云函数

操作指南



腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【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】

腾讯云

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

操作指南
- 配额管理
  - 配额限制说明
  - 配额超限管理
- 函数管理
  - 函数概述
  - 创建函数
  - 更新函数
  - 查询函数
  - 调试云函数
  - 测试函数
  - 部署函数
  - 删除函数
  - 复制函数
- Web 函数管理
  - 函数概述
  - 创建及测试函数
  - 启动文件说明
  - 触发器管理
  - Web 函数计费说明
  - 命令行部署 Web 函数
  - WebSocket 协议支持
  - SSE 协议支持
  - Web 函数请求并发管理
- 日志管理
  - 日志检索教程
  - 日志结构说明
  - 日志投递配置
  - 日志投递配置（旧）
- 并发管理
  - 并发概述
  - 并发管理体系
  - 预置并发
  - 定时预置
  - 动态指标预置
  - 并发超限
- 触发器管理
  - 创建触发器
  - 删除触发器
  - 启停触发器
- 函数 URL
  - 函数 URL 概述
  - 创建函数 URL
  - 函数 URL 认证鉴权配置
- 自定义域名
  - 配置自定义域名
- 版本管理
  - 版本管理概述
  - 查看版本
  - 发布版本
  - 使用版本

版权所有：腾讯云计算（北京）有限责任公司

别名管理
- 别名管理相关操作
- 流量路由配置
- 使用别名实现 SCF 灰度发布

权限管理
- 权限管理概述
- 角色与策略
- SCF 策略语法
- 子用户与授权

监控与告警管理
- 监控指标说明
- 配置告警
- 查看运行日志
- 应用性能管理
  - 应用性能管理概述
  - 应用性能观测
  - 使用博睿数据 APM

网络配置
- 网络配置管理
- 固定公网出口 IP
- 私有网络通信

层管理
- 层管理概述
- 创建层
- 云函数绑定层
- 使用层

执行配置
- 异步执行
- 状态追踪
- 异步执行事件管理

命名空间管理

ICP 备案

扩展存储管理
- 挂载 CFS 文件系统

DNS 缓存配置

资源托管模式管理

版权所有：腾讯云计算（北京）有限责任公司

操作指南
配额管理
配额限制说明

最近更新时间：2025-03-24 15:57:42

云函数 SCF 针对每个用户账号，均有一定的配额限制。

用户账号配额限制

内容	默认配额限制
每个地域下的函数代码总体积	100GB
每个地域下的总函数并发配额	128000MB（广州、上海、北京、成都、中国香港）
	64000MB（新加坡、东京、硅谷、法兰克福、深圳金融、上海金融）
每个地域下命名空间个数	5
每个命名空间下的总函数并发配额	可购买 函数套餐包 进行配额调整
每个命名空间下函数个数	50

函数配额限制

内容	默认配额限制
函数名称长度限制	60字符，命名空间名称和函数名称的总字符长度不超过118个字符。
单个函数的版本个数	1000个
单个函数（版本）的代码压缩前体积（包含绑定的层）	500MB
单个函数的同类型触发器数量	10
单个函数的环境变量大小	4KB
单个函数版本绑定的层版本个数	5

层配额限制

内容	默认配额限制
每个地域下层个数	20
每个层版本个数	200

注意：

- 云函数平台目前支持百万MB级的并发，对于并发需求较高的场景如电商促销，医疗生物数据并行处理等均可有效支持。
- 云函数平台在每个地域的默认状态下所有函数共享并发。用户可以自行配置 函数并发 以满足需要，如果您需要提升各项配额或者增加命名空间维度的并发配额管理可直接购买 函数套餐包 。
- 云函数 SCF 下的 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。

函数运行环境的限制

内容	配额限制

版权所有：腾讯云计算（北京）有限责任公司

内存分配	最小64MB ，最大3072MB。从128MB起，以128MB为阶梯递增。
临时缓存空间，即 `/tmp` 目录大小	512MB
超时时间	最小1秒，最大900秒
文件描述符数量	1024
进程和线程总数	1024
同步请求事件大小	6MB
同步请求响应大小	6MB
异步请求事件大小	128KB

注意：
向云函数中传入文件时，若文件在 Base64 编码后小于6MB，您可以通过函数 URL 将编码后的内容传入 SCF。若文件在 Base64 编码后大于等于6MB，建议您将文件上传至 COS，并将 Object 地址传递给 SCF，由 SCF 从 COS 拉取文件，以完成大文件的上传。

配额超限管理

最近更新时间：2024-11-29 17:29:02

超出云函数 SCF 配额限制及解决方案如下表所示：

内容	解决方案
每个地域下命名空间个数达到上限	可通过 提交工单 提升配额限制。
命名空间下函数个数达到上限	每个地域下可支持多个命名空间，可优先使用其他命名空间函数配额。 如当前地域下命名空间及函数个数均达到上限，可通过 提交工单 提升配额限制。 云开发通过套餐进行配额管理，请查看 云开发产品定价 了解各套餐包额度并通过升级套餐包提升配额。
单个函数的触发器个数达到上限	建议将函数业务粒度细化，通过多个函数分别绑定触发器的方式解决单个函数触发器上限问题。 如因业务需要无法拆分，可通过 提交工单 提升配额限制。
每个地域的函数总并发配额达到上限	请在函数并发配额页尝试调整当前地域下总并发配额。
函数初始化超时时间超限	可通过 提交工单 提升配额限制。

版权所有：腾讯云计算（北京）有限责任公司

函数管理
函数概述

最近更新时间：2023-08-21 09:41:21

函数是云函数 SCF 管理、运行的基本单元，通常由一系列配置与一系列可运行代码/软件包组成。您可以通过 API 触发函数运行，还可以通过不同的触发器向函数传递不同的事件触发函数运行并对事件进行处理。

函数相关概念

地域

函数资源必须归属于某个地域，SCF 已经支持的地域可参见 支持地域 。

命名空间

函数资源必须创建在某个地域的某个命名空间下，每个地域默认具有一个 `default` 命名空间，支持用户 新建命名空间 ，命名空间名称创建后不可修改。

函数名称

函数名称为函数的唯一标识，同一个命名空间下的函数名称不可重复，创建后不可修改。

函数类型

SCF 支持 `event` 函数和 `Web` 函数两种函数类型。

- event 函数由指定格式的事件触发，例如定时触发事件、COS 触发事件等，事件结构详情见 触发器 。
- Web 函数专注于优化 Web 服务场景，可以直接接受并处理 HTTP 请求，详情见 Web 函数 。

时区

云函数内默认使用 UTC 时间，您可以通过配置环境变量 TZ 修改。在您选择时区后，将自动添加对应时区的 TZ 环境变量。

运行环境

函数的代码运行环境，SCF 现已支持 Python 、Node.js 、PHP 、JAVA 、Golang 、Custom Runtime 和 镜像部署 。

函数执行方法

执行方法表明了调用云函数时需要从哪个文件中的哪个函数开始执行，可分为以下三种方式：

- 一段式格式为 **"[文件名]"** ，支持 Golang 环境时使用，例如 "main"。
- 两段式格式为 **"[文件名].[函数名]"** ，支持 Python、Node.js 及 PHP 环境时使用，例如 "index.main_handler"。

> **说明**
> 两段式的执行方法中，前一段指向代码包中不包含后缀的文件名，后一段指向文件中的入口函数名。需确保代码包中的文件名后缀与语言环境匹配，例如 Python 环境为 `.py` 文件，Node.js 环境为 `.js` 文件。

- 三段式格式为 **"[package].[class]::[method]"**，支持 Java 环境时使用，例如 "example.Hello::mainHandler"。

函数描述

可用于记录函数相关作用等信息。

函数相关配置

除上述配置外，还可通过控制台编辑函数配置或 更新函数配置 修改以下内容，配置更多函数运行时的信息：

资源类型

函数支持算力包含 CPU、GPU。

资源规格

设置资源类型对应的规格，如 CPU 不同内存配置，GPU 不同卡类型等，详情见 函数算力支持 。

初始化超时时间

指定函数初始化阶段最长运行时间，可选值范围为3-300秒，镜像部署函数默认90秒，其他函数默认60秒。

> **注意**
> - 函数初始化阶段包括准备函数代码、准备镜像、准备层等相关资源以及执行函数主流程代码。如果您的函数具有较大的镜像或复杂的函数主流程业务逻辑，请适当调大初始化超时时间。
> - 初始化超时时间仅在触发实例冷启动调用的场景下生效。
> - 客户端的等待时间建议稍大于初始化超时时间与执行超时时间的和。

执行超时时间

指定函数的最长运行时间，可选值范围为1秒- 900秒，默认3秒。

环境变量

在配置中定义的环境变量可在函数运行时从环境中获取到。详情见 环境变量 。

运行角色

将角色中包含的策略对应权限授权给函数，详情见 权限管理 。例如，函数代码中执行将某对象写入对象存储 COS 的动作，需要为该函数配置具有写 COS 权限的运行角色。

日志配置

将函数调用日志投递至指定的日志主题，详情见 日志管理 。

网络配置

配置函数网络访问权限，详情见 网络配置 。
- 公网访问：默认启用，关闭后函数无法访问公网资源。
- 固定出口 IP：启用后平台将为函数分配一个固定的公网出口 IP。
- 私有网络：启用后，函数可以访问同一个私有网络下的资源。

文件系统

启用后，函数可以访问所挂载的文件系统的资源。详情见 挂载 CFS 文件系统 。

执行配置

执行配置包含异步执行、状态追踪和异步执行事件管理，详情见 执行配置 。
- 异步执行：启用后，函数执行超时时间最大可支持 24 小时，函数创建后该配置无法修改。
- 链路追踪：仅在异步执行启用的情况下可开启，开启后针对异步执行的事件，将开始记录响应事件的实时状态，并提供事件的统计、查询及终止服务，产生的事件状态数据将为您保留3天。

异步调用配置

通过 异步调用配置 设置异步调用场景下的重试策略，还可以配置 死信队列 收集错误事件信息、分析失败原因。

应用性能观测

启用后，SCF 将上报函数运行基本耗时到指定的应用性能监控 APM 业务系统，您还可以在函数代码中埋点进行自定义上报，帮助您跟踪和监控函数的执行。详情见 应用性能观测 。

DNS 配置

在云函数的使用场景下，域名解析延时有可能导致函数执行超时失败，影响正常的业务逻辑；在函数高频调用的情况下，有可能导致 DNS 服务器解析超出频率限制，同样导致函数执行失败。云函数提供了 DNS 缓存配置来解决上述问题。DNS 缓存可以提升域名解析效率，缓解网络抖动等因素对域名解析成功率的影响。详情见 DNS 缓存配置 。

函数可执行的操作

- 创建函数：创建一个新的函数。
- 更新函数：
  - 更新函数配置：更新函数的各项配置。

- 更新函数代码：更新函数的运行代码。

- 获取详情：获取函数配置、触发器及代码详情。
- 测试运行函数：根据需要，通过同步或异步方法触发函数运行。
- 获取日志：获取函数运行情况及输出的日志。
- 删除函数：删除不再需要的函数。
- 复制函数：复制函数到指定的地域、指定的名称、指定的配置。

函数的触发器相关操作有：

- 创建触发器：创建一个新的触发器。
- 删除触发器：删除已有触发器。
- 启停触发器：通过设置触发器启动或停止来临时停止云函数被事件源所发生的事件触发。

版权所有：腾讯云计算（北京）有限责任公司

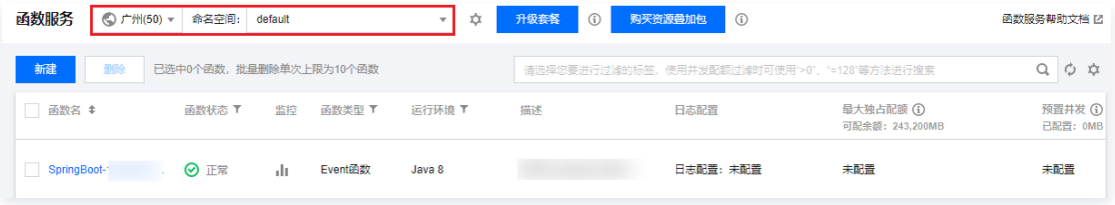
创建函数

最近更新时间：2024-11-07 10:23:52

腾讯云云函数提供多种方式创建函数，本文向您介绍如何通过控制台和命令行工具创建函数。

使用控制台创建函数

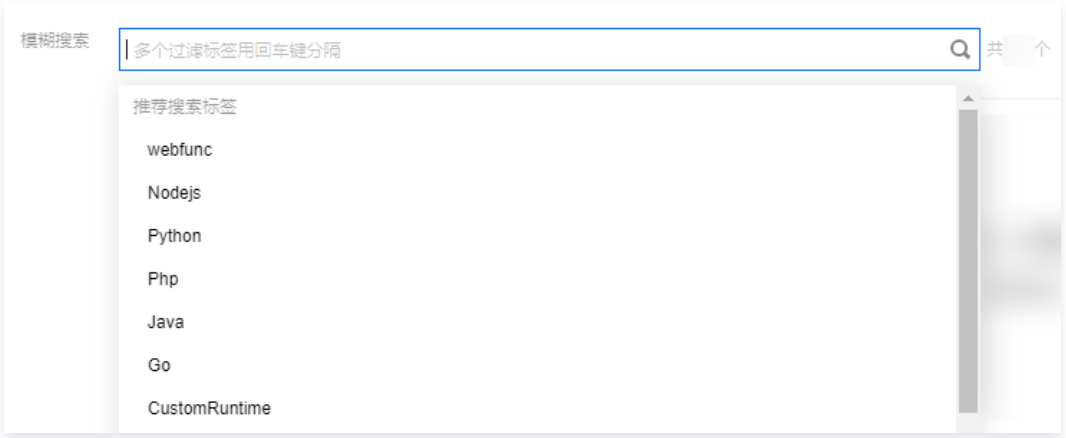
1. 登录 Serverless 控制台，单击左侧导航栏的**函数服务**。
2. 在函数服务页面上方选择期望创建函数的地域和命名空间，并单击**新建**，进入函数创建流程。如下图所示：



3. 在“新建函数”页面，您可以根据实际需求选择创建函数的方式。
   - **模板创建**：通过填写必选的函数名称，使用函数模板中的配置来完成函数的创建。
   - **从头开始**：通过填写必填的函数名称、运行环境来完成函数的创建。
   - **使用容器镜像**：基于容器镜像来创建函数。详情见 使用镜像部署函数 。
4. 配置函数基础信息。

模板创建

1. 在“模糊搜索”中添加标签查询模板。如下图所示：



2. 选择模板后，单击**下一步**。
3. 填写函数基础信息。
   - **函数名称**：函数名称默认填充，可根据需要自行修改。
   - **地域**：地域默认填充，可根据需要自行修改。
   - **时区**：云函数内默认使用 UTC 时间，您可以通过配置环境变量 TZ 修改。在您选择时区后，将自动添加对应时区的 TZ 环境变量。

从头开始

填写函数基础信息。

- **函数类型**：支持选择**事件函数**和**Web 函数**。
  - 事件函数：接收云 API、多种触发器的 JSON 格式事件触发函数执行。详情见 事件函数概述 。
  - Web 函数：直接接收 HTTP 请求触发函数执行，适用于 Web 服务场景。详情见 Web 函数概述 。
- **函数名称**：函数名称默认填充，可根据需要自行修改。
- **地域**：地域默认填充，可根据需要自行修改。

版权所有：腾讯云计算（北京）有限责任公司

- **运行环境**：运行环境默认填充，可根据需要自行修改。
- **时区**：云函数内默认使用 UTC 时间，您可以通过配置环境变量 TZ 修改。在您选择时区后，将自动添加对应时区的 TZ 环境变量。

使用容器镜像

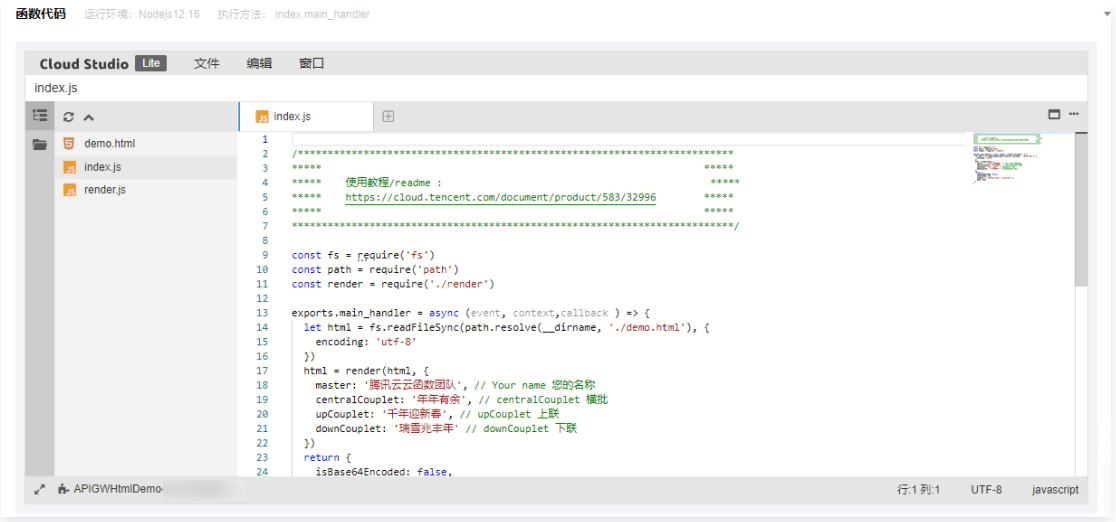
填写函数基础信息。

- **函数类型**：支持选择**事件函数**和**Web 函数**。
  - 事件函数：接收云 API、多种触发器的 JSON 格式事件触发函数执行。详情见 事件函数概述 。
  - Web 函数：直接接收 HTTP 请求触发函数执行，适用于 Web 服务场景。详情见 Web 函数概述 。
- **函数名称**：函数名称默认填充，可根据需要自行修改。
- **地域**：选择函数部署的地域，请务必与镜像仓库处于同一地域。
- **时区**：云函数内默认使用 UTC 时间，您可以通过配置环境变量 TZ 修改。在您选择时区后，将自动添加对应时区的 TZ 环境变量。

5. 配置函数代码。

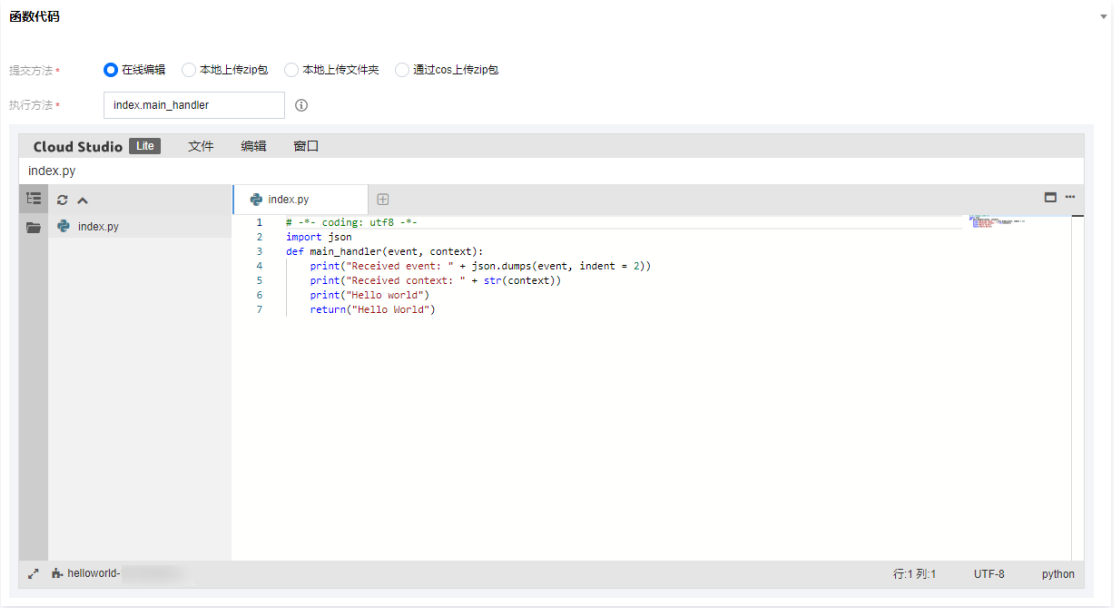
模板创建

运行环境、执行方法默认填充。如下图所示：



从头开始

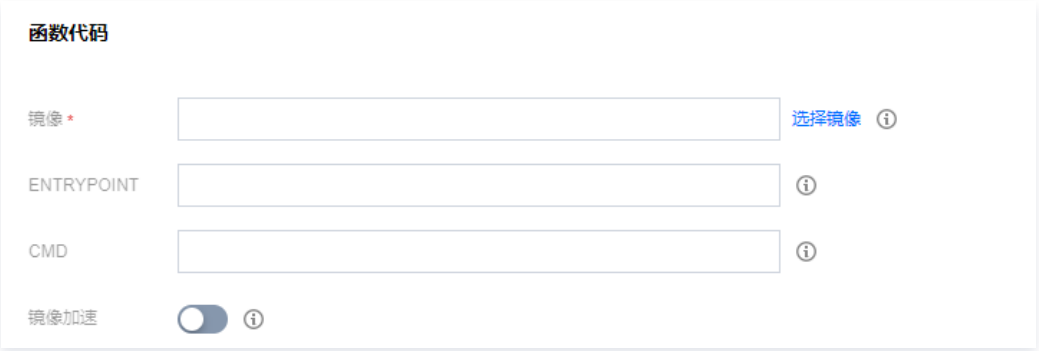
选择函数代码提交方法和执行方法。如下图所示：



- **提交方法**：支持在线编辑、本地上传zip包、本地上传文件夹、通过 cos 上传 zip 包。
  - 针对脚本类语言：可直接使用函数代码编辑器。
  - 针对非脚本类语言：通过 zip 包上传、通过对象存储 COS 上传的方式提交函数代码进行编辑。
- **执行方法**：执行方法表明了调用云函数时需要从哪个文件中的哪个函数开始执行。详情见 函数执行方法 。

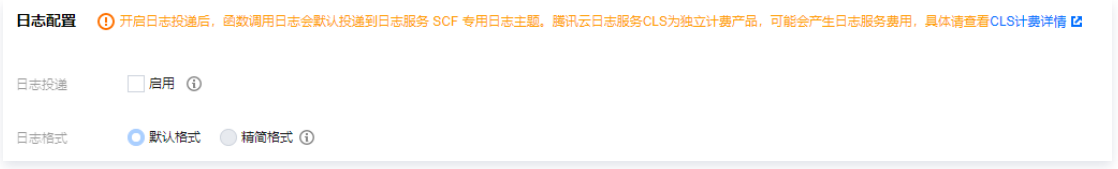
使用容器镜像

填写镜像相关内容。如下图所示：



- **镜像**：请选择当前地域镜像仓库已经构建好的镜像。
- **ENTRYPOINT**：容器的启动命令。该参数为可选参数，如果不填写，则默认使用 Dockerfile 中的 Entrypoint。输入规范，填写可运行的指令，例如 `python` 。
- **CMD**：容器的启动参数。该参数为可选参数，如果不填写，则默认使用 Dockerfile 中的 CMD。输入规范，以“空格”作为参数的分割标识，例如 `-u app.py` 。
- **镜像加速**：默认不开启。开启加速后，云函数将较大程度减少拉取镜像的耗时。开启过程需要 30 秒以上时间，请耐心等待。

6. 在日志配置中，选择是否开启日志投递。如下图所示：



日志投递默认不开启。启用时，可将函数运行日志实时投递到指定位置。详情见 日志投递配置 。

> ⚠ **注意：**
> 镜像部署函数和 Web 函数暂不支持日志格式选择。

7. 在高级配置中，您可以根据实际需求对函数进行环境配置、权限配置、层配置、网络配置等，详情见 函数相关配置 。
8. 在触发器配置中，选择是否创建触发器。如果您选择“自定义创建”，详情见 触发器概述 。
9. 单击**完成**。您可以在 函数服务 中查看已创建的函数。

使用命令行工具创建函数

您可根据实际需求，选择更多方式创建函数：

- 使用 Serverless Cloud Framework 命令行工具，可参考 使用 CLI 创建函数 。
- 使用 VS Code ，可参考 使用 VS Code 插件创建函数 。

版权所有：腾讯云计算（北京）有限责任公司

更新函数

最近更新时间：2024-07-22 11:57:53

本文向您介绍如何通过控制台和命令行工具更新函数配置和函数代码。

更新函数配置

控制台更新函数配置

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. 切换至函数配置页面，单击右上角的**编辑**，进入编辑模式。如下图所示：



4. 可根据需求修改函数的基础配置、环境配置、权限配置、日志配置和网络配置等信息。详情请参见 函数相关配置 。
5. 修改完成后，单击**保存**，保存修改后的配置。

   如需取消操作，可单击**取消**，取消修改的配置。

Serverless Cloud Framework 更新函数配置

1. 如需修改函数配置，可直接修改函数根目录下的 `serverless.yml` 配置文件。如下所示：
   - 老版本 CLI（serverless-cloud-framework@1.2.0以下）

```
# serverless.yml
component: scf # (必填) 引用 component 的名称，当前用到的是 tencent-scf 组件
name: scfdemo # (必填) 该组件创建的实例名称


inputs:
  name: scfFunctionName
  src: ./src
  runtime: Nodejs10.15 # 云函数的运行时环境。除 Nodejs10.15 外，可选值为：Python2.7、Python3.6、
Nodejs6.10、Nodejs8.9、Nodejs12.16、PHP5、PHP7、Golang1、Java8。
  region: ap-guangzhou
  handler: index.main_handler
  events:
 - apigw:
     name: serverless_api
     parameters:
       protocols:
         - http
         - https
       serviceName:
       description: The service of Serverless Cloud Framework
       environment: release
       endpoints:
         - path: /index
```

```
          method: GET
```

- 新版本 CLI（serverless-cloud-framework@1.2.0 及以上），如您使用的是函数 URL，请将 `events` 修改为：

```
events:
  - http:
      parameters:
        netConfig:
          enableIntranet: true
          enableExtranet: true
        qualifier: $DEFAULT
        authType: NONE
```

2. 修改完成后，通过 Serverless Cloud Framework，执行 `scf deploy` 命令部署函数。

更新函数代码

控制台更新函数代码

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. 切换至函数代码页面，选择提交方法通过以下方式进行函数代码编辑：
   - 针对脚本类语言：可直接使用函数代码编辑器。
   - 针对非脚本类语言：通过 zip 包上传、通过对象存储 COS 上传的方式提交函数代码进行编辑。
4. 修改完成后，单击**部署**，将修改后的代码部署至$LATEST 版本。

Serverless Cloud Framework 更新函数代码

在本地修改函数代码后，通过 Serverless Cloud Framework 执行 `scf deploy` 命令，即可部署函数并完成代码更新。

> **说明：**
> Serverless Cloud Framework 的开发模式支持函数的同步更新，详情请参见 开发模式与云端调试。

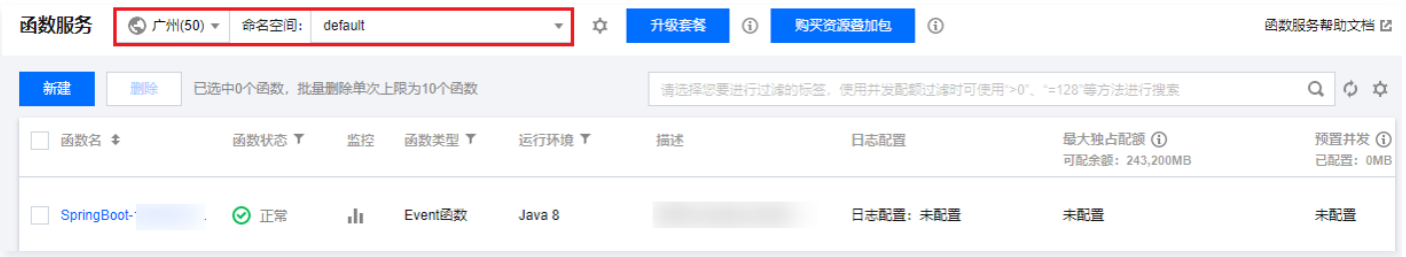
查询函数

最近更新时间：2023-08-31 09:28:31

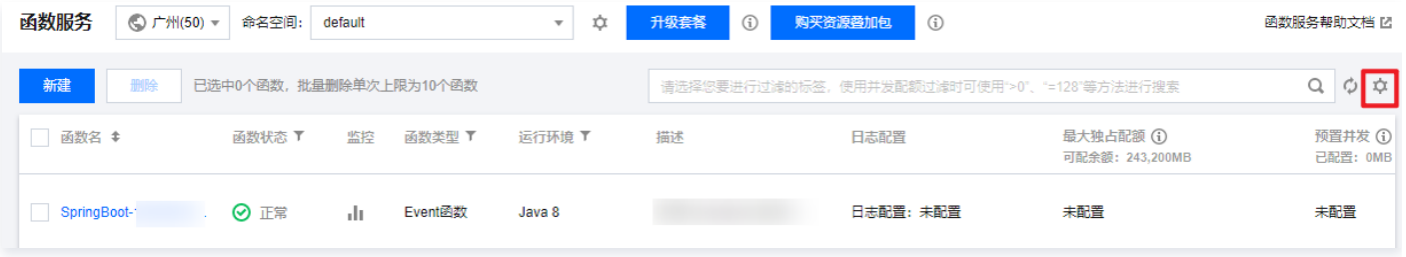
通过控制台或 Serverless Cloud Framework 命令行均可以完成函数查询。

通过控制台查看函数

1. 登录 Serverless 控制台，选择左侧导航栏中的**函数服务**。
2. 在“函数服务”页面上方选择期望查看函数所在的地域及命名空间。通过函数列表，可查看指定地域及命名空间内的全部函数。如下图所示：



3. 函数列表中包括了函数名、监控、函数类型、运行环境、日志配置、创建时间等，您可根据自身需求自定义列表字段。单击函数列表右侧 ⚙ 。如下图所示：



在弹窗中勾选您想显示的列表详细信息，单击**确定**。如下图所示：



4. 单击函数名称，可进入该函数的详情页面。如下图所示：



函数详情页面包含以下内容：

- **函数管理**：可查看并管理函数配置、函数代码、函数层。
- **版本管理**：可通过发布版本固定函数代码及配置内容。详情见 函数版本。
- **别名管理**：使用别名可以调用已绑定的函数。详情见 别名管理。
- **触发管理**：展示函数已配置触发器，并可以通过此页面创建触发器。详情见 触发器管理。
- **监控信息**：显示函数运行监控信息。详情见 函数监控。
- **日志查询**：查看函数运行日志，并可以根据一定条件过滤查询日志。详情见 日志信息。
- **并发配额**：展示函数的并发额度，可以通过此页面设定函数最大独占配额和预置并发。详情见 并发管理。
- **部署日志**：查看云函数部署日志信息。

通过 Serverless Cloud Framework 命令行获取部署信息

> **说明**
> 在使用 Serverless Cloud Framework 工具之前，请参考 安装 Serverless Cloud Framework 完成安装。

您可通过 Serverless Cloud Framework，执行 `scf info` 命令查看部署信息。

调试云函数

最近更新时间：2023-09-11 21:26:31

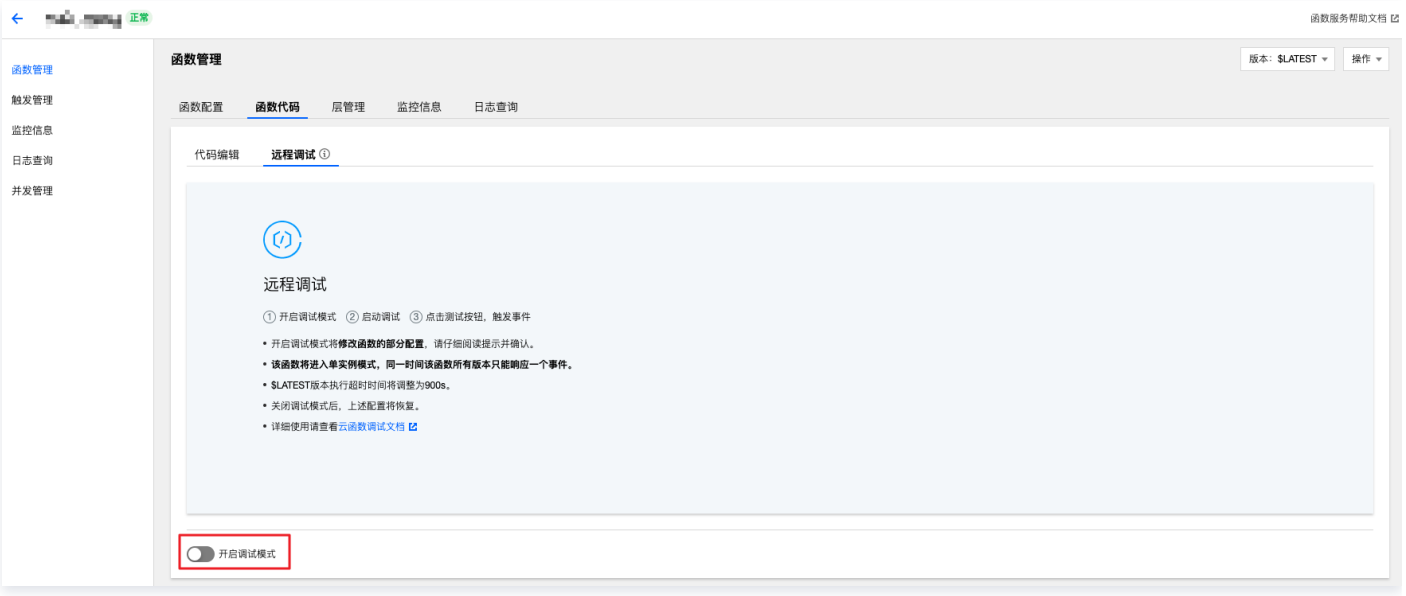
云函数控制台现已支持在线调试功能，您可以通过控制台调试与定位问题。

> **注意**
> 目前在线调试功能仅支持使用 Chrome 浏览器，以及仅支持 Node.js 10.15 和 Node.js 12.16 开发语言。

开启调试模式

> **注意**
> 在使用在线调试之前，需要您手动开启函数的调试模式。**开启函数的调试模式将会变更函数的部分配置**，关闭调试模式后将会恢复，可能对您的业务产生影响，请您务必确认以下内容：
> - 该函数将进入单实例模式，同一时间该函数所有版本只能响应一个事件，并发超出的事件将会调用失败。
> - 执行超时时间调整为900秒，调试期间执行超时时间不可设置。
> - 预置多个实例会缩至单个实例。
> - 开启调试模式的函数执行性能会降低。该函数将进入单实例模式，同一时间该函数所有版本只能响应一个事件，并发超出的事件将会调用失败。

1. 登录 云函数控制台，在左侧选择**函数服务**。
2. 在**函数服务**页面上方，选择期望开启调试模式函数的地域。并在页面中单击期望开启调试模式的函数名，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 > 远程调试**，并单击**开启调试模式**。如下图所示：



4. 在弹出窗口中单击**确认**，即可完成调试模式的开启。如下图所示：

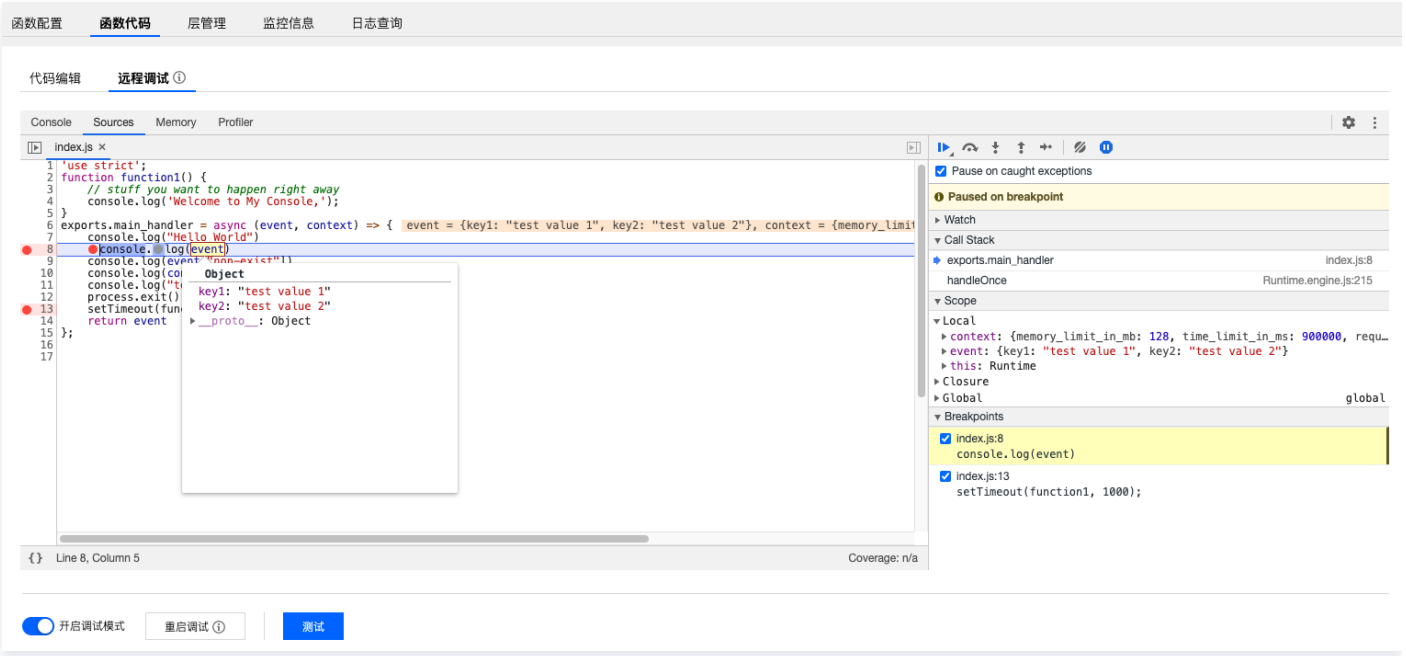


调试步骤

1. 开启调试模式 后，函数更新后会自动启动调试。

> **注意**
> 若调试模式已开启，当您再次进入调试界面时，则需手动选择**启动调试**。

2. 待加载完成后，页面将自动展示入口文件。若要打开任意您需要的文件，可使用快捷键 Cmd + P（Mac）或 Ctrl + P（Windows）。
3. 您可根据需要设置断点，单击**测试**即可根据测试模板触发测试。如下图所示：



> **说明**
> 更多关于调试工具的内容，可查阅 Chrome DevTools。

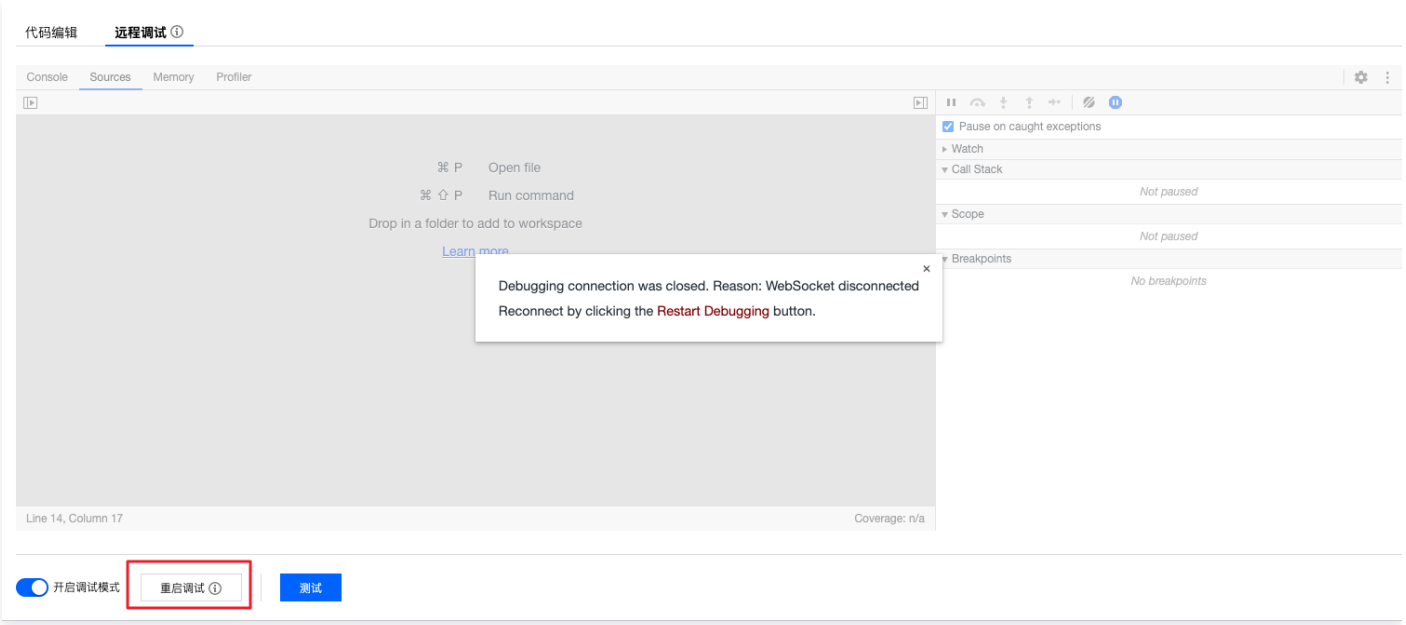
关闭调试模式

1. 在**函数管理**页面中，选择**函数代码** > **远程调试**。
2. 关闭**开启调试模式**按钮，即可关闭调试模式，函数配置将恢复。

> **注意**
> 在调试页面修改代码不会同步到云端。如果您需要保存更改的代码，请保存并使用代码在线编辑功能。

常见问题

- 由于网络、代码异常等情况可能造成 inspector 断开连接，当出现诸如下图情形时，需要您单击**重启调试**重新连接。



- 若您的函数运行正常，但在调试模式遇到 Out Of Memory 错误，您需要调大函数的内存配置，以解决开启调试模式时函数所需内存增加导致内存不足的问题。

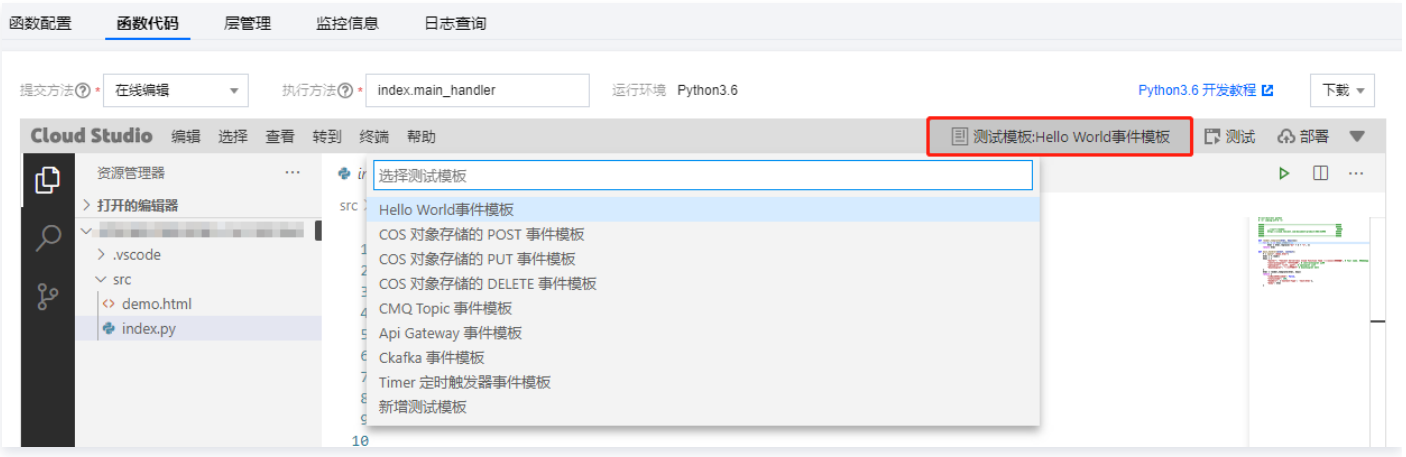
测试函数

最近更新时间：2023-09-01 10:40:01

云函数的测试功能，用于通过控制台直接发起函数调用，模拟触发器发送的触发事件，并展示云函数的执行情况、返回内容、运行日志。在控制台中的函数详情页面，可以通过进入函数代码子页面，单击**测试**，测试运行函数。以下视频将为您介绍测试函数：
观看视频

操作步骤

1. 登录 云函数控制台，在左侧选择**函数服务**。
2. 在**函数服务**页面，单击目标函数名，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**。
4. 在编辑器中选择期望使用的测试模板。如下图所示：



5. 单击**测试**即可完成函数测试。

测试事件模板

在产品迭代过程中，默认测试事件模板会不断新增。
测试事件模板用来模拟在相应触发器触发云函数运行时，传递给云函数的事件和内容，在函数中以 event 入参的形式体现。测试事件模板需要是 JSON 格式的数据结构。目前已包含的默认测试事件模板和说明如下：

- Hello World 事件模板：简单、自定义的事件模板，在通过云 API 触发函数时，可输入自定义事件内容。
- COS 上传、删除文件事件模板：模拟绑定 COS 对象存储触发器后，在 Bucket 中有文件上传或删除时触发云函数所产生和传递的事件。
- CMQ Topic 事件模板：模拟绑定 CMQ 消息队列主题订阅后，在消息队列中收到消息的情况下触发云函数所产生和传递的事件。
- API 网关事件模板：模拟 API 网关绑定云函数后，在有 API 请求到达 API 网关时触发云函数所产生和传递的事件。

> **说明：**
> 函数控制台测试场景下有调用超时时间限制。超时时间60s以内为同步调用，60s以上为异步调用。

自定义测试事件模板

在测试前，可以直接选择默认测试模板，也根据自身的事件情况对测试模板进行修改并保持为自定义测试模板。修改后的测试模板将用来作为触发函数运行的事件内容传递给函数。修改后的测试模板需要为 JSON 格式。

自定义测试事件模板使用限制

针对自定义测试事件模板，有如下使用限制：

- 自定义测试事件模板基于账号范围，同一账号下不同函数共用相同测试事件模板。
- 单个账号最多可配置 5 个自定义测试模板。
- 每个自定义测试模板内容最大 64 KB。

新建和保存自定义测试事件模板

在测试时，如果不想要每次均修改模板，可以将修改后的测试模板保存为自定义模板。在选定需要修改的模板后，可以单击**新建模板**按钮，完成对模板的修改，并输入一个容易记忆的名字后保存。后续在使用保存的模板测试后，再次进入测试界面时，会仍然保存为上次测试使用的函数模板。

删除自定义测试事件模板

对于不再使用的自定义模板，可以通过选择模板后单击**删除**按钮进行删除。

部署函数

最近更新时间：2022-12-07 15:01:11

通过控制台部署

部署程序包是 SCF 平台运行的所有代码和依赖项的 zip 集合文件，在创建函数时需要指定部署程序包。用户可以在本地环境创建部署程序包并上传至 SCF 平台，或直接在 SCF 控制台上编写代码由控制台为您创建并上传部署程序包。请根据以下条件确定您是否可使用该控制台创建部署程序包：

- 简单场景：如果自定义代码只需要使用标准库及腾讯云提供的 COS、SCF 等 SDK 库，且只有一个代码文件时，则可以使用 SCF 控制台中的内联编辑器。控制台会将代码及相关的配置信息自动压缩至一个能够运行的部署程序包中。
- 高级场景：如果编写的代码需要用到其他资源（如使用图形库进行图像处理，使用 Web 框架进行 Web 编程，使用数据库连接库用于执行数据库命令等），则需要先在本地环境创建函数部署程序包，然后再使用控制台上传部署程序包。

打包要求

ZIP 格式

直接上传至 SCF 平台，或通过上传 COS 再导入 SCF 方式提交的代码包，要求为 ZIP 格式 。用于压缩或解压的工具，在 Windows 平台下可使用例如 7-Zip 工具，在 Linux 平台下可使用 zip 命令行工具。

打包方式

打包时，需要针对文件进行打包，而不是针对代码整体目录进行打包；打包完成后，入口函数文件需要位于包内的根目录。

- 在 Windows 下打包时，可以进入函数代码目录，全选所有文件以后，单击鼠标右键，选择“压缩为 zip 包”，生成部署程序包。通过 7-Zip 等工具打开 zip 包浏览时，包内应该直接包含入口程序与其他库。
- 在 Linux 下打包时，可以进入函数代码目录，通过调用 zip 命令时，将源文件指定为代码目录下的所有文件，实现生成部署程序包，例如 `zip /home/scf_code.zip * -r` 。

部署程序包示例

下面展示在本地环境创建 Python 部署程序包的示例过程。

> ⚠ 注意
> - 通常情况下在本地安装的依赖库在 SCF 平台上也能很好运行，但少部分情况下安装的 binary 文件可能产生兼容性问题，如果发生了此问题请您尝试 联系我们 。
> - 示例中针对 Python 开发语言，将在本地使用 pip 工具安装库及依赖项，请确保您本地已经安装了 Python 和 pip。

Linux 下创建 Python 部署程序包

1. 创建一个目录：

```
mkdir /data/my-first-scf
```

2. 将创建的此函数所有 Python 源文件（.py 文件）保存在此目录。
3. 使用 pip 安装所有依赖项至此目录：

```
pip install <module-name> -t /data/my-first-scf
```

例如，以下命令会将 Pillow 库安装在 my-first-scf 目录下：

```
pip install Pillow -t /data/my-first-scf
```

4. 在 my-first-scf 目录下，压缩所有内容。特别注意，需要压缩目录内的内容而不是目录本身：

```
cd /data/my-first-scf && zip my_first_scf.zip * -r
```

> ⚠ 注意

- 针对有编译过程的库，为保持和 SCF 运行环境的统一，建议打包过程在 CentOS 7 下进行。
- 如果在安装过程中或编译过程中，有其他软件、编译环境、开发库的需求，请根据安装提示解决编译和安装问题。

Windows 下创建 Python 部署程序包

建议您将已经在 Linux 环境下运行成功的依赖包和代码打包成 zip 包作为函数的执行代码，具体操作请参考 代码实操 – 获取COS上的图像并创建缩略图 。
针对 Windows 系统，同样可以使用 `pip install <module-name> -t <code-store-path>` 命令安装 Python 库，但是针对需要编译或带有静态、动态库的包，由于 Windows 下编译生成的库无法在 SCF 的运行环境（CentOS 7）中被调用运行，因此 Windows 下的库安装仅适合纯 Python 实现的库。

通过 Serverless Cloud Framework 命令行部署

说明
在使用 Serverless Cloud Framework 工具之前，请通过 安装 Serverless Cloud Framework 完成安装。

您可通过 Serverless Cloud Framework，执行 `scf deploy` 命令部署函数。

删除函数

最近更新时间：2022-12-15 16:22:16

通过控制台或 Serverless Cloud Framework 命令行均可以完成函数删除操作。

通过控制台删除函数

1. 登录 Serverless 控制台，在左侧选择函数服务。
2. 在“函数服务”页面选择地域和命名空间，查看指定地域内的全部函数。
3. 在函数列表中勾选需删除的函数后，单击**删除**。如下图所示：



4. 在“删除函数”弹窗中确认信息后，单击**确定**即可删除函数。

通过 Serverless Cloud Framework 删除函数

> **说明**
> 在使 Serverless Cloud Framework 工具之前，请参考 安装 Serverless Cloud Framework 完成安装。

您可通过 Serverless Cloud Framework，执行 `scf remove` 命令删除部署项目。

复制函数

最近更新时间：2024-08-19 18:08:31

操作场景

您可通过云函数控制台实现跨地域、跨空间的函数复制。在复制函数时，您可选择仅复制函数代码，或同时复制函数代码及函数配置两种方式。对于代码重复程度较高的函数，我们可以通过复制功能快速创建函数，修改代码，快速的实现多个有细微差异的云函数。

功能简介

复制源及复制目标

函数类型	描述	使用限制
源函数	被复制的云函数成为源函数	可以选择任意地域、任意命名空间的云函数进行复制。 默认复制源函数的 $LATEST 版本内容。
目标函数	复制到的函数称为目标云函数	复制的目标函数可以选择任意地域、任意命名空间及自定义命名。 在选择的地域、命名空间内，若有同名函数，复制操作将覆盖同名函数。 复制的目标函数仅生成或更新 $LATEST 版本。

说明：

`$LATEST` 版本为开发和测试使用的版本，用于代码的进一步开发和调试。

复制方式

云函数可复制的内容包括函数代码及其配置：

- 函数代码：包括函数的代码包、运行环境、执行方法。
- 函数配置：包含函数的内存、超时、描述、环境变量、网络、日志等配置内容，**不包含触发器配置**。

函数复制时，可选择以下两种复制方式：

复制方式	描述	使用限制
仅复制代码	仅将源函数的代码复制到目标函数的代码	目标函数如果存在，则使用原配置，否则使用默认配置。 目标函数如果存在，运行环境需要与源函数运行环境相同。
复制代码及配置	将源函数的代码与配置均复制到目标函数	目标函数如果存在，运行环境需要与源函数运行环境相同。

