FunctionGraph

Developer Guide

Issue 01

Date 2026-01-08

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2026. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Cloud Computing Technologies Co., Ltd.

Trademarks and Permissions


and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei Cloud and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Cloud Computing Technologies Co., Ltd.

Address: Huawei Cloud Data Center Jiaoxinggong Road
Qianzhong Avenue
Gui'an New District
Gui Zhou 550029
People's Republic of China

Website: https://www.huaweicloud.com/intl/en-us/

Contents

1 Overview 1
1.1 Function Development 1
1.2 Supported Event Sources 3
1.3 Function Project Packaging Rules 9
1.4 Referencing DLLs in Functions 14
2 Initializer 16
2.1 What Is Initializer? 17
2.2 Initializer Definition 19
3 Developing Functions 22
3.1 Developing Functions in Node.js 22
3.2 Developing Functions in Python 26
3.3 Developing Functions in Java 29
3.3.1 Developing Functions in Java (Using Eclipse) 29
3.4 Developing Functions in Go 40
3.5 Developing Functions in C# 43
3.5.1 C# Function Development 43
3.5.2 JSON Serialization and Deserialization 46
3.5.2.1 Using .NET Core CLI 46
3.5.2.2 Using Visual Studio 48
3.6 Developing Functions in PHP 55
3.7 Developing Functions in Cangjie 58
3.7.1 Developing Functions in Cangjie (Using Visual Studio Code) 58
3.7.2 Developing an Event Function 60
4 Developing Functions with Plug-ins 63
4.1 Eclipse Plug-in 63
4.2 PyCharm Plug-in 66

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

1 Overview

1.1 Function Development

Supported Runtimes

The Node.js, Java, Python, Go, C#, PHP, Cangjie, and custom runtimes are supported. **Table 1-1** lists the supported runtimes.

NOTE

You are advised to use the latest runtime version.

Table 1-1 Runtime description

Runtime	Supported Version	SDK Download Link
Node.js	6.10, 8.10, 10.16, 12.13, 14.18, 16.17, 18.15, 20.15	-
Python	2.7, 3.6, 3.9, 3.10, 3.12	-
Java	8, 11, 17, 21	**Java SDK** **NOTE** The Java runtime has integrated with Object Storage Service (OBS) SDKs.
Go	1.x	**Go1.8.3 SDK**
C#	.NET Core 2.1, .NET Core 3.1, .NET Core 6.0, .NET Core 8.0	**CsharpSDK**
PHP	7.3, 8.3	-
Custom	-	-
Cangjie	1.0	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Third-Party Components Integrated with the Node.js Runtime

Table 1-2 Third-party components integrated with the Node.js runtime

Name	Usage	Version
q	Asynchronous method encapsulation	1.5.1
co	Asynchronous process control	4.6.0
lodash	Common tool and method library	4.17.10
esdk-obs-nodejs	OBS SDK	2.1.5
express	Simplified web-based application development framework	4.16.4
fgs-express	Uses the Node.js application framework to run serverless applications and REST APIs in FunctionGraph and API Gateway. This component provides an example of using the Express framework to build serverless web applications or services and RESTful APIs.	1.0.1
request	Simplifies HTTP invocation and supports HTTPS and redirection.	2.88.0

Non-Standard Libraries Integrated with the Python Runtime

Table 1-3 Non-standard libraries integrated with the Python Runtime

Library	Usage	Version
dateutil	Date and time processing	2.6.0
requests	HTTP library	2.7.0
httplib2	HTTP client	0.10.3

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Library	Usage	Version
numpy	Mathematical computation	1.13.1
redis	Redis client	2.10.5
obsclient	OBS client	-
smnsdk	Simple Message Notification (SMN) access	1.0.1

Sample Project Packages

Table 1-4 provides the links for downloading the sample project packages mentioned in this document. You can download the project packages to a local path and upload them when creating functions.

Table 1-4 Download links of the sample project packages

Function	Project Package
Node.js function	**fss_examples_nodejs6.10.zip**
Python function	**fss_examples_python2.7.zip**
Java function	**fss_example_java8.jar**
Go function	**fss_examples_go1.8.zip**
C# function	**fss_example_csharp2.0** and **fss_example_csharp2.1**
PHP function	**fss_examples_php7.3.zip**

1.2 Supported Event Sources

This section describes the cloud services that can be configured as event sources for your FunctionGraph functions. After you preconfigure the event source mapping, these event sources automatically invoke the relevant function when detecting events.

SMN

Simple Message Notification (SMN) sends messages to email addresses, mobile phones, or HTTP/HTTPS URLs. If you create a function with an SMN trigger, messages published to a specified topic will be passed as a parameter (**SMN example event**) to invoke the function. Then, the function processes the event, for example, publishing messages to other SMN topics or sending them to other

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

cloud services. For details, see section "Using an SMN Trigger" in the *FunctionGraph User Guide*.

API Gateway

API Gateway is an API hosting service that helps enterprises to build, manage, and deploy APIs at any scale. With API Gateway, your function can be invoked through HTTPS by using a custom REST API and a specified backend. You can map each API operation (such as, GET and PUT) to a specific function. API Gateway invokes the relevant function when an HTTPS request (**API Gateway example event**) is sent to the API backend. For details, see section "Using an APIG Trigger" in the *FunctionGraph User Guide*.

OBS

Object Storage Service (OBS) is a stable, secure, efficient, and easy-to-use cloud storage service. You can create a function to process OBS bucket events, for example, creating and deleting objects. When an image is uploaded to a specified bucket, OBS invokes the function to read the image and create a thumbnail. For details, see section "Using an OBS Trigger" in the *FunctionGraph User Guide*.

Table 1-5 Event types supported by OBS

Event	Description
ObjectCreated	All kinds of object creation operations, including PUT, POST, and COPY of objects, as well as the merging of parts.
Put	Use the PUT method to upload objects.
Post	Use the POST method to upload objects.
Copy	Use the COPY method to replicate objects.
CompleteMultipartUpload	Merge parts of multi-part tasks.
ObjectRemoved	Delete objects.
Delete	Delete objects by versions.
DeleteMarkerCreated	Delete objects without specifying versions.

NOTE

Multiple event types can be used on the same object. For example, if you have selected **Put**, **Copy**, and **Delete** in an event notification rule, a notification message will be sent to you when the specified object is uploaded to, copied to, or deleted from the bucket. **ObjectCreated** contains **Put**, **Post**, **Copy**, and **CompleteMultipartUpload**. If you select **ObjectCreated**, the others are automatically selected and cannot be selected again. Similarly, if you select **ObjectRemoved**, **Delete** and **DeleteMarkerCreated** are automatically selected and cannot be selected again.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Timer

You can schedule a timer (**timer example event**) to invoke your code based on a fixed rate of minutes, hours, or days or a cron expression. For details, see section "Using a Timer Trigger" in the *FunctionGraph User Guide*.

LTS

Log Tank Service (LTS) collects and stores logs, allowing you to query them in real time. If you create a function with an LTS trigger, subscribed logs collected by LTS will be passed as a parameter (**LTS example event**) to invoke the function. Then, the function processes or analyzes the logs, or loads the logs to other systems. For details, see section "Using an LTS Trigger" in the *FunctionGraph User Guide*.

CTS

Cloud Trace Service (CTS) collects operation records of subscribed cloud resources. If you create a function with a CTS trigger, collected operation records of specified cloud services will be passed as a parameter (**CTS example event**) to invoke the function. Then, the function analyzes and processes key information in the operation records, automatically recovers system or network modules, or reports alarms to service personnel by SMS or email. For details, see section "Using a CTS Trigger" in the *FunctionGraph User Guide*.

DMS for Kafka

DMS for Kafka is a message queuing service that provides Kafka premium instances. If you create a Kafka trigger for a function, when a message is sent to a Kafka instance topic, FunctionGraph will retrieve the message and trigger the function to perform other operations. For details, see section "Using a Kafka Trigger" in the *FunctionGraph User Guide*.

Cloud Eye

Cloud Eye is a multi-dimensional resource monitoring platform. FunctionGraph is interconnected with Cloud Eye to report metrics, allowing you to view function metrics and alarm messages through Cloud Eye. For more information about metrics, see section "Viewing Function Metrics" in the *FunctionGraph User Guide*.

DMS for RabbitMQ

When a DMS (for RabbitMQ) trigger is used, FunctionGraph periodically polls for new messages in a specific topic bound to the exchange of a RabbitMQ instance and passes the messages as input parameters to invoke functions.

Example Events

- SMN example event

  In the following example, the function name is **test** and the topic name is **serverless_Test**. **subject** is the message header, and **message** is the message body.

  ```
  {
    "record": [{
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
        "event_version": "1.0",
        "smn": {
            "message_attributes": null,
            "subject": "This is the message title of smn trigger",
            "message_id": "a5dbd701d70d4ac2a153e7dd6dd9b601",
            "topic_urn": "urn:smn:xx-xxxxx-
x:bb8695913bb74682b9ca82a8803b60d8:serverless_Test",
            "type": "notification",
            "message": "This is the message body of smn trigger",
            "timestamp": "2017-11-15T04:02:24Z"
            },
        "event_source": "smn",
        "event_subscription_urn": "urn:fss:xx-xxxxx-
xxx:bb8695913bb74682b9ca82a8803b60d8:function:default:obsTrigger-Test1:latest"
    }],
    "functionname": "test",
    "requestId": "7c307f6a-cf68-4e65-8be0-4c77405a1b2c",
    "timestamp": "Wed Nov 15 2017 12:00:00 GMT+0800 (CST)"
}
```

- API Gateway example event

  NOTE

  When a function is invoked due to an API Gateway event, the request body is encrypted by default.

```
{
    "httpMethod": "GET",     //Request method
    "path": "/test/hello",   //Request path
    "pathParameters": {      //Path parameters
        "proxy": "hello"
    },
    "queryStringParameters": { //Query parameters
        "name": "me"
    },
    "headers": {               //Request header
        "x-stage": "RELEASE",
        "Host": "3f96e175-d5e4-4d4b-ae7e-f6d264e63b23-apigw.xx-xxx-x.xxx.com",
        "User-Agent": "lua-resty-http/0.10 (Lua) ngx_lua/10008",
        ...
    },
    "body": "...",              //Request body
    "isBase64Encoded":false/true,  //Indicates whether the request body is encoded using
Base64.
    "requestContext": {
        "stage": "test",          //Environment name
        "requestId": "dd4337362c02c7d77299e78781beb4b1",
        "apiId": "41b45ea3-70b5-11e6-b7bd-69b5aaebc7d9"
    },
}
```

  The function returns characters strings by using the following structure.

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

- DIS example event

  In the following example, the stream name is **dis-swtest**. This example shows the format of a request received by a function associated with the DIS trigger.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
{
   "ShardID": "shardId-0000000000",
   "Message": {
      "next_partition_cursor":
"eyJnZXRJdGVyYXRvclBhcmFtIjp7InN0cmVhbS1uYW1lIjoiZGlzLXN3dGVzdCIsInBhcnRpdGlvbi
1pZCI6InNoYXJkSWQtMDAwMDAwMDAwMCIsImN1cnNvci10eXBlIjoiVFJJTV9IT1JJWk9OIiwi
c3RhcnRpbmctc2VxdWVuY2UtbnVtYmVyIjoiNCJ9LCJnZW5lcmF0ZVRpbWVzdGFtcCI6MTUw
OTYwNjM5MjE5MX0",
      "records": [{
         "partition_key": "shardId_0000000000",
         "data": "PEJ1ZmZlcj48L0J1ZmZlcj4=",
         "sequence_number": "0"
      },
      {
         "partition_key": "shardId_0000000000",
         "data": "PEJ1ZmZlcj48L0J1ZmZlcj4=",
         "sequence_number": "1"
      },
      {
         "partition_key": "shardId_0000000000",
         "data": "PEJ1ZmZlcj48L0J1ZmZlcj4=",
         "sequence_number": "2"
      },
      {
         "partition_key": "shardId_0000000000",
         "data": "PEJ1ZmZlcj48L0J1ZmZlcj4=",
         "sequence_number": "3"
      }],
      "millis_behind_latest": ""
   },
   "Tag": "latest",
   "StreamName": "dis-swtest"
}
```


- Timer example event

```
{
  "version": "v1.0",
  "time": "2006-01-02T15:04:05-07:00",  // Local time
  "trigger_type": "TIMER",   // Trigger type
  "trigger_name": "Trigger name"
  "user_event": "User-defined event"   //Event configured when the timer trigger is created
}
```

- LTS example event


```
{
   "lts": {
   "data":
"eyJsb2dzIjpbIntcIm1lc3NhZ2VcIjpcIjIwMTgtMDYtMjYvMTg6NDA6NTMgW0lORl0gW2Nvbm
ZpZy5nbzo3Ml0gU3VjY2Vzc2Z1bGx5IGxvYWRlZCBnZW5lcmFsIGNvbmZpZ3VyYXRpb24gZml
sZVxcclxcblwiLFwidGltZVwiOjE1MzAwMDk2NTMwNTksXCJob3N0X25hbWVcIjpcImVjcy10Z
XN0YWdlbnQubm92YWxvY2FsXCIsXCJpcFwiOlwiMTkyLjE2OC4xLjk4XCIsXCJwYXRoXCI6XCIv
dXNyL2xvY2FsL3RlbGVzY29wZS9sb2cvY29tbW9uLmxvZ1wiLFwibG9nX3VpZFwiOlwiNjYzZD
Y5MzAtNzkyZC0xMWU4LThiMDgtMjg2ZWQ0ODhjZTcwXCIsXCJsaW5lX25vXCI6NjE1fSIsIntcI
m1lc3NhZ2VcIjpcIjIwMTgtMDYtMjYvMTg6NDA6NTMgW1dST0gW2NvbmZpZy5nbzo4Ml0g
VGhlIHByb2plY3RJZCBvciBpbnN0YW5jZUlkIG9mIGNvbmZpZy5qc29uIGlzIG5vdCBjb25zaXN0
ZW50IHdpdGggbWV0YWRhdGEsIHVzZSBtZXRhZGF0YS5cXG5cIixcInRpbWVcIjoxNTMwMDA
5NjUzMDU5LFwiaG9zdF9uYW1lXCI6XCJlY3MtdGVzdGFnZW50Lm5vdmFsb2NhbFwiLFwiaXB
cIjpcIjE5Mi4xNjguMS45OFwiLFwicGF0aFwiOlwiL3Vzci9sb2NhbC90ZWxlc2NvcGUvbG9nL2N
vbW1vbi5sb2dcIixcImxvZ191aWRcIjpcIjY2M2Q2OTMwLTc5MmQtMTFlOC04YjA5LTI4NmVk
NDg4Y2U3MFwiLFwibGluZV9ub1wiOjYxNn0iLCJ7XCJtZXNzYWdlXCI6XCIgSW4gY29uZi5qc29
uLCBwcm9qZWN0SWQgaXMgW10sIGluc3RhbmNlSWQgaXMgW10uIE1ldGFEYXRhIGlzIHs0
NTQzMjkzYS01YjJjLTQ0YzQtYjdhMC1kZTIxOGY3ZjJmYTYgNjI4MGUxNzBiZDkzNGY2MGE0Z
Dg1MWNmNWNhMDUxMjkgIH1cXHJcXG5cIixcInRpbWVcIjoxNTMwMDA5NjUzMDU5LFwia
```

G9zdF9uYW1lXCI6XCJlY3MtdGVzdGFnZW50Lm5vdmFsb2NhbFwiLFwiaXBcIjpcIjE5Mi4xNjguMS45OFwiLFwicGF0aFwiOlwiL3Vzci9sb2NhbC90ZWxlc2NvcGUvbG9nL2NvbW1vbi5sb2dcIixcImxvZ191aWRcIjpcIjY2M2Q2OTMwLTc5MmQtMTFlOC04YjBhLTI4NmVkNDg4Y2U3MFwiLFwibGluZV9ub1wiOjYxN30iXSwib3duZXIiOiI2MjgwZTE3MGJkOTM0ZjYwYTRkODUxY2Y1Y2EwNTEyOSIsImxvZ19ncm91cF9pZCI6Ijk3YTlkMjg0LTQ0NDgtMTFlOC04ZmE0LTI4NmVkNDg4Y2U3MCIsImxvZ190b3BpY19pZCI6IjFhOTY3NWE3LTc4NGQtMTFlOC05ZjcwLTI4NmVkNDg4Y2U3MCJ9"
    }
}

- CTS example event

```
{
    "cts": {
        "time": "2018/06/26 08:54:07 GMT+08:00", //Timestamp of the sender
        "user": { //Information about the user that initiates the request
            "name": "userName",
            "id": "5b726c4fbfd84821ba866bafaaf56aax",
            "domain": {
                "name": "domainName",
                "id": "b2b3853af40448fcb9e40dxj89505ba"
            }
        },
        "request": {}, //Content of the trace request
        "response": {}, //Content of the trace response
        "code": 204, //Trace response codes, such as, 200 or 400
        "service_type": "FunctionGraph", //Name of the sender in abbreviated form, such as VPC and ECS
        "resource_type": "graph", //Resource type of the sender, such as VM and VPN
        "resource_name": "workflow-2be1", //Resource name, for example, the name of a VM on Elastic Cloud Server (ECS)
        "resource_id": "urn:fgs:xx-xxx-x:2d1d891d93054bbaa69b9e866c0971ac:graph:workflow-2be1", //Resource ID, for example, the ID of a VM on ECS
        "trace_name": "deleteGraph", //Trace name, such as startServer and shutDown
        "trace_type": "ConsoleAction", //Type of the trace source, such as ApiCall
        "record_time": "2018/06/26 08:54:07 GMT+08:00", //Time when CTS receives the trace
        "trace_id": "69be64a7-0233-11e8-82e4-e5d37911193e", //Trace ID
        "trace_status": "normal"
    }
}
```

- Kafka example event

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

- RabbitMQ example event

