# API 网关

## 用户指南（开放 API）

# 用户指南（开放 API）

# 概述

API 网关（API Gateway），提供高性能、高可用的API托管服务，帮助您对外开放您部署在ECS、容器服务等阿里云产品上的应用，为您提供完整的API发布、管理、维护生命周期管理。您只需简单操作，即可快速、低成本、低风险的开放数据或服务。

在 API 网关您可以：

**管理您的 API**

您可以对API的整个生命周期进行管理，包括API的创建、测试、发布、下线、版本切换等操作。

**便捷转换数据**

支持自定义映射规则，您可以配置映射将调用请求转换成后端需要的格式。

**预设请求校验**

您可以预先设置参数类型、参数值（范围、枚举、正则、Json Schema）校验，由网关帮助您过滤掉非法请求，减少您的后端对非法请求的处理成本。

**灵活控制流量**

您可以对API、用户、应用设置按分钟、小时、天的调用量控制。您还可以设置特例用户或者应用，对某个用户或应用单独配置流量控制。

**轻松安全防护**

支持 Appkey 认证，HMAC（SHA-1,SHA-256）算法签名。

支持 SSL/TSL 加密，并借助阿里云盾防病毒、防攻击。

**全面监控与报警**

为您提供可视化 API 实时监控，包括：调用量、调用方式、响应时间、错误率，并支持历史情况查询，以便统筹分析。您还可以配置预警方式（短信、Email）,订阅预警信息，以便实时掌握 API 运行情况。

**降低开放成本**

为您自动生成 API 文档和 SDK（服务端、移动端），降低 API 开放成本。

# API 创建

创建 API 即录入 API 的定义，支持“入参映射”和“入参透传”两种请求模式。

## 创建API

创建 API 需要录入 API 的基本信息、服务信息、请求信息、返回信息。

此外，网关支持您配置常量参数、系统参数，这些参数对您的用户不可见，但是网关可以在中转时将常量参数、系统参数加入请求中，传递至后端服务，满足您后端的一些业务需求。比如您需要网关每次向您发送请求都带有一个 keyword **aligateway**，您就可以把 **aligateway** 配置为常量参数，并指定接收的位置。

### 第一部分：定义请求的基本信息

**API 基本信息包括 API 分组、 API 名称、安全认证方式、 API 类型、描述。**

- API 创建时需要选择分组。分组是 API 的管理单元，创建 API 之前您需要先创建分组（ API 分组的详细说明见 API 开放），选择分组即选择 Region。
- API 名称：API名称标识，需要手工录入。

安全认证方式：是 API 请求的认证方式，目前支持 **阿里云APP**、**OpenID Connect** 和 **无认证** 三种认证方式。

i. **阿里云APP**：认证方式即要求请求者调用该 API 时能够通过对APP的身份认证。
ii. **OpenID Connect**：是一套基于 OAuth 2.0 协议的轻量级规范，提供通过RESTful APIs 进行身份交互的框架。可以使用OpenID Connect和您的自有账号系统无缝对接，详细介绍请参照 OpenID Connect。
iii. **无认证**：认证方式即任何人知晓该 API 的请求定义后，均可发起请求，网关不对其做身份验证，均会转发至您后端服务。（强烈不建议使用此模式）

API 类型分为公开和私有两种。

1.**私有** 类型的 API ，当所在分组上架云市场时，默认不包括该类型的 API 。如果有用户想要调用您的 API ，您需要主动操作授权，否则用户无渠道获取 API 信息。
2.**公开** 类型的 API ，所有用户均有机会在 **发现 API** 页面看到 API 的部分信息。**公开** 类型的 API 都会跟 API 分组上架到云市场，供用户购买和调用。

- 描述： API 功能描述，会生成在 API 文档中。

## 第二部分：定义 API 请求

这部分是定义用户如何请求您的 API ，包括协议、Method、Path、入参的定义。

- 协议。 API 调用支持 HTTP/HTTPS 协议。

  Method。支持标准的 HTTP Method，可选择 PUT、GET、POST、DELETE、HEAD。

  **Path**。Path 指相对于服务 host， API 的请求路径。请求 Path 可以与后端服务实际 Path 不同，您可以随意撰写合法的有明确语义的 Path 给用户使用。您可以在请求 Path 中配置动态参数，即要求用户在 Path 中传入参数，同时您的后端又可以不在 Path 中接收，可以映射为在 Query、Header 等位置接收。在 **开放 API 接入 API 网关** 中，有详细的举例说明，更有清晰的截图展示。

- **入参请求模式**。网关对入参的处理模式，支持“入参映射”和“入参透传”两种模式。
  i. **入参映射**：既 API 网关在接收到您的 API 请求时，通过映射关系将请求转换成您后端需要的格式。使用入参映射模式：
     a. 定义方式：此类API需要在定义时添加前后端参数映射关系。
     b. 使用场景
        - 同一接口，在 API 网关定义不同的API，以服务差异化的用户；
        - 通过 API 网关规范化陈旧系统接口。
     c. 实现功能
        - 支持您配置前端和后端的全映射，即参数混排。可以让 API 的消费者在 **Query** 传入参数, 后端从 **Header** 里接收等。
          - 参数名称转换
          - 参数位置转换
        - 可以定义参数的校验规则，实现请求参数的预校验，降低后端处理非法请求的压力。
          - 参数长度校验
          - 参数数值大小校验
          - 参数正则校验
          - 参数Json Schame校验
     - 入参透传：即 API 网关在接收到 API 请求后，不对请求进行处理，直接转发到后端服务。此模式下：
       a. 无法实现参数校验
       b. 无法生成详细的 API 调用文档
       c. 自动生成的SDK不包含请求入参
- **入参**。定义您 API 的请求入参，包含参数名、参数位置、类型、是否必填、默认值、示例、描述等信息。**入参透传模式下不需要录入参数**。

- 参数名。展示给用户的参数名称。
- 参数位置。参数在请求中的位置，包含Head、Query、Body、Path（Parameter Path），当您在 Path 中配置了动态参数，存在参数位置为Parameter Path的同名参数。
- 类型。字段的类型，支持：String、Int、Long、Float、Double、Boolean。
- 是否必填。指此参数是否为必填指，当选择为是时，网关会校验用户的请求中是否包含了此参数，若不存在则拒绝用户请求。
- 默认值。当“是否必填”为否时生效，在用户请求中不包含此参数时，网关自动添加默认值给后端服务。若用户传递，则按用户请求传递给后端服务。
- 示例。指参数的填写示例，生成 API 文档、上架 API 市场时的参数示例。
- 描述。参数的用途描述及使用的注意事项，生成API文档、上架API市场时的参数描述。
- **参数校验规则**。每个入参后可点击 **编辑更多** 配置校验规则。如参数值的长度、取值大小、枚举、正则、Json Shcame等等。网关会参照校验规则对请求做初步校验，如果入参不合法，则不会到达您的后端服务，大大的降低了后端服务的压力。

## 第三部分：定义后端服务信息

这部分主要是定义一些参数的前后端映射，具体描述的是您后端真实服务的 API 配置。用户请求到达网关后，网关会根据您的后端配置映射为对应实际后端服务的请求形式，去请求您的后端。包括后端服务地址、后端Path、后端超时时间、参数映射、常量参数、系统参数。

- 后端服务类型。目前支持HTTP/HTTPS、FunctionCompute两种。
  - HTTP/HTTPS。若您的服务为HTTP/HTTPS服务，则选择此项。注意若您是HTTPS服务，后端服务必须有SSL证书。
  - FunctionCompute。为阿里云函数计算，使用方法请参照：以函数计算作为 API 网关后端服务
- VPC 通道。当您的后端服务在 VPC 中时，需要使用此通道，使用方法：专属网络 VPC 环境开放 API
- 后端服务地址。后端服务的 host，可以是一个域名，也可以是 http(s)://host:port 的形式。填写时，必须填写http://、https://
- 后端 Path。Path 是您的 API 服务在您后端服务器上的请求路径，实际请求路径。若您后端 Path 需要接收动态参数，那么需要声明该参数是调用者从哪个位置哪个参数传入的，即声明映射关系。
- 后端超时时间。指 API 请求到达网关后，网关去调 API 后端服务的响应时间。由网关请求后端开始到网关收到后端返回结果。该值不能超过30秒。超过该值网关会放弃请求后端服务，并给用户返回相应的错误信息。
- 参数映射。网关支持参数在前端、后端的全映射，包括名称映射和位置映射。位置映射包括 Path、Header、Query、Body 的混排映射。也就是说，您可以将您的后端服务通过映射完成包装成更规范、更专业的 API 形态。这部分就是在声明前后端 API 映射关系的。在 开放 API 接入 API 网关 中，有详细的举例说明，更有清晰的截图展示。**注意前后端参数名称不能重复**。
- 常量参数。比如您需要网关每次请求您后端时都带有标记 **apigateway**，那么您可以直接将标记配置为常量参数。常量参数对您的用户不可见，请求达到网关后，网关会自动在指定位置加上该参数再去请求您的后端。

  系统参数。指 API 网关的系统参数，这些参数默认不会传递给您，但是如果您需要获取，您可以在 API 里配置接收位置和名称。具体内容如下表：

| 参数名称 | 参数含义 |
| --- | --- |

| | |
|---|---|
| CaClientIp | 发送请求的客户端IP |
| CaDomain | 发送请求的域名 |
| CaRequestHandleTime | 请求时间（格林威治时间） |
| CaAppId | 请求的APP的ID |
| CaRequestId | RequestId |
| CaApiName | API 名称 |
| CaHttpSchema | 用户调用 API 使用的协议，http或者https |
| CaProxy | 代理（AliCloudApiGateway） |

**注意：**您所有录入的参数，包括 Path 中的动态参数、Headers 参数、Query 参数、Body 参数（非二进制）、常量参数、系统参数，参数名称保证全局唯一。即如果您同时在 Headers 和 Query 里各有一个名为 **name** 的参数，是不允许的。

### 第四部分 定义返回结果

您需要录入返回ContentType、返回结果示例、失败返回结果示例和错误码定义，将会生成 API 文档和在 API 市场的展示。

## API调试

API 定义录入完成后，您可以在API调试页面调试API，以确定 API 的可用性。

## 发布

完成以上定义后，您就完成了 API 的创建。下一步您可以测试、发布、开放 API 服务到市场，还可以为 API 绑定签名密钥和流量控制等安全配置。

# API 开放

API 创建完成后，您就可以开放 API 服务了。要开放 API 服务您需要绑定一个在阿里云系统备案成功的独立域名，且该域名要完成 CNAME 解析。而独立域名是绑定在 API 分组上面的，所以在这个部分为您详细说明一下开放 API 服务需要了解的 API 分组和域名。

### 第一部分：API 分组

API 分组是 API 的管理单元。您创建 API 之前，需要先创建分组，然后在某个分组下创建 API。分组包含名称

、描述、区域（Region）、域名几大属性。

- 分组的区域（Region）在分组创建时选定不可更改。创建 API 时，如果选定分组那么 Region 也一同选定，不可更改。
- 每个账号 API 分组个数上限为50个，每个分组 API 个数上限为200个。
- 域名。分组创建时，系统会为分组分配一个二级域名。如果需要开放 API 服务，您需要为分组绑定一个在阿里云系统备案成功的独立域名，且将独立域名 CNAME 到相应的二级域名上。每个分组最多只能绑定5个独立域名。具体请看下文——域名及证书。

## 第二部分：环境管理

关于环境需要理解两个概念，**环境**和**环境变量**。

**环境**是 API 分组上的一个配置，每个分组有若干个环境。 API 录入后，未经发布时，就只是 API 定义。发布到某个环境后才是能够对外提供服务的 API 。

**环境变量**是在环境上用户可创建可管理的一种变量，该配置是固定于环境上的。如在线上环境创建变量，变量名为 **Path**，变量值为 **/stage/release**.

在 API 定义中的 **Path** 位置，写作 **#Path#**，即配置为变量标识，变量名为 **Path**。

那么将该 API 发布到线上环境时，该 API 在线上环境的运行定义，Path 处的 **#Path#**，会取值为 **/stage/release**。

而将该 API 发布到其他环境时，若环境上没有环境变量 **#Path#**，则无法取值，那么 API 就无法调用。

使用环境变量可以解决后端服务需要区分环境的问题，通过不同的环境上配置不同的服务地址和Path，来调用不同的后端服务，同时 API 的定义配置又是一套。使用时需要注意以下几点：

- 在 API 定义中配置了变量标识后，在 API **列表—管理—调试** 页面将无法调试。
- 变量名严格区分大小写。
- 如果在 API 定义中设置了变量，那么一定要在要发布的环境上配置 **变量名**和**变量值**，否则变量无赋值， API 将无法正常调用。

## 第三部分：域名及证书

API 网关通过域名来定位到一个唯一的 API 分组，再通过 **Path+HTTPMethod** 确定唯一的 API 。如果要开放 API 服务，您需要了解 **二级域名** 和 **独立域名**。

- **二级域名**是分组创建时系统分配的，唯一且不可更改。在您还没有独立域名之前，您可以通过访问二级域名来测试调用您的 API 。二级域名仅能用于测试，默认每天只能请求1000次。

  **独立域名**即自定义域名，是您开放 API 服务需要绑定的，用户通过访问您的独立域名来调用您开放的 API 服务。您可以为一个分组绑定多个独立域名，上限为5个。对于独立域名的配置您需要注意以下几点：

独立域名不必须是根域名，可以是二级、三级域名。

独立域名如果尚未备案，则可以在阿里云做 **首次备案**。

独立域名若已在其他系统备案，则需要在阿里云做 **备案接入**。

独立域名需要 CNAME 解析到分组的二级域名上。

满足上述的备案和解析两个要求，域名才能成功绑定。

当您的 API 服务支持 HTTPS 协议时，需要为该域名上传 SSL 证书，在 **分组详情** 页面进行添加即可。SSL 证书上传不支持文件上传，需要填写 **证书名称**、**内容** 和 **私钥**。

## 第四部分：测试、线上、授权

通过上述操作您已经完成 API 的创建和域名绑定，接下来就可以将 API 发布到测试或者线上环境，进行调试和开放了。其中一个重要的环节是授权，授权即授予某个 APP 可以调用某个 API 的权限。

- 当您完成 API 创建之后，您就可以将 API 发布到测试或者线上，并给自己创建的 APP 授权，通过访问二级域名来调用指定环境中的 API，进行测试。
- 成功绑定独立域名之后，您的 API 就可以在市场上架，供客户购买、调用。您还可以不经过购买将 API 授权给合作伙伴的 APP，供其调用。

**至此，您完成 API 服务的开放。在 API 创建到开放的整个过程中，您还可以随时操作 API 的创建、修改、删除、查看、测试、发布、下线、授权、解除授权、发布历史及版本切换等操作。**

# API 管理

API 定义就是指您创建 API 时对 API 的请求结构的各方面定义。您可以在控制台完成 API 定义的查看、编辑、删除、创建、复制。您需要注意以下几点：

- 当您需要编辑某个 API 的定义时，如果该 API 已经发布，对定义的修改不会对线上产生影响，定义修改后需要再次发布才能把修改后的定义同步到线上环境。
- 当您想要删除某个 API，如果该 API 已经发布，则不允许直接删除 API 定义，需要先将 API 下线，然后删除。
- API 网关为您提供了复制定义的功能。您可以从测试环境/线上环境复制线上的定义覆盖当前的最新定义，然后重新点击编辑进行修改。

### API 发布管理

当您完成 API 的创建后，您可以将 API 发布到测试或者线上。也可以将测试或者线上的 API 下线。您需要注意以下几点：

- API 创建完成后，发布到某环境，通过二级域名或者独立域名访问时，需要在请求的Header指定要请求的环境，参见 **请求示例**。
- 当您要发布某个 API 时，如果该 API 在测试或者线上已经有版本在运行，您的此次发布将使测试或者线上的该 API 被覆盖，实时生效。但是历史版本及定义会有记录，您可以快速回滚。
- 您可以将测试或者线上的某个 API 下线，下线之后，与策略、密钥、 APP 的绑定或者授权关系依然存在，再次上线时会自动生效。如果要解除关系，需要专门操作删除。