注意：

- 如果源函数与目标函数处于不同地域，在复制代码及配置时，函数配置中的网络、日志配置将无法复制到目标函数中。
- 由于在跨地域时不具有相同的对象，若有地域属性的配置项，将会导致该配置无法复制。如需补充配置，您可以在完成复制后，手动编辑云函数，修改所需配置。

操作步骤

1. 登录 Serverless 控制台 。
2. 在左侧导航栏中，选择 函数服务 ，进入函数服务管理页面。
3. 在**函数服务**上方，选择期望更新的函数所在地域，查看该地域下的所有函数。
4. 在函数列表中，选择需要复制的源函数行，单击**操作**栏中的**复制**。
5. 在弹出的**函数复制**窗口中，填写以下信息：
   - 所属地域：目标函数的所属地域。
   - 命名空间：目标函数所属命名空间。
   - 函数名：目标函数名。
   - 复制内容：通过勾选**函数配置**实现**仅复制函数代码**或**复制函数代码与配置**。
   - 覆盖目标函数：勾选此项，则会覆盖目标地域下的同名函数。

版权所有：腾讯云计算（北京）有限责任公司

- 描述：目标函数描述信息，此项可选。

6. 单击**提交**完成复制。
   如果目标函数已存在，请在警告窗口中再次确认，或取消后重新修改函数名。

Web 函数管理

函数概述

最近更新时间：2024-07-22 11:57:53

Web 函数（Web Function）是云函数的一种函数类型，区别于事件函数（Event Function）对于事件格式的限制，专注于优化 Web 服务场景，用户可以直接发送 HTTP 请求到 URL 触发函数执行。新增 API 网关触发器均迁移至函数 URL。

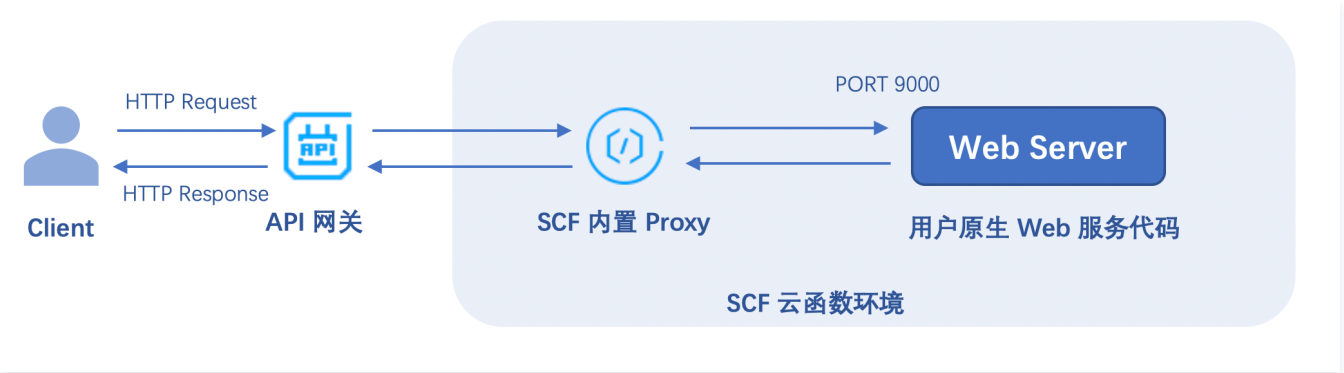
功能与优势

相较于事件型函数，Web 函数在支持 Web 服务场景的能力上，具备以下优势：

- 函数可以直接接收并处理 HTTP 或 WebSocket 原生请求，API 网关不再需要做 json 格式转换，减少请求处理环节，提升 Web 服务性能。
- Web 函数的编写体验更贴近编写原生 Web 服务，可以使用 Node.js 原生接口，保证和本地开发服务体验一致。
- 丰富的框架支持，您可以使用常见的 Web 框架（例如 Nodejs Web 框架：`Express`、`Koa`）编写 Web 函数，也可以将您本地的 Web 框架服务以极小的改造量快速迁移上云。
- Web 函数自动为您创建 API 网关服务，部署完成后，网关侧会自动生成一个默认 URL 供用户访问和调用，简化了学习成本和调试过程。
- 控制台提供了测试能力，您可以在函数控制台快速测试您的服务。

运行原理

Web 函数运行原理如下图所示：



用户发送的 HTTP 请求经过 API 网关后，网关侧将原生请求直接透传的同时，在请求头部添加了网关触发函数时需要的函数名、函数地域等内容，并一起传递到函数环境，触发后端函数执行。
函数环境内，通过内置的 Proxy 实现 Nginx 转发，并去除头部非产品规范的请求信息，将原生 HTTP 请求通过指定端口发送给用户的 Web Server 服务。
用户的 Web Server 配置好指定的监听端口 `9000` 和服务启动文件后部署到云端，通过该端口获取 HTTP 请求并进行处理。

使用限制

功能限制

- 目前 Web 函数只支持绑定 API 网关触发器和函数 URL。
- 同一个函数支持绑定多个 API 触发器和函数 URL，但所有 API 都必须在一个 API 服务下。
- 不支持异步调用，不支持重试。
- 在腾讯云标准环境下，仅 `/tmp` 目录可读可写，输出文件时请注意选择 `/tmp` 路径，否则会导致服务因缺少写权限而异常退出。
- 对于 JAVA、Go 等需要打包部署的项目，请保证您的 `scf_bootstrap` 也在 zip 包中一起上传，否则可能导致找不到启动文件。

请求限制

- Web 函数只能通过 API 网关调用，不支持通过函数 API 接口触发。
- 在 `Response headers` 中有以下限制：
  - 所有 key 和 value 的大小不超过4KB。
  - body 的大小不超过6MB。
- 部署您的 Web 服务时，必须监听指定的 `9000` 端口和地址 `0.0.0.0`。
- 目前 HTTP 请求 Header 里的 `Connection` 字段不支持自定义配置。

函数公共请求头

用户的 Web Server 从云函数环境中接收到的公共请求头如下表所示，以下字段均不支持自定义：

Header 字段	描述
X-Scf-Request-Id	当前请求 ID。
X-Scf-Memory	函数实例运行时可使用的最大内存。
X-Scf-Timeout	函数执行的超时时间。
X-Scf-Version	函数版本。
X-Scf-Name	函数名称。
X-Scf-Namespace	函数所在命名空间。
X-Scf-Region	函数所在地域。
X-Scf-Appid	函数所有者的 Appid。
X-Scf-Uin	函数所有者的 Uin。
X-Scf-Session-Token	临时 SESSION TOKEN，函数开启运行角色后会有该字段。
X-Scf-Secret-Id	临时 SECRET ID，函数开启运行角色后会有该字段。
X-Scf-Secret-Key	临时 SECRET KEY，函数开启运行角色后会有该字段。

版权所有：腾讯云计算（北京）有限责任公司

创建及测试函数

最近更新时间：2024-12-10 20:57:22

操作场景

本文介绍如何快速创建一个 Web 函数，您可通过本文了解 Web 函数创建过程及云函数控制台基本操作。

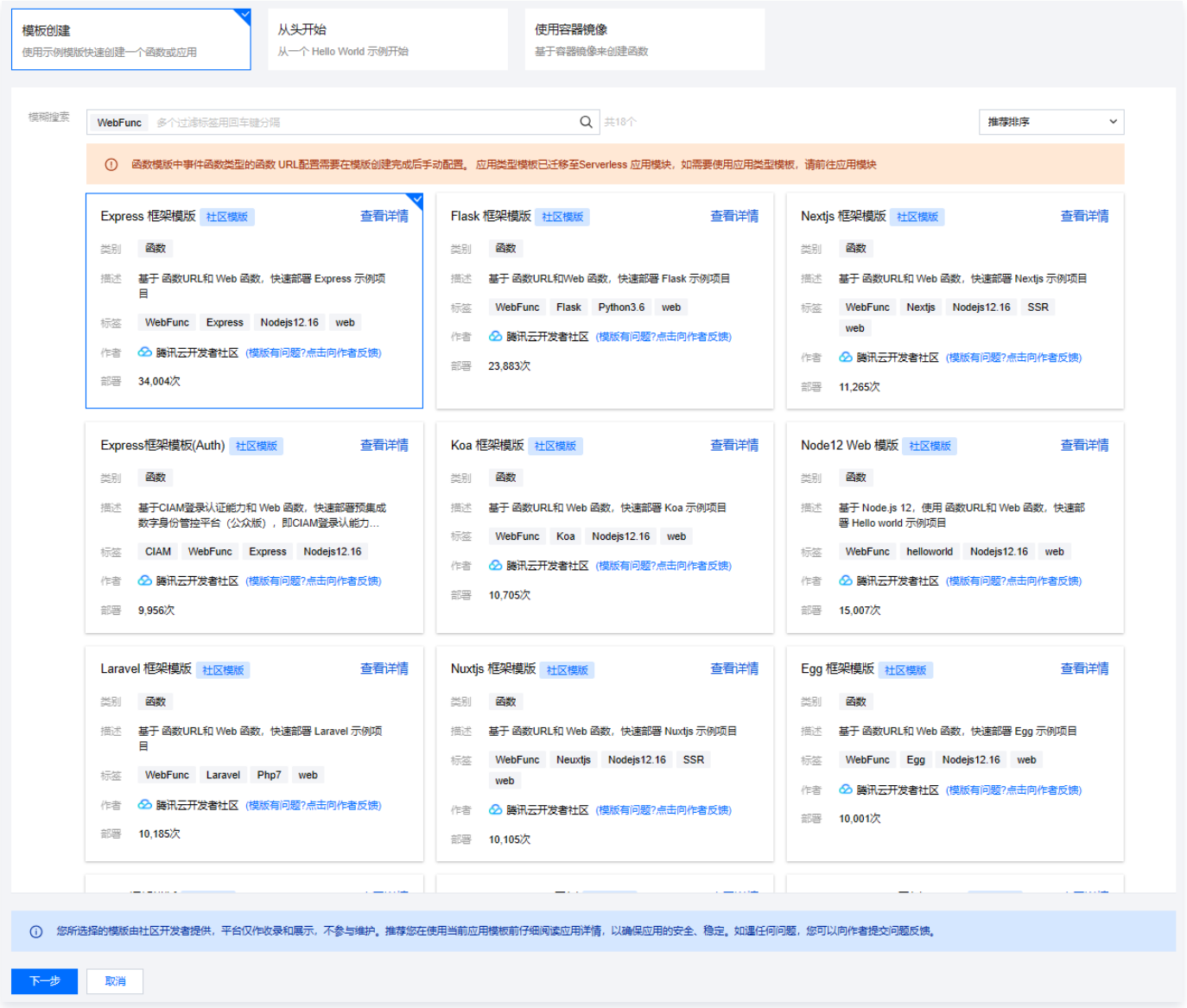
前提条件

在使用腾讯云云函数之前，您需要 注册腾讯云账号 并完成 实名认证 。

操作步骤

通过模板创建函数

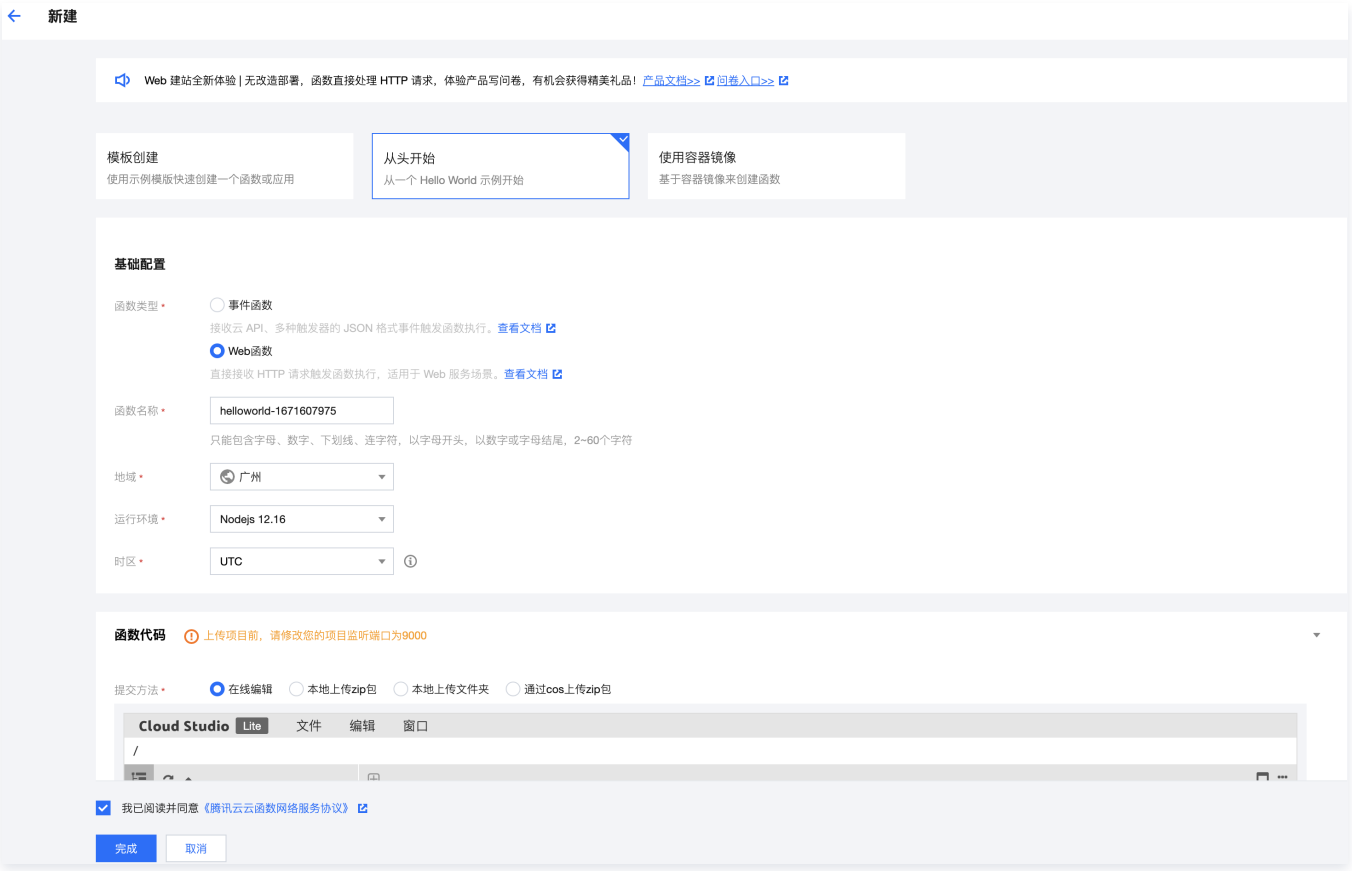
1. 登录 Serverless 控制台，单击左侧导航栏的**函数服务**。
2. 在主界面上方选择期望创建函数的地域，并单击**新建**，进入函数创建流程。
3. 选择使用**模板创建**来新建函数，在搜索框里筛选 `WebFunc` ，筛选所有 Web 函数模板，选择您想使用的模板，单击**下一步**。如下图所示：



4. 在**配置**页面，您可以查看模板项目的具体配置信息并进行修改。
5. 单击**完成**，即可创建函数。
   函数创建完成后，您可在**函数管理**页面，查看 Web 函数的基本信息，并通过 API 网关生成的访问路径 URL 进行访问。

自定义创建函数

1. 登录 Serverless 控制台，单击左侧导航栏的**函数服务**。
2. 在主界面上方选择期望创建函数的地域，并单击**新建**，进入函数创建流程。
3. 选择使用**从头开始**来新建函数，并填写函数基础配置，如下图所示：



   - **函数类型**：选择 “Web 函数”。
   - **函数名称**：填写您自己的函数名称。
   - **地域**：填写您的函数部署地域。
   - **运行环境**：此处以 Nodejs 框架为例，选择 “Nodejs 12.16”。
4. 在**高级配置**中，查看其它必填配置项。
   - **命名空间**：默认为 default，您也可以选择其它空间部署。
   - **启动命令**：对于 Web 函数，您必须为您的项目配置 scf_bootstrap 启动文件，保证 Web Server 在函数环境中可以正常启动。您可以选择 SCF 为您提供的默认框架模板，也可以使用自定义模板，编写您自己的启动命令。详情可参见 启动文件说明。
5. 在**触发器配置**中，触发器目前只支持 API 网关触发和函数 URL，将自动按照默认配置创建触发器。
   - API 网关触发：



- 函数 URL：

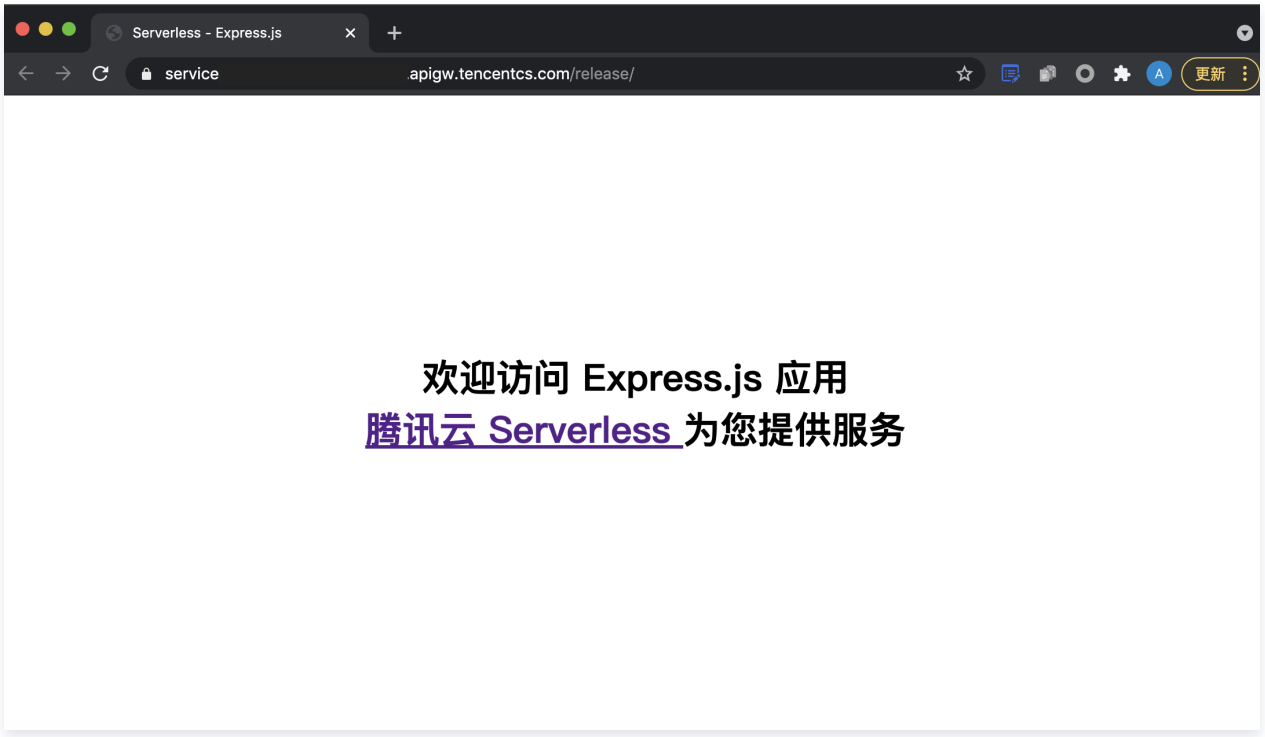


6. 单击**完成**，即可创建函数。
   函数创建完成后，您可在**函数管理**页面，查看 Web 函数的基本信息，并通过 API 网关/ 函数 URL 生成的访问路径 URL 进行访问。

云端测试

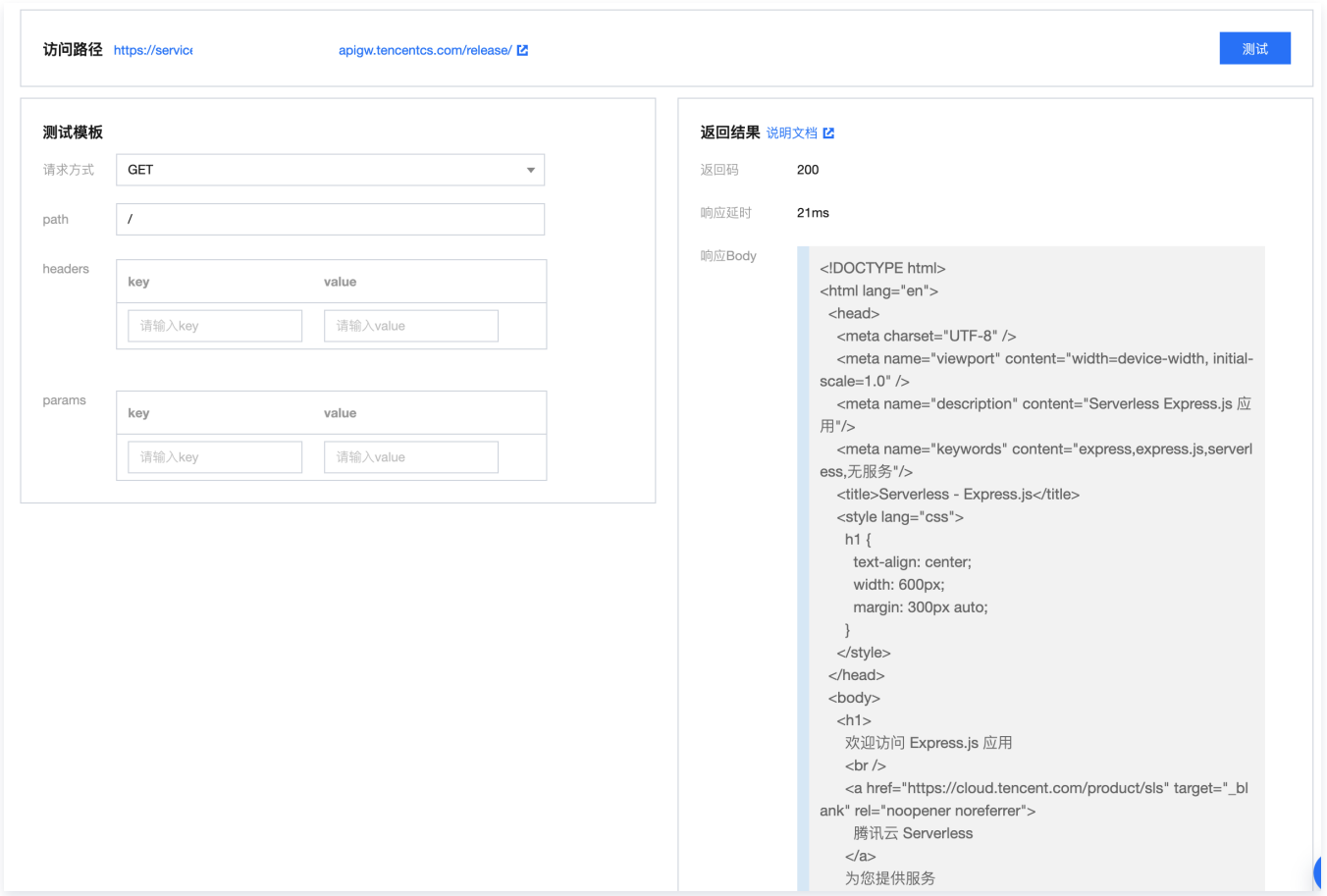
方式1

您可以在浏览器里打开该访问路径 URL，如果可以正常访问，则说明函数创建成功。如下图所示：



方式2

您可以在函数代码页面，通过测试能力，拼装指定的 HTTP 请求进行测试，通过 HTTP 响应结果查看函数是否部署成功。



注意：
控制台测试通过网关 API 接口进行测试调用，如果失败，API 侧会自动执行重试逻辑，最多重试4次，因此您的一次失败请求会看到多条执行日志。

方式3

您可以使用其他 HTTP 测试工具，如 CURL、POSTMAN 等测试您已创建成功的 Web 函数。

查看日志

Web 函数场景下，各个请求的返回 Body 信息不会自动上报到日志，您可以根据自己的开发语言，通过 `console.log()` 或 `print()` 等语句，在代码里自定义上报。
对于 PHP，由于所有的输入会自动作为返回体，您需要执行以下命令，将日志输出到 stdout 中，完成日志上报：

```
<?php
   $stdout = fopen("php://stderr","w");
   fwrite($stdout,"123\n");
?>
```

在已创建函数的详情页面，选择**日志查询**，即可查看函数详细日志。详情可参见 查看运行日志 。

查看监控

在已创建函数的详情页面，选择**监控信息**，即可查看函数调用次数、运行时间等情况。详情可参见 监控指标说明 。

> **注意：**
> 监控统计的粒度最小为1分钟。您需要等待1分钟后，才可查看当次的监控记录。

常见错误码解决方法

常见错误分为用户错误与平台错误两种类型：

- **用户错误**：用户操作不当导致的运行失败，例如发送的请求不符合标准、启动文件命令写错、未监听正确端口、内部业务代码写错等，返回错误码为4xx。
- **平台错误**：由于函数平台内部错误导致的运行失败，错误码为500。

下表描述了请求错误和函数错误可能出现的场景，以便您迅速排查问题。更多错误码详情可参见 云函数状态码 。

2xx状态码

状态码	返回信息	说明
200	Success	函数执行成功，如果看到该返回码，但返回信息与预期不符，请检查您代码逻辑是否正确。

4xx状态码

状态码	返回信息	说明
404	InvalidSubnetID	当函数执行执调用时子网 id 错误时，会有该返回信息，请检查函数的网络配置信息是否正确以及子网 id 是否有效。
405	ContainerStateExitedByUser	容器进程正常退出，请检查您的启动文件是否编写正确。详情请参见 状态码相关问题 。
406	RequestTooLarge	函数调用请求参数体太大时，会有该返回信息，同步请求事件最大为6MB。
407	The HTTP response body exceeds the size limit.	函数返回 Body 过大，超出6MB限制，请调整函数返回值大小后重试。
430	User code exception caught	当用户代码执行出现错误时，会有该返回信息，可以根据控制台的错误日志，查看代码错误堆栈信息，检查代码是否能正常执行。
433	TimeLimitReached	当函数执行时间超出超时配置，会有该返回信息，请检查业务代码是否有大量耗时处理操作，或在函数配置页调整执行超时时间。
439	User process exit when running	当函数执行时用户进程意外退出时，会有该返回信息，可根据返回错误信息查询进程退出原因修复函数代码。
446	PortBindingFailed	未监听指定端口，请检查您的业务代码是否监听 `9000` 端口。
499	kRequestCanceled	用户手动中断请求。

5xx状态码

状态码	返回信息	说明
500	InternalError	内部错误，请稍后重试。若仍无法解决，请联系 在线客服 。

本地调试注意事项

在本地容器调试时，为了保证和云上标准容器环境一致，需注意本地环境内的可读写文件限制。本地容器启动命令可参考如下命令：

> **注意：**
> 此命令仅为参考，请修改为您自己的镜像环境。

```
docker run -ti  --read-only -w /var/user \
    -v /usr/local/cloudfunction/runtime:/var/runtime:ro \
    -v ${PWD}:/var/user:ro \
```

```
    -v /tmp:/tmp \
    -v /usr/local/cloudfunction/runtime:/var/runtime:ro \
    -v /usr/local/cloudfunction/lang:/var/lang:ro \
    ccr.ccs.tencentyun.com/cloudfunc/qcloud-func bash
```

启动文件说明

最近更新时间：2024-11-29 17:29:02

Web 函数基于函数内置的标准语言镜像环境中，您需要创建一个可执行文件 scf_bootstrap 以启动 Web Server，并将该文件和您的代码文件一起打包部署，完成 Web 函数创建。实际处理请求时，您的 Web Server 通过监听指定的 `9000` 端口接收 HTTP 请求，并转发给后端服务完成逻辑处理并返回给用户。

启动文件作用

`scf_bootstrap` 为 Web Server 的启动文件，保证您的 Web 服务正常启动并监听请求。除此之外，您还可以根据需要在 `scf_bootstrap` 中自定义实现更多个性化操作：

- 设定运行时依赖库的路径及环境变量等。
- 加载自定义语言及版本依赖的库文件及扩展程序等，如仍有依赖文件需要实时拉取，可下载至 `/tmp` 目录。
- 解析函数文件，并执行函数调用前所需的全局操作或初始化程序（如开发工具包客户端 HTTP CLIENT 等初始化、数据库连接池创建等），便于调用阶段复用。
- 启动安全、监控等插件。

> ⚠ **注意：**
> - 云函数 SCF 仅支持读取 `scf_bootstrap` 作为启动文件名称，其他名称将无法正常启动服务。
> - 在腾讯云标准环境下，仅 `/tmp` 目录可读可写，输出文件时请注意选择 `/tmp` 路径，否则会导致服务因缺少写权限而异常退出。

使用前提

- 需具有可执行权限，请确保您的 scf_bootstrap 文件具备777或755权限，否则会因为权限不足而无法执行。
- 能够在 SCF 系统环境（CentOS 7.6）中运行。
- 如果启动命令文件是 shell 脚本，第一行需有 `#!/bin/bash`。
- 启动命令必须为绝对路径 `/var/lang/${specific_lang}${version}/bin/${specific_lang}`，否则无法正常调用，详情请参见 标准语言环境绝对路径。
- 建议使用监听地址为 `0.0.0.0`，不可以使用内部回环地址 `127.0.0.1`。
- 结尾必须以 LF 回车结束。

创建方式

本地打包上传

您可以本地编写您的 `scf_bootstrap` 启动文件，确保文件权限满足要求后，和项目代码一起打包部署在 Web 函数上。

控制台快速创建

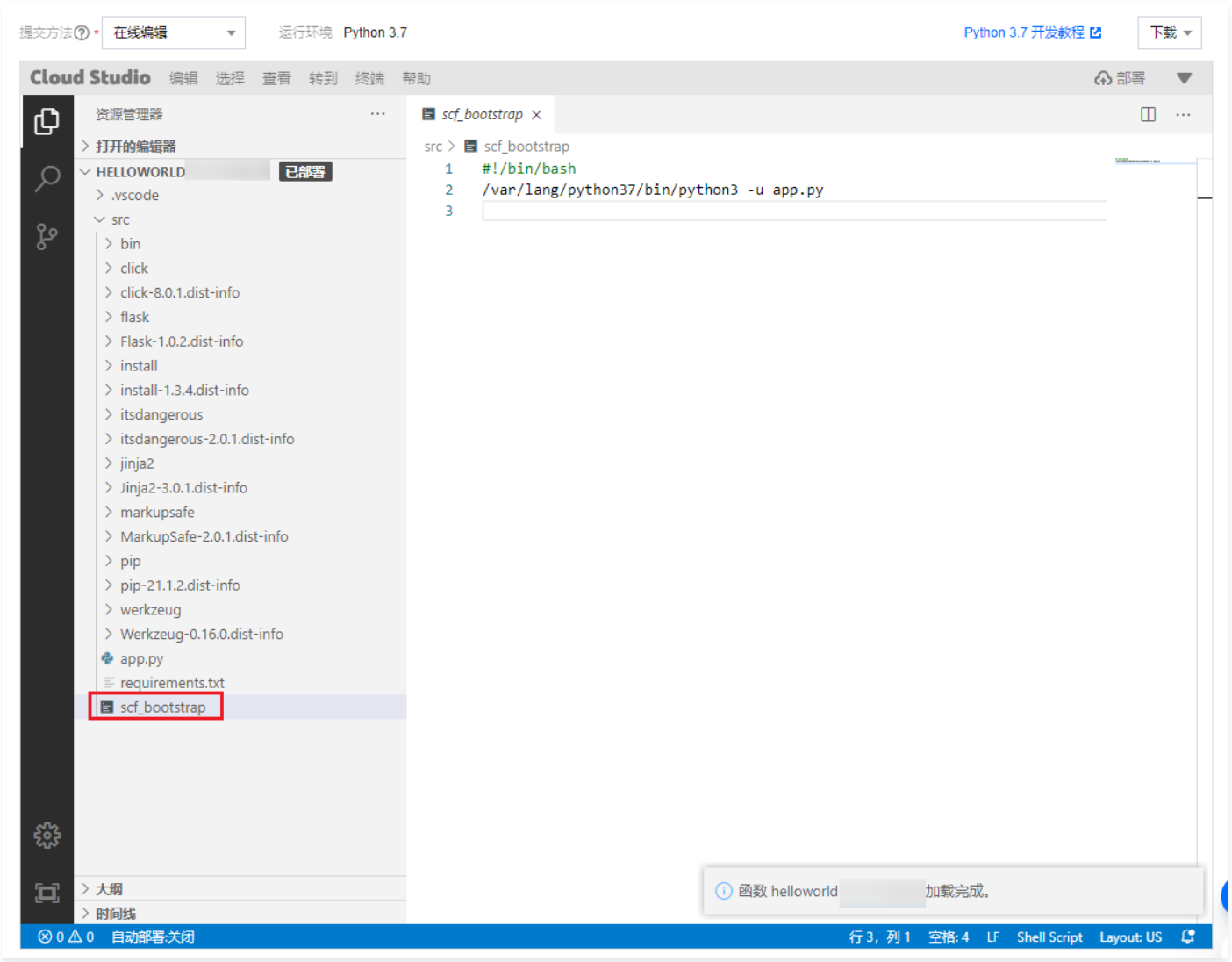
您可以在 Serverless 控制台 中创建 Web 函数。
创建函数 流程中，在**高级配置 > 启动命令**中编辑您的启动文件，云函数 SCF 为常用 Web 框架提供了通用启用模板，您也可以根据实际情况进行修改。如下图所示：



函数创建完成后，控制台将自动把您的代码和 `scf_bootstrap` 一起打包部署。

> **注意：**
> 控制台配置仅在上传的代码里未检测到 `scf_bootstrap` 时生效，如果您的项目里有 `scf_bootstrap` 文件，系统会以项目里的 `scf_bootstrap` 为准进行部署。

部署完成后，您可以在代码编辑器中查看 scf_bootstrap 文件并进行编辑。如下图所示：



常见错误定位

执行文件 scf_bootstrap 作为容器启动命令，必须保证容器可以正常启动运行，执行代码逻辑，因此，请确保您的启动命令写法正确。如遇到 `405` 错误码信息，通常为执行文件无法正常运行导致，请确保您的启动文件写法正确。

标准语言环境绝对路径

语言版本	绝对路径
Node.js 18.15	`/var/lang/node18/bin/node`
Node.js 16.13	`/var/lang/node16/bin/node`
Node.js 14.18	`/var/lang/node14/bin/node`
Node.js 12.16	`/var/lang/node12/bin/node`
Node.js 10.15	`/var/lang/node10/bin/node`
Python 3.10	`/var/lang/python310/bin/python3.10`
Python 3.9	`/var/lang/python39/bin/python3.9`
Python 3.7	`/var/lang/python37/bin/python3.7`

版权所有：腾讯云计算（北京）有限责任公司

Python 3.6	`/var/lang/python3/bin/python3.6`
Python 2.7	`/var/lang/python2/bin/python2`
PHP 8.0	`/var/lang/php80/bin/php`
PHP 7.4	`/var/lang/php74/bin/php`
PHP 7.2	`/var/lang/php7/bin/php`
PHP 5.6	`/var/lang/php5/bin/php`
JAVA 11	`/var/lang/java11/bin/java`
JAVA 8	`/var/lang/java8/bin/java`

常见 Web Server 启动命令模板

Nodejs

```
#!/bin/bash
export PORT=9000
/var/lang/node12/bin/node app.js # 改为您自己的启动函数名
```

Python

```
#!/bin/bash
export PORT=9000
/var/lang/python3/bin/python3 app.py # 改为您自己的启动文件名
```

PHP

```
#!/bin/bash
/var/lang/php7/bin/php -c /var/runtime/php7 -S 0.0.0.0:9000 hello.php # 改为您自己的入口函数名
```

版权所有：腾讯云计算（北京）有限责任公司

触发器管理

最近更新时间：2023-12-07 14:37:11

目前 Web 函数只支持创建 **API 网关触发器**，您可以通过 Serverless 控制台 绑定 API 网关触发器，也可以通过 API 网关控制台 绑定后端函数。

触发器类型说明

对于 Web 函数，触发器支持**默认创建**与**自定义创建**两种创建方式，您可以根据实际情况，选择合适的创建方案：

功能	默认创建（基础型网关：共享型实例）	自定义创建（标准型网关：共享型实例&专享型实例）
提供默认域名	支持	支持
自定义域名绑定	手动绑定	API 网关控制台管理
请求方法配置	支持	支持
发布环境配置	支持	支持
鉴权方法配置	支持	支持
API 网关控制台可见	不可见	可见
API 网关高阶能力（插件/独占实例等）	不支持	支持
计费方式	网关调用次数不计费	按照 API 网关标准计费方案计费
类型转换	可升级为标准型 API 网关，升级后可使用网关全部能力，按照 API 网关标准计费方案计费。	不可转换，标准型网关无法回退为默认创建的基础型网关。

计费方式详情见 Web 函数计费说明 。

触发器介绍

HTTP 类型 API 网关触发器具有以下特点：

- **透传 HTTP 请求**
  API 网关在接受到 HTTP 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，此时 API 网关会将 HTTP 请求直接透转，不再做 event 类型格式转换。HTTP 请求的相关信息包含了例如具体接受到请求的服务和 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。调用类型详情请参见 调用类型 。

触发器配置

- 基础型网关只能通过云函数控制台以默认创建的方式绑定。
- 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。触发器配置详情可参见 API 网关触发器配置 。

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。

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。Web 函数下，API 网关会在 header 里加上函数触发所需要的信息，并将原始请求直接透传，触发后端函数运行。

> ⚠ **注意**
> 以下参数不支持用户自定义配置：
> - connection 字段
> - 以 X-SCF- 开头的自定义字段

Web 函数计费说明

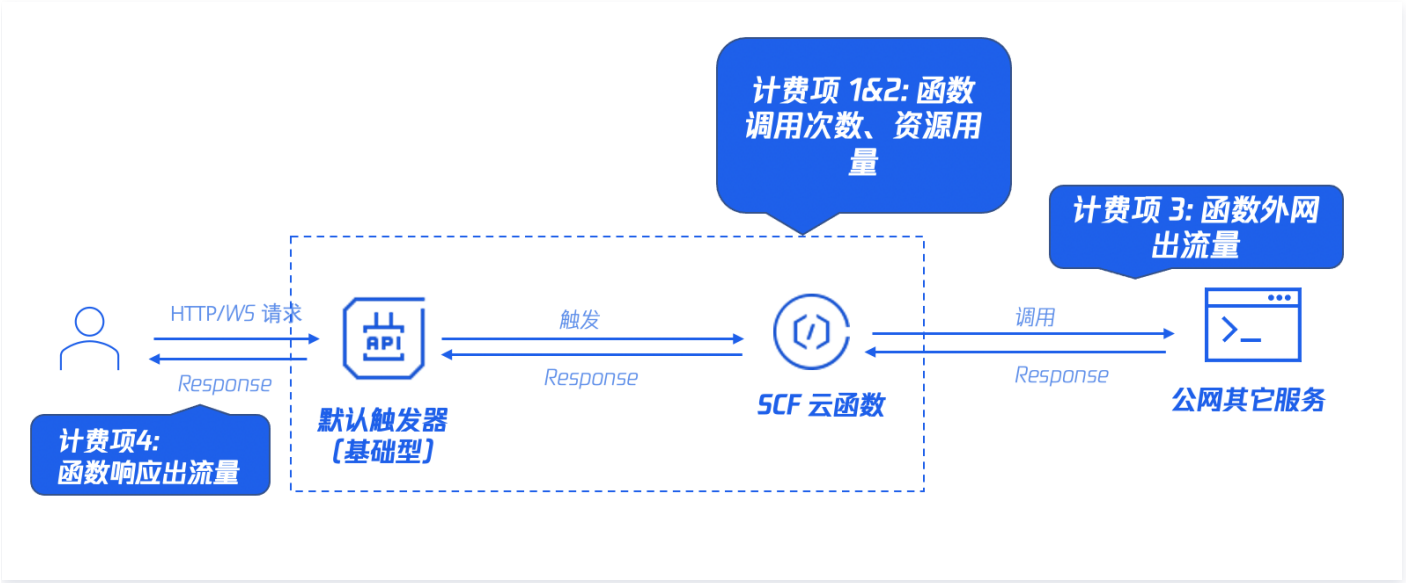
最近更新时间：2023-12-08 14:14:41

对于 Web 函数，提供两种触发器创建方式：默认创建与自定义创建，不同创建方式下，计费逻辑有所不同。

版权所有：腾讯云计算（北京）有限责任公司

默认创建

选择“默认创建”，云函数将自动为您创建一个基础型 API 网关服务触发器（属于共享型实例），该类型触发器只为您提供一个 URL 访问链接，在 API 网关控制台不可见。在该场景下，Web 函数计费方案统计如下：

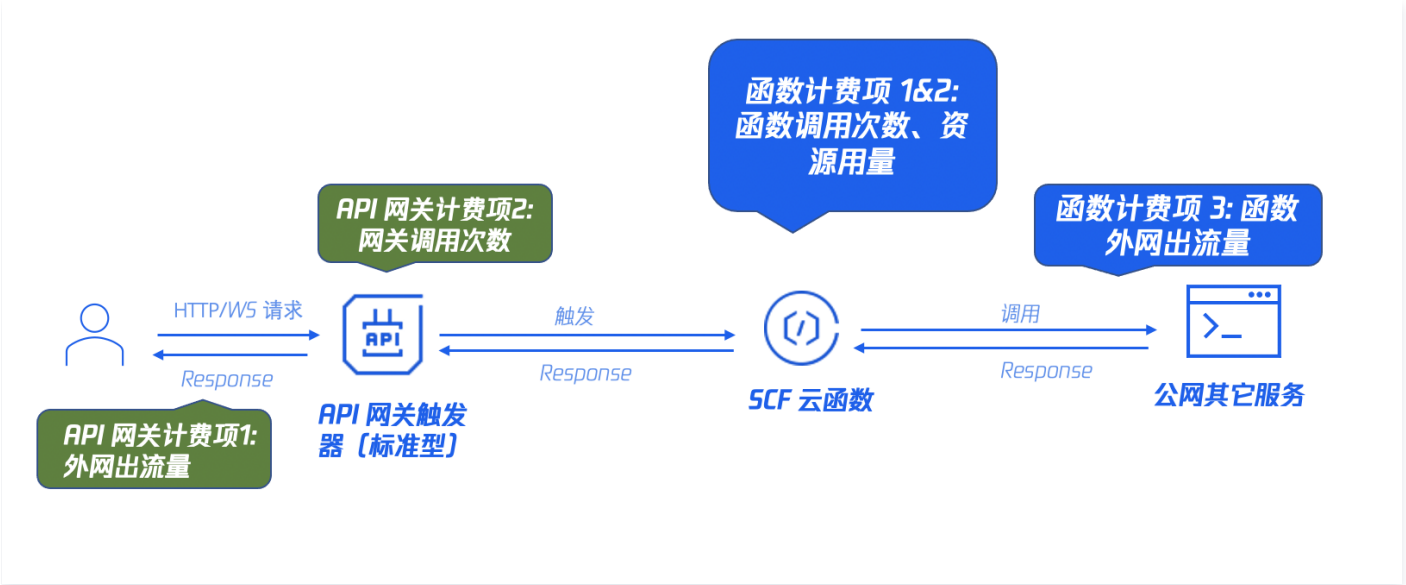


- **触发器侧：** 调用不再计费，出流量转移至函数侧统计。
- **函数侧：** 在标准计费项之外，新增响应出流量计费项。

> **注意**
> 默认创建的为基础型 API 网关（API 网关共享型实例），您可在云函数控制台升级至标准版，升级后可使用网关全部能力，按照 API 网关标准计费方式计费，该升级操作不可回退。

自定义创建

选择“自定义创建”，您需要在函数控制台选择通用标准型API 网关触发器，支持共享型、专享型实例，并绑定已创建的相关服务，支持，在该场景下，计费方案与现有计费方式相同，函数和触发器按照各自的计费规范来统计费用，以标准 API 网关触发器为例，Web 函数计费方案统计如下：



- **触发器侧：** 按照产品本身计费规范统计费用。
- **函数侧：** 按照标准计费项统计（调用次数、资源用量、出流量），响应流量不在函数侧统计。

触发器能力对比

功能	默认创建（基础型网关：共享型实例）	自定义创建（标准型网关：共享型实例&专享型实例）
提供默认域名	支持	支持
自定义域名绑定	手动绑定	API 网关控制台管理
请求方法配置	支持	支持
发布环境配置	支持	支持
鉴权方法配置	不支持	支持
API 网关控制台可见	不可见	可见
API 网关高阶能力（插件/独占实例等）	不支持	支持
计费方式	网关调用次数不计费	按照 API 网关标准计费方案计费
类型转换	可升级为标准型API网关，升级后可使用网关全部能力，按照 API 网关标准计费方案计费	不可转换，标准型网关无法回退为默认创建的基础型网关
后端超时时间	15s，不可修改	可配置

版权所有：腾讯云计算（北京）有限责任公司

命令行部署 Web 函数

最近更新时间：2023-07-20 09:44:57

操作场景

Web 函数是腾讯云云函数 SCF 新支持的函数能力，区别于事件函数（Event Function）对于事件格式的限制，该类型函数专注于优化 Web 服务场景，用户可以直接发送 HTTP 请求到 URL 触发函数执行，详情请参见 函数概述 。
Serverless Cloud Framework SCF 组件现已支持 Web 类型函数部署，您可以通过 SCF 组件，快速创建与部署 Web 函数。

操作步骤

1. 执行以下命令，初始化 Serverless Web 函数模板。

```
scf init scf-nodejs
```

2. 进入示例项目，查看目录结构。示例如下：

```
. http-demo
├── serverless.yml  # 配置文件
├── package.json # 依赖项文件
├── scf_bootstrap # 项目启动文件
└── index.js # 服务函数
```

其中 `scf_bootstrap` 为项目启动文件，具体编写规则请参见 启动文件说明 。

3. 打开 `serverless.yml` ，查看配置信息。
您只需要在 `yml` 里新增 `type` 参数，指定函数类型，即可完成 Web 类型函数部署。

> **注意**
> - 对于 Web 类型函数，无需再指定入口函数。
> - 不填 `type` 参数时，默认为事件型函数。
> - 如果本地代码里无 `scf_bootstrap` 启动文件，您可以在 `yml` 里指定 `entryFile` 参数指定入口函数，组件会根据运行语言，为您生成默认 `scf_bootstrap` 启动文件完成部署。部署完成后，需根据您的实际项目情况，在 云函数控制台 修改 `scf_bootstrap` 文件内容。

示例 yml 如下：

```
component: scf
name: http
inputs:
src:
  src: ./
  exclude:
    - .env
# 指定 SCF 类型为 Web 类型
type: web
name: web-function
region: ap-guangzhou
runtime: Nodejs12.16
# 对于 Node.js，可以支持打开自动安装依赖
installDependency: true
events:
  - apigw:
      parameters:
        protocols:
          - http
          - https
        environment: release
```

```
        endpoints:
          - path: /
            method: ANY
```

4. 在根目录下执行 `scf deploy` 命令，即可完成服务部署。示例如下：

```
$ scf deploy
serverless-cloud-framework
Action: "deploy" - Stage: "dev" - App: "http" - Name: "http"
type:         web
functionName: web-function
description:  This is a function in http application
namespace:    default
runtime:      Nodejs12.16
handler:
memorySize:   128
lastVersion:  $LATEST
traffic:      1
triggers:
  -
    NeedCreate:  true
    created:     true
    serviceId:   service-xxxxxx
    serviceName: serverless
    subDomain:   service-xxxxxx.cd.apigw.tencentcs.com
    protocols:   http&https
    environment: release
    apiList:
      -
        path:            /
        method:          ANY
        apiName:         index
        created:         true
        authType:        NONE
        businessType:    NORMAL
        isBase64Encoded: false
        apiId:           api-xxxxxx
        internalDomain:
        url:             https://service-xxxx.cd.apigw.tencentcs.com/release/
18s › http › 执行成功
```

相关命令

查看访问日志

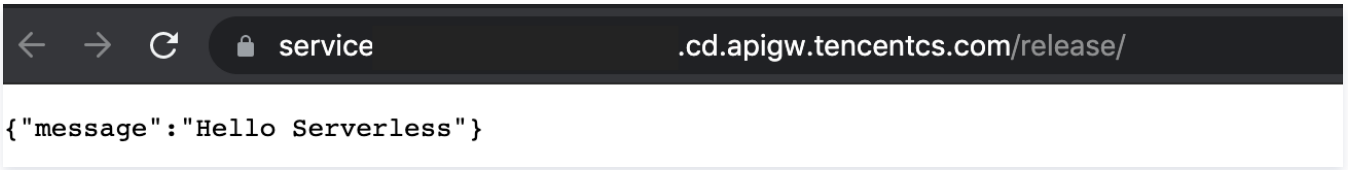
与事件型函数相同，可直接通过 scf log 命令查看部署完成的函数最近10条日志信息。示例如下：

```
$ scf log
serverless-cloud-framework
Action: "log" - Stage: "dev" - App: "http" - Name: "http"
-
  requestId:   xxxxx
  retryNum:    0
  startTime:   1624262955432
  memoryUsage: 0.00
  duration:    0
  message:
    """
-
```

```
  requestId: xxxxx
  retryNum:     0
  startTime:    1624262955432
  memoryUsage: 0.00
  duration:     0
  message:
    """
```

测试服务

- 方案 1：在浏览器直接打开输出的路径 URL，如果可以正常访问，则说明函数创建成功。如下图所示：



- 方案 2：您可以使用其他 HTTP 测试工具，例如 CURL、POSTMAN 等工具测试您已创建成功的 Web 函数。如下示例为通过 CURL 工具测试：

```
curl https://service-xxx.cd.apigw.tencentcs.com/release/
```

删除服务

执行以下命令，即可移除您已部署的云上资源。

```
scf remove
```

Web 框架迁移

Serverless Cloud Framework 还提供了专门针对 Web 框架部署的 HTTP 组件，快速实现 Web 框架部署、创建层、静态资源分离、CDN 加速等功能，使用方式请参见 通过命令行完成框架部署 。

WebSocket 协议支持

最近更新时间：2025-04-24 11:35:41

Web 函数目前已经支持通过原生 WebSocket 协议在客户端和函数运行的服务端间建立连接。

工作原理

服务启动

在支持 WebSocket 协议的 Web 函数运行环境中，通过启动文件启动 WebSocket 服务器，并在**指定端口（9000）**上进行监听。客户端可通过函数 URL 提供的 ws/wss 路径进行连接。

建立 WebSocket 连接

客户端通过函数 URL 提供的 ws/wss 地址发起连接，云函数平台将透传连接至运行环境中的服务进程上。连接协商及通信过程由服务端代码处理。连接建立后，客户端及服务端按 WebSocket 协议进行正常通讯。

WebSocket 连接生命周期

在 Web 函数的 WebSocket 支持下， WebSocket 一次连接的生命周期，等同于一次函数调用请求。WS 连接建立过程等同于请求发起阶段，WS 连接断开等同于请求结束。
同时，函数实例与连接一一对应，即同一实例在某一时刻仅处理一个 WS 连接。在有更多客户端的连接请求发起时，将启动对应数量的实例进行处理。

- 当 WS 连接请求时，函数实例启动，并接受连接建立的请求。
- 当 WS 连接建立后，实例持续运行，根据实际业务情况来接受处理客户端的上行数据，或服务端主动推送下行数据。
- 当 WS 连接中断后，实例停止运行。

连接断开

在如下情况中，WS 连接会中断，且由于请求生命周期与连接生命周期相同，也会使得当次请求运行周期结束：

断开情况	函数表现	函数状态码
客户端或服务端发起连接结束、关闭连接操作，结束状态码为1000、1010（客户端发送）、1011（服务端发送）。	函数正常执行结束，运行状态为成功	200
客户端或服务端发起连接结束、关闭连接操作，结束状态码非1000、1010、1011。	函数异常结束，运行状态为失败	439（服务端关闭） 456（客户端关闭）
在 WS 连接上无消息上行或下行发送，达到配置的空闲超时时间的情况下，连接被函数平台断开。	函数异常结束，运行状态为失败	455
在连接建立后持续使用，函数运行时间达到最大运行时长，连接被函数平台断开。	函数异常结束，运行状态失败	433

