目录

目录 1
授权管理 3
授权流程 3
命名空间管理 3
查看命名空间 3
管理命名空间 3
创建命名空间 3
修改命名空间 3
删除命名空间 3
函数概述 3
函数相关概念 3
函数管理 4
前提条件 4
创建函数 4
查看函数 4
修改函数配置 4
删除函数 4
构建函数 4
支持的编程语言 4
创建代码包 5
函数配置 5
基本信息 5
环境配置 5
执行配置 5
触发配置 6
日志配置 6
网络配置 6
层管理 6
概述 6
功能原理 6
构建层 7
各运行时使用层说明 7
各运行环境的层ZIP包文件结构 7
Python Runtime 7
Node.js Runtime 7
Java Runtime 7
构建层的ZIP包 7
Python Runtime 8
Node.js Runtime 8
Java Runtime 8
函数中引用层的依赖 10
Python Runtime 10
Node.js Runtime 10
Java Runtime 10
在函数中使用层 10
创建层 10
配置层 11
触发器简介 11
事件触发 11
HTTP请求触发 11

触发器 11
使用对象存储触发器 11
KS3事件定义 12
KS3触发器的事件消息结构 12
KS3触发器配置 12
触发器配置 12
触发路径 12
触发器管理 12
使用HTTP触发器 12
协议支持 13
HTTP触发器配置 13
使用限制 13
触发器限制 13
HTTP协议使用限制 13
默认域名 13
签名机制 13
步骤一： 创建一个正规化请求 13
要创建正规化请求，请将以下来自每个步骤的部分连接为一个字符串： 14
步骤二：将正规化请求进行哈希处理 15
步骤三：创建签名字符串 15
步骤四：计算签名字符串 15
步骤五：计算签名 15
函数调用 16
HTTP请求函数 16
应用场景 16
使用限制 16
事件请求函数 16
应用场景 16
重试策略 16
弹性管理概述 16
概述 16
什么是冷启动 16
弹性管理 17
弹性管理配置 17
弹性管理基础配置 17
弹性策略 17
事件函数：队列堆积指标 17
HTTP函数：并发指标 18
定时弹性伸缩 18
内存指标弹性伸缩 18
多弹性策略组合 18
函数监控 18
日志管理 19
配置日志采集 19
查看调用日志 19
调用请求列表 19

授权管理

当您初次使用云函数时，需在开通服务的同时，将系统默认角色KsyunKCFDefaultVersionRole授权给云函数服务。服务开通后，云函数将默认使用角色策略KsyunKCFDefaultRolePolicy访问其他云产品中的资源。

授权流程

1. 登录云函数控制台。
2. 初次使用云函数服务，或未进行正确角色授权时，系统将进行如下提示：

3. 点击**同意授权**，进行服务开通与角色授权。
4. 开通完成，即可进入云函数控制台页面，进入正常使用。

注：

- 如您需要查看默认角色详细的策略信息，可以登录IAM控制台进行查看。
- 如需修改角色权限，请前往角色管理设置，需要注意的是，错误的配置可能导致金山云云函数服务无法获取必要的权限，造成服务的不可用。

命名空间管理

命名空间为一组函数提供管理空间，创建云函数时，您可以选择函数所在的命名空间，根据管理需求对多个函数进行分组管理。

注：每个地域下最多创建5个命名空间，其中default命名空间不支持修改名称和删除。

查看命名空间

1. 登录云函数控制台。
2. 进入函数管理页面，在函数列表上方可查看当前**地域**及**命名空间**。 
3. 您可通过下拉列表切换**命名空间**，进入不同命名空间的函数管理列表页。

管理命名空间

创建命名空间

1. 点击命名空间下拉列表中的**管理**按钮，进入命名空间管理页。 
2. 点击**新增命名空间**，输入命名空间名称与描述，点击**确定**，即可完成命名空间的创建。

命名空间名称需满足规则：只允许使用字母、数字和连字符，且以字母开头。长度1-24。

修改命名空间

您可在命名空间列表页编辑命名空间的描述，暂不支持创建后修改命名空间名称。

删除命名空间

点击**删除**，并在弹窗后点击**确认**，即可删除命名空间。

注：删除命名空间前，请清理该命名空间下的所有函数资源，否则该命名空间将无法删除。

函数概述

函数是金山云函数服务（Kingsoft Cloud Function, KCF）管理和运行的基本单元，由业务代码和函数配置构成，在被触发运行时进行环境准备、执行代码逻辑，完成触发事件处理。

函数相关概念

地域：函数资源具有地域属性，目前KCF支持的地域请参见支持地域。

命名空间：命名空间为一组函数提供管理空间，创建云函数时，您可以选择函数所在的命名空间，根据管理需求对多个函数进行分组管理。

调用次数：某函数被调用的总请求次数。

失败次数：某函数的历史调用中，发生异常请求且函数不能正确执行的次数，包含客户端错误、系统错误 、函数错误造成的调用失败。

调用日志：函数调用日志中，记录了每一次请求的Request ID、调用结果、执行时间、函数代码中打印的业务日志等信息。调用信息模块可对调用日志进行限定时间段内的调用日志进行检索分析。

构建函数：函数中包含的业务代码需要用户自行创建及打包，如用户在本地以Java运行时开发程序并编译打包后，可以ZIP包格式上传至KCF。函数代码开发规范请参考开发手册。

函数配置：除业务代码外，在函数中还需对代码运行环境及执行方式进行配置，包含运行环境、函数入口、实例规格、超时时间、并发配置等参数。

触发器：触发器是触发函数执行的方式，当满足触发器定义的规则时，触发器关联的函数将被自动调用。触发器可在创建函数时配置，也可在创建函数后再进行配置/修改。更多信息可参考触发器简介。

函数管理

在金山云云函数服务（KCF）中，您可以函数为基本单位对服务资源进行管理。本文将介绍如何通过函数服务控制台创建、查询、更新和删除函数。

前提条件

您已开通函数服务并完成角色授权，操作步骤请参考授权管理。

创建函数

1. 登录云函数控制台。
2. 在顶部菜单栏，选择**地域**及**命名空间**。
3. 在**函数管理**页面，单击**新建函数**。
4. 在新建函数页面，参照函数配置对函数运行环境等信息进行配置，然后点击**创建**。

等待函数创建完毕，在**函数管理**页面，可查看已创建的函数。

> 注：在开通函数服务后，初次创建函数时，资源初始化过程会花费约3分钟左右时间。后续创建函数及函数实例扩容再无此过程，启动时间不受影响。

查看函数

通过**函数管理**页面函数列表，您可查看函数的关键信息，包含函数名称/ID、请求处理类型、运行环境、监控、调用次数、失败次数、实例配置、创建时间、最近触发时间。

