API 签名指南

文档版本 01

发布日期 2023-09-11

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目 录

1 概述

本手册将介绍如何使用AK/SK签名认证方式调用通过API网关开放的云服务API，提供签名流程与实现逻辑，以及Java、Go、Python、C等多种不同语言的签名SDK和调用示例。

说明

1. 部分云服务开放的API，不通过API网关，签名认证流程请先参考云服务自身提供的API参考手册。

   各云服务API参考手册中的“如何调用API”章节，介绍了认证方法。
2. SDK打包在示例中，可单独获取SDK，然后参考示例与各语言的API调用说明部分，将SDK集成到您的应用中。
3. 如果本手册的多语言签名示例没有涵盖您使用的编程语言，请根据**签名流程与算法**，自主实现请求的签名。
4. API调用的另一种认证方式为携带Token，Token方式的说明与示例包含在各服务的API参考手册。
5. AK/SK签名认证方式，仅支持Body体大小为12M以内，12M以上的请求，需使用Token认证。
6. 云服务具体的API在各云服务的API参考手册中列明。
7. 客户端须注意本地时间与时钟服务器的同步，避免请求消息头X-Sdk-Date的值出现较大误差。

   API网关除了校验时间格式外，还会校验该时间值与网关收到请求的时间差，如果时间差超过15分钟，API网关将拒绝请求。

2 AK/SK 签名认证算法详解

2.1 AK/SK 签名认证流程

客户端涉及的AK/SK签名以及请求发送的流程概述如下：

1. **构造规范请求**。
   将待发送的请求内容按照与API网关后台约定的规则组装，确保客户端签名、API网关后台认证时使用的请求内容一致。
2. 使用规范请求和其他信息**创建待签字符串**。
3. 使用AK/SK和待签字符串**计算签名**。
4. 将生成的**签名信息作为请求消息头添加到HTTP请求**中，或者作为查询字符串参数添加到HTTP请求中。

以下做详细介绍。

说明

本章节主要用于帮助用户自行完成API请求的签名。

签名SDK与demo章节提供了常用语言的签名SDK以及demo，与本章节提供的算法逻辑一致，您可以直接使用，省去自签名步骤。

2.2 构造规范请求

使用AK/SK方式进行签名与认证，首先需要规范请求内容，然后再进行签名。客户端与云服务API网关使用相同的请求规范，可以确保同一个HTTP请求的前后端得到相同的签名结果，从而完成身份校验。

HTTP请求规范伪代码如下：


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

以虚拟私有云服务的一个查询VPC列表接口为例，说明规范请求的构造步骤。

假设原始请求：

```
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
```

步骤1 构造**HTTP请求方法**（**HTTPRequestMethod**），以换行符结束。

HTTP请求方法，如GET、PUT、POST等。请求方法示例：

```
GET
```

步骤2 添加规范**URI参数**（**CanonicalURI**），以换行符结束。

释义：

规范URI，即请求资源路径，是URI的绝对路径部分的URI编码。

格式：

根据RFC 3986规范化URI路径，移除冗余和相对路径部分，路径中每个部分必须为URI编码。如果URI路径不以“/”结尾，则在尾部添加“/”。

举例：

URI具体见云服务的API参考手册，每个API章节都提供对应的资源路径。如虚拟私有云服务的查询VPC列表：/v1/{project_id}/vpcs，此时规范的URI编码为：

```
GET
/v1/77b6a44cba5143ab91d13ab9a8ff44fd/vpcs/
```

说明

计算签名时，URI必须以“/”结尾。发送请求时，可以不以“/”结尾。

步骤3 添加**规范查询字符串**（**CanonicalQueryString**），以换行符结束。

释义：

查询字符串，即查询参数。如果没有查询参数，则为空字符串，即规范后的请求为**空行**。

格式：

规范查询字符串需要满足以下内容：

- 根据以下规则对每个参数名和值进行URI编码：
  - 请勿对RFC 3986定义的任何非预留字符进行URI编码，这些字符包括：A-Z、a-z、0-9、-、_、.和~。
  - 使用%XY对所有非预留字符进行百分比编码，其中X和Y为十六进制字符（0-9和A-F）。例如，空格字符必须编码为%20，扩展UTF-8字符必须采用“%XY%ZA%BC”格式。

- 对于每个参数，追加“*URI编码的参数名称=URI编码的参数值*”。如果没有参数值，则以空字符串代替，但不能省略“=”。

  例如以下含有两个参数，其中第二个参数parm2的值为空。

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

- 按照字符代码以升序顺序对参数名进行排序。例如，以大写字母F开头的参数名排在以小写字母b开头的参数名之前。
- 以排序后的第一个参数名开始，构造规范查询字符串。

举例：

查询VPC列表有两个可选参数：limit（每页返回的个数）、marker（分页查询的起始VPC资源ID）

```
GET
/v1/77b6a44cba5143ab91d13ab9a8ff44fd/vpcs/
limit=2&marker=13551d6b-755d-4757-b956-536f674975c0
```

步骤4 添加规范**消息头**（**CanonicalHeaders**），以换行符结束。

释义：

规范消息头，即请求消息头列表。包括签名请求中的所有HTTP消息头列表。消息头必须包含X-Sdk-Date，用于校验签名时间，格式为ISO8601规范的UTC时间格式：YYYYMMDDTHHMMSSZ。

⚠ 注意

客户端须注意本地时间与时钟服务器的同步，避免请求消息头X-Sdk-Date的值出现较大误差。

API网关除了校验时间格式外，还会校验该时间值与网关收到请求的时间差，如果时间差超过15分钟，API网关将拒绝请求。

格式：

CanonicalHeaders由多个请求消息头共同组成，**CanonicalHeadersEntry0 + CanonicalHeadersEntry1 + ...**，其中每个请求消息头（CanonicalHeadersEntry ）的格式为**Lowercase(HeaderName) + ':' + Trimall(HeaderValue) + '\n'**

说明

- Lowercase表示将所有字符转换为小写字母的函数。
- Trimall表示删除值前后的多余空格的函数。
- 最后一个请求消息头也会携带一个换行符。叠加规范中**CanonicalHeaders**自身携带的换行符，因此会出现一个空行。

举例：

查询VPC列表的消息头，需要包含签名时间（X-Sdk-Date），云服务Endpoint（Host）、内容类型（Content-Type）。

```
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
```


须知

规范消息头需要满足以下内容：

- 将消息头名称转换为小写形式，并删除前导空格和尾随空格。
- 按照字符代码对消息头名称进行升序排序。

例如原始消息头为：

```
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
```

对消息头名称转小写，按消息头名称字符代码对消息头排序，将消息头的值去掉前导空格与尾随空格。最终得到规范消息头：

```
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
```

步骤5 添加用于**签名的消息头声明**（**SignedHeaders**），以换行符结束。

释义：

用于签名的请求消息头列表。通过添加此消息头，向API网关告知请求中哪些消息头是签名过程的一部分，以及在验证请求时API网关可以忽略哪些消息头。X-Sdk-date必须作为已签名的消息头。

格式：

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

已签名的消息头需要满足的内容：将已签名的消息头名称转换为小写形式，按照字符代码对消息头进行排序，并使用“;”来分隔多个消息头。

Lowercase表示将所有字符转换为小写字母。

举例：

以下表示有三个消息头参与签名：Content-Type、Host、X-Sdk-Date

```
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
```

签名后消息头将为：

```
SignedHeaders=content-type;host;x-sdk-date
```

消息头添加到请求的具体示例请参考**添加签名信息到请求头**。

步骤6 使用SHA 256哈希函数以基于HTTP或HTTPS请求正文中的body体（**RequestPayload**），创建哈希值。

释义：

请求消息体。消息体需要做两层转换：HexEncode(Hash(*RequestPayload*))，其中Hash表示生成消息摘要的函数，当前支持SHA-256算法。HexEncode表示以小写字母形式返回摘要的Base-16编码的函数。例如，HexEncode("m") 返回值为“6d”而不是“6D”。输入的每一个字节都表示为两个十六进制字符。

说明

计算RequestPayload的哈希值时，对于“RequestPayload==null”的场景，直接使用空字符串""来计算。

举例：

本示例为GET方法，body体为空。经过哈希处理的body（空字符串）如下：

```
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
```

至此，规范请求构造完成。

步骤7 对构造好的规范请求进行哈希处理，算法与对RequestPayload哈希处理的算法相同。经过哈希处理的规范请求必须以小写十六进制字符串形式表示。

算法伪代码：**Lowercase(HexEncode(Hash.SHA256(CanonicalRequest)))**

经过哈希处理的规范请求示例：

```
b25362e603ee30f4f25e7858e8a7160fd36e803bb2dfe206278659d71a9bcd7a
```

----结束

2.3 创建待签字符串

对HTTP请求进行规范并取得请求的哈希值后，将其与签名算法、签名时间一起组成待签名字符串。

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

伪代码中参数说明如下。

- **Algorithm**

  签名算法。对于SHA 256，算法为SDK-HMAC-SHA256。

- **RequestDateTime**

  请求时间戳。与请求消息头X-Sdk-Date的值相同，格式为YYYYMMDDTHHMMSSZ。

- **HashedCanonicalRequest**

  将**构造规范请求**中构造的规范请求以SHA256算法生成的hash值。

上述例子得到的待签字符串为：

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


2.4 计算签名

将SK（Secret Access Key）和创建的待签字符串作为加密哈希函数的输入，计算签名，将二进制值转换为十六进制表示形式。

伪代码如下：

```
signature = HexEncode(HMAC(Secret Access Key, string to sign))
```

其中HMAC指密钥相关的哈希运算，HexEncode指转十六进制。伪代码中参数说明如表2-1所示。

表 2-1 参数说明

参数名称	参数解释
Secret Access Key	签名密钥
string to sign	创建的待签字符串

假设Secret Access Key为MFyfvK41ba2giqM7Uio6PznpdUKGpownRZlmVmHc，则计算得到的signature为：

```
7be6668032f70418fcc22abc52071e57aff61b84a1d2381bb430d6870f4f6ebe
```

2.5 添加签名信息到请求头

在计算签名后，将它添加到Authorization的HTTP消息头。Authorization消息头未包含在已签名消息头中，主要用于身份验证。

伪代码如下：

```
Authorization header创建伪码：
Authorization: algorithm Access=Access key, SignedHeaders=SignedHeaders, Signature=signature
```

需要注意的是算法与Access之前有空格但没有逗号，但是SignedHeaders与Signature之前需要使用逗号隔开。

得到的签名消息头为：

```
SDK-HMAC-SHA256 Access=QTWAOYTTINDUT2QVKYUC, SignedHeaders=content-type;host;x-sdk-date,
Signature=7be6668032f70418fcc22abc52071e57aff61b84a1d2381bb430d6870f4f6ebe
```

得到签名消息头后，将其增加到原始HTTP请求内容中，请求将被发送给云服务API网关，由API网关完成身份认证。身份认证通过后，该请求才会发送给具体的云服务进行业务处理。

包含签名信息的完整请求如下：

```
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=QTWAOYTTINDUT2QVKYUC, SignedHeaders=content-type;host;x-sdk-date, Signature=7be6668032f70418fcc22abc52071e57aff61b84a1d2381bb430d6870f4f6ebe
```

Curl方式样例如下：

```
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=QTWAOYTTINDUT2QVKYUC, SignedHeaders=content-type;host;x-sdk-date, Signature=7be6668032f70418fcc22abc52071e57aff61b84a1d2381bb430d6870f4f6ebe" -d $''
```


3 AK/SK 签名认证操作指导

3.1 AK/SK 签名认证操作流程

AK/SK签名认证操作流程如下：

1. API调用信息收集。
   需要获取以下信息，包括：
   - 用于组成请求URL的Endpoint和URI。
   - 用于签名和认证的AK/SK。
   - 用于区分租户的项目ID、子项目ID。
   - 用于区分租户的帐号名、帐号ID。

   表 3-1 信息收集项

信息项	说明
Endpoint	地区与终端节点，即云服务在不同Region有不同的访问域名。 获取方式请参考**获取EndPoint**。
URI	API接口的调用路径及参数。 请参考各云服务的详细接口章节获取。

信息项	说明
AK/SK	访问密钥对，包含密钥ID与密钥。AK/SK用于对请求内容进行签名。 获取方式请参考**获取AK/SK**。
Project_Id	项目ID，在大多数API接口调用时需要配置在URI中，用以识别不同的项目。 获取方式请参考**获取项目ID**。
X-Project-Id	子项目ID，在多项目场景中使用。如果云服务资源创建在子项目中，AK/SK认证方式下，操作该资源的接口调用需要在请求消息头中携带X-Project-Id。 获取方式请参考**获取项目ID**。
X-Domain-Id	帐号ID，用途： ● Token认证方式下获取Token。 ● AK/SK认证方式下，全局服务的接口调用，需在请求消息头中配置X-Domain-Id。（全局服务：服务部署时不区分物理区域。如OBS、CDN等。） 获取方式请参考**获取帐号名和帐号ID**。

2. API调用。

   本手册提供Java、Go、Python、C等多种不同语言的签名SDK和调用示例，您可以从**签名SDK与demo**中选择需要的语言，然后参考示例与API调用说明部分，将SDK集成到您的应用中。

