ALIBABA CLOUD

Alibaba Cloud

API 网关

调用API

文档版本：20220314

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.概述

完成API的创建和发布之后，本章节重点会从客户端（如其他业务系统的代码、手机APP或HTML页面中等）角度来介绍如何调用您发布在API网关上的API，因此重点会涉及到如下部分的内容： - 了解API的认证方式、适用场景并如何实现客户端代码 - 了解客户端处在不同的网络环境下如何调用API - 了解做为API网关管理员，如何对API授权进行管理和分配

1. API认证方式

- **使用无认证方式调用API**

客户端可以直接调用API，安全性很低，适合临时测试的场景，不建议使用。

- **使用AppCode调用API（简单身份认证）**

AppCode认证方式，调用简单，但安全性较低，适合于内网系统之间的API调用。API网关通过检查客户端调用API时的Request Header或Query中，是否包含有满足要求的AppCode来进行认证。具体操作过程请参考使用简单认证（AppCode）方式调用API

- **使用摘要签名认证方式调用API**

相比简单身份认证，摘要签名方式安全性较高，适合于较多场景。客户端在调用API时，需要先根据事前分配好的APP Key和APP Secret对请求内容计算签名，在发起HTTP Request时，需将APP Key和签名信息传输给API网关进行验证。

- API网关提供的SDK内置了签名实现，您可以直接用API网关控制台为您提供的多语言SDK来调用API，具体操作请参考使用SDK调用API

- 如果您需要自己在客户端实现签名计算过程，可以参考客户端签名计算过程说明文档

- **使用JWT认证方式调用API**

还可通过配置JWT插件的方式，实现JWT认证方式，安全强度最高，适合在Javascript、Web前端等不安全的客户端场景。详情可参考JWT认证插件

2. 客户端在不同的网络环境下调用API

客户端可以通过互联网或VPC内网方式调用API。互联网可直接参考**章节1**的认证方式来调用API。

VPC内网目前支持三种情况：

- VPC内网访问API网关，配置请参考VPC内网访问API网关 。
- 函数计算内网访问API网关，配置请参考函数计算内网访问API网关 。
- 同Region的API自调用默认走内网，详情请参考API网关自调用说明。

3. 授权管理

若创建API认证方式为“阿里云APP”，这里需要将APP和API的授权关系建立好，才能够正常访问。 调用前您可以创建APP授权给API或是根据APPID授权他人创建的APP。作为您调用API的身份，每个APP有一对AppKey和AppSecret密钥对，AppKey需要在请求时作为参数在Header传入，AppSecret需要用于计算请求签名。

详情可参考API授权。

2.客户端调用API示例

您可以通过 API 网关，调用由其他阿里云用户或者第三方服务商开放的 API 服务。API 网关将为您提供一系列管理服务与支撑。

调用API

您可以直接用API网关控制台为您提供的多语言SDK来调用API（SDK和文档下载），也可以自行编写HTTP(s)请求调用API。

API调用说明及示例如下：

1.请求

请求地址组成：域名+path

域名您需要从分组上绑定的域名（如果还未绑定域名，可以使用分组上的二级域名）

```
http://e710888d3ccb4638a723ff8d03837095-cn-qingdao.aliapi.com/demo/post
```

请求方法：

```
POST
```

请求体：

```
FormParam1=FormParamValue1&FormParam2=FormParamValue2
//HTTP Request Body
```

请求头部：

```
Host: e710888d3ccb4638a723ff8d03837095-cn-qingdao.aliapi.com
Date: Mon, 22 Aug 2016 11:21:04 GMT
User-Agent: Apache-HttpClient/4.1.2 (java 1.6)
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
//请求体类型，请根据实际请求体内容设置。

Accept: application/json
//请求响应体类型，部分 API 可以根据指定的响应类型来返回对应数据格式，建议手动指定此请求头，如果不设置，部分 HTTP 客户端会设置默认值 */*，导致签名错误。

X-Ca-Request-Mode: debug
//是否开启 Debug 模式，大小写不敏感，不设置默认关闭，一般 API 调试阶段可以打开此设置。

X-Ca-Version: 1
// API 版本号，目前所有 API 仅支持版本号『1』，可以不设置此请求头，默认版本号为『1』。

X-Ca-Signature-Headers: X-Ca-Request-Mode,X-Ca-Version,X-Ca-Stage,X-Ca-Key,X-Ca-Timestamp
//参与签名的自定义请求头，服务端将根据此配置读取请求头进行签名，此处设置不包含 Content-Type、Accept、Content-MD5、Date 请求头，这些请求头已经包含在了基础的签名结构中，详情参照请求签名说明文档。

X-Ca-Stage: RELEASE
//请求 API 的 Stage，目前支持 TEST、PRE、RELEASE 三个 Stage，大小写不敏感，API 提供者可以选择发布到哪个 Stage，只有发布到指定 Stage 后 API 才可以调用，否则会提示 API 找不到或 Invalid Url。

X-Ca-Key: 60022326
//请求的 AppKey，请到 API 网关控制台生成，只有获得 API 授权后才可以调用，通过云市场等渠道购买的 API 默认已经给 APP 授过权，阿里云所有云产品共用一套 AppKey 体系，删除 ApppKey 请谨慎，避免影响到其他已经开通服务的云产品。

X-Ca-Timestamp: 1471864864235
//请求的时间戳，值为当前时间的毫秒数，也就是从1970年1月1日起至今的时间转换为毫秒，时间戳有效时间为15分钟。

X-Ca-Nonce:b931bc77-645a-4299-b24b-f3669be577ac
//请求唯一标识，15分钟内 AppKey+API+Nonce 不能重复，与时间戳结合使用才能起到防重放作用。

X-Ca-Signature: FJleSrCYPGCU7dMlLTG+UD3Bc5Elh3TV3CWHtSKh1Ys=
//请求签名。

CustomHeader: CustomHeaderValue
//自定义请求头，此处仅作为示例，实际请求中根据 API 定义可以设置多个自定义请求头。
```

2.响应

状态码：

```
400
//响应状态码，大于等于200小于300表示成功；大于等于400小于500为客户端错误；大于500为服务端错误。
```

响应头：

```
X-Ca-Request-Id: 7AD052CB-EE8B-4DFD-BBAF-EFB340E0A5AF
//请求唯一 ID，请求一旦进入 API 网关应用后，API 网关就会生成请求 ID 并通过响应头返回给客户端，建议客户端与后端服务都记录此请求 ID，可用于问题排查与跟踪。

X-Ca-Error-Message: Invalid Url
//API网关返回的错误消息，当请求出现错误时 API 网关会通过响应头将错误消息返回给客户。

X-Ca-Debug-Info: {"ServiceLatency":0,"TotalLatency":2}
//当打开 Debug 模式后会返回 Debug 信息，此信息后期可能会有变更，仅用做联调阶段参考
```

您调用API时，无论使用HTTP还是HTTPS协议提交请求，都需要在请求中包含签名信息。AppKey用于标识您的身份，AppSecret是用于加密签名字符串和服务器端验证签名字符串的密钥。详细加密签名的计算传递方式，请查看文档使用摘要签名认证方式调用API。

签名的计算demo请参照API网关控制台SDK下载页面的SDK示例。

若需要了解更多详情，请您查看概述。

3.API授权

API授权是指应用（APP）与API建立授权关系。应用（APP）是调用API时的身份，应用（APP）需要获得API的授权才能调用该API。

前提条件

API认证方式为"阿里云APP认证"

1 应用（APP）

应用（APP）是调用API的身份，每个APP有一对AppKey和AppSecret密钥对，AppKey需要在请求时作为参数在Header传入，AppSecret需要用于计算请求签名。详细摘要签名的计算传递方式，请查看文档使用摘要签名认证方式调用API。

- AppKey和AppSecret密钥对，具备该APP的全部权限，需要妥善保管。如果发生泄漏，您可以在API网关的控制台进行重置。
- 可以创建多个APP，根据业务需求分别授权给不同的API。注意，API的授权对象是APP而不是阿里云用户账号。
- 可以在API网关控制台完成对APP的创建、修改、删除、查看详情、密钥管理、查看已授权API等管理操作。

创建应用（APP）：

登录API网关控制台，左侧栏选择**调用API——应用管理**，点击右上角的**创建APP**即可。

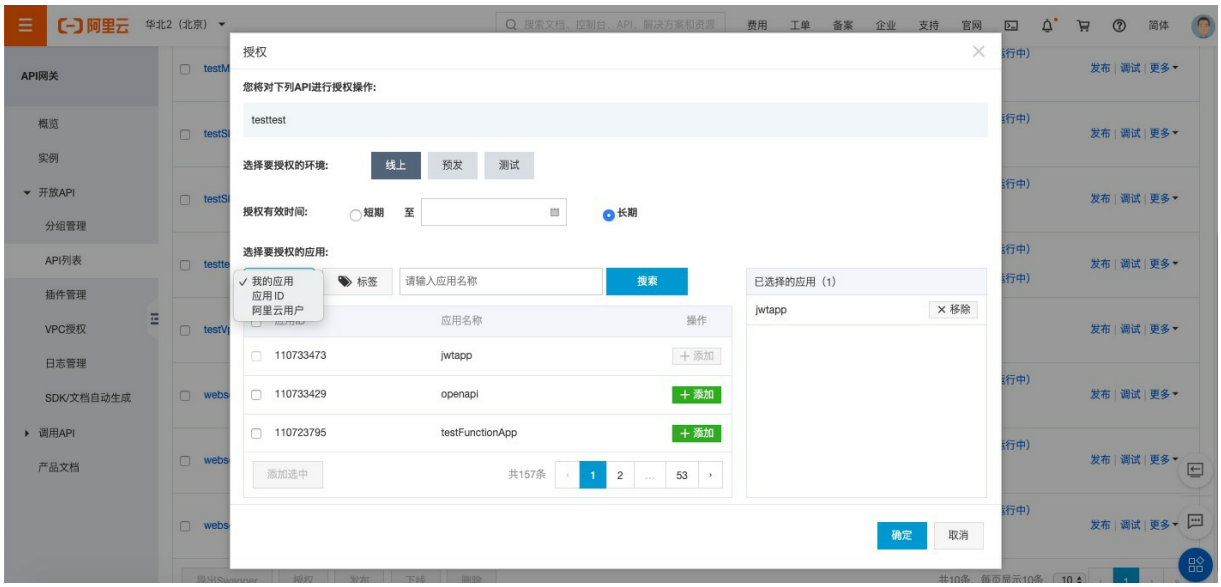
2 API授权

授权，是指授予某个APP调用某个API的权限。APP需要获得API的授权才能调用该API。

- 自己创建的APP，API可以直接授权。
- 云市场购买的API，用户有权操作已购买的API授权给自己的APP；如果没有APP，购买时云市场会为您创建一个APP，并且授权，具体请前往云市场的控制台查看。
- 如果想要使用合作伙伴（其他账号）提供的API，并无购买行为，是线下协议。那么需要自行创建APP，并且向API的提供者提供您的应用ID。API的提供者可以通过搜索应用ID来操作授权。

