API Request

Signing Guide

Issue	01
Date	2026-03-05

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


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

Notice

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

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

Contents

1 Overview

This document describes how to call cloud service APIs registered with API Gateway using the AK/SK signature authentication. You need to obtain the API information and AK/SK by referring to **Obtaining API Information** and **Obtaining an AK/SK** first, and then perform signature authentication based on the signature SDK and calling example of each language by referring to **Java Signing Guide**, **Go Signing Guide**, **Python Signing Guide**, **C# Signing Guide**, **JavaScript Signing Guide**, **PHP Signing Guide**, **C++ Signing Guide**, **C Signing Guide** and **Android Signing Guide**. If the preceding signing examples do not contain the programming language you use, sign requests according to the **API Signature Authentication Mechanism** and **Example of the API Signature Authentication Mechanism**.

- For the authentication of certain cloud service APIs that are not registered with API Gateway, see the *API Reference* of the corresponding service.
- For the APIs provided by a cloud service, see the *API Reference* of the cloud service. The *API Reference* contains a section named "Calling APIs" that describes API authentication methods.
- The SDK of each programming language is packaged in the sample code and can be obtained separately. You can integrate the SDK into your application by referring to the API calling example.
- AK/SK authentication supports API requests with a body less than or equal to 12 MB. For API requests with a larger body, token authentication is recommended. The description and example of token-based authentication are included in the section "Authentication" in the API reference of each cloud service.
- The local time on the client must be synchronized with the clock server to avoid a large offset in the value of the **X-Sdk-Date** request header.

  API Gateway checks the time format and compares the time with the time when API Gateway receives the request. If the time difference exceeds 15 minutes, API Gateway will reject the request.

2 Preparations

2.1 Obtaining API Information

The following information is required for API calling:

- Endpoint and URI that will constitute the request URL
- AK/SK used for signing and authentication
- Project ID and subproject ID
- Account name and account ID
- API environment
- Domain name for **Host**

Table 2-1 Required information to collect

Item	Description
Endpoint	Endpoint of a cloud service in a region. Contact the environment administrator to obtain the region and endpoint. **NOTE** For all example request URLs in this document, the endpoint **service.region.example.com** is used as an example.
URI	API request path and parameters. Obtain the request path and parameters from the *API Reference* of the cloud service.
AK/SK	Access key ID (AK) and secret access key (SK), which are used to sign API requests. For details on how to obtain the AK/SK, see **Obtaining an AK/SK**.

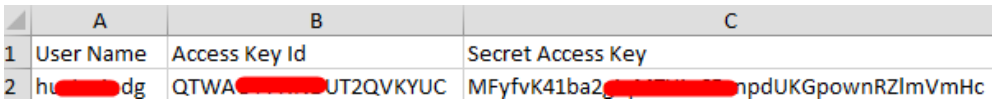
Item	Description
Project_Id	Project ID, which needs to be configured for the URI of most APIs to identify different projects. To obtain the project ID, perform the following steps: 1. Go to the **console homepage**. 2. Hover over the username in the upper right, click the username, choose **My Credentials** from the drop-down list, and then view the project ID. Projects physically isolate cloud server resources by region. Multiple projects can be created in the same region to implement more fine-grained isolation. Find the region where your server locates on the right of the page, and obtain the corresponding project ID from the list on the left.
X-Project-Id	Subproject ID, which is used in multi-project scenarios. To access resources in a subproject through AK/SK-based authentication, the **X-Project-Id** field must be added to the request header. To obtain the project ID, perform the following steps: 1. Go to the **console homepage**. 2. Hover the cursor on the username and choose **My Credentials** from the drop-down list. 3. Find the project, click the plus sign (+) on the left of the project, and obtain the subproject ID.
X-Domain-Id	Account ID, which is used to: • Obtain a token for token authentication. • Call APIs of global services through AK/SK authentication. Global services (such as IAM, OBS, and CDN) are deployed without specifying physical regions. To obtain the account ID, perform the following steps: 1. Go to the **console homepage**. 2. Hover the cursor on the username and choose **My Credentials** from the drop-down list. 3. View the **Account Name** and **Account ID**.
x-stage	For details, see the API environment information of each cloud service.
Host	Debugging domain name or independent domain name of the group to which the API belongs. For details, see the domain name information in the API group to which the API of each cloud service belongs.

2.2 Obtaining an AK/SK

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

The following figure shows an AK and SK.

Figure 2-1 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

Important Notes

1. You can create a maximum of two access keys with identical permissions and unlimited validity. **Each access key can be downloaded only once when created.** Keep your access keys secure and change them periodically for security purposes. To change an access key, delete it and create a new one.
2. Federated users cannot create access keys, but can create temporary access credentials (temporary AK/SK and security tokens). For details, see **Temporary Access Key (for Federated Users)**
3. If you are an IAM user, hover over the username in the upper right corner of the management console, choose **Security Settings**, click the **Critical Operations** tab, and check the enabling status of the **Access Key Management** feature.
   - Disabled: All IAM users under the account can manage (create, enable, disable, and delete) their own access keys.
   - Enabled: Only the administrator can manage users' access keys.
4. If you cannot manage your access keys, request the **administrator** to perform either of the following operations:
   - **Manage access keys** and send it to you.
   - Assign required permissions to you or change access key status. For details about how to assign permissions, see **Assigning Permissions to an IAM User**. For details about how to change access key status, see **Access Key Management**.
5. If you are an administrator, you can view the AK of an IAM user on the user details page. The SK is kept by the user.

Procedure

Step 1 Go to the **console homepage**.

Step 2 Hover the cursor on the username and choose **My Credentials** from the drop-down list.

Step 3 Click the **Access Keys** tab.

Step 4 Click **Create Access Key**.

- You can create a maximum of two access keys. **The quota cannot be increased**. If you already have two access keys and want to create a new one, delete one first.
- To change an access key, delete it and create a new one.
- For newly created access keys, the last used time is the same as the creation time, but will change the next time you use them.

Step 5 Enter a description, and click **OK**.

Step 6 In the displayed dialog box, click **Download** to save the access key.

You can obtain the AK from the access key list and SK from the downloaded CSV file.

- For details about how to obtain a temporary AK/SK, see the **IAM API Reference***Identity and Access Management API Reference*.
- Keep the CSV file properly. You can only download the file right after the access key is created. However, if you cannot find the file to obtain the key information, you can create a key.
- Open the CSV file in the lower left corner, or choose **Downloads** in the browser and open the CSV file.
- Keep your access keys secure and change them periodically for security purposes.

----End

3 Java Signing Guide

This section uses Eclipse as an example to describe how to integrate the Java SDK for API request signing. You can import the sample project in the code package, and integrate the signing SDK into your application by referring to the API calling example.

Preparing the Environment

- Download the **Eclipse** installation file or package, and install the tool or decompress the package.
- You have installed JDK version 8 or a later version but prior to version 17. You can download JDK from **Oracle official download page**.

Obtaining the SDK

Log in to the APIG console and choose **Help Center** > **SDK Process Flow**. Then download the SDK.

The following table shows the directory structure of the package.

Name	Description
libs\java-sdk-core-x.x.x.jar	Signing SDK and dependencies
libs\commons-codec-x.x.jar	
libs\commons-logging-x.x.jar	
libs\httpclient-x.x.x.jar	
libs\httpcore-x.x.x.jar	
src\com\apig\sdk\demo\Main.java	Sample code for signing requests
.classpath	Sample project files
.project	

Configuring Eclipse

Importing the Sample Project

Step 1 Start Eclipse and choose **File** > **Import**. In the **Import** dialog box, choose **General** > **Existing Projects into Workspace**, and select the **APIGW-java-sdk-*x.x.x*** folder.

Figure 3-1 Import

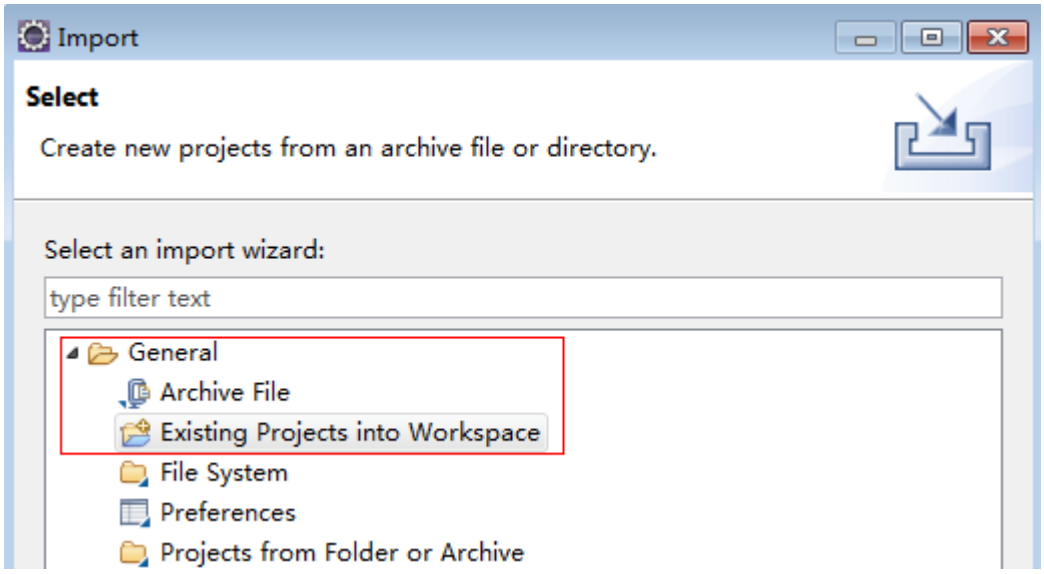
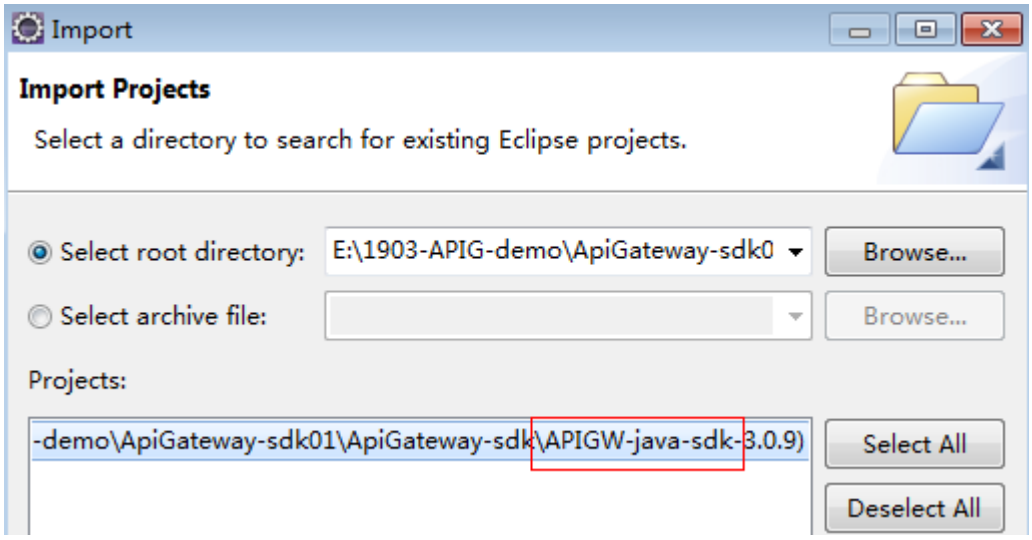
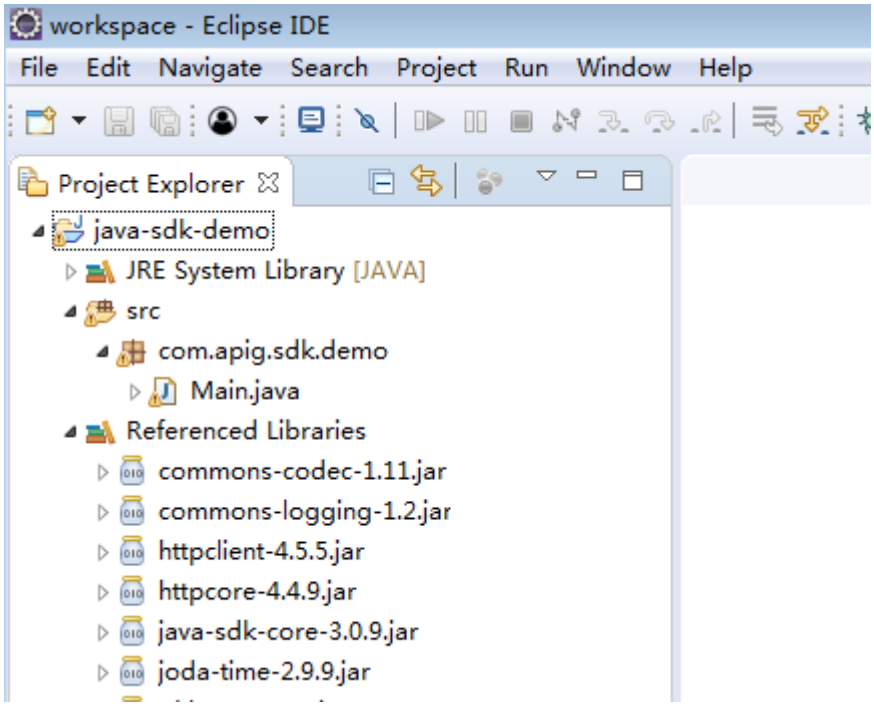


Figure 3-2 Import Projects



Step 2 Click **Finish**. The following figure shows the imported sample project.

