

阿里云

函数计算 服务集成

文档版本：20211110

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。
2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。
3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。
4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。
5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。
6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
 危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	 危险 重置操作将丢失用户配置数据。
 警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	 警告 重启操作将导致业务中断，恢复业务时间约十分钟。
 注意	用于警示信息、补充说明等，是用户必须了解的内容。	 注意 权重设置为0，该服务器不会再接受新请求。
 说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	 说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击设置> 网络> 设置网络类型。
粗体	表示按键、菜单、页面名称等UI元素。	在结果确认页面，单击确定。
Courier字体	命令或代码。	执行 <code>cd /d C:/window</code> 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	<code>bae log list --instanceid</code> <code>Instance_ID</code>
[] 或者 [a b]	表示可选项，至多选择一个。	<code>ipconfig [-all -t]</code>
{ } 或者 {a b}	表示必选项，至多选择一个。	<code>switch {active stand}</code>

目录

1.服务列表	06
2.触发器简介	08
3.触发器列表	10
4.触发器管理	12
5.触发器event格式	14
6.OSS触发器	27
6.1. OSS触发器概述	27
6.2. 配置OSS触发器	29
7.HTTP触发器	36
7.1. HTTP触发器概述	36
7.2. 配置HTTP触发器	38
8.MNS主题触发器	44
8.1. MNS主题触发器概述	44
8.2. 配置MNS主题触发器	45
9.SLS触发器	51
9.1. SLS触发器概述	51
9.2. 配置SLS触发器	52
10.Tablestore触发器	59
10.1. Tablestore触发器概述	59
10.2. 配置Tablestore触发器	60
11.CDN事件触发器	66
11.1. CDN事件触发器概述	66
11.2. 配置CDN触发器	67
12.定时触发器	77
12.1. 定时触发器概述	77
12.2. 配置定时触发器	77

13.API网关触发器	83
13.1. API网关触发器概述	83
13.2. 配置API网关触发器	83
14.消息队列Kafka版	94
15.EventBridge触发器	107
15.1. EventBridge触发器简介	107
15.2. RocketMQ触发器	108
15.3. RabbitMQ触发器	113
15.4. MNS队列触发器	118
15.5. 官方事件源触发器	123
16.Serverless工作流	129
17.CDN源站	131

1.服务列表

本文提供与函数计算集成的服务列表。

- API服务
 - [API网关](#)
 - [CDN源站](#)
 - [HTTP服务](#)
- 存储
 - [对象存储服务OSS](#)
 - [日志服务SLS](#)
 - [表格存储Tablestore](#)
 - [NAS文件系统](#)
- 消息服务
 - [消息服务MNS主题模型](#)
 - [MNS队列触发器](#)
 - [路由到函数计算](#)
 - [RocketMQ触发器](#)
- Stream
 - [创建FC Sink Connector](#)
 - [DataHub](#)
- 函数编排
 - [Serverless工作流](#)
 - [运维编排OOS](#)
 - [资源编排ROS](#)
- IoT
 - [IoT物联网平台](#)
- 可观测集成
 - [函数日志（日志服务SLS）](#)
 - [云监控](#)
 - [链路追踪](#)
 - [应用实时监控服务ARMS](#)
- 安全
 - [操作审计ActionTrail](#)
- 其他事件
 - [定时服务](#)
 - [云监控](#)
 - [CDN事件](#)
 - [HTTPCDN](#)

- 函数输出方向集成
 - 事件总线EventBridge
 - 消息服务MNS

2. 触发器简介

触发器是触发函数执行的方式。在事件驱动的计算模型中，事件源是事件的生产者，函数是事件的处理者，而触发器提供了一种集中、统一的方式来管理不同的事件源。在事件源中，当事件发生时，如果满足触发器定义的规则，事件源会自动调用触发器关联的函数。

什么是触发器

函数计算提供了一种事件驱动的计算模型。函数的执行是由事件驱动的。函数的执行可以通过[函数计算控制台](#)或[SDK](#)触发，也可以由其他一些事件源来触发。您可以在指定函数中创建触发器，该触发器描述了一组规则，当某个事件满足这些规则，事件源就会触发关联的函数。

场景示例

- 示例一：对象存储OSS中的图片状态变更触发函数执行
某应用使用对象存储OSS存放上传的图片，您可以通过直接调用函数的方式去下载图片进行处理，并将结果存入OSS或者其他服务。如果OSS能够帮助我们关注新上传的图片，并且自动去调用关联的函数，您就不需要再去自己调用函数了，从而简化了开发和使用流程。OSS触发器的作用就是关注这些事件并调用函数计算的函数。配置了OSS触发器后，当有新图片上传，OSS触发器会自动触发函数下载并处理图片。
- 示例二：日志服务SLS中日志更新触发函数执行
某应用使用日志服务SLS定时采集更新的日志，您可以通过直接调用函数对增量的日志进行查询和分析。如果SLS能够帮助我们关注更新的日志，并自动调用关联的函数，您就不需要再去自己调用函数。SLS触发器的作用就是关注这些事件并调用函数计算的函数。配置了SLS触发器后，当有日志更新，SLS触发器会自动触发函数消费增量的日志。
- 示例三：在指定时间触发函数执行
某应用需要每隔1小时收集一次数据。您可以每隔1小时通过直接调用函数收集数据并处理。如果函数计算中的函数能每隔1小时自动执行，您就不需要再去关注时间。定时触发器的作用就是关注时间事件并调用函数计算的函数。配置了定时触发器后，在指定的时间，定时触发器会自动触发函数收集和处理数据。

触发器类型

触发器分为双向集成触发器、单向集成触发器和EventBridge触发器，三类触发器的区别如下：

- 双向集成触发器：您既可以在函数计算又可以在事件源端配置触发器。
- 单向集成触发器：您只需要在事件源端配置触发器，无需在函数计算配置。
- EventBridge触发器：您只需要在函数计算配置触发器，无需在事件源端配置。

从函数调用方式的角度，触发器又可以分为同步调用触发器和异步调用触发器，两种调用方式的区别如下所示。更多信息，请参见[函数调用概述](#)。

- 同步调用：事件被函数处理后直接返回结果。例如，使用控制台调用函数属于同步调用。
- 异步调用：事件在写入到函数计算内部队列后返回结果，函数计算系统会保证该消息被可靠地处理。

双向集成触发器

触发器名称	调用方式	文档链接
定时触发器	异步调用	定时触发器概述
OSS触发器	异步调用	OSS触发器概述
SLS触发器	同步调用	SLS触发器
CDN事件触发器	同步调用	CDN事件触发器概述

触发器名称	调用方式	文档链接
Tablestore触发器	同步调用	Tablestore触发器概述
MNS主题触发器	异步调用	MNS主题触发器
HTTP触发器	同步调用	HTTP触发器概述

单向集成触发器

触发器名称	调用方式	示例链接
API网关触发器	同步调用	API网关触发器
消息队列Kafka版Connector触发器	同步调用	创建FC Sink Connector

EventBridge触发器

触发器名称	调用方式	示例链接
消息队列RocketMQ版触发器	同步调用	RocketMQ触发器
消息队列RabbitMQ触发器	同步调用	RabbitMQ触发器
消息服务MNS队列触发器	同步调用	MNS队列触发器
阿里云官方事件源触发器	同步调用	官方事件源触发器

3. 触发器列表

函数计算支持的触发器分为双向集成触发器、单向集成触发器和EventBridge触发器。三类触发器的区别如下：

- 双向集成触发器：您既可以在函数计算又可以在事件源端配置触发器。
- 单向集成触发器：您只需要在事件源端配置触发器，无需在函数计算配置。
- EventBridge触发器：您只需要在函数计算配置触发器，无需在事件源端配置。

双向集成触发器

触发器名称	文档链接
定时触发器	定时触发器概述
OSS触发器	OSS触发器概述
SLS触发器	SLS触发器
CDN事件触发器	CDN事件触发器概述
Tablestore触发器	Tablestore触发器概述
MNS主题触发器	MNS主题触发器概述
HTTP触发器	HTTP触发器概述

单向集成触发器

触发器名称	示例链接
API网关触发器	API网关触发器
消息队列Kafka版Connector触发器	创建FC Sink Connector

EventBridge触发器

触发器名称	示例链接
消息队列RocketMQ版触发器	RocketMQ触发器
消息队列RabbitMQ触发器	RabbitMQ触发器
消息服务MNS队列触发器	MNS队列触发器
阿里云官方事件源触发器	官方事件源触发器

 **说明** 阿里云官方事件源触发器是指操作审计触发器、云监控触发器等阿里云官方事件源类型的触发器。关于更多支持的阿里云官方事件源类型触发器，请参见[阿里云官方事件源](#)。

触发器简易预览

图解函数计算&事件源服务

4. 触发器管理

本文介绍如何在函数计算控制台上创建、修改和删除触发器。

前提条件

- [创建服务](#)
- [创建函数](#)

创建触发器

- [创建OSS触发器](#)
- [创建HTTP触发器](#)
- [创建MNS主题触发器](#)
- [创建SLS触发器](#)
- [创建Tablestore触发器](#)
- [创建CDN触发器](#)
- [创建定时触发器](#)
- [创建RocketMQ触发器](#)
- [创建RabbitMQ触发器](#)
- [创建MNS队列触发器](#)
- [创建官方事件源触发器](#)

修改触发器

1. 登录[函数计算控制台](#)。
2. 在左侧导航栏，单击[服务及函数](#)。
3. 在顶部菜单栏，选择地域。
4. 找到目标服务下的目标函数，单击函数名称。

 **说明** 若目标服务存在多个版本，需要选择LATEST版本。

5. 在函数详情页面，单击[触发器管理](#)页签。在触发器列表中，找到目标触发器，然后在其右侧操作列单击[编辑](#)。
6. 在编辑触发器面板根据需要修改触发器的配置，然后单击[确定](#)。

 **说明** 创建的MNS主题触发器不支持修改。

删除触发器

1. 登录[函数计算控制台](#)。
2. 在左侧导航栏，单击[服务及函数](#)。
3. 在顶部菜单栏，选择地域。
4. 找到目标服务下的目标函数，单击函数名称。

 **说明** 若目标服务存在多个版本，需要选择LATEST版本。

5. 在函数详情页面，单击**触发器管理**页签。在触发器列表中，找到目标触发器，然后在其右侧**操作**列选择**更多 > 删除**（或单击**删除**）。
6. 在**确认**对话框，阅读提示信息，然后单击**删除**。

5.触发器event格式

不同的触发器，其触发的事件传递到函数接口的event格式不同，本文列举了不同触发器的event格式。

定时触发器

```
{
  "triggerTime":"2018-02-09T05:49:00Z",
  "triggerName":"timer-trigger",
  "payload":"awesome-fc"
}
```

参数	类型	示例值	描述
triggerTime	String	2018-02-09T05:49:00Z	函数被触发的时间。
triggerName	String	timer-trigger	定时触发器的名称。
payload	String	awesome-fc	您在触发器配置里输入的自定义参数，即触发消息的值。

OSS触发器

```
{
  "events": [
    {
      "eventName": "ObjectCreated:PutObject",
      "eventSource": "acs:oss",
      "eventTime": "2021-08-13T06:45:43.000Z",
      "eventVersion": "1.0",
      "oss": {
        "bucket": {
          "arn": "acs:oss:cn-shanghai:123456789:bucketname",
          "name": "testbucket",
          "ownerIdentity": "123456789"
        },
        "object": {
          "deltaSize": 122539,
          "eTag": "688A7BF4F233DC9C88A80BF985AB7329",
          "key": "image/a.jpg",
          "size": 122539
        },
        "ossSchemaVersion": "1.0",
        "ruleId": "9adac8e253828f4f7c0466d941fa3db81161****"
      },
      "region": "cn-shanghai",
      "requestParameters": {
        "sourceIPAddress": "140.205.XX.XX"
      },
      "responseElements": {
        "requestId": "58F9FF2D3DF792092E12044C"
      },
      "userIdentity": {
        "principalId": "123456789"
      }
    }
  ]
}
```

event中涉及的参数解释如下表所示。

参数	类型	示例值	描述
eventName	String	ObjectCreated:PutObject	事件类型。
eventSource	String	acs:oss	事件源，固定为“acs:oss”。
eventTime	String	2021-08-13T06:45:43.000Z	事件产生的时间。使用ISO-8601标准时间格式。
eventVersion	String	1.0	事件协议的版本。
oss	Map		OSS事件内容。

参数	类型	示例值	描述
bucket	Map		bucket参数内容。
name	String	testbucket	Bucket的名称。
arn	String	acs:oss:cn-shanghai:123456789:bucketname	Bucket的唯一标识符。
ownerIdentity	String	123456789	创建Bucket的用户ID。
object	Map		object参数内容。
size	Int	122539	object的大小。单位：Byte。
deltaSize	Int	122539	object的大小变化量。单位：Byte。 - 如果新增一个文件，此参数的值表示文件大小。 - 如果同名覆盖一个文件，此参数的值表示新文件与旧文件的大小差值。
eTag	String	688A7BF4F233DC9C88A80BF985AB7329	Object的标签。
key	String	image/a.jpg	Object的名称。
ossSchemaVersion	String	1.0	OSS模式的版本号。
ruleId	String	9adac8e253828f4f7c0466d941fa3db81161****	事件匹配的规则ID。
region	String	cn-shanghai	Bucket所在的地域。
requestParameters	Map		请求参数。
sourceIPAddress	String	140.205.XX.XX	请求的源IP地址。
responseElements	Map		响应元素。
requestId	String	58F9FF2D3DF792092E12044C	请求对应的Request ID。
userIdentity	Map		用户属性。
principalId	String	123456789	请求发起者的阿里云账号ID。

SLS触发器

event是函数计算的入口参数。具体格式如下：

```
{
  "parameter": {},
  "source": {
    "endpoint": "http://cn-shanghai-intranet.log.aliyuncs.com",
    "projectName": "log-com",
    "logstoreName": "log-en",
    "shardId": 0,
    "beginCursor": "MTUyOTQ4MDIwOTY1NTk3ODQ2Mw==",
    "endCursor": "MTUyOTQ4MDIwOTY1NTk3ODQ2NA=="
  },
  "jobName": "1f7043ced683de1a4e3d8d70b5a412843d81****",
  "taskId": "c2691505-38da-4d1b-998a-f1d4bb8c****",
  "cursorTime": 1529486425
}
```

参数	描述	本文示例
parameter	您配置触发器时填写的调用参数的值。	无。
source	设置函数读取的日志块信息。 • endpoint：日志服务Project所属的阿里云地域。 • projectName：日志服务Project名称。 • logstoreName：Logstore名称。 • shardId：Logstore中一个确定的Shard。 • beginCursor：开始消费数据的位置。 • endCursor：停止消费数据的位置。	<pre>{ "endpoint": "http://cn-shanghai-intranet.log.aliyuncs.com", "projectName": "log-com", "logstoreName": "log-en", "shardId": 0, "beginCursor": "MTUyOTQ4MDIwOTY1NTk3ODQ2Mw==", "endCursor": "MTUyOTQ4MDIwOTY1NTk3ODQ2NA==" }</pre>

参数	描述	本文示例
jobName	日志服务ETL Job名字，函数配置的SLS触发器对应一个日志服务的ETL Job。	1f7043ced683de1a4e3d8d70b5a412843d81****
taskId	对于ETL Job而言，taskId是一个确定性的函数调用标识。	c2691505-38da-4d1b-998a-f1d4bb8c****
cursorTime	最后一条日志到达日志服务端的unix_timestamp。	1529486425

CDN触发器

```
{ "events": [
  {
    "eventName": "CdnDomainStarted",
    "eventVersion": "1.0.0",
    "eventSource": "cdn",
    "region": "cn-hangzhou",
    "eventTime": "2018-03-16T14:19:55+08:00",
    "traceld": "cf89e5a8-7d59-4bb5-a33e-4c3d08e25acf",
    "resource": {
      "domain": "example.com"
    },
    "eventParameter": {
      "domain": "example.com"
    },
    "userIdentity": {
      "aliUid": "164901546557****"
    }
  }
]
```

event中涉及的参数解释如下表所示。

参数	类型	示例值	描述
eventName	String	CachedObjectsRefreshed	事件类型。
eventSource	String	cdn	事件源名称。
region	String	cn-hangzhou	CDN所在地域，默认为cn-hangzhou。
eventVersion	String	1.0.0	事件触发版本。
eventTime	String	2018-06-14T15:31:49+08:00	事件发生时间。
traceld	String	c6459282-6a4d-4413-894c-e4ea39686738	事件源传递的ID，用于排查问题。

参数	类型	示例值	描述
resource	String		资源信息。
domain	String	example.com	域名。
eventParameter	String		事件参数。
domain	String	example.com	域名。
userIdentity	String		用户属性。
aliUid	String	164901546557****	阿里云账号ID。

Tablestore触发器

```
{
  "Version": "Sync-v1",
  "Records": [
    {
      "Type": "PutRow",
      "Info": {
        "Timestamp": 1506416585740836
      },
      "PrimaryKey": [
        {
          "ColumnName": "pk_0",
          "Value": 1506416585881590900
        },
        {
          "ColumnName": "pk_1",
          "Value": "2017-09-26 17:03:05.8815909 +0800 CST"
        },
        {
          "ColumnName": "pk_2",
          "Value": 1506416585741000
        }
      ],
      "Columns": [
        {
          "Type": "Put",
          "ColumnName": "attr_0",
          "Value": "hello_table_store",
          "Timestamp": 1506416585741
        },
        {
          "Type": "Put",
          "ColumnName": "attr_1",
          "Value": 1506416585881590900,
          "Timestamp": 1506416585741
        }
      ]
    }
  ]
}
```

event中涉及的参数解释如下表所示。

参数	类型	示例值	描述
Version	String	Sync-v1	Payload版本号。
Records	Array		数据表中的增量数据行数组。

参数	类型	示例值	描述
Type	String	PutRow	数据行类型，取值说明如下： - PutRow：插入一行数据。 - UpdateRow：更新指定行数据。 - DeleteRow：删除指定行。
Info	String		数据行基本信息。
Timestamp	Int64	1506416585740836	该数据行的最后修改UTC时间。
PrimaryKey	Array		主键列数组。
ColumnName	String	pk_0	主键列名称。
Value	String、Integer或Blob	1506416585881590900	主键列内容。
Columns	Array		属性列数组。
Type	String	Put	属性列类型，取值说明如下： - Put：增加一个版本。 - DeleteOneVersion：删除一个版本。 - DeleteAllVersions：删除所有版本。
ColumnName	String	attr_0	属性列名称。
Value	Integer、Boolean、Double、String或Blob	hello_table_store	属性列内容。
Timestamp	Int64	1506416585741	属性列最后修改UTC时间。

MNS主题触发器

- 创建触发器时，若event格式设置为STREAM。
 - 当消息中不含消息属性（MessageAttributes）时，event格式如下。

 说明 当消息中不含消息属性（MessageAttributes）时，event的内容格式为JSON字符串。

消息正文。
'hello topic'

- 当消息中含有消息属性（MessageAttributes）时，event格式如下。

 说明 event的内容中包含MessageAttributes相关的键值对。更多信息，请参见[PublishMessage](#)。

```
{
  "body": "hello topic",
  "attrs": {
    "Extend": "{\\"key\\":\\"value\\"}"
  }
}
```

- 创建触发器时，若event格式设置为JSON。
 - 当消息中不含消息属性（MessageAttributes）时，event格式如下。

```
{
  "TopicOwner": "118620210433****",
  "Message": "hello topic",
  "Subscriber": "118620210433****",
  "PublishTime": 1550216480040,
  "SubscriptionName": "test-fc-subscribe",
  "MessageMD5": "BA4BA9B48AC81F0F9C66F6C909C3****",
  "TopicName": "Mytopic",
  "MessageId": "2F5B3C082B923D4EAC694B76D928****"
}
```

- 当消息中含有消息属性（MessageAttributes）时，event格式如下。

 说明 event的内容中包含MessageAttributes相关的键值对。更多信息，请参见[PublishMessage](#)。

```
{
  "key": "value",
  "TopicOwner": "118620210433****",
  "Message": "hello topic",
  "Subscriber": "118620210433****",
  "PublishTime": 1550216302888,
  "SubscriptionName": "test-fc-subscribe",
  "MessageMD5": "BA4BA9B48AC81F0F9C66F6C909C3****",
  "TopicName": "Mytopic",
  "MessageId": "2F5B3C281B283D4EAC694B742528****"
}
```

event中涉及的参数解释如下表所示。

参数	类型	示例值	描述
key	String	value	消息属性相关的键值对。

参数	类型	示例值	描述
TopicOwner	String	118620210433****	订阅Topic的AccountId。
Message	String	hello topic	消息内容。
Subscriber	String	118620210433****	用户的AccountId。
PublishTime	Map	1550216302888	消息发布时间。
SubscriptionName	String	test-fc-subscribe	订阅的名称。
MessageMD5	String	BA4BA9B48AC81F0F9C66F6C909C3****	消息正文的MD5值。
TopicName	String	Mytopic	Topic名称。
MessageId	String	2F5B3C281B283D4EAC694B742528****	消息的编号。

RocketMQ触发器

```
{
  "id": "94ebc15f-f0db-4bbe-acce-56fb72fb****",
  "source": "RocketMQ-Function-rocketmq-trigger",
  "specversion": "1.0",
  "type": "mq:Topic:SendMessage",
  "datacontenttype": "application/json; charset=utf-8",
  "subject": "acs:mq:cn-hangzhou:164901546557****:MQ_INST_164901546557****_BXhFHryi%TopicName",
  "time": "2021-04-08T06:01:20.766Z",
  "aliyunaccountid": "164901546557****",
  "aliyunpublishtime": "2021-10-15T02:05:16.791Z",
  "aliyunoriginalaccountid": "164901546557****",
  "aliyuneventbusname": "RocketMQ-Function-rocketmq-trigger",
  "aliyunregionid": "cn-chengdu",
  "aliyunpublishaddr": "42.120.XX.XX",
  "data": {
    "topic": "TopicName",
    "systemProperties": {
      "MIN_OFFSET": "0",
      "TRACE_ON": "true",
      "MAX_OFFSET": "8",
      "MSG_REGION": "cn-hangzhou",
      "KEYS": "systemProperties.KEYS",
      "CONSUME_START_TIME": 1628577790396,
      "TAGS": "systemProperties.TAGS",
      "INSTANCE_ID": "MQ_INST_164901546557****_BXhFHryi"
    },
    "userProperties": {
    },
    "body": "TEST"
  }
}
```

data字段包含的参数解释如下表所示。关于CloudEvents规范中定义的参数解释，请参见[事件概述](#)。

参数	类型	示例值	描述
topic	String	TopicName	Topic名称。
systemProperties	Map		系统属性。
MIN_OFFSET	Int	0	最低位点。
TRACE_ON	Boolean	true	是否有消息轨迹。取值说明如下： <i>true</i>：有消息轨迹。 <i>false</i>：无消息轨迹。
MAX_OFFSET	Int	8	最高位点。
MSG_REGION	String	cn-hangzhou	发送消息的地域。
KEYS	String	systemProperties.KEYS	过滤属性。
CONSUME_START_TIME	Long	1628577790396	开始消费时间。单位：毫秒。
UNIQ_KEY	String	AC14C305069E1B28CDFA3181CDA2****	消息唯一键。
TAGS	String	systemProperties.TAGS	过滤属性。
INSTANCE_ID	String	MQ_INST_123456789098****_BXhFHryi	实例ID。
userProperties	Map	无	用户属性。
body	String	TEST	消息内容。