API授权：

2.1 登录API网关控制台，在左侧栏选择**开放API——API列表**，点击API后的**更多**，选择**授权**。



2.2 选择要授权的环境，和要授权的应用。左侧为我的应用，此时直接点击搜索，会自动加载本账号下的APP。

如果要给其他账号下APP授权，需要选择应用ID，然后在文本框中输入对应的应用ID，再点击搜索进行查询。

4.使用简单认证（AppCode）方式调用API

阿里云API网关提供多种针对客户端请求的安全认证方式，包括阿里云APP认证方式、OpenID Connect等。对于”阿里云APP“这种认证方式，目前用户可以设置两种认证形式： 1. 签名认证； 2. 简单认证。

1.概述

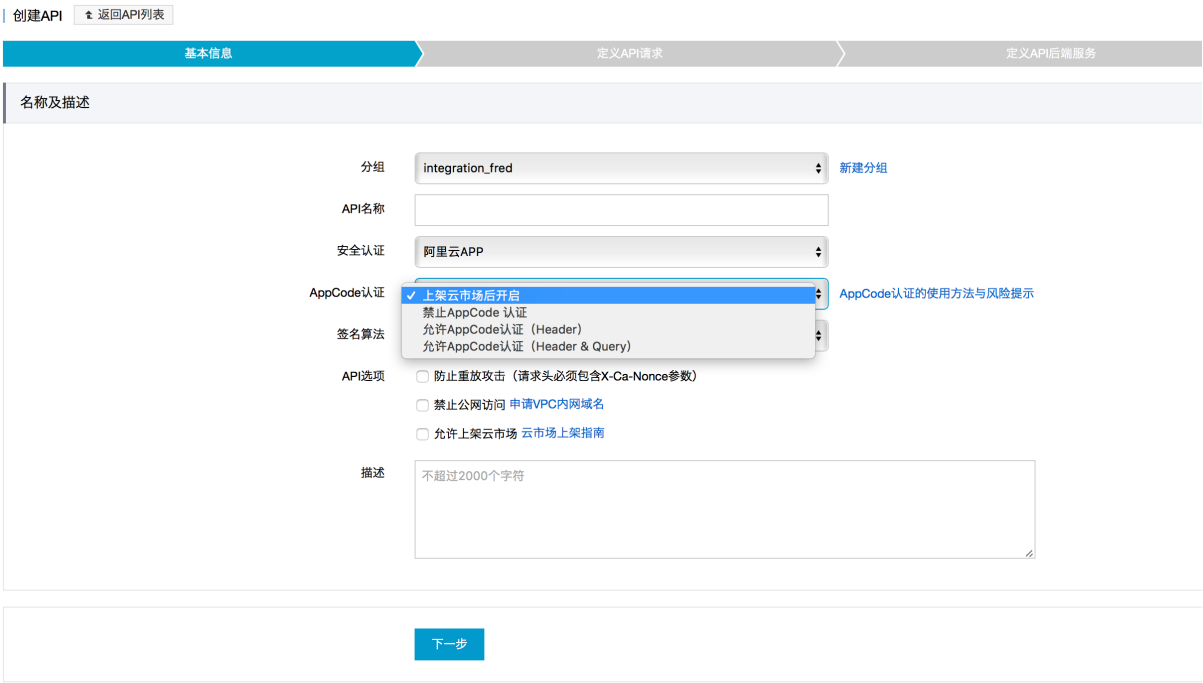
对于请求的签名认证方式，可以参考这个文档：请求签名说明文档：使用摘要签名认证方式调用API。

本文将详细描述简单认证方式的设置方式和调用方式。简单认证，顾名思义，和签名认证方式相比要简单很多，省去了复杂的生成签名的过程。简单认证方式直接使用API网关颁发的AppCode进行身份认证，调用者将AppCode放到请求头中，或者放到请求的Query参数中进行身份认证，实现快速调用API的能力。下面是对整个流程的描述：

1. API提供者创建API的时候选择”阿里云APP“认证模式，且支持AppCode认证（所有云市场的API默认都支持AppCode）；
2. API调用者在API网关”应用管理“创建一个APP。云市场用户在云市场购买API时云市场会为您创建一个APP；
3. API提供者给调用者的APP进行API授权，具体授权方式请参考文档： 创建后端服务为HTTP的API；
4. API调用者到API网关控制台的”应用管理“找到AppCode/AppSecret进行签名认证的调用或者AppCode进行简单认证的API调用。

2. 创建支持简单认证方式API

1. API提供者在创建API安全认证方式时需要选择”阿里云APP”，或者选择”OpenId Connect & 阿里云APP“；
2. AppCode认证选项选择允许AppCode认证相关的几个选项；



下面解释下AppCode认证四个选项的含义进行详细解释：

- 上架云市场后开启：默认不开启，如果上架API上架云市场，则支持将AppCode放在Header中进行认证；
- 禁止AppCode认证：无论API是否上架云市场，都不开启，都需要使用签名方式调用；
- 允许AppCode认证（Header）：无论API是否上架云市场，都开启，但只支持将AppCode放在Header中进行认证；
- 允许AppCode认证（Header & Query）：无论API是否上架云市场，都开启，同时支持将AppCode放在Header中，或者将AppCode放在Query中进行认证；

注意：定义API参数的时候，不需要定义携带AppCode的头或者Header参数。

3. 调用方法

API提供者将API设置成允许AppCode调用之后，API调用者就可以使用简单认证方式进行调用了，不用再在客户端实现复杂的签名算法了。本章我们介绍下如何通过简单认证方式调用API。主要有两种方式，一种是将AppCode放在Header中进行调用，一种是将AppCode放在Query参数中进行调用。

3.1 将AppCode放在Header中

- 请求Header中添加一个”Authorization“参数；
- Authorization字段的值的格式为“APPCODE + 半角空格 + APPCODE值”。

格式：

```
Authorization:APPCODE AppCode值
```

示例：

```
Authorization:APPCODE 3F2504E04F8911D39A0C0305E82C3301
```

3.2 将AppCode放在Query中

- 请求Query中添加的“AppCode”参数（同时支持”appcode”，“appCode”，“APPCODE”，“APPCode”四种写法）；
- “AppCode”参数的值为AppCode的值。

示例：

```
http://www.aliyum.com?AppCode=3F2504E04F8911D39A0C0305E82C3301
```

4.风险提示

简单认证方式调用非常省事，免去了复杂的签名过程，但是把AppCode作为明文暴露网络中传输，会带来一些安全隐患，务必重视：

客户端和API网关之间务必使用HTTPS进行通信，避免使用HTTP/WebSocket协议进行数据传输。因为简单认证方式，AppCode在传输过程中使用明文，而HTTP/WebSocket通信协议没有加密，一旦网络通信的网络包被黑客抓取，有非常大的丢失AppCode的风险。

5.使用SDK调用API

本文主要引导用户如何通过调用API网关自动生成的SDK，来在业务系统中调用发布好的API。

概述

在控制台自动生成的SDK内置了签名实现，可以免去繁琐的签名计算，若需要自行实现签名认证您可以参考客户端签名说明文档

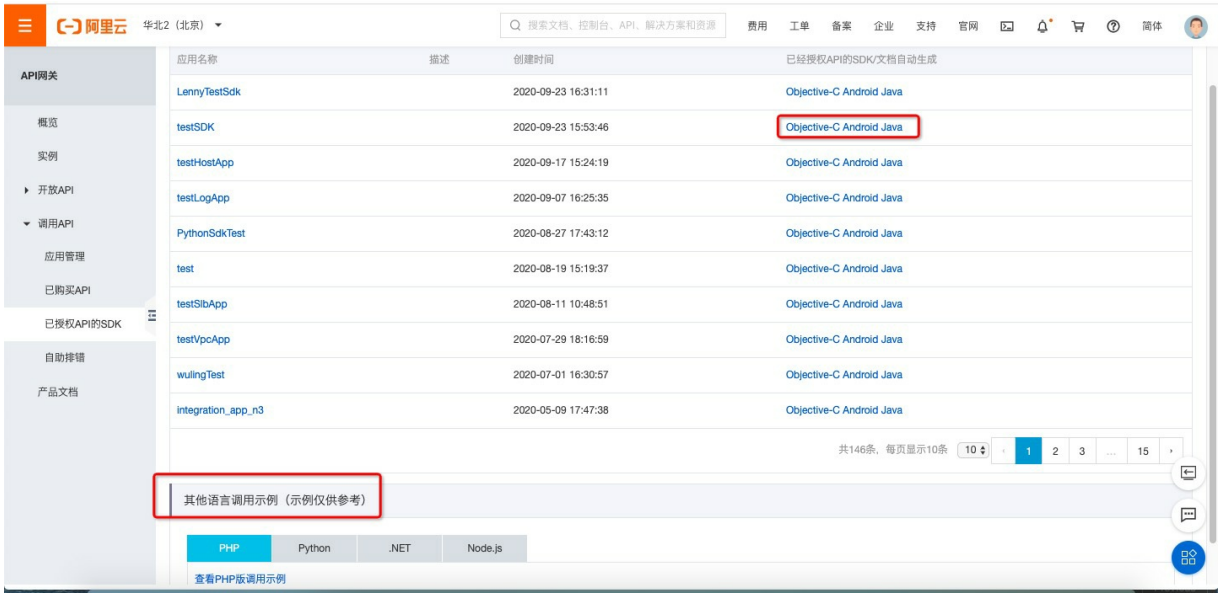
调用API的前期步骤可参考创建后端服务为HTTP的API，支持SDK调用的API需满足：

