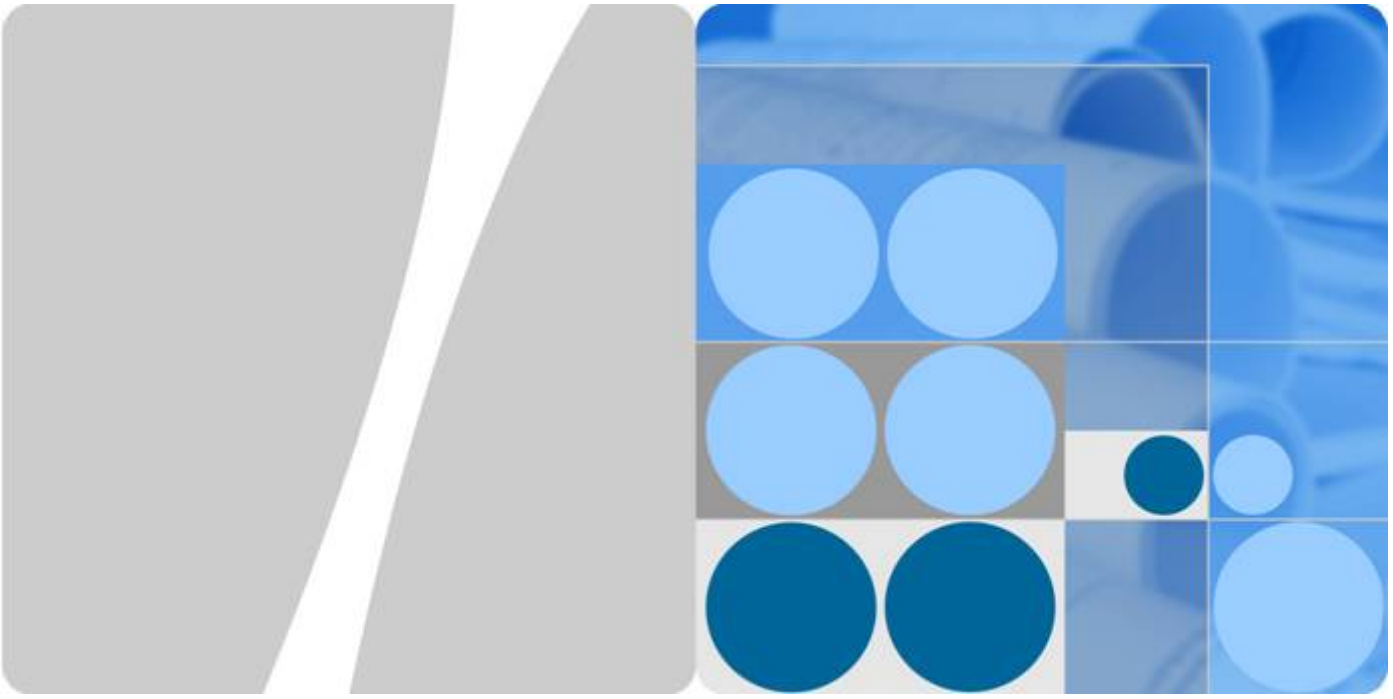


函数工作流

开发指南

发布日期　2023-09-30

目 录

1 概述

1.1 函数开发简介

函数支持的运行时语言

FunctionGraph函数Runtime支持多种运行时语言：Python 、Node.js、Java、Go、C#、PHP及自定义运行时，说明如表1-1所示。

说明

建议使用相关语言的最新版本。

表 1-1 运行时说明

运行时语言	支持版本	SDK下载
Node.js	6.10、8.10、10.16、12.13、16.17、18.15	-
Python	2.7、3.6、3.9、3.10	-
Java	8、11	**Java SDK下载** **说明** Java SDK集成了云服务OBS SDK。
Go	1.x	
C#	.NET Core 2.1、.NET Core 3.1	**CsharpSDK**
PHP	7.3	-
定制运行时	-	-

Node.js Runtime 集成的三方件

表 1-2 Node.js Runtime 集成的三方件

名称	功能	版本号
q	异步方法封装	1.5.1
co	异步流程控制	4.6.0
lodash	常用工具方法库	4.17.10
esdk-obs-nodejs	OBS SDK	2.1.5
express	极简web开发框架	4.16.4
fgs-express	在FunctionGraph和API Gateway之上使用现有的Node.js应用程序框架运行无服务器应用程序和REST API 。提供的示例允许您使用Express框架轻松构建无服务器Web应用程序/服务和RESTful API 。	1.0.1
request	简化http调用，支持HTTPS并默认遵循重定向	2.88.0

Python Runtime 集成的非标准库

表 1-3 Python Runtime 集成的非标准库

模块	功能	版本号
dateutil	日期/时间处理	2.6.0
requests	http库	2.7.0
httplib2	httpclient	0.10.3
numpy	数学计算	1.13.1
redis	redis客户端	2.10.5
obsclient	OBS客户端	-
smnsdk	访问SMN服务	1.0.1

函数样例工程包下载

本手册使用样例工程包下载地址如表1-4所示，可以下载到本地，创建函数时上传使用。

表 1-4 样例工程包下载

函数	工程包下载	软件包校验文件
Node.js函数	fss_examples_nodejs.zip	-
Python函数	fss_examples_python2.7.zip	-
Java函数	fss_example_java8.jar	-
Go函数	fss_examples_go1.8.zip	-
C#函数	fss_example_csharp2.0、fss_example_csharp2.1	-
PHP函数	fss_examples_php7.3.zip	-

1.2 函数支持的事件源

本节列出了FunctionGraph函数支持的云服务，可以将这些服务配置为FunctionGraph函数的事件源。在预配置事件源映射后，这些事件源检测事件时将自动调用FunctionGraph函数。

API Gateway

可以通过HTTPS调用FunctionGraph函数，使用API Gateway自定义REST API和终端节点来实现。可以将各个API操作（如GET和PUT）映射到特定的FunctionGraph函数，当向该API终端节点发送HTTPS请求时（**APIG示例事件**），API Gateway会调用相应的FunctionGraph函数。HTTPS调用触发函数的使用过程请参考**使用APIG触发器**。

对象存储服务 OBS

可以编写FunctionGraph函数来处理OBS存储桶事件，例如对象创建事件或对象删除事件。当用户将一张照片上传到存储桶时，OBS存储桶调用FunctionGraph函数，实现读取图像和创建照片缩略图。OBS对象操作触发函数的过程请参考**使用OBS触发器**。

表 1-5 OBS 支持事件类型

事件	说明
ObjectCreated	表示所有创建对象的操作，包含Put、Post、Copy对象以及合并段。
Put	使用Put方法上传对象。

事件	说明
Post	使用Post方法上传对象。
Copy	使用copy方法复制对象。
CompleteMultipartUpload	表示合并分段任务。
ObjectRemoved	表示删除对象。
Delete	指定对象版本号删除对象。
DeleteMarkerCreated	不指定对象版本号删除对象。

说明

多个事件类型可以作用于同一个目标对象，例如：同时选择“事件类型”复选框中的Put、Copy、Delete等方法作用于某目标对象，则用户往该桶中上传、复制、删除符合前后缀规则的目标对象时，均会发送事件通知给用户。ObjectCreated包含了Put、Post、Copy和CompleteMultipartUpload，如果选择了ObjectCreated，则不能再选择Put、Post、Copy或CompleteMultipartUpload。同理如果选择了ObjectRemoved，则不能再选择Delete或DeleteMarkerCreated。

定时触发器 TIMER

可以使用TIMER的计划事件功能定期调用您的代码，可以指定固定频率（分钟、小时、天数）或指定 cron 表达式定期调用函数（**TIMER示例事件**）。定时触发器的使用请参考**使用定时触发器**。

日志触发器 LTS

可以编写FunctionGraph函数来处理云日志服务订阅的日志，当云日志服务采集到订阅的日志后，可以通过将采集到的日志作为参数传递（**LTS示例事件**）来调用FunctionGraph函数，FunctionGraph函数代码可以对其进行自定义处理、分析或将其加载到其他系统。LTS日志触发的使用过程请参考**使用LTS触发器**。

云审计服务触发器 CTS

可以编写FunctionGraph函数，根据CTS云审计服务类型和操作订阅所需要的事件通知，当CTS云审计服务获取已订阅的操作记录后，通过CTS触发器将采集到的操作记录作为参数传递（**CTS示例事件**）来调用FunctionGraph函数。经由函数对日志中的关键信息进行分析和处理，对系统、网络等业务模块进行自动修复，或通过短信、邮件等形式产生告警，通知业务人员进行处理。CTS触发器的使用请参考**使用CTS触发器**。

分布式消息服务 Kafka

使用Kafka触发器，当向Kafka实例的Topic生产消息时，FunctionGraph会消费消息，触发函数以执行额外的工作，关于Kafka触发器的使用请参见**使用Kafka触发器**。

示例事件

- OBS示例事件

```
{
   "Records": [
      {
         "eventVersion": "2.0",
         "eventTime": "2018-01-09T07:50:50.028Z",
         "requestParameters": {
            "sourceIPAddress": "103.218.216.125"
         },
         "s3": {
            "configurationId": "UK1DGFPYUKUZFHNQ00000160CC0B471D101ED30CE24DF4DB",
            "object": {
               "eTag": "9d377b10ce778c4938b3c7e2c63a229a",
               "sequencer": "00000000160D9E681484D6B4C0000000",
               "key": "job.png",
               "size": 777835
            },
            "bucket": {

               "name": "functionstorage-template",
               "ownerIdentity": {
                  "PrincipalId": "0ed1b73473f24134a478962e631651eb"
               }
            }
         },
         "Region": "{region}",
         "eventName": "ObjectCreated:Post",
         "userIdentity": {
            "principalId": "9bf43789b1ff4b679040f35cc4f0dc05"
         }
      }
   ]
}
```

表 1-6 参数说明

参数	类型	示例值	描述
eventVersion	String	2.0	事件协议的版本。
eventTime	String	2018-01-09T07:50:50.028Z	事件产生的时间。使用ISO-8601标准时间格式。
sourceIPAddress	String	103.218.216.125	请求的源IP地址
s3	Map	参考示例	OBS事件内容
object	Map	参考示例	object参数内容
bucket	Map	参考示例	bucket参数内容
ownerIdentity	Map	参考示例	创建Bucket的用户ID
Region	String	ap-southeast-4	Bucket所在的地域
eventName	String	ObjectCreated:Post	配置的触发函数的事件

参数	类型	示例值	描述
userIdentity	Map	参考示例	请求发起者的账号ID

- APIG示例事件

```
{
    "body": "{\"test\":\"body\"}",
    "requestContext": {
        "apiId": "bc1dcffd-aa35-474d-897c-d53425a4c08e",
        "requestId": "11cdcdcf33949dc6d722640a13091c77",
        "stage": "RELEASE"
    },
    "queryStringParameters": {
        "responseType": "html"
    },
    "httpMethod": "GET",
"pathParameters": {
"path":"value"
},
    "headers": {
        "accept-language": "en-US;q=0.3,en;q=0.2",
        "accept-encoding": "gzip, deflate, br",
        "x-forwarded-port": "443",
        "x-forwarded-for": "103.218.216.98",
        "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
        "upgrade-insecure-requests": "1",
        "host": "50eedf92-c9ad-4ac0-827e-d7c11415d4f1.apigw.region.cloud.com",
        "x-forwarded-proto": "https",
        "pragma": "no-cache",
        "cache-control": "no-cache",
        "x-real-ip": "103.218.216.98",
        "user-agent": "Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:57.0) Gecko/20100101 Firefox/57.0"
    },
    "path": "/apig-event-template",
    "isBase64Encoded": true
}
```