```
{
    "event_version": "v1.0",
    "event_time": 1576737962,
    "trigger_type": "RABBITMQ",
    "region": "{region}",
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

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

Table 1-6 Parameters

Parameter	Type	Example Value	Description
event_version	String	v1.0	Event version
Region	String	xx-xxx-x	Region where the RabbitMQ instance is located. Set it based on the site requirements.
instance_id	String	81335d56-b9fe-4679-ba95-7030949cc76b	RabbitMQ instance ID

1.3 Function Project Packaging Rules

Packaging Rules

In addition to inline code editing, you can create a function by uploading a local ZIP file or JAR file, or uploading a ZIP file from Object Storage Service (OBS). **Table 1-7** describes the rules for packaging a function project.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 1-7 Function project packaging rules

Runtime	JAR File	ZIP File	ZIP File on OBS
Node.js	Not supported.	• If the function project files are saved under the **~/Code/** directory, select and package all files under this directory to ensure that the function handler is under the root directory after the ZIP file is decompressed. • If the function project uses third-party dependencies, package the dependencies into a ZIP file, and import the ZIP file on the function code page. Alternatively, package the third-party dependencies and the function project files together.	Compress project files into a ZIP file and upload it to an OBS bucket.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Runtime	JAR File	ZIP File	ZIP File on OBS
PHP	Not supported.	• If the function project files are saved under the **~/Code/** directory, select and package all files under this directory to ensure that the function handler is under the root directory after the ZIP file is decompressed. • If the function project uses third-party dependencies, package the dependencies into a ZIP file, and import the ZIP file on the function code page. Alternatively, package the third-party dependencies and the function project files together.	Compress project files into a ZIP file and upload it to an OBS bucket.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Runtime	JAR File	ZIP File	ZIP File on OBS
Python	Not supported.	• If the function project files are saved under the **~/Code/** directory, select and package all files under this directory to ensure that the function handler is under the root directory after the ZIP file is decompressed. • If the function project uses third-party dependencies, package the dependencies into a ZIP file, and import the ZIP file on the function code page. Alternatively, package the third-party dependencies and the function project files together.	Compress project files into a ZIP file and upload it to an OBS bucket.
Java	If the function does not reference third-party components, compile only the function project files into a JAR file.	If the function references third-party components, compile the function project files into a JAR file, and compress all third-party components and the function JAR file into a ZIP file.	Compress project files into a ZIP file and upload it to an OBS bucket.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Runtime	JAR File	ZIP File	ZIP File on OBS
Go 1.x	Not supported.	Compress project files into a ZIP file, and ensure that the name of the dynamic library file is consistent with the handler plugin name. For example, if the name of the dynamic library file is **testplugin.so**, set the handler plugin name to **testplugin.Handler**. **Handler** indicates the function handler.	Compress project files into a ZIP file and upload it to an OBS bucket.
C#	Not supported.	Compress project files into a ZIP file. The ZIP file must contain the following files: *Project_name***.deps.json**, *Project_name***.dll**, *Project_name***.runtimeconfig.json**, *Project_name***.pdb**, and **HC.Serverless.Function.Common.dll**.	Compress project files into a ZIP file and upload it to an OBS bucket.
Custom	Not supported.	Compress project files into a ZIP file. The ZIP file must contain a bootstrap file.	Compress project files into a ZIP file and upload it to an OBS bucket.
Cangjie	Not supported.	Zip the compiled file and ensure that the name of the binary file is consistent with that of the handler. For example, if the name of the binary file is **libuser_func_test_success.so**, set the name of the handler to **libuser_func_test_success.so**.	Compress project files into a ZIP file and upload it to an OBS bucket.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Example ZIP Project Packages

- Example directory of a Nods.js project package

```
Example.zip                  Example project package
|--- lib                     Service file directory
|--- node_modules            NPM third-party component directory
|--- index.js                .js handler file (mandatory)
|--- package.json            NPM project management file
```

- Example directory of a PHP project package

```
Example.zip                  Example project package
|--- ext                     Extension library directory
|--- pear                    PHP extension and application repository
|--- index.php               PHP handler file
```

- Example directory of a Python project package

```
Example.zip                  Example project package
|--- com                     Service file directory
|--- PLI                     Third-party dependency PLI directory
|--- index.py                .py handler file (mandatory)
|--- watermark.py            .py file for image watermarking
|--- watermark.png           Watermarked image
```

- Example directory of a Java project package

```
Example.zip                  Example project package
|--- obstest.jar             Service function JAR file
|--- esdk-obs-java-3.20.2.jar          Third-party dependency JAR file
|--- jackson-core-2.10.0.jar           Third-party dependency JAR file
|--- jackson-databind-2.10.0.jar       Third-party dependency JAR file
|--- log4j-api-2.12.0.jar              Third-party dependency JAR file
|--- log4j-core-2.12.0.jar             Third-party dependency JAR file
|--- okhttp-3.14.2.jar                 Third-party dependency JAR file
|--- okio-1.17.2.jar                   Third-party dependency JAR file
```

- Example directory of a Go project package

```
Example.zip                  Example project package
|--- testplugin.so           Service function package
```

- Example directory of a C# project package

```
Example.zip                                Example project package
|--- fssExampleCsharp2.0.deps.json         File generated after project compilation
|--- fssExampleCsharp2.0.dll               File generated after project compilation
|--- fssExampleCsharp2.0.pdb               File generated after project compilation
|--- fssExampleCsharp2.0.runtimeconfig.json   File generated after project compilation
|--- Handler                               Help file, which can be directly used
|--- HC.Serverless.Function.Common.dll     .dll file provided by FunctionGraph
```

- Example directory of a Cangjie project package

```
fss_example_cangjie.zip                    Example project package
|--- libuser_func_test_success.so          Service function package
```

- Custom

```
Example.zip                  Example project package
|--- bootstrap               Executable boot file
```

1.4 Referencing DLLs in Functions

- By default, the root directory and the **lib** folder in this directory have been configured in the **LD_LIBRARY_PATH** environment variable. You only need to add dynamic link libraries (DLLs) here.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- You can directly modify the **LD_LIBRARY_PATH** variable in the code.
- If the dependent **.so** file is stored in a non-root directory, specify the directory in the **LD_LIBRARY_PATH** variable on the **Configuration** tab page.
- If a library in a mounted file system is used, specify its directory in the **LD_LIBRARY_PATH** variable on the **Configuration** tab page.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

2 Initializer

Overview

An initializer is a logic entry for initializing functions. For a function with an initializer, FunctionGraph invokes the initializer to initialize the function and then invokes the handler to process function requests. For a function without an initializer, FunctionGraph only invokes the handler to process function requests.

Applicable Scenario

FunctionGraph executes a function in the following steps:

1. Allocate container resources to the function.
2. Download function code.
3. Use the runtime to load the function code.
4. Initialize the function.
5. Process the function request and return the result.

Steps **1**, **2**, and **3** are completed after a systematic cold start, and a stable latency can be ensured through proper resource scheduling and process optimization. Step **4** is performed during an application-layer cold start in complex scenarios, such as loading large models for deep learning, building database connection pools, and loading function dependencies.

To reduce the latency caused by an application-layer cold start, FunctionGraph provides the initializer to identify function initialization logic for proper resource scheduling.

Benefits of the Initializer

- Isolate function initialization and request processing to enable clearer program logic and better structured and higher-performance code.
- Ensure a smooth function upgrade to prevent performance loss during the application layer's cold start initialization. Enable new function instances to automatically execute initialization logic before processing requests.
- Identify the overhead of application layer initialization, and accurately determine the time for resource scaling and the quantity of required resources. This feature makes request latency more stable when the application load increases and more function instances are required.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- If there are continuous requests and the function is not updated, the system may still reclaim or update existing containers. Although no code starts on the platform side, there are cold starts on the service side. The initializer can be used to ensure that requests can be processed properly.

Initializer Specifications

The initializer of each runtime has the following features:

- No custom parameters

  The initializer does not support custom parameters and only uses the variables in **context** for logic processing.

- No return values

  No values will be returned for initializer invocation.

- Initialization timeout

  You can set an initialization timeout (≤ 300s) different from the timeout for invoking the handler.

- Time for execution

  Function instances are processes that execute function logic in a container and automatically scale if the number of requests changes. When a new function instance is generated, the system invokes the initializer and then executes the handler logic if the invocation is successful.

- One-time execution

  After each function instance starts, the initializer can only be executed once. If an instance fails to execute the initializer, the instance is abandoned and another instance starts to execute the initializer. A maximum of three attempts are allowed. If the initializer is executed successfully, the instance will only process requests upon invocation and will no longer execute the initializer again within its lifecycle.

- Naming rule

  For all runtimes except Java, the initializer can be named in the format of *[File name].[Initializer name]*, which is similar with the format of a handler name. For Java, a class needs to be defined to implement the predefined initializer.

- Billing

  The initializer execution duration will be billed at the same rate as the function execution duration.

2.1 What Is Initializer?

Introduction

An initializer is a logic entry for initializing functions. For a function with an initializer, FunctionGraph invokes the initializer to initialize the function and then invokes the handler to process function requests. For a function without an initializer, FunctionGraph only invokes the handler to process function requests.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Application Scenarios

FunctionGraph executes a function in the following steps:

1. Allocate container resources to the function.
2. Download the function code.
3. Use the runtime to load the function code.
4. Initialize the function.
5. Process the function request and return the result.

Steps **1**, **2**, and **3** are performed during a systematic cold start, ensuring a stable latency through proper resource scheduling and process optimization. Step **4** is performed during an application-layer cold start in complex scenarios, such as loading large models for deep learning, building database connection pools, and loading function dependencies.

To reduce the latency caused by an application-layer cold start, FunctionGraph provides the initializer to identify function initialization logic for proper resource scheduling.

Benefits of the Initializer

- Function initialization and request processing are isolated to enable clearer program logic and structure better and higher-performance code.
- Smooth function upgrade prevents performance loss caused by cold start initialization on the application layer. After new function instances are started, FunctionGraph automatically executes the initialization logic and then processes requests.
- The overhead of application layer initialization is identified to accurately determine when resource scaling will be performed and how many resources will be required. This feature makes request latency more stable when the application load increases and more function instances are required.
- If there are continuous requests and the function is not updated, the system may still reclaim or update existing containers. Although no code starts on the platform side, there are cold starts on the service side. The initializer can be used to ensure that requests can be processed properly.

Features of the Initializer

The initializer of each runtime has the following features:

- No custom parameters

  The initializer does not support the definition of custom parameters, but only uses the variables in **context** for logic processing.
- No return values

  No values will be returned for initializer invocation.
- Initialization timeout

  You can set an initialization timeout (≤ 300s) different from the timeout for invoking the handler.
- Time for execution

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function instances are processes that execute function logic in a container and automatically scale if the number of requests changes. When a new function instance is generated, the system invokes the initializer and then executes the handler logic if the invocation is successful.

- One-time execution

  After each function instance starts, the initializer can only be executed once. If an instance fails to execute the initializer, the instance is abandoned and another instance is generated. A maximum of three attempts are allowed. If the initializer is executed successfully, the instance will only process requests upon invocation and will no longer execute the initializer again within its lifecycle.

- Naming rule

  For all runtimes except Java, the initializer can be named in the format of *[File name].[Initializer name]*, which is similar with the format of a handler name. For Java, a class needs to be defined to implement the predefined initializer.

- Billing

  The initializer execution duration will be billed at the same rate as the function execution duration.

2.2 Initializer Definition

This section describes how to define the initializer for the runtimes that support inline code editing.

Node.js

FunctionGraph supports the following Node.js runtimes:

- Node.js 6.1 (runtime = Node.js6)
- Node.js 8.9 (runtime = Node.js8)
- Nodejs10.16(runtime = Node.js10)
- Nodejs12.13(runtime = Node.js12)

Initializer syntax:

[File name].[Initializer name]

For example, if the initializer is named **index.initializer**, FunctionGraph loads the initializer function defined in the **index.js** file.

To use Node.js to build initialization logic, define a Node.js function as the initializer. The following is a simple initializer:

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

- Function name

  The function name **exports.initializer** must be the initializer function name specified for a function.

  For example, if the initializer is named **index.initializer**, FunctionGraph loads the initializer function defined in the **index.js** file.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- context

  The **context** parameter contains the runtime information about a function. For example, request ID, temporary AK, and function metadata.

- callback

  The **callback** parameter is used to return the invocation result. The signature of this parameter is **function(err, data)**, which is the same as that of the common **callback** parameter used in Node.js. If the value of **err** is not null, the function will return **HandledInitializationError**. The value of **data** is invalid because no value will be returned for function initialization. You can set the **data** parameter as null as it is set in the previous example.

Python

FunctionGraph supports the following Python runtimes:

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

Initializer syntax:

[File name].[Initializer name]

For example, if the initializer is named **main.my_initializer**, FunctionGraph loads the my_initializer function defined in the **main.py** file.

To use Python to build initialization logic, define a Python function as the initializer. The following is a simple initializer:

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

- Function name

  The function name **my_initializer** must be the initializer function name specified for a function. For example, if the initializer is named **main.my_initializer**, FunctionGraph loads the my_initializer function defined in the **main.py** file.

- context

  The **context** parameter contains the runtime information about a function. For example, request ID, temporary AK, and function metadata.

PHP

FunctionGraph supports the following PHP runtime:

- Php 7.2 (runtime = Php7.2)

Initializer syntax:

[File name].[Initializer name]

For example, if the initializer is named **main.my_initializer**, FunctionGraph loads the my_initializer function defined in the **main.php** file.

To use PHP to build initialization logic, define a PHP function as the initializer. The following is a simple initializer:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

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

- Function name

  The function name **my_initializer** must be the initializer function name specified for a function.

  For example, if the initializer is named **main.my_initializer**, FunctionGraph loads the my_initializer function defined in the **main.php** file.

- context

  The **context** parameter contains the runtime information about a function. For example, request ID, temporary AK, and function metadata.

Java

FunctionGraph supports the following Java runtime:

- Java 8 (runtime = Java8)

  Initializer syntax:

  [Package name].[Class name].[Execution function name]

  For example, if the initializer is named **com.Demo.my_initializer**, FunctionGraph loads the my_initializer function defined in the **com** file.

  To use Java to build the initialization logic, define a Java function as the initializer. The following is a simple initializer:

  public void my_initializer(Context context)

  {

  RuntimeLogger log = context.getLogger();

  log.log(String.format("ak:%s", context.getAccessKey()));

  }

- Function name

  The function name **my_initializer** must be the initializer function name specified for a function.

  For example, if the initializer is named **com.Demo.my_initializer**, FunctionGraph loads the my_initializer function defined in the **com** file.

- context

  The **context** parameter contains the runtime information about a function. For example, request ID, temporary AK, and function metadata.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

3 Developing Functions

3.1 Developing Functions in Node.js

Function Syntax

- Node.js 6.10

  Use the following syntax when creating a handler function in Node.js 6.10:

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

  - **handler**: name of the function that FunctionGraph invokes to execute your code. The name must be consistent with that you define when creating a function.
  - **event**: event parameter defined for the function. The parameter is in JSON format.
  - **context**: runtime information provided for executing the function. For details, see **SDK APIs**.
  - **callback**: used to return the defined **err** and **message** information to the frontend. The general syntax is **callback(err, message)**. You can define the error or message content, for example, a character string.
  - Function handler: **index.handler**.

    The function handler is in the format of *[File name].[Function name]*. For example, if you set the handler to **index.handler** in your function, FunctionGraph will load the handler function defined in the **index.js** file.

- Node.js 8.10 and later

  Node.js 8.10 and later are compatible with the APIs of Node.js 6.10, and supports an **async** handler.

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

  Responses are output through **return**.

Node.js Initializer

FunctionGraph supports the following Node.js runtimes:

- Node.js 6.10

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Node.js 8.10
- Node.js 10.16
- Node.js 12.13
- Node.js 14.18
- Node.js 16.17
- Node.js 18.15
- Node.js 20.15

Initializer syntax:

[File name].[Initializer name]

For example, if the initializer is named **index.initializer**, FunctionGraph loads the initializer function defined in the **index.js** file.

To use Node.js to build initialization logic, define a Node.js function as the initializer. The following is a simple initializer:

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

- Function name

  The function name **exports.initializer** must be the initializer function name specified for a function.

  For example, if the initializer is named **index.initializer**, FunctionGraph loads the **initializer** function defined in the **index.js** file.

- context

  The **context** parameter contains the runtime information about a function. For example, request ID, temporary AK, and function metadata.

- callback

  The **callback** parameter is used to return the invocation result. The signature of this parameter is **function(err, data)**, which is the same as that of the common **callback** parameter used in Node.js. If the value of **err** is not null, the function will return **HandledInitializationError**. The value of **data** is invalid because no value will be returned for function initialization. You can set the **data** parameter to null by referring to the previous example.

SDK APIs

Table 3-1 describes the context methods provided by FunctionGraph.

