API 网关

认证与安全

产品文档

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2019 腾讯云版权所有

本文档著作权归腾讯云单独所有，未经腾讯云事先书面许可，任何主体不得以任何形式复制、修改、抄袭、传播全部或部分本文档内容。

【商标声明】

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。

【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或模式的承诺或保证。

文档目录

认证与安全
- 认证与安全概述
- 密钥对认证
- 免鉴权
- OAuth2.0
- CAM 访问管理

认证与安全
认证与安全概述

最近更新时间：2019-11-04 17:32:33

为了保护您的 API，避免恶意访问、未授权访问、应用漏洞、黑客攻击等导致的数据损失、资产损失，我们提供了多种 API 认证方式和 API 防护策略。
目前 API 网关主要有 密钥对认证 、免鉴权 、OAuth2.0 三种方式。

> 说明：
> OAuth2.0 鉴权功能目前为灰度测试阶段，如需使用请联系您的客户经理或 提交工单 申请开启。

密钥对认证

最近更新时间：2019-09-25 15:23:21

您可以使用 secret_id 和 secret_key 对您的 API 进行认证管理。secret_id 和 secret_key 成对出现，这里将它们将称为 secret_id/secret_key 对。
在使用 secret_id/secret_key 对认证前，您需要先创建一对 secret_id 和 secret_key。

密钥对鉴权 API

创建 API 时，您可以选择鉴权类型为“密钥对鉴权”。当选择密钥对鉴权类型的 API 发布时，只有使用正确密钥对发起的访问能够通过 API 网关的校验，不携带密钥或携带错误的密钥对不能通过 API 网关的鉴权。
API 创建完成后，您需要使用 “使用计划” 功能将密钥对与 API 或 API所在服务进行绑定。配置详情请参考 使用计划。

密钥内容

【secret_id】示例：AKIDCgOPWjQ6BAxvHtyckhWABJVYSBj548pN 用于标识所使用的哪个密钥，并参与签名计算，传输过程中体现。
【secret_key】示例：ZxF2whO0RhuwnVCj5JMMAuqcDcN2oPrC 用于签名计算，传递过程中无体现。

计算方法

最终发送内容

最终发送的 HTTP 请求内至少包含两个 header：Date 和 X-Date 二选一以及 Authorization，可以包含更多 header。

Date header 的值为 GMT 格式的 HTTP 请求构造时间，例如 Fri, 09 Oct 2015 00:00:00 GMT。

X-Date header 的值为 GMT 格式的 HTTP 请求构造时间，例如 Mon, 19 Mar 2018 12:08:40 GMT。15分钟超时。

Authorization header 的形如 `Authorization: hmac id="secret_id", algorithm="hmac-sha1", headers="date source", signature="Base64(HMAC-SHA1(signing_str, secret_key))"` 。

现对 Authorization 内的各部分分别解释：
【hmac】固定内容，用于标识计算方法。
【ID】其值为密钥内的 secret_id 的值。

【algorithm】加密算法，当前支持的是 hmac-sha1。
【headers】参与签名计算的 header，按实际计算时的顺序排列。
【signature】计算签名后得到的签名，signing_str 是签名内容。

签名计算方法

签名由两部分并根据指定加密算法进行计算，以 hmac-sha1 算法举例：

签名内容

首先生成签名内容，签名内容由自定义的 header 组成，header 内建议至少包含 date，可以包含更多其他 header。

header 按如下要求转换后按顺序排列：

- header 名转换为小写，跟 **ascii 字符**和**ascii 空格字符**。
- 然后附加 header 值。
- 如果不是最后一条需构造签名的 header，附上**ascii 换行字符 \n** 。

例如有两个 header 参与构建签名内容（仅为示例，请根据业务实际情况填写字段）：

> **Date**:Fri, 09 **Oct** 2015 00:00:00 **GMT**
> **Source**:AndriodApp

生成的签名内容为：

> date: Fri, 09 Oct 2015 00:00:00 GMT
> source: AndriodApp

计算签名

将上一步生成的签名内容，使用 Base64(HMAC-SHA1(signing_str, secret_key)) 算法进行计算生成签名，也就是：

- 使用签名内容作为输入信息，密钥内的 secret_key 内容作为密钥，使用 HMAC-SHA1 算法进行计算得出加密签名内容。
- 使用 Base64 对算出的加密签名内容进行转换生成可传递的签名内容。

使用签名

如**最终发送内容**中所示的一样，在 Authorization header 的 signature 处填入上一步计算完成后的签名。

注意事项

header 对应

Authorization 中 headers 位置填入的需要是参与计算签名的 header 的名称，并建议转换为小写，以 ascii 空格分隔。

签名内容生成

排列内容时，请注意 header 名后面跟的冒号和空格，如有遗失也可能导致校验无法通过。SecretId、SecretKey、URL、Host 需要修改为真实信息。

常用语言的签名 Demo>>

免鉴权

最近更新时间：2019-07-31 11:38:42

您在创建自己的 API 时，可勾选是否免鉴权。

- 若勾选免鉴权，则 API 网关在收到匿名请求时，也可以通过认证，同时绑定的使用计划也可生效。
- 若使用使用计划中的密钥做签名认证，则使用计划中的限流生效。
- 若匿名用户访问，则以腾讯云本身每个 API 的最大限流为生效限流。

