云函数

开发指南

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2024 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

开发指南
- 基本概念
- 测试云函数
- 环境变量
- 依赖安装
- 错误类型与重试策略
- 死信队列
- 自动化部署
- 云函数状态码
- 常见错误码解决方法

开发指南
基本概念

最近更新时间：2023-08-31 09:28:31

云函数（Serverless Cloud Function，SCF）提供代码部署、镜像部署两种部署方式，支持事件函数和 Web 函数两种函数类型。不同的部署方式以及函数类型在代码开发时需要采用不同的规范。本文主要介绍代码部署的事件函数的编写规范及相关概念，镜像部署 和 Web 函数 详情请参考对应文档。
SCF 事件函数有三个基本概念：执行方法、函数入参和函数返回。

> **注意**
> 上述概念在通常的项目开发中分别对应：
> **执行方法**：对应项目的主函数，是程序执行的起点。
> **函数入参**：即通常理解的函数入参，但在云函数环境下，入口函数的入参为平台固定值，详情见 函数入参 。
> **函数返回**：对应项目中主函数的返回值，函数返回后，代码执行结束。

执行方法

SCF 平台在调用云函数时，首先会寻找执行方法作为入口，执行用户的代码。此时，用户需以**文件名.执行方法名**的形式进行设置。例如，用户设置的执行方法为 `index.handler`，则 SCF 平台会首先寻找代码程序包中的 `index` 文件，并找到该文件中的 `handler` 方法开始执行。
在执行方法中，用户可对入口函数入参进行处理，也可任意调用代码中的其他方法。SCF 的某个函数以入口函数执行完成或函数执行异常作为执行结束。

函数入参

函数入参，是指函数在被触发调用时所传递给函数的内容。通常情况下，函数入参包括 **event** 和 **context** 两部分，但根据开发语言和环境的不同，入参个数可能有所不同，详情请参见 开发语言说明 。

event 入参

作用

event 参数类型为 dict，event 中包含了触发函数执行的基本信息，可以是平台定义的格式，也可以自定义格式。函数被触发开始执行后，可以在代码内部对 event 进行处理。

使用说明

有两种方法可以触发云函数 SCF 执行：
1. 通过调用 云 API 触发函数执行。
2. 通过绑定 触发器 触发函数执行。

SCF 两种触发方式对应两种event格式：
- 云 API 触发函数执行：
  可以在调用方和函数代码之间自定义一个 dict 类型的参数。调用方按照定义好的格式传入数据，函数代码按格式获取数据。

示例：
定义一个 dict 类型的数据结构 {"key":"XXX"}，当调用方传入数据 {"key":"abctest"} 时，函数代码可以通过 event[key] 来获得值 abctest。

- 触发器触发函数执行：
  SCF 和 API 网关、对象存储 COS、消息队列 Ckafka 等多种云服务打通，可以通过给函数绑定对应的云服务触发器触发函数执行。触发器触发函数时，event 会以一种平台预定义的、不可更改的格式作为 event 参数传给函数，您可以根据此格式编写代码并从 event 参数中获取信息。
  示例：
  通过对象存储 COS 触发函数时会将对象存储桶及文件的具体信息以 JSON 格式 传递给 event 参数，在函数代码中通过解析 event 信息即可完成对触发事件的处理。

context 入参

作用

context 为 SCF 平台提供的入参，将 context 入参传递给执行方法，代码可通过解析 context 入参对象，获取到运行环境及当前请求的相关信息。

使用说明

SCF 提供的入参 context 包含的字段及含义如下：

字段名称	描述
memory_limit_in_mb	函数配置内存
time_limit_in_ms	函数执行超时时间
request_id	函数执行请求 ID
environment	函数命名空间信息
environ	函数命名空间信息
function_version	函数版本信息
function_name	函数名称
namespace	函数命名空间信息
tencentcloud_region	函数所在地域
tencentcloud_appid	函数所属腾讯云账号 APPID
tencentcloud_uin	函数所属腾讯云账号 ID

> ⚠ 注意

> 为保证兼容性，context 中保留了 SCF 不同阶段对命名空间的描述方式。
> context 结构内容将会随着 SCF 平台的开发迭代而增加。

您可以在函数代码中通过标准输出语句打印 context 信息，以 `python` 运行环境为例：

```
# -*- coding: utf8 -*-
import json
def main_handler(event, context):
    print(context)
    return("Hello World")
```

可得到以下context信息：

```
{"memory_limit_in_mb": 128, "time_limit_in_ms": 3000, "request_id": "f03dc946-3df4-45a0-8e54-xxxxxxxxxxxx", "environment": "{\"SCF_NAMESPACE\":\"default\"}", "environ": "SCF_NAMESPACE=default;SCF_NAMESPACE=default", "function_version": "$LATEST", "function_name": "hello-from-scf", "namespace": "default", "tencentcloud_region": "ap-guangzhou", "tencentcloud_appid": "12xxxxx384", "tencentcloud_uin": "10000xxxxx36"}
```

了解 event 入参和 context 入参的基本用法后，在编写函数代码时您还需注意以下几点：

- 为保证针对各开发语言和环境的统一性，event 入参和 context 入参均使用 JSON 数据格式统一封装。
- 不同触发器在触发函数时，所传递的数据结构均有所不同。详情请参见 函数触发器说明 。
- 当云函数不需要任何输入时，您可以在代码中忽略 event 和 context 参数。

函数返回

SCF 平台会获取到云函数执行完成后的返回值，并根据下表中不同的触发方式进行处理。

触发方式	处理方式
同步触发	通过 API 网关、云 API 同步 invoke 触发函数的方式为同步触发。 使用同步方式触发的函数在执行期间，SCF 平台不会返回触发结果。 在函数执行完成后，SCF 平台会将函数返回值封装为 JSON 格式并返回给调用方。
异步触发	使用异步方式触发的云函数，SCF 平台接收触发事件后，会返回触发请求 ID 。 在函数执行完成后，函数的返回值会封装为 JSON 格式并存储在日志中。 用户可在函数执行完成后，通过返回的请求 ID 查询日志获取该异步触发函数的返回值。

当函数中的代码返回具体值时，通常返回特定的数据结构。例如：

运行环境	返回数据结构类型
Python	简单数据结构或 dict 数据结构
Node.js	JSON Object
PHP	Array 结构

GO	简单的数据结构或带有 JSON 描述的 struct

为保证针对各开发语言和环境的统一性，函数返回会使用 **JSON 数据格式统一封装**。SCF 平台在获取到例如以上运行环境函数的返回值后，将会对返回的数据结构进行 JSON 化，并返回 JSON 内容到调用方。

> **注意**
> - 需确保函数的返回值均可被 JSON 化，若直接返回对象且不具备 JSON 化方法，将会导致 SCF 平台在 JSON 化操作时失败并报错。
> - 例如以上运行环境的返回值，无需在 return 前自行 JSON 化，否则会导致输出的字符串会进行再次 JSON 化。

异常处理

若函数在调试和运行过程中出现异常，SCF 平台会尽最大可能捕获异常并将异常信息写入日志中。函数运行产生的异常包括捕获的异常（Handled error）和未捕获的异常（Unhandled Error）。

处理方式

您可以前往 Serverless 控制台，按照以下步骤进行异常处理测试：

1. 新建函数并复制以下函数代码，不添加任何触发器。
2. 单击控制台**测试**，选择 “Hello World” 测试示例进行测试。

本文提供以下三种抛出异常方式，您可根据实际需求选择在代码中进行异常处理。

显式抛出异常

示例

```
def always_failed_handler(event,context):
    raise Exception('I failed!')
```

说明

此函数在运行过程中将引发异常，返回以下错误信息，SCF 平台会将此错误信息记录到函数日志中。

```
File "/var/user/index.py", line 2, in always_failed_handler
raise Exception('I failed!')
Exception: I failed!
```

继承 Exception 类

示例

```
class UserNameAlreadyExistsException(Exception): pass
def create_user(event):
    raise UserNameAlreadyExistsException('
        The username already exists,
```

```
        please change a name!')
```

说明

您可以在代码中自行定义错误的处理方式，保障应用程序的健壮性和可扩展性。

使用 Try 语句捕获错误

示例

```
def create_user(event):
    try:
        createUser(event[username],event[pwd])
    except UserNameAlreadyExistsException,e: //catch error and do something
```

说明

您可以在代码中自行定义错误的处理方式，保障应用程序的健壮性和可扩展型。

返回错误信息

当用户的代码逻辑中未进行异常处理及错误捕获时，SCF 平台会尽可能的捕获错误。例如，用户函数在运行过程中突然崩溃退出，当出现此类平台也无法捕获错误的情况时，系统将会返回一个通用的错误信息。
下表展示了代码运行中常见的一些错误：

错误场景	返回消息
使用 raise 抛出异常	{File "/var/user/index.py", line 2, in always_failed_handler raise Exception('xxx') Exception: xxx}
处理方法不存在	{'module' object has no attribute 'xxx'}
依赖模块并不存在	{global name 'xxx' is not defined}
超时	{"time out"}

日志

SCF 平台会将函数调用的所有记录及函数代码中的全部输出存储在日志中，请使用编程语言中的打印输出语句或日志语句生成输出日志，方便调试及故障排除时使用。详情见 日志管理 。

注意事项

由于 SCF 的特点，您须以**无状态**的风格编写您的函数代码。本地文件存储等函数生命周期内的状态特征，在函数调用结束后将随之销毁。
因此，持久状态建议存储在关系型数据库 TencentDB、对象存储 COS、云数据库 Memcached 或其他云存储服务中。

