ROMA Connect

Developer Guide

Issue	03
Date	2024-10-10

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions

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 Developer Guide for Data Integration

1.1 Overview

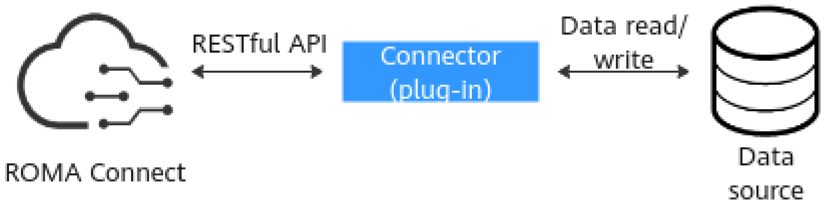
1.1.1 Scenarios

Description

With ROMA Connect, you can develop your own plug-ins to implement data read and write for currently incompatible data sources.

The plug-ins serve as a RESTful API to which ROMA Connect is able to connect.

Such data source plug-ins are also called connectors, and the data sources accessed through these connectors are custom data sources.



1.1.2 Specifications

Suggestions

Suggestions on data integration development:

- ROMA Connect provides only the RESTful API specifications of connectors. You need to develop your own RESTful API and connector to read and write data sources in a language you prefer.
- Deploy the connector once you complete the development. Ensure that the network between the connector and ROMA Connect is normal.

1.2 RESTful API Specifications of Connectors

Data Reading API

API Specification Definition

- URI

  POST /reader

- API request

```
{
    "job_name": "job_name",
    "datasource": {
        "para1": "******",
        "para2": "******",
         ...
    },
    "params": {
        "extend": {
            "ex_para1": "******",
            "ex_para2": "******",
             ...
        },
        "pagination": {
            "offset": 1,
            "limit": "10"
        },
        "migration": {
            "begin": **********,
            "end": **********
        }
    }
}
```

- API response

```
{
    "datas": [
        {
            "para1": "******",
            "para2": "******",
             ...
        },
        {
            "para1": "******",
            "para2": "******",
             ...
        },
        ...
    ]
}
```

API Parameters

- Request parameters

Table 1-1 Request parameters

Parameter	Mandatory	Type	Description
job_name	Yes	String	Task name. The value can contain 4 to 64 characters, including letters, digits, hyphens (-), and underscores (_).
datasource	Yes	Object	Data source object. The JSON body consists of the parameters on which the connector depends to connect to the database and all parameters.
params	Yes	**Params**	Parameter object required by the connector.

Table 1-2 Parameter description of params

Parameter	Mandatory	Type	Description
pagination	No	**Pagination**	Pagination object.
migration	No	**Migration**	Incremental migration object.
extend	No	Object	Extended parameters to which a connector belongs. The value is a JSON body consisting of extended parameters.

Table 1-3 Parameter description of pagination

Parameter	Mandatory	Type	Description
offset	No	Integer	Offset from which the query starts.
limit	No	Integer	Number of records displayed on each page.

Table 1-4 Parameter description of migration

Parameter	Mandatory	Type	Description
begin	No	Date	Start time of data migration.

Parameter	Mandatory	Type	Description
end	No	Date	End time of data migration.

- Response parameters

Table 1-5 Response parameters

Parameter	Type	Description
datas	List<Object>	List of data to be read. The value of this parameter must be in JSON array format and is determined by the connector based on the site requirements.

Data Writing API

API Specification Definition

- URI

  POST /writer

- API request

```
{
    "job_name": "job_name",
    "datasource": {
        "para1": "******",
        "para2": "******",
         ...
    },
    "params": {
        "extend": {
            "ex_para1": "******",
            "ex_para2": "******",
             ...
        }
    },
    "meta-data": [
        {
            "name": "id",
            "type": "String",
            "format": "",
            "path": "datas[i].id"
        },
        {
            "name": "company",
            "type": "String",
            "format": "",
            "path": "datas[i].company"
        },
        ...
    ],
    "datas": [
        {
            "data1": "******",
            "data2": "******",
             ...
        },
        {
            "data1": "******",
            "data2": "******",
             ...
        },
        ...
    ]
}
```

- API response

```
{
    "num_success": "2",
    "num_fail": "0",
    "fail_datas": [
        {}
    ]
}
```

API Parameters

- Request parameters

Table 1-6 Request parameters

Parameter	Mandatory	Type	Description
job_name	Yes	String	Task name. The value can contain 4 to 64 characters, including letters, digits, hyphens (-), and underscores (_).
datasource	Yes	Object	Data source object. The JSON body consists of the parameters on which the connector depends to connect to the database and all parameters.
params	Yes	**Params**	Parameter object required by the connector.
meta-data	Yes	List<**Meta-data**>	Metadata parameter list.
datas	Yes	List<Object>	List of data processed by the connector.

Table 1-7 Parameter description of params

Parameter	Mandatory	Type	Description
extend	No	Object	Extended parameters to which a connector belongs. The value is a JSON body consisting of extended parameters.

Table 1-8 Parameter description of meta-data

Parameter	Mandatory	Type	Description
name	Yes	String	Data field name.
type	Yes	String	Data field type. The value can be **String**, **Integer**, **Date**, or **Long**.
format	No	String	Format string of data. Format of a character string. This field needs to be specified when **type** is set to **Date**.
path	Yes	String	Path of a field in the source data.

- Response parameters

Table 1-9 Response parameters

Parameter	Type	Description
num_success	Integer	Number of times data is successfully written.
num_fail	Integer	Number of times data failed to be written.
fail_datas	List<Object>	List of data that fails to be processed.

1.3 (Example) Developing a Custom Data Source for a Scheduled Task

Scenarios

FDI supports MySQL, which is a common database type. This section uses MySQL as an example to describe how to develop a custom connector in Java. Refer to **MysqlConnctor.rar** for demo code.

Prerequisites

- A Linux server that runs JDK 1.8 or later is available.
- IntelliJ IDEA: 2018.3.5 or later; Eclipse: 3.6.0 or later
- Obtain **MysqlConnctor.rar** from the **demo** (sha256:34c9bc8d99eba4ed193603019ce2b69afa3ed760a452231ece3c89fd7dd74da1) package.
- A custom connector must support idempotent write.
- Processing a RESTful API request cannot take more than 60 seconds.
- FDI cyclically calls the RESTful API address until all data is read.

Procedure

1. Create a Spring Boot template project.

   Sample code:

```
@SpringBootApplication
public class MysqlConnectorApplication {
    public static void main(String[] args) {
        SpringApplication.run(MysqlConnectorApplication.class, args);
    }
}
```

2. Define the controller layer of the RESTful APIs.

   Sample code:

```
@RequestMapping(value = "mysql/reader", method = RequestMethod.POST)
    public ReaderResponseBody send(@RequestBody ReaderRequestBody readerRequestBody) throws 
Exception {
        if (readerRequestBody == null) {
            throw new RuntimeException("The reader request body is empty");
        }
        LOGGER.info("Accept a reader request, request={}", 
JSONObject.toJSONString(readerRequestBody));

        MysqlConfig mysqlConfig = getAndCheckMysqlConfig(readerRequestBody.getDatasource());
        String jdbcUrl = buildMysqlUrl(mysqlConfig);
        JSONArray dataList = mysqlReaderService.queryData(jdbcUrl, mysqlConfig, 
readerRequestBody.getParams());

        ReaderResponseBody readerResponseBody = new ReaderResponseBody();
        readerResponseBody.setDatas(dataList);
        return readerResponseBody;
    }
```

3. Implement the service layer of the read/write APIs.

   Sample code:

```
@Service
public class MysqlReaderService {
    public JSONArray queryData(String jdbcUrl, MysqlConfig mysqlConfig, ReaderParams 
readerParams) throws Exception {
        Connection conn = DBUtils.getConn(jdbcUrl, mysqlConfig);
        //Obtain pagination parameters.
        int limit = 0;
        int offset = 0;
        if (readerParams.getPagination() != null) {
            Pagination pagination = readerParams.getPagination();
            limit = pagination.getLimit() == 0 ? 10 : pagination.getLimit();
            offset = pagination.getOffset() == 0 ? 1 : pagination.getOffset();
        }
        //Obtain the name of the table to read.
        String tableName = readerParams.getExtend().getString("table_name");

        //Build SQL statements.
        StringBuilder sqlBuilder = new StringBuilder();
        sqlBuilder.append("select * from ").append(tableName);
sqlBuilder.append(" limit ?,? ");
        PreparedStatement preparedStatement = conn.prepareStatement(sqlBuilder.toString());
        preparedStatement.setInt(1, (offset - 1) * limit);
        preparedStatement.setInt(2, limit);
        ResultSet resultSet = preparedStatement.executeQuery();

        //Obtain the column name.
        List<String> columnList = getColumnInfo(resultSet);
        //Read the queried data.
        JSONArray dataArray = new JSONArray();
        while (resultSet.next()) {
            JSONObject data = new JSONObject();
            for (int i = 1; i <= columnList.size(); i++) {
                data.put(columnList.get(i - 1), resultSet.getString(i));
            }
            dataArray.add(data);
        }
        return dataArray;
    }
}
```

4. Define the input and output parameters of the read and write APIs.

   Sample code:

```
public class ReaderRequestBody {
    private String job_name;

    private JSONObject datasource;

    private ReaderParams params;
}
```

5. Run the following command in the **root** directory to generate an executable JAR package, for example, **MysqlConnector-1.0-SNAPSHOT.jar**, in **MysqlConnector\target**.

   # mvn package

6. Use Linux or Windows to upload the **MysqlConnector-1.0-SNAPSHOT.jar** package to the user server that runs JDK, and run the following command:

   # java -jar MysqlConnector-1.0-SNAPSHOT.jar &

   NOTE

   During development and debugging, start the **MysqlConnectorApplication.java** class through IntelliJ IDEA or Eclipse.

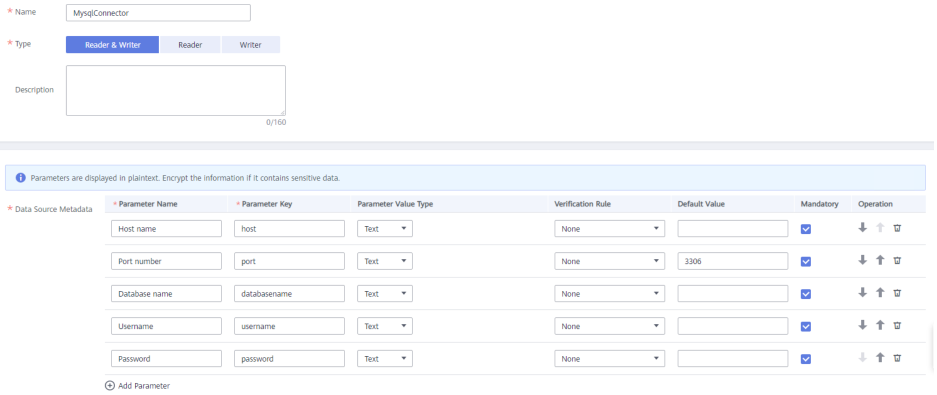
7. Create a custom connector model.

   a. Log in to the ROMA Connect console and choose **Assets** in the navigation pane on the left.

   b. Click **Create Connector** in the upper right corner of the page and set connector information by referring to **Creating a Connector**.

      Take MySQL as an example. Enter the host name, port number, database name, username, and password in the data source definition.

      Figure 1-1 Connector configuration 1



* Name: MysqlConnector

* Type: Reader & Writer | Reader | Writer

Description: 0/160

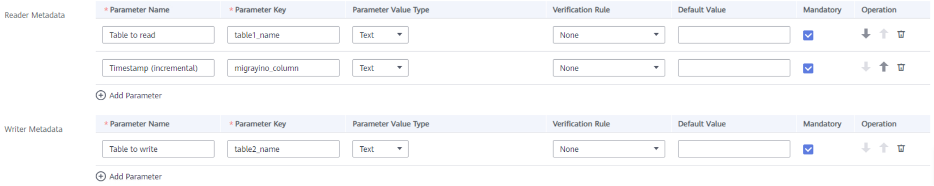
Parameters are displayed in plaintext. Encrypt the information if it contains sensitive data.

* Data Source Metadata

* Parameter Name	* Parameter Key	Parameter Value Type	Verification Rule	Default Value	Mandatory	Operation
Host name	host	Text	None		☑	↓ ↑ 🗑
Port number	port	Text	None	3306	☑	↓ ↑ 🗑
Database name	databasename	Text	None		☑	↓ ↑ 🗑
Username	username	Text	None		☑	↓ ↑ 🗑
Password	password	Text	None		☑	↓ ↑ 🗑

      In the read/write parameter definitions, enter the additional information required when the custom plug-in reads or writes, such as the name of the table to read/write and the name of the timestamp field for incremental read.

Figure 1-2 Connector configuration 2



Reader Metadata

* Parameter Name	* Parameter Key	Parameter Value Type	Verification Rule	Default Value	Mandatory	Operation
Table to read	table1_name	Text	None		☑	
Timestamp (incremental)	migrayino_column	Text	None		☑	

Add Parameter

Writer Metadata

* Parameter Name	* Parameter Key	Parameter Value Type	Verification Rule	Default Value	Mandatory	Operation
Table to write	table2_name	Text	None		☑	

Add Parameter

8. Publish the connector.

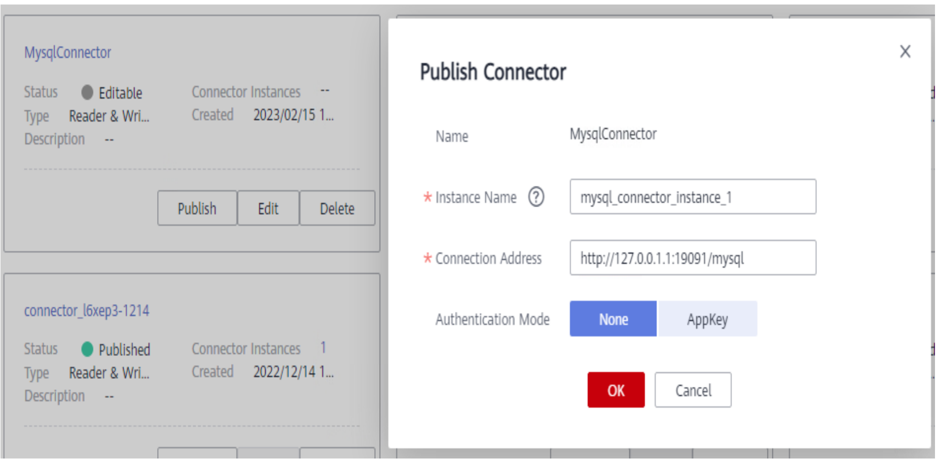
   After the connector is created, click **Publish** to publish its instance.

   NOTE

   The relationship between a connector and a connector instance is similar to that between a class and a class object.

   A connector defines specifications for a data source, while a connector instance corresponds to a specific RESTful service. The RESTful service's access address is required, which is determined by the user server address.

Figure 1-3 Publishing a connector



MysqlConnector

Status: Editable
Type: Reader & Wri...
Description: --
Connector Instances: --
Created: 2023/02/15 1...

connector_l6xep3-1214

Status: Published
Type: Reader & Wri...
Description: --
Connector Instances: 1
Created: 2022/12/14 1...

Publish Connector

Name: MysqlConnector

* Instance Name: mysql_connector_instance_1

* Connection Address: http://127.0.0.1:19091/mysql

Authentication Mode: None

9. Connect to the custom data source.

   a. In the navigation pane on the left, choose **Data Sources**. In the upper right corner of the page, click **Access Data Source**.

   b. On the **Custom** tab page, select the custom connector **MysqlConnector** and click **Next**.

   c. On the page, configure the data source connection information. Select an instance of the connector as the connection instance and enter the data source information defined by the connector.

10. The following uses a custom data source as the source and MySQL as the destination to describe how to create a scheduled task.

    Connect the custom data source at the source and the MySQL data source at the destination and create a scheduled task. For details, see **Creating a Common Data Integration Task**. After the configuration is complete, run the task to migrate data from the custom data source to MySQL tables.

NOTE

After the task is executed, FDI reads or writes data based on the connection address (**http://127.0.0.1:19091/mysql**) defined by the custom connection instance. (Add **/reader** to the address for data read or add **/writer** for data write.)

1.4 (Example) Developing a Custom Data Source for a Real-Time Task

Scenarios

Custom connectors can access real-time data sources through message forwarding. This section uses an MQS data source and Java as an example. Refer to **RealtimeConnector.rar** for demo code.

Prerequisites

- A Linux server that runs JDK 1.8 or later is available.
- IntelliJ IDEA: 2018.3.5 or later; Eclipse: 3.6.0 or later
- Obtain **MysqlConnctor.rar** from the **demo** (sha256:34c9bc8d99eba4ed193603019ce2b69afa3ed760a452231ece3c89fd7dd74da1) package.
- The TPS for user programs to write messages to MQS cannot exceed 6000.

Procedure

1. Create a Spring Boot template project, start real-time data source consumption in the **Main** method, and use the MQS SDK to produce the consumed data to MQS.

   Sample code:

```
@SpringBootApplication
public class RealtimeConnectorApplication {
    private static final Logger LOGGER = LoggerFactory.getLogger(RealtimeConnectorApplication.class);

    public static void main(String[] args) throws MQClientException {
        DefaultMQPushConsumer rocketMQConsumer = createRocketMQConsumer();
        MqsProducer mqsProducer = new MqsProducer();
        MessageListenerConcurrently rocketmqMessageListener = new MessageListenerConcurrently() {
            @Override
            public ConsumeConcurrentlyStatus consumeMessage(List<MessageExt> messageList,
                ConsumeConcurrentlyContext context) {
                for (MessageExt message : messageList) {
                    String jsonString = convertMessageToJsonString(message);
                    //Write JSON data to MQS. mqs-topic indicates the created topic, which will be
consumed by FDI tasks.
                    mqsProducer.produce("mqs-topic", jsonString);
                }
                LOGGER.info("Success to process {} data", messageList.size());
                //Mark the message as consumed.
                return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
            }
        };
        //Register the callback implementation class to process the messages obtained from RocketMQ.
        rocketMQConsumer.registerMessageListener(rocketmqMessageListener);
        //Start RocketMQ consumption.
        rocketMQConsumer.start();
    }
    private static DefaultMQPushConsumer createRocketMQConsumer() throws MQClientException {
        //Instantiate the RocketMQ consumer. Enter the actual consumer group name.
        DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("myCompanyGroup");
        //Set the NameServer IP address.
        consumer.setNamesrvAddr("localhost:9876");
        //Subscribe to one or more topics and use tags to filter messages to consume.
        consumer.subscribe("RocketMQTopic", "*");
        return consumer;
    }

    /**
     * Convert the messages read by RocketMQ into JSON strings. The actual conversion is
implemented based on the RocketMQ message content.
     *
     * @param messageExt
     * @return
     */
    private static String convertMessageToJsonString(MessageExt messageExt) {
        JSONObject jsonObject = new JSONObject();
        jsonObject.put("id", 1);
        jsonObject.put("name", "zhangsan");
        return jsonObject.toJSONString();
    }
}
```

2. Run the following command in the **root** directory to generate an executable JAR package, for example, **RealtimeConnector-1.0-SNAPSHOT.jar**, in **RealtimeConnector\target**.

   # **mvn package**

3. Use Linux or Windows to upload the **RealtimeConnector-1.0-SNAPSHOT.jar** package to the user server that runs JDK, and run the following command:

   # **java -jar RealtimeConnector-1.0-SNAPSHOT.jar &**

4. The following uses an MQS data source as the source and MySQL as the destination to describe how to create a real-time task.

   Connect the MQS data source at the source and the MySQL data source at the destination and create a real-time task. For details, see **Creating a Common Data Integration Task**. After the configuration is complete, run the task to migrate data from the MQS data source to MySQL tables.

2 Developer Guide for Service Integration

2.1 Overview

2.1.1 Scenarios

Description

ROMA Connect service integration involves the following development scenarios:

- **API calling authentication**: When a service system calls an API opened by APIC and the API uses App or IAM authentication, API calling authentication should be developed for the service system to add authentication information to the API request.
  - App authentication (with a signature): The service system integrates the ROMA Connect SDK to sign API requests.
  - IAM authentication (with a token): The service system obtains the authentication token from the cloud service and secures the API request with the token.
  - IAM authentication (with an AK/SK pair): The service system integrates the ROMA Connect SDK to sign API requests.
- **Custom backend**: When you create a function or data backend using an APIC custom backend, function scripts or database execution statements should be compiled.
  - Function backend: Compile function scripts to develop functions. ROMA Connect has provided some Java functions for your reference.

  - Data backend: Compile execution statements to perform operations on the data source.
- **Backend service signature verification**: If an API is bound to a signature key on ROMA Connect, the request sent by ROMA Connect to the backend service of the API carries the corresponding signature information. The backend service of the API must integrate the SDK provided by ROMA Connect and verify the signature information in the request.

2.1.2 Specifications

Development Requirements for API Calling Authentication

- **Development tool versions**:
  - IntelliJ IDEA: 2018.3.5 or later
  - Eclipse: 3.6.0 or later
  - Visual Studio: 2019 version 16.8.4 or later.
- **Development language versions**:
  - Java: Java Development Kit 1.8.111 or later
  - Go: 1.14 or later
  - Python: 2.7 or 3.X
  - JavaScript: Node.js of 15.10.0 or later
  - PHP: 8.0.3 or later
  - Android: Android Studio 4.1.2 or later
- **Browser version**: Chrome 89.0 or later
- **SDK signature restrictions**:
  - When you use the SDK to sign API requests, only the requests whose body is 12 MB or smaller can be signed.
  - When sending an API request, the SDK adds the current time to the **X-Sdk-Date** header and adds the signature information to the **Authorization** header. The signature is valid only within a limited period of time.
  - In addition to verifying the time format of **X-Sdk-Date**, ROMA Connect also verifies the time difference between the time specified by **X-Sdk-Date** and the actual time when the request is received. If the time difference exceeds 15 minutes, ROMA Connect rejects the request. Therefore, the client must synchronize the local time with the NTP server to prevent a large offset of **X-Sdk-Date** in the request header.

Custom Backend Development Requirements

- **Function backends**:
  - Only JavaScript can be used for function compilation, which complies with the Java Nashorn standard and supports **ECMAScript Edition 5.1**.
  - The maximum script size supported by a function backend is 32 KB.
- **Data backends**:
  - If a large amount of data is obtained by executing statements at a data backend, you are advised to add the **offset** and **limit** parameters for

result paging to prevent response timeout caused by massive response data. The following shows the usage.

```
select * from table01 limit '${limit}' offset ${offset}
```

Keys of the **offset** and **limit** parameters can be transferred in the headers, parameters, or body of backend requests.

  - A maximum of 2000 records can be displayed on one page. You are advised to disable result paging.
  - The maximum statement size supported by a data backend is 32 KB.

Development Requirements for Backend Service Signature Verification

- **Development tool versions**:
  - IntelliJ IDEA: 2018.3.5 or later
  - Eclipse: 3.6.0 or later
  - Visual Studio: 2019 version 16.8.4 or later.
- **Development language versions**:
  - Java: Java Development Kit 1.8.111 or later
  - Python: 2.7 or 3.X
- **SDK usage restrictions**:
  - The Java SDK supports only basic and HMAC backend service signatures.
  - The Python SDK supports only HMAC backend service signatures.
  - The C# SDK supports only HMAC backend service signatures.

2.2 Developing API Calling Authentication (App)

2.2.1 Preparations

Obtaining API Calling Information

- Obtaining API request information

  Old version: On the ROMA Connect instance console, choose **API Connect** > **API Management**. On the **APIs** tab page, obtain the domain name, request method, and request path of an API. Click the API name to go to the details page. On the **API Calling** tab page, obtain the request protocol and input parameters of the API.

  New version: On the ROMA Connect instance console, choose **API Connect** > **APIs** and obtain the domain name, request method, and URL of an API. Click the API name to go to the details page and obtain the request protocol and input parameters of the API.
- Obtaining API authentication information

  Old version: To sign an API request cryptographically using App authentication (signature authentication), the key and secret (or the client AppKey and AppSecret) of a credential authorized by the API are required.

NOTE

- Key/AppKey: access key ID of the application. It is a unique identifier associated with a secret access key, both of which are used together to sign requests cryptographically.
- Secret/AppSecret: secret access key used together with the key/AppKey to sign requests. The secret/AppSecret identifies a request sender to prevent the request from being modified.

- Obtaining the key and secret of an integration application

  On the ROMA Connect console, choose **Integrated Applications**. Click the name of an integrated application authorized by the API. On the details page that is displayed, obtain the key and secret of the integrated application.

- Obtaining the AppKey and AppSecret of a client

  On the ROMA Connect console, choose **API Connect** > **API Calling**. On the **Clients** tab page, click the name of a client bound to the API. On the client details page that is displayed, obtain the AppKey and AppSecret of the client.

New version: To sign an API request cryptographically using App authentication (signature authentication), the key and secret of a credential authorized by the API are required.

On the ROMA Connect instance console, choose **API Connect** > **Credentials**. Click the name of a credential authorized by the API. On the page displayed, obtain the key and secret of the credential.

Preparing the Development Environment

- Installing a development tool

  Select a proper development tool based on the language used.

  - Download the installation package of IntelliJ IDEA 2018.3.5 or later from the **official IntelliJ IDEA website**.
  - Download the Visual Studio 2019 installation package of 16.8.4 or later from the **official Visual Studio page**.

- Installing a development language
  - Java: Download the JDK of 1.8.111 or later from the **official Oracle website**.
  - Go: Download the Go installation package of 1.14 or a later from the **official Go download page**.
  - Python: Download the Python 2.7 or 3.X installation package from the **official Python download page**.
  - JavaScript: Download the Node.js installation package of 15.10.0 or later from the **official Node.js download page**.
  - PHP: Download the PHP installation package of 8.0.3 or later from the **official PHP download page**.
  - Android: Download the Android Studio installation package of 4.1.2 or later from the **official Android Studio page**.

2.2.2 App Authentication Principles

Process

1. Construct a standard request.

   Assemble the request content according to the rules of APIC, ensuring that the client signature is consistent with that in the backend request.

2. Create a to-be-signed string using the standard request and other related information.
3. Calculate a signature using the AK/SK and to-be-signed string.
4. Add the generated signature to an HTTP request as a header or query parameter.
5. After receiving the request, APIC performs **1** to **3** to calculate a signature.
6. The new signature is compared with the signature generated in **3**. If they are consistent, the request is processed; otherwise, the request is rejected.

NOTE

The body of a signing request in app authentication mode cannot exceed 12 MB.

Step 1: Construct a Standard Request

To access an API through app authentication, standardize the request content, and then sign the request. The client must follow the same request specifications as APIC so that each HTTP request can obtain the same signing result from the frontend and backend to complete identity authentication.

The pseudocode of standard HTTP requests is as follows:

```
CanonicalRequest =
    HTTPRequestMethod + '\n' +
    CanonicalURI + '\n' +
    CanonicalQueryString + '\n' +
    CanonicalHeaders + '\n' +
    SignedHeaders + '\n' +
    HexEncode(Hash(RequestPayload))
```

The following example shows how to construct a standard request.

Original request:

```
GET https://30030113-3657-4fb6-a7ef-90764239b038.apigw.exampleRegion.com/app1?b=2&a=1 HTTP/1.1
Host: 30030113-3657-4fb6-a7ef-90764239b038.apigw.exampleRegion.com
X-Sdk-Date: 20180330T123600Z
```

1. Specify an HTTP request method (**HTTPRequestMethod**) and end with a carriage return line feed (CRLF).

   Specify GET, PUT, POST, or another request method. Example of a request method:

   ```
   GET
   ```

2. Add a standard URI (**CanonicalURI**) and end with a CRLF.

   Description

   Path of the requested resource, which is the URI code of the absolute path.

   Format

According to RFC 3986, each part of a standard URI except the redundant and relative paths must be URI-encoded. If a URI does not end with a slash (/), add a slash at its end.

Example

For the URI **/app1**, the standard URI code is as follows:

```
GET
/app1/
```

NOTE

During signature calculation, the URI must end with a slash (/). When a request is sent, the URI does not need to end with a slash (/).

3. Add a standard query string (**CanonicalQueryString**) and end with a CRLF.

   Description

   Query parameters. If no query parameters are configured, the query string is an empty string.

   Format

   Standard query strings must meet the following requirements:

   – Perform URI encoding on each parameter and value according to the following rules:

     ■ Do not perform URI encoding on any non-reserved characters defined in RFC 3986, including A–Z, a–z, 0–9, hyphen (-), underscore (_), period (.), and tilde (~).

     ■ Use **%XY** to perform percent encoding on all non-reserved characters. **X** and **Y** indicate hexadecimal characters (0–9 and A–F). For example, the space character must be encoded as **%20**, and an extended UTF-8 character must be encoded in the "%XY%ZA%BC" format.

   – Add "*URI-encoded parameter name*=*URI-encoded parameter value*" to each parameter. If no value is specified, use a null string instead. The equal sign (=) is required.

     For example, in the following string that contains two parameters, the value of parameter **parm2** is null.

     ```
     parm1=value1&parm2=
     ```

   – Sort the parameters in alphabetically ascending order. For example, a parameter starting with uppercase letter **F** precedes another parameter starting with lowercase letter **b**.

   – Construct standard query strings from the first parameter after sorting.

   Example

   The following example contains two optional parameters **a** and **b**.

   ```
   GET
   /app1/
   a=1&b=2
   ```

4. Add standard headers (**CanonicalHeaders**) and end with a CRLF.

   Description

   List of standard request headers, including all HTTP message headers in the to-be-signed request. The **X-Sdk-Date** header must be included to verify the signing time, which is in the UTC time format *YYYYMMDDTHHMMSSZ* as specified in ISO 8601. When publishing an API in a non-RELEASE environment, you need to specify an environment name.

NOTICE

The client must synchronize the local time with the clock server to avoid a large offset in the value of **X-Sdk-Date** in the request header.

In addition to verifying the time format of **X-Sdk-Date**, ROMA Connect also verifies the time difference between the time specified by **X-Sdk-Date** and the actual time when the request is received. If the time difference exceeds 15 minutes, ROMA Connect rejects the request.

Format

CanonicalHeaders consists of multiple message headers, for example, **CanonicalHeadersEntry0 + CanonicalHeadersEntry1 + ...**. Each message header (**CanonicalHeadersEntry**) is in the format of **Lowercase(HeaderName) + ':' + Trimall(HeaderValue) + '\n'**.

NOTE

- **Lowercase**: a function for converting all letters into lowercase letters.
- **Trimall**: a function for deleting the spaces before and after a value.
- The last message header carries a newline character. Therefore, an empty line appears because the **CanonicalHeaders** field also contains a newline character according to the specifications.

Example

```
GET
/app1/
a=1&b=2
host:30030113-3657-4fb6-a7ef-90764239b038.apigw.exampleRegion.com
x-sdk-date:20180330T123600Z
```

NOTICE

Standard message headers must meet the following requirements:

- All letters in a header are converted to lowercase letters, and all spaces before and after the header are deleted.
- All headers are sorted in alphabetically ascending order.

For example, the original headers are as follows:

```
Host:30030113-3657-4fb6-a7ef-90764239b038.apigw.exampleRegion.com\n
Content-Type: application/json;charset=utf8\n
My-header1:    a   b   c  \n
X-Sdk-Date:20180330T123600Z\n
My-Header2:    "a   b   c"  \n
```

A standard header is as follows:

```
content-type:application/json;charset=utf8\n
host:30030113-3657-4fb6-a7ef-90764239b038.apigw.exampleRegion.com\n
my-header1:a   b   c\n
my-header2:"a   b   c"\n
x-sdk-date:20180330T123600Z\n
```

5. Add message headers (**SignedHeaders**) for request signing, and the headers end with a newline character.

Description

List of message headers used for request signing. This step is to determine which headers are used for signing the request and which headers can be ignored during request verification. The **X-Sdk-date** header must be included.

Format

SignedHeaders = Lowercase(HeaderName0) + ';' + Lowercase(HeaderName1) + ";" + ...

Letters in the message headers are converted to lowercase letters. All headers are sorted alphabetically and separated with commas.

Lowercase is a function for converting all letters into lowercase letters.

Example

In the following example, two message headers **host** and **x-sdk-date** are used for signing the request.

```
GET
/app1/
a=1&b=2
host:30030113-3657-4fb6-a7ef-90764239b038.apigw.exampleRegion.com
x-sdk-date:20180330T123600Z

host;x-sdk-date
```

6. Use a hash function, such as SHA-256, to create a hash value based on the body (**RequestPayload**) of the HTTP or HTTPS request.

   Description

   Request message body. The message body needs two layers of conversion: **HexEncode(Hash(RequestPayload))**. **Hash** is a function for generating message digest. Currently, SHA-256 is supported. **HexEncode**: the Base16 encoding function for returning a digest consisting of lowercase letters. For example, **HexEncode("m")** returns **6d** instead of **6D**. Each byte you enter is expressed as two hexadecimal characters.

   NOTE

   If **RequestPayload** is null, the null value is used for calculating a hash value.

   Example

   For a request with the GET method and an empty body, the body (empty string) after hash processing is as follows:

```
GET
/app1/
a=1&b=2
host:30030113-3657-4fb6-a7ef-90764239b038.apigw.exampleRegion.com
x-sdk-date:20180330T123600Z

host;x-sdk-date
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

7. Perform hash processing on the standard request in the same way as that on the **RequestPayload**. After hash processing, the standard request is expressed with lowercase hexadecimal strings.

   Algorithm pseudocode: **Lowercase(HexEncode(Hash.SHA256(CanonicalRequest)))**

   Example of the standard request after hash processing:

```
aa521bbe74d13cd8cf536c1a03a5dd85d1934179d33d47110b528eae8b7251e1
```

Step 2: Create a To-Be-Signed String

After a standard HTTP request is constructed and the request hash value is obtained, create a to-be-signed string by combining them with the signature algorithm and signing time.

```
StringToSign =
   Algorithm + \n +
   RequestDateTime + \n +
   HashedCanonicalRequest
```

Parameters in the pseudocode are described as follows:

- **Algorithm**

  Signature algorithm. For SHA256, the value is **SDK-HMAC-SHA256**.

- **RequestDateTime**

  Request timestamp, which is the same as **X-Sdk-Date** in the request header. The format is *YYYYMMDDTHHMMSSZ*.

- **HashedCanonicalRequest**

  Standard request generated after hash processing.

In this example, the following to-be-signed string is obtained:

```
SDK-HMAC-SHA256
20180330T123600Z
aa521bbe74d13cd8cf536c1a03a5dd85d1934179d33d47110b528eae8b7251e1
```

Step 3: Calculate the Signature

Use the AppSecret and created character string as the input of the encryption hash function, and convert the calculated binary signature into a hexadecimal expression.

The pseudocode is as follows:

```
signature = HexEncode(HMAC(APP secret, string to sign))
```

HMAC indicates hash calculation, and **HexEncode** indicates hexadecimal conversion. **Table 2-1** describes the parameters in the pseudocode.

Table 2-1 Parameter description

Name	Description
APP secret	Signature key.
string to sign	Character string to be signed.

Assuming that the AppSecret is **12345678-1234-1234-1234-123456781234**, a signature similar to the following will be calculated:

```
121c2501e8951ff7d5574423939b9acaa283e55a27c0107d767bb0d68b5ffcab
```

Step 4: Add the Signature to the Request Header

Add the signature to the HTTP Authorization header. The Authorization header is used for identity authentication and not included in the signed headers.

The pseudocode is as follows:

```
Pseudocode for Authorization header creation:
Authorization: algorithm Access=APP key, SignedHeaders=SignedHeaders, Signature=signature
```

There is no comma before the algorithm and **Access**. **SignedHeaders** and **Signature** must be separated with commas.

The signed headers are as follows:

```
Authorization: SDK-HMAC-SHA256 Access=071fe245-9cf6-4d75-822d-c29945a1e06a, SignedHeaders=host;x-
sdk-date, Signature=121c2501e8951ff7d5574423939b9acaa283e55a27c0107d767bb0d68b5ffcab
```

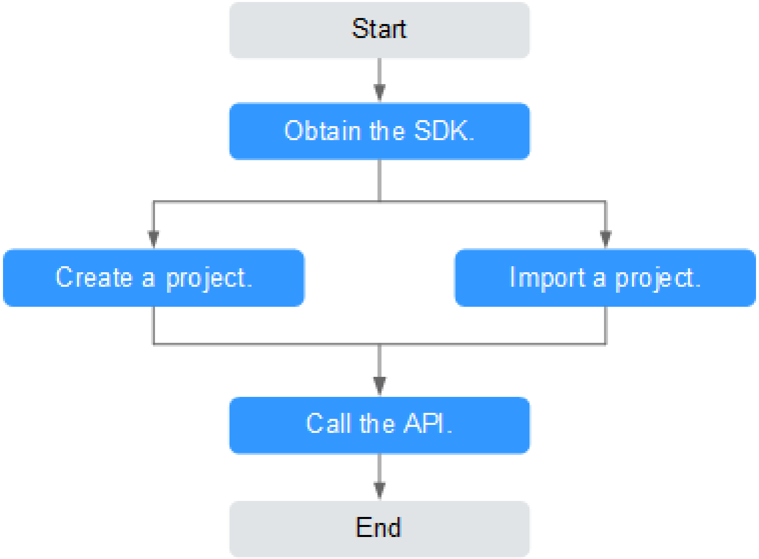
The signed headers are added to the HTTP request for identity authentication. If the identity authentication is successful, the request is sent to the corresponding backend service for processing.

2.2.3 Java

Scenarios

To use Java to call an API through App authentication, obtain the Java SDK, create a project or import an existing project, and then call the API by referring to the API calling example.

Figure 2-1 API calling process



Prerequisites

- You have obtained API calling information. For details, see **Preparations**.
- You have installed the development tool and Java development environment. For details, see **Preparations**.

Obtaining the SDK

Old version: Log in to the ROMA Connect console, choose **API Connect** > **API Calling** > **SDKs**, and download the SDK.

New version: Log in to the ROMA Connect console, choose **API Connect** > **Credentials** > **SDKs**, and download the SDK.

The following shows the directory structure after the decompression.

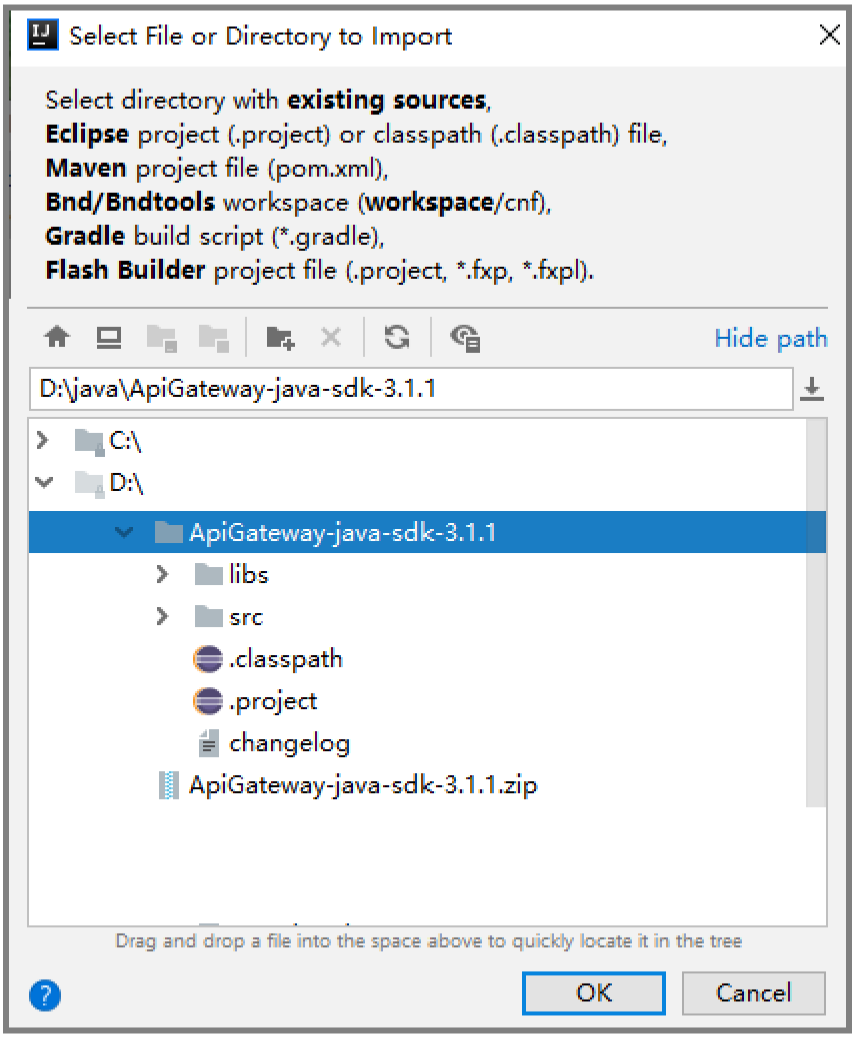
Name	Description
libs\	SDK dependencies
libs\java-sdk-core-*x.x.x*.jar	SDK package
src\com\apig\sdk\demo\Main.java	Sample code for signing requests
src\com\apig\sdk\demo\OkHttpDemo.java	
src\com\apig\sdk\demo\LargeFileUploadDemo.java	
.classpath	Java project configuration files
.project	

Importing a Project

1. Start IntelliJ IDEA and choose **Import Project**.

   The **Select File or Directory to Import** dialog box is displayed.

2. Select the directory where the SDK is decompressed and click **OK**.



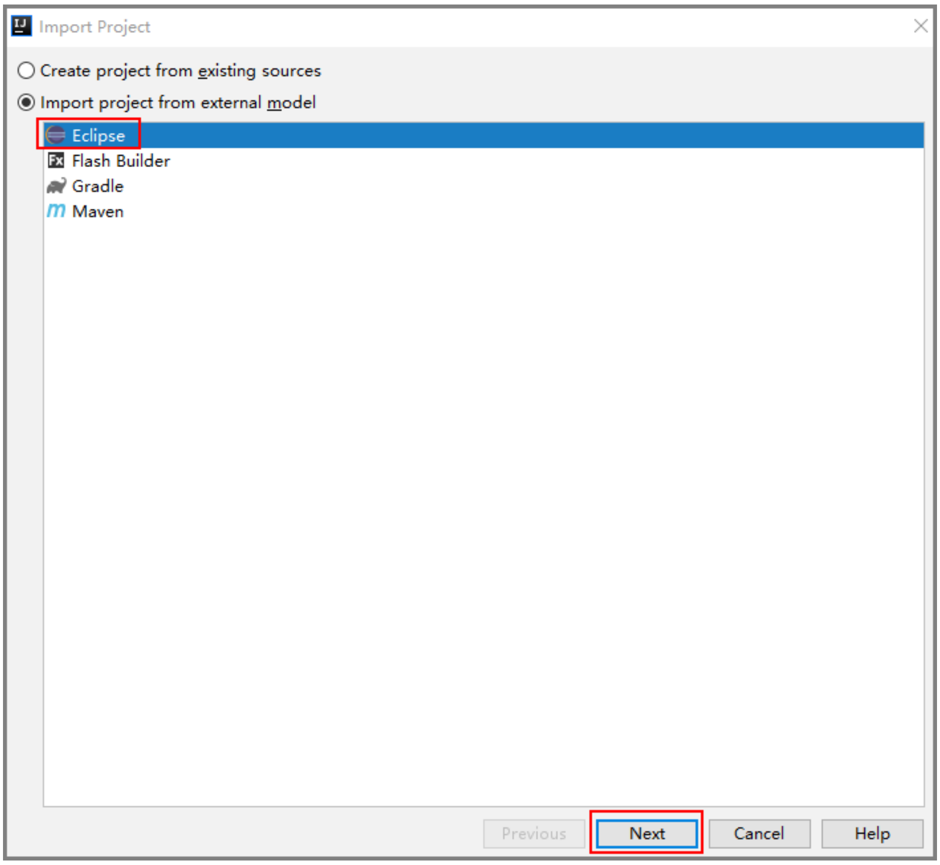
Select File or Directory to Import

Select directory with **existing sources**,
Eclipse project (.project) or classpath (.classpath) file,
Maven project file (pom.xml),
Bnd/Bndtools workspace (**workspace**/cnf),
Gradle build script (*.gradle),
Flash Builder project file (.project, *.fxp, *.fxpl).

Hide path
D:\java\ApiGateway-java-sdk-3.1.1
C:\
D:\
ApiGateway-java-sdk-3.1.1
libs
src
.classpath
.project
changelog
ApiGateway-java-sdk-3.1.1.zip
Drag and drop a file into the space above to quickly locate it in the tree
OK
Cancel

3. Select **Eclipse** for **Import project from external model** and click **Next**. Retain the default settings and click **Next** until the **Please select project SDK** page is displayed.

Figure 2-2 Import Project



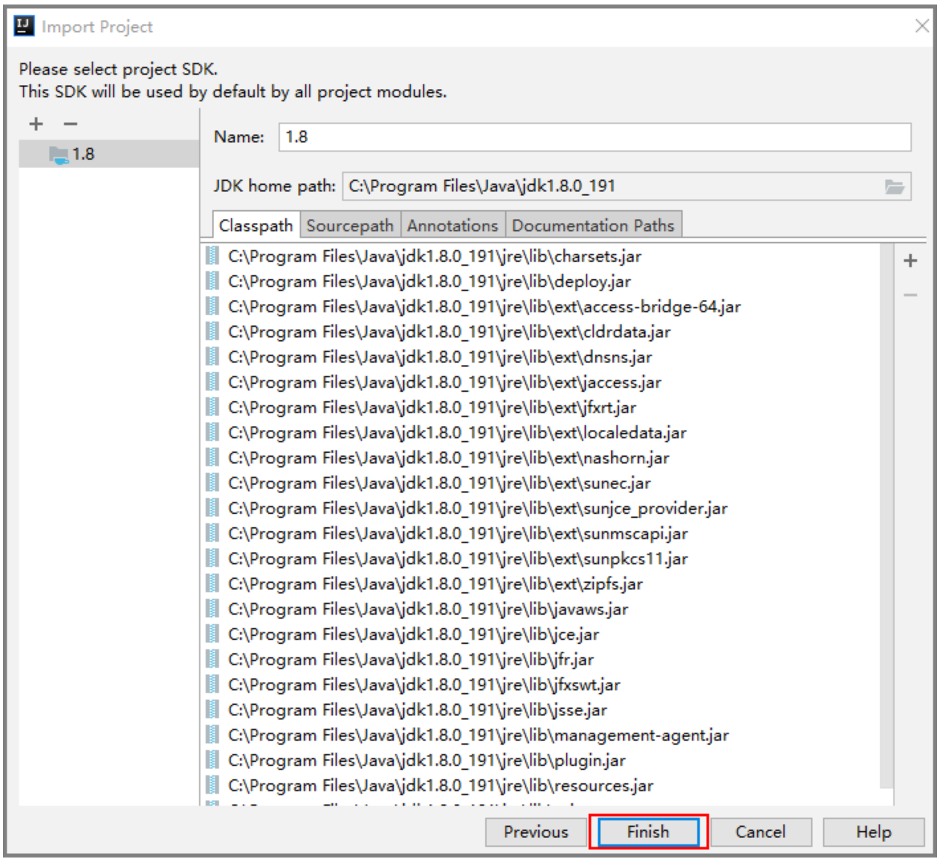
Import Project

○ Create project from existing sources

● Import project from external model

4. Click **Finish**.

Figure 2-3 Finish



Please select project SDK.
This SDK will be used by default by all project modules.

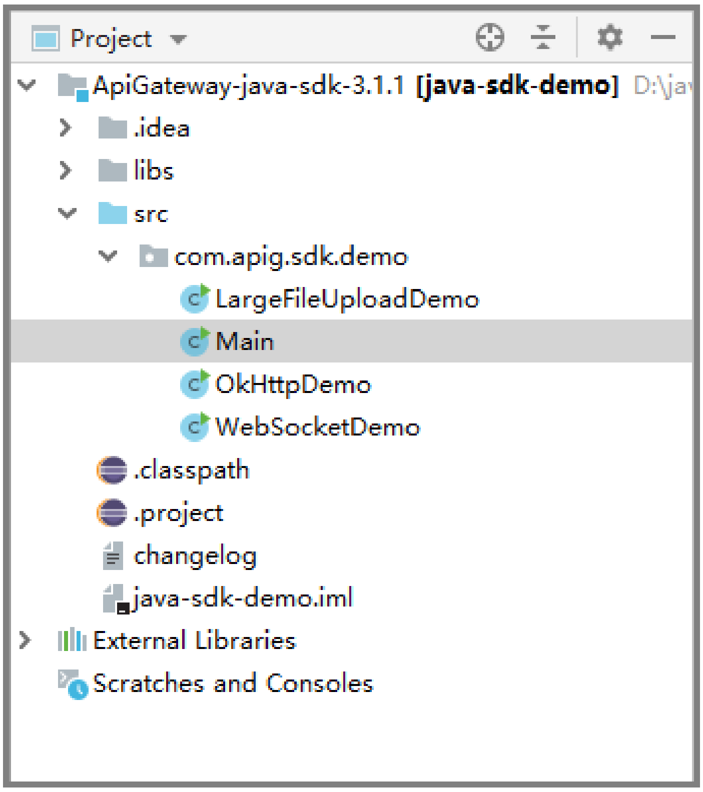
Name: 1.8

JDK home path: C:\Program Files\Java\jdk1.8.0_191