### API 授权管理

您的 API 如果上架到市场，那么购买者有权利操作给自己的某个 APP 授权。

如果不经过购买行为，您需要在线下跟合作伙伴建立使用关系，那么您需要通过授权来建立 API 和 APP 的权限关系。您将 API 发布到线上环境后，需要给客户的 APP 授权，客户才能用该 APP 进行调用。您有权对此类授权操作建立或者解除某个 API 与某个 APP 的授权关系， API 网关会对权限关系进行验证。操作授权时，您需要注意以下几点：

- 您可以将一个或者多个 API 授权给一个或者多个 APP 。批量操作时，建议不要同时操作多个分组下的 API 。
- 批量操作时，先选择 API 后选择环境。比如一个 API 在测试和线上均有发布，最后选择了测试，就只会将测试下的该 API 授权。
- 您可以通过客户提供给您的AppID或者阿里云邮箱账号来定位 APP 。
- 当您需要解除某个 API 下某个 APP 的授权时，您可以查看 API 的授权列表，在列表页进行解除。

### 历史与版本切换

您可以查看您每个 API 的发布历史记录，包括每次发布的版本号、说明、环境、时间和具体定义内容。

查看历史时，您可以选定某个版本然后操作切换到此版本，该操作会使该版本直接在指定环境中替换之前的版本，实时生效。

# 签名密钥

## 什么是签名密钥

签名密钥是由您创建的一对 Key 和 Secret，相当于您给网关颁发了一个账号密码，网关向您后端服务请求时会将密钥计算后一起传过去，您后端获取相应的字符串做对称计算，就可以对网关做身份验证。使用时您需要了解以下几点：

- 创建密钥时需要选择 **Region**，**Region** 一旦选定不能更改，而且密钥只能被绑定到同一个 **Region** 下

的API上。
- 一个 API 仅能绑定一个密钥，密钥可以被替换和修改，可以与 API 绑定或者解绑。
- 您将密钥绑定到 API 之后，由网关抛向您服务后端的该 API 的请求均会加上签名信息。您需要在后端做对称计算来解析签名信息，从而验证网关的身份。具体 HTTP 加签说明请查看文档——后端签名密钥说明文档

# 密钥泄露 修改替换

当您遇到如下情况：

您的某一个密钥发生了泄露，您可能想要保留该密钥与 API 的绑定关系，但是想要修改密钥的 Key 和 Secret。

当您操作将密钥应用于 API 时，可能该 API 已经绑定了某个密钥，需要替换密钥。

**以上两种情况都建议按照下面的流程来操作：**

1. 先在后端同时支持两个密钥：原来的密钥和即将修改或替换的密钥，确保切换过程中的请求能够通过签名验证，不受修改或替换的影响。
2. 后端配置完备后，完成修改，确定新 Key 和 Secret 生效后再将之前已泄露或废弃的密钥删除。

# 流量控制策略

**流量控制策略和 API 分别是独立管理的，操作两者绑定后，流控策略才会对已绑定的 API 起作用。**

在已有的流量控制策略上，可以额外配置特殊用户和特殊应用（APP），这些特例也是针对当前策略已绑定的API生效。

流量控制策略可以配置对 API、用户、应用三个对象的流控值，流控的单位可以是分钟、小时、天。使用流量控制策略您需要了解以下几点：

流量控制策略可以涵盖下表中的维度：

| | |
|---|---|
| **API 流量限制** | 该策略绑定的API在单位时间内被调用的次数不能超过设定值，单位时间可选分钟、小时、天，如5000次/分钟。 |
| **APP 流量限制** | 每个APP对该策略绑定的任何一个API在单位时间内的调用次数不能超过设定值。如50000次/小时。 |
| **用户流量限制** | 每个阿里云账号对该策略绑定的任何一个 API 在单位时间内的调用次数不能超过设定值。一个阿 |

| | |
|---|---|
| | 里云账号可能有多个 APP，所以对阿里云账号的流量限制就是对该账号下所有 APP 的流量总和的限制。如 50 万次/天。 |

在一个流控策略里面，这三个值可以同时设置。请注意，用户流量限制应不大于 API 流量限制，APP 流量限制应不大于用户流量限制。即 APP 流量限制 <= 用户流量限制 <= API 流量限制。

此外，您可以在流控策略下添加特殊应用（APP）和特殊用户。对于特例，流控策略基础的 **API 流量限制** 依然有效，您需要额外设定一个阈值作为该 APP 或者该用户的流量限制值，该值不能超过策略的 **API流量限制** 值，同时流控策略基础的 **APP流量限制** 和 **用户流量限制** 对该 APP 或用户失效。

与签名密钥相似，当您创建流量控制策略时，需要选择 **Region**，**Region** 一旦选定不可更改，且仅能被应用于同一个 Region 下的 API。

由于 API 网关限制，当您设置 **API 流量限制** 值时，考虑每个 API 分组的默认流控上限是 500QPS（该值可以通过提交工单申请提高）。

绑定 API。您可以将策略绑定于多个 API，流控策略的限制值和特例将对该策略绑定的每一个 API 单独生效。当您绑定 API 时，如果该 API 已经与某个策略绑定，您的此次操作将替换之前的策略，实时生效。

特殊对象。如果您想要添加特殊应用或者特殊用户，您需要获得应用 ID 即 AppID 或者用户的阿里云邮箱账号。

在 API 网关控制台，您可以完成对流量控制策略的创建、修改、删除、查看等基本操作。还有流控策略与 API 的绑定解绑，流量控制策略特殊对象的添加删除等操作。

# 监控预警

- API网关为您提供可视化的实时监控和预警。您可以获得您开放的API被调用数据，包括调用量、流量、响应时间、错误分布等多个维度、多种时间单位的数据统计结果。同时支持历史数据查询，以便您统筹分析。
- 后续我们会陆续支持报表输出，给您提供最周到的数据支持。
- 您还可以配置预警，以便实时掌握API运行情况。

# API 网关使用限制

| 限制项 | 限制描述 |
| --- | --- |
| 使用API网关服务的用户限制 | 用户需实名认证 |
| 用户创建API分组数量限制 | 每个账号下，API分组的个数上限是50个 |
| 用户创建API数量限制 | 每个API分组下，最多可以创建200个API。即每个账号最多可创建50*200 = 10000个API。 |
| 用户的API分组绑定独立域名个数限制 | 每个分组最多绑定5个独立域名 |
| API的TPS限制 | 每个API分组接受访问的TPS上限为500。如需要将该限制调高，请提工单申请。对高TPS配置，网关会收取一定费用。 |
| API分组官方二级域名限制 | API分组创建成功后，API网关会为分组颁发二级域名。该域名用于测试该分组下的API，访问次数限制为1000次/天。请不要直接使用该二级域名对外提供API服务。 |
| 参数大小限制 | Body位置的参数（包括Form和非Form形式）总体不能超过 2 Mb，其他位置的参数（包括Header和Query）总体不能超过 128 Kb。 |

# 后端签名密钥说明文档

## 概述

- API 网关提供后端 HTTP 服务签名验证功能，创建签名密钥并将签名密钥绑定到 API 即可开启后端签名（请妥善保管此密钥，API 网关会对密钥进行加密存储来保障密钥的安全性）。
- 开启后端签名后 API 网关到后端HTTP服务的请求将会添加签名信息，后端 HTTP 服务读取 API 网关的签名字符串，然后对收到的请求进行本地签名计算，比对网关与本地签名结果是否一致。
- 所有您定义的参数都会参与签名，包括您录入的业务参数、您定义的常量系统参数和 API 网关系统参数（如 CaClientIp 等）。
- 后端对 API 网关的签名字符串校验后，如果校验失败，建议返回 errorcode = 403；errormessage = “InvalidSignature”。
- 若您的后端服务为VPC环境，且通过内网对接（专属网络VPC环境开放API），您无需使用后端签名，通道自身是安全的。

# 读取 API 网关签名方法

网关计算的签名保存在 Request 的 Header 中，Header 名称：X-Ca-Proxy-Signature

# 后端 HTTP 服务加签方法

签名计算的详细 demo（JAVA）请参照链接：https://github.com/aliyun/api-gateway-demo-sign-backend-java。

签名计算方法步骤如下：

## 组织参与加签的数据

```
String stringToSign=
HTTPMethod + "\n" + //Method全大写
Content-MD5 + "\n" + //Content-MD5 需要判断是否为空，如果为空则跳过，但是为空也需要添加换行符 "\n"
Headers + //Headers 如果为空不需要添加"\n"，不为空的Headers中包含了"\n"，详见下面组织Headers的描述
Url
```

## 计算签名

```
Mac hmacSha256 = Mac.getInstance("HmacSHA256");
byte[] keyBytes = secret.getBytes("UTF-8");
hmacSha256.init(new SecretKeySpec(keyBytes, 0, keyBytes.length, "HmacSHA256"));
String sign = new String(Base64.encodeBase64(Sha256.doFinal(stringToSign.getBytes("UTF-8")),"UTF-8"));
```

secret 为绑定到 API 上的签名密钥

# 补充说明

### Content-MD5

Content-MD5 是指 Body 的 MD5 值，只有 HttpMethod 为 PUT 或者 POST 且 Body 为非 Form 表单时才计算 MD5，计算方式为：

String content-MD5 = Base64.encodeBase64(MD5(bodyStream.getbytes（“UTF-8”)));

### Headers

Headers 指所有参与签名计算的 Header的Key、Value。参与签名计算的 Header 的 Key 从 Request Header 中读取，Key为：” X-Ca-Proxy-Signature-Headers” ，多个 Key 用英文逗号分割。

**Headers 组织方法**：

先对所有参与签名计算的 Header 的 Key 按照字典排序，然后将 Header 的 Key 转换成小写后按照如下方式拼接：

String headers = HeaderKey1.toLowerCase() + ":" + HeaderValue1 + "\n" + HeaderKey2.toLowerCase() + ":" + HeaderValue2 + "\n" + ... HeaderKeyN.toLowerCase() + ":" + HeaderValueN + "\n"

**Url**

Url指 Path+Query+Body 中 Form 参数，组织方法：如果有 Query 或 Form 参数则加 **?**，然后对 Query+Form 参数按照字典对 Key 进行排序后按照如下方法拼接，如果没有 Query 或 Form 参数，则 Url = Path

```
String url =
Path +
"?" +
Key1 + "=" + Value1 +
"&" + Key2 + "=" + Value2 +
...
"&" + KeyN + "=" + ValueN
注意:这里 Query 或 Form 参数的 Value 可能有多个，多个的时候只取第一个 Value 参与签名计算
```

**调试模式**

为了方便后端签名接入与调试，可以开启 Debug 模式进行调试，具体方法如下：

请求API网关的 Header 中添加 X-Ca-Request-Mode = debug

后端服务在 Header 中读取 X-Ca-Proxy-Signature-String-To-Sign 即可，因为 HTTP Header 中值不允许有换行符，因此换行符被替换成了 "|"。

**注意：X-Ca-Proxy-Signature-String-To-Sign 不参与后端签名计算。**

**时间戳校验**

如果后端需要对请求进行时间戳校验，可以在 API 定义中选择系统参数 "CaRequestHandleTime"，值为网关收到请求的格林威治时间。

# OpenID Connect 认证

OpenID Connect 是一套基于 OAuth 2.0 协议的轻量认证级规范，提供通过 API 进行身份交互的框架。较 OAuth 而言， OpenID Connect 方式除了认证请求之外，还标明请求的用户身份。

API 网关依据 OpenID Connect 的标准，提供两种认证方式：

**OpenID Connect**

标准的OpenID Connect模式，调用API时先通过用户名密码获取Token，之后的每次API请求都通过Token来验证。

**OpenID Connect & 阿里云APP**

在OpenID Connect基础上增加Appkey认证， 会同时验证API请求中的 Appkey和Token（Token 由 API 提供者的系统颁发，网关颁发 Appkey）。

两种OpenID Connect认证区别

OpenID Connect & 阿里云APP需要认证APPkey，OpenID Connect则不需要。无论哪种模式，都推荐具有登陆态的APP、服务端或者前端程序来使用。

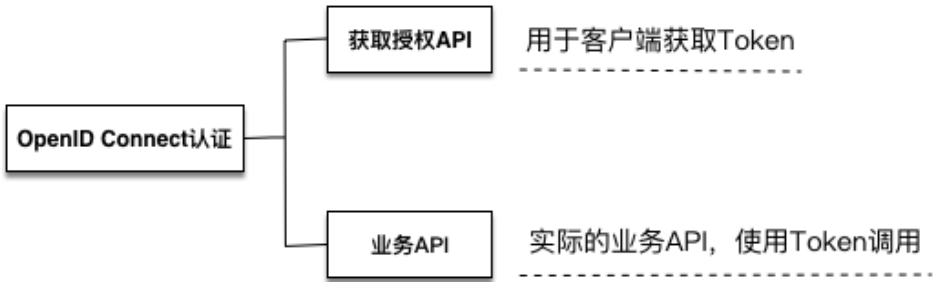
使用这两种认证方式建议：配置好流量控制，避免恶意用户进行暴力破解用户名/密码。

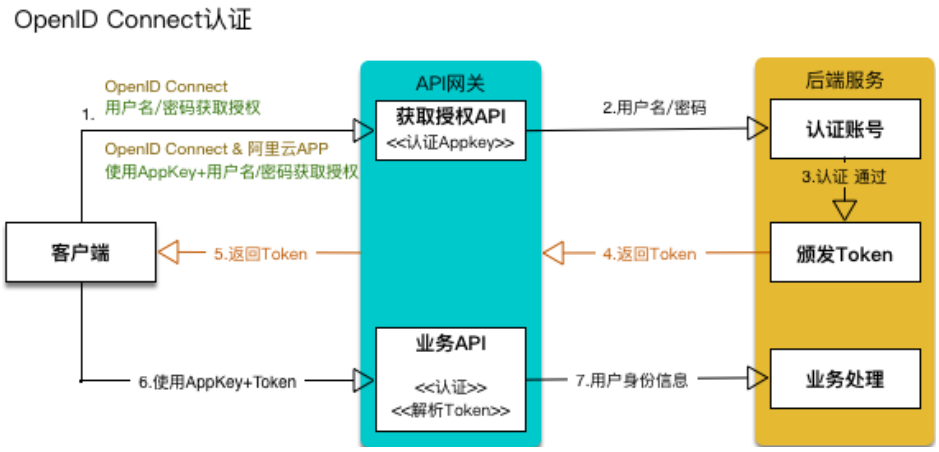
**使用标准OpenID Connect将无法使用的功能**

- APP鉴权，无法使用。
- APP级流量控制，无法配置/无法生效。
- 阿里云用户级流量控制，无法配置/无法生效。

# 实现原理

使用 OpenID Connect认证，需要您在 API 网关配置**获取授权 API** 和 **业务 API** 两种类型的API 。






- 获取授权API：用于您的客户端获取Token 。配置这个API时，您需要告知API网关您Token对应的Key和解析Token使用的公钥。
- 业务类接口，是您实际的业务接口，比如获取用户息、进行某个操作等。配置这类API时，你需要告知API网关你请求中表示Token的参数名称。当客户端调用这类API的请求到达API网关后，API网关自动验证这个请求的Appkey和Token是否合法。

# 认证方式

**客户端调用 “获取授权 API” 获取Token的流程**

i. 客户端使用认证信息来获取Token，
   i. OpenID Connect：需要使用用户的**“用户名/密码”**调用**“获取授权” API** 获取Token。。
   ii. OpenID Connect & 阿里云APP： **“Appkey 签名” + “用户名/密码”** 调用 **“获取授权” API** 获取Token。
   iii. 调用API请参照：**调用API**

API 网关收到请求后，OpenID Connect & 阿里云APP模式（OpenID Connect不需要）认证您的 **Appkey**后，调用后端服务的账号系统认证您传递的 **“用户名/密码”** 。

后端服务认证认证用户名/密码，通过后返回 **Token** 给您，您可凭 **Token** 来调用 **“业务API”** 。

**客户端调用业务类 API ，来实现业务功能**

客户端使用 **“获取授权 API”** 得到的 **Token** 和 **签名后的 Appkey** 来调用 **“业务API”** 。(调用API请参照：**调用API**)