开发流程

了解更多云函数开发流程，请参见 使用流程 。

测试云函数

最近更新时间：2022-04-26 10:47:52

在创建并编写完云函数之后，您可以通过以下方式测试云函数，了解函数运行情况，并检查代码执行流程。

- SCF VS Code 插件：云端调试
- SCF 控制台：云端测试

测试事件及模板

云函数通过事件触发的方式运行，不同的触发器在触发函数时，传递的事件数据结构均有所不同。云函数的测试方法，即为通过发送模拟的测试事件，触发函数运行。
云函数控制台提供了如下事件模板模拟对应事件：

- **Hello World 事件模板**：简单数据结构及内容，可用于触发 hello world 模板所创建的函数。
- **COS 对象存储文件事件模板**：模拟 COS 对象存储的文件上传、删除事件。
- **CMQ Topic 事件模板**：模拟 CMQ 消息队列主题模式收到消息事件。
- **API Gateway 事件模板**：模拟 API 网关收到 API 请求事件。
- **Ckafka 事件模板**：模拟 Ckafka topic 收到消息事件。

通过控制台模板管理位置的**更换**操作，更换当前使用的测试模板，也可以更换为提供的测试事件模板或自定义的测试模板。关于事件模板的消息结构，详情请参见 触发器事件消息结构汇总 。

自定义模板配置及使用

在已提供的事件模板之外，我们还可以创建更多的自定义模板。通过控制台模板管理位置的**配置**操作，可以基于已有的模板，修改并保存为自定义模板，也可以直接输入自身设计的测试事件并保存为自定义模板。

注意事项

在使用测试事件模板时，您需注意以下几点：

- 测试事件模板的名称，目前仅支持英文、数字、-、_，且需要以英文字符开头。
- 已创建的自定义测试模板，如果不再使用，也可以通过配置界面删除。
- 针对同一个函数，目前自定义测试模板仅支持配置5个。如需配置新的测试模板，请先删除不再使用的旧测试模板。

环境变量

最近更新时间：2023-09-04 10:44:31

在创建或编辑云函数时，您可以通过修改配置中的环境变量，为云函数的运行环境增加、删除或修改环境变量。
在配置环境变量后，环境变量将在函数运行时配置到所在的操作系统环境中。函数代码可以使用读取系统环境变量的方式来获取到设置的具体值并在代码中使用。

新增环境变量

使用控制台新增环境变量

1. 登录 Serverless 控制台，单击左侧导航栏的函数服务。
2. 在创建函数的过程中，或针对已创建的函数进行编辑时，可在“环境变量”中，增加环境变量。
   环境变量通常以 `key-value` 对的形式出现，请在环境变量的输入框中，前一输入框输入所需的环境变量 key，后一输入框输入所需的环境变量 value。注意，key、value 的取值必须以字母起始，只能包含字母、数字及"_"，长度不小于 2 位且不大于 64 字节。

本地新增环境变量

1. 本地开发时，在 serverless.yml 文件中找到需要配置环境变量的函数，添加 Environment 配置项。如下所示：

```
component: scf # (必选) 组件名称，在该实例中为scf
name: scfdemo # (必选) 组件实例名称。

#组件参数配置
inputs:
  name: scfdemo # 云函数名称，默认为 ${name}-${stage}-${app}
  namespace: default
  # 1. 默认写法，新建特定命名的 cos bucket 并上传
  src: ./src
  type: event # 函数类型，默认为 event(事件类型)，web(web类型)
  handler: index.main_handler #入口（函数类型为事件类型时生效）
  runtime: Nodejs10.15 # 运行环境 默认 Nodejs10.15
  region: ap-guangzhou # 函数所在区域
  description: This is a function in ${app} application.
  memorySize: 128 # 内存大小，单位MB
  timeout: 20 # 函数执行超时时间，单位秒
  initTimeout: 3 # 初始化超时时间，单位秒
  environment: #  环境变量
    variables: #  环境变量对象
      TEST1: value1
      TEST2: value2
```

2. 保存 serverless.yml 文件，并在命令行窗口中执行 `scf deploy` 命令部署到云端。

查看环境变量

在配置好云函数的环境变量后，可通过查看云函数的函数配置，查询到具体已配置的环境变量，环境变量以 `key=value` 的形式显示。

版权所有：腾讯云计算（北京）有限责任公司

使用环境变量

已配置的环境变量，会在函数运行时配置到函数所在的运行环境中，可通过代码读取系统环境变量的方式来获取到具体值并在代码中使用。需要注意的是，**环境变量无法在本地进行读取**。
假设针对云函数，配置的环境变量的 key 为 `key`，以下为各运行环境读取并打印此环境变量值的示例代码。

- **在 Python 运行环境中**，读取环境变量的方法为：

```
import os
value = os.environ.get('key')
print(value)
```

- **在 Node.js 运行环境中**，读取环境变量的方法为：

```
var value = process.env.key
console.log(value)
```

- **在 Java 运行环境中**，读取环境变量的方法分为临时授权字段和其他字段两种情况：
  - 临时授权字段包括：`TENCENTCLOUD_SESSIONTOKEN`、`TENCENTCLOUD_SECRETID`、`TENCENTCLOUD_SECRETKEY`，读取环境变量的方法为：

```
System.out.println("value: "+ System.getProperty("key"));
```

  - 其他字段，读取环境变量的方法为：

```
System.out.println("value: "+ System.getenv("key"));
```

- **在 Golang 运行环境中**，读取环境变量的方法为：

```
import "os"
var value string
value = os.Getenv("key")
```

- **在 PHP 运行环境中**，读取环境变量的方法为：

```
$value = getenv('key');
```

使用场景

- **可变值提取**：针对业务中有可能会变动的值，提取至环境变量中，可避免需要根据业务变更而修改代码。
- **加密信息外置**：认证、加密相关的 key，从代码中提取至环境变量，可避免相关 key 硬编码在代码中而引起的安全风险。
- **环境区分**：针对不同开发阶段所要进行的配置和数据库信息，可提取到环境变量中。针对开发和发布的不同阶段，仅需要修改环境变量的值，分别执行开发环境数据库和发布环境数据库即可。

使用限制

针对云函数的环境变量，有如下使用限制：

- key 必须以字母 [a-zA-Z] 开头，只能包含字母数字字符和下划线（ [a-zA-Z0-9_]）。
- 预留的环境变量 key 无法配置。预留的 key 包括：
  - SCF_ 开头的 key，例如 SCF_RUNTIME。
  - QCLOUD_ 开头的 key，例如 QCLOUD_APPID。
  - TECENTCLOUD_ 开头的 key，例如 TENCENTCLOUD_SECRETID。

已内置环境变量

目前运行环境中已内置的环境变量的 Key 及 Value 见下表：

环境变量 Key	具体值或值来源
TENCENTCLOUD_SESSIONTOKEN	{临时 SESSION TOKEN}
TENCENTCLOUD_SECRETID	{临时 SECRET ID}
TENCENTCLOUD_SECRETKEY	{临时 SECRET KEY}
_SCF_SERVER_PORT	28902
TENCENTCLOUD_RUNENV	SCF
USER_CODE_ROOT	/var/user/
TRIGGER_SRC	timer（使用定时触发器时）
PYTHONDONTWRITEBYTECODE	x
PYTHONPATH	/var/user:/opt
CLASSPATH	/var/runtime/java x:/var/runtime/java x/lib/*:/opt（x 为 8 或 11）
NODE_PATH	/var/user:/var/user/node_modules:/var/lang/node x/lib/node_modules:/opt:/opt/node_modules（x 为 16、14、12、10、8 或 6）
PHP_INI_SCAN_DIR	/var/user/php_extension:/opt/php_extension
_	/var/lang/python3/bin/python x（x 为 37、3 或 2）
PWD	/var/user
LOGNAME	qcloud
LANG	en_US.UTF8
LC_ALL	en_US.UTF8
USER	qcloud（事件函数在 Node.js 16.13环境下有该内置变量，Web 函数则无该变量）

版权所有：腾讯云计算（北京）有限责任公司

HOME	/home/qcloud
PATH	/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SHELL	/bin/bash
SHLVL	3
LD_LIBRARY_PATH	/var/runtime/java x:/var/user:/opt（x 为 8 或 11）
HOSTNAME	{host id}
SCF_RUNTIME	函数运行时
SCF_FUNCTIONNAME	函数名
SCF_FUNCTIONVERSION	函数版本
TENCENTCLOUD_REGION	区域
TENCENTCLOUD_APPID	账号 APPID
TENCENTCLOUD_UIN	账号 UIN
TENCENTCLOUD_TZ	时区，当前为 UTC

依赖安装

最近更新时间：2023-03-20 17:41:07

内置依赖

云函数 SCF 各个运行时已内置部分常用依赖库，您可前往各运行时代码开发中查询：

- Node.js
- Python
- PHP
- Golang

安装依赖库

您可以将 SCF 代码所有的依赖库保存在代码包中，并上传至云端以供 SCF 使用。SCF 已支持的运行时及使用方法如下：

Node.js 运行时

Node.js 运行时支持以下三种依赖库安装方法：

依赖库同代码一起打包上传

通过依赖管理工具，例如 npm，在本地安装依赖后同函数代码一同打包上传。

> **注意**
> 打包时函数入口文件需要在 `zip` 包的根目录下。如果打包整个文件夹并上传 `zip` 包，则会因解压后无法在根目录找到入口文件而导致函数创建失败。