Figure 3-3 Sample project for signing requests



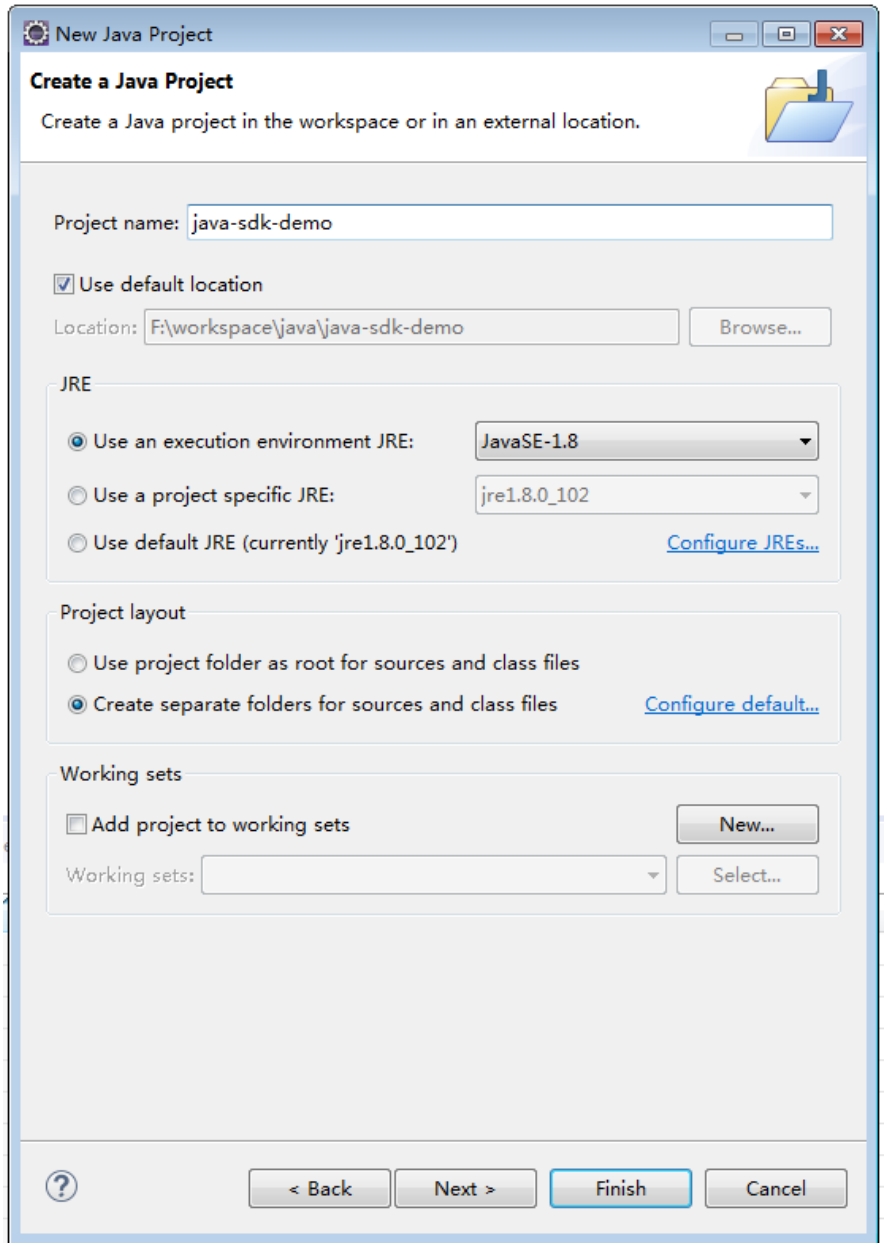
----End

NOTE

If Eclipse is installed, the JDK environment has been configured. Therefore, no more descriptions about JDK environment are provided.

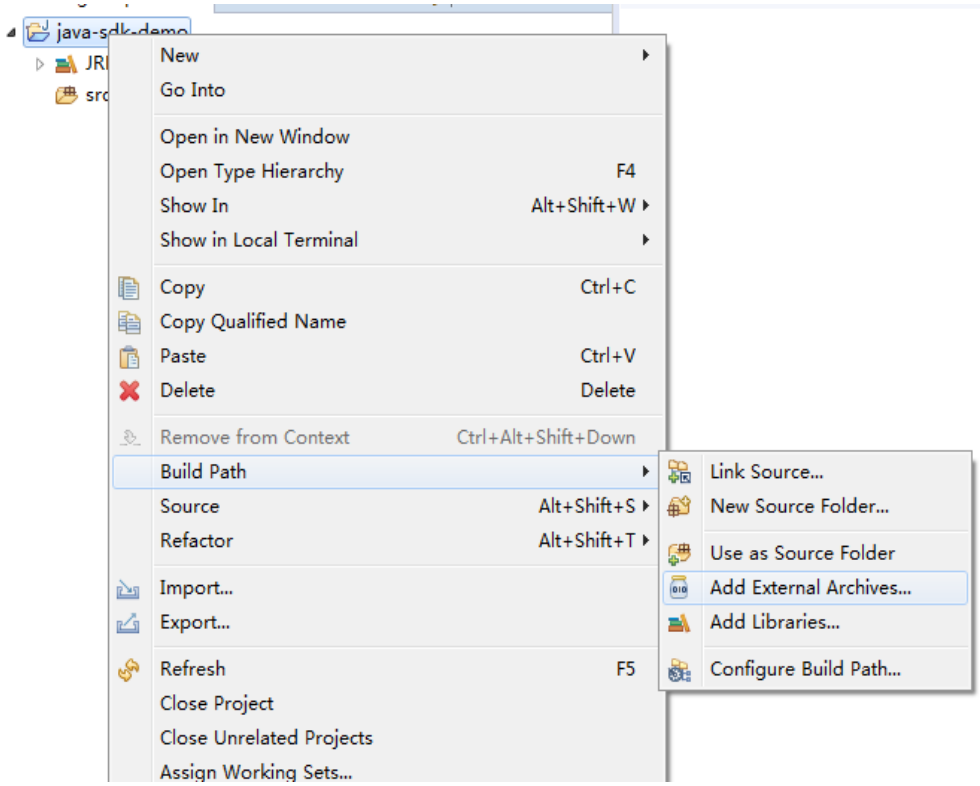
Creating a Project with the Signing SDK

Step 1 Start Eclipse and create a Java project. Specify a project name, for example, **java-sdk-demo**. Retain the default values for other parameters and click **Finish**.

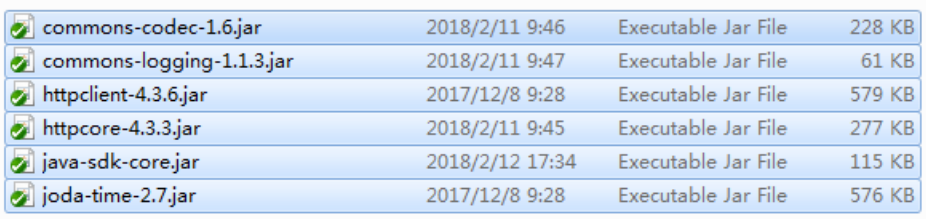


Step 2 Import the .jar files in the Java SDK.

1. Right-click the created project **java-sdk-demo** and choose **Build Path** > **Add External Archives**.



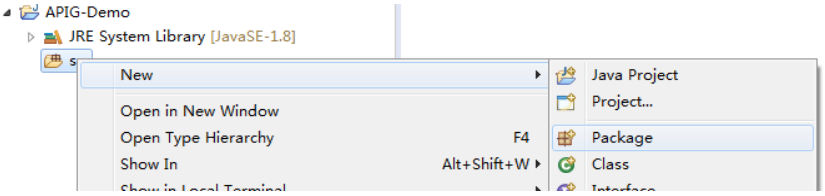
2. Select all .jar files in the **java\libs** directory.



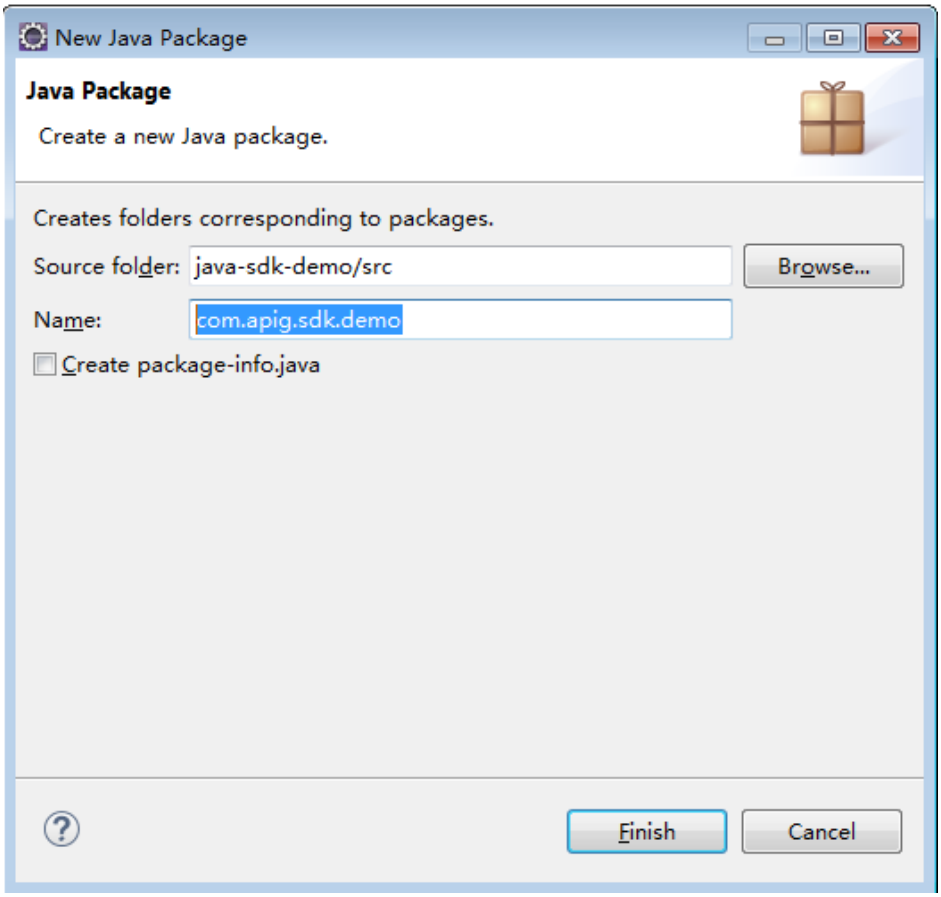
3. Click **Open**.

Step 3 Create a package and a class named **Main**.

1. Right-click **src** and choose **New** > **Package**.



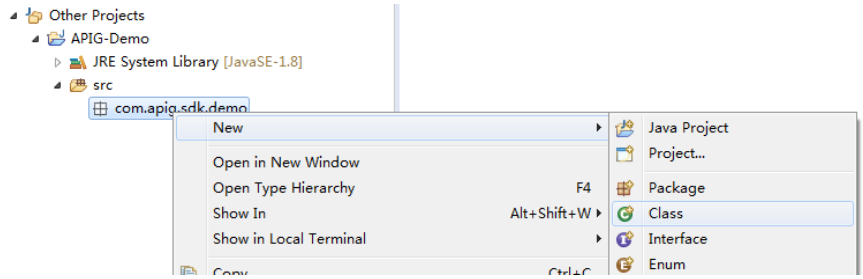
2. Enter **com.apig.sdk.demo** for **Name**.



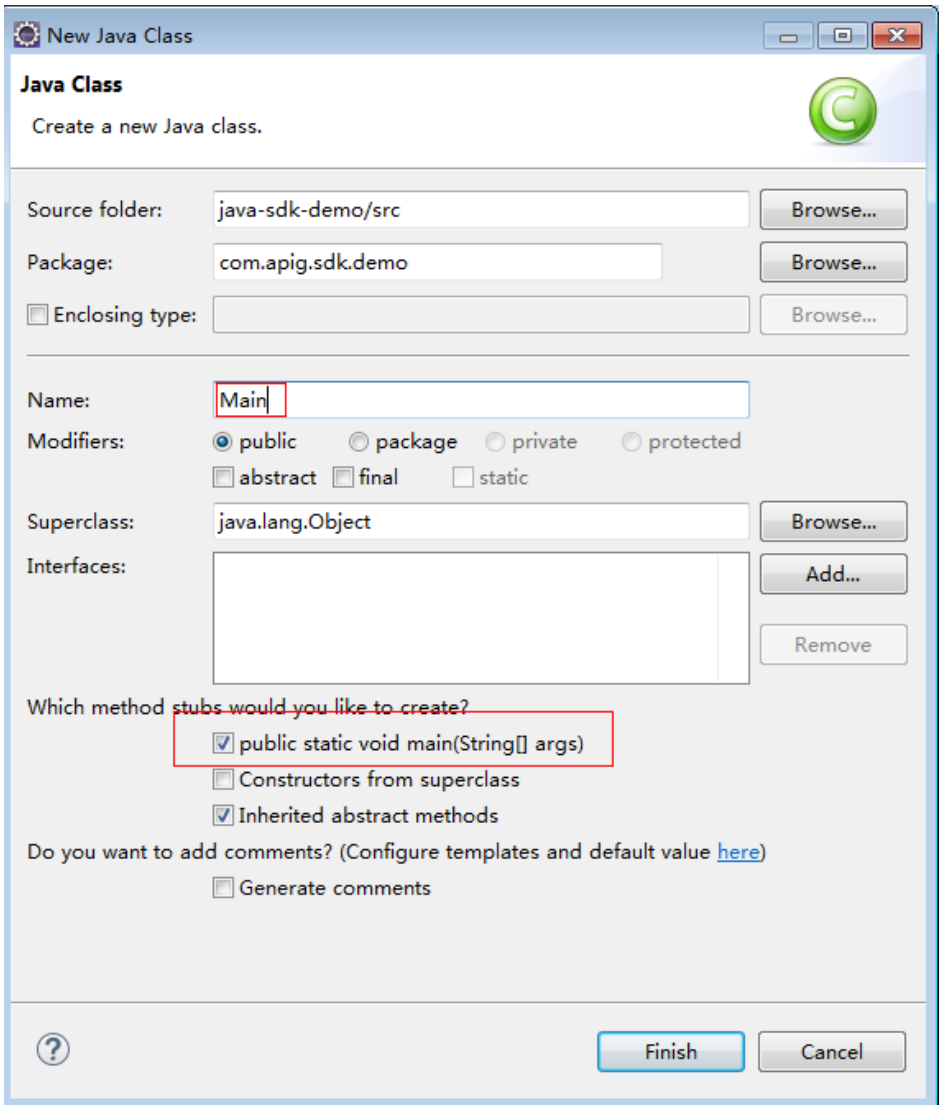
3. Click **Finish**.

