



函数 workflow

# 开发指南

发布日期 2024-05-09

---

# 目录

---

<b>1 概述</b>	<b>1</b>
1.1 函数开发简介	1
1.2 函数支持的事件源	3
1.3 函数工程打包规范	9
1.4 在函数中引入动态链接库	12
<b>2 函数初始化入口 Initializer</b>	<b>14</b>
<b>3 Node.js</b>	<b>16</b>
3.1 开发事件函数	16
3.2 制作依赖包	19
<b>4 Python</b>	<b>21</b>
4.1 开发事件函数	21
4.2 制作依赖包	24
<b>5 Java</b>	<b>25</b>
5.1 开发事件函数	25
5.1.1 Java 函数开发指南（使用 Eclipse 工具）	25
5.2 制作依赖包	28
<b>6 Go</b>	<b>29</b>
6.1 开发事件函数	29
<b>7 开发工具</b>	<b>32</b>
7.1 VS Code-plugin	32
7.2 Eclipse-plugin	36
7.3 PyCharm-Plugin	39
<b>A 修订记录</b>	<b>46</b>

# 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、14.18、16.17、18.15	-
Python	2.7、3.6、3.9、3.10	-
Java	8、11	<a href="#">Java SDK下载</a> 说明 Java SDK集成了云服务OBS SDK。
Go	1.x	
C#	.NET Core 2.1、.NET Core 3.1	<a href="#">CsharpSDK</a>
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	<a href="#">fss_examples_nodejs.zip</a>	-
Python函数	<a href="#">fss_examples_python2.7.zip</a>	-
Java函数	<a href="#">fss_example_java8.jar</a>	-
Go函数	<a href="#">fss_examples_go1.8.zip</a>	-
C#函数	<a href="#">fss_example_csharp2.0</a> 、 <a href="#">fss_example_csharp2.1</a>	-
PHP函数	<a href="#">fss_examples_php7.3.zip</a>	-

## 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。

### 消息通知服务 SMN

可以编写FunctionGraph函数来处理SMN的通知，在将消息发布到SMN主题时，服务可以通过将消息负载作为参数传递（[SMN示例事件](#)）来调用FunctionGraph函数，FunctionGraph函数代码可以处理事件，比如将消息发布到其他SMN主题或将消息发送到其他云服务。SMN消息触发的使用过程请参考[使用SMN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](#)。

## 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](#)。

## 云监控服务 CES

FunctionGraph函数实现了与云监控服务对接，函数上报云监控服务的监控指标，用户可以通过云监控服务来查看函数产生的监控指标和告警信息。查看监控指标请参考[查看监控图表](#)。

## 分布式消息服务 RabbitMQ 版

使用RabbitMQ触发器，FunctionGraph会定期轮询RabbitMQ实例指定Exchange绑定的队列下的新消息，FunctionGraph将轮询得到的消息作为参数传递来调用函数。

## 示例事件

- SMN示例事件

```
{
  "record": [
    {
      "event_version": "1.0",
      "smn": {
        "topic_urn": "urn:smn:{region}:0162c0f220284698b77a3d264376343a:{function_name}",
        "timestamp": "2018-01-09T07:11:40Z",
        "message_attributes": null,
        "message": "this is smn message content",
        "type": "notification",
        "message_id": "a51671f77d4a479cacb09e2cd591a983",
        "subject": "this is smn message subject"
      },
      "event_subscription_urn": "urn:fss:
{region}:0162c0f220284698b77a3d264376343a:function:default:read-smn-message:latest",
      "event_source": "smn"
    }
  ],
  "functionname": "test",
  "requestId": "7c307f6a-cf68-4e65-8be0-4c77405a1b2c",
  "timestamp": "Wed Nov 15 2017 12:00:00 GMT+0800 (CST)"
}
```

表 1-6 参数说明

参数	类型	示例值	描述
event_version	String	1.0	事件协议的版本。
topic_urn	String	参考示例	SMN事件唯一编号
type	String	notification	事件的类型
RequestId	String	7c307f6a-cf68-4e65-8be0-4c77405a1b2c	请求ID。每个请求的ID取值唯一。
message_id	String	a51671f77d4a479cacb09e2cd591a983	消息ID。每条消息的ID取值唯一。
Message	String	this is smn message content	消息内容
event_source	String	smn	事件源
event_subscription_urn	String	参考示例	添加的订阅唯一编号
timestamp	String	Wed Nov 15 2017 12:00:00 GMT+0800 (CST)	事件发生的时间

- OBS示例事件

```

{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventTime": "2018-01-09T07:50:50.028Z",
      "requestParameters": {
        "sourceIPAddress": "103.218.216.125"
      },
      "s3": {
        "configurationId": "UK1DGFYUKUZFHNQ00000160CC0B471D101ED30CE24DF4DB",
        "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 1-7 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	-ae-ad-1	Bucket所在的地域
eventName	String	ObjectCreated:Post	配置的触发函数的事件