API 网关认证、解析 **Token** 的内容，并将 **Token** 中包含的用户信息传递给后端。

在此阶段的操作中，API 的提供者需要事先进行如下操作：

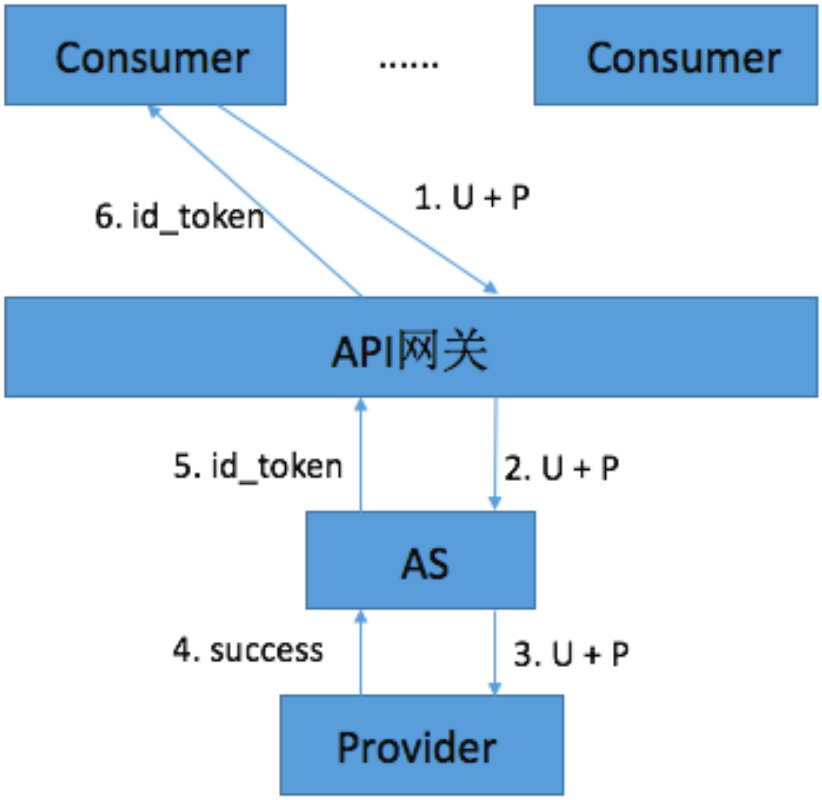
a. 开放账号系统，允许 API 网关对请求中 **用户名/密码** 进行验证，并依据网关提供的加密方式，颁发 Token。详细内容请参照下文 **如何实现 AS 模块**。
b. 在 API 网关定义 API。详细内容参照下文 **在 API 网关配置 API**。
**注意**：**用户名/密码** 是极为敏感的信息，在网络中明文传输存在风险，建议在传输前对用户名密码再次加密，并使用 HTTPS 协议传输。

# 实现方案简介

实现方案分为两个重要的部分：

## 1. Authorization server（AS）：认证服务器，负责生成 id_Token 并管理公钥私钥对。

**这一步需要您自行实现**。实现方法，请参照下文 **在 API 网关配置 API**。




**参考上图，流程简述如下：**

1. Consumer（调用者）向API网关发送获取 id_token 认证请求，比如：通过用户名和密码（U+P）的方式。
2. API 网关透传该请求到 AS。
3. AS 向 Provider（服务提供方）发送认证用户信息请求。

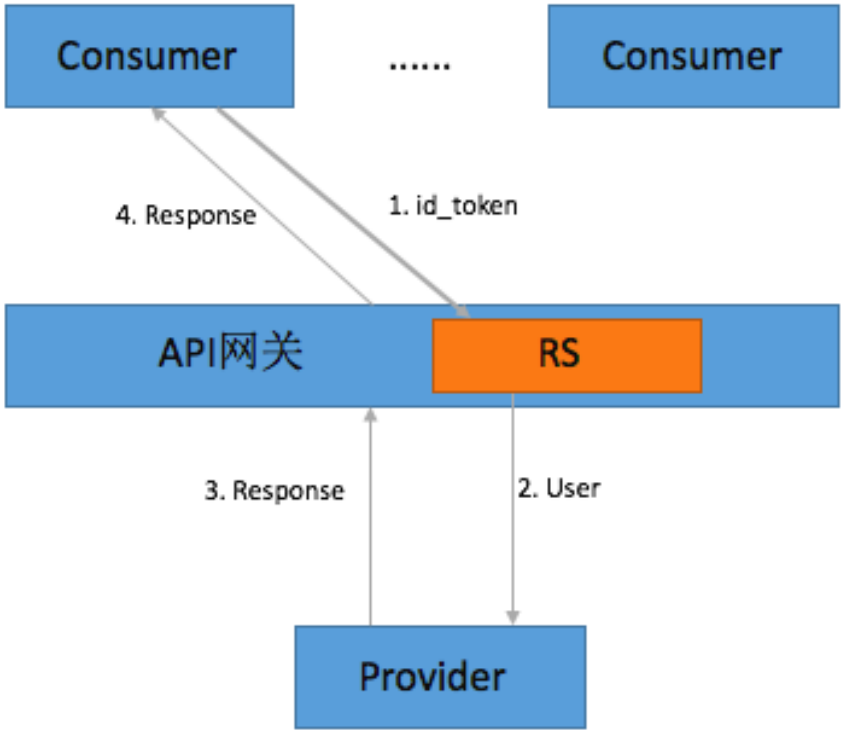
4. Provider 响应认证结果，若失败则直接响应错误信息。
5. 认证结果成功，AS 生成 id_token，id_token 中包含了 User 信息（可扩展，也可包含其他必要信息）。

   API 网关将 AS 返回的 id_token 响应给 Consumer。

   **说明**：AS 不用必须是单独部署的应用，完全可以集成在 Provider 中，在整个体系中担任 id_token 生产者角色，所生成的 id_token 必须符合 OIDC（1.0版本）协议中的 规范。

## 2. Resource server（RS）：资源服务器，负责校验 id_token，并解析出相应的信息。

**此部分由网关来完成。**因为 API 网关目前已经集成了 RS 功能，服务提供方只需要按照相应的加密规则生成 id_token 即可。




**参考上图，流程简述如下：**

1. Consumer 用带有 id_token 的参数去请求 API 网关。
2. API 网关会保存校验所使用的公钥，验证并解析 id_token 获取其中的 User 信息传给 Provider，若验证失败则直接返回错误信息。
3. Provider 处理请求并返回结果给 API 网关。
4. API 网关透传 Provider 响应的结果给 Consumer。

**说明**：RS 在整个体系中担任 id_token 消费者角色，只有 id_token 校验通过，才能将请求转发给 Provider。

# 如何实现 AS 模块

## AS 中使用 OIDC 生成 id_token 的说明

- id_token，也叫 ID Token，是在 OIDC 协议中定义的一种令牌，详细内容参见 OpenID Connect Core 1.0。
- id_token 生成需要 KeyPair, keyId 与 Claims (有关Claims更多信息请访问 ID_Token）。

### KeyId 说明

KeyId 必须保证唯一, 比如使用 UUID 生成的长度至少32位的随机字符串, 可以全为数字或数字+字母。
参考示例(JAVA)

```
String keyId = UUID.randomUUID().toString().replaceAll("-", "");
```

或

```
String keyId = String.valueOf(UUID.randomUUID().getMostSignificantBits()) +
String.valueOf(UUID.randomUUID().getMostSignificantBits());
```

### KeyPair 说明

KeyPair 是一个基于 PKI 体系的非对称算法的公私钥组合, 每一对包括公钥(publicKey)与私钥(privateKey); 公钥放置在 RS 中,在校验(verify)时使用, 私钥放置在 AS 中,在生成 id_token 时做数字签名使用;
KeyPair 使用 RSA SHA256 加密算法, 为保证足够安全其加密的位数为2048;
AS 中使用的 KeyPair 均为 JSON 格式的数据,一个示例如下:
**publicKey:**

```
{"kty":"RSA","kid":"671741829679797099139504717892261817 21","alg":"ES256","n":"oH5WunqaqIopfOFBz9RfBVVIIcmk0WDJagAcROKFiLJScQ8N\_nrexgbCMlu-dSCUWq7XMnp1ZSqw-XBS2-XEy4W4l2Q7rx3qDWY0cP8pY83hqxTZ6-8GErJm\_0yOzR4WO4plIVVWt96-mxn3ZgK8kmaeotkS0zS0pYMb4EEOxFFnGFqjCThuO2pimF0imxiEWw5WCdREz1v8RW72WdEfLpTLJEOpP1FsFyG3OIDbTYOqowD1YQEf5Nk2TqN\_7pYrGRKsK3BPpw4s9aXHbGrpwsCRwYbKYbmeJst8MQ4AgcorE3NPmp-E6RxA5jLQ4axXrwC0T458LIVhypWhDqejUw","e":"AQAB"}
```

**privateKey:**

```
{"kty":"RSA","kid":"671741829679797099139504717892261817 21","alg":"ES256","n":"oH5WunqaqIopfOFBz9RfBVVIIcmk0WDJagAcROKFiLJScQ8N\_nrexgbCMlu-dSCUWq7XMnp1ZSqw-XBS2-XEy4W4l2Q7rx3qDWY0cP8pY83hqxTZ6-8GErJm\_0yOzR4WO4plIVVWt96-mxn3ZgK8kmaeotkS0zS0pYMb4EEOxFFnGFqjCThuO2pimF0imxiEWw5WCdREz1v8RW72WdEfLpTLJEOpP1FsFyG3OIDbTYOqowD1YQEf5Nk2TqN\_7pYrGRKsK3BPpw4s9aXHbGrpwsCRwYbKYbmeJst8MQ4AgcorE3NPmp-
```

```
E6RxA5jLQ4axXrwC0T458LIVhypWhDqejUw","e":"AQAB","d":"aQsHnLnOK-1xxghw2KP5JTZyJZsiwt-
ENFqqJfPUzmlYSCNAV4T39chKpkch2utd7hRtSN6Zo4NTnY8EzGQQb9yvunaiEbWUkPyJ6kM3RdlkkGLvVtp0sRwPCZ2
EAYBlsMad9jkyrtmdC0rtf9jerzt3LMLC7XWbnpC3WAl8rsRDR1CGs\_-
u4sfZfttsaUbJDD9hD0q4NfLDCVOZoQ\_8wkZxyWDAQGCe6GcCbu6N81fTp2CSVbiBj7DST\_4x2NYUA2KG8vyZYcwvi
NTxQzk4iPfdN2YQz\_9aMTZmmhVUGlmTvAjE5ebBqcqKAS0NfhOQHg2uR46eBKBy\_OyVOLohsQ","p":"8Tdo3DCs-
0t9JMtM0lYqPRP4wYJs37Rv6S-ygRui2MI\_hadTY9I2A199JMYw7Fjke\_wa3gqJLa98pbybdLWkrOxXbKEkwE4uc4-
fuNjLbUTC5tqdM5-
nXmpL887uREVYnk8FUzvWeXYTCNCb7OLw5l8yPJ1tR8aNcd0fJNDKh98","q":"qlRrGSTsZzBkDgDi1xlCoYvoM76cbmx
rCUK-
mc\_kBRHfMjlHosxFUnAbxqIBE4eAJEKVfIJLQrHFvIDjQb3kM9ylmwMCu9f8u9DHrT8J7LSDlLqDaXuiM2oiKtW3bAaBP
uiR7sVMFcuB5baCebHU487YymJCBTfeCZtFdi6c4w0","dp":"gVCROKonsjiQCG-s6X4j-saAL016jJsw-
7QEYE6uiMHqR\_6iJ\_uD1V8Vuec-
RxaItyc6SBsh24oeqsNoG7Ndaw7w912UVDwVjwJKQFCJDjU0v4oniItosKcPvM8M0TDUB1qZojuMCWWRYsJjNSWcvA
QA7JoBAd-h6I8AqT39tcU","dq":"BckMQjRg2zhnjZo2Gjw\_aSFJZ8iHo7CHCi98LdlD03BB9oC\_kCYEDMLGDr8d7j3h-
llQnoQGbmN\_ZeGy1l7Oy3wpG9TEWQEDEpYK0jWb7rBK79hN8l1CqyBlvLK5oi-
uYCaiHkwRQ4RACz9huyRxKLOz5VvlBixZnFXrzBHVPlk","qi":"M5NCVjSegf\_KP8kQLAudXUZi\_6X8T-
owtsG\_gB9xYVGnCsbHW8gccRocOY1Xa0KMotTWJl1AskCu-
TZhOJmrdeGpvkdulwmbIcnjA\_Fgflp4lAj4TCWmtRI6982hnC3XP2e-
nf\_z2XsPNiuOactY7W042D\_cajyyX\_tBEJaGOXM"}
```

**生成 KeyPair 参考示例(JAVA)**

```
import java.security.PrivateKey;

import org.jose4j.json.JsonUtil;
import org.jose4j.jwk.RsaJsonWebKey;
import org.jose4j.jwk.RsaJwkGenerator;
import org.jose4j.jws.AlgorithmIdentifiers;
import org.jose4j.jws.JsonWebSignature;
import org.jose4j.jwt.JwtClaims;
import org.jose4j.jwt.NumericDate;
import org.jose4j.lang.JoseException;



String keyId = UUID.randomUUID().toString().replaceAll("-", "");
RsaJsonWebKey jwk = RsaJwkGenerator.generateJwk(2048);
jwk.setKeyId(keyId);
jwk.setAlgorithm(AlgorithmIdentifiers.ECDSA_USING_P256_CURVE_AND_SHA256);
String publicKey = jwk.toJson(RsaJsonWebKey.OutputControlLevel.PUBLIC_ONLY);
String privateKey = jwk.toJson(RsaJsonWebKey.OutputControlLevel.INCLUDE_PRIVATE);
```

# 生成 id_token 参考步骤

通过 OIDC 协议中定义的 Claims 属性(aud, sub, exp, iat, iss)与其属性值，生成 Claims(全称 JwtClaims)

**示例代码(JAVA)**

```
JwtClaims claims = new JwtClaims();
claims.setGeneratedJwtId();
```

```
claims.setIssuedAtToNow();
//expire time
NumericDate date = NumericDate.now();
date.addSeconds(120);
claims.setExpirationTime(date);
claims.setNotBeforeMinutesInThePast(1);
claims.setSubject("YOUR_SUBJECT");
claims.setAudience("YOUR_AUDIENCE");
//添加自定义参数

claims.setClaim(key, value);
```

通过 keyId, Claims, privateKey 与使用的数字签名算法 (RSA SHA256 )生成 JWS( Json Web Signature)

**示例代码(JAVA)**

```
JsonWebSignature jws = new JsonWebSignature();
jws.setAlgorithmHeaderValue(AlgorithmIdentifiers.RSA_USING_SHA256);
jws.setKeyIdHeaderValue(keyId);
jws.setPayload(claims.toJson());
PrivateKey privateKey = new RsaJsonWebKey(JsonUtil.parseJson(privateKeyText)).getPrivateKey();
jws.setKey(privateKey);
```

通过 JWS 获取 id_token 值

**示例代码(JAVA)**

```
String idToken = jws.getCompactSerialization();
```

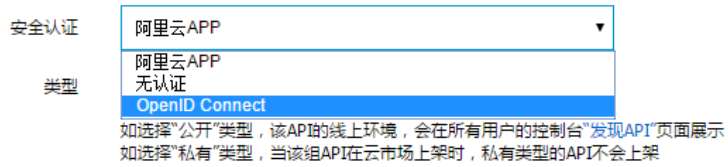
**一个生成的 id_token 示例：**

```
eyJhbGciOiJSUzI1NiIsImtpZCI6Ijg4NDgzNzI3NTU2OTI5MzI2NzAzMzA5OTA0MzUxMTg1ODE1NDg5In0.eyJ1c2VySWQiOiIzMzcwMTU0NDA2ODI1OTY4NjI3IiwidGFnTmFtZSI6ImNvbmFuVGVzdCIsImV4cCI6MTQ4MDU5Njg3OSwiYXVkIjoiQWxpX0FQSV9Vc2VyIiwianRpIjoiTm9DMFVVeW5xV0N0RUFEVjNoeEIydyIsImlhdCI6MTQ4MDU5MzI3OSwibmJmIjoxNDgwNTkzMjE5LCJzdWIiOiJ7ZGF0YU1hcD0ne3VzZXJJZD0zMzcwMTU0NDA2ODI1OTY4NjI3fScsIHN0YXR1c0NvZGU9JzAnLCBlcnJvcnM9J1tdJ30ifQ.V3rU2VCziSt6uTgdCktYRsIwkMEMsO_jUHNCCIW_Sp4qQ5ExjtwNt9h9mTGKFRujk2z1E0k36smWf9PbNGTZTWmSYN8rvcQqdsupcC6LU9r8jreA1Rw1CmmeWY4HsfBfeInr1wCFrEfZl6_QOtf3raKSK9AowhzEsnYRKAYuc297gmV8qlQdevAwU75qtg8j8ii3hZpJqTX67EteNCHZfhXn8wJjckl5sHz2xPPyMqj8CGRQ1wrZEHjUmNPw-unrUkt6neM0UrSqcjlrQ25L8PEL2TNs7nGVdl6iS7Nasbj8fsERMKcZbP2RFzOZfKJuaivD306cJIpQwxfS1u2bew
```