   The package is created.

4. Right-click **com.apig.sdk.demo** and choose **New** > **Class**.



5. Enter **Main** for **Name** and select **public static void main(String[] args)**.



6. Click **Finish**.

   The **Main** file is created.

Step 4 The project is created.

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

----End

Calling APIs

After the calling information is modified, the sample code can be directly called. For details about the calling information, see **Preparations**. The following is a procedure for invoking the SDK in an application to sign requests.

Step 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;
```

Step 2 Create a request and set required parameters.

1. In this example, the AK and SK stored in the environment variables are used. Specify the environment variables *CLOUD_SDK_AK* and *CLOUD_SDK_SK* in the local environment first. The following uses Linux as an example to describe how to set the **obtained AK/SK** as environment variables.

   a. Open the terminal and run the following command to open the environment variable configuration file:

      vi ~/.bashrc

   b. Set environment variables, save the file, and exit the editor.

      ```
      export CLOUD_SDK_AK="Obtained AK"
      export CLOUD_SDK_SK="Obtained SK"
      ```

   c. Run the following command to apply the modification:

      source ~/.bashrc

2. Create a request and enter the created environment variables.

   Sample code and annotations:

```
Request request = new Request();
        try {
            //Set the AK/SK to sign and authenticate the request.



            request.setKey(System.getenv("CLOUD_SDK_AK"));
            request.setSecret(System.getenv("CLOUD_SDK_SK"));

            //The following example shows how to set the request URL and parameters to query a VPC list.

            //Specify a request method, such as GET, PUT, POST, DELETE, HEAD, and PATCH.
            request.setMethod("GET");

            //Set a request URL in the format of https://{Endpoint}/{URI}.
            request.setUrl("https://endpoint.example.com/v1/77b6a44cba5143ab91d13ab9a8ff44fd/vpcs?limit=2");

            //Set parameters for the request URL.
            request.addQueryStringParam("limit", "2");

            //Add header parameters, for example, Content-Type application/json.
            request.addHeader("Content-Type", "application/json");


            //Add a body if you have specified the PUT or POST method. Special characters, such as the double quotation mark ("), contained in the body must be escaped.
            //request.setBody("demo");
           //setBody can only be a string.

        } catch (Exception e) {
            e.printStackTrace();
            return;
        }
```

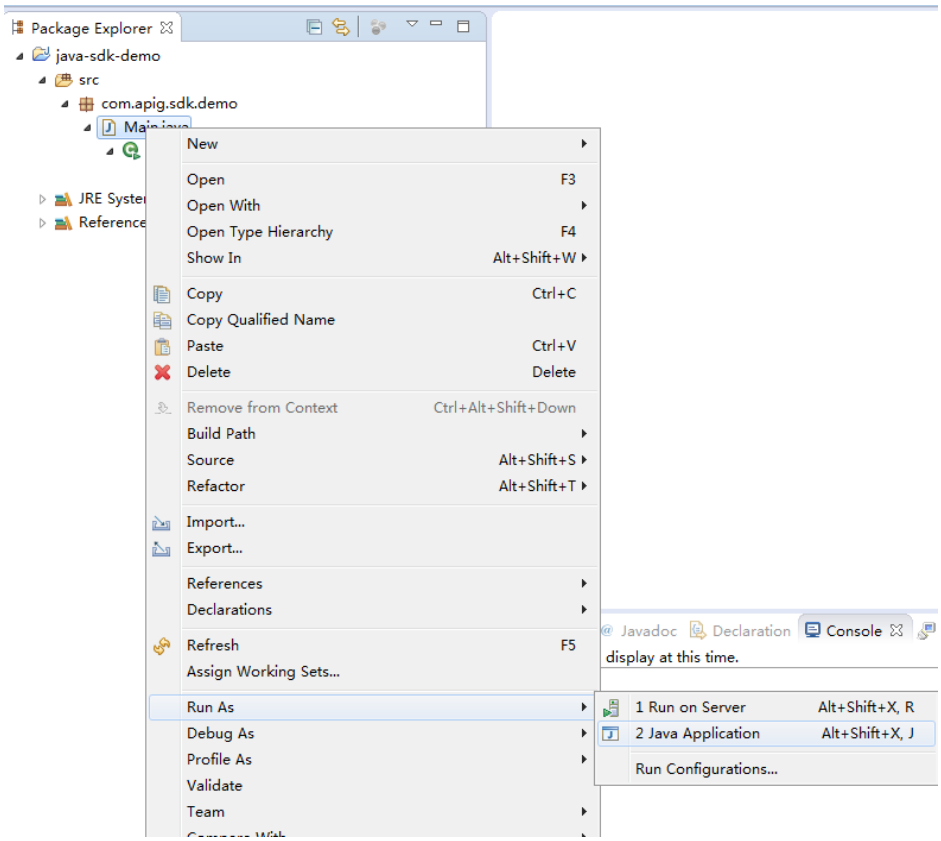

Step 3 Sign the request, access the API, and print the result.

The sample code is as follows:

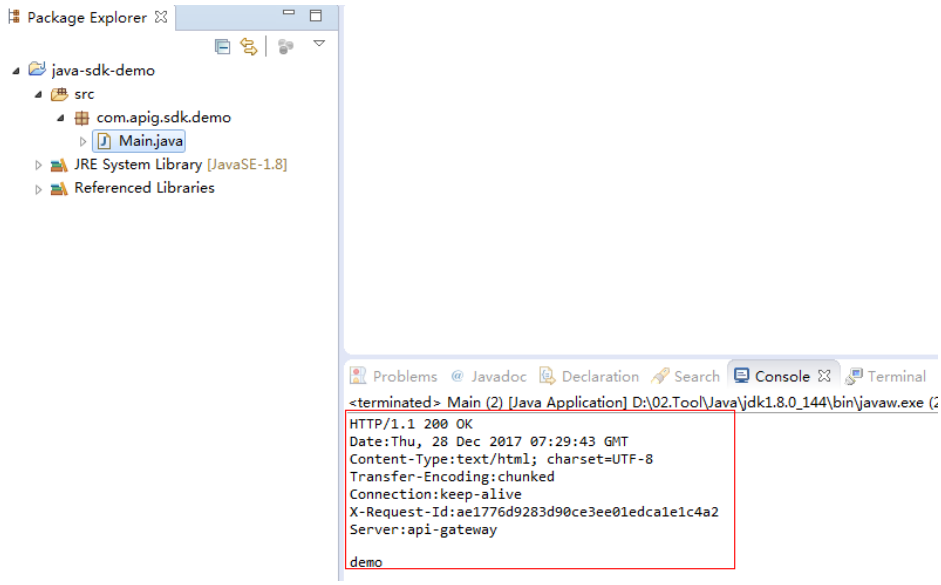
```
CloseableHttpClient client = null;
try
{
    HttpRequestBase signedRequest = Client.sign(request);

    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();
    }
}
```

Step 4 Right-click **Main.java** and choose **Run As** > **Java Application**.

Run the project test code.



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



----End

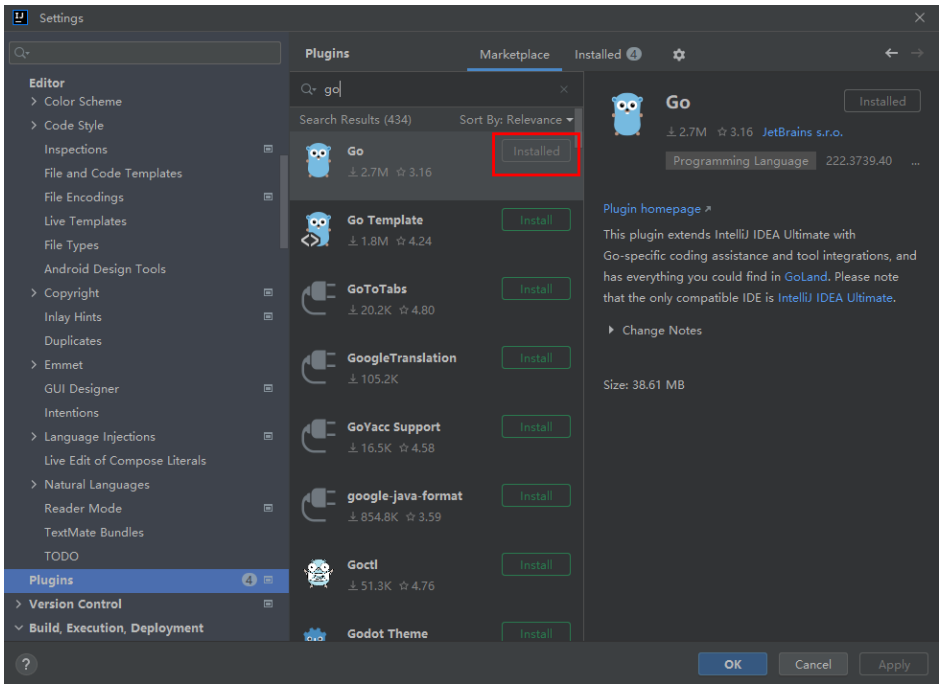
4 Go Signing Guide

This section uses IntelliJ IDEA as an example to describe how to integrate the Go SDK for API request signing. You can import the sample project in the code package, and integrate the signing SDK into your application by referring to the API calling example.

Preparing the IDEA Development Environment

- Download IntelliJ IDEA 2022.2.1 or later from the **IntelliJ IDEA official website** and install it.
- Download the Go 1.14 or later from the **Go official website** and install it.
- You have installed the Go plug-in on IntelliJ IDEA. Otherwise, install it according to **Figure 4-1**.

Figure 4-1 Installing the Go plug-in



Obtaining the SDK

Log in to the APIG console and choose **Help Center** > **SDK Process Flow**. Then download the SDK.

Develop your application using the SDK and sample code.

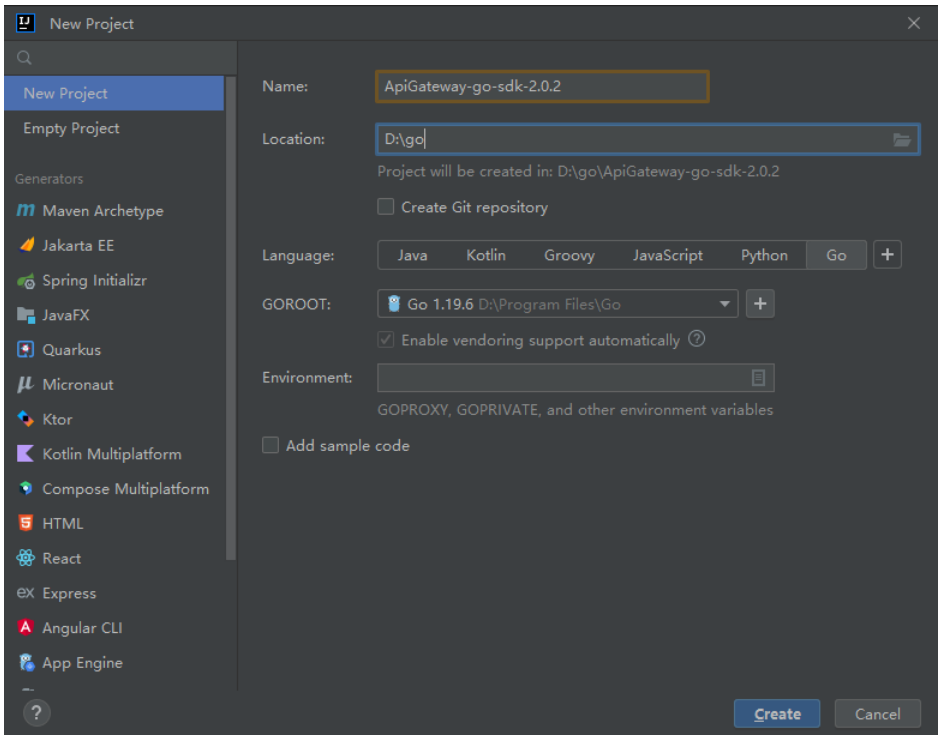
Name	Description
core\escape.go	Used for escaping special characters.
core\signer.go	Signing SDK
demo.go	Sample code

Creating a Project

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

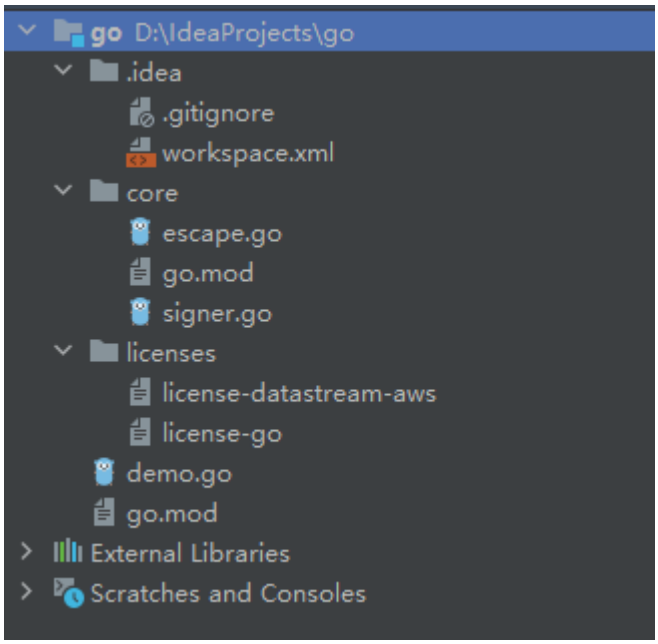
In the displayed **New Project** dialog box, set **Name** to the name of the folder in the SDK package, **Location** to the decompression path of the folder, and **Language** to **Go**, and click **Create**.

Figure 4-2 Go



Step 2 View the directory structure shown in the following figure.

Figure 4-3 Directory structure of the new project go



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

----End

Request Signing and API Calling

After the calling information is modified, the sample code can be directly called. For details about the calling information, see **Preparations**.

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