   说明

   API选择华为IAM认证，也支持临时AK/SK，具体请参考**使用临时AK/SK做签名**。

3.2 获取 EndPoint

地区与终端节点，即云服务在不同Region有不同的访问域名。

获取方式请参考**地区和终端节点**。

说明

本手册所有关于请求URL的举例，其中Endpoint以service.region.example.com为例。

3.3 获取 AK/SK

如果已生成过AK/SK，则可跳过此步骤，找到原来已下载的AK/SK文件，文件名一般为：credentials.csv。

如下图所示，AK（Access Key Id），SK（Secret Access Key）。

图 3-1 credential.csv 文件内容

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

操作步骤

步骤1 登录**控制台**。

步骤2 将鼠标移至页面右上角的用户名处，在下拉列表中单击“我的凭证”。

步骤3 单击“访问密钥”。

步骤4 单击“新增访问密钥”，进入“新增访问密钥”页面。

步骤5 输入描述信息，单击“确定”，下载密钥，请妥善保管。

图 3-2 获取访问密钥

说明

获取临时AK/SK，请参考**IAM接口文档**。

----结束

3.4 获取项目 ID

在调用接口的时候，部分URL中需要填入项目编号，获取token时，同样需要获取项目编号，所以需要先在管理控制台上获取到项目编号。项目编号获取步骤如下：

1. 登录**控制台**。
2. 将鼠标移至页面右上角的用户名处，在下拉列表中单击“我的凭证”，查看“项目ID”。

   项目用于对云服务器资源进行物理隔离，默认有region级别的隔离，也可在Region下**建立多项目**，做更细级别的隔离。因此，请参考下图，在右侧列表中找到您的服务器资源对应的Region（所属区域），在其左侧列表中获取项目ID。

图 3-3 查看项目 ID

说明

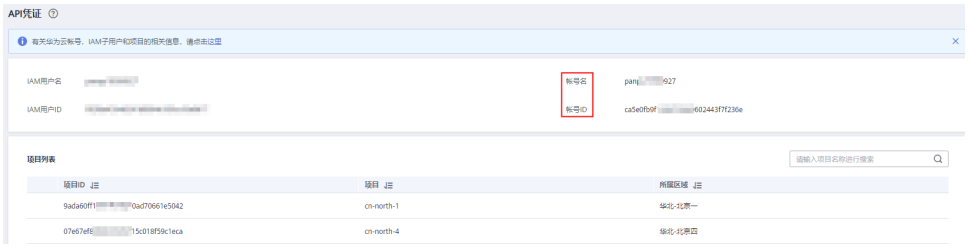
如果要查看子项目ID，请单击该项目，展开子项目列表后获取。

3.5 获取帐号名和帐号 ID

在调用接口的时候，部分URL中需要填入帐号名和帐号ID，所以需要先在管理控制台上获取到帐号名和帐号ID。帐号名和帐号ID获取步骤如下：

1. 登录**控制台**。
2. 将鼠标移至页面右上角的用户名处，在下拉列表中单击“我的凭证”。
   查看帐号名和帐号ID。

图 3-4 查看帐号名和帐号 ID



3.6 签名 SDK 与 demo

3.6.1 Java

本节以IDEA工具为例，介绍如何在Java环境中集成API请求签名的SDK。您可以直接导入示例代码，然后参考调用说明部分将签名SDK集成到您的应用中。

说明

签名SDK只包含签名功能，不包含云服务的SDK功能，云服务SDK请参见**SDK**。

准备环境

- 获取并安装IDEA，可至**IDEA官方网站**下载可执行文件进行安装，或者下载全量压缩包并解压后直接使用。
- JDK：Java Development Kit 1.8.111及以上版本，可至**Oracle官方下载页面**下载。暂不支持Java Development Kit 17或以上版本。

- Maven仓地址：**https://repo.huaweicloud.com/apache/maven/maven-3/**。

获取 SDK

点此下载SDK与Demo。

解压后目录结构如下：

名称	说明
libs\java-sdk-core-x.x.x.jar	签名SDK
pom.xml	构建Maven工程所需，定义其他依赖包
changelog	变更日志
src	验证签名SDK的demo代码: • WebSocketDemo.java • OkHttpDemo.java、 • LargeFileUploadDemo.java • HttpClientDemo.java 引用类: • Constant.java • SSLCipherSuiteUtil.java • UnsupportProtocolException.java

如果使用maven构建，可以使用SDK包中的libs\java-sdk-core-x.x.x.jar，也可以使用maven镜像仓库中的java-sdk-core-x.x.x.jar，maven仓库地址为**https://mirrors.huaweicloud.com/repository/maven/huaweicloudsdk/com/huawei/apigateway/java-sdk-core/**，配置maven源的方法可参见**https://bbs.huaweicloud.com/forum/forum.php?mod=viewthread&tid=1779**。

加入java-sdk-core依赖的maven配置项为：

```
<dependency>
    <groupId>com.huawei.apigateway</groupId>
    <artifactId>java-sdk-core</artifactId>
    <version>SDK包版本号</version>
</dependency>
```


说明

使用maven构建时，settings.xml文件需要修改，增加以下内容：

1. 在profiles节点中添加如下内容：

```
<profile>
    <id>MyProfile</id>
    <repositories>
        <repository>
            <id>HuaweiCloudSDK</id>
            <url>https://mirrors.huaweicloud.com/repository/maven/huaweicloudsdk/</url>
            <releases>
                <enabled>true</enabled>
            </releases>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
    </repositories>
    <pluginRepositories>
        <pluginRepository>
            <id>HuaweiCloudSDK</id>
            <url>https://mirrors.huaweicloud.com/repository/maven/huaweicloudsdk/</url>
            <releases>
                <enabled>true</enabled>
            </releases>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </pluginRepository>
    </pluginRepositories>
</profile>
```

2. 在mirrors节点中增加：

```
<mirror>
    <id>huaweicloud</id>
    <mirrorOf>*,!HuaweiCloudSDK</mirrorOf>
    <url>https://repo.huaweicloud.com/repository/maven/</url>
</mirror>
```

3. 增加activeProfiles标签激活配置：

```
<activeProfiles>
    <activeProfile>MyProfile</activeProfile>
</activeProfiles>
```

IDEA 的配置方式

支持以下几种配置方式，用户可根据业务需求选择。

- **导入示例代码**
- **创建IDEA maven工程**

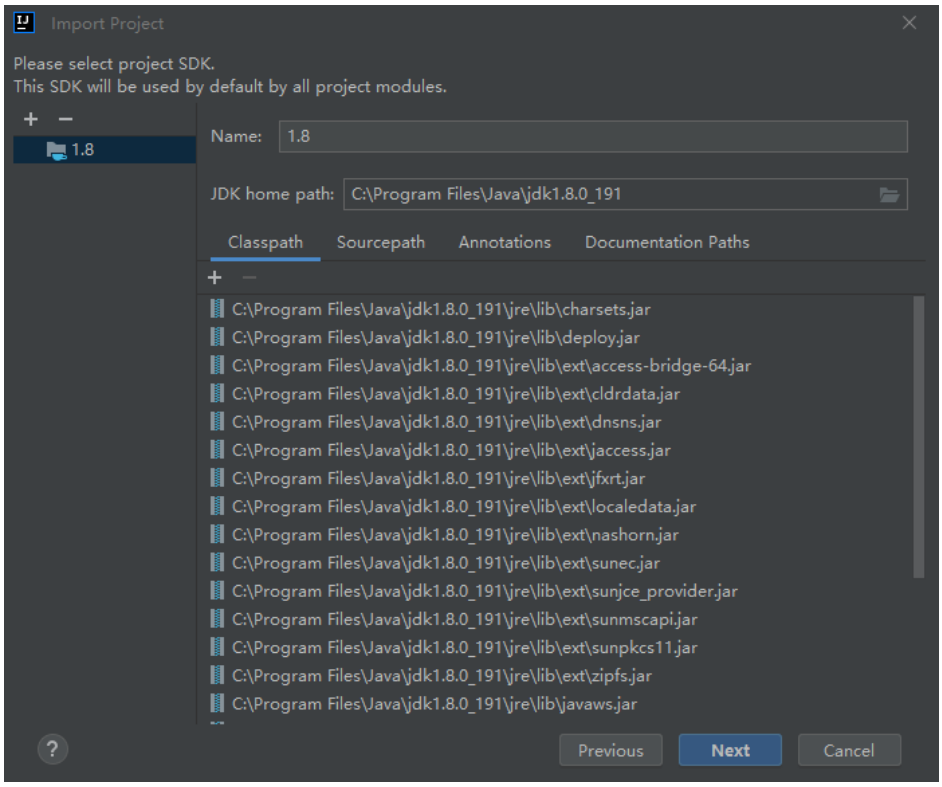
1. **导入示例代码**

   a. 打开IDEA，在菜单栏选择“File > New > Project from Existing Sources”。

   选择解压后的“APIGW-java-sdk-*x.x.x*”文件夹，单击“OK”，导入示例工程。

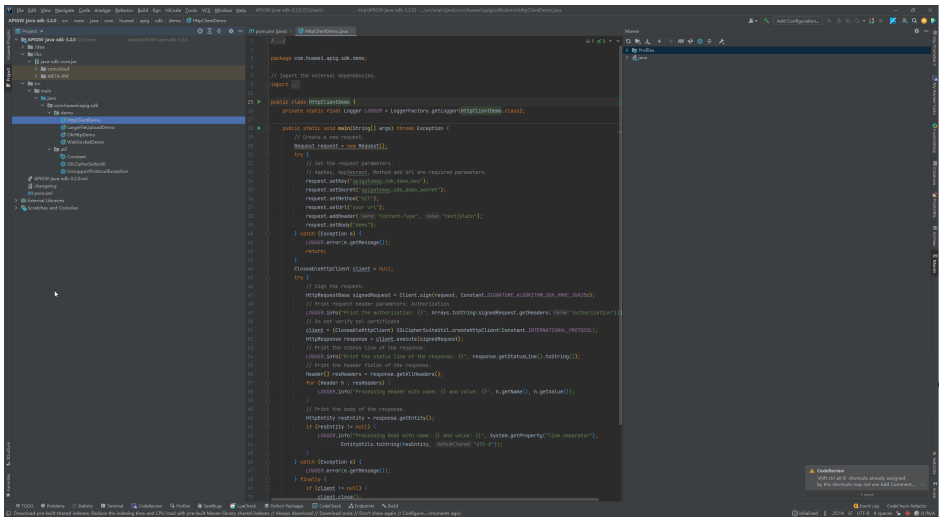
   b. 在“Import Project”页面，选择“Create project from existing sources”。

