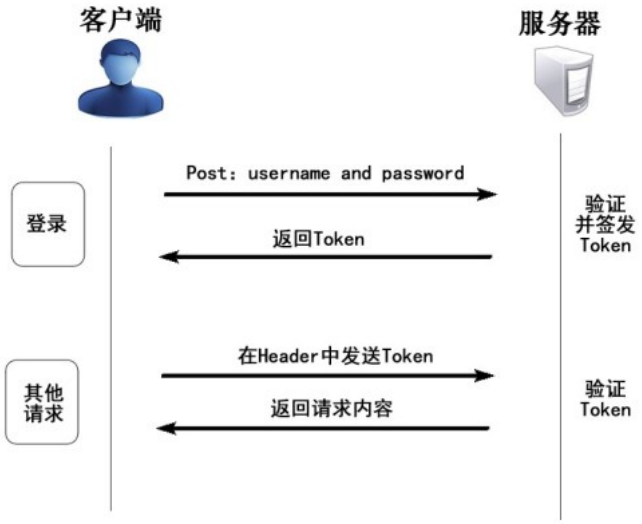
原创

asp.net core 2.0 web api基于JWT自定义策略授权

桂素伟 关注

JWT(json web token)是一种基于json的身份验证机制，流程如下：



通过登录，来获取Token，再在之后每次请求的Header中追加Authorization为Token的凭据，服务端验证通过即可能获取想要访问的资源。关于JWT的技术，可参考网络上文章，这里不作详细说明，

这篇博文，主要说明在asp.net core 2.0中，基于jwt的web api的权限设置，即在asp.net core中怎么用JWT，再次就是不同用户或角色因为权限问题，即使援用Token，也不能访问不该访问的资源。

基本思路是我们自定义一个策略，来验证用户，和验证用户授权，PermissionRequirement是验证传输授权的参数。在Startup的ConfigureServices注入验证（Authentication），授权（Authorization），和JWT(JwtBearer)

自定义策略：

已封闭成AuthorizeRolicy.JWT nuget包，并发布到nuget上：

https://www.nuget.org/packages/AuthorizePolicy.JWT/

源码如下：

JwtToken.cs

```
/// <summary>
/// 获取基于JWT的Token
/// </summary>
/// <param name="username"></param>
/// <returns></returns>
public static dynamic BuildJwtToken(Claim[] claims, PermissionRequirement permissionRequir
{
    var now = DateTime.UtcNow;
    var jwt = new JwtSecurityToken(
        issuer: permissionRequirement.Issuer,
        audience: permissionRequirement.Audience,
        claims: claims,
        notBefore: now,
```

```
            var encodedJwt = new JwtSecurityTokenHandler().WriteToken(jwt);
            var response = new
            {
                Status = true,
                access_token = encodedJwt,
                expires_in = permissionRequirement.Expiration.TotalMilliseconds,
                token_type = "Bearer"
            };
            return response;
        }
```

Permission.cs

```
    /// <summary>
    /// 用户或角色或其他凭据实体
    /// </summary>
    public class Permission
    {
        /// <summary>
        /// 用户或角色或其他凭据名称
        /// </summary>
        public virtual string Name
        { get; set; }
        /// <summary>
        /// 请求Url
        /// </summary>
        public virtual string Url
        { get; set; }
    }
```