点击函数名称，即可进入函数详情页，在**函数配置**页签下，您可查看此函数的完整配置信息。

修改函数配置

点击**修改配置**，您可对函数的环境配置、执行配置、网络配置、日志配置进行更新，关于配置的说明请参考函数配置。

> 注：更新函数时上传的代码包，将对原代码包进行覆盖。

删除函数

函数列表中，点击**删除函数**，将删除函数及相关配置。

构建函数

函数执行的核心逻辑由业务代码构成，本文将介绍如何进行函数中的代码构建与配置。

支持的编程语言

运行环境	版本	代码上传方式
Java	Java 8 Java 11 Java 17	- 通过ZIP包上传 - 通过对象存储（KS3）上传
Golang	Go 1	- 通过ZIP包上传 - 通过对象存储（KS3）上传
Python	Python 3.10	- 通过ZIP包上传 - 通过对象存储（KS3）上传
Node.js	Node.js 10.24 Node.js 12.22 Node.js 14.20 Node.js 16.17	- 通过ZIP包上传 - 通过对象存储（KS3）上传

创建代码包

程序包包含业务代码和依赖项的文件，目前KCF支持通过ZIP包上传/对象存储上传两种方式将代码包导入函数。您可在本地开发程序后进行打包，格式需是ZIP包。关于代码开发可参考 开发手册。

> 注：
>
> 代码包文件大小上限为50MB。

函数配置

在准备好程序包后，您可通过函数服务（KCF）控制台对函数进行配置，包括函数基本信息、环境配置、执行配置、触发配置、日志配置、网络配置。本文将介绍关于如何进行函数的相关配置。

基本信息

函数名称：指定函数名称，不能与已有函数名称重复。名称需以英文字母（a-z）、（A-Z）或下划线（_）开始，只能包含字母、数字、下划线和中划线。

请求处理类型：您可通过选择请求处理类型区分HTTP请求/事件请求函数的配置，HTTP请求类函数和事件请求类函数分别只支持配置对应类型的触发器。函数创建后，您仍可通过删除重建触发器的方式修改函数请求处理类型。

描述信息：可为函数配置描述信息，长度在1-256字符之间。

环境配置

运行环境：选择函数的代码运行环境，KCF目前支持的运行环境详见构建函数。

代码上传：选择通过本地代码包上传或对象存储（KS3）上传，代码包准备方式参照开发手册。

启动命令：配置程序的启动命令用于启动您的函数，执行路径为您上传的代码包的根目录，如java -jar demo.jar。

> 您可通过添加启动参数的方式自定义函数启动命令，如在Java环境的函数实例中，您可通过设置JVM堆空间大小（-Xms，-Xmx，其他JVM参数），保障JVM运行性能。

监听端口：指定函数实例中HTTP Server监听的端口(默认为8080)，用于接收HTTP请求并转发给后端服务，完成逻辑处理后返回给用户。

实例配置：设置函数实例内存，您可手动输入内存大小（需为64MB的倍数），自定义函数实例执行内存。

环境变量：设置函数运行环境中的环境变量，以键值对的方式配置函数中需要的配置信息。

高级配置：

- 健康检查：支持通过周期性HTTP请求的方式探测函数实例健康状态。存活检查用于检测函数实例是否存活，存活检查失败时，将对该函数实例执行重启操作；就绪检查用于检测函数实例是否准备好开始处理请求，就绪检查失败时，将屏蔽请求访问该容器。两类健康检查方式下，您可以通过定义检查协议、检查端口、请求路径、响应超时时间对检查方式进行详细配置。

执行配置

超时时间：设置超时时间，当超过此时间时，函数将以执行失败结束。默认值为60秒，最大值为86400秒。

注意：超时时间不包含函数实例资源准备时间（如扩容、冷启动环节），请参考函数执行时间合理配置。

单实例并发数：设置单个实例能并发处理的请求数，默认值为1。当设置单实例并发数大于1时，函数实例在利用完一个实例的并发数后才会创建新的实例。

异步调用重试：事件类函数中，对函数异步调用执行失败后的重试策略进行配置。

- 重试次数：设置最大重试次数，超过设置值后，异步触发失败的调用将不再重试。默认值为2，支持范围为0-3次。
- 重试间隔：设置函数重试执行的时间间隔。默认值为60秒，支持范围为60-120秒。
- 最长保留时间：设置函数的异步事件队列中，事件保留的最长时间，超过此时间的事件将被丢弃。默认事件为2小时，支持时间范围为1分钟-6小时。

触发配置

当前云函数支持事件触发类触发器，同一个函数支持配置多个触发器，您可在创建函数时进行配置，也可选择**暂不配置**，在完成函数配置后再进行触发器配置。详细配置可参考触发器管理。

日志配置

当前云函数支持将函数调用执行的日志投递至金山云日志服务KLog进行存储与检索分析，您可在创建函数时开启**使用日志服务KLog**，指定函数日志的消费端，在KLog中由日志项目+日志池构成。

使用已有日志实例：如果您已为函数日志规划好明确的日志池和所属日志项目，可在使用已有日志实例模式下选择，或通过**新建日志项目与日志池**，跳转至KLog控制台创建好日志实例后，再选择具体日志项目与日志池。

自动新建日志实例：若无需指定特定日志项目和日志池，您可以选择自动新建日志实例，后台将自动在KLog服务中创建名为kcf-log-{function-id}的日志项目和名为{function-id}的日志池。

> 注：自动新建的日志池，日志分区数默认为2，日志保存时长默认为14天。若您有调整日志分区数和日志保存时长的需要，可在日志实例创建后，到日志服务控制台，进入**日志项目>日志池**，选择**编辑**日志池。对日志池的分区数与保存时长进行调整。

网络配置

默认状态下，函数无法访问公网及VPC环境。若您希望函数能访问VPC内/公网环境资源（如云数据库、文件存储、云服务器等），可手动为函数开启相关配置。

允许函数访问公网：勾选后将为函数开启公网访问能力。

允许函数访问VPC内资源：勾选后将为函数开启访问特定VPC的能力。

VPC：在列表中选择函数可以访问的VPC。

子网：选择函数绑定的弹性网卡所在子网，函数将通过弹性网卡访问此VPC内资源。

安全组：选择函数所在的安全组，通过安全组配置函数在VPC中的出入站规则。

> 注：目标访问资源所在的安全组的入站规则，需加入函数所属网段，以使函数至目标访问资源可通。

层管理

概述

如果您的云函数（KCF）拥有较多的依赖库或公共代码文件，您可以使用 KCF 中的层进行管理。使用层管理，您可以将依赖放在层中而不是部署包中，可确保部署或更新函数时保持较小的体积。对于 Java、Python和Node.js函数，只要将部署程序包保持在50MB以下，就可以在 KCF 控制台中在线编辑函数代码。