说明

- 通过APIG服务调用函数服务时，isBase64Encoded的值默认为true，表示APIG传递给FunctionGraph的请求体body已经进行Base64编码，需要先对body内容Base64解码后再处理。
- 函数必须按以下结构返回字符串。

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

表 1-7 参数说明

参数	类型	示例值	描述
body	String	"{\"test\":\"body\"}"	记录实际请求转换为String字符串后的内容。

参数	类型	示例值	描述
requestContext	Map	参考示例	请求来源的API网关的配置信息、请求标识、认证信息、来源信息。
httpMethod	String	GET	记录实际请求的HTTP方法
queryStringParameters	Map	参考示例	记录在API网关中配置过的Query参数以及实际取值。
pathParameters	Map	参考示例	记录在API网关中配置过的Path参数以及实际取值。
headers	Map	参考示例	记录实际请求的完整Header内容
path	String	/apig-event-template	记录实际请求的完整的Path信息
isBase64Encoded	Boolean	True	默认为true

- TIMER示例事件

```
{
    "version": "v1.0",
    "time": "2018-06-01T08:30:00+08:00",
    "trigger_type": "TIMER",
    "trigger_name": "Timer_001",
    "user_event": "User Event"
}
```

表 1-8 参数说明

参数	类型	示例值	描述
version	String	V1.0	事件协议的版本
time	String	2018-06-01T08:30:00+08:00	事件产生的时间
trigger_type	String	TIMER	触发器的类型
trigger_name	String	Timer_001	触发器的名字
user_event	String	User Event	在创建触发器时配置的附加信息

- LTS示例事件

```
{
    "lts": {
```

```
    "data":
"ICB7CiAgICAibG9ncyI6W3sKICAgICAgICAgIm1lc3NhZ2UiOiIyMDE4LTA4LTA4LzA4OjA4OjA4IFtXUk5dIFt0ZXN0LmdvOjA4XVRoaXMgaXMgYSB0ZXN0IG1lc3NhZ2UuIiwKICAgICAgICAgInRpbWUiOjE1MzAwMDk2NTMwNTksCiAgICAgICAgICJob3N0X25hbWUiOiJlY3MtdGVzdCIsCiAgICAgICAgICJpcCI6IjE5Mi4xNjguMS4xIiwKICAgICAgICAgInBhdGgiOiJ2YXIvbG9nL3Rlc3QubG9nIiwKICAgICAgICAgImxvZ191aWQiOiI2NjNkNjkzMC03OTJkLTExZTgtOGIwOC0yODZlZDQ4OGNlNzAiLAogICAgICAgICAibluZV9ubyI6MQogICAgICAgICAgImH1dLAogICAgICAgICAgIm93bmVyIjogIjYyODBlMTcwYmQ5MzRmNjBhNGQ4NTFjZjVjYTA1MTI5IiwKICAgICAgICAgJsb2dfZ3JvdXBfaWQiOiAiOTdhOWQyODQtNDQ0OC0xMWU4LThmYTQtMjg2ZWQ0ODhjZTcwIiwKICAgICAgICAgJsb2dfdG9waWNfaWQiOiAiMWE5Njc1YTctNzg0ZC0xMWU4LTlmNzAtMjg2ZWQ0ODhjZTcwIgogfQ=="
    }
}
```


表 1-9 Event 中涉及的参数解释

参数	类型	示例值	描述
data	Sting	参考示例	Base64编码后的数据

- CTS示例事件


```
{
    "cts": {
        "time": "2018/06/26 08:54:07 GMT+08:00",
        "user": {
            "name": "userName",
            "id": "5b726c4fbfd84821ba866bafaaf56aax",
            "domain": {
                "name": "domainName",
                "id": "b2b3853af40448fcb9e40dxj89505ba"
            }
        },
        "request": {},
        "response": {},
        "code": 204,
        "service_type": "vpc",
        "resource_type": "VPC",
        "resource_name": "workflow-2be1",
        "resource_id": "urn:fgs:{region}:2d1d891d93054bbaa69b9e866c0971ac:graph:workflow-2be1",
        "trace_name": "deleteGraph",
        "trace_type": "ConsoleAction",
        "record_time": "2018/06/26 08:54:07 GMT+08:00",
        "trace_id": "69be64a7-0233-11e8-82e4-e5d37911193e",
        "trace_status": "normal"
    }
}
```


表 1-10 参数说明

参数	类型	示例值	描述
User	Map	参考示例	本次请求的发起用户信息
Request	Map	参考示例	事件请求内容
Response	Map	参考示例	事件响应内容
Code	Int	204	事件响应码，例如200、400

参数	类型	示例值	描述
service_type	String	vpc	发送方的简写，比如vpc，ecs等等
resource_type	String	VPC	发送方资源类型，比如vm，vpn等等
resource_name	String	workflow-2be1	资源名称，例如ecs服务中某个虚拟机的名称
trace_name	String	deleteGraph	事件名称,比如:startServer, shutDown等
trace_type	String	ConsoleAction	事件发生源头类型,例如ApiCall
record_time	string	2018/06/26 08:54:07 GMT +08:00	cts服务接受到这条trace的时间
trace_id	String	69be64a7-0233-11e8-82e4-e5d37911193e	事件的唯一标识符
trace_status	String	normal	事件的状态

- Kafka示例事件

```
{
    "event_version": "v1.0",
    "event_time": 1576737962,
    "trigger_type": "KAFKA",
    "region": "{region}",
    "instance_id": "81335d56-b9fe-4679-ba95-7030949cc76b",
    "records": [
        {
            "messages": [
                "kafka message1",
                "kafka message2",
                "kafka message3",
                "kafka message4",
                "kafka message5"
            ],
            "topic_id": "topic-test"
        }
    ]
}
```

表 1-11 参数说明

参数	类型	示例值	描述
event_version	String	v1.0	事件协议的版本

参数	类型	示例值	描述
event_time	String	2018-01-09T07:50:50.028Z	事件发生的时间
trigger_type	String	KAFKA	事件类型
region	String		Kafka实例所在的地域
instance_id	String	81335d56-b9fe-4679-ba95-7030949cc76b	创建的Kafka实例的唯一标识符。
messages	String	参考示例	消息内容
topic_id	String	topic-test	消息的唯一标识符

- RabbitMQ示例事件

```
{
    "event_version": "v1.0",
    "event_time": 1576737962,
    "trigger_type": "RABBITMQ",
    "region": "{region}",
    "records": [
        {
            "messages": [
                "rabbitmq message1",
                "rabbitmq message2",
                "rabbitmq message3",
                "rabbitmq message4",
                "rabbitmq message5"
            ],
            "instance_id": "81335d56-b9fe-4679-ba95-7030949cc76b",
            "exchange": "exchange-test"
        }
    ]
}
```

表 1-12 参数说明

参数	类型	示例值	描述
event_version	String	v1.0	事件协议的版本
Region	String		RabbitMQ实例所在的地域
instance_id	String	81335d56-b9fe-4679-ba95-7030949cc76b	创建的RabbitMQ实例的唯一标识符。

1.3 函数工程打包规范

打包规范说明

函数除了支持在线编辑代码，还支持上传ZIP、JAR、引入OBS文件等方式上传代码，函数工程的打包规范说明如表1-13所示。

表 1-13 函数工程打包规范

编程语言	JAR包	ZIP包	OBS文件
Node.js	不支持该方式	• 假如函数工程文件保存在“~/Code/”文件夹下，在打包的时候务必进入Code文件夹下选中所有工程文件进行打包，这样做的目的是：入口函数是程序执行的入口，确保解压后，入口函数所在的文件位于根目录。 • 如果函数工程引入了第三方依赖，可以将第三方依赖打成ZIP包，在函数代码界面设置外部依赖包；也可以将第三方依赖和函数工程文件一起打包。	将工程打成ZIP包，上传到OBS存储桶。

编程语言	JAR包	ZIP包	OBS文件
PHP	不支持该方式	• 假如函数工程文件保存在“~/Code/”文件夹下，在打包的时候务必进入Code文件夹下选中所有工程文件进行打包，这样做的目的是：入口函数是程序执行的入口，确保解压后，入口函数所在的文件位于根目录。 • 如果函数工程引入了第三方依赖，可以将第三方依赖打成ZIP包，在函数代码界面设置外部依赖包；也可以将第三方依赖和函数工程文件一起打包。	将工程打成ZIP包，上传到OBS存储桶。
Python 2.7	不支持该方式	• 假如函数工程文件保存在“~/Code/”文件夹下，在打包的时候务必进入Code文件夹下选中所有工程文件进行打包，这样做的目的是：入口函数是程序执行的入口，确保解压后，入口函数所在的文件位于根目录。 • 如果函数工程引入了第三方依赖，可以将第三方依赖打成ZIP包，在函数代码界面设置外部依赖包；也可以将第三方依赖和函数工程文件一起打包。	将工程打成ZIP包，上传到OBS存储桶。

编程语言	JAR包	ZIP包	OBS文件
Python 3.6	不支持该方式	• 假如函数工程文件保存在“~/Code/”文件夹下，在打包的时候务必进入Code文件夹下选中所有工程文件进行打包，这样做的目的是：入口函数是程序执行的入口，确保解压后，入口函数所在的文件位于根目录。 • 如果函数工程引入了第三方依赖，可以将第三方依赖打成ZIP包，在函数代码界面设置外部依赖包；也可以将第三方依赖和函数工程文件一起打包。	将工程打成ZIP包，上传到OBS存储桶。
Java 8	如果函数没有引用第三方件，可以直接将函数工程编译成Jar包。	如果函数引用第三方件，将函数工程编译成Jar包后，将所有依赖三方件和函数jar包打成ZIP包。	将工程打成ZIP包，上传到OBS存储桶。
Go 1.8	不支持该方式	必须在编译之后打zip包，编译后的动态库文件名称必须与函数执行入口的插件名称保持一致，例如：动态库名称为testplugin.so，则“函数执行入口”命名为testplugin.Handler，其中Handler为入口函数。	将工程打成ZIP包，上传到OBS存储桶。
Go 1.x	不支持该方式	必须在编译之后打zip包，编译后的二进制文件必须与执行函数入口保持一致，例如二进制名称为Handler，则执行入口为Handler。	将工程打成ZIP包，上传到OBS存储桶。

编程语言	JAR包	ZIP包	OBS文件
C#	不支持该方式	必须在编译之后打zip包，必须包含“工程名.deps.json”，“工程名.dll”，“工程名.runtimeconfig.json”，“工程名.pdb”和“HC.Serverless.Function.Common.dll”文件。	将工程打成ZIP包，直接上传到OBS存储桶。
定制运行时	不支持该方式	打zip包，必须包含“bootstrap”可执行引导文件。	将工程打成ZIP包，直接上传到OBS存储桶。