   连续单击“Next”，并在此页面选择JDK版本“1.8”，然后单击“Next”，最后单击“Create”。



c. IDEA支持在当前窗口或新窗口创建工程。此处，在弹窗中单击“New Window”。

d. 在左侧工程，右键单击工程名称，选择“Add Framework Support”，选择Maven，单击“OK”。



2. **创建IDEA maven工程**

a. 打开IDEA，在菜单栏选择“File > NEW > Project”。

b. 在弹窗中选择“Maven Archetype”，填写并选择以下参数后，单击“Create”。

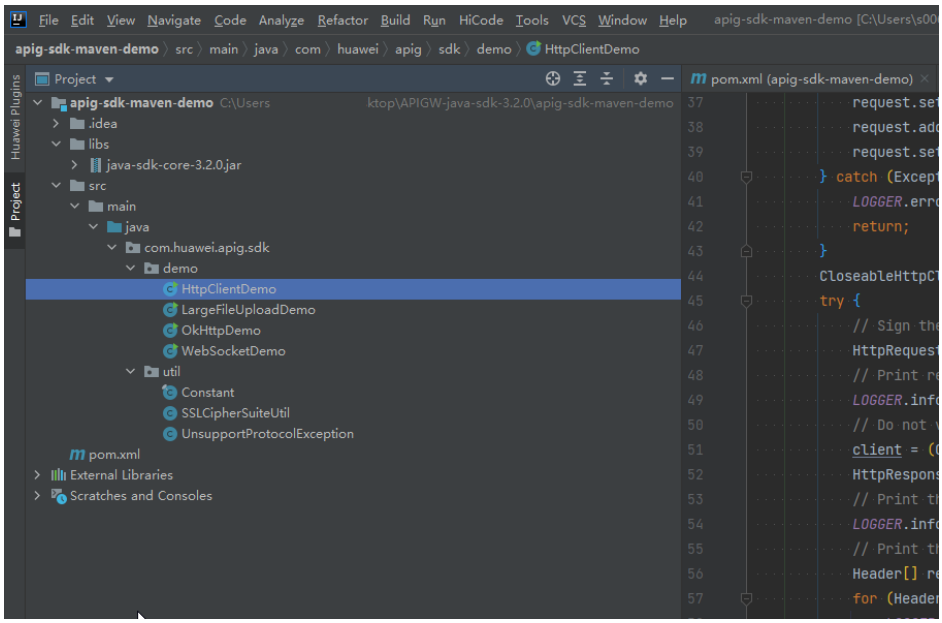
Name：填写“apig-sdk-maven-demo”。

Archetype：选择“org.apache.tapestry:quickstart”。

JDK：用户自己的版本。

c. IDEA支持在当前窗口或新窗口创建工程。此处，在弹窗中单击“New Window”。

d. 把示例工程中的“src”和“libs”文件夹复制到apig-sdk-maven-demo工程下。



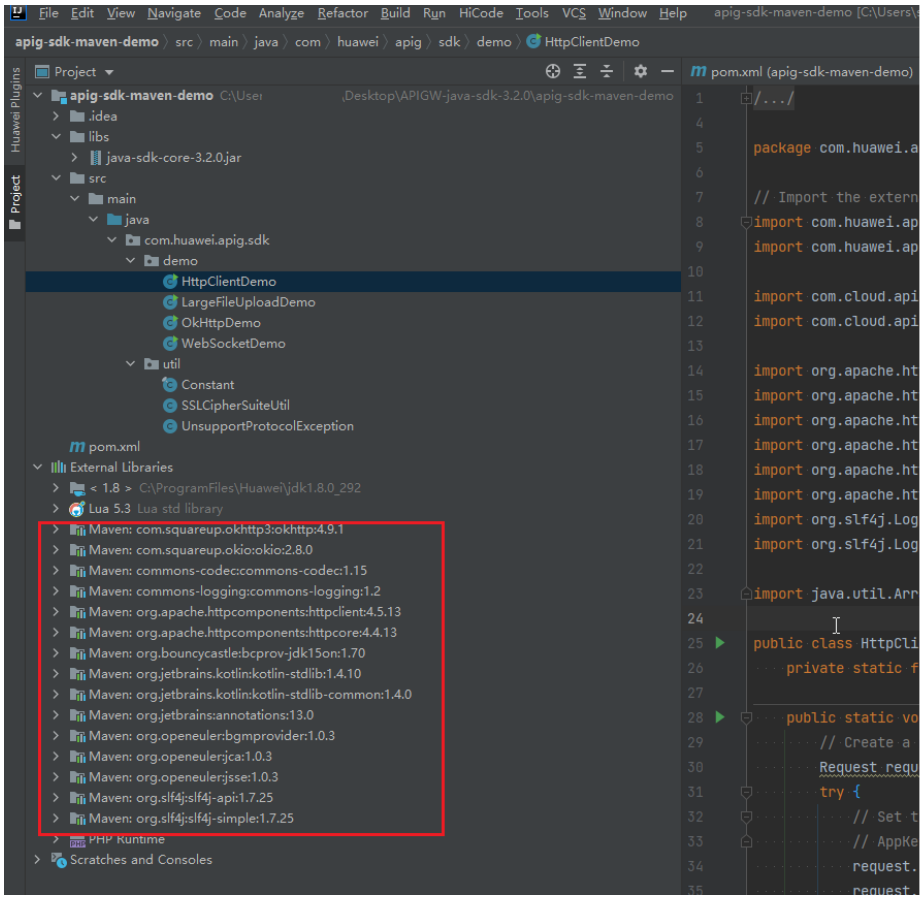
e. 配置新建Maven工程的pom.xml文件。

在左侧展开工程文件，双击“pom.xml”将以下内容复制粘贴替换原有内容。

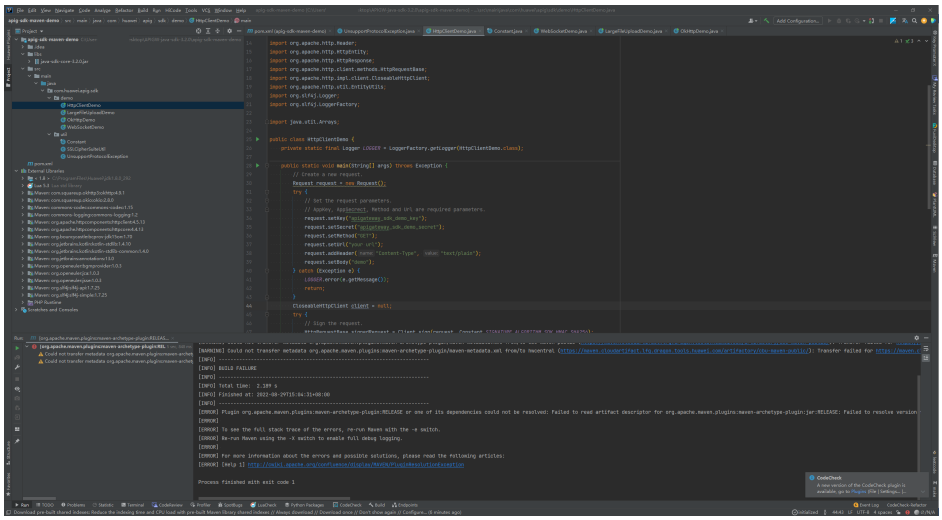
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.huawei.apigateway</groupId>

    <artifactId>java</artifactId>
    <version>1.0.0</version>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.0</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                    <encoding>UTF-8</encoding>
                </configuration>
            </plugin>
        </plugins>
    </build>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>commons-codec</groupId>
            <artifactId>commons-codec</artifactId>
```

```
        <version>1.15</version>
    </dependency>
    <dependency>
        <groupId>commons-logging</groupId>
        <artifactId>commons-logging</artifactId>
        <version>1.2</version>
    </dependency>
    <dependency>
        <groupId>org.apache.httpcomponents</groupId>
        <artifactId>httpclient</artifactId>
        <version>4.5.13</version>
    </dependency>
    <dependency>
        <groupId>com.squareup.okhttp3</groupId>
        <artifactId>okhttp</artifactId>
        <version>4.9.1</version>
    </dependency>
    <dependency>
        <groupId>org.apache.httpcomponents</groupId>
        <artifactId>httpcore</artifactId>
        <version>4.4.13</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>1.7.25</version>
        <scope>compile</scope>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-simple</artifactId>
        <version>1.7.25</version>
    </dependency>
    <dependency>
        <systemPath>${project.basedir}/libs/java-sdk-core-3.2.0.jar</systemPath>
        <groupId>com.huawei.apigateway</groupId>

    <artifactId>java-sdk-core</artifactId>
    <version>3.2.0</version>
    <scope>system</scope>
    </dependency>
    <dependency>
        <groupId>org.openeuler</groupId>
        <artifactId>bgmprovider</artifactId>
        <version>1.0.3</version>
    </dependency>
  </dependencies>

</project>
```

f. 下载Maven依赖，选择“pom.xml”并右键选择“New > Maven > Reload project”。



g. 在左侧工程下展开“src”文件，双击“HttpClientDemo”，如下图有绿色箭头表示创建成功。



API 调用

示例代码修改**调用环境信息**后可直接调用。以下以新建工程为例，介绍如何在您的应用中调用SDK进行请求签名。

步骤1 把API信息替换到HttpClientDemo.java中对应位置。

例如，以AK/SK认证的API，API信息如下：

Ak: 3afe****ba29

Sk: ade8****37c2

Host: 3560ad08298c40f3a3be3ef8c224f60f.apig.cn-north-4.huaweicloudapis.com

URL: https://3560ad08298c40f3a3be3ef8c224f60f.apig.cn-north-4.huaweicloudapis.com/apig/sdk

说明

HttpClientDemo中引用以下类，可在“获取SDK”包中的“src”文件下查看：

- Constant：demo中用到的常量。
- SSLCipherSuiteUtil：tls认证配置参数的工具类，比如配置客户端不校验证书。
- UnsupportProtocolException：异常处理类。

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

    public static void main(String[]
args) throws Exception {
// Create a new request.
        Request request = new Request();
        try {
// Set the request parameters.
            // AppKey, AppSecrect, Method and Url are required parameters.
            request.setKey("3afe****ba29");
            request.setSecret("ade8****37c2");
            request.setMethod("GET");


            request.setUrl("https://3560ad08298c40f3a3be3ef8c224f60f.apig.cn-north-4.huaweicloudapis.com/
apig/sdk");
            request.addHeader("Host", "3560ad08298c40f3a3be3ef8c224f60f.apig.cn-
north-4.huaweicloudapis.com");

        } catch (Exception e) {
           LOGGER.error(e.getMessage());
            return;
        }
CloseableHttpClient client = null;
        try {
// Sign the request.
            HttpRequestBase signedRequest = Client.sign(request,
Constant.SIGNATURE_ALGORITHM_SDK_HMAC_SHA256);
            client = (CloseableHttpClient)
SSLCipherSuiteUtil.createHttpClient(Constant.INTERNATIONAL_PROTOCOL);
            HttpResponse response = client.execute(signedRequest);
            // Print the status line of the response.
            LOGGER.info("Print the status line of the response: {}", response.getStatusLine().toString());
            // Print the header fields of the response.
            Header[] resHeaders = response.getAllHeaders();
// Print the body of the response.
            HttpEntity resEntity = response.getEntity();
            if (resEntity != null)
{
               LOGGER.info("Processing Body with name: {} and value: {}", System.getProperty("line.separator"),
                     EntityUtils.toString(resEntity, "UTF-8"));
            }
       } catch (Exception e)
{
           LOGGER.error(e.getMessage());
        } finally {
if (client != null)
{
client.close();
```

```
        }
    }
  }
}
```

步骤2 运行HttpClientDemo.java，对请求进行签名、访问API并打印结果。

示例结果如下：

```
[main] INFO com.huawei.apig.sdk.demo.HttpClientDemo - Print the authorization: [Authorization: SDK-HMAC-SHA256 Access=3afe0280a6e1466e9cb6f23bcccdba29, SignedHeaders=host;x-sdk-date, Signature=26b2abfa40a4acf3c38b286cb6cbd9f07c2c22d1285bf0d4f6cf1f02d3bfdbf6]
[main] INFO com.huawei.apig.sdk.demo.HttpClientDemo - Print the status line of the response: HTTP/1.1 200 OK
[main] INFO com.huawei.apig.sdk.demo.HttpClientDemo - Processing Header with name: Date and value: Fri, 26 Aug 2022 08:58:51 GMT
[main] INFO com.huawei.apig.sdk.demo.HttpClientDemo - Processing Header with name: Content-Type and value: application/json
[main] INFO com.huawei.apig.sdk.demo.HttpClientDemo - Processing Header with name: Transfer-Encoding and value: chunked
[main] INFO com.huawei.apig.sdk.demo.HttpClientDemo - Processing Header with name: Connection and value: keep-alive
[main] INFO com.huawei.apig.sdk.demo.HttpClientDemo - Processing Header with name: Server and value: api-gateway
[main] INFO com.huawei.apig.sdk.demo.HttpClientDemo - Processing Header with name: X-Request-Id and value: 10955c5346b9512d23f3fd4c1bf2d181
[main] INFO com.huawei.apig.sdk.demo.HttpClientDemo - Processing Body with name:
 and value: {"200": "sdk success"}
```

显示**{"200": "sdk success"}**，表示签名成功，API成功请求到后端。

如果改变AK或SK的值，API网关将返回的错误信息error_msg。

----结束

3.6.2 Go

本节以IntelliJ IDEA工具为例，介绍如何在Go环境中集成API请求签名的SDK。您可以直接导入示例工程体验，然后参考调用说明部分将签名SDK集成到您的应用中。

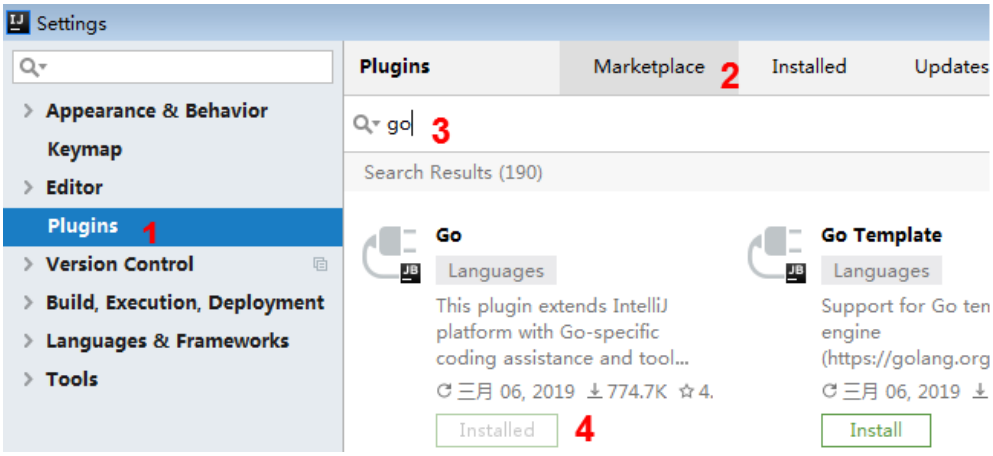
说明

签名SDK只包含签名功能，不包含云服务的SDK功能，云服务SDK请参见**SDK**。

准备 IDEA 开发环境

- 获取并安装IntelliJ IDEA，可至**IntelliJ IDEA官方网站**下载。
- 获取并安装Go安装包，可至**Go官方下载页面**下载。
- 在IDEA中安装Go插件，请打开“File > Settings”，如下图所示。