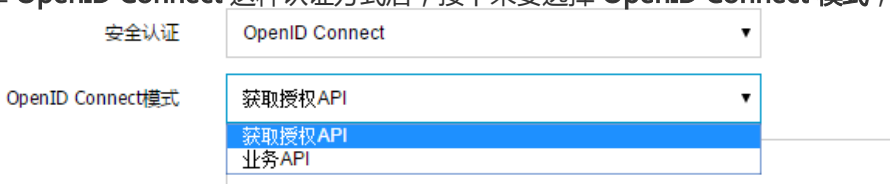
# 在 API 网关配置 API

API 编辑功能中，基本信息栏目中安全认证增加 **OpenID Connect** 选项，这种方式同时也包括了 **阿**

**里云APP** 认证方式，也就是说只有被授权的 APP 才能调用这个 API。




选择 **OpenID Connect** 这种认证方式后，接下来要选择 **OpenID Connect 模式**，有两个选项：

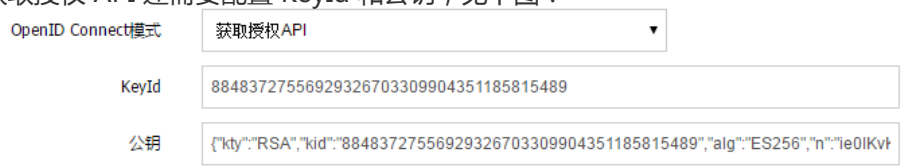



i. 获取授权 API：您用来换取 Token 的 API，比如：通过 U+P 换取 Token。

ii. 业务 API：也就是服务提供商提供服务的 API，调用者会把之前获取到的 Token 作为入参进行调用。

**OpenID Connect** 认证方式主要包含的就是以上这两种 API，下文将分别说明这两种API是如何配置的。

获取授权 API 还需要配置 KeyId 和公钥，见下图：




KeyId：公钥私钥对 对应的一个唯一 Id，由 As 模块负责生成的；示例：

```
88483727556929326703309904351185815489
```

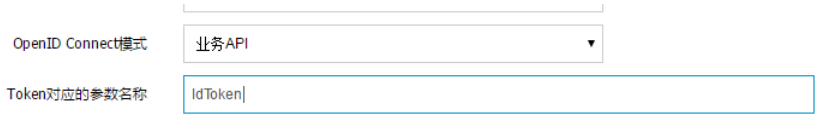
公钥：负责验证和解析 Token，由 As 模块负责生成的，示例：

```
{"kty":"RSA","kid":"88483727556929326703309904351185815489","alg":"ES256","n":"ie0IKvKLd7Y3izHcZemdDsVVXg5QtWtGF7XEkILnn66R2\_3a30DikqV409OVL7Hv0ElACgCaBLEgZeGHTcdLE1xxDTna8MMBnBNuMVghvFERCKh8uzpxlQsfcnFd5IFdJWj1x5Tscetrow6lA3h5zYx0rF5TkZzC4DclxgDmITRam0dsHBxr3uk9m9YYBz2mX0ehjY0px7vIo7hZH2J3gODEPorIZkk3x8GPdlaA4P9OFAO4au9-zcVQop9vLirxdwDedk2p-F9GP6UiQC9V2LTWqkVw\_oPBf9Rlh8Qdi19jA8SeCfzAxJZYlbOTK8dYAFAVEFsvXCFvdaxQefwWFw","e":"AQAB"}
```

设置完这些，后面的设置就和之前普通的 API 一样了，在此就不赘述了。

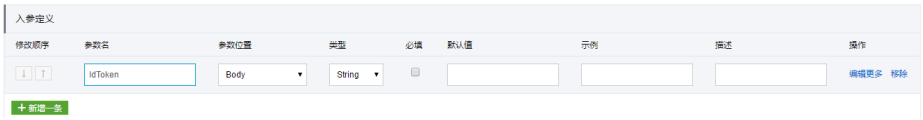
不管是创建 API 还是修改 API，所设置的 KeyId 和公钥都是在 API 发布之后才会生效。

业务 API 需要配置 **Token所对应的参数名称。**




如上图，Token 所对应的参数名称：就是调用者调用 API 传入 id_token 所使用的参数名，API 网关会识别这个参数值，校验并解析。

然后在入参定义中，必须要定义一个对应参数，否则系统会出现错误提示，见下图。




iii. 设置自定义系统参数：业务 API在 **定义 API 后端服务** 标签中会开放配置自定义系统参数，举个例子，见下图：

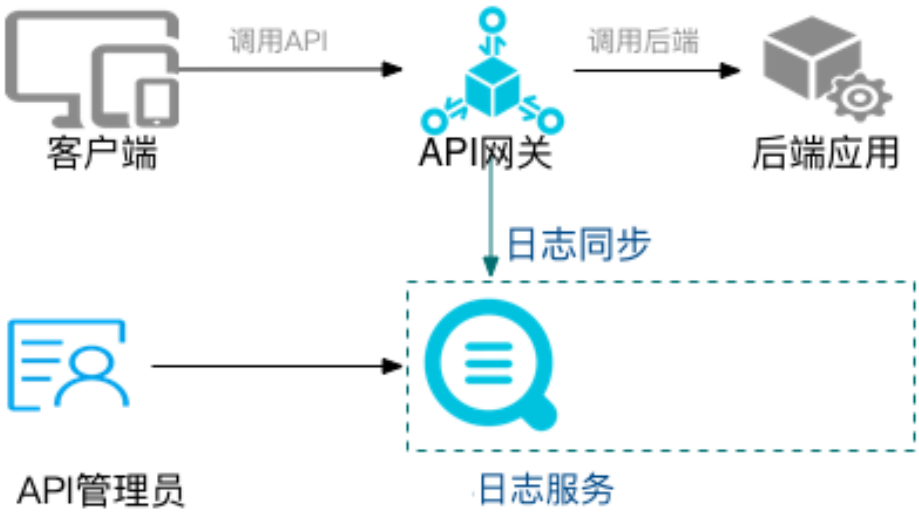

比如：AS 生成的 id_token 中包含了调用者的 userId，那么如果按照上图这样配置的话，就会把从调用者传过来的 id_token 中解析出来的 userId 传给服务提供方，配置方式和系统参数类似。

除了以上3处，定义 API 其他配置和前文一样，也就不再赘述了。

**以上就是 第三方账号认证 OpenID Connect 在 API 网关配置的全部内容，供您参考！**

# 通过日志服务查看API调用日志

API网关和日志服务实现无缝集成，通过日志服务您可以进行实时日志查询、下载、多维度统计分析等，您也可以将日志投递到OSS或者MaxCompute。




更多日志服务功能请参照：**日志服务帮助文档。**

日志服务每个月前500MB免费，具体价格请参照：**日志服务定价。**

# 1 功能简介

## 1.1 日志在线查询

可根据日志中任意关键字进行快速的精确、模糊检索，可用于问题定位或者统计查询。

## 1.2 详细调用日志

您可以检索API调用的详细日志包含：

| 日志项 | 描述 |
| --- | --- |
| apiGroupUid | API的分组ID |
| apiGroupName | API分组名称 |
| apiUid | API的ID |
| apiName | API名称 |
| apiStageUid | API环境ID |
| apiStageName | API环境名称 |
| httpMethod | 调用的HTTP方法 |
| path | 请求的PATH |
| domain | 调用的域名 |

| | |
|---|---|
| statusCode | HttpStatusCode |
| errorMessage | 错误信息 |
| appId | 调用者应用ID |
| appName | 调用者应用名称 |
| clientIp | 调用者客户端IP |
| exception | 后端返回的具体错信息 |
| providerAliUid | API提供者帐户ID |
| region | 区域，如：cn-hangzhou |
| requestHandleTime | 请求时间，UTC |
| requestId | 请求ID，全局唯一 |
| requestSize | 请求大小，单位：字节 |
| responseSize | 返回数据大小,单位：字节 |
| serviceLatency | 后端延迟，毫秒 |

## 1.4 自定义分析图表

您可以根据统计需求将任意日志项自定义统计图表，以满足您日常的业务需要。

## 1.3 预置分析报表

为能让您快速使用，API网关预定义了一些统计图表（全局）。包括：请求量大小、成功率、错误率、延时情况、调用API的APP数量，错误情况统计、TOP 分组、TOP API、Top 延迟等等。

# 2 使用日志服务查看API日志

## 2.1 配置日志服务

使用此功能前，请确保您已经开通了日志服务，并创建Project和Logstore，点击开始创建。

您可以在API网关的控制台上来配置，也可以在日志服务控制台上来配置。

### 2.1.1 在API网关控制台配置

1）打开API网关控制台-【开放API】-【日志管理】，选择您服务所在的区域，下图以华北1为例：


2）点击”创建日志配置”，进入日志配置界面


3）选择您所需要输入的日志服务的Project或者Logstore，若无法选择，请点击“授权日志写操作”，点击后会进行授权操作


4）确认后，完成网关与日志服务的关联

5）开启索引后完成配置

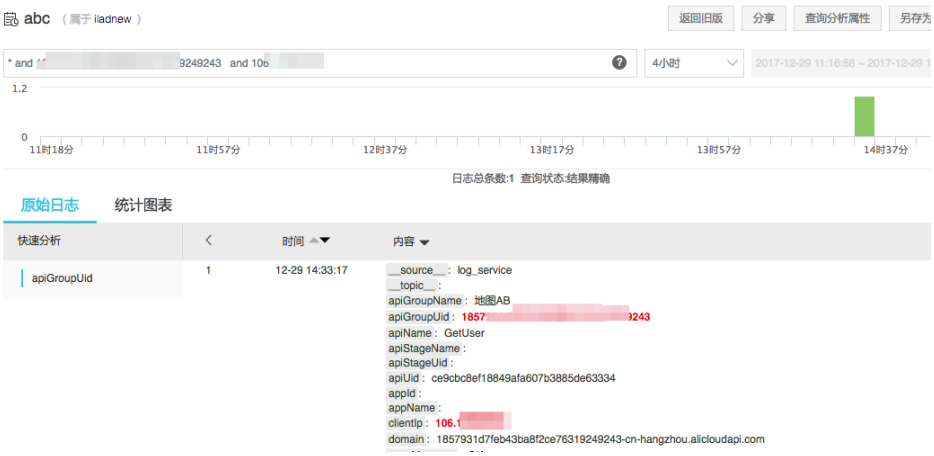
### 2.1.1.2 在日志服务控制台配置

您可以参照：API网关访问日志

*配置完成后，您的API调用记录即可进入日志服务的Logstore中*

## 2.2 查看日志

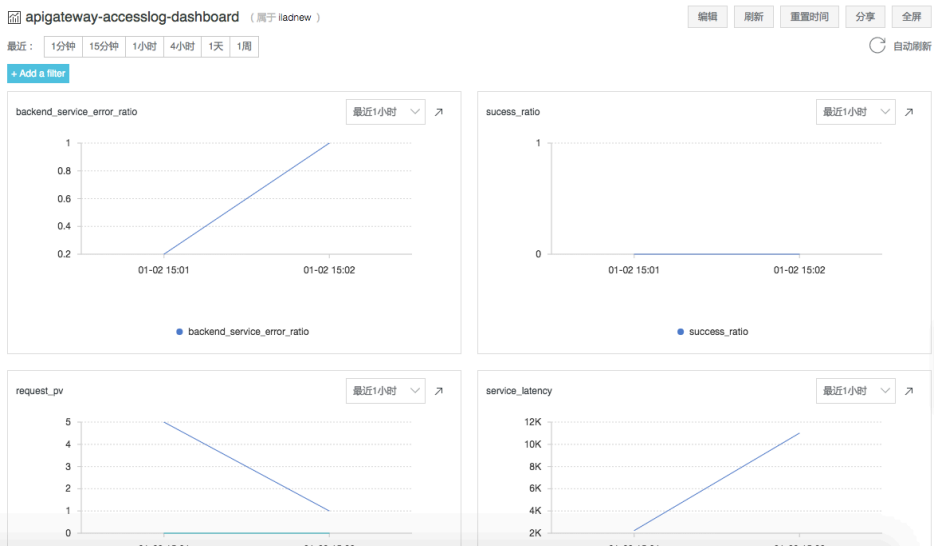
你可以点击API网关控制台-【开放API】-【日志管理】-“访问日志”，跳转到日志控制台，根据日志服务的查询语法，在线检索调用日志。如图：




你也可以登录日志服务控制台查看日志，可参照：查询日志

## 2.3 查询预定义报表

预定义报表，是API网关为了方便用户统计查询，而预置的一些统计报表，可以点击API网关控制台-【开放API】-【日志管理】-“日志统计”来访问。也可以在日志服务控制台来查看:




## 2.4 自定义查询报表

你可以根据自身业务需要自定义查询报表，请参照定义方法：仪表盘

# 3 维护日志

您打开API网关控制台-【开放API】-【日志管理】，进行“修改配置”或者删除配置。

- **修改配置**：为更换新日志服务的Project或者Logstore，更换后的API调用日志将会写入新的Logstore，但历史数据仍会保留在原Logstore中，不会跟随迁移。
- **删除配置**：为删除API网关与日志服务的映射关系，删除后将不会再同步API调用日志到日志服务，但并不会删除Logstore中已有的数据。

# 使用 RAM 管理 API

API 网关结合阿里云访问控制（RAM）来实现企业内多职员分权管理 API。API 提供者可以为员工建立子账户，并控制不同职员负责不同的 API 管理。

- 使用 RAM 可以允许子帐号，查看、创建、管理、删除 API 分组、API、授权、流控策略等。但子帐号不是资源的所有者，其操作权限随时都可以被主帐号收回。
- 在查看本文前，请确保您已经祥读了 RAM 帮助手册 和 API 网关 API 手册。
- **若您不无此业务场景， 请跳过此章节。**

你可以使用 RAM 的控制台 或者 API 来添加操作。

## 第一部分：策略管理

授权策略（Policy），来描述授权的具体内容，授权内容主要包含效力( Effect )、资源( Resource )、对资源所授予的操作权限( Action )以及限制条件( Condition )这几个基本元素。

### 系统授权策略

API 网关已经预置了两个系统权限，AliyunApiGatewayFullAccess和AliyunApiGatewayReadOnlyAccess，可以到 RAM 的在 RAM 控制台-策略管理 进行查看。


- AliyunApiGatewayFullAccess: 管理员权限，拥有主帐号下包含 API 分组、API、流控策略、应用等所有资源的管理权限。
- AliyunApiGatewayReadOnlyAccess：可以查看主帐号下包含 API 分组、API、流控策略、应用等所有资源，但不可以操作。

### 自定义授权策略

您可以根据需要自定义管理权限，支持更为精细化的授权，可以为某个操作，也可以是某个资源。如：API

GetUsers 的编辑权限。可以在 RAM 控制台-策略管理-自定义授权策略查看已经定义好的自定义授权：自定义授权查看、创建、修改、删除方法请参照：授权策略管理。
授权策略内容填写方法请参照：Policy 基本元素 和 Policy 语法结构 和下文的授权策略。

# 第二部分：授权策略

授权策略是一组权限的集合，它以一种策略语言来描述。通过给用户或群组附加授权策略，用户或群组中的所有用户就能获得授权策略中指定的访问权限。
授权策略内容填写方法请参照：Policy 基本元素 和 Policy 语法结构。
**示例:**

```
{
"Version": "1",
"Statement": [
{
"Action": "apigateway:Describe*",
"Resource": "*",
"Effect": "Allow"
}
]
}
```

