API 网关

用户指南（调用 API）

用户指南（调用 API）

概述及样例

概述

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

调用 API

您可以直接用 API 网关控制台为您提供的多语言调用示例来测试调用。您也可以自行编辑 HTTP(s) 请求调用 API 。签名方式您可以参照控制台的 SDK示例下载。

API 调用方式说明及示例如下：（调用 API 前期流程请参照 快速入门（调用 API））。

1.请求

请求地址

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

请求方法

```
POST
```

请求体

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

请求头部

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

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

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

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

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

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

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

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

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

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

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

2.响应

状态码

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

响应头

```
X-Ca-Request-Id: 7AD052CB-EE8B-4DFD-BBAF-EFB340E0A5AF
//请求唯一 ID，请求一旦进入 API 网关应用后，API 网关就会生成请求 ID 并通过响应头返回给客户端，建议客户端与后端服
```

```
务都记录此请求 ID，可用于问题排查与跟踪。

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

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

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

签名的计算 demo 请参照 API 网关控制台 SDK下载 页面的 SDK 示例。

若需要了解更多详情，请您查看 用户指南（调用API）。

应用（App）

您需要创建应用（APP）作为您调用 API 的身份，每个 APP 有一对 AppKey 和 AppSecret 密钥对，AppKey 需要在请求时作为参数在 Header 传入，AppSecret 需要用于计算请求签名。

1. 在 API 网关，您需要创建应用（APP）作为请求者的身份。APP 创建时，系统会自动分配一对 AppKey 和 AppSecret。在您请求 API 时，需要用到密钥。详细加密签名的计算传递方式，请查看文档——请求签名说明文档。
2. AppKey 和 AppSecret 密钥对，具备该 APP 的全部权限，需要妥善保管。如果发生泄漏，您可以在 API 网关的控制台进行重置。
3. 您可以拥有多个 APP，可以根据您的业务需求分别被授权不同的 API。注意，API 的授权对象是 APP 而不是阿里云用户账号。
4. 您可以在 API 网关控制台完成对 APP 的创建、修改、删除、查看详情、密钥管理、查看已授权等管理操作。

授权

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

- 如果您在市场购买了 API，那么您有权操作已购买的 API 授权给您的 APP；如果您没有 APP,购买时云市场会为您创建一个 APP，并且授权，具体请前往云市场的 控制台 查看。
- 如果您想要使用合作伙伴提供的 API，并无购买行为，是线下协议。那么需要 API 的提供者主动操作授权。您需要自行创建 APP，并且向 API 的提供者提供您的 AppId。

加密签名

您调用 API 时，需要拼接签名字符串，并将签名计算后的字符串放在请求的 Header 传入，网关会通过对称计算签名来验证请求者的身份。

1. 在请求的 Header 需要传入一个计算后的签名字符串。
2. 您需要将入参信息按照请求签名说明文档组织成为 String to sign，再用 SDK 样例中的算法计算签名。计算结果就是上面提到的计算后的签名字符串。
3. HTTP 和 HTTPS 请求，都需要加入请求签名。

请求签名 String to sign 的组织方法详见请求签名说明文档，您只需要在 SDK 样例中，将 AppKey、AppSecret 换成您自己的真实密钥值，再根据实际 API 文档按照签名文档组织 String to sign，您就可以成功发起请求了。

使用限制

限制项	是否可调整	调整方式
每个账号下 app 个数上限为30个，app 名称应为账号下唯一	是	工单
调用 API 的流控限制为，单个 IP，QPS 不超过100	否	-
单次 API 调用请求包最大 2MB	否	-
您有权操作购买的 API 与 app 的授权和解除授权。由服务提供方授权给您 app 的 API，您无权操作解除授权	否	-
您的请求需要包含签名信息，请参照文档 请求签名说明文档	否	-

请求签名说明文档

域名

- 每个 API 服务都属于一个 API 分组，每个 API 分组有不同的域名。域名是由服务端绑定的独立域名，API 网关通过域名来寻址定位 API 分组。
- 域名的格式为 www.[独立域名].com/[Path]?[HTTPMethod]。
- API 网关通过域名定位到一个唯一的分组，通过 Path+HTTPMethod 确定该分组下唯一的 API。
- 您购买后在控制台 **已购买的 API** 可以获得 API 文档说明，若无购买行为，则可以联系 API 提供者操作授权，授权后您就可以在控制台 **已授权的 API** 获得 API 文档说明。

系统级 Header

- 【必选】X-Ca-Key：AppKey。
- 【必选】X-Ca-Signature：签名字符串。
- 【可选】X-Ca-Timestamp：API 调用者传递时间戳，值为当前时间的毫秒数，也就是从1970年1月1日起至今的时间转换为毫秒，时间戳有效时间为15分钟。
- 【可选】X-Ca-Nonce：API 调用者生成的 UUID，结合时间戳防重放。
- 【可选】Content-MD5 当请求 Body 非 Form 表单时，可以计算 Body 的 MD5 值传递给云网关进行 Body MD5 校验。
- 【可选】X-Ca-Stage请求 API 所属 Stage，目前仅支持 TEST 、PRE 和 RELEASE，默认 RELEASE，若您调用的 API 不在线上环境，请一定要指定该参数的值，否则会报 URL 错误。

签名校验

组织参与签名计算的字符串