本文以安装 `lodash` 库为例：

1. 在本地终端中执行 `mkdir test-package` 命令，创建一个目录用于存放函数代码和依赖库。
2. 执行以下命令，在该目录下安装 `lodash` 依赖库。

```
cd test-package
npm install lodash
```

3. 在该目录下创建函数入口文件 `index.js` 并在代码中引用 `lodash` 库。

```
'use strict';
const _ = require('lodash');
exports.main_handler = async (event, context) => {
    console.log("Hello World")
    console.log(event)
    console.log(event["non-exist"])
    console.log(context)
    return event
};
```

版权所有：腾讯云计算（北京）有限责任公司

4. 将函数代码及依赖库一同压缩为 zip 包，在 Serverless 控制台 中上传打包的 zip 包并创建一个新函数。操作步骤如下：
   4.1 登录 Serverless 控制台，单击左侧导航栏的**函数服务**。
   4.2 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。
   4.3 在“新建函数”页面，填写函数基本信息。如下图所示：

   - **创建方式**：选择使用**从头开始**来新建函数。
   - **运行环境**：选择**Node.js12.16**。
   - **提交方法**：选择**本地上传zip包**。

   4.4 单击**完成**即可创建函数。

在线依赖安装

Node.js 运行时提供了在线依赖安装功能，可根据 `package.json` 中配置的依赖信息在线安装依赖包。详情请参见 在线依赖安装 。

使用依赖管理工具

云函数在线编辑器 Serverless Web IDE 提供了终端功能，并在终端中内置了包管理工具 npm 。

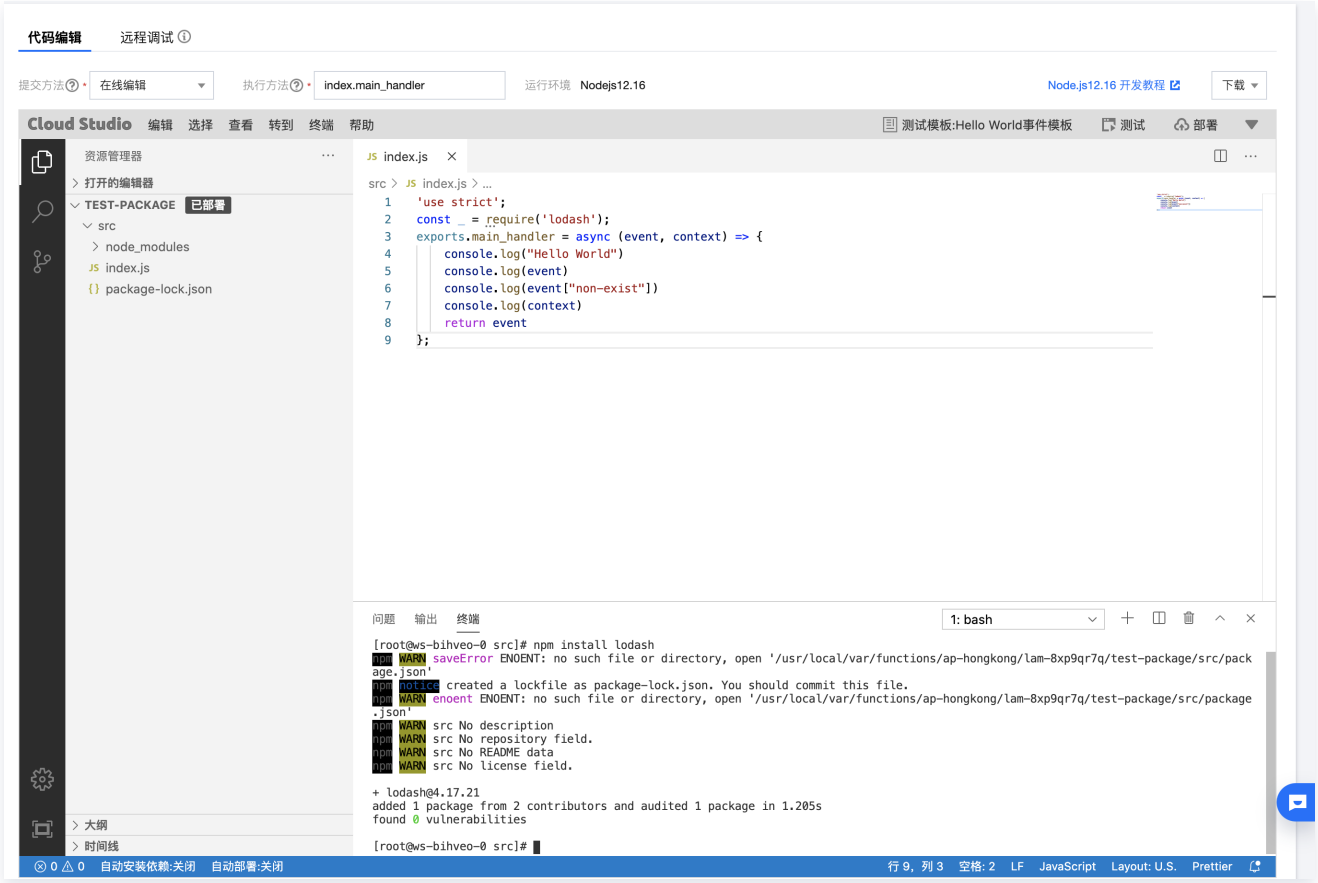
> **注意**
> Serverless Web IDE 对较新版本运行环境支持存在延时，如果对应运行环境下控制台未开放 Serverless Web IDE，请使用依赖库同代码一起打包上传或在线依赖安装方式进行依赖安装。

本文以在终端中安装 lodash 库为例：

1. 登录 Serverless 控制台，在左侧选择**函数服务**。
2. 在函数列表中，单击函数名，进入该函数的详情页面。
3. 在“函数管理”页面中，选择**函数代码** > **代码编辑**，查看并编辑函数。
4. 在 IDE 顶部的菜单栏**终端**中选择**新终端**，打开终端窗口。
5. 在终端中执行如下命令，安装依赖库 lodash ：

```
cd src  # 依赖库需要安装在与函数入口文件同一级的目录下，即需要进入`src`目录后再执行依赖安装操作。
npm install lodash # 可通过终端查看 npm 版本
```

6. 安装完成后，在 IDE 左侧的文件树中查看 package.json 和 node_modules 。
7. 单击**部署**后依赖库即可与函数代码一同打包上传到云端。如下图所示：



Python 运行时

Python 运行时支持以下两种依赖库安装方法：

依赖库同代码一起打包上传

通过依赖管理工具，例如 pip，在本地安装依赖后同函数代码一同打包上传。

> ⚠ 注意
> - 打包时函数入口文件需要在 zip 包的根目录下。如果打包整个文件夹并上传 zip 包，则会因解压后无法在根目录找到入口文件而导致函数创建失败。
> - 由于运行环境差异，请确认安装的依赖版本适配函数的运行环境。在线安装依赖时，建议使用 pip3 、pip36 等命令来指明 pip 等包管理工具的版本，详情请参见 WebIDE 常见命令 。
> - 函数运行系统为 CentOS 7，您需要在相同环境下进行安装。若环境不一致，则可能导致上传后运行时出现无法找到依赖的错误。
> - 若部分依赖涉及动态链接库，例如 Python 3.6 安装 pandas，需手动复制相关依赖包到依赖安装目录后再打包上传。详情请参见 使用 Docker 安装依赖 或使用在线 IDE 进行安装。

本文以安装 numpy 库为例：

1. 在本地终端中执行 mkdir test-package 命令，创建一个目录用于存放函数代码和依赖库。
2. 执行以下命令，在该目录下安装 numpy 依赖库。

```
cd test-package
pip install numpy -t . # 请关注所用 pip 版本是否适配函数运行环境，例如 python3.6 下请使用 pip36
```

3. 在该目录下创建函数入口文件 index.py 并在代码中引用 numpy 库。

```
# -*- coding: utf8 -*-
import json
import numpy
def main_handler(event, context):
  print("Received event: " + json.dumps(event, indent = 2))
  print("Received context: " + str(context))
  print("Hello world")
  return("Hello World")
```

4. 将函数代码及依赖库一同压缩为 zip 包，在 Serverless 控制台 中上传打包的 zip 包并创建一个新函数。操作步骤如下：
   4.1 登录 Serverless 控制台，单击左侧导航栏的**函数服务**。
   4.2 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。
   4.3 在“新建函数”页面，填写函数基本信息。如下图所示：

- **创建方式**：选择使用**从头开始**来新建函数。
- **运行环境**：选择**Python 3.6**。
- **提交方法**：选择**本地上传zip包**。

4.4 单击**完成**即可创建函数。

使用依赖管理工具

云函数在线编辑器 Serverless Web IDE 提供了终端功能，并在终端中内置了包管理工具 `pip` 。

> **注意**
> Serverless Web IDE 对较新版本运行环境支持存在延时，如果对应运行环境下控制台未开放 Serverless Web IDE，请使用依赖库同代码一起打包上传或在线依赖安装方式进行依赖安装。

