



下载APP



## 03 | 授权服务：授权码和访问令牌的颁发流程是怎样的？

2020-07-04 王新栋

OAuth 2.0实战课

[进入课程 >](#)**讲述：李海明**

时长 23:04 大小 21.14M



你好，我是王新栋。

在上一讲，我从为什么需要授权码这个问题开始，为你串了一遍授权码许可流程整体的通信过程。在接下来的三讲中，我会着重为你讲解关于授权服务的工作流程、授权过程中的令牌，以及如何接入 OAuth 2.0。这样一来，你就可以吃透授权码许可这一最经典、最完备、最常用的授权流程了，以后再处理授权相关的逻辑就更得心应手了。现在呢，让我们开始这一讲。

在介绍授权码许可类型时，我提到了很多次“授权服务”。一句话概括，授权服务就负责颁发访问令牌的服务。更进一步地讲，OAuth 2.0 的核心是授权服务，而授权服务的核心就是令牌。



为什么这么说呢？当第三方软件比如小兔，要想获取小明在京东店铺的订单，就必须先从京东商家开放平台的授权服务那里获取访问令牌，进而通过访问令牌来“代表”小明去请求小明的订单数据。这不恰恰就是整个 OAuth 2.0 授权体系的核心吗？

那么，授权服务到底是怎么生成访问令牌的，这其中包含了哪些操作呢？还有一个问题是，访问令牌过期了而用户又不在场的情况下，又如何重新生成访问令牌呢？

带着这两个问题，我们就以授权码许可类型为例，一起深入探索下授权服务这个核心组件吧。

## 授权服务的工作过程


开始之前，你还是要先回想下小明给小兔软件授权订单数据的整个流程。

我们说小兔软件先要让小明去京东商家开放平台那里给它授权数据，那这里是不是你觉得很奇怪？你总不能说，“嘿，京东，你把数据给小兔用吧”，那京东肯定会回复说，“小明，小兔是谁啊，没在咱家备过案，我不能给他，万一是骗子呢？”

对吧，你想想是不是这个逻辑。所以，授权这个大动作的前提，肯定是小兔要去平台那里“备案”，也就是注册。注册完后，京东商家开放平台就会给小兔软件 app\_id 和 app\_secret 等信息，以方便后面授权时的各种身份校验。

同时，注册的时候，第三方软件也会请求受保护资源的可访问范围。比如，小兔能否获取小明店铺 3 个月以前的订单，能否获取每条订单的所有字段信息等等。这个权限范围，就是 scope。后面呢，我还会详细讲述范围控制。

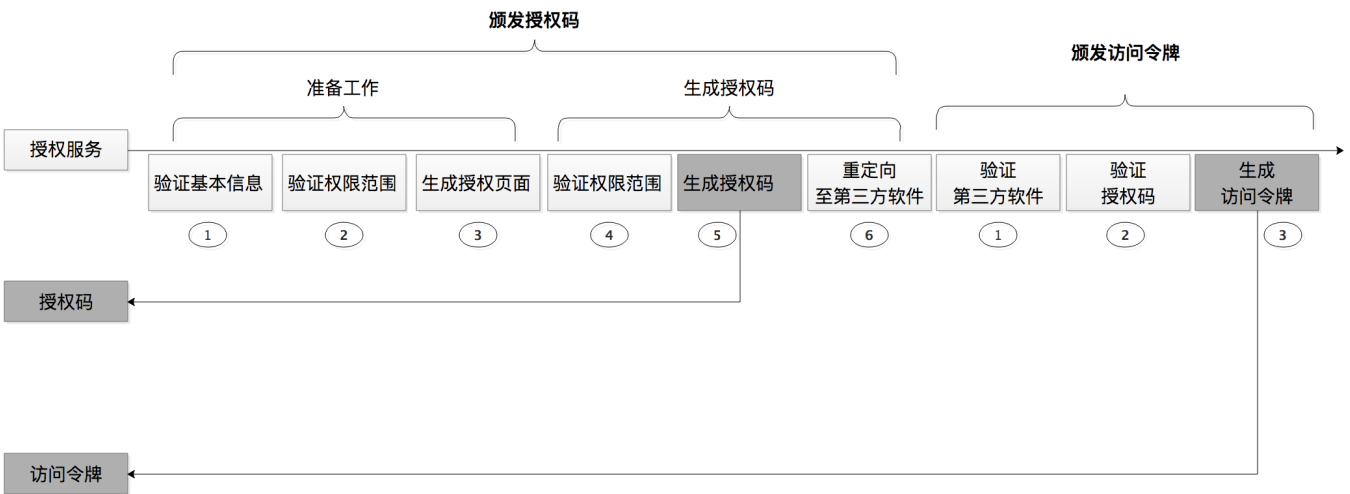
文字说起来有点抽象，咱们还是直接上代码吧。关于注册后的数据存储，我们使用如下 Java 代码来模拟：

