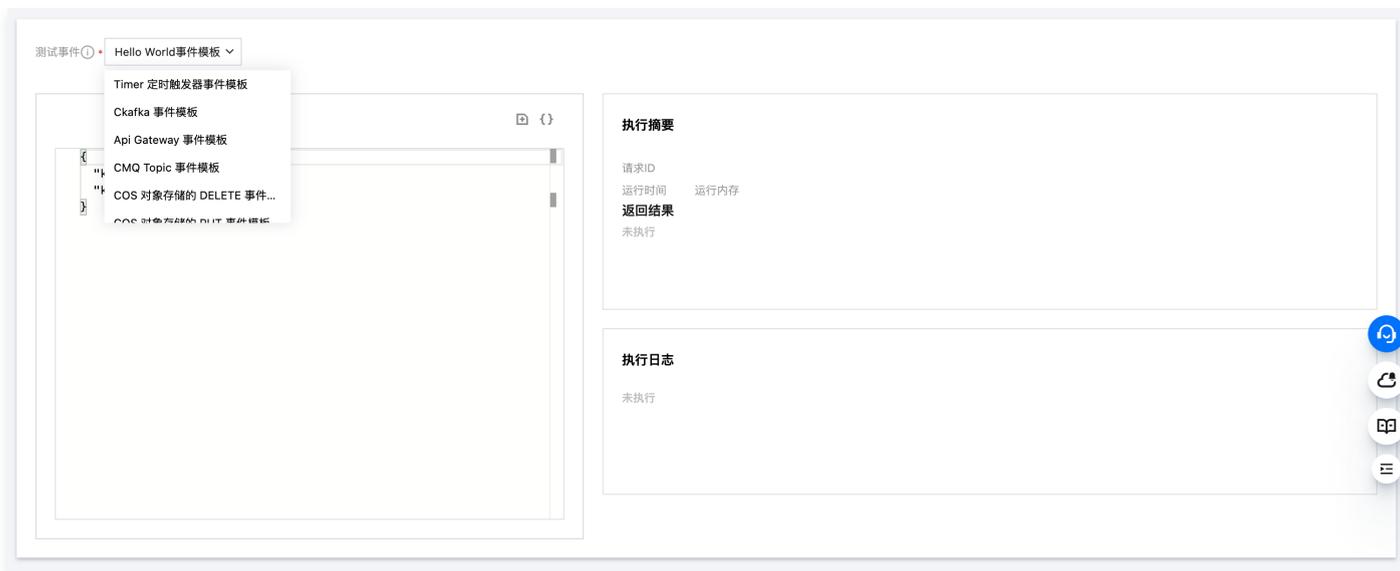
切换部署方式和启用在线依赖安装可通过单击 IDE 下方操作区箭头的**自动安装依赖**、**自动部署**进行切换，**自动部署: 关闭**则代表手动部署模式。



## 函数测试

您可以单击 IDE 下方的**测试**触发函数运行，并在输出中查看函数运行结果。

- **选择测试模板:** 单击测试事件模块的**测试模板**选择函数测试触发事件。

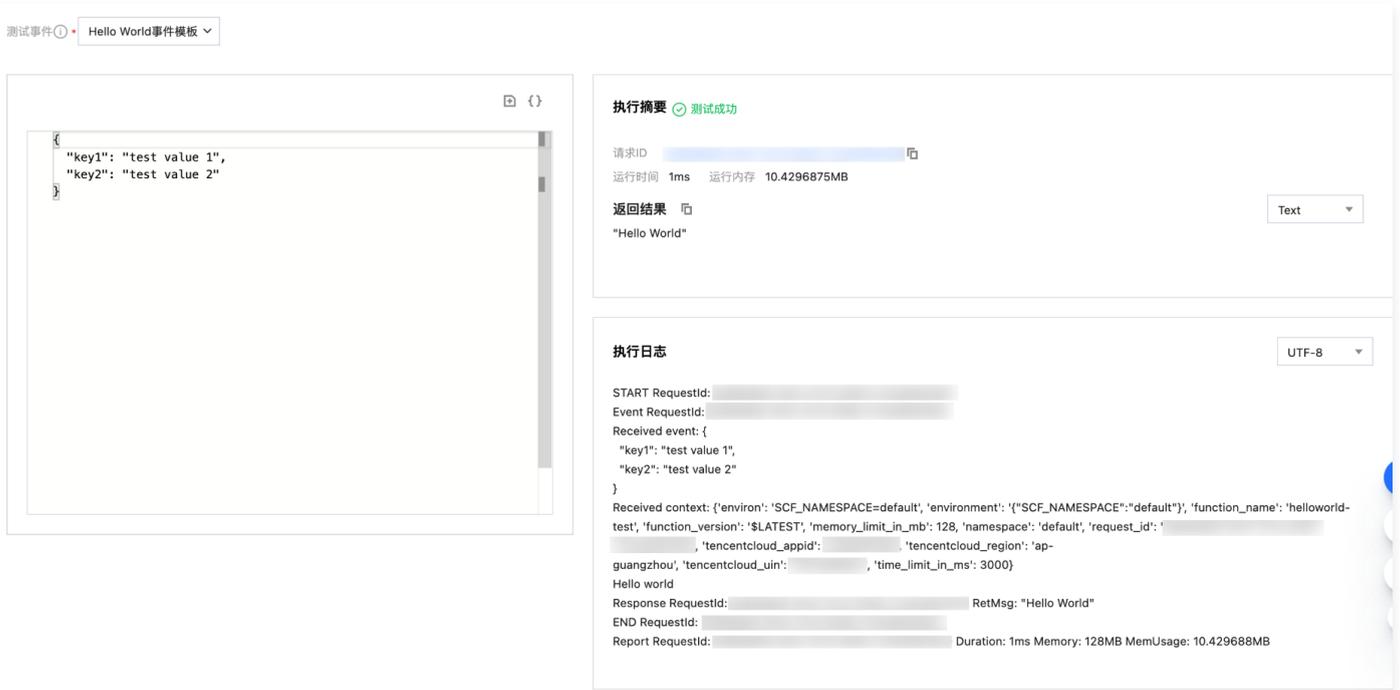


- **新增测试模板:** 如果现有的测试模板不能满足您的测试需求，您可以直接修改测试模板，并另存为新的自定义测试事件，新增测试事件将以 JSON 文件的格式存储在函数根目录 /src 下的 scf\_test\_event 文件夹中，跟随函数一起部署到云端。如下图所示:



## 查看日志

您可以在输出中查看函数测试结果，包括返回数据 Response、日志 Output 和函数执行摘要 Summary。



## 更多操作

在资源管理器函数文件上单击右键展开的列表中，包含了函数相关的全部操作。除部署、测试、新增测试模板等操作外，还提供以下内容：

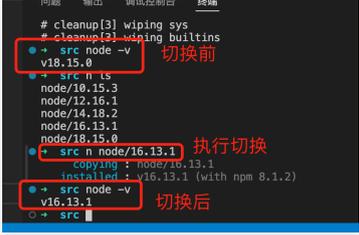
- **生成 serverless.yml**：将函数当前的配置写入配置文件 serverless.yml，可以使用 Serverless Cloud Framework 命令行工具进行二次开发。
- **丢弃当前修改**：重新拉取云端已经部署的函数覆盖当前工作区。

## IDE 操作

Serverless Web IDE 中常用命令、运行环境和预置的扩展版本如下所示：

### 常用命令

命令	版本
python3	Python 3.7.12 ( Web IDE 默认的 Python 版本，如果想使用其他 Python 版本执行命令，需在命令前指定具体版本，如 <code>python39 --version</code> )。

python37	Python 3.7.12
python36	Python 3.6.1
python27	Python 2.7.13
python	Python 2.7.13
node	<p>Node.js 18.15.0。node 命令默认跟随最新 Node.js 版本，当前环境中已安装 Node.js 18.15.0、Node.js 16.13.1、Node.js 14.18.2、Node.js 12.16.1、Node.js 10.15.3，可在终端执行 <code>n</code> 命令进行切换 如 <code>n node/16.13.1</code>。</p> 
php80	PHP 8.0.13
php74	PHP 7.4.26
php	<p>PHP 7.2.2。终端内默认的 PHP 版本为7.2.2，可在终端执行 <code>ln</code> 命令切换版本。                      例如执行 <code>ln -sf /var/lang/php80/bin/php /usr/bin/php</code> 以切换至 PHP 8.0。</p>
pip3	pip 22.0.4 (python 3.7)
pip37	pip 22.0.4 (python 3.7)
pip36	pip 21.3.1 (python 3.6)
pip	pip 20.3.4 (python 2.7)
npm	8.1.2
composer	2.2.9

**常用工具**

工具	版本
yarn	1.22.18
wget	1.14
zip、unzip	6
Git	2.24.1
zsh	5.0.2
dash	0.5.10.2
make	3.82
jupyter	4.6.3
pylint	1.9.5
<a href="#">Serverless Cloud Framework</a>	3.2.1

**运行环境**

运行环境	版本
Node.js	16.13、14.18、12.16、10.15
Python	3.7、3.6、2.7
PHP	8.0、7.4、7.2、5.6

## 扩展

扩展	版本
Python	2020.11.371526539
Jupyter	2020.12.411183155
PHP-IntelliSense	2.3.14
ESLint	2.1.13
Prettier	5.8.0

## 配额限制

- IDE 为每个用户提供5GB的存储空间，超出将无法执行写入操作，请及时清理。请注意，删除函数不会清空 IDE 的存储空间，在备份工作空间更改后，请您手动进行“重置工作空间”操作。您也可以选择切换到旧版编辑器以规避此限制。
- 为了保证良好的使用体验，不建议在多个浏览器页面中同时打开3个以上的函数。

## 注意事项

在 IDE 中执行以下操作可能会带来数据泄露等安全隐患，如必须执行，请谨慎操作。

- 安装 phpmyadmin、struts2 等高危开源组件。

## 常见问题

### 1. IDE 加载异常如何处理？

如果遇到工作空间异常无法正常启动等情况，您可以单击 IDE 右下角“使用遇到问题”，在展开的页面中单击“重置工作空间”将工作空间初始化。

### 2. 函数在终端中执行成功，点击测试执行失败？

在线 IDE 的终端和 SCF 的云上运行环境相互独立，单击“测试”后返回的运行结果即为函数真实的执行结果。

### 3. 在终端中执行命令，结果不符合预期？

如果执行依赖包安装相关命令，请确保在 `src` 目录下进行操作。

在使用命令前，请先查看命令的版本，默认支持的命令列表请参见 [常用命令](#)。

# Serverless Cloud Framework

## 概述

最近更新时间：2025-01-21 18:21:42

### 简介

Serverless Cloud Framework 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 Cloud Framework 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。

### 云函数 SCF 组件

Serverless Cloud Framework 提供了云函数 SCF 组件，您使用该组件快速打包部署云函数项目。可通过以下步骤，快速熟悉并使用组件：

1. 通过 [函数操作](#)，快速上手 Serverless Cloud Framework。
2. 通过 [开发调试](#)，了解 Serverless Cloud Framework 如何开发及调试云函数。
3. 通过 [项目应用](#)，了解如何进行多个云函数的项目管理和资源编排。

### 应用实践

Serverless Cloud Framework 提供的 SCF 组件，实现了对云函数的资源创建和编排。同时针对一些典型的用户应用场景提供了更多组件的封装和最佳案例实践。例如 Express 框架支持及网站部署等，详情请参见 [Serverless 应用中心](#)。

项目	描述
<a href="#">部署静态网站托管</a>	通过 Serverless Website 组件快速托管一个静态网站。
<a href="#">部署 Express.js 应用</a>	通过 Serverless SCF 组件快速构建一个 Express.js 项目。
<a href="#">部署 Nuxt.js 应用</a>	通过 Serverless Components Nuxt.js 组件，快速部署一个基于 Nuxt.js 的 SSR 项目。

# 安装

最近更新时间：2022-10-20 15:22:54

您可以通过 NPM 安装 Serverless Cloud Framework。

## 通过 NPM 安装

### 安装前提

使用 npm 安装前，需要确保您的环境中已安装好了 Node（版本需要 > 12）以及 npm（查看 [Node.js 安装指南](#)）。

```
$ node -v
v12.18.0

$ npm -v
7.0.10
```

#### ⚠ 注意

- 为保证安装速度和稳定性，建议您使用 cnpm 来完成安装：先下载安装 cnpm，然后将下面所有使用的 npm 命令替换为 cnpm 即可。
- scf 是 serverless-cloud-framework 命令的简写。

### 安装步骤

在命令行中运行如下命令：

```
npm i -g serverless-cloud-framework
```

#### 📌 说明

如 MacOS 提示无权限，则需要运行 `sudo npm i -g serverless-cloud-framework` 进行安装。

如果之前您已经安装过 Serverless Cloud Framework，可以通过以下命令升级到最新版。

```
npm update -g serverless-cloud-framework
```

### 查看版本信息

安装完毕后，通过运行 `scf -v` 命令，查看 Serverless Cloud Framework 的版本信息：

```
scf -v
```

## 相关操作

下一步：快速开始

- [快速部署函数模板](#)
- [快速创建应用模板](#)

# 权限管理

最近更新时间：2025-06-13 15:22:02

本文为您介绍 Serverless Cloud Framework 的几种授权方式以及通过配置子账号权限进行实际操作演示。

## 前提条件

Serverless Cloud Framework 帮助您将项目快速部署到腾讯云 Serverless 应用中心，因此在部署前，请确认您已经 [注册腾讯云账号](#) 并完成 [实名认证](#)。

## 授权方式

### 扫码一键授权

通过 `scf deploy` 进行部署时，您可以通过扫描二维码，一键授权并快速部署，详细步骤请参见 [部署函数](#)，扫码授权后，会生成临时密钥信息（过期时间为 60 分钟）写入当前目录下的 `.env` 文件中：

```
TENCENT_APP_ID=xxxxxxx #授权账号的 AppId
TENCENT_SECRET_ID=xxxxxxx #授权账号的 SecretId
TENCENT_SECRET_KEY=xxxxxxx #授权账号的 SecretKey
TENCENT_TOKEN=xxxxxxx #临时 token
```

一键授权时获取的权限详情请参见 [scf\\_QcsRole 角色权限列表](#)。

#### 说明

如果您的账号为腾讯云子账号，扫码部署前需要主账号先进行策略授权配置。配置详情请参见 [子账号权限配置](#)。

## 本地密钥授权

为了避免扫码授权过期进行重复授权，您可以采用密钥授权方式。在部署的根目录下创建 `.env` 文件，并配置腾讯云的 SecretId 和 SecretKey 信息：

```
# .env
TENCENT_SECRET_ID=xxxxxxxxxxxx #您账号的 SecretId
TENCENT_SECRET_KEY=xxxxxxxxxx #您账号的 SecretKey
```

SecretId 和 SecretKey 可以在 [API 密钥管理](#) 中获取。

#### 说明

为了账号安全性，密钥授权时建议使用子账号密钥。子账号必须先被授予相关权限才能进行部署。配置详情请参见 [子账号权限配置](#)。

## 永久密钥配置

通过 `scf credentials` 指令，可以快速设置全局密钥信息永久保存。该指令必须在已经创建好的 scf 项目下进行配置，请确保您已经通过 `scf init` 或已经手动创建好您的带有 `serverless.yml` 的项目。

#### 全部指令参考如下：

```
scf credentials 管理全局用户授权信息
set             存储用户授权信息
  --secretId / -i      (必填) 腾讯云 CAM 账号 secretId
  --secretKey / -k    (必填) 腾讯云 CAM 账号 secretKey
  --profile / -n {name} 授权名称，默认为 "default"
  --overwrite / -o   覆写已有授权名称的密钥
remove         移除用户授权信息
  --profile / -n {name} (必填) 授权名称
list          查看用户授权信息
```

