ALIBABA CLOUD

Alibaba Cloud

API 网关

插件使用

文档版本：20220513

阿里云

法律声明

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

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

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

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

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

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

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

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

目录

1.概述

目前插件仅在共享实例（VPC）与专享实例（VPC）可用，共享实例（经典网络）目前暂不支持。

> 说明
>
> 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（缓存），Routing（后端路由），参数访问控制，错误码映射，断路器等也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。

1. 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。

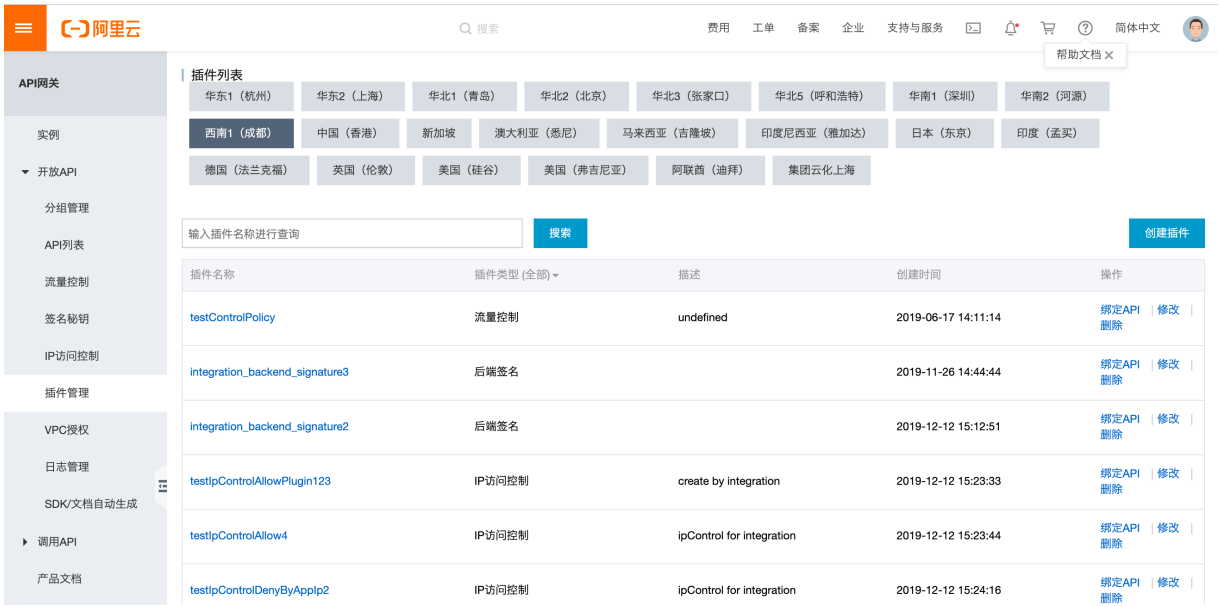
2. 支持插件列表

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
- 参数访问控制
- 后端签名插件
- JWT认证插件
- CORS（跨域资源访问）
- Caching（缓存）
- Routing（后端路由）
- 错误码映射
- 断路器（仅专享实例可用）

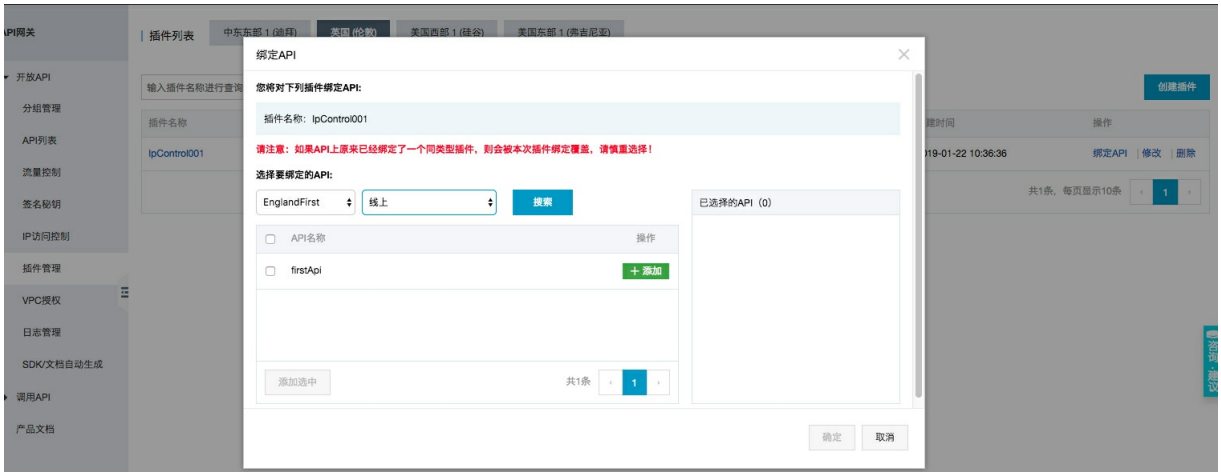
3. 快速使用

- 访问API网关插件控制台。



- 点击创建插件按钮创建插件。

- 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。

4. 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

5. 使用限制

- 单个插件元数据的大小限制为**50KB**。
- 每个用户在每个Region创建插件的限制为**10000**个。
- 控制台的调试功能暂时不支持JWT插件，建议使用Postman或者系统命令行 `curl` 调试。

2.后端签名插件

后端签名密钥是由您创建的一对 Key 和 Secret，相当于您给API网关颁发了一个账号密码。API绑定后端签名插件后 ，API 网关向您后端服务请求时会使用这一对Key和Secret对请求内容进行加签处理，您后端服务可以对网关发送过来的请求做对称加签计算，对比网关的签名和服务器端计算的签名是否一致就可以对网关做身份验证。

1. 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。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。

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。

2. 插件绑定

您将密钥绑定到 API 之后，由网关发送给您服务后端的该 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。

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。

3. 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: APIGW_BACKEND
key: SampleKey
secret: SampleSecret
```

4. 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。

5. 后端签名规则

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：

1. API网关需要从发送给后端的Http请求中提取出关键数据，组合成一个签名串，生成的签名串的格式如下：

   ```
   HTTPMethod
   Content-MD5
   Headers
   PathAndParameters
   ```

以上4个字段构成整个签名串，字段之间使用\n间隔，如果Headers为空，则不需要加\n，其他字段如果为空都需要保留\n。签名大小写敏感。下面介绍下每个字段的提取规则：

- HTTPMethod：HTTP的方法，全部大写，比如POST
- Content-MD5：网关从客户端请求Header中拿到Content-MD5头的值，只有在请求存在Body且Body为非Form形式时才计算Content-MD5头，下面是Java的Content-MD5值的参考计算方式：

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

- 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。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"
```

- PathAndParameters

这个字段包含Path，Query和Form中的所有参数，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 PathAndParameters = Path

```
String PathAndParameters =
 Path +
 "?" +
 Key1 + "=" + Value1
+ "&" + Key2 + "=" + Value2 +
 ...
"&" + KeyN + "=" + ValueN
```

说明 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； 只要传递了的参数，签名过程中的等号 “=“无论什么情况都需要保留，比如这两个query参数传递时的形式："path?a=&b"，签名时需要写成："path?a=&b="。

2. 计算签名：

```
Mac hmacSha256 = Mac.getInstance("HmacSHA256");
byte[] keyBytes = secret.getBytes("UTF-8");  //secret 为绑定到 API 上的签名密钥
hmacSha256.init(new SecretKeySpec(keyBytes, 0, keyBytes.length, "HmacSHA256"));
String sign = new String(Base64.encodeBase64(Sha256.doFinal(stringToSign.getBytes("UTF-8"))
,"UTF-8"));
```

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

6. 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。

7. 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。

3.参数与条件表达式的使用

本文讲述API网关的参数以及条件表达式的使用。

1. 概述

在 `授权控制` , `流量控制` , `后端路由` , `错误码映射` 插件中，支持用户从当前的 `请求` 、 `应答` 或 `系统上下文` 中获取参数，并使用自定义的**条件表达式**对参数进行判断，本文描述**参数**的定义方法与**条件表达式**的书写方法。

2. 参数的定义

2.1. 定义方式

在使用条件表达式前，需要先在 `parameters` 字段中将所有需要在条件表达式中需要使用的参数进行显式定义，参考下面的例子：

```
---
parameters:
  method: "Method"
  appId: "System:CaAppId"
  action: "Query:action"
  userId: "Token:UserId"
```

`parameters` 字段类型是字符串类型的键值对， `key` 和 `value` 的定义规范为

- `key` 表示将来在条件表达式中使用的变量名，变量名的规则为 `[a-zA-Z_][a-zA-Z0-9]+` ，唯一
- `value` 表示参数所在的位置，使用 `{location}` 或 `{location}:{name}` 的方式进行定义，请参考如下的例子：
- ◦ `location` 表示参数所在的位置，参考下一节的详细描述
  - ◦ `name` 表示参数的名称，用于在所在位置中定位参数，比如 `Query:q1` 表示请求的QueryString中名字为q1的第一个值