 复制代码

```
1 Map<String,String> appMap = new HashMap<String, String>(); //模拟第三方软件注册之
2
3 appMap.put("app_id","APPID_RABBIT");
4 appMap.put("app_secret","APPSECRET_RABBIT");
5 appMap.put("redirect_uri","http://localhost:8080/AppServlet-ch03");
6 appMap.put("scope","nickname address pic");
```

备完案之后，咱们接着继续前进。小明过来让平台把他的订单数据给小兔，平台咔咔一查，对了下暗号，发现小兔是合法的，于是就要推进下一步了。

咱们上节课讲过，在授权码许可类型中，授权服务的工作，可以划分为两大部分，一个是**颁发授权码 code**，一个是**颁发访问令牌 access\_token**。为了更能表达授权码和访问令牌的存在，我在图中用深色将其标注了出来：



我们先看看颁发授权码 code 的流程。

### 过程一：颁发授权码 code

在这个过程中，授权服务需要完成两部分工作，分别是**准备工作**和**生成授权码 code**。

你可能会问了，这个“准备”都包括哪些工作？我们可以想到，小明在给第三方软件小兔打单软件进行授权的时候，会看到授权页面上有一个授权按钮，但是授权服务在小明看到这个授权按钮之前，实际上已经做了一系列动作。


这些动作，就是所谓的准备工作，包括验证基本信息、验证权限范围（第一次）和生成授权请求页面这三步。我们具体分析下。

#### 第一步，验证基本信息。

验证基本信息，包括对第三方软件小兔合法性和回调地址合法性的校验。

在 Web 浏览器环境下，颁发 code 的整个请求过程，都是浏览器通过前端通信来完成，这就意味着所有信息都有被冒充的风险。因此，授权服务必须对第三方软件的存在性做判断。

同样，回调地址也是可以伪造的。比如，不法分子将其伪装成钓鱼页面，或者是带有恶意攻击性的软件下载页面。因此从安全上考虑，授权服务需要对回调地址做基本的校验。

 复制代码


```
1 if(!appMap.get("redirect_uri").equals(redirectUri)){  
2     //回调地址不存在  
3 }
```

在授权服务的程序中，这两步验证通过后，就会生成或者响应一个页面（**属于授权服务器上的页面**），以提示小明进行授权。

## 第二步，验证权限范围（第一次）。

既然是授权，就会涉及范围。比如，我们使用微信登录第三方软件的时候，会看到微信提示我们，第三方软件可以获得你的昵称、头像、性别、地理位置等。如果你不想让第三方软件获取你的某个信息，那么可以不选择这一项。同样在小兔中也是一样，当小明为小兔进行授权的时候，也可以选择给小兔的权限范围，比如是否授予小兔获取 3 个月以前的订单的访问权限。

这就意味着，我们需要对小兔传过来的 scope 参数，与小兔注册时申请的权限范围做比对。如果请求过来的权限范围大于注册时的范围，就需要作出越权提示。**记住，此刻是第一次权限校验。**

 复制代码

```
1 String scope = request.getParameter("scope");  
2 if(!checkScope(scope)){  
3     //超出注册的权限范围  
4 }
```

## 第三步，生成授权请求页面。

这个授权请求页面就是授权服务上的页面，如下图所示：

Are you sure you want the authorization code?

appid: APPID\_RABBIT

☒ today

☐ history

approve

refuse

页面上显示了小兔注册时申请的 today、history 两种权限，小明可以选择缩小这个权限范围，比如仅授予获取 today 信息的权限。

至此，颁发授权码 code 的准备工作就完成了。你要注意哈，我一直强调说这也是准备工作，因为当用户点击授权按钮“approve”后，才会**生成授权码 code 值和访问令牌 acces\_token 值**，“一切才真正开始”。


这里需要说明下：在上面的准备过程中，我们忽略了小明登录的过程，但只有用户登录了才可以对第三方软件进行授权，授权服务才能够获得用户信息并最终生成 code 和 app\_id（第三方软件的应用标识）+ user（资源拥有者标识）之间的对应关系。你可以把登录部分的代码，作为附加练习。

小明点击“approve”按钮之后，生成授权码 code 的流程就正式开始了，主要包括验证权限范围（第二次）、处理授权请求生成授权码 code 和重定向至第三方软件这三大步。接下来，我们一起分析下这三步。

#### 第四步，验证权限范围（第二次）。

在步骤二中，生成授权页面之前授权服务进行的第一次校验，是对比小兔请求过来的权限范围 scope 和注册时的权限做的比对。这里的第二次验证权限范围，是用小明进行授权之后的权限，再次与小兔软件注册的权限做校验。