```
String stringToSign=
HTTPMethod + "\n" +
Accept + "\n" + //建议显示设置 Accept Header。当 Accept 为空时，部分 Http 客户端会给 Accept 设置默认值为 */*，导致签名校验失败。
Content-MD5 + "\n"
Content-Type + "\n" +
Date + "\n" +
Headers +
Url
```

HTTPMethod 为全大写，如 POST。

Accept、Content-MD5、Content-Type、Date 如果为空也需要添加换行符”\n” ，Headers如果为空不需要添加”\n” 。

Content-MD5

Content-MD5 是指 Body 的 MD5 值，**只有当 Body 非 Form 表单时才计算 MD5**，计算方式为：

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

bodyStream 为字节数组。

Headers

Headers 是指参与 Headers 签名计算的 Header 的 Key、Value 拼接的字符串，建议对 X-Ca 开头以及自定义 Header 计算签名，注意如下参数不参与 Headers 签名计算：X-Ca-Signature、X-Ca-Signature-Headers、Accept、Content-MD5、Content-Type、Date。

- Headers 组织方法：

先对参与 Headers 签名计算的 Header的Key **按照字典排序**后使用如下方式拼接，如果某个 Header 的 Value 为空，则使用 HeaderKey + “:” + “\n”参与签名，需要保留 Key 和英文冒号。

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

将 Headers 签名中 Header 的 Key 使用英文逗号分割放到 Request 的 Header 中，Key为：X-Ca-Signature-Headers。

Url

Url 指 **Path + Query + Body** 中 **Form** 参数，组织方法：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 **Query** 或 **Form** 参数为空，则 Url = Path，不需要添加 **？**，如果某个参数的 **Value** 为空只保留 **Key** 参与签名，等号不需要再加入签名。

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

注意这里 **Query** 或 **Form** 参数的 **Value** 可能有多个，多个的时候只取第一个 Value 参与签名计算。

计算签名

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

secret 为 APP 的密钥。

传递签名

将计算的签名结果放到 Request 的 Header 中，Key为：X-Ca-Signature。

签名错误排查方法

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

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

因为 HTTP Header 中无法表示换行，因此 StringToSign 中的换行符都被过滤掉了，对比时请忽略换行符。

签名 demo

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

在 API 网关控制台，**调用 API—SDK 下载** 处还有更多语种的调用 demo。

通过Postman实现API网关的请求签名与调试

通过Postman实现API网关的请求签名与调试

1. 前言

Postman是一个非常强大的HTTP发包测试工具, 目前Postman已经提供了Windows/Mac/Linux系统的客户端的下载，使用很方便。不过API网关的调试，需要对HTTP请求进行签名才能调用，无法使用简单的curl等发包工具完成，但我们可以使用Postman工具提供的Pre-request Script脚本来实现API网关的签名功能，实现API的调试功能，本文主要介绍。

2. API网关签名算法介绍

API网关的签名机制详细可以参考请求签名说明文档，这里简要介绍一下。

API网关的签名需要通过API网关的AppKey和AppSecret进行，Key/Secret可以在API网关的控制台上获得，并确保API已经发布，并且针对特定的APP做了授权操作。

针对一个普通请求，API网关的签名过程如下

2.1. 添加以下头用于辅助签名与安全认证

