|  |  |  |  |
| --- | --- | --- | --- |
| 密级： 内部 |  | 编 号： | Lecture-2015-1 |
| 本文知识产权属于南华大学计算机科学与技术学院，未经我院书面同意，不得复制、传播、发表和用于其他方面 |  | 页 数： | 共47页 |
|  | 版 本： | A |
|  | 保管期限： | 长期 |



ASP.NET安全控制

ASP.NET Security Control

授课讲义

|  |  |  |
| --- | --- | --- |
| 编 写 | ： | 李 萌 |
| 校 对 | ： | 李 萌 |
| 审 核 | ： | 李 萌 |
| 批 准 | ： | 李 萌 |
| 生效日期 | ： | 2015 年 2 月 26 日 |

南华大学计算机与科学与技术学院/软件学院

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ASP.NET安全控制 | | | | | | | |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| A | 2015-02-26 | CFC | 李萌 |  |  |  |  |
| 版本 | 日 期 | 状 态 | 编写/日期 | 校对/日期 | 审核/日期 | 审定/日期 | 批准/日期 |
| 会签： | | | | | | | |
| 说明：初版  编写人签字/盖章： 李萌  日期：2015-02-26 | | | | | | | |

**文件修改记录**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 版本 | 日期 | 章节号 | 修改原因 | 修改内容 | | |
| A | 2015-01-07 |  |  | 初版 | | |
| 参考设计文件编码 | |  | | 分  类 | A |  |
| B |  |
| C |  |

目录

[1. 前言 6](#_Toc412675935)

[2. 自定义属性 6](#_Toc412675936)

[2.1. 新增Password属性 6](#_Toc412675937)

[2.2. 修改ViewModel 7](#_Toc412675938)

[2.3. 修改Controller 8](#_Toc412675939)

[2.4. 修改View 10](#_Toc412675940)

[2.5. 运行效果 11](#_Toc412675941)

[3. 授权管理 13](#_Toc412675942)

[3.1. Role管理 13](#_Toc412675943)

[3.1.1. 增加ApplicationRole 13](#_Toc412675944)

[3.1.2. 新增ApplicationRoleManager 13](#_Toc412675945)

[3.1.3. 启用ApplicationRoleManager 14](#_Toc412675946)

[3.1.4. 新增ViewModel 14](#_Toc412675947)

[3.1.5. 添加CRUD管理功能 15](#_Toc412675948)

[3.1.6. 运行效果 15](#_Toc412675949)

[3.2. User-Role分析 17](#_Toc412675950)

[3.3. Permission管理 19](#_Toc412675951)

[3.3.1. 新建ApplicationPermission 19](#_Toc412675952)

[3.3.2. 建立ViewModel 20](#_Toc412675953)

[3.3.3. 自动获取Permission 21](#_Toc412675954)

[3.3.4. CRUD管理功能 23](#_Toc412675955)

[3.3.5. 运行效果 23](#_Toc412675956)

[3.4. Role-Permission 26](#_Toc412675957)

[3.4.1. 新建RolePermission 27](#_Toc412675958)

[3.4.2. 添加RolePermission列表 27](#_Toc412675959)

[3.4.3. 建立Role-Permission多对多关系 29](#_Toc412675960)

[3.4.4. 建立ViewModel 30](#_Toc412675961)

[3.4.5. 建立Controller 31](#_Toc412675962)

[3.4.6. CRUD管理功能 31](#_Toc412675963)

[3.4.7. 运行效果 32](#_Toc412675964)

[4. 验证管理 33](#_Toc412675965)

[4.1. 新建验证Attribute 34](#_Toc412675966)

[4.2. 应用验证特性 36](#_Toc412675967)

[4.3. 修改登出逻辑 36](#_Toc412675968)

[5. 典型应用场景 36](#_Toc412675969)

[5.1. EF数据存储 36](#_Toc412675970)

[5.1.1. 新增 36](#_Toc412675971)

[5.1.2. 修改 37](#_Toc412675972)

[5.1.3. 删除 37](#_Toc412675973)

[5.1.4. 查询 37](#_Toc412675974)

[5.2. 自定义比较器 38](#_Toc412675975)

[5.2.1. 相等比较IEqualityComparer 38](#_Toc412675976)

[5.2.2. 大小比较IComparer 39](#_Toc412675977)