- API的安全认证方式为“阿里云APP”或“OpenID Connect & 阿里云APP”。
- API与APP建立授权关系。
- API已经发布到线上。

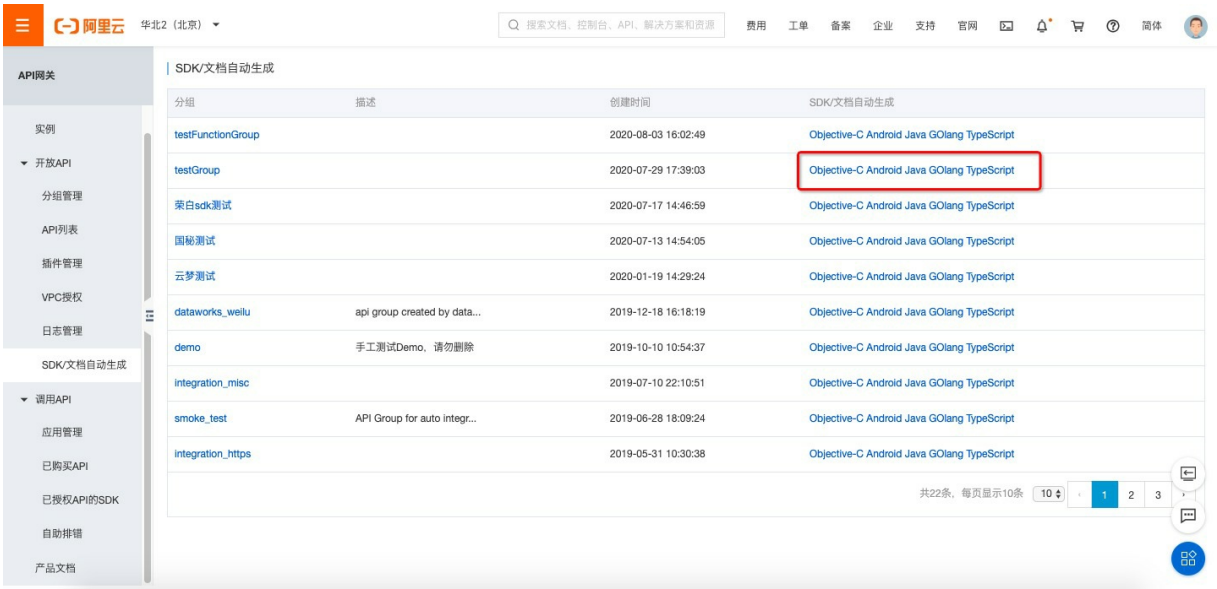
1. 下载SDK

控制台提供两种SDK的下载方式，如下：

1.1 在 `调用API` 的 `已授权API的SDK` 中下载应用APP的SDK，此SDK是基于APP生成的。目前支持自动生成的SDK语言包括Objective-C 、Android 、Java。其他语言调用示例在页面下方。



1.2 在 `开放API` 的 `SDK/文档自动生成` 中下载API分组的SDK。此SDK是基于分组生成的。目前支持自动生成的SDK语言包括Objective-C 、Android 、Java、GOlang、TypeScript。



2. 使用SDK调用API

通过控制台下载的SDK，下载解压后的目录结构如图：

- SDK文件夹

* sdk/{{regionId}}`JavaSDK文件夹，包含每个Group的所有API的接口调用代码`

* HttpApiClient{{group}}.java `包含对应Group所有HTTP通道的API方法`

* HttpsApiClient{{group}}.java`包含对应Group所有HTTPS通道API方法`

* WebSocketApiClient{{group}}.java `包含对应Group所有WebSocket通道的API法`

* Demo{{group}}.java `包含对应Group所有API调用示例`

- doc/{{regionId}}文件夹

* ApiDocument_{{group}}.md`对应Group的API接口文档`

- lib文件夹

* sdk-core-java-1.1.5.jar `sdk的core包，为本sdk的依赖包`

* sdk-core-java-1.1.5-sources.jar`上述依赖包的源码`

* sdk-core-java-1.1.5-javadoc.jar`core包的文档`

Readme.md`本SDK使用指南`

LICENSE `版权许可`

其中Readme.md文档为本SDK的使用指南，有详细的步骤，请参考**Readme.md**文档来实现SDK的调用。

重要提示：“AppKey”和“AppSecret”是网关认证用户请求的密钥，这两个配置如果保存在客户端，请妥善加密。

3. 排错

使用SDK调用API若出现报错，您可以在请求的ResponseHeader中获取RequestId，然后根据自助排错文档来排查。

6.使用摘要签名认证方式调用API

发布的API如果使用摘要签名认证方式（APP Key和APP Secret），客户端在调用API时，需要使用签名秘钥对请求内容进行签名计算，并将签名同步传输给服务器端进行签名验证。API网关提供的SDK内置了签名实现，您只需要将签名秘钥配置在SDK中，即可实现发起携带正确签名的请求。如果您需要自己在客户端实现签名计算过程，可以参考本文档。

1 摘要签名认证方式

API网关提供前端签名及验签能力，前端签名及验签主要两点用途：

- 验证客户端请求的合法性，确认请求中携带授权后的AK生成的签名；
- 防止请求数据在网络传输过程中被篡改。

API的拥有者可以在API网关控制台的应用管理页面生成APP，每个APP会携带一对签名秘钥（APP Key和APP Secret）,API拥有者将API授权给指定的APP（APP可以是API拥有者颁发或者API调用者所有）后，API调用者就可以用APP的签名秘钥来调用相关的API了。

客户端调用 API 时，需要使用已授权签名秘钥对请求内容的关键数据进行加密签名计算，并且将APP Key和加密后生成的字符串放在请求的 Header 传输给API网关，API网关会读取请求中的APP Key的头信息，并且根据APP Key的值查询到对应的APP Secret的值，使用APP Secret对收到的请求中的关键数据进行签名计算，并且使用自己的生成的签名和客户端传上来的签名进行比对，来验证签名的正确性。只有签名验证通过的请求才会发送给后端服务，否则API网关会认为该请求为非法请求，直接返回错误应答。

API网关只会验证安全认证类型为“阿里云APP”或者“OpenID Connect & 阿里云APP”两种类型的API请求的签名，其他安全类型的API请求不会验证签名。

2 使用SDK调用

签名计算的详细实现可以参考API网关提供的SDK，在API网关控制台的下面两个页面中可以下载Java/Android/Objective-C的带源码的SDK：

- 开放API-SDK/文档自动生成
- 调用API-已授权的API的SDK

具体可参考使用SDK调用API

3 摘要签名认证方式原理说明

3.1签名生成流程和认证流程概述

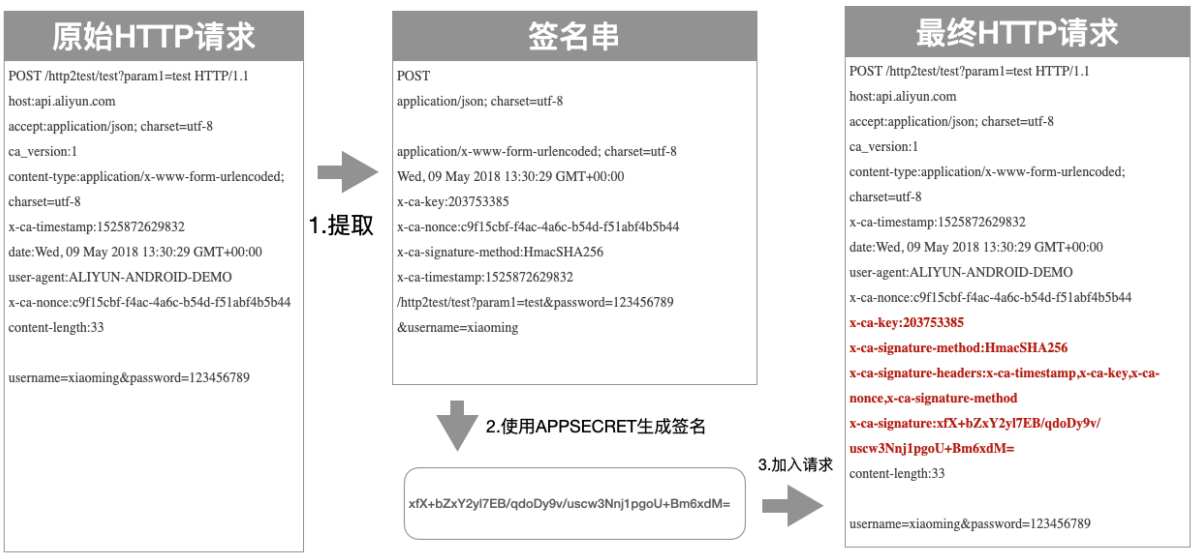
3.1.1 前置条件

API的调用方需要在调用API之前获取到已经授权给对应API的签名秘钥对：

- APP Key
- APP Secret

被调用的API的安全认证类型为“阿里云APP”或者“OpenID Connect & 阿里云APP”两种类型之一。

3.1.2 客户端生成签名



客户端生成签名一共分三步处理：

1. 从原始请求中提取关键数据，得到一个用来签名的字符串；
2. 使用加密算法加APP Secret对关键数据签名串进行加密处理，得到签名；
3. 将签名所相关的所有头加入到原始HTTP请求中，得到最终HTTP请求。

3.1.3 服务器端验证签名

服务器验证客户端签名一共分四步处理：

1. 从接收到的请求中提取关键数据，得到一个用来签名的字符串；
2. 从接收到的请求中读取APP Key，通过APP Key查询到对应的APP Secret；
3. 使用加密算法和APP Secret对关键数据签名串进行加密处理，得到签名；
4. 从接收到的请求中读取客户端签名，对比服务器端签名和客户端签名的一致性。