那这里为什么又要校验一次呢？因为这相当于一次用户的输入权限。小明选择了一定的权限范围给到授权服务，对于权限的校验我们要重视对待，凡是输入性数据都会涉及到合法性检查。另外，这也是要求我们养成一种**在服务端对输入数据的请求，都尽可能做一次合法性校验的好习惯**。

 复制代码

```
1 String[] rscope =request.getParameterValues("rscope");
2
3 if(!checkScope(rscope)){
4     //超出注册的权限范围
5 }
```


### 第五步，处理授权请求，生成授权码 code。

当小明同意授权之后，授权服务会校验响应类型 response\_type 的值。response\_type 有 code 和 token 两种类型的值。在这里，我们是用授权码流程来举例的，因此代码要验证 response\_type 的值是否为 code。

 复制代码

```
1 String responseType = request.getParameter("response_type");
2 if("code".equals(responseType)){
3
4 }
```

在授权服务中，需要将生成的授权码 code 值与 app\_id、user 进行关系映射。也就是说，一个授权码 code，表示某一个用户给某一个第三方软件进行授权，比如小明给小兔软件进行的授权。同时，我们需要将 code 值和这种映射关系保存起来，以便在生成访问令牌 access\_token 时使用。

 复制代码


```
1 String code = generateCode(appId,"USERTEST");//模拟登录用户为USERTEST
2
3 private String generateCode(String appId,String user) {
4     ...
5     String code = strb.toString();
6     codeMap.put(code,appId+"|"+user+"|"+System.currentTimeMillis());
7     return code;
8 }
```



在生成了授权码 code 之后，我们也按照上面所述绑定了响应的映射关系。这时，你还记得我之前讲到的授权码是临时的、一次性凭证吗？因此，我们还需要为 code 设置一个有效期。

OAuth 2.0 规范建议授权码 code 值有效期为 10 分钟，并且**一个授权码 code 只能被使用一次**。不过根据经验呢，在生产环境中 code 的有效期一般不会超过 5 分钟。关于授权码 code 相关的安全方面的内容，我还会在第 8 讲中详细讲述。


同时，授权服务还需要**将生成的授权码 code 跟已经授权的权限范围 rscope 进行绑定并存储**，以便后续颁发访问令牌时，我们能够通过 code 值取出授权范围并与访问令牌绑定。因为第三方软件最终是通过访问令牌来请求受保护资源的。

 复制代码

```
1 Map<String,String[]> codeScopeMap = new HashMap<String, String[]>();
2
3 codeScopeMap.put(code,rscope); //授权范围与授权码做绑定
```

## 第六步，重定向至第三方软件。

生成授权码 code 值之后，授权服务需要将该 code 值告知第三方软件小兔。开始时我们提到，颁发授权码 code 是通过前端通信完成的，因此这里采用重定向的方式。这一步的重定向，也是我在上一讲中提到的第二次重定向。

 复制代码

```
1 Map<String, String> params = new HashMap<String, String>();
2 params.put("code",code);
3
4 String toAppUrl = URLParamsUtil.appendParams(redirectUri,params); //构造第三方软件
5
6 response.sendRedirect(toAppUrl); //授权码流程的“第二次”重定向
```

到此，颁发授权码 code 的流程全部完成。当小兔获取到授权码 code 值以后，就可以开始请求访问令牌 access\_token 的值了，也就是我们即将开始的过程二。

## 过程二：颁发访问令牌 access\_token

我们在过程一中介绍了授权码 code 的生成流程，但小兔最终是要获取到访问令牌 access\_token，才可以去请求受保护资源。而授权码呢，正如我在上一讲提到的，只是一个换取访问令牌 access\_token 的临时凭证。

当小兔拿着授权码 code 来请求的时候，授权服务需要为之生成最终的请求访问令牌。这个过程主要包括验证第三方软件小兔是否存在、验证 code 值是否合法和生成 access\_token 值这三大步。接下来，我们一起分析下每一步。

### 第一步，验证第三方软件是否存在。

此时，接收到的 grant\_type 的类型为 authorization\_code。

[复制代码](#)

```
1 String grantType = request.getParameter("grant_type");
2 if("authorization_code".equals(grantType)){
3
4 }
```

由于颁发访问令牌是通过后端通信完成的，所以这里除了要校验 app\_id 外，还要校验 app\_secret。


[复制代码](#)

```
1 if(!appMap.get("app_id").equals(appId)){
2     //app_id不存在
3 }
4
5 if(!appMap.get("app_secret").equals(appSecret)){
6     //app_secret不合法
7 }
```

