ASP.NET Core 2.0 授权实现方法

在 asp.net core 2.0 中可以通过以下五种方法实现访问授权：

一、 基于策略或角色（授权要求）

基本流程是，一个受保护的资源（可以理解为一个控制器或控制器中的动作），受多个策略（策略- Policy 只是一个名称，是一个字符串，它不包含任务代码，它像一个指针一样指向一段代码，这段代码就是要求- Requirement）保护，一个策略有一个或多个要求（要求是一个供策略使用的类，它必须继承自 IAuthorizationRequirement，要求可以不包含任务代码，也可以包含一些属性，如果包含一些属性，这些属性一般需要在构造函数中初始化，初始化的数据来自 Startup 文件中的 ConfigureServices 方法，并且要求只在这里实例化。注意，这个要求类会自动注入到它的处理程序- Handler 当中），一个要求有一个或多个处理程序（要求的处理程序是真正执行访问授权的地方，要求的处理程序必须继承自 AuthorizationHandler<T>接口，其中的 T 就是要求的类名，这样要求就和它的处理程序关联起来了。要求的处理程序必须实现接口的 HandleRequirementAsync（AuthorizationHandlerContext context, TRequirement requirement）方法，其中的 TRequirement 参数就是要求的类型，系统会自动注入这个参数，所以在此方法中可以直接访问要求类的公共属性和方法。此方法就是执行授权逻辑的地方）。

示例代码：

要求- Requirement：

```
public class IsAdminRequirement: IAuthorizationRequirement
{
    public IAuthData MyAuthData;
    public IsAdminRequirement(IAuthData authData)
    {
        MyAuthData = authData;
    }
}
```

要求的处理程序-Handler：

```
public class AdminAuthorizationHandler : AuthorizationHandler<IsAdminRequirement>
{
    protected override Task HandleRequirementAsync(AuthorizationHandlerContext context, IsAdminRequirement requirement)
    {

        var filterContext = context.Resource as AuthorizationFilterContext;

        if (filterContext?.ActionDescriptor is ControllerActionDescriptor action)
        {
            var permisson = string.Format("{0}.{1}", action.ActionName, action.ControllerName);
            if (context.User.HasClaim(c => c.Type == "Permisson" && c.Value == permisson))
            {
                context.Succeed(requirement);
            }

        }

        return Task.FromResult(0);
    }
}
```

授权规则，如果当前用户的声明中包含要访问的控制器和动作，则授权成功，否则失败
用户声明的信息需要在登录时填充。

注册策略：

```
services.AddAuthentication("Cookies").AddCookie();
services.AddSingleton<IAuthorizationHandler, AdminAuthorizationHandler>();

IServiceProvider provider = services.BuildServiceProvider();

services.AddAuthorization(options =>
{
    options.AddPolicy(
        "Admin",
        policyBuilder => policyBuilder.AddRequirements(new IsAdminRequirement(provider.GetService<IAuthData>())));
});
```

策略的名称

使用策略：

```
[Authorize("Admin")]
public IActionResult Contact()
{
    ViewData["Message"] = "Your contact page.";

    return View();
}
```

策略的名称

登录方法：

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Login(LoginViewModel model, string returnUrl = null)
{
    ViewData["ReturnUrl"] = returnUrl;
    if (ModelState.IsValid)
    {
        if (model.Email == "jasstion@gmail.com" && model.Password == "admin")
        {
            var identity = new ClaimsIdentity(CookieAuthenticationDefaults.AuthenticationScheme);

