ALIBABA CLOUD

阿里云

函数计算

代码开发

文档版本：20200915

阿里云

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 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.开发语言列表

函数计算支持多种语言开发，本文列出了目前支持的开发语言。

语言类型	相关文档
Node.js	Node.js运行环境
Python	Python运行环境
PHP	PHP运行环境
Java	Java运行环境
C#	.NET Core运行环境
Go	Go Custom Runtime
Ruby	Ruby Custom Runtime
PowerShell	PowerShell Custom Runtime
TypeScript	TypeScript Custom Runtime
F#	F# Custom Runtime
其他语言	Custom Runtime简介

2.基本概念

本文介绍了使用函数计算编写代码过程中经常遇到的基本概念，包括函数入口、入口函数、函数入参、日志记录和错误处理。

函数入口

在创建函数时，需要指定函数入口，函数计算会从这个函数入口开始执行。函数入口的格式为 `[文件名].[函数名]` 。

以Node.js为例，创建函数时指定的Handler为*index.handler*，那么函数计算会去加载index.js中定义的handler函数。

入口函数

函数入口指定的函数叫做入口函数，类似于本地开发中的main()函数。入口函数需要满足函数计算提供的编程模型。

入口函数分为事件函数与HTTP函数：

- 事件函数是普通的入口函数，是一段业务逻辑代码。除了HTTP函数的入口函数都是事件函数。
- HTTP函数是设置了HTTP触发器的函数，可以直接处理HTTP Request并返回HTTP Response，适用于搭建Web应用。编程模型与事件函数不同。

以Python运行环境的事件函数为例：

```
def handler(event, context):
  return 'hello world'
```

模型中指定固定的event事件数据（用户自定义）和context环境上下文（平台定义）作为函数入参。 在入口函数中，需对参数进行处理，并且可调用代码中定义的其他函数。 除了入口函数的定义外，代码的组织逻辑与本地函数相同。

各个编程语言的入口函数模型详情请参考对应的编程语言。

Initializer函数

函数计算平台会根据请求动态分配实例执行函数，当函数有连续调用时，会复用同一实例。Initializer是初始化函数，保证在同一实例中执行且成功执行一次。

您可以将数据库场景下连接池构建、函数依赖库加载等耗时长的业务逻辑放到Initializer函数中，避免每次运行函数都会做重复的操作，降低函数延时。

Initializer函数在函数计算平台分配实例后执行，早于入口函数的执行。虽然HTTP函数与事件函数的编程模型不同，但他们对应的Initializer函数的编程模型都是相同的。

各个编程语言的Initializer函数模型详情请参考对应的编程语言。

函数入参

函数入参是指函数在调用时所传递给函数的内容。通常情况下，函数入参包括event入参和context入参两部分，但是由于开发语言和环境的不同，这两部分中的入参个数可能会有所不同。

- event参数

event参数是用户自定义的函数入参，以字节流的形式传给函数，数据结构由您自行定义，它可以是一个简单的字符串 、一个JSON对象、一张图片（二进制数据）。函数计算不对event参数的内容进行任何解释，您需要在函数中将字节流转换成相应的类型。

对于不同的函数触发情况，event参数的值会有以下区别：

- 事件源服务触发函数时，事件源服务会将事件以一种平台预定义的格式作为event参数传给函数，您可以根据此格式编写代码并从event参数中获取信息。例如使用OSS触发器触发函数时会将Bucket及文件的具体信息以JSON格式传递给event参数。
- 函数通过SDK直接调用时，您可以在调用方和函数代码之间自定义event参数。调用方按照定义好的格式传入数据，函数代码按格式获取数据。例如定义一个JSON类型的数据结构 `{"key":"val"}` 作为 `event` ，当调用方传入数据 `{"key":"val"}` 时，函数代码先将字节流转换成JSON，再通过 `event["key"]` 来获得值 `val` 。

下文以Python为例，演示event参数的使用。

```
import json
def handler(event, context):
  evt = json.loads(event)
  print(evt['key'])
  return 'success'
```

- context参数

  context参数是函数计算平台定义的函数入参，它的数据结构由函数计算设计，包含函数运行时的信息，函数结构如下所示。

  - 函数结构

```
{
  requestId: '9cda63c3-1ac9-45ba-8a59-2593bb9bc101',
  credentials: {
    accessKeyId: 'xxx',
    accessKeySecret: 'xxx',
    securityToken: 'xxx'
  },
  function: {
    name: 'xxx',
    handler: 'index.handler',
    memory: 512,
    timeout: 60,
    initializer: 'index.initializer',
    initializationTimeout: 10
  },
  service: {
    name: 'xxx',
    logProject: 'xxx',
    logStore: 'xxx',
    qualifier: 'xxx',
    versionId: 'xxx'
  },
  region: 'xxx',
  accountId: 'xxx'
}
```

- 使用场景
  - 用户的临时密钥信息可以通过 `context.credentials` 获取，通过 `context` 中的临时密钥去访问阿里云的其他服务（使用示例中以访问OSS为例），避免了在代码中使用密钥硬编码。
  - 在 `context` 中可以获取本次执行的基本信息，例如requestId、serviceName、functionName、qualifier等。

- 使用示例

  下文以Python为例，演示使用 `context` 中的个人信息访问OSS的过程。

```
import json
import oss2
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, 'oss-endpoint', 'your-bucket-name')
  bucket.put_object('object-name', 'Awesome FC')
  return 'success'
```

日志记录

函数计算与日志记录集成，函数计算会将函数调用的记录以及函数代码中打印的日志全部存储到日志库中，您可以使用函数计算提供的日志语句记录函数日志，方便调试及定位问题。

> **说明**
> - 您需要在Service级别配置日志库，函数计算将函数日志发送到指定日志库中。
> - 通过控制台创建的函数、服务会为您自动创建并配置日志库。