ZIP 工程包示例

- Nods.js工程ZIP包目录示例

```
Example.zip                  示例工程包
|--- lib                     业务文件目录
|--- node_modules            npm三方件目录
|--- index.js                入口js文件（必选）
|--- package.json            npm项目管理文件
```

- PHP工程ZIP包目录示例

```
Example.zip                  示例工程包
|--- ext                     扩展库目录
|--- pear                    PHP扩展与应用仓库
|--- index.php               入口PHP文件
```

- Python工程ZIP包目录示例

```
Example.zip                  示例工程包
|--- com                     业务文件目录
|--- PLI                     第三方依赖PLI目录
|--- index.py                入口py文件（必选）
|--- watermark.py            实现打水印功能的py文件
|--- watermark.png           水印图片
```

- Java工程ZIP包目录示例

```
Example.zip                  示例工程包
|--- obstest.jar             业务功能JAR包
|--- esdk-obs-java-3.20.2.jar      第三方依赖JAR包
|--- jackson-core-2.10.0.jar       第三方依赖JAR包
|--- jackson-databind-2.10.0.jar   第三方依赖JAR包
|--- log4j-api-2.12.0.jar          第三方依赖JAR包
|--- log4j-core-2.12.0.jar         第三方依赖JAR包
|--- okhttp-3.14.2.jar             第三方依赖JAR包
|--- okio-1.17.2.jar               第三方依赖JAR包
```

- Go工程ZIP包目录示例

```
Example.zip                  示例工程包
|--- testplugin.so           业务功能包
```

- C#工程ZIP包目录示例

```
Example.zip                                示例工程包
|--- fssExampleCsharp2.0.deps.json         工程编译产生文件
|--- fssExampleCsharp2.0.dll               工程编译产生文件
|--- fssExampleCsharp2.0.pdb               工程编译产生文件
```

```
|--- fssExampleCsharp2.0.runtimeconfig.json   工程编译产生文件
|--- Handler                                  帮助文件，可直接使用
|--- HC.Serverless.Function.Common.dll        函数工作流提供的dll
```

- 定制运行时

```
Example.zip                                   示例工程包
|--- bootstrap                                可执行引导文件
```

1.4 在函数中引入动态链接库

- 函数运行环境中已经默认将代码根目录和根目录下的lib目录加入到LD_LIBRARY_PATH中，只需要将动态链接库放到此处即可。
- 在代码中直接修改LD_LIBRARY_PATH环境变量。
- 如果依赖的.so文件放在其他目录，可以在配置页面设置LD_LIBRARY_PATH环境变量指明对应的目录。
- 如果使用了挂载文件系统中的库，可以在配置页面设置LD_LIBRARY_PATH环境变量指明挂载文件系统中对应的目录。

2 函数初始化入口 Initializer

概述

Initializer是函数的初始化逻辑入口，不同于请求处理逻辑入口的handler，在有函数初始化的需求场景中，设置了Initializer后，FunctionGraph首先调用initializer完成函数的初始化，之后再调用handler处理请求；如果没有函数初始化的需求则可以跳过initializer，直接调用handler处理请求。

适用场景

用户函数执行调度包括以下几个阶段：

1. FunctionGraph预先为函数分配执行函数的容器资源。
2. 下载函数代码。
3. 通过runtime运行时加载代码。
4. 用户函数内部进行初始化逻辑。
5. 函数处理请求并将结果返回。

其中1、2和3是系统层面的冷启动开销，通过对调度以及各个环节的优化，函数服务能做到负载快速增长时稳定的延时。4是函数内部初始化逻辑，属于应用层面的冷启动开销，例如深度学习场景下加载规格较大的模型、数据库场景下连接池构建、函数依赖库加载等等。

为了减小应用层冷启动对延时的影响，FunctionGraph推出了initializer接口，系统能识别用户函数的初始化逻辑，从而在调度上做相应的优化。

引入 initializer 接口的价值

- 分离初始化逻辑和请求处理逻辑，程序逻辑更清晰，让用户更易写出结构良好，性能更优的代码。
- 用户函数代码更新时，系统能够保证用户函数的平滑升级，规避应用层初始化冷启动带来的性能损耗。新的函数实例启动后能够自动执行用户的初始化逻辑，在初始化完成后再处理请求。
- 在应用负载上升，需要增加更多函数实例时，系统能够识别函数应用层初始化的开销，更准确的计算资源伸缩的时机和所需的资源量，让请求延时更加平稳。
- 即使在用户有持续的请求且不更新函数的情况下，系统仍然有可能将已有容器回收或更新，这时没有平台方的冷启动，但是会有业务方冷启动，Initializer可以最大限度减少这种情况。

initializer 接口规范

各个runtime的initializer接口有以下共性：

- 无自定义参数

  Initializer不支持用户自定义参数，只能获取FunctionGraph提供的context参数中的变量进行相关逻辑处理。

- 无返回值

  开发者无法从invoke的响应中获取initializer预期的返回值。

- 超时时间

  开发者可单独设置initializer的超时时间，与handler的超时相互独立，但最长不超过 300 秒。

- 执行时间

  运行函数逻辑的进程称之为函数实例，运行在容器内。FunctionGraph会根据用户负载伸缩函数实例。每当有新函数实例创建时，系统会首先调用initializer。系统保证一定initializer执行成功后才会执行handler逻辑。

- 最多成功执行一次

  FunctionGraph保证每个函数实例启动后只会成功执行一次initializer 。如果执行失败，那么该函数实例执行失败，选取下一个实例重新执行，最多重试3次。一旦执行成功，在该实例的生命周期内不会再执行initializer，收到Invoke请求之后只执行请求处理函数。

- initializer入口命名

  除Java外，其他runtime的initializer入口命名规范与原有的执行函数命名保持一致，格式为 [文件名].[initializer名]，其中initializer名可自定义。Java需要定义一个类并实现函数计算预定义的初始化接口。

- 计量计费

  Initializer的执行时间也会被计量，用户需要为此付费，计费方式同执行函数。

3 Node.js

3.1 开发事件函数

函数定义

说明

建议使用Node.js 12.13版本。

- Node.js 6.10函数定义

  Node.js6.10 函数的接口定义如下所示。

  export.handler = function(event, context, callback)

  - 入口函数名（handler）：入口函数名称， 需和函数执行入口处用户自定义的入口函数名称一致。
  - 执行事件（event）：函数执行界面由用户输入的执行事件参数， 格式为JSON对象。
  - 上下文环境（context）：Runtime提供的函数执行上下文，其接口定义在**SDK接口**说明。
  - 回调函数（callback）：callback方法完整声明为callback(err, message)，用户通过此方法可以返回err和message至前台结果显示页面。具体的err或message内容需要用户自己定义，如字符串。
  - 函数执行入口： index.handler

    函数执行入口格式为“[文件名].[函数名]”。例如创建函数时设置为index.handler，那么FunctionGraph会去加载index.js中定义的handler函数。

- Node.js 8.10、Node.js 10.16、Node.js 12.13、Node.js14.18函数定义

  Node.js 8.10、Node.js 10.16、Node.js 12.13、Node.js14.18 Runtime除了兼容Node.js 6.10 Runtime函数的接口定义规范，还支持使用async的异步形式作为函数入口。

  exports.handler = async (event, context, callback[可选]) => { return data;}

  通过return进行返回。

Node.js 的 initializer 入口介绍

FunctionGraph目前支持以下Node.js运行环境：

- Node.js6.10 (runtime = Node.js6)
- Node.js8.10 (runtime = Node.js8)
- Nodejs10.16(runtime = Node.js10)
- Nodejs12.13(runtime = Node.js12)
- Node.js16.17(runtime = Node.js16)
- Node.js18.15(runtime = Node.js18)

Initializer入口格式为：

[文件名].[initializer名]

示例：实现initializer接口时指定的Initializer入口为“index.initializer”，那么函数服务会去加载index.js中定义的initializer函数。

在函数服务中使用Node.js编写initializer逻辑，需要定义一个Node.js函数作为initializer入口，一个最简单的initializer示例如下。

```
exports.initializer = function(context, callback) {
    callback(null, '');
    };
```

- 函数名

  exports.initializer需要与实现initializer接口时的Initializer字段相对应。

  示例：创建函数时指定的Initializer入口为index.initializer，那么FunctionGraph会去加载index.js中定义的initializer函数。

- context参数

  context参数中包含一些函数的运行时信息。例如：request id、临时AccessKey、function meta等。

- callback参数

  callback参数用于返回调用函数的结果，其签名是function(err, data)，与Nodejs中惯用的callback一样，它的第一个参数是error，第二个参数data。如果调用时error不为空，则函数将返回HandledInitializationError，由于屏蔽了初始化函数的返回值，所以data中的数据是无效的，可以参考上文的示例设置为空。

SDK 接口

Context类中提供了许多上下文方法供用户使用，其声明和功能如表3-1所示。

表 3-1 Context 类上下文方法说明

方法名	方法说明
getRequestID()	获取请求ID。
getRemainingTimeInMilliSeconds ()	获取函数剩余运行时间。
getAccessKey()	获取用户委托的AccessKey（有效期24小时），使用该方法需要给函数配置委托。 **说明** 当前函数工作流已停止维护Runtime SDK 中getAccessKey接口，您将无法使用getAccessKey获取临时AK。

方法名	方法说明
getSecretKey()	获取用户委托的SecretKey（有效期24小时），使用该方法需要给函数配置委托。 **说明** 当前函数工作流已停止维护Runtime SDK 中getSecretKey接口，您将无法使用getSecretKey获取临时SK。
getSecurityAccessKey()	获取用户委托的SecurityAccessKey（有效期24小时），使用该方法需要给函数配置委托。
getSecuritySecretKey()	获取用户委托的SecuritySecretKey（有效期24小时），使用该方法需要给函数配置委托。
getSecurityToken()	获取用户委托的SecurityToken（有效期24小时），使用该方法需要给函数配置委托。
getUserData(string key)	通过key获取用户通过环境变量传入的值。
getFunctionName()	获取函数名称。
getRunningTimeInSeconds ()	获取函数超时时间。
getVersion()	获取函数的版本。
getMemorySize()	分配的内存。
getCPUNumber()	获取函数占用的CPU资源。
getPackage()	获取函数组。
getToken()	获取用户委托的token（有效期24小时），使用该方法需要给函数配置委托。
getLogger()	获取context提供的logger方法，返回一个日志输出类，通过使用其info方法按“时间-请求ID-输出内容”的格式输出日志。 如调用info方法输出日志： logg = context.getLogger() logg.info("hello")
getAlias	获取函数的别名

警告

getToken()、getAccessKey()和getSecretKey()方法返回的内容包含敏感信息，请谨慎使用，避免造成用户敏感信息的泄露。

执行结果

执行结果由3部分组成：函数返回、执行摘要和日志。

表 3-2 执行结果说明

参数项	执行成功	执行失败

函数返回	返回函数中定义的返回信息。	返回包含错误信息和错误类型的JSON文件。格式如下：
		{
		"errorMessage": "",
		"errorType":"",
		}
		errorMessage：Runtime返回的错误信息
		errorType：错误类型

