FunctionGraph

Developer Guide

Issue 01

Date 2024-12-05

HUAWEI

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.

Contents

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.	-
Python	2.7, 3.6, 3.9	-
Java	8, 11	**Java SDK** (software package verification file: **fss-java-sdk-2.0.0_sha256**) **NOTE** The Java runtime has integrated with Object Storage Service (OBS) SDKs.
Go	1.x	**Go1.x SDK** (software package verification file: **Go1.x SDK_sha256**)

Runtime	Supported Version	SDK Download Link
C#	.NET Core 2.1, .NET Core 3.1	C# SDK (software package verification file: fssCsharp2.0-1.0.1_sha256)
PHP	7.3	-
Custom	-	-
Cangjie	1.0	-

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
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	Software Package Verification File
Node.js function	fss_examples_nodejs.zip	fss_examples_nodejs.sha256
Python function	fss_examples_python2.7.zip	fss_examples_python2.7_sha256
Java function	fss_example_java8.jar	fss_example_java8_sha256
Go function	fss_examples_go1.8.zip	fss_examples_go1.8_sha256
C# function	fss_example_csharp2.0 and fss_example_csharp2.1	fss_example_csharp2.0_sha256 fss_example_csharp2.1_sha256
PHP function	fss_examples_php7.3.zip	fss_examples_php7.3_sha256

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 cloud services. For details, see **Using an SMN Trigger**.

APIG

API Gateway (APIG) is an API hosting service that helps enterprises to build, manage, and deploy APIs at any scale. With APIG, 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. APIG invokes the relevant function when an HTTPS request (**APIG example event**) is sent to the API backend. For details, see **Using an APIG Trigger**.

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 **Using a Timer Trigger**.

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 **Using a Kafka Trigger**.

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 **Viewing Function Metrics**.

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

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

Table 1-5 Parameter description

Parameter	Type	Example Value	Description
event_version	String	1.0	Event version
topic_urn	String	See the example.	ID of an SMN event
type	String	notification	Event type
RequestID	String	7c307f6a-cf68-4e65-8be0-4c77405a1b2c	Request ID. The ID of each request is unique.
message_id	String	a51671f77d4a479cacb09e2cd591a983	Message ID. The ID of each message is unique.
Message	String	this is smn message content	Message content
event_source	String	smn	Event source
event_subscription_urn	String	See the example.	Subscription ID
timestamp	String	Wed Nov 15 2017 12:00:00 GMT+0800 (CST)	Time when an event occurs

- APIG example event

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

NOTE

- When calling a function using APIG, **isBase64Encoded** is valued **true** by default, indicating that the request body transferred to FunctionGraph is encoded using Base64 and must be decoded for processing.
- The function must return characters strings by using the following structure.

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

Table 1-6 Parameter description

Parameter	Type	Example Value	Description
body	String	"{\"test\":\"body\"}"	Actual request in string format
requestContext	Map	See the example.	Request information, including the API gateway configuration, request ID, authentication information, and source

Parameter	Type	Example Value	Description
httpMethod	String	GET	HTTP method
queryStringParameters	Map	See the example.	Query strings configured in APIG and their actual values
pathParameters	Map	See the example.	Path parameters configured in APIG and their actual values
headers	Map	See the example.	Complete headers
path	String	/apig-event-template	Complete path
isBase64Encoded	Boolean	True	Default value: **true**

- Timer example event

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

Table 1-7 Parameter description

Parameter	Type	Example Value	Description
version	String	V1.0	Event version
time	String	2018-06-01T08:30:00+08:00	Time when an event occurs.
trigger_type	String	TIMER	Trigger type
trigger_name	String	Timer_001	Trigger name
user_event	String	User Event	Additional information of the trigger

- Kafka example event

```
{
    "event_version": "v1.0",
    "event_time": 1576737962,
    "trigger_type": "KAFKA",
    "region": "{region}",
    "instance_id": "81335d56-b9fe-4679-ba95-7030949cc76b",
    "records": [
        {
```



```
        "messages": [
            "kafka message1",
            "kafka message2",
            "kafka message3",
            "kafka message4",
            "kafka message5"
        ],
        "topic_id": "topic-test"
    }
  ]
}
```

Table 1-8 Parameter description

Parameter	Type	Example Value	Description
event_version	String	v1.0	Event version
event_time	String	2018-01-09T07:50:50.028Z	Time when an event occurs
trigger_type	String	KAFKA	Event type
region	String	ap-southeast-3	Region where a Kafka instance resides
instance_id	String	81335d56-b9fe-4679-ba95-7030949cc76b	Kafka instance ID
messages	String	See the example.	Message content
topic_id	String	topic-test	Message ID

- RabbitMQ example event

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


Table 1-9 Parameter description

Parameter	Type	Example Value	Description
event_version	String	v1.0	Event version
Region	String	ap-southeast-3	Region where a RabbitMQ instance resides
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 1-10** describes the rules for packaging a function project.

Table 1-10 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.

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.

Runtime	JAR File	ZIP File	ZIP File on OBS
Python 2.7	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.

Runtime	JAR File	ZIP File	ZIP File on OBS
Python 3.6	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 8	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.

Runtime	JAR File	ZIP File	ZIP File on OBS
Go 1.x	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 **Handler**, set the name of the handler to **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.

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

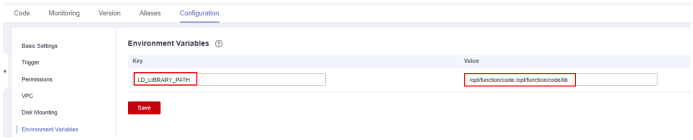
```
Example.zip                      Example project package
|--- bootstrap                   Executable boot file
```

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.
- You can directly modify the **LD_LIBRARY_PATH** variable in the code.
- If the dependent **.so** file is stored in another directory, you can specify it when setting the **LD_LIBRARY_PATH** environment variable. For details, see **Configuring Environment Variables**.

  In the following example, **/opt/function/code** and **/opt/function/code/lib** indicate the project directories of the function code.

Figure 1-1 Setting environment variables



- If a library in a mounted file system is used, specify its directory in the **LD_LIBRARY_PATH** variable on the **Configuration** tab page.

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 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

- 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.

- 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 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.

- Execution duration

  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.

3 Node.js

3.1 Developing an Event Function

Function Syntax

NOTE

You are advised to use Node.js 12.13.

- 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, Node.js 10.16, Node.js 12.13, Node.js 14.18, Node.js 16.17, and Node.js 18.15

  Node.js 8.10, Node.js 10.16, Node.js 12.13, Node.js 14.18, Node.js 16.17, and Node.js 18.15 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.js6.10 (runtime = Node.js6)
- Node.js8.10 (runtime = Node.js8)
- Nodejs10.16(runtime = Node.js10)
- Nodejs12.13(runtime = Node.js12)
- Node.js14.18(runtime = Node.js14)
- Node.js16.17(runtime = Node.js16)

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.

Method	Description
getAccessKey()	Obtains the AK (valid for 24 hours) with 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) with 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()	CPU usage of a function.
getPackage()	Obtains a function group, that is, an app.
getToken()	Obtains the token (valid for 24 hours) with an agency. If you use this method, you need to configure an agency for the function.

Method	Description
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")
getAlias	Obtains function alias.

NOTICE

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

Developing a Node.js Function

The following procedure uses a local function as an example. You can also create a new one on the console.

Step 1 Create a function project.

1. Write function code with a sync handler.

   Open the text editor, compile a function, and save the code file as **index.js**. The following is a sync handler:

```
exports.handler = function (event, context, callback) {
    const error = null;
    const output = {
        'statusCode': 200,
        'headers':
        {
            'Content-Type': 'application/json'
        },
        'isBase64Encoded': false,
        'body': JSON.stringify(event),
    }
    callback(error, output);
}
```


NOTE

1. If the first parameter returned by **callback** is not **null**, the function execution fails and the HTTP error message defined in the second parameter is returned.
2. When an APIG trigger is used, the response must be in the output format used in this example. The body only supports the following values: For details about the constraints, see **Base64 Decoding and Response Structure**.

   null: The HTTP response body is empty.

   []byte: The content in this byte array is the body of an HTTP response.

   string: The content in this string is the body of an HTTP response.

2. Write function code with an async handler and runtime 8.10 or later.

```
exports.handler = async (event, context) => {
    const output =
    {
        'statusCode': 200,
        'headers':
        {
            'Content-Type': 'application/json'
        },
        'isBase64Encoded': false,
        'body': JSON.stringify(event),
    }
    return output;
}
```

If your Node.js function contains an asynchronous task, use **Promise** to execute the task in the current invocation. You can directly return the declared **Promise** or await to execute it. The asynchronous task can be executed only before the function responds to requests.

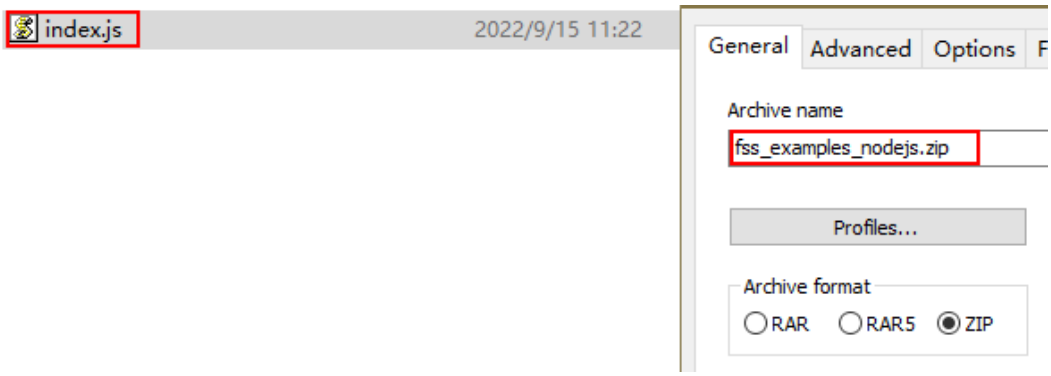
```
exports.handler =  async(event, context ) => {
    const output =
    {
        'statusCode': 200,
        'headers':
        {
            'Content-Type': 'application/json'
        },
        'isBase64Encoded': false,
        'body': JSON.stringify(event),
    }

    const promise = new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve(output)
        }, 2000)
    })
    return promise
    // anthor way
    // res = await promise;
    // return res
}
```

If you want the function to first respond and then execute the task, use an SDK or directly call the **asynchronous invocation** API. When using an APIG trigger, click the trigger name to go to the APIG console, and call the API in asynchronous mode.

Step 2 Package the project files.

This step uses an async handler as an example. After creating the function project, you get the following directory. Select all files under the directory and package them into the **fss_examples_nodejs.zip** file.

Figure 3-1 Packaging the project files



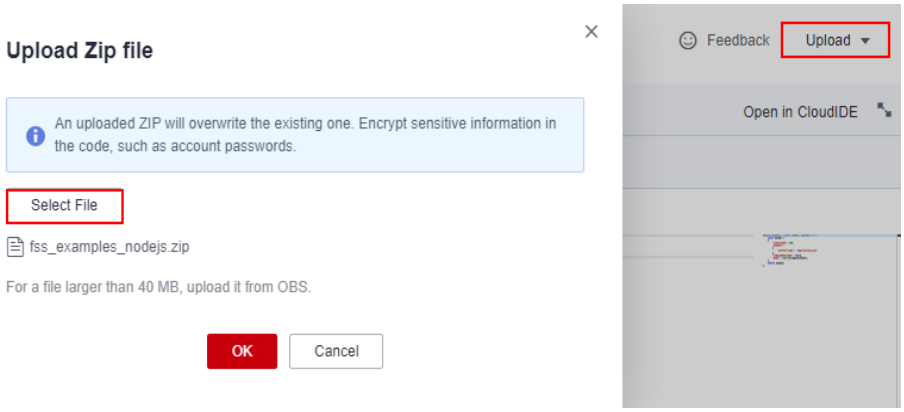
NOTICE

In this example, the function project files are saved under the **~/Code/** directory. Select and package all files under the directory to ensure that the **index.js** file, the handler of your FunctionGraph function, is under the root directory when the **fss_examples_nodejs.zip** file is decompressed.

Step 3 Create a FunctionGraph function and upload the code package.

Log in to the FunctionGraph console, create a Node.js function, and upload the **fss_examples_nodejs.zip** file, as shown in **Figure 3-2**.

Figure 3-2 Uploading the code package

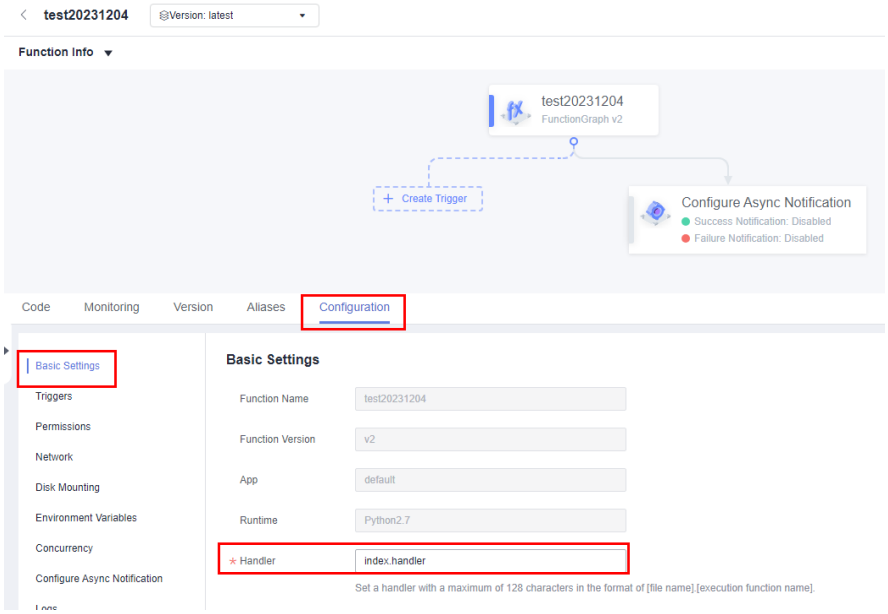


NOTE

1. In the function configuration, the **index** of the **handler** must be consistent with the name of the function file created in **Step 1**, because the file name will help to locate the function file.
2. The **handler** is a function name, which must be the same as that in the **index.js** file created in **Step 1**.

In the navigation pane on the left of the FunctionGraph console, choose **Functions** > **Function List**. Click the name of the function to be set. On the function details page that is displayed, choose **Configuration** > **Basic Settings** and set the **Handler** parameter, as shown in **Figure 3-3**. The parameter value is in the format of **index.handler**. The values of **index** and **handler** can be customized.

Figure 3-3 Handler parameter

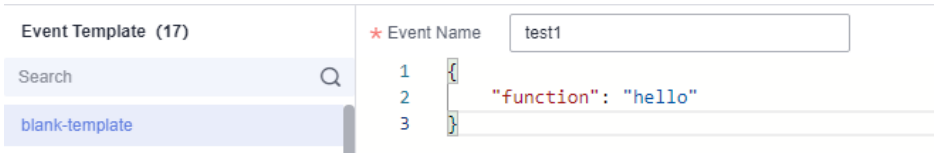


Step 4 Test the function.

1. Create a test event.

   On the function details page that is displayed, click **Configure Test Event**. Configure the test event information, as shown in **Figure 3-4**, and then click **Create**.

   Figure 3-4 Configuring a test event

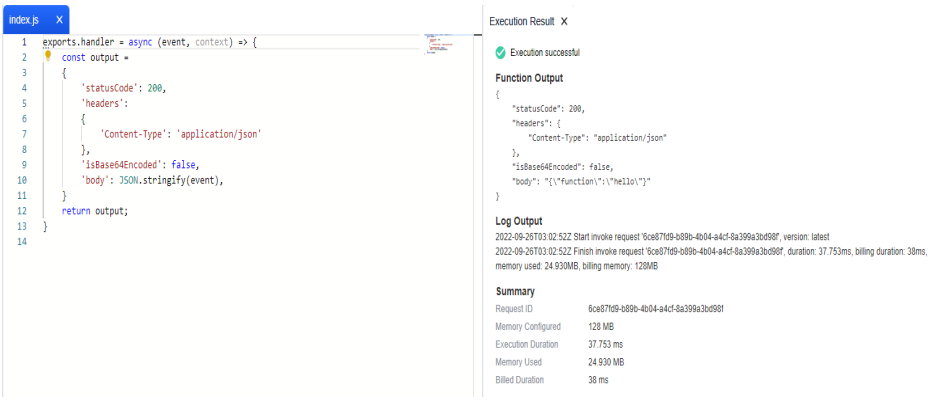
   

2. On the function details page, select the configured test event, and click **Test**.

Step 5 View the function execution result.

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), as shown in **Figure 3-5**.

Figure 3-5 Test result



----End

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.
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 an HTTP Function

Deploy the Koa framework by using an HTTP function. For details, see **Creating an HTTP Function**.

Prerequisites

1. You have prepared a bootstrap file as the startup file of the HTTP function. Example:
   ```
   /opt/function/runtime/nodejs14.18/rtsp/nodejs/bin/node $RUNTIME_CODE_ROOT/index.js
   ```
   - **/opt/function/runtime/nodejs14.18/rtsp/nodejs/bin/node**: path of the Node.js compilation environment.
   - *$RUNTIME_CODE_ROOT*: system variable that represents the **/opt/function/code** path for storing project code in a container.
   - **index.js**: project file. You can also define a custom name.

   Table 3-3 lists the supported Node.js versions and the corresponding paths.

   Table 3-3 Node.js paths

Language	Path
Node.js 6	/opt/function/runtime/nodejs6.10/rtsp/nodejs/bin/node
Node.js 8	/opt/function/runtime/nodejs8.10/rtsp/nodejs/bin/node
Node.js 10	/opt/function/runtime/nodejs10.16/rtsp/nodejs/bin/node
Node.js 12	/opt/function/runtime/nodejs12.13/rtsp/nodejs/bin/node
Node.js 14	/opt/function/runtime/nodejs14.18/rtsp/nodejs/bin/node

2. Install the Node.js environment on a Linux host and prepare the Node.js project file.

   Koa web application

   a. Create a project folder.
      ```
      mkdir koa-example && cd koa-example
      ```
   b. Initialize the Node.js project and download the Koa framework. The **node_modules** folder and the **package.json** and **package-lock.json** files are created in the folder.
      ```
      npm init -y
      npm i koa
      ```
   c. Create the **index.js** file to reference the Koa framework. For details about how to use this framework, see **Koa's guide**.

      Sample code:
      ```
      const Koa = require("koa");
      const app = new Koa();
      const main = (ctx) = >{
          if (ctx.request.path == ("/koa")) {
              ctx.response.type = " application/json";
              ctx.response.body = "Hello World, user!";
              ctx.response.status = 200;
          } else {
              ctx.response.type = " application/json";
              ctx.response.body = 'Hello World!';
              ctx.response.status = 200;
          }
      };
      app.use(main);
      app.listen(8000, '127.0.0.1');
      console.log('Node.js web server at port 8000 is running..')
      ```


NOTE

- HTTP functions can only use APIG or APIC triggers. According to the forwarding protocol between FunctionGraph and APIG/APIC, a valid HTTP function response must contain **body(String)**, **statusCode(int)**, **headers(Map)**, and **isBase64Encoded(boolean)**. By default, the response is encoded using Base64. The default value of **isBase64Encoded** is **true**. The same applies to other frameworks. For details about the constraints, see **Base64 Decoding and Response Structure**.
- By default, port 8000 is enabled for HTTP functions.
- **Table 3-1** describes the context methods provided by FunctionGraph.

d. Create a bootstrap file.

```
/opt/function/runtime/nodejs14.18/rtsp/nodejs/bin/node $RUNTIME_CODE_ROOT/index.js
```

3. Compress the project files and the bootstrap file into a ZIP package. The Koa framework is used as an example.

```
[root@                koa-example]# ls
bootstrap  index.js  koa.zip  node_modules  package.json  package-lock.json
```

3.3 Node.js Template

```
exports.handler = async (event, context) => {
    const output =
    {
        'statusCode': 200,
        'headers':
        {
            'Content-Type': 'application/json'
        },
        'isBase64Encoded': false,
        'body': JSON.stringify(event),
    }
    return output;
}
```

3.4 Creating a Dependency

You are advised to create function dependencies in Huawei Cloud EulerOS 2.0. If other OSs are used, an error may occur due to underlying dependent libraries. For example, the dynamic link library cannot be found.

NOTE

- If the modules to be installed need dependencies such as **.dll**, **.so**, and **.a**, archive them to a **.zip** package.

Creating a Dependency for a Node.js Function

Ensure that the corresponding Node.js version has been installed in the environment.

To install the MySQL dependency for a Node.js 8.10 function, run the following command:

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


The **node_modules** folder is generated under the current directory.

- Linux OS

  Run the following command to generate a ZIP package.

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

  The required dependency is generated.

- Windows OS

  Compress **node_modules** into a ZIP file.

To install multiple dependencies, create a **package.json** file first. For example, enter the following content into the **package.json** file and then run the following command:

```
{
  "name": "test",
  "version": "1.0.0",
  "dependencies": {
      "redis": "~2.8.0",
      "mysql": "~2.17.1"
  }
}
npm install --save
```

NOTE

Do not run the **CNPM** command to generate Node.js dependencies.

Compress **node_modules** into a ZIP package. This generates a dependency that contains both MySQL and Redis.

For other Node.js versions, you can create dependencies in the way stated above.

4 Python

4.1 Developing an Event Function

Function Syntax

> NOTE
>
> You are advised to use Python 3.6.

FunctionGraph supports Python 2.7, Python 3.6, Python 3.9, and Python 3.10.