```
C:\Program Files\Java\jdk1.8.0_191\jre\lib\charsets.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\deploy.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\access-bridge-64.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\cldrdata.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\dnsns.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\jaccess.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\jfxrt.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\localedata.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\nashorn.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\sunec.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\sunjce_provider.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\sunmscapi.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\sunpkcs11.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\zipfs.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\javaws.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\jce.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\jfr.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\jfxswt.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\jsse.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\management-agent.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\plugin.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\resources.jar
```

5. After the import is complete, the directory structure is shown in the following figure.

Figure 2-4 Directory structure



Project

- ApiGateway-java-sdk-3.1.1 **[java-sdk-demo]** D:\ja
  - .idea
  - libs
  - src
    - com.apig.sdk.demo
      - LargeFileUploadDemo
      - Main
      - OkHttpDemo
      - WebSocketDemo
  - .classpath
  - .project
  - changelog
  - java-sdk-demo.iml
- External Libraries
- Scratches and Consoles

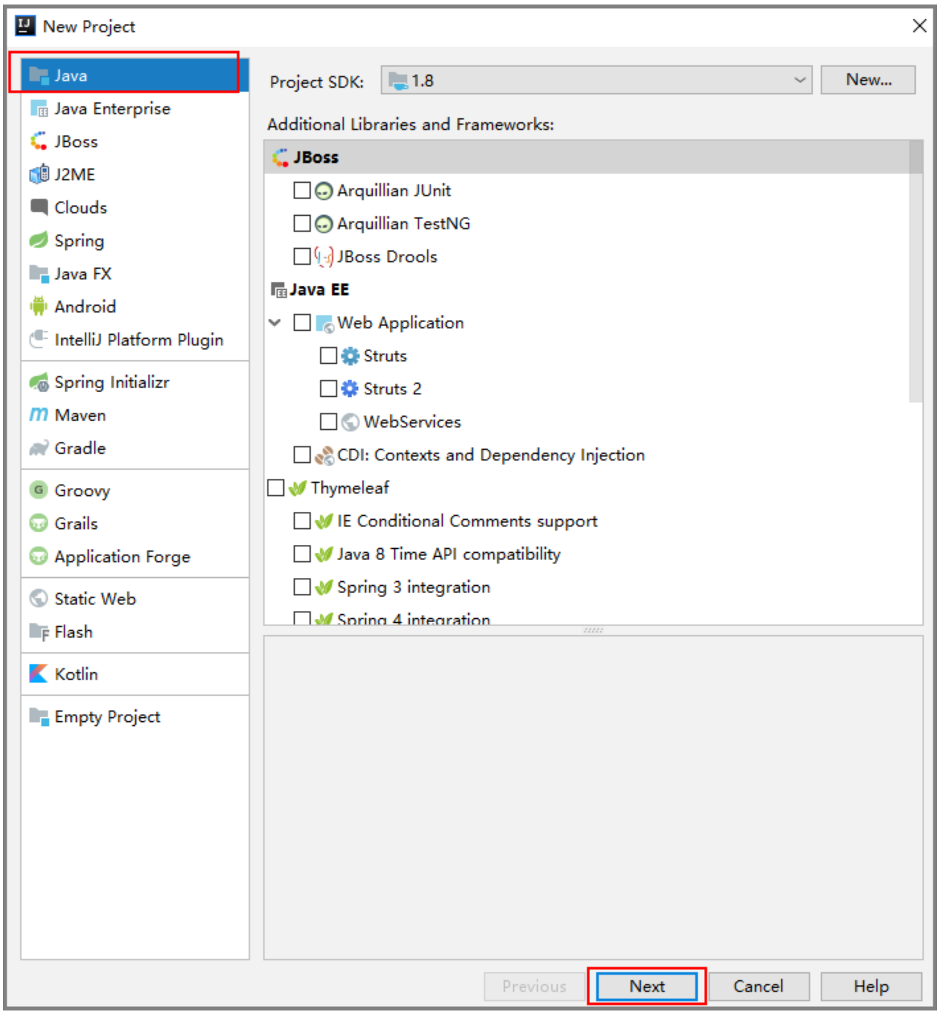
Creating a Project

1. Start IntelliJ IDEA and choose **Create New Project**.

   The **New Project** dialog box is displayed.

2. In the right pane, select **Java** and click **Next**.

Figure 2-5 New Project dialog box



Project SDK: 1.8 New...

Additional Libraries and Frameworks:

JBoss

- □ Arquillian JUnit
- □ Arquillian TestNG
- □ JBoss Drools

Java EE

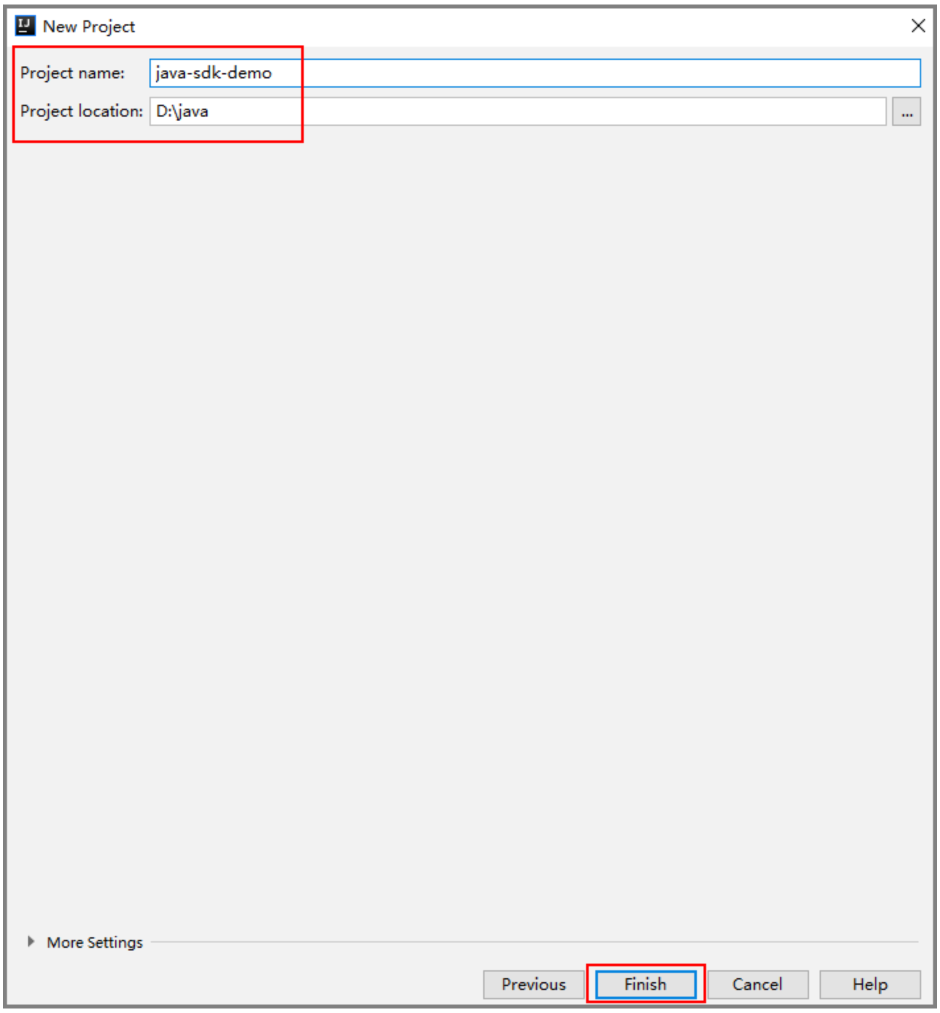
- □ Web Application
  - □ Struts
  - □ Struts 2
  - □ WebServices
- □ CDI: Contexts and Dependency Injection

□ Thymeleaf

- □ IE Conditional Comments support
- □ Java 8 Time API compatibility
- □ Spring 3 integration
- □ Spring 4 integration

3. Retain the default settings and click **Next**. On the page displayed, set **Project name** and select the local directory where the project is created for **Project location**.

Figure 2-6 New Project dialog box



New Project

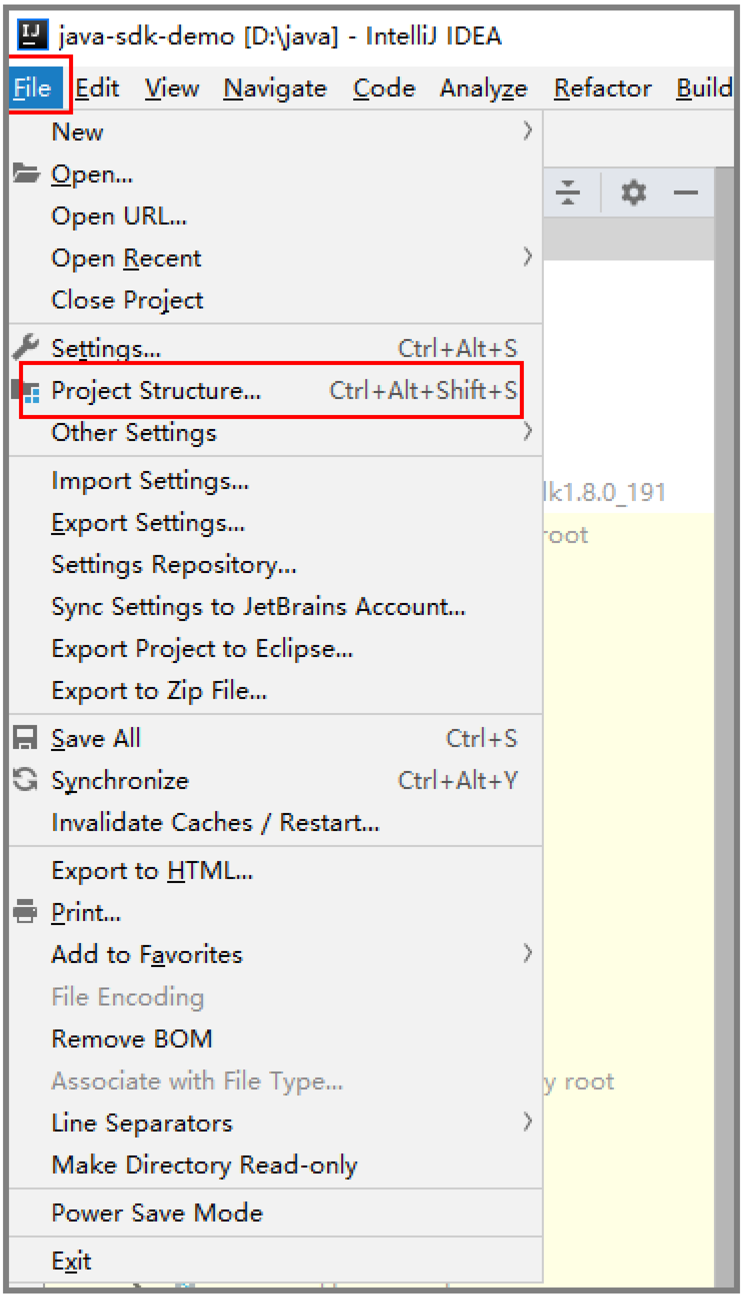
Project name: java-sdk-demo

Project location: D:\java

4. Import the **.jar** files in the Java SDK.

   a. Choose **File** > **Project Structure**. The **Project Structure** dialog box is displayed.

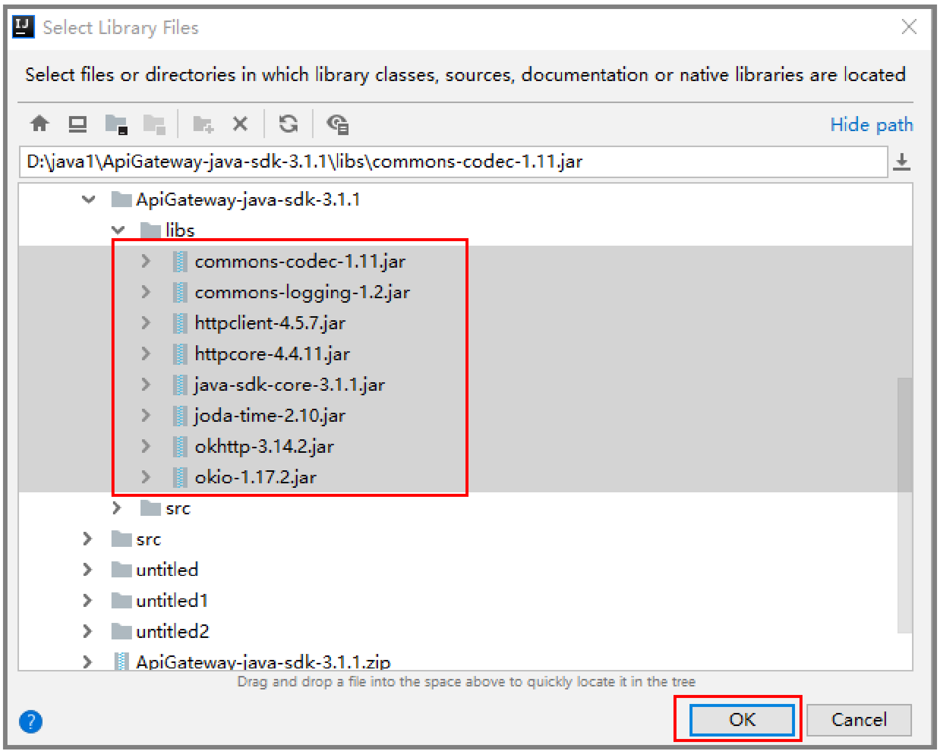
Figure 2-7 Importing the .jar files



b. In the **Project Structure** dialog box, choose **Libraries** > **+** > **Java**. The **Select Library Files** dialog box is displayed.

c. Select all **.jar** files in **\libs** of the directory where the SDK is located and click **OK**.

Figure 2-8 Selecting the .jar files



Select Library Files

Select files or directories in which library classes, sources, documentation or native libraries are located

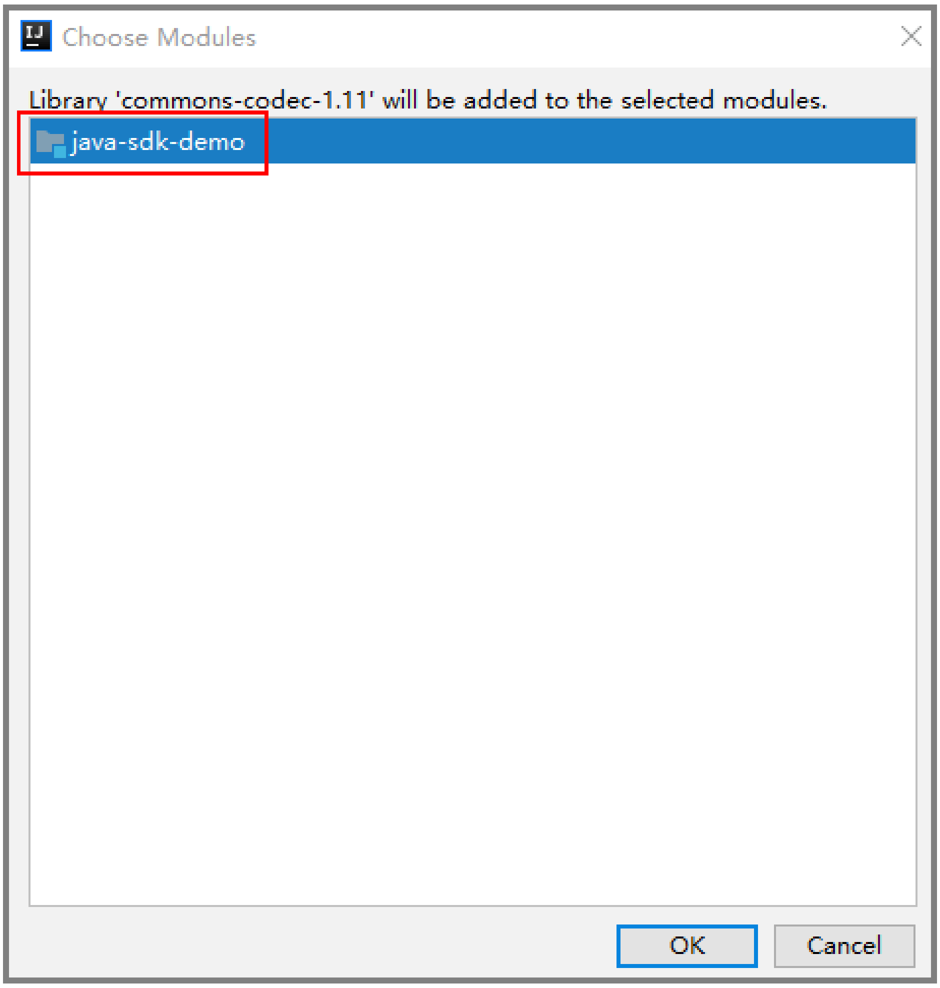
D:\java1\ApiGateway-java-sdk-3.1.1\libs\commons-codec-1.11.jar

- ApiGateway-java-sdk-3.1.1
  - libs
    - commons-codec-1.11.jar
    - commons-logging-1.2.jar
    - httpclient-4.5.7.jar
    - httpcore-4.4.11.jar
    - java-sdk-core-3.1.1.jar
    - joda-time-2.10.jar
    - okhttp-3.14.2.jar
    - okio-1.17.2.jar
  - src
- src
- untitled
- untitled1
- untitled2
- ApiGateway-java-sdk-3.1.1.zip

Drag and drop a file into the space above to quickly locate it in the tree

d. Select the project created in step 3 and click **OK**.

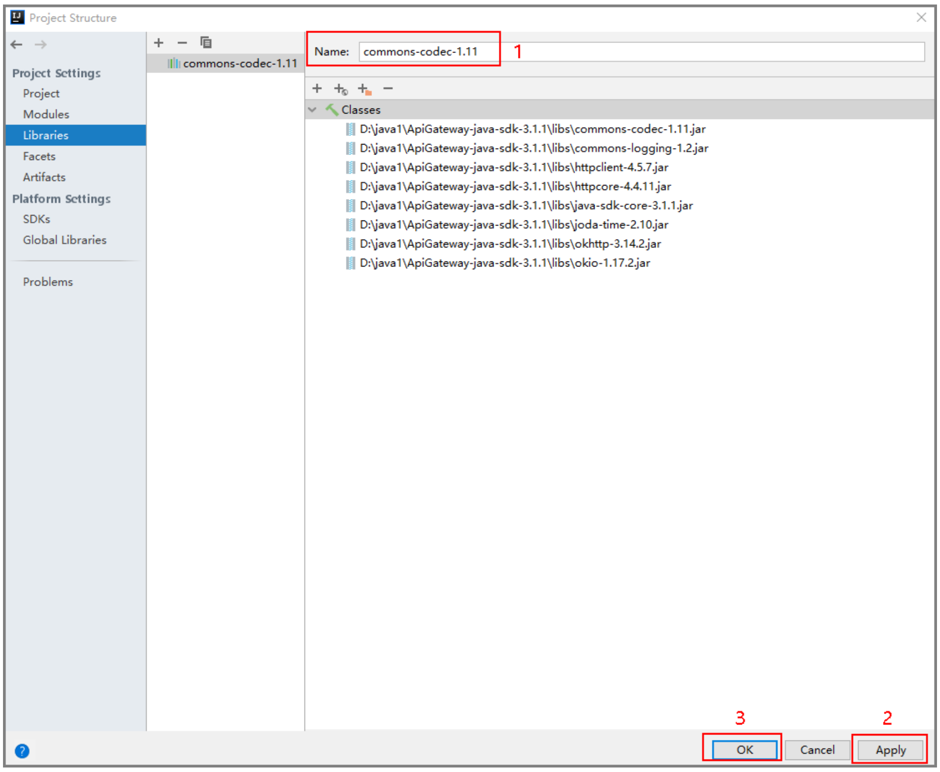
Figure 2-9 Selecting a project



Library 'commons-codec-1.11' will be added to the selected modules.

e. Enter the name of the directory where the JAR file is located and click **Apply** and **OK**.

Figure 2-10 JAR file directory

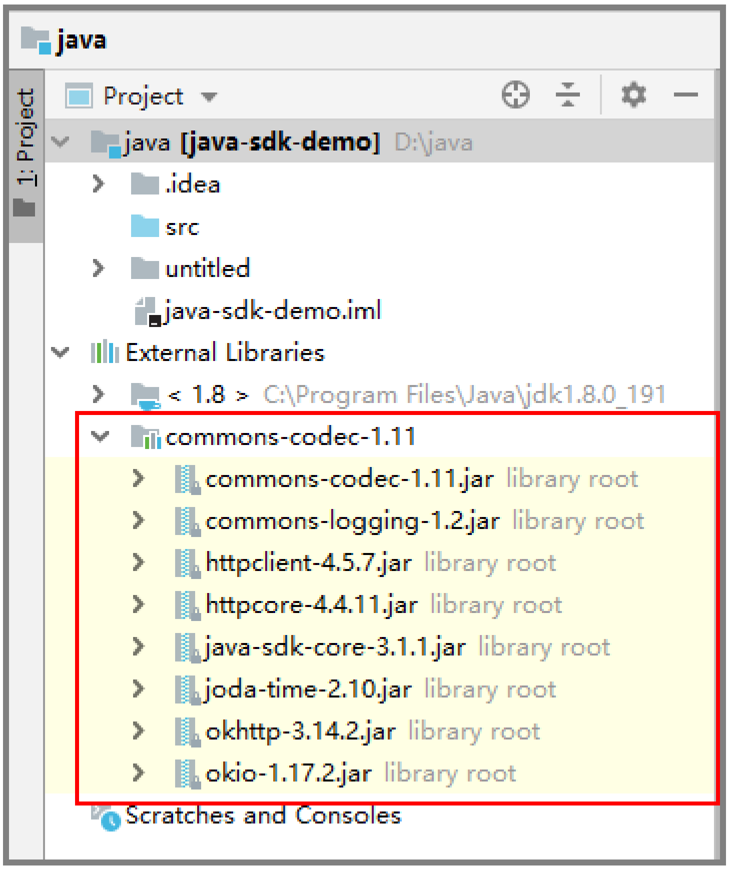


Name: commons-codec-1.11

```
D:\java1\ApiGateway-java-sdk-3.1.1\libs\commons-codec-1.11.jar
D:\java1\ApiGateway-java-sdk-3.1.1\libs\commons-logging-1.2.jar
D:\java1\ApiGateway-java-sdk-3.1.1\libs\httpclient-4.5.7.jar
D:\java1\ApiGateway-java-sdk-3.1.1\libs\httpcore-4.4.11.jar
D:\java1\ApiGateway-java-sdk-3.1.1\libs\java-sdk-core-3.1.1.jar
D:\java1\ApiGateway-java-sdk-3.1.1\libs\joda-time-2.10.jar
D:\java1\ApiGateway-java-sdk-3.1.1\libs\okhttp-3.14.2.jar
D:\java1\ApiGateway-java-sdk-3.1.1\libs\okio-1.17.2.jar
```

f. After the JAR file is imported, the directory structure is shown in the following figure.

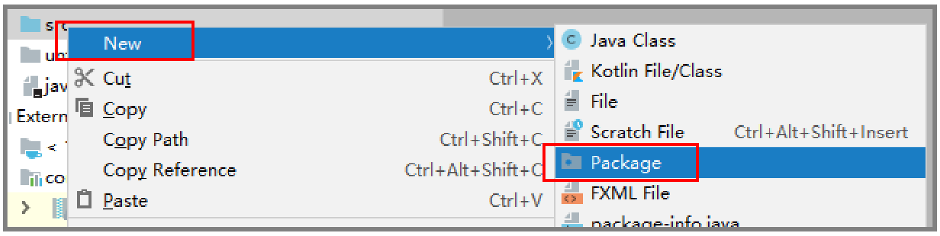
Figure 2-11 Directory structure


java
1: Project
Project
java [java-sdk-demo] D:\java
.idea
src
untitled
java-sdk-demo.iml
External Libraries
< 1.8 > C:\Program Files\Java\jdk1.8.0_191
commons-codec-1.11
commons-codec-1.11.jar library root
commons-logging-1.2.jar library root
httpclient-4.5.7.jar library root
httpcore-4.4.11.jar library root
java-sdk-core-3.1.1.jar library root
joda-time-2.10.jar library root
okhttp-3.14.2.jar library root
okio-1.17.2.jar library root
Scratches and Consoles

5. Create a package and a class named **Main**.

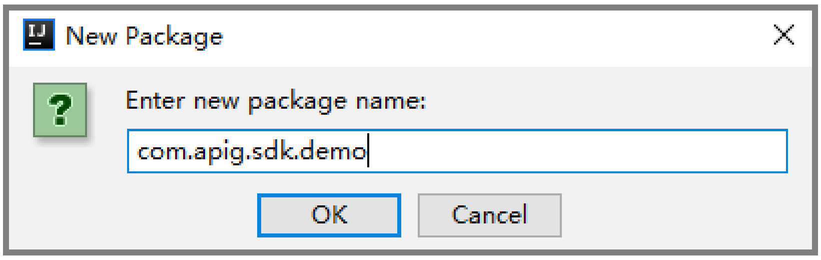
   a. Right-click **src** and choose **New** > **Package** from the shortcut menu.

   Figure 2-12 Creating a package



   b. Enter **com.apig.sdk.demo** for **Name**.

   Figure 2-13 Setting a package name

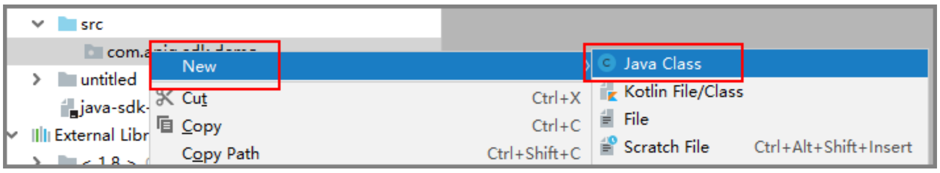


Enter new package name: com.apig.sdk.demo

   c. Click **OK**.

d. Right-click **com.apig.sdk.demo**, choose **New** > **Java Class** from the shortcut menu, enter **Main** in the **Name** text box, and click **OK**.

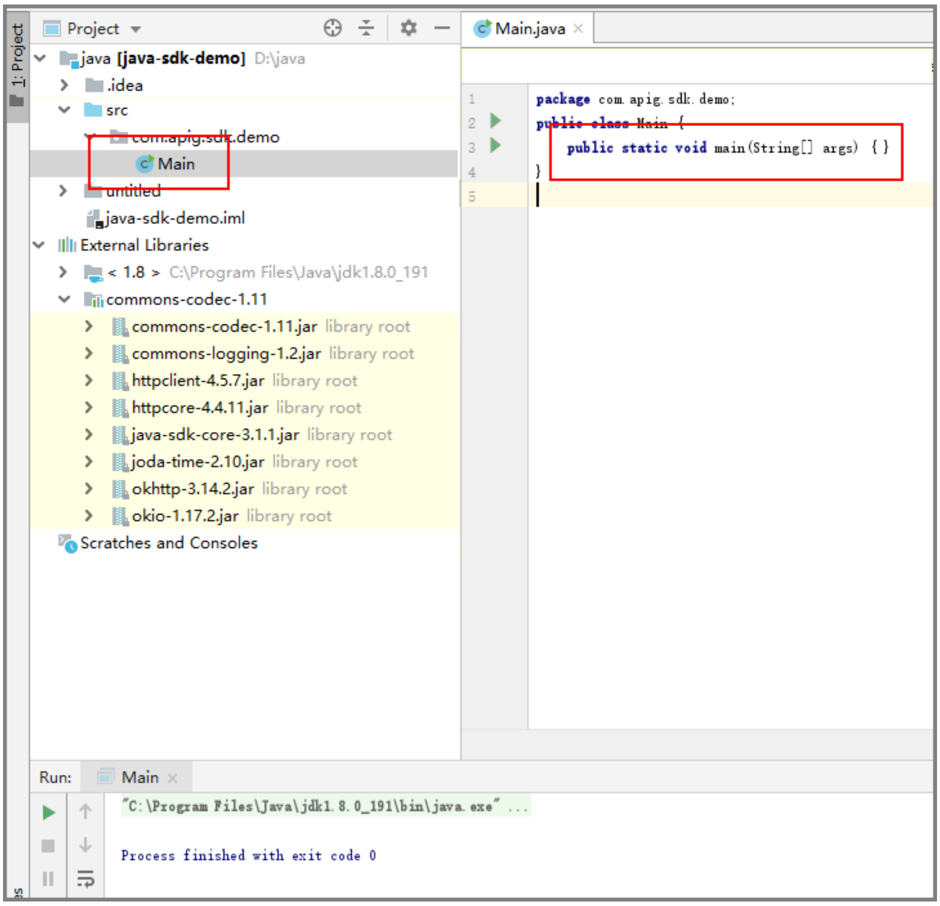
Figure 2-14 Creating a class



e. Configure a class.

After the class is created, open the Main file and add **public static void main(String[] args)** to the file.

Figure 2-15 Configuring the class

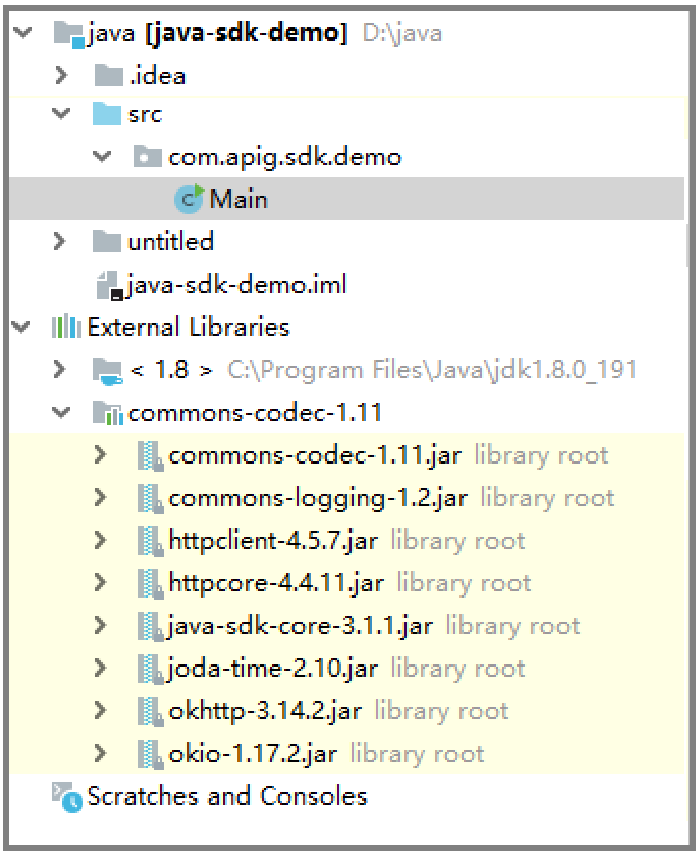

1: Project
Project
java [java-sdk-demo] D:\java
.idea
src
com.apig.sdk.demo
Main
untitled
java-sdk-demo.iml
External Libraries
< 1.8 > C:\Program Files\Java\jdk1.8.0_191
commons-codec-1.11
commons-codec-1.11.jar library root
commons-logging-1.2.jar library root
httpclient-4.5.7.jar library root
httpcore-4.4.11.jar library root
java-sdk-core-3.1.1.jar library root
joda-time-2.10.jar library root
okhttp-3.14.2.jar library root
okio-1.17.2.jar library root
Scratches and Consoles
Main.java
Run:
Main

```
package com.apig.sdk.demo;
public class Main {
    public static void main(String[] args) { }
}
```

```
"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...

Process finished with exit code 0
```

6. View the directory structure of the project.

Figure 2-16 Directory structure of the new project



```
java [java-sdk-demo] D:\java
  .idea
  src
    com.apig.sdk.demo
      Main
  untitled
  java-sdk-demo.iml
External Libraries
  < 1.8 > C:\Program Files\Java\jdk1.8.0_191
  commons-codec-1.11
    commons-codec-1.11.jar library root
    commons-logging-1.2.jar library root
    httpclient-4.5.7.jar library root
    httpcore-4.4.11.jar library root
    java-sdk-core-3.1.1.jar library root
    joda-time-2.10.jar library root
    okhttp-3.14.2.jar library root
    okio-1.17.2.jar library root
Scratches and Consoles
```

Before using **Main.java**, enter the required code according to **API Calling Example**.

API Calling Example

NOTE

- This section demonstrates how to access a published API.
- Before accessing an API, you must create and publish the API on the ROMA Connect console. You can specify the Mock backend for the API.
- The backend of this API is a fake HTTP service, which returns response code **200** and message body **Congratulations, sdk demo is running**.

1. Add the following references to **Main.java**:

```
import java.io.IOException;
import javax.net.ssl.SSLContext;

import org.apache.http.Header;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.methods.HttpRequestBase;
import org.apache.http.conn.ssl.AllowAllHostnameVerifier;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import org.apache.http.conn.ssl.SSLContexts;
import org.apache.http.conn.ssl.TrustSelfSignedStrategy;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;
import com.cloud.apigateway.sdk.utils.Client;
import com.cloud.apigateway.sdk.utils.Request;
```

2. Create a request with the following parameters. For details about how to obtain the values, see **Obtaining API Calling Information**.

   – **Key**: key of the credential authorized by the API. Set this parameter as required.

   – **Secret**: secret of the credential authorized by the API. Set this parameter as required.

     NOTE

     Hard-coded or plaintext AK and SK are risky. For security purposes, encrypt your AK and SK and store them in the configuration file or environment variables. In this example, the AK and SK are stored in environment variables. Before running the code in this example, configure environment variables **HUAWEICLOUD_SDK_AK** and **HUAWEICLOUD_SDK_SK**.

   – **Method**: request method. The sample code uses **POST**.

   – **url**: request URL of the API, excluding the **QueryString** and **fragment** parts. For the domain name, use your own independent domain name bound to the group to which the API belongs. The example code uses **http://serviceEndpoint/java-sdk** as an example.

   – **QueryStringParam**: query parameters carried in the URL. Characters (0-9a-zA-Z./;[]\-=~#%^&_+:") are allowed. The sample code uses **name=value**.

   – Header: request header. Set a request header as required. It cannot contain underscores (_). The sample code uses **Content-Type:text/plain**. If you are going to publish the API in a non-RELEASE environment, specify an environment name. The sample code uses **x-stage:publish_env_name**.

   – **body**: request body. The sample code uses **demo**.

   The sample code is as follows:

```
        Request request = new Request();
        try
        {
            // Directly writing AK/SK in code is risky. For security, encrypt your AK/SK and store them in the configuration file or environment variables.
            // In this example, the AK/SK are stored in environment variables for identity authentication.
            // Before running this example, set environment variables HUAWEICLOUD_SDK_AK and HUAWEICLOUD_SDK_SK.
            request.setKey(System.getenv("HUAWEICLOUD_SDK_AK"));
            request.setSecret(System.getenv("HUAWEICLOUD_SDK_SK"));
            request.setMethod("POST");
            request.setUrl("http://serviceEndpoint/java-sdk");
            request.addQueryStringParam("name", "value");
            request.addHeader("Content-Type", "text/plain");
            //request.addHeader("x-stage", "publish_env_name"); //If the API is published in an environment other than RELEASE, uncomment this line and add an environment name.
            request.setBody("demo");
        } catch (Exception e)
        {
            e.printStackTrace();
            return;
        }
```

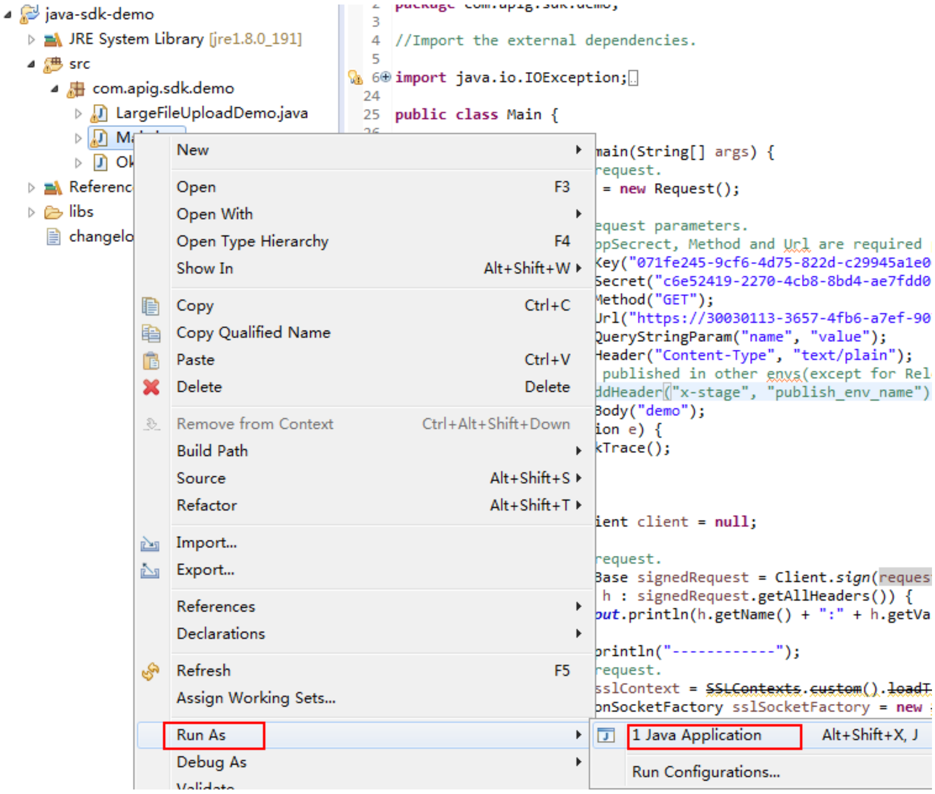
3. Sign the request, access the API, and print the result.

   The sample code is as follows:

```
        CloseableHttpClient client = null;
        try
        {
            HttpRequestBase signedRequest = Client.sign(request);

//If the subdomain name allocated by the system is used to access the API of HTTPS requests, 
uncomment the two lines of code to ignore the certificate verification.
            // SSLContext sslContext = SSLContexts.custom().loadTrustMaterial(null, new 
TrustSelfSignedStrategy()).useTLS().build();
            // SSLConnectionSocketFactory sslSocketFactory = new 
SSLConnectionSocketFactory(sslContext, new AllowAllHostnameVerifier());

//If the subdomain name allocated by the system is used to access the API of HTTPS requests, 
add .setSSLSocketFactory(sslSocketFactory) to the end of custom() to ignore the certificate 
verification.
            client = HttpClients.custom().build();

            HttpResponse response = client.execute(signedRequest);
            System.out.println(response.getStatusLine().toString());
            Header[] resHeaders = response.getAllHeaders();
            for (Header h : resHeaders)
            {
                System.out.println(h.getName() + ":" + h.getValue());
            }
            HttpEntity resEntity = response.getEntity();
            if (resEntity != null)
            {
                System.out.println(System.getProperty("line.separator") + EntityUtils.toString(resEntity, 
"UTF-8"));
            }

        } catch (Exception e)
        {
            e.printStackTrace();
        } finally
        {
            try
            {
                if (client != null)
                {
                    client.close();
                }
            } catch (IOException e)
            {
                e.printStackTrace();
            }
        }
```

4. Choose **Main.java**, right-click, and choose **Run As** > **Java Application** to run the project test code.

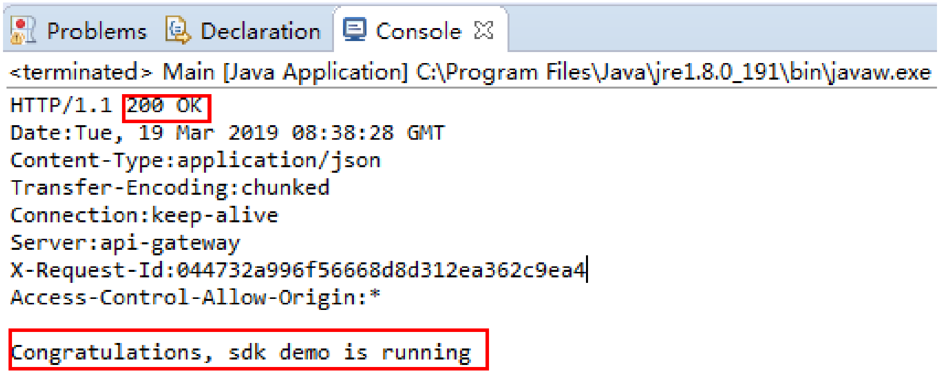
Figure 2-17 Running the project test code



```
package com.apig.sdk.demo;

//Import the external dependencies.

import java.io.IOException;

public class Main {
main(String[] args) {
request.
= new Request();

equest parameters.
ppSecrect, Method and Url are required
Key("071fe245-9cf6-4d75-822d-c29945a1e0
Secret("c6e52419-2270-4cb8-8bd4-ae7fdd0
Method("GET");
Url("https://30030113-3657-4fb6-a7ef-90
QueryStringParam("name", "value");
Header("Content-Type", "text/plain");
 published in other envs(except for Rel
ddHeader("x-stage", "publish_env_name")
Body("demo");
ion e) {
kTrace();

ient client = null;

request.
Base signedRequest = Client.sign(reques
h : signedRequest.getAllHeaders()) {
out.println(h.getName() + ":" + h.getVa

println("------------");
request.
sslContext = SSLContexts.custom().loadT
onSocketFactory sslSocketFactory = new
```

5. On the **Console** tab page, view the running result.

Figure 2-18 Response displayed if the calling is successful



```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_191\bin\javaw.exe
HTTP/1.1 200 OK
Date:Tue, 19 Mar 2019 08:38:28 GMT
Content-Type:application/json
Transfer-Encoding:chunked
Connection:keep-alive
Server:api-gateway
X-Request-Id:044732a996f56668d8d312ea362c9ea4
Access-Control-Allow-Origin:*

Congratulations, sdk demo is running
```

2.2.4 Go

Scenarios

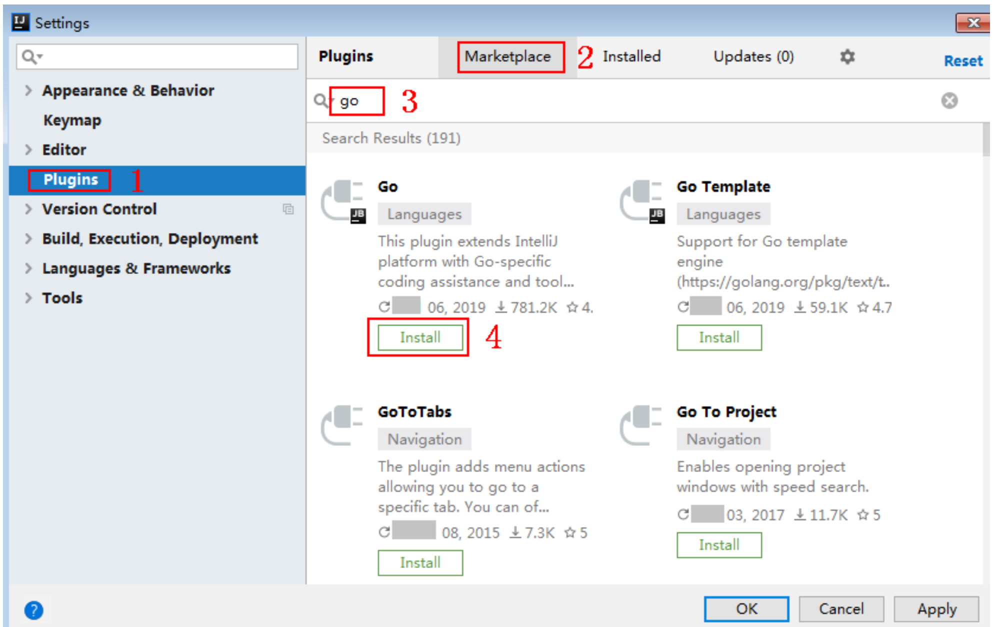
To use Go to call an API through App authentication, obtain the Go SDK, create a project, and then call the API by referring to the API calling example.

Prerequisites

- You have obtained API calling information. For details, see **Preparations**.
- You have installed the development tool and Go development environment. For details, see **Preparations**.

- You have installed the Go plug-in on IntelliJ IDEA. Otherwise, install it according to **Figure 2-19**.

Figure 2-19 Installing the Go plug-in



This plugin extends IntelliJ platform with Go-specific coding assistance and tool...

Support for Go template engine (https://golang.org/pkg/text/t..

The plugin adds menu actions allowing you to go to a specific tab. You can of...

Enables opening project windows with speed search.

Obtaining the SDK

Old version: Log in to the ROMA Connect console, choose **API Connect** > **API Calling** > **SDKs**, and download the SDK.

New version: Log in to the ROMA Connect console, choose **API Connect** > **Credentials** > **SDKs**, and download the SDK.

The following shows the directory structure after the decompression.

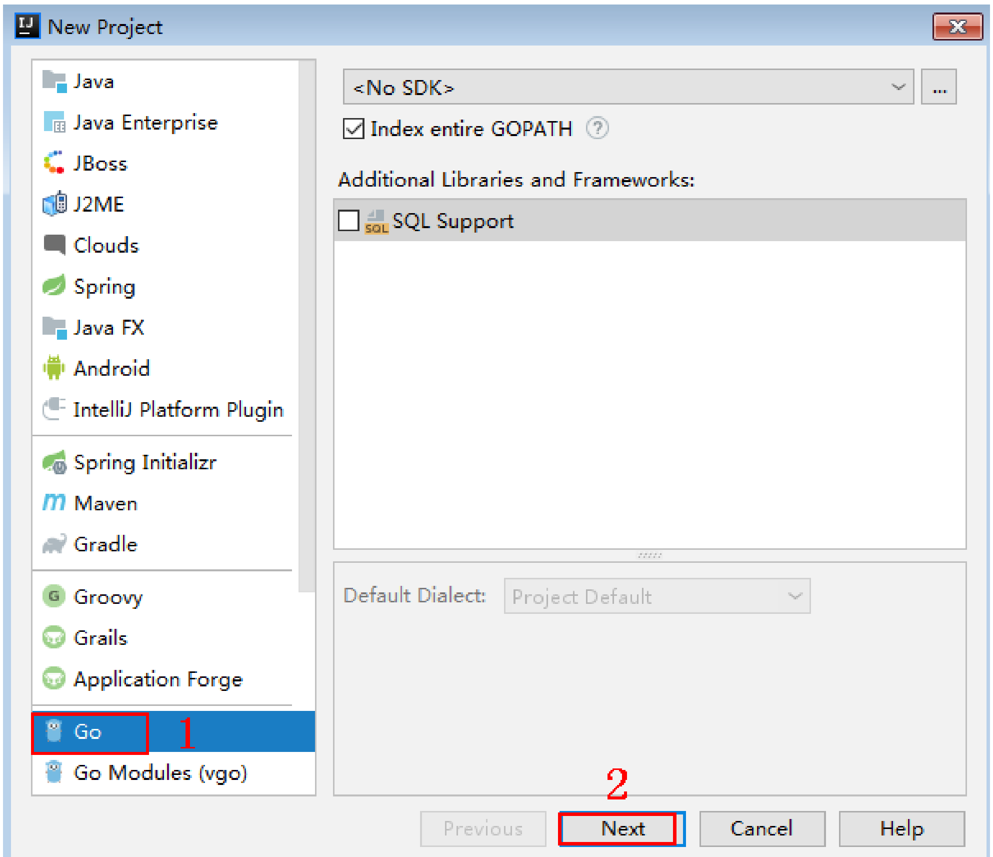
Name	Description
core\escape.go	SDK code
core\signer.go	
demo.go	Sample code

Creating a Project

1. Start IntelliJ IDEA and choose **File** > **New** > **Project**.

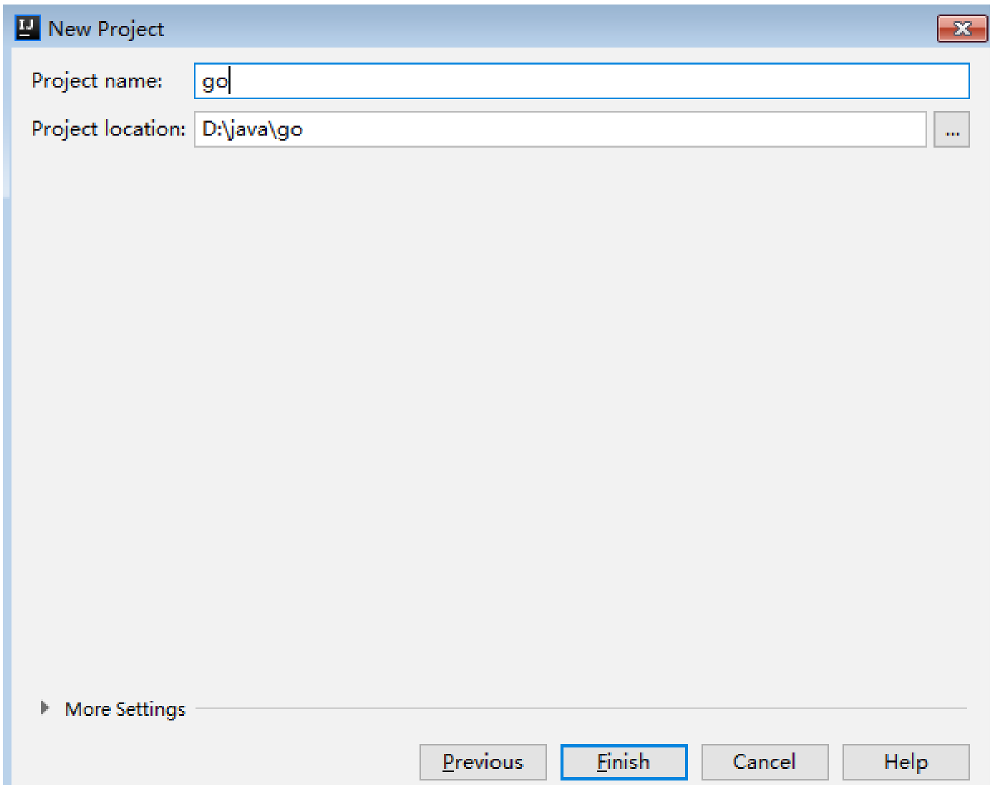
   On the displayed **New Project** page, choose **Go** and click **Next**.

Figure 2-20 New Project



2. Click **...**, select the directory where the SDK is decompressed, and click **Finish**.

Figure 2-21 Selecting the SDK directory after decompression

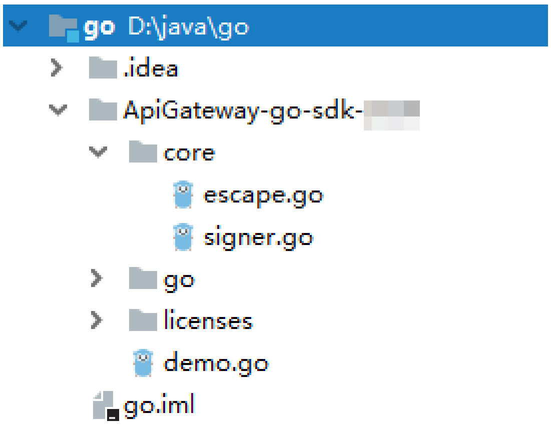


Project name: go

Project location: D:\java\go

3. View the directory structure of the project.

Figure 2-22 Directory structure of the new project



```
go D:\java\go
  .idea
  ApiGateway-go-sdk-[redacted]
    core
      escape.go
      signer.go
    go
    licenses
    demo.go
  go.iml
```

Modify the parameters in sample code **demo.go** as required. For details about the sample code, see **API Calling Example**.

API Calling Example

1. Import the Go SDK (signer.go) to the project.

```
import "apig-sdk/go/core"
```

2. Generate a signer and enter the key and secret of the authorized credential. For details about how to obtain the information, see **Obtaining API Calling Information**.

```
s := core.Signer{
// Directly writing AK/SK in code is risky. For security, encrypt your AK/SK and store them in the configuration file or environment variables.
 // In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, set environment variables HUAWEICLOUD_SDK_AK and HUAWEICLOUD_SDK_SK.
        Key: os.Getenv("HUAWEICLOUD_SDK_AK"),
        Secret:os.Getenv("HUAWEICLOUD_SDK_SK"),
}
```

3. Generate a request, and specify the domain name, method, request URL, query, and body. For details about how to obtain the information, see **Obtaining API Calling Information**.

```
r, _ := http.NewRequest("POST", "http://c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com/api?a=1&b=2",
                        ioutil.NopCloser(bytes.NewBuffer([]byte("foo=bar"))))
```

4. Add the **x-stage** header to the request to specify the environment. Add other headers to sign as required.

```
r.Header.Add("x-stage", "RELEASE")
```

5. Execute the following function to add the **X-Sdk-Date** and **Authorization** headers for signing:

```
s.Sign(r)
```

6. If the subdomain name allocated by the system is used to access the API of HTTPS requests, ignore the certificate verification. Otherwise, skip this step.

```
client:=&http.Client{
  Transport:&http.Transport{
    TLSClientConfig:&tls.Config{InsecureSkipVerify:true},
  },
}
```

7. Access the API and view the access result.