执行摘要	显示请求ID、配置内存、执行时长、实际使用内存和收费时长。	显示请求ID、配置内存、执行时长、实际使用内存和收费时长。
日志	打印函数日志，最多显示4KB的日志。	打印报错信息，最多显示4KB的日志。

3.2 制作依赖包

制作函数依赖包推荐在EulerOS环境中进行。使用其他系统打包可能会因为底层依赖库的原因，运行出问题，比如找不到动态链接库。

说明

- 如果安装的依赖模块需要添加依赖库，请将依赖库归档到zip依赖包文件中，例如，添加.dll、.so、.a等依赖库。

为 Nodejs 函数制作依赖包

需要先保证环境中已经安装了对应版本的Nodejs。

为Nodejs 8.10安装MySQL依赖包，可以执行如下命令。

```
npm install mysql --save
```

可以看到当前目录下会生成一个node_modules文件夹。

- Linux系统

  Linux系统下可以使用以下命令生成zip包。

  ```
  zip -rq mysql-node8.10.zip node_modules
  ```

  即可生成最终需要的依赖包。

- windows系统

  用压缩软件将node_modules目录压缩成zip文件即可。

如果需要安装多个依赖包，也可以先新建一个package.json文件，例如在package.json中填入如下内容后，执行如下命令。

```
{
  "name": "test",
  "version": "1.0.0",
```

```
    "dependencies": {
        "redis": "~2.8.0",
        "mysql": "~2.17.1"
    }
}
npm install --save
```

说明

不要使用**CNPM**命令制作nodejs依赖包。

然后将node_modules打包成zip即可生成一个既包含MySQL也包含redis的依赖包。

Nodejs其他版本制作依赖包过程与上述相同。

4 Python

4.1 开发事件函数

函数定义

说明

建议使用Python 3.6版本。

对于Python，FunctionGraph运行时支持Python 2.7版本、Python 3.6、Python3.9版本。

函数有明确的接口定义，如下所示。

def handler (event, context)

- 入口函数名（handler）：入口函数名称，需和函数执行入口处用户自定义的入口函数名称一致。
- 执行事件（event）： 函数执行界面由用户输入的执行事件参数， 格式为JSON对象。
- 上下文环境（Context）：Runtime提供的函数执行上下文，其接口定义在**SDK接口**说明。

Python 的 initializer 入口介绍

FunctionGraph目前支持以下Python运行环境。

- Python 2.7 (runtime = python2.7)
- Python 3.6 (runtime = python3)
- Python 3.9（runtime = python3）

Initializer入口格式为：

[文件名].[initializer名]

示例：实现initializer接口时指定的Initializer入口为main.my_initializer，那么FunctionGraph会去加载main.py中定义的my_initializer函数。

在FunctionGraph中使用Python编写initializer，需要定义一个Python函数作为initializer入口，一个最简单的initializer（以Python 2.7版本为例）示例如下。

```
def my_initializer(context):
    print 'hello world!'
```

- 函数名

  my_initializer需要与实现initializer接口时的Initializer字段相对应，实现initializer接口时指定的Initializer入口为main.my_initializer ，那么函数服务会去加载main.py中定义的my_initializer函数。

- context参数

  context参数中包含一些函数的运行时信息，例如：request id、临时AccessKey、function meta等。

SDK 接口

Context类中提供了许多上下文方法供用户使用，其声明和功能如表4-1所示。

表 4-1 Context 类上下文方法说明

方法名	方法说明
getRequestID()	获取请求ID。
getRemainingTimeInMilliSeconds ()	获取函数剩余运行时间。
getAccessKey()	获取用户委托的AccessKey（有效期24小时），使用该方法需要给函数配置委托。 **说明** 当前函数工作流已停止维护Runtime SDK 中getAccessKey接口，您将无法使用getAccessKey获取临时AK。
getSecretKey()	获取用户委托的SecretKey（有效期24小时），使用该方法需要给函数配置委托。 **说明** 当前函数工作流已停止维护Runtime SDK 中getSecretKey接口，您将无法使用getSecretKey获取临时SK。
getSecurityAccessKey()	获取用户委托的SecurityAccessKey（有效期24小时），使用该方法需要给函数配置委托。
getSecuritySecretKey()	获取用户委托的SecuritySecretKey（有效期24小时），使用该方法需要给函数配置委托。
getSecurityToken()	获取用户委托的SecurityToken（有效期24小时），使用该方法需要给函数配置委托。
getUserData(string key)	通过key获取用户通过环境变量传入的值。
getFunctionName()	获取函数名称。
getRunningTimeInSeconds ()	获取函数超时时间。

方法名	方法说明
getVersion()	获取函数的版本。
getMemorySize()	分配的内存。
getCPUNumber()	获取函数占用的CPU资源。
getPackage()	获取函数组。
getToken()	获取用户委托的token（有效期24小时），使用该方法需要给函数配置委托。
getLogger()	获取context提供的logger方法，返回一个日志输出类，通过使用其info方法按“时间-请求ID-输出内容”的格式输出日志。 如调用info方法输出日志： log = context.getLogger() log.info("test")
getAlias	获取函数的别名

警告

getToken()、getAccessKey()和getSecretKey()方法返回的内容包含敏感信息，请谨慎使用，避免造成用户敏感信息的泄露。

执行结果

执行结果由3部分组成：函数返回、执行摘要和日志。

表 4-2 执行结果说明

参数项	执行成功	执行失败

函数返回	返回函数中定义的返回信息。	返回包含错误信息、错误类型和堆栈异常报错信息的JSON文件。格式如下：
		{
		"errorMessage": "",
		"errorType": "",
		"stackTrace": []
		}
		errorMessage：Runtime返回的错误信息
		errorType：错误类型
		stackTrace：Runtime返回的堆栈异常报错信息

执行摘要	显示请求ID、配置内存、执行时长、实际使用内存和收费时长。	显示请求ID、配置内存、执行时长、实际使用内存和收费时长。

参数项	执行成功	执行失败
日志	打印函数日志，最多显示4KB的日志。	打印报错信息，最多显示4KB的日志。

4.2 制作依赖包

制作函数依赖包推荐在EulerOS环境中进行。使用其他系统打包可能会因为底层依赖库的原因，运行出问题，比如找不到动态链接库。

说明

- 如果安装的依赖模块需要添加依赖库，请将依赖库归档到zip依赖包文件中，例如，添加.dll、.so、.a等依赖库。

为 Python 函数制作依赖包

打包环境中的Python版本要和对应函数的运行时版本相同，如Python2.7建议使用2.7.12及以上版本，Python3.6建议使用3.6.3以上版本。

为Python 2.7安装PyMySQL依赖包，并指定此依赖包的安装路径为本地的/tmp/pymysql下，可以执行如下命令。

```
pip install PyMySQL --root /tmp/pymysql
```

执行成功后，执行以下命令。

```
cd /tmp/pymysql/
```

进入子目录直到site-packages路径下（一般路径为usr/lib64/python2.7/site-packages/），接下来执行以下命令。

```
zip -rq pymysql.zip *
```

所生成的包即为最终需要的依赖包。

说明

如果需要安装存放在的本地wheel安装包，直接输入：

```
pip install piexif-1.1.0b0-py2.py3-none-any.whl --root /tmp/piexif
//安装包名称以piexif-1.1.0b0-py2.py3-none-any.whl为例，请以实际安装包名称为准
```

5 Java

5.1 开发事件函数

5.1.1 Java 函数开发指南（使用 Eclipse 工具）

函数定义

函数有明确的接口定义，如下：

作用域 返回参数 函数名（函数参数，Context参数）

- 作用域：提供给FunctionGraph调用的用户函数必须定义为public。
- 返回参数：用户定义，FunctionGraph负责转换为字符串，作为HTTP Response返回。对于返回参数对象类型，HTTP Response该类型的JSON字符串。
- 函数名：用户定义函数名称。
- 用户定义参数，当前函数只支持一个用户参数。对于复杂参数，建议定义为对象类型，以JSON字符串提供数据。FunctionGraph调用函数时，解析JSON为对象。
- Context：runtime提供函数执行上下文，其接口定义在**SDK接口**说明。

创建Java函数时，函数入口参数需要提供函数完整的名字空间，参数格式为：包名.类名.函数名。

Java 的 initializer 入口介绍

函数服务目前支持以下Java运行环境。

- Java 8 (runtime = Java8)

  Initializer格式为：

  [包名].[类名].[执行函数名]

  示例：创建函数时指定的initializer为com.Demo.my_initializer，那么FunctionGraph会去加载com包，Demo类中定义的my_initializer函数。

  在函数服务中使用Java实现initializer接口，需要定义一个java函数作为initializer入口，一个最简单的initializer示例如下。

```
public void my_initializer(Context context)
{
RuntimeLogger log = context.getLogger();
log.log(String.format("ak:%s", context.getAccessKey()));
}
```

- 函数名

  my_initializer需要与实现initializer接口时的initializer字段相对应。

  示例：实现initializer接口时指定的Initializer入口为com.Demo.my_initializer，那么FunctionGraph会去加载com包，Demo类中定义的my_initializer函数。

- context参数

  context参数中包含一些函数的运行时信息，例如：request id、临时AccessKey、function meta等。

SDK 接口

FunctionGraph函数JavaSDK提供了Context接口和日志记录接口，SDK下载地址见Java SDK下载（校验文件：fss-java-sdk-sha256）。

- Context接口

  Context接口提供函数获取函数执行上下文，例如，用户委托的AccessKey/SecretKey、当前请求ID、函数执行分配的内存空间、CPU数等。

  Context接口说明如表5-1所示。

  表 5-1 Context 类上下文方法说明

方法名	方法说明
getRequestID( )	获取请求ID。
getRemainingTimeInMilliSeconds ( )	获取函数剩余运行时间。
getAccessKey( )	获取用户委托的AccessKey（有效期24小时），使用该方法需要给函数配置委托。 **说明** 当前函数工作流已停止维护Runtime SDK 中getAccessKey接口，您将无法使用getAccessKey获取临时AK。
getSecretKey( )	获取用户委托的SecretKey（有效期24小时），使用该方法需要给函数配置委托。 **说明** 当前函数工作流已停止维护Runtime SDK 中getSecretKey接口，您将无法使用getSecretKey获取临时SK。
getSecurityAccessKey( )	获取用户委托的SecurityAccessKey（有效期24小时），使用该方法需要给函数配置委托。
getSecuritySecretKey( )	获取用户委托的SecuritySecretKey（有效期24小时），使用该方法需要给函数配置委托。
getSecurityToken( )	获取用户委托的SecurityToken（有效期24小时），使用该方法需要给函数配置委托。

方法名	方法说明
getUserData(string key)	通过key获取用户通过环境变量传入的值。
getFunctionName( )	获取函数名称。
getRunningTimeInSeconds ( )	获取函数超时时间。
getVersion( )	获取函数的版本。
getMemorySize( )	分配的内存。
getCPUNumber( )	获取函数占用的CPU资源。
getPackage( )	获取函数组。
getToken( )	获取用户委托的token（有效期24小时），使用该方法需要给函数配置委托。
getLogger( )	获取context提供的logger方法（默认会输出时间、请求ID等信息）。
getAlias	获取函数的别名

