本篇文章介绍如何在ASP.NET上实现OpenID Connect服务器

**前置知识**

1. OAuth2 协议
2. OpenID Connect 协议

这些知识你可以在github目录的的 [\平台无关\web] 目录找到

**框架**

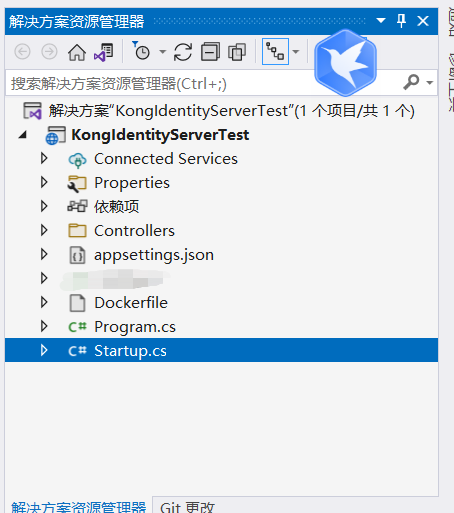
.net 5.0

IdentityServer4包：版本4.1.2 （推荐使用4.x.x版本，不同版本会有细微差别）

**新建API项目**

我们需要新建一个ASP.NET Core API项目，并安装IdentityServer4

我们的项目看起来如下



**API资源，Identity资源，客户端代码**

我们新建一个Config.cs文件，其代码如下

    public class Config

    {

        // 身份资源

        public static IEnumerable<IdentityResource> GetIdentityResources()

        {

            return new List<IdentityResource>

            {

                new IdentityResources.OpenId(),

                new IdentityResources.Profile(),

                new IdentityResources.Email(),

            };

        }

        // API资源

        public static IEnumerable<ApiResource> GetApis()

        {

            return new List<ApiResource>

            {

                new ApiResource("api", "Demo API")

            };

        }

        // 客户端

        public static IEnumerable<Client> GetClients()

        {

            return new List<Client>

            {

                new Client

                {

                    ClientId = "hybrid",

                    ClientSecrets = { new Secret("secret".Sha256()) },

                    // 混合模式

                    AllowedGrantTypes = GrantTypes.Hybrid,

                    RequirePkce = false,

                    // 登录后重定向地址

                    RedirectUris = { "https://notused" },

                    // 推出登录后重定向地址

                    PostLogoutRedirectUris = { "https://notused" },

                    // 允许url的形式传送access token

                    AllowAccessTokensViaBrowser = true,

                    AllowOfflineAccess = true,

                    // 允许访问的域(api资源, Identity资源)

                    AllowedScopes = { "openid", "profile", "email", "api" }

                }

            };

        }

    }

**Startup配置**

我们将API资源，Identity资源，客户端配置到IdentityServer中，并启用IdentityServer

    public class Startup

    {

        public void ConfigureServices(IServiceCollection services)

        {

            services.AddIdentityServer()

                // 使用开发者密钥签名token

                .AddDeveloperSigningCredential()

                // 添加身份资源

                .AddInMemoryIdentityResources(Config.GetIdentityResources())

                // 添加Api资源

                .AddInMemoryApiResources(Config.GetApis())

                // 添加客户端

                .AddInMemoryClients(Config.GetClients());

            // 添加权限

            services.AddAuthentication();

            ...

        }

        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

        {

            // 启用IdentityServer

            app.UseIdentityServer();

            app.UseRouting();

            // 启用权限

            app.UseAuthorization();

            ....

        }

    }

**登录与退出登录方法**

我们新建一个控制器，执行登录与退出登录

    [ApiController]

    [Route("[controller]/[action]")]

    public class AccountController : ControllerBase

    {

        private readonly IIdentityServerInteractionService \_interaction;

        private readonly IEventService \_events;

        public AccountController(

            IIdentityServerInteractionService interaction,

            IEventService events)

        {

            \_interaction = interaction;

            \_events = events;

        }

        /// <summary>

        /// 登录页面

        /// </summary>

        /// <returns></returns>

        [HttpGet]

        public async Task<IActionResult> Login() {

            // 在这个Action返回登录视图

            // 由于笔者好久没有写 Razor 页面了，所以偷懒直接返回null，如果使用前后分类，则可以不用这个Action

            return Ok(null);

        }

        /// <summary>

        /// 登录

        /// </summary>

        [HttpPost]

        public async Task<IActionResult> Login(LoginInputModel model)

        {

            // 根据返回地址获取授权请求

            var context = await \_interaction.GetAuthorizationContextAsync(model.ReturnUrl);

            // 验证用户名密码

            if(!(model.Username == "test" && model.Password == "123456"))

            {

                // 向 IdentityServer 发送登录失败事件

                await \_events.RaiseAsync(new UserLoginFailureEvent(model.Username, "invalid credentials", clientId: context?.Client.ClientId));

                return Ok(new {

                    success = false,

                    message = "用户名密码错误"

                });

            }

            // 向 IdentityServer 发送用户登录事件

            await \_events.RaiseAsync(new UserLoginSuccessEvent(model.Username, model.Username, model.Username, clientId: context?.Client.ClientId));