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 (runtime = python2.7)
- Python 3.6 (runtime = python3)
- Python 3.9 (runtime = python3)
- Python 3.10 (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 (Python 2.7 is used as an example):

```
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.

SDK APIs

Table 4-1 describes the context methods provided by FunctionGraph.

Table 4-1 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) with 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) with 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.

Method	Description
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()	CPU usage of a function.
getPackage()	Obtains a function group, that is, an app.
getToken()	Obtains the token (valid for 24 hours) with 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: log = context.getLogger() log.info("test")
getAlias	Obtains function alias.

NOTICE

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

Developing a Python Function

Perform the following steps to develop a Python function:

> NOTE
>
> Python 2.7 is used an example.

Step 1 Create a function project.

1. Write code for printing text **helloworld**.

   Open the text editor, compile a HelloWorld function, and save the code file as **helloworld.py**. The code is as follows:

   ```
   def printhello():
       print 'hello world!'
   ```

2. Define a FunctionGraph function.

   Open the text editor, define a function, and save the function file as **index.py** under the same directory as the **helloworld.py** file. The function code is as follows:

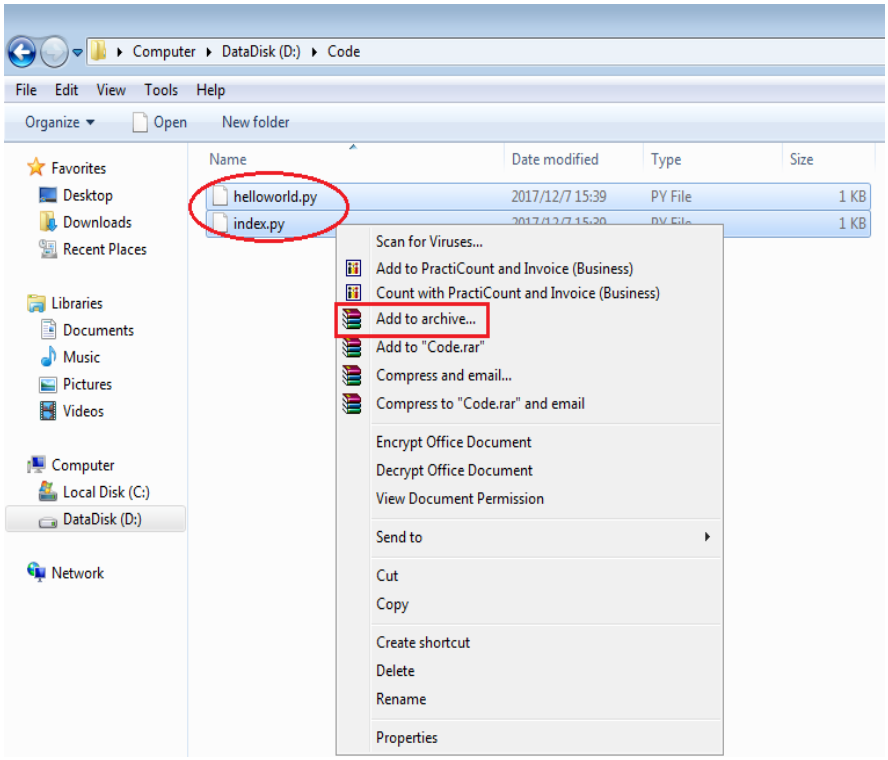
   ```
   import json
   import helloworld

   def handler (event, context):
       output =json.dumps(event)
       helloworld.printhello()
       return output
   ```

> NOTE
>
> FunctionGraph can return only the following types of values:
>
> - **None**: The HTTP response body is empty.
> - **String**: The content in this string is the body of an HTTP response.
> - **Other**: For a value rather than **None** or **String**, FunctionGraph encodes the value in JSON, and uses the encoded object as the body of an HTTP response. The **Content-Type** header of the HTTP response is set to **application/json**.
> - For details about the constraints for the APIG event source, see **Base64 Decoding and Response Structure**.

Step 2 Package the project files.

After creating the function project, you get the following directory. Select all files under the directory and package them into the **fss_examples_python2.7.zip** file, as shown in **Figure 4-1**.

Figure 4-1 Packaging the project files



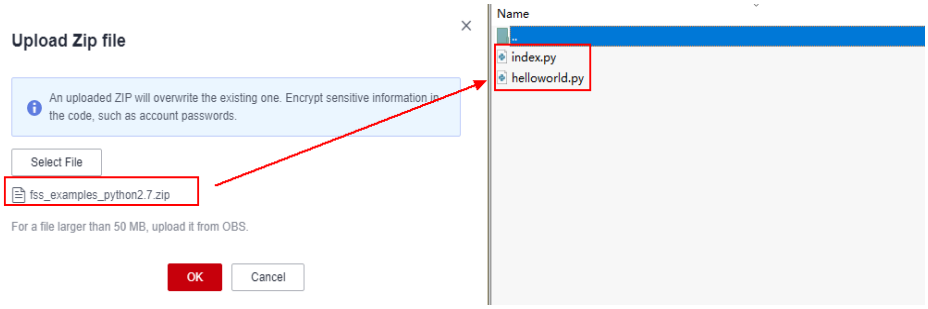
NOTICE

- In this example, the function project files are saved under the **~/code/** directory. Select and package all files under the directory to ensure that the **index.py** file, the handler of your FunctionGraph function, is under the **root** directory when the **fss_examples_python2.7.zip** file is decompressed.
- When you write code in Python, do not name your package with the same suffix as a standard Python library, such as **json**, **lib**, and **os**. Otherwise, an error will be reported indicating a module loading failure.

Step 3 Create a FunctionGraph function and upload the code package.

Log in to the FunctionGraph console, create a Python function, and upload the **fss_examples_python2.7.zip** file, as shown in **Figure 4-2**.

Figure 4-2 Uploading the code package

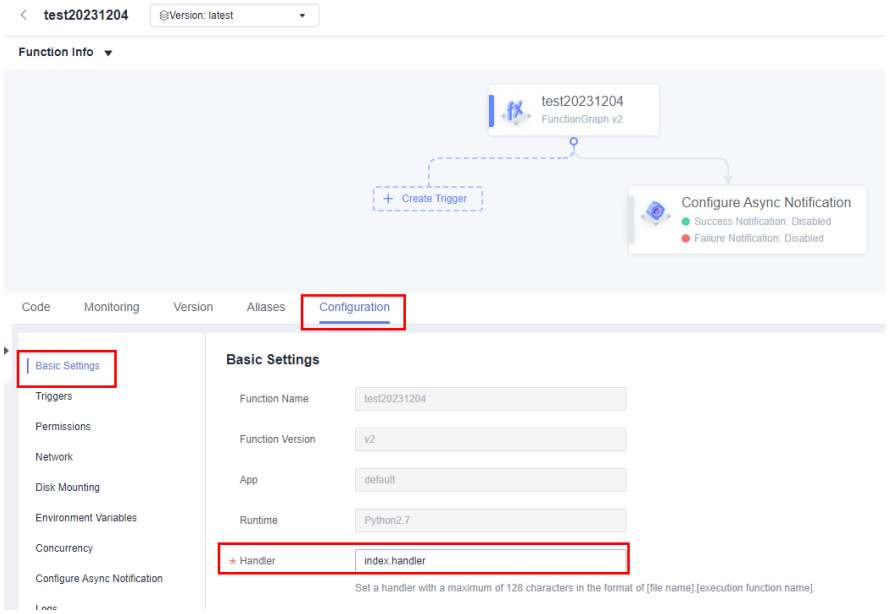


NOTE

1. The **index** of the **handler** must be consistent with the name of the file created in **Step 1 (2)**, because the file name will help to locate the function file.
2. The **handler** is the function name, which must be the same as that in the **index.py** file created in **Step 1 (2)**.
3. After you upload the **fss_examples_python2.7.zip** file to OBS, when the function is triggered, FunctionGraph decompresses the file to locate the function file through **index** and locate the function defined in the **index.py** file through **handler**, and then executes the function.

In the navigation pane on the left of the FunctionGraph console, choose **Functions** > **Function List**. Click the name of the function to be set. On the function details page that is displayed, choose **Configuration** > **Basic Settings** and set the **Handler** parameter, as shown in **Figure 4-3**. The parameter value is in the format of **index.handler**. The values of **index** and **handler** can be customized.

Figure 4-3 Handler parameter

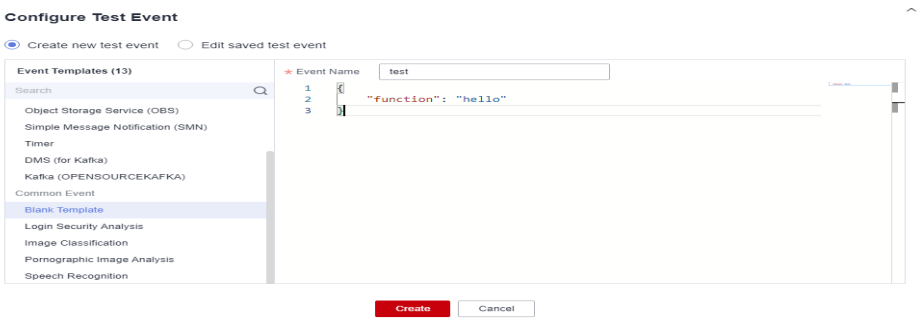


Step 4 Test the function.

1. Create a test event.

   On the function details page that is displayed, click **Configure Test Event**. Configure the test event information, as shown in **Figure 4-4**, and then click **Create**.

   Figure 4-4 Configuring a test event



2. On the function details page, select the configured test event, and click **Test**.

Step 5 View the function execution result.

The function execution result consists of three parts: function output (returned by **callback**), summary, and logs (output by using the **print()** method), as shown in **Figure 4-5**.

Figure 4-5 Test result

----End

Execution Result

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

Table 4-2 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**, **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.

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.

4.2 Python Template

```
# -*- coding:utf-8 -*-
import json
def handler (event, context):
    return {
        "statusCode": 200,
        "isBase64Encoded": False,
        "body": json.dumps(event),
        "headers": {
            "Content-Type": "application/json"
        }
    }
```

4.3 Creating a Dependency

You are advised to create function dependencies in Huawei Cloud EulerOS 2.0. If other OSs are used, an error may occur due to underlying dependent libraries. For example, the dynamic link library cannot be found.

NOTE

- If the modules to be installed need dependencies such as **.dll**, **.so**, and **.a**, archive them to a **.zip** package.

Creating a Dependency for a Python Function

Ensure that the Python version of the packaging environment is the same as that of the function. For Python 2.7, Python 2.7.12 or later is recommended. For Python 3.6, Python 3.6.3 or later is recommended.

To install the PyMySQL dependency for a Python 2.7 function in the local **/tmp/pymysql** directory, run the following command:

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

After the command is successfully executed, go to the **/tmp/pymysql** directory:

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

Go to the **site-packages** directory (generally, **usr/lib64/python2.7/site-packages/**) and then run the following command:

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

The required dependency is generated.

NOTE

To install the local wheel installation package, run the following command:

```
pip install piexif-1.1.0b0-py2.py3-none-any.whl --root /tmp/piexif
//Replace piexif-1.1.0b0-py2.py3-none-any.whl with the actual installation package name.
```


5 Java

5.1 Developing an Event Function

5.1.1 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**.

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

Initializer syntax:

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

For example, if the initializer is named **com.huawei.Demo.my_initializer**, FunctionGraph loads the my_initializer function defined in the **com.huawei** 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.huawei.Demo.my_initializer**, FunctionGraph loads the my_initializer function defined in the **com.huawei** 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 (verification file: fss-java-sdk-2.0.5.sha256)** provides context and logging APIs.

- Event APIs

  Event structure definitions are added to the Java SDK. Currently, DMS, DIS, SMN, timer, APIG, and Kafka triggers are supported. The definitions make coding much simpler when triggers are required.

  a. **APIG trigger**

  i. APIGTriggerEvent methods

Table 5-1 APIGTriggerEvent methods

Method	Description
isBase64Encoded()	Checks whether the body of an event is encoded using Base64.
getHttpMethod()	Obtains the HTTP request method.
getPath()	Obtains the HTTP request path.
getBody()	Obtains the HTTP request body.
getPathParameters()	Obtains all path parameters.
getRequestContext()	Obtains API Gateway configurations (returned as an **APIGRequestContext** object).

Method	Description
getHeaders()	Obtains the HTTP request header.
getQueryStringParameters()	Obtains query parameters. **NOTE** The value of a query parameter cannot be an array. To support an array, customize the corresponding event structure.
getRawBody()	Obtains the content before Base64 encoding.
getUserData()	Obtains the user data set in the APIG custom authorizer.

Table 5-2 APIGRequestContext methods

Method	Description
getApiId()	Obtains the API ID.
getRequestId()	Obtains the request ID of an API request.
getStage()	Obtains the name of the environment in which an API has been published.
getSourceIp()	Obtains the source IP address in the APIG custom authorizer.

ii. APIGTriggerResponse methods

Table 5-3 APIGTriggerResponse construction methods

Method	Description
APIGTriggerResponse()	Set the following parameters: • **headers**: null • **statusCode**: 200 • **body**: "" • **isBase64Encoded**: false
APIGTriggerResponse(statusCode, headers, body)	Set the value of **isBase64Encoded** to **false**, and use the input values of other parameters.

Method	Description
APIGTriggerResponse(statusCode, headers, isBase64Encoded, body)	Set the parameters in sequence.

Table 5-4 APIGTriggerResponse methods

Method	Description
setBody(String body)	Sets the message body.
setHeaders(Map<String,String> headers)	Sets the HTTP response header to be returned.
setStatusCode(int statusCode)	Sets the HTTP status code.
setBase64Encoded(boolean isBase64Encoded)	Configures Base64 encoding for the response body.
setBase64EncodedBody(String body)	Encodes the input with Base64 and configures it in the body.
addHeader(String key, String value)	Adds a group of HTTP headers.
removeHeader(String key)	Removes the specified header.
addHeaders(Map<String,String> headers)	Adds multiple headers.

NOTE

APIGTriggerResponse methods have the **headers** attribute, which can be initialized using the setHeaders method or a constructor function with the **headers** parameter.

b. **DIS trigger**

Table 5-5 DISTriggerEvent methods

Method	Description
getShardID()	Obtains the partition ID.
getMessage()	Obtains the DIS message body (**DISMessage structure**).
getTag()	Obtains the version of a function.
getStreamName()	Obtains the stream name.

Table 5-6 DISMessage methods

Method	Description
getNextPatitionCursor()	Obtains the next partition cursor.
getRecords()	Obtains message records (**DISRecord structure**).
getMillisBehindLatest()	Reserved (Currently, **0** is returned.)

Table 5-7 DISRecord methods

Method	Description
getPartitionKey()	Obtains the data partition.
getData()	Obtains data.
getRawData()	Obtains UTF-8 data strings decoded using Base64.
getSequenceNumber()	Obtains the sequence number (ID of each record).

c. **DMS trigger**

Table 5-8 DMSTriggerEvent methods

Method	Description
getQueueId()	Obtains the queue ID.
getRegion()	Obtains the region name.
getEventType()	Obtains the event type (**MessageCreated** is returned).
getConsumerGroupId()	Obtains the consumer group ID.
getMessages()	Obtains the DMS message (**DMSMessage structure**).

Table 5-9 DMSMessage methods

Method	Description
getBody()	Obtains the message body.
getAttributes()	Obtains the message attribute set.

d. **SMN trigger**

Table 5-10 SMNTriggerEvent method

Method	Description
getRecord()	Obtains message records (**SMNRecord structure**).

Table 5-11 SMNRecord methods

Method	Description
getEventVersion()	Obtains the event version. (Currently, the version is **1.0**.)
getEventSubscriptionUrn()	Obtains the subscription Uniform Resource Name (URN).
getEventSource()	Obtains the event source.
getSmn()	Obtains the message body (**SMNBody structure**).

Table 5-12 SMNBody methods

Method	Description
getTopicUrn()	Obtains the topic URN.
getTimeStamp()	Obtains the timestamp of a message.
getMessageAtrributes()	Obtains the message attribute set.
getMessage()	Obtains the message body.
getType()	Obtains the message type.
getMessageId()	Obtains the message ID.
getSubject()	Obtains the message topic.

e. **Timer trigger**

Table 5-13 TimerTriggerEvent methods

Method	Description
getVersion()	Obtains the version. (Currently, the version is **v1.0**.)

Method	Description
getTime()	Obtains the current time.
getTriggerType()	Obtains the trigger type (**Timer**).
getTriggerName()	Obtains the trigger name.
getUserEvent()	Obtains the additional information of the trigger.

f. **Kafka trigger**

Table 5-14 Kafka trigger

Method	Description
getEventVersion	Obtains event version.
getRegion	Obtains region information.
getEventTime	Obtains event time.
getTriggerType	Obtains trigger type.
getInstanceId	Obtains instance ID.
getRecords	Obtains record.

NOTE

1. When using an APIG trigger, set the first parameter of the handler function (for example, **handler**) to **handler(APIGTriggerEvent event, Context context)**.
2. The preceding TriggerEvent methods have corresponding **set** methods, which are recommended for local debugging. DIS and LTS triggers have getRawData() methods, but do not have setRawData() methods.

- 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 5-15 describes the context APIs provided by FunctionGraph.

  Table 5-15 Context methods

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

Method	Description
getAccessKey( )	Obtains the AK (valid for 24 hours) with 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) with 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( )	CPU usage of a function.
getPackage( )	Obtains a function group, that is, an app.

Method	Description
getToken( )	Obtains the token (valid for 24 hours) with 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.
getAlias	Obtains function alias.

NOTICE

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

- Logging API

  Table 5-16 describes the logging API provided in the Java SDK.

  Table 5-16 Logging API

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

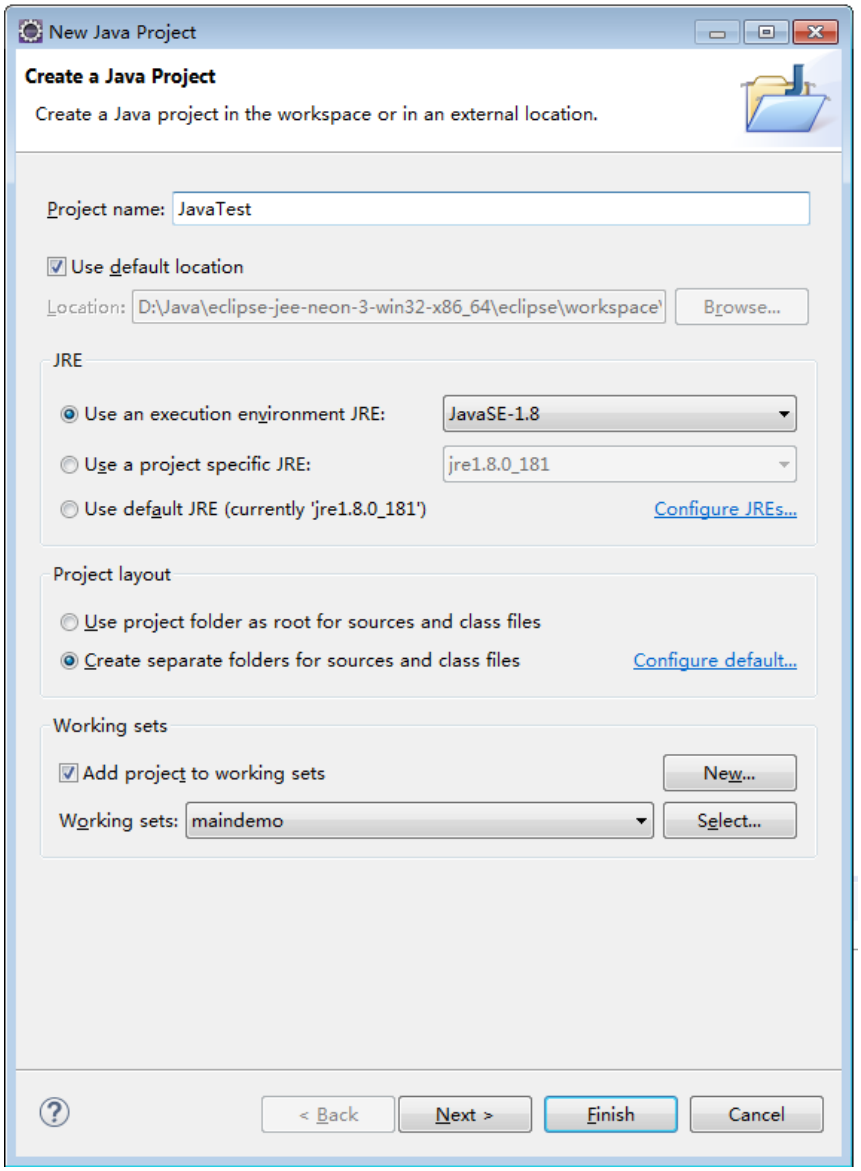
Developing a Java Function

Perform the following procedure to develop a Java function:

Step 1 Create a function project.

1. Configure Eclipse and create a Java project named JavaTest, as shown in **Figure 5-1**.

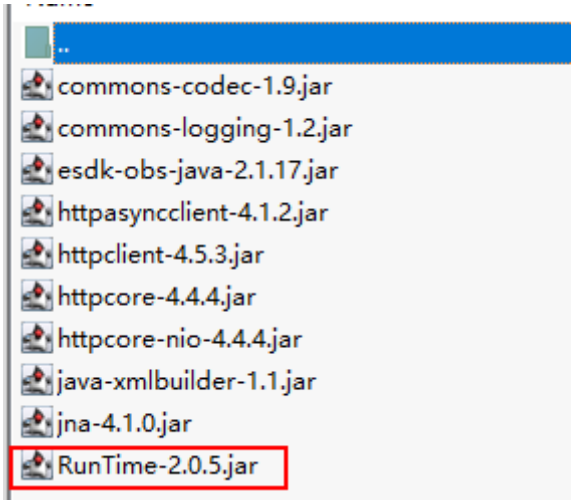
Figure 5-1 Creating a project



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 5-2**.

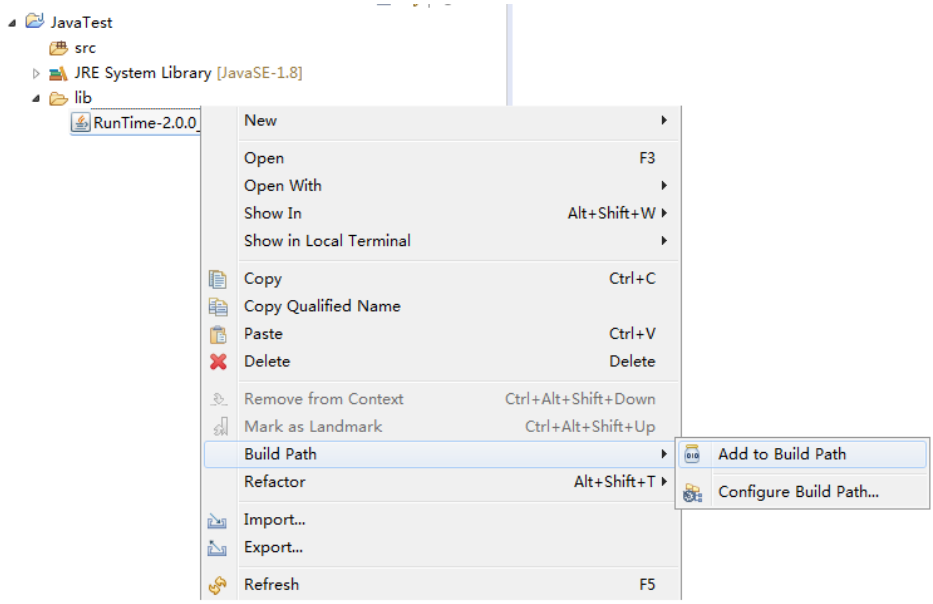
Figure 5-2 Decompressing the downloaded SDK



3. Configure the dependency.

   Create a folder named **lib** in the project directory, copy the **Runtime-2.0.5.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 5-3**.

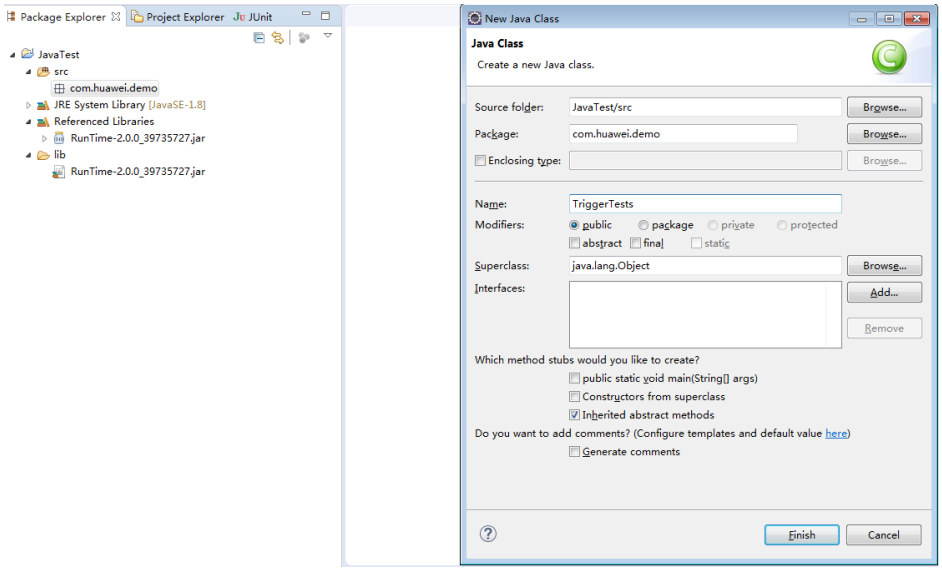
Figure 5-3 Configuring the dependency



Step 2 Create a function.

1. Create a package named **com.huawei.demo**, and then create a class named **TriggerTests** under the package, as shown in **Figure 5-4**.

Figure 5-4 Creating a package named com.huawei.demo



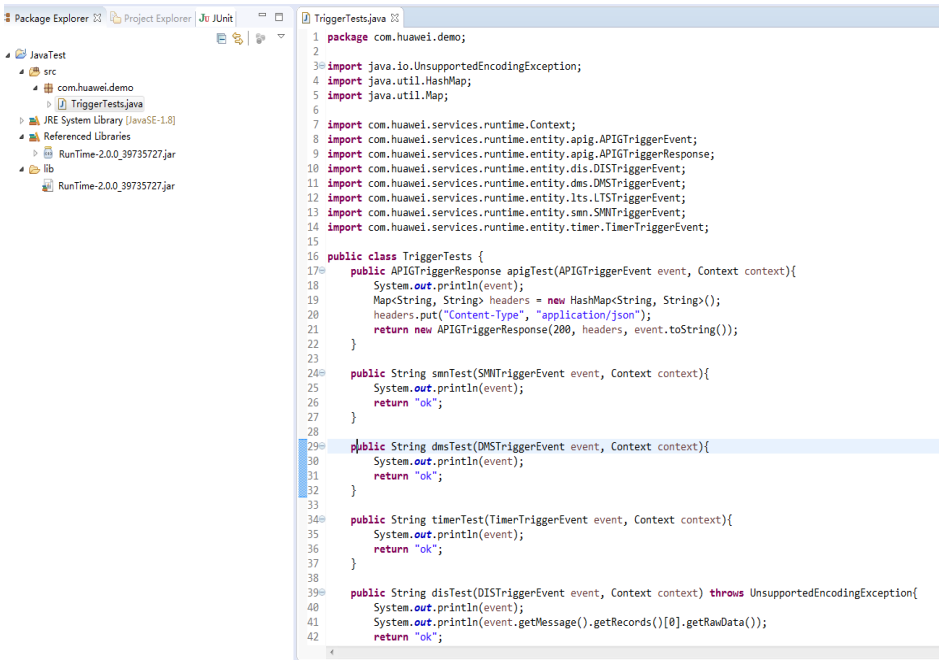
2. Define the function handler in **TriggerTests.java**, as shown in **Figure 5-5**.