警告

getToken()、getAccessKey()和getSecretKey()方法返回的内容包含敏感信息，请谨慎使用，避免造成用户敏感信息的泄露。

- 日志接口

  Java SDK日志接口日志说明如表5-2所示。

  表 5-2 日志接口说明表

方法名	方法说明
RuntimeLogger()	记录用户输入日志。包含方法如下：log(String string)。

执行结果

执行结果由3部分组成：函数返回、执行摘要和日志。

表 5-3 执行结果说明

参数项	执行成功	执行失败

函数返回	返回函数中定义的返回信息。	返回包含错误信息和堆栈异常报错信息的JSON文件。格式如下：
		{
		"errorMessage": "",
		"stackTrace": []
		}
		errorMessage：Runtime返回的错误信息
		stackTrace：Runtime返回的堆栈异常报错信息

执行摘要	显示请求ID、配置内存、执行时长、实际使用内存和收费时长。	显示请求ID、配置内存、执行时长、实际使用内存和收费时长。
日志	打印函数日志，最多显示4KB的日志。	打印报错信息，最多显示4KB的日志。

5.2 制作依赖包

制作函数依赖包推荐在EulerOS环境中进行。使用其他系统打包可能会因为底层依赖库的原因，运行出问题，比如找不到动态链接库。

说明

- 如果安装的依赖模块需要添加依赖库，请将依赖库归档到zip依赖包文件中，例如，添加.dll、.so、.a等依赖库。

使用Java编译型语言开发函数时，依赖包需要在本地编译。

6 Go

6.1 开发事件函数

函数定义

函数有明确的接口定义，如下所示：

func Handler (payload []byte, ctx context.RuntimeContext)

- 入口函数名（Handler）：入口函数名称。
- 执行事件体（payload）： 函数执行界面由用户输入的执行事件参数， 格式为JSON对象。
- 上下文环境（ctx ）：Runtime提供的函数执行上下文，其接口定义在**SDK接口**说明。

SDK 接口

FunctionGraph函数GoSDK提供了Context接口和日志记录接口。Go SDK下载（Go SDK下载.sha256）。

- Context接口

  Context接口提供函数获取函数执行上下文，例如，用户委托的AccessKey/SecretKey、当前请求ID、函数执行分配的内存空间、CPU数等。

  Context接口说明如**表6-1**所示。

  表 6-1 Context 类上下文方法说明

方法名	方法说明
getRequestID( )	获取请求ID。
getRemainingTimeInMilligetRunningTimeInSecondsSeconds( )	获取函数剩余运行时间。

方法名	方法说明
getAccessKey( )	获取用户委托的AccessKey（有效期24小时），使用该方法需要给函数配置委托。 **说明** 当前函数工作流已停止维护Runtime SDK 中getAccessKey接口，您将无法使用getAccessKey获取临时AK。
getSecretKey( )	获取用户委托的SecretKey（有效期24小时），使用该方法需要给函数配置委托。 **说明** 当前函数工作流已停止维护Runtime SDK 中getSecretKey接口，您将无法使用getSecretKey获取临时SK。
getSecurityAccessKey()	获取用户委托的SecurityAccessKey（有效期24小时），使用该方法需要给函数配置委托。
getSecuritySecretKey()	获取用户委托的SecuritySecretKey（有效期24小时），使用该方法需要给函数配置委托。
getSecurityToken()	获取用户委托的SecurityToken（有效期24小时），使用该方法需要给函数配置委托。
getUserData(string key)	通过key获取用户通过环境变量传入的值。
getFunctionName( )	获取函数名称。
getRunningTimeInSeconds ( )	获取函数超时时间。
getVersion( )	获取函数的版本。
getMemorySize( )	分配的内存。
getCPUNumber( )	获取函数占用的CPU资源。
getPackage( )	获取函数组。
getToken( )	获取用户委托的token（有效期24小时），使用该方法需要给函数配置委托。
getLogger( )	获取context提供的logger方法（默认会输出时间、请求ID等信息）。
getAlias	获取函数的别名

⚠ 警告

GetToken()、GetAccessKey()和GetSecretKey()方法返回的内容包含敏感信息，请谨慎使用，避免造成用户敏感信息的泄露。

- 日志接口Go SDK日志接口日志说明如表6-2所示。

表 6-2 日志接口说明表

方法名	方法说明
RuntimeLogger()	• 记录用户输入日志对象，包含方法如下：Logf(format string, args ...interface{}) • 该方法会将内容输出到标准输出，格式："时间-请求ID-输出内容"，示例如下：2017-10-25T09:10:03.328Z 473d369d-101a-445e-a7a8-315cca788f86 test log output。

执行结果

执行结果由3部分组成：函数返回、执行摘要和日志。

表 6-3 执行结果说明

参数项	执行成功	执行失败

函数返回	返回函数中定义的返回信息。	返回包含错误信息和错误类型的JSON文件。格式如下：
		{
		"errorMessage": "",
		"errorType":"",
		}
		errorMessage：Runtime返回的错误信息
		errorType：错误类型

执行摘要	显示请求ID、配置内存、执行时长、实际使用内存和收费时长。	显示请求ID、配置内存、执行时长、实际使用内存和收费时长。
日志	打印函数日志，最多显示4KB的日志。	打印报错信息，最多显示4KB的日志。

7 C#

7.1 开发事件函数

7.1.1 C#函数开发

函数定义

说明

建议使用.NET Core 3.1版本。

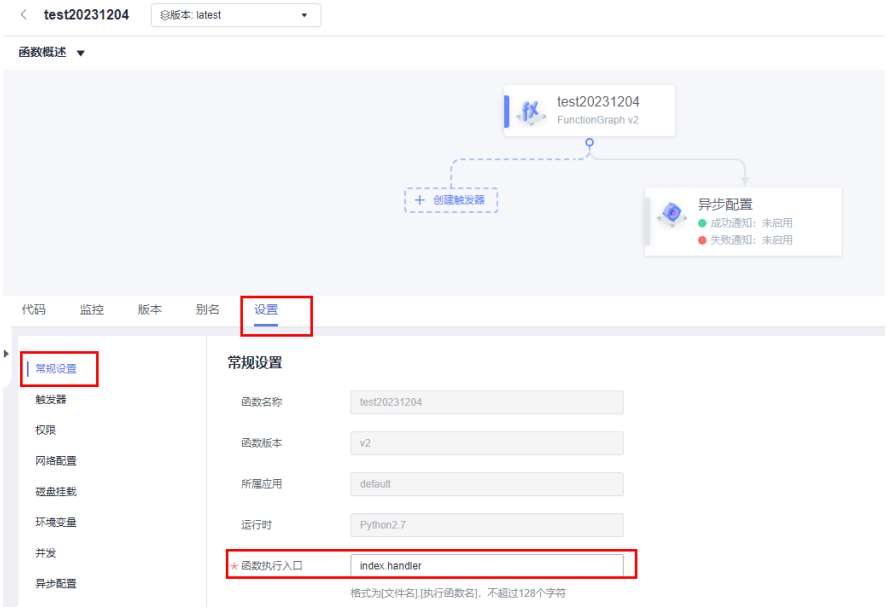
对于C#，FunctionGraph运行时目前支持C#(.NET Core 2.0)、C#(.NET Core 2.1)、C#(.NET Core 3.1)版本。

作用域 返回参数 函数名（函数参数，Context参数）

- 作用域：提供给FunctionGraph调用的用户函数必须定义为public。
- 返回参数：用户定义，FunctionGraph负责转换为字符串，作为HTTP Response返回。
- 函数名：用户自定义函数名称，需要和函数执行入口处用户自定义的入口函数名称一致。

  在函数工作流控制台左侧导航栏选择“函数 > 函数列表”，单击需要设置的“函数名称”进入函数详情页，选择“设置 > 常规设置”，配置“函数执行入口”参数，如图7-1所示。其中参数值为“index.handler”格式，“**index**”和“**handler**”支持自定义命名。

图 7-1 函数执行入口参数



- 执行事件体：函数执行界面由用户输入的执行事件参数。
- 上下文环境（context）：Runtime提供的函数执行上下文，相关属性定义在对象说明中。

  HC.Serverless.Function.Common –部署在FunctionGraph服务中的项目工程需要引入该库，其中包含IFunctionContext对象，详情见context类说明。

  创建csharp函数时，需要定义某个类中的方法作为函数执行入口，该方法可以通过定义IFunctionContext类型的参数来访问当前执行函数的信息。例如：

```
public Stream handlerName(Stream input,IFunctionContext context)
{
    // TODO
}
```

函数 Handler 定义

ASSEMBLY::NAMESPACE.CLASSNAME::METHODNAME

- .ASSEMBLY为应用程序的.NET程序集文件的名称，假设文件夹名称为HelloCsharp。
- NAMESPACE、CLASSNAME即入口执行函数所在的namespace和class名称。
- METHODNAME即入口执行函数名称。例如：

  创建函数时Handler：HelloCsharp::Example.Hello::Handler。

SDK 接口

- Context接口

  Context类中提供了许多属性供用户使用，如表7-1所示。

表 7-1 Context 对象说明

属性名	属性说明
String RequestId	请求ID。
String ProjectId	Project Id
String PackageName	函数所在分组名称
String FunctionName	函数名称
String FunctionVersion	函数版本
Int MemoryLimitInMb	分配的内存。
Int CpuNumber	获取函数占用的CPU资源。
String Accesskey	获取用户委托的AccessKey（有效期24小时），使用该方法需要给函数配置委托。 **说明** 当前函数工作流已停止维护Runtime SDK 中String AccessKey接口，您将无法使用String AccessKey获取临时AK。
String Secretkey	获取用户委托的SecretKey（有效期24小时），使用该方法需要给函数配置委托。 **说明** 当前函数工作流已停止维护Runtime SDK 中String SecretKey接口，您将无法使用String SecretKey获取临时SK。
String SecurityAccessKey	获取用户委托的SecurityAccessKey（有效期24小时），使用该方法需要给函数配置委托。
String SecuritySecretKey	获取用户委托的SecuritySecretKey（有效期24小时），使用该方法需要给函数配置委托。
String SecuritySecretToken	获取用户委托的SecuritySecretToken（有效期24小时），使用该方法需要给函数配置委托。
String Token	获取用户委托的Token（有效期24小时），使用该方法需要给函数配置委托。
Int RemainingTimeInMilliSeconds	函数剩余运行时间
String GetUserData(string key,string defvalue=" ")	通过key获取用户通过环境变量传入的值。

- 日志接口

  FunctionGraph中C# SDK中接口日志说明如所示。

表 7-2 日志接口说明

方法名	方法说明

Log(string message)	利用context创建logger对象：
	var logger = context.Logger;
	logger.Log("hello CSharp runtime test(v1.0.2)");

Logf(string format, args ...interface{})	利用context创建logger对象：
	var logger = context.Logger;
	var version = "v1.0.2"
	logger.Logf("hello CSharp runtime test({0})", version);

执行结果

执行结果由3部分组成：函数返回、执行摘要和日志。

表 7-3 执行结果说明

参数项	执行成功	执行失败

函数返回	返回函数中定义的返回信息。	返回包含错误信息和错误类型的JSON文件。格式如下：
		{
		"errorMessage": "",
		"errorType": ""
		}
		errorMessage：Runtime返回的错误信息
		errorType：错误类型

