SOFAStack

API 网关

开发指南

产品版本：AntStack Plus 1.11.0

文档版本：20221009

蚂蚁集团
ANT GROUP

法律声明

蚂蚁集团版权所有©2022，并保留一切权利。

商标声明

免责声明

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

本文主要介绍网关的几个核心概念，以便更好地进行 API 服务开发与调用。

协议转换

API 网关不仅是一个简单的反向代理服务，同时还提供协议转换的能力，支持使用 A 协议调用 B 协议的接口。协议转换与使用的编程语言无关，但是仅 Java 语言支持使用高级功能，比如服务端签名校验、数据加解密等，其他语言目前只能使用基础能力。

API 网关将协议分为 DownProtocol 和 UpProtocol，如下图所示。

- **DownProtocol** 指的是从调用方发送请求到网关时使用的协议。当前 API 网关支持的 DownProtocol 包括：HTTP/HTTPS、SOFARPC、DUBBO。
- **UpProtocol** 指的是网关将请求转发到提供方使用的协议。当前 API 网关支持的 UpProtocol 包括：HTTP/HTTPS、SOFARPC、SOFAREST 和 DUBBO。

API 网关支持的 **协议转换** 如下：

- HTTP/HTTPS > HTTP/HTTPS
- DUBBO > HTTP/HTTPS
- HTTP/HTTPS > DUBBO
- HTTP/HTTPS > SOFARPC/SOFAREST
- SOFARPC > HTTP/HTTPS/SOFAREST
- SOFARPC > SOFARPC

请求认证

为了保证数据在传输过程中的安全性，API 网关支持对数据进行签名校验，包括网关对订阅方和网关之间数据的双向校验，以及提供方和网关之间数据的双向校验。其中，订阅方请求网关的时候必须对请求进行加签，而提供方无要求。

为了实现签名校验的能力，需要在发起请求时进行加签，收到响应后进行验签。当前 API 网关提供了以下语言的 SDK，方便用户使用：

- **API 服务提供者**：Java-SDK
- **API 服务订阅者**：js-signature-sdk、Java-SDK

其他语言的 SDK 会陆续推出，用户也可自行实现。

数据加密

统一网关提供了对客户端与网关之间传输的数据进行加密的能力，支持 **RSA**、**ECC**、**国密** 三种加密算法，保证数据传输过程中的安全性。

> **说明**
>
> 目前仅支持对 HTTP 类型的 API 调用进行数据加密。

2.加签验签

为保证应用与服务端之间的通信安全，网关支持对客户端和服务端分别进行加签验签操作。

客户端加签

客户端通过为应用生成 Access Key 和 Secret Key 来完成加签，操作如下：

1. 登录网关控制台。
2. 在左侧导航栏单击 **API 订阅** > **应用管理**。
3. 在应用管理页面单击需要加签的 **应用名称**。
4. 在**应用详情页** 单击 **密钥配置** 页签即可设置 Access Key 和 Secret Key 参数。

> **说明**
>
> 创建应用后，系统会自动为您生成一组 Access Key 和 Secret Key 参数，您可以单击 **密钥配置** 旁的修改按钮进行自定义修改，或者单击 **生成密钥**，系统会随机生成一组 Access Key 和 Secret Key 参数。

服务端加签

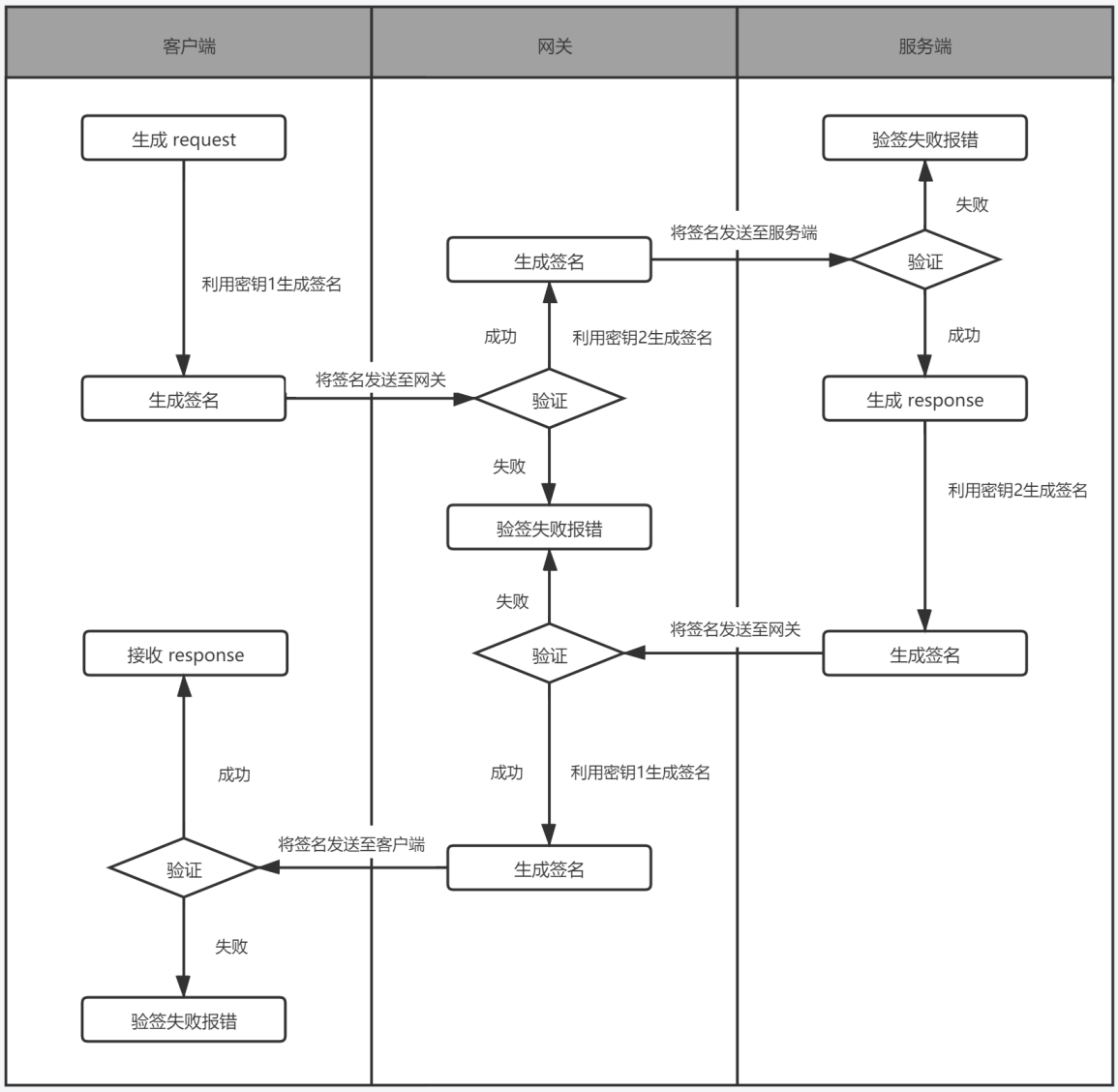
服务端通过为系统集群生成 Access Key 和 Secret Key 来完成加签，操作如下：

1. 登录网关控制台。
2. 在左侧导航栏单击 **API 发布** > **集群管理**。
3. 在应用管理页面单击需要加签的 **系统集群名称**。
4. 在 **集群详情页** 单击 **认证配置** 页签即可设置 Access Key 和 Secret Key 参数。

> **说明**
>
> 您可以单击 **认证配置** 旁的修改按钮对认证方式和参数进行自定义修改。如认证方式是 **密钥**，单击 **生成密钥**，系统会随机生成一组 Access Key 和 Secret Key 参数。

加签验签逻辑

当您在客户端和服务端都开启了密钥认证后，一个完整的加签验签流程如下：



说明

- 上述流程图的前提是客户端和服务端都开启了密钥认证，在实际应用过程中，您可以根据具体的业务需求来选择是否在客户端或服务端开启密钥认证。
- 密钥 1 指的是客户端即应用的 Access Key 和 Secret Key。
- 密钥 2 指的是服务端即系统集群的 Access Key 和 Secret Key

详细的加签验签流程如下：

1. 客户端生成 request 后，使用密钥1进行加签生成签名 A，将签名 A 和请求发送至网关。
2. 网关使用密钥 1 对签名进行验证：解析出计算签名 A 时用到的算法以及加签的 request ，网关用同样的算法，使用密钥 1 对同样的 request 进行加签，得到签名 B，网关比对 A 和 B，如果一样，则签名验证成功，否则，验签失败。
3. 签名验证成功后，网关会使用密钥 2 对 request 进行加签，生成签名 C，并同时向服务端发送。

4. 服务端使用密钥 2 对签名 C 进行验证：解析出计算签名 C 时用到的算法以及加签的 request ，服务端用同样的算法，对同样的 request 进行加签，得到签名 D，服务端比对 C 和 D，如果一样，则签名验证成功并生成 response，否则，验签失败。
5. 服务端生成 response，同样需要经过网关进行加签验签，具体流程与上述 1-4 步的客户端发送 request 流程相同。

注意事项

- 如果开启客户端签名认证，但在 SDK 中未打开校验开关，则会在网关拦截该请求，并报错。
- 如果未在客户端签名认证，但在 SDK 中打开校验开关，则会在客户端校验时，报错解析失败。
- 如果开启后端集群签名认证，且未开启客户端签名验证，同时在 SDK 逻辑里开启了客户端验签的开关，就会产生，客户端收到了网关的请求，该请求携带着服务端返回的签名信息，客户端会用客户端的密钥去验证服务端的 signature，从而导致签名认证失败。
- 当您创建 HTTP 类型的 API，并使用 SOFARPC 协议将请求转发到后端，即 HTTP 转 SOFARPC 时，此时服务端不支持加签功能。

3.获取服务端信息

为了让客户端与网关建立连接，您必须在 API 网关控制台上获取以下信息：

- **应用信息**
  - 应用标识 APPID
  - 应用验证密钥
- **API 信息**
  - API 域名地址
  - API 系统集群密钥
  - API 请求路径/接口/方法