```
resp, err := http.DefaultClient.Do(r)
body, err := ioutil.ReadAll(resp.Body)
```

2.2.5 Python

Scenarios

To use Python to call an API through App authentication, obtain the Python SDK, create a project, and then call the API by referring to the API calling example.

Prerequisites

- You have obtained API calling information. For details, see **Preparations**.
- You have installed the development tool and Python development environment. For details, see **Preparations**.

  After Python is installed, run the **pip** command to install the **requests** library.

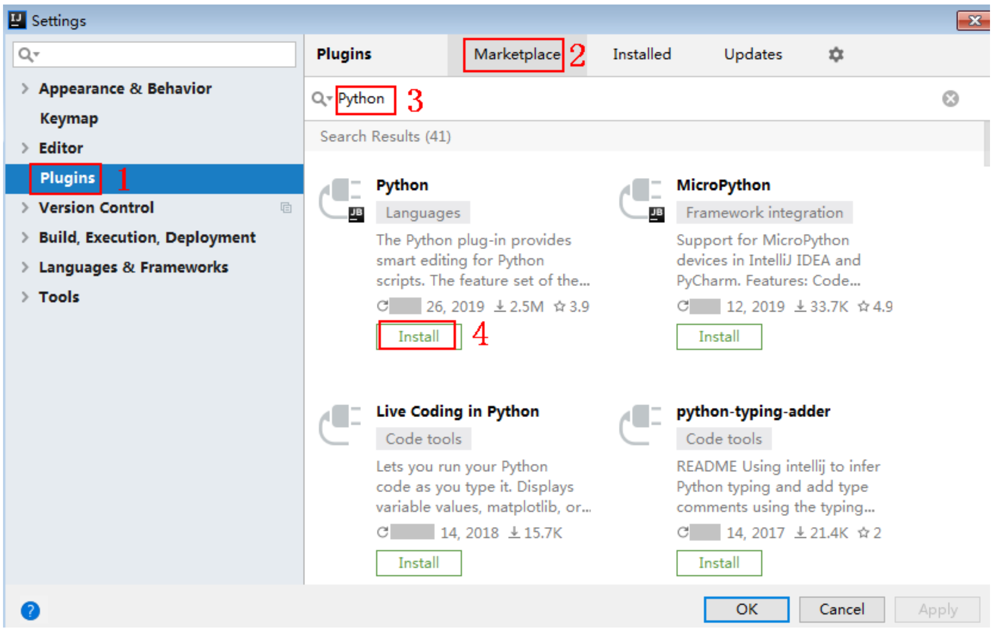
  ```
  pip install requests
  ```

  NOTE

  If a certificate error occurs during the installation, download the **get-pip.py** file to upgrade the pip environment, and try again.

- You have installed the Python plug-in on IntelliJ IDEA. Otherwise, install it according to **Figure 2-23**.

Figure 2-23 Installing the Python plug-in


Settings
Appearance & Behavior
Keymap
Editor
Plugins
1
Version Control
Build, Execution, Deployment
Languages & Frameworks
Tools
Plugins
Marketplace
2
Installed
Updates
Python
3
Search Results (41)
Python
Languages
The Python plug-in provides smart editing for Python scripts. The feature set of the...
26, 2019 2.5M 3.9
Install
4
MicroPython
Framework integration
Support for MicroPython devices in IntelliJ IDEA and PyCharm. Features: Code...
12, 2019 33.7K 4.9
Install
Live Coding in Python
Code tools
Lets you run your Python code as you type it. Displays variable values, matplotlib, or...
14, 2018 15.7K
Install
python-typing-adder
Code tools
README Using intellij to infer Python typing and add type comments using the typing...
14, 2017 21.4K 2
Install
OK
Cancel
Apply

Obtaining the SDK

Old version: Log in to the ROMA Connect console, choose **API Connect** > **API Calling** > **SDKs**, and download the SDK.

New version: Log in to the ROMA Connect console, choose **API Connect** > **Credentials** > **SDKs**, and download the SDK.

The following shows the directory structure after the decompression.

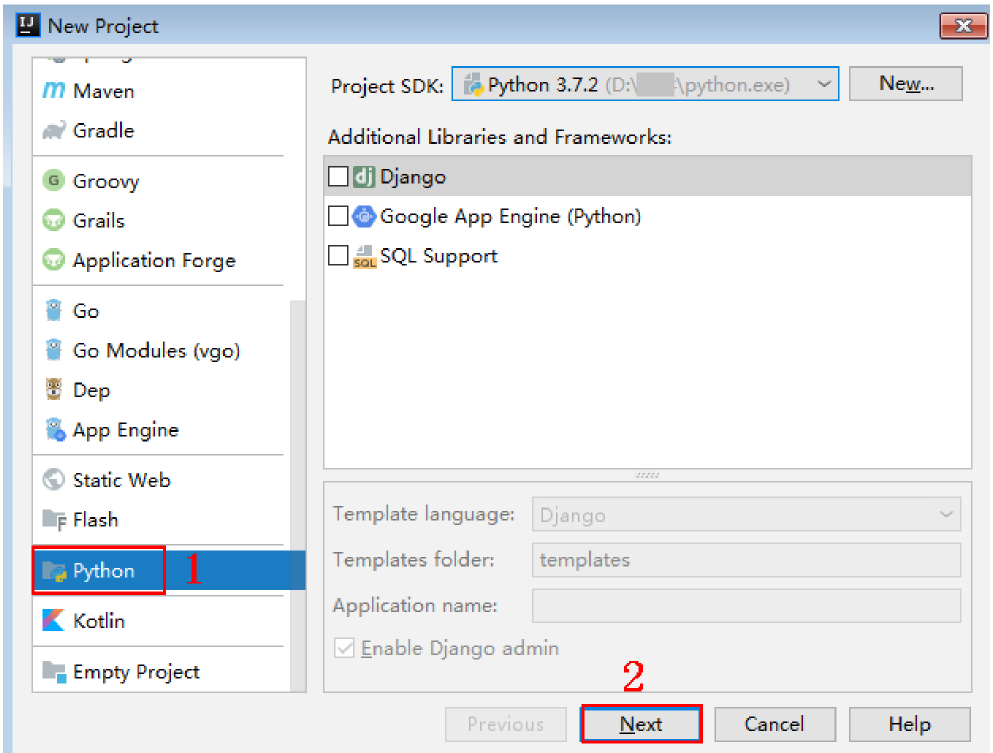
Name	Description
apig_sdk\__init__.py	SDK code
apig_sdk\signer.py	
main.py	Sample code
backend_signature.py	Sample code for backend signing
licenses\license-requests	Third-party licenses

Creating a Project

1. Start IDEA and choose **File** > **New** > **Project**.

   On the displayed **New Project** page, choose **Python** and click **Next**.

Figure 2-24 New Project



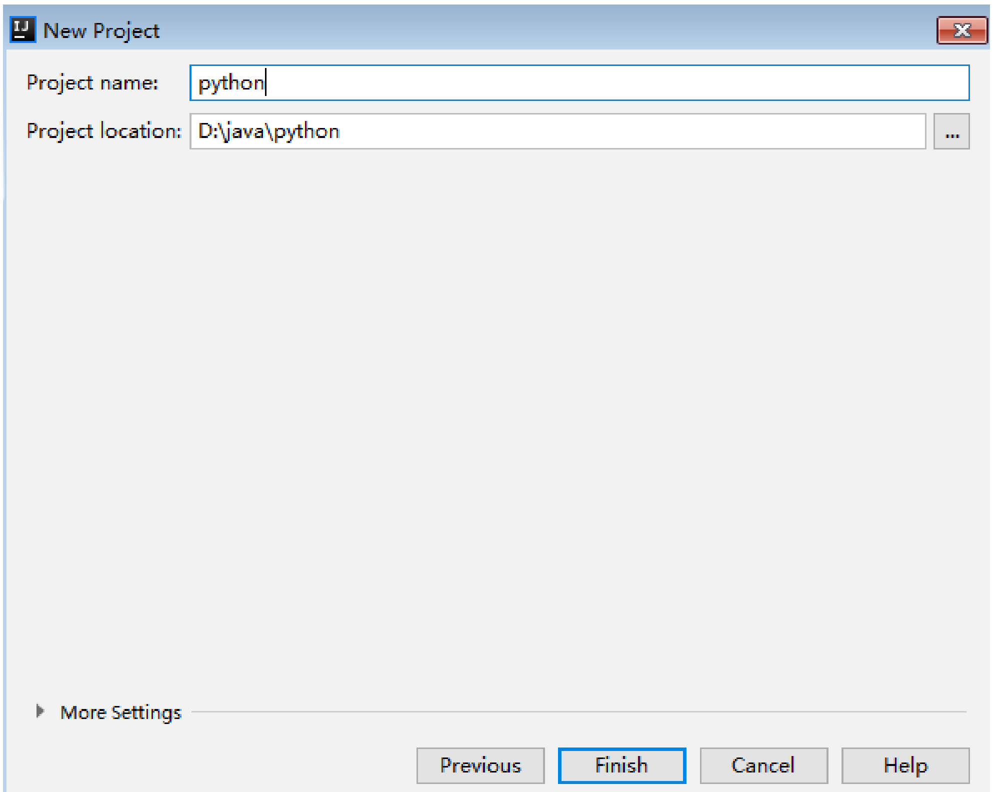
Project SDK: Python 3.7.2 (D:\[redacted]\python.exe)

□ Django
□ Google App Engine (Python)
□ SQL Support

Template language: Django
Templates folder: templates
Application name:
☑ Enable Django admin

2. Click **Next**. Click **...**, select the directory where the SDK is decompressed, and click **Finish**.

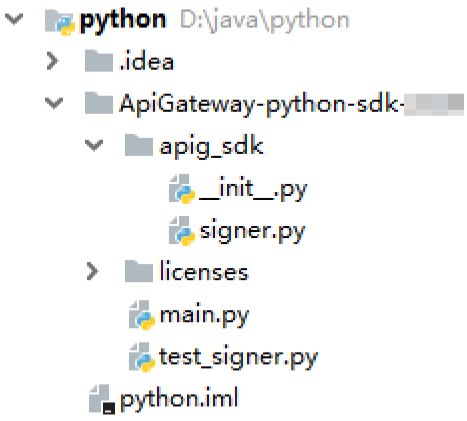
Figure 2-25 Selecting the SDK directory after decompression



Project name: python
Project location: D:\java\python

3. View the directory structure of the project.

Figure 2-26 Directory structure of the new project



```
python D:\java\python
  .idea
  ApiGateway-python-sdk-[redacted]
    apig_sdk
      __init__.py
      signer.py
    licenses
    main.py
    test_signer.py
  python.iml
```

Modify the parameters in sample code **main.py** as required. For details about the sample code, see **API Calling Example**.

API Calling Example

1. Import **apig_sdk** to the project.

```
from apig_sdk import signer
import requests
import os
```

2. Generate a signer and enter the key and secret of the authorized credential. For details about how to obtain the information, see **Obtaining API Calling Information**.

```
sig = signer.Signer()
// Directly writing AK/SK in code is risky. For security, encrypt your AK/SK and store them in the configuration file or environment variables.
// In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, set environment variables HUAWEICLOUD_SDK_AK and HUAWEICLOUD_SDK_SK.
sig.Key = os.getenv('HUAWEICLOUD_SDK_AK')
sig.Secret = os.getenv('HUAWEICLOUD_SDK_SK')
```

3. Generate a request, and specify the method, request URL, header, and body. For details about how to obtain the information, see **Obtaining API Calling Information**.

```
r = signer.HttpRequest("POST",
                       "https://c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com/app1?a=1",
                       {"x-stage": "RELEASE"},
                       "body")
```

4. Execute the following function to add the X-Sdk-Date and Authorization headers for signing:

NOTE

X-Sdk-Date is a request header parameter required for signing requests.

```
sig.Sign(r)
```

5. Access the API and view the access result.

```
//If the subdomain name allocated by the system is used to access the API of HTTPS requests, add ,verify=False to the end of data=r.body to ignore the certificate verification.
resp = requests.request(r.method, r.scheme + "://" + r.host + r.uri, headers=r.headers, data=r.body)
```

```
print(resp.status_code, resp.reason)
print(resp.content)
```

2.2.6 C#

Scenarios

To use C# to call an API through App authentication, obtain the C# SDK, open the project file in the SDK, and then call the API by referring to the API calling example.

Prerequisites

- You have obtained API calling information. For details, see **Preparations**.
- You have installed the C# development environment. For details, see **Preparations**.

Obtaining the SDK

Old version: Log in to the ROMA Connect console, choose **API Connect** > **API Calling** > **SDKs**, and download the SDK.

New version: Log in to the ROMA Connect console, choose **API Connect** > **Credentials** > **SDKs**, and download the SDK.

The following shows the directory structure after the decompression.

Name	Description
apigateway-signature\Signer.cs	SDK code
apigateway-signature\HttpEncoder.cs	
sdk-request\Program.cs	Sample code for signing requests
backend-signature\	Sample project for backend signing
csharp.sln	Project file
licenses\license-referencesource	Third-party licenses

Opening the Sample Project

Double-click **csharp.sln** in the SDK package to open the project. The project contains the following:

- **apigateway-signature**: Shared library that implements the signature algorithm. It can be used in the .Net Framework and .Net Core projects.
- **backend-signature**: Example of a backend service signature.

- **sdk-request**: Example of invoking the signature algorithm. Modify the parameters as required. For details about the sample code, see **API Calling Example**.

API Calling Example

1. Import the SDK to the project.

```
using APIGATEWAY_SDK;
```

2. Generate a signer and enter the key and secret of the authorized credential. For details about how to obtain the information, see **Obtaining API Calling Information**.

```
Signer signer = new Signer();
// Directly writing AK/SK in code is risky. For security, encrypt your AK/SK and store them in the
configuration file or environment variables.
// In this example, the AK/SK are stored in environment variables for identity authentication. Before
running this example, set environment variables HUAWEICLOUD_SDK_AK and
HUAWEICLOUD_SDK_SK.
signer.Key = Environment.GetEnvironmentVariable("HUAWEICLOUD_SDK_AK");
signer.Secret = Environment.GetEnvironmentVariable("HUAWEICLOUD_SDK_SK");
```

3. Generate a request, and specify the method, request URL, and body. For details about how to obtain the information, see **Obtaining API Calling Information**.

```
HttpRequest r = new HttpRequest("POST",
              new Uri("https://c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com/
app1?query=value"));
r.body = "{\"a\":1}";
```

4. Add the **x-stage** header to the request to specify the environment. Add other headers to sign as required.

```
r.headers.Add("x-stage", "RELEASE");
```

5. Execute the following function to generate **HttpWebRequest**, and add the **X-Sdk-Date** and **Authorization** headers for signing the request:

```
HttpWebRequest req = signer.Sign(r);
```

6. If the subdomain name allocated by the system is used to access the API of HTTPS requests, ignore the certificate verification. Otherwise, skip this step.

```
System.Net.ServicePointManager.ServerCertificateValidationCallback = new
System.Net.Security.RemoteCertificateValidationCallback(delegate { return true; });
```

7. Access the API and view the access result.

```
var writer = new StreamWriter(req.GetRequestStream());
writer.Write(r.body);
writer.Flush();
HttpWebResponse resp = (HttpWebResponse)req.GetResponse();
var reader = newStreamReader(resp.GetResponseStream());
Console.WriteLine(reader.ReadToEnd());
```

2.2.7 JavaScript

Scenarios

To use JavaScript to call an API through App authentication, obtain the JavaScript SDK, create a project, and then call the API by referring to the API calling example.

The JavaScript SDK supports two operating environments: Node.js and browsers. This section uses Node.js as an example.

Prerequisites

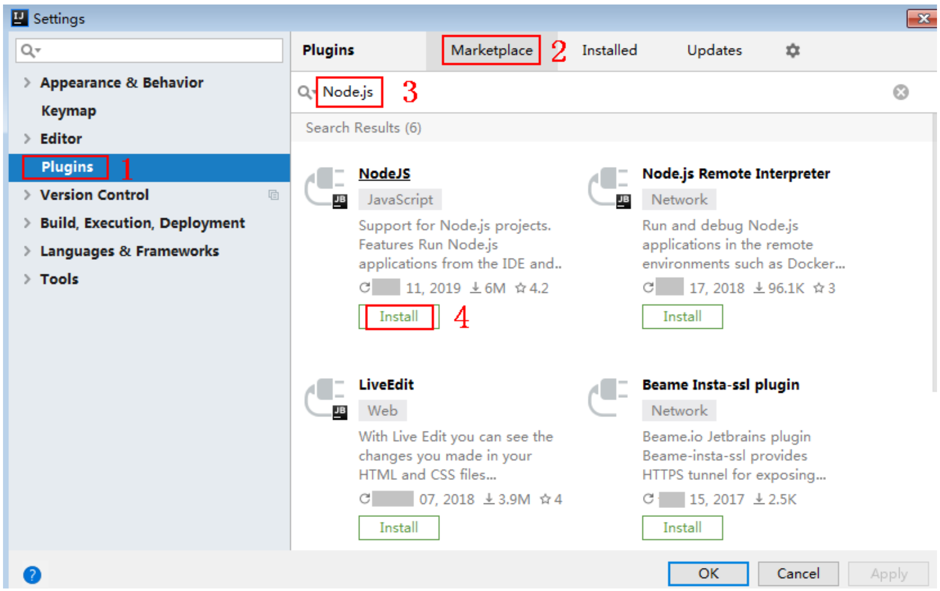
- You have obtained API calling information. For details, see **Preparations**.

- You have installed the development tool and JavaScript development environment. For details, see **Preparations**.
  - After Node.js is installed, run the **npm** command to install the **moment** and **moment-timezone** modules.

```
npm install moment --save
npm install moment-timezone --save
```

  - You have installed the Node.js plug-in on IntelliJ IDEA. Otherwise, install it according to **Figure 2-27**.

Figure 2-27 Installing the Node.js plug-in


Settings
Appearance & Behavior
Keymap
Editor
Plugins
1
Version Control
Build, Execution, Deployment
Languages & Frameworks
Tools
Plugins
Marketplace
2
Installed
Updates
Node.js
3
Search Results (6)
NodeJS
JavaScript
Support for Node.js projects. Features Run Node.js applications from the IDE and..
11, 2019 6M 4.2
Install
4
Node.js Remote Interpreter
Network
Run and debug Node.js applications in the remote environments such as Docker...
17, 2018 96.1K 3
Install
LiveEdit
Web
With Live Edit you can see the changes you made in your HTML and CSS files...
07, 2018 3.9M 4
Install
Beame Insta-ssl plugin
Network
Beame.io Jetbrains plugin Beame-insta-ssl provides HTTPS tunnel for exposing...
15, 2017 2.5K
Install
OK
Cancel
Apply

- The browser must be Chrome 89.0 or later.

Obtaining the SDK

Old version: Log in to the ROMA Connect console, choose **API Connect** > **API Calling** > **SDKs**, and download the SDK.

New version: Log in to the ROMA Connect console, choose **API Connect** > **Credentials** > **SDKs**, and download the SDK.

The following shows the directory structure after the decompression.

Name	Description
signer.js	SDK code
node_demo.js	Node.js sample code
demo.html	Browser sample code
demo_require.html	Browser sample code (loaded using **require**)
test.js	Test Cases
js\hmac-sha256.js	Dependencies
licenses\license-crypto-js	Third-party licenses

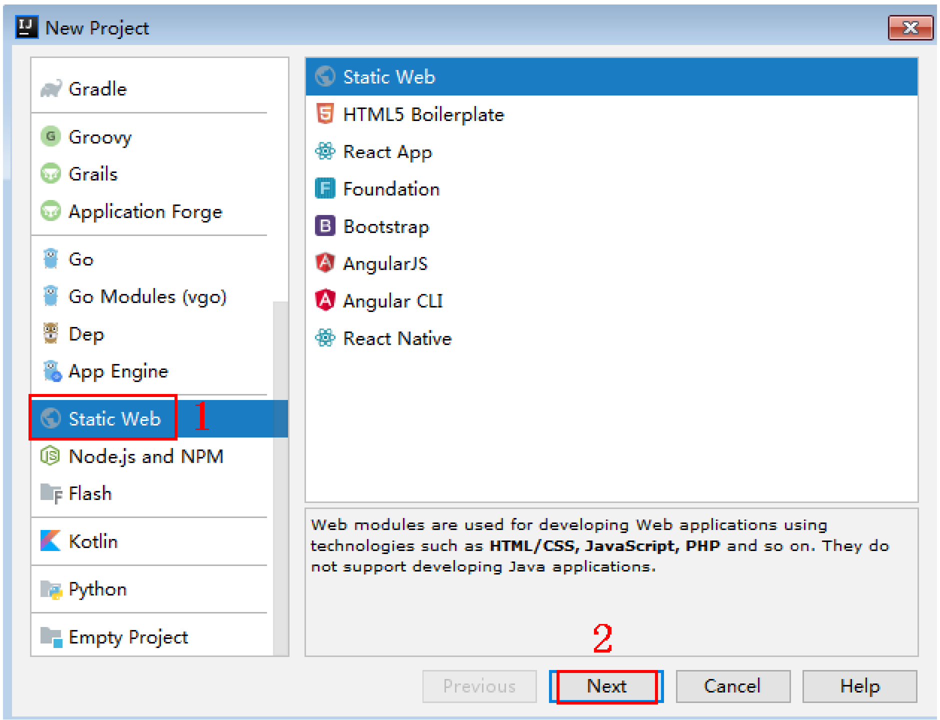
Name	Description
licenses\license-node	

Creating a Project

1. Start IntelliJ IDEA and choose **File** > **New** > **Project**.

   In the **New Project** dialog box, choose **Static Web** and click **Next**.

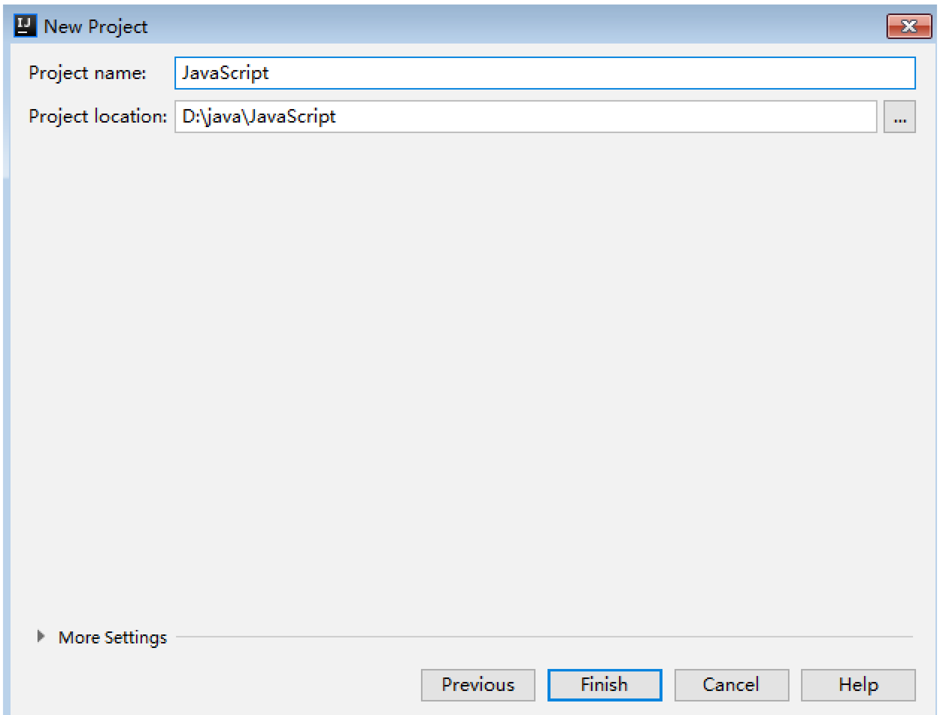
   Figure 2-28 New Project

   Web modules are used for developing Web applications using technologies such as **HTML/CSS, JavaScript, PHP** and so on. They do not support developing Java applications.

2. Click **...**, select the directory where the SDK is decompressed, and click **Finish**.

Figure 2-29 Selecting the SDK directory after decompression



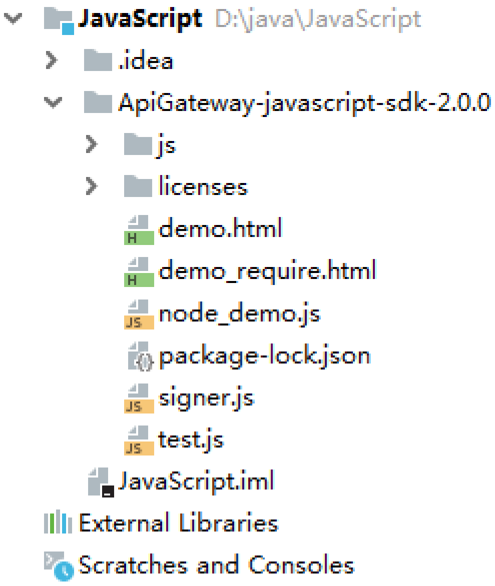
New Project

Project name: JavaScript

Project location: D:\java\JavaScript

3. View the directory structure of the project.

Figure 2-30 Directory structure of the new project

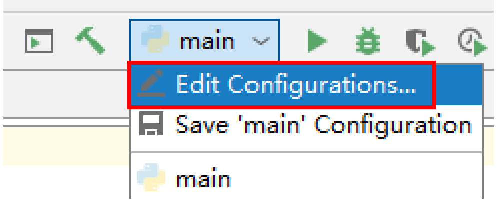


```
JavaScript D:\java\JavaScript
  .idea
  ApiGateway-javascript-sdk-2.0.0
    js
    licenses
    demo.html
    demo_require.html
    node_demo.js
    package-lock.json
    signer.js
    test.js
  JavaScript.iml
External Libraries
Scratches and Consoles
```

- **node_demo.js**: Sample code in Node.js. Modify the parameters in the sample code as required. For details about the sample code, see **API Calling Example (Node.js)**.
- **demo.html**: Browser sample code. Modify the parameters in the sample code as required. For details about the sample code, see **API Calling Example (Browser)**.

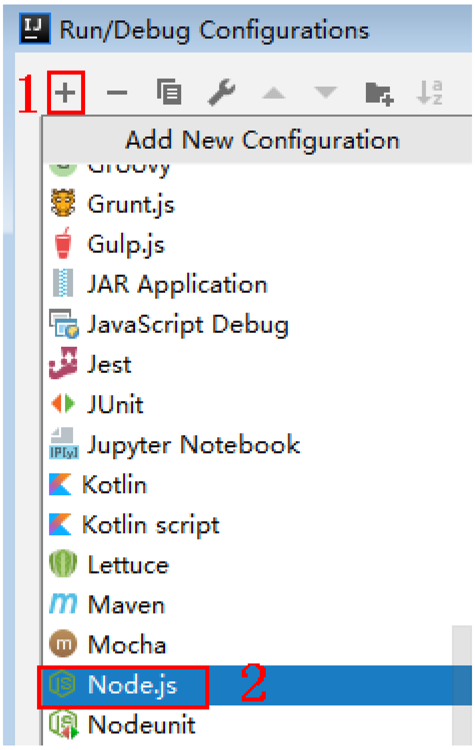
4. Click **Edit Configurations**.

Figure 2-31 Edit Configurations



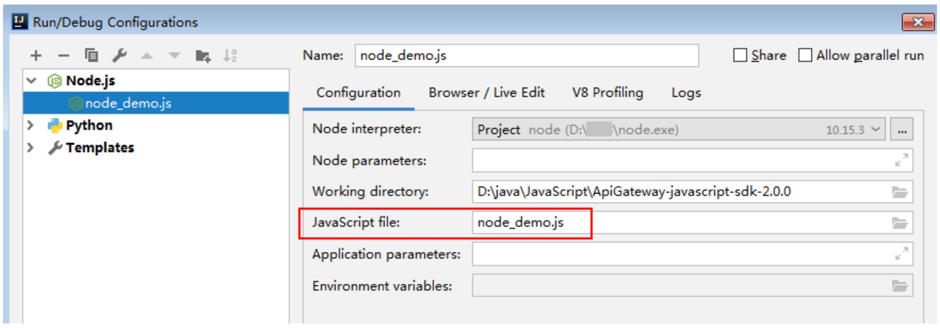
5. Click **+** and select **Node.js**.

Figure 2-32 Selecting Node.js



6. Set **JavaScript file** to **node_demo.js** and click **OK**.

Figure 2-33 Selecting node_demo.js



Name: node_demo.js

□ Share □ Allow parallel run

Node interpreter: Project node (D:\[redacted]\node.exe) 10.15.3

Node parameters:

Working directory: D:\java\JavaScript\ApiGateway-javascript-sdk-2.0.0

JavaScript file: node_demo.js

Application parameters:

Environment variables:

API Calling Example (Node.js)

1. Import **signer.js** to your project.

```
var signer = require('./signer')
var http = require('http')
```

2. Generate a signer and enter the key and secret of the authorized credential. For details about how to obtain the information, see **Obtaining API Calling Information**.

   ```
   var sig = new signer.Signer()
   // Directly writing AK/SK in code is risky. For security, encrypt your AK/SK and store them in the configuration file or environment variables.
   // In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, set environment variables HUAWEICLOUD_SDK_AK and HUAWEICLOUD_SDK_SK.
   sig.Key = process.env.HUAWEICLOUD_SDK_AK
   sig.Secret = process.env.HUAWEICLOUD_SDK_SK
   ```

3. Generate a request, and specify the method, request URL, and body. For details about how to obtain the information, see **Obtaining API Calling Information**.

   ```
   var r = new signer.HttpRequest("POST", "c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com/app1?a=1");
   r.body = '{"a":1}'
   ```

4. Add the **x-stage** header to the request to specify the environment. Add other headers to sign as required.

   ```
   r.headers = { "x-stage":"RELEASE" }
   ```

5. Execute the following function to generate HTTP(S) request parameters, and add the **X-Sdk-Date** and **Authorization** headers for signing the request:

   ```
   var opts = sig.Sign(r)
   ```

6. Access the API and view the access result. If you access the API using HTTPS, change **http.request** to **https.request**.

   ```
   var req=http.request(opts, function(res){
           console.log(res.statusCode)
           res.on("data",    function(chunk){
           console.log(chunk.toString())
       })
   })
   req.on("error",function(err){
       console.log(err.message)
   })
   req.write(r.body)
   req.end()
   ```

API Calling Example (Browser)

To use a browser to access APIs, you need to register an API that supports the OPTIONS method. For details, see **Creating an API in OPTIONS Mode**. The response header contains Access-Control-Allow-* headers. You can add these headers by enabling CORS when creating an API.

1. Import **signer.js** and dependencies to the HTML page.

   ```
   <script src="js/hmac-sha256.js"></script>
   <script src="js/moment.min.js"></script>
   <script src="js/moment-timezone-with-data.min.js"></script>
   <script src='signer.js'></script>
   ```

2. Sign the request and access the API.

   ```
   var sig = new signer.Signer()
   sig.Key = process.env.HUAWEICLOUD_SDK_AK
   sig.Secret = process.env.HUAWEICLOUD_SDK_SK
   var r= new signer.HttpRequest()
   r.host = "c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com"
   r.method = "POST"
   r.uri = "/app1"
   r.body = '{"a":1}'
   r.query = { "a":"1","b":"2" }
   r.headers = { "Content-Type":"application/json" }
   var opts = sig.Sign(r)
   ```

```
var scheme = "https"
$.ajax({
    type: opts.method,
    data: req.body,
    processData: false,
    url: scheme + "://" + opts.hostname + opts.path,
    headers: opts.headers,
    success: function (data) {
        $('#status').html('200')
        $('#recv').html(data)
    },
    error: function (resp) {
        if (resp.readyState === 4) {
            $('#status').html(resp.status)
            $('#recv').html(resp.responseText)
        } else {
            $('#status').html(resp.state())
        }
    },
    timeout: 1000
});
```

2.2.8 PHP

Scenarios

To use PHP to call an API through App authentication, obtain the PHP SDK, create a project, and then call the API by referring to the API calling example.

This section uses IntelliJ IDEA 2018.3.5 as an example.

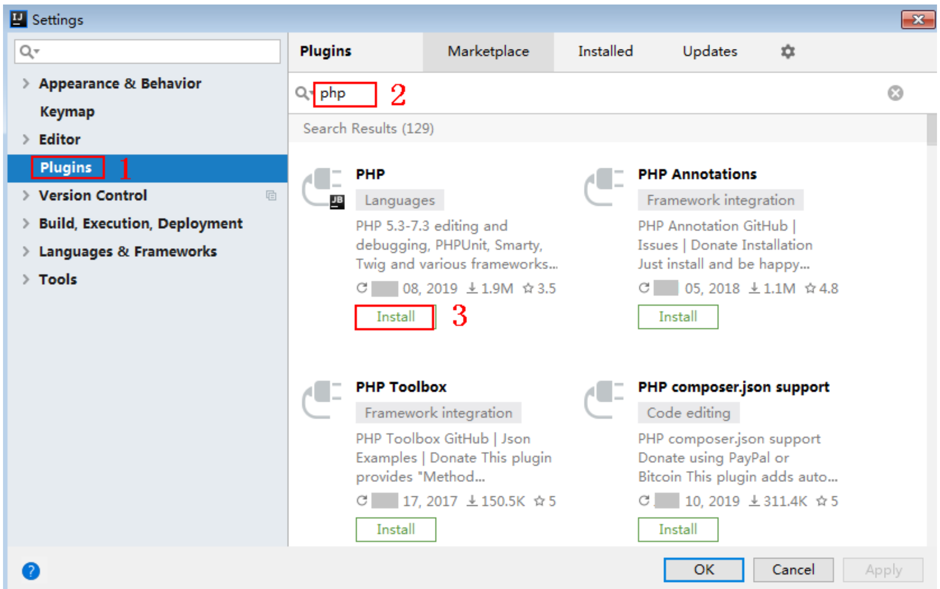
Prerequisites

- You have obtained API calling information. For details, see **Preparations**.
- You have installed the development tool and PHP development environment. For details, see **Preparations**.
  - Copy the **php.ini-production** file from the PHP installation directory to the **C:\windows\** directory, rename the file as **php.ini**, and then add the following lines to the file:

    ```
    extension_dir = "PHP installation directory/ext"
    extension=openssl
    extension=curl
    ```

  - You have installed the PHP plug-in on IntelliJ IDEA. Otherwise, install it according to **Figure 2-34**.

Figure 2-34 Installing the PHP plug-in


Settings
Appearance & Behavior
Keymap
Editor
Plugins
1
Version Control
Build, Execution, Deployment
Languages & Frameworks
Tools
Plugins
Marketplace
Installed
Updates
php
2
Search Results (129)
PHP
Languages
PHP 5.3-7.3 editing and debugging, PHPUnit, Smarty, Twig and various frameworks...
08, 2019 1.9M 3.5
Install
3
PHP Annotations
Framework integration
PHP Annotation GitHub | Issues | Donate Installation Just install and be happy...
05, 2018 1.1M 4.8
Install
PHP Toolbox
Framework integration
PHP Toolbox GitHub | Json Examples | Donate This plugin provides "Method...
17, 2017 150.5K 5
Install
PHP composer.json support
Code editing
PHP composer.json support Donate using PayPal or Bitcoin This plugin adds auto...
10, 2019 311.4K 5
Install
OK
Cancel
Apply

Obtaining the SDK

Old version: Log in to the ROMA Connect console, choose **API Connect** > **API Calling** > **SDKs**, and download the SDK.

New version: Log in to the ROMA Connect console, choose **API Connect** > **Credentials** > **SDKs**, and download the SDK.

The following shows the directory structure after the decompression.

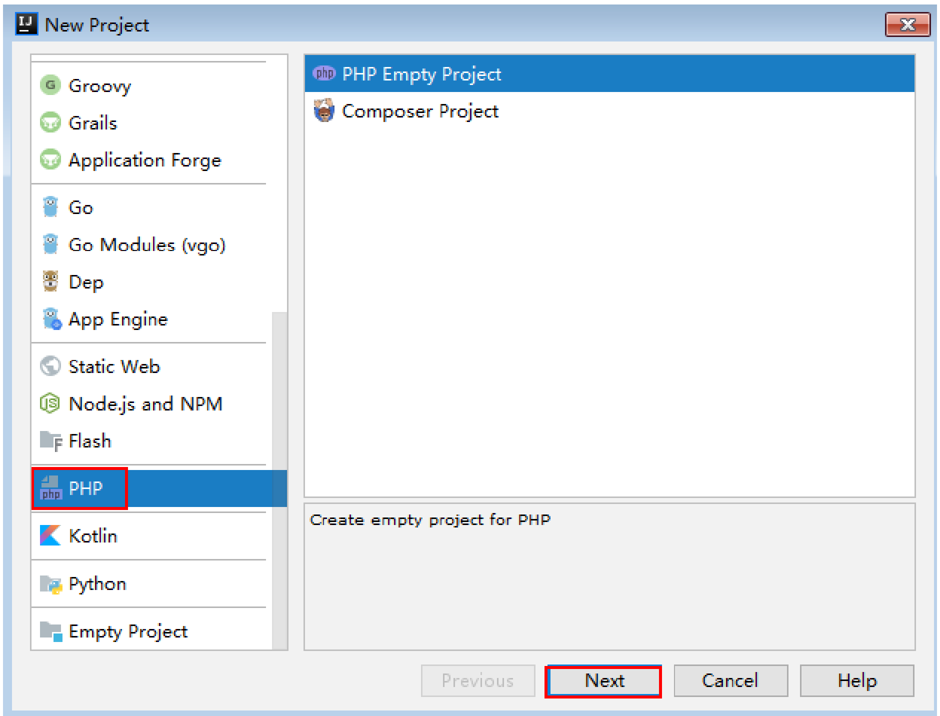
Name	Description
signer.php	SDK code
index.php	Sample code

Creating a Project

1. Start IDEA and choose **File** > **New** > **Project**.

   On the displayed **New Project** page, choose **PHP** and click **Next**.

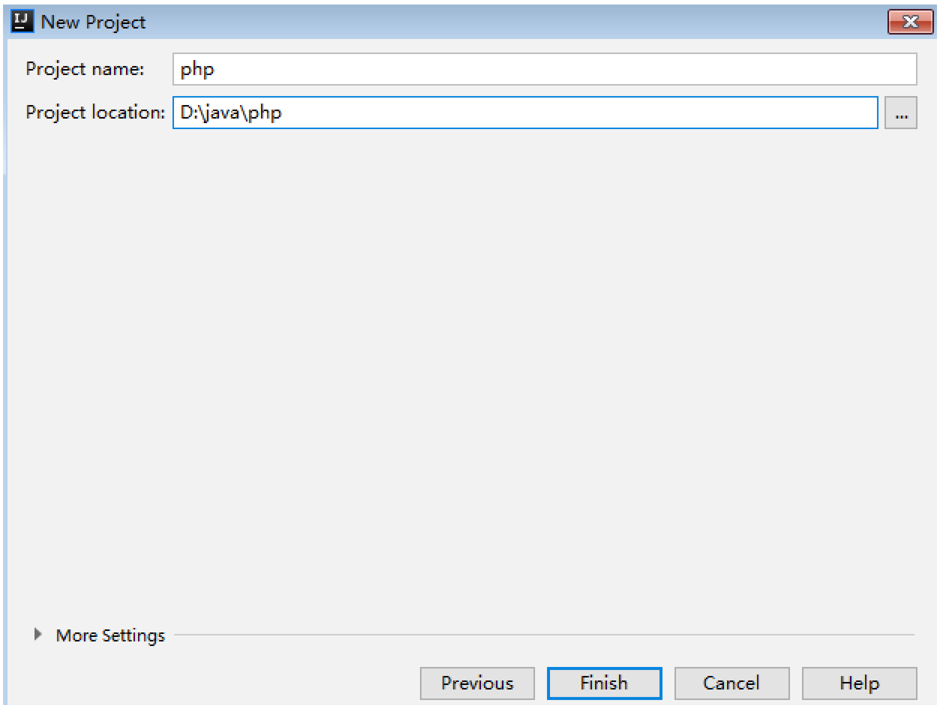
Figure 2-35 New Project



Create empty project for PHP

2. Click **...**, select the directory where the SDK is decompressed, and click **Finish**.

Figure 2-36 Selecting the SDK directory after decompression

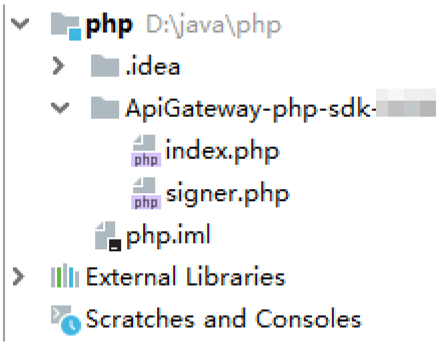


Project name: php

Project location: D:\java\php

3. View the directory structure of the project.

Figure 2-37 Directory structure of the new project



Modify the parameters in sample code **signer.php** as required. For details about the sample code, see **API Calling Example**.

API Calling Example

1. Import the PHP SDK to your code.

```
require 'signer.php';
```

2. Generate a signer and enter the key and secret of the authorized credential. For details about how to obtain the information, see **Obtaining API Calling Information**.

```
$signer = new Signer();
// Directly writing AK/SK in code is risky. For security, encrypt your AK/SK and store them in the configuration file or environment variables.
// In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, set environment variables HUAWEICLOUD_SDK_AK and HUAWEICLOUD_SDK_SK.
$signer->Key = getenv('HUAWEICLOUD_SDK_AK');
$signer->Secret = getenv('HUAWEICLOUD_SDK_SK');
```

3. Generate a request, and specify the method, request URL, and body. For details about how to obtain the information, see **Obtaining API Calling Information**.

```
$req = new Request('GET', "https://c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com/app1?a=1");
$req->body = '';
```

4. Add the **x-stage** header to the request to specify the environment. Add other headers to sign as required.

```
$req->headers = array(
   'x-stage' => 'RELEASE',
);
```

5. Execute the following function to generate a **$curl** context variable.

```
$curl = $signer->Sign($req);
```

6. If the subdomain name allocated by the system is used to access the API of HTTPS requests, ignore the certificate verification. Otherwise, skip this step.

```
curl_setopt($curl, CURLOPT_SSL_VERIFYHOST, 0);
curl_setopt($curl, CURLOPT_SSL_VERIFYPEER, 0);
```

7. Access the API and view the access result.

```
$response = curl_exec($curl);
echo curl_getinfo($curl, CURLINFO_HTTP_CODE);
echo $response;
curl_close($curl);
```

2.2.9 C++

Scenarios

To use C++ to call an API through App authentication, obtain the C++ SDK, and then call the API by referring to the API calling example.

Prerequisites

- You have obtained API calling information. For details, see **Preparations**.
- Install the OpenSSL library.
  ```
  apt-get install libssl-dev
  ```
- Install the curl library.
  ```
  apt-get install libcurl4-openssl-dev
  ```

Obtaining the SDK

Old version: Log in to the ROMA Connect console, choose **API Connect** > **API Calling** > **SDKs**, and download the SDK.

New version: Log in to the ROMA Connect console, choose **API Connect** > **Credentials** > **SDKs**, and download the SDK.

The following shows the directory structure after the decompression.

Name	Description
hasher.cpp	SDK code
hasher.h	
header.h	
RequestParams.cpp	
RequestParams.h	
signer.cpp	
signer.h	
Makefile	**Makefile** file
main.cpp	Sample code

API Calling Example

1. Add the following references to **main.cpp**:
   ```
   #include <stdio.h>
   #include <stdlib.h>
   #include <string.h>
   #include <curl/curl.h>
   #include "signer.h"
   ```
2. Generate a signer and enter the key and secret of the authorized credential. For details about how to obtain the information, see **Obtaining API Calling Information**.

```
// Directly writing AK/SK in code is risky. For security, encrypt your AK/SK and store them in the
configuration file or environment variables.
// In this example, the AK/SK are stored in environment variables for identity authentication. Before
running this example, set environment variables HUAWEICLOUD_SDK_AK and
HUAWEICLOUD_SDK_SK.
Signer signer(getenv("HUAWEICLOUD_SDK_AK"), getenv("HUAWEICLOUD_SDK_SK"));
```

3. Generate a new **RequestParams** request, and specify the method, domain name, request URI, query strings, and request body. For details about how to obtain the information, see **Obtaining API Calling Information**.

```
 RequestParams* request = new RequestParams("POST", "c967a237-cd6c-470e-906f-
a8655461897e.apigw.exampleRegion.com", "/app1",
        "Action=ListUsers&Version=2010-05-08", "demo");
```

4. Add the **x-stage** header to the request to specify the environment. Add other headers to sign as required.

```
request->addHeader("x-stage", "RELEASE");
```

5. Execute the following function to add the generated headers as request variables.

```
signer.createSignature(request);
```

6. Use the curl library to access the API and view the access result.

```
static size_t
WriteMemoryCallback(void *contents, size_t size, size_t nmemb, void *userp)
{
    size_t realsize = size * nmemb;
    struct MemoryStruct *mem = (struct MemoryStruct *)userp;

    mem->memory = (char*)realloc(mem->memory, mem->size + realsize + 1);
    if (mem->memory == NULL) {
        /* out of memory! */
        printf("not enough memory (realloc returned NULL)\n");
        return 0;
    }

    memcpy(&(mem->memory[mem->size]), contents, realsize);
    mem->size += realsize;
    mem->memory[mem->size] = 0;

    return realsize;
}

//send http request using curl library
int perform_request(RequestParams* request)
{
    CURL *curl;
    CURLcode res;
    struct MemoryStruct resp_header;
    resp_header.memory = (char*)malloc(1);
    resp_header.size = 0;
    struct MemoryStruct resp_body;
    resp_body.memory = (char*)malloc(1);
    resp_body.size = 0;

    curl_global_init(CURL_GLOBAL_ALL);
    curl = curl_easy_init();

    curl_easy_setopt(curl, CURLOPT_CUSTOMREQUEST, request->getMethod().c_str());
    std::string url = "http://" + request->getHost() + request->getUri() + "?" + request-
>getQueryParams();
    curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
    struct curl_slist *chunk = NULL;
    std::set<Header>::iterator it;
    for (auto header : *request->getHeaders()) {
        std::string headerEntry = header.getKey() + ": " + header.getValue();
        printf("%s\n", headerEntry.c_str());
        chunk = curl_slist_append(chunk, headerEntry.c_str());
    }
    printf("-------------\n");
    curl_easy_setopt(curl, CURLOPT_HTTPHEADER, chunk);
    curl_easy_setopt(curl, CURLOPT_COPYPOSTFIELDS, request->getPayload().c_str());
    curl_easy_setopt(curl, CURLOPT_NOBODY, 0L);
    curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteMemoryCallback);
    curl_easy_setopt(curl, CURLOPT_HEADERDATA, (void *)&resp_header);
    curl_easy_setopt(curl, CURLOPT_WRITEDATA, (void *)&resp_body);
    //curl_easy_setopt(curl, CURLOPT_VERBOSE, 1L);
    res = curl_easy_perform(curl);
    if (res != CURLE_OK) {
        fprintf(stderr, "curl_easy_perform() failed: %s\n", curl_easy_strerror(res));
    }
    else {
        long status;
        curl_easy_getinfo(curl, CURLINFO_HTTP_CODE, &status);
        printf("status %d\n", status);
        printf(resp_header.memory);
        printf(resp_body.memory);
    }
    free(resp_header.memory);
    free(resp_body.memory);
    curl_easy_cleanup(curl);

    curl_global_cleanup();

    return 0;
}
```

7. Run the **make** command to obtain a **main** file, execute the file, and then view the execution result.

2.2.10 C

Scenarios

To use C to call an API through App authentication, obtain the C SDK, and then call the API by referring to the API calling example.

Prerequisites

- You have obtained API calling information. For details, see **Preparations**.
- Install the OpenSSL library.
  ```
  apt-get install libssl-dev
  ```
- Install the curl library.
  ```
  apt-get install libcurl4-openssl-dev
  ```

Obtaining the SDK

Old version: Log in to the ROMA Connect console, choose **API Connect** > **API Calling** > **SDKs**, and download the SDK.

New version: Log in to the ROMA Connect console, choose **API Connect** > **Credentials** > **SDKs**, and download the SDK.

The following shows the directory structure after the decompression.

Name	Description
signer_common.c	SDK code
signer_common.h	

Name	Description
signer.c	
signer.h	
Makefile	**Makefile** file
main.c	Sample code

API Calling Example

1. Add the following references to **main.c**:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <curl/curl.h>
#include "signer.h"
```

2. Generate a variable of the **sig_params_t** type and enter the key and secret of the authorized credential. For details, see **Obtaining API Calling Information**.

```
sig_params_t params;
sig_params_init(&params);
// Directly writing AK/SK in code is risky. For security, encrypt your AK/SK and store them in the configuration file or environment variables.
// In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, set environment variables HUAWEICLOUD_SDK_AK and HUAWEICLOUD_SDK_SK.
sig_str_t app_key = sig_str(getenv("HUAWEICLOUD_SDK_AK"));
sig_str_t app_secret = sig_str(getenv("HUAWEICLOUD_SDK_SK"));
params.key = app_key;
params.secret = app_secret;
```

3. Specify the method, domain name, request URI, query strings, and request body. For details about how to obtain the information, see **Obtaining API Calling Information**

```
sig_str_t host = sig_str("c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com");
sig_str_t method = sig_str("GET");
sig_str_t uri = sig_str("/app1");
sig_str_t query_str = sig_str("a=1&b=2");
sig_str_t payload = sig_str("");
params.host = host;
params.method = method;
params.uri = uri;
params.query_str = query_str;
params.payload = payload;
```

4. Add the **x-stage** header to the request to specify the environment. Add other headers to sign as required.

```
sig_headers_add(&params.headers, "x-stage", "RELEASE");
```

5. Execute the following function to add the generated headers as request variables.

```
sig_sign(&params);
```

6. Use the curl library to access the API and view the access result.

```
static size_t
WriteMemoryCallback(void *contents, size_t size, size_t nmemb, void *userp)
{
    size_t realsize = size * nmemb;
    struct MemoryStruct *mem = (struct MemoryStruct *)userp;

    mem->memory = (char*)realloc(mem->memory, mem->size + realsize + 1);
    if (mem->memory == NULL) {
        /* out of memory! */
        printf("not enough memory (realloc returned NULL)\n");
        return 0;
    }

    memcpy(&(mem->memory[mem->size]), contents, realsize);
    mem->size += realsize;
    mem->memory[mem->size] = 0;

    return realsize;
}

//send http request using curl library
int perform_request(RequestParams* request)
{
    CURL *curl;
    CURLcode res;
    struct MemoryStruct resp_header;
    resp_header.memory = malloc(1);
    resp_header.size = 0;
    struct MemoryStruct resp_body;
    resp_body.memory = malloc(1);
    resp_body.size = 0;

    curl_global_init(CURL_GLOBAL_ALL);
    curl = curl_easy_init();

    curl_easy_setopt(curl, CURLOPT_CUSTOMREQUEST, params.method.data);
    char url[1024];
    sig_snprintf(url, 1024, "http://%V%V?%V", &params.host, &params.uri, &params.query_str);
    curl_easy_setopt(curl, CURLOPT_URL, url);
    struct curl_slist *chunk = NULL;
    for (int i = 0; i < params.headers.len; i++) {
        char header[1024];
        sig_snprintf(header, 1024, "%V: %V", &params.headers.data[i].name,
&params.headers.data[i].value);
        printf("%s\n", header);
        chunk = curl_slist_append(chunk, header);
    }
    printf("-------------\n");
    curl_easy_setopt(curl, CURLOPT_HTTPHEADER, chunk);
    curl_easy_setopt(curl, CURLOPT_POSTFIELDS, params.payload.data);
    curl_easy_setopt(curl, CURLOPT_NOBODY, 0L);
    curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteMemoryCallback);
    curl_easy_setopt(curl, CURLOPT_HEADERDATA, (void *)&resp_header);
    curl_easy_setopt(curl, CURLOPT_WRITEDATA, (void *)&resp_body);
    //curl_easy_setopt(curl, CURLOPT_VERBOSE, 1L);
    res = curl_easy_perform(curl);
    if (res != CURLE_OK) {
        fprintf(stderr, "curl_easy_perform() failed: %s\n", curl_easy_strerror(res));
    }
    else {
        long status;
        curl_easy_getinfo(curl, CURLINFO_HTTP_CODE, &status);
        printf("status %d\n", status);
        printf(resp_header.memory);
        printf(resp_body.memory);
    }
    free(resp_header.memory);
    free(resp_body.memory);
    curl_easy_cleanup(curl);

    curl_global_cleanup();

    //free signature params
    sig_params_free(&params);
    return 0;
}
```

7. Run the **make** command to obtain a **main** file, execute the file, and then view the execution result.

2.2.11 Android

Scenarios

To use Android to call an API through App authentication, obtain the Android SDK, create a project, and then call the API by referring to the API calling example.

Prerequisites

- You have obtained API calling information. For details, see **Preparations**.
- You have installed the Android development environment. For details, see **Preparations**.

Obtaining the SDK

Old version: Log in to the ROMA Connect console, choose **API Connect** > **API Calling** > **SDKs**, and download the SDK.

New version: Log in to the ROMA Connect console, choose **API Connect** > **Credentials** > **SDKs**, and download the SDK.

The following shows the directory structure after the decompression.

Name	Description
app\	Android project code
gradle\	Gradle files
build.gradle	Gradle configuration files
gradle.properties	
settings.gradle	
gradlew	Gradle Wrapper scripts
gradlew.bat	

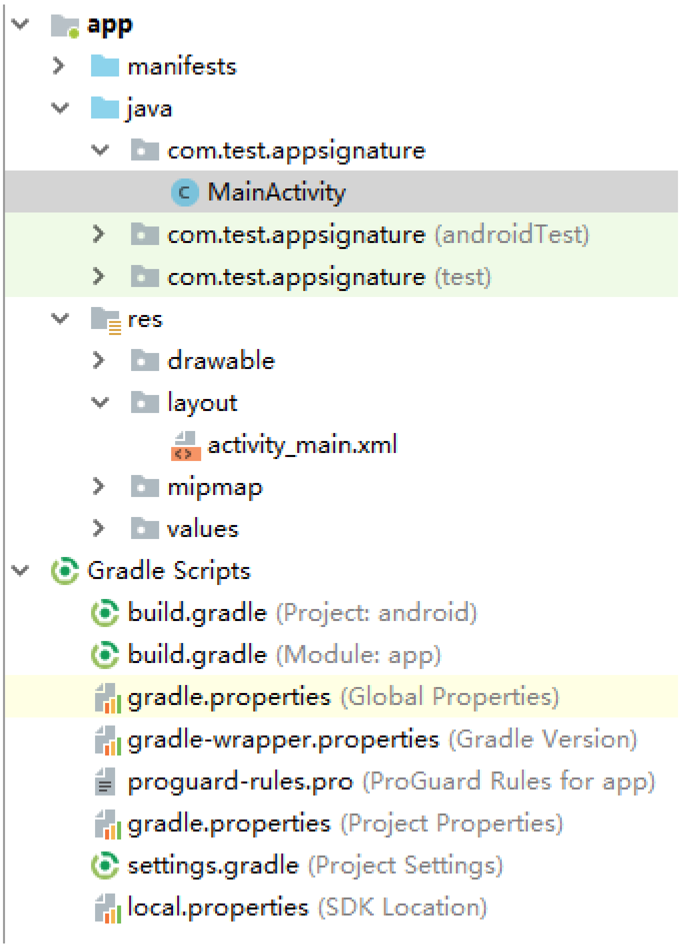
Opening the Sample Project

1. Start Android Studio and choose **File** > **Open**.

   Select the directory where the SDK is decompressed.

2. View the directory structure of the project shown in the following figure.

Figure 2-38 Project directory structure



```
app
  manifests
  java
    com.test.appsignature
      MainActivity
    com.test.appsignature (androidTest)
    com.test.appsignature (test)
  res
    drawable
    layout
      activity_main.xml
    mipmap
    values
Gradle Scripts
  build.gradle (Project: android)
  build.gradle (Module: app)
  gradle.properties (Global Properties)
  gradle-wrapper.properties (Gradle Version)
  proguard-rules.pro (ProGuard Rules for app)
  gradle.properties (Project Properties)
  settings.gradle (Project Settings)
  local.properties (SDK Location)
```

API Calling Example

1. Add required JAR files to the **app/libs** directory of the Android project. The following JAR files must be included:
   - java-sdk-core-x.x.x.jar
   - joda-time-2.10.jar
2. Add dependencies of the **okhttp** library to the **build.gradle** file.

   Add **implementation 'com.squareup.okhttp3:okhttp:3.14.2'** in the **dependencies** field of the **build.gradle** file.

   ```
   dependencies {
       ...
       ...
       implementation 'com.squareup.okhttp3:okhttp:3.14.3'
   }
   ```
3. Create a request, enter the key and secret of the authorized credential, and specify the method name, request URL, and body. For details, see **Obtaining API Calling Information**.

   ```
   Request request = new Request();
   try {
   // Directly writing AK/SK in code is risky. For security, encrypt your AK/SK and store them in the configuration file or environment variables.
   // In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, set environment variables HUAWEICLOUD_SDK_AK and HUAWEICLOUD_SDK_SK.
   ```

```
    request.setKey(System.getenv("HUAWEICLOUD_SDK_AK"));
    request.setSecrect(System.getenv("HUAWEICLOUD_SDK_SK"));
    request.setMethod("POST");
    request.setUrl("https://c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com/app1");
    request.addQueryStringParam("name", "value");
    request.addHeader("Content-Type", "text/plain");
    request.setBody("demo");
} catch (Exception e) {
    e.printStackTrace();
    return;
}
```

4. Sign the request to generate an **okhttp3.Request** object for API access.

```
okhttp3.Request signedRequest = Client.signOkhttp(request);
OkHttpClient client = new OkHttpClient.Builder().build();
Response response = client.newCall(signedRequest).execute();
```

2.2.12 curl

Scenarios

To use the curl command to call an API through App authentication, download the JavaScript SDK to generate the curl command, and copy the command to the CLI to call the API.

Prerequisites

- You have obtained API calling information. For details, see **Preparations**.
- The browser must be Chrome 89.0 or later.

Obtaining the SDK

Old version: Log in to the ROMA Connect console, choose **API Connect** > **API Calling** > **SDKs**, and download the SDK.

New version: Log in to the ROMA Connect console, choose **API Connect** > **Credentials** > **SDKs**, and download the SDK.