开发语言	编程语言内嵌的打印日志语句	函数计算提供的日志语句	参考文档
Node.js	console.log()	console.log()	打印日志
Python	print()	logging.getLogger().info()	打印日志
Java	System.out.println()	context.getLogger().info()	打印日志
PHP	echo "" . PHP_EOL	$GLOBALS['fcLogger']->info()	打印日志
C#	Console.WriteLine("")	context.Logger.LogInformation()	打印日志

使用编程语言内嵌的打印输出语句打印的日志也会被收集到日志库里，而使用函数计算提供的日志语句打印的每条日志前都会带上请求ID，方便日志的筛选。

```
#使用编程语言内嵌的打印输入语句打印的日志
# print('hello world')
message：hello world

#使用函数计算提供的日志语句打印的日志
# logger.info('hello world')
message：2020-03-13T04:06:49.099Z f84a9f4f-2dfb-41b0-9d6c-1682a2f3a650 [INFO] hello world
```

日志组成

函数运行日志包括服务名称、函数名称、当前执行版本、别名、代码日志。

函数运行日志数据结构如下所示。

```
__source__:
__tag__:__receive_time__：1584072413
__topic__：myService
functionName：myFunction
message：2020-03-13T04:06:49.099Z f84a9f4f-2dfb-41b0-9d6c-1682a2f3a650 [INFO] hello world
qualifier：LATEST
serviceName：myService
versionId:
```

- 在函数开始执行时，系统会打印 `FC Invoke Start RequestId: f84a9f4f-2dfb-41b0-9d6c-1682a2f3a650` ，标志函数执行开始。
- 在函数执行完成时，系统会打印 `FC Invoke End RequestId: f84a9f4f-2dfb-41b0-9d6c-1682a2f3a650` ，标志函数执行结束。

错误处理

函数计算的错误类型有两种，分别是 `HandledInvocationError` 和 `UnhandledInvocationError` 。

- HandledInvocationError

  只有在Node.js中通过 `callback` 返回的错误是 `HandledInvocationError` ，错误信息在响应内容中。

  ```
  'use strict';
   module.exports.handler = function(event, context, callback) {
    console.log('hello world');
    callback('this is error', 'hello world');
   }
  ```

  响应内容如下所示。

```
{"errorMessage":"this is error"}
```

- UnhandledInvocationError

  除了 `HandledInvocationError` ，其余的错误都是 `UnhandledInvocationError` 。

  `UnhandledInvocationError` 的错误 `stackTrace` 会打印到日志中，您可以进入日志查看上下文，找到对应的 `stackTrace` 。

3.Custom Runtime

3.1. F#

通过函数计算的Custom Runtime + HTTP Trigger的方式，可以将F#的ASP.NETCore项目一键迁移，并可直接使用浏览器或者curl等HTTP客户端工具访问函数。

前提条件

您已完成以下操作：

- 安装8.6.0及以上版本的Node.js。详情请参见官方下载地址。
- 安装Docker。详情请参见Dockerhub。

> 说明
> - 本文提供的示例适用于安装Docker的场景。如果您不想使用Docker，那么您需要安装.NET Core 3.1。安装详情请参见dot.net.core3.1。运行命令详情请参见fc-custom-demo
> - 如果您已安装Funcraft可直接跳转至步骤二：部署和调用函数。

步骤一：准备环境

安装Funcraft，最简单的方式就是直接下载可执行的二进制文件。

1. 安装Funcraft到本机。详情请参见安装Funcraft。
2. 执行 `fun --version` 检查安装是否成功。
3. 执行 `fun config` 配置Funcraft。然后按照提示依次配置Account ID、Access Key ID、Access Key Secret、Default region name。

```
fun config
Aliyun Account ID 1234xxx
Aliyun Access Key ID xxxx
Aliyun Access Key Secret xxxx
Default region name cn-xxxx
The timeout in seconds for each SDK client invoking 300
The maximum number of retries for each SDK client 5
Allow to anonynously report usage statistics to improve the tool over time? (Y/n)
```

步骤二：部署和调用函数

1. 执行以下命令克隆示例工程到本地。

```
git clone https://github.com/awesome-fc/fc-custom-demo
```

> 说明 如果您没有安装Git，可以直接在浏览器地址栏输入https://github.com/awesome-fc/fc-custom-demo，然后单击下载按钮直接下载。

2. 执行以下命令进入克隆的示例项目中。

```
cd fc-custom-demo
cd FSharp-demo
```

3. 执行以下命令部署函数。

```
make deploy
```

执行结果如下。

```
docker run -it -v $(pwd):/tmp mcr.microsoft.com/dotnet/core/sdk:3.1 bash -c "cd /tmp/FSharpDem
o && dotnet publish -r linux-x64 -c Release --self-contained true && cd /tmp/FSharpDemo/bin/Rel
ease/netcoreapp3.1/linux-x64/publish && mv FSharpDemo bootstrap && chmod +x bootstrap"
Microsoft (R) Build Engine version 16.5.0+d4cbfca49 for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

  Restore completed in 15.2 sec for /tmp/FSharpDemo/FSharpDemo.fsproj.
  FSharpDemo -> /tmp/FSharpDemo/bin/Release/netcoreapp3.1/linux-x64/FSharpDemo.dll
  FSharpDemo -> /tmp/FSharpDemo/bin/Release/netcoreapp3.1/linux-x64/publish/
fun deploy -y
...
Waiting for service fsharp_demo to be deployed...
    Waiting for function fc_fsharp to be deployed...
        Waiting for packaging function fc_fsharp code...
        The function fc_fsharp has been packaged. A total of 338 files were compressed and the
final size was 39.3 MB
        Waiting for HTTP trigger http_t to be deployed...
        triggerName: http_t
        methods: [ 'GET', 'POST', 'PUT', 'DELETE' ]
        url: https://19861144305****.cn-hangzhou.fc.aliyuncs.com/2016-08-15/proxy/fsharp_demo
/fc_fsharp/
        Http Trigger will forcefully add a 'Content-Disposition: attachment' field to the response
header, which cannot be overwritten
        and will cause the response to be downloaded as an attachment in the browser. This iss
ue can be avoided by using CustomDomain.

        trigger http_t deploy success
    function fc_fsharp deploy success
service fsharp_demo deploy success

Detect 'DomainName:Auto' of custom domain 'my_domain'
Fun will reuse the temporary domain 16225220-19861144****.test.functioncompute.com, expired at
2020-04-17 10:07:00, limited by 1000 per day.

Waiting for custom domain my_domain to be deployed...
custom domain my_domain deploy success
```