```
package com.huawei.demo;

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

import com.huawei.services.runtime.Context;
import com.huawei.services.runtime.entity.apig.APIGTriggerEvent;
import com.huawei.services.runtime.entity.apig.APIGTriggerResponse;
import com.huawei.services.runtime.entity.dis.DISTriggerEvent;
import com.huawei.services.runtime.entity.dms.DMSTriggerEvent;
import com.huawei.services.runtime.entity.smn.SMNTriggerEvent;
import com.huawei.services.runtime.entity.timer.TimerTriggerEvent;

public class TriggerTests {
    public APIGTriggerResponse apigTest(APIGTriggerEvent event, Context context){
        System.out.println(event);
        Map<String, String> headers = new HashMap<String, String>();
        headers.put("Content-Type", "application/json");
        return new APIGTriggerResponse(200, headers, event.toString());
    }

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

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

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

    public String disTest(DISTriggerEvent event, Context context) throws
UnsupportedEncodingException{
        System.out.println(event);
        System.out.println(event.getMessage().getRecords()[0].getRawData());
        return "ok";
    }
```



```
}
```

Figure 5-5 Defining the function handler



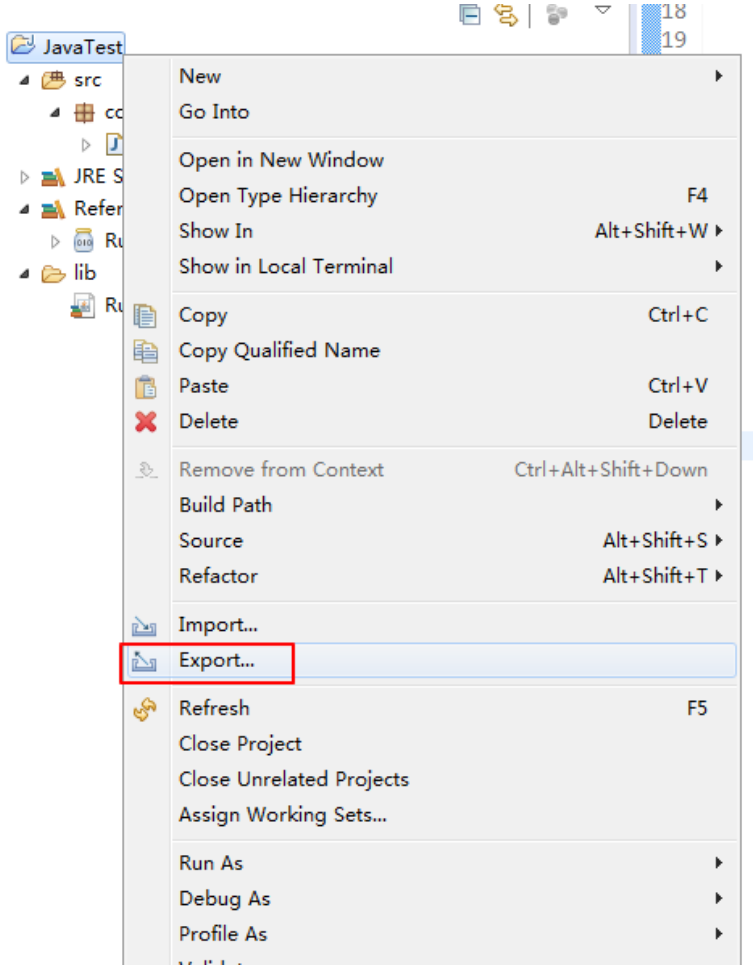
NOTE

For details about the constraints for the APIG event source, see **Base64 Decoding and Response Structure**.

Step 3 Package the project files.

1. Right-click the **JavaTest** project and choose **Export**, as shown in **Figure 5-6**.

Figure 5-6 Packaging the project files



2. Export the project to a directory as a JAR file, as shown in **Figure 5-7** and **Figure 5-8**.

Figure 5-7 Selecting a format

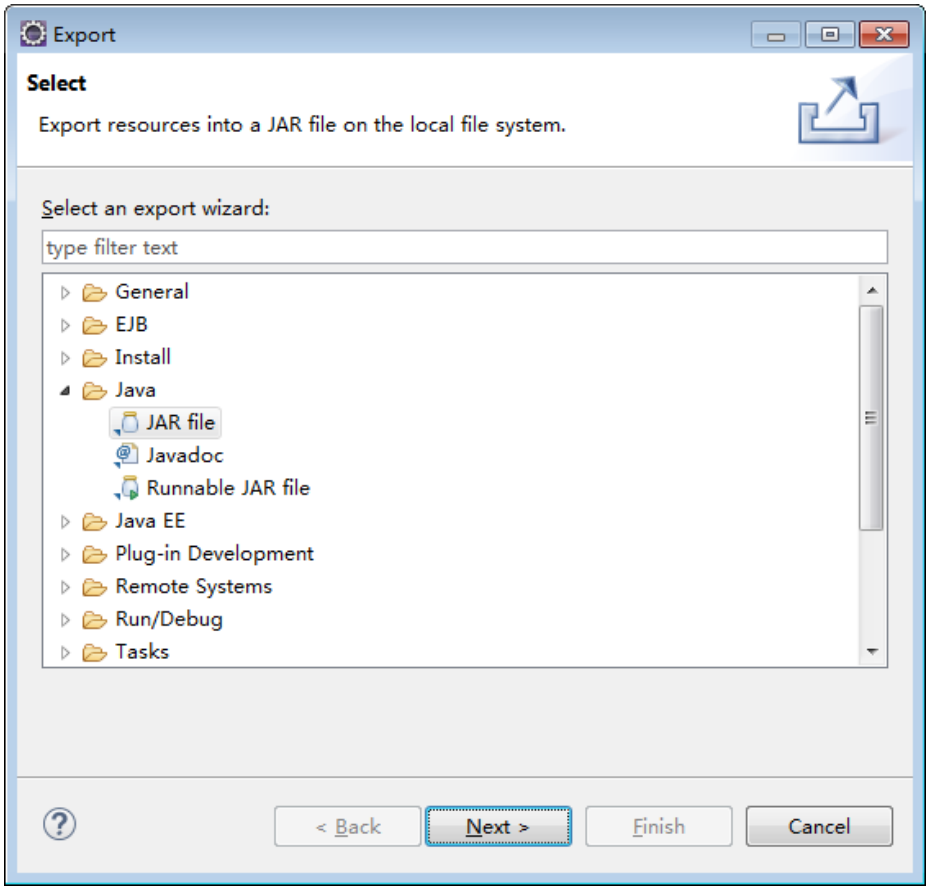
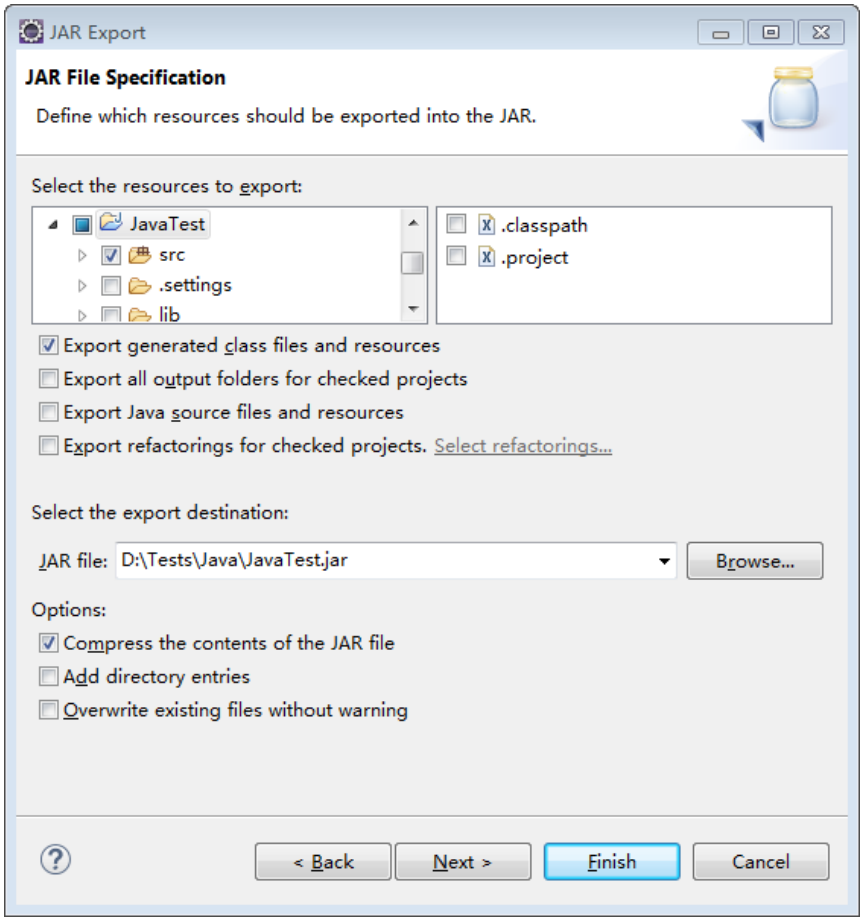


Figure 5-8 Specifying the destination path



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 test event.

   Select **apig-event-template** and click **Create**.

2. 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).

3. Create an APIG trigger.

4. Access the API URL, as shown in **Figure 5-9**.

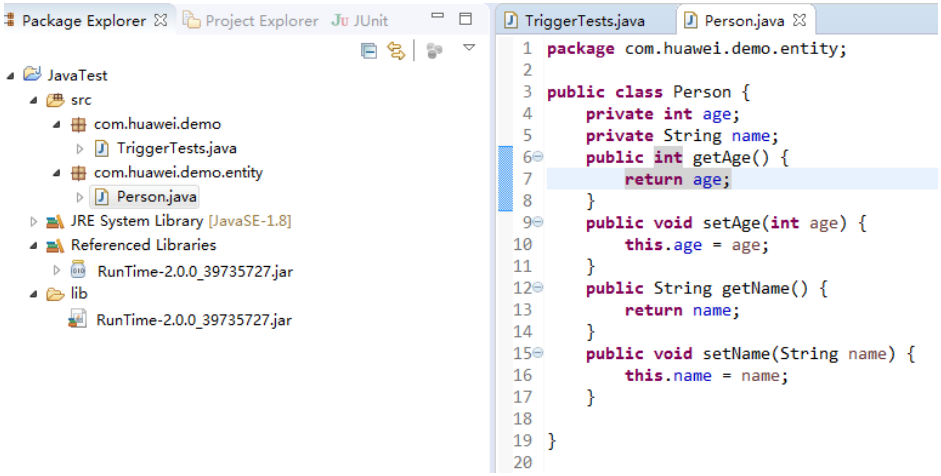
Figure 5-9 Accessing the API URL

Change the handler to **com.huawei.demo.TriggerTests.smnTest** and change the event template to **smn-event-template**.

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

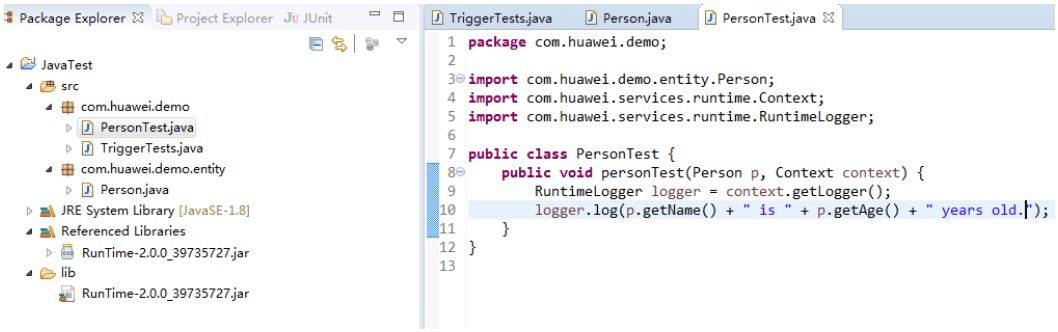
Create a Person class in the project, as shown in **Figure 5-10**.

Figure 5-10 Creating a Person class



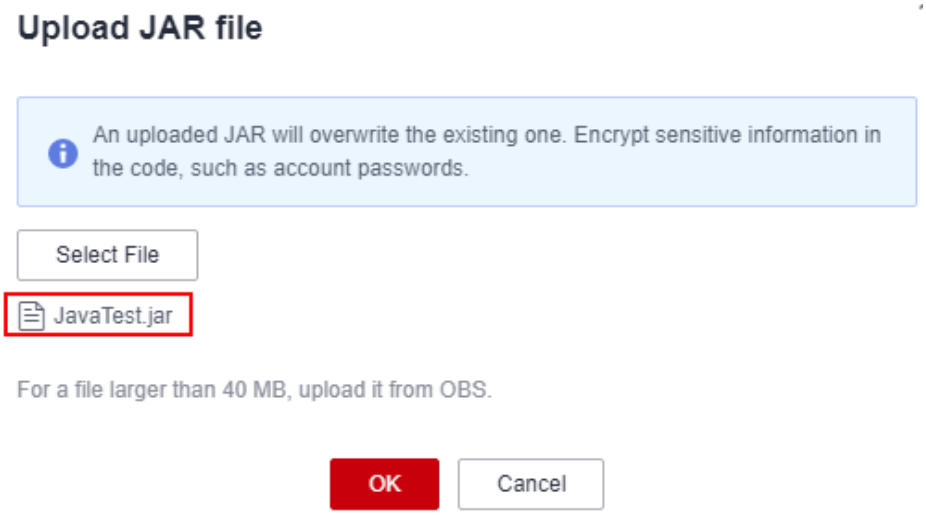
Create a class named **PersonTest.java**, and add a handler function to the class, as shown in **Figure 5-11**.

Figure 5-11 Creating a class named PersonTest.java



Upload the exported JAR file on the function details page, change the handler to **com.huawei.demo.PersonTest.personTest**, and save the change.

Figure 5-12 Uploading a JAR file



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

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

----End

Execution Result

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

Table 5-17 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.

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.

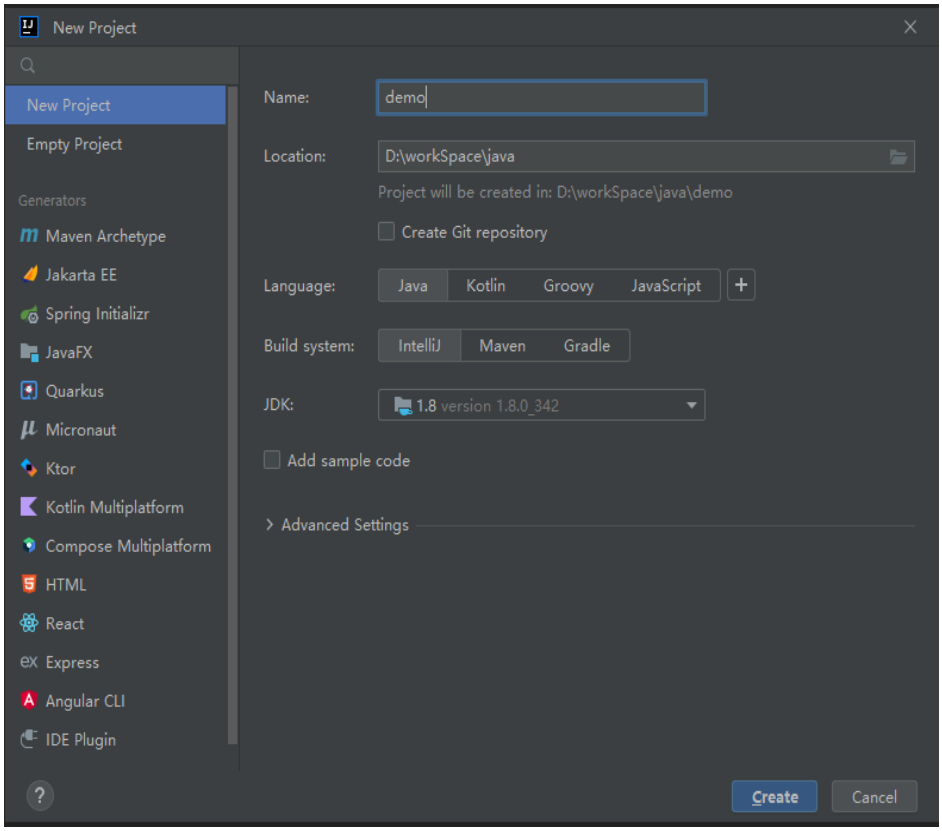
5.1.2 Developing Functions in Java (Using an IDEA Java Project)

Perform the following procedure to develop a Java function:

Step 1 Create a function project.

1. Configure the IDEA and create a Java project named **JavaTest**, as shown in Figure 5-13.

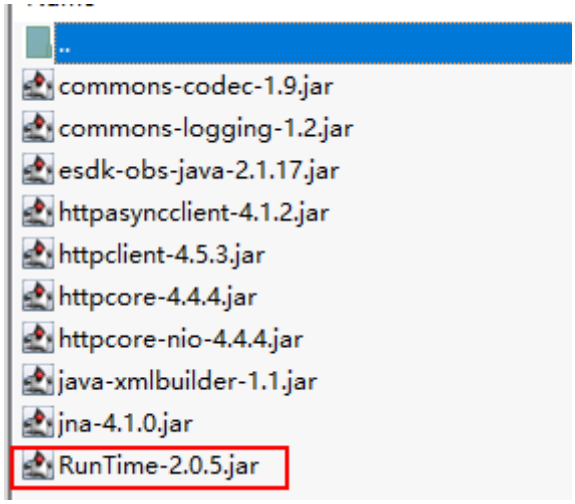
   Figure 5-13 Creating a project

2. Add dependencies to the project.

   Download the **Java SDK** to a local development environment, and decompress the SDK package, as shown in Figure 5-14.

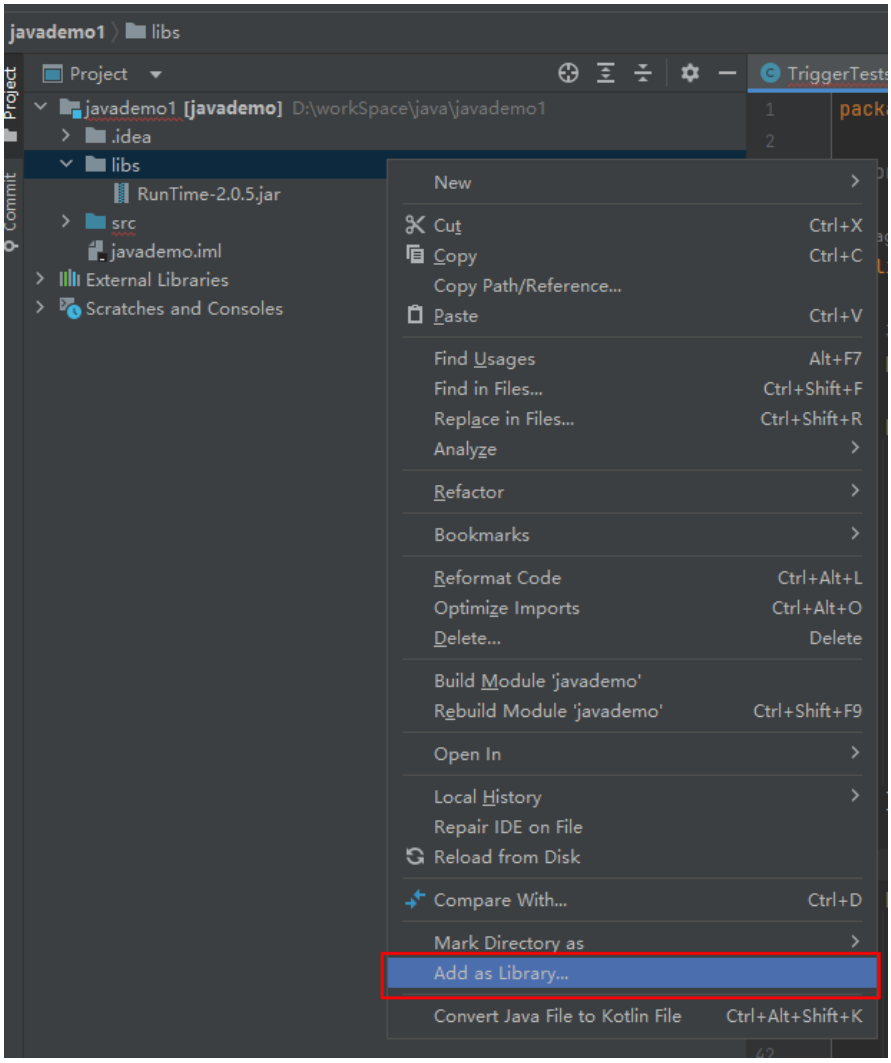
Figure 5-14 Decompressing the downloaded SDK



3. Configure dependencies.

   Create a folder named **lib** in the project directory, copy the **Runtime2.0.5.jar** file and other required dependencies to the **lib** folder, and add the JAR files as the dependencies of the project, as shown in **Figure 5-15**.

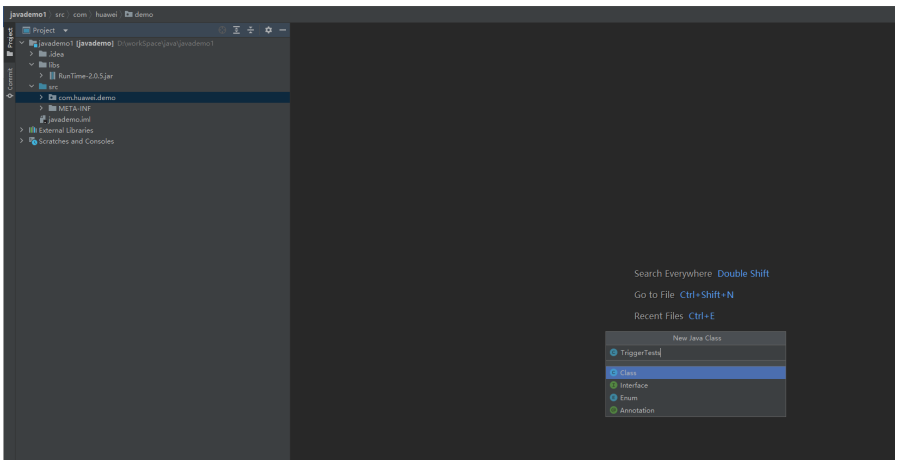
Figure 5-15 Configuring dependencies



Step 2 Create a function.

1. Create a package named **com.huawei.demo**, and then create a class named **TriggerTests** under the package, as shown in **Figure 5-16**.

Figure 5-16 Creating the TriggerTests class



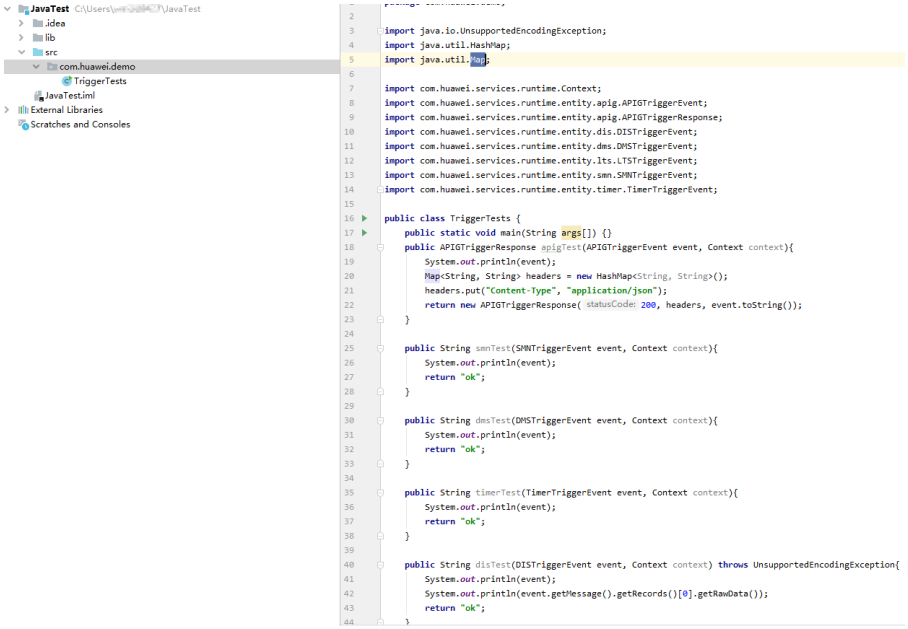
2. Define the function handler in **TriggerTests.java**, as shown in **Figure 5-17**.

