06 | 除了授权码许可类型，OAuth 2.0还支持什么授权流程？

2020-07-11 王新栋

OAuth 2.0实战课 进入课程 >

讲述：李海明
时长 13:24 大小 12.28M

你好，我是王新栋。

在前面几讲学习授权码许可类型的原理与工作流程时，不知道你是不是一直有这样一个疑问：授权码许可的流程最完备、最安全没错儿，但它适合所有的授权场景吗？在有些场景下使用授权码许可授权，是不是过于复杂了，是不是根本就没必要这样？

比如，小兔打单软件是京东官方开发的一款软件，那么小明在使用小兔的时候，还需要小兔再走一遍授权码许可类型的流程吗？估计你也猜到答案了，肯定是不需要了。

你还记得授权码许可流程的特点么？它通过授权码这种临时的中间值，让小明这样的用户参与进来，从而让小兔软件和京东之间建立联系，进而让小兔代表小明去访问他在京东店铺的订单数据。

现在小兔被“招安”了，是京东自家的了，是被京东充分信任的，没有“第三方软件”的概念了。同时，小明也是京东店铺的商家，也就是说软件和用户都是京东的资产。这时，显然没有必要再使用授权码许可类型进行授权了。但是呢，小兔依然要通过互联网访问订单数据的 Web API，来提供为小明打单的功能。

于是，为了保护这些场景下的 Web API，又为了让 OAuth 2.0 更好地适应现实世界的更多场景，来解决比如上述小兔软件这样的案例，OAuth 2.0 体系中还提供了资源拥有者凭据许可类型。

资源拥有者凭据许可

从“资源拥有者凭据许可”这个命名上，你可能就已经理解它的含义了。没错，资源拥有者的凭据，就是用户的凭据，就是用户名和密码。可见，这是最糟糕的一种方式。那为什么 OAuth 2.0 还支持这种许可类型，而且编入了 OAuth 2.0 的规范呢？

我们先来思考一下。正如上面我提到的，小兔此时就是京东官方出品的一款软件，小明也是京东的用户，那么小明其实是可以使用用户名和密码来直接使用小兔这款软件的。原因很简单，那就是这里不再有“第三方”的概念了。

但是呢，如果每次小兔都是拿着小明的用户名和密码来通过调用 Web API 的方式，来访问小明店铺的订单数据，甚至还有商品信息等，在调用这么多 API 的情况下，无疑增加了用户名和密码等敏感信息的攻击面。

如果是使用了 token 来代替这些“满天飞”的敏感信息，不就能很大程度上保护敏感信息数据了吗？这样，小兔软件只需要使用一次用户名和密码数据来换回一个 token，进而通过 token 来访问小明店铺的数据，以后就不会再使用用户名和密码了。