图 3-5 IDEA 安装 Go 插件



获取 SDK

点此下载SDK与Demo

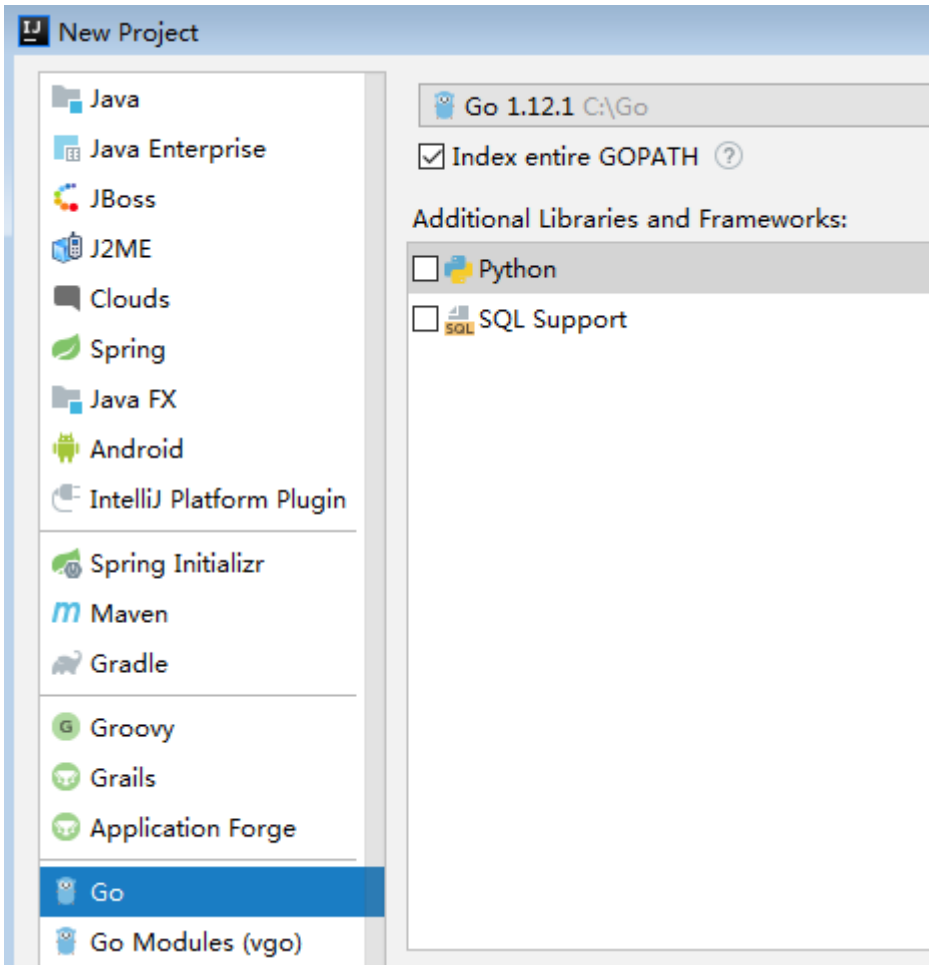
根据解压后获取的SDK代码及示例代码进行开发。

名称	说明
core\escape.go	特殊字符转义
core\signer.go	签名SDK
demo.go	示例代码

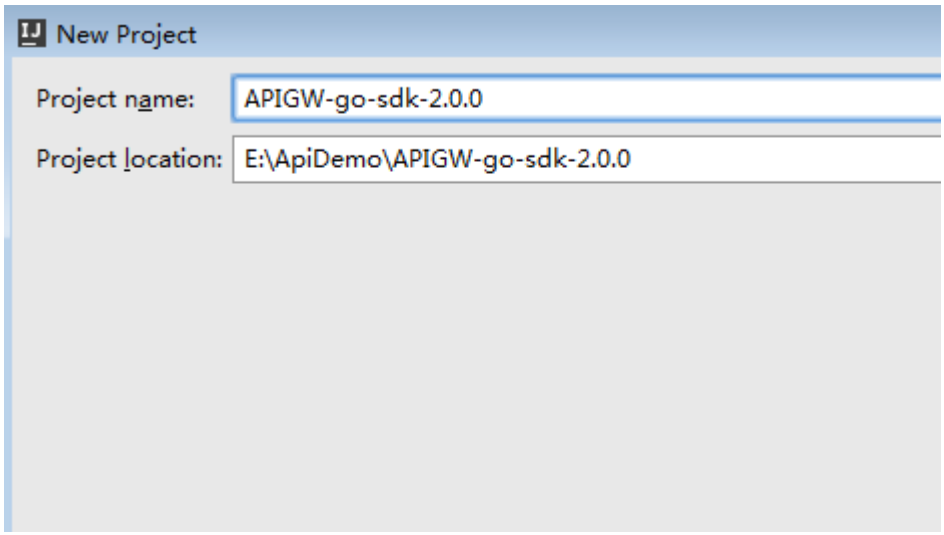
IDEA 工程中导入示例

步骤1 打开IDEA，选择菜单“File > New > Project”。

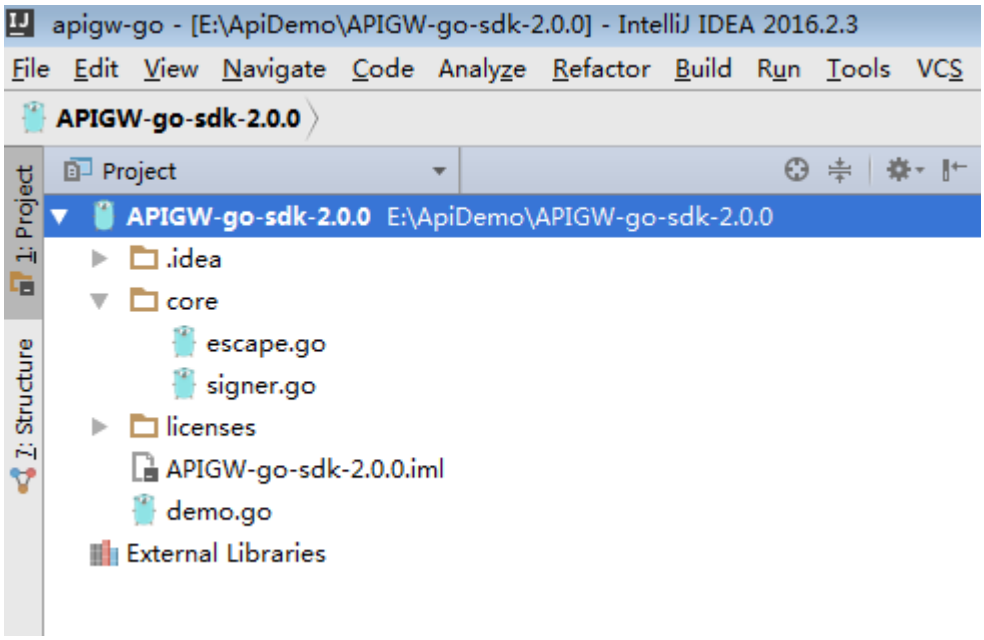
弹出“New Project”对话框，选择“Go”，单击“Next”。



步骤2 单击“...”，在弹出的对话框中选择解压后的SDK路径，单击“Finish”。



步骤3 完成工程创建后，目录结构如下。



----结束

请求签名与 API 调用

步骤1 在工程中引入sdk（signer.go）。

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

步骤2 生成一个新的Signer，分别输入AK和SK值。

```
s := core.Signer{
        Key: "QTWAOY********KYUC",
        Secret: "MFyfvK41ba2giqM7**********KGpownRZlmVmHc",
}
```

步骤3 生成一个新的Request，指定域名、方法名、请求url和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(""))))
```

步骤4 添加需要签名的其他头域，或者其他用途的头域，如**多项目**场景中添加X-Project-Id，或者**全局服务**场景中添加X-Domain-Id。

```
/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")
```

步骤5 进行签名，执行此函数会在请求中添加用于签名的X-Sdk-Date头和Authorization头。

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

步骤6 访问API，查看访问结果。

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

----结束

3.6.3 Python

本节以IntelliJ IDEA工具为例，介绍如何在Python环境中集成API请求签名的SDK。您可以直接导入示例工程体验，然后参考调用说明部分将签名SDK集成到您的应用中。

说明

签名SDK只包含签名功能，不包含云服务的SDK功能，云服务SDK请参见**SDK**。

准备环境

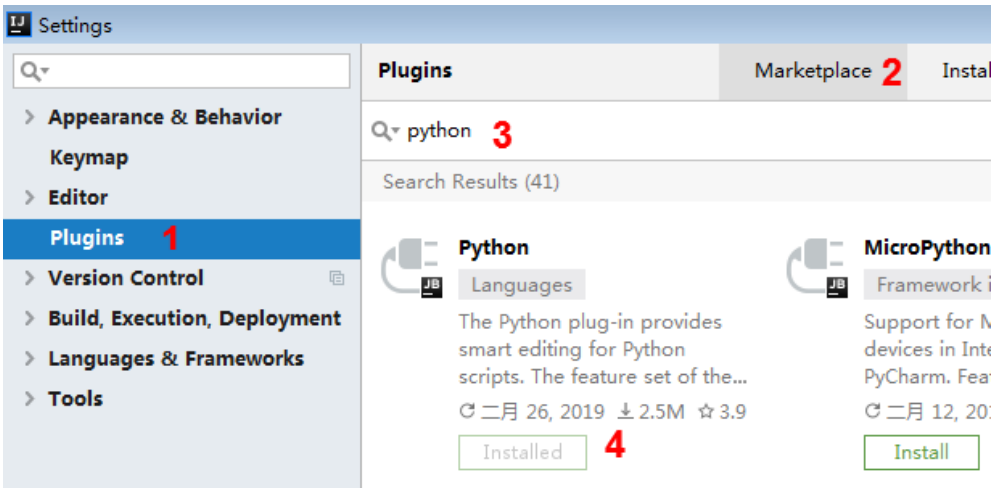
- 获取并安装IntelliJ IDEA，可至**IntelliJ IDEA官方网站**下载。
- 获取并安装Python安装包（可使用2.7.9+或3.X，包含2.7.9），可至**Python官方下载页面**下载。

  Python安装完成后，在命令行中使用pip安装“requests”库。

  ```
  pip install requests
  ```

  说明

  如果pip安装requests遇到证书错误，请下载并使用Python执行**此文件**，升级pip，然后再执行以上命令安装。
- 在IDEA中安装Python插件，如下图所示。



获取 SDK

点此下载SDK与Demo。

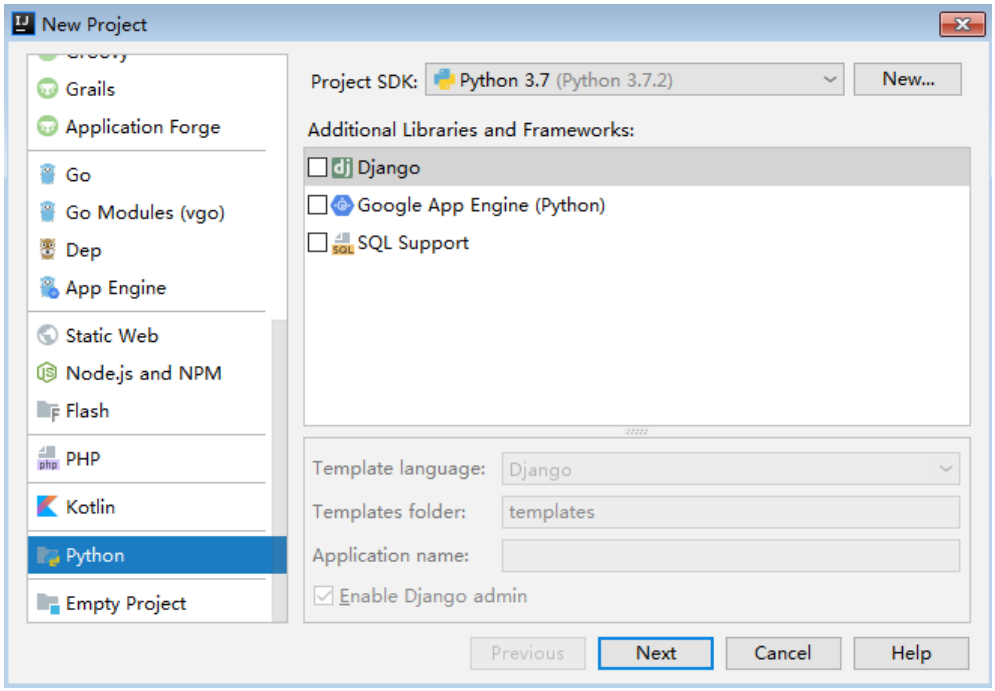
解压后目录结构如下：

名称	说明
apig_sdk__init__.py	SDK代码
apig_sdk\signer.py	
main.py	示例代码
licenses\license-requests	第三方库license文件

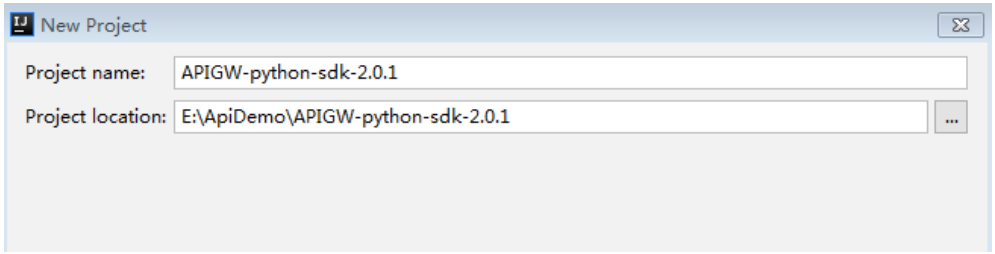
示例工程导入

步骤1 打开IDEA，选择菜单“File > New > Project”。

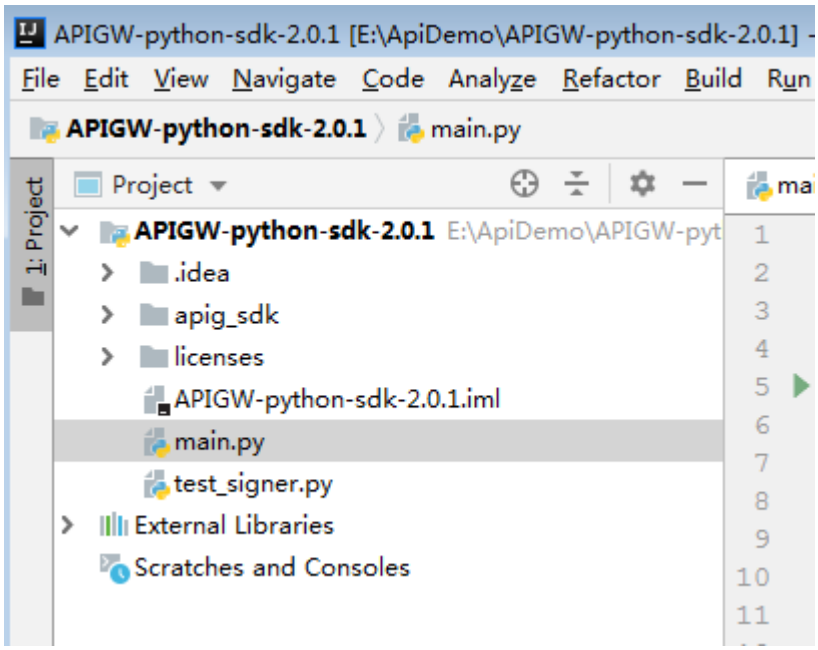
弹出“New Project”对话框，选择“Python”，单击“Next”。



步骤2 再次单击“Next”，弹出以下对话框。单击“...”，在弹出的对话框中选择解压后的SDK路径，单击“Finish”。



步骤3 完成工程创建后，目录结构如下。



----结束

请求签名与 API 调用

步骤1 在命令行中，使用pip安装“requests”库。