- 配置全局授权信息:

```
# 通过默认 profile 名称配置授权信息
$ scf credentials set --secretId xxx --secretKey xxx

# 通过指定 profile 名称配置授权信息
$ scf credentials set --secretId xxx --secretKey xxx --profile profileName1

# 更新指定 profile 名称里的授权信息
$ scf credentials set --secretId xxx --secretKey xxx --profile profileName1 --overwrite
```

- 删除全局授权信息:

```
$ scf credentials remove --profile profileName1
```

- 查看当前所有授权信息:

```
$ scf credentials list
```

- 通过全局授权信息部署:

```
# 通过默认 profile 部署
$ scf deploy
# 通过指定 profile 部署
$ scf deploy --profile newP
# 忽略全局变量, 扫码部署
$ scf deploy --login
```

## 子账号权限配置

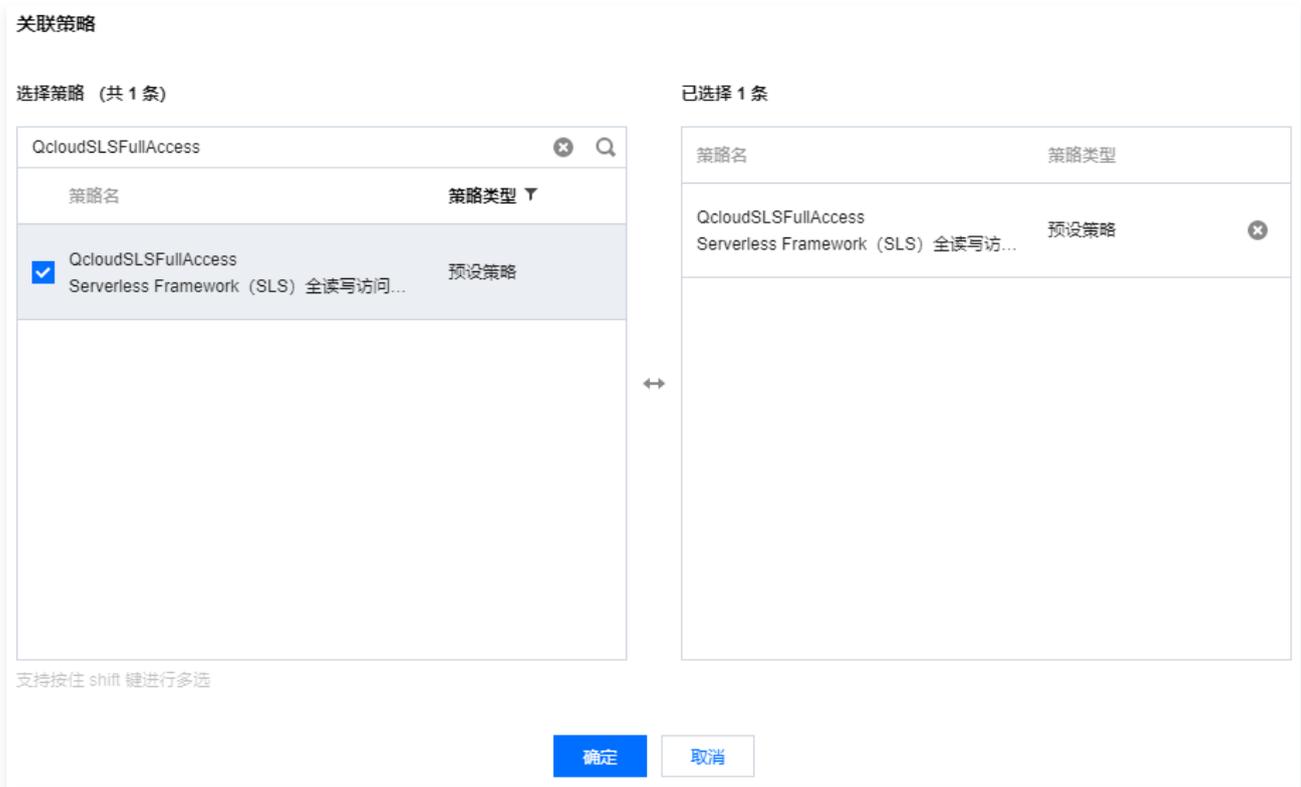
### 配置步骤

如果您的操作账号为腾讯云子账号, 没有默认操作权限, 则需要主账号 (或拥有授权操作的子账号) 进行如下授权操作:

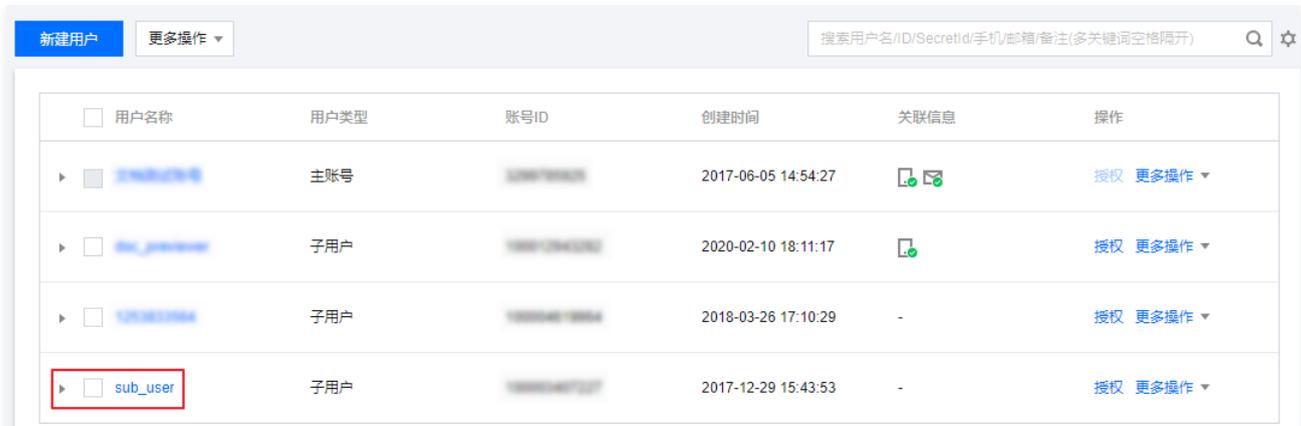
1. 在 [CAM 用户列表](#) 页, 选取对应子账号, 单击授权。

<input type="checkbox"/>	用户名称	用户类型	账号ID	创建时间	关联信息	操作
<input type="checkbox"/>	主账号	主账号		2017-06-05 14:54:27		<a href="#">授权</a> <a href="#">更多操作</a>
<input type="checkbox"/>	子用户	子用户		2020-02-10 18:11:17		<a href="#">授权</a> <a href="#">更多操作</a>
<input type="checkbox"/>	子用户	子用户		2018-03-26 17:10:29	-	<a href="#">授权</a> <a href="#">更多操作</a>
<input type="checkbox"/>	sub_user	子用户		2017-12-29 15:43:53	-	<a href="#">授权</a> <a href="#">更多操作</a>

2. 在弹出的窗口内，搜索并选中 `QcloudSLSFullAccess`，单击确定，完成授予子账号 Serverless Framework 所有资源的操作权限。



3. 在 **CAM 用户列表** 页，选取对应子账号，单击用户名称，进入用户详情页。



4. 单击**关联策略**，在添加策略页面单击从策略列表中选择策略关联 > **新建自定义策略**。

关联策略页面：



新建策略页面：



5. 选择按策略语法创建 > 空白模板，填入如下内容，注意角色参数替换为您的主账号 UIN：

```

{
  "version": "2.0",
  "statement": [
    {
      "action": [
        "cam:PassRole"
      ],
      "resource": [
        "qcs::cam::uin/${填入账号的 uin}:roleName/scf_QcsRole"
      ],
      "effect": "allow"
    },
    {
      "resource": [
        "*"
      ],
      "action": [
        "name/sts:AssumeRole"
      ],
      "effect": "allow"
    }
  ]
}
    
```

6. 完成自定义策略配置后，回到第 4 步的授权页面，搜索刚刚创建的自定义策略，单击下一步> 确定，即可授予子账号 scf\_QcsRole 的操作权限，此时，您的子账号应该拥有一个自定义策略和一个 QcloudscfFullAccess 的预设策略，可以完成 Serverless Framework 的正常使用。



**说明**

除了授权调用默认角色 `scf_QcsRole` 外，也可给予子账号授权调用自定义角色。通过自定义角色中的细粒度权限策略，达到权限收缩的目的。详情请参见 [指定操作角色配置](#)。

**scf\_QcsRole 角色权限列表**

策略	描述
QcloudCOSFullAccess	COS（对象存储）全读写访问权限
QcloudSCFFullAccess	SCF（云函数）全读写权限
QcloudSSLFullAccess	SSL 证书（SSL）全读写访问权限
QcloudTCBFullAccess	TCB（云开发）全读写权限
QcloudAPIGWFullAccess	APIGW（API 网关）全读写权限
QcloudVPCFullAccess	VPC（私有网络）全读写权限
QcloudMonitorFullAccess	Monitor（腾讯云可观测平台）全读写权限
QcloudslsFullAccess	sls（Serverless Framework）全读写权限
QcloudCDNFullAccess	CDN（内容分发网络）全读写权限
QcloudCKafkaFullAccess	CKafka（消息队列 CKafka）全读写权限
QcloudCodingFullAccess	CODING DevOps 全读写访问权限
QcloudPostgreSQLFullAccess	云数据库 PostgreSQL 全读写访问权限
QcloudCynosDBFullAccess	云数据库 CynosDB 全读写访问权限
QcloudCLSFullAccess	日志服务（CLS）全读写访问权限
QcloudAccessForscfRole	该策略供 Serverless Framework（sls）服务角色（scf_QCSRole）进行关联，用于 scf 一键体验功能访问其他云服务资源。包含访问管理（CAM）相关操作权限

# 函数操作

最近更新时间：2024-08-28 16:44:51

## ⚠ 注意：

由于域名备案更新，目前 cli 部署流程无法通过扫码登录，您可以通过本地配置永久密钥，或者根据命令行提示，通过访问 URL 完成登录，详情见[账号和权限配置](#)。

## 操作场景

该任务指导您通过 Serverless Cloud Framework，在腾讯云上快速创建、配置和部署一个 SCF 云函数应用。

## 前提条件

- 已经 [安装 Serverless Cloud Framework 1.67.2](#) 以上版本。

```
npm install -g serverless-cloud-framework
```

- 已经 [注册腾讯云账号](#) 并完成 [实名认证](#)。

## 📌 说明：

如果您的账号为[腾讯云子账号](#)，请先联系主账号，参考[账号和权限配置](#)进行授权。

## 操作步骤

### 快速部署

在空文件夹目录下，执行如下指令：

```
serverless-cloud-framework
```

接下来按照交互提示，完成项目初始化，应用请选择 `scf-starter` 模板，并选择您希望使用的运行时（此处以 Node.js 为例）：

```
serverless-cloud-framework: 当前未检测到 Serverless 项目，是否希望新建一个项目? Yes
serverless-cloud-framework: 请选择您希望创建的 Serverless 应用 scf-starter - 快速部署一个云函数

  react-starter - 快速部署一个 React.js 应用
  restful-api - 快速部署一个 REST API 使用 python + API gateway
  > scf-starter - 快速部署一个云函数
  vue-starter - 快速部署一个 Vue.js 基础应用
  website-starter - 快速部署一个静态网站
  eggjs-starter - 快速部署一个Egg.js 基础应用
  express-starter - 快速部署一个 Express.js 基础应用

serverless-cloud-framework: 请选择应用的运行时 scf-nodejs - 快速部署一个 nodejs 云函数

  scf-golang - 快速部署一个 golang 云函数
  > scf-nodejs - 快速部署一个 nodejs 云函数
  scf-php - 快速部署一个 PHP 云函数
  scf-python - 快速部署一个 python 云函数

serverless-cloud-framework: 请输入项目名称 demo
serverless-cloud-framework: 正在安装 scf-nodejs 应用...

scf-nodejs > Created
```

```
demo 项目已成功创建!
```

选择立即部署，将已经初始化好的项目快速部署到云函数控制台：

```
serverless-cloud-framework: 是否希望立即将该项目部署到云端? Yes

xxxxxxx
x  QR  x
x CODE x
xxxxxxx
请使用微信扫描上方二维码或者点击下方链接登录
https://scflogin.qcloud.com/XKYUcbaK
登录成功!

serverless-cloud-framework
Action: "deploy" - Stage: "dev" - App: "scfApp" - Instance: "scfdemo"

functionName: helloworld
description: helloworld 空白模板函数
namespace: default
runtime: Nodejs10.15
handler: index.main_handler
memorySize: 128
lastVersion: $LATEST
traffic: 1
triggers:
  apigw:
    - http://service-xxxxxxx.gz.apigw.tencentcs.com/release/

27s > scfdemo > Success
```

部署完毕后，通过以下指令，完成函数的远程调用：

```
scf invoke --inputs function=helloworld
```

**说明：**  
scf 是 serverless-cloud-framework 命令的简写。

## 查看部署信息

如果希望再次查看应用的部署状态和资源，可以进入到部署成功的文件夹，运行如下命令，查看对应信息：

```
cd demo #进入项目目录，此处请改为您的项目目录名称
scf info
```

## 查看目录结构

在初始化的项目目录下，可以看到一个 Serverless 函数项目的最基本结构：

```
.
├── serverless.yml # 配置文件
├── index.js      # 入口函数
└── .env         # 环境变量文件
```

- serverless.yml 配置文件实现了函数基本信息的快速配置，函数控制台支持的配置项都支持在 yml 文件里配置（查看 [云函数的全量配置信息](#)）。
- index.js 为项目的入口函数，此处为 helloworld 模板。

- .env 文件里存放了用户登录的鉴权信息，您也可以在里面配置其它环境变量。

## 重新部署

在本地项目目录下，您可以对函数模板项目内容与配置文件进行修改，并通过以下指令进行重新部署：

```
scf deploy
```

### ⓘ 说明：

如需查看移除过程中的详细信息，可以增加 `--debug` 参数进行查看。

## 持续开发

部署完成后，Serverless Cloud Framework 支持通过不同指令，帮助您完成项目的持续开发部署、灰度发布等能力，您也可以结合其他组件一起使用，完成多组件应用的部署管理。

详情请参考文档 [应用管理](#) 与 [支持命令列表](#)。

## 常见问题

### 输入 `serverless-cloud-framework` 时没有默认弹出中文引导

解决方案：在 .env 文件中增加配置 `SERVERLESS_PLATFORM_VENDOR=tencent` 即可。

### 在境外网络环境，执行 `scf deploy` 后部署十分缓慢

解决方案：在 .env 文件中增加配置 `GLOBAL_ACCELERATOR_NA=true` 则开启境外加速。

### 执行 `scf deploy` 后部署报网络错误

解决方案：在 .env 文件中增加以下代理配置。

```
HTTP_PROXY=http://127.0.0.1:12345 #请将'12345'替换为您的代理端口
HTTPS_PROXY=http://127.0.0.1:12345 #请将'12345'替换为您的代理端口
```

# 开发调试

最近更新时间：2022-10-19 16:20:30

## 开发模式

Serverless Cloud Framework 支持开发模式（dev 模式），处于开发状态下的项目可以更便捷的进行代码编写及开发调试。在开发模式中，用户可以持续地进行开发 - 调试的过程，减少了打包、更新等其他工作的干扰。

### 进入开发模式

在项目下执行 `scf dev` 命令，可以进入项目的开发模式。示例如下：

#### ⚠ 注意

目前 `scf dev` 仅支持 Node.js 10.15 及 12.16 运行环境。

```
$ scf dev
serverless-cloud-framework
Dev Mode - Watching your Component for changes and enabling streaming logs, if supported...
Debugging listening on ws://127.0.0.1:9222.
For help see https://nodejs.org/en/docs/inspector.
Please open chrome, and visit chrome://inspect, click [Open dedicated DevTools for Node] to debug your
code.
----- The realtime log -----
17:13:38 - express-api-demo - deployment
region: ap-guangzhou
apigw:
  serviceId:  service-b77xtixx
  subDomain:  service-b77xtixx-12539702xx.gz.apigw.tencentcs.com
  environment:  release
  url:         http://service-b77xtixx-12539702xx.gz.apigw.tencentcs.com/release/
scf:
  functionName:  express_component_6r6xkh60k
  runtime:       Nodejs10.15
  namespace:    default
express-api-demo > Watching
```