[5.3. 缓存 40](#_Toc412675978)

[5.3.1. Application 40](#_Toc412675979)

[5.3.2. Session 41](#_Toc412675980)

[5.4. Ignite Grid展示数据 41](#_Toc412675981)

[5.4.1. 修改BundleConfig 42](#_Toc412675982)

[5.4.2. 修改\_Layout.cshtml 42](#_Toc412675983)

[5.4.3. Action添加特性 42](#_Toc412675984)

[5.4.4. 修改视图 43](#_Toc412675985)

[5.5. JQuery与Action交互 44](#_Toc412675986)

[5.5.1. JQuery取数据 44](#_Toc412675987)

[5.5.2. Action处理 45](#_Toc412675988)

[5.6. 跨站点攻击 46](#_Toc412675989)

**ASP.NET Identity “角色-权限”管理**

本文是基于ASP.NET Identity v2的实施的“角色-权限”实验小结，不对基础知识进行介绍，读者需理解面向对象、接口编程、AOP、MVC，掌握ASP.NET MVC、JavaScript和EF。

环境：VS2013 update4，EF6，ASP.NET MVC 5，bootstrap，Ignite UI Grid，Automapper等。

# 前言

VS2013 ASP.NET MVC模板只提供基础的权限管理，如：账号管理Account，登录注册等，为提高实用性，从四个方面展开实验，1）增加自定义属性，User增加用户名与密码明文，Role增加角色说明；2）开启角色管理、用户管理、用户-角色管理，增加权限管理与角色-权限管理；3）使用FilterAttribute的基于AOP的权限验证；4）展示ASP.NET MVC客户端与服务端的典型应用场景，如EF数据存储、自定义比较器、Application和Session缓存、Client数据Post、Ignite Grid数据展示、跨站点攻击处理等。

本文组织结构如下：

1. 自定义属性
2. 授权管理
3. 验证管理
4. 典型应用场景

# 自定义属性

参考：[ASP.NET Identity 2.0: Customizing Users and Roles](http://typecastexception.com/post/2014/06/22/ASPNET-Identity-20-Customizing-Users-and-Roles.aspx)

以扩展ApplicationUser为例。

## 新增Password属性

修改IdentityModel.cs，ApplicationUser继承自IdentityUser，只需为它增加Password属性，用来保存密码明文。

|  |
| --- |
| public class ApplicationUser : IdentityUser  {  public ApplicationUser() : base() { }  public ApplicationUser(string userName) : base(userName) { }  /// <summary>  ///密码明文  /// </summary>  [Required]  [Display(Name = "密码")]  public string Password { get; set; }  public async Task<ClaimsIdentity> GenerateUserIdentityAsync(ApplicationUserManager manager)  {  // 请注意，authenticationType 必须与 CookieAuthenticationOptions.AuthenticationType 中定义的相应项匹配  var userIdentity = await manager.CreateIdentityAsync(this, DefaultAuthenticationTypes.ApplicationCookie);  // 在此处添加自定义用户声明  return userIdentity;  }  } |

## 修改ViewModel

修改AccountViewModel.cs，采用用户名登录，为登录与注册ViewModel增加用户名。

|  |
| --- |
| public class LoginViewModel  {  [Required]  [Display(Name = "用户名")]  public string Username { get; set; }  [Required]  [DataType(DataType.Password)]  [Display(Name = "密码")]  public string Password { get; set; }  [Display(Name = "记住我?")]  public bool RememberMe { get; set; }  } |

|  |
| --- |
| public class RegisterViewModel  {  [Required]  [Display(Name = "用户名")]  public string Username { get; set; }  [Required]  [EmailAddress]  [Display(Name = "电子邮件")]  public string Email { get; set; }  [Required]  [StringLength(100, ErrorMessage = "{0} 必须至少包含 {2} 个字符。", MinimumLength = 6)]  [DataType(DataType.Password)]  [Display(Name = "密码")]  public string Password { get; set; }  [DataType(DataType.Password)]  [Display(Name = "确认密码")]  [Compare("Password", ErrorMessage = "密码和确认密码不匹配。")]  public string ConfirmPassword { get; set; }  } |