执行摘要	显示请求ID、配置内存、执行时长、实际使用内存和收费时长。	显示请求ID、配置内存、执行时长、实际使用内存和收费时长。
日志	打印函数日志，最多显示4KB的日志。	打印报错信息，最多显示4KB的日志。

8 PHP

8.1 开发事件函数

函数定义

PHP 7.3函数的接口定义如下所示：

function handler($event, $context)

- 入口函数名（$handler）：入口函数名称， 需和函数执行入口处用户自定义的入口函数名称一致。
- 执行事件（$event）：函数执行界面由用户输入的执行事件参数， 格式为JSON对象。
- 上下文环境（$context）：Runtime提供的函数执行上下文，其接口定义在**SDK接口**说明。
- 函数执行入口： index.handler。
- 函数执行入口格式为“[文件名].[函数名]”。例如创建函数时设置为index.handler，那么FunctionGraph会去加载index.php中定义的handler函数。

PHP 的 initializer 入口介绍

函数服务目前支持以下PHP运行环境。

- Php 7.3 (runtime = Php7.3)

Initializer格式为：

[文件名].[initializer名]

示例：创建函数时指定的initializer为main.my_initializer，那么FunctionGraph会去加载main.php中定义的my_initializer函数。

在函数服务中使用PHP实现initializer接口，需要定义一个PHP函数作为initializer入口，一个最简单的initializer示例如下。

```
<?php
Function my_initializer($context) {
    echo 'hello world' . PHP_EOL;
```

```
  }
?>
```

- 函数名

  my_initializer需要与实现initializer接口时的initializer字段相对应。

  示例：实现initializer接口时指定的Initializer入口为main.my_initializer，那么FunctionGraph会去加载main.php中定义的my_initializer函数。

- context参数

  context参数中包含一些函数的运行时信息，例如：request id、临时AccessKey、function meta等。

SDK 接口

Context类中提供了许多上下文方法供用户使用，其声明和功能如所示。

表 8-1 Context 类上下文方法说明

方法名	方法说明
getRequestID()	获取请求ID。
getRemainingTimeInMilliSeconds ()	获取函数剩余运行时间。
getAccessKey()	获取用户委托的AccessKey（有效期24小时），使用该方法需要给函数配置委托。 **说明** 当前函数工作流已停止维护Runtime SDK 中getAccessKey接口，您将无法使用getAccessKey获取临时AK。
getSecretKey()	获取用户委托的SecretKey（有效期24小时），使用该方法需要给函数配置委托。 **说明** 当前函数工作流已停止维护Runtime SDK 中getSecretKey接口，您将无法使用getSecretKey获取临时SK。
getSecurityAccessKey()	获取用户委托的SecurityAccessKey（有效期24小时），使用该方法需要给函数配置委托。
getSecuritySecretKey()	获取用户委托的SecuritySecretKey（有效期24小时），使用该方法需要给函数配置委托。
getSecurityToken()	获取用户委托的SecurityToken（有效期24小时），使用该方法需要给函数配置委托。
getUserData(string key)	通过key获取用户通过环境变量传入的值。
getFunctionName()	获取函数名称。
getRunningTimeInSeconds ()	获取函数超时时间。
getVersion()	获取函数的版本。
getMemorySize()	分配的内存。

方法名	方法说明
getCPUNumber()	获取函数占用的CPU资源。
getPackage()	获取函数组。
getToken()	获取用户委托的token（有效期24小时），使用该方法需要给函数配置委托。
getLogger()	获取context提供的logger方法，返回一个日志输出类，通过使用其info方法按“时间-请求ID-输出内容”的格式输出日志。 如调用info方法输出日志： logg = context.getLogger()$ $logg->info("hello")
getAlias	获取函数的别名

说明

getToken()、getAccessKey()和getSecretKey()方法返回的内容包含敏感信息，请谨慎使用，避免造成用户敏感信息的泄露。

执行结果

执行结果由3部分组成：函数返回、执行摘要和日志。

表 8-2 执行结果说明

参数项	执行成功	执行失败

函数返回	返回函数中定义的返回信息。	返回包含错误信息、错误类型和堆栈异常报错信息的JSON文件。格式如下：
		{
		"errorMessage": "",
		"errorType": "",
		"stackTrace": {}
		}
		errorMessage：Runtime返回的错误信息
		errorType：错误类型
		stackTrace：Runtime返回的堆栈异常报错信息

执行摘要	显示请求ID、配置内存、执行时长、实际使用内存和收费时长。	显示请求ID、配置内存、执行时长、实际使用内存和收费时长。
日志	打印函数日志，最多显示4KB的日志。	打印报错信息，最多显示4KB的日志。

9 开发工具

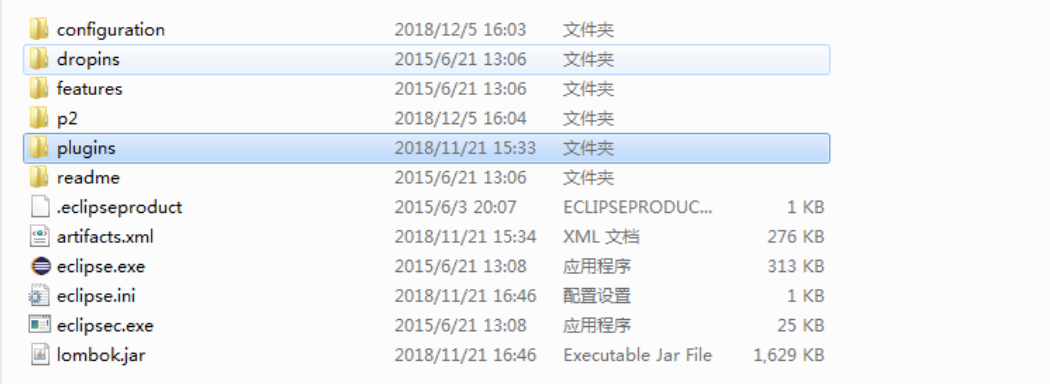
9.1 Eclipse-plugin

当前java没有对应的模板功能，且只支持传包到OBS上，不支持在线编辑，所以需要一个插件，能够支持在java的主流开发工具（Eclipse）上，实现一键创建java模板、java打包、上传到OBS和部署。

步骤1 获取Eclipse 插件（软件包校验文件：Eclipse插件.sha256）。

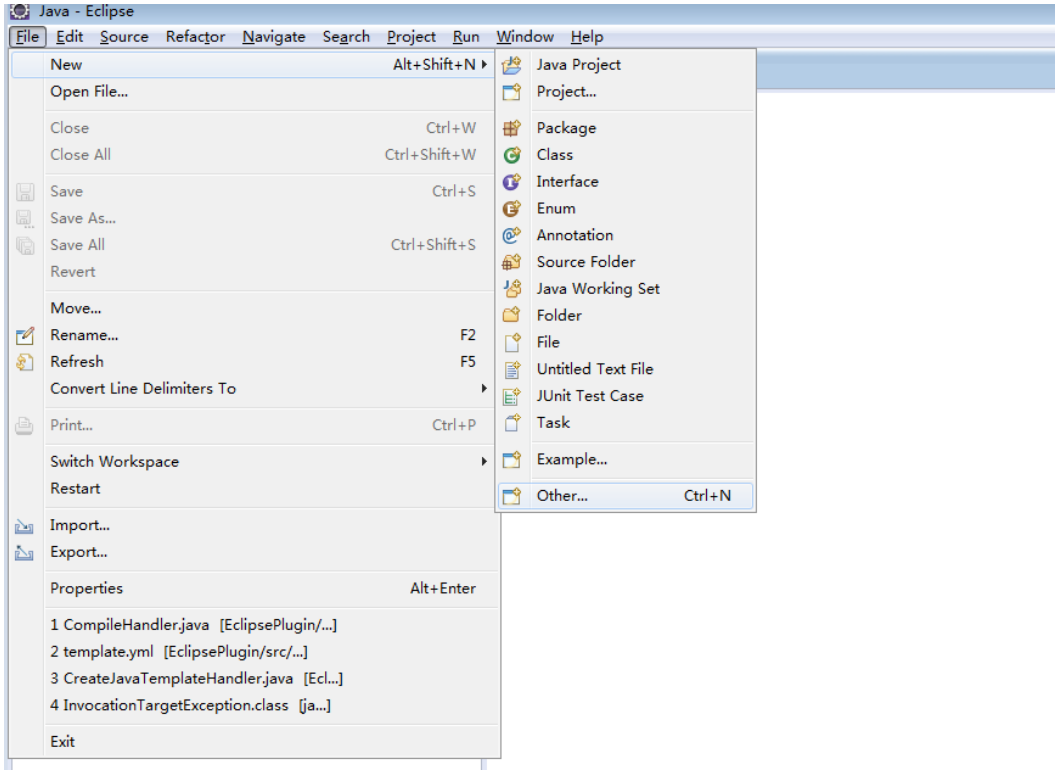
步骤2 将获取的Eclipse插件jar/zip包，放入Eclipse安装目录下的plugins文件夹中，重启Eclipse，即可开始使用Eclipse插件。如图9-1所示。

图 9-1 安装插件



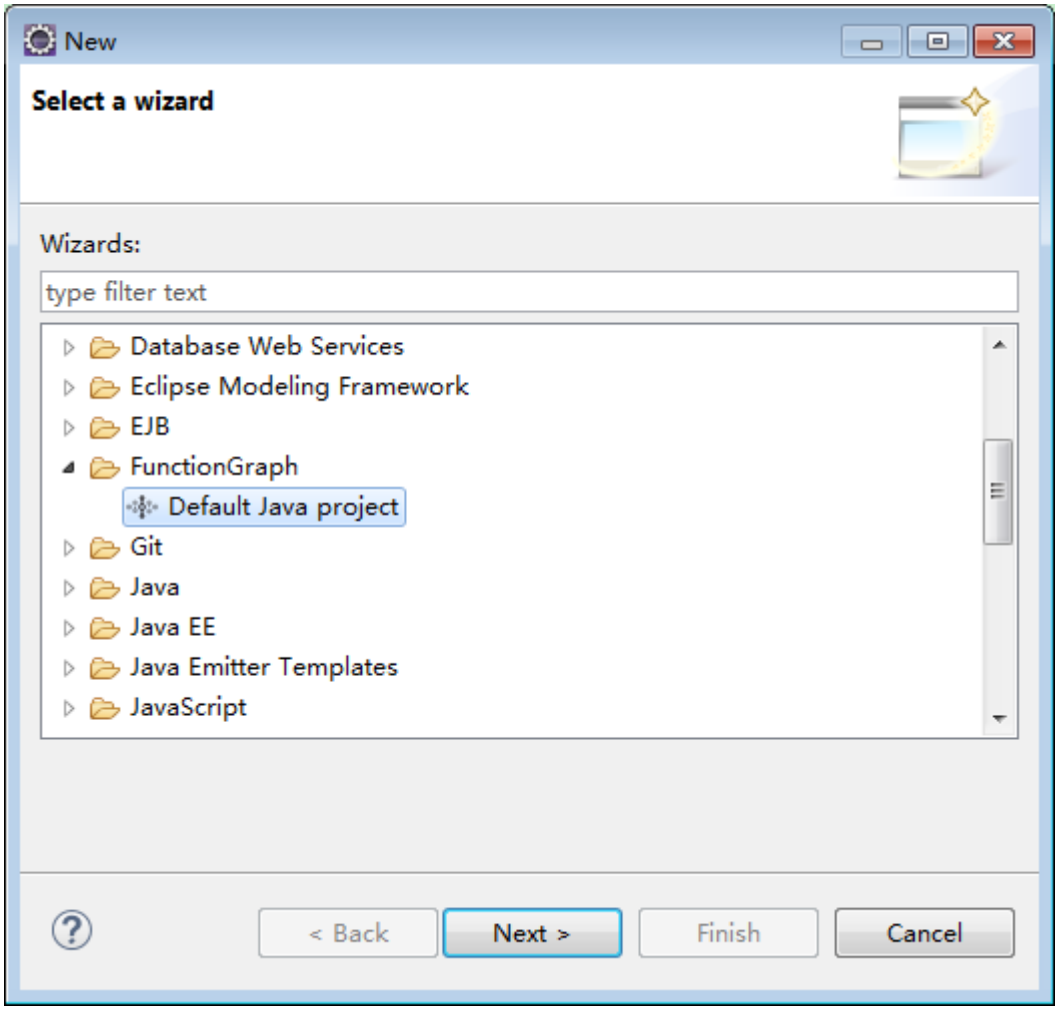
步骤3 打开Eclipse，单击“File”，选择“New > Other”，如图9-2所示。