Table 3-1 Context methods

Method	Description
getRequestID()	Obtains a request ID.
getRemainingTimeInMilliSeconds ()	Obtains the remaining running time of a function.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method	Description
getAccessKey()	Obtains the AK (valid for 24 hours) of an agency. If you use this method, you need to configure an agency for the function. **NOTE** FunctionGraph has stopped maintaining the getAccessKey API in the Runtime SDK. You cannot use this API to obtain a temporary AK.
getSecretKey()	Obtains the SK (valid for 24 hours) of an agency. If you use this method, you need to configure an agency for the function. **NOTE** FunctionGraph has stopped maintaining the getSecretKey API in the Runtime SDK. You cannot use this API to obtain a temporary SK.
getSecurityAccessKey()	Obtains the SecurityAccessKey (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getSecuritySecretKey()	Obtains the SecuritySecretKey (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getSecurityToken()	Obtains the SecurityToken (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getUserData(string key)	Uses keys to obtain the values passed by environment variables.
getFunctionName()	Obtains the name of a function.
getRunningTimeInSeconds ()	Obtains the timeout of a function.
getVersion()	Obtains the version of a function.
getMemorySize()	Obtains the allocated memory.
getCPUNumber()	Obtains CPU usage of a function.
getProjectID()	Obtains a project ID.
getPackage()	Obtains a function group, that is, an app.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method	Description
getToken()	Obtains the token (valid for 24 hours) of an agency. If you use this method, you need to configure an agency for the function.
getLogger()	Obtains the **logger** method provided by the context and returns a log output class. Logs are output in the format of *Time-Request ID-Content* by using the **info** method. For example, use the **info** method to output logs: logg = context.getLogger() logg.info("hello")

NOTICE

Results returned by using the **getToken()**, **getAccessKey()**, and **getSecretKey()** methods contain sensitive information. Exercise caution when using these methods.

Execution Result

The execution result consists of the function output, summary, and log output.

Table 3-2 Description of the execution result

Parameter	Successful Execution	Failed Execution

Function Output	The defined function output information is returned.	A JSON file that contains **errorMessage** and **errorType** is returned. The format is as follows:
		{
		"errorMessage": "",
		"errorType":"",
		}
		errorMessage: Error message returned by the runtime.
		errorType: Error type.

Summary	**Request ID**, **Memory Configured**, **Execution Duration**, **Memory Used**, and **Billed Duration** are displayed.	**Request ID**, **Memory Configured**, **Execution Duration**, **Memory Used**, and **Billed Duration** are displayed.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Successful Execution	Failed Execution
Log Output	Function logs are printed. A maximum of 4 KB logs can be displayed.	Error information is printed. A maximum of 4 KB logs can be displayed.

3.2 Developing Functions in Python

Function Syntax

Syntax for creating a handler function in Python:

def handler (event, context)

- **handler**: name of the function that FunctionGraph invokes to execute your code. The name must be consistent with that you define when creating a function.
- **event**: event parameter defined for the function. The parameter is in JSON format.
- **Context**: runtime information provided for executing the function. For details, see **SDK APIs**.

Python Initializer

FunctionGraph supports the following Python runtimes:

- Python 2.7
- Python 3.6
- Python 3.9
- Python 3.10
- Python 3.12

Initializer syntax:

[File name].[Initializer name]

For example, if the initializer is named **main.my_initializer**, FunctionGraph loads the my_initializer function defined in the **main.py** file.

To use Python to build initialization logic, define a Python function as the initializer. The following is a simple initializer:

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

- Function name

  The function name **my_initializer** must be the initializer function name specified for a function. For example, if the initializer is named **main.my_initializer**, FunctionGraph loads the my_initializer function defined in the **main.py** file.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- context

  The **context** parameter contains the runtime information about a function. For example, request ID, temporary AK, and function metadata.

SDK APIs

Table 3-3 describes the context methods provided by FunctionGraph.

Table 3-3 Context methods

Method	Description
getRequestID()	Obtains a request ID.
getRemainingTimeInMilliSeconds ()	Obtains the remaining running time of a function.
getAccessKey()	Obtains the AK (valid for 24 hours) of an agency. If you use this method, you need to configure an agency for the function. **NOTE** FunctionGraph has stopped maintaining the getAccessKey API in the Runtime SDK. You cannot use this API to obtain a temporary AK.
getSecretKey()	Obtains the SK (valid for 24 hours) of an agency. If you use this method, you need to configure an agency for the function. **NOTE** FunctionGraph has stopped maintaining the getSecretKey API in the Runtime SDK. You cannot use this API to obtain a temporary SK.
getUserData(string key)	Uses keys to obtain the values passed by environment variables.
getSecurityAccessKey()	Obtains the SecurityAccessKey (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getSecuritySecretKey()	Obtains the SecuritySecretKey (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getSecurityToken()	Obtains the SecurityToken (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getFunctionName()	Obtains the name of a function.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method	Description
getRunningTimeInSeconds ()	Obtains the timeout of a function.
getVersion()	Obtains the version of a function.
getMemorySize()	Obtains the allocated memory.
getCPUNumber()	Obtains CPU usage of a function.
getProjectID()	Obtains a project ID.
getPackage()	Obtains a function group, that is, an app.
getToken()	Obtains the token (valid for 24 hours) of an agency. If you use this method, you need to configure an agency for the function.
getLogger()	Obtains the **logger** method provided by the context and returns a log output class. Logs are output in the format of *Time-Request ID-Content* by using the **info** method. For example, use the **info** method to output logs: logg = context.getLogger() logg.info("hello")

NOTICE

Results returned by using the **getToken()**, **getAccessKey()**, and **getSecretKey()** methods contain sensitive information. Exercise caution when using these methods.

Execution Result

The execution result consists of the function output, summary, and log output.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 3-4 Description of the execution result

Parameter	Successful Execution	Failed Execution

Function Output	The defined function output information is returned.	A JSON file that contains **errorMessage**, **errorType**, and **stackTrace** is returned. The format is as follows:
		`{`
		`  "errorMessage": "",`
		`  "errorType": "",`
		`  "stackTrace": []`
		`}`
		errorMessage: Error message returned by the runtime.
		errorType: Error type.
		stackTrace: Stack error information returned by the runtime.

Summary	**Request ID**, **Memory Configured**, **Execution Duration**, **Memory Used**, and **Billed Duration** are displayed.	**Request ID**, **Memory Configured**, **Execution Duration**, **Memory Used**, and **Billed Duration** are displayed.
Log Output	Function logs are printed. A maximum of 4 KB logs can be displayed.	Error information is printed. A maximum of 4 KB logs can be displayed.

3.3 Developing Functions in Java

3.3.1 Developing Functions in Java (Using Eclipse)

Function Syntax

The following is the syntax for creating a handler function in Java:

Scope Return parameter Function name (User-defined parameter, Context)

- *Scope*: It must be defined as **public** for the function that FunctionGraph invokes to execute your code.
- *Return parameter*: user-defined output, which is converted into a character string and returned as an HTTP response. The HTTP response is a JSON string.
- *Function name*: user-defined function name.
- *User-defined parameter*: FunctionGraph supports only one user-defined parameter. For complex parameters, define them as an object and provide data through JSON strings. When invoking a function, FunctionGraph parses the JSON strings as an object.
- *Context*: runtime information provided for executing the function. For details, see **SDK APIs**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

When creating a function in Java, define a handler in the format of *[Package name].[Class name].[Function name]*.

Java Initializer

FunctionGraph supports the following Java runtime:

- Java 8
- Java 11
- Java 17
- Java 21

Initializer syntax:

[Package name].[Class name].[Execution function name]

For example, if the initializer is named **com.Demo.my_initializer**, FunctionGraph loads the my_initializer function defined in the **com.Demo** file.

To use Java to build initialization logic, define a Java function as the initializer. The following is a simple initializer:

```
public void my_initializer(Context context)

{

RuntimeLogger log = context.getLogger();

log.log(String.format("ak:%s", context.getAccessKey()));

}
```

- Function name

  The function name **my_initializer** must be the initializer function name specified for a function.

  For example, if the initializer is named **com.Demo.my_initializer**, FunctionGraph loads the my_initializer function defined in the **com.Demo** file.

- context

  The **context** parameter contains the runtime information about a function. For example, request ID, temporary AK, and function metadata.

SDK APIs

The **Java SDK** provides context, and logging APIs.

- Context APIs

  The context APIs are used to obtain the context, such as agency AK/SK, current request ID, allocated memory space, and number of CPUs, required for executing a function.

  Table 3-5 describes the context APIs provided by FunctionGraph.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 3-5 Context methods

Method	Description
getRequestID( )	Obtains a request ID.
getRemainingTimeInMilligetRunningTimeInSecondsSeconds ( )	Obtains the remaining running time of a function.
getAccessKey( )	Obtains the AK (valid for 24 hours) of an agency. If you use this method, you need to configure an agency for the function. **NOTE** FunctionGraph has stopped maintaining the getAccessKey API in the Runtime SDK. You cannot use this API to obtain a temporary AK.
getSecretKey( )	Obtains the SK (valid for 24 hours) of an agency. If you use this method, you need to configure an agency for the function. **NOTE** FunctionGraph has stopped maintaining the getSecretKey API in the Runtime SDK. You cannot use this API to obtain a temporary SK.
getSecurityAccessKey( )	Obtains the SecurityAccessKey (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getSecuritySecretKey( )	Obtains the SecuritySecretKey (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getSecurityToken( )	Obtains the SecurityToken (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getUserData(string key)	Uses keys to obtain the values passed by environment variables.
getFunctionName( )	Obtains the name of a function.
getRunningTimeInSeconds ( )	Obtains the timeout of a function.
getVersion( )	Obtains the version of a function.
getMemorySize( )	Obtains the allocated memory.
getCPUNumber( )	Obtains CPU usage of a function.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method	Description
getProjectID( )	Obtains a project ID.
getPackage( )	Obtains a function group, that is, an app.
getToken( )	Obtains the token (valid for 24 hours) of an agency. If you use this method, you need to configure an agency for the function.
getLogger( )	Obtains the **logger** method provided by the context. By default, information such as the time and request ID is output.

NOTICE

Results returned by using the **getToken()**, **getAccessKey()**, and **getSecretKey()** methods contain sensitive information. Exercise caution when using these methods.

- Logging API

  Table 3-6 describes the logging API provided in the Java SDK.

  Table 3-6 Logging API

Method	Description
RuntimeLogger()	Records user input logs using the method **log(String string)**.

Developing a Java Function

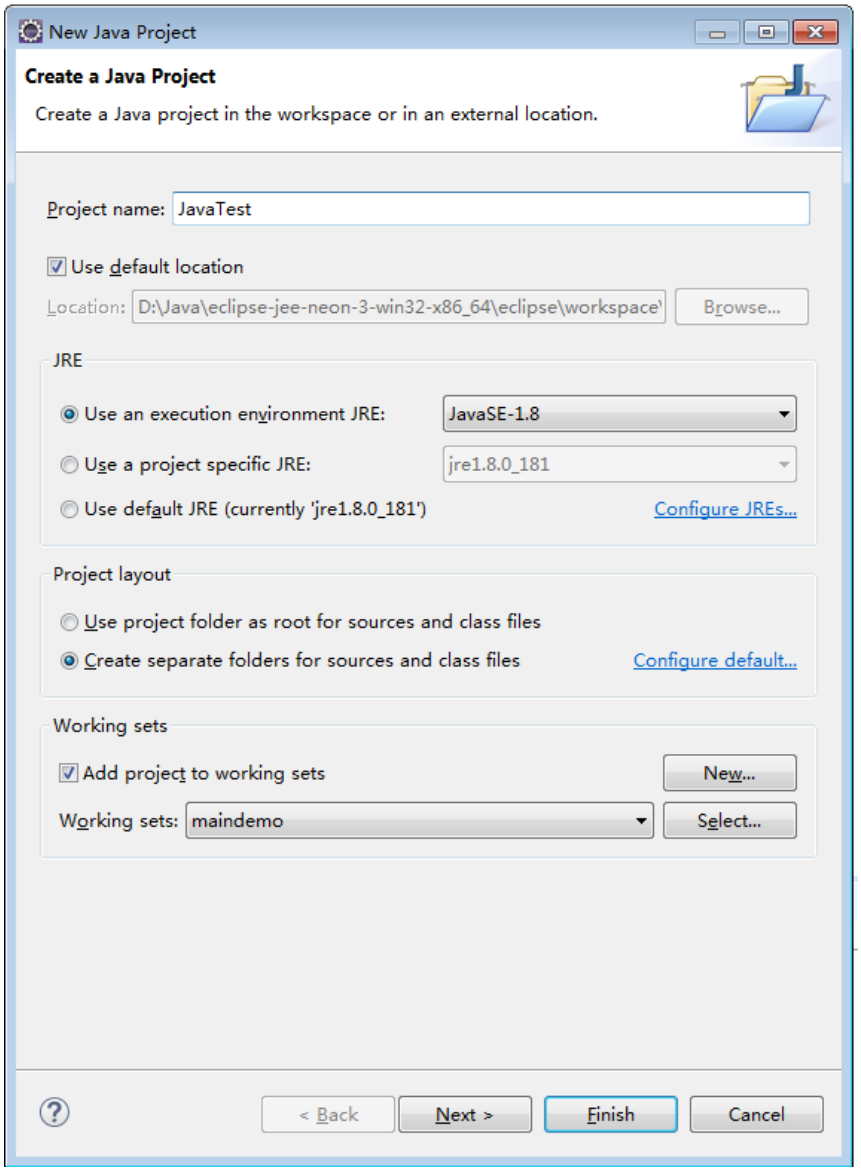
The following shows the procedure of developing a Java function using Eclipse.

Step 1 Create a function project.

1. Configure Eclipse and create a Java project named JavaTest, as shown in Figure 3-1.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 3-1 Creating a project

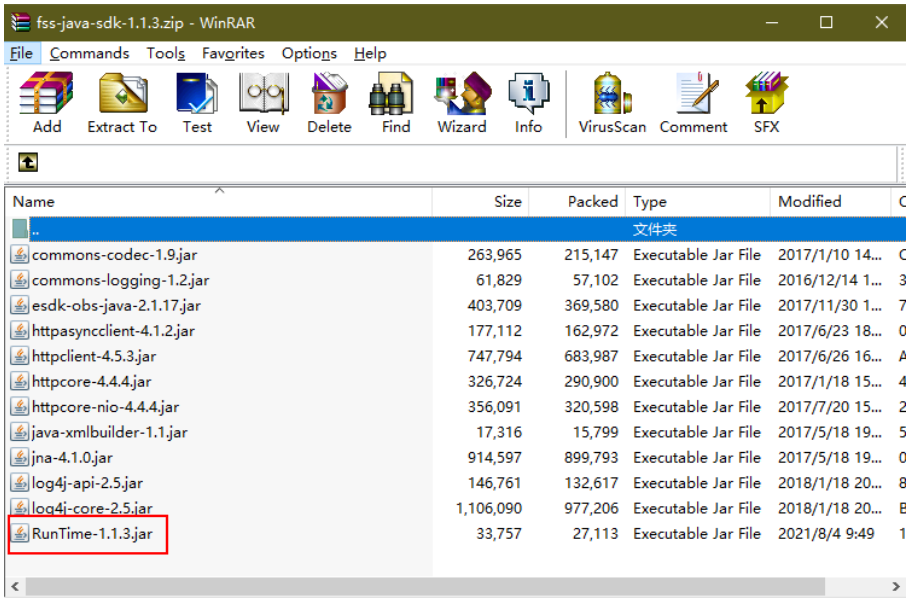


2. Add a dependency to the project.

   Download the **Java SDK** to a local development environment, and decompress the SDK package, as shown in **Figure 3-2**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

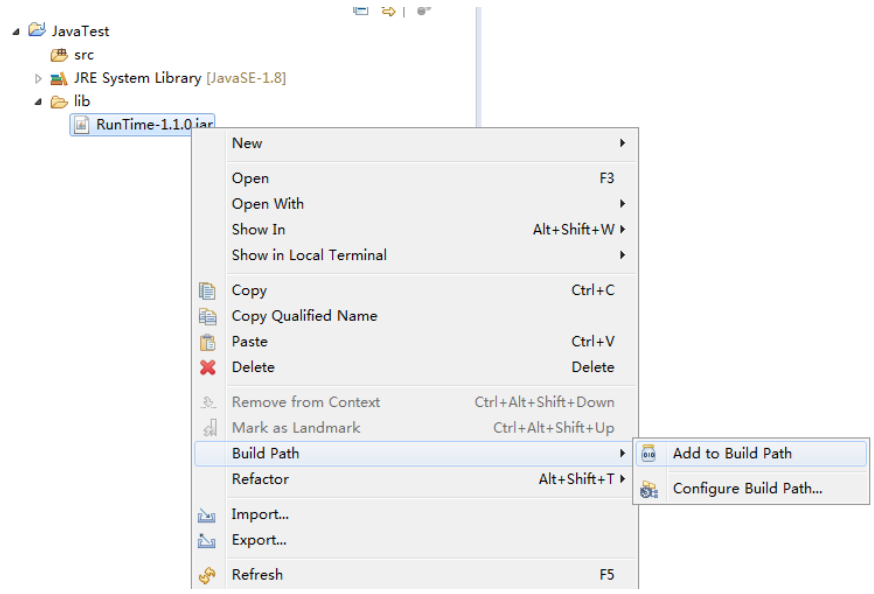
Figure 3-2 Decompressing the downloaded SDK



3. Configure the dependency.

   Create a folder named **lib** in the project directory, copy the **Runtime-1.1.3.jar** file in the SDK package to the **lib** folder, and add the JAR file as a dependency of the project, as shown in **Figure 3-3**.