RabbitMQ触发器

```
{
  "id": "bj694332-4cj1-389e-9d8c-b137h30b****",
  "source": "RabbitMQ-Function-rabbitmq-trigger",
  "specversion": "1.0",
  "type": "amqp:Queue:SendMessage",
  "datacontenttype": "application/json;charset=utf-8",
  "subject": "acs:amqp:cn-hangzhou:164901546557****:/instances/amqp-cn-tl32e756****/vhosts/eb-connect/queues/housekeeping",
  "time": "2021-08-12T06:56:40.709Z",
  "aliyunaccountid": "164901546557****",
  "aliyunpublishtime": "2021-10-15T08:58:55.140Z",
  "aliyunoriginalaccountid": "164901546557****",
  "aliyuneventbusname": "RabbitMQ-Function-rabbitmq-trigger",
  "aliyunregionid": "cn-chengdu",
  "aliyunpublishaddr": "42.120.XX.XX",
  "data": {
    "envelope": {
      "deliveryTag": 98,
      "exchange": "",
      "redeliver": false,
      "routingKey": "housekeeping"
    },
    "body": {
      "Hello": "RabbitMQ"
    },
    "props": {
      "contentEncoding": "UTF-8",
      "messageId": "f7622d51-e198-41de-a072-77c1ead7****"
    }
  }
}
```

data字段包含的参数解释如下表所示。关于CloudEvents规范中定义的参数解释，请参见[事件概述](#)。

参数	类型	示例值	描述
body	Map		消息内容。
Hello	String	Event Bridge	用户数据。
props	Map		消息属性。
contentEncoding	String	utf-8	消息内容编码。
messageId	String	f7622d51-e198-41de-a072-77c1ead7****	消息ID。每条消息的ID取值唯一。
envelope	Map		消息的envelope信息。
deliveryTag	Int	98	消息的Tag。
exchange	String	无	消息的Exchange。

参数	类型	示例值	描述
redeliver	Boolean	false	是否支持重发消息。取值说明如下： <i>true</i>：支持 <i>false</i>：不支持
routingKey	String	housekeeping	消息的路由规则。

MNS队列触发器

```
{
  "id": "c2g71017-6f65-fhcf-a814-a396fc8d****",
  "source": "MNS-Function-mnstrigger",
  "specversion": "1.0",
  "type": "mns:Queue:SendMessage",
  "datacontenttype": "application/json; charset=utf-8",
  "subject": "acs:mns:cn-hangzhou:164901546557****:queues/zeus",
  "time": "2021-04-08T06:28:17.093Z",
  "aliyunaccountid": "1649015465574023",
  "aliyunpublishtime": "2021-10-15T07:06:34.028Z",
  "aliyunoriginalaccountid": "164901546557****",
  "aliyuneventbusname": "MNS-Function-mnstrigger",
  "aliyunregionid": "cn-chengdu",
  "aliyunpublishaddr": "42.120.XX.XX",
  "data": {
    "requestId": "606EA3074344430D4C81****",
    "messageId": "C6DB60D1574661357FA227277445****",
    "messageBody": "TEST"
  }
}
```

data字段包含的参数解释如下表所示。关于CloudEvents规范中定义的参数解释，请参见[事件概述](#)。

参数	类型	示例值	描述
requestId	String	606EA3074344430D4C81****	请求ID。每个请求的ID取值唯一。
messageId	String	C6DB60D1574661357FA227277445****	消息ID。每条消息的ID取值唯一。
messageBody	String	TEST	消息内容。

HTTP触发器

HTTP触发器有别于其他触发器，函数签名是请求（Request）和响应（Response）对象，而不是事件（event）对象。所以HTTP触发器没有事件格式。具体信息，请参见[配置HTTP触发器](#)。

6.OSS触发器

6.1. OSS触发器概述

对象存储服务OSS（Object Storage Service）是阿里云提供的海量、安全、低成本、高可靠的云存储服务。OSS与函数计算集成后，OSS事件能触发相关函数执行，实现对OSS中的数据进行自定义处理。

背景信息

OSS和函数计算通过OSS事件触发器实现无缝集成，您可以编写函数对OSS事件进行自定义处理，当OSS系统捕获到指定类型的事件后，OSS事件触发相应的函数执行。例如，您可以设置函数来处理Put Object事件，当您调用OSS的Put Object接口上传图片到OSS后，相关联的函数会自动被触发来处理该图片。

OSS和函数计算集成后，您可以自由的调用各种函数处理图像或音频数据，再把结果写回到多种存储服务中。整个架构中，您只需要专注于函数逻辑的编写，系统将以实时的、可靠的、大规模并行的方式处理海量的数据。

OSS事件定义

当OSS系统捕获到相关事件后，会将事件信息编码为JSON字符串，传递给事件处理函数。OSS事件通知格式の詳細信息，请参见消息通知。已支持的OSS事件定义如下表所示。

事件名称	说明
oss:ObjectCreated:PutObject	调用PutObject接口上传文件，更多信息，请参见PutObject。API执行成功后触发函数。
oss:ObjectCreated:PutSymlink	调用PutSymlink接口针对OSS上的Target Object创建软链接，您可以通过该软链接访问Target Object，更多信息，请参见PutSymlink。API执行成功后触发函数。
oss:ObjectCreated:PostObject	调用PostObject接口使用HTML表单上传文件到指定的Bucket，更多信息，请参见PostObject。API执行成功后触发函数。
oss:ObjectCreated:CopyObject	调用CopyObject接口拷贝一个在OSS上已经存在的对象，更多信息，请参见CopyObject。API执行成功后触发函数。
oss:ObjectCreated:InitiateMultipartUpload	使用Multipart Upload模式传输数据前，必须先调用InitiateMultipartUpload接口来通知OSS初始化一个Multipart Upload事件，更多信息，请参见InitiateMultipartUpload。API执行成功后触发函数。
oss:ObjectCreated:UploadPart	初始化一个Multipart Upload之后，可以根据指定的对象名和Upload ID来分块（Part）上传数据，更多信息，请参见UploadPart。API执行成功后触发函数。
oss:ObjectCreated:UploadPartCopy	UploadPartCopy通过从一个已存在的Object中拷贝数据来上传一个Part，更多信息，请参见UploadPartCopy。API执行成功后触发函数。
oss:ObjectCreated:CompleteMultipartUpload	在将所有数据Part都上传完成后，必须调用CompleteMultipartUpload接口来完成整个文件的Multipart Upload。更多信息，请参见CompleteMultipartUpload。API执行成功后触发函数。

事件名称	说明
oss:ObjectCreated:AppendObject	调用AppendObject接口以追加写的方式上传文件，更多信息，请参见AppendObject。API执行成功后触发函数。
oss:ObjectCreated:*	任何上述ObjectCreated类型的API执行成功后都会触发函数。
oss:ObjectRemoved:DeleteObject	调用DeleteObject接口删除某个对象，更多信息，请参见DeleteObject。API执行成功后触发函数。
oss:ObjectRemoved:DeleteObjects	调用DeleteObjects接口批量删除文件。API执行成功后触发函数。
oss:ObjectRemoved:AbortMultipartUpload	调用 AbortMultipartUpload 接口可以根据用户提供的Upload ID中止其对应的MultipartUpload事件，更多信息，请参见AbortMultipartUpload。API执行成功后触发函数。

 **说明** 一个Bucket最多能创建10个触发规则事件。

触发规则

使用OSS触发器一定要避免循环触发。一个典型的循环触发场景是OSS的某个Bucket上传文件触发函数执行，这个函数又生成了一个或多个文件，写回到OSS的Bucket里，这个写入动作又触发了函数执行，形成了链状循环。

上述过程类似于一个无限递归，为了避免这种循环触发函数产生不必要的费用，强烈建议您配置前缀或后缀，例如将触发函数的Bucket目录前缀设置成src/，生成的文件写入的目录前缀设置为dst/，这样生成的文件就不会再次触发函数。

一个Bucket的不同触发器如果指定了相同的事件类型，则前缀和后缀不能重复。假如一个触发器配置了oss:ObjectCreated:PutObject事件，source前缀和zip后缀。下表列出了是否可以给该Bucket配置其它触发器。

事件	前缀	后缀	是否合法	说明
oss:ObjectCreated:PutObject	source	zip	否	前后缀一样
oss:ObjectCreated:PutObject	source	无	否	前缀重合，后缀不设置包含了后缀为zip的对象。
oss:ObjectCreated:PutObject	无	zip	否	后缀重合，前缀不设置包含了前缀为source的对象。
oss:ObjectCreated:PutObject	source1	1zip	是	不重合
oss:ObjectCreated:PutObject	source	zip1	是	后缀不重合
oss:ObjectCreated:PutObject	1source	zip	是	前缀不重合

事件	前缀	后缀	是否合法	说明
oss:ObjectCreated:PostObject	source	zip	是	事件不同

说明

如果您需要让一个触发器触发多个函数，可以结合使用函数计算和Serverless工作流服务。先通过触发器触发一个函数，该函数启动Serverless工作流流程，在流程里您可以调用多个函数。更多信息，请参见[示例](#)。

6.2. 配置OSS触发器

本文以新对象存入OSS Bucket触发函数执行为例，介绍如何在函数计算控制台配置OSS触发器，包括创建触发器、编写函数和测试函数。

示例场景

您可以配置一个OSS触发器，在触发规则中设置前缀为 `source`。当有图片存入指定的OSS Bucket中的 `source` 目录下，会自动触发函数执行，函数将图片缩放并存储到同一个Bucket的 `processed` 目录下。即 `source/serverless.png` 会被处理成 `processed/serverless.png`。

前提条件

- 函数计算
 - [创建服务](#)
 - [创建函数](#)
- 对象存储OSS
 - [创建存储空间](#)

创建触发器

- 登录[函数计算控制台](#)。
- 在左侧导航栏，单击[服务及函数](#)。
- 在顶部菜单栏，选择地域。
- 在[服务列表](#)页面，找到目标服务，在其右侧操作列单击[函数管理](#)。
- 在[函数管理](#)页面，单击目标函数名称。



- 在函数详情页面，单击[触发器管理](#)页签，从版本或别名下拉列表选择要创建触发器的版本或别名，然后单击[创建触发器](#)。



7. 在创建触发器面板，填写相关信息。然后单击**确定**。

在函数 Function 中创建触发器

* 触发器类型

对象存储 OSS 异步调用

没有找到您想要的触发器？通过阿里云事件总线 EventBridge 您可以对接**近百款阿里云产品**。

* 名称

trigger-oss11/128

只能包含字母、数字、下划线和中划线。不能以数字、中划线开头。长度在 1-128 之间。

版本或别名

LATEST

* Bucket 名称

bucket-zh

文件前缀

source

文件后缀

zip

* 触发事件

oss:ObjectCreated:PutObject

* 角色名称

AliyunOSSEventNotificationRole

刷新

创建新的角色

确定

取消

参数	操作	本文示例
触发器类型	选择对象存储 OSS。	对象存储 OSS
名称	填写自定义的触发器名称。	oss-trigger
版本或别名	默认值为LATEST，如果您需要创建其他版本或别名的触发器，需先在函数详情页的版本或别名下拉列表选择该版本。关于版本简介，请参见 版本简介 。	LATEST
Bucket 名称	选择已创建的OSS Bucket。	bucket-zh****
文件前缀	输入要匹配的文件名称的前缀。强烈建议您配置前缀和后缀，避免触发事件嵌套循环触发引起额外费用。另外，一个Bucket的不同触发器如果指定了相同的事件类型，则前缀和后缀不能重复。详细信息，请参见。	source
文件后缀	输入要匹配的文件名称的后缀。强烈建议您配置前缀和后缀，避免触发事件嵌套循环触发引起额外费用。另外，一个Bucket的不同触发器如果指定了相同的事件类型，则前缀和后缀不能重复。详细信息，请参见。	zip
触发事件	选择一个或多个触发事件。关于对象存储OSS的事件类型，请参见 OSS事件定义 。 本示例选择oss:ObjectCreated:PutObject。	oss:ObjectCreated:PutObject
角色名称	选择AliyunOSSEventNotificationRole。  说明 如果您第一次创建该类型的触发器，则需要在单击确定后，在弹出的对话框中选择立即授权。	AliyunOSSEventNotificationRole

创建完成后，在触发器名称列表中显示已创建的触发器。

← 函数 Function 详情						版本或别名 LATEST	监控大盘	设置报警 已	日志仓库 已	删除
函数概述										
函数代码 触发器管理 函数配置 监控指标 调用日志 链路追踪 异步调用										
创建触发器 可按触发器名称进行模糊搜索。										
触发器名称 %	触发器类型 %	配置信息	状态 %	创建时间 %	操作					
trigger-oss	 对象存储 OSS 异步调用 已	Bucket 名称: bucket-zh**** 触发事件: oss:ObjectCreated:PutObject 文件前缀: source	已启用	今天 15:37:49	详情 编辑 删除					

编写函数

完成创建OSS触发器后，您可以开始编写函数代码，实现OSS事件触发函数计算自定义处理OSS Bucket 中的数据。

1. 在函数详情页面，单击函数代码页签，在代码编辑器中编写代码。

代码示例如下。

Nodejs

Python

PHP

Java

其中Java示例代码需增加如下依赖到pom.xml。

```
<dependencies>
  <dependency>
    <groupId>com.aliyun.fc.runtime</groupId>
    <artifactId>fc-java-core</artifactId>
    <version>1.3.0</version>
  </dependency>
  <dependency>
    <groupId>com.aliyun.fc.runtime</groupId>
    <artifactId>fc-java-event</artifactId>
    <version>1.0.0</version>
  </dependency>
</dependencies>
```

测试函数

完成函数编写后，您需要调试函数以验证代码的正确性。在实际操作过程中当发生OSS事件时，会自动触发函数执行。

OSS事件源会以event的形式作为输入参数传递给函数，您可以手动将event传给函数模拟触发事件，测试函数代码是否正确。

1. 在函数详情页面，单击**函数代码**页签，然后单击图标，从下拉列表中，选择**配置测试参数**。
2. 在**配置测试参数**对话框，选择**创建新测试事件**或**编辑已有测试事件**页签，填写事件名称和事件内容。然后单击**确定**。
event是函数计算的入口参数，当指定的OSS Bucket发生对象创建或对象删除事件时，会将事件数据以JSON格式发送给绑定的函数。具体格式如下所示。

```
{
  "events": [
    {
      "eventName": "ObjectCreated:PutObject",
      "eventSource": "acs:oss",
      "eventTime": "2021-08-13T06:45:43.000Z",
      "eventVersion": "1.0",
      "oss": {
        "bucket": {
          "arn": "acs:oss:cn-shanghai:123456789:bucketname",
          "name": "testbucket",
          "ownerIdentity": "123456789"
        },
        "object": {
          "deltaSize": 122539,
          "eTag": "688A7BF4F233DC9C88A80BF985AB7329",
          "key": "image/a.jpg",
          "size": 122539
        },
        "ossSchemaVersion": "1.0",
        "ruleId": "9adac8e253828f4f7c0466d941fa3db81161****"
      },
      "region": "cn-shanghai",
      "requestParameters": {
        "sourceIPAddress": "140.205.XX.XX"
      },
      "responseElements": {
        "requestId": "58F9FF2D3DF792092E12044C"
      },
      "userIdentity": {
        "principalId": "123456789"
      }
    }
  ]
}
```

event中涉及的参数解释如下表所示。

参数	类型	示例值	描述
eventName	String	ObjectCreated:PutObject	事件类型。
eventSource	String	acs:oss	事件源，固定为“acs:oss”。
eventTime	String	2021-08-13T06:45:43.000Z	事件产生的时间。使用ISO-8601标准时间格式。
eventVersion	String	1.0	事件协议的版本。
oss	Map		OSS事件内容。

参数	类型	示例值	描述
bucket	Map		bucket参数内容。
name	String	testbucket	Bucket的名称。
arn	String	acs:oss:cn-shanghai:123456789:b ucketname	Bucket的唯一标识符。
ownerIdentity	String	123456789	创建Bucket的用户ID。
object	Map		object参数内容。
size	Int	122539	object的大小。单位：Byte。
deltaSize	Int	122539	object的大小变化量。单位：Byte。 - 如果新增一个文件，此参数的值表示文件大小。 - 如果同名覆盖一个文件，此参数的值表示新文件与旧文件的大小差值。
eTag	String	688A7BF4F233DC9C88A 80BF985AB7329	Object的标签。
key	String	image/a.jpg	Object的名称。
ossSchemaVersion	String	1.0	OSS模式的版本号。
ruleId	String	9adac8e253828f4f7c0 466d941fa3db81161** **	事件匹配的规则ID。
region	String	cn-shanghai	Bucket所在的地域。
requestParameters	Map		请求参数。
sourceIPAddress	String	140.205.XX.XX	请求的源IP地址。
responseElements	Map		响应元素。
requestId	String	58F9FF2D3DF792092E1 2044C	请求对应的Request ID。
userIdentity	Map		用户属性。
principalId	String	123456789	请求发起者的阿里云账号ID。

3. 在函数详情页面，单击**函数代码**页签，然后单击**测试函数**。

更多信息

除了函数计算控制台，您还可通过以下方式配置触发器：

- 通过Serverless Devs工具配置触发器。更多操作，请参见[Serverless Devs](#)。
- 通过SDK配置触发器。更多操作，请参见[SDK列表](#)。

如需对创建的触发器进行修改或删除，具体操作，请参见[触发器管理](#)。

7.HTTP触发器

7.1. HTTP触发器概述

HTTP触发器通过发送HTTP请求触发函数执行，主要适用于快速构建Web服务等场景。HTTP触发器支持HEAD、POST、PUT、GET和DELETE方式触发函数。本文介绍HTTP触发器的优势、使用限制及入口函数形式。

优势

HTTP触发器与API网关触发器均可应用于Web应用的创建。相较于API网关触发器，HTTP触发器有以下优势：

- 降低了开发人员的学习成本和简化了开发人员的调试过程，帮助开发人员快速使用函数计算搭建Web Service和API。
- 支持选择熟悉的HTTP测试工具验证函数计算侧的功能和性能。
- 减少请求处理环节，HTTP触发器支持更高效的请求、响应格式，不需要编码或解码成JSON格式，性能更优。
- 方便对接其他支持Webhook回调的服务，例如CDN回源、MNS等。

使用限制

- 函数设置HTTP触发器后不能再设置其他类型的触发器。
- 每个函数只能创建一个HTTP触发器。
- 若您使用了版本管理功能，每个函数对应的版本或别名只能创建一个HTTP触发器。即一个函数的一个版本或一个别名对应可以创建一个HTTP触发器。版本和别名的详细信息，请参见[版本简介](#)。
- Request Headers不支持以x-fc-开头的自定义及以下字段的自定义：
 - accept-encoding
 - connection
 - keep-alive
 - proxy-authorization
 - te
 - trailer
 - transfer-encoding
- Response Headers不支持以x-fc-开头的自定义及以下字段的自定义：
 - connection
 - content-length
 - content-encoding
 - date
 - keep-alive
 - proxy-authenticate
 - server
 - trailer
 - transfer-encoding
 - upgrade

- content-disposition:attachment

 **说明** 从安全角度考虑, 使用函数计算默认的aliyuncs.com域名, 服务端会在Response Headers中强制添加content-disposition: attachment字段, 此字段会使得返回结果在浏览器中以附件的方式下载。如果要移除该限制, 需设置自定义域名。详细信息, 请参见[绑定自定义域名](#)。

- Request限制项。
如果超过以下限制, 会返回 400 状态码和 InvalidArgument 错误码。
 - Headers大小: Headers中的所有Key和Value的总大小不得超过4 KB。
 - Path大小: 包括所有的Query Params, Path的总大小不得超过4 KB。
 - Body大小: HTTP Body的总大小不得超过6 MB。
- Response限制项。
如果超过以下限制, 会返回 502 状态码和 BadResponse 错误码。
 - Headers大小: Headers中的所有Key和Value的总大小不得超过4 KB。
- 其他使用说明。
您可以通过绑定自定义域名为HTTP函数映射不同的HTTP访问路径。详细信息, 请参见[绑定自定义域名](#)。您还可以使用API网关, 后端服务类型选择HTTP服务, 设置HTTP函数为后端服务地址, 实现类似功能。详细信息, 请参见[使用函数计算作为API网关后端服务](#)。

HTTP触发器的入口函数形式

设置HTTP触发器的函数和普通函数的入口函数是有差异的, 左边的普通函数与右边设置了HTTP触发器函数的对比示例, 如下图所示。如果您设置HTTP触发器后仍然使用普通接口, 请及时修改。

- Node.js

```
1 module.exports.handler = function(event, context, callback) {  
2   console.log('hello world');  
3   callback(null, 'hello world');  
4 };  
~  
1 module.exports.handler = function(req, resp, context) {  
2   console.log("hello world");  
3   resp.send("hello world");  
4 }
```

- Python

```
1 #-*- coding: utf-8 -*-  
2 import logging  
3  
4 def handler(event, context):  
5   logger = logging.getLogger()  
6   logger.info('hello world')  
7   return 'hello world'  
  
1 #-*- coding: utf-8 -*-  
2 HELLO_WORLD = b"Hello world!\n"  
3  
4 def handler(env, start_response):  
5   context = env['fc.context']  
6   request_uri = env['fc.request_uri']  
7   for k, v in env.items():  
8     if k.startswith("HTTP_"):  
9       # process custom request headers  
10      pass  
11   # do something here  
12   status = '200 OK'  
13   response_headers = [('Content-type', 'text/plain')]  
14   start_response(status, response_headers)  
15   return [HELLO_WORLD]
```

其他更多内容请参见以下文档:

- [Node.js HTTP函数](#)
- [Python HTTP函数](#)
- [PHP HTTP函数](#)
- [Java HTTP函数](#)
- [C# HTTP函数](#)
- [简介](#)

7.2. 配置HTTP触发器

函数计算支持HTTP触发器，配置HTTP触发器的函数可以通过HTTP请求被触发执行。此时函数可以看做一个Web Server，对HTTP请求进行处理，并将处理结果返回给调用端。本文介绍如何在函数计算控制台配置HTTP触发器。

前提条件

创建服务

背景信息

创建HTTP触发器与创建其他类型的触发器不同。HTTP触发器相关参数是在创建函数的过程中配置，而其他类型的触发器相关参数是在已创建的函数中配置。

创建触发器

1. 登录[函数计算控制台](#)。
2. 在左侧导航栏，单击[服务及函数](#)。
3. 在顶部菜单栏，选择地域。
4. 在[服务列表](#)页面，找到目标服务，在其右侧操作列单击[函数管理](#)。
- 5.
6. 在[创建函数](#)页面，选择[从零开始创建](#)方式，然后在基本设置区域，填写函数相关信息后单击[创建](#)。

← 创建函数

请选择一种创建函数的方式

从零开始创建

从一个简单的 Hello World 示例开始，从头开始创建一个函数。

使用容器镜像创建

选择一个容器镜像来部署您的函数。

使用模版创建

使用示例模版，快速创建一个函数。

参数	操作	本文示例
名称	填写自定义的函数名称。	http-function
运行环境	选择您熟悉的语言，例如Python、Java、PHP、Node.js等。	Python3
函数触发方式	选择通过事件触发或通过 HTTP 请求触发。	通过 HTTP 请求触发
实例类型	选择弹性实例或性能实例。	弹性实例
内存规格	设置函数执行内存。取值范围如下： - 选择弹性实例，内存规格可以为128 MB、256 MB、512 MB、1 GB、2 GB或3 GB。 - 选择性能实例，内存规格可以为4 GB、8 GB、16 GB或32 GB。	512 MB

函数创建完成后，会自动创建一个默认的HTTP触发器。单击目标函数，然后单击[触发器管理](#)页签，您可以看到默认创建的HTTP触发器。



编写函数

完成创建HTTP触发器后，您可以开始编写函数代码。

- 1. 在函数详情页面，单击函数代码页签，在代码编辑器中编写代码。

代码示例如下：

Node.js Python PHP

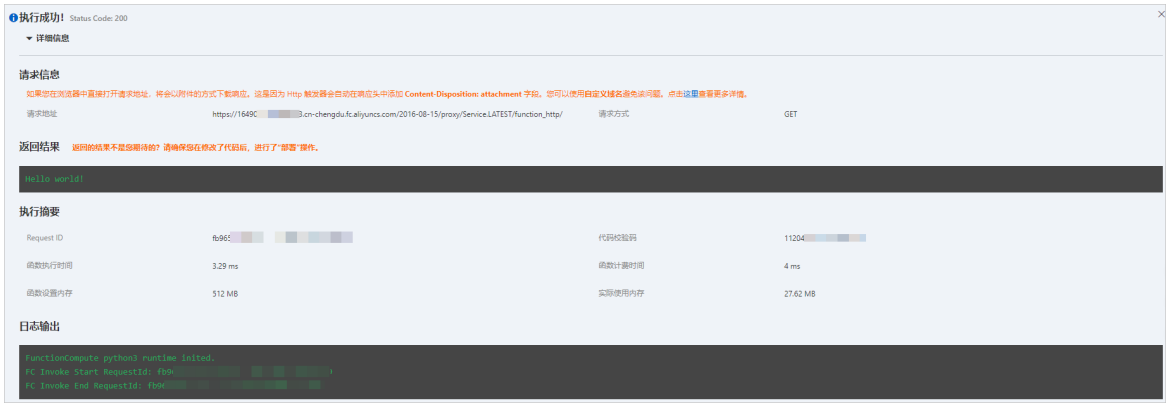
```
var getRawBody = require('raw-body')
module.exports.handler = function (request, response, context) {
  // get request info
  getRawBody(request, function (err, data) {
    var params = {
      path: request.path,
      queries: request.queries,
      headers: request.headers,
      method: request.method,
      body: data,
      url: request.url,
      clientIP: request.clientIP,
    }
    // you can deal with your own logic here
    // set response
    var respBody = new Buffer.from(JSON.stringify(params));
    // var respBody = new Buffer( )
    response.statusCode(200)
    response.setHeader('content-type', 'application/json')
    response.send(respBody)
  })
};
```

测试函数

您可以使用函数计算控制台或浏览器发送HTTP请求来测试函数的正确性。

使用控制台测试函数

- 1. 在函数详情页面，单击函数代码页签，然后单击测试函数。
执行完成后，在函数代码页签，您可以看到执行成功的提示。



使用浏览器测试函数

1. 组装HTTP触发器URL。

HTTP触发器URL的组装规则如下：

```
<account_id>.<region>.fc.aliyuncs.com/<version>/proxy/<serviceName>/<functionName>/[action?queries]
```

示例如下：

```
123456.cn-shanghai.fc.aliyuncs.com/2016-08-15/proxy/serviceName/functionName/action?hello=world
```

URL中的参数说明如下：

参数	说明
account_id	阿里云账号ID。 您可以在 安全设置 中获取您的账号ID。如果您是RAM用户，将鼠标移至控制台右上角的头像处可以看到您的阿里云账号ID。
region	函数计算服务所在的地域。
version	函数计算的API版本。
serviceName	服务名称。
functionName	函数名称。
action	自定义的请求路径。
queries	查询参数。

2. 将HTTP触发器URL输入浏览器地址栏，按回车键执行。
浏览器中会返回执行结果。

常见问题

为什么HTTP函数无法执行？

- 如果是您新创建的触发器，会有10s左右的缓存更新时间，请稍后再试。
- 请检查函数的入口函数是否正确，HTTP函数与普通函数的入口函数不同。不同语言的HTTP函数请参见以下文档：

- Node.js HTTP函数
- Python HTTP函数
- PHP HTTP函数
- Java HTTP函数
- C# HTTP函数
- Custom Runtime HTTP函数

为什么函数无法结束?