## 修改Controller

修改AccountController.cs的Login与Register方法。

|  |
| --- |
| [HttpPost]  [AllowAnonymous]  [ValidateAntiForgeryToken]  public async Task<ActionResult> Login(LoginViewModel model, string returnUrl)  {  if (!ModelState.IsValid)  {  return View(model);  }  // 这不会计入到为执行帐户锁定而统计的登录失败次数中  // 若要在多次输入错误密码的情况下触发帐户锁定，请更改为 shouldLockout: true  var result = await SignInManager.PasswordSignInAsync(model.Username, model.Password, model.RememberMe, shouldLockout: false);  switch (result)  {  case SignInStatus.Success:  return RedirectToLocal(returnUrl);  case SignInStatus.LockedOut:  return View("Lockout");  case SignInStatus.RequiresVerification:  return RedirectToAction("SendCode", new { ReturnUrl = returnUrl, RememberMe = model.RememberMe });  case SignInStatus.Failure:  default:  ModelState.AddModelError("", "无效的登录尝试。");  return View(model);  }  } |

|  |
| --- |
| [HttpPost]  [AllowAnonymous]  [ValidateAntiForgeryToken]  public async Task<ActionResult> Register(RegisterViewModel model)  {  if (ModelState.IsValid)  {  var user = new ApplicationUser  {  UserName = model.Username,  Password = model.Password,  Email = model.Email  };  var result = await UserManager.CreateAsync(user, model.Password);  if (result.Succeeded)  {  await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);  // 有关如何启用帐户确认和密码重置的详细信息，请访问 http://go.microsoft.com/fwlink/?LinkID=320771  // 发送包含此链接的电子邮件  // string code = await UserManager.GenerateEmailConfirmationTokenAsync(user.Id);  // var callbackUrl = Url.Action("ConfirmEmail", "Account", new { userId = user.Id, code = code }, protocol: Request.Url.Scheme);  // await UserManager.SendEmailAsync(user.Id, "确认你的帐户", "请通过单击 <a href=\"" + callbackUrl + "\">這裏</a>来确认你的帐户");  return RedirectToAction("Index", "Home");  }  AddErrors(result);  }  // 如果我们进行到这一步时某个地方出错，则重新显示表单  return View(model);  } |

## 修改View

Login.cshtml

|  |
| --- |
| <h4>使用本地帐户登录。</h4>  <hr />  @Html.ValidationSummary(true, "", new { @class = "text-danger" })  <div class="form-group">  @Html.LabelFor(m => m.Username, new { @class = "col-md-2 control-label" })  <div class="col-md-10">  @Html.TextBoxFor(m => m.Username, new { @class = "form-control" })  @Html.ValidationMessageFor(m => m.Username, "", new { @class = "text-danger" })  </div>  </div>  <div class="form-group">  @Html.LabelFor(m => m.Password, new { @class = "col-md-2 control-label" })  <div class="col-md-10">  @Html.PasswordFor(m => m.Password, new { @class = "form-control" })  @Html.ValidationMessageFor(m => m.Password, "", new { @class = "text-danger" })  </div>  </div>  <div class="form-group">  <div class="col-md-offset-2 col-md-10">  <div class="checkbox">  @Html.CheckBoxFor(m => m.RememberMe)  @Html.LabelFor(m => m.RememberMe)  </div>  </div>  </div>  <div class="form-group">  <div class="col-md-offset-2 col-md-10">  <input type="submit" value="登录" class="btn btn-default" />  </div>  </div> |

Register.cshtml

|  |
| --- |
| <h4>创建新帐户。</h4>  <hr />  @Html.ValidationSummary("", new { @class = "text-danger" })  <div class="form-group">  @Html.LabelFor(m => m.Username, new { @class = "col-md-2 control-label" })  <div class="col-md-10">  @Html.TextBoxFor(m => m.Username, new { @class = "form-control" })  </div>  </div>  <div class="form-group">  @Html.LabelFor(m => m.Email, new { @class = "col-md-2 control-label" })  <div class="col-md-10">  @Html.TextBoxFor(m => m.Email, new { @class = "form-control" })  </div>  </div>  <div class="form-group">  @Html.LabelFor(m => m.Password, new { @class = "col-md-2 control-label" })  <div class="col-md-10">  @Html.PasswordFor(m => m.Password, new { @class = "form-control" })  </div>  </div>  <div class="form-group">  @Html.LabelFor(m => m.ConfirmPassword, new { @class = "col-md-2 control-label" })  <div class="col-md-10">  @Html.PasswordFor(m => m.ConfirmPassword, new { @class = "form-control" })  </div>  </div>  <div class="form-group">  <div class="col-md-offset-2 col-md-10">  <input type="submit" class="btn btn-default" value="注册" />  </div>  </div> |