            AuthenticationProperties props = new AuthenticationProperties

            {

                IsPersistent = true,

                ExpiresUtc = DateTimeOffset.UtcNow.Add(new TimeSpan(365, 0, 0, 0))

            };

            // issue authentication cookie with subject ID and username

            var isuser = new IdentityServerUser(model.Username)

            {

                DisplayName = model.Username

            };

            // 执行 IdentityServer 登录

            await HttpContext.SignInAsync(isuser, props);

            return Ok(null);

        }

        /// <summary>

        /// 退出登录

        /// </summary>

        [HttpPost]

        public async Task<IActionResult> Logout()

        {

            // 如果用户已在认证服务器登录

            if (User?.Identity.IsAuthenticated == true)

            {

                // 退出登录

                await HttpContext.SignOutAsync();

                // 发送退出登录事件

                await \_events.RaiseAsync(new UserLogoutSuccessEvent(User.GetSubjectId(), User.GetDisplayName()));

            }

            return Ok(null);

        }

    }

所用到的LoginInputModel如下

    public class LoginInputModel

    {

        [Required]

        public string Username { get; set; }

        [Required]

        public string Password { get; set; }

        public bool RememberLogin { get; set; }

        public string ReturnUrl { get; set; }

    }

至此，我们可以验证OIDC的部署情况了

**Postman测试**

1. 登录跳转

1）登录跳转URL

http://localhost:5000/connect/authorize?client\_id=hybrid&scope=openid profile&response\_type=code id\_token token&redirect\_uri=https://notused&state=abc&nonce=xyz

client\_id：客户端ID

scope：想要访问的域

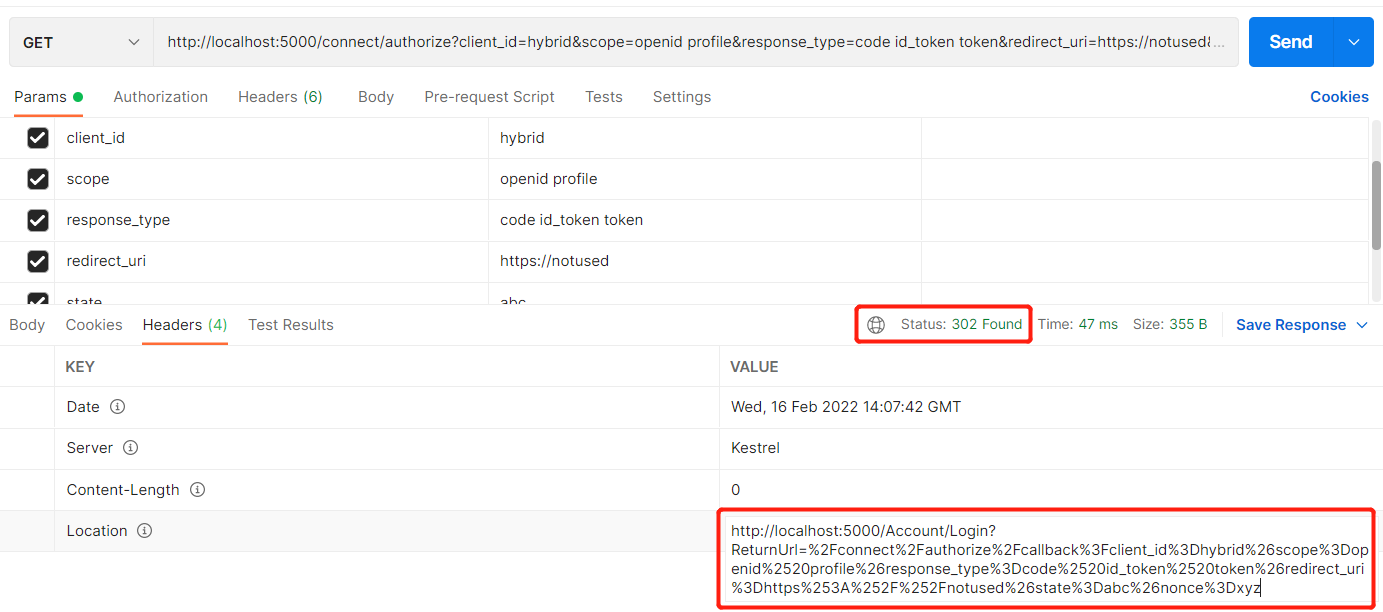
response\_type：登录成功后将返回code id\_token token，并非随意填写，请安装OIDC协议填写

redirect\_uri：登录后重定向地址，需要与Client的配置相同

state：任意值，登录成功后重定向时原样返回Client站点

nonce：任意值

1. IdentityServer返回



IdentityServer返回302，将我们重定向至 /Account/Login，并携带参数ReturnUrl，改参数是我们登录IdentityServer成功后要执行的跳转

1. 登录IdentityServer
2. 登录

我们执行如下请求登录IdentityServer

POST <http://localhost:5000/Account/Login>

{

 "Username": "test",

 "Password": "123456",

 "ReturnUrl": "%2Fconnect%2Fauthorize%2Fcallback%3Fclient\_id%3Dhybrid%26scope%3Dopenid%2520profile%26response\_type%3Dcode%2520id\_token%2520token%26redirect\_uri%3Dhttps%253A%252F%252Fnotused%26state%3Dabc%26nonce%3Dxyz"

