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。

创建 API

创建 API 即录入 API 的定义，需要录入 API 的基本信息、服务信息、请求信息、和返回信息，然后对创建的 API 进行调试，及进行安全配置。经测试证明 API 可用后，可发布上线供用户使用。

定义 API

在 API 网关控制台中 **API 列表** 页面，单击 **创建 API**，即进入 API 的创建和定义流程。

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、和描述。

- **API 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）。
- **API 名称**：API 名称标识需在所属分组内具有唯一性。

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** 、**OpenID Connect & 阿里云APP** 和 **无认证** 四种认证方式。

- **阿里云APP**：要求请求者调用该 API 时，需通过对 APP 的身份认证。
- **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。详细介绍请参见 OpenID Connect。
- **OpenID Connect & 阿里云APP**：同时进行OpenID Connect和阿里云APP认证。
- **无认证**：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，均会将请求转发至您的后端服务。（强烈不建议使用此模式。）

API 类型：分为 **公开** 和 **私有** 两种。

- **公开** 类型的 API：所有用户均可以在 API 网关控制台，**已发布 API** 页面看到 API 的部分信息。**公开** 类型的 API 会跟随 API 分组上架到云市场，供用户购买和调用。

    - **私有** 类型的 API：不能上架到云市场中售卖。如果有用户想要调用您的私有类型的 API，需要您主动操作授权，否则用户无渠道获取 API 信息。
- **描述**： API 功能描述。

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、请求Path、HTTP Method、入参请求模式、和入参定义。

- **协议**：支持 HTTP 和 HTTPS 协议。
- **请求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 中接收参数，而是映射为在 Query、Header 等位置接收。在 开放 API 接入 API 网关 中，有详细的举例说明和操作截图展示。
- **HTTP Method**：支持标准的 HTTP Method，可选择 PUT、GET、POST、DELETE、PATCH、或 HEAD。
- **入参请求模式**：API 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。
    - **入参映射**：即 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。使用入参映射模式的特点：
        - 定义方式：定义此类 API 时，需添加前后端参数映射关系。
        - 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 网关规范化陈旧的系统接口。
        - 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 用户在 **Query** 中传入参数, 后端从 **Header** 里接收等。支持：参数名称转换和参数位置转换。
            - 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。支持：参数长度校验、参数数值大小校验、参数正则校验、和参数 JSON Schema 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：
        - 无法实现参数校验。
        - 无法生成详细的 API 调用文件。
        - 自动生成的 SDK 不包含请求入参。
- **入参定义**：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、和 Parameter Path。

      注意：如果您在 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、和 Boolean。
    - **必填**：指此参数是否为必填。当选择为 **是** 时，API 网关会校验用户的请求中是否包含了此

参数。若不包含此参数，则拒绝该请求。
- **默认值**：当 **必填** 为 **否** 时生效。在用户请求中不包含此参数时，API 网关自动添加默认值发送给后端服务。
- **示例**：指参数的填写示例。
- **描述**：参数的用途描述及使用注意事项等。
- **参数校验规则**：单击 **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 Schema等。API 网关会参照校验规则对请求做初步校验。如果入参不合法，则不会发送给您的后端服务，以降低后端服务的压力。

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，即 API 后端服务的配置，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。用户请求到达 API 网关后，API 网关会根据您的后端配置，映射为对应的后端服务的请求形式，请求后端服务。

- **后端服务类型**：目前支持 HTTP/HTTPS和函数计算两种类型。
  - HTTP/HTTPS：若您的服务为 HTTP/HTTPS 服务，则选择此项。

    注意：若是 HTTPS 服务，后端服务必须有 SSL 证书。

  - 函数计算：若选择函数计算为后端服务，需先在 **函数计算控制台** 中创建函数，并需填入函数服务名称和函数名称，和获取函数计算角色 Arn。
- **VPC 通道**：当您的后端服务在 VPC 中时，需要选择 **使用 VPC 通道**。使用方法，请参见 **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，必须包含 http:// 或 https:// 。
- **后端请求 Path**：Path 是您的 API 服务在您后端服务器上的实际请求路径。若您后端 Path 需要接收动态参数，则需要声明调用者需传入参数的具体位置和参数名，即声明映射关系。
- **后端超时时间**：指 API 请求到达 API 网关后，API 网关调用 API 后端服务的响应时间，即由 API 网关请求后端服务开始直至 API 网关收到后端返回结果的时长。单位为毫秒。设置值不能超过 30 秒。如果响应时间超过该值，API 网关会放弃请求后端服务，并给用户返回相应的错误信息。
- **常量参数**：您可以配置常量参数。您配置常量参数对您的用户不可见，但是 API 网关会在中转请求时，将这些参数加入到请求中的指定位置，再传递至后端服务，实现您的后端的一些业务需求。比如，您需要 API 网关每次向后端服务发送请求都带有一个关键词 **aligateway**，您就可以把 **aligateway** 配置为常量参数，并指定接收的位置。

系统参数：指 API 网关的系统参数。默认系统参数不会传递给您，但是如果您需要获取系统参数，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：您需确保您录入的所有参数的参数名称全局唯一，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。如果您同时在 Headers 和 Query 里各有一个名为 **name** 的参数，将会导致错误。

第四部分 定义返回结果

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。

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。

API 创建、定义完成后，页面自动跳转到 **API 列表** 页。您可以通过此页面按钮，测试创建的 API 是否可用，请求链路是否正确。

1. 单击 API 名称或 **管理** 按钮，进入 **API 定义** 页面。
2. 单击左侧导航栏中 **调试 API**。
3. 输入请求参数，单击 **发送请求**。返回结果将显示在右侧页面。如果调试返回成功结果，则说明该 API 可以使用。如果返回代码为 4XX 或 5XX，则表示存在错误。请参见 **如何获取错误信息** 和 **错误代码表**。

后续步骤

完成以上定义后和初步调试后，您就完成了 API 的创建。您可以发布 API 到测试、预发、线上环境，继续调试或供用户使用。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。

插件

概述

在2019年发布的API网关新版本中，原有功能: 流量控制、IP访问控制、后端签名、JWT(OpenId Connect)的功能被统一到了插件体系中。新增功能: CORS(跨域资源访问)，Caching可以通过插件来配置。未来会有越来越多的插件加入到API网关产品中来，API的编辑界面也会变得更加的简洁，

目前插件仅在以下Region可用, 其余Region也将会近期推出, 敬请期待。

- 英国(eu-west-1)
- 迪拜(me-west-1)
- 美东(us-east-1)
- 美西(us-west-1)
- 张家口(cn-zhangjiakou)
- 呼和浩特(cn-huhehaote)

插件使用规则

- 一个API只能绑定一个相同类型的插件
- 插件仅与Region相关，可以绑定至本Region的API, 每个用户的插件限额为500个
- 插件策略和API分别是独立管理的，将插件绑定到API的指定环境后，插件策略才会对已绑定的API起作用。
- 必须要发布API后才可将插件绑定至API对应发布的环境
- 插件的绑定、解绑、更新会实时生效，不需要重新发布API，对于风险比较高的API，请先在测试API上测试通过。
- API的下线操作不影响插件的绑定关系，再次发布后仍然会带有下线前绑定的插件
- 如果插件上有已发布或者发布过但未删除的API, 则插件无法执行删除操作。

支持插件列表

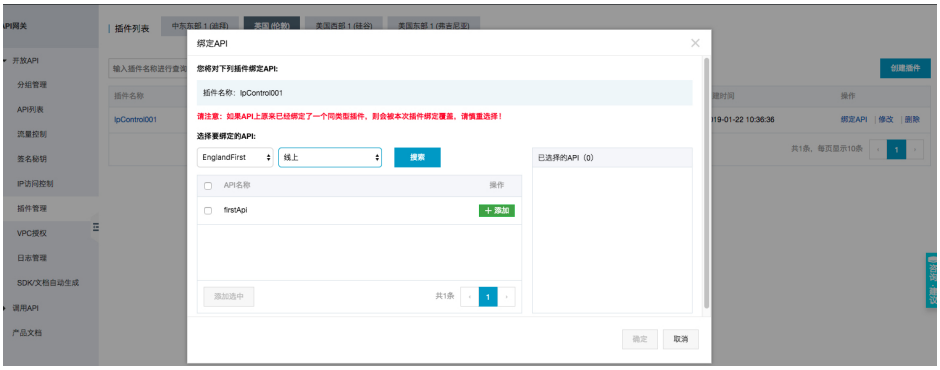
目前API网关支持下列6种插件,请点击链接查看:

- 流量控制
- IP访问控制
- 后端签名
- JWT授权(OpenID Connect)
- CORS跨域资源访问)
- Caching(缓存)

快速使用

- 访问API网关插件控制台创建插件

- 通过插件控制台将插件绑定至已发布的API当中



- 绑定后插件即可生效

开发者指南(OpenAPI)

插件管理相关的OPENAPI如下

- 创建插件:CreatePlugin
- 修改插件:ModifyPlugin
- 删除插件:DeletePlugin
- 查询插件:DescribePlugins
- 绑定API插件:AttachPlugin
- 解绑API插件:DetachPlugin
- 查询插件绑定的APIs:DescribePluginApis
- 查询API绑定的插件:DescirbePluginsByApi

流量控制

流量控制现在合并进了插件体系。现存的流量控制界面与接口仍然可以使用，流量控制策略与流量控制插件属

于同一种插件类型，受到同类型插件绑定的限制。

使用原有的流量控制接口或控制台创建或更改流量控制，会同步数据至插件系统，但不会反向同步。

使用说明

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。

在一个流控策略插件里面，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。

其余操作与限制，请参考插件总体介绍, 开发指南指南OPENAPI

插件配置

可以选择json或者yaml格式的来配置您的插件，两种格式的schema相同，可以搜索yaml to json转换工具来进行配置格式的转换，yaml格式的模板见下表

```
---
unit: SECOND # 控制区间, 取值: SECOND, MINIUTE, HOUR, DAY
apiDefault: 1000 # 允许的总流量值
userDefault: 30 # (可选)每个用户的默认流量最大值, 0表示不进行限制, 不能大于总流量值
appDefault: 30 # (可选)每个APP允许的流量最大值, 0表示不进行限制, 不能大于总流量值
specials: # (可选)特殊流控, 支持"APP"和"USER"两种特殊维度
- type: "APP" # 针对不同应用(AK)进行的流控
policies:
- key: 10123123 # AppId, AppId的取值请在API网关控制台->应用管理->应用详情处查看
value: 10 # 特殊流控值, 不能大于总流量值
- key: 10123123 # AppId控
value: 10 # 特殊流控值, 不能大于总流量值
- type: "USER" # 针对不同的阿里云账户进行的流控
policies:
- key: 123455 # 阿里云账号ID, 可点击阿里云控制台左上角查看账号ID
value: 100 # 特殊流控值, 不能大于总流量值
```

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。

IP访问控制现在合并进了插件体系。现存的控制台界面与接口仍然可以使用，IP访问控制策略与IP访问控制插件属于同一种插件类型，受到同类型插件绑定的限制。

使用原有的IP访问控制接口或控制台创建或更改IP访问控制，会同步数据至插件系统，但不会反向同步。

如何使用

支持白名单或黑名单方式

- **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。

插件配置

可以选择json或者yaml格式的来配置您的插件，两种格式的schema相同，可以搜索yaml to json转换工具来进行配置格式的转换，yaml格式的模板见下表

```
---
type: ALLOW # 控制模式，支持白名单模式'ALLOW'和黑名单模式'REFUSE'
items:
- blocks: # IP地址段
- 78.11.12.2 # 使用IP地址方式配置
- 61.3.9.0/24 # 使用CIDR方式配置
appId: 219810 # (可选) 如果配置了appId则该条目仅对该AppId生效
- blocks: # IP地址段
- 79.11.12.2 # 使用IP地址方式配置
```

后端签名插件

什么是后端签名

后端签名(原名: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。

原签名密钥功能现在合并进了插件体系。现存的控制台界面与接口仍然可以使用，原签名密钥功能与后端签名插件属于同一种插件类型，受到同类型插件绑定的限制。

使用原签名密钥功能接口或控制台创建或更改，会同步数据至插件系统，但不会反向同步。

如何使用

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

当您想替换某个API的密钥时，请直接修改要修改插件中的key和secret, 已绑定的API将会即时生效。

插件配置

可以选择json或者yaml格式的来配置您的插件，两种格式的schema相同，可以搜索yaml to json转换工具来进行配置格式的转换，yaml格式的模板见下表

```
---
key: "SampleKey"
secret: "SampleSecret"
```

JWT认证插件

Json Web Token(RFC7519), 是一种非常方便的用于身份认证的选择，API 网关可以通过托管用户的Public JWK实现对请求JWT进行验签处理方便用户进行开发。

原有在API上配置的OpenIdConnect功能目前可以通过JWT认证插件实现了, 如果你配置了JWT认证插件并绑定到了已配置OpenIdConnect功能的API上，JWT认证插件的功能将会覆盖掉原有OpenIdConnect的功能。

如何使用

- 配置JWT认证插件首先需要生成Json Web Key，您可以通过自行生成，或搜索Json Web Key Generator寻找可用的在线生成工具。一个可用的Json Web Key大概如下所示，

```
{
"kty": "RSA",
"e": "AQAB",
```

```
"kid": "O9fpdhrViq2zaaaBEWZITz",
"use": "sig",
"alg": "RS256",
"n": "qSVxcknOm0uCq5vGsOmaorPDzHUubBmZZ4UXj-9do7w9X1uKFXAnqfto4TepSNuYU2bA_-tzSLAGBsR-
BqvT6w9SjxakeiyQpVmexxnDw5WZwpWenUAcYrfSPEoNU-0hAQwFYgqZwJQMN8ptxkd0170PFauwACOx4Hfr-
9FPGy8NCoIO4MfLXzJ3mJ7xqgIZp3NIOGXz-
GIAbCf13ii7kSStpYqN3L_zzpvXUAos1FJ9IPXRV84tIZpFVh2lmRh0h8ImK-vI42dwlD_hOIzayL1Xno2R0T-
d5AwTSdnep7g-Fwu8-sj4cCRWq3bd61Zs2QOJ8iustH0vSRMYdP5oYQ"
}
```

*这里展示的是json格式，当使用yaml格式配置插件，需要转换**

- JWT认证插件只需要配置Public Key, 请妥善保存好您的Private Key，目前仅支持RSA256算法
- 如果您需要配置多个Json Web Key，kid字段是必须的，否则kid是可选的
- 可以配置多个Json Web Key, jwks配置可以与jwk一起使用
- JWT认证插件会使用配置的parameter与parameterLocation来读取JWT的值，
    - 如：parameter: X-Token, parameterLocation: header表示从请求的X-Token头读取JWT
    - 如果API上配置了与paramter配置同名的参数时，paramterLocation可以忽略，否则会在调用时报错