- 更多 WebSocket 结束码请参见 WebSocket Status Codes 。
- 更多函数状态码请参见 云函数状态码列表 。

使用限制

使用 WebSocket 时有如下限制：

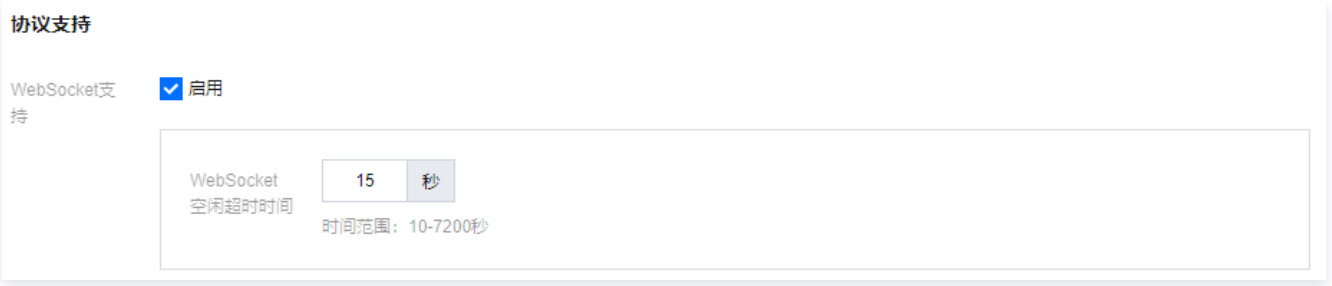
- 空闲超时时间设置：10~7200 秒，函数配置的执行超时时间需要大于等于空闲超时时间。
- 单次请求或返回包最大体积：256KB，可 联系我们 提升配额限制。
- 单连接请求大小限制：128KB/s，可 联系我们 提升配额限制。
- 单连接请求 QPS 限制：10，可 联系我们 提升配额限制。

操作步骤

创建函数

1. 登录 Serverless 控制台，单击左侧导航栏的**函数服务**。
2. 在主界面上方选择期望创建函数的地域和命名空间，并单击**新建**，进入函数创建流程。

3. 选择使用**从头开始**来新建函数，函数类型选择 **Web 函数**。
4. 在**高级配置**中查看协议支持选项。通过勾选 **WebSocket 支持**，配置好 WebSocket 空闲超时时间，来完成 WebSocket协议支持。如下图所示：



5. 在函数 URL配置中，按需勾选公网或者内网访问。



6. 单击完成。函数创建后，可以在函数 URL 页查看到对应的 ws/wss 地址。如下图所示：



> **说明：**
> 在完成创建后，WebSocket 的协议支持不可取消，但可以根据需求修改空闲超时时间配置。

示例代码

目前可以通过如下的 Demo 代码来创建函数，体验 WebSocket 效果：

- Python 示例：使用 websockets 库 实现 WebSocket 服务端。
- Nodejs 示例：使用 ws 库 实现 WebSocket 服务端。

SSE 协议支持

最近更新时间：2024-03-25 15:42:31

SSE（Server-sent Events）是 WebSocket 的一种轻量代替方案，是一种服务器端到客户端（浏览器）的单向流式消息推送协议，在 AI 生成对话等场景下较为常见。Web 函数目前已经支持通过 SSE 协议在客户端和函数运行的服务端间建立连接。

协议启用方式

SSE 协议默认支持，无需在控制台进行任何配置。

SSE 连接生命周期

在 Web 函数的 SSE 支持下， SSE 一次连接的生命周期，等同于一次函数调用请求。函数实例与连接一一对应，即同一实例在某一时刻仅处理一个 SSE 连接。在有更多连接请求发起时，将启动对应数量的实例进行处理。

操作步骤

创建函数

1. 登录 Serverless 控制台，单击左侧导航栏的**函数服务**。
2. 在主界面上方选择期望创建函数的地域和命名空间，并单击**新建**，进入函数创建流程。
3. 在**新建函数**页面，选择使用**从头开始**来新建函数，函数类型选择 **Web 函数**。
4. 本文以运行环境选择 Python 3.7 为例，在**函数代码**中选择**在线编辑**，并将以下 app.py 示例代码复制粘贴至函数代码中：

```
import json
import time

from flask import Flask, Response, stream_with_context

app = Flask(__name__)

@app.route('/stream')
def stream_data():
    msg = ['SSE','empowering','GPT','applications','!','Happy','chatting','!']
    # 可以使用 yield 逐字返回内容
    def generate_response_data():
        for i,word in enumerate(msg):
            json_data = json.dumps(
                {'id': i, 'content': word})
            yield f"data:{json_data}\n\n"
            time.sleep(1)
    return Response(stream_with_context(generate_response_data()), mimetype="text/event-stream")

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=9000)
```

5. 单击**完成**。函数创建完成后，可以在函数列表中查看。

测试函数

可以在本地终端使用 curl 工具发起 SSE 连接，示例命令如下：

```
curl -v -H 'Accept:text/event-stream' {函数暴露的API网关地址}/stream
```

返回内容如下：

```
> GET /release/stream HTTP/1.1
> Host: XXXXXXXXXXXXXXX.XX.apigw.tencentcs.com
> User-Agent: curl/8.0.1
```

```
> Accept: */*
> 'Accept:text/event-stream'
>
< HTTP/1.1 200 OK
< Content-Type: text/event-stream; charset=utf-8
< Transfer-Encoding: chunked
< Connection: keep-alive
< X-Api-RequestId: 22ad36c38536ee65bd07c44cb5311e1d
< Vary: Accept-Encoding
<
data:{"id": 0, "content": "SSE"}

data:{"id": 1, "content": "empowering"}

data:{"id": 2, "content": "GPT"}

data:{"id": 3, "content": "applications"}

data:{"id": 4, "content": "!"}

data:{"id": 5, "content": "Happy"}

data:{"id": 6, "content": "chatting"}

data:{"id": 7, "content": "!"}

* Connection #0 to host XXXXXXXXXXXXXXXXXX.XX.apigw.tencentcs.com left intact
```

Web 函数请求并发管理

最近更新时间：2022-06-01 11:35:05

请求并发概述

请求单并发

默认情况下，在调用函数时，云函数会分配一个并发实例处理请求或事件。函数代码运行完毕返回后，该实例会处理其他请求。如果在请求到来时，所有实例都在运行中，云函数则会分配一个新的并发实例。一个并发实例同一时刻仅处理一个事件的运行逻辑，保障每个事件的处理效率和稳定性。

请求多并发

在大多数情况下，请求单并发都是值得推荐使用的模式，无需在写代码时考虑多个请求同时处理带来的典型并发难题，例如线程安全、阻塞调用、异常处理等。而在 Web 应用中，典型的业务场景是 IO 密集型——函数内访问数据库或其他系统的接口等下游服务，会有较多时间在等待这些下游服务响应。这种等待一般都是在做 iowait，不消耗 CPU，此时，如果开启了请求多并发，让一个实例可以同时处理多个请求，则可以更充分利用单个实例的 CPU 资源。
Web 函数目前已经支持 开启请求多并发 配置，您可以根据业务需要进行启用和配置。请求多并发支持**自定义静态并发**、**智能动态并发**两种模式。

- **自定义静态并发**
  启用后，当同时有多个请求，将不超过指定并发值的请求调度到同一函数实例内执行。并发增多，将增加函数实例的CPU、内存等消耗，建议配合压力测试进行合理设置，避免函数执行异常。目前支持的并发范围为 2~100 并发。
- **智能动态并发**
  启用后，在函数实例负载允许的情况下，智能动态调度更多请求到同一函数实例内运行。将于后续推出，敬请期待。

请求并发优势

- 在 IO 密集型场景中，如 Websocket 长连接业务，可减少计费执行时长，节省费用。
- 多个请求并发在同一个实例中可复用数据库连接池，减缓下游服务压力。
- 请求并发密集时，多个请求只需要一个实例进行处理，无需拉起多个实例，从而降低实例冷启动几率，降低响应延迟。

操作步骤

开启请求多并发

1. 登录云函数控制台，选择左侧导航栏中的 函数服务 。
2. 在“函数服务”列表页面，选择需进行配置的 Web 函数名。
3. 在“函数管理”页面中，选择**函数配置**。
4. 在“函数配置”页面中，单击“编辑”，进入编辑模式。
5. 在“请求多并发”中，勾选“启用”，开启请求多并发模式，在弹出的“自定义静态并发”下方的输入框中输入需要的并发值。如下图所示：

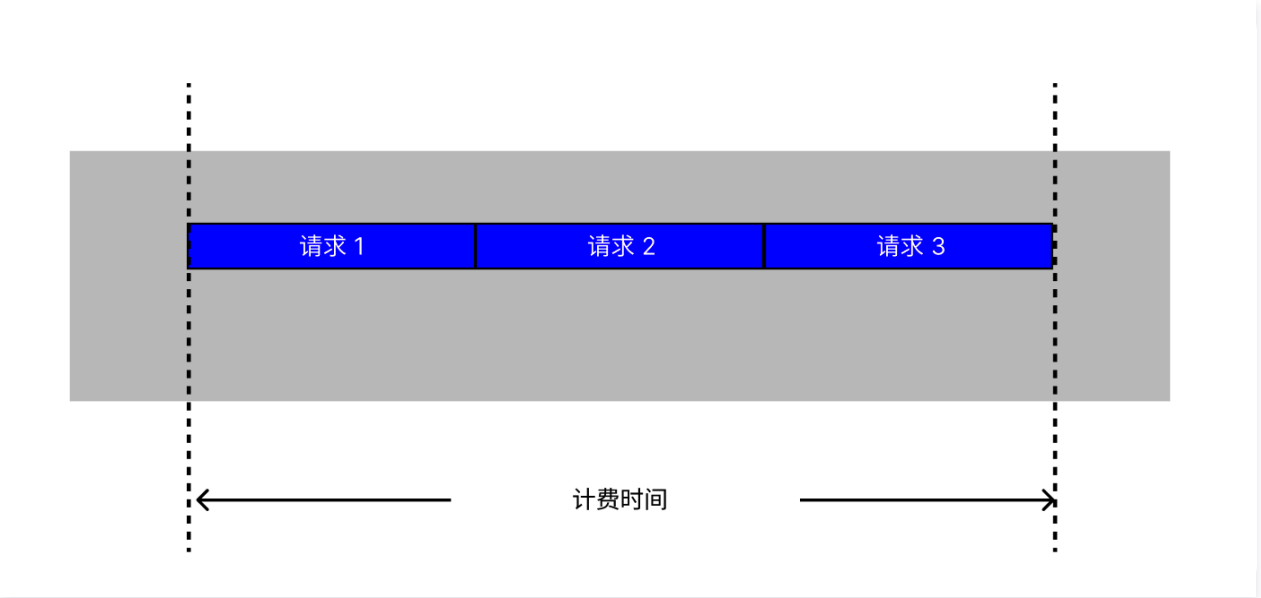


6. 单击“保存”完成配置。

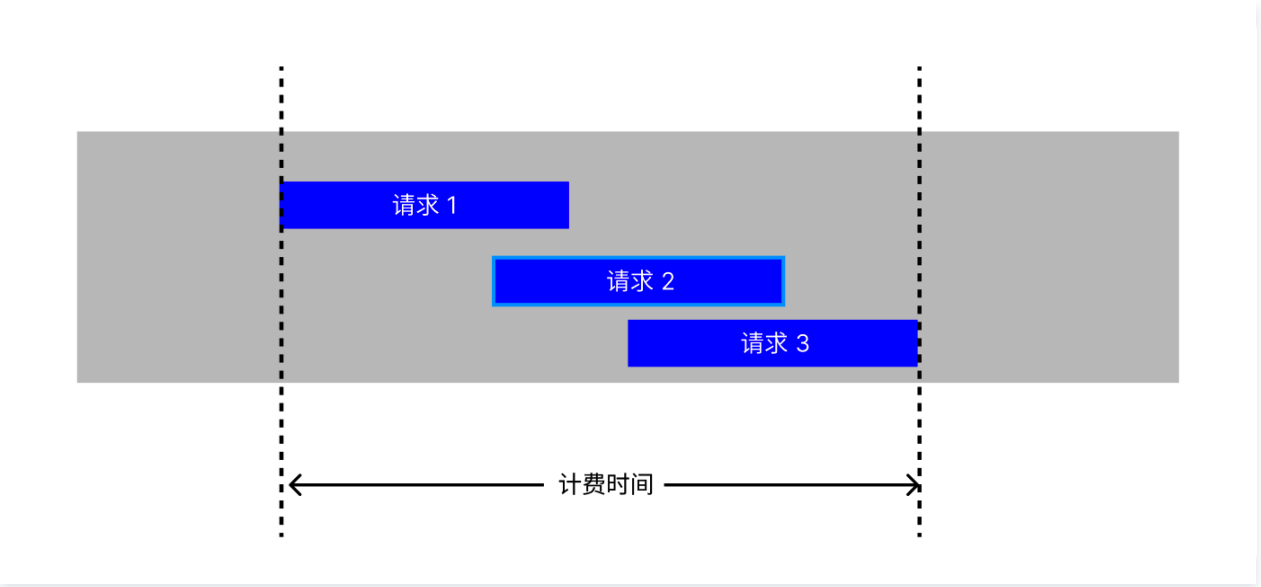
注意事项

计费

未开启请求多并发时，单个函数实例一次只会处理一个请求，第一个请求处理完成才会开始处理下一个请求，内存时间的计费时长是每个请求的执行时长的加和，如下图所示：



开启请求多并发之后，单个函数实例一次会处理多个并发请求，第一个请求未结束时，如果第二个请求进来，则会有一段时间两个请求同时在处理，此时，交叠的这段时间只会计算一次。如下图所示：



其他计费项保持不变，详情见 计费概述 。

日志

开启请求多并发后，由于多个并发请求同时处理，每个请求产生的日志在流式上报时，可能会出现日志和 RequestID 无法一一对应。此时，应该在代码中正确设置 logger，将 RequestID 打印到日志中，以解决该问题。RequestID 从 Web 函数中接收到的公共请求头里的 `X-Scf-Request-Id` 字段（部分框架为 `x-scf-request-id` ）获取。

NodeJS 示例代码

```
let WebSocketServer = require('ws').Server;
let wss = new WebSocketServer({ port: 9000 });

wss.on('connection', function connection(ws) {

  let requestID = ws.upgradeReq.headers['x-scf-request-id'];
  console.log('requestID: %s', requestID);

  ws.on('message', function incoming(message) {
```

```
    console.log('requestID: %s', requestID);
    console.log('received: %s', message);
  });

});
```

超限错误

内存超限

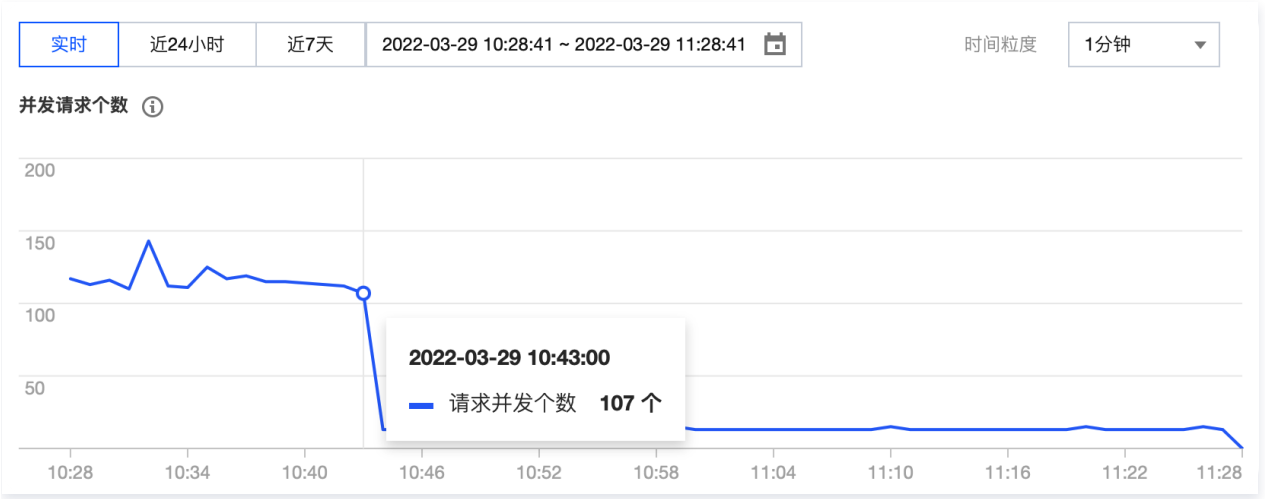
请求多并发会增加内存超限的概率，在内存超限 OOM 发生时，实例会进行重启，此时，实例内正在处理的多个请求会同时出现 abort 中断错误，错误码为 434 MemoryLimitReached。请在设置并发值之前，先对函数进行压测以确定安全的并发值，以避免内存超限带来的影响。

超时

请求耗时过长，在配置的执行超时时间范围内没有执行完成时，会终止该请求，向客户端返回错误码 433 TimeLimitReached。实例内的其他正在进行中的请求不受影响。

监控

开启请求多并发后，在监控页面会出现“并发请求个数”面板，可直观看到指定时间段内的请求并发情况。



监控常见问题

对一段时间没有调用的函数发起并发请求，会出现虽然并发请求数量未超过设定的并发值，但监控“并发实例个数和预置并发”面板中显示的并发实例个数大于 1 的情况。这是因为一段时间没有调用，函数实例回收资源，此时发起请求，会出现冷启动，为保障及时响应处理进来的请求，此时会并发拉起函数实例，直到第一个实例可以正常接受请求为止。如果是普通的 HTTP 请求，过一段时间之后，新请求会集中在若干个函数实例上进行处理，其余的函数实例会在请求处理结束后逐步下线，监控中的并发实例数恢复正常。如果是 WebSocket 连接，则在连接未断开之前，并发实例数都会维持在一开始拉起的数量。通过配置动态预置避免冷启动，可以减少这类问题发生的概率。

日志管理
日志检索教程

最近更新时间：2022-12-15 18:19:58

操作场景

云函数 SCF 于2021年01月29日起进行日志服务升级，接入腾讯云日志服务 CLS，在此日期前创建的函数正在按地域逐渐进行迁移，详情可参见 云函数日志服务变更说明 。
云函数 SCF 向2021年01月29日之后的新增函数及迁移完成的函数默认提供两种日志查询方式：

- **控制台检索：** Serverless 控制台 内嵌日志服务 CLS 检索分析页面，支持关键词搜索，您可以在云函数控制台日志高级检索页面使用查询语法组合关键词进行检索。
- **API 检索：** 您可通过调用日志服务 CLS 搜索日志 接口查询函数调用日志。

> **注意**
> - SCF 写入 CLS 的日志结构请参见 日志结构说明 。
> - 若您的函数于2021年1月29日前创建且尚未进行迁移，如需使用更多日志分析功能，则请参见 日志投递配置（旧） 将函数调用日志投递到日志服务 CLS 使用。

控制台检索

参考以下步骤对函数使用日志高级检索功能：

1. 登录 Serverless 控制台，选择左侧导航栏中的**函数服务**。
2. 在“函数服务”列表页面，选择需检索日志的函数名，进入函数详情页。
3. 选择**日志查询 > 高级检索**。在“高级检索”页中，您可使用关键词进行搜索，或使用查询语法组合关键词进行检索。**语法规则详情可参见** 日志检索语法与规则 。
4. 配置检索内容后，单击右侧的**检索分析**即可查询检索结果。

API 检索

下文介绍 CLS 搜索日志 API 与 SCF 相关参数如何获取和填充。

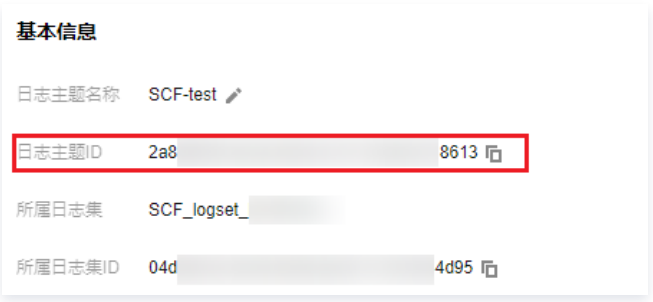
TopicId

`TopicId` 为函数日志投递的 CLS 日志主题 ID。获取 `TopicId` 操作步骤如下：

1. 登录 Serverless 控制台，选择左侧导航栏中的**函数服务**。
2. 在“函数服务”列表页面，选择需检索日志的函数名，进入函数详情页。
3. 在“函数配置”页查看“日志配置”。如下图所示：



4. 单击**日志主题**对应的链接，跳转到日志服务控制台后，可以获取日志主题 ID。如下图所示：



> **说明**
> 您也可以通过调用 获取函数详细信息 API 取 `ClsTopicId` 获取 `Topicid` 。

Query

> **注意**
> - 返回结果的各字段含义请参见 日志结构说明 。
> - SCF 调用日志为实时上报，如果请求存在但返回值为空，可能为函数尚未执行完成，该条日志没有上报到 CLS 导致，请在函数执行完成后重试检索。

获取某函数全部调用日志

以函数名称为 `hello-scf` ，所在命名空间为 `default` 为例：

```
SCF_Namespace:"default" AND SCF_FunctionName:"hello-scf"
```

获取某个请求（RequestId）调用日志

以请求为 `09c346d3-8417-49c5-8569-xxxxxxxxxxxx` 为例：

```
SCF_RequestId:"09c346d3-8417-49c5-8569-xxxxxxxxxxxx"
```

获取某个请求（RequestId）的执行结果

执行结果包括请求开始时间、执行结果、执行耗时、内存消耗、日志等级，以请求为 `09c346d3-8417-49c5-8569-xxxxxxxxxxxx` 为例：

```
SCF_RequestId:"09c346d3-8417-49c5-8569-xxxxxxxxxxxx" AND SCF_Type:Platform AND SCF_Message:Report*
```

获取某个请求（RequestId）的返回值

以请求为 `09c346d3-8417-49c5-8569-xxxxxxxxxxxx` 为例：

```
SCF_RequestId:"09c346d3-8417-49c5-8569-xxxxxxxxxxxx" AND SCF_Type:Platform AND SCF_Message:Response*
```

> **注意**
> - SCF 日志接入 CLS 后，依然为函数保留返回数据。返回数据将写入 CLS 的 `SCF_Message` 字段中，格式为 `Response RequestId:xxx RetMsg:xxx` 。
> - SCF_Message 的值长度限制为8KB，超出时将截取前8KB。

获取一段时间内函数的请求列表

以函数名称为 `hello-scf` ，所在命名空间为 `default` ，查询别名为 `$DEFAULT` ，查询版本为1为例：

- 获取由于内部错误导致的失败请求列表：

```
SCF_Type:Platform AND SCF_Message:Report* | select SCF_RequestId as requestId, SCF_RetryNum as
retryNum,SCF_StartTime as startTime where SCF_FunctionName='hello-scf' and SCF_Namespace='default' and
SCF_Qualifier='1' and SCF_Alias:'$DEFAULT' and SCF_StatusCode = 500 order by startTime desc
```

- **获取执行时间超过3秒的请求列表：**

```
SCF_Type:Platform AND SCF_Message:Report* | select SCF_RequestId as requestId, SCF_RetryNum as
retryNum,SCF_StartTime as startTime where SCF_FunctionName='hello-scf' and SCF_Namespace='default' and
SCF_Qualifier='1' and SCF_Alias:'$DEFAULT' and SCF_Duration>3000 order by startTime desc
```

日志结构说明

最近更新时间：2025-01-03 14:58:23

云函数 SCF 日志按条写入日志服务 CLS，每次请求的日志由多条组成，每条日志均为固定的 `键:值` 格式。

单条日志键值格式

默认格式

默认格式下每条日志均由 14 个固定的 `键:值` 组成，如下表所示：

字段名称	字段类型	字段含义
SCF_FunctionName	text	函数名称。
SCF_Namespace	text	函数所在命名空间。
SCF_StartTime	long	调用开始时间。
SCF_LogTime	long	日志产生时间。
SCF_RequestId	text	请求 ID。
SCF_Duration	long	函数运行时间（单位：毫秒）。
SCF_Alias	text	别名。
SCF_Qualifier	text	版本。
SCF_MemUsage	double	函数运行内存。
SCF_Level	text	Log4J 日志级别，默认为 INFO。
SCF_Message	text	日志内容。
SCF_Type	text	日志类型，Platform 指平台日志，Custom 指用户日志。
SCF_StatusCode	long	函数运行 状态码，202 表示请求状态为运行中。
SCF_RetryNum	long	重试次数。

精简格式

精简格式键值组成相对默认格式进行了删减，仅保留了日志查询场景下必须依赖的字段。精简格式下区分用户日志及平台日志，不同类型的日志对应不同的格式，如下表所示：

用户日志

用户日志为用户在代码中的标准输出，平台会对标准输出进行捕捉并上报到日志服务 CLS。

字段名称	字段类型	字段含义
SCF_FunctionName	text	函数名称。
SCF_Namespace	text	函数所在命名空间。
SCF_LogTime	long	日志产生时间。
SCF_RequestId	text	请求 ID。
SCF_Alias	text	别名。
SCF_Qualifier	text	版本。
SCF_Message	text	日志内容。

SCF_Type	text	日志类型，Platform 指平台日志，Custom 指用户日志。
SCF_RetryNum	long	重试次数。

平台日志

平台日志为在每次请求的开始和结束时打印的日志的内容，用于标记请求的开始、结束和记录执行情况，不支持取消。
平台日志有两种键值组成，记录请求执行情况的 `report` 日志键值组成为下表1，其他平台日志（如 START、END、ERROR、Response）键值组成为下表2。

表1

字段名称	字段类型	字段含义
SCF_FunctionName	text	函数名称。
SCF_Namespace	text	函数所在命名空间。
SCF_StartTime	long	调用开始时间。
SCF_LogTime	long	日志产生时间。
SCF_RequestId	text	请求 ID。
SCF_Duration	long	函数运行时间（单位：毫秒）。
SCF_Alias	text	别名。
SCF_Qualifier	text	版本。
SCF_MemUsage	double	函数运行内存。
SCF_Level	text	Log4J 日志级别，默认为 INFO。
SCF_Message	text	日志内容。
SCF_Type	text	日志类型，Platform 指平台日志，Custom 指用户日志。
SCF_StatusCode	long	函数运行 状态码 。
SCF_RetryNum	long	重试次数。

表2

字段名称	字段类型	字段含义
SCF_FunctionName	text	函数名称。
SCF_Namespace	text	函数所在命名空间。
SCF_StartTime	long	调用开始时间。
SCF_LogTime	long	日志产生时间。
SCF_RequestId	text	请求 ID。
SCF_Alias	text	别名。
SCF_Qualifier	text	版本。
SCF_Message	text	日志内容。

SCF_Type	text	日志类型，Platform 指平台日志，Custom 指用户日志。
SCF_RetryNum	long	重试次数。

日志格式选择与切换

1. 登录 云函数控制台，在左侧选择**函数服务**。
2. 在**函数服务**页面上方，选择函数的地域和命名空间，并在页面中单击期望切换日志格式的函数名，进入该函数的详情页面。
3. 在**函数管理**页面中，选择**函数配置** > **日志配置** > **日志格式**，进行日志格式的选择和切换。如下图所示：



注意：

- 日志配置为函数级，配置更新后对 `$LATEST` 版本和已发布版本均立即生效。
- 获取函数运行日志 接口暂不支持精简格式日志查询，请使用 CLS 搜索日志接口 。如果已经使用 API 对日志键值进行处理，请详细阅读不同日志格式下的字段调整情况后谨慎操作。
- 精简格式相对于默认格式可降低日志产生的费用，但不再提供实时展示函数执行期间运行耗时、运行内存等信息。
- 精简格式仅支持非镜像部署的事件函数。

单次请求日志结构

SCF 单次请求日志结构分为调用日志和预置日志两种。

调用日志结构

SCF 调用日志以平台日志标记请求开始、请求结束、请求错误信息、函数返回信息以及请求执行情况，用户日志封装在请求开始至请求结束之间。日志结构如下（表格中仅展示 `SCF_Message` 字段示例）：

SCF_Message	日志类型	内容含义
START RequestId:09c346d3-8417-49c5-8569-xxxxxxxxxxxx	平台日志	标记请求开始。
init log	用户日志	用户在函数初始化阶段打印的日志内容，容器仅在冷启动场景下会执行初始化逻辑，非冷启动场景下无初始化日志输出。
Init Report RequestId: 09c346d3-8417-49c5-8569-xxxxxxxxxxxx Coldstart: 236ms (PullCode: 70ms InitRuntime: 8ms InitFunction: 158ms) Memory: 640MB MemUsage: 57.86MB	平台日志	初始化执行情况日志，Coldstart 为初始化阶段总耗时，其中 PullCode 为初始化阶段拉取用户函数和层代码耗时或拉取镜像耗时，InitRuntime 为初始化阶段平台耗时，InitFunction 为初始化阶段用户代码执行耗时，Memory 为函数配置内存，MemUsage 为初始化阶段运行内存。容器仅在冷启动场景下会执行初始化逻辑，非冷启动场景下无初始化日志输出。
invoke log	用户日志	用户在函数调用阶段打印的日志内容。
ERROR RequestId:09c346d3-8417-49c5-8569-xxxxxxxxxxxx Result:xxx	平台日志	函数错误原因，函数执行正常时无 ERROR 日志。

Response RequestId:09c346d3-8417-49c5-8569-xxxxxxxxxxxx RetMsg:"Hello World"	平台日志	函数返回信息记录在 RetMsg 中。
END RequestId:09c346d3-8417-49c5-8569-xxxxxxxxxxxx	平台日志	标记请求结束。
Report RequestId:09c346d3-8417-49c5-8569-c55033b17f51 Duration:1ms Memory:128MB MemUsage:29.734375MB	平台日志	函数调用执行情况日志，Duration 为函数执行耗时，Memory 为函数配置内存，MemUsage 为函数执行阶段运行内存。

以 `Python` 运行环境的 Web 函数，打印一行初始化日志及一行调用日志为例，代码如下：

```
from flask import Flask
app = Flask(__name__)
print("init log")

@app.route('/')
def hello_world():
   return 'Hello World'

if __name__ == '__main__':
   app.run(host='0.0.0.0',port=9000)
```

输出日志结构如下（仅展示 `SCF_Message` 字段内容）：

```
START RequestId:09c346d3-8417-49c5-8569-xxxxxxxxxxxx
 * Serving Flask app "app" (lazy loading)
 * Environment: production
 WARNING: Do not use the development server in a production environment.
 Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:9000/ (Press CTRL+C to quit)
init log
Init Report RequestId:09c346d3-8417-49c5-8569-xxxxxxxxxxxx Coldstart: 640ms (PullCode: 119ms
InitRuntime: 2ms InitFunction: 519ms) Memory: 640MB MemUsage: 5.21MB
Hello world
Response RequestId:09c346d3-8417-49c5-8569-xxxxxxxxxxxx RetMsg:"Hello World"
END RequestId:09c346d3-8417-49c5-8569-xxxxxxxxxxxx
Report RequestId:09c346d3-8417-49c5-8569-xxxxxxxxxxxx Duration:1ms Memory:128MB MemUsage:29.734375MB
```

预置日志结构

SCF 预置日志以用户打印日志开始，以平台日志标记预置结束，日志结构如下（表格中仅展示 `SCF_Message` 字段示例）：

SCF_Message	日志类型	内容含义
provision log	用户日志	用户在函数初始化阶段打印的日志内容，在预置实例场景下会被记录在日志中。
ERROR RequestId:09c346d3-8417-49c5-8569-xxxxxxxxxxxx Result:xxx	平台日志	预置实例失败原因，预置实例成功时无 ERROR 日志。
Provisioned Report RequestId: c6af0fb4-1c07-4a92-8307-xxxxxxxxxxxx Coldstart: 640ms (PullCode: 119ms InitRuntime: 2ms	平台日志	预置实例执行情况日志，Coldstart 为预置实例总耗时。其中 PullCode 为预置实例过程中拉取用户函数和层代码耗时或拉取镜像耗时，InitRuntime 为预置实例过程中平台耗时，InitFunction 为预置

InitFunction: 519ms) Memory: 640MB MemUsage: 5.14MB		实例过程中用户代码执行耗时。Memory 为函数配置内存，MemUsage 为预置阶段运行内存。该日志仅在预置场景下会输出。

以 `Python` 运行环境的 Web 函数，打印一行初始化日志及一行调用日志为例，代码如下：

```
from flask import Flask
app = Flask(__name__)
print("init log")

@app.route('/')
def hello_world():
   return 'Hello World'

if __name__ == '__main__':
   app.run(host='0.0.0.0',port=9000)
```

预置实例场景下输出的日志结构如下（仅展示 `SCF_Message` 字段内容）：

```
* Serving Flask app "app" (lazy loading)
* Environment: production
WARNING: Do not use the development server in a production environment.
Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:9000/ (Press CTRL+C to quit)
init log
Previsioned Report RequestId:09c346d3-8417-49c5-8569-xxxxxxxxxxxx Coldstart: 640ms (PullCode: 119ms
InitRuntime: 2ms InitFunction: 519ms) Memory: 640MB MemUsage: 5.21MB
```

日志输出方法

当函数输出的单行日志为 JSON 格式时，JSON 内容将被解析，并在投递至日志服务时按字段:值的方式进行投递。JSON 内容的解析仅能解析第一层，更多的嵌套结构将作为值进行记录。
您可以执行以下代码进行测试：

```
# -*- coding: utf8 -*-
import json

def main_handler(event, context):
    print(json.dumps({"key1": "test value 1","key2": "test value 2"}))
    return("Hello World!")
```

日志投递配置

最近更新时间：2024-11-26 18:15:42

> **说明：**
> 若您的函数于2021年1月29日前创建且尚未进行迁移，如需使用更多日志分析功能，则请参见 日志投递配置（旧），将函数调用日志投递到日志服务 CLS 使用。

云函数 SCF 于2021年1月29日起全量接入腾讯云 日志服务 CLS，在此之后创建的函数调用日志将投递至 CLS，并支持日志实时输出，在此日期前创建的函数正在按地域逐渐进行迁移，详情可参见 云函数日志服务变更说明 。
本文介绍云函数 SCF 所提供的 默认投递 和 自定义投递 两种日志服务投递方式及其配置方法。

权限说明

为保证日志正常查看，子账号下至少拥有日志服务 CLS 只读权限 `QcloudCLSReadOnlyAccess` 。主账号为子账号授权方法请参见 授权管理 。

限制说明

函数调用日志投递至日志服务的限制如下：

- 每个请求5秒内打印的日志量上限为1MB。
- 每个请求5秒内打印的日志条数上限为5000条。
- 每条日志长度上限为8KB，超出将截取前8KB。

其他限制请参见 日志服务规格说明 ，请关注日志服务配置是否能够满足业务需求，超限可能会导致日志写入失败。

操作步骤

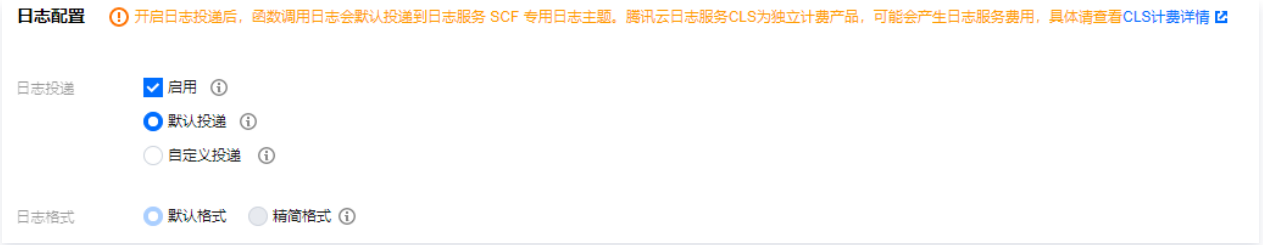
默认投递

新建函数时，如不指定日志投递主题，将会使用默认投递日志能力。默认投递日志时，SCF 将会为您开通日志服务并将函数调用日志投递至 SCF 专用日志集下的日志主题中，SCF 专用日志集和日志主题分别以 `SCF_logset` 和 `SCF_logtopic` 为前缀命名，如不存在将自动创建。函数调用日志默认保留7天，您可在 日志服务控制台 查看及管理。

> **注意：**
> - 日志服务为独立计费产品，SCF 专用日志主题会占用日志服务免费额度，详情可参见 日志服务计费详情 。
> - 为保证 SCF 控制台日志正常展示，SCF 专用日志主题不建议修改索引配置，关闭索引会导致该主题下所有日志查询检索功能不可用。如需自定义日志索引配置，请参考下文 自定义投递 配置函数日志主题。

配置日志服务

1. 登录 Serverless 控制台，选择左侧导航栏中的 函数服务 。
2. 在主界面上方选择期望创建函数的地域和命名空间，并单击**新建**，进入函数创建流程。
3. 在日志配置中，选择**默认投递**。如下图所示：



4. 单击**完成**即可创建函数，并完成函数日志默认投递。您可在**函数管理** > **函数配置**中查看日志配置。如下图所示：



查看和管理日志服务

您可单击函数配置中“日志配置”的日志集 ID，前往 日志服务控制台 查看和管理日志。SCF 专用日志集在日志服务控制台已用 `SCF` 字样进行标记，如有日志持久化存储、投递或消费、对日志内容进行监控告警等需要，均可在日志服务控制台完成配置。

自定义投递

新建函数时，如需指定函数调用日志投递主题，可选择使用日志自定义投递能力。在使用日志自定义投递能力之前，需保证账号已经开通 日志服务 。

创建日志集和日志主题

登录 日志服务控制台 并 创建日志主题 。本文以在广州创建 `SCF-test` 日志主题为例。如下图所示：



> **注意：**
> 日志集地域请选择函数服务所在地域，暂不支持跨地域日志推送。

配置日志服务

1. 登录 Serverless 控制台，选择左侧导航栏中的 函数服务 。
2. 在主界面上方选择期望创建函数的地域，并单击**新建**，进入函数创建流程。
3. 在日志配置中，选择**自定义投递**，并选择已为该函数创建的日志主题，本文以 `SCF-test` 为例。如下图所示：

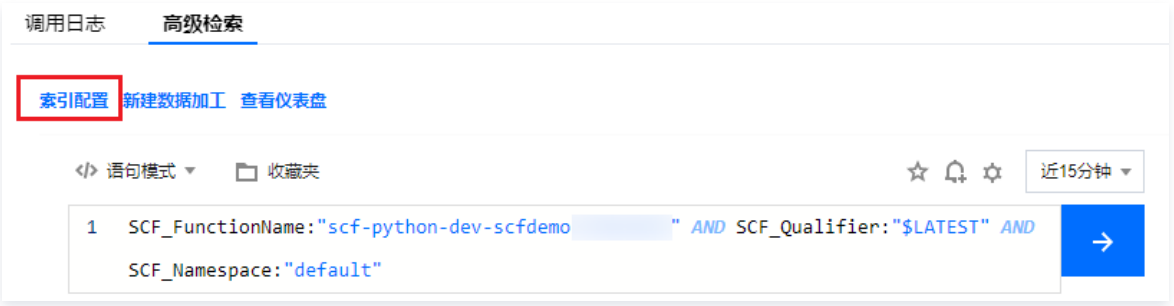


4. 单击**保存**即可。

索引配置

日志检索依赖日志主题的索引配置，在函数创建时，SCF 会自动为您完成索引配置。如遇索引异常无法正常查看日志，请参考如下步骤配置索引：

1. 登录 Serverless 控制台，选择左侧导航栏中的 函数服务 。
2. 在函数服务列表中，选择日志索引异常的函数名，进入**函数管理**页面。
3. 在日志查询页签中，选择**高级检索**中的**索引配置**。如下图所示：



4. 在**索引配置**中，开启**索引状态**和**键值索引**。如下图所示：



该配置方法仅对日志主题中已有函数调用日志的场景有效，日志主题中无函数调用日志，请参照下表手动配置**键值索引**。

字段名称	字段类型	字段含义
SCF_FunctionName	text	函数名称。

SCF_Namespace	text	函数所在命名空间。
SCF_StartTime	long	调用开始时间。
SCF_LogTime	long	日志产生时间。
SCF_RequestId	text	请求 ID。
SCF_Duration	long	函数运行时间。
SCF_Alias	text	别名。
SCF_Qualifier	text	版本。
SCF_MemUsage	double	函数运行内存。
SCF_Level	text	Log4J 日志级别，默认为 INFO。
SCF_Message	text	日志内容。
SCF_Type	text	日志类型，Platform 指平台日志，Custom 指用户日志。
SCF_StatusCode	long	函数运行 状态码 。
SCF_RetryNum	long	重试次数。

为保证云函数控制台日志展示效果，请在键值索引配置中为字段打开“开启统计”能力。

5. 完成索引配置后单击**确定**保存。

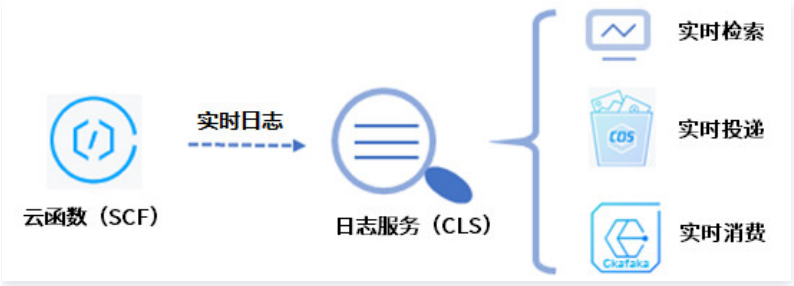
日志投递配置（旧）

最近更新时间：2024-11-29 17:29:02

> **说明：**
> 云函数 SCF 于2021年01月29日起全量接入腾讯云 日志服务 CLS，在此之后创建的函数调用日志将默认投递至 CLS，且支持日志实时输出。若您的函数于2021年1月29日前创建，且需进行日志检索与日志投递，请参考本文档使用该功能。

操作场景

在使用云函数 SCF 进行函数计算时，会产生大量的函数运行日志，如果您需要将日志进行持久化存储、投递或消费，对日志内容进行监控告警，您可将日志投递到腾讯云日志服务 CLS 平台。如下图所示：



前提条件

在使用云函数实时日志服务功能之前，需开通 日志服务 。

> **说明：**
> 了解日志服务相关限制可参见 规格说明，超出限制可能会导致日志丢失。

操作步骤

创建日志主题

登录 日志服务控制台 并 创建日志主题 。本文以在广州创建 `SCF-test` 日志主题为例。

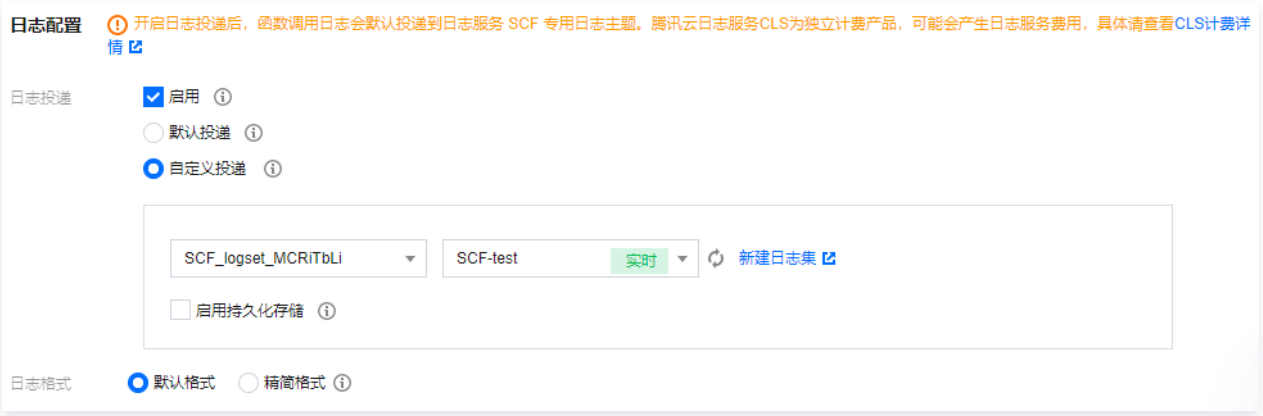
> **注意：**
> 日志集地域请选择函数服务所在地域，暂不支持跨地域日志推送。

配置日志服务

1. 登录 Serverless 控制台，选择左侧导航栏中的 函数服务 。
2. 在页面上方选择函数所在地域及命名空间，并在列表中单击需实时采集日志的函数名。
3. 在函数配置页面，单击右上角的**编辑**。如下图所示：



4. 在“日志投递”中，勾选“启用”并选择已为该函数创建的日志集和日志主题，本文以 `SCF-test` 为例。如下图所示：



5. 单击**保存**即可成功接入日志服务平台。

开启索引

日志检索依赖日志主题的索引配置，函数关联日志主题后，SCF 自动为日志主题配置索引。如仍遇索引异常导致日志拉取失败，请参考此步骤调整索引配置。

1. 登录日志服务控制台，选择左侧导航栏中的 日志主题 。
2. 单击已创建的日志主题 ID，进入“基本信息”页面。
3. 选择日志主题所在行右侧的**管理**，进入日志主题“基本信息”页面。
4. 在日志主题“基本信息”页面，单击**索引配置**。如下图所示：



5. 单击右上角的**编辑**，开启“键值索引”后按照下表添加“字段名称”、“字段类型”。

> **说明：**

对于配置了日志服务的函数，为保证云函数控制台日志展示效果，请在**键值索引**配置中为字段打开“开启统计”能力。如下图所示：



字段名称	字段类型	字段含义
SCF_FunctionName	text	函数名称。
SCF_Namespace	text	函数所在命名空间。
SCF_StartTime	long	调用开始时间。
SCF_LogTime	long	日志产生时间。
SCF_RequestId	text	请求 ID。
SCF_Duration	long	函数运行时间。
SCF_Alias	text	别名。
SCF_Qualifier	text	版本。
SCF_MemUsage	double	函数运行内存。
SCF_Level	text	Log4J 日志级别，默认为 INFO。
SCF_Message	text	日志内容。
SCF_Type	text	日志类型，Platform 指平台日志，Custom 指用户日志。
SCF_StatusCode	long	函数运行 状态码 。
SCF_RetryNum	long	重试次数。

如需使用更多功能，例如日志实时检索、日志投递和消费等，请参考 日志服务文档 并前往 日志服务控制台 开始使用。

版权所有：腾讯云计算（北京）有限责任公司

并发管理
并发概述

最近更新时间：2025-03-24 15:57:42

并发是云函数在某个时刻同时处理的请求数。在业务其他服务可以支撑的情况下，您可以通过简单的配置实现云函数从几个并发到数以万计并发的拓展。

并发运行原理

在调用函数时，云函数会分配一个并发实例处理请求或事件。函数代码运行完毕返回后，该实例会处理其他请求。如果在请求到来时，所有实例都在运行中，云函数则会分配一个新的并发实例。
云函数遵循一个并发实例同一时刻仅处理一个事件的运行逻辑，保障每个事件的处理效率和稳定性。

异步调用的并发处理

异步事件会先进入云函数平台的队列，并按照先进先出的原则依次处理异步请求。异步请求会根据队列长度、当前函数并发数等情况，选择合适的并发处理方式，拉起足够处理事件的并发实例依次处理事件。
若异步调用失败，云函数平台将遵循一定的规则重试，详情可参见 异步调用重试策略 。

同步调用的并发处理

同步事件到达云函数平台后，平台将会检测是否有空闲并发实例。如有，则事件将会立刻发送给空闲并发实例进行处理。如没有，则平台将会启动新的并发实例来处理事件。
若同步调用失败，您需要自行重试。

并发的计算

云函数的并发指的是函数代码同时处理请求或调用的数量，您可以通过以下公式估算：
并发数 = 请求速率 × 函数运行时间 = 每秒请求次数 × 每个请求的平均耗时
您可以在监控信息中的“运行时间”看到每个请求的平均耗时。
例如：某业务 QPS 为2000，每个请求的平均耗时为0.02s，则每个时刻的并发数为 2000qps × 0.02s = 40

并发实例复用与回收

当并发实例处理完事件请求后，不会立刻被回收，而是会保留一段时间以便复用。在保留期内，如有新的请求事件需要处理，将会优先使用保留中的并发实例，从而实现事件的快速处理，无需新启动并发实例。
保留期过后，如果没有请求需要该实例处理，云函数平台则会回收该实例。对于低并发的场景，不再设置保留期，平台将启动智能回收机制进行回收。
并发保留的时间由云函数平台根据情况动态调整，故函数业务代码中不能假设某个特定保留时间进行程序编写。

并发扩容

如果请求到来时，没有该版本的并发实例可以处理该请求，云函数平台会启动新的并发实例来处理。新启动的并发实例在初始化的过程后，便可以处理事件，我们称之为由弹性并发带来的扩容，而弹性并发的扩容速度上限即**函数 burst**。
在地域维度，每个账号的**弹性并发的扩容速度上限（函数 burst）默认限制为500个/分钟**，即该地域下所有函数在1分钟内累计最多可以启动500个新的并发实例。如在1分钟内已经达到了当前限制，则将无法再启动新的并发实例，持续到下1分钟。在此期间有新的并发扩容请求，将会产生扩容受限错误（429 ResourceLimit），详情可参见 云函数状态码 。
例如，广州地域的账号默认并发额度可以支撑128MB函数的1000个并发实例。有大量请求到来时，第一分钟可以从0个并发实例启动到500个并发实例。如果还有请求需要处理，第二分钟可以从500个并发实例启动到1000个并发实例，直至并发实例可以满足请求的需要或达到并发上限。
目前500个/分钟的函数 burst 可以满足多数业务场景。如果您的业务遇到该扩容速度的限制或者需要增加命名空间级函数 burst的管理能力，您可以选择使用预置并发进行预热或购买 函数套餐包 提高函数 burst 限制。

预置并发

云函数平台弹性并发扩容的并发实例需要经历初始化的过程：包括运行环境初始化及业务代码初始化等过程。
您可以使用**预置并发**功能，预先配置并发实例。**云函数平台将在您配置后开始启动并发实例，同时不会主动回收预置并发的实例，尽可能地保障有相应数量的并发实例**。如遇到并发实例因代码内存泄漏等错误，云函数平台会将其替换为新的实例。详情可参见 预置并发 。

并发服务承诺

并发扩容限制

并发扩容限制	默认限制	额外可申请
弹性扩容并发速度限制（函数 burst）	500并发/分钟	支持万级并发扩容速度，可通过购买函数套餐包提升配额。
预置扩容并发速度限制（预置 burst）	100并发/分钟	根据业务情况自动调整预置并发的启动速度。

在地域维度，用户弹性并发的扩容速度默认限制为500并发/分钟。例如客户有5w并发的诉求，需要5w/ 500 =100分钟就能完成扩容操作，如需提升函数 burst 可直接购买 函数套餐包 。云函数平台会根据您业务的情况调整预置并发的启动速度，预置扩容并发速度平均为100并发/分钟。

函数并发配额

云函数平台默认提供函数粒度的并发管理能力供您灵活控制不同函数的并发情况。每个账号在地域维度有总并发额度的限制，不同地域配置的函数并发配额并不相同，详情见表格。如果您需要提升各项配额或者增加命名空间维度的并发配额管理可直接购买 函数套餐包 。

	地域	默认配额	额外可申请
函数并发配额	广州、上海、北京、成都、中国香港	128,000MB	百万MB级并发配额，可通过购买函数套餐包提升配额。
	新加坡、东京、硅谷、法兰克福、深圳金融、上海金融	64,000MB	

并发管理

云函数平台提供函数粒度的并发管理能力，详情可参见 并发管理 。

并发内存与并发数

为了您更准确的进行并发管理，云函数的并发配额按照内存为单位计算。例如256MB的并发配额代表着1个256MB内存的并发实例，或者是2个128MB内存的并发实例。

最大独占配额

当您为一个函数设置最大独占配额，将会有以下两个效果：

- 最大独占配额是**此函数的并发额度上限**，所有版本的并发额度加和小于等于最大独占配额。
- 并发额度划给该函数后为**函数独享**，不再提供给其它函数。

并发监控

函数的并发在处理实际请求时，该并发会被标记为执行并发。从云函数的监控中，可以查询到函数、函数某个具体版本、别名的执行并发。由于执行并发的采集有一定的时间间隔，如函数的执行时间很短而并发较高，则当前监控可能会有一定偏差。

使用场景

通过综合使用最大独占配额、预置并发等能力，您可以灵活调配多个函数间的资源用量情况，并按需预热函数。

共享配额

在未进行各项配置的情况下，各函数默认共享使用账号额度。如果有某个函数产生了突增业务调用，可以充分利用空闲未使用的额度，来保证突增不会引起函数的调用并发超限。

并发保障

如果有某个函数的业务功能比较敏感或关键，需要尽力保障请求的成功率，此时可以使用最大独占配额。最大独占配额可以给到函数独享额度，确保并发的可靠性，不会由于多函数的争抢导致的调用并发超限。

预置并发

在函数对冷启动敏感、或代码初始化流程耗时较长、加载库较多的情况下，可以通过设置具体函数版本的预置并发，预先启动函数实例来保障运行。

并发管理体系

最近更新时间：2025-06-13 15:09:02

云函数平台提供函数粒度的并发管理能力，供您灵活控制不同函数的并发情况。

并发管理体系

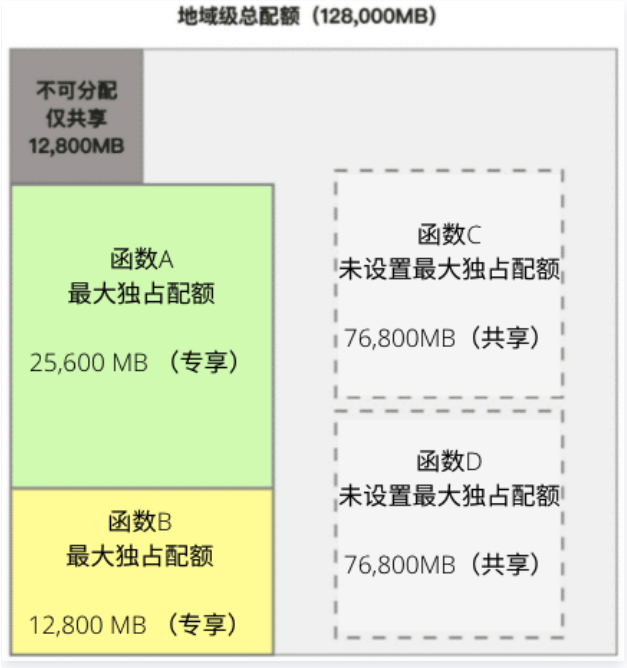
目前云函数有两个层次的并发管理能力，分别是账号并发额度和函数的最大独占配额。

```
账号级并发额度
  |- 函数级最大独占配额
```