}

1. IdentityServer返回

由于我们的登录方法返回的是null，所以登录的返回是200 null

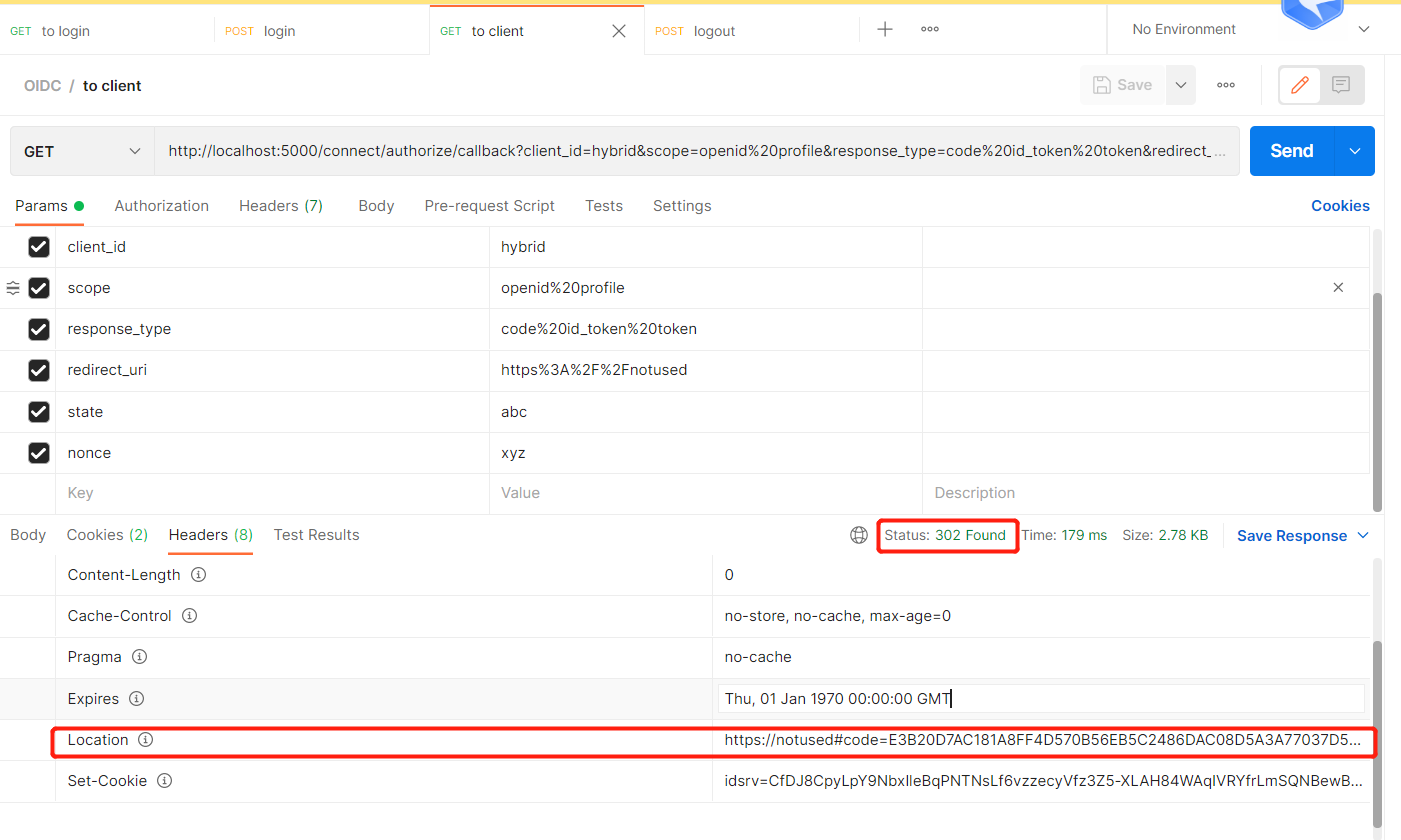
1. 请求token

登录后，我们请求步骤1的ReturnUrl

1. 请求token Url

http://localhost:5000/connect/authorize/callback?client\_id=hybrid&scope=openid%20profile&response\_type=code%20id\_token%20token&redirect\_uri=https%3A%2F%2Fnotused&state=abc&nonce=xyz

1. IdentityServer返回



IdentityServer将我们重定向到一个地址，地址如下：

https://notused#code=E3B20D7AC181A8FF4D570B56EB5C2486DAC08D5A3A77037D5541E797BB86946A&id\_token=...&access\_token=...&token\_type=Bearer&expires\_in=3600&scope=openid%20profile&state=abc&session\_state=95wBjc81OZWJYmUtBvcEzhVpxW3oSVdqXvm4uOFfbao.A7736FCE0B8B366DA9642BF10725B70C

我们可以看到其重定向到我们的站点https://notused，并携带code、id\_token、access\_token等信息

至此，我们的OIDC成功部署