插件配置

可以选择json或者yaml格式的来配置您的插件，两种格式的schema相同，可以搜索yaml to json转换工具来进行配置格式的转换，yaml格式的模板见下表

```
---
parameter: X-Token # 从指定的参数中获取JWT, 对应API的参数
parameterLocation: header # API为映射模式时可选, API为透传模式下必填, 用于指定JWT的读取位置, 仅支持
`query`,`header`
claimParameters: # claims参数转换, 网关会将jwt claims映射为后端参数
- claimName: aud # claim名称,支持公共和私有
parameterName: X-Aud # 映射后参数名称
location: header # 映射后参数位置, 支持`query,header,path,formData`
- claimName: userId # claim名称,支持公共和私有
parameterName: userId # 映射后参数名称
location: query # 映射后的参数位置, 支持`query,header,path,formData`
preventJtiReplay: false # 是否开启针对`jti`的防重放检查, 默认: false
#
# `Json Web Key`的`Public Key`
jwk:
kty: RSA
e: AQAB
use: sig
kid: O8fpdhrViq2zaaaBEWZITz # 在只配置一个JWK时,kid是可选的,但如果中JWT包含了kid,网关会校验kid的一致性
alg: RS256
n: qSVxcknOm0uCq5vGsOmaorPDzHUubBmZZ4UXj-9do7w9X1uKFXAnqfto4TepSNuYU2bA_-tzSLAGBsR-
BqvT6w9SjxakeiyQpVmexxnDw5WZwpWenUAcYrfSPEoNU-0hAQwFYgqZwJQMN8ptxkd0170PFauwACOx4Hfr-
9FPGy8NCoIO4MfLXzJ3mJ7xqgIZp3NIOGXz-
GIAbCf13ii7kSStpYqN3L_zzpvXUAos1FJ9IPXRV84tIZpFVh2lmRh0h8ImK-vI42dwlD_hOIzayL1Xno2R0T-
d5AwTSdnep7g-Fwu8-sj4cCRWq3bd61Zs2QOJ8iustH0vSRMYdP5oYQ
#
# 可以支持配置多个JWK(最多5个), 可以与jwk字段一起配置
```

```
# 配置多个JWK时,kid是必选的,如果JWT中没有提供kid,则校验失败
jwks:
- kid: O9fpdhrViq2zaaaBEWZITz # 在只配置一个JWK时,kid是可选的,但如果中JWT包含了kid,网关会校验kid的一致性
kty: RSA
e: AQAB
use: sig
alg: RS256
n: qSVxcknOm0uCq5vGsOmaorPDzHUubBmZZ4UXj-9do7w9X1uKFXAnqfto4TepSNuYU2bA_-tzSLAGBsR-
BqvT6w9SjxakeiyQpVmexxnDw5WZwpWenUAcYrfSPEoNU-0hAQwFYgqZwJQMN8ptxkd0170PFauwACOx4Hfr-
9FPGy8NCoIO4MfLXzJ3mJ7xqgIZp3NIOGXz-
GIAbCf13ii7kSStpYqN3L_zzpvXUAos1FJ9IPXRV84tIZpFVh2lmRh0h8ImK-vI42dwlD_hOIzayL1Xno2R0T-
d5AwTSdnep7g-Fwu8-sj4cCRWq3bd61Zs2QOJ8iustH0vSRMYdP5oYQ
- kid: 10fpdhrViq2zaaaBEWZITz # 在只配置一个JWK时,kid是可选的,但如果中JWT包含了kid,网关会校验kid的一致性
kty: RSA
e: AQAB
use: sig
alg: RS256
n: qSVxcknOm0uCq5vGsOmaorPDzHUubBmZZ4UXj-9do7w9X1uKFXAnqfto4TepSNuYU2bA_-tzSLAGBsR-
BqvT6w9SjxakeiyQpVmexxnDw5WZwpWenUAcYrfSPEoNU-0hAQwFYgqZwJQMN8ptxkd0170PFauwACOx4Hfr-
9FPGy8NCoIO4MfLXzJ3mJ7xqgIZp3NIOGXz-
GIAbCf13ii7kSStpYqN3L_zzpvXUAos1FJ9IPXRV84tIZpFVh2lmRh0h8ImK-vI42dwlD_hOIzayL1Xno2R0T-
d5AwTSdnep7g-Fwu8-sj4cCRWq3bd61Zs2QOJ8iustH0vSRMYdP5oYQ
```

跨域资源访问(CORS)

关于CORS跨域资源访问的基础知识可参考这篇文章

插件配置

可以选择json或者yaml格式的来配置您的插件，两种格式的schema相同，可以搜索yaml to json转换工具来进行配置格式的转换，yaml格式的模板见下表

```
---
allowOrigins: api.foo.com,api2.foo.com # 允许的ORIGIN,用逗号分隔, 默认"*"
allowMethods: GET,POST,PUT # 允许的方法,用逗号分隔
allowHeaders: X-Ca-RequestId # 允许的请求头部,用逗号分隔
exposeHeaders: X-RC1,X-RC2 # 允许暴露给XMLHttpRequest对象的头,用逗号分隔
allowCredentials: true # 是否允许Cookie
maxAge: 172800
```

缓存插件

通过缓存可以将后端返回的应答缓存在API网关服务层面，有效降低后端的负荷，增加平滑度

使用方法

- 仅缓存**GET**方法的应答
- 不缓存使用默认分组二级域名的请求, 默认分组二级域名有1000次/天的限制，仅用于测试场合
- 可在插件中加入如下的配置来区分不同的缓存
    - **varyByApp**: 针对不同的App区分缓存
    - **varyByParameters**: 针对不同的参数取值区分缓存,从绑定的API取同名的参数
    - **varyByHeaders**: 针对不同的请求头区分缓存，如Accept,Accept-Language
- 现在用户在每个Region可以使用的缓存空间为5M(按用户区分), 使用超期释放策略, 缓存满了后不缓存后续的应答。
- 网关遵守后端应答中的Cache-Control头的约定来处理缓存，如果后端不返回Cache-Control头，则默认缓存，使用插件中配置的duration字段作为缓存超期时间。
- 最长允许的超期时间为48小时(172800秒), 超过这个时间的配置无效，被视为48小时超期
- 网关默认不处理客户端的Cache-Control头，可以通过clientCacheControl来进行配置，mode取值为
    - off: 忽略所有客户端请求中的Cache-Control头
    - all: 处理对所有客户端的请求中的Cache-Control头
    - app: 仅处理AppId在apps配置列表中的请求的Cache-Control头
- 对于应答的Header, 默认情况网关仅缓存Content-Type头, 如果你需要更多的Headers, 可使用配置中的cacheableHeaders配置添加

插件配置

可以选择json或者yaml格式的来配置您的插件，两种格式的schema相同，可以搜索yaml to json转换工具来进行配置格式的转换，yaml格式的模板见下表

```
---
varyByApp: false # 是否按照访问方的AppId来区分缓存内容, 默认为false
varyByParameters: # 按照不同的参数取值来区分缓存内容
- userId # 参数名称，如果后端参数映射为不同的名称，这里请填写映射后的参数名称
varyByHeaders: # 是否按照不同的请求Header值来区分缓存内容
- Accept # `Accept`表示按照不同的Accept头来区分缓存
clientCacheControl: # 允许客户端通过`Cache-Control`头来影响缓存策略, 默认`off`
mode: app # off: 拒绝所有端, all: 允许所有端, apps: 允许appId取值在`apps`列表中的端
apps: # 如果mode配置为`app`, 则允许`AppId`在以下列表中的取值
- 1992323 # 消费者AppId, 注意不是AppKey
- 1239922 # 消费者AppId, 注意不是AppKey
cacheableHeaders: # 允许缓存的后端Headers, 默认只缓存`Content-Type`和`Content-Length`
- X-Customer-Token # 允许缓存的Header名称
duration: 3600 # 默认超期的秒数
```

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。
- 配置可参考文章 API网关监控配置

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100个
用户创建API数量限制	每个API分组下，最多可以创建1000个API。即每个账号最多可创建100*1000 = 100000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。

API网关 OpenID Connect 使用指南

阿里云API网关在OpenID Connect协议的基础上实现了一套基于用户体系对用户的API进行授权访问的机制，满足用户个性化安全设置的需求。

1. OpenID Connect简介

OpenID Connect是2014年初发布的开放标准，定义了一种基于OAuth2的可互操作的方式来来提供用户身份认证。它使用简单的REST/JSON消息流来实现，和之前任何一种身份认证协议相比，开发者可以轻松集成。OpenID Connect使用 API 进行身份交互的框架，允许客户端根据授权服务器的认证结果最终确认用户的身份，以及获取基本的用户信息；支持包括Web、移动、JavaScript在内的所有客户端类型；它是可扩展的协议，允许你使用某些可选功能，如身份数据加密、OpenID提供商发现、会话管理等。

1.1 OpenID Connect 角色介绍

1. 客户端：直接为终端用户提供服务
2. 认证服务：OpenID 提供方，通常是一个 OpenID 认证服务器，它能为第三方颁发用于认证的ID Token
3. 业务服务：提供业务服务，比如查询用户资料
4. 终端用户：指持有资源拥有人

1.2 OpenID Connect 流程描述

1. 客户端发送认证请求给认证服务；
2. 终端用户在认证页面进行授权确认（可选）；
3. 认证服务对认证请求进行验证，发送ID Token给客户端；
4. 客户端向业务请求，请求中携带ID Token；
5. 业务服务验证ID Token是否合法后返回业务应答；

这个流程的第二步是可选的，如果终端用户在客户端输入了用户名和密码，第一步中的认证请求中携带了用户名和密码，那么认证服务在验证了用户名和密码后，省去第二步，可以直接在应答中返ID Token。这种模式更加简洁，阿里云的API网关的OpenID Connect验证模式就是这种，本文将在第二节仔细描述。

1.3 JWT(JSON Web Token)格式数据

认证服务返回的ID Token需要严格遵守JWT(JSON Web Token)的定义，下面是JWT(JSON Web Token)的定义细节：

1. iss：必须。Issuer Identifier，认证服务的唯一标识，一个区分大小写的https URL，不包含query和fragment组件
2. sub：必须。Subject Identifier，iss提供的终端用户的标识，在iss范围内唯一，最长为255个ASCII个字符，区分大小写
3. aud：必须。Audience(s)，标识ID Token的受众，必须包含OAuth2的client_id，分大小写的字符串数组
4. exp：必须。Expiration time，超过此时间的ID Token会作废
5. iat：必须。Issued At Time，JWT的构建的时间
6. auth_time：AuthenticationTime，终端用户完成认证的时间。
7. nonce：发送认证请求的时候提供的随机字符串，用来减缓重放攻击，也可以用来关联客户端

Session。如果nonce存在，客户端必须验证nonce
8. acr：可选。Authentication Context Class Reference，表示一个认证上下文引用值，可以用来标识认证上下文类
9. amr：可选。Authentication Methods References，表示一组认证方法
10. azp：可选。Authorized party，结合aud使用。只有在被认证的一方和受众（aud）不一致时才使用此值，一般情况下很少使用

下面是一个典型ID Token的示例，供参考

```
{
"iss": "https://1.2.3.4:8443/auth/realms/kubernetes",
"sub": "547cea22-fc8a-4315-bdf2-6c92592a6e7c",
"aud": "kubernetes",
"exp": 1525158711,
"iat": 1525158411,
"auth_time": 0,
"nonce": "n-0S6_WzA2Mj",
"acr": "1",
"azp": "kubernetes",
"nbf": 0,
"typ": "ID",
"session_state": "150df80e-92a1-4b0c-a5c5-8c858eb5a848",
"userId": "123456",
"preferred_username": "theone",
"given_name": "the",
"family_name": "one",
"email": "theone@mycorp.com"
}
```

关于ID Token的更详细的定义请参考： https://openid.net/specs/openid-connect-core-1_0.html

2. API网关OpenID Connect业务流程

阿里云API网关将OpenID Connect作为用户自主授权API的一种认证模式。作为处于客户端和后端服务中间的API网关，可以为后端服务去验证ID Token的合法性，并将没有拿到后端服务颁发的合法ID Token的请求在API网关层面给拒绝了。

2.1 准备条件

用户需要在API网关上配置以下两点，才能正常使用API网关的OpenID Connect认证模式：

1. 配置一个获取授权API，并且在这个API上指定一个公钥；
2. 所有需要保护的业务API设置成为OpenID Connect的业务API；

具体的配置方法参见第三章。

2.2 业务流程

上图是API网关处理OpenID Connect的的整个业务流程，下面我们用文字来详细描述下：

1. 客户端向API网关发起认证请求，请求中一般会携带终端用户的用户名和密码；
2. API网关将请求直接转发给后端服务；
3. 后端服务读取请求中的验证信息（比如用户名、密码）进行验证，验证通过后使用私钥生成标准的ID Token，返回给API网关；
4. API网关将携带ID Token的应答返回给客户端，客户端需要将这个ID Token缓存到本地；
5. 客户端向API网关发送业务请求，请求中携带ID Token；
6. API网关使用用户设定的公钥对请求中的ID Token进行验证，验证通过后，将请求透传给后端服务；
7. 后端服务进行业务处理后应答；
8. API网关将业务应答返回给客户端。

在这个整个过程中，API网关利用OpenID Connect的认证机制，实现了用户使用自己的用户体系对自己API进行授权的能力。

2.3 授权范围与时效

API网关会认为用户颁发的ID Token有权利访问整个分组下的所有OpenID Connect的业务API。如果需要更细力度的权限管理，还需要后端服务自己解开ID Token进行权限认证。API网关会验证ID Token中的exp字段，一旦这个字段过期了，API网关会认为这个ID Token无效而将请求直接打回。过期时间这个值必须设置，并且过期时间一定要小于7天。

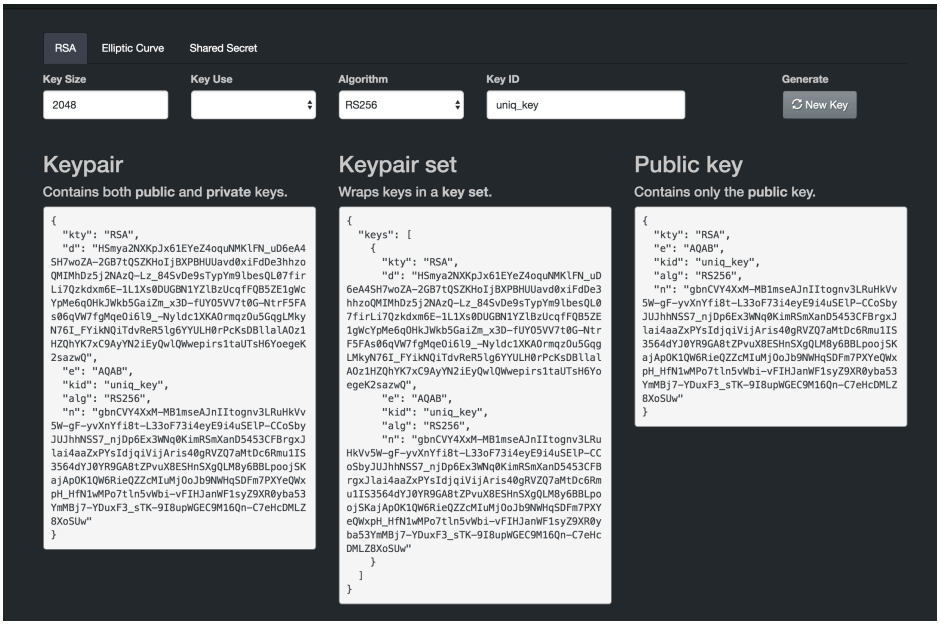
3. 在API配置OpenID Connect

本章主要教大家如何在API网关配置OpenID Connect，主要有三步：

1. 生成一个公私秘钥对；
2. 配置一个授权API，在这个API上将公钥配置好；
3. 所有业务API安全认证类型设置为OpenID Connect模式。下面是具体细节。

3.1 准备公钥与私钥

用户可以在 这个站点https://mkjwk.org 生成用于ID Token生成与验证的公钥与私钥，目前API网关支持的秘钥对的加密算法为RSA SHA256，秘钥对的加密的位数为2048。



在这个网站生成的公钥用于本文3.1中设置授权API：

```
{
"kty": "RSA",
"e": "AQAB",
"kid": "uniq_key",
"alg": "RS256",
"n": "gbnCVY4XxM-MB1mseAJnIItognv3LRuHkVv5W-gF-yvXnYfi8t-L33oF73i4eyE9i4uSElP-
CCoSbyJUJhhNSS7_njDp6Ex3WNq0KimRSmXanD5453CFBrgxJlai4aaZxPYsIdjqiVijAris40gRVZQ7aMtDc6Rmu1IS3564
dYJ0YR9GA8tZPvuX8ESHnSXgQLM8y6BBLpoojSKajApOK1QW6RieQZZcMIuMjOoJb9NWHqSDFm7PXYeQWxpH_Hf
N1wMPo7tln5vWbi-vFIHJanWF1syZ9XR0yba53YmMBj7-YDuxF3_sTK-9I8upWGEC9M16Qn-C7eHcDMLZ8XoSUw"
}
```