功能原理

- 函数代码和依赖库或依赖的静态文件分离，保持函数代码较小体积。在使用IDE开发工具、命令行工具编辑函数时，可以做到快速部署更新。
- 构建层时，需要将所有内容打包到ZIP文件中。云函数运行时会将层的内容解压并部署在/opt目录下。
- 当函数配置多个层时，这些层的内容将被合并至/opt目录，多个层按照层绑定时的顺序进行解压。
- 如果某一文件与其他层中的文件同名，则后配置层中的该文件会覆盖先配置层中的该同名文件。
- 如果层中的代码依赖二进制的库或可执行文件，则需要使用Linux系统编译构建层。

构建层

各运行时使用层说明

层中的文件均在/opt/目录下，可以在函数代码中通过绝对路径进行访问。除此之外，云函数内置的环境变量添加了运行时语言的依赖包搜索路径，如果在层ZIP包中定义了与其相同的文件夹结构，则函数代码无需指定路径即可访问层。

Python、Node.js 环境变量见下表：

相关环境变量	特定目录
PYTHONPATH	/opt/python
NODE_PATH	/opt/nodejs/node_modules

各运行环境的层ZIP包文件结构

Python Runtime

使用flask依赖打包后的文件结构

```
python-layer-code.zip
└── python
    └── flask
```

ZIP包解压部署后的路径

```
/opt
└── python
    └── flask
```

Node.js Runtime

使用express依赖打包后的文件结构

```
nodejs-layer-code.zip
└── nodejs
    ├── node_modules
    │   └── express
    │   └── function-bind
    ├── package-lock.json
    └── package.json
```

ZIP包解压部署后的路径

```
/opt
└── nodejs
    ├── node_modules
    │   └── express
    │   └── function-bind
    ├── package-lock.json
    └── package.json
```

Java Runtime

使用cloudevents-core依赖打包后的文件结构

```
java-layer-code.zip
└── java
    └── lib
        └── cloudevents-core-2.3.0.jar
```

ZIP包解压部署后的路径

```
/opt
└── java
    └── lib
        └── cloudevents-core-2.3.0.jar
```

构建层的ZIP包

创建层时，需要将所有内容打包到ZIP文件中。云函数运行时会将层的内容解压并部署在/opt目录下。

构建层的ZIP包的方式和构建代码包的方式类似，为使函数在运行时能正确加载以层发布的库，库的代码目录结构需遵从各个语言标准的目录规范，具体信息，请参见各运行时使用层说明。对于部署于层的函数依赖库，如果按照规范的方式打包，云函数运行时会为您自动添加各语言的依赖库搜索路径，您无需指定全路径。如您想自定义层的目录结构，需要在代码中显式添加依赖库搜索地址。具体操作，请参见函数中引用层的依赖。

各运行时构建层ZIP包的操作步骤如下所示：

说明：下面创建的工作目录仅为示例，在实际操作过程中您可以按照自身需求创建目录名称。

Python Runtime

1、执行mkdir命令创建工作目录

```
mkdir python-layer-code
```

2、进入已创建的工作目录

```
cd python-layer-code
```

3、执行pip命令安装依赖库到python-layer-code/python。

```
pip3 install --target ./python flask
```

其中flask是您要安装的依赖包的名称，具体以实际项目需要为准。 安装完成后，目录结构如下：

```
python-layer-code
└── python
    ├── flask
    ├── Flask-2.2.2.dist-info
    ├── itsdangerous
    └── itsdangerous-2.1.2.dist-info
```

4、在python-layer-code目录，执行zip命令打包依赖

```
zip -r python-layer-code.zip python
```

Node.js Runtime

1、执行mkdir命令创建工作目录

```
mkdir nodejs-layer-code
```

2、进入已创建的工作目录

```
cd nodejs-layer-code
```

3、执行npm命令安装依赖库到nodejs-layer-code/nodejs。

```
npm install --prefix ./nodejs --save express
```

其中express是您要安装的依赖报的名称，具体以实际安装为准。 安装完成后，目录结构如下：

```
nodejs-layer-code
└── nodejs
    ├── node_modules
    │   └── express
    │       └── function-bind
    ├── package-lock.json
    └── package.json
```

4、在nodejs-layer-code目录，执行zip命令打包依赖

```
zip -r nodejs-layer-code.zip nodejs
```

Java Runtime

1、执行mkdir命令创建工作目录

```
mkdir java-layer-code
```

2、进入已创建的工作目录

```
cd java-layer-code
```

3、使用maven工具安装依赖 在java-layer-code目录创建一个pom文件 pom文件配置如下：

```
<?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 https://maven.
apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>
    <groupId>com.kcf.demo</groupId>
    <version>1.0</version>
    <artifactId>layer-dependency</artifactId>
```

```
    <dependencies>
        <dependency>
              <dependency>
            <groupId>io.cloudevents</groupId>
            <artifactId>cloudevents-core</artifactId>
            <version>2.3.0</version>
        </dependency>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                 <!-- 项目文件打包 -->
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-jar-plugin</artifactId>
                <configuration>
                    <archive>
                        <!-- 生成的jar中，不要包含pom.xml和pom.properties这两个文件 -->
                        <addMavenDescriptor>false</addMavenDescriptor>
                        <manifest>
                            <!-- 是否要把第三方jar加入到类构建路径 -->
                            <addClasspath>false</addClasspath>
                        </manifest>
                    </archive>
                </configuration>
            </plugin>

            <!-- maven依赖打包插件 -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-dependency-plugin</artifactId>
                <executions>
                    <execution>
                        <id>copy-dependencies</id>
                        <phase>package</phase>
                        <goals>
                            <goal>copy-dependencies</goal>
                        </goals>
                    </execution>
                </executions>
                <configuration>
                    <!-- 是否排除间接依赖。默认false，不排除 -->
                    <excludeTransitive>false</excludeTransitive>
                    <!-- 是否消除依赖jar包后缀的版本信息。默认是false，不取消版本信息 -->
                    <stripVersion>false</stripVersion>
                    <!-- 输出文件路径 -->
                    <outputDirectory>target/java/lib</outputDirectory>
                    <stripVersion>false</stripVersion>
                    <includeScope>runtime</includeScope>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

pom文件示例解析如下：

- 要安装的依赖包为cloudevents-core。
- 使用maven-jar-plugin将项目文件打包成一个轻量级的jar包。
- 使用maven-dependency-plugin将需要安装的依赖包拷贝到target/java/lib目录下。

  在java-layer-code目录下执行以下命令安装依赖。