3.2 生成与传递签名

3.2.1 提取签名串

客户端需要从Http请求中提取出关键数据，组合成一个签名串，生成的签名串的格式如下：

```
HTTPMethod
Accept
Content-MD5
Content-Type
Date
Headers
PathAndParameters
```

以上7个字段构成整个签名串，字段之间使用\n间隔，如果Headers为空，则不需要加\n，其他字段如果为空都需要保留\n。签名大小写敏感。下面介绍下每个字段的提取规则：

- HTTPMethod：HTTP的方法，全部大写，比如POST
- Accept：请求中的Accept头的值，可为空。建议显式设置 Accept Header。当 Accept 为空时，部分 Http 客户端会给 Accept 设置默认值为 */*，导致签名校验失败。
- Content-MD5：请求中的Content-MD5头的值，可为空只有在请求存在Body且Body为非Form形式时才计算Content-MD5头，下面是Java的Content-MD5值的参考计算方式：

```
String content-MD5 = Base64.encodeBase64(MD5(bodyStream.getbytes("UTF-8")));
```

- Content-Type：请求中的Content-Type头的值，可为空

> **注意**
>
> 当客户端使用微信小程序来传输文件或是底层一些逻辑导致签名的Content-Type异常时，可以增加一个自定义Content-Type头"X-Ca-Signed-Content-Type:multipart/form-data"，客户端使用这个头进行签名，网关也会用这个头进行签名。

- Date：请求中的Date头的值，可为空
- Headers：用户可以选取指定的header参与签名，关于header的签名串拼接方式有以下规则：
  - 参与签名计算的Header的Key按照字典排序后使用如下方式拼接

```
HeaderKey1 + ":" + HeaderValue1 + "\n"\+
HeaderKey2 + ":" + HeaderValue2 + "\n"\+
...
HeaderKeyN + ":" + HeaderValueN + "\n"
```

  - 某个Header的Value为空，则使用HeaderKey+":"+"\n"参与签名，需要保留Key和英文冒号。
  - 所有参与签名的Header的Key的集合使用英文逗号分割放到Key为X-Ca-Signature-Headers的Header中；
  - 以下Header不参与Header签名计算：X-Ca-Signature、X-Ca-Signature-Headers、Accept、Content-MD5、Content-Type、Date。
- PathAndParameters这个字段包含Path，Query和Form中的所有参数，具体组织形式如下：

```
Path + "?" + Key1 + "=" + Value1 + "&" + Key2 + "=" + Value2 + ... "&" + KeyN + "=" + Val
ueN
```

  - Query和Form参数对的Key按照字典排序后使用上面的方式拼接；
  - Query和Form参数为空时，则直接使用Path，不需要添加？；

- 参数的Value为空时只保留Key参与签名，等号不需要再加入签名；
- Query和Form存在数组参数时（key相同，value不同的参数），取第一个Value参与签名计算。

下面我们看一个例子，这里有一个普通的HTTP请求：

```
POST /http2test/test?param1=test HTTP/1.1
host:api.aliyun.com
accept:application/json; charset=utf-8
ca_version:1
content-type:application/x-www-form-urlencoded; charset=utf-8
x-ca-timestamp:1525872629832
date:Wed, 09 May 2018 13:30:29 GMT+00:00
user-agent:ALIYUN-ANDROID-DEMO
x-ca-nonce:c9f15cbf-f4ac-4a6c-b54d-f51abf4b5b44
content-length:33
username=xiaoming&password=123456789
```

生成的正确签名串为：

```
POST
application/json; charset=utf-8
application/x-www-form-urlencoded; charset=utf-8
Wed, 09 May 2018 13:30:29 GMT+00:00
x-ca-key:203753385
x-ca-nonce:c9f15cbf-f4ac-4a6c-b54d-f51abf4b5b44
x-ca-signature-method:HmacSHA256
x-ca-timestamp:1525872629832
/http2test/test?param1=test&password=123456789&username=xiaoming
```

3.2.2 计算签名

客户端从HTTP请求中提取出关键数据组装成签名串后，需要对签名串进行加密及编码处理，形成最终的签名，目前API网关支持以下两种加密算法：

- HmacSHA256
- HmacSHA1

具体的加密形式如下，其中stringToSign是提取出来的签名串，secret是APP Secret：

```
Mac hmacSha256 = Mac.getInstance("HmacSHA256");
byte[] keyBytes = secretKey.getBytes(SdkConstant.CLOUDAPI_ENCODING);
hmacSha256.init(new SecretKeySpec(keyBytes, 0, keyBytes.length, "HmacSHA256"));
```

```
Mac hmacSha1 = Mac.getInstance("HmacSHA1");
hmacSha1.init(new SecretKeySpec(secret.getBytes("UTF-8"), 0, keyBytes.length, "HmacSHA1"));
byte[] md5Result = hmacSha1.doFinal(stringToSign.getBytes("UTF-8"));
String sign = Base64.encodeBase64String(md5Result);
```

抽象一下就是将串（StringToSign）使用UTF-8解码后得到Byte数组，然后使用加密算法对Byte数组进行加密，然后使用Base64算法进行编码，形成最终的签名。

3.2.3 传输签名

客户端需要将以下四个Header放在HTTP请求中传输给API网关，进行签名校验：

- x-ca-key：取值APP Key，必选；
- x-ca-signature-method：签名算法，取值HmacSHA256或者HmacSHA1，可选，默认值为HmacSHA256；
- X-Ca-Signature-Headers：所有签名头的Key的集合，使用英文逗号分隔，可选；
- X-Ca-Signature：签名，必选；

下面是携带签名的整个HTTP请求的示例：

```
POST /http2test/test?param1=test HTTP/1.1
host:api.aliyun.com
accept:application/json; charset=utf-8
ca_version:1
content-type:application/x-www-form-urlencoded; charset=utf-8
x-ca-timestamp:1525872629832
date:Wed, 09 May 2018 13:30:29 GMT+00:00
user-agent:ALIYUN-ANDROID-DEMO
x-ca-nonce:c9f15cbf-f4ac-4a6c-b54d-f51abf4b5b44
x-ca-key:203753385
x-ca-signature-method:HmacSHA256
x-ca-signature-headers:x-ca-timestamp,x-ca-key,x-ca-nonce,x-ca-signature-method
x-ca-signature:xfX+bZxY2yl7EB/qdoDy9v/uscw3Nnj1pgoU+Bm6xdM=
content-length:33
username=xiaoming&password=123456789
```

3.3 签名排错方法

API网关签名校验失败时，会将服务端的签名串（StringToSign）放到HTTP Response的Header中返回到客户端，Key为：X-Ca-Error-Message，用户只需要将本地计算的签名串（StringToSign）与服务端返回的签名串进行对比即可找到问题；

如果服务端与客户端的StringToSign一致请检查用于签名计算的APP Secret是否正确；

因为HTTP Header中无法表示换行，因此StringToSign中的换行符都被替换成#。

```
errorMessage:  Invalid Signature, Server StringToSign:`GET#application/json##application/json##X-Ca-Key:200000#X-Ca-Timestamp:1589458000000#/app/v1/config/keys?keys=TEST`
```

说明服务器的签名是：

```
GET
application/json
application/json
X-Ca-Key:200000
X-Ca-Timestamp:1589458000000
/app/v1/config/keys?keys=TEST
```

7.VPC内网访问API网关

API网关支持用户通过公网或VPC内网访问API网关，本文重点介绍如何在VPC内网访问到API网关。

1. 概述

VPC内网访问时，需要通过每个分组的“内网VPC二级域名”来进行访问，该域名有如下特点：

- 内网VPC二级域名只能在VPC内使用，支持直接访问，且没有每天1000次调用限制。
- 内网VPC二级域名不支持HTTPS访问，如果要实现HTTPS访问，请绑定自己域名。

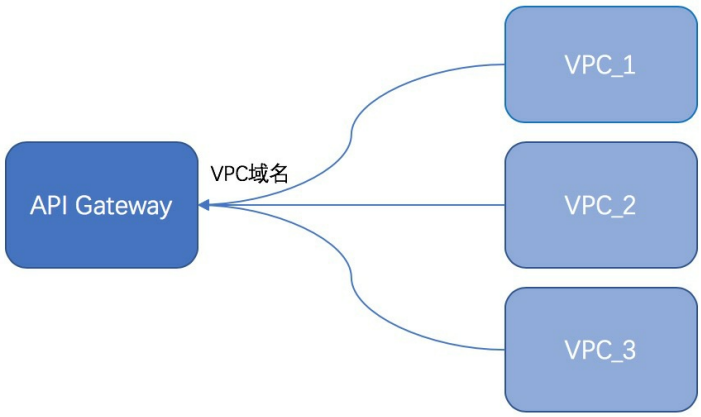
不同的实例类型，内网VPC二级域名的开放方法和生效范围不同，具体参考如下两个章节说明。

如果您希望在混合云结构下，通过VPC内网访问到API网关，如通过云企业网（CEN）、VPN（如IPsec VPN）搭建的混合云结构。建议您使用专享实例的API网关，此时运行在专享实例上的API分组，其内网VPC二级域名所对应的IP为您所绑定的入访VPC的私网地址，这样便于您的本地路由设置。共享实例上的API分组的内网VPC二级域名统一为100.x.x.x的IP地址，无法分配私网地址。

关于采用混合云情况下的更多配置，可参考混合云API集中管理

2. 共享实例设置方式

API网关共享实例的内网访问支持同region所有用户VPC访问。

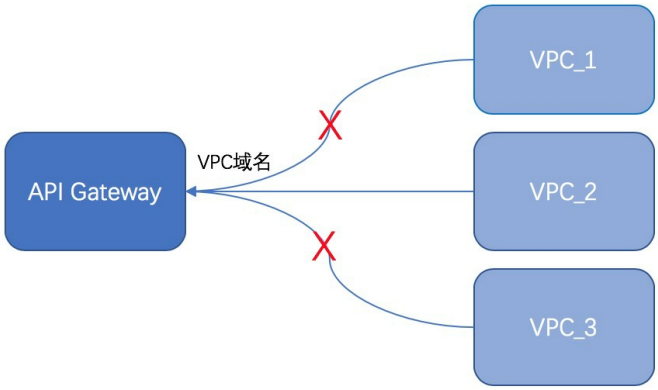


配置方法：

在API网关控制台->开放API->分组管理->分组详情，点击“开通VPC二级域名”，网关会给自动给该分组创建一个内网VPC二级域名，通过该域名可以直接访问API。