## 运行效果

登录



注册



# 授权管理

## Role管理

参考1：[ASP.NET Identity 2.0: Customizing Users and Roles](http://typecastexception.com/post/2014/06/22/ASPNET-Identity-20-Customizing-Users-and-Roles.aspx)

参考2：[asp.net identity 2.2.0 中角色启用和基本使用（一）](http://www.cnblogs.com/chonghanyu/p/4090631.html)

### 增加ApplicationRole

新建ApplicationRole，可参考ApplicationUser，过程如下所述。

修改IdentityModel.cs，新增ApplicationRole，继承自IdentityRole，增加属性Description。

|  |
| --- |
| public class ApplicationRole : IdentityRole  {  public ApplicationRole(): base(){ }  public ApplicationRole(string roleName)  : this()  {  base.Name = roleName;  }  [Display(Name = "角色描述")]  public string Description { get; set; }  } |

### 新增ApplicationRoleManager

修改IdentityConfig.cs，增加ApplicationRoleManager，继承自RoleManager，提供静态方法Create。

|  |
| --- |
| public class ApplicationRoleManager : RoleManager<ApplicationRole>  {  public ApplicationRoleManager(IRoleStore<ApplicationRole, string> roleStore)  : base(roleStore)  {  }  public static ApplicationRoleManager Create(IdentityFactoryOptions<ApplicationRoleManager> options, IOwinContext context)  {  return new ApplicationRoleManager(new RoleStore<ApplicationRole>(context.Get<ApplicationDbContext>()));  }  } |

### 启用ApplicationRoleManager

修改Startup.Auth.cs，配置角色管理器，本质上是在MVC启动阶段注册实例，供后继的请求服务使用。

|  |
| --- |
| public partial class Startup  {  // 有关配置身份验证的详细信息，请访问 http://go.microsoft.com/fwlink/?LinkId=301864  public void ConfigureAuth(IAppBuilder app)  {  // 配置数据库上下文、用户管理器和登录管理器，以便为每个请求使用单个实例  app.CreatePerOwinContext(ApplicationDbContext.Create);  app.CreatePerOwinContext<ApplicationUserManager>(ApplicationUserManager.Create);  app.CreatePerOwinContext<ApplicationRoleManager>(ApplicationRoleManager.Create); //添加的角色管理器  app.CreatePerOwinContext<ApplicationSignInManager>(ApplicationSignInManager.Create); |

### 新增ViewModel

新建AdminViewModel.cs，添加RoleViewModel、EditUserViewModel。

|  |
| --- |
| public class RoleViewModel  {  public string Id { get; set; }  [Required(AllowEmptyStrings = false)]  [Display(Name = "角色名称")]  public string Name { get; set; }  [Display(Name="角色描述")]  public string Description { get; set; }  } |

EditUserViewModel

|  |
| --- |
| public class EditUserViewModel  {  public string Id { get; set; }  [Display(Name="用户名")]  [Required]  public string UserName { get; set; }  [Required(AllowEmptyStrings = false)]  [Display(Name = "电邮地址")]  [EmailAddress]  public string Email { get; set; }  public IEnumerable<SelectListItem> RolesList { get; set; }  } |

### 添加CRUD管理功能

为Role与User管理添加相应的MVC部件，这里不再累述可参考AccountController等，为了方便可先使用MVC的支架功能，然后修改细节。

### 运行效果

运行效果如下图所示。

1. 用户管理





1. 角色管理





## User-Role分析

想必大家已经注意到了Microsoft.AspNet.Identity.EntityFramework是对Microsoft.AspNet.Identity.Core的EF实现，微软是如何处理IdentityUser与IdentityRole的关系？因两者为多对多关系，会在关系型数据库增加一张关联表，故增加IdentityUserRole，并在IdentityUser与IdentityRole中添加IdentityUserRole列表，代码如下所示。

|  |
| --- |
| public class IdentityUserRole<TKey>  {  public virtual TKey RoleId { get; set; }    public virtual TKey UserId { get; set; }  } |