应用信息

应用标识 APPID

在网关中，应用在环境中的唯一标识称为 APPID。您必须在客户端中设置调用方应用的 APPID 以建立连接。

操作步骤：

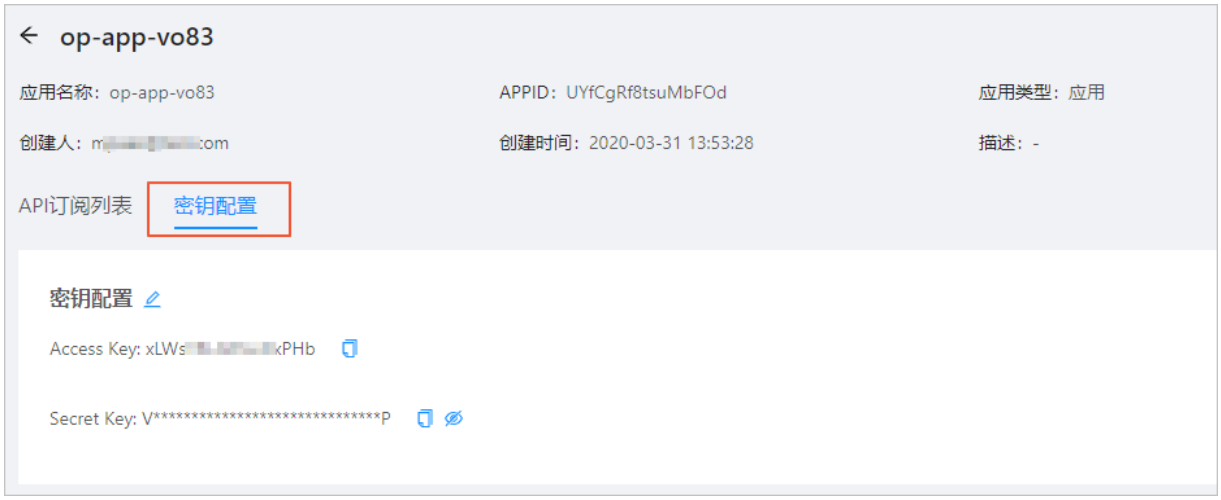
1. 进入 API 网关控制台 > **API 订阅** > **应用管理** 页面。
2. 在列表中，即可找到目标应用，查看其 **APPID**。

应用密钥

API 订阅方客户端访问网关时必须通过身份验证。身份验证信息由 Access Key / Secret Key 组成，二者成对出现。

操作步骤：

1. 进入 API 网关控制台 > **API 订阅** > **应用管理** 页面。
2. 在列表中找到要连接的应用名称，单击进入其详情页面。
3. 切换至 **密钥配置** 标签，即可查看并复制 Access Key 及 Secret Key 信息。



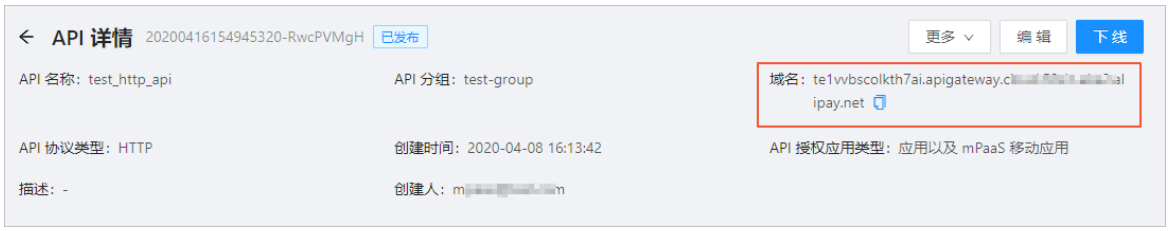
API 信息

API 域名地址

API 在网关上的访问地址，即 API 域名，由系统在创建 API 时自动生成。同一 API 分组下的所有 API 共用一个域名。域名格式为 `{id}.antfinapigw.{env}.alipay.com` 。其中，`{id}` 为 API 分组 ID，`{env}` 为环境信息。

操作步骤：

1. 进入 API 网关控制台 > **API 发布** > **API 管理** 页面。
2. 在 API 列表中，找到目标 API，单击进入其详情页。
3. 在 **基本信息** 栏，即可查看并复制 **域名地址** 信息。



API 系统集群密钥

API 发布者客户端连接网关时必须通过身份验证。身份验证信息由 Access Key / Secret Key 组成，二者成对出现。

操作步骤：

1. 进入 API 网关控制台 > **API 发布** > **系统集群** 页面。
2. 在列表中，找到并单击要查看的系统集群，进入系统集群详情页。

3. 切换至 **认证配置** 标签页，即可查看并复制系统集群身份密钥 AK/SK。

API 请求路径/接口名称/方法

应用发送请求至 API 的请求路径以进行 API 调用。

操作步骤：

1. 进入 API 网关控制台 > **API 订阅** > **应用管理** 页面。
2. 在列表中，找到目标应用，单击其名称，进入应用详情页。
3. 在 **API 订阅列表** 中，找到目标 API，即可获取到该 API 请求路径/接口名称以及方法等。

4.API 提供者接入

4.1. 概述

HTTP 类型的 API

如果服务提供方式 HTTP API，在不需要网关和服务端之间双向认证的情况下，不需要接入 API 网关的 SDK，直接编写 HTTP 接口即可。

如果需要网关和服务端之间双向认证的能力，则需要接入 API 网关的 SDK。目前仅提供 JAVA SDK。详见 系统集群验证签名。

SOFARPC 类型的 API

如果是 HTTP 转 SOFARPC 类型的 API，当前不支持网关和服务端之间双向认证，直接使用原生的编码方式发布 API 即可。

如果是 SOFARPC 转 SOFARPC 类型的 API，强制开启校验，直接使用官方 SOFARPC SDK 即可。当前版本的 API 网关暂不支该能力。

SOFAREST类型的 API

如果服务提供方式 SOFAREST API，在不需要网关和服务端之间双向认证的情况下，不需要接入 API 网关的 SDK，直接编写 SOFAREST 接口即可。

如果需要网关和服务端之间双向认证的能力，则需要接入 API 网关的 SDK。目前仅提供 JAVA SDK。详见 系统集群验证签名。

DUBBO 类型的 API

如果是 HTTP 转 SOFARPC 类型的 API，当前不支持网关和服务端之间双向认证，直接使用原生的编码方式发布 API 即可。

如果是 SOFARPC 转 SOFARPC 类型的 API，强制开启校验，直接使用官方 SOFARPC SDK 即可。当前版本的 API 网关暂不支该能力。

4.2. API 提供者 Demo

网关支持用户 下载服务端 Demo 进行测试使用，Demo 的代码分支为：server_demo。

网关依赖如下：

```
<dependency>
<groupId>com.alipay.sofa</groupId>
<artifactId>gateway-sdk</artifactId>
<version>2.5.0</version>
</dependency>
```

完成 Demo 的下载后，您可以进行以下多种类型的 API 服务开发与发布：

- HTTP API 服务
- SOFARPC API 服务
- SOFAREST 服务
- DUBBO API 服务

4.3. 外部授权

API 网关支持外部授权功能。通过该功能，用户可以在网关将请求转发给 real-server 之前，插入一个自定义的远程接口。网关会先将请求转发给这个自定义接口，该接口继而选择是否允许这个请求正常转发到 real-server。

外部授权服务编写须知

您需要在本地开发一个外部授权接口。当 API 需要验证授权关系时，会调用该外部授权接口进行授权校验。外部授权接口不限制协议类型，但会限制请求体和响应体。请求和响应均为 JSON 字符串，格式如下：

- **请求体**

```
{
"context":{
"key":"value"
}
}
```

字段说明：Request 中只有一个 context 字段，格式为 kv。这些参数来自于 client 的 request，需要在 API 网关控制台平台进行配置。

- **响应体**

```
{
"success":true/false,
"principal":{
"key":"value"
},
"failResponseHeader":{
"key":"value"
},
"failResponseBody": jsonarray/jsonobject
"failResponseStatus": ${httpcode}
}
```

字段说明：

  - `success`：是否允许该请求转发到 real-server。
  - `principal`：如果允许转发到 real-server，可以将一些信息传递给 real-server，kv 格式。
    - 如果 real-server 是 HTTP 接口，则 principal 会放到 header 中。
    - 如果 real-server 是 SOFARPC 接口，则 principal 会放到 baggage 中。
  - `failResponseHeader`：如果不允许转发到 real-server，可以设置响应头返回给 client。
  - `failResponseBody`：如果不允许转发到 real-server，可以设置响应 body 给 client。
  - `failResponseStatus`：如果不允许转发到 real-server，可以设置响应 HTTP 状态码。

对应外部授权 API 的定义标准如下：

- **AuthRequest**

```
publicclassAuthRequest{
private Map<String,String> context;
}
```

- AuthResponse

```
publicclassAuthResponse{
private boolean             success;
private Map<String,String>  principal;
private Map<String,String> failResponseHeader;
private Object              failResponseBody;
private int                 failResponseStatus;
}
```

> **说明**
>
> AuthRequest 和 AuthResponse 需要按照本文提供的数据结构不能改动，类名可以自定义修改。

```
public class AuthRequest {
    private Map<String, String> context;

    public Map<String, String> getContext() {
        return context;
    }

    public void setContext(Map<String, String> context) {
        this.context = context;
    }
}
@Data
public class AuthResponse {

    private boolean success;
    private Map<String, String> principal;
    private Map<String, String> failResponseHeader;
    private int failResponseStatus;
    private Object failResponseBody;

    public void setFailResponse(Object failResponse) {
        this.failResponseBody = failResponse;
    }

}
@RestController
@RequestMapping("/api")
public class AuthController {

    @RequestMapping("/auth")
    public AuthResponse auth(@RequestBody AuthRequest request) {
        return AuthUtil.buildAuthRes(request);
    }

}
```

```
public class AuthUtil {

    private static final Logger LOGGER = LoggerFactory.getLogger(AuthUtil.class);

    public static AuthResponse buildAuthRes(AuthRequest request) {
        AuthResponse response = new ObjectAuthResponse();
        //---以下都是在外部授权配置页面中配置的参数key才能在这边取得到值----
        String headerKey = request.getContext().get("headerKey");
        String path = request.getContext().get("x-mosn-path");
        String method = request.getContext().get("x-mosn-method");
        String bodyKey = request.getContext().get("bodyKey");
        String queryKey = request.getContext().get("queryKey");
        LOGGER.info("headerKey:" + headerKey + ",bodyKey:" + bodyKey + ",queryKey:" + query
Key + ",path:" + path + ",method:" + method);
        //------ end ------
        if ("h".equalsIgnoreCase(headerKey)) {
            // success
            response.setSuccess(true);//外部授权通过
            Map<String, String> principal = new HashMap<>();
            principal.put("param-to-server", "abc");
            principal.put("userName", "user name is tom aaa");
            response.setPrincipal(principal);//需要透传到后端服务的参数
            return response;
        } else {
            // error
            response.setSuccess(false);//外部授权失败
            Map<String, String> failResponseHeader = new HashMap<>();
            queryKey = request.getContext().get("queryKey");
            if ("q".equalsIgnoreCase(queryKey)) {
                failResponseHeader.put("header-to-client", "query");
            } else {
                failResponseHeader.put("header-to-client", "no-query");
            }
            response.setFailResponseHeader(failResponseHeader);//失败的响应头
            Map<String, String> failResponseBody = new HashMap<>();
            String context = JSON.toJSONString(request.getContext());
            failResponseBody.put("test", context);
            response.setFailResponse(failResponseBody);//失败的错误信息
            response.setFailResponseStatus(401);//失败响应码
            LOGGER.info("buildAuthRes|response|{}|{}", (response instanceof ByteAuthRespons
e), JSON.toJSONString(response));
            return response;
        }
    }
}
```

