文章编号：2096-1472(2021)-04-34-05
DOI:10.19644/j.cnki.issn2096-1472.2021.04.008

基于令牌认证方案的改进研究与实践

胡献宇

(大连东软信息学院，辽宁 大连 116023)
✉huxianyu@neusoft.edu.cn

摘　要：就前后端分离的软件开发模式而言，保护后端数据接口不被非法调用是十分重要的。令牌作为获取保护资源的凭证，需要提供过期时间，否则认证功能就失去了意义。针对活跃用户，需要在有效时间内提供自动登录功能，可以提升使用体验。本文研究了OAuth(Open Authorization，一种开放的授权标准)的认证机制，并在ASP.NET Web API框架基础上，实现身份认证方案，当访问令牌过期后，增加令牌刷新机制，既能够改善用户体验，也能够有效保护数据接口。该方案具有通用性，适用于前后端分离的软件开发。通过测试，表明了该方案具有有效性和可行性。

关键词：OAuth；身份验证和授权；前后端分离；刷新
中图分类号：TP311.11　**文献标识码**：A

Research and Practice of Improving Token Authentication Scheme

HU Xianyu

(*Dalian Neusoft University of Information*, *Dalian* 116023, *China*)
✉huxianyu@neusoft.edu.cn

Abstract: For software development model that separates front and back ends, it is very important to protect back-end data interface from being illegally invoked. The token, as a voucher for obtaining protected resources, needs to provide an expiration time, otherwise authentication function will lose its meaning. For active users, automatic login function needs to be provided within valid time to improve user experience. This paper proposes to implement an identity authentication scheme based on the ASP.NET Web API framework after studying an authentication mechanism of OAuth (Open Authorization, an open authorization standard). When the access token expires, token refresh mechanism is added, which can not only improve user experience, but also effectively protect data interfaces. The proposed scheme is versatile and suitable for software development with separation of front and back ends. Tests show its effectiveness and feasibility.

Keywords: OAuth; authentication and authorization; separation of front and back ends; refresh

1 引言(Introduction)

随着Web应用的快速发展，前后端分离的开发方式成为主流趋势。图1是一种主流的前后端分离的交互模型，使用前按需设计数据接口，后端(主要是指服务器端)可以为前端(主要是指PC端平台、APP及各类小程序)提供HTTP(Hyper Text Transfer Protocol，超文本传输协议)服务，而不需要关注业务的详细展示[1]。此外，前端主要负责接收和组织展示后端传过来的数据，根据具体的业务进行页面逻辑的设计。这种前后端交互模型结构简洁，方便前后端开发者关注自己的业务，降低了前后端的耦合性，可成功解耦，为后续开发工作打下了坚实的基础。

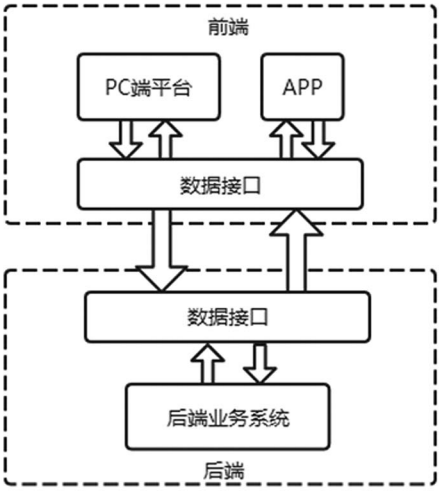

图1 前后端交互模型图

Fig.1 Front-end interaction model diagram

然而，使用前后端分离的软件开发模式，需要关注后端数据接口的安全性，保护接口不被非法调用。针对这种情况，本文实现了基于OAuth(Open Authorization,一种开放的授权标准)2.0的身份验证和授权方案，一方面能够保护接口的安全性；另一方面，能够在令牌过期之后实现刷新机制，改善用户体验，并通过Postman工具进行完整测试，表明方案的可行性。

2 身份验证与授权(Authentication and authorization)

2.1 身份验证

身份验证是验证一个系统实体或系统资源具有某种属性值的过程。对于HTTP而言，客户端和服务器端是两个独立的系统实体，因此，需要从两个方面去考虑。一方面，若想实现服务器端的身份验证，就得需要向客户端保证，请求消息只会发送给正确的源服务器，在这种情况下，消息的发送者需要在发送消息之前核实消息的接收者，通常是认证传输连接的另一端。另一方面，若想实现客户端的身份验证，就得需要客户端向服务器端发送验证请求，即判断请求消息是否应该得到授权[2]。常用的身份验证包含基于密码的HTTP基础身份验证和基于令牌的身份验证。