接下来，我们一起看下这种许可类型的流程，如下图所示：

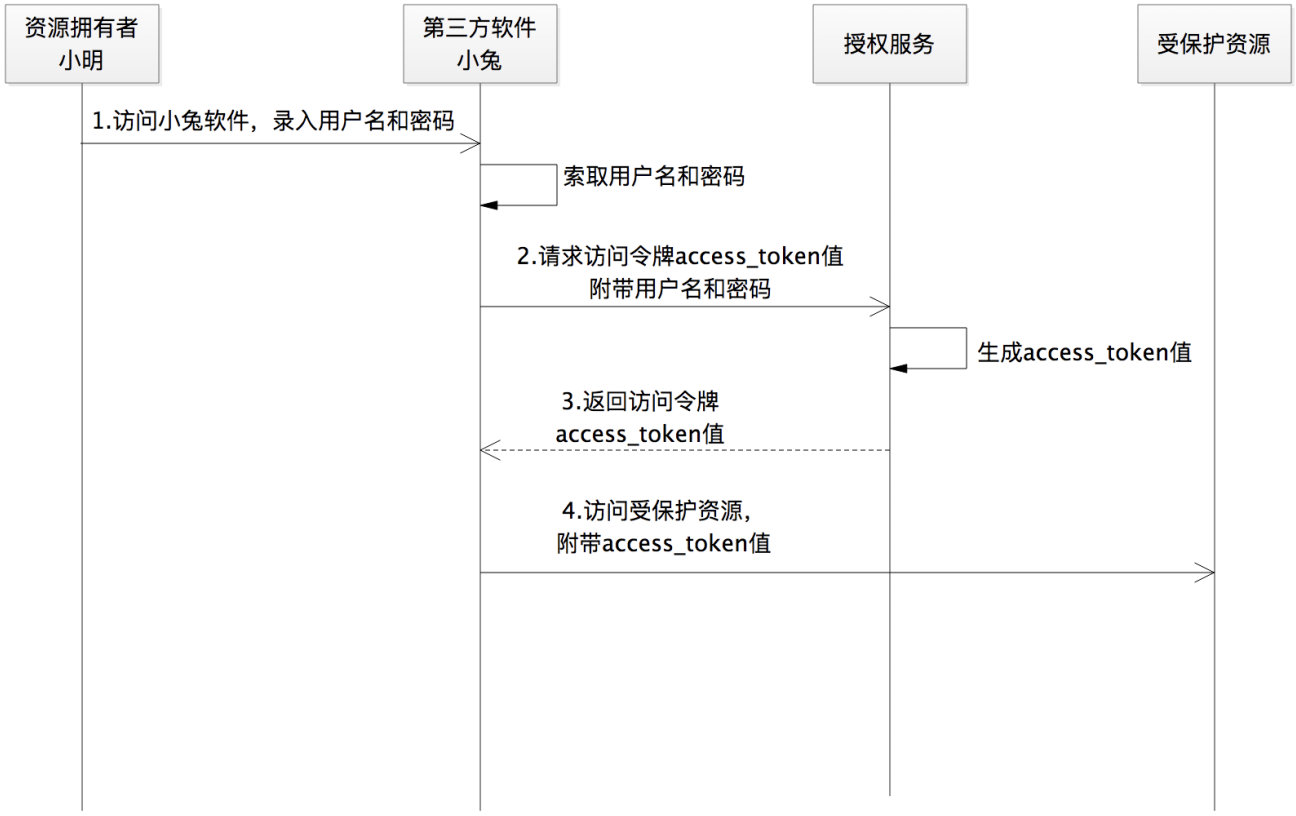

图1 资源拥有者凭据许可类型的流程

步骤 1：当用户访问第三方软件小兔时，会提示输入用户名和密码。索要用户名和密码，就是资源拥有者凭据许可类型的特点。

步骤 2：**这里的 grant_type 的值为 password**，告诉授权服务使用资源拥有者凭据许可凭据的方式去请求访问。

复制代码

```
Map<String, String> params = new HashMap<String, String>();
params.put("grant_type","password");
params.put("app_id","APPIDTEST");
params.put("app_secret","APPSECRETTEST");
params.put("name","NAMETEST");
params.put("password","PASSWORDTEST");

String accessToken = HttpURLClient.doPost(oauthURl,HttpURLClient.mapToStr(para
```

步骤 3：授权服务在验证用户名和密码之后，生成 access_token 的值并返回给第三方软件。

复制代码

```
if("password".equals(grantType)){
    String appSecret = request.getParameter("app_secret");
    String username = request.getParameter("username");
    String password = request.getParameter("password");

    if(!"APPSECRETTEST".equals(appSecret)){
        response.getWriter().write("app_secret is not available");
        return;
    }
    if(!"USERNAMETEST".equals(username)){
        response.getWriter().write("username is not available");
        return;
    }
    if(!"PASSWORDTEST".equals(password)){
        response.getWriter().write("password is not available");
        return;
    }
    String accessToken = generateAccessToken(appId,"USERTEST");//生成访问令牌acc
    response.getWriter().write(accessToken);
```

到了这里，你可以掌握到一个信息：如果软件是官方出品的，又要使用 OAuth 2.0 来保护我们的 Web API，那么你就可以使用小兔软件的做法，采用资源拥有者凭据许可类型。

无论是我们的架构、系统还是框架，都是致力于解决现实生产中的各种问题的。除了资源拥有者凭据许可类型外，OAuth 2.0 体系针对现实的环境还提供了客户端凭据许可和隐式许可类型。接下来，让我们继续看看这两种授权许可类型吧。

客户端凭据许可

如果没有明确的资源拥有者，换句话说就是，小兔软件访问了一个不需要用户小明授权的数据，比如获取京东 LOGO 的图片地址，这个 LOGO 信息不属于任何一个第三方用户，再比如其它类型的第三方软件来访问平台提供的省份信息，省份信息也不属于任何一个第三方用户。