```
- Date: 日期头
- X-Ca-Key：{AppKey}
- X-Ca-Nonce：API调用者生成的 UUID, 实现防重放功能
- Content-MD5: 当请求Body为非Form表单时，用于校验Body是否被篡改,
```

2.2. 组织需要签名的字符串StringToSign

```
{HTTPMethod} + "\n" +
{Accept} + "\n" +
{Content-MD5} + "\n"
{Content-Type} + "\n" +
{Date} + "\n" +
{SignatureHeaders} +
{UrlToSign}
```

- Accept、Content-MD5、Content-Type、Date 如果为空也需要添加换行符”\n”
- 只有From为非表单的方式才需要计算Content-MD5，计算方法为 base64Encode(md5(body.getBytes("UTF-8"))
- SignatureHeaders: 以{HeaderName}:{HeaderValue} + "\n"的方式按照字符串顺序从小到大顺序添加, 建议加入签名的头为X-Ca-Key,X-Ca-Nonce, 其他头客户端实现可自行选择是否加入签名。
- UrlToSign: 将所有的Form字段和QueryString字段放在一起按照Name进行排序，如果Content-Type不是application/x-www-form-urlencoded类型则不拆开Form字段。将排序好的键值对加到Path后面得到UrlToSign, 例如请求/demo?c=1&a=2, Form为b=3则 UrlToSign=/demo?a=2&b=3&c=1

2.3. 计算签名并附加签名相关Headers

目前推荐使用HMacSHA256算法计算签名，签名的计算需要appSecret，计算方法为:signature = base64(hmacSHA256(stringToSign.getBytes("UTF-8), appSecret)), 计算完毕后还需要添加以下Headers:

- 添加Header: X-Ca-Siguature:{signature}
- 添加Header: X-Ca-SignatureMethod:HmacSHA256
- 添加Header: X-Ca-SignatureHeaders:X-Ca-Key,X-Ca-Nonce

2.4. 签名错误排查方法

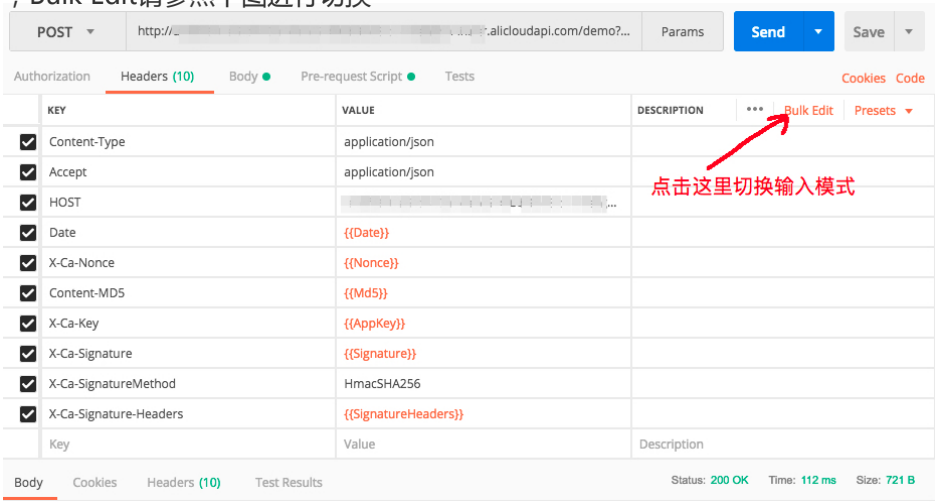
- 当签名校验失败时，API网关会将服务端的StringToSign放到HTTP应答的Header中返回到客户端，Key为：X-Ca-Error-Message，只需要将本地计算的StringToSign与服务端返回的StringToSign进行对比即可找到问题，注意服务端返回的StringToSign将回车替换为了#；
- 如果服务端与客户端的签名串是一致的，请检查用于签名计算的密钥是否正确；

3. 使用Pre-request Script实现签名算法

根据上一节的描述，实现API网关调试的关键问题在于如何实现请求签名，Postman提供了可以通过JavaScript进行定制的, 通过阅读Pre-request Script的开发文档, 我们可以通过Pre-request Script脚本实现API网关的签名功能。

3.1. 使用全局变量预制签名需要添加的头

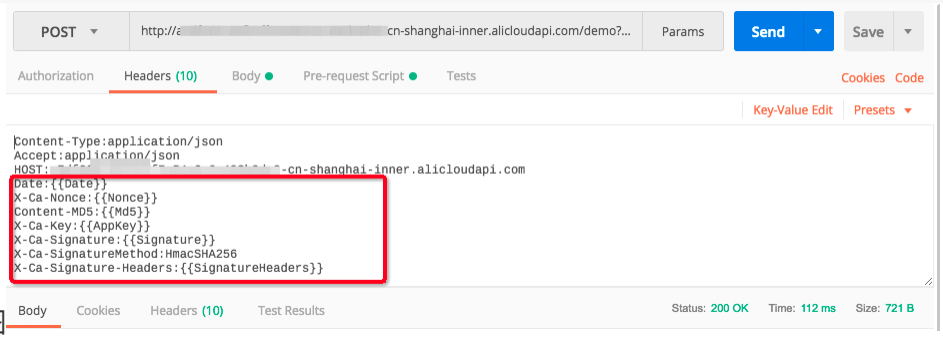
不过目前Postman不允许直接在脚本中修改请求，所以我们只能使用预制签名头并使用全局变量赋值的方式完成签名头的添加，我们将需要签名的头都预制在Postman的请求Header中，可以通过Bulk Edit模式实现添加，Bulk Edit请参照下图进行切换



切换为Bulk Edit模式后，可以将如下字符串复制粘贴到输入框当中，被{{}}括住的就是Postman的全局变量，我们在脚本中实现替换。Form内容的可以不添加Content-MD5头