本文以在终端中安装 `numpy` 库为例：

1. 登录 Serverless 控制台，在左侧选择**函数服务**。
2. 在函数列表中，单击函数名，进入该函数的详情页面。
3. 在“函数管理”页面中，选择**函数代码** > **代码编辑**，查看并编辑函数。
4. 在 IDE 顶部的菜单栏**终端**中选择**新终端**，打开终端窗口。
5. 在终端中执行如下命令，安装依赖库 `numpy`：

```
cd src  # 依赖库需要安装在与函数入口文件同一级的目录下，即需要进入`src`目录后再执行依赖安装操作。
pip install numpy -t . # 可通过终端查看 pip 版本，确认是否适配函数运行环境
```

6. 安装完成后，在 IDE 左侧的文件树中查看已安装的依赖库。
7. 单击**部署**后依赖库即可与函数代码一同打包上传到云端。

> **注意**
> - 您可以使用 `pip freeze > requirements.txt` 生成本地环境下所有依赖的 `requirements.txt` 文件。
> - 在 IDE 的终端中执行 `pip install -r requirements.txt -t .` 即可根据 `requirements.txt` 的配置安装依赖包。

PHP 运行时

> **注意**
> SCF 支持的 PHP 版本为 PHP 8.0、PHP 7.4、PHP 7.2 和 PHP 5.6，PHP 不同的小版本号存在不兼容的可能，请核对版本号后进行依赖安装。

安装自定义库

通过依赖管理工具，例如 composer，在本地安装依赖后同函数代码一同打包上传。

> **注意**
> 打包时函数入口文件需要在 `zip` 包的根目录下。如果打包整个文件夹并上传 `zip` 包，则会因解压后无法在根目录找到入口文件而导致函数创建失败。

本文以 PHP7.2 安装 `requests` 库为例：

1. 在本地终端中执行 `mkdir test-package` 命令，创建一个目录用于存放函数代码和依赖库。
2. 在 `test-package` 下创建 `Composer.json` 并指定需要安装的依赖库及版本。

```
{
"require": {
 "requests": ">=1.0"
 }
}
```

3. 执行以下命令，在该目录下安装 `requests` 依赖库。

```
cd test-package
composer install
```

4. 在该目录下创建函数入口文件 `index.php` 并在代码中引用 `requests` 库。

```
<?php
require 'vendor/autoload.php';
function main_handler($event, $context) {
 return "hello world";
}
?>
```

5. 将函数代码及依赖库一同压缩为 zip 包，在 Serverless 控制台 中上传打包的 zip 包并创建一个新函数。操作步骤如下：

5.1 登录 Serverless 控制台，单击左侧导航栏的**函数服务**。

5.2 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。

5.3 在“新建函数”页面，填写函数基本信息。如下图所示：

- **创建方式：选择使用从头开始来新建函数。**
- **运行环境：选择Php7.2。**
- **提交方法：选择本地上传zip包。**

5.4 单击**完成**即可创建函数。

安装自定义扩展

在函数入口文件的同级目录下创建扩展文件夹 `php_extension` 并添加自定义扩展文件 `.so` 和配置文件 `php.ini`，同函数代码一起打包上传。

本文以 `PHP7.2` 安装自定义扩展 `swoole.so` 为例。

1. 在本地终端中执行 `mkdir test-package` 命令，创建一个目录用于存放函数代码和依赖库。
2. 执行以下命令在 `test-package` 创建文件夹 `php_extension`，并将扩展对应的配置文件 `php.ini` 和扩展文件 `.so` 放在该目录下，目录结构如下：

> **说明**
> - 扩展文件夹 `php_extension` 和配置文件 `php.ini` 为固定命名，如使用其他命名可能导致扩展加载失败。
> - 扩展文件夹 `php_extension` 和配置文件 `php.ini` 以及自定义扩展 `.so` 文件需要具备可执行权限。

```
|____php_extension
| |____php.ini
| |____swoole.so
|____index.php
```

3. 自定义扩展支持从代码中或层中加载，如果扩展以层的形式上传，请确保上传到层的 zip 解压后的目录格式如下：

```
|____php_extension
| |____swoole.so
```

4. php.ini 写法：
   - 扩展在代码目录下：

```
extension=/var/user/php_extension/swoole.so
```

   - 扩展在层目录下：

```
extension=/opt/php_extension/swoole.so
```

5. 在该目录下创建函数入口文件 `index.php`，可通过 `extension_loaded()` 函数检查扩展是否加载成功，加载成功返回 `true`，否则返回 `false`。

```
<?php
function main_handler($event, $context) {
    var_dump(extension_loaded('swoole'));
   return "hello world";
}
?>
```

6. 将函数代码及依赖库一同压缩为 zip 包，在 Serverless 控制台 中上传打包的 zip 包并创建一个新函数。操作步骤如下：
   1. 登录 Serverless 控制台，单击左侧导航栏的**函数服务**。
   6.1 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。
   6.2 在“新建函数”页面，填写函数基本信息。如下图所示：

- **创建方式**：选择使用**从头开始**来新建函数。
- **运行环境**：选择**Php7.2**。
- **提交方法**：选择**本地上传zip包**。

6.3 单击**完成**即可创建函数。

Java 运行时

通过依赖管理工具，例如 maven，在本地安装依赖后同函数代码一同打包上传。

1. **在本地终端中执行** `mkdir test-package` **命令，创建一个目录用于存放函数代码和依赖库。**
2. **在该目录下创建** `pom.xml` **，并在** `pom.xml` **中配置依赖信息。**
3. **在项目文件夹根目录下执行** `mvn package` **命令，编译输出如下：**