请检查是否调用返回函数。

- Node.js需调用 `response.send()` 。
- Python需调用 `return` 。
- PHP需调用 `return new Response()` 。
- Java需调用 `HttpServletResponse` 。
- C#需调用 `return` 。
- Custom Runtime以各语言示例为准。

错误排查

错误主要分为以下两种：

- 请求错误是指发送的Request不符合标准，在Response里报错状态码为4xx。
- 函数错误即编写的函数有问题，会报5xx状态码。

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。

错误类型	X-Fc-Error-Type	HTTP状态码	原因分析	是否计费
	FcCommonError	400	您的请求超过Request限制项的限制。详情请参见 使用限制 。	否
	FcCommonError	400	调用需要身份认证的函数的Request没有传入Date信息或Authorization信息。	否

错误类型	X-Fc-Error-Type	HTTP状态码	原因分析	是否计费
请求错误	FcCommonError	403	调用需要身份认证的函数的Request的签名错误，即Authorization不正确，由于Date参与签名计算，且超过15 min，签名失效，一种常见的原因是使用需要访问认证的HTTP触发器，Request header中发送的Date据当前时间超过15 min，导致签名失效。	否
	FcCommonError	403	您的Request请求使用了HTTP触发器中未配置的请求方法。例如，HTTP触发器中的请求方法只配置了GET方法，却发送POST方法的HTTP请求。	否
	FcCommonError	404	向没有设置HTTP触发器的函数发送HTTP请求。	否
用户流控	FcCommonError	429	用户被流控，可减小并发量或者联系函数计算开发团队提高并发度。	否
函数错误	UnhandledInvocationError	502	函数的返回值超过Response限制项的限制。详情请参见 使用限制 。	是
	UnhandledInvocationError	502	函数代码有语法错误或者异常。	是
	UnhandledInvocationError	502	向未使用HTTP入口函数的函数发送HTTP请求。	是
系统错误	FcCommonError	500	函数计算系统错误，可重试解决。	否
系统流控	FcCommonError	503	函数计算系统流控。可用指数退避方式重试。	否

如果问题还未能解决，请[联系我们](#)。

更多信息

除了函数计算控制台，您还可通过以下方式配置触发器：

- 通过Serverless Devs工具配置触发器。更多操作，请参见[Serverless Devs](#)。
- 通过SDK配置触发器。更多操作，请参见[SDK列表](#)。

如需对创建的触发器进行修改或删除，具体操作，请参见[触发器管理](#)。

8.MNS主题触发器

8.1. MNS主题触发器概述

消息服务MNS的主题（Topic）作为事件源通过事件总线EventBridge与函数计算集成后（即主题的订阅者是函数），通过消息服务MNS主题触发器（以下简称MNS主题触发器）能够触发关联函数执行，通过函数可以对发布到消息服务MNS主题中的消息进行自定义处理。

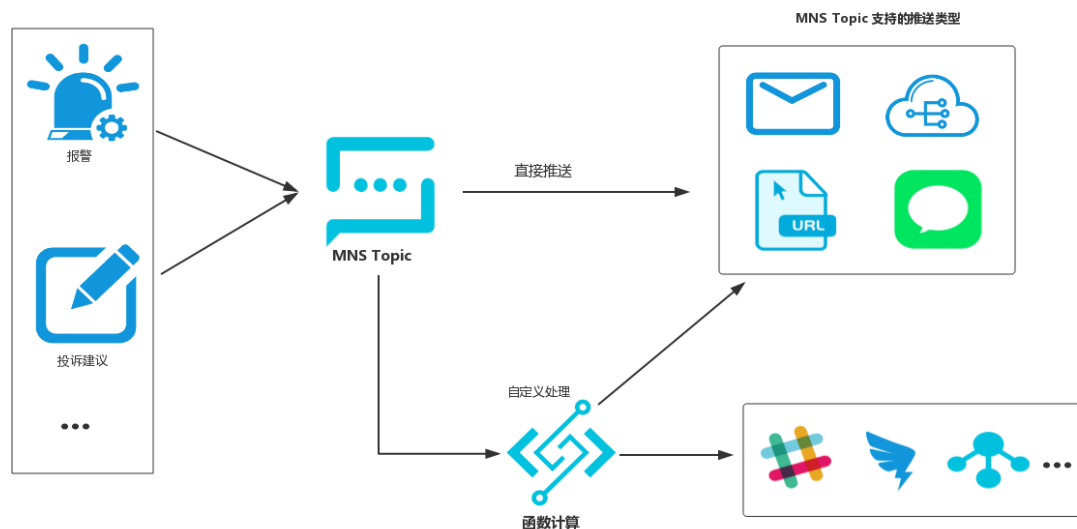
背景介绍

消息服务MNS是一种高效、可靠、安全、便捷、可弹性扩展的分布式消息服务。帮助应用开发者在其应用的分布式组件上自由地传递数据、通知消息，构建松耦合系统。在消息服务MNS中，主题是发布消息的目的地。发布者可以通过PublishMessage接口向主题发布消息，主题的订阅者接收该消息。接口信息，请参见[PublishMessage](#)。

配置一个MNS主题触发器，相当于将函数注册为这个MNS主题的订阅者，当发布者向MNS主题发布消息的时候，就会把消息内容通知给函数，即触发函数执行，同时消息内容作为函数入口的event参数。具体信息，请参见[函数入参](#)。

消息服务MNS与函数计算集成有以下优势：

- 可以实现对消息进行一些高阶处理再发送邮件或者短信。
- HTTP Endpoint不需要有自建的服务。
- 支持丰富的自定义处理。例如，把消息发送给slack，或者对于特定的消息进行持久化存储。



注意事项

- 强烈建议消息服务MNS创建的主题和函数计算的函数在相同的地域。
MNS主题触发器支持主题和函数在不同的地域，但是在不同的地域会增加网络延时，尤其当函数所在的地域和主题所在的地域分别是国内和国外时。
- 避免出现循环调用的情况。
编写函数时，注意不要出现以下逻辑：Topic A触发函数B，函数B又发布新的消息到Topic A，从而造成函数无限循环调用。

相关文档

● 配置MNS主题触发器

8.2. 配置MNS主题触发器

本文介绍如何通过函数计算控制台完成消息服务MNS主题触发器（以下简称MNS主题触发器）的配置，包括创建触发器、编写函数和测试函数。

前提条件

- 函数计算
 - 创建服务
 - 创建函数
- 消息服务MNS
 - 创建主题

创建触发器

1. 登录[函数计算控制台](#)。
2. 在左侧导航栏，单击服务及函数。
3. 在顶部菜单栏，选择地域。
4. 在服务列表页面，找到目标服务，在其右侧操作列单击函数管理。
5. 在函数管理页面，单击目标函数名称。



6. 在函数详情页面，单击触发器管理页签，从版本或别名下拉列表选择要创建触发器的版本或别名，然后单击创建触发器。



7. 在创建触发器面板，填写相关信息。然后单击确定。

在函数 Function 中创建触发器

* 触发器类型

消息服务 MNS topic 触发 异步调用

没有找到您想要的触发器？通过阿里云事件总线 EventBridge 您可以对接**近百款阿里云产品**。

* 名称

trigger-mns11/128

只能包含字母、数字、下划线和中划线。不能以数字、中划线开头。长度在 1-128 之间。

版本或别名

LATEST

* MNS 地域

西南1 (成都)

* 主题

Mytopic

过滤标签

tag

Event 格式

JSONSTREAM

重试策略

退避重试指数衰减重试

* 角色名称

AliyunMNSNotificationRole

刷新

创建新的角色

确定

取消

参数	操作	本文示例
触发器类型	选择消息服务 MNS。	消息服务MNS
名称	填写自定义的触发器名称。	trigger-mns

参数	操作	本文示例
版本或别名	默认值为 LATEST ，如果您需要创建其他版本或别名的触发器，需先在函数详情页的右上角切换到该版本或别名。关于版本简介，请参见 版本简介 。	LATEST
MNS地域	选择Topic所在的地域。 强烈建议MNS主题和函数计算的函数部署在相同的地域，否则会增加网络延时。	西南1（成都）
主题	在列表中选择已创建的Topic。	Mytopic
过滤标签	填写消息过滤标签。 只有收到包含了此处设置的过滤标签字符串的消息时，才会触发函数执行。	tag
Event格式	选择Event格式。取值： ◦ STREAM ◦ JSON	JSON
重试策略	选择重试策略。取值： ◦ 退避重试 ◦ 指数衰减 如何选择重试策略，请参见 NotifyStrategy 。	退避重试
角色名称	选择AliyunMNSNotificationRole。  说明 如果您第一次创建该类型的触发器，则需要在单击确定后，在弹出的对话框中选择立即授权。	AliyunMNSNotificationRole

创建完成后，在触发器名称列表中显示已创建的触发器。



编写函数

完成MNS主题触发器创建后，您可以开始编写函数代码。

1. 在函数详情页面，单击**函数代码**页签，在代码编辑器中编写代码。

本文以Python函数代码为例。以下示例代码可以作为MNS主题触发器的函数模板。

```
import json
import logging
def handler(event, context):
    logger = logging.getLogger()
    logger.info("mns_topic trigger event = {}".format(event))
    # 例如，将事件记录到表格存储。
    return "OK"
```

测试函数

完成函数编写后，您需要调试函数以验证代码的正确性。

1. 在函数详情页面，单击**函数代码**页签，然后单击图标，从下拉列表中，选择**配置测试参数**。
2. 在**配置测试参数**对话框，选择**创建新测试事件**或**编辑已有测试事件**页签，填写事件名称和事件内容。然后单击**确定**。

发布在MNS主题上的消息根据notifyContentFormat进行处理，即入口函数的event。更多信息，请参见[NotifyContentFormat](#)。

- 创建触发器时，若**event格式**设置为**STREAM**。
 - 当消息中不含消息属性（MessageAttributes）时，event格式如下。

 **说明** 当消息中不含消息属性（MessageAttributes）时，event的内容格式为JSON字符串。

```
# 消息正文。
'hello topic'
```

- 当消息中含有消息属性（MessageAttributes）时，event格式如下。

 **说明** event的内容中包含MessageAttributes相关的键值对。更多信息，请参见[PublishMessage](#)。

```
{
  "body": "hello topic",
  "attrs": {
    "Extend": "{\\"key\\":\\"value\\"}"
  }
}
```

- 创建触发器时，若event格式设置为**JSON**。

- 当消息中不含消息属性（MessageAttributes）时，event格式如下。

```
{
  "TopicOwner": "118620210433****",
  "Message": "hello topic",
  "Subscriber": "118620210433****",
  "PublishTime": 1550216480040,
  "SubscriptionName": "test-fc-subscribe",
  "MessageMD5": "BA4BA9B48AC81F0F9C66F6C909C3****",
  "TopicName": "Mytopic",
  "MessageId": "2F5B3C082B923D4EAC694B76D928****"
}
```

- 当消息中含有消息属性（MessageAttributes）时，event格式如下。

 说明 event的内容中包含MessageAttributes相关的键值对。更多信息，请参见[PublishMessage](#)。

```
{
  "key": "value",
  "TopicOwner": "118620210433****",
  "Message": "hello topic",
  "Subscriber": "118620210433****",
  "PublishTime": 1550216302888,
  "SubscriptionName": "test-fc-subscribe",
  "MessageMD5": "BA4BA9B48AC81F0F9C66F6C909C3****",
  "TopicName": "Mytopic",
  "MessageId": "2F5B3C281B283D4EAC694B742528****"
}
```

event中涉及的参数解释如下表所示。

参数	类型	示例值	描述
key	String	value	消息属性相关的键值对。
TopicOwner	String	118620210433****	订阅Topic的AccountId。
Message	String	hello topic	消息内容。
Subscriber	String	118620210433****	用户的AccountId。
PublishTime	Map	1550216302888	消息发布时间。
SubscriptionName	String	test-fc-subscribe	订阅的名称。
MessageMD5	String	BA4BA9B48AC81F0F9C66F6C909C3****	消息正文的MD5值。
TopicName	String	Mytopic	Topic名称。

参数	类型	示例值	描述
MessageId	String	2F5B3C281B283D4EAC694B742528****	消息的编号。

3. 在函数详情页面，单击**函数代码**页签，然后单击**测试函数**。

更多信息

除了函数计算控制台，您还可通过以下方式配置触发器：

- 通过Serverless Devs工具配置触发器。更多操作，请参见[Serverless Devs](#)。
- 通过SDK配置触发器。更多操作，请参见[SDK列表](#)。

如需对创建的触发器进行修改或删除，具体操作，请参见[触发器管理](#)。

9.SLS触发器

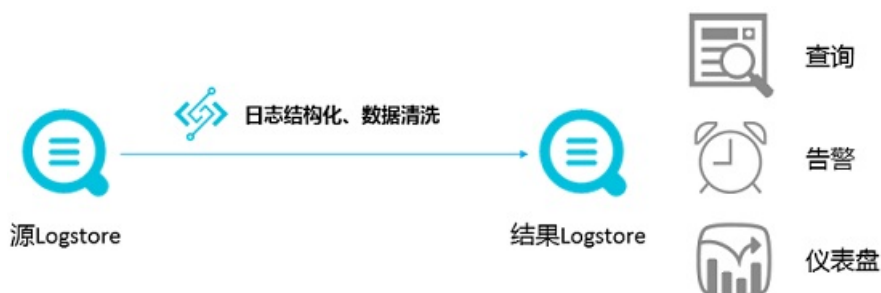
9.1. SLS触发器概述

依托阿里云函数计算服务，日志服务提供流式数据加工服务。您可以通过配置一个日志服务ETLJob，定时获取更新的数据并触发函数的执行，进而增量消费日志服务Logstore的数据，并完成自定义加工任务。

使用场景

日志服务触发器（以下简称SLS触发器）可以实现函数计算与日志服务的集成，集成的使用场景如下：

- 数据清洗、加工场景。
通过日志服务，快速完成日志采集、加工、查询、分析。



- 数据投递场景。
为数据的目的端落地提供支撑，构建云上大数据产品间的数据管道。

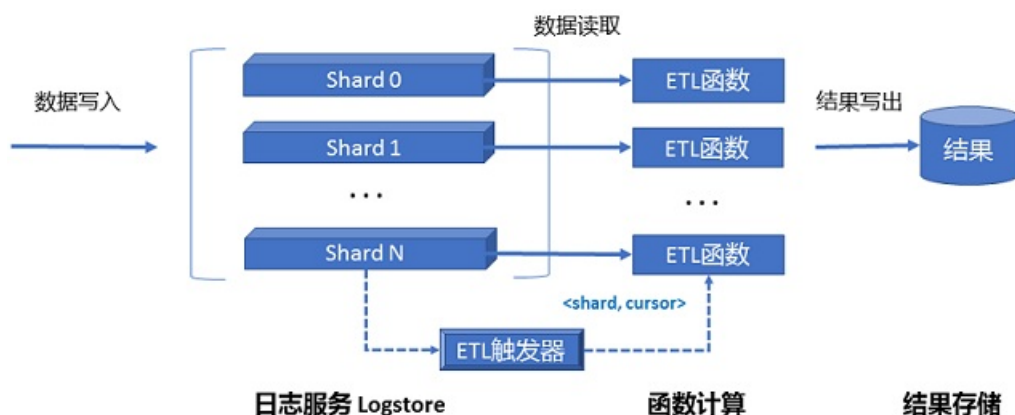


ETL函数

- 函数类型
 - 模板函数
更多信息，请参见[aliyun-log-fc-functions](#)。
 - 自定义函数
函数配置的格式与函数的具体实现有关。更多信息，请参见[ETL函数开发指南](#)。
- 函数计算触发机制
日志服务ETLJob对应于函数计算的一个触发器，当创建日志服务ETLJob后，日志服务会根据该ETLJob的配置启动定时器，定时器轮询Logstore中的Shard信息，当发现有新的数据写入时，即生成 `<shard_id,begin_cursor,end_cursor>` 三元组信息作为函数Event，并触发函数执行。

② 说明 当存储系统升级时，即使没有新数据写入，也可能发生Cursor变化，在这种情况下，每个Shard会额外空触发一次。针对这种情况，您可以在函数内通过Cursor尝试获取Shard的数据，如果获取不到数据说明是一次空触发，可以在函数内做忽略处理。更多信息，请参见[自定义函数开发指南](#)。

日志服务的ETL任务触发机制是时间触发。例如：您设置的ETLJob触发间隔为60秒，Logstore的Shard0一直有数据写入，那么Shard每60秒就会触发一次函数执行（如果Shard没有新的数据写入则不会触发函数执行），函数执行的输入为最近60秒的cursor区间。在函数内，可以根据cursor读取Shard0数据进行下一步处理。



9.2. 配置SLS触发器

您可以通过创建日志服务SLS触发器（以下简称SLS触发器）将日志服务SLS与函数计算连接起来，当有新日志产生时触发函数执行，对日志进行处理。本文以日志服务SLS调用函数计算，函数计算获取日志并打印为例，介绍如何在函数计算控制台上完成SLS触发器的配置，包括创建触发器、编写函数和测试函数。

示例场景

您可以配置一个SLS触发器，该触发器将定时获取更新的数据并触发函数执行，增量消费日志服务Logstore中的数据，在函数里完成自定义加工任务（例如，数据清洗和加工）以及将数据投递给第三方服务。本示例中只演示如何获取日志数据并打印。

② 说明 用于数据加工的函数可以是日志服务提供的模板，也可以是您的自定义函数。

前提条件

- 函数计算
 - [创建服务](#)
 - [创建函数](#)
- 日志服务SLS
 - [创建日志项目和日志仓库](#)

② 说明 需要创建一个日志项目和两个日志仓库。一个日志仓库用于处理日志及数据源，另一个日志仓库用于存储函数计算产生的日志。

创建触发器

1. 登录[函数计算控制台](#)。

- 2. 在左侧导航栏，单击服务及函数。
- 3. 在顶部菜单栏，选择地域。
- 4. 在服务列表页面，找到目标服务，在其右侧操作列单击函数管理。
- 5. 在函数管理页面，单击目标函数名称。



- 6. 在函数详情页面，单击触发器管理页签，从版本或别名下拉列表选择要创建触发器的版本或别名，然后单击创建触发器。



- 7. 在创建触发器面板，填写相关信息。然后单击确定。

在函数 Function 中创建触发器

* 触发器类型

日志服务 SLS 同步调用

没有找到您想要的触发器？通过阿里云事件总线 EventBridge 您可以对接近百款阿里云产品。

* 名称

log-trigger11/128

只能包含字母、数字、下划线和中划线。不能以数字、中划线开头。长度在 1-128 之间。

版本或别名

LATEST

* 日志项目

aliyun-fc-cn-chengdu-2238f0df-a742-524f-9f90-976ba457

* 日志仓库

function-log

* 触发间隔

60秒

* 重试次数

3次

* 触发器日志

function-log2

* 调用参数

* 角色名称

AliyunLogETLRole

刷新

创建新的角色

确定

取消

参数	操作	本文示例
触发器类型	选择日志服务 SLS。	日志服务SLS
名称	填写自定义的触发器名称。	log-trigger

参数	操作	本文示例
版本或别名	默认值为 LATEST ，如果您需要创建其他版本或别名的触发器，需先在函数详情页的右上角切换到该版本或别名。关于版本简介，请参见 版本简介 。	LATEST
日志项目	选择已创建的日志项目。	aliyun-fc-cn-chengdu-2238f0df-a742-524f-9f90-976ba457****
日志仓库	选择已创建的日志仓库，当前触发器会定时从该日志仓库中订阅数据到函数服务进行自定义加工。	function-log
触发间隔	填写日志服务触发函数运行的时间间隔。 取值范围：[3,600]，单位：秒。默认值：60。	60
重试次数	填写单次触发允许的最大重试次数。 取值范围：[0,100]。默认值：3。	3
触发器日志	选择已创建的日志仓库，日志服务触发函数执行过程的日志会记录到该日志仓库中。	function-log2
调用参数	如果您想传入自定义参数，可以在此处配置。该参数将作为event的parameter参数传入函数。该参数取值必须是JSON格式的字符串。 默认值为空。	无
角色名称	选择AliyunLogETLRole。  说明 如果您第一次创建该类型的触发器，则需要在单击确定后，在弹出的对话框中选择立即授权。	AliyunLogETLRole

创建完成后，在**触发器名称**列表中显示已创建的触发器。



编写函数

完成日志触发器创建后，您可以开始编写函数代码。日志服务收集增量日志时触发该函数，函数计算获取对应日志，然后打印收集的日志。

- 在函数详情页面，单击**函数代码**页签，在代码编辑器中编写代码。

本文以Python函数代码为例。以下示例代码可以作为提取大部分逻辑日志的模板。其中 `accessKeyId` 和 `accessKey` 可以填写在代码内，也可以从 `context` 和 `creds` 中获取。

```
# -*- coding: utf-8 -*-
import logging
import json
from aliyun.log import LogClient
from time import time
def logClient(endpoint, creds):
    logger = logging.getLogger()
    logger.info('creds info')
    logger.info(creds.access_key_id)
    logger.info(creds.access_key_secret)
    logger.info(creds.security_token)
    accessKeyId = 'your accessKeyId'
    accessKey = 'your accessKeyId scr'
    client = LogClient(endpoint, accessKeyId, accessKey)
    return client
def handler(event, context):
    logger = logging.getLogger()
    logger.info('start deal SLS data')
    logger.info(event.decode().encode())
    info_arr = json.loads(event.decode())
    fetchdata(info_arr['source'],context)
    return 'hello world'
def fetchdata(event,context):
    logger = logging.getLogger()
    endpoint = event['endpoint']
    creds = context.credentials
    client = logClient(endpoint, creds)
    if client == None :
        logger.info("client create failed")
        return False
    project = event['projectName']
    logstore = event['logstoreName']
    start_cursor = event['beginCursor']
    end_cursor = event['endCursor']
    loggroup_count = 10
    shard_id = event['shardId']
    while True:
        res = client.pull_logs(project, logstore, shard_id, start_cursor, loggroup_count, end_cursor)
        res.log_print()
        next_cursor = res.get_next_cursor()
        if next_cursor == start_cursor :
            break
        start_cursor = next_cursor
    #log_data = res.get_loggroup_json_list()
    return True
```