### 第二步，验证授权码 code 值是否合法。

授权服务在颁发授权码 code 的阶段已经将 code 值存储了起来，此时对比从 request 中接收到的 code 值和从存储中取出来的 code 值。在我们给出的课程 [🔗 相关代码](#) 中，code 值对应的 key 是 app\_id 和 user 的组合值。



 复制代码

```
1 String code = request.getParameter("code");
2 if(!isExistCode(code)){//验证code值
3     //code不存在
4     return;
5 }
6 codeMap.remove(code);//授权码一旦被使用，须立即作废
```


这里我们一定要记住，**确认过授权码 code 值有效以后，应该立刻从存储中删除当前的 code 值**，以防止第三方软件恶意使用一个失窃的授权码 code 值来请求授权服务。

### 第三步，生成访问令牌 access\_token 值。

关于按照什么规则来生成访问令牌 access\_token 的值，OAuth 2.0 规范中并没有明确规定，但必须符合三个原则：**唯一性、不连续性、不可猜性**。在我们给出的 Demo 中，我们使用 UUID 来作为示例的。

和授权码 code 值一样，我们需要将访问令牌 access\_token 值存储起来，并将其与第三方软件的应用标识 app\_id 和资源拥有者标识 user 进行关系映射。也就是说，**一个访问令牌 access\_token 表示某一个用户给某一个第三方软件进行授权**。

同时，**授权服务还需要将授权范围跟访问令牌 access\_token 做绑定**。最后，还需要为该访问令牌设置一个过期时间 expires\_in，比如 1 天。

 复制代码

```
1 Map<String,String[]> tokenScopeMap = new HashMap<String, String[]>();
2
3 String accessToken = generateAccessToken(appId,"USERTEST");//生成访问令牌access_
4 tokenScopeMap.put(accessToken,codeScopeMap.get(code));//授权范围与访问令牌绑定
5
6 //生成访问令牌的方法
7 private String generateAccessToken(String appId,String user){
8
9     String accessToken = UUID.randomUUID().toString();
10    String expires_in = "1";//1天时间过期
11    tokenMap.put(accessToken,appId+"|"+user+"|"+System.currentTimeMillis()+"|"+e
12
13    return accessToken;
14 }
```

正因为 OAuth 2.0 规范没有约束访问令牌内容的生成规则，所以我们有更高的自由度。我们既可以像 Demo 中那样生成一个 UUID 形式的数据存储起来，让授权服务和受保护资源共享该数据；也可以将一些必要的信息通过结构化的处理放入令牌本身。**我们将包含了一些信息的令牌，称为结构化令牌，简称 JWT。**在下一讲中，我还会与你详细讲述 JWT。

至此，授权码许可类型下授权服务的两大主要过程，也就是颁发授权码和颁发访问令牌的流程，我就与你讲完了。

接下来，你在阅读别人的授权流程代码，或者是使用诸如通过微信登录的第三方软件的时候，就会明白背后的原理了。同时，你在自己搭建一个授权服务流程时，也会更加得心应手。这一切的原因，都在于颁发授权码和颁发访问令牌，就是授权服务的核心。

到这里，你应该还会注意到一个问题，在生成访问令牌的时候，我们还给它附加了一个过期时间 `expires_in`，这意味着访问令牌会在一定的时间后失效。访问令牌失效，就意味着资源拥有者给第三方软件的授权失效了，第三方软件无法继续访问资源拥有者的受保护资源了。

这时，如果你还想继续使用第三方软件，就只能重新点击授权按钮，比如小明给小兔软件授权以后，正在愉快地处理他店铺的订单数据，结果没过多久，突然间小兔软件再次让小明进行授权。此刻，我们可以替小明感受一下他的心情。

显然，这样的用户体验非常糟糕。为此，OAuth 2.0 中引入了刷新令牌的概念，也就是刷新访问令牌 `access_token` 的值。这就意味着，有了刷新令牌，用户在一定期限内无需重新点击授权按钮，就可以继续使用第三方软件。

接下来，我们就一起看看刷新令牌的工作原理吧。

## 刷新令牌

刷新令牌也是给第三方软件使用的，同样需要遵循**先颁发再使用**的原则。因此，我们还是从颁发和使用两个环节来学习刷新令牌。不过，这个颁发和使用流程和访问令牌有些是相同的，所以我只会和你重点讲述其中的区别。

## 颁发刷新令牌

其实，颁发刷新令牌和颁发访问令牌是一起实现的，都是在过程二的步骤三生成访问令牌 `access_token` 中生成的。也就是说，第三方软件得到一个访问令牌的同时，也会得到一个刷新令牌：

