# **参考**

<http://www.ruanyifeng.com/blog/2019/04/oauth_design.html>

# 介绍

简单说，OAuth 就是一种授权机制。数据的所有者告诉系统，同意授权第三方应用进入系统，获取这些数据。系统从而产生一个短期的进入令牌（token），用来代替密码，供第三方应用使用。

# **令牌与密码**

令牌（token）与密码（password）的作用是一样的，都可以进入系统，但是有三点差异。

（1）令牌是短期的，到期会自动失效，用户自己无法修改。密码一般长期有效，用户不修改，就不会发生变化。

（2）令牌可以被数据所有者撤销，会立即失效。以上例而言，屋主可以随时取消快递员的令牌。密码一般不允许被他人撤销。

（3）令牌有权限范围（scope），比如只能进小区的二号门。对于网络服务来说，只读令牌就比读写令牌更安全。密码一般是完整权限。

上面这些设计，保证了令牌既可以让第三方应用获得权限，同时又随时可控，不会危及系统安全。这就是 OAuth 2.0 的优点。

注意，只要知道了令牌，就能进入系统。系统一般不会再次确认身份，所以令牌必须保密，泄漏令牌与泄漏密码的后果是一样的。 这也是为什么令牌的有效期，一般都设置得很短的原因。

OAuth 2.0 对于如何颁发令牌的细节，规定得非常详细。具体来说，一共分成四种授权类型（authorization grant），即四种颁发令牌的方式，适用于不同的互联网场景。

# **授权**

OAuth 的核心就是向第三方应用颁发令牌。也就是说，OAuth 2.0 规定了四种获得令牌的流程。你可以选择最适合自己的那一种，向第三方应用颁发令牌。下面就是这四种授权方式。

授权码（authorization-code）

隐藏式（implicit）

密码式（password）：

客户端凭证（client credentials）

注意，不管哪一种授权方式，第三方应用申请令牌之前，都必须先到系统备案，说明自己的身份，然后会拿到两个身份识别码：客户端 ID（client ID）和客户端密钥（client secret）。这是为了防止令牌被滥用，没有备案过的第三方应用，是不会拿到令牌的。

## 第一种授权方式：授权码（authorization code）

指的是第三方应用先申请一个授权码，然后再用该码获取令牌。这种方式是最常用的流程，安全性也最高，它适用于那些有后端的 Web 应用。授权码通过前端传送，令牌则是储存在后端，而且所有与资源服务器的通信都在后端完成。这样的前后端分离，可以避免令牌泄漏。

### 第一步：客户端向资源端申请code

A 网站提供一个链接，用户点击后就会跳转到 B 网站，授权用户数据给 A 网站使用。下面就是 A 网站跳转 B 网站的一个示意链接。

[http://localhost:8846/uaa/oauth/authorize?response\_type=code&client\_id=1552294656514&redirect\_uri=http://localhost:7070/callback&scope=read\_userinfo](http://localhost:8846/uaa/oauth/authorize?response_type=code&client_id=1552294656514&redirect_uri=http:/localhost:7070/callback&scope=read_userinfo)

http://localhost:8846/uaa/oauth/authorize?response\_type=code&client\_id=1552294656514&redirect\_uri=http://www.baidu.com&scope=read\_userinfo

固定参数：Responset\_type=code

配置在表oauth\_client\_details表中：Client\_id、Redirect\_uri、scope

### 第二步：资源端返回code给客户端

http://localhost:7070/callback?code=G0tgFx

资源端验证了client\_id\scope\redirect\_uri，是在DB中预先配置的后返回code

### 第三步：客户端根据code向资源端请求令牌

[http://localhost:8846/uaa/oauth/token?client\_id=1552294656514&client\_secret=123456&grant\_type=authorization\_code&code=V1KUMt&scope=read\_userinfo&redirect\_uri=http://localhost:7070/test](http://localhost:8846/uaa/oauth/token?client_id=1552294656514&client_secret=123456&grant_type=authorization_code&code=V1KUMt&scope=read_userinfo&redirect_uri=http:/localhost:7070/test)

http://localhost:8846/uaa/oauth/token?client\_id=1552294656514&client\_secret=123456&grant\_type=authorization\_code&code=W9ycil&scope=read\_userinfo&redirect\_uri=http://localhost:7070/test

可能有点用

很多人躺在这里

<https://mp.weixin.qq.com/s/b1PZl8OVj3m7Q5IaJR_dpQ>

<https://www.oschina.net/question/124056_2200918?sort=time>

遇到的坑

BCryptPasswordEncoder 注意不同的版本，加密后是否需要{bcrypt}，加密方法同一个串每次加密的值是不同的，需要用match方法判断；

There is no id错误，百度大多是在说密码加密的问题引发的，但采用了将用户写在内存中问题依然存在。

token的存储最初用redis，用户和clients初始化使用的DB，最初怀疑过redis到底有没有缓存，没有验证，最终采用的是db 存token。

### 第四步：资源端向客户端返回令牌及明细

资源端验证了客户端的client\_id\client\_secret\grant\_type\code\scope\redirect\_uri，返回用户json

{

"access\_token": "eyJhbGciOiJ.......Ul0pEUgdYpA4",

"token\_type": "bearer",

"expires\_in": 43199,

"scope": "read\_userinfo",

"userName": "系统管理员\_张宇",

"roles": [

{

"rid": "1",

"roleName": "管理员角色",

"roleCode": "ROLE\_ADMIN",

"authority": "ROLE\_ADMIN"