```
pip install requests
```

步骤2 在工程中引入apig_sdk。

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

步骤3 生成一个新的Signer，填入AK和SK。

```
sig = signer.Signer()
# Set the AK/SK to sign and authenticate the request.
sig.Key = "QTWAOY********VKYUC"
sig.Secret = "MFyfvK41ba2giqM7**********KGpownRZlmVmHc"
```

步骤4 生成一个新的Request，指定域名、方法名、请求uri和body。

以虚拟私有云服务的**查询VPC列表**接口为例，HTTP方法为**GET**，域名（Endpoint）为**service.region.example.com**，请求URI：**/v1/77b6a44cba5143ab91d13ab9a8ff44fd/vpcs?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}"
```

步骤5 添加需要签名的请求消息头，或者其他用途的头域，如**多项目**场景中添加X-Project-Id，或者**全局服务**场景中添加X-Domain-Id。如果添加多个请求消息头，使用英文逗号分隔。

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

步骤6 进行签名，执行此函数会在请求参数中添加用于签名的X-Sdk-Date头和Authorization头。

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


说明

- X-Sdk-Date是一个必须参与签名的请求消息头参数。
- 您无需关注哪些消息头参数参与了签名，由SDK自行完成。

步骤7 访问API，查看访问结果。

```
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)
```

----结束

3.6.4 C#

本节以Visual Studio工具为例，介绍如何在C#环境中集成API请求签名的SDK。您可以直接导入示例工程体验，然后参考调用说明部分将签名SDK集成到您的应用中。

说明

签名SDK只包含签名功能，不包含云服务的SDK功能，云服务SDK请参见**SDK**。

准备环境

获取并安装Visual Studio，可至**Visual Studio官方网站**下载。

获取 SDK

点此下载SDK与Demo。

解压后目录结构如下：

名称	说明
apigateway-signature\Signer.cs	SDK代码
apigateway-signature\HttpEncoder.cs	
sdk-request\Program.cs	签名请求示例代码
csharp.sln	工程文件
licenses\license-referencesource	第三方库license文件

打开工程

双击SDK包中的“csharp.sln”文件，打开工程。其中，apigateway-signature项目为实现签名算法的共享库，可用于.Net Framework与.Net Core项目。“sdk-request”项目为调用示例。

请求签名与 API 调用

步骤1 在工程中引入sdk。

```
using System;
using System.Net;
using System.IO;
```

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

步骤2 生成一个新的Signer， 填入AK和SK。

```
Signer signer = new Signer();
signer.Key = "QTWAOYT**********VKYUC";
signer.Secret = "MFyfvK41ba2giqM7**********KGpownRZlmVmHc";
```

步骤3 生成一个新的Request，指定域名、方法名、请求uri和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 = "";
```

步骤4 添加需要签名的其他头域，或者其他用途的头域，如**多项目**场景中添加X-Project-Id，或者**全局服务**场景中添加X-Domain-Id。

```
//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.Add("X-Project-Id", "xxx");
```

步骤5 进行签名，执行此函数会生成一个新的HttpWebRequest，并在请求参数中添加用于签名的X-Sdk-Date头和Authorization头。

如果您使用HTTP Client，可以从请求中获取头部信息使用。关于头部信息，请参考**AK/SK签名认证算法详解**。

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

步骤6 访问API，查看访问结果。

```
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("----------------");
```

----结束

3.6.5 JavaScript

本节以IntelliJ IDEA工具为例，介绍如何在Js环境中集成API请求签名的SDK。您可以直接导入示例工程体验，然后参考调用说明部分将签名SDK集成到您的应用中。

以搭建Node.js开发环境为例介绍JavaScript的SDK集成。

说明

签名SDK只包含签名功能，不包含云服务的SDK功能，云服务SDK请参见**SDK**。

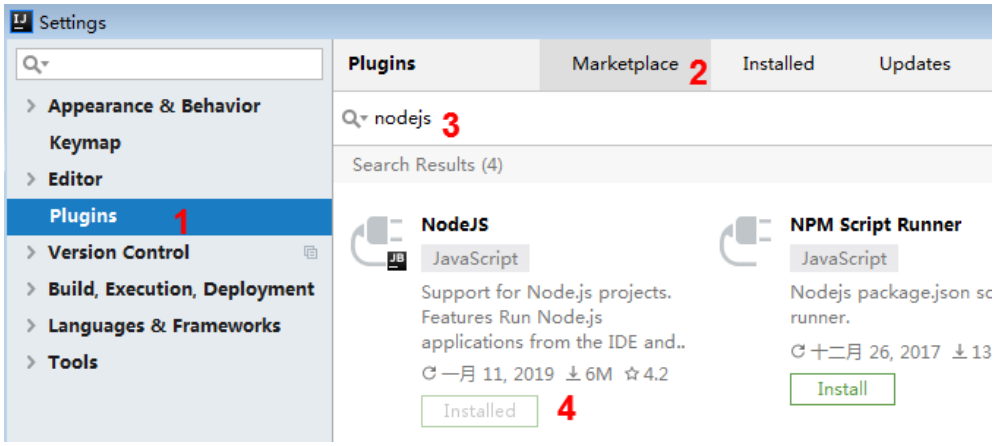
准备环境

- 获取并安装IntelliJ IDEA，可至**IntelliJ IDEA官方网站**下载。
- 获取并安装Nodejs安装包，可至**Nodejs官方下载页面**下载。

  NodeJs安装后，在命令行中，用npm安装“moment”和“moment-timezone”模块。

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

- 在IDEA中安装Nodejs插件，如下图所示。



获取 SDK

点此下载SDK与Demo。

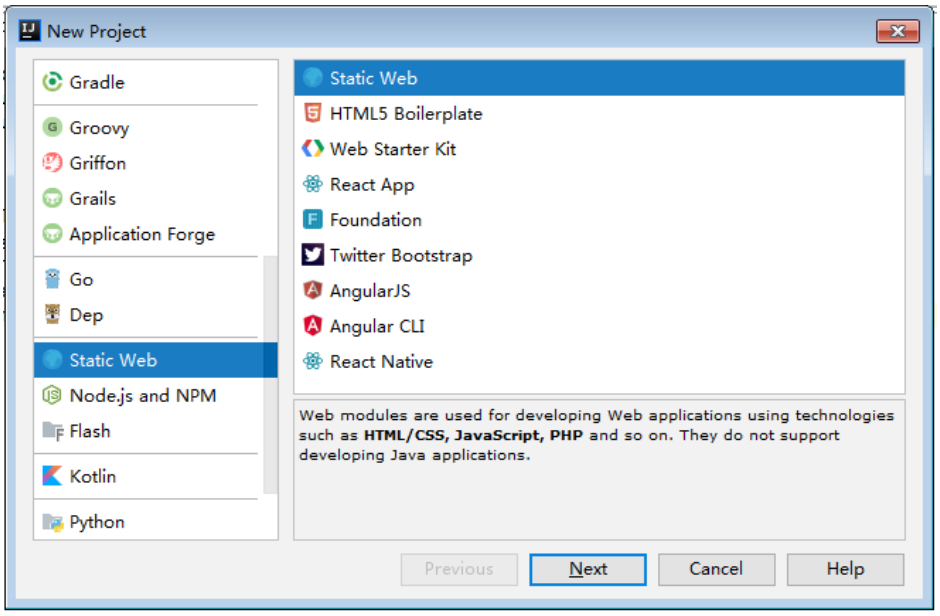
解压时选择解压到当前文件夹，解压后目录结构如下：

名称	说明
signer.js	SDK代码
node_demo.js	Nodejs示例代码
test.js	测试用例
licenses\license-crypto-js	第三方库license文件
licenses\license-node	

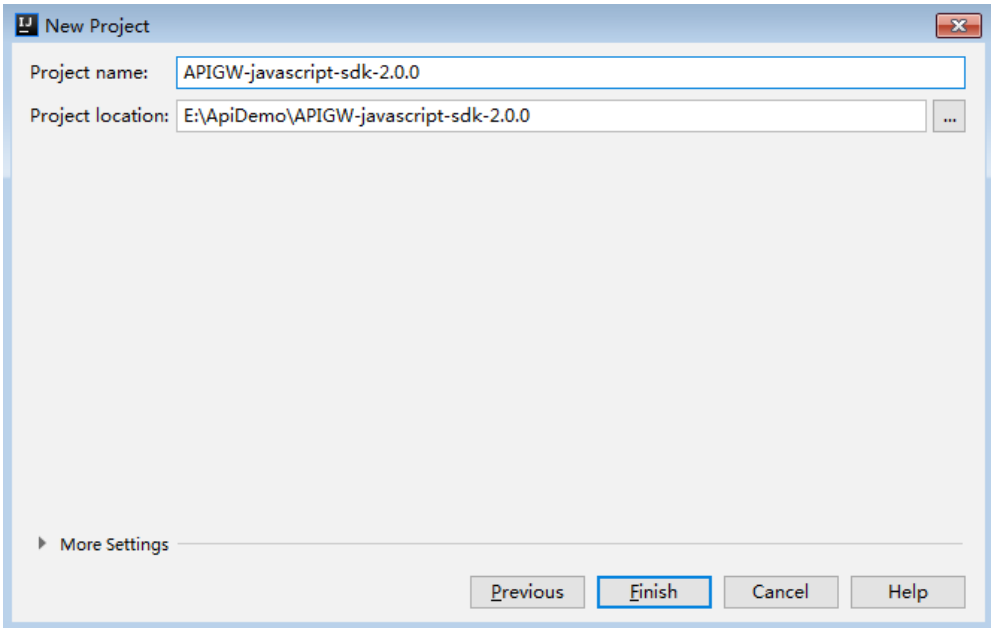
创建工程

步骤1 打开IDEA，选择菜单“File > New > Project”。

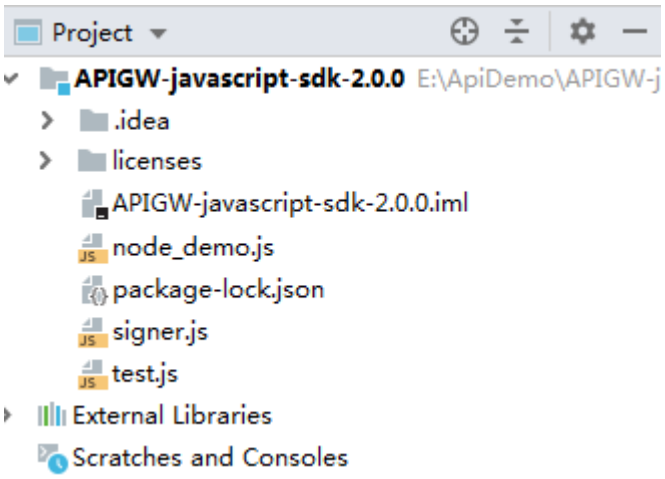
弹出“New Project”对话框。选择“Static Web”，单击“Next”。



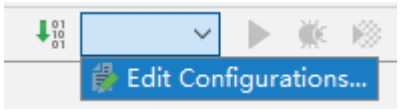
步骤2 单击“...”，在弹出的对话框中选择解压后的SDK路径，单击“Finish”。



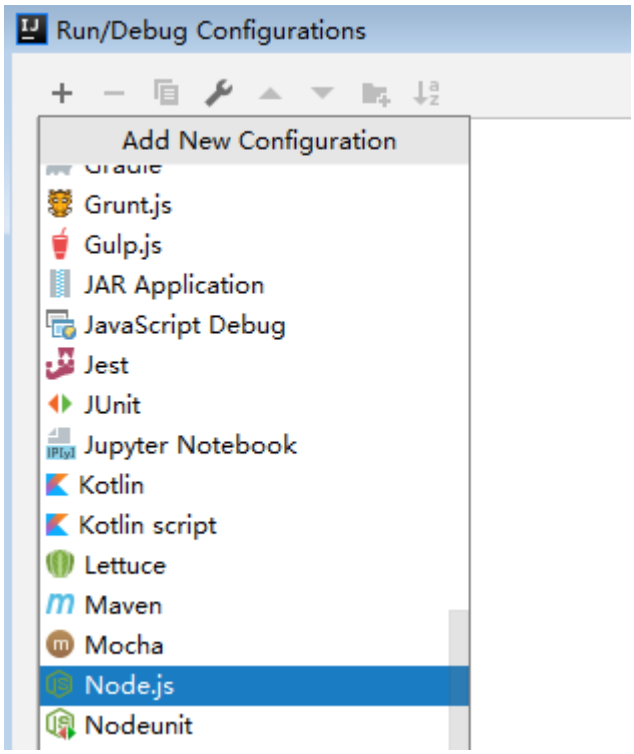
步骤3 完成工程创建后，目录结构如下。



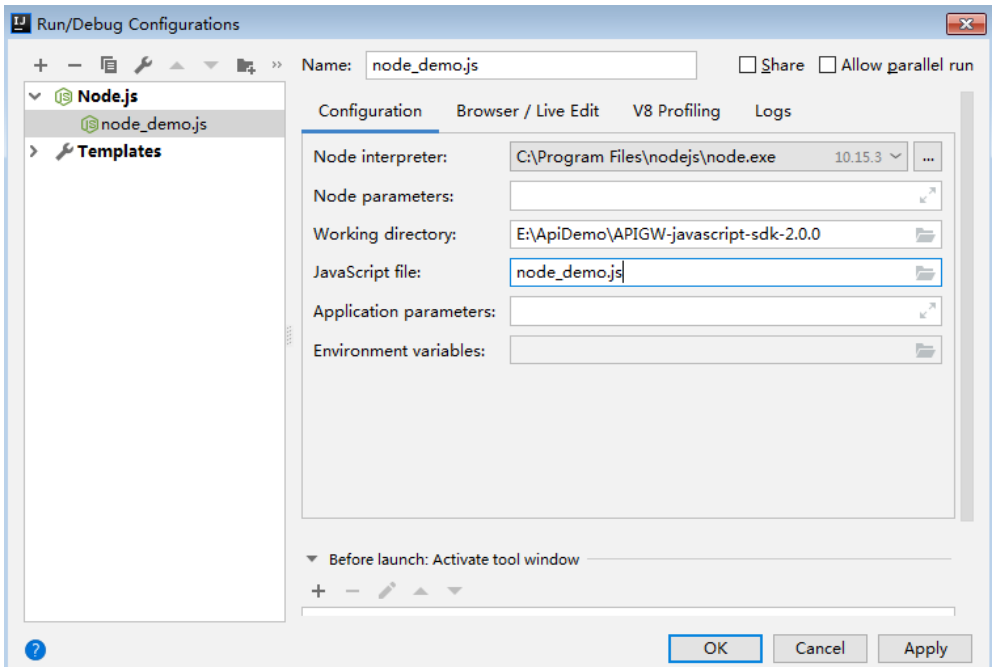
步骤4 在IDEA窗口右上方，单击“Edit Configurations”或“Add Configurations”。



步骤5 单击“+”，选择“Node.js”。



步骤6 “JavaScript file”选择“node_demo.js”，单击“OK”，完成配置。



----结束

API 调用（Node.js）

步骤1 在命令行中，用npm安装“moment”和“moment-timezone”模块。

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

步骤2 在工程中引入signer.js。

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

步骤3 生成一个新的Signer，填入“Key”和“Secret”。