此示例表示：允许所有的查看操作

**Action（操作名称列表）格式为：**

```
"Action":"<service-name>:<action-name>"
```

其中：

- **service-name** 为：阿里云产品名称,请填写 **apigateway**
- **action-name** 为：API 接口名称，请参照下表, 支持通配符 *****

```
"Action": "apigateway:Describe*" 表示所有的查询操作
" Action": "apigateway:*" 表示 API 网关所有操作
```

# 第三部分：Resource（操作对象列表）

Resource 通常指操作对象， API 网关中的 API 分组、流控策略、应用都被称为 Resource，书写格式：

```
acs:<service-name>:<region>:<account-id>:<relative-id>
```

其中：

- **acs**：Alibaba Cloud Service 的首字母缩写，表示阿里云的公有云平台

- **service-name** 为：阿里云产品名称,请填写 apigateway
- **region**：地区信息，可以使用通配符*号来代替，*表示所有区域
- **account-id**：账号 ID，比如 1234567890123456，也可以用*代替
- **relative-id**：与 API 网关相关的资源描述部分，这部分的格式描述支持类似于一个文件路径的树状结构。

示例：

```
acs:apigateway:$regionid:$accountid:apigroup/$groupId
```

书写：

```
acs:apigateway:*:$accountid:apigroup/
```

请结合 API 网关的 API 手册 来查看下表

| action-name | 资源( Resource ) |
| --- | --- |
| AbolishApi | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| AddTrafficSpecialControl | acs:apigateway:$regionid:$accountid:trafficcontrol/$trafficcontrolid |
| CreateApi | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| CreateApiGroup | acs:apigateway:$regionid:$accountid:apigroup/* |
| CreateTrafficControl | acs:apigateway:$regionid:$accountid:trafficcontrol/* |
| DeleteAllTrafficSpecialControl | acs:apigateway:$regionid:$accountid:trafficcontrol/$trafficcontrolid |
| DeleteApi | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| DeleteApiGroup | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| DeleteDomain | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| DeleteDomainCertificate | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| DeleteTrafficControl | acs:apigateway:$regionid:$accountid:trafficcontrol/$trafficcontrolId |
| DeleteTrafficSpecialControl | acs:apigateway:$regionid:$accountid:trafficcontrol/$trafficcontrolId |
| DeployApi | acs:apigateway:$regionid:$accountid:apigroup/$groupId |

| | |
|---|---|
| DescribeApi | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| DescribeApiError | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| DescribeApiGroupDetail | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| DescribeApiGroups | acs:apigateway:$regionid:$accountid:apigroup/* |
| DescribeApiLatency | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| DescribeApiQps | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| DescribeApiRules | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| DescribeApis | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| DescribeApisByRule | acs:apigateway:$regionid:$accountid:trafficcontrol/$trafficcontrolId oracs:apigateway:$regionid:$accountid:secretkey/$secretKeyId |
| DescribeApiTraffic | acs:apigateway:$regionid:$accountid:apigroup/$groupid |
| DescribeAppsByApi | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| AddBlackList | acs:apigateway:$regionid:$accountid:blacklist/* |
| DescribeBlackLists | acs:apigateway:$regionid:$accountid:blacklist/* |
| DescribeDeployedApi | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| DescribeDeployedApis | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| DescribeDomain | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| DescribeDomainResolution | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| DescribeHistoryApi | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| DescribeHistoryApis | acs:apigateway:$regionid:$accountid:apigroup/* |
| DescribeRulesByApi | acs:apigateway:$regionid:$accountid:group/$groupId |
| DescribeSecretKeys | acs:apigateway:$regionid:$accountid:secretke |

| | |
|---|---|
| | y/* |
| DescribeTrafficControls | acs:apigateway:$regionid:$accountid:trafficcontrol/* |
| ModifyApi | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| ModifyApiGroup | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| ModifySecretKey | acs:apigateway:$regionid:$accountid:secretkey/$secretKeyId |
| RecoverApiFromHistorical | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| RefreshDomain | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| RemoveAccessPermissionByApis | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| RemoveAccessPermissionByApps | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| RemoveAllBlackList | acs:apigateway:$regionid:$accountid:blacklist/* |
| RemoveApiRule | acs:apigateway:$regionid:$accountid:apigroup/$groupId(acs:apigateway:$regionid:$accountid:secretkey/$secretKeyId oracs:apigateway:$regionid:$accountid:trafficcontrol/$trafficcontrolId) |
| RemoveAppsFromApi | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| RemoveBlackList | acs:apigateway:$regionid:$accountid:blacklist/$blacklistid |
| SetAccessPermissionByApis | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| SetAccessPermissions | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| SetApiRule | acs:apigateway:$regionid:$accountid:apigroup/$groupId(acs:apigateway:$regionid:$accountid:secretkey/$secretKeyId oracs:apigateway:$regionid:$accountid:trafficcontrol/$trafficcontrolId) |
| SetDomain | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| SetDomainCertificate | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| SwitchApi | acs:apigateway:$regionid:$accountid:apigroup/$groupId |
| CreateSecretKey | acs:apigateway:$regionid:$accountid:secretke |

| | |
|---|---|
| | y/* |
| DeleteSecretKey | acs:apigateway:$regionid:$accountid:secretkey/$secretKeyId |

# 专属网络 VPC 环境开放 API

使用阿里云专属网络VPC，可以构建出一个隔离的网络环境，并可以自定义IP 地址范围、网段、路由表和网关等；此外，也可以通过专线/VPN/GRE等连接方式实现云上VPC与传统IDC的互联，构建混合云业务。

API网关也支持您部署在专属网络VPC中的服务开放API。在开始此文章之前，请确认您已经了解专属网络VPC的使用方法。

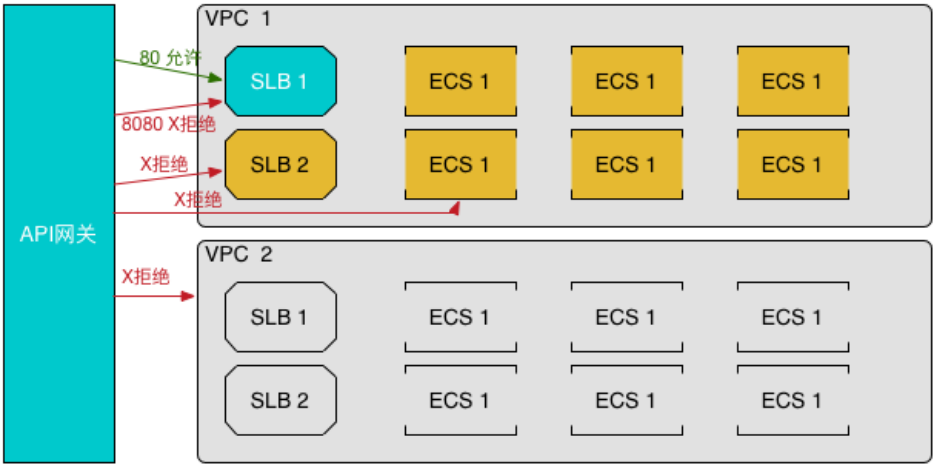
若您的后端服务在VPC环境，需要进行授权API网关访问才可开放相应API。创建API流程如下：


# 1 授权与绑定VPC

开放VPC环境的API，需要您先授权API网关可访问您VPC内的服务。授权时需指定API网关可以访问的资源+端口，如：SLB 的443端口、ECS 的80端口。

- 授权成功后，API网关将通过内网访问VPC内部资源
- 此授权只会被用作API网关访问相应后端资源
- API网关不可访问未被授权的资源或者端口

例如：只将VPC 1中SLB 1的80端口授权给API网关，那么API网关只能访问vpc 1中SLB 1的80端口




# 1.1 准备VPC环境

1) 购买VPC环境的，SLB、ECS，并搭建服务，可以参照VPC使用手册

2）查询VPC信息，需要准备如下VPC信息

- VPC ID：您后端服务所在的VPCID
- 实例ID：您后端所在的实例ID，可以是ECS或者SLB的实例ID，若使用了SLB，请填写SLB的实例ID
- 端口号：调用您后端服务所使用的端口号

# 1.2 授权API网关访问

进入【API网关控制台】-【开放API】-【VPC授权】，点击“创建授权”


进入授权页面，填写相应信息

- VPC名称：为此条授权的名称标识，供创建API时选择后端地址使用，所以为了便于后续管理，请保证此名称的唯一。


点击确定，完成授权

**若您有多个VPC，或者要授权多个实例、端口，请重复如上步骤**

# 2 创建API

创建API的流程与其他类型API方式一致，请参照：创建API。

在选择后端服务地址时，请选择：

- VPC通道：请选择“使用VPC通道”
- VPC授权：请选按需求选择所创建的授权

其他部分内容，与其他API定义方式一致


保存后，则API创建完成。

# 3 安全组授权

**视情况必需**：对于后端使用SLB，或者没有改动过ECS安全组授权策略的用户，可以跳过此步。

若您API的后端服务为ECS，且您修改过安全组“内网入方向”访问策略，需要增加问策略允许如下IP段访问（请根据服务所区域配置）。

| 区域 | 方向 | IP |
| --- | --- | --- |
| 杭州 | 内网入方向 | 100.104.13.0/24 |
| 北京 | 内网入方向 | 100.104.106.0/24 |
| 深圳 | 内网入方向 | 100.104.8.0/24 |
| 上海 | 内网入方向 | 100.104.8.0/24 |
| 香港 | 内网入方向 | 100.104.175.0/24 |
| 新加坡 | 内网入方向 | 100.104.175.0/24 |

# 3 API测试

可以到通过以下方式测试您的API

- 调试API
- 下载SDK
- 使用API调用Demo

# 4 解除授权

若您授权的资源或者端口不再提供服务，请删除相应授权。


# 5 FAQ

## 使用此功能是否有额外费用？

否，此功能免费使用，无额外费用产生。

## 是否可以绑定多个VPC？

可以，若您的后端服务在多个VPC，可以添加多个授权。

## 为什么我无法授权我的VPC

请确认，VPCID、实例ID和端口号的正确，并保证授权策略和VPC在同一个区域。

## 授权API网关后，我的VPC安全么？

当您的VPC授权API网关可访问，网关与VPC的网络联通。在安全方面做了限制，不会造成VPC的安全问题。

1. 安全控制授权操作：只有VPC的所有者能够操作授权。
2. 授权后API网关与VPC建立专享通道：他人不可使用此同道。
3. 授权是针对某个资源的端口：无其他端口或资源的访问权限。

# Mock 您的 API

在项目开发过程中，往往是多个合作方一同开发，多个合作方相互依赖，而这种依赖在项目过程中会造成相互

制约，理解误差也会影响开发进度，甚至影响项目的工期。所以在开发过程中，一般都会使用 Mock 来模拟最初预定的返回结果，来降低理解偏差，从而提升开发效率。

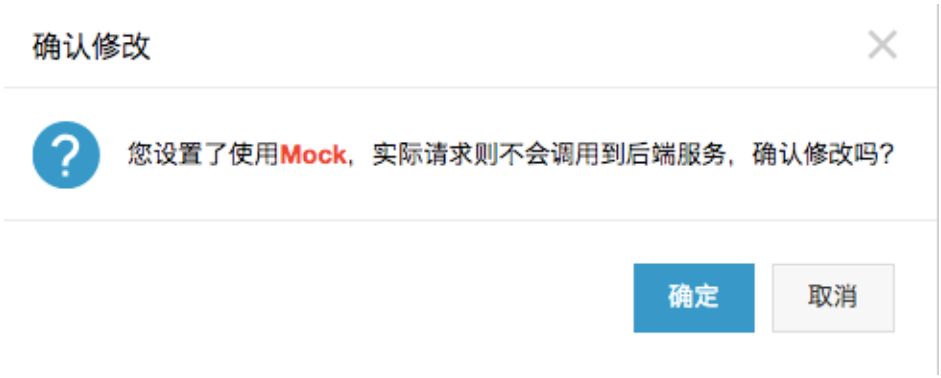
API 网关也支持 Mock 模式的简单配置。

# 配置 Mock

在 API 编辑页面——后端基础定义，来配置 Mock。


选择 Mock 模式

可以选择使用 Mock 或者不使用 Mock，选择使用 Mock 时会进行二次确认




填写 Mock 返回结果

Mock 返回结果，可以填写您真实的返回结果。目前支持是 Json、XML、文本等格式作为 Mock 返回结果。如

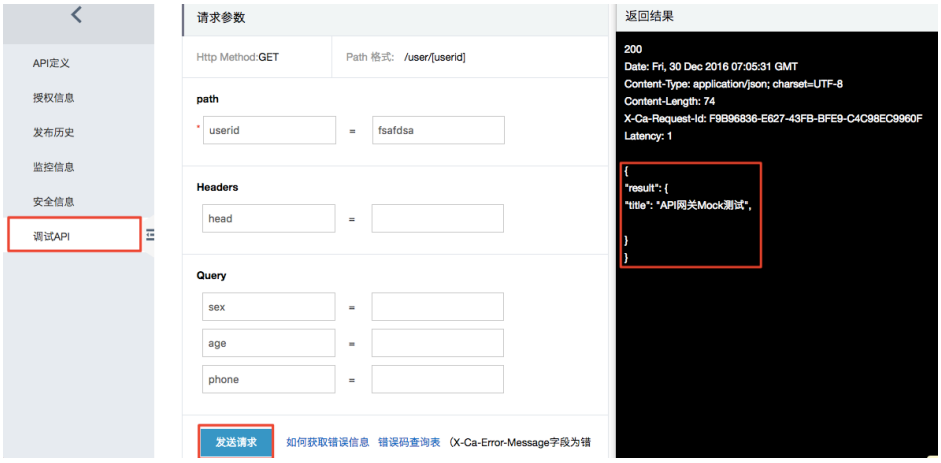
```
{
"result": {
"title": " API 网关 Mock 测试",

}
}
```

保存后 Mock 设置成功，请根据实际需要 **发布** 到测试或线上环境进行测试，也可以在 API 调试页面进行调试。

调试

可以在调试 API 页面发起 API 调用来测试设置结果：




这表示 Mock 设置成功。

## 解除 Mock

若您需要解除 Mock，可以将第一幅图中的 Mock，修改为 **不使用 Mock** 即可，而 Mock 返回结果中的值不会被清除，以便您进行下一次的 Mock 。修改完成后请 **发布**，只有发布后才会真正生效。

# 双向通信使用指南

## 一.概述

移动端APP大多数功能都能通过客户端向服务器端发送请求，服务器应答来完成。比如：用户注册，获取商品列表等能力。

但有一些场景需要服务器向客户端推送应用内通知，如：用户之间的即时通信等功能。这种时候就需要建立一个通信通道，让服务器能够给指定的客户端发送下行通知请求。也就是客户端和服务器端之间具备双向通信的能力。

**具备双向通行能力的架构对于移动APP属于刚性需求。**

# 使用须知

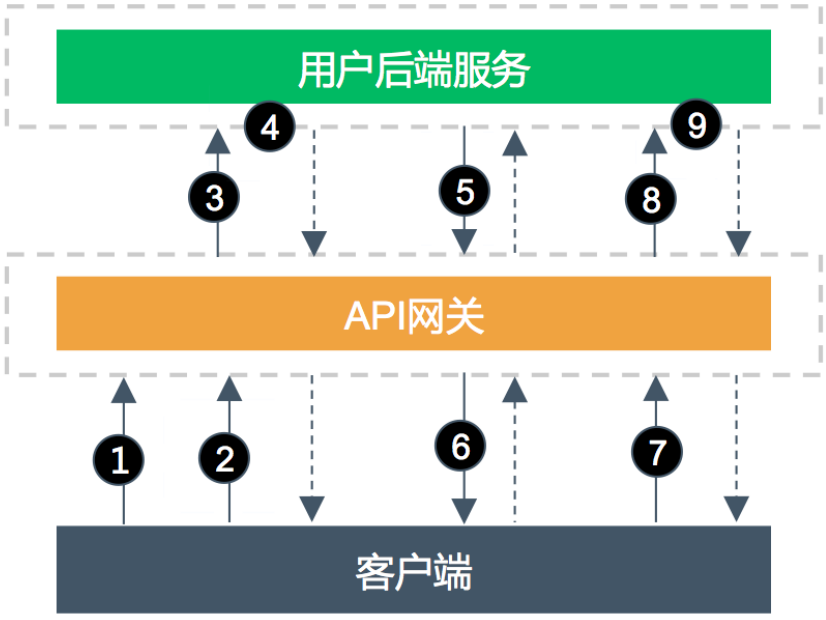
API网关将逐渐放开此能力给所有用户，目前仅放开了杭州和香港两个Region。且双向通信能力构建于WebSocket协议之上，目前仅提供Android版本的SDK，后期将提供iOS的SDK。