```

"instance_id": "81335d56-b9fe-4679-ba95-7030949cc76b",
"records": [
  {
    "messages": [
      "kafka message1",
      "kafka message2",
      "kafka message3",
      "kafka message4",
      "kafka message5"
    ],
    "topic_id": "topic-test"
  }
]

```

表 1-10 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
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	ae-ad-1	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 1-11 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
Region	String	ae-ad-1	RabbitMQ实例所在的地域
instance_id	String	81335d56- b9fe-4679- ba95-7030949cc 76b	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1-12所示。

表 1-12 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	<ul style="list-style-type: none"><li>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。</li><li>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。</li></ul>	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	<ul style="list-style-type: none"> <li>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。</li> <li>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。</li> </ul>	将工程打成ZIP包，上传到OBS存储桶。
Python 2.7	不支持该方式	<ul style="list-style-type: none"> <li>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。</li> <li>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。</li> </ul>	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	<ul style="list-style-type: none"> <li>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。</li> <li>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。</li> </ul>	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，将所有依赖第三方件和函数jar包打成ZIP包。	将工程打成ZIP包，上传到OBS存储桶。
Go 1.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。
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)
- Node.js10.16(runtime = Node.js10)
- Node.js12.13(runtime = Node.js12)
- Node.js14.18(runtime = Node.js14)
- 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。

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。 <b>说明</b> 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。 <b>说明</b> 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： <pre>{   "errorMessage": "",   "errorType": "" }</pre> 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。 <b>说明</b> 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。 <b>说明</b> 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()	<p>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。</p> <p>如调用info方法输出日志：</p> <pre>log = context.getLogger() log.info("test")</pre>
