

43 | 实战（三）：“画图”程序后端实战

2019-09-20 许式伟

许式伟的架构课

进入课程 >



讲述：姚迪迈

时长 09:41 大小 8.88M



你好，我是七牛云许式伟。

在上一章，我们实现了一个 mock 版本的服务端，代码如下：

<https://github.com/qiniu/qpaint/tree/v31/paintdom>

我们这一章实战的目标，是要把它改造成一个产品级的服务端程序。

前面两讲，我们一讲谈了 RESTful API 层，一讲谈了底层是业务逻辑的实现层。今天，我们要谈的是帐号（Account）与认证（Authorization）。

我们之前实现的 mock 版本服务端是匿名可访问的，不需要授权。在上一讲，我们开始引入了多租户，但为了简化，用的是一种 mock 的认证方式。

接下来我们就要动真格了。

但在此之前，我们仍然要先理解一下帐号和认证这两个概念。虽然这是两个大家非常耳熟能详的东西。

帐号 (Account)

帐号，简单说就是某种表征用户身份的实体，它代表了一个“用户”。虽然一个物理的自然人用户可能会在同一个网站开多个帐号，但从业务角度，我们往往把这些帐号看作不同的用户。

互联网帐号的表征方式有很多，比较常见的有：

电子邮件；

手机号；

用户自定义的网络 ID；

自动分配的唯一 ID。

前三者大家容易理解。对于自动分配的 UUID，其实最典型的是银行。你的银行帐号从来都不是你自己定义的，而是预先分配好的一个卡号。

当然还有一些冷门的选择。比如有的网站选择用身份证号作为帐号 ID，这通常发生在政府公共服务类的业务。

授权 (Authorization)

那么授权是什么？授权是帐号对服务的访问方式。

从这句话字面去理解，授权和帐号相关。有帐号，就会有授权。但是帐号和授权并不是对应的关系。同一个帐号，可能会有多种授权。

常见的授权机制有哪些？

前面我们在 [“40 | 服务端的业务架构建议”](#) 这一讲提过，当前主流的授权方式有两种：一种是基于 Token，一种是基于 AK/SK。

但实际上还有一种最常见的授权机制没有被提到，那就是：用户名 + 密码。

这里的“用户名”其实就是指“帐号”。

当然，没有提的原因是因为当时我们是在讨论网络 API 协议的授权机制选择。我们在业界基本上看不到用“用户名 + 密码”来作为网络 API 的授权机制。

为什么不用？因为不安全。假如在每一次 API 请求中都带上密码，那么显然密码泄漏的概率会更大。

所以，安全性上的需求会导致我们倾向于尽可能减少密码在网络中传输的次数。“用户名 + 密码”这种授权方式，必然会以尽可能少的频率去使用。

哪些情况会用“用户名 + 密码”授权？

其一，登录 (login)。对于一个 Web 应用而言，授权的第一步是登录。登录最经典的方式就是“用户名 + 密码”授权。

“用户名 + 密码”授权往往只发生在登录那一下，登录后就会生成一个会话 (Session) 用途的 Cookie。此后 Web 应用的授权都基于 Session，直到 Session 过期。

抱歉，我们的词汇有点贫乏。这里说的 Session 授权，和浏览器引入的 Session 不是一回事。Session 授权发生在登录之后，一般并不会随浏览器窗口的关闭而消失，往往有几天的有效期。

甚至有一些网站的 Session 有效期会自动顺延。也就是说，只要你在会话期内活跃的话，Session 授权就不会过期。超时时间从你最后一次活动算起，只有你连续几天都不活跃才会导致 Session 过期。

其二，作为 Token 授权的入口。其实 RESTful API 层中的 Token 授权，和 Web 应用中的 Session 授权的地位是非常像的。

Session 授权会有过期时间，Token 授权也会有过期时间。Session 授权有自动顺延，Token 授权有 Refresh。Session 授权的典型入口是登录（login），Token 授权也一样有“用户名 + 密码”授权这个入口。