版权所有：腾讯云计算（北京）有限责任公司

OAuth2.0

最近更新时间：2019-11-04 17:32:22

OAuth2.0 概述

OAuth 2.0 是一个开放授权标准，它允许用户让**第三方应用**访问该用户在某服务的**特定私有资源**但是**不提供账号密码信息**给第三方应用。

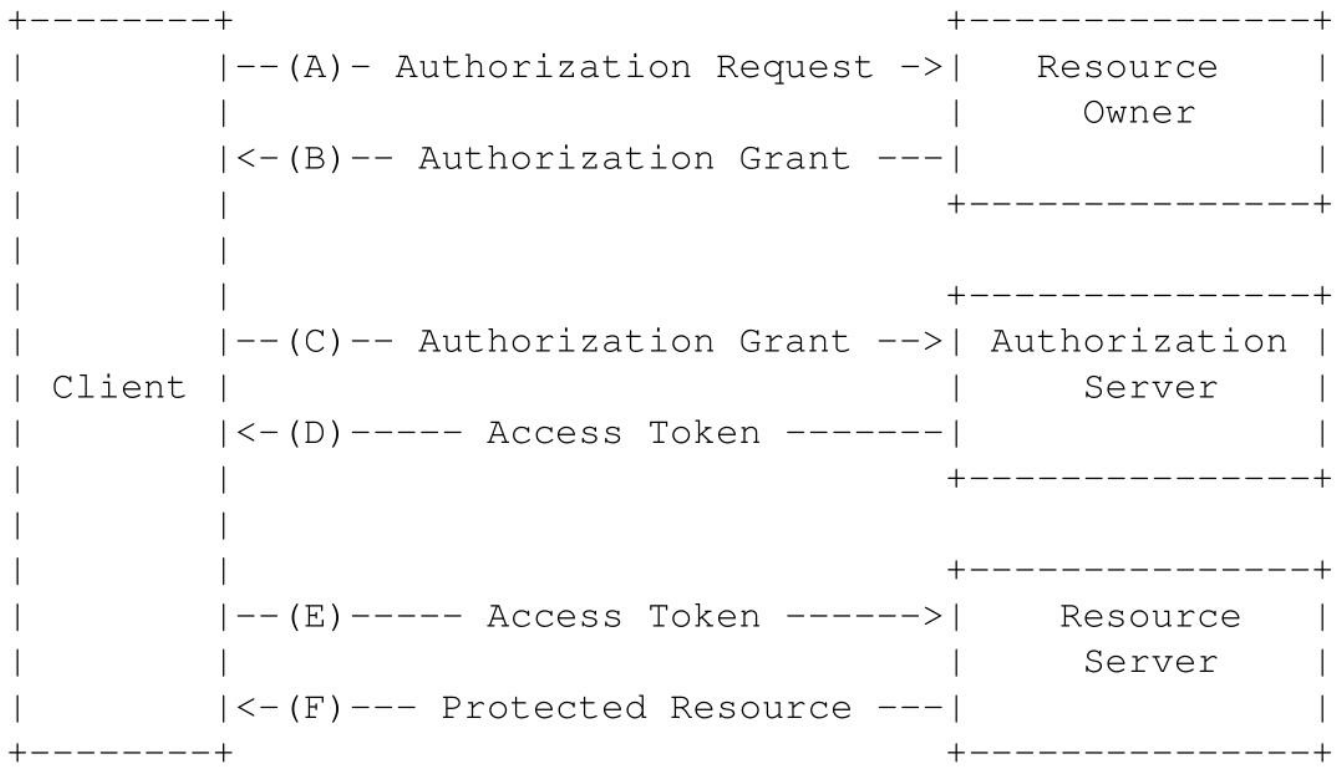
> 说明：
> OAuth2.0 是一个授权协议，不是认证协议。

OAuth2.0 角色说明

OAuth2.0 有以下四个角色：

- Resource Owner：资源拥有者。
- Resource Server：资源服务器。
- Client：第三方应用客户端，指任何可以消费资源服务器的第三方应用。
- Authorization Server：授权服务器，管理上述三个角色的中间层。