IdentityUser

|  |
| --- |
| public class IdentityUser<TKey, TLogin, TRole, TClaim> : IUser<TKey> where TLogin: IdentityUserLogin<TKey> where TRole: IdentityUserRole<TKey> where TClaim: IdentityUserClaim<TKey>  {  public IdentityUser()  {  this.Roles = new List<TRole>();  }  public ICollection<TRole> Roles { virtual get; private set; }  其它代码省略….  } |

IdentityRole

|  |
| --- |
| public class IdentityRole<TKey, TUserRole> : IRole<TKey> where TUserRole: IdentityUserRole<TKey>  {  public IdentityRole()  {  this.Users = new List<TUserRole>();  }    public TKey Id { get; set; }    public string Name { get; set; }    public ICollection<TUserRole> Users { virtual get; private set; }  } |

EF分别配置IdentityUser、IdentityRole与IdentityUserRole的1对多关系。

|  |
| --- |
| public class IdentityDbContext: DbContext  {  public IdentityDbContext() : this("DefaultConnection")  {  }    protected override void OnModelCreating(DbModelBuilder modelBuilder)  {  if (modelBuilder == null)  {  throw new ArgumentNullException("modelBuilder");  }  EntityTypeConfiguration<TUser> configuration = modelBuilder.Entity<TUser>().ToTable("AspNetUsers");  configuration.HasMany<TUserRole>(u => u.Roles).WithRequired().HasForeignKey<TKey>(ur => ur.UserId);  IndexAttribute indexAttribute = new IndexAttribute("UserNameIndex") {  IsUnique = true  };  configuration.Property((Expression<Func<TUser, string>>) (u => u.UserName)).IsRequired().HasMaxLength(0x100).HasColumnAnnotation("Index", new IndexAnnotation(indexAttribute));  configuration.Property((Expression<Func<TUser, string>>) (u => u.Email)).HasMaxLength(0x100);  modelBuilder.Entity<TUserRole>().HasKey(r => new { UserId = r.UserId, RoleId = r.RoleId }).ToTable("AspNetUserRoles");  EntityTypeConfiguration<TRole> configuration2 = modelBuilder.Entity<TRole>().ToTable("AspNetRoles");  IndexAttribute attribute2 = new IndexAttribute("RoleNameIndex") {  IsUnique = true  };  configuration2.Property((Expression<Func<TRole, string>>) (r => r.Name)).IsRequired().HasMaxLength(0x100).HasColumnAnnotation("Index", new IndexAnnotation(attribute2));  configuration2.HasMany<TUserRole>(r => r.Users).WithRequired().HasForeignKey<TKey>(ur => ur.RoleId);  }    public virtual IDbSet<TRole> Roles { get; set; }    public virtual IDbSet<TUser> Users { get; set; }  } |

模仿上述设计，实现Role-Permission关系。

## Permission管理

参考1：[Asp.Net大型项目实践(11)-基于MVC Action粒度的权限管理](http://www.cnblogs.com/legendxian/archive/2010/01/25/1655551.html)

参考2：[ASP.NET MVC三个重要的描述对象：ActionDescriptor](http://www.cnblogs.com/artech/archive/2012/05/10/action-descriptor.html)

这里Permission指的是Action，即供用户调用的功能。

### 新建ApplicationPermission

修改IdentityModel.cs，新增ApplicationPermission，此处设计了属性Id、Controller、Action、Params、Description。

|  |
| --- |
| public class ApplicationPermission  {  public ApplicationPermission()  {  Id = Guid.NewGuid().ToString();  Roles = new List<ApplicationRolePermission>();  }  /// <summary>  /// 主键  /// </summary>  public string Id { get; set; }  /// <summary>  /// 控制器名  /// </summary>  public string Controller { get; set; }  /// <summary>  /// 方法名  /// </summary>  public string Action { get; set; }  /// <summary>  /// 参数字符串  /// </summary>  public string Params { get; set; }  /// <summary>  /// 功能描述  /// </summary>  public string Description { get; set; }  } |