```
import "./core"
```

Step 2 Generate a new signer and enter the AK and SK.

1. In this example, the AK and SK stored in the environment variables are used. Specify the environment variables *CLOUD_SDK_AK* and *CLOUD_SDK_SK* in the local environment first. The following uses Linux as an example to describe how to set the **obtained AK/SK** as environment variables.
   a. Open the terminal and run the following command to open the environment variable configuration file:

      vi ~/.bashrc

   b. Set environment variables, save the file, and exit the editor.

      ```
      export CLOUD_SDK_AK="Obtained AK"
      export CLOUD_SDK_SK="Obtained SK"
      ```

   c. Run the following command to apply the modification:

      source ~/.bashrc

2. Generate a new signer and enter the configured environment variables.

   ```
   s := core.Signer{
         Key: os.Getenv("CLOUD_SDK_AK"),
         Secret: os.Getenv("CLOUD_SDK_SK"),
   }
   ```


Step 3 Generate a new request, and specify the domain name, method, request URI, and body.

```
//The following example shows how to set the request URL and parameters to query a VPC list.
//Add a body if you have specified the PUT or POST method. Special characters, such as the double quotation mark ("), contained in the body must be escaped.
r, _ := http.NewRequest("GET", "https://service.region.example.com/v1/{project_id}/vpcs?a=1",
ioutil.NopCloser(bytes.NewBuffer([]byte("")))
```

Step 4 Add other headers required for request signing or other purposes. For example, add the **X-Project-Id** header in **multi-project** scenarios or the **X-Domain-Id** header for a **global service**.

```
/Add header parameters, for example, X-Domain-Id for invoking a global service and X-Project-Id for invoking a project-level service.
r.Header.Add("X-Project-Id", "xxx")
```

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

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

Step 6 Access the API and view the access result.

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

----End

5 Python Signing Guide

This section uses IntelliJ IDEA as an example to describe how to integrate the Python SDK for API request signing. You can import the sample project in the code package, and integrate the signing SDK into your application by referring to the API calling example.

Preparing the Environment

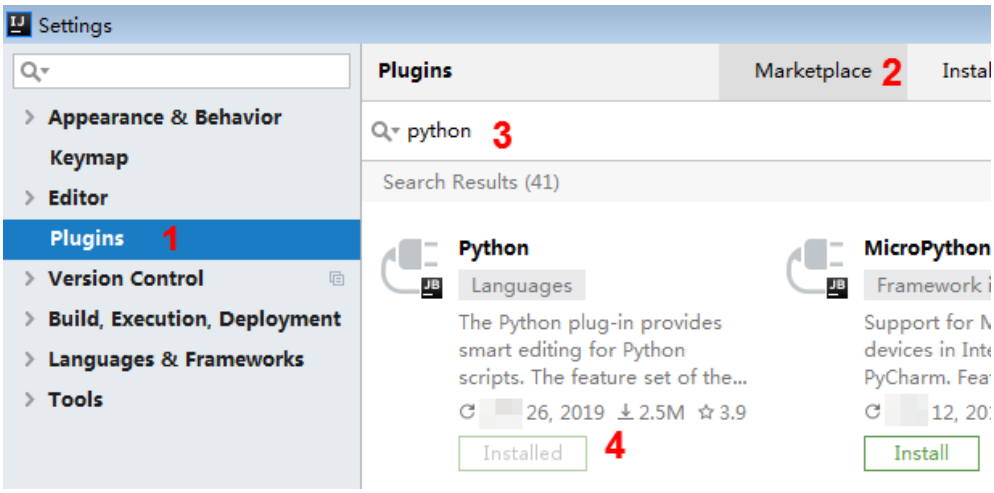
- Download IntelliJ IDEA 2018.3.5 or later from the **IntelliJ IDEA official website** and install it.
- Download the Python installation package (version 2.7.9 or later, or 3.x) from the **Python official website** and install it.

  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.
- Install the Python plug-in on IDEA.



Obtaining the SDK

Log in to the APIG console and choose **Help Center** > **SDK Process Flow**. Then download the SDK.

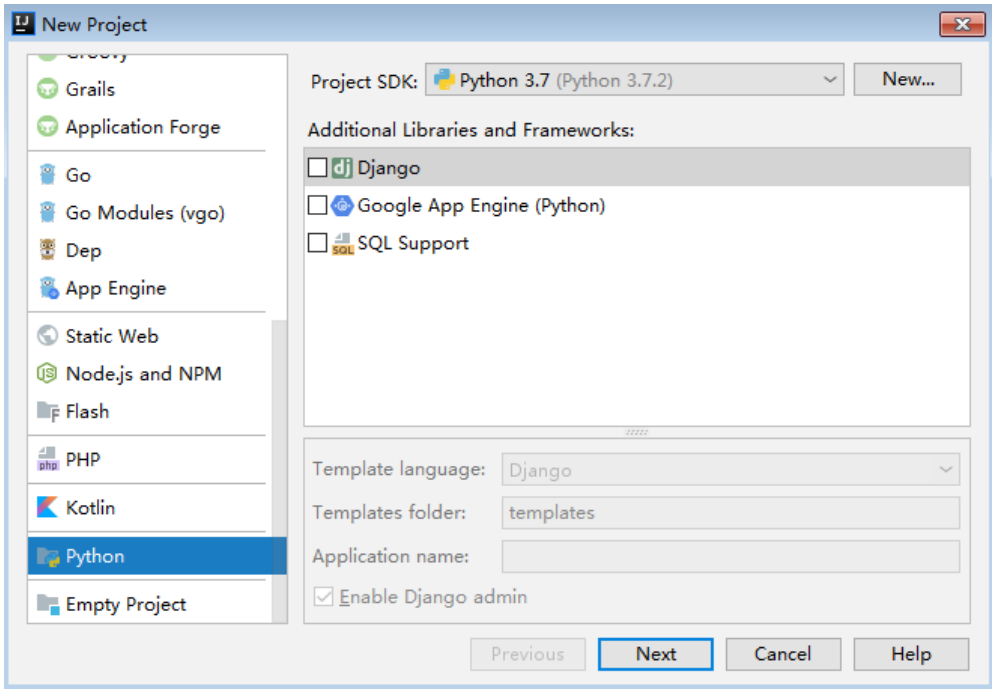
The following table shows the directory structure of the downloaded package.

Name	Description
apig_sdk__init__.py	SDK code
apig_sdk\signer.py	
main.py	Sample code
licenses\license-requests	Third-party license

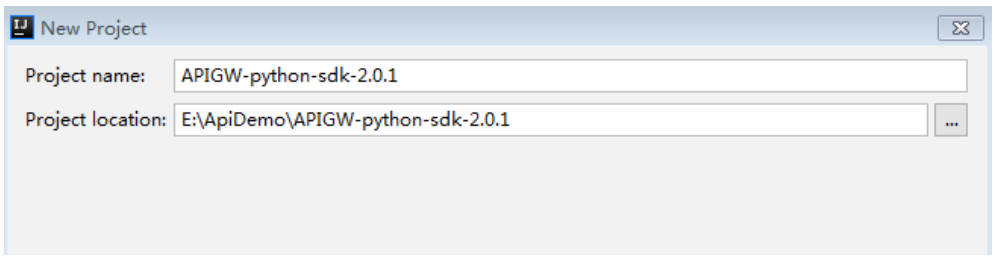
Importing the Sample Project

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

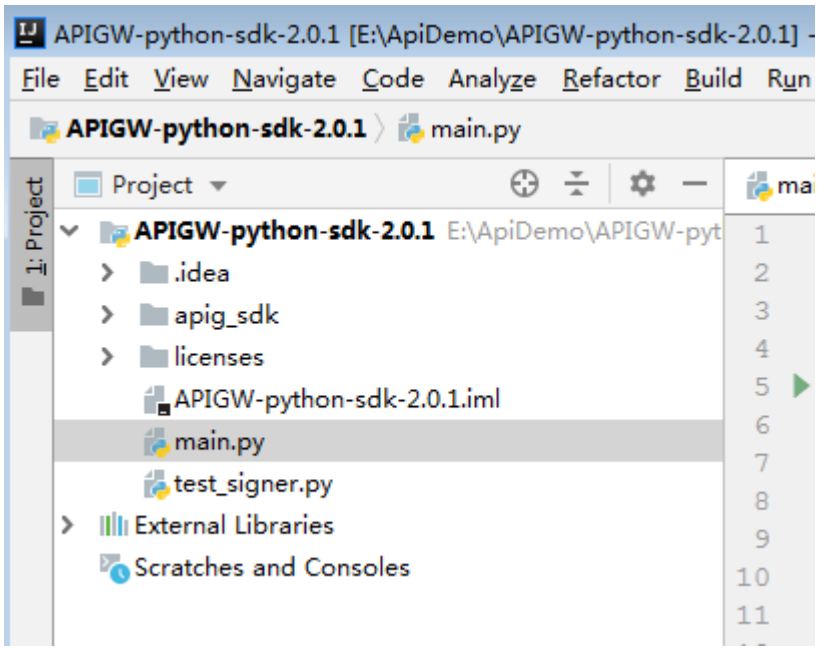
On the displayed **New Project** page, choose **Python** and click **Next**.



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



Step 3 View the directory structure shown in the following figure.



----End

Request Signing and API Calling

After the calling information is modified, the sample code can be directly called. For details about the calling information, see **Preparations**.

Step 1 Run the **pip** command to install the **requests** library.

```
pip install requests
```

Step 2 Import **apig_sdk** to the project.

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

Step 3 Generate a new signer and enter the AK and SK.

1. In this example, the AK and SK stored in the environment variables are used. Specify the environment variables *CLOUD_SDK_AK* and *CLOUD_SDK_SK* in the local environment first. The following uses Linux as an example to describe how to set the **obtained AK/SK** as environment variables.
   a. Open the terminal and run the following command to open the environment variable configuration file:

      vi ~/.bashrc

   b. Set environment variables, save the file, and exit the editor.

      ```
      export CLOUD_SDK_AK="Obtained AK"
      export CLOUD_SDK_SK="Obtained SK"
      ```

   c. Run the following command to apply the modification:

      source ~/.bashrc

2. Generate a new signer and enter the configured environment variables.

   ```
   sig = signer.Signer()
   # Set the AK/SK to sign and authenticate the request.
   sig.Key = os.getenv('CLOUD_SDK_AK')
   sig.Secret = os.getenv('CLOUD_SDK_SK')
   ```


Step 4 Generate a new request, and specify the domain name, method, request URI, and body.

Take the API for **querying VPCs** as an example: HTTP method **GET**, endpoint **service.region.example.com**, URI **/v1/77b6a44cba5143ab91d13ab9a8ff44fd/vpcs,** and query parameter **limit=1**.

```
# The following example shows how to set the request URL and parameters to query a VPC list.
r = signer.HttpRequest("GET", "https://service.region.example.com/v1/77b6a44cba5143ab91d13ab9a8ff44fd/vpcs?limit=1")
# r.body = "{\"a\":1}"
```

Step 5 Add other headers required for request signing or other purposes. For example, add the **X-Project-Id** header in **multi-project** scenarios or the **X-Domain-Id** header for a **global service**. Separate multiple headers with commas.

```
r.headers = {"X-Project-Id": "xxx"}
r.headers = {"X-Domain-Id": "xxx"}
```

Step 6 Execute the following function to add the **X-Sdk-Date** and **Authorization** headers for signing:

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

- X-Sdk-Date is a timestamp request header parameter that must be signed.
- The Authorization header is a mandatory request header parameter for security authentication.
- The SDK automatically completes signing requests, and you do not need to know which header parameters are involved in the signing process.

Step 7 Access the API and view the access result.

```
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)
```

The following uses the **API for querying VPCs** as an example. The access result is as follows:

```
{
   "vpcs": [
      {
         "id": "13551d6b-755d-4757-b956-536f674975c0",
         "name": "default",
         "description": "test",
         "cidr": "172.16.0.0/16",
         "status": "OK",
         "enterprise_project_id": "0",
         "routes": [],
         "block_service_endpoint_states": "off",
         "enable_network_address_usage_metrics": false,
         "tenant_id": "087679f0aa80d32a2f4ec0172f5e902b",
         "created_at": "2022-12-15T02:11:13",
         "updated_at": "2022-12-15T02:11:13"
      }
   ]
}
```

----End

6 C# Signing Guide

This section uses Visual Studio as an example to describe how to integrate the C# SDK for API request signing. You can import the sample project in the code package, and integrate the signing SDK into your application by referring to the API calling example.

Preparing the Environment

Download Visual Studio 2019 16.8.4 or later from the **Visual Studio official website** and install it.

Obtaining the SDK

Log in to the APIG console and choose **Help Center** > **SDK Process Flow**. Then download the SDK.

The following table shows the directory structure of the downloaded package.

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

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.

Request Signing and API Calling

After the calling information is modified, the sample code can be directly called. For details about the calling information, see **Preparations**.

Step 1 Import the SDK to the project.