（若您有此之外的需求可以工单或者钉钉群：11747055进行咨询）

# 实现方式

API网关目前已经具备双向通信的能力，用户只需要在API网关上设置三个API，然后下载自动生成的SDK到客户端，简单嵌入到客户端就能完美实现客户端和服务器端之间的双向通信的功能。

下面是利用API网关实现双向通信的能力的业务流程简图：






## 流程描述

(1) 客户端在启动的时候和API网关建立了WebSocket连接，并且将自己的设备ID告知API网关；

(2) 客户端在WebSocket通道上发起注册信令；

(3) API网关将注册信令转换成HTTP协议发送给用户后端服务，并且在注册信令上加上设备ID参数；

(4) 用户后端服务验证注册信令，如果验证通过，记住用户设备ID，返回200应答；

(5) 用户后端服务通过HTTP/HTTPS/WebSocket三种协议中的任意一种向API网关发送下行通知信令，请求中携带接收请求的设备ID；

(6) API网关解析下行通知信令，找到指定设备ID的连接，将下行通知信令通过WebSocket连接发送给指定客户端；

(7) 客户端在不想收到用户后端服务通知的时候，通过WebSocket连接发送注销信令给API网关，请求中不携带设备ID;

(8) API网关将注销信令转换成HTTP协议发送给用户后端服务，并且在注册信令上加上设备ID参数；

(9) 用户后端服务删除设备ID，返回200应答。

# 二.双向通信三种管理信令

要使用API网关的双向通信能力，首先要了解API网关双向通信相关的三种信令，需要注意的是，这三个信令其实就是API网关上的三个API，需要用户去API网关创建后才能使用。

## 1.注册信令

注册信令是客户端发送给用户后端服务的信令，起到两个作用：

（1）将客户端的设备ID发送给用户后端服务，用户后端服务需要记住这个设备ID。用户不需要定义设备ID字段，设备ID字段由API网关的SDK自动生成；

（2）用户可以将此信令定义为携带用户名和密码的API，用户后端服务在收到注册信令的验证客户端的合法性。用户后端服务在返回注册信令应答的时候，返回非200时，API网关会视此情况为注册失败。

客户端要想收到用户后端服务发送过来的通知，需要先发送注册信令给API网关，收到用户后端服务的200应答后正式注册成功。

## 2.下行通知信令

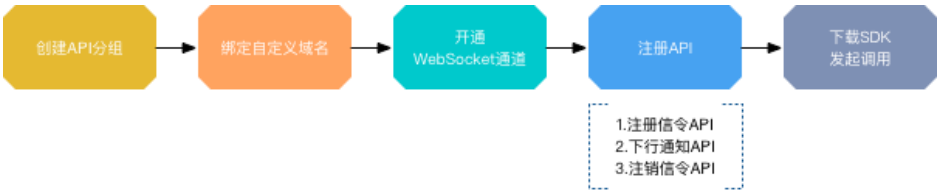
用户后端服务，在收到客户端发送的注册信令后，记住注册信令中的设备ID字段，然后就可以向API网关发送接收方为这个设备的下行通知信令了。只要这个设备在线，API网关就可以将此下行通知发送到端。

## 3.注销信令

客户端在不想收到用户后端服务的通知时发送注销信令发送给API网关，收到用户后端服务的200应答后注销成功，不再接受用户后端服务推送的下行消息。

# 三.API网关设置双向通信

# 1.开通绑定分组域名的WebSocket通道




## 1.1 创建分组

*已经有分组的情况可忽略本节。*

要使用API网关的基本功能，首先需要在API网关上创建一个分组，关于分组的创建，请参见文档：
https://help.aliyun.com/document_detail/29493.html

## 1.2 在分组上绑定域名

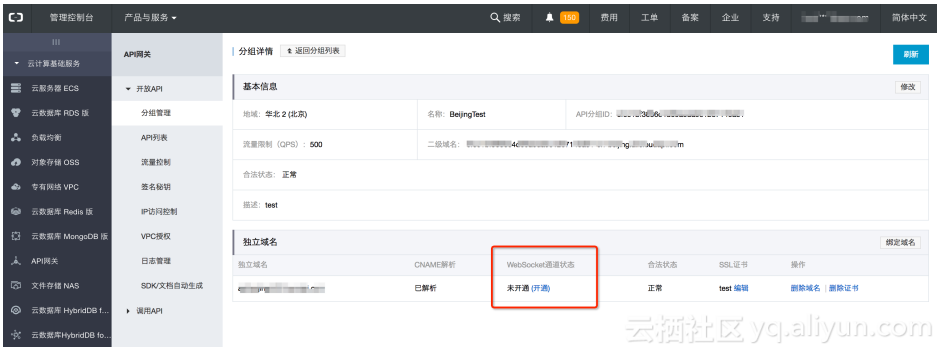
*已经已经绑定了域名的情况可忽略本节。*

创建完分组后，需要在分组上绑定一个域名，关于分组上域名的绑定，请参见文档：
https://help.aliyun.com/document_detail/29494.html

## 1.3 开通域名的WebSocket通道

绑定好域名后，需要开通域名上的WebSocket通道。

具体开通方法如下：开通页面路径：【API网关控制台】->【开放API】->分组详情




# 2.创建注册、下行通知、注销信令的API

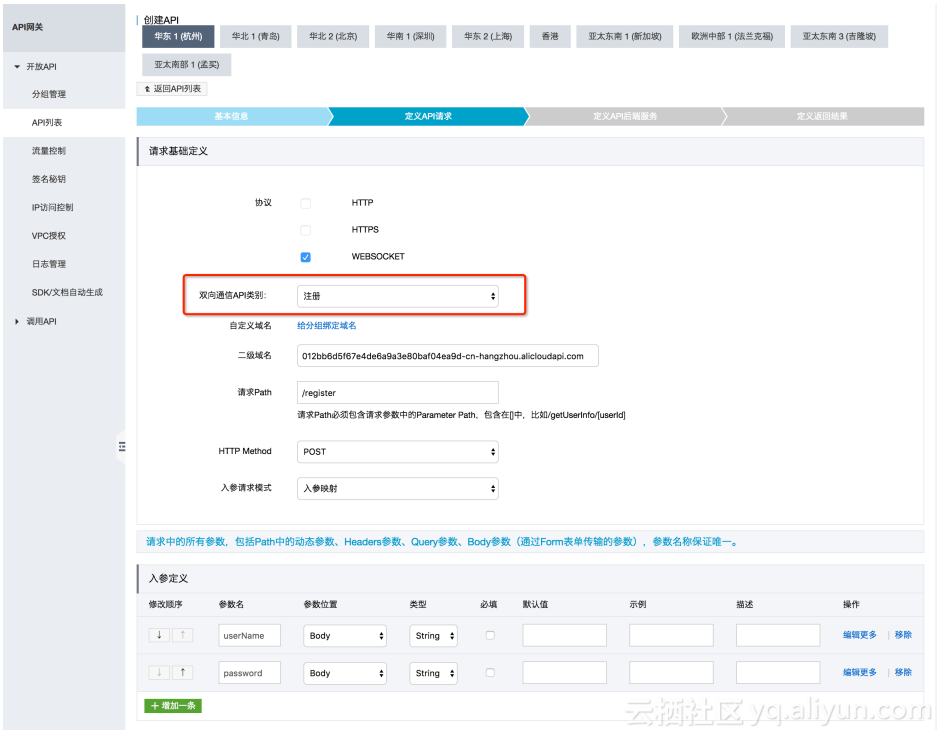
需要刚才创建的分组下创建三个API，普通API的创建流程请参见文档：
https://help.aliyun.com/document_detail/29478.html

这三个API在创建的时候，需要特别注意的是，需要选择“双向通信API类别”选项。“双向通信API类别”选项在勾选“WebSocket协议”时会自动弹出，如下图：


# 2.1 注册信令API

注册信令是客户端发送给用户后端服务的信令，创建的时候需要注意的是：

(1) 一般会包含用户名和密码两个字段，如图所示：




(2) 只能通过WebSocket协议传输

当然这个信令的定义由用户自己去定义，包含什么参数都是可以的。重要的是，这个信令的应答必须是200，客户端才算注册成功。

## 2.2 下行通知API

下行通知信令是用户后端服务发送给客户端的信令，创建的时候需要注意的是：

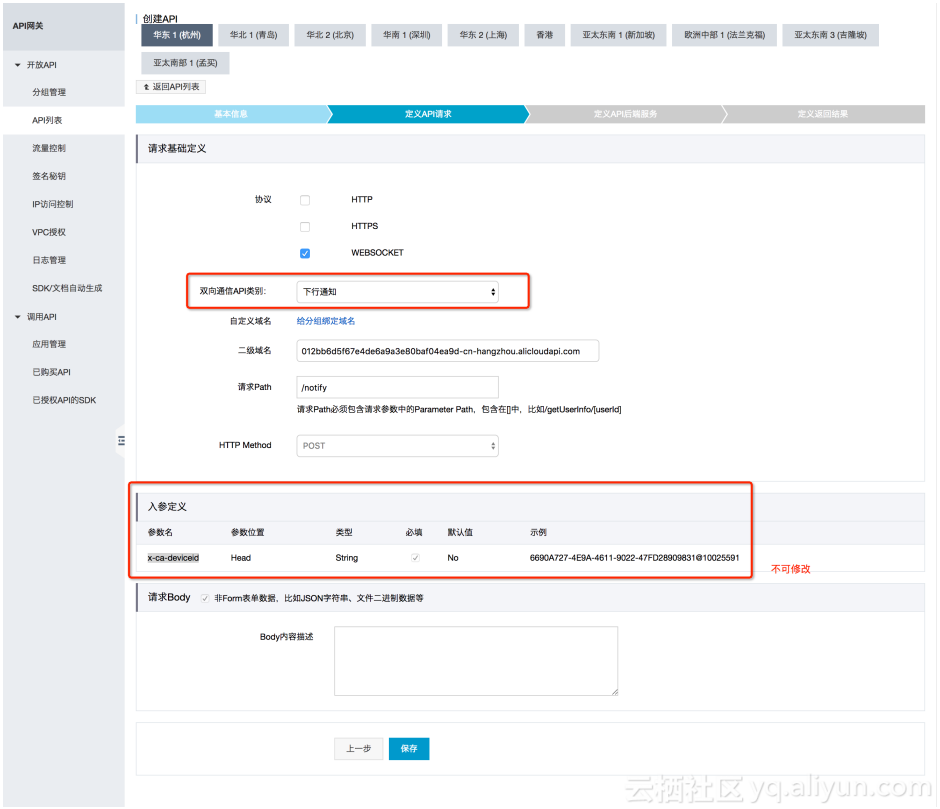
(1) 强烈建议使用和客户端不同的APP进行授权，区分用户后端服务和客户端调用权限；

(2) 可以使用HTTP/HTTPS/WebSocket中任意协议调用此API；

(3) 因为是发送给客户端的，因此不需要和其他API一样定义后端服务参数；

(4) 请求中必须携带接收通知的客户端ID的x-ca-deviceid头，且不可修改；

创建页面如下图：




## 2.3 注销信令API

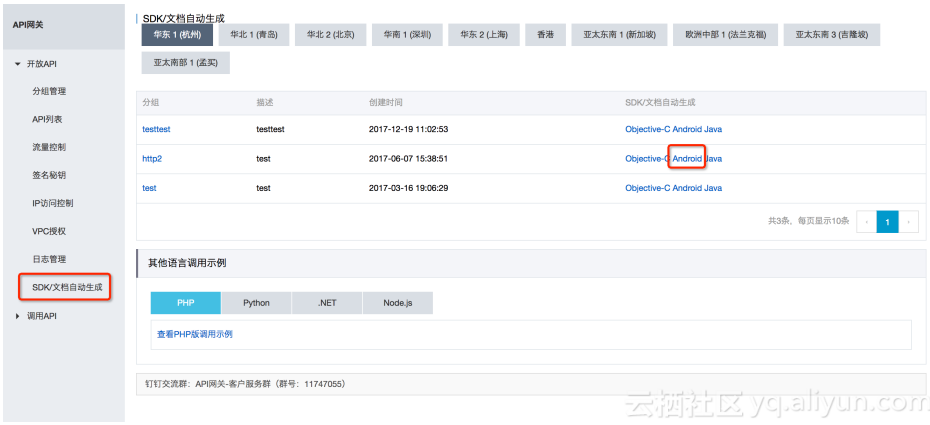
注销信令是客户端发送给用户后端服务的信令，创建的时候需要注意的是只能通过WebSocket协议传输。

## 3.生成、下载SDK

在三个信令API全部创建成功后，需要将分别授权到指定APP上。授权后，需要发布到线上环境。在授权、发布完成后，就可以到SDK文档自动生成页面去生成、下载SDK了。

目前API网关提供的SDK中具备WebSocket通信能力的只有Android，其他语言的SDK目前还不具备WebSocket通信能力。近期将升级iOS的SDK，让它支持WebSocket通信通道。

因此您可以下载Android的SDK作为客户端SDK，使用这个SDK来使用WebSocket和API网关进行通信，并在此SDK上发送注册信令，接收用户后端服务发送的下行通知信令。您可以下载JAVA语言的SDK作为服务器端SDK，用来发送下行通知信令。两个SDK配合完成双向通信功能。下面是下载页面：




## 4.调用SDK的注册信令,并在SDK中读取下行通知信令的内容

Android的SDK下载下去后，需要仔细阅读SDK的安装和使用说明，也就是ReadMe.txt文件。

自动生成的SDK里面有所有API的调用入口，我们找到调用入口，就可以调用这个API来发送注册信令了。下面是一个Demo示例，在此Demo中，我们在启动的时候先初始化一个WebSocket通道出来，在通道中注册接收用户后端服务发送的下行通知的函数onNotify，客户端接收到用户后端服务发送的下行通知，SDK会调用这个函数来通知客户端。然后Demo提供注册函数registerWebsocketTest供外部调用。

```
public class Demo_HangZhou {
static{
WebSocketClientBuilderParams websocketParam = new WebSocketClientBuilderParams();
websocketParam.setAppKey("12345678");
websocketParam.setAppSecret("12345678");
websocketParam.setApiWebSocketListner(new ApiWebSocketListner() {
@Override
//客户端接收到用户后端服务发送的下行通知，SDK会调用这个函数来通知客户端
public void onNotify(String message) {
System.out.println(message);
}

@Override
public void onFailure(Throwable t, ApiResponse response) {
if(null != t){
t.printStackTrace();
}

if(null != response){
System.out.println(response.getCode());
System.out.println(response.getMessage());
}
```

```
}
});

WebSocketApiClient_hangzhou.getInstance().init(websocketParam);


}



public static void registerWebsocketTest(){
WebSocketApiClient_hangzhou.getInstance().register("fred" , "123456" , new ApiCallback() {

@Override
public void onFailure(ApiRequest request, Exception e) {
e.printStackTrace();
}

@Override
public void onResponse(ApiRequest request, ApiResponse response) {
try {
System.out.println(getResultString(response));
}catch (Exception ex){
ex.printStackTrace();
}
}
});
}
}
```

## 5.用户后端服务发送下行通知

用户后端服务在接收到客户端发送的注册信令后，需要记住信令请求中设备ID。然后用户后端服务给客户端发送下行通知就变得非常容易，就和调用普通API一样，发送一个标准的API调用给API网关就可以了。下面是调用代码示例：

```
public class Demo_Hanzhou {

static{
HttpClientBuilderParams param = new HttpClientBuilderParams();
param.setAppKey("123456");
param.setAppSecret("123456");
HttpApiClient_BeiJing.getInstance().init(param);

}

public static void HanZhouNotifyTest(){
HttpApiClient_HanZhou.getInstance().notify("NotifyContent" , new ApiCallback() {

@Override
public void onFailure(ApiRequest request, Exception e) {
e.printStackTrace();
```

```
}

@Override
public void onResponse(ApiRequest request, ApiResponse response) {
try {
System.out.println(response.getCode());
}catch (Exception ex){
ex.printStackTrace();
}
}
});
}

}
```