PermissionRequirement.cs

```
    /// <summary>
    /// 必要参数类
    /// </summary>
    public class PermissionRequirement : IAuthorizationRequirement
    {
        /// <summary>
        /// 用户权限集合
        /// </summary>
        public List<Permission> Permissions { get; private set; }
        /// <summary>
        /// 无权限action
        /// </summary>
        public string DeniedAction { get; set; }
        /// <summary>
        /// 认证授权类型
        /// </summary>
        public string ClaimType { internal get; set; }
        /// <summary>
        /// 请求路径
        /// </summary>
        public string LoginPath { get; set; } = "/Api/Login";
        /// <summary>
        /// 发行人
        /// </summary>
        public string Issuer { get; set; }
        /// <summary>
        /// 订阅人
        /// </summary>
        public string Audience { get; set; }
        /// <summary>
```

```
        public TimeSpan Expiration { get; set; } = TimeSpan.FromMinutes(5000);
        /// <summary>
        /// 签名验证
        /// </summary>
        public SigningCredentials SigningCredentials { get; set; }
        /// <summary>
        /// 构造
        /// </summary>
        /// <param name="deniedAction">无权限action</param>
        /// <param name="userPermissions">用户权限集合</param>
        /// <summary>
        /// 构造
        /// </summary>
        /// <param name="deniedAction">拒约请求的url</param>
        /// <param name="permissions">权限集合</param>
        /// <param name="claimType">声明类型</param>
        /// <param name="issuer">发行人</param>
        /// <param name="audience">订阅人</param>
        /// <param name="signingCredentials">签名验证实体</param>
        public PermissionRequirement(string deniedAction, List<Permission> permissions, string
        {
            ClaimType = claimType;
            DeniedAction = deniedAction;
            Permissions = permissions;
            Issuer = issuer;
            Audience = audience;
            SigningCredentials = signingCredentials;
        }
    }
```

自定义策略类PermissionHandler.cs

```
    /// <summary>
    /// 权限授权Handler
    /// </summary>
    public class PermissionHandler : AuthorizationHandler<PermissionRequirement>
    {
        /// <summary>
        /// 验证方案提供对象
        /// </summary>
        public IAuthenticationSchemeProvider Schemes { get; set; }
        /// <summary>
        /// 自定义策略参数
        /// </summary>
        public PermissionRequirement Requirement
        { get; set; }
        /// <summary>
        /// 构造
        /// </summary>
        /// <param name="schemes"></param>
        public PermissionHandler(IAuthenticationSchemeProvider schemes)
        {
            Schemes = schemes;
        }
        protected override async Task HandleRequirementAsync(AuthorizationHandlerContext conte
        {
            ////赋值用户权限
            Requirement = requirement;
            //从AuthorizationHandlerContext转成HttpContext，以便取出表求信息
            var httpContext = (context.Resource as Microsoft.AspNetCore.Mvc.Filters.AuthorizationI
            //请求Url
```

```
            foreach (var scheme in await Schemes.GetRequestHandlerSchemesAsync())
            {
                var handler = await handlers.GetHandlerAsync(httpContext, scheme.Name) as IAuthe
                if (handler != null && await handler.HandleRequestAsync())
                {
                    context.Fail();
                    return;
                }
            }
            //判断请求是否拥有凭据，即有没有登录
            var defaultAuthenticate = await Schemes.GetDefaultAuthenticateSchemeAsync();
            if (defaultAuthenticate != null)
            {
                var result = await httpContext.AuthenticateAsync(defaultAuthenticate.Name);
                //result?.Principal不为空即登录成功
                if (result?.Principal != null)
                {
                    httpContext.User = result.Principal;
                    //权限中是否存在请求的url
                    if (Requirement.Permissions.GroupBy(g => g.Url).Where(w => w.Key.ToLower() ==
                    {
                        var name = httpContext.User.Claims.SingleOrDefault(s => s.Type == require
                        //验证权限
                        if (Requirement.Permissions.Where(w => w.Name == name && w.Url.ToLower()
                        {
                            //无权限跳转到拒绝页面
                            httpContext.Response.Redirect(requirement.DeniedAction);
                        }
                    }
                    context.Succeed(requirement);
                    return;
                }
            }
            //判断没有登录时，是否访问登录的url,并且是Post请求，并助是form表单提交类型，否则为失
            if (!questUrl.Equals(Requirement.LoginPath.ToLower(), StringComparison.Ordinal) && (!htt
                || !httpContext.Request.HasFormContentType))
            {
                context.Fail();
                return;
            }
            context.Succeed(requirement);
        }
    }
```

新建asp.net core 2.0的web api项目，并在项目添加AuthorizePolicy.JWT如图

先设置配置文件，用户可以定义密匙和发生人，订阅人

```
  "Audience": {
    "Secret": "ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890",
```

}

在ConfigureServices中注入验证（Authentication），授权（Authorization），和JWT(JwtBearer)