```
using System;
using System.Net;
using System.IO;
using System.Net.Http;
using System.Threading;
using APIGATEWAY_SDK;
```

Step 2 Generate a new signer and enter the AK and SK.

1. In this example, the AK and SK stored in the environment variables are used. Specify the environment variables *CLOUD_SDK_AK* and *CLOUD_SDK_SK* in the local environment first. The following uses Linux as an example to describe how to set the **obtained AK/SK** as environment variables.
   a. Open the terminal and run the following command to open the environment variable configuration file:

      vi ~/.bashrc

   b. Set environment variables, save the file, and exit the editor.

      ```
      export CLOUD_SDK_AK="Obtained AK"
      export CLOUD_SDK_SK="Obtained SK"
      ```

   c. Run the following command to apply the modification:

      source ~/.bashrc

2. Generate a new signer and enter the configured environment variables.

   ```
   Signer signer = new Signer();
   signer.Key = Environment.GetEnvironmentVariable("CLOUD_SDK_AK");
   signer.Secret = Environment.GetEnvironmentVariable("CLOUD_SDK_SK");
   ```

Step 3 Generate a new request, and specify the domain name, method, request URI, and body.

```
//The following example shows how to set the request URL and parameters to query a VPC list.
HttpRequest r = new HttpRequest("GET", new Uri("https://
{service}.region.example.com/v1/77b6a44cba5**********9a8ff44fd/vpcs?limit=1"));
//Add a body if you have specified the PUT or POST method. Special characters, such as the double quotation mark ("), contained in the body must be escaped.
r.body = "";
```

Step 4 Add other headers required for request signing or other purposes. For example, add the **x-stage** header for **API environment**, **X-Project-Id** header in **multi-project** scenarios or the **X-Domain-Id** header for a **global service**.

```
r.headers.Add("x-stage", "RELEASE");
r.headers.Add("X-Project-Id", "xxx");
r.headers.Add("X-Domain-Id", "xxx");
//Add header parameters, for example, X-Domain-Id for invoking a global service and X-Project-Id for invoking a project-level service.
```

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

If you use HttpClient, you can obtain header information from the request. For details about headers, see **API Signature Authentication Mechanism**.


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

Step 6 Access the API and view the access result.

```
try
{
    var writer = new StreamWriter(req.GetRequestStream());
    writer.Write(r.body);
    writer.Flush();
    HttpWebResponse resp = (HttpWebResponse)req.GetResponse();
    var reader = new StreamReader(resp.GetResponseStream());
    Console.WriteLine(reader.ReadToEnd());
}
catch (WebException e)
{
    HttpWebResponse resp = (HttpWebResponse)e.Response;
    if (resp != null)
    {
        Console.WriteLine((int)resp.StatusCode + " " + resp.StatusDescription);
        var reader = new StreamReader(resp.GetResponseStream());
        Console.WriteLine(reader.ReadToEnd());
    }
    else
    {
        Console.WriteLine(e.Message);
    }
}
Console.WriteLine("----------------");
```

----End

7 JavaScript Signing Guide

This section uses IntelliJ IDEA as an example to describe how to integrate the JavaScript SDK for API request signing. You can import the sample project in the code package, and integrate the signing SDK into your application by referring to the API calling example.

The descriptions in this section are provided based on the Node.js environment.

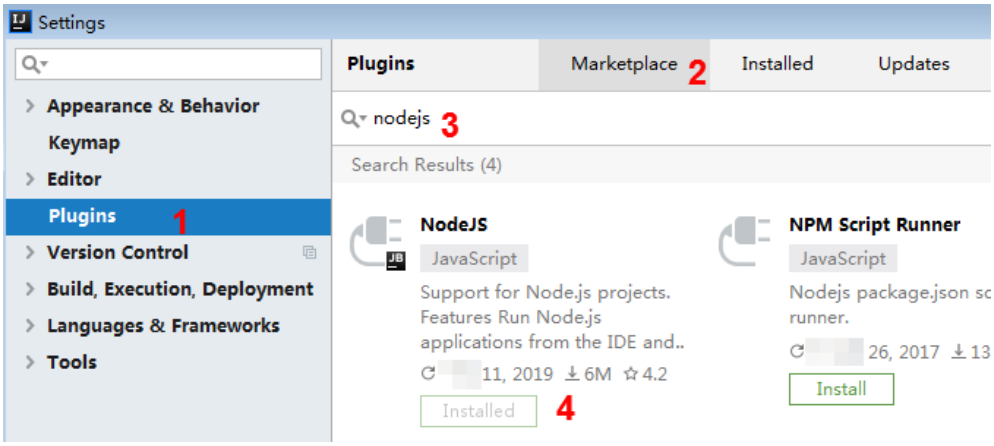
Preparing the Environment

- Download IntelliJ IDEA 2018.3.5 or later from the **IntelliJ IDEA official website** and install it.
- Download the Node.js 15.10.0 or later from the **Node.js official website** and install it.

  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
  ```

- Install the Node.js plug-in on IDEA.



Obtaining the SDK

Log in to the APIG console and choose **Help Center** > **SDK Process Flow**. Then download the SDK.

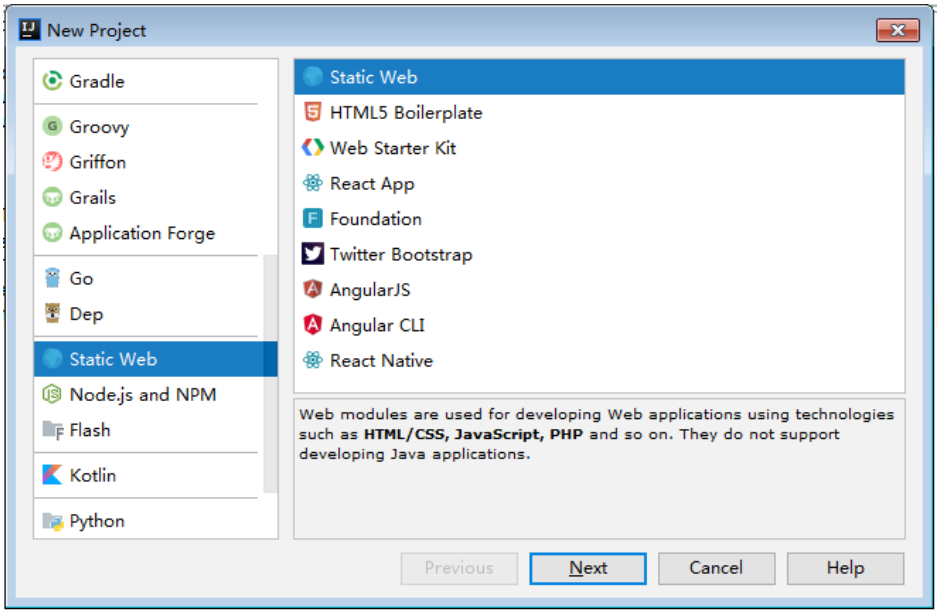
Decompress the downloaded package to the current folder. The following table shows the directory structure.

Name	Description
signer.js	SDK code
node_demo.js	Node.js sample code
test.js	Test case
licenses\license-crypto-js	Third-party licenses
licenses\license-node	

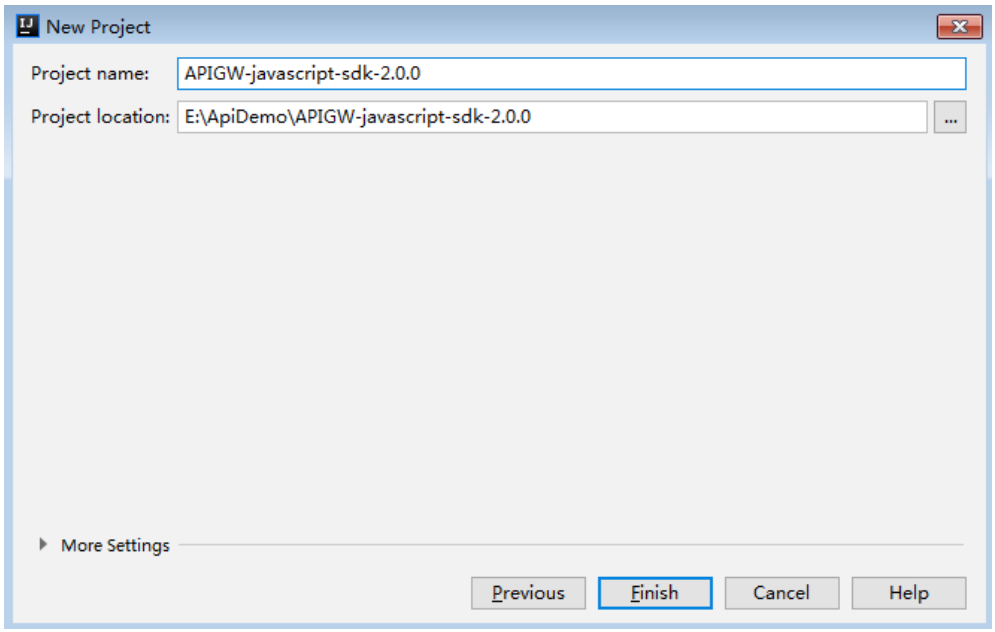
Creating a Project

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

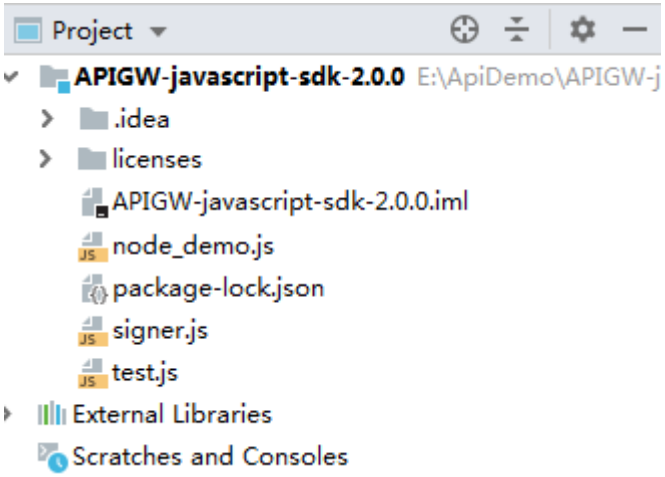
In the **New Project** dialog box, choose **Static Web** and click **Next**.



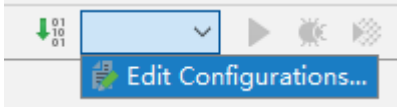
Step 2 Click **...**, select the directory where the SDK is decompressed, and click **Finish**.



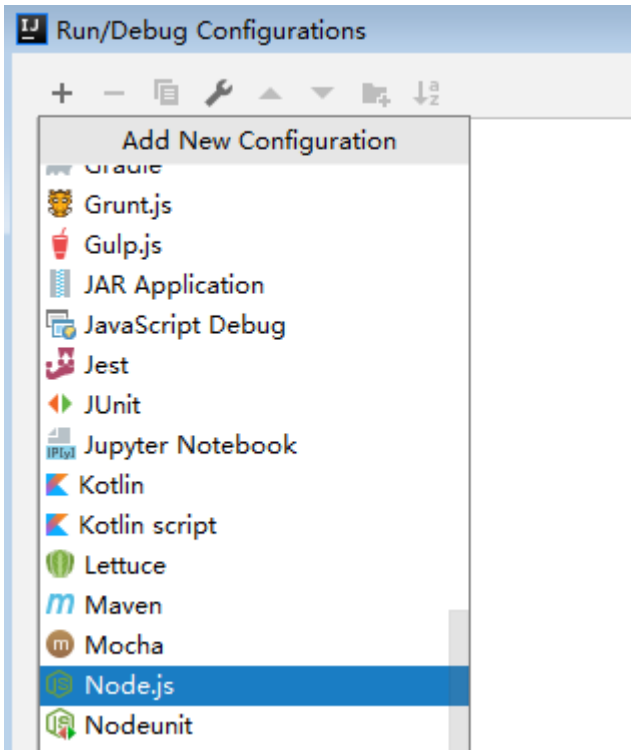
Step 3 View the directory structure shown in the following figure.