测试函数

完成函数编写后，您需要调试函数以验证代码的正确性。在实际操作过程中当日志仓库中有新增日志进入时会自动触发函数的执行。

1. 在函数详情页面，单击**函数代码**页签，然后单击图标，从下拉列表中，选择**配置测试参数**。
2. 在**配置测试参数**对话框，选择**创建新测试事件**或**编辑已有测试事件**页签，填写事件名称和事件内

容。然后单击**确定**。

event是函数计算的入口参数。具体格式如下：

```
{
  "parameter": {},
  "source": {
    "endpoint": "http://cn-shanghai-intranet.log.aliyuncs.com",
    "projectName": "log-com",
    "logstoreName": "log-en",
    "shardId": 0,
    "beginCursor": "MTUyOTQ4MDIwOTY1NTk3ODQ2Mw==",
    "endCursor": "MTUyOTQ4MDIwOTY1NTk3ODQ2NA=="
  },
  "jobName": "1f7043ced683de1a4e3d8d70b5a412843d81****",
  "taskId": "c2691505-38da-4d1b-998a-f1d4bb8c****",
  "cursorTime": 1529486425
}
```

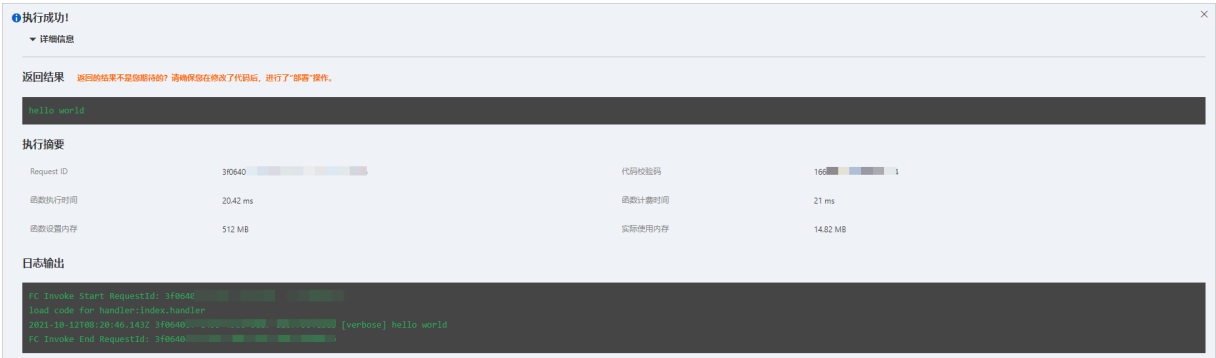
参数	描述	本文示例
parameter	您配置触发器时填写的调用参数的值。	无。
source	设置函数读取的日志块信息。 ◦ endpoint：日志服务Project所属的阿里云地域。 ◦ projectName：日志服务Project名称。 ◦ logstoreName：Logstore名称。 ◦ shardId：Logstore中一个确定的Shard。 ◦ beginCursor：开始消费数据的位置。 ◦ endCursor：停止消费数据的位置。	<pre>{ "endpoint": "http://cn-shanghai-intranet.log.aliyuncs.com", "projectName": "log-com", "logstoreName": "log-en", "shardId": 0, "beginCursor": "MTUyOTQ4MDIwOTY1NTk3ODQ2Mw==", "endCursor": "MTUyOTQ4MDIwOTY1NTk3ODQ2NA==" }</pre>

参数	描述	本文示例
jobName	日志服务ETL Job名字，函数配置的SLS触发器对应一个日志服务的ETL Job。	1f7043ced683de1a4e3d8d70b5a412843d81****
taskId	对于ETL Job而言，taskId是一个确定性的函数调用标识。	c2691505-38da-4d1b-998a-f1d4bb8c****
cursorTime	最后一条日志到达日志服务端的unix_timestamp。	1529486425

3. 在函数详情页面，单击函数代码页签，然后单击测试函数。

验证结果

在函数代码页签，您可以看到执行成功的指示。



更多信息

除了函数计算控制台，您还可通过以下方式配置触发器：

- 通过Serverless Devs工具配置触发器。更多操作，请参见[Serverless Devs](#)。
- 通过SDK配置触发器。更多操作，请参见[SDK列表](#)。

如需对创建的触发器进行修改或删除，具体操作，请参见[触发器管理](#)。

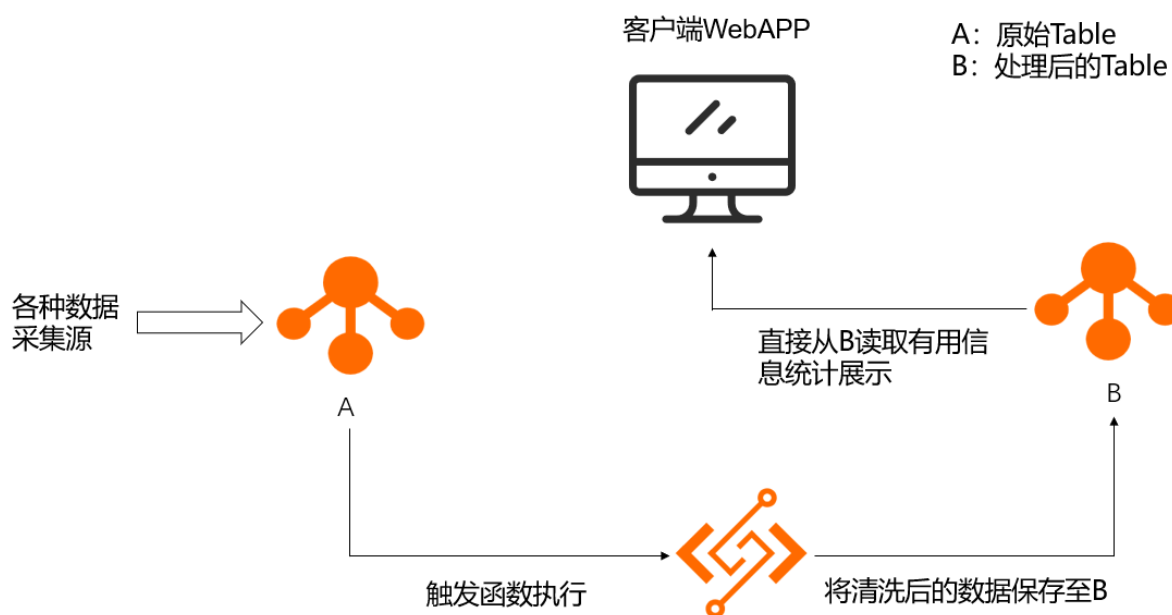
10.Tablestore触发器

10.1. Tablestore触发器概述

阿里云表格存储Tablestore是构建在阿里云飞天分布式系统之上的分布式NoSQL数据存储服务。Tablestore与阿里云函数计算集成，将Tablestore作为事件源接入函数计算，当Tablestore中的数据发生变更时，就会以数据变更信息作为参数触发函数的执行。本文介绍如何在函数计算中使用表格存储触发器（简称Tablestore触发器）。

典型场景

典型的使用场景如下图所示。



原始信号源数据存储到原始Table A，当Table A中的数据发生变更时会触发函数自定义清洗数据，将清洗后的数据存入Table B，您可以直接读取Table B的数据统计展示，完成一个弹性可伸缩的Serverless Web运用。

首次使用Tablestore触发器请务必先阅读[地域限制](#)和[注意事项](#)。

地域限制

目前支持Tablestore触发器的地域为：华北2（北京）、华东1（杭州）、华东2（上海）、华南1（深圳）、日本（东京）、新加坡、澳大利亚（悉尼）、德国（法兰克福）、中国香港。

注意事项

- 您编写函数的时候，请注意不要出现以下逻辑：表格存储Table A触发函数B，函数B又更新Table A的数据，从而造成函数无限循环调用。
- 若您需要使用内网访问Tablestore触发器对应的函数，您可以使用VPC Endpoint：{instance}.{region}.vpc.tablestore.aliyuncs.com，不可以使用Tablestore内网Endpoint。详细信息，请参见[开启VPC功能对可用性的影响](#)。
- 触发的函数执行时间不能超过一分钟。
- 若函数执行出现异常，函数将无限重试直到Tablestore中的日志数据过期。

更多信息

- [使用函数计算](#)
- [表格存储触发函数计算示例之Node.js、PHP、Java和C# Runtime](#)
- [联系我们](#)

10.2. 配置Tablestore触发器

完成Tablestore触发器的创建后，当Tablestore中的数据更新时会自动触发函数执行，对这些数据进行自定义处理。本文以使用Tablestore触发器触发函数获取数据为例，介绍如何在函数计算控制台完成Tablestore触发器的配置，包括创建触发器、编写函数和测试函数。

前提条件

- 函数计算
 - [创建服务](#)
 - [创建函数](#)
- 表格存储Tablestore
 - [创建Tablestore实例](#)
 - [创建数据表](#)

创建触发器

1. 登录[函数计算控制台](#)。
2. 在左侧导航栏，单击服务及函数。
3. 在顶部菜单栏，选择地域。
4. 在服务列表页面，找到目标服务，在其右侧操作列单击函数管理。
5. 在函数管理页面，单击目标函数名称。



6. 在函数详情页面，单击触发器管理页签，从版本或别名下拉列表选择要创建触发器的版本或别名，然后单击创建触发器。



7. 在创建触发器面板，填写相关信息，然后单击确定。

在函数 Function 中创建触发器

* 触发器类型

表格存储 Tablestore

同步调用

没有找到您想要的触发器？通过阿里云事件总线 EventBridge 您可以对接近百款阿里云产品。

* 名称

Tablestore-trigger

18/128

只能包含字母、数字、下划线和中划线。不能以数字、中划线开头。长度在 1-128 之间。

版本或别名

LATEST

* 实例

d00dd8xm

* 表格

mytable

* 角色名称

AliyunTableStoreStreamNotificationRole

刷新

创建新的角色

确定

取消

参数	操作	本文示例
触发器类型	选择表格存储 Tablestore。	表格存储Tablestore
名称	自定义填写触发器名称。	Tablestore-trigger

参数	操作	本文示例
版本或别名	默认值为LATEST，如果您需要创建其他版本或别名的触发器，需先在函数详情页的版本或别名下拉列表选择该版本或别名。关于版本简介，请参见 版本简介 。	LATEST
实例	在列表中选择已创建的Tablestore实例。	d00dd8xm****
表格	在列表中选择已创建的表格。	mytable
角色名称	选 择AliyunTableStoreStreamNotificationRole。  说明 如果您第一次创建该类型的触发器，则需要在单击确定后，在弹出的对话框中选择立即授权。	AliyunTableStoreStreamNotificationRole

创建完成后，在触发器名称列表中显示已创建的触发器。



编写函数

完成Tablestore触发器创建后，您可以开始编写函数代码。

1. 在函数详情页面，单击[函数代码](#)页签，在代码编辑器中编写代码。
- 本文以Python函数代码为例。如果您想使用其他运行环境，更多代码示例，请参见[表格存储触发函数计算示例](#)。

```
import logging
import cbor
import json
def get_attribute_value(record, column):
    attrs = record[u'Columns']
    for x in attrs:
        if x[u'ColumnName'] == column:
            return x['Value']
def get_pk_value(record, column):
    attrs = record[u'PrimaryKey']
    for x in attrs:
        if x['ColumnName'] == column:
            return x['Value']
def handler(event, context):
    logger = logging.getLogger()
    logger.info("Begin to handle event")
    #records = cbor.loads(event)
    records = json.loads(event)
    for record in records['Records']:
        logger.info("Handle record: %s", record)
        pk_0 = get_pk_value(record, "pk_0")
        attr_0 = get_attribute_value(record, "attr_0")
    return 'OK'
```

测试函数

完成函数编写后，您需要调试函数以验证代码的正确性。在实际操作过程中当Tablestore中有数据更新时会自动触发函数执行。

1. 在函数详情页面，单击**函数代码**页签，然后单击图标，从下拉列表中，选择**配置测试参数**。
2. 在**配置测试参数**对话框，选择**创建新测试事件**或**编辑已有测试事件**页签，填写事件名称和事件内容。然后单击**确定**。

表格存储触发器使用**CBOR**格式对增量数据进行编码，构成函数计算的event。具体格式如下所示。

```
{
  "Version": "Sync-v1",
  "Records": [
    {
      "Type": "PutRow",
      "Info": {
        "Timestamp": 1506416585740836
      },
      "PrimaryKey": [
        {
          "ColumnName": "pk_0",
          "Value": 1506416585881590900
        },
        {
          "ColumnName": "pk_1",
          "Value": "2017-09-26 17:03:05.8815909 +0800 CST"
        },
        {
          "ColumnName": "pk_2",
          "Value": 1506416585741000
        }
      ],
      "Columns": [
        {
          "Type": "Put",
          "ColumnName": "attr_0",
          "Value": "hello_table_store",
          "Timestamp": 1506416585741
        },
        {
          "Type": "Put",
          "ColumnName": "attr_1",
          "Value": 1506416585881590900,
          "Timestamp": 1506416585741
        }
      ]
    }
  ]
}
```

event中涉及的参数解释如下表所示。

参数	类型	示例值	描述
Version	String	Sync-v1	Payload版本号。
Records	Array		数据表中的增量数据行数 组。

参数	类型	示例值	描述
Type	String	PutRow	数据行类型，取值说明如下： - PutRow: 插入一行数据。 - UpdateRow: 更新指定行数据。 - DeleteRow: 删除指定行。
Info	String		数据行基本信息。
Timestamp	Int64	1506416585740836	该数据行的最后修改UTC时间。
PrimaryKey	Array		主键列数组。
ColumnName	String	pk_0	主键列名称。
Value	String、Integer或Blob	1506416585881590900	主键列内容。
Columns	Array		属性列数组。
Type	String	Put	属性列类型，取值说明如下： - Put: 增加一个版本。 - DeleteOneVersion: 删除一个版本。 - DeleteAllVersions: 删除所有版本。
ColumnName	String	attr_0	属性列名称。
Value	Integer、Boolean、Double、String或Blob	hello_table_store	属性列内容。
Timestamp	Int64	1506416585741	属性列最后修改UTC时间。

3. 在函数详情页面，单击[函数代码](#)页签，然后单击[测试函数](#)。

更多信息

除了函数计算控制台，您还可通过以下方式配置触发器：

- 通过Serverless Devs工具配置触发器。更多操作，请参见[Serverless Devs](#)。
- 通过SDK配置触发器。更多操作，请参见[SDK列表](#)。

如需对创建的触发器进行修改或删除，具体操作，请参见[触发器管理](#)。

11.CDN事件触发器

11.1. CDN事件触发器概述

阿里云内容分发网络CDN（Content Delivery Network）和函数计算无缝集成，您可以编写函数对CDN事件进行自定义处理。当CDN系统捕获到指定类型的、满足过滤条件的事件后，通过CDN事件触发器触发函数执行。

背景信息

阿里云CDN是建立并覆盖在承载网之上、由分布在不同区域的边缘节点服务器群组成的分布式网络。阿里云CDN可以替代传统以Web Server为中心的数据传输模式，将源站资源缓存到阿里云全国各地的边缘服务器，供您就近快速获取，提升用户体验，降低源站压力。在函数计算中通过配置内容分发网络事件触发器（以下简称“CDN事件触发器”）集成CDN服务可以实现您对CDN的各类事件进行自定义处理。例如，您可以设置函数和对应的CDN触发器来处理www.taobao.com域名下的资源刷新事件，当该域名下有资源刷新事件时，CDN事件触发器会自动触发函数执行。

使用场景

CDN事件触发器可以实现函数计算与CDN服务的集成，集成的使用场景如下：

- CDN在预热（CachedObjectsPushed）和刷新（CachedObjectsRefreshed）用户数据后，通过触发器执行函数。用户可以及时得知资源预热刷新的状态并进行下一步处理，避免不断轮询列表查询最新状态。
- 当在CDN上发现违禁内容（CachedObjectsBlocked）时，通过触发器执行函数直接去源站删除资源。您不需要等待CDN团队响应，可以及时去源站删除资源。
- 日志文件生成后（LogFileCreated），通过触发器执行函数处理日志。您不需要长时间等待日志，可以及时转存或处理日志。
- 当某加速域名被停用（CdnDomainStopped）或者被启用（CdnDomainStarted），通过触发器执行函数及时作出相应的处理。

CDN事件定义

当CDN系统捕获到相关事件后，会将事件信息编码为JSON字符串，传递给函数进行处理。CDN事件触发器当前支持的事件及版本如下表所示。

事件名称	事件版本	过滤参数	参考文档
CachedObjectsRefreshed	1.0.0	domain	刷新节点上的文件内容
CachedObjectsBlocked	1.0.0	domain	CDN封禁资源
CachedObjectsPushed	1.0.0	domain	预热源站内容到缓存节点
LogFileCreated	1.0.0	domain	获取加速域名的日志信息
CdnDomainStarted	1.0.0	domain	启用状态为停用的加速域名
CdnDomainStopped	1.0.0	domain	停用加速域名
CdnDomainAdded	1.0.0	domain	添加加速域名
CdnDomainDeleted	1.0.0	domain	删除已添加的加速域名

11.2. 配置CDN触发器

CDN触发器配置完成后，每当CDN服务在指定的加速域名下生成一个离线日志文件，CDN事件就会触发函数计算转存CDN离线日志。本文介绍如何在函数计算控制台上完成CDN事件触发器的配置，包括创建触发器、编写函数和测试函数。

示例场景

阿里云CDN支持的源站类型包括OSS域名、IP地址、源站域名和函数计算域名，每种源站类型都支持配置多个源站地址，多源站场景下，支持设置源站的主备优先级和权重，实现负载均衡。本文以OSS作为源站，您需要在[OSS控制台](#)创建一个存储空间，获取域名作为源站，然后为源站添加加速域名。如需变更源站类型，请参见[配置源站](#)。

前提条件

- 函数计算
 - [创建服务](#)
 - [创建函数](#)
- 对象存储OSS
 - [创建存储空间](#)
- CDN
 - [添加加速域名](#)

创建触发器

- 登录[函数计算控制台](#)。
- 在左侧导航栏，单击服务及函数。
- 在顶部菜单栏，选择地域。
- 在服务列表页面，找到目标服务，在其右侧操作列单击函数管理。
- 在函数管理页面，单击目标函数名称。



- 在函数详情页面，单击触发器管理页签，从版本或别名下拉列表选择要创建触发器的版本或别名，然后单击创建触发器。



- 在创建触发器面板，填写相关信息。然后单击确定。

在函数 Function 中创建触发器

* 触发器类型

CDN 同步调用

没有找到您想要的触发器？通过[阿里云事件总线 EventBridge](#) 您可以对接[近百款阿里云产品](#)。

* 名称

cdn-trigger11/128

只能包含字母、数字、下划线和中划线。不能以数字、中划线开头。长度在 1-128 之间。

版本或别名

LATEST

* 触发事件

CdnDomainStarted

触发事件版本

1.0.0

* 域名

bucket-zh-oss-cn-chengdu-internal.aliyuncs.com

+ 添加域名

* 描述

cdntrigger

* 角色名称

AliyunCDNEventNotificationRole

刷新

创建新的角色

确定

取消

参数	操作	本文示例
触发器类型	选择CDN。	CDN
名称	填写自定义的触发器名称。	cdn-trigger

参数	操作	本文示例
版本或别名	默认值为 LATEST ，如果您需要创建其他版本或别名的触发器，需先在函数详情页的 版本或别名 下拉列表选择该版本。关于版本简介，请参见 版本简介 。	LATEST
触发事件	选择一个触发事件类型。 如何选择触发事件类型，请参见 CDN事件定义 。	CdnDomainStarted
触发事件版本	触发事件的版本号。 说明 目前仅支持1.0.0事件版本。	1.0.0
域名	填写加速域名。	bucket-zh****.oss-cn-chengdu-internal.aliyuncs.com
描述	填写域名的描述。	cdntrigger
角色名称	选择 AliyunCDNEventNotificationRole 。 说明 如果您第一次创建该类型的触发器，单击确定后，则需在弹出的对话框中选择立即授权。	AliyunCDNEventNotificationRole

创建完成后，在**触发器名称**列表中显示已创建的触发器。



编写函数

完成CDN事件触发器的创建后，您可以开始写入函数代码，实现CDN事件触发函数计算转存CDN离线日志。

1. 在函数详情页面，单击**函数代码**页签，在代码编辑器中编写代码。

本文以Python函数代码为例。

```
import json
import oss2
import requests
def handler(event, context):
    creds = context.credentials
    auth = oss2.StsAuth(creds.access_key_id, creds.access_key_secret, creds.security_token)
    bucket = oss2.Bucket(auth, 'your endpoint', 'your bucket name')
    # Parse the event to get the source object info.
    eventObj = json.loads(event)["events"]
    fileUrl = eventObj[0]["eventParameter"]["filePath"]
    file = requests.get(fileUrl)
    result = bucket.put_object("your log name", fileUrl)
    return result.status
```

测试函数

完成函数编写后，您需要调试函数以验证代码的正确性。在实际操作过程中当发生CDN事件时，触发器会自动触发函数的执行。

1. 在函数详情页面，单击**函数代码**页签，然后单击图标，从下拉列表中，选择**配置测试参数**。
2. 在**配置测试参数**对话框，选择**创建新测试事件**或**编辑已有测试事件**页签，填写事件名称和事件内容。然后单击**确定**。

event是函数计算的入口参数，具体格式为如下。其中，对于不同的CDN事件类型，eventParameter中包含的键值对不同。

- o LogFileCreated事件的event示例。
替换 `filePath` 为您CDN日志的路径，或任何测试文件。

```
{
  "events": [
    {
      "eventName": "LogFileCreated",
      "eventSource": "cdn",
      "region": "cn-hangzhou",
      "eventVersion": "1.0.0",
      "eventTime": "2018-06-14T15:31:49+08:00",
      "traceld": "c6459282-6a4d-4413-894c-e4ea39686738",
      "userIdentity": {
        "aliUid": "164901546557****"
      },
      "resource": {
        "domain": "example.com"
      },
      "eventParameter": {
        "domain": "example.com",
        "endTime": 1528959900,
        "fileSize": 1788115,
        "filePath": "http://cdnlog.cn-hangzhou.oss.aliyun-inc.com/www.aliyun.com/2017_12_27/www.aliyun.com_2017_12_27_0800_0900.gz?OSSAccessKeyId=xxxx&Expires=xxxx&Signature=xxxx",
        "startTime": 1528959600
      }
    }
  ]
}
```

event中涉及的参数解释如下表所示。

参数	类型	示例值	描述
eventName	String	LogFileCreated	事件类型。
eventSource	String	cdn	事件源名称。
region	String	cn-hangzhou	CDN所在地域，默认为cn-hangzhou。
eventVersion	String	1.0.0	事件触发版本。
eventTime	String	2018-06-14T15:31:49+08:00	事件发生时间。
traceld	String	c6459282-6a4d-4413-894c-e4ea39686738	事件源传递的ID，用于排查问题。
userIdentity	String		用户属性。
aliUid	String	164901546557****	阿里云账号ID。
resource	String		资源信息。
domain	String	example.com	域名。

参数	类型	示例值	描述
eventParameter	String		事件参数。
endTime	String	1528959900	日志文件的结束时间。
fileSize	String	1788115	日志文件大小。
filePath	String	http://cdnlog.cn-hangzhou.oss.aliyun-inc.com/www.aliyun.com/2017_12_27/www.aliyun.com_2017_12_27_0800_0900.gz?OSSAccessKeyId=xxxx&Expires=xxxx&Signature=xxxx	日志文件地址。
startTime	String	1528959600	日志文件的起始时间。