Startup.cs

```
public void ConfigureServices(IServiceCollection services)
{
    //读取配置文件
    var audienceConfig = Configuration.GetSection("Audience");
    var symmetricKeyAsBase64 = audienceConfig["Secret"];
    var keyByteArray = Encoding.ASCII.GetBytes(symmetricKeyAsBase64);
    var signingKey = new SymmetricSecurityKey(keyByteArray);
    var tokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = signingKey,
        ValidateIssuer = true,
        ValidIssuer = audienceConfig["Issuer"],
        ValidateAudience = true,
        ValidAudience = audienceConfig["Audience"],
        ValidateLifetime = true,
        ClockSkew = TimeSpan.Zero
    };
    var signingCredentials = new SigningCredentials(signingKey, SecurityAlgorithms.HmacSh
    services.AddAuthorization(options =>
    {
        //这个集合模拟用户权限表,可从数据库中查询出来
        var permission = new List<Permission> {
                        new Permission {  Url="/", Name="admin"},
                        new Permission {  Url="/api/values", Name="admin"},
                        new Permission {  Url="/", Name="system"},
                        new Permission {  Url="/api/values1", Name="system"}
                    };
        //如果第三个参数，是ClaimTypes.Role，上面集合的每个元素的Name为角色名称，如果(
        var permissionRequirement = new PermissionRequirement("/api/denied", permission,
        options.AddPolicy("Permission",
                    policy => policy.Requirements.Add(permissionRequirement));
    }).AddAuthentication(options =>
    {
        options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
        options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
    })
    .AddJwtBearer(o =>
    {
        //不使用https
        o.RequireHttpsMetadata = false;
        o.TokenValidationParameters = tokenValidationParameters;
    });
    //注入授权Handler
    services.AddSingleton<IAuthorizationHandler, PermissionHandler>();
    services.AddMvc();
}
```

在需要授的Controller上添加授权特性

[Authorize("Permission")]

PermissionController类有两个方法，一个是登录，验证用户名和密码是否正确，如果正确就发放Token，如果失败，验证失败，别一个成功登后的无权限导航action。

```
[Authorize("Permission")]
public class PermissionController : Controller
{
```

```
        PermissionRequirement _requirement;
        public PermissionController(IAuthorizationHandler authorizationHander)
        {
            _requirement = (authorizationHander as PermissionHandler).Requirement;
        }
        [AllowAnonymous]
        [HttpPost("/api/login")]
        public IActionResult Login(string username,string password,string role)
        {
            var isValidated = username == "gsw" && password == "111111";
            if (!isValidated)
            {
                return new JsonResult(new
                {
                    Status = false,
                    Message = "认证失败"
                });
            }
            else
            {
                //如果是基于角色的授权策略，这里要添加用户;如果是基于角色的授权策略，这里要添加
                var claims =new Claim[]{ new Claim(ClaimTypes.Name, username),new Claim(ClaimTypes
                //用户标识
                var identity = new ClaimsIdentity(JwtBearerDefaults.AuthenticationScheme);
                identity.AddClaims(claims);
                //登录
                HttpContext.SignInAsync(JwtBearerDefaults.AuthenticationScheme, new ClaimsPrincip
                var token = JwtToken.BuildJwtToken(claims, _requirement);
                return new JsonResult(token);
            }
        }
        [AllowAnonymous]
        [HttpGet("/api/denied")]
        public IActionResult Denied()
        {
            return new JsonResult(new
            {
                Status = false,
                Message = "你无权限访问"
            });
        }
    }
```