让我们再来整理一下API网关双向通信能力的使用流程：

1. 开通分组绑定的域名的WebSocket通道；
2. 创建注册、下行通知、注销三个API，给这三个API授权、并上线；
3. 用户后端服务实现注册，注销信令逻辑，下载JAVA的SDK发送下行通知；
4. 下载AndroidSDK，嵌入到客户端，建立WebSocket连接，发送注册请求，监听下行通知；

如果在使用中遇到棘手的问题，请加入我们钉钉交流群：API网关-客户服务群（群号：11747055）

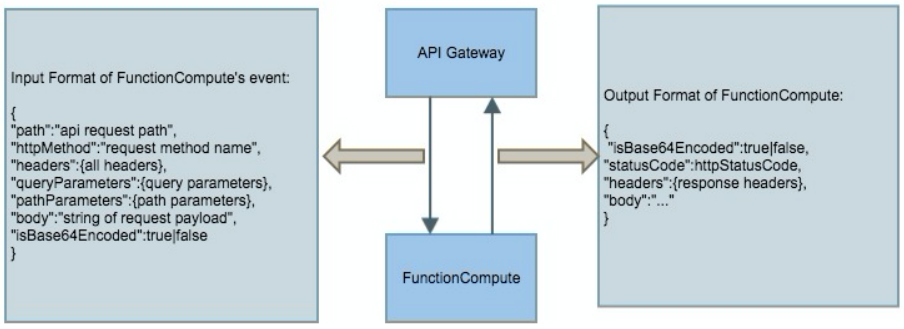
# 以函数计算作为 API 网关后端服务

函数计算（Function Compute）是一个事件驱动的服务，通过函数计算，用户无需管理服务器等运行情况，只需编写代码并上传，函数计算准备计算资源，并以弹性伸缩的方式运行用户代码。而用户只需根据实际代码运行所消耗的资源付费。详细了解请查看函数计算帮助文档。

API网关与函数计算对接，可以让您以API形式开放您的函数，并且解决认证、流量控制、数据转换等问题(查看API网关功能），让您的函数服务可以安全、简单的以API形式对外开放。

## 1 实现原理

API网关调用函数服务时，会将API的相关数据包装为一个Map形式传给函数计算服务，函数计算服务处理后，需要按照图中Output Format的格式返回statusCode，headers，body等相关数据，API gateway再将函数计算返回的内容映射到statusCode，header,body等位置返回给客户端。




# 2 快速配置API

配置方法：


## 2.1 创建函数计算的Function

1）到函数计算控制台创建Service。去往 函数计算控制台。如果您已经创建过Function，可以忽略此步。


2）在刚创建的Service下创建Function，点击“新建函数”。并输入您的代码,详细操作请参照创建函数。


3）触发器页面点击“跳过” 即可，最后点击完成。

## 2.2 配置跨服务授权Role和policy

在此步，您需要将此函数的调用权限授权给API网关。

1）到RAM控制台创建角色，去往RAM控制台。在“角色管理”页面，点击“创建角色”。


创建角色
1：选择角色类型
2：填写类型信息
3：配置角色基本信息
4：创建成功
用户角色
受信云账号下的子用户可以通过扮演该角色来访问您的云资源，受信云账号可以是当前云帐号，也可以是其他云账号。
服务角色
受信云服务可以通过扮演该角色来访问您的云资源。
创建角色
1：选择角色类型
2：填写类型信息
3：配置角色基本信息
4：创建成功
选择受信服务，受信服务将可以使用此角色来访问您的云资源。
OAS 归档存储服务
将OSS Bucket设置为归档存储服务的数据源时，您需要创建一个以归档存储服务为受信服务的角色，归档存储服务将扮演该角色来读写您OSS的数据。
LOG 日志服务
将日志服务收集的日志导入OSS时，需要创建一个以日志服务为受信服务的角色，日志服务将扮演该角色来将数据写入您的OSS中。
ApiGateway API网关服务
在将函数服务设置为API网关的后端服务时，您需要创建一个以API网关服务为受信服务的角色，API网关服务将扮演该角色去调用您的函数服务
上一步
创建角色
1：选择角色类型
2：填写类型信息
3：配置角色基本信息
4：创建成功
* 角色名称：
apigatewayAccessFC
长度为1-64个字符，允许英文字母、数字，或"-"
备注：
API网关访问FunctionCompute
上一步
创建

2) 绑定角色授权策略


该授权策略能访问账号下所有的Function。如果想限制授权范围，可以按照下面步骤新创建自定义策略，然后绑定。

3）创建自定义策略。在“策略管理”页面，点击“新建授权策略”。


图中策略内容为：

```
{
"Version": "1",
"Statement": [
{
"Effect": "Allow",
"Action": [
"fc:InvokeFunction"
],
"Resource": [
"acs:fc:*:1227466664334133:services/apitest/functions/*"
]
}
]
}
```

其中的Resource代表要授权的资源，需要更改为自己账号下的资源。

> 当限定到某个具体的Function时，格式为"acs:fc:*:[AccountId]:services/[ServiceName]/functions/[FunctionName]".
>
> 当要授权某个账号下的所有Function时，可以为"acs:fc:*:[AccountId]:services/*"

注意:当授权策略为单个Function时，后续更改Function，还需要重新修改该策略或者新增策略。

## 2.3 配置函数计算作为API后端服务

在API编辑中，在选择API后端服务时，选择“FunctionCompute”.


- 区域：函数计算对应的区域，当API和函数计算区域相同时，API通过内网调用函数计算，当区域不一样时，则走公网。
- RoleArn：因为最终是由API网关去调用函数计算，为跨服务调用，因此需要用户授权后，API网关才能成功调用。RoleArn为授权角色上的Arn. 可到RAM控制台角色详情中查看。

# 3 API网关和函数计算对接的格式要求

## 3.1 API网关给函数计算的输入格式

当以函数计算作为API网关的后端服务时，API网关会把请求参数通过一个固定的Map结构传给函数计算的入参event，函数计算通过如下结构去获取需要的参数，然后进行处理，该结构如下：

```
{
"path":"api request path",
"httpMethod":"request method name",
"headers":{all headers,including system headers},
"queryParameters":{query parameters},
"pathParameters":{path parameters},
"body":"string of request payload",
"isBase64Encoded":"true|false, indicate if the body is Base64-encode"
}
```

需要特别说明的是：当isBase64Encoded=true时，表明API网关传给函数计算的body内容是经过Base64编码的，函数计算需要先对body内容进行Base64解码后再处理。反之，isBase64Encoded=false时，表明API网关没有对body内容进行Base64编码。

## 3.2 函数计算给API网关的输出格式

同时，函数计算需要将输出内容通过如下JSON格式返回给API网关，方便API网关解析。

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

说明：

1) 当body内容为二进制时，需在函数计算中对body内容进行Base64编码，同时设置isBase64Encoded=true。如果body无需Base64编码，isBase64Encoded可以设置为false。API网关会对isBase64Encoded=true的body内容进行Base64解码后再透出给客户端。

2) 在nodejs版本的函数计算中，callback需要根据不同的情况进行设置。 当返回一个成功的请求，callback{null,{ “statusCode” :200,” body” :” …” }}。需要抛一个异常时，callback{new Error( ‘internal server error’ ),null}。当返回一个客户端错误 callback{null,{ “statusCode” :400,” body” :” param error” }}。

3) 如果函数计算返回不符合上面的格式要求，API网关将返回503 Service Unavailable给客户端。

## 3.3 示例

如在函数计算中配置一个demo程序：

这是一个撞墙式返回程序的代码示例，会原装返回您所有的请求内容。

```
module.exports.handler = function(event, context, callback) {
var responseCode = 200;
console.log("request: " + JSON.stringify(event.toString()));
//将event转化为JSON对象
event=JSON.parse(event.toString());
var isBase64Encoded=false;
//根据用户输入的statusCode返回，可用于测试不同statusCode的情况
if (event.queryParameters !== null && event.queryParameters !== undefined) {
if (event.queryParameters.httpStatus !== undefined && event.queryParameters.httpStatus !== null &&
event.queryParameters.httpStatus !== "") {
console.log("Received http status: " + event.queryParameters.httpStatus);
responseCode = event.queryParameters.httpStatus;
}
}
//如果body是Base64编码的，FC中需要对body内容进行解码
if(event.body!==null&&event.body!==undefined){
if(event.isBase64Encoded!==null&&event.isBase64Encoded!==undefined&&event.isBase64Encoded){
event.body=new Buffer(event.body,'base64').toString();
}
}
//input是API网关给FC的输入内容
var responseBody = {
message: "Hello World!",
input: event
};
```

```
//对body内容进行Base64编码，可根据需要处理
var base64EncodeStr=new Buffer(JSON.stringify(responseBody)).toString('base64');

//FC给API网关返回的格式，须如下所示。isBase64Encoded根据body是否Base64编码情况设置
var response = {
isBase64Encoded:true,
statusCode: responseCode,
headers: {
"x-custom-header" : "header value"
},
body: base64EncodeStr
};
console.log("response: " + JSON.stringify(response));
callback(null, response);
};
```

以POST form形式请求如下一个API的url, path为：

```
/fc/test/invoke/[type]。


POST http://test.alicloudapi.com/fc/test/invoke/test?param1=aaa&param2=bbb

"X-Ca-Signature-Headers":"X-Ca-Timestamp,X-Ca-Version,X-Ca-Key,X-Ca-Stage",
"X-Ca-Signature":"TnoBIdxxRHrFferGIzzkGcQsaezK+ZzySloKqCOsv2U=",
"X-Ca-Stage":"RELEASE",
"X-Ca-Timestamp":"1496652763510",
"Content-Type":"application/x-www-form-urlencoded; charset=utf-8",
"X-Ca-Version":"1",
"User-Agent":"Apache-HttpClient\/4.1.2 (java 1.6)",
"Host":"test.alicloudapi.com",
"X-Ca-Key":"testKey",
"Date":"Mon, 05 Jun 2017 08:52:43 GMT","Accept":"application/json",
"headerParam":"testHeader"

{"bodyParam":"testBody"}
```

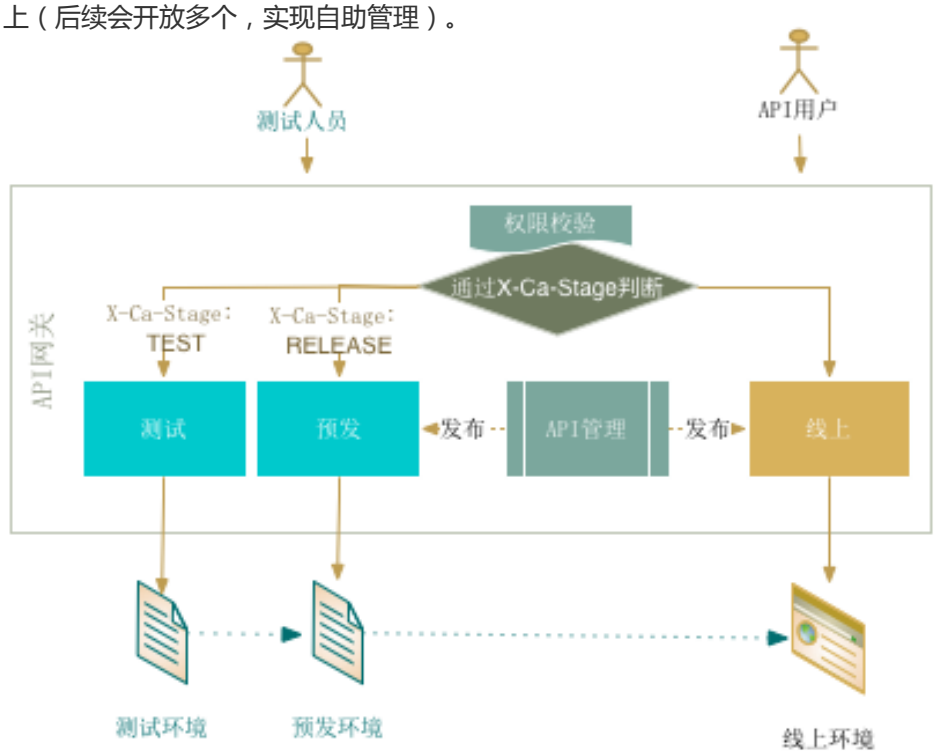
API网关返回内容：

```
200
Date: Mon, 05 Jun 2017 08:52:43 GMT
Content-Type: application/json; charset=UTF-8
Content-Length: 429
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET,POST,PUT,DELETE,HEAD,OPTIONS，PATCH
Access-Control-Allow-Headers: X-Requested-With, X-Sequence,X-Ca-Key,X-Ca-Secret,X-Ca-Version,X-Ca-
Timestamp,X-Ca-Nonce,X-Ca-API-Key,X-Ca-Stage,X-Ca-Client-DeviceId,X-Ca-Client-AppId,X-Ca-Signature,X-Ca-
Signature-Headers,X-Forwarded-For,X-Ca-Date,X-Ca-Request-Mode,Authorization,Content-Type,Accept,Accept-
Ranges,Cache-Control,Range,Content-MD5
Access-Control-Max-Age: 172800
X-Ca-Request-Id: 16E9D4B5-3A1C-445A-BEF1-4AD8E31434EC
x-custom-header: header value
```

{"message":"Hello World!","input":{"body":"{\"bodyParam\":\"testBody\"}","headers":{"X-Ca-Api-Gateway":"16E9D4B5-3A1C-445A-BEF1-4AD8E31434EC","headerParam":"testHeader","X-Forwarded-For":"100.81.146.152","Content-Type":"application/x-www-form-urlencoded; charset=UTF-8"},"httpMethod":"POST","isBase64Encoded":false,"path":"/fc/test/invoke/test","pathParameters":{"type":"test"},"queryParameters":{"param1":"aaa","param2":"bbb"}}}