```
mvn dependency:copy-dependencies
```

安装完成后，目录结构如下：

```
java-layer-code
└── target
    └── java
        └── lib
            └── cloudevents-core-2.3.0.jar
```

4、在java-layer-code/target目录，执行zip命令打包依赖

```
zip -r java-layer-code.zip java
```

函数中引用层的依赖

对于部署于层的函数依赖库，如果按照规范的方式打包，函数计算运行时会为您自动添加各语言的依赖库搜索路径，您无需指定全路径。如您想使用自定义层的目录结构，需要在代码中显式添加依赖库搜索地址。

Python Runtime

按照文档构建的层，其依赖库在/opt/python目录下；需要将该目录添加到模块搜索路径中。 方式1：在函数配置中设置PYTHONPATH环境变量，添加层所在的目录。 示例如下。

```
PYTHONPATH=/opt/python
```

方式2：在您项目的入口文件里添加以下语句，需要在导入层的依赖库前执行。

```
#server.py
import json
import os
import logging
import sys
sys.path.append('/opt/python')
# 必须在 import numpy 前执行
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route("/health")
def health_check():
    print("python path= ", os.sys.path)
    return "<p>Hello, World Flask!</p>", 200

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=8080)
```

注意：必须在引用层中的模块（如flask）前执行 sys.path.append()

Node.js Runtime

按照文档构建的层，其依赖库在/opt/nodejs/node_modules目录下，需要将该目录添加到模块搜索路径中。

```
NODE_PATH=/opt/nodejs/node_modules
```

在函数配置中设置NODE_PATH环境变量，添加层所在的目录。

Java Runtime

在启动命令中设置-cp参数，添加层所在/opt/java/lib/*目录。

```
java -cp java-slim.jar:/opt/java/lib/* com.example.demo.Application
```

说明：

- java-slim.jar: 项目文件构建的轻量级jar包
- /opt/java/lib/*: 层依赖的路径
- com.example.demo.Application: mainclass的路径

在函数中使用层

在您准备好层和函数后，可通过KCF控制台完成层的创建和函数中配置。

创建层

1. 登录云函数控制台。
2. 进入**层管理**，点击**新建层**，完成相关配置，进行层的创建。
   - 层名称：设置层的名称。
   - 描述信息：设置层的描述信息。
   - 运行环境：选择层兼容的运行环境，支持多选。
   - 层上传：将您构建好的层以ZIP格式从本地/通过对象存储（KS3）上传。

创建层的压缩文件将按照层的版本进行存储。层创建成功后，将自动生成层的版本，版本号从1开始递增。如需新增版本，可在层的详情页，创建新的版本。

说明：

已创建的层或层版本不支持修改，您可以通过创建新的层或新版本来替代老版本。注意需同步在函数的层配置中进行绑定。

配置层

您可在创建函数时，或函数详情中，为函数配置层。

层在与函数进行绑定时，将按照具体的层版本与函数版本进行绑定。一个函数目前最多支持绑定5个层的具体版本，并在绑定时有一定顺序。

说明：

- 层中的文件将会添加到 /opt 目录中，此目录在函数执行期间可访问。
- 如果您的函数已绑定了多个层，这些层将按顺序合并到 /opt 目录中。如果同一个文件出现在多个层中，KCF平台将会保留高优先级层里的文件。其中新添加的层优先级数值小，优先级高。

触发器简介

金山云云函数根据函数处理请求类型的不同，将函数区分为事件请求类函数/HTTP请求类函数。根据您对函数配置的触发器，函数被置为不同的请求处理类型，和调用方式的对应关系见下表：

函数请求处理类型	支持的触发器类型	调用方式
处理事件请求	对象存储触发器	异步调用
处理HTTP请求	HTTP触发器	同步调用

事件触发

云函数提供了一种事件驱动的计算模型，函数的执行可以由各类事件源或云函数控制台/SDK触发。这种计算模型可以应用到如下一些典型场景中：

音视频文件处理：用户上传视频/图片文件后，对文件进行转码/切片/压缩等处理。此场景中，对象存储为事件源，可选择对象存储类触发器，在文件创建时将事件投递至云函数，函数执行文件处理操作。

数据处理：数据上传至对象存储后，对原始数据进行清洗和加工，将处理后数据加入数据集。此场景中，数据上传类事件可作为触发事件，触发云函数执行数据清洗工作。

触发函数执行的事件源可以分为两类：

云产品事件源：云函数触发器会逐步与金山云相关云产品进行集成，支持对象存储等云产品事件触发函数执行。

第三方应用程序：通过对接事件总线API，将第三方应用程序的事件接入至自定义总线作为函数的事件源。

HTTP请求触发

区别于事件函数对事件格式的限制，HTTP请求可以通过云函数内部网关直接传入云函数环境，触发函数的运行与处理。这种触发方式适用的典型场景如：

Web服务：云函数作为Web应用的后端，实现服务端应用逻辑，通过API对外提供服务。基于云函数的Serverless和弹性特性，开发者能免除运维方面压力，轻松构建可弹性扩展的Web应用程序。

触发器

您可以在函数中创建触发器，触发器描述了一组规则，当时间满足规则时，事件源会触发关联的函数。

对象存储（KS3）触发器：KS3中Bucket的特定动作，在符合触发路径规则的前提下，将事件数据推送给KCF函数。KS3使用异步调用方式调用函数，结果不会返回到调用方。

HTTP请求触发器：支持多种HTTP请求方式，HTTP请求通过云函数内置网关传入至函数环境，使用同步调用方式调用函数，云函数将函数返回值封装为JSON格式并返回给调用方。

使用对象存储触发器

对象存储KS3（Kingsoft Cloud Standard Storage Service，KS3）是金山云提供的海量、低成本、强安全、高可靠的分布式云存储服务，为用户解决存储扩容、数据可靠安全以及分布式访问等相关复杂问题。用户在KS3中存储和获取图片、音频、视

频、文本等各类数据文件的事件，可触发云函数的执行。

KS3事件定义

当前KS3触发器支持的时间类型见下表。

事件类型	描述
ks3:ObjectCreated:PutObject	文件上传时触发函数，暂不区分上传为新上传或覆盖上传
ks3:ObjectCreated:CompleteMultipartUpload	完成分片上传时触发函数
ks3:ObjectCreated:SyncMirror	通过同步回源生成文件时触发函数
ks3:ObjectCreated:AsyncMirror	通过异步回源生成文件时触发函数
ks3:ObjectRemoved:DeleteObject	删除文件时触发函数

KS3触发器的事件消息结构

关于KS3触发器的事件消息结构可参考CloudEvents信息格式规范说明。

KS3触发器配置

通过函数控制台配置KS3触发器，操作步骤如下：

1. 登录云函数控制台。
2. 在顶部菜单栏，选择**地域**及**命名空间**。
3. 在**函数管理**页面，点击函数名称，进入函数详情页。
4. 在函数详情页，点击**触发管理**页签，进入触发器管理页。
5. 点击**创建触发器**，对触发器进行配置。

   触发器配置

   触发类型：选择KS3触发类型。

   触发器名称：配置触发器名称，不能与已有触发器名称重复。名称需以英文字母（a-z）、（A-Z）或下划线（_）开始，只能包含字母、数字、下划线和中划线。

   Bucket列表：选择已创建的KS3 Bucket，所选Bucket需与函数位于同一地域。

   函数入口：为触发器指定调用入口，指定调用函数时从哪个路径开始执行。您需提前在代码中以path的形式进行设置，设置方式可参考开发手册中示例代码。

   触发路径

   支持创建多条触发路径，即满足触发器中任一触发路径，都将触发函数执行。触发路径由触发事件、前缀过滤、后缀过滤构成。每个触发器中最多支持三条触发路径。

   触发事件：选择一个触发事件，事件定义请参考KS3事件定义。

   前缀过滤：通常用于过滤指定目录下的文件事件，例如前缀过滤为test/，则仅test/ 目录下的文件事件才可以触发函数，其他目录下的文件事件不会触发函数。

   后缀过滤：通常用于过滤指定类型或后缀的文件事件。例如，后缀过滤为 .jpg，则仅 .jpg 结尾的文件的事件才可以触发函数，其他后缀的文件事件不触发函数。

触发器管理

触发器创建完成后，可在触发器列表中查看和管理已创建的触发器。支持修改触发器状态、触发器配置、删除触发器。

开启/关闭触发器：触发器默认处于关闭状态，若您需要启用某触发场景，可选择开启对应触发器。

修改配置：KS3触发器支持修改Bucket列表、函数入口、触发路径。

使用HTTP触发器

HTTP触发器通过接收HTTP请求，根据HTTP方法和URL找到匹配的函数，将HTTP请求相关信息传入并执行函数，获取执行结果。

协议支持

支持HTTP/HTTPS协议，通过GET, POST, PUT, DELETE, PATCH请求方式触发。

HTTP触发器配置

通过函数控制台配置HTTP触发器，操作步骤如下：

1. 登录云函数控制台。
2. 在顶部菜单栏，选择**地域**及**命名空间**。
3. 在**函数管理**页面，点击函数名称，进入函数详情页。
4. 在函数详情页，点击**触发管理**页签，进入触发器管理页。
5. 点击**创建触发器**，对触发器进行配置。

- **触发类型**：选择**HTTP触发器**
- **触发器名称**：配置触发器名称，不能与已有触发器名称重复。名称需以小写英文字母（a~z）开始，只能包含字母、数字、下划线。
- **签名认证**：可选择**开启**或**关闭**，默认状态下签名认证关闭，支持匿名访问HTTP函数。若需对HTTP请求进行身份验证，可开启签名认证。关于签名算法的说明请参考签名机制。
- **请求方式**：支持GET, POST, PUT, DELETE, PATCH方式触发，以及支持选择多种方式。

使用限制

触发器限制

- HTTP触发器和其他事件类触发器不能并存，如您希望修改函数的请求处理类型，可以在删除HTTP请求触发器后再配置事件类触发器。
- 每个函数只能创建一个HTTP触发器。

HTTP协议使用限制

- HTTP Request限制
  - Headers字段：不支持以下自定义字段
  - 以X-Amz-开头的字段
  - Authorization
  - Body大小：Body的总大小不超过65535 字节

默认域名

云函数会在创建HTTP触发器时为函数分配默认地址，您可以通过此URL调用函数。