Figure 3-3 Configuring the dependency

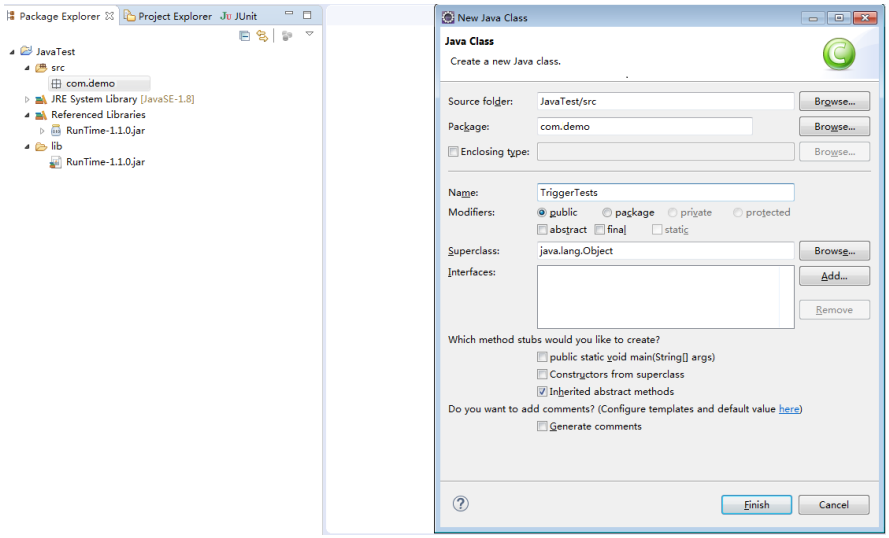


Step 2 Create a function.

1. Create a package named **com.demo**, and then create a class named **TriggerTests** under the package, as shown in **Figure 3-4**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 3-4 Creating a package named com.demo



2. Define the function handler in **TriggerTests.java**.

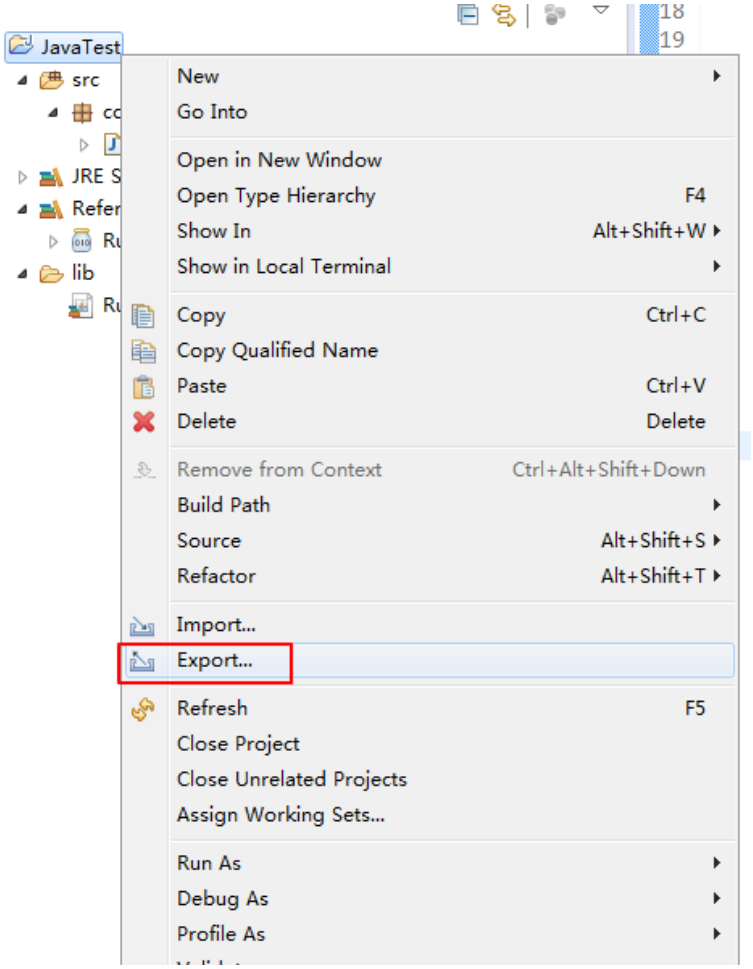
```
package com.demo;

import java.io.UnsupportedEncodingException;
import java.util.HashMap;
import java.util.Map;

import com.services.runtime.Context;
import com.services.runtime.entity.smn.SMNTriggerEvent;
import com.services.runtime.entity.timer.TimerTriggerEvent;

public class TriggerTests {

    public String smnTest(SMNTriggerEvent event, Context context){
        System.out.println(event);
        return "ok";
    }

    }

    public String timerTest(TimerTriggerEvent event, Context context){
        System.out.println(event);
        return "ok";
    }


}
```

Step 3 Package the project files.

1. Right-click the **JavaTest** project and choose **Export**, as shown in **Figure 3-5**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

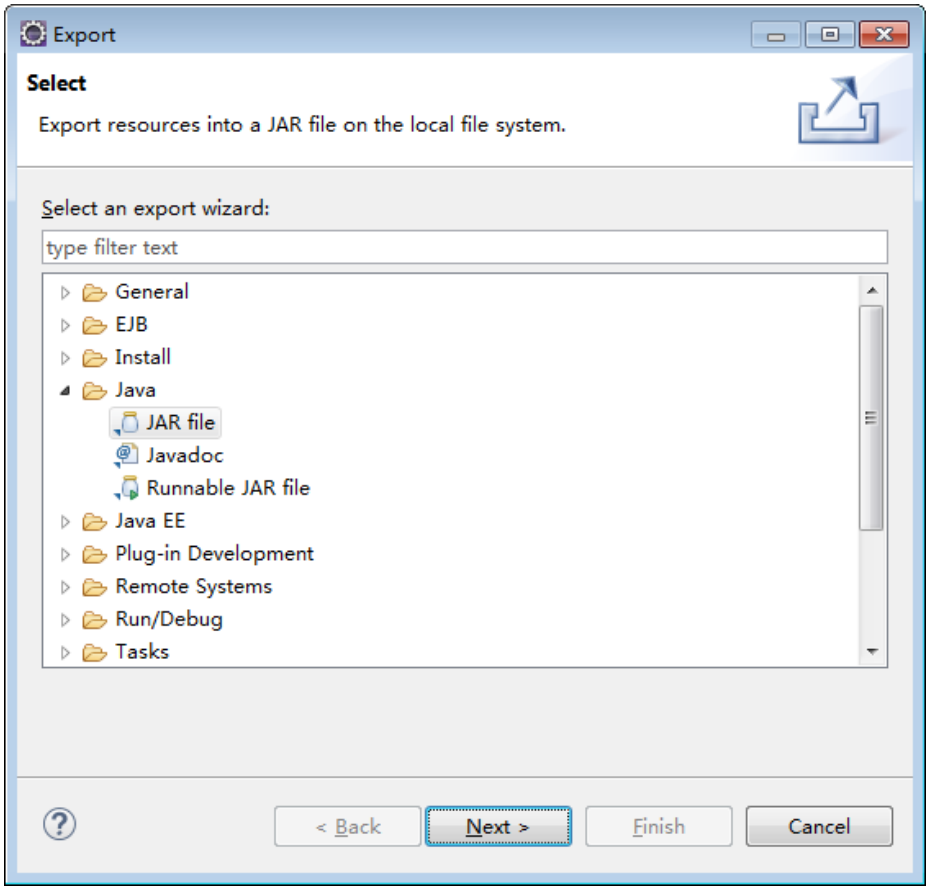
Figure 3-5 Packaging the project files



2. Export the project to a directory as a JAR file, as shown in **Figure 3-6** and **Figure 3-7**.

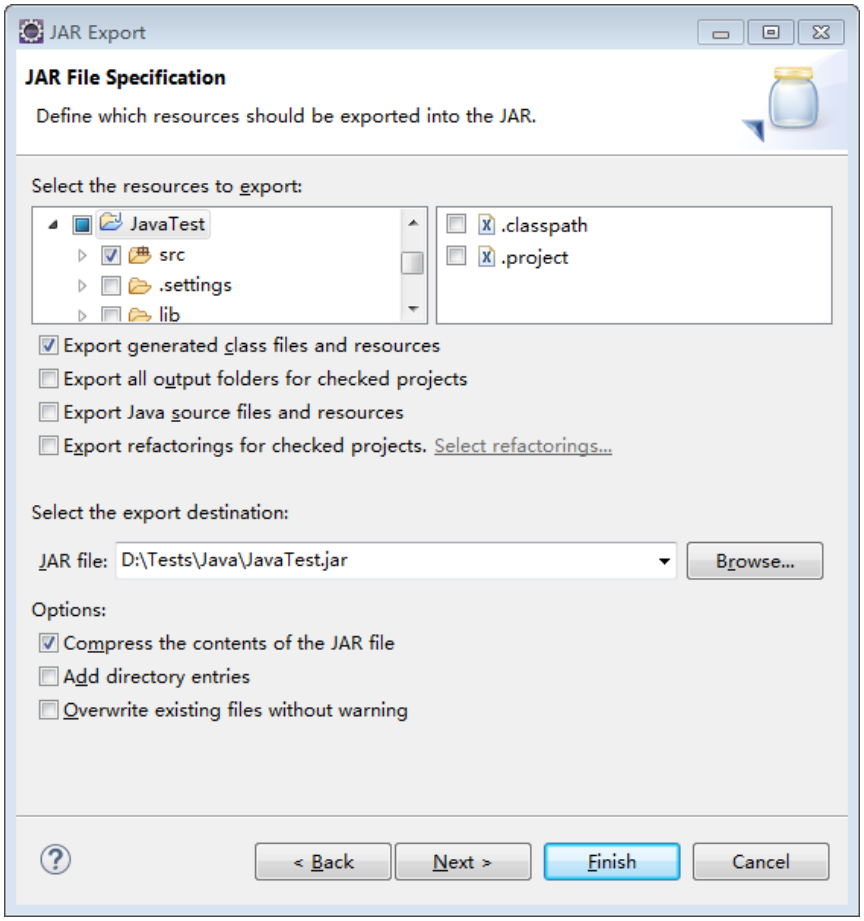
 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 3-6 Selecting a format



 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 3-7 Specifying the destination path



Step 4 Log in to the FunctionGraph console, create a Java function, and upload the code package.

Step 5 Test the function.

1. Create a timer trigger.
2. Create a test event.

   Select **Timer** and click **Save**.
3. Click **Test**.

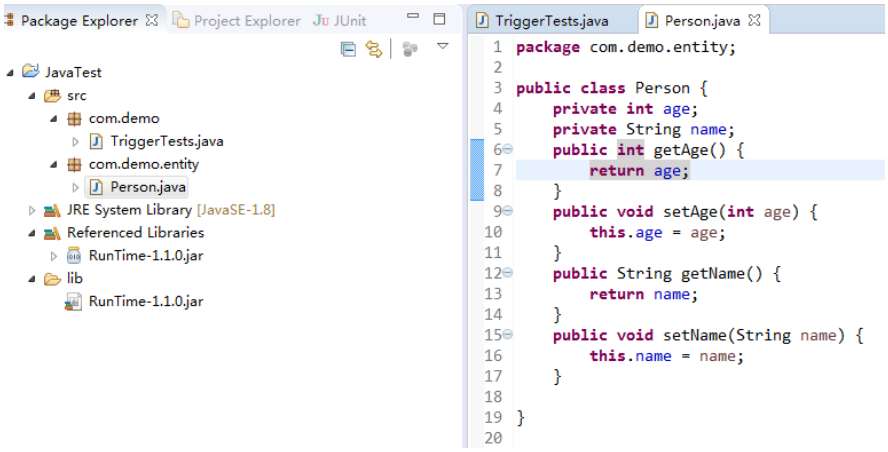
   The function execution result consists of three parts: function output (returned by **callback**), summary, and logs (output by using the **console.log** or **getLogger()** method).

Step 6 Use a user-defined parameter in the Java code.

Create a Person class in the project, as shown in **Figure 3-8**.

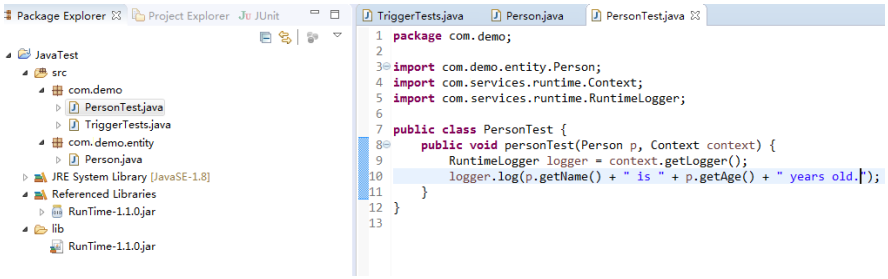
 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 3-8 Creating a Person class



Create a class named **PersonTest.java**, and add a handler function to the class, as shown in **Figure 3-9**.

Figure 3-9 Creating a class named PersonTest.java



After exporting the new package, upload it to the function, change the function handler to **com.demo.PersonTest.personTest**, and click **Save**.

In the **Configure Test Event** dialog box, select **blank-template**, enter a test event, and click **Save**.

Click **Create**, and then click **Test**.

----End

Execution Result

The execution result consists of the function output, summary, and log output.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 3-7 Description of the execution result

Parameter	Successful Execution	Failed Execution

Function Output	The defined function output information is returned.	A JSON file that contains **errorMessage** and **stackTrace** is returned. The format is as follows:
		`{`
		`  "errorMessage": "",`
		`  "stackTrace": []`
		`}`
		errorMessage: Error message returned by the runtime.
		stackTrace: Stack error information returned by the runtime.

Summary	**Request ID**, **Memory Configured**, **Execution Duration**, **Memory Used**, and **Billed Duration** are displayed.	**Request ID**, **Memory Configured**, **Execution Duration**, **Memory Used**, and **Billed Duration** are displayed.
Log Output	Function logs are printed. A maximum of 4 KB logs can be displayed.	Error information is printed. A maximum of 4 KB logs can be displayed.

3.4 Developing Functions in Go

Function Syntax

NOTE

You are advised to use Go 1.x.

Syntax for creating a handler function in Go:

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

- **Handler**: name of the function that FunctionGraph invokes to execute your code. The name must be consistent with that you define when creating a function, and must start with an uppercase letter.
- **payload**: event parameter defined for the function. The parameter is in JSON format.
- **ctx**: runtime information provided for executing the function. For details, see **SDK APIs**.

SDK APIs

- Context APIs

  The Go SDK provides multiple context APIs and a logging API. The download link of the Go SDK is provided in **Table 1-1**. **Table 3-8** describes the context APIs provided by FunctionGraph.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 3-8 Context methods

Method	Description
GetRequestID() string	Obtains a request ID.
GetRemainingTimeInMilliSeconds()	Obtains the remaining running time of a function.
GetAccessKey() string	Obtains the AK (valid for 24 hours) of an agency. If you use this method, you need to configure an agency for the function. **NOTE** FunctionGraph has stopped maintaining the getAccessKey API in the Runtime SDK. You cannot use this API to obtain a temporary AK.
GetSecretKey() string	Obtains the SK (valid for 24 hours) of an agency. If you use this method, you need to configure an agency for the function. **NOTE** FunctionGraph has stopped maintaining the getSecretKey API in the Runtime SDK. You cannot use this API to obtain a temporary SK.
getSecurityAccessKey()	Obtains the SecurityAccessKey (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getSecuritySecretKey()	Obtains the SecuritySecretKey (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getSecurityToken()	Obtains the SecurityToken (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
GetUserData(key string) string	Uses keys to obtain the values passed by environment variables.
GetFunctionName() string	Obtains the name of a function.
GetRunningTimeInSeconds() int	Obtains the timeout of a function.
GetVersion() string	Obtains the version of a function.
GetMemorySize() int	Obtains the allocated memory.
GetCPUNumber() int	Obtains CPU usage of a function.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method	Description
GetProjectID() string	Obtains a project ID.
GetPackage() string	Obtains a function group, that is, an app.
GetToken() string	Obtains the token (valid for 24 hours) of an agency. If you use this method, you need to configure an agency for the function.
GetLogger() context.RuntimeLogger	Obtains the **logger** method provided by the context. By default, information such as the time and request ID is output.

NOTICE

Results returned by using the **GetToken()**, **GetAccessKey()**, and **GetSecretKey()** methods contain sensitive information. Exercise caution when using these methods.

- Logging API

  Table 3-9 describes the logging API provided in the Go SDK.

  Table 3-9 Logging API

Method	Description
RuntimeLogger()	• Records user input logs by using the method **Logf(format string, args ...interface{})**. • This method outputs logs in the format of *Time-Request ID-Output*, for example, **2017-10-25T09:10:03.328Z 473d369d-101a-445e-a7a8-315cca788f86 test log output**.

Execution Result

The execution result consists of the function output, summary, and log output.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 3-10 Description of the execution result

Parameter	Successful Execution	Failed Execution

Function Output	The defined function output information is returned.	A JSON file that contains **errorMessage** and **errorType** is returned. The format is as follows:
		`{`
		`  "errorMessage": "",`
		`  "errorType":"",`
		`}`
		errorMessage: Error message returned by the runtime.
		errorType: Error type.