> **说明：**
> 预置并发 不在并发管理能力中，仅作为预先启动实例的能力。同一个函数下的版本共享该函数的并发。

账号级并发额度

每个账号在地域维度有总并发额度限制，默认为 128,000 MB 或 64,000MB，详情可参见 配额限制 。地域间的并发额度相互独立，彼此不受影响。
默认情况下，一个地域下的所有函数共享账号级并发额度，即在某一具体时刻，所有函数处于运行状态的实际并发额之和，最大可以达到账号并发额度。超出并发额度的请求将遇到超限错误（432 ResourceLimitReached），可以通过 购买套餐包 来进一步提升账号配额。
您可以使用函数的 最大独占配额能力，将地域维度并发分配至某个函数上，从而实现对函数的并发进行管理。为了避免账号级额度全部分配后，未设置最大独占配额的函数无法调用，云函数平台将 12,800MB 的账号级并发额度限制为不可分配、仅供未配置保留的函数使用。如下图所示：



最大独占配额

最大独占配额是函数维度上的并发管理能力。当您为一个函数设置最大独占配额，将会有以下两个效果：

- **最大独占配额是此函数的并发额度上限**，所有版本的并发额度加和小于等于最大独占配额。
- 并发额度划给该函数后为**函数独享**，不再提供给其它函数。

最大独占配额是函数并发额度的上限，您可以通过该能力进行函数并发的管理，费用的管控，避免出现费用失控的情况。同时，您也可以通过将函数**最大独占配额设置为0来实现对函数关停**的操作。所有针对该函数的请求，都会出现并发超限的错误。
设置函数保留会占用地域级的并发额度。若地域级未占用额度（地域级额度 − 分配给其他函数的最大独占配额 − 12,800MB）不足，则无法设置。

设置最大独占配额

参考以下步骤可以针对函数设定期望的最大独占配额额度。

1. 登录 Serverless 控制台，选择左侧导航栏中的**函数服务**。
2. 在“函数服务”列表页面，选择需进行配置的函数名，进入“函数管理”页面。
3. 选择左侧**并发配额**，在“最大独占配额”中，单击右上角**设置**。
4. 在弹出的“设置函数最大独占配额”窗口中，设置期望的最大独占配额，单击**提交**即可。如下图所示：



设置完成后，您可在“并发管理”页面的“最大独占配额”页中查看配置状态。

删除最大独占配额

当您不再计划使用最大独占配额时，可进行删除操作。删除最大独占配额后，函数将与其他函数共享账号维度的并发额度。

> **说明：**
> 删除最大独占配额与最大独占配额为0是不同的配置。
> - 删除最大独占配额：函数没有专享额度，使用地域下的共享额度，上限由共享额度的使用情况而定。
> - 最大独占配额为0：函数专享额度为0，函数并发上限为0，函数无法运行，停止对触发事件的响应。

1. 登录 Serverless 控制台，选择左侧导航栏中的**函数服务**。
2. 在“函数服务”列表页面，选择需进行配置的函数名，进入“函数管理”页面。
3. 选择左侧**并发配额**，在“最大独占配额”中，单击页面右侧的**删除**。
4. 在弹出的“删除函数最大独占配额”窗口中单击**确认**即可。

预置并发

最近更新时间：2022-12-22 16:48:48

预置并发支持并发实例按配置预先启动，同时云函数平台不会主动回收这些实例，会尽可能地保障有相应数量的可以处理请求的并发实例。
您可通过此功能，为函数的指定版本设定预置并发额度。通过配置预置并发，可预先进行计算资源的准备，降低冷启动、运行环境初始化及业务代码初始化引起的耗时。

概述

预置并发是在版本维度上解决请求到来时遇到并发实例初始化的问题。当您为一个函数版本配置预置并发之后，将会有以下效果：
1. 云函数平台**立刻开始启动并发实例**，直至达到配置值。
2. 云函数平台**不会主动回收预置并发实例**，同时会尽可能地保障预置并发实例数。

预置并发与弹性调用的并发实例启动速度是分开的，预置并发的启动不会占用地域维度500个/分钟的弹性扩容速度。云函数平台会根据您业务的情况调整预置并发的启动速度，默认为100个/分钟。
云函数平台不会主动回收预置并发实例，但并发实例可能由于进程退出、内存超限等问题不可用。一旦有不可用的实例，云函数平台会回收同时准备新的并发实例以达到预置并发实例的配置。期间可能出现短暂的实际并发实例数小于预置并发实例的情况，未启动的并发实例不会纳入计费范围。您可以在函数的监控信息“并发执行个数和预置并发”图中查看预置并发启动情况。
预置并发**只能配置在已发布的版本上，无法配置在 `$LATEST` 版本上。** `$LATEST` 版本处于可编辑态，而预置并发需要在请求到来前启动并发实例。为了保障业务的稳定，避免因代码和配置编辑带来的版本不一致问题，预置并发只能配置在已发布的版本上。已发布版本的代码和配置无法修改，适合生产环境使用，详情可参见 版本管理 。

预置并发与并发管理

预置并发可以帮您解决函数初始化时间过长的问题，更快地进行响应请求。需注意的是，预置并发不是并发管理体系中的能力，设置预置并发不会对函数可以处理的并发上限有影响。每个函数可以并发处理的请求量完全依赖于函数最大独占配额或地域级并发额度。
以配置内存为128MB的某个函数版本为例：

业务场景	业务平均并发	预置并发	函数最大独占配额	效果
默认情况	100并发	未配置	未配置	所有第一次处理请求的并发实例有初始化过程。函数并发额度受账号其他函数的影响，可能会有超限。
函数关停	100并发	未配置	0MB（0并发）	最大独占配额为0并发，函数关停，所有请求均超限错误。
无需预置并发	100并发	未配置	19,200 MB（150并发）	所有第一次处理请求的并发实例有初始化过程。可以保障150并发，超过会有并发超限错误。
80%预置	100并发	10,240 MB（80并发）	19,200 MB（150并发）	持续有80个并发实例的无需初始化，20个并发实例第一次调用需要初始化。可以保障150并发，超过会有并发超限错误。
100%预置	100并发	12,800 MB（100并发）	19,200 MB（150并发）	持续有100个并发实例的无需初始化，超出的并发实例第一次调用需要初始化。可以保障150并发，超过会有并发超限错误。
完全预置	100并发	19,200 MB（150并发）	19,200 MB（150并发）	所有并发实例无需初始化。可以保障150并发，超过会有并发超限错误。
超额预置	100并发	25,600MB（200并发）	19,200 MB（150并发）	和完全预置相比，多了50个预置并发的费用，其他相同。

并发管理体系（地域级并发额度、函数最大独占配额）负责同一时间内可以处理的请求量，预置并发负责保障有可以处理请求的并发实例。两者解耦的关系可以实现 无初始化过程的流量切换 等能力。

预置并发限制

预置并发配置额受限于账号维度的额度，即地域下所有函数所有版本的**总预置并发额度小于等于账号维度的并发配额**。

操作步骤

新增预置并发

针对函数已发布的版本，可以设定期望数量的预置并发数。

> ⚠ 注意
> 函数进行 发布版本 操作后，才能针对版本进行预置设置。

1. 登录 Serverless 控制台，单击左侧导航栏的函数服务。
2. 在“函数服务”列表页面，单击目标函数名，进入“函数管理”页面。
3. 选择左侧**并发配额**，在“预置并发”页面中，单击**新增预置并发**。
4. 在弹出的“新增函数预置并发”窗口中，选择期望版本及预置并发数，单击**提交**即可。
   设置完成后，您可在“预置并发”中查看配置的状态。云函数后台将花费一定的时间完成预置并发的扩容，并将已启动准备的并发数、完成情况展示在列表中。

更新预置并发

当云函数后台完成预置并发的扩容时，您可按需修改并发数。

1. 登录 Serverless 控制台，单击左侧导航栏的函数服务。
2. 在“函数服务”列表页面，单击目标函数名，进入“函数管理”页面。
3. 选择左侧**并发配额**，在“预置并发”页面中，选择需更新版本所在行右侧的**设置**。
4. 在弹出的“设置函数预置并发”窗口中，更新设置值并单击**提交**即可。
   设置完成后，云函数平台将根据您的修改情况，在一定时间内再次完成并发数的增加或减少。

删除预置并发

当您不再计划使用某个预置并发配置时，可进行删除操作。

1. 登录 Serverless 控制台，单击左侧导航栏的函数服务。
2. 在“函数服务”列表页面，单击目标函数名，进入“函数管理”页面。
3. 选择左侧**并发配额**，在“预置并发”页面中，选择目标版本所在行右侧的**删除**。
4. 在弹出的“删除函数预置并发配额”窗口中单击**确认**即可。
   配置删除后，云函数后台将逐步回收并发实例。

相关操作

利用预置并发进行流量切换

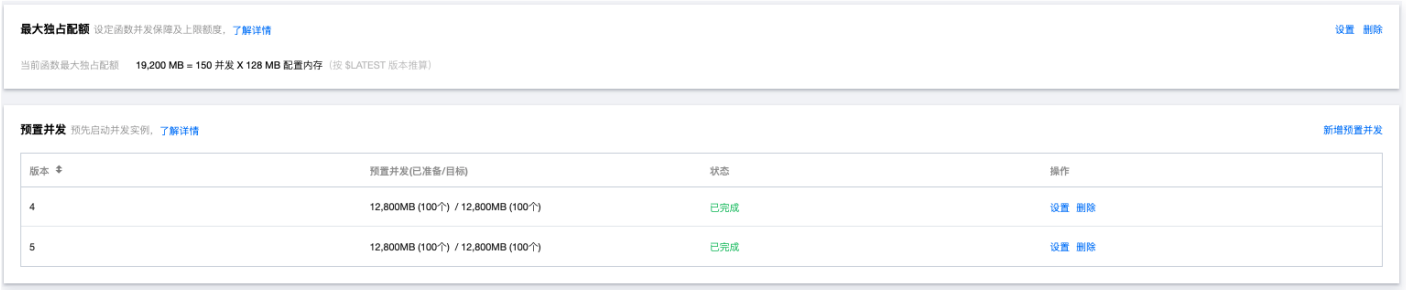
您可以根据业务并发量设置函数最大独占配额，根据流量切换需要配置预置并发。操作过程如下：

1. 发布新版本。
2. 在新版本上设置需要的预置值。
3. 等待新版本的预置完全启动。
4. 通过 流量灰度，逐步将流量从旧版本切到新版本。如遇到问题，则流量切回旧版本。
5. 流量完全切至新版本，观察一段时间无异样后，删除旧版本的预置并发。

以下表中的场景为例，您可以在函数最大独占配额150并发的情况（该函数能最多可以并发处理150个请求）下，同时给多个版本设置100预置并发（每个版本都有100个启动好的实例），从而进行无初始化过程的流量切换。

业务场景	业务平均并发	预置并发	函数最大独占配额	效果
100%预置	100并发	12,800 MB（100 并发）	19,200 MB（150 并发）	持续有100个并发实例的无需初始化，超出的并发实例第一次调用需要初始化。可以保障150并发，超过会有并发超限错误。

如下图所示，版本4和版本5配置100预置并发。此时您可以通过流量灰度能力，将业务的100并发从版本4灰度到版本5。无论100并发以任何比例分配至版本4和版本5，都不会遇到有初始化过程的实例，从而方便您更快地发布版本与流量切换。


最大独占配额 设定函数并发保障及上限额度，了解详情
设置 删除
当前函数最大独占配额 19,200 MB = 150 并发 X 128 MB 配置内存（按 $LATEST 版本推算）
预置并发 预先启动并发实例，了解详情
新增预置并发
版本
预置并发(已准备/目标)
状态
操作
4
12,800MB (100个) / 12,800MB (100个)
已完成
设置 删除
5
12,800MB (100个) / 12,800MB (100个)
已完成
设置 删除

定时预置

最近更新时间：2025-06-24 17:24:11

概述

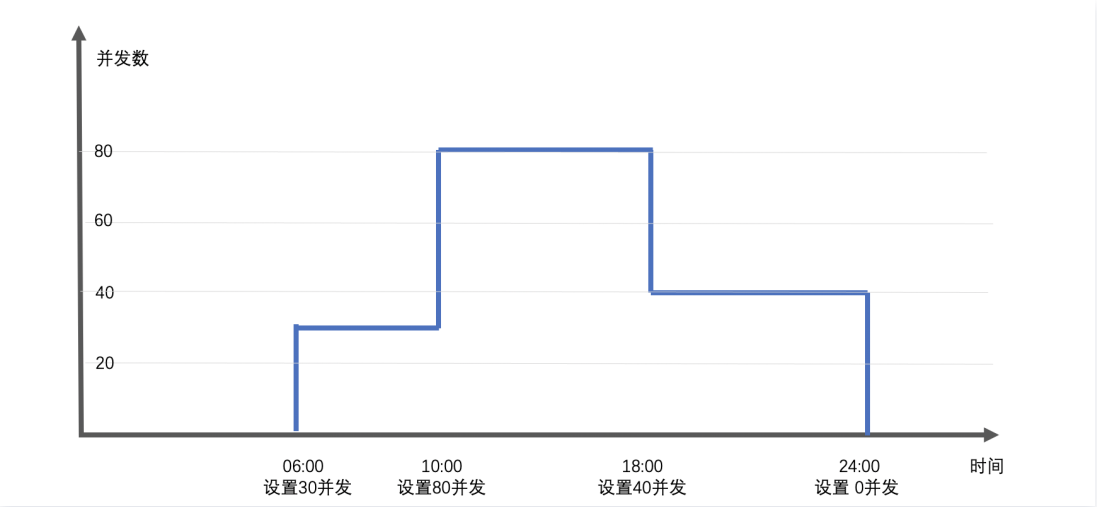
定时预置属于 预置并发 的弹性策略，可以根据业务情况合理配置预置并发，在指定时间对预置并发进行升降配置，提高预置并发的利用率，降低过多的闲置费用。当函数实际所需并发大于定时预置值时，会通过按量模式进行弹性扩容操作。定时预置支持：不重复、每天、周一至周五、周六周日、自定义几种任务类型。

适用场景

- 具有周期性波动规律的函数，例如数据处理等场景。
- 可预知业务流量高峰的函数，例如举办活动等业务有明确时间的场景。

定时预置功能及相关限制

定时预置设置值为该时间段内目标值，例如按照业务需求需要设置四个定时任务，即6点设置30并发，10点设置80并发，18:00设置40并发，24点设置0并发，那么最终预置并发的波动情况如下：



说明

- 定时预置 Cron 表达式有七个必需字段，按空格分隔。更多信息，请参见 Cron 表达式 。
- 用户账号同一函数版本下，定时预置任务数量有一定限制，详情请参见 配额限制 。如需增加定时任务的配额数量（即配额提升），可通过 提交工单 申请。
- 云函数平台会根据您业务的情况调整预置并发的启动速度，默认为100个/分钟，请合理配置定时预置启动时间，详情请参见 并发服务承诺 。
- 定时预置任务在同一个时间点有重叠任务时后一个任务会覆盖前一个任务。

场景示例

业务需要在2021年11月03日12:00流量高峰开始定时1个预置并发，流量高峰过后，在2021年11月03日16:00结束定时任务。具体操作如下：

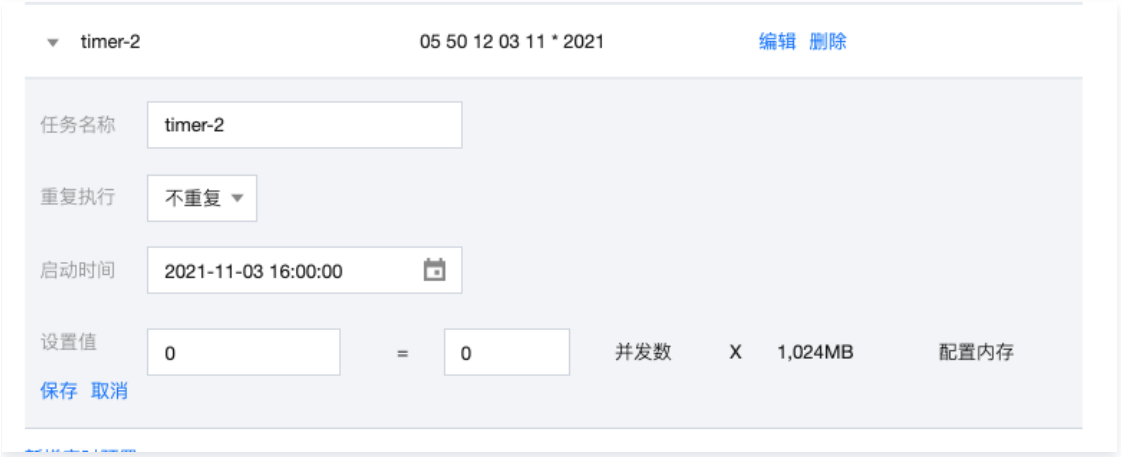
启动定时任务

定时启动预置任务，需要新增定时任务，选择启动时间，将预置设置为目标值，具体操作如下：



结束定时任务

定时结束预置任务，需要**额外新增定时预置任务**，选择结束时间，将预置设置值改为0，具体操作如下：



动态指标预置

最近更新时间：2022-02-14 17:42:01

概述

动态指标预置属于 预置并发 的弹性策略，云函数系统将周期性采集函数实际并发执行情况，结合已配置的最大、最小并发数以及目标并发利用率指标来控制预置并发功能的动态伸缩，使函数预置并发数更加接近资源的真实使用量，提高预置并发的利用率，降低了过多的闲置费用。当函数实际所需并发大于动态指标预置的并发数时，则通过按量模式进行弹性扩容操作。

适用场景

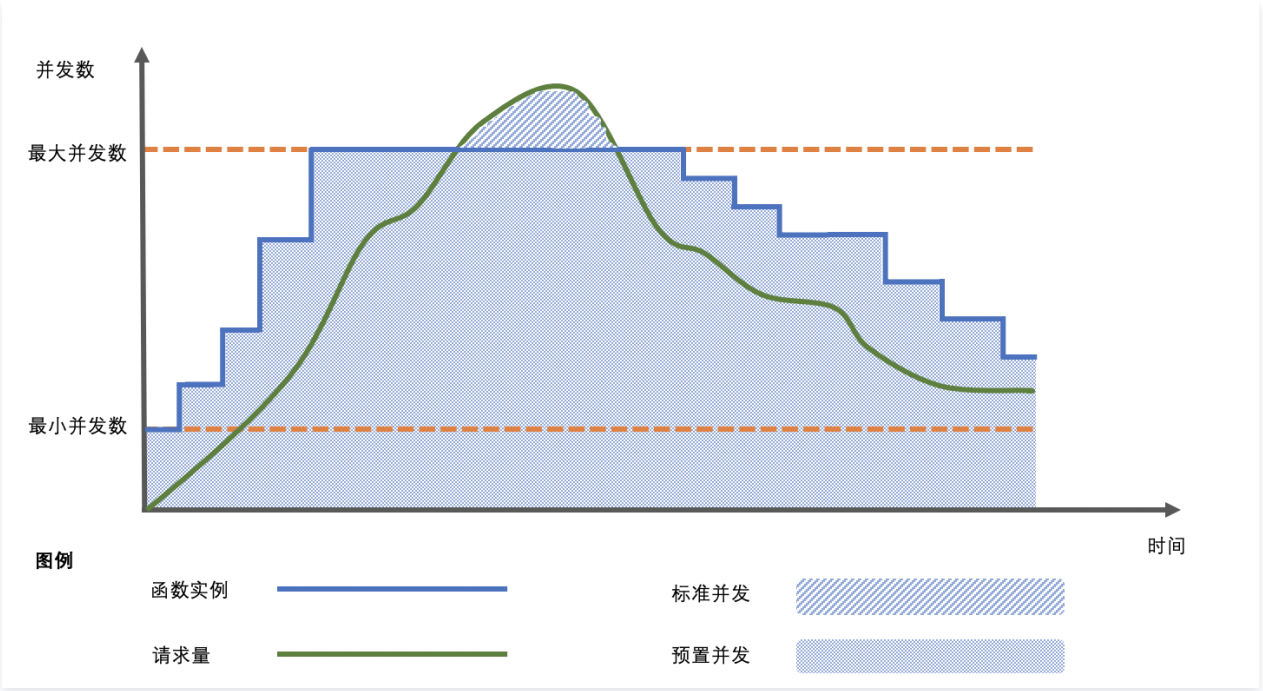
- 对预置闲置费用非常敏感的业务，可使用动态指标预置功能降低预置闲置费用。
- 对冷启动比较敏感且无法预知业务流量高峰的函数。

实现原理

动态指标预置时会根据业务配置的动态策略进行伸缩。若业务设置了最小、最大并发数以及并发利用率指标，系统将会保证最小并发数的预置资源，同时预置并发数将会在最小值和最大值之间动态伸缩。

扩缩容策略

- **扩容**：当业务实际请求量不断增加，触发扩容阈值时系统开始扩容，达到最大并发数上限时则停止扩容操作。超出部分的请求将会通过按量模式进行扩容。
  扩容频率：每10秒进行一次扩容操作，扩容没有窗口时间。
- **缩容**：当业务实际请求量不断减小，触发缩容阈值时系统开始缩容，达到最小并发数下限时则停止缩容操作。
  缩容频率：缩容时通过10分钟的窗口时间来实现相对保守的缩容过程，即在执行动态伸缩操作后，在窗口时间内不会再进行缩容操作，可以理解为类似等一下放技能的冷却时间。若此前没有执行过扩缩容操作，则10秒就可以进行缩容操作。



预置目标值

预置目标值由当前并发数、目标并发利用率指标共同决定。

- **预置目标值** = 当前函数总实例数 × 当前并发利用率 ÷ 目标并发利用率 = 当前函数总实例数 ×（当前并发数 ÷ 当前函数总实例数）÷ 目标并发利用率 = **当前并发数/目标并发利用率指标**
- 预置目标值计算示例：当前并发数为100，目标并发利用率为80%，经过计算100 / 80% = 125，即预置目标值的会扩容到125个。

并发利用率

函数的并发利用率是指当前函数实例正在响应的请求并发值与当前函数总实例数占比，指标取值范围为[0,1)。

最小并发数

最小并发数代表该函数最少需要预置的并发个数，即缩容的下限值。

最大并发数

最大并发数代表该函数最多可预置的并发个数，即扩容的上限值。

版权所有：腾讯云计算（北京）有限责任公司

操作步骤

新增动态指标预置

1. 登录 Serverless 控制台，在左侧选择**函数服务**。
2. 在“函数服务”列表页面，选择需进行配置的函数名，进入“函数管理”页面。
3. 选择左侧**并发配额 > 预置并发**，进入“预置并发”页面。
4. 在“预置并发”页面中，单击**新增预置并发**。如下图所示：



5. 在弹出的“新增函数预置并发”窗口中，选择预置类型为动态指标预置，函数版本。按照业务场景设置最小并发数、最大并发数以及目标并发利用率指标，单击**提交**即可。如下图所示：



设置完成后，您可在“预置并发”中查看配置的状态。云函数后台将花费一定的时间完成预置并发的扩容，并将已启动准备的并发数、完成情况展示在列表中。

更新动态指标预置

更新动态指标预置时，您可以修改预置类型，最小并发数、最大并发数以及目标并发利用率指标等参数。

1. 登录 Serverless 控制台，在左侧选择**函数服务**。
2. 在“函数服务”列表页面，选择需更新预置并发函数，进入“函数管理”页面。
3. 选择左侧**并发配额 > 预置并发**，进入“预置并发”页面。
4. 在“预置并发”页面中，选择需更新版本所在行右侧的**设置**。
5. 在弹出的“设置函数预置并发”窗口中，更新设置值并单击**提交**即可。如下图所示：



> **注意**
> 预置类型支持基础预置，动态指标预置。二者任选其一，业务更新预置类型后，此前设置的预置类型将会失效。

删除动态指标预置

1. 登录云函数控制台，选择左侧导航栏中的 函数服务 。
2. 在“函数服务”列表页面，选择需删除预置并发函数，进入“函数管理”页面。
3. 选择左侧**并发配额 > 预置并发**，进入“预置并发”页面。
4. 在“预置并发”页面中，选择需调整版本所在行右侧的**删除**。如下图所示：



5. 在弹出的“删除函数预置并发配额”窗口中单击**确认**即可。

并发超限

最近更新时间：2024-06-20 14:36:21

并发超限

并发超限（ResourceLimitReached）指云函数 SCF 在同一时刻执行的并发数超过 配额限制 导致的函数报错。并发超限分为同步调用、异步调用两种情况。

异步调用

异步调用包含 云 API 触发器 的异步调用、COS 触发器、定时触发器、CMQ Topic 触发器、CLS 触发器 及 MPS 触发器 等，具体触发器调用类型请参考相关触发器说明文档。
当异步调用并发超限时其处理逻辑由云函数 SCF 进行自动重试，详情可参见 异步调用重试策略 。

同步调用

同步调用包含 云 API 触发器 的同步调用、API 网关触发器 及 CKafka 触发器 。
由于同步调用的过程中，错误信息会直接返回给用户，所以在同步调用中发生错误时，平台不会自动重试，重试策略（是否重试、重试几次）均由调用方决定。同步调用场景下云函数 SCF 将返回 432状态码，请求不会进行重试。

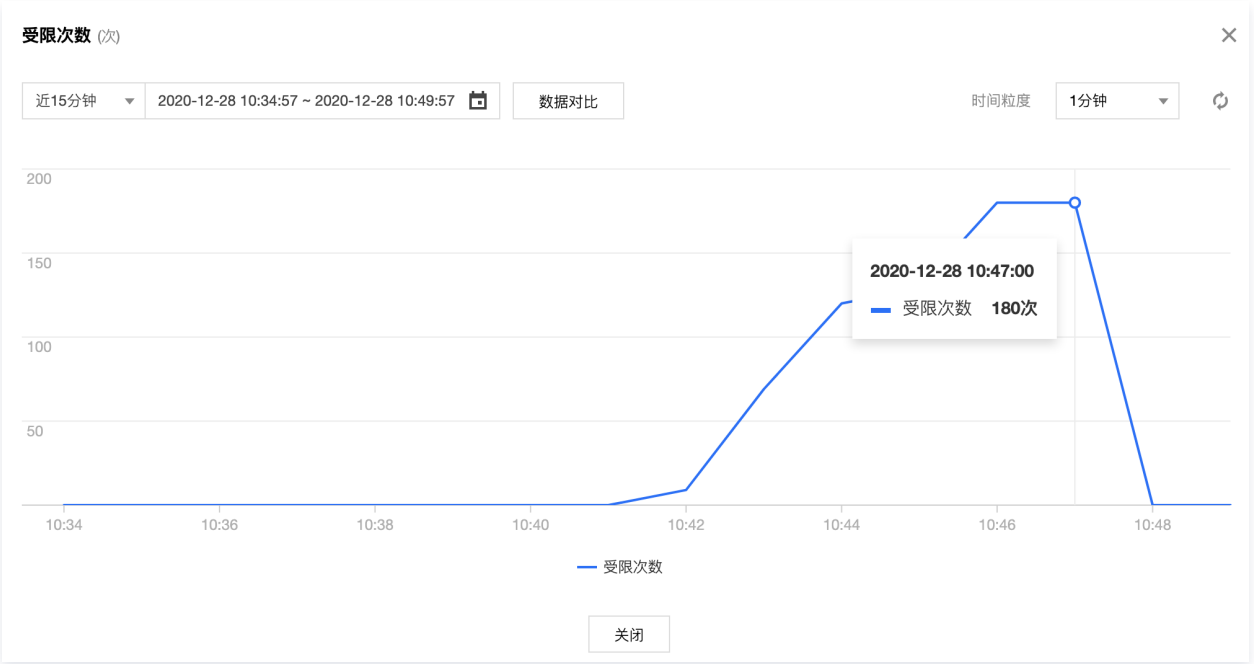
解决并发超限相关指引

查看并发超限监控

您可通过云函数控制台查看相关函数的函数受限次数和具体受限日志。

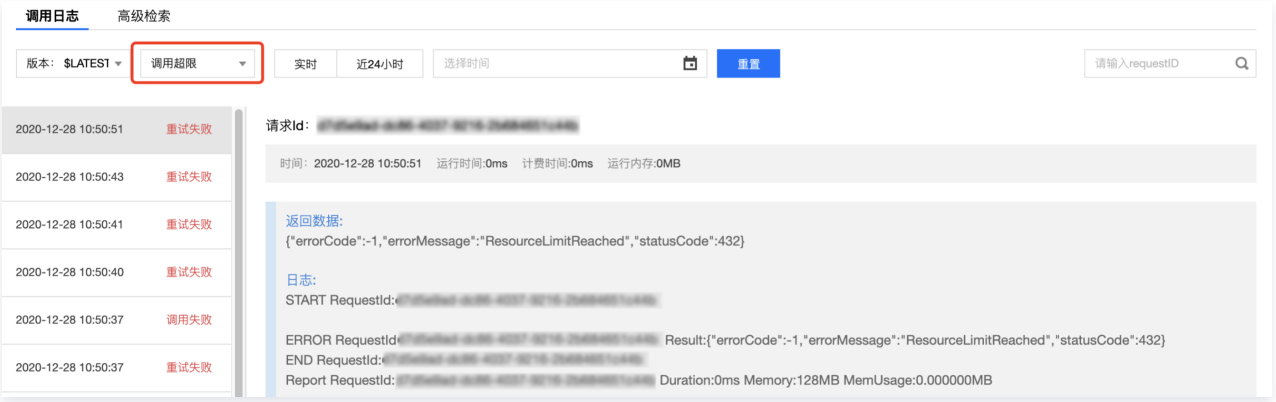
查看函数受限次数

1. 登录 Serverless 控制台，在左侧选择**函数服务**。
2. 在**函数服务**页中，选择需要查看的函数名，进入该函数的详情页面。
3. 在**函数管理**中，选择**监控信息** > **受限次数**，查看相关函数的受限次数情况。如下图所示：



查看函数受限日志

1. 登录 Serverless 控制台，在左侧选择**函数服务**。
2. 在**函数服务**页中，选择需要查看的函数名，进入该函数的详情页面。
3. 在**日志查询**中，选择**调用日志** > **调用超限**，查看相关函数的具体受限日志。如下图所示：



并发超限处理

- **异步调用**对并发超限场景有平台重试策略帮助用户自动对并发超限进行处理并重试，通常情况下异步调用的并发超限用户无需进行任何操作，在设定的最长等待时间内，函数平台会自动对并发超限错误进行重试。
- **同步调用**发生错误时，错误信息会直接返回给用户，请求不会进行重试。

> **注意**
> 异步调用中，如对时效性比较敏感可以通过配置最大独占配额来减少或降低超限对业务系统的影响。例如需要重要消息超过设置的最长保留时间后不会丢失则应设置死信队列兜底。

配置死信队列

死信队列 DLQ 是一个用户账号下的 CMQ 队列，可用于收集错误事件信息、分析失败原因。如果您为函数配置了死信队列，由于超限导致的重试失败的消息都将发送到死信队列。详情可参见 死信队列创建 。

配置最大独占配额

最大独占配额额度是用于保障函数可用并发的最大额度，通过配置最大独占配额额度，函数可以在额度内启动足够并发数量，并发最大可以达到配置额度。通过设置最大独占配额额度，函数不再与其他函数共享账号并发额度，可以降低出现并发超限的可能，获得更有保障的运行。详情可参见 设置最大独占配额 。

触发器管理
创建触发器

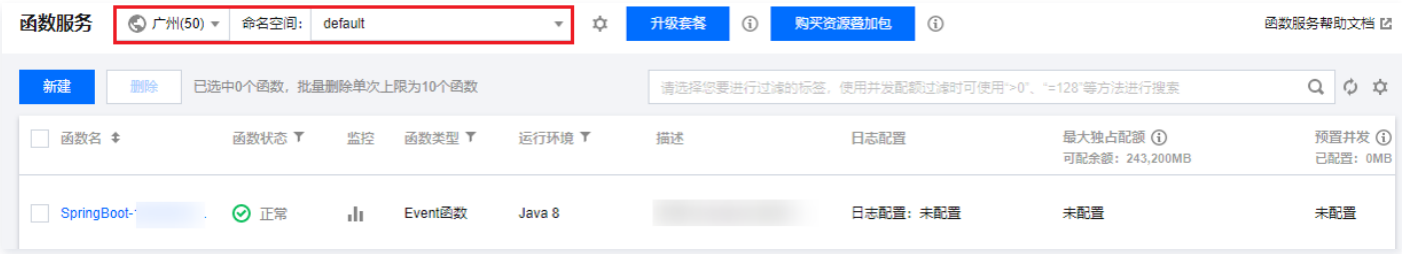
最近更新时间：2023-03-20 11:32:55

云函数创建完成后，可以通过创建触发器来将云函数与事件源进行关联。关联后的事件源，会在事件产生时，根据设计方式，以同步或异步的方式完成云函数触发运行，并在触发时将事件作为入参传递给入口函数。
通过控制台或 Serverless Cloud Framework 命令行均可以完成云函数触发器创建。

> **注意**
> Web 类型函数只支持创建 API 网关触发器，详情请查看 创建 Web 函数 。

通过控制台完成触发器创建

1. 登录 Serverless 控制台，单击左侧导航栏的函数服务。
2. 在“函数服务”列表页面上方，选择函数所在的地域及命名空间。如下图所示：



3. 单击函数名，进入该函数的详情页面。
4. 选择左侧的**触发管理**，进入触发器浏览及操作界面，单击**创建触发器**，开始创建一个新的触发器。如下图所示：



5. 在弹出的“创建触发器”窗口中，选择触发别名/版本，并选择触发方式。如下图所示：



- 触发别名/版本：切换至期望创建触发器的版本。触发器可以在函数的指定版本上创建。当创建在云函数的指定版本上时，事件将触发指定的版本代码。详情见 版本管理 。

  > **注意**
  > 由于云函数的触发器总数量、各种类触发器数量的限制，在版本下配置的触发器会占用当前函数的触发器配置限额。如需调大触发器限额，可 联系我们 提升限额。

- 触发方式：选择不同触发方式所需填写的内容也将不同。例如：定时触发器需添加触发器名称、周期和启用情况，对象存储 COS 触发器需要添加触发的 COS Bucket、事件类型以及前后缀过滤方式。详情见各 触发器 的说明文档。

6. 完成触发器配置后，单击**提交**，完成触发器创建。

通过 Serverless Cloud Framework 命令行完成触发器创建

> **说明**
> 在使用 Serverless Cloud Framework 工具之前，请 安装 Serverless Cloud Framework 。

本地函数请在 `serverless.yml` 文件下新增触发器描述，并通过 Serverless Cloud Framework 执行 `scf deploy` 命令，即可为函数新增触发器。

视频教程

以下视频将为您介绍如何创建、删除及启停触发器：
观看视频

删除触发器

最近更新时间：2022-12-13 16:09:54

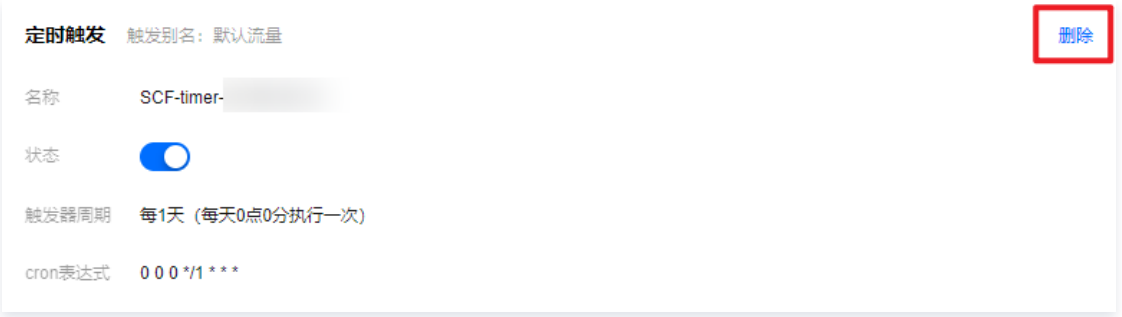
可以通过删除触发器来解除云函数与事件源的关联。解除关联后，事件源将不会再触发云函数的执行。您可通过控制台完成云函数触发器删除。

通过控制台完成触发器删除

1. 登录 Serverless 控制台，单击左侧导航栏的函数服务。
2. 在“函数服务”列表页面上方，选择函数所在的地域及命名空间。如下图所示：



3. 单击函数名，进入该函数的详情页面。
4. 选择左侧的**触发管理**，进入触发器浏览及操作界面，单击触发器右上角的**删除**。如下图所示：



在弹出窗口中确认删除即可。

启停触发器

最近更新时间：2024-11-07 10:23:52

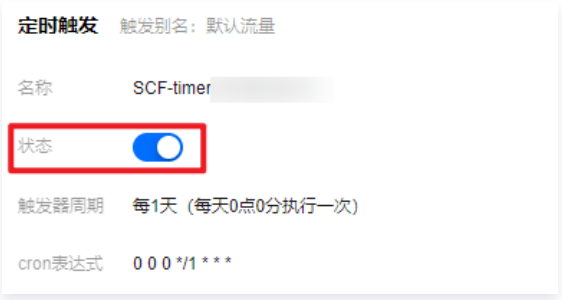
可以通过设置触发器启动或停止，来临时停止云函数被事件源所发生的事件触发。本文介绍如何通过控制台设置触发器的启停状态。

通过控制台完成触发器启停

1. 登录 Serverless 控制台，单击左侧导航栏的函数服务。
2. 在**函数服务**中，选择函数所在的地域及命名空间。如下图所示：



3. 单击函数名，进入该函数的详情页面。
4. 选择左侧的**触发管理**，进入触发器浏览及操作界面，单击期望启停触发器“状态”中的，切换触发器的启停状态。如下图所示：



创建触发器时设置启停状态

在创建触发器时，可以设置触发器的启停状态。当触发器创建完成后，会处于设置好的状态上。
例如，创建定时触发器时，希望该触发器不立刻生效，而是稍后按需生效，则可以取消勾选**立即启用**。如下图所示：



完成触发器创建后，可以通过切换启停状态使得触发器生效。

注意事项

目前有部分触发器暂时还未支持启停状态切换，控制台上将无法看到启停切换按键。在稍后触发器支持启停能力后，将能看到触发器的启停状态和切换按键。

函数 URL
函数 URL 概述

最近更新时间：2025-02-18 10:54:23

简介

函数 URL 是函数的专用 HTTP(S) 端点。为函数配置函数 URL 后，可以通过 Web 浏览器、curl、Postman 或任何 HTTP 客户端通过其 HTTP(S) 端点调用函数。
您可以通过 SCF 控制台或 SCF API/CLI 创建和配置函数 URL。创建函数 URL 后，其 URL 端点将永久不变。函数 URL 的端点格式如下：

```
公网：https://<app-id>-<url-id>.<region>.tencentscf.com
内网：https://<app-id>-<url-id>.in.<region>.tencentscf.com
```

函数 URL 和触发器同级存在，适用于事件函数和 Web 函数。您可以在启用函数 URL 的同时配置 API 网关等触发器。
函数 URL 和函数的版本、别名一对一绑定，您需要手动为每个版本和别名开启或关闭函数 URL。默认情况下，函数 URL 是关闭的。

> **说明：**
> 如需生成 WSS 地址，请在函数配置中启用 WebSocket 支持。

调用参数

事件函数

请求参数

URL 在接受到请求后，函数将会被触发运行，同时 URL 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。请求的相关信息包含了例如具体接受到请求的服务和 API 规则、请求的实际路径、方法、请求的 path、headers、query 等内容：

```
// Event 详细信息示例【兼容apigw协议，去掉headerParameters、isBase64Encoded、pathParameters、
queryStringParameters、requestContext 相关字段】：
{
    "body":"{\"test\":\"hello world\"}",
    "headers":{
        "accept":"*/*",
        "accept-encoding":"gzip, deflate, br",
        "cache-control":"no-cache",
        "connection":"keep-alive",
        "content-length":"17",
        "x-scf-remote-addr":"111.206.96.145"//此字段为客户端的请求IP
    },
    "httpMethod":"POST",
    "path":"/",
    "queryString":{
        "a":"1",
        "b":"2"
    }
}
```

响应参数

当函数返回响应时，函数会解析响应并将其转换为 HTTP 响应，标准响应负载：

```
{
   "statusCode": 201,
    "headers": {
        "Content-Type": "application/json",
        "My-Custom-Header": "Custom Value"
```

```
    },
    "body": "{ \"message\": \"Hello, world!\" }"
}
```

函数会为您推断响应格式。如果您的函数返回有效的 JSON 并且没有返回 statusCode，函数会假设 statusCode 为200，content-type 为 application/json，body 是函数响应。
函数响应标准响应参数格式如下：

函数输出	HTTP 响应（客户端看到的内容）

`"Hello, world!"`	`HTTP/2 200`
	`date: Wed, 08 Sep 2021 18:02:24 GMT`
	`content-type: application/json`
	`content-length: 15`
	
	`"Hello, world!"`

`{`
`  "message": "Hello, world!"`
`}`
	`HTTP/2 200`
	`date: Wed, 08 Sep 2021 18:02:24 GMT`
	`content-type: application/json`
	`content-length: 34`
	
	`{`
	`  "message": "Hello, world!"`
	`}`

`{`
`  "statusCode": 201,`
`  "headers": {`
`  "Content-Type": "application/json",`
`  "My-Custom-Header": "Custom Value"`
`  },`
`  "body": JSON.stringify({`
`  "message": "Hello, world!"`
`  })`
`}`
	`HTTP/2 201`
	`date: Wed, 08 Sep 2021 18:02:24 GMT`
	`content-type: application/json`
	`content-length: 27`
	`my-custom-header: Custom Value`
	
	`{`
	`  "message": "Hello, world!"`
	`}`

Web 函数

URL 在接受到 HTTP 请求后，该函数将会被触发运行，此时 URL 会将 HTTP 请求直接透转，不再做 event 类型格式转换，同时请求响应也直接透转。

创建函数 URL

最近更新时间：2025-05-08 16:30:22

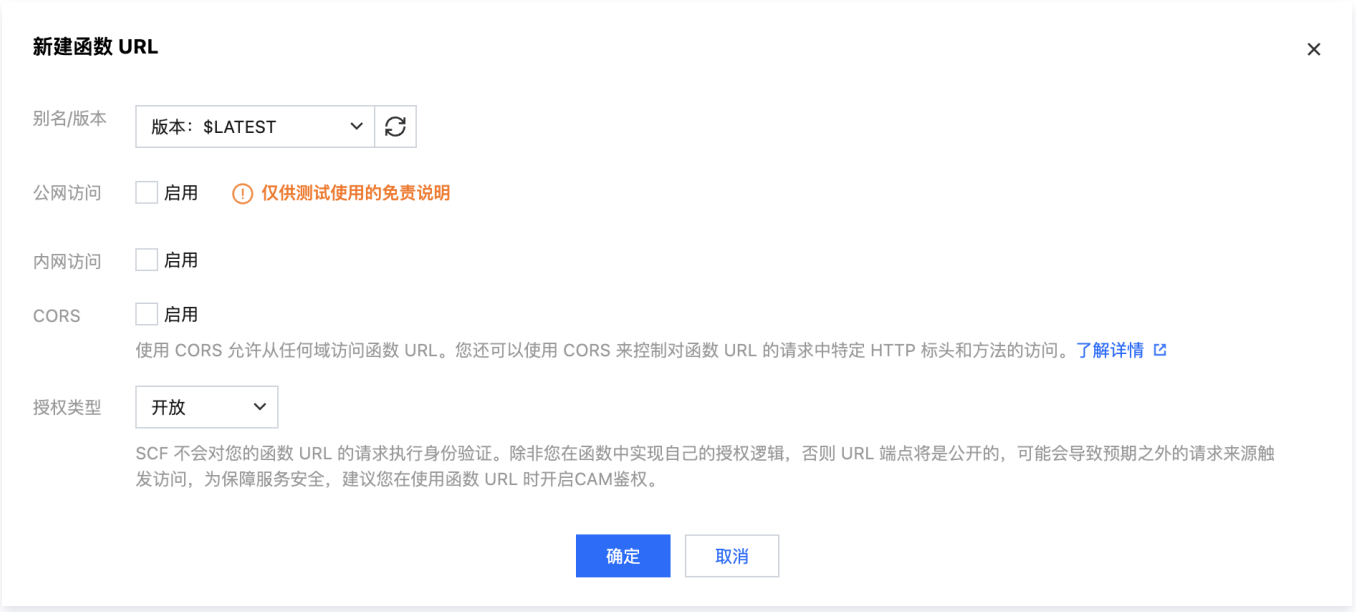
本文向您介绍如何使用控制台和使用 API 创建函数 URL。

使用控制台创建函数 URL

1. 登录 Serverless 控制台，单击左侧导航栏的**函数服务**。
2. 在**函数服务**页面，单击函数名，进入该函数详情页面。
3. 选择左侧导航中的**函数 URL**，单击**新建函数 URL**。如下图所示：



4. 在**新建函数 URL** 页面，参考以下信息进行创建。



配置项	描述
别名/版本	URL 绑定在别名或者版本维度，每一个别名或者版本仅允许创建一个 URL。
公网/内网访问	可以根据业务需求选择开启公网或者内网 URL 访问。
CORS	使用 CORS 允许从任何域访问函数 URL。您还可以使用 CORS 来控制对函数 URL 的请求中特定 HTTP 标头和方法的访问。具体设置方法查看 跨源资源共享（CORS）。
授权类型	授权类型支持选择**开放**和 **CAM 鉴权**。 • **开放**：不需要对函数请求进行身份验证，支持匿名访问，任何人都可以发起 HTTP 请求调用您的函数。 • **CAM 鉴权**：需要对函数 CAM 鉴权验证，用户可以基于函数 InvokeFunctionUrl 接口进行资源管理和使用权限配置，详情请参见 函数 URL 认证鉴权配置。

5. 单击**确定**完成创建。

使用 API 创建函数 URL

创建函数 URL 与创建触发器共用接口，公用参数详情请参见 设置函数触发方式，其中 `Type` 参数请填写 `http`， `TriggerDesc` 参数配置说明如下：

名称	类型	必选	描述
AuthType	String	是	授权类型，CAM 表示需使用 函数 URL 认证鉴权配置，NONE 表示无需授权。
NetConfig	NetConfig	是	网络访问配置，示例值：`{ "EnableIntranet": true, "EnableExtranet": false }`
CorsConfig	CorsConfig	否	CORS 配置，示例值：`{"Enable":true,"Origins":["*"],"Headers":["content-type","custom-header"],"Methods":["POST","PATCH"],"ExposeHeaders":["*"],"MaxAge":10,"Credentials":true}`

NetConfig

名称	类型	必选	描述
EnableIntranet	Bool	是	是否开启内网访问。
EnableExtranet	Bool	是	是否开启公网访问。

CorsConfig

名称	类型	必选	描述
Enable	Bool	是	是否开启 CORS。
Origins	Array of String	是	Access-Control-Allow-Origin 参数指定了单一的源，告诉浏览器允许该源访问资源。或者，对于不需要携带身份凭证的请求，服务器可以指定该字段的值为通配符“*”，表示允许来自任意源的请求。
Headers	Array of String	否	Access-Control-Allow-Headers 标头字段用于预检请求的响应。其指明了实际请求中允许携带的标头字段。这个标头是服务器端对浏览器端 Access-Control-Request-Headers 标头的响应。
Methods	Array of String	是	Access-Control-Allow-Methods 标头字段指定了访问资源时允许使用的请求方法，用于预检请求的响应。其指明了实际请求所允许使用的 HTTP 方法。
ExposeHeaders	Array of String	否	Access-Control-Expose-Headers 头将指定标头放入允许列表中，供浏览器的 JavaScript 代码（如 getResponseHeader()）获取。
MaxAge	Integer	否	Access-Control-Max-Age 头指定了 preflight 请求的结果能够被缓存多久。
Credentials	Bool	否	Access-Control-Allow-Credentials 头指定了当浏览器的 credentials 设置为 true 时是否允许浏览器读取 response 的内容。

参数示例

```
trigger_desc = {
    "AuthType": "NONE",
    "NetConfig": {
        "EnableIntranet": true,
        "EnableExtranet":  false
    },
    "CorsConfig": {
        "Enable":true,
        "Origins":["*"],
        "Headers":["content-type","custom-header"],
        "Methods":["POST","PATCH"],
        "ExposeHeaders":["*"],
        "MaxAge":10,
        "Credentials":true
```

```
        }
}
params = {
    "FunctionName": "helloworld",
    "TriggerName": "func_url",
    "TriggerDesc": json.dumps(trigger_desc),
    "Type": "http",
    "Namespace": "default",
    "Enable": "OPEN",
}
```

函数 URL 认证鉴权配置

最近更新时间：2025-06-05 09:53:12

简介

您可以通过配置认证鉴权策略来控制对函数 URL 的访问。

在配置函数 URL 时，必须指定以下认证选项之一：

- **CAM 鉴权**：需要对函数 CAM 鉴权验证，用户可以基于函数 InvokeFunctionUrl 接口进行资源管理和使用权限配置。您可以通过配置 InvokeFunctionUrl 策略权限来开放或限制接口的访问。
- **开放**：不需要对函数请求进行身份验证，支持匿名访问，任何人都可以发起 HTTP 请求调用您的函数。

配置 InvokeFunctionUrl 策略权限

您可以参考以下步骤配置 InvokeFunctionUrl 策略权限来开放或限制接口的访问。

1. 在访问管理控制台的 策略 页面，单击左上角的**新建自定义策略**。
2. 在弹出的选择创建方式窗口中，单击**按策略生成器创建**，进入编辑策略页面。
3. 在**可视化策略生成器**中添加服务与操作栏，补充以下信息，编辑一个授权声明。
   - 效果（必选）：选择**允许**。
   - 服务（必选）：选择云函数 (scf)。
   - 操作（必选）：单击全部操作 (scf:*)右侧的**展开**，搜索 InvokeFunctionUrl，并勾选。如下图所示：



   - 资源（必填）：选择全部资源或您要授权的特定资源。
   - 条件（选填）：设置上述授权的生效条件。
4. 完成策略授权声明编辑后，单击**下一步**，进入基本信息和关联用户/用户组/角色页面。
5. 在关联用户/用户组/角色页面，补充策略名称和描述信息，可同时关联用户/用户组/角色快速授权。
6. 单击**完成**，完成按策略生成器创建自定义策略的操作。