2.1. 参数的支持位置

在使用条件表达式前，我们需要先定义参与条件表达式计算的参数，目前API网关支持在插件中直接使用如下位置的参数

位置名称	适用范围	说明

位置名称	适用范围	说明
Method	请求	HTTP请求方法（大写），如： `GET` , `POST` ...
Path	请求	HTTP完整请求路径，如： `/path/to/query`
StatusCode	应答	后端的HTTP应答码，如： `200` , `400`
ErrorCode	应答	API网关系统错误码
Header	请求/应答	使用 `Header:{Name}` 获取名字为 `{Name}` 的HTTP头的第一个值
Query	请求	使用 `Query:{Name}` 获取QueryString中名字为 `{Name}` 的第一个值
Form	请求	使用 `Form:{Name}` 获取请求Form中名字为 `{Name}` 的第一个值
Host	请求	使用 `Host:{Name}` 获取匹配到的泛域名模板参数

位置名称	适用范围	说明
Parameter	请求	使用 `Parameter:{Name}` 获取用户API自定义参数中名字为 `Name` 的第一个值
BodyJsonField	应答*	使用 `BodyJson:{JPath}` 以 `JSONPath` 方式获取请求/应答包体中的JSON字段值
System	请求/应答	使用 `System:{Name}` 获取名字为 `{Name}` 的系统参数值
Token	请求/应答	当处于jwt插件认证时，使用 `Token:{Name}` 获取token中名字为 `{Name}` 的值

取值规则的说明

- 参数访问控制插件,流量控制插件,后端路由插件发生在请求阶段，插件仅支持适用范围为请求的参数位置：`Method` , `Path` , `Header` , `Query` , `Form` , `Parameter` , `System` , `Token`
- 错误码映射插件发生在应答阶段，插件仅支持适用范围为应答的参数位置： `StatusCode` , `ErrorCode` , `Header` , `BodyJsonField` , `System` , `Token`
- `Method` , `Path` , `StatusCode` , `ErrorCode` 这几个位置仅提供Location即可，不需要Name
- `Header` 位置的参数，当用于请求阶段的插件时，读取来自客户端请求中的Header; 当用于应答阶段的插件时，读取来自后端应答中的Header
- `Parameter` 位置的参数，仅用于请求阶段的插件，查找API定义中同名的参数，使用 **参数名** 而非 **后端参数名** 查找，当参数不存在时返回为 `null`
- `Host` 位置的参数，仅支持在泛域名参数提取，参考分组的域名绑定文档的3.2章节, 完整的Host请使用 `System:CaDomain` 读取系统参数
- `Path` 返回为完整的请求路径，如果需要 `Path` 位置的参数，请使用 `Parameter` 位置的参数

- `BodyJsonField` 位置的参数，目前仅支持在错误码映射插件中使用，使用 `JSONPath` 方式读取应答json包体的值，参考2.4. JSONPath使用说明
- `Token` 当配置了JWT认证插件时，使用 `Token:{CliamName}` 可以读取在插件定义的值，参考对应插件的文档

2.3. JSONPath使用说明

用于 `BodyJsonField` 位置，目前仅用于错误码映射插件，用于从后端返回的 `json` 包体中提取 `json` 中的字段，关于JSONPath的详细规范说明，请参考JSONPath介绍文档。

- 例子：使用 `code:"BodyJsonField:$.result_code"` 可以从如下的包体中得到 `result_code` 的值，对于如下的包体，可以解析到 `code` : `ok`

```
{ "result_code": "ok", "message": ... }
```

2.4. 系统参数列表

参数名称	参数含义	取值列表
CaClientIp	请求来源客户端IP	如： `37.78.XX.XX` ，`fe80::1849:59fd:993c:fcff`
CaDomain	请求的完整域名(Host头)	如：`api.foo.com`
CaAppId	请求的 APP 的 ID	如：`49382332`
CaAppKey*	请求的 APP 的 Key	如：`12983883923`
CaRequestId	网关生成的唯一请求Id	如：`CCE4DEE6-26EF-46CB-B5EB-327A9FE20ED1`

参数名称	参数含义	取值列表
CaApiName	API 名称	如： `TestAPI`
CaHttpSchema	调用协议	取值范围：`http`，`https`， `ws`
CaClientUa	客户端的UserAgent头	透传客户端上传的值
CaCloudMarketInstanceId	云市场购买关系Id	云市场
CaMarketExpriencePlan	是否开通云市场体验计划	开通：`true` 未开通：`false`

3. 条件表达式

条件表达式可在插件或其他场景中使用，用于在需要的场景中执行灵活的条件判断

3.1. 基本语法

- 条件表达式与SQL表达式类似，比如：`$A > 100 and '$B = 'B'`
- 表达式的基本为 `{参数} {操作符} {参数}` 格式，如：`$A > 100` 参数支持 **变量参数** 或 **常量参数**
- **变量参数** 以 `$` 开头，用于引用在上下文中定义好的参数，例如：`parameters` 中定义了 `q1:"Query:q1"` 的参数，在表达式中可以使用变量 `$q1`，这个变量的值为当前请求中名为 `q1` 的Query参数的值
- **常量参数** 支持 **字符串**、**数字**、**布尔类型**，如 `"Hello"`，`'foo'`，`100`，`-1`，`0.1`，`true`，请参考**3.2. 参数类型与判断规则**
- 支持以下 **操作符**：
- `=`，`==`：等于判断
- `<>`，`!=`：不等于判断
- `>`，`>=`，`<`，`<=`：比较判断

- `like` , `!like` : 相似判断，在字符串头尾的 `%` 可用于判断字符串相似，如 `$Query like 'Prefix%'`
- `in_cidr` , `!in_cidr` : 判断IP地址的掩码，例如: `$ClientIp in_cidr '47.89.XX.XX/24'`
- 可使用 `null` 来判断参数是否为空，如: `$A == null` 或 `$A != null`
- 可以使用 `and` , `or` , `xor` 来组合连接不同的表达式，默认连接顺序为从右到左
- 可以用小括号 `(` , `)` 来指定条件的优先级
- 使用 `!(` , `)` 可以对括起来的表达式执行取反操作，如: `!(1=1)` 结果为 `false`
- 系统内置了一些函数，用于一些特殊场景的判断
- - `Random()` : 产生一个 `0-1` 的浮点类型参数，用于蓝绿发布等一些需要随机的场合
  - `Timestamp()` : 返回以毫秒计数的 `Unix时间戳`
  - `TimeOfDay()` : 按 `GMT` 时区，返回当前时间到当日零点的毫秒数

3.2. 参数类型与判断规则

- 表达式中支持以下类型
  - `STRING` : 字符串类型，支持单引号或双引号，如: `"Hello"` , `'Hello'`
  - `NUMBER` : 整数或浮点数类型，如: `1001` , `-1` , `0.1` , `-100.0`
  - `BOOLEAN` : 布尔类型，如: `true` 、 `false`
- 对于 `等于` , `不等于` , `比较` 操作符，不同类型的判断依据为
  - `STRING` : 使用字符串顺序执行判断，例如如:
  - - `'123' > '1000'` 结果为true
    - `'A123' > 'A120'` 结果为true
    - `'' < 'a'` 结果为true
  - `NUMBER` : 使用数字大小进行判断
  - - `123 > 1000` 结果为false
    - `100.0 == 100` 结果为true
  - `BOOLEAN` : 布尔值的判断依据为: `true` 大于 `false`

- ■ `true == true` 结果为true
- ■ `false == false` 结果为true
- ■ `true > false` 结果为true

- 对于 **等于** ， **不等于** ， **比较** 操作符，如果左右两侧的参数类型不一致，参考以下的判断依据：
  - `STRING` `NUMBER` ：如果左值能转换为 `NUMBER` 类型，则按照数字大小判断，否则按照字符串大小判断，例如：
  - ■ `'100' = 100.0` 结果为true
  - ■ `'-100' > 0` 结果为false
  - `STRING` `BOOLEAN` ：当左值能够转换为 `BOOLEAN` 类型时（忽略大小写等于 `true` 或 `false` 是），按照 `BOOLEAN` 类型判断，否则除了 `!=` 判断外，其余的判断结果均为 `false` ，参考如下例子：
  - ■ `'True' = true` 结果为true
  - ■ `'False' = false` 结果为true
  - ■ `'bad' = false` 结果为false
  - ■ `'bad' != false` 结果为true，对于左值非 `true` 或 `false` 的场景，除了 `!=` 操作符结果为 `true` ，其余结果均为 `false`
  - ■ `'bad' != true` 结果为true
  - ■ `'0' > false` 结果为false
  - ■ `'0' <= false` 结果为false
  - `NUMBER` `BOOLEAN` ：永远返回 `false`
- `null` 值可用于字段是否为空的判断，对于 **等于** ， **不等于** ， **比较** 操作符，判断依据为
  - 当参数 `$A` 为空时， `$A == null` 结果为true， `$A != null` 结果为false
  - 空字符串 `''` 不等于 `null` 时： `'' == null` 结果为 `false` ， `'' == ''` 结果为 `true`
  - 当用于 **比较** 操作符时，任意一测值为 `null` ，则结果为 `false`
- `like` ， `!like` 操作符，可用于字符串前缀、后缀、包含的判断，判断依据为