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。	<p>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：</p> <pre>{   "errorMessage": "",   "errorType": "",   "stackTrace": [] }</pre> <p>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</p>
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本地的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-2.0.5.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。 <b>说明</b> 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。 <b>说明</b> 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： <pre>{   "errorMessage": "",   "stackTrace": [] }</pre> 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。。

- 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。 <b>说明</b> 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。 <b>说明</b> 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()	<ul style="list-style-type: none"> <li>• 记录用户输入日志对象，包含方法如下：Logf(format string, args ...interface{})</li> <li>• 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。</li> </ul>

## 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： <pre>{   "errorMessage": "",   "errorType": "" }</pre> 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 开发工具

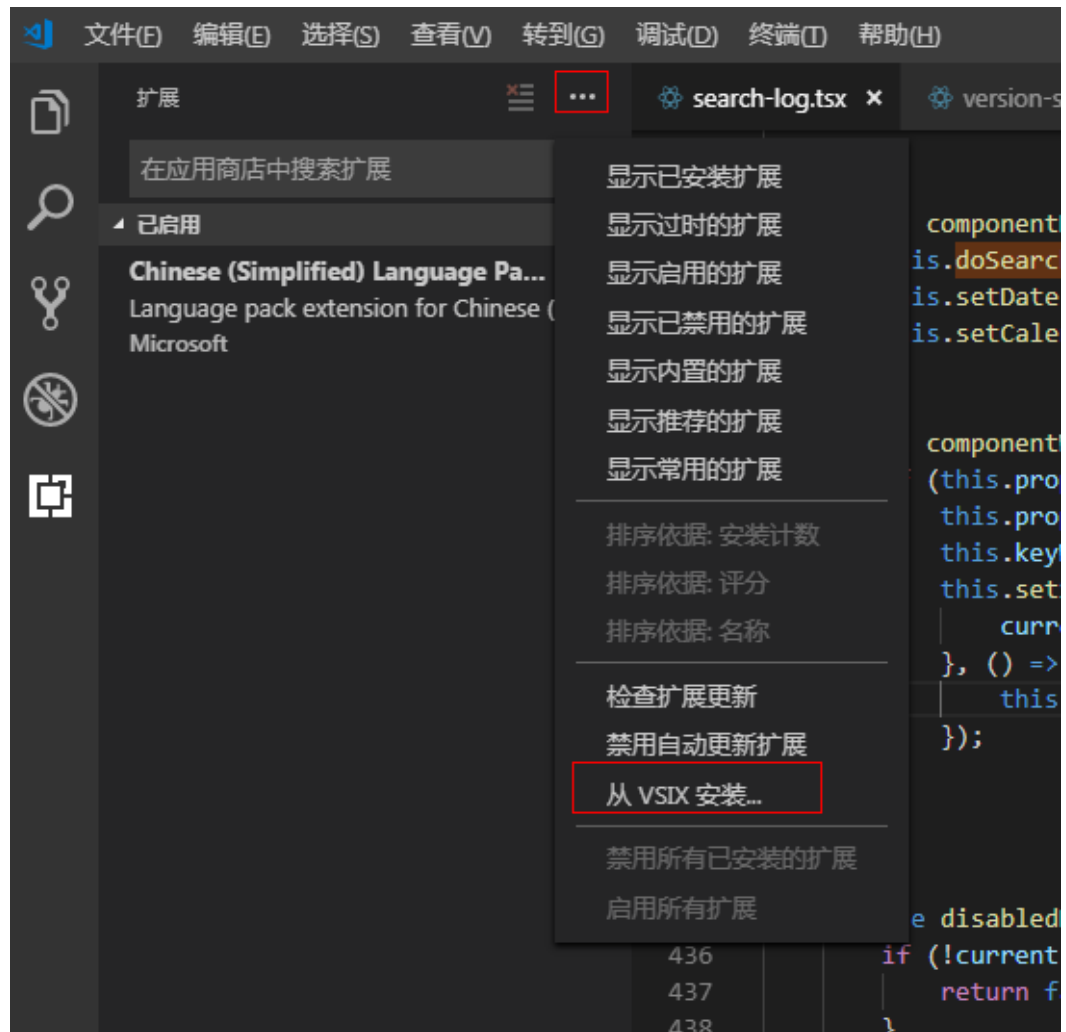
---

## 7.1 VS Code-plugin

**步骤1** 获取[VS Code插件](#)。

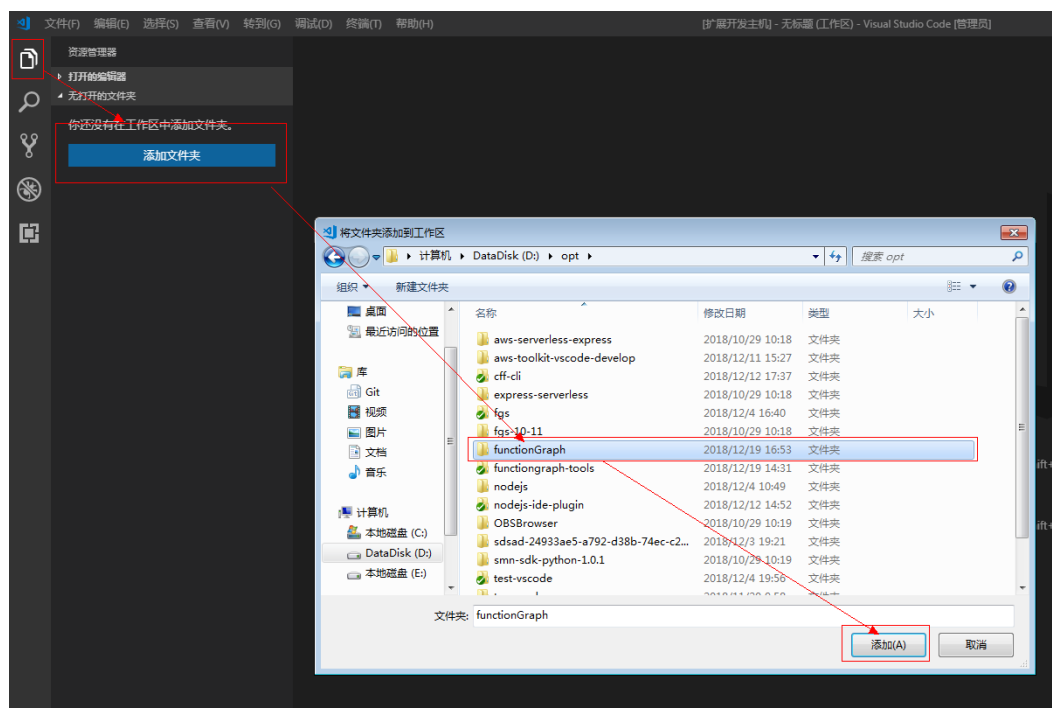
**步骤2** 下载安装VS Code后，手动安装funcitongraph-tools。如[图7-1](#)所示。

图 7-1 安装 funcitongraph-tools



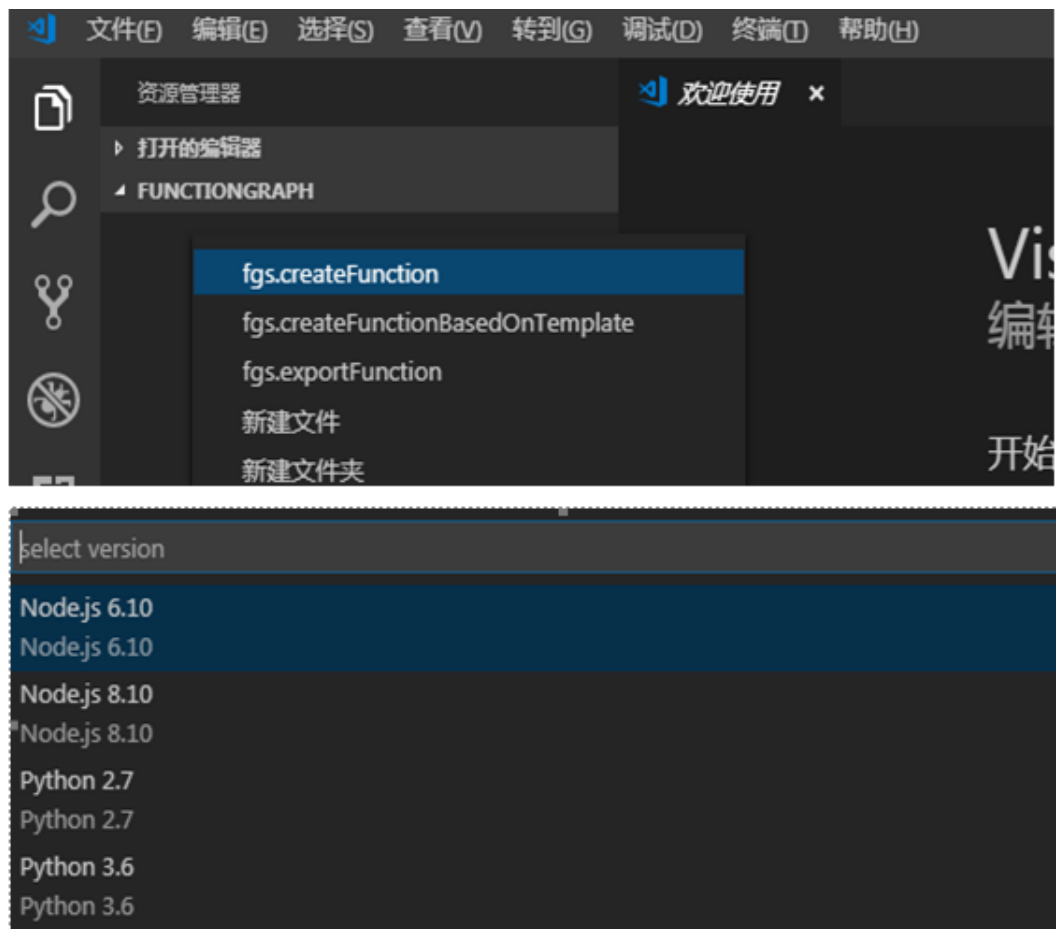
步骤3 添加相应的文件夹，如图7-2所示。

图 7-2 添加文件夹



**步骤4** 单击鼠标右键，选择“fgs.createFunction”，选择相应的“runtime”语言创建函数。如图7-3所示。

图 7-3 创建函数



创建函数时，会自动生成对应的配置文件“functiongraph.yaml”和“index.js”。如图7-4所示。

图 7-4 配置文件



**步骤5** 单击鼠标右键，选择“fgs.createFunctionBasedOnTemplate”，创建函数模板。

**步骤6** 单击鼠标右键，选择“fgs.exportFunction”，可以导出函数，上传至functiongraph服务。

----结束