这样来看，Token 授权和 Session 授权的差别只是应用场景不同，一个用于 API 层，一个用于 Web。而这也导致承载它们的机制有些不同，Token 授权基于 HTTP 的 Authorization 头，而 Session 授权则基于 Cookie。

OAuth 2.0

由于 QPaint 程序是一个 To C 的应用，所以在 API 层的授权机制选择上，我们很自然会选择 Token 授权。

当前推荐的 Token 授权标准是 OAuth 2.0，它得到了广泛的支持，如果你在使用各类 C 端应用程序的开放接口，会发现它们往往都是基于 OAuth 2.0 的。

有两种场景下我们会考虑 OAuth 2.0。

第一种场景，也是 OAuth 的核心场景，就是提供开放接口。

对于一个服务提供方来说，通过推广自己的 App，来让更多用户使用自己的服务是一个常规的办法。但还有一个非常值得考虑的方式，就是把服务以 API 方式开放出来，让更多的 App 接入自己的服务。

一旦我们希望授权第三方应用程序来调用我们的服务，最好的选择是 OAuth 2.0。

第二种场景，是作为 OpenID 提供方。也就是说，第三方应用接入我的 OAuth 接口，并不是为了要调用我的什么能力，而只是为了复用我的用户。

这当然不是谁都能够做得到的，还是要有足够大的用户基数，并且有一定的入口价值才有可能被接受。国内被广泛使用的典型 OpenID 提供方有：

微信和 QQ；

支付宝；

新浪微博。

为了支持 OAuth 2.0 作为 OpenID 的场景，OpenID Foundation 还专门引入了 OpenID Connect 协议规范。详细资料如下：

<https://openid.net/connect/>

今天我们重点还是关注 OAuth 2.0 的核心场景。它涉及到以下三个角色：

服务提供商。包括授权服务（Authorization Server）和资源服务（Resource Server）。

终端用户，也就是资源拥有方（Resource Owner）。终端用户是服务提供商的用户，它的资源也存在于服务提供商提供的服务中。但是这些资源的归属是属于终端用户的，所以我们称之为资源拥有方。

第三方应用，也就是客户端（Client）。在 OAuth 的视角中，官方应用和第三方应用并无大的区别，以相同的机制在工作。从这一点来说，称之为客户端会更加合理。

这三个角色交互的基本场景是：

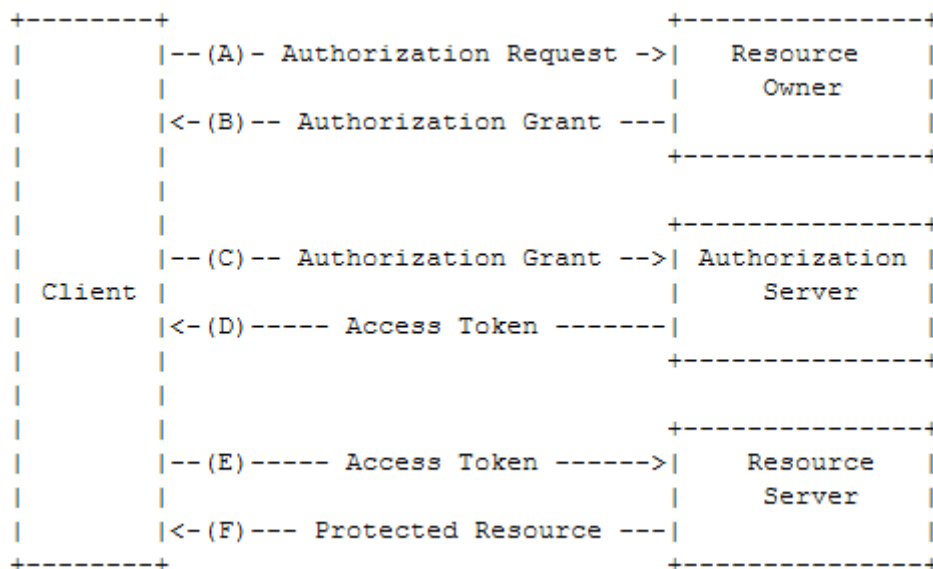
首先，第三方应用，也就是客户端（Client），向服务提供商提出接入申请。这一步可以理解为类似把 App 注册到应用商店的过程，每个应用只需要做一次。