3. 专享实例设置方式

专享实例的VPC内网访问能力仅只对某一个VPC开放，其他VPC则无法通过内网方式访问该实例下的API，这种方式更加安全。



配置方法：

1、请到API网关控制台->实例页面对应的专享实例上配置允许访问API的VPC。

2、在该实例下分组详情页面中，开通内网访问，即可通过内网二级域名访问API（或者通过CNAME解析到该二级域名后，绑定自己的域名访问API）。

注意事项：

- 专享实例上未配置“入访VPC”，则分组不能开通内网VPC域名。
- 当专享实例上“入访VPC”发生变更，该实例下的所有分组的VPC二级域名将重新配置为对新的VPC开放，原来的VPC就不能再访问该API了。
- 当分组从共享实例（经典或者VPC）迁移到专享实例时，如果分组上开通了内网VPC域名，需要先开通专享实例的入访VPC后，才能完成实例的迁移。**请注意：此时通过内网访问API，只能从入访VPC发起访问。**

8.函数计算内网访问API网关

本文介绍在函数计算中如何通过VPC访问API网关，包括两种情况：在同一Region，以及在跨Region情况下如何访问。

1 概述

API网关即能够和函数计算集成，构建起serverless架构，同时在实际场景中，也会出现在函数中需要调用API网关上发布的API，同时很多时候出于安全考虑，往往会希望函数从内网能够访问API。因此本文着重介绍两个场景的实现方式：

- 在同一Region内，函数计算如何内网访问API网关
- 跨Region情况下，函数计算如何内网访问API网关

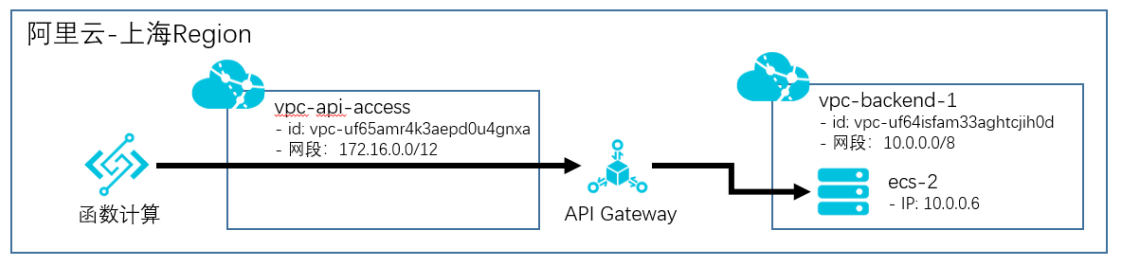
无论是哪种场景，主要的配置原则如下：

- 需要基于VPC实现内网访问；
- API网关的内网访问，需要设定一个VPC允许API网关接入，具体过程详见 VPC访问API网关；
- 函数计算的内网访问，也需要通过VPC实现，具体过程详见 配置网络 。

2 场景1 同一Region内访问

步骤1： 准备工作

构建如下图所示的结构：



说明：

- 在上海Region内创建2个VPC，分别为vpc-api-access和vpc-backend-1；
- 在上海Region内创建了1个API网关专享实例；
- 在vpc-backend-1中，创建一个ecs实例做为API网关的后端服务，此ecs实例提供了http服务接口可对外访问，且对外可以访问的http服务地址为 http://localhost:8080/web/cloudapi 。同时为此ecs实例也设置了对应的安全组，允许API网关的访问。

步骤2： 配置后端服务类型为VPC的API

此步骤的详细过程可参见使用VPC内资源作为API的后端服务，配置要点如下：

- 创建VPC授权，创建成功后如下图所示：

- 创建后端服务为VPC的API，为了方便后续的调用测试，API使用无认证方式，如下图所示：

- API保存完后，需要进行发布，本例为了测试方便，发布到“线上”环境。

步骤3：配置VPC到API网关的内网访问权限

在专享实例页面。点击“入访VPC”后的选择VPC，选择vpc-api-access的vpc id，表示可以通过vpc-api-access内网访问到API网关。

步骤4： 开通API分组的内网域名

在分组详情页面，开通内网二级域名，点击开通后，API网关会给分组分配一个内网VPC二级域名。该域名可以直接调用该分组下的API。

注意：API分组默认开通互联网访问，您可以根据业务情况通过关闭公网二级域名来停止互联网访问，但注意禁止后，将不能通过API网关控制台进行在线调试。

步骤5： 新建函数

在函数计算中，创建应用，并创建运行环境为python的函数，函数内容如下所示：

函数中仅是使用curl去访问vpc二级域名下的API，如果此时执行，将无法访问。

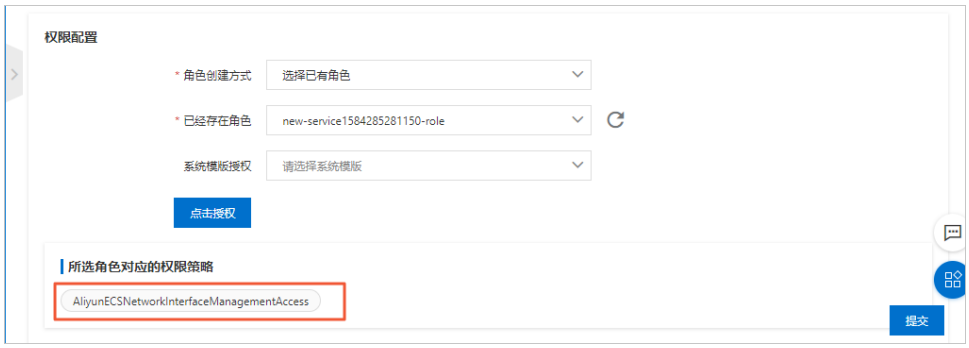
步骤6： 配置函数计算的VPC访问

首先需要在vpc-api-access中再创建一个vswitch，用于函数计算的接入，如下图所示：

其次在函数计算控制台中，在上一步骤中新建应用的配置服务菜单中进行配置，具体过程以函数计算的文档为准 配置网络。

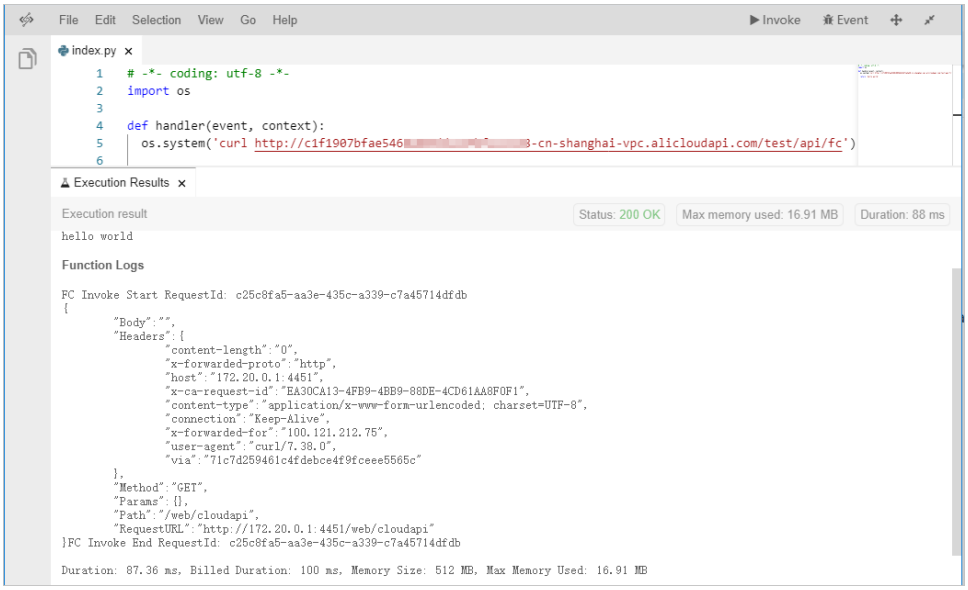
在专有网络配置中，专有网络选择vpc-api-access，交换机选择本步骤创建的vswitch。

权限配置中，需要新增一个角色，系统模板授权选择AliyunECSNetworkInterfaceManagementAccess，按控制台操作向导执行完后如下图所示：



步骤7： 执行函数进行测试

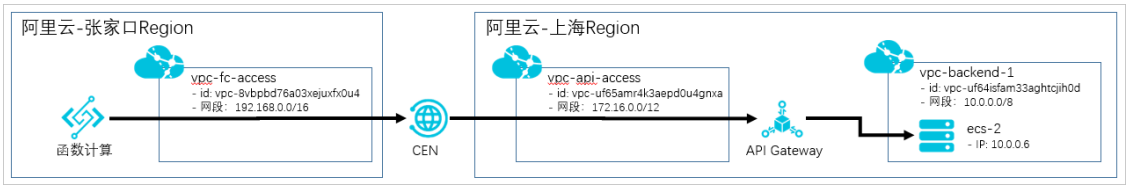
执行后可以看到函数计算已经可以通过vpc二级域名访问到API。



3 场景2 跨Region访问

步骤1：准备工作

构建如下图所示的结构：



说明：

- 张家口Region内，创建1个函数计算应用，用于本例中发起API调用请求；创建1个VPC（vpc-fc-access），用于帮助函数计算接入云企业网（CEN）；
- 上海Region内，1个API网关专享实例；同时1个VPC（vpc-api-access），用于帮助API网关接入云企业网（CEN）；
- 上海Region内，创建VPC（vpc-backend-1），并创建一个ecs实例做为API网关的后端服务，此ecs实例提供了http服务接口可对外访问，且对外可以访问的http服务地址为 http://localhost:8080/web/cloudapi 。同时为此ecs实例也设置了对应的安全组，允许API网关的访问。