此时，在授权流程中，就不再需要资源拥有者这个角色了。当然了，**你也可以形象地理解为 “资源拥有者被塞进了第三方软件中” 或者 “第三方软件就是资源拥有者”**。这种场景下的授权，便是客户端凭据许可，第三方软件可以直接使用注册时的 app_id 和 app_secret 来换回访问令牌 token 的值。

我们还是以小明使用小兔软件为例，来看下客户端凭据许可的整个授权流程，如下图所示：

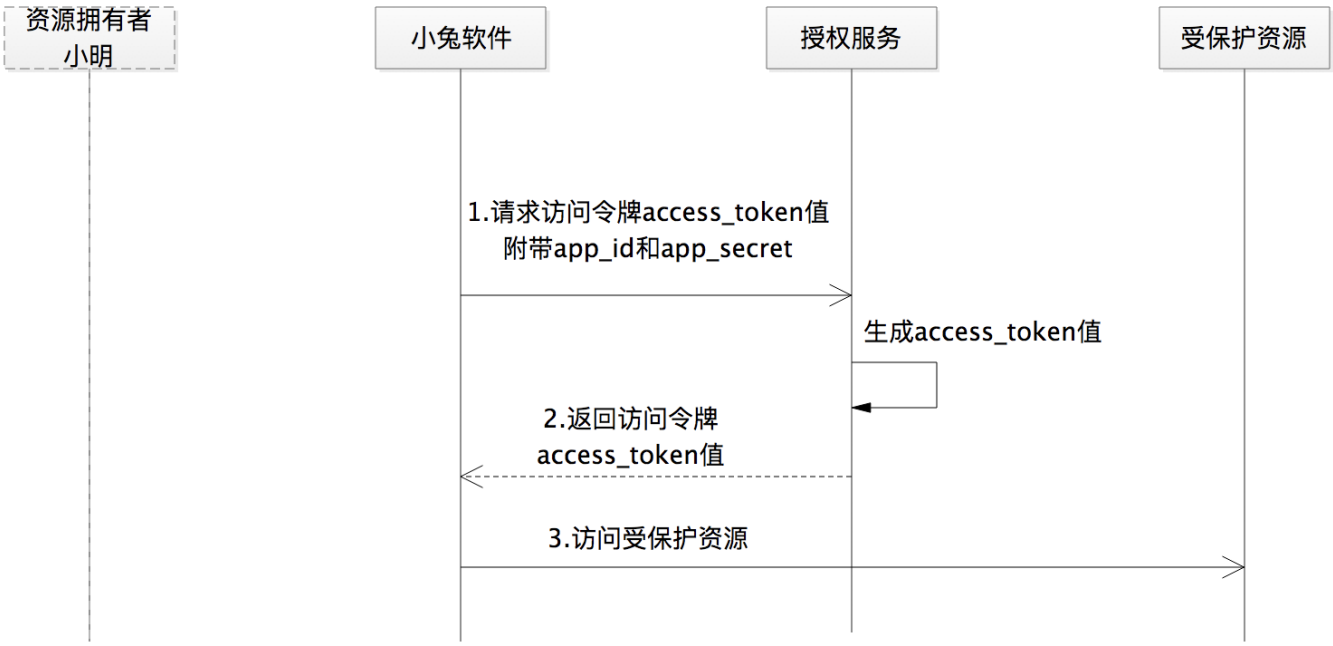

图2 客户端凭据许可授权流程

另外一点呢，因为授权过程没有了资源拥有者小明的参与，小兔软件的后端服务可以随时发起 access_token 的请求，所以这种授权许可也不需要刷新令牌。

这样一来，客户端凭据许可类型的关键流程，就是以下两大步。

步骤 1：第三方软件小兔通过后端服务向授权服务发送请求，**这里 grant_type 的值为 client_credentials**，告诉授权服务要使用第三方软件凭据的方式去请求访问。

复制代码

```
Map<String, String> params = new HashMap<String, String>();
params.put("grant_type","client_credentials");
params.put("app_id","APPIDTEST");
params.put("app_secret","APPSECRETTEST");

String accessToken = HttpURLClient.doPost(oauthURl,HttpURLClient.mapToStr(para
```