### 建立ViewModel

在AdminViewModel.cs中添加PermissionViewModel。

|  |
| --- |
| public class PermissionViewModel  {  /// <summary>  /// 主键  /// </summary>  [Display(Name = "权限ID")]  public string Id { get; set; }  /// <summary>  /// 控制器名  /// </summary>  [Required(AllowEmptyStrings = false)]  [Display(Name = "控制器名")]  public string Controller { get; set; }  /// <summary>  /// 方法名  /// </summary>  [Required(AllowEmptyStrings = false)]  [Display(Name = "方法名")]  public string Action { get; set; }  /// <summary>  /// 功能描述  /// </summary>  [Required(AllowEmptyStrings = true)]  [Display(Name = "功能描述")]  public string Description { get; set; }  [Display(Name = "选择")]  public bool Selected { get; set; }  } |

### 自动获取Permission

核心思想：利用反射机制读取各Action的元数据，如：所属Controller、Action名称、参数、功能描述，为此要使用特性Description。

1. 特性Description示例

添加引用System.ComponentModel，为Action添加Description特性。

|  |
| --- |
| using System.ComponentModel;  public class UsersAdminController : BaseController  {  // GET: UsersAdmin  [Description("用户列表")]  public async Task<ActionResult> Index()  {  return View(await \_userManager.Users.ToListAsync());  }  省略部分代码… |

1. 读取程序集中Action信息

新建ActionPermissionService.cs，利用MVC中的ReflectedControllerDescriptor与ActionDescriptor获取元数据。

|  |
| --- |
| internal static class ActionPermissionService  {  /// <summary>  /// 使用Descriptor,取程序集中所有Action的元数据  /// </summary>  /// <returns></returns>  public static IEnumerable<ApplicationPermission> GetAllActionByAssembly()  {  var result = new List<ApplicationPermission>();  //取程序集中的全部类型  var types = Assembly.Load("AspNetIdentity2Permission.Mvc").GetTypes();  //取控制器  foreach (var type in types)  {  if (type.BaseType == typeof(BaseController))//如果是BaseController  {  //反射控制器  var controller = new ReflectedControllerDescriptor(type);  //取控制器的Action,共有实例方法  var actions = controller.GetCanonicalActions();  //构建权限  foreach (var action in actions)  {  //创建权限  var ap = new ApplicationPermission()  {  Action = action.ActionName,  Controller = controller.ControllerName,  //Params = FormatParams(action),  Description = GetDescription(action)  };  result.Add(ap);  }  }  }  return result;  }  } |

获取Action的Description特性中的描述信息，因为ActionDescriptor实现了接口ICustomAttributeProvider，所以传入参数类型为接口。

|  |
| --- |
| /// <summary>  /// 取Action的描述文本  /// </summary>  /// <param name="action"></param>  /// <returns></returns>  public static string GetDescription(ICustomAttributeProvider action)  {  //取自定义特性数组  var description = action.GetCustomAttributes(typeof(DescriptionAttribute), false);  //取出Description，否则为空  var result = description.Length > 0 ? (description[0] as DescriptionAttribute).Description : null;  return result;  } |

格式化Action的参数。

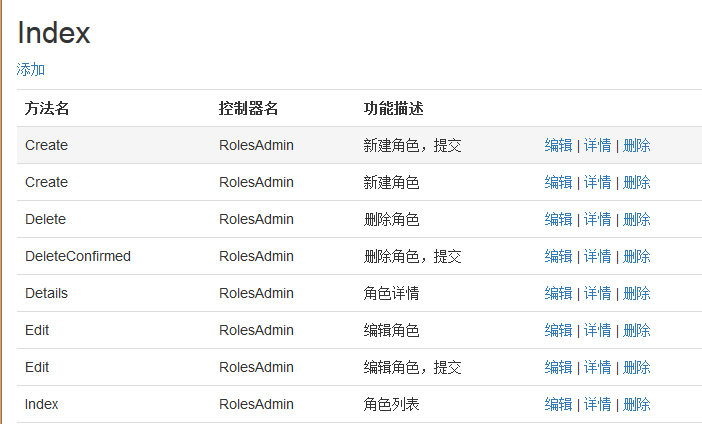
|  |
| --- |
| /// <summary>  /// 格式化Action的参数字符串  /// </summary>  /// <param name="action"></param>  /// <returns></returns>  public static string FormatParams(ActionDescriptor action)  {  var param = action.GetParameters();  var result = new StringBuilder();  if (param.Length > 0)  {  foreach (var item in param)  {  result.Append(string.Format("Type:{0}, Name:{1}; ", item.ParameterType, item.ParameterName));  }  return result.ToString();  }  else  {  return null;  }  } |