- CachedObjectsRefreshed、CachedObjectsPushed和CachedObjectsBlocked事件的event示例。

```
{
  "events": [
    {
      "eventName": "CachedObjectsRefreshed",
      "eventVersion": "1.0.0",
      "eventSource": "cdn",
      "region": "cn-hangzhou",
      "eventTime": "2018-03-16T14:19:55+08:00",
      "traceld": "cf89e5a8-7d59-4bb5-a33e-4c3d08e25acf",
      "resource": {
        "domain": "example.com"
      },
      "eventParameter": {
        "objectPath": [
          "/2018/03/16/13/33b430c57e7.mp4",
          "/2018/03/16/14/4ff6b9bd54d.mp4"
        ],
        "createTime": 1521180769,
        "domain": "example.com",
        "completeTime": 1521180777,
        "objectType": "File",
        "taskId": 2089687230
      },
      "userIdentity": {
        "aliUid": "164901546557****"
      }
    }
  ]
}
```

event中涉及的参数解释如下表所示。

参数	类型	示例值	描述
eventName	String	CachedObjectsRefreshed	事件类型。
eventSource	String	cdn	事件源名称。
region	String	cn-hangzhou	CDN所在地域，默认为cn-hangzhou。
eventVersion	String	1.0.0	事件触发版本。
eventTime	String	2018-06-14T15:31:49+08:00	事件发生时间。
traceId	String	c6459282-6a4d-4413-894c-e4ea39686738	事件源传递的ID，用于排查问题。
resource	String		资源信息。
domain	String	example.com	域名。
eventParameter	String		事件参数。
objectPath	String	/2018/03/16/13/33b430c57e7.mp4	资源标识。
createTime	String	1521180769	刷新开始时间。
completeTime	String	1521180777	刷新结束时间。
objectType	String	File	刷新类型，取值说明如下： - File: 文件 - Directory: 文件夹
taskId	String	2089687230	资源刷新任务ID。
userIdentity	String		用户属性。
aliUid	String	164901546557****	阿里云账号ID。

- CdnDomainStarted和CdnDomainStopped事件的event示例。

```
{ "events": [
  {
    "eventName": "CdnDomainStarted",
    "eventVersion": "1.0.0",
    "eventSource": "cdn",
    "region": "cn-hangzhou",
    "eventTime": "2018-03-16T14:19:55+08:00",
    "traceld": "cf89e5a8-7d59-4bb5-a33e-4c3d08e25acf",
    "resource": {
      "domain": "example.com"
    },
    "eventParameter": {
      "domain": "example.com",
      "status": "online"
    },
    "userIdentity": {
      "aliUid": "164901546557****"
    }
  }
]
```

event中涉及的参数解释如下表所示。

参数	类型	示例值	描述
eventName	String	CachedObjectsRefreshed	事件类型。
eventSource	String	cdn	事件源名称。
region	String	cn-hangzhou	CDN所在地域，默认为cn-hangzhou。
eventVersion	String	1.0.0	事件触发版本。
eventTime	String	2018-06-14T15:31:49+08:00	事件发生时间。
traceld	String	c6459282-6a4d-4413-894c-e4ea39686738	事件源传递的ID，用于排查问题。
resource	String		资源信息。
domain	String	example.com	域名。
eventParameter	String		事件参数。
status	String	online	域名状态。
userIdentity	String		用户属性。
aliUid	String	164901546557****	阿里云账号ID。

- CdnDomainAdded和CdnDomainDeleted事件的event示例。

```
{ "events": [
  {
    "eventName": "CdnDomainStarted",
    "eventVersion": "1.0.0",
    "eventSource": "cdn",
    "region": "cn-hangzhou",
    "eventTime": "2018-03-16T14:19:55+08:00",
    "traceld": "cf89e5a8-7d59-4bb5-a33e-4c3d08e25acf",
    "resource": {
      "domain": "example.com"
    },
    "eventParameter": {
      "domain": "example.com"
    },
    "userIdentity": {
      "aliUid": "164901546557****"
    }
  }
]
```

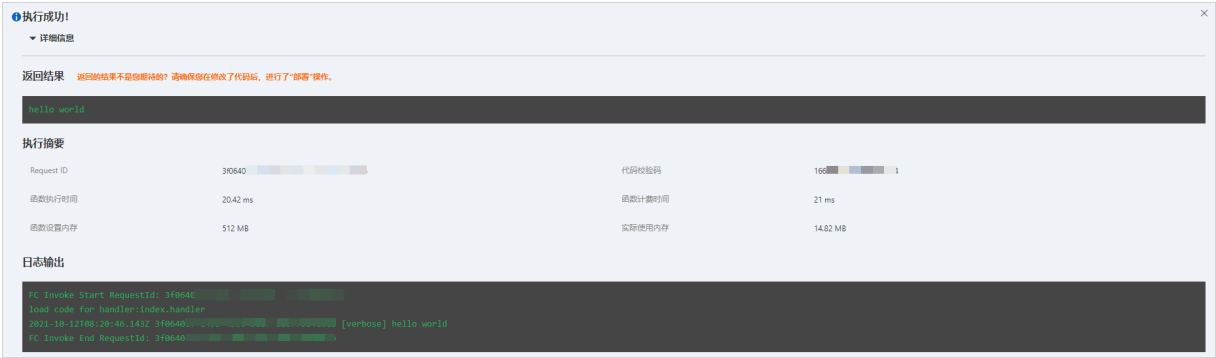
event中涉及的参数解释如下表所示。

参数	类型	示例值	描述
eventName	String	CachedObjectsRefreshed	事件类型。
eventSource	String	cdn	事件源名称。
region	String	cn-hangzhou	CDN所在地域，默认为cn-hangzhou。
eventVersion	String	1.0.0	事件触发版本。
eventTime	String	2018-06-14T15:31:49+08:00	事件发生时间。
traceld	String	c6459282-6a4d-4413-894c-e4ea39686738	事件源传递的ID，用于排查问题。
resource	String		资源信息。
domain	String	example.com	域名。
eventParameter	String		事件参数。
domain	String	example.com	域名。
userIdentity	String		用户属性。
aliUid	String	164901546557****	阿里云账号ID。

3. 在函数详情页面，单击函数代码页签，然后单击测试函数。

验证结果

在函数代码页签，您可以看到执行成功的指示。



更多信息

除了函数计算控制台，您还可通过以下方式配置触发器：

- 通过Serverless Devs工具配置触发器。更多操作，请参见[Serverless Devs](#)。
- 通过SDK配置触发器。更多操作，请参见[SDK列表](#)。

如需对创建的触发器进行修改或删除，具体操作，请参见[触发器管理](#)。

12. 定时触发器

12.1. 定时触发器概述

函数计算支持配置定时触发器（Time Trigger），可以在指定的时间点自动触发函数执行。

使用场景

定时触发器会在指定时间自动触发函数执行。定时任务的场景非常广泛，包括但不限于以下使用场景：

- 批量数据的定时处理，例如每1小时收集全量数据并生成报表。
- 日常行为的调度，例如整点发送优惠券。
- 与业务解耦的异步任务，例如每天0点清理数据。

12.2. 配置定时触发器

本文介绍如何在函数计算控制台配置定时触发器，包括创建触发器、编写函数和测试函数。

前提条件

- [创建服务](#)
- [创建服务](#)

创建触发器

1. 登录[函数计算控制台](#)。
2. 在左侧导航栏，单击服务及函数。
3. 在顶部菜单栏，选择地域。
4. 在服务列表页面，找到目标服务，在其右侧操作列单击函数管理。
5. 在函数管理页面，单击目标函数名称。



6. 在函数详情页面，单击触发器管理页签，从版本或别名下拉列表选择要创建触发器的版本或别名，然后单击创建触发器。



7. 在创建触发器面板，填写相关信息。然后单击确定。

在函数 Function 中创建触发器

* 触发器类型

 定时触发器

异步调用

 没有找到您想要的触发器？通过[阿里云事件总线 EventBridge](#) 您可以对接[近百款阿里云产品](#)。

* 名称

my_trigger

10/128

只能包含字母、数字、下划线和中划线。不能以数字、中划线开头。长度在 1-128 之间。

版本或别名

①

LATEST

触发方式

按照时间间隔触发

按照 CRON 表达式触发

* 时间间隔

5

分钟

触发消息

①

{awesome-fc}

确定

取消

参数	操作	本文示例
触发器类型	选择定时触发器。	定时触发器
名称	填写自定义的触发器名称。	my_trigger

参数	操作	本文示例
版本或别名	默认值为 LATEST ，如果您需要创建其他版本或别名的触发器，需先在函数详情页的右上角切换到该版本或别名。关于版本简介，请参见 版本简介 。	LATEST
触发方式	根据需要选择触发方式： - 按照时间间隔触发：在时间间隔文本框输入正整数n，表示每n分钟触发一次函数执行。 - 按照 CRON 表达式触发：在CRON 表达式文本框中输入Cron表达式，到达指定的时间点触发一次函数执行。	时间间隔：5分钟
触发消息	输入自定义的参数。该触发消息将会作为event中payload的值。	<pre>{awesome-fc}</pre>

时间表达式

- Cron表达式
Cron表达式默认以UTC时间运行，即北京时间减去8个小时。例如北京时间每天12:00调度函数，那么转化为UTC时间就是每天4:00调度函数，则可以使用 `0 0 4 * * *`。
- 时区表达式
如果您的任务需要按照特定时区运行，可以通过CRON_TZ指定，例如在北京时间每个月一号的04:00触发函数执行，则可以使用 `CRON_TZ=Asia/Shanghai 0 0 4 1 * *`。不同地域的时区表达式存在差异，请以实际情况为准。

说明

如果您使用的时区存在夏令时和冬令时的区分，在夏令时和冬令时切换的过程中可能会出现重复执行或少执行的情况，建议将执行时间设置在夏令时和冬令时切换的时间段外。

下表是时间表达式常用示例。例如您希望函数计算的函数在北京时间12:00被执行，对应的Cron表达式为 `0 0 4 * * *`，时区表达式为 `CRON_TZ=Asia/Shanghai 0 0 12 * * *`。

示例	Cron表达式	时区表达式
每天12:00调度函数	<code>0 0 4 * * *</code>	<code>CRON_TZ=Asia/Shanghai 0 0 12 * * *</code>
每天12:30调度函数	<code>0 30 4 * * *</code>	<code>CRON_TZ=Asia/Shanghai 0 30 12 * * *</code>
每小时的26分，29分，33分调度函数	<code>0 26,29,33 * * * *</code>	<code>CRON_TZ=Asia/Shanghai 0 26,29,33 * * * *</code>
周一到周五的每天12:30调度函数	<code>0 30 4 ? * MON-FRI</code>	<code>CRON_TZ=Asia/Shanghai 0 30 12 ? * MON-FRI</code>
周一到周五的每天12:00 ~ 14:00 每5分钟调度函数	<code>0 0/5 4-6 ? * MON-FRI</code>	<code>CRON_TZ=Asia/Shanghai 0 0/5 12-14 ? * MON-FRI</code>

示例	Cron表达式	时区表达式
一月到四月每天12:00调度函数	0 0 4 ? JAN,FEB,MAR,APR *	CRON TZ=Asia/Shanghai 0 0 1 2 ? JAN,FEB,MAR,APR *

Cron表达式的字段说明如下：

○ 字段说明

字段名	取值范围	允许的特殊字符
Seconds	0 ~ 59	无
Minutes	0 ~ 59	, - * /
Hours	0 ~ 23	, - * /
Day-of-month	1 ~ 31	, - * ? /
Month	1 ~ 12或JAN ~ DEC	, - * /
Day-of-week	0 ~ 6或MON ~ SUN	, - * ?

○ 特殊字符说明

字符名	定义	示例
*	表示任一或每一。	在Minutes字段中：0表示每分钟的0秒都执行。
,	表示列表值。	在Day-of-week字段中：MON,WED,FRI 表示星期一，星期三和星期五。
-	表示一个范围。	在Hours字段中：10-12表示UTC时间从10点到12点。
?	表示不确定的值。	与其他指定值一起使用。例如，如果指定了一个特定的日期，但您不在乎它是星期几，那么在Day-of-week字段中就可以使用。
/	表示一个值的增加幅度，n/m表示从n开始，每次增加m。	在minute字段中：3/5表示从3分钟开始，每隔5分钟执行一次。

创建完成后，在触发器名称列表中显示已创建的触发器。

← 函数 Function 详情						版本选择名 LATEST	监控大盘	设置报警 已	日志查询 已	删除
函数概览										
函数代码 触发器管理 函数配置 监控指标 调用日志 错误追踪 异步调用										
创建触发器 可按触发器名称进行模糊搜索。										
触发器名称 已	触发器类型 已	配置信息	状态 已	创建时间 已	操作					
timer-trigger	定时触发器 异步调用 已	CRON 表达式 @every 5m 触发器名称 awesome-fc	已启用	今天 14:01:08	详情 编辑 禁用 更多					

编写函数

完成定时触发器创建后，您可以开始编写函数代码，实现定时触发函数执行。

- 1. 在函数详情页面，单击**函数代码**页签，在代码编辑器中编写代码。

本文以Python函数代码为例，示例代码如下。

```
import json
import logging
logger = logging.getLogger()
def handler(event, context):
    logger.info('event: %s', event)
    evt = json.loads(event)
    triggerName = evt["triggerName"]
    triggerTime = evt["triggerTime"]
    message = evt["payload"]
    creds = context.credentials
    logger.info('access_key_id: %s', creds.access_key_id)
    logger.info("message = %s", message)
```

测试函数

完成函数编写后，您需要调试函数以验证代码的正确性。在实际操作过程中当到达指定的时间，定时触发器会自动触发函数执行。

- 1. 在函数详情页面，单击**函数代码**页签，然后单击 图标，从下拉列表中，选择**配置测试参数**。
- 2. 在**配置测试参数**对话框，选择**创建新测试事件**或**编辑已有测试事件**页签，填写事件名称和事件内容。然后单击**确定**。

定时触发器会按照以下event格式来触发函数。

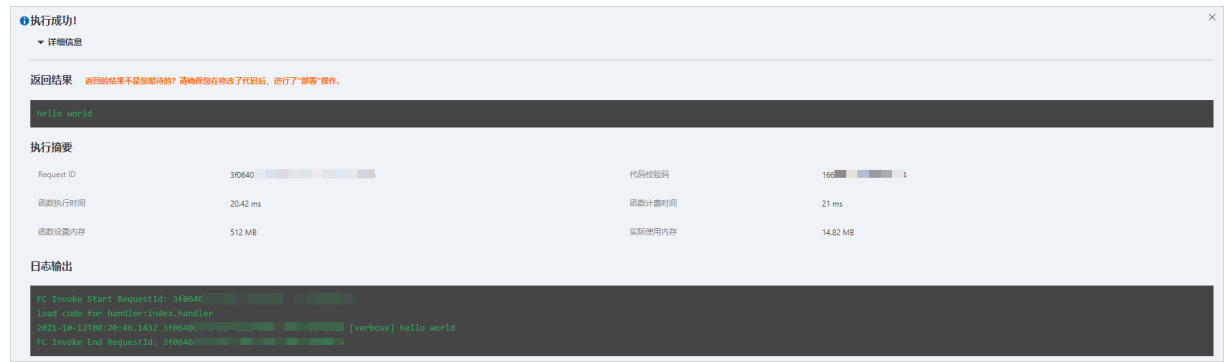
```
{
  "triggerTime": "2018-02-09T05:49:00Z",
  "triggerName": "timer-trigger",
  "payload": "awesome-fc"
}
```

参数	类型	示例值	描述
triggerTime	String	2018-02-09T05:49:00Z	函数被触发的时间。
triggerName	String	timer-trigger	定时触发器的名称。
payload	String	awesome-fc	您在触发器配置里输入的自定义参数，即 触发消息 的值。

- 3. 在函数详情页面，单击**函数代码**页签，然后单击**测试函数**。

验证结果

在**函数代码**页签，您可以看到执行成功的指示。



更多信息

除了函数计算控制台，您还可通过以下方式配置触发器：

- 通过Serverless Devs工具配置触发器。更多操作，请参见[Serverless Devs](#)。
- 通过SDK配置触发器。更多操作，请参见[SDK列表](#)。

如需对创建的触发器进行修改或删除，具体操作，请参见[触发器管理](#)。

13.API网关触发器

13.1. API网关触发器概述

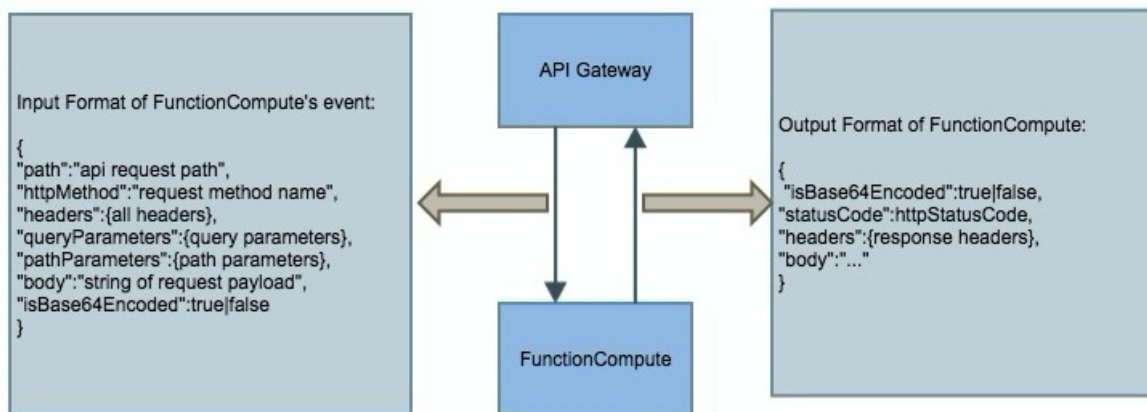
函数计算支持API网关作为事件源，当有请求到达后端服务设置为函数计算的API网关时，API网关会触发函数的执行。函数计算会将执行结果返回给API网关。

使用场景

API网关触发器与HTTP触发器类似，可应用于搭建Web应用。相较于HTTP触发器，您可以使用API网关进行IP白名单或黑名单设置等高级操作。

实现原理

API网关调用函数计算服务时，会将API的相关数据转换为Map形式传给函数计算服务。函数计算服务处理后，按照Output Format格式返回statusCode、headers、body等相关数据。API网关再将函数计算返回的内容映射到statusCode、header、body等位置返回给客户端。



13.2. 配置API网关触发器

函数计算支持API网关作为事件源，即将函数计算设置为API的后端服务。配置完成后，当请求设置后端服务为函数计算时，API网关会触发关联的函数执行一次，函数计算将执行结果返回给API网关。本文介绍配置API网关触发函数执行的流程，包括编写函数，测试函数和验证结果等。

前提条件

- 函数计算
 - [创建服务](#)
 - [创建函数](#)
- API网关
 - [创建后端为函数计算的API](#)

编写函数

完成API创建后，您可以开始编写函数代码。

1. 登录[函数计算控制台](#)。
2. 在左侧导航栏，单击[服务及函数](#)。
3. 在顶部菜单栏，选择地域。

4. 在服务列表页面，找到目标服务，在其右侧操作列单击函数管理。
5. 在函数管理页面，单击目标函数名称。



6. 在函数详情页面，单击函数代码页签，在代码编辑器中编写代码。

不同语言的示例代码如下：

- o Node.js

```
module.exports.handler = function(event, context, callback) {
  var event = JSON.parse(event);
  var content = {
    path: event.path,
    method: event.method,
    headers: event.headers,
    queryParameters: event.queryParameters,
    pathParameters: event.pathParameters,
    body: event.body
  };
  // 您可以在这里编写您自己的逻辑。
}
var response = {
  isBase64Encoded: false,
  statusCode: '200',
  headers: {
    'x-custom-header': 'header value'
  },
  body: content
};
callback(null, response);
};
```

- o Python

```
# -*- coding: utf-8 -*-
import json
def handler(event, context):
    event = json.loads(event)
    content = {
        'path': event['path'],
        'method': event['httpMethod'],
        'headers': event['headers'],
        'queryParameters': event['queryParameters'],
        'pathParameters': event['pathParameters'],
        'body': event['body']
    }
    # 您可以在这里编写您自己的逻辑。
    rep = {
        "isBase64Encoded": "false",
        "statusCode": "200",
        "headers": {
            "x-custom-header": "no"
        },
        "body": content
    }
    return json.dumps(rep)
```

- PHP

```
<?php
function handler($event, $context) {
    $event = json_decode($event, $assoc = true);
    $content = [
        'path' => $event['path'],
        'method' => $event['httpMethod'],
        'headers' => $event['headers'],
        'queryParameters' => $event['queryParameters'],
        'pathParameters' => $event['pathParameters'],
        'body' => $event['body'],
    ];
    $rep = [
        "isBase64Encoded" => "false",
        "statusCode" => "200",
        "headers" => [
            "x-custom-header" => "no",
        ],
        "body" => $content,
    ];
    return json_encode($rep);
}
```

- Java

使用Java编程时，必须要实现一个类，需要实现函数计算预定义的handler，目前有两个预定义的handler可以实现（任选其一即可）。函数计算Java运行环境，请参见[Java运行环境](#)。

- （推荐）使用PojoRequestHandler<I, O> handler。

```
import com.aliyun.fc.runtime.Context;
import com.aliyun.fc.runtime.PojoRequestHandler;
import java.util.HashMap;
import java.util.Map;
public class ApiTriggerDemo implements PojoRequestHandler<ApiRequest, ApiResponse> {
    public ApiResponse handleRequest(ApiRequest request, Context context) {
        // 获取API请求信息。
        context.getLogger().info(request.toString());
        String path = request.getPath();
        String httpMethod = request.getHttpMethod();
        String body = request.getBody();
        context.getLogger().info("path: " + path);
        context.getLogger().info("httpMethod: " + httpMethod);
        context.getLogger().info("body: " + body);
        // 您可以在这里编写您自己的逻辑。
        // API返回示例。
        Map headers = new HashMap();
        boolean isBase64Encoded = false;
        int statusCode = 200;
        String responseBody = "";
        return new ApiResponse(headers, isBase64Encoded, statusCode, responseBody);
    }
}
```

- 两个 POJO 类、ApiRequest 类和 ApiResponse 类定义如下。

 说明 POJO 类的 set() 和 get() 方法要写全。

```
import java.util.Map;
public class ApiRequest {
    private String path;
    private String httpMethod;
    private Map headers;
    private Map queryParameters;
    private Map pathParameters;
    private String body;
    private boolean isBase64Encoded;
    @Override
    public String toString() {
        return "Request{" +
            "path='" + path + '\'' +
            ", httpMethod='" + httpMethod + '\'' +
            ", headers=" + headers +
            ", queryParameters=" + queryParameters +
            ", pathParameters=" + pathParameters +
            ", body='" + body + '\'' +
            ", isBase64Encoded=" + isBase64Encoded +
            '}';
    }
    public String getPath() {
        return path;
    }
    public void setPath(String path) {
```

```
        this.path = path;
    }
    public String getHttpMethod() {
        return httpMethod;
    }
    public void setHttpMethod(String httpMethod) {
        this.httpMethod = httpMethod;
    }
    public Map getHeaders() {
        return headers;
    }
    public void setHeaders(Map headers) {
        this.headers = headers;
    }
    public Map getQueryParameters() {
        return queryParameters;
    }
    public void setQueryParameters(Map queryParameters) {
        this.queryParameters = queryParameters;
    }
    public Map getPathParameters() {
        return pathParameters;
    }
    public void setPathParameters(Map pathParameters) {
        this.pathParameters = pathParameters;
    }
    public String getBody() {
        return body;
    }
    public void setBody(String body) {
        this.body = body;
    }
    public boolean getIsBase64Encoded() {
        return this.isBase64Encoded;
    }
    public void setIsBase64Encoded(boolean base64Encoded) {
        this.isBase64Encoded = base64Encoded;
    }
}
```

```
import java.util.Map;
public class ApiResponse {
    private Map headers;
    private boolean isBase64Encoded;
    private int statusCode;
    private String body;
    public ApiResponse(Map headers, boolean isBase64Encoded, int statusCode, String body) {
        this.headers = headers;
        this.isBase64Encoded = isBase64Encoded;
        this.statusCode = statusCode;
        this.body = body;
    }
    public Map getHeaders() {
        return headers;
    }
    public void setHeaders(Map headers) {
        this.headers = headers;
    }
    public boolean getIsBase64Encoded() {
        return isBase64Encoded;
    }
    public void setIsBase64Encoded(boolean base64Encoded) {
        this.isBase64Encoded = base64Encoded;
    }
    public int getStatusCode() {
        return statusCode;
    }
    public void setStatusCode(int statusCode) {
        this.statusCode = statusCode;
    }
    public String getBody() {
        return body;
    }
    public void setBody(String body) {
        this.body = body;
    }
}
```