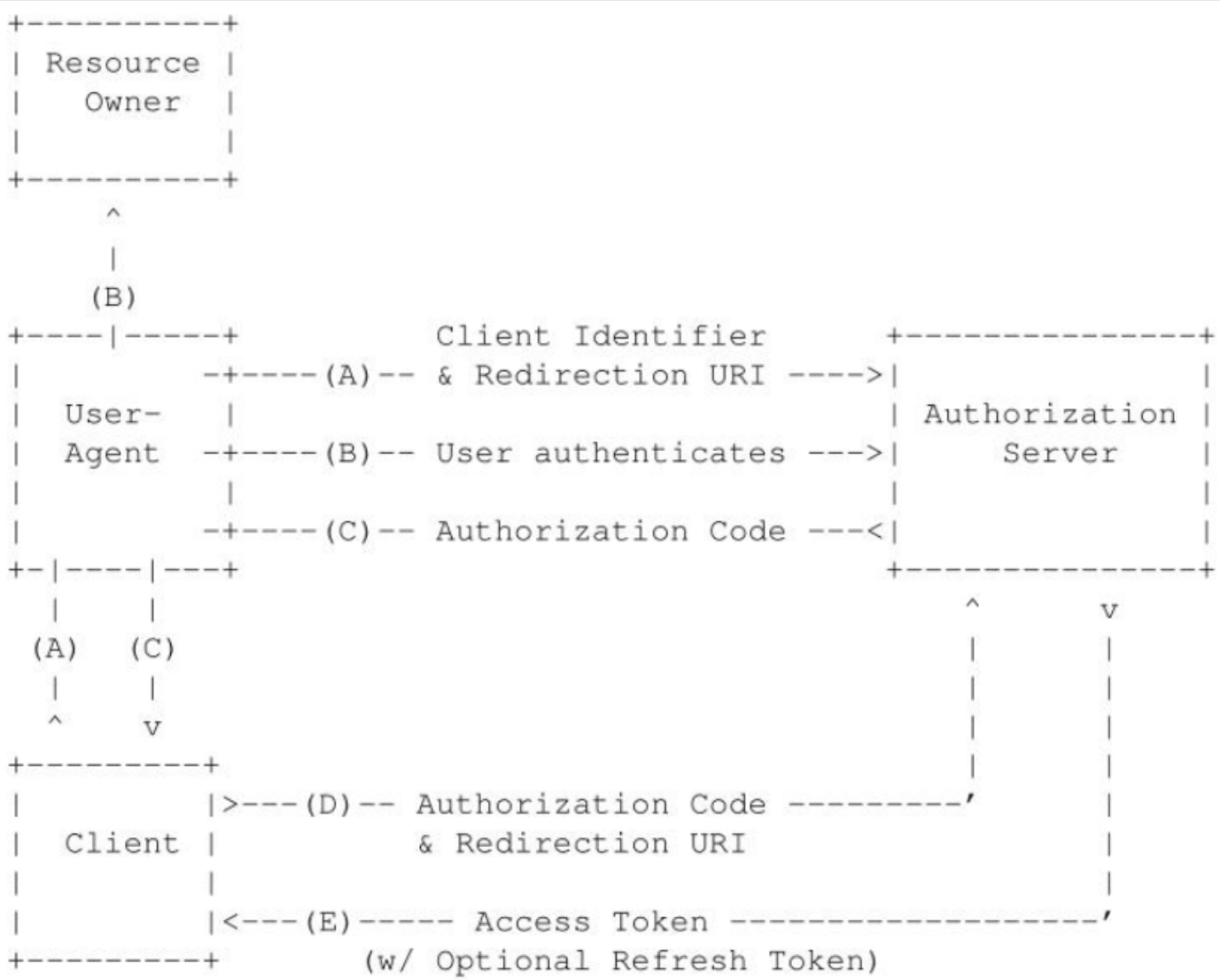
授权流程



（A）：客户端请求资源所有者的授权。
（B）：资源所有者同意授权。
（C）：客户端获得了资源所有者的授权之后，向授权服务器申请授权令牌
（D）：授权服务器验证客户端无误后发放授权令牌
（E）：客户端拿到授权令牌之后请求资源服务器发送用户信息
（F）：资源服务器验证令牌无误后将用户信息发放给客户端

Authorization Grant

- 授权许可是一个代表资源所有者授权（访问受保护资源）的凭据，客户端用它来获取访问令牌。
- Authorization Code：授权码。Client 需要提供 Server 来处理授权码（API 网关授权许可方式）。



（A）Client 使用浏览器（用户代理）访问 Authorization Server。
（B）Authorization Server 验证 Client 在（A）中传递的参数信息，如果无误则提供一个页面供 Resource Owner 登录。
（C）在（B）无误后返回一个授权码给 Client。
（D）Client 拿着（C）中获得的授权码和客户端标识、重定向 URL 等信息作为参数，请求 Authorization Server 提供的获取访问令牌的 URL。
（E）Authorization Server 返回访问令牌和可选的刷新令牌以及令牌有效时间等信息给 Client。

- Implicit：隐式许可省掉了授权码，直接返回 Token。适用于没有 Server 服务器来接受处理 Authorization Code 的第三方应用。
- Resource Owner Password Credentials：资源所有者密码凭据，直接使用 Owner 账号密码请求 Token。
- Client Credentials ：客户端凭据，客户端提供自己的身份参数直接获得 Token。

OIDC 说明

- OIDC（OpenID Connect）是在 OOAuth2.0 上构建的一个身份层，是一个基于 OAuth2.0 协议的身份认证标准协议。
- OAuth 2.0 定义了一些机制用于获取和使用访问令牌来访问受保护资源，但未定义用于提供标识信息的标准方法。
- OIDC 以 id_token 的形式提供有关最终用户的信息，可验证用户的标识，并提供有关用户的基本配置文件信息。

id_token

- OIDC 对 OAuth2.0 最主要的扩展就是提供了 ID Token。
- ID Token 是一个安全令牌，是一个授权服务器提供的包含用户信息（由一组 Cliams 构成以及其他辅助的 Cliams）的 JWT 格式的数据结构。
- JWT（JSON Web Token）：是一个定义一种紧凑的，自包含的并且提供防篡改机制的传递数据的方式的标准协议。**API 网关校验的是 JWT 标准生成 id_token。**

参数	英文全称	是否必选	说明	取值要求
iss	Issuer Identifier	是	提供认证信息者的唯一标识。	一般是一个 HTTPS 的 URL（不包含 querystring 和 fragment 部分）
sub	Subject Identifier	是	iss 提供的 EU 的标识，在 iss 范围内唯一。它会被 RP 用来标识唯一的用户。	最长为255个 ASCII 字符
aud	Audience(s)	是	标识 ID Token 的受众。	必须包含 OAuth2 的 client_id
exp	Expiration time	是	过期时间，超过此时间的 ID Token 会作废不再被验证通过。	-
iat	Issued At Time	是	JWT 构建的时间。	-
auth_time	AuthenticationTime	否	EU 完成认证的时间 。如果 RP 发送 AuthN 请求的时候携带 max_age 的参数，则此 Claim 是必须的。	-
nonce	-	否	RP 发送请求的时候提供的随机字符串，用来减缓重放攻击，也可以来关联 ID Token 和 RP 本身的 Session 信息。	-

