# **OAuth 2 详解**

## **简介**

[OAuth 2](https://link.zhihu.com/?target=https://tools.ietf.org/html/rfc6749" \t "https://zhuanlan.zhihu.com/p/_blank) 是一种授权框架，允许第三方应用通过用户授权的形式访问服务中的用户信息，最常见的场景是授权登录；再复杂一点的比如第三方应用通过 Github 给开发者提供的接口访问权限内的用户信息或仓库信息。OAuth2 广泛应用于 web 、桌面应用、移动 APP 的第三方服务提供了授权验证机制，以此实现不同应用间的数据访问权限。 下面分别从不同角色、授权类型、使用场景及流程的纬度详细介绍 OAuth2

## **OAuth Roles**

OAuth 定义了四种角色

* 资源拥有者 (Resource Owner)
* 客户端 (Client)
* 资源服务器 (Resource Server)
* 授权服务器 (Authorization Server)

资源拥有者其实就是真实的用户，用户授权给第三方应用访问在其他系统的用户信息。第三方应用访问授权用户的信息范围 scope 属于申请接入服务时选择的权限之内(例如：读或写访问权限)

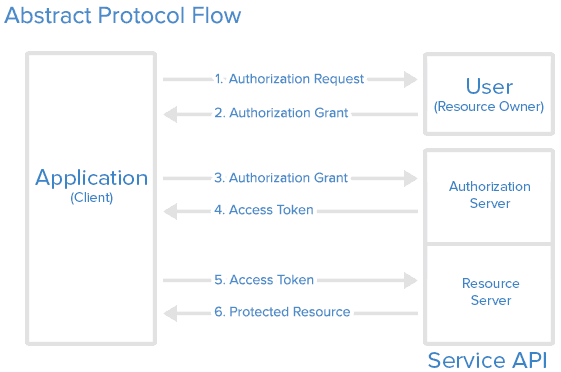
资源服务控制用户的信息，授权服务验证用户提供的信息是否正确并返回 access token 给第三方应用。 站在第三方开发者的角度看，被接入的系统提供的服务 API 同时实现了资源和授权角色。在这里把资源服务端和授权服务端统一为“服务角色或 API 角色”。

客户端就是要求接入的第三方应用，获取用户在提供服务的系统的账户信息。对于客户端而言，最终获取到用户在服务端的账户信息首先需要用户授权，用户授权后传给提供服务端验证成功之后返回 access token ，在通过 access token 请求提供服务的系统（在这里我们成为 API ，下文也是）获取用户在 API 中的账户信息。

## **授权流程**

接下来文中提到的客户端均为第三方应用，服务端均为被接入的服务。譬如拉勾网有微博登录功能，那么拉勾网就是客户端，微博就是服务端。

上一节解释了 OAuth 系统中角色的概念，接下来可以解释整个授权的流程。



流程图解释：

1. 用户点击客户端提供的授权请求
2. 客户端请求服务的授权页面呈现给用户，用户点击确认授权后服务端返回授权许可凭证给客户端
3. 客户端通过步骤二接收到的授权许可凭证及在服务端注册的应用信息请求服务端
4. 如果步骤三验证通过服务端则返回 access token 给客户端
5. 客户端通过第四步获取的 access token 请求服务端获取资源
6. 如果服务端校验 access token 成功，则返回指定资源给客户端

以上步骤是 OAuth2 授权登录的步骤，当然不同类型的授权许步骤会不一样，下文会详细逐一讨论。

## **第三方应用创建**

第三方接入某平台的 OAuth 2 服务之前，通常需要在平台提供的注册网站（通常叫做 developer" or "API"）新建一个应用，注册成功之后平台会生成应用的 ID 和密码。 读者可以访问 Github 的开发中心看看，国内写的规范的有 [coding developer](https://link.zhihu.com/?target=https://open.coding.net/references/oauth/" \t "https://zhuanlan.zhihu.com/p/_blank)

在平台的注册页面至少得提供如下信息

* 应用名称
* 应用网址
* 回调链接

回调域名的作用：用户点击同意按钮之后，服务端向客户端返回授权码或 access token ,当然如果是禁止操作也需回调告知客户端结果。

在 OAuth 2 服务提供平台新建一个应用之后，平台会提供一对客户端 ID 和密码作为客户端凭证。客户端 ID 是可以公开的字符串，用以构建授权请求链接；用户授权之后，客户端使用服务端的授权码和平台分配的密码获取 access token 。密码应当妥善保管以免泄漏。