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
```

```
[INFO] Building java-example 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.785 s
[INFO] Finished at: 2017-08-25T10:53:54+08:00
[INFO] Final Memory: 17M/214M
[INFO] ------------------------------------------------------------------------
```

4. 将函数代码及依赖库一同压缩为 jar 包，在 Serverless 控制台 中上传打包的 jar 包并创建一个新函数。操作步骤如下：

4.1 登录 Serverless 控制台，单击左侧导航栏的**函数服务**。

4.2 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。

4.3 在“新建函数”页面，填写函数基本信息。如下图所示：

- **创建方式：**选择使用**从头开始**来新建函数。
- **运行环境：**选择**Java8**。
- **提交方法：**选择**本地上传zip包**。

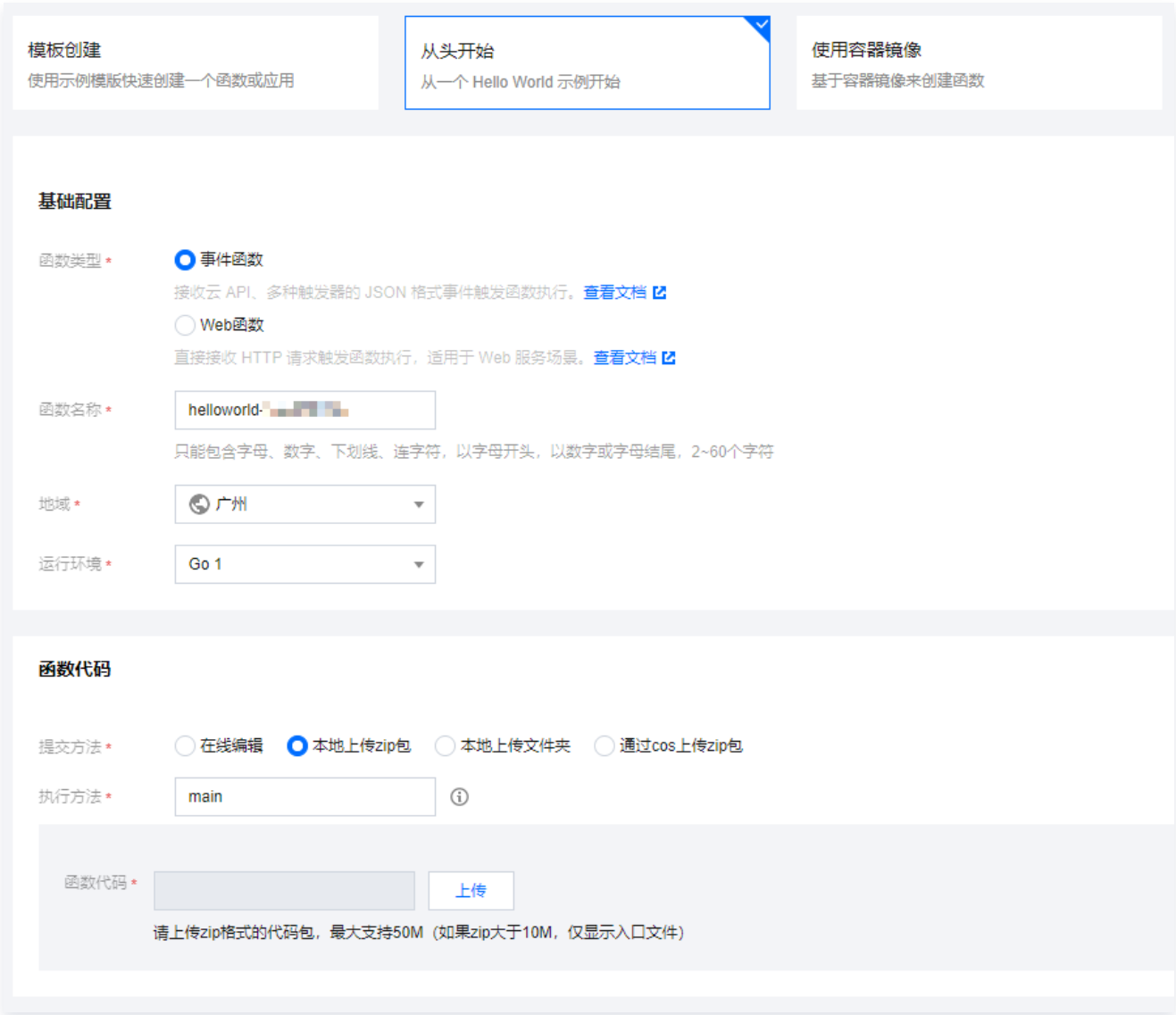
4.4 单击**完成**即可创建函数。

Go 运行时

使用方法：打包时上传最终的二进制文件。

Go 运行时的依赖库同代码一起编译后得到二进制文件，在 Serverless 控制台 中上传打包的二进制文件并创建一个新函数。操作步骤如下：

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. 在“新建函数”页面，填写函数基本信息。如下图所示：



- **创建方式：**选择使用**从头开始**来新建函数。
- **运行环境：**选择**Go1**。
- **提交方法：**选择**本地上传zip包**。

4. 单击**完成**即可创建函数。

错误类型与重试策略

最近更新时间：2022-12-20 16:22:50

在函数调用的过程中，可能有多种原因导致函数调用失败。不同的**错误类型**以及**调用方式（同步调用、异步调用）**都会影响重试策略。您可以配置 死信队列 收集错误事件信息、分析失败原因。

错误类型

在函数调用的过程中，可能有多种原因导致函数调用失败。错误类型分为以下几类：

调用错误

调用错误发生在函数实际执行前。以下情形均会产生调用错误：

- **调用请求错误**。例如传入的 Event 数据结构过大、入参不符合要求、函数不存在等。
- **调用方错误**。主要出现在调用方权限不足的情形。
- **超限错误**。调用的并发数超出 最大并发数 限制。

运行错误

运行错误发生在函数实际运行中。运行错误有以下情形：

- **用户代码运行错误**。这类错误出现在用户代码执行过程中，例如函数代码抛出异常，或者返回结果格式问题等。
- **Runtime 错误**。函数运行过程中，Runtime 负责拉起用户代码并执行。Runtime 错误指的是 Runtime 发现并上报的错误，例如函数运行超时（超时的时间限制请参见 限制说明 、代码语法报错等。

系统错误

函数平台的错误，例如 internal error。

重试策略

不同**错误类型**以及**调用方式（同步调用、异步调用）**都会影响重试策略。

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 、 CKafka 触发器 及 CLB 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。

> **注意**
> Ckafka 触发器会创建后台模块作为消费者，连接 CKafka 实例并消费消息。后台模块在获取到消息后，同步调用触发函数。由于 CKafka 触发器的后台模块是由云函数侧维护，即使是同步调用，其重试策略仍由云函数侧控制。
> - 对于运行错误（含用户代码错误和 Runtime 错误），CKafka 触发器会按照您配置的重试次数进行重试。默认重试10000次。
> - 对于超限错误和系统错误，CKafka 触发器会采用指数退避的方式持续重试，直至成功为止。

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 及 CMQ Topic 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 。

您可以根据业务诉求在函数配置中修改和自定义默认的**重试次数**，**最长等待时间**配置，该配置只适用于异步调用场景。

- **重试次数：**函数返回错误时云函数重试的次数，该参数只适用于**运行错误**的策略配置，默认配置为2次。
- **最长保留时间：**云函数在异步事件队列中保留事件的最长时间，该参数适用于所有异步调用的重试配置，默认配置为 6 小时，最大长度支持 10 万条。

异步调用发生各种错误类型的重试策略：

错误类型	重试策略
系统错误	函数请求执行状态码为 500。当发生该类错误时，函数平台会根据您配置的最长保留时间持续重试（默认持续重试6小时），重试间隔 1 分钟。如果您配置了死信队列，重试超过最长保留时间仍失败的事件会被发送到死信队列，您可进一步处理，否则事件将被函数平台丢弃。
超限错误	函数请求执行状态码为 432。当发生该类错误时，函数平台会根据您配置的最长保留时间持续重试（默认持续重试6小时），重试间隔 1 分钟。如果您配置了死信队列，重试超过最长保留时间仍失败的事件会被发送到死信队列，您可进一步处理，否则事件将被函数平台丢弃。
运行错误（除系统错误与超限错误外，其他错误均为运行错误）	当发生该类错误时，函数平台按照配置的重试次数进行重试，重试间隔 1 分钟。在自动重试的同时，新的触发事件仍可正常处理。如果您配置了死信队列，按照配置的重试次数重试后仍失败的事件或超出最长等待时间的事件将传入死信队列，否则事件将被函数平台丢弃。

注意：

1. 由于运行机制差异，重试及死信队列配置对于异步调用的 异步执行 函数在执行过程中发生错误的请求无效。
2. 如何判断是否超出最长等待时间？事件重试的时间-事件首次触发的时间大于最长等待时间即为超出最长等待时间。

死信队列

最近更新时间：2022-12-22 17:17:39

操作场景

死信队列 DLQ 是一个用户账号下的消息队列，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，以下情形的事件会被发送到死信队列：

- 用户代码运行错误重试2次依然失败。
- 超限错误和系统错误重试超过24小时。
- 异步队列 消息堆积达到上限。

> **说明**
> 死信队列功能目前处于内测阶段，如需使用请 提交工单 申请开通消息队列 CMQ。

死信队列消息属性

- **RequestId**：（字符串）事件调用请求的唯一标识
- **ErrorCode**：（数字）错误码状态
- **ErrorMessage**：（字符串）错误信息

死信队列投递至消息队列 CMQ 时会把属性信息与事件信息封装在一个 JSON 请求体中，格式如下：