```
var sig = new signer.Signer()
sig.Key = "QTWAOYT******QVKYUC"
sig.Secret = "MFyfvK41ba2giqM7**********KGpownRZlmVmHc"
```

步骤4 生成一个新的Request，指定域名、方法名、请求uri和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 = '';
```

步骤5 添加需要签名的其他头域，或者其他用途的头域，如**多项目**场景中添加X-Project-Id，或者**全局服务**场景中添加X-Domain-Id。

```
//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"};
```

步骤6 进行签名，执行此函数会生成请求参数，用于创建https请求，请求参数中添加了用于签名的X-Sdk-Date头和Authorization头。

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

步骤7 访问API，查看访问结果。


```
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()
```

----结束

3.6.6 PHP

本节以IntelliJ IDEA工具为例，介绍如何在PHP环境中集成API请求签名的SDK。您可以直接导入示例工程体验，然后参考调用说明部分将签名SDK集成到您的应用中。

说明

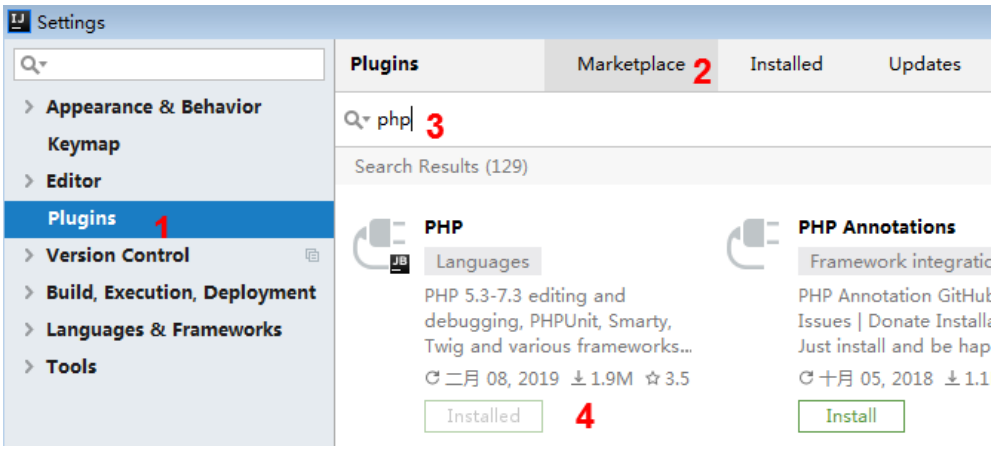
签名SDK只包含签名功能，不包含云服务的SDK功能，云服务SDK请参见**SDK**。

准备环境

- 获取并安装IntelliJ IDEA，可至**IntelliJ IDEA官方网站**下载。
- 获取并安装PHP安装包，可至**PHP官方下载页面**下载。
- 将PHP安装目录中的“php.ini-production”文件复制到“C:\windows”，改名为“php.ini”，并在文件中增加如下内容。

```
extension_dir = "{php安装目录}/ext"
extension=openssl
extension=curl
```

- 在IDEA中安装PHP插件，如下图所示。



获取 SDK

点此下载SDK与Demo。

解压时选择解压到当前文件夹，解压后目录结构如下：

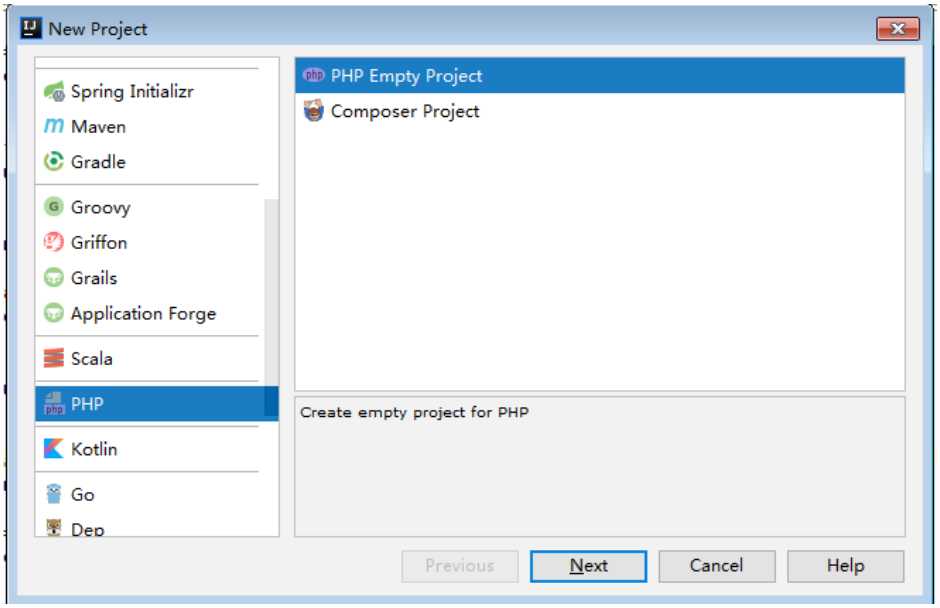
名称	说明
signer.php	SDK代码

名称	说明
index.php	示例代码

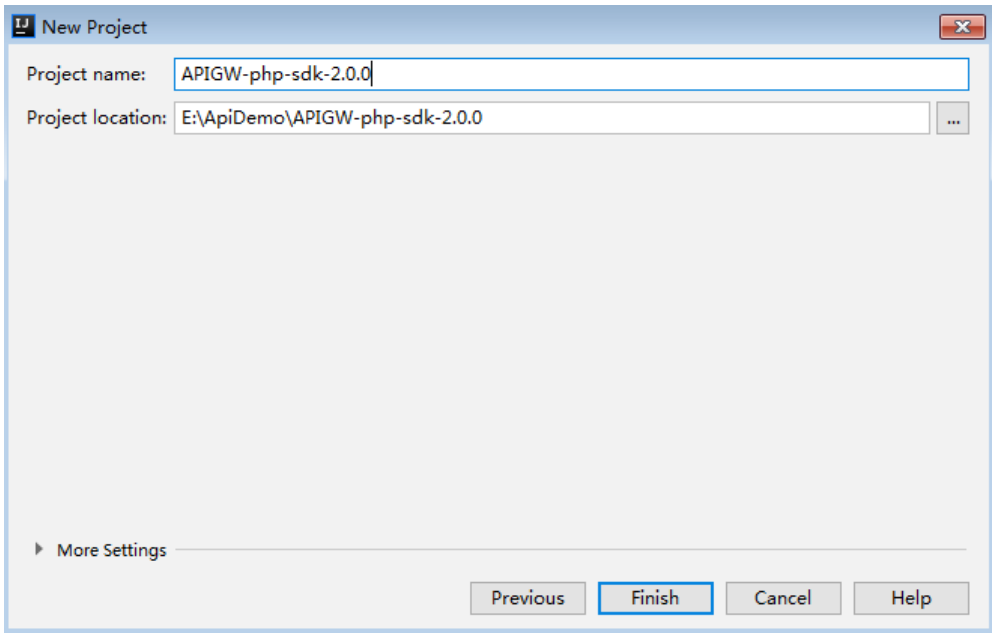
创建工程

步骤1 打开IDEA，选择菜单“File > New > Project”。

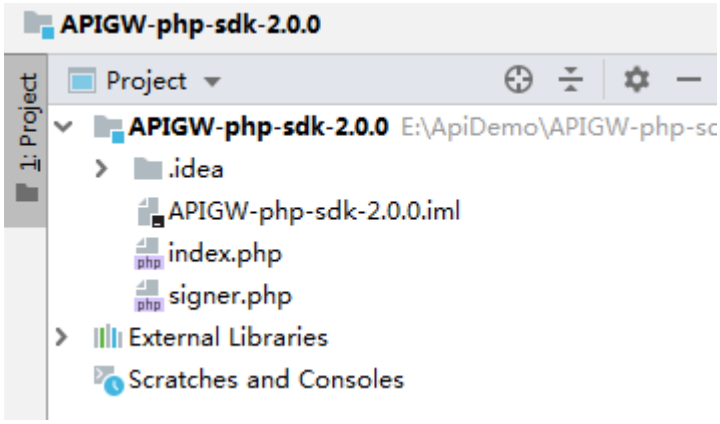
弹出“New Project”对话框，选择“php”，单击“Next”。



步骤2 单击“...”，在弹出的对话框中选择解压后的SDK路径，单击“Finish”。



步骤3 完成工程创建后，目录结构如下。



----结束

请求签名与 API 调用

步骤1 在代码中引入sdk。

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

步骤2 生成一个新的Signer， 填入AK和SK。

```
$signer = new Signer();
$signer->Key = 'QTWAOY*********QVKYUC';
$signer->Secret = 'MFyfvK41ba2giqM7**********KGpownRZlmVmHc';
```

步骤3 生成一个新的Request，指定域名、方法名、请求uri和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 = '';
```

步骤4 添加需要签名的其他头域，或者其他用途的头域，如**多项目**场景中添加X-Project-Id，或者**全局服务**场景中添加X-Domain-Id。