签名生成和认证流程

客户端生成签名

签名算法请参见 安全凭证服务签名方法 。代码示例如下：

Java

```
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.TimeZone;
import java.util.TreeMap;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import javax.xml.bind.DatatypeConverter;

public class TencentCloudAPITC3Demo {
    private final static Charset UTF8 = StandardCharsets.UTF_8;
    // 需要设置环境变量 TENCENTCLOUD_SECRET_ID，值为示例的 AKID**************
```

```
    private final static String SECRET_ID = System.getenv("TENCENTCLOUD_SECRET_ID");
    // 需要设置环境变量 TENCENTCLOUD_SECRET_KEY，值为示例的 Gu5t9xGARNpq86cd98joQYCN3*******
    private final static String SECRET_KEY = System.getenv("TENCENTCLOUD_SECRET_KEY");
    private final static String CT_JSON = "application/json";

    public static byte[] hmac256(byte[] key, String msg) throws Exception {
        Mac mac = Mac.getInstance("HmacSHA256");
        SecretKeySpec secretKeySpec = new SecretKeySpec(key, mac.getAlgorithm());
        mac.init(secretKeySpec);
        return mac.doFinal(msg.getBytes(UTF8));
    }

    public static String sha256Hex(String s) throws Exception {
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        byte[] d = md.digest(s.getBytes(UTF8));
        return DatatypeConverter.printHexBinary(d).toLowerCase();
    }

    public static void main(String[] args) throws Exception {
        String service = "scf";
        String host = "1253970226-xxxxxxx-cq.scf.tencentcs.com";
        String uin = "xxxxxx";//此处替换为需要使用的真实uin
        String algorithm = "TC3-HMAC-SHA256";
        String timestamp = String.valueOf(System.currentTimeMillis() / 1000);
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        // 注意时区，否则容易出错
        sdf.setTimeZone(TimeZone.getTimeZone("UTC"));
        String date = sdf.format(new Date(Long.valueOf(timestamp + "000")));

        // ************* 步骤 1：拼接规范请求串 *************
        String httpRequestMethod = "POST";
        String canonicalUri = "/";
        String canonicalQueryString = "";
        String canonicalHeaders = "content-type:application/json; charset=utf-8\n"
                + "host:" + host + "\n" ;
        String signedHeaders = "content-type;host";

        // 请求体
        String payload = "{\"Limit\": 1, \"Filters\": [{\"Values\": [\"\\u672a\\u547d\\u540d\"], \"Name\":
\"instance-name\"}]}";
        String hashedRequestPayload = sha256Hex(payload);
        String canonicalRequest = httpRequestMethod + "\n" + canonicalUri + "\n" + canonicalQueryString +
"\n"
                + canonicalHeaders + "\n" + signedHeaders + "\n" + hashedRequestPayload;
        System.out.println(canonicalRequest);

        // ************* 步骤 2：拼接待签名字符串 *************
        String credentialScope = date + "/" + service + "/" + "tc3_request";
        String hashedCanonicalRequest = sha256Hex(canonicalRequest);
        String stringToSign = algorithm + "\n" + timestamp + "\n" + credentialScope + "\n" +
hashedCanonicalRequest;
        System.out.println(stringToSign);

        // ************* 步骤 3：计算签名 *************
        byte[] secretDate = hmac256(("TC3" + SECRET_KEY).getBytes(UTF8), date);
        byte[] secretService = hmac256(secretDate, service);
        byte[] secretSigning = hmac256(secretService, "tc3_request");
        String signature = DatatypeConverter.printHexBinary(hmac256(secretSigning,
stringToSign)).toLowerCase();
        System.out.println(signature);
```

```
        // ************* 步骤 4: 拼接 Authorization *************
        String authorization = algorithm + " " + "Credential=" + SECRET_ID + "/" + credentialScope + ", "
                + "SignedHeaders=" + signedHeaders + ", " + "Signature=" + signature;
        System.out.println(authorization);


        StringBuilder sb = new StringBuilder();
        sb.append("curl -X POST https://").append(host+canonicalUri)
        .append(" -H \"Authorization: ").append(authorization).append("\"")
        .append(" -H \"Content-Type: application/json; charset=utf-8\"")
        .append(" -H \"Host: ").append(host).append("\"")
        .append(" -H \"X-Scf-Cam-Uin: ").append(uin).append("\"")
        .append(" -H \"X-Scf-Cam-Timestamp: ").append(timestamp).append("\"")
        .append(" -d '").append(payload).append("'");
        System.out.println(sb.toString());
    }
}
```

Go

```
package main

import (
    "crypto/hmac"
    "crypto/sha256"
    "encoding/hex"
    "fmt"
    "os"
    "time"
)

func sha256hex(s string) string {
    b := sha256.Sum256([]byte(s))
    return hex.EncodeToString(b[:])
}

func hmacsha256(s, key string) string {
    hashed := hmac.New(sha256.New, []byte(key))
    hashed.Write([]byte(s))
    return string(hashed.Sum(nil))
}

func main() {
    // 需要填写账户UIN
    uin := "xxxx"
    // 需要设置环境变量 TENCENTCLOUD_SECRET_ID，值为示例的 AKID***************
    secretId := os.Getenv("TENCENTCLOUD_SECRET_ID")
    // 需要设置环境变量 TENCENTCLOUD_SECRET_KEY，值为示例的 Gu5t9xGARNpq86cd98joQYCN3*******
    secretKey := os.Getenv("TENCENTCLOUD_SECRET_KEY")
    host := "1253970226-xxxxxxx-cq.scf.tencentcs.com"

    algorithm := "TC3-HMAC-SHA256"
    service := "scf"
    var timestamp int64 = time.Now().Unix()

    // step 1: build canonical request string
    httpRequestMethod := "POST"
    canonicalURI := "/"
    canonicalQueryString := ""
    canonicalHeaders := fmt.Sprintf("content-type:%s\nhost:%s\n",
```

```
        "application/json", host)
    signedHeaders := "content-type;host"
    payload := `{"Limit": 1, "Filters": [{"Values": ["\u672a\u547d\u540d"], "Name": "instance-name"}]}`
    hashedRequestPayload := sha256hex(payload)
    canonicalRequest := fmt.Sprintf("%s\n%s\n%s\n%s\n%s\n%s",
        httpRequestMethod,
        canonicalURI,
        canonicalQueryString,
        canonicalHeaders,
        signedHeaders,
        hashedRequestPayload)
    fmt.Println("canonicalRequest => ", canonicalRequest)

    // step 2: build string to sign
    date := time.Unix(timestamp, 0).UTC().Format("2006-01-02")
    credentialScope := fmt.Sprintf("%s/%s/tc3_request", date, service)
    hashedCanonicalRequest := sha256hex(canonicalRequest)
    string2sign := fmt.Sprintf("%s\n%d\n%s\n%s",
        algorithm,
        timestamp,
        credentialScope,
        hashedCanonicalRequest)
    fmt.Println("string2sign ==>", string2sign)

    // step 3: sign string
    secretDate := hmacsha256(date, "TC3"+secretKey)
    secretService := hmacsha256(service, secretDate)
    secretSigning := hmacsha256("tc3_request", secretService)
    signature := hex.EncodeToString([]byte(hmacsha256(string2sign, secretSigning)))
    fmt.Println(signature)

    // step 4: build authorization
    authorization := fmt.Sprintf("%s Credential=%s/%s, SignedHeaders=%s, Signature=%s",
        algorithm,
        secretId,
        credentialScope,
        signedHeaders,
        signature)
    fmt.Println(authorization)

    curl := fmt.Sprintf(`curl -X %s https://%s -H "Authorization: %s" -H "Content-Type: application/json"
-H "Host: %s"  -H "X-Scf-Cam-Uin: %s" -H "X-Scf-Cam-Timestamp: %d" -d '%s'`,
        httpRequestMethod, host, authorization, host, uin, timestamp, payload)
    fmt.Println(curl)
}
```

NodeJS

```
const crypto = require('crypto');
function sha256(message, secret = '', encoding) {
    const hmac = crypto.createHmac('sha256', secret)
    return hmac.update(message).digest(encoding)
}

function getHash(message, encoding = 'hex') {
    const hash = crypto.createHash('sha256')
    return hash.update(message).digest(encoding)
}

function getDate(timestamp) {
```

```
    const date = new Date(timestamp * 1000)
    const year = date.getUTCFullYear()
    const month = ('0' + (date.getUTCMonth() + 1)).slice(-2)
    const day = ('0' + date.getUTCDate()).slice(-2)
    return `${year}-${month}-${day}`
}
function main({secretId,secretKey,uin,endpoint}){
     const service = "scf"
    const timestamp = Math.round(new Date().getTime() / 1000);
    console.log(timestamp)
    //时间处理，获取世界时间日期
    const date = getDate(timestamp)

    // ************* 步骤 1：拼接规范请求串 *************
    const payload = "{\"Limit\": 1, \"Filters\": [{\"Values\": [\"\\u672a\\u547d\\u540d\"], \"Name\":
\"instance-name\"}]}"

    const hashedRequestPayload = getHash(payload);
    const httpRequestMethod = "POST"
    const canonicalUri = "/"
    const canonicalQueryString = ""

    const canonicalHeaders = "content-type:application/json\n"
        + "host:" + endpoint + "\n"
    const signedHeaders = "content-type;host"

    const canonicalRequest = httpRequestMethod + "\n"
                        + canonicalUri + "\n"
                        + canonicalQueryString + "\n"
                        + canonicalHeaders + "\n"
                        + signedHeaders + "\n"
                        + hashedRequestPayload
    console.log(canonicalRequest)

    // ************* 步骤 2：拼接待签名字符串 *************
    const algorithm = "TC3-HMAC-SHA256"
    const hashedCanonicalRequest = getHash(canonicalRequest);
    const credentialScope = date + "/" + service + "/" + "tc3_request"
    const stringToSign = algorithm + "\n" +
                    timestamp + "\n" +
                    credentialScope + "\n" +
                    hashedCanonicalRequest
    console.log(stringToSign)

    // ************* 步骤 3：计算签名 *************
    const kDate = sha256(date, 'TC3' + secretKey)
    const kService = sha256(service, kDate)
    const kSigning = sha256('tc3_request', kService)
    const signature = sha256(stringToSign, kSigning, 'hex')
    console.log(signature)

    // ************* 步骤 4：拼接 Authorization *************
    const authorization = algorithm + " " +
                    "Credential=" + secretId + "/" + credentialScope + ", " +
                    "SignedHeaders=" + signedHeaders + ", " +
                    "Signature=" + signature
    console.log(authorization)

    const curlcmd = 'curl -X POST ' + "https://" + endpoint + canonicalUri
                            + ' -H "Authorization: ' + authorization + '"'
                            + ' -H "Content-Type: application/json"'
```

```
                            + ' -H "Host: ' + endpoint + '"'
                            + ' -H "X-Scf-Cam-Uin: ' + uin + '"'
                            + ' -H "X-Scf-Cam-Timestamp: ' + timestamp.toString() + '"'
                            + " -d '" + payload + "'"

    console.log('curlcmd:',curlcmd)
}

// 密钥参数
// secretId，值为示例的 AKID***************
// secretKey，值为示例的 Gu5t9xGARNpq86cd98joQYCN3*******
// endpoint: 函数uri的host，如 ****-4m1ipprwba.ap-beijing.tencentscf.com
// uin:主账号UIN

main({
    secretId: '********************',
    secretKey: '********************',
    endpoint: '*******-4m1ipprwba.ap-beijing.tencentscf.com',
    uin: '********'
})
```

Python

```
# -*- coding: utf-8 -*-
import hashlib, hmac, json, os, sys, time
from datetime import datetime

# 密钥参数
# 需要设置环境变量 TENCENTCLOUD_SECRET_ID，值为示例的 AKID**************
secret_id = os.environ.get("TENCENTCLOUD_SECRET_ID")
# 需要设置环境变量 TENCENTCLOUD_SECRET_KEY，值为示例的 Gu5t9xGARNpq86cd98joQYCN3*******
secret_key = os.environ.get("TENCENTCLOUD_SECRET_KEY")

service = "scf"
host = "1253970226-xxxxxxx-cq.scf.tencentcs.com"
endpoint = "https://" + host
algorithm = "TC3-HMAC-SHA256"
timestamp = int(time.time())
date = datetime.utcfromtimestamp(timestamp).strftime("%Y-%m-%d")
params = {"Limit": 1, "Filters": [{"Values": [u"未命名"], "Name": "instance-name"}]}

# ************* 步骤 1：拼接规范请求串 *************
http_request_method = "POST"
canonical_uri = "/"
canonical_querystring = ""
ct = "application/json"
payload = json.dumps(params)
canonical_headers = "content-type:%s\nhost:%s\n" % (ct, host)
signed_headers = "content-type;host"
hashed_request_payload = hashlib.sha256(payload.encode("utf-8")).hexdigest()
canonical_request = (http_request_method + "\n" +
                     canonical_uri + "\n" +
                     canonical_querystring + "\n" +
                     canonical_headers + "\n" +
                     signed_headers + "\n" +
                     hashed_request_payload)
print(canonical_request)

# ************* 步骤 2：拼接待签名字符串 *************
credential_scope = date + "/" + service + "/" + "tc3_request"
```

```
hashed_canonical_request = hashlib.sha256(canonical_request.encode("utf-8")).hexdigest()
string_to_sign = (algorithm + "\n" +
                  str(timestamp) + "\n" +
                  credential_scope + "\n" +
                  hashed_canonical_request)
print(string_to_sign)


# ************* 步骤 3：计算签名 *************
# 计算签名摘要函数
def sign(key, msg):
    return hmac.new(key, msg.encode("utf-8"), hashlib.sha256).digest()
secret_date = sign(("TC3" + secret_key).encode("utf-8"), date)
secret_service = sign(secret_date, service)
secret_signing = sign(secret_service, "tc3_request")
signature = hmac.new(secret_signing, string_to_sign.encode("utf-8"), hashlib.sha256).hexdigest()
print(signature)

# ************* 步骤 4：拼接 Authorization *************
authorization = (algorithm + " " +
                 "Credential=" + secret_id + "/" + credential_scope + ", " +
                 "SignedHeaders=" + signed_headers + ", " +
                 "Signature=" + signature)
print(authorization)

print('curl -X POST ' + endpoint
      + ' -H "Authorization: ' + authorization + '"'
      + ' -H "Content-Type: application/json"'
      + ' -H "Host: ' + host + '"'
      + ' -H "X-Scf-Cam-Uin: ' + uin + '"'
      + ' -H "X-Scf-Cam-Timestamp: ' + str(timestamp) + '"'
      + " -d '" + payload + "'")
```

```
algorithm := "TC3-HMAC-SHA256"
service := "scf"
var timestamp int64 = time.Now().Unix()

// step 1: build canonical request string
httpRequestMethod := "POST"
canonicalURI := "/"
canonicalQueryString := ""
canonicalHeaders := fmt.Sprintf("content-type:%s\nhost:%s\n",
    "application/json", host)
signedHeaders := "content-type;host"
payload := `{"Limit": 1, "Filters": [{"Values": ["\u672a\u547d\u540d"], "Name": "instance-name"}]}`
hashedRequestPayload := sha256hex(payload)
canonicalRequest := fmt.Sprintf("%s\n%s\n%s\n%s\n%s\n%s",
    httpRequestMethod,
    canonicalURI,
    canonicalQueryString,
    canonicalHeaders,
    signedHeaders,
    hashedRequestPayload)
fmt.Println("canonicalRequest => ", canonicalRequest)

// step 2: build string to sign
date := time.Unix(timestamp, 0).UTC().Format("2006-01-02")
credentialScope := fmt.Sprintf("%s/%s/tc3_request", date, service)
```

```
hashedCanonicalRequest := sha256hex(canonicalRequest)
string2sign := fmt.Sprintf("%s\n%d\n%s\n%s",
    algorithm,
    timestamp,
    credentialScope,
    hashedCanonicalRequest)
fmt.Println("string2sign ==>", string2sign)

// step 3: sign string
secretDate := hmacsha256(date, "TC3"+secretKey)
secretService := hmacsha256(service, secretDate)
secretSigning := hmacsha256("tc3_request", secretService)
signature := hex.EncodeToString([]byte(hmacsha256(string2sign, secretSigning)))
fmt.Println(signature)

// step 4: build authorization
authorization := fmt.Sprintf("%s Credential=%s/%s, SignedHeaders=%s, Signature=%s",
    algorithm,
    secretId,
    credentialScope,
    signedHeaders,
    signature)
fmt.Println(authorization)

curl := fmt.Sprintf(`curl -X %s https://%s -H "Authorization: %s" -H "Content-Type: application/json" -H
"Host: %s"  -H "X-Scf-Cam-Uin: %s" -H "X-Scf-Cam-Timestamp: %d" -d '%s'`,
    httpRequestMethod, host, authorization, host, uin, timestamp, payload)
fmt.Println(curl)
```

客户端调用参数

在 URL 请求的 Header 中需要添加以下参数：

参数	描述
Authorization	签名相关参数，必传。示例： `TC3-HMAC-SHA256 Credential=AKID***************/2019-02-25/scf/tc3_request, SignedHeaders=content-type;host;xxx, Signature=be4f67d323c78ab9acb7395e43c0dbcf822a9cfac32fea2449a7bc7726b770a3`
X-Scf-Cam-Timestamp	生成签名所使用的时间戳，必传。
X-Scf-Cam-Uin	主账户 Uin，必传。
X-Scf-Cam-Token	如果使用临时密钥生成签名，需要传入 token 信息。

服务器端验证签名

服务端调用 CAM 服务的签名和鉴权，请参见 访问管理 。

自定义域名
配置自定义域名

最近更新时间：2024-10-17 17:13:01

功能概述

函数平台现已支持“自定义域名”功能，允许用户通过浏览器或其他终端访问您的函数。
通过添加自定义域名，您可以将该域名指向任意一个开启了函数 URL 的函数，或通过路径映射的方式指向多个函数。此外，您还可以为该域名开启 HTTPS 协议、Web 应用防火墙等选项，以确保其安全性。

通过自定义域名访问函数的实现原理

函数平台为每个账户的每个地域分配了一个 CNAME，格式如下：

```
<appid>.<region>.tencentscf.com
```

例如，您的自定义域名为 `test.com`，在 DNS 服务商处添加解析记录解析到该域名，并在函数平台配置路径映射到指定的函数。当您的用户访问您的自定义域名 `test.com` 的指定路径时，经过 DNS 解析，通过 CNAME 被引导到函数平台，并通过路径映射，将请求最终发送到对应的函数，函数响应请求并返回结果给用户。

前提条件

在中国大陆地区提供面向公网服务时，按照国家相关法律法规，您需要先将域名完成备案，然后绑定到服务，从而保障您的用户可通过您的域名访问该服务。中国香港和境外地域的函数绑定的自定义域名不需要备案。

操作步骤

步骤1：添加自定义域名

1. 登录 函数服务控制台 。
2. 在左侧导航栏选择**高级能力 > 自定义域名**，单击**添加自定义域名**。如下图所示：

> **注意：**
> 域名有地域属性，域名只能指向同一地域下的函数。



3. 在**添加自定义域名**内填写已经完成备案的自定义域名。支持单域名（例如 `test.com`），暂不支持泛域名（例如 `*.test.com`）。如下图所示：



4. 在弹窗内，获取**公网 CNAME** 或**内网 CNAME**，前往您的 DNS 解析平台，将您的自定义域名解析到该 CNAME。确认完成解析后，进行下一步。



CNAME 格式如下：

- 公网 CNAME：`<appid>.<region>.tencentscf.com` 示例：`123456.ap-guangzhou.tencentscf.com`
- 内网 CNAME：`<appid>.in.<region>.tencentscf.com` 示例：`123456.in.ap-guangzhou.tencentscf.com`

步骤2：添加路径映射

您可以将不同的路径映射到不同的函数，从而实现不同的请求路径触发执行不同的函数。默认情况下，您只需配置根路径（/）映射到指定的函数的版本或别名即可。



路径映射匹配规则

精确路径：

请求的路径和设置的路径完全一致才可以触发对应的函数。

假设，设置路径为 /a/，对应的命名空间为 n1，对应函数为 f1，对应的版本为1。那么只有来自路径 /a/ 的请求才能触发版本1下的 f1 函数执行，来自路径 /a 或 /a/b 的请求无法触发版本1下的 f1 函数执行。
假设，设置路径为 /a，对应的命名空间为 n1，对应函数为 f1，对应的版本为1。那么只有来自路径 /a 的请求才能触发版本1下的 f1 函数执行，来自路径 /a/ 的请求无法触发版本1下的f1函数执行。

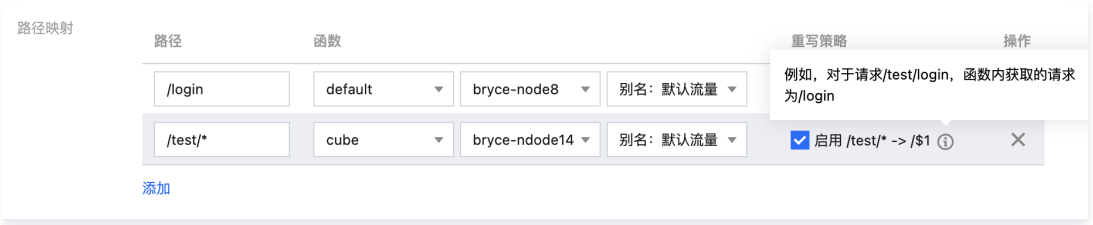
模糊路径：

支持使用通配符（*）设置路径，且通配符（*）只能放到路径的最后；最长前缀匹配原则。
假设，设置路径为 /xxx/*，对应命名空间为 n2，对应函数为 f2，对应版本为1。那么路径前缀为 /login/（例如 /xxx/a、/xxx/b/c/d）的请求都会触发版本1下的 f2 函数执行。

路径重写策略

当您将子路径指向函数，默认情况下，函数内接收到的请求中的 path 是该子路径。例如，以下两种情况的表现是：
1. 精确路径 /home 指向函数 a，则访问 abc.test.com/home 时，函数 a 接收到的请求中的 path 为 /home 。
2. 模糊路径 /test/* 指向函数 b，则访问 abc.test.com/test/login 时，函数 b 接收到的请求中的 path 为 /test/login 。

如果您希望函数接收到的请求可以去除掉子路径，则可开启“重写策略”。对于以上两种情况，启用“重启策略”后，表现是：
1. 精确路径 /home 指向函数 a，则访问 abc.test.com/home 时，函数 a 接收到的请求中的 path 为 / 。
2. 模糊路径 /test/* 指向函数 b，则访问 abc.test.com/test/login 时，函数 b 接收到的请求中的 path 为 /login 。



步骤3：（可选）HTTPS 设置

您也可以选择启用 HTTPS 协议访问自定义域名。



启用后，您需要从 SSL 证书服务选择与该域名匹配的证书，即可完成配置。如果证书下拉列表为空，说明您尚未在 SSL 证书服务拥有与该域名匹配的证书，可先前往 SSL 证书服务完成证书的上传或购买。
完成配置后，即可支持使用 HTTP 或 HTTPS 协议访问该域名。
您还可以启用 **强制 HTTPS** 选项，启用后，仅支持 HTTPS 协议访问该域名，所有的 HTTP 协议访问请求都将 301 重定向到 HTTPS 协议。

步骤4：（可选）Web 应用防火墙设置

您还可以选择启用 Web 应用防火墙，支持对请求流量进行 BOT、爬虫、恶意注册等防护，有效保障业务安全稳定运行。该能力会产生一定的费用，详情请参见 Web 应用防火墙 。
配置步骤如下：
1. 登录 腾讯云控制台 。
2. 进入 Web 应用防火墙购买页 。
3. 在 Web 应用防火墙购买页，选择“负载均衡型实例”。如下图所示：



4. 购买完成后，返回云函数控制台，在自定义域名配置页面，勾选 **Web 应用防火墙启用**，在 WAF 实例的下拉列表中选择对应的实例 ID。



步骤5：验证自定义域名

您可以选择通过浏览器测试或通过命令行 curl 测试。访问自定义域名后，验证是否调用了指定的函数。

- **浏览器测试：**

  i. 打开浏览器，输入您的自定义域名，例如 https://abc.test.com。

  ii. 检查是否正确调用了指定的函数，并返回预期的结果。

- **命令行 curl 测试：**

```
curl -v https://abc.test.com
```

检查输出结果，验证是否正确调用了指定的函数，并返回预期的结果。

版本管理
版本管理概述

最近更新时间：2023-03-20 11:32:55

简介

云函数（Serverless Cloud Function，SCF）的版本包含了函数的代码及配置。在实际的开发过程中，可通过发布版本固定函数代码及配置内容，减少影响业务系统的问题因素。

相关概念

最近版本/最新版本（$LATEST）

函数在创建后缺省具有一个最近版本/最新版本（$LATEST），仅 $LATEST 版本的配置和代码支持修改。发布时以 $LATEST 版本的配置和代码作为基础进行发布，生成新版本。

相关操作

版本可以具有的操作包括：

- 查看版本
- 发布版本
- 使用版本

以下视频将为您介绍如何查看版本、发布版本及使用版本：
观看视频

查看版本

最近更新时间：2022-12-23 14:53:13

操作场景

当您需要查看某个函数的版本配置、代码等信息时，可参考本文档进行操作。

操作步骤

1. 登录 Serverless 控制台，单击左侧导航栏的函数服务。
2. 在“函数服务”列表页面上方，选择需查看函数所在的地域及命名空间。
3. 单击函数名，进入该函数详情页面。
4. 选择函数详情页面右上角的“版本”下拉列表，单击需查看版本的名称。本文以查看版本 `1` 为例，如下图所示：



即可查看该版本相关信息。如下图所示：



说明

- 切换到版本后，**函数配置**、**函数代码**、**层管理**、**监控信息**及**日志查询**页签将显示为对应版本的内容。各页签的内容详情请参考 查询函数 。
- 切换到非 `$LATEST` 版本后，函数配置及代码保持发布时状态，无法修改。
- 触发器可以在不同版本上进行不同的配置。
- 日志和监控分别显示对应版本的具体调用日志和监控数据。

发布版本

最近更新时间：2022-12-27 14:19:10

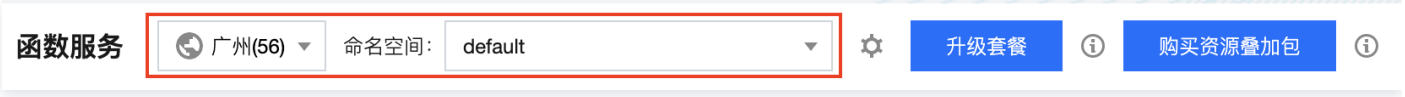
操作场景

在完成云函数的配置、提交代码并通过在线测试后，您可以通过发布版本的方式，固化云函数的版本，避免后续因修改代码和测试引起在线业务错误或执行失败。您可以随时发布版本，云函数任何一次的版本发布都将 `$LATEST` 版本发布为最新版本。

操作步骤

> **说明**
> 云函数在创建时就具备 `$LATEST` 版本的属性。`$LATEST` 版本指向目前可编辑的版本，且始终保持存在及可编辑状态。

1. 登录 Serverless 控制台，单击左侧导航栏的函数服务。
2. 在“函数服务”列表页面上方，选择待查看函数所在的地域及命名空间。如下图所示：



3. 单击函数名，进入函数信息页面。
4. 选择函数信息页面右上角的**操作** > **发布新版本**。如下图所示：



5. 在弹出的“发布新版本”窗口，填写版本描述并单击**提交**即可发布。如下图所示：



> **注意**
> - 提交发布时，云函数平台将会把当前函数 `$LATEST` 版本的配置、代码内容生成副本，并作为版本内容保存。
> - 发布完成后，会生成当次发布的版本号。版本号从1开始随每次发布时递增，当前版本号无上限。

- 发布的版本仅记录及固化当前函数 `$LATEST` 版本的配置及代码，不记录函数的触发器配置。新发布的函数版本无任何触发器。

使用版本

最近更新时间：2025-06-24 17:24:11

版本功能主要用于对于函数配置和代码的固化，避免开发调试及测试时，对业务的影响。发布云函数的版本 后，您可以通过调用指定版本的云函数，来使用版本。

> **说明：**
> `$LATEST` 版本为开发和测试使用的版本，用于代码的进一步开发和调试。

版本的触发器

目前云函数已发布的版本均可以独立绑定触发器。同一函数，版本与版本之间独立，每个触发器都独立触发函数运行。

> **说明：**
> 用户账号下触发器数量有一定限制，详情请参见 配额限制 。如需增加触发器的配额数量（即配额提升），可通过 提交工单 申请。

云 API 触发版本

使用云 API InvokeFunction 接口触发云函数调用时，可通过可选参数 `Qualifier` 指定需要触发的具体版本。如果没有此参数，默认触发 `$DEFAULT` 别名，详情请参见 别名管理 。

别名管理
别名管理相关操作

最近更新时间：2022-12-23 14:45:19

简介

云函数（Serverless Cloud Function，SCF）的别名是指向已绑定函数版本的指针，通过使用别名可以调用已绑定的函数。在实际的开发过程中，别名可以帮助您更好的管理项目版本的更新和回滚。同一个版本的函数可以有一个或多个别名，关于函数版本管理请参见 版本管理概述 。

使用场景

可以通过别名的设置，为函数创建出多个不同的环境（stage）的区分。例如：

- 可通过创建 test、release 别名，并配置触发器指向这些别名，来使得不同的代码和配置生效。
- 可使用别名绑定不同的函数版本，待版本通过测试环境验证后，将正式环境的流量通过路由配置转移到新版本上。流量路由配置的方法请参见 流量路由配置说明 。

默认别名

函数在创建后缺省具有一个默认别名（$DEFAULT），默认别名生成时指向最近版本（$LATEST）。默认别名不可删除或修改名称，但支持流量路由配置。

默认别名的使用

配置触发器和通过云 API 来触发函数时，建议将调用时的 `Qualifier` 参数置为默认别名（$DEFAULT）。

> **说明**
> 通过配置默认别名，可以将触发器和云 API 调用产生的默认流量进行路由控制。

别名的触发器

目前云函数已创建的别名均可以独立绑定触发器。触发器的调用将经过别名，并依据别名的路由配置拉起具体的版本执行。

操作步骤

创建别名

1. 登录 Serverless 控制台，单击左侧导航栏的函数服务。
2. 在“函数服务”列表页面中，单击函数名，进入该函数详情页面。
3. 选择页面右上角的**操作** > **创建别名**。如下图所示：



4. 在弹出的“创建别名”窗口中，参考以下信息进行创建。如下图所示：

> **注意**
> 别名创建完成后，将无法修改别名的名称。



主要参数信息如下：

- **别名名称**：自定义名称。最长60个字符，最短2字符，以字母开头，可包含 `a - z`、`A - Z`、`0 - 9`、`-`、`_`，且需要以数字或字母结尾，例如 `Tencent-cloud_scf` 。
- **别名描述**：自定义描述。最长1000个字符，可包含英文字母、数字、空格、逗号、句号、中文。
- **路由方法及版本权重配置**：详情请参见 流量路由配置 。

5. 单击**提交**即可完成创建。

修改函数版本绑定的别名

1. 选择函数详情页右上角的**操作 > 流量设置**。如下图所示：



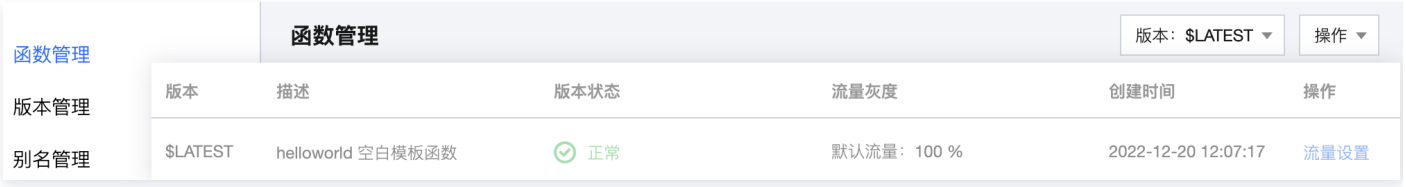
2. 在弹出的“流量设置”窗口中，参考以下信息进行设置。如下图所示：



主要参数信息如下：

- **别名**：在下拉列表中，选择该版本期望绑定的别名。本文以 `test02` 为例。
- **路由方法及版本权重配置**：配置方法详情请参见 流量路由配置，本文以修改 `$LATEST` 版本绑定别名为例：
  - 路由方法选择为**按权重路由**。
  - 版本权重配置为：版本 `$LATEST` 的权重路由为70%，版本 `1` 的权重路由为30%。

3. 单击**提交**即可完成修改，打开版本下拉列表，即可查看修改后效果。如下图所示：



删除别名

> **注意**
> 仅永久删除该别名，不包括底层版本代码和配置。

1. 选择在函数详情页右上角的**操作 > 删除别名**。如下图所示：



2. 在弹出的“删除别名”窗口中，在下拉列表中选择需删除的别名，并单击**提交**即可。本文以删除别名 `test02` 为例。如下图所示：



流量路由配置

最近更新时间：2025-06-13 15:09:02

操作场景

云函数（Serverless Cloud Function，SCF）支持流量路由设置。通过该设置，您可便捷控制函数版本在实际使用场合或环境中的灰度上线或回滚流程，避免一次性上线可能带来的风险。
在创建别名或进行流量配置调整时，可通过控制台控制流量指向两个函数版本，实现流量在版本间按照一定的规则路由。目前支持**按权重随机路由**和**按规则路由**两种路由方案：

- 当您希望任意两个版本按设定的百分比权重进行随机路由时，可进行 按权重随机路由 操作。
- 当您想将包含有某个特定内容的请求路由到某一个版本时，可进行 按规则路由 操作。

操作步骤

按权重随机路由

本文以在创建别名时配置为例。完成创建后，流量将按设定的百分比在两个版本间随机路由。步骤如下：

1. 参考 新建别名 步骤，进入“创建别名”窗口。
2. 在“创建别名”窗口中，参考以下信息进行流量路由配置。如下图所示：



主要参数信息如下：

- **路由方法**：选择**按权重路由**。
- **版本权重设置**：可通过下拉列表选择两个版本，并进行百分比权重配置。

3. 单击**提交**即可完成设置。

按规则路由

使用按规则配置路由时，目前的规则语法包括以下三部分：

匹配 Key

匹配时的取值位置，即通过定位来取值以判断是否命中。
Key 目前支持的写法为 `invoke.headers.[userKey]`，其中 `[userKey]` 部分代表可修改内容。此写法含义为通过匹配 `invoke` 接口调用时，HTTP 请求 headers 中的 userKey 部分来进行匹配。

匹配方法

匹配时通过方法与表达式进行对比，目前支持的匹配方法有 `exact`，`range`。

- `exact`：精确匹配，在使用 `exact` 方法时，匹配表达式需为字符串。通过匹配 key 读取到的值与表达式精确相等时，即为命中规则。
- `range`：范围匹配，在使用 `range` 方法时，匹配表达式需为 (a,b) 或 [a，b] 的写法，其中 a，b 要求为整数。通过匹配 key 读取到的值为整数，且在表达式定义的区间中时，即为命中规则。

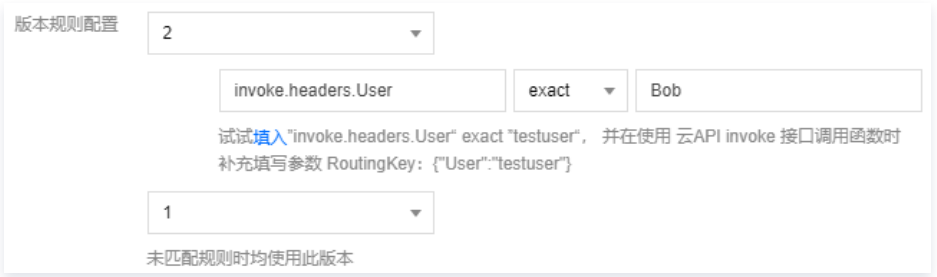
匹配表达式

匹配设定值即为命中，表达式写法可参见 匹配方法 说明。
您可按照以下步骤，配置按规则路由：
1. 参考 新建别名 步骤，进入“创建别名”窗口。
2. 在“创建别名”窗口中，参考以下信息进行流量路由配置，并单击**提交**即可完成配置。如下图所示：



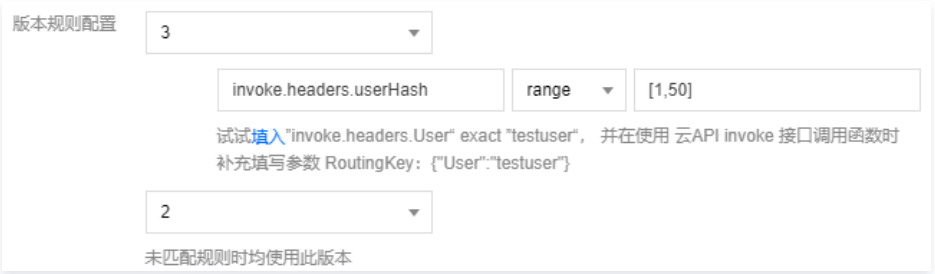
主要参数信息如下：

- **路由方法**：选择**按规则路由**。
- **版本规则设置**：请结合以下示例，按需配置：
  - 例如，您有两个版本（版本2和版本1），并且期望版本2匹配规则设置为 `invoke.headers.User exact Bob`，版本1设置为未命中。可参照下图进行设置：



根据此配置，云函数平台在通过 `invoke` 接口调用函数别名时，若将 `routingKey` 参数设置为 `{"User":"Bob"}`，则此次执行将使用版本2的代码和配置。若未设置 `routingKey` 参数或 `routingKey` 为其他值，则此次执行将使用版本1的代码和配置。

  - 例如，您有两个版本（版本3和版本2），并且期望版本3匹配规则为 `invoke.headers.userHash range [1,50]`，版本2设置为未命中。可参照下图进行配置：



根据此配置，云函数平台在通过 `invoke` 接口调用函数别名时，若将 `routingKey` 参数设置为 `{"userHash":30}`，则此次执行将使用版本3的代码和配置。若未设置 `routingKey` 参数或 `routingKey` 为除了 [1,50] 外的其他值，例如 `{"userHash":80}`，则此次执行将使用版本2的代码和配置。

使用别名实现 SCF 灰度发布

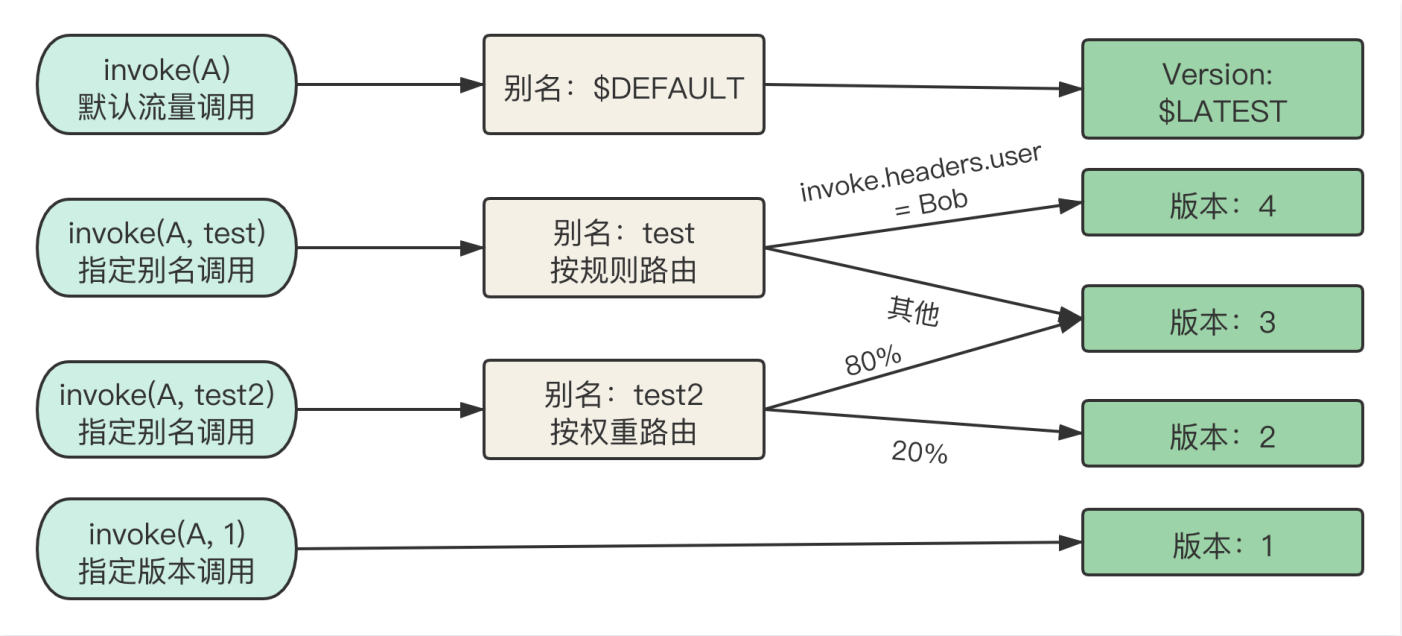
最近更新时间：2022-10-18 15:54:55

概述

使用云函数（Serverless Cloud Function，SCF）的别名可以实现云函数的灰度发布方案，其优势如下：

- 支持用户在多版本间按需分配流量，无需在外部或各触发器位置频繁修改设置。
- 支持流量平滑分配，避免流量漏发。
- 通过相同的流量切换方案，可以在故障时进行版本快速回退。

方案示意图如下所示：



更多关于别名相关操作，请参见 别名管理相关操作 。

名词解释

函数、云函数（Function）

用户创建的云函数。

版本（Version）

云函数版本包含代码及函数配置信息，由具体的数字版本号指明，您可通过发布操作生成具体版本及版本号。仅支持修改最近版本的代码及配置，但任何版本都可被调用。更多云函数版本信息，请参见 版本管理概述 。

最近版本（$LATEST）

可修改代码及配置的版本。创建函数后默认具有最近版本，进行发布时需使用最近版本发布带有数字版本号的具体版本。

别名（Alias）

别名名称可自定义，需使用英文字母开头的字符串指定。别名是可配置指向具体某一个或两个版本的引用。当指向两个版本时，可针对两个版本设置百分比流量。任何别名均可以被调用。

默认流量、默认别名（$DEFAULT）

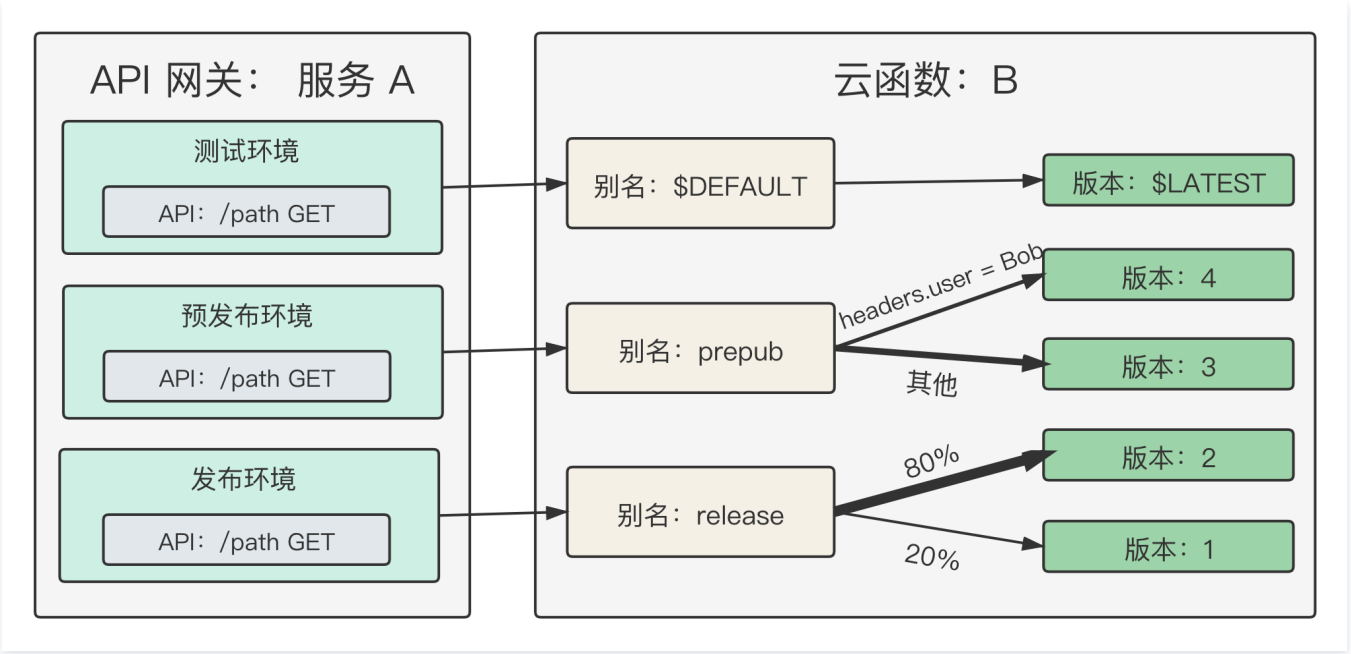
特殊别名，当调用请求未指定任何版本或其他别名时，缺省使用默认别名。默认别名缺省指向最近版本，支持修改版本指向。

方案示例

基于 API 网关触发器的使用示例

背景

- 用户已 创建云函数，且未发布新版本及创建别名。
- 用户期望区分测试环境、预发布环境和发布环境。云函数需在每个阶段测试后，再进入下一阶段。且期望发布时灰度流量，以确保平稳过渡上线期。
  总体方案示意图如下：



初始配置过程

1. 创建别名：
在云函数 B 中创建别名 release、prepub，可暂时指向 $LATEST 版本。
2. 创建 API 网关：
在 API 网关中创建服务 A，配置 API 指向函数 B 的别名 release，并发布到 API 服务的 release stage 中。如何创建并发布 API 请参见 API 创建 及 API 发布 。
3. 修改 API 配置：
a. 指向函数 B 的别名 prepub，并发布到 API 服务的 prepub stage 中。
b. 指向函数 B 的默认流量，并发布到 API 服务的 dev stage 中。
至此已分离测试环境、预发布环境和发布环境，但三个环境均指向 $LATEST 版本。API 网关的配置已完成，后续无需再次修改及发布 API 网关配置。

开发测试发布过程：持续开发、测试、发布、上线

1. 发布版本：
在云函数上持续开发并依次发布版本1、2、3、4。假设版本1已在发布环境，版本2在预发布环境测试，版本3和版本4在测试环境测试。
2. 开发需要测试的最近版本：
配置 $DEFAULT 别名指向 $LATEST 版本，开发人员可基于此版本持续地进行开发，开发完成后可以发布版本。
3. 预发布环境测试：
假设版本3可进入预发布环境时，配置函数 B 的 prepub 别名指向版本3，即可在预发环境进行测试和体验。
4. 预发布环境按用户灰度：
假设版本4可进入预发布环境，需要将用户 Bob 的调用路由至函数 B 的版本4，将其他用户路由至版本3，将函数 B 的 prepub 别名配置为按规则路由，内容为 `invoke.headers.User exact Bob` 。如何按规则路由，请参见 按规则路由 。
5. 预发布环境至发布环境灰度发布：
假设版本2已经在预发布环境完成体验可以上线时，将函数 B 的 release 别名的流量配置逐渐从版本1切换至版本2，并在灰度的过程中持续观察。如何配置别名的流量，请参见 云函数流量路由配置 。
6. 发布过程中持续监控：
通过监控及日志查看灰度过程，版本2的流量是否正常上涨，版本1的流量是否正常下降，监控发布过程中的各版本错误情况及总体错误情况。

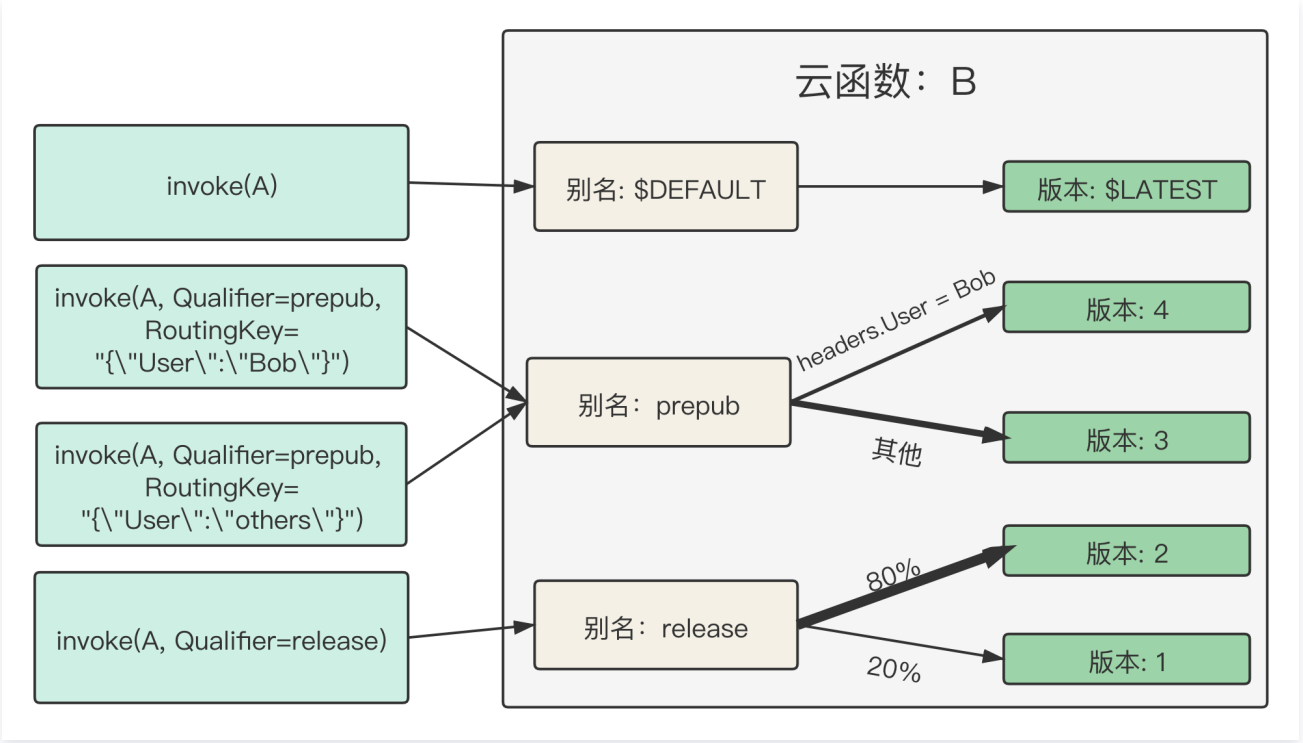
回滚过程：发布有故障时及时回滚

假设版本2在上线时有故障，需回滚至之前版本，则修改云函数 B 的 release 别名流量全部指向版本1即可。

基于云 API invoke 接口的使用示例

背景

- 用户已 创建云函数，且未发布新版本及创建别名。
- 用户直接使用 API 或 SDK 运行云函数。
- 用户期望区分测试环境、预发布环境和发布环境。云函数需在每个阶段测试后，再进入下一阶段。且期望发布时灰度流量，以确保平稳过渡上线期。
  总体方案示意图如下：



初始配置过程

在云函数 B 中创建别名 release、prepub，可暂时指向 $LATEST 版本。

开发测试发布过程：持续开发、测试、发布、上线

1. 发布版本：
在云函数上持续开发并依次发布版本1、2、3、4。假设版本1已在发布环境，版本2在预发布环境测试，版本3和版本4在测试环境测试。
2. 开发需要测试的最近版本：
配置 $DEFAULT 别名指向 $LATEST 版本，开发人员可基于此版本持续地进行开发，开发完成后可以发布版本。
3. 预发布环境测试：
假设版本3可进入预发布环境时，配置函数 B 的 prepub 别名指向版本3，即可在预发环境进行测试和体验。
4. 预发布环境按规则路由：
假设版本4可进入预发布环境，开发人员可配置云函数 B 的按规则路由，自定义传入的 `key` 和 `value`，将其指向版本4。在 `invoke` 接口时，将键值对以 `json` 格式存入参数 `RoutingKey` 中，若 `RoutingKey` 中存在符合规则的键值对，则路由到版本4。如何按规则路由请参见 按规则路由 及 通过 API 运行函数 。
5. 预发布环境至发布环境灰度发布：
假设版本2已经在预发布环境完成体验可以上线时，将函数 B 的 release 别名的流量配置逐渐从版本1切换至版本2，在灰度的过程中持续观察。如何配置别名的流量，请参见 云函数流量路由配置 。
6. 发布过程中持续监控：
通过监控及日志查看灰度过程，版本2的流量是否正常上涨，版本1的流量是否正常下降，监控发布过程中的各版本错误情况及总体错误情况。

回滚过程：发布有故障时及时回滚

假设版本2在上线时有故障，需回滚至之前版本，则修改云函数 B 的 release 别名流量全部指向版本1即可。

Serverless Cloud Framework 的使用示例

在使用 Serverless Cloud Framework 时，可以通过 stage 区分测试环境、预发布环境和发布环境。在发布环境灰度时，可使用以下命令实现逐步过渡。详细操作步骤请参见 使用 tencent-express 组件部署 express 网站 。

```
scf deploy --inputs.traffic=0.1 #部署并切换10%流量到$latest版本上
scf deploy --inputs.traffic=1.0 #部署并切换100%流量到$latest版本上
scf deploy --inputs.traffic=1.0 #部署并切换100%流量到$latest版本上
```

权限管理
权限管理概述

最近更新时间：2023-07-05 09:44:01

简介

腾讯云云函数 SCF 通过 访问管理（Cloud Access Management，CAM）来实现权限管理。CAM 是腾讯云提供的权限及访问管理服务，主要用于帮助客户安全管理腾讯云账户下的资源的访问权限。用户可以通过 CAM 创建、管理和销毁用户（组），并使用身份管理和策略管理控制其他用户使用腾讯云资源的权限。

SCF 支持管理的权限

SCF 用户可以通过主账号给子账号或者协作者赋予不同的权限。当前 SCF 支持的权限粒度如下：

服务	策略语法	云 API	控制台	授权粒度	临时证书
云函数	✔	✔	✔	资源级	✔

当前 SCF 支持的云 API 接口如下：

接口名称	描述	级别
ListFunctions	获取账号下的函数列表	账号级
GetAccount	获取账号下的限额配置	账号级
CreateFunction	新建一个新函数	资源级
DeleteFunction	删除指定的函数	资源级
InvokeFunction	触发函数，分为同步和异步触发	资源级
UpdateFunction	更新函数，包括配置和/或代码	资源级
SetTrigger	对指定函数配置触发器	资源级
DeleteTrigger	删除指定函数的触发器	资源级
GetFunction	获取指定函数的配置信息	资源级
ListVersion	获取指定函数的版本信息	资源级
GetFunctionLogs	获取指定函数的日志信息	资源级