```
# 公网访问地址
https://{function_id}-{region-id}.ksyuncf.com

# 内网访问地址
https://{function_id}-{region-id}-vpc.ksyuncf.com
```

签名机制

HTTP函数调用支持AWS签名算法版本4，详情可以参考AWS文档。

签名计算的主要流程如下：

- 步骤一：创建一个正规化请求。
- 步骤二：将正规化请求进行哈希处理。
- 步骤三：创建签名字符串。
- 步骤四：计算签名。

步骤一： 创建一个正规化请求

在签名前，请按照以下步骤创建正规化请求，确保金山云在收到请求时计算出的签名与您计算出的签名相同，否则请求将被拒绝。 **示例：正规化请求伪代码**

```
CanonicalRequest =
HTTPRequestMethod + '\n' +
CanonicalURI + '\n' +
```

```
CanonicalQueryString + '\n' +
CanonicalHeaders + '\n' +
SignedHeaders + '\n' +
HexEncode(Hash(RequestPayload))
```

在此伪代码中： （1）Hash指代计算哈希的算法，目前使用SHA-256。 （2）HexEncode表示以小写字母形式返回摘要的 base-16 编码的函数。例如，HexEncode(“m”) 返回值 6d 而不是 6D。输入的每一个字节都必须表示为两个十六进制字符。 （3）参数含义说明 HTTPRequestMethod:HTTP正规化请求方法参数

- CanonicalURI：正规化URI参数
- CanonicalQueryString：正规化Query参数
- CanonicalHeaders：正规化Header参数
- SignedHeaders：正规化签名的Header参数
- HexEncode(Hash(RequestPayload))：正规化body参数

要创建正规化请求，请将以下来自每个步骤的部分连接为一个字符串：

1. 抽取HTTP请求方法（如GET、PUT、POST）结尾附加“换行符”。

2. 添加正规化URI参数，后跟换行符。

- 正规化URI是将绝对路径行URI编码.
- 如果绝对路径为空，那么使用前斜线"/"，结尾附加“换行符”。

3.添加正规化Query参数数，后跟换行符。

如果请求不包括查询字符串，请使用空字符串（实际上是空白行）。

要构建正规化查询字符串，请完成以下步骤： a.按照ASCII字节顺序对参数名称严格排序(具有重复名称的参数应按值进行排序)。 b.对Querystring的每个参数名称和值进行 URI 编码。（注：GET方式需要包含哈希算法、信任状、签名日期和签名header等全部参数） c.以排序后的列表中第一个参数名称开头，构造正规化查询字符串。 d.对于每个参数，追加 URI 编码的参数名称，后跟等号字符 (=)，再接 URI 编码的参数值。对没有值的参数使用空字符串。 e.在每个参数值后追加与字符(&)，列表中最后一个值除外。

4.添加正规化Header参数，后跟换行符。

正规化Header参数包括您要包含在签名请求中的所有 HTTP Header参数的列表。具体构造规则如下： a.请按照ASCII字节顺序对header名称严格排序。 b.请将所有Header参数名称转换为小写形式。 c.使用冒号（:）连接参数名称和参数值。参数值有多个时使用分号（“;”）来分隔。请勿对有多个值的Header进行值排序。 d.添加一个新行（“\n”）。

以下伪代码描述如何构造正规化Header参数列表：