参数	英文全称	是否必选	说明	取值要求
acr	Authentication Context Class Reference	否	表示一个认证上下文引用值，可以用来标识认证上下文类。	-
amr	Authentication Methods References	否	表示一组认证方法。	-
azp	Authorized party	否	结合 aud 使用。只有在被认证的一方和受众（sud）不一致时才使用此值，一般情况下很少使用。	-

API 网关 OAuth2.0 操作方法

API 网关已经对外提供了 OAuth 功能，实践一个 Auth 的 Demo 需要一个 Client 和 AS，可能需要 RS。
功能需求有两个：

- 授权 API 调用：请求 Token、刷新 Token
- 业务 API 调用（需要一个 RS Demo）直接用 Mock：Client 收发请求，postman 等模拟

AS 服务

使用 SpringBoot 快速开发
主要功能：处理授权 API 请求、生成 token_id、刷新 token_id

生成 RSA 公私钥

- 公钥：放在 API 网关，用于验证 JWT 签名。

- 私钥： AS 保存。

```
/**
 * RSA SHA256 签名，密钥位数2048
 */
public static String genereteRsaKey(){
    RsaJsonWebKey jwk = null;
    try{
        jwk = RsaJwkGenerator.generateJwk( bits: 2048);
    } catch (Exception e){
        e.printStackTrace();
        System.exit( status: 1);
    }
    jwk.setAlgorithm(AlgorithmIdentifiers.RSA_USING_SHA256);
    String privateKey = jwk.toJson(RsaJsonWebKey.OutputControlLevel.INCLUDE_PRIVATE);
    String publicKey = jwk.toJson(RsaJsonWebKey.OutputControlLevel.PUBLIC_ONLY);
    System.out.println("private key: "+privateKey);
    System.out.println("public key: "+publicKey);
    return privateKey;
}
```

JSON 格式输出：

```
private key: {"kty":"RSA","alg":"RS256","n":"ms5563cxLPU3BT6-cs1-NTSfxMtajanOPJw7ebYsjISCry6stfnHD7Ve9-z1lTJVoZuMYXmhA-j5MFUM2JDyLxJbkO
public key: {"kty":"RSA","alg":"RS256","n":"ms5563cxLPU3BT6-cs1-NTSfxMtajanOPJw7ebYsjISCry6stfnHD7Ve9-z1lTJVoZuMYXmhA-j5MFUM2JDyLxJbkOx
```

生成 id_token

```java
/**
 * 生成id_token时，需要用户设置OIDC协议定义JWT的payload中的
 * Claims属性(iss, aud, iat, exp, sub)。
 * 其中，iat跟exp必须设置。其他为非必须设置项
 *
 */

public static String generateToken(){
    PrivateKey privateKey = null;
    JwtClaims claims = new JwtClaims();
    claims.setIssuer("Issuer");
    claims.setAudience("Audience");
    claims.setExpirationTimeMinutesInTheFuture(10);
    claims.setIssuedAtToNow();
    claims.setSubject("Subject");
    try{
        privateKey = new RsaJsonWebKey(JsonUtil.parseJson(privateKeyJson)).getPrivateKey();
    } catch (Exception e){
        e.printStackTrace();
        System.exit( status: 2);
    }
    JsonWebSignature jws = new JsonWebSignature();
    jws.setPayload(claims.toJson());
    jws.setKey(privateKey);
    jws.setAlgorithmHeaderValue(AlgorithmIdentifiers.RSA_USING_SHA256);
    String jwt = null;
    try{
        jwt = jws.getCompactSerialization();
        System.out.println("JWT: "+jwt);
    } catch (Exception e){
        e.printStackTrace();
        System.exit( status: 3);
    }
    return jwt;
}
```

- 生成 Token 时，需要用户设置 OIDC 协议定义 JWT 的 payload 中的 Claims 属性(iss, aud, iat, exp, sub)。其中，iat 和 exp 必须设置，其他为非必须设置项。