步骤2：创建云企业网

首先创建一个云企业网（CEN），打通上海VPC(vpc-api-access)和张家口的VPC(vpc-fc-access)，实现内网互通。关于CEN的更多配置方式，请详见云企业网帮助文档 。先通过 云企业网控制台 创建一个云企业网实例，多次加载网络实例，将vpc-api-access，vpc-fc-access都添加到云企业网中，完成后如下图所示。

步骤3：配置带宽

购买一个带宽包，作为云企业网内通信需要。本例购买了一个最低的2M的带宽，您可以根据实际需要按需购买。

配置跨地域带宽设置，指定的互通地域配置带宽值，还可以将1个带宽包拆分到多个互通地域中。

步骤4：配置跨VPC路由

本步骤需要给CEN团队提工单。注意按照 ResolveAndRouteServiceInCen 接口的参数说明，提供配置信息。打通API网关和张家口VPC的互通。

```
AccessRegionIds.1=cn-zhangjiakou
AccessRegionIds.2=cn-shanghai
CenId=cen-uggzcthgz7cwsl7prr       #云企业网的实例ID
Host=100.104.255.128/26                #通过API网关专享实例内网VPC出口地址
HostRegionId=cn-shanghai
HostVpcId=vpc-uf65amr4k3aepd0u4gnxa      #API网关在上海region，这个是vpc-api-access的VPCID
```

其中API网关专享实例接入内网VPC的出口地址，可以在实例管理中查询到，如下图所示

工单回复配置完成后，在云企业网控制台查看配置的路由，能看到上海和张家口region都加了一些路由策略。而且都有一条自定义路由，是根据上面提供的信息添加的。

步骤5： 配置后端服务类型为VPC的API

同 **场景1** 中的 **步骤2：** 配置后端服务类型为VPC的API。

步骤6： 配置VPC到API网关的内网访问权限

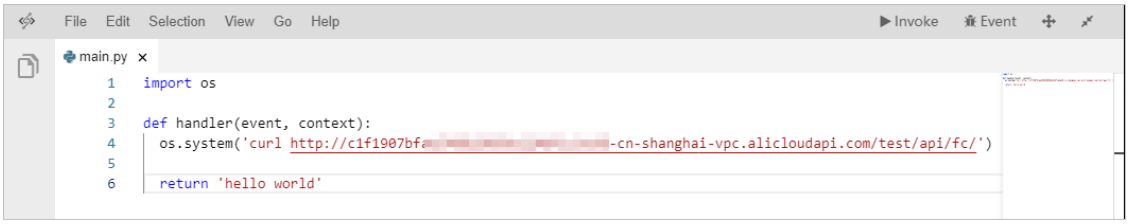
同 **场景1** 中的 **步骤3：** 配置VPC到API网关的内网访问权限。

步骤7： 开通API分组的内网域名

同 **场景1** 中的 **步骤4**：开通API分组的内网域名。

步骤8：新建函数

在张家口Region的函数计算中，创建应用，并创建运行环境为python的函数，函数内容如下所示：

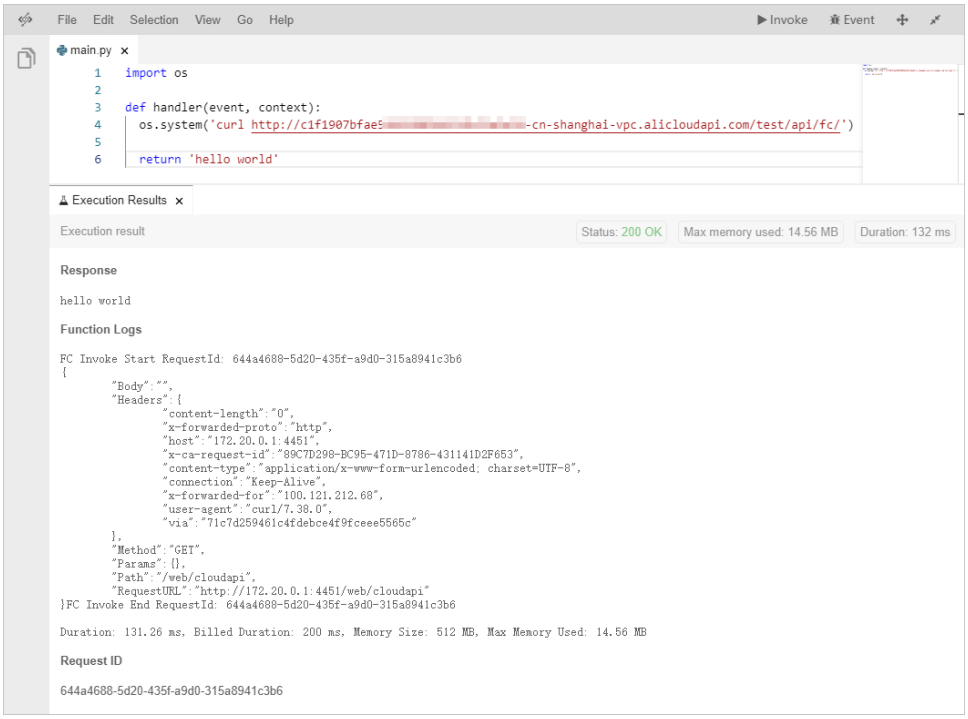


步骤9：配置函数计算的VPC访问

与 **场景1** 的 **步骤6**：配置函数计算的VPC访问类似，在vpc-fc-access中新建一个vswitch，然后在函数计算控制台中的配置服务菜单中进行配置。

步骤10：执行函数进行测试

执行后可以看到函数计算已经可以通过vpc二级域名访问到API。



4 使用限制

- 仅限API网关专享实例。

9.API网关双向通信SDK实现指南

阿里云API网关对外提供双向通信能力，官方提供的SDK的语言有限，有些语言需要用户自己去开发SDK，本文档把SDK的实现细节描述出来，供用户在其他语言上实现SDK时参考。API网关官方提供的Android和Objective-C的SDK，是完整支持双向通信的，开发同学也可以参考Android和iOS的Objective-C结合本文档来实现目标语言的SDK。

关于API网关提供的双向通信的使用流程请参见：双向通信使用指南。

1. 通信协议及相关命令字

客户端和API网关之间通过WebSocket通道进行通信，通信类型分两种：调用API和发送命令。在通道上制定了一些简单的通信规则：

1.1 API调用格式转换

客户端调用API，请求和应答是纯Json格式的字符串，也就是把HTTP请求对象按照Json的语法格式化后传输，就是将下面这个对象转化成Json格式：

```
public class WebSocketApiRequest {
    String method;
    String host;
    String path;
    Map<String , String> querys;
    Map<String, List<String>> headers;
    int isBase64 = 0;
    String body;
}
```

下面是一个示例，首先展示一个标准的HTTP请求报文：

```
POST /http2test/test?param1=test HTTP/1.1
host: apihangzhou444.foundai.com
x-ca-seq:0
x-ca-key:12344133
ca_version:1
content-type:application/x-www-form-urlencoded; charset=utf-8
x-ca-timestamp:1525872629832
date:Wed, 09 May 2018 13:30:29 GMT+00:00
x-ca-signature:kYjdIuCnCrxx+EyLMTTx5NiXxqvfTTHBQAe68tv33Tw=
user-agent:ALIYUN-ANDROID-DEMO
x-ca-nonce:c9f15cbf-f4ac-4a6c-b54d-f51abf4b5b44
x-ca-signature-headers:x-ca-seq,x-ca-key,x-ca-timestamp,x-ca-nonce
content-length:33

username=xiaoming&password=123456789
```

WebSocket在通信的时候会将这个HTTP请求报文格式化为下面的字符串格式进行传输：

```
{
        "headers": {
                "accept": ["application/json; charset=utf-8"],
                "host": ["apihangzhou444.foundai.com"],
                "x-ca-seq": ["0"],
                "x-ca-key": ["12344133"],
                "ca_version": ["1"],
                "content-type": ["application/x-www-form-urlencoded; charset=utf-8"],
                "x-ca-timestamp": ["1525872629832"],
                "date": ["Wed, 09 May 2018 13:30:29 GMT+00:00"],
                "x-ca-signature": ["kYjdIuCnCrxx+EyLMTTx5NiXxqvfTTHBQAe68tv33Tw="],
                "user-agent": ["ALIYUN-ANDROID-DEMO"],
                "x-ca-nonce": ["c9f15cbf-f4ac-4a6c-b54d-f51abf4b5b44"],
                "x-ca-signature-headers": ["x-ca-seq,x-ca-key,x-ca-timestamp,x-ca-nonce"]
        },
        "host": "apihangzhou444.foundai.com",
        "isBase64": 0,
        "method": "POST",
        "path": "/http2test/test",
        "querys": {"param1":"test"},
        "body":"username=xiaoming&password=123456789"
}
```

1.2 双向通信命令

在双向通信过程中，除了正常的API调用，还有一系列定制的命令：

1.2.1 客户端注册

命令字： RG
含义： 在API网关注册长连接，携带DeviceId
命令类型： 请求
发送端： 客户端
格式： RG#DeviceId
示例： RG#ffd3234343dae324342@12344133

命令字： RO
含义： DeviceId注册成功时，API网关返回成功，并将连接唯一标识和心跳间隔配置返回
命令类型： 应答
发送端： API网关
格式： RO#ConnectionCredential#keepAliveInterval
示例： RO#1534692949977#25000

失败命令字： RF
含义： API网关返回注册失败应答
命令类型： 应答
发送端： API网关
格式： RF#ErrorMessage
示例： RF#ServerError

1.2.2 客户端保持心跳

命令字：H1
含义：客户端心跳请求信令
命令类型：请求
发送端：客户端
没有其他参数，直接发送命令字

命令字：HO
含义：API网关返回心跳保持成功应答信令
命令类型：应答
发送端：API网关
格式：HO#ConnectionCredential
示例：HO#ffd3234343dae324342

命令字：HF
含义：API网关返回心跳保持失败应答信令，收到此应答需要客户端重新发送注册请求
命令类型：应答
发送端：API网关
格式：HF
没有其他参数，直接发送命令字

1.2.3 下行通知

命令字：NF
含义：API网关发送下行通知请求
命令类型：请求
发送端：API网关
格式为NF#MESSAGE
示例：NF#HELLO WORLD!