```
CanonicalHeaders = CanonicalHeadersEntry0 + CanonicalHeadersEntry1 + ... + CanonicalHeadersEntryN
```

其中：

```
CanonicalHeadersEntry = Lowercase(HeaderName) + ':' + Trimall(HeaderValue) + '\n'
```

- Lowercase 表示将所有字符转换为小写字母的函数。
- Trimall 函数删除值前后的多余空格并将连续空格转换为单个空格，但是不去掉双引号中间的任何空格。

5.添加正规化签名的Header参数，后跟换行符。 该值是您包含在正规化Header参数中的Header列表。通过添加此Header参数列表，您可以向金山云告知请求中的哪些Header参数是签名过程的一部分以及在验证请求时金山云可以忽略哪些Header参数（例如，由代理添加的任何附加Header参数）。

其中如果header里存在host、x-amz-date，则必须添加进来。

具体构造规则如下：

a.请按照ASCII字节顺序对header名称严格排序。 b.请将所有Header参数名称转换为小写形式。 c.使用分号（“;”）来分隔这些Header参数名称。最后一个Header参数无需加分号。

以下伪代码描述如何构造签名Header参数列表。Lowercase 表示将所有字符转换为小写字母的函数。

```
SignedHeaders = Lowercase(HeaderName0) + ';' + Lowercase(HeaderName1) + ";" + ... + Lowercase(HeaderNameN)
```

6.添加正规化签名的Header参数。 对请求body使用哈希算法（SHA256）计算哈希值，必须以小写十六进制字符串形式表示。

如果body为空，则使用空字符串作为哈希函数的输入。

伪代码如下：

```
HashedPayload = Lowercase(HexEncode(Hash(requestPayload)))
```

7.将上诉1-6步骤中的结果连接成一个字符串，即为正规化请求。

步骤二：将正规化请求进行哈希处理

将步骤一中得到的正规划请求使用哈希算法（SHA256）计算哈希值，必须以小写十六进制字符串形式表示。

步骤三：创建签名字符串

签名字符串主要包含请求以及正规化请求的元数据信息，由签名算法、请求日期、信任状和正规化请求哈希值连接组成。 **待签字符串结构伪代码：**

```
StringToSign =
Algorithm + '\n' +
RequestDate + '\n' +
CredentialScope + '\n' +
HashedCanonicalRequest
```

其中：

- Algorithm：签名算法固定为AWS4-HMAC-SHA256
- RequestDate：请求日期格式为格式YYYYMMDD'T'HHMMSS'Z'
- CredentialScope：信任状格式为 *YYYYMMDD/region/service/*aws4_request（包括请求日期（ISO 8601 基本格式））
- HashedCanonicalRequest：正规化请求哈希值为上述步骤二的结果，注意结果不要附加换行符。

步骤四：计算签名字符串

在计算签名前，首先从私有访问密钥（secret AccessKey）派生出签名密钥（signing key）。派生签名密钥会指定日期、服务和区域，可以提供了更高程度的保护。

1.生成派生签名密钥

伪代码如下：

```
kSecret = *Your KSC Secret Access Key*
kDate = HMAC("AWS4" + kSecret, Date)
kRegion = HMAC(kDate, Region)
kService = HMAC(kRegion, Service)
kSigning = HMAC(kService, "aws4_request")
```

Access Key:可以为主账号或子用户的访问密钥（secret AccessKey） Date:请求的日期格式为YYYYMMDD（例如，20150830），不包括时间。 Region：要访问的目标区域。每个服务支持的region可能不同，详见各服务openapi文档说明。 Service:要访问的目标服务简称，一般可通过服务接入地址获取。接入地址格式一般为{service}.{region}.api.ksyun.com或{service}.api.ksyun.com。如访问控制的服务接入地址为访问控制的服务接入地址为：iam.api.ksyun.com，其中sevice为iam。

其中 HMAC(key, data) 表示以二进制格式返回输出的 HMAC-SHA256 函数。每个哈希函数的结果将成为下一个函数的输入。请注意：

- 哈希过程中所使用的日期的格式为 YYYYMMDD（例如，20150830），不包括时间。
- 确保以正确的顺序为您要使用的编程语言指定 HMAC 参数。在此示例中，密钥是第一个参数，数据（消息）是第二个参数，但您使用的函数可能以不同顺序指定密钥和数据。
- HMAC算法采用HMAC-SHA256，返回值为哈希值二进制形式（256bit，32字节），不需要做8/16进制编码显示。

示例输入：

```
HMAC(HMAC(HMAC(HMAC("AWS4" + kSecret,"20150830"),"cn-beijing-6"),"iam"),"aws4_request")
```

示例派生签名密钥：

```
c4afb1cc5771d871763a393e44b703571b55cc28424d1a5e86da6ed3c154a4b9
```

步骤五：计算签名

请使用派生的签名密钥和待签字符串作为加密哈希函数的输入。在计算签名后，将二进制值转换为十六进制表示形式。

伪代码如下：

```
signature = HexEncode(HMAC(derived-signing-key, string-to-sign))
```

确保以正确的顺序为您要使用的编程语言指定 HMAC 参数。在此示例中，密钥是第一个参数，数据（消息）是第

二个参数，但您使用的函数可能以不同顺序指定密钥和数据。

示例签名：