```
//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',
);
```

步骤5 进行签名，执行此函数会生成一个$curl上下文变量。

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

步骤6 访问API，查看访问结果。

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

----结束

3.6.7 C++

准备环境

以Linux Ubuntu系统为例，首先需要安装SSL相关工具。

1. 安装openssl库。
   ```
   apt-get install libssl-dev
   ```
2. 安装curl库。
   ```
   apt-get install libcurl4-openssl-dev
   ```

获取 SDK

说明

签名SDK只包含签名功能，不包含云服务的SDK功能，云服务SDK请参见**SDK**。

点此下载SDK与Demo。

解压时选择解压到当前文件夹，解压后目录结构如下：

名称	说明
hasher.cpp	SDK代码
hasher.h	
header.h	
RequestParams.cpp	
RequestParams.h	
signer.cpp	
signer.h	
constants.h	
Makefile	Makefile文件
main.cpp	示例代码

请求签名与 API 调用

步骤1 在main.cpp中加入以下引用。

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

步骤2 生成一个新的Signer，将AK/SK分别填入。

```
//Set the AK/SK to sign and authenticate the request.
Signer signer("QTWAOYT********VKYUC", "MFyfvK41ba2giqM7**********KGpownRZlmVmHc");
```

步骤3 生成一个新的RequestParams，指定方法名、域名、请求uri、查询字符串和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", "");
```


步骤4 添加需要签名的头域，或者其他用途的头域，如**多项目**场景中添加X-Project-Id，或者**全局服务**场景中添加X-Domain-Id。

```
//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");
```

步骤5 进行签名，执行此函数会将生成的签名头加入request变量中。

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

步骤6 使用curl库访问API，查看访问结果。

```
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;
}
```

步骤7 运行make命令编译，得到可执行文件main，执行main文件，查看结果。

----结束

3.6.8 C

准备环境

以Linux Ubuntu系统为例，首先需要安装SSL相关工具。

1. 安装openssl库。
   ```
   apt-get install libssl-dev
   ```
2. 安装curl库。
   ```
   apt-get install libcurl4-openssl-dev
   ```

获取 SDK

说明

签名SDK只包含签名功能，不包含云服务的SDK功能，云服务SDK请参见**SDK**。

点此下载SDK与Demo。

解压时选择解压到当前文件夹，解压后目录结构如下：

名称	说明
signer_common.c	签名SDK代码
signer_common.h	
signer.c	
signer.h	
Makefile	Makefile文件
main.c	示例代码

请求签名与 API 调用

步骤1 在main.c中加入以下引用。

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

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

步骤2 生成一个sig_params_t类型的变量， 填入AK和SK。

```
sig_params_t params;
sig_params_init(&params);
sig_str_t ak = sig_str("QTWAOY******QVKYUC");
sig_str_t sk = sig_str("MFyfvK41ba2giqM7**********KGpownRZlmVmHc");
params.key = ak;
params.secret = sk;
```

步骤3 指定方法名、域名、请求uri、查询字符串和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;
```

步骤4 增加头部参数，或者其他用途的头域，如**多项目**场景中添加X-Project-Id，或者**全局服务**场景中添加X-Domain-Id。

```
//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");
```

步骤5 进行签名，执行此函数会将生成的签名头加入request变量中。

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

步骤6 使用curl库访问API，查看访问结果。

```
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;
}
```

步骤7 运行make命令编译，得到可执行文件main，执行main文件，查看结果。

----结束

3.6.9 Android

本节以Android Studio工具为例，介绍如何在Android环境中集成API请求签名的SDK。您可以直接导入示例工程体验，然后参考调用说明部分将签名SDK集成到您的应用中。

准备环境

获取并安装Android Studio，可至**Android Studio官方网站**下载。

获取 SDK

说明

签名SDK只包含签名功能，不包含云服务的SDK功能，云服务SDK请参见**SDK**。

点此下载SDK与Demo。

解压后目录结构如下：

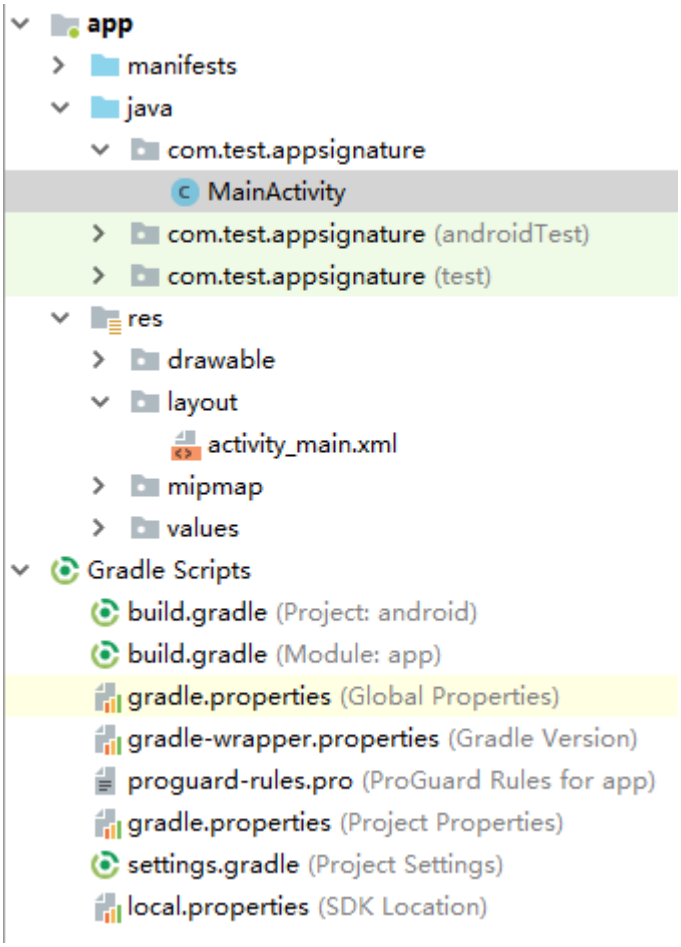
名称	说明
app\	安卓工程代码
build.gradle	gradle配置文件
gradle.properties	
settings.gradle	

打开工程

步骤1 打开Android Studio，选择“File > Open”。

在弹出的对话框中选择解压后的SDK路径。

步骤2 打开工程后，目录结构如下。



----结束

请求签名与 API 调用

步骤1 在Android工程中的“app/libs”目录下，加入SDK所需jar包。其中jar包必须包括：

- java-sdk-core-x.x.x.jar
- commons-logging-1.2.jar
- joda-time-2.9.9.jar

步骤2 在“build.gradle”文件中加入okhttp库的依赖。

在“build.gradle”文件中的“dependencies”下加入“implementation 'com.squareup.okhttp3:okhttp:3.11.0'”。

```
dependencies {
    ...
    ...
    implementation 'com.squareup.okhttp3:okhttp:3.11.0'
}
```

步骤3 创建request，输入AK和SK，并指定域名、方法名、请求uri和body。

```
Request request = new Request();
try {
        request.setKey("QTWAOYTT*******KYUC");
        request.setSecrect("MFyfvK41ba2giqM7**********KGpownRZlmVmHc");
        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;
}
```

步骤4 对请求进行签名，生成okhttp3.Request对象来访问API。

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

----结束

4 错误码说明

当您使用各云服务的API时，如果遇到“APIGW”开头的错误码，请参考如下语义与解决方案进行处理。

表 4-1 错误码

错误码	错误信息	HTTP状态码	语义	解决方案
APIGW.0101	The API does not exist or has not been published in the environment.	404	API不存在或未发布到环境	请参考**"The API does not exist or has not been published in the environment."如何解决?** 处理
APIGW.0101	The API does not exist.	404	API请求方法不存在	检查API请求方法是否与API定义的方法相同
APIGW.0103	The backend does not exist.	404	无法找到后端	联系技术支持
APIGW.0104	The plug-ins do not exist.	400	无法找到插件配置	联系技术支持
APIGW.0105	The backend configurations do not exist.	400	无法找到后端配置	联系技术支持
APIGW.0106	Orchestration error.	400	编排错误	检查API配置的前后端参数是否合理
APIGW.0201	API request error.	400	请求格式不合法	使用合法的请求
APIGW.0201	Request entity too large.	413	请求body过大（大于12M）	减小请求body大小

错误码	错误信息	HTTP状态码	语义	解决方案
APIGW.0201	Request URI too large.	414	请求URI过大	减小请求URI大小
APIGW.0201	Request headers too large.	494	请求头过大	减小请求头大小
APIGW.0201	Backend unavailable.	502	后端不可用	检查API配置的后端地址是否可用
APIGW.0201	Backend timeout.	504	后端超时	增大超时时间或缩小后端的处理时间
APIGW.0301	Incorrect IAM authentication information.	401	IAM认证信息错误	请参考**IAM认证信息错误**处理
APIGW.0302	The IAM user is not authorized to access the API.	403	IAM用户不允许访问API	检查用户是否被黑/白名单限制
APIGW.0303	Incorrect app authentication information.	401	APP认证信息错误	APP签名认证时，做如下检查： • 检查请求的方法、路径、查询参数、请求体和签名使用的方法、路径、查询参数、请求体是否一致 • 检查客户端机器时间是否正确 请参考**使用APP认证调用API**检查签名代码的问题。 APPCODE简易认证时，做如下检查：检查请求是否携带了X-Apig-AppCode头域
APIGW.0304	The app is not authorized to access the API.	403	APP不允许访问API	检查APP是否授权访问API
APIGW.0305	Incorrect authentication information.	401	认证信息错误	检查认证信息是否正确
APIGW.0306	API access denied.	403	不允许访问API	检查是否授权访问API

错误码	错误信息	HTTP状态码	语义	解决方案
APIGW.0307	The token must be updated.	401	Token需要更新	• 重新从IAM服务获取Token • 也有可能是调用了不同region的接口，导致判断为Token失效，建议检查接口URL
APIGW.0308	The throttling threshold has been reached.	429	超出流控值限制	• 等待流控刷新后访问。默认每个API每秒最多访问200次 • 云服务开放的API，一般无法调整限额，请等待流控刷新后访问 • 您在API网关服务中自行创建API，如需调整限额，请提工单联系技术支持
APIGW.0310	The project is unavailable.	403	project不可使用	使用其他project访问
APIGW.0311	Incorrect debugging authentication information.	401	调试认证信息错误	联系技术支持
APIGW.0401	Unknown client IP address.	403	无法识别客户端IP地址	联系技术支持
APIGW.0402	The IP address is not authorized to access the API.	403	IP地址不允许访问	检查IP地址是否被黑/白名单限制
APIGW.0404	Access to the backend IP address has been denied.	403	后端IP不允许访问	后端IP地址或后端域名对应的IP地址不允许访问，请检查IP地址是否被黑/白名单限制，或检查对应后端IP是否存在。
APIGW.0501	The app quota has been used up.	405	APP已经超出配额	购买APP配额

错误码	错误信息	HTTP状态码	语义	解决方案
APIGW.0502	The app has been frozen.	405	APP被冻结	余额不足，请前往“资金管理”充值。
APIGW.0601	Internal server error.	500	内部错误	联系技术支持
APIGW.0602	Bad request.	400	非法请求	检查请求是否合法
APIGW.0605	Domain name resolution failed.	500	域名解析失败	检查域名拼写，以及域名是否绑定了正确的后端地址
APIGW.0606	Failed to load the API configurations.	500	未加载API配置	联系技术支持
APIGW.0607	The following protocol is supported: {xxx}	400	协议不被允许，允许的协议是*xxx* 注意：xxx以实际响应中的内容为准	改用支持的协议（HTTP/HTTPS）访问
APIGW.0608	Failed to obtain the admin token.	500	无法获取管理租户	联系技术支持
APIGW.0609	The VPC backend does not exist.	500	找不到vpc后端	联系技术支持
APIGW.0610	No backend available.	502	没有可连接的后端	检查所有后端是否可用
APIGW.0611	The backend port does not exist.	500	后端端口未找到	联系技术支持
APIGW.0612	An API cannot call itself.	500	API调用自身	修改API后端配置，递归调用层数不能超过10层
APIGW.0613	The IAM service is currently unavailable.	503	IAM服务暂时不可用	联系技术支持
APIGW.0705	Backend signature calculation failed.	500	计算后端签名失败	联系技术支持
APIGW.0801	The service is unavailable in the currently selected region.	403	服务在当前region不可访问	检查所访问的服务是否支持跨region访问

错误码	错误信息	HTTP状态码	语义	解决方案
APIGW.0802	The IAM user is forbidden in the currently selected region.	403	该IAM用户在当前region中被禁用	联系技术支持

5 常见问题

5.1 子项目场景下怎么调用 API

如果您的资源在子项目中，调用云服务API时需要增加一个请求消息头参数**X-Project-Id**，参数值填您的**获取项目ID**，增加参数**X-Project-Id**请参见**签名SDK与demo**。

5.2 API 调用是否支持长连接

API网关支持长连接，但注意适当使用，避免占用太多资源。

5.3 Body 体是否可以不参与签名

当您不想对Body进行签名时，请在消息头添加以下参数和参数值：

X-Sdk-Content-Sha256:UNSIGNED-PAYLOAD

添加之后，对body计算hash的位置的值为UNSIGNED-PAYLOAD。

5.4 请求消息头参数是否可以不参与签名

如果您按照**AK/SK签名认证算法详解**自行完成API请求签名，请注意X-Sdk-Date必须参与签名，其他请求消息头参数可选。

如果您使用华为云提供的SDK进行签名，则不需要考虑哪些请求消息头参数参与签名，SDK的签名方法自行决定哪些参数参与签名，并生成签名信息。如果某个请求消息头参数的值在签名后会发生变化，请完成签名后再对其赋值。

5.5 使用临时 AK/SK 做签名

如果使用临时AK/SK对请求签名，您将签名SDK集成到应用时，请在消息头添加以下参数和参数值：

X-Security-Token:*{securityToken}*

然后使用临时AK/SK对请求进行签名，签名SDK与AK/SK的签名SDK一致。

步骤1 创建一个API，安全认证选择“华为IAM认证”，并发布。

步骤2 获取当前帐号的临时AK/SK与*{securityToken}*，请参考**IAM接口文档**。

例如获得响应参数为：