角色与授权

SCF 通过使用访问管理 CAM 的角色能力，完成服务和用户资源间的权限打通。SCF 的角色分为**配置角色**和**运行角色**，您可以通过使用配置角色使 SCF 在服务配置流程中访问用户资源；也可以通过使用运行角色，为运行代码申请临时授权，便于代码通过角色的授权机制实现权限打通和资源访问。

版权所有：腾讯云计算（北京）有限责任公司

角色与策略

最近更新时间：2024-10-15 15:17:21

相关概念

角色

角色（Role）是腾讯云 访问管理（Cloud Access Management，CAM）提供的拥有一组权限的虚拟身份，角色也可被授予策略，主要用于对 角色载体 授予腾讯云中服务、操作和资源的访问权限，这些权限附加到角色后，通过将角色赋予腾讯云的服务，允许服务代替用户完成对授权资源的操作。腾讯云云函数 SCF 的角色分为**配置角色**和**运行角色**，您可以通过使用配置角色使 SCF 在服务配置流程中访问用户资源；也可以通过使用运行角色，为运行代码申请临时授权，便于代码通过角色的授权机制实现权限打通和资源访问。

策略

策略 是定义和描述一条或多条权限的语法规范。CAM 支持两种类型的策略，预设策略和自定义策略。预设策略是由腾讯云创建和管理的一些常见的权限集合，如超级管理员、云资源管理员等，这类策略只读不可写。自定义策略是由用户创建的更精细化的描述对资源管理的权限集合。预设策略不能具体描述某个资源，粒度较粗，而自定义策略可以灵活的满足用户的差异化权限管理需求。

权限

权限 是描述在某些条件下允许或拒绝执行某些操作访问某些资源。默认情况下，主账号是资源的拥有者，拥有其名下所有资源的访问权限。子账号没有任何资源的访问权限。资源创建者不自动拥有所创建资源的访问权限，需要资源拥有者进行授权。

操作场景

您在创建云函数 SCF 时，可能会操作部分 SCF 以外的云产品，不同的操作可能需要不同的权限，例如 COS 触发器创建和删除所需的 COS 权限、API 网关触发器创建和删除所需的 API 网关权限、COS 代码文件的 zip 包读取权限等，通过角色的配置和选择可以实现授权。

配置角色

配置角色用于提供 SCF 配置对接其他云上资源的相关权限，在已关联策略的权限范围内访问您的其他云服务资源，包括但不限于代码文件访问、触发器配置。配置角色的预设策略可支持函数执行基本操作，基本覆盖了 SCF 常用场景所需要的权限。

角色详情

SCF 默认的配置角色为 `SCF_QcsRole`，其角色详情如下：

- 角色名：`SCF_QcsRole`
- 角色载体：`产品服务-scf.qcloud.com`
- 角色描述：SCF 默认配置角色。该服务角色用于提供 SCF 配置对接其他云上资源的权限，包括但不限于代码文件访问、触发器配置。配置角色的预设策略可支持函数执行的基本操作。
- 角色已关联策略：此角色所拥有 `QcloudAccessForScfRole` 策略，具备以下功能：
  - 配置 COS 对象存储触发器时向 Bucket 配置中写入触发配置信息。
  - 读取 COS 对象存储 Bucket 中的触发器配置信息。
  - 在使用 COS 对象存储更新代码时，从 Bucket 完成代码 zip 包的读取操作。
  - 配置 API 网关触发器时，完成 API 网关的服务、API 创建，以及服务发布等操作。
  - 配置和使用日志服务 CLS 的读写访问等操作。
  - 配置和使用消息队列 CMQ 的读写访问等操作。
  - 配置和使用消息队列 Ckafka 的创建、列表等操作。

> ⚠ **注意：**
> 您可以前往 CAM 控制台 查看并修改当前配置角色 `SCF_QcsRole` 所关联的策略，但修改角色的关联策略可能会造成 SCF 无法正常执行等问题，故不建议修改。

服务授权

1. 如果您是首次使用 SCF，打开 Serverless 控制台 时会提示您进行服务授权。如下图所示：



2. 选择**前往访问管理**进入**角色管理**页面，单击**同意授权**确认授权。如下图所示：



3. 确认授权后，将会为您自动创建角色 `SCF_QcsRole` 。可在 角色 中查看。如下图所示：

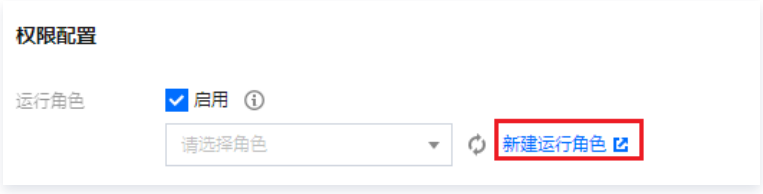


运行角色

运行角色服务于用户代码，角色载体为 `产品服务-scf.qcloud.com` 。用户为函数添加对应的运行角色后，SCF 在运行角色已关联策略的权限范围内为用户的运行代码申请临时授权，便于代码通过角色的授权机制实现权限打通和其他云上资源访问。
以 `SCF_QcsRole` 为例，用户也可以选择 `SCF_QcsRole` 作为函数的运行角色，这意味着将 `SCF_QcsRole` 关联策略对应的权限授权给 SCF，使 SCF 获得为用户代码申请访问其他云上资源的权利。

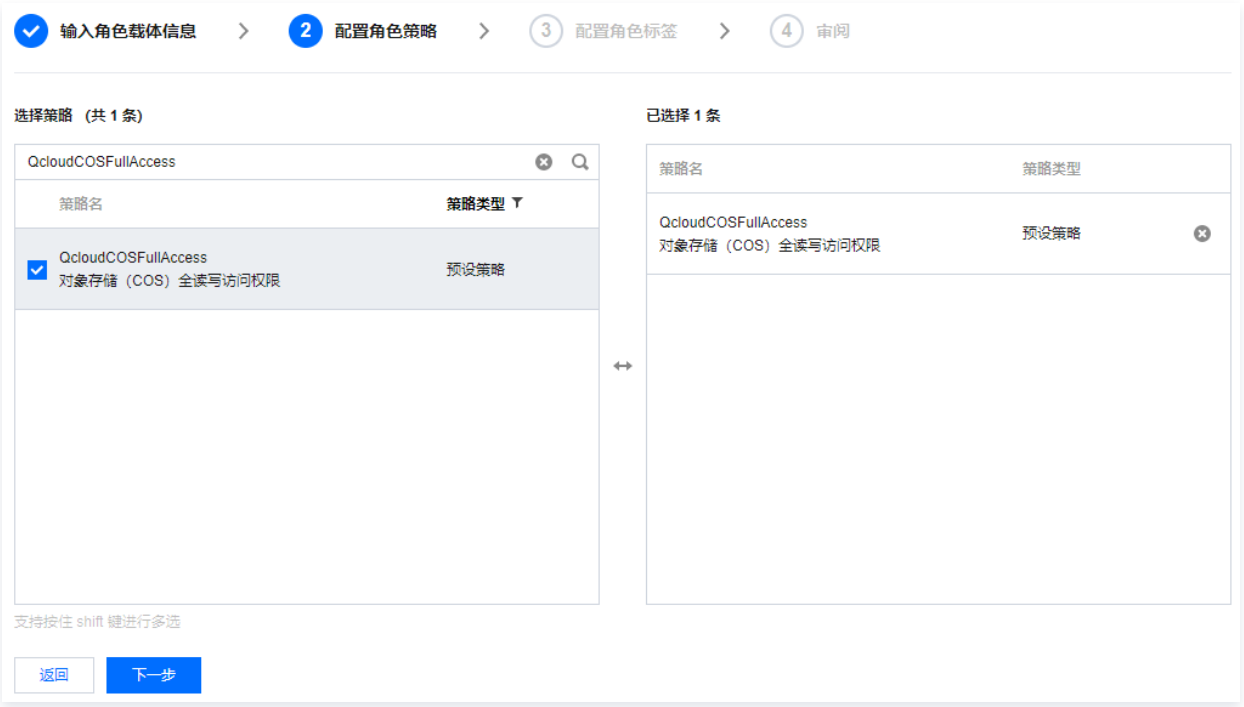
创建运行角色

1. 登录 Serverless 控制台，单击左侧导航栏的函数服务。
2. 在**函数服务**列表页面，单击需创建运行角色的函数名，进入函数配置页。
3. 选择函数配置页右上角的**编辑**，勾选“运行角色”中的**启用**，并单击**新建运行角色**。如下图所示：

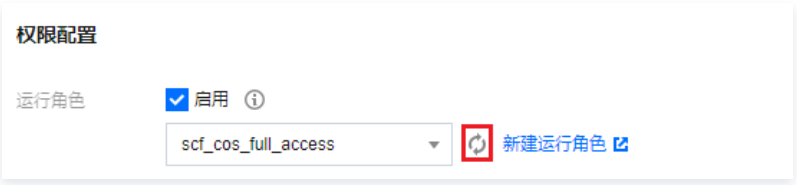


4. 在“输入角色载体信息”步骤中勾选**云函数（scf）**，并单击**下一步**。
5. 在“配置角色策略”步骤中，选择函数所需策略并单击**下一步**。如下图所示：

> **说明：**
> 本文以选择 `QcloudCOSFullAccess` 对象存储（COS）全读写访问权限为例，请根据实际需求进行选择。



6. 按需配置角色标签后，在“审阅”步骤中填写角色名称，并单击**完成**。本文以 `scf_cos_full_access` 角色名称为例。
7. 返回函数配置页，单击“运行角色”右侧的 ，即可在下拉列表中选择刚创建的运行角色。如下图所示：



> **注意：**
> 在为运行角色添加策略时，除了选择预置策略外，还可以通过自定义策略的方式做更细粒度的权限划分，SCF 的策略语法遵循 CAM 的 语法结构 和 资源描述方式，策略语法以 JSON 格式为基础，具体可参考 SCF 策略语法 。

获取运行角色临时密钥信息

在函数运行时，SCF 服务将会使用选定的运行角色完成临时 SecretId、SecretKey、SessionToken 的申请。

- **对于非镜像创建的函数：**
  将以环境变量的形式将相关内容传递到运行环境中。如下图所示：

```
TENCENTCLOUD_SECRETID: 'AKIDhOeFP                                    3Hm0ji8Qqov9xVj',
TENCENTCLOUD_SECRETKEY: 'r24xDBulcl                                  TxwkE=',
TENCENTCLOUD_SESSIONTOKEN: 'GJmlp                                    yEZE86V9c
```

以 Python 为例，您可以通过在 `main` 函数中增加下述代码将临时密钥信息传递到函数运行环境中，并以环境变量的方式获取。

```
def main_handler(event, context);
    secret_id = os.environ.get('TENCENTCLOUD_SECRETID')
    secret_key = os.environ.get('TENCENTCLOUD_SECRETKEY')
```

```
    token= os.environ.get('TENCENTCLOUD_SESSIONTOKEN')
```

- **对于镜像创建的函数：**
  将以 http header 的形式将相关内容传递到入参 context 中，具体请参见 镜像函数入参说明 。

SCF 策略语法

最近更新时间：2023-03-20 11:23:46

策略语法

创建自定义策略流程可参考 CAM 的 创建自定义策略 。SCF 的策略语法遵循 CAM 的 语法结构 和 资源描述方式 ，策略语法以 JSON 格式为基础，所有资源均可采用下述的六段式描述方式，示例如下：

```
qcs::scf:region:uin/uin-id:namespace/namespace-name/function/function-name
```

> **注意**
> 在配置策略语法时，还需要配合使用 monitor 相关的接口以获得账号下的监控信息，使用方法请参考 策略示例 。

策略示例

```
{
        "version":"2.0",
        "statement":
        [
           {
              "effect":"allow",
              "action":
              [
                "scf:ListFunctions",
                "scf:GetAccountSettings",
                "monitor:*"
              ],
              "resource":["*"]
          },
          {
             "effect": "allow",
             "action":
             [
                "scf:DeleteFunction",
                "scf:CreateFunction",
                "scf:InvokeFunction",
                "scf:UpdateFunction",
                "scf:GetFunctionLogs",
                "scf:SetTrigger",
                "scf:DeleteTrigger",
                "scf:GetFunction",
                "scf:ListVersion"
             ],
             "resource":
             [
                 "qcs::scf:ap-guangzhou:uin/******:namespace/default/function/Test1",
                 "qcs::scf:ap-guangzhou:uin/******:namespace/default/function/Test2"
             ]
          }
       ]
}
```

- 操作（action）为需要关联资源的操作时，resource 定义为 `*` ，表示关联所有资源。
- 操作（action）为不需要关联资源的操作时，resource 都需要定义为 `*` 。
- 该示例可以实现子账号拥有主账号下某些函数的操作权限，resource 中的资源描述为主账号下的某个函数。

指定条件

访问策略语言可使您在授予权限时指定条件。例如，限制用户访问来源或限制授权时间等。下面列出了目前支持的条件操作符列表、通用的条件键和示例等信息。

条件操作符	含义	条件名	示例
ip_equal	IP 等于	qcs:ip	{"ip_equal":{"qcs:ip ":"10.121.2.0/24"}}
ip_not_equal	IP 不等于	qcs:ip	{"ip_not_equal":{"qcs:ip ":["10.121.1.0/24", "10.121.2.0/24"]}}
date_not_equal	时间不等于	qcs:current_time	{"date_not_equal":{"qcs:current_time":"2016-06-01T00:01:00Z"}}
date_greater_than	时间大于	qcs:current_time	{"date_greater_than":{"qcs:current_time":"2016-06-01T00:01:00Z"}}
date_greater_than_equal	时间大于等于	qcs:current_time	{"date_greater_than_equal":{"qcs:current_time":"2016-06-01T00:01:00Z"}}
date_less_than	时间小于	qcs:current_time	{"date_less_than":{"qcs:current_time":"2016-06-01T 00:01:00Z"}}
date_less_than_equal	时间小于等于	qcs:current_time	{"date_less_than":{"qcs:current_time":"2016-06-01T 00:01:00Z"}}
date_less_than_equal	时间小于等于	qcs:current_time	{"date_less_than_equal":{"qcs:current_time":"2016-06-01T00:01:00Z"}}

- 限制来访 IP 为 `10.121.2.0/24` 网段内。如下所示：

```
"ip_equal":{"qcs:ip ":"10.121.2.0/24"}
```

- 限制来访 IP 为 `101.226.\*\*\*.185` 和 `101.226.\*\*\*.186` 。如下所示：

```
"ip_equal": {
  "qcs:ip": [
    "101.226.***.185",
    "101.226.***.186"
  ]
}
```

用户策略更新说明

SCF 于2020年4月完善了预设策略权限，针对预设策略 `QcloudSCFFullAccess` 和 `QcloudSCFReadOnlyAccess` 完成修改，针对配置角色 `SCF_QcsRole` 添加了 `QcloudAccessForScfRole` 策略。详情如下：

预设策略 QcloudSCFFullAccess

当前权限如下：

```
{
    "version": "2.0",
    "statement": [
        {
            "action": [
                "scf:*",
                "tag:*",
                "cam:DescribeRoleList",
                "cam:GetRole",
                "cam:ListAttachedRolePolicies",
                "apigw:DescribeServicesStatus",
                "apigw:DescribeService",
```

```
                "apigw:DescribeApisStatus",
                "cmqtopic:ListTopicDetail",
                "cmqqueue:ListQueueDetail",
                "cmqtopic:GetSubscriptionAttributes",
                "cmqtopic:GetTopicAttributes",
                "cos:GetService",
                "cos:HeadBucket",
                "cos:HeadObject",
                "vpc:DescribeVpcEx",
                "vpc:DescribeSubnetEx",
                "cls:getTopic",
                "cls:getLogset",
                "cls:listLogset",
                "cls:listTopic",
                "ckafka:List*",
                "ckafka:Describe*",
                "ckafka:ListInstance",
                "monitor:GetMonitorData",
                "monitor:DescribeBasicAlarmList",
                "monitor:DescribeBaseMetrics",
                "monitor:DescribeSortObjectList",
                "monitor:DescribePolicyConditionList",
                "cdb:DescribeDBInstances"
            ],
            "resource": "*",
            "effect": "allow"
        }
    ]
}
```

预设策略 QcloudSCFReadOnlyAccess

当前权限如下：

```
{
    "version": "2.0",
    "statement": [
        {
            "action": [
                "scf:Get*",
                "scf:List*",
                "ckafka:List*",
                "ckafka:Describe*",
                "monitor:GetMonitorData",
                "monitor:DescribeBasicAlarmList",
                "monitor:DescribeBaseMetrics",
                "monitor:DescribeSortObjectList",
                "cam:GetRole",
                "cam:ListAttachedRolePolicies",
                "vpc:DescribeVpcEx",
                "vpc:DescribeSubnetEx",
                "cls:getLogset",
                "cls:getTopic",
                "cls:listTopic",
                "apigw:DescribeService",
                "cmqtopic:GetTopicAttributes",
                "cmqtopic:GetSubscriptionAttributes",
                "cos:HeadBucket",
                "cos:GetService",
                "cos:GetObject"
```

```
            ],
            "resource": "*",
            "effect": "allow"
        }
    ]
}
```

预设策略 QcloudAccessForScfRole

当前权限如下：

```
{
    "version": "2.0",
    "statement": [
        {
            "action": [
                "cos:GetBucket*",
                "cos:HeadBucket",
                "cos:PutBucket*",
                "apigw:*",
                "cls:*",
                "cos:List*",
                "cos:Get*",
                "cos:Head*",
                "cos:OptionsObject",
                "cmqqueue:*",
                "cmqtopic:*",
                "ckafka:List*",
                "ckafka:Describe*",
                "ckafka:AddRoute",
                "ckafka:CreateRoute"
            ],
            "resource": "*",
            "effect": "allow"
        }
    ]
}
```

预设策略 QcloudAccessForScfRole 具备以下功能：

- 配置 COS 对象存储触发器时，向 Bucket 配置中写入触发配置信息。
- 读取 COS 对象存储 Bucket 中的触发器配置信息。
- 在使用 COS 对象存储更新代码时，从 Bucket 完成代码 zip 包的读取操作。
- 配置 API 网关触发器时，完成 API 网关的服务、API 创建以及服务发布等操作。
- 配置 Ckafka 触发器时，完成创建消费者操作。

版权所有：腾讯云计算（北京）有限责任公司

子用户与授权

最近更新时间：2023-03-02 16:16:16

> **注意**
> 主账户需要在 角色 页面查看是否具有 `SCF_QcsRole` ，如果没有，请按照 角色与策略 中的**服务授权**操作完成授权，否则子用户无法正常使用 Serverless 控制台和通过 SCF 调用其他云上资源。

创建子用户并授予 SCF 的所有操作权限

步骤1：使用主账号创建子用户

1. 登录 访问管理控制台，选择左侧导航栏中的**用户** > **用户列表**。
2. 在**用户列表**页面，选择**新建用户** > **自定义创建**，进入**新建子用户**页面。
3. 在**选择类型**步骤中，选择**可访问资源并接收消息**后，单击**下一步**填写用户信息。
4. 根据页面提示填写并确认信息，单击**完成**，完成自定义创建子用户操作。

> **说明**
> 相关文档参考： 创建子用户 。

步骤2：创建自定义策略

1. 在访问管理控制台的 策略 页面，单击左上角的**新建自定义策略**。
2. 在弹出的选择创建方式窗口中，单击**按策略生成器创建**，进入编辑策略页面。
3. 在“可视化策略生成器”中选择服务的页面，补充以下信息，编辑一个授权声明。
   - **效果**：允许
   - **服务**：云函数
   - **操作**：全部
   - **资源描述**：`*`
   - **条件（可选）**：置空
4. 完成策略授权声明编辑后，单击**下一步**，进入基本信息和关联用户/用户组/角色页面。
5. 在关联用户/用户组/角色页面，补充策略名称和描述信息，可同时关联用户/用户组/角色快速授权。
6. 单击**完成**，完成按策略生成器创建自定义策略的操作。

步骤3：为子用户添加 CAM 只读权限

1. 登录访问管理控制台，进入 用户列表 管理页面。
2. 在用户列表管理页面，选择需要设置权限的子用户。
3. 单击右侧操作列的**授权**。
4. 在弹出的关联策略窗口里勾选 `QcloudCamReadOnlyAccess` 策略。
5. 单击**确定**完成子用户“用户与权限（CAM）只读访问权限”的授权。

完成

以上设置完成后，用户可以登录子账号查看权限。
登录访问管理控制台，选择左侧的导航栏中的概览进入概览页面，即可查看子用户登录地址。

创建子用户并授予 SCF 的部分操作权限

步骤1：使用主账号创建子用户

1. 登录 访问管理控制台，选择左侧导航栏中的**用户** > **用户列表**。
2. 在**用户列表**页面，选择**新建用户** > **自定义创建**，进入**新建子用户**页面。
3. 在**选择类型**步骤中，选择**可访问资源并接收消息**后，单击**下一步**填写用户信息。
4. 根据页面提示填写并确认信息，单击**完成**，完成自定义创建子用户操作。

> **说明**
> 相关文档参考： 创建子用户 。

步骤2：创建自定义策略

1. 在访问管理控制台的 策略 页面，单击左上角的**新建自定义策略**。
2. 在弹出的选择创建方式窗口中，单击**按策略生成器创建**，进入**编辑策略**页面。
3. 复制 SCF 策略语法 中策略示例的代码，在**编辑策略 > JSON** 中对策略内容进行修改。

> **注意**
> resource 中的资源描述，需要替换成主账号的 ID 和主账号下函数名，region 需要和函数保持一致。

4. 单击**下一步**，进入基本信息和关联用户/用户组/角色页面。
5. 在**关联用户/用户组/角色**页面，补充策略名称和描述信息，可同时关联用户/用户组/角色快速授权。
6. 单击**完成**，完成按策略生成器创建自定义策略的操作。

步骤3：为子用户添加 CAM 只读权限

1. 登录访问管理控制台，进入 用户列表 管理页面。
2. 在用户列表管理页面，选择需要设置权限的子用户。
3. 单击右侧操作列的**授权**。
4. 在弹出的关联策略窗口里勾选 `QcloudCamReadOnlyAccess` 策略。
5. 单击**确定**完成子用户“用户与权限（CAM）只读访问权限”的授权。

完成

以上设置完成后，用户可以登录子账号查看权限。在左侧的导航栏中单击 概览 进入概览页面，可以查看子用户登录地址。

> **说明**
> 策略生效后，当前子账号可以看到所有的函数名，但是只能对 resource 中的函数进行操作和查看。

监控与告警管理
监控指标说明

最近更新时间：2023-08-21 09:41:22

腾讯云可观测平台为云函数 SCF 提供以下监控指标：
当前共支持两个维度的监控指标，函数维度的监控指标支持在具体的函数内查看。地域维度的监控指标支持在概览页中选取特定地域，查看该地域下所有函数监控指标的统计值。

指标中文名	指标英文名	指标含义	单位	维度
运行时间	duration	函数/地域级别的运行时间，指用户的函数代码从执行开始到结束的时间，按粒度（1分钟、5分钟）统计求平均。	毫秒	函数 地域
调用次数	invocation	函数/地域级别的请求次数，按粒度（1分钟、5分钟）统计求和。	次	函数 地域
错误次数	error	函数执行后产生的错误请求次数，当前包含客户的错误次数和平台错误次数之和，按粒度（1分钟、5分钟）统计求和。	次	函数 地域
并发执行次数	concurrent_executions	同一时间点并发处理的请求数，按粒度（1分钟、5分钟）统计求和，在函数/地域维度统计求最大值。	次	函数 地域
受限次数	throttle	函数/地域级别被流控的请求次数，达到函数并发后的请求将受限，按粒度（1分钟、5分钟）统计求和。	次	函数 地域
运行内存	mem	函数运行时实际使用的内存，按粒度（1分钟、5分钟）统计求最大值。	MB	函数
时间内存	mem_duration	资源使用量，函数运行时长 × 函数运行所用内存，按粒度（1分钟、5分钟）统计求和。	MB ms	函数
外网出流量	out_flow	在函数内访问外网资源时产生对外的流量，按粒度（1分钟、5分钟）统计求和。	KB	函数
系统内部错误（HTTP 5xx）	syserr	函数执行后返回 5xx 状态码的个数，按粒度（1分钟、5分钟）统计求和。	次	函数
函数错误次数（HTTP 4xx）	http_4xx	函数执行后返回 4xx 状态码的个数，按粒度（1分钟、5分钟）统计求和。	次	函数
正确调用次数（HTTP 2xx）	http_2xx	函数执行后返回 2xx 状态码的个数，按粒度（1分钟、5分钟）统计求和。	次	函数
资源超过限制（HTTP 432）	http_432	函数执行后返回 432 状态码的个数，按粒度（1分钟、5分钟）统计求和。	次	函数
函数执行超时（HTTP 433）	http_433	函数执行后返回 433 状态码的个数，按粒度（1分钟、5分钟）统计求和。	次	函数
内存超过限制（HTTP 434）	http_434	函数执行后返回 434 状态码的个数，按粒度（1分钟、5分钟）统计求和。	次	函数

说明
如需**获取所需监控数据**的相关信息，可访问 云函数监控接口 。

配置告警

最近更新时间：2024-11-07 10:23:52

操作场景

您可以通过 腾讯云可观测平台 对云函数配置告警策略，对函数运行状态进行监控。
目前云函数可以配置告警的监控指标有运行时间、调用次数、错误次数等。全部支持列表请参见 监控指标说明 。同时，告警支持选择接收告警的用户组，选择接收邮件、短信、微信等方式的告警渠道。

操作步骤

1. 登录 Serverless 控制台，选择左侧导航栏中的**函数服务**。
2. 在**函数服务**列表页，单击函数名称，进入函数详情页。
3. 在左侧导航中选择**监控信息**，在监控信息详情页单击**设置告警**。如下图所示：



4. 在**新建告警策略**页面，配置告警策略，配置说明如下：
   - **策略名称**：自定义。
   - **监控类型**：选择 “云产品监控” 。
   - **策略类型**：支持选择“云函数/版本”或“云函数/别名”。
   - **告警对象**：根据实际需求进行设置。若选择 “实例 ID”，其默认地域设置为广州。在不同的地域下，可以查看到相应的函数，请选择需要使用告警策略的函数。
     更加详细的告警策略配置，请参见 新建告警策略 。
5. 单击**完成**。在**告警管理** > 策略管理 中，查看到已配置好的策略。并可根据实际需求，随时选择启停。

视频教程

以下视频将为您介绍如何配置告警及查看日志：
观看视频

查看运行日志

最近更新时间：2024-08-19 18:08:31

操作场景

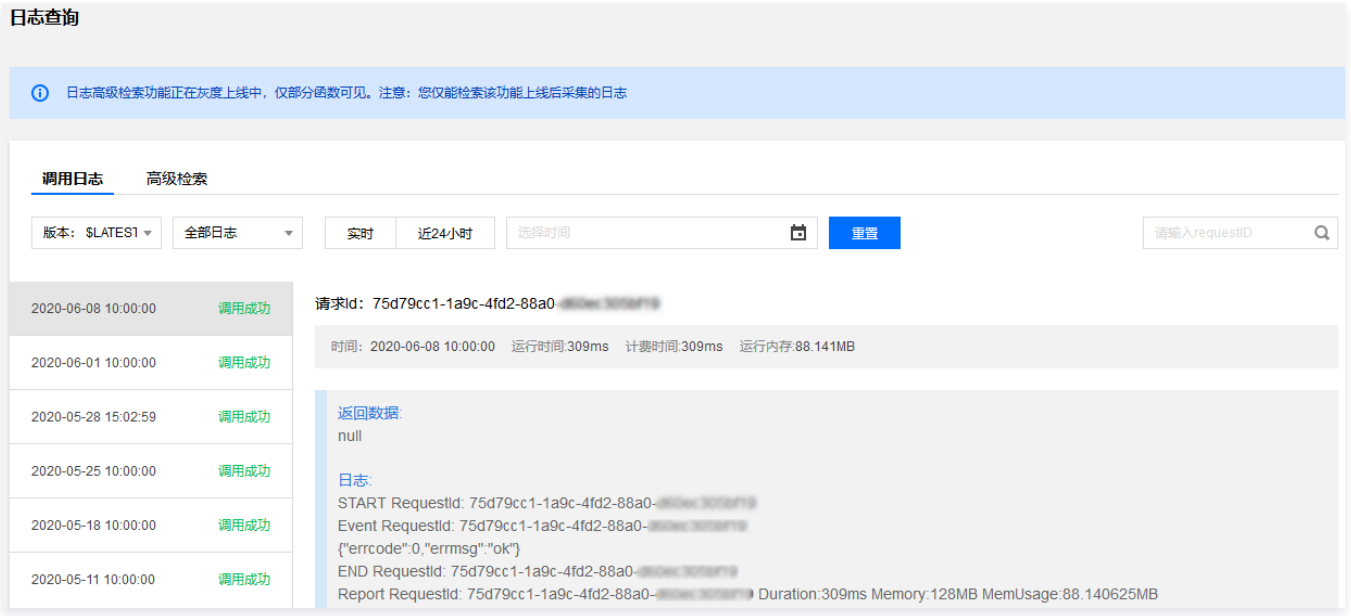
在腾讯云云函数控制台中，您可以查看有关函数运行状态的日志，自定义查找日志的时间范围，或者查看实时日志以及近24小时的日志。目前腾讯云云函数控制台支持查看全部日志、调用成功、调用失败、调用超时、调用超限及代码异常的日志。

操作步骤

您可通过云函数控制台查看云函数日志信息。

控制台查看运行日志

1. 登录 Serverless 控制台，单击左侧导航栏的函数服务。
2. 单击函数名，进入该函数详情页面。
3. 在该函数详情页，选择左侧**日志查询**打开该函数**调用日志**界面。如下图所示：

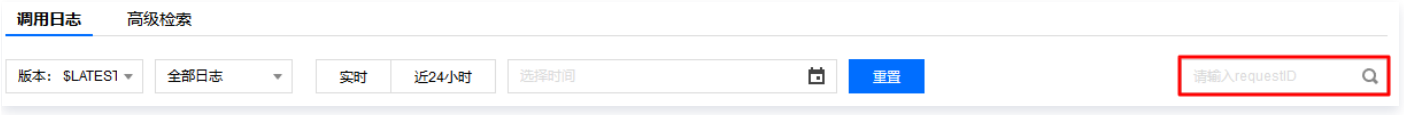


查找运行日志

您可以根据实际需求，查找运行日志。

调用日志

在右上角的搜索框中输入待查看运行日志的 requestID，按 Enter，查看具体某个运行日志。如下图所示：



根据实际需求，在左上角设置自定义查找条件，查看您想看的运行日志。

- **全部日志**：可选调用成功、调用失败日志。
- **选择日期**：可查看当前日期及前6天的运行日志，暂不支持开始时间到结束时间超出24小时的日志检索。
- **实时**：函数当前的运行日志。
- **近24小时**：可查看包含当前时间内24小时的运行日志。

高级检索

云函数日志检索支持关键词搜索，您可以使用查询语法组合关键词进行检索。详情请参见 日志检索教程 。

应用性能管理
应用性能管理概述

最近更新时间：2024-11-15 17:24:13

概述

应用性能监控（Application Performance Management，APM）技术旨在监控和管理应用程序的性能和可用性、检测和诊断复杂应用程序的性能问题，以保证预期的服务水平。目前 APM 技术在云服务器、容器场景下已相对成熟，使用 APM 技术可以实时观测系统的运行状态，通过链路追踪分析每一次的运行和异常，能够快速发现系统中的性能瓶颈，助力解决问题，保障用户体验。

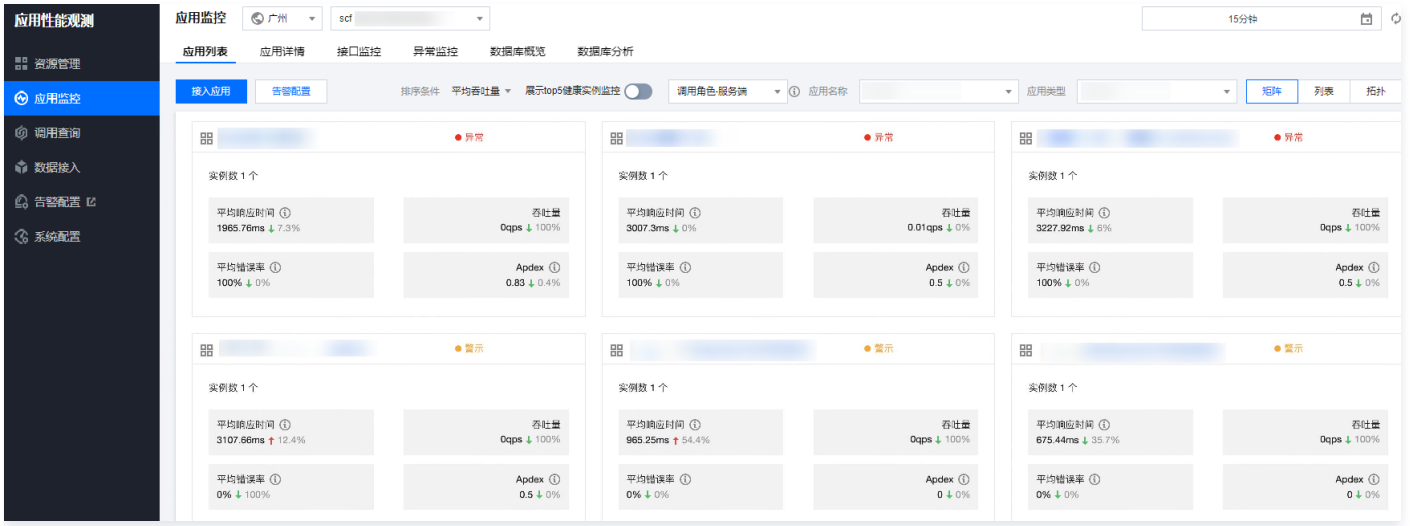
腾讯云 Serverless 与行业 APM 解决方案集成

为了提升 Serverless 用户使用 APM 技术时的可观察性，腾讯云 Serverless 聚焦应用性能管理，与 应用性能监控 、博睿数据 、 听云 等团队在 APM 领域展开更为深入的合作，为企业的开发人员、运维人员以及个人开发者提供更多、更完善的应用级监控。
腾讯云云函数 与 APM 集成，将可观测性的重点从单个系统转为整体系统。在 Serverless 场景下，即从对单个函数的观测转为对 Serverless 应用（包含多个函数及其他服务）的观测，通过丰富的指标监控采集分析、依赖拓扑图、调用链分析、日志分析等能力，为开发者全面的展示整个应用的运行情况。
SCF 现已配置化接入腾讯云应用性能监控 APM 产品，启用即可快速将函数执行基础数据上报至 APM，同时支持用户自定义埋点上报。除腾讯云 APM 产品外，还可以自由选择博睿数据、听云等产品，享受国内优质的 APM 服务。使用 Serverless APM 服务，将会有以下几个方面的优势：

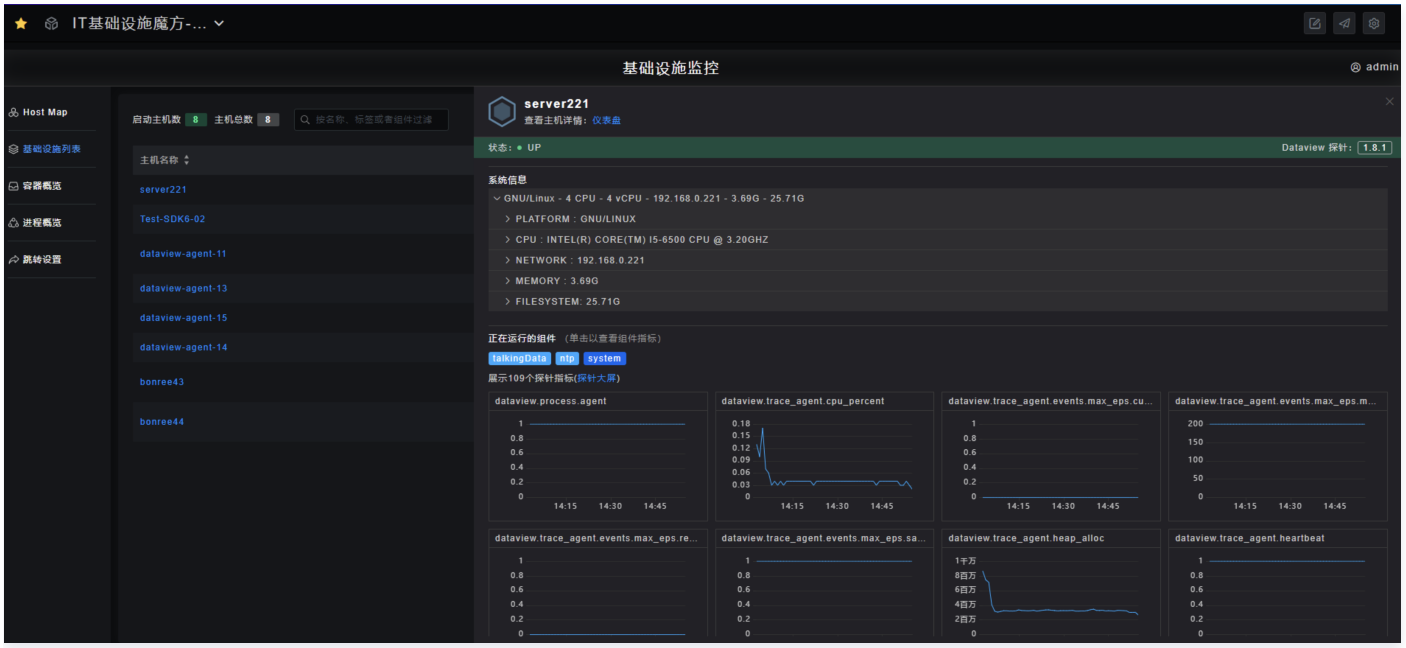
- 丰富的基础监控指标采集与展示。
- 链路追踪能力。
- 调用链分析。

更丰富的基础监控指标采集与展示

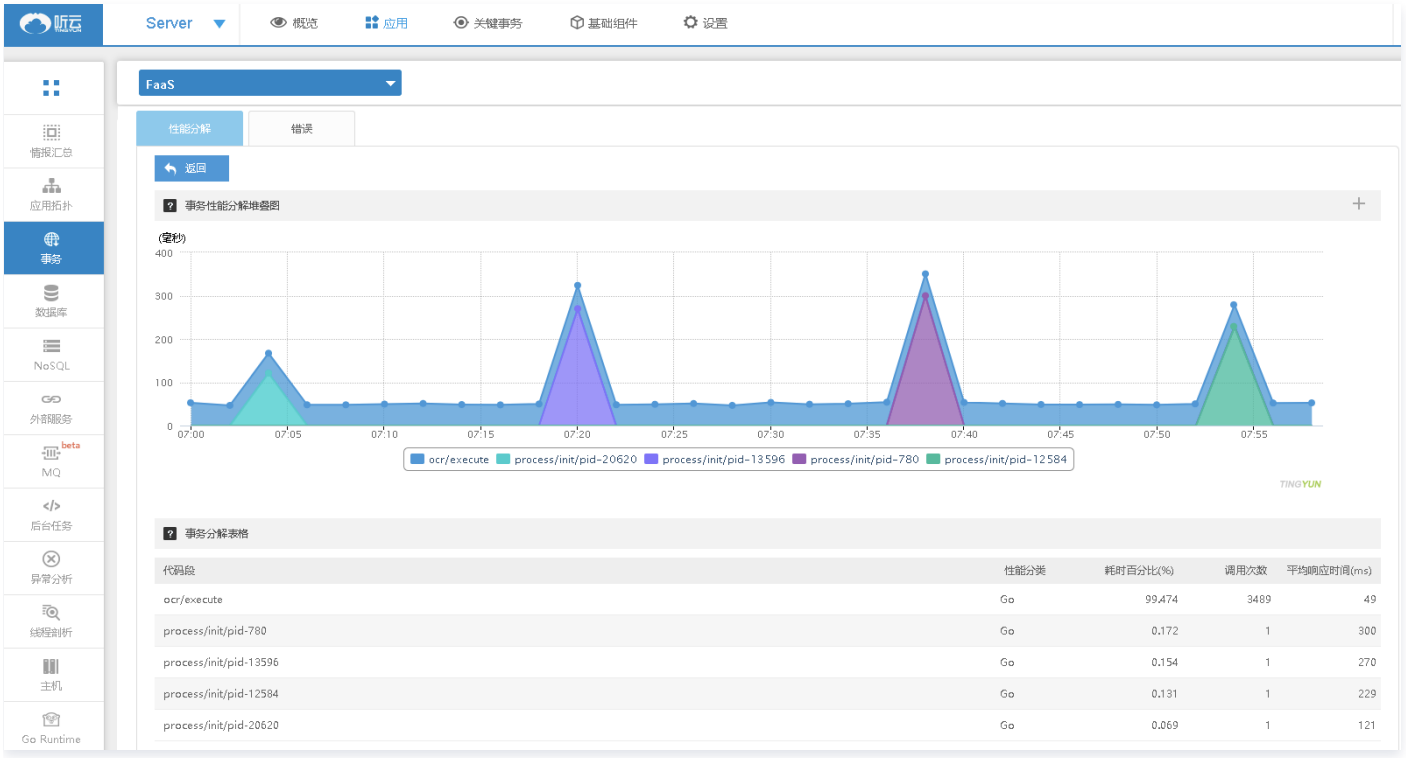
云函数为用户提供了调用次数、运行时间、受限次数等基础监控指标的展示。APM 产品可以补充更为丰富的基础监控指标，例如初始化次数、冷启动时间、超时次数、吞吐率等，更为准确的评估函数及应用的初始化、运行情况。在现有指标的基础上，APM 产品提供多种呈现形式，包括个性化仪表盘等。
同时，用户不仅可以使用服务端监控，也可以使用各 APM 产品的客户端监控，实现在一个平台乃至一个数据大屏上同时监测业务的服务端和客户端。
腾讯云 APM 应用列表页面如下图所示：



博睿 Server 监控页面图如下所示：

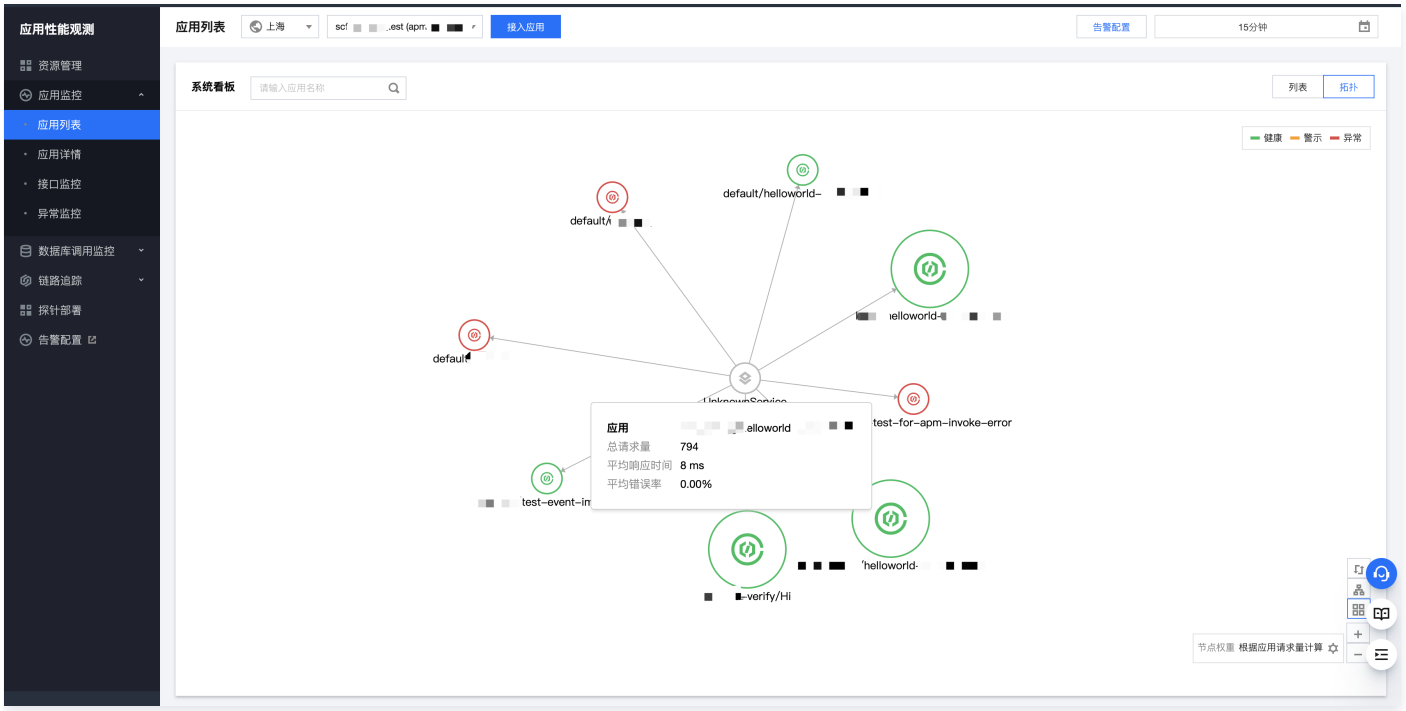


听云针对 Serverless 的性能分解图表如下所示：



链路追踪能力

一个 Serverless 应用可能包含一个或多个函数、API 网关及其他云服务或者第三方服务。凭借链路追踪能力，用户可以根据依赖拓扑图，高效地分析 Serverless 应用中各组件的调用关系及延时情况，可在复杂系统中快速定位性能瓶颈和异常情况。
腾讯云 APM 拓扑图如下所示：



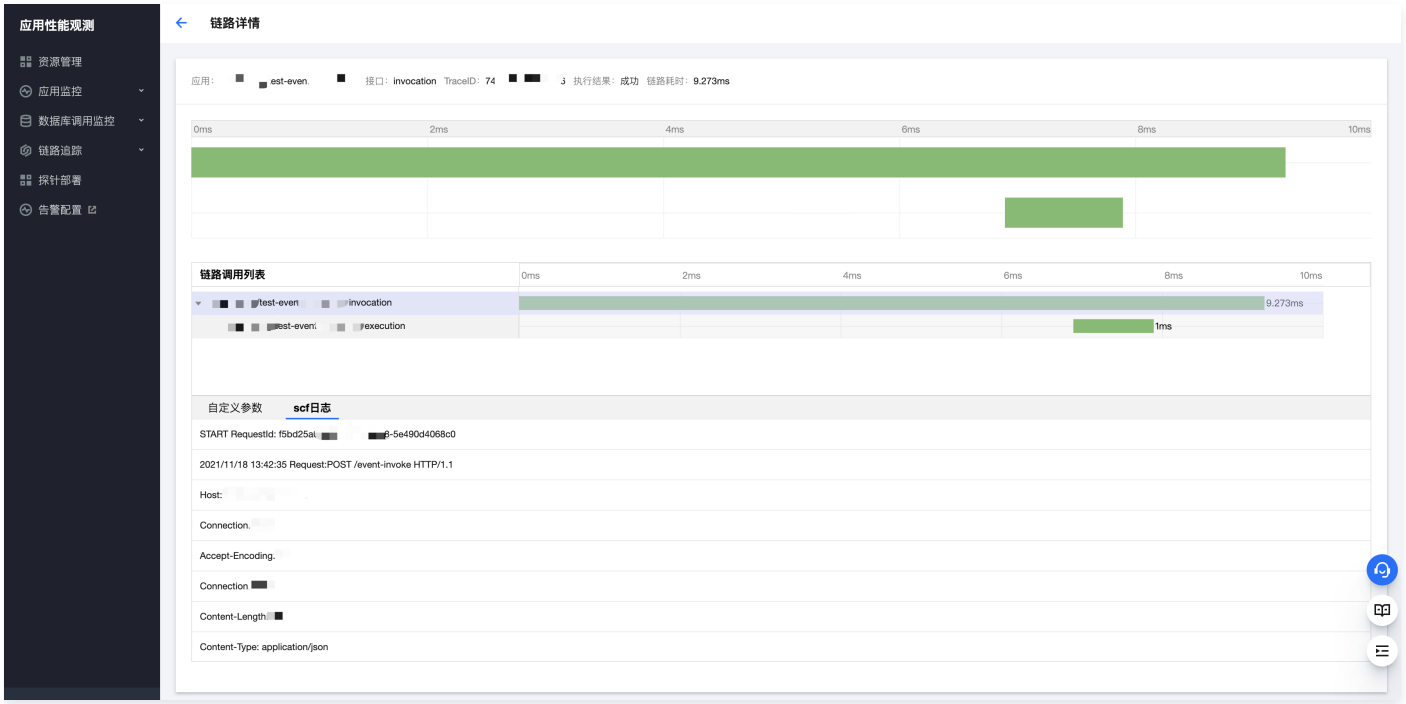
博睿 Server 依赖拓扑图如下所示：



调用链分析

调用链分析可与依赖拓扑图进行配合，使用调用链分析可清晰的展示请求在系统内所有链路的处理情况，还原请求响应过程的完整现场。通过分析链路上每个服务的状态和耗时，可将每个服务的处理耗时、服务间调用的网络耗时以瀑布图的形式直观的展示出来。便于用户进行“异常”请求的问题定位，获得更好更高效的应用体验。

应用性能监控 APM 瀑布图示意图如下：


应用性能观测
资源管理
应用监控
数据库调用监控
链路追踪
探针部署
告警配置
链路详情
应用：
接口：invocation TraceID：74
执行结果：成功 链路耗时：9.273ms
0ms
2ms
4ms
6ms
8ms
10ms
链路调用列表
invocation
9.273ms
execution
1ms
自定义参数
scf日志
START RequestId: f5bd25a
2021/11/18 13:42:35 Request:POST /event-invoke HTTP/1.1
Host:
Connection.
Accept-Encoding.
Connection
Content-Length
Content-Type: application/json

应用性能观测

最近更新时间：2023-08-16 20:14:41

简介

应用性能监控（Application Performance Management，APM）是一款应用性能管理平台，基于实时的多语言应用探针全量采集技术，为您提供分布式应用性能分析和故障自检能力，全方位保障系统的可用性和稳定性。协助您在复杂的业务系统快速定位性能问题，降低 MTTR（平均故障恢复时间）。实时了解并追踪应用性能情况，提升用户体验。
APM 基于 OpenTracing 开源协议，支持多种主流框架和编程语言，为您提供应用性能观测一站式解决方案。
SCF 已经接入 APM 产品，开启函数应用性能观测后，SCF 将使用基于 OpenTracing 的 Jaeger 实现将函数运行总耗时、冷启动耗时、执行耗时三段关键时间上报至 APM：

- **函数运行总耗时**：作为父分段上报，对应 APM 链路中 `invocation` 接口，表示函数从接收到调用命令开始到函数执行完成总耗时。
- **冷启动耗时**：作为函数运行总耗时的子分段上报，对应 APM 链路中 `initialization` 接口，表示函数从接收到调用命令开始，到实例准备完成、函数初始化逻辑执行完成耗时。（该分段仅出现在冷启动调用请求中）
- **执行耗时**：作为函数运行总耗时的子分段上报，对应 APM 链路中 `execution` 接口，表示入口函数执行耗时（事件函数）或完成9000端口监听后每次执行耗时（Web 函数和镜像函数）。

除此之外，还可以根据业务需要自定义埋点进行上报。

> **注意**
> APM 为独立计费产品，APM 计费方式请参考 计费概述 。

权限说明

1. 首次使用请按照操作步骤引导完成授权。子账号如无权限操作，请联系主账号管理员完成授权流程。
2. 为保证 APM 数据正常查看，子账号下至少拥有应用性能观测 APM 只读权限 `QcloudAPMReadOnlyFullAccess` 。主账号为子账号授权方法请参见 授权管理 。

设置应用性能观测

1. 登录 Serverless 控制台，单击左侧导航栏中的**函数服务**。
2. 在页面上方选择地域，单击需要进行应用性能观测配置的函数名。
3. 在**函数配置**页面，选择右上角的**编辑**，勾选**启用**应用性能观测。



4. APM 的资源单元为业务系统，请选择数据上报的地域并选择对应的业务系统。新建业务系统请参考 应用性能观测资源管理 。



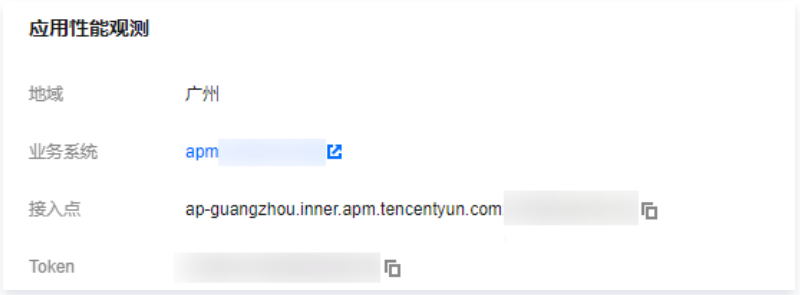
> **注意**

- 建议选择与函数所在地域相同的地域，如需跨地域上报，请在函数网络配置中启用公网访问。（函数通过公网上报 APM 可能会产生额外的费用，请按需使用）
- 业务系统选择完成后，会展示业务系统对应的接入点和 Token 信息，供业务代码自定义上报使用。