- 表达式仅支持右值为 `STRING` 常量的写法，如 `$Path like '/users/%'`
- 右值两端的 `'%'` 字符可用于支持前缀、后缀、与包含判断，如
- ■ 前缀判断 `$Path like '/users/%'` ， `$Path !like '/admin/%'`
  - 后缀判断 `$q1 like '%search'` ， `$q1 !like '%.do'` ，
  - 包含判断 `$ErrorCode like '%400%'` ， `$ErrorCode !like '%200%'` ，
- 当左值为非 `NUMBER` 或 `BOOLEAN` 类型时，会转换为 `STRING` 格式后执行判断
- 当左值为空值 `null` 时，结果为 `false`

- `in_cidr` ， `!in_cidr` 表达式，可用于IP地址段掩码的判断，判断依据为
  - 表达式仅支持右值为 `STRING` 常量且符合IPv4或IPv6 CIDR格式的写法，如：
  - ■ `$ClientIP in_cidr '10.0.0.0/8'`
    - `$ClientIP !in_cidr '0:0:0:0:0:FFFF::/96'`
  - 当左值类型为 `STRING` 类型时，会作为IPv4地址后进行判断
  - 当左值类型为 `NUMBER` ， `BOOLEAN` 或为空时，结果为 `false`
  - `System:CaClientIp` 参数会返回客户端的IP地址，一般使用这种方式执行判断

4. 使用案例

- 随机5%的几率：

```
Random() < 0.05
```

- 判断当前API请求的是测试环境：

```
parameters:
  stage: "System:CaStage"
```

```
$CaStage = 'TEST'
```

- 自定义参数中的UserName是Admin且来源IP在 `47.47.XX.XX/24` 范围内

```
parameters:
  UserName: "Token:UserName"
  ClientIp: "System:CaClientIp"
```

```
$UserName = 'Admin' and $CaClientIp in_cidr '47.47.XX.XX/24'
```

- 当前请求的用户Id是1001,1098,2011中的一个，且使用HTTPS协议请求

```
parameters:
  CaAppId: "System:CaAppId"
  HttpSchema: "System:CaHttpSchema"
```

```
$CaHttpScheme = 'HTTPS' and ($CaAppId = 1001 or $CaAppId = 1098 or $CaAppId = 2011)`
```

- 当应答StatusCode=200，包体 `json` 包体存在 `result_code` 且不为 `ok` 时

```
parameters:
  StatusCode: "StatusCode"
  ResultCode: "BodyJsonField:$.result_code"
```

```
$StatusCode = 200 and ($ResultCode <> null and $ResultCode <> 'ok')
```

5. 使用限制

- 单个插件中的参数定义个数不超过16个
- 单个表达式的字符数不超过512个字符
- BodyJsonField对请求或应答包体的限制为16K Bytes, 超过限制后将不会生效

4.流量控制插件

1. 概述

- 流量控制插件用于对API进行限流，流量控制插件可以对 `API` ， `App（访问的AK）` ， `用户（访问方的App归属用户）` ，以及 `自定义参数` 进行多种维度的限流。
- `流量控制插件` 目前支持两种配置模板：
  - **参数流控配置**，支持 `自定义参数` 的流控配置。
  - **基础流控配置**，与控制台上的 `流量控制` 功能保持兼容。
- `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，如果你绑定了 `流量控制插件` ，则 `流量控制策略会失效` 。
- 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。

2. 基础流控配置

2.1. 流控能力

基础流控支持以下的流控维度：

- **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，单位时间可选秒、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。

2.2. 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，MINUTE，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
```

3. 参数流控配置

参数流控可以针对用户的请求参数以及条件执行进行流控，参数流控配置支持如下特性：

- 支持秒、分钟、小时、天的流控维度
- 可以根据请求参数、系统参数设置条件，来执行不同的流控维度
- 可以使用单个参数、或多个参数的组合来设置流控
- 可以设置流控的范围为API或插件

3.1. 快速开始

有这样一个场景，我们希望执行如下的规则执行流控，针对每个访问的客户端IP地址，当用户使用了 `AppId:10001` 的Key做了签名认证时，设置流控为 `100请求/秒` ，对其他的场合为 `10请求/秒`

针对这个场景我们的插件配置文件为，这里我们使用 `yaml` 来配置插件。

```
---
scope: "PLUGIN"
#
# 这个流控依赖两个系统参数
# 1. 用户签名的AppId，通过系统参数CaAppId获取
# 2. 用户来源的ClientIP，通过系统参数CaClientIp获取
parameters:
  AppId: "System: CaAppId"
  ClientIP: "System: CaClientIp"
rules:
  # 第一条流控策略，当`AppId`为`10001`时生效，对每个ClientIP限流为`100/秒`
  - name: "Vip"
    condition: "$AppId = 10001"
    byParameters: "ClientIP"
    value: 100
    period: SECOND
  # 第二条流控策略名为`PerClientIP`，对每个ClientIP限流为`10/秒`
  - name: "PerClientIP"
    byParameters: "ClientIP"
    value: 10
    period: SECOND
```

3.1. 插件配置

参数流控插件使用 `yaml` 或等价的 `json` 格式进行插件元数据配置。

```
---
scope: "PLUGIN"                 # 流控插件的作用范围：目前可选为"PLUGIN", "API"
defaultLimit: 100               # 默认流控值，如果设置了默认流控值，
defaultPeriod: SECOND           # 默认流控周期
defaultErrorMessage: "Throttled by 100/SECOND"
parameters:                     # 参数列表，可用于流控的参数
  clientIp: "System:CaClientIp"
  userId: "Token:userId"
rules:
  - name: "ByClientIp"
    byParameters: "clientIp"
    condition: "$clientIp !in_cidr '61.7.XX.XX/24'"
    limit: 10
    period: MINUTE
    errorMessage: "Throttled by 10/MINUTE from ${clientIp}"
  - name: "每个用户限制10条/分钟,管理员除外"
    byParameters: "clientIp"
    condition: "$userId !like 'admin%'"
    limit: 10
    period: MINUTE
  - name: "每个IP限制10条/分钟"
    byParameters: "clientIp"
    condition: "$clientIp in_cidr '67.0.XX.XX/8'"
    limit: 10
    period: MINUTE
  - name: "每个用户限制15条/分钟"
    condition: "$userId !like 'admin%'"
    limit: 15
    period: MINUTE
    byParameters: "clientIp"
```

插件配置的字段说明如下：

- `scope`（必选）：流控插件的作用范围，支持`API`，`PLUGIN`两种取值，如果多个API都绑定了同一个插件，则`scope`的取值会影响流控策略的作用范围，比如：某个策略取值为`10次/秒`。
  - 当取值为`API`时：流控策略在每个API中分别生效，在本例子中，每个API的限流均为`10次/秒`
  - 当取值为`PLUGIN`时：所有绑定了本插件的API共享这个限流，如果本例子中的插件绑定了一堆API，则这一堆API的总流控限制为`10次/秒`
- `parameters`（必选）：参与流控的参数表，参考参数与条件表达式的使用文档中的描述
- `rules`（可选）：流控策略的列表，如果没有设置默认流控`defaultLimit`与`period`，则不能为空，每个流控策略包含以下字段：
  - `name`（必选）：流控策略的名称，合法值为`[A-Za-z0-9_-]+`，在同一个插件中保持唯一

- `byParameters`（必选）：流控参数，如果使用多个参数组合流控，以“，”分割，比如：`ClientIP`表示，针对每个`ClientIP`的取值分别进行流控，`UserId,Action`表示对这两个参数的组合取值分别进行流控
- `condition`（可选）：如果设置了条件，只有当条件符合时，才会执行此条流控策略
- `limit`（必选）：流控值，正整数，当为`-1`时表示当命中此条件时不需要流控
- `period`（必选）：流控周期，取值：`SECOND`，`MINUTE`，`HOUR`，`DAY`
- `errorMessage`（必选）：定制错误信息，可以以模板的方式来定义，在`parameters`中定义的参数，可以在`${Name}`的方式来配置

- `defaultLimit`（可选）：默认流控值，正整数
- `defaultPeriod`（可选）：流控周期，取值：`SECOND`，`MINUTE`，`HOUR`，`DAY`
- `defaultErrorMessage`（可选）：定制错误信息，当配置了定制的错误信息后，返回的`X-Ca-Error-Message`头会使用定制的错误信息，这个信息无法使用参数

3.2. 参数说明

流控插件支持以下位置的参数。

位置名称	适用范围	说明
Method	请求	HTTP请求方法（大写），如：`GET`，`POST`...
Path	请求	HTTP完整请求路径，如：`/path/to/query`
Header	请求	使用`Header:{Name}`获取名字为`{Name}`的HTTP头的第一个值
Query	请求	使用`Query:{Name}`获取QueryString中名字为`{Name}`的第一个值

位置名称	适用范围	说明
Form	请求	使用 `Form:{Name}` 获取请求Form中名字为 `{Name}` 的第一个值
Host	请求	使用 `Host:{Name}` 获取匹配到的泛域名模板参数
Parameter	请求	使用 `Parameter:{Name}` 获取用户API自定义参数中名字为 `Name` 的第一个值
System	请求	使用 `System:{Name}` 获取名字为 `{Name}` 的系统参数值
Token	请求	当处于 `jwt` ， `oauth2` 授权场景时，使用 `Token:{Name}` 获取token中名字为 `{Name}` 的值

3.3. 执行规则

API网关按照如下的顺序执行参数流控：

- 插件会使用 `parameters` 的配置，从请求上下文中获取参数表。
- 所有 `condition` 执行结果为 `true` 或未配置 `condition` 的策略都会被执行。
- 如果命中策略列表中有多条策略的 `byParameters` 配置相同，则选择配置顺序靠前的那一条执行，其余的策略不会生。

4. 配置样例

4.1. 基础流控配置样例

基础流控可支持API流控、不同AppKey、以及不同用户级别的流控。