在这个网站生成的秘钥对用于本文4章中生成IdToken编程时使用：

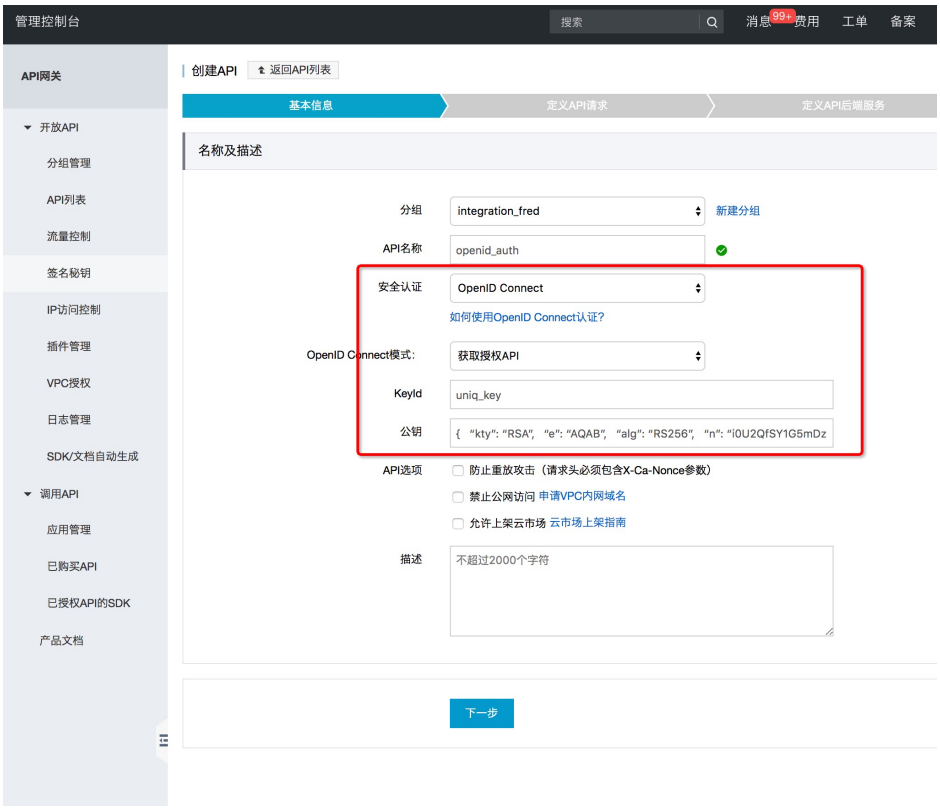
```
{
"kty": "RSA",
"d": "HSmya2NXKpJx61EYeZ4oquNMKlFN_uD6eA4SH7woZA-
2GB7tQSZKHoIjBXPBHUUavd0xiFdDe3hhzoQMIMhDz5j2NAzQ-Lz_84SvDe9sTypYm9lbesQL07firLi7Qzkdxm6E-
1L1Xs0DUGBN1YZlBzUcqfFQB5ZE1gWcYpMe6qOHkJWkb5GaiZm_x3D-fUYO5VV7t0G-
NtrF5FAs06qVW7fgMqeOi6l9_-
Nyldc1XKAOrmqzOu5GqgLMkyN76I_FYikNQiTdvReR5lg6YYULH0rPcKsDBllalAOz1HZQhYK7xC9AyYN2iEyQwlQWwe
pirs1taUTsH6YoegeK2sazwQ",
"e": "AQAB",
"kid": "uniq_key",
"alg": "RS256",
"n": "gbnCVY4XxM-MB1mseAJnIItognv3LRuHkVv5W-gF-yvXnYfi8t-L33oF73i4eyE9i4uSElP-
CCoSbyJUJhhNSS7_njDp6Ex3WNq0KimRSmXanD5453CFBrgxJlai4aaZxPYsIdjqiVijAris40gRVZQ7aMtDc6Rmu1IS3564
```

dYJ0YR9GA8tZPvuX8ESHnSXgQLM8y6BBLpoojSKajApOK1QW6RieQZZcMIuMjOoJb9NWHqSDFm7PXYeQWxpH_HfN1wMPo7tln5vWbi-vFIHJanWF1syZ9XR0yba53YmMBj7-YDuxF3_sTK-9I8upWGEC9M16Qn-C7eHcDMLZ8XoSUw"
}

3.2 配置授权API和业务API

3.2.1 授权API

下图是配置授权API的第一页，也是最重要的一页，除了这一页意外，授权API的其他配置均和普通的API配置相同。

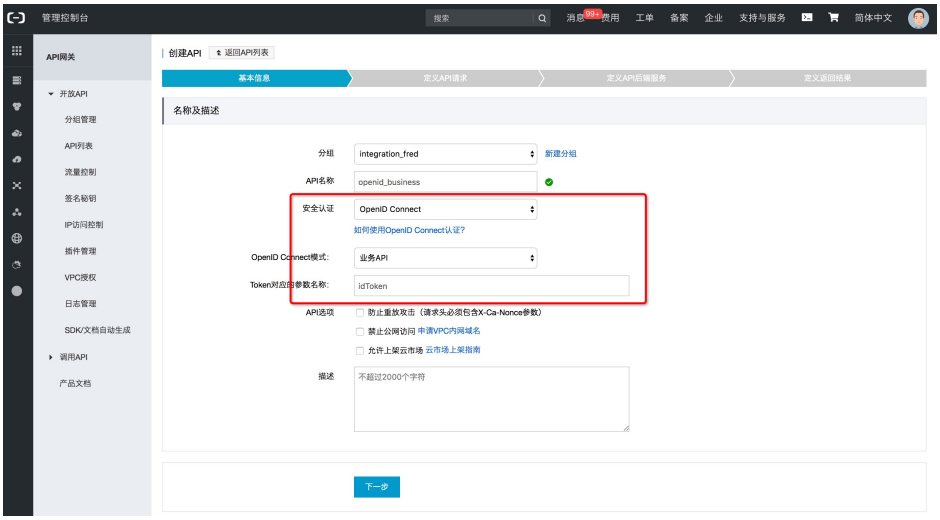


这一页中标注出来的注意项如下：

1. 安全认证：选择OpenID Connect或者 OpenID Connect & 阿里云APP，这两者的区别是，后者除了验证OpenID Connect的ID Token之外，还需要验证阿里云自己的APP，签名，具体签名算法参见 https://help.aliyun.com/document_detail/29475.html
2. OpenID Connect 模式：选择获取授权API；
3. KeyId，填写一个分组范围内唯一的秘钥名称，尽量使用英文加数字；
4. 公钥，填写刚才生成的公钥，是整个JSON串都需要填入。

3.2.2 业务API

下图是配置OpenID Connect 业务API的第一页，也是最重要的一页，除了这一页意外，OpenID Connect 业务API的其他配置均和普通的API配置相同。



1. 安全认证：选择OpenID Connect或者 OpenID Connect && 阿里云APP，这两者的区别是，后者除了验证OpenID Connect的ID Token之外，还需要验证阿里云自己的APP，签名，具体签名算法参见 https://help.aliyun.com/document_detail/29475.html
2. OpenID Connect 模式：选择业务API；
3. Token对应的参数名称，填写请求中对应的ID Token的参数名称，比如idToken；

3.3 通过API插件配置

API网关还可以通过插件方式配置OpenID Connect，具体请参考文档：
https://help.aliyun.com/document_detail/103228.html

4. 后端服务实现ID Token颁发

```
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;


public class GenerateJwtDemo {
public static void main(String[] args) throws JoseException {
//使用在API网关设置的keyId
String keyId = "uniq_key";
//使用本文3.2节生成的Keypare
String privateKeyJson = "{\n"
+ " \"kty\": \"RSA\",\n"
+ " \"d\": "
+
"\"O9MJSOgcjjiVMNJ4jmBAh0mRHF_TlaVva70Imghtlgwxl8BLfcf1S8ueN1PD7xV6Cnq8YenSKsfiNOhC6yZ_fjW1syn5r
```

```
aWfj68eR7cjHWjLOvKjwVY33GBPNOvspNhVAFzeqfWneRTBbga53Agb6jjN0SUcZdJgnelzz5JNdOGaLzhacjH6YPJKpb
uzCQYPkWtoZHDqWTzCSb4mJ3n0NRTsWy7Pm8LwG_Fd3pACl7JIY38IanPQDLoighFfo-
Lriv5z3IdlhwbPnx0tk9sBwQBTRdZ8JkqqYkxUiB06phwr7mAnKEpQJ6HvhZBQ1cCnYZ_nIlrX9-I7qomrlE1UoQ\",\n"
+ " \"e\": \"AQAB\",\n"
+ " \"kid\": \"myJwtKey\",\n"
+ " \"alg\": \"RS256\",\n"
+ " \"n\": \"vCuB8MgwPZfziMSytEbBoOEwxsG7XI3MaVMoocziP4SjzU4IuWuE_DodbOHQwb_thUru57_Efe"
+
"--
sfATHEa0Odv5ny3QbByqsvjyeHk6ZE4mSAV9BsHYa6GWAgEZtnDceeeDc0y76utXK2XHhC1Pysi2KG8KAzqDa099Yh7s
31AyoueoMnrYTmWfEyDsQL_OAIiwgXakkS5U8QyXmWicCwXntDzkIMh8MjfPskesyli0XQD1AmCXVV3h2Opm1Amx0
ggSOOiINUR5YRD6mKo49_cN-nrJWjtwSouqDdxHYP-4c7epuTcdS6kQHiQERBd1ejdpAxV4c0t0FHF7MOy9kw\"\n"
+ "}";

JwtClaims claims = new JwtClaims();
claims.setGeneratedJwtId();
claims.setIssuedAtToNow();
//过期时间一定要设置，并且小于7天
NumericDate date = NumericDate.now();
date.addSeconds(120*60);
claims.setExpirationTime(date);
claims.setNotBeforeMinutesInThePast(1);
claims.setSubject("YOUR_SUBJECT");
claims.setAudience("YOUR_AUDIENCE");
//添加自定义参数，所有值请都使用String类型
claims.setClaim("userId", "1213234");
claims.setClaim("email", "userEmail@youapp.com");

JsonWebSignature jws = new JsonWebSignature();
jws.setAlgorithmHeaderValue(AlgorithmIdentifiers.RSA_USING_SHA256);
//必须设置
jws.setKeyIdHeaderValue(keyId);
jws.setPayload(claims.toJson());
PrivateKey privateKey = new RsaJsonWebKey(JsonUtil.parseJson(privateKeyJson)).getPrivateKey();
jws.setKey(privateKey);
String jwtResult = jws.getCompactSerialization();
System.out.println("Generate Json Web token , result is " + jwtResult);
}
}
```

以上是生成ID Token的Java示例代码，其中有以下几个地方需要重点关注：

1. keyId需要三个环节都一致，且全局唯一：
    - 在本文3.1节，https://mkjwk.org 生成秘钥时填写的keyId;
    - 在本文3.2.1节设置的keyId；
    - 代码中的keyId
2. JsonWebSignature 对象的KeyIdHeaderValue必须设置；
3. privateKeyJson 使用3.1节中生成的Keypair Json字符串（三个方框内的第一个）；
4. 过期时间一定要设置，并且小于7天；
5. 添加自定义参数，所有值请都使用String类型；

5. 常见问题

5.1 APP签名认证

如果安全认证选择“OpenID Connect & 阿里云APP”时，请求中必须携带使用阿里云AK加密的签名，具体签名方式请参见文档： https://help.aliyun.com/document_detail/29475.html

5.2 Swagger导入支持OPENID CONNECT吗？

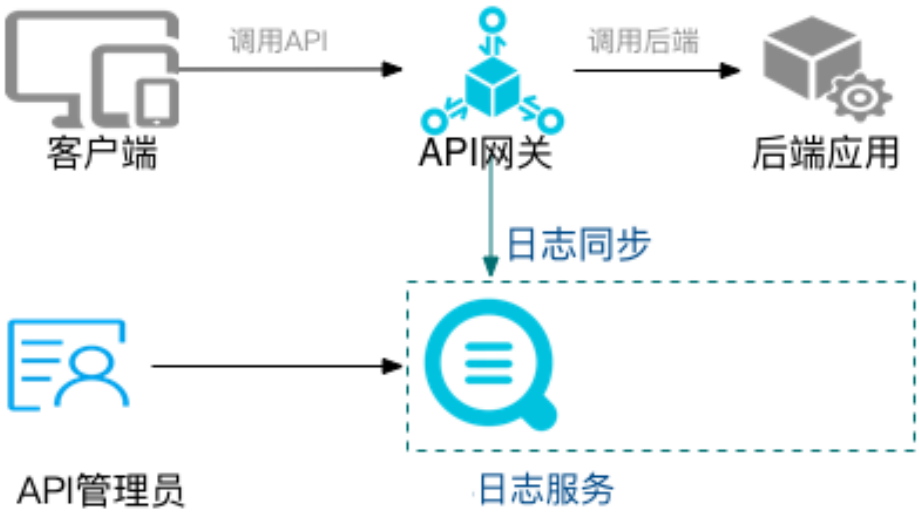
目前暂时不支持；

5.3 API网关错误应答列表

StatusCode	ErrorMessage	含义
401	“OpenId Connect Verify Fail, IdToken Not Exist”	业务APP请求中没有携带定义好的IDToken参数
401	“234, JWS set idToken exception”	解析IDToken时出现Exception
401	“234, Not found by keyId”	根据Token中的KeyId，找不到用户设置的公钥
401	“235, JWS set Public-Key exception”	根据Token中的KeyId，找到的公钥不可用
401	” 237, Verify signature failed “	验证签名失败
401	“245, IdToken is out of scope”	请求中的IDToken不属于当前分组，无权访问
401	“238, JWS get payload exception”	请求中的IDToken找不到payload
401	“239, Parse payload to JwtClaims exception”	转化payload为 JwtClaims 时出错
401	“239, idToken expired”	请求中的IDToken已经过期
401	“Invalid OpenId Connect Config”	API配置错误，获取不到OpenId Connect相关配置

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

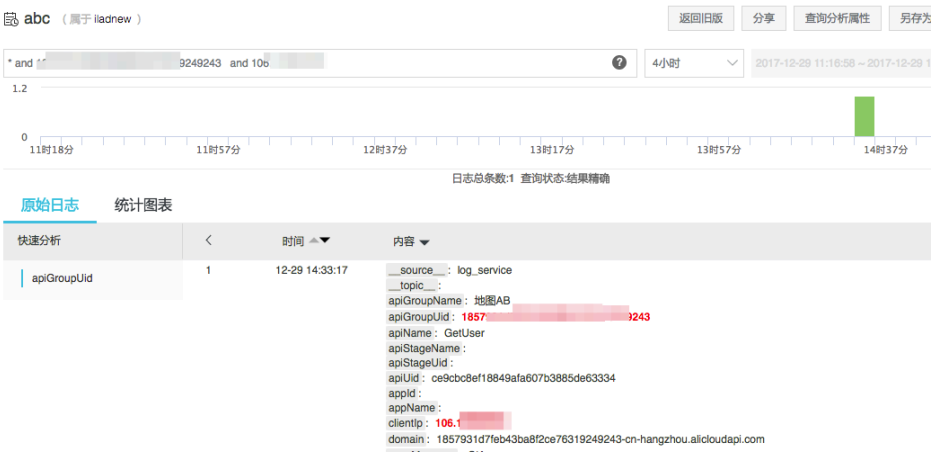
2.1.1.2 在日志服务控制台配置

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

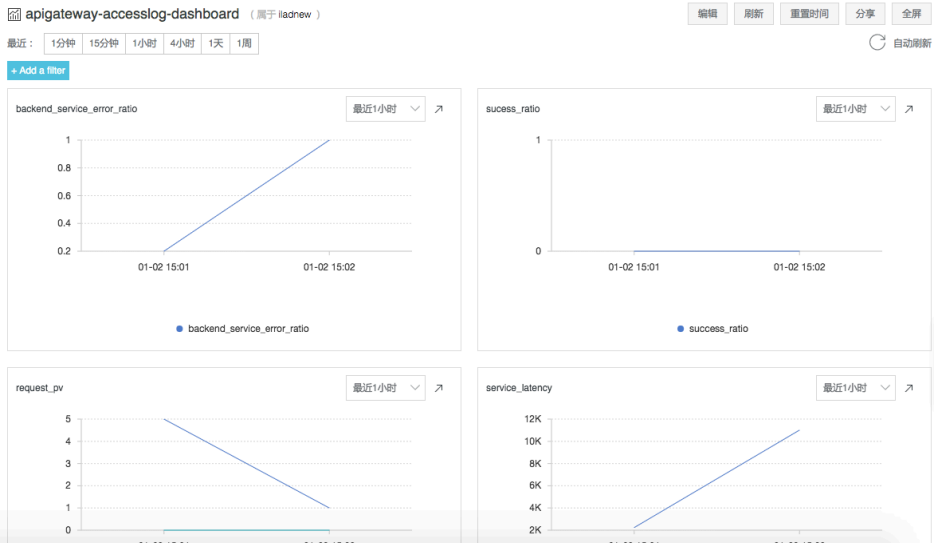
你可以点击API网关控制台-【开放API】-【日志管理】-“访问日志”，跳转到日志控制台，根据日志服务的查询语法，在线检索调用日志。如图：