Summary	**Request ID**, **Memory Configured**, **Execution Duration**, **Memory Used**, and **Billed Duration** are displayed.	**Request ID**, **Memory Configured**, **Execution Duration**, **Memory Used**, and **Billed Duration** are displayed.
Log Output	Function logs are printed. A maximum of 4 KB logs can be displayed.	Error information is printed. A maximum of 4 KB logs can be displayed.

3.5 Developing Functions in C#

3.5.1 C# Function Development

Function Syntax

FunctionGraph supports C# (.NET Core 2.1), C# (.NET Core 3.1), C# (.NET Core 6.0), and C# (.NET Core 8.0).

Scope Return parameter Function name (User-defined parameter, Context)

- *Scope*: It must be defined as **public** for the function that FunctionGraph invokes to execute your code.
- *Return parameter*: user-defined output, which is converted into a character string and returned as an HTTP response.
- *Function name*: user-defined function name. The name must be consistent with that you define when creating a function.
- Event: event parameter defined for the function.
- **context**: runtime information provided for executing the function. For details, see the description of SDK APIs.

  The **HC.Serverless.Function.Common** library needs to be referenced when you deploy a project in FunctionGraph. For details about the **IFunctionContext** object, see the context description.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

When creating a C# function, you need to define a method as the handler of the function. The method can access the function by using specified IFunctionContext parameters. Example:

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

Function Handler

ASSEMBLY::NAMESPACE.CLASSNAME::METHODNAME

- **ASSEMBLY**: name of the .NET assembly file for your application, for example, **HelloCsharp**.
- **NAMESPACE** and **CLASSNAME**: names of the namespace and class to which the handler function belongs.
- **METHODNAME**: name of the handler function. Example:

  Set the handler to **HelloCsharp::Example.Hello::Handler** when you create a function.

SDK APIs

- Context APIs

  Table 3-11 describes the provided context attributes.

  Table 3-11 Context objects

Attribute	Description
String RequestId	Request ID.
String ProjectId	Project Id
String PackageName	Name of the group to which the function belongs.
String FunctionName	Function name.
String FunctionVersion	Function version.
Int MemoryLimitInMb	Allocated memory.
Int CpuNumber	Obtains CPU usage of a function.
String Accesskey	Obtains the AK (valid for 24 hours) of an agency. If you use this method, you need to configure an agency for the function. **NOTE** FunctionGraph has stopped maintaining the String AccessKey API in the Runtime SDK. You cannot use this API to obtain a temporary AK.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Attribute	Description
String Secretkey	Obtains the SK (valid for 24 hours) of an agency. If you use this method, you need to configure an agency for the function. **NOTE** FunctionGraph has stopped maintaining the String SecretKey API in the Runtime SDK. You cannot use this API to obtain a temporary SK.
String SecurityAccessKey	Obtains the SecurityAccessKey (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
String SecuritySecretKey	Obtains the SecuritySecretKey (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
String SecurityToken	Obtains the SecurityToken (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
String Token	Obtains the token (valid for 24 hours) of an agency. If you use this method, you need to configure an agency for the function.
Int RemainingTimeInMilli-Seconds	Remaining running time of a function.
String GetUserData(string key,string defvalue=" ")	Uses keys to obtain the values passed by environment variables.

- Logging APIs

  The following table describes the logging APIs provided in the C# SDK.

  Table 3-12 Logging APIs

Method	Description

Log(string message)	Creates a logger object by using context.
	var logger = context.Logger;
	logger.Log("hello CSharp runtime test(v1.0.2)");

Logf(string format, args ...interface{})	Creates a logger object by using context.
	var logger = context.Logger;
	var version = "v1.0.2"
	logger.Logf("hello CSharp runtime test({0})", version);

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Execution Result

The execution result consists of the function output, summary, and log output.

Table 3-13 Description of the execution result

Parameter	Successful Execution	Failed Execution
Function Output	The defined function output information is returned.	A JSON file that contains **errorMessage** and **errorType** is returned. The format is as follows: `{ "errorMessage": "", "errorType": "" }` **errorMessage**: Error message returned by the runtime. **errorType**: Error type.
Summary	**Request ID**, **Memory Configured**, **Execution Duration**, **Memory Used**, and **Billed Duration** are displayed.	**Request ID**, **Memory Configured**, **Execution Duration**, **Memory Used**, and **Billed Duration** are displayed.
Log Output	Function logs are printed. A maximum of 4 KB logs can be displayed.	Error information is printed. A maximum of 4 KB logs can be displayed.

3.5.2 JSON Serialization and Deserialization

3.5.2.1 Using .NET Core CLI

C# supports JSON serialization and deserialization interfaces and provides the **HC.Serverless.Function.Common.JsonSerializer.dll** file.

The interfaces are as follows:

T Deserialize<T>(Stream ins): Deserializes data into objects of function programs.

Stream Serialize<T>(T value): Serializes data to the returned response payload.

The following shows how to create a project named **test** based on .NET Core 2.1. The procedure is similar for .NET Core 3.1 and .NET Core 6.0. The .NET SDK 2.1 has been installed in the execution environment.

Creating a Project

1. Run the following command to create the **/tmp/csharp/projects /tmp/csharp/release** directory:
   ```
   mkdir -p  /tmp/csharp/projects;mkdir -p  /tmp/csharp/release
   ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

2. Run the following command to go to the **/tmp/csharp/projects/** directory:
```
cd /tmp/csharp/projects/
```
3. Create the project file **test.csproj** with the following content:
```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>netcoreapp2.1</TargetFramework>
    <RootNamespace>test</RootNamespace>
    <AssemblyName>test</AssemblyName>
  </PropertyGroup>
  <ItemGroup>
    <Reference Include="HC.Serverless.Function.Common">
      <HintPath>HC.Serverless.Function.Common.dll</HintPath>
    </Reference>
    <Reference Include="HC.Serverless.Function.Common.JsonSerializer">
      <HintPath> HC.Serverless.Function.Common.JsonSerializer.dll</HintPath>
    </Reference>
  </ItemGroup>
</Project>
```

Generating a Code Library

1. Upload the **.dll file package**.

   Upload the **HC.Serverless.Function.Common.dll** and **HC.Serverless.Function.Common.JsonSerializer.dll** files in the package to the **/tmp/csharp/projects/** directory.
2. Create the **Class1.cs** file in the **/tmp/csharp/projects/** directory. The code is as follows:
```
using HC.Serverless.Function.Common;
using System;
using System.IO;

namespace test
{
    public class Class1
    {
        public Stream ContextHandlerSerializer(Stream input, IFunctionContext context)
        {
            var logger = context.Logger;
            logger.Logf("CSharp runtime test(v1.0.2)");
            JsonSerializer test = new JsonSerializer();
            TestJson Testjson = test.Deserialize<TestJson>(input);
            if (Testjson != null)
            {
                logger.Logf("json Deserialize KetTest={0}", Testjson.KetTest);

            }
            else
            {
                return null;
            }

          return test.Serialize<TestJson>(Testjson);
        }

        public class TestJson
        {
            public string KetTest { get; set; }//Define the attribute of the serialization class as KetTest.

        }
    }
}
```
3. Run the following command to generate a code library:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
/home/tools/dotnetcore-sdk/dotnet-sdk-2.1.302-linux-x64/dotnet build /tmp/csharp/
projects/test.csproj -c Release -o /tmp/csharp/release
```

NOTE

.NET directory: /home/tools/dotnetcore-sdk/dotnet-sdk-2.1.302-linux-x64/dotnet

4. Run the following command to go to the **/tmp/csharp/release** directory:
   ```
   cd /tmp/csharp/release
   ```
5. View the compiled .dll files in the **/tmp/csharp/release** directory.
   ```
   -rw-r--r-- 1 root root   5120 Jan 21 16:40 HC.Serverless.Function.Common.JsonSerializer.dll
   -rw-r--r-- 1 root root   5120 Jan 21 16:40 HC.Serverless.Function.Common.dll
   -rw-r--r-- 1 root root    232 Jan 21 17:10 test.pdb
   -rw-r--r-- 1 root root   3584 Jan 21 17:10 test.dll
   -rw-r--r-- 1 root root   1659 Jan 21 17:10 test.deps.json
   ```
6. Create the **test.runtimeconfig.json** file in the **/tmp/csharp/release** directory. The file content is as follows:
   ```
   {
     "runtimeOptions": {
       "framework": {
         "name": "Microsoft.NETCore.App",
         "version": "2.1.0"
       }
     }
   }
   ```

   NOTE

   - The name of the ***.runtimeconfig.json** file is the name of an assembly.
   - The **Version** parameter in the file indicates the version number of the target framework. If the framework is .NET Core 2.0, enter **2.0.0**. If the framework is .NET Core 2.1, enter **2.1.0**.
   - For .NET Core 2.0, check whether **Newtonsoft.Json** is referenced in the ***.deps.json** file. If **Newtonsoft.Json** is not referenced, reference it manually.
     1. Reference the following content in **targets**:
        ```
        "Newtonsoft.Json/9.0.0.0": {
              "runtime": {
                "Newtonsoft.Json.dll": {
                  "assemblyVersion": "9.0.0.0",
                  "fileVersion": "9.0.1.19813"
                }
              }
            }
        ```
     2. Reference the following content in **libraries**:
        ```
        "Newtonsoft.Json/9.0.0.0": {
            "type": "reference",
            "serviceable": false,
            "sha512": ""
          }
        ```
7. Run the following command to package the **test.zip** file in the **/tmp/csharp/release** directory:
   ```
   zip -r test.zip ./*
   ```

3.5.2.2 Using Visual Studio

C# supports JSON serialization and deserialization interfaces and provides the **HC.Serverless.Function.Common.JsonSerializer.dll** file.

The interfaces are as follows:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

T Deserialize<T>(Stream ins): Deserializes data into objects of function programs.

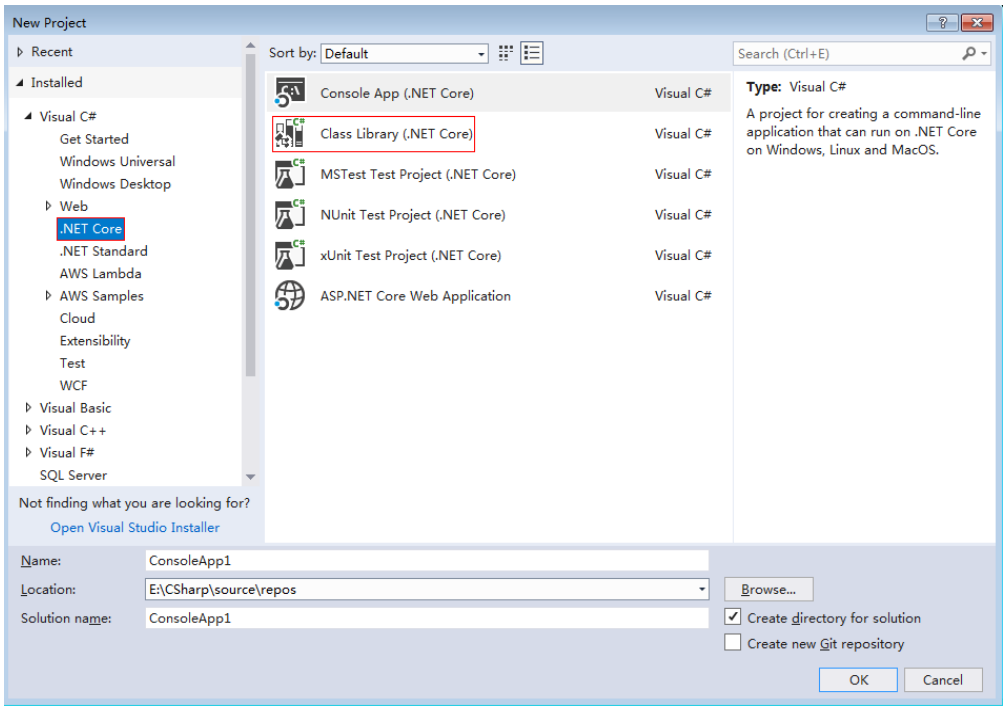
Stream Serialize<T>(T value): Serializes data to the returned response payload.

The following describes how to create a project named **test** based on .NET Core 2.0 by using Visual Studio 2017. The procedure is similar for .NET Core 2.1 and .NET Core 3.1.

Creating a Project

1. On the toolbar, choose **File** > **New** > **Project**, select **.NET Core**, select **Class Library (.NET Core)**, and change the project name to **test**, as shown in **Figure 3-10**.

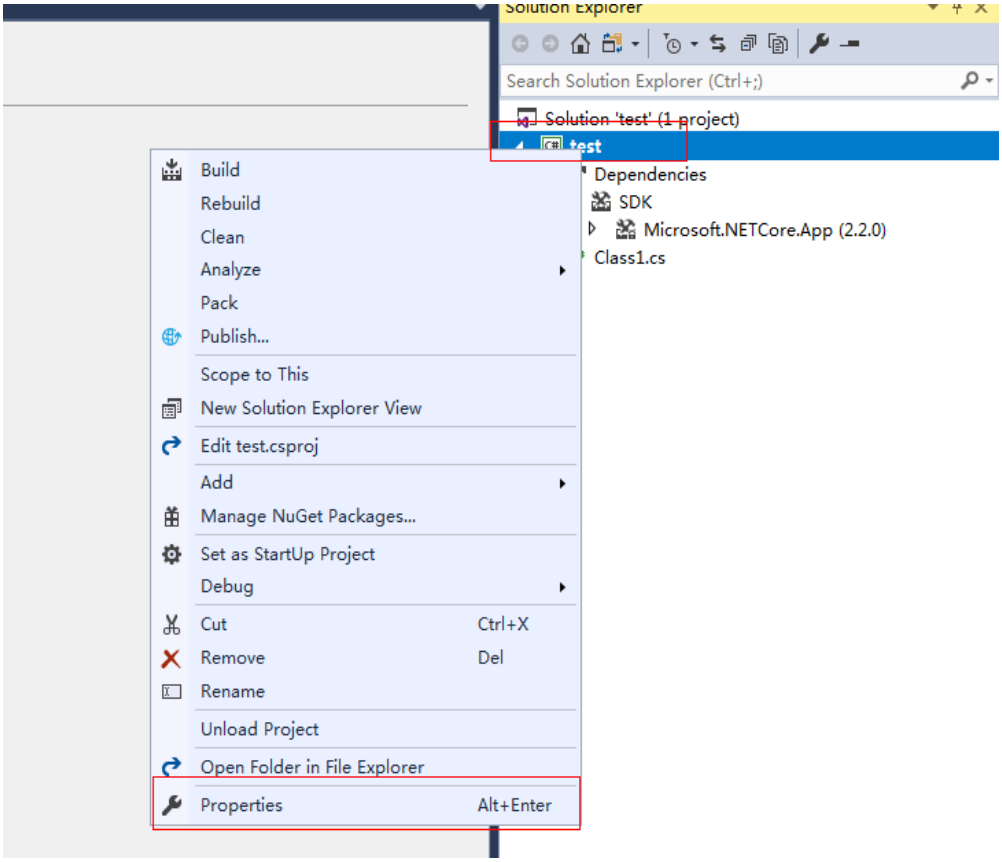
   Figure 3-10 Creating a project

2. Select the **test** project in the navigation pane, right-click, and then choose **Properties**, as shown in **Figure 3-11**.

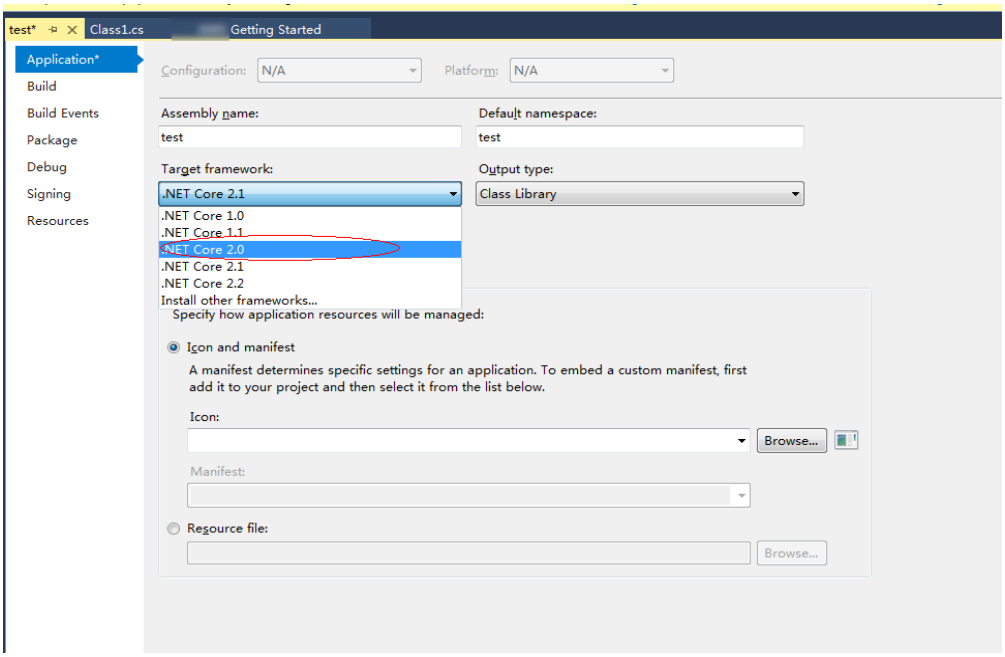
 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 3-11 Properties



3. Choose **Application** and then set **Target framework** to **.NET Core 2.0**, as shown in **Figure 3-12**.

Figure 3-12 Selecting a target framework

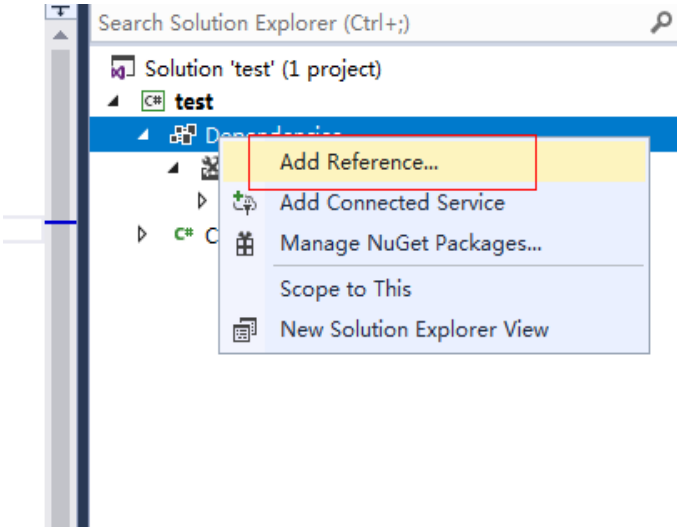


 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Adding a Reference

1. Select the **test** project in **Search Solution Explorer**, right-click, and then choose **Add Reference** to reference the downloaded **.dll file package**, as shown in **Figure 3-13**.