步骤 2：在验证 app_id 和 app_secret 的合法性之后，生成 access_token 的值并返回。

复制代码

```
String grantType = request.getParameter("grant_type");
```

```
String appId = request.getParameter("app_id");

if(!"APPIDTEST".equals(appId)){
    response.getWriter().write("app_id is not available");
    return;
}
if("client_credentials".equals(grantType)){
    String appSecret = request.getParameter("app_secret");
    if(!"APPSECRETTEST".equals(appSecret)){
        response.getWriter().write("app_secret is not available");
        return;
    }
    String accessToken = generateAccessToken(appId,"USERTEST");//生成访问令牌acc
    response.getWriter().write(accessToken);
}
```

到这里，我们再小结下。在获取一种不属于任何一个第三方用户的数据时，并不需要类似小明这样的用户参与，此时便可以使用客户端凭据许可类型。

接下来，我们再一起看看今天要讲的最后一种授权许可类型，就是隐式许可类型。

隐式许可

让我们再想象一下，如果小明使用的小兔打单软件应用没有后端服务，就是在浏览器里面执行的，比如纯粹的 JavaScript 应用，应该如何使用 OAuth 2.0 呢？

其实，这种情况下的授权流程就可以使用隐式许可流程，可以理解为第三方软件小兔直接嵌入浏览器中了。

在这种情况下，小兔软件对于浏览器就没有任何保密的数据可以隐藏了，也不再需要应用密钥 app_secret 的值了，也不用再通过授权码 code 来换取访问令牌 access_token 的值了。因为使用授权码的目的之一，就是把浏览器和第三方软件的信息做一个隔离，确保浏览器看不到第三方软件最重要的访问令牌 access_token 的值。

因此，**隐式许可授权流程的安全性会降低很多**。在授权流程中，没有服务端的小兔软件相当于是嵌入到了浏览器中，访问浏览器的过程相当于接触了小兔软件的全部，因此我用虚线框来表示小兔软件，整个授权流程如下图所示：