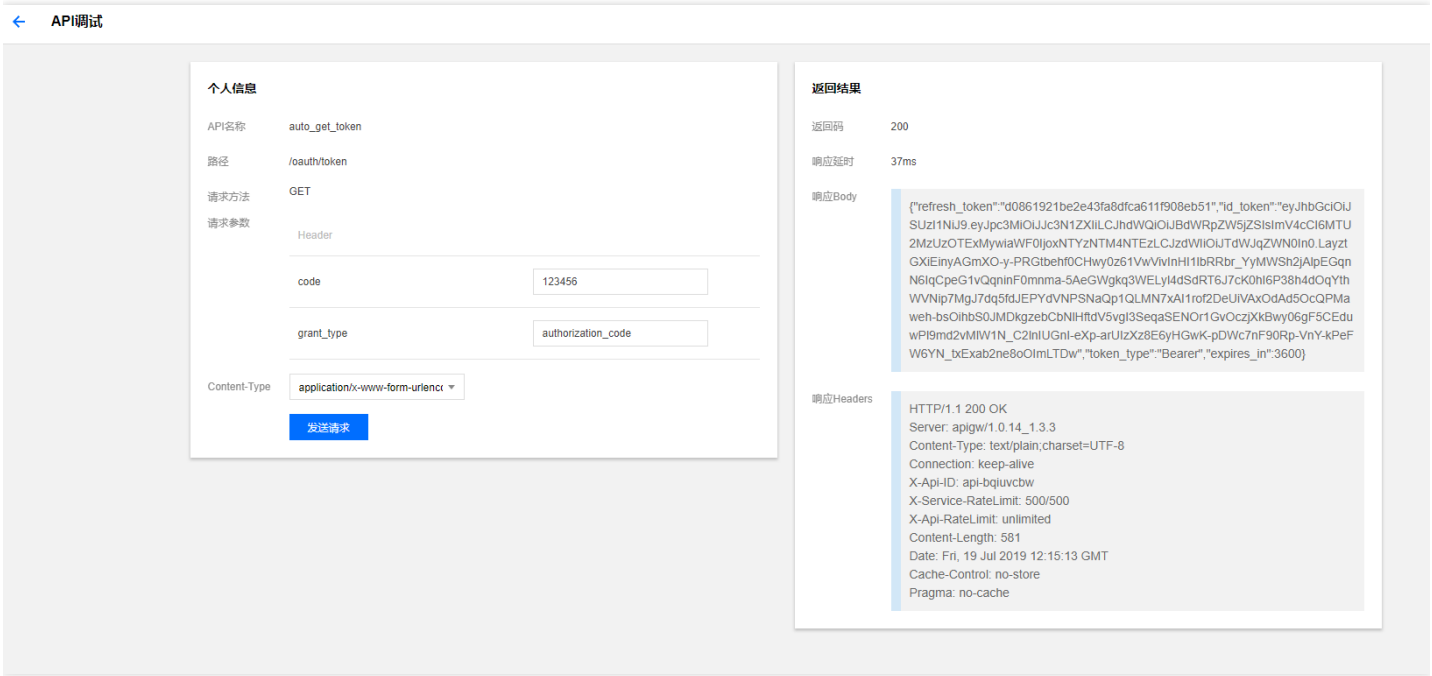
处理请求 Token

```
@RestController
public class AuthController {
    private static final String c_id = "qcloud";
    private static final String c_secret = "qcloud";
    @RequestMapping(value = "/oauth/token", method = RequestMethod.GET)
    public Object getToken(@RequestParam(value = "code") String code,
                           @RequestParam(value = "redirect_uri", required = false) String uri,
                           @RequestParam(value = "OAuth_Client_ID") String client_id,
                           @RequestParam(value = "OAuth_Client_Secret") String client_secret){
        if(!code.equals("qcloud") || !client_id.equals(c_id) || !client_secret.equals(c_secret))
            return "error param!!";
        String refresh_token = UUIDGenerator.uuidStr();
        Map<String,Object> respond = new HashMap<>();
        respond.put("refresh_token",refresh_token);
        respond.put("id_token", JWTManager.generateToken());
        return JsonUtil.toJson(respond);
    }
```

刷新 id_token

```
@RequestMapping(value = "/oauth/refresh", method = RequestMethod.GET)
public Object refreshToken(@RequestParam(value = "refresh_token") String refresh,
                           @RequestParam(value = "OAuth_Client_ID") String client_id,
                           @RequestParam(value = "OAuth_Client_Secret") String client_secret){
    if(!refresh.equals(refresh_token) || !client_id.equals(c_id) || !client_secret.equals(c_secret))
        return "error param!!";
    Map<String, Object> res = new HashMap<>();
    refresh_token = UUIDGenerator.uuidStr();
    res.put("refresh_token",refresh_token);
    res.put("id_token", JWTManager.generateToken());
    return JsonUtil.toJson(res);
}
```