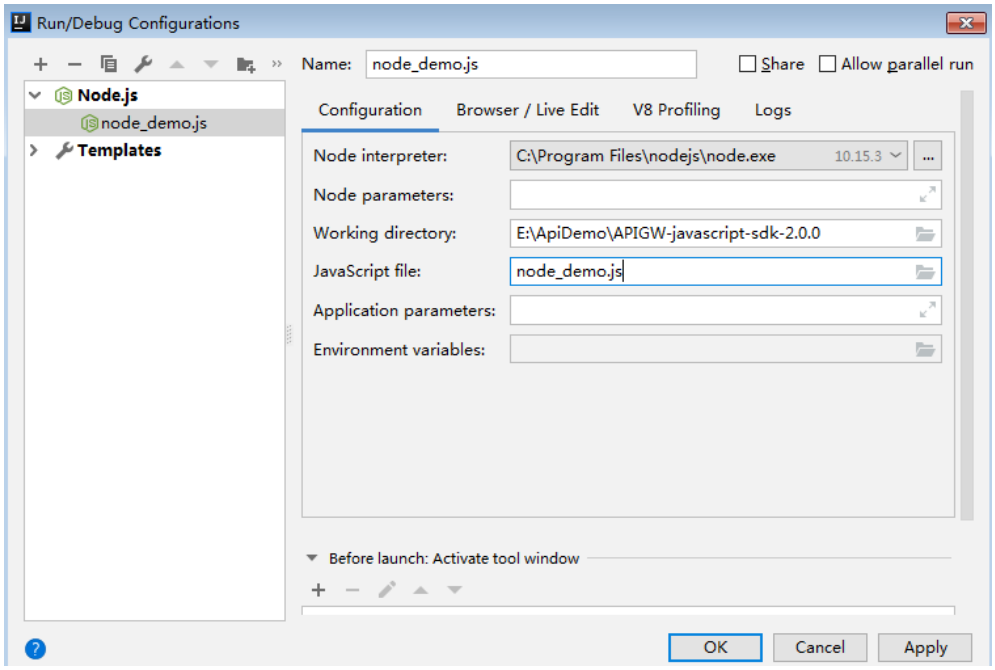
Step 4 In the upper right corner of the IDEA window, click **Edit Configurations** or **Add Configurations**.



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



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



----End

Calling APIs (Node.js)

After the calling information is modified, the sample code can be directly called. For details about the calling information, see **Preparations**.

Step 1 Run the **npm** command to install the **moment** and **moment-timezone** modules.


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

Step 2 Import **signer.js** to your project.

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

Step 3 Generate a new signer and enter the AK and SK.

1. In this example, the AK and SK stored in the environment variables are used. Specify the environment variables *CLOUD_SDK_AK* and *CLOUD_SDK_SK* in the local environment first. The following uses Linux as an example to describe how to set the **obtained AK/SK** as environment variables.

   a. Open the terminal and run the following command to open the environment variable configuration file:

      vi ~/.bashrc

   b. Set environment variables, save the file, and exit the editor.

      ```
      export CLOUD_SDK_AK="Obtained AK"
      export CLOUD_SDK_SK="Obtained SK"
      ```

   c. Run the following command to apply the modification:

      source ~/.bashrc

2. Generate a new signer and enter the configured environment variables.

   ```
   var sig = new signer.Signer()
   sig.Key = process.env.CLOUD_SDK_AK
   sig.Secret = process.env.CLOUD_SDK_SK
   ```

Step 4 Generate a new request, and specify the domain name, method, request URI, and body.

```
//The following example shows how to set the request URL and parameters to query a VPC list.
var r = new signer.HttpRequest("GET",
"service.region.example.com/v1/77b6a44cba5143ab91d13ab9a8ff44fd/vpcs?limie=1");

//Add a body if you have specified the PUT or POST method. Special characters, such as the double quotation mark ("), contained in the body must be escaped.
r.body = '';
```

Step 5 Add other headers required for request signing or other purposes. For example, add the **X-Project-Id** header in **multi-project** scenarios or the **X-Domain-Id** header for a **global service**.

```
//Add header parameters, for example, X-Domain-Id for invoking a global service and X-Project-Id for invoking a project-level service.
r.headers = {"X-Project-Id": "xxx"};
```

Step 6 Execute the following function to generate HTTPS request parameters, and add the **X-Sdk-Date** and **Authorization** headers for signing the request:

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

Step 7 Access the API and view the access result.

```
var req = https.request(opt, 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()
```

----End

8 PHP Signing Guide

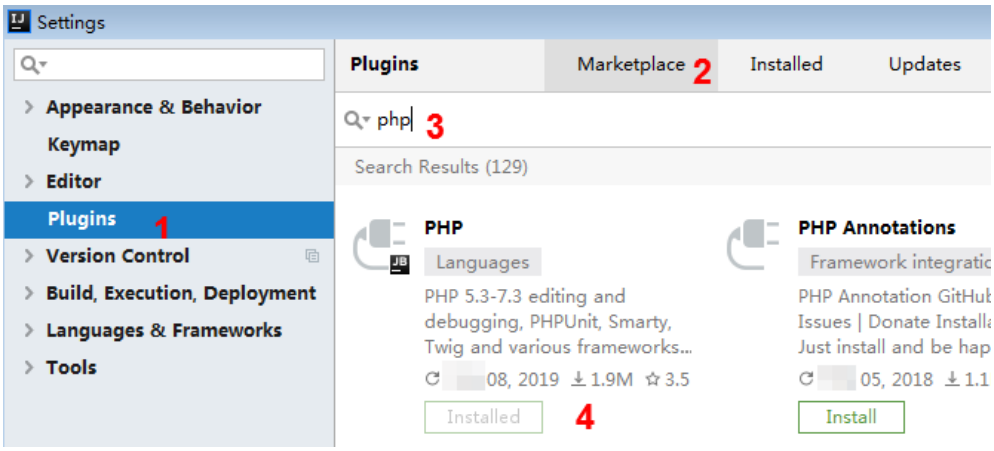
This section uses IntelliJ IDEA as an example to describe how to integrate the PHP SDK for API request signing. You can import the sample project in the code package, and integrate the signing SDK into your application by referring to the API calling example.

Preparing the Environment

- Download IntelliJ IDEA 2018.3.5 or later from the **IntelliJ IDEA official website** and install it.
- Download the PHP 8.0.3 or later from the **PHP official website** and install it.
- 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
```

- Install the PHP plug-in on IDEA.



Obtaining the SDK

Log in to the APIG console and choose **Help Center** > **SDK Process Flow**. Then download the SDK.

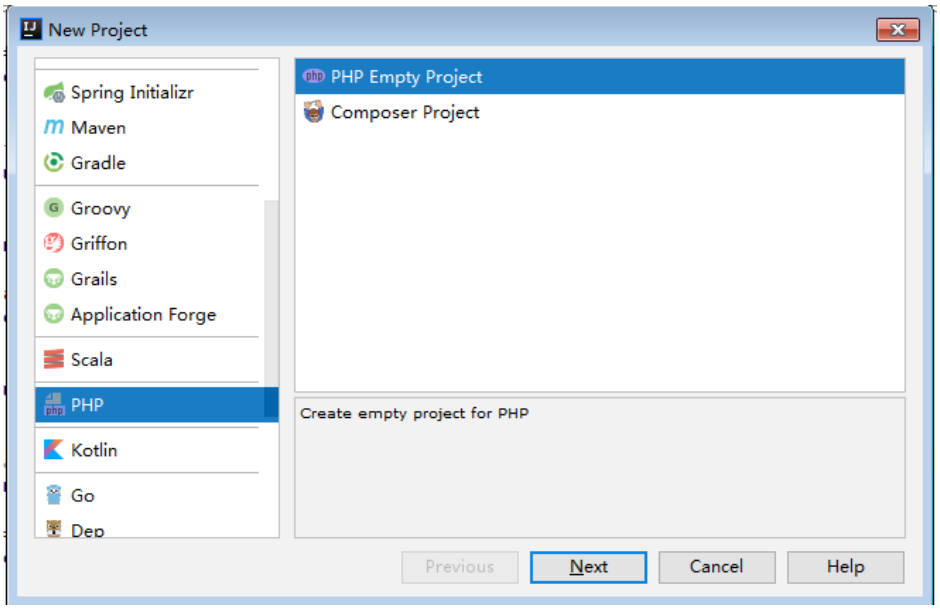
Decompress the downloaded package to the current folder. The following table shows the directory structure.

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

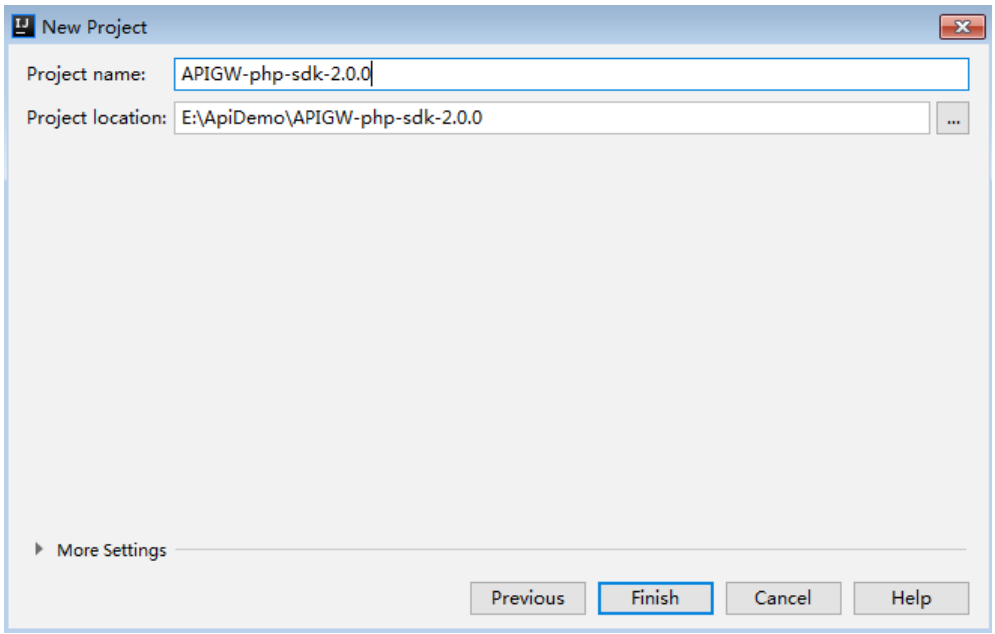
Creating a Project

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

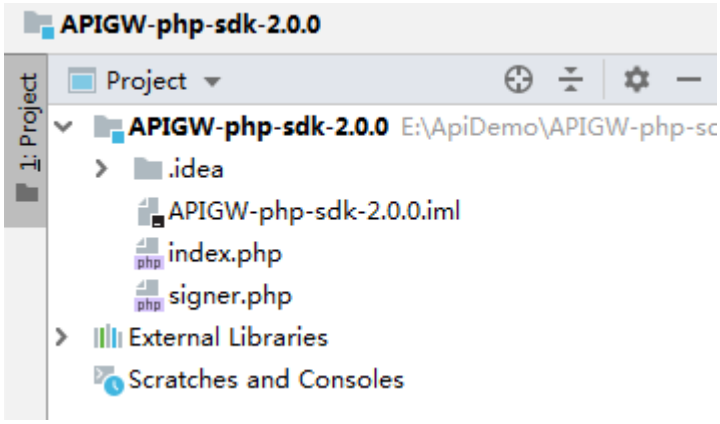
On the displayed **New Project** page, choose **PHP** and click **Next**.



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



Step 3 View the directory structure shown in the following figure.



----End

Request Signing and API Calling

After the calling information is modified, the sample code can be directly called. For details about the calling information, see **Preparations**.

Step 1 Import the PHP SDK to your code.

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

Step 2 Generate a new signer and enter the AK and SK.

1. In this example, the AK and SK stored in the environment variables are used. Specify the environment variables *CLOUD_SDK_AK* and *CLOUD_SDK_SK* in the local environment first. The following uses Linux as an example to describe how to set the **obtained AK/SK** as environment variables.
   a. Open the terminal and run the following command to open the environment variable configuration file:

      vi ~/.bashrc

   b. Set environment variables, save the file, and exit the editor.

      ```
      export CLOUD_SDK_AK="Obtained AK"
      export CLOUD_SDK_SK="Obtained SK"
      ```

   c. Run the following command to apply the modification:

      source ~/.bashrc

2. Generate a new signer and enter the configured environment variables.

   ```
   $signer = new Signer();
   $signer->Key = getenv('CLOUD_SDK_AK');
   $signer->Secret = getenv('CLOUD_SDK_SK');
   ```

Step 3 Generate a new request, and specify the domain name, method, request URI, and body.

```
//The following example shows how to set the request URL and parameters to query a VPC list.
$req = new Request('GET', 'https://service.region.example.com/v1/{project_id}/vpcs?limit=1');
//Add a body if you have specified the PUT or POST method. Special characters, such as the double quotation mark ("), contained in the body must be escaped.
$req->body = '';
```

Step 4 Add other headers required for request signing or other purposes. For example, add the **X-Project-Id** header in **multi-project** scenarios or the **X-Domain-Id** header for a **global service**.


```
//Add header parameters, for example, X-Domain-Id for invoking a global service and X-Project-Id for invoking a project-level service.
$req->headers = array(
   'X-Project-Id' => 'xxx',
);
```

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

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

Step 6 Access the API and view the access result.

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

----End

9 C++ Signing Guide

Preparing the Environment

This section uses Linux Ubuntu as an example. Before calling APIs, install the required SSL tools.

1. Install the OpenSSL library.
   ```
   apt-get install libssl-dev
   ```
2. Install the curl library.
   ```
   apt-get install libcurl4-openssl-dev
   ```

Obtaining the SDK

Log in to the APIG console and choose **Help Center** > **SDK Process Flow**. Then download the SDK.

Decompress the downloaded package to the current folder. The following table shows the directory structure.

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

Request Signing and API Calling

After the calling information is modified, the sample code can be directly called. For details about the calling information, see **Preparations**.

Step 1 Add the following references to **main.cpp**:

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

Step 2 Generate a new signer and enter the AK and SK.

1. In this example, the AK and SK stored in the environment variables are used. Specify the environment variables *CLOUD_SDK_AK* and *CLOUD_SDK_SK* in the local environment first. The following uses Linux as an example to describe how to set the **obtained AK/SK** as environment variables.
   a. Open the terminal and run the following command to open the environment variable configuration file:

      vi ~/.bashrc

   b. Set environment variables, save the file, and exit the editor.

      ```
      export CLOUD_SDK_AK="Obtained AK"
      export CLOUD_SDK_SK="Obtained SK"
      ```

   c. Run the following command to apply the modification:

      source ~/.bashrc

2. Generate a new signer and enter the configured environment variables.

   ```
   //Set the AK/SK to sign and authenticate the request.
   Signer signer(getenv("CLOUD_SDK_AK"), getenv("CLOUD_SDK_SK"));
   ```

Step 3 Generate a new **RequestParams** request, and specify the method, domain name, request URI, query strings, and request body.