4.4. API 缓存配置

API 网关支持对某些请求的结果进行缓存。如果一个接口本身是幂等的，并且返回值在一定时间内不会发生变化，那么可以在 API 网关上配置缓存。

- 当前支持的缓存最长时间为 5 分钟。

- 缓存的 key 可以从 Header、Cookie、QueryString 或 Body 中选择。
- 如果选择 Body，Body 必须是一个 JSON 字符串，暂时不支持 pb 格式的 Body。

> **说明**
>
> 目前 API 缓存只支持 downstream 为 HTTP 的请求，暂时不支持 SOFARPC 的请求。

配置说明

假设 API 的配置如下：

```
- method: GET
- path: /test/cache
- cacheConfig: header:["version","temp"], queryString:["a","b"]; ttl 10s
```

此处的 `cacheConfig` 含义是：

- 如果 Header 中存在 `version` 和 `temp`，并且 QueryString 中存在 `a` 和 `b`，那么这个请求的响应将会被缓存。
- 如果第二个请求中 Header 和 QueryString 的 value 和第一个请求一样，那么网关会直接将第一个请求的缓存返回，而不会再去调用服务提供方。

例如，第一个请求如下：

```
curl -H "version=1"-H "temp=true"127.0.0.1?a=a&b=b
```

该请求包含了 a 和 b 两个 QueryString key，也包含 version 和 temp 两个 Header key，所以这个请求的响应会被缓存 10s。

假设 10s 内的第二个请求如下：

```
curl -H "version=1"-H "temp=true"-H "other=other"127.0.0.1?a=a&b=b&c=c
```

该请求也包含了相应的 key，同时值也和第一个请求相同。这个请求网关会直接返回缓存值，而不会调用到后端 server。

设置 Key

当前，API 网关支持设置 Header、Body、Query、Cookie key。

- Header、Query、Cookie 都比较简单，直接设置 key 就可以的。
- 但是 Body 一般都比较复杂，无法直接通过一个简单的字符串定位到，所以设置的方式比较特殊。当前仅支持 Body 为 JSON 的请求。

Body key 设置示例

假设 Body 是如下字符串：

```
{
"name":{"first":"Tom","last":"Anderson"},
"age":37,
"children":["Sara","Alex","Jack"],
"fav.movie":"Deer Hunter",
"friends":[
{"first":"Dale","last":"Murphy","age":44,"nets":["ig","fb","tw"]},
{"first":"Roger","last":"Craig","age":68,"nets":["fb","tw"]},
{"first":"Jane","last":"Murphy","age":47,"nets":["ig","tw"]}
]
}
```

设置 Body key 的时候，需要使用一个简单的表达式：

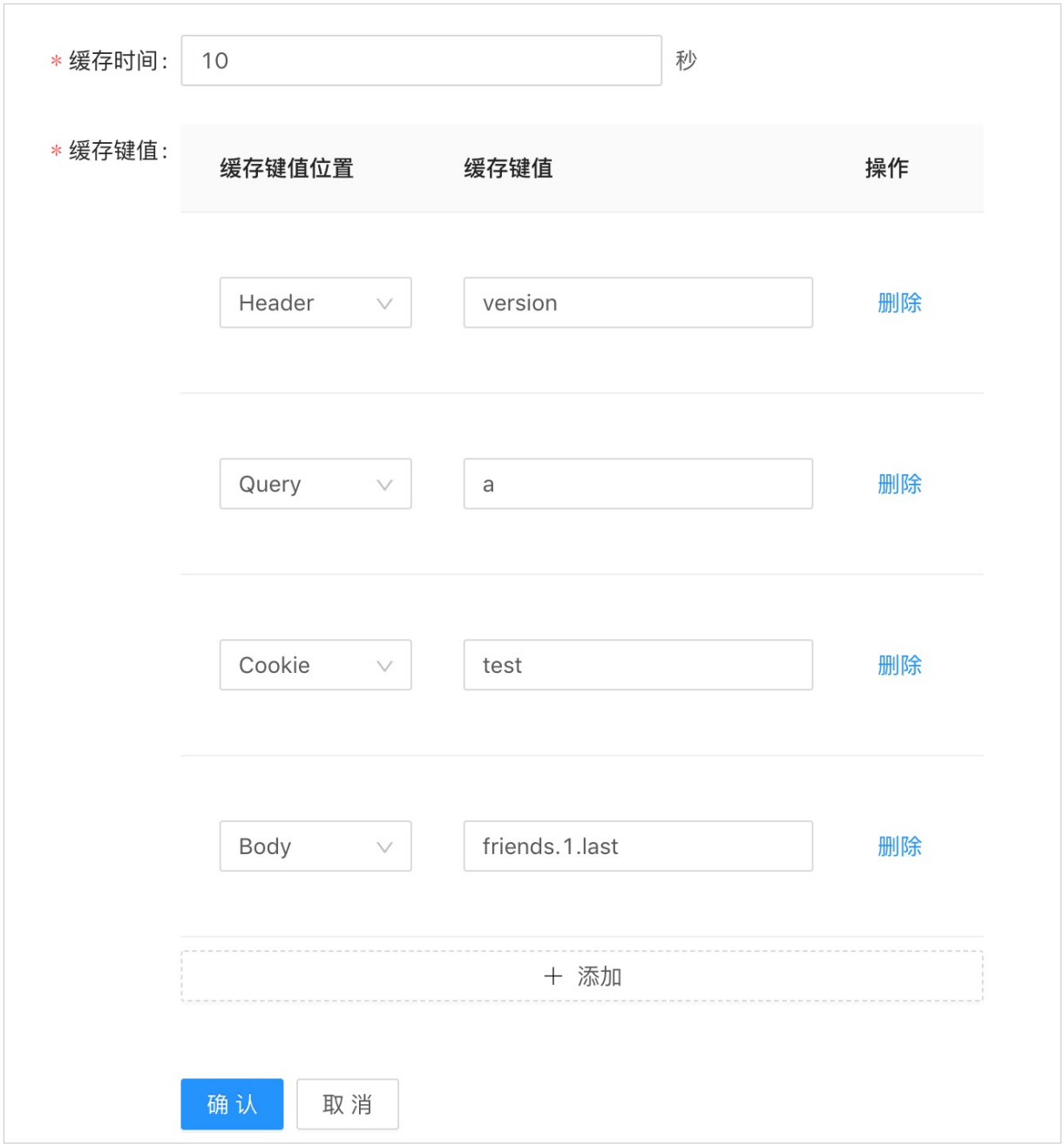
```
"name.last">>"Anderson"
"age">>37
"children">>["Sara","Alex","Jack"]
"children.#">>3
"children.1">>"Alex"
"child*.2">>"Jack"
"c?ildren.0">>"Sara"
"fav\.movie">>"Deer Hunter"
"friends.#.first">>["Dale","Roger","Jane"]
"friends.1.last">>"Craig"
```

您还可以使用 `#(...)` 查询第一个匹配项的数组，或使用 `#(...)#` 查找所有匹配项。支持 `==` 、 `!=` 、 `<` 、 `<=` 、 `>` 、 `>=` 和简单的模式匹配 `%` （like）和 `!%` （not like）。

```
friends.#(last=="Murphy").first    >>"Dale"
friends.#(last=="Murphy")#.first   >>["Dale","Jane"]
friends.#(age>45)#.last>>["Craig","Murphy"]
friends.#(first%"D*").last>>"Murphy"
friends.#(first!%"D*").last>>"Craig"
friends.#(nets.#(=="fb"))#.first   >>["Dale","Roger"]
```

控制台配置

您可以直接在 API 网关控制台**> API 详情页 > 流量治理 > 缓存** 进行相应的 API 缓存配置，如下图所示。



> **说明**
>
> 当前 Body 的嵌套最多支持 5 层，比如 a.b.c.d.e.f。超过 5 层暂不支持。

4.5. 路径匹配规则

API 网关支持 **绝对匹配**、**参数匹配** 和 **前缀匹配** 三种路径匹配规则。

绝对匹配

绝对匹配，表示只有请求的路径和配置的路径完全一样时才能匹配成功。

示例

例如，当前 API 的匹配规则为 `GET /path` 。

请求示例：

- 请求一：

```
curl http://127.0.0.1/path
```

- 请求二：

```
curl http://127.0.0.1/other
```

根据 `GET /path` 的匹配规则，以上请求中，仅请求一能匹配到当前 API。

参数匹配

当无法确定请求的具体路径，仅明确知道一定存在一个值时，可以在路径中配置一个参数。

示例

例如，当前 API 的匹配规则为 `GET /path/{id}` 。

请求示例：

- 请求一：

```
curl http://127.0.0.1/path
```

- 请求二：

```
curl http://127.0.0.1/path/1
```

- 请求三：

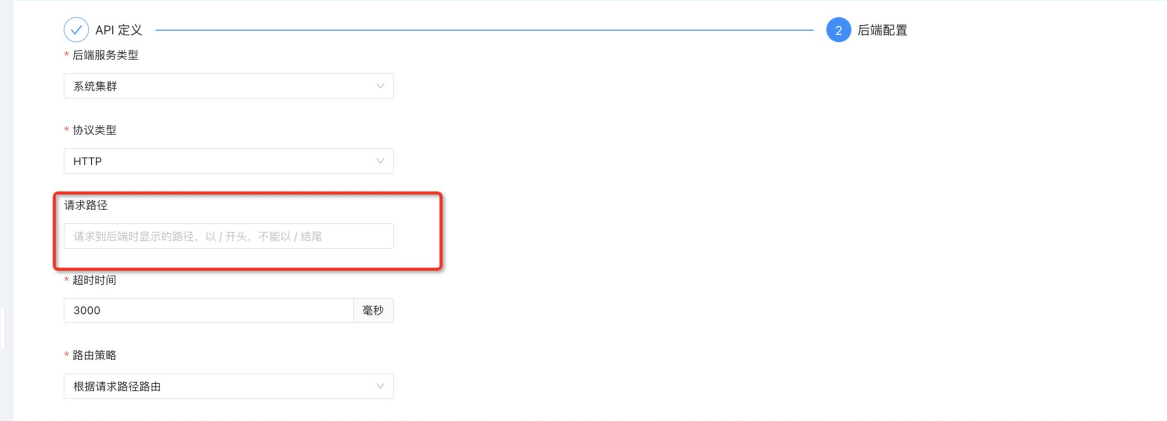
```
curl http://127.0.0.1/path/2
```

- 请求四：

```
curl http://127.0.0.1/path/1/2
```

根据 `GET /path/{id}` 的匹配规则，以上请求中，仅请求二、三能匹配到当前 API。

> **说明**
>
> 如果需要参数匹配中的参数透传到服务后端，那么在网关控制台配置 API 时，后端服务的请求路径必须为空（下图红框处），若填写了后端服务请求路径，后端服务会以绝对匹配的方式去请求此路径。
>
> 例如：后端请求路径填写了 `/user/{name}`，发起请求后，`{name}` 占位符不会生效，后端仅能请求到 `path=/user/{name}` 的接口。