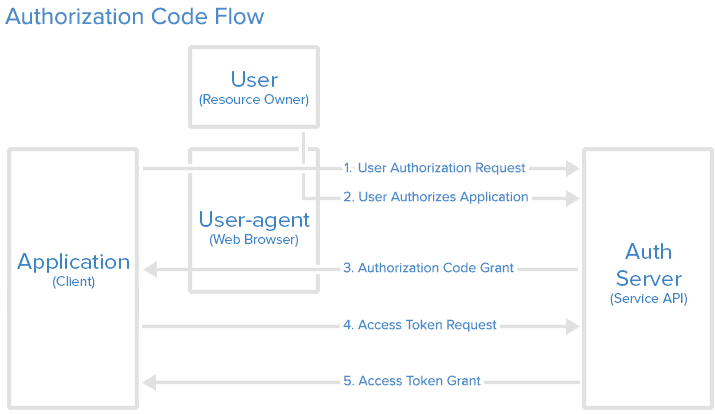
## **授权许可**

在第一张授权流程图中，前四步包含了获取授权许可获取 access token ,授权许可有四种类型，服务端返回哪种类型取决于客户端在请求链接中构建的 response\_type 参数。四种类型的授权许可有不同的应用场景：

* **授权码**：通过授权码，服务端通过 Ajax 把 access token 返回给客户端
* **隐式**：用于移动 APP 或 web 应用（应用运行在用户的设备上）
* **用户密码凭证**：同一个公司不同系统之间内部账户互联互通，比如国内某社区的代码托管系统通过社区的账户也可以登录。
* **客户端凭证**：第三方应用自身服务访问提供 OAuth2 服务提供的平台资源

## **授权码 (Grant Type: Authorization Code)**

授权码是 OAuth2 授权最广泛的方式，得益于平台给第三方应用分配的 ID 和密码都是隐藏在第三方应用的后端代码中。因为授权码是基于重定向的方式，要想使用授权码的方式，第三方应用必须能够调用用户系统中的应用（譬如浏览器）、和提供一个接口接收平台服务的回调获取授权码。



一步步解释授权码的流程图

## **步骤一：构建获取授权码请求链接**

下文提到的例子都是基于 digiterocean 的 OAuth2

第一步是构建一个请求 Auth Server 获取授权码的链接，向服务端发起请求

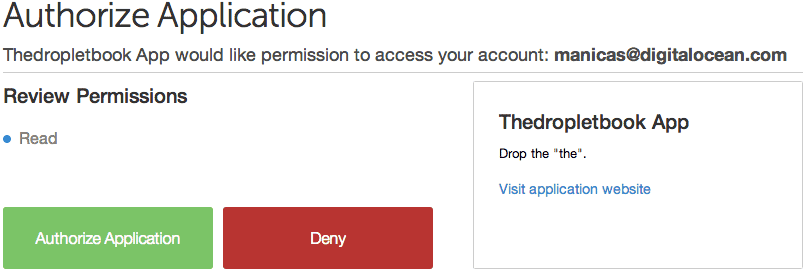
https://cloud.digitalocean.com/v1/oauth/authorize?response\_type=code&client\_id=CLIENT\_ID&redirect\_uri=CALLBACK\_URL&scope=read

分解请求链接：

* **[https://cloud.digitalocean.com/v1/oauth/authorize](https://link.zhihu.com/?target=https://cloud.digitalocean.com/v1/oauth/authorize" \t "https://zhuanlan.zhihu.com/p/_blank)** 表示服务端授权 endpoint
* **client\_id** 平台分配给第三方应用的 ID
* **redirect\_uri=CALLBACK\_URL** 第三方开发者在平台新建第三方应用时填写的回调 URL
* **response\_type=code** 指定服务端返回授权码
* **scope=read** 指定第三方应用请求权限类型

## **步骤二：用户授权给第三方应用**

当用户点击第一步构建的链接之后，在用户已经登录服务端之后才能点击确认授权（譬如想通过微信登录某个第三方网址，你必须首先已经登录微博）。这是平台会给用户提供一个页面给用户确认是否授权。



## **步骤三：服务端给客户端返回授权码**

当用户点击同意授权之后，服务端发起一个请求重定向到第三方平台填写的回调链接，且请求链接中同时包含了服务端生成的授权码。

https://dropletbook.com/callback?code=AUTHORIZATION\_CODE

## **步骤四：第三方应用请求服务端，获取 access token**

客户端获取到服务端返回的授权码之后，接着使用授权码和平台分配的密码请求服务端获取服务端的 access token 。第三方应用后端代码拼接的 URL 格式形似：

https://cloud.digitalocean.com/v1/oauth/token?client\_id=CLIENT\_ID&client\_secret=CLIENT\_SECRET&grant\_type=authorization\_code&code=AUTHORIZATION\_CODE&redirect\_uri=CALLBACK\_URL

在这里需注意两个参数，其一是 client\_secret 没有暴露出去，在第三方应用后端代码中拼接请求链接；另一是 redirect\_uri ，这里不是在平台新建应用时填写的回调链接，而是第三方应用已经实现的另一个 action 处理服务端返回 access token 的请求。

## **步骤五：第三方应用接收 access token**

如果第四部客户端发送的信息被服务端校验成功，服务端则返回 access token ，有一些平台同时也同时传递 refresh token 。返回的 json 信息形似：

{

"access\_token":"ACCESS\_TOKEN",

"token\_type":"bearer",

"expires\_in":2592000,

"refresh\_token":"REFRESH\_TOKEN",

"scope":"read",

"uid":100101,

"info":