```
//Specify a request method, such as GET, PUT, POST, DELETE, HEAD, and PATCH.
//Set a request URL.
//Set parameters for the request URL.
//Add a body if you have specified the PUT or POST method. Special characters, such as the double quotation mark ("), contained in the body must be escaped.
RequestParams* request = new RequestParams("GET", "service.region.example.com", "/v1/{project_id}/vpcs",
      "limit=2", "");
```

Step 4 Add other headers required for request signing or other purposes. For example, add the **X-Project-Id** header in **multi-project** scenarios or the **X-Domain-Id** header for a **global service**.

```
//Add header parameters, for example, X-Domain-Id for invoking a global service and X-Project-Id for invoking a project-level service.
request->addHeader("X-Project-Id", "xxx");
```

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

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

Step 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;
}
```

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

----End

10 C Signing Guide

Preparing the Environment

This section uses Linux Ubuntu as an example. Before calling APIs, install the required SSL tools.

1. Install the OpenSSL library.
   ```
   apt-get install libssl-dev
   ```
2. Install the curl library.
   ```
   apt-get install libcurl4-openssl-dev
   ```

Obtaining the SDK

Log in to the APIG console and choose **Help Center** > **SDK Process Flow**. Then download the SDK.

Decompress the downloaded package to the current folder. The following table shows the directory structure.

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

Request Signing and API Calling

After the calling information is modified, the sample code can be directly called. For details about the calling information, see **Preparations**.

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

```
#include <stdio.h>
#include <stdlib.h>
```



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

Step 2 Generate a sig_params_t variable, and enter the AK and SK.

1. In this example, the AK and SK stored in the environment variables are used. Specify the environment variables *CLOUD_SDK_AK* and *CLOUD_SDK_SK* in the local environment first. The following uses Linux as an example to describe how to set the **obtained AK/SK** as environment variables.
   a. Open the terminal and run the following command to open the environment variable configuration file:

      vi ~/.bashrc

   b. Set environment variables, save the file, and exit the editor.

      ```
      export CLOUD_SDK_AK="Obtained AK"
      export CLOUD_SDK_SK="Obtained SK"
      ```

   c. Run the following command to apply the modification:

      source ~/.bashrc

2. Generate a **sig_params_t** variable, and enter the configured environment variables.

   ```
   sig_params_t params;
   sig_params_init(&params);
   sig_str_t ak = sig_str(getenv("CLOUD_SDK_AK"));
   sig_str_t sk = sig_str(getenv("CLOUD_SDK_SK"));
   params.key = ak;
   params.secret = sk;
   ```

Step 3 Specify the method, domain name, request URI, query strings, and request body.

```
sig_str_t host = sig_str("service.region.example.com");
sig_str_t method = sig_str("GET");
sig_str_t uri = sig_str("/v1/{project_id}/vpcs");
sig_str_t query_str = sig_str("limit=2");
sig_str_t payload = sig_str("");
params.host = host;
params.method = method;
params.uri = uri;
params.query_str = query_str;
params.payload = payload;
```

Step 4 Add header parameters or other headers required for other purposes. For example, add the **X-Project-Id** header in **multi-project** scenarios or the **X-Domain-Id** header for a **global service**.

```
//Add header parameters, for example, X-Domain-Id for invoking a global service and X-Project-Id for invoking a project-level service.
sig_headers_add(&params.headers, "X-Project-Id", "xxx");
```

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

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

Step 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;
}
```

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

----End

11 Android Signing Guide

This section uses Android Studio as an example to describe how to integrate the Android SDK for API request signing. You can import the sample project in the code package, and integrate the signing SDK into your application by referring to the API calling example.

Preparing the Environment

Download Android Studio 4.1.2 or later at the **Android Studio official website** and install it.

Obtaining the SDK

Log in to the APIG console and choose **Help Center** > **SDK Process Flow**. Then download the SDK.

The following table shows the directory structure of the downloaded package.

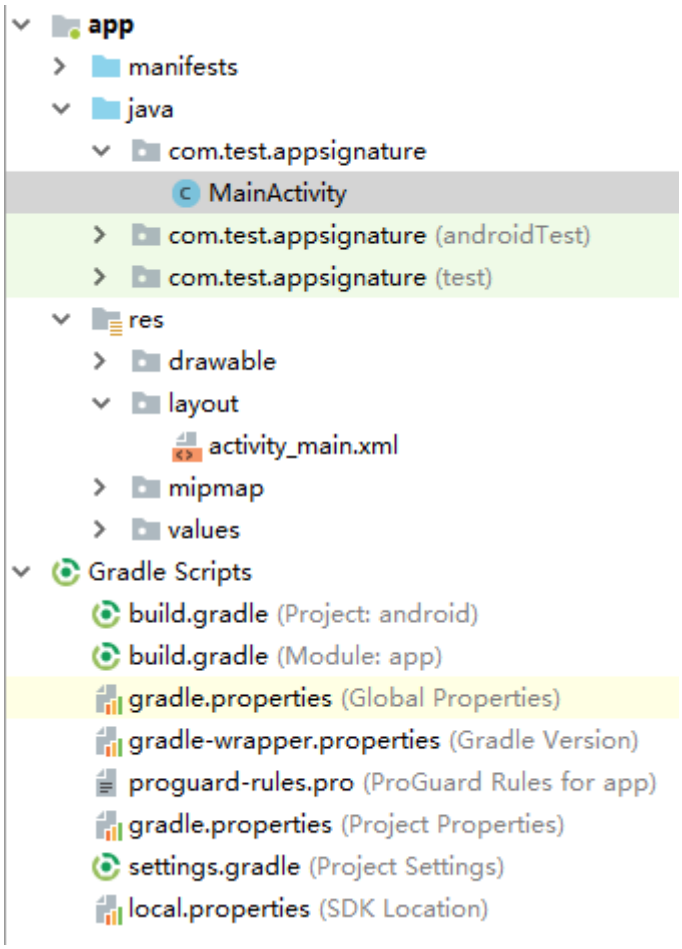
Name	Description
app\	Android project code
build.gradle	Gradle configuration files
gradle.properties	
settings.gradle	

Opening the Sample Project

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

Select the directory where the SDK is decompressed.

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



----End

Request Signing and API Calling

After the calling information is modified, the sample code can be directly called. For details about the calling information, see **Preparations**.

Step 1 Ensure that the **java-sdk-core-x.x.x.jar** package has been added to the **app/libs** directory of the Android project.

Step 2 Create a request, enter the AK and SK, and specify the domain name, method, request URI, and body.

1. In this example, the AK and SK stored in the environment variables are used. Specify the environment variables *CLOUD_SDK_AK* and *CLOUD_SDK_SK* in the local environment first. The following uses Linux as an example to describe how to set the **obtained AK/SK** as environment variables.
   a. Open the terminal and run the following command to open the environment variable configuration file:

      vi ~/.bashrc

   b. Set environment variables, save the file, and exit the editor.

      ```
      export CLOUD_SDK_AK="Obtained AK"
      export CLOUD_SDK_SK="Obtained SK"
      ```

   c. Run the following command to apply the modification:

source ~/.bashrc

2. Create a request, enter the configured environment variables, and specify the domain name, method, request URI, and body.

```
Request request = new Request();
try {
        request.setKey(System.getenv("CLOUD_SDK_AK"));
        request.setSecret(System.getenv("CLOUD_SDK_SK"));
        request.setMethod("GET");
        request.setUrl("https://service.region.example.com3/v1/{project_id}/vpcs");
        request.addQueryStringParam("name", "value");
        request.addHeader("Content-Type", "text/plain");
        //request.setBody("demo");
} catch (Exception e) {
    e.printStackTrace();
    return;
}
```

Step 3 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();
```

----End

12 API Signature Authentication Principles

12.1 API Signature Authentication Mechanism

If the preceding signing examples do not contain the programming language you use, sign requests according to the API signature authentication mechanism.

Step 1: Constructing a Standard Request

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

The pseudocode of standard HTTP requests is as follows:

```
CanonicalRequest =
    HTTPRequestMethod + '\n' + //HTTP request method
    CanonicalURI + '\n' + //Canonical URI
    CanonicalQueryString + '\n' + //Canonical query string
    CanonicalHeaders + '\n' + //Canonical message header
    SignedHeaders + '\n' + //Signed message header
    HexEncode(Hash(RequestPayload)) //Calculate the hash value of RequestPayload.
```

- **HTTPRequestMethod**

  HTTP request methods, such as GET, PUT, and POST, followed by a newline character.

- **CanonicalURI**

  Path of the requested resource, which is the URI code of the absolute path. The value ends with a newline character.

  According to RFC 3986, each part of a valid URI except the redundant and relative paths must be URI-encoded. **A URI must end with a slash (/) for signature calculation. However, the slash is not required when a request is sent.**

- **CanonicalQueryString**

  Query strings. The value ends with a newline character. If no query strings are configured, an empty string is used.

  Pay attention to the following to ensure valid query strings:

  - 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 an empty 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 a standard query string from the first parameter after sorting.

- **CanonicalHeaders**

  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. The value ends with a newline character.

  CAUTION

  - The local time on the client must be synchronized with the clock server to avoid a large offset in the value of the **X-Sdk-Date** request header.
  - API Gateway checks the time format and compares the time with the time when API Gateway receives the request. If the time difference exceeds 15 minutes, API Gateway will reject the request.

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

  - **Lowercase** is a function for converting all letters into lowercase letters.
  - **Trimall** is 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.
  - All headers are sorted in alphabetically ascending order.

  For example, the original headers are as follows:

  ```
  Host: service.region.example.com\n
  Content-Type: application/json;charset=utf8\n
  ```



```
My-header1:    a    b    c  \n
X-Sdk-Date:20190318T094751Z\n
My-Header2:    "x    y    \n
```

The message header names are converted into lowercase letters, the message headers are sorted in alphabetical order, and spaces before and after the header values are deleted. The standardized message headers are as follows:

```
content-type:application/json;charset=utf8\n
host:service.region.example.com\n
my-header1:a    b    c\n
my-header2:"x    y\n
x-sdk-date:20190318T094751Z\n
```

- **SignedHeaders**

  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.

  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.

- **RequestPayload**

  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** is 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.

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

Step 2: Creating 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.

The following is the pseudocode for creating a to-be-signed string:

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

- **Algorithm**

  Signature algorithm. For the SHA-256, the function is SDK-HMAC-SHA256.

- **RequestDateTime**

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

- **HashedCanonicalRequest**

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

  The pseudocode of the algorithm is as follows:


```
Lowercase(HexEncode(Hash.SHA256(CanonicalRequest)))
```

Step 3: Calculating the Signature

Use the SK and to-be-signed 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(Secret Access Key, string to sign))
```

- **HexEncode**

  HexEncode indicates the hexadecimal conversion.
- **HMAC**

  HMAC indicates the key-related hash operation.
- **Secret Access Key**

  Secret Access Key indicates the signature key SK.
- **string to sign**

  string to sign is the string created in **Step 2: Creating a To-Be-Signed String**.

Step 4: Adding the Signature to the Request Header

After calculating the signature, add the signature to the HTTP message header in Authorization for identity authentication.

The pseudocode for creating the Authorization header is as follows:

```
Authorization: algorithm Access=Access key, SignedHeaders=SignedHeaders, Signature=signature
```

There is no comma but a space between the algorithm and **Access**. **SignedHeaders** and **Signature** must be separated with commas.

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 cloud service for processing.

12.2 Example of the API Signature Authentication Mechanism

The following procedure uses the Virtual Private Cloud (VPC) query API as an example. Assume that the original request is as follows:

```
GET https://service.region.example.com/v1/77b6a44cba5143ab91d13ab9a8ff44fd/vpcs?
limit=2&marker=13551d6b-755d-4757-b956-536f674975c0 HTTP/1.1
Host: service.region.example.com
X-Sdk-Date: 20191115T033655Z
```

Step 1 Construct a standard request.

```
GET
/v1/77b6a44cba5143ab91d13ab9a8ff44fd/vpcs/
limit=2&marker=13551d6b-755d-4757-b956-536f674975c0
content-type:application/json
host:service.region.example.com
x-sdk-date:20191115T033655Z
```



```
content-type;host;x-sdk-date
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

- **HTTPRequestMethod**
  ```
  GET
  ```
- **CanonicalURI**

  The URI of the VPC query API is **/v1/{project_id}/vpcs**, where **project_id** is **77b6a44cba5143ab91d13ab9a8ff44fd**. The standard URI is as follows:
  ```
  /v1/77b6a44cba5143ab91d13ab9a8ff44fd/vpcs/
  ```
- **CanonicalQueryString**

  The VPC query API has two optional parameters: **limit** (number of records returned on each page) and **marker** (start VPC ID for pagination query). The standard query string is as follows:
  ```
  limit=2&marker=13551d6b-755d-4757-b956-536f674975c0
  ```
- **CanonicalHeaders**

  Requests for calling the VPC query API contain the **X-Sdk-Date**, **Host** (cloud service endpoint), and **Content-Type** headers. A standard header is as follows:
  ```
  content-type:application/json
  host:service.region.example.com
  x-sdk-date:20191115T033655Z
                          //This is a blank line.
  ```
- **SignedHeaders**

  Add the following three headers: **Content-Type**, **Host**, and **X-Sdk-Date**.
  ```
  content-type;host;x-sdk-date
  ```
- **RequestPayload**

  This example uses GET as an example, and the request body is empty. After hash processing, the request body (empty string) is as follows:
  ```
  e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
  ```

Step 2 Create a to-be-signed string.

```
SDK-HMAC-SHA256
20191115T033655Z
b25362e603ee30f4f25e7858e8a7160fd36e803bb2dfe206278659d71a9bcd7a
```

- **Algorithm**
  ```
  SDK-HMAC-SHA256
  ```
- **RequestDateTime**
  ```
  20191115T033655Z
  ```
- **HashedCanonicalRequest**

  Generate a hash value using the SHA-256 algorithm based on the standard request constructed in **Step 1**.
  ```
  b25362e603ee30f4f25e7858e8a7160fd36e803bb2dfe206278659d71a9bcd7a
  ```

Step 3 Calculate a signature.

```
Signature=f12f84a5ecf9eff3206499c4a55b13d1adad745dc8624a2e31f15c6b381d5b80
```

Assume that the SK is **MFyf***VmHc**. The signature value is obtained by performing the hash operation on the SK and the signature character string in **Step 2**.

```
signature = HexEncode(HMAC(MFyf***VmHc,
b25362e603ee30f4f25e7858e8a7160fd36e803bb2dfe206278659d71a9bcd7a))
```

Step 4 Add the signature to the request header.