   Figure 3-13 Adding a reference

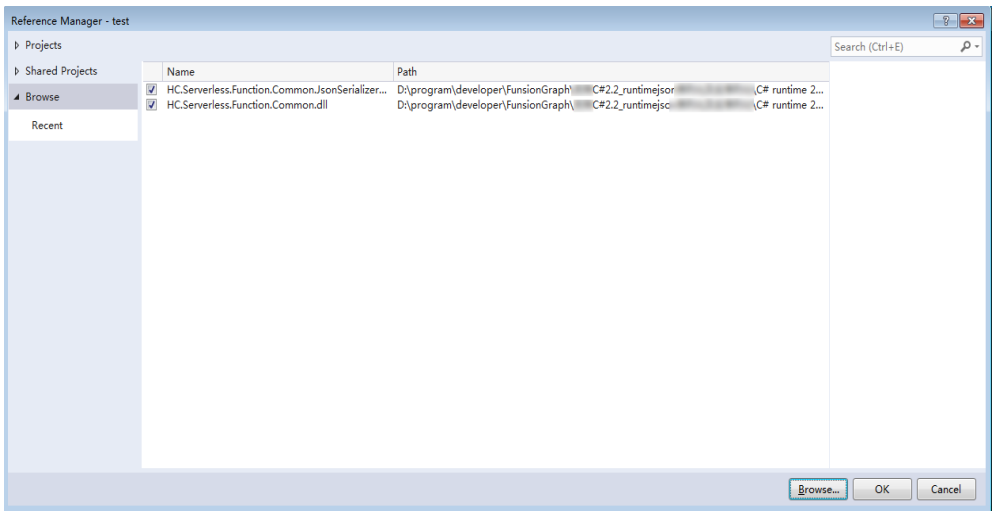
   

   NOTE

   Store **HC.Serverless.Function.Common.dll** and **HC.Serverless.Function.Common.JsonSerializer.dll** in a lib file.

2. Choose **Browse**, click **Browse(B)**, reference the **HC.Serverless.Function.Common.dll** and **HC.Serverless.Function.Common.JsonSerializer.dll** files, and click **OK**, as shown in **Figure 3-14**.

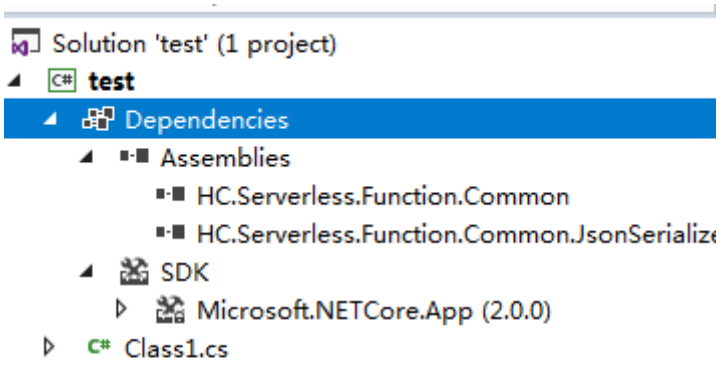
   Figure 3-14 Referencing files

3. View the references, as shown in **Figure 3-15**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 3-15 References



Packing Code

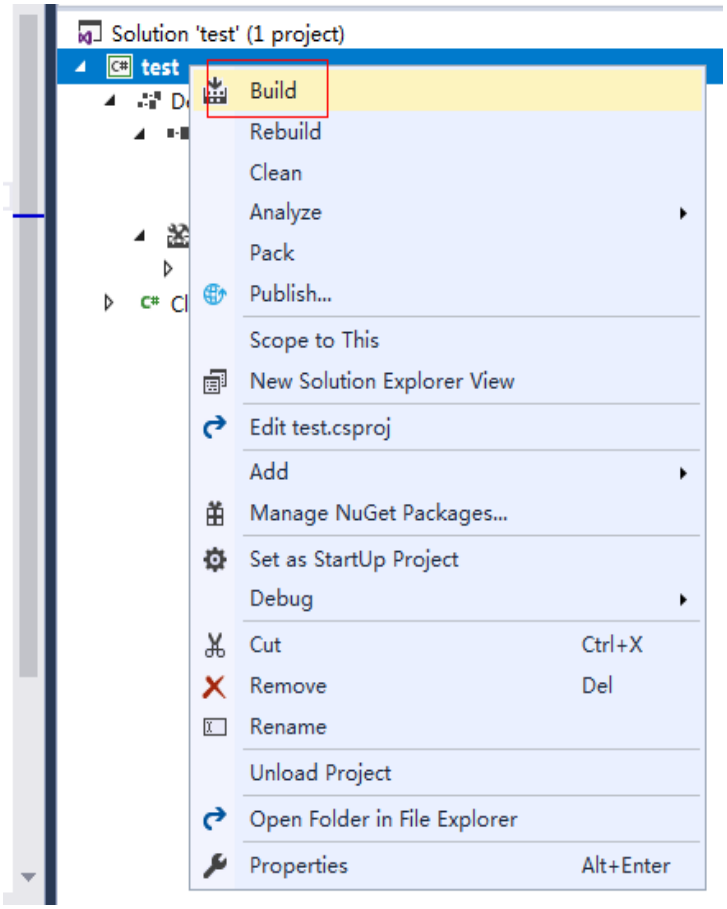
Sample code:

```
using HC.Serverless.Function.Common;
using System;
using System.IO;

namespace test
{
    public class Class1
    {
        public Stream ContextHandlerSerializer(Stream input, IFunctionContext context)
        {
            var logger = context.Logger;
            logger.Logf("CSharp runtime test(v1.0.2)");
            JsonSerializer test = new JsonSerializer();
            TestJson Testjson = test.Deserialize<TestJson>(input);
            if (Testjson != null)
            {
                logger.Logf("json Deserialize KetTest={0}", Testjson.KetTest);
            }

            return test.Serialize<TestJson>(Testjson);
        }

        public class TestJson
        {
            public string KetTest { get; set; }//Define the attribute of the serialization class as KetTest.

        }
    }
}
```

1. Right-click the **test** project and choose **Build**, as shown in **Figure 3-16**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 3-16 Build



2. Copy the path **C:\Users\xxx\source\repos\test\test\bin\Release\netcoreapp2.0** of the .dll files, as shown in **Figure 3-17**.

Figure 3-17 Path of .dll files

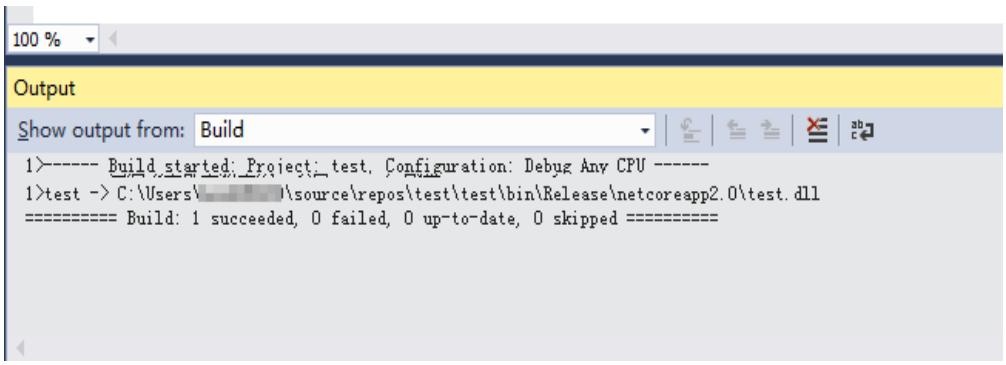
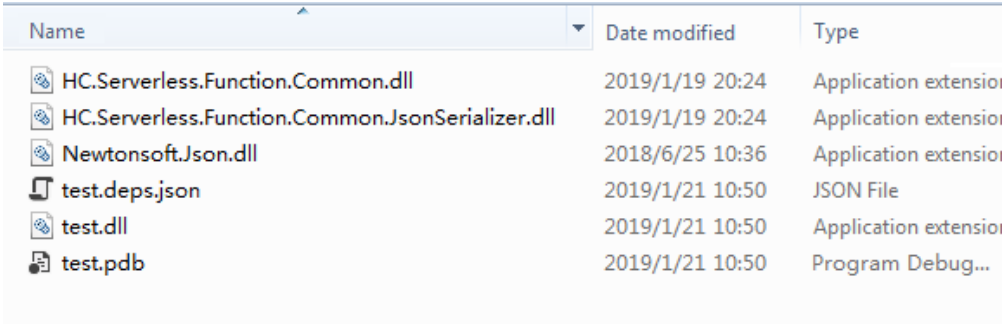


Figure 3-18 shows the files in the path.

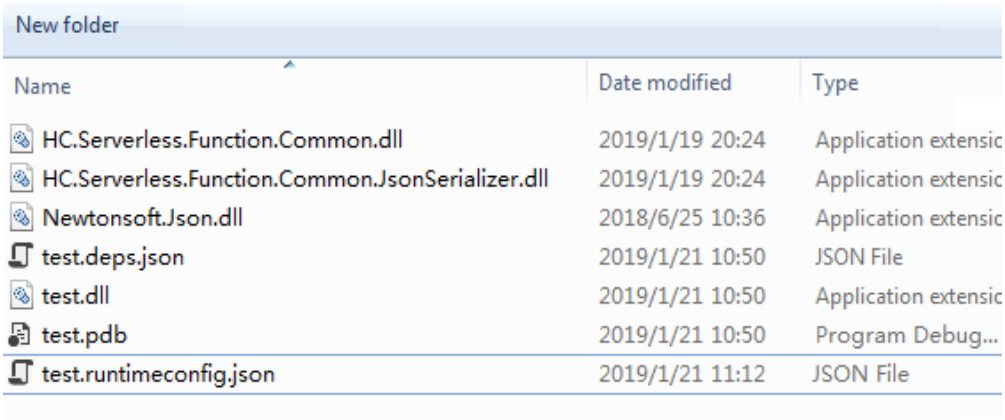
 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 3-18 Files



3. Create a **test.runtimeconfig.json** file in the path, as shown in **Figure 3-19**.

Figure 3-19 New file



Add the following content in the file:

```
{

"runtimeOptions": {

"framework": {

"name": "Microsoft.NETCore.App",

"version": "2.0.0"
      }
   }
}
```

NOTE

- The name of the ***.runtimeconfig.json** file is the name of an assembly.
- The **Version** parameter in the file indicates the version number of the target framework. If the framework is .NET Core 2.0, enter **2.0.0**. If the framework is .NET Core 2.1, enter **2.1.0**.

4. Compress the file into **netcoreapp2.0.zip**.

NOTE

The package name can be customized but must be ended with .zip.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

3.6 Developing Functions in PHP

Function Syntax

Use the following syntax when creating a handler function in PHP:

function handler($event, $context)

- **$handler**: name of the function that FunctionGraph invokes to execute your code. The name must be consistent with that you define when creating a function.
- **$event**: event parameter defined for the function. The parameter is in JSON format.
- **$context**: runtime information provided for executing the function. For details, see **SDK APIs**.
- Function handler: **index.handler**.
- The function handler is in the format of *[File name].[Function name]*. For example, if you set the handler to **index.handler** in your function, FunctionGraph will load the handler function defined in the **index.php** file.

PHP Initializer

FunctionGraph supports the following PHP runtime:

- PHP 7.3
- PHP 8.3

Initializer syntax:

[File name].[Initializer name]

For example, if the initializer is named **main.my_initializer**, FunctionGraph loads the my_initializer function defined in the **main.php** file.

To use PHP to build initialization logic, define a PHP function as the initializer. The following is a simple initializer:

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

- Function name

  The function name **my_initializer** must be the initializer function name specified for a function.

  For example, if the initializer is named **main.my_initializer**, FunctionGraph loads the my_initializer function defined in the **main.php** file.

- context

  The **context** parameter contains the runtime information about a function. For example, request ID, temporary AK, and function metadata.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

SDK APIs

The following table describes the context methods provided by FunctionGraph.

Table 3-14 Context methods

Method	Description
getRequestID()	Obtains a request ID.
getRemainingTimeIn-MilliSeconds ()	Obtains the remaining running time of a function.
getAccessKey()	Obtains the AK (valid for 24 hours) of an agency. If you use this method, you need to configure an agency for the function. **NOTE** FunctionGraph has stopped maintaining the getAccessKey API in the Runtime SDK. You cannot use this API to obtain a temporary AK.
getSecretKey()	Obtains the SK (valid for 24 hours) of an agency. If you use this method, you need to configure an agency for the function. **NOTE** FunctionGraph has stopped maintaining the getSecretKey API in the Runtime SDK. You cannot use this API to obtain a temporary SK.
getSecurityAccessKey()	Obtains the SecurityAccessKey (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getSecuritySecretKey()	Obtains the SecuritySecretKey (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getSecurityToken()	Obtains the SecurityToken (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.
getUserData(string key)	Uses keys to obtain the values passed by environment variables.
getFunctionName()	Obtains the name of a function.
getRunningTimeInSec-onds ()	Obtains the timeout of a function.
getVersion()	Obtains the version of a function.
getMemorySize()	Obtains the allocated memory.
getCPUNumber()	Obtains CPU usage of a function.
getProjectID()	Obtains a project ID.
getPackage()	Obtains a function group, that is, an app.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Method	Description
getToken()	Obtains the token (valid for 24 hours) of an agency. If you use this method, you need to configure an agency for the function.
getLogger()	Obtains the **logger** method provided by the context and returns a log output class. Logs are output in the format of *Time-Request ID-Content* by using the **info** method. For example, use the **info** method to output logs: logg = context.getLogger()$ $logg->info("hello")

NOTE

Results returned by using the **getToken()**, **getAccessKey()**, and **getSecretKey()** methods contain sensitive information. Exercise caution when using these methods.

Execution Result

The execution result consists of the function output, summary, and log output.

Table 3-15 Description of the execution result

Parameter	Successful Execution	Failed Execution

Function Output	The defined function output information is returned.	A JSON file that contains **errorMessage**, **errorType**, and **stackTrace** is returned. The format is as follows:
		{
		"errorMessage": "",
		"errorType": "",
		"stackTrace": {}
		}
		errorMessage: Error message returned by the runtime.
		errorType: Error type.
		stackTrace: Stack error information returned by the runtime.

Summary	**Request ID**, **Memory Configured**, **Execution Duration**, **Memory Used**, and **Billed Duration** are displayed.	**Request ID**, **Memory Configured**, **Execution Duration**, **Memory Used**, and **Billed Duration** are displayed.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Successful Execution	Failed Execution
Log Output	Function logs are printed. A maximum of 4 KB logs can be displayed.	Error information is printed. A maximum of 4 KB logs can be displayed.

3.7 Developing Functions in Cangjie

3.7.1 Developing Functions in Cangjie (Using Visual Studio Code)

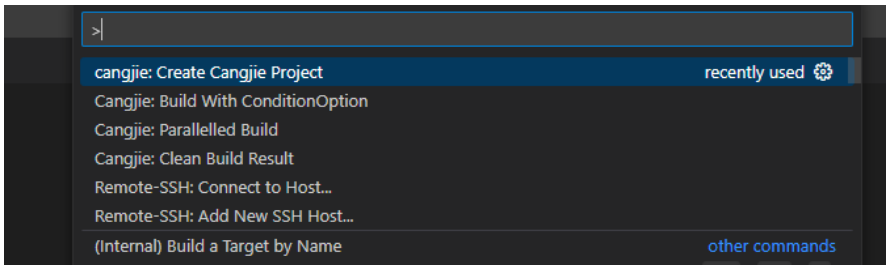
NOTE

- Cangjie functions can be developed **only in the Linux environment**.
- You can compile a function only via uploading a ZIP package with all required dependencies.

Step 1 Create a function project.