在进入 dev 模式后，Serverless 工具将输出部署的内容，并启动持续文件监控。当代码文件有更新时，将自动再次进行部署，将本地文件更新到云端。

### 退出开发模式

在开发模式下，可通过 `Ctrl+C` 退出。返回结果如下所示：

```
express-api-demo > Disabling Dev Mode & Closing ...
express-api-demo > Dev Mode Closed
```

## 命令调试

Serverless Cloud Framework 支持使用 `invoke` 命令触发云函数进行调试。使用 `scf deploy` 命令部署成功的云函数，可在项目目录下执行以下命令，进行调试：

```
scf invoke --inputs function=functionName clientContext='{"weights":{"2":0.1}}'
```

#### 📌 说明

- `invoke` 命令须在该函数部署的 `serverless.yml` 文件同目录下执行。

- `clientContext` 为触发函数时传递的 JSON 字符串。可以根据 [触发事件模板](#) 的 JSON 字符串格式模拟不同触发事件。

## 云端调试

Runtime 为 Node.js 10+ 的项目，可开启云端调试，使用调试工具来连接远程环境并进行调试。例如，Chrome DevTools、VS Code Debugger。

### 开启云端调试

执行步骤 [进入开发模式](#) 时，如果项目是 Runtime 为 Node.js 10及以上版本的函数，会自行开启云端调试，并输出调试相关信息。

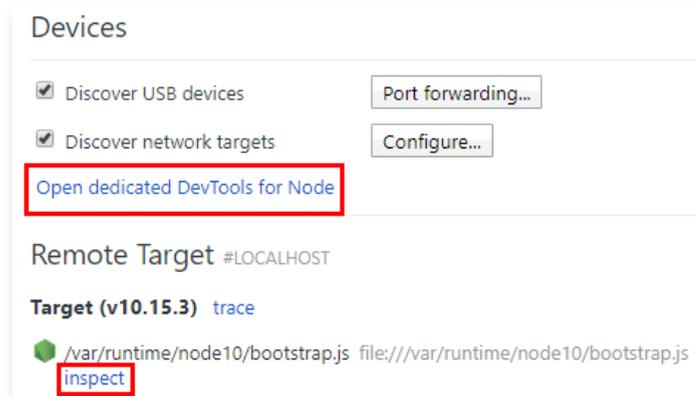
例如，在开启开发模式时，输出结果包含类似如下信息，则代表已经启动该项目的云端调试：

```
Debugging listening on ws://127.0.0.1:9222.  
For help see https://nodejs.org/en/docs/inspector.  
Please open chrome, and visit chrome://inspect, click [Open dedicated DevTools for Node] to debug your  
code.
```

## 使用调试工具 Chrome DevTools

以下步骤说明如何使用 Chrome 浏览器的 DevTools 工具来连接远程环境并进行调试：

- 启动 Chrome 浏览器。
- 在地址栏中输入 `chrome://inspect/` 并访问。
- 可通过以下两种方式打开 DevTools。如下图所示：



3.1（推荐）单击 Devices 下的 **Open dedicated DevTools for Node**。

3.2 选择 Remote Target #LOCALHOST 中具体 Target 下的 **inspect**。

如果无法打开或者没有 Target，请检查 Device 的 Configure 中是否已有 `localhost:9229` 或 `localhost:9222` 的配置，该配置对应开启云端调试时的输出。

- 通过选择 **Open dedicated DevTools for Node** 方式打开的 DevTools 调试工具，可单击 **Sources** 页签看远端代码。函数的实际代码在 `/var/user/` 目录下。

在 **Sources** 页签中查看的代码可能处于加载中，会随着调试进行而展示出更多远端文件。

- 可按需打开文件，在文件的指定位置设置断点。
- 通过任意方式，例如 URL 访问、页面触发、命令触发、接口触发等方式触发函数，会使得远端环境开始运行，并会在设置了断点的位置中断，等待进一步的运行。
- 通过 DevTools 的右侧工具栏，可以控制中断的程序继续执行、单步执行、步入步出等操作，也可以直接查看当前变量，或设定需跟踪查看的变量。DevTools 的进一步使用可以搜索查询 DevTools 使用说明文档。

### 关闭云端调试

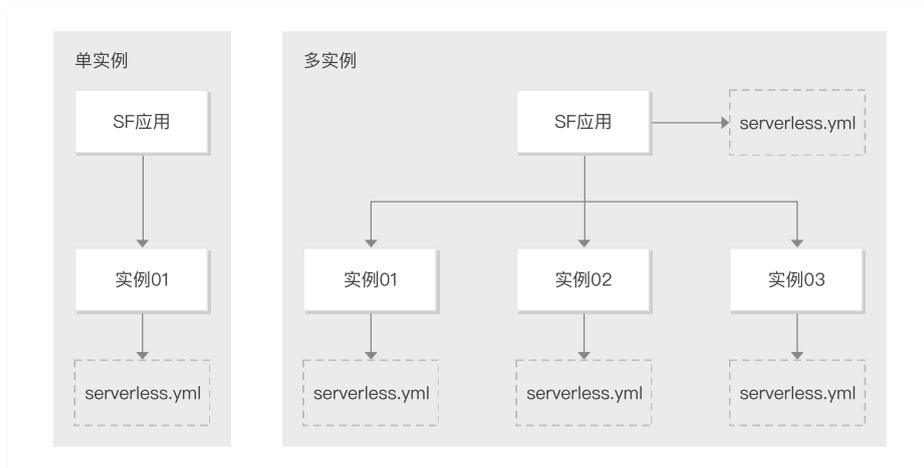
在退出开发模式时，将会自动关闭云端调试功能。

# 项目应用

最近更新时间：2023-03-31 11:15:49

## 简介

在 [函数操作](#) 中介绍了如何使用 Serverless Cloud Framework 创建云函数。对于 Serverless Cloud Framework 框架本身而言，即通过云函数组件部署了一个 Serverless 单实例应用。一个 Serverless 应用可以由单个或者多个实例构成，每个组件部署对应一个实例。每个实例都会涉及一个 `serverless.yml` 文件，该文件定义了组件的参数，这些参数在部署时用于生成实例的信息。例如，`region` 定义了资源的所在区。如下图所示：



您可以通过本文了解单实例及多实例应用，并可结合实际场景进行云函数的项目管理和资源编排。

## 单实例应用

单实例应用即项目中只引入一个组件，部署时只生成一个组件实例。单实例应用中的应用名称由 Serverless Cloud Framework 框架默认生成。

**适用场景：** Serverless Cloud Framework 只作为 CLI 工具创建更新函数，用户需自行编排和管理函数资源。

## 多实例应用

多实例应用即项目中引入多个组件，部署时生成多个组件实例。多实例应用需要有固定的应用名称，以保证所有组件在统一的应用下进行管理。

**适用场景：** 用户需通过 Serverless Framework 组织和编排项目中多个云函数资源。

## 开发项目

Serverless Cloud Framework 提供了一套 [资源编排](#)、[环境隔离](#) 及 [灰度发布](#) 的管理机制。除了云函数创建，Serverless Cloud Framework 还提供了丰富的组件用于操作 API 网关、对象存储 COS 和访问管理 CAM 等云资源产品。使用 Serverless Cloud Framework 开发项目，可以专注于业务本身的开发，提升效率。项目开发详情请参见 [Serverless Cloud Framework](#)。

# 支持命令列表

最近更新时间：2024-08-19 18:08:31

Serverless 应用基于 Serverless Cloud Framework 部署，支持的 CLI 命令如下：

## ⚠ 注意：

`scf` 是 `serverless-cloud-framework` 命令的简写。

1. **scf registry**: 查看可用的 Components 列表。
2. **scf registry publish**: 发布 Component 到 Serverless 组件仓库。
  - `--dev` : 支持 `dev` 参数用于发布 `@dev` 版本的 Component，用于开发或测试。
3. **scf init xxx**: 从组件仓库下载指定模板，`xxx` 为模板名称。例如：`$ scf init fullstack`
  - `--name my-app` : 支持自定义项目目录名称。
  - `--debug` : 列出模板下载过程中的日志信息。
4. **scf deploy**: 部署 Component 实例到云端。
  - `--debug` : 列出组件部署过程中 `console.log()` 输出的部署操作和状态等日志信息。
  - `--inputs publish=true` : 部署函数时发布新版本。
  - `--inputs traffic=0.1` : 部署时切换 10% 流量到 `$latest` 函数版本，其余流量到最后一次发布的函数版本上。

## 📌 说明：

旧版本命令为 `scf deploy --inputs.key=value`，Serverless CLI V3.2.3 后命令统一格式为

`scf deploy --inputs key=value`，旧版本命令在新版本 Serverless CLI 中不可用，升级 Serverless CLI 的用户请使用新版本命令。

5. **scf remove**: 从云端移除一个 Component 实例。
  - `--debug` : 列出组件移除过程中 `console.log()` 输出的移除操作和状态等日志信息。
6. **scf info**: 获取并展示一个 Component 实例的相关信息。
  - `--debug` : 列出更多 state。
7. **scf dev**: 启动 DEV MODE 开发者模式，通过检测 Component 的状态变化，自动部署变更信息。同时支持在命令行中实时输出运行日志，调用信息和错误等。此外，支持对 Node.js 应用进行云端调试。
8. **scf login**: 支持通过 `login` 命令，通过微信扫描二维码的方式，登录腾讯云账号并授权对关联资源进行操作。

# SCF VS Code 插件

## 开始使用

最近更新时间：2023-02-10 09:58:09

### 简介

Tencent Serverless Toolkit for VS Code 是腾讯云 Serverless 产品的 VS Code ( Visual Studio Code ) IDE 的插件。该插件可以让您更好地在本地进行 Serverless 项目开发 and 代码调试，并且轻松将项目部署到云端。

您可参考本文，开始使用腾讯云云函数 VS Code 插件 Tencent Serverless。

通过该 VS Code 插件，您可以：

- 拉取云端的云函数列表，并触发云函数在云端运行。
- 在本地快速创建云函数项目。
- 在本地开发、调试及测试您的云函数代码。
- 使用模拟的 COS、CMQ、CKafka、API 网关等触发器事件来触发函数运行。
- 上传函数代码到云端，更新函数配置。

### 快速入门

插件安装及使用方法请参见 [快速入门](#)。

### 配置规范

当前插件使用的是 Serverless Cloud Framework 中 tencent-component 的语法规范，具体语法规范可参见 [说明文档](#)。

### 常见问题

- 安装或使用过程中如遇到问题，可参考 [SCF 工具类常见问题](#) 解决，您也可以通过在 [Tencent Serverless Toolkit for VS Code 项目](#) 中 [提交 issue](#) 的方式来记录和跟踪您碰到的问题。
- VS Code 插件暂不支持镜像部署函数和 Web 函数。

# 快速入门

最近更新时间：2024-08-20 15:55:41

## 操作场景

Tencent Serverless Toolkit for VS Code 是腾讯云 Serverless 产品的 VS Code ( Visual Studio Code ) IDE 的插件。该插件可以让您更好的在本地进行 Serverless 项目开发和代码调试，并且轻松将项目部署到云端。

通过该 VS Code 插件，您可以：

- 拉取云端的云函数列表，并触发云函数。
- 在本地快速创建云函数项目。
- 使用模拟的 COS、CMQ、CKafka、API 网关等触发器事件来触发函数运行。
- 上传函数代码到云端，更新函数配置。
- 在云端运行、调试函数代码。

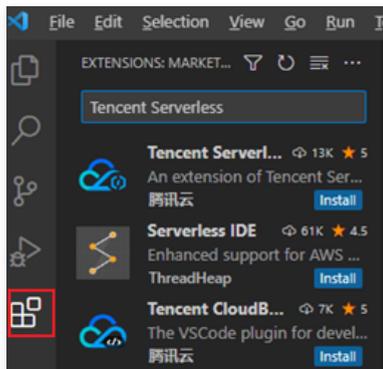
## 前提条件

Tencent Serverless Toolkit for VS Code 均可在 Windows, Linux 和 MacOS 中安装。在安装 Tencent Serverless Toolkit for VS Code 之前，需要确保系统中已有以下组件/信息：

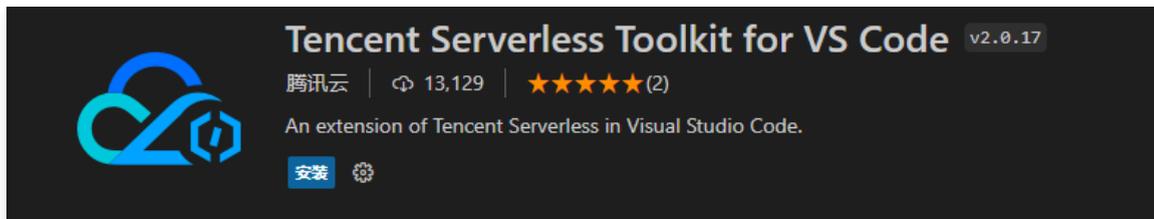
- VS Code：在 [VS Code 下载页面](#) 下载对应的 IDE 并安装，其版本要求为 v1.43.0+。
- 已注册腾讯云账户。若未注册腾讯云账户，可 [点此](#) 进入注册页面。

## 步骤1：通过 VS Code IDE 安装插件

1. 运行 VS Code IDE。
2. 打开 VS Code 插件市场。如下图所示：



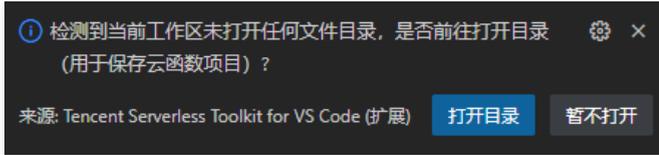
3. 在搜索框中输入“Tencent Serverless”，单击搜索框下方列表中的 Tencent Serverless Toolkit。查看详情并选择 **Install**。如下图所示：



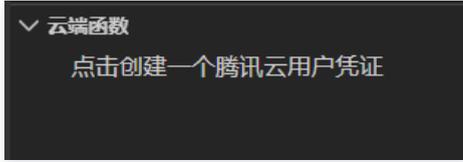
更多安装方法请参见 [安装说明](#)。

## 步骤2：配置插件

1. 单击左侧导航栏中的 ，打开已安装好的 Tencent Serverless 插件。
2. 选择文件目录，用于保存云函数项目。如下图所示：



3. 单击创建一个腾讯云用户凭证。如下图所示：



4. 根据提示依次输入账号的 APPID，SecretId 及 SecretKey 信息，作为插件调用云 API 时的认证信息。并在认证成功后，选择您希望部署函数的地域。配置信息获取途径如下：

- 账号的 APPID：通过访问控制台中的 [账号信息](#)，可以查看您的 APPID。
- SecretId 及 SecretKey：通过访问控制台中的 [API 密钥管理](#)，获取相关密钥或创建相关密钥。
- 地域：地域列表及对应的英文写法请参见 [支持地域](#)。