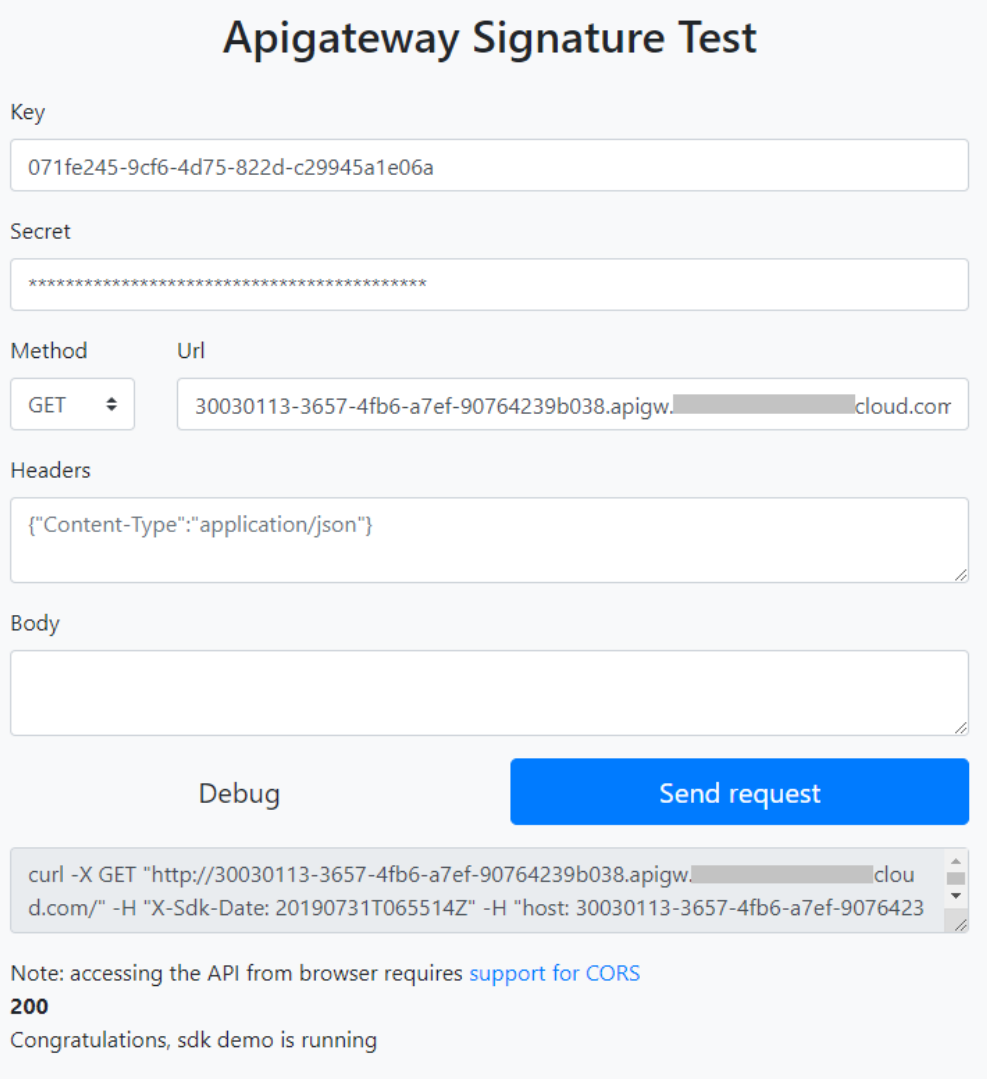
The following shows the directory structure after the decompression.

Name	Description
signer.js	SDK code
node_demo.js	Node.js sample code
demo.html	Browser sample code
demo_require.html	Browser sample code (loaded using **require**)
test.js	Test cases
js\hmac-sha256.js	Dependencies
licenses\license-crypto-js	Third-party licenses
licenses\license-node	

API Calling Example

1. Use the JavaScript SDK to generate the curl command.

   Decompress the SDK. Open **demo.html** in a browser. The following figure shows the demo page.



Apigateway Signature Test

Key: 071fe245-9cf6-4d75-822d-c29945a1e06a

Secret: **

Method: GET

Url: 30030113-3657-4fb6-a7ef-90764239b038.apigw.[redacted]cloud.com

Headers: {"Content-Type":"application/json"}

Body:

```
curl -X GET "http://30030113-3657-4fb6-a7ef-90764239b038.apigw.[redacted]cloud.com/" -H "X-Sdk-Date: 20190731T065514Z" -H "host: 30030113-3657-4fb6-a7ef-9076423
```

Note: accessing the API from browser requires support for CORS

200

Congratulations, sdk demo is running

2. Enter the key and secret of the authorized credential, and specify the method name and request URL. For details, see **Obtaining API Calling Information**. Example:

```
// Directly writing AK/SK in code is risky. For security, encrypt your AK/SK and store them in the configuration file or environment variables.
// In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, set environment variables HUAWEICLOUD_SDK_AK and HUAWEICLOUD_SDK_SK.
Key=4f5f626b-073f-402f-a1e0-e52171c6100c
Secret=******
Method=POST
Url=https://c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com/app1
```

3. Enter query and header parameters in JSON format, and set the request body.

4. Click **Send request** to generate a **curl** command. Copy the **curl** command to the CLI to access the API.

```
//If the subdomain name allocated by the system is used to access the API of HTTPS requests, add -k to the end of -d to ignore the certificate verification.
```

```
$ curl -X POST "https://c967a237-cd6c-470e-906f-a8655461897e.apigw.exampleRegion.com/app1" -H
"X-Sdk-Date: 20180530T115847Z" -H "Authorization: SDK-HMAC-SHA256
Access=071fe245-9cf6-4d75-822d-c29945a1e06a, SignedHeaders=host;x-sdk-date,
Signature=9e5314bd156d517******dd3e5765fdde4" -d ""
Congratulations, sdk demo is running
```

NOTE

The **curl** command generated using an SDK does not meet the format requirements of Windows. Please run the **curl** command in Git Bash.

2.3 Developing API Calling Authentication (IAM)

2.3.1 Token Authentication

Scenarios

To use token authentication, you need to obtain a token and add **X-Auth-Token** to the request header when making API calls.

NOTE

You can use either of the following authentication modes to call APIs.

- Token authentication: Requests are authenticated using a token.
- AK/SK authentication: Requests are encrypted using an AK/SK.

Calling an API Through Token Authentication

1. Obtain a token.

   For details, see **Obtaining a User Token** in the *Identity and Access Management (IAM) API Reference*.

   The token is the value of **X-Subject-Token** in the response.

   The following is an example request:

```
POST https://{iam_endpoint}/v3/auth/tokens
Content-Type: application/json

{
    "auth": {
        "identity": {
            "methods": [
                "password"
            ],
            "password": {
                "user": {
                    "name": "username",
                    "password": "********",
                    "domain": {
                        "name": "domainname"
                    }
                }
            }
        },
        "scope": {
            "project": {
                "id": "xxxxxxxx"
            }
        }
    }
}
```

In the preceding command:

- For details about *{iam_endpoint}*, see **Regions and Endpoints**.
- ***username*** indicates the username.
- ***domainname*** indicates the account name of the user.
- ************** indicates the login password of the user.
- ***xxxxxxxx*** indicates the project ID.

  On the management console, click the username in the upper right corner, choose **My Credentials** from the drop-down list, and then view the project ID.

2. To call a service API, add **X-Auth-Token** to the request header. The value of **X-Auth-Token** is that of the token obtained in **1**.

2.3.2 AK/SK Authentication

This section describes how to use AK and SK to sign requests.

NOTE

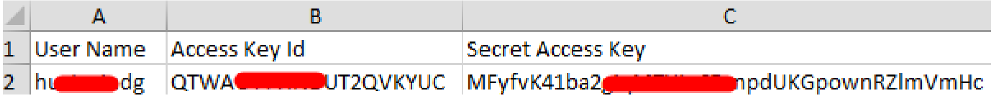
- AK indicates the access key ID. It is a unique identifier associated with a secret access key and is used in conjunction with a secret access key to sign requests cryptographically.
- SK indicates the secret access key used together with the access key ID to sign requests. AK and SK can be used together to identify a request sender to prevent the request from being modified.

Generating an AK and SK Pair

If an AK/SK pair has already been generated, skip this step. Find the downloaded AK/SK file, which is usually named **credentials.csv**.

As shown in the following figure, the file contains the username, access key ID, and secret access key.

Figure 2-39 Content of the credential.csv file

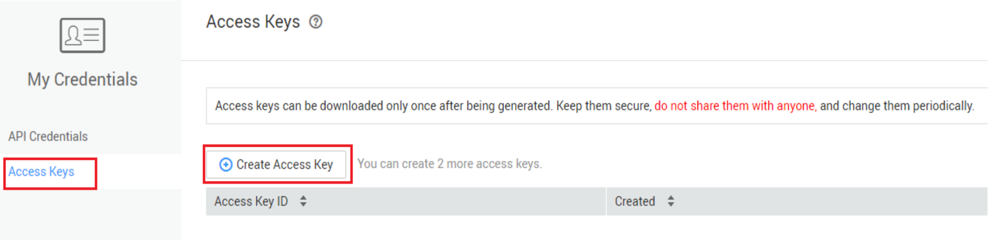


	A	B	C
1	User Name	Access Key Id	Secret Access Key
2	hu[redacted]dg	QTWA[redacted]UT2QVKYUC	MFyfvK41ba2[redacted]npdUKGpownRZlmVmHc

Perform the following procedure to generate an AK/SK pair:

1. Log in to the management console.
2. Click the username in the upper right corner and choose **My Credentials** from the drop-down list.
3. On the **My Credentials** page, choose **Access Keys**.
4. Click **Create Access Key**.
5. Enter the password and verification code, and click **OK** to download the access key. Keep the access key secure.

Figure 2-40 Obtaining an access key



Access keys can be downloaded only once after being generated. Keep them secure, do not share them with anyone, and change them periodically.

You can create 2 more access keys.

Generating a Signature

Generate a signature in the same way as in **Developing API Calling Authentication (App)**. Replace the AppKey with the AK and replace the AppSecret with the SK to complete the signing and request processing.

2.4 Developing Custom Function Backends

2.4.1 Overview

Overview

The function backend encapsulates multiple services into a service by compiling function scripts. Before developing a function backend, familiarize yourself with **Custom Backend Development Requirements**.

Function backends can be compiled using JavaScript only. The JavaScript engine runs on the Java Virtual Machine (JVM) and can call the **Java class** provided by ROMA Connect.

Script development must be based on the built-in recommended encryption algorithms.

Common Configuration Reference

You can add global common configurations, such as variables, passwords, and certificates, to the custom backend and reference them in function scripts. For details, see **Adding Public Configurations for a Custom Backend**.

If the configuration name is **example**, the reference format is as follows:

- Template variable: #{example}
- Password: CipherUtils.getPlainCipherText("example")
- Certificate: CipherUtils.getPlainCertificate("example")

2.4.2 AesUtils

Path

com.roma.apic.livedata.common.v1.AesUtils

Description

This class is used to provide the AES encryption and decryption methods.

Example

Encryption:

```
importClass(com.roma.apic.livedata.common.v1.AesUtils);
function execute(data) {
  var plainText = "plainText";
//Hard-coded encryption and decryption keys pose higher security risks. You are advised to store the key in
the configuration file. This example uses the secretKey configured for the custom backend.
  var secretKey = "#{secretKey}";
  var initialVector = "initialVector";
  var encryptText = AesUtils.encrypt(plainText, secretKey, initialVector, AesUtils.KEYLENGTH.KL_128,
AesUtils.MODE.GCM, AesUtils.PAD.NOPADDING);
  return encryptText
}
```

Decryption:

```
importClass(com.roma.apic.livedata.common.v1.AesUtils);
function execute(data) {
  var encryptText = "encryptText";
//Hard-coded encryption and decryption keys pose higher security risks. You are advised to store the key in
the configuration file. This example uses the secretKey configured for the custom backend.
  var secretKey = "#{secretKey}";
  var initialVector = "initialVector";
  var decryptText = AesUtils.decrypt(encryptText, secretKey, initialVector, AesUtils.KEYLENGTH.KL_128,
AesUtils.MODE.GCM, AesUtils.PAD.NOPADDING);
  return decryptText
}
```

Method List

Returned Type	Method and Description
static java.lang.String	decrypt(java.lang.String encryptText, java.lang.String secretKey) Decrypt the ciphertext using the AES algorithm and a private key.
static java.lang.String	encrypt(java.lang.String plainText, java.lang.String secretKey) Encrypt the plaintext using the AES algorithm and a private key.
static java.lang.String	decrypt(java.lang.String encryptText, java.lang.String secretKey, java.lang.String initialVector, AesUtils.KEYLENGTH length, AesUtils.MODE mode, AesUtils.PAD pad) Decrypt the ciphertext using the AES algorithm and a private key with specified mode and length.
static java.lang.String	encrypt(java.lang.String plainText, java.lang.String secretKey, java.lang.String initialVector, AesUtils.KEYLENGTH length, AesUtils.MODE mode, AesUtils.PAD pad) Encrypt the plaintext using the AES algorithm and a private key with specified mode and length.

Method Details

- **public static java.lang.String decrypt(java.lang.String encryptText, java.lang.String secretKey)**

  Decrypting ciphertext using the AES algorithm and a private key

  Input Parameters

  – *encryptText*: ciphertext to be decrypted (a maximum of 8192 characters).

  – *secretKey*: secret key.

  Returns

  Decrypted data

- **public static java.lang.String encrypt(java.lang.String plainText, java.lang.String secretKey)**

  Encrypting plaintext using the AES algorithm and a private key

  Input Parameters

  – *plainText*: plaintext to be encrypted (a maximum of 4096 characters).

  – *secretKey*: secret key.

  Returns

  Encrypted data

- **public static java.lang.String decrypt(java.lang.String encryptText, java.lang.String secretKey, java.lang.String initialVector, AesUtils.KEYLENGTH length, AesUtils.MODE mode, AesUtils.PAD pad)**

  Decrypting ciphertext using the AES algorithm and a private key with mode and length specified.

  Input Parameters

  – *encryptText*: ciphertext to be decrypted (a maximum of 8192 characters).

  – *secretKey*: secret key.

  – *initialVector*: initial vector.

  – *length*: length of the key. The options are **AesUtils.KEYLENGTH.KL_0**, **AesUtils.KEYLENGTH.KL_128**, **AesUtils.KEYLENGTH.KL_192**, and **AesUtils.KEYLENGTH.KL_256**.

  – *mode*: working mode. The options are **AesUtils.MODE.GCM** and **AesUtils.MODE.CTR**.

  – *pad*: padding mode. The options are **AesUtils.PAD.PKCS5PADDING** and **AesUtils.PAD.NOPADDING**.

  Returns

  Decrypted data

- **public static java.lang.String encrypt(java.lang.String plainText, java.lang.String secretKey, java.lang.String initialVector, AesUtils.KEYLENGTH length, AesUtils.MODE mode, AesUtils.PAD pad)**

  Encrypt the plaintext using the AES algorithm and a private key with specified mode and length.

  Input Parameters

  – *plainText*: plaintext to be encrypted (a maximum of 4096 characters).

  – *secretKey*: secret key.

- *initialVector*: initial vector.
- *length*: length of the key. The options are **AesUtils.KEYLENGTH.KL_0**, **AesUtils.KEYLENGTH.KL_128**, **AesUtils.KEYLENGTH.KL_192**, and **AesUtils.KEYLENGTH.KL_256**.
- *mode*: working mode. The options are **AesUtils.MODE.GCM** and **AesUtils.MODE.CTR**.
- *pad*: padding mode. The options are **AesUtils.PAD.PKCS5PADDING** and **AesUtils.PAD.NOPADDING**.

Returns

Encrypted data

2.4.3 APIConnectResponse

Path

com.roma.apic.livedata.provider.v1.APIConnectResponse

Description

This class is used to specify the HTTP status code, header, and body to be returned after a function API is called. To achieve this, the class object must be returned in the execute function.

Example

```
importClass(com.roma.apic.livedata.provider.v1.APIConnectResponse);
function execute(data) {
    return new APIConnectResponse(401, {"X-Type":"Demo"}, "unauthorized", false);
}
```

In this example, the HTTP status code returned when the function API is called is 401, the response header contains "X-Type: Demo", and the response body is "unauthorized".

Constructor Details

- **public APIConnectResponse(Integer statusCode)**

  Constructs an APIConnectResponse.

  Parameter: **statusCode** indicates the response status code.

- **public APIConnectResponse(Integer statusCode, Map<String,String> headers)**

  Constructs an APIConnectResponse.

  Parameters: **statusCode** indicates the response status code, and headers indicates the response header.

- **public APIConnectResponse(Integer statusCode, Map<String,String> headers, Object body)**

  Constructs an APIConnectResponse.

  Parameters: **statusCode** indicates the response status code, **headers** indicates the response header, and **body** indicates the response body.

- **public APIConnectResponse(Integer statusCode, Map<String,String> headers, String body, Boolean base64Encoded)**

Constructs an APIConnectResponse.

Parameters: **statusCode** indicates the response status code, **headers** indicates the response header, **body** indicates the response body, and **base64Encoded** indicates whether the body is encoded using Base64.

Method List

Returned Type	Method and Description
Object	**getBody**() Obtain the response body.
Map<String,String >	**getHeaders**() Obtain the response header.
Integer	**getStatusCode**() Obtain the response status code.
Boolean	**isBase64Encoded**() Check whether the body is encoded using Base64.
void	**setBase64Encoded**(Boolean base64Encoded) Set whether the body is encoded using Base64.
void	**setBody**(Object body) Set the response body.
void	**setHeaders**(Map<String,String> headers) Set the response header.
void	**setStatusCode**(Integer statusCode) Set the response status code.

Method Details

- **public Object getBody()**

  Obtain the response body.

  Returns

  Response body.

- **public Map<String,String> getHeaders()**

  Obtain the response header.

  Returns

  Map set of the request header.

- **public Integer getStatusCode()**

  Obtain the response status code.

  Returns

  Response status code.

- **public Boolean isBase64Encoded()**

  Check whether the body is encoded using Base64.

  Returns

  – **true**: Base64 encoding has been performed.

  – **false**: Base64 encoding is not performed.

- **public void setBase64Encoded(Boolean base64Encoded)**

  Set whether the body is encoded using Base64.

  Input Parameter

  base64Encoded: If the value is **true**, Base64 encoding has been performed. If the value is **false**, Base64 encoding is not performed.

- **public void setBody(Object body)**

  Set the response body.

  Input Parameter

  body indicates the body object.

- **public void setHeaders(Map<String,String> headers)**

  Set the response header.

  Input Parameter

  headers indicates the map set of headers.

- **public void setStatusCode(Integer statusCode)**

  Set the response status code.

  Input Parameter

  statusCode indicates the status code.

2.4.4 Base64Utils

Path

com.roma.apic.livedata.common.v1.Base64Utils

Description

This class is used to provide the Base64Utils encoding and decoding functions.

Example

Base64 encoding:

```
importClass(com.roma.apic.livedata.common.v1.Base64Utils);
function execute(data) {
    var sourceCode = "Hello world!";
    return Base64Utils.encode(sourceCode);
}
```

multipart/form-data file upload:

```
importClass(com.roma.apic.livedata.common.v1.Base64Utils);
function execute(data) {
  var image = data.body.get("image")
  return {
    size: image.getSize(),
    name: image.getFileItem().getName(),
    base64: Base64Utils.encode(image.getFileItem().get())
  }
}
```

Method List

Returned Type	Method and Description
static java.lang.String	**decode**(java.lang.String content) Perform Base64 decoding on a character string.
static java.lang.String	**decodeUrlSafe**(java.lang.String content) Perform Base64 decoding on a character string (using the character set compatible with the URL).
static java.lang.String	**encode**(byte[] content) Perform Base64 encoding on a byte array.
static java.lang.String	**encode**(java.lang.String content) Perform Base64 encoding on a character string.
static java.lang.String	**encodeUrlSafe**(byte[] content) Perform Base64 encoding on a byte array (using the character set compatible with the URL).
static java.lang.String	**encodeUrlSafe**(java.lang.String content) Perform Base64 encoding on a character string (using the character set compatible with the URL).

Method Details

- **public static java.lang.String decode(java.lang.String content)**

  Perform Base64 decoding on a character string.

  Input Parameter

  content indicates a character string encrypted by using Base64.

  Returns

  Decrypted character string.

- **public static java.lang.String decodeUrlSafe(java.lang.String content)**

  Perform Base64 decoding on a byte array (using the character set compatible with the URL).

  Input Parameter

  content indicates a character string encrypted by using Base64.

  Returns

  Decrypted character string.

- **public static java.lang.String encode(byte[] content)**

  Perform Base64 encoding on a byte array.

  Input Parameter

  content indicates a byte array to be encrypted.

  Returns

  Encrypted character string.

- **public static java.lang.String encode(java.lang.String content)**

  Perform Base64 encoding on a character string.

  Input Parameter

  content indicates a character string to be encrypted.

  Returns

  Encrypted character string.

- **public static java.lang.String encodeUrlSafe(byte[] content)**

  Perform Base64 encoding on a byte array (using the character set compatible with the URL).

  Input Parameter

  content indicates a byte array to be encrypted.

  Returns

  Encrypted character string.

- **public static java.lang.String encodeUrlSafe(java.lang.String content)**

  Perform Base64 encoding on a character string (using the character set compatible with the URL).

  Input Parameter

  content indicates a character string to be encrypted.

  Returns

  Encrypted character string.

2.4.5 CacheUtils

Path

com.huawei.livedata.lambdaservice.util.CacheUtils

Description

This class is used to save and obtain cache information.

Example

Before using CacheUtils, create an object first.

The get method of CacheUtils allows only the following items in the whitelist to be used as a key:

```
"DICT:api_gw_rest_addr", "DICT:api_gw_rest_float_addr", "DICT:api_gw_rest_eip_addr",
"DICT:livedata_private_address"
```

Example:

```
importClass(com.huawei.livedata.lambdaservice.util.CacheUtils);
function execute(data) {
  var cacheUtils = new CacheUtils
  var value = cacheUtils.get("DICT:livedata_private_address")
  return value
}
```

The returned result is the private IP address of LiveData.

Example of putCache and getCache methods:

```
importClass(com.huawei.livedata.lambdaservice.util.CacheUtils);
function execute(data) {
  var cacheUtils = new CacheUtils
  code = cacheUtils.putCache("age", "20")
  if (code != true) {
    return code
  }
  var name = cacheUtils.getCache("age")
  return name
}
```

The returned result is **20**.

Method List

Returned Type	Method and Description
static boolean	**putCache**(String key, String value) Save cache information.
static boolean	**putCache**(String key, String value, int time) Save the cache information with the timeout interval.
static String	**getCache**(String key) Obtain cache information.
static long	**removeCache**(String key) Remove cache information.
static String	**get**(String key) Obtain dictionary cache information.

Method Details

- **public static boolean putCache(String key, String value)**

  Save cache information.

  Input Parameter

  – **key** indicates the key value of cache information.

  – **value** indicates the cache information.

  Returns

  Corresponding boolean value.

- **public static boolean putCache(String key, String value, int time)**

  Save the cache information with the timeout interval.

  Input Parameter

  - **key** indicates the key value of cache information.
  - **value** indicates the cache information.
  - **time** indicates the timeout duration, in seconds. Cache information will be deleted after the timeout. Querying this information will return a null value.

  Returns

  Corresponding boolean value.

- **public static String getCache(String key)**

  Obtain cache information.

  Input Parameter

  key indicates the key value of cache information.

  Returns

  Cache information corresponding to the key value.

- **public static long removeCache(String key)**

  Remove cache information.

  Input Parameter

  key indicates the key value of cache information to be removed.

  Returns

  Execution result.

- **public static String get(String key)**

  Obtain dictionary cache information.

  Input Parameter

  key indicates the key value of dictionary cache information.

  Returns

  Dictionary cache information corresponding to the key value.

2.4.6 CipherUtils

Path

com.huawei.livedata.lambdaservice.security.CipherUtils

Description

This class is used to decrypt the key value of the password in the password box.

NOTE

When obtaining the key value of a common password in the decryption password box, protect sensitive information from being disclosed.

Method List

Returned Type	Method and Description
static String	**getPlainCipherText**(String key) Decrypt the key value of a common password in the password box.
static Response	**getPlainCertificate**(String key) Decrypt the key value of a certificate password in the password box.

Method Details

- **public static String getPlainCipherText(String key)**

  Decrypt the key value of a common password in the password box.

  Input Parameter

  key indicates the key value of a common password.

  Returns

  Decrypted password.

- **public static Response getPlainCertificate(String key)**

  Decrypt the key value of a certificate password in the password box.

  Input Parameter

  key indicates the key value of a certificate password.

  Returns

  Message body of the decrypted certificate password. The message body format is as follows:

```
{
"cipherType": "CERTIFICATE",
"passphrase": "xxx",
"privateKey": "xx",
"privateKey": "xx",
}
```

2.4.7 ConnectionConfig

Path

com.roma.apic.livedata.config.v1.ConnectionConfig

Description

This class is used with **RabbitMqConfig** and **RabbitMqProducer** to configure the connection to a RabbitMQ client.

Constructor Details

public ConnectionConfig(String host, int port, String userName, String pw)

Constructs a RabbitMQ client connection configuration.

2.4.8 DataSourceClient

Path

com.roma.apic.livedata.client.v1.DataSourceClient

Description

This class is used to connect to data sources and run SQL statements, stored procedures, or NoSQL query statements.

Example

SQL data source example:

```
importClass(com.roma.apic.livedata.client.v1.DataSourceClient);
importClass(com.roma.apic.livedata.config.v1.DataSourceConfig);
function execute(data){
    var config = new DataSourceConfig()
    config.setType("mysql")
    config.setUrl("jdbc:mysql://127.0.0.1:3306/db?allowPublicKeyRetrieval=true")
    config.setUser("username")
    config.setPassword("password")
    var ds = new DataSourceClient(config)
    return ds.execute("SELECT * FROM person where name = ? and age = ?", "Tom", 20);
}
```

```
importClass(com.roma.apic.livedata.client.v1.DataSourceClient);
importClass(com.roma.apic.livedata.config.v1.DataSourceConfig);
function execute(data){
  var config = new DataSourceConfig()
  config.setType("oracle")
  config.setUrl("jdbc:oracle:thin:@127.0.0.1:1521/db")
  config.setUser("username")
  config.setPassword("password")
  var ds = new DataSourceClient(config)
  return ds.execute("select table_name from user_tables");
}
```

NoSQL data source example:

```
importClass(com.roma.apic.livedata.client.v1.DataSourceClient);
importClass(com.roma.apic.livedata.config.v1.DataSourceConfig);
function execute(data){
    var config = new DataSourceConfig()
    config.setType("redis")
    config.setUrl("127.0.0.1:6379")
    config.setPassword("password")
    var ds = new DataSourceClient(config)
    return ds.execute("GET key");
}
```

Constructor Details

public DataSourceClient(DataSourceConfig config)

Import the data source configuration and construct a data source connector.

Method List

Returned Type	Method and Description
Object	execute(String sql, Object... prepareValue) Run SQL statements, stored procedures, or NoSQL query statements.

Method Details

public Object execute(String sql, Object... prepareValue)

Run SQL statements, stored procedures, or NoSQL query statements.

Input Parameter

prepareValue: This parameter is valid only in SQL statements and is used to replace "?" in SQL statements to prevent SQL injection.

Returns

Statement execution results

2.4.9 DataSourceConfig

Path

com.roma.apic.livedata.config.v1.DataSourceConfig

Description

This class is used with **DataSourceClient** to configure data sources.

Constructor Details

public DataSourceConfig()

Constructs a DataSourceConfig without parameters.

public DataSourceConfig(String type, String url, String user, String password)

Enter the data source type, connection string, username, and password to construct a DataSourceConfig.

Method List

Returned Type	Method and Description
String	**getType**() Obtain the data source type.
String	**getUrl**() Obtain the connection string.

Returned Type	Method and Description
String	getUser() Obtain the username.
String	getPassword() Obtain the password.
void	setType() Set the data source type. The value can be **mysql**, **mssql**, **oracle**, **postgresql**, **hive**, **redis**, or **mongodb**.
void	setUrl() Set the data source connection string.
void	setUser() Set the data source username.
void	setPassword() Set the data source password.

Method Details

- **public String getType()**

  Obtain the data source type.

  Returns

  Data source type.

- **public String getUrl()**

  Obtain the connection string.

  Returns

  Connection string.

- **public String getUser()**

  Obtain the username.

  Returns

  Username.

- **public String getPassword()**

  Obtain the password.

  Returns

  Password.

- **public void setType(String type)**

  Set the data source type. The value can be **mysql**, **mssql**, **oracle**, **postgresql**, **hive**, **redis**, or **mongodb**.

  Input Parameter

  – **type**: specifies the data source type.

- **public void setUrl(String url)**

  Set the data source connection string.

  If **type** is **mysql**, **mssql**, **oracle**, **postgresql**, or **hive**, set this parameter to the jdbc connection string. For example, "jdbc:mysql://127.0.0.1:8888/db?useUnicode=true&characterEncoding=utf8".

  If **type** is **redis**, the format is "127.0.0.1:6379@0", in which @0 indicates the Redis database ID and can be omitted.

  If **type** is **mongodb**, the format is "127.0.0.1:27017@db", in which db indicates the database name.

  Input Parameter

  – **url** indicates the connection string.

- **public void setUser(String user)**

  Set the data source username. If **type** is **redis**, you do not need to set this parameter.

  Input Parameter

  – **user** indicates the username.

- **public void setPassword(String password)**

  Set the data source password.

  Input Parameter

  – **password** indicates the password.

2.4.10 ExchangeConfig

Path

com.roma.apic.livedata.config.v1.ExchangeConfig

Description

This class is used with **RabbitMqConfig** and **RabbitMqProducer** to configure an exchange.

Constructor Details

public ExchangeConfig(String exchange, String type, boolean durable, boolean autoDelete, boolean internal, Map<String, Object> arguments)

Constructs an exchange configuration.

Parameters:

- **exchange** indicates the exchange name.
- **type** indicates the exchange type.
- **durable** indicates whether persistency is supported. The value **true** indicates persistency is supported, and the value **false** indicates that persistency is not supported.
- **autoDelete** indicates whether automatic deletion is supported. The value **true** indicates that automatic deletion is supported. The prerequisite for automatic deletion is that at least one queue or exchange is bound to the

exchange to be deleted. After automatic deletion, all queues or exchanges bound to the deleted exchange are unbound.

- **internal** indicates whether the exchange is a built-in exchange. The value **true** indicates that the exchange is a built-in exchange. The client cannot directly send messages to the exchange, but sending messages to another exchange first, which will forward the messages to the destination exchange.
- **arguments** indicates other attributes.

2.4.11 HttpClient

Path

- com.roma.apic.livedata.client.v1.HttpClient
- com.huawei.livedata.lambdaservice.livedataprovider.HttpClient

Description

This class is used to send HTTP requests.

NOTE

Some common HTTP response headers, such as **Location**, cannot be returned.

Example

- com.roma.apic.livedata.client.v1.HttpClient

```
importClass(com.roma.apic.livedata.client.v1.HttpClient);
importClass(com.roma.apic.livedata.provider.v1.APIConnectResponse);
function execute(data) {
    var httpClient = new HttpClient();
    var resp = httpClient.request('GET', 'http://apigdemo.exampleRegion.com/api/echo', {}, null, 
'application/json');
    myHeaders = resp.headers();
    proxyHeaders = {};
    for (var key in myHeaders) {
        proxyHeaders[key] = myHeaders.get(key);
    }
    return new APIConnectResponse(resp.code(), proxyHeaders, resp.body().string(), false);
}
```

- com.huawei.livedata.lambdaservice.livedataprovider.HttpClient

```
importClass(com.huawei.livedata.lambdaservice.livedataprovider.HttpClient);
function excute(data) {
    var httpExecutor = new HttpClient();
    var obj = JSON.parse(data);
    var host = 'xx.xx.xxx.xx:xxxx';
    var headers = {
        'clientapp' : 'FunctionStage'
    };
    var params = {
        'employ_no' :'00xxxxxx'
    };
    var result = httpExecutor.callGETAPI(host,'/livews/rest/apiservice/iData/personInfo/
batch',JSON.stringify(params),JSON.stringify(headers));
    return result;
}
```

Constructor Details

- com.roma.apic.livedata.client.v1.HttpClient

public HttpClient()

Constructs an HttpClient without parameters.

public HttpClient(HttpConfig config)

Constructs an HttpClient that contains the **HttpConfig** configuration information.

Parameter: **config** indicates the HttpClient configuration information.

- com.huawei.livedata.lambdaservice.livedataprovider.HttpClient

  public HttpClient()

  Constructs an HttpClient without parameters.

Method List

- com.roma.apic.livedata.client.v1.HttpClient

Returned Type	Method and Description
okhttp3.Response	**request**(**HttpConfig** config) Send REST requests.
okhttp3.Response	**request**(String method, String url) Send a REST request by specifying the request method and path.
okhttp3.Response	**request**(String method, String url, Map<String,String> headers) Send a REST request by specifying the request method, path, and header.
okhttp3.Response	**request**(String method, String url, Map<String,String> headers, String body) Send a REST request by specifying the request method, path, header, and body.
okhttp3.Response	**request**(String method, String url, Map<String,String> headers, String body, String contentType) Send a REST request by specifying the request method, path, header, body, and content type.

- com.huawei.livedata.lambdaservice.livedataprovider.HttpClient

Returned Type	Method and Description
String	**callGETAPI**(String url) Use the get method to invoke the HTTP or HTTPS service.
String	**callGETAPI**(String host, String service, String params, String header) Use the get method to invoke the HTTP or HTTPS service.

Returned Type	Method and Description
Response	**get**(String url, String header) Use the get method to invoke the HTTP or HTTPS service.
String	**callPostAPI**(String host, String service, String content, String header, String contentType) Use the post method to invoke the HTTP or HTTPS service.
String	**callPostAPI**(String url, String header, String requestBody, String type) Use the post method to invoke the HTTP or HTTPS service.
Response	**post**(String url, String header, String content, String type) Use the post method to invoke the HTTP or HTTPS service.
String	**callFormPost**(String url, String header, String/Map param) Invoke the HTTP or HTTPS service in the formdata format.
Response	**callFormPost**(String url, String header, String param, FormDataMultiPart form) Invoke the HTTP or HTTPS service in the formdata format.
String	**callDelAPI**(String url, String header, String content, String type) Use the delete method to invoke the HTTP or HTTPS service.
String	**callPUTAPI**(String url, String header, String content, String type) Use the put method to invoke the HTTP or HTTPS service.
String	**callPatchAPI**(String url, String header, String content, String type) Use the patch method to invoke the HTTP or HTTPS service.
Response	**put**(String url, String header, String content, String type) Use the put method to invoke the HTTP or HTTPS service.

Method Details

- com.roma.apic.livedata.client.v1.HttpClient
  - **public okhttp3.Response request(HttpConfig config)**

    Send REST requests.

    Input Parameter

    config indicates the **HttpConfig** configuration information.

    Returns

    Response body.
  - **public okhttp3.Response request(String method, String url)**

Send a REST request by specifying the request method and path.

Input Parameter

- **method** indicates a request method.
- **url** indicates a request URL.

Returns

Response body.

- **public okhttp3.Response request(String method, String url, Map<String,String> headers)**

  Send a REST request by specifying the request method, path, and header.

  Input Parameter

  - **method** indicates a request method.
  - **url** indicates a request URL.
  - **headers** indicates the request header information of the map type.

  Returns

  Response body.

- **public okhttp3.Response request(String method, String url, Map<String,String> headers, String body)**

  Send a REST request by specifying the request method, path, header, and body.

  Input Parameter

  - **method** indicates a request method.
  - **url** indicates a request URL.
  - **headers** indicates the request header information of the map type.
  - **body** indicates the request body.

  Returns

  Response body.

- **public okhttp3.Response request(String method, String url, Map<String,String> headers, String body, String contentType)**

  Send a REST request by specifying the request method, path, header, body, and content type.

  Input Parameter

  - **method** indicates a request method.
  - **url** indicates a request URL.
  - **headers** indicates the request header information of the map type.
  - **body** indicates the request body.
  - **contentType** indicates the content type of the request body.

Returns

Response body.

- com.huawei.livedata.lambdaservice.livedataprovider.HttpClient
  - **public String callGETAPI(String url)**

    Use the get method to invoke the HTTP or HTTPS service.

    Input Parameter

    url indicates the service address.

    Returns

    Response body.
  - **public String callGETAPI(String host, String service, String params, String header)**

    Use the get method to invoke the HTTP or HTTPS service.

    Input Parameter

    - **host** indicates the service address.
    - **service** indicates the service path.
    - **params** indicates the HTTP parameter information.
    - **header** indicates the HTTP header information.

    Returns

    Response body.
  - **public Response get(String url, String header)**

    Use the get method to invoke the HTTP or HTTPS service.

    Input Parameter

    - **url** indicates the service address.
    - **header** indicates the request header information.

    Returns

    Response body.
  - **public String callPostAPI(String host, String service, String content, String header, String contentType)**

    Use the post method to invoke the HTTP or HTTPS service.

    Input Parameter

    - **host** indicates the service address.
    - **service** indicates the service path.
    - **content** indicates the message body.
    - **header** indicates the request header information.
    - **contentType** indicates the content type.

    Returns

    Response body.

- **public String callPostAPI(String url, String header, String requestBody, String type)**

  Use the post method to invoke the HTTP or HTTPS service.

  Input Parameter

  - **url** indicates the service address.
  - **header** indicates the request header information.
  - **requestBody** indicates the message body.
  - **type** indicates the MIME type.

  Returns

  Response body.

- **public Response post(String url, String header, String content, String type)**

  Use the post method to invoke the HTTP or HTTPS service.

  Input Parameter

  - **url** indicates the service address.
  - **header** indicates the request header information.
  - **content** indicates the message body.
  - **type** indicates the MIME type.

  Returns

  Response body.

- **public String callFormPost(String url, String header, String/Map param)**

  Invoke the HTTP or HTTPS service in the formdata format.

  Input Parameter

  - **url** indicates the service address.
  - **header** indicates the request header information.
  - **param** indicates the parameter information.

  Returns

  Response body.

- **public Response callFormPost(String url, String header, String param, FormDataMultiPart form)**

  Invoke the HTTP or HTTPS service in the formdata format.

  Input Parameter

  - **url** indicates the service address.
  - **header** indicates the request header information.
  - **param** indicates the parameter information.

  - **form** indicates the body parameter.

  Returns

  Response body.

- **public String callDelAPI(String url, String header, String content, String type)**

  Use the delete method to invoke the HTTP or HTTPS service.

  Input Parameter

  - **url** indicates the service address.
  - **header** indicates the request header information.
  - **content** indicates the message body.
  - **type** indicates the MIME type.

  Returns

  Response body.

- **public String callPUTAPI(String url, String header, String content, String type)**

  Use the put method to invoke the HTTP or HTTPS service.

  Input Parameter

  - **url** indicates the service address.
  - **header** indicates the request header information.
  - **content** indicates the message body.
  - **type** indicates the MIME type.

  Returns

  Response body.

- **public String callPatchAPI(String url, String header, String content, String type)**

  Use the patch method to invoke the HTTP or HTTPS service.

  Input Parameter

  - **url** indicates the service address.
  - **header** indicates the request header information.
  - **content** indicates the message body.
  - **type** indicates the MIME type.

  Returns

  Response body.

- **public Response put(String url, String header, String content, String type)**

  Use the put method to invoke the HTTP or HTTPS service.

  Input Parameter

- **url** indicates the service address.
- **header** indicates the request header information.
- **content** indicates the message body.
- **type** indicates the MIME type.

Returns

Response body.

2.4.12 HttpConfig

Path

com.roma.apic.livedata.config.v1.HttpConfig

Description

This class is used together with **HttpClient** to configure HTTP requests.

Example

```
importClass(com.roma.apic.livedata.client.v1.HttpClient);
importClass(com.roma.apic.livedata.config.v1.HttpConfig);
function execute(data) {
    var requestConfig = new HttpConfig();

    requestConfig.setAccessKey("071fe245-9cf6-4d75-822d-c29945a1e06a");
    requestConfig.setSecretKey("c6e52419-2270-****-****-ae7fdd01dcd5");

    requestConfig.setMethod('POST');
    requestConfig.setUrl("https://30030113-3657-4fb6-a7ef-90764239b038.apigw.exampleRegion.com/
app1");
    requestConfig.setContent("body");
    requestConfig.setContentType('application/json');

    var client = new HttpClient();
    var resp =  client.request(requestConfig);
    return resp.body().string()
}
```

Constructor Details

public HttpConfig()

Constructs an HttpConfig without parameters.

Method List

Returned Type	Method and Description
void	**addHeaderToSign**(String headerName) Add a request header to be signed.

Returned Type	Method and Description
String	**getAccessKey**() Obtain the AccessKey. Requests for which AccessKey and SecretKey are set use the **AK/SK signature algorithm** for signing.
String	**getCaCertData**() Obtain a CA certificate.
String	**getCharset**() Obtain the HTTP request encoding format.
String	**getClientCertData**() Obtain a client certificate.
String	**getClientKeyAlgo**() Obtain the encryption algorithm of the client private key.
String	**getClientKeyData**() Obtain a client private key.
String	**getClientKeyPassphrase**() Obtain the password of a client private key.
int	**getConnectionTimeout**() Obtain the connection timeout interval.
int	**getConnectTimeout**() Obtain the connection timeout interval.
Object	**getContent**() Obtain the content of an HTTP request.
String	**getContentType**() Obtain the content format of an HTTP request.
String	**getHeader**(String name) Obtain the HTTP request header with a specified name.
Set<String>	**getHeaderNames**() Obtain the name of the HTTP request header.
Map<String,String[]>	**getHeaders**() Obtain all request headers.
String[]	**getHeaders**(String name) Obtain all HTTP request headers with a specified name.
Set<String>	**getHeadersToSign**() Obtain a request header to be signed.

Returned Type	Method and Description
String	**getHttpProxy**() Obtain the HTTP proxy.
String	**getHttpsProxy**() Obtain the HTTPS proxy.
int	**getMaxConcurrentRequests**() Obtain the maximum number of concurrent requests.
int	**getMaxConcurrentRequests...**() Obtain the maximum number of concurrent requests per host.
String	**getMethod**() Obtain the HTTP method.
String[]	**getNoProxy**() Obtain a list of IP addresses that do not use the proxy.
String	**getParameter**(String name) Obtain the HTTP request parameters with a specified name.
Set<String>	**getParameterNames**() Obtain all HTTP request parameter names.
Map<String,String>	**getParameters**() Obtain the HTTP request parameters.
String	**getProxyPassword**() Obtain the proxy password.
String	**getProxyUsername**() Obtain the proxy username.
RequestConfig	**getRequestConfig**() Obtain request configuration information.
String	**getRequestId**() Obtain the request ID.
int	**getRequestTimeout**() Obtain the request timeout.
long	**getRollingTimeout**() Obtain the rolling timeout interval.
long	**getScaleTimeout**() Obtain the scaling timeout interval.

Returned Type	Method and Description
String	**getSecretKey**() Obtain the SecretKey of the request signature. Requests for which AccessKey and SecretKey are set use the **AK/SK signature algorithm** for signing.
okhttp3.TlsVersion[]	**getTlsVersions**() Obtain the TLS version.
String	**getUrl**() Obtain the URL.
String	**getUserAgent**() Obtain the user agent.
long	**getWebsocketPingInterval**() Obtain the WebSocket heartbeat interval.
long	**getWebsocketTimeout**() Obtain the WebSocket timeout interval.
boolean	**isRedirects**() Allow redirection or not.
boolean	**isSsl**() Check whether HTTPS is used. The default value is **false**.
boolean	**isSslRedirects**() Check whether to obtain the value of sslRedirects. The options are **true** and **false**.
boolean	**isTrustCerts**() Check whether all certificates are trusted. The options are **true** and **false**.
void	**setAccessKey**(String accessKey) Set the AccessKey of the request signature. Requests for which AccessKey and SecretKey are set use the **AK/SK signature algorithm** for signing.
void	**setBodyForm**(Map<String,String> content) Set the HTTP request content of the map type.
void	**setBodyText**(String content) Set the HTTP request content of the string type.
void	**setCaCertData**(String caCertData) Set the CA certificate.

Returned Type	Method and Description
void	**setCharset**(String charset) Set the HTTP request encoding format.
void	**setClientCertData**(String clientCertData) Set a client certificate.
void	**setClientKeyAlgo**(String clientKeyAlgo) Set the encryption algorithm of the client private key.
void	**setClientKeyData**(String clientKeyData) Set a client private key.
void	**setClientKeyPassphrase**(String clientKeyPassphrase) Set the password of a client private key.
void	**setConnectionTimeout**(int connectionTimeout) Set the connection timeout interval.
void	**setConnectTimeout**(int connectTimeout) Set the connection timeout interval.
void	**setContent**(Object content) Set the HTTP request content in object format.
void	**setContentType**(String contentType) Set the content type of an HTTP request.
void	**setHeader**(String name, String value) Set the request header with the specified name and value.
void	**setHeader**(String name, String[] value) Set the request header with the specified name and value.
void	**setHeaders**(Map<String,String> headers) Set the request header.
void	**setHeaderValues**(Map<String,String[]> headers) Set the request header.
void	**setHttpProxy**(String httpProxy) Set the HTTP proxy.
void	**setHttpsProxy**(String httpsProxy) Set the HTTPS proxy.
void	**setMaxConcurrentRequests**(int maxConcurrentRequests) Set the maximum number of concurrent requests.

Returned Type	Method and Description
void	**setMaxConcurrentRequests...**(int maxConcurrentRequestsPer-Host) Set the maximum number of concurrent requests per host.
void	**setMethod**(String method) Set the HTTP method.
void	**setNoProxy**(String[] noProxy) Set a list of IP addresses that do not use the proxy.
void	**setParameter**(String name, String value) Set the HTTP request parameters.
void	**setParameters**(java.util.Map<String,String> parameters) Set the HTTP request parameters.
void	**setProxyPassword**(String proxyPassword) Set a proxy password.
void	**setProxyUsername**(String proxyUsername) Set a proxy username.
void	**setRedirects**(boolean redirects) Set whether redirection is allowed.
void	**setRequestId**(String requestId) Set the request ID.
void	**setRequestTimeout**(int requestTimeout) Set the request timeout interval.
void	**setRollingTimeout**(long rollingTimeout) Set the rolling timeout interval.
void	**setScaleTimeout**(long scaleTimeout) Set the scaling timeout interval.
void	**setSecretKey**(String secretKey) Set the SecretKey of the request signature. Requests for which AccessKey and SecretKey are set use the **AK/SK signature algorithm** for signing.
void	**setSsl**(boolean ssl) Set whether HTTPS is used.
void	**setSslRedirects**(boolean sslRedirects) Set the value of sslRedirects.

Returned Type	Method and Description
void	**setTlsVersions**(okhttp3.TlsVersion[] tlsVersions) Set the TLS version.
void	**setTrustCerts**(boolean trustCerts) Set whether all certificates are trusted.
void	**setUrl**(String url) Set the URL.
void	**setUserAgent**(String userAgent) Set the user agent.
void	**setWebsocketPingInterval**(long websocketPingInterval) Set the WebSocket heartbeat interval.
void	**setWebsocketTimeout**(long websocketTimeout) Set the WebSocket timeout interval.

Method Details

- **public void addHeaderToSign(String headerName)**

  Add the header to the signature.

  Input Parameter

  headerName indicates the request header name.

- **public String getAccessKey()**

  Obtain the AccessKey of the request signature. Requests for which AccessKey and SecretKey are set use the **AK/SK signature algorithm** for signing.

  Returns

  AccessKey of the request signature.

- **public String getCaCertData()**

  Obtain a CA certificate.

  Returns

  CA certificate.

- **public String getCharset()**

  Obtain the HTTP request encoding format.

  Returns

  HTTP request encoding format.

- **public String getClientCertData()**

  Obtain a client certificate.

  Returns

  Client certificate.

- **public String getClientKeyAlgo()**

  Obtain the encryption algorithm of the client private key.

  Returns

  Encryption algorithm of the client private key.

- **public String getClientKeyData()**

  Obtain a client private key.

  Returns

  Client private key.

- **public String getClientKeyPassphrase()**

  Obtain the password of a client private key.

  Returns

  Password of a client private key.

- **public int getConnectionTimeout()**

  Obtain the connection timeout interval.

  Returns

  Connection timeout interval.

- **public int getConnectTimeout()**

  Obtain the connection timeout interval.

  Returns

  Connection timeout interval.

- **public Object getContent()**

  Obtain the content of an HTTP request.

  Returns

  HTTP request content.

- **public String getContentType()**

  Obtain the content format of an HTTP request.

  Returns

  Content format of an HTTP request.

- **public String getHeader(String name)**

  Obtain the HTTP request header with a specified name.

  Input Parameter

  name indicates the request header name.

  Returns

  Request header information with a specified name.

- **public Set<String> getHeaderNames()**

  Obtain the name of the HTTP request header.

  Returns

  Name of the HTTP request header.

- **public Map<String,String[]> getHeaders()**

  Obtain all request headers.

  Returns

All request headers.

- **public String[] getHeaders(String name)**

  Obtain all HTTP request headers with a specified name.

  Input Parameter

  name indicates the request header name.

  Returns

  All HTTP request headers with a specified name.

- **public Set<String> getHeadersToSign()**

  Obtain the request header in a signature.

  Returns

  Request header in a signature.

- **public String getHttpProxy()**

  Obtain the HTTP proxy.

  Returns

  HTTP proxy.

- **public String getHttpsProxy()**

  Obtain the HTTPS proxy.

  Returns

  HTTPS proxy.

- **public int getMaxConcurrentRequests()**

  Obtain the maximum number of concurrent requests.

  Returns

  Maximum number of concurrent requests.

- **public int getMaxConcurrentRequestsPerHost()**

  Obtain the maximum number of concurrent requests per host.

  Returns

  Maximum number of concurrent requests per host.

- **public String getMethod()**

  Obtain the HTTP method.

  Returns

  HTTP method.

- **public String[] getNoProxy()**

  Obtain a list of IP addresses that do not use the proxy.

  Returns

  A list of IP addresses that do not use the proxy.

- **public String getParameter(String name)**

  Obtain the HTTP request parameters with a specified name.

  Input Parameter

  name indicates the HTTP name.

  Returns

  HTTP request parameters with a specified name.

- **public Set<String> getParameterNames()**

  Obtain the HTTP request parameters.

  Returns

  HTTP request parameters.

- **public Map<String,String> getParameters()**

  Obtain the HTTP request parameters.

  Returns

  HTTP request parameters.

- **public String getProxyPassword()**

  Obtain a proxy password.

  Returns

  Proxy password.

- **public String getProxyUsername()**

  Obtain a proxy username.

  Returns

  Proxy username.

- **public RequestConfig getRequestConfig()**

  Obtain the request configuration.

  Returns

  Boolean value of sslRedirects.

- **public String getRequestId()**

  Obtain the request ID.

  Returns

  Request ID.

- **public int getRequestTimeout()**

  Obtain the request timeout interval.

  Returns

  Request timeout interval.

- **public long getRollingTimeout()**

  Obtain the rolling timeout interval.

  Returns

  Rolling timeout interval.

- **public long getScaleTimeout()**

  Obtain the scaling timeout interval.

  Returns

  Scaling timeout interval.

- **public String getSecretKey()**

  Obtain the SecretKey of the request signature. Requests for which AccessKey and SecretKey are set use the **AK/SK signature algorithm** for signing.

  Returns

  SecretKey of the request signature.

- **public okhttp3.TlsVersion[] getTlsVersions()**

  Obtain the TLS version.

  Returns

  TLS version.

- **public String getUrl()**

  Obtain the URL.

  Returns

  URL.

- **public String getUserAgent()**

  Obtain the user agent.

  Returns

  User agent.

- **public long getWebsocketPingInterval()**

  Obtain the WebSocket heartbeat interval.

  Returns

  WebSocket heartbeat interval.

- **public long getWebsocketTimeout()**

  Obtain the WebSocket timeout interval.

  Returns

  WebSocket timeout interval.

- **public boolean isRedirects()**

  Allow redirection or not.

  Returns

  true or false

- **public boolean isSsl()**

  Check whether HTTPS is used. The default value is **false**.

  Returns

  true or false

- **public boolean isSslRedirects()**

  Check whether to obtain the value of sslRedirects. The options are **true** and **false**.

  Returns

  Value of sslRedirects.

- **public boolean isTrustCerts()**

  Check whether all certificates are trusted.

  Returns

  All trusted certificates

- **public void setAccessKey(String accessKey)**

  Set the AccessKey of the request signature. Requests for which AccessKey and SecretKey are set use the **AK/SK signature algorithm** for signing.

  Input Parameter

accessKey indicates the AccessKey of the request signature.

- **public void setBodyForm(Map<String,String> content)**

  Set the HTTP request content of the map type.

  Input Parameter

  content indicates the HTTP request content.

- **public void setBodyText(String content)**

  Set the HTTP request content of the string type.

  Input Parameter

  content indicates the HTTP request content.

- **public void setCaCertData(String caCertData)**

  Set the CA certificate.

  Input Parameter

  caCertData indicates the CA certificate.

- **public void setCharset(String charset)**

  Set the HTTP request encoding format.

  Input Parameter

  charset indicates the encoding format of the HTTP request.

- **public void setClientCertData(String clientCertData)**

  Set a client certificate.

  Input Parameter

  clientCertData indicates a client certificate.

- **public void setClientKeyAlgo(String clientKeyAlgo)**

  Set the encryption algorithm of the client private key.

  Input Parameter

  clientKeyAlgo indicates the encryption algorithm of the client private key.

- **public void setClientKeyData(String clientKeyData)**

  Set a client private key.

  Input Parameter

  clientKeyData indicates a client private key.

- **public void setClientKeyPassphrase(String clientKeyPassphrase)**

  Set the password of a client private key.

  Input Parameter

  clientKeyPassphrase indicates the password of a client private key.