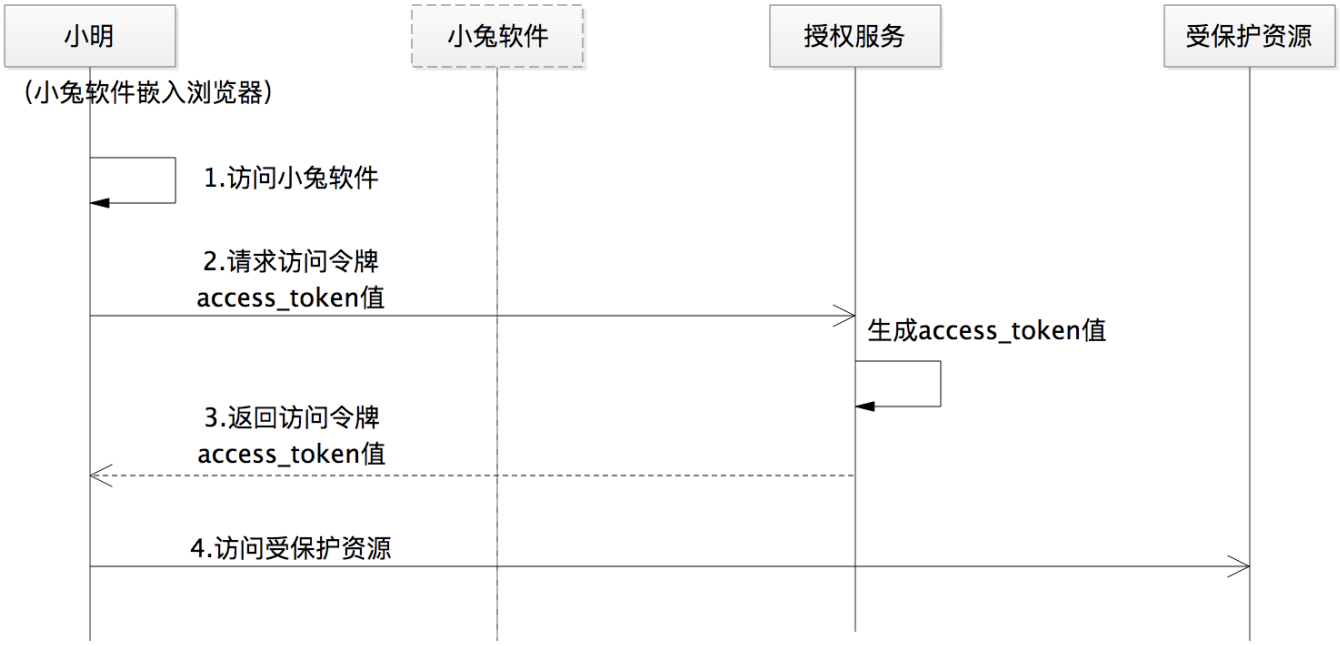

图3 隐式许可授权流程

接下来，我使用 Servlet 的 Get 请求来模拟这个流程，一起看看相关的示例代码。

步骤 1：用户通过浏览器访问第三方软件小兔。此时，第三方软件小兔实际上是嵌入浏览器中执行的应用程序。

步骤 2：这个流程和授权码流程类似，只是需要特别注意一点，**response_type 的值变成了 token**，是要告诉授权服务直接返回 access_token 的值。随着我们后续的讲解，你会发现隐式许可流程是唯一在前端通信中要求返回 access_token 的流程。对，就这么 “大胆” ，但 “不安全” 。

复制代码

```
Map<String, String> params = new HashMap<String, String>();
params.put("response_type","token");//告诉授权服务直接返回access_token
params.put("redirect_uri","http://localhost:8080/AppServlet-ch02");
params.put("app_id","APPIDTEST");

String toOauthUrl = URLParamsUtil.appendParams(oauthUrl,params);//构造请求授权的

response.sendRedirect(toOauthUrl);
```

步骤 3：生成 acccess_token 的值，通过前端通信返回给第三方软件小兔。

复制代码

```
String responseType = request.getParameter("response_type");
```

```
String redirectUri =request.getParameter("redirect_uri");
String appId = request.getParameter("app_id");
if(!"APPIDTEST".equals(appId)){
    return;
}

if("token".equals(responseType)){
    //隐式许可流程（模拟），DEMO CODE，注意：该流程全部在前端通信中完成
    String accessToken = generateAccessToken(appId,"USERTEST");//生成访问令牌acc

    Map<String, String> params = new HashMap<String, String>();
    params.put("redirect_uri",redirectUri);
    params.put("access_token",accessToken);

    String toAppUrl = URLParamsUtil.appendParams(redirectUri,params);//构造第三
    response.sendRedirect(toAppUrl);//使用sendRedirect方式模拟前端通信
}
```

如果你的软件就是直接嵌入到了浏览器中运行，而且还没有服务端的参与，并且还想使用 OAuth 2.0 流程的话，也就是像上面我说的小兔这个例子，那么便可以直接使用隐式许可类型了。

如何选择？

现在，我们已经理解了 OAuth 2.0 的 4 种授权许可类型的原理与流程。那么，我们应该如何选择到底使用哪种授权许可类型呢？

这里，我给你的建议是，在对接 OAuth 2.0 的时候先考虑授权码许可类型，其次再结合现实生产环境来选择：

> 如果小兔软件是官方出品，那么可以直接使用资源拥有者凭据许可；
>
> 如果小兔软件就是只嵌入到浏览器端的应用且没有服务端，那就只能选择隐式许可；
>
> 如果小兔软件获取的信息不属于任何一个第三方用户，那可以直接使用客户端凭据许可类型。

总结

好了，我们马上要结束这篇文章了，在这之前呢，我们一直讲的是授权码许可类型，你已经知道了这是一种流程最完备、安全性最高的授权许可流程。不过呢，现实世界总是有各

种各样的变化，OAuth 2.0 也要适应这样的变化，所以才有了我们今天讲的另外这三种许可类型。同时，关于如何来选择使用这些许可类型，我前面也给了大家一个建议。

加上前面我们讲的授权码许可类型，我们一共讲了 4 种授权许可类型，它们最显著的区别就是**获取访问令牌 access_token 的方式不同**。最后，我通过一张表格来对比下：

授权许可类型	获取访问令牌的方式
授权码许可	通过授权码code获取access_token
客户端凭据许可	通过第三方软件的app_id和app_secret获取access_token
隐式许可	通过嵌入浏览器中的第三方软件的app_id来获取access_token
资源拥有者凭据许可	通过资源拥有者的用户名和密码获取access_token

