证件单元: ClaimTypes 保存姓名等信息

身份证:ClaimIdentity 添加或者删除证件

string Name 定义证件对象的基本功能

string AuthenticationType 用于标识证件的载体类型

bool IsAuthenticated 是否合法证件

证件当事人:ClaimsPrincipal

### 认证管理员: AuthenticationManager 处于httpcontent

public abstract class AuthenticationManager{

//AuthenticateContext包含了需要认证的上下文，里面就有小李

public abstract Task AuthenticateAsync(AuthenticateContext context);

//握手

public abstract Task ChallengeAsync(string authenticationScheme, AuthenticationProperties properties, ChallengeBehavior behavior);

//登入

public abstract Task SignInAsync(string authenticationScheme, ClaimsPrincipal principal, AuthenticationProperties properties);

//登出

public abstract Task SignOutAsync(string authenticationScheme, AuthenticationProperties properties);

}

IAuthenticationHandler: 这个接口是在 AuthenticationManager 实现类 DefaultAuthenticationManager 中延伸出来的，所以大家不用再去看里面的源码了，记住以后如果需要重写认证相关的东西，实现IAuthenticationHandler就可以了。

Authentication 中间件

对 IAuthenticationHandler 的初步实现，封装了 AuthenticationHandler 这个抽象类，把具体的核心功能都交给下游去实现了，下面的CookieAuthentication 中间件核心类 CookieAuthenticationHandler 就是继承自AuthenticationHandler， 知道这么多就够了。

### 准备

* 1. 创建一个名为AuthorizationForoNetCore的（web）解决方案,选择Empty模板
  2. 添加相关nuget包引用Microsoft.AspNetCore.Mvc（选择最新版本）
  3. 编辑Startup.cs文件，添加mvcservice并进行默认路由配置

1 public class *Startup* 2 { 3 public void ConfigureServices(IServiceCollection services) 4 { 5 services.AddMvc(); 6 } 7 8 public void Configure(IApplicationBuilder app) 9 {10 app.UseMvc(routes =>11 {12 routes.MapRoute(13 name: "default",14 template: "{controller=Home}/{action=Index}/{id?}");15 });16 }17 }

* 1. 添加Controllers文件夹，添加HomeContrller

public class HomeController : Controller

{ public IActionResult Index() { return View();

}

}

* 1. 创建Views/Home文件夹,并添加Index(Action)对应的Index.cshtml文件

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <!--Index.cshtml-->  假如生活欺骗了你  假如生活欺骗了你，  不要悲伤，不要心急！  忧郁的日子里须要镇静：  相信吧，快乐的日子将会来临！ |

### 使用Authorization

1. 添加相关nuget包（均使用最新版本）
   1. Microsoft.AspNetCore.Authorization
   2. Microsoft.AspNetCore.Authentication.Cookies
2. 在ConfigureServices()方法中添加对应服务:  services.AddAuthorization()
3. 在Index(Action)方法上添加 [Authorize] 特性，毫无疑问，添加后执行dotnet run 指令后后会返回401的授权码，那么接着操作
4. 编辑Startup.cs在Configureapp.UseMvc()方法之前，我们添加一个cookie 中间件，用于持久化请求管道中的身份配置信息

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | app.UseCookieAuthentication(new CookieAuthenticationOptions  {                 AuthenticationScheme = "MyCookieMiddlewareInstance",                 LoginPath = new PathString("/Account/Unauthorized/"),                 AccessDeniedPath = new PathString("/Account/Forbidden/"),                 AutomaticAuthenticate = true,                 AutomaticChallenge = true  }); |