2.1.1 基于密码的HTTP基础身份验证

基于密码的HTTP基础身份验证，就是将用户名和密码信息进行加密处理，生成Ticket(票据)，客户端请求的时候需要校验Ticket[3],主要步骤如下：

(1)使用冒号作为分隔符，连接用户名和对应的密码，使用Base64对连接字符串进行加密处理。

(2)将加密之后的字符串放在Authorization标头的方案标识符Basic之后。

(3)若通过后台校验，客户端将会接收Ticket，并保存为验证信息，之后的每一次请求都需要携带，以方便服务器进行验证。验证成功，则获取相关信息，否则返回未通过的提示信息。

2.1.2 基于令牌的身份验证

基于令牌的身份验证，就是使用安全令牌进行验证。安全令牌是指用于在身份验证过程中验证用户标识的数据对象[4]。我们在Web应用程序中使用的身份验证Cookie，就是基于令牌的身份验证，其流程如下：

(1)使用密码进行初始认证操作，之后会生成一个Cookie，将其返回给客户端。

(2)客户端之后发出的每个请求，都通过这个Cookie进行身份验证，不再需要初始的身份信息。

2.1.3 身份验证小结

基于密码的HTTP基础身份验证方法存在如下一些问题：

(1)客户端必须保存密码，同时，由于密码采用明文保存，或使用一种可逆的保存方式，增加了密码泄露的风险。

(2)服务器对每个请求都进行密码验证，会增加很多开销，影响效率。

(3)为防御字典攻击而使用相关技术，会导致验证过程计算开销较大。

(4)不适合分布式应用。分布式系统中，身份验证通常委托给外部系统。

基于如上考虑，本方案将使用基于令牌的认证方案。

2.2 授权

2.2.1 授权简介

授权，可以理解为控制主体对受保护资源进行相应的操作[5]。通常来说，授权问题的核心主要是主体、操作和资源三者的关系，即主体是否可以操作资源[6]，如图2所示。

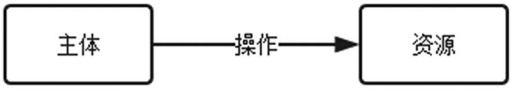

图2 基础授权模式

Fig.2 Basic authorization model

在Web API中，对于授权我们可以有如下理解：

(1)资源，指HTTP资源，而HTTP资源是请求消息要访问的资源。

(2)操作，指HTTP方法，包括GET请求、POST请求等。

(3)主体，指执行HTTP请求的HTTP客户端。

2.2.2 OAuth 2.0简介

OAuth 2.0是OAuth 1.0协议的升级版本[6]，它的主要目标是让客户端的开发工作变得简单与便捷，为Web应用程序、手机APP、桌面软件等提供特定的授权方式。OAuth 2.0允许第三方应用程序获得HTTP服务的访问权限，同时，可以通过HTTP服务提供方授权资源拥有者，允许第三方应用程序凭借自己的客户端凭证获取在资源服务器上的受保护资源的访问权限。

在OAuth 2.0中，定义了授权码模式、简化模式、密码模式和客户端模式四种授权方式[7]，综合考虑本方案的需求，决定采用密码模式。密码模式的流程如图3所示。

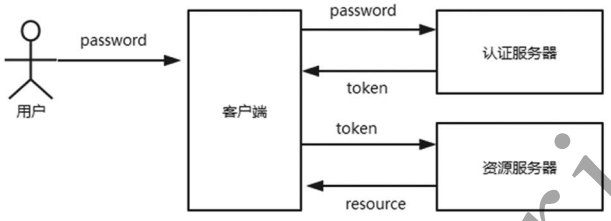

图3 OAuth 2.0的密码模式授权图

Fig.3 OAuth 2.0 password mode authorization diagram

这种模式的流程如下：

(1)用户通过提供账号和密码来使用客户端。

(2)客户端将账号和密码发送给认证服务器，即请求token。

(3)通过认证服务器验证，将会向客户端提供访问token。

(4)客户端使用token向资源服务器请求资源。

3 身份验证和授权方案的设计与实现(Design and implementation of identity verification and authorization scheme)

3.1 身份验证和授权方案的设计

在ASP.NET Web API框架上，使用微软开发的基于OWIN规范的Katana项目中的中间件，能够很方便地完成基于OAuth 2.0的身份验证和授权。虽然OAuth 2.0关注的主要内容是授权，但是在该中间件中OAuth 2.0可以提供基于令牌的身份验证功能[8]，其处理访问令牌的流程如图4所示。