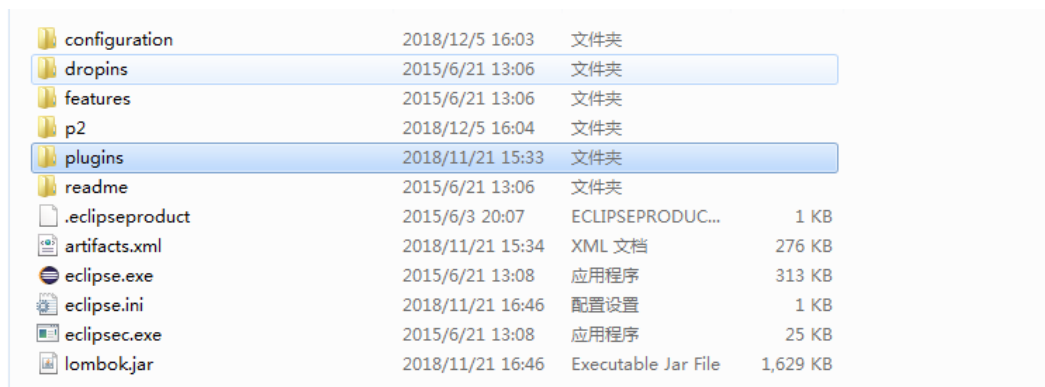
## 7.2 Eclipse-plugin

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，所以需要有一个插件，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](#)）。

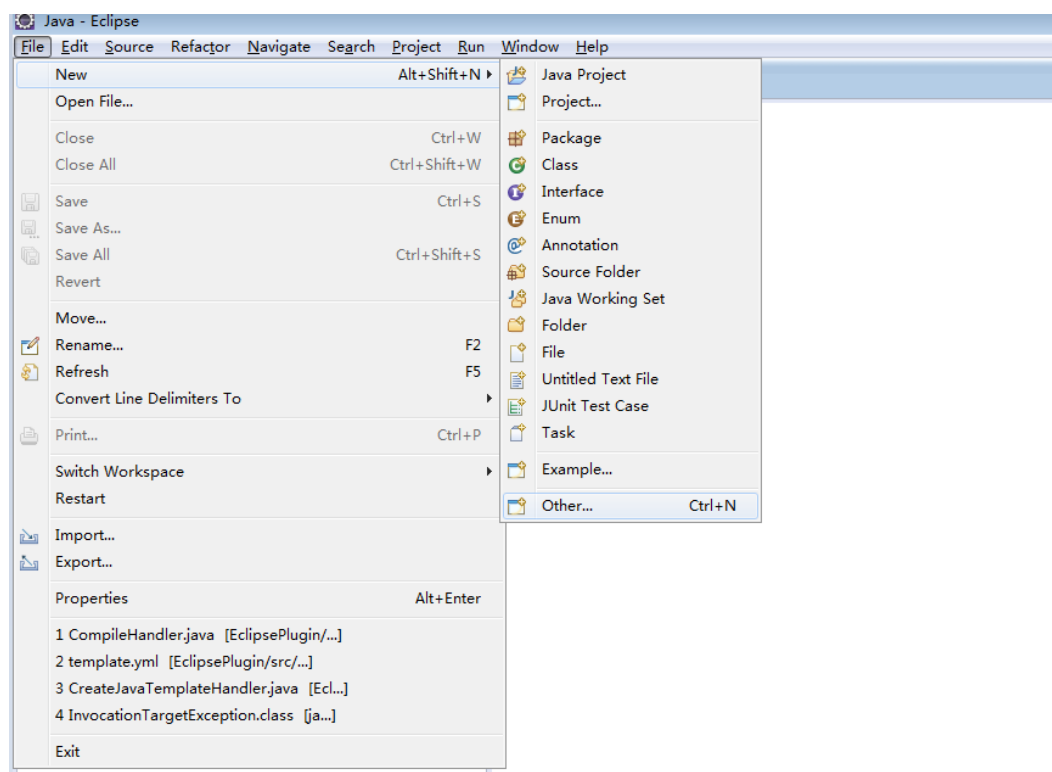
**步骤2** 将获取的Eclipse插件jar/zip包，放入Eclipse安装目录下的plugins文件夹中，重启Eclipse，即可开始使用Eclipse插件。如[图7-5](#)所示。

图 7-5 安装插件



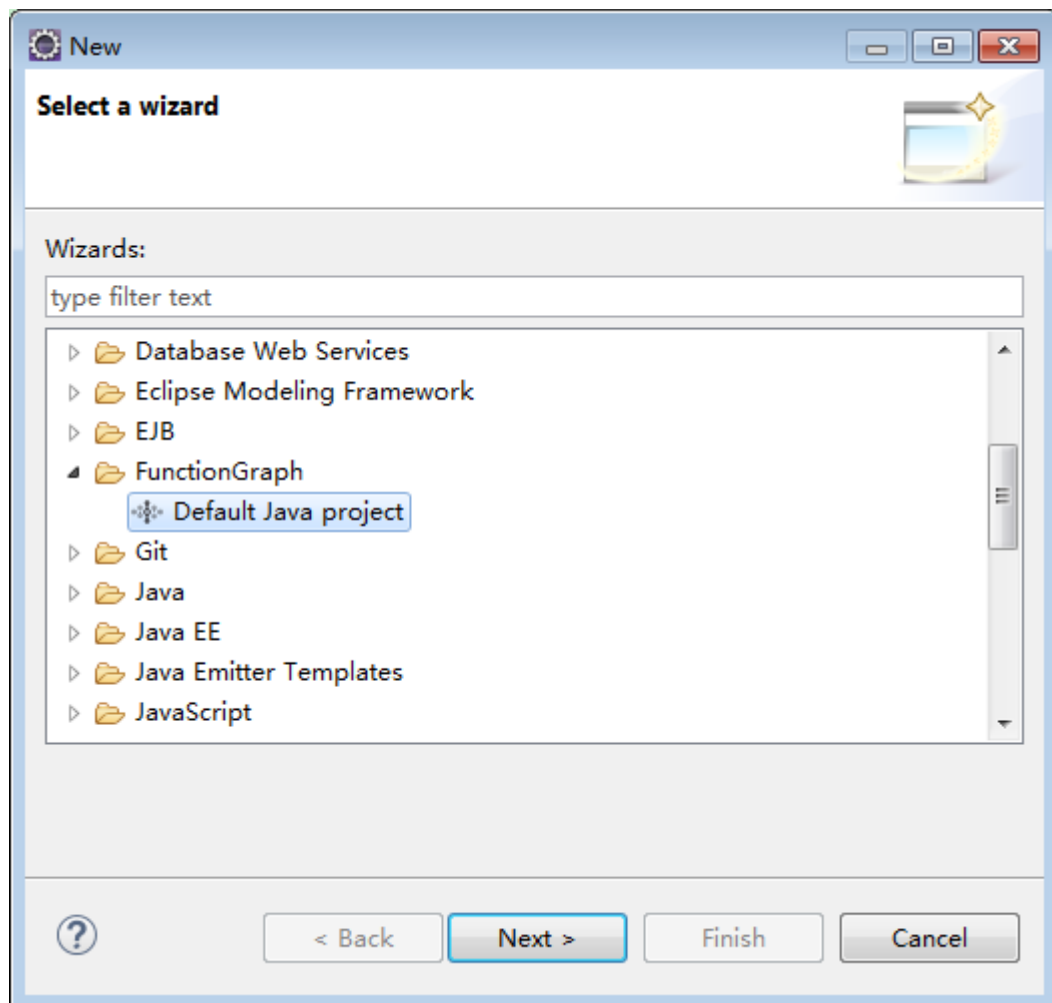
**步骤3** 打开Eclipse，单击“File”，选择“New > Other”，如[图7-6](#)所示。

图 7-6 新建模板



步骤4 选择“FunctionGraph”文件下的“Default Java project”节点。如图7-7所示。

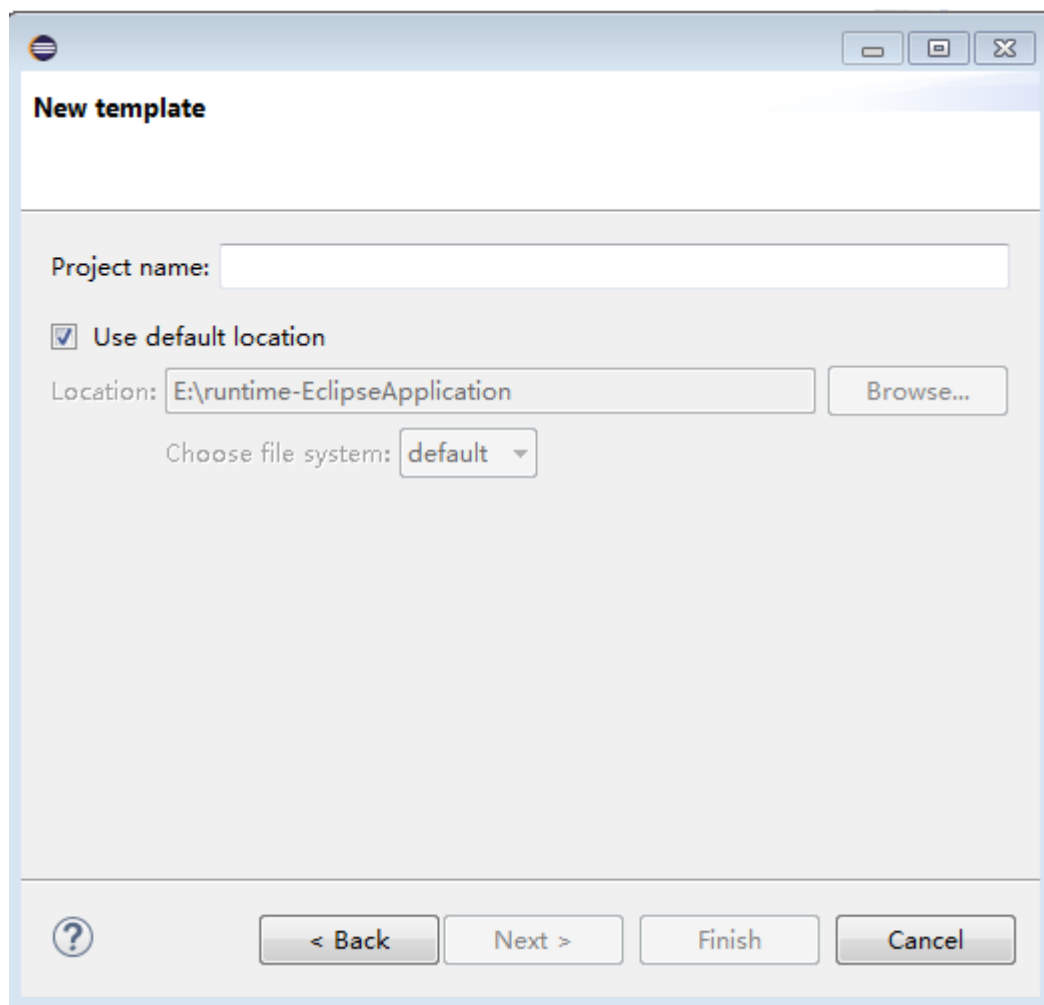
图 7-7 选择默认 Java 模板



**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7-8所示。



图 7-8 完成创建



----结束

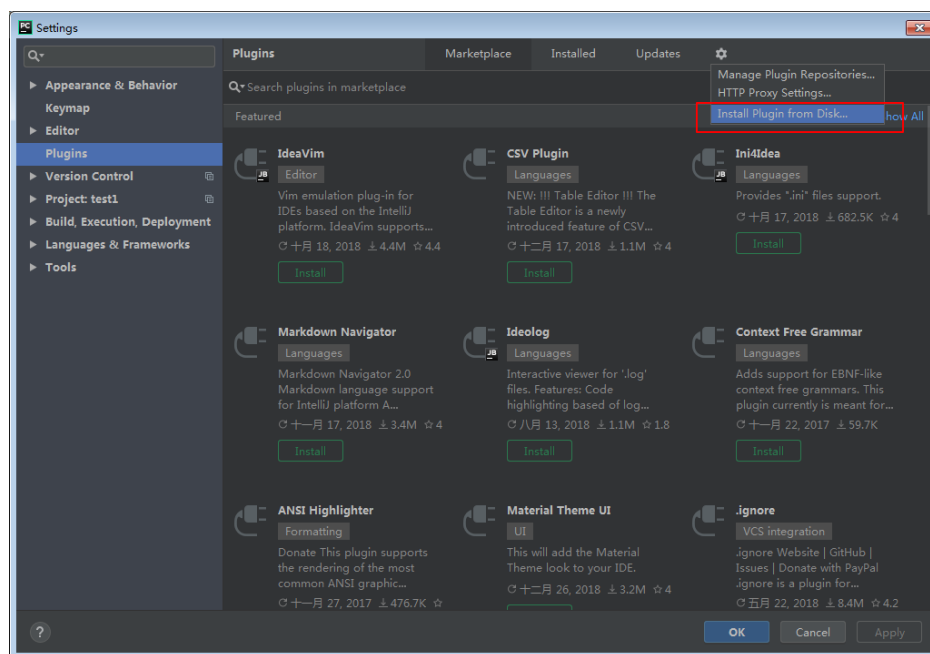
## 7.3 PyCharm-Plugin

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](#)）。

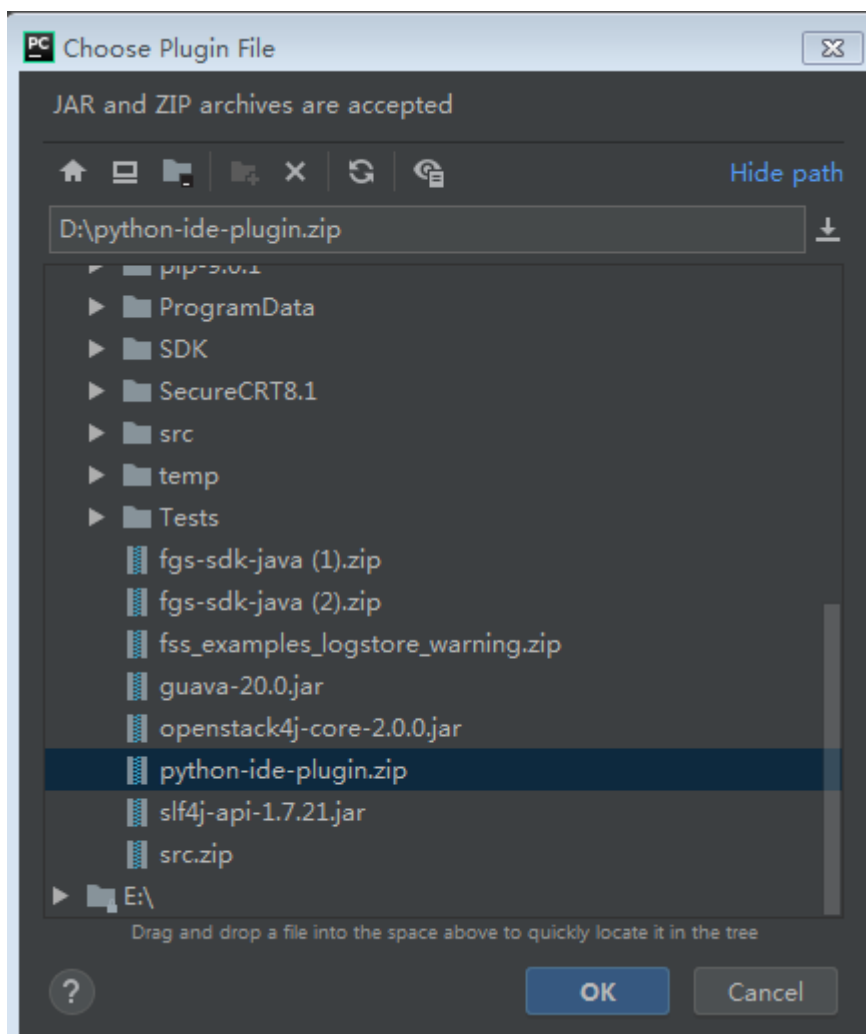