前缀匹配

如果希望带有某个前缀的请求全部匹配到某一个 API，则可以使用前缀匹配。

示例

例如，当前 API 的匹配规则为 `GET /path/*`。

请求示例：

- 请求一：

```
curl http://127.0.0.1/path/1
```

- 请求二：

```
curl http://127.0.0.1/path/2
```

- 请求三：

```
curl http://127.0.0.1/path/1/2
```

根据 `GET /path/*` 的匹配规则，以上所有请求都可以匹配到该 API。

> **说明**
>
> 如果设置前缀匹配的路径为 `/path`，则只能请求到路径为 `/path/xxx` 的接口，无法请求到路径为 `/path` 的接口。

冲突解决

API 网关支持三种匹配方式。不可避免的，多个 API 之间会出现冲突。比如 `/hello` 和 `/{id}` 这两个 API 当遇到路径为 `/hello` 的请求时，就会冲突。

针对这种冲突情况，API 网关会遵循 **最长路径优先** 规则。当路径长度相同时，继而遵循 **绝对匹配 > 参数匹配 > 前缀匹配** 的规则。

示例

本文提供以下几个 API。

以下配置，优先级由低到高，*为前缀匹配

```
GET /*
GET /{id}
GET /path
GET /{id}/*
GET /{id}/{name}
GET /{id}/temp
GET /path/*
GET /path/{id}
GET /path/temp
GET /{id}/{name}/{path}
```

配置路径匹配规则

在 API 网关控制台创建 API 时，您可以配置 **路径匹配规则**，支持 **绝对匹配** 或 **前缀匹配**。两种匹配方式都可以在路径中添加参数。

4.6. 系统集群验证签名

本文介绍如何进行系统集群验证签名。

> 说明
>
> 网关提供服务端 Demo 代码供用户下载测试，下载 API 提供者 Demo 后，Demo 接口位置为：com.alipay.gateway.common.GatewayFilter。

操作步骤如下：

1. 引入网关 SDK
2. 配置 API 服务密钥
3. 配置 Filter

引入网关 SDK

API 网关的客户端依赖 `gateway-sdk` 。对服务端而言主要是配置 spring 拦截器，对请求进行验签以及对响应进行加签。您需要在本地工程 `pom.xml` 中添加如下 SDK。

> 说明
>
> - 如网关和服务端之间不需要双向认证，则无需引入 `gateway-sdk` ，直接编写 HTTP API 服务即可。
> - 网关 SDK 新增了数据加密的功能，请参见 数据加密。

```
<dependency>
<groupId>com.alipay.sofa</groupId>
<artifactId>gateway-sdk</artifactId>
<version>2.5.0</version>
</dependency>
```

配置密钥

如果 API 服务发布的系统集群配置了 **密钥** 的认证方式，您还需要在工程中配置相应的 Access Key 和 Secret Key。您可以在 API 网关控制台系统集群详情页获取。具体获取方法，参见 获取服务端信息。

```
# 服务发布方的ak、sk，也就是系统集群的密钥信息
pub.app.accessKey=<yourAccessKeyId>
pub.app.secretKey=<yourAccessKeySecret>
```

配置 Filter

您需要为服务添加验签 filter。HTTP API 服务支持 REST 及 Spring MVC 框架。

Spring MVC 服务

```
    @Bean
    public FilterRegistrationBean filterRegistrationBean(@Value("${pub.app.accessKey}") Str
ing keyId,
                                                         @Value("${pub.app.secretKey}") Str
ing secretKey) {
        FilterRegistrationBean registration = new FilterRegistrationBean();
        registration.setFilter(apiServletSignFilter(keyId, secretKey));
        registration.addUrlPatterns("/sign"); // filter生效的uri路径，表示对所有路径生效
        registration.setName("apiServletSignFilter");
        registration.setOrder(Ordered.HIGHEST_PRECEDENCE);
        return registration;
    }

    @Bean
    public ApiServletSignFilter apiServletSignFilter(@Value("${pub.app.accessKey}") String
keyId,
                                                     @Value("${pub.app.secretKey}") String
secretKey
    ) {
        ApiSecretKey apiSecretKey = new ApiSecretKey(keyId, secretKey);
        ApiServletSignFilter filter = new ApiServletSignFilter();
        List<ApiSecretKey> secretKeys = new ArrayList<>();
        secretKeys.add(apiSecretKey);
        filter.setApiSecretKeys(secretKeys);
        filter.setCheckSign(true);
        filter.setSignUriRegex("/sign"); // 开启验签加签的uri路径，表示对 /j 开头的路径做加签验签
，其他路径则不验签
        return filter;
    }
```

REST 服务

> 说明
>
> 当系统集群需要密钥认证时，需要开启响应验签，并在方法中加上 `@NeedSign` 。

```
    // 给 rest 服务添加filter，请求验签
    @Bean
    public ApiRestSignFilter apiRestSignFilter(@Value("${pub.app.accessKey}") String keyId,
                                               @Value("${pub.app.secretKey}") String secret
Key) {
        ApiSecretKey apiSecretKey = new ApiSecretKey(keyId, secretKey);
         List<ApiSecretKey> secretKeys = new ArrayList<>();
         secretKeys.add(apiSecretKey);
         ApiRestSignFilter filter = new ApiRestSignFilter();
         filter.setApiSecretKeys(secretKeys);
         filter.setCheckSign(false);//请求验签开关
         //
         JAXRSProviderManager.registerCustomProviderInstance(filter);

         return filter;
    }
```

4.7. HTTP API 服务

网关提供服务端 HTTP Demo 代码供用户下载测试，下载 API 提供者 Demo 后，HTTP Demo 代码位置为：com/alipay/gateway/controller/http/MvcServerController.java。

> 注意
>
> - POST 请求的 Body 参数必须使用 `@RequestBody` 接收参数。
> - Query 参数可以使用 `@RequestParam()` 接收指定参数，如果不使用 RequestParam，参数默认和请求参数名一致。

Demo 案例

```
    /**
     * 测试场景： 没有参数的测试
     *
     * @return
     */
    @RequestMapping(value = "/param/noParam", method = {RequestMethod.GET, RequestMethod.PO
ST, RequestMethod.PUT, RequestMethod.DELETE})
    @ResponseBody
    @Override
    public Map<String, String> hello() {
        return super.hello();
    }

    /**
     * 测试场景：简单参数
     *
     * @param param
     * @return
     */
```

```
    @RequestMapping(value = "/param/simpleParam", method = {RequestMethod.GET, RequestMetho
d.DELETE})
    @ResponseBody
    @Override
    public String simpleParam(@RequestParam("param") String param) {
        return super.simpleParam(param);
    }

    /**
     * 测试场景：简单参数
     *
     * @param param
     * @return
     */
    @RequestMapping(value = "/param/simpleParam", method = {RequestMethod.POST, RequestMeth
od.PUT})
    @ResponseBody
    public String simpleBodyParam(@RequestBody String param) {
        return super.simpleParam(param);
    }

    /**
     * 测试场景：简单参数
     *
     * @param param
     * @return
     */
    @RequestMapping(value = "/param/simpleParamForMap", method = {RequestMethod.GET, Reques
tMethod.DELETE})
    @ResponseBody
    @Override
    public Map<String, String> simpleParamForMap(@RequestParam("param") String param) {
        return super.simpleParamForMap(param);
    }

    /**
     * 测试场景：简单参数
     *
     * @param param
     * @return
     */
    @RequestMapping(value = "/param/simpleParamForMap", method = {RequestMethod.POST, Reque
stMethod.PUT})
    @ResponseBody
    public Map<String, String> simpleBodyParamForMap(@RequestBody String param) {
        return super.simpleParamForMap(param);
    }



    /**
     * 测试场景：简单参数
     *
     * @param param
```

```
     * @return
     */
    @RequestMapping(value = "/param/intParam", method = {RequestMethod.GET, RequestMethod.P
OST, RequestMethod.PUT, RequestMethod.DELETE})
    @ResponseBody
    @Override
    public Integer intParam(Integer param) {
        return super.intParam(param);
    }

    /**
     * 测试场景：多参数
     *
     * @return
     */
    @RequestMapping(value = "/param/multiParam", method = {RequestMethod.GET, RequestMethod
.POST, RequestMethod.PUT, RequestMethod.DELETE})
    @ResponseBody
    @Override
    public Map<String, String> multiParam(@RequestParam("param1") String param1,
                                          @RequestParam("param2") String param2) {
        return super.multiParam(param1, param2);
    }

    /**
     * 测试场景：多参数
     *
     * @return
     */
    @RequestMapping(value = "/param/multiParamList", method = {RequestMethod.GET, RequestMe
thod.POST, RequestMethod.PUT, RequestMethod.DELETE})
    @ResponseBody
    @Override
    public Map<String, String> multiParamList(@RequestBody List<String> params) {
        return super.multiParamList(params);
    }

    /**
     * 测试场景：路径参数
     *
     * @param uid
     * @return
     */
    @RequestMapping(value = "/param/{uid}", method = {RequestMethod.GET, RequestMethod.POST
, RequestMethod.PUT, RequestMethod.DELETE})
    @ResponseBody
    @Override
    public Map<String, String> pathParam(@PathVariable("uid") String uid) {
        return super.pathParam(uid);
    }

    /**
     * 测试场景：json参数
     *
```

```
     * @param user
     * @return
     */
    @RequestMapping(value = "/param/jsonParam", method = {RequestMethod.POST, RequestMethod
.PUT})
    @ResponseBody
    public User jsonBodyParam(@RequestBody User user) {
        return super.jsonParam(user);
    }


    /**
     * 场景：获取cookie|header|query 内容
     *
     * @param request
     * @return
     */
    @RequestMapping(value = "/param/withHeader", method = {RequestMethod.POST, RequestMetho
d.PUT})
    @ResponseBody
    public Map<String, String> withHeaderBodyParam(HttpServletRequest request, @RequestBody
String name) {
        return super.withHeader(request, name);
    }

    /**
     * 场景：获取header 多参数共传
     *
     * @return
     */
    @RequestMapping(value = "/param/moreParam", method = {RequestMethod.GET, RequestMethod.
POST, RequestMethod.PUT, RequestMethod.DELETE})
    @ResponseBody
    @Override
    public Map<String, Object> moreParam(@CookieParam("param") String param, @HeaderParam("
userName") String userName, @QueryParam("age") String age) {
        return super.moreParam(param, userName, age);
    }

    /**
     * 场景：模拟超时
     *
     * @param param
     */
    @RequestMapping(value = "/timeout", method = {RequestMethod.GET, RequestMethod.DELETE})
    @ResponseBody
    @Override
    public Map<String, String> timeout(@RequestParam("param") String param) {
        return super.timeout(param);
    }


    /**
     * 测试场景：测试缓存
```

```
     *
     * @param param
     * @return
     */
    @RequestMapping(value = "/param/cache", method = {RequestMethod.GET, RequestMethod.DELE
TE})
    @ResponseBody
    @Override
    public Map<String, String> cache(@RequestParam("param") String param) {
        return super.cache(param);
    }


    /**
     * 测试场景：外部授权body传参
     *
     * @param param
     * @return
     */
    @RequestMapping(value = "/param/authBodyParam", method = {RequestMethod.GET, RequestMet
hod.POST, RequestMethod.PUT, RequestMethod.DELETE})
    @ResponseBody
    @Override
    public Map<String, String> authBodyParam(String param, HttpServletRequest request) {
        return super.authBodyParam(param, request);
    }

    /**
     * 测试场景：请求参数映射
     *
     * @param
     * @return
     */
    @RequestMapping(value = "/param/requestParamMapping", method = {RequestMethod.GET, Requ
estMethod.POST, RequestMethod.PUT, RequestMethod.DELETE})
    @ResponseBody
    @Override
    public List<JSONObject> requestParamMapping(HttpServletRequest request, @RequestBody Li
st<JSONObject> jsonObject) {
        return super.requestParamMapping(request, jsonObject);
    }

    /**
     * 测试场景：响应参数映射
     *
     * @param
     * @return
     */
    @RequestMapping(value = "/param/responseParamMapping", method = {RequestMethod.GET, Req
uestMethod.POST, RequestMethod.PUT, RequestMethod.DELETE})
    @ResponseBody
    @Override
    public JSONObject responseParamMapping(HttpServletRequest request, @RequestBody List<JS
ONObject> jsonObject) {
```

```
        return super.responseParamMapping(request, jsonObject);
    }

    /**
     * 测试场景：LDC路由
     *
     * @param
     * @return
     */
    @RequestMapping(value = "/param/ldc", method = {RequestMethod.GET, RequestMethod.POST,
RequestMethod.PUT, RequestMethod.DELETE})
    @ResponseBody
    @Override
    public Map<String, Object> ldc(HttpServletResponse response, HttpServletRequest request
) {
        return super.ldc(response, request);
    }

    /**
     * 测试场景：返回 Object 有参数
     *
     * @param
     * @return
     */
    @RequestMapping(value = "/param/objectSimpleParam", method = {RequestMethod.GET, Reques
tMethod.POST, RequestMethod.PUT, RequestMethod.DELETE})
    @ResponseBody
    @Override
    public Object objectSimpleParam(@RequestBody User user) {
        return super.objectSimpleParam(user);
    }

    /**
     * 测试场景：返回 Map 有参数
     *
     * @param
     * @return
     */
    @RequestMapping(value = "/param/mapSimpleParam", method = {RequestMethod.GET, RequestMe
thod.POST, RequestMethod.PUT, RequestMethod.DELETE})
    @ResponseBody
    @Override
    public Map<String, User> mapSimpleParam(@RequestBody Map<String, User> map) {
        return super.mapSimpleParam(map);
    }

    /**
     * 测试场景：返回 List 有参数
     *
     * @param
     * @return
     */
    @RequestMapping(value = "/param/listSimpleParam", method = {RequestMethod.GET, RequestM
ethod.POST, RequestMethod.PUT, RequestMethod.DELETE})
    @ResponseBody
```

```
    @Override
    public List<User> listSimpleParam(@RequestBody List<User> list) {
        return super.listSimpleParam(list);
    }

    /**
     * 测试场景：返回StringOutput StringInput参数
     *
     * @param
     * @return
     */
    @RequestMapping(value = "/param/stringInputParam", method = {RequestMethod.GET, Request
Method.POST, RequestMethod.PUT, RequestMethod.DELETE})
    @ResponseBody
    @Override
    public StringOutput inputParam(@RequestBody StringInput input) {
        return super.inputParam(input);
    }

}
```