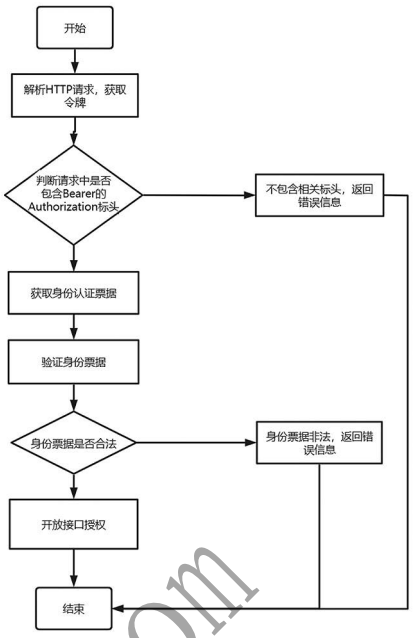

图4 身份验证和授权流程图

Fig.4 Authentication and authorization flow chart

(1)获取令牌。如果HTTP请求存在Bearer的Authorization标头，那么将该标头作为访问令牌，否则身份验证方法不返回任何身份信息。

(2)获取身份认证票据。解析访问令牌，可以获取身份验证票据信息以及属性信息。

(3)验证身份票据，并检查其有效性。

3.2 身份验证和授权方案的实现

(1)使用Postman发送仿真模拟请求，并尝试使用GET方式访问已有的数据接口，其结果如图5所示。此时，我们是在没有限制的情况下获取相关数据信息。对于我们的方法，如果不加以限制，那么任何客户端都可无条件访问我们的接口获取数据，这样是不安全的，必须对接口进行限制，只有通过验证的用户才可以获取数据信息。

图5 GET请求获取数据

Fig.5 GET request to get data

(2)在项目中，新建一个MyAuthorizationServerProvider类，用于实现身份验证，并且重写ValidateClientAuthentication方法和GrantResourceOwnerCredentials方法。

关键代码如下：

```
public class MyAuthorizationServerProvider ：
OAuthAuthorizationServerProvider
{
    public override Task ValidateClientAuthentication(
    OAuthValidateClientAuthenticationContext context)
    {
        context.Validated();
        return Task.FromResult<object>(null);
    }
    public override async Task GrantResourceOwner
Credentials(
    OAuthGrantResourceOwnerCredentialsContext
context)
    {
        context.OwinContext.Response.Headers.
        Add("Access-Control-Allow-Origin", new[] { "*" });
        AccountService accService = new
AccountService();
        string md5Pwd = LoginHelperMD5CryptoPasswd
        (context.Password);
        IList<object[]> ul = accService.Login(context.
UserName, md5Pwd);
        if (ul.Count() == 0)
        {
        context.SetError("授权无效", "用户名或密码错误！");
        return;
        }
        var identity = new
ClaimsIdentity(context.Options.AuthenticationType);
        context.Validated(identity);
    }
}
```

为上述数据接口添加Authorize标签，达到过滤的目的，进而实现对接口的保护。此时，我们再次使用Postman进行仿真请求，尝试请求接口，返回信息如图6所示，即显示已拒绝为该请求授权。

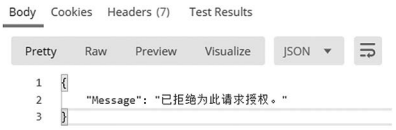

图6 GET请求受限

Fig.6 GET request limited

(3)当前接口已经被保护了，我们在访问接口的时候，就得通过Access_token。因此，我们必须拿到Access_token的值，其获取方法如下：发送POST请求，访问token的URL；发送请求之前，在Body内添加grant_type、username、password这三个参数，由于我们采用密码模式进行授权，因此，grant_type的参数是password，如图7所示，我们发送一个POST请求之后，响应的信息包含access_token、token_type等值，响应详情如图8所示。

图7 POST请求获取token

Fig.7 POST request to get token

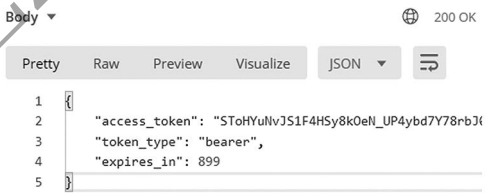

图8 获取token相关返回值

Fig.8 Get the return value of the token

(4)我们重新访问受限的接口，在发送GET请求的时候，如图9所示。在请求的标头里将Authorization设置为KEY，此时需要使用上一步获取到的Access_token，将其作为VALUE参数的一部分。最终，VALUE对应的值应该为Bearer+access_token，如图9所示，我们能够顺利获取到数据信息。这种情况下，15分钟内客户端在访问服务器端的时候，无须进行验证就可使用该token访问受保护的资源。