命令字：NO
含义：客户端返回接收下行通知应答
命令类型：应答
发送端：客户端
没有其他参数，直接发送命令字

1.2.4 其他信令

命令字：OS
含义：客户端请求量达到API网关流控阈值，API网关会给客户端发送这个命令，需要客户端主动断掉连接，主动重连。主动重连将不会影响用户体验。否则API网关不久后会主动断链长连接。
命令类型：请求
发送端：API网关
没有其他参数，直接发送命令字

命令字：CR
含义：连接达到长连接生命周期，API网关会给客户端发送这个命令，需要客户端主动断掉连接，主动重连。主动重连将不会影响用户体验。否则API网关不久后会主动断链长连接。
命令类型：请求
发送端：API网关
没有其他参数，直接发送命令字

1.2.5 设备唯一标识

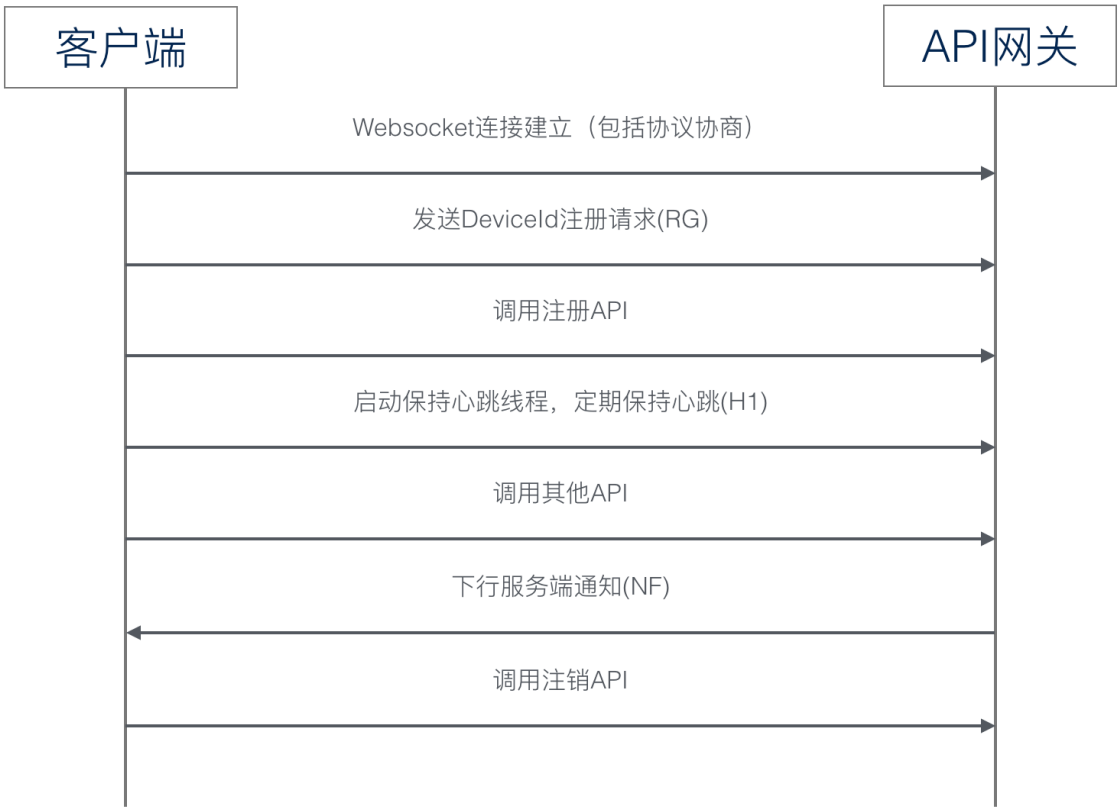
DeviceId唯一标记客户端的长连接，格式为UUID@AppKey，下面是建议的JAVA实现。所有DeviceId在APP维度必须是唯一的。也就是同一个APP，不允许出现重复的DeviceId，否则在注册的时候会报错。

```
String deviceId = UUID.randomUUID().toString().replace("-" , "")+ "@" + appKey;
```

下面是一个示例：ffd3234343dae324342@12344133。

2. 通信流程

下图是SDK和API网关之间交互流程图：



从上图中我们可以看到，SDK主要需要完成一下工作：

- WebSocket连接建立（包括协议协商）
- 发送DeviceId注册请求（RG）
- 调用注册API
- 启动保持心跳线程，定期保持心跳（H1）
- 调用其他API
- 接收API网关发送的通知
- 调用注销API请求

除了正常流程，SDK还需要处理以下异常流程：

- 断线重连

- 触及流控

2.1 WebSocket连接建立（包括协议协商）

API网关的双向通信是基于WebSocket实现的，客户端和API网关之间的连接建立过程是标准的WebSocket连接建立流程。WebSocket连接建立过程请参考WebSocket协议定义。

一般的语言都会有WebSocket协议客户端开源实现，比如Android用的就是OkHttp的WebSocket开源实现。

2.2 发送DeviceId注册请求（RG）

客户端在WebSocket连接建立之后，需要马上向API网关发送一条注册请求，请求中需要携带本机DeviceId；

API网关在收到注册信令后，会在接入层注册一条通信长连接，并且使用DeviceId标识这条长连接。API网关会给客户端返回注册成功的应答。

具体信令请参见1.2.1中的信令描述。

2.3 调用注册API

客户端在发送完注册信令后，需要马上发送一个注册API的请求。注册API需要在API网关提前定义好，具体的定义方法请参见双向通信使用指南文档的第二章。

API请求的发送格式请参见本文的1.1节。在发送注册API请求的时候，需要注意以下两点：

1. 在API网关对RG信令正确应答后再发送；
2. 需要增加一个标识注册API的头：x-ca-websocket_api_type:REGISTER

2.4 启动保持心跳线程，定期保持心跳（H1）

客户端需要每25秒给API网关发送一个心跳请求，API网关在接收到心跳请求后会更新用户的在线时间并且给出一个心跳处理成功的应答。

具体信令请参见1.2.2中的信令描述。

2.5 调用其他API

客户端在Websocket长连接建立好之后，随时可以给API网关发送普通的API请求，发送格式请参见本文的1.1节。

2.6 接收API网关发送的通知

客户端在注册API发送成功，并且收到API网关的200应答后，就可以正式接收API网关发送过来的下行通知了，下行通知的格式非常简单，请参见本文的1.2.3。

2.7 调用注销API请求

客户端在用户退出登录之前，需要发送一个注销API的请求。注册API需要在API网关提前定义好，具体的定义方法请参见双向通信使用指南文档的第二章。

API请求的发送格式请参见本文的1.1节。在发送注册API请求的时候，需要注意一点： 需要增加一个标识注册API的头：x-ca-websocket_api_type:UNREGISTER。

2.8 处理API网关发送过来的断线重连信令

每条长连接都会存在一个生命周期，API网关的长连接生命周期在处理完2000个API请求后结束。API网关会在处理完1500个请求的时候，给客户端发送一条要求客户端主动断线重连的信令（CR），具体信令格式参见本文1.2.4。API网关会在长连接的请求数积累到2000个请求的时候，删除这条长连接。

客户端在收到断线重连这两种信令后，需要暂时停止接收上层API发送请求，并且在发送完已经接收到的API请求后，重新连接API网关，重新发送注册信令和注册API，完成断线重连的过程。建议在第一次发送注册API的时候，将注册API的请求对象缓存在本地，每次断线重连的时候直接使用即可。