```
package com.huawei.demo;

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

import com.huawei.services.runtime.Context;
import com.huawei.services.runtime.entity.apig.APIGTriggerEvent;
import com.huawei.services.runtime.entity.apig.APIGTriggerResponse;
import com.huawei.services.runtime.entity.dis.DISTriggerEvent;
import com.huawei.services.runtime.entity.dms.DMSTriggerEvent;
import com.huawei.services.runtime.entity.lts.LTSTriggerEvent;
import com.huawei.services.runtime.entity.smn.SMNTriggerEvent;
import com.huawei.services.runtime.entity.timer.TimerTriggerEvent;

public class TriggerTests {
    public static void main(String args[]) {}
    public APIGTriggerResponse apigTest(APIGTriggerEvent event, Context context){
        System.out.println(event);
        Map<String, String> headers = new HashMap<String, String>();
        headers.put("Content-Type", "application/json");
        return new APIGTriggerResponse(200, headers, event.toString());
    }

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

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

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

    public String disTest(DISTriggerEvent event, Context context) throws
UnsupportedEncodingException{
        System.out.println(event);
        System.out.println(event.getMessage().getRecords()[0].getRawData());
        return "ok";
    }

    public String ltsTest(LTSTriggerEvent event, Context context) throws
UnsupportedEncodingException {
        System.out.println(event);
        event.getLts().getData();
        System.out.println("raw data: " + event.getLts().getRawData());
        return "ok";
    }
}
```

NOTE

A common Java project needs to be compiled using artifacts, and a main function needs to be defined.

Figure 5-17 Defining the function handler



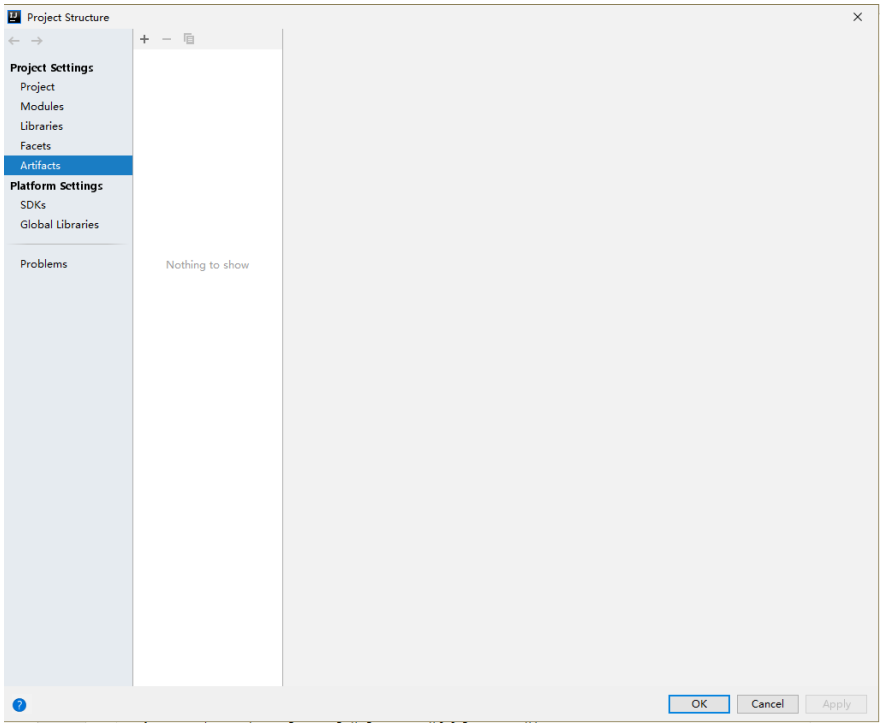
NOTE

For details about the constraints for the APIG event source, see **Base64 Decoding and Response Structure**.

Step 3 Pack the project file.

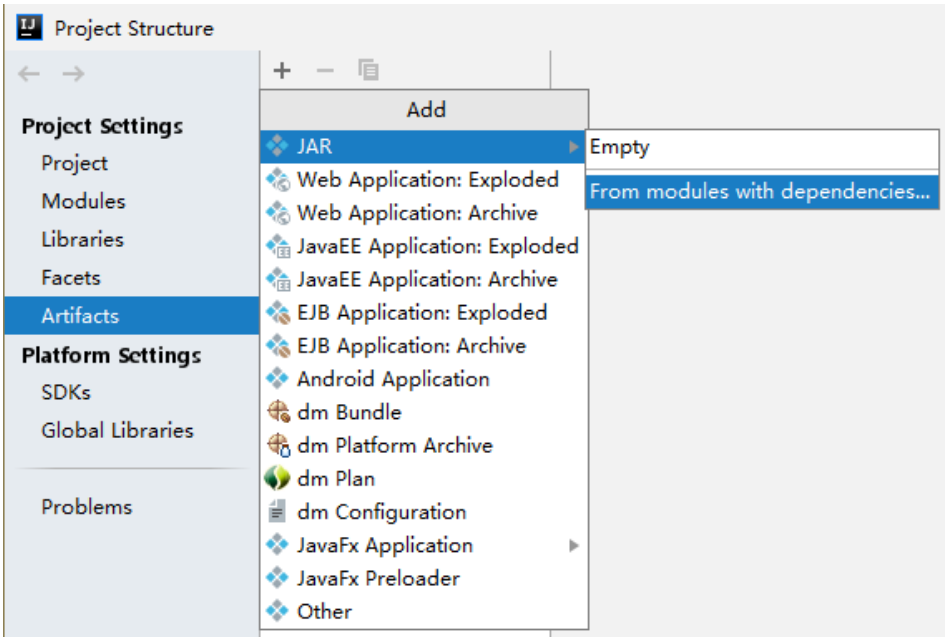
1. Choose **File** > **Project Structure**. The **Project Structure** page is displayed, as shown in **Figure 5-18**.

Figure 5-18 Going to the Project Structure page



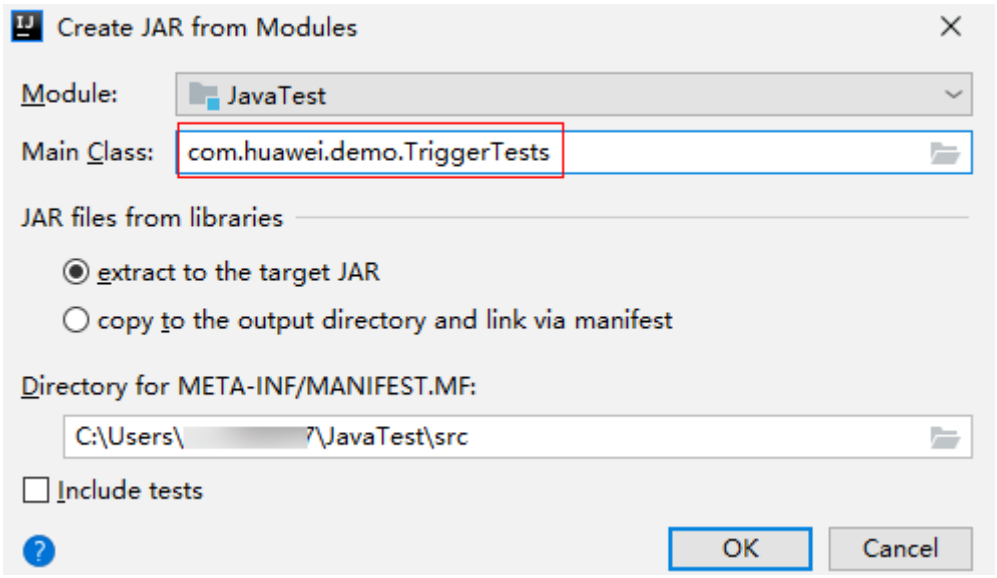
2. Choose **Artifacts** and click **+** to add artifacts, as shown in **Figure 5-19**.

Figure 5-19 Adding artifacts



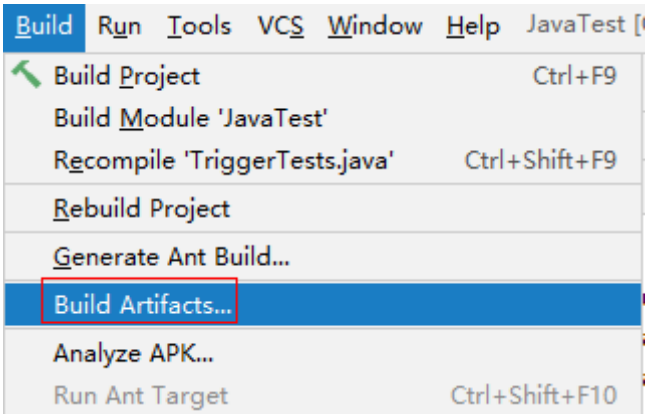
3. Add a main class, as shown in **Figure 5-20**.

Figure 5-20 Adding a main class



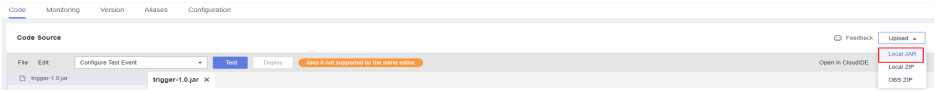
4. Choose **Build** > **Build Artifacts** to compile the JAR file, as shown in **Figure 5-21**.

Figure 5-21 Choosing Build Artifacts to compile the JAR file



Step 4 Log in to the FunctionGraph console, create a Java function, and upload the JAR file, as shown in **Figure 5-22**.

Figure 5-22 Uploading a JAR file



Step 5 Test the function.

1. Create a test event.

   Select **timer-event-template** from the **Event Template** drop-down list and click **Create**.

2. Click **Test**.

   The function execution result consists of three parts: function output (returned by **callback**), summary, and log (output by using the **System.out.println()** method).

----End

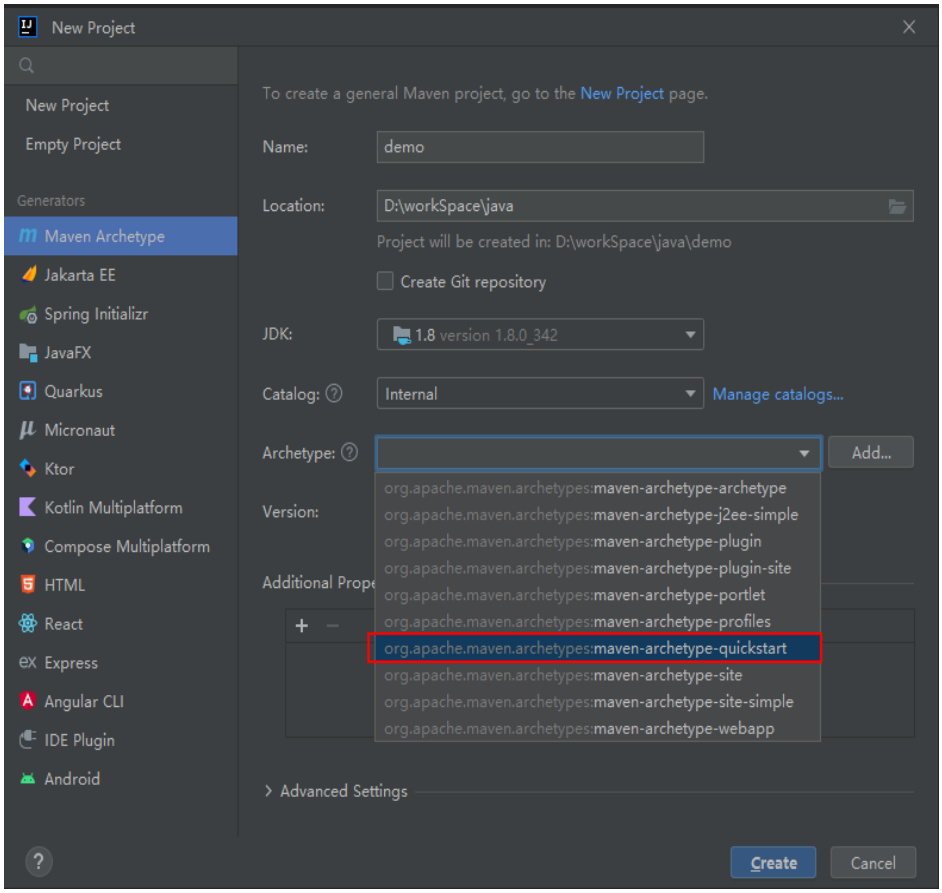
5.1.3 Developing Functions in Java (Using an IDEA Maven Project)

Perform the following procedure to develop a Java function:

Step 1 Create a function project.

1. Configure the IDEA and create a Maven project, as shown in **Figure 5-23**.

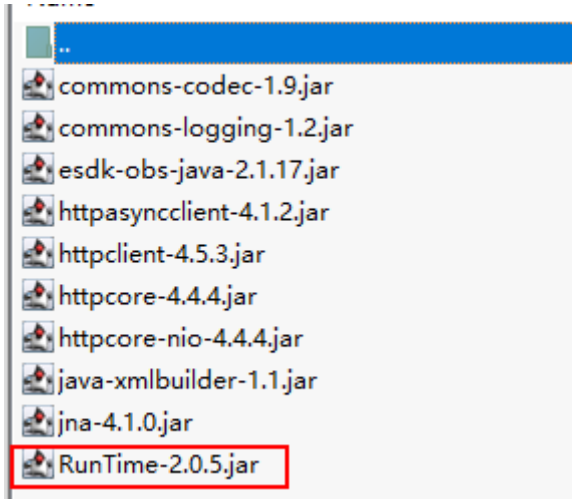
Figure 5-23 Creating a project



2. Add dependencies to the project.

   Download the **Java SDK** to a local development environment, and decompress the SDK package, as shown in **Figure 5-24**.

Figure 5-24 Decompressing the downloaded SDK



3. Copy **Runtime-2.0.5.jar** in the package to a local directory, for example, **d:\\runtime**. Then, run the following command in the CLI to install the runtime to the local Maven repository:

   ```
   mvn install:install-file -Dfile=d:\runtime\RunTime-2.0.5.jar -DgroupId=RunTime -DartifactId=RunTime -Dversion=2.0.5 -Dpackaging=jar
   ```


4. Configure the dependency in the **pom.xml** file.

```
<dependency>
<groupId>Runtime</groupId>
<artifactId>Runtime</artifactId>
<version>2.0.5</version>
</dependency>
```

5. Configure other dependencies. The following uses the OBS dependency as an example.