4. 执行以下命令调用部署的函数。

   在该示例中，直接通过curl或浏览器访问部署成功后输出的临时域名 `16225220-198611443****.test.functioncompute.com` 。

```
curl 16225220-1986114430****.test.functioncompute.com/weatherforecast
```

执行结果如下。

```
[{"date":"2020-04-07T07:46:29.4315198+00:00","temperatureC":49,"summary":"Hot","temperatureF":120},{"date":"2020-04-08T07:46:29.431527+00:00","temperatureC":11,"summary":"Bracing","temperatureF":51},{"date":"2020-04-09T07:46:29.4315276+00:00","temperatureC":45,"summary":"Bracing","temperatureF":112},{"date":"2020-04-10T07:46:29.431528+00:00","temperatureC":13,"summary":"Chilly","temperatureF":55},{"date":"2020-04-11T07:46:29.4315284+00:00","temperatureC":-8,"summary":"Mild","temperatureF":18}]
```

3.2. Ruby

通过函数计算的Custom Runtime，您可以在函数计算平台使用Ruby来编写函数。本文介绍快速部署和调用Ruby函数的详细步骤。

前提条件

您已安装8.6.0及以上版本的Node.js。详情请参见官方下载地址。

> **说明** 如果您已安装Funcraft可直接跳转至步骤二：部署和调用函数。

步骤一：准备环境

安装Funcraft，最简单的方式就是直接下载可执行的二进制文件。

1. 安装Funcraft到本机。详情请参见安装Funcraft。
2. 执行 `fun --version` 检查安装是否成功。
3. 执行 `fun config` 配置Funcraft。然后按照提示依次配置Account ID、Access Key ID、Access Key Secret、Default region name。

```
fun config
Aliyun Account ID 1234xxx
Aliyun Access Key ID xxxx
Aliyun Access Key Secret xxxx
Default region name cn-xxxx
The timeout in seconds for each SDK client invoking 300
The maximum number of retries for each SDK client 5
Allow to anonynously report usage statistics to improve the tool over time? (Y/n)
```

步骤二：部署和调用函数

1. 执行以下命令克隆示例工程到本地。

```
git clone https://github.com/awesome-fc/fc-custom-demo
```

> 说明 如果您没有安装Git，可以直接在浏览器地址栏输入https://github.com/awesome-fc/fc-custom-demo，然后单击下载按钮直接下载。

2. 执行以下命令进入克隆的示例项目中。

```
cd fc-custom-demo
cd ruby-demo
```

3. 执行以下命令将项目部署至函数计算。

```
fun deploy -y
```

返回结果如下。

```
...
Waiting for service ruby-demo to be deployed...
    Waiting for function fc-ruby to be deployed...
        Waiting for packaging function fc-ruby code...
        The function fc-ruby has been packaged. A total of 4 files were compressed and the final size was 1.45 KB
    function fc-ruby deploy success
service ruby-demo deploy success
```

4. 执行以下命令调用部署的函数。

```
fun invoke -e "Hello World"
```

返回结果如下。

```
...
========= FC invoke Logs begin =========
FC Invoke Start RequestId: a9c4dc8a-4b5b-48e0-9a3f-70310531ae61
Hello World
FC Invoke End RequestId: a9c4dc8a-4b5b-48e0-9a3f-70310531ae61

Duration: 0.69 ms, Billed Duration: 100 ms, Memory Size: 512 MB, Max Memory Used: 5.06 MB
========= FC invoke Logs end =========

FC Invoke Result:
Hello World
```

3.3. TypeScript

通过函数计算的Custom Runtime，您可以在函数计算平台使用TypeScript来编写函数。本文介绍快速部署和调用TypeScript函数的详细步骤。

前提条件

您已安装8.6.0及以上版本的Node.js。详情请参见官方下载地址。

> 说明 如果您已安装Funcraft可直接跳转至步骤二：部署和调用函数。

步骤一：准备环境

安装Funcraft，最简单的方式就是直接下载可执行的二进制文件。

1. 安装Funcraft到本机。详情请参见安装Funcraft。
2. 执行 `fun --version` 检查安装是否成功。
3. 执行 `fun config` 配置Funcraft。然后按照提示依次配置Account ID、Access Key ID、Access Key Secret、Default region name。

```
fun config
Aliyun Account ID 1234xxx
Aliyun Access Key ID xxxx
Aliyun Access Key Secret xxxx
Default region name cn-xxxx
The timeout in seconds for each SDK client invoking 300
The maximum number of retries for each SDK client 5
Allow to anonynously report usage statistics to improve the tool over time? (Y/n)
```

步骤二：部署和调用函数

1. 执行以下命令克隆示例工程到本地。

```
git clone https://github.com/awesome-fc/fc-custom-demo
```

> 说明 如果您没有安装Git，可以直接在浏览器地址栏输入https://github.com/awesome-fc/fc-custom-demo，然后单击下载按钮直接下载。

2. 执行以下命令进入克隆的示例项目中。

```
cd fc-custom-demo
cd ts-demo
```

3. 执行以下命令将项目部署至函数计算。

```
fun deploy -y
```

返回结果如下。

```
...
Waiting for service ts-demo to be deployed...
    Waiting for function fc-ts to be deployed...
        Waiting for packaging function fc-ts code...
        The function fc-ts has been packaged. A total of 336 files were compressed and the final size was 9.41 MB
    function fc-ts deploy success
service ts-demo deploy success
```