4.8. SOFARPC API 服务

网关提供服务端 SOFARPC Demo 代码供用户下载测试，下载 API 提供者 Demo 后，SOFARPC Demo 代码位置为：com/alipay/gateway/sofarpc/GatewayRpcServiceImpl.java。

RPC 接入配置说明（必填）

```
#服务实例标识，instanceId
com.alipay.instanceid=J79FOTMP8886
#acvip地址用作订阅注册中心
com.antcloud.antvip.endpoint=100.88.93.7
#操作员ak/sk
com.antcloud.mw.access=AC8imi57g6TiRr9u
com.antcloud.mw.secret=bL6QgJzDh1vuLFWftsWmoEBi1qPrzwQz

#rpc服务端口
com.alipay.sofa.rpc.bolt-port=12200
```

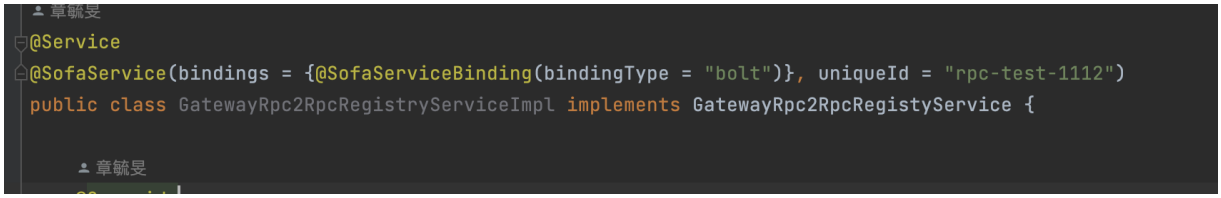
发布服务

> **说明**
>
> 服务可以通过注解的形式或者通过 XML 配置文件进行发布。

XML 配置形式

```
<bean id="gatewayRpcService" class="com.alipay.gateway.sofarpc.GatewayRpcServiceImpl"/>
    <sofa:service ref="gatewayRpcService" interface="com.alipay.gateway.facade.rpc.GatewayR
pcService"
                  unique-id="rpc-test-1112">
        <sofa:binding.bolt/>
    </sofa:service>
```

注解形式



Demo 案例

```
@Service
@SofaService(bindings = {@SofaServiceBinding(bindingType = "bolt")}, uniqueId = "rpc-test-1
112")
public class GatewayRpcServiceImpl implements GatewayRpcService {

    @Override
    public String hello() {
        return "hello noParamForString";
    }

    @Override
    public void testVoid(String param) {

    }

    @Override
    public void testException() throws Exception {

    }

    @Override
    public Map<String, String> noParamForMap() {
        System.out.println("rpc-test");
        Map<String, String> map = Maps.newHashMap();
        map.put("result", "noParamForMap");
        return map;
    }

    @Override
    public String simpleParam(String param) {
        return "hello simpleParamForString " + param;
    }

    @Override
    public Map<String, String> simpleParamForMap(String param) {
        Map<String, String> map = Maps.newHashMap();
```

```
        map.put("result", "simpleParamForMap " + param);
        return map;
    }

    @Override
    public String jsonParam(JSONObject param) {
        return "hello jsonParam";
    }

    @Override
    public Map<String, String> jsonParamForMap(JSONObject param) {
        Map<String, String> map = Maps.newHashMap();
        map.put("result", "jsonParamForMap");
        return map;
    }

    @Override
    public User jsonParamForUser(User user) {
        return user;
    }

    @Override
    public List<String> listNoParam() {
        List<String> list = new ArrayList<>();
        list.add("aaa");
        return list;
    }

    @Override
    public List<User> listSimpleParam(List<User> list) {
        return list;
    }

    @Override
    public String registryQuestionTest() throws UnknownHostException {
        return JSONObject.toJSONString(InetAddress.getLocalHost());
    }

    @Override
    public User user1(User user) {
        return user;
    }

    @Override
    public User user2(String name) {
        User user = new User(name, 20);
        return user;
    }

    @Override
    public AccountQueryResult queryAccountBalance(String accountNo) {

        AccountQueryResult accountQueryResult = new AccountQueryResult();
        Account account = new Account(accountNo, new BigDecimal(11111111), new BigDecimal(2
```

```
2222222), new BigDecimal(33333333));
        accountQueryResult.setSuccess(true);
        accountQueryResult.setAccount(account);
        accountQueryResult.setMsgCode("200");
        accountQueryResult.setMsgText("success");

        return accountQueryResult;
    }
```

4.9. SOFAREST 服务

网关提供服务端 SOFAREST Demo 代码供用户下载测试，下载 API 提供者 Demo 后，代码位置为：com/alipay/gateway/sofarest。

SOFAREST接入配置说明（必填）

```
#服务实例标识，instanceId
com.alipay.instanceid=J79FOTMP8886
#acvip地址用作订阅注册中心
com.antcloud.antvip.endpoint=100.88.93.7
#操作员ak/sk
com.antcloud.mw.access=AC8imi57g6TiRr9u
com.antcloud.mw.secret=bL6QgJzDh1vuLFWftsWmoEBi1qPrzwQz

#rpc服务端口
com.alipay.sofa.rpc.bolt-port=12200
```

发布服务

> 说明
>
> 服务可以通过通过 XML 配置文件进行发布。

XML 配置形式

```
<bean id="sofaRestPostService" class="com.alipay.gateway.sofarest.nosign.SofaRestPostServic
eImpl"/>
    <sofa:service ref="sofaRestPostService" interface="com.alipay.gateway.facade.rest.nosig
n.SofaRestPostService"
                  unique-id="rpc-test-1112">
        <sofa:binding.rest/>
    </sofa:service>
```

Demo案例

```
@Path("rest")
public interface SofaRestPostService {
    @POST
    @Path("noParam")
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
```

```
    String noParam();

    @POST
    @Path("sampleParam")
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    String sampleParam(String param) throws InterruptedException;

    @POST
    @Path("param/{uid}")
    String uidParam(@PathParam("uid") String uid);

    @POST
    @Path("moreParam")
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    String moreParam(@QueryParam("param") String param, @HeaderParam("param1") String param
1, String param2);

    @POST
    @Path("headerParam")
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    Map<String, String> headerParam(HttpServletRequest request);

    @POST
    @Path("jsonParam")
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    Map<String, Object> jsonParam(@RequestBody User user);

    @POST
    @Path("formParam")
    @Produces(MediaType.APPLICATION_FORM_URLENCODED)
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    Map<String, Object> formParam(@FormParam("name") String name, @FormParam("age") String
age);

    @POST
    @Path("timeout")
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    String timeout(String sleepTime);

    @POST
    @Path("exception")
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    void exception() throws Exception;

    @POST
    @Path("auth")
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
```

```
    AuthResponse auth(AuthRequest authRequest);

    @POST
    @Path("hello")
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    Map<String, String> hello();

    @POST
    @Path("objectNoParam")
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    User objectNoParam();

    @POST
    @Path("objectSimpleParam")
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    User objectSimpleParam(User user);

    @POST
    @Path("mapNoParam")
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    Map<String, String> mapNoParam();

    @POST
    @Path("mapSimpleParam")
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    Map<String, User> mapSimpleParam(Map<String, User> map);

    @POST
    @Path("listNoParam")
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    List<String> listNoParam();

    @POST
    @Path("listSimpleParam")
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    List<User> listSimpleParam(List<User> list);

    @POST
    @Path("getApiParam2")
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    String getApiParam2(String param);

    @POST
    @Path("getApiParam3")
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
```

```
    String getApiParam3(String param);

    @POST
    @Path("getApiParam3Sign")
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    String getApiParam3Sign(@HeaderParam("param") String param);

    @POST
    @Path("weightRouter")
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    String router();
}
```