- **public void setConnectionTimeout(int connectionTimeout)**

  Set the connection timeout interval.

  Input Parameter

  connectionTimeout indicates the connection timeout interval.

- **public void setConnectTimeout(int connectTimeout)**

  Set the connection timeout interval.

  Input Parameter

  connectTimeout indicates the connection timeout interval.

- **public void setContent(Object content)**

  Set the HTTP request content of the string and file types.

  Input Parameter

  content indicates the HTTP request content.

- **public void setContentType(String contentType)**

  Set the content type of an HTTP request.

  Input Parameter

  contentType indicates the content type of an HTTP request.

- **public void setHeader(String name, String value)**

  Set the request header.

  Input Parameter

  – **name** indicates the request header name.

  – **value** indicates the request header value.

- **public void setHeader(String name, String[] value)**

  Set the request header.

  Input Parameter

  – **name** indicates the request header name.

  – **value** indicates the request header value.

- **public void setHeaders(Map<String,String> headers)**

  Set the request header.

  Input Parameter

  headers indicates the request header information.

- **public void setHeaderValues(Map<String,String[]> headers)**

  Set the request header.

  Input Parameter

  headers indicates the request header information.

- **public void setHttpProxy(String httpProxy)**

  Set the HTTP proxy.

  Input Parameter

  httpProxy indicates the HTTP proxy.

- **public void setHttpsProxy(String httpsProxy)**

  Set the HTTPS proxy.

  Input Parameter

  httpsProxy indicates the HTTPS proxy.

- **public void setMaxConcurrentRequests(int maxConcurrentRequests)**

  Set the maximum number of concurrent requests.

  Input Parameter

  maxConcurrentRequests:

- **public void setMaxConcurrentRequestsPerHost(int maxConcurrentRequestsPerHost**

  Set the maximum number of concurrent requests per host.

Input Parameter

maxConcurrentRequestsPerHost:

- **public void setMethod(String method)**

  Set the HTTP method.

  Input Parameter

  method indicates the HTTP method.

- **public void setNoProxy(String[] noProxy)**

  Set a list of IP addresses that do not use the proxy.

  Input Parameter

  noProxy indicates a list of IP addresses that do not use the proxy.

- **public void setParameter(String name, String value)**

  Set the HTTP request parameters.

  Input Parameter

  - **name** indicates the name of an HTTP request parameter.
  - **value** indicates the value of an HTTP request parameter.

- **public void setParameters(Map<String,String> parameters)**

  Set the HTTP request parameters.

  Input Parameter

  parameters indicates the HTTP request parameters.

- **public void setProxyPassword(String proxyPassword)**

  Set a proxy password.

  Input Parameter

  proxyPassword indicates the proxy password.

- **public void setProxyUsername(String proxyUsername)**

  Set a proxy username.

  Input Parameter

  proxyUsername indicates the proxy username.

- **public void setRedirects(boolean redirects)**

  Set whether redirection is allowed.

  Input Parameter

  redirects indicates whether redirection is allowed.

- **public void setRequestId(String requestId)**

  Set the request ID.

  Input Parameter

  requestId indicates the request ID.

- **public void setRequestTimeout(int requestTimeout)**

  Set the request timeout interval.

  Input Parameter

  requestTimeout indicates the request timeout interval.

- **public void setRollingTimeout(long rollingTimeout)**

  Set the rolling timeout interval.

Input Parameter

rollingTimeout indicates the rolling timeout interval.

- **public void setScaleTimeout(long scaleTimeout)**

  Set the scaling timeout interval.

  Input Parameter

  scaleTimeout indicates the scale timeout interval.

- **public void setSecretKey(String secretKey)**

  Set the SecretKey of the request signature. Requests for which AccessKey and SecretKey are set use the **AK/SK signature algorithm** for signing.

  Input Parameter

  secretKey indicates the SecretKey of the request signature.

- **public void setSsl(boolean ssl)**

  Set whether HTTPS is used.

  Input Parameter

  ssl indicates whether HTTPS is used.

- **public void setSslRedirects(boolean sslRedirects)**

  Set whether to obtain the value of sslRedirects. The options are **true** and **false**.

  Input Parameter

  sslRedirects indicates whether sslRedirects is set (**true**/**false**).

- **public void setTlsVersions(okhttp3.TlsVersion[] tlsVersions)**

  Set the TLS version.

  Input Parameter

  tlsVersions indicates the TLS version.

- **public void setTrustCerts(boolean trustCerts)**

  Set whether all certificates are trusted.

  Input Parameter

  trustCerts indicates whether all certificates are trusted.

- **public void setUrl(String url)**

  Set the URL.

  Input Parameter

  url indicates the URL.

- **public void setUserAgent(String userAgent)**

  Set the user agent.

  Input Parameter

  userAgent indicates the user agent.

- **public void setWebsocketPingInterval(long websocketPingInterval)**

  Set the WebSocket heartbeat interval.

  Input Parameter

  websocketPingInterval indicates the WebSocket heartbeat interval.

- **public void setWebsocketTimeout(long websocketTimeout)**

Set the WebSocket timeout interval.

Input Parameter

websocketTimeout indicates the WebSocket timeout interval.

2.4.13 JedisConfig

Path

com.roma.apic.livedata.config.v1.JedisConfig

Description

This class is used together with **RedisClient** to configure the Redis connection.

Example

```
importClass(com.roma.apic.livedata.client.v1.RedisClient);
importClass(com.roma.apic.livedata.config.v1.JedisConfig);
function execute(data) {
    var config = new JedisConfig();
    config.setIp(["1.1.1.1"]);
    config.setPort(["6379"]);
    config.setMode("SINGLE");
    var redisClient = new RedisClient(config);
    var count = redisClient.get("visit_count")
    if (!count)
    {
        redisClient.put("visit_count", 1);
    }else {
        redisClient.put("visit_count", parseInt(count) + 1);
    }
    return redisClient.get("visit_count");
}
```

Constructor Details

public JedisConfig()

Constructs a JedisConfig without parameters.

Method List

Returned Type	Method and Description
int	**getDatabase**() Obtain the Jedis database. The default value is **0**.
String[]	**getIp**() Obtain the IP address list of the Redis.
String	**getMaster**() Obtain the master name of the Jedis. This parameter is valid when **mode** is set to **MASTER_SLAVE**.

Returned Type	Method and Description
int	**getMaxAttempts**() Obtain the number of retry times of the Jedis. The default value is **10000**.
int	**getMaxIdel**() Obtain the maximum number of idle connections in the Jedis connection pool. The default value is **5**.
int	**getMaxWait**() Obtain the upper limit of the waiting time (in seconds) when the Jedis connection pool is exhausted. The default value is **60**.
String	**getMode**() Obtain the Jedis type. The value can be **SINGLE**, **CLUSTER**, or **MASTER_SLAVE**.
String	**getPassPhrase**() Obtain the password of the Jedis.
String[]	**getPort**() Obtain all port numbers.
int	**getSoTimeout**() Obtain the read timeout interval of the Jedis. The default value is **600**.
int	**getTimeout**() Obtain the timeout interval of the Jedis. The default value is **1000**.
void	**setDatabase**(int database) Set the database of the Jedis.
void	**setIp**(String[] ip) Set the IP address.
void	**setMaster**(String master) Set the master name of the Jedis. This parameter is valid when **mode** is set to **MASTER_SLAVE**.
void	**setMaxAttempts**(int maxAttempts) Set the number of retries of the Jedis. The default value is **10000**.
void	**setMaxIdel**(int maxIdel) Set the maximum number of idle connections in the Jedis connection pool. The default value is **5**.

Returned Type	Method and Description
void	setMaxWait(int maxWait) Set the upper limit of the waiting time when the Jedis connection pool is exhausted. The default value is **60**.
void	setMode(String mode) Set the Jedis type. The value can be **SINGLE**, **CLUSTER**, or **MASTER_SLAVE**.
void	setPassPhrase(String passPhrase) Set the password of the Jedis.
void	setPort(String[] port) Set the port number.
void	setSoTimeout(int soTimeout) Set the read timeout interval of the Jedis.
void	setTimeout(int timeout) Set the timeout interval of the Jedis.

Method Details

- **public int getDatabase()**

  Obtain the Redis database. The default value is **0**.

  Returns

  Database.

- **public String[] getIp()**

  Obtain all IP addresses.

  Returns

  String array of IP addresses.

- **public String getMaster()**

  Obtain the master name of the Redis. This parameter is valid when **mode** is set to **MASTER_SLAVE**.

  Returns

  Master name.

- **public int getMaxAttempts()**

  Obtain the number of retry times of the Redis. The default value is **10000**.

  Returns

  Number of retry times.

- **public int getMaxIdel()**

  Obtain the maximum number of idle connections in the Jedis connection pool. The default value is **5**.

  Returns

Maximum number of idle connections in the connection pool.

- **public int getMaxWait()**

  Obtain the upper limit of the waiting time (in seconds) when the Jedis connection pool is exhausted. The default value is **60**.

  Returns

  Upper limit of the waiting time when the connection pool is exhausted.

- **public String getMode()**

  Obtain the Redis type. The value can be **SINGLE**, **CLUSTER**, or **MASTER_SLAVE**.

  Returns

  Redis type.

- **public String getPassPhrase()**

  Obtain the password of the Redis.

  Returns

  Redis password.

- **public String[] getPort()**

  Obtain all port numbers.

  Returns

  String array of port numbers.

- **public int getSoTimeout()**

  Obtain the read timeout interval (in seconds) of the Jedis. The default value is **600**.

  Returns

  Value of soTimeout.

- **public int getTimeout()**

  Obtain the timeout interval (in seconds) of the Jedis. The default value is **1000**.

  Returns

  Timeout interval.

- **public void setDatabase(int database)**

  Set the database of the Redis.

  Input Parameter

  database indicates a database.

- **public void setIp(String[] ip)**

  Set the IP address.

  Input Parameter

  ip indicates an IP address.

- **public void setMaster(String master)**

  Set the master name of the Redis. This parameter is valid when **mode** is set to **MASTER_SLAVE**.

  Input Parameter

  master indicates the master name of the Redis.

- **public void setMaxAttempts(int maxAttempts)**

  Set the number of retries of the Jedis.

  Input Parameter

  maxAttempts indicates the number of retries.

- **public void setMaxIdel(int maxIdel)**

  Set the maximum number of idle connections in the Jedis connection pool. The default value is **5**.

  Input Parameter

  maxIdel indicates the maximum number of idle connections in the connection pool.

- **public void setMaxWait(int maxWait)**

  Set the upper limit of the waiting time (in seconds) when the Jedis connection pool is exhausted. The default value is **60**.

  Input Parameter

  maxWait indicates the upper limit of the waiting time when the connection pool is exhausted.

- **public void setMode(String mode)**

  Set the Redis type. The value can be **SINGLE**, **CLUSTER**, or **MASTER_SLAVE**.

  Input Parameter

  mode indicates the type.

- **public void setPassPhrase(String passPhrase)**

  Set the password of the Redis.

  Input Parameter

  passPhrase indicates the password.

- **public void setPort(String[] port)**

  Set the port number.

  Input Parameter

  port indicates the port number.

- **public void setSoTimeout(int soTimeout)**

  Set the read timeout interval (in seconds) of the Jedis. The default value is **600**.

  Input Parameter

  soTimeout indicates the read timeout interval.

- **public void setTimeout(int timeout)**

  Set the timeout interval of the Jedis.

  Input Parameter

  timeout indicates the timeout duration, in seconds.

2.4.14 JSON2XMLHelper

Path

com.huawei.livedata.util.JSON2XMLHelper

Description

This class is used to perform conversion between JSON and XML.

Method List

Returned Type	Method and Description
static String	**JSON2XML**(String json, boolean returnFormat) Convert from JSON to XML.
static String	**XML2JSON**(String xml) Convert from XML to JSON.

Method Details

- **public static String JSON2XML(String json, boolean returnFormat)**

  Convert from JSON to XML.

  Input Parameter

  - **json** indicates a character string in JSON format.
  - **returnFormat** indicates the return format.

  Returns

  Character string in the XML format.

- **public static String XML2JSON(String xml)**

  Convert from XML to JSON.

  Input Parameter

  xml indicates a character string in XML format.

  Returns

  Character string in the XML format.

2.4.15 JSONHelper

Path

com.huawei.livedata.lambdaservice.util.JSONHelper

Description

This class is used to perform conversion between JSON and XML and between JSON and Map.

Method List

Returned Type	Method and Description
static String	**json2Xml**(String json) Convert from JSON to XML.
static String	**xml2Json**(String xml) Convert from XML to JSON.
static String	**json2XmlWithoutType**(String json) Convert from JSON to XML.
static HashMap	**jsonToMap**(String json) Convert from JSON to Map.

Method Details

- **public static String json2Xml(String json)**

  Convert from JSON to XML.

  Input Parameter

  json indicates a character string in JSON format.

  Returns

  Character string in the XML format.

- **public static String xml2Json(String xml)**

  Convert from XML to JSON.

  Input Parameter

  xml indicates a character string in XML format.

  Returns

  Character string in the JSON format.

- **public static String json2XmlWithoutType(String json)**

  Convert from JSON to XML.

  Input Parameter

  json indicates a character string in JSON format.

  Returns

  Character string in the XML format.

- **public static HashMap jsonToMap(String json)**

  Convert from JSON to Map.

  Input Parameter

  json indicates a character string in JSON format.

  Returns

  Character string in the Map format.

2.4.16 JsonUtils

Path

com.roma.apic.livedata.common.v1.JsonUtils

Description

This class is used to provide the conversion between JSON and objects and between JSON and XML.

Example

```
importClass(com.roma.apic.livedata.common.v1.JsonUtils);
function execute(data) {
    return JsonUtils.convertJsonToXml('{"a":1}')
}
```

Method List

Returned Type	Method and Description
static String	**convertJsonToXml**(String json) Convert JSON into XML.
static String	**convertJsonToXml**(String json, String rootName) Convert JSON into XML.
static <T> T	**toBean**(String json, Class<T> clazz) Convert JSON into an object.
static String	**toJson**(Object object) Convert an object to a character string in JSON format.
static String	**toJson**(Object object, Map<String,Object> config) Convert an object to a character string in JSON format and use the configuration in the config file. For example, you can set "date-format" in config to "yyyy-MM-dd HH:mm:ss".
static Map<String,Object>	**toMap**(String json) Convert JSON into MAP.

Method Details

- **public static String convertJsonToXml(String json)**

  Convert JSON into XML.

  Input Parameter

  json indicates a character string in JSON format.

Returns

Character string in XML format

- **public static String convertJsonToXml(String json, String rootName)**

  Convert JSON into XML.

  Input Parameter

  – **json** indicates a character string in JSON format.

  – **rootName** indicates the root node name of the XML file.

  Returns

  Character string in XML format

- **public static <T> T toBean(String json, Class<T> clazz)**

  Convert JSON into an object.

  Input Parameter

  – **json** indicates a character string in JSON format.

  – **clazz** indicates the class.

  Returns

  Class object.

- **public static String toJson(Object object)**

  Convert an object to a character string in JSON format.

  Input Parameter

  object indicates an object.

  Returns

  Character string in JSON format obtained after conversion.

- **public static String toJson(Object object, Map<String,Object> config)**

  Convert an object to a character string in JSON format and use the configuration in the config file.

  For example, you can set "date-format" in config to "yyyy-MM-dd HH:mm:ss".

  Input Parameter

  – **object** indicates an object.

  – **config** indicates the configuration used for conversion.

  Returns

  Character string in JSON format obtained after conversion.

- **public static Map<String,Object> toMap(String json)**

  Convert JSON into MAP.

  Input Parameter

  json indicates a character string in JSON format.

  Returns

  Character string in MAP format.

2.4.17 JWTUtils

Path

com.huawei.livedata.util.JWTUtils

Description

This class is used to generate an SHA256 signature.

Method List

Returned Type	Method and Description
static String	createToken(String appId, String appKey, String timestamp) Generate SHA256 signature.

Method Details

public static String createToken(String appId, String appKey, String timestamp)

Generate SHA256 signature.

Input Parameter

- **appId** indicates the integration application ID.
- **appKey** indicates the key of the integration application.
- **timestamp** indicates the timestamp.

Returns

SHA256 signature.

2.4.18 KafkaConsumer

Path

com.roma.apic.livedata.client.v1.KafkaConsumer

Description

This class is used to consume Kafka messages.

Example

```
importClass(com.roma.apic.livedata.client.v1.KafkaConsumer);
importClass(com.roma.apic.livedata.config.v1.KafkaConfig);

var kafka_brokers = '1.1.1.1:26330,2.2.2.2:26330'
var topic = 'YourKafkaTopic'
var group = 'YourKafkaGroupId'
```

```
function execute(data) {
    var config = KafkaConfig.getConfig(kafka_brokers, group)
    var consumer = new KafkaConsumer(config)
    var records = consumer.consume(topic, 5000, 10);
    var res = []
    var iter = records.iterator()
    while (iter.hasNext()) {
        res.push(iter.next())
    }
    return JSON.stringify(res);
}
```

Constructor Details

public KafkaConsumer(Map configs)

Constructs a Kafka message consumer.

Parameter: **configs** indicates configuration information of the Kafka.

Method List

Returned Type	Method and Description
List<String>	consume(String topic, long timeout, long maxItems) Consume messages.

Method Details

public List<String> consume(String topic, long timeout, long maxItems)

Consume messages.

Input Parameter

- **topic** indicates a message queue.
- **timeout**: indicates the read timeout interval (maximum value: 30,000 ms). Set this parameter to a value less than the backend timeout of the frontend API.
- **maxItems** indicates the maximum number of messages that can be read.

Returns

Message array that has been consumed by Kafka. The content of multiple messages forms an array.

2.4.19 KafkaProducer

Path

com.roma.apic.livedata.client.v1.KafkaProducer

Description

This class is used to produce Kafka messages.

Example

```
importClass(com.roma.apic.livedata.client.v1.KafkaProducer);
importClass(com.roma.apic.livedata.config.v1.KafkaConfig);

var kafka_brokers = '1.1.1.1:26330,2.2.2.2:26330'
var topic = 'YourKafkaTopic'

function execute(data) {
    var config = KafkaConfig.getConfig(kafka_brokers, null)
    var producer = new KafkaProducer(config)
    var record = producer.produce(topic, "hello, kafka.")
    return {
        offset: record.offset(),
        partition: record.partition(),
        code: 0,
        message: "OK"
    }
}
```

Constructor Details

public KafkaProducer(Map configs)

Constructs a Kafka message producer.

Parameter: **configs** indicates configuration information of the Kafka.

Method List

Returned Type	Method and Description
org.apache.kafka.clients.producer.**RecordMetadata**	**produce**(String topic, String message) Produce messages.

NOTE

The produce(String topic, String message) method cannot be directly returned. Otherwise, the returned information is empty. For example, do not use the **return record** statement directly in the preceding example. Otherwise, the returned information is empty.

Method Details

public org.apache.kafka.clients.producer.RecordMetadata produce(String topic, String message)

Produce messages.

Input Parameter

- **topic** indicates a message queue.
- **message** indicates the message content.

Returns

Message record.

2.4.20 KafkaConfig

Path

com.roma.apic.livedata.config.v1.KafkaConfig

extends

java.util.Properties

Description

This class is used together with **KafkaProducer** or **KafkaConsumer** to configure a Kafka client.

Constructor Details

public KafkaConfig()

Constructs a KafkaConfig without parameters.

Method List

Returned Type	Method and Description
static **KafkaConfig**	**getConfig**(String servers, String groupId) Obtain a configuration for accessing Kafka provided by MQS (sasl_ssl disabled).
static **KafkaConfig**	**getSaslConfig**(String servers, String groupId, String username, String password) Obtain a configuration for accessing Kafka provided by MQS (sasl_ssl enabled).

Method Details

- **public static KafkaConfig getConfig(String servers, String groupId)**

  Access Kafka (sasl_ssl not enabled) provided by MQS.

  Input Parameter

  – **servers** indicates the bootstrap server information in kafkaConfig.

  – **groupId** indicates the group ID in kafkaConfig.

  Returns

  KafkaConfig object.

- **public static KafkaConfig getSaslConfig(String servers, String groupId, String username, String password)**

  Access Kafka (sasl_ssl enabled) provided by MQS.

  Input Parameter

  – **servers** indicates the bootstrap server information in kafkaConfig.

- **groupId** indicates the group ID in kafkaConfig.
- **username** indicates the username.
- **password** indicates the password.

Returns

KafkaConfig object.

2.4.21 MD5Encoder

Path

com.huawei.livedata.lambdaservice.util.MD5Encoder

Description

This class is used to calculate the MD5 value.

Method List

Returned Type	Method and Description
static String	md5(String source) Calculate the MD5 value of a character string.

Method Details

public static String md5(String source)

Calculate the MD5 value of a character string.

Input Parameter

source indicates the character string for which the MD5 value needs to be calculated.

Returns

MD5 value of a character string.

2.4.22 Md5Utils

Path

com.roma.apic.livedata.common.v1.Md5Utils

Description

This class is used to calculate the MD5 value.

Example

```
importClass(com.roma.apic.livedata.common.v1.Md5Utils);
function execute(data) {
    var sourceCode = "Hello world!";
    return Md5Utils.encode(sourceCode);
}
```

Method List

Returned Type	Method and Description
static String	**encode**(String content) Calculate the MD5 value of a character string.

Method Details

public static String encode(String content)

Calculate the MD5 value of a character string.

Input Parameter

content: character string whose MD5 is to be calculated.

Returns

MD5 value of a character string.

2.4.23 QueueConfig

Path

com.roma.apic.livedata.config.v1.QueueConfig

Description

This class is used with **RabbitMqConfig** and **RabbitMqProducer** to configure a queue.

Constructor Details

public QueueConfig(String queueName, boolean durable, boolean exclusive, boolean autoDelete, Map<String, Object> arguments)

Constructs a queue configuration.

Parameters:

- **queueName** indicates the queue name.
- **durable** indicates whether persistency is supported. The value **true** indicates persistency is supported, and the value **false** indicates that persistency is not supported.
- **exclusive** indicates whether a queue is exclusive. The value **true** indicates that a queue is exclusive, that is, a queue can be consumed by only one consumer.

- **autoDelete** indicates whether automatic deletion is supported. The value **true** indicates that automatic deletion is supported.
- **arguments** indicates other attributes.

2.4.24 RabbitMqConfig

Path

com.roma.apic.livedata.config.v1.RabbitMqConfig

Description

This class is used with **ConnectionConfig**, **QueueConfig**, **ExchangeConfig**, and **RabbitMqProducer** to configure a RabbitMQ client.

Constructor Details

public RabbitMqConfig(ConnectionConfig connectionConfig, QueueConfig queueConfig, ExchangeConfig exchangeConfig)

Constructs a RabbitMQ client configuration.

Parameters:

- **connectionConfig** indicates the client connection configuration.
- **queueConfig** indicates the queue configuration.
- **exchangeConfig** indicates the switch configuration.

2.4.25 RabbitMqProducer

Path

com.roma.apic.livedata.client.v1.RabbitMqProducer

Description

This class is used to produce RabbitMQ messages. If no exception occurs during message sending, messages are sent successfully. If an exception occurs during message sending, messages fail to be sent.

Example

- Use the direct switch to generate messages and route the messages to the queue in which the bindingKey and routingKey are fully matched.

```
importClass(com.roma.apic.livedata.client.v1.RabbitMqProducer);
importClass(com.roma.apic.livedata.config.v1.RabbitMqConfig);
importClass(com.roma.apic.livedata.config.v1.QueueConfig);
importClass(com.roma.apic.livedata.config.v1.ExchangeConfig);
importClass(com.roma.apic.livedata.config.v1.ConnectionConfig);

function execute(data) {
    var connectionConfig = new ConnectionConfig("10.10.10.10", 5672, "admin", "123456");
    var queueConfig = new QueueConfig("directQueue", false, false, false, null);
    var exchangeConfig = new ExchangeConfig("directExchange", "direct", true, false, false, null);
    var config = new RabbitMqConfig(connectionConfig, queueConfig, exchangeConfig);
    var producer = new RabbitMqProducer(config);
    producer.produceWithDirectExchange("direct.exchange", "PERSISTENT_TEXT_PLAIN", "direct
exchange message");

    return "produce successful.";
}
```

- Use the topic switch to generate messages and route the messages to the queue in which the bindingKey and routingKey are matched in fuzzy mode.

```
importClass(com.roma.apic.livedata.client.v1.RabbitMqProducer);
importClass(com.roma.apic.livedata.config.v1.RabbitMqConfig);
importClass(com.roma.apic.livedata.config.v1.QueueConfig);
importClass(com.roma.apic.livedata.config.v1.ExchangeConfig);
importClass(com.roma.apic.livedata.config.v1.ConnectionConfig);

function execute(data) {
    var connectionConfig = new ConnectionConfig("10.10.10.10", 5672, "admin", "123456");
    var queueConfig = new QueueConfig ("topicQueue", false, false, false, null);
    var exchangeConfig = new ExchangeConfig("topicExchange", "topic", true, false, false, null);
    var config = new RabbitMqConfig(connectionConfig, queueConfig, exchangeConfig);

    var producer = new RabbitMqProducer(config);
   producer.produceWithTopicExchange("topic.#", "topic.A", null, "message");
    return "produce successful.";
}
```

- Use the fanout switch to generate messages and route all messages sent to the exchange to all the queues bound to it.

```
importClass(com.roma.apic.livedata.client.v1.RabbitMqProducer);
importClass(com.roma.apic.livedata.config.v1.RabbitMqConfig);
importClass(com.roma.apic.livedata.config.v1.QueueConfig);
importClass(com.roma.apic.livedata.config.v1.ExchangeConfig);
importClass(com.roma.apic.livedata.config.v1.ConnectionConfig);

function execute(data) {
    var connectionConfig = new ConnectionConfig("10.10.10.10", 5672, "admin", "123456");
    var queueConfig = new QueueConfig ("fanoutQueue", false, false, false, null);
    var exchangeConfig = new ExchangeConfig ("fanoutExchange", "fanout", true, false, null)
    var config = new RabbitMqConfig(connectionConfig, queueConfig, exchangeConfig);

    var producer = new RabbitMqProducer(config);
    producer.produceWithFanoutExchange(null, "message")

    return "produce successfull"
}
```

Constructor Details

public RabbitMqProducer(RabbitMqConfig rabbitMqConfig)

Constructs a RabbitMQ message producer.

Parameter: **rabbitMqConfig** indicates configuration information of RabbitMQ.

Method List

Returned Type	Method and Description
void	produceWithDirectExchange(String routingKey, String props, String message) Use the direct switch to generate messages and route the messages to the queue in which the bindingKey and routingKey are fully matched.

Returned Type	Method and Description
void	produceWithTopicExchange(String bindingKey, String routingKey, String props, String message) Use the topic switch to generate messages and route the messages to the queue in which the bindingKey and routingKey are matched in fuzzy mode.
void	produceWithFanoutExchange(String props, String message) Use the fanout switch to generate messages and route all messages sent to the exchange to all the queues bound to it.

Method Details

- **public void produceWithDirectExchange(String routingKey, String props, String message)**

  Use the direct switch to generate messages and route the messages to the queue in which the bindingKey and routingKey are fully matched.

  Input Parameters

  - **routingKey** indicates the message routing key.
  - **props** indicates the message persistency setting, which is optional.
  - **message** indicates the message content.

- **public void produceWithTopicExchange(String bindingKey, String routingKey, String props, String message)**

  Use the topic switch to generate messages and route the messages to the queue in which the bindingKey and routingKey are matched in fuzzy mode.

  Input Parameters

  - **bindingKey** indicates the queue binding key.
  - **routingKey** indicates the message routing key.
  - **props** indicates the message persistency setting, which is optional.
  - **message** indicates the message content.

- **produceWithFanoutExchange(String props, String message)**

  Use the fanout switch to generate messages and route all messages sent to the exchange to all the queues bound to it.

  Input Parameters

  - **props** indicates the message persistency setting, which is optional.
  - **message** indicates the message content.

2.4.26 RedisClient

Path

com.roma.apic.livedata.client.v1.RedisClient

Description

This class is used to connect to the Redis or read the Redis cache. If JedisConfig is not specified, connect to the default Redis provided by the function API of the custom backend.

Example

```
importClass(com.roma.apic.livedata.client.v1.RedisClient);
function execute(data) {
    var redisClient = new RedisClient;
    var count = redisClient.get("visit_count")
    if (!count)
    {
        redisClient.put("visit_count", 1);
    }else {
        redisClient.put("visit_count", parseInt(count) + 1);
    }
    return redisClient.get("visit_count");
}
```

Constructor Details

public RedisClient()

Constructs a RedisClient and connects it to the default Redis provided by the function API (livedata) of the custom backend.

public RedisClient(JedisConfig jedisConfig)

Constructs a RedisClient using jedisConfig.

Parameter **jedisConfig** indicates the RedisClient information.

Method List

Returned Type	Method and Description
String	**get**(String key) Obtain the value corresponding to the key in the Redis cache.
String	**put**(String key, int expireTime, String value) Update the Redis cache content and expiration time, and return the execution result.
String	**put**(String key, String value) Update the Redis cache content and return the execution result.
Long	**remove**(String key) Delete cached messages of a specified key value.

Method Details

- **public String get(String key)**

Obtain the value corresponding to the key in the Redis cache.

Input Parameter

key indicates the key value.

Returns

Obtain the value of the key in the Redis cache.

- **public String put(String key, int expireTime, String value)**

  Update the Redis cache and expiration time.

  Input Parameter

  - **key** indicates the key value of the cache to be updated.
  - **expireTime** indicates the expiration time of the cache content to be updated, in seconds.
  - **value** indicates the value of the cache to be updated.

  Returns

  Execution result.

- **public String put(String key, String value)**

  Update the Redis cache.

  Input Parameter

  - **key** indicates the key value of the cache to be updated.
  - **value** indicates the value of the cache to be updated.

  Returns

  Execution result.

- **public Long remove(String key)**

  Delete cached messages of a specified key value.

  Input Parameter

  key indicates the key value of the cache to be deleted.

  Returns

  Execution result.

2.4.27 RomaWebConfig

Path

com.huawei.livedata.lambdaservice.config.RomaWebConfig

Description

This class is used to obtain the ROMA configuration.

Method List

Returned Type	Method and Description
static String	getAppConfig(String key) Obtain the configuration of an integration application based on the config key.

Method Details

public

Obtain configurations based on the config key.

Input Parameter

key indicates the key of the integration application.

Returns

Configuration of the integration application.

2.4.28 RSAUtils

Path

com.roma.apic.livedata.common.v1.RSAUtils

Description

This class is used to provide the RSA encryption and decryption methods.

Example

Use the following Java code to generate a public key and a private key:

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.util.Base64;

public class Main {

    public static void main(String[] args) {
        try {
            KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
            keyPairGenerator.initialize(1024);
            KeyPair keyPair = keyPairGenerator.generateKeyPair();
            PublicKey publicKey = keyPair.getPublic();
            System.out.println("publicKey:" + new 
String(Base64.getEncoder().encode(publicKey.getEncoded())));

            PrivateKey privateKey = keyPair.getPrivate();
            System.out.println("privateKey:" + new 
String(Base64.getEncoder().encode(privateKey.getEncoded())));
        } catch (Exception e) {
            e.printStackTrace();
            return;
        }
    }
}
```

Add the public key and private key to the following code:

```
importClass(com.roma.apic.livedata.common.v1.RSAUtils);
importClass(com.roma.apic.livedata.common.v1.Base64Utils);

function execute(data) {
    var publicKeyString = "MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDd4CRRppmYVlFl3dX4iVGN
+2Twy5gLeEPRbvhOko/xFipGF7XV0weTp4wCakgdnm+DR4gBBrQtfAuKwYIBPIr
+C1FI5sKYA3NxazDWUcXR3xlPM5D0DWjacjcMjnaj2v21WZxGpwHZHQ9TLd4OBBq3fva1r/
cE8s1Lji5QeFiklwIDAQAB";

    var privateKeyString = "**********";

    var publicKey = RSAUtils.getPublicKey(publicKeyString)
    var privateKey = RSAUtils.getPrivateKey(privateKeyString)

    var origin = "hello rsa"
    var encrypted = RSAUtils.encrypt(Base64Utils.encode(origin), publicKey)

    var decrypted = RSAUtils.decrypt(encrypted, privateKey)
    return decrypted
}
```

Constructor Details

public RSAUtils()

Constructs an RSAUtils class without parameters.

Method List

Returned Type	Method and Description
static byte[]	**decodeBase64**(String base64) Decode a Base64 character string to binary data.
static byte[]	**decrypt**(java.security.PrivateKey privateKey, byte[] encryptData) Decrypt data using the RSA algorithm.
static String	**decrypt**(String source, java.security.interfaces.RSAPrivateKey privateKey) Decrypt data using the RSA algorithm (the source is encoded using Base64).
static String	**decrypt**(String source, java.security.interfaces.RSAPrivateKey privateKey, Map<String, String> config) Decrypt data using the RSA algorithm (the source is encoded using Base64).

Returned Type	Method and Description
static String	**decrypt**(byte[] source, java.security.interfaces.RSAPrivateKey privateKey) Decrypt data using the RSA algorithm.
static String	**decrypt**(byte[] source, java.security.interfaces.RSAPrivateKey privateKey, Map<String, String> config) Decrypt data using the RSA algorithm.
static String	**encodeBase64**(byte[] bytes) Encode binary data to a Base64 character string.
static byte[]	**encrypt**(java.security.PublicKey publicKey, byte[] source) Encrypt data using the RSA algorithm.
static String	**encrypt**(String source, java.security.PublicKey publicKey) Encrypt data using the RSA algorithm (both the source and returned data is encoded using Base64).
static String	**encrypt**(String source, java.security.PublicKey publicKey, Map<String, String> config) Encrypt data using the RSA algorithm (both the source and returned data is encoded using Base64).
static String	**encrypt**(byte[] source, java.security.PublicKey publicKey) Encrypt data using the RSA algorithm (the returned data is encoded using Base64).
static String	**encrypt**(byte[] source, java.security.PublicKey publicKey, Map<String, String> config) Encrypt data using the RSA algorithm (the returned data is encoded using Base64).
static java.security.interfaces.RSAPrivateKey	**getPrivateKey**(byte[] privateKeyByte) Create an RSA private key by using a private key byte array.
static java.security.interfaces.RSAPrivateKey	**getPrivateKey**(String privateKeyByte) Create an RSA private key by using a Base64-encoded private key.
static java.security.interfaces.RSAPrivateKey	**getPrivateKey**(String modulus, String exponent) Create an RSA private key by using the modulus and exponent.
static java.security.interfaces.RSAPublicKey	**getPublicKey**(byte[] publicKeyByte) Create an RSA public key by using a public key byte array.

Returned Type	Method and Description
static java.security.interfaces.RSAPublicKey	**getPublicKey**(String publicKeyByte) Create an RSA public key by using a Base64-encoded public key.
static java.security.PublicKey	**getPublicKey**(String modulus, String exponent) Create an RSA public key by using the modulus and exponent.

Method Details

- **public static byte[] decodeBase64(String base64)**

  Decode a Base64 character string to binary data.

  Input Parameter

  base64 indicates the data encoded using Base64.

  Returns

  Data decoded by using Base64

- **public static byte[] decrypt(java.security.PrivateKey privateKey, byte[] encryptData)**

  Decrypt data using the RSA algorithm.

  Input Parameter

  – **privateKey** indicates a private key.
  – **encryptData** indicates the data to be decrypted.

  Returns

  Decrypted data.

- **public static String decrypt(String source, java.security.interfaces.RSAPrivateKey privateKey)**

  Decrypt data using the RSA algorithm.

  Input Parameter

  – **source** indicates the Base64 code of the data to be decrypted.
  – **privateKey** indicates a private key.

  Returns

  Decrypted data.

- **public static String decrypt(String source, java.security.interfaces.RSAPrivateKey privateKey, Map<String,String>config)**

  Decrypt data using the RSA algorithm.

  Input Parameter

  – **source** indicates the Base64 code of the data to be decrypted.
  – **privateKey** indicates a private key.
  – **config** indicates the decryption configuration. The options are as follows:

transformation: specifies the decryption algorithm/mode/padding, for example, **RSA/ECB/OAEPPadding**. For details, see the **parameter description**.

Returns

Decrypted data.

- **public static String decrypt(byte[] source, java.security.interfaces.RSAPrivateKey privateKey)**

  Decrypt data using the RSA algorithm.

  Input Parameter

  - **source** indicates data to be decrypted.
  - **privateKey** indicates a private key.

  Returns

  Decrypted data.

- **public static String decrypt(byte[] source, java.security.interfaces.RSAPrivateKey privateKey, Map<String,String>config)**

  Decrypt data using the RSA algorithm.

  Input Parameter

  - **source** indicates data to be decrypted.
  - **privateKey** indicates a private key.
  - **config** indicates the decryption configuration. The options are as follows:

    transformation: specifies the decryption algorithm/mode/padding, for example, **RSA/ECB/OAEPPadding**. For details, see the **parameter description**.

  Returns

  Decrypted data.

- **public static String encodeBase64(byte[] bytes)**

  Encode binary data to a Base64 character string.

  Input Parameter

  bytes indicates data to be encoded.

  Returns

  Base64 encoding.

- **public static byte[] encrypt(java.security.PublicKey publicKey, byte[] source)**

  Encrypt data using the RSA algorithm.

  Input Parameter

  - **publicKey** indicates a public key.
  - **source** indicates the content to be encrypted.

  Returns

  Encrypted data content.

- **public static String encrypt(String source, java.security.PublicKey publicKey)**

  Encrypt data using the RSA algorithm.

Input Parameter

- **source** indicates the Base64 code of the data to be encrypted.
- **publicKey** indicates a public key.

Returns

Base64 code of the encrypted data content.

- **public static String encrypt(String source, java.security.PublicKey publicKey, Map<String, String> config)**

  Encrypt data using the RSA algorithm.

  Input Parameter

  - **source** indicates the Base64 code of the data to be encrypted.
  - **publicKey** indicates a public key.
  - **config** indicates the encryption option. The options are as follows:

    transformation: specifies the encryption algorithm/mode/padding, for example, **RSA/ECB/OAEPPadding**. For details, see the **parameter description**.

- **public static String encrypt(byte[] source, java.security.PublicKey publicKey)**

  Encrypt data using the RSA algorithm.

  Input Parameter

  - **source** indicates the content to be encrypted.
  - **publicKey** indicates a public key.

  Returns

  Base64 code of the encrypted data content.

- **public static String encrypt(byte[] source, java.security.PublicKey publicKey, Map<String, String> config)**

  Encrypt data using the RSA algorithm.

  Input Parameter

  - **source** indicates the content to be encrypted.
  - **publicKey** indicates a public key.
  - **config** indicates the encryption option. The options are as follows:

    transformation: specifies the encryption algorithm/mode/padding, for example, **RSA/ECB/OAEPPadding**. For details, see the **parameter description**.

  Returns

  Base64 code of the encrypted data content.

- **public static java.security.interfaces.RSAPrivateKey getPrivateKey(byte[] privateKeyByte)**

  Create an RSA private key by using the private key of the x509 format.

  Input Parameter

  privateKeyByte indicates the private key encoded in x509 format

  Returns

  Private key.

- **public static java.security.interfaces.RSAPrivateKey getPrivateKey(String privateKeyByte)**

  Create an RSA private key by using the private key of the x509 format.

  Input Parameter

  privateKeyByte indicates the private key encoded in x509 format

  Returns

  Private key.

- **public static java.security.interfaces.RSAPrivateKey getPrivateKey(String modulus, String exponent)**

  Create an RSA private key by using the modulus and exponent.

  Input Parameter

  - **modulus** indicates the modulus required for generating a private key.
  - **exponent** indicates the exponent required for generating a private key.

  Returns

  RSA private key.

- **public static java.security.interfaces.RSAPublicKey getPublicKey(byte[] publicKeyByte)**

  Create an RSA public key by using a public key encoded in x509 format.

  Input Parameter

  publicKeyByte indicates the public key encoded in x509 format.

  Returns

  Public key.

- **public static java.security.PublicKey getPublicKey(String modulus, String exponent)**

  Create an RSA public key by using the modulus and exponent.

  Input Parameter

  - **modulus** indicates the modulus required for generating a public key.
  - **exponent** indicates the exponent required for generating a public key.

  Returns

  RSA public key.

2.4.29 SapRfcClient

Path

com.roma.apic.livedata.client.v1.SapRfcClient

Description

This class is used to access SAP functions in RFC mode.

Example

```
importClass(com.roma.apic.livedata.client.v1.SapRfcClient);
importClass(com.roma.apic.livedata.config.v1.SapRfcConfig);
```

```
function execute(data) {
    var config = new SapRfcConfig();
    config.put("jco.client.ashost", "10.95.152.107");//Server
    config.put("jco.client.sysnr", "00"); //Instance ID
    config.put("jco.client.client", "400"); //SAP group
    config.put("jco.client.user", "SAPIDES");//SAP username
    config.put("jco.client.passwd", "****");//Password
    config.put("jco.client.lang", "zh");//Login language
    config.put("jco.destination.pool_capacity", "3");//Maximum number of connections
    config.put("jco.destination.peak_limit", "10");//Maximum number of connection threads
    var client = new SapRfcClient(config);
    var res = client.executeFunction("FUNCTION1", {
        "A":"200",
        "B":"2",
    })
    return res
}
```

Constructor Details

public SapRfcClient(SapRfcConfig config)

Constructs a SapRfcClient that contains the **SapRfcConfig** configuration information.

Parameter: **config** indicates the SapRfcClient configuration information.

Method List

Returned Type	Method and Description
Map<String, Object>	**executeFunction**(String functionName, Map<String, Object> params) Access SAP functions in RFC mode.

Method Details

executeFunction(String functionName, Map<String, Object> params)

Access SAP functions in RFC mode.

Input Parameter

- **functionName** indicates a function name.
- **params** indicates the input parameters of the SAP function.

Returns

Output parameters of the SAP function.

2.4.30 SapRfcConfig

Path

com.roma.apic.livedata.config.v1.SapRfcConfig

extends

java.util.Properties

Description

This class is used together with **SapRfcClient** to configure the SAP client.

Method List

Returned Type	Method and Description
Object	**put**(String key, Object value) Set configuration parameters.

Method Details

public Object put(String key, Object value)

Set configuration parameters.

Input Parameter

- **key** indicates the key in configuration information.
- **value** indicates the key value in configuration information.

  The following configurations are supported:
  - jco.client.ashost: SAP server IP address
  - jco.client.sysnr: system ID
  - jco.client.client: SAP group
  - jco.client.user: SAP username
  - jco.client.passwd: password
  - jco.client.lang: login language
  - jco.destination.pool_capacity: maximum number of connections
  - jco.destination.peak_limit: maximum number of connection threads
  - apic.async: indicates whether asynchronous calling is used. The value **true** indicates asynchronous calling, and the value **false** indicates synchronous calling. The default value is **false**.

Returns

Key values.

2.4.31 SoapClient

Path

com.roma.apic.livedata.client.v1.SoapClient

Description

This class is used to send SOAP requests.

Example

```
importClass(com.roma.apic.livedata.client.v1.SoapClient);
importClass(com.roma.apic.livedata.config.v1.SoapConfig);
importClass(com.roma.apic.livedata.common.v1.XmlUtils);

function execute(data) {
    var soap = new SoapConfig();
    soap.setUrl("http://test.webservice.com/ws");
    soap.setNamespace("http://spring.io/guides/gs-producing-web-service");
    soap.setOperation("getCountryRequest");

    soap.setNamespacePrefix("ser");
    soap.setBodyPrefix("ser");
    soap.setEnvelopePrefix("soapenv");
    var content = {
        "getCountryRequest": {
            "ser:name": "Spain"
        },
    };
    soap.setContent(content);

    var client = new SoapClient(soap);
    var result = client.execute();
    var body = result.getBody();

    return XmlUtils.toJson(body);
}
```

Constructor Details

public SoapClient(SoapConfig soapCfg)

Constructs a SOAP request that contains the **SoapConfig** information.

Parameter **soapCfg** indicates the SoapClient information.

Method List

Returned Type	Method and Description
APIConnect Response	execute() Send SOAP requests.

2.4.32 SoapConfig

Path

com.roma.apic.livedata.config.v1.SoapConfig

Description

This class is used together with **SoapClient** to configure SOAP requests.

Constructor Details

public SoapConfig()

Constructs a SoapConfig without parameters.

Method List

Returned Type	Method and Description
String	**buildSoapMessage**() Construct a SOAP request packet.
String	**getBodyPrefix**() Obtain the node prefix of a request packet.
String	**getCharset**() Obtain the HTTP request encoding format.
int	**getConnectTimeout**() Obtain the connection timeout interval.
Object	**getContent**() Obtain the request content.
String	**getContentType**() Obtain the packet parameter type.
String	**getEnvelopePrefix**() Obtain the envelope prefix.
String	**getHeader**(String name) Obtain the request header value based on the request header name.
Map<String,String>	**getHeaders**() Obtain request header information.
String	**getMethod**() Obtain the request method.
String	**getNamespace**() Obtain the namespace.
String	**getNamespacePrefix**() Obtain the namespace prefix.
String	**getOperation**() Obtain the operation name.
String	**getParameter**(String name) Obtain SOAP request parameters based on the specified name.
Map<String,String>	**getParameters**() Obtains the SOAP request parameters.

Returned Type	Method and Description
String	**getProtocol**() Obtain the request protocol.
int	**getReadTimeout**() Obtain the read timeout.
String	**getSoapAction**() Obtain the operation request address.
String	**getUrl**() Obtain the request address.
boolean	**isRedirects**() Allow redirection or not.
void	**setBodyPrefix**(String bodyPrefix) Set the node prefix of a request packet.
void	**setCharset**(String charset) Set the HTTP request encoding format.
void	**setConnectTimeout**(int connectTimeout) Set the connection timeout interval.
void	**setContent**(Object content) Set the request content.
void	**setContentType**(String contentType) Set the packet parameter type.
void	**setEnvelopePrefix**(String envelopePrefix) Set the envelope prefix.
void	**setHeader**(String name, String value) Set request header information.
void	**setHeaders**(Map<String,String> headers) Set request header information.
void	**setMethod**(String method) Set the request method.
void	**setNamespace**(String namespace) Set the namespace.
void	**setNamespacePrefix**(String namespacePrefix) Set the namespace prefix.
void	**setOperation**(String operation) Set the operation name.

Returned Type	Method and Description
void	**setParameter**(String name, String value) Set a SOAP request parameter.
void	**setParameters**(Map<String,String> parameters) Set the SOAP request parameters.
void	**setProtocol**(String protocol) Set the request protocol.
void	**setReadTimeout**(int readTimeout) Set the read timeout.
void	**setRedirects**(boolean redirects) Set whether to redirect.
void	**setSoapAction**(String soapAction) Set the operation request address.
void	**setUrl**(String url) Set the request address.

Method Details

- **public String buildSoapMessage()**

  Construct a SOAP request packet.

  Returns

  SOAP request packet.

- **public String getBodyPrefix()**

  Obtain the node prefix of a request packet.

  Returns

  Node prefix of a request packet.

- **public String getCharset()**

  Obtain the HTTP request encoding format.

  Returns

  HTTP request encoding format.

- **public int getConnectTimeout()**

  Obtain the connection timeout interval.

  Returns

  Connection timeout.

- **public Object getContent()**

  Obtain the request content.

  Returns

  Request content.

- **public String getContentType()**

  Obtain the packet parameter type.

  Returns

  Packet parameter type.

- **public String getEnvelopePrefix()**

  Obtain the envelope prefix.

  Returns

  Envelope prefix.

- **public String getHeader(String name)**

  Obtain the request header value based on the request header name.

  Input Parameter

  name indicates the request header name.

  Returns

  Request header value corresponding to the request header name

- **public Map<String,String> getHeaders()**

  Obtain request header information.

  Returns

  Request header information.

- **public String getMethod()**

  Obtain the request method.

  Returns

  Request method.

- **public String getNamespace()**

  Obtain the namespace.

  Returns

  Namespace.

- **public String getNamespacePrefix()**

  Obtain the namespace prefix.

  Returns

  Namespace prefix.

- **public String getOperation()**

  Obtain the operation name.

  Returns

  Operation name.

- **public String getParameter(String name)**

  Obtain SOAP request parameters based on the specified name.

  Input Parameter

  name indicates the name of a SOAP request parameter.

  Returns

  SOAP request parameter.

- **public Map<String,String> getParameters()**

  Obtain the SOAP request parameters.

  Returns

  SOAP request parameters.

- **public String getProtocol()**

  Obtain the request protocol.

  Returns

  Request protocol.

- **public int getReadTimeout()**

  Obtain the read timeout.

  Returns

  Read timeout.

- **public String getSoapAction()**

  Obtain the operation request address.

  Returns

  Operation request address.

- **public String getUrl()**

  Obtain the request address.

  Returns

  Request address.

- **public boolean isRedirects()**

  Allow redirection or not.

  Returns

  true or false

- **public void setBodyPrefix(String bodyPrefix)**

  Set the node prefix of a request packet.

  Input Parameter

  bodyPrefix indicates the node prefix of a request packet.

- **public void setCharset(String charset)**

  Set the HTTP request encoding format.

  Input Parameter

  charset indicates the encoding format of the HTTP request.

- **public void setConnectTimeout(int connectTimeout)**

  Set the connection timeout interval.

  Input Parameter

  Connection timeout indicates the connection timeout interval.

- **public void setContent(Object content)**

  Set the request content.

  Input Parameter

  content indicates the request content.

- **public void setContentType(String contentType)**

  Set the packet parameter type.

  Input Parameter

  contentType indicates the packet parameter type.

- **public void setEnvelopePrefix(String envelopePrefix)**

  Set the envelope prefix.

  Input Parameter

  envelopePrefix indicates the envelope prefix.

- **public void setHeader(String name, String value)**

  Set request header information.

  Input Parameter

  - **name** indicates the request header name.
  - **value** indicates the request header value.

- **public void setHeaders(Map<String,String> headers)**

  Set request header information.

  Input Parameter

  headers indicates the request header information.

- **public void setMethod(String method)**

  Set the request method.

  Input Parameter

  method indicates a request method.

- **public void setNamespace(String namespace)**

  Set the namespace.

  Input Parameter

  namespace indicates the namespace.

- **public void setNamespacePrefix(String namespacePrefix)**

  Set the namespace prefix.

  Input Parameter

  namespacePrefix indicates the namespace prefix.

- **public void setOperation(String operation)**

  Set the operation name.

  Input Parameter

  operation indicates the operation name.

- **public void setParameter(String name, String value)**

  Set the SOAP request parameters.

  Input Parameter

  - **name** indicates the name of a SOAP request parameter.
  - **value** indicates the value of a SOAP request parameter.

- **public void setParameters(Map<String,String> parameters)**

  Set the SOAP request parameters.

  Input Parameter

parameters indicates the SOAP request parameters.

- **public void setProtocol(String protocol)**

  Set the request protocol.

  Input Parameter

  protocol indicates the request protocol.

- **public void setReadTimeout(int readTimeout)**

  Set the read timeout.

  Input Parameter

  readTimeout indicates the read timeout interval.

- **public void setRedirects(boolean redirects)**

  Set whether to redirect.

  Input Parameter

  redirects indicates whether to redirect.

- **public void setSoapAction(String soapAction)**

  Set the operation request address.

  Input Parameter

  soapAction indicates the operation request address.

- **public void setUrl(String url)**

  Set the request address.

  Input Parameter

  url indicates the request URL.

2.4.33 StringUtils

Path

com.roma.apic.livedata.common.v1.StringUtils

Description

This class is used to convert character strings.

Example

```
importClass(com.roma.apic.livedata.common.v1.StringUtils);
function execute(data){
    return StringUtils.toString([97,96,95,94,93,92], "UTF-8")
}
```

Method List

Returned Type	Method and Description
static String	**toString**(byte[] bytes, String encoding) Convert a byte array into a string.

Returned Type	Method and Description
static String	**toString**(byte[] bytes) Convert a byte array into a UTF-8 encoded string.
static String	**toHexString**(byte[] data) Convert a byte array into a hexadecimal lowercase string.
static byte[]	**hexToByteArray**(String hex) Convert a hexadecimal string into a byte array.

Method Details

- **public static String toString(byte[] bytes, String encoding)**

  Converts a byte array into a string.

  Input Parameter

  - **bytes** indicates the byte array to be converted.
  - **encoding** indicates encoding.

  Returns

  String after conversion.

- **public static String toString(byte[] bytes)**

  Converts a byte array into a UTF-8 encoded string.

  Input Parameter

  bytes indicates the byte array to be converted.

  Returns

  String after conversion.

- **public static String toHexString(byte[] data)**

  Converts a byte array into a hexadecimal lowercase string.

  Input Parameter

  data indicates the byte array to be converted.

  Returns

  Hexadecimal character string after conversion.

- **public static byte[] hexToByteArray(String hex)**

  Converts a hexadecimal string into a byte array.

  Input Parameter

  hex indicates the hexadecimal character string to be converted.

  Returns

  Byte array after conversion.

2.4.34 TextUtils

Path

com.roma.apic.livedata.common.v1.TextUtils

Description

This class is used to provide the formatting function.

Method List

Returned Type	Method and Description
static Map<String,String >	encodeByUrlEncoder(Map<String,String> map) Encode the key and value in the map.
static boolean	parseBoolean(String value, boolean defaultValue) Convert a character string into a Boolean type.
static String	toHttpParameters(Map<String,String> map) Convert the map content to parameters in HTTP URL.

Method Details

- **public static Map<String,String> encodeByUrlEncoder(Map<String,String> map)**

  Encode the key and value in the map.

  Input Parameter

  map indicates the map containing URL parameters.

  Returns

  Map after the URL encoding.

- **public static boolean parseBoolean(String value, boolean defaultValue)**

  Convert a character string into a Boolean type.

  Input Parameter

  - **value** indicates the character content to be converted.
  - **defaultValue** indicates the default Boolean value. It is used when **value** is invalid.

  Returns

  Boolean value.

- **public static String toHttpParameters(Map<String,String> map)**

  Convert the content in the map to the parameters in the HTTP URL.

  Input Parameter

  map indicates the map containing URL parameters.

  Returns

  Parameters in the HTTP URL.

2.4.35 XmlUtils

Path

com.roma.apic.livedata.common.v1.XmlUtils

Description

This class is used to provide the XML conversion function.

Example

```
importClass(com.roma.apic.livedata.common.v1.XmlUtils);
function execute(data) {
    var xml = '<a><id>2</id><name>1</name></a>'
    return XmlUtils.toMap(xml)
}
```

Method List

Returned Type	Method and Description
static String	**toJson**(String xml) Convert a character string in the XML format into a JSON file.
static Map<String,Object>	**toMap**(String xml) Convert XML into Map.
static String	**toXml**(Object object) Convert an object into an XML file.
static String	**toXml**(Object object, Map<String,Object> config) Convert an object into an XML file.

Method Details

- **public static String toJson(String xml)**

  Convert a character string in the XML format into a JSON file.

  Input Parameter

  xml indicates the character string in XML format.

  Returns

  Character string in JSON format.

- **public static Map<String,Object> toMap(String xml)**

  Convert XML into Map.

  Input Parameter

  xml indicates the character string in XML format.

  Returns

Character string in MAP format.

- **public static String toXml(Object object)**

  Convert an object into an XML file.

  Input Parameter

  object indicates the object to be converted.

  Returns

  Character string in XML format.

- **public static String toXml(Object object, Map<String,Object> config)**

  Convert an object into XML.

  Input Parameter

  - **object** indicates the object to be converted.
  - **config** indicates the conversion configuration.

  Returns

  Character string in XML format.

2.5 Developing Custom Data Backends

2.5.1 SQL Syntax

SQL Syntax Differences Between Data Backends and Databases

- To transfer parameters carried in a backend request to an SQL statement, use *${parameter name}* to mark the parameters. Parameters of the String type must be enclosed in single quotation marks, whereas parameters of the int type do not need to be enclosed.

  In the following example, *name* is a parameter of the String type and *id* is a parameter of the int type.