4. 执行以下命令调用部署的函数。

```
fun invoke -e "Hello World"
```

返回结果如下。

```
...
========= FC invoke Logs begin =========
FC Invoke Start RequestId: 7ab0a86a-be32-4086-ac17-3ce0797cda41
Hello World
FC Invoke End RequestId: 7ab0a86a-be32-4086-ac17-3ce0797cda41

Duration: 13.48 ms, Billed Duration: 100 ms, Memory Size: 512 MB, Max Memory Used: 162.38 MB
========= FC invoke Logs end =========

FC Invoke Result:
Hello World
```

3.4. PowerShell

通过函数计算的Custom Runtime，您可以在函数计算平台使用PowerShell来编写函数。本文介绍快速部署和调用PowerShell函数的详细步骤。

前提条件

您已安装8.6.0及以上版本的Node.js。详情请参见官方下载地址。

> 说明 如果您已安装Funcraft可直接跳转至步骤二：部署并调用函数。

步骤一：准备环境

安装Funcraft，最简单的方式就是直接下载可执行的二进制文件。

1. 安装Funcraft到本机。详情请参见安装Funcraft。
2. 执行 `fun --version` 检查安装是否成功。
3. 执行 `fun config` 配置Funcraft。然后按照提示依次配置Account ID、Access Key ID、Access Key

Secret、Default region name。

```
fun config
Aliyun Account ID 1234xxx
Aliyun Access Key ID xxxx
Aliyun Access Key Secret xxxx
Default region name cn-xxxx
The timeout in seconds for each SDK client invoking 300
The maximum number of retries for each SDK client 5
Allow to anonynously report usage statistics to improve the tool over time? (Y/n)
```

步骤二：部署并调用函数

1. 执行以下命令克隆示例工程到本地。

```
git clone https://github.com/awesome-fc/fc-custom-demo
```

> 说明 如果您没有安装Git，可以直接在浏览器地址栏输入https://github.com/awesome-fc/fc-custom-demo，然后单击下载按钮直接下载。

2. 执行以下命令进入克隆的示例项目中。

```
cd fc-custom-demo
cd powershell-demo
```

3. 执行以下命令安装依赖。

```
fun install -v
```

返回结果如下。

```
start installing function dependencies without docker

building powershell-demo/fc-powershell
Funfile exist, Fun will use container to build forcely
Step 1/7 : FROM registry.cn-beijing.aliyuncs.com/aliyunfc/runtime-custom:build-1.9.4
...
Step 7/7 : RUN fun-install apt-get install powershell
...
sha256:7c6476a3a4496bbf3da83bfb8966acc90fa94f4f8baccd248cd79c18c4fae409
Successfully built 7c6476a3a449
Successfully tagged fun-cache-965bbabb-ab6f-44e7-9910-7dd1df1d3c3e:latest
copying function artifact to /Users/songluo/gitpro/fc-custom-demo/powershell-demo
copy from container /mnt/auto/. to localNasDir

Install Success
```

4. 执行以下命令将项目部署至函数计算。

```
fun deploy -y
```

返回结果如下。

```
....
? Do you want to let fun to help you automate the configuration? Yes
? You have already configured 'NasConfig: Auto'. We want to use this configuration to store your function dependencies. Yes

Fun automatically backups the original template.yml file to /Users/txd123/Desktop/jack/fc-custom-demo/powershell-demo/.template.yml.backup
Fun add .fun/root to /Users/txd123/Desktop/jack/fc-custom-demo/powershell-demo/.nas.yml
Fun add environment variables to 'powershellDemo/fc-powershell' in /Users/txd123/Desktop/jack/fc-custom-demo/powershell-demo/template.yml

starting upload /Users/txd123/Desktop/jack/fc-custom-demo/powershell-demo/.fun/root to nas://powershellDemo/mnt/auto/root/

start fun nas init...
....
fun nas init Success

zipping /Users/txd123/Desktop/jack/fc-custom-demo/powershell-demo/.fun/root
✔ upload done
unzipping file
✔ unzip done
✔ upload completed!

....
? Region cn-qingdao only supports capacity NAS. Do you want to create it automatically? Yes
    ....
  Waiting for function fc-powershell to be deployed...
    Waiting for packaging function fc-powershell code...
    The function fc-powershell has been packaged. A total of 3 files were compressed and the final size was 1.31 KB
  function fc-powershell deploy success
  function fc-powershell deploy success
service powershellDemo deploy success
```

5. 执行以下命令调用部署的函数。

```
fun invoke -e "Hello World"
```

返回结果如下。

```
...
========= FC invoke Logs begin =========
FC Invoke Start RequestId: cd30369e-7dfa-439c-a68d-7fe16d5a7e05
Hello World
FC Invoke End RequestId: cd30369e-7dfa-439c-a68d-7fe16d5a7e05

Duration: 54.13 ms, Billed Duration: 100 ms, Memory Size: 512 MB, Max Memory Used: 133.70 MB
========= FC invoke Logs end =========

FC Invoke Result:
Hello World
```

3.5. Go

通过函数计算的Custom Runtime，可以将Go项目一键部署至函数计算。本文介绍快速部署和调用函数的详细步骤。

前提条件

您已完成以下操作：

- 安装8.6.0及以上版本的Node.js。详情请参见官方下载地址。
- 安装Docker。详情请参见Dockerhub。

> **说明** 如果您已安装Funcraft可直接跳转至步骤二：部署和调用函数。

步骤一：准备环境

安装Funcraft，最简单的方式就是直接下载可执行的二进制文件。

1. 安装Funcraft到本机。详情请参见安装Funcraft。
2. 执行 `fun --version` 检查安装是否成功。
3. 执行 `fun config` 配置Funcraft。然后按照提示依次配置Account ID、Access Key ID、Access Key Secret、Default region name。

```
fun config
Aliyun Account ID 1234xxx
Aliyun Access Key ID xxxx
Aliyun Access Key Secret xxxx
Default region name cn-xxxx
The timeout in seconds for each SDK client invoking 300
The maximum number of retries for each SDK client 5
Allow to anonynously report usage statistics to improve the tool over time? (Y/n)
```