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。

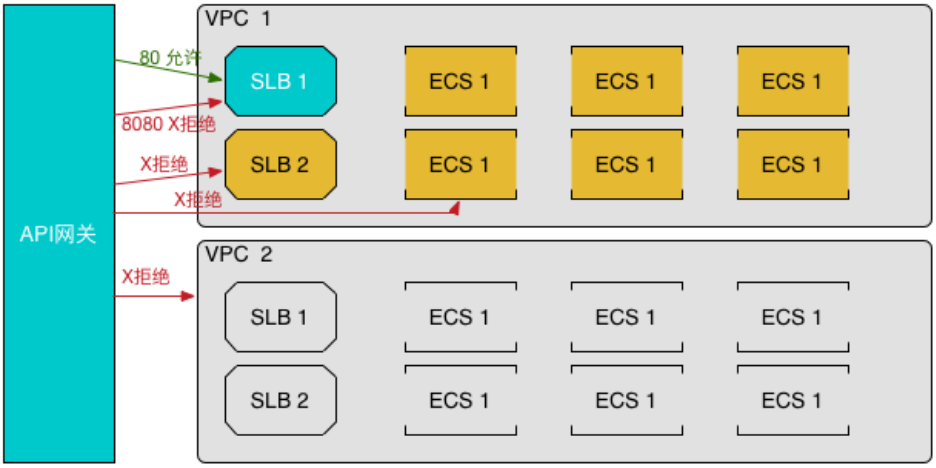
若您的后端服务在VPC环境，需要进行授权API网关访问才可开放相应API。创建API流程如下：

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
德国	内网入方向	100.104.72.0/24
亚太东南3（吉隆坡）	内网入方向	100.104.112.0/24
亚太南部1（孟买）	内网入方向	100.104.233.0/24
亚太东南5（雅加达）	内网入方向	100.104.72.0/24
亚太东北1（东京）	内网入方向	100.104.188.0/24
亚太东南2（悉尼）	内网入方向	100.104.143.192/26

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。

1. 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。

2. 填写 Mock 请求响应statusCode

支持的statusCode 取值如下表，兼容HTTP 1.1 Response Status Code的格式返回及其状态, 如果您定义的statusCode不在下表中，将提示参数无效：

http code	http message

200	OK
201	Created
202	Accepted
203	Non-Authoritative Information
204	No Content
205	Reset Content
206	Partial Content
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
306	(Unused)
307	Temporary Redirect
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Request Entity Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Satisfiable
417	Expectation Failed

450	Parameter Requried
451	Method Connect Exception
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported

3. 定义Mock Header

API网关支持自定义Mock Header, 允许Header同名，Header定义后取值不能为空，设置Mock时, Header Name不允许为空，并且只能由数字、字母、下划线、减号组成。Header Value不允许为空。

解除 Mock

若您需要解除 Mock，选择其他后端服务类型即可，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。

通过导入Swagger创建API

Swagger是一种用于描述API定义的规范，被广泛应用于定义和描述后端应用服务的API。现在，API网关支持导入Swagger 2.0的文件来创建API，您可以参考ImportSwagger，或在控制台上进行操作，入口见下图:

API网关的Swagger扩展基于Swagger 2.0，可以创建API实体的Swagger定义，并将Swagger导入API网关用于批量创建或者更新API实体。API网关的预配置是swagger2.0，它可支持并兼容大部分的Swagger规范，但存在一定的差异性（Swagger兼容性说明）。

本章节主要对基于Swagger的API网关扩展进行介绍，并提供相应的示例来阐述用法。

注：**Swagger中所有参数与取值均大小写敏感**

一、Swagger扩展：

Swagger扩展主要是对于swagger原生Operation Object进行扩展，增加认证、参数映射以及后端服务等扩展。除此之外，增加处理any方法的扩展，用于捕获任意http请求方法。所有扩展均以x-aliyun-apigateway-开头，具体扩展内容如下：

1.1 x-aliyun-apigateway-auth-type: 授权类型

授权类型，应用于Operation Object。用于指定该API的授权类型，其中：

取值范围:

- APP(默认值): 阿里云API网关APP授权
- ANONYMOUS: 匿名

示例：

```
...
paths:
'path/':
get:
x-aliyun-apigateway-auth-type: ANONYMOUS
...
```

1.3 x-aliyun-apigateway-api-market-enable: 云市场支持

授权类型，应用于Operation Object。用于指定该API是否可以上架云市场，其中：

取值范围:

- true
- false(默认)

示例：

```
...
paths:
'path/':
get:
x-aliyun-apigateway-api-market-enable: true
...
```

1.4 x-aliyun-apigateway-api-force-nonce-check: 强制NONCE校验

授权类型，应用于Operation Object。用于指定该API是否进行NONCE强制校验，其中：

取值范围:

- true
- false(默认)

示例：

```
...
paths:
'path/':
get:
x-aliyun-apigateway-api-force-nonce-check: true
...
```

1.5 x-aliyun-apigateway-paramater-handling: API映射关系

API映射关系, 应用于Operation Object，用于指定请求参数与后端服务参数的映射关系。当映射关系选择PASSTHROUGH时，Parameter Object不支持 x-aliyun-apigateway-backend-location 和 x-aliyun-apigateway-backend-name 属性。

取值范围:

- PASSTHROUGH(默认值): 入参透传
- MAPPING: 入参映射

示例：

```
...
paths:
'path/':
get:
x-aliyun-apigateway-paramater-handling: MAPPING
...
```

1.6 x-aliyun-apigateway-backend: 后端类型

后端类型, 应用于Operation Object。用于设置后端服务信息。根据后端服务类型，决定具体的属性，详情如下：

1.6.1 后端服务类型:HTTP

HTTP的后端类型用于直接配置后端服务地址，一般用于后端地址可以直接访问的场合。

属性说明:

属性名称	类型	描述

type	string	必填；值为HTTP
address	string	必填；用于标识后端服务地址
path	string	选填:用于标识后端服务路径。支持路径变量。默认与跟path值相同
method	string	必填：后端请求方法
timeout	int	选填，默认为10000。该属性取值范围为[500,30000]

示例:

```
...
x-aliyun-apigateway-backend:
type: HTTP
address: http://10.10.100.2:8000
path: "/users/{userId}"
method: GET
timeout: 7000
...
```

1.6.2 后端服务类型:HTTP-VPC

HTTP-VPC的后端类型用于后端服务在VPC网络内的场合，需要先创建VPC授权，使用VPC授权名导入。

属性说明:

属性名称	类型	描述
type	string	必填；值为：HTTP-VPC
vpcAccessName	string	必填；后端服务使用的vpc实例名称
path	string	选填:用于标识后端服务路径。支持路径变量。默认与根path相等
method	string	必填：后端请求方法
timeout	int	选填，默认为10000。该属性取值范围为[500,30000]

示例:

```
...
x-aliyun-apigateway-backend:
type: HTTP_VPC
vpcAccessName: vpcAccess1
path: "/users/{userId}"
method: GET
```

```
timeout: 10000
...
```

1.6.3 后端服务类型:FC

API网关后端服务类型为FUNCTION COMPUTE。

属性说明:

属性名称	类型	描述
type	string	必填；值为：FC
fcRegion	string	必填；函数计算所在区域
serviceName	string	必填: 函数计算服务名
functionName	string	必填：函数计算函数名
arn	string	选填，函数计算RAM授权

示例:

```
...
x-aliyun-apigateway-backend:
type: FC
fcRegion: cn-shanghai
serviceName: fcService
functionName: fcFunction
arn: acs:ram::111111111:role/aliyunapigatewayaccessingfcrole
...
```

1.6.4 后端服务类型: MOCK

MOCK的后端类型，用来模拟最初预定的返回结果。

属性说明:

属性名称	类型	描述
type	string	必填；值为:MOCK
mockResult	string	必填；mock返回结果
mockStatusCode	Integer	选填
mockHeaders	Header	选填

Header 的 说明

属性名称	类型	描述
name	string	必填
value	string	必填

示例:

```
...
x-aliyun-apigateway-backend:
type: MOCK
mockResult: mock resul sample
mockStatusCode: 200
mockHeaders:
- name: server
value: mock
- name: proxy
value: GW
...
```

1.7 x-aliyun-apigateway-constant-parameters: 常量参数

常量参数, 应用于Operation Object，用于定义后端服务的常量参数。

属性说明:

属性名称	类型	描述
backendName	string	必填；后端参数名称
value	string	必填；常量值
location	String	必填；常量参数存放位置，可选值:[query,header]
description	string	可选；用于对该常量进行说明

示例:

```
    ...
x-aliyun-apigateway-constant-parameters:
- backendName: swaggerConstant
value: swaggerConstant
location: header
description: description of swagger
...
```

1.8 x-aliyun-apigateway-system-parameters： 后端服务参数

后端服务参数, 应用于Operation Object，用于定义API后端服务的系统参数。

属性说明:

属性名称	类型	描述

systemName	string	必填；系统参数名称
backendName	string	必填；后端参数名称
location	String	必填；常量参数存放位置，可选值:[query,header]

示例:

```
    ...
x-aliyun-apigateway-system-parameters:
- systemName: CaAppId
backendName: appId
location: header
...
```

1.9 x-aliyun-apigateway-backend-location: 后端参数位置

后端参数位置, 应用于 Parameter Object。该属性仅在 x-aliyun-apigateway-paramater-handling: MAPPING设置时生效，用于设置参数映射后，在后端服务请求时的参数位置。

取值范围:

- path
- header
- query
- formData

示例:

```
    ...
parameters:
- name: swaggerHeader
in: header
required: false
type: number
format: double
minimum: 0.1
maximum: 0.5
x-aliyun-apigateway-backend-location: query
x-aliyun-apigateway-backend-name: backendQuery
...
```

1.10 x-aliyun-apigateway-backend-name: 后端参数名称

后端参数名称, 应用于Parameter Object。该属性仅在 x-aliyun-apigateway-paramater-handling: MAPPING设置时生效，用于设置参数映射后，在后端服务请求时的参数名称。

示例:

```
    ...
parameters:
- name: swaggerHeader
in: header
required: false
type: number
format: double
minimum: 0.1
maximum: 0.5
x-aliyun-apigateway-backend-location: query
x-aliyun-apigateway-backend-name: backendQuery
...
```

1.11 x-aliyun-apigateway-query-schema: query对Model支持

后端参数名称, 应用于Parameter Object。用于对Query进行模型定义。当parameter为String类型，同时定义为query参数时，可以使用。

示例:

```
    ...
parameters:
- name: event_info
in: query
required: true
type: string
x-aliyun-apigateway-query-schema:
$ref: "#/definitions/EvnetInfo"
...
```

1.12 x-aliyun-apigateway-any-method: ANY方法

ANY方法, 应用于Path Item Object，用于设置API可以接受任意类型的http请求。

示例：

```
...
paths:
'path/':
x-aliyun-apigateway-any-method:
...
...
```

二、兼容性说明

API网关与Swagger规范在定义API时存在一定差异性，包括：

2.1 Swagger参数类型与原有API网关类型的对照表

Swagger类型	API网关类型	支持的校验参数及规则
- type:integer - format:int32	Int	- mininum - maxnum
- type:integer - format:int64	Long	- mininum - maxnum
- type:number - format:float	Float	- mininum - maxnum
- type:number - format:double	Doulbe	- mininum - maxnum
- type:string	String	- maxLength - enumValues - pattern
- type:boolean - format:Boolean	Boolean	-

2.2 consumes字段支持

当Swagger配置文件存在formdata类型参数时，需要配置consumes节点。API网关现在只支持application/x-www-form-urlencoded类型。

```
consumes:
- application/x-www-form-urlencoded
```

三、Swagger示例

本章节提供三个基于API网关的swagger扩展示例。示例几乎涵盖了所有的扩展内容，为基于swagger扩展自定

义API实体提供便利。

注：示例仅供参考。

3.1 API网关后端服务为HTTP的Swagger示例

```
swagger: '2.0'
basePath: /
info:
version: '0.9'
title: Aliyun Api Gateway Swagger Sample
schemes:
- http
- https
x-aliyun-apigateway-paramater-handling: MAPPING
x-aliyun-apigateway-api-market-enable: true
x-aliyun-apigateway-api-force-nonce-check: true
x-aliyun-apigateway-backend:
type: HTTP
address: 'http://www.aliyun.com'
method: get
timeout: 10000
paths:
'/http/get/mapping/{userId}':
get:
operationId: case1
schemes:
- http
- https
x-aliyun-apigateway-paramater-handling: MAPPING
x-aliyun-apigateway-api-market-enable: true
x-aliyun-apigateway-auth-type: ANONYMOUS
parameters:
- name: userId
in: path
required: true
type: string
- name: swaggerQuery
in: query
required: false
default: '123465'
type: integer
format: int32
minimum: 0
maximum: 100
- name: swaggerHeader
in: header
required: false
type: number
format: double
minimum: 0.1
maximum: 0.5
x-aliyun-apigateway-backend-location: query
```

```
x-aliyun-apigateway-backend-name: backendQuery
x-aliyun-apigateway-constant-parameters:
- backendName: swaggerConstant
value: swaggerConstant
location: header
description: description of swagger
x-aliyun-apigateway-system-parameters:
- systemName: CaAppId
backendName: appId
location: header
responses:
'200':
description: 200 description
'400':
description: 400 description
'/echo/test/post/{userId}':
post:
operationId: testpost
schemes:
- http
- https
x-aliyun-apigateway-paramater-handling: MAPPING
x-aliyun-apigateway-backend:
type: HTTP
address: 'http://www.aliyun.com'
method: post
timeout: 10000
consumes:
- application/x-www-form-urlencoded
parameters:
- name: userId
required: true
in: path
type: string
- name: swaggerQuery1
in: query
required: false
default: '123465'
type: integer
format: int32
minimum: 0
maximum: 100
x-aliyun-apigateway-enum: 1,2,3
- name: swaggerQuery2
in: query
required: false
type: string
x-aliyun-apigateway-backend-location: header
x-aliyun-apigateway-backend-name: backendHeader
x-aliyun-apigateway-query-schema:
$ref: '#/definitions/AiGeneratePicQueryVO'
- name: swaggerHeader
in: header
required: false
type: number
format: double
```

```
minimum: 0.1
maximum: 0.5
x-aliyun-apigateway-backend-location: query
x-aliyun-apigateway-backend-name: backendQuery
- name: swaggerFormdata
in: formData
required: true
type: string
responses:
'200':
description: 200 description
schema:
$ref: '#/definitions/ResultOfGeneratePicturesVO'
'400':
description: 400 description
x-aliyun-apigateway-any-method:
operationId: case2
schemes:
- http
- https
x-aliyun-apigateway-paramater-handling: MAPPING
x-aliyun-apigateway-backend:
type: HTTP
address: 'http://www.aliyun.com'
path: '/builtin/echo/{abc}'
method: post
timeout: 10000
parameters:
- name: userId
in: path
required: false
default: '123465'
type: integer
format: int32
minimum: 0
maximum: 100
x-aliyun-apigateway-backend-name: abc
x-aliyun-apigateway-backend-location: path
responses:
'200':
description: 200 description
'400':
description: 400 description
definitions:
AiGeneratePicQueryVO:
type: object
properties:
transactionId:
type: string
description: 异步任务ID
GeneratePictureVO:
type: object
properties:
id:
type: integer
format: int64
```

```
description: 图片ID
name:
type: string
description: 图片名称
GeneratePicturesVO:
type: object
properties:
failSize:
type: integer
format: int64
description: 失败数量
list:
type: array
description: 图片列表
items:
$ref: '#/definitions/GeneratePictureVO'
title: GeneratePictureVO
successSize:
type: integer
format: int32
description: 成功数量
totalSize:
type: number
format: float
description: 请求总数
ResultOfGeneratePicturesVO:
type: object
properties:
model:
description: 返回内容
$ref: '#/definitions/GeneratePicturesVO'
title: GeneratePicturesVO
requestId:
type: string
description: 请求ID
```