  ```
  select * from table01 where name='${name}' and id=${id}
  ```

- Parameters can be transferred in the headers, parameters, or body of backend requests.
- If the character string in an SQL statement contains keywords, you must escape the character string.

  For example, if a field name is **delete**, the SQL statement must be written in the following format:

  ```
  select `delete` from table01
  ```

- If **Precompiling** is selected during data backend configuration, input parameters are used for fuzzy match query, and the match field contains %, use the CONCAT function for concatenation.

  In the following example, *name* is a string.

  ```
  select * from table01 where name like concat('%',${name})
  ```

NOTE

If precompiling has been enabled and an SQL statement references backend request parameters of multiple data types, the input parameters will be converted to String by default. Therefore, when the SQL statement is executed, the corresponding function needs to be called to convert non-String parameters.

For example, if both the **name** (String type) and **id** (int type) parameters are transferred to an SQL statement, the **id** parameter will be converted to the String type. Therefore, you need to use a conversion function to convert the **id** parameter back to the int type in the SQL statement. The following uses the cast() function as an example. The conversion function varies depending on the database type in use.

```
select * from table01 where name='${name}' and id=cast('${id}' as int)
```

SQL Query Examples (Similar to UPDATE and INSERT)

- Query with parameters specified

  Transfer parameters (Headers, Parameters, or Body) carried in backend requests to SQL statements to provide flexible conditional query or data processing capabilities for the SQL statements.

  – For APIs using the GET or DELETE method, obtain parameters from the request URL.

  – For APIs using the POST or PUT method, obtain parameters from the request body. Note: The body is in application/x-www-form-urlencoded format.

  ```
  select * from table01 where 1=1 and col01 = ${param01};
  ```

- Query with optional parameters

  ```
  select * from table01 where 1=1 [and col01 = ${param01}] [and col02 = ${param02}]
  ```

- IN query

  ```
  select * from table01 where 1=1 and col01 in ('${param01}','${param02}');
  ```

- UNION query

  By default, duplicate data will be deleted. To return all data, use the keywords **union all**.

  ```
  select * from table01
  union [all | distinct]
  select * from table02;
  ```

- Nested query

  ```
  select * from table01 where 1=1 and col01 in (select col02 from table02 where col03 is not null);
  ```

Native Commands Compatible with NoSQL (such as MongoDB and Redis)

- Command formats supported by the Redis data source:

  GET, HGET, HGETALL, LRANGE, SMEMBERS, ZRANGE, ZREVRANGE, SET, LPUSH, SADD, ZADD, HMSET, DEL

- Command formats supported by the MongoDB data source:

  find

NoSQL Examples

- Insert a key of the String type. The value is obtained from the request parameter.

  ```
  set hello ${parm01}
  ```

- Query the key of the String type.

  ```
  get hello
  ```

2.5.2 Calling a Stored Procedure

Currently, a data API cannot create stored procedures, but can execute stored procedures of the MySQL, Oracle, and PostgreSQL data sources. The Oracle database is used as an example.

Data Source Description

Assume that the database contains a table. The table structure is as follows:

```
create table sp_test(id number,name varchar2(50),sal number);
```

Insert data into the table. The following table shows the data set.

Table 2-2 sp_test table data set

ID	NAME	SAL
1	ZHANG	5000
2	LI	6000
3	ZHAO	7000
4	WANG	8000

The Oracle database contains a stored procedure for querying the value of **sal** based on **name**.

```
create or replace procedure APICTEST.sb_test(nname in varchar, nsal out number) as
begin
   select sal into nsal from sp_test where name = nname;
end;
```

Statements in a Data API

When a data API calls a stored procedure, parameters can be transferred through Headers, Parameters, or Body of a backend request. The syntax of a parameter name is as follows: *{Parameter name}.{Data type}.{Transmission type}*.

- The data type can be **String** or **int**.
- *Transmission type* indicates whether the parameter is an input parameter or output parameter. **in** indicates an input parameter, and **out** indicates an output parameter.

The following script is an example statement used for calling the stored procedure in the data API:

```
call sb_test(${nname.String.in},${nsal.int.out})
```

In the example script, **nname** is an input parameter of the String type and the parameter name is **nname.String.in**. The value is the parameter to be queried. **nsal** is an output parameter of the numeric type and the parameter name is **nsal.int.out**. Due to the format restriction, the value of the output parameter must be set. You can set it to any value that meets the data type requirements, which does not affect the output result.

NOTE

- The data API uses String and int to distinguish character strings and values when calling a stored procedure. Single quotation marks are not required. This is different from the SQL requirement.
- The parameter names defined in Headers, Body, or Parameters of a backend request must be different. Otherwise, they will be overwritten.

- The following script is an example of transferring parameters in Body:

  Body of the backend request:

  ```
  {
    "nname.String.in": "zhang",
    "nsal": 0
  }
  ```

  Response result:

  ```
  {
    "test": [
      5000
    ]
  }
  ```

- The following script is an example of transferring parameters in Parameters:

  Parameters of the backend request:

  ```
  https://example.com?nname.String.in=zhang&nsal=0
  ```

  Response result:

  ```
  {
    "test": [
      5000
    ]
  }
  ```

2.5.3 Orchestrating Data Sources

A data API can contain multiple data sources. Therefore, an API request can call multiple data sources. For example, the query result of the first data source can be used as the parameter of the second data source.

MySQL is used as an example. Assume that the data API contains data source 1 and data source 2, **user01** is the data table of data source 1, and **user02** is the data table of data source 2. The structures of the two tables are as follows:

Table 2-3 Table structures

Data Source	Table Name	Parameter
Data source 1	user01	● id (int) ● name (varchar)
Data source 2	user02	● user_id (int) ● user_age (int) ● user_sex (varchar)

The data source SQL statement is designed as follows:

For data source 1, query the ID of the data record whose name is **zhang** in table **user01**. Assume that the return object from data source 1 is **default1**.

```
select id from user01 where name='zhang';
```

For data source 2, go to table **user02** and find the data record of **user_age** corresponding to the ID found in table **user01**. Assume that the return object from data source 2 is **default2**.

```
select user_age from user02 where user_id=${default1[0].id};
```

$*{default1[0].id}* indicates the query result from data source 1. (**default1** indicates the return object from data source 1, and *id* indicates the query field of data source 1.)

Assume that the data tables user01 and user02 contain the following data records:

user01		user02		
id	**name**	**user_id**	**user_age**	**user_sex**
1	zhang	2	17	Female
2	li	3	18	Male
3	wang	1	18	Male

The following response is returned when the data API is called:

```
{
    "default1":[{
        "id":1
    }],
    "default2":[{
        "user_age":18
    }]
}
```

2.5.4 Using Optional Parameters

In a data API, the square brackets ([]) are used to mark optional parameters. An example SQL statement is as follows:

```
select * from table01 where id=${id} [or sex='${sex}']
```

The statement enclosed in square brackets ([]) indicates that the parameter takes effect only when the backend request carries the ${sex} parameter. If ${sex} is not carried, the statement enclosed in [] is ignored during execution.

- If the backend request carries the id=88 parameter but does not carry the optional parameter sex, run the following SQL statement:
  ```
  select * from table01 where id=88;
  ```
- If the backend request carries both id=88 and sex=female, run the following SQL statement:
  ```
  select * from table01 where id=88 or sex='female';
  ```

2.6 Developing Signature Verification for Backend Services

2.6.1 Preparations

Obtaining Signature Key Information

Old version: On the ROMA Connect console, choose **API Connect** > **API Management**. On the **Signature Keys** tab page, click the name of the signature key bound to the API. On the details page that is displayed, obtain the signature key and secret.

New version: On the ROMA Connect console, choose **API Connect** > **API Policies**. On the **Policies** tab page, filter policies by signature key, and click the name of the signature key bound to the API. On the details page that is displayed, obtain the signature key and secret.

Preparing the Development Environment

- Installing a development tool

  Select a proper development tool based on the language used.

  – Download the installation package of IntelliJ IDEA 2018.3.5 or later from the **official IntelliJ IDEA website**.

  – Download the Visual Studio 2019 installation package of 16.8.4 or later from the **official Visual Studio page**.

- Installing a development language

  – Java: Download the JDK of 1.8.111 or later from the **official Oracle website**.

  – Python: Download the Python 2.7 or 3.X installation package from the **official Python download page**.

2.6.2 Java

Scenarios

To use Java to sign backend requests, obtain the Java SDK, import the project, and verify the backend signature by referring to the example provided in this section.

NOTE

The Java SDK supports only basic and HMAC backend service signatures.

Prerequisites

- A signature key has been created on the console and bound to an API. For details, see **Configuring Signature Verification for Backend Services**.
- You have obtained the signature key and secret. For details, see **Preparations**.

- You have installed the development tool and Java development environment. For details, see **Preparations**.

Obtaining the SDK

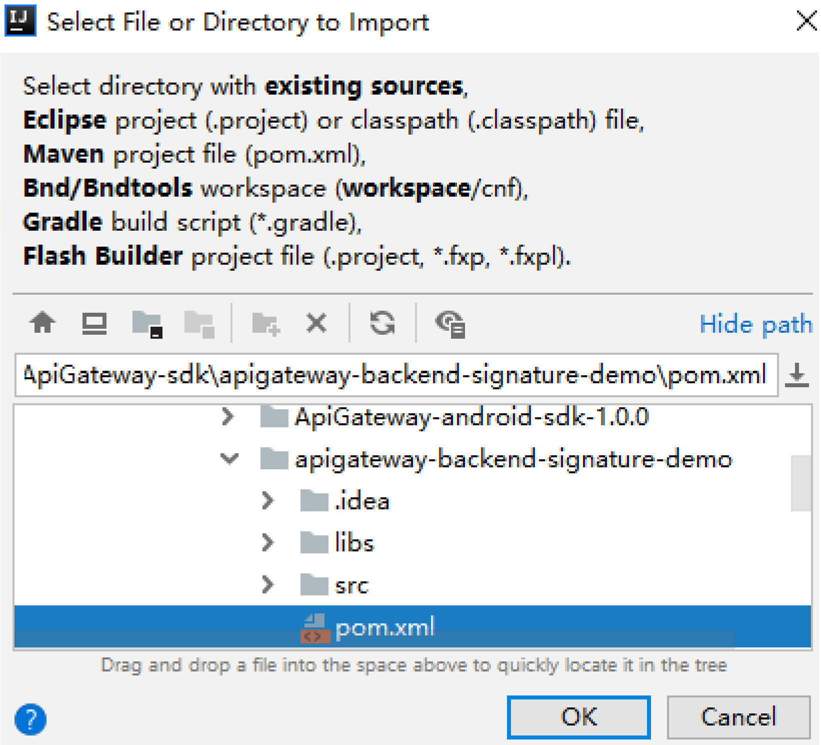
Old version: Log in to the ROMA Connect console, choose **API Connect** > **API Management** > **Signature Keys**, and download the SDK.

New version: Log in to the ROMA Connect console, choose **API Connect** > **Credentials** > **SDKs**, and download the SDK.

Importing a Project

1. Open IntelliJ IDEA, choose **File** > **New** > **Project from Existing Sources**, select the **apigateway-backend-signature-demo\pom.xml** file, and click **OK**.

Figure 2-41 Select File or Directory to Import



Select File or Directory to Import

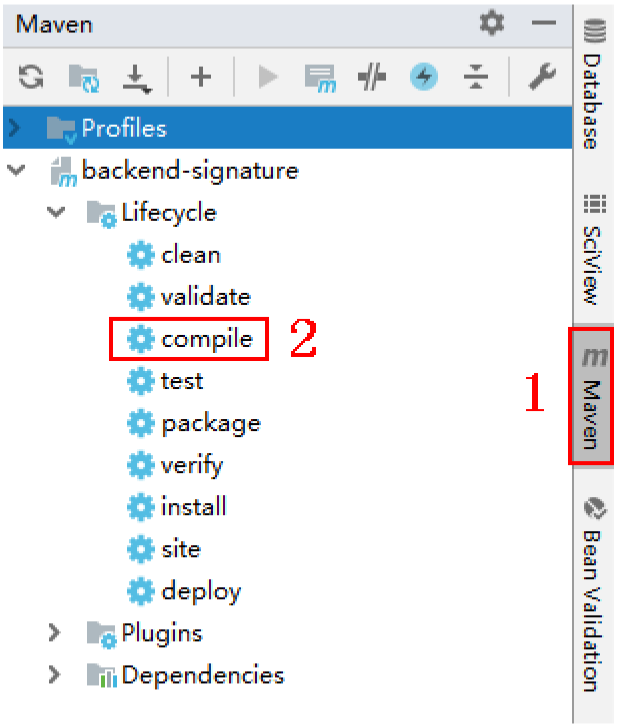
Select directory with **existing sources**,
Eclipse project (.project) or classpath (.classpath) file,
Maven project file (pom.xml),
Bnd/Bndtools workspace (**workspace**/cnf),
Gradle build script (*.gradle),
Flash Builder project file (.project, *.fxp, *.fxpl).

ApiGateway-sdk\apigateway-backend-signature-demo\pom.xml

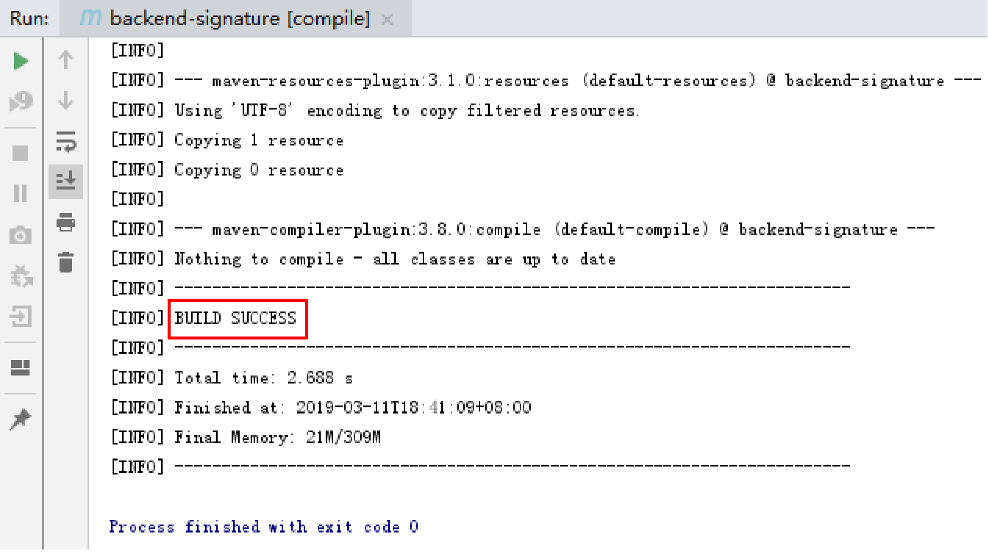
Hide path
ApiGateway-android-sdk-1.0.0
apigateway-backend-signature-demo
.idea
libs
src
pom.xml
Drag and drop a file into the space above to quickly locate it in the tree
OK
Cancel

2. Retain the default settings, click **Next** for the following four steps, and then click **Finish**.
3. On the **Maven** tab page on the right, double-click **compile** to compile the file.

Figure 2-42 Compiling the project



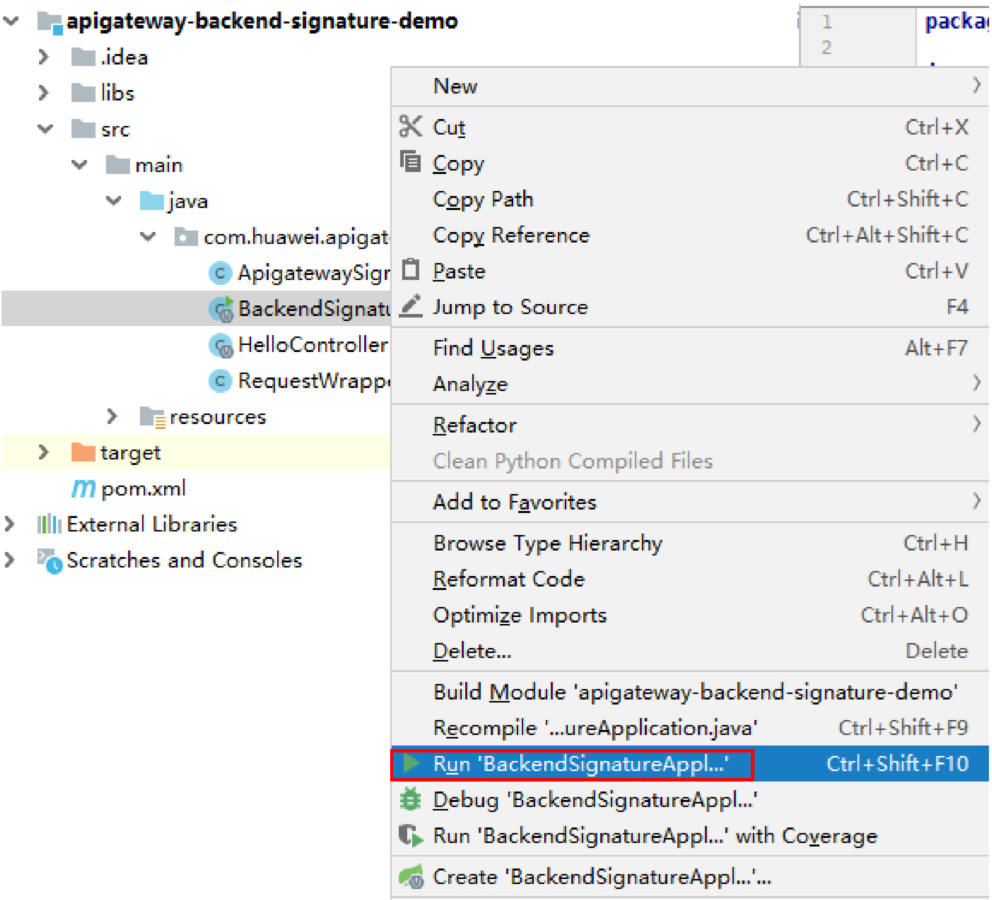
If the message "BUILD SUCCESS" is displayed, the compilation is successful.



```
[INFO]
[INFO] --- maven-resources-plugin:3.1.0:resources (default-resources) @ backend-signature ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:compile (default-compile) @ backend-signature ---
[INFO] Nothing to compile - all classes are up to date
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2.688 s
[INFO] Finished at: 2019-03-11T18:41:09+08:00
[INFO] Final Memory: 21M/309M
[INFO] ------------------------------------------------------------------------

Process finished with exit code 0
```

4. Right-click **BackendSignatureApplication** and choose **Run**.

Figure 2-43 Running the BackendSignatureApplication service


apigateway-backend-signature-demo
.idea
libs
src
main
java
com.huawei.apigat
ApigatewaySigr
BackendSignatu
HelloController
RequestWrappe
resources
target
pom.xml
External Libraries
Scratches and Consoles
New
Cut Ctrl+X
Copy Ctrl+C
Copy Path Ctrl+Shift+C
Copy Reference Ctrl+Alt+Shift+C
Paste Ctrl+V
Jump to Source F4
Find Usages Alt+F7
Analyze
Refactor
Clean Python Compiled Files
Add to Favorites
Browse Type Hierarchy Ctrl+H
Reformat Code Ctrl+Alt+L
Optimize Imports Ctrl+Alt+O
Delete... Delete
Build Module 'apigateway-backend-signature-demo'
Recompile '...ureApplication.java' Ctrl+Shift+F9
Run 'BackendSignatureAppl...' Ctrl+Shift+F10
Debug 'BackendSignatureAppl...'
Run 'BackendSignatureAppl...' with Coverage
Create 'BackendSignatureAppl...'...

Modify the parameters in sample code **ApigatewaySignatureFilter.java** as required. For details about the sample code, see **Example of Verifying the Backend Signature of hmac Type**.

Example of Verifying the Backend Signature of hmac Type

NOTE

- This example demonstrates how to write a Spring boot–based server as the backend of an API and implement a filter to verify the signature of requests sent from APIC.
- Signature information is added to requests sent to access the backend of an API after a signature key of hmac type is bound to the API.

1. Compile a controller in the **/hmac** directory.

```
// HelloController.java

@RestController
@EnableAutoConfiguration
public class HelloController {

    @RequestMapping("/hmac")
    private String hmac() {
        return "Hmac authorization success";
    }
}
```

2. Compile a filter that matches all request paths and methods, and put the signature key and secret in a **Map**.

```
public class ApigatewaySignatureFilter implements Filter {
    private static Map<String, String> secrets = new HashMap<>();
    static {
        // Directly writing AK/SK in code is risky. For security, encrypt your AK/SK and store them in the
configuration file or environment variables.
        // In this example, the AK/SK are stored in environment variables for identity authentication.
Before running this example, set environment variables HUAWEICLOUD_SDK_AK1,
HUAWEICLOUD_SDK_SK1, and HUAWEICLOUD_SDK_AK2, HUAWEICLOUD_SDK_SK2.
        secrets.put(System.getenv("HUAWEICLOUD_SDK_AK1"),
System.getenv("HUAWEICLOUD_SDK_SK1"));
        secrets.put(System.getenv("HUAWEICLOUD_SDK_AK2"),
System.getenv("HUAWEICLOUD_SDK_SK2"));
    }

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain
chain) {
       //Signature verification code
        ...
    }
}
```

3. The doFilter function is the signature verification code. To ensure that the body can be read in the filter and controller, wrap the request and send it to the filter and controller. For the implementation of wrapper classes, see RequestWrapper.java.

```
RequestWrapper request = new RequestWrapper((HttpServletRequest) servletRequest);
```

4. Use a regular expression to parse the **Authorization** header to obtain **signingKey** and **signedHeaders**.

```
private static final Pattern authorizationPattern = Pattern.compile("SDK-HMAC-SHA256\\s
+Access=([^,]+),\\s?SignedHeaders=([^,]+),\\s?Signature=(\\w+)");

...

String authorization = request.getHeader("Authorization");
if (authorization == null || authorization.length() == 0) {
    response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Authorization not found.");
    return;
}

Matcher m = authorizationPattern.matcher(authorization);
if (!m.find()) {
    response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Authorization format incorrect.");
    return;
}
String signingKey = m.group(1);
String signingSecret = secrets.get(signingKey);
if (signingSecret == null) {
    response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Signing key not found.");
    return;
}
String[] signedHeaders = m.group(2).split(";");
```

For example, for **Authorization** header:

```
SDK-HMAC-SHA256 Access=signature_key1, SignedHeaders=host;x-sdk-date,
Signature=e11adf65a20d1b82c25419b5********8d0ba12fed1ceb13ed00
```

The parsing result is as follows:

```
signingKey=signature_key1
signedHeaders=host;x-sdk-date
```

5. Find **signingSecret** based on **signingKey**. If **signingKey** does not exist, the authentication failed.

```
String signingSecret = secrets.get(signingKey);
if (signingSecret == null) {
    response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Signing key not found.");
    return;
}
```

6. Create a request, and add the method, URL, query, and signedHeaders headers to the request. Determine whether the body needs to be set.

   The body is read if there is no **x-sdk-content-sha256** header with value **UNSIGNED-PAYLOAD**.

```
Request apiRequest = new DefaultRequest();
apiRequest.setHttpMethod(HttpMethodName.valueOf(request.getMethod()));
String url = request.getRequestURL().toString();
String queryString = request.getQueryString();
try {
    apiRequest.setEndpoint((new URL(url)).toURI());
    Map<String, String> parametersmap = new HashMap<>();
    if (null != queryString && !"".equals(queryString)) {
        String[] parameterarray = queryString.split("&");
        for (String p : parameterarray) {
            String[] p_split = p.split("=", 2);
            String key = p_split[0];
            String value = "";
            if (p_split.length >= 2) {
                value = p_split[1];
            }
            parametersmap.put(URLDecoder.decode(key, "UTF-8"), URLDecoder.decode(value, "UTF-8"));
        }
        apiRequest.setParameters(parametersmap); //set query
    }
} catch (URISyntaxException e) {
    e.printStackTrace();
}

boolean needbody = true;
String dateHeader = null;
for (int i = 0; i < signedHeaders.length; i++) {
    String headerValue = request.getHeader(signedHeaders[i]);
    if (headerValue == null || headerValue.length() == 0) {
        ((HttpServletResponse) response).sendError(HttpServletResponse.SC_UNAUTHORIZED, "signed
header" + signedHeaders[i] + " not found.");
    } else {
        apiRequest.addHeader(signedHeaders[i], headerValue);//set header
        if (signedHeaders[i].toLowerCase().equals("x-sdk-content-sha256") &&
headerValue.equals("UNSIGNED-PAYLOAD")) {
            needbody = false;
        }
        if (signedHeaders[i].toLowerCase().equals("x-sdk-date")) {
            dateHeader = headerValue;
        }
    }
}

if (needbody) {
        apiRequest.setContent(new ByteArrayInputStream(request.getBody()));    //set body
}
```

7. Check whether the signature has expired. Obtain the time from the **X-Sdk-Date** header, and check whether the difference between this time and the server time is within 15 minutes. If **signedHeaders** does not contain **X-Sdk-Date**, the authentication failed.

```
private static final DateTimeFormatter timeFormatter =
DateTimeFormat.forPattern("yyyyMMdd'T'HHmmss'Z'").withZoneUTC();

...

if (dateHeader == null) {
    response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Header x-sdk-date not found.");
    return;
}
long date = timeFormatter.parseMillis(dateHeader);
long duration = Math.abs(DateTime.now().getMillis() - date);
if (duration > 15 * 60 * 1000) {
    response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Signature expired.");
    return;
}
```

8. Add the **Authorization** header to the request, and invoke the **verify** method to verify the request signature. If the verification is successful, the next filter is executed. Otherwise, the authentication failed.

```
DefaultSigner signer = (DefaultSigner) SignerFactory.getSigner();
boolean verify = signer.verify(apiRequest, new BasicCredentials(signingKey, signingSecret));
if (verify) {
    chain.doFilter(request, response);
} else {
    response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "verify authroization failed.");
}
```

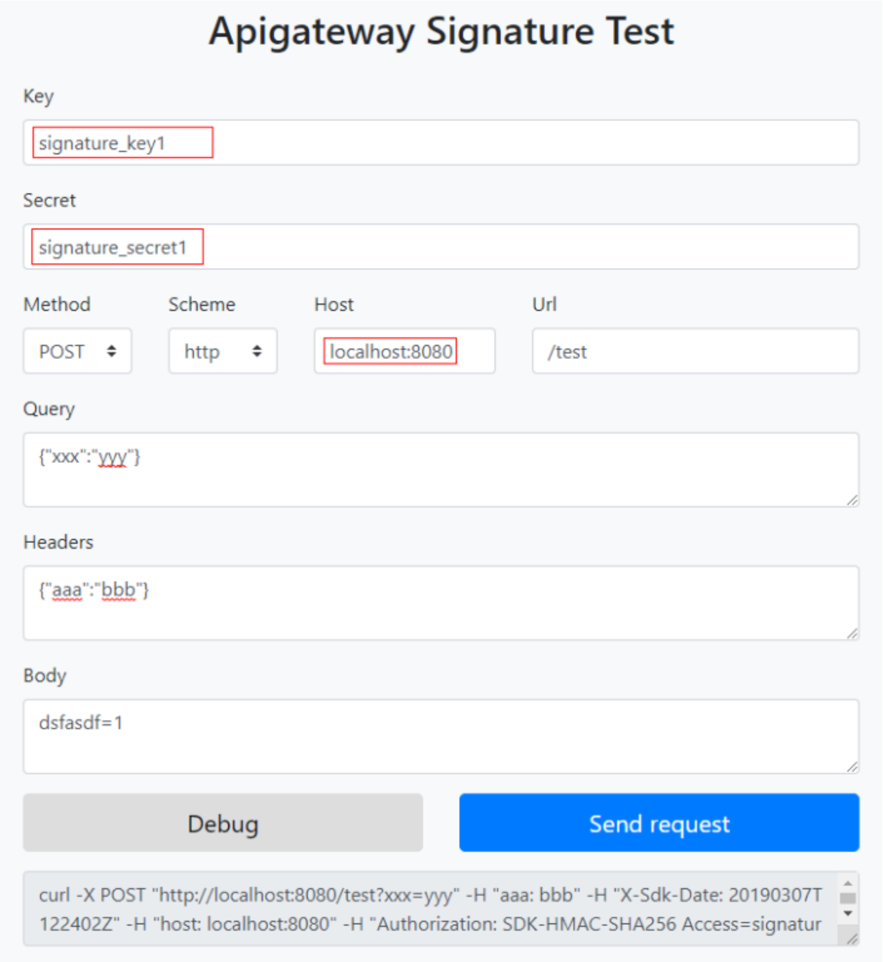
9. Register the mapping between filters and paths.

```
@Configuration
public class FilterConfig {
    @Bean
    public FilterRegistrationBean registApigatewaySignatureFilter() {
        FilterRegistrationBean registration = new FilterRegistrationBean();
        registration.setFilter(new ApigatewaySignatureFilter());
        registration.addUrlPatterns("/hmac");
        registration.setName("ApigatewaySignatureFilter");
        return registration;
    }
}
```

10. Run the server to verify the code. The following example uses the HTML signature tool in the JavaScript SDK to generate a signature.

    Set the parameters according to the following figure, and click **Send request**. Copy the generated curl command, execute it in the CLI, and check whether the server returns **Hello World!**

    If an incorrect key or secret is used, the server returns **401**, which means authentication failure.



Apigateway Signature Test

Key

signature_key1

Secret

signature_secret1

Method	Scheme	Host	Url
POST	http	localhost:8080	/test

Query

{"xxx":"yyy"}

Headers

{"aaa":"bbb"}

Body

dsfasdf=1

```
curl -X POST "http://localhost:8080/test?xxx=yyy" -H "aaa: bbb" -H "X-Sdk-Date: 20190307T
122402Z" -H "host: localhost:8080" -H "Authorization: SDK-HMAC-SHA256 Access=signatur
```

Example of Verifying the Backend Signature of basic Type

NOTE

- This example demonstrates how to write a Spring boot–based server as the backend of an API and implement a filter to verify the signature of requests sent from APIC.
- Basic authentication information is added to requests sent to the backend of an API after a basic signature key is bound to an API. The username for basic authentication is the key of the signature key, and the password is the secret of the signature key.

1. Compile a controller in the **/basic** directory.

```
// HelloController.java

@RestController
@EnableAutoConfiguration
public class HelloController {

    @RequestMapping("/basic")
    private String basic() {
        return "Basic authorization success";
    }
}
```

2. Compile a filter. According to the basic authentication rule, the Authorization header is in the format of "Basic "+base64encode(username+":"+password). The following is the verification code compiled according to the rule:

```
// BasicAuthFilter.java
public class BasicAuthFilter implements Filter {
    private static final String CREDENTIALS_PREFIX = "Basic ";
    private static Map<String, String> secrets = new HashMap<>();

    static {
        secrets.put("signature_key1", "signature_secret1");
        secrets.put("signature_key2", "signature_secret2");
    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain 
chain) {
        HttpServletRequest request = (HttpServletRequest) servletRequest;
        HttpServletResponse response = (HttpServletResponse) servletResponse;
        try {
            String credentials = request.getHeader("Authorization");
            if (credentials == null || credentials.length() == 0) {
                response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Authorization not found.");
                return;
            }

            if (!credentials.startsWith(CREDENTIALS_PREFIX)) {
                response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Authorization format 
incorrect.");
                return;
            }
            String authInfo = credentials.substring(CREDENTIALS_PREFIX.length());
            String decoded;
            try {
                decoded = new String(Base64.getDecoder().decode(authInfo));
            } catch (IllegalArgumentException e) {
                response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Authorization format 
incorrect.");
                return;
            }
            String[] spl = decoded.split(":", 2);
            if (spl.length < 2) {
                response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Authorization format 
incorrect.");
                return;
            }
            String signingSecret = secrets.get(spl[0]);
            if (signingSecret == null) {
                response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Username not found.");
                return;
            }
            if (signingSecret.equals(spl[1])) {
                chain.doFilter(request, response);
            } else {
                response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Incorrect username or 
password");
            }
        } catch (Exception e) {
            e.printStackTrace();
            try {
                response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
            } catch (IOException e1) {
            }
        }
    }
}
```

3. Register the mapping between filters and paths.

```
@Configuration
public class FilterConfig {
    @Bean
    public FilterRegistrationBean registBasicAuthFilter() {
        FilterRegistrationBean registration = new FilterRegistrationBean();
        registration.setFilter(new BasicAuthFilter());
        registration.addUrlPatterns("/basic");
        registration.setName("BasicAuthFilter");
        return registration;
    }
}
```

4. Run the server to verify the code. Generate the Authorization header field of the basic authentication based on the username and password and send the header field to the request interface. If an incorrect username or password is used, the server returns **401**, which means authentication failure.

2.6.3 Python

Scenarios

To use Python to sign backend requests, obtain the Python SDK, import the project, and verify the backend signature by referring to the example provided in this section.

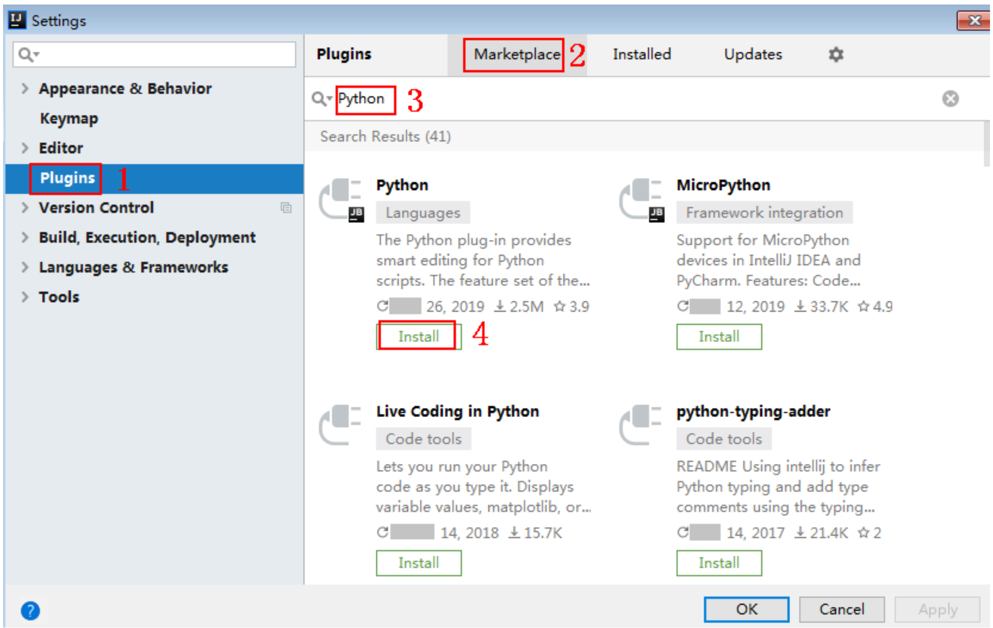
NOTE

The Python SDK supports only HMAC backend service signatures.

Prerequisites

- A signature key has been created on the console and bound to an API. For details, see **Configuring Signature Verification for Backend Services**.
- You have obtained the signature key and secret. For details, see **Preparations**.
- You have installed the development tool and Python development environment. For details, see **Preparations**.
- You have installed the Python plug-in on IntelliJ IDEA. Otherwise, install it according to **Figure 2-44**.

Figure 2-44 Installing the Python plug-in



The Python plug-in provides smart editing for Python scripts. The feature set of the...

Support for MicroPython devices in IntelliJ IDEA and PyCharm. Features: Code...

Lets you run your Python code as you type it. Displays variable values, matplotlib, or...

README Using intellij to infer Python typing and add type comments using the typing...

Obtaining the SDK

Old version: Log in to the ROMA Connect console, choose **API Connect** > **API Management** > **Signature Keys**, and download the SDK.

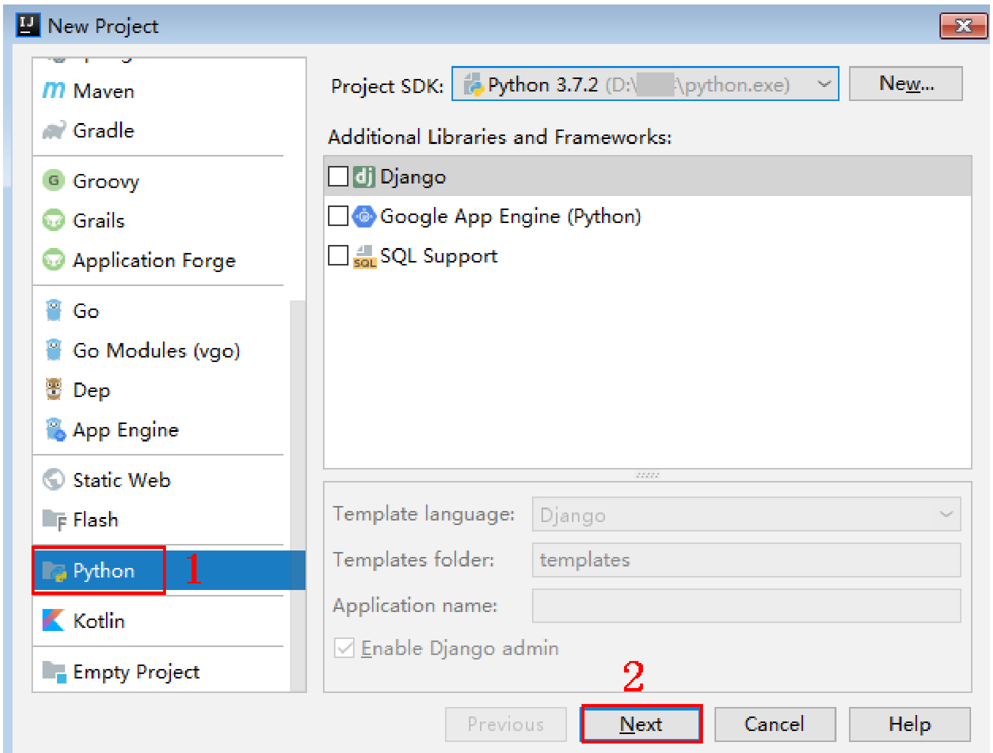
New version: Log in to the ROMA Connect console, choose **API Connect** > **Credentials** > **SDKs**, and download the SDK.

Importing a Project

1. Start IntelliJ IDEA and choose **File** > **New** > **Project**.

   On the displayed **New Project** page, choose **Python** and click **Next**.

   Figure 2-45 New Project



New Project

Project SDK: Python 3.7.2 (D:\[redacted]\python.exe)

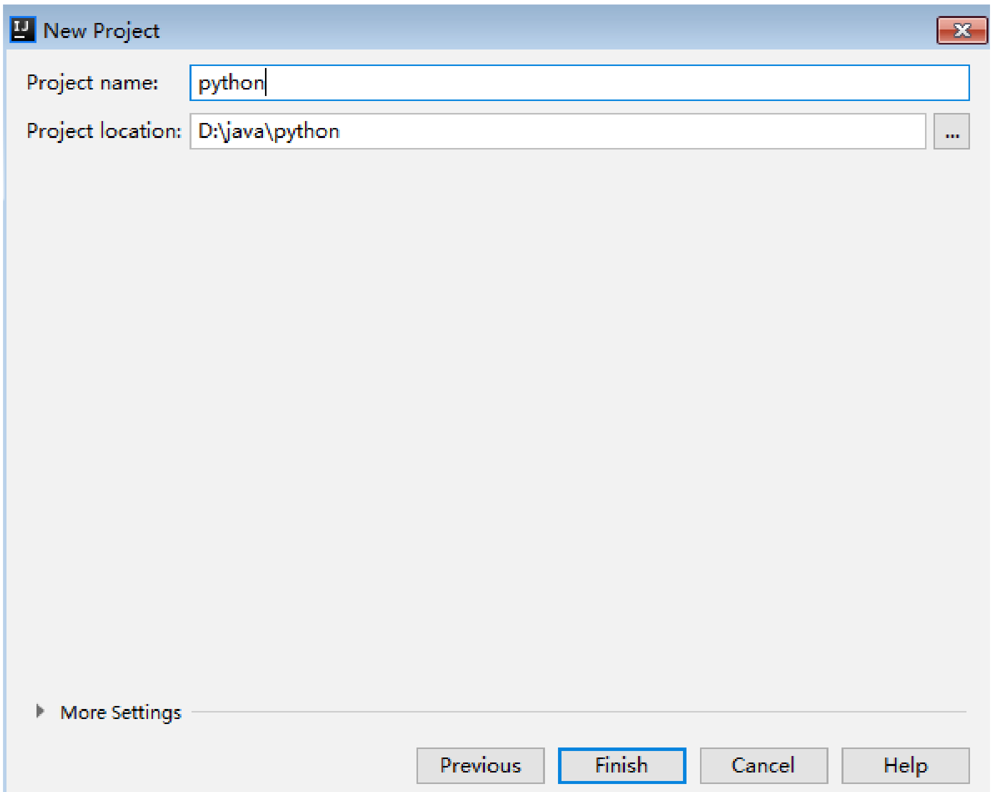
Additional Libraries and Frameworks:

☐ Django
☐ Google App Engine (Python)
☐ SQL Support

Template language: Django
Templates folder: templates
Application name:
☑ Enable Django admin

2. Click **Next**. Click **...**, select the directory where the SDK is decompressed, and click **Finish**.

Figure 2-46 Selecting the SDK directory after decompression



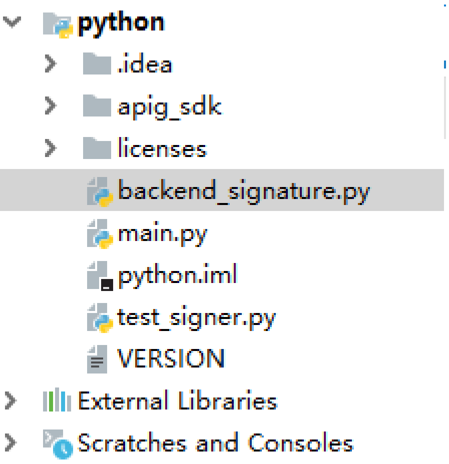
New Project

Project name: python

Project location: D:\java\python

3. View the directory structure of the project.

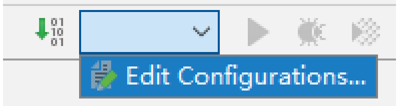
Figure 2-47 Directory structure



```
python
  .idea
  apig_sdk
  licenses
  backend_signature.py
  main.py
  python.iml
  test_signer.py
  VERSION
External Libraries
Scratches and Consoles
```

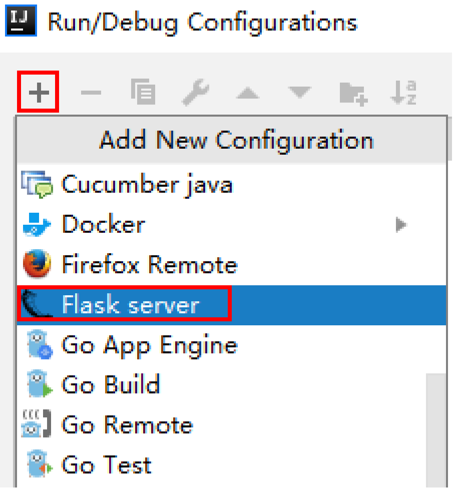
4. Click **Edit Configurations**.

Figure 2-48 Edit Configurations

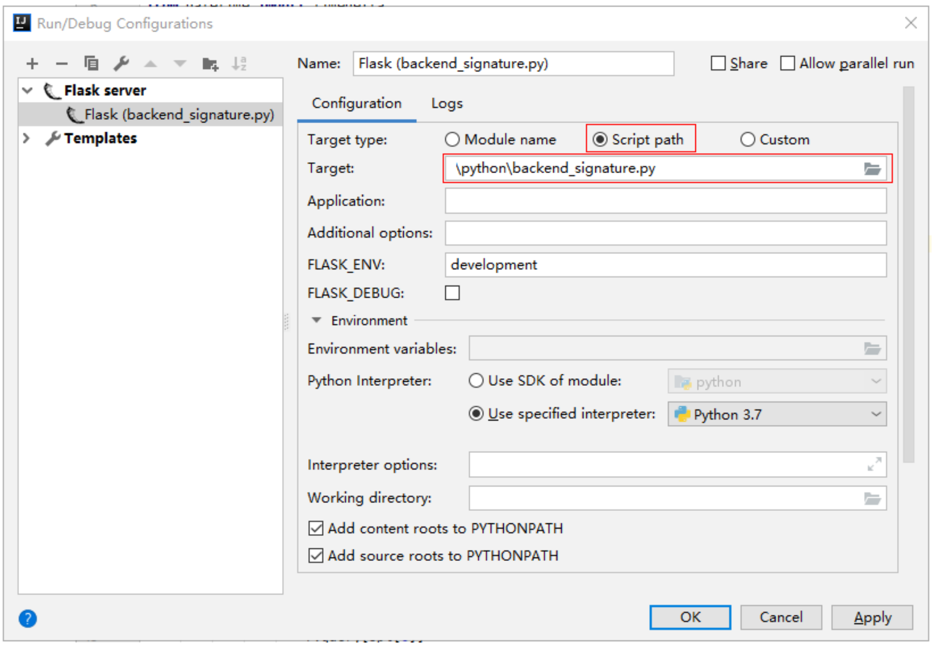


5. Click **+** and choose **Flask server**.

Figure 2-49 Choosing Flask server



6. Set **Target Type** to **Script path**, select **backend_signature.py** from the **Target** drop-down list box, and click **OK**.



Name: Flask (backend_signature.py)

□ Share □ Allow parallel run

Target type: ○ Module name ● Script path ○ Custom

Target: \python\backend_signature.py

Application:

Additional options:

FLASK_ENV: development

FLASK_DEBUG: □

Environment

Environment variables:

Python Interpreter: ○ Use SDK of module: python

● Use specified interpreter: Python 3.7

Interpreter options:

Working directory:

☑ Add content roots to PYTHONPATH

☑ Add source roots to PYTHONPATH

Backend Signature Verification Example

NOTE

- This example demonstrates how to write a Flask-based server as the backend of an API and implement a wrapper to verify the signature of requests sent from APIC.
- Signature information is added to requests sent to access the backend of an API only after a signature key is bound to the API.

1. Compile an API that returns **Hello World!**, and uses the **GET, POST**, **PUT**, or **DELETE** method and the **requires_apigateway_signature** wrapper.

```
app = Flask(__name__)

@app.route("/<id>", methods=['GET', 'POST', 'PUT', 'DELETE'])
@requires_apigateway_signature()
def hello(id):
    return "Hello World!"
```

2. Implement **requires_apigateway_signature** by putting the signature key and secret in a **dict**.

```
def requires_apigateway_signature():
    def wrapper(f):

        # Directly writing AK/SK in code is risky. For security, encrypt your AK/SK and store them in the configuration file or environment variables.
        # In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, set environment variables HUAWEICLOUD_SDK_AK1, HUAWEICLOUD_SDK_SK1, and HUAWEICLOUD_SDK_AK2, HUAWEICLOUD_SDK_SK2.
        secrets = {
            os.getenv('HUAWEICLOUD_SDK_AK1'): os.getenv('HUAWEICLOUD_SDK_SK1'),
            os.getenv('HUAWEICLOUD_SDK_AK2'): os.getenv('HUAWEICLOUD_SDK_SK2'),
        }
        authorizationPattern = re.compile(
            r'SDK-HMAC-SHA256\s+Access=([^,]+),\s?SignedHeaders=([^,]+),\s?Signature=(\w+)')
        BasicDateFormat = "%Y%m%dT%H%M%SZ"

        @wraps(f)
        def wrapped(*args, **kwargs):
            //Signature verification code
            ...

            return f(*args, **kwargs)
        return wrapped
    return wrapper
```

3. The **wrapped** function is the signature verification code. The verification process is as follows: Use a regular expression to parse the Authorization header. Obtain the key and signedHeaders.

```
if "authorization" not in request.headers:
    return 'Authorization not found.', 401
authorization = request.headers['authorization']
m = authorizationPattern.match(authorization)
if m is None:
    return 'Authorization format incorrect.', 401
signingKey = m.group(1)
signedHeaders = m.group(2).split(";")
```

For example, for **Authorization** header:

```
SDK-HMAC-SHA256 Access=signature_key1, SignedHeaders=host;x-sdk-date, 
Signature=e11adf65a20d1b82c25419b5********8d0ba12fed1ceb13ed00
```

The parsing result is as follows:

```
signingKey=signature_key1
signedHeaders=host;x-sdk-date
```

4. Find **secret** based on **key**. If **key** does not exist, the authentication failed.

```
if signingKey not in secrets:
    return 'Signing key not found.', 401
signingSecret = secrets[signingKey]
```

5. Create an HttpRequest, and add the method, URL, query, and signedHeaders headers to the request. Determine whether the body needs to be set.

   The body is read if there is no **x-sdk-content-sha256** header with value **UNSIGNED-PAYLOAD**.

```
r = signer.HttpRequest()
r.method = request.method
r.uri = request.path
r.query = {}
for k in request.query_string.decode('utf-8').split('&'):
    spl = k.split("=", 1)
    if len(spl) < 2:
        r.query[spl[0]] = ""
    else:
        r.query[spl[0]] = spl[1]
r.headers = {}
```

```
needbody = True
dateHeader = None
for k in signedHeaders:
    if k not in request.headers:
        return 'Signed header ' + k + ' not found', 401
    v = request.headers[k]
    if k.lower() == 'x-sdk-content-sha256' and v == 'UNSIGNED-PAYLOAD':
        needbody = False
    if k.lower() == 'x-sdk-date':
        dateHeader = v
    r.headers[k] = v
if needbody:
    r.body = request.get_data()
```

6. Check whether the signature has expired. Obtain the time from the **X-Sdk-Date** header, and check whether the difference between this time and the server time is within 15 minutes. If **signedHeaders** does not contain **X-Sdk-Date**, the authentication failed.

```
if dateHeader is None:
    return 'Header x-sdk-date not found.', 401
t = datetime.strptime(dateHeader, BasicDateFormat)
if abs(t - datetime.utcnow()) > timedelta(minutes=15):
    return 'Signature expired.', 401
```

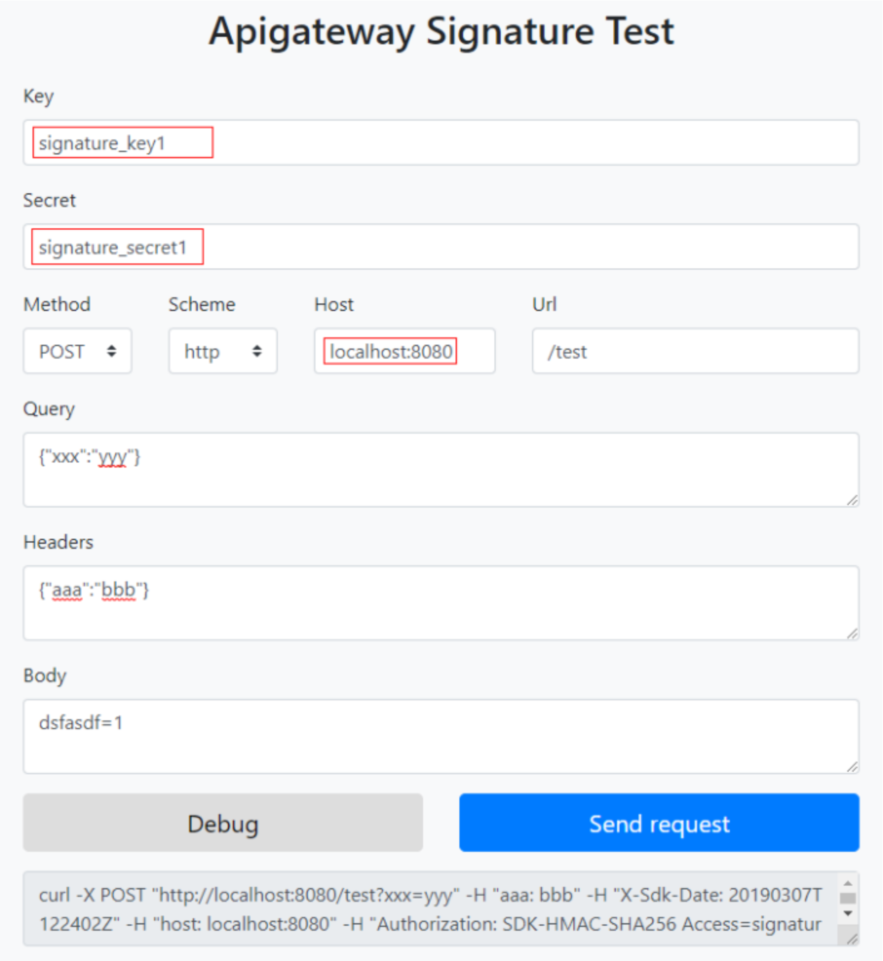
7. Invoke the **verify** method to verify the signature of the request, and check whether the verification is successful.