            IList<Claim> claims = new List<Claim>();

            claims.Add(new Claim("Permisson", "Home.Index"));
            claims.Add(new Claim("Permisson", "Home.Contact"));

            identity.AddClaims(claims);

            await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, new ClaimsPrincipal(identity));
            return RedirectToLocal(returnUrl);
        }
        else
        {
            ModelState.AddModelError(string.Empty, "Invalid login attempt.");
            return View(model);
        }
    }
    return View(model);
}
```

这就是用户所拥有的权限，可以从数据库表中获取

二、 使用过滤器

过滤器的作用是在 Action 执行前或执行后做一些加工处理。基于这个特性我们可以使用过滤器来实现访问授权。

在 asp.net core 中提供了五种过滤器，分别用于实现不同的功能，它们分别是：Authorization Filter、Resource Filter、Action Filter、Exception Filter、Result Filter。

要实现访问授权的功能我们只需要实现 Authorization 过滤器即可，编写此过滤器需要继承 IAuthorizationFilter（同步）或 IAsyncAuthorizationFilter（异步）接口，然后实现接口的 OnAuthorization/OnAuthorizationAsync 方法，此方法是访问授权的关键，授权逻辑都写在此方法中。如果验证通过则不需要作任何处理，如果验证失败可以对进行 context.Result 赋值（可以是一些对用户的提示信息，比如您没有权限访问此方法）。

过滤器编写完成后需要在 Startup 文件中进行注册才能生效。注意，过滤器的注册分为全局注册和区域注册（就是在控制器或动作上注册过滤器，需要使用

[TypeFilter(type)]特性，其中 type 就是过滤器类型），如果注册为全局，那么所有的请求都会经过此过滤器；如果是区域注册，那么只有访问注册了过滤器的控制器才会生效。

示例代码：

编写一个过滤器：

```
public class AuthorizationFilter : IAuthorizationFilter
{
    public AuthorizationFilter(IAuthData authData)
    {
        //试一下过滤器的注入

    }
    public void OnAuthorization(AuthorizationFilterContext context)
    {
        if (!context.HttpContext.User.Identity.IsAuthenticated)
        {
            context.Result = new RedirectResult("/Account/Login");
        }
        else
        {
            var action = context.ActionDescriptor as ControllerActionDescriptor;
            var permisson = string.Format("{0}.{1}", action.ControllerName, action.ActionName);
            if (!context.HttpContext.User.HasClaim(c => c.Type == "Permisson" && c.Value == permisson))
            {
                context.Result = new ContentResult { Content = "您无限访问此操作" };
            }
        }
    }
}
```

注册为全局过滤器：

```
//注册全局过滤器
services.AddMvc(config=>config.Filters.Add(new AuthorizationFilter()));
```

注册为区域过滤器：

```
[TypeFilter(typeof(AuthorizationFilter))]
public IActionResult About()
{
    ViewData["Message"] = "Your application description page.";

    return View();
}
```

关于过滤的注入，如果需要在过滤器使用其它组件，可以通过构造函数注入

的方式直接把组件注入到过滤器，但是要确保在使用过滤器之前把需要的组件加到了 ServiceCollection 集合中，其它的不需要额外代码。

关于当前用户的权限，当前用户的权限可以在用户登录时通过声明的方式直接保存到 httpcontext.user 对象中，使用方式和基于策略的方式相同，也可以把 DbContext 注入到过滤器，然后根据当前用户的标识去数据查询此用户所拥有的权限。

最后值得注意的是，使用此方法可以不依赖任何系统自带的授权组件，可以完全自定义。

三、 使用自定义特性

自定义属性的使用方法和过滤器类似，本质上仍然使用的是过滤器，唯一不同的是注册过滤的方式不同。注册过滤时需要借助 AddMVC 方法的 Filters.Add 方法或者 TypeFiler 特性，而使用自定义属性的方法相当于重写了一个 TypeFiler。

方法是先编写一个继承自 TypeFilterAttribute 的特性类，然后在构造函数中把自定义的过滤器传递给基类，从而使此特性与指定的过滤器关联起来，当访问来到此特性修饰的控制器或动作时会先执行过滤器的中的方法。

示例代码：

编写一个特性类：

```
public class AuthAttribute : TypeFilterAttribute
{
    public AuthAttribute(PermissionItem item, PermissionAction action)
    : base(typeof(AuthorizeActionFilter))
    {
        Arguments = new object[] { item, action };
    }
}
```

编写一个过滤器：