```
<dependency>
<groupId>com.huaweicloud</groupId>
<artifactId>esdk-obs-java</artifactId>
<version>3.21.4</version>
</dependency>
```

6. Add a plug-in to **pom.xml** to pack the code and dependencies together.

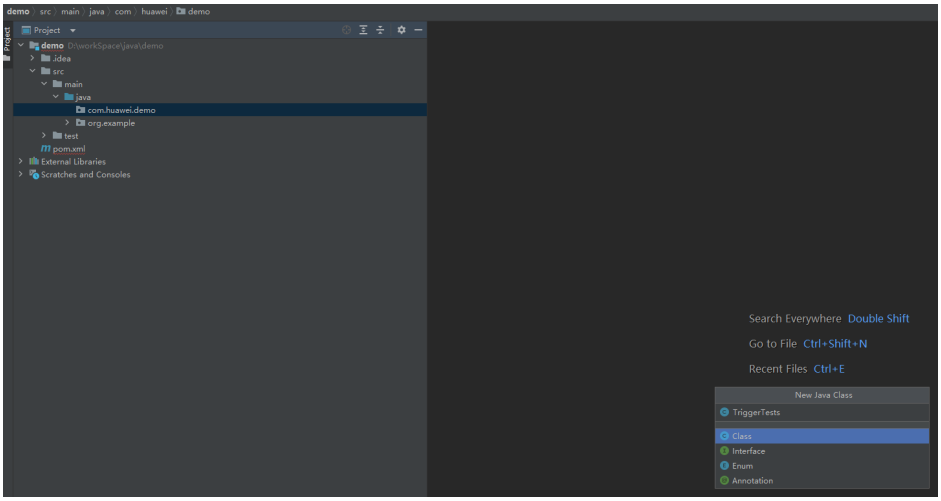
```
<build>
 <plugins>
  <plugin>
   <artifactId>maven-assembly-plugin</artifactId>
   <configuration>
    <descriptorRefs>
     <descriptorRef>jar-with-dependencies</descriptorRef>
    </descriptorRefs>
    <archive>
     <manifest>
      <mainClass>com.huawei.demo.TriggerTests</mainClass>
     </manifest>
    </archive>
    <finalName>${project.name}</finalName>
   </configuration>
   <executions>
    <execution>
     <id>make-assembly</id>
     <phase>package</phase>
     <goals><goal>single</goal>
     </goals>
    </execution>
   </executions>
  </plugin>
 </plugins>
</build>
```

Replace *mainClass* with the actual class.

Step 2 Create a function.

1. Create a package named **com.huawei.demo**, and then create a class named **TriggerTests** under the package, as shown in **Figure 5-25**.

Figure 5-25 Creating the TriggerTests class



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

```
package com.huawei.demo;

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

import com.huawei.services.runtime.Context;
import com.huawei.services.runtime.entity.apig.APIGTriggerEvent;
import com.huawei.services.runtime.entity.apig.APIGTriggerResponse;
import com.huawei.services.runtime.entity.dis.DISTriggerEvent;
import com.huawei.services.runtime.entity.dms.DMSTriggerEvent;
import com.huawei.services.runtime.entity.lts.LTSTriggerEvent;
import com.huawei.services.runtime.entity.smn.SMNTriggerEvent;
import com.huawei.services.runtime.entity.timer.TimerTriggerEvent;

public class TriggerTests {
    public APIGTriggerResponse apigTest(APIGTriggerEvent event, Context context){
        System.out.println(event);
        Map<String, String> headers = new HashMap<String, String>();
        headers.put("Content-Type", "application/json");
        return new APIGTriggerResponse(200, headers, event.toString());
    }

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

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

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

    public String disTest(DISTriggerEvent event, Context context) throws
UnsupportedEncodingException{
        System.out.println(event);
        System.out.println(event.getMessage().getRecords()[0].getRawData());
        return "ok";
    }

    public String ltsTest(LTSTriggerEvent event, Context context) throws
UnsupportedEncodingException {
```



```
        System.out.println(event);
        event.getLts().getData();
        System.out.println("raw data: " + event.getLts().getRawData());
        return "ok";
    }
}
```

NOTE

For details about the constraints for the APIG event source, see **Base64 Decoding and Response Structure**.

Step 3 Compile and pack the project file.

Run the following command to compile and pack the project file:

```
mvn package assembly:single
```

After compilation is complete, the **demo-jar-with-dependencies.jar** file is generated in the target directory.

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

Step 5 Test the function.

1. Create a test event.

   Select **timer-event-template** from the **Event Template** drop-down list and click **Create**.

2. Click **Test**.

   The function execution result consists of three parts: function output (returned by **callback**), summary, and log (output by using the **System.out.println()** method).

----End

5.2 Java Template

```
package com.huawei.demo;
import com.huawei.services.runtime.Context;
import com.huawei.services.runtime.entity.apig.APIGTriggerEvent;
import com.huawei.services.runtime.entity.apig.APIGTriggerResponse;
import com.huawei.services.runtime.entity.dis.DISTriggerEvent;
import com.huawei.services.runtime.entity.dms.DMSTriggerEvent;
import com.huawei.services.runtime.entity.lts.LTSTriggerEvent;
import com.huawei.services.runtime.entity.smn.SMNTriggerEvent;
import com.huawei.services.runtime.entity.timer.TimerTriggerEvent;
import java.io.UnsupportedEncodingException;
import java.util.HashMap;
import java.util.Map;
public class TriggerTests {
  public APIGTriggerResponse apigTest(APIGTriggerEvent event, Context context) {
    System.out.println(event);
    Map<String, String> headers = new HashMap<>();
    headers.put("Content-Type", "application/json");
    return new APIGTriggerResponse(200, headers, event.toString());
  }
  public String smnTest(SMNTriggerEvent event, Context context) {
    System.out.println(event);
    return "ok";
  }
  public String dmsTest(DMSTriggerEvent event, Context context) {
    System.out.println(event);
```



```
    return "ok";
  }
  public String timerTest(TimerTriggerEvent event, Context context) {
    System.out.println(event);
    return "ok";
  }
  public String disTest(DISTriggerEvent event, Context context) throws UnsupportedEncodingException {
    System.out.println(event);
    System.out.println(event.getMessage().getRecords()[0].getRawData());
    return "ok";
  }
  public String ltsTest(LTSTriggerEvent event, Context context) throws UnsupportedEncodingException {
    System.out.println(event);
    System.out.println("raw data: " + event.getLts().getRawData());
    return "ok";
  }
}
```

5.3 Creating a Dependency

You are advised to create function dependencies in Huawei Cloud EulerOS 2.0. If other OSs are used, an error may occur due to underlying dependent libraries. For example, the dynamic link library cannot be found.

NOTE

- If the modules to be installed need dependencies such as **.dll**, **.so**, and **.a**, archive them to a **.zip** package.

When you compile a function using Java, dependencies need to be compiled locally. For details about how to add a dependency to a Java function, see **Developing Functions in Java (Using an IDEA Java Project)**.

6 Go

6.1 Developing an Event Function

Function Syntax

Syntax for creating a handler function in Go:

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

- **Handler**: name of the handler function.
- **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

The Go SDK provides event, context, and logging APIs. **Download the Go SDK (Go SDK.sha256)**.

- Event APIs

  Event structure definitions are added to the Go SDK. Currently, DIS, DDS, SMN, Timer, and APIG triggers are supported. The definitions make coding much simpler when triggers are required.

  a. **APIG Trigger Field Description**

  i. APIGTriggerEvent fields

  Table 6-1 APIGTriggerEvent fields

Field Name	Description
IsBase64Encoded	Whether the body of an event is encoded using Base64.
HttpMethod	HTTP request method.

Field Name	Description
Path	HTTP request path.
Body	HTTP request body.
PathParameters	All path parameters.
RequestContext	API Gateway configurations (**APIGRequestContext object**).
Headers	HTTP request header.
QueryStringParameters	Query parameters.
UserData	User data set in the APIG custom authorizer.

Table 6-2 APIGRequestContext fields

Field Name	Description
ApiId	API ID.
RequestId	API request ID.
Stage	Name of the environment in which an API has been published.

ii. APIGTriggerResponse fields

Table 6-3 APIGTriggerResponse fields

Field Name	Description
Body	Message body.
Headers	HTTP response header to be returned.
StatusCode	HTTP status code. Type: **int**.
IsBase64Encoded	Whether the body has been encoded using Base64. Type: **bool**.

NOTE

APIGTriggerEvent provides the GetRawBody() method to obtain the body decoded using Base64. APIGTriggerResponse provides the SetBase64EncodedBody() method to set the body encoded using Base64.

b. **DIS Trigger Field Description**

Table 6-4 DISTriggerEvent fields

Field Name	Description
ShardID	Partition ID.
Message	DIS message body (**DISMessage structure**).
Tag	Function version.
StreamName	Stream name.

Table 6-5 DISMessage fields

Field Name	Description
NextPartitionCursor	Next partition cursor.
Records	Message records (**DISRecord structure**).
MillisBehindLatest	Reserved parameter.

Table 6-6 DISRecord fields

Field Name	Description
PartitionKey	Data partition.
Data	Data.
SequenceNumber	Sequence number (ID of each record).

c. **Kafka Trigger Field Description**

Table 6-7 KAFKATriggerEvent fields

Field Name	Description
InstanceId	Instance ID.
Records	Message records (**Table 6-8**).
TriggerType	Trigger type (Kafka).
Region	region

Field Name	Description
EventTime	Time when an event occurred (seconds).
EventVersion	Event version.

Table 6-8 KAFKARecord parameters

Field Name	Description
Messages	DMS message body.
TopicId	Topic ID.

d. **SMN Trigger Field Description**

Table 6-9 SMNTriggerEvent fields

Field Name	Description
Record	Message records (**SMNRecord structure**).

Table 6-10 SMNRecord fields

Field Name	Description
EventVersion	Event version. (Currently, the version is **1.0**.)
EventSubscriptionUrn	Subscription Uniform Resource Name (URN).
EventSource	Event source.
Smn	Message body (**SMNBody structure**).

Table 6-11 SMNBody fields

Field Name	Description
TopicUrn	Topic URN.
TimeStamp	Message timestamp.
MessageAtrributes	Message attribute set.

Field Name	Description
Message	Message body.
Type	Message format.
MessageId	Message ID.
Subject	Message topic.

e. **Timer Trigger Field Description**

Table 6-12 TimerTriggerEvent fields

Field Name	Description
Version	Version. (Currently, the version is **v1.0**.)
Time	Current time.
TriggerType	Trigger type (Timer).
TriggerName	Trigger name.
UserEvent	Additional information about the trigger.

NOTE

1. When using an APIG trigger, set the first parameter of the handler function (for example, **handler**) to **handler(APIGTriggerEvent event, Context context)**. For details about the constraints, see **Base64 Decoding and Response Structure**.
2. The preceding TriggerEvent methods have corresponding **set** methods, which are recommended for local debugging. DIS and LTS triggers have getRawData() methods, but do not have setRawData() methods.

- 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 6-13 describes the context APIs provided by FunctionGraph.

Table 6-13 Context methods

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

Method	Description
getAccessKey( )	Obtains the AK (valid for 24 hours) with 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) with 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( )	CPU usage of a function.
getPackage( )	Obtains a function group, that is, an app.

Method	Description
getToken( )	Obtains the token (valid for 24 hours) with 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.
getAlias	Obtains function alias.

NOTICE

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

- Table 6-14 describes the logging API provided in the Go SDK.

Table 6-14 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**.

Developing a Go Function

Log in to the Linux server where the Go 1.x SDK has been installed. (Currently, Ubuntu 14.04, Ubuntu 16.04, SUSE 11.3, SUSE 12.0 and SUSE 12.1 are supported.)

- If the Go version (1.11.1 or later) supports go mod, perform the following steps to compile and package code:

Step 1 Create a temporary directory, for example, **/home/fssgo**, decompress the **Go SDK** of FunctionGraph to the created directory, and enable the go module function.

$ mkdir -p /home/fssgo

$ unzip functiongraph-go-runtime-sdk-1.0.1.zip -d /home/fssgo

$ export GO111MODULE="on"

Step 2 Generate the **go.mod** file in the **/home/fssgo** directory. Assume that the module name is **test**:

$ go mod init test

Step 3 Edit the **go.mod** file in the **/home/fssgo** directory as required (that is, add the content in bold).

```
module test

go 1.14

require (
    huaweicloud.com/go-runtime v0.0.0-00010101000000-000000000000
)

replace (
    huaweicloud.com/go-runtime => ./go-runtime
)
```

Step 4 Create a function file under the **/home/fssgo** directory and implement the following interface:

func Handler(payload []byte, ctx context.RuntimeContext) (interface{}, error)

In this interface, **payload** is the body of a client request, and **ctx** is the runtime context object provided for executing a function. For more information about the methods, see **Table 6-13**. The following uses **test.go** as an example.

```
package main

import (
    "fmt"
    "huaweicloud.com/go-runtime/go-api/context"
    "huaweicloud.com/go-runtime/pkg/runtime"
    "huaweicloud.com/go-runtime/events/apig"
    "huaweicloud.com/go-runtime/events/cts"
    "huaweicloud.com/go-runtime/events/dds"
    "huaweicloud.com/go-runtime/events/dis"
    "huaweicloud.com/go-runtime/events/kafka"
    "huaweicloud.com/go-runtime/events/lts"
    "huaweicloud.com/go-runtime/events/smn"
    "huaweicloud.com/go-runtime/events/timer"
    "encoding/json"
)

func ApigTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var apigEvent apig.APIGTriggerEvent
    err := json.Unmarshal(payload, &apigEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", apigEvent.String())
    apigResp := apig.APIGTriggerResponse{
        Body: apigEvent.String(),
        Headers: map[string]string {
            "content-type": "application/json",
        },
        StatusCode: 200,
    }
    return apigResp, nil
}

func CtsTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var ctsEvent cts.CTSTriggerEvent
```



```
    err := json.Unmarshal(payload, &ctsEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", ctsEvent.String())
    return "ok", nil
}

func DdsTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var ddsEvent dds.DDSTriggerEvent
    err := json.Unmarshal(payload, &ddsEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", ddsEvent.String())
    return "ok", nil
}

func DisTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var disEvent dis.DISTriggerEvent
    err := json.Unmarshal(payload, &disEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", disEvent.String())
    return "ok", nil
}

func KafkaTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var kafkaEvent kafka.KAFKATriggerEvent
    err := json.Unmarshal(payload, &kafkaEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", kafkaEvent.String())
    return "ok", nil
}

func LtsTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var ltsEvent lts.LTSTriggerEvent
    err := json.Unmarshal(payload, &ltsEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", ltsEvent.String())
    return "ok", nil
}

func SmnTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var smnEvent smn.SMNTriggerEvent
    err := json.Unmarshal(payload, &smnEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", smnEvent.String())
    return "ok", nil
}

func TimerTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var timerEvent timer.TimerTriggerEvent
    err := json.Unmarshal(payload, &timerEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
```



```
    return "invalid data", err
  }
  return timerEvent.String(), nil
}

func main() {
  runtime.Register(ApigTest)
}
```

NOTICE

1. If the **error** parameter returned by a function is not **nil**, the function execution fails.
2. If the **error** parameter returned by a function is **nil**, FunctionGraph supports only the following types of values:

   nil: The HTTP response body is empty.

   []byte: The content in this byte array is the body of an HTTP response.

   string: The content in this string is the body of an HTTP response.

   Other: FunctionGraph returns a value for JSON encoding, and uses the encoded object as the body of an HTTP response. The **Content-Type** header of the HTTP response is set to **application/json**.
3. **The preceding example uses an APIG trigger as an example. For other trigger types, you need to modify the content of the main function. For example, change the CTS trigger to runtime.Register(CtsTest). Currently, only one entry can be registered.**
4. For details about the constraints for the APIG event source, see **Base64 Decoding and Response Structure**.

Step 5 Compile and package the function code.

After completing the function code, compile and package it as follows:

1. Compile the code.

   $ cd /home/fssgo

   $ go build -o handler test.go

   NOTE

   The handler can be customized, which is used as the function entry.
2. Package the code.

   $ zip fss_examples_go1.x.zip handler

----End

- If the Go version (earlier than 1.11.1) does not support go mod, perform the following steps to compile and package code:

Step 1 Create a temporary directory, for example, **/home/fssgo/src/huaweicloud.com**, and decompress the **Go SDK** to the created directory.

$ mkdir -p /home/fssgo/src/huaweicloud.com

$ unzip functiongraph-go-runtime-sdk-1.0.1.zip -d /home/fssgo/src/huaweicloud.com

Step 2 Create a function file under the **/home/fssgo/src** directory and implement the following interface:

func Handler(payload []byte, ctx context.RuntimeContext) (interface{}, error)

In this interface, **payload** is the body of a client request, and **ctx** is the runtime context object provided for executing a function. For more information about the methods, see the SDK APIs. The following uses **test.go** as an example.

```
package main

import (
    "fmt"
    "huaweicloud.com/go-runtime/go-api/context"
    "huaweicloud.com/go-runtime/pkg/runtime"
    "huaweicloud.com/go-runtime/events/apig"
    "huaweicloud.com/go-runtime/events/cts"
    "huaweicloud.com/go-runtime/events/dds"
    "huaweicloud.com/go-runtime/events/dis"
    "huaweicloud.com/go-runtime/events/kafka"
    "huaweicloud.com/go-runtime/events/lts"
    "huaweicloud.com/go-runtime/events/smn"
    "huaweicloud.com/go-runtime/events/timer"
    "encoding/json"
)

func ApigTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var apigEvent apig.APIGTriggerEvent
    err := json.Unmarshal(payload, &apigEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", apigEvent.String())
    apigResp := apig.APIGTriggerResponse{
        Body: apigEvent.String(),
        Headers: map[string]string {
            "content-type": "application/json",
        },
        StatusCode: 200,
    }
    return apigResp, nil
}

func CtsTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var ctsEvent cts.CTSTriggerEvent
    err := json.Unmarshal(payload, &ctsEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", ctsEvent.String())
    return "ok", nil
}

func DdsTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var ddsEvent dds.DDSTriggerEvent
    err := json.Unmarshal(payload, &ddsEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", ddsEvent.String())
    return "ok", nil
}
```



```
func DisTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var disEvent dis.DISTriggerEvent
    err := json.Unmarshal(payload, &disEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", disEvent.String())
    return "ok", nil
}

func KafkaTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var kafkaEvent kafka.KAFKATriggerEvent
    err := json.Unmarshal(payload, &kafkaEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", kafkaEvent.String())
    return "ok", nil
}

func LtsTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var ltsEvent lts.LTSTriggerEvent
    err := json.Unmarshal(payload, &ltsEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", ltsEvent.String())
    return "ok", nil
}

func SmnTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var smnEvent smn.SMNTriggerEvent
    err := json.Unmarshal(payload, &smnEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    ctx.GetLogger().Logf("payload:%s", smnEvent.String())
    return "ok", nil
}

func TimerTest(payload []byte, ctx context.RuntimeContext) (interface{}, error) {
    var timerEvent timer.TimerTriggerEvent
    err := json.Unmarshal(payload, &timerEvent)
    if err != nil {
        fmt.Println("Unmarshal failed")
        return "invalid data", err
    }
    return timerEvent.String(), nil
}

func main() {
    runtime.Register(ApigTest)
}
```


NOTICE

1. If the **error** parameter returned by a function is not **nil**, the function execution fails.
2. If the **error** parameter returned by a function is **nil**, FunctionGraph supports only the following types of values:

   nil: The HTTP response body is empty.

   []byte: The content in this byte array is the body of an HTTP response.

   string: The content in this string is the body of an HTTP response.

   Other: FunctionGraph returns a value for JSON encoding, and uses the encoded object as the body of an HTTP response. The **Content-Type** header of the HTTP response is set to **application/json**.
3. **The preceding example uses an APIG trigger as an example. For other trigger types, you need to modify the content of the main function. For example, change the CTS trigger to runtime.Register(CtsTest). Currently, only one entry can be registered.**
4. For details about the constraints for the APIG event source, see **Base64 Decoding and Response Structure**.

Step 3 Compile and package the function code.

After completing the function code, compile and package it as follows:

1. Set environment variables **GOROOT** and **GOPATH**.

   $ export GOROOT=/usr/local/go (Assume that the Go SDK is installed under the /usr/local/go directory.)

   $ export PATH=$GOROOT/bin:$PATH

   $ export GOPATH=/home/fssgo
2. Compile the function code.

   $ cd /home/fssgo

   $ go build -o handler test.go

   NOTE

   The handler can be customized, which is used as the function entry.
3. Package the code.

   $ zip fss_examples_go1.x.zip handler

- Creating a function

  Log in to the FunctionGraph console, create a Go 1.x function, and upload the **fss_examples_go1.x.zip** file.

NOTE

If you edit code in Go, package the compiled file into a ZIP file, and ensure that the name of the compiled file is consistent with the handler plugin name. For example, if the name of the binary file is **handler**, set the handler plugin name to **handler**. The handler must be consistent with that defined in **Step 3.2**.

- Testing the function

a. Create a test event.

On the function details page that is displayed, click **Configure Test Event**. Configure the test event information, as shown in **Figure 6-1**, and then click **Create**.

Figure 6-1 Configuring a test event

b. On the function details page, select the configured test event, and click **Test**.

- Executing the function

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

----End

Execution Result

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

Table 6-15 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** 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.

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.

7 C#

7.1 Developing an Event Function

7.1.1 C# Function Development

Function Syntax

NOTE

You are advised to use .NET Core 3.1.

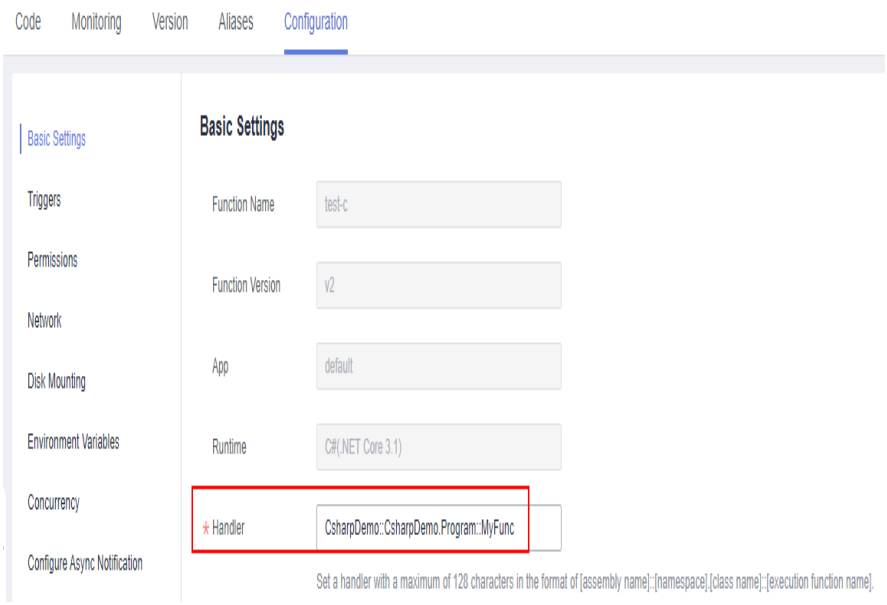
FunctionGraph supports C# (.NET Core 2.1) and C# (.NET Core 3.1).

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.

  In the navigation pane on the left of the FunctionGraph console, choose **Functions** > **Function List**. Click the name of the function to be set. On the function details page that is displayed, choose **Configuration** > **Basic Settings** and set the **Handler** parameter, as shown in **Figure 7-1**. The parameter value is **CsharpDemo::CsharpDemo.Program::MyFunc** (format: *[assembly]*::*[namespace]*.*[class name]*::*[execution function name]*).

Figure 7-1 Handler parameter



- 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.

  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 7-1 describes the provided context attributes.

Table 7-1 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	CPU usage of a function.
String Accesskey	Obtains the AK (valid for 24 hours) with 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.
String Secretkey	Obtains the SK (valid for 24 hours) with 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) with 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 7-2 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);

Execution Result

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

Table 7-3 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.

7.1.2 JSON Serialization and Deserialization

7.1.2.1 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** using .NET Core 2.1. The procedure is similar for NET Core 3.1. 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/projets /tmp/csharp/release** directory:
   ```
   mkdir -p  /tmp/csharp/projets;mkdir -p  /tmp/csharp/release
   ```
2. Run the following command to go to the **/tmp/csharp/projets/** directory:
   ```
   cd /tmp/csharp/projets/
   ```
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**.

   Upload the **HC.Serverless.Function.Common.dll**, **HC.Serverless.Function.Common.JsonSerializer.dll**, and **Newtonsoft.Json.dll** files in the package to the **/tmp/csharp/projets/** directory.
2. Create the **Class1.cs** file in the **/tmp/csharp/projets/** 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:

```
/home/tools/dotnetcore-sdk/dotnet-sdk-2.1.302-linux-x64/dotnet build /tmp/csharp/projets/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 468480 Jan 21 16:40 Newtonsoft.Json.dll
-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 ./*
   ```

7.1.2.2 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:

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

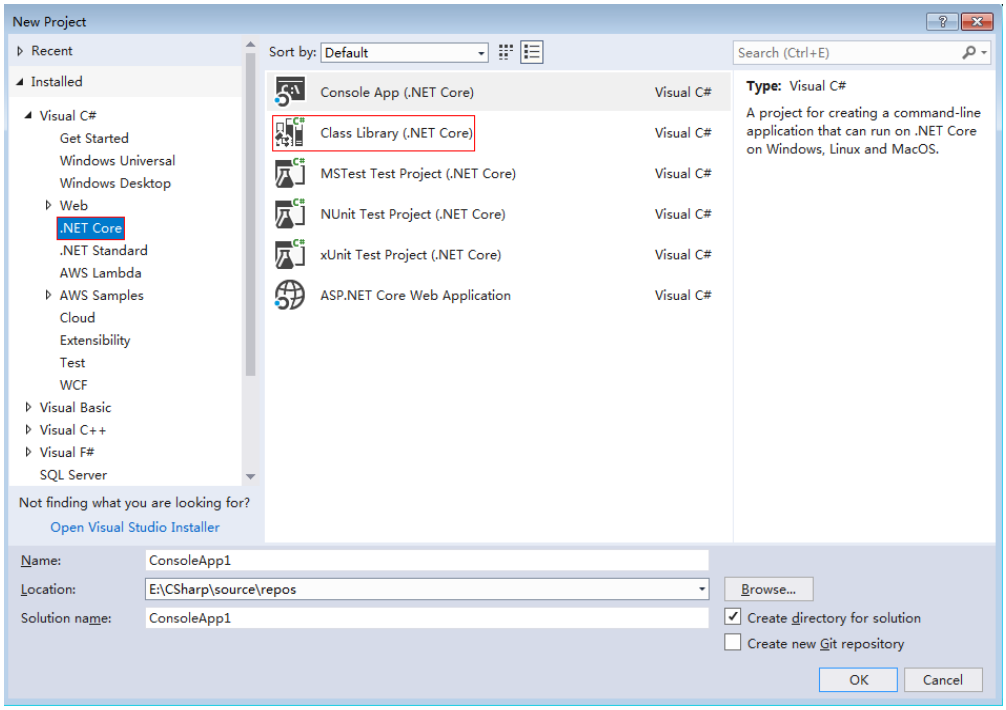
Stream Serialize<T>(T value): Serializes data to the returned response payload.

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

Creating a Project

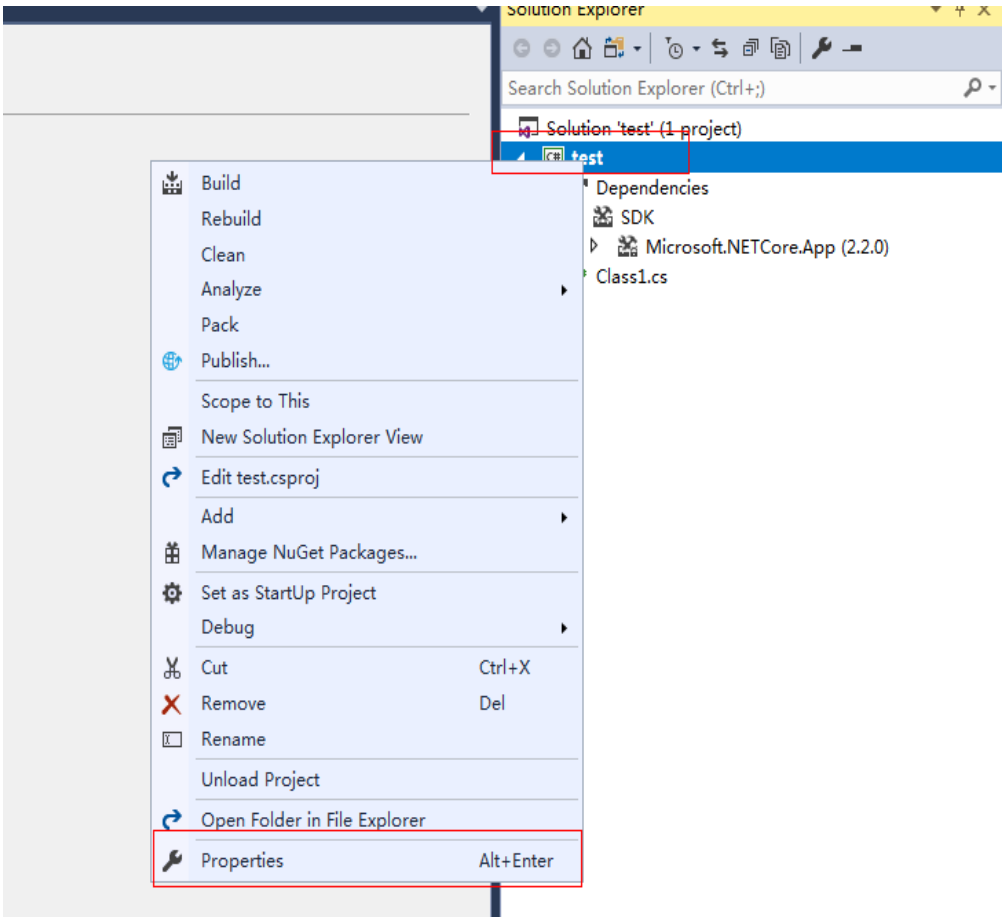
1. On the toolbar, choose **File** > **New** > **Project**, select **.NET Core**, select **Class Library (.NET Core)**, and change the project name to **test**, as shown in **Figure 7-2**.

Figure 7-2 Creating a project



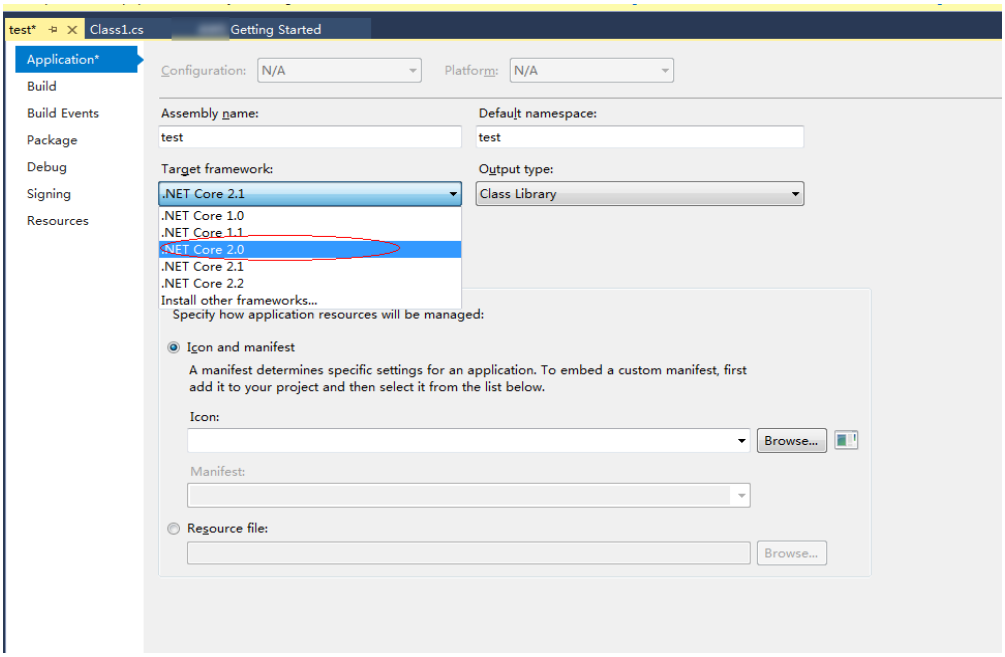
2. Select the **test** project in the navigation pane, right-click, and then choose **Properties**, as shown in **Figure 7-3**.

Figure 7-3 Properties



3. Choose **Application** and then set **Target framework** to **.NET Core 2.0**, as shown in **Figure 7-4**.

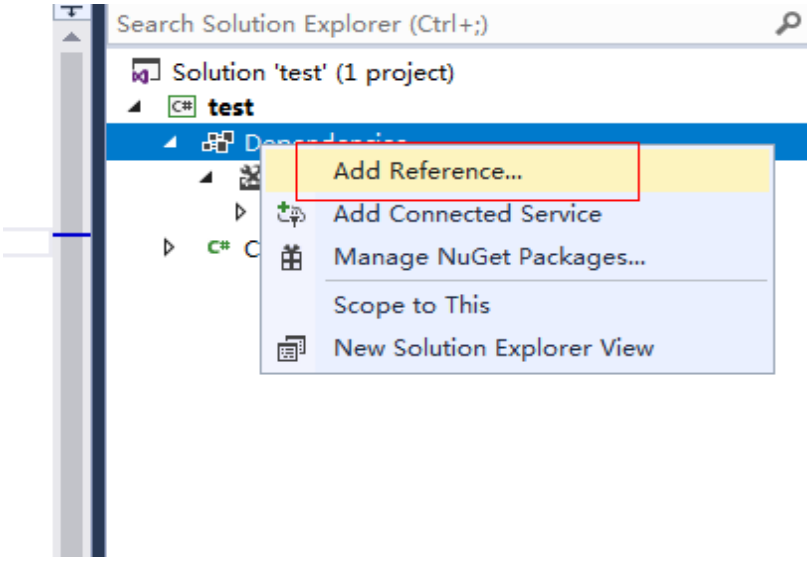
Figure 7-4 Selecting a target framework



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, as shown in **Figure 7-5**.

Figure 7-5 Adding a reference

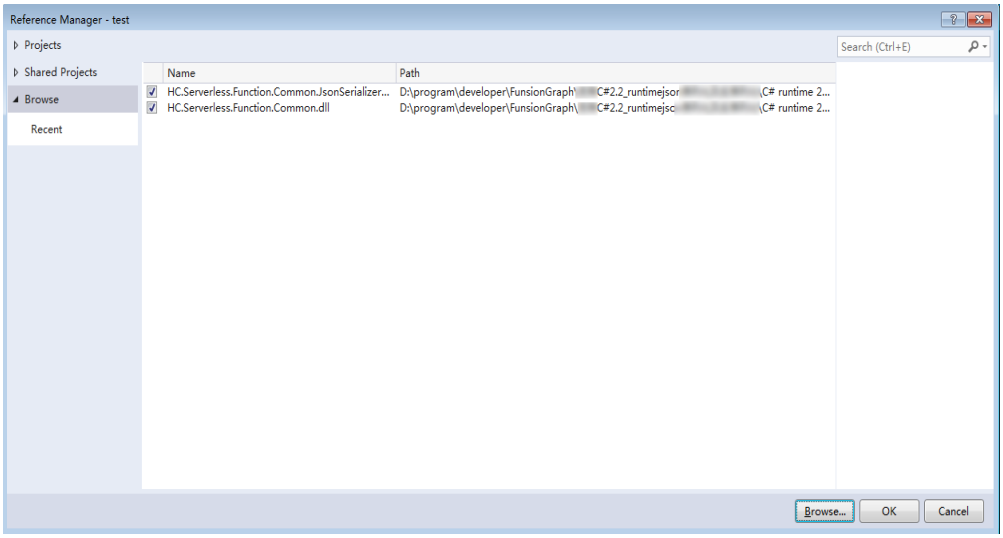


NOTE

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

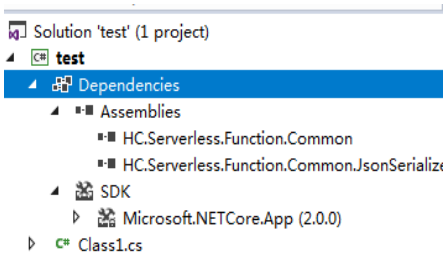
2. Choose **Browse**, click **Browse(B)**, reference the **HC.Serverless.Function.Common.dll** and **HC.Serverless.Function.Common.JsonSerializer.dll** files, and click **OK**, as shown in **Figure 7-6**.

Figure 7-6 Referencing files



3. View the references, as shown in **Figure 7-7**.

Figure 7-7 References



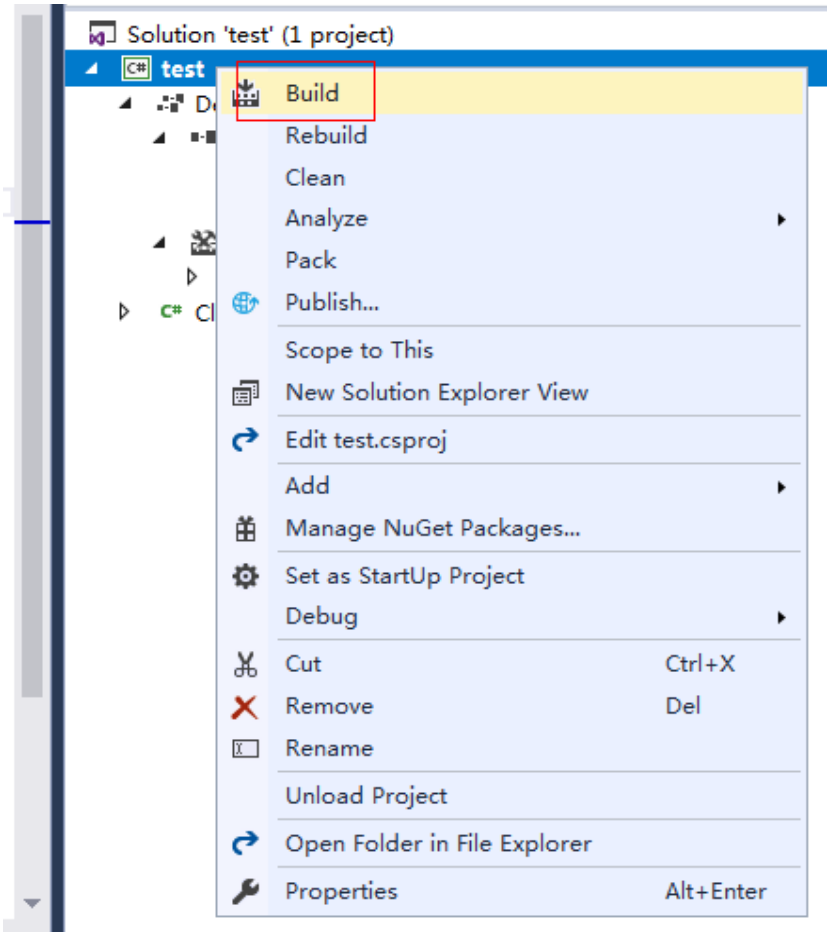
Packing Code

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 7-8**.

Figure 7-8 Build



2. Copy the path **C:\Users\xxx\source\repos\test\test\bin\Release\netcoreapp2.0** of the .dll files, as shown in **Figure 7-9**.

Figure 7-9 Path of .dll files

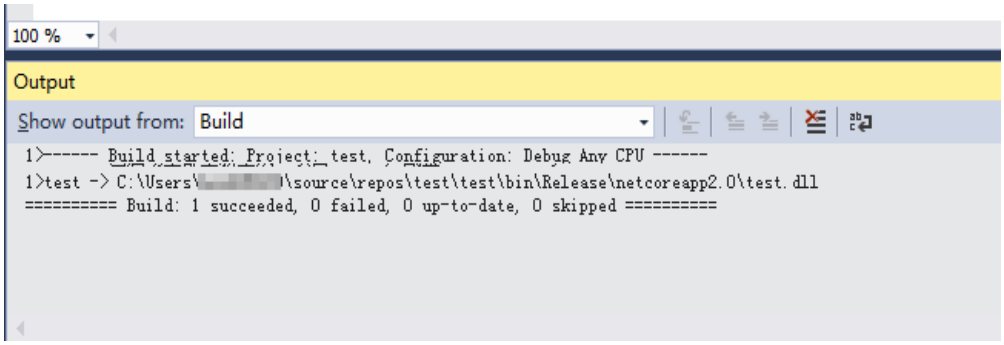
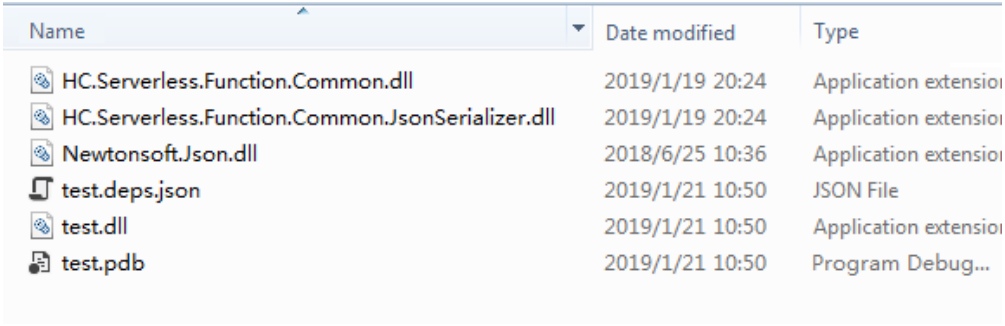


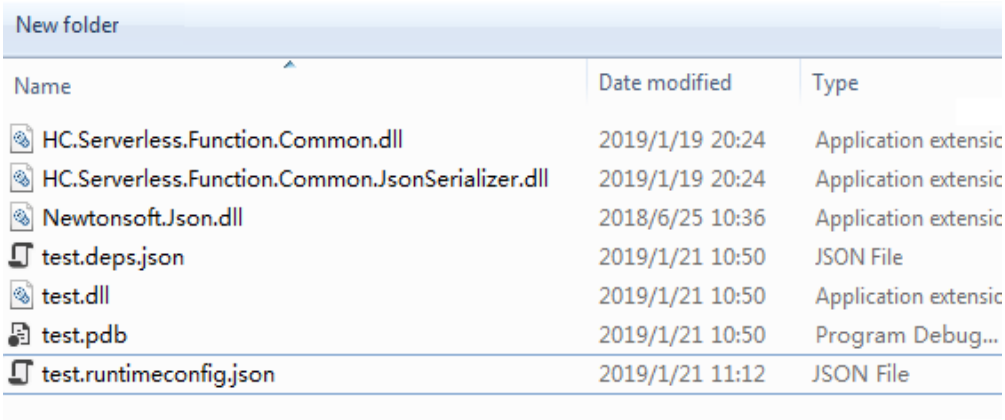
Figure 7-10 shows the files in the path.

Figure 7-10 Files



3. Create a **test.runtimeconfig.json** file in the path, as shown in **Figure 7-11**.

Figure 7-11 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.

8 PHP

8.1 Developing an Event Function

Function Syntax

Use the following syntax when creating a handler function in PHP 7.3:

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 (runtime = Php7.3)

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.

SDK APIs

The following table describes the context methods provided by FunctionGraph.

Table 8-1 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) with 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) with 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.

Method	Description
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()	CPU usage of a function.
getPackage()	Obtains a function group, that is, an app.
getToken()	Obtains the token (valid for 24 hours) with 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")
getAlias	Obtains function alias.

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 8-2 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**, **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.

8.2 Creating a Dependency

You are advised to create function dependencies in Huawei Cloud EulerOS 2.0. If other OSs are used, an error may occur due to underlying dependent libraries. For example, the dynamic link library cannot be found.

NOTE

- If the modules to be installed need dependencies such as **.dll**, **.so**, and **.a**, archive them to a **.zip** package.

Creating a Dependency for a PHP Function

EulerOS 2.9.6 is recommended.

By default, Composer and PHP 7.3 have been installed in the environment. Install Protobuf 3.19 using Composer.

Create the **composer.json** file with the following content:

```
{
    "require": {
        "google/protobuf": "^3.19"
    }
}
```


Run the following command:

```
Composer install
```

The **vendor** folder is generated with the **autoload.php**, **composer**, and **google** subfolders in the current directory.

- Linux

  Run the following command to generate a ZIP package.