```
5d672d79c15b13162d9279b0855cfba6789a8edb4c82c400e06b5924a6f2b5d7
```

函数调用

云函数从调用方式上可以分为两类：

- 同步调用：调用请求被函数处理后会直接返回执行结果，适用于Web服务、实时流处理等业务场景。
- 异步调用：调用请求加入请求队列中就返回，不会等待函数执行结果，适用于批量数据处理等业务场景。

根据您对函数配置的触发器，函数被置为不同的请求处理类型，和调用方式的对应关系见下表：

函数请求处理类型	支持的触发器类型	调用方式
处理事件请求	对象存储触发器	异步调用
处理HTTP请求	HTTP触发器	同步调用

HTTP请求函数

应用场景

当您同步调用一个函数时，事件将直接触发函数，云函数会运行该函数并等待响应。适用于由HTTP请求触发，或需及时查看执行结果的场景。

使用限制

- 当前HTTP请求函数仅支持配置一个HTTP触发器。
- 资源调用限制：当前KCF限制单个账号在单个地域内函数实例的上限数为300，相应对每个函数的限流配置为单实例并发数*300。

事件请求函数

应用场景

金山云云函数与云产品事件源进行了集成，通过对应的触发器能够触发相关函数执行。云函数系统在接收异步调用请求后，会将请求持久化并立即返回响应，而不是等待请求执行完成后再返回。对于不需要查询每个执行结果，任务量多且密集，存在服务调用关系的业务场景，可以使用事件请求函数。

重试策略

当事件请求执行失败时，云函数系统可进行错误重试，您可在函数配置>**执行配置**中对异步调用的重试规则进行自定义配置。

异步调用重试：开启后将支持异步请求的错误重试

- 重试次数：设置最大重试次数，超过设置值后，异步触发失败的调用将不再重试。默认值为2，支持范围为0-3次。
- 重试间隔：设置函数重试执行的时间间隔。默认值为60秒，支持范围为60-120秒。
- 最长保留时间：设置函数的异步事件队列中，事件保留的最长时间，超过此时间的事件将被丢弃。默认事件为2小时，支持时间范围为1分钟-6小时。

弹性管理概述

概述

云函数的算力资源由底层具备弹性扩缩能力的函数实例构成，函数启动和弹性伸缩的响应效率影响着业务并发的执行效率。为了提升函数在冷启动和并发波动场景的执行效率，金山云云函数提供关于预留实例和弹性策略的灵活配置。您可通过弹性管理基础配置和多种自定义弹性策略，定义一套符合业务场景需求的弹性规则。

什么是冷启动

冷启动指的是函数生命周期中，函数实例被完全释放后，新的函数实例第一次被调用后启动的过程。此过程包含函数调用链路中函数实例调度、代码下载、函数实例启动、运行时初始化、代码初始化等环节带来的耗时。

如下图所示，如请求2在函数实例被释放前发生，由于函数执行环境已经准备好，请求2将无需等待函数实例的冷启动时间，在已有的函数实例环境中被执行。如请求3在函数实例被释放后发生，函数执行环境被释放，请求3需等待一个新的函数实例的冷启动时间。

为解决冷启动带来的延迟毛刺，您可以从弹性配置上降低冷启动发生概率，比如：

- 配置预留实例：预留实例将提前准备好函数执行环境，且不会被云函数平台主动回收。当请求发生时，调用请求将优先调度至预留实例，降低冷启动的发生次数。
- 配置定时弹性策略：对于有时间规律的业务场景，定时弹性策略可以帮助预热函数，减少冷启动的发生。

弹性管理

函数的生命周期中，调用请求量通常处于波动状态。处理函数调用请求时，云函数会优先调度已有的可用实例，在当前实例不足以处理调用请求时扩容新的实例以进行补充，同时在请求减少时缩容冗余的实例以节省资源消耗。金山云云函数为弹性管理提供灵活的配置，您可以通过基础配置和弹性策略两个模块，为函数配置弹性范围内的预留和弹性策略。

弹性管理配置

弹性管理基础配置

最小实例数-最大实例数：触发弹性后，函数实例将在最小实例数和最大实例数范围内进行扩缩容。

> 注：因在账号下地域维度存在函数实例总数限制，您可结合业务并发需求和各函数间资源分配规划，为函数配置实例数上限。

预留实例数：保持常驻的实例数量。此部分实例将按配置预先启动，且云函数平台不会主动回收这些实例，降低冷启动耗时。预留实例数范围需属于[最小实例数，最大实例数]，默认值为0。

缩容稳定窗口期：为保证业务执行的稳定性，在执行弹性伸缩动作后，指定窗口时间内不会再进行缩容操作，缩容稳定窗口期默认值为30s。

冷却时间：为提升资源利用率，无弹性伸缩策略命中时，等待指定冷却时间后，实例数缩减为预留实例数。冷却时间默认值为150s。

弹性策略

云函数提供了多种弹性策略，应用于不同函数类型和弹性场景：

- **队列堆积指标**：事件类函数的默认弹性策略，与事件类触发器同步生效/销毁。在事件类请求中，根据队列堆积度判断在当前函数实例的处理能力下异步事件在队列中的积压程度，达到指标设定值时自动扩缩容。
- **并发指标**：HTTP请求类函数的默认弹性策略，与HTTP触发器同步生效/销毁。在HTTP请求中，根据HTTP请求并发度判断当前函数并发处理能力下是否需要扩缩容。
- **定时弹性伸缩**：定时弹性策略不依赖函数类型/触发器，可使函数实例在指定时间范围，从0扩容至期望实例数。
- **内存指标弹性伸缩**：内存指标弹性策略不依赖函数类型/触发器，除将请求处理情况作为弹性伸缩指标，函数实例的内存使用量/使用率也可以反映当前实例处理能力是否充足，达到指标设定值时自动扩缩容。

事件函数：队列堆积指标

当函数配置事件类触发器时，函数将默认创建一个持续生效的队列堆积指标策略：在当前函数处理能力下，队列堆积度将以80%为临界值进行扩缩容。其中可配置参数如下：

- 单实例吞吐量：单实例吞吐量展示了单实例并发处理请求的能力，默认值为函数的单实例并发数。
- 队列堆积度：队列堆积度反映了当前函数实例处理请求时事件的堆积程度，事件的入队速率与消费速率的差值越大，队列堆积度越高。在单实例吞吐量稳定的基础上，函数将通过扩容实例的方式提升消费速率，您可以通过设置队列堆积度期望值，来控制函数实例的扩容阈值。在一个扩缩容周期内，当事件的入队速率较高，使队列堆积度超出期望值时，将触发函数实例扩容，反之则触发函数实例缩容。

  算法示例：目标实例数=当前函数总实例数×当前队列堆积度÷队列堆积度期望值

  如队列堆积度期望值配置为50%，当前函数实例数为20、队列堆积度为80%，则扩容后目标实例数=20×80÷50=32

队列堆积指标作为事件类触发器的默认指标，生效时间范围默认为持续生效，仅支持随触发器删除。

HTTP函数：并发指标