然后，客户端（Client）向终端用户，也就是资源拥有方（Resource Owner）申请访问权限。这个申请发生在服务提供商提供的环境中，所以服务提供商可以感知资源拥有方是拒绝，还是接受了客户端的请求。

然后，客户端（Client）向服务提供商的授权服务（Authorization Server）发起授权请求，并得到了可用于访问资源的 Token。

最后，客户端（Client）通过 Token 向服务提供商的资源服务（Resource Server）发起资源访问请求。

整个过程的具体流程如下：



- (A) 终端用户打开客户端以后，客户端要求终端用户给予授权。
- (B) 终端用户同意给予客户端授权。
- (C) 客户端使用上一步获得的授权，向认证服务器申请令牌（Token）。
- (D) 认证服务器对客户端进行认证以后，确认无误，同意发放令牌。
- (E) 客户端使用令牌，向资源服务器申请获取资源。
- (F) 资源服务器确认令牌无误，同意向客户端开放资源。

这个图体现了 OAuth 2.0 的核心思想。但不同场景下，具体的授权流程有一定的差异。常见的授权模式有如下几种：

授权码模式（Authorization Code）；

简化模式（Implicit）；

用户名 + 密码模式（Resource Owner Password Credentials）；

客户端模式（Client Credentials）；

访问令牌（Access Token）；

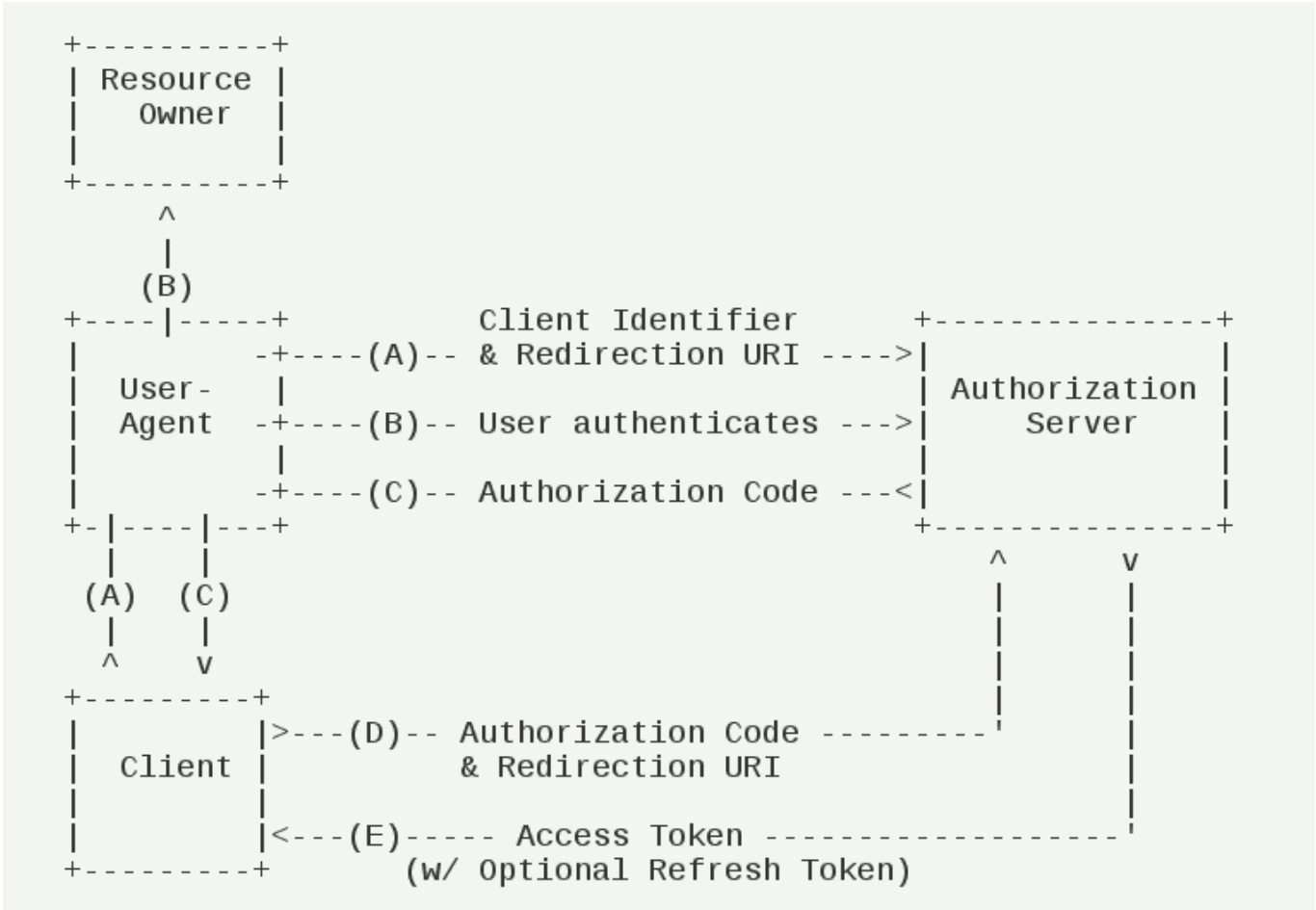
更新令牌（Refresh Token）。

其中，基于访问令牌（Access Token）的授权模式是最核心的一种，请求频率最大。更新令牌（Refresh Token）则次之。每次访问令牌（Access Token）失效后，通过更新令牌（Refresh Token）获得新的访问令牌（Access Token）。

其他所有的授权方式，是在不同场景下的授权入口。通过这些授权入口的任何一个都可以同时获得访问令牌（Access Token）和更新令牌（Refresh Token）。

用户名 + 密码模式（Resource Owner Password Credentials）不用过多解释，这是我们最为熟悉的一种授权方式。