```
sig = signer.Signer()
sig.Key = signingKey
sig.Secret = signingSecret
if not sig.Verify(r, m.group(3)):
    return 'Verify authroization failed.', 401
```

8. Run the server to verify the code. The following example uses the HTML signature tool in the JavaScript SDK to generate a signature.

   Set the parameters according to the following figure, and click **Send request**. Copy the generated curl command, execute it in the CLI, and check whether the server returns **200**.

   If an incorrect key or secret is used, the server returns **401**, which means authentication failure.



Apigateway Signature Test

Key: signature_key1

Secret: signature_secret1

Method: POST

Scheme: http

Host: localhost:8080

Url: /test

Query: {"xxx":"yyy"}

Headers: {"aaa":"bbb"}

Body: dsfasdf=1

```
curl -X POST "http://localhost:8080/test?xxx=yyy" -H "aaa: bbb" -H "X-Sdk-Date: 20190307T
122402Z" -H "host: localhost:8080" -H "Authorization: SDK-HMAC-SHA256 Access=signatur
```

2.6.4 C#

Scenarios

To use C# to sign backend requests, obtain the C# SDK, import the project, and verify the backend signature by referring to the example provided in this section.

NOTE

The C# SDK supports only HMAC backend service signatures.

Prerequisites

- A signature key has been created on the console and bound to an API. For details, see **Configuring Signature Verification for Backend Services**.
- You have obtained the signature key and secret. For details, see **Preparations**.
- You have installed the C# development environment. For details, see **Preparations**.

Obtaining the SDK

Old version: Log in to the ROMA Connect console, choose **API Connect** > **API Management** > **Signature Keys**, and download the SDK.

New version: Log in to the ROMA Connect console, choose **API Connect** > **Credentials** > **SDKs**, and download the SDK.

Opening the Sample Project

Double-click **csharp.sln** in the SDK package to open the project. The project contains the following:

- **apigateway-signature**: Shared library that implements the signature algorithm. It can be used in the .Net Framework and .Net Core projects.
- **backend-signature**: Example of a backend signature. Modify the parameters as required. For details about the sample code, see **Backend Signature Verification Example**.
- **sdk-request**: Example of invoking the signature algorithm.

Backend Signature Verification Example

NOTE

- This example demonstrates how to write an ASP.Net Core–based server as the backend of an API and implement an IAuthorizationFilter to verify the signature of requests sent from APIC.
- Signature information is added to requests sent to access the backend of an API only after a signature key is bound to the API.

1. Write a controller that provides the GET, POST, PUT, and DELETE interfaces, and add the **ApigatewaySignatureFilter** attribute.

```
// ValuesController.cs

namespace backend_signature.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    [ApigatewaySignatureFilter]
    public class ValuesController : ControllerBase
    {
        // GET api/values
        [HttpGet]
        public ActionResult<IEnumerable<string>> Get()
        {
            return new string[] { "value1", "value2" };
        }

        // POST api/values
        [HttpPost]
        public void Post([FromBody] string value)
        {
        }

        // PUT api/values/5
        [HttpPut("{id}")]
        public void Put(int id, [FromBody] string value)
        {
        }

        // DELETE api/values/5
        [HttpDelete("{id}")]
        public void Delete(int id)
        {
        }
    }
}
```

2. Implement **ApigatewaySignatureFilter** by putting the signature key and secret in a **Dictionary**.

```
// ApigatewaySignatureFilter.cs

namespace backend_signature.Filters
{
    public class ApigatewaySignatureFilter : Attribute, IAuthorizationFilter
    {
        private Dictionary<string, string> secrets = new Dictionary<string, string>
        {
        // Directly writing AK/SK in code is risky. For security, encrypt your AK/SK and store them in the configuration file or environment variables.
        // In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, set environment variables HUAWEICLOUD_SDK_AK1, HUAWEICLOUD_SDK_SK1, and HUAWEICLOUD_SDK_AK2, HUAWEICLOUD_SDK_SK2.
            {Environment.GetEnvironmentVariable("HUAWEICLOUD_SDK_AK1"), Environment.GetEnvironmentVariable("HUAWEICLOUD_SDK_SK1")},
            {Environment.GetEnvironmentVariable("HUAWEICLOUD_SDK_AK2"), Environment.GetEnvironmentVariable("HUAWEICLOUD_SDK_SK2")},
        };

        public void OnAuthorization(AuthorizationFilterContext context) {
           //Signature verification code
             ...
        }
    }
}
```

3. The OnAuthorization function is the signature verification code. The verification process is as follows: Use a regular expression to parse the Authorization header. Obtain the key and signedHeaders.

```
private Regex authorizationPattern = new Regex("SDK-HMAC-SHA256\\s+Access=([^,]+),\\s?SignedHeaders=([^,]+),\\s?Signature=(\\w+)");

...

string authorization = request.Headers["Authorization"];
if (authorization == null)
{
    context.Result = new UnauthorizedResult();
    return;
}
var matches = authorizationPattern.Matches(authorization);
if (matches.Count == 0)
{
    context.Result = new UnauthorizedResult();
    return;
}
var groups = matches[0].Groups;
string key = groups[1].Value;
string[] signedHeaders = groups[2].Value.Split(';');
```

For example, for **Authorization** header:

```
SDK-HMAC-SHA256 Access=signature_key1, SignedHeaders=host;x-sdk-date, Signature=e11adf65a20d1b82c25419b5********8d0ba12fed1ceb13ed00
```

The parsing result is as follows:

```
signingKey=signature_key1
signedHeaders=host;x-sdk-date
```

4. Find **secret** based on **key**. If **key** does not exist, the authentication failed.

```
if (!secrets.ContainsKey(key))
{
    context.Result = new UnauthorizedResult();
    return;
}
string secret = secrets[key];
```

5. Create an HttpRequest, and add the method, URL, query, and signedHeaders headers to the request. Determine whether the body needs to be set.

   The body is read if there is no **x-sdk-content-sha256** header with value **UNSIGNED-PAYLOAD**.

```
HttpRequest sdkRequest = new HttpRequest();
sdkRequest.method = request.Method;
sdkRequest.host = request.Host.Value;
sdkRequest.uri = request.Path;
Dictionary<string, string> query = new Dictionary<string, string>();
foreach (var pair in request.Query)
{
    query[pair.Key] = pair.Value;
}
sdkRequest.query = query;
WebHeaderCollection headers = new WebHeaderCollection();
string dateHeader = null;
bool needBody = true;
foreach (var h in signedHeaders)
{
    var value = request.Headers[h];
    headers[h] = value;
    if (h.ToLower() == "x-sdk-date")
    {
        dateHeader = value;
    }
    if (h.ToLower() == "x-sdk-content-sha256" && value == "UNSIGNED-PAYLOAD")
    {
        needBody = false;
    }
}
sdkRequest.headers = headers;
if (needBody)
{
    request.EnableRewind();
    using (MemoryStream ms = new MemoryStream())
    {
        request.Body.CopyTo(ms);
        sdkRequest.body = Encoding.UTF8.GetString(ms.ToArray());
    }
    request.Body.Position = 0;
}
```

6. Check whether the signature has expired. Obtain the time from the **X-Sdk-Date** header, and check whether the difference between this time and the server time is within 15 minutes. If **signedHeaders** does not contain **X-Sdk-Date**, the authentication failed.

```
private const string BasicDateFormat = "yyyyMMddTHHmmssZ";

...

if(dateHeader == null)
{
    context.Result = new UnauthorizedResult();
    return;
}
DateTime t = DateTime.ParseExact(dateHeader, BasicDateFormat, CultureInfo.CurrentCulture);
if (Math.Abs((t - DateTime.Now).Minutes) > 15)
{
    context.Result = new UnauthorizedResult();
    return;
}
```

7. Invoke the **verify** method to verify the signature of the request, and check whether the verification is successful.

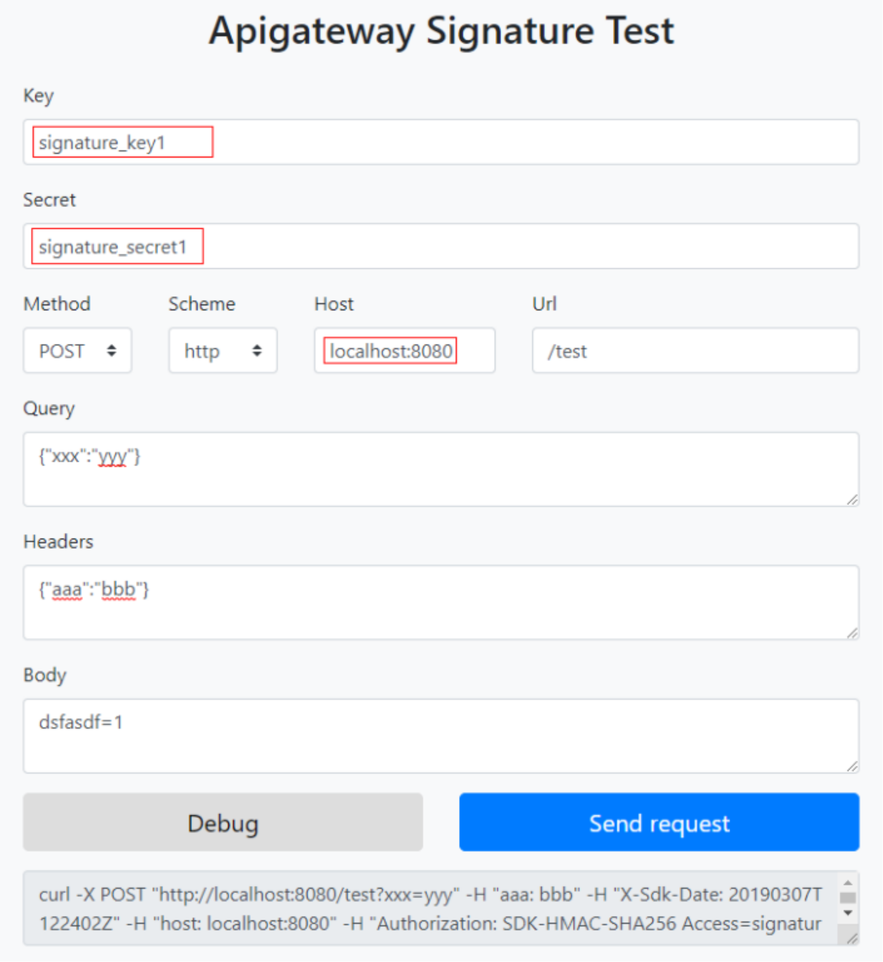
```
Signer signer = new Signer();
signer.Key = key;
signer.Secret = secret;
```

```
if (!signer.Verify(sdkRequest, groups[3].Value))
{
    context.Result = new UnauthorizedResult();
}
```

8. Run the server to verify the code. The following example uses the HTML signature tool in the JavaScript SDK to generate a signature.

   Set the parameters according to the following figure, and click **Send request**. Copy the generated curl command, execute it in the CLI, and check whether the server returns **200**.

   If an incorrect key or secret is used, the server returns **401**, which means authentication failure.



Apigateway Signature Test

Key: signature_key1

Secret: signature_secret1

Method: POST
Scheme: http
Host: localhost:8080
Url: /test

Query: {"xxx":"yyy"}

Headers: {"aaa":"bbb"}

Body: dsfasdf=1

```
curl -X POST "http://localhost:8080/test?xxx=yyy" -H "aaa: bbb" -H "X-Sdk-Date: 20190307T
122402Z" -H "host: localhost:8080" -H "Authorization: SDK-HMAC-SHA256 Access=signatur
```

3 Developer Guide for Message Integration

3.1 Overview

3.1.1 Scenarios

Description

MQS of ROMA Connect is fully compatible with the open-source Kafka protocol. Service applications need to develop and integrate an **open-source Kafka client** or the RESTful APIs provided by ROMA Connect to implement message connection with MQS.

- Open-source client integration: Service applications can integrate with the open-source Kafka client and connect to MQS through the client to produce and consume messages.
- RESTful API integration: Service applications call RESTful APIs to connect to MQS and produce and consume messages.

3.1.2 Specifications

- **Development tool versions**:
  - IntelliJ IDEA: 2018.3.5 or later
  - Eclipse: 3.6.0 or later
  - Visual Studio: 2019 version 16.8.4 or later.
- **Development language versions**:

  - Java: Java Development Kit 1.8.111 or later
  - Python: 2.7 or 3.*X*
  - Go: 1.14 or later
  - C#: .NET 6.0 or later
- **Suggested client versions:**

Kafka Version	Recommended Kafka Client Version
1.1.0	● Java: 1.1.0 or later ● Python: 2.0.1 or later ● Go: 1.8.2 or later ● C#: 1.5.2 or later
2.7	● Java: 2.7.2 or later ● Python: 2.0.1 or later ● Go: 1.8.2 or later ● C#: 1.5.2 or later

- **Client development and configuration suggestions:**

  For details, see **Recommendations for Client Usage** and **Setting Parameters for Clients**.

3.1.3 Recommendations for Client Usage

Applicability for Consumers

- Ensure that the owner thread does not exit abnormally. Otherwise, the client may fail to initiate consumption requests and the consumption will be blocked.
- Ensure that the commit operation is performed after messages are processed. This is to avoid the failure of processing service messages and the failure to retrieve the message that fails to be processed.
- A consumer cannot frequently join or leave a group. Otherwise, the consumer will frequently perform rebalancing, which blocks consumption.
- The number of consumers cannot be greater than the number of partitions in the topic. Otherwise, some consumers may fail to poll for messages.
- Ensure that the consumer polls at regular intervals to keep sending heartbeats to the server. If the consumer stops sending heartbeats for long enough, the consumer session will time out and the consumer will be considered to have stopped. This will also block consumption.
- Ensure that there is a limitation on the size of messages buffered locally to avoid an out-of-memory (OOM) situation.
- The timeout interval of the consumer session is set to 30 seconds, and **session.timeout.ms** is set to **30000**. This prevents the consumer from performing rebalance due to frequent timeout. Frequent timeout will block consumption.

- ROMA Connect may consume repeated messages. The service side must ensure the idempotency of message processing.
- Always close the consumer before exiting. Otherwise, consumers in the same group may block the **session.timeout.ms** time.

Applicability for Producers

- Synchronous replication: Set **acks** to **-1**.
- Retry message sending: Set **retries** to **3**.
- Message sending optimization: Set **linger.ms** to **0**.
- Ensure that the producer has sufficient JVM memory to avoid blockages.

Applicability of Topics

- Recommended configuration: three copies
- The recommended maximum number of partitions for a topic is 20.
- Each topic can have 3 replicas (the number of replicas cannot be modified once configured).

Other Suggestions

- Maximum number of connections: 3000
- Maximum size of a message: 10 MB
- Access ROMA Connect using SASL_SSL. Ensure that your DNS service is capable of resolving an IP address to a domain name. Alternatively, map all ROMA Connect broker IP addresses to host names in the **hosts** file. Prevent Kafka clients from performing reverse resolution. Otherwise, connections may fail to be established.
- Apply for a disk space size that is more than twice the size of service data multiplied by the number of replicas. In other words, keep 50% of the disk space idle.
- Avoid frequent full GC in JVM. Otherwise, message production and consumption will be blocked.
- Configure log dump on the Kafka client, or logs may take up the disk space.
- A maximum of 500 consumers in the same consumer group can connect to the same MQS. If the number of consumers exceeds 500, the connection fails. If a consumer group with over 500 consumers needs to connect to an MQS, put the consumers into multiple consumer groups.

NOTE

- If both SASL_SSL and intra-VPC plaintext access are enabled for MQS of the ROMA Connect instance, the SASL mode cannot be used for connecting to MQS topics in the VPC.
- If the SASL mode is used for connecting to MQS topics, you are advised to configure the mapping between the host and IP address in the **/etc/hosts** file on the host where the client is located. Otherwise, network delay will occur.

  Set the IP address to the connection address of MQS and set the host to the name of each instance host. Ensure that the name of each host is unique. Example:

  10.10.10.11 host01

  10.10.10.12 host02

  10.10.10.13 host03

3.1.4 Setting Parameters for Clients

This section provides recommendations on configuring common parameters for Kafka producers and consumers.

Table 3-1 Producer parameters

Parameter	Default Value	Recommended Value	Description
acks	1	**all** (if high reliability mode is selected) **1** (if high throughput mode is selected)	Number of acknowledgments the producer requires the server to return before considering a request complete. This controls the durability of records that are sent. Options: ● **0**: The producer will not wait for any acknowledgment from the server at all. The record will be immediately added to the socket buffer and considered sent. No guarantee can be made that the server has received the record, and the retries configuration will not take effect (as the client generally does not know of any failures). The offset given back for each record will always be set to –1. ● **1**: The leader will write the record to its local log but will respond without waiting until receiving full acknowledgment from all followers. ● **all**: The leader will wait for the full set of replicas to acknowledge the record. This is the strongest available guarantee because the record will not be lost even if there is just one replica that works.

Parameter	Default Value	Recommended Value	Description
retries	0	/	Number of times that the client resends a message. Setting this parameter to a value greater than zero will cause the client to resend any record that failed to be sent. Note that these retries are no different than those the client perform upon receiving a message sending error. Allowing retries will potentially change the message order. For example, if two messages are sent to the same partition, and the first fails and is retried but the second succeeds, then the second message may appear first.
request.timeout.ms	30000	/	Maximum amount of time the client will wait for the response of a request. If the response is not received before the timeout elapses, the client will throw a timeout exception. Setting this parameter to a large value, for example, **120000** (120s), can prevent records from failing to be sent in high-concurrency scenarios.
block.on.buffer.full	TRUE	TRUE	Setting this parameter to **TRUE** indicates that when buffer memory is exhausted, the producer must stop receiving new message records or throw an exception. By default, this parameter is set to **TRUE**. However, in some cases, non-blocking usage is desired and it is better to throw an exception immediately. Setting this parameter to **FALSE** will cause the producer to instead throw "BufferExhaustedException" when buffer memory is exhausted.

Parameter	Default Value	Recommended Value	Description
batch.size	16384	262144	Default maximum number of bytes of messages that can be processed at a time. The producer will attempt to batch process the message records to reduce the number of requests. This improves the performance between the client and the server. No attempt will be made to batch records larger than this size. Requests sent to brokers will contain multiple batches, one for each partition with data available to be sent. A smaller batch size will make batching less common and may reduce throughput (a batch size of zero will disable batching entirely). A larger batch size may use more memory as a buffer of the specified batch size will always be allocated in anticipation of additional records.
buffer.memory	33554432	67108864	Total bytes of memory the producer can use to buffer records waiting to be sent to the server. If the data generation speed is greater than the speed of sending data to the broker, the producer blocks or throws an exception, which is indicated by block.on.buffer.full. This setting should correspond roughly to the total memory the producer will use, but is not a rigid bound since not all memory the producer uses is used for buffering. Some additional memory will be used for compression (if compression is enabled) as well as for maintaining in-flight requests.
max.request.size	1048576	5242880	Maximum number of message bytes a producer can send to a server at a time. This parameter affects the number of message records produced at a time.

Table 3-2 Consumer parameters

Parameter	Default Value	Recommended Value	Description
auto.commit.enable	TRUE	FALSE	If this parameter is set to **TRUE**, the offset of messages already fetched by the consumer will be periodically committed to ZooKeeper. This committed offset will be used when the process fails as the position from which the new consumer will begin. Constraints: If this parameter is set to **FALSE**, to avoid message loss, an offset must be committed to ZooKeeper after the messages are successfully consumed.
auto.offset.reset	latest	earliest	Indicates what to do when there is no initial offset in ZooKeeper or if the current offset has been deleted. Options: **earliest**: The offset is automatically reset to the smallest offset. **latest**: Automatically reset to the largest offset. **none**: The system throws an exception to the consumer if no offset is available. **anything else**: The system throws an exception to the consumer.
connections.max.idle.ms	600000	30000	Indicates the timeout interval for an idle connection. The server closes the idle connection after this period of time ends. Setting this parameter to 30000 can reduce the server response failures when the network condition is poor.

3.2 Preparations

Obtaining MQS Connection Information

- Obtain the connection address and port.

  On the ROMA Connect instance console, choose **Instance Information** > **Basic Information** to view the MQS connection addresses.

  - Use the Kafka client to connect to MQS through the internal network: View the MQS intranet addresses under **Connection Addresses**.
  - Use the Kafka client to connect to MQS through the public network: View the MQS public addresses under **Connection Addresses**.

   - Connect to MQS through a RESTful API: View the message RESTful API under **MQS Information**.
- Obtain the topic name.

  On the ROMA Connect console, choose **Message Queue Service** > **Topic Management** and view the topic name.
- SASL authentication information

  If MQS SASL_SSL is enabled for the ROMA Connect instance, you need to obtain the username, password, and client certificate.
   - Username and password

     On the **Integration Applications** page of the ROMA Connect console, click the name of the integration application to which the topic belongs. In the **Basic Information** area of the **Overview** tab page, you can view the values of **Key** and **Secret**, that is, the username and password.
   - Client certificate

     On the ROMA Connect console, choose **Message Queue Service** > **Topic Management**, and click **Download SSL Certificate** to download the **client certificate**.

Preparing the Development Environment

- Install a development tool.

  Select a proper development tool based on the language used.
   - Download the installation package of IntelliJ IDEA 2018.3.5 or later from the **official IntelliJ IDEA website**.
   - Install Apache Maven 3.0.3 or a later version. Download the installation package from the **official Maven website**.
   - Download the Visual Studio 2019 installation package of 16.8.4 or later from the **official Visual Studio page**.
- Install a development language.
   - Java: Download the Java Development Kit of 1.8.111 or later from the **official Oracle website**.
   - Python: Download the Python 2.7 or 3.*X* installation package from the **official Python website**.
   - Go: Download the installation package of Go 1.14 or a later version from the **official Go website**.
   - C#: Download the installation package of .NET 6.0 or a later version from the **official .NET website**.

3.3 Configuring MQS Connection (Open-Source Client)

3.3.1 Configuring a Kafka Client in Java

Scenarios

This section describes how to connect to a Java-based Kafka client and how to produce and consume messages.

Prerequisites

- You have obtained MQS connection information. For details, see **Preparations**.
- You have installed the development tool and Java development environment. For details, see **Preparations**.

Installing the Kafka Client

MQS is developed based on Kafka 1.1.0 and 2.7. View the Kafka version information in the **MQS Information** area on the **Instance Information** page on the ROMA Connect console. For details about how to use the Java open-source client, see **suggested client versions**.

Select the client version based on the Kafka version of the instance. The following uses the 2.7.2 version as an example.

```
<dependency>
   <groupId>org.apache.kafka</groupId>
   <artifactId>kafka-clients</artifactId>
   <version>2.7.2</version>
</dependency>
```

Modifying Configuration Information

The following describes example producer and consumer configuration files. If SASL authentication is enabled for a ROMA Connect instance, you must configure SASL authentication information in the configuration file of the Java client. Otherwise, the connection fails. If SASL authentication is not enabled, comment out the related configuration.

- Producer configuration file (corresponding to the **mqs.sdk.producer.properties** file in the production message code)

  The information in bold is subject to different MQSs and must be modified based on site requirements. Other parameters of the client can be added as required.

```
#The topic name is in the specific production and consumption code.
#######################
#For example, bootstrap.servers=192.168.0.196:9095,192.168.0.196:9096,192.168.0.196:9094.
bootstrap.servers=ip1:port1,ip2:port2,ip3:port3
#Send acknowledgment parameters.
acks=all
#Sequence mode of the key.
key.serializer=org.apache.kafka.common.serialization.StringSerializer
#Sequence mode of the value.
value.serializer=org.apache.kafka.common.serialization.StringSerializer
#Total bytes of memory the producer can use to buffer records waiting to be sent to the server.
buffer.memory=33554432
#Number of retries.
retries=0
#######################
#If SASL authentication is not used, comment out the following parameters:
#######################
#Set the username and password.
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
    username="username" \
    password="********";
#SASL authentication mode.
sasl.mechanism=PLAIN
#Encryption protocol. Currently, the SASL_SSL protocol is supported.
security.protocol=SASL_SSL
```

```
#Location of the SSL truststore file.
ssl.truststore.location=E:\\temp\\client.truststore.jks
#Password of the SSL truststore file. The value is fixed and cannot be changed. This password is used
to access the JKS file generated by Java.
ssl.truststore.password=dms@kafka
ssl.endpoint.identification.algorithm=
```

The parameters in the example code are as follows. For details about how to obtain the parameter values, see **Obtaining MQS Connection Information**.

- **bootstrap.servers**: MQS connection addresses and ports
- **username** and **password**: username and password used for SASL_SSL authentication
- **ssl.truststore.location**: client certificate used for SASL_SSL authentication

- Consumer configuration file (corresponding to the **mqs.sdk.consumer.properties** file in the consumption message code)

  The information in bold is subject to different MQSs and must be modified based on site requirements. Other parameters of the client can be added as required.

  ```
  #The topic name is in the specific production and consumption code.
  #######################
  #For example, bootstrap.servers=192.168.0.196:9095,192.168.0.196:9096,192.168.0.196:9094.
  bootstrap.servers=ip1:port1,ip2:port2,ip3:port3
  #A character string that uniquely identifies the group to which the consumer process belongs. You
  can set it as required.
  #If group id is set to the same value, the processes belong to the same consumer group.
  group.id=1
  #Sequence mode of the key.
  key.deserializer=org.apache.kafka.common.serialization.StringDeserializer
  #Sequence mode of the value.
  value.deserializer=org.apache.kafka.common.serialization.StringDeserializer
  #Offset mode.
  auto.offset.reset=earliest
  #######################
  #If SASL authentication is not used, comment out the following parameters:
  #######################
  #Set the jaas username and password on the console.
  sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
      username="username" \
      password="********";
  #SASL authentication mode.
  sasl.mechanism=PLAIN
  #Encryption protocol. Currently, the SASL_SSL protocol is supported.
  security.protocol=SASL_SSL
  #Location of the SSL truststore file.
  ssl.truststore.location=E:\\temp\\client.truststore.jks
  #Password of the SSL truststore file for accessing the JKS file generated by Java.
  ssl.truststore.password=dms@kafka
  ssl.endpoint.identification.algorithm=
  ```

  The parameters in the example code are as follows. For details about how to obtain the parameter values, see **Obtaining MQS Connection Information**.

  - **bootstrap.servers**: MQS connection addresses and ports
  - **group.id**: consumer group name. If the specified consumer group does not exist, the system automatically creates one.
  - **username** and **password**: username and password used for SASL_SSL authentication
  - **ssl.truststore.location**: client certificate used for SASL_SSL authentication

Producing Messages

- Test code:

```
package com.mqs.producer;

import org.apache.kafka.clients.producer.Callback;
import org.apache.kafka.clients.producer.RecordMetadata;
import org.junit.Test;

public class MqsProducerTest {
    @Test
    public void testProducer() throws Exception {
        MqsProducer<String, String> producer = new MqsProducer<String, String>();
        int partiton = 0;
        try {
            for (int i = 0; i < 10; i++) {
                String key = null;
                String data = "The msg is " + i;
                //Enter the name of the topic you created. There are multiple APIs for producing messages.
For details, see the Kafka official website or the following production message code.
                producer.produce("topicName", partiton, key, data, new Callback() {
                    public void onCompletion(RecordMetadata metadata,
                        Exception exception) {
                        if (exception != null) {
                            exception.printStackTrace();
                            return;
                        }
                        System.out.println("produce msg completed");
                    }
                });
                System.out.println("produce msg:" + data);
            }
        } catch (Exception e) {
            //TODO: troubleshooting
            e.printStackTrace();
        } finally {
            producer.close();
        }
    }
}
```

- Production message code:

```
package com.mqs.producer;

import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.List;
import java.util.Properties;

import org.apache.kafka.clients.producer.Callback;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerRecord;

public class MqsProducer<K, V> {
    //Introduce configuration information about production messages. For details, see the preceding
description.
    public static final String CONFIG_PRODUCER_FILE_NAME = "mqs.sdk.producer.properties";

    private Producer<K, V> producer;

    MqsProducer(String path)
    {
        Properties props = new Properties();
        try {
```

```
        InputStream in = new BufferedInputStream(new FileInputStream(path));
        props.load(in);
    }catch (IOException e)
    {
        e.printStackTrace();
        return;
    }
    producer = new KafkaProducer<K,V>(props);
}
MqsProducer()
{
    Properties props = new Properties();
    try {
        props = loadFromClasspath(CONFIG_PRODUCER_FILE_NAME);
    }catch (IOException e)
    {
        e.printStackTrace();
        return;
    }
    producer = new KafkaProducer<K,V>(props);
}

/**
 * Production messages
 *
 * @param topic        topic object
 * @param partition    partition
 * @param key          message key
 * @param data         message data
 */
public void produce(String topic, Integer partition, K key, V data)
{
    produce(topic, partition, key, data, null, (Callback)null);
}

/**
 * Production messages
 *
 * @param topic        topic object
 * @param partition    partition
 * @param key          message key
 * @param data         message data
 * @param timestamp    timestamp
 */
public void produce(String topic, Integer partition, K key, V data, Long timestamp)
{
    produce(topic, partition, key, data, timestamp, (Callback)null);
}
/**
 * Production messages
 *
 * @param topic        topic object
 * @param partition    partition
 * @param key          message key
 * @param data         message data
 * @param callback     callback
 */
public void produce(String topic, Integer partition, K key, V data, Callback callback)
{
    produce(topic, partition, key, data, null, callback);
}

public void produce(String topic, V data)
{
    produce(topic, null, null, data, null, (Callback)null);
}

/**
 * Production messages
```

```
     *
     * @param topic         topic object
     * @param partition     partition
     * @param key           message key
     * @param data          message data
     * @param timestamp     timestamp
     * @param callback     callback
     */
    public void produce(String topic, Integer partition, K key, V data, Long timestamp, Callback 
callback)
    {
        ProducerRecord<K, V> kafkaRecord =
                timestamp == null ? new ProducerRecord<K, V>(topic, partition, key, data)
                        : new ProducerRecord<K, V>(topic, partition, timestamp, key, data);
        produce(kafkaRecord, callback);
    }

    public void produce(ProducerRecord<K, V> kafkaRecord)
    {
        produce(kafkaRecord, (Callback)null);
    }

    public void produce(ProducerRecord<K, V> kafkaRecord, Callback callback)
    {
        producer.send(kafkaRecord, callback);
    }

    public void close()
    {
        producer.close();
    }

    /**
     * get classloader from thread context if no classloader found in thread
     * context return the classloader which has loaded this class
     *
     * @return classloader
     */
    public static ClassLoader getCurrentClassLoader()
    {
        ClassLoader classLoader = Thread.currentThread()
                .getContextClassLoader();
        if (classLoader == null)
        {
            classLoader = MqsProducer.class.getClassLoader();
        }
        return classLoader;
    }

    /**
     *Load configuration information from classpath.
     *
     * @param configFileName configuration file name
     * @return configuration information
     * @throws IOException
     */
    public static Properties loadFromClasspath(String configFileName) throws IOException
    {
        ClassLoader classLoader = getCurrentClassLoader();
        Properties config = new Properties();

        List<URL> properties = new ArrayList<URL>();
        Enumeration<URL> propertyResources = classLoader
                .getResources(configFileName);
        while (propertyResources.hasMoreElements())
        {
            properties.add(propertyResources.nextElement());
        }
        for (URL url:properties)
        {
            InputStream is = null;
            try
            {
                is = url.openStream();
                config.load(is);
            }
            finally
            {
                if (is != null)
                {
                    is.close();
                    is = null;
                }
            }
        }

        return config;
    }
}
```

Consuming Messages

- Test code:

```
package com.mqs.consumer;

import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.junit.Test;
import java.util.Arrays;

public class MqsConsumerTest {
    @Test
    public void testConsumer() throws Exception {
        MqsConsumer consumer = new MqsConsumer();
        //Enter the name of the topic that consumes messages.
        consumer.consume(Arrays.asList("topicName"));
        try {
            for (int i = 0; i < 10; i++){
                ConsumerRecords<Object, Object> records = consumer.poll(1000);
                System.out.println("the numbers of topic:" + records.count());
                for (ConsumerRecord<Object, Object> record : records)
                {
                    System.out.println(record.toString());
                }
            }
        }catch (Exception e)
        {
            //TODO: troubleshooting
            e.printStackTrace();
        }finally {
            consumer.close();
        }
    }
}
```

- Consumption message code:

```
package com.mqs.consumer;

import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.util.*;
```

```
public class MqsConsumer {

    public static final String CONFIG_CONSUMER_FILE_NAME = "mqs.sdk.consumer.properties";

    private KafkaConsumer<Object, Object> consumer;

    MqsConsumer(String path)
    {
        Properties props = new Properties();
        try {
            InputStream in = new BufferedInputStream(new FileInputStream(path));
            props.load(in);
        }catch (IOException e)
        {
            e.printStackTrace();
            return;
        }
        consumer = new KafkaConsumer<Object, Object>(props);
    }

    MqsConsumer()
    {
        Properties props = new Properties();
        try {
            props = loadFromClasspath(CONFIG_CONSUMER_FILE_NAME);
        }catch (IOException e)
        {
            e.printStackTrace();
            return;
        }
        consumer = new KafkaConsumer<Object, Object>(props);
    }
    public void consume(List topics)
    {
        consumer.subscribe(topics);
    }

    public ConsumerRecords<Object, Object> poll(long timeout)
    {
        return consumer.poll(timeout);
    }

    public void close()
    {
        consumer.close();
    }

    /**
     * get classloader from thread context if no classloader found in thread
     * context return the classloader which has loaded this class
     *
     * @return classloader
     */
    public static ClassLoader getCurrentClassLoader()
    {
        ClassLoader classLoader = Thread.currentThread()
                .getContextClassLoader();
        if (classLoader == null)
        {
            classLoader = MqsConsumer.class.getClassLoader();
        }
        return classLoader;
    }

    /**
     *Load configuration information from classpath.
     *
     * @param configFileName configuration file name
     * @return configuration information
     * @throws IOException
     */
    public static Properties loadFromClasspath(String configFileName) throws IOException
    {
        ClassLoader classLoader = getCurrentClassLoader();
        Properties config = new Properties();

        List<URL> properties = new ArrayList<URL>();
        Enumeration<URL> propertyResources = classLoader
                .getResources(configFileName);
        while (propertyResources.hasMoreElements())
        {
            properties.add(propertyResources.nextElement());
        }


        {
            InputStream is = null;
            try
            {
                is = url.openStream();
                config.load(is);
            }
            finally
            {
                if (is != null)
                {
                    is.close();
                    is = null;
                }
            }
        }

        return config;
    }
}
```

3.3.2 Configuring a Kafka Client in Python

Scenarios

This section uses the Linux CentOS environment as an example to describe how to connect a Python Kafka client to MQS (including Kafka client installation), and how to produce and consume messages.

Prerequisites

- You have obtained MQS connection information. For details, see **Preparations**.
- You have installed the development tool and Python development environment. For details, see **Preparations**.

Installing the Kafka Client

MQS is developed based on Kafka 1.1.0 and 2.7. View the Kafka version information in the **MQS Information** area on the **Instance Information** page on the ROMA Connect console. For details about how to use the Python open-source client, see **suggested client versions**.

Run the following command to install the Python Kafka client of the corresponding version:

```
pip install kafka-python==2.0.1
```

Producing Messages

- SASL authentication mode

  Replace the information in bold with the actual values.

```
from kafka import KafkaProducer
import ssl
##Connection information
conf = {
    'bootstrap_servers': ["ip1:port1","ip2:port2","ip3:port3"],
    'topic_name': 'topic_name',
    'sasl_plain_username': 'username',
    'sasl_plain_password': 'password'
}

context = ssl.create_default_context()
context = ssl.SSLContext(ssl.PROTOCOL_SSLv23)
context.verify_mode = ssl.CERT_REQUIRED
##Certificate file
context.load_verify_locations("phy_ca.crt")

print('start producer')
producer = KafkaProducer(bootstrap_servers=conf['bootstrap_servers'],
                         sasl_mechanism="PLAIN",
                         ssl_context=context,
                         security_protocol='SASL_SSL',
                         sasl_plain_username=conf['sasl_plain_username'],
                         sasl_plain_password=conf['sasl_plain_password'])

data = bytes("hello kafka!", encoding="utf-8")
producer.send(conf['topic_name'], data)
producer.close()
print('end producer')
```

  The parameters in the example code are as follows. For details about how to obtain the parameter values, see **Obtaining MQS Connection Information**.

  - **bootstrap_servers**: MQS connection addresses and ports
  - **topic_name**: name of the topic that produces messages
  - **sasl_plain_username** and **sasl_plain_password**: username and password used for SASL_SSL authentication
  - **context.load_verify_locations**: client certificate used for SASL_SSL authentication

- Non-SASL authentication mode

  Replace the information in bold with the actual values.

```
from kafka import KafkaProducer

conf = {
    'bootstrap_servers': ["ip1:port1","ip2:port2","ip3:port3"],
    'topic_name': 'topic_name',
}

print('start producer')
producer = KafkaProducer(bootstrap_servers=conf['bootstrap_servers'])

data = bytes("hello kafka!", encoding="utf-8")
producer.send(conf['topic_name'], data)
producer.close()
print('end producer')
```

  The parameters in the example code are as follows. For details about how to obtain the parameter values, see **Obtaining MQS Connection Information**.

  - **bootstrap_servers**: MQS connection addresses and ports

- **topic_name**: name of the topic that produces messages

Consuming Messages

- SASL authentication mode

  Replace the information in bold with the actual values.

```
from kafka import KafkaConsumer
import ssl
##Connection information
conf = {
    'bootstrap_servers': ["ip1:port1","ip2:port2","ip3:port3"],
    'topic_name': 'topic_name',
    'sasl_plain_username': 'username',
    'sasl_plain_password': 'password',
    'consumer_id': 'consumer_id'
}

context = ssl.create_default_context()
context = ssl.SSLContext(ssl.PROTOCOL_SSLv23)
context.verify_mode = ssl.CERT_REQUIRED
##Certificate file
context.load_verify_locations("phy_ca.crt")

print('start consumer')
consumer = KafkaConsumer(conf['topic_name'],
                         bootstrap_servers=conf['bootstrap_servers'],
                         group_id=conf['consumer_id'],
                         sasl_mechanism="PLAIN",
                         ssl_context=context,
                         security_protocol='SASL_SSL',
                         sasl_plain_username=conf['sasl_plain_username'],
                         sasl_plain_password=conf['sasl_plain_password'])

for message in consumer:
    print("%s:%d:%d: key=%s value=%s" % (message.topic, message.partition,message.offset, message.key,message.value))

print('end consumer')
```

  The parameters in the example code are as follows. For details about how to obtain the parameter values, see **Obtaining MQS Connection Information**.

  - **bootstrap_servers**: MQS connection addresses and ports
  - **topic_name**: name of the topic that consumes messages
  - **sasl_plain_username** and **sasl_plain_password**: username and password used for SASL_SSL authentication
  - **consumer_id**: consumer group name. If the specified consumer group does not exist, the system automatically creates one.
  - **context.load_verify_locations**: client certificate used for SASL_SSL authentication

- Non-SASL authentication mode

  Replace the information in bold with the actual values.

```
from kafka import KafkaConsumer

conf = {
    'bootstrap_servers': ["ip1:port1","ip2:port2","ip3:port3"],
    'topic_name': 'topic_name',
    'consumer_id': 'consumer_id'
}

print('start consumer')
consumer = KafkaConsumer(conf['topic_name'],
                bootstrap_servers=conf['bootstrap_servers'],
                group_id=conf['consumer_id'])

for message in consumer:
    print("%s:%d:%d: key=%s value=%s" % (message.topic, message.partition,message.offset,
message.key,message.value))

print('end consumer')
```

The parameters in the example code are as follows. For details about how to obtain the parameter values, see **Obtaining MQS Connection Information**.

- **bootstrap_servers**: MQS connection addresses and ports
- **topic_name**: name of the topic that consumes messages
- **consumer_id**: consumer group name. If the specified consumer group does not exist, the system automatically creates one.

3.3.3 Configuring a Kafka Client in Go

Scenarios

This section uses Linux CentOS as an example to describe how to connect a Go Kafka client to MQS (including Kafka client installation), and how to produce and consume messages.

Prerequisites

- You have obtained MQS connection information. For details, see **Preparations**.
- You have installed the development tool and Python development environment. For details, see **Preparations**.

Installing the Kafka Client

MQS is developed based on Kafka 1.1.0 and 2.7. View the Kafka version information in the **MQS Information** area on the **Instance Information** page on the ROMA Connect console. For details about how to use the Go open-source client, see **suggested client versions**.

Run the following command to install the Go Kafka client of the corresponding version:

```
go get github.com/confluentinc/confluent-kafka-go/kafka
```

Producing Messages

- SASL authentication mode

  Replace the information in bold with the actual values.

```
package main

import (
    "bufio"
    "fmt"
    "github.com/confluentinc/confluent-kafka-go/kafka"
    "log"
    "os"
    "os/signal"
    "syscall"
)
```

```
var (
    brokers  = "ip1:port1,ip2:port2,ip3:port3"
    topics   = "topic_name"
    user     = "username"
    password = "password"
    caFile   = "phy_ca.crt"
)

func main() {
    log.Println("Starting a new kafka producer")

    config := &kafka.ConfigMap{
        "bootstrap.servers": brokers,
        "security.protocol": "SASL_SSL",
        "sasl.mechanism":    "PLAIN",
        "sasl.username":     user,
        "sasl.password":     password,
        "ssl.ca.location":   caFile,
    }
    producer, err := kafka.NewProducer(config)
    if err != nil {
        log.Panicf("producer error, err: %v", err)
        return
    }

    go func() {
        for e := range producer.Events() {
            switch ev := e.(type) {
            case *kafka.Message:
                if ev.TopicPartition.Error != nil {
                    log.Printf("Delivery failed: %v\n", ev.TopicPartition)
                } else {
                    log.Printf("Delivered message to %v\n", ev.TopicPartition)
                }
            }
        }
    }()

    // Produce messages to topic (asynchronously)
    fmt.Println("please enter message:")
    go func() {
        for {
            err := producer.Produce(&kafka.Message{
                TopicPartition: kafka.TopicPartition{Topic: &topics, Partition: kafka.PartitionAny},
                Value:          GetInput(),
            }, nil)
            if err != nil {
                log.Panicf("send message fail, err: %v", err)
                return
            }
        }
    }()

    sigterm := make(chan os.Signal, 1)
    signal.Notify(sigterm, syscall.SIGINT, syscall.SIGTERM)
    select {
    case <-sigterm:
        log.Println("terminating: via signal")
    }
    // Wait for message deliveries before shutting down
    producer.Flush(15 * 1000)
    producer.Close()
}

func GetInput() []byte {
    reader := bufio.NewReader(os.Stdin)
    data, _, _ := reader.ReadLine()
    return data
}
```

The parameters in the example code are as follows. For details about how to obtain the parameter values, see **Obtaining MQS Connection Information**.

- **brokers**: MQS connection addresses and ports
- **topics**: names of the topics that produce messages
- **user** and **password**: username and password used for SASL_SSL authentication
- **caFile**: client certificate used for SASL_SSL authentication

- Non-SASL authentication mode

  Replace the information in bold with the actual values.

```
package main

import (
    "bufio"
    "fmt"
    "github.com/confluentinc/confluent-kafka-go/kafka"
    "log"
    "os"
    "os/signal"
    "syscall"
)

var (
    brokers  = "ip1:port1,ip2:port2,ip3:port3"
    topics   = "topic_name"
)

func main() {
    log.Println("Starting a new kafka producer")

    config := &kafka.ConfigMap{
        "bootstrap.servers": brokers,
    }
    producer, err := kafka.NewProducer(config)
    if err != nil {
        log.Panicf("producer error, err: %v", err)
        return
    }

    go func() {
        for e := range producer.Events() {
            switch ev := e.(type) {
            case *kafka.Message:
                if ev.TopicPartition.Error != nil {
                    log.Printf("Delivery failed: %v\n", ev.TopicPartition)
                } else {
                    log.Printf("Delivered message to %v\n", ev.TopicPartition)
                }
            }
        }
    }()

    // Produce messages to topic (asynchronously)
    fmt.Println("please enter message:")
    go func() {
        for {
            err := producer.Produce(&kafka.Message{
                TopicPartition: kafka.TopicPartition{Topic: &topics, Partition: kafka.PartitionAny},
                Value:          GetInput(),
            }, nil)
            if err != nil {
                log.Panicf("send message fail, err: %v", err)
                return
            }
        }
    }()

    sigterm := make(chan os.Signal, 1)
    signal.Notify(sigterm, syscall.SIGINT, syscall.SIGTERM)
    select {
    case <-sigterm:
        log.Println("terminating: via signal")
    }
    // Wait for message deliveries before shutting down
    producer.Flush(15 * 1000)
    producer.Close()
}

func GetInput() []byte {
    reader := bufio.NewReader(os.Stdin)
    data, _, _ := reader.ReadLine()
    return data
}
```

The parameters in the example code are as follows. For details about how to obtain the parameter values, see **Obtaining MQS Connection Information**.

- **brokers**: MQS connection addresses and ports
- **topics**: names of the topics that produce messages

Consuming Messages

- SASL authentication mode

  Replace the information in bold with the actual values.

```
package main

import (
    "fmt"
    "github.com/confluentinc/confluent-kafka-go/kafka"
    "log"
    "os"
    "os/signal"
    "syscall"
)

var (
    brokers  = "ip1:port1,ip2:port2,ip3:port3"
    group    = "group_id"
    topics   = "topic_name"
    user     = "username"
    password = "password"
    caFile   = "phy_ca.crt"
)

func main() {
    log.Println("Starting a new kafka consumer")

    config := &kafka.ConfigMap{
        "bootstrap.servers": brokers,
        "group.id":          group,
        "auto.offset.reset": "earliest",
        "security.protocol": "SASL_SSL",
        "sasl.mechanism":    "PLAIN",
        "sasl.username":     user,
        "sasl.password":     password,
        "ssl.ca.location":   caFile,
    }

    consumer, err := kafka.NewConsumer(config)
    if err != nil {
        log.Panicf("Error creating consumer: %v", err)
        return
    }

    err = consumer.SubscribeTopics([]string{topics}, nil)
    if err != nil {
        log.Panicf("Error subscribe consumer: %v", err)
        return
    }

    go func() {
        for {
            msg, err := consumer.ReadMessage(-1)
            if err != nil {
                log.Printf("Consumer error: %v (%v)", err, msg)
            } else {
                fmt.Printf("Message on %s: %s\n", msg.TopicPartition, string(msg.Value))
            }
        }
    }()

    sigterm := make(chan os.Signal, 1)
    signal.Notify(sigterm, syscall.SIGINT, syscall.SIGTERM)
    select {
    case <-sigterm:
        log.Println("terminating: via signal")
    }
    if err = consumer.Close(); err != nil {
        log.Panicf("Error closing consumer: %v", err)
    }
}
```

The parameters in the example code are as follows. For details about how to obtain the parameter values, see **Obtaining MQS Connection Information**.

- **brokers**: MQS connection addresses and ports
- **group**: consumer group name. If the specified consumer group does not exist, the system automatically creates one.
- **topics**: names of the topics that consume messages
- **user** and **password**: username and password used for SASL_SSL authentication
- **caFile**: client certificate used for SASL_SSL authentication

- Non-SASL authentication mode

  Replace the information in bold with the actual values.

```
package main

import (
    "fmt"
    "github.com/confluentinc/confluent-kafka-go/kafka"
    "log"
    "os"
    "os/signal"
    "syscall"
)

var (
    brokers  = "ip1:port1,ip2:port2,ip3:port3"
    group    = "group_id"
    topics   = "topic_name"
)

func main() {
    log.Println("Starting a new kafka consumer")

    config := &kafka.ConfigMap{
        "bootstrap.servers": brokers,
        "group.id":          group,
        "auto.offset.reset": "earliest",
    }

    consumer, err := kafka.NewConsumer(config)
    if err != nil {
        log.Panicf("Error creating consumer: %v", err)
        return
    }

    err = consumer.SubscribeTopics([]string{topics}, nil)
    if err != nil {
        log.Panicf("Error subscribe consumer: %v", err)
        return
    }

    go func() {
        for {
            msg, err := consumer.ReadMessage(-1)
            if err != nil {
                log.Printf("Consumer error: %v (%v)", err, msg)
            } else {
                fmt.Printf("Message on %s: %s\n", msg.TopicPartition, string(msg.Value))
            }
        }
    }()

    sigterm := make(chan os.Signal, 1)
    signal.Notify(sigterm, syscall.SIGINT, syscall.SIGTERM)
    select {
    case <-sigterm:
        log.Println("terminating: via signal")
    }
    if err = consumer.Close(); err != nil {
        log.Panicf("Error closing consumer: %v", err)
    }
}
```

The parameters in the example code are as follows. For details about how to obtain the parameter values, see **Obtaining MQS Connection Information**.

- **brokers**: MQS connection addresses and ports
- **group**: consumer group name. If the specified consumer group does not exist, the system automatically creates one.
- **topics**: names of the topics that consume messages

3.3.4 Configuring a Client in C#

Scenarios

This section describes how to connect a C# Kafka client to MQS (including Kafka client installation), and how to produce and consume messages.

Prerequisites

- You have obtained MQS connection information. For details, see **Preparations**.
- You have installed the development tool and C# development environment. For details, see **Preparations**.

Installing the Kafka Client

MQS is developed based on Kafka 1.1.0 and 2.7. View the Kafka version information in the **MQS Information** area on the **Instance Information** page on the ROMA Connect console. For details about how to use the C# open-source client, see **suggested client versions**.

Run the following command to install the C# Kafka dependency libraries:

```
dotnet add package -v 1.5.2 Confluent.Kafka
```

Producing Messages

- SASL authentication mode

  Replace the information in bold with the actual values.

```
using System;
using Confluent.Kafka;

class Producer
{
    public static void Main(string[] args)
    {
        var conf = new ProducerConfig {
            bootstrap_servers = "ip1:port1,ip2:port2,ip3:port3",
            context.load_verify_locations = "phy_ca.crt",
            sasl_mechanism = "PLAIN",
            security_protocol= "SASL_SSL",
            SaslUsername = "username",
            SaslPassword = "password",
            };

        Action<DeliveryReport<Null, string>> handler = r =>
            Console.WriteLine(!r.Error.IsError
                ? $"Delivered message to {r.TopicPartitionOffset}"
                : $"Delivery Error: {r.Error.Reason}");

        string topic = "topic_name";

        using (var p = new ProducerBuilder<Null, string>(conf).Build())
        {
            for (int i=0; i<100; ++i)
            {
                p.Produce(topic, new Message<Null, string> { Value = i.ToString() }, handler);
            }
            p.Flush(TimeSpan.FromSeconds(10));
        }
    }
}
```

  The parameters in the example code are as follows. For details about how to obtain the parameter values, see **Obtaining MQS Connection Information**.

  - **bootstrap_servers**: MQS connection addresses and ports
  - **topics**: names of the topics that produce messages
  - **SaslUsername** and **SaslPassword**: username and password used for SASL_SSL authentication
  - **context.load_verify_locations**: client certificate used for SASL_SSL authentication

- Non-SASL authentication mode

  Replace the information in bold with the actual values.

```
using System;
using Confluent.Kafka;
```

```
class Producer
{
    public static void Main(string[] args)
    {
        var conf = new ProducerConfig {
            bootstrap_servers = "ip1:port1,ip2:port2,ip3:port3",
            };

        Action<DeliveryReport<Null, string>> handler = r =>
            Console.WriteLine(!r.Error.IsError
                ? $"Delivered message to {r.TopicPartitionOffset}"
                : $"Delivery Error: {r.Error.Reason}");

        string topic = "topic_name";

        using (var p = new ProducerBuilder<Null, string>(conf).Build())
        {
            for (int i=0; i<100; ++i)
            {
                p.Produce(topic, new Message<Null, string> { Value = i.ToString() }, handler);
            }
            p.Flush(TimeSpan.FromSeconds(10));
        }
    }
}
```

The parameters in the example code are as follows. For details about how to obtain the parameter values, see **Obtaining MQS Connection Information**.

- **bootstrap_servers**: MQS connection addresses and ports
- **topics**: names of the topics that produce messages

Consuming Messages

- SASL authentication mode

  Replace the information in bold with the actual values.

