**实现认证处理器**

为了提供登录功能，我们必须继承SignInAuthenticationHandler，如下：

public class MySignInAuthenticationHandler<TOptions> : SignInAuthenticationHandler<TOptions> where TOptions : AuthenticationSchemeOptions, new ()

{

    public MySignInAuthenticationHandler (IOptionsMonitor<TOptions> options, ILoggerFactory logger, UrlEncoder encoder, ISystemClock clock) : base (options, logger, encoder, clock) { }

    /// <summary>

    /// 处理认证

    /// </summary>

    /// <returns></returns>

    protected override Task<AuthenticateResult> HandleAuthenticateAsync ()

    {

        // 从Cookie中获取加密字符串

        string encryptString = Context.Request.Cookies["myAuth"];

        if (encryptString == null) {

            return Task.FromResult (AuthenticateResult.NoResult ());

        }

        // 解密，0123456789012345为密钥

        string ticketSting = AesDecrypt (encryptString, "0123456789012345");

        if (ticketSting == null) {

            return Task.FromResult (AuthenticateResult.NoResult ());

        }

        // base64解码

        byte[] ticketBytes;

        try {

            ticketBytes = Convert.FromBase64String (ticketSting);

        } catch {

            return Task.FromResult (AuthenticateResult.NoResult ());

        }

        // 反序列化，获取票据

        TicketSerializer ticketSerializer = new TicketSerializer ();

        AuthenticationTicket ticket = ticketSerializer.Deserialize (ticketBytes);

        if (ticket?.Principal == null) {

            return Task.FromResult (AuthenticateResult.NoResult ());

        }

        // 返回认证成功

        return Task.FromResult (AuthenticateResult.Success (ticket));

}

    /// <summary>

    /// 处理登录

    /// </summary>

    /// <param name="user"></param>

    /// <param name="properties"></param>

    /// <returns></returns>

    protected override Task HandleSignInAsync (ClaimsPrincipal user, AuthenticationProperties properties)

    {

        // 生成票据

        var ticket = new AuthenticationTicket (user, properties, "myScheme");

        // 序列化票据

        var ticketSerializer = new TicketSerializer ();

        byte[] ticketBytes = ticketSerializer.Serialize (ticket);

        // 编码为base64

        string ticketSting = Convert.ToBase64String (ticketBytes);

        // 加密票据，0123456789012345为密钥

        string encryptString = AesEncrypt (ticketSting, "0123456789012345");

        // 设置 Cookie

        Response.Cookies.Append ("myAuth", encryptString, new CookieOptions () {

            Expires = DateTime.Now.AddDays (1),

                IsEssential = true

        });

        // 返回完成

        return Task.CompletedTask;

}

    /// <summary>

    /// 处理退出

    /// </summary>

    /// <param name="properties"></param>

    /// <returns></returns>

    protected override Task HandleSignOutAsync (AuthenticationProperties properties)

    {

        // 清理Cookie

        Context.Response.Cookies.Append ("myAuth", "", new CookieOptions () {

            Expires = DateTime.Now,

            IsEssential = true

        });

        return Task.CompletedTask;

}

    /// <summary>

    /// 加密

    /// </summary>

    /// <param name="str">待加密字符串</param>

    /// <param name="key">密钥 KEY（必须128位或256位，注：一个UTF-8英文字符站8位）</param>

    /// <returns></returns>

    private static string AesEncrypt (string str, string key)

    {

        string result;

        try {

            if (string.IsNullOrEmpty (str)) {

                result = null;

            } else {

                byte[] bytes = Encoding.UTF8.GetBytes (str);

                RijndaelManaged rijndaelManaged = new RijndaelManaged {

                    Key = Encoding.UTF8.GetBytes (key),

                    Mode = CipherMode.ECB,

                    Padding = PaddingMode.PKCS7

                };

                ICryptoTransform cryptoTransform = rijndaelManaged.CreateEncryptor ();

                byte[] array = cryptoTransform.TransformFinalBlock (bytes, 0, bytes.Length);

                result = Convert.ToBase64String (array, 0, array.Length);

                return result;

            }

        } catch (Exception ex) {

            result = null;

            System.Console.WriteLine (ex);

        }

        return result;

}

    /// <summary>

    /// 解密

    /// </summary>

    /// <param name="str">待解密的字符串</param>

    /// <param name="key">密钥 KEY（必须128位或256位，注：一个UTF-8英文字符站8位）</param>

    /// <returns></returns>

    private static string AesDecrypt (string str, string key)

    {

        if (string.IsNullOrEmpty (str)) {

            return null;

        }

        string result;

        try {

            byte[] array = Convert.FromBase64String (str);

            RijndaelManaged rijndaelManaged = new RijndaelManaged {

                Key = Encoding.UTF8.GetBytes (key),

                Mode = CipherMode.ECB,

                Padding = PaddingMode.PKCS7

            };

            ICryptoTransform cryptoTransform = rijndaelManaged.CreateDecryptor ();

            byte[] bytes = cryptoTransform.TransformFinalBlock (array, 0, array.Length);

            result = Encoding.UTF8.GetString (bytes);

        } catch (Exception ex) {

            result = null;

            System.Console.WriteLine (ex);

        }

        return result;

    }

}

**注册认证处理器**

public void ConfigureServices (IServiceCollection services)

{

    services.AddAuthentication (options => {

            // 默认使用的 Scheme 为 myScheme

            options.DefaultScheme = "myScheme";

        })

        // 添加我们的认证处理器

        // 指定我们的认证处理器的 Scheme 为 myScheme

        .AddScheme<AuthenticationSchemeOptions, MySignInAuthenticationHandler<AuthenticationSchemeOptions>> ("myScheme", "", (AuthenticationSchemeOptions options) => {

            // 这里配置的 AuthenticationSchemeOptions 可以在我们的认证处理器中访问

            options.ClaimsIssuer = "mysite";

        });

    ...

}

public void Configure (IApplicationBuilder app, IHostingEnvironment env)

{

    app.UseCookiePolicy ();

    // 添加认证处理中间件

app.UseAuthentication ();

    ...

}

**在控制器中添加登录退出功能**

public class MyController : Controller

{

    // 通过依赖容器注入数据库上下文

    public MyController () { }

    public IActionResult Index () {

        return View ();

}

    [Authorize]

    // 访问数据库上下文

    public IActionResult MyAction () {

        ViewData["Name"] = "MyAction";

        return View ();

}

    public IActionResult Login () {

        var claimIdentity = new ClaimsIdentity ("myAuthenticationType");

        // 添加几个Claim

        claimIdentity.AddClaim (new Claim (ClaimTypes.Name, "bob"));

        var principal = new ClaimsPrincipal (claimIdentity);

        // 登录

        HttpContext.SignInAsync (principal);

        return Redirect ("/My/MyAction");

}

    public IActionResult Logout () {

        // 退出

        HttpContext.SignOutAsync ();

        return Redirect ("/");

    }

}