步骤二：部署和调用函数

1. 执行以下命令克隆示例工程到本地。

```
git clone https://github.com/awesome-fc/fc-custom-demo
```

> 说明 如果您没有安装Git，可以直接在浏览器地址栏输入https://github.com/awesome-fc/fc-custom-demo，然后单击下载按钮直接下载。

2. 执行以下命令进入克隆的示例项目中。

```
cd fc-custom-demo
cd go-demo
```

3. 执行以下命令部署函数。

```
make deploy
```

返回结果如下。

```
docker build -t fc-go-runtime  -f build-image/Dockerfile build-image
Sending build context to Docker daemon  3.072kB
Step 1/5 : FROM golang:1.12.16-stretch
 ---> 7ad03a8aece5
...
Step 5/5 : RUN go get github.com/awesome-fc/golang-runtime
 ---> Using cache
 ---> 262e7f38ac05
Successfully built 262e7f38ac05
Successfully tagged fc-go-runtime:latest
docker run -it -v $(pwd):/tmp fc-go-runtime bash -c "go build -o /tmp/code//bootstrap /tmp/code
/main.go"
chmod +x code/bootstrap
fun deploy -y
using template: template.yml
...
Waiting for service go_demo to be deployed...
    Waiting for function fc_go to be deployed...
        Waiting for packaging function fc_go code...
        The function fc_go has been packaged. A total of 1 file were compressed and the final si
ze was 3.76 MB
    function fc_go deploy success
service go_demo deploy success
```

4. 执行以下命令调用部署的函数。

```
fun invoke -e "Hello World"
```

返回结果如下。

```
...
========= FC invoke Logs begin =========
FC Invoke Start RequestId: 4c1451b2-f29b-4554-87e5-126f3bc11fcf
2020-04-07T02:53:01.981Z: 4c1451b2-f29b-4554-87e5-126f3bc11fcf [INFO]  hello golang!
FC Invoke End RequestId: 4c1451b2-f29b-4554-87e5-126f3bc11fcf

Duration: 1.03 ms, Billed Duration: 100 ms, Memory Size: 512 MB, Max Memory Used: 4.39 MB
========= FC invoke Logs end =========

FC Invoke Result:
Hello World
```

更多信息

如果您需将Go应用框架一键迁移到函数计算，请参见以下文档：

- 迁移Beego到函数计算
- 迁移Gin到函数计算

3.6. C++

通过函数计算的Custom Runtime，您可以在函数计算平台使用C++来编写函数。本文介绍快速部署和调用C++函数的详细步骤。

前提条件

您已安装8.6.0及以上版本的Node.js。详情请参见官方下载地址。

> 说明 如果您已安装Funcraft可直接跳转至步骤二：部署和调用函数。

步骤一：准备环境

安装Funcraft，最简单的方式就是直接下载可执行的二进制文件。

1. 安装Funcraft到本机。详情请参见安装Funcraft。
2. 执行 `fun --version` 检查安装是否成功。
3. 执行 `fun config` 配置Funcraft。然后按照提示依次配置Account ID、Access Key ID、Access Key Secret、Default region name。

```
fun config
Aliyun Account ID 1234xxx
Aliyun Access Key ID xxxx
Aliyun Access Key Secret xxxx
Default region name cn-xxxx
The timeout in seconds for each SDK client invoking 300
The maximum number of retries for each SDK client 5
Allow to anonynously report usage statistics to improve the tool over time? (Y/n)
```

步骤二：部署和调用函数

1. 执行以下命令克隆示例工程到本地。

```
git clone https://github.com/awesome-fc/fc-custom-demo
```

> 说明 如果您没有安装Git，可以直接在浏览器地址栏输入https://github.com/awesome-fc/fc-custom-demo，然后单击下载按钮直接下载。

2. 执行以下命令进入克隆的示例项目中。

```
cd fc-custom-demo
cd cpp-demo
```

3. 执行以下命令将项目部署到函数计算。

```
make deploy
```

返回结果如下。

```
docker build -t fc-cpp-runtime  -f build-image/Dockerfile build-image
Sending build context to Docker daemon  2.048kB
Step 1/3 : FROM aliyunfc/runtime-custom:base
 ---> 5bbdcf6377bd
...
Step 3/3 : RUN apt-get install -y cmake
 ---> Using cache
 ---> a244cd26cec2
Successfully built a244cd26cec2
Successfully tagged fc-cpp-runtime:latest
docker run --rm -it -v $(pwd):/tmp fc-cpp-runtime bash -c "cd /tmp && ./build.sh"
-- The C compiler identification is GNU 6.3.0
-- The CXX compiler identification is GNU 6.3.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
...
```

```
Scanning dependencies of target cppruntime
[ 33%] Building CXX object CMakeFiles/cppruntime.dir/src/handler.cpp.o
[ 66%] Building CXX object CMakeFiles/cppruntime.dir/src/logging.cpp.o
[100%] Linking CXX shared library /tmp/bin/libcppruntime.so
...
-- Build files have been written to: /tmp/sample/release
...
[100%] Built target bootstrap
fun deploy -y

Waiting for service cpp_demo to be deployed...
    Waiting for function fc_cpp_event to be deployed...
        Waiting for packaging function fc_cpp_event code...
        The function fc_cpp_event has been packaged. A total of 2 files were compressed and th
e final size was 446.51 KB
    function fc_cpp_event deploy success
    Waiting for function fc_cpp_http to be deployed...
        Waiting for packaging function fc_cpp_http code...
        The function fc_cpp_http has been packaged. A total of 2 files were compressed and the
final size was 446.51 KB
        Waiting for HTTP trigger http_t to be deployed...
        triggerName: http_t
        methods: [ 'GET', 'POST', 'PUT', 'DELETE' ]
        url: https://123456789.cn-hangzhou.fc.aliyuncs.com/2016-08-15/proxy/cpp_demo/fc_cpp_
http/
        Http Trigger will forcefully add a 'Content-Disposition: attachment' field to the response
header, which cannot be overwritten
        and will cause the response to be downloaded as an attachment in the browser. This iss
ue can be avoided by using CustomDomain.

        trigger http_t deploy success
    function fc_cpp_http deploy success
service cpp_demo deploy success

Detect 'DomainName:Auto' of custom domain 'my_domain'
Fun will reuse the temporary domain http://17904911-123456789.test.functioncompute.com, expire
d at 2020-05-06 20:41:51, limited by 1000 per day.

Waiting for custom domain my_domain to be deployed...
custom domain my_domain deploy success
```