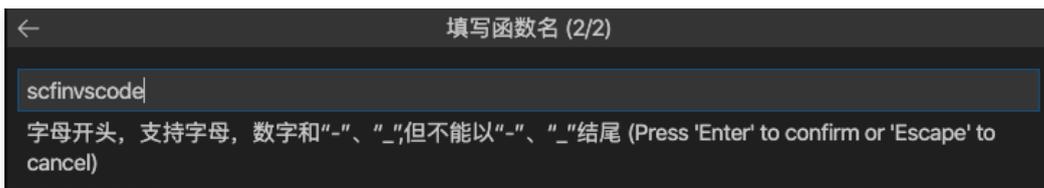
更多账号配置相关内容请参见 [账号配置](#)。

### 步骤3：创建函数

1. 在配置账户对应地域下的云端函数列表中，单击本地函数右侧的 ，在本地初始化新的函数项目。
2. 根据提示依次选择函数运行时 runtime，输入函数名。如下图所示：



本文 runtime 以 Python3.6，函数名以 `scfinvscode` 为例。



3. 函数信息录入成功后，将开始创建。  
创建成功后，列表中会展示新建的本地函数。如下图所示：

```

TENCENT-SCF
本地函数
scfinscode

Welcome index.py x
scfinscode > src > index.py
1 # -*- coding: utf8 -*-
2 def main_handler(event, context):
3     print(str(event))
4     return "hello world"

```

## 步骤4：部署函数

### 说明：

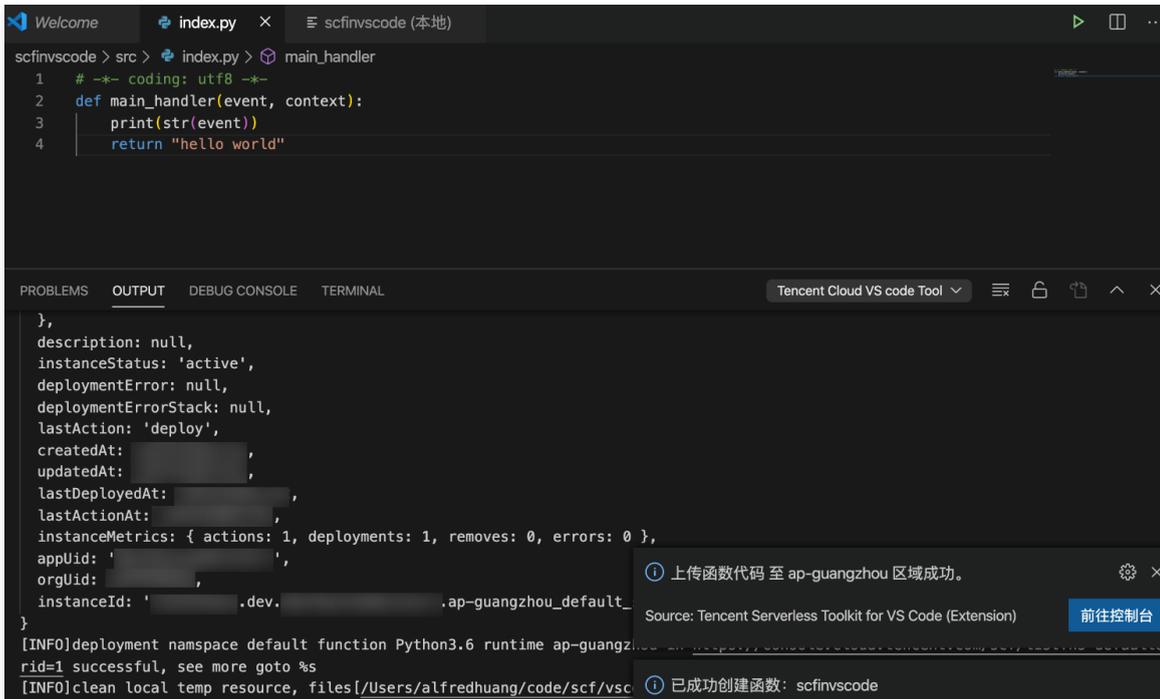
完成函数代码的编辑后步骤后，可通过插件将函数一键部署到云端。

1. 单击左侧列表中的函数名称，可打开该函数基本信息页面，查看函数信息。如下图所示：



2. 单击列表中函数名称右侧的 ，等待函数上传完毕。

在此过程中，可以通过 OUTPUT 输出口，看到函数的部署过程中的相关信息。如下图所示：



3. 部署完成后，您可以云端函数列表中查看函数的相关信息。

至此您已经完成了插件安装配置和函数创建及部署。您可以根据需要，进一步使用其他功能，例如云端调用、下载函数至本地、云端调试等。

# 安装说明

最近更新时间：2022-10-19 16:21:12

## 操作场景

本文介绍 [通过插件市场直接安装](#) 及 [通过 VS Code IDE 安装](#) 两种方式安装 Tencent Serverless Toolkit for VS Code，您可选择其中一种方式进行安装操作。

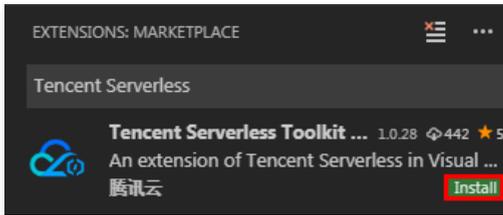
## 操作步骤

### 通过插件市场直接安装

进入 [插件市场](#)，单击Install进行安装。

### 通过 VS Code IDE 安装

1. 运行 VS Code IDE。
2. 打开 VS Code 插件市场。
3. 在搜索框中输入“Tencent Serverless”，单击搜索框下方列表中的 Tencent Serverless Toolkit。查看详情并选择Install。如下图所示：



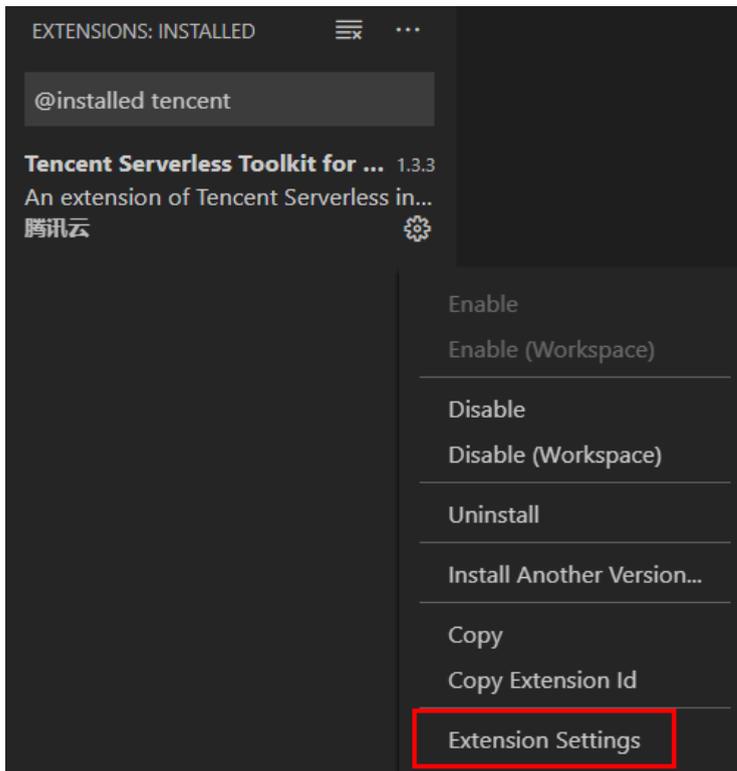
## 安装完成

安装完成后，左侧栏中会展示已安装完毕的 Tencent Serverless Toolkit。可以通过单击左侧 ，切换到插件提供的页面中开始使用。

## 相关操作

### 插件配置

可在插件管理页中，选择插件右侧的  > **Extension Settings** 进入插件配置页面。如下图所示：



### 设置开启调试输出

在插件的配置页面选择勾选 **Enable display interface call log**，可以启动接口的 log 输出，此配置项默认开启。

开启此配置项后，在部署函数或调用函数时，VS Code 的 Output 窗口将会默认输出对应接口的调用信息，包括部署输出、调用日志、返回内容等相关信息。

# 账号配置

最近更新时间：2025-06-13 15:09:02

## 操作场景

配置账号信息，是插件可以连接腾讯云并进行相关操作的必要条件。通过插件上的账号配置和管理能力，可便捷的将账号信息记录到本地，并在后续的操作中按需使用。

## 操作步骤

### 新增账号配置

1. 单击左侧导航栏中的 ，打开已安装好的 Tencent Serverless Toolkit。
2. 单击创建一个腾讯云用户凭证。如下图所示：



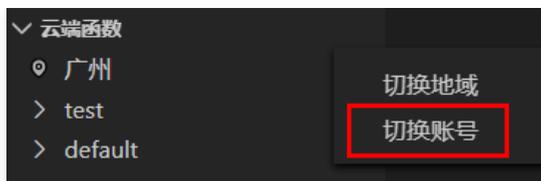
3. 根据提示依次输入账号的 APPID，SecretId 及 SecretKey 信息，作为插件调用云 API 时的认证信息。并在认证成功后，选择您希望部署函数的地域。配置信息获取途径如下：
  - 账号的 APPID：通过访问控制台中的 [账号信息](#)，可以查看您的 APPID。
  - SecretId 及 SecretKey：通过访问控制台中的 [API 密钥管理](#)，获取相关密钥或创建相关密钥。
  - 地域：地域列表及对应的英文写法请参见 [支持地域](#)。

### 选择账号配置

#### 说明：

您可在插件中配置多个账号，并在具备多个账号配置的情况下，选择切换当前使用的账号。

1. 单击左侧导航栏中的 ，打开已安装好的 Tencent Serverless Toolkit。
2. 单击云端函数右侧的 ，并在弹出菜单中选择切换账号。如下图所示：



3. 在窗口上方弹出的用户选择或新增选项上，可以查看当前正在使用的账号。如下图所示：



4. 可以通过选择期望使用的账号，切换到对应账号。或选择[登录新用户](#)并按照 [新增账号配置](#) 步骤增加新账号。

## 管理账号配置

通过插件配置的账号，目前均存储在用户目录下的 `.tcli_config.ini` 文件中。如果需要修改已配置账号，或快速编辑、添加账号，您可以通过文本编辑器打开此文件后直接修改。`.tcli_config.ini` 文件中存储的账号信息如下所示：

```
[USER_1]
region = ap-guangzhou
secret-key = *****
secret-id = *****
using-cos = false
appid = 123456789

[USER_2]
region = ap-guangzhou
secret-key = *****
secret-id = *****
using-cos = false
appid = 123456789

[OTHERS]
allow-report = true
version-time = 2020-06-04
no-color = true
language = none
curr-user = USER_1
```

# 函数操作

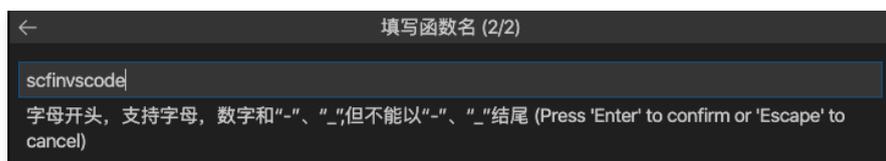
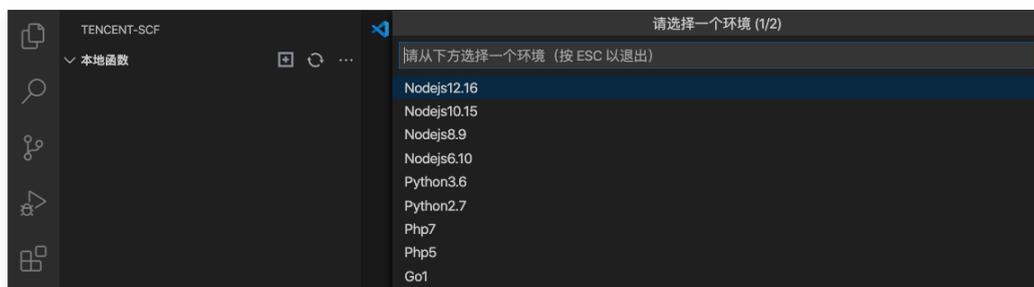
最近更新时间：2025-01-03 14:36:52

本文介绍如何通过插件进行本地及云端的函数操作，包含本地新建、部署函数、云端调用、下载函数至本地及测试函数。

## 本地函数操作

### 本地新建函数

1. 单击左侧导航栏中的 ，打开已安装好的 Tencent Serverless Toolkit。
2. 选择“本地函数”右侧的  创建函数，并选择所需使用的 runtime，输入函数名。如下图所示：



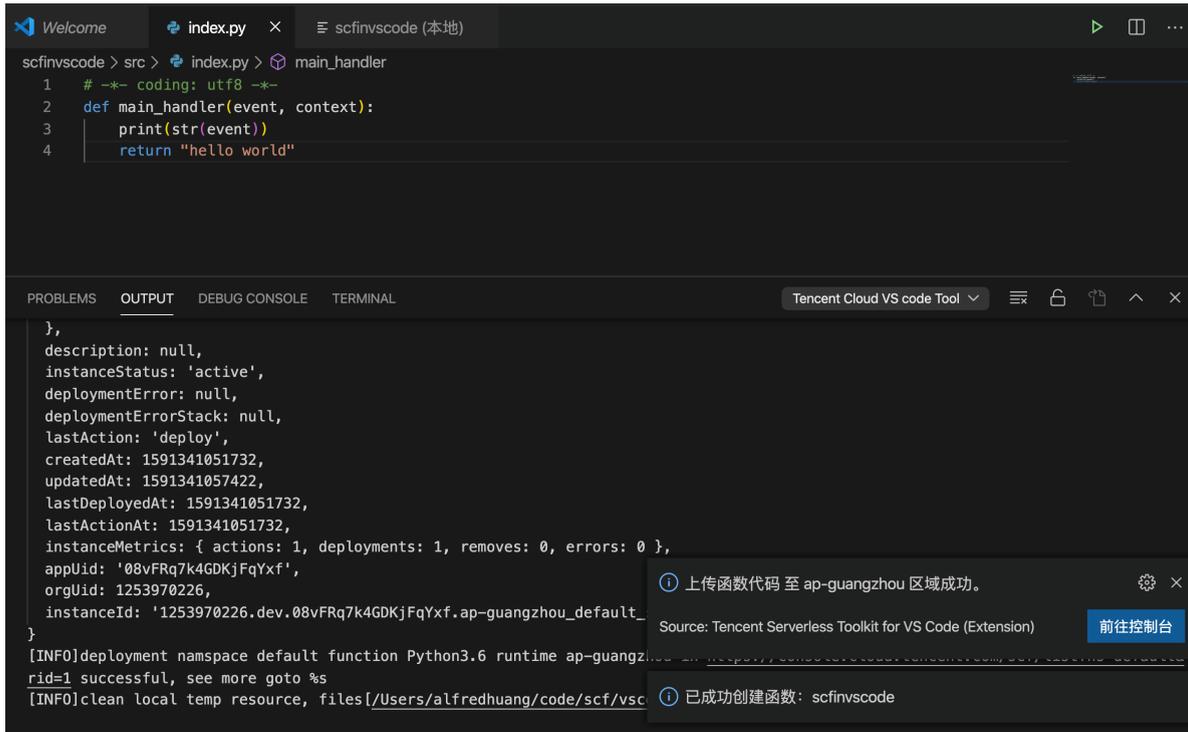
3. 创建完成后，“本地函数”列表中显示已有的函数。同时在当前的工作目录下，会创建以函数名命名的文件夹项目。如下图所示：



### 本地部署函数

单击“本地函数”列表中函数名称右侧的 ，可将本地函数部署至云端。

部署过程中，可以通过 OUTPUT 输出口，看到函数的部署过程中的相关信息。如下图所示：



```
scfinscode > src > index.py > main_handler
1  # -*- coding: utf8 -*-
2  def main_handler(event, context):
3      print(str(event))
4      return "hello world"
```

```
},
description: null,
instanceStatus: 'active',
deploymentError: null,
deploymentErrorStack: null,
lastAction: 'deploy',
createdAt: 1591341051732,
updatedAt: 1591341057422,
lastDeployedAt: 1591341051732,
lastActionAt: 1591341051732,
instanceMetrics: { actions: 1, deployments: 1, removes: 0, errors: 0 },
appUid: '08vFRq7k4GDKjFqYxf',
orgUid: 1253970226,
instanceId: '1253970226.dev.08vFRq7k4GDKjFqYxf.ap-guangzhou_default_...'
}
[INFO]deployment namespace default function Python3.6 runtime ap-guangzhou...
rid=1 successful, see more goto %s
[INFO]clean local temp resource, files [/Users/alfredhuang/code/scf/vsc...
```