1. Create a Cangjie project.

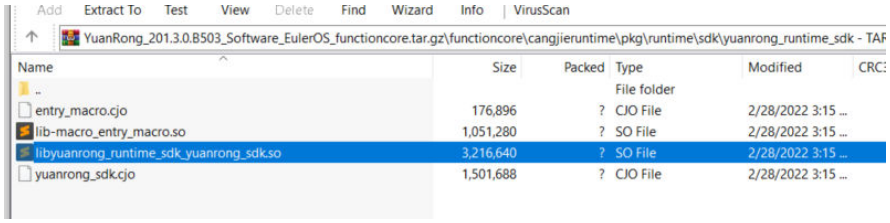
   Figure 3-20 Creating a Cangjie project

2. Add dependencies to the project.

   Download **CangjieRuntime SDK** to the local development environment, decompress it, and copy **libyuanrong_runtime_sdk_yuanrong_sdk.so** to the file directory.

   Figure 3-21 Adding dependencies to the project

3. Configure dependencies.

   In the **module.json** file of the user function, add the downloaded SDK on which the file depends.

   ```
   {
     "cjc_version": "0.xx.xx",
   ```

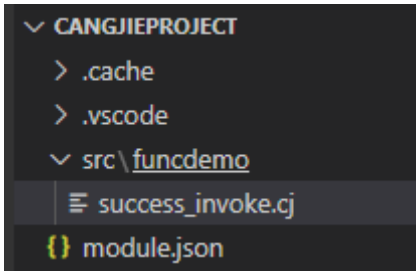
 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 "organization": "yuanrong",
 "name": "user_func_test",
 "description": "user func demo",
 "version": "1.0.0",
 "build_dir": "",
       "requires": {},
 "dev_requires": {},
 "package_requires": {
   "path_option":["./yuanrong_runtime_sdk"]
 },
 "foreign_requires": {},
 "output_type": "dynamic",
 "command_option": "",
 "condition_option": {
              "bind_now": "--link-options \"-z now\""
       },
 "link_option": "",
 "cross_compile_configuration": {},
 "scripts": {},
 "package_configuration": {}
}
```

Step 2 Create a function.

1. Create the **funcdemo** folder and create the **success_invoke** class in the package.

   Figure 3-22 Creating the success_invoke class

2. Define the function handler in **success_invoke.cj**.

```
package funcdemo

from yuanrong_runtime_sdk import yuanrong_sdk.Context
from yuanrong_runtime_sdk import yuanrong_sdk.APIGTriggerEvent
from yuanrong_runtime_sdk import yuanrong_sdk.APIGTriggerResponse
from yuanrong_runtime_sdk import yuanrong_sdk.FuncRegister
from yuanrong_runtime_sdk import entry_macro.Entry
from encoding import json.*
from std import collection.*
from std import os.*
from serialization import serialization.*

@Entry
public func handleRequest(args: APIGTriggerEvent, context: Context): APIGTriggerResponse {
    var contextRunningTime: String = context.getRunningTimeInSeconds().toString()
    println("args is ${args.getBody()}")
    var resp = APIGTriggerResponse()
    resp.setBody(args.getBody())
    resp.setStatusCode(200)
    resp
}
```

Step 3 Package the project files.

Use **cjpm build** to build a project.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Step 4 Create a Cangjie function by uploading the code package in ZIP format. For details, see section "Creating a Function from Scratch" in the *FunctionGraph User Guide*.

Compress the generated **.so** file and upload it to the Yuanrong environment.

```
[root@dggphis36582 user_func_test]# ls
incremental-cache.json  libuser_func_test_success.so  success.cjo
[root@dggphis36582 user_func_test]# zip cangjie-demo.zip libuser_func_test_success.so
  adding: libuser_func_test_success.so (deflated 74%)
[root@dggphis36582 user_func_test]# ls
cangjie-demo.zip  incremental-cache.json  libuser_func_test_success.so  success.cjo
[root@dggphis36582 user_func_test]#
```

----End

Testing Function Invocation

Step 1 On the function details page, choose **Code** > **Configure Test Event**. In the dialog box that is displayed, click **Create new test event** > **Blank Template**.

The contents of the test cases are as follows:

```
{
   "isBase64Encoded": false,
   "httpMethod": "POST",
   "path": "",
   "body": "test_cangjie"
}
```

NOTE

Parameters must be transferred based on the **APIGTriggerEvent** class defined in the Yuanrong SDK. Otherwise, serialization and deserialization exceptions may occur.

```
public open class APIGTriggerEvent {
    private var isBase64Encoded: Bool = false;
    private var httpMethod: String = "POST";
    private var path: String = "";
    private var body: String = "";
}
```

Step 2 After the test event is configured, click **Test**.

----End

3.7.2 Developing an Event Function

You can compile a function only via uploading a ZIP package with all required dependencies.

Preparation

Obtain the Cangjie demo project package from technical support.

Compiling and Packaging

Run the **cjpm build** command to generate a **libxxxx.so** file in the **build** directory. You need to compress the **.so** file into a .zip file, as shown in **Figure 3-23**.

Figure 3-23 Packaging the file into a .zip file

```
[root@pekphis113029 user_func_test]# vim code.zip

" zip.vim version v28
" Browsing zipfile /home/lzq/develop-run/runtime-cangjie/user_func_test/build/release/user_func_test/code.zip
" Select a file with cursor and press ENTER

libuser_func_test_success.so
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Creating a Function

Step 1 Log in to the FunctionGraph console and click **Create Function** in the upper right corner. The **Create Function** page is displayed.

Step 2 Select **Create from scratch** and set the following parameters:

- **Function Type**: Select **Event Function**.
- **Region**: Select a region.
- **Function Name**: Enter a custom name.
- **Enterprise Project**: Select an enterprise project as required.
- **Agency**: Select an existing agency. For details about how to create an agency, see section "Creating an Agency" in the *FunctionGraph User Guide*.
- **Runtime**: select **Cangjie 1.0**.

Step 3 Click **Create Function**.

----End

Configuring Functions

Step 1 Return to the FunctionGraph console, choose **Functions** > **Function List** on the left, locate the created function, and click its name. The function details page is displayed.

Step 2 On the **Code** tab page, choose **Upload** > **Local ZIP** > **Select File** on the right to upload the prepared demo package. After the upload is successful, click **OK**.

Step 3 Choose **Configuration** > **Basic Settings**. Set the handler to the name of the .so file in **Compiling and Packaging**, for example, **libuser_func_test_success.so**, and click **Save**.

----End

Testing Function Invocation

Step 1 On the function details page, choose **Code** > **Configure Test Event**. In the dialog box that is displayed, click **Create new test event** > **Blank Template**.

The contents of the test cases are as follows:

```
{
    "isBase64Encoded": false,
    "httpMethod": "POST",
    "path": "",
    "body": "test_cangjie"
}
```