```
{
 "RequestId": "b615b896-d197-47d7-8919-xxx",
 "ErrorCode": -1,
 "ErrorMessage": "Traceback (most recent call last):\n File \"/var/user/index.py\", line 5, in
main_handler\n if 'key1' in event.keys():\nNameError: global name 'event' is not defined",
 "Body": {
  "AppId": xxx,
  "Uin": "xxx",
  "SubAccountUin": "xxx",
  "RequestSource": "TRIGGER_TIMER",
  "FunctionName": "tabortest",
  "Namespace": "default",
  "Qualifier": "$DEFAULT",
  "InvocationType": "RequestResponse",
  "ClientContext": "{\"Type\":\"Timer\",\"TriggerName\":\"tabortimer\",\"Time\":\"2020-10-
10T01:22:00Z\",\"Message\":\"\"}",
  "LogType": "",
  "TimeStampForInvoker": "160229310xxx",
  "RequestId": "b615b896-d197-47d7-8919-xxx",
  "PushTime": "2020-10-10T09:22:00.061824591+08:00",
  "RetryNum": 2,
  "Ttl": 0
 }
}
```

结构名	内容
AppId	APPID。
Uin	主账号 ID。
SubAccountUin	创建者的子账号 ID（此字段可能返回 null，表示取不到有效值）。
RequestSource	触发器请求来源。
FunctionName	函数名称。
Namespace	命名空间。
Qualifier	触发函数的版本/别名。
ClientContext	运行函数时的参数，以 JSON 格式传入，最大支持的参数长度请参见 配额限制 。
TimeStampForInvoker	调用函数时的毫秒级时间戳。
RequestId	唯一请求 ID，每次请求都会返回。定位问题时需要提供该次请求的 RequestId。
PushTime	推送至消息队列的时间信息。
RetryNum	重试次数（0 - 2）。
Ttl	异步队列事件保留时长。

操作步骤

死信队列创建

> **说明**
> 云函数目前支持消息队列 CMQ 的主题模式和队列模式作为死信队列，您可自行选择配置。
> 函数别名的死信队列会跟随主版本的死信队列，主版本死信队列即在控制台创建别名时第一个选择并配置的死信队列。

1. 登录 CMQ 消息队列控制台，创建您的死信队列。
   CMQ 的主题模式支持标签过滤与路由匹配两种筛选方案，为了确保您的订阅者可以收到所有错误信息，请在添加订阅者时标签栏筛选设为**空**，BindingKey 筛选填**“#”**。
2. 登录 Serverless 控制台，创建您的函数。
3. 配置死信队列。
   您可以在“新建函数”页面或“函数配置”页面进行死信队列的配置。

死信队列监控

使用死信队列时，权限错误、资源误配或消息的总大小超过目标队列或主题的限制大小均会导致死信投递失败。您可在云函数监控信息中查询“死信队列投递失败次数/次”。

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查看死信队列监控的函数所在地域，并单击列表页中期望查看监控的函数，进入函数详情页面。
3. 在函数详情页中，单击**监控信息**即可查看死信队列投递失败次数。

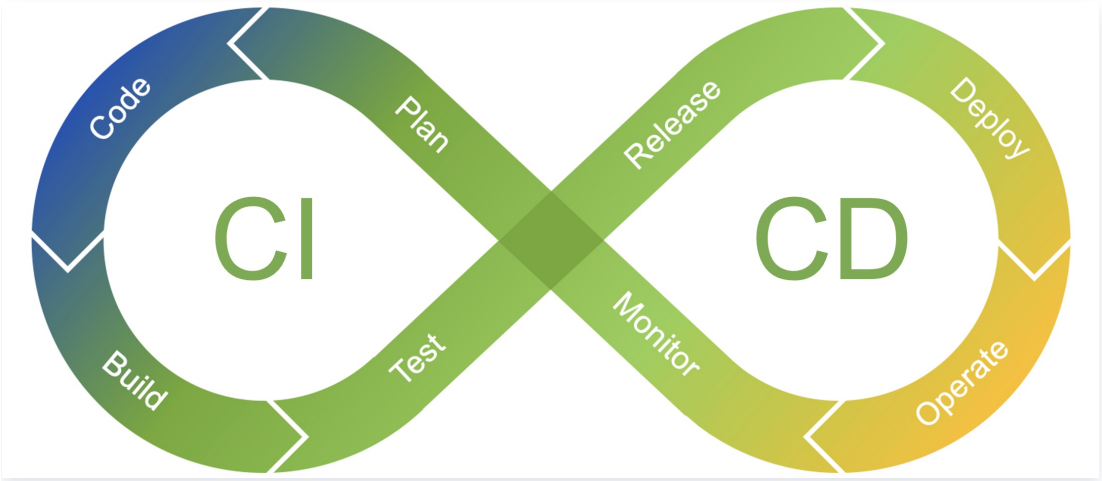
自动化部署

最近更新时间：2023-02-09 17:04:18

简介

随着敏捷和 DevOps 的流行，CI/CD 已经成了所有开发者在开发过程中必不可少的最佳实践，主要目标是以更快的速度、更短的周期向用户交付行之有效的软件。

CI/CD



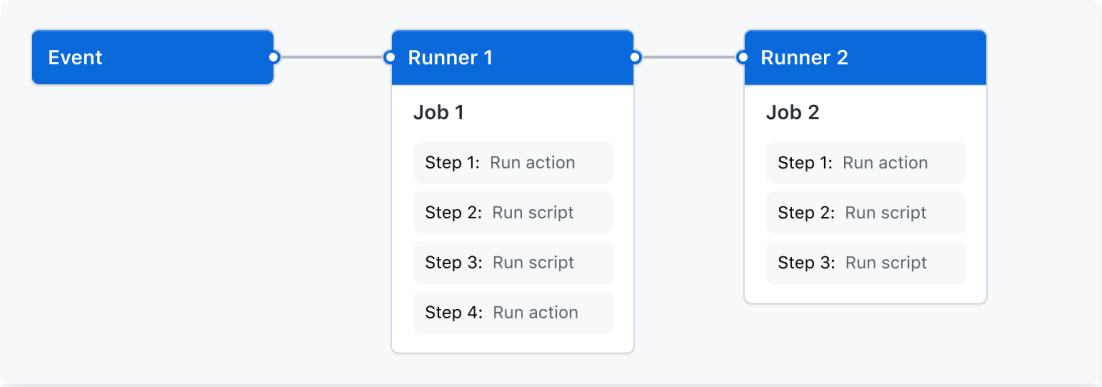
CI/CD 优势

- 缩短发布周期
- 降低风险
- 提高代码质量
- 更高效的反馈循环
- 可视化过程

本文以 GitHub、Jenkins、CODING 三个平台为例介绍如何结合 Serverless Cloud Framework 为 Serverless 项目快速搭建 CI/CD。

基于 GitHub 的自动化部署

GitHub Actions 是 GitHub 推出的自动化软件开发工作流。通过 Actions 可以执行任何任务，其中就包括 CI/CD。



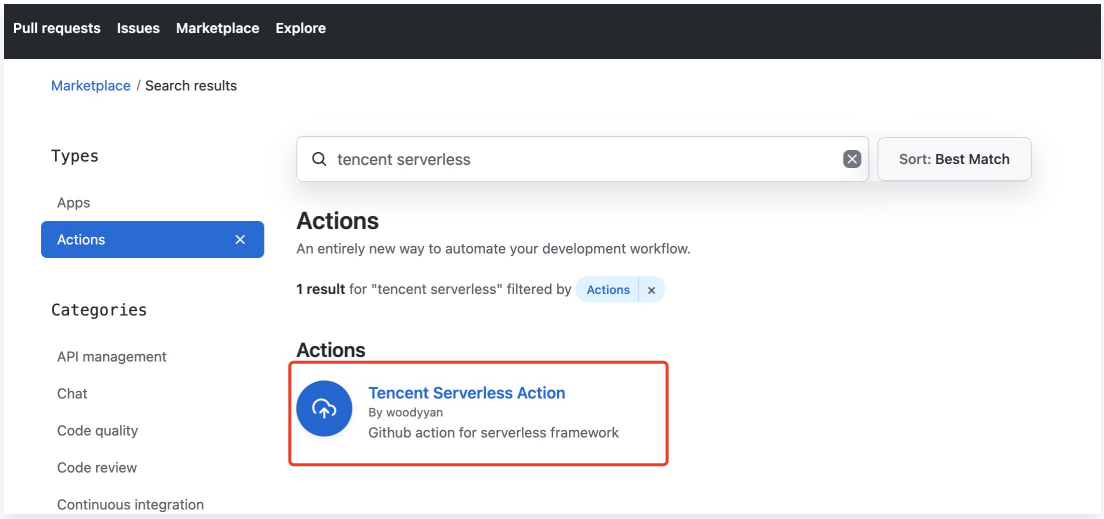
前提条件

- Serverless 项目已经托管到 GitHub。
- 项目中需要包含 Serverless Cloud Framework 执行需要用到的配置文件 `serverless.yml` 。
- 如果使用 Web 函数，请将 scf_bootstrap 文件放在项目根目录下。

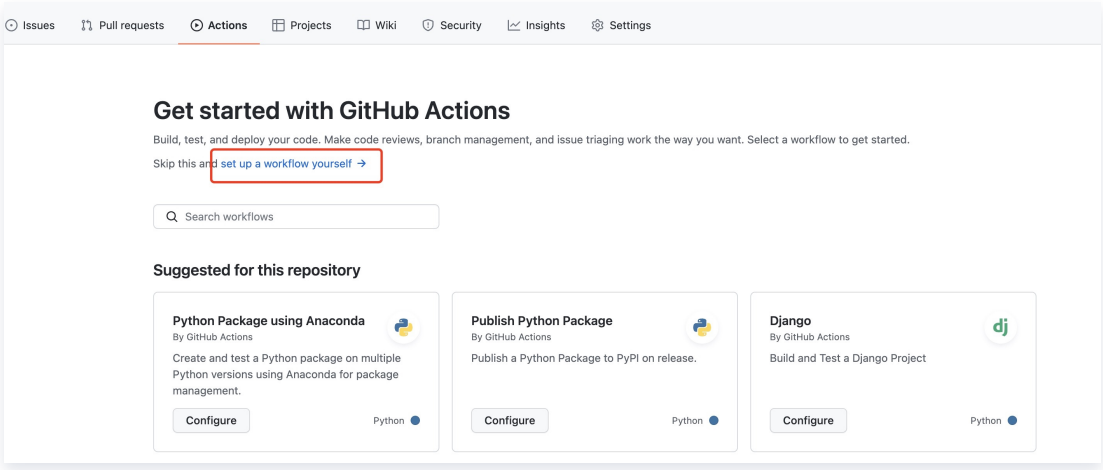
操作步骤

> **注意**
> SCF 已经在 GitHub 市场发布了 腾讯云 Serverless 部署的 Action 。

1. 在 GitHub 市场中搜索 “Tencent Serverless Action”。



2. 在 Actions 中选择 **Set up a workflow yourself**，如下图所示：



3. 使用方式：

- 如果熟悉 Action 用法，您可以使用下文的命令，其中已经封装了安装 Serverless Cloud Framework 和执行部署命令的步骤。

```
- name: serverless scf deploy
  uses: woodyyan/tencent-serverless-action@main
```

- 如果不熟悉 Action 用法，您可以根据不同的语言选择下列不同的 yml 写法，本文列举了Python、Java、NodeJS 三种语言的 yml 写法供参考。

Python

```
# 当代码推送到 main 分支时，执行当前工作流程
# 更多配置信息: https://docs.github.com/cn/actions/getting-started-with-github-actions
name: deploy serverless scf
on: #监听的事件和分支配置
  push:
    branches:
      - main
jobs:
  deploy:
    name: deploy serverless scf
    runs-on: ubuntu-latest
    steps:
      - name: clone local repository
        uses: actions/checkout@v2
      - name: deploy serverless
        uses: woodyyan/tencent-serverless-action@main
        env: # 环境变量
          STAGE: dev #您的部署环境
          SERVERLESS_PLATFORM_VENDOR: tencent #serverless 境外默认为 aws，配置为腾讯
          TENCENT_SECRET_ID: ${{ secrets.TENCENT_SECRET_ID }} #您的腾讯云账号 sercret ID，请在Settings-Secrets中配置
          TENCENT_SECRET_KEY: ${{ secrets.TENCENT_SECRET_KEY }} #您的腾讯云账号 sercret key，请在Settings-Secrets中配置
```

Java

