序章 Identity(身份验证)

# 登录原理

通常用两个同名动作方法和一个视图视图就可以完成验证。

[HttpGet] 返回登录页给用户。

[HttpPost] 接受表单，执行登录操作。

# 验证控件

ValidationMessage()

ValidationSummary()

# 重要的类

UserManager<IdentityUser>（控制器构造函数注入）

.CreateAsync(var user, string password) 注册用户。返回var user(IdentityResult)。

.DeleteAsync(var user) 删除用户。返回IdentityResult。

.FindByNameAsync(string name) 按名字查找用户。返回用户var user(IdentityUser)。

.IsInRoleAsync(var user, string RoleName) 是否属于某角色？返回bool。

.AddToRoleAsync(var user, string role) 添加进角色。返回IdentityResult。

SignInManager<IdentityUser>（控制器构造函数注入）（视图@inject注入）

.PasswordSignInAsync(var user, string password, false, false)密码登陆。返回IdentityResult。

.SignOutAsync() 注销用户。

.IsSignedIn(User) 用户是否已登录。（一般在视图中用@inject注入此类后调用）

RoleManager<IdentityUser>（控制器构造函数注入）

.CreateAsync(new IdentityRole(){}) 添加角色。返回IdentityResult。

.DeleteAsync(var role) 删除角色。返回IdentityResult。

.UpdateAsync(var role) 更新(保存修改)。返回IdentityResult。

. FindByIdAsync(string roleid) 按ID查找。返回IdentityRole。

注意：

一般用LoginViewModel来模型绑定获取表单值，不用IdentityUser。

# (必要步骤)添加验证中间件

app.UseAuthentication();

该中间件会根据当前Http请求中的Cookie信息来设置HttpContext.User属性

添加中间件后才能登录验证成功。否则无法登录成功，IsAuthentication为false。

# (必要步骤)注册IdentityDbContext服务依赖

在ConfigureServices方法中注册:

services.AddDbContext<IdentityDbContext>(…….);

services.AddDefaultIdentity<IdentityUser>().AddEntityFrameworkStores<IdentityDbContext>();

(为了在 Identity 框架中使用 EF 框架，我们需要使用 AddEntityFrameworkStores() 方法来使用 EF 框架存储数据)

数据迁移:

Add-Migration InitialIdentity –Context IdentityDbContext

Update-Database –Context IdentityDbContext

注意：

**1、以上使用的是默认的IdentityDbContext、默认的IdentityUser、默认的服务依赖.AddDefaultIdentity()（也可以继承IdentityDbContext、IdentityUser自己写……Context和……User然后用.AddIdentity<……User,……Role>().AddEntityFrameworkStores<……Context>()注册依赖）**

**2、IdentityDbContext需要Microsoft.AspNetCore.Identity.EntityFrameworkCore命名空间。**

**3、IdentityUser和UserManager等类需要Microsoft.AspNetCore.Identity命名空间。**

**4、用–Context 指定上下文，防止与其他上下文冲突。**

**5、options.UseSqlServer方法添加重载参数: b => b.MigrationsAssembly("项目名")**

# 验证授权的过滤器

[Authorize]特性：访问仅限于登录的用户。

[Authorize(Role = “,,,”)] 特性：只有特定角色可以。

[AllowAnonymous]特性：允许匿名用户。（当控制器为[Authorize]时可以把它用于动作方法）

# 未验证授权跳转页面

用AddDefaultIdentity<>注册依赖:

使用自带的默认登录视图，还需要添加一个\_LoginPartial.cshtml分部视图。

用AddIdentity<,>()注册依赖:

默认指定的登录Url为：/Accout/Login

# 补充：密码格式设置

services.Configure<IdentityUser>(options =>

{

options.Password.RequiredLength = 1;

options.Password.RequireDigit = false;

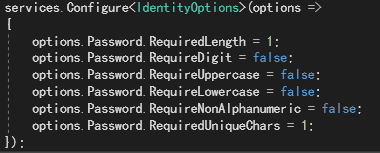
options.Password.RequireUppercase = false;

options.Password.RequireLowercase = false;

options.Password.RequireNonAlphanumeric = false;

options.Password.RequiredUniqueChars = 1;

})



# 补充：获取当前用户

Context.User或者User。（是预定义对象）

# 自定义Cookie登录认证

在ASP.NET CORE中使用Cookie认证登录用户的方法和传统的FormsAuthentication不太一样，大致步骤如下：

* 创建Claim类型的数组，将登录用户的所有信息（如用户名）存储在Claim类型的字符串键值对中
* 将上面创建的Claim类型的数组传入ClaimsIdentity中，用来构造一个ClaimsIdentity对象
* 将上面创建的ClaimsIdentity对象传入ClaimsPrincipal中，用来构造一个ClaimsPrincipal对象
* 调用HttpContext.SignInAsync方法，传入上面创建的ClaimsPrincipal对象，完成用户登录

PowerCoder:所以我们可以看到整个ASP.NET CORE的Cookie认证登录流程比以前ASP.NET的FormsAuthentication还是要复杂许多，毕竟以前一个FormsAuthentication.SetAuthCookie方法就搞定了。

记得注册服务： services.AddAuthentication(……).AddCookie(……);