```
---
unit: SECOND            # 默认的流控单位，支持：SECOND,MINUTE,HOUR,DAY
apiDefault: 50          # API整体流控
appDefault: 20          # (可选)针对每个APP的流控值，不大于API整体流控
userDefault: 30         # (可选)针对每个用户的流控值，不大于API整体流控
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(AppKey)进行的流控
    policies:
      - key: 10001      # 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: 3        # 特殊流控值，不大于API整体流控
      - key: 10003
        value: 40
  - type: "USER"        # 针对不同的阿里云账户进行的流控
    policies:
      - key: 102        # 阿里云账号ID，可点击阿里云控制台上角查看账号ID
        value: 10       # 特殊流控值，不大于API整体流控
      - key: 233
        value: 35
```

4.2. 按照源IP进行参数限流

在这个例子中，我们配置了的限流策略。

- 每个源IP允许 `100次/分钟` 的调用
- 客户端IP处于 `58.66.XX.XX/24` 范围时，不限制访问
- 对客户端IP为处于 `63.0.XX.XX` ， `73.0.XX.XX/24` 范围时，访问限制为 `5次/天`

```
---
scope: API              # 限流的作用范围，可选API，PLUGIN
parameters:             # 设置限流的参数，我们仅针对客户端IP进行限流，客户端IP从系统变量`CaClientIp
`中获取
  ClientIp: "System:CaClientIp"
rules:
  - name: whitelist     # 白名单策略，当客户端IP符合条件时，不执行限流
    condition: "$ClientIp in_cidr '58.66.XX.XX/24'"
    limit: -1           # `-1`表示不进行限流
  - name: banList       # 特殊限制策略，当客户端IP符合条件时，按照`ClientIp`参数执行每天5次的限流
    condition: "$ClientIp in_cidr '63.0.XX.XX' or $ClientIp in_cidr '73.0.XX.XX/24'"
    byParameters: "ClientIp"
    limit: 5
    period: DAY
  - name: 100perIp      # 默认策略，每个IP，100次/分钟访问
    byParameters: "ClientIp"
    limit: 100
    period: MINUTE      # 周期，支持：SECOND, MINUTE, HOUR, DAY
```

4.3 防CC攻击配置

在这个例子中，我们配置如何防CC攻击。

- 每个源IP允许 `3次/秒钟` 的调用
- 当客户端源IP超出 `3次/秒钟` 范围时，访问将屏蔽10秒钟

```
---
scope: API              # 限流的作用范围，可选API, PLUGIN
defaultLimit: 3000        # 默认流控值
defaultPeriod: SECOND     # 默认流控周期
parameters:         # 设置限流的参数，我们仅针对客户端IP进行限流，客户端IP从系统变量`CaClientIp`中获取
  clientIp: "system:CaClientIp"
rules:
  - name: "每个源IP每秒只能访问3次，一旦触及阈值，屏蔽10秒钟"
    byParameters: "clientIp"
    limit: 3
    period: SECOND
    blockingPeriodBySecond: 10
```

5. 相关错误码

T429ID	429	Throttled by INNER DOMAIN Flow Control, ${Domain} is a test domain, only 1000 requests per day	当使用默认二级域名访问时，限制1000次/天，请绑定正式域名以解除这个限制
T429IN	429	Throttled by INSTANCE Flow Control	触发当前实例的流控限制
T429GR	429	Throttled by GROUP Flow Control	触发当前分组的流控限制
T429PA	429	Throttled by API Flow Control	触发插件上的默认API流控
T429PR	429	Throttled by PLUGIN Flow Control	触发插件的特殊流控
T429UP	429	Throttled by Usage Plan Flow Control	触发使用计划的流控

6. 使用限制

- 参数定义个数不超过16个。
- 单个表达式的字符数不超过512个字符。
- 插件元数据的大小限制为50KB。

- 每个插件中 `rules` 最大不超过16条。
- 每个 `rule` 中的 `byParameters` 最大不超过3个。
- 当限流参数的分散度太高时，可能释放掉实际使用数据不高的数据，比如：使用了天级别的源IP流控时，系统存储了过多的流控记录时，会释放掉一部分记录，在共享实例下的参数流控插件允许的不同参数个数为1000，在专享实例下的参数流控插件允许的不同参数个数为100000。

5.IP访问控制插件

IP访问控制插件 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。

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'
resource: "XFF:-1"    # 可选字段，如果设置本字段，取X-Forwarded-For头中的IP作为判断依据，本示例取XFF头的倒数第一个IP作为客户端源IP判断，
items:
- blocks:          # IP地址段
  - 61.3.XX.XX/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
  - 79.11.XX.XX     # 使用IP地址方式配置
- blocks:          # 用户VPC
  - 192.168.XX.XX/32     # 专享实例场景，从用户VPC内访问API网关的请求，API网关看到的来源地址处于这个地址段
```

需要特别注意最后一点，专享实例场景，API网关允许从用户VPC内发送请求到API网关，此时API网关可以直接读取来源VPC内的地址，比如192.168.XX.XX，用户设置黑白名单时可以直接使用VPC内地址作为配置值。

穿透WAF配置

如果API网关前面有WAF等中间件，业务需要实现API级别IP黑白名单能力时，就需要用到resource字段，resource字段为可选字段，如果不填写，取和网关直接连接前一跳的IP作为判断依据。如果填写本字段，允许设置X-Forwarded-For头中的值作为IP判断依据。

> **说明**
>
> WAF会将他收到的请求的源IP加入到X-Forwarded-For头的最后发送给API网关，API网关去判断X-Forwarded-For头中的这个值就可以。这种情况建议用户使用"XFF:-1"来判断WAF前一跳的IP地址。
>
> resource字段填写格式为“XFF:index”，index的值为X-Forwarded-For头中IP排序序号，从0开始算，允许负数，比如X-Forwarded-For的值为IP1,IP2,IP3，那么如果index值为0，取IP1，如果index的值为-1，取IP3，也就是倒数第一个IP。

跨VPC访问

在跨VPC访问的情况，API网关可以直接从请求中读取来源VPC内的IP地址，您可以放心编写针对VPC内IP的IP访问控制插件。API网关还可以读取到来源VPC的VPCID，您可以通过设置参数访问控制插件来设置只允许某些指定的VPC访问您的API。

接入WAF的同时跨VPC访问

有一种场景，用户的API需要同时对公网和内网提供服务，而在公网提供服务的时候，在API网关前接入了WAF中间件，需要API网关对WAF前的客户端做IP访问控制，同时需要对内网来源做IP访问控制。网关为了这种场景做了适配，用户可以用resource字段和allowResourceMissing字段配合使用来实现同时控制内网和使用WAF等中间件的公网的来源IP。用户通过设置resource字段来指定API网关从X-Forwarded-For头中获取透过WAF中间件过来的客户端的IP，通过allowResourceMissing字段来配置API网关遇到客户端没有传X-Forwarded-For头时的内网访问场景，从前一跳获取客户端的IP作为判断依据。下面是例子：

```
---
type: ALLOW            # 控制模式，支持白名单模式'ALLOW'和黑名单模式'REFUSE'
resource: "XFF:-1"    # 可选字段，如果设置本字段，取X-Forwarded-For头中的IP作为判断依据，本示例取XFF头的倒数第一个IP作为客户端源IP判断
allowResourceMissing: "true"  #允许Resource缺失，缺失时取API网关的前一跳的IP作为判断条件
items:
- blocks:          # IP地址段
  - 61.3.XX.XX/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
  - 79.11.XX.XX     # 使用IP地址方式配置
- blocks:          # 用户VPC
  - 192.168.XX.XX/32     # 专享实例场景，从用户VPC内访问API网关的请求，API网关看到的来源地址处于这个地址段
```

6.JWT认证插件

Json Web Token(RFC7519), 是一种便捷的可用于网关进行请求认证的鉴权方案， API 网关 可以通过托管用户的 Public JWK 实现对请求进行JWT认证，并将 claims 作为后端参数转发给后端，已方便用户简化开发后端应用的开发。本文主要是JWT认证插件的配置，如果需要了解JWT的token认证流程及基础知识可以参考基于JWT的token认证。

原有在API上配置的 OpenId Connect 功能目前可以通过 JWT认证插件 实现，推荐使用 JWT认证 插件配置，相比配置在API上的 OpenId Connect 配置， JWT认证 插件具备以下优势：