```
name: deploy serverless scf
on: #监听的事件和分支配置
  push:
    branches:
      - main
jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      packages: write
    steps:
      - uses: actions/checkout@v2
      - name: Set up JDK 11
        uses: actions/setup-java@v2
        with:
          java-version: '11'
          distribution: 'temurin'
```

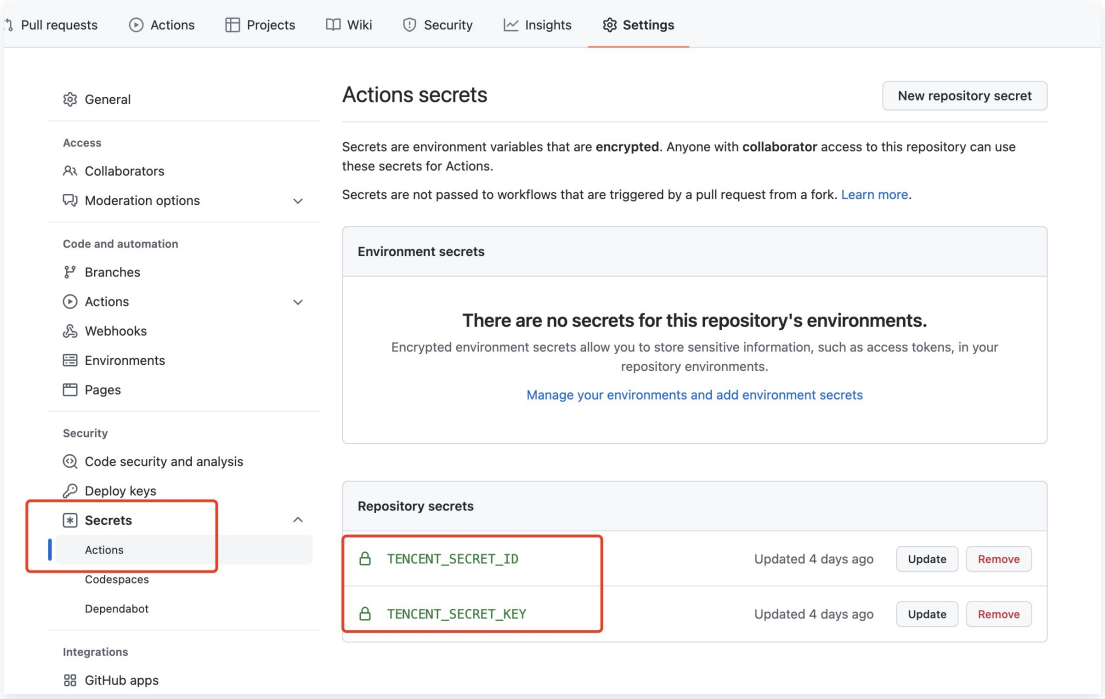
```
        server-id: github # Value of the distributionManagement/repository/id field of the pom.xml
        settings-path: ${{ github.workspace }} # location for the settings.xml file
      - name: Build with Gradle # Gradle项目用这个
        uses: gradle/gradle-build-action@937999e9cc2425eddc7fd62d1053baf041147db7
        with:
          arguments: build
      - name: Build with Maven # Maven项目用这个
        run: mvn -B package --file pom.xml
      - name: create zip folder # 此步骤仅用于Java Web函数，用于存放jar和scf_bootstrap文件。Java事件函数只需要在Serverless.yml中指定Jar目录就好。
        run: mkdir zip
      - name: move jar and scf_bootstrap to zip folder # 此步骤仅用于Java Web函数，用于移动jar和scf_bootstrap文件。Java事件函数只需要在Serverless.yml中指定Jar目录就好。注意如果是Maven编译请修改下面的jar路径为/target。
        run: cp ./build/libs/XXX.jar ./scf_bootstrap ./zip
      - name: deploy serverless
        uses: woodyyan/tencent-serverless-action@main
        env: # 环境变量
          STAGE: dev #您的部署环境
          SERVERLESS_PLATFORM_VENDOR: tencent #serverless 境外默认为 aws，配置为腾讯
          TENCENT_SECRET_ID: ${{ secrets.TENCENT_SECRET_ID }} #您的腾讯云账号 sercret ID
          TENCENT_SECRET_KEY: ${{ secrets.TENCENT_SECRET_KEY }} #您的腾讯云账号 sercret key
```

NodeJS

```
# 当代码推送到 main 分支时，执行当前工作流程
# 更多配置信息: https://docs.github.com/cn/actions/getting-started-with-github-actions
name: deploy serverless scf
on: #监听的事件和分支配置
  push:
    branches:
      - main
jobs:
  deploy:
    name: deploy serverless scf
    runs-on: ubuntu-latest
    steps:
      - name: clone local repository
        uses: actions/checkout@v2
      - name: install dependency
        run: npm install
      - name: build
        run: npm build
      - name: deploy serverless
        uses: woodyyan/tencent-serverless-action@main
        env: # 环境变量
          STAGE: dev #您的部署环境
          SERVERLESS_PLATFORM_VENDOR: tencent #serverless 境外默认为 aws，配置为腾讯
          TENCENT_SECRET_ID: ${{ secrets.TENCENT_SECRET_ID }} #您的腾讯云账号 sercret ID，请在Settings-Secrets中配置
```

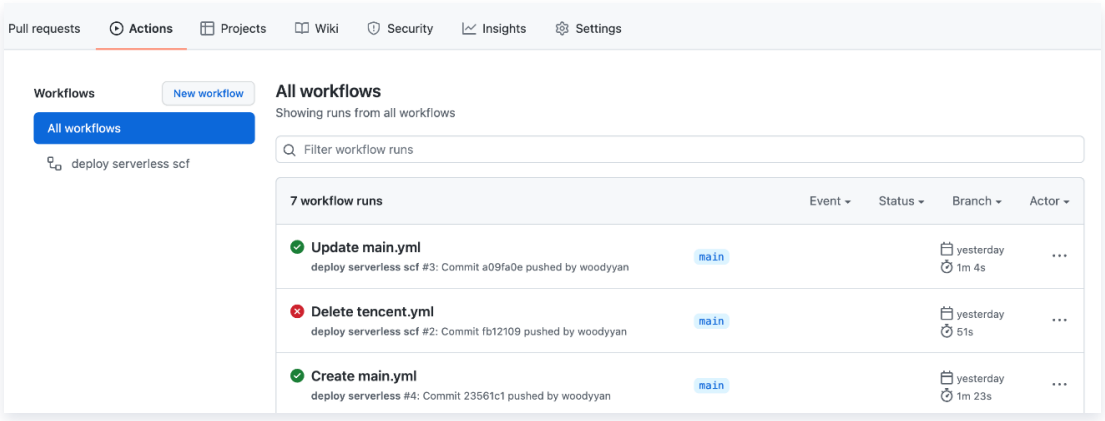
```
        TENCENT_SECRET_KEY: ${{ secrets.TENCENT_SECRET_KEY }} #您的腾讯云账号 sercret key，请在Settings-Secrets中配置
```

部署时需要用到腾讯云的 `TENCENT_SECRET_ID` 和 `TENCENT_SECRET_KEY`，您需要在 Github 代码仓库的设置中的 Secrets 里进行变量配置，如下图所示：



腾讯云 Secret ID 和Secret KEY 可在腾讯云的 访问管理 中获取。

4. 配置完成后，每次推送代码，都会自动触发部署流程，同时在 Actions 中可以实时看到执行结果与错误日志，如下图所示：



除此之外，您还可以根据项目需要，在流程中添加测试、安全检查、发布等步骤。

基于 Jenkinsfile 的自动化部署

Jenkinsfile 是通用于 Jenkins、CODING 等平台的，完成 Jenkinsfile 的配置，则能在这些平台上完成自动化部署。

前提条件

- 已将您的 Serverless 项目托管到 CODING/GitHub/GitLab/码云等平台。
- 项目中需要包含 Serverless Cloud Framework 执行需要用到的配置文件 `serverless.yml`。

- 如果使用 Web 函数，请将 scf_bootstrap 文件放在项目根目录下。

操作步骤

本文提供了 Python、Java、NodeJS 三种语言的 Jenkinsfile 写法供参考，请仔细阅读注释。

```
pipeline {
  agent any
  stages {
    stage('检出') {
      steps {
        checkout([$class: 'GitSCM', branches: [[name: env.GIT_BUILD_REF]],
            userRemoteConfigs: [[url: env.GIT_REPO_URL, credentialsId: env.CREDENTIALS_ID]]])
      }
    }
        stage('Package'){ // 此stage仅用于Java项目
       steps{
        container("maven") {
          echo 'Package start'
          sh "mvn package" // 此行用于Java Maven项目
                    sh "./gradlew build" // 此行用于Java Gradle项目
                    sh "mkdir zip" // 此行仅用于Java Web函数，用于存放jar和scf_bootstrap文件。Java事件函数只需要在Serverless.yml中指定Jar目录就好。
          sh "cp ./build/libs/XXX.jar ./scf_bootstrap ./zip" // 此行仅用于Java Web函数，用于移动jar和scf_bootstrap文件。Java事件函数只需要在Serverless.yml中指定Jar目录就好。注意如果是Maven编译请修改下面的jar路径为/target。
        }
          }
    }
    stage('安装依赖') {
      steps {
        echo '安装依赖中...'
        sh 'npm i -g scf'
        sh 'npm install' // 此行用于NodeJS项目
        echo '安装依赖完成.'
      }
    }
    stage('部署') {
      steps {
        echo '部署中...'
        withCredentials([
          cloudApi(
            credentialsId: "${env.TENCENT_CLOUD_API_CRED}",
            secretIdVariable: 'TENCENT_SECRET_ID',
            secretKeyVariable: 'TENCENT_SECRET_KEY'
          ),
        ]) {
            // 生成凭据文件
            sh 'echo
"TENCENT_SECRET_ID=${TENCENT_SECRET_ID}\nTENCENT_SECRET_KEY=${TENCENT_SECRET_KEY
}" > .env'
          // 部署
```