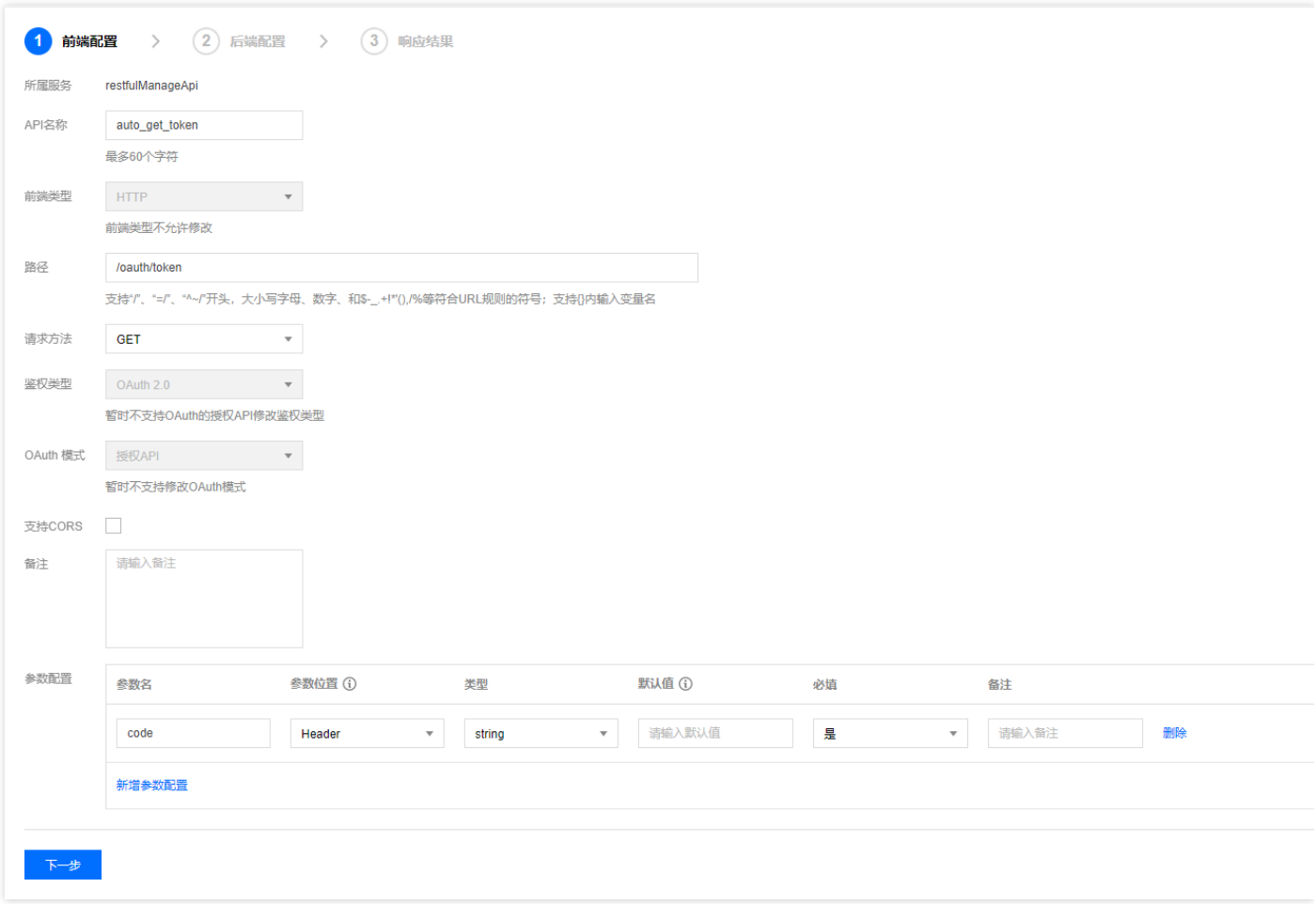
成功授权显示如下：



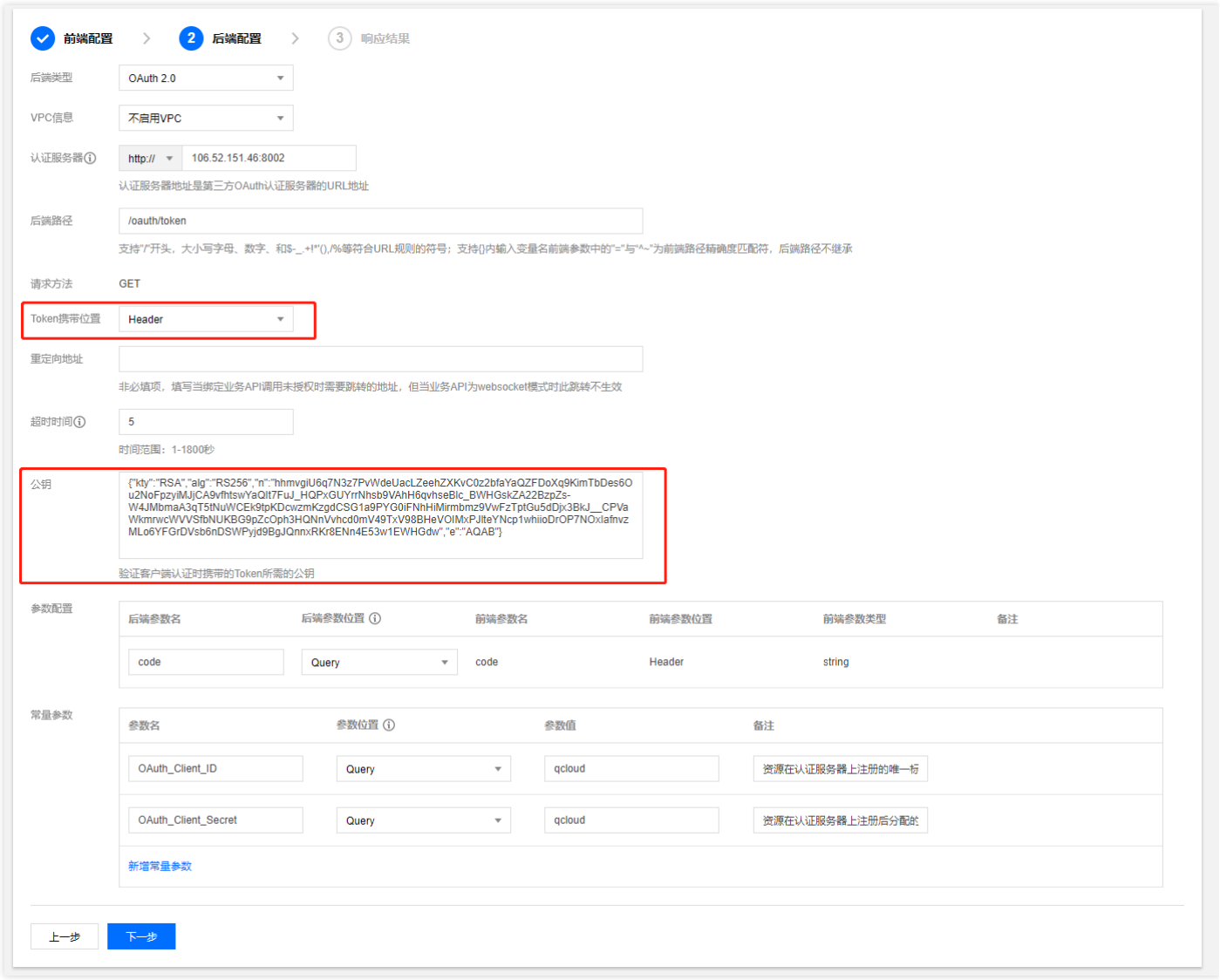
演示

先将 AS 服务部署到后台服务器并跑起来，注意如果是部署到 CVM，需要在安全组中开放端口。

- 在 API 管理平台创建授权 API

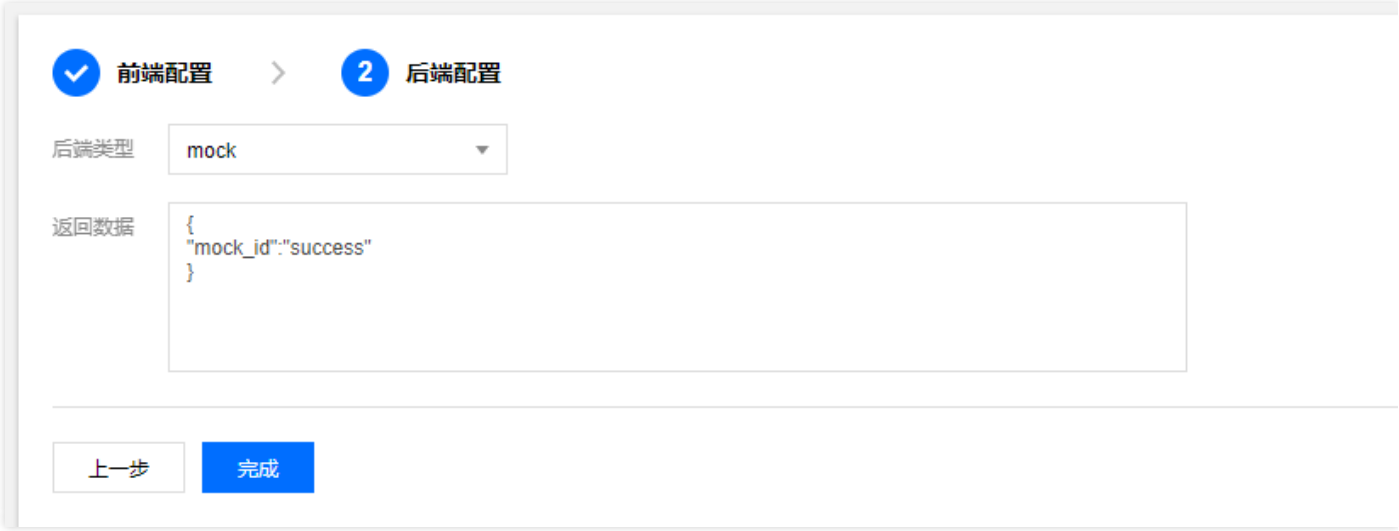


- 填入公钥等 OAuth 信息



- 在 API 网关平台创建业务 API，关联授权 API

- 发布 API



- 调用授权 API 拿到 id_token

- 使用 id_token 访问业务 API

CAM 访问管理

最近更新时间：2019-09-16 14:32:52

CAM 简介

注意：
API 网关于2018年9月20日晚-21日对接 CAM，为避免影响现有客户所有账户的使用，我们会默认对20日前的 API 网关用户的账户子账户均授予 QcloudAPIGWFullAccess 权限，这意味着您的子账户跨账户等均拥有对 API 网关现有服务及 API 的完全管理权限，此授权不影响其他产品的权限，若您希望更精确管理账户的权限策略，则可参考下面的示例进行更改。

基本概念

根账户通过给子账户绑定策略实现授权，策略设置可精确到 **[API，资源，用户/用户组，允许/拒绝，条件]** 维度。

账户

- **根账号**：腾讯云资源归属、资源使用计量计费的基本主体，可登录腾讯云服务。
- **子账号**：由根账号创建账号，有确定的身份 ID 和身份凭证，且能登录到腾讯云控制台。根账号可以创建多个子账号(用户)。**子账号默认不拥有资源，必须由所属根账号进行授权。**