### CRUD管理功能

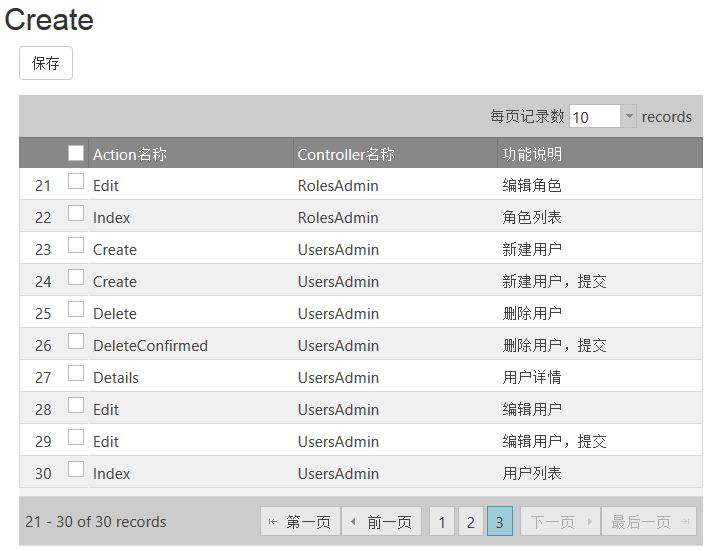
为Permission添加相应的MVC部件，这里不再累述可参考前面章节。

### 运行效果

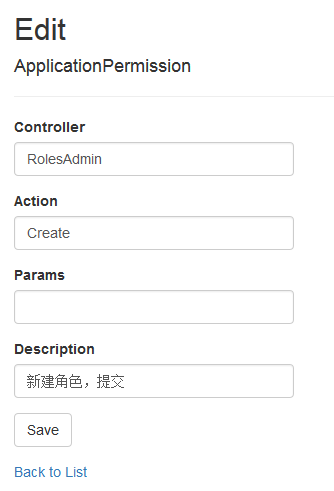
Index列表



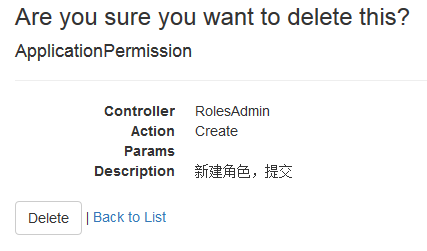
Create新增



编辑

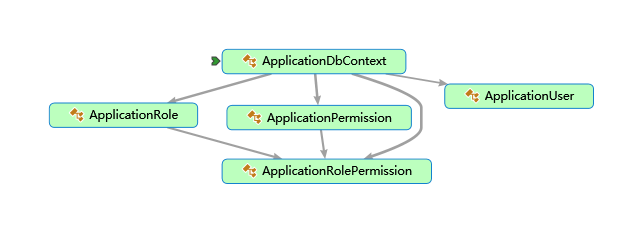


删除



## Role-Permission

代码图如下。



### 新建RolePermission

|  |
| --- |
| public class ApplicationRolePermission  {  public virtual string RoleId { get; set; }  public virtual string PermisssionId { get; set; }  } |

### 添加RolePermission列表

向ApplicationRole中添加RolePermission列表。

|  |
| --- |
| public class ApplicationRole : IdentityRole  {  public ApplicationRole()  : base()  {  Permissions = new List<ApplicationRolePermission>();  }  public ApplicationRole(string roleName)  : this()  {  base.Name = roleName;  }  [Display(Name = "角色描述")]  public string Description { get; set; }  /// <summary>  /// 权限列表  /// </summary>  public ICollection<ApplicationRolePermission> Permissions { get; set; }  } |

向ApplicationPermission中添加RolePermission列表。

|  |
| --- |
| public class ApplicationPermission  {  public ApplicationPermission()  {  Id = Guid.NewGuid().ToString();  Roles = new List<ApplicationRolePermission>();  }  /// <summary>  /// 主键  /// </summary>  public string Id { get; set; }  /// <summary>  /// 控制器名  /// </summary>  public string Controller { get; set; }  /// <summary>  /// 方法名  /// </summary>  public string Action { get; set; }  /// <summary>  /// 参数字符串  /// </summary>  public string Params { get; set; }  /// <summary>  /// 功能描述  /// </summary>  public string Description { get; set; }  /// <summary>  /// 角色列表  /// </summary>  public ICollection<ApplicationRolePermission> Roles { get; set; }  } |