NOTE

Parameters must be transferred based on the **APIGTriggerEvent** class defined in the Yuanrong SDK. Otherwise, serialization and deserialization exceptions may occur.

```
public open class APIGTriggerEvent {
    private var isBase64Encoded: Bool = false;
    private var httpMethod: String = "POST";
    private var path: String = "";
    private var body: String = "";
}
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Step 2 After the test event is configured, click **Test**.

----End

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4 Developing Functions with Plug-ins

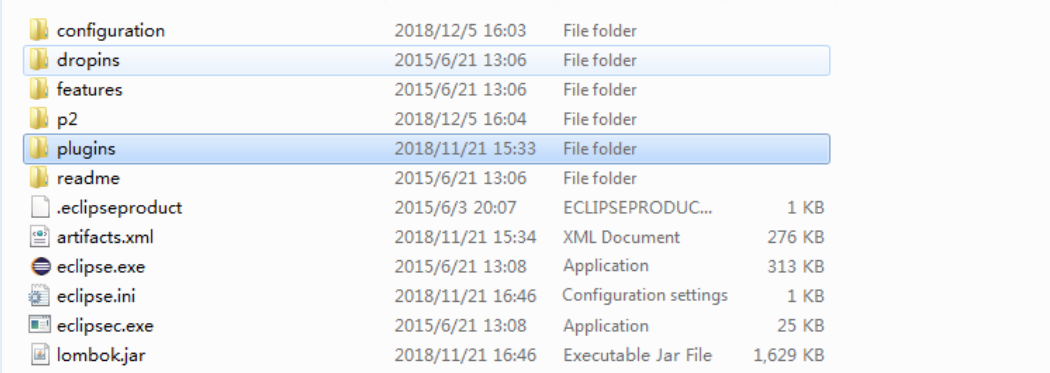
4.1 Eclipse Plug-in

Currently, FunctionGraph does not provide Java function templates and only allows you to upload Java function packages through OBS. With the Eclipse plug-in, you can quickly create Java function templates, package function project files, upload function packages to OBS, and deploy functions.

Step 1 Download the **Eclipse plug-in**.

Step 2 Put the Eclipse plug-in package (.jar or .zip) in the **plugins** folder under the Eclipse installation directory. Then restart Eclipse. **Figure 4-1** shows the Eclipse installation directory.

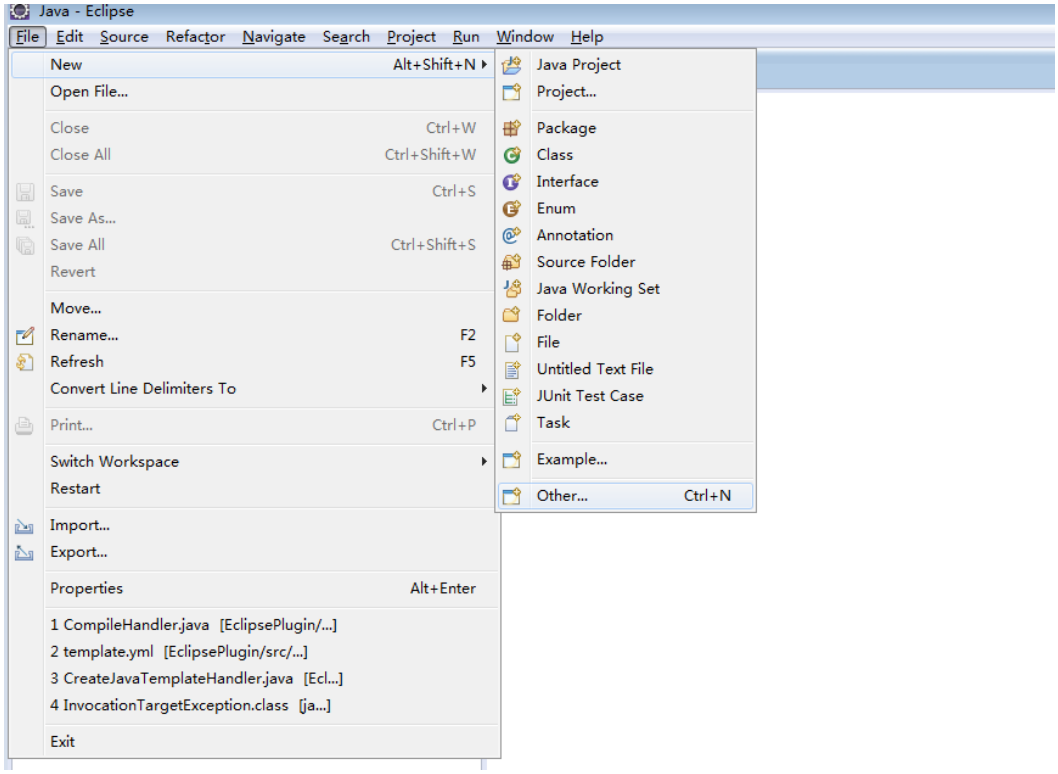
Figure 4-1 Installing the Eclipse plug-in



Step 3 Open Eclipse and choose **File** > **New** > **Other**, as shown in **Figure 4-2**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

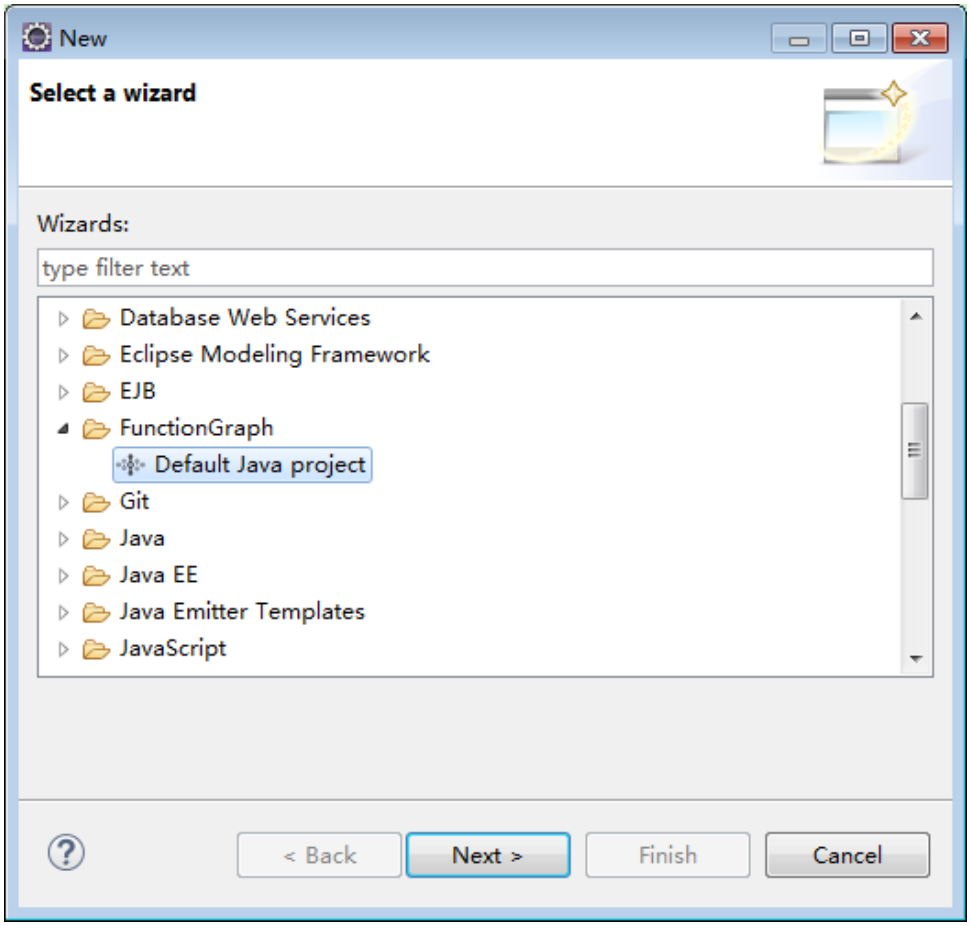
Figure 4-2 Creating a template



Step 4 Choose **FunctionGraph** > **Default Java project**, as shown in **Figure 4-3**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

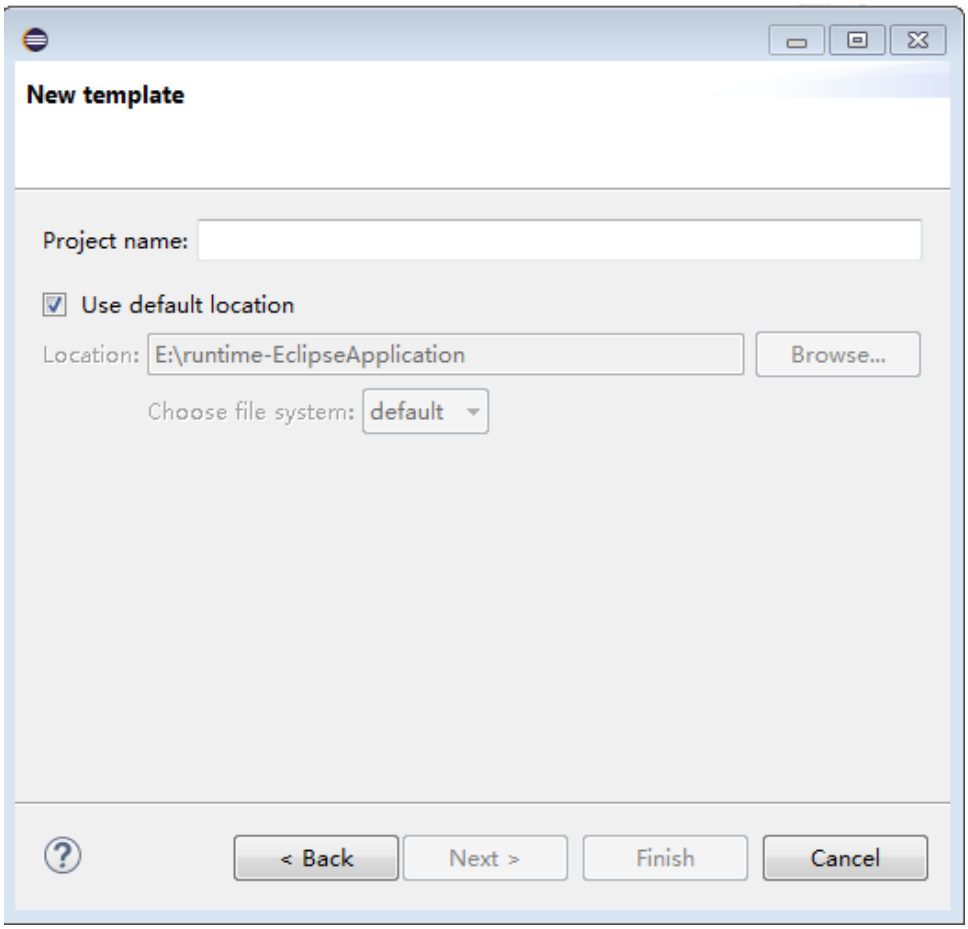
Figure 4-3 Selecting the default Java template



Step 5 Enter a project name, specify a project directory (or use the default directory), and click **Finish**, as shown in **Figure 4-4**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 4-4 Setting the project name and directory



----End

4.2 PyCharm Plug-in

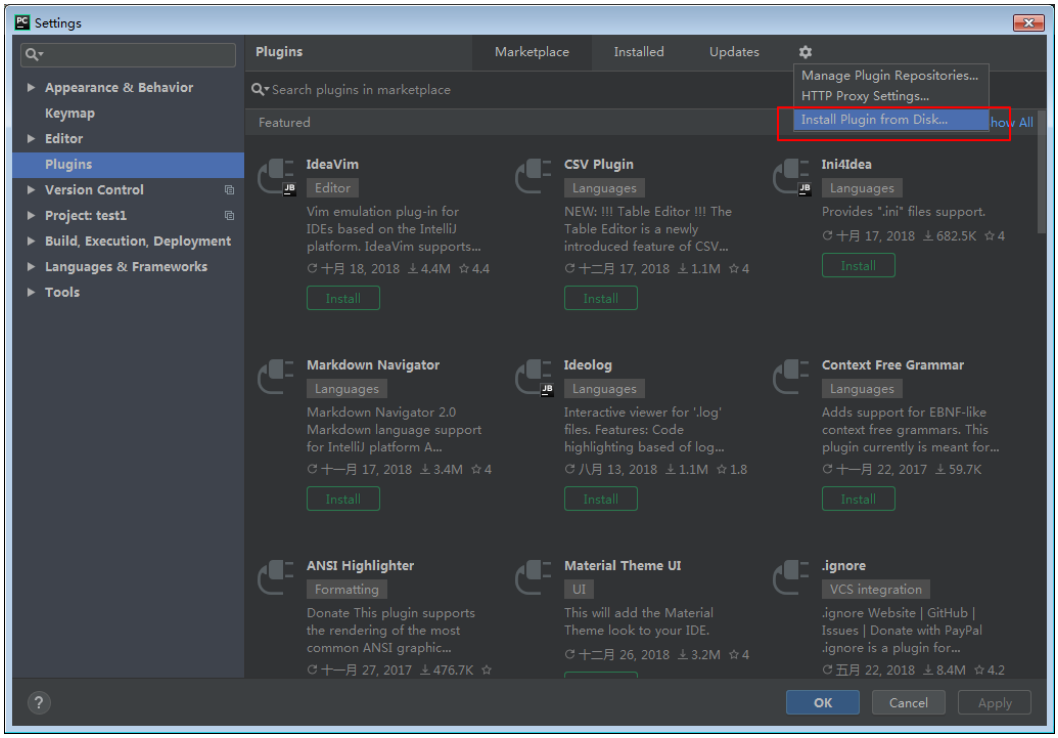
With PyCharm, you can quickly generate Python templates, package project files, and deploy Python functions.

Step 1 Obtain the **PyCharm plugin** (**plugin.sha256**).

Step 2 Run JetBrains PyCharm. Choose **File** > **Settings**, choose **Plugins** in the left pane, and then click **Install Plugin from Disk** in the upper right corner, as shown in **Figure 4-5**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

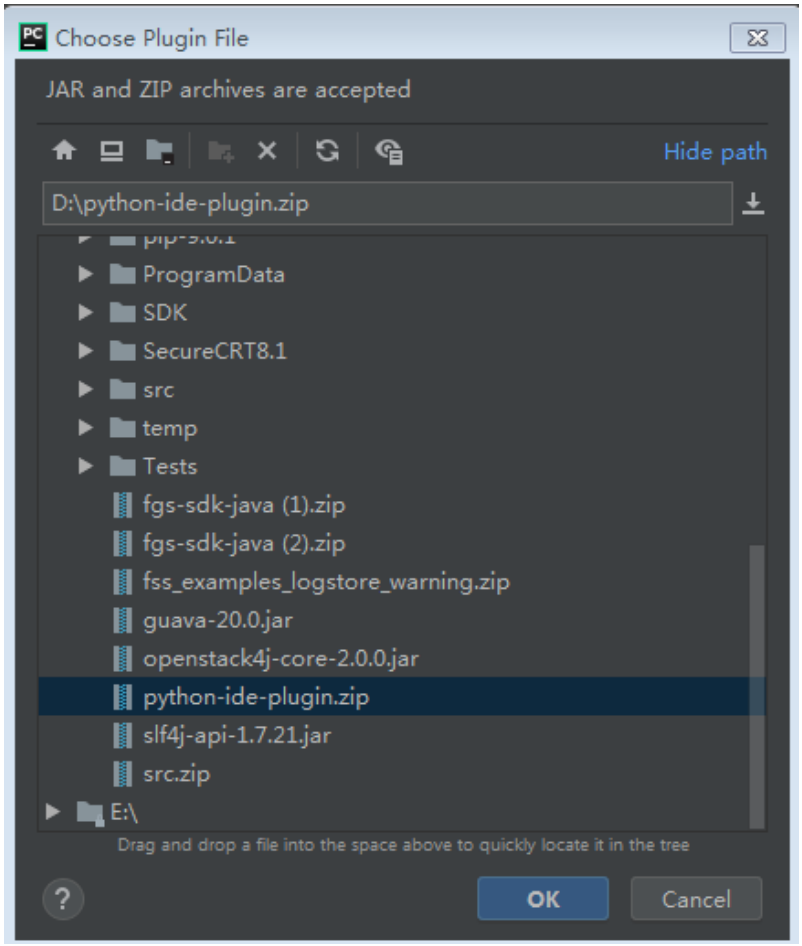
Figure 4-5 Installing the plug-in



Step 3 Select the plug-in package you want to install, and click **OK**, as shown in **Figure 4-6**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

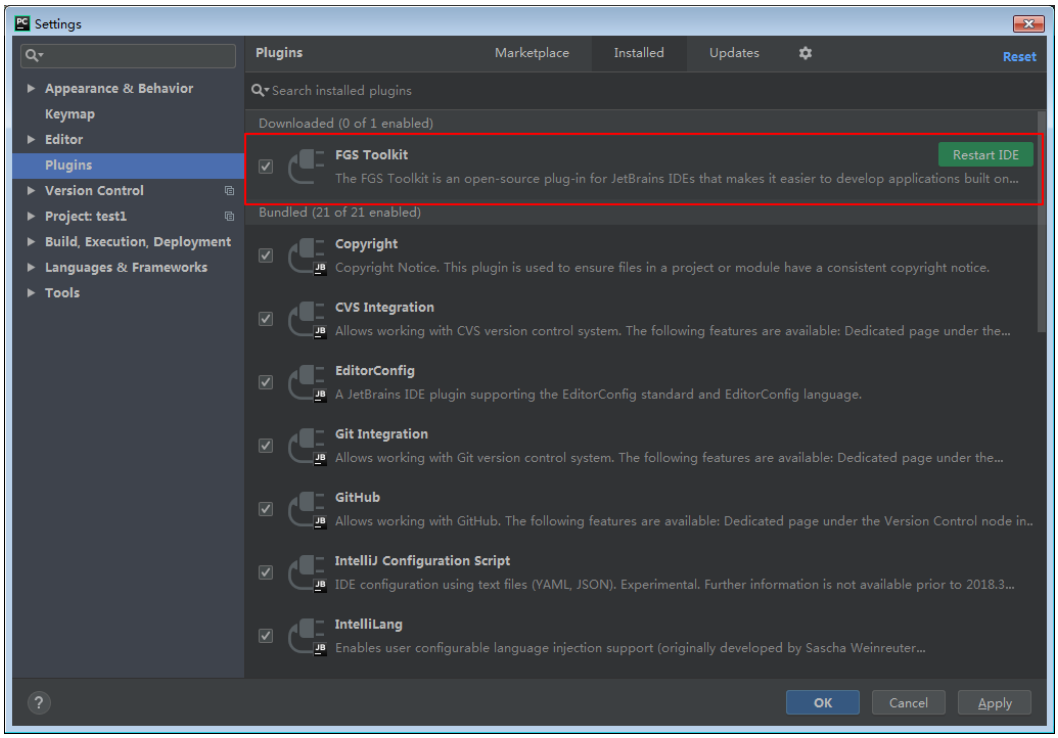
Figure 4-6 Selecting a plug-in package



Step 4 In the plug-in list, select the desired plug-in and click **Restart IDE**, as shown in **Figure 4-7**.

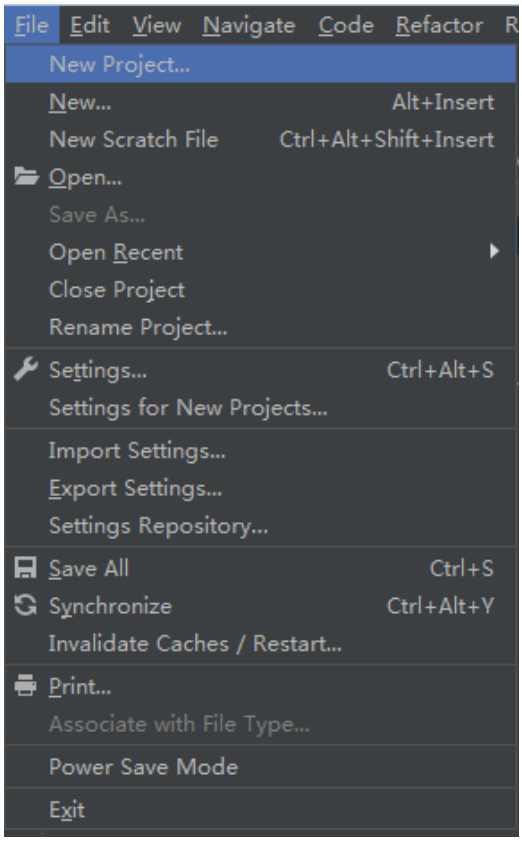
 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 4-7 Restarting the IDE



Step 5 Choose **File** > **New Project**, as shown in **Figure 4-8**.

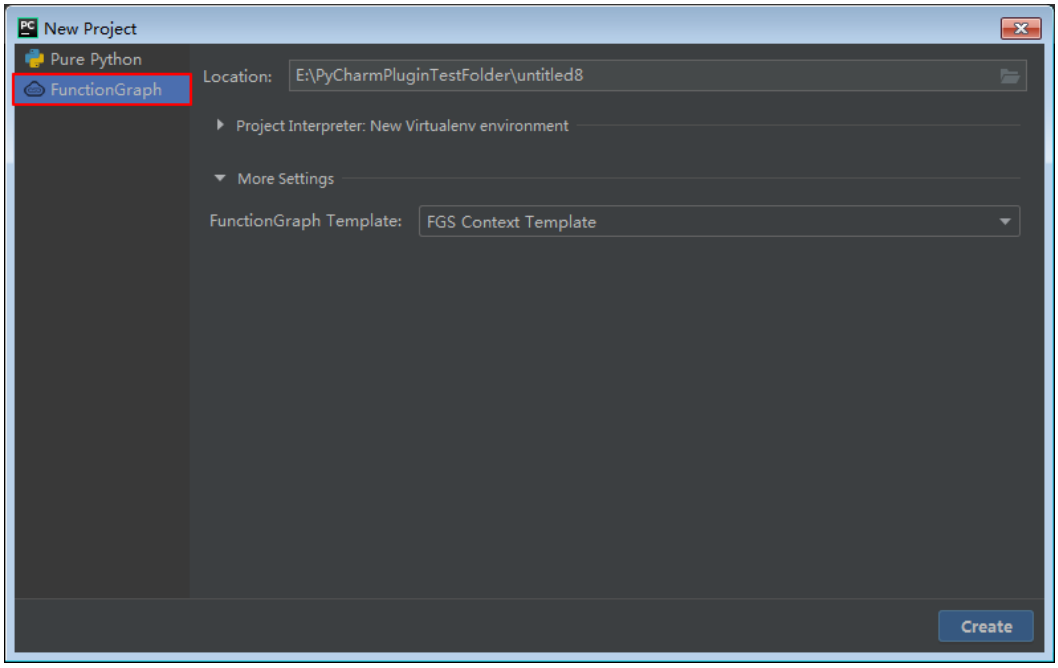
Figure 4-8 Creating a project



 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

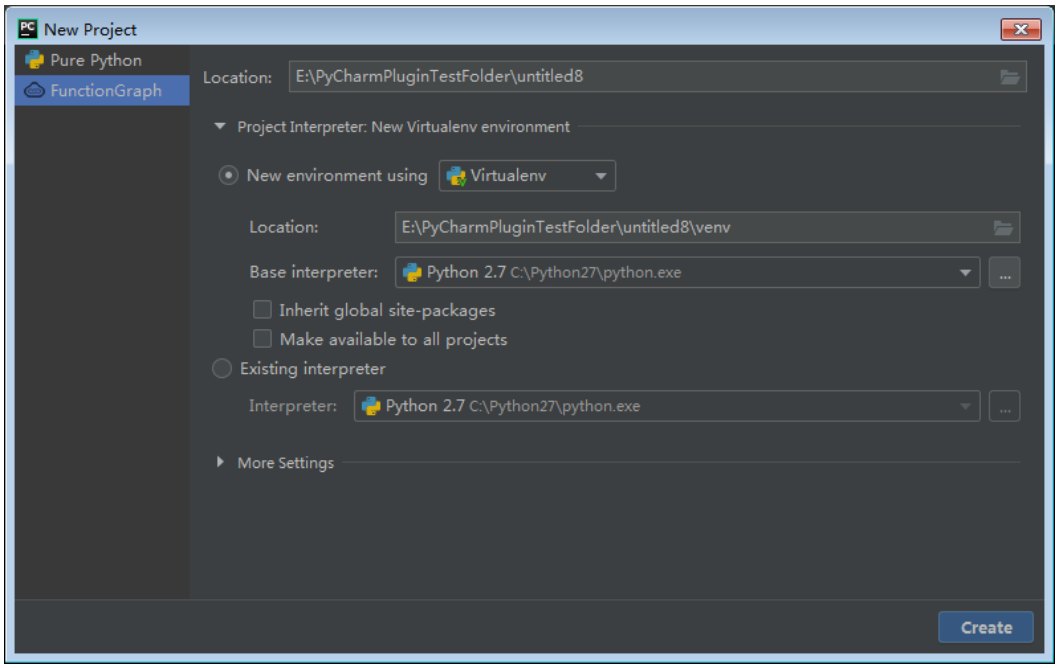
Step 6 On the displayed **New Project** page, choose **FunctionGraph**, as shown in **Figure 4-9**.

Figure 4-9 FunctionGraph



Step 7 Select the path in which the project will be stored in **Location**, and select a Python version in **Base interpreter**, as shown in **Figure 4-10**.

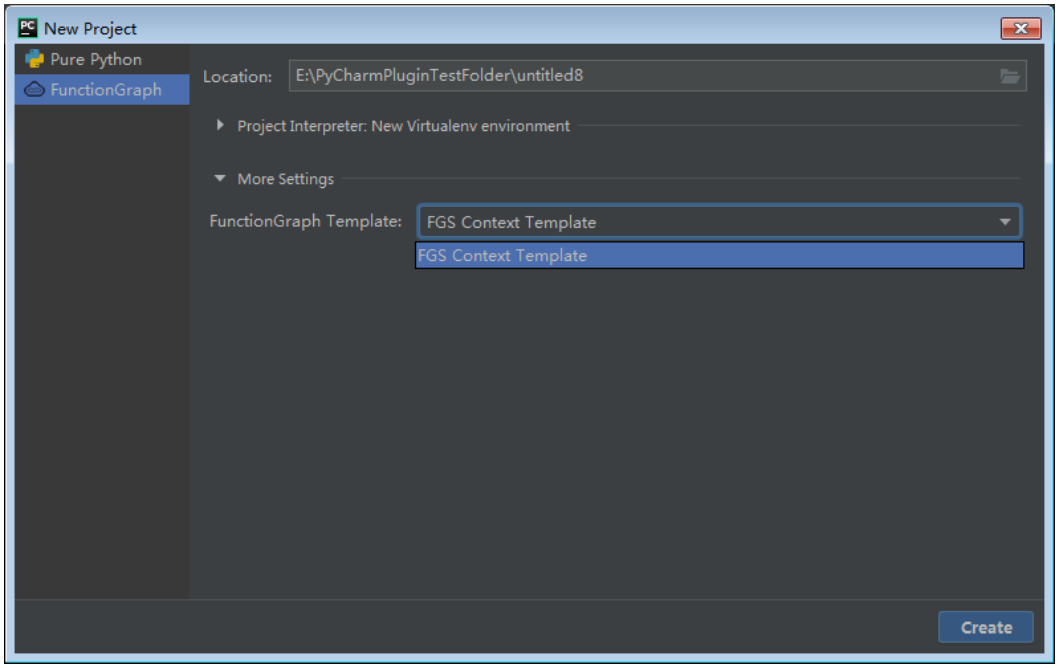
Figure 4-10 Selecting a version



Step 8 Select a template you want to create in the **More Settings** area, as shown in **Figure 4-11**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 4-11 Selecting a template



NOTE

Currently, only the Python 2.7 context template is supported.

Step 9 Click **Create**.

----End

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.