- **身份凭证**：包括登录凭证和访问证书两种，**登录凭证** 指用户登录名和密码，**访问证书** 指云 API 密钥（SecretId 和 SecretKey）。

资源与权限

- **资源**：资源是云服务中被操作的对象，如一个云服务器实例、COS 存储桶、VPC 实例等。
- **权限**：权限是指允许或拒绝某些用户执行某些操作。默认情况下，**根账号拥有其名下所有资源的访问权限**，而**子账号没有根账号下任何资源的访问权限。**
- **策略**：策略是定义和描述一条或多条权限的语法规范。**根账号**通过将**策略关联**到用户/用户组完成授权。

相关文档

阅读目的	链接
了解策略和用户之间关系	策略管理
了解策略的基本结构	策略语法
了解还有哪些产品支持 CAM	支持 CAM 的产品

单击查看更多 CAM 文档 >>

API 网关资源

- qcs::APIgateway:_`region`_:uin/_`uin—id`_:service/_`serviceid`_
- qcs::APIgateway:_`region`_:uin/_`uin—id`_:service/_`serviceid`_/API/_`apiid`_
- qcs::APIgateway:_`region`_:uin/_`uin—id`_:usagePlan/_`usagePlanid`_
- qcs::APIgateway:_`region`_:uin/_`uin—id`_:secret/_`secretid`_
- qcs::APIgateway:_`region`_:uin/_`uin—id`_:IPStrategy/_`IPStrategyId`_
- qcs::APIgateway:_`region`_:uin/_`uin—id`_:logRule/_`logRuleId`_