3.2 API网关后端服务为HTTP-VPC的Swagger示例

```
swagger: '2.0'
basePath: /
info:
version: '0.9'
title: Aliyun Api Gateway Swagger Sample
schemes:
- http
- https
paths:
'/http/get/mapping/{userId}':
get:
operationId: case1
schemes:
- http
- https
x-aliyun-apigateway-paramater-handling: MAPPING
```

```
x-aliyun-apigateway-backend:
type: HTTP-VPC
vpcAccessName: vpcName1
path: '/builtin/echo/{userId}'
method: get
timeout: 10000
parameters:
- name: userId
in: path
required: true
type: string
- name: swaggerQuery
in: query
required: false
default: '123465'
type: integer
format: int32
minimum: 0
maximum: 100
- name: swaggerHeader
in: header
required: false
type: number
format: double
minimum: 0.1
maximum: 0.5
x-aliyun-apigateway-backend-location: query
x-aliyun-apigateway-backend-name: backendQuery
responses:
'200':
description: 200 description
'400':
description: 400 description
'/echo/test/post':
post:
operationId: testpost
schemes:
- http
- https
x-aliyun-apigateway-paramater-handling: MAPPING
x-aliyun-apigateway-backend:
type: HTTP-VPC
vpcAccessName: vpcName2
path: '/builtin/echo'
method: post
timeout: 10000
consumes:
- application/x-www-form-urlencoded
parameters:
- name: swaggerQuery1
in: query
required: false
default: '123465'
type: integer
format: int32
minimum: 0
```

```
maximum: 100
- name: swaggerQuery2
in: query
required: false
type: string
x-aliyun-apigateway-backend-location: header
x-aliyun-apigateway-backend-name: backendHeader
- name: swaggerHeader
in: header
required: false
type: number
format: double
minimum: 0.1
maximum: 0.5
x-aliyun-apigateway-backend-location: query
x-aliyun-apigateway-backend-name: backendQuery
- name: swaggerFormdata
in: formData
required: true
type: string
responses:
'200':
description: 200 description
'400':
description: 400 description
x-aliyun-apigateway-any-method:
operationId: case2
schemes:
- http
- https
x-aliyun-apigateway-paramater-handling: PASSTHROUGH
x-aliyun-apigateway-backend:
type: HTTP-VPC
vpcAccessName: vpcName3
path: '/builtin/echo'
method: post
timeout: 10000
responses:
'200':
description: 200 description
'400':
description: 400 description
```

3.3 API网关后端服务为FunctionCompute的Swagger示例

```
swagger: '2.0'
basePath: /
info:
version: '0.9'
title: Aliyun Api Gateway Swagger Sample
schemes:
- http
- https
paths:
```

```
'/http/get/mapping/{userId}':
get:
operationId: case1
schemes:
- http
- https
x-aliyun-apigateway-paramater-handling: MAPPING
x-aliyun-apigateway-backend:
type: FC
fcRegion: cn-shanghai
serviceName: fcService
functionName: fcFunction
arn: acs:ram::111111111:role/aliyunapigatewayaccessingfcrole
parameters:
- name: userId
in: path
required: true
type: string
responses:
'200':
description: 200 description
'400':
description: 400 description
```

3.4 API网关后端服务为MOCK的Swagger示例

```
swagger: '2.0'
basePath: /
info:
version: '0.9'
title: Aliyun Api Gateway Swagger Sample
schemes:
- http
paths:
'/mock/get/mapping/{userId}':
get:
operationId: case1
schemes:
- http
- https
x-aliyun-apigateway-paramater-handling: MAPPING
x-aliyun-apigateway-backend:
type: MOCK
mockResult: mock resul sample
mockStatusCode: 200
mockHeaders:
- name: server
value: mock
- name: proxy
value: GW
parameters:
- name: userId
in: path
required: true
```

```
type: string
responses:
'200':
description: 200 description
'400':
description: 400 description
```

四、特别说明

以下内容请密切关注，直接影响swagger的导入功能使用

4.1 全局域的支持

以下标签支持全局作用域的定义。当该标签原本作用域不存在该定义时，会自动填充全局域的定义值；若存在，则优先使用原本作用域的定义值。

- x-aliyun-apigateway-backend
- x-aliyun-apigateway-api-market-enable
- x-aliyun-apigateway-api-force-nonce-check
- x-aliyun-apigateway-paramater-handling
- x-aliyun-apigateway-auth-type

4.2 Swagger Definition说明

swagger导入现在支持模型定义，但是与原有swagger规范存在差异模型的定义主要用于SDK的生成，所以会在原有swagger规范的基础上增加了一些限制条件：

- swagger中schema标签仅支持：$ref类型
- swagger中Definition的model仅支持object type类型的模型定义
- swagger中Definition的model中存在array定义，其引用$ref应与title标签配合使用。array类型默认在SDK生成时对应生成ArrayList

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考下面文档：

https://help.aliyun.com/document_detail/66031.html

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

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（https://tools.ietf.org/html/rfc6455）。

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：https://help.aliyun.com/document_detail/66031.html

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：https://help.aliyun.com/document_detail/66031.html

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

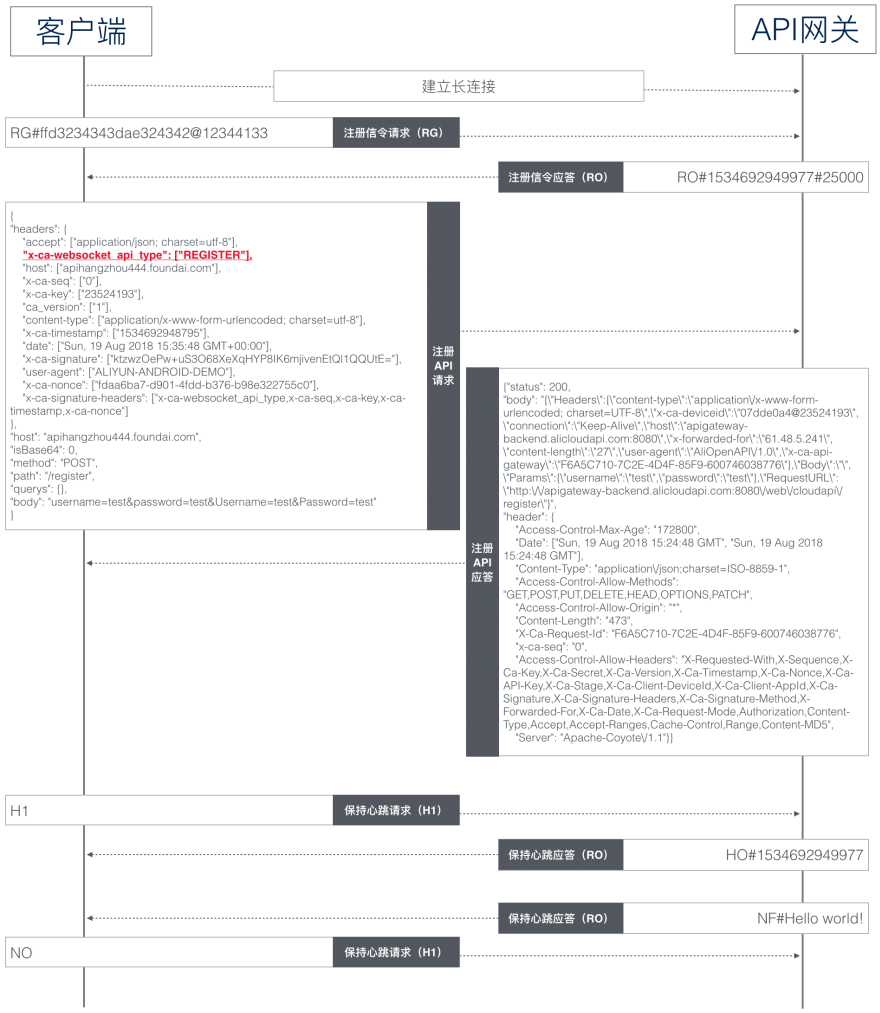
API网关有流控限制，客户端的请求QPS如果超过流控限制会触发API网关的流控，客户端会收到API网关发送

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。

三.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需要做哪些事情：



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_COMMAND_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_COMMAND_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 lost , 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 ApiCallback 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, TimeUnit.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.getWebSocketApiType().toString());


return true;
}


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.CLOUDAPI_HTTP_HEADER_CONTENT_TYPE));

if(null != apiRequest.getFormParams() && apiRequest.getFormParams().size() > 0){
webSocketApiRequest.setBody(HttpCommonUtil.buildParamString(apiRequest.getFormParams()));
}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() , Base64.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。

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

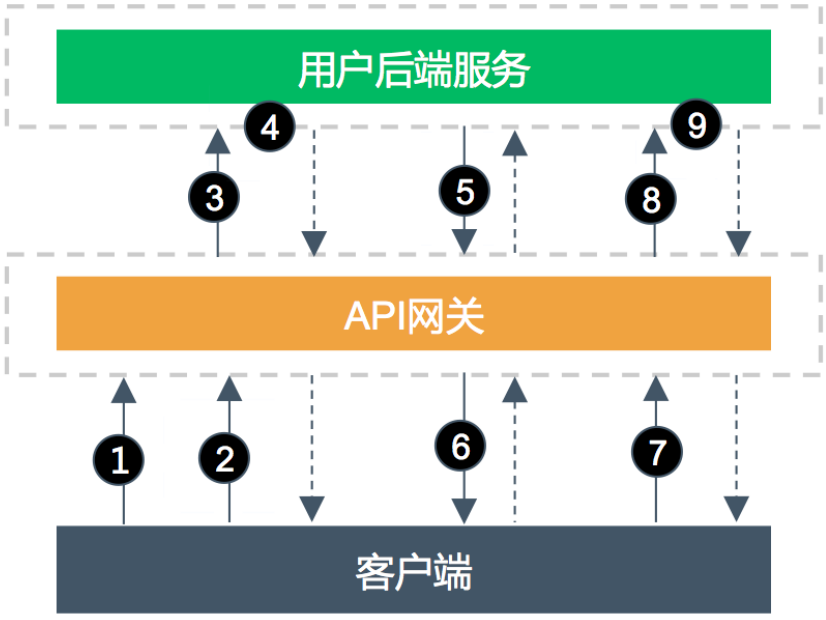
API网关已经在所有Region开放双向通信能力，双向通信能力构建于WebSocket协议之上，目前Android，Objective-C，JAVA三种SDK均支持双向通信。

（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网关目前已经在所有Region提供双向通信的能力，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

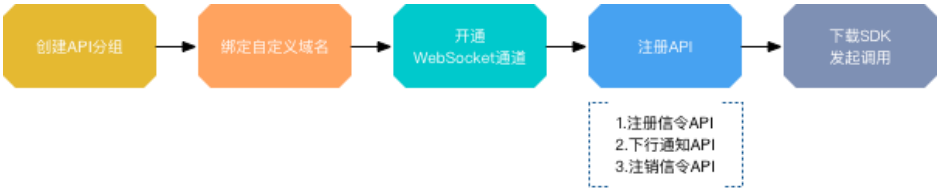
用户后端服务，在收到客户端发送的注册信令后，记住注册信令中的设备ID字段，然后就可以向API网关发送接收方为这个设备的下行通知信令了。只要这个设备在线，API网关就可以将此下行通知发送到端。

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

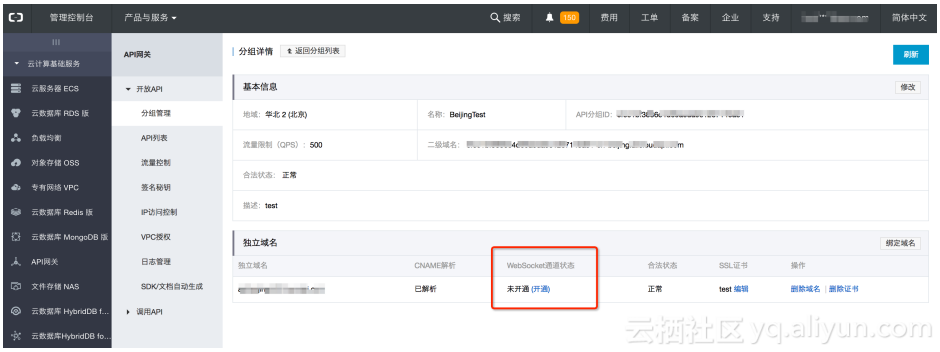
已经已经绑定了域名的情况可忽略本节。

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

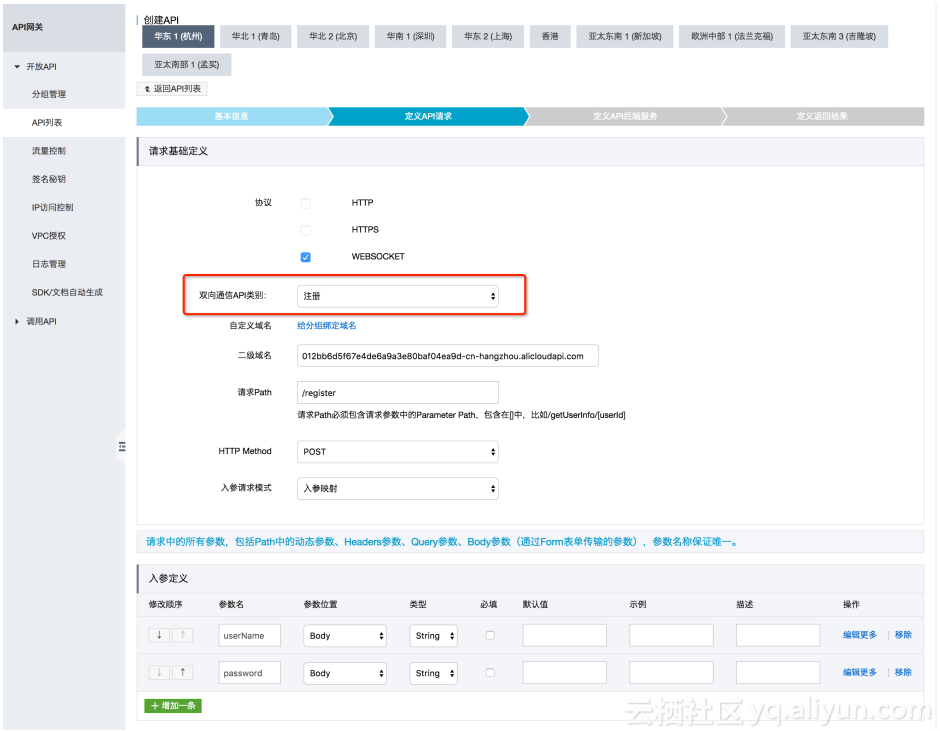
需要刚才创建的分组下创建三个API，普通API的创建流程请参见文档：
https://help.aliyun.com/document_detail/29478.html