{

"name":"Mark E. Mark",

"email":"mark@thefunkybunch.com"

}

}

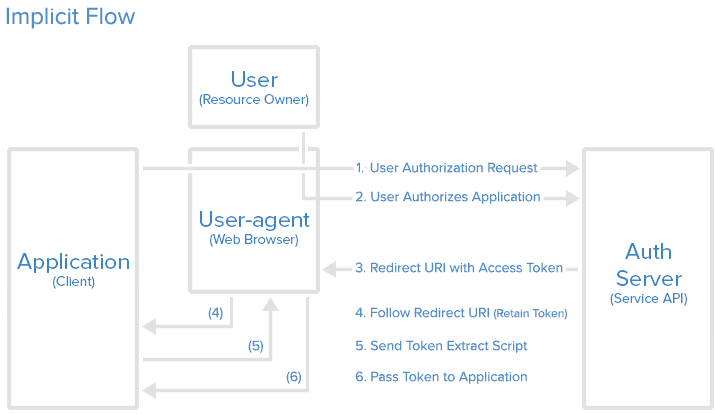
至此，已经走完 oauth 2 授权码的方式所有流程。在 access\_token 没有失效的前提下，可通过 access\_token 可以访问平台服务端提供的资源。另外如果还提供了 refresh\_token ，在 access\_token 失效的情况下，可以通过 refresh\_token 再次去获取有效的 access\_token 。

## **隐式 (Grant Type: Implicit)**

相比授权码的授权许可这种后端实现，隐式授权类型在应用于前端实现（移动客户端或者浏览器），其缺陷就是并不保证平台分配给第三方平台的密码凭证足够安全。隐式授权同样也是基于重定向的方式，但是 **access\_token 通过网页重定向返回给第三方平台而不是通过接口调用的方式** 暴露了 access\_token 在重定向链接中，这点和授权码不同。另外，也不支持服务端校验客户端第三方应用密码，只是依赖在平台新建应用是填写的回调链接。

隐式授权不支持 refresh\_token

授权大概流程如下：用户请求授权，同意授权后服务端把 access\_token 拼接到回调链接上通过浏览器重定向到客户端，然后客户端获取到 access\_token 。



通过流程图可看出在第三步和授权码不用的是服务端重定向到网页而不是通过接口把 access\_token 返回给客户端。

## **步骤一：构建隐式授权请求链接**

与授权码的步骤一差不多一致，只是 response\_type 字段的参数值为 token

## **步骤二：用户授权给第三方**

与授权码授权的步骤二完全一致

## **步骤三：通过服务端重定向链接，User-agent (浏览器或者 APP) 获取 access\_token**

与授权码不同的是，用户点击授权之后服务端通过把 access\_token 通过网页重定向到回调链接，客户端通过重定向链接才能获取到 access\_token ；而授权码获取 access\_token 是通过服务端通过 ajax 请求的形式直接访问第三方应用的后端接口，把 access\_token 传给第三方后端。

## **第四步：前端根据重定向调整**

前端浏览器重定向请求第三方平台的后端

## **第五步：第三方应用后端通过脚本获取在重定向链接中的 access\_token**

第三方应用后端通过脚本获取在重定向链接上的 access\_token

## **第六步：User-Agent 执行第三方应用后端返回的脚本把 access\_token 返回给第三方平台后端。**

## **用户密码凭证**

用户直接给第三方应用提供在提供服务端账号密码，获取服务端的 access\_token 。这种授权方式常用于一个企业不同服务之间的账号互联互通。 用户给第三方应用提供账号密码之后，客户端发送 POST 请求给服务端获取 access\_token

https://oauth.example.com/token?grant\_type=password&username=USERNAME&password=PASSWORD&client\_id=CLIENT\_ID

如果服务端校验客户端（第三方应用）传过来的账户密码正确，则把 access\_token 返回给客户端，用户授权完成。

## **客户端凭证**

这种授权方式常用于第三方应用想要更改自身在服务提供方注册的应用信息，比如更改应用描述或回调链接地址。

第三方应用通过发送服务端分配的 ID 和密码给后端校验，POST 的 URL 格式形似：

https://oauth.example.com/token?grant\_type=client\_credentials&client\_id=CLIENT\_ID&client\_secret=CLIENT\_SECRET

## **Access Token 和 Refresh Token**

第三方应用从服务提供平台获取到有效的 access\_token 之后，即可根据平台提供的接口访问服务端的资源。

curl -X POST -H "Authorization: Bearer ACCESS\_TOKEN""https://api.digitalocean.com/v2/$OBJECT"

如果服务提供平台支持 refresh\_token ，那么第三方应用的 access\_token 失效之后，可通过 refresh\_token 再次获取有效的 access\_token

例如 digitalocean 支持第三方应用通过 POST 请求再此获取有效的 access\_token

https://cloud.digitalocean.com/v1/oauth/token?grant\_type=refresh\_token&client\_id=CLIENT\_ID&client\_secret=CLIENT\_SECRET&refresh\_token=REFRESH\_TOKEN