4.10. DUBBO API 服务

网关提供服务端 DUBBO Demo 代码供用户下载测试，下载 API 提供者 Demo 后，DUBBO Demo 接口位置为：com.alipay.gateway.dubbo.DubboServiceImpl。

> **说明**
>
> DUBBO 和 DUBBOX 的使用方法一致，只需根据协议引入正确的 DUBBO 依赖即可。

DUBBO 服务 Demo

```
public class DubboServiceImpl implements DubboService {

    @Override
    public Map<String, String> booleanParam(Boolean booleanParam) {
        Map map = new HashMap();
        map.put("boolean", "hello " + booleanParam);
        return map;
    }

    @Override
    public Map<String, String> byteParam(Byte byteParam) {
        Map map = new HashMap();
        map.put("byte", "hello " + byteParam);
        return map;
    }

    @Override
    public Map<String, String> charParam(Character charParam) {
        Map map = new HashMap();
        map.put("char", "hello " + charParam);
        return map;
    }

    @Override
    public Map<String, String> shortParam(Short shortParam) {
        Map map = new HashMap();
```

```
        map.put("short", "hello " + shortParam);
        return map;
    }

    @Override
    public Map<String, String> intParam(Integer intParam) {
        Map map = new HashMap();
        map.put("int", "hello " + intParam);
        return map;
    }

    @Override
    public Map<String, String> longParam(Long longParam) {
        Map map = new HashMap();
        map.put("long", "hello " + longParam);
        return map;
    }

    @Override
    public Map<String, String> floatParam(Float floatParam) {
        Map map = new HashMap();
        map.put("float", "hello " + floatParam);
        return map;
    }

    @Override
    public Map<String, String> doubleParam(Double doubleParam) {
        Map map = new HashMap();
        map.put("double", "hello " + doubleParam);
        return map;
    }

    @Override
    public Map<String, String> noParam() {
        Map map = new HashMap();
        map.put("zym", "hello");
        return map;
    }

    @Override
    public Person personParam(Person person) {
        return person;
    }

    @Override
    public Map<String, String> stringParam(String param) {
        Map map = new HashMap();
        map.put("string", "hello " + param);
        return map;
    }
}

public interface DubboService {
```

```
    public Map<String, String> booleanParam(Boolean booleanParam);

    public Map<String, String> byteParam(Byte byteParam);

    public Map<String, String> charParam(Character charParam);

    public Map<String, String> shortParam(Short shortParam);

    public Map<String, String> intParam(Integer intParam);

    public Map<String, String> longParam(Long longParam);

    public Map<String, String> floatParam(Float floatParam);

    public Map<String, String> doubleParam(Double doubleParam);

    public Map<String, String> noParam();

    public Person personParam(Person person);

    public Map<String, String> stringParam(String param);

}
```

发布服务

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://code.alibabatech.com/schema/dubbo
    http://code.alibabatech.com/schema/dubbo/dubbo.xsd">
    <dubbo:application name="${spring.application.name}"/>
    <!-- 声明需要暴露的服务接口（注意是接口,不是实现类）
     这里是具体实现类,id和上面的暴露的服务接口ref要一致,dubbo就是通过这个来注册对应的服务
     同模块的registry使用local，不同模块的使用远程的register -->
    <dubbo:registry id="register" address="zookeeper://${zookeeper.address}:2181"/>
    <dubbo:protocol serialization="fastjson" name="dubbo" port="20881" />

    <bean id="dubboService" class="com.alipay.gateway.dubbo.DubboServiceImpl"/>
    <dubbo:service version="2.0.0"  registry="register"
    interface="com.alipay.gateway.facade.dubbo.DubboService"
    ref="dubboService" protocol="dubbo" group="zym_group" />

</beans>
```

5.API 订阅者接入

5.1. API 订阅者 Demo

网关支持用户 下载客户端 Demo 进行测试使用，Demo 的代码分支为：client_demo。

网关依赖如下：

```
<dependency>
       <groupId>com.alipay.sofa</groupId>
       <artifactId>gateway-sdk</artifactId>
       <version>2.5.0</version>
       </dependency>
 <dependency>
       <groupId>cn.com.antcloud.api</groupId>
       <artifactId>antcloud-api-apigateway</artifactId>
       <version>1.1.210</version>
 </dependency>
```

5.2. 发送 HTTP 请求（Java）

本文介绍 API 订阅方应用如何使用 Java 语言发送 HTTP 请求接入网关。

> **说明**
>
> 网关提供客户端 Demo 代码供用户参考，下载 API 订阅者 Demo 后，Demo 代码位置为：src/test/java/com/alipay/gateway/web/test/usercase/mainchain/http/

前提条件

在进行本地应用开发前，您需要确保已经完成以下操作：

- 已在 API 网关控制台创建了一个应用。
- 已将该应用的 APPID 提供给了 API 发布者，并获得了授权。
- 已获取了如下服务配置信息。具体获取方法，参见 获取服务端信息。
  - 应用的访问密钥（Access Key/Secret Key）
  - API 的域名地址（host）、请求路径（path）、方法（method）、请求体（body）

操作步骤

API 网关提供了一个 Java SDK，即 `gateway-sdk` 。该 SDK 集成了加签、加密、验签的逻辑，同时默认支持序列化和反序列化。使用该 SDK 接入调用 HTTP 服务的操作步骤如下：

1. 引入 SDK
2. 创建 apiClient
3. 发起调用请求

引入 SDK

您需要在本地工程 `pom.xml` 中添加如下 SDK。

```
<dependency>
      <groupId>com.alipay.sofa</groupId>
      <artifactId>gateway-sdk</artifactId>
       <version>2.5.0</version>
</dependency>
```

创建 apiClient

使用获取的 AK、SK、host 创建一个 apiClient，代码示例如下：

```
    private String subAppAccessKey = "m9hoHiqtjylG****";

    private String subAppSecretKey = "sxCg1uk6UjGARr1hPswE0W2KHe86****";

    private String gatewayUrl = "5jzkcn0idt3w****.apigateway.inc.alipay.net";

  private final String publicKey = "-----BEGIN PUBLIC KEY-----\n" +
            "MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQClT8sFlj+Abb3ZD6ul039SUGy/\n" +
            "US9H/P+z/kRKIw/48LYD5s7maw1McKtQcbvvsWDOwOhtWCotDIFIinx/e0+y0Ucw\n" +
            "lQ/fLjpEbau1CSzXBExoBa/zicWx5yhBepL2FNvrqEV+iNn8YgDie6qlzwd3rkqz\n" +
            "Bb4vrn3GbML/7JfS6QIDAQAB\n" +
            "-----END PUBLIC KEY-----";

    private ApiClient apiClient;
    private ApiClient apiClientNoAuth;
    private ApiClient encryptApiClient;

    @Before
    public void initClient() {
        // 初始化请求客户端
        ApiSecretKey apiSecretKey = new ApiSecretKey(subAppAccessKey, subAppSecretKey);
        List<ApiSecretKey> secretKeys = new ArrayList<>();
       //若无需加签不需要添加secretKeys
        secretKeys.add(apiSecretKey);
       //无需加签client
        apiClientNoAuth = new DefaultApiClient(gatewayUrl);
        //需要加签client
        apiClient = new DefaultApiClient(gatewayUrl, secretKeys);
       //需要加密client,加密算法有ECC/RAS/SM,根据使用情况选择
       //注意：需要加密的请求必须带上secretKeys
        encryptApiClient = new DefaultApiClient(gatewayUrl, secretKeys, EncryptEnum.RSA, pu
blicKey);
    //注意：以上的client根据所需场景创建一个即可
    }
```

发起调用请求

apiClient 配置完成之后，即可对 API 发起调用。响应会自动序列化到 response 对象里，直接使用即可，代码示例如下：

```
    @Test
    public void testNoSignHttp() {
        GwSdkApiRequest request = new GwSdkApiRequest();
        request.setPath("simple/demo");//api路径，前面不要带上/,sdk会给path拼接/path的
        request.setRequestType("POST");//请求方法 POST/GET/PUT/DELETE
       //是否开启加密,默认为false
        request.setNeedEncrypt(false);

        // 是否对响应进行签名校验
        request.setClientCheckSign(false);

        //若网关没有做泛域名代理或者dns解析需要传请求头 key : x-gateway-host ; value : api分组域
名
        request.getHeaderParams().put("x-gateway-host", CommonUtils.buildDomain(getGroupId(
)));
    /*填写入参，他是一个object可以将实体类作为参数传进来，若想使用query方式传参需要将参数拼接至gatewa
yUrl
       注意：若api后端协议是SOFARPC/DUBBO入参必须和后端服务接口入参完成一直，
        否则会出现序列化失败或者找不到接口的情况，参数字段为空也需要传{"key":null}
       **/
        request.setBody("{\"name\":\"tom\",\"age\":null}");
        ApiResponse response = apiClient.execute(request);

        System.out.println(JSON.toJSONString(response));
    }
```

> **说明**
>
> 当前网关需要通过域名辅助进行 API 匹配，当域名无法直接访问时，可以通过 header 传递，具体方法请参加下文 通过 header 设置 host 。

通过 header 设置 host

当网关的域名无法直接访问时，您可以发送请求到网关，通过 header 设置 host 进行访问。

示例请求代码如下：

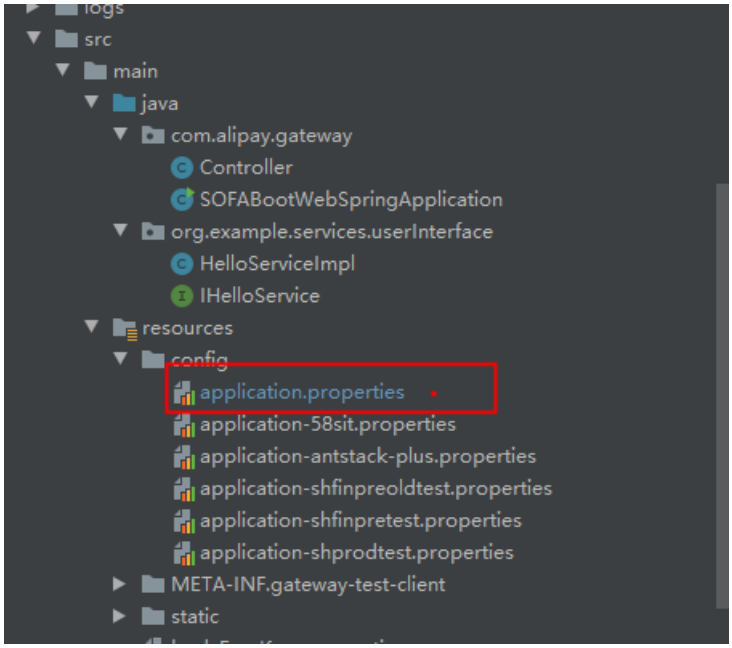
> **说明**
>
> 在发送请求到网关时，您需要获取 **API 分组域名** 和 **API 网关公网域名** 信息。