[复制代码](#)

```
1 Map<String,String> refreshTokenMap = new HashMap<String, String>();
2
3 String refreshToken = generateRefreshToken(appId,"USERTEST");//生成刷新令牌refre
4
5 private String generateRefreshToken(String appId,String user){
6
7     String refreshToken = UUID.randomUUID().toString();
8
9     refreshTokenMap.put(refreshToken,appId+"|"+user+"|"+System.currentTimeMillis
10     return refreshToken;
11
12 }
```

看到这里你可能要问了，为什么要一起生成访问令牌和刷新令牌呢？

其实，这就回到了刷新令牌的作用上了。刷新令牌存在的初衷是，在访问令牌失效的情况下，为了不让用户频繁手动授权，用来通过系统重新请求**生成一个新的访问令牌**。那么，如果访问令牌失效了，而“身边”又没有一个刷新令牌可用，岂不是又要麻烦用户进行手动授权了。所以，它必须得和访问令牌一起生成。

到这里，我们就解决了刷新令牌的颁发问题。


## 使用刷新令牌

说到刷新令牌的使用，我们需要先明白一点。在 OAuth 2.0 规范中，刷新令牌是一种特殊的授权许可类型，是嵌入在授权码许可类型下的一种特殊许可类型。在授权服务的代码里，当我们接收到这种授权许可请求的时候，会先比较 `grant_type` 和 `refresh_token` 的值，然后做下一步处理。

这其中的流程主要包括如下两大步骤。


### 第一步，接收刷新令牌请求，验证基本信息。

此时请求中的 `grant_type` 值为 `refresh_token`。

 复制代码


```
1 String grantType = request.getParameter("grant_type");
2 if("refresh_token".equals(grantType)){
3
4 }
```

和颁发访问令牌前的验证流程一样，这里我们也需要验证第三方软件是否存在。需要注意的是，这里需要同时验证刷新令牌是否存在，目的就是要保证传过来的刷新令牌的合法性。

 复制代码

```
1 String refresh_token = request.getParameter("refresh_token");
2
3 if(!refreshTokenMap.containsKey(refresh_token)){
4     //该refresh_token值不存在
5 }
```

另外，我们还需要验证刷新令牌是否属于该第三方软件。授权服务是将颁发的刷新令牌与第三方软件、当时的授权用户绑定在一起的，因此这里需要判断该刷新令牌的归属合法性。

 复制代码

```
1 String appStr = refreshTokenMap.get("refresh_token");
2 if(!appStr.startsWith(appId+"|"+"USERTEST")){
3     //该refresh_token值不是颁发给该第三方软件的
4 }
```

**需要注意，一个刷新令牌被使用以后，授权服务需要将其废弃，并重新颁发一个刷新令牌。**

## 第二步，重新生成访问令牌。

生成访问令牌的处理流程，与颁发访问令牌环节的生成流程是一致的。授权服务会将新的访问令牌和新的刷新令牌，一起返回给第三方软件。这里就不再赘述了。

## 总结

今天的课马上又要结束了，我和你讲了授权码许可类型下授权服务的工作原理。授权服务可以说是整个 OAuth 2.0 体系中的“灵魂”组件，任何一种许可类型都离不开它的支持，它也是最复杂的组件。

这是因为它将复杂性尽可能地“揽在了自己身上”，才使得诸如小兔这样的第三方软件接入 OAuth 2.0 的时候更加便捷。那关于如何快速地接入 OAuth 2.0，我在第 5 讲中和你详细展开。

授权服务的步骤流程比较多，因此我把这节课配套的代码放到了 [GitHub](#) 上，可以帮助你更好地理解授权服务的流程。

总结来讲，关于这一讲，我希望你能记住以下 3 点。

1. 授权服务的核心就是，**先颁发授权码 code 值，再颁发访问令牌 access\_token 值。**
2. 在颁发访问令牌的同时还会颁发刷新令牌 refresh\_token 值，这种机制可以在无须用户参与的情况下用于生成新的访问令牌。正如我们讲到的小明使用小兔软件的例子，当访问令牌过期的时候，刷新令牌的存在可以大大提高小明使用小兔软件的体验。
3. 授权还要有授权范围，**不能让第三方软件获得比注册时权限范围还大的授权，也不能获得超出了用户授权的权限范围，始终确保最小权限安全原则。**比如，小明只为小兔软件授予了获取当天订单的权限，那么小兔软件就不能访问小明店铺里面的历史订单数据。