  ```
  zip –rq vendor.zip vendor
  ```

- Windows

  Compress **vendor** into a ZIP file.

  If multiple dependencies need to be installed, specify them in the **composer.json** file, compress the **vendor** folder into a ZIP file and upload it.

NOTE

To use third-party dependencies downloaded using Composer in PHP project code, load the dependencies through **require "./vendor/autoload.php"**. By default, files decompressed from the uploaded ZIP package are placed in a directory at the same level as the project code.

9 Development Tools

9.1 Local Debugging with VSCode

Introduction

Huawei Cloud FunctionGraph is a Visual Studio Code (VSCode) plug-in of Huawei Cloud serverless products. With this plug-in, you can:

- Quickly create local functions.
- Run and debug local functions and deploy them to the cloud.
- Pull the function list from the cloud, call cloud functions, and upload ZIP packages to the cloud.

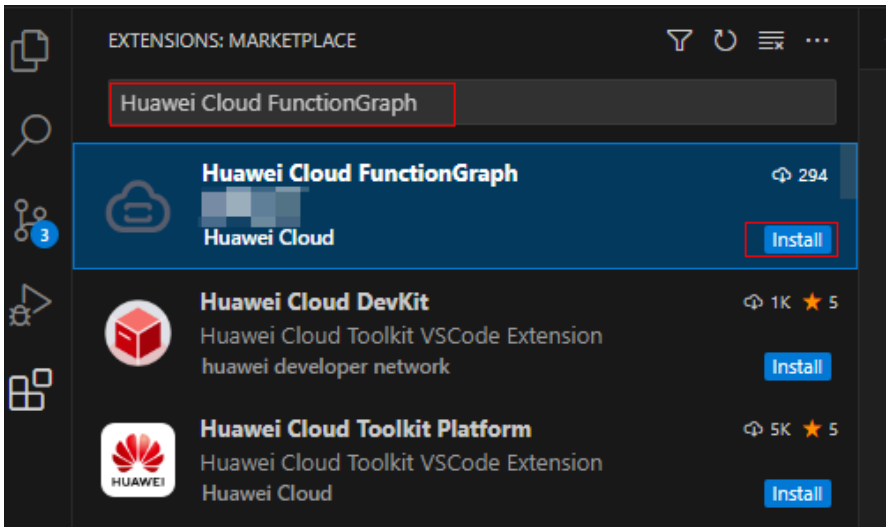
Prerequisites

You have downloaded the **VSCode tool (later than 1.63.0)** and installed it.

Installing the Plug-in

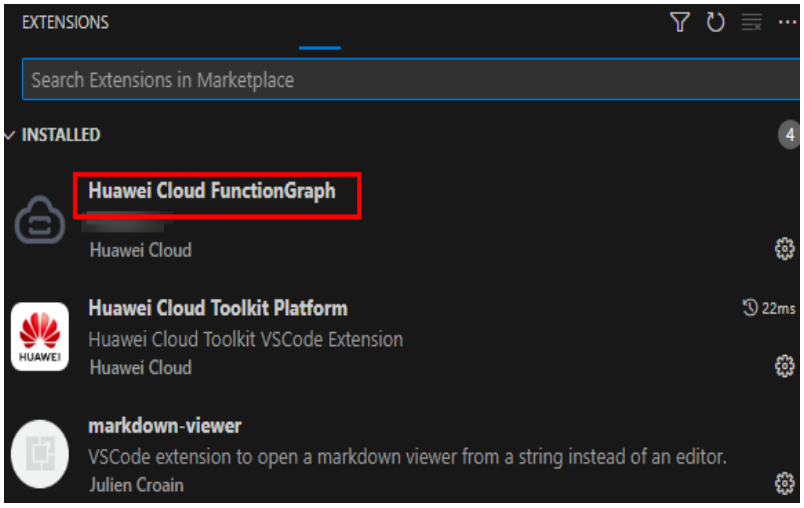
1. Open VSCode, search for **Huawei Cloud FunctionGraph** in the app store, and install it.

Figure 9-1 Searching for and installing Huawei Cloud FunctionGraph



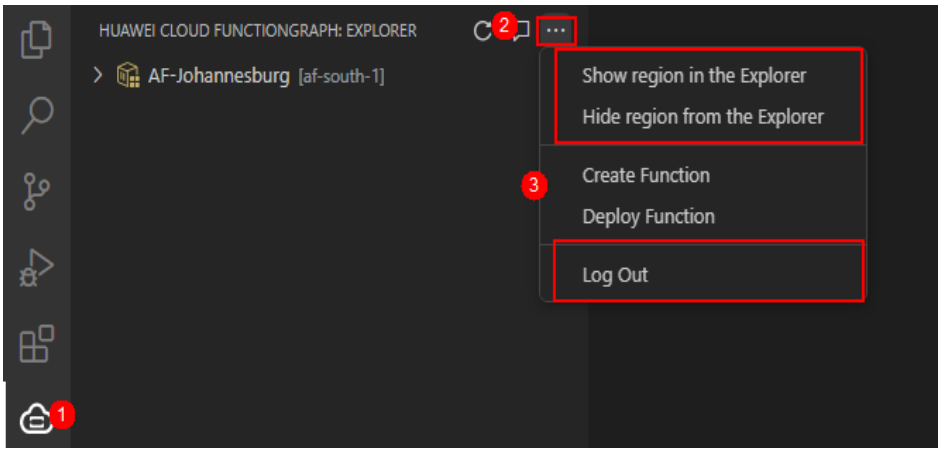
2. After the installation is successful, **Huawei Cloud FunctionGraph** is displayed in the plug-in list.

Figure 9-2 Installed plug-ins



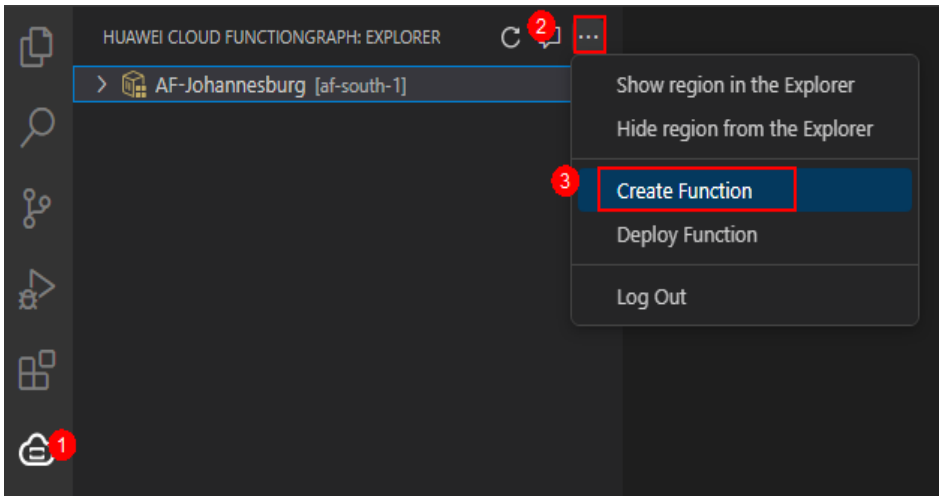
Logging In to FunctionGraph Plug-in

1. Click the **Huawei Cloud FunctionGraph** plug-in icon, click the login link, and select a login mode. If you select login with AK/SK, obtain an AK/SK. For details, see **Creating an Access Key**.
2. Select a region to view function information.
3. Show or hide desired regions, or log out of your account by referring to the following figure.
   - **Show region in the Explorer**: Show the regions where you need to perform operations.
   - **Hide region from the Explorer**: Hide the regions you do not need.
   - **Log Out**: Log out of your account.



Creating a Function

1. On the plug-in panel, select **Create Function** or press **Ctrl+Shift+p** to search for the **Create Function** command. Then, specify the runtime, template, function name, and local folder as prompted. A function is created in the specified folder.



2. After the local function is created, the handler file is automatically opened.
3. You can customize the function's configuration by modifying the automatically generated configuration file. The parameters are as follows:

```
HcCrmTemplateVersion: v2
Resources:
  Type: HC::Serverless::Function
  Properties:
    Name: functionName //Function name
    Handler: handler // Handler of the function
    Runtime: runtime // Function runtime
    CodeType: inline // Default
    CodeFileName: index.zip // Default
    CodeUrl: ""
    Description: '' // Function runtime
    MemorySize: 128 // Memory for executing the function
    Timeout: 30 // Function timeout, in seconds.
    Version: latest // Default
    Environment:
      Variables: {} // Environment variables
    InitializerHandler: "" // Function initializer
    InitializerTimeout: 0 // Initialization timeout of the function
```



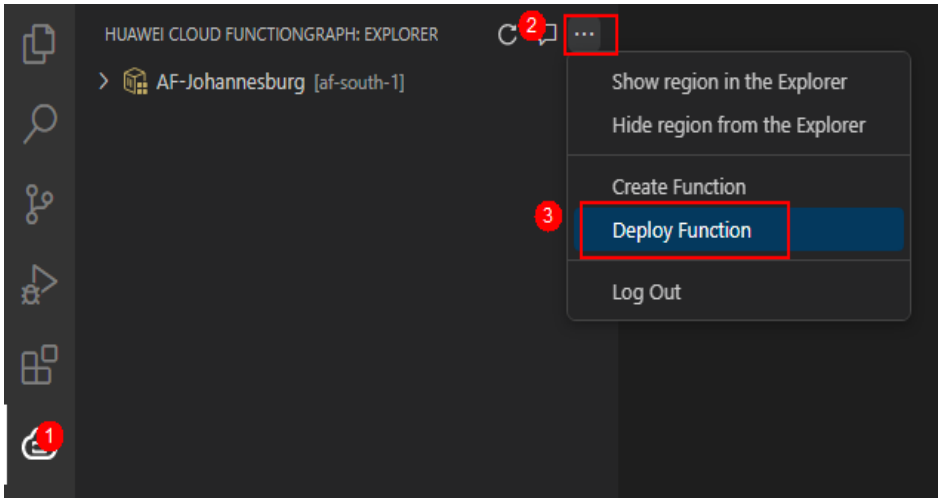
```
EnterpriseProjectId: "0" // Enterprise project
FuncType: v2
URN: "" // Function URN, which is generated after the function is downloaded.
```

Deploying a Function

- **Prerequisites**

  Ensure that the function code path is correct. The code of Node.js, Python, and PHP functions is stored in the **src** directory, and the code of other functions is stored in the root directory.

On the plug-in panel, select **Deploy Function** or press **Ctrl+Shift+p** to search for the **Deploy Function** command, and select a function and region as prompted.



- If the deployment is successful, a success message is displayed in the lower right corner of the page. Switch to the target region to view the deployment result.
- If the deployment fails, view the error log in the **Output** area and rectify the fault.

Local Debugging

1. Node.js
   - **Prerequisites**

     Node.js has been installed in the local environment.
   - **Default mode**

     Click **Local Debug** of the handler method, configure the event content, and click **Invoke** for debugging.

Figure 9-3 Clicking Local Debug

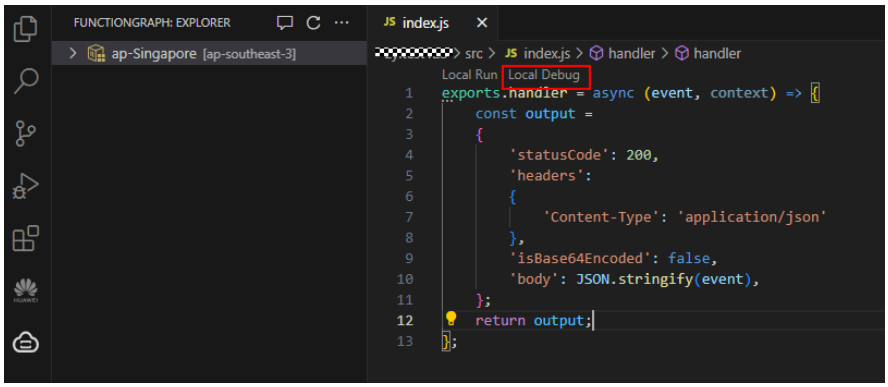
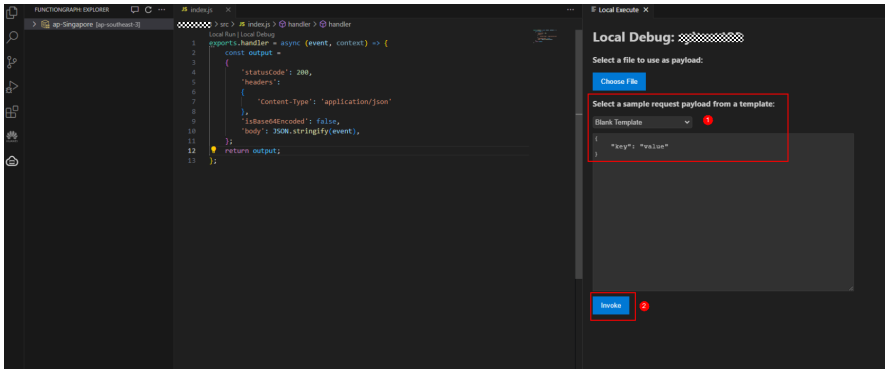


Figure 9-4 Configuring the event content



- **Debugging with VS Code**

  Create the **main.js** file in the function folder and copy the following content to this file. Click the  icon on the left. Then, click **Add Config**, select Node.js, and press **F5** to start debugging.

```
const handler = require('./index'); //Path of the function handler file. Modify it as required.
const event = { 'hello': 'world' }; //Test event. Modify it as required.
const context = {}; //Context class.
console.log(handler.handler(event, context));
```

2. Python

   - **Prerequisites**

     Python has been installed in the local environment.

   Create the **main.py** file in the function folder and copy the following content to this file. Click the  icon on the left. Then, click **Add Config**, select Python, and press **F5** to start debugging.

```
import sys
import index #Path of the function handler file. Modify it as required.