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：

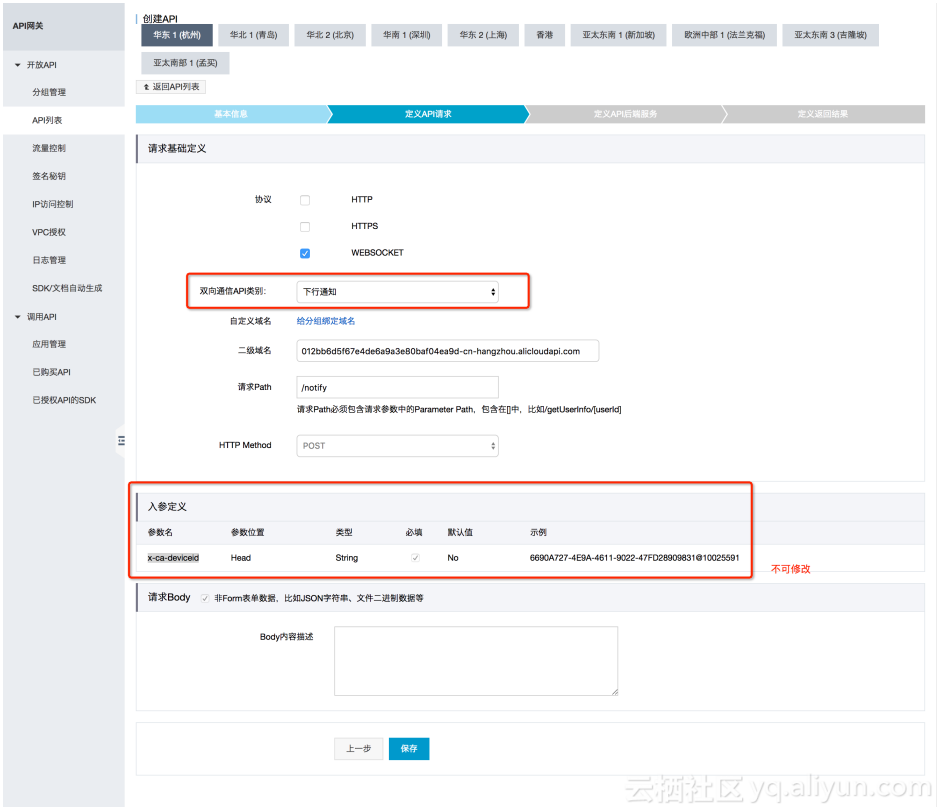
(1) 强烈建议使用和客户端不同的APP进行授权，区分用户后端服务和客户端调用权限；

(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

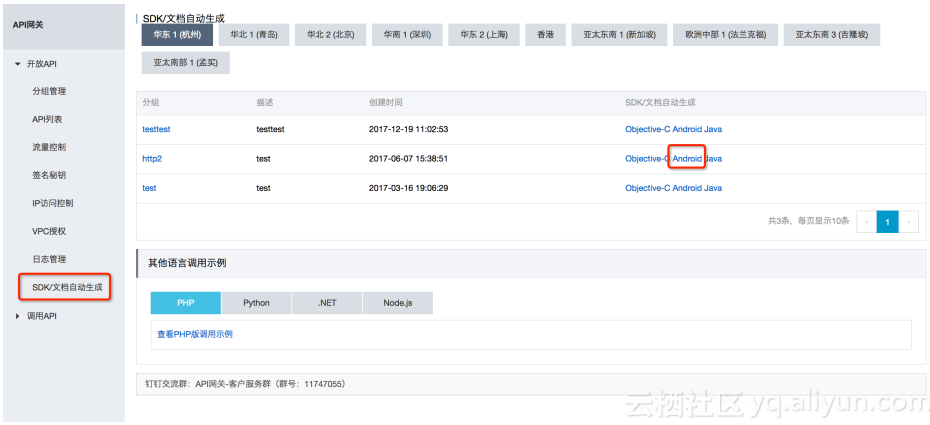
注销信令是客户端发送给用户后端服务的信令，创建的时候需要注意的是只能通过WebSocket协议传输。

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；

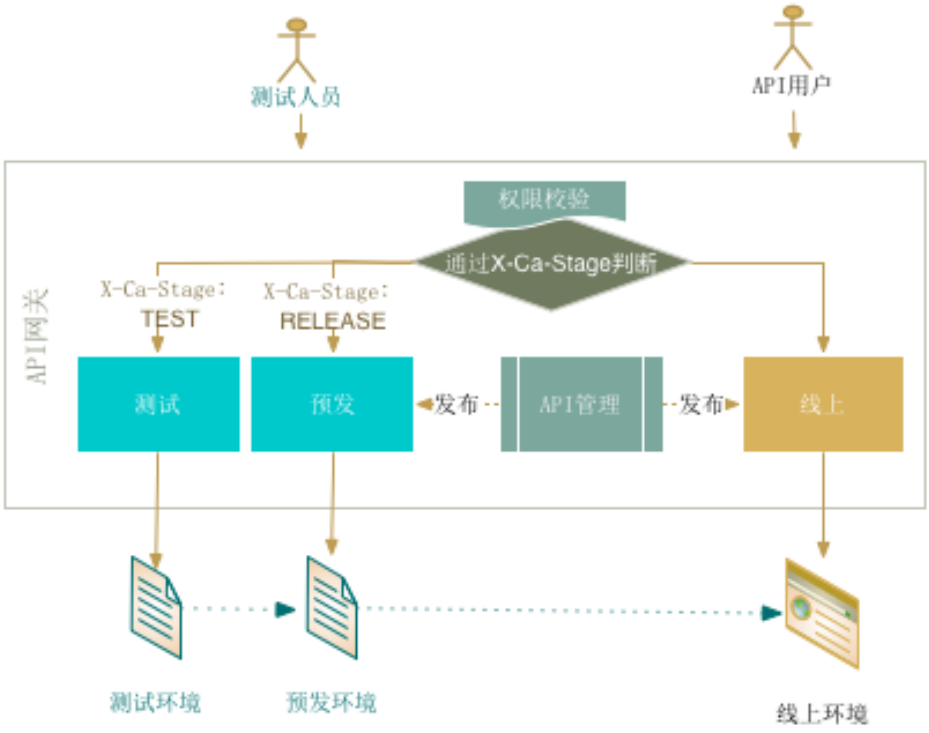
如果在使用中遇到棘手的问题，请加入我们钉钉交流群：API网关-客户服务群（群号：11747055）

环境管理

什么是环境管理

目前每个 API 分组有三个环境：测试、预发和线上。其中测试环境和预发环境为测试人员测试或调试 API 时使用的环境，线上环境主要为 API 用户使用的环境。

通过 API 分组的 **环境管理** 设置环境变量参数，为 API 分组的测试、预发、线上环境分别定义一个变量。环境变量参数，即为每个环境自定义的公共常量参数。当调用 API 时，可以将环境参数放置于请求的任意位置，传递给后端服务。API 网关将适配您请求中的环境参数信息来区分请求环境。



环境变量参数配置方法

首先，在 API 分组的 **环境管理** 中，为每个环境创建变量。然后，在 **API 定义** 中配置已创建的环境变量。

创建环境变量

要实现通过环境变量参数区分请求环境，您需为测试、预发、线上三个环境，分别新增一个变量。

目前，每个环境允许配置最多 50 个环境变量。

1. 登录 API 网关控制台。
2. 单击 **分组管理** > **环境管理**。

3. 选择要增加变量的环境：线上、预发、或测试，再单击 **新增变量**。您需为不同的环境逐个新增一个变量。

   填写变量名称和值，再单击 **新增**。

   Name：自行定义变量名称，但需 **保持三个环境中的对应的变量名称相同**。

   如果您有多个 API，建议 Name 标识有实际意义，以便后续查询。

   Value：变量的值。

如果以函数计算为 API 网关的后端服务，Value 值请填写您在函数计算服务中创建的服务名称或者函数名称。您需填写正确的服务名称或者函数名称，否则可能造成无法调用 API。以函数计算服务为例。一个函数服务，测试、预发、线上环境的名称分别为：TestServiceD、PreServiceD、ServiceD。（函数计算中环境变量设置，请参见函数计算文档 **环境变量**。）为 API 分组的测试、预发、线上环境分别定义一个的变量。变量可命名为 Service，并填写相应的服务名称作为 Value。

您还可以以函数名称录入名为 Function 的环境变量，并为三个环境分别设置变量。

在 API 定义里配置环境变量

API 定义时，在 **请求Path、入参定义、定义 API 后端服务**等部分加入变量。

表示方法：#变量名#。如，#Service#、#Function#。如，在定义以函数计算作为 API 网关后端服务时，将服务名称和函数名称定义为已创建的变量。

调用多环境 API

API 发布后，便可发起 API 调用。

线上环境调用

直接发起 API 调用，即调用线上环境。

预发环境调用

调用预发环境的 API， 则在调用 API 时，在 Header 中增加入参 X-Ca-Stage: PRE， 即可访问预发环境的 API。

测试环境调用

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

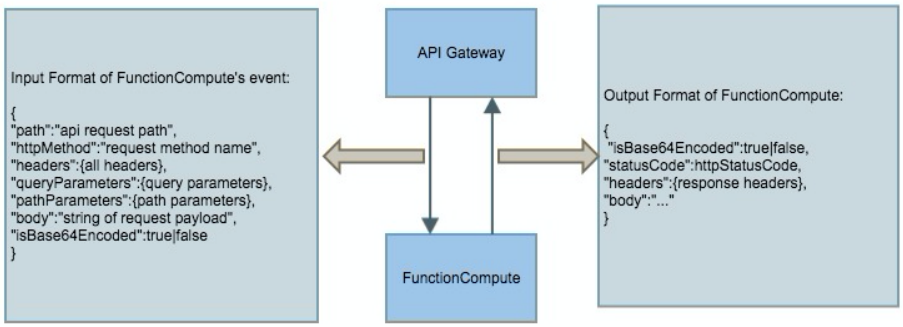
以函数计算作为 API 网关后端服务

函数计算是一个事件驱动的服务。函数的执行可以由事件驱动，即当某个事件发生时，该事件触发函数的执行。现在，函数计算支持以 API 网关作为事件源。当请求设置函数计算为后端服务的 API 时，API 网关会触发相应的函数，函数计算会将执行结果返回给 API 网关。

API 网关与函数计算对接，可以让您以 API 形式安全地对外开放您的函数，并且解决认证、流量控制、数据转换等问题（查看 API 网关功能）。

实现原理

API 网关调用函数计算服务时，会将 API 的相关数据转换为 Map 形式传给函数计算服务。函数计算服务处理后，按照下图中 Output Format 的格式返回 statusCode、headers、body 等相关数据。API 网关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。



API 网关向函数计算传入参数格式

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。

```
{
"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"
}
```

注意：

- 如果 "isBase64Encoded" 的值为 "true"，表示 API 网关传给函数计算的 body 内容已进行 Base64 编码。函数计算需要先对 body 内容进行 Base64 解码后再处理。
- 如果"isBase64Encoded" 的值为 "false"，表示 API 网关没有对 body 内容进行 Base64 编码。

函数计算的返回参数格式

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

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

注意：

- 当 body 内容为二进制编码时，需在函数计算中对 body 内容进行 Base64 编码，设置 "isBase64Encoded" 的值为 "true"。如果 body 内容无需 Base64 编码，"isBase64Encoded" 的值为 "false"。API 网关会对 "isBase64Encoded" 的值为 "true" 的 body 内容进行 Base64 解码后,再返回给客户端。
- 在 Node.js 环境中，函数计算根据不同的情况设置 callback。
  - 返回成功请求： callback{null,{ “statusCode” :200,” body” :” …” }}。
  - 返回异常：callback{new Error(‘internal server error’),null}。
  - 返回客户端错误： callback{null,{ “statusCode” :400,” body” :” param error” }}。
- 如果函数计算返回不符合格式要求的返回结果，API 网关将返回 503 Service Unavailable 给客户端。

配置 API 网关触发函数计算

配置 API 网关触发函数计算服务的操作步骤：

1. 在函数计算控制台创建函数
2. 创建并定义以函数计算为后端服务的 API
3. 调试 API
4. 将 API 发布到线上

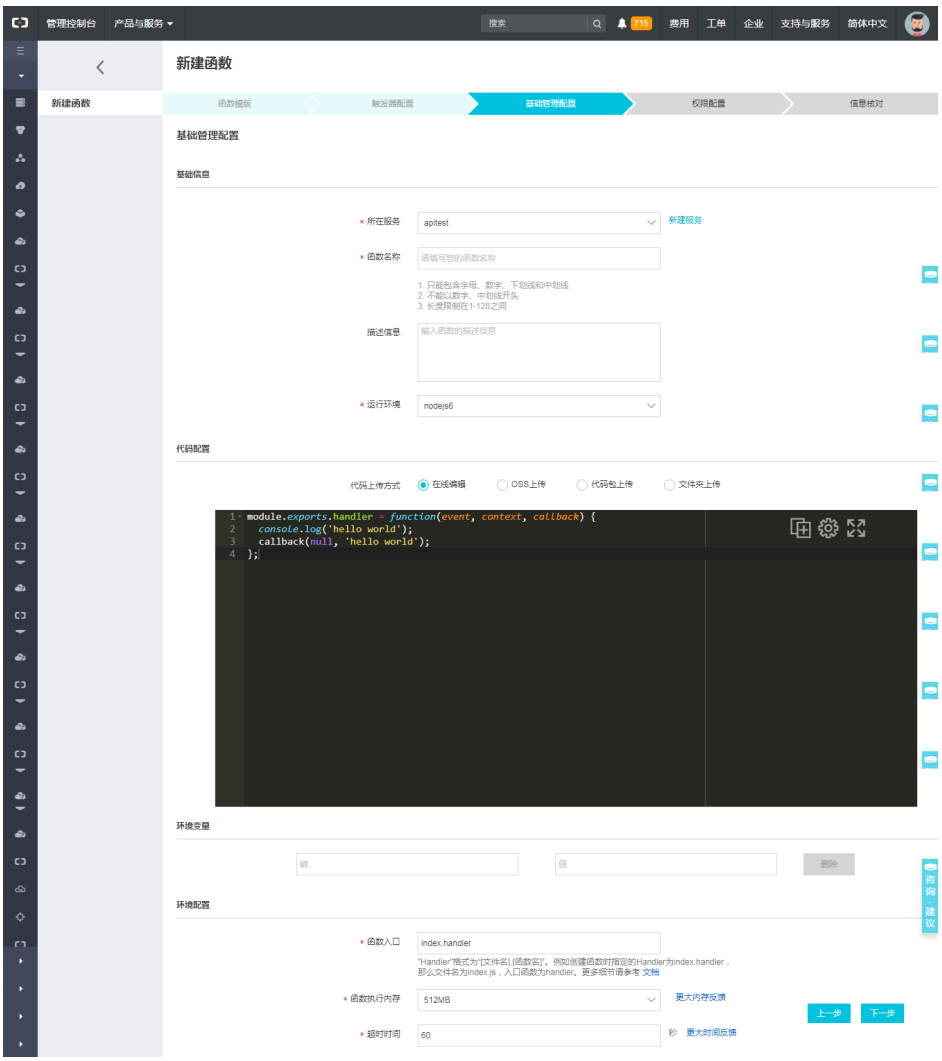
在函数计算控制台创建函数

创建服务。登录 **函数计算控制台**， 选择您要创建服务和函数的 **所属区域**，单击 **新建服务**，并在弹出对话框中完成服务创建。

注意：服务创建成功后，无法更换区域，请谨慎选择 **所属区域**。

2. 在已创建的服务中，创建函数。在该服务页面上，单击 **新建函数** 进入函数创建流程：
   i. 选择函数模板。函数计算控制台中，提供了 Node.js 6 环境的 API 网关后端实现模板 api-gateway-nodejs6 供您使用。如果 api-gateway-nodejs6 模板不适用您的业务场景，请选择 **空白函数**。选择使用 **空白函数** 模板后，在 **基础管理配置** 中需提交您自己编写的代码。请提前准备好代码包，以便上传。

   ii. 触发器配置。**触发器类型** 选择为 **不创建触发器**，单击 **下一步**。
   iii. 基础管理配置：填写基础信息、配置代码、**设置环境变量**、和配置环境，再单击 **下一步**。
   关于代码编写示例，可参见 API网关触发函数计算 文档中，**编写函数代码** 部分。



iv. 请忽略权限配置，直接单击 **下一步**。因为我们已在 RAM 控制台配置了相应的 roleArn 的权限，所以无需您手动配置。 您只需在 API 网关控制台创建 API 时，单击 **获取授权** ，即可自动获取授权。

v. 核对信息，信息无误，则单击 **创建**。成功创建函数后，您可以在 **函数列表** 中，查看所创建函数的基本信息。

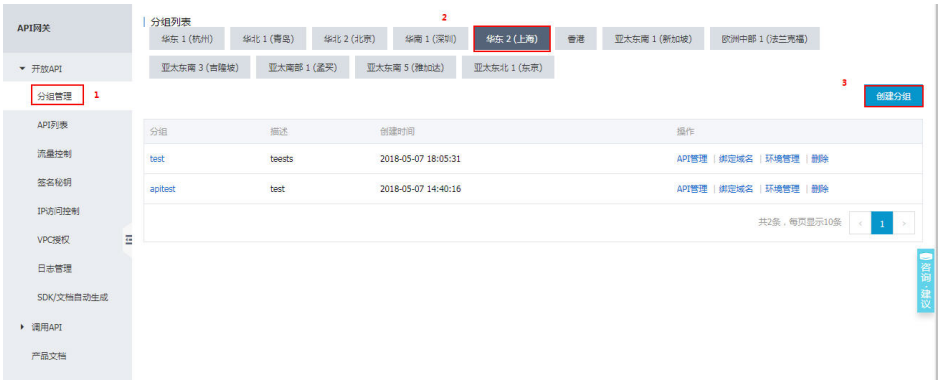
创建并定义以函数计算为后端服务的 API

您需在 API 网关控制台 创建 API， 并定义此 API 的后端服务为函数计算。

1. 登录 API 网关控制台。

单击控制台左侧导航栏中 **分组管理**，选择分组列表的地域，再单击 **创建分组**。（如果已创建分组，请忽略此步骤。）

> **注意**：如果函数计算与 API 不在同一地域，将通过公网访问您的函数计算服务。若您对数据安全和网络延迟有较高要求，请选择API 与函数计算为同一地域。



API 分组创建成功后，您可以通过 **环境管理** 为此分组设置环境变量。目前有三种环境：测试、预发、和线上。为避免环境转换导致后端地址变化，您可以通过增加环境变量参数的来实现请求的自动路由。环境变量配置方法，请参见 **环境管理**。