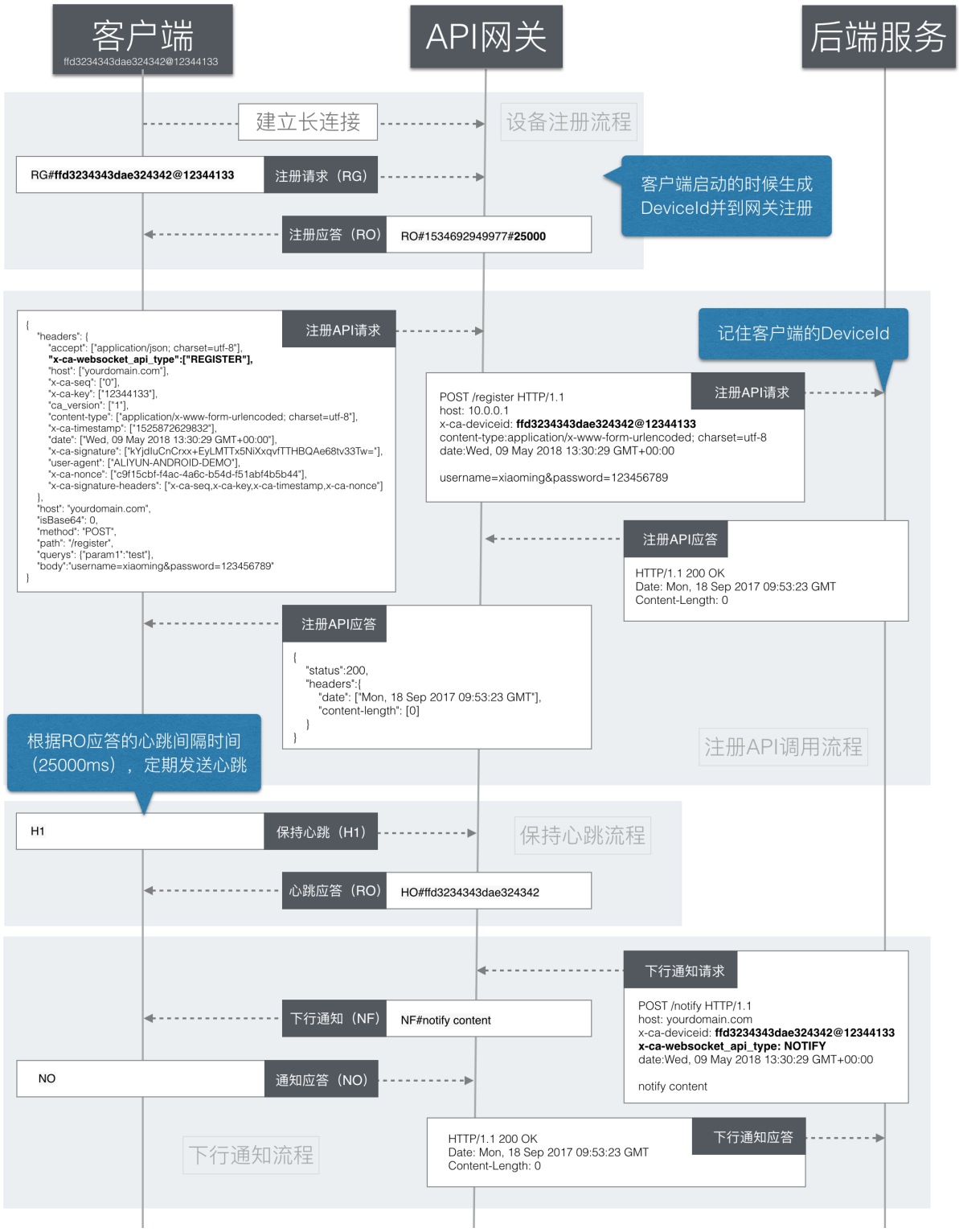
2.9 处理API网关发送过来的流控限制信令

API网关有流控限制，客户端的请求QPS如果超过流控限制会触发API网关的流控，客户端会收到API网关发送的触发流控阈值的信令（OS），具体信令格式请参见本文1.2.4节。这个时候客户端应该控制报文的发送速度。如果客户端不理会此信令，并且让QPS持续增长，API网关会删除这条长连接。

3. 总结

3.1 流程总结

画一张图总结一下支持API网关双向通信的SDK需要做的事情：



注：普通API的请求应答报文格式和注册API的请求应答报文格式是一样的，故没有示例。

3.2 代码参考

目前API网关的Android和Objective-C已经具备双向通信能力，并且严格按照前两节规范实现的。下面我们使用Android的代码来逐步过第二章中提到相关流程的实现。

3.2.1 WebSocket连接建立（包括协议协商）

```
//使用绑定的域名加8080端口作为连接地址
String websocketUrl = "ws:" + yourdomain + ":8080";

//新建一个client对象
OkHttpClient client = new OkHttpClient.Builder()
                .readTimeout(params.getReadTimeout(), TimeUnit.MILLISECONDS)
                .writeTimeout(params.getWriteTimeout(), TimeUnit.MILLISECONDS)
                .connectTimeout(params.getConnectionTimeout(), TimeUnit.MILLISECONDS)
                .build();

//建立WebSocket连接需要的HttpRequest请求
Request connectRequest = new Request.Builder().url(websocketUrl).build();

//连接建立好后的事件监听类
webSocketListener = new WebSocketListener() {
        ......
}

//建立长连接
client.newWebSocket(connectRequest, webSocketListener);
```

3.2.2 发送DeviceId注册请求/心跳请求

```
        String deviceId = generateDeviceId();
        webSocketListener = new WebSocketListener() {
                @Override
                //连接连接好后，回调本方法
                public void onOpen(WebSocket webSocket, Response response) {
                                //发送注册信令
                    String registerCommand = SdkConstant.CLOUDAPI_COMMAND_REGISTER_REQUEST
+ "#" + deviceId;
                    webSocketRef.getObj().send(registerCommand);
                }

                @Override
                //收到API网关的数据后，调用本方法
                public void onMessage(WebSocket webSocket, String text) {
                    if(null  == text || "".equalsIgnoreCase(text)) {
                        return;
                    }
                    //注册成功
                    else if(text.length() > 2 && text.startsWith(SdkConstant.CLOUDAPI_COMMA
ND_REGISTER_SUCCESS_RESPONSE)){
                        //解析API网关注册成功应答
                        String responseObject[] = text.split("#");
                        connectionCredential = responseObject[1];
                        //从API网关应答中获取心跳时间间隔
                        heartBeatInterval = Integer.parseInt(responseObject[2]);

                        }

                                                //启动心跳线程，发送心跳信令
                        if (null != heartBeatManager) {
```

```
                            heartBeatManager.stop();
                        }
                        heartBeatManager = new HeartBeatManager(instance, heartBeatInterval
);
                        heartbeatThread = new Thread(heartBeatManager);
                        heartbeatThread.start();
                        return;
                    }
                    //注册失败
                    else if(text.length() > 2 && text.startsWith(SdkConstant.CLOUDAPI_COMMA
ND_REGISTER_FAIL_REQUEST)){

                        String responseObject[] = text.split("#");
                        errorMessage.setObj(responseObject[1]);
                                                //停止发送心跳
                        if (null != heartBeatManager) {
                            heartBeatManager.stop();
                        }
                        return;
                }
            };
        }

    private String generateDeviceId(){
        return UUID.randomUUID().toString().replace("-" , "").substring(0 , 8);
    }
```

3.2.3 发送API请求：注册API、普通API、注销API

```
protected void sendAsyncRequest(final ApiRequest apiRequest , final ApiCallback apiCallback
){
        checkIsInit();
                //判断连接是否建立
        synchronized (connectionLock) {
            if (null != connectLatch.getObj() && connectLatch.getObj().getCount() == 1) {
                try {
                    connectLatch.getObj().await(10, TimeUnit.SECONDS);
                } catch (InterruptedException ex) {
                    throw new SdkException("WebSocket connect server failed ", ex);
                } finally {
                    connectLatch.setObj(null);
                }
            }

            if (status == WebSocketConnectStatus.LOST_CONNECTION) {
                apiCallback.onFailure(apiRequest, new SdkException("WebSocket conection los
t , connecting"));
                return;
            }

                        //针对注册、注销类API，做特殊处理
            if (WebSocketApiType.COMMON != apiRequest.getWebSocketApiType()) {
                if(!preSendWebsocketCommandApi(apiRequest , apiCallback)) {
```

```
                    return;
                }
            }

            Integer seqNumber = seq.getAndIncrement();
            apiRequest.addHeader(SdkConstant.CLOUDAPI_X_CA_SEQ, seqNumber.toString());
            callbackManager.add(seqNumber, new ApiContext(apiCallback, apiRequest));

            //生成需要发送的字符串
            String request = buildRequest(apiRequest);
            webSocketRef.getObj().send(request);
        }

    }

    private boolean preSendWebsocketCommandApi(final ApiRequest apiRequest , final ApiCallb
ack apiCallback){
        //注册类API，需要判断注册信令是否完成
        if(WebSocketApiType.REGISTER == apiRequest.getWebSocketApiType()) {
            try {
                if (null != registerLatch.getObj() && !registerLatch.getObj().await(10, Tim
eUnit.SECONDS)) {
                    Thread.sleep(5000);
                    close();
                    apiCallback.onFailure(apiRequest, new SdkException("WebSocket conection
lost , connecting"));
                    return false;
                }
            } catch (InterruptedException ex) {
                throw new SdkException("WebSocket register failed ", ex);
            } finally {
                registerLatch.setObj(null);
            }

            if (!registerCommandSuccess.getObj()) {
                apiCallback.onFailure(null, new SdkException("Register Comand return error
:" + errorMessage.getObj()));
                return false;
            }

                        //记录注册API，用于断线重连
            lastRegisterReqeust = apiRequest.duplicate();
            lastRegisterCallback = apiCallback;

        }
                //增加注册、注销API的标识头
        apiRequest.addHeader(SdkConstant.CLOUDAPI_X_CA_WEBSOCKET_API_TYPE, apiRequest.getWe
bSocketApiType().toString());


        return true;
    }


     private String buildRequest(ApiRequest apiRequest){
```

```
    private String buildRequest(ApiRequest apiRequest){
        apiRequest.setHost(host);
        apiRequest.setScheme(scheme);
        ApiRequestMaker.make(apiRequest , appKey , appSecret);


        WebSocketApiRequest webSocketApiRequest = new WebSocketApiRequest();
        webSocketApiRequest.setHost(host);
        webSocketApiRequest.setPath(apiRequest.getPath());
        webSocketApiRequest.setMethod(apiRequest.getMethod().getValue());
        webSocketApiRequest.setQuerys(apiRequest.getQuerys());
        webSocketApiRequest.setHeaders(apiRequest.getHeaders());
        webSocketApiRequest.setIsBase64(apiRequest.isBase64BodyViaWebsocket() == true ? 1 :
0);
        MediaType bodyType = MediaType.parse(apiRequest.getFirstHeaderValue(HttpConstant.CL
OUDAPI_HTTP_HEADER_CONTENT_TYPE));

        if(null != apiRequest.getFormParams() && apiRequest.getFormParams().size() > 0){
            webSocketApiRequest.setBody(HttpCommonUtil.buildParamString(apiRequest.getFormP
arams()));
        }else if(null != apiRequest.getBody()){
            webSocketApiRequest.setBody(new String(apiRequest.getBody() , bodyType.charset(
SdkConstant.CLOUDAPI_ENCODING)));
        }

        if(apiRequest.isBase64BodyViaWebsocket()){
            webSocketApiRequest.setBody(new String(Base64.encode(apiRequest.getBody() , Bas
e64.DEFAULT) , bodyType.charset(SdkConstant.CLOUDAPI_ENCODING)));
        }

        return JSON.toJSONString(webSocketApiRequest);
}
```

3.2.4 接收API网关发送的通知

```
        ApiWebSocketListner apiWebSocketListner = params.getApiWebSocketListner();
        webSocketListener = new WebSocketListener() {
                ......

                @Override
                //收到API网关的数据后，调用本方法
                public void onMessage(WebSocket webSocket, String text) {
                     String message  = text.substring(3);
                     apiWebSocketListner.onNotify(message);
                     if(status == WebSocketConnectStatus.CONNECTED && webSocketRef.getObj()
!= null){
                            webSocketRef.getObj().send(SdkConstant.CLOUDAPI_COMMAND_NOTIFY_
RESPONSE);
                     }
                     return ;
                     }

     ......
        }
```

具体代码请下载Android SDK后，在src文件夹中可以找到。