- *pom.xml*文件如下。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>apiTrigger</groupId>
  <artifactId>apiTrigger</artifactId>
  <version>1.0-SNAPSHOT</version>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
  <dependencies>
    <dependency>
      <groupId>com.aliyun.fc.runtime</groupId>
      <artifactId>fc-java-core</artifactId>
      <version>1.0.0</version>
    </dependency>
  </dependencies>
</project>
```

- 使用StreamRequestHandler handler。

使用该handler，需要将输入的 `InputStream` 转换为对应的 `POJO` 类，示例代码如下。
*pom.xml*文件配置与使用PojoRequestHandler<I, O> handler相同。

```
import com.aliyun.fc.runtime.Context;
import com.aliyun.fc.runtime.StreamRequestHandler;
import com.aliyun.fc.runtime.Context;
import com.google.gson.Gson;
import java.io.*;
import java.util.Base64;
import java.util.HashMap;
import java.util.Map;
public class ApiTriggerDemo2 implements StreamRequestHandler {
  public void handleRequest(InputStream inputStream, OutputStream outputStream, Context context) {
    try {
      // 将InputStream转化成字符串。
      BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(inputStream));
      StringBuffer stringBuffer = new StringBuffer();
      String string = "";
      while ((string = bufferedReader.readLine()) != null) {
```

```

        stringBuffer.append(string);
    }
    String input = stringBuffer.toString();
    context.getLogger().info("inputStream: " + input);
    Request req = new Gson().fromJson(input, Request.class);
    context.getLogger().info("input req: ");
    context.getLogger().info(req.toString());
    String bodyReq = req.getBody();
    Base64.Decoder decoder = Base64.getDecoder();
    context.getLogger().info("body: " + new String(decoder.decode(bodyReq)));
    // 您可以在这里处理您自己的逻辑。
    // 返回结构。
    Map headers = new HashMap();
    headers.put("x-custom-header", " ");
    boolean isBase64Encoded = false;
    int statusCode = 200;
    Map body = new HashMap();
    Response resp = new Response(headers, isBase64Encoded, statusCode, body);
    String respJson = new Gson().toJson(resp);
    context.getLogger().info("outputStream: " + respJson);
    outputStream.write(respJson.getBytes());
} catch (IOException e) {
    e.printStackTrace();
} finally {
    try {
        outputStream.close();
        inputStream.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}
}

class Request {
    private String path;
    private String httpMethod;
    private Map headers;
    private Map queryParameters;
    private Map pathParameters;
    private String body;
    private boolean isBase64Encoded;
    @Override
    public String toString() {
        return "Request{" +
            "path=" + path + "\" +
            ", httpMethod=" + httpMethod + "\" +
            ", headers=" + headers +
            ", queryParameters=" + queryParameters +
            ", pathParameters=" + pathParameters +
            ", body=" + body + "\" +
            ", isBase64Encoded=" + isBase64Encoded +
            "}";
    }
    public String getBody() {
        return body;
    }
}

```

```
    }  
    }  
    // 函数计算需要以以下JSON格式返回对API网关的响应。  
    class Response {  
        private Map headers;  
        private boolean isBase64Encoded;  
        private int statusCode;  
        private Map body;  
        public Response(Map headers, boolean isBase64Encoded, int statusCode, Map body) {  
            this.headers = headers;  
            this.isBase64Encoded = isBase64Encoded;  
            this.statusCode = statusCode;  
            this.body = body;  
        }  
    }  
}
```

测试函数

完成函数编写后，您需要调试函数以验证代码的正确性。API网关触发函数执行时，API网关的信息以event的形式作为输入参数传给函数，您可以将API网关传入的event信息作为参数，调试函数代码编写是否正确。

1. 在函数详情页面，单击**函数代码**页签，然后单击图标，从下拉列表中，选择**配置测试参数**。
2. 在**配置测试参数**对话框，选择**创建新测试事件**或**编辑已有测试事件**页签，填写事件名称和事件内容。然后单击**确定**。

event格式示例如下所示：

```
{  
  "path": "api request path",  
  "httpMethod": "request method name",  
  "headers": {all headers, including system headers},  
  "queryParameters": {query parameters},  
  "pathParameters": {path parameters},  
  "body": "string of request payload",  
  "isBase64Encoded": "true|false, indicate if the body is Base64-encode"  
}
```

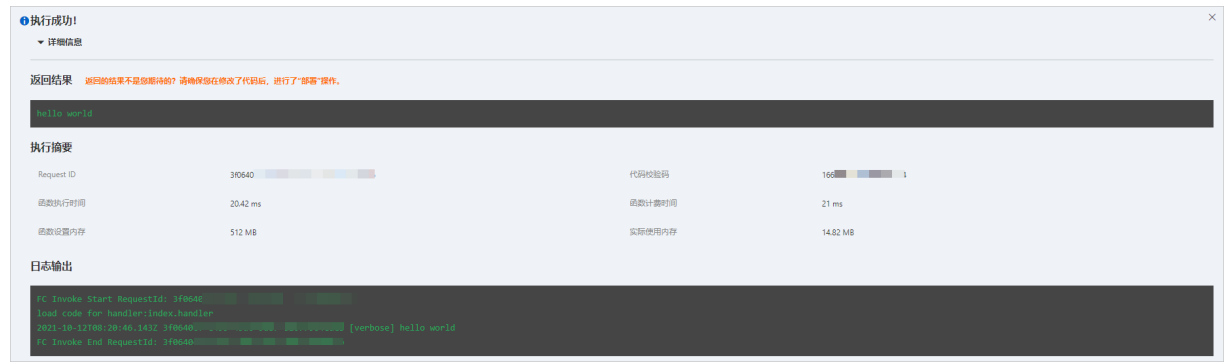
说明

- 如果 `isBase64Encoded` 的值为 `true`，表示API网关传给函数计算的body内容已Base64编码。函数计算需要先对body内容Base64解码后再处理。
- 如果 `isBase64Encoded` 的值为 `false`，表示API网关没有对body内容Base64编码，在函数中可以直接获取body内容。

3. 在函数详情页面，单击**函数代码**页签，然后单击**测试函数**。

验证结果

在**函数代码**页签，您可以看到执行成功的指示。



函数计算需要将执行的结果按照以下JSON格式返回给API网关，然后由API网关解析。返回格式示例如下：

```
{
  "isBase64Encoded":true|false,
  "statusCode":httpStatusCode,
  "headers":{response headers},
  "body":"..."
}
```

说明

- 如果函数计算返回给API网关的结果不符合格式要求，API网关会返回503 Service Unavailable。
- 当body内容为二进制时，需对其进行Base64编码，即isBase64Encoded设置为 *true*；当body内容为非二进制时，无需对其进行Base64编码，即isBase64Encoded设置为 *false*。

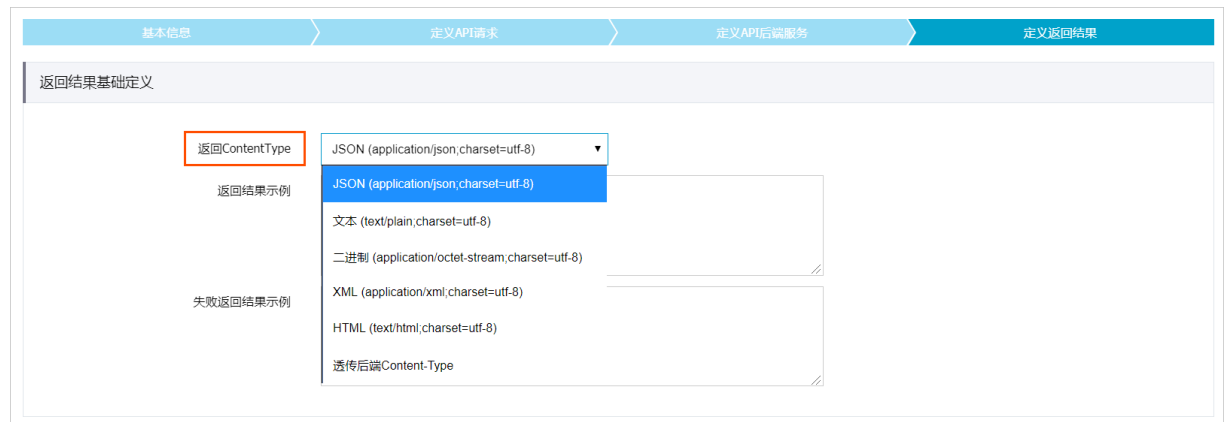
常见问题

API网关触发函数执行时报503，查看函数日志，函数已经执行成功了，这是怎么回事？

API网关和函数计算的对接有格式要求，如果函数计算返回给API网关的结果没有按规定的格式返回，那么API网关就认为后端服务不可用。关于API网关和函数计算的对接格式要求，请参见[event格式说明](#)和[函数计算的返回参数格式](#)。

如何设置返回响应的content-type？

如图所示，在API设置的时候可以设置响应的content-type。详细内容，请参见[创建后端为函数计算的API](#)。



API网关触发函数计算执行，已经调通的函数，一段时间不调用，再次调用会报503，这是什么原因？

一段时间不调用后，函数重新调用需要准备执行环境，有冷启动时延，在API网关设置的超时时间内没有调用完，API网关会认为后端服务不可用。延长API网关的超时时间即可解决问题。

为什么函数中接收到API网关传过来的body是经过Base64编码的？

API网关对FORM形式的body传输是不进行Base64编码的（使用FORM形式需要在API网关选择入参映射），其他形式body都会进行Base64编码，避免内容传输错误或者丢失。建议您在使用时，先判断event中isBase64是否为true。如果isBase64为true，则body需要在函数中进行解码。关于API网关传给函数计算的event格式，请参见[event格式说明](#)。

更多信息

除了函数计算控制台，您还可通过以下方式配置触发器：

- 通过Serverless Devs工具配置触发器。更多操作，请参见[Serverless Devs](#)。
- 通过SDK配置触发器。更多操作，请参见[SDK列表](#)。

如需对创建的触发器进行修改或删除，请参见[触发器管理](#)。

14.消息队列Kafka版

本文说明如何创建FC Sink Connector，您可以通过FC Sink Connector将数据从消息队列Kafka版实例的数据源Topic导出至函数计算的函数。

前提条件

在创建FC Sink Connector前，请确保您已完成以下操作：

- 为消息队列Kafka版实例开启Connector。更多信息，请参见[开启Connector](#)。
- 为消息队列Kafka版实例创建数据源Topic。更多信息，请参见[步骤一：创建Topic](#)。本文以名称为fc-test-input的Topic为例。
- 在函数计算创建函数。更多信息，请参见[使用控制台创建函数](#)。

 **注意** 函数类型必须为事件函数。

本文以服务名称为guide-hello_world、函数名称为hello_world、运行环境为Python的事件函数为例。该示例函数的代码如下：