5. 单击**保存**完成函数应用性能观测配置。配置更新后对函数 $LATEST 版本即刻生效，已发布版本配置不会变更。

查看函数应用性能观测配置

1. 登录 Serverless 控制台，单击左侧导航栏中的**函数服务**。
2. 在页面上方选择地域，并单击函数名，即可在“函数配置”中查看当前应用性能观测配置。

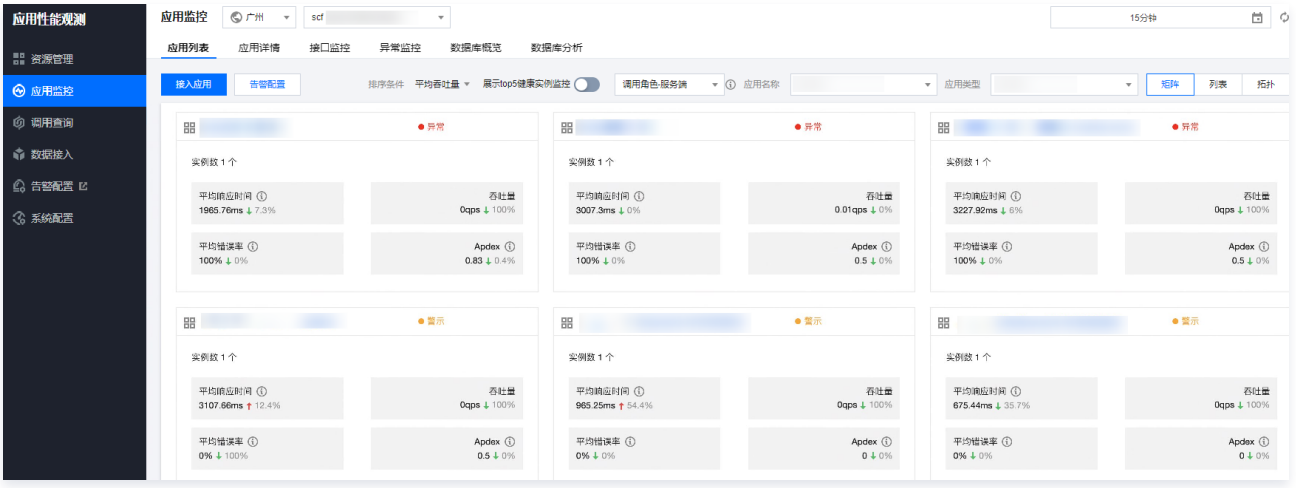


查看函数及所在业务系统健康情况

函数应用性能观测功能开启后，SCF 将上报函数执行基本信息到指定业务系统，一个函数将对应业务系统下的一个应用。应用命名方式为：命名空间名称/函数名称。

查看函数所在业务系统健康情况

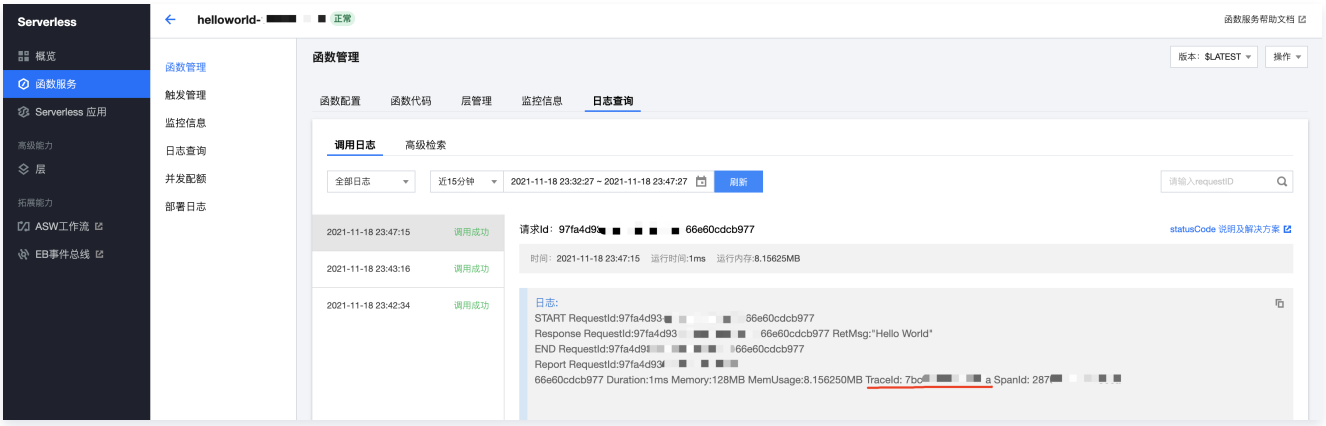
1. 在函数应用性能观测配置中单击业务系统，即可跳转至应用性能观测控制台。
2. 单击左侧导航栏中的应用监控-应用列表，可查看业务系统下各应用的健康情况。



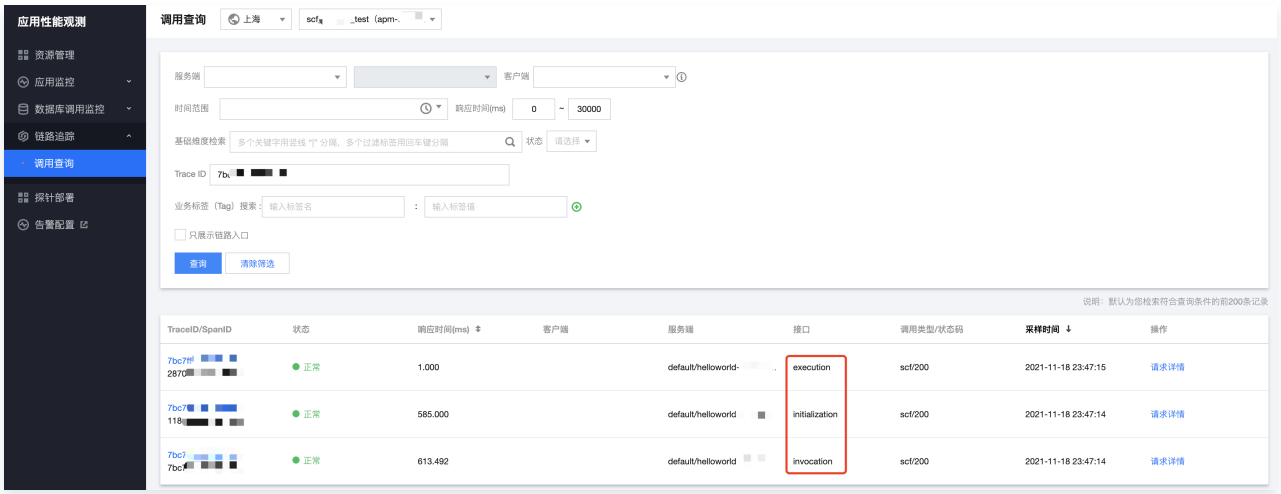
3. 单击系统看板右上角的**拓扑**，可查看该业务系统下应用之间的调用关系拓扑图。

函数调用链路分析

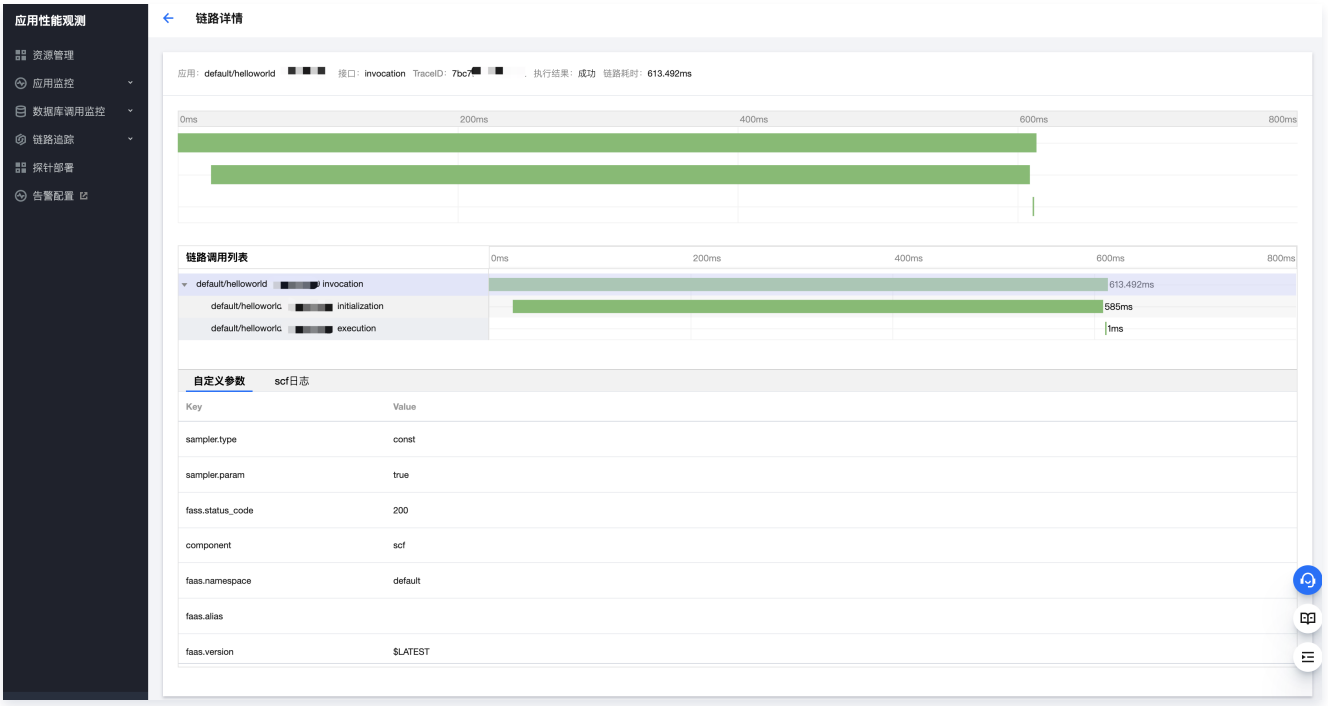
1. 开启了应用性能观测的函数，可在函数执行日志中获取到 APM TraceId。函数执行未重试场景下，TraceId 与函数执行 RequestId 一一对应；函数重试场景下，多次重试具有相同的 RequestId，TraceId 与函数执行 RequestId 为多对一的关系。



2. 登录 应用性能观测控制台，单击左侧导航栏**链路追踪 > 调用查询**。
3. 在页面上方选择地域和业务系统。
4. 输入需要查询的 TraceID，单击**查询**。以下图一个冷启动调用的函数请求为例，查询结果中的内容含义如下：
   - TraceID/SpanID：本次请求的 TraceID（地域唯一）/当前 Span 的 SpanID（Trace 内唯一）
   - 状态：状态码为 200 时为正常，其他为异常。
   - 响应时间：分段运行耗时。
   - 服务端：命名空间/函数名称。
   - 接口：`invocation` 表示函数运行总耗时，`initialization` 表示冷启动耗时，`execution` 表示函数执行耗时。
   - 调用类型/状态码：云函数平台上报分段调用类型记为 scf，状态码为函数请求状态码。



5. 单击**请求详情**，可查看链路瀑布图及函数执行日志。





SCF 自定义参数及含义

Key	含义	示例
fass.status_code	函数执行 状态码 。	200
component	固定字段，填 scf。	scf
faas.namespace	函数所在命名空间。	default
faas.alias	触发的函数别名，可能为空。	$DEFAULT
faas.version	触发的函数版本。	$LATEST
faas.runtime	函数运行环境。	Python3.6
faas.trigger	函数触发源。	YUNAPI
faas.request_id	函数请求 id。	97fa4d93-xxxx-xxxx-xxxx-66e60cdcb977
faas.async_execution	异步执行函数标识，true 表示是，false 表示否。	false
faas.websocket	websocket 函数标识，true 表示是，false 表示否。	false
faas.function_type	函数类型，事件函数或 Web 函数。	Event
faas.region	函数所在地域，参考函数 支持地域 。	ap-guangzhou

faas.retrynum	请求重试次数。	0
faas.coldstart	是否为冷启动调用，true 表示是，false 表示否。	true

自定义埋点上报

启用应用性能观测后，平台会默认上报函数运行总耗时、冷启动耗时（仅在冷启动调用下上报）、执行耗时三段基本时间到 APM。除此之外，您还可以通过在需要监控的代码段前后埋点进行自定义上报。下文将描述如何进行自定义上报：

> **注意**
> SCF 平台使用基于 OpenTracing 的 Jaeger 上报函数信息到 APM，在函数代码中进行自定义上报时，如需自定义上报 Span 与 SCF 平台上报 Span 使用同一个 TraceId 进行串联，自定义上报需要使用 Jaeger SDK 或与 Jaeger 兼容的 SDK。

获取 TraceId

启用应用性能观测后，平台默认上报 APM 数据的同时，会将 APM 的 Trace 信息注入到函数环境中，可通过如下方式进行获取：

- 非镜像部署的事件函数：
  可在函数入参 context 中获取 `function_trace` 中的 `Uber-Trace-Id`。
- 镜像部署函数：
  可在函数请求 header 中获取 `Uber-Trace-Id`。

自定义上报 Span

Node.js

1. 请参考 依赖安装 安装 `jaeger-client` SDK。
2. 创建 `Node.js` 函数，启用应用性能观测，并参考以下示例进行自定义埋点：

```
'use strict';

const initTracer = require('jaeger-client').initTracer;
const spanContext = require('jaeger-client').SpanContext;

function sleep (time) {
  return new Promise((resolve) => setTimeout(resolve, time));
}

const config = {
        serviceName: 'namespace/functionname', // 服务名称，根据业务需要自行修改，对应 APM 业务系统中的应用名称，SCF 默认上报应用名称命名规范为 命名空间/函数名称
        sampler: {
            type: 'const',
            param: 1
        },
        reporter: {
            logSpans: true,
            collectorEndpoint: 'http://ap-shanghai.apm.tencentcs.com:14268/api/traces' //应用性能观测配置中的接入点信息
        }
    };
    const options = {
        tags: {
            token: 'IJc**************aQ'//应用性能观测配置中的 Token 信息
        }
    };

// ******** 初始化 tracer 实例对象 ********
const tracer = initTracer(config, options);
```

```
exports.main_handler = async (event, context) => {
    // ******** 获取 context 中 trace 信息，未启用应用性能观测状态下，该信息不存在 ********
    const obj = JSON.parse(context['function_trace'])
    const TraceId = obj['Uber-Trace-Id'][0]

    // ******** 自定义创建一个子 span，命名为 custom-span ********
    const ScfSpanContext = spanContext.fromString(TraceId)
    const span = tracer.startSpan('custom-span', {
        childOf: ScfSpanContext
    })

    sleep(200).then(() => {
        console.log("custom span");
        // ******** 标记 Span 结束 ********
        span.finish();
        tracer.close();
    })

    // ******** 自定义设置 span 的标签（可选，支持多个）********
    //span.setTag('span.kind', 'server');
    //span.setTag('tagName', 'tagValue');
    //span.log({ event: 'timestamp', value: Date.now() });

    return ("hello from SCF");
}
```

3. 接入点信息与 Token 可参考上文 查看函数应用性能观测配置 获取。
4. 函数调用后，各 Span 信息会上报至指定的 APM 业务系统。

Python

> **注意**
> 1. Jaeger 最新版 Python SDK 已经不支持 HTTP 协议上报，因此在 Python 运行环境下，推荐使用 Open Telemetry SDK。
> 2. Open Telemetry SDK 仅支持 Python 3.7 及以上版本，本文提供基于镜像部署的自定义上报示例。

1. 请参考 使用镜像部署函数 完成前提条件准备。
2. 参考以下示例代码准备镜像函数源码。
   - index.py

```
from flask import Flask, request
import time
from opentelemetry import trace
from opentelemetry.exporter.otlp.proto.grpc.trace_exporter import OTLPSpanExporter
from opentelemetry.sdk.resources import Resource
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import BatchSpanProcessor
from opentelemetry.trace import NonRecordingSpan, SpanContext, TraceFlags
from opentelemetry.trace import set_span_in_context

class OpenTelemetrySLSProvider(object):

    def __init__(self, service="", token="", endpoint=""):
        '''
        :param service: Your Application Service Name
        :param service: server name
```

```
        :param token: scf project
        '''

        self.sls_otel_endpoint = endpoint
        self.resource = Resource(attributes={
            "service.name": service,
            "token": token})

    def initTracer(self):
        trace.set_tracer_provider(TracerProvider(resource=self.resource))
        otlp_exporter = OTLPSpanExporter(endpoint=self.sls_otel_endpoint)
        trace.get_tracer_provider().add_span_processor(BatchSpanProcessor(otlp_exporter))


server = "namespace/functionname" # 服务名称，根据业务需要自行修改，对应 APM 业务系统中的应用名称，SCF 默认上报应用名称命名规范为 命名空间/函数名称
endpoint = "ap-shanghai.apm.tencentcs.com:4317" # 应用性能观测配置中的接入点信息
instance = "apm-Q*******Q" # 应用性能观测配置中的业务系统 ID 信息
token = "Tr***********a" # 应用性能观测配置中的 Token 信息

# ******** 初始化 tracer 实例对象 ********
sls_ot_provider = OpenTelemetrySLSProvider(service=server, token=token, endpoint=endpoint)
sls_ot_provider.initTracer()
tracer = trace.get_tracer("__name__")

app = Flask(__name__)

# ******** 事件函数 ********
@app.route('/event-invoke', methods=['POST'])
def invoke():
    # ******** 获取请求 header 中 trace 信息，未启用应用性能观测状态下，该信息不存在 ********
    TraceId = request.headers.get("Uber-Trace-Id", "")

    SCF_span = TraceId.split(":")
    tid = int(SCF_span[0], 16)
    sid = int(SCF_span[1], 16)
    fid = int(SCF_span[3], 16)
    span_context = trace.SpanContext(
        trace_id=tid,
        span_id=sid,
        is_remote=True,
        trace_flags=trace.TraceFlags(fid),
    )

    ctx = trace.set_span_in_context(NonRecordingSpan(span_context))

    # ******** 自定义创建一个子 span，命名为 custom-span ********
    child = tracer.start_span("custom-span", context=ctx)

    time.sleep(0.2)
    print("child")
    # ******** 标记 Span 结束 ********
    child.end()

    return "hello from SCF"

# ******** Web 函数 ********
@app.route('/web-invoke/python-flask-http', methods=['POST', 'GET'])
def web_invoke():
    # ******** 获取请求 header 中 trace 信息，未启用应用性能观测状态下，该信息不存在 ********
    TraceId = request.headers.get("Uber-Trace-Id", "")
```

```
    SCF_span = TraceId.split(":")
    tid = int(SCF_span[0], 16)
    sid = int(SCF_span[1], 16)
    fid = int(SCF_span[3], 16)
    span_context = trace.SpanContext(
        trace_id=tid,
        span_id=sid,
        is_remote=True,
        trace_flags=trace.TraceFlags(fid),
    )

    ctx = trace.set_span_in_context(NonRecordingSpan(span_context))


    # ******** 自定义创建一个子 span，命名为 custom-span ********
    child = tracer.start_span("custom-span", context=ctx)

    time.sleep(0.2)
    print("child")

    # ******** 标记 Span 结束 ********
    child.end()

    return "hello from SCF"


if __name__ == '__main__':
    app.run(host='0.0.0.0', port=9000)
```

- requirements.txt

```
Flask==1.1.2
opentelemetry-api==1.7.1
opentelemetry-exporter-jaeger==1.7.1
opentelemetry-exporter-jaeger-proto-grpc==1.7.1
opentelemetry-exporter-jaeger-thrift==1.7.1
opentelemetry-exporter-otlp==1.7.1
opentelemetry-instrumentation-flask==0.26b1
opentelemetry-instrumentation-requests==0.26b1
opentelemetry-sdk==1.7.1
opentelemetry-semantic-conventions==0.26b1
```

- dockerfile

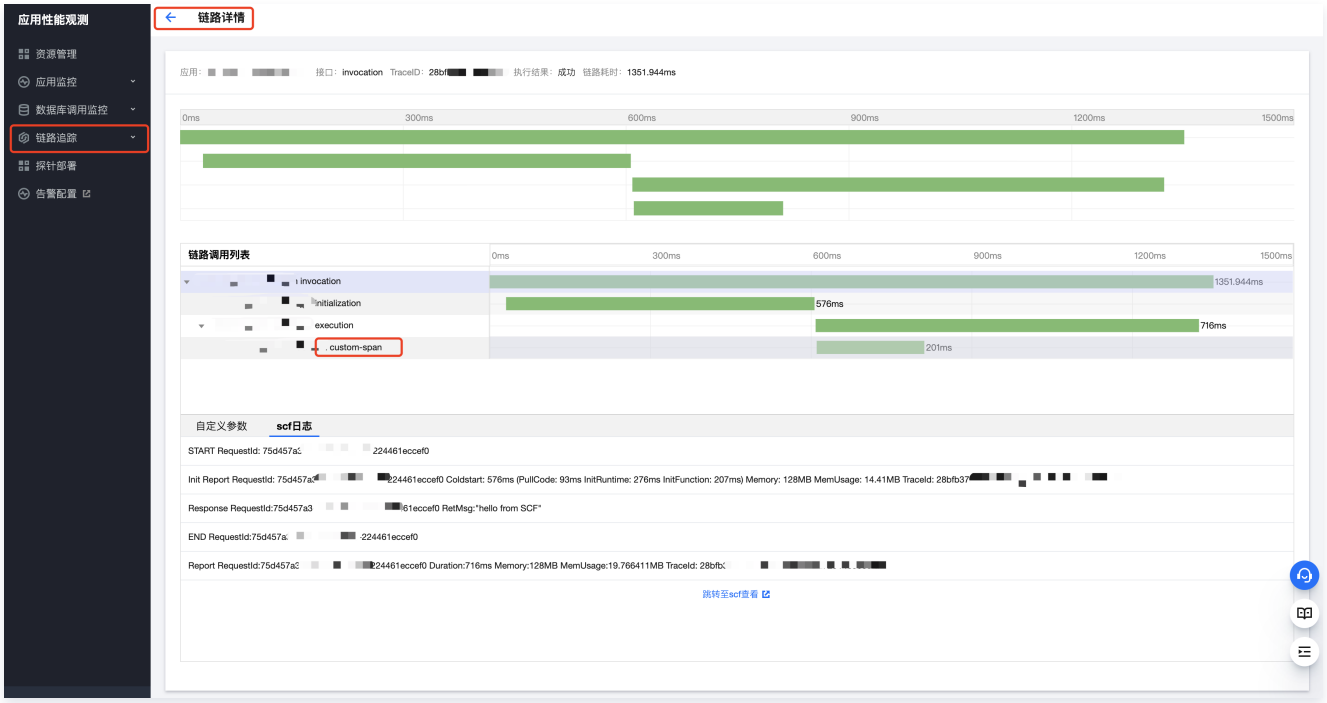
```
FROM python:3.7.12-slim

WORKDIR /usr/src/app
COPY ./requirements.txt ./
RUN python -m pip install --upgrade pip  && pip install --no-cache-dir -r ./requirements.txt
COPY . .

CMD [ "python", "-u", "./index.py" ]
```

3. 将步骤2中的 `index.py` 、 `requirements.txt` 、 `dockerfile` 文件放在同一目录下，参考 使用控制台创建函数 完成镜像构建与函数创建。上述示例代码执行后上报的链路详情如下：

TraceID/SpanID	状态	响应时间(ms)	客户端	服务端	接口	调用类型/状态码	采样时间	操作
28b[redacted] 32a3[redacted]	正常	201.000		[redacted]	custom-span	成功	2021-11-28 17:25:49	请求详情
28b[redacted] 0d7[redacted]	正常	716.000			execution	scf/200	2021-11-28 17:25:49	请求详情
28b[redacted] 506[redacted]	正常	576.000		[redacted]	initialization	scf/200	2021-11-28 17:25:49	请求详情
28b[redacted] 28bf[redacted]	正常	1351.944		[redacted]	invocation	scf/200	2021-11-28 17:25:49	请求详情



使用博睿数据 APM

最近更新时间：2025-05-07 15:14:32

本文将为您介绍云函数如何接入和使用博睿数据 APM。

前提条件

- 已注册 博睿 Server 账号。
- 已 创建云函数 并开启公网访问。

> **说明：**
> 博睿探针目前支持 Python 和 Node.js 的多数主流框架，且仅在使用支持的框架时，博睿 smartAgent 才可自动捕获。详情请参见 博睿探针支持列表 。

操作步骤

使用云函数控制台接入

您可以使用云函数控制台接入博睿，详细步骤如下：

绑定探针

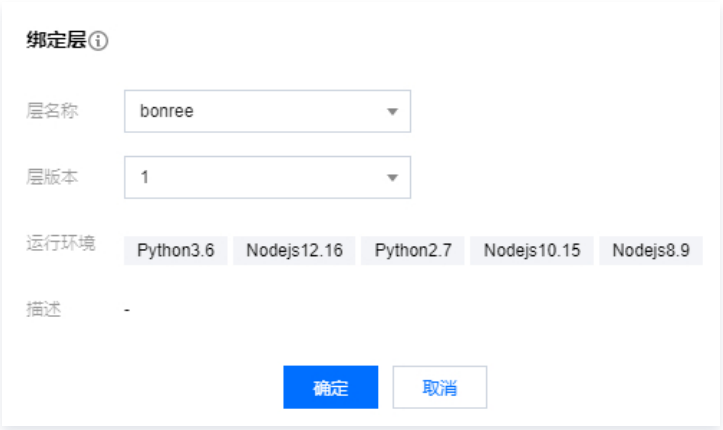
您需要下载博睿探针，将该探针上传到层并绑定在函数上。

1. 下载 博睿 Serverless 版探针 。
2. 登录云函数控制台，选择左侧菜单栏中的 层 。
3. 在**层管理**页面，单击**新建**。
4. 在**新建层**页面，根据提示信息进行配置。如下图所示：



   - **层名称**：输入层名称。只能包含字母、数字、下划线、连字符，以字母开头，以数字或字母结尾，2~60个字符。
   - **提交方法**：选择**本地上传 zip 包**。
   - **层代码**：选择步骤1下载的探针文件。
   - **运行环境**：根据实际运行环境进行选择。目前支持 Python 和 Node.js。
5. 单击**确定**即可创建层。
6. 选择左侧菜单栏中的**函数服务**，进入函数服务页面。
7. 单击需要绑定层的函数名称，进入函数管理页面。

8. 选择**层管理** > **绑定**，在绑定层窗口按照提示绑定上述步骤创建的层。如下图所示：



9. 单击**确定**即可完成探针的绑定。

引入探针

绑定层后，探针并不会自动启动，需要在代码入口引入探针。探针运行会占用少量内存，但不会影响您的业务运行。如业务代码本身占用了大量内存，探针将触发熔断机制以保障业务运行。目前提供 Nods.js 和 Python 引入：

Node.js 的引入

在云函数的入口函数所在的文件 `require` 博睿探针。例如您可以在 `sl_handler.js` 文件中加入如下引入的代码：

```
require("/opt/bonree/apm/agent/nodejs/serverless/Bonree/index.js");
```

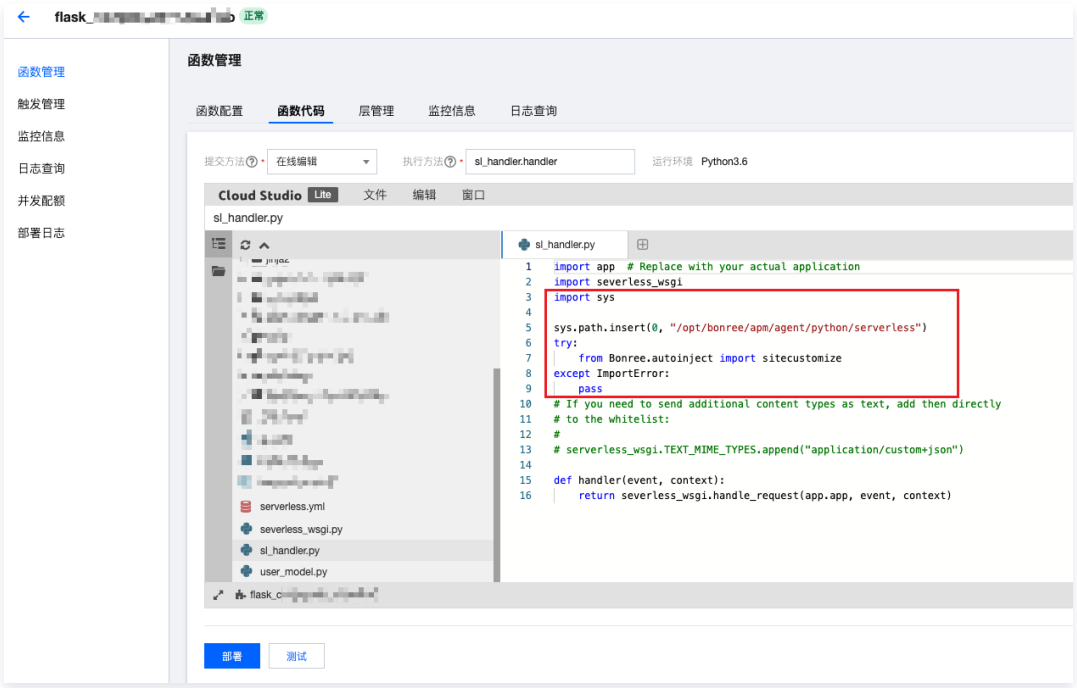
Python 的引入

在云函数的入口函数所在的文件 `import` 博睿探针。以 Flask 框架为例，您可以在 `sl_handler.py` 文件中添加如下代码：

```
import sys

sys.path.insert(0, "/opt/bonree/apm/agent/python/serverless")
try:
   from Bonree.autoinject import sitecustomize
except ImportError:
   pass
```

如下图所示：



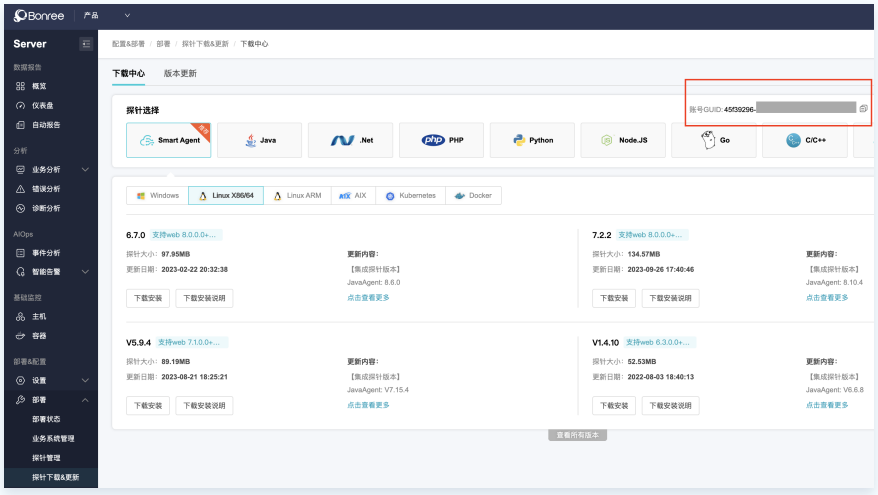
增加云函数环境变量

您需要在每个绑定博睿探针的云函数中增加环境变量，探针将根据环境变量中的账户信息上报。需新增以下变量：

环境变量 Key	Value
BONREE_SMARTAGENT_SDK_PATH	/opt/bonree/apm/agent/c/serverless/lib/libagentsdk-x64-linux.so
BONREE_APM_ACCOUNT_GUID	博睿账户 GUID

说明：

您可以在 博睿 Server – 探针下载&更新界面的右上角找到 GUID 信息。如下图所示：



使用 Serverless Cloud Framework 接入

您还可以使用 Serverless Cloud Framework 的 bonree component 上传博睿探针。本文以 Flask 框架为例，介绍如何使用 bonree component 来绑定和使用博睿探针。您也可以单独使用 bonree component 上传层，再进行 层绑定 。

创建层并绑定至函数

1. 在 apm 目录下新建 `serverless.yml` 文件，`serverless.yml` 文件内容如下：

```
component: bonree
name: bonree_agent
org: tencent
app: tencent
stage: dev
inputs:
  name: bonree_agent
  region: ap-beijing
  runtimes:
    - Nodejs10.15
    - Python3.6
```

2. 同样在 src 目录下创建 Flask 的 `serverless.yml` 文件，在 `layers` 参数填写 bonree component 的信息，在环境变量中填写博睿账号的GUID和SDK路径参数。`serverless.yml` 文件内容如下：

```
component: flask
name: bonree_flask
org: tencent
app: tencent
stage: dev
inputs:
  region: ap-beijing
  runtime: Python3.6
  layers:
    - name: ${output:${stage}:${app}:bonree_agent.name}
      version: ${output:${stage}:${app}:bonree_agent.version}
  functionConf:
    memorySize: 128
    environment:
      variables:
        BONREE_APM_ACCOUNT_GUID: your_bonree_GUID
        BONREE_SMARTAGENT_SDK_PATH: /opt/bonree/apm/agent/c/serverless/lib/libagentsdk-x64-linux.so
```

3. 查看目录结构，具体目录结构如下所示：

```
.
├── apm
|   └── serverless.yml
└── src
    └── serverless.yml
```

4. 只使用 bonree component 便可以完成层的创建。在云函数中配置 layers 参数可以完成绑定操作，您也可以选择在云函数控制台手动绑定层。
5. 在根目录下执行以下命令，进行应用部署。

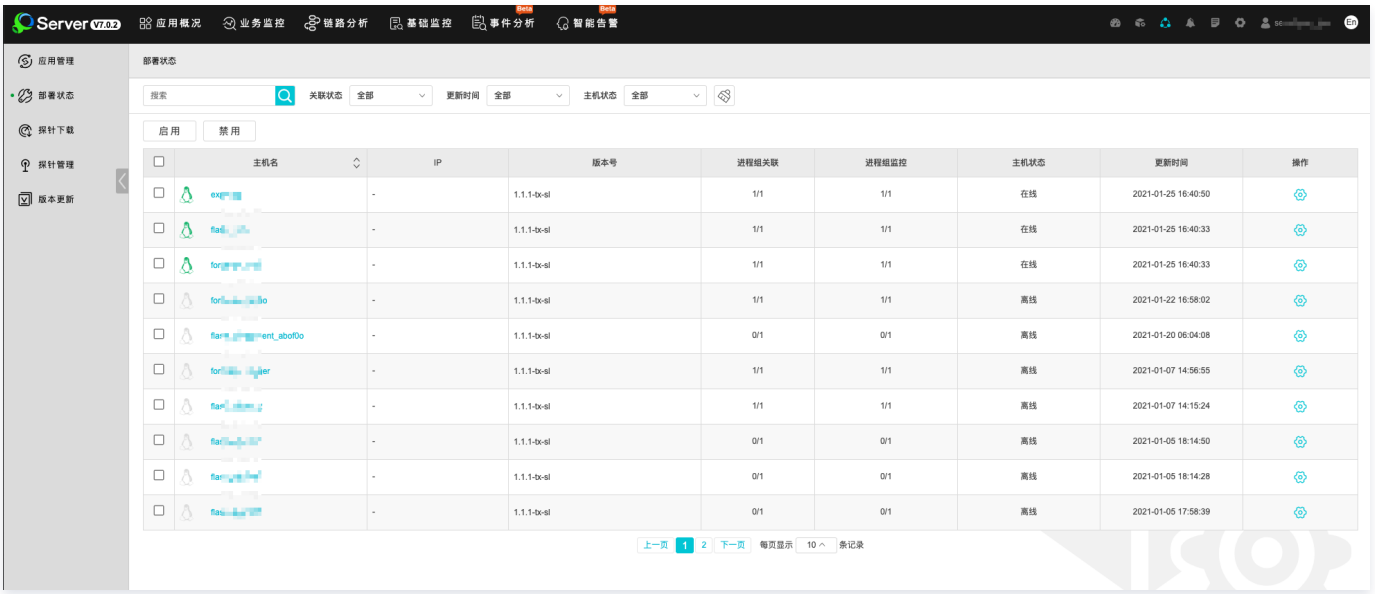
```
scf deploy
```

引入探针

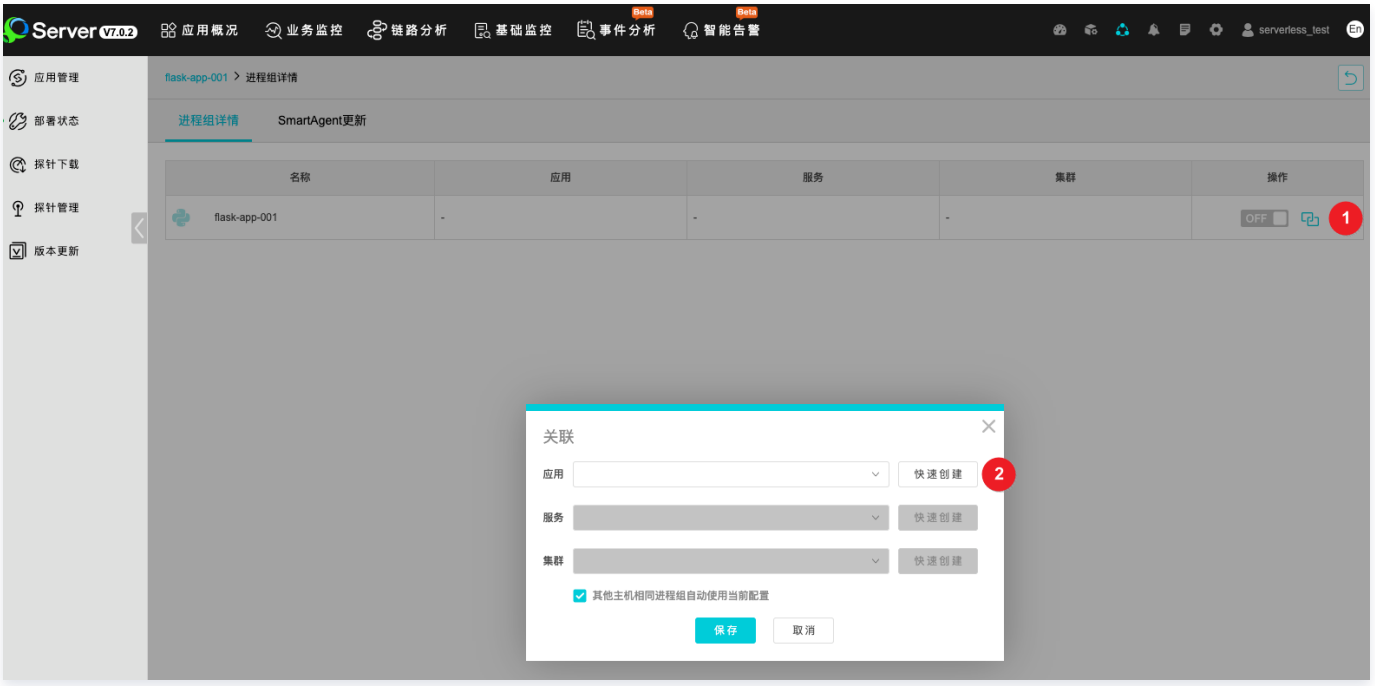
1. 登录 Serverless 控制台，单击左侧导航栏的函数服务。
2. 在函数服务页面，单击对应的函数名称，进入函数管理页面。
3. 单击**函数代码**页签，在 `sl_handler.py` 文件中加入引入探针的代码，详细内容可参考上文 引入探针 。

使用博睿 Server

1. 登录 博睿 Server 控制台，待数据上报至博睿。
2. 在博睿 Server 控制台右上角中选择 > **部署状态**，进入部署状态页面查看已进行数据上报的函数。如下图所示：



3. 将该函数关联一个应用，即可查看应用运行的情况。如下图所示：



4. 您可以将多个函数上报后关联至同一个应用，便可查看调用链路情况。如下图所示：



更多操作指导可以查看 博睿 Server 产品文档 。

博睿探针支持列表

博睿探针目前支持以下主流框架和库：

Python

Python 支持的框架及库如下：

框架	版本
Flask	0.10及以上/1.1.2
Django	1.5及以上/3.1
Tornado	3.0及以上/6.1
Web.py	0.33及以上/0.6.2
Pyramid	1.3及以上/1.10.5
Bottle	0.12及以上/0.12.19
Cherrypy	10.0及以上/18.6.0
Sanic	0.5.0及以上/20.9.1
Odoo	8.0及其以上/14.0
fastapi	0.23.0及以0.63.0
quart	0.11.0及以0.14.1

starlette	0.12.0及以上0.14.1
Pymysql	0.7.1及以上/0.10.1
mysqlclient	1.3.0及以上/2.0.1
mysql-connector-python	8.0.5及以上/8.0.22
psycopg2	2.6.2及以上/2.8.6
Cx-Oracle	6.0及以上/8.0.1
pyhive	0.1.6及以上/0.6.3
Pymongo	3.3.0及以上/3.11.1
python-memcached	1.57及以上/1.5.9
pyssdb	0.4.0及以上/0.4.2
redis	2.10.0及以上/3.5.3
redis-py-cluster	0.1.0及以上/2.1.0
urllib3	1.18及以上/1.26.2
requests	2.12.0及以上/2.25.0
httplib/http	标准库
tornado_httpclient	3.1及以上6.1
elasticsearch	5.2.0及以上/7.10.0
grpcio	1.0.0及以上/1.33.2
xmlrpclib	python 2.6/2.7
xmlrpc	python3.4+
thrift	0.10.0及以上/0.13.0
aiohttp	3.0.0及以上/3.7.3
kafka-python	1.3.0及以上/2.0.2
stomp.py	4.1.20及以上/6.1.0
kombu	3.0.30及以上/5.0.2
librabbitmq	1.6.0及以上2.0.0
Logging	标准库
logbook	1.3.0及以上/1.5.3
Eliot	0.8.0及以上/1.12.0
celery	3.1.0及以上/5.0.2

Node.js

Node.js 支持的框架及库如下：

框架	版本

Express	3.4.8及以上
Koa	2.2.0及以上
Hapi	17.0.0及以上
Promise	8.0.1及以上
Bluebird	3.5.1及以上
when	3.7.8及以上
Async	2.6.0及以上
q	1.5.1及以上
request	2.18.0
superagent	3.6.0及以上
mysql	2.13.0及以上
pg	6.2.4及以上
ioredis	2.5.0及以上
redis	2.8.0及以上
hiredis	0.5.0及以上
mongodb	2.2.31及以上
mongoose	5.0.10及以上
rabbit.js	0.4.4及以上
amqplib	0.5.2及以上

版权所有：腾讯云计算（北京）有限责任公司

网络配置
网络配置管理

最近更新时间：2024-07-22 17:41:31

操作场景

云函数创建成功后，默认只有公网访问权限。即云函数在运行时，只能访问暴露在公共网络环境中的资源，例如腾讯云官网等。
云函数的网络配置项支持用户进行以下设置：

网络配置项	说明
仅公网访问	-
公网访问和固定公网出口 IP	进行此设置后，该云函数会使用一个固定的 IP 地址访问公共网络资源。
仅内网访问	进行此设置后，云函数可以访问已配置该私有网络的云服务器 CVM、轻量应用服务器 LH、数据库、Redis 等资源。
仅内网访问和固定内网出口 IP	进行此设置后，云函数会使用一个固定的内网 IP 地址访问已配置该私有网络的云服务器 CVM、轻量应用服务器 LH、数据库、Redis 等资源。
同时开启内网访问和公网访问	进行此设置后，云函数可以访问公共网络资源和已配置该私有网络的云服务器 CVM、轻量应用服务器 LH、数据库、Redis 等资源。
同时开启内网访问、固定内网出口 IP、公网访问和固定公网出口 IP	进行此设置后，云函数会使用一个固定的公网 IP 地址访问公共网络资源。同时可以使用固定的内网 IP 地址访问已配置该私有网络的数据库、Redis 等资源。

> **说明：**
> 云函数获取固定公网出口 IP 配置，详情请参见 固定公网出口 IP 。

前提条件

- 已注册腾讯云账户。若未注册，请前往 注册页面 。
- 已 创建云函数 。

操作步骤

设置网络配置

1. 登录 Serverless 控制台，单击左侧导航栏的**函数服务**。
2. 在页面上方选择地域，单击需要进行网络配置的函数名。
3. 在**函数配置**页面，选择右上角的**编辑**。
4. 您可根据实际需求，参考 固定公网出口 IP 和 私有网络通信 进行配置。

查看网络配置

1. 登录 Serverless 控制台，单击左侧导航栏的**函数服务**。
2. 在页面上方选择地域，并单击函数名，即可在**函数配置**中查看当前网络配置。

网络限制

并发连接数限制

当前针对访问目标的同一 ip：port ，并发连接数限制为六万个。由于短连接涉及到中间设备的释放时间，因此在短连接情况下，连接数会进一步下降。
如果您有多个函数或同一函数的多个并发访问同一 ip：port ，请注意此处的限制，并可通过以下方案来避免快速耗尽连接数导致代码错误。

- 尽量使用长连接。在函数初始化阶段完成连接并持续复用连接，来避免实际调用过程中使用短连接带来的频繁连接、释放。该措施可以充分使用连接数，但仍存在连接数上限的限制。
- 访问目标提供多个 ip：port 对。通过访问目标的多个 ip：port 对，将连接尽量分散到多个连接目标上，可以避免触碰到连接数上限。

版权所有：腾讯云计算（北京）有限责任公司

带宽限制

外网

配置公网访问且需要固定公网出口 IP 的情况时，公网带宽（出）限制为 0.1Gbps。

内网

在配置 VPC 连接内网的情况下，当前针对某一特定 VPC 内网带宽（出+入）限制为 1.5Gbps。带宽由配置了相同 VPC 的函数和函数的多个并发实例共享。如需提升内网带宽，请 提交工单 申请。

固定公网出口 IP

最近更新时间：2023-09-11 21:26:31

操作场景

当用户在云函数中访问数据库、微信公众号的 API 接口或其他第三方的服务时，可以使用云函数的固定公网出口 IP 功能，实现云函数网络配置的控制与管理。
云函数的固定公网出口 IP 功能具有以下特点：

- 当云函数启用固定公网出口 IP 功能后，该云函数将会获得一个随机分配的弹性公网 IP。该云函数访问公网的流量，将会基于该弹性公网 IP 统一进行转发。
- 当在云函数同时开启公网访问、内网访问并启用固定公网出口 IP 功能时，访问公网的流量会基于弹性公网 IP 进行转发，访问内网的流量会基于私有网络进行转发。

使用限制

- 弹性公网 IP 在同一账号的同一地域下共享。
- 同一账号的同一地域下，已开启固定公网出口 IP 功能的云函数将共享弹性公网 IP。
- 同一账号的同一地域下的云函数需更换固定出口 IP 时，所有云函数需关闭固定公网出口 IP 功能。再次开启此功能时，会随机产生一个新的弹性公网 IP。
- 弹性公网 IP 基于私有网络的子网共享。
  某个云函数配置了私有网络，且同时开启了固定公网出口 IP 功能，则该云函数会获得一个随机分配的弹性公网 IP。同一私有网络子网下的云函数在开启固定出口 IP 功能时，会共享此固定出口 IP。

示例

为了便于您理解固定公网出口 IP 的使用限制，以下为您进行一个简单的示例说明。
假设您的账号在某地域有如下场景：

- 命名空间 A 下已创建了云函数 a 和云函数 b。
- 命名空间 B 下已创建了云函数 c 和云函数 d。
- 弹性公网 IP-x、弹性公网 IP-y 分别表示两个不同的弹性公网 IP。

它们的弹性公网 IP 和云函数的绑定关系如下表所示：

网络配置	命名空间 A		命名空间 B	
	函数 a	函数 b	函数 c	函数 d
仅公网访问	无弹性公网 IP	无弹性公网 IP	无弹性公网 IP	无弹性公网 IP
仅内网访问	无弹性公网 IP	无弹性公网 IP	无弹性公网 IP	无弹性公网 IP
公网访问且固定公网出口 IP	弹性公网 IP-x	弹性公网 IP-x	弹性公网 IP-x	弹性公网 IP-x
同一私有网络访问且固定公网出口 IP	弹性公网 IP-y	弹性公网 IP-y	弹性公网 IP-y	弹性公网 IP-y

操作步骤

> **注意**
> 每个用户在每个地域固定 IP 限额为5个。

1. 登录 Serverless 控制台，单击左侧导航栏的函数服务。
2. 在页面上方选择云函数所在地域，单击函数名。
3. 进入**函数配置**页签，单击右上角的**编辑**。
4. 根据您的实际需求，进行该云函数的网络配置。如下图所示：

> **注意**
> - 云函数开启公网访问后，才可选择开启固定公网出口 IP。
> - 您无法手动选择或编辑随机生成的弹性公网 IP。



配置完成后，单击**保存**即可。

私有网络通信

最近更新时间：2024-11-29 17:29:02

操作场景

腾讯云云函数默认部署在公共网络中，本文介绍了通过私有网络配置实现云函数访问内网中的资源，例如 TencentDB、CVM、Redis、Kafka 等，确保了数据安全及连接安全。

注意事项

在进行私有网络配置时，需注意以下几点：

- 部署在 VPC 中的云函数默认隔离外网。若想使云函数同时具备内网访问和外网访问能力，可通过以下两种方式实现：
  - 通过配置云函数公网访问能力，且公网访问可控制出口地址唯一，请参见 固定公网出口 IP 。
  - 通过 VPC 添加 NAT 网关，请参见 私有网络中配置 NAT 。
- 云函数目前不支持对接到基础网络里的资源。

前提条件

已 创建云函数 。

操作步骤

修改网络配置

1. 登录 Serverless 控制台，单击左侧导航栏的函数服务。
2. 在页面上方选择地域，单击需要配置的函数名。
3. 在**函数配置**页面，单击右上角的**编辑**。
4. 开启**私有网络**功能，选择需要接入的 VPC 网络和所需要的使用的子网。

使用 VPC 网络

在云函数完成内网访问配置，并开始使用 VPC 网络时，云函数将从当前独立的网络环境切换至已配置的 VPC 中。云函数启动时，将占用用户 VPC 子网中的 IP 地址作为云函数运行环境的 IP 地址。
云函数启动后，可通过代码及内网 IP 地址访问 VPC 中的资源，例如 云数据库 TencentDB for Redis 、云关系型数据库 、用户配置在 VPC 中的 CVM 等各种访问入口位于 VPC 中的资源。
以下为访问 云数据库 TencentDB for Redis 的示例代码，其中 Redis 实例在 VPC 内的 IP 地址为 `10.0.0.86` 。

```
# -*- coding: utf8 -*-
import redis
def main_handler(event,context):
    r = redis.StrictRedis(host='10.0.0.86', port=6379, db=0,password="crs-i4kg86dg:abcd1234")
    print(r.set('foo', 'bar'))
    print(r.get('foo'))
    return r.get('foo')
```

VPC 网络中访问自定义域名

使用私有域解析访问 VPC 网络中的自定义域名（推荐）

在使用 VPC 网络的过程中，若需要通过使用域名来访问内网自建服务，可以通过使用腾讯云提供的 私有域解析 Private DNS 来实现内网自定义域名配置及访问解析。

设置云函数环境中的 Name Server

如果需要对接自定义域名解析服务器，需要在云函数环境内自定义 `name server` 配置，当前可通过配置 `OS_NAMESERVER` 环境变量来实现。实际配置如下表：

环境变量名	值设置规则	作用
OS_NAMESERVER	可以为一个或多个 IP 地址、或域名，多个地址时使用;分号分隔。 最多可以支持配置5个自定义 name server。	配置自定义 name server。

使用如下 Python 语言实现的示例代码，可通过打印输出 `/etc/resolv.conf` 文件检查配置生效情况。

```
with open("/etc/resolv.conf") as f:
    print(f.readlines())
```

相关操作

查看网络配置

1. 登录 Serverless 控制台，单击左侧导航栏的**函数服务**。
2. 在页面上方选择地域，并单击已配置内网访问的函数名，即可通过**所属网络**和**所属子网**了解到具体配置。

层管理
层管理概述

最近更新时间：2022-12-16 11:34:45

概述

如果您的云函数（SCF）拥有较多的依赖库或公共代码文件，您可以使用 SCF 中的层进行管理。使用层管理，您可以将依赖放在层中而不是部署包中，可确保部署包保持较小的体积。对于 Node.js、Python 和 PHP 函数，只要将部署程序包保持在10MB以下，就可以在 SCF 控制台中在线编辑函数代码。

工作方式

创建与绑定

创建层的压缩文件将按照层的版本进行存储。层在与函数进行绑定时，将按照具体的层版本与函数版本进行绑定。一个函数目前最多支持绑定5个层的具体版本，并在绑定时有一定顺序。

运行时加载与访问

已绑定层的函数被触发运行，启动并发实例时，将会解压加载函数的运行代码至 **/var/user/** 目录下，同时会将层内容解压加载至 **/opt** 目录下。
若需使用或访问的文件 **file**，放置在创建层时压缩文件的根目录下。则在解压加载后，可直接通过目录 **/opt/file** 访问到该文件。若在创建层时，通过文件夹进行压缩 **dir/file**，则在函数运行时需通过 **/opt/dir/file** 访问具体文件。
在函数绑定了多个层的情况下，层中文件的解压加载将按照绑定时的顺序进行。将按序号从小到大的顺序进行排序，排序越靠后侧层加载时间也相应靠后，但均会在函数的并发实例启动前完成加载。在函数代码初始化时，就已经可使用层中的文件了。