## 思考题

刷新令牌有过期时间吗，会一直有效吗？和我说说你的想法吧。

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

更多课程推荐

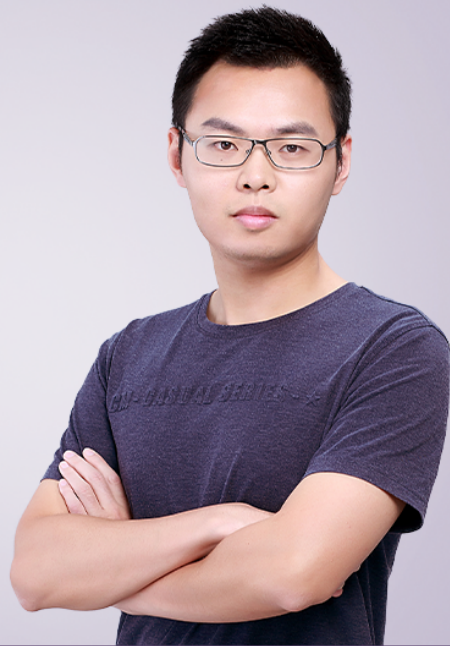
# 设计模式之美

前 Google 工程师手把手教你写高质量代码

王争

前 Google 工程师

《数据结构与算法之美》专栏作者



涨价倒计时 🕒

限时秒杀 **¥149**，7月31日涨价至 **¥299**

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 02 | 授权码许可类型中，为什么一定要有授权码？

下一篇 04 | 在OAuth 2.0中，如何使用JWT结构化令牌？

## 精选留言 (34)

💬 写留言



哈德韦

2020-07-05

请问什么是 rscope ? 和上下文中的 scope 是一回事吗 ? 这个 r 代表什么 ?

作者回复: 对应的权限 都是同一个权限，这里用rscope是受保护资源服务再次确认的权限，r是replay。

💬 1

👍 3



申玉宝

2020-07-05

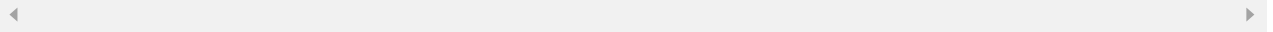
为什么要 通过刷新令牌让第三方不断刷新token有效期，而不是直接给访问token一个更



长的有效期？后者更简单

展开 ∨

作者回复: 为了安全性的考虑，是不可以让“token一个更长的有效期”存在的。



💬 6

👍 3



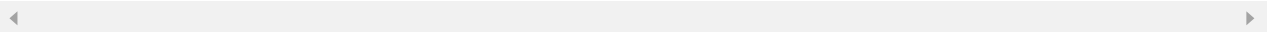
暖色浮生

2020-07-04

刷新令牌有过期时间吧，不过一般设置的时间比较长。反正微信公众号的挺长的，还有一直不明白 scope 的 权限范围指的是一个什么样的范围

展开 ∨

作者回复: SCOPE的权限范围非常重要，OAuth 2.0 本着【最小权限范围】原则，来支持用户对第三方软件授权。比如小兔打单软件，他的主要“行当”就是帮助小明打印订单，那么它的权限范围就是调用跟订单打印相关的API，比如单条查询订单API、批量查询订单API，那么查询小明店铺其它的API就要受限，在小兔打单软件申请成为开放平台的应用的时候就要做一次权限范围的选择，另外，当小明给小兔进行授权的时候，也会让小明去选择并确认，总之就是不要让小兔打单软件有超过其正常权限的范围，来充分保护小明店铺的数据。后面的课程，我们还会详细讲解关于SCOPE的种类和用法。



💬 1

👍 3



约书亚

2020-07-05

在上一节回答留言时，老师提到，用refresh刷新access时，如果access没过期，那会给这个access续期而不会重新生成？

这节课没提到这个。请问这是oauth规定的么？或者一般都这样实现？

因为我感觉无限续期会增加access被破解的风险

展开 ∨

作者回复: 感谢 约书亚 指正 我翻看了 之前的回复 做了修改。

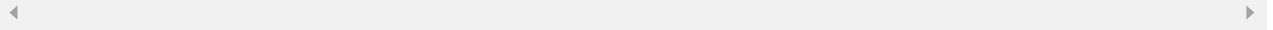
1、：若access\_token未超时，那么进行refresh\_token有两种方式，（1）不会改变access\_token，但超时时间会刷新，相当于续期access\_token（2）更新access\_token的值，我们建议【统一更新access\_token的值】。

2、延期access\_token并不是一个最好的方式，尽管有的开放平台是这么做的。

3、在我们这节课中，我们是提到了这点，本文中的描述是：【用来通过系统重新请求生成一个新的访问令牌】

而且咱们这节课的建议也是更换一个新的访问令牌。

4、“因为我感觉无限续期会增加access被破解的风险”  
刷新令牌也有有效期。



💬 1

👍 2



马成

2020-07-05

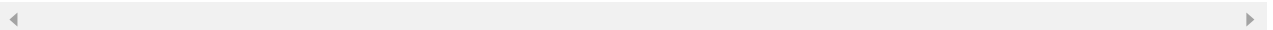
刷新令牌和授权码的处理机制感觉非常相似。只是code是第一次生成的，刷新令牌是后续生成的，功能都是换取访问令牌。我觉得这里还有一个点没说，就是在访问令牌有效期的前半段时间，使用刷新令牌换取的访问令牌是不变的。想问一下设计上可不可使用访问令牌来换取新的访问令牌？