```
# -*- coding: utf-8 -*-
import logging
# To enable the initializer feature
# please implement the initializer function as below:
# def initializer(context):
#     logger = logging.getLogger()
#     logger.info('initializing')
def handler(event, context):
    logger = logging.getLogger()
    logger.info('hello world:' + bytes.decode(event))
    return 'hello world:' + bytes.decode(event)
```

操作流程

使用FC Sink Connector将数据从消息队列Kafka版实例的数据源Topic导出至函数计算的函数的操作流程如下：

1. （可选）

 **注意** 如果您不需要使FC Sink Connector跨地域访问函数计算，您可以直接跳过该步骤。

2. （可选）使FC Sink Connector跨账号访问函数计算

 **注意** 如果您不需要使FC Sink Connector跨账号访问函数计算，您可以直接跳过该步骤。

- （可选）
- （可选）
- （可选）

3. （可选）创建FC Sink Connector依赖的Topic和Consumer Group

 注意

- 如果您不需要自定义Topic和Group的名称，您可以直接跳过该步骤。
- 部分FC Sink Connector依赖的Topic的存储引擎必须为Local存储，大版本为0.10.2版本的消息队列Kafka版实例不支持手动创建Local存储的Topic，只支持自动创建。

- (可选)
- (可选)

4.

5. 结果验证

-
-

(可选)

为FC Sink Connector开启公网访问

如需使FC Sink Connector跨地域访问其他阿里云服务，您需要为FC Sink Connector开启公网访问。具体操作，请参见[为Connector开启公网访问](#)。

(可选)

创建自定义权限策略

在目标账号下创建访问函数计算的自定义权限策略。

1. 登录[访问控制控制台](#)。
2. 在左侧导航栏，选择**权限管理 > 权限策略管理**。
3. 在**权限策略管理**页面，单击**创建权限策略**。
4. 在**新建自定义权限策略**页面，创建自定义权限策略。
 - i. 在**策略名称**文本框，输入 `KafkaConnectorFcAccess`。
 - ii. 在**配置模式**区域，选择**脚本配置**。
 - iii. 在**策略内容**区域，输入自定义权限策略脚本。

访问函数计算的自定义权限策略脚本示例如下：

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "fc:InvokeFunction",
        "fc:GetFunction"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

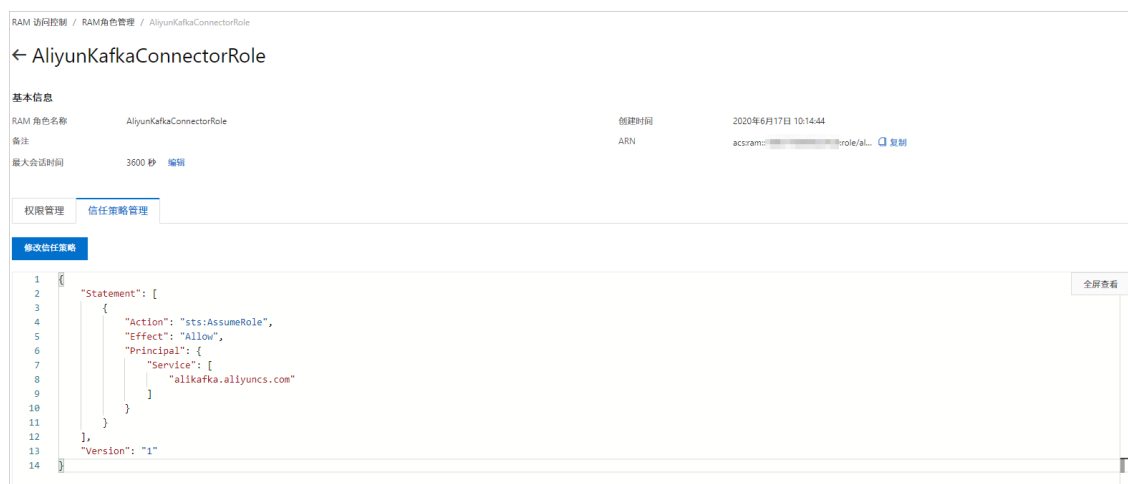
- iv. 单击**确定**。

(可选)

创建RAM角色

在目标账号下创建RAM角色。由于RAM角色不支持直接选择消息队列Kafka版作为受信服务，您在创建RAM角色时，需要选择任意支持的服务作为受信服务。RAM角色创建后，手工修改信任策略。

1. 在左侧导航栏，单击**RAM角色管理**。
2. 在**RAM角色管理**页面，单击**创建RAM角色**。
3. 在**创建RAM角色**面板，创建RAM角色。
 - i. 在当前可信实体类型区域，选择**阿里云服务**，单击**下一步**。
 - ii. 在角色类型区域，选择**普通服务角色**，在角色名称文本框，输入**AliyunKafkaConnectorRole**，从选择受信服务列表，选择**函数计算**，然后单击**完成**。
4. 在**RAM角色管理**页面，找到**AliyunKafkaConnectorRole**，单击**AliyunKafkaConnectorRole**。
5. 在**AliyunKafkaConnectorRole**页面，单击**信任策略管理**页签，单击**修改信任策略**。
6. 在**修改信任策略**面板，将脚本中**fc**替换为**alikafka**，单击**确定**。



(可选)

添加权限

在目标账号下为创建的RAM角色授予访问函数计算的权限。

1. 在左侧导航栏，单击**RAM角色管理**。
2. 在**RAM角色管理**页面，找到**AliyunKafkaConnectorRole**，在其右侧操作列，单击**添加权限**。
3. 在**添加权限**面板，添加**KafkaConnectorFcAccess**权限。
 - i. 在选择权限区域，选择**自定义策略**。
 - ii. 在**权限策略名称**列表，找到**KafkaConnectorFcAccess**，单击**KafkaConnectorFcAccess**。
 - iii. 单击**确定**。
 - iv. 单击**完成**。

(可选)

创建FC Sink Connector依赖的Topic

您可以在消息队列Kafka版控制台手动创建FC Sink Connector依赖的5个Topic，包括：任务位点Topic、任务配置Topic、任务状态Topic、死信队列Topic以及异常数据Topic。每个Topic所需要满足的分区数与存储引擎会有差异，具体信息，请参见。

1. 登录**消息队列Kafka版控制台**。

2. 在概览页面的资源分布区域，选择地域。

 **注意** Topic需要在应用程序所在的地域（即所部署的ECS的所在地域）进行创建。Topic不能跨地域使用。例如Topic创建在华北2（北京）这个地域，那么消息生产端和消费端也必须运行在华北2（北京）的ECS。

- 3. 在实例列表页面，单击目标实例名称。
- 4. 在左侧导航栏，单击Topic 管理。
- 5. 在Topic 管理页面，单击创建 Topic。
- 6. 在创建 Topic面板，设置Topic属性，然后单击确定。

创建 Topic

*

名称

demo

长度限制为 3 ~ 64 个字符，只能包含英文、数字、短横线 (-) 以及下划线 (_)，且至少包含一个英文或数字。

*

描述

demo test

9/64

*

分区数

12

建议分区数是6的倍数，减少数据倾斜风险，分区数限制（1~360），特殊需求请提交工单。

存储引擎

云存储

Local 存储

消息类型

普通消息

标签

demo

底层接入阿里云云盘，具有低时延、高性能、持久性、高可靠等特点，采用分布式3副本机制。

默认情况下，保证相同 Key 的消息分布在同一个分区中，且分区内消息按照发送顺序存储。集群中出现机器宕机时，可能会造成消息乱序。

参数	说明	示例
名称	Topic名称。	demo
描述	Topic的简单描述。	demo test
分区数	Topic的分区数量。	12

97

> 文档版本：20211110

参数	说明	示例
存储引擎	Topic消息的存储引擎。 消息队列Kafka版支持以下两种存储引擎。 ◦ 云存储：底层接入阿里云云盘，具有低时延、高性能、持久性、高可靠等特点，采用分布式3副本机制。实例的规格类型为标准版（高写版）时，存储引擎只能为云存储。 ◦ Local 存储：使用原生Kafka的ISR复制算法，采用分布式3副本机制。	云存储
消息类型	Topic消息的类型。 ◦ 普通消息：默认情况下，保证相同Key的消息分布在同一个分区中，且分区内消息按照发送顺序存储。集群中出现机器宕机时，可能会造成消息乱序。当存储引擎选择云存储时，默认选择普通消息。 ◦ 分区顺序消息：默认情况下，保证相同Key的消息分布在同一个分区中，且分区内消息按照发送顺序存储。集群中出现机器宕机时，仍然保证分区内按照发送顺序存储。但是会出现部分分区发送消息失败，等到分区恢复后即可恢复正常。当存储引擎选择Local 存储时，默认选择分区顺序消息。	普通消息

参数	说明	示例
日志清理策略	Topic日志的清理策略。 当存储引擎选择Local 存储时，需要配置日志清理策略。 消息队列Kafka版支持以下两种日志清理策略。 ◦ Delete：默认的消息清理策略。在磁盘容量充足的情况下，保留在最长保留时间范围内的消息；在磁盘容量不足时（一般磁盘使用率超过85%视为不足），将提前删除旧消息，以保证服务可用性。 ◦ Compact：使用Kafka Log Compaction日志清理策略。Log Compaction清理策略保证相同Key的消息，最新的value值一定会被保留。主要适用于系统宕机后恢复状态，系统重启后重新加载缓存等场景。例如，在使用Kafka Connect或Confluent Schema Registry时，需要使用Kafka Compact Topic存储系统状态信息或配置信息。  注意 Compact Topic一般只用在某些生态组件中，例如Kafka Connect或Confluent Schema Registry，其他情况的消息收发请勿为Topic设置该属性。具体信息，请参见消息队列Kafka版Demo库。	Compact
标签	Topic的标签。	demo

创建完成后，在Topic 管理页面的列表中显示已创建的Topic。

（可选）

创建FC Sink Connector依赖的Group

您可以在消息队列Kafka版控制台手动创建FC Sink Connector数据同步任务使用的Group。该Group的名称必须为connect-任务名称，具体信息，请参见。

1. 登录[消息队列Kafka版控制台](#)。
2. 在概览页面的资源分布区域，选择地域。
3. 在实例列表页面，单击目标实例名称。
4. 在左侧导航栏，单击Group 管理。
5. 在Group 管理页面，单击创建 Group。

6. 在创建 Group 面板的Group ID文本框输入Group的名称，在描述文本框简要描述Group，并给Group添加标签，单击确定。
创建完成后，在Group 管理页面的列表中显示已创建的Group。

创建并部署FC Sink Connector

创建并部署用于将数据从消息队列Kafka版同步至函数计算的FC Sink Connector。

- 1. 登录消息队列Kafka版控制台。
- 2. 在概览页面的资源分布区域，选择地域。
- 3. 在实例列表页面，单击目标实例名称。
- 4. 在左侧导航栏，单击Connector 管理。
- 5. 在Connector 管理页面，单击创建 Connector。
- 6. 在创建Connector配置向导页面，完成以下操作。
 - i. 在配置基本信息页签，按需配置以下参数，然后单击下一步。

参数	描述	示例值
名称	Connector的名称。命名规则： - 可以包含数字、小写英文字母和短划线（-），但不能以短划线（-）开头，长度限制为48个字符。 - 同一个消息队列Kafka版实例内保持唯一。 Connector的数据同步任务必须使用名称为connect-任务名称的Group。如果您未手动创建该Group，系统将为您自动创建。	kafka-fc-sink
实例	默认配置为实例的名称与实例ID。	demo alikafka_post-cn-st21p8vj****

- ii. 在配置源服务页签，选择数据源为消息队列Kafka版，并配置以下参数，然后单击下一步。

 说明 如果您已创建好Topic和Group，那么请选择手动创建资源，并填写已创建的资源信息。否则，请选择自动创建资源。

配置源服务参数列表

参数	描述	示例值
数据源 Topic	需要同步数据的Topic。	fc-test-input
消费线程并发数	数据源Topic的消费线程并发数。默认值为6。取值说明如下： 6 12	6
消费初始位置	开始消费的位置。取值说明如下： - 最早位点：从最初位点开始消费。 - 最近位点：从最新位点开始消费。	最早位点

参数	描述	示例值
VPC ID	数据同步任务所在的VPC。单击 配置运行环境 显示该参数。默认为消息队列Kafka版实例所在的VPC，您无需填写。	vpc-bp1xpdnd3l***
VSwitch ID	数据同步任务所在的交换机。单击 配置运行环境 显示该参数。该交换机必须与消息队列Kafka版实例处于同一VPC。默认为部署消息队列Kafka版实例时填写的交换机。	vsw-bp1d2jgg81***
失败处理	消息发送失败后，是否继续订阅出现错误的Topic的分区。单击配置运行环境显示该参数。取值说明如下。 ■ 继续订阅：继续订阅出现错误的Topic的分区，并打印错误日志。 ■ 停止订阅：停止订阅出现错误的Topic的分区，并打印错误日志 ② 说明 ■ 如何查看日志，请参见查看Connector日志。 ■ 如何根据错误码查找解决方案，请参见错误码。 ■ 如需恢复对出现错误的Topic的分区的订阅，您需要提交工单联系消息队列Kafka版技术人员。	继续订阅
创建资源方式	选择创建Connector所依赖的Topic与Group的方式。单击配置运行环境显示该参数。 ■ 自动创建 ■ 手动创建	自动创建
Connector 消费组	Connector的数据同步任务使用的Group。单击 配置运行环境 显示该参数。该Group的名称必须为connect-任务名称。	connect-kafka-fc-sink
任务位点 Topic	用于存储消费位点的Topic。单击配置运行环境显示该参数。 ■ Topic：建议以connect-offset开头。 ■ 分区数：Topic的分区数量必须大于1。 ■ 存储引擎：Topic的存储引擎必须为Local存储。 ■ cleanup.policy：Topic的日志清理策略必须为compact。	connect-offset-kafka-fc-sink
任务配置 Topic	用于存储任务配置的Topic。单击配置运行环境显示该参数。 ■ Topic：建议以connect-config开头。 ■ 分区数：Topic的分区数量必须为1。 ■ 存储引擎：Topic的存储引擎必须为Local存储。 ■ cleanup.policy：Topic的日志清理策略必须为compact。	connect-config-kafka-fc-sink

参数	描述	示例值
任务状态 Topic	用于存储任务状态的Topic。单击配置运行环境显示该参数。 - Topic：建议以connect-status开头。 - 分区数：Topic的分区数量建议为6。 - 存储引擎：Topic的存储引擎必须为Local存储。 - cleanup.policy：Topic的日志清理策略必须为compact。	connect-status-kafka-fc-sink
死信队列 Topic	用于存储Connect框架的异常数据的Topic。单击配置运行环境显示该参数。该Topic可以和异常数据Topic为同一个Topic，以节省Topic资源。 - Topic：建议以connect-error开头。 - 分区数：Topic的分区数量建议为6。 - 存储引擎：Topic的存储引擎可以为Local存储或云存储。	connect-error-kafka-fc-sink
异常数据 Topic	用于存储Sink的异常数据的Topic。单击配置运行环境显示该参数。该Topic可以和死信队列Topic为同一个Topic，以节省Topic资源。 - Topic：建议以connect-error开头。 - 分区数：Topic的分区数量建议为6。 - 存储引擎：Topic的存储引擎可以为Local存储或云存储。	connect-error-kafka-fc-sink

iii. 在**配置目标服务**页签，选择目标服务为函数计算，并配置以下参数，然后单击**创建**。

参数	描述	示例值
是否跨账号/地域	FC Sink Connector是否跨账号/地域向函数计算服务同步数据。默认为否。取值： - 否：同地域同账号模式。 - 是：跨地域同账号模式、同地域跨账号模式或跨地域跨账号模式。	否
服务地域	函数计算服务的地域。默认为FC Sink Connector所在地域。如需跨地域，您需要为Connector开启公网访问，然后选择目标地域。更多信息，请参见FC Sink Connector开启公网访问。  注意 是否跨账号/地域为是时，显示服务地域。	cn-hangzhou

参数	描述	示例值
服务接入点	函数计算服务的接入点。在函数计算控制台的概览页的常用信息区域获取。 - 内网Endpoint：低延迟，推荐。适用于消息队列Kafka版实例和函数计算处于同一地域场景。 - 公网Endpoint：高延迟，不推荐。适用于消息队列Kafka版实例和函数计算处于不同地域的场景。如需使用公网Endpoint，您需要为Connector开启公网访问。更多信息，请参见为FC Sink Connector开启公网访问。  注意 是否跨账号/地域为是时，显示服务接入点。	http://188***.cn-hangzhou.fc.aliyuncs.com
服务账号	函数计算服务的阿里云账号ID。在函数计算控制台的概览页的常用信息区域获取。  注意 是否跨账号/地域为是时，显示服务账号。	188***
授权角色名	消息队列Kafka版访问函数计算服务的RAM角色。 - 如不需跨账号，您需要在本账号下创建RAM角色并为其授权，然后输入该授权角色名。操作步骤，请参见和。 - 如需跨账号，您需要在目标账号下创建RAM角色并为其授权，然后输入该授权角色名。操作步骤，请参见和。  注意 是否跨账号/地域为是时，显示授权角色名。	AliyunKafkaConnectorRole
服务名	函数计算服务的名称。	guide-hello_world
函数名	函数计算服务的函数名称。	hello_world
版本或别名	函数计算服务的版本或别名。	LATEST
发送模式	消息发送模式。取值： 异步：推荐。 同步：不推荐。同步发送模式下，如果函数计算的处理时间较长，消息队列Kafka版的处理时间也会较长。当同一批次消息的处理时间超过5分钟时，会触发消息队列Kafka版客户端Rebalance。	异步
	批量发送的消息条数。默认为20。Connector根据发送批次大小和请求大小限制（同步请求大小限制为6 MB，异步请求大小限制为128 KB）将多条消息聚合后发送。例如，发送模式为异步，发送批次大小为20，如果要发送18条消息，其中有17条消息的总大小为127 KB，有1条消息的大小为200 KB，Connector会将总大小不超过128 KB的17条消息聚合后发送，将大小超过	

参数	128 KB的1条消息单独发送。 描述	示例值
发送批大小	② 说明 如果您在发送消息时将key设置为null，则请求中不包含key。如果将value设置为null，则请求中不包含value。 ■ 如果批量发送的多条消息的大小不超过请求大小限制，则请求中包含消息内容。请求示例如下： <pre>[{ "key":"this is the message's key2", "offset":8, "overflowFlag":false, "partition":4, "timestamp":1603785325438, "topic":"Test", "value":"this is the message's value2", "valueSize":28 }, { "key":"this is the message's key9", "offset":9, "overflowFlag":false, "partition":4, "timestamp":1603785325440, "topic":"Test", "value":"this is the message's value9", "valueSize":28 }, { "key":"this is the message's key12", "offset":10, "overflowFlag":false, "partition":4, "timestamp":1603785325442, "topic":"Test", "value":"this is the message's value12", "valueSize":29 }, { "key":"this is the message's key38", "offset":11, "overflowFlag":false, "partition":4, "timestamp":1603785325464, "topic":"Test", "value":"this is the message's value38", "valueSize":29 }]</pre>	50

参数	描述	示例值
	■ 如果发送的单条消息的大小超过请求大小限制，则请求中不包含消息内容。请求示例如下： <pre>[{ "key": "123", "offset": 4, "overflowFlag": true, "partition": 0, "timestamp": 1603779578478, "topic": "Test", "value": "1", "valueSize": 272687 }]</pre>  说明 如需获取消息内容，您需要根据位点主动拉取消息。	
重试次数	消息发送失败后的重试次数。默认为2。取值范围为1~3。部分导致消息发送失败的错误不支持重试。错误码与是否支持重试的对应关系如下： ■ 4XX：除429支持重试外，其余错误码不支持重试。 ■ 5XX：支持重试。  说明 Connector调用InvokeFunction向函数计算发送消息。关于InvokeFunction的更多信息，请参见API概览。	2

创建完成后，在**Connector 管理**页面，查看创建的Connector。

7. 创建完成后，在**Connector 管理**页面，找到创建的Connector，单击其操作列的部署。

如需配置函数计算资源，单击其操作列的**更多 > 配置函数**，跳转至函数计算控制台完成操作。

发送测试消息

部署FC Sink Connector后，您可以向消息队列Kafka版的数据源Topic发送消息，测试数据能否被同步至函数计算。

1. 在**Connector 管理**页面，找到目标Connector，在其右侧操作列，单击**测试**。

2. 在发送消息面板，发送测试消息。

○ **发送方式选择控制台。**

a. 在消息 **Key**文本框中输入消息的Key值，例如demo。

b. 在消息内容文本框输入测试的消息内容，例如 {"key": "test"}。

c. 设置发送到指定分区，选择是否指定分区。

■ 单击是，在分区 ID文本框中输入分区的ID，例如0。如果您需查询分区的ID，请参见[查看分区状态](#)。

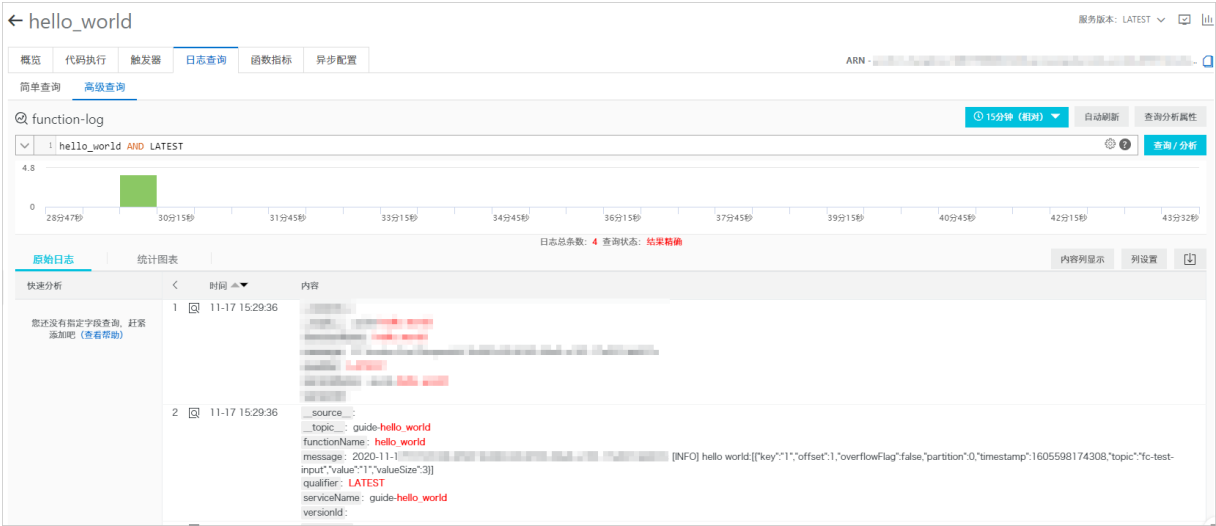
■ 单击否，不指定分区。

- 发送方式选择 Docker，执行运行 Docker 容器生产示例消息区域的 Docker 命令，发送消息。
- 发送方式选择 SDK，根据您的业务需求，选择需要的语言或者框架的 SDK 以及接入方式，通过 SDK 发送消息。

查看函数日志

向消息队列Kafka版的数据源Topic发送消息后，查看函数日志，验证是否收到消息。更多信息，请参见[配置日志](#)。

日志中显示发送的测试消息。



15.EventBridge触发器

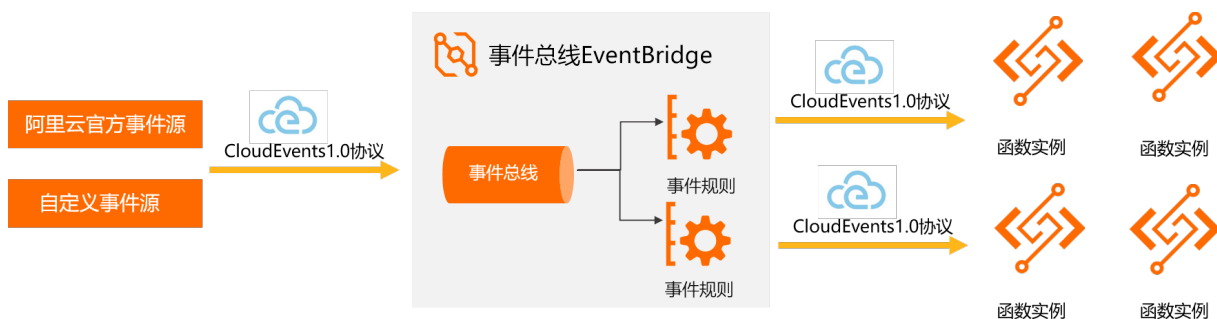
15.1. EventBridge触发器简介

事件总线EventBridge作为一种Serverless的事件总线服务，支持包括弹性计算、存储服务、数据库、容器、大数据处理、可观测性服务及中间件服务在内的几乎所有阿里云官方事件源，以及多种自定义事件源。事件总线EventBridge与函数计算集成后，接入事件总线EventBridge的事件源能够触发相关函数执行，完成对阿里云官方事件源及自定义事件源相关事件的处理。

什么是EventBridge触发器

在事件驱动模型中，事件源是事件的生产者，函数是事件的处理者，而触发器提供了一种集中、统一的方式来管理不同的事件源。事件总线EventBridge不是单一的事件源，而是作为阿里云官方事件源和自定义事件源的事件中心。EventBridge触发器支持阿里云官方事件源和自定义事件源作为触发源触发相关函数执行，实现事件总线EventBridge与函数计算的无缝集成。

EventBridge触发器的原理如下图所示。



优势

事件总线EventBridge与函数计算集成之后，事件总线EventBridge和函数计算的触发源将会同步。当事件总线EventBridge侧增加了事件源后，函数计算同步更新触发器的触发源，降低后续对接成本。

EventBridge触发器的类型

EventBridge触发器包括RocketMQ触发器、RabbitMQ触发器、MNS队列触发器和官方事件源触发器。其中官方事件源触发器是云监控触发器、云服务器ECS触发器、阿里云物联网IoT触发器等多种云服务类型的触发器的统称。

官方事件源触发器对应的所有阿里云服务及其事件类型如下所示。

弹性计算

存储

数据库

安全

大数据

人工智能

网络和CDN

视频服务

容器和中间件

开发和运维

域名和网站

物联网

企业应用与服务

15.2. RocketMQ触发器

消息队列RocketMQ版

作为事件源通过事件总线EventBridge与函数计算集成后，通过

消息队列RocketMQ版

触发器（以下简称RocketMQ触发器）能够触发关联函数执行，通过函数可以对发布到

消息队列RocketMQ版

中的消息进行自定义处理。本文介绍如何在函数计算控制台创建RocketMQ触发器、编写函数和测试函数。

功能简介

您在函数计算的控制台提交触发器创建请求之后，函数计算根据触发器的配置信息，将自动创建如下与事件总线EventBridge相关的资源：

- 事件总线RocketMQ-函数名称-触发器名称
- 事件源RocketMQ-函数名称-触发器名称
- 事件规则服务名称-函数名称-触发器名称

创建完成后，您可以在函数计算控制台查看触发器信息，同时也可以事件总线EventBridge控制台查看自动创建的资源信息。当事件源指定类型的事件投递到事件总线时，将触发该触发器关联的函数执行一次。

 **说明** 自动创建的事件总线EventBridge相关资源的名称必须符合名称取值限制和要求，否则触发器无法创建成功。具体信息，请参见[使用限制](#)。

注意事项

- 事件总线EventBridge限制同一地域下自定义事件总线不能超过10个，每个自定义总线上创建的事件规则总数也不能超过10个。超过以上限制后，将无法再创建EventBridge触发器。
- 创建的
消息队列RocketMQ版
的实例和函数计算的函数必须在相同的地域。
- RocketMQ触发器暂时不支持通过Serverless Devs工具或SDK创建。

前提条件

- 事件总线EventBridge
 - [开通事件总线EventBridge并授权](#)
- 函数计算
 - [创建服务](#)
 - [创建函数](#)
- 消息队列RocketMQ版
 - [开通消息队列服务并授权](#)
 - [创建实例、Topic和Group ID](#)

步骤一：创建触发器

1. 登录[函数计算控制台](#)。
2. 在左侧导航栏，单击[服务及函数](#)。
3. 在顶部菜单栏，选择地域。
4. 在[服务列表](#)页面，找到目标服务，在其右侧操作列单击[函数管理](#)。
5. 在[函数管理](#)页面，单击目标函数名称。



6. 在函数详情页面，单击[触发器管理](#)页签，从版本或别名下拉列表选择要创建触发器的版本或别名，然后单击[创建触发器](#)。



7. 在创建触发器面板，填写相关信息。然后单击[确定](#)。

在函数 Function 中创建触发器

* 触发器类型



消息队列 RocketMQ 版

同步调用

▼

* 名称

rocketmq-trigger

16/128

只能包含字母、数字、下划线和中划线。不能以数字、中划线开头。长度在 1-128 之间。

版本或别名 ?

LATEST

* RocketMQ 实例

MQ_INST_164901546557****_BXkhFHryi

▼

* Topic

Mytopic

▼

Tag

RocketMQ中用于过滤消息的Tag

* Group ID ?

GID_group1

▼

消费位点

最新位点

最早位点

指定时间戳

确定

取消

参数	操作	本文示例
触发器类型	选择消息队列 RocketMQ 版。	消息队列RocketMQ版
名称	填写自定义的触发器名称。	rocketmq-trigger
版本或别名	默认值为 LATEST ，如果您需要创建其他版本或别名的触发器，首先您需要在函数详情页的右上角切换到该版本或别名。关于版本的简介，请参见 版本简介 。	LATEST
RocketMQ 实例	选择已创建的消息队列RocketMQ版的实例。	MQ_INST_164901546557****_BXhFHryi
Topic	选择已创建的消息队列RocketMQ版实例的Topic。	Mytopic
过滤标签	填写消息过滤标签。 只有收到包含此处设置的过滤标签字符串的消息时，才会触发函数执行。	tag

参数	操作	本文示例
Group ID	选择已创建的消息队列RocketMQ版实例的Group ID。	GID_group1
消费位点	选择消息的消费位点，即消息队列RocketMQ版从事件总线开始拉取消息的位置。取值说明如下： - 最新位点：从最新位点开始消费。 - 最早位点：从最早位点开始消费。 - 指定时间戳：从指定时间戳开始消费。	最新位点

创建完成后，在触发器名称列表中显示已创建的触发器。



步骤二：编写函数

完成创建RocketMQ触发器之后，您可以编写函数代码。

- 1. 在函数详情页面，单击函数代码页签，在代码编辑器中编写代码。

代码示例如下。

```
'use strict';
/*
To enable the initializer feature
please implement the initializer function as below:
exports.initializer = (context, callback) => {
  console.log('initializing');
  callback(null, '');
};
*/
exports.handler = (event, context, callback) => {
  console.log("event: %s", event);
  handle_event(event);
  callback(null, 'return result');
}
```

步骤三：测试函数

完成函数编写后，您需要调试函数以验证代码的正确性。在实际操作过程中，当

消息队列RocketMQ版

事件通过事件总线Event Bridge投递到函数计算时，会自动触发函数执行。

消息队列RocketMQ版

事件源会以event的形式作为输入参数传递给函数，您可以手动将event传给函数模拟触发事件，测试函数代码是否正确。

- 1. 在函数详情页面，单击**函数代码**页签，然后单击 图标，从下拉列表中，选择**配置测试参数**。
- 2. 在**配置测试参数**对话框，选择**创建新测试事件**或**编辑已有测试事件**页签，填写事件名称和事件内容。然后单击**确定**。

event是函数计算的入口参数，具体格式如下所示。

```
{
  "id": "94ebc15f-f0db-4bbe-acce-56fb72fb****",
  "source": "RocketMQ-Function-rocketmq-trigger",
  "specversion": "1.0",
  "type": "mq:Topic:SendMessage",
  "datacontenttype": "application/json; charset=utf-8",
  "subject": "acs:mq:cn-hangzhou:164901546557****:MQ_INST_164901546557****_BXhFHryi%TopicName",
  "time": "2021-04-08T06:01:20.766Z",
  "aliyunaccountid": "164901546557****",
  "aliyunpublishtime": "2021-10-15T02:05:16.791Z",
  "aliyunoriginalaccountid": "164901546557****",
  "aliyuneventbusname": "RocketMQ-Function-rocketmq-trigger",
  "aliyunregionid": "cn-chengdu",
  "aliyunpublishaddr": "42.120.XX.XX",
  "data": {
    "topic": "TopicName",
    "systemProperties": {
      "MIN_OFFSET": "0",
      "TRACE_ON": "true",
      "MAX_OFFSET": "8",
      "MSG_REGION": "cn-hangzhou",
      "KEYS": "systemProperties.KEYS",
      "CONSUME_START_TIME": 1628577790396,
      "TAGS": "systemProperties.TAGS",
      "INSTANCE_ID": "MQ_INST_164901546557****_BXhFHryi"
    },
    "userProperties": {
    },
    "body": "TEST"
  }
}
```

data字段包含的参数解释如下表所示。关于CloudEvents规范中定义的参数解释，请参见[事件概述](#)。

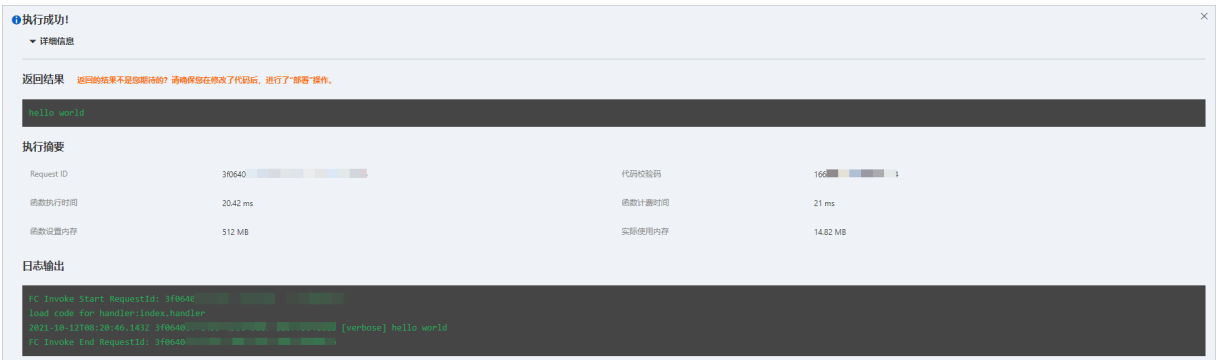
参数	类型	示例值	描述
topic	String	TopicName	Topic名称。
systemProperties	Map		系统属性。
MIN_OFFSET	Int	0	最低位点。

参数	类型	示例值	描述
TRACE_ON	Boolean	true	是否有消息轨迹。取值说明如下： <code>true</code>：有消息轨迹。 <code>false</code>：无消息轨迹。
MAX_OFFSET	Int	8	最高位点。
MSG_REGION	String	cn-hangzhou	发送消息的地域。
KEYS	String	systemProperties.KEYS	过滤属性。
CONSUME_START_TIME	Long	1628577790396	开始消费时间。单位：毫秒。
UNIQ_KEY	String	AC14C305069E1B28CDF A3181CDA2****	消息唯一键。
TAGS	String	systemProperties.TAGS	过滤属性。
INSTANCE_ID	String	MQ_INST_12345678909 8****_BXhFHryi	实例ID。
userProperties	Map	无	用户属性。
body	String	TEST	消息内容。

3. 在函数详情页面，单击**函数代码**页签，然后单击**测试函数**。

验证结果

在**函数代码**页签，您可以看到执行成功的指示。



更多信息

如需对创建的触发器进行修改或删除，具体操作，请参见[触发器管理](#)。

15.3. RabbitMQ触发器

消息队列RabbitMQ版作为事件源通过事件总线EventBridge与函数计算集成后，通过消息队列RabbitMQ版触发器（以下简称RabbitMQ触发器）能够触发关联函数执行，通过函数可以对发布到消息队列RabbitMQ版中的消息进行自定义处理。本文介绍如何在函数计算控制台创建RabbitMQ触发器、编写函数和测试函数。

功能简介

您在函数计算的控制台提交触发器创建请求之后，函数计算根据触发器的配置信息，将自动创建如下与事件总线EventBridge相关的资源：

- 事件总线RabbitMQ-函数名称-触发器名称
- 事件源RabbitMQ-函数名称-触发器名称
- 事件规则服务名称-函数名称-触发器名称

创建完成后，您可以在函数计算控制台查看触发器信息，同时也可以事件总线EventBridge控制台查看自动创建的资源信息。当事件源指定类型的事件投递到事件总线时，将触发该触发器关联的函数执行一次。

 **说明** 自动创建的事件总线EventBridge相关资源的名称必须符合名称取值限制和要求，否则触发器无法创建成功。具体信息，请参见[使用限制](#)。

注意事项

- 事件总线EventBridge限制同一地域下自定义事件总线不能超过10个，每个自定义总线上创建的事件规则总数也不能超过10个。超过以上限制后，将无法再创建EventBridge触发器。
- 创建的消息队列RabbitMQ版的实例和函数计算的函数必须在相同的地域。
- RabbitMQ触发器暂时不支持通过Serverless Devs工具或SDK创建。

前提条件

- 事件总线EventBridge
 - [开通事件总线EventBridge并授权](#)
- 函数计算
 - [创建服务](#)
 - [创建函数](#)
- 消息队列RabbitMQ版
 - [开通消息队列RabbitMQ版服务](#)
 - [创建实例、Vhost和Queue](#)

步骤一：创建触发器

1. 登录[函数计算控制台](#)。
2. 在左侧导航栏，单击[服务及函数](#)。
3. 在顶部菜单栏，选择地域。
4. 在[服务列表](#)页面，找到目标服务，在其右侧操作列单击[函数管理](#)。
5. 在[函数管理](#)页面，单击目标函数名称。



6. 在函数详情页面，单击触发器管理页签，从版本或别名下拉列表选择要创建触发器的版本或别名，然后单击创建触发器。



7. 在创建触发器面板，填写相关信息。然后单击确定。

在函数 Function 中创建触发器

* 触发器类型

消息队列 RabbitMQ 版 同步调用

* 名称

rabbitmq-trigger16/128

只能包含字母、数字、下划线和中划线。不能以数字、中划线开头。长度在 1-128 之间。

版本或别名

LATEST

* RabbitMQ 实例

amqp-cn-tl32e756

* Vhost

rabbit-host

* Queue

rabbit-queue

确定

取消

参数	操作	本文示例
触发器类型	选择消息队列 RabbitMQ 版。	消息队列RabbitMQ版
名称	填写自定义的触发器名称。	rabbitmq-trigger

参数	操作	本文示例
版本或别名	默认值为LATEST，如果您需要创建其他版本或别名的触发器，首先您需要在函数详情页的右上角切换到该版本或别名。关于版本的简介，请参见 版本简介 。	LATEST
RabbitMQ 实例	选择已创建的消息队列RabbitMQ版的实例。	amqp-cn-tl32e756****
Vhost	选择已创建的消息队列RabbitMQ版实例的Vhost。	rabbit-host
Queue	选择已创建的消息队列RabbitMQ版实例的Queue。	rabbit-queue

创建完成后，在触发器名称列表中显示已创建的触发器。



步骤二：编写函数

完成创建RabbitMQ触发器之后，您可以编写函数代码。

- 1. 在函数详情页面，单击函数代码页签，在代码编辑器中编写代码。

代码示例如下。