```
{
   "credential": {
      "access": "P0HEQUQ4XBWXY5WD69X0",
      "expires_at": "2022-10-17T18:51:25.231000Z",
      "secret": "3WJu****hDVs",
      "securitytoken": "XXXXXX....."
   }
}
```

步骤3 构造请求，填写签名参数。

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

----结束

5.6 IAM 认证信息错误

IAM认证信息错误有：

- **Incorrect IAM authentication information: verify aksk signature fail**
- **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: verify aksk signature fail

```
{
  "error_msg": "Incorrect IAM authentication information: verify aksk signature fail, .....
  "error_code": "APIGW.0301",
  "request_id": "******"
}
```

可能原因

签名认证算法使用有问题，客户端计算的签名结果与API网关计算的签名结果不同。

解决方法

方法一：查看日志。

步骤1 获取API网关计算的canonicalRequest。

从报错信息的body中获取“request_id”，通过“request_id”查找shubao节点的error.log（error.log在CLS上查看），在error.log中获取canonicalRequest。

```
2019/01/26 11:34:27 [error] 1211#0: *76 [lua] responses.lua:170: rewrite():
473a4370fbaf69e42f9da243eb8f8c52;app-1;Incorrect IAM authentication information: verify signature
fail;SDK-HMAC-SHA256 Access=071fe245-9cf6-4d75-822d-c29945a1e06a, SignedHeaders=host;x-sdk-date,
Signature=b2ef2cddcef89cbfe22974c988909c1a94b1ac54114c30b8fe083d34a259e0f5;canonicalRequest:GE
T
/app1/

host:test.com
x-sdk-date:20190126T033427Z

host;x-sdk-date
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855, client: 192.168.0.1, server:
shubao, request: "GET /app1 HTTP/1.1", host: "test.com"
```

步骤2 通过打印日志或调试中断的方式得到客户端计算的canonicalRequest，每种语言SDK中计算canonicalRequest的位置如下：

表 5-1 常见语言 SDK 中计算 canonicalRequest 的位置

语言	位置
java	libs/java-sdk-core-..*.jar中 com.cloud.sdk.auth.signer.DefaultSigner.class中的sign函数。
c	signer.c中的sig_sign函数。
c++	signer.cpp中的Signer::createSignature函数。
c#	signer.cs中的Sign函数。
go	signer.go中的Sign函数。
JavaScript	signer.js中的Signer.prototype.Sign函数。
python	signer.py中的Sign函数。
php	signer.php中的Sign函数。

例如，在调试中断位置获取的canonicalRequest。

```
POST
/app1/

host:test.com
x-sdk-date:20190126T033950Z

host;x-sdk-date
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

步骤3 比较**步骤1**和**步骤2**中的canonicalRequest是否一致。

- 是，请检查appsecret或sk是否正确。（常见问题：appsecret或sk中多填了空格）

- 否。
  - 第1行不同：请求方法要保持一致。
  - 第2行不同：请求路径要保持一致。
  - 第3行不同：请求参数要保持一致。
  - 第4-5行不同：请求头信息，每行都要保持一致。
  - 第7行不同：请求头参数名个数要和请求头信息行数保持一致。
  - 第8行不同：请求body要保持一致。

表 5-2 比较 API 网关和客户端计算的 canonicalRequest

行数	参数	API网关	客户端
1	请求方法	GET	POST
2	请求路径	/app1/	/app1/
3	请求参数	空	空
4	请求头信息	host:test.com	host:test.com
5	请求头信息	x-sdk-date:20190126T033427Z	x-sdk-date:20190126T033950Z
6	空行	-	-
7	请求头参数名列表	host;x-sdk-date	host;x-sdk-date
8	请求body的hash值	e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855	e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855

----结束

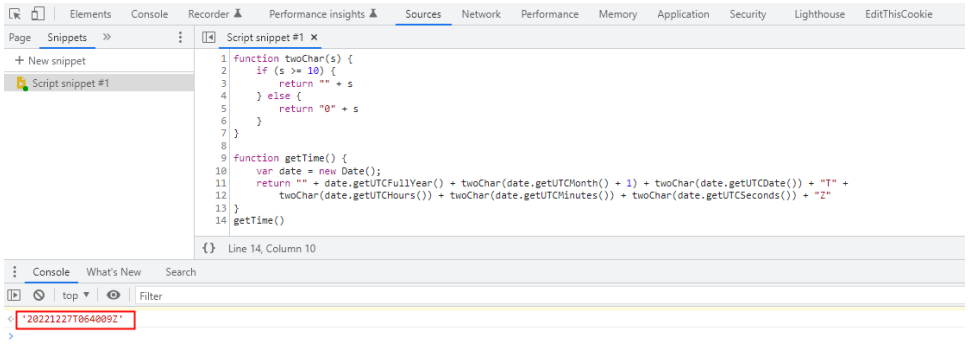
方法二：本地签名字符串比对。

步骤1 下载**js版本**，查看可视化签名SDK，获取签名字符串。

步骤2 解压压缩包，使用浏览器打开“demo.html”文件。

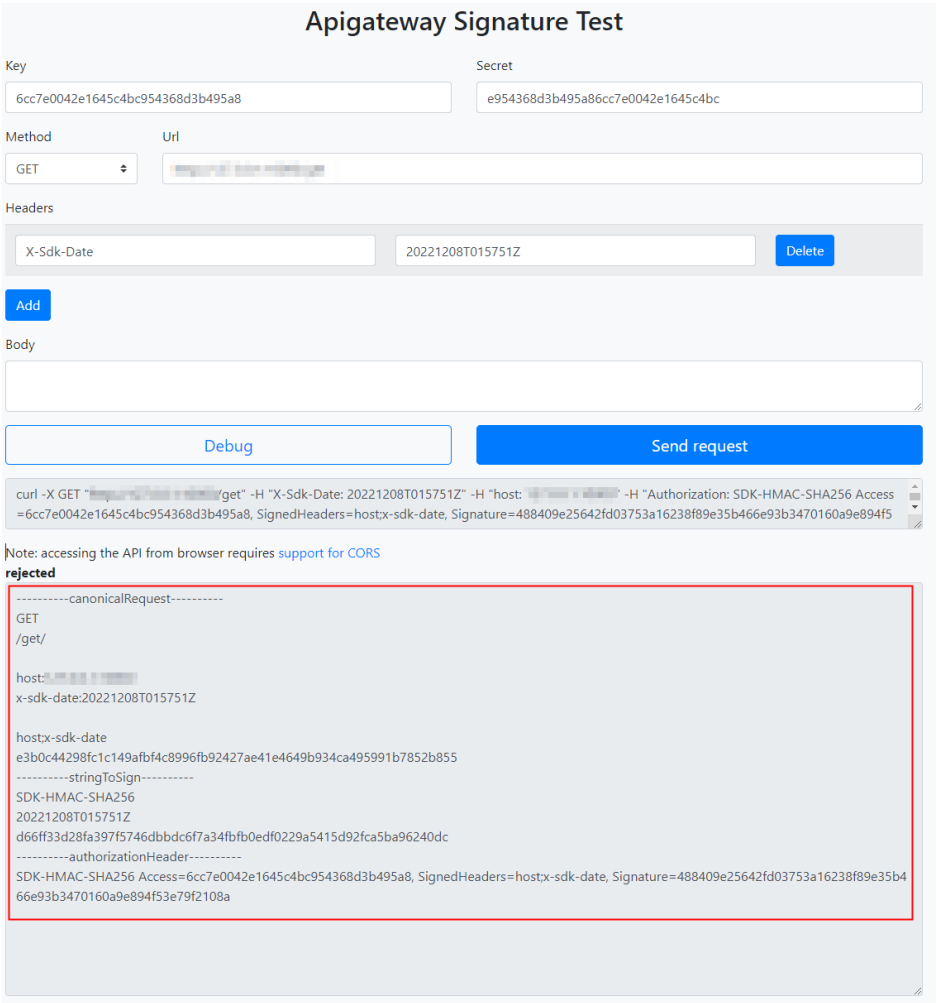
步骤3 获取x-sdk-date值，x-sdk-date值必须与当前时间相差在15min以内。

1. 在键盘中按下“F12”，并在页面中选择“Sources > Snippets > New snippet”。
2. 将以下代码复制到右侧的Script snippet中，然后在左侧右键Script snippet名称，选择“Run”后，“Console”中打印的值就是x-sdk-date值。




```
function twoChar(s) {
    if (s >= 10) {
        return "" + s
    } else {
        return "0" + s
    }
}

function getTime() {
    var date = new Date();
    return "" + date.getUTCFullYear() + twoChar(date.getUTCMonth() + 1) + twoChar(date.getUTCDate()) +
"T" +
        twoChar(date.getUTCHours()) + twoChar(date.getUTCMinutes()) + twoChar(date.getUTCSeconds()) +
"Z"
}
getTime()
```

步骤4 将x-sdk-date添加到Headers中，并填写其余参数，单击“debug”获取签名字符串。



非get\delete\head请求，需要携带body体，需要在上图Body框中添加body（格式与发送请求的body一致）。

步骤5 复制**步骤4**图中的curl命令，在cmd命令行中执行，curl命令执行后再进行下一步。

```
curl -X GET "http://192.168.0.1:10000/get" -H "X-Sdk-Date: 20221208T015751Z" -H "host: 192.168.0.1:10000" -H "Authorization: SDK-HMAC-SHA256 Access=6cc7e0042e1645c4bc954368d3b495a8, SignedHeaders=host;x-sdk-date, Signature=488409e25642fd03753a16238f89e35b466e93b3470160a9e894f53e79f2108a" -d $''
```

如果自定义authorization，则需要把curl命令里面的Authorization替换为自定义名称。

步骤6 比较本地代码中签名结果与js可视化签名结果。

例如排查java语言签名代码中的**canonicalRequest**、**stringToSign**、**authorizationHeader**值，与js可视化签名字符串是否一致。


```
public void sign(Request request) throws UnsupportedEncodingException {
    String singerDate = getHeader(request, X_SDK_DATE);
    SimpleDateFormat sdf = new SimpleDateFormat( pattern: "yyyyMMdd'T'HHmmss'Z'");
    sdf.setTimeZone(TimeZone.getTimeZone("UTC"));

    if (singerDate == null) {
        singerDate = sdf.format(new Date());
        request.addHeader(X_SDK_DATE, singerDate);
    }
    addHostHeader(request);

    String messageDigestContent = calculateContentHash(request);

    String[] signedHeaders = getSignedHeaders(request);

    final String canonicalRequest = createCanonicalRequest(request, signedHeaders, messageDigestContent);

    final byte[] signingKey = deriveSigningKey(request.getSecrect());

    String stringToSign = createStringToSign(canonicalRequest, singerDate);
    byte[] signature = computeSignature(stringToSign, signingKey);
    String signatureResult = buildAuthorizationHeader(signedHeaders, signature, request.getKey());

    request.addHeader(AUTHORIZATION, signatureResult);
}
```

----结束

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

可能原因

- aksk签名计算错误。请参考**Incorrect IAM authentication information: verify aksk signature fail**解决方法。
- ak对应的sk不匹配。
- aksk频繁出现鉴权出错，连续错误5次以上，被锁定5分钟（5分钟内鉴权失败，误以为是异常的鉴权请求）。
- token鉴权时，token过期。

Incorrect IAM authentication information: decrypt token fail

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

可能原因

用户的API所属IAM认证，TOKEN解析失败。

解决办法

- 检查获取token的方法，token是否正确。

- 检查获取token的环境与调用的环境是否一致。

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

可能原因

用户的API所属IAM认证，使用AK/SK签名方式访问，但是AK不存在。

解决方法

检查AK填写是否正确。

5.7 "The API does not exist or has not been published in the environment."如何解决?

调用API网关中开放的API报错，请按以下顺序排查可能原因：

1. 调用API所使用的域名、请求方法、路径不正确。
   - 比如创建的API为POST方法，您使用了GET方法调用。
   - 比如访问的URL比API详情中的URL少一个“/”也会导致无法匹配上此API，例如http://7383ea59c0cd49a2b61d0fd1d351a619.apigw.region.cloud.com/test/和http://7383ea59c0cd49a2b61d0fd1d351a619.apigw.region.cloud.com/test会匹配上不同的API。
2. API没有发布。API创建后，需要发布到具体的环境后才能使用。具体操作请参考**发布API**。如果发布到非生产环境，检查请求“X-Stage”头是否为发布的环境名。
3. 域名解析不正确。如果API的域名、请求方法、路径正确，且已发布到环境，有可能是没有准确解析到您的API所在分组。请检查API所在的分组域名，例如您有多个API分组，每个分组有自己的独立域名，API调用时，使用了其他分组的独立域名。
4. 检查API是否使用OPTIONS跨域请求，如果使用OPTIONS跨域请求，请在API中开启CORS，并创建OPTIONS方式的API。具体操作请参考**开启跨域共享**。