上传函数代码 至 ap-guangzhou 区域成功。  
Source: Tencent Serverless Toolkit for VS Code (Extension) [前往控制台](#)

已成功创建函数: scfinscode

在完成部署后，可在“云端函数”列表中查看到已上传部署的函数。

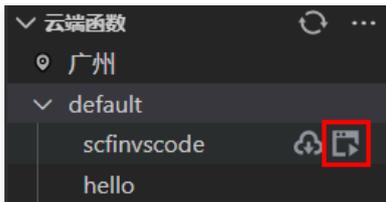
## 云端函数操作

### 云端调用

#### 说明：

针对云端的函数，可以通过 VS Code 插件进行云端的触发，更改事件模板等操作。

1. 单击“云端函数”列表中函数名右侧的 ，发起对云端函数的调用。如下图所示：



2. 在启用云端调用后，可在弹出菜单中选择需调用的触发事件。

#### 注意：

API 网关产品即将退市并安排下线，建议使用其他触发器。

3. 发起调用后，VS Code Output 中将输出调用内容，包含了函数返回、日志、执行情况等相关信息。如下图所示：

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL Tencent Cloud VS code Tool
Invoke function scfinvscode [RequestId:7d390911-a50b-48fc-9f0a-5c0837090115]:

Response:
"hello world"

Output:
START RequestId: 7d390911-a50b-48fc-9f0a-5c0837090115
Event RequestId: 7d390911-a50b-48fc-9f0a-5c0837090115
{'requestContext': {'serviceId': 'service-f94sy04v', 'path': '/test/{path}', 'httpMethod': 'POST', 'requestId':
'c6af9ac6-7b61-11e6-9a41-93e8deadbeef', 'identity': {'secretId': 'abcdxxxxxxxxsdfs'}, 'sourceIp': '127.0.0.1', 'stage': 'release'}, 'headers':
{'Accept-Language': 'en-US,en,cn', 'Accept': 'text/html,application/xml,application/json', 'Host': 'service-3ei3tii4-251000691.ap-guangzhou.
apigateway.myqcloud.com', 'User-Agent': 'User Agent String'}, 'body': '{"msg": "hello apigateway"}', 'pathParameters': {'path': 'value'},
'queryStringParameters': {'foo': 'bar'}, 'headerParameters': {'Refer': '127.0.0.1'}, 'stageVariables': {'stage': 'release'}, 'path': '/test/
value', 'queryString': {'foo': 'bar', 'bob': 'alice'}, 'httpMethod': 'POST'}

END RequestId: 7d390911-a50b-48fc-9f0a-5c0837090115
Report RequestId: 7d390911-a50b-48fc-9f0a-5c0837090115 Duration:1ms Memory:128MB MemUsage:11.2266MB

Summary:
Request ID: 7d390911-a50b-48fc-9f0a-5c0837090115
Run Duration: 1 ms
Bill Duration: 1 ms
Usage Memory: 11.2265625 MB
Invoke Result: 0

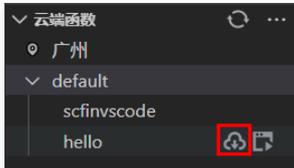
Invoke function scfinvscode success
```

## 导入本地

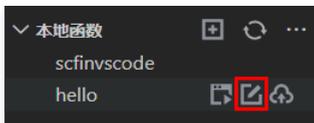
### 说明:

如果您已在 [腾讯云函数控制台](#) 创建了函数，则可以在 VS Code 插件里直接将云端函数导入到本地。

1. 单击“云端函数”列表中函数名右侧的 ，将目标函数导入到本地。如下图所示：



2. 在弹出窗口中选择函数需保存的目录，单击 **下载**。下载完成后会自动打开代码工作区。
3. 在插件页，您也可以选择目标函数右侧的 ，即可打开函数代码编辑视图。如下图所示：

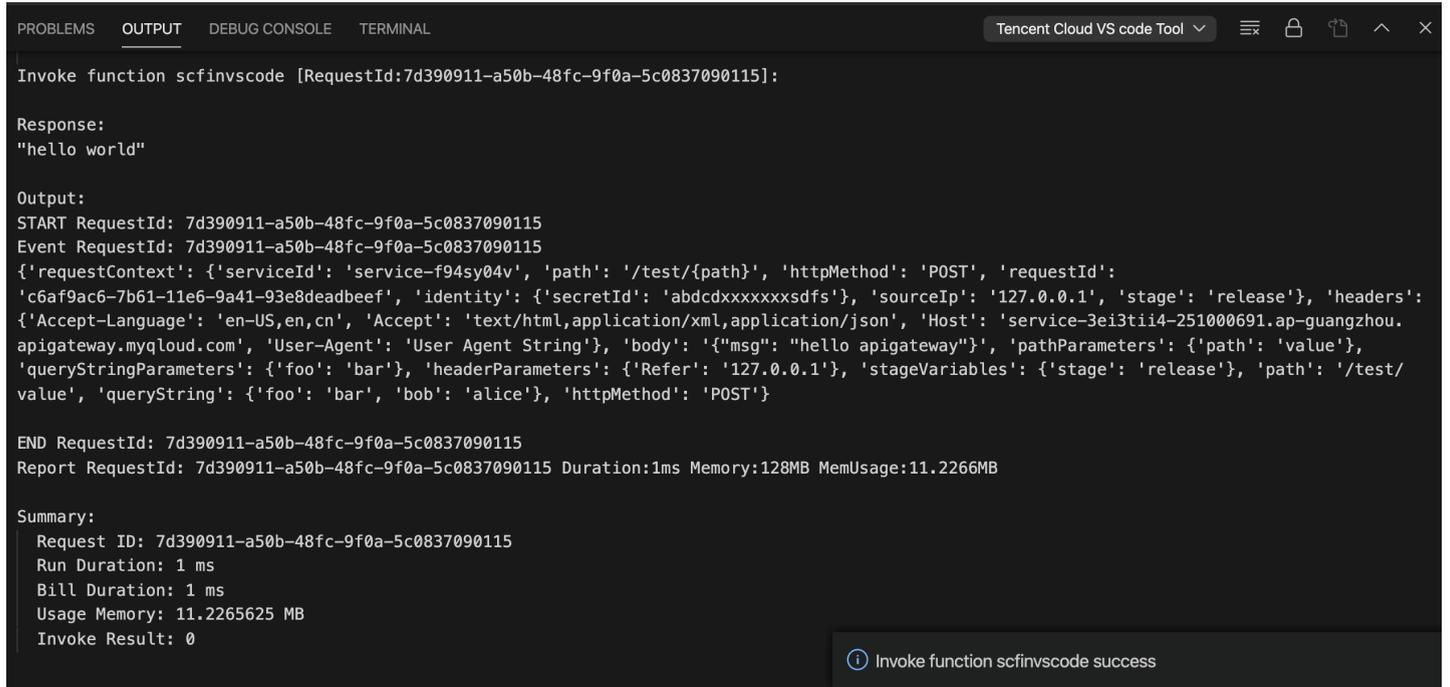


## 测试模板

调用时，可根据函数功能选择不同的测试模板，SCF VS Code 插件中暂不支持自定义模板。更多测试模板相关内容，详情请参见 [触发器](#)。

## 查看日志

云端调用的日志会输出到 VS Code。如下图所示：



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL Tencent Cloud VS code Tool
Invoke function scfinvscodetool [RequestId:7d390911-a50b-48fc-9f0a-5c0837090115]:

Response:
"hello world"

Output:
START RequestId: 7d390911-a50b-48fc-9f0a-5c0837090115
Event RequestId: 7d390911-a50b-48fc-9f0a-5c0837090115
{'requestContext': {'serviceId': 'service-f94sy04v', 'path': '/test/{path}', 'httpMethod': 'POST', 'requestId':
'c6af9ac6-7b61-11e6-9a41-93e8deadbeef', 'identity': {'secretId': 'abdcxxxxxxxxsdfs'}, 'sourceIp': '127.0.0.1', 'stage': 'release'}, 'headers':
{'Accept-Language': 'en-US,en,cn', 'Accept': 'text/html,application/xml,application/json', 'Host': 'service-3ei3tii4-251000691.ap-guangzhou.
apigateway.myqcloud.com', 'User-Agent': 'User Agent String'}, 'body': '{"msg": "hello apigateway"}', 'pathParameters': {'path': 'value'},
'queryStringParameters': {'foo': 'bar'}, 'headerParameters': {'Refer': '127.0.0.1'}, 'stageVariables': {'stage': 'release'}, 'path': '/test/
value', 'queryString': {'foo': 'bar', 'bob': 'alice'}, 'httpMethod': 'POST'}

END RequestId: 7d390911-a50b-48fc-9f0a-5c0837090115
Report RequestId: 7d390911-a50b-48fc-9f0a-5c0837090115 Duration:1ms Memory:128MB MemUsage:11.2266MB

Summary:
Request ID: 7d390911-a50b-48fc-9f0a-5c0837090115
Run Duration: 1 ms
Bill Duration: 1 ms
Usage Memory: 11.2265625 MB
Invoke Result: 0

Invoke function scfinvscodetool success
```

您也可以前往控制台打开函数页面选择运行日志，查看所有历史日志，详情请参见 [函数日志](#)。

# 云端调试

最近更新时间：2024-11-26 18:15:43

## 操作场景

VS Code 插件支持 Node.js 10.15 及以上版本的 Node.js 运行环境的函数进行在线调试。本文介绍通过在线调试功能，实现在本地的开发环境中查看远端的代码执行过程，并进行断点设置、单步跟踪及查看堆栈等能力。

## 前提条件

目前支持运行环境为 Node.js 10.15 的函数开启调试能力。

## 使用须知及注意事项

当前云函数的云端调试能力处于 Beta 阶段，欢迎试用并向我们反馈试用过程中遇到的问题或建议。

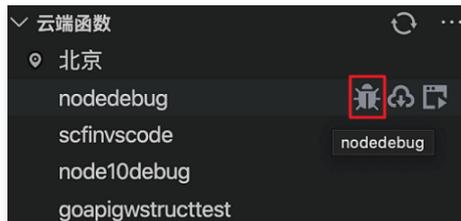
在使用云函数的远程调试功能时，需要了解如下信息及注意点：

1. 远程调试使用了云函数的一个实际运行的并发实例来进行调试。
2. 由于触发事件的随机性，如果有多个实例存在的情况下，触发事件可能随机的落到某个实例上，因此不是任意请求均能命中调试并发实例并可以开始调试。
3. 调试断点暂停运行时：
  - 长时间未运行且未返回的情况下，可能会导致触发端报错，例如 API 网关提示超时。
  - 实例仍然处于计时状态，并会在此次调试完成时，继续执行直到函数执行完成。整个过程记录耗费的总时长作为此次函数的运行时长。
4. 从触发并发实例运行，到最终完成调试，单次执行完成的最长时间为900秒。即在调试时如果中断执行900秒后，将会强制终止此次执行，按函数运行时长900秒并超时进行统计和计量。
5. 当前版本的调试能力，会使得云函数超时配置为900秒，在正常退出调试时将会重新设置超时为正常值。如果调试命令异常退出或强行终止，会导致云函数超时未能设置为正常值，此时可以通过再次部署（命令行）或手工编辑（控制台）的方式修改云函数的超时配置。详情请参见 [创建函数](#) 及 [更新函数](#)。

## 操作步骤

### 开启远程调试

1. 单击左侧导航栏中的 , 打开已安装好的 Tencent Serverless Toolkit。
2. 选择“云端函数”列表中需远程调试函数右侧的 , 开启当前函数的远程调试。如下图所示：