4. 执行以下命令调用部署的函数。

```
fun invoke cpp_demo/fc_cpp_event -e "Hello World"
```

返回结果如下。

```
...
========= FC invoke Logs begin =========
/invoke is called.
FC Invoke Start RequestId: bf282a87-0f0b-4953-b159-a31792bad22a
2020-04-27T08:01:08 bf282a87-0f0b-4953-b159-a31792bad22a [INFO] handling invoke
FC Invoke End RequestId: bf282a87-0f0b-4953-b159-a31792bad22a

Duration: 9.11 ms, Billed Duration: 100 ms, Memory Size: 512 MB, Max Memory Used: 4.25MB
========= FC invoke Logs end =========

FC Invoke Result:
Hello World
$ curl http://17904911-123456789.test.functioncompute.com -d "Hello World"
Hello World
```

3.7. Lua

通过函数计算的Custom Runtime，您可以在函数计算平台使用Lua来编写函数。本文介绍快速部署和调用Lua函数的详细步骤。您可以修改Lua示例中的代码，实现自己的需求。

前提条件

您已安装8.6.0及以上版本的Node.js。详情请参见官方下载地址。

> **说明** 如果您已安装Funcraft可直接跳转至步骤二：部署和调用函数。

步骤一：准备环境

安装Funcraft，最简单的方式就是直接下载可执行的二进制文件。

1. 安装Funcraft到本机。详情请参见安装Funcraft。
2. 执行 `fun --version` 检查安装是否成功。
3. 执行 `fun config` 配置Funcraft。然后按照提示依次配置Account ID、Access Key ID、Access Key Secret、Default region name。

```
fun config
Aliyun Account ID 1234xxx
Aliyun Access Key ID xxxx
Aliyun Access Key Secret xxxx
Default region name cn-xxxx
The timeout in seconds for each SDK client invoking 300
The maximum number of retries for each SDK client 5
Allow to anonynously report usage statistics to improve the tool over time? (Y/n)
```

步骤二：部署和调用函数

1. 执行以下命令克隆示例工程到本地。

```
git clone https://github.com/awesome-fc/fc-custom-demo
```

> 说明 如果您没有安装Git，可以直接在浏览器地址栏输入https://github.com/awesome-fc/fc-custom-demo，然后单击下载按钮直接下载。

2. （可选）将*lua-demo*目录中的代码示例进行修改实现您的业务逻辑。
3. 执行以下命令进入克隆的示例项目中。

```
cd fc-custom-demo
cd lua-demo
```

4. 执行以下命令将项目部署至函数计算。

```
fun deploy -y
```

返回结果如下。

```
...
Waiting for service lua-demo to be deployed...
    Waiting for function fc-lua to be deployed...
        Waiting for packaging function fc-lua code...
        The function fc-lua has been packaged. A total of 7 files were compressed and the final s
ize was 10.62 MB
    function fc-lua deploy success
service lua-demo deploy success
```

5. 执行以下命令调用部署的函数。

```
fun invoke -e "Hello World"
```

返回结果如下。

```
...
========= FC invoke Logs begin =========
FC Invoke Start RequestId: dcc2ff81-2318-4a89-abae-c181ede22b79, client: 21.0.3.254, server: , req
uest: "POST /invoke HTTP/1.1", host: "21.0.3.1:9000"
2020/05/10 13:16:21 [notice] 7#7: *2 [lua] main.lua:17: FC Invoke End RequestId: dcc2ff81-2318-4a8
9-abae-c181ede22b79, client: 21.0.3.254, server: , request: "POST /invoke HTTP/1.1", host: "21.0.3.
1:9000"
21.0.3.1 21.0.3.254 0.000 [10/May/2020:13:16:21 +0000] "POST /invoke HTTP/1.1" 200 22 "-" "Go-http-
client/1.1" "-" dcc2ff81-2318-4a89-abae-c181ede22b79

Duration: 3.42 ms, Billed Duration: 100 ms, Memory Size: 512 MB, Max Memory Used: 4.66 MB
========= FC invoke Logs end =========

FC Invoke Result:
Hello World
```

4.Custom Container

4.1. 简介

本文介绍Custom Container Runtime（自定义容器运行环境）的背景介绍、使用限制、工作原理、HTTP Server配置要求、日志格式、冷启动优化以及计费说明。