```
    private String subAppAccessKey = "m9hoHiqtjylG****";

    private String subAppSecretKey = "sxCg1uk6UjGARr1hPswE0W2KHe86****";

 // ！！！直接将请求发送给网关，域名可以随便定义
    private String gatewayUrl = "http://apigateway.alipay.com";

    private ApiClient apiClient;

    @Before
    public void initClient() {
        // 初始化请求客户端
        ApiSecretKey apiSecretKey = new ApiSecretKey(subAppAccessKey, subAppSecretKey);
        List<ApiSecretKey> secretKeys = new ArrayList<>();
        secretKeys.add(apiSecretKey);
        apiClient = new DefaultApiClient(gatewayUrl, secretKeys);
    }

    @Test
    public void testHost() {
        ParamPostRequest request = new ParamPostRequest();
        request.setPath("simple/demo");

        // ！！！ 通过 header 设置分组 host ！！！
        request.getHeaderParams().put("x-gateway-host","5jzkcn0idt3w****.apigateway.inc.alipay.net");

        // 是否对响应进行签名校验
        request.setClientCheckSign(false);
        ApiResponse response = apiClient.execute(request);
        System.out.println(JSON.toJSONString(response));
    }
```

注意事项

当 API 在测试环境完成测试后，通过导入导出功能把 API 信息迁移到生产环境，此时 API 对应的分组域名会发生变换（分组名称不变），所以最好通过在配置文件中配置域名的方式，将分组域名引入生产环境，方便客户端区分生产环境和测试环境。

以下图为例，在项目文件中，将以下代码粘贴到对应环境的 `*.properties` 文件中。


logs
src
main
java
com.alipay.gateway
Controller
SOFABootWebSpringApplication
org.example.services.userInterface
HelloServiceImpl
IHelloService
resources
config
application.properties
application-58sit.properties
application-antstack-plus.properties
application-shfinpreoldtest.properties
application-shfinpretest.properties
application-shprodtest.properties
META-INF.gateway-test-client
static

```
在application.properties中配置
subAppAccessKey=xxx
subAppSecretKey=xx
gatewayUrl=http://apigateway.alipay.com
#如果有多个分组可以配置多个key-value域名
apihost=aaaa.apigateway.inc.alipay.net
apihost1=xxx.apigateway.inc.alipay.net
apihost1=zzzz.apigateway.inc.alipay.net
```

```
@Value("${subAppAccessKey}")
private String subAppAccessKey ;
@Value("${subAppSecretKey}")
   private String subAppSecretKey;

// ！！！直接将请求发送给网关，域名可以随便定义
@Value("${gatewayUrl}")
   private String gatewayUrl;
@Value("${apihost}")
   private String apihost;
   private ApiClient apiClient;

   @Before
   public void initClient() {
       // 初始化请求客户端
       ApiSecretKey apiSecretKey = new ApiSecretKey(subAppAccessKey, subAppSecretKey);
       List<ApiSecretKey> secretKeys = new ArrayList<>();
       secretKeys.add(apiSecretKey);
       apiClient = new DefaultApiClient(gatewayUrl, secretKeys);
   }

   @Test
   public void testHost() {
       ParamPostRequest request = new ParamPostRequest();
       request.setPath("simple/demo");

       // ！！！ 通过 header 设置分组 host ！！！
       request.getHeaderParams().put("x-gateway-host",apihost);

       // 是否对响应进行签名校验
       request.setClientCheckSign(false);
       ApiResponse response = apiClient.execute(request);
       System.out.println(JSON.toJSONString(response));
   }
```

5.3. 发送 HTTP 请求（NodeJS）

本文介绍 API 订阅方应用如何使用 NodeJS 语言发送 HTTP 请求接入网关。

前提条件

在进行本地应用开发前，您需要确保已经完成以下操作：

- 已在 API 网关控制台创建了一个应用。
- 已将该应用的 APPID 提供给了 API 发布者，并获得了授权。
- 已获取了如下服务配置信息。具体获取方法，参见 获取服务端信息。
  - 应用的访问密钥（Access Key/Secret Key）
  - API 的域名地址（host）、请求路径（path）、方法（method）、请求体（body）

操作步骤

发送请求到网关本质上就是一个简单的 HTTP 请求，但是由于当前 API 网关要求订阅方必须加签，所以在发起 HTTP 请求之前，您需要生成一个加签信息，并加入请求 Header 中。

1. 从网关获取订阅方应用的密钥（Access Key/Secret Key）。
2. 获取 API 的 host、method、path，并组装好必要的参数。
3. 引入 `sofa-signature-js` ，对请求进行加签。
   i. 若拉取不到 sofa-signature-js 则执行 `npm i sofa-signature-js` 下载。
   ii. 访问 https://www.npmjs.com/package/sofa-signature-js 执行命令 npm 拉取。
4. 组装请求。
5. 使用 HttpClient（fetch、axios 等）发起 HTTP 请求到 API 网关。

NodeJS 代码示例

```
import {
  Signature,
  SignatureAlgorithm,
  HTTPDigestWithBase64,
  DigestAlgorithm
} from "sofa-signature-js";

import fetch from 'node-fetch';

// api 信息
const gateway = "5jzkcn0idt3w****.apigateway.inc.alipay.net";
const method = "POST";
const path = "/simple/demo";
const headers = {
  'Content-Type': 'application/json'
};


// app ak
const accesskey = "m9hoHiqtjylG****";
// app sk
const secretkey = "sxCg1uk6UjGARr1hPswE0W2KHe86****";
```

```
// 为 request body 生成 digest，可以保证 request body 不被篡改
const digest = HTTPDigestWithBase64(DigestAlgorithm.HMACSHA256, body);
headers["digest"] = digest;

// 创建加签对象
const sign = new Signature(
  accesskey,
  secretkey,
  SignatureAlgorithm.HMACSHA256,
  signheaders
);

// signheaders 决定对哪些 header 进行加签
const signheaders = ["(request-target)", "digest"];
// 生成签名
const authorization = sign.doSignature(secretkey, headers, method, path);
// 将签名放到 header
headers["Signature"] = authorization;


const body = JSON.stringify({
  name:'jack',
});

// 发起请求
fetch(`${gateway}${path}`,{
  method,
  headers,
  body
})
.then(function(res) {
  return res.json();
})
```

5.4. 发送 HTTP 请求（C# ）

本文介绍 API 订阅方应用如何使用 C# 语言对 HTTP 请求加签、验签。 API 网关提供了 C# SDK，即 mosng-sdk-csharp.zip。该 SDK 集成了加签、验签的逻辑，同时默认支持序列化和反序列化。

前提条件

在进行本地应用开发前，您需要确保已经完成以下操作：

- 已在 API 网关控制台创建了一个应用。
- 已将该应用的 APPID 提供给了 API 发布者，并获得了授权。
- 已获取了如下服务配置信息。具体获取方法，参见 获取服务端信息。
  - 应用的访问密钥（Access Key/Secret Key）
  - API 的域名地址（host）、请求路径（path）、方法（method）、请求体（body）

操作步骤

1. 获取并安装 Visual Studio，可至 Visual Studio 官方网站下载。

2. 引入 C# SDK。

   i. 下载 SDK 及二进制文件。

      a. 请访问 mosng-sdk-csharp.zip 和 HttpSignatures.dll.zip 下载 SDK 和 SDK 的二进制文件。 下载“mosng-sdk-csharp.zip”压缩包并解压进入对应的目录后，双击 SDK 包“signature.sln”文件，将 C# 的 SDK 引入Visual Studio。其中，HttpSignatures 项目为实现签名算法的共享库，可用于 .Net Framework 与 .Net Core 项目。“HttpSignatureTests”项目为调用示例。

   ii. 解压 mosng-sdk-csharp.zip 进入对应的目录后，双击 SDK 包 `signature.sln` 文件，将 C# SDK 引入 Visual Studio。

   其中，HttpSignatures 项目为实现签名算法的共享库，可用于 `.Net Framework` 与 `.Net Core` 项目。HttpSignatureTests 项目为调用示例。

3. 定义要加签的 headers 和算法。

```
var spec =newSignatureSpecification(){
Algorithm="hmac-sha256",
Headers=newstring[]{"content-length","host","date","(request-target)"},
KeyId= keyId
};
```

4. 构建 request 对象。 您需要构建 request 对象，用于组装请求，代码示例如下：

```
var request =newRequest();
 request.Path="/foo/Bar";
 request.Method=HttpMethod.Get;
 request.SetHeader("content-length","18");
 request.SetHeader("host","example.org");
 request.SetHeader("date","Tue, 07 Jun 2014 20:51:35 GMT");
```

5. 构建一个 signer 对象。

   AuthorizationParser 对象包含 hmac-sha1、hmac-sha256、hmac-sha512 三种算法。
   HttpSignatureStringExtractor 对象初始化，用来加签 request 请求 header 中的字段。

   代码示例如下：

```
var signer =newHttpSigner(newAuthorizationParser(),newHttpSignatureStringExtractor());
 signer.Sign(request, spec, keyId, privateKey);
```

6. 签名计算。

   计算签名之后，签名信息放在 request 的 header[“Signature”]里。

   代码示例如下：

```
var signer =newHttpSigner(newAuthorizationParser(),newHttpSignatureStringExtractor());
 signer.Sign(request, spec, keyId, privateKey);
```

7. 签名验证。

   取出 request 的 header[“Signature”]里的加签信息并查询验证结果。

   代码示例如下：

```
string s = signer.CalculateSignature(request, spec, privateKey);
Console.WriteLine(s);
Assert.AreEqual("yT/NrPI9mKB5R7FTLRyxxxx+QLQOEAvbGmauxxxx+Jg=", s);
```

完整样例

```
[Test]
publicvoidSignRequestTest2()
{
String authorization ="Signature keyId=\"hmac-key-1\",algorithm=\"hmac-sha256\",headers=\"c
ontent-length host date (request-target)\",signature=\"yT/NrPI9mKB5R7FTLRyFWvB+QLQOEAvbGmau
C0tI+Jg=\"";
String keyId ="hmac-key-1";// accessKey
String privateKey ="don't tell";// secretKey

// 定义要加签的headers和算法
var spec =newSignatureSpecification()
{
Algorithm="hmac-sha256",
Headers=newstring[]{"content-length","host","date","(request-target)"},
KeyId= keyId
};

// http 请求
var request =newRequest();
     request.Path="/foo/Bar";
     request.Method=HttpMethod.Get;
     request.SetHeader("content-length","18");
     request.SetHeader("host","example.org");
     request.SetHeader("date","Tue, 07 Jun 2014 20:51:35 GMT");