我们重点解释下授权码模式（Authorization Code），这是 OAuth 作为第三方开放接口用的最多的一种场景。它的业务流程如下：



(A) 终端用户访问某个网站，客户端（通常是一个标准的浏览器）将终端用户重定向到认证服务。

(B) 终端用户选择是否给予该网站相应的授权。

(C) 如果授权，认证服务器将用户导向网站事先指定好的“重定向 URI”（Redirection URI），同时附上一个授权码。

(D) 该网站收到授权码，附上早先的“重定向 URI”，向认证服务器申请令牌。这一步是在网站的后端服务器上完成的，对终端用户不可见。

(E) 认证服务器核对了授权码和重定向 URI，确认无误后，网站的后端服务器返回访问令牌（access token）和更新令牌（refresh token）。

此后，该网站就可以通过后端服务器去访问相应的服务了。

结语

今天我们主要聊了帐号与授权相关的基础体系，重点介绍 OAuth 2.0 背后的逻辑。下一讲我们会讨论如何基于 OAuth 来完成 QPaint 的帐号与授权机制。

如果你对今天的内容有什么思考与解读，欢迎给我留言，我们一起讨论。下一讲开始我们继续实战。

如果你觉得有所收获，也欢迎把文章分享给你的朋友。感谢你的收听，我们下期再见。



许式伟的架构课

从源头出发, 带你重新理解架构设计

许式伟
七牛云 CEO



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 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 42 | 实战（二）：“画图”程序后端实战

精选留言 (2)

写留言



Aaron Cheung

2019-09-20

早起打卡 43 对前后端分离式的企业式类似ldap认证一直有点困惑 看看许老师的后续课程



humor

2019-09-20

授权码模式中的第C步已经将用户导向到了重定向url，为什么第D步还要附上重定向url呢？第D步附上的这个重定向url的作用是什么呢？

展开 ∨

作者回复: 用来验证