当函数配置HTTP请求触发器时，函数将默认创建一个持续生效的并发指标触发器：在函数的并发请求数增加/减少至300次每分钟时，对函数实例进行扩缩容。您可对默认的并发请求数进行调整：

- 并发请求数：您可设置不同单位时间内（支持分钟/秒）并发请求数阈值。当业务请求量不断增加，触发扩容阈值时函数开始扩容，请求量减少至低于阈值且满足缩容稳定窗口期时函数开始缩容。

并发指标作为HTTP触发器的默认指标，生效时间范围默认为持续生效，仅支持随触发器删除。

定时弹性伸缩

定时弹性伸缩不依赖触发器，支持函数实例从0开始扩容，适用于有明确潮汐效应的业务场景。

- 定时类型：支持每天，每周一-周五，每周六-周日的指定时间范围内重复执行，也支持自定义模式选择“xx年xx月xx日xx时xx分xx秒”特定时间范围内执行。
- 时间范围：定时类型选择为周期重复执行时，时间范围支持配置至“xx时xx分xx秒”；定时类型选择为自定义模式时，时间范围支持配置至“xx年xx月xx日xx时xx分xx秒”。
- 期望实例数：配置时间范围内的期望实例数，数值范围需属于[最小实例数，最大实例数]。

内存指标弹性伸缩

内存指标弹性伸缩需配置触发器才生效，不依赖触发器类型。对于内存敏感型的任务，您可配置内存指标类伸缩策略，根据内存使用量/使用率来进行函数实例的扩缩容。

- 内存指标：支持选择内存使用率/内存使用量指标，为函数配置内存使用率/使用量阈值。参考为函数配置的实例规格，此处内存使用量配置不能大于实例规格。
- 时间范围：配置内存指标策略的生效时间，支持选择**持续生效**或**自定义时间**。

  算法示例：目标实例数=当前函数总实例数×当前内存使用率÷内存使用率阈值

  如内存使用率阈值配置为80%，当前函数实例数为20、内存使用率为20%，缩容后目标实例数=20×20÷80=5

  注意：因内存指标特殊性，配置后函数将在最小实例数和最大实例数范围内扩缩容，不会缩容至0。

多弹性策略组合

基于函数请求类型和业务场景，您可以组合使用多种弹性策略，使函数具备灵活的伸缩性。典型的使用场景包括但不限于：

- 配置预留实例及定时弹性策略：在有明确高峰时段的业务场景，配置预留实例用于处理业务低峰期发生的少量请求，高峰期根据定时弹性策略提前扩容函数实例以应对突增请求，保证函数在低峰期的冷启动效率和高峰期的处理能力。
- 配置默认弹性策略及内存指标弹性策略：函数将根据对应请求类型的默认策略进行扩缩容，除了根据队列堆积度和并发请求数进行扩缩容外，在函数实例的内存使用量/使用率超出阈值时也进行扩缩容，保证函数实例的内存资源充足。

  在多个弹性策略同时命中时，函数实例将以扩缩容后实例数最多的弹性策略为准，执行扩缩容动作。

函数监控

目前云函数默认提供函数维度下表中监控指标，您可进入函数详情中**监控信息**页签进行查询。

指标名称	单位	描述
函数总调用次数（FunctionTotalInvocations）	次	函数总调用次数。按时间粒度统计求和。
客户端错误（FunctionClientErrors）	次	在调用某个指定函数时，由于云函数客户端原因导致函数未被执行的总调用次数。按时间粒度求和。
服务端错误（FunctionServerErrors）	次	在调用某个指定函数时，由于云函数服务端原因导致函数未被执行的总调用次数。按时间粒度统计求和。
函数错误（FunctionFunctionErrors）	次	在调用某个指定函数时，由于函数自身原因导致函数调用失败的次数。按时间粒度统计求和。
函数执行平均时间（FunctionExecutionAvg）	毫秒	函数代码从执行开始到结束的时间。时间粒度内统计平均值。
函数执行最大时间（FunctionExecutionMax）	毫秒	函数代码从执行开始到结束的时间。时间粒度内统计最大值。

端到端平均耗时（FunctionLatencyAvg）	毫秒	在调用时，函数执行请求从抵达云函数系统开始到离开云函数系统所消耗的时间，且包含平台消耗的时间。时间粒度内统计平均值。
端到端最大耗时（FunctionLatencyMax）	毫秒	在调用函数时，函数执行请求从抵达云函数系统开始到离开云函数系统所消耗的时间，且包含平台消耗的时间。时间粒度内统计最大值。
异步调用请求入队（FunctionEnqueueCount）	次	在调用函数时，函数异步调用时，入队请求次数。按时间粒度统计求和。
异步调用请求处理完成（FunctionDequeueCount）	次	在调用函数时，函数异步调用时，处理完成的总请求次数。按时间粒度统计求和。 注：当请求处理完成数远小于入队请求数时，将导致消息积压，请调整函数并发度。
异步消息平均处理延时（FunctionAsyncMessageLatencyAvg）	毫秒	函数异步调用时，从消息入队开始到处理完成为止的平均延时。时间粒度内统计平均值。
异步消息最大处理延时（FunctionAsyncMessageLatencyMax）	毫秒	函数异步调用时，从消息入队开始到处理完成为止的延时。时间粒度内统计最大值。

日志管理

当前云函数支持将函数调用执行的日志投递至金山云日志服务KLog进行存储与检索分析，您可在创建函数时开启**使用日志服务KLog**，指定函数日志的消费端，在KLog中由日志项目+日志池构成。

配置日志采集

使用已有日志实例：如果您已为函数日志规划好明确的日志池和所属日志项目，可在使用已有日志实例模式下选择，或通过**新建日志项目与日志池**，跳转至KLog控制台创建好日志实例后，再选择具体日志项目与日志池。

自动新建日志实例：若无需指定特定日志项目和日志池，您可以选择自动新建日志实例，后台将自动在KLog服务中创建名为kcf-log-{function-id}的日志项目和名为{function-id}的日志池。

> 注：自动新建的日志池，日志分区数默认为2，日志保存时长默认为14天。若您有调整日志分区数和日志保存时长的需要，可在日志实例创建后，到日志服务控制台，进入**日志项目>日志池**，选择**编辑**日志池。对日志池的分区数与保存时长进行调整。

查看调用日志

在函数详情页面，选择**调用信息**页签，查询当前函数的调用日志。

调用请求列表

调用列表包含每一次调用请求的调用时间、Request ID、调用结果、执行时间等关键信息。

- **查询日志**：您可通过指定**时间范围**/**Request ID**，查询关键日志。
- **查看日志详情**：点击**日志详情**，可查看调用请求和日志详情。
- **查看原始日志**：点击列表上方的**查看原始日志**，可以跳转至日志服务控制台查询和分析原始日志。