推荐使用方式

层中通常用来存储不经常变更的静态文件或代码依赖库。在存储代码依赖库时，可以直接将可用的依赖库打包并上传至层中。例如，在 Python 环境中，可以将依赖库的代码包文件夹直接打包并创建为层，则在函数代码中可直接通过 import 引用。在 Nodejs 环境中，可以将项目的 node_modules 依赖库文件夹打包并创建为层，则在函数代码中可直接通过 require 引用。
通过使用层，可以将函数代码和依赖库或依赖的静态文件分离，保持函数代码较小体积。在使用命令行工具、IDE 插件或控制台编辑函数时，均可以快速上传更新。

说明事项

- 层中的文件将会添加到 `/opt` 目录中，此目录在函数执行期间可访问。
- 如果您的函数已绑定了多个层，这些层将按顺序合并到 `/opt` 目录中。如果同一个文件出现在多个层中，SCF 平台将会保留最大序号层里的文件。

相关操作

您可以通过 Serverless 控制台 创建层 、绑定层 并 使用层 。

创建层

最近更新时间：2024-11-29 17:29:02

本文介绍如何通过 Serverless 控制台创建层。新建层后，将会自动帮您生成一个版本。

操作步骤

1. 登录 Serverless 控制台，选择左侧导航栏中的**高级能力 > 层**。
2. 在**层管理**页面，选择需使用层的地域，并单击**新建**。
3. 在**新建层**页面，根据实际需求设置层信息。如下图所示：



   - **层名称**：输入自定义层名称。
   - **描述**：层的描述信息，根据实际情况填写。
   - **提交方法**：支持**本地上传zip包**、**本地上传文件夹**及**通过 cos 上传 zip 包**，结合实际情况选择层文件提交方式。
     - **本地上传 zip 包**：单击**上传**，请上传 zip 格式的代码包，最大支持50MB。
     - **本地上传文件夹**：单击**上传**，请选择文件夹，最大支持250MB。该方式上传后无法保留文件可执行权限，如包含可执行文件，请先在本地设置可执行权限并通过 zip 包方式上传。
     - **通过 cos 上传 zip 包**：选择用作事件源的 COS 存储桶，该存储桶必须位于函数所在地域。填写 COS 对象文件路径，从 Bucket 根目录("/")开始的 zip 代码文件完整路径。
   - **添加运行环境**：该层的兼容运行环境，最多可设置5个。
   - **标签**：该层资源所属的业务标签，只有标签匹配的用户才可见该资源。同时在函数服务添加层资源时，也只能添加与用户标签匹配的层资源。
4. 单击**确定**。创建完成后您可以在层列表中查看层。

云函数绑定层

最近更新时间：2023-09-27 10:37:01

本文介绍如何通过 Serverless 控制台为云函数绑定层。

操作步骤

1. 登录 Serverless 控制台，选择左侧导航栏中的**函数服务**。
2. 在**函数服务**页面，选择需进行层管理的函数 ID，进入函数详情页面。
3. 选择**层管理**页签，并单击**绑定**。如下图所示：



4. 在弹出的**绑定层**窗口中，选择对应**层名称**及**层版本**。如下图所示：



5. 单击**确定**即可完成绑定。

使用层

最近更新时间：2025-06-16 18:01:42

本文介绍如何通过 Serverless 控制台使用层。

使用说明

层中的文件均在 `/opt/` 目录下，可以在函数代码中通过绝对路径进行访问。除此之外，各运行时内置的环境变量中也包含了层路径，可以按照环境变量中层文件的路径上传文件，即可在代码中通过相对路径进行引用。
Python、Java、Node.js 环境变量见下表：

相关环境变量	路径
PYTHONPATH	/var/user:/opt
CLASSPATH	/var/runtime/java8:/var/runtime/java8/lib/*:/opt
NODE_PATH	/var/user:/var/user/node_modules:/var/lang/node6/lib/node_modules:/opt:/opt/node_modules

操作步骤

Node.js

以 Node.js 运行环境，在代码中引用层中的 `node_modules` 中的 `cos-nodejs-sdk-v5` 依赖为例：
1. 参考 创建层 步骤将 `node_modules` 上传生成层。本地函数目录结构如下图所示：

```
bash-3.2$ ls
index.js        node_modules    package.json
```

2. 参考 部署函数 将本地函数代码打包上传，打包时执行以下命令排除 `node_modules` 文件夹。

```
zip -r 包名.zip . -x "node_modules/*"
```

如下图所示：

```
bash-3.2$ zip -r demo.zip . -x "node_modules/*"
  adding: index.js (stored 0%)
  adding: package.json (deflated 31%)
```

3. 参考 绑定云函数 步骤，将已创建的层绑定至部署好的函数。
4. 完成上述步骤后，即可开始在函数中引用层中的文件。

```
'use strict'
var COS = require('cos-nodejs-sdk-v5')
```

> **注意：**
> - 由于 NODE_PATH 环境变量包含 `/opt/node_modules` 路径，所以无需指定依赖的绝对路径，SCF 运行时会按照环境变量中指定的路径加载文件。
> - 如层中文件路径和环境变量包含路径不一致，请在文件引用时使用绝对路径。

Python

以 Python 运行环境，在代码中引用层中的 `cos-python-sdk-v5` 依赖为例：
1. 参考 创建层 步骤将 `cos-python-sdk-v5` 上传生成层。
2. 参考 部署函数 将本地函数代码打包上传，已经上传到层中的文件无需跟随函数代码再次进行上传。
3. 参考 绑定云函数 步骤，将已创建的层绑定至部署好的函数。
4. 完成上述步骤后，即可开始在函数中引用层中的文件。

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

```
import cos-python-sdk-v5
```

> **注意：**
> - 由于 PYTHONPATH 环境变量包含 `/opt` 路径，所以无需指定依赖的绝对路径，SCF 运行时会按照环境变量中指定的路径加载文件。
> - 如层中文件路径和环境变量包含路径不一致，请在文件引用时使用绝对路径。

使用示例

本示例介绍如何使用层和测试函数。

1. 前往 scf_layer_demo，选择 **Clone or download** > **Download ZIP** 下载示例到本地并解压。
2. 登录 Serverless 控制台，创建层。操作步骤详情请参见 创建层。参数设置如下图所示：



   - **层名称**：自定义，本文以 `demo` 为例。
   - **提交方法**：选择**本地上传文件夹**，并选择上传 步骤1 中已获取文件夹中的 `layer` 文件夹。
   - **运行环境**：选择 **Nodejs 12.16**。
3. 登录 Serverless 控制台，新建函数。操作详情见 创建函数。基础配置如下所示：
   - **创建方式**：选择**从头开始**。
   - **函数类型**：选择**事件函数**。
   - **函数名称**：本文以 `layerDemo` 为例。
   - **地域**：地域默认填充。
   - **运行环境**：选择 **Nodejs 12.16**。
   - **时区**：云函数内默认使用 UTC 时间。
4. 在**函数代码**中，选择**本地上传文件夹**，并选择上传 步骤1 中已获取文件夹中的 `function` 文件夹。如下图所示：



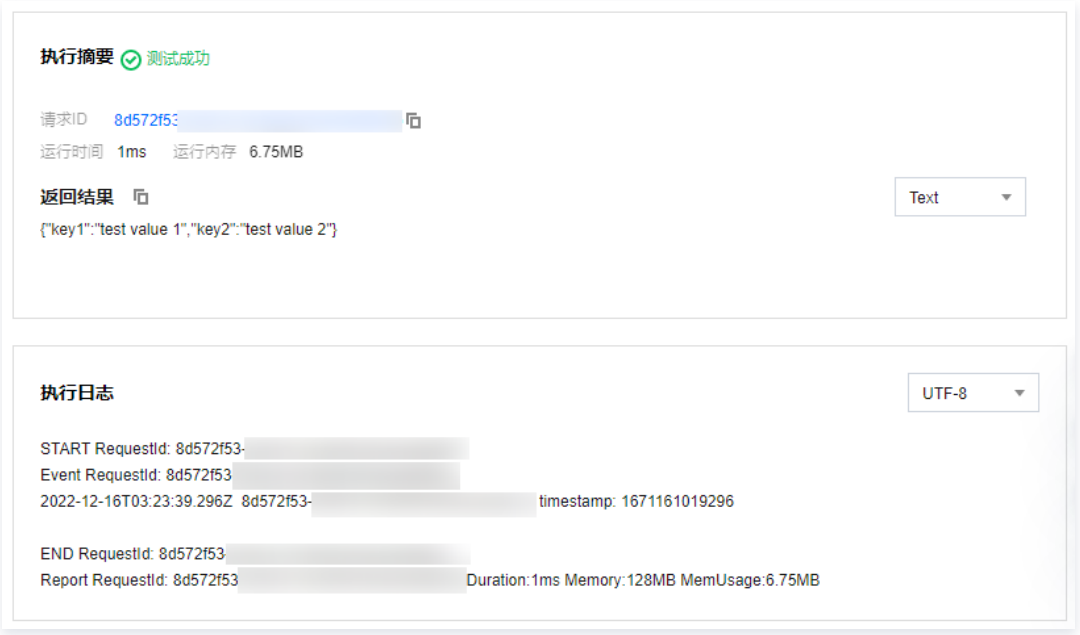
5. 在**高级配置** > **层配置**中，单击**添加层**。

6. 为函数选择层名称和层版本。如下图所示：



- **层名称**：选择 步骤2 中已创建的层 `demo` 。
- **层版本**：选择版本1。

7. 单击页面下方的**完成**。创建完成后即可查看函数详情。
8. 在**函数管理**中选择**函数代码**页签，单击页面下方的**测试**即可查看结果。如下图所示：



执行配置
异步执行

最近更新时间：2025-02-20 11:35:43

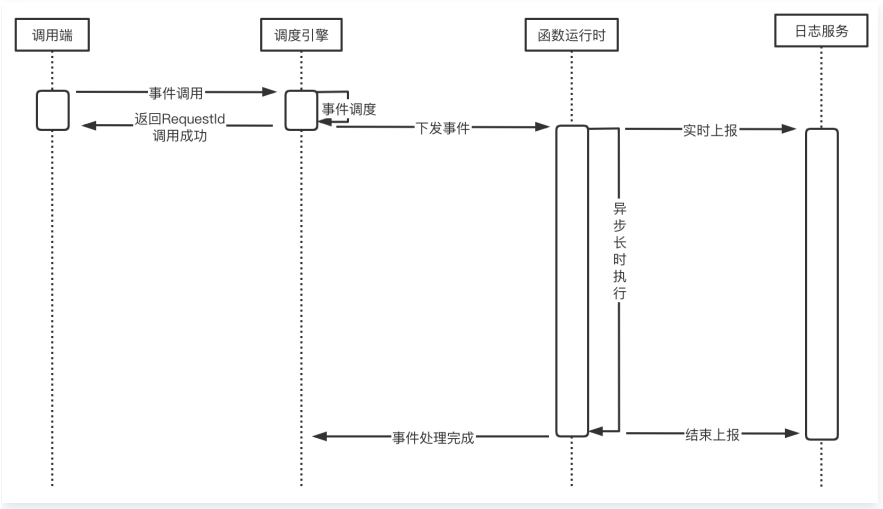
使用场景

在音视频转码、ETL 大体量数据处理、AI 推理等单任务重计算的场景下，函数的单实例运行时需要更多算力及更长时间的稳定运行。若函数的调用端长时间阻塞等待执行结果，不仅会持续占用调用方资源，还会对调用链路的稳定性产生较高要求。
云函数 SCF 提供了一种全新的函数运行机制，您可通过 SCF 提供的函数异步执行模式，提升执行超时时间上限和解决现有运行机制的问题。

运行机制

基础原理

函数启用异步执行后，通过同步（例如函数 URL）或异步（例如 COS、CKafka、Timer 等）调用端进行事件调用，函数将以异步执行模式响应事件。
即完成事件调度后立即返回事件的调用标识 RequestId，并结束调用操作，调用端无需阻塞等待。返回 RequestId 的同时，调用引擎将并行下发事件到函数运行时，开启函数逻辑执行。进入异步执行状态后，执行日志将实时上报至日志服务，提供对异步执行事件运行情况的实时反馈。其原理如下图所示：



注意事项

- 由于运行机制差异：
  - 暂不支持切换同步/异步执行模式。仅支持创建函数时选择是否开启“异步执行”功能，函数创建后该配置将锁定，不提供修改更新操作。
  - 暂不支持异步调用时，函数执行过程中报错的重试设置。
  - 异步执行函数执行异常均会触发实例回收。
- 事件调用成功，返回信息只包含 RequestId。事件执行结果需要在函数代码逻辑中自行实现回调特定的 API 或者发送通知消息。
- 异步执行目前支持最长执行时长为24小时。如需更长运行时长，可 提交工单 申请。
- 如果通过函数运行角色获取对其他云服务组件的访问权限，角色密钥有效期为 8~12 小时，如函数实际执行时间超出 8 小时，建议使用永久密钥。
- 异步运行函数调用 QPS 限制为 1000，超出部分将被限制，造成响应失败。

操作步骤

1. 登录 Serverless 控制台，单击左侧导航栏的**函数服务**。
2. 在主界面上方选择期望创建函数的地域和命名空间，并单击**新建**，进入函数创建流程。
3. 选择使用**模板创建**或选择使用**从头开始**来新建函数。
4. 在**函数配置**页面，展开**高级设置**，并勾选**异步执行**。如下图所示：



5. 单击**完成**即可创建函数。

状态追踪

最近更新时间：2023-03-16 16:45:17

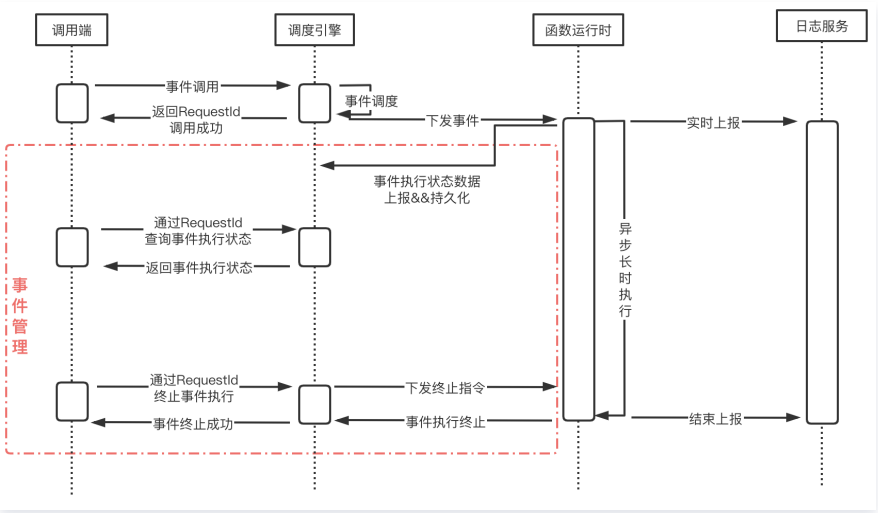
使用场景

异步执行函数通常用来处理大量异步长时任务，为了更好的对异步长时任务进行管理，SCF 提供了状态追踪功能，记录并上报事件响应的实时状态，并提供事件状态的统计、查询等事件管理相关服务。

运行机制

基础原理

异步执行函数状态追踪功能开启后，平台将开始记录并上报事件实时状态。其原理如下图所示：



注意事项

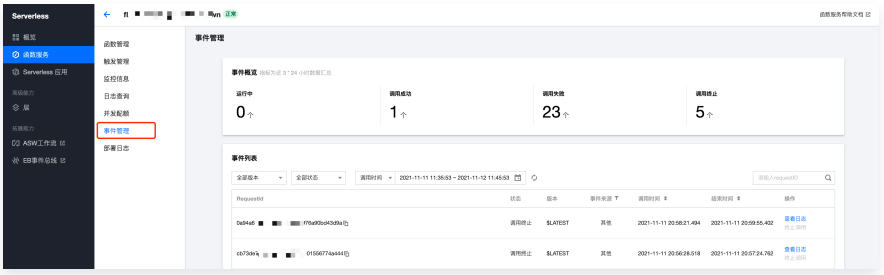
- 异步执行事件状态仅保留3天，将以3天为时间窗口滑动清理。如需保留全部记录，则需要定期拉取并保存至自有存储。
- 关闭状态追踪后，将停止提供异步执行事件相关记录、统计、查询等事件管理相关服务，已产生的事件状态数据将在3天内清空。
- 由于请求 QPS 超限、账户欠费等原因，事件调用将由调度引擎直接返回对应异常，不会生成事件状态记录。

操作步骤

1. 登录 Serverless 控制台，单击左侧导航栏的**函数服务**。
2. 在主界面上方选择期望创建函数的地域和命名空间，并单击**新建**，进入函数创建流程。
3. 选择使用**模板创建**或选择使用**从头开始**来新建函数。
4. 在**函数配置**页面，展开**高级设置**，勾选**异步执行**后，勾选**状态追踪**。如下图所示：



5. 单击**完成**。函数创建完成后，可单击**事件管理**查看异步事件列表。



异步执行事件管理

最近更新时间：2023-05-22 11:02:28

对于异步执行事件管理，SCF 提供了获取异步事件列表及状态、终止函数异步事件功能。

> **注意**
> 本文涉及功能仅支持 异步执行 函数。

获取异步事件列表及状态

异步执行函数事件有以下四种运行状态：

- 运行中：事件异步执行中。
- 调用完成：事件异步执行成功，正常返回。
- 调用失败：事件异步执行失败，异常返回。
- 调用终止：用户主动对运行中的事件发起终止，异步执行停止并返回。

相关接口

API 接口	说明	相关文档
ListAsyncEvents	异步执行函数事件信息数据列表，提供根据事件 RequestId、函数名、函数版本、事件状态、事件调用/结束时间等条件对异步执行事件信息的查询功能。 仅可查询数据范围为开启事件追踪后3天内数据。	拉取函数异步事件列表
GetAsyncEventStatus	提供根据请求 RequestId 获取异步事件执行状态的功能，事件状态保留3 * 24小时（从事件结束开始计时）。	获取函数异步事件状态

> **注意：**
> 使用时请关注各 API 的频率限制，获取异步事件列表接口 `ListAsyncEvents` 不建议高频调用，如需查询异步事件执行结果，请使用获取函数异步事件状态 `GetAsyncEventStatus` 接口。

终止函数异步事件

SCF 提供**终止调用**和**发送终止信号**两种异步事件终止方式，区别和使用方式如下：

相关接口

API 接口	说明	相关文档
TerminateAsyncEvent	异步执行函数通过调用返回的事件 RequestId，对运行中异步执行事件下发终止指令，终止完成后返回。本接口默认行为是终止调用，参数 GraceShutdown 为 True 时向请求发送 SIGTERM 终止信号，可在函数内部对该信号进行监听并自定义信号处理逻辑。	终止正在运行中的函数异步事件

终止调用

在调用函数时，SCF 会分配一个实例处理函数请求或事件。函数代码运行完毕返回后，该实例会处理其他请求。如果在请求到来时，所有实例都在运行中，云函数则会分配一个新的实例。（更多实例相关信息请参考 并发概述 ）
当异步执行函数事件接收到终止调用后，SCF 会强制停止实例运行，并回收该实例，下一次请求到来时，如无空闲实例，SCF 会分配一个新的实例来处理请求。
适用场景
异步执行函数运行异常、死循环等需要提前中断函数执行的场景。
注意事项
终止调用会强制停止实例并触发实例回收，这意味着实例中缓存的信息将无法正常获取（比如 /tmp 目录下的文件）。如需使用该功能，请及时将实例中缓存的问题写入其他持久化存储介质，避免实例回收后文件丢失。

发送终止信号

当调用终止函数异步事件 API 并指定 `GraceShutdown` 参数为 `True` 时，SCF 将会向 API 入参中指定的事件发送一个终止信号，该信号固定为 `SIGTERM`，可在函数中监听该信号并自定义接收到信号后的处理逻辑，包括但不限于停止函数执行。
当异步执行函数事件接收到 `SIGTERM` 信号后：

- 如果函数代码中监听并定义了信号处理函数，会开始执行对应的信号处理函数逻辑；
- 如果函数代码中没有监听信号，函数进程会退出，并返回 439 错误码（用户进程退出 `User process exit when running`）。
- SCF 会将事件的信号接收情况记录进函数执行日志中：
  - 信号接收成功：日志中将记录 [PLATFORM] Signal received successfully.
  - 信号接收失败：日志中将记录 [PLATFORM] Signal reception failed.

适用场景

在函数执行过程中，由于业务需要而停止函数运行，并在函数停止运行前自定义处理逻辑。

使用方法

下文以监听到 `SIGTERM` 信号后通过自定义信号处理函数终止函数运行为例：

代码部署

Python

```python
# -*- coding: utf8 -*-
import time
import signal

class GracefulKiller:
    kill_now = False
    def __init__(self):
    # Register signal processing function
        signal.signal(signal.SIGTERM, self.graceshutdown)
    def graceshutdown(self, *agrg):
        print("do something before shutdown.")
        self.kill_now = True

def main_handler(event, context):
    killer = GracefulKiller()

    while not killer.kill_now:
        time.sleep(1)
        print(killer.kill_now)

    print("Graceful shutdown.")
    return("END")
```

Golang

```golang
package main

import (
    "context"
    "fmt"
    "log"
    "os"
    "os/signal"
    "syscall"
    "time"

    "github.com/tencentyun/scf-go-lib/cloudfunction"
```

```
)

type DefineEvent struct {
    // test event define
    Key1 string `json:"key1"`
    Key2 string `json:"key2"`
}

func hello(ctx context.Context, event DefineEvent) (string, error) {
    go graceshutdown()
    sleepNum := 0
    for {
        sleepNum++
        fmt.Println("sleep:", sleepNum)
        time.Sleep(time.Second)
    }
}

// Register signal processing function
func graceshutdown() {
    sigs := make(chan os.Signal, 1)
    signal.Notify(sigs, syscall.SIGTERM)
    sig := <-sigs
    log.Printf("receive signal %s", sig.String())
    //do something before shutdown.
    os.Exit(0)
}

func main() {
    // Make the handler available for Remote Procedure Call by Cloud Function
    cloudfunction.Start(hello)
}
```

镜像部署

Python

```
# -*- coding: utf8 -*-
from flask import Flask, request
import time
import signal
app = Flask(__name__)

class GracefulKiller:
    kill_now = False
    def __init__(self):
    # Register signal processing function
        signal.signal(signal.SIGTERM, self.graceshutdown)
    def graceshutdown(self, *agrg):
        print("do something before shutdown.")
        self.kill_now = True

@app.route('/event-invoke', methods = ['POST'])
def invoke():
    while not killer.kill_now:
        time.sleep(1)
        print(killer.kill_now)

    print("Graceful shutdown.")
```

```
    return("END")

if __name__ == '__main__':
    killer = GracefulKiller()
    app.run(host='0.0.0.0', port=9000)
```

版权所有：腾讯云计算（北京）有限责任公司

命名空间管理

最近更新时间：2022-12-22 16:27:07

操作场景

命名空间为函数提供了相对独立的运行环境，在创建云函数时，您可以选择函数所在的命名空间，从而更有效地管理您的云函数。

使用限制

针对命名空间的使用，有如下限制：

- 命名空间名称最长60个字符，以字母开头，名称可包含 a－z，A－Z，0－9，-，_，且需要以数字或字母结尾，例如 `Tencent-Cloud_Space1` 。
- Default 空间不可修改或删除。
- 目前每个地域内默认可创建5个命名空间，每个空间内可新建50个函数。若想提高配额限制，您可通过 提交工单 进行申请。

操作步骤

通过控制台查看命名空间

1. 登录 Serverless 控制台 。
2. 单击左侧导航栏**函数服务**，进入函数服务管理页面。
   在该页面左上方即可查看到**命名空间**的信息。例如 default(1) ，其中，default 为当前函数所在命名空间，1为命名空间内的函数数量。
   如需查看不同命名空间下部署的函数，可单击**命名空间**下拉列表进行选择和切换。

管理命名空间

您可通过 Serverless 控制台 对命名空间进行自主管理。例如命名空间的创建、修改、删除等操作，具体操作步骤如下：

1. 登录 Serverless 控制台 。
2. 单击左侧导航栏**函数服务**，进入函数服务管理页面。
3. 在该页面左上方找到**命名空间**，单击右侧 ⚙，进入命名空间管理界面。如下图所示：



> ⚠ **注意**
> - 空间名称一旦确定不可更改。
> - **default** 空间为默认命名空间，如未特别设置，函数将默认创建在此空间内。

4. 在命名空间管理页面内，可进行如下操作：
   - 新建该地域内的命名空间：单击**新增命名空间**，输入命名空间名称并选择**提交**即可完成创建。
   - 修改命名空间描述：对描述信息进行编辑，完成后单击**提交**即可完成修改。
   - 删除命名空间：请先移除需删除命名空间下的所有函数，并在“命名空间管理”页面中单击该命名空间右侧**删除**即可删除。

ICP 备案

最近更新时间：2024-11-12 17:26:22

为什么要备案?

根据国务院令第292号《互联网信息服务管理办法》和工信部令第33号《非经营性互联网信息服务备案管理办法》规定，国家对经营性互联网信息服务实行许可制度，对非经营性互联网信息服务实行备案制度。未获取许可或者未履行备案手续的，不得从事互联网信息服务，否则属于违法行为。
因此，使用中国内地（大陆）的 Serverless 服务开办网站并绑定域名服务时必须先办理网站备案，备案成功并获取通信管理局下发的 ICP 备案号后才能开通域名访问。

- ICP 备案号以工信部网站公共查询为准：查询入口
- 更多相关法律法规请参阅：法律法规

备案场景

如果您的网站托管在腾讯云中国内地（大陆）的 Serverless 服务中，且网站的主办者和域名从未办理过备案，则在开通 Serverless 服务并且使用云函数 SCF 进行自定义域名的 HTTP 访问服务前，需在腾讯云备案系统进行首次备案的操作。

备案准备

- 为了节约备案时间和顺利通过备案，建议您提前了解备案流程。
- 因各地管局要求不同，需准备的材料也有所不同。建议您提前了解各省、自治区、直辖市管局的 备案要求，以及相关 备案限制。
- Serverless 备案要求：备案本身不收取任何费用，但通过 Serverless 方式备案需购买云函数**5000万次**调用次数包与**40万GBs**资源用量包。请前往 资源包购买页面 完成购买。

> ⚠ **注意：**
> Serverless 备案方式已向全部用户开放。

备案流程

请参考 首次备案，通过小程序完成备案操作。

> ⚠ **注意：**
> 在执行 填写网站信息 步骤时，需先开启云函数备案，即在备案类型中选择 **Serverless**。

常见问题

使用 Serverless 的访问域名必须要备案吗?

需根据实际情况进行判断。若默认采用 API 网关提供的三级域名访问自身是无需进行备案的，只有需要自定义域名且该域名指向中国内地（大陆）的 Serverless 服务才需要备案。例如，访问博客页面等场景。您可根据以下场景判断是否需要备案：

- 不需要备案：
  域名解析指向托管于非中国内地（大陆）的 Serverless 服务时，例如中国香港服务器，则不需要备案。
- 需要备案：
  域名指向中国内地（大陆）的 Serverless 服务时，需要完成备案。

Serverless 备案流程与云服务器 CVM 备案流程是否一致?

Serverless 备案与 CVM 备案流程上并无实质性差别，所有体验完全一致。唯一的差异点在于，CVM 备案过程中填写的 IP 将直接暴露给用户，Serverless 备案过程中 IP 将由系统自动拉取并提供。

为什么备案时 IP 地址不可访问?

备案在腾讯云，解析也需要在腾讯云。用户在使用腾讯云 Serverless 方式备案过程中，域名解析 IP 需指向腾讯云站点的 IP 地址 。

Serverless 备案有限制吗?

由于 Serverless 较为轻量，可能会实时删除或新增。为了迎合用户使用 Serverless 习惯，Serverless 备案将以账号作为维度，每个账号支持购买1个云函数资源包，支持备案2个网站。

已经在 CVM 备案的域名是否需要重新备案?

若同账号下已在 CVM 进行 ICP 备案的域名，可直接通过 API 网关绑定，无需重复备案。

注册的域名当前不使用，还需要备案吗?

域名本身无需备案的，但需实名认证。仅当该域名开通 Web 服务时，才需要备案。

Serverless 站点未完成搭建，需要办理备案吗?

不需要备案。在您的 Serverless 站点正式提供对外访问前，才需要备案。由于备案需要一定的办理时间，建议您提前在腾讯云办理备案，以便您的站点做好之后可以马上投入使用。

已备案域名接入腾讯云 Serverless 服务，是否需要重新备案?

不需要重新备案，但要办理接入备案。详情请参见 接入备案 。

什么情况下需要新增网站备案?

- 如果您存在多个域名，都需要进行备案。
- 您已经有域名进行过备案，现在需要备案新的域名。

接入备案是否可以接入多个 Serverless 服务?

同一主体备案信息可以同时接入多个网站信息，最多同时可以接入10个云函数或 CVM 备案信息。

Serverless 备案资源包购买后是否支持退款?

资源包购买之后立即生效，暂不支持退款。五天无理由退款及其他特殊退款请通过 在线客服 进行咨询。通过人工渠道成功退款后，将无法重购备案资源包。

扩展存储管理
挂载 CFS 文件系统

最近更新时间：2022-12-26 15:31:20

操作场景

腾讯云文件存储 CFS 提供可扩展的共享文件存储服务，可与腾讯云服务器、容器服务或者批量处理等服务搭配使用。CFS 符合标准的 NFS 文件系统访问协议，为多个计算节点提供共享的数据源，支持弹性容量和性能的扩展，现有应用无需修改即可挂载使用，是一种高可用、高可靠的分布式文件系统，适合于大数据分析、媒体处理和内容管理等场景。
CFS 成本低廉，采用按量计费模式，以小时为计费周期，您只需为实际使用的存储空间付费。CFS 计费详情请参见 计费概述 。
腾讯云云函数 SCF 支持与 CFS 无缝集成，只需进行相关配置，您的函数即可轻松访问存储在 CFS 文件系统中的文件。使用 CFS 的优势如下：

- 函数执行空间不受限。
- 多个函数可共用一个文件系统，实现文件共享。

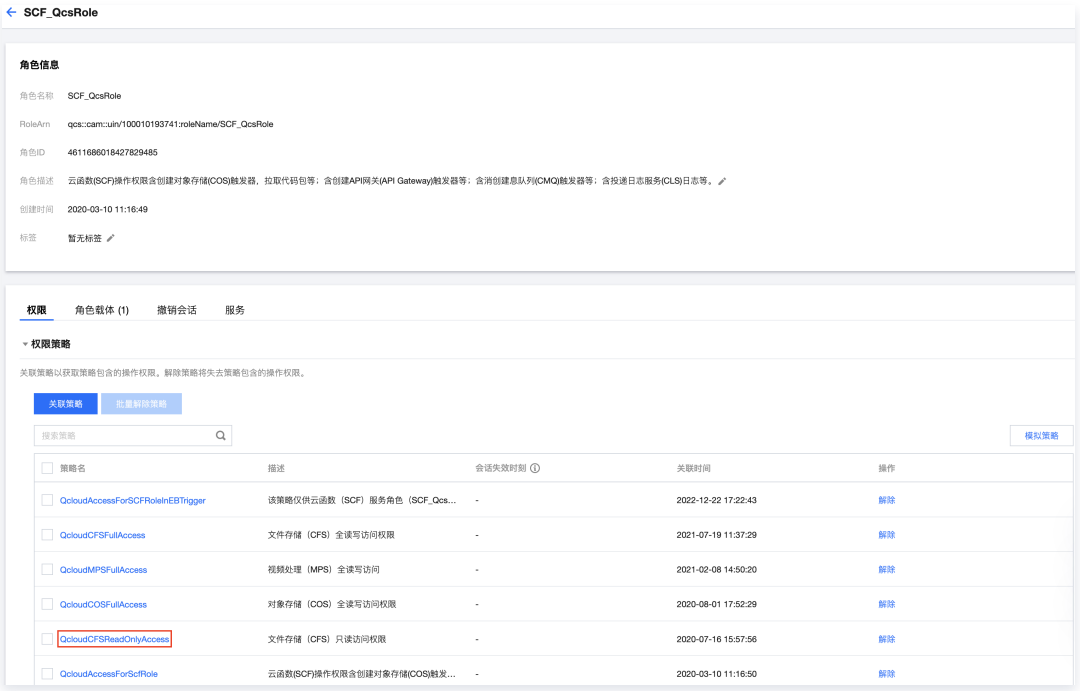
操作步骤

关联授权策略

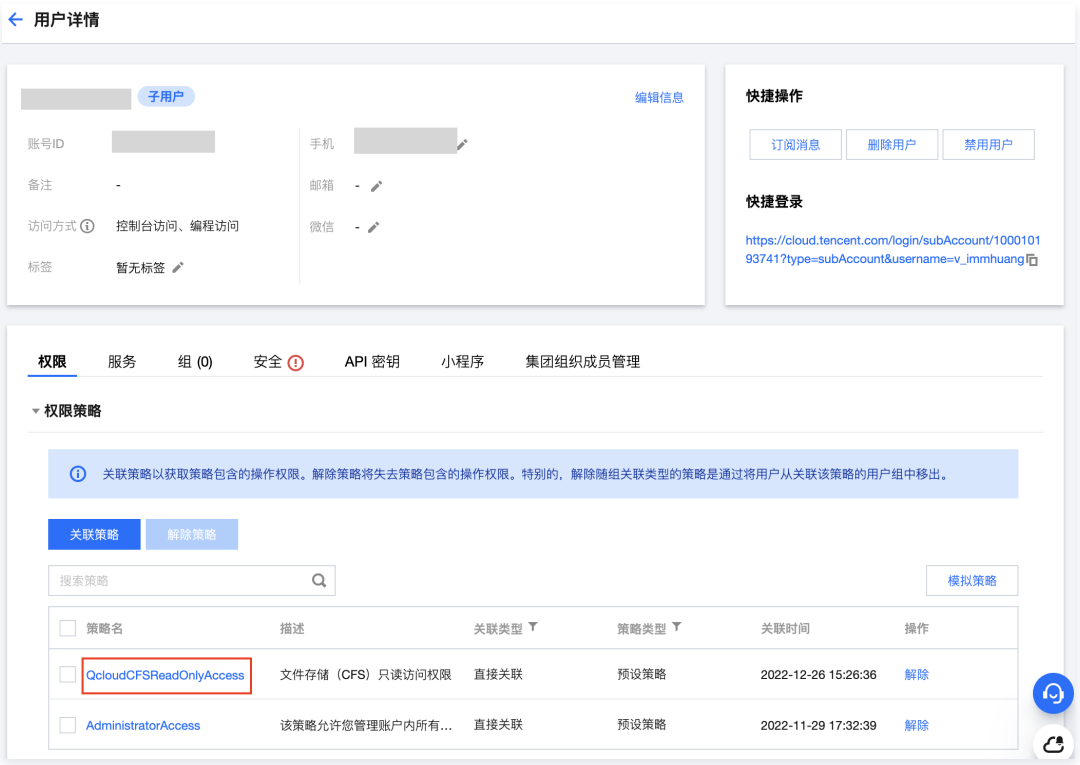
> **注意**
> 如需使用 CFS 功能，云函数需要能够操作您 CFS 资源的权限。

请参考以下步骤为账号进行授权操作：

1. 请参考 修改角色，为 `SCF_QcsRole` 角色关联 `QcloudCFSReadOnlyAccess` 策略。关联成功则如下图所示：
   如您使用的账号未进行该操作，则可能出现函数无法保存，CFS 相关功能无法使用等问题。



2. 如您使用账号为子账号，则请联系主账号并参考 子用户权限设置 为您的子账号关联 `QcloudCFSReadOnlyAccess` 策略。关联成功则如下图所示：
   如您使用的子账号未进行该操作，则可能出现无法使用 CFS 相关功能的问题。



创建私有网络 VPC

请参考 快速搭建 IPv4 私有网络 完成 VPC 创建。

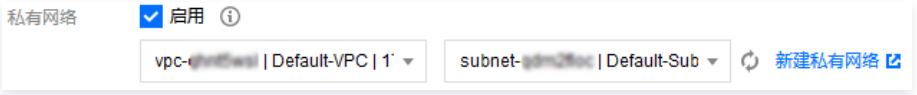
创建 CFS 资源

请参考 创建 CFS 文件系统 完成创建操作。

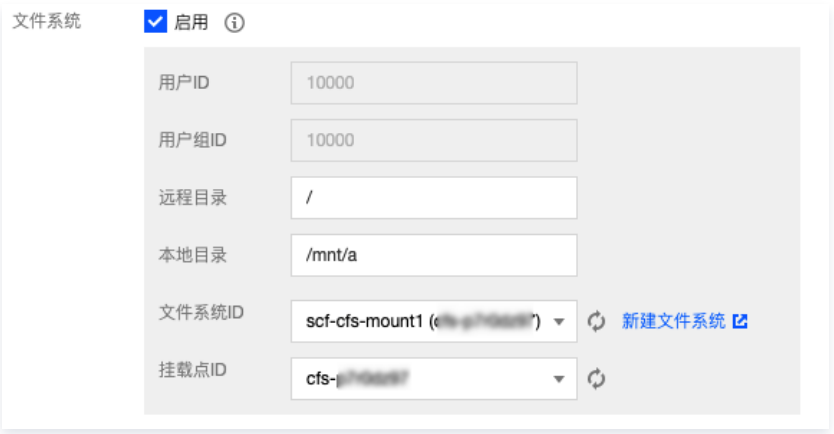
> **注意**
> 目前云函数仅支持添加网络类型为 VPC 的 CFS 文件系统作为挂载点。请在创建的 CFS 文件系统时，选择与函数所在相同的 VPC，以确保网络能够互通。

挂载并使用 CFS 文件系统

1. 登录 Serverless 控制台，单击左侧导航栏的函数服务。
2. 在“函数服务”页面，选择需配置的函数名。
3. 在“函数管理”页面的**函数配置**页签中，单击右上角的**编辑**。
4. 在“私有网络”中，勾选启用并选择 CFS 文件系统所在的 VPC。如下图所示：



5. 在“文件系统”中勾选启用，并按照以下信息进行挂载。如下图所示：



- **用户ID及用户组ID**：这两个值等同于 CFS 文件系统中的用户及用户组。云函数默认用户及用户组值为 10000，来操作您的 CFS 文件系统。请按需设置文件的拥有者及相应组的权限，并确保您的 CFS 文件系统已配置相应权限。一个简单的例子是执行如下命令：`chown 10000:10000 -R /mnt/folder`。详情请参见 权限设置。
- **远程目录**：为云函数需访问 CFS 文件系统的远端目录，由文件系统和远端目录两部分组成。
- **本地目录**：为本地文件系统的挂载点。您可使用 `/mnt/` 目录的子目录挂载 CFS 文件系统。
- **文件系统ID**：在下拉列表中选择需挂载的文件系统。
- **挂载点ID**：在下拉列表中选择对应文件系统的挂载点 ID。

6. 单击页面下方的**保存**即可完成配置。
您可执行以下函数代码，开始使用 CFS 文件系统。

```
'use strict';
var fs = require('fs');
exports.main_handler = async (event, context) => {
  await fs.promises.writeFile('/mnt/myfolder/file1.txt', JSON.stringify(event));
  return event;
};
```

SCF 使用 CFS 文件系统性能测试

您可以使用此 脚本 测试 SCF 使用 CFS 时的性能。

DNS 缓存配置

最近更新时间：2023-05-12 17:01:27

概述

当客户端向某个地址发起访问时，通常会查询本地 DNS 缓存中是否有相关记录，有则会直接访问对应 IP 地址，如果没有则会委托递归服务器进行全球查询。
由于 DNS 域名解析采用 UDP 协议通讯，受网络环境影响较大，极端情况下域名解析可能有数秒的延时。在云函数的使用场景下，域名解析延时有可能导致函数执行超时失败，影响正常的业务逻辑；在函数高频调用的情况下，有可能导致 DNS 服务器解析超出频率限制，同样导致函数执行失败。
云函数提供了 DNS 缓存配置来解决上述问题。DNS 缓存可以提升域名解析效率，缓解网络抖动等因素对域名解析成功率的影响。

适用场景

适用于在函数代码中请求了某个地址，且函数被高频调用的场景。

操作步骤

由于实现机制的不同，代码部署的事件函数、Web 函数、镜像部署的函数请分别参考以下步骤开启 DNS 缓存。

代码部署的事件函数

1. 登录 Serverless 控制台，选择需要启用 DNS 缓存配置的函数，进入函数详情页。
2. 在函数配置页面，单击右上角**编辑**，在编辑状态中勾选**启用 DNS 缓存**。如下图所示：



3. 单击**保存**完成函数配置更新。

Web 函数

1. 在 Web 函数的启动文件 scf_bootstrap 中添加下述命令，以启动 nscd 进程开启 DNS 缓存。

```
/var/lang/bin/nscd -f /var/lang/conf/nscd.conf
```

2. 将更新后的 `scf_bootstrap` 同函数代码一起部署到云上，函数代码更新后的调用即可使用 DNS 缓存功能。

镜像部署函数

1. 在镜像制作过程中安装 nscd。以 centos 为例，可执行以下命令安装 nscd。

```
yum install nscd -y
```

2. 将默认的 `/etc/nscd.conf` 更新为以下内容：

```
#
# /etc/nscd.conf
#
# An example Name Service Cache config file. This file is needed by nscd.
#
# WARNING: Running nscd with a secondary caching service like sssd may lead to
# unexpected behaviour, especially with how long entries are cached.
#
# Legal entries are:
#
# logfile <file>
# debug-level <level>
# threads <initial #threads to use>
# max-threads <maximum #threads to use>
```

```
# server-user <user to run server as instead of root>
# server-user is ignored if nscd is started with -S parameters
# stat-user <user who is allowed to request statistics>
# reload-count unlimited|<number>
# paranoia <yes|no>
# restart-interval <time in seconds>
#
# enable-cache <service> <yes|no>
# positive-time-to-live <service> <time in seconds>
# negative-time-to-live <service> <time in seconds>
# suggested-size <service> <prime number>
# check-files <service> <yes|no>
# persistent <service> <yes|no>
# shared <service> <yes|no>
# NOTE: Setting 'shared' to a value of 'yes' will accelerate the lookup,
# but those lookups will not be counted as cache hits
# i.e. 'nscd -g' may show '0%'.
# max-db-size <service> <number bytes>
# auto-propagate <service> <yes|no>
#
# Currently supported cache names (services): passwd, group, hosts, services
#


# logfile /var/log/nscd.log
# threads 4
# max-threads 32
server-user root
# stat-user somebody
debug-level 0
reload-count 2
paranoia no
# restart-interval 3600


enable-cache passwd no
positive-time-to-live passwd 600
negative-time-to-live passwd 20
suggested-size passwd 211
check-files passwd yes
persistent passwd yes
shared passwd yes
max-db-size passwd 33554432
auto-propagate passwd yes


enable-cache group no
positive-time-to-live group 3600
negative-time-to-live group 60
suggested-size group 211
check-files group yes
persistent group yes
shared group yes
max-db-size group 33554432
auto-propagate group yes


enable-cache hosts yes
positive-time-to-live hosts 300
negative-time-to-live hosts 0
suggested-size hosts 211
```

```
check-files hosts no
persistent hosts no
shared hosts yes
max-db-size hosts 8388608


enable-cache services no
positive-time-to-live services 600
negative-time-to-live services 3
suggested-size services 211
check-files services yes
persistent services yes
shared services yes
max-db-size services 33554432


enable-cache netgroup no
positive-time-to-live netgroup 28800
negative-time-to-live netgroup 20
suggested-size netgroup 211
check-files netgroup yes
persistent netgroup yes
shared netgroup yes
max-db-size netgroup 33554432
```

3. 在启动文件 `scf_bootstrap` 中添加下述命令，以启动 nscd 进程开启 DNS 缓存。
以 centos 为例，将下述命令添加到启动文件中：

```
${PATH}/nscd -f /etc/nscd.conf
```

> **注意**
> `${PATH}` 为 nscd 安装的绝对路径。

资源托管模式管理

最近更新时间：2024-03-25 16:21:42

函数资源托管模式概述

函数资源托管模式决定了函数 SCF 运行时的资源池。默认情况下，开启函数服务后，平台会为每一个地域都分配一个函数公有云资源池，资源池由一些底层机器组成，体现为 128GB 的函数并发配额，您还可以通过购买套餐包的方式提升地域甚至是命名空间的并发额度，平台会根据新的并发额度自动分配匹配的机器，保障函数运行。
为了更好的支持您在不同业务场景下的需求，函数现已支持自定义资源托管模式，以满足函数运行到您指定的基础设施上，例如公有云 TKE 集群、混合云、IDC 等。
目前平台已推出 K8s 资源托管模式，以支持函数运行在您自己的 TKE 集群中，以实现在统一的云原生资源底座上使用函数加速业务开发。后续将逐步迭代支持混合云等更多云原生基础设施。

函数资源托管模式类型

SCF 支持**默认资源托管模式**和 **K8s 资源托管模式**两种类型。

- **默认资源托管模式**，函数运行在函数平台各个地域下的公有云资源池中，由函数平台完全掌控底层机器的供给和调度，您只需关注实际业务量级需要，通过调整函数的并发额度以保障业务运行。
- **K8s 资源托管模式**，函数运行在您指定的 K8s 集群中，由您管理 K8s 集群中的资源供给，函数平台完全掌控函数的请求调度，在给定的资源池中智能调用，充分利用资源。

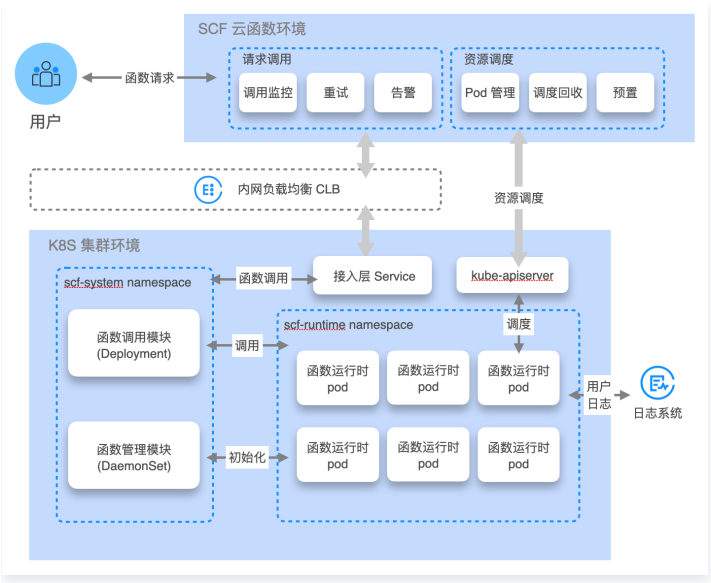
k8s 资源托管模式

概述

在 K8s 资源托管模式下，可选择一个 TKE 集群作为函数的计算资源池，函数的请求调用将全部调度到该资源池中，函数侧将不产生费用。目前支持 TKE 集群原生节点和普通节点，暂不支持超级节点。
使用上，只需要在函数命名空间中，配置资源托管模式为 K8s，并绑定一个 TKE 集群，即可开启该模式。该命名空间下的所有函数请求都将调度到绑定的 TKE 集群中。

运行原理

K8s 资源托管模式下的函数调度原理如下图所示：



TKE 集群初始化

当您为函数命名空间指定了 TKE 集群和函数运行时 namespace 后，平台将在该集群中自动创建 scf-system namespace，并创建部署管理函数代码等元信息的 daemonset、请求转发 pod、内网 clb service 组件等。

函数请求调度生命周期

用户发送的函数请求首先会发送到函数环境中的请求调用入口，函数调度管理模块会分析该函数在 TKE 集群中是否有空闲的函数运行时 Pod 资源可供运行：

1. 如果没有空闲资源，则会通过资源调度模块下发调度请求到 TKE 集群，进行函数运行时 Pod 资源准备。在 Pod 完备后，会通过 TKE 集群中的函数管理模块将函数的代码等元信息准备好，最后将新增的运行资源上报到函数环境的调度管理模块。
2. 如果有空闲资源，则会通过内网 CLB 将请求转发到 TKE 集群中的接入层 Service，然后通过集群中的函数的调用模块将请求下发到函数运行时 Pod，进入函数执行阶段。函数执行过程中，日志会实时上报到用户的 CLS 日志系统。函数执行结束后，执行结果、监控指标等信息会发回函数环境中的请求调用模块。

功能与优势

相较于默认资源托管模式，K8s 资源托管模式具备以下优势：

- 函数可以跑在您指定的 K8s 集群中，更加灵活可控，可实现更好的成本控制和更强的基础设施资源管理。
- 函数的主动调度机制，可大幅度提升您的 K8s 集群资源利用率，不仅通过函数开发体验提升了研发效率，还能降低资源浪费，真正实现了降本增效。
- 通过和 K8s 生态融合，可实现全栈云原生研效体系和服务治理机制，为业务开发者带来先进的开发体验，为线上业务带来更高的可用性保障。

操作步骤

创建函数命名空间并绑定 TKE 集群

1. 登录 Serverless 控制台，单击左侧导航栏的**函数服务**。
2. 在函数服务页面上方选择期望创建函数的地域，单击命名空间右侧的⚙，进入命名空间管理。如下图所示：



3. 在**命名空间**弹窗中，单击**新增命名空间**，进入命名空间创建页面。如下图所示：



4. 在**资源托管模式**选项中，选择 K8s，如果是第一次操作，会弹出容器服务角色授权弹窗，根据指示完成授权后可进行下一步操作。
5. 在 **TKE 集群配置**中，选择 TKE 集群以及该集群下的 namespace，平台将在指定集群的 scf-system namespace 下创建 daemonset、内网 clb 等函数服务支撑组件，同时将在指定的 namespace 下创建函数运行时 Pod。**请注意所选 TKE 集群内需有节点，且节点类型是普通节点和原生节点，以确保初始化过程顺利完成。**
6. 在**函数 vpc 子网**配置中，指定子网，平台将在该子网下消耗一个 ip 创建一个内网 clb 作为函数请求入口，实现函数请求转发到 TKE 集群。**请注意子网不支持 9.x.x.x 网段。**
7. 除以上的基础配置项外，还可以根据需要配置下面几项：
   - **函数目录**：指定一个 TKE 集群节点上的路径，用以存储函数代码、layer 代码及用作函数运行过程中产生的日志等临时存储。
   - **支撑服务端口**：指定一个可用的端口号，函数支撑服务将监听该端口号以实现函数调度链路。
   - **NodeSelector**: 可根据调度规则，将函数实例调度到符合预期的Label的节点中。详见下述 设置函数实例在 TKE 集群中的调度策略 章节 。
   - **污点容忍调度**：可根据调度规则，将函数实例调度到符合预期的污点的节点中。详见下述 设置函数实例在 TKE 集群中的调度策略 章节 。
8. 单击**创建**，在弹出的二次确认弹窗中单击**继续**，进入 TKE 集群的函数支撑组件初始化流程，该过程将花费 20s 左右，完成后，将会在**命名空间管理**页面中看到状态更新为“正常”，如下图所示：



9. 切换到已创建好的函数命名空间下创建函数开始使用。

设置函数实例在 TKE 集群中的调度策略



通过设置 NodeSelector 和污点容忍调度策略，指定函数实例在 TKE 集群内进行调度。更加有效利用集群内的资源，详情见 容器服务-资源合理分配 。存在以下应用场景：

- 将函数实例运行在指定的节点上。
- 将函数实例运行在某一作用域（作用域可以是可用区、机型等属性）的节点上。

前置条件

- 设置工作负载高级设置中的调度规则，且集群的 Kubernetes 版本必须是1.7以上的版本。
- 为确保您的 Pod 能够调度成功，请确保您设置的调度规则完成后，节点有空余的资源用于容器的调度。
- 使用自定义调度功能时，需要为节点设置对应 Label 或 污点。详情请参见 设置节点 Label 和 设置节点污点 。

设置调度规则

NodeSelector

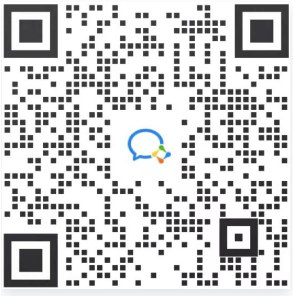
可通过自定义调度规则，匹配节点标签，将函数实例调度到指定节点上。调度期间如果满足亲和性条件，则调度到对应 Node。如果没有节点满足条件，则调度失败。
详情请参见 k8s 节点亲和性 。

污点容忍调度

可通过自定义调度规则，容忍节点污点，将函数实例调度到指定节点上。
详情请参见 k8s 污点和容忍度 。

联系我们

如果您在使用过程中遇到任何问题，请随时加入我们的群组进行沟通：



版权所有：腾讯云计算（北京）有限责任公司