// 签名计算，计算之后，签名信息放在request的 header["Signature"] 里
var signer =newHttpSigner(newAuthorizationParser(),newHttpSignatureStringExtractor());
     signer.Sign(request, spec, keyId, privateKey);
Console.WriteLine(request.GetHeader("Signature"));
Assert.AreEqual(authorization, request.GetHeader("Signature"));

// 独立的签名计算方法
string s = signer.CalculateSignature(request, spec, privateKey);
Console.WriteLine(s);
Assert.AreEqual("yT/NrPI9mKB5R7FTLRyFWvB+QLQOEAvbGmauC0tI+Jg=", s);
}
```

加签算法

```
Base64(hmac-sha256(signing_string, Encoding.UTF8.GetBytes(private_key)))
Base64(hmac-sha512(signing_string, Encoding.UTF8.GetBytes(private_key)))
Base64(hmac-sha1(signing_string, Encoding.UTF8.GetBytes(private_key)))
```

- signing_string：加签的字符串，根据 SignatureSpecification 里的 Headers 里定义的头生成。
- secret_key：私钥，用 UTF-8 编码转成字节数组。

限制说明

目前没有对 HTTP 请求的 body 做摘要（digest）加签。

5.5. SOFARPC 客户端接入

本文适用于订阅方使用 SOFARPC 作为客户端接入 API 网关。

前置条件

在进行本地订阅方应用开发、接入网关前，需要确保已经完成以下操作：

- 已在 API 网关控制台创建了一个应用。
- 已将该应用的 APPID 提供给了 API 发布者，并获得了授权。
- 已获取了如下服务配置信息。具体获取方法，参见 获取服务端信息。
  - 应用的访问密钥（Access Key/Secret Key）。
  - API 的域名地址（host）、接口名称（interface）、方法（method）等。

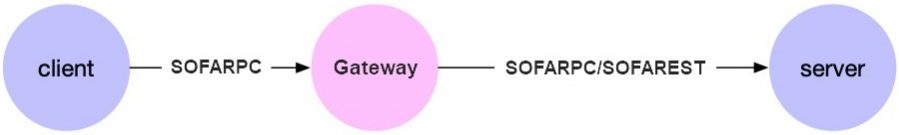
操作步骤

跟进 API 后端服务系统协议类型的不同，SOFARPC 订阅方接入步骤也有所不同。

- API 后端服务协议为：SOFARPC/SOFAREST
- API 后端服务协议为：HTTP

SOFARPC/SOFAREST

在 API 的后端服务系统协议为 SOFARPC/SOFAREST 时，其网络拓扑如下图所示：



API 服务示例

假设服务端发布了如下 SOFARPC 或 SOFAREST 服务，且已在 API 网关控制台上完成了相应 API 配置。

```
package com.alipay.gateway.serverdemo.service

publicinterfaceHelloService{
publicString sayHello();
}
```

API 调用示例

根据上述 API 服务示例，客户端可以使用标准的 SOFARPC reference 编码方式进行调用，只需添加几个特殊的网关参数即可。

1. 在 `application-properties` 文件中，添加以下配置：

> 说明
>
> 若以下配置已经存在，请确认配置无误。

```
# 走网关寻址
com.alipay.sofa.rpc.registries.gateway=gateway://${acvip}:80
com.alipay.instanceid=000001
com.antcloud.antvip.endpoint=acvip
```

2. 引入服务端提供的 facade 包。
3. 根据上文 前置条件 中获取的 API 服务端信息，修改 reference 方式，示例如下：

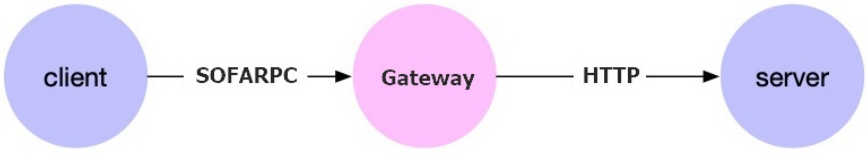
```
<sofa:reference jvm-first="false" id="helloService"
interface="com.alipay.gateway.serverdemo.service.HelloService">
<sofa:binding.bolt>
  //通过acvip获取网关rpc地址
<sofa:global-attrs connect.timeout="10000" registry="gateway"/>
  //也可以通过直连的方式填写网关地址test-url=网关的slbIp:12222
  <!--            <sofa:global-attrs connect.timeout="10000" test-url="127.0.0.1:12222"
serialize-type="json"/>-->
<sofa:parameter key="gateway.host" value="eq1ejclxfgde****.cloud.58dev.alipay.net"/>
//开启数据加密后，在调用请阅的 API 时必须传入应用的 AK 和 SK 完成加签。
<sofa:parameter key="gateway.ak" value=<yourAccessKeyId>
<sofa:parameter key="gateway.sk" value=<yourAccessKeySecret>
</sofa:binding.bolt>
</sofa:reference>
```

4. 调用 HelloService。

```
publicclassClient{

@Autowire
privateHelloService helloService;

// use helloService
}
```

HTTP

在 API 的后端服务系统协议为 HTTP 时，其网络拓扑如下图所示：



与 SOFARPC 和 SOFAREST 不同的是，一般 HTTP 服务的服务端不会提供 facade 包，但是使用 SOFARPC 调用需要一个 interface，所以客户端需要自行构造这个 interface。

API 服务示例

假设服务端是一个 SpringMVC 接口，如下所示：

```
package com.alipay.gateway.serverdemo.controller

// 服务端提供一个 /hello/world 服务

@RestController
@RequestMapping("/hello")
publicclassSpringController{

@RequestMapping("/world")
publicStringHello(){
return"world";
}
}
```

API 服务提供者在 API 网关控制台完成了相应的 API 配置，如下图所示。

API 调用示例

根据上述 API 服务示例，服务提供者要求客户端通过 `com.alipay.gateway.client.service.xxxx.hello()` 的方式来调用服务端提供的 HTTP 接口。

此时，客户端需要按照如下步骤进行编码配置：

1. 在 `application-properties` 文件中，添加以下配置：

> 说明
>
> 若以下配置已经存在，请确认配置无误。

```
# 走网关寻址
com.alipay.sofa.rpc.registries.gateway=gateway://${acvip}:80
com.alipay.instanceid=000001
com.antcloud.antvip.endpoint=acvip
```

2. 构造 interface，示例如下：

```
package com.alipay.gateway.client.service

// client 使用 HelloSpirngMVC.Hello 来访问服务端的  /hello/world 服务
publicinterfaceHelloSpirngMVC{
String hello();
}
```

3. 根据上文 前置条件 中获取的 API 服务端信息，修改 reference 方式，示例如下：

```
<sofa:reference jvm-first="false" id="helloSpirngMVC"
interface="com.alipay.gateway.client.service.HelloSpirngMVC">
<sofa:binding.bolt>
<sofa:global-attrs connect.timeout="10000" registry="gateway"/>
<sofa:parameter key="gateway.host" value="eq1ejclxfgde5j7p.xxxxx.alipay.net"/>
<sofa:parameter key="gateway.ak" value="NbpzogN60sKoxxxx"/>
<sofa:parameter key="gateway.sk" value="qPMF34KZ3ja2hSpRnzB6xQwatDxxxxxx"/>
</sofa:binding.bolt>
</sofa:reference>
```

4. 调用 HelloSpirngMVC。

```
publicclassClient{

@Autowire
privateHelloSpirngMVC helloSpirngMVC;

// use helloSpirngMVC
}
```

5.6. DUBBO 客户端接入

网关提供客户端 DUBBO Demo 代码供用户参考，下载客户端 Demo后，DUBBO Demo 代码位置为：com/alipay/gateway/web/test/usercase/mainchain/dubbo/DubboClientServiceTest.java。

操作步骤

1. 编写后端 Server。

   例如：

```
@Controller
@RequestMapping("/springmvc/test")
public class PostMvcServerController {

    /**
     * 测试场景： 没有参数的测试
     *
     * @return
     */
    @RequestMapping(value = "/param/noParam", method = RequestMethod.POST)
    @ResponseBody
    public Map<String, String> hello() {
        Map<String, String> map = new HashMap<>();
        map.put("post", "hello");
        System.out.println(JSONObject.toJSONString(map));
        return map;
    }
}
```

2. 登录网关控制台。
3. 创建系统集群，并使其指向本地后端 Server。

> **注意**
>
> 如果此处系统集群的认证方式选择了 **密钥**，由于网关目前仅提供 JavaSDK，其他语言类型需用户自行实现。

4. 创建分组和 API，指定前端的调用方式和后端的转发方式。

   如下图所示：

> **注意**
>
> DUBBO 类型的 API 暂不支持签名认证和数据加密等需要客户端配置的需求。

5. 编写调用方代码。

- 编写调用代码

```
package com.alipay.gateway.facade.dubbo;

public interface GatewayDubbo2HttpService {

    Map<String, String> stringParam(String param);
}
```

- 编写 DUBBO Reference

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:dubbo="http://dubbo.apache.org/schema/dubbo"
       xsi:schemaLocation="http://www.springframework.org/schema/beans        http://
www.springframework.org/schema/beans/spring-beans-4.3.xsd        http://dubbo.apache.
org/schema/dubbo        http://dubbo.apache.org/schema/dubbo/dubbo.xsd">

    <!-- 消费方应用名，用于计算依赖关系，不是匹配条件，不要与提供方一样 -->
    <dubbo:application name="consumer-of-helloworld-app"  />
     <!-- 协议必须是fastjson序列化 -->
    <dubbo:protocol name="dubbo" serialization="fastjson" />

    <!-- 生成远程服务代理，可以和本地bean一样使用demoService url=网关ip:20888 -->
    <dubbo:reference id="gatewayDubbo2HttpService" interface="com.alipay.gateway.faca
de.dubbo.GatewayDubbo2HttpService" timeout="100000" protocol="dubbo" url="网关ip:20888
?serialization=fastjson" version="1.0.0" >
    </dubbo:reference>
</beans>
```

- DUBBO 调用

从 API 详情获取分组 ID 并写入 `x-mosng-host` 。

```
public class DubboClientServiceTest extends AbstractTestBase {


 @Test
    public void dubbo2HttpStringParam() {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                "classpath:META-INF/gateway-test-client/dubbo-consumer.xml");
        RpcContext.getContext().setAttachment("x-mosng-host", "lv51bpe4v3kiy6nn");

        GatewayDubbo2HttpService gatewayDubbo2HttpService = (GatewayDubbo2HttpService
)context.getBean("gatewayDubbo2HttpService"); // 获取远程服务代理
    //注意：入参需要和在网关页面上配置的类型一致，且响应参数需要和dubbo对齐否则会报错
        Map<String, String> dubbo2HttpStringParam = gatewayDubbo2HttpService.stringPa
ram("aaa");
        System.out.println(JSON.toJSONString(dubbo2HttpStringParam)); // 显示调用结果
    }
}
```

6. 发起请求。

调用结果如下：