展开 ▾

作者回复: 1.刷新令牌和授权码完全两个东西  
2.刷新令牌换回来的访问令牌一定是变的  
3.不可以，有2所以3不成立

对于第2点，可能是马成看了上节课的留言回复，当时有不准确的描述，我已修改，最好的方案是：更新访问令牌的值，这也是咱们这节课给出的建议【用来通过系统重新请求生成一个新的访问令牌】。

“若access\_token未超时，那么进行refresh\_token有两种方式，（1）不改变access\_token，但超时时间会刷新，相当于续期access\_token（2）更新access\_token的值，我们建议【统一更新access\_token的值】”



💬 1

👍 1



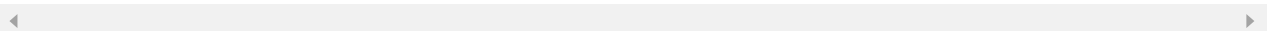
AA

2020-07-04

淘宝的refresh\_token也有过期时间，通过refresh\_token刷新后，返回来accessToken和refresh\_token，但refresh\_token过期时间不会重新刷新，这是为什么要这样设置呢，当refresh\_token为0时，是不是只能通过重新登录授权

展开 ▾

作者回复: 是的，当刷新令牌也过期了，只能重新登录再授权。



💬 1

👍 1

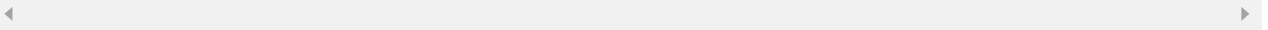
**稻草人**

2020-07-22

老师好，刷新令牌生成新的令牌，这个新生成的令牌有没有特定的规则，或者说算法，还是只是一个随机数，当我调用刷新令牌接口的时候是不是需要做一下权限的验证，不是谁都可以随便调用刷新令牌的接口的吧？期待老师回复

展开 ∨

作者回复: OAuth 2.0 内部并没有规定我们令牌使用什么样的算法生成，我们可以是一个随机数，只要符合唯一性、不可猜测性就可以。在使用刷新令牌的时候，也是需要应用传递它的app\_id和app\_secret的。

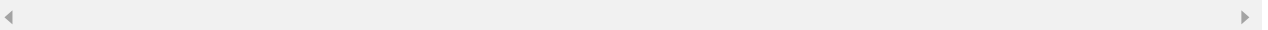
**稻草人**

2020-07-22

老师好，refresh\_token 令牌 获取新的令牌，只需要调用刷新令牌接口就可以了么，不需要再次校验appkey和密钥了吗，不是谁都可以调的吧

展开 ∨

作者回复: 需要再次校验的。

**慎独明强**

2020-07-19

音频时长20分钟,自己开了1.25倍速,再加上自己的整理笔记,加回答问题加看下方评论,一节课差不多要一个多小时左右,再加上后面的温故知新

展开 ∨

**慎独明强**

2020-07-19

刷新令牌是有过期时间的,相当于是重新生成访问令牌的备份,过期时间应该一般和访问令牌的过期时间是一致的.相当于刷新令牌给访问令牌续期,那么在什么场景给访问令牌续期呢,比如说访问令牌的有效期为一天,超过一天就需要重新用户登录与授权.如果刷新令牌没有过期时间,感觉只要授权一次,就会一直有效了.那么安全性不是很好.再想到自己生活中的app,淘宝可能是一个星期一个月,才需要重新输密码登录,银行的app,我退出去一次,就会需要重...

展开 ∨

**Geek\_6a58c7**

2020-07-14

老师你好，有一个问题请教一下，授权码最终还是转成access\_token并在网络中传输，这样从安全角度来说还是还是不安全的，这和授权服务直接返回access\_token区别是什么，用授权码主要是解决什么问题呢？

作者回复: OAuth 2.0 最初的场景就是发生在Web环境下，你可以看咱们的02课程里面，我们有提到。

**piboye**

2020-07-14

appsecret不应该是直接判断相等吧，appsecret不会出现在包中吧，他只是做hash验证吧

作者回复: 可以直接判断，也可以将其作为【因子】加入到签名算法中。

**名:海东**

2020-07-13

刷新令牌应该也是由过期时间的，不然可能存在安全问题。有个问题，需要请教老师，第三方软件(如小兔打单软件)注册到授权服务时是怎样的？它注册时有没有申请权限一说？应该没有用户(如小明)参与吧？第三方软件在用户授权时应该提前在授权服务做了注册。