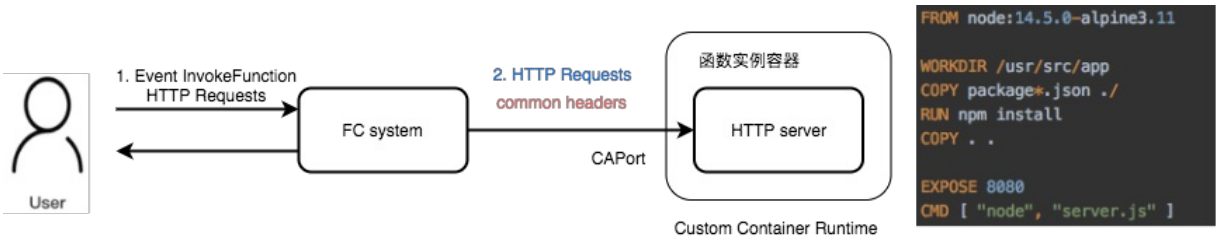
背景介绍

在云原生时代，容器镜像已经逐渐变成了软件部署和开发的标准工具。为了简化开发者体验、提升开发和交付效率，函数计算提供了Custom Container Runtime。开发者将容器镜像作为函数的交付物，通过HTTP协议和函数计算系统交互。使用Custom Container Runtime简化了以下场景：

- 低成本迁移，无需修改代码或是重新编译二进制、共享对象（*.so），保持开发和线上环境一致。
- 解压前镜像大小最大支持1 GB，避免代码和依赖分离，简化分发和部署。
- 容器镜像天然的分层缓存，提高代码上传和拉取效率。
- 标准可复现的第三方库引用、分享、构建、代码上传、存储和版本管理，丰富的开源生态CI/CD交付体验。

工作原理

Custom Container Runtime工作原理与Custom Runtime基本相同。函数计算系统初始化实例前会扮演该函数的服务角色（Service Role），获得临时用户名和密码并拉取镜像。拉取成功后根据指定的启动命令（Command）和参数（Args）启动您定义的HTTP Server。该Server监听您设置的监听端口（CAPort），并与函数计算系统交互，请求处理逻辑即为函数逻辑。函数计算将事件调用和HTTP调用转发至该Server触发函数逻辑执行。



HTTP Server配置要求

- 监听在任何IP（0.0.0.0）的指定端口（端口可以读取环境变量FC_SERVER_PORT，默认为9000）。
- HTTP Server需配置 `connection keep-alive` 。
- 请求超时时间设置为15分钟以上。
- HTTP Server需要在25秒内启动完毕。

函数日志格式

Custom Container Runtime的日志格式与Custom Runtime的日志格式一致，详情请参见日志格式。

冷启动优化最佳实践

相比于代码包，容器镜像依赖的基础环境带来了额外的数据下载和解压的时间，为了更好的冷启动体验，推荐您使用以下最佳实践：

- 容器镜像地址推荐使用与函数计算同地域的VPC镜像地址，例如registry-vpc.cn-

hangzhou.aliyuncs.com/fc-demo/helloworld:v1beta1，以获得最优的镜像拉取延时和稳定性。

- 镜像最小化，基于类似alphe或ubuntu这样的最小镜像或者是其他镜像中精简版本构建自定义镜像。仅保留必要的依赖，删除不必要的文档，数据和其他文件。
- 容器镜像配合预留实例一起使用，详情请参见预留实例简介。
- 在资源允许和线程安全的情况下，搭配单实例多并发一同使用，可避免不必要的冷启动，同时降低成本。

计费说明

Custom Container Runtime的计费项与其他类型Runtime的计费项一致，详情请参见计费说明。其中，镜像资源使用量中的执行时间是从仓库拉取镜像到拉取镜像结束的时长。例如，1024 MB内存的实例拉取镜像耗时10秒，则本次拉取镜像的资源消耗量为10 CU-秒。容器镜像在一段时间一定范围内有缓存，因此并不是每次冷启动一定会产生镜像拉取的费用。

使用限制

- 镜像大小：
  - 若您的函数执行内存小于1 GB，则解压前镜像大小不能超过256 MB。
  - 若您的函数执行内存不小于1 GB，则解压前镜像大小不能超过1024 MB。
- 镜像仓库：目前仅支持阿里云容器镜像服务中默认实例的镜像，未来会支持其他镜像仓库。
- 镜像访问：目前仅支持同账号同地域（Region）下私有镜像仓库读取，未来会支持读取公共镜像。
- 容器内文件读写权限：容器run-as-user UID在10000到10999中随机分配。容器中默认*/tmp*目录可写，其他目录读写权限由镜像文件系统控制。如果运行的UID没有读写某个文件的权限，则需要在*Dockerfile*中修改。
- 容器可写层存储空间限制：512 MB，排除只读镜像层，容器产生的数据最大不能超过该限制。

> **说明** 容器可写层数据不是持久化的，数据会随着容器被销毁而删除。如果需要持久化存储，可以考虑使用函数计算挂载NAS，详情请参见配置NAS，或者使用其他共享存储服务，例如对象存储或表格存储等。

4.2. 事件函数

文本介绍Custom Container事件函数的入参、返回结果及代码示例。

背景介绍

在Custom Container Runtime中，函数计算系统会将Common Headers、调用的请求体（Body）、POST方法以及*/invoke*、*/initialize*等路径转发给容器中的HTTP Server。您可以选择实现类似官方支持的Runtime（例如Golang Runtime）的context、event这样的函数签名。您也可以直接使用入参请求头（Headers）和请求体 （Body）来编写函数的业务逻辑。更多信息，请参见Custom Runtime事件函数调用。

函数入参

- event：POST请求体（Body）。
- context：
  - 通过x-fc-access-key-id、x-fc-access-key-secret和x-fc-security-token请求头获取服务角色（Service Role）中的临时访问凭证访问云服务。
  - 通过x-fc-request-id获取当前请求ID。