- 不再需要额外配置一个 授权API ，可以用任何方式来生成与分发 JWT ，API网关仅负责通过 公钥JWK 认证 JWT
- 支持不含 kid 的 jwk
- 支持配置多个 jwk
- 支持直接从 header 或 query 读取Token，不需要每个API都设置Token参数
- 支持从请求的cookie头的字段中读取Token
- 当 JWT 以 Authorization bearer {token} 方式传输时，可以通过 parameter: Authorization ， parameterLocation: header 方式配置已实现正确的Token读取
- 通过添加 preventJtiReplay: true 配置，可支持基于 claim:jti 的防重放检查
- 通过添加 bypassEmptyToken: true 配置，可在Token不存在时跳过验证直接转发给后端
- 通过添加 ignoreExpirationCheck: true 配置，可忽略Token的 exp 超期校验

如果你配置了 JWT认证插件 并绑定到了已配置了 OpenId Connect 功能的API上，原有 API 上的 OpenId Connect 的功能将被插件的配置覆盖。

1. 获取JWK (Json Web Key)

JWT认证插件通过Json Web Key(RFC7517)，实现JWT的签名与认证，配置 JWT认证插件 首先需要生成一个有效的 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，如mkjwk.org，一个可用的 Json Web Key 大概如下所示，其中私钥用于对Token进行签名，公钥需要配置在 JWT认证 插件中用于对Token进行签名，一个合法的JWK大概格式如下：

```
{
  "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JWT认证插件支持以下算法：

签名算法	支持的 `alg` 取值
RSASSA-PKCS1-V1_5 with SHA-2	RS256, RS384, RS512
Elliptic Curve (ECDSA) with SHA-2	ES256, ES384, ES512
HMAC using SHA-2	HS256, HS384, HS512

> **注意**
>
> 当配置HS256,HS384,HS512类型的Key时，密钥需要为Base64 UrlEncode后的值，如遇到Invalid Signature问题，请检查您的Key的格式是否与生成Token的Key一致

2. 插件配置

您可以选择json或者yaml格式的来配置您的插件，两种格式的schema相同，请搜索 `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`
preventJtiReplay: false       # 是否开启针对`jti`的防重放检查，默认：false
bypassEmptyToken: false       # 当`jwt`为空时是否允许验证通过
ignoreExpirationCheck: false # 是否允许忽略`exp`超期时间的检查
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`
#
# `Json Web Key`的`Public Key`
jwk:
  kty: RSA
  e: AQAB
  use: sig
  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，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: qSVxcknOm0uCq5v....
- 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: qSVxcknOm0uCq5v...
```

- `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

- 如果使用Authorization头传输Token，如：`Authorization bearer {token}`，可以通过`parameter: Authorization`，`parameterLocation: header`方式配置，JWT插件可以正确读取到Token的取值
- 当配置了`preventJtiReplay: true`时，插件会使用`claims`中的`jti`字段进行防重放检查
- 当配置了`bypassEmptyToken: true`时，当Token参数为空时，允许跳过检查，直接转发请求给后端
- 当配置了`ignoreExpirationCheck: true`时，会跳过`exp`的超期检查，否则网关会检查Token是否已超期
- 如果需要API网关将Token中的`claims`转发给后端应用，可以通过`tokenParameters`字段配置转发参数
  - `claimName`：token中claim的名字，当配置HS256,HS384,HS512类型的Key时，密钥为Base64 UrlEncode后的值，如遇到Invalid Signature问题，请检查您的Key的格式是否与生成Token的Key一致支持配置`kid`
  - `parameterName`：转发到后端的参数名称
  - `location`：转发到后端的参数位置，支持`header`，`query`，`path`，`formData`
    - 当配置为`path`时，后端path必须包含同名的参数，如`/path/{userId}`
    - 当配置为`formData`时，仅支持在**参数映射**且包体为`form`格式下才能正确映射到后端的form中
- 可以在`jwk`字段中配置单个Key，或在`jwks`字中配置多个Key
  - 不包含`kid`字段的Key只允许配置一个
  - 包含`kid`的Key可以配置多个，但`kid`必须唯一

3. 校验规则

- 插件会通过配置的`paramater`与`parameterToken`来获取到Token, 如果希望在Token不存在时仍然转发请求给后端，需要配置`bypassEmptyToken: true`
- 当配置多个Key是的选择原则为
  - 优先选择与请求Token中`kid`一致的Key来进行签名校验
  - 不含`kid`的Key只能配置一个，当不存在`kid`与请求Token一致的情况时，使用不含`kid`的Key执行签名校验

- 如果没有配置不含 `kid` 的Key，如果请求Token中未包含 `kid` ，或通过 `kid` 未匹配到Key则报错 `A403JK`

- 如果Token中包含了 `iat` ， `nbf` ， `exp` 字段，JWT插件会校验时间字段的合法性
- 默认情况下，API网关会校验Token的 `exp` 过期时间字段，如果希望跳过 `exp` 的超期检查，需要配置 `ignoreExpirationCheck: true`
- 如果配置了 `tokenParameters` 字段转发，当Token的 `claims` 中包含对应的字段时， `claim` 字段会转发给后端，不存在的 `claim` 不会转发

4. 配置案例

4.1. 配置单个JWK

```
---
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
  alg: RS256
  n: qSVxcknOm0uCq5vGsOmaorPDzHUubBmZZ4UXj-9do7w9X1uKFXAnqfto4TepSNuYU2bA_-tzSLAGBsR-BqvT6w9SjxakeiyQpVmexxnDw5WZwpWenUAcYrfSPEoNU-0hAQwFYgqZwJQMN8ptxkd0170PFauwACOx4Hfr-9FPGy8NCoIO4MfLXzJ3mJ7xqgIZp3NIOGXz-GIAbCf13ii7kSStpYqN3L_zzpvXUAos1FJ9IPXRV84tIZpFVh2lmRh0h8ImK-vI42dwlD_hOIzayL1Xno2R0T-d5AwTSdnep7g-Fwu8-sj4cCRWq3bd61Zs2QOJ8iustH0vSRMYdP5oYQ
```

4.2. 配置多个JWK

```
---
parameter: Authorization   # 从Authorization头中获取Token
parameterLocation: header  # 从Authorization头中获取Token
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: true     # 是否开启针对`jti`的防重放检查，默认：false
jwks:
- kid: O9fpdhrViq2zaaaBEWZITz         # 配置多个JWK时，需要使用不同的`kid`
  kty: RSA
  e: AQAB
  use: sig
  alg: RS256
  n: qSVxcknOm0uCq5v....
- kid: 10fpdhrViq2zaaaBEWZITz         # 配置多个JWK时，需要使用不同的`kid`
  kty: RSA
  e: AQAB
  use: sig
  alg: RS256
  n: qSVxcknOm0uCq5v...
```

4.3. 从请求的Cookie中的字段中读取Token

```
---
parameter: cookie          # 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`
parameterSection: token    # 例如cookie的值为 username=tom ; token=abcsef
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: true     # 是否开启针对`jti`的防重放检查，默认：false
jwks:
- kid: O9fpdhrViq2zaaaBEWZITz         # 配置多个JWK时，需要使用不同的`kid`
  kty: RSA
  e: AQAB
  use: sig
  alg: RS256
  n: qSVxcknOm0uCq5v....
- kid: 10fpdhrViq2zaaaBEWZITz         # 配置多个JWK时，需要使用不同的`kid`
  kty: RSA
  e: AQAB
  use: sig
  alg: RS256
  n: qSVxcknOm0uCq5v...
```

有些WEB场景，为了安全考虑，用户想把Token放在Cookie的一个指定的字段保存，API网关的JWT插件支持从请求的Cookie的字段中读取Token，使用parameterSection参数特性就可以指定字段名称了。比如请求的Cookie头如下所示，API网关就可以把Cookie中的Token提取出来。

```
Cookie: acw_tc=123; token=0QzRCMDBBQUYwRjE1MjYxQzU0QjY4NEM5MTc1NTQ1OUVCOTIzNzA4RDk3MDg5MzlDOTMQTVENDZCRUI1NkYyMEUyO; csrf=073957d8d2823be4f6c0cad23c764558
```

5. 相关错误码

Status	Code	Message	Description
400	I400JR	JWT required	未找到JWT参数
403	S403JI	Claim `jti` is required when `preventJtiReplay:true`	当在 `JWT授权插件` 中配置了防重放功能时，请求未提供有效的 `jti`

Status	Code	Message	Description
403	S403JU	Claim `jti` in JWT is used	当在`JWT授权插件`中配置了防重放功能时，请求提供的`jti`已被使用
403	A403JT	Invalid JWT: ${Reason}	请求中提供的`JWT`非法
400	I400JD	JWT Deserialize Failed: `${Token}`	请求中提供的`JWT`解析失
403	A403JK	No matching JWK, `kid:${kid}` not found	请求`JWT`中的`kid`没有匹配的`JWK`
403	A403JE	JWT is expired at `${Date}`	请求中提供的`JWT`已过期
400	I400JP	Invalid JWT plugin config: ${JWT}	`JWT授权`插件配置错误