```
Date:{{Date}}
```

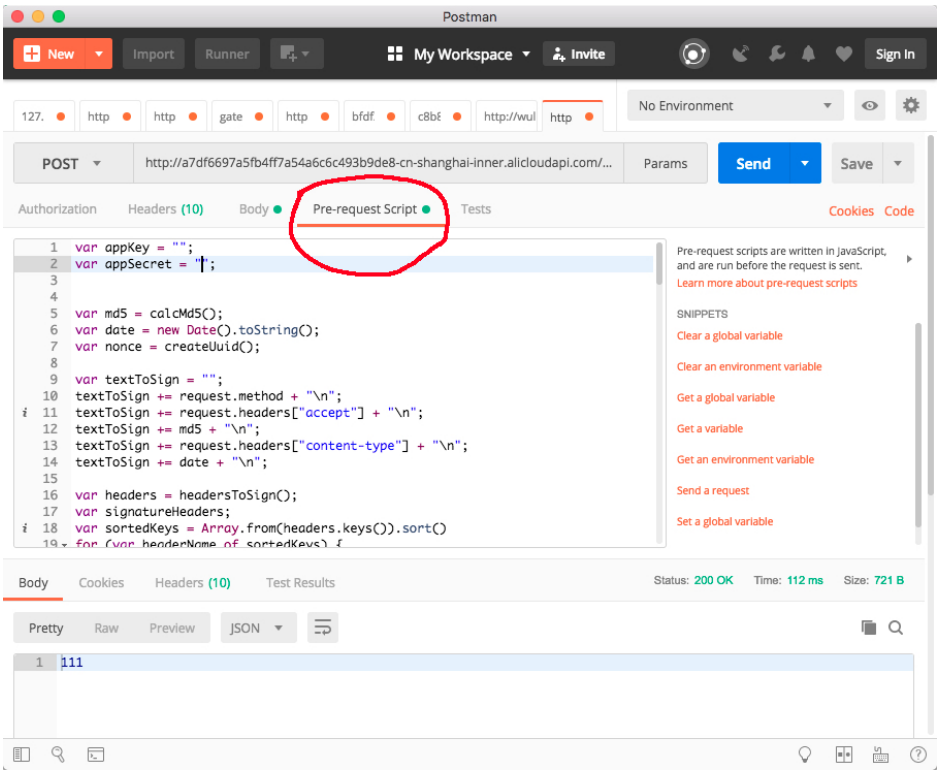
```
Content-MD5:{{Md5}}
X-Ca-Nonce:{{Nonce}}
X-Ca-Key:{{AppKey}}
X-Ca-Signature:{{Signature}}
X-Ca-SignatureMethod:HmacSHA256
X-Ca-Signature-Headers:{{SignatureHeaders}}
```



粘贴后效果如图

3.2. 使用Pre-request Script脚本实现签名功能

点击红圈圈住的位置，可以输入Pre-request Script，请复制粘贴下面提供的代码到文本框当中