# 4. 多环境

可以通过API网关的环境管理功能来实现测试环境的管理。目前每个API分组可以有三个环境：测试、预发和线上（后续会开放多个，实现自助管理）。




API网关，为避免用户测试、线上不停变化后端地址，增加环境级变量参数的来实现请求的自动路由。

环境级变量参数，即在每个环境中可以自定义公共常量参数。当用户发API调用时，可以在放置请求任意位置，传递给后端服务，以实现网关对请求的路由。

API网关将适配您请求中的环境参数信息来区分请求环境。

## 4.1 配置方法

### 4.1.1 定义环境变量

1)您可以在【开放API】-【API分组】菜单找到环境配置按钮：

2)在测试、预发、线上环境，分别新增一个变量。

Name：是变量的名称，可以自行定义，但要**保持三个环境中都有相同的变量名称**。

小提示：您如果有多个API，建议Name标识有实际意义，以便后续查询。

Value：对应你在函数服务中的Service或者Function名称。

注意：请按您实际的名称填写，否则可能造成无法调用。

下面以Service为例介绍：

比如：我有个函数服务，测试、预发、线上的名称分别为：TestServiceD、PreServiceD、ServiceD。

那么我需要测试、预发、线上分别定义一个Service的变量，并填写相应Value


函数名称，也可以同理录入，您可以录入一个叫Function的环境变量。

## 4.1.2 定义API

在定义API是可以在Service、Function的位置用“#”隔开，输入您环境变量的Name，如图：


**注意：使用多环境后，将暂时无法使用“调试”功能**

## 4.2 调用多环境API

发布您的API后，您即可发起API调用。

### 4.1 正式环境

直接发起您的API调用，即调用测试环境。

### 4.2 预发环境

您只需要在调用API时，在Header中增加入参X-Ca-Stage: RELEASE 即可访问预发环境的API。

### 4.2 测试环境

您只需要在调用API时，在Header中增加入参X-Ca-Stage: TEST 即可访问测试环境的API。

# 5. FAQ

为什么我无法录入我已有的Function？

函数计算服务由很严格的权限控制机制，所以您必须授权API网关可以使用，因此请在确认您的Function存在，并且可用的情况下，检查一下授权关系是否存在。

我是否可以将多个Function作为API的后端服务？

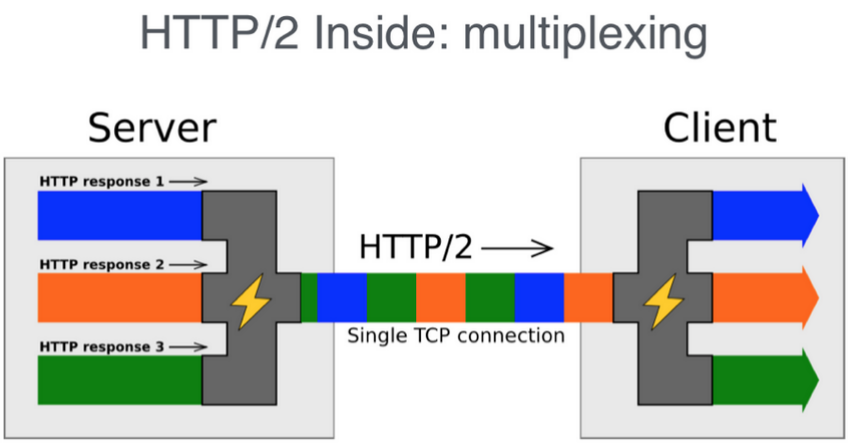
不可以，目前API和Function是一对一的关系存在。

# 支持HTTP2.0

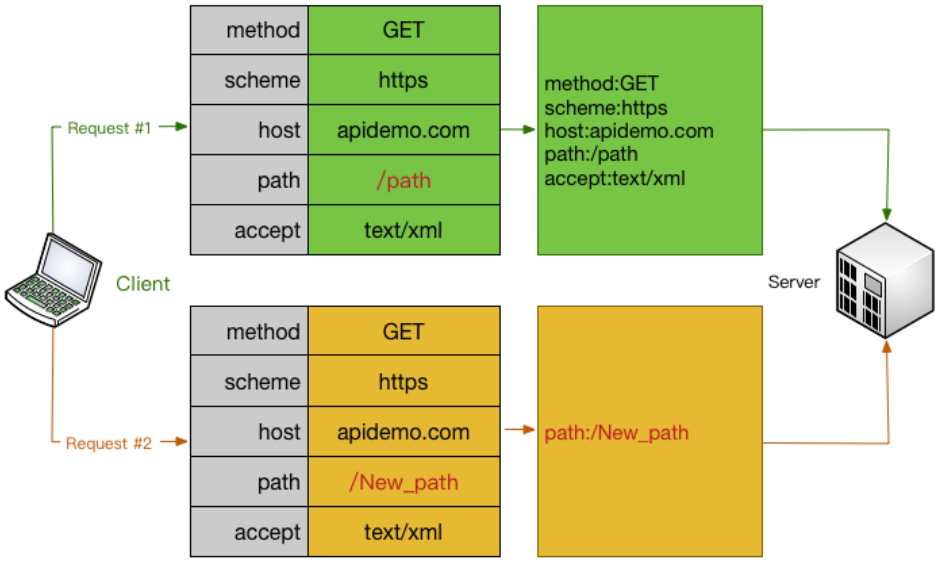
# API网关支持HTTP2.0

API网关支持HTTP2.0新特性，支持API请求多路复用、支持请求头压缩。

- 多路复用（MultiPlexing）：消除了 HTTP 1.x 中并行处理和发送请求及响应时对多个连接的依赖。可客户端和服务器可以把HTTP消息分解为互不依赖的帧，然后乱序发送，最后再在另一端把它们重新组合起来。从而避免不必要的延迟，提升效率，在请求量比较大的场景，客户端也可以轻松使用少量连接完成大量请求数据的传输。




- header压缩：如上文中所言，HTTP1.x 的header带有大量信息，而且每次都要重复发送。HTTP 2.0 使在客户端和服务器端使用“首部表”来跟踪和存储之前发送的键值对，对于相同的数据，不再通过每次请求和响应发送；“首部表”在 HTTP 2.0 的连接存续期内始终存在，由客户端和服务器共同渐进地更新；每个新的首部键值对要么追加到当前表的末尾，要么替换表中之前的值。从而减少每次请求的数据量。




## 如何开启HTTP 2.0？

新建的API分组（2017年7月14日以后）

HTTPS的API都可以使用HTTP2协议进行客户端和API网关的通信。（由于 HTTP 2.0 只许在HTTPS下运行，所以需要您开启 HTTPS方可启用 HTTP 2.0）

存量API分组

您需稍做等待，后续将提供功能手动开启。

# 支持 HTTPS

HTTPS在HTTP的基础上加入了SSL协议，对信息、数据加密，用来保证数据传输的安全。现如今被广泛使用。

API网关也支持使用HTTPS对您的API请求进行加密。可以控制到API级别，即您可以强制您的API只支持HTTP、HTTPS或者两者均支持。

## 如果您的API需要支持HTTPS，以下为操作流程：

### 步骤1:准备

您需要准备如下材料：

- 一个自有可控域名。

  为这个域名申请一个SSL证书

  SSL证书会包含两部分内容:XXXXX.key、XXXXX.pem，可以使用文本编辑器打开

  示例：

  KEY

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEA8GjIleJ7rlo86mtbwcDnUfqzTQAm4b3zZEo1aKsfAuwcvCud
....
-----END RSA PRIVATE KEY-----
```

  PEM

```
-----BEGIN CERTIFICATE-----
MIIFtDCCBJygAwIBAgIQRgWF1j00cozRl1pZ+ultKTANBgkqhkiG9w0BAQsFADBP
...
-----END CERTIFICATE-----
```

## 步骤2:绑定SSL证书

准备好以上材料，需要进行如下操作进行，登陆API网关管理控制台【开放API】-【分组管理】，单击您需要绑定SSL证书的分组，查看分组详情

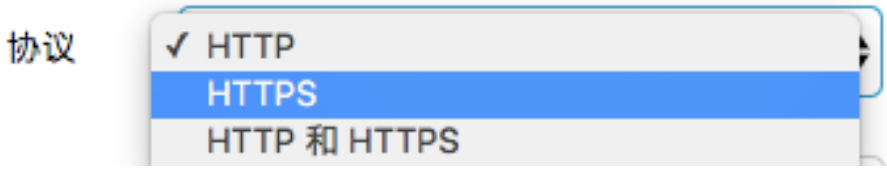
在绑定SSL证书，您首先需要您在API分组上绑定【独立域名】


【独立域名】-添加SSL证书


- 证书名称：用户自定义名称，以供后续识别
- 证书内容：证书的完整内容，需要复制XXXXX.pem 中的全部内容
- 私钥：证书的私钥，需要复制XXXXX.key中的内容。点击【确定】后，完成SSL证书的绑定。

## 步骤3：API配置调整

绑定SSL证书后，您可以按API控制不同的访问方式，支持HTTP、HTTPS、HTTP和HTTPS三种，出于安全考虑，建议全部配置成HTTPS。



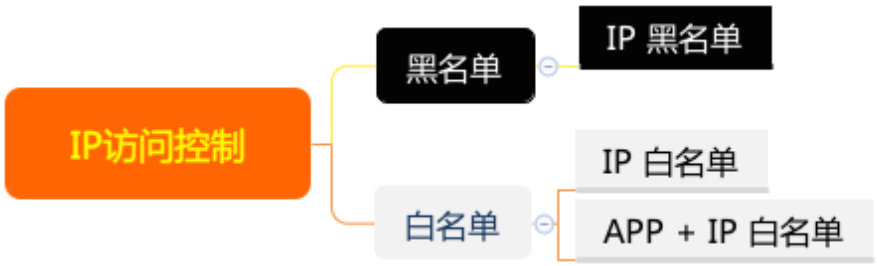

可以在【开放API】-【API列表】找到相应API，【API定义】-编辑-【请求基础定义】中进行修改。

**API支持的协议包括：**

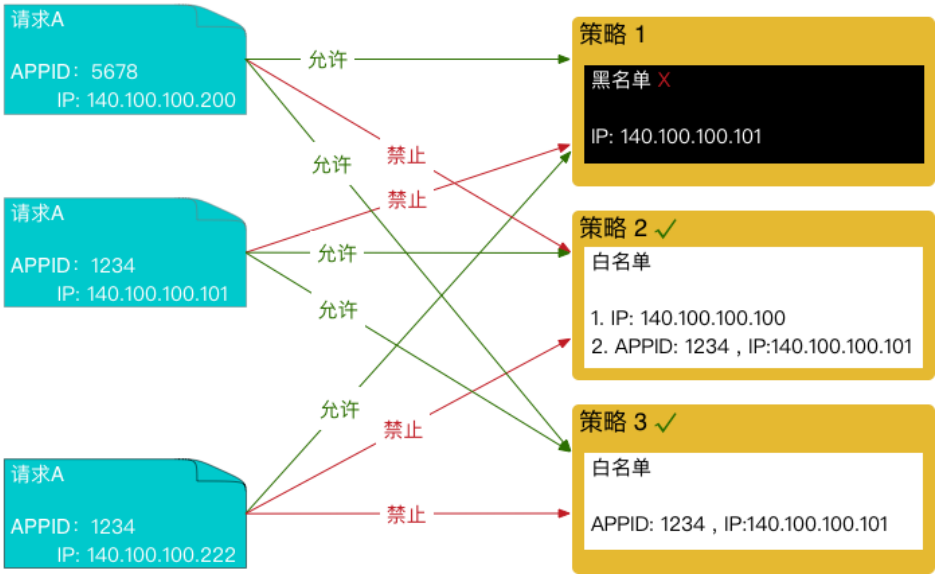
- HTTP：只允许HTTP访问，不允许HTTPS
- HTTPS：只允许HTTPS访问，不允许HTTP
- HTTP和HTTPS：两者均可调整后，API支持HTTPS协议配置完成。您的API将支持HTTPS访问。

# IP访问控制

IP 访问控制是 API 网关提供的 API 安全防护组件之一，负责控制 API 的调用来源 IP （支持IP段）。您可以通过配置某个 API 的 IP 白名单/黑名单来允许/拒绝某个来源的API请求。




- 白名单，支持配置 IP 或者 APPID + IP 的白名单访问，不在白名单列表的请求将会被拒绝。
  - IP 白名单，只允许设定的调用来源 IP 的请求被允许。
  - APPID + IP 此APPID只能在设定的 IP 下访问，其他 IP 来源将被拒绝。
- 黑名单，您可以配置 IP 黑名单，黑名单中 IP 的访问将被 API 网关拒绝。




# 使用方法

## 添加流程

您需要先创建IP访问控制策略，并绑定到需要控制的API上即可完成设置。

# 创建IP访问控制

打开API网关控制台-【开放API】-【IP访问控制】

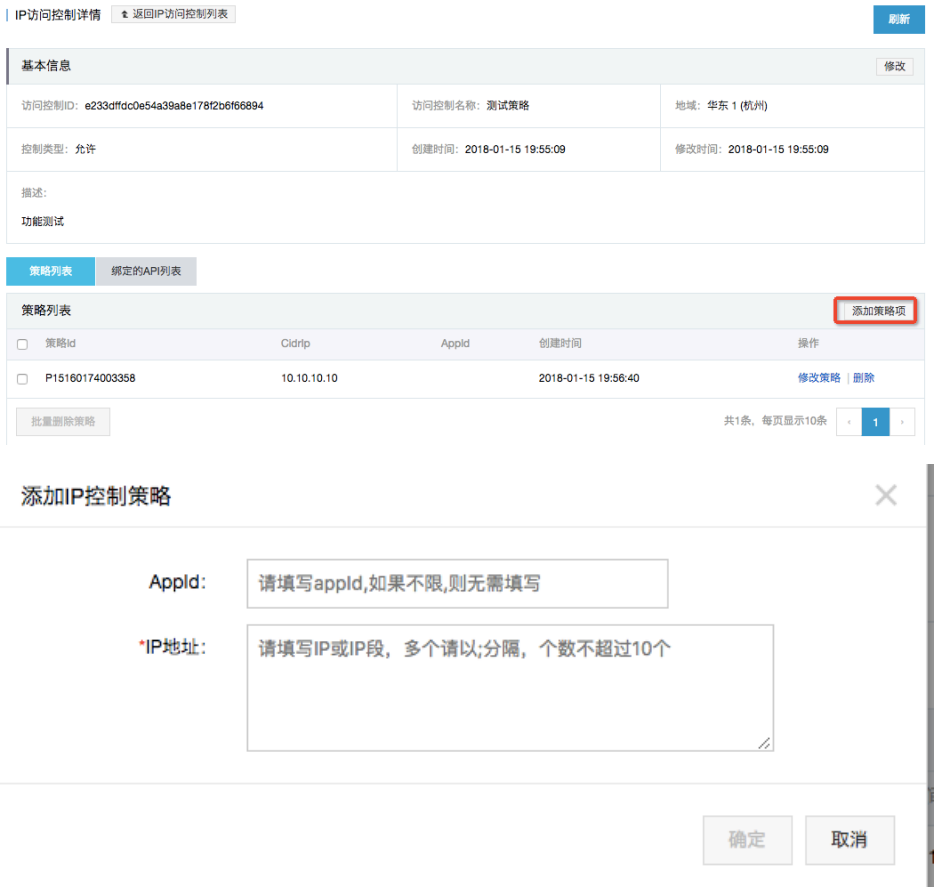

点击“创建访问控制”，弹出IP访问控制创建窗口：


按要求填写相应信息，并确定后完成创建。

- 允许：白名单
- 拒绝：黑名单

## 添加策略

创建后需要填写相应控制策略，白名单可以填写APP、IP 或者APPP+ IP，黑名单，可以填写 IP



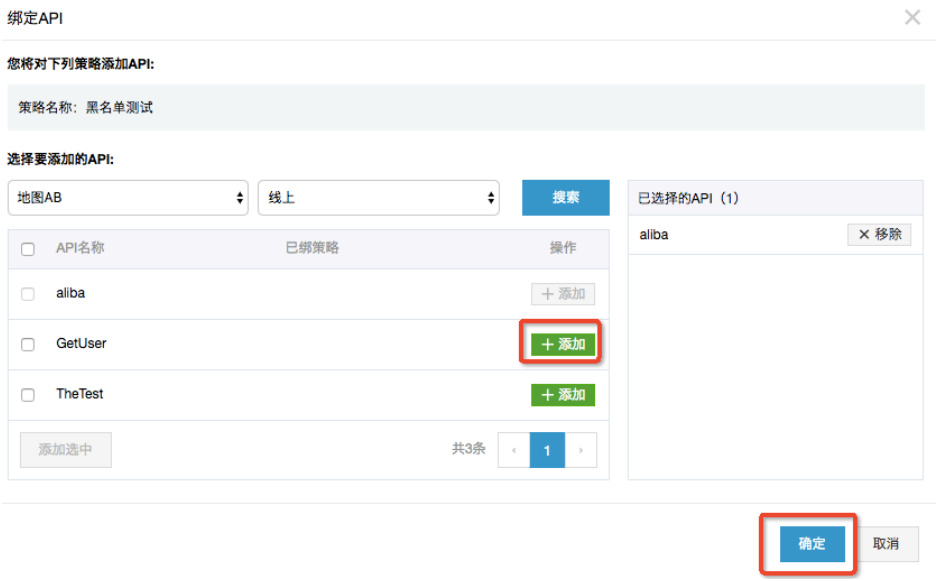

确定后完成策略添加。

## 绑定API

IP 访问控制需要绑定到API之后才能真正发挥作用。

在 IP 访问控制列表页面：


找到您所需要的策略，点击“绑定API”：




按要求选择相应API即可完成绑定。

*注：无论是黑名单还是白名单，在一个API只能绑定一条访问控制。*

## 删除IP访问策略

您只需要在 IP 访问控制列表页面，删除相应策略即可。

*注：已绑定API的访问控制，需要先解绑API再做删除。*

## 查询绑定的API

您可以在 IP 访问控制详情页面，查询此访问控制的策略即可。

# FAQ

1. 绑定/删除 IP 访问控制策略何时生效？API网关采用实时策略，绑定即生效，无延迟。
2. 一个API不同环境是否可以绑定不同的 IP 访问控制？是的，您可以针对不同环境绑定相应 IP 访问控制。建议您的测试环境、预发环境、绑定您的指定 IP ，来保护您测试环境的安全。
3. 为什么不设计 APP 黑名单？API调用需要进行APP授权，若想禁止某个APP调用，删除其授权即可，无需配置黑名单。