开启远程调试后，VS Code 将切换进入 debugger 模式，并将自动依据以下调试启动配置来启动调试：

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "SCF Remote Debugger for Node10+ - node10debug",
      "type": "node",
      "request": "attach",
      "port": 9222,
      "localRoot": "${workspaceFolder}/node10debug/src",
      "remoteRoot": "/var/user"
    }
  ]
}
```

**说明:**

- `node10debug` 为当前示例的函数名。
- `localRoot` 将使用当前启动调试函数的具体代码目录，用于映射远端函数代码和本地函数代码。

**进行远程调试**

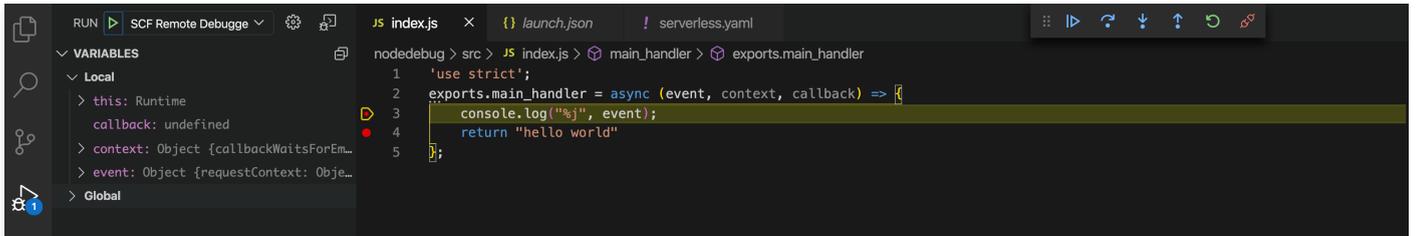
1. 远程调试启动后，在本地对应的代码上增加断点，用于中断函数运行并进行后续的调试操作。

2. 触发函数运行，可通过以下方式：

- 选择“云端函数”列表右侧的 ，向云端函数发送触发事件。
- 通过其他方式，例如带有 API 网关触发器的函数可通过访问 URL 触发函数运行。

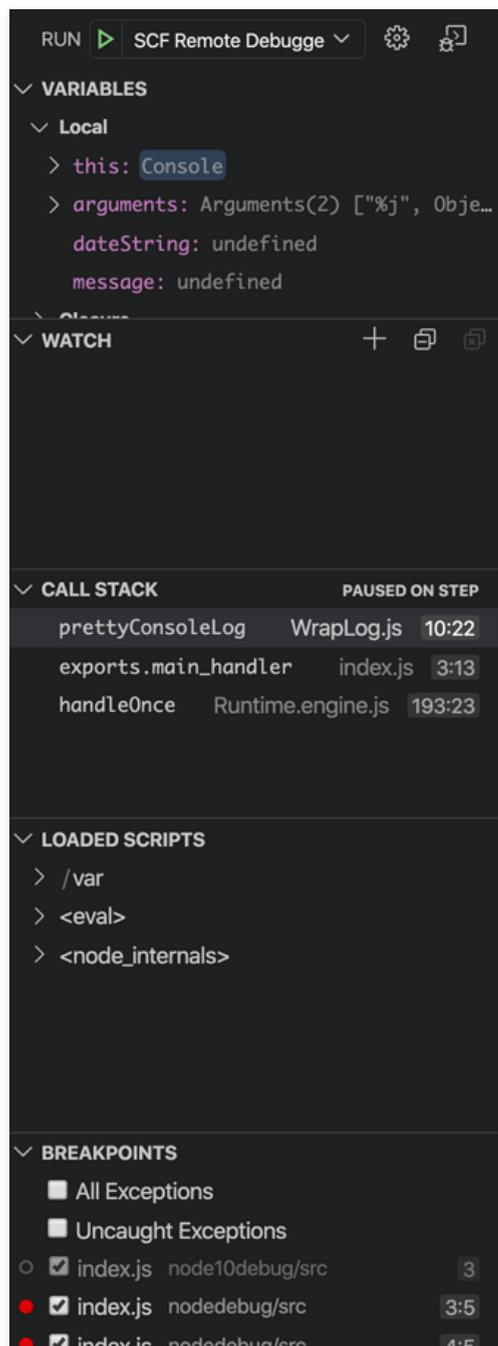
3. 在触发运行到断点后，本地调试窗口将显示目前中断位置。如下图所示：

通过调试的控制按钮，可以进行调试过程的 Continue、Step Over、Step Out、Step Into 等操作。



您还可以通过调试窗口的其他能力，进行查看当前的变量具体值、增加自身期望监控的变量及当前的调用堆栈等操作。例如，可通过 Loaded Scripts 窗

口，直接开启远端代码查看及设置断点。如下图所示：



## 关闭远程调试

在关闭 VSCode debugger 时，将会自动关闭远程调试功能。

# 插件更新日志

最近更新时间：2024-11-29 17:29:03

## v2.0.0 版本

发布时间：2020-06-12

本次发布包含了以下内容：

### 配置文件规范调整：

- 调整插件使用的规范为 Serverless Cloud Framework 中 tencent-scf component 规范。
- 支持原有 TCSAM 规范兼容转换为 Serverless Cloud Framework tencent-scf component 规范。  
详情请参见 [2.0版本升级说明](#) 中的配置规范调整说明。

### 新增 runtime 支持：

增加支持 Node.js 10.15 及 12.16 运行环境。

### 新增云端调试支持：

支持 Node.js 10 及以上运行环境进行云端调试。

## v1.3.1 版本

发布时间：2019-11-12

本次发布包含了以下内容：

### 体验优化：

- 下载云端函数至本地时，支持自定义下载目录。
- Timer 触发器字段 template.yaml 支持 `message` 字段。

### 问题修复：

- 本地修改函数部分配置，上传后未更新。
- 其他已知问题修复。

## v1.2.6 版本

发布时间：2019-10-28

本次发布包含了以下内容：

### 体验优化：

- 新增问题反馈入口，用户可直接扫码进群交流。
- 部分操作提示优化。

### 问题修复：

- Win 10 操作系统下，函数路径配置无法识别。
- 其他已知问题修复。

## v1.2.5 版本

发布时间：2019-10-11

本次发布包含了以下内容：

### 新增功能：

- 将插件和 CLI 解耦，可以独立使用插件，无需再安装 CLI。
- 将开启 COS 上传，Python 路径配置放到 VS Code 设置面板中。
- 初始化的函数模板默认添加 SCF 和联动资源角色权限。

### 体验优化：

- 函数目录名和 template.yaml 里的函数名解除强关联。
- 其他已知问题修复。

## v1.2.4 版本

发布时间：2019-09-06

本次发布包含了以下内容：

### 新增功能：

- 上传函数集成至工作区右键，无需切换至插件页上传。
- 创建函数流程优化，提供自动检测。
- 支持定义忽略上传内容，使用规则忽略上传说明。

**体验优化：**

- 云端本地函数区分展示。
- 导入本地/编辑代码/本地调用功能放至函数列表一级视图。
- 其他已知问题修复。

## v1.1.14 版本

发布时间：2019-08-15

本次发布包含了以下内容：

**新增功能：**

函数相关操作（创建，调试，上传）时，插件自动检测 SCF CLI 的安装状况。

**问题修复：**

- 云端函数同步至本地时，无权限等异常提醒完善。
- 插件自动安装 CLI 的错误信息输出。
- 云端函数同步至本地后默认打开工作区。
- 其他体验调优。

## v1.1.11 版本

发布时间：2019-08-09

本次发布包含了以下内容：

**新增功能：**

- 一键调试功能
- 新增、部署、导入等操作反馈。
- 日志输出到 output。
- 创建函数，同步云端函数后自动打开工作区。
- 插件自动检测并安装 SCF CLI。

**问题修复：**

- 部分已知问题修复。
- 其他体验调优。

## v1.1.2 版本

发布时间：2019-07-24

本次发布包含了以下内容：

**新增功能：**

- 适配 SCF CLI v0.1.0+。
- 自动检测 SCF CLI 版本并提示更新。
- 新建模板时自动检测命名规范。

**问题修复：**

- 新建函数或模板列表不自动更新问题。
- 其他体验调优。

## v1.0.28 版本

发布时间：2019-07-12

本次发布包含了以下内容：

**问题修复：**

新用户安装时默认账号信息配置获取问题。

## v1.0.27 版本

发布时间：2019-07-11

本次发布包含了以下内容：

**新增功能：**

- 支持云端函数导入本地时，自动加载 `template.yaml` ，可右键直接跳转编辑代码。
- 优化云端、本地、云端&本地状态展示。
- 其他体验优化。

**问题修复：**

- Python 函数本地调试异常。
- 不同命名空间下创建的函数，均归属于 `default namespace` 。
- 其他已知问题修复。

## 2.0版本升级说明

最近更新时间：2022-10-18 15:46:17

### 功能变更概述

2.0版本包含了若干项功能调整和新增，如下：

- 配置规范调整为使用 Serverless Cloud Framework 中 `tencent-scf component` 规范，详细规范内容请参见 [tencent-scf component 配置说明文档](#)。
- Runtime 支持最近新上线的 Node.js10.15 及 Node.js12.16。
- 支持 Node.js10 及以上版本 runtime 的云函数进行云端调试。

其中配置规范的调整，可能在一定程度上会对原项目造成影响，相关的调整方式、兼容措施、升级方案等说明如下。

### 配置规范调整

配置规范是针对本地项目中用于描述函数相关配置的 yaml 文件的格式规范。在 `tencent scf vscode` 插件的 1.x 版本中，使用了腾讯云自定义的 TCSAM 规范，详情请参见 [TCSAM 规范说明](#)。

为了进一步提升 `tencent scf vscode` 插件的适用范围、产品联动能力以及后续的扩展能力。2.0 版本中切换并使用了 Serverless Cloud Framework 中的配置规范，目前兼容的是针对 `tencent-scf component` 的配置规范，详情请参见 [tencent-scf component 规范说明文档](#)。

### 原有本地项目升级转换

为了便于原有本地项目中保留的配置规范文件 `template.yaml` 可以迅速转换到新版本的配置规范文件 `serverless.yaml`，新版本插件中增加了自动的转换能力。

在本地项目部署到云端时，插件会自动进行检测，并在缺失 `serverless.yaml` 文件的情况下，自动生成并填充相关内容，具体内容来自原有的 `template.yaml` 文件中的内容。生成的 `serverless.yaml` 格式如下所示：

```
component: scf
name: ap-guangzhou_alfredtest2_testvscodec1 ## 生成的项目部署名，由原配置文件中的地域、命名空间、函数名构成
org: '1253970226' ## 部署组织或部署人，由用户 id 组成
app: testvscodec1 ## 使用函数名作为应用名
stage: dev ## 默认设置为 dev
inputs:
  name: testvscodec1 ## 函数名
  src: ./
  description: This is a template function
  handler: index.main_handler
  runtime: Python3.6
namespace: alfredtest2 ## 支持命名空间，不指定、不填写的情况下默认为 default
region: ap-guangzhou ## 指定部署地域
memorySize: 128
timeout: 3
environment:
  variables:
    ENV_FIRST: env1
    ENV_SECOND: env2
```

#### ⚠ 注意

完成升级后，原有的 `template.yaml` 文件将不再被继续使用，在本地项目中存在 `serverless.yaml` 的情况下，后续的部署过程均以 `serverless.yaml` 文件内容为准。请注意在调整配置时，需使用 `serverless.yaml` 文件。

### 云端下载项目升级转换

从云端下载函数到本地进行编辑开发时，2.0 版本同样会协助进行项目的升级转换。

下载云端函数到本地时，插件将会检测下载的本地项目目录中是否存在新的配置规范文件 `serverless.yaml`。当本地项目中不存在 `serverless.yaml` 时，插件将会从云端下载 `serverless.yaml` 文件并放置在本地项目中，内容以云端函数的当前配置为准。同时，下载的项目将按如下的目录结构进行设计

并放置函数代码：

```
project
|- src
  |- index.js
  |- other.file
|- serverless.yaml
```

# 函数间调用 SDK

## Node.js SDK

最近更新时间：2022-12-20 15:29:17

### Tencentcloud-Serverless-Nodejs SDK 简介

Tencentcloud-Serverless-Nodejs 是腾讯云云函数 SDK，集成云函数业务流接口，简化云函数的调用方法。在使用该 SDK 的情况下，用户可以方便的从本地、云服务器（CVM）、容器、以及云端函数里快速调用某一个云函数，无需再进行公有云 API 的接口封装。

### 功能特性

Tencentcloud-Serverless-Nodejs SDK 的功能特性可分为以下几点：

- 高性能，低时延的进行函数调用。
- 填写必须的参数后，即可快速进行函数间的调用（SDK 会默认获取环境变量中的参数，例如 region，secretId 等）。
- 支持内网域名的访问。
- 支持 keepalive 能力。
- 支持跨地域函数调用。

#### 说明

函数间调用 SDK 仅适用于事件函数，Web 函数可通过在函数代码中请求 Web 函数对应路径的方式发起调用。

### 快速开始

#### 开发准备

- 开发环境  
已安装 Node.js 8.9 及以上版本。
- 运行环境  
已安装 tencentcloud-serverless-nodejs SDK，支持 Windows、Linux 和 Mac 操作系统。
- 建议使用 [Serverless Cloud Framework](#)，可快速部署本地云函数。

### 安装 tencentcloud-serverless-nodejs SDK

#### 通过 npm 安装（推荐）

1. 根据实际需求，选择目录路径，并在该路径下创建新的目录。  
例如，创建一个名称为 `testNodejsSDK` 的项目目录，项目路径为 `/Users/xxx/Desktop/testNodejsSDK`。
2. 进入 `testNodejsSDK` 目录，并依次执行以下命令，安装 `tencentcloud-serverless-nodejs` SDK。

```
npm init -y
npm install tencentcloud-serverless-nodejs
```

安装完成后，在 `testNodejsSDK` 目录下可以查看到 `node_modules`，`package.json` 和 `package-lock.json`。

#### 通过源码包安装

前往 [Github 代码托管地址](#) 下载最新源码包，解压源码包后进行安装。

#### 使用云函数在线依赖安装

使用 [云函数在线依赖安装](#)，在 `package.json` 中执行以下命令并进行安装。

```
{
  "dependencies": {
    "tencentcloud-serverless-nodejs": "*"
  }
}
```

}

## 云端函数互调

### 示例

#### ⚠ 注意

- 不同地域下的函数互调，须指定地域，命名规则参见 [地域列表](#)。
- 如果不指定地域，默认为同地域下函数互调。
- 命名空间不指定，默认为 default。
- 需要打开调用方函数外网访问权限。
- 如果没有手动传入 secretId 和 secretKey 等参数，函数需绑定有 SCF Invoke 权限（或者包含 SCF Invoke，例如 SCF FullAccess）的角色，可参考 [创建函数运行角色](#)。

1. 创建一个地域为北京，名称为“FuncInvoked”，并用于被调用的 Node.js 云函数。该云函数内容如下：

```
'use strict';
exports.main_handler = async (event, context, callback) => {
  console.log("\n Hello World from the function being invoked\n")
  console.log(event)
  console.log(event["non-exist"])
  return event
};
```

2. 在 testNodejsSDK 目录下新建文件 index.js，并输入如下示例代码，创建发起调用的 Node.js 云函数。

```
const { SDK, LogType } = require('tencentcloud-serverless-nodejs')
exports.main_handler = async (event, context) => {
  context.callbackWaitsForEmptyEventLoop = false
  const sdk = new SDK({
    region: 'ap-beijing'
  }) //如果在云函数中运行并且绑定了有SCF调用资格的运行角色，会默认取环境变量中的鉴权信息
  const res = await sdk.invoke({
    functionName: 'FuncInvoked',
    logType: LogType.Tail,
    data: {
      name: 'test',
      role: 'test_role'
    }
  })
  console.log(res)
  // return res
}
```

其中主要参数获取途径如下：

- **region**: 被调用云函数所在地域，本文以 [步骤1](#) 中的北京地域为例。
  - **functionName**: 被调用云函数名称，本文以 [步骤1](#) 中已创建的 FuncInvoked 函数为例。
  - **qualifier**: 被调用云函数版本，如未指定则默认使用 `$LATEST`。详情请参见 [查看版本](#)。
  - **namespace**: 被调用云函数所在命名空间，如未指定则默认 `default`。详情请参见 [命名空间管理](#)。
  - **data**: 传递给被调用云函数的数据，被调用的云函数可以从 event 入参中读取此数据。
3. 创建一个地域为成都，名称为“NodejsInvokeTest”，并用于调用的 Node.js 云函数。该云函数主要设置信息如下：
- 执行方法：选择 `index.main_handler`。

- 代码提交方式：选择本地上传 zip 包。

将 `testNodejsSDK` 目录下的所有文件压缩为 zip 格式，并上传到云端。

4. 在 Serverless 控制台 单击新建的云函数，进入函数管理的代码编辑页面，然后单击测试，运行函数。输出结果如下：

```
"Already invoked a function!"
```

## 本地调用云端函数

### 示例

1. 创建一个地域为北京，名称为 “FuncInvoked”，并且用于被调用的 Node.js 云函数。该云函数内容如下：

```
'use strict';
exports.main_handler = async (event, context, callback) => {
  console.log("\n Hello World from the function being invoked\n")
  console.log(event)
  console.log(event["non-exist"])
  return event
};
```

2. 在 `testNodejsSDK` 目录下新建文件 `index.js`，作为发起调用的 Node.js 云函数，并输入如下示例代码：

```
const { SDK, LogType } = require('tencentcloud-serverless-nodejs')
exports.main_handler = async (event, context) => {
  context.callbackWaitsForEmptyEventLoop = false
  const sdk = new SDK({
    region: 'ap-beijing',
    secretId: 'AKXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXj',
    secretKey: 'WtXXXXXXXXXXXXXXXXXXXXXXXXXXXXqL'
  }) //如果在云函数中运行并且绑定了有SCF调用资格的运行角色，会默认取环境变量中的鉴权信息
  const res = await sdk.invoke({
    functionName: 'FuncInvoked',
    logType: LogType.Tail,
    data: {
      name: 'test',
      role: 'test_role'
    }
  })
  console.log(res)
  // return res
}
```

#### ⚠ 注意

`secretId` 及 `secretKey`：指云 API 的密钥 ID 和密钥 Key。您可以通过登录 [访问管理控制台](#)，选择访问密钥 > API 密钥管理，获取相关密钥或创建相关密钥。

3. 进入 `index.js` 所在文件目录，执行以下命令，查看结果。

- Linux 及 Mac 操作系统，执行以下命令：

```
export NODE_ENV=development && node index.js
```

- Windows 操作系统执行以下命令：

```
set NODE_ENV=development && node index.js
```

输出结果如下:

```
prepare to invoke a function!
{"key":"value"}
Already invoked a function!
```

## 接口列表

### API Reference

- [Init](#)
- [Invoke](#)

#### Init

在使用 SDK 前, 建议执行 `npm init` 命令进行初始化 SDK。

##### ! 说明

- 初始化命令可传入 `region`, `secretId`, `secretKey` 参数。
- 完成初始化后, 调用 API 接口时可复用初始化的配置。

#### 参数信息:

参数名	是否必填	类型	描述
region	否	String	地域
secretId	否	String	默认会取 process.env.TENCENTCLOUD_SECRETID
secretKey	否	String	默认会取 process.env.TENCENTCLOUD_SECRETKEY
token	否	String	默认会取 process.env.TENCENTCLOUD_SESSIONTOKEN

#### Invoke

调用函数, 目前支持同步调用。

#### 参数信息:

参数名	是否必填	类型	描述
functionName	是	String	函数名称
qualifier	否	String	函数版本, 默认为 \$LATEST
data	否	String	函数运行入参
namespace	否	String	命名空间, 默认为 default
region	否	String	地域
secretId	否	String	默认会取 process.env.TENCENTCLOUD_SECRETID
secretKey	否	String	默认会取 process.env.TENCENTCLOUD_SECRETKEY
token	否	String	默认会取 process.env.TENCENTCLOUD_SESSIONTOKEN

# Python SDK

最近更新时间：2023-08-23 09:25:41

## Tencentserverless SDK 简介

Tencentserverless 是腾讯云云函数 SDK，集成云函数业务流接口，简化云函数的调用方法。在使用该 SDK 的情况下，用户可以方便的从本地、云服务器（CVM）、容器、以及云端函数里快速调用某一个云函数，无需再进行公有云 API 的接口封装。

## 功能特性

Tencentserverless SDK 的功能特性可分为以下几点：

- 高性能，低时延的进行函数调用。
- 填写必须的参数后，即可快速进行函数间的调用（SDK 会默认获取环境变量中的参数，例如 region，SecretId 等）。
- 支持内网域名的访问。
- 支持 keepalive 能力。
- 支持跨地域函数调用。
- 支持 Python 原生调用方式。

### 说明

函数间调用 SDK 仅适用于事件函数，Web 函数可通过在函数代码中请求 Web 函数对应路径的方式发起调用。

## 快速开始

### 云端函数互调

#### 示例

### 注意

- 不同地域下的函数互调，须指定地域，命名规则参见 [地域列表](#)。
- 如果不指定地域，默认为同地域下函数互调。
- 命名空间不指定，默认为 default。

1. 在云端创建一个被调用的 Python 云函数，地域为广州，命名为“FuncInvoked”。函数内容如下：

```
# -*- coding: utf8 -*-

def main_handler(event, context):
    if 'key1' in event.keys():
        print("value1 = " + event['key1'])
    if 'key2' in event.keys():
        print("value2 = " + event['key2'])
    return "Hello World from the function being invoked" #return
```

2. 在云端创建调用的 Python 云函数，地域为成都，命名为“PythonInvokeTest”。可通过以下两种方式，结合您的实际情况编辑 PythonInvokeTest 函数。

- 方式 1：如果您不需要频繁的调用函数，可使用如下示例代码：

```
from tencentserverless import scf
from tencentserverless.scf import Client
from tencentserverless.exception import TencentServerlessSDKException
from tencentcloud.common.exception.tencent_cloud_sdk_exception import TencentCloudSDKException

def main_handler(event, context):
```

```
print("prepare to invoke a function!")
try:
    data = scf.invoke('FuncInvoked', region="ap-guangzhou", data={"a": "b"})
    print (data)
except TencentServerlessSDKException as e:
    print (e)
except TencentCloudSDKException as e:
    print (e)
except Exception as e:
    print (e)
return "Already invoked a function!" # return
```

输出结果如下:

```
"Already invoked a function!"
```

- 方式 2: 如果您需要频繁调用函数, 可选择通过 Client 的方式连接并触发。可使用如下示例代码:

```
# -*- coding: utf8 -*-

from tencentserverless import scf
from tencentserverless.scf import Client
from tencentserverless.exception import TencentServerlessSDKException
from tencentcloud.common.exception.tencent_cloud_sdk_exception import TencentCloudSDKException

def main_handler(event, context):
    #scf = Client(region="ap-guangzhou") # 使用该方法进行 Client 连接, 请在函数配置中启用“运行角色”功能, 并选择具有调用函数权限的运行角色。
    scf =
Client(secret_id="AKIxxxxxxxxxxxxxxxxxxxxxxxxggB4Sa", secret_key="3vZzxxxxxxxxxxaeTC", region="ap-guangzhou", token="") # 使用该方法进行 Client 连接, 请将 secret_id 和 secret_key 替换为您的 secret_id 和 secret_key, 该组密钥需要包含调用函数的权限。
    print("prepare to invoke a function!")
    try:
        data = scf.invoke('FuncInvoked', data={"a": "b"})
        # data = scf.FuncInvoked(data={"a": "b"}) #使用Python原生调用方式, 需要首先通过Client进行初始化
        print (data)
    except TencentServerlessSDKException as e:
        print (e)
    except TencentCloudSDKException as e:
        print (e)
    except Exception as e:
        print (e)
    return "Already invoked a function!" # return
```

输出结果如下:

```
"Already invoked a function!"
```

#### ⚠ 注意

secret\_id 及 secret\_key: 指云 API 的密钥 ID 和密钥 Key。您可以通过登录 [访问管理控制台](#), 选择云 API 密钥 > API 密钥管理, 获取相关密钥或创建相关密钥。

## 本地调用云端函数

### 开发准备

- 开发环境  
已安装 Python2.7 或者 Python3.6。
- 运行环境  
已安装 tencentserverless SDK，支持 Windows、Linux 和 Mac 操作系统。

#### ! 说明

本地调用云端函数须进行以上开发准备，推荐函数在本地开发完成后上传到云端，使用云端函数互调进行调试。

### 通过 pip 安装（推荐）

执行以下命令，安装 tencentserverless Python SDK。

```
pip install tencentserverless
```

### 通过源码包安装

前往 Github 代码托管地址下载最新源码包，待源码包解压后依次执行以下命令进行安装。

```
cd tencent-serverless-python-master
python setup.py install
```

### 配置 tencentserverless Python SDK

执行以下命令，升级 tencentserverless Python SDK。

```
pip install tencentserverless -U
```

执行以下命令，查看 tencentserverless Python SDK 信息。

```
pip show tencentserverless
```

### 示例

1. 在云端创建一个被调用的 Python 云函数，地域为广州，命名为“FuncInvoked”。函数内容如下：

```
# -*- coding: utf8 -*-

def main_handler(event, context):
    if 'key1' in event.keys():
        print("value1 = " + event['key1'])
    if 'key2' in event.keys():
        print("value2 = " + event['key2'])
    return "Hello World from the function being invoked" #return
```

2. 创建完毕后，在本地创建一个名为 PythonInvokeTest.py 的文件。内容如下：

```
# -*- coding: utf8 -*-

from tencentserverless import scf
from tencentserverless.scf import Client
from tencentserverless.exception import TencentServerlessSDKException
from tencentcloud.common.exception.tencent_cloud_sdk_exception import TencentCloudSDKException
```

```
def main_handler(event, context):
    print("prepare to invoke a function!")
    scf =
Client(secret_id="AKIxxxxxxxxxxxxxxxxxxxxxxxxggB4Sa",secret_key="3vZxxxxxxxxxxxxaeTC",region="ap-
guangzhou",token="")# 替换为您的 secret_id 和 secret_key
    try:
        data = scf.invoke('FuncInvoked',data={"a":"b"})
        # data = scf.FuncInvoked(data={"a":"b"})
        print (data)
    except TencentServerlessSDKException as e:
        print (e)
    except TencentCloudSDKException as e:
        print (e)
    except Exception as e:
        print (e)
    return "Already invoked a function!" # return

main_handler("", "")
```

进入 PythonInvokeTest.py 所在文件目录，执行以下命令，查看结果。

```
python PythonInvokeTest.py
```

输出结果如下：

```
prepare to invoke a function!"Hello World form the function being invoked"
```

## 接口列表

### API Reference

- [Client](#) (类)
- [invoke](#) (方法)
- [TencentserverlessSDKException](#) (类)

#### Client

方法：

- [\\_\\_init\\_\\_](#)

参数信息：

参数名	是否必填	类型	描述
region	否	String	地域信息，默认与调用接口的函数所属地域相同，本地调用默认是广州。
secret_id	否	String	用户 SecretId，默认是从云函数环境变量中获取，本地调试必填。
secret_key	否	String	用户 SecretKey，默认是从云函数环境变量中获取，本地调试必填。
token	否	String	用户 token，默认是从云函数环境变量中获取

- [invoke](#)

参数信息：

参数名	是否必填	类型	描述
-----	------	----	----

function_name	是	String	函数名称。
qualifier	否	String	函数版本，默认为 \$LATEST。
data	否	Object	函数运行入参，必须可以被 json.dumps 的对象。
namespace	否	String	命名空间，默认为 default。

## invoke

调用函数，暂时只支持同步调用。

参数信息：

参数名	是否必填	类型	描述
region	否	String	地域信息，默认与调用接口的函数所属地域相同，本地调用默认是广州。
secret_id	否	String	用户 SecretId，默认是从云函数环境变量中获取，本地调试必填。
secret_key	否	String	用户 SecretKey，默认是从云函数环境变量中获取，本地调试必填。
token	否	String	用户 token，默认从云函数环境变量中获取。
function_name	是	String	函数名称。
qualifier	否	String	函数版本，默认为 \$LATEST。
data	否	String	函数运行入参，必须可以被 json.dumps 的对象。
namespace	否	String	命名空间，默认为 default。

## TencentserverlessSDKException

属性：

- [code]
- [message]
- [request\_id]
- [response]
- [stack\_trace]

方法及描述：

方法名	描述
get_code	返回错误码信息
get_message	返回错误信息
get_request_id	返回 RequestId 信息
get_response	返回 response 信息
get_stack_trace	返回 stack_trace 信息

# 第三方工具

## Malagu Framework

### 框架介绍

最近更新时间：2024-11-15 17:24:13

#### 说明：

Malagu 为第三方开发工具，暂时无法提供腾讯云官方支持，如有任何问题或反馈，欢迎访问 [Malagu 社区](#)，以 issue 方式进行讨论或社区共建。

## 什么是 Malagu?

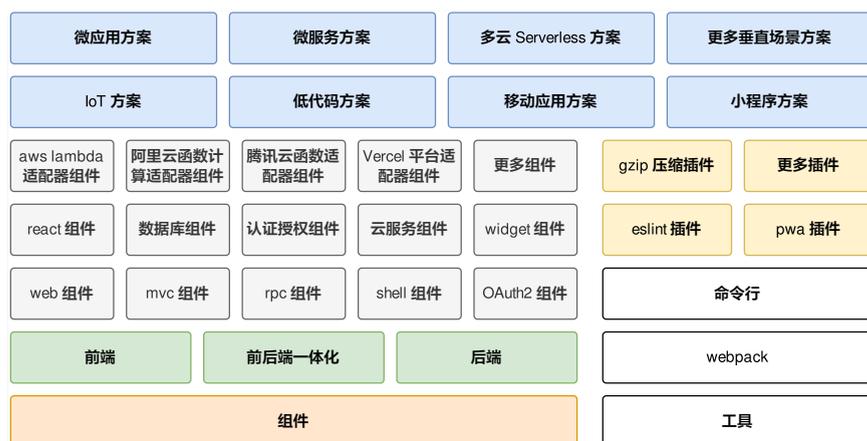
Malagu 是基于 TypeScript 的 Serverless First、组件化、平台无关的渐进式应用框架，又称为 M 框架。使用同一套编程语言和 IoC 设计，用于开发前端、后端和前后端一体化应用。并且结合了 OOP（面向对象编程）、AOP（面向切面编程）等元素，借鉴了很多 Spring Boot 设计思想。

在后端，Malagu 抽象一套接口，方便适配任意的平台和基础框架，是一个平台或基础框架无关的上层框架。平台有腾讯云云函数、AWS Lambda、Vercel 等，基础框架有 Express、Koa、Fastify 等。

在 Serverless 场景，Malagu 是以应用为单位开发项目，一个应用一般包含多个 API 接口。如果应用很大，应该将大应用拆解成一个个小的微应用或者微服务。如同微服务架构的粒度拆分原则一样，合理的粒度拆分，能够更好的管理应用。框架会保证一个应用在一个函数中的运行性能。

了解更多请参见 [Malagu 官方文档](#)。

## Malagu 架构图



## 为什么需要 Malagu

### 坚信 Serverless 是未来

Serverless 是云计算新一代计算引擎，为取代传统云服务器架构而生。Serverless 核心理念是让开发者无需关心服务器的存在，专注于业务代码。

### Serverless 现状

目前，所有云厂商和社区都在大力推广和布道 Serverless 理念，通过 Serverless 可以低成本高质量快速落地商业方案。业内普遍认为 Serverless = FaaS + BaaS，未来也可能是其他的形态，不管形态如何变化，Serverless 核心理念不变。Serverless 的开发体验关键在 FaaS 的开发体验，而 FaaS 目前开发体验不是很理想，存在着很多痛点。部分痛点短时间内在 FaaS 底层上可能难于解决，部分痛点可能在工具、框架层面去解决更合理。例如冷启动、CI/CD、微服务、数据库访问、本地开发调试运行、平台不锁定等。未来，会有越来越多的 Serverless First 的开发框架出现，而不仅仅是资源编排运维工具，结合 Serverless First 开发框架提供更高级的 Serverless 开发平台或者低代码平台。

#### 说明：

如何解决这些痛点？我们可以换一个思路，从开发框架层面尝试解决这些问题（事实证明，可以通过开发框架来解决这些问题）。那么，面临新的抉择是采用传统框架，还是需要一个全新的框架？如果选择做一个新框架，那么是选择特定编程语言，还是通用编程语言？

### 为什么需要一个全新的框架

面对使用多年的传统框架，在开发体验上开发者大多可以接受。但是传统框架开发的应用在迁移到 Serverless 环境时，往往会遇到各种各样难以解决的问题，这些问题往往与框架底层设计密切相关。虽然可以通过框架扩展能力去解决或者缓解部分问题，实践下来的结果是框架改造门槛很高、效果不太理想、需要 Hack，不够优雅。

当您在 Serverless 采用传统框架时，往往会感觉虽然应用可以运行起来，但是真正应用到生产级别时可能会有所顾虑。当然，随着 Serverless 平台底层技术的不断发展，传统框架在 Serverless 场景上的处境也会有极大的改善。如需达到最佳状态，单方面的改变可能往往不够，需要框架也能合理去适配 Serverless 场景。就像前端 UI 框架需要 Mobile First，浏览器提供响应式支持，前端 UI 框架提供相关的适配。因此我们需要一个全新的、Serverless First 的开发框架，才能极大发挥 Serverless 优势，并让 Serverless 开发体验继承甚至超越传统开发体验。

## 为什么选择特定编程语言

目前，开源社区也存在不少语言无关的 Serverless 工具或框架，例如 Funcraft、Serverless Cloud Framework、Vercel 等。这类通用语言型 Serverless 工具在运维层面确实可以做到不错的体验，也可以形成通用的标准。但在应用代码开发、调试、运行等开发体验上可能不是很理想。每种编程语言在开发、调试和运行等方面都有其独特的地方，通用语言型 Serverless 工具很难做到统一的开发体验，同时还能做到很好。只有选择特定的编程语言，才能让开发、调试和运行等方面的体验更佳。

## 为什么选择 Typescript

Serverless 让后端开发门槛变得极低，前端开发者基于 Serverless 开发后端应用的学习成本也极低。未来越来越多的前端开发者成为全栈开发者。

Typescript 既可以开发前端，又可以开发后端，对于前端或者全栈开发者来说十分友好。

前端架构是一种类 Serverless 架构，例如，前端浏览器需要加载前端代码来执行，而 Serverless 场景也需要加载用户的代码来执行。因此前端的很多解决方案天然适合 Serverless 场景，例如，前端通过打包、压缩、Tree Shaking 来减少代码体积，减少代码部署和冷启动时间。同样，该优化方案也适用于 Serverless 场景，所以选择 Typescript，则相当于直接拥有了经过无数真实场景打磨的现成解决方案。

另外，Typescript 和 Java 很接近，Java 开发者也能很方便切换到 Typescript 技术栈。

## Malagu 带来的价值

Malagu 是基于 TypeScript 的 Serverless First、可扩展和组件化的渐进式应用开发框架。Malagu 屏蔽掉不同 Serverless 平台底层细节和 Serverless 场景存在的大部分痛点。Malagu 基于真实业务场景打磨，提供生产级别可用的解决方案。提供多云解决方案，云厂商不锁定。

## 如何使用 Malagu

Malagu 框架由一系列组件组成，每一个组件为一个 node 模块，根据您的业务场景选择合适的组件，您也可以基于组件机制开发属于自己的组件。为了快速开发，Malagu 提供了一个命令行工具，命令行工具内置不同场景的开箱即用的模板，通过命令行工具可以快速创建您的应用。

1. 执行以下命令安装命令行工具。