```
using System;
using System.Threading;
using Confluent.Kafka;

class Consumer
{
    public static void Main(string[] args)
    {
        var conf = new ConsumerConfig {
            GroupId = "group_id",
            BootstrapServers = "ip1:port1,ip2:port2,ip3:port3",
            SslCaLocation = "phy_ca.crt",
            SaslMechanism = "PLAIN",
            SecurityProtocol = SASL_SSL,
            SaslUsername = "username",
            SaslPassword = "password",
            AutoOffsetReset = "earliest"
        };

        string topic = "topic_name";

        using (var c = new ConsumerBuilder<Ignore, string>(conf).Build())
        {
            c.Subscribe(topic);

            CancellationTokenSource cts = new CancellationTokenSource();
            Console.CancelKeyPress += (_, e) => {
                e.Cancel = true;
                cts.Cancel();
            };
            try
            {
                while (true)
                {
                    try
                    {
                        var cr = c.Consume(cts.Token);
                        Console.WriteLine($"Consumed message '{cr.Value}' at: '{cr.TopicPartitionOffset}'.");
                    }
                    catch (ConsumeException e)
                    {
                        Console.WriteLine($"Error occured: {e.Error.Reason}");
                    }
                }
            }
            catch (OperationCanceledException)
            {
                c.Close();
            }
        }
    }
}
```

The parameters in the example code are as follows. For details about how to obtain the parameter values, see **Obtaining MQS Connection Information**.

- **BootstrapServers**: MQS connection addresses and ports
- **GroupId**: consumer group name If the specified consumer group does not exist, the system automatically creates one.
- **topics**: names of the topics that consume messages
- **SaslUsername** and **SaslPassword**: username and password used for SASL_SSL authentication
- **SslCaLocation**: client certificate used for SASL_SSL authentication

- Non-SASL authentication mode

  Replace the information in bold with the actual values.

```
using System;
using System.Threading;
using Confluent.Kafka;

class Consumer
{
    public static void Main(string[] args)
    {
        var conf = new ConsumerConfig {
            GroupId = "group_id",
            BootstrapServers = "ip1:port1,ip2:port2,ip3:port3",
            AutoOffsetReset = "earliest"
        };

        string topic = "topic_name";

        using (var c = new ConsumerBuilder<Ignore, string>(conf).Build())
        {
            c.Subscribe(topic);

            CancellationTokenSource cts = new CancellationTokenSource();
            Console.CancelKeyPress += (_, e) => {
                e.Cancel = true;
                cts.Cancel();
            };

            try
            {
                while (true)
                {
                    try
                    {
                        var cr = c.Consume(cts.Token);
                        Console.WriteLine($"Consumed message '{cr.Value}' at: '{cr.TopicPartitionOffset}'.");
                    }
                    catch (ConsumeException e)
                    {
                        Console.WriteLine($"Error occured: {e.Error.Reason}");
                    }
                }
            }
            catch (OperationCanceledException)
            {
                c.Close();
            }
        }
    }
}
```

The parameters in the example code are as follows. For details about how to obtain the parameter values, see **Obtaining MQS Connection Information**.

- **BootstrapServers**: MQS connection addresses and ports
- **GroupId**: consumer group name If the specified consumer group does not exist, the system automatically creates one.
- **topics**: names of the topics that consume messages

3.3.5 Configuring Kafka Clients in Other Languages

MQS is fully compatible with Kafka open-source clients.

You can obtain clients in other programming languages and access your instance as instructed by the official Kafka website.

For details about how to get the client address, see the **Kafka official website**.

3.3.6 Appendix: Methods for Improving the Message Processing Efficiency

The reliability in sending and retrieving messages is the result of joint efforts from ROMA Connect, message producers, and message consumers. The following lists the best practices for ROMA Connect producers and consumers.

Optimizing the Acknowledgment Process of Message Production and Consumption

Message Production

The producer decides whether to re-send the message based on the ROMA Connect response.

Each time the producer sends a message, it waits for an API response to confirm that the message is successfully sent. If an exception occurs when sending the message, the producer will not receive a success response and must decide whether to re-send the message. If a success response is received, it indicates that the message has been stored in ROMA Connect.

Message Consumption

The consumer acknowledges successful message retrieval.

Messages are stored in ROMA Connect in the order that they are created. During message retrieval, the consumer obtains messages stored in ROMA Connect in the order that they are stored. After the consumer retrieves the messages, the message retrieval status is recorded as successful or failed. The status is then submitted to ROMA Connect. Based on the retrieval status, ROMA Connect determines whether to retrieve the next batch of messages or retrieve the messages that failed to be retrieved.

During this process, the message retrieval status may not be successfully submitted due to an exception. In this case, the corresponding messages will be re-obtained by the consumer in the next message retrieval request.

Idempotent Transferring of Message Production and Consumption

ROMA Connect provides a series of reliability measures to ensure that messages are not lost. For example, the message synchronization storage mechanism is used to prevent the system and server from being abnormally restarted or powered off. The ACK mechanism is used to solve the exceptions that occur during message transmission.

Considering the extreme conditions such as network exceptions, you need to use ROMA Connect to design message sending and consumption in addition to confirming message production and consumption.

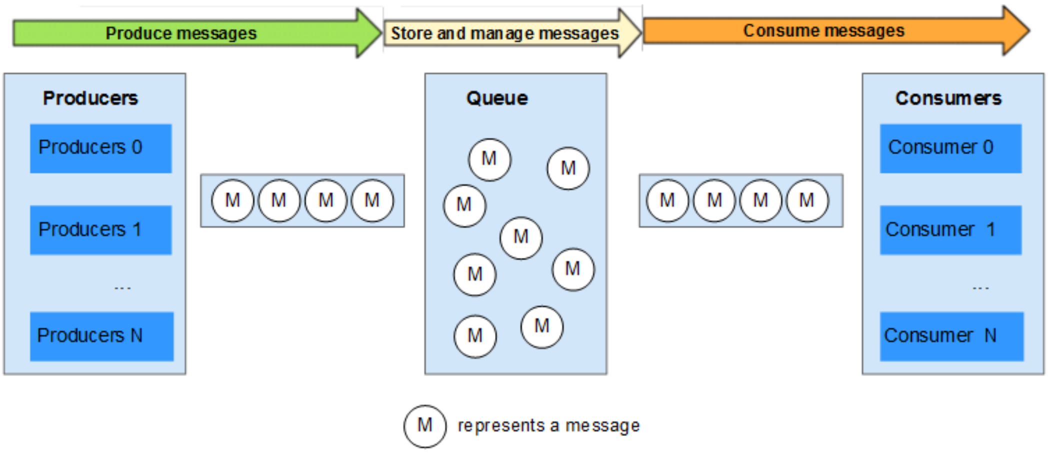
- If the message sending cannot be confirmed, the producer needs to send the message to ROMA Connect repeatedly.
- After consuming a message that has been processed, the consumer needs to notify that ROMA Connect consumption is successful and ensure that the message is not processed repeatedly.

Producing and Consuming Messages in Batches

To improve the message sending and consumption efficiency, consumers are advised to use the batch message sending and consumption mode. Generally, messages are consumed in batches by default, and messages are sent in batches if possible, which effectively reduces the number of API calls.

Refer to the following two figures.

Figure 3-1 Producing and consuming messages in batches

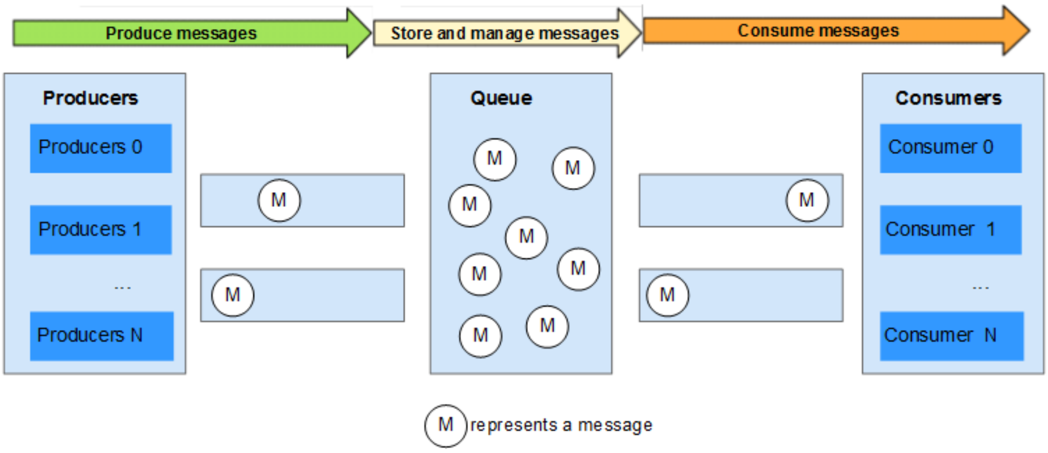


NOTICE

A maximum of 10 messages can be sent in batches. The total size cannot exceed 512 KB.

Message production (sending) in batches can be flexibly used. When there are a large number of concurrent messages, the messages are sent in batches. When the number of concurrent messages is small, the messages are sent one by one. This is done to reduce the number of API calls and ensure real-time message sending.

Figure 3-2 Producing and consuming messages one by one



When consuming messages in batches, consumers need to process and confirm messages in the sequence of receiving messages. Therefore, when a message in the batch fails to be consumed, the consumer does not need to consume the rest messages, and directly submit consumption confirmations of the successfully consumed messages to ROMA Connect.

Using Consumer Groups to Assist O&M

You can use ROMA Connect as a message management system. Reading message content from queues is helpful to fault locating and service debugging.

When problems occur during message production and consumption, you can create different consumer groups to locate and analyze problems or debug services for interconnecting with other services. To ensure that other services can continue to process messages in topics, you can create a consumer group to retrieve and analyze the messages.

3.3.7 Appendix: Restrictions on Spring Kafka Interconnection

Overview

Spring Kafka is compatible with open-source Kafka clients. For details about the version mapping between Spring Kafka and open-source Kafka clients, see the **Spring official website**. Spring Kafka is mainly compatible with Kafka client 2.x.x, whereas the Kafka server version used by ROMA Connect MQS is 1.1.0 or 2.7.

Therefore, when Spring Kafka is used to connect to ROMA Connect in the Spring Boot project, ensure that the Kafka client version is the same as the Kafka server version.

If the ROMA Connect instance connected to Spring Kafka uses Kafka 1.1.0, most functions can be used, and only a few new functions are not supported. If you encounter problems not listed in the following, submit a service ticket to contact technical support. The following lists the functions that are not supported:

Unsupported zstd Compression Type

Kafka 2.1.0 supports the zstd compression type, but the Kafka server in version 1.1.0 does not support.

- Configuration file:

  src/main/resources/application.yml

- Configuration item:

```
spring:
  kafka:
    producer:
      compression-type: xxx
```

- Restriction:

  Do not set **compression-type** to **zstd**.

Static Members Not Supported for Consumers

The parameter **group.instance.id** is added to the Kafka client in version 2.3. Consumers with this ID are considered as static members.

- Configuration file:

  src/main/resources/application.yml

- Configuration item:

```
spring:
  kafka:
    consumer:
      properties:
        group.instance.id: xxx
```

- Restriction:

  Do not add the **group.instance.id** parameter.

3.4 Configuring MQS Connection (RESTful API)

3.4.1 Using Java Demo

Scenarios

In addition to the native Kafka client described in the preceding sections, MQS instances can also be accessed via HTTP RESTful connections, including sending messages to specified topics, consuming messages, and acknowledging message consumption.

This is used to adapt to the original service system architecture and facilitate unified access using the HTTP protocol.

Procedure

1. You have obtained MQS connection information. For details, see **Preparations**.

   NOTE

   - If both SASL_SSL and intra-VPC plaintext access are enabled for MQS of the ROMA Connect instance, the SASL mode cannot be used for connecting to MQS topics in the VPC.
   - If the SASL mode is used for connecting to MQS topics, you are advised to configure the mapping between the host and IP address in the **/etc/hosts** file on the host where the client is located. Otherwise, network delay will occur.

     Set the IP address to the connection address of MQS and set the host to the name of each instance host. Ensure that the name of each host is unique. Example:

     10.10.10.11 host01

     10.10.10.12 host02

     10.10.10.13 host03

2. Assemble an API request, including the signature of the API request, by referring to the sample code.

   API request signature: The SASL username and password are used as a key pair to sign the request URL and message header timestamp for backend service verification. **Learn about the signature process.**

3. For details about the structure of response messages returned when a demo project is used to create, retrieve, and confirm messages in a specified topic, see **Message Production API**, **Message Consumption API**, and **Message Retrieval Confirmation API**.

Prerequisites

This section provides an example of sending RESTful API requests in Java. A Maven project developed in IntelliJ IDEA is used as an example. If you want to use the project in the local environment, the following environments (Windows 10 as an example) should be installed and configured:

- You have obtained MQS connection information. For details, see **Preparations**.
- You have installed the development tool and Java development environment. For details, see **Preparations**.
- You have obtained the demo.

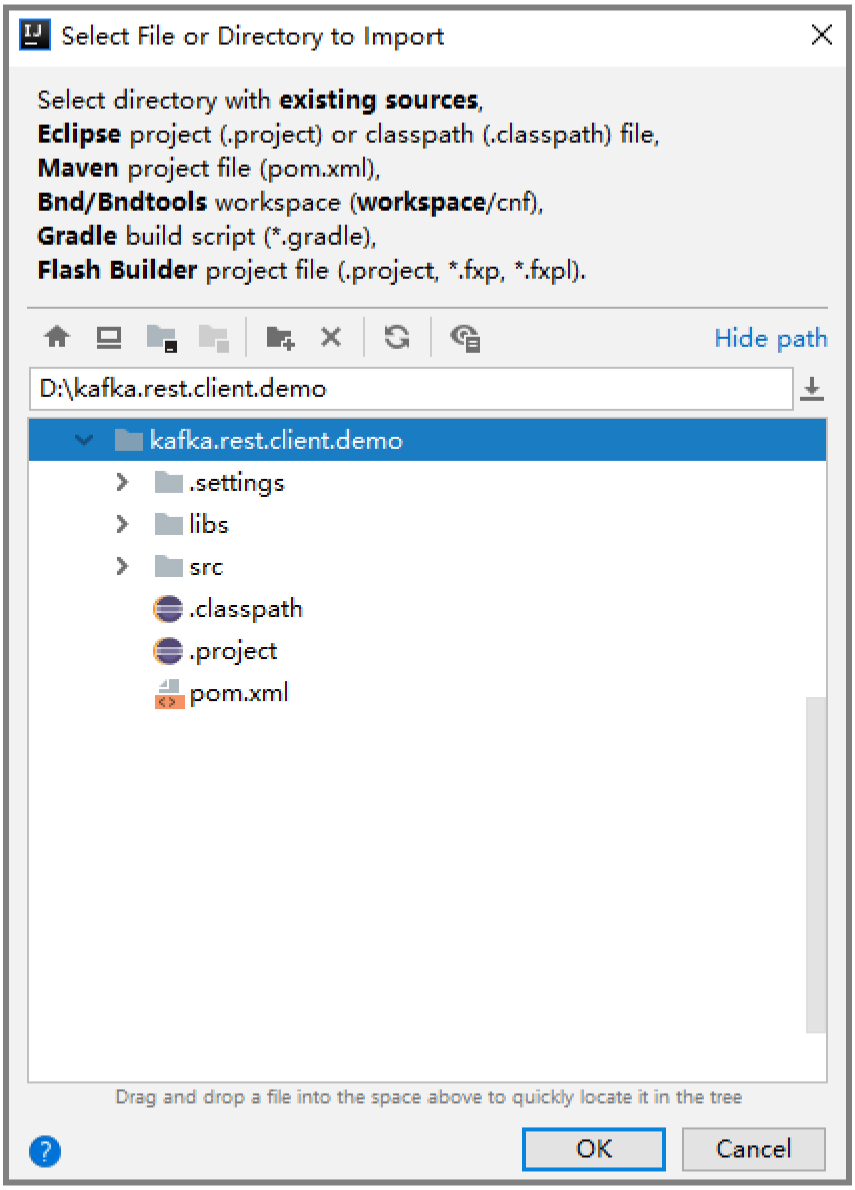
  On the ROMA Connect console, choose **Message Queue Service** > **Topic Management**. In the upper right corner of the page, choose **Download** > **Download RESTful API Java Demo Package** to download the **demo**.

Importing a Project

1. Start IntelliJ IDEA and choose **Import Project**.

   The **Select File or Directory to Import** dialog box is displayed.

2. Select the directory where the RESTful API Java demo is decompressed and click **OK**.



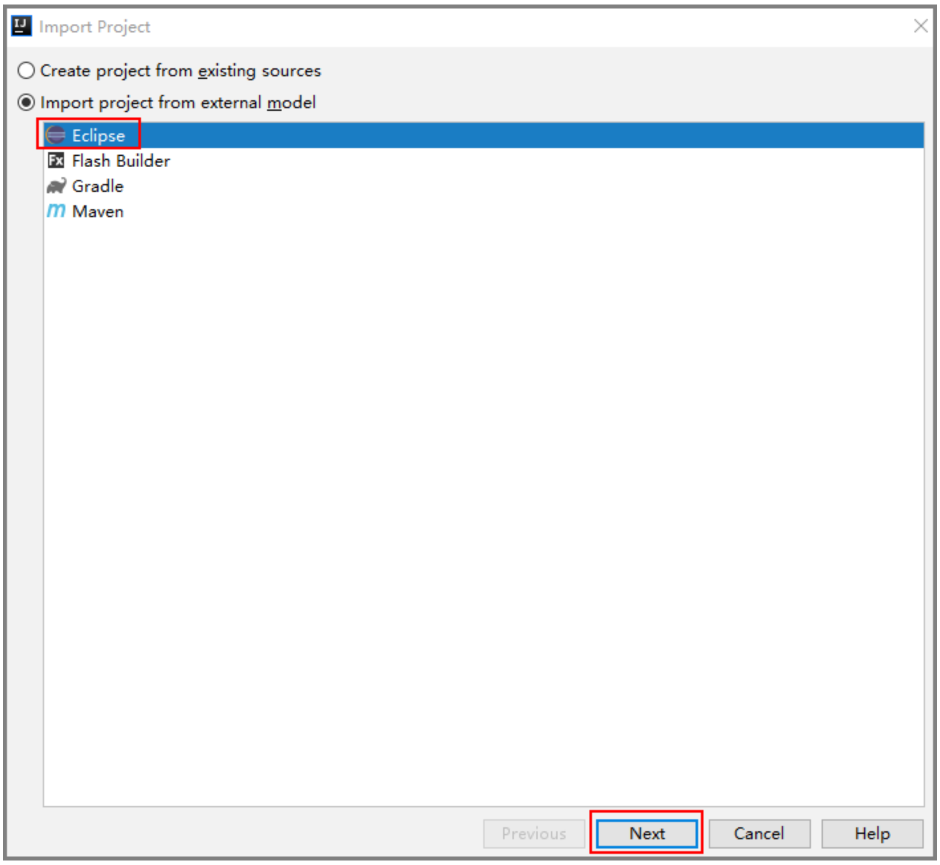
Select File or Directory to Import

Select directory with **existing sources**,
Eclipse project (.project) or classpath (.classpath) file,
Maven project file (pom.xml),
Bnd/Bndtools workspace (**workspace**/cnf),
Gradle build script (*.gradle),
Flash Builder project file (.project, *.fxp, *.fxpl).

D:\kafka.rest.client.demo

3. Select **Eclipse** for **Import project from external model** and click **Next**. Retain the default settings and click **Next** until the **Please select project SDK** page is displayed.

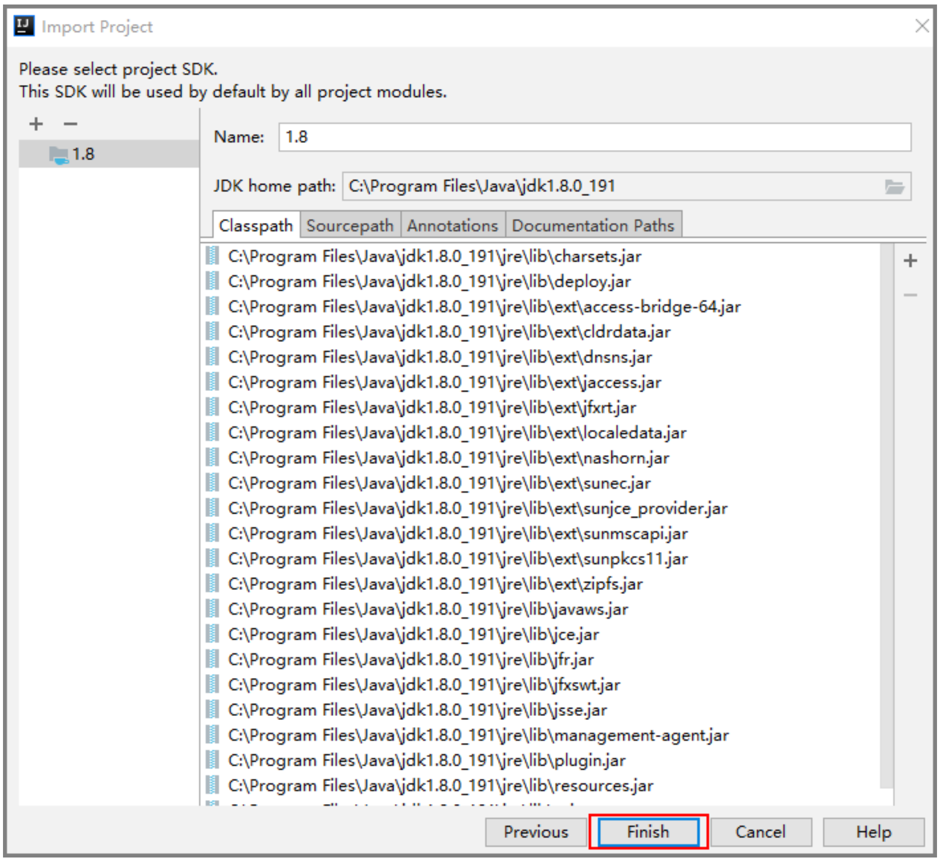
Figure 3-3 Import Project dialog box



Import Project
○ Create project from existing sources
● Import project from external model
Eclipse
Flash Builder
Gradle
Maven
Previous
Next
Cancel
Help

4. Click **Finish**.

Figure 3-4 Finish



Please select project SDK.
This SDK will be used by default by all project modules.

1.8

Name: 1.8

JDK home path: C:\Program Files\Java\jdk1.8.0_191

```
C:\Program Files\Java\jdk1.8.0_191\jre\lib\charsets.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\deploy.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\access-bridge-64.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\cldrdata.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\dnsns.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\jaccess.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\jfxrt.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\localedata.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\nashorn.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\sunec.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\sunjce_provider.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\sunmscapi.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\sunpkcs11.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\zipfs.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\javaws.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\jce.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\jfr.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\jfxswt.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\jsse.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\management-agent.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\plugin.jar
C:\Program Files\Java\jdk1.8.0_191\jre\lib\resources.jar
```

5. Edit the **rest-config.properties** file.

   The file is located in the **src/main/resources** directory. Enter the obtained Kafka instance connection address, topic name, and SASL information in the following configuration. **kafka.rest.group** indicates the consumer group ID, which can be specified on the client.

   ```
   # Kafka rest endpoint.
   kafka.rest.endpoint=https://{MQS_Instance_IP_Addr}:9292
   # Kafka topic name.
   kafka.rest.topic=topic_name_demo
   # Kafka consume group.
   kafka.rest.group=group_id_demo
   # Kafka sasl username.
   kafka.rest.username=sasl_username_demo
   # Kafka sasl password.
   kafka.rest.password=sasl_user_passwd_demo
   ```

6. Edit log4j.properties.

   For example, modify the directory for storing logs.

   ```
   log.directory=D://workspace/logs
   ```

7. Run the sample project to view the message production and consumption examples.

   The main method for producing and consuming messages is in the **RestMain.java** file. You can run the main method in Java Application mode.

Code of the Sample Project

- Project entry

  The project entry is in the **RestMain.java** file.

```
public class RestMain
{
    private static final Logger LOGGER = LoggerFactory.getLogger(RestMain.class);

    public static void main(String[] args) throws InterruptedException
    {
        //Initialize the request object. The RestServiceImpl class file also contains the RESTful APIs and
request signature.
        IRestService restService = new RestServiceImpl();
        Base64.Decoder decoder = Base64.getDecoder();
        //The following are message production, message consumption, and consumption confirmation.
        // Produce message
        ProduceReq messages = new ProduceReq();
        messages.addMessage("{[{'id': '00001', 'name': 'John'}, {'id': '00002', 'name':
'Mike'}]}").addMessage("Kafka rest client demo!");
        LOGGER.debug("produce message: {}", JsonUtils.convertObject2Str(messages));
        restService.produce(messages);

        // Consume message
        List<ConsumeResp> consumeResps = restService.consume();
        CommitReq commitReq = new CommitReq();
        consumeResps.forEach(resp ->
        {
            LOGGER.debug("handler: {}, content: {}", resp.getHandler(), new
String(decoder.decode(resp.getMessage().getContent())));
            commitReq.addCommit(resp.getHandler());
        });

        // Commit message
        if (commitReq.getMessages().size() != 0)
        {
            CommitResp resp = restService.commit(commitReq);
            LOGGER.info("Commit resp: success: {}, failed: {}", resp.getSuccess(), resp.getFail());
        }
        else
        {
            LOGGER.warn("Commit is empty.");
        }
    }
}
```

- Message assembling and sending

  The following uses message production as an example to describe how to assemble and sign a message. After the signature method is invoked, two message headers are returned: **Authorization** and **X-Sdk-Date**. **Authorization** contains signature information of the requested content. Another parameter **Content-Type** in the message header must be added to the code. For details, see the createRequest() method in the example.

```
 public List<ProduceResp> produce(ProduceReq messages)
    {
        List<ProduceResp> prodResp = null;
        try
        {
            Request request = createRequest();
            request.setUrl(produceURI);
            request.setMethod("POST");
            request.setBody(JsonUtils.convertObject2Str(messages));
            //After the request content is signed, two parameters are added to the request header:
Authorization and X-Sdk-Date. Authorization contains signature information of the requested
content.
            HttpRequestBase signedRequest = Client.sign(request);
            LOGGER.debug("Request uri: {}, headers: {}", signedRequest.getURI(),
signedRequest.getAllHeaders());
            LOGGER.debug("Request body: {}", request.getBody());

            HttpResponse response = HttpUtils.execute(signedRequest);
            if (response.getStatusLine().getStatusCode() == HttpStatus.SC_CREATED)
            {
                String jsonStr = EntityUtils.toString(response.getEntity(), "UTF-8");
                prodResp = JsonUtils.convertStr2ListObject(jsonStr, new 
TypeReference<List<ProduceResp>>() { });
                LOGGER.info("Produce response: {}", jsonStr);
                return prodResp;
            }
            else
            {
                LOGGER.error("Produce message failed. statusCode: {}, error msg: {}",
                        response.getStatusLine().getStatusCode(),
                        EntityUtils.toString(response.getEntity(), "UTF-8"));
            }
        }
        catch (Exception e)
        {
            LOGGER.error("Produce message failed.");
        }
        return prodResp;
    }
```

3.4.2 Message Production API

Function

This API is used to send messages to a queue. Multiple messages can be sent at a time. The following requirements must be met:

- A maximum of 10 messages can be sent at a time.
- The aggregated size of messages sent at a time cannot exceed 2 MB.
- The endpoint is https://{rest_connect_address}:9292. You can query the value of rest_connect_address through a specified instance interface.

URI

POST /v1/topic/{topic_name}/messages

Table 3-3 Parameter description

Parameter	Type	Mandatory	Description
topic_name	String	Yes	Topic name.

Request

Request parameter

Parameter	Type	Mandatory	Description
messages	Array	Yes	Message list. The array size cannot exceed 10 and cannot be null.

Table 3-4 Parameter description of messages

Parameter	Type	Mandatory	Description
content	Object	Yes	Message content.
id	String	Yes	Message sequence number, which must be unique.

Example request

```
{
  "messages": [
    {
      "content": "hello roma-1",
      "id": "1"
    },
    {
      "content": "hello roma-2",
      "id": "2"
    },
    {
      "content": "hello roma-3",
      "id": "3"
    }
  ]
}
```

Response

Response parameter

Parameter	Type	Description
state	String	Result status. The value can be **success** or **fail**.
id	String	Message sequence number.

Example response

```
[
  {
    "state": "success",
    "id": "1"
  },
  {
    "state": "success",
    "id": "2"
  },
  {
    "state": "success",
    "id": "3"
  }
]
```

3.4.3 Message Consumption API

Function

This API is used to consume messages in a specified queue. Multiple messages can be consumed at the same time.

- When there are only a few messages in a queue, the number of messages actually consumed at a time may be less than the message quantity specified in the consumption request. However, all messages in the queue will be eventually obtained by the message consumer after multiple rounds of consumption. If the returned message is an empty array, no message is consumed.
- The endpoint is https://{rest_connect_address}:9292. You can query the value of rest_connect_address through a specified instance interface.

URI

GET /v1/topic/{topic_name}/group/{group_name}/messages?ack_wait={ack_wait}&time_wait={time_wait}&max_msgs={max_msgs}

Table 3-5 Parameter description

Parameter	Type	Mandatory	Description
topic_name	String	Yes	Topic name.
group_name	String	Yes	Consumer group name. The value can contain a maximum of 249 characters, including letters, digits, hyphens (-), and underscores (_).
ack_wait	Integer	No	Timeout duration that the API call can wait for message consumption acknowledgement. The client needs to submit the message consumption acknowledgement within the specified time. If the message consumption is not acknowledged within this period of time, the system displays a message, indicating that message consumption acknowledgement has timed out or the handler is invalid. In this case, the system determines that the message fails to be consumed by default. Value range: 1–300s Default value: **15s**

Parameter	Type	Mandatory	Description
time_wait	Integer	No	Amount of time that the API call can wait for a message to arrive in the empty queue before returning an empty response. If a message is available during the wait period, the message consumption result is returned immediately. If no dead letter message is available until the wait period expires, an empty response will be returned after the wait period ends. Value range: 1–30s Default value: **3s**
max_msgs	Integer	No	Number of consumable messages that can be obtained per time. Value range: 1–10 Default value: **10**
max_bytes	Integer	No	Maximum message load that can be consumed each time. Value range: 1–2097152 Default value: **524288**

Request

Request parameters

None.

Example request

None.

Response

Response parameters

Parameter	Type	Description
handler	String	Message handler.
message	Object	Message content.

Table 3-6 Parameter description of message

Parameter	Type	Description
content	String	Message body content, which is encrypted using Base64.

Example response

```
[
  {
    "handler": "NCMxMDAjMTgjMA==",
    "message": {
      "content": "ImhlbGxvIGh1YXdlaWNsb3VkLTIi"
    }
  }
]
```

3.4.4 Message Retrieval Confirmation API

Function

This API is used to acknowledge consumption of specified messages.

- When a message is being consumed, it remains in the queue. It cannot be consumed again by the same consumer group within 30s since the start of the consumption. If consumption is not acknowledged within this period, MQS determines that this message fails to be consumed, and this message can be consumed again.
- The endpoint is https://{rest_connect_address}:9292. You can query the value of rest_connect_address through a specified instance interface.

URI

POST /v1/topic/{topic_name}/group/{group_name}/messages

Table 3-7 Parameter description

Parameter	Type	Mandatory	Description
topic_name	String	Yes	Topic name.
group_name	String	Yes	Consumer group name.

Request

Request parameter

Parameter	Type	Mandatory	Description
messages	Array	Yes	Message list. The array size cannot exceed 10 and cannot be null.

Table 3-8 Parameter description

Parameter	Type	Mandatory	Description
handler	String	Yes	Message handler.
status	String	Yes	Consumption status. The value can only be **success** or **fail**.

Example request

```
{
  "messages": [
    {
      "handler": "NCMxMDAjMTgjMA==",
      "status": "success"
    }
  ]
}
```

Response

Response parameter

Parameter	Type	Description
success	Integer	Number of consumptions that are successfully acknowledged.
fail	Integer	Number of consumptions that fail to be acknowledged.

Example response

```
{
  "success": 1,
  "fail": 0
}
```

4 Developer Guide for Device Integration

4.1 Overview

4.1.1 Scenarios

Description

LINK allows devices to access ROMA Connect and report data using MQTT.

An MQTT client needs to be developed and integrated into a device, and the access information of the device needs to be written during the integration.

After the development is complete, the device can access ROMA Connect after being powered on and connected to the network.

4.1.2 Specifications

- **Development tool versions**:
  - IntelliJ IDEA: 2018.3.5 or later
  - Eclipse: 3.6.0 or later
- **Development language versions**:

  Java: Java Development Kit 1.8.111 or later
- **Device development requirements:**
  - When MQTT devices are used for access, only QoS0 and QoS1 in MQTT are supported.
  - To prevent device disconnection due to network instability or instance upgrade, you are advised to add an automatic reconnection mechanism during device development. If the device demo provided by ROMA

Connect is used, the reconnection mechanism is enabled by default. If the open-source MQTT client is used, you need to configure the reconnection mechanism based on the open-source code. If the connection is lost after the automatic reconnection function is enabled, the client keeps automatically reconnecting to the server until the connection is successful.

4.2 Preparations

Obtaining Device Access Information.

On the ROMA Connect instance console, choose **LINK** > **Device Management** to view the MQTT/MQTTS connection address. In the device list on the **Device** tab page, view the client ID, username, and password of the device to be connected.

Preparing the Development Environment

- Install a development tool.

  Download the installation package of IntelliJ IDEA 2018.3.5 or later from the **official IntelliJ IDEA website**.

- Install a development language.

  Download the Java Development Kit of 1.8.111 or later from the **official Oracle website**.

4.3 Configuring Device Integration

Scenarios

This section describes how to configure device integration for device access as well as message sending and receiving.

NOTICE

Device integration supports the standard MQTT protocol. You can use the open-source **Eclipse paho MQTT Client** to connect to LINK. In this example, the demo uses the Java SDK.

Prerequisites

- You have obtained the device access information. For details, see **Preparations**.
- You have installed the development tool and Java development environment. For details, see **Preparations**.
- Download the **LINK Demo**.

  A demo contains two files. The **DeviceConnectDemo.java** file is used to connect to devices, and the **DeviceControlDemo.java** file is used to call APIs of control devices.

Configuring Device Connection Information

1. Decompress the demo package and find the DeviceConnectDemo.java file in the bottom-layer path of the **src** directory.
2. Use the Java editing tool to open the file and edit the device connection information. After the running is successful, you can view the status of the online device on the **Device Management** page.

   NOTE

   The software packages on which the demo project depends are stored in the **lib** directory. When using the demo, you need to set the **lib** directory of the demo to the **lib** directory of the current project.

```
        //Device connection address: tcp://ip:port
        final String host = "";
        //Device client ID
        final String clientId = "";
        // The example is used only for testing or illustration. The username for device authentication is
sensitive. Do not hardcode it.
        final String userName = "";
        // The example is used only for testing or illustration. The password for device authentication is
sensitive. Do not hardcode it.
        final String password = "";
       //Topic with the PUB permission
        final String pubTopic = "";
       //Topic with the SUB permission
        final String subTopic = "";
        //Content of the message sent by the device
        final String payload = "hello world.";
```

Sending and Receiving Messages

The **DeviceConnectDemo.java** file has preset messages of topics with the PUB permission. If you call an API for sending control messages to a device, the device can receive the message immediately.

```
 client.subscribe(subTopic, (s, mqttMessage) -> {
               String recieveMsg = "Receive message from topic:" + s + "\n";
               System.out.println(recieveMsg + new String(mqttMessage.getPayload(),
StandardCharsets.UTF_8));
           });
```

1. Call APIs for sending control messages.
   a. Use the Java editor to open the **DeviceControlDemo.java** file and change the parameters of the API for sending control messages to the created device information.

      Enter the following information: AppKey, AppSecret, device client ID, topic with the SUB permission, access address of the API for sending control messages, access port, and message content.

```
public static void main(String[] args)
    {
        // The example is used only for testing or illustration. The AppKey for API authentication is
sensitive. Do not hardcode it.
        String appKey = "";
        //The example is used only for testing or illustration. The AppSecret for API authentication
is sensitive. Do not hardcode it.
        String appSecret = "";
        //ID of the device client that needs to send control messages
        String clientId = "";
        //Topic with the SUB permission of the device that needs to send control messages
        String subTopic = "";
        //Access address of the API for sending control messages
        String host = "";
        //Access port of the API for sending control messages
        String port = "";
        //Content of the message to be sent to the device
        String payload = "hello world.";

        String url = "https://" + host + ":" + port + "/v1/devices/" + clientId;
        controlDevice(url, appKey, appSecret, clientId, subTopic, payload);
    }
```

- The values of **appKey** and **appSecret** can be obtained by clicking the name of the integration application to which the device belongs on the **Integration Applications** page of the ROMA Connect console and viewing the key and secret from basic information about the integration application.
- The port number is 7443. The values of **clientId**, **subTopic**, **host**, and **port** can be obtained by clicking the device name after you choose **LINK** > **Device Management** on the ROMA Connect console.

b. Recompile and run the DeviceControlDemo class. If the device is connected and subscribes to a topic with the SUB permission, the device immediately receives a message and prints it on the IDE console. The request IP address of the API is the same as the IP address for connecting to the device, and the port number is 7443.

2. Send messages.

You can set the content and frequency of messages to be sent by a device. For example, you can instruct a device to send a message to LINK every 10 seconds. After the code runs, LINK receives a message every 10 seconds.

```
try
        {
            final MqttClient client = new MqttClient(host, clientId);
            client.connect(mqttConnectOptions);
            System.out.println("Device connect success. client id is " + clientId + ", host is " + host);

            final MqttMessage message = new MqttMessage();
            message.setQos(1);
            message.setRetained(false);
            message.setPayload(payload.getBytes(StandardCharsets.UTF_8));

            Runnable pubTask = () -> {
                try
                {
                    client.publish(pubTopic, message);
                }
                catch (MqttException e)
                {
                    System.out.println(e.getMessage());
                }
            };

            client.subscribe(subTopic, (s, mqttMessage) -> {
                String recieveMsg = "Receive message from topic:" + s + "\n";
                System.out.println(recieveMsg + new String(mqttMessage.getPayload(),
StandardCharsets.UTF_8));
            });

            ScheduledExecutorService service = Executors
                    .newSingleThreadScheduledExecutor();
            service.scheduleAtFixedRate(pubTask, 0, 10, TimeUnit.SECONDS);
        }
```

NOTE

The Connect code simulates the function of connecting the MQTT.fx client to the device. After the connection is successful, the device displays "Connected."

4.4 MQTT Topic Specifications

4.4.1 Before You Start

- When the IoT platform functions as the message subscriber, it has subscribed to related topics by default. The IoT platform can receive messages sent by devices to the corresponding topics.
- When a device functions as a message subscriber, the device needs to subscribe to related topics first so that the device can receive messages sent by the IoT platform to the corresponding topics. The device determines the topics to be subscribed to based on the specific business requirements.

Topic	Supported Protocol	Publisher	Subscriber	Function
/v1/devices/{gatewayId}/topo/add	MQTT	Edge device	IoT platform	The edge device adds a subdevice.
/v1/devices/{gatewayId}/topo/addResponse		IoT platform	Edge device	The IoT platform returns a response for adding a subdevice.
/v1/devices/{gatewayId}/topo/update		Edge device	IoT platform	The edge device updates the subdevice status.
/v1/devices/{gatewayId}/topo/updateResponse		IoT platform	Edge device	The IoT platform returns a response for updating the subdevice status.
/v1/devices/{gatewayId}/topo/delete		IoT platform	Edge device	The IoT platform deletes a subdevice.
/v1/devices/{gatewayId}/topo/query		Edge device	IoT platform	The edge device queries gateway information.

Topic	Supported Protocol	Publisher	Subscriber	Function
/v1/devices/{gatewayId}/topo/queryResponse		IoT platform	Edge device	The IoT platform returns a response for querying gateway information.
/v1/devices/{gatewayId}/command		IoT platform	Edge device	The IoT platform delivers a command to a device or an edge device.
/v1/devices/{gatewayId}/commandResponse		Edge device	IoT platform	The edge device returns a command response to the IoT platform.
/v1/devices/{gatewayId}/datas		Edge device	IoT platform	The edge device reports data.

NOTE

In the preceding table, {*gatewayId*} indicates the device ID. Specifications of previous sites are inherited by delete, while specifications are not provided for deleteResponse now.

4.4.2 Gateway Login

The IoT platform supports ROMA Connect's message API using the MQTT protocol to obtain the authentication information **clientId**, **Username**, and **Password**.

Parameter Description

Parameter	Mandatory/Optional	Type	Description
clientId	Mandatory	String(256)	The value of this parameter consists of the device or node ID, authentication type, password signature type, and timestamp, which are separated by underscores (_). ● The authentication type is 1 byte long and can be set to one of the following values – **0**: The device ID, which is unique for each device, is used for access – **2**: The node ID, which is unique for each device, is used for access ● The signature type is 1 byte long and can be set to one of the following values: – **0**: The timestamp is not verified using the HMAC-SHA256 algorithm. – **1**: The timestamp is verified using the HMAC-SHA256 algorithm. ● The timestamp is the UTC time when the device was connected to the platform, in the format of YYYYMMDDHH. For example, if the UTC time is 2018/7/24 17:56:20, the timestamp is **2018072417**. For example, the client ID of the device ID is D39564861q3gDa_0_0_2018072417.
Username	Mandatory	String(256)	Username, which is unique for each device. ● When the device accesses the platform using deviceId, set this parameter to the value of **deviceId** used when the device is registered successfully. ● When the device accesses the platform using nodeId, set this parameter to the value of **nodeId** used when the device is registered successfully.
Password	Mandatory	String(256)	The value of this parameter is the value of the device secret encrypted by using the HMAC-SHA256 algorithm with the timestamp as the key. The value of this parameter is the secret returned by the platform during device registration or is the secret of the device.

4.4.3 Adding a Gateway Subdevice

Topic

Topic	/v1/devices/{gatewayId}/topo/add
Publisher	Edge device
Subscriber	IoT platform

Parameter Description

Field	Mandatory/Optional	Type	Description
mid	Mandatory	Integer	Command ID.
deviceInfos	Mandatory	List<DeviceInfos>	Subdevice information list. The list contains 1 to 100 records.

DeviceInfos struct description

Field	Mandatory/Optional	Type	Description
nodeId	Mandatory	String	Device identifier. The value must contain 2 to 64 characters and can consist of only uppercase and lowercase letters, digits, and hyphens (-).
name	Optional	String	Device name. The value must contain 2 to 64 characters and can consist of only letters, digits, hyphens (-), and number signs (#).
description	Optional	String	Device description. The value length cannot exceed 200 characters.

Field	Mandatory/ Optional	Type	Description
manufacturerId	Mandatory	String	Manufacturer ID. The value must contain 2 to 50 characters and can consist of only uppercase and lowercase letters, digits, hyphens (-), and underscores (_).
model	Mandatory	String	Product model. The value must contain 2 to 50 characters and can consist of only uppercase and lowercase letters, digits, hyphens (-), and underscores (_).

Example

```
{
      "deviceInfos": [{
            "manufacturerId": "Test_n",
            "model": "A_n",
            "nodeId": "n-device"
      }],
      "mid": 7
}
```

4.4.4 Response for Adding a Gateway Subdevice

Topic

Topic	/v1/devices/{gatewayId}/topo/addResponse
Publisher	IoT platform
Subscriber	Edge device

Parameter Description

After a subdevice is added successfully, a response containing information about the new subdevice is returned. During secondary development, information about the new subdevice needs to be saved locally. The returned **deviceId** field is used for reporting subdevice data, updating the subdevice status, and deleting the subdevice.

Response parameter description

Field	Mandatory/ Optional	Type	Description
mid	Mandatory	Integer	Command ID.
statusCode	Mandatory	Integer	Result code for request processing. The options are as follows: ● **0**: success ● non-0: failure
statusDesc	Optional	String	Response status description.
data	Mandatory	List< AddDeviceRsp >	Information about the added subdevice.

AddDeviceRsp struct description

Field	Mandatory/ Optional	Type	Description
statusCode	Mandatory	Integer	Result code for request processing. The options are as follows: ● **0**: success ● non-0: failure
statusDesc	Optional	String	Response status description.
deviceInfo	Optional	DeviceInfo	Device information.

DeviceInfo struct description

Field	Mandatory/ Optional	Type	Description
nodeId	Mandatory	String	Device identifier. The value must contain 2 to 64 characters and can consist of only uppercase and lowercase letters, digits, and hyphens (-).

Field	Mandatory/ Optional	Type	Description
deviceId	Mandatory	String	Unique device ID generated by the IoT platform, which corresponds to the device client ID.
name	Mandatory	String	Device name. The value must contain 2 to 64 characters and can consist of only uppercase and lowercase letters, digits, hyphens (-), and number signs (#).
description	Optional	String	Device description. The value length cannot exceed 200 characters.
manufacturerId	Mandatory	String	Manufacturer ID. The value must contain 2 to 50 characters and can consist of only uppercase and lowercase letters, digits, hyphens (-), and underscores (_).
model	Mandatory	String	Product model. The value must contain 2 to 50 characters and can consist of only uppercase and lowercase letters, digits, hyphens (-), and underscores (_).

Example

```
{
      "data": [{
            "deviceInfo": {
                  "manufacturerId": "Test_n",
                  "name": "n-device",
                  "model": "A_n",
                  "nodeId": "n-device",
                  "deviceId": "D59eGSxy"
            },
            "statusCode": 0
      }],
      "mid": 7,
      "statusCode": 0
}
```

4.4.5 Updating the Gateway Subdevice Status

Topic

Topic	/v1/devices/{gatewayId}/topo/update
Publisher	Edge device
Subscriber	IoT platform

Parameter Description

Field	Mandatory/ Optional	Type	Description
mid	Mandatory	Integer	Command ID.
deviceStatuses	Mandatory	List<DeviceStatus>	Device status list. The list contains 1 to 100 records.

deviceStatus struct description

Field	Mandatory/ Optional	Type	Description
deviceId	Mandatory	String	Unique device ID generated by the IoT platform, which corresponds to the device client ID.
status	Mandatory	String	Subdevice status. The options are as follows: ● **OFFLINE** ● **ONLINE**

Example

```
{
	"deviceStatuses": [{
		"deviceId": "D59eGSxy",
		"status": "ONLINE"
	}],
	"mid": 9
}
```

4.4.6 Response for Updating the Gateway Subdevice Status

Topic

Topic	/v1/devices/{gatewayId}/topo/updateResponse
Publisher	IoT platform
Subscriber	Edge device

Parameter Description

Field	Mandatory/Optional	Type	Description
mid	Mandatory	Integer	Command ID.
statusCode	Mandatory	Integer	Result code for request processing. The options are as follows: • **0**: success • non-0: failure
statusDesc	Optional	String	Response status description.
data	Optional	List< UpdateStatusRsp >	Device status information after being updated.

UpdateStatusRsp struct description

Field	Mandatory/Optional	Type	Description
statusCode	Mandatory	Integer	Result code for request processing. The options are as follows: • **0**: success • non-0: failure
statusDesc	Optional	String	Result description.

Field	Mandatory/ Optional	Type	Description
deviceId	Mandatory	String	Unique device ID generated by the IoT platform, which corresponds to the device client ID.

Example

```
{
      "data": [{
            "deviceId": "D59eGSxy",
            "statusCode": 0
      }],
      "mid": 9,
      "statusCode": 0
}
```

4.4.7 Deleting a Gateway Subdevice

Topic

Topic	/v1/devices/{gatewayId}/topo/delete
Publisher	IoT platform
Subscriber	Edge device

Parameter Description

Field	Mandatory/ Optional	Type	Description
id	Mandatory	Integer	ID of the command for deleting a subdevice.
deviceId	Mandatory	String	Unique device ID generated by the IoT platform, which corresponds to the device client ID.
requestTime	Mandatory	Timestamp	Request timestamp.
request	Mandatory	JsonObject	Subdevice information.

JsonObject struct description

Field	Mandatory/ Optional	Type	Description
manufacturerName	Mandatory	String	Manufacturer name. The value must contain 2 to 64 characters.
manufacturerId	Mandatory	String	Manufacturer ID. The value must contain 2 to 50 characters and can consist of only uppercase and lowercase letters, digits, hyphens (-), and underscores (_).
model	Mandatory	String	Product model. The value must contain 2 to 50 characters and can consist of only uppercase and lowercase letters, digits, hyphens (-), and underscores (_).

Example

```
{
	"requestTime": 1576639584536,
	"request": {
		"manufacturerName": "ATest_n",
		"manufacturerId": "Test_n",
		"model": "A_n"
	},
	"id": 8,
	"deviceId": "n-device"
}
```

4.4.8 Querying Gateway Information

Topic

Topic	/v1/devices/{gatewayId}/topo/query
Publisher	Edge device
Subscriber	IoT platform

Parameter Description

Field	Mandatory/ Optional	Type	Description
mid	Mandatory	Integer	Command ID.
nodeId	Mandatory	String	Device identifier. The value must contain 2 to 64 characters and can consist of only uppercase and lowercase letters, digits, and hyphens (-).

Example

```
{
    "mid": 2,
    "nodeId": "test123"
}
```

4.4.9 Response for Querying Gateway Information

Topic

Topic	/v1/devices/{gatewayId}/queryResponse
Publisher	IoT platform
Subscriber	Edge device

Parameter Description

Field	Mandatory/ Optional	Type	Description
mid	Mandatory	Integer	Command ID.
statusCode	Mandatory	Integer	Result code for request processing. The options are as follows: ● **0**: success ● non-0: failure
statusDesc	Optional	String	Response status description.
data	Optional	List<DeviceInfo>	Device information.

Field	Mandatory/ Optional	Type	Description
count	Optional	String	Device quantity.
marker	Optional	String	Tag.

Table 4-1 DeviceInfo structure

Field	Mandatory/ Optional	Type	Description
deviceId	Mandatory	String	Unique device ID generated by the IoT platform, which corresponds to the device client ID.
nodeId	Mandatory	String	Device identifier. The value must contain 2 to 64 characters and can consist of only uppercase and lowercase letters, digits, and hyphens (-).
name	Mandatory	String	Device name. The value must contain 2 to 64 characters and can consist of only uppercase and lowercase letters, digits, hyphens (-), and number signs (#).
description	Optional	String	Device description. The value length cannot exceed 200 characters.
manufactureId	Mandatory	String	Manufacturer ID. The value must contain 2 to 50 characters and can consist of only uppercase and lowercase letters, digits, hyphens (-), and underscores (_).
model	Mandatory	String	Product model. The value must contain 2 to 50 characters and can consist of only uppercase and lowercase letters, digits, hyphens (-), and underscores (_).

Example

```
{
    "mid": 2,
    "statusCode": 0,
    "statusDesc": "",
        "marker": "",
        "count": "1",
    "data": [
        {
            "deviceId": "D59eGSxy",
            "nodeId": "test123",
            "name": "n-device",
            "description": "addsSubDevice",
            "manufacturerId": "Test_n",
            "model": "A_n"
        }
    ]
}
```

4.4.10 Delivering a Command to a Device

Topic

Topic	/v1/devices/{gatewayId}/command
Publisher	IoT platform
Subscriber	Edge device

Parameter Description

Field	Mandatory/ Optional	Type	Description
deviceId	Mandatory	String	Unique device ID generated by the IoT platform, which corresponds to the device client ID.
msgType	Mandatory	String	This field has a fixed value of **cloudReq**, which indicates a request delivered by the IoT platform.
serviceId	Mandatory	String	Service ID.
cmd	Mandatory	String	Command name of a service.
paras	Mandatory	ObjectNode	Command parameter.
mid	Mandatory	Int	Command ID.

Example

```
{
        "msgType": "cloudReq",
        "mid": 54132,
        "cmd": "command1",
        "paras": {
                "temperature": 123
        },
        "serviceId": "service1",
        "deviceId": "D23pigXo"
}
```

4.4.11 Response for Delivering a Command to a Device

Topic

Topic	/v1/devices/{gatewayId}/commandResponse
Publisher	Edge device
Subscriber	IoT platform

Parameter Description

Field	Mandatory/Optional	Type	Description
msgType	Mandatory	String	This field has a fixed value of **deviceRsp**, which indicates a response returned by a device.
mid	Mandatory	Int	Command ID.
errcode	Mandatory	Int	Result code for request processing. The options are as follows: ● **0**: success ● non-0: failure
body	Optional	ObjectNode	Command response.

Example

```
{
        "body": {
                "orginParameters": {
                        "temperature": 123
                },
                "state": "ok"
        },
        "errcode": 0,
        "mid": 54132,
        "msgType": "deviceRsp"
}
```

4.4.12 Reporting Device Data

Topic

Topic	/v1/devices/{gatewayId}/datas
Publisher	Edge device
Subscriber	IoT platform

Parameter Description

Field	Mandatory/ Optional	Type	Description
devices	Mandatory	DeviceS[]	Device data.

DeviceS struct description

Field	Mandatory/ Optional	Type	Description
deviceId	Mandatory	String(256)	Unique device ID generated by the IoT platform, which corresponds to the device client ID.
services	Mandatory	List<Services>	Service list.

Services struct description

Field	Mandatory/ Optional	Type	Description
serviceId	Mandatory	String(256)	Service ID.
data	Mandatory	ObjectNode	Service data.

Field	Mandatory/ Optional	Type	Description
eventTime	Mandatory	String(256)	Time. The format is ***yyyyMMdd*'T'*HHmmss*'Z'**, for example, **20151212T121212Z**.

Example

```
{
        "devices": [{
                "deviceId": "D68NZxB4",
                "services": [{
                        "data": {
                                "key": "value"
                        },
                        "eventTime": "20191023T173625Z",
                        "serviceId": "serviceName"
                }]
        }]
}
```