```
var appKey = "<YOUR APP KEY>";
var appSecret = "<YOUR APP SECRET>";

var md5 = calcMd5();
var date = new Date().toString();
var nonce = createUuid();
```

```
var textToSign = "";
textToSign += request.method + "\n";
textToSign += request.headers["accept"] + "\n";
textToSign += md5 + "\n";
textToSign += request.headers["content-type"] + "\n";
textToSign += date + "\n";

var headers = headersToSign();
var signatureHeaders;
var sortedKeys = Array.from(headers.keys()).sort()
for (var headerName of sortedKeys) {
textToSign += headerName + ":" + headers.get(headerName) + "\n";
signatureHeaders = signatureHeaders ? signatureHeaders + "," + headerName : headerName;
}
textToSign += urlToSign();
console.log("textToSign\n" + textToSign.replace(/\n/g, "#"));
var hash = CryptoJS.HmacSHA256(textToSign, appSecret)
console.log("hash:" + hash)
var signature = hash.toString(CryptoJS.enc.Base64)
console.log("signature:" + signature)

pm.globals.set('AppKey', appKey);
pm.globals.set('Md5', md5);
pm.globals.set("Date", date);
pm.globals.set("Signature", signature);
pm.globals.set("SignatureHeaders", signatureHeaders);
pm.globals.set("Nonce", nonce);

function headersToSign() {
var headers = new Map();
for (var name in request.headers) {
name = name.toLowerCase();
if (!name.startsWith('x-ca-')) {
continue;
}
if (name === "x-ca-signature" || name === "x-ca-signature-headers" || name == "x-ca-key" || name === 'x-ca-
nonce') {
continue;
}
var value = request.headers[name];
headers.set(name, value);
}
headers.set('x-ca-key', appKey);
headers.set('x-ca-nonce', nonce);
return headers;
}

function urlToSign() {
var params = new Map();
var contentType = request.headers["content-type"];
if (contentType && contentType.startsWith('application/x-www-form-urlencoded')) {
const formParams = request.data.split("&");
formParams.forEach((p) => {
const ss = p.split('=');
params.set(ss[0], ss[1]);
})
```

```
}

const ss = request.url.split('?');
if (ss.length > 1 && ss[1]) {
const queryParams = ss[1].split('&');
queryParams.forEach((p) => {
const ss = p.split('=');
params.set(ss[0], ss[1]);
})
}

var sortedKeys = Array.from(params.keys())
sortedKeys.sort();

var l1 = ss[0].lastIndexOf('/');
var url = ss[0].substring(l1);
var first = true;
var qs
for (var k of sortedKeys) {
var s = k + "=" + params.get(k);
qs = qs ? qs + "&" + s : s;
console.log("key=" + k + " value=" + params.get(k));
}
return qs ? url + "?" + qs : url;
}

function calcMd5() {
var contentType = request.headers["content-type"];
if (request.data && !contentType.startsWith('application/x-www-form-urlencoded')) {
var data = request.data;
var md5 = CryptoJS.MD5(data);
var md5String = md5.toString(CryptoJS.enc.Base64);
console.log("data:" + data + "\nmd5:" + md5String);
return md5String;
} else {
return "";
}
}

function createUuid() {
return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, function(c) {
var r = Math.random()*16|0, v = c == 'x' ? r : (r&0x3|0x8);
return v.toString(16);
});
}
```

接下来我们就可以实现API网关的调试了。