- 所有请求头信息请参见Common Headers。

函数返回结构

函数结果通过HTTP响应结构体返回。

代码示例

在以下Node.js Express示例中，POST方法和*/initialize*路径会在函数实例初始化时被函数计算调用，POST方法和*/invoke*路径为函数计算被调用时的Handler，通过 `req.headers` 以及 `req.body` 获取context、event并将函数返回结果通过HTTP Response结构体输出。

```
'use strict';

const express = require('express');

// Constants
const PORT = 9000;
const HOST = '0.0.0.0';

// event function invocation handler
app.post('/invoke', (req, res) => {
  requestID = req.headers['x-fc-request-id'];
  event = req.body
  res.send('Hello FunctionCompute: invoke ' + requestID + ', event: ' + event);
});

// event function initializer
app.post('/initialize', (req, res) => {
  requestID = req.headers['x-fc-request-id'];
  res.send('Hello FunctionCompute: initialize ' + requestID);
});

app.listen(PORT, HOST);
console.log(`Running on http://${HOST}:${PORT}`);
```

4.3. HTTP函数

本文介绍Custom Container HTTP函数的请求、响应规范以及代码示例。

背景信息

函数计算系统会将用户的请求，包括请求路径（Path）、请求头（Headers）、请求体（Body）以及Common Headers转发给HTTP server。与事件函数不同，HTTP函数不要求实现 `/invoke` 以及 `/initialize` 等路径，您可以平滑迁移一个已有的HTTP Web应用。更详细的介绍请参见Custom Runtime函数调用。

函数入参

- event：POST请求体（Body）。
- context：
  - 通过x-fc-access-key-id、x-fc-access-key-secret和x-fc-security-token请求头获取服务角色（Service Role）中的临时访问凭证访问云服务。
  - 通过x-fc-request-id获取当前请求ID。
  - 所有请求头信息请参见Common Headers。

函数返回结构

函数结果通过HTTP响应结构体返回。

代码示例

以下Node.js Express示例中，GET和POST方法分别路由至不同的Handler，您也可以根据需要做任意的Path和Handler映射。

```
'use strict';

const express = require('express');

// Constants
const PORT = 9000;
const HOST = '0.0.0.0';

// HTTP function get
const app = express();
app.get('/*', (req, res) => {
  res.send('Hello FunctionCompute, http GET');
});

const app = express();
app.post('/*', (req, res) => {
  res.send('Hello FunctionCompute, http POST');
});

app.listen(PORT, HOST);
console.log(`Running on http://${HOST}:${PORT}`);
```

4.4. 创建函数

本文介绍如何在函数计算控制台上或使用Funcraft工具创建Custom Container Runtime函数。

前提条件

1. 创建默认实例版容器命名空间
2. 创建镜像仓库

使用控制台创建

1. 执行以下命令将您的函数镜像推送至默认实例镜像仓库。

   本例中函数计算地域为cn-shenzhen，镜像仓库名称为nodejs-express。

   ```
   cd /tmp
   git clone https://github.com/awesome-fc/custom-container-docs.git && cd custom-container-docs
   /nodejsexpress

   # 指定ACR镜像地址
   export IMAGE_NAME="your ACR image name"  # e.g. registry.cn-shenzhen.aliyuncs.com/fc-demo/n
   odejs-express:v0.2
   docker build -t $IMAGE_NAME .
   docker push $IMAGE_NAME
   ```

2. 创建服务并为服务设置权限。
   i. 在函数计算控制台创建服务，具体操作步骤请参见创建服务。
   ii. 为目标服务绑定权限策略AliyunContainerRegistryReadOnlyAccess或者AliyunContainerRegistryFullAccess，详细步骤请参见配置服务权限。

   > **说明** 函数计算需要使用上述策略中的权限去获取容器镜像服务中默认实例的临时账号，然后利用该临时账号推送位于您的私有镜像仓库中的镜像。

3. 创建函数。本文以创建事件函数为例，创建HTTP函数的参数设置类似。
   i. 登录函数计算控制台。
   ii. 在顶部菜单栏，选择地域。
   iii. 在左侧导航栏，单击服务/函数。
   iv. 选择新建函数 > 事件函数 > 下一步。
   v. 在配置函数区域，填写函数相关信息，然后单击完成。

参数	操作
所在服务	在下拉列表中选择步骤2中创建的服务。
函数名称	在文本框中填写自定义的函数名称。
运行环境	在下拉列表中选择custom-container。
容器镜像	在文本框中填写容器镜像的地址。 建议填写VPC网络地址registry-vpc.Endpoint，例如registry-vpc.cn-hangzhou.aliyuncs.com/fc-demo/helloworld:v1beta1。
Command	在文本框中填写启动命令，例如 `["/code/myserver"]` 。 该参数为可选参数，如果不填写，则默认使用镜像中的Entrypoint/CMD。
Args	在文本框中填写参数，例如 `["-arg1", "value1"]` 。 该参数为可选参数，如果不填写，则默认使用镜像中的CMD。

参数	操作
函数执行内存	在下拉列表中选择函数执行需要的内存大小，内存的设置不能小于521 MB。
超时时间	在文本框中输入请求超时时间。
单实例并发度	在文本框中输入单实例并发度。 更多信息请参见单实例多并发简介。
监听端口	在文本框中填写监听端口。 该参数为可选参数，如果不填写，则默认端口为9000。您可以直接在此处修改监听端口， 无需去修改镜像。

创建完成后，您可以在目标服务下的函数列表中看到刚创建的函数。

使用Funcraft工具创建

使用Funcraft工具可以一键构建、推送容器镜像并部署函数。

1. 执行以下命令克隆custom-container-docs示例。

```
git clone https://github.com/awesome-fc/custom-container-docs.git
cd custom-container-docs/nodejsexpress
```

2. 编辑*template.yml*，将参数Image的值替换成您的ACR镜像地址。
3. 执行以下命令构建镜像，部署函数。

```
# Build the Docker image.
fun build --use-docker

# Deploy the function, push the image via the internet registry host (the function config uses the VPC registry for faster image pulling).
fun deploy --push-registry acr-internet
```

创建完成后，您可以登录函数计算控制台，在目标服务下的函数列表中看到刚创建的函数。