图 9-2 新建模板



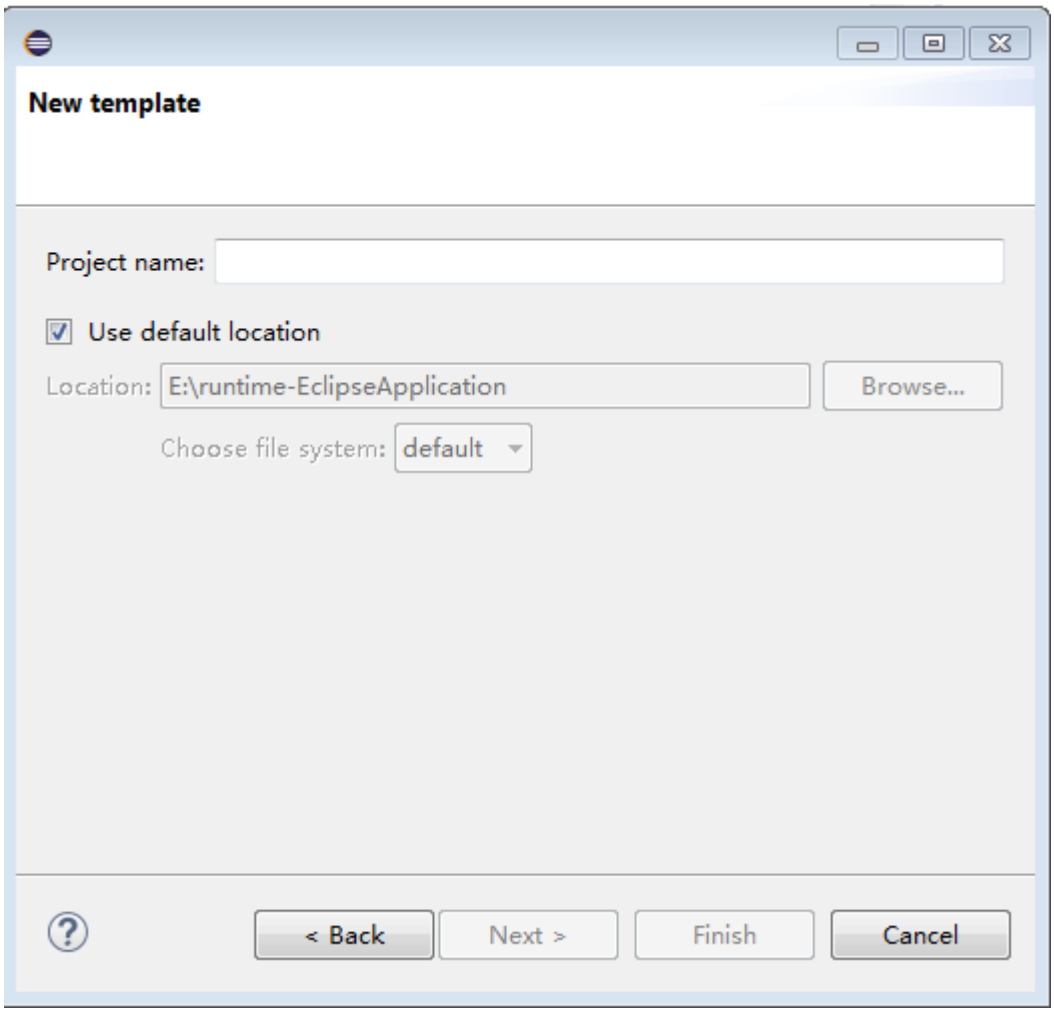
步骤4 选择 “FunctionGraph”文件下的“Default Java project”节点。如图9-3所示。

图 9-3 选择默认 Java 模板



步骤5 输入工程名称，选择工程目录（也可以使用默认目录），单击“Finish”完成模板创建。如图9-4所示。

图 9-4 完成创建



----结束

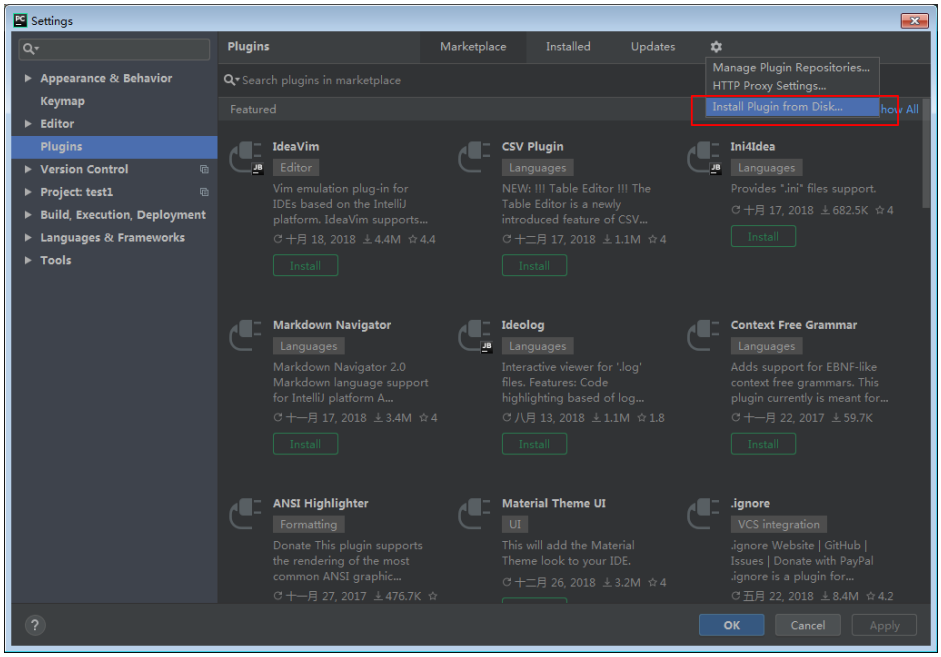
9.2 PyCharm-Plugin

在Python主流开发工具（PyCharm）上实现一键生成python模板工程、打包、部署等功能。

步骤1 获取插件（插件.sha256）。

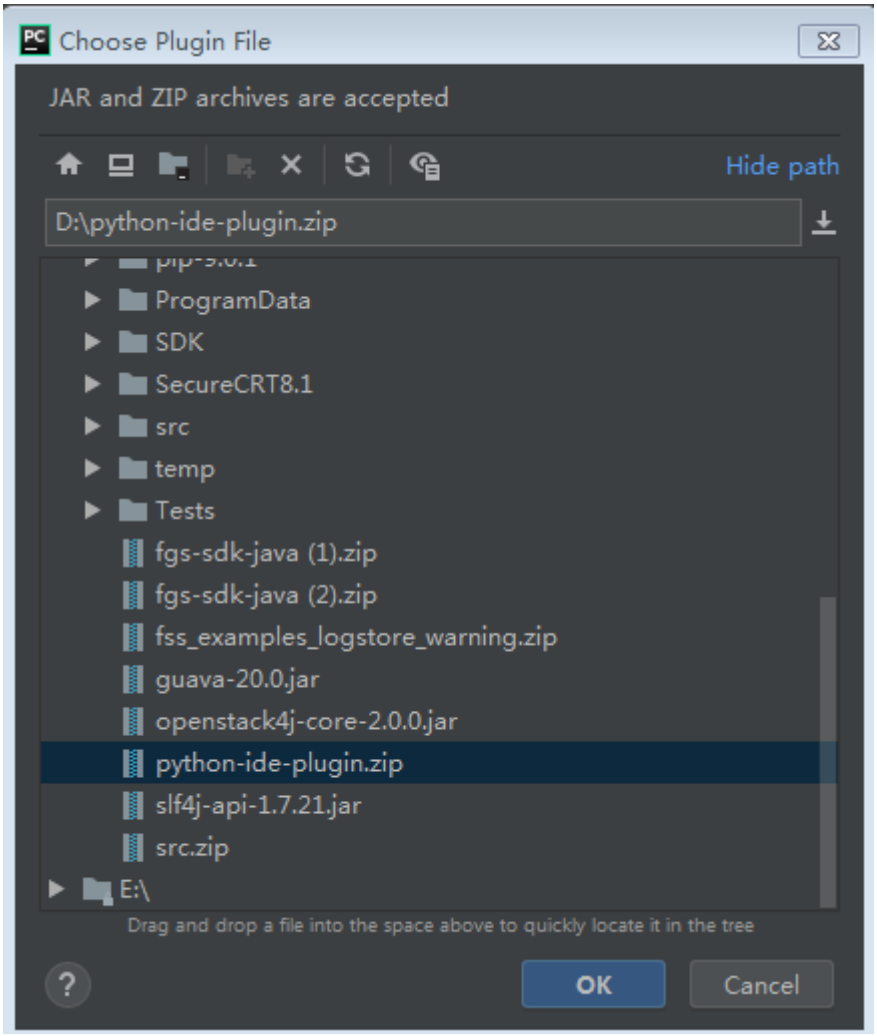
步骤2 打开JetBrains PyCharm，单击“File”菜单，选择“Settings”，在弹出界面的菜单中选择“Plugins”页面，单击右上角设置按钮中的“Install plugin from disk...”，如图9-5所示。