#The main method is used for debugging, and event is the selected debugging event.
if __name__ == '__main__':
....event = { 'hello': 'world' } #Test event. Modify it as required.
    context = ''
    content = index.handler(event, context)
....print('Returned value:')
    print(content)
```


3. Java

   – **Prerequisites**

     Java has been installed. VS Code supports Java testing.

   In the **test** directory of the function folder, open the **TriggerTestsTest.java** file, and click the  icon on the left. Then, click **Add Config**, select Java, and press **F5** to start debugging.

Other Functions

- **Opening in Portal**

  Right-click the target function, and choose **Open in Portal**. The function details page is displayed.

- **Executing a Cloud Function**

  a. Right-click the target function and choose **Invoke Function...** from the shortcut menu.

  b. In the **Invoke Function** panel, select the event to be passed and click **Invoke**. The function log and result are displayed in the **Output** area.

- **Downloading a Cloud Function**

  – **Prerequisites**

    You have been granted the permission (**obs:object:GetObject**) for obtaining bucket objects.

  Right-click the function you want to download and choose **Download...** from the shortcut menu. The function code is downloaded from the cloud to the specified local path, and the handler file is automatically opened.

- **Updating a Cloud Function**

  Right-click the target function, choose **Upload Function...** from the shortcut menu, and select a ZIP package to upload.

- **Deleting a Cloud Function**

  a. Right-click the function to be deleted and choose **Delete...** from the shortcut menu.

  b. In the confirmation dialog box, click **Delete** to delete the function.

- **Copying URN**

  Right-click the target function and choose **Copy URN** from the shortcut menu.

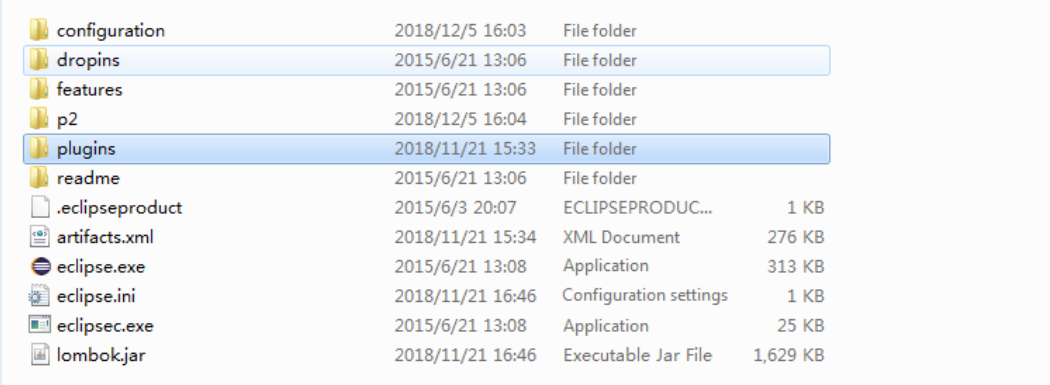
9.2 Eclipse Plug-in

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 Obtain the **Eclipse plug-in** (software package verification file: **Eclipse plug-in.sha256**).

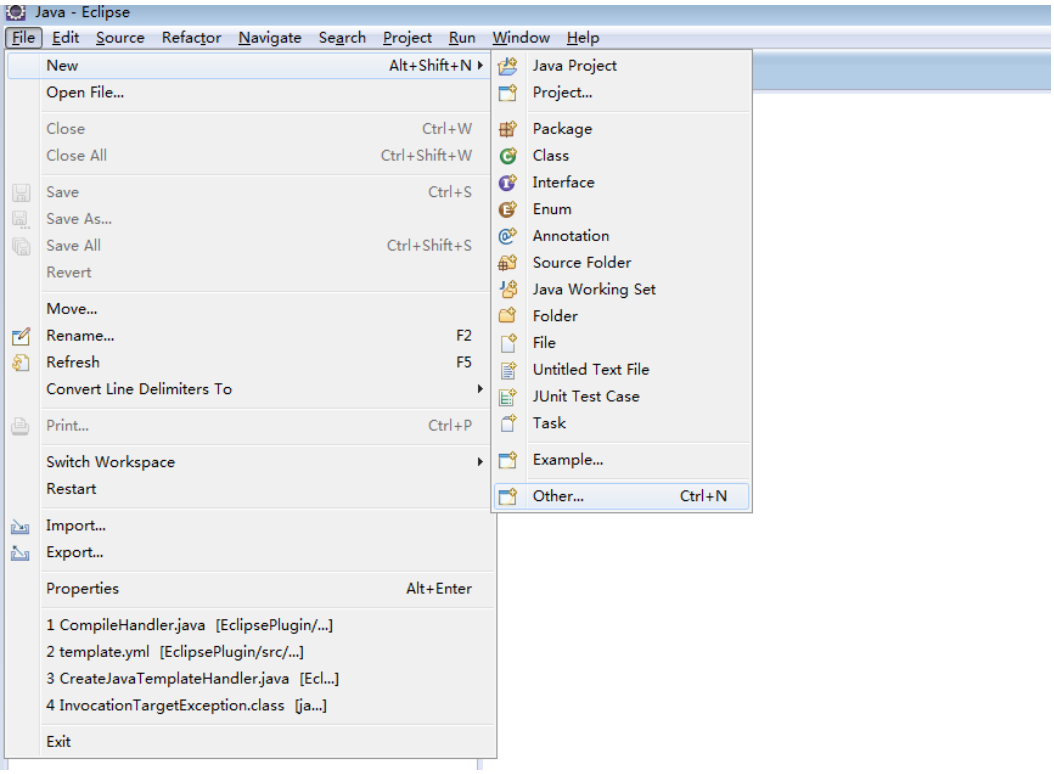
Step 2 Put the Eclipse plug-in package (.jar or .zip) in the **plugins** folder under the Eclipse installation directory. Then restart Eclipse. **Figure 9-5** shows the Eclipse installation directory.

Figure 9-5 Installing the Eclipse plug-in



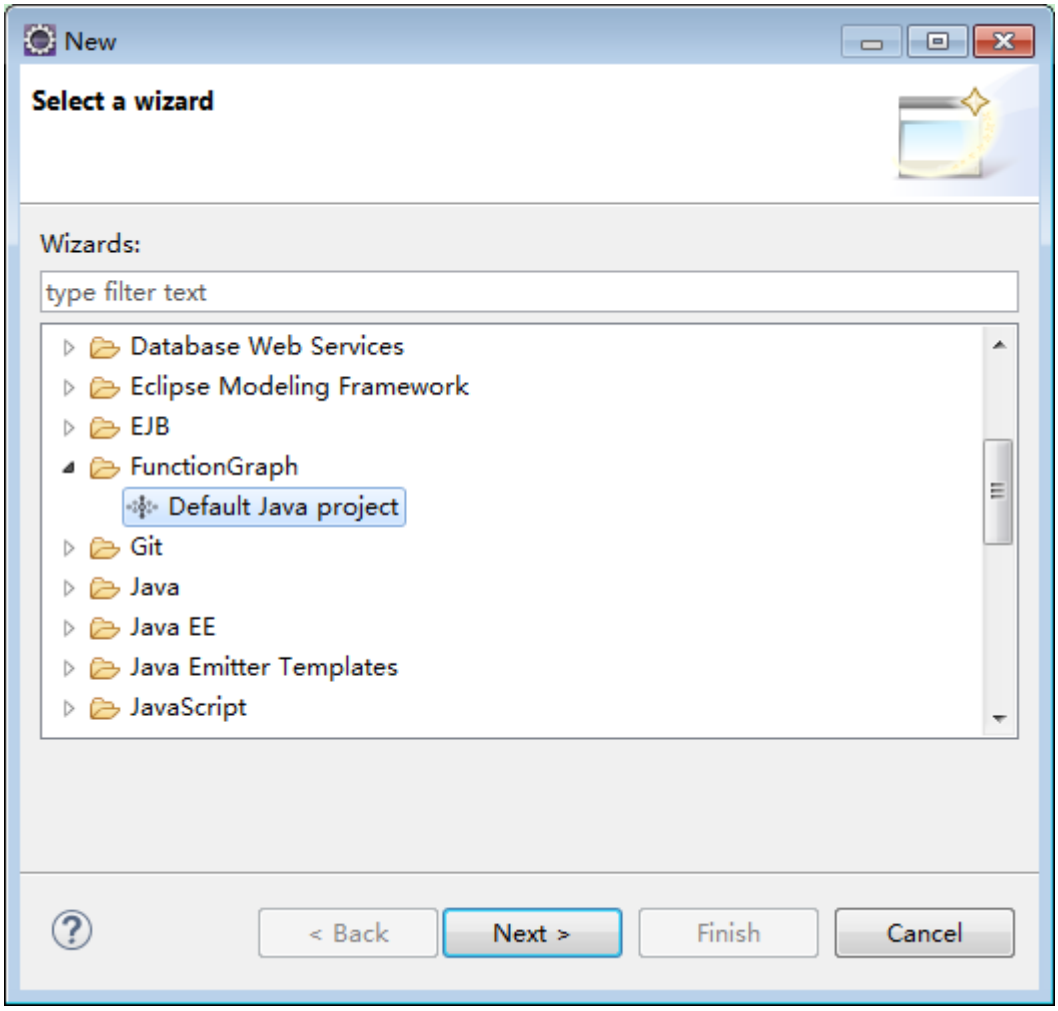
Step 3 Open Eclipse and choose **File** > **New** > **Other**, as shown in **Figure 9-6**.

Figure 9-6 Creating a template



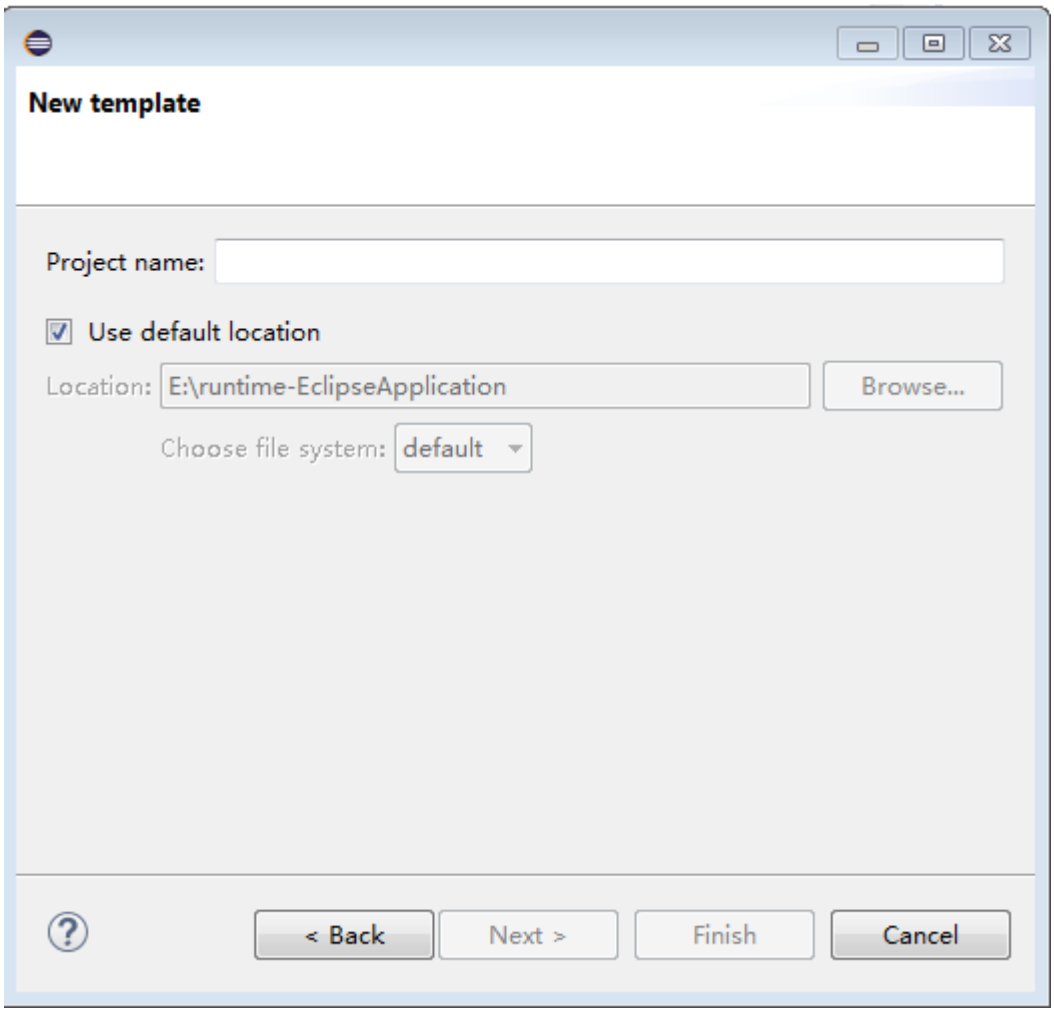
Step 4 Choose **FunctionGraph** > **Default Java project**, as shown in **Figure 9-7**.

Figure 9-7 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 9-8**.

Figure 9-8 Setting the project name and directory



----End

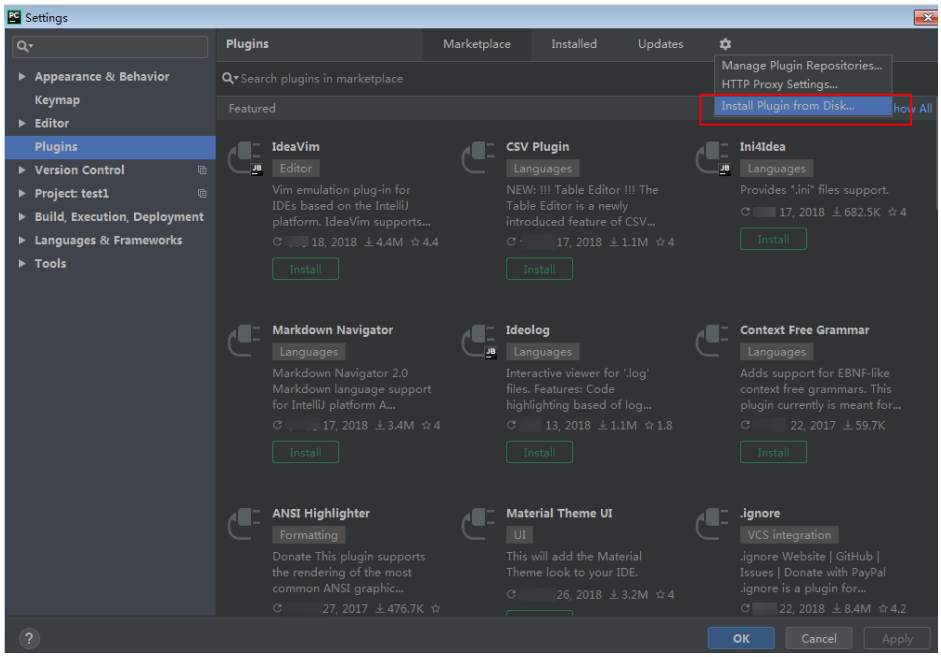
9.3 PyCharm Plug-in

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

Step 1 Obtain the **plug-in** (**Plug-in.sha256**) .

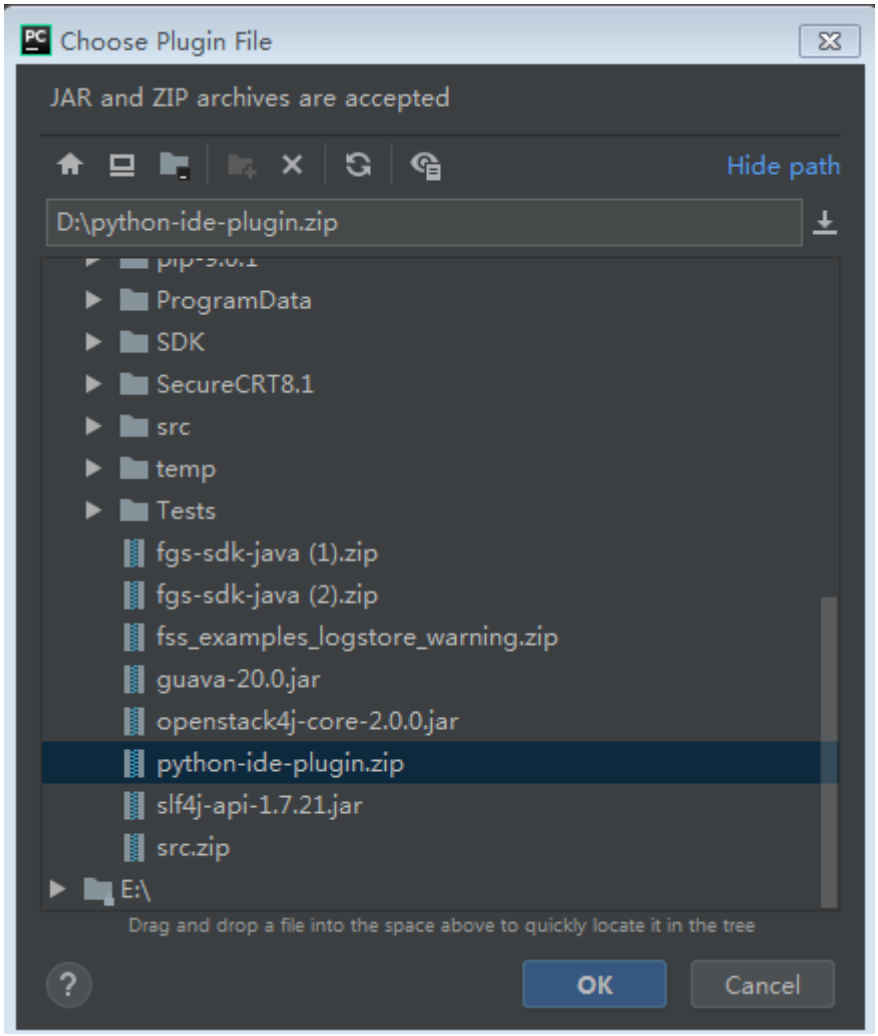
Step 2 Run JetBrains PyCharm. Choose **File** > **Settings**, choose **Plugins** in the left pane, and then click **Install Plugin from Disk** in the upper right corner, as shown in **Figure 9-9**.

Figure 9-9 Installing the plug-in



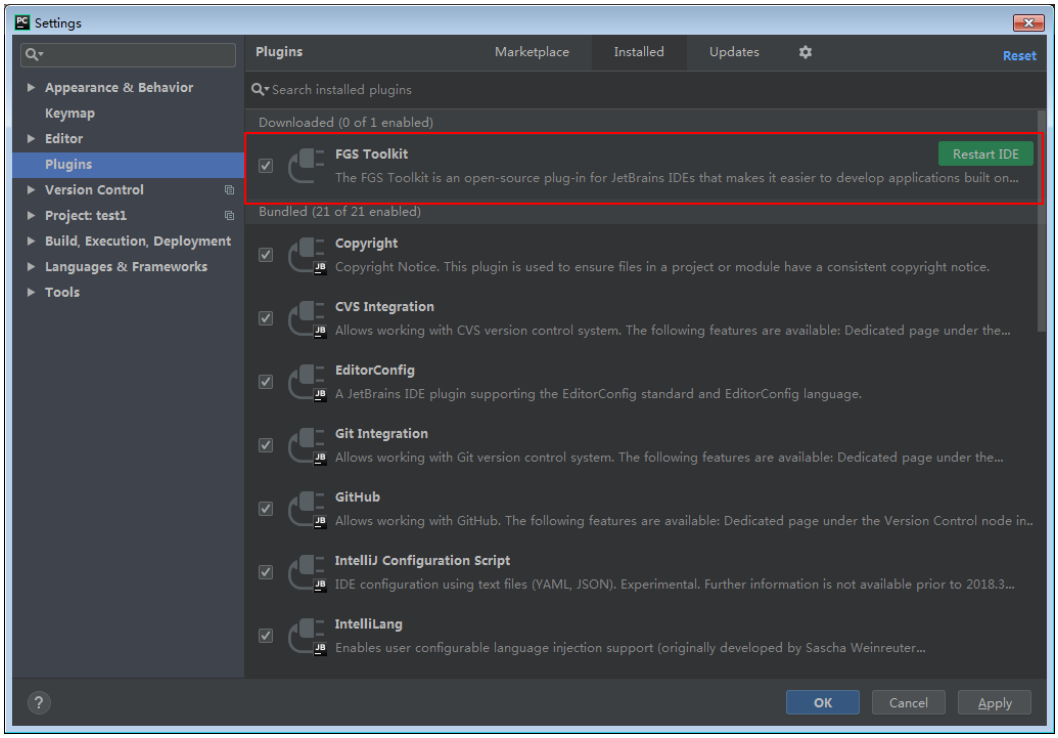
Step 3 Select the plug-in package you want to install, and click **OK**, as shown in **Figure 9-10**.

Figure 9-10 Selecting a plug-in package



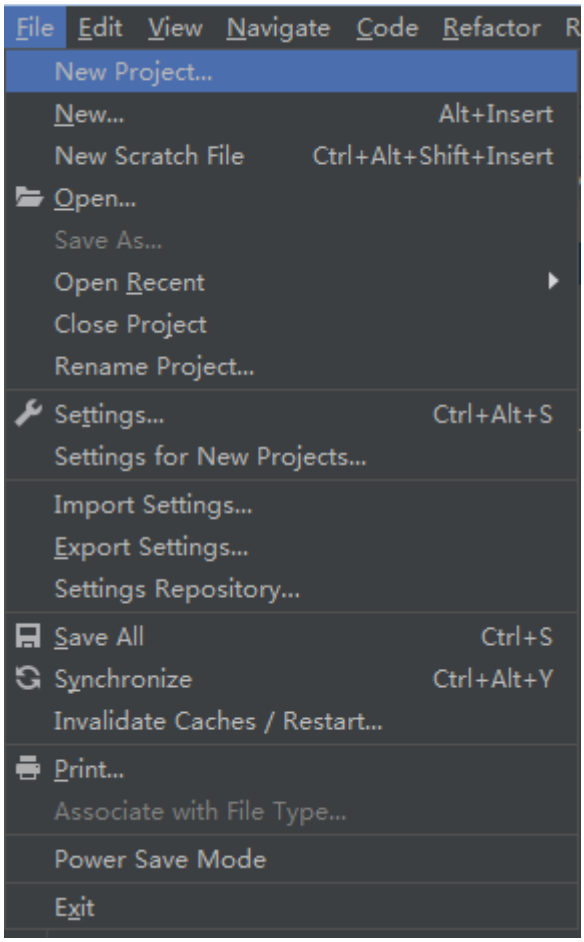
Step 4 In the plug-in list, select the desired plug-in and click **Restart IDE**, as shown in **Figure 9-11**.

Figure 9-11 Restarting the IDE



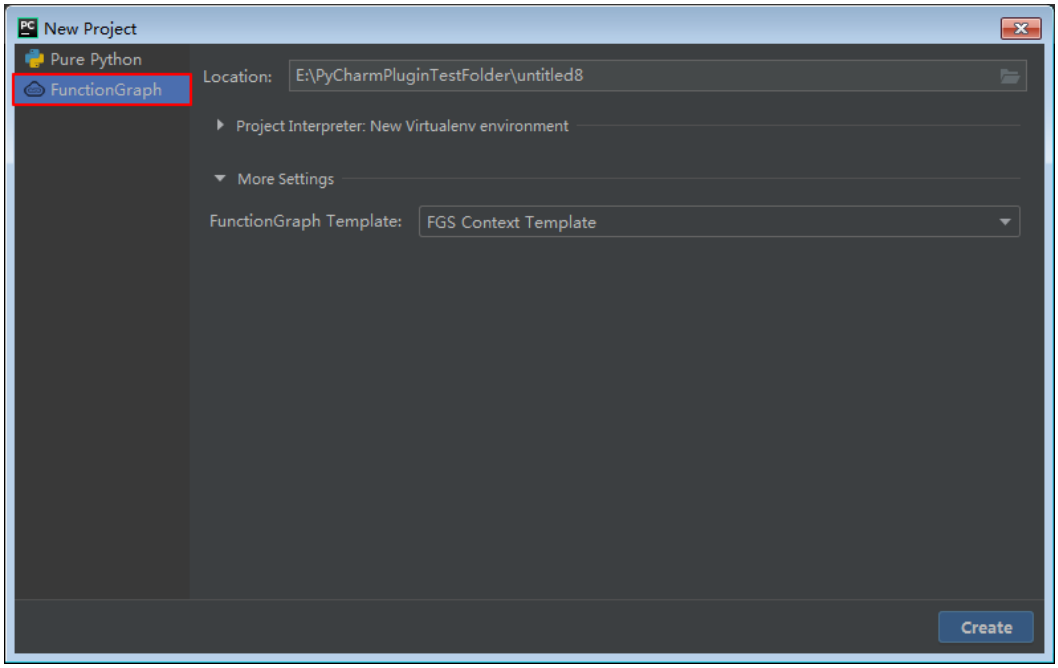
Step 5 Choose **File** > **New Project**, as shown in **Figure 9-12**.

Figure 9-12 Creating a project



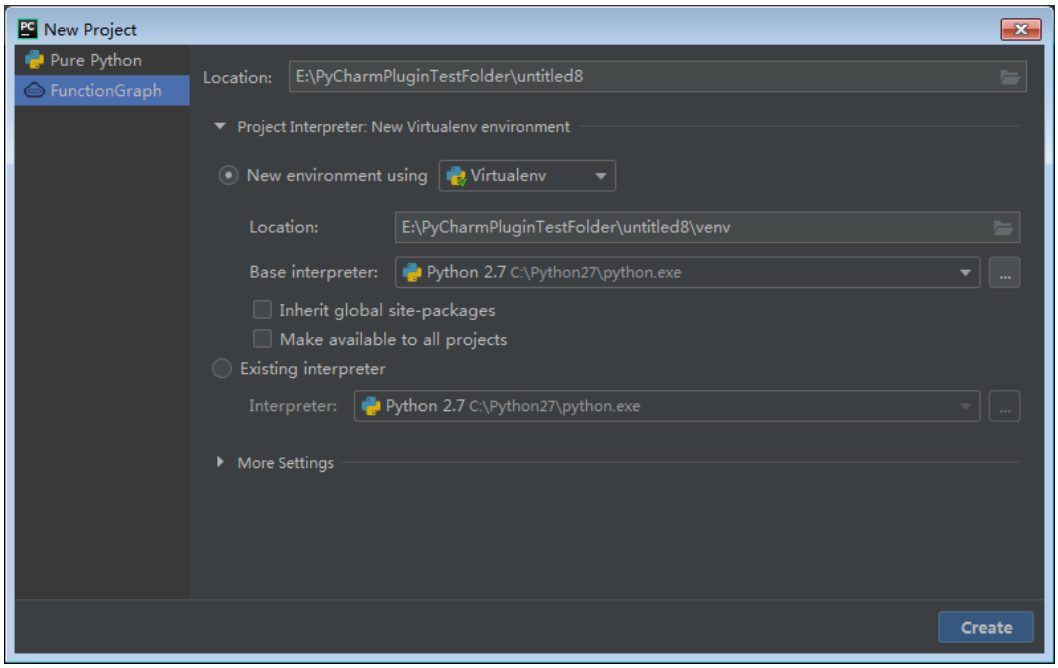
Step 6 On the displayed **New Project** page, choose **FunctionGraph**, as shown in **Figure 9-13**.

Figure 9-13 FunctionGraph



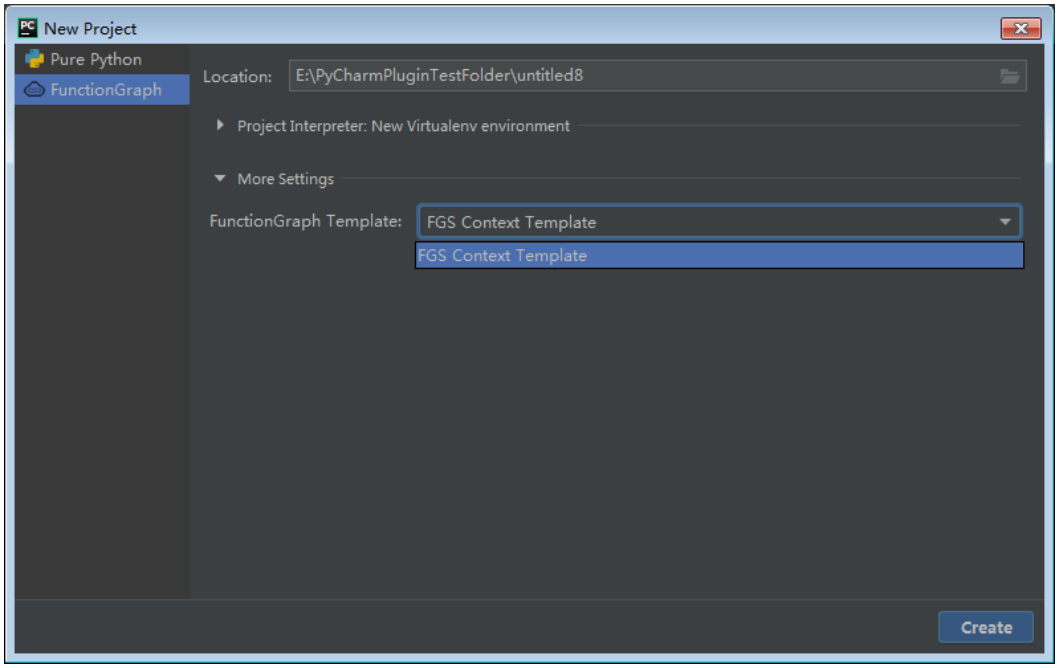
Step 7 Select the path in which the project will be stored in **Location**, and select a Python version in **Base interpreter**, as shown in **Figure 9-14**.

Figure 9-14 Selecting a version



Step 8 Select a template you want to create in the **More Settings** area, as shown in **Figure 9-15**.

Figure 9-15 Selecting a template



NOTE

Currently, only the Python 2.7 context template is supported.

Step 9 Click **Create**.

----End