创建和定义 API。

i. 创建分组成功后，单击该分组操作栏中 **API 管理** 按钮，进入相应的 **API 列表** 页面。
ii. 单击 **创建 API**, 进入 API 创建和定义流程。

填写基本信息，再单击 **下一步**。

注意：如果选择类型为 **私有**，该 API 将不能在云市场上架。

定义 API 请求，再单击 **下一步**。

注意：如果 **入参请求模式** 选择为 **入参透传**，则发送给 API 网关的参数 body 内容不经处理，直接作为参数透传给函数计算。

定义 API 后端服务，再单击 **下一步**。

注意：在此页面，您需：

i. 选择 **后端服务类型** 为 **函数计算**。
ii. 填写 **服务名称** 为您在 **函数计算控制台** 创建的服务名称。
iii. 填写 **函数名称** 为您在 **函数计算控制台** 创建的函数名称。
iv. 单击 **获取授权**，自动获取角色 Arn。如果这是您第一次获取函数计算为 API 网关后端服务的角色授权，当您单击 **获取授权** 后，会弹出 RAM 控制台的授权页面。您需单击 RAM 控制台的授权权限，然后返回 API 创建页面再次单击 **获取授权**，该角色 Arn将自动显示在选项框中。

定义返回结果，然后单击 **创建**。

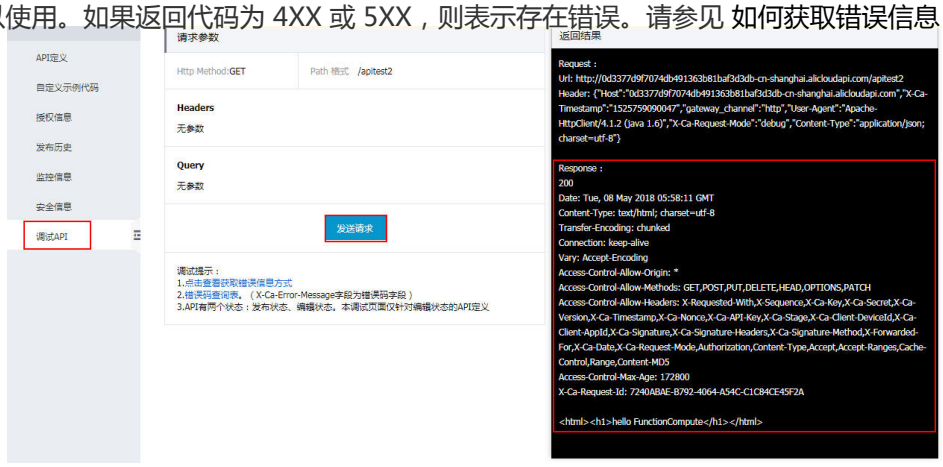
注意：返回结果示例为必填，且格式需遵循 **函数计算的返回参数格式** 。

如需更多帮助，请参见 API 创建。

调试 API

API 创建、定义完成后，页面自动跳转到 **API 列表** 页。您可以通过此页面按钮，对创建的 API 进行测试是否可用，请求链路是否正确。

1. 单击 API 名称或 **管理** 按钮，进入 **API 定义** 页面。
2. 单击左侧导航栏中 **调试 API**。
3. 输入请求参数，单击 **发送请求**。返回结果将显示在右侧页面。如果调试返回成功结果，则说明该 API 可以使用。如果返回代码为 4XX 或 5XX，则表示存在错误。请参见 **如何获取错误信息** 和 **错误代码表**。



4. 将 API 发布到 **预发** 环境，做上线前测试。测试证明 API 可用后，可返回 **API 定义** 页面，将 API 发布到 **预发** 环境。然后，通过访问二级域名来进行测试调用，模拟真实的用户请求。

   注意： 如果在 API 定义中设置了环境变量，需在请求 Header 中增加入参 X-Ca-Stage: RELEASE 才能正确调用预发环境的 API。

将 API 发布到线上

API 通过调试，证明可以使用后，可将 API 进行发布。

1. 在 **API 列表** 页面，单击 API 名称或 **管理** 按钮，进入 **API 定义** 页面。

2. 单击页面右上方 **发布** 按钮，弹出 **发布 API** 对话框。
3. 选择要发布的环境为**线上**，填写备注信息，单击 **发布**。将 API 发布到线上后，您的用户便可以调用此 API。

更多发布相关细节，请参见文档 发布 API。

示例

以下提供三个示例，分别为：函数代码示例、API 请求示例、和 API 网关返回示例。

函数代码示例

在函数计算中配置的代码示例。

```
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);
};
```

请求示例

以 POST 形式请求 path 为如下的 API：

```
/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":"TnoBIdxxRHrFferGlzzkGcQsaezK+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 网关返回示例

```
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"}}}
```

常见问题

为什么我无法录入我已有的函数？

请确认您输入的函数计算的服务名称和函数名称是否与您在函数计算控制台创建的服务和函数的名称一致。

我是否可以将多个函数作为一个 API 的后端服务？

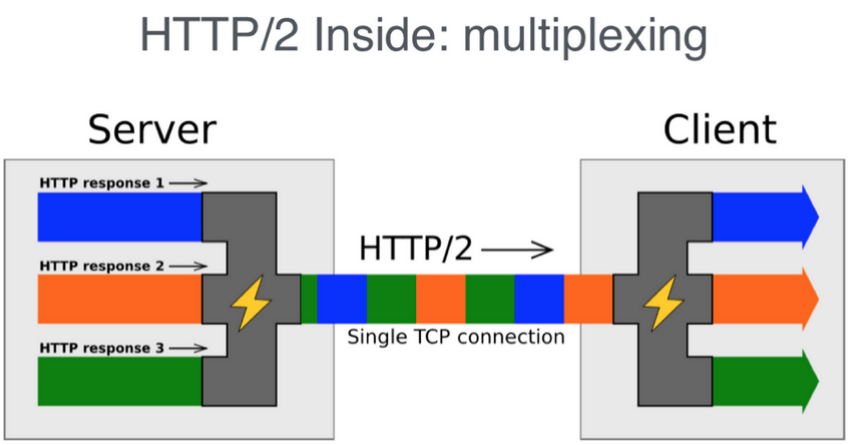
不可以，目前 API 和函数是一对一的关系存在。

支持HTTP2.0

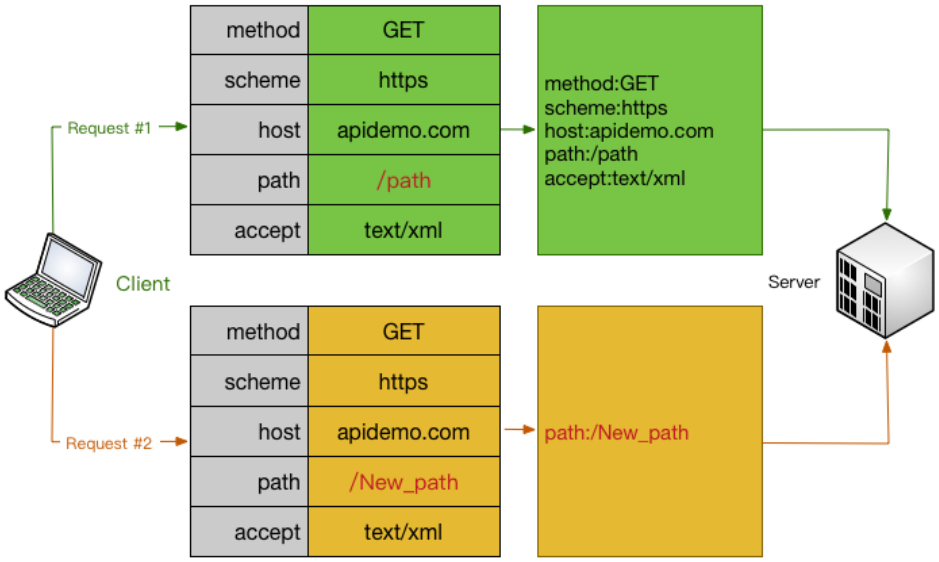
API网关支持HTTP2.0

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

自定义上传证书，包含证书/私钥，均为 PEM 格式， 证书格式说明 (注：API网关Tengine服务是基于Nginx，因此只支持Nginx能读取的证书，即PEM格式)。

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-----
MIIEpAIBAAKCAQEA8GjIeJ7rlo86mtbwcDnUfqzTQAm4b3zZEo1aKsfAuwcvCud
....
-----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

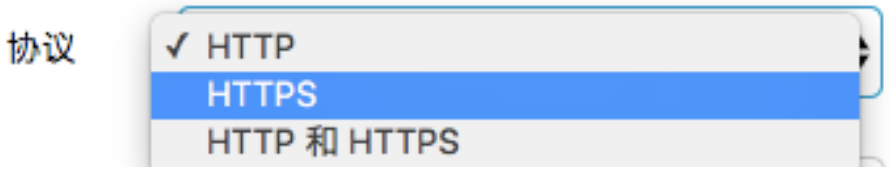
在绑定SSL证书，您首先需要您在API分组上绑定【独立域名】

【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。

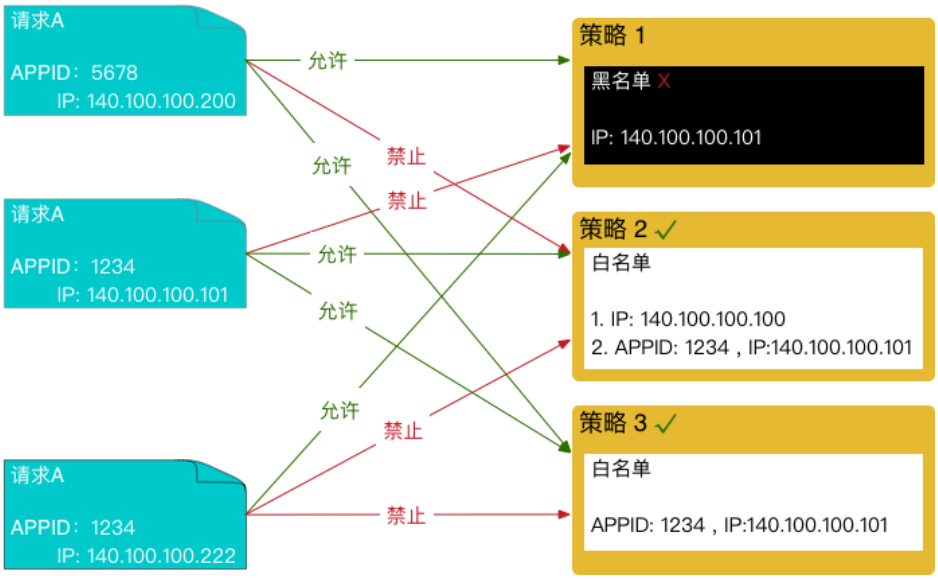
API支持的协议包括：

- 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】

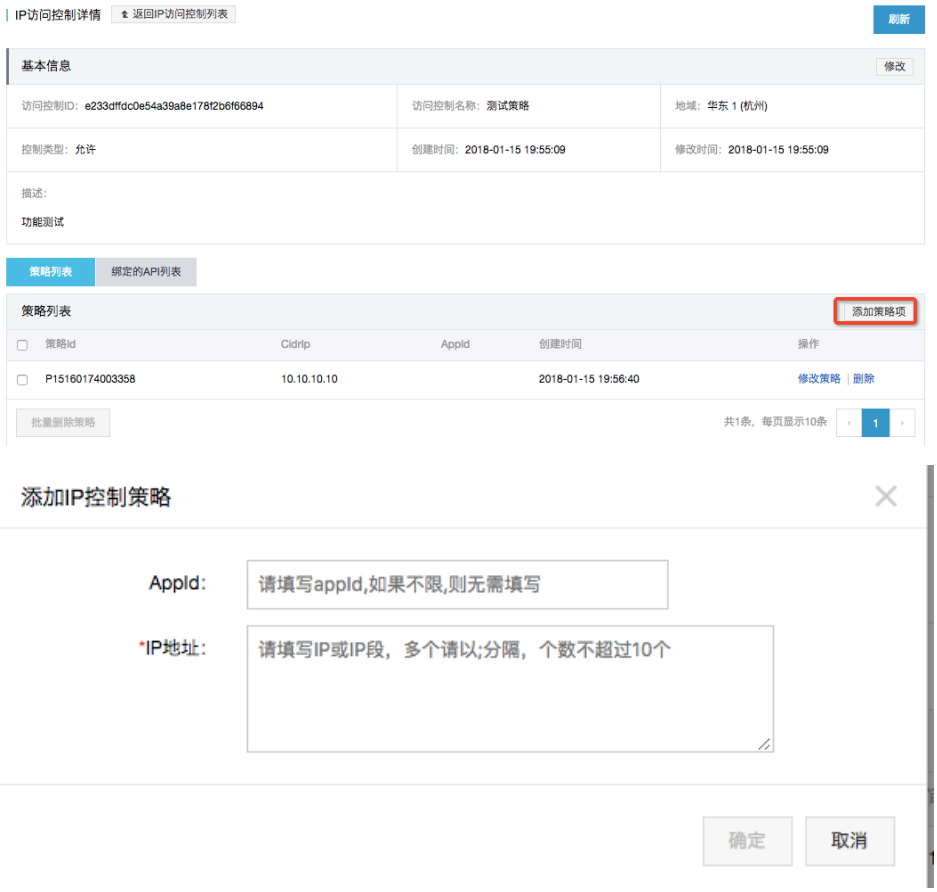
点击“创建访问控制”，弹出IP访问控制创建窗口：

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



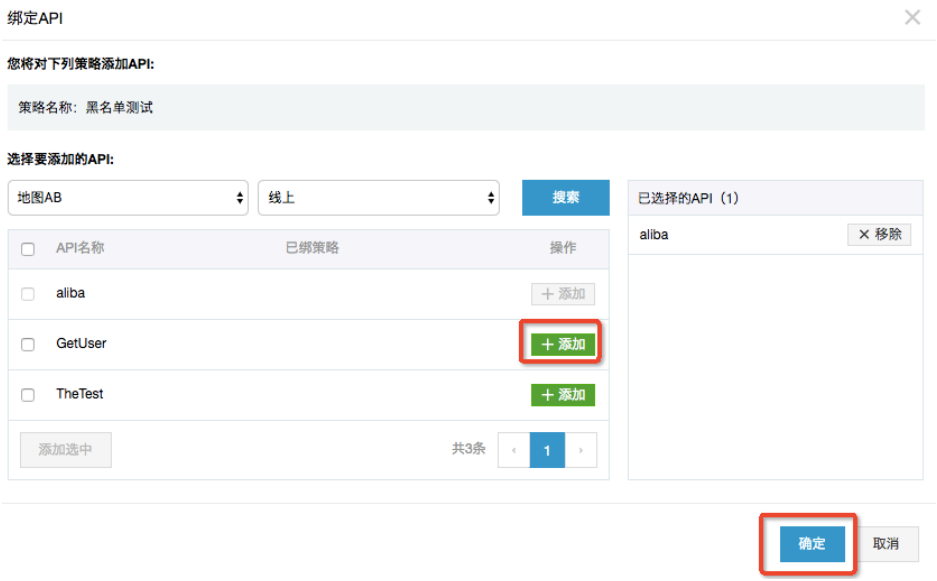
确定后完成策略添加。

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。

API网关实现跨域资源共享（CORS）

一.跨域带来的安全问题及浏览器的限制访问

当一个资源从与该资源本身所在的服务器不同的域或端口请求一个资源时，资源会发起一个跨域 HTTP 请求。比如，站点 http://www.aliyun.com 的某 HTML 页面通过img的src请求 http://www.alibaba.com/image.jpg 。 网络上的许多页面都会加载来自不同域的CSS样式表，图像和脚本等资源。

出于安全原因，浏览器限制从页面脚本内发起的跨域请求，有些浏览器不会限制跨域请求的发起，但是会将结果拦截了。 这意味着使用这些API的Web应用程序只能加载同一个域下的资源，除非使用CORS机制（Cross-Origin Resource Sharing 跨源资源共享）获取目标服务器的授权来解决这个问题。

上图画的是典型的跨域场景，目前主流浏览器为了用户的安全，都会默认禁止跨域访问，但是主流浏览器都支持W3C推荐了一种跨域资源共享机制（CORS）。服务器端配合浏览器实现CORS机制，可以突破浏览器对跨域资源访问的限制，实现跨域资源请求。

二.跨域资源共享CORS介绍

2.1 两种验证模式

跨域资源共享CORS的验证机制分两种模式：简单请求和预先请求。

当请求**同时满足下面三个条件时**，CORS验证机制会使用简单模式进行处理。

1.请求方法是下列之一：

- GET
- HEAD
- POST

2.请求头中的Content-Type请求头的值是下列之一：

- application/x-www-form-urlencoded
- multipart/form-data
- text/plain

3.Fetch规范定义了CORS安全头的集合（跨域请求中自定义的头属于安全头的集合）该集合为：

- Accept
- Accept-Language
- Content-Language
- Content-Type （需要注意额外的限制）
- DPR

- Downlink
- Save-Data
- Viewport-Width
- Width

否则CORS验证机制会使用预先请求模式进行处理。

2.2 简单请求模式

简单请求模式，浏览器直接发送跨域请求，并在请求头中携带Origin的头，表明这是一个跨域的请求。服务器端接到请求后，会根据自己的跨域规则，通过Access-Control-Allow-Origin和Access-Control-Allow-Methods响应头，来返回验证结果。

应答中携带了跨域头 Access-Control-Allow-Origin。使用 Origin 和 Access-Control-Allow-Origin 就能完成最简单的访问控制。本例中，服务端返回的 Access-Control-Allow-Origin: * 表明，该资源可以被任意外域访问。如果服务端仅允许来自 http://www.aliyun.com 的访问，该首部字段的内容如下：