图 9-5 安装 Plugins



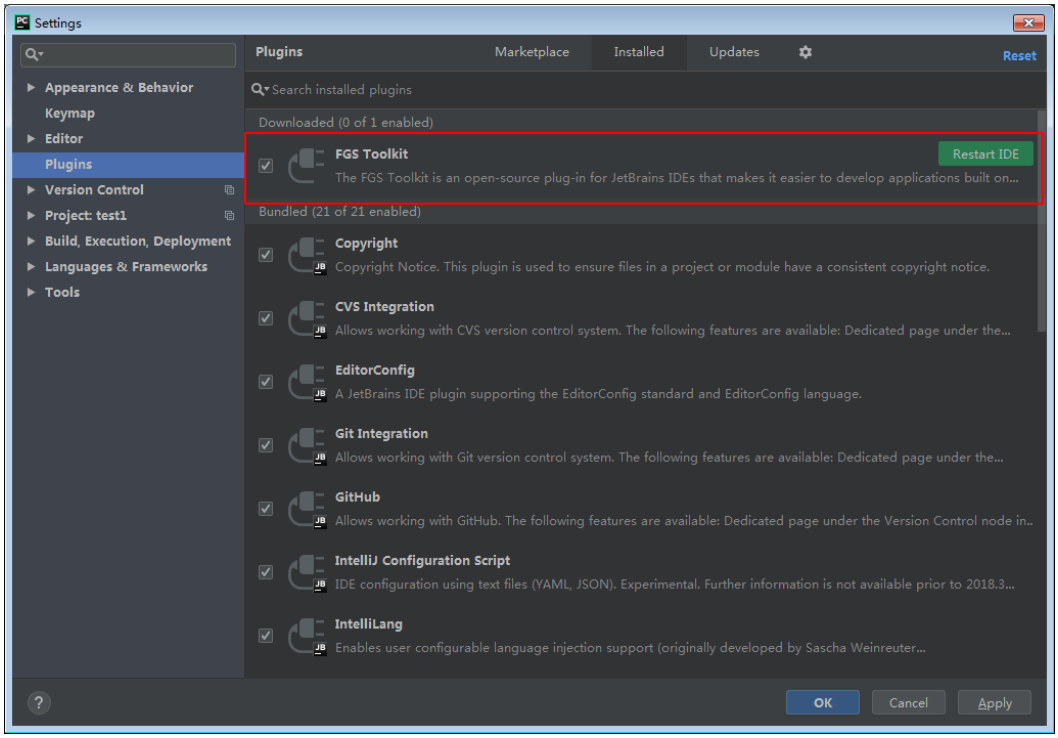
步骤3 在弹出的界面中，选择插件包，单击“OK”，如图9-6所示。

图 9-6 选择插件包



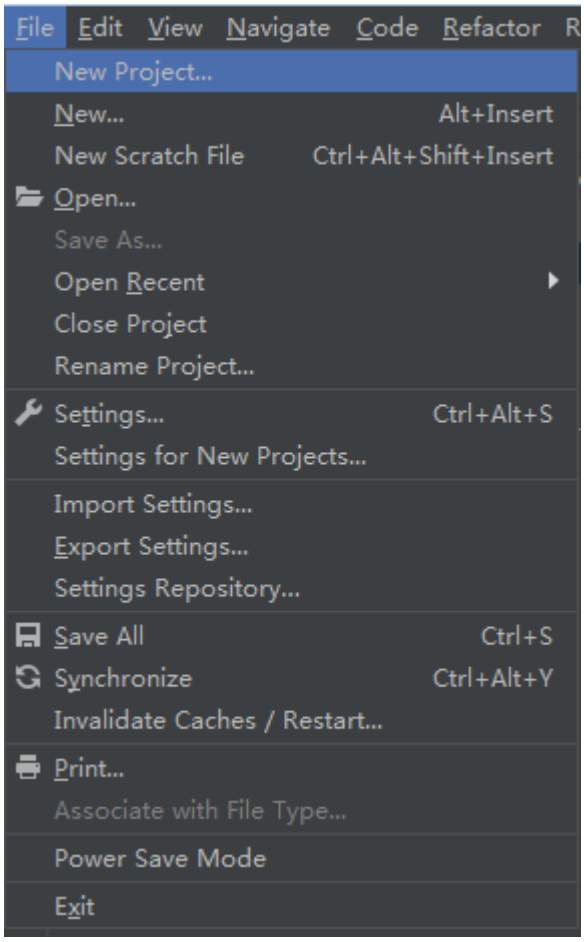
步骤4 在插件列表中，勾选插件名称，单击“Restart IDE”，如图9-7所示。

图 9-7 重启 IDE



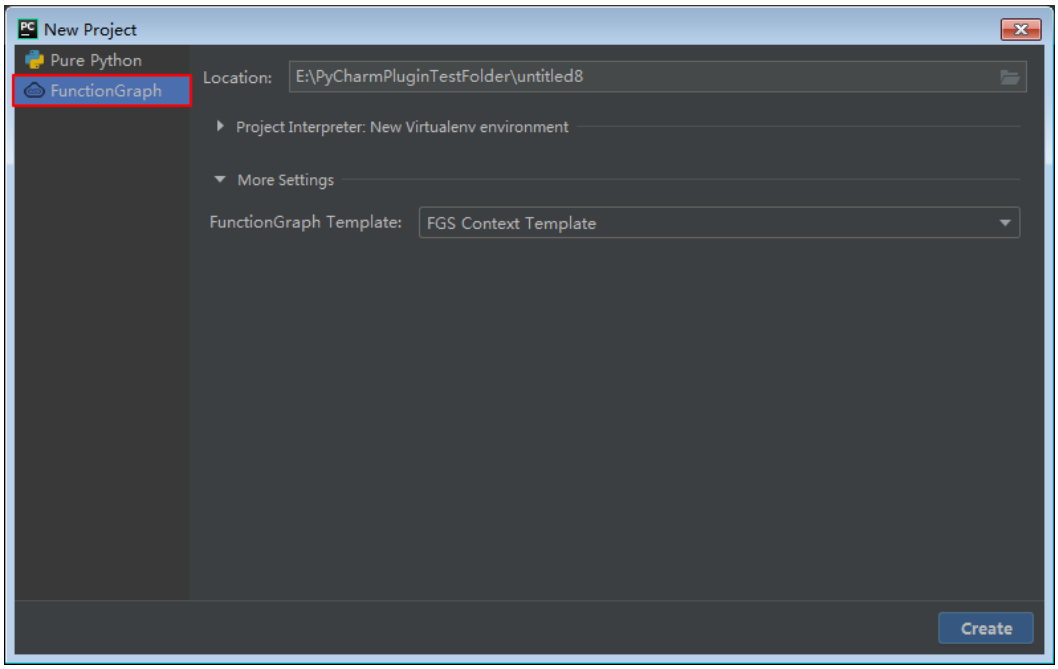
步骤5 单击“File”菜单，选择“New Project”，如图9-8所示。

图 9-8 新建工程



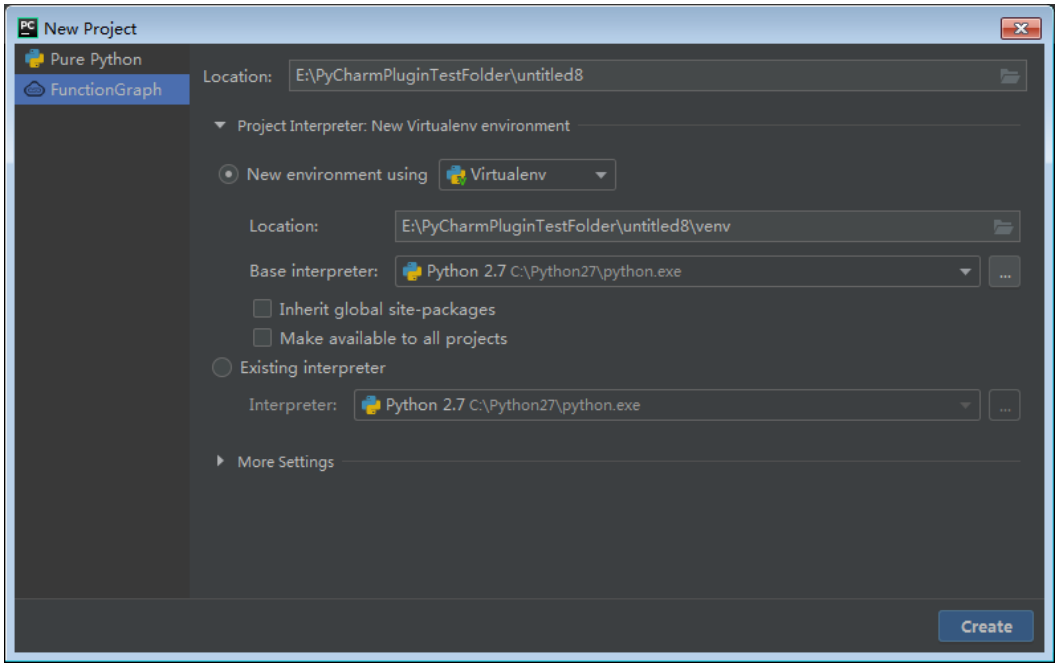
步骤6 在弹出的新建工程页面中，选择“FunctionGraph”，如图9-9所示。

图 9-9 FunctionGraph



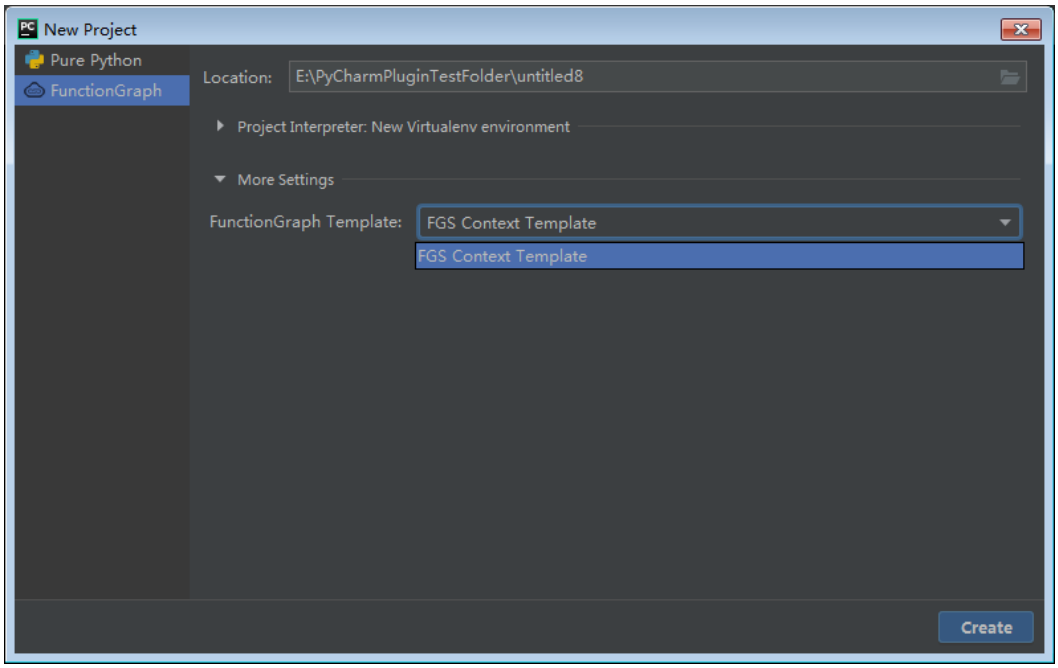
步骤7 在“Location”栏中选择工程的路径，在“Project Interpreter: New Virtualenv environment”中选择使用python的版本。如图9-10所示。

图 9-10 选择版本



步骤8 在“More Settings”中选择要创建的模板，如图9-11所示。

图 9-11 选择模板



说明

目前仅支持python 2.7的Context模板。

步骤9 单击“Create”，完成创建。

----结束