当出现非预期应答码是，请检查HTTP应答中的`X-Ca-Error-Code`头中获取`ErrorCode`，从`X-Ca-Error-Message`头中获取`ErrorMessage`当出现`A403JT`或`I400JD`错误码时，可访问jwt.io网站来检查自己的Token合法性与格式。

6. 使用限制

- 插件元数据的大小限制为**50KB**。
- 转发参数列表最大限制为**16**个，`claimName`与`parameterName`长度不超过**32**个字符, 仅支持[A-Za-z0-9-_]。
- JWK的`alg`支持列表为：RS256, RS384, RS512, ES256, ES384, ES512, HS256, HS384, HS512。

7.BasicAuth插件

关于BasicAuth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。

```
---
users:
  - username: alice
    password: 123456
  - username: bob
    password: 666666
  - username: charlie
    password: 888888
  - username: dave
    password: 111111
```

8.跨域资源访问（CORS）

关于CORS跨域资源访问的基础知识可参考API网关实现跨域资源共享（CORS）。

插件配置

可以选择JSON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
```

9.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。

1. 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` , `Content-Encoding` , `Content-Language` 头，如果你需要更多的Headers, 可使用配置中的 `cacheableHeaders` 配置添加

2. 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
```

3. 运行规则

- 当API网关命中Cache后，返回的应答中会包含头 `X-Ca-Caching: true`

4. 使用限制

- 插件元数据的大小限制为**50KB**。
- 超过128K的应答BODY不会被缓存。
- 对于共享实例来讲，每个用户每个Region的总限制为5M Bytes，专享实例请参考实例规格说明。

10.后端路由

1. 概述

Routing插件具备根据请求的参数取值与系统参数取值，变更后端地址、路径、参数或类型的能力。可以用于租户路由，蓝绿发布，不同环境的区分等场合。

2. 配置方式

2.1 配置模板

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。

```
---
routes:
# 当调用方的AppId为"123456"时，路由至独立的地址，例子中指定了VPC授权名为"slbAddressForVip"的地址
- name: Vip
  condition: "$CaAppId = 123456"
  backend:
    type: "HTTP-VPC"
    vpcAccessName: "slbAccessForVip"
# 针对太老的客户端，返回不再支持的应答，ClientVersion是API中的自定义参数
- name: MockForOldClient
  condition: "$ClientVersion < '2.0.5'"
  backend:
    type: "MOCK"
    statusCode: 400
    mockBody: "This version is not supported!!!"
# 蓝绿发布场景：将5%的请求路由给蓝绿发布后端
- name: BlueGreenPercent05
  condition: "Random() < 0.05"
  backend:
    type: "HTTP"
    address: "https://beta-version.api.foo.com"
  constant-parameters:
  - name: x-route-blue-green
    location: header
    value: "route-blue-green"
```

配置模板的根对象为 `routes` ，包含多个 `route` 对象，每个 `route` 对象用于指定一条路由，每条路由以下几个部分组成：

- **name**：表示这个路由的名称，大小写英文字母和数字，在每个插件中保持唯一，如果命中了这条路由，后端收到的请求会包含一个名为 `X-Ca-Routing-Name` 的HTTP头，指示本次命中的路由名称
- **condition**：路由的条件表达式，条件表达式的规范见本文2.2节中的描述，如果当前的请求符合条件表达式，则本条路由命中，Routing插件会按照插件的配置顺序从上到下进行判断，并将请求路由给命中的第一条记录，并忽略后面的记录，如果您配置了多条路由记录，请仔细检查配置的顺序与您期望的逻辑一致。

- **backend**: 后端描述，与API网关的Swagger标准保持一致，请参考[]，后端在原有API的后端上覆盖backend配置后处理，如果覆盖后的backend不完整，会提示客户端 `X-Ca-Error-Code: I504RB` ，当看到这个错误时，请检查自己的backend配置是否完整，关于Backend描述，请参照2.3节，Routing插件中的Backend覆盖规则
- **constant-parameters**: 可以自行设置路由的常量参数，常量参数会附加在命中路由的后端请求上，用于后端的业务逻辑处理，location支持 `header` 和 `query` 。

2.2. Condition条件表达式

2.2.1 基本语法

- Condition表达式与SQL表达式类似，基础表达形式为 `$A = 'A' and '$B = 'B'`
- 参数以 `$` 开头，可以引用在API中定义的参数（API为映射与透传方式均可），如果您在API中定义了名为query1的参数，则 `$query1` 可以表示当前请求中的query1参数s
- 支持以下常量类型：
  - `STRING` : 字符串类型，支持单引号或双引号，如： "Hello"
  - `INTEGER` : 整数类型，如：1001，-1
  - `NUMBER` : 浮点数类型，如： 0.1, 100.0
  - `BOOLEAN` : 布尔类型，如：true、false
- 可以使用 `and` 、 `or` 来连接不同的表达式
- 可以用小括号 `(` ， `)` 来指定条件判断的优先级
- `Random()` 作为内置函数，可以产生一个 `0-1` 的 `NUMBER` 浮点类型参数，用于随机的判断
- 可以使用 `$CaAppId` 形式来引用当前请求的系统参数，系统参数不需要API中被定义即可引用，但如果您在API中定义了重名的参数，取到的值会被自定义参数覆盖，可用于路由插件的系统参数列表如下：
  - CaStage: 当前API请求的环境， `RELEASE` ， `PRE` ， `TEST`
  - CaDomain： 当前请求的域名
  - CaRequestHandleTime：请求处理的UTC时间
  - CaAppId：请求参数的AppId
  - CaAppKey：请求参数的AppKey
  - CaClientIp：客户端IP
  - CaApiName：创建时候的API名字
  - CaHttpScheme：请求的协议 `HTTP` ， `HTTPS` ， `WS`

- CaClientUa: 客户端上传的UserAgent字段

- 如果表达式中使用了不存在的参数，如 `$UnknonwParameter = 1` ，则表达式的判断会返回false

2.2.2 条件表达式样例

- 5%的几率为真

```
Random() < 0.05
```

- 当前API请求的是测试环境

```
$CaStage = 'TEST'
```

- 自定义参数中的UserName是Admin且来源IP是 `47.47.74.77`

```
$UserName = 'Admin' and $CaClientIp = '47.47.74.77'
```

- 当前请求的用户ID是1001,1098,2011中的一个，且使用HTTPS协议请求

```
$CaHttpScheme = 'HTTPS' and ($CaAppId = 1001 or $CaAppId = 1098 or $CaAppId = 2011)
```

2.3. Backend的定义与覆盖规则

Backend的结构与API网关导入Swagger时的结构一致，详细请参考文档通过导入Swagger创建API，目前支持的后端类型及配置样例如下，如果插件中不需要变更后端类型，只需要变更部分设置，则不需要填写完整的字段，仅填写需要变更的即可。

- HTTP

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

- HTTP-VPC

```
---
backend:
  type: HTTP-VPC
  vpcAccessName: vpcAccess1
  path: "/users/{userId}"
  method: GET
  timeout: 10000
```

- FC

```
---
backend:
  type: FC
  fcRegion: cn-shanghai
  fcType: FCEvent
  serviceName: fcService
  functionName: fcFunction
  roleArn: "acs:ram::111111111:role/aliyunapigatewayaccessingfcrole"
backend:
  type: FC
  fcRegion: cn-shenzhen
  method: GET
  fcType: HttpTrigger
  fcUrl: https://1833848375796824.cn-shenzhen.fc.aliyuncs.com/2016-08-15/proxy/servicetest/
fctest3/fctest3
  roleArn: acs:ram::1833848375796824:role/aliyunapigatewayaccessingfcrole
```

- MOCK

```
---
backend:
  type: MOCK
  mockResult: "mock resul sample"
  mockStatusCode: 200
  mockHeaders:
    - name: server
      value: mock
    - name: proxy
      value: GW
```

2.4. 限制

- 插件的文本大小限制为**16384字节**，超过后会报错 `InvalidPluginData.TooLarge`
- 单个Routing插件的条数限制为**160条**，超过后会报错 `InvalidPluginData.TooManyRoutes` **（当这个插件绑定至共享实例API时只有前16条生效，绑定至专享实例API时全部生效）**
- 单条Condition语句的大小限制为**512字节**，超过后会报错 `InvalidPluginData.ConditionTooLong`
- 插件更新会实时生效至所有已绑定该插件的API，插件的更新频率被限制为**45秒一次**，超出时会报错，`InvalidPluginData.UpdateTooBusy`

3. 典型应用场景

3.1. 配置租户路由（根据AppId分配不同的后端地址）

AppId为10098,10099的用户是我的VIP用户，我希望来自这两个用户的请求路由到独立的服务器集群去。

```
---
routes:
# 当调用方的AppId为10098或10099时，路由至独立的地址，
# 例子中指定了VPC授权名为"slbAddressForVip"的地址
- name: Vip
  condition: "$CaAppId = 10098 or $CaAppId = 10099"
  backend:
    type: "HTTP-VPC"
    vpcAccessName: "slbAccessForVip"
```

3.2. Stage路由（测试/预发/生产环境变更）

我的正式环境在阿里云的VPC中，但我希望将所有访问测试环境的API指向我的公网测试服务器。

```
---
routes:
# 当访问测试环境时，将路由转向我的公网服务器
- name: Vip
  condition: "$CaStage = 'TEST'"
  backend:
    type: "HTTP"
    address: "https://test-env.foo.com"
```