### 建立Role-Permission多对多关系

重写ApplicationDbContext的OnModelCreating，配置Role-RolePermission和Permission-RolePermission的1对多关系。

|  |
| --- |
| public class ApplicationDbContext : IdentityDbContext<ApplicationUser>  {  public ApplicationDbContext()  : base("DefaultConnection")  {  // 在第一次启动网站时初始化数据库添加管理员用户凭据和admin 角色到数据库  Database.SetInitializer<ApplicationDbContext>(new ApplicationDbInitializer());  }  protected override void OnModelCreating(DbModelBuilder modelBuilder)  {  if (modelBuilder == null)  {  throw new ArgumentNullException("modelBuilder");  }  //配置permission与rolePermission的1对多关系  EntityTypeConfiguration<ApplicationPermission> configuration = modelBuilder.Entity<ApplicationPermission>().ToTable("ApplicationPermissions");  configuration.HasMany<ApplicationRolePermission>(u => u.Roles).WithRequired().HasForeignKey(ur => ur.PermisssionId);  //配置role与persmission的映射表RolePermission的键  modelBuilder.Entity<ApplicationRolePermission>().HasKey(r => new { PermisssionId = r.PermisssionId, RoleId = r.RoleId }).ToTable("ApplicationRolePermissions");  //配置role与RolePermission的1对多关系  EntityTypeConfiguration<ApplicationRole> configuration2 = modelBuilder.Entity<ApplicationRole>();  configuration2.HasMany<ApplicationRolePermission>(r => r.Permissions).WithRequired().HasForeignKey(ur => ur.RoleId);  base.OnModelCreating(modelBuilder);  }  public static ApplicationDbContext Create()  {  return new ApplicationDbContext();  }  public new IDbSet<ApplicationRole> Roles { get; set; }  public virtual IDbSet<ApplicationPermission> Permissions { get; set; }  } |

注意：因为需要的类型是ApplicationRole，所以覆盖了父类中属性Roles定义。

### 建立ViewModel

向PermissionViewModel中添加RoleID属性。

|  |
| --- |
| public class PermissionViewModel  {  /// <summary>  /// 主键  /// </summary>  [Display(Name = "权限ID")]  public string Id { get; set; }  /// <summary>  /// 控制器名  /// </summary>  [Required(AllowEmptyStrings = false)]  [Display(Name = "控制器名")]  public string Controller { get; set; }  /// <summary>  /// 方法名  /// </summary>  [Required(AllowEmptyStrings = false)]  [Display(Name = "方法名")]  public string Action { get; set; }  /// <summary>  /// 功能描述  /// </summary>  [Required(AllowEmptyStrings = true)]  [Display(Name = "功能描述")]  public string Description { get; set; }  [Display(Name = "选择")]  public bool Selected { get; set; }  [Display(Name = "角色ID")]  public string RoleID { get; set; }  } |

### 建立Controller

### CRUD管理功能

Role-Permission关系管理无需编辑功能，比Permission管理多了一个传入参数RoleId，新建RolePermissionsController.cs，添加相应的MVC部件，这里不再累述可参考前面章节。

Index

|  |
| --- |
| public async Task<ActionResult> Index(string roleId)  {  //取role列表  var roles = \_roleManager.Roles.ToList();  //roleId是否为空  if (roleId == null)  {  //取第一个role的id  roleId = roles.FirstOrDefault().Id;  }  //放入viewbag，设置默认值  ViewBag.RoleID = new SelectList(roles, "ID", "Description", roleId);  //取角色权限列表  var permissions = await \_roleManager.GetRolePermissionsAsync(roleId);  //创建ViewModel  var permissionViews = new List<PermissionViewModel>();  var map = Mapper.CreateMap<ApplicationPermission, PermissionViewModel>();  permissions.Each(t =>  {  var view = Mapper.Map<PermissionViewModel>(t);  view.RoleID = roleId;  permissionViews.Add(view);  });  //排序  permissionViews.Sort(new PermissionViewModelComparer());  return View(permissionViews);  } |

HttpPost方法的Create。