}

],

"jti": "024062c6-879d-4a9f-9de0-5a01209b3ee8"

}

## 第二种授权方式：隐藏式（implicit）

有些 Web 应用是纯前端应用，没有后端。这时就不能用上面的方式了，必须将令牌储存在前端。RFC 6749 就规定了第二种方式，允许直接向前端颁发令牌。这种方式没有授权码这个中间步骤，所以称为（授权码）"隐藏式"（implicit）。

### 第一步：客户端向资源端申请token

A 网站提供一个链接，要求用户跳转到 B 网站，授权用户数据给 A 网站使用。

http://localhost:8846/uaa/oauth/authorize?client\_id=1552294656514&response\_type=token&scope=read\_userinfo&redirect\_uri=http://www.baidu.com

### 第二步：资源端转向客户端并传递token

https://www.baidu.com/#access\_token=a\*\*\*\*&token\_type=bearer&expires\_in=39356&userName=%E7%B3%BB%E7%BB%9F%E7%AE%A1%E7%90%86%E5%91%98\_%E5%BC%A0%E5%AE%87&roles=%5Bcn.itcraftsman.luban.platform.kernel.oauth2.domain.Oauth2Role@67e593b3,%20cn.itcraftsman.luban.platform.kernel.oauth2.domain.Oauth2Resource@4d97c33b,%20cn.itcraftsman.luban.platform.kernel.oauth2.domain.Oauth2Resource@3aa104c4%5D&jti=024062c6-879d-4a9f-9de0-5a01209b3ee8

上面 URL 中，access\_token参数就是令牌，A 网站因此直接在前端拿到令牌。

注意，令牌的位置是 URL 锚点（fragment），而不是查询字符串（querystring），这是因为 OAuth 2.0 允许跳转网址是 HTTP 协议，因此存在"中间人攻击"的风险，而浏览器跳转时，锚点不会发到服务器，就减少了泄漏令牌的风险。这种方式把令牌直接传给前端，是很不安全的。因此，只能用于一些安全要求不高的场景，并且令牌的有效期必须非常短，通常就是会话期间（session）有效，浏览器关掉，令牌就失效了。

## 第三种授权方式：密码式(password)

如果你高度信任某个应用，RFC 6749 也允许用户把用户名和密码，直接告诉该应用。该应用就使用你的密码，申请令牌，这种方式称为"密码式"（password）。

### 第一步：客户端向资源端申请token

A 网站要求用户提供 B 网站的用户名和密码。拿到以后，A 就直接向 B 请求令牌。

http://localhost:8846/uaa/oauth/token?username=symenzhang@163.com&password=123456&grant\_type=password&client\_id=1552294656514&client\_secret=123456

### 第二步：资源端转向客户端并传递token

https://www.baidu.com/#access\_token=a\*\*\*\*&token\_type=bearer&expires\_in=39356&userName=%E7%B3%BB%E7%BB%9F%E7%AE%A1%E7%90%86%E5%91%98\_%E5%BC%A0%E5%AE%87&roles=%5Bcn.itcraftsman.luban.platform.kernel.oauth2.domain.Oauth2Role@67e593b3,%20cn.itcraftsman.luban.platform.kernel.oauth2.domain.Oauth2Resource@4d97c33b,%20cn.itcraftsman.luban.platform.kernel.oauth2.domain.Oauth2Resource@3aa104c4%5D&jti=024062c6-879d-4a9f-9de0-5a01209b3ee8

B 网站验证身份通过后，直接给出令牌。注意，这时不需要跳转，而是把令牌放在 JSON 数据里面，作为 HTTP 回应，A 因此拿到令牌。这种方式需要用户给出自己的用户名/密码，显然风险很大，因此只适用于其他授权方式都无法采用的情况，而且必须是用户高度信任的应用。

## 第四种授权方式：凭证式(client\_credentials)

这种模式直接根据client的id和密钥即可获取token，无需用户参与

这种模式比较合适消费api的后端服务，比如拉取一组用户信息等

不支持refresh token，主要是没有必要。

### 第一步：客户端向资源端申请token

A 网站要求用户提供 B 网站的用户名和密码。拿到以后，A 就直接向 B 请求令牌。

http://localhost:8846/uaa/oauth/token?grant\_type=client\_credentials&client\_id=1552294656514&client\_secret=123456

### 第二步：资源端转向客户端并传递token

{

"access\_token": "......",

"token\_type": "bearer",

"expires\_in": 43193,

"scope": "read\_userinfo",

"userName": "no\_user",

"roles": "no\_roles",

"jti": "b00ddead-93f6-47e4-a01a-ffc20fe874fa"

}

# **对比表**

|  |  |  |  |
| --- | --- | --- | --- |
| 密码模式 | 授权码模式 | 简化模式 | 客户端模式 |
| 支持refresh token | 支持refresh token | 不支持refresh token | 不支持refresh token |
| 不安全，开放了用户和密码 | 标准应用，比较烦索，但很安全 | 这种模式比授权码模式少了code环节，回调url直接携带token。这种模式的使用场景是基于浏览器的应用。这种模式基于安全性考虑，建议把token时效设置短一些 | 这种模式直接根据client的id和密钥即可获取token，无需用户参与；这种模式比较合适消费api的后端服务，比如拉取一组用户信息等。 |
| Authorized\_Grant\_types  password,refresh\_token | Authorized\_Grant\_types  authorization\_code,refresh\_token | Authorized\_Grant\_types  implicit | Authorized\_Grant\_types  client\_credentials |
|  |  |  |  |