3.3. 蓝绿发布

我准备将5%的流量指向一组beta服务器地址，进行蓝绿发布。

```
---
routes:
# 蓝绿发布场景：将5%的请求路由给蓝绿发布后端
- name: BlueGreenPercent05
  condition: "Random() < 0.05"
  backend:
    type: "HTTP"
    address: "https://beta-version.api.foo.com"
```

11.参数访问控制插件

1. 概述

访问控制插件可以根据请求参数或上下文，来执行条件判断，用于过滤不希望传递到后端的请求，参考参数与条件表达式的使用来了解如何定义参数和使用条件表达式。

2. 配置说明

在这个例子中假设我们的API请求Path为 `/{userId}/...` , API使用JWT认证，JWT中有userId和userType两个claim, 我们这个插件的校验条件为：

- 当userType=admin时，允许所有的路径。
- 当userType=user时，仅允许/{userId}路径一致的请求。

```
---
#
# 在这个例子中假设我们的API请求Path为`/{userId}/...`,
# API使用JWT认证，JWT中有userId和userType两个claim
# 我们这个插件的校验条件为
# - 当userType=admin时，允许所有的路径，
# - 当userType=user时，仅允许/{userId}路径一致的请求
parameters:
  userId: "Token:userId"
  userType: "Token:userType"
  pathUserId: "path:userId"
#
# 关于Rules的处理规则，依次演算条件，按照返回值为`true`或者`false`，处理`ifTrue`的逻辑或`ifFalse`的结果
# `ALLOW`会直接判断为成功，而`DENY`则会直接返回错误代码给客户端，
# 如果没有触发`ALLOW`或`DENY`逻辑，则执行下一条
rules:
  - name: admin
    condition: "$userType = 'admin'"
    ifTrue: "ALLOW"
  - name: user
    condition: "$userId = $pathUserId"
    ifFalse: "DENY"
    statusCode: 403
    errorMessage: "Path not match ${userId} vs /${pathUserId}"
    responseHeaders:
      Content-Type: application/xml
    responseBody:
      <Reason>Path not match ${userId} vs /${pathUserId}</Reason>
```

3. 相关错误码

错误代码	Http状态码	Message	描述
A403AC	403	Access Control Forbidden by ${RuleName}	被授权控制插件阻止。

4. 使用限制

- 参数定义个数不超过160个。
- 单个表达式的字符数不超过1024个字符。
- 插件配置大小限制为50KB。
- 最大允许的 `rules` 条数为160条。

> **说明**
>
> 当这个插件绑定至共享实例API时，参数定义及rules只有前16条生效，绑定至专享实例API时全部生效）

12.断路器插件

1. 断路器概述

断路器是API网关在后端出现性能问题时保护系统的内置机制，默认配置下，当某个API的后端在30秒钟内出现1000次超时，系统会触发断路器保护，断路器进入打开状态，断路时间为90秒，90秒内所有的请求均快速返回Status=503,X-Ca-Error-Code=D503CB错误码，90秒后断路器进入半开状态，允许少量并发请求通过，如果后端恢复正常，则断路器置为关闭状态，请求恢复正常。

当您使用 `专享实例` 时，可以通过 `断路器插件` 来定制 `断路器` 配置，可定制的配置如下：

- 断路器触发条件，可配置超时模式或者错误码模式
- 配置超时窗口
- 配置断路器打开状态的超时时间
- 配置断路器打开后的降级后端

2. 配置规则

断路器插件配置仅在API运行在专享实例时生效，如果您使用的是共享实例，则即使API绑定了断路器插件，也只会使用默认断路器配置。

2.1 按后端超时配置降级策略

可以按照后端超时的方式配置降级策略。若API定义的后端超时时间为10s，请求后端10s没有应答就会计为后端超时

```
timeoutThreshold: 15          # 超时的阈值
windowInSeconds: 30           # 超时时间窗口
openTimeoutSeconds: 15        # 断路器开的时间
downgradeBackend:             # 降级后的后端配置
  type: mock
  statusCode: 418
```

配置字段说明：

- `timeoutThreshold`：超时阈值，上限为5000，注意如果配置过低，可能导致几次超时就触发断路器。
- `windowsInSeconds`：判断时间窗口，合法值为10~90秒。
- `openTimeoutSeconds`：断路器开的持续时间，合法值为15~300秒。
- `downgradeBackend`：（可选）当断路器出现降级时，降级后端。

2.2 按后端响应时间配置降级策略

可以按照后端的响应时间配置降级策略。后端响应时间为网关给后端发送请求到收到后端应答的耗时。

```
---
errorThreshold: 10          # 错误出现的阈值
windowInSeconds: 60             # 判断错误次数的时间窗口
openTimeoutSeconds: 120          # 断路器打开的持续时间
errorCondition: "$LatencyMilliSeconds > 500"      # 错误条件：后端响应时间大于500ms
downgradeBackend:               # 降级后的后端配置
  type: mock
  statusCode: 403
```

配置字段说明：

- `errorThreshold`：错误出现的阈值。
- `windowsInSeconds`：判断时间窗口，合法值为10~90秒。
- `openTimeoutSeconds`：断路器开的持续时间，合法值为15~300秒。
- `errorCondition`：错误条件表达式，`$LatencyMilliSeconds` 和 `$LatencySeconds` 两个变量可以用于对后端耗时进行判断。`$LatencyMilliSeconds` 的单位为毫秒（ms），`$LatencySeconds` 的单位为秒（s）。
- `downgradeBackend`：（可选）当断路器出现降级时，降级后端。

2.3 按后端报错配置降级策略

可以使用按照后端错误码的方式配置降级策略。

```
errorCondition: "$StatusCode == 503"  # 错误条件
errorThreshold: 1000                  # 错误阈值
windowInSeconds: 30                   # 超时窗口时间
openTimeoutSeconds: 15                # 断路器开的时间
downgradeBackend:                     # 降级后端
  type: "HTTP"
  address: "http://api.foo.com"
  path: "/system-busy.json"
  method: GET
```

- `errorCondition`：错误条件表达式，`$StatusCode` 和 `$LatencySeconds` 两个变量可以用于对后端的应答码以及耗时（秒）进行判断。
  - 如：`$StatusCode = 503 or $StatusCode = 504`，当后端应答为503或504。
  - 如：$LatancySeconds > 30，当超时大于30秒时。
- `errorThreshold`：错误出现的阈值。
- `windowsInSeconds`：判断时间窗口，合法值为10~90秒。
- `openTimeoutSeconds`：断路器开的持续时间，合法值为15~300秒。

- `downgradeBackend`：（可选） 当断路器出现降级时，降级后端。

3. 降级后端配置

当断路器打开时，可以通过配置 `downgradeBackend` 来配置断路器打开后的返回，返回的结构与API网关的Swagger结构一致，详细请参考文档通过导入Swagger创建API，目前支持的后端类型及配置样例如下：

- HTTP后端

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

- HTTP后端(VPC)

```
---
backend:
  type: HTTP-VPC
  vpcAccessName: vpcAccess1
  path: "/users/{userId}"
  method: GET
  timeout: 10000
```

- 函数计算

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

- MOCK

```
---
backend:
  type: MOCK
  mockResult: "mock resul sample"
  mockStatusCode: 200
  mockHeaders:
    - name: Content-Type
      value: text-plain
    - name: Content-Language
      value: zhCN
```

4. 相关错误码

错误代码	Http状态码	Message	描述
D503BB	503	Backend circuit breaker busy	API被断路器阻止。
D503CB	503	Backend circuit breaker open, ${Reason}	API处于熔断/断路器开状态，请检查后端性能后稍后再试。

5. 使用限制

- 仅专享实例生效。
- 单个表达式的字符数不超过512个字符。
- 插件配置大小限制为50KB。

13.错误码映射插件

错误码映射插件 用于将后端应答中返回的非正常请求，映射为客户端期望的错误应答的场景

1. 概述

错误码映射插件用于将后端应答中返回的非正常请求，映射为客户端期望的错误应答的场景。

2. 快速开始

请先参考下面的例子，某后端的返回中，HTTP应答码为200, 但错误信息包含在包体中json字段中。

```
HTTP 200 OK
Content-Type:application/json

{"req_msg_id":"d02afa56394f4588832bed46614e1772","result_code":"ROLE_NOT_EXISTS"}
```

- 在这个场景中，客户端期望得到非200的应答，但希望通过不修改后端的方式实现

```
HTTP 404
X-Ca-Error-Message: Role Not Exists, ResultId=d02afa56394f4588832bed46614e1772
```

针对描述的中的场景，我们可以按照如下的方式配置错误码映射插件

```
---
# 参与映射的字段
parameters:
  statusCode: "StatusCode"
  resultCode: "BodyJsonField:$.result_code"
  resultId: "BodyJsonField:$.req_msg_id"
# 映射条件
errorCondition: "$statusCode = 200 and $resultCode <> 'OK'"
# 错误码字段
errorCode: "resultCode"
# 映射项
mappings:
  - code: "ROLE_NOT_EXISTS"
    statusCode: 404
    errorMessage: "Role Not Exists, RequestId=${resultId}"
  - code: "INVALID_PARAMETER"
    statusCode: 400
    errorMessage: "Invalid Parameter, RequestId=${resultId}"
# 默认映射（可选）
defaultMapping:
  statusCode: 500
  errorMessage: "Unknown Error, ${resultCode}, RequestId=${resultId}"
```