图4 OAuth 2.0的4种授权许可类型对比

除了上面这张表格所展现的 4 种授权许可类型的区别之外，我希望你还能记住以下两点。

1. 所有的授权许可类型中，授权码许可类型的安全性是最高的。因此，只要具备使用授权码许可类型的条件，我们一定要首先授权码许可类型。
2. 所有的授权许可类型都是为了解决现实中的实际问题，因此我们还要结合实际的生产环境，在保障安全性的前提下选择最合适的授权许可类型，比如使用客户端凭据许可类型的小兔软件就是一个案例。

我把今天用到的代码放到了 GitHub 上，你可以点击这个链接查看。

思考题

如果受限于应用特性所在的环境，比如在没有浏览器参与的情况下，我们应该如何选择授权许可类型呢，还可以使用授权码许可流程吗？

欢迎你在留言区分享你的观点，也欢迎你把今天的内容分享给其他朋友，我们一起交流。

提建议

上一篇 05 | 如何安全、快速地接入OAuth 2.0？

下一篇 07 | 如何在移动App中使用OAuth 2.0？

精选留言 (18)

写留言

Geek_4b64df
2020-07-13

可以，微信小程序就用的授权码，通过ajax获取code

展开

作者回复: 是的

1 2

蒋胜琳
2020-07-13

搜索了好多博客，都没有把各种许可讲明白的，在这里可算明白了

作者回复: 感谢支持

1

leros
2020-07-11

理论上讲，没有浏览器的情况下，也可以实现授权码许可流程。这种流程需要一个user agent让用户和第三方软件互动，并接受来自授权服务器的重定向，而浏览器只是最常见的user agent，不过不太清楚具体实践中这一块怎么处理的。

展开

作者回复: 在07我们讲到了不需要浏览器参与使用授权码的场景。

1

tongmin_tsai
2020-07-28

老师，如果类似于传统使用session那样，如果我每次使用access_token请求后，希望重置过期时间，怎么做才是最佳实践？

展开

慎独明强
2020-07-23

OAuth 2.0授权码许可类型，这个是最安全的，需要用户登录与授权，返回授权码给第三方软件，第三方软件再去换取token和刷新token。 资源拥有者访问类型：用户 第三方软

件与授权服务都是官方的，那么用户通过输入用户名和密码，去授权服务拿到token。

展开

 

在路上
2020-07-19

王老师，公司内部的统一登录，一般选用OAuth那种验证类型呢?

 3 

suhuijie
2020-07-16

资源拥有者凭据许可，直接用用户名密码是前端登陆还是后端登陆？例子中还需要携带appID和秘钥，那就是需要后端登陆。那这样的情况就是，官方出品所的产品都要自己提供登录接口，然后后端服务绕一圈去登录？能否直接在前端直接用用户名和密码登录？

展开

作者回复:
登录和授权是两个通路的事情，任何授权都是在用户登录之后进行的。

用户的用户名和密码是来进行登陆的，appID和秘钥是用来换取访问令牌的。

授权的本质是令牌，令牌是怎么换来的呢，是用户登录之后的授权，那生成令牌的时候是怎么跟用户的登录关联上的呢，因为授权和登录的后台处理都是”一家“的。

并不是说后端服务绕一圈再去登录，而且生产中登录和授权本来也是有分别的系统来进行处理。

 

千
2020-07-15

在这里系统的学习了一下oauth2.0，讲的很清楚，收获很多

作者回复: 感谢支持

 

kylexy_0817
2020-07-14

感觉用户名密码获取到的access_token，和sessionid很像，只是session id只要用户有操

作，就会自动续约，但access_token，会定期更新？

展开

作者回复: sessionid和access_token是两个完全不同的事物，sessionid是会话，access_token不能等同于会话，它是第三方软件代表用户访问数据的凭证。

Ryan Pan
2020-07-14

资源拥有者凭据许可是用用户帐户密码取得token，那有没有从第三方帐户（微信等）取得token的方式呢？
例如用从第三方取得的access_token换授权服务的token之类的

