## **基于资源的授权**

### **定义资源Requirement**

我们先定义一个代表操作的****Requirement****：

public class MyRequirement : IAuthorizationRequirement

{

public string Name { get; set; }

}

然后，我们预定义一些常用的操作，方便业务中的调用：

public static class Operations

{

public static MyRequirement Create = new MyRequirement { Name = "Create" };

public static MyRequirement Read = new MyRequirement { Name = "Read" };

public static MyRequirement Update = new MyRequirement { Name = "Update" };

public static MyRequirement Delete = new MyRequirement { Name = "Delete" };

}

### **实现资源授权Handler**

public interface IDocument

{

string Creator { get; set; }

}

然后实现我们的授权Handler:

public class DocumentAuthorizationHandler : AuthorizationHandler<OperationAuthorizationRequirement>

{

protected override Task HandleRequirementAsync(AuthorizationHandlerContext context, OperationAuthorizationRequirement requirement, IDocument resource)

{

// 如果是Admin角色就直接授权成功

if (context.User.IsInRole("admin"))

{

context.Succeed(requirement);

}

else

{

// 允许任何人创建或读取资源

if (requirement == Operations.Create || requirement == Operations.Read)

{

context.Succeed(requirement);

}

else

{

// 只有资源的创建者才可以修改和删除

if (context.User.Identity.Name == resource.Creator)

{

context.Succeed(requirement);

}

else

{

context.Fail();

}

}

}

return Task.CompletedTask;

}

}

不要忘了，还需要将DocumentAuthorizationHandler注册到DI系统中：

services.AddSingleton<IAuthorizationHandler, DocumentAuthorizationHandler>();

### **调用AuthorizationService**

可以在我们的应用代码中调用IAuthorizationService来完成授权了

IAuthorizationService接口：

public interface IAuthorizationService

{

Task<AuthorizationResult> AuthorizeAsync(ClaimsPrincipal user, object resource, IEnumerable<IAuthorizationRequirement> requirements);

Task<AuthorizationResult> AuthorizeAsync(ClaimsPrincipal user, object resource, string policyName);

}

[Authorize]

public class DocumentsController : Controller

{

public async Task<ActionResult> Details(int? id)

{

IDocument document = \_docStore.Find(id.Value);

if (document == null)

{

return NotFound();

}

// 验证权限

if ((await \_authorizationService.AuthorizeAsync(User, document, Operations.Read)).Succeeded)

{

return View(document);

}

else

{

return new ForbidResult();

}

}

}

如上，在授权失败时，我们返回了ForbidResult，建议不要返回ChallengeResult，因为我们要明确的告诉用户是无权访问，而不是未登录。

### **自定义授权过滤器**

[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method, AllowMultiple = true, Inherited = true)]

public class PermissionFilter : Attribute, IAsyncAuthorizationFilter

{

public PermissionFilter(string name)

{

Name = name;

}

public string Name { get; set; }

public async Task OnAuthorizationAsync(AuthorizationFilterContext context)

{

var authorizationService = context.HttpContext.RequestServices.GetRequiredService<IAuthorizationService>();

var authorizationResult = await authorizationService.AuthorizeAsync(context.HttpContext.User, null, new PermissionAuthorizationRequirement(Name));

if (!authorizationResult.Succeeded)

{

context.Result = new ForbidResult();

}

}

}

接下来，我们就可以直接在控制器中使用PermissionFilter过滤器来完成基于权限的授权了：

[Authorize]

public class UserController : Controller

{

[PermissionFilter(Permissions.UserRead)]

public ActionResult Index()

{

return View(\_userStore.GetAll());

}

[PermissionFilter(Permissions.UserCreate)]

public ActionResult Create()

{

}

[PermissionFilter(Permissions.UserCreate)]

[HttpPost]

[ValidateAntiForgeryToken]

public IActionResult Create([Bind("Title")] User user)

{

}

[PermissionFilter(Permissions.UserUpdate)]

public IActionResult Edit(int? id)

{

}

[PermissionFilter(Permissions.UserUpdate)]

[HttpPost]

[ValidateAntiForgeryToken]

public IActionResult Edit(int id, [Bind("Id,Title")] User user)

{

}

}