1. tip：相关配置参数细节请参阅：https://docs.asp.net/en/latest/security/authentication/cookie.html
2. 添加Controllers/Account文件夹，添加 AccountController.cs 控制器文件，实现上述指定的方法，可能这里你会疑惑，为什么文档里不是一个 /Account/Login 这类的，文档说了别较真，这就是个例子而已，继续你就明白了。
3. 添加并实现上述中间件重定向的action 方法如下，你可以看到其实Unauthorized方法模拟实现了登陆的过程。tip:假如你添加Unauthorized视图，并且没有该不实现模拟登陆，那么运行你会直接看到 Unauthorized.cshtml 的内容，这里我们不需要添加该视图，仅作说明。
4. public class AccountController : Controller
5. { public async Task<IActionResult> Unauthorized(string returnUrl = null) {
6. List<Claim> claims = new List<Claim>();
7. claims.Add(new Claim(ClaimTypes.Name, "halower", ClaimValueTypes.String, "https://www.cnblogs.com/rohelm")); var userIdentity = new ClaimsIdentity("管理员");
8. userIdentity.AddClaims(claims); var userPrincipal = new ClaimsPrincipal(userIdentity); await HttpContext.Authentication.SignInAsync("MyCookieMiddlewareInstance", userPrincipal, new AuthenticationProperties
9. {
10. ExpiresUtc = DateTime.UtcNow.AddMinutes(20),
11. IsPersistent = false,
12. AllowRefresh = false
13. });
14. if (Url.IsLocalUrl(returnUrl))
15. { return Redirect(returnUrl);
16. } else
17. { return RedirectToAction("Index", "Home");
18. }
19. }
20. public IActionResult Forbidden() { return View();
21. }

}

1. 编辑 Home/Index.schtml

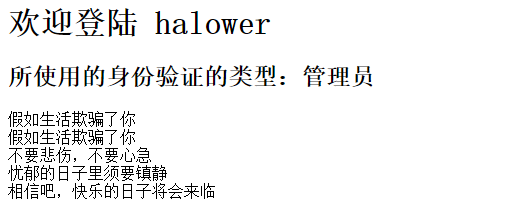
@using System.Security.Claims;

@if (User.Identities.Any(u => u.IsAuthenticated))

{<h1>欢迎登陆 @User.Identities.First(u => u.IsAuthenticated).FindFirst(ClaimTypes.Name).Value</h1>

<h2>所使用的身份验证的类型：@User.Identity.AuthenticationType</h2> }

<article>假如生活欺骗了你<br />假如生活欺骗了你<br />不要悲伤，不要心急<br />忧郁的日子里须要镇静<br />相信吧，快乐的日子将会来临 　</article>

1. 运行代码你会看到如下结果（程序获取我们提供的由issuer发布claims并展示在视图中,后续会检查Claims看他们是否匹配）  
   

### 使用全局授权策略

1. 去除Home/Index （Action）上的  [Authorize]  特性
2. 添加 Views/Account/Forbidden.cshtml 页面,内容为 <h1>拒绝访问</h1>
3. 修改 ConfigureServices 方法中的 services.AddMvc() 使用它的 AddMvc(this IServiceCollection services, Action<MvcOptions> setupAction) 重载
4. 运行查看结果，你会发现这几乎成了一个无限的重定向从而造成错误，因为每个页面都需要授权。
5. 为 AccountController 添加 [AllowAnonymous] 特性，启动匿名访问，再次运行项目，查看结果
6. 结果就是重定向到了 Forbidden.cshtml 页面

### 使用角色授权

1. 在 HomeController 上添加 [Authorize(Roles = "Administrator")] 特性
2. 在模拟登陆处（ Unauthorized方法中 ）添加角色说明的身份信息条目：
3. claims.Add(new Claim(ClaimTypes.Role, "Administrator", ClaimValueTypes.String, "https://www.cnblogs.com/rohelm"));
4. 运行项目查看结果

可以使用中你会发现Asp.net Core安全验证方面较以往的版本最大的改变就是全部采用中间件的方式进行验证授权，并很好的使用了Policy （策略）这个概念，下那么继续~。

### 基于声明的授权

1. 返回Startup.cs,修改 services.AddAuthorization() 方法如下：
2. services.AddAuthorization(options =>{
3. options.AddPolicy("EmployeeOnly", policy => policy.RequireClaim("EmployeeNumber"));

});

1. 修改HomeController上的特性，添加 [Authorize(Policy = "EmployeeOnly")]
2. 运行项目查看结果，发现被拒绝了
3. 在模拟登陆处 Unauthorize方法添加：

claims.Add(new Claim("EmployeeNumber", "123456", ClaimValueTypes.String, "http://www.cnblogs.com/rohelm"));

1. goto 3.
2. 多重策略的应用，与之前的版本几乎一样，例如本次修改的结果可以为：

[Authorize(Roles = "Administrator")] public class HomeController:Controller

{

[Authorize(Policy = "EmployeeOnly")] public IActionResult Index() { return View();