展开 ∨

作者回复: 小兔打单软件应用对应的研发人员需要事先在授权服务一侧注册好，生成app\_id和app\_secret等信息，注册的时候就要申请一次权限，比如你这个应用是什么类型的，能访问哪些API，注册的时候没有小明的参与。

**唐朝农民**

2020-07-12

```
String appStr = refreshTokenMap.get("refresh_token");
if(!appStr.startsWith(appId+"|"+"USERTEST")){//该refresh_token值 不是颁发给该 第
三方软件的
return;
```

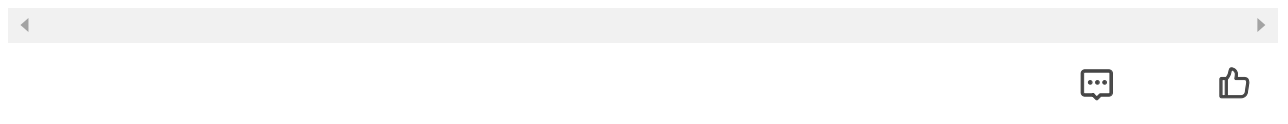
}...

展开 ▾

作者回复: 在实际生产环境中，这个用户ID是不需要传递的也不可能被传递，因为OAuth 2.0的目的之一就是不让第三方软件接触到用户ID。

第一次跟用户ID有关系的时候就是在用户给第三方软件授权的时候，这个时候如果用户没有登录就会先登录再授权，这一切都是发生在平台的一方，所以平台能够拿到用户的ID，继而在给第三方软件生成访问令牌的时候，就可以让这个访问令牌access\_token的值跟第三方软件的app\_id和用户ID做一个映射关系，同理refresh\_token的生成也是一样的，并且refresh\_token和access\_token是一起生成并返回给第三方软件。

我们在代码中为了演示refresh\_token和access\_token的生成都是在app\_id和用户这个粒度的，是“特意”固定写死了一个USERTEST值。

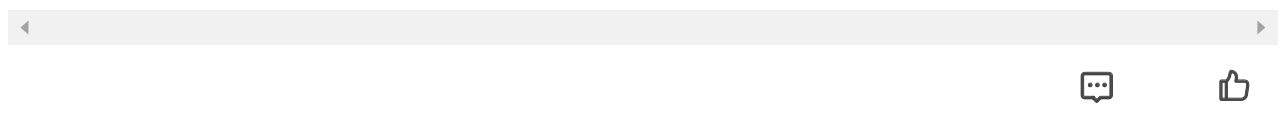
**hom**

2020-07-10

请教下两个问题：

- 1、不断用刷新令牌一直刷新，那token是不是可以无限延长？
- 2、code在流程中被劫持了怎么办？

作者回复: 1、刷新令牌也有有效期  
2、code在前端传输，还有在后端传输的app\_secret

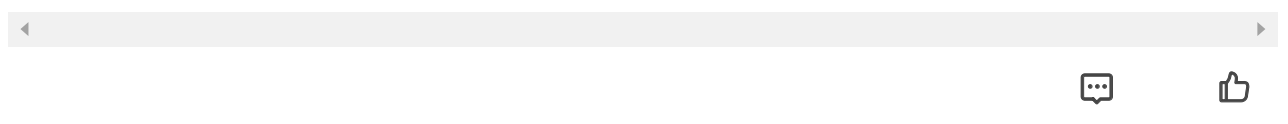
**chuck**

2020-07-09

权限范围如何验证？

展开 ▾

作者回复: 第三方应用在申请成为平台的开发者的时候会注册申请一个所能调用的API权限列表，在用户给第三方应用授权的时候也会让用户来确认给三方软件授予哪些权限，在三方软件请求受保护资源的时候，受保护资源要根据以上去做判断，不能超过当初申请注册的权限范围，也不能超过用户授予的权限范围。



**chuck**

2020-07-09

授权页面是要存储在授权服务的后端？有没有可能存在前端（项目前后端分离的）

作者回复: 这个没有要求，看授权服务如何实现自己的系统结构，只要是授权服务一侧的“资产”就可以。

**Blue**

2020-07-08

老师，github的项目克隆不下来

展开 ∨

**小瞿同学**

2020-07-07

还是没弄懂AT和RT的主要差别 我觉得AT如果因为泄漏导致不安全的话 那RT也有同样的问题啊

作者回复: 要看AT和RT的差别，不能从安全的角度，要从功能的角度，没有RT且AT失效的情况下，对用户很不友好，因为RT的时间很短，就需要用户频繁授权。

**Jxin**

2020-07-07

问题

1.为什么不直接用访问令牌当刷新令牌用？

展开 ∨