在这个例子中，我们将后端应答码以及后端应答JSON包体中的 `result_code` 字段设置为了应答条件，如果后端应答码为200，但 `result_code` 字段不等于 `'OK'` 时，开始执行错误码映射，这个例子中，我们使用 `result_code` 作为错误码字段执行映射，配置了两个错误码：当 `ROLE_NOT_EXISTS` 会返回给客户端404应答，而 `INVALID_PARAMETER` 会返回给客户端400应答，其他错误码返回500应答。

3. 插件配置与映射规则

3.1. 插件配置

错误映射插件配置使用 `json` 或 `yaml` 格式配置，各个字段的详细说明如下：

- `parameters`（必选）：下面配置参与映射的字段，以 `map` 方式配置，参考文档参数与条件表达式的使用
- `errorCondition`（必选）：应答是否属于错误应答的条件表达式，当执行结果为 `true` 时，执行映射
- `errorCode`（可选）：用于指定 **错误代码** 字段，用于匹配 `mappings` 映射列表中的 `code` 字段
- `mappings`（必选）：用于指定 **映射记录** 列表，网关将会用符合 **错误代码** 或 **错误条件** 记录重新构造返回应答，详细字段为
  - `code`（可选）：唯一，设置这个字段是 `codeParameter` 字段为必选，当 **错误代码** 与 `code` 字段一致时，执行当前 **映射记录**
  - `condition`（可选）：错误条件表达式，当表达式演算为 `true` 时，执行当前 **映射记录**
  - `statusCode`（必选）：用于指定当前 **映射记录** 的HTTP返回码
  - `errorMessage`（可选）：用于指定当前 **映射记录** 的错误信息，体现在应答的 `X-Ca-Error-Message` 头及日志的 `errorMessage` 字段中
  - `responseHeaders`（可选）：以Map方式配置，用于设置当前 **映射记录** 的应答头
  - `responseBody`（可选）：用于覆写当前 **映射记录** 的应答包体
- `defaultMapping`（可选）：默认 **映射记录**，如果 `mappings` 中的所有记录均未命中，则使用本条记录作为应答
  - `statusCode`（必选）：用于指定当前 **映射记录** 的HTTP返回码
  - `errorMessage`（可选）：用于指定当前 **映射记录** 的错误信息，体现在应答的 `X-Ca-Error-Message` 头及日志的 `errorMessage` 字段中
  - `responseHeaders`（可选）：以Map方式配置，用于设置当前 **映射记录** 的应答头

- `responseBody`（可选）：用于覆写当前 **映射记录** 的应答包体

配置规则：

- `mappingCondition`，`mappings[].condition` 中的条件表达式使用的参数，与 `parameters` 字段中定义的字段必须对应，否则会报错，关于参数定义与条件表达式，参考参数与条件表达式的使用文档
- `errorCode` 字段中使用的参数必须为 `parameters` 中定义的字段
- `mappings` 列表记录中的 `code` 与 `condition` 必须配置其中的一个，当配置 `code` 时值必须唯一，当配置 `condition` 时，将按照配置顺序执行，越靠前的优先级越高
- `errorMessage`，`responseBody` 可以使用类似 `"${Code}: ${Message}"` 的格式对消息进行模板替换，字段取值来自 `parameters` 中提取到的值
- `responseHeaders` 的值，也可以使用 `${Message}` 的方式执行模板替换
- `responseBody` 不配置时，透传后端应答
- `responseHeaders` 不配置时，透传后端应答的Headers, 否则使用配置键值对覆盖后端应答的Header，当值配置为 `''` 时，删除对应的Header
- `defaultMapping` 没有配置时，错误码映射不生效，透传后端应答

3.2. 映射参数

映射参数可通过如下的方式在 `parameters` 参数中配置为键值对，其中键作为变量名定义，值可以使用 `Location:Name` 的方式配置，表示从当前应答或系统上下文的特定位置中读取。

```
---
# 参与映射的字段
parameters:
  statusCode: "StatusCode"
  resultCode: "BodyJsonField:$.result_code"
  resultId: "BodyJsonField:$.req_msg_id"
```

错误码映射目前支持以下位置的参数，具体参数描述请参考参数与条件表达式的使用文档。

位置名称	适用范围	说明

位置名称	适用范围	说明
StatusCode	应答	后端的HTTP应答码，如：`200`，`400`
ErrorCode	应答	API网关系统错误码
ErrorMessage	应答	API网关系统错误消息
Header	应答	使用`Header:{Name}`获取名字为`{Name}`的HTTP头的第一个值
BodyJsonField	应答*	使用`BodyJson:{JPath}`以`JSONPath`方式获取请求/应答包体中的JSON字段值
System	应答	使用`System:{Name}`获取名字为`{Name}`的系统参数值
Token	应答	当处于`jwt`，`oauth2`授权场景是，使用`Token:{Name}`获取token中名字为`{Name}`的值

- `ErrorCode`和`ErrorMessage`会返回API网关的系统错误码与系统错误信息，请参考错误代码表文档。
- 使用`BodyJsonField`可以使用JSONPath来提取后端返回json的值，但如果后端应答的包体大小超过15360 Bytes则这个字段将会无法提取到值，得到的是空值`null`

3.3. 执行规则

错误码映射插件会按照如下的顺序执行：

- i. 根据 `parameters` 中配置的参数列表，从应答及系统上下文中获取当前的参数表
- i. 依据 `步骤1` 得到的参数表，执行 `errorCondition` 中配置的条件表达式，为 `true` 则继续执行，`false` 不生效
- i. 如果配置了 `errorCode` 字段，则获取 `errorCode` 字段的值，并寻找与 `mappings` 中 `code` 匹配的 `映射记录` ，
- i. 如果步骤3没有匹配到 `映射记录` ，则依次执行 `mappings` 字段配置了 `condition` 的 `映射记录`
- i. 如果步骤3或4中匹配到了 `映射记录` ，根据对应的配置构造应答，否则构造默认应答

3.4. 系统错误的映射及日志

- API网关系统错误码出现在网关的检查、校验、流控、插件处理等场合，参考错误代码表文档，使用 `ErrorCode` 作为映射参数，可以支持对系统错误码的映射，可用于诸如：客户端仅支持200应答，但希望将被限流的429应答映射为200应答的场景。
- 当出现系统错误时，位置为 `StatusCode` ， `Header` ， `BodyJsonField` 等来自应答的字段取值都将为 `null` ，在写条件表达式时需要注意，未出现系统错误时 `ErrorCode` 位置的取值为 `OK` 。
- API网关的系统错误码会出现在 `X-Ca-Error-Code` 应答头及日志的 `errorCode` 字段中，这个值不会被 `错误码映射插件` 改写。
- 日志中的 `statusCode` 字段记录的是网关递送给客户端的应答码的值，会被 `错误码映射插件` 改写。

4. 配置示例

4.1. 映射包体错误码

映射

```
---
# 参与映射的字段
parameters:
  statusCode: "StatusCode"
  resultCode: "BodyJsonField:$.result_code"
  resultId: "BodyJsonField:$.req_msg_id"
# 映射条件
errorCondition: "$statusCode = 200 and $resultCode <> 'OK'"
# 错误码字段
errorCode: "resultCode"
# 映射项
mappings:
  - code: "ROLE_NOT_EXISTS"
    statusCode: 404
    errorMessage: "Role Not Exists, RequestId=${resultId}"
  - code: "INVALID_PARAMETER"
    statusCode: 400
    errorMessage: "Invalid Parameter, RequestId=${resultId}"
# 默认映射(可选)
defaultMapping:
  statusCode: 500
  errorMessage: "Unknown Error, ${resultCode}, RequestId=${resultId}"
```

4.2 应答body映射

```
#
# 这个例子用于给前端返回定制的json错误body
---
# 指定映射参数
parameters:
  statusCode: "StatusCode"
  resultCode: "Header:X-Ca-Error-Code"
  requestId: "Header:X-Ca-Request-Id"
  errorMessage: "Header:X-Ca-Error-Message"

# 映射条件
errorCondition: "$statusCode != 200"
# 错误码字段
errorCode: "resultCode"
# 映射项
mappings:
  - code: "I400MH"
    statusCode: 200
    responseHeaders:
        Content-Type: "application/xml"
        X-Ca-Error-Message: ""
        X-Ca-Error-Code: ""
    responseBody: |
        {
            "code":"89",
            "message":"${errorMessage}",
            "resultCode":"${resultCode}"

        }
```

5. 使用限制

- 参数定义个数不超过16个。
- 单个表达式的字符数不超过512个字符。
- `BodyJsonField` 位置的字段对应答包体的限制为16380 Bytes, 超过后直接返回空值。
- 插件配置大小限制为50KB。
- `mappings` 中以 `condition` 方式配置 `映射记录` 不超过20条。