```
public class AuthorizeActionFilter : IAsyncActionFilter
{
    private readonly PermissionItem _item;
    private readonly PermissionAction _action;
    private readonly IAuthData _authData;
    public AuthorizeActionFilter(PermissionItem item, PermissionAction action,IAuthData authData)
    {
        _item = item;
        _action = action;
        _authData = authData;
    }
    public async Task OnActionExecutionAsync(ActionExecutingContext context, ActionExecutionDelegate next)
    {
        bool isAuthorized = MumboJumboFunction(context.HttpContext.User, _item, _action);

        if (!isAuthorized)
        {
            context.Result = new UnauthorizedResult();

        }
        else
        {
            await next();
        }
    }
    private bool MumboJumboFunction(ClaimsPrincipal user,PermissionItem item,PermissionAction action)
    {
        bool result = false;
        if (!user.Identity.IsAuthenticated)
        {
            result=user.HasClaim(m => m.Type == item.ToString() && m.Value == action.ToString());
        }
        return result;
    }
}
```

使用特性：

```
[Auth(PermissionItem.User, PermissionAction.Delete)]
public IActionResult Delete()
{
    return View();
}
```

这种只所以没有注册过滤器，是因为过滤器是通过特性调用的，像 Authorize 特性会调用默认的身份验证处理程序一样。

四、 使用中间件

中间件是组装成应用程序管道以处理请求和响应的类，每个中间件都可以将请求传递给下一个中间件，也可以终止当前的请求。

中间件就是一个普通的类文件，但是需要在构造函数中授收一下 RequestDelegate 参数，这个参数一个 http 请求委托，通过个请求委托来处理 http 请求，以形成 http 请求管道。

为方便使用，一般情况下需要为自定义的中间件编写一个扩展类，这个扩展类为 IApplicationBuilder 接口扩展一个使用自定义中间的方法，方法可以随便写。

代码示例：

编写一个中间件：

```
public class AuthMiddleware
{
    private readonly RequestDelegate _next;

    public AuthMiddleware(RequestDelegate next)
    {
        _next = next;
    }

    public async Task Invoke(HttpContext httpContext, IAuthData authData)
    {
        if (!httpContext.User.Identity.IsAuthenticated)
        {
            httpContext.Request.Path = PathString.FromUriComponent("/Account/Login");
        }
        await _next(httpContext);
    }

}
```

编写中间件的扩展：

```
public static class AuthMiddlewareExtensions
{
    public static IApplicationBuilder UseAuth(this IApplicationBuilder builder)
    {
        return builder.UseMiddleware<AuthMiddleware>();
    }
}
```

使用中间件：

```
//使用自定义的中间件
app.UseAuth();
```

值得注意的是，使用中间件时所有的请求都会经过此中间件。

五、 使用控制器基类

使用控制器基类实现访问授权是最简单的一种方式，它是的原理是重写 Controller 中的 OnActionExecuting 方法，此方法在任何 action 调用之前被调用。所以，可以在执行 action 之前先对请求用户进行访问权限判断，如果拥有访问权限则不进行任何处理，否则返回“无权执行此操作”的提示或者直接导航到登录页面。

示例代码：

编写基类：

```
public class BaseController : Controller
{
    public override void OnActionExecuting(ActionExecutingContext context)
    {
        if (!context.HttpContext.User.Identity.IsAuthenticated)
        {
            context.Result= new RedirectResult("/Account/Login");
        }
        else
        {
            var action = context.ActionDescriptor as ControllerActionDescriptor;
            var permisson = string.Format("{0}.{1}", action.ControllerName, action.ActionName);
            if (!context.HttpContext.User.HasClaim(c => c.Type == "Permisson" && c.Value == permisson))
            {
                context.Result = new ContentResult { Content = "您无限访问此操作" };
            }
        }
    }
}
```

使用基类：

```
public class UserController : BaseController
{
    public IActionResult Index()
    {
        return View();
    }
}
```