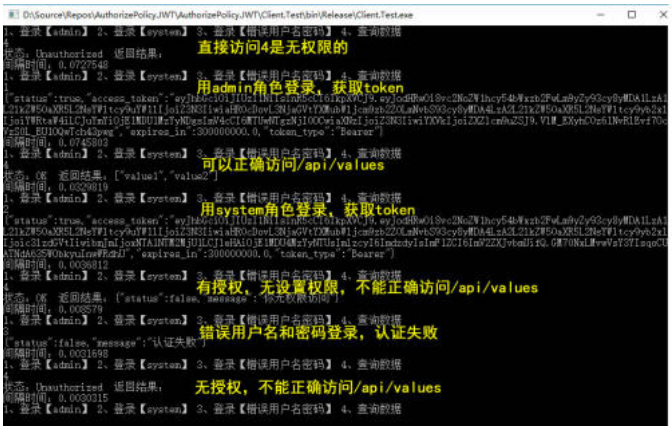
下面定义一个控制台（.NetFramewrok）程序，用RestSharp来访问我们定义的web api，其中1为admin角色登录，2为system角色登录，3为错误用户密码登录，4是一个查询功能，在startup.cs中，admin角色是具有查询/api/values的权限的，所以用admin登录是能正常访问的，用system登录，能成功登录，但没有权限访问/api/values，用户名密码错误，访问/aip/values，直接是没有授权的

```
    class Program
    {
        /// <summary>
        /// 访问Url
        /// </summary>
        static string _url = "http://localhost:39286";
        static void Main(string[] args)
        {
            dynamic token = null;
            while (true)
            {
                Console.WriteLine("1、登录【admin】 2、登录【system】 3、登录【错误用户名密码】
                var mark = Console.ReadLine();
                var stopwatch = new Stopwatch();
```

```
            {
                case "1":
                    token = AdminLogin();
                    break;
                case "2":
                    token = SystemLogin();
                    break;
                case "3":
                    token = NullLogin();
                    break;
                case "4":
                    AdminInvock(token);
                    break;
            }
            stopwatch.Stop();
            TimeSpan timespan = stopwatch.Elapsed;
            Console.WriteLine($"间隔时间：{timespan.TotalSeconds}");
        }
    }
    static dynamic NullLogin()
    {
        var loginClient = new RestClient(_url);
        var loginRequest = new RestRequest("/api/login", Method.POST);
        loginRequest.AddParameter("username", "gswaa");
        loginRequest.AddParameter("password", "111111");
        //或用用户名密码查询对应角色
        loginRequest.AddParameter("role", "system");
        IRestResponse loginResponse = loginClient.Execute(loginRequest);
        var loginContent = loginResponse.Content;
        Console.WriteLine(loginContent);
        return Newtonsoft.Json.JsonConvert.DeserializeObject(loginContent);
    }
    static dynamic SystemLogin()
    {
        var loginClient = new RestClient(_url);
        var loginRequest = new RestRequest("/api/login", Method.POST);
        loginRequest.AddParameter("username", "gsw");
        loginRequest.AddParameter("password", "111111");
        //或用用户名密码查询对应角色
        loginRequest.AddParameter("role", "system");
        IRestResponse loginResponse = loginClient.Execute(loginRequest);
        var loginContent = loginResponse.Content;
        Console.WriteLine(loginContent);
        return Newtonsoft.Json.JsonConvert.DeserializeObject(loginContent);
    }
    static dynamic AdminLogin()
    {
        var loginClient = new RestClient(_url);
        var loginRequest = new RestRequest("/api/login", Method.POST);
        loginRequest.AddParameter("username", "gsw");
        loginRequest.AddParameter("password", "111111");
        //或用用户名密码查询对应角色
        loginRequest.AddParameter("role", "admin");
        IRestResponse loginResponse = loginClient.Execute(loginRequest);
        var loginContent = loginResponse.Content;
        Console.WriteLine(loginContent);
        return Newtonsoft.Json.JsonConvert.DeserializeObject(loginContent);
    }
    static void AdminInvock(dynamic token)
    {
        var client = new RestClient(_url);
        //这里要在获取的令牌字符串前加Bearer
```

```
            IRestResponse response = client.Execute(request);
            var content = response.Content;
            Console.WriteLine($"状态：{response.StatusCode} 返回结果：{content}");
        }
    }
```

运行结果：



源码： https://github.com/axzxs2001/AuthorizePolicy.JWT

web api asp .net core

2 收藏 分享

上一篇：asp.net core策略授权 下一篇：我的友情链接

猜你喜欢

我的友情链接

asp.net core策略授权

C#如何设置Excel文档保护——工作簿、工作表、单元格

在.NET数据库访问方面的Dapper类库介绍

C#/VB.NET 如何添加、获取、删除PDF附件

C#/VB.NET 创建PDF项目符号列表和多级编号列表

C# /VB.NET 操作Word (一)——插入、修改、删除Word…

C# 操作Excel数据透视表