Add the signature information to the Authorization message header. The SignedHeaders field includes the three headers from **Step 1**: **Content-Type**, **Host**, and **X-Sdk-Date**. Assume that the access key (AK) is **QTWA***KYUC.

```
Authorization: SDK-HMAC-SHA256 Access=QTWA***KYUC, SignedHeaders=content-type;host;x-sdk-date,
Signature=f12f84a5ecf9eff3206499c4a55b13d1adad745dc8624a2e31f15c6b381d5b80
```

Step 5 Complete signature request.

```
GET /v1/77b6a44cba5143ab91d13ab9a8ff44fd/vpcs?limit=2&; marker=13551d6b-755d-4757-
b956-536f674975c0 HTTP/1.1
Host: service.region.example.com
Content-Type: application/json
x-sdk-date: 20191115T033655Z
Authorization: SDK-HMAC-SHA256 Access=QTWA***KYUC, SignedHeaders=content-type;host;x-sdk-date,
Signature=f12f84a5ecf9eff3206499c4a55b13d1adad745dc8624a2e31f15c6b381d5b80
```

----End

Example request for calling an API with a curl command:

```
curl -X GET "https://service.region.example.com/v1/77b6a44cba5143ab91d13ab9a8ff44fd/vpcs?
limit=2&marker=13551d6b-755d-4757-b956-536f674975c0" -H "content-type: application/json" -H "X-Sdk-
Date: 20191115T033655Z" -H "host: service.region.example.com" -H "Authorization: SDK-HMAC-SHA256
Access=QTWA***KYUC, SignedHeaders=content-type;host;x-sdk-date,
Signature=f12f84a5ecf9eff3206499c4a55b13d1adad745dc8624a2e31f15c6b381d5b80" -d $''
```


13 Error Codes

If an error code starting with a cloud service name is displayed when you call an API, rectify the fault by referring to section "Error Code" in the API reference manual of the cloud service. If an error code starting with APIGW is displayed when you call an API, seek solutions in the following table.

Table 13-1 Error codes

Error Code	Error Message	HTTP Status Code	Description	Corrective Action
APIGW.0101	The API does not exist or has not been published in the environment.	404	The API does not exist or has not been published in the environment.	Rectify the fault by following the instructions in **What Should I Do If "The API does not exist or has not been published in the environment." Is Displayed?**.
APIGW.0101	The API does not exist.	404	The request method does not exist.	Check whether the request method is the same as the method specified for the API.
APIGW.0103	The backend does not exist.	404	The backend service is not found.	Contact technical support.
APIGW.0104	The plug-ins do not exist.	400	No plugin configurations are found.	Contact technical support.

Error Code	Error Message	HTTP Status Code	Description	Corrective Action
APIGW.0105	The backend configurations do not exist.	400	No backend configurations are found.	Contact technical support.
APIGW.0106	Orchestration error.	400	Orchestration error.	Check whether the frontend and backend parameters are properly set for the API.
APIGW.0201	API request error.	400	Invalid request parameters.	Set valid request parameters.
APIGW.0201	Request entity too large.	413	The request body exceeds 12 MB.	Reduce the size of the request body.
APIGW.0201	Request URI too large.	414	The request URI is too large.	Reduce the size of the request URI.
APIGW.0201	Request headers too large.	494	The request headers are too large.	Reduce the size of the request headers.
APIGW.0201	Backend unavailable.	502	The backend service is currently unavailable.	Check whether the backend address configured for the API is accessible.
APIGW.0201	Backend timeout.	504	The backend service timed out.	Increase the timeout duration of the backend service or shorten the processing time.
APIGW.0301	Incorrect IAM authentication information.	401	The IAM authentication information is incorrect.	Rectify the fault by following the instructions in **Common Errors Related to IAM Authentication Information**.
APIGW.0302	The IAM user is not authorized to access the API.	403	The IAM user is not allowed to access the API.	Check whether the user has been blacklisted or whitelisted.

Error Code	Error Message	HTTP Status Code	Description	Corrective Action
APIGW.0303	Incorrect app authentication information.	401	The app authentication information is incorrect.	Perform the following checks for app authentication: • Check whether the request method, path, query parameters, and request body are consistent with those used for signing. • Check whether the client time is correct. Check whether the signing code is correct by referring to section "Calling APIs Through App Authentication" of the *Developer Guide*. In the case of AppCode-based simple authentication, check whether the request contains the **X-Apig-AppCode** header.
APIGW.0304	The app is not authorized to access the API.	403	The app is not allowed to access the API.	Check whether the app has been authorized to access the API.
APIGW.0305	Incorrect authentication information.	401	The authentication information is incorrect.	Check whether the authentication information is correct.

Error Code	Error Message	HTTP Status Code	Description	Corrective Action
APIGW.0306	API access denied.	403	Access to the API is not allowed.	Check whether you have been authorized to access the API.
APIGW.0307	The token must be updated.	401	The token needs to be updated.	• Obtain a new token from IAM. • An API of another region may have been called. Check the API URL.
APIGW.0308	The throttling threshold has been reached.	429	The request throttling threshold is reached.	• Try again after the throttling resumes. By default, an API can be accessed a maximum of 200 times per second. • The rate limits of cloud service APIs cannot be adjusted. Try again after the throttling resumes. • To adjust the rate limit of an API you have created in API Gateway, contact technical support by submitting a service ticket.
APIGW.0310	The project is unavailable.	403	The project is currently unavailable.	Select another project and try again.

Error Code	Error Message	HTTP Status Code	Description	Corrective Action
APIGW.0311	Incorrect debugging authentication information.	401	The debugging authentication information is incorrect.	Contact technical support.
APIGW.0401	Unknown client IP address.	403	The client IP address cannot be identified.	Contact technical support.
APIGW.0402	The IP address is not authorized to access the API.	403	The IP address is not allowed to access the API.	Check whether the IP address has been blacklisted or whitelisted.
APIGW.0404	Access to the backend IP address has been denied.	403	The backend IP address cannot be accessed.	The backend IP address or the IP address of the backend domain cannot be accessed. Check whether the IP address exists or has been blacklisted or whitelisted.
APIGW.0501	The app quota has been used up.	405	The app quota has been reached.	Increase the app quota.
APIGW.0502	The app has been frozen.	405	The app has been frozen.	Account balance is insufficient. Go to the **Funds Management** page to top up your account.
APIGW.0601	Internal server error.	500	Internal error.	Contact technical support.
APIGW.0602	Bad request.	400	Invalid request.	Check whether the request is valid.
APIGW.0605	Domain name resolution failed.	500	Domain name resolution failed.	Check whether the domain name is correct and has been bound to a correct backend address.

Error Code	Error Message	HTTP Status Code	Description	Corrective Action
APIGW.0606	Failed to load the API configurations.	500	API configurations are not loaded.	Contact technical support.
APIGW.0607	The following protocol is supported: {xxx}	400	The protocol is not supported. Only *xxx* is supported. *xxx* indicates the protocol in a response.	Use HTTP or HTTPS to access the API.
APIGW.0608	Failed to obtain the admin token.	500	The tenant information cannot be obtained.	Contact technical support.
APIGW.0609	The VPC backend does not exist.	500	The VPC backend service cannot be found.	Contact technical support.
APIGW.0610	No backend available.	502	No backend services are available.	Check whether all backends are available.
APIGW.0611	The backend port does not exist.	500	The backend port is not found.	Contact technical support.
APIGW.0612	An API cannot call itself.	500	An API cannot call itself.	Modify the backend configurations, and ensure that the number of layers the API is recursively called does not exceed 10.
APIGW.0613	The IAM service is currently unavailable.	503	IAM is currently unavailable.	Contact technical support.
APIGW.0705	Backend signature calculation failed.	500	Backend signature calculation failed.	Contact technical support.

Error Code	Error Message	HTTP Status Code	Description	Corrective Action
APIGW.0801	The service is unavailable in the currently selected region.	403	The service is inaccessible in the current region.	Check whether the service supports cross-region access.
APIGW.0802	The IAM user is forbidden in the currently selected region.	403	The IAM user is not allowed to access the region.	Contact technical support.

14 FAQs

14.1 How Do I Call APIs in a Subproject?

To access resources in a subproject by calling APIs, add the **X-Project-Id** parameter to the request header and set the parameter value to the subproject ID.

For details about how to obtain the value of **X-Project-Id**, see **Obtaining API Information**.

14.2 Does APIG Support Persistent Connections?

Yes.

But you should use them properly to avoid occupying too many resources.

14.3 Must the Request Body Be Signed?

No. If you do not want to sign the request body, add the following parameter and value to the message header:

X-Sdk-Content-Sha256:UNSIGNED-PAYLOAD

UNSIGNED-PAYLOAD indicates the hashed position value calculated based on the request body.

14.4 Are Request Header Parameters Required for Signing Requests?

If you sign API requests by following the instructions in **API Signature Authentication Mechanism**, only **X-Sdk-Date** is required and other request header parameters are optional.

If you use an SDK provided by Huawei Cloud to sign API requests, you do not need to consider which request header parameters are required. The SDK determines which parameters are required and automatically generates the signature

information. If the value of a request header parameter changes after requests are signed, assign a value to the parameter again after signing.

14.5 How Do I Use a Temporary AK/SK to Sign Requests?

Integrate the signing SDK into your application and add the following parameter and value to the message header:

X-Security-Token:*{securityToken}*

Then, use a temporary AK/SK to sign the request. The signing SDK can be the SDK used for AK/SK-based authentication.

Step 1 Obtain a temporary AK/SK and *{securityToken}*. For details, see the *Identity and Access Management API Reference*.

A response similar to the following is displayed:

```
{
   "credential": {
      "access": "P0HE****69X0",
      "expires_at": "2022-10-17T18:51:25.231000Z",
      "secret": "3WJu****hDVs",
      "securitytoken": "XXXXXX......"
   }
}
```

Step 2 Construct a request with signature parameters.

```
...
request.setKey("P0HE****69X0");
request.setSecret("3WJu****hDVs");
request.setMethod("GET");
request.setUrl("url");
request.addHeader("X-Security-Token", "XXXXXX.....");
...
```

----End

14.6 Common Errors Related to IAM Authentication Information

You may encounter the following errors related to IAM authentication information:

- **Incorrect IAM authentication information: AK access failed to reach the limit, forbidden**
- **Incorrect IAM authentication information: decrypt token fail**
- **Incorrect IAM authentication information: Get secretKey failed**

Incorrect IAM authentication information: AK access failed to reach the limit, forbidden

```
{
  "error_msg": "Incorrect IAM authentication information: AK access failed to reach the limit,forbidden." .....
  "error_code": "APIGW.0301",
```



```
 "request_id": "******"
}
```

Possible Causes

- The AK and SK do not match.
- AK/SK authentication fails for more than five consecutive times, and the AK/SK pair is locked for five minutes. (Authentication requests are rejected within this period).

Solution

- Check whether the SK is correct.
- Try again 5 minutes later.

Incorrect IAM authentication information: decrypt token fail

```
{
 "error_msg": "Incorrect IAM authentication information: decrypt token fail",
 "error_code": "APIGW.0301",
 "request_id": "******"
}
```

Possible Cause

The token cannot be parsed for IAM authentication of the API.

Solution

- Check whether the token is correct.
- Check whether the token has been obtained in the environment where the API is called.

Incorrect IAM authentication information: Get secretKey failed

```
{
"error_msg": "Incorrect IAM authentication information: Get secretKey failed,ak:******,err:ak not exist",
"error_code": "APIGW.0301",
"request_id": "******"
}
```

Possible Cause

The AK used for IAM authentication of the API does not exist.

Solution

Check whether the AK is correct.

14.7 What Should I Do If "The API does not exist or has not been published in the environment." Is Displayed?

If an open API of a cloud service failed to be called, troubleshoot the failure by performing the following operations:

1. The domain name, request method, or path used for calling the API is incorrect.
   - For example, an API created using the POST method is called with GET.

- Missing a slash (/) in the access URL will lead to a failure in matching the URL in the API details. For example, URLs **https://vpc.region.cloud.com/test/** and **https://vpc.region.cloud.com/test** represent two different APIs.

2. The domain name is resolved incorrectly. If the domain name, request method, and path for calling the API are correct, the API may not be correctly resolved.

A Change History

Table A-1 Change history

Date	Description
2022-09-08	This issue is the first official release.