}

}

1. 详情请参阅：https://docs.asp.net/en/latest/security/authorization/claims.html的说明

### 自定义授权策略

自定义授权策略的实现，包括实现一个 IAuthorizationRequirement 的Requirement，和实现 AuthorizationHandler<TRequirement> 的处理器，这里使用文档

https://docs.asp.net/en/latest/security/authorization/policies.html中的Code。

1. 添加 MinimumAgeHandler 处理器实现

public class MinimumAgeRequirement: AuthorizationHandler<MinimumAgeRequirement>, IAuthorizationRequirement

{ int \_minimumAge; public MinimumAgeRequirement(int minimumAge) {

\_minimumAge = minimumAge;

} protected override void Handle(AuthorizationContext context, MinimumAgeRequirement requirement) { if (!context.User.HasClaim(c => c.Type == ClaimTypes.DateOfBirth))

{ return;

} var dateOfBirth = Convert.ToDateTime(

context.User.FindFirst(c => c.Type == ClaimTypes.DateOfBirth).Value); int calculatedAge = DateTime.Today.Year - dateOfBirth.Year; if (dateOfBirth > DateTime.Today.AddYears(-calculatedAge))

{

calculatedAge--;

} if (calculatedAge >= \_minimumAge)

{

context.Succeed(requirement);

}

}

}

1. 在 AddAuthorization  中添加一个名为Over21的策略

options.AddPolicy("Over21", policy => policy.Requirements.Add(new MinimumAgeRequirement(21)));

1. 在HomeController上应用该策略  [Authorize(Policy = "Over21")]
2. 在 Unauthorized 函数中添加对应的声明信息条目 claims.Add(new Claim(ClaimTypes.DateOfBirth, "1900-01-01", ClaimValueTypes.Date));
3. 修改时间（例如小于21岁的生日，2000-01-01）并运行调试，查看结果

### 对一个Requirement应用多个处理器

tip：上面的演示，我们使用了一个同时实现AuthorizationHandler<MinimumAgeRequirement>, IAuthorizationRequirement的MinimumAgeRequirement来做演示，但是如果一个Requirement徐要实现多个处理器就需要分开写了，原因很简单，这里无法实现类的多重继承。

下面我们实现一个使用Token登陆的需求

1. 添加一个LoginRequirement的需求
2. public class LoginRequirement: IAuthorizationRequirement
3. {

}

1. 添加一个使用用户名密码登陆的处理器

public class HasPasswordHandler : AuthorizationHandler<LoginRequirement>

{ protected override void Handle(AuthorizationContext context, LoginRequirement requirement) { if (!context.User.HasClaim(c => c.Type == "UsernameAndPassword" && c.Issuer == "http://www.cnblogs.com/rohelm")) return;

context.Succeed(requirement);

}

}

1. 在一些场景中我们也会使用发放访问令牌的方式让用户登陆

public class HasAccessTokenHandler : AuthorizationHandler<LoginRequirement>

{ protected override void Handle(AuthorizationContext context, LoginRequirement requirement) { if (!context.User.HasClaim(c => c.Type == "AccessToken" && c.Issuer == "http://www.cnblogs.com/rohelm")) return; var toeknExpiryIn = Convert.ToDateTime(context.User.FindFirst(c => c.Type == "AccessToken" && c.Issuer == "http://www.cnblogs.com/rohelm").Value); if (toeknExpiryIn > DateTime.Now)

{

context.Succeed(requirement);

}

}

}

1. 在 AddAuthorization  中添加一个名为CanLogin的策略

options.AddPolicy("CanLogin", policy => policy.Requirements.Add(new LoginRequirement()));

1. 注册自定义策略
2. services.AddSingleton<IAuthorizationHandler, HasPasswordHandler>();

services.AddSingleton<IAuthorizationHandler, HasAccessTokenHandler>();

1. 在Unauthorized 函数中添加对应的声明信息条目
2. claims.Add(new Claim("UsernameAndPassword", "123456", ClaimValueTypes.String, "http://www.cnblogs.com/rohelm"));
3. // 测试切换登陆声明方式

// claims.Add(new Claim("AccessToken", DateTime.Now.AddMinutes(1).ToString(), ClaimValueTypes.String, "http://www.cnblogs.com/rohelm"));