图9 封装请求头进行GET请求

Fig.9 Encapsulate request headers for GET requests

(5)当Access_token的值失效时，我们就需要重新获取新的令牌值。此时，我们需要在项目目录中新建一个MyRefreshTokenProvider类，用于令牌到期之前进行刷新，以便获取新的令牌，并重写Create方法和Receive方法，代码

如下：

```
public class MyRefreshTokenProvider ： AuthenticationTokenProvider
{
    private static ConcurrentDictionary<string, string> _refreshTokens= new ConcurrentDictionary<string, string>();
    //用于生成refresh_token
    public override void Create(AuthenticationTokenCreateContext context)
    {
        context.Ticket.Properties.IssuedUtc = DateTime.UtcNow；
        context.Ticket.Properties.ExpiresUtc = DateTime.UtcNow.AddDays(60)；
        context.SetToken(Guid.NewGuid().ToString("n"))；
        _refreshTokens[context.Token] = context.SerializeTicket()；
    }
    //由 refresh_token 解析成 access_token
    public override void Receive(AuthenticationTokenReceiveContext context)
    {
        string value；
        if (_refreshTokens.TryRemove(context.Token, out value))
        {
            context.DeserializeTicket(value)；
        }
    }
}
```

(6)修改好相关代码之后，我们需要按照第(4)步的方式重新获取Token，因为这次获取数据的时候，会额外获取到Refresh_token的值，如图10所示，该值将用于重新获取新的令牌，而不需要重新使用密码的授权方式获取。

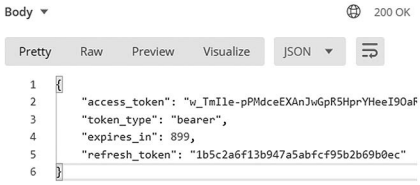

图10 GET请求获取refresh_token

Fig.10 GET request to get refresh_token

(7)当Token过期后，我们需要使用上次得到的Refresh_token重新获取令牌，即发送POST请求访问获取令牌的URL。发送请求的时候，需要在Body内添加grant_type、refresh_token这两个参数，由于本次需要使用刷新令牌的方式获取新的令牌，因此Grant_type的值为Refresh_token，而Refresh_token的值就是我们在第一次获取令牌的时候里面的Refresh_token，如图11所示。

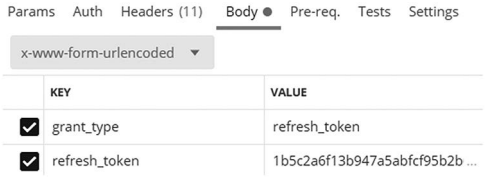

图11 获取refresh_token

Fig.11 Get refresh_token

只要在客户端进行相关设置，即可实现在不影响用户操作体验的情况下动态延长token有效期限，有效地解决了token失效问题。

4 结论(Conclusion)

本文通过研究身份验证和授权的机制，改进了基于令牌的认证方案，增加刷新令牌，能够在访问令牌过期之后，使用刷新令牌实现自动登录和授权，不仅提升了用户体验，还能有效地保护后端数据接口的安全性；并使用Postman进行接口测试，验证了方案的可行性，为前后端分离的身份验证和授权提供了有效解决方案。

参考文献(References)

[1] 程冬梅,王瑞聪,刘燕,等.基于REST架构风格的物联网服务平台研发[J].计算机工程与应用,2012,48(14):74-78.

[2] 刘文元.基于ASP.NET Web API的摘要身份验证改进算法[J].电脑迷,2017(4):56.

[3] GLENN B.ASP.NET Web API设计[M].北京:人民邮电出版社,2015:331-333.

[4] 蒋金楠.ASP.NET Web API2框架揭秘[M].北京:电子工业出版社,2014:591-594.

[5] 王仲洲,杨晓洪,王剑平,等.基于REST风格的WEB API架构研究[J].微处理机,2016,37(5):52-55.

[6] 沈海波,陈强,陈勇昌.基于OAuth 2.0扩展的客户端认证方案[J].计算机工程与设计,2017,38(2):350-354.

[7] 王婷婷,赵松泽.基于OAuth 2.0协议的安全授权模型研究[J].软件工程,2017,20(01):55-59.

[8] DAVIS J, RAJASREE M S. A framework for the description, discovery and composition of RESTful semantic web services[C]. Computational Science, Engineering and Information Technology, 2012:88-93.

作者简介：

胡献宇(1994-)，男，硕士，讲师.研究领域：软件开发.