```
            sh 'scf deploy --debug'
            // 移除凭据
            sh 'rm .env'
        }
        echo '部署完成'
    }
  }
 }
}
```

使用上面的 Jenkinsfile 即可在 Jenkins、CODING 等平台一键完成 CI/CD 配置。

> **注意**
> 部署时需要用到腾讯云的 `TENCENT_SECRET_ID` 和 `TENCENT_SECRET_KEY`，可在 访问管理 中获取。

云函数状态码

最近更新时间：2023-09-11 21:26:32

对于函数运行后抛出的错误信息，您可以检索错误内容找到对应的问题产生原因和解决方案。

状态码及状态消息	说明	解决方法
200 Success	成功。	-
400 InvalidParameterValue	当运行事件函数传入的请求事件为非 json 类型会有该返回信息。	请参考 运行函数 API 文档 、同步调用 API 文档 修改后重试。
401 InvalidCredentials	权限认证失败。	您的账号没有操作该函数的权限，可参考 权限管理概述 对权限授予的说明修改后重试。
402 ServiceSuspended	服务临时冻结。	您的函数服务临时冻结导致服务停止，可参考 手动恢复服务 说明修改后重试。
403 Invalid scf response format	使用集成响应时，返回的数据结构没有被API网关成功解析。	API网关后端配置开启了集成响应时， 请按照 API 网关触发器的集成响应返回数据结构 修改后重试。
404 InvalidSubnetID	函数网络配置中子网 id 异常时，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 ContainerStateExited	容器退出。	请检查您的镜像或启动文件，是否可以本地正常启动。如本地可正常启动，请确定是否符合云函数 SCF 的使用限制，例如 RootFS 只读，仅允许 /tmp 可写。详情请参见 排查文档 。本地调试命令参考： `docker run -itd --read-only -v /tmp:/tmp`
406 RequestTooLarge	调用函数的入参 event ，即函数的请求事件大小超出 配额限制 。	请求事件大小配额限制：同步请求事件最大为6MB，异步请求事件最大为128KB。请对照配额调整请求事件大小后重试。
407 The size of response exceeds the	函数返回值超出6MB限制。	请调整函数返回值大小后重试。

upper limit (6MB)		
410 InsufficientBalance	账号余额不足。	由于您的腾讯云账户欠费导致服务停止，请充值后重试。
429 ResourceLimit	并发突增导致容器资源请求速度过高超出限制时，会有该返回信息。	每个账号弹性并发的扩容速度上限（函数 burst）默认为每个地域下500个/min，在并发突增时，如果没有足够的容器承载将会触发大量的容器请求动作，超出账号限制后会有该返回信息。 评估函数并发后为函数配置 预置并发 提前准备好容器，避免并发突增导致容器请求速度超限制。 如果经过评估后预置并发无法满足业务场景需要，可购买 函数套餐包 提升地域下函数burst。
430 User code exception caught	当用户代码执行出现错误时，会有该返回信息。	请根据云函数控制台提供的调用日志查看代码错误堆栈信息，检查并修改代码后重试。
432 ResourceLimitReached	当并发超出限制时，会有该返回信息。超出了账号地域下的配额超出了保留	对于配置了最大独占配额的函数，当函数的并发超出最大独占配额会返回 Function [xxx] concurrency exceeded reserved quota xxx MB，可评估业务需要后调整函数最大独占配额或查看 解决并发超限相关指引 文档。 对于未配置最大独占配额的函数，在函数实际使用的并发额度超出地域下剩余未占用并发额度后会返回 Function [xxx] concurrency exceeded region unreserved quota xxx MB，可评估业务需要后为函数配置最大独占配额，如地域剩余可用配额不足以满足业务需要，可购买 函数套餐包 提升地域下总并发配额。
433 TimeLimitReached	当函数在配置的 执行超时时间 范围内没有执行完成时，会有该返回信息。	检查业务代码是否有大量耗时处理操作。 在函数配置页调整执行超时时间，如果当前已是最大时间设置，可参考 异步执行 文档创建异步执行函数，可获得最长 24 小时函数执行时间。 该状态码会触发 实例回收 。
434 MemoryLimitReached	当函数运行中实际使用内存超过配置内存时，会有该返回信息。	检查代码逻辑，是否存在内存泄露。 在函数配置页面将内存配置调大，也可在函数内存配置页面申请大规格内存，可获得最大 120GB 函数执行内存。 该状态码会触发 实例回收 。
435 FunctionNotFound	当用户函数不存在时，会有该返回信息。	查看传入参数和期望调用的函数信息是否匹配。 查看函数在调用时是否存在，有无删除动作导致函数删除后再被调用。
436 InvalidParameterValue	参数不合法。invoke的传参不符合规范	参数不符合规范，请参考 API 文档 修改后重试。
437 HandlerNotFound	当函数包加载错误时，会有该返回信息。	请确认压缩包状态正常。 未找到函数执行入口文件，请确认入口文件在代码包解压后的根目录下。 请确认代码包中入口文件和 执行方法 。
438	函数状态异常或函数关停。	函数状态非正常时发起调用，请等待函数状态正常后重试。 由于您的腾讯云账户欠费导致服务停止，请充值后重试。

FunctionStatusError		
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后重试。 该状态码会触发 实例回收 。
441 UnauthorizedOperation	当函数执行时，用户 CAM 鉴权不通过，会有该返回信息。	需确认函数调用角色的 CAM 鉴权信息是否传参正确。可参考 权限管理概述 对权限授予的说明。
442 QualifierNotFound	当函数指定版本调用时，未找到对应版本，会有该返回信息。	确认传入指定版本信息是否正确，确认控制台是否配置别名版本信息正确。
443 UserCodeError	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。
444 PullImageFailed	拉取镜像失败。	请您确认所选择镜像的完整性和有效性后重试，如本地可正常下载。若仍无法解决，请联系 在线客服 或 提交工单。
445 ContainerInitError	容器启动失败。	容器启动失败，请检查您的启动文件是否已成功上传，并且保证调用路径正确。 镜像部署函数，请确认控制台传入的 Command 或者 Args 参数格式是否正确，详情可参见 使用镜像部署函数使用方法 。 代码部署函数，请检查您的启动文件是否已成功上传，并且保证调用路径正确。
446 PortBindingFailed	端口监听失败。	容器初始化超过 初始化超时时间 。 请检查您的监听端口是否为9000 请检查代码包或容器镜像中文件是否全部为必须文件，适当精简可提升容器初始化速度。 请检查初始化代码中是否有异常或者高耗时业务逻辑，可适当调大初始化超时时间后重试。
447 PullImageTimeOut	拉取镜像超时。	可能是由于镜像较大或网络抖动原因引起的超时，建议在最小化镜像或者调大 初始化超时时间 后重试。若仍无法解决，请联系 在线客服 或 提交工单 。
449 InsufficientResources	地域下没有该函数所选资源规格的可用资源。	若资源类型为 CPU 大规格或 GPU，可搭配预置使用。若仍无法解决，请 提交工单 。
450 InitContainerTimeout	容器启动超时情况下，会有该返回信息。	容器启动超出 初始化超时时间 ，请最小化代码或者调大初始化超时时间后重试。
452 NetworkSuspended	函数长时间无调用，网络处于冻结状态。	重新部署函数代码或更新函数配置。

499 RequestCanceled	函数执行请求取消后，会有该返回信息。	异步执行函数，用户执行中断请求动作后会有该返回信息。 Web 函数，API 网关触发器的超时时间小于函数的初始化时间和执行时间总和时，会有该返回信息，请检查代码中是否有异常耗时业务逻辑或调大 API 的后端超时时间后重试。
500 InternalError	内部错误。	内部错误，请稍后重试。若仍无法解决，请联系 在线客服 或 提交工单。

相关概念

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。如下图所示：

- 一段式格式为**文件名**，Golang 环境时使用。例如 `main` 。
- 两段式格式为**文件名.函数名**，Python、Node.js 及 PHP 环境时使用。例如 `index.main_handler` 。
  - 此执行方法**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**。需要确保代码包中的文件名后缀与语言环境匹配，如 Python 环境为 `.py` 文件，Node.js 环境为 `.js` 文件。更多执行方法相关说明，请参见 执行方法详情说明 。
- 三段式格式为**package.class::method**，JAVA 环境时使用。例如 `example.Hello::mainHandler` 。
- 非固定段式格式，只针对 Custom Runtime 运行环境开放使用，根据自定义语言实现来设定执行方法。

常见错误码解决方法

最近更新时间：2023-03-02 16:16:19

常见错误码解决方法如下表所示：

错误码	处理方法
InvalidParameter.FunctionName	FunctionName 取值与规范不符，请参见 API 文档 修正后重试。
InvalidParameterValue.Action	所请求的 API 不存在，请参见 API 文档 修正后重试。
InvalidParameterValue.CosBucketRegion	CosBucketRegion 取值与规范不符，请参见 COS 地域和访问域名 修正后重试。
InvalidParameterValue.DeadLetterConfig	DeadLetterConfig 取值与规范不符，Type 取值不在 CMQ-TOPIC、CMQ-QUEUE、topic、queue 范围内或 Name 为空会触发该错误信息，请修正后重试。
InvalidParameterValue.Enable	Enable 取值与规范不符，Enable 取值不在 OPEN 和 CLOSE 范围内会触发该错误信息，请修正后重试。
InvalidParameterValue.Memory	Memory 取值与规范不符，函数运行时内存大小默认为128MB，可选范64MB、128MB - 3072MB，以128MB为阶梯，请修正后重试。
InvalidParameterValue.OrderBy	OrderBy 取值与规范不符，请参考对应 API 支持取值范围修正后重试。
InvalidParameterValue.RoutingConfig	RoutingConfig 取值与规范不符，请参见 API 文档 。