```
npm install -g @malagu/cli
```

2. 初始化一个项目。

```
malagu init project-name  
cd project-name # 进入项目根目录
```

3. 本地运行测试。

```
malagu serve # 启动应用，默认端口为 3000
```

4. 打开浏览器访问 <http://localhost:3000/>。

# 快速开始

最近更新时间：2023-07-18 09:24:31

我们可以使用 `@malagu/scf-plugin` 组件将应用部署到腾讯云云函数。基于约定大于配置原则，零配置，开箱即用。

## 云资源

适配器组件有一套默认的部署规则，该规则可以被覆盖。适配器组件在执行部署任务时，将使用平台提供的 SDK，根据部署规则，创建需要的云资源。如果发现云资源已经存在，则差异更新云资源。**适配器组件总是以尽可能安全的方式，创建或更新云资源。**例如，当配置了自定义域名，适配器组件则尝试创建或更新该自定义域名资源。

适配器组件将应用部署到一个函数中，也就是说一个应用对应着一个函数，如果应用较大，应该将大应用拆解成一个一个小的微应用或者微服务。如同微服务架构的粒度拆分原则，合理的粒度拆分，能够更好地进行应用管理。框架会保证一个应用在一个函数中运行的性能。

## 环境隔离

在 Malagu 框架中，提供了一个配置属性 `stage` 表示环境。而在 `@malagu/cloud-plugin` 组件约定的部署规则中，使用 `mode` 属性映射 `stage` 属性。默认提供了三套环境：测试、预发和生产。表达式规则如下：

```
stage: "${'test' in mode ? 'test' : 'pre' in mode ? 'pre' : 'prod' in mode ? 'prod' : cliContext.prod ? 'prod' : 'test'}" # test, pre, prod
```

`stage` 取值规则如下：

- **test**: 测试环境，当 `mode` 属性包含 `test` 模式，或者 `mode` 都不包含 `test`、`pre`、`prod`，且命令行参数 `-p, --prod` 未被指定。
- **pre**: 预发环境，当 `mode` 属性包含 `pre` 模式。
- **prod**: 生产环境，当 `mode` 属性包含 `prod` 模式，或者命令行参数 `-p, --prod` 被指定。

通过指定特殊的 `mode` 表示不同的部署环境：

```
# 部署到测试环境
malagu deploy -m test # 或者 malagu deploy

# 部署到预发环境，也可以直接跳过预发环境的部署，直接部署到生产环境
malagu deploy -m pre

# 部署到生成环境
malagu deploy -m prod
```

## 隔离级别

环境的隔离级别支持控制。可以使用账号隔离环境，不同环境对于不同配置文件，不同配置文件分别配置不同的云账号。同理，也可以使用 `Region`、服务别名来隔离环境。框架默认提供的是服务别名隔离环境。隔离方式可以互相叠加。

`stage` 属性值与服务别名关联（以下是默认规则，无需配置）：

```
malagu:
  cloud:
    alias:
      name: ${stage}
```

API 网关的 `environment` 关联（以下是默认规则，无需配置）：

```
malagu:
  cloud:
    apiGateway:
      release:
        environmentName: "release"
```

## 部署模式

适配器组件通过 mode 属性定义部署模式，支持的部署模式如下：

- **http**：基于自定义 API 网关 + Web 函数的部署模式。部署过程中，创建或更新 API 网关、命名空间、函数等云资源。
- **timer**：基于定时触发器 + 事件函数的部署模式。部署过程中，创建或更新定时触发器、命名空间、函数等云资源。
- **api-gateway**：基于自定义 API 网关 + 事件函数的部署模式。部署过程中，创建或更新 API 网关、命名空间、函数等云资源。
- **api-gateway-basic**：基于默认 API 网关 + 事件函数的部署模式。部署过程中，创建或更新 默认API 网关等云资源，本模式下，API 网关调用次数不计费。

```
mode:  
  - http
```

## 自定义部署规则

可以通过同名覆盖自定义部署规则。

### 默认规则

默认规则定义在 [@malagu/scf-plugin](#) 组件的 `malagu-remote.yml` 配置文件中。

### 自定义部署类型

```
mode:  
  - http # 默认值是 http, 目前支持 http、timer、api-gateway、api-gateway-basic
```

### 自定义命名空间

```
malagu:  
  cloud:  
    namespace:  
      name: xxxx # 默认值是 default
```

#### ⓘ 说明

命名空间的其他属性配置方式类似。

### 自定义函数名

```
malagu:  
  cloud:  
    function:  
      name: xxxx # 默认值是 ${pkg.name}
```

#### ⓘ 说明

函数的其他属性配置方式类似。

## 属性配置

```
malagu:  
  cloud:  
    namespace:  
      name:  
      description:  
    function:  
      name: ''
```

```
namespace:
handler:
publish:
l5Enable:
type:
codeSource:
description:
memorySize:
timeout:
runtime:
role:
clsLogsetId:
ClsTopicId:
env:
vpcConfig:
  vpcId:
  subnetId:
layers:
  name:
  version:
deadLetterConfig:
  type:
  name:
  filterType:
publicNetConfig:
  PublicNetStatus:
  eipConfig:
  eipStatus:
alias:
  name:
  functionName:
  namespace:
  description:
  routingConfig:
    additionalVersionWeights:
      version:
      weight:
    addtionVersionMatches:
      version:
      key:
      method:
      expression:
apiGateway:
  usagePlan:
    name:
    environment:
    desc:
    maxRequestNum:
    maxRequestNumPreSec:
  strategy:
    name:
    environmentName:
    strategy:
api:
  name:
  serviceTimeout:
  protocol:
  desc:
  authType:
```

```
enableCORS:
businessType:
serviceScfFunctionName:
serviceWebsocketTransportFunctionName:
serviceScfFunctionNamespace:
serviceScfFunctionQualifier:
serviceWebsocketTransportFunctionNamespace:
serviceWebsocketTransportFunctionQualifier:
isDebugAfterCharge:
serviceScfIsIntegratedResponse:
isDeleteResponseErrorCodes:
responseSuccessExample:
responseFailExample:
authRelationApiId:
userType:
oauthConfig:
  publicKey:
  tokenLocation:
  loginRedirectUrl:
responseErrorCodes:
  code:
  msg:
  desc:
  convertedCode:
  needConvert:
requestConfig:
  ApiRequestConfig:
  path:
  method:
requestParameters:
  name:
  desc:
  position:
  type:
  defaultValue:
  required:
  RequestParameter:
service:
  exclusiveSetName:
  name:
  protocol:
  description:
  netTypes:
  ipVersion:
  setServerName:
  appIdType:
release:
  environmentName:
  desc:
customDomain:
  name:
  isDefaultMapping:
  certificateId:
  protocol:
  netType:
  pathMappingSet:
    path:
  Environment:
```

## 访问数据库

最近更新时间：2023-07-19 16:35:51

Malagu 框架可以方便地集成第三方数据库操作相关的框架，例如 Sequelize、Typeorm 等。基于 Malagu 的组件机制，第三库扩展性更强，且支持属性配置，开箱即用。

目前，框架提供了对 Typeorm 库的集成，可以通过框架配置文件，配置数据库链接相关信息。另外，Malagu 框架是 Serverless First，框架在集成 Typeorm 时，对 Serverless 场景进行了最佳实践适配。同时借鉴了 Spring 事务管理机制，提供了无侵入式的事务管理，并支持事务的传播行为。

## 使用方法

1. 安装 malagu 软件，详细安装步骤请参见 [使用 Malagu](#)。
2. 框架提供了一个内置模板 database-app，执行以下命令可以快速初始化一个有关数据库操作的模板应用。

```
malagu init demo database-app
```

3. 初始化完成后，只需将数据库链接配置改成当前实际环境的配置。通过执行以下命令也可以在项目里直接安装 `@malagu/typeorm` 组件。

```
yarn add @malagu/typeorm  
# 或者执行 npm i @malagu/typeorm
```

## 配置数据源链接

在 Malagu 中，数据源链接配置与 Typeorm 类似，只是配置形式和位置稍微不同。框架为使第三库的配置方式与框架组件的配置方式保持统一，框架在集成 Typeorm 时，将 Typeorm 的原有配置方式适配成了框架组件的配置方式。更多 Typeorm 数据源链接配置说明，请参见 [Typeorm 官方文档](#)。

### 单数据源

数据源链接名称如果未设置，则默认是 default。

```
# malagu.yml  
backend:  
  malagu:  
    typeorm:  
      ormConfig:  
        - type: mysql  
          host: localhost  
          port: 3306  
          synchronize: true  
          username: root  
          password: root  
          database: test
```

### 多数据源

为区分不同的数据源链接，需要为数据源链接设置名称，有且只有一个可以不设置名称，且该链接默认名称为 default。在使用 OrmContext 相关的 API 时，将会使用数据源链接名称。

```
# malagu.yml  
backend:  
  malagu:  
    typeorm:  
      ormConfig:  
        - type: mysql  
          host: localhost  
          port: 3306  
          synchronize: true  
          username: root
```

```
password: root
database: test
- type: mysql
  name: 'datasource2'
  host: xxxx
  port: 3306
  synchronize: true
  username: root
  password: root
  database: test
```

## 数据库操作

以下示例使用 rest 风格来实现 API。

### 说明

您也可以使用 RPC 风格来实现，这两种风格类似。

```
import { Controller, Get, Param, Delete, Put, Post, Body } from '@malagu/mvc/lib/node';
import { Transactional, OrmContext } from '@malagu/typeorm/lib/node';
import { User } from './entity';

@Controller('users')
export class UserController {
  @Get()
  @Transactional({ readOnly: true })
  list(): Promise<User[]> {
    const repo = OrmContext.getRepository(User);
    return repo.find();
  }
  @Get('/:id')
  @Transactional({ readOnly: true })
  get(@Param('id') id: number): Promise<User | undefined> {
    const repo = OrmContext.getRepository(User);
    return repo.findOne(id);
  }
  @Delete('/:id')
  @Transactional()
  async remove(@Param('id') id: number): Promise<void> {
    const repo = OrmContext.getRepository(User);
    await repo.delete(id);
  }
  @Put()
  @Transactional()
  async modify(@Body() user: User): Promise<void> {
    const repo = OrmContext.getRepository(User);
    await repo.update(user.id, user);
  }
  @Post()
  @Transactional()
  create(@Body() user: User): Promise<User> {
    const repo = OrmContext.getRepository(User);
    return repo.save(user);
  }
}
```

## 数据库上下文

在 Malagu 框架中，Typeorm 的事务托管至框架管理。框架提供了一个装饰器 `@Transactional`，用于框架在执行方法前后如何开启、传播、提交和回滚事务。同时框架会将托管的 `EntityManager` 对象放到数据库上下文中，方便在业务代码中使用。另外您也可以手动管理数据库事务和创建 `EntityManager` 对象。

数据库上下文基于请求上下文实现，因此数据库上下文也是请求级别。在数据库上下文中主要提供了获取 `EntityManager` 和 `Repository` 对象相关的方法：

```
export namespace OrmContext {
  export function getEntityManager(name = DEFAULT_CONNECTION_NAME): EntityManager {
    ...
  }
  export function getRepository<Entity>(target: ObjectType<Entity>|EntitySchema<Entity>|string,
name?: string): Repository<Entity> {
    ...
  }
  export function getTreeRepository<Entity>(target: ObjectType<Entity>|EntitySchema<Entity>|string,
name?: string): TreeRepository<Entity> {
    ...
  }
  export function getMongoRepository<Entity>(target: ObjectType<Entity>|EntitySchema<Entity>|string,
name?: string): MongoRepository<Entity> {
    ...
  }
  export function getCustomRepository<T>(customRepository: ObjectType<T>, name?: string): T {
    ...
  }
  export function pushEntityManager(name: string, entityManager: EntityManager): void {
    ...
  }
  export function popEntityManager(name: string): EntityManager | undefined {
    ...
  }
}
```

## 事务管理

Malagu 框架提供了一个装饰器 `@Transactional`，以声明的方式定义事务的行为，Malagu 框架根据装饰器声明决定事务的开启、传播、提交和回滚行为。

### @Transactional

`@Transactional` 装饰器可以加在类和方法上，如果类和方法同时加上，最终的配置是使用方法的配置去合并类，且方法的配置优先级高于类上。装饰器配置选项如下：

```
export interface TransactionalOption {
  name?: string; // 多数据源链接情况下，指定数据源链接名称，默认为 default
  isolation?: IsolationLevel; // 数据库隔离级别
  propagation?: Propagation; // 事务的传播行为，支持 Required 和 RequiresNew，默认为 Required
  readOnly?: boolean; // 只读，不开启事务，默认为开启事务
}
```

示例如下：

```
@Put()
@Transactional()
async modify(@Body() user: User): Promise<void> {
  const repo = OrmContext.getRepository(User);
```

```
await repo.update(user.id, user);
}
```

## @Transactional 与 OrmContext

Malagu 框架根据装饰器的配置，在方法调用前开启事务（也可能不开启），并将 EntityManager 托管在 OrmContext 上下文中，通过 OrmContext 取到框架帮助开启过事务的 EntityManager，其中 Repository 是通过托管的 EntityManager 创建。为正确获取到 EntityManager，请确保装饰器配置的名称与通过 OrmContext 要获取的 EntityManager 名称保持一致，不指定名称，则默认为 default。

方法执行后，框架根据方法的执行情况，自动决定事务是提交还是回滚，方法执行出现异常则回滚事务，否则提交事务。

当方法存在嵌套调用带 @Transactional 装饰器的方法，由事务传播行为的配置决定是复用上层方法的事务，还是重新开启新的事务。

### 数据库查询

数据库查询大部分情况无需开启事务，但建议最好在方式加上 @Transactional 装饰器，并将 readonly 配置为 true，让框架创建一个不开启事务的 EntityManager，保持代码风格统一。示例如下：

```
@Get ()
@Transactional({ readonly: true })
list(): Promise<User[]> {
  const repo = OrmContext.getRepository(User);
  return repo.find();
}
```

### 事务传播行为

事务传播行为决定事务在需要事务的不同方法之间如何传播，目前支持两种事务传播行为：

```
export enum Propagation {
  Required, RequiresNew
}
```

- **Required**: 需要开启一个事务，如果上一层方法已经开启过事务，则复用上一个事务，否则开启一个新事务。
- **RequiresNew**: 无论上一层方法是否开启过事务，都将开启一个新事物。

#### ⚠ 注意

事务在不同的方法传播的时候，请保证方法之间是同步调用的。示例如下：

```
...
@Transactional()
async foo(): Promise<void> {
  ...
  await bar(); // 必须加上 await
}
...

...
@Transactional()
async bar(): Promise<void> {
  ...
}
```

### 绑定实体类

框架提供了一个方法 autoBindEntities 用于绑定实体类，该方法一般在模块入口文件里调用。方法包含以下两个参数：

- **entities**: 您定义的实体类。
- **name**: 您希望实体类与哪个数据源连接绑定，默认与 default 绑定。

```
export function autoBindEntities(entities: any, name = DEFAULT_CONNECTION_NAME) {  
}
```

示例如下:

```
import { autoBindEntities } from '@malagu/typeorm';  
import * as entities from './entity';  
import { autoBind } from '@malagu/core';  
  
autoBindEntities(entities);  
export default autoBind();
```

## 工具类

工具	描述
DEFAULT_CONNECTION_NAME	默认数据库连接名称 default。
autoBindEntities	绑定实体类。