Geek_9ae2b9
2020-07-13

对比了授权码许可类型，本文介绍的三种授权流程中，都不需要经过用户小明的“授权”操作吗？

作者回复: 只有客户端凭据许可不需要

Geek_4b64df
2020-07-13

您是 隐式许可 讲的最透彻的一位老师

展开

作者回复: 感谢支持

DB聪
2020-07-12

请问”如果小兔软件是官方出品，那么可以直接使用客户端凭据许可；” 是否应该是”如果小兔软件是官方出品，那么可以直接使用资源拥有者凭据许可；” ？

展开

作者回复: 感谢指正，这里是一个错误。

在如何选择，这一段，应该更正为：【如果小兔软件是官方出品，那么可以直接使用资源拥有者凭据许可；】

其实，我们在刚开始的时候一直在讲官方出品-资源拥有者凭据许可。再次感谢，已做修改。

1

Geek_9d0e04
2020-07-12

总结部分：如果小兔软件是官方出品，那么可以直接使用客户端凭据许可；这个写错了吧，应该是使用资源拥有者凭证许可。还有一个问题，隐式许可时，没有refresh_token，那accesss_token过期了，每次过期都需要资源拥有者参与，再走一遍认证过程嘛？？这不是没法在生产中使用吗

展开

作者回复: 感谢指正，这里是一个错误。已联系更正修改。

针对隐式许可类型的场景的特点是，浏览器内的应用都是短暂运行，只会在被加载到浏览器的这期间保持会话，所以刷新令牌在这里的作用很有限。

hhhh
2020-07-12

对于授权服务而言，支持隐式类型还要对注册的域名增加跨域支持吧？

作者回复: 隐式许可类型实际应用在生产环境中的机会非常小了，比如直接嵌入浏览器这样的应用，在我们的实际生产环境中很少。因此，大家的重点不要放在这个授权许可类型上。

所以，在这节课的最后，我们建议：【所有的授权许可类型中，授权码许可类型的安全性是最高的。因此，只要具备使用授权码许可类型的条件，我们一定要首先授权码许可类型。】

1

大秦皇朝

2020-07-11

老师好，文中说的“因为使用授权码的目的之一，就是把浏览器和第三方软件的信息做一个隔离，确保浏览器看不到第三方软件最重要的访问令牌 access_token 的值。”，这个没有理解，因为没有后台服务呀，直接请求完的access_token不直接返回给浏览器了么？为什么浏览器看不到access_token？

展开

作者回复: “因为使用授权码的目的之一，就是把浏览器和第三方软件的信息做一个隔离，确保浏览器看不到第三方软件最重要的访问令牌 access_token 的值。”
这句话呢是在解释【授权码许可流程】中【授权码】的作用之一，也是在反向说明【隐式许可】不需要授权码了。

岁月不饶人
2020-07-11

如果内网两个服务器需要鉴权通信，就可以使用隐式授权？

作者回复: 隐式许可的特点是第三方软件无法对浏览器隐藏任何秘密，因为第三方软件的任何信息都暴露在了浏览器里面，第三方软件也就没有办法，也无必要再持有秘钥，失去了一层保护，授权服务会直接返回access_token给到“嵌入”浏览器中的第三方软件，但是用户授权的动作还是要有的，用户不授权，授权服务不颁发access_token，当然这个access_token也暴露在了浏览器下面，所以隐式许安全性低了很多，另外也有它的使用局限性，仅仅在浏览器短暂运行的第三方软件的场景。

内部服务之间的鉴权的时候，【一般】的做法是这样，比如服务A调用服务B，B 要对 A 鉴权，当A申请去调用B的时候，B会”留下“A的app_id，同时会告知一个token，某种【层度】上跟隐式许可类似，这里没有了秘钥，但隐式许可是需要【用户】这个角色参与的，所以严格上来说不是OAuth 2.0 里面的隐式许可授权类型。倒是有点像失去了秘钥的客户端凭据许可。

OAuth 2.0 从思想层面来理解是包含了一种授权的思想，可以【简单的理解】就是用令牌代替**去访问**。

Zhou
2020-07-11

client_credential不需要refresh_token，是不是意味着每次调用api之前都要调用auth server拿token?password模式还是需要refresh_token的吧？

展开

作者回复: 不用，token有有效期，一段时间内都可用。
password模式支持refresh_token，四种基本授权许可类型中隐式许可和客户端凭据许可没有refresh_token。

1