1. 在HomeController上应用该策略  [Authorize(Policy = "CanLogin")]
2. 运行并查看结果。

### 基于资源的Requirements

在实际开发者中，除了基于用户的授权验证外，通过我们也会遇到针对一些资源的授权限制，例如有的人可以编辑文档，有的人只能查看文档，由此引出该话题

https://docs.asp.net/en/latest/security/authorization/resourcebased.html

1. 定义一个Document类
2. public class Document
3. { public int Id { get; set; } public string Author { get; set; }

}

1. 定义Document仓储接口
2. public interface *IDocumentRepository*
3. {
4. IEnumerable<Document> Get();
5. Document Get(int id);

}

1. 模拟实现上述接口

public class FakeDocumentRepository : IDocumentRepository

{ static List<Document> \_documents = new List<Document> { new Document { Id = 1, Author = "halower" }, new Document { Id = 2, Author = "others" }

}; public IEnumerable<Document> Get() { return \_documents;

} public Document Get(int id) { return \_documents.FirstOrDefault(d => d.Id == id);

}

}

1. 注册接口实现类

services.AddSingleton<IDocumentRepository, FakeDocumentRepository>();

1. 创建一个 DocumentController 并修改为如下内容

public class DocumentController : Controller

{

IDocumentRepository \_documentRepository; public DocumentController(IDocumentRepository documentRepository) {

\_documentRepository = documentRepository;

} public IActionResult Index() { return View(\_documentRepository.Get());

} public IActionResult Edit(int id) { var document = \_documentRepository.Get(id); if (document == null) return new NotFoundResult(); return View(document);

}

}

1. 添加对应 Index.cshtml  视图文件

@model IEnumerable<AuthorizationForoNetCore.Modles.Document>

<h1>文档列表</h1>@foreach (var document in Model)

{ <p>

@Html.ActionLink("文档 #" + document.Id, "编辑", new { id = document.Id }) </p>}

1. 添加对应的 Edit.cshtml 视图文件
2. @model AuthorizationForoNetCore.Modles.Document<h1>文档 #@Model.Id</h1>

<h2>作者: @Model.Author</h2>

1. 定义EditRequirement
2. public class EditRequirement : IAuthorizationRequirement
3. {

}

1. 添加对应的编辑文档处理器

public class DocumentEditHandler : AuthorizationHandler<EditRequirement, Document>

{ protected override void Handle(AuthorizationContext context, EditRequirement requirement, Document resource) { if (resource.Author == context.User.FindFirst(ClaimTypes.Name).Value)

{

context.Succeed(requirement);

}

}

}

1. 在 ConfigureServices() 方法中注册处理器实现

1 services.AddSingleton<IAuthorizationHandler, DocumentEditHandler>();

1. 由于对于文档的授权服务仅仅反正在操作方法的内部，因此我们需要直接注入 IAuthorizationService 对象并在需要的Action内部直接处理

public class DocumentController : Controller

{

IDocumentRepository \_documentRepository;

IAuthorizationService \_authorizationService; public DocumentController(IDocumentRepository documentRepository, IAuthorizationService authorizationService) {

\_documentRepository = documentRepository;

\_authorizationService = authorizationService;

} public IActionResult Index() { return View(\_documentRepository.Get());

} public async Task<IActionResult> Edit(int id) { var document = \_documentRepository.Get(id); if (document == null) return new NotFoundResult(); if (await \_authorizationService.AuthorizeAsync(User, document, new EditRequirement()))

{ return View(document);

} else

{ return new ChallengeResult();

}

}

}

1. 运行查看结果

### 在视图中进行授权

问题来了额，上面示例的视图中怎么做限制了，那就继续了

1.使用  @inject 命令注入 AuthorizationService

2.应用该上述同样策略，做简单修改

@using Microsoft.AspNetCore.Authorization

@model IEnumerable<AuthorizationForoNetCore.Modles.Document>@inject IAuthorizationService AuthorizationService

@using AuthorizationForoNetCore.Policy<h1>文档列表</h1>@{ var requirement = new EditRequirement(); foreach (var document in Model)

{ if (await AuthorizationService.AuthorizeAsync(User, document, requirement)) { <p>

@Html.ActionLink("文档 #" + document.Id, "编辑", new { id = document.Id }) </p>

}

}

}

 请在运行时清理Cookie，或者在试验时直接暂时禁用