```
Access-Control-Allow-Origin: http://www.aliyun.com
```

现在，除了 http://www.aliyun.com ，其它外域均不能访问该资源。

2.3 预先请求模式

浏览器在发现页面发出的请求非简单请求，并不会立即执行对应的请求代码，而是会触发预先请求模式。预先请求模式会先发送Preflighted requests（预先验证请求），Preflighted requests是一个OPTION请求，用于询问要被跨域访问的服务器，是否允许当前域名下的页面发送跨域的请求。在得到服务器的跨域授权后才能发送真正的HTTP请求。

OPTIONS请求头部中会包含以下头部：Origin、Access-Control-Request-Method、Access-Control-Request-Headers。服务器收到OPTIONS请求后，设置Access-Control-Allow-Origin、Access-Control-Allow-Method、Access-Control-Allow-Headers、Access-Control-Max-Age头部与浏览器沟通来判断是否允许这个请求。如果Preflighted requests验证通过，浏览器才会发送真正的跨域请求。

请求中的跨域头 Access-Control-Request-Method 告知服务器，实际请求将使用 GET 方法。请求中的跨域头 Access-Control-Request-Headers 告知服务器，实际请求将携带两个自定义请求首部字段：x-ca-nonce 与 content-type。服务器据此决定该实际请求是否被允许。

应答中的跨域头 Access-Control-Allow-Methods 表明服务器允许客户端使用 GET 方法发起请求。值为逗号分割的列表。

应答中的跨域头 Access-Control-Allow-Headers 表明服务器允许请求中携带字段 x-ca-nonce 与 content-type。与 Access-Control-Allow-Methods 一样，Access-Control-Allow-Headers 的值为逗号分割的列表。

应答中的跨域头 Access-Control-Max-Age 表明该响应的有效时间为 86400 秒，也就是 24 小时。在有效时间内，浏览器无须为同一请求再次发起预检请求。请注意，浏览器自身维护了一个最大有效时间，如果该首部

字段的值超过了最大有效时间，将不会生效。

三.在API网关实现CORS跨域资源共享

3.1实现简单请求模式

API网关默认所有API允许跨域访问，因此如果用户的API后端服务的应答中不做特殊返回，API网关会返回允许所有域跨域访问的相关头，下面是一个示例：

客户端的API请求

```
GET /simple HTTP/1.1
Host: www.alibaba.com
orgin: http://www.aliyun.com
content-type: application/x-www-form-urlencoded; charset=utf-8
accept: application/json; charset=utf-8
date: Mon, 18 Sep 2017 09:53:23 GMT
```

后端服务应答

```
HTTP/1.1 200 OK
Date: Mon, 18 Sep 2017 09:53:23 GMT
Content-Type: application/json; charset=UTF-8
Content-Length: 12

{"200","OK"}
```

API网关应答

```
HTTP/1.1 200 OK
Date: Mon, 18 Sep 2017 09:53:23 GMT
Access-Control-Allow-Origin: *
X-Ca-Request-Id: 104735BD-8968-458F-9929-DBFA43F324C6
Content-Type: application/json; charset=UTF-8
Content-Length: 12

{"200","OK"}
```

从上面三个报文可以看出，API网关会对用户的后端服务应答做一定修改，增加一个跨域头：

```
Access-Control-Allow-Origin: *
```

这个跨域头的意思是，本API允许所有域的请求访问。

如果用户需要定制针对简单请求的应答的跨域头，只需要在后端服务应答中，增加Access-Control-Allow-Origin这个跨域头即可，后端服务应答中的头会默认覆盖掉API网关自己增加的头。下面是一个例子，这个例子

中的API只允许http://www.aliyun.com 这一个域访问：

客户端的API请求

```
GET /simple HTTP/1.1
Host: www.alibaba.com
orgin: http://www.aliyun.com
content-type: application/x-www-form-urlencoded; charset=utf-8
accept: application/json; charset=utf-8
date: Mon, 18 Sep 2017 09:53:23 GMT
```

后端服务应答

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: http://www.aliyun.com
Date: Mon, 18 Sep 2017 09:53:23 GMT
Content-Type: application/json; charset=UTF-8
Content-Length: 12

{"200","OK"}
```

API网关应答

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: http://www.aliyun.com
X-Ca-Request-Id: 104735BD-8968-458F-9929-DBFA43F324C6
Date: Mon, 18 Sep 2017 09:53:23 GMT
Content-Type: application/json; charset=UTF-8
Content-Length: 12

{"200","OK"}
```

3.2 实现预先请求模式

API网关允许用户设置方法为OPTIONS的API，并且将后端服务的OPTIONS应答透传给客户端。新建方法为OPTIONS的API，定义的其他部分与正常API一样，有两点需要注意：

- 定义API认证方式时选择无认证；
- 定义API请求时，需要设置path为/，并且匹配所有子路径。选定方法为OPTIONS，API网关控制台会默认设置请求模式为透传模式，且不可修改，用户不需要定义请求参数；

用户可以在每个API分组下建立一个方法的OPTIONS的API，来定义这一组API绑定的域名的跨域资源策略。用户可以是用CURL方法来测试自己的跨域API应答情况，下面是针对一个定义好的OPTIONS的API访问的一个示例：

```
sudo curl -X OPTIONS -H "Access-Control-Request-Method:POST" -H "Access-Control-Request-Headers:X-CUSTOM-HEADER" http://ec12ac094e734544be02c928366b7b26-cn-qingdao.alicloudapi.com/optinstest -i
```

```
HTTP/1.1 200 OK
Server: Tengine
Date: Sun, 02 Sep 2018 15:32:19 GMT
Connection: keep-alive
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET,POST,PUT,DELETE,HEAD,OPTIONS,PATCH
Access-Control-Allow-Headers: X-CUSTOM-HEADER
Access-Control-Max-Age: 172800
X-Ca-Request-Id: 1016AC86-E345-405C-8049-A6C24078F65F
```

用户在实现方法为OPTIONS的API的时候需要注意的一点是：**API网关会对用户的后端服务应答做一定修改，增加四个跨域头（Access-Control-Allow-Origin、Access-Control-Allow-Methods、Access-Control-Allow-Headers、Access-Control-Max-Age），后端服务应答中，需要返回所有跨域头来覆盖API网关默认跨域头。**

下面是一个完整的预先请求模式的请求与应答示例。

客户端的方法为OPTIONS的API请求

```
OPTIONS /simple HTTP/1.1
Host: www.alibaba.com
orgin: http://www.aliyun.com
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-PINGOTHER, Content-Type
accept: application/json; charset=utf-8
date: Mon, 18 Sep 2017 09:53:23 GMT
```

后端服务应答

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: http://www.aliyun.com
Access-Control-Allow-Methods: GET,POST
Access-Control-Allow-Headers: X-CUSTOM-HEADER
Access-Control-Max-Age: 10000
Date: Mon, 18 Sep 2017 09:53:23 GMT
Content-Type: application/json; charset=UTF-8
```

API网关应答

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: http://www.aliyun.com
Access-Control-Allow-Methods: GET,POST
Access-Control-Allow-Headers: X-CUSTOM-HEADER
Access-Control-Max-Age: 10000
X-Ca-Request-Id: 104735BD-8968-458F-9929-DBFA43F324C6
Date: Mon, 18 Sep 2017 09:53:23 GMT
Content-Type: application/json; charset=UTF-8
```

客户端发送正常业务请求

```
GET /simple HTTP/1.1
Host: www.alibaba.com
orgin: http://www.aliyun.com
content-type: application/x-www-form-urlencoded; charset=utf-8
accept: application/json; charset=utf-8
date: Mon, 18 Sep 2017 09:53:23 GMT
```

后端服务应答

```
HTTP/1.1 200 OK
Date: Mon, 18 Sep 2017 09:53:23 GMT
Content-Type: application/json; charset=UTF-8
Content-Length: 12

{"200","OK"}
```

API网关应答

```
HTTP/1.1 200 OK
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: 104735BD-8968-458F-9929-DBFA43F324C6
Date: Mon, 18 Sep 2017 09:53:23 GMT
Content-Type: application/json; charset=UTF-8
Content-Length: 12

{"200","OK"}
```

使用简单认证（AppCode）方式调用API

1.概述

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。

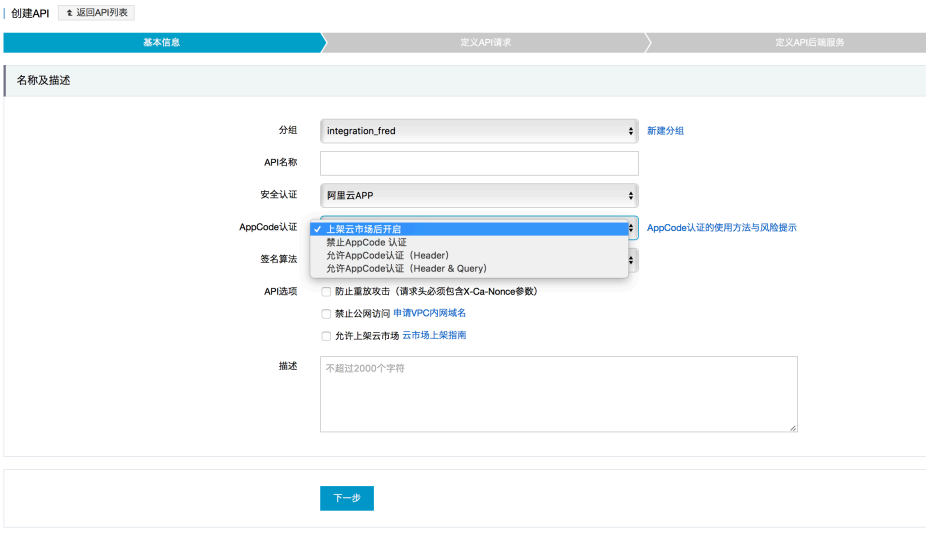
对于请求的签名认证方式，可以参考这个文档：请求签名说明文档：
https://help.aliyun.com/document_detail/29475.html

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：
https://help.aliyun.com/document_detail/29497.html
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）；
- Authorization字段的值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。

模型管理

1. 模型的简介及限定

模型用于描述HTTP协议的请求数据和响应数据。API网关通过使用JSON Schema定义模型，用来描述用户API约定数据的组织方式，比如参数或者返回值有哪些字段，这些字段的取值范围等。同时，通过定义模型，并在用户创建的API中加以引用，用户在API的SDK导出时，关联的Model会自动生成对应的POJO类。这样可以增强用户传入参数的便利性，同时可以方便用户反序列化返回的数据。

API网关模型定义基于JSON架构草案4的规范，但存在一定的条件限制：

1. 仅支持创建元素属性为Object类型的JSON Schema
2. $ref仅支持本用户的内部Model引用。Model的‘ref’引用地址可以通过CreateModel和Describe Models获取。‘ref’不支持循环引用。

Api网关支持的模型可以参考如下定义：

```
{
"required": ["name", "photoUrls"],
"type": "object",
"properties": {
"id": {
"format": "int64",
"type": "integer"
},
"category": {
"$ref":
"https://apigateway.aliyun.com/models/bbc725be4b0b48b79bdd2f6ebbdcc8c0/a5e7741d8a3a4bcb9746275a0db15
fcf"
},
"name": {
"pattern": "^\\d{3}-\\d{2}-\\d{4}$",
"type": "string"
},
"status": {
"type": "string"
},
"dogProject": {
"type": "object",
"properties": {
"id": {
"format": "int64",
```

```
"maximum": 100,
"exclusiveMaximum": true,
"type": "integer"
},
"name": {
"maxLength": 10,
"type": "string"
}
}
}
}
}
```

2. 创建模型

您可以通过阿里云提供的Openapi-CreateModel进行模型创建。同样，您也可以通过API网关的控制台进行创建。

模型相关操作的控制台入口：

1. 点击分组管理
2. 点击模型管理，进入模型管理界面来创建模型

Swagger导入创建模型：

Api网关支持Swagger导入。Swagger文件中的Model相关内容会在Swagger导入成功后，会在该分组下自动生成模型。注意：**通过Swagger导入模型时，同名模型将直接被覆盖，不会进行用户确认。**

3. 修改和查看模型

完成模型的创建后，可以在模型管理界面点击查看所需的模型。在模型的详情页，可以看到模型的名称，模型的定义，以及系统为其分配的URI。API网关模型间可以通过‘$ref:{URI}’来实现模型间的项目引用。

如果用户希望对当前模型的信息进行修改，可以点击右上角修改按钮完成模型的修改。需要注意的是：**模型的URI不随模型的更改发生改变**

4. 删除模型

用户可以对分组下的模型进行删除操作。注意:**API网关不维护模型和API的关联关系，删除模型时可能会引起线上API的SDK导出失败等问题。因此，删除模型请谨慎操作**