**步骤2** 打开JetBrains PyCharm，单击“File”菜单，选择“Settings”，在弹出界面的菜单中选择“Plugins”页面，单击右上角设置按钮中的“Install plugin from disk...”，如[图7-9](#)所示。

图 7-9 安装 Plugins



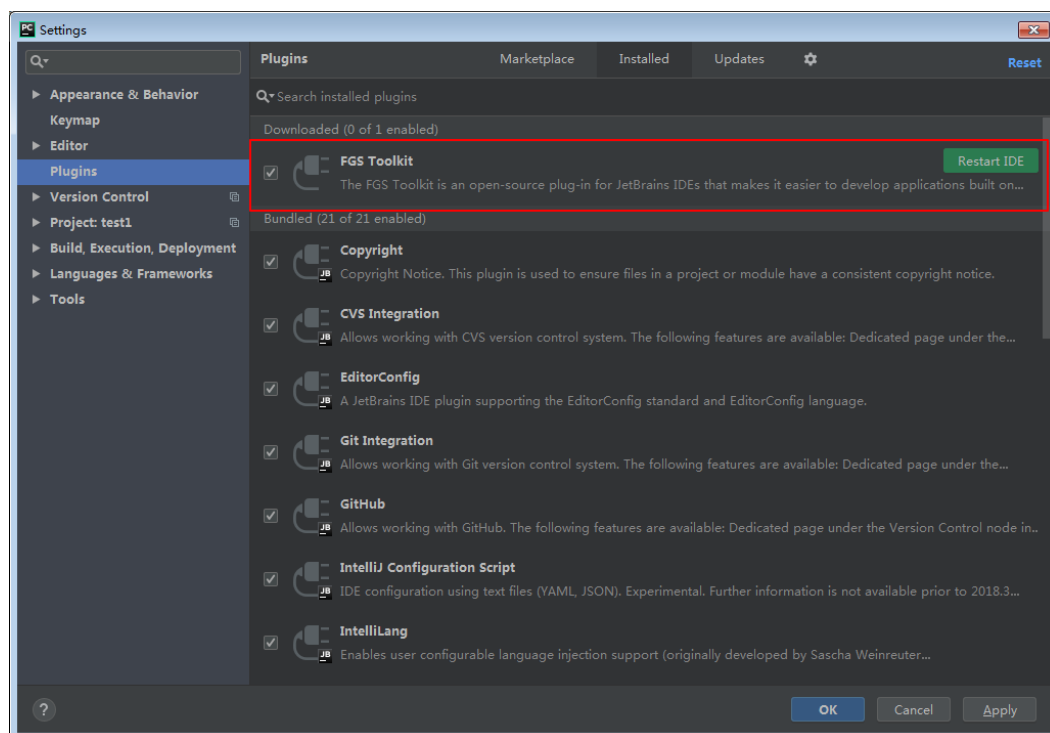
步骤3 在弹出的界面中，选择插件包，单击“OK”，如图7-10所示。

图 7-10 选择插件包



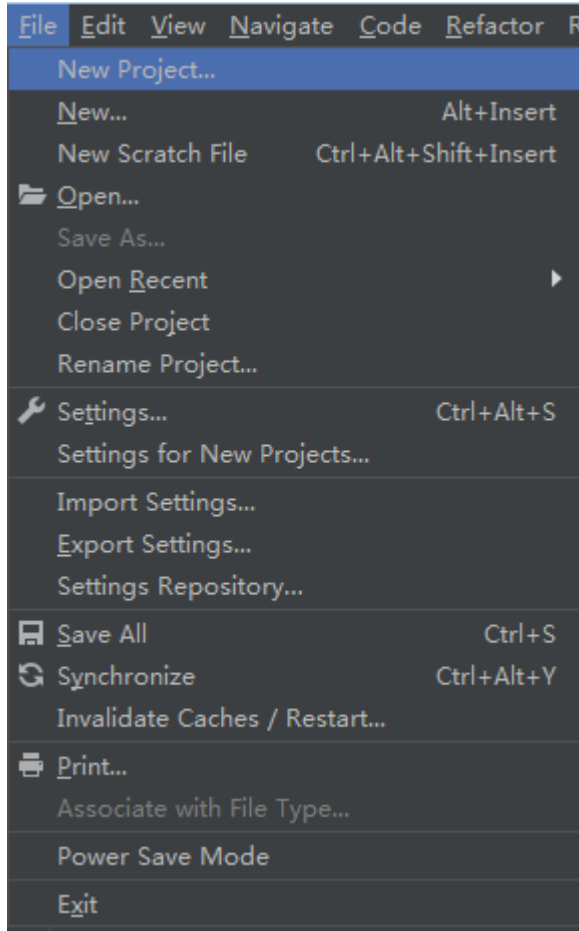
步骤4 在插件列表中，勾选插件名称，单击“Restart IDE”，如图7-11所示。

图 7-11 重启 IDE



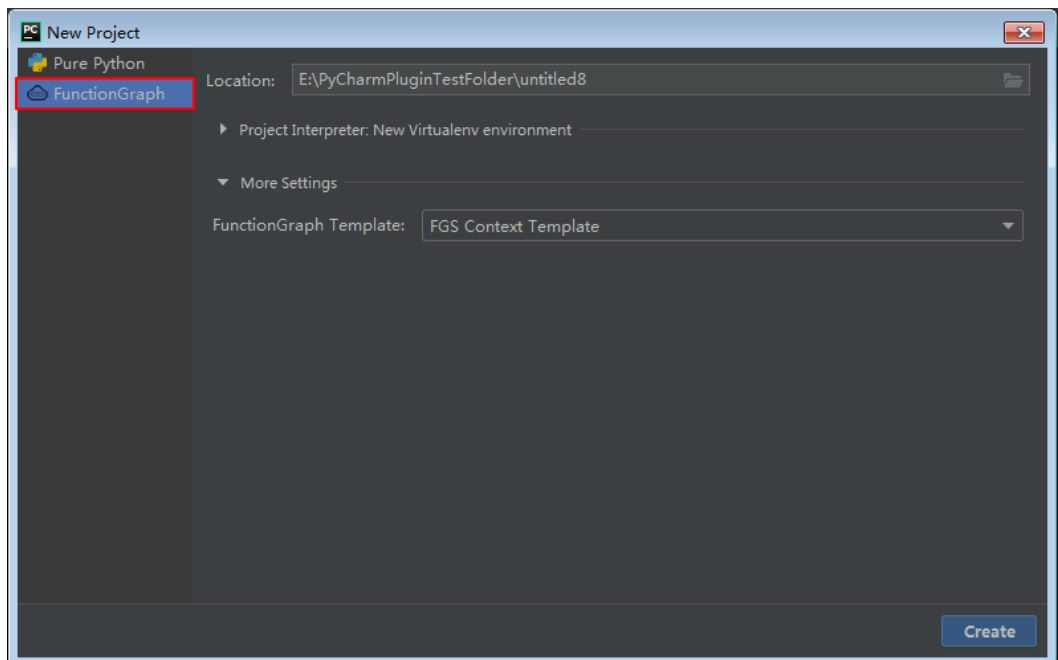
步骤5 单击“File”菜单，选择“New Project”，如图7-12所示。

图 7-12 新建工程



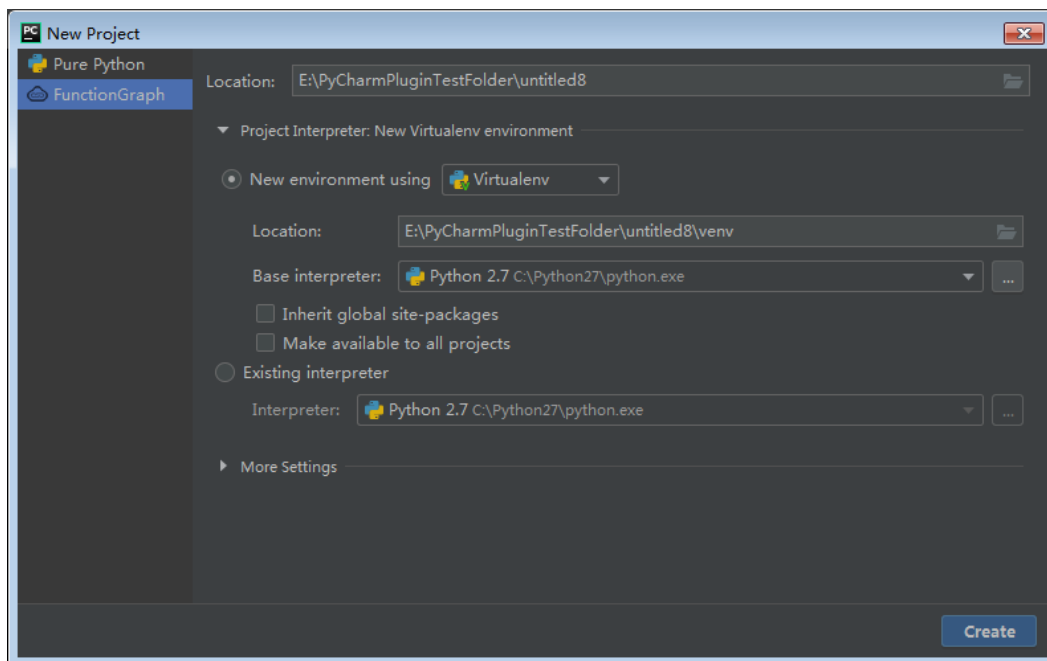
步骤6 在弹出的新建工程页面中，选择“FunctionGraph”，如图7-13所示。

图 7-13 FunctionGraph



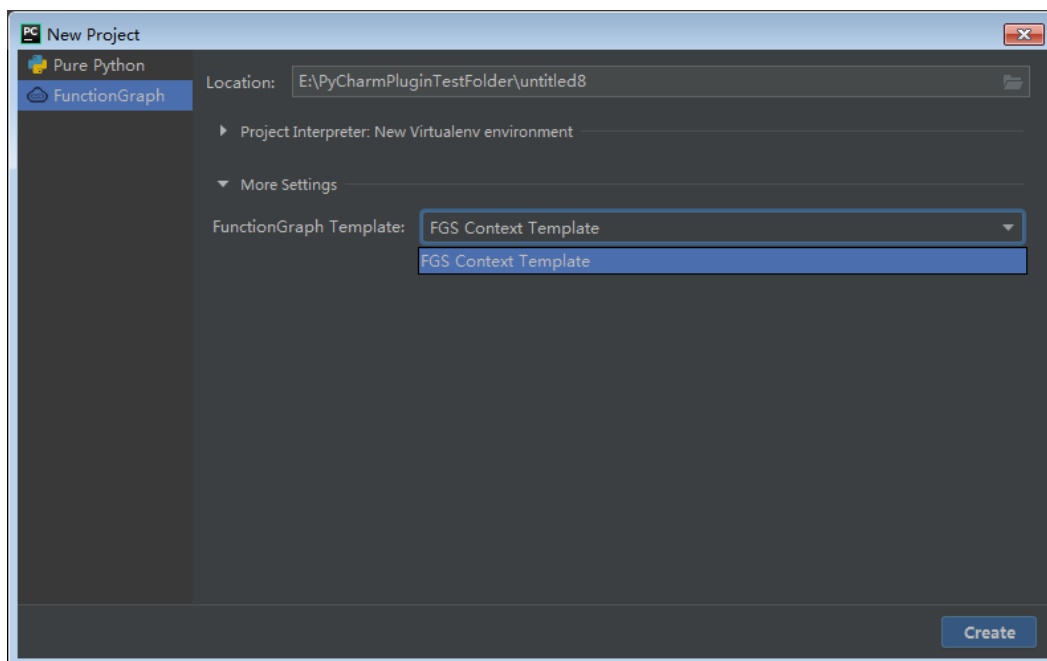
**步骤7** 在“Location”栏中选择工程的路径，在“Project Interpreter: New Virtualenv environment”中选择使用python的版本。如图7-14所示。

图 7-14 选择版本



**步骤8** 在“More Settings”中选择要创建的模板，如图7-15所示。

图 7-15 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

# A 修订记录

表 A-1 修订记录

日期	修订记录
2022-11-30	本次变更： 目录架构调整。
2020-10-30	第一次发布。