所有的创建接口为账户级别，而其他接口均为资源级别。

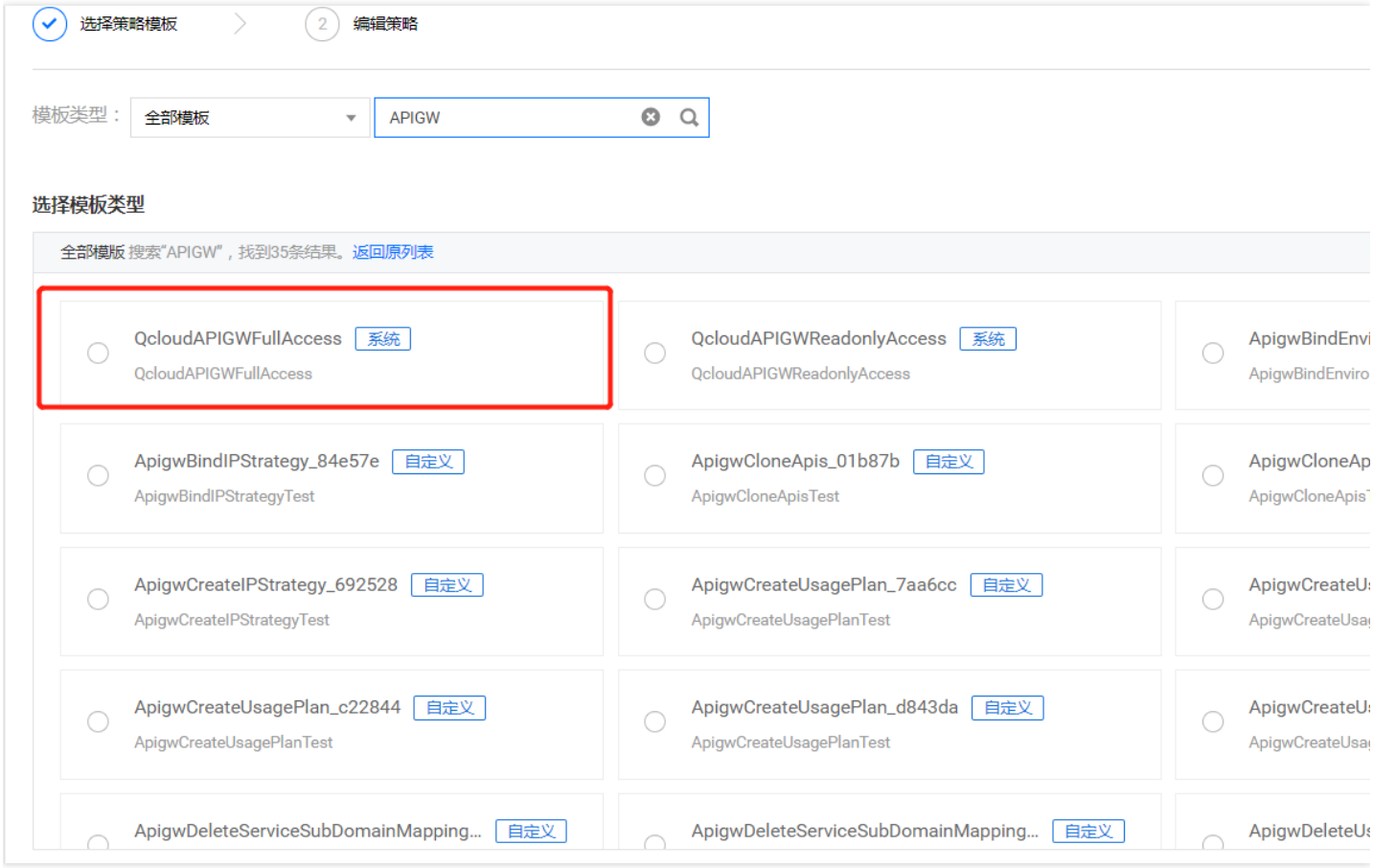
访问控制策略示例

API 网关全读写策略

授权一个子用户以 API 服务的完全管理权限（创建、管理等全部操作）。

```
{
"version": "2.0",
"statement": [
{
"action": [
"apigw:*"
],
"resource": "*",
"effect": "allow"
}
]
}
```

也可以通过设置系统的 全读写策略 支持。



API 网关单个服务完全管理策略

授权一个子用户某个特定 API 服务的完全管理权限（创建、管理等全部操作）：

```
{
"version": "2.0",
"statement": [
{
"action": [
"apigw:*"
],
"resource":"qcs::apigw:ap-guangzhou:uin/{ownerUin}:service/service-id/API/api-id",
"effect": "allow"
}
]
}
```

API 网关单个服务只读策略

1. 按照策略生成器创建，授权列表类权限和产品监控权限，这里是对账户下的所有资源授予只读权限。

```
{
"version": "2.0",
"statement": [
{
"action": [
"apigw:Describe*",
"apigw:GenerateApiDocument"
],
"resource": "*",
"effect": "allow"
}
]
}
```

2. 授权单 API 只读权限。

```
{
"version": "2.0",
"statement": [
{
"action": [
"ckafka:Get*",
"ckafka:List*"
],
"resource": "qcs::apigw:ap-guangzhou:uin/{ownerUin}:service/service-id/API/api-id",
"effect": "allow"
}
]
}
```

版权所有：腾讯云计算（北京）有限责任公司