```
'use strict';
/*
To enable the initializer feature
please implement the initializer function as below:
exports.initializer = (context, callback) => {
  console.log('initializing');
  callback(null, '');
};
*/
exports.handler = (event, context, callback) => {
  console.log("event: %s", event);
  handle_event(event);
  callback(null, 'return result');
}
```

步骤三：测试函数

完成函数编写后，您需要调试函数以验证代码的正确性。在实际操作过程中，当消息队列RabbitMQ版事件通过事件总线EventBridge投递到函数计算时，会自动触发函数执行。

消息队列RabbitMQ版事件源会以event的形式作为输入参数传递给函数，您可以手动将event传给函数模拟触发事件，测试函数代码是否正确。

- 1. 在函数详情页面，单击函数代码页签，然后单击 图标，从下拉列表中，选择配置测试参数。
- 2. 在配置测试参数对话框，选择创建新测试事件或编辑已有测试事件页签，填写事件名称和事件内容。然后单击确定。

event是函数计算的入口参数，具体格式如下所示。

```
{
  "id": "bj694332-4cj1-389e-9d8c-b137h30b****",
  "source": "RabbitMQ-Function-rabbitmq-trigger",
  "specversion": "1.0",
  "type": "amqp:Queue:SendMessage",
  "datacontenttype": "application/json;charset=utf-8",
  "subject": "acs:amqp:cn-hangzhou:164901546557****:/instances/amqp-cn-tl32e756****/vhosts/eb-connect/queues/housekeeping",
  "time": "2021-08-12T06:56:40.709Z",
  "aliyunaccountid": "164901546557****",
  "aliyunpublishtime": "2021-10-15T08:58:55.140Z",
  "aliyunoriginalaccountid": "164901546557****",
  "aliyuneventbusname": "RabbitMQ-Function-rabbitmq-trigger",
  "aliyunregionid": "cn-chengdu",
  "aliyunpublishaddr": "42.120.XX.XX",
  "data": {
    "envelope": {
      "deliveryTag": 98,
      "exchange": "",
      "redeliver": false,
      "routingKey": "housekeeping"
    },
    "body": {
      "Hello": "RabbitMQ"
    },
    "props": {
      "contentEncoding": "UTF-8",
      "messageId": "f7622d51-e198-41de-a072-77c1ead7****"
    }
  }
}
```

data字段包含的参数解释如下表所示。关于CloudEvents规范中定义的参数解释，请参见[事件概述](#)。

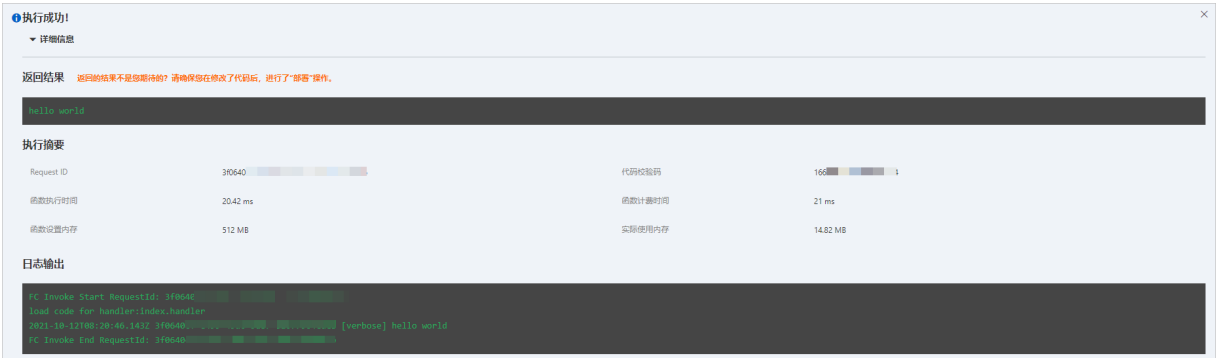
参数	类型	示例值	描述
body	Map		消息内容。
Hello	String	EventBridge	用户数据。
props	Map		消息属性。
contentEncoding	String	utf-8	消息内容编码。
messageId	String	f7622d51-e198-41de-a072-77c1ead7****	消息ID。每条消息的ID取值唯一。
envelope	Map		消息的envelope信息。
deliveryTag	Int	98	消息的Tag。
exchange	String	无	消息的Exchange。

参数	类型	示例值	描述
redeliver	Boolean	false	是否支持重发消息。取值说明如下： <code>true</code>：支持 <code>false</code>：不支持
routingKey	String	housekeeping	消息的路由规则。

3. 在函数详情页面，单击函数代码页签，然后单击测试函数。

验证结果

在函数代码页签，您可以看到执行成功的指示。



更多信息

如需对创建的触发器进行修改或删除，具体操作，请参见[触发器管理](#)。

15.4. MNS队列触发器

消息服务MNS的队列（Queue）作为事件源通过事件总线EventBridge与函数计算集成后，通过消息服务MNS队列触发器（以下简称MNS队列触发器）能够触发关联函数执行，通过函数可以对发布到消息服务MNS队列中的消息进行自定义处理。本文介绍如何在函数计算控制台创建MNS队列触发器、编写函数和测试函数。

功能简介

您在函数计算的控制台提交触发器创建请求之后，函数计算根据触发器的配置信息，将自动创建如下与事件总线EventBridge相关的资源：

- 事件总线MNS-函数名称-触发器名称
- 事件源MNS-函数名称-触发器名称
- 事件规则服务名称-函数名称-触发器名称

创建完成后，您可以在函数计算控制台查看触发器信息，同时也可以事件总线EventBridge控制台查看自动创建的资源信息。当事件源指定类型的事件投递到事件总线时，将触发该触发器关联的函数执行一次。

 **说明** 自动创建的事件总线EventBridge相关资源的名称必须符合名称取值限制和要求，否则触发器无法创建成功。具体信息，请参见[使用限制](#)。

注意事项

- 事件总线EventBridge限制同一地域下自定义事件总线不能超过10个，每个自定义总线上创建的事件规则总数也不能超过10个。超过以上限制的数量后，将无法再创建EventBridge触发器。
- 创建的消息服务MNS的队列和函数计算的函数必须在相同的地域。
- MNS触发器暂时不支持通过Serverless Devs工具或SDK创建。

前提条件

- 事件总线EventBridge
 - 开通事件总线EventBridge并授权
- 函数计算
 - 创建服务
 - 创建函数
- 消息服务MNS
 - 开通消息服务MNS并授权
 - 创建队列

步骤一：创建触发器

1. 登录[函数计算控制台](#)。
2. 在左侧导航栏，单击服务及函数。
3. 在顶部菜单栏，选择地域。
4. 在服务列表页面，找到目标服务，在其右侧操作列单击函数管理。
5. 在函数管理页面，单击目标函数名称。



6. 在函数详情页面，单击触发器管理页签，从版本或别名下拉列表选择要创建触发器的版本或别名，然后单击创建触发器。



7. 在创建触发器面板，填写相关信息。然后单击确定。

在函数 Function 中创建触发器

* 触发器类型

消息服务 MNS queue 触发 同步调用

* 名称

mns-queue-trigger17/128

只能包含字母、数字、下划线和中划线。不能以数字、中划线开头。长度在 1-128 之间。

版本或别名

LATEST

* 队列名称

Myqueue

确定

取消

参数	操作	本文示例
触发器类型	选择消息服务 MNS 。	消息服务MNS
名称	填写自定义的触发器名称。	mns-queue-trigger
版本或别名	默认值为 LATEST ，如果您需要创建其他版本或别名的触发器，首先您需要在函数详情页的右上角切换到该版本或别名。关于版本的简介，请参见 版本简介 。	LATEST
队列名称	选择已创建的消息服务MNS的队列。	Myqueue

创建完成后，在触发器名称列表中显示已创建的触发器。

← 函数 Function 详情

版本或别名 LATEST 监控大盘 设置报警 日志仓库 删除

函数概述

函数代码 触发器管理 函数配置 监控指标 调用日志 错误追踪 异步调用

创建触发器

可按触发器名称进行模糊搜索。

触发器名称	触发器类型	配置信息	状态	创建时间	操作
mns-trigger	MNS 同步调用	事件源 MNS	已启用	2021年10月14日	详情 编辑 禁用 更多

步骤二：编写函数

完成创建MNS队列触发器之后，您可以编写函数代码。

1. 在函数详情页面，单击**函数代码**页签，在代码编辑器中编写代码。

代码示例如下。

```
'use strict';
/*
To enable the initializer feature
please implement the initializer function as below:
exports.initializer = (context, callback) => {
  console.log('initializing');
  callback(null, '');
};
*/
exports.handler = (event, context, callback) => {
  console.log("event: %s", event);
  handle_event(event);
  callback(null, 'return result');
}
```

步骤三：测试函数

完成函数编写后，您需要调试函数以验证代码的正确性。在实际操作过程中，当消息服务MNS自定义事件源产生的事件通过事件总线Event Bridge投递到函数计算时，会自动触发函数执行。

消息服务MNS事件源会以event的形式作为输入参数传递给函数，您可以手动将event传给函数模拟触发事件，测试函数代码是否正确。

1. 在函数详情页面，单击**函数代码**页签，然后单击图标，从下拉列表中，选择**配置测试参数**。
2. 在**配置测试参数**对话框，选择**创建新测试事件**或**编辑已有测试事件**页签，填写事件名称和事件内容。然后单击**确定**。

event是函数计算的入口参数，具体格式如下所示。

```
{
  "id": "c2g71017-6f65-fhcf-a814-a396fc8d****",
  "source": "MNS-Function-mnstrigger",
  "specversion": "1.0",
  "type": "mns:Queue:SendMessage",
  "datacontenttype": "application/json; charset=utf-8",
  "subject": "acs:mns:cn-hangzhou:164901546557****:queues/zeus",
  "time": "2021-04-08T06:28:17.093Z",
  "aliyunaccountid": "1649015465574023",
  "aliyunpublishtime": "2021-10-15T07:06:34.028Z",
  "aliyunoriginalaccountid": "164901546557****",
  "aliyuneventbusname": "MNS-Function-mnstrigger",
  "aliyunregionid": "cn-chengdu",
  "aliyunpublishaddr": "42.120.XX.XX",
  "data": {
    "requestId": "606EA3074344430D4C81****",
    "messageId": "C6DB60D1574661357FA227277445****",
    "messageBody": "TEST"
  }
}
```

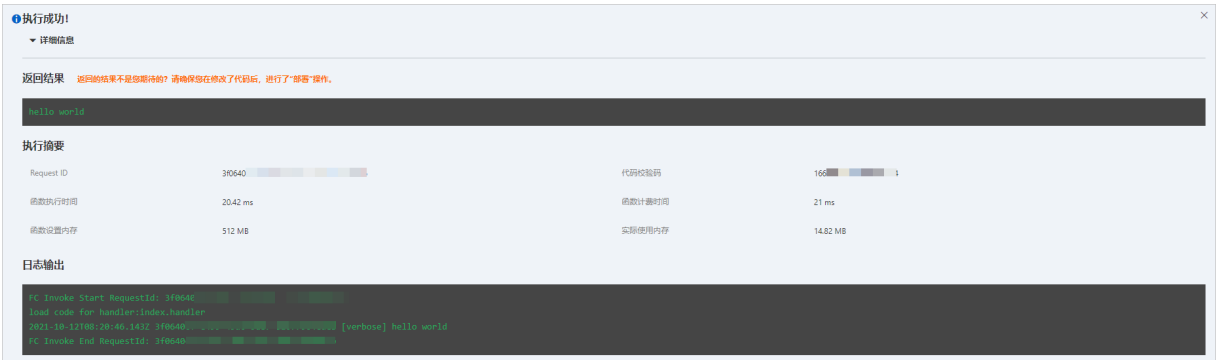
data字段包含的参数解释如下表所示。关于CloudEvents规范中定义的参数解释，请参见[事件概述](#)。

参数	类型	示例值	描述
requestId	String	606EA3074344430D4C81****	请求ID。每个请求的ID取值唯一。
messageId	String	C6DB60D1574661357FA227277445****	消息ID。每条消息的ID取值唯一。
messageBody	String	TEST	消息内容。

3. 在函数详情页面，单击[函数代码](#)页签，然后单击[测试函数](#)。

验证结果

在[函数代码](#)页签，您可以看到执行成功的指示。



更多信息

如需对创建的触发器进行修改或删除，具体操作，请参见[触发器管理](#)。

15.5. 官方事件源触发器

云监控触发器、云服务器ECS触发器、阿里云物联网IoT触发器等多种云服务类型的触发器统称为官方事件源触发器。本文以云服务器触发器为例介绍如何在函数计算控制台创建官方事件源触发器、编写函数和测试函数。

功能简介

您在函数计算控制台提交创建触发器的请求后，函数计算根据触发器的配置信息，将自动在阿里云官方事件总线default下创建一个事件规则**服务名称-函数名称-触发器名称**。创建完成后，您可以在函数计算控制台查看触发器信息，同时也可以事件总线EventBridge控制台查看自动创建的事件规则信息。当事件源指定类型的事件投递到事件总线时，将触发该触发器关联的函数执行一次。

 **说明** 自动创建的事件总线EventBridge的资源事件规则名称必须符合名称取值限制和要求，否则触发器无法创建成功。具体信息，请参见[使用限制](#)。

注意事项

- 事件总线EventBridge的云服务专用事件总线default上最多只能创建10个事件规则，超过10个之后将无法再创建官方事件源触发器。
- Rocket MQ触发器暂时不支持通过Serverless Devs工具或SDK创建。

前提条件

- 事件总线EventBridge
 - [开通事件总线EventBridge并授权](#)
- 函数计算
 - [创建服务](#)
 - [创建函数](#)

步骤一：创建触发器

- 登录[函数计算控制台](#)。
- 在左侧导航栏，单击**服务及函数**。
- 在顶部菜单栏，选择地域。
- 在**服务列表**页面，找到目标服务，在其右侧操作列单击**函数管理**。
- 在**函数管理**页面，单击目标函数名称。



- 在函数详情页面，单击**触发器管理**页签，从**版本**或**别名**下拉列表选择要创建触发器的版本或别名，然后单击**创建触发器**。



7. 在创建触发器面板，填写相关信息。然后单击**确定**。

在函数 Function 中创建触发器

* 触发器类型

云服务器 同步调用

* 名称

ecs-trigger11/128

只能包含字母、数字、下划线和中划线。不能以数字、中划线开头。长度在 1-128 之间。

版本或别名

LATEST

事件类型

自定义事件类型选中全部事件类型

保留云盘

事件模式内容

```
1 {
2   "source": [
3     "acs:ecs"
4   ],
5   "type": [
6     "ecs:Disk:ConvertToPostpaidCompleted"
7   ]
8 }
```

确定

取消

参数	操作	本文示例
触发器类型	选择云服务器。	云服务器
名称	填写自定义的触发器名称。	ecs-trigger
版本或别名	默认值为LATEST，如果您需要创建其他版本或别名的触发器，首先您需要在函数详情页的右上角切换到该版本或别名。关于版本的简介，请参见 版本简介 。	LATEST
事件类型	选择自定义事件类型或选中全部事件类型。如果选择自定义事件类型，您可以选择云服务器的一个或多个事件类型。	保留云盘
事件模式内容	不支持手动编辑，您在事件类型参数中选择事件类型后，事件模式内容自动填充。关于事件模式的信息，请参见 事件模式 。	<pre>{ "source": ["acs.ecs"], "type": ["ecs:Disk:ConvertToPostpaidCompleted"] }</pre>

创建完成后，在触发器名称列表中显示已创建的触发器。



步骤二：编写函数

完成云服务器触发器的创建之后，您可以编写函数代码。

1. 在函数详情页面，单击[函数代码](#)页签，在代码编辑器中编写代码。
代码示例如下。

```
'use strict';
/*
To enable the initializer feature
please implement the initializer function as below:
exports.initializer = (context, callback) => {
  console.log('initializing');
  callback(null, '');
};
*/
exports.handler = (event, context, callback) => {
  console.log("event: %s", event);
  handle_event(event);
  callback(null, 'return result');
}
```

步骤三：测试函数

完成函数编写后，您需要调试函数以验证代码的正确性。在实际操作过程中，当阿里云官方事件源云服务器ECS产生的事件通过事件总线EventBridge投递到函数计算时，会自动触发函数执行。

云服务器ECS事件会以event的形式作为输入参数传递给函数，您可以手动将event传给函数模拟触发事件，测试函数代码是否正确。

1. 在函数详情页面，单击**函数代码**页签，然后单击图标，从下拉列表中，选择**配置测试参数**。
2. 在**配置测试参数**对话框，选择**创建新测试事件**或**编辑已有测试事件**页签，填写事件名称和事件内容。然后单击**确定**。

event是函数计算的入口参数，具体格式如下所示。关于支持的所有官方事件源的事件内容，请参见[云服务事件](#)。

```
{
  "datacontenttype": "application/json;charset=utf-8",
  "aliyunaccountid": "123456789098****",
  "data": {
    "result": "accomplished",
    "diskId": "d-bp11ba7acc69nkta****"
  },
  "subject": "acs:ecs:cn-hangzhou:123456789098****:disk/d-bp11ba7acc69nkta****",
  "source": "acs.ecs",
  "type": "ecs:Disk:ConvertToPostpaidCompleted",
  "aliyunpublishtime": "2021-01-18T03:58:31.762Z",
  "specversion": "1.0",
  "aliyuneventbusname": "default",
  "id": "70c0414c-b260-4923-b584-1d6e5646****",
  "time": "2021-01-18T11:58:31.125+08:00",
  "aliyunregionid": "cn-hangzhou",
  "aliyunpublishaddr": "172.25.XX.XX"
}
```

event中涉及的参数解释如下表所示。

参数	类型	示例值	描述
----	----	-----	----

参数	类型	示例值	描述
datacontenttype	String	application/json;charset=utf-8	参数data的内容形式。datacontenttype只支持application/json格式。
aliyunaccountid	String	123456789098****	阿里云账号ID。
data	Struct	<pre>{ "result": "accomplished", "diskId": "d- bp11ba7acc69nкта ****" }</pre>	事件内容。JSON对象，内容由发起事件的服务决定。CloudEvents可能包含事件发生时由事件生产者给定的上下文，data中封装了这些信息。
subject	String	acs:ecs:cn-hangzhou:123456789098****:disk/d-bp11ba7acc69nкта****	事件主题。
source	String	acs.ecs	事件源。
type	String	ecs:Disk:ConvertToPostpaidCompleted	事件类型。
aliyunpublishtime	Timestamp	2021-01-18T03:58:31.762Z	接收事件的时间。
specversion	String	1.0	CloudEvents协议版本。
aliyuneventbusname	String	default	接收事件的事件总线名称。
id	String	70c0414c-b260-4923-b584-1d6e5646****	事件ID。
time	Timestamp	2021-01-18T11:58:31.125+08:00	事件产生的时间。
aliyunregionid	String	cn-hangzhou	接收事件的地域。
aliyunpublishaddr	String	172.25.XX.XX	接收事件的服务器地址。

3. 在函数详情页面，单击**函数代码**页签，然后单击**测试函数**。

验证结果

在函数代码页签，您可以看到执行成功的指示。

执行成功!

详细信...

返回结果

返回的结果不是您期待的？请确保您在修改了代码后，进行了“部署”操作。

hello world

执行摘要

Request ID	3f0640...	代码包大小	166...
函数执行时间	20.42 ms	函数计费时间	21 ms
函数设置内存	512 MB	实际使用内存	14.82 MB

日志输出

```
FC Invoke Start RequestId: 3f0640...
load code for handler: index.handler
2021-10-12T08:28:46.143Z 3f0640... [verbose] hello world
FC Invoke End RequestId: 3f0640...
```

更多信息

如需对创建的触发器进行修改或删除，具体操作，请参见[触发器管理](#)。

16.Serverless工作流

本文介绍了工作流调度固定版本或预留资源函数和Serverless工作流实现的具体步骤。

简介

在实际生产场景中，任务流所调度的函数因为业务场景变化可能需要频繁变更，您会考虑到如何避免变更带来的非预期行为，控制变更稳定性。在Serverless工作流任务步骤使用固定版本的函数将对以下场景产生帮助：

- 流程A编排了多个函数f1、f2和f3，同一次任务必须执行函数的相同版本。假如流程A正在执行中，已执行完了函数f1，但是此时进行了函数更新，则正在执行的流程可能会执行函数f2、f3的最新版本造成非预期情况发生。那么必须保证，每次流程执行函数的版本在执行时已经固定。
- 某个函数的快速回滚。上线后，发现流程执行失败是由于新变更引起的，那么需要快速回滚流程执行上一固定版本。
- 通过函数别名调用预留资源函数、降低函数冷启动时间及[优化函数成本](#)。

函数计算[版本](#)及[别名](#)可以有效支持类似上述场景的持续集成、持续发布的各种需求。下文将通过一个具体示例展示如何在流程中使用别名调用预留资源函数。预留资源函数是依赖于固定版本函数的，因此对于其他需要固定版本的场景，也可参见本示例操作。

步骤一：创建函数预留实例

1. 首先创建一个名为 `fnf-demo` 的函数计算服务，并在该服务下创建一个名为 `provision` 的Python3函数并发布版本、别名，生成预留实例，详细步骤，请参见[弹性伸缩（含预留资源）](#)。

假设创建的函数版本为1，别名为online，并发布了一个预留实例。函数内容如下。

```
import logging
def handler(event, context):
    logger = logging.getLogger()
    logger.info('Started function test')
    return {"success": True}
```

步骤二：创建工作流

Serverless工作流对函数计算版本及别名进行了原生支持。

在Serverless工作流的Task步骤中，默认填写的 `resourceArn` 一般为 `acs:fc:{region}:{accID}:services/fnf/functions/test`。按照函数执行规则，默认会选择最新的函数版本执行。您可以发布[版本](#)或[别名](#)，并在工作流Task步骤中填写 `resourceArn` 一项为 `acs:fc:{region}:{accID}:services/fnf.{别名or版本}/functions/test` 来实现对特定版本函数的调用。因此，流程可以定义如下：

```
version: v1
type: flow
steps:
  - type: task
    resourceArn: acs:fc:::services/fnf-demo.online/functions/provision
    # 也可以使用版本号，resourceArn: acs:fc:::services/fnf-demo.1/functions/provision。
    name: TestFCProvision
```

步骤三：使用命令行或控制台执行并观察预留函数执行

1. 启动工作流执行。使用预留模式前。

+	4	TaskStarted	TestFCProvision	2019年11月15日 17:35:31	57
+	5	TaskSucceeded	TestFCProvision	2019年11月15日 17:35:31	564

2. 使用预留模式后。

+	4	TaskStarted	TestFCProvision	2019年11月15日 17:44:40	61
+	5	TaskSucceeded	TestFCProvision	2019年11月15日 17:44:40	298

从工作流的执行时间来看，在task步骤从未使用预留模式前的耗时500 ms缩短为预留模式后的230 ms。

17.CDN源站

阿里云CDN支持的源站类型包括OSS域名、IP、源站域名和函数计算域名，每种源站类型都支持配置多个源站地址，多源站场景下，支持设置源站的主备优先级和权重，实现负载均衡。本文介绍如何新增或修改源站信息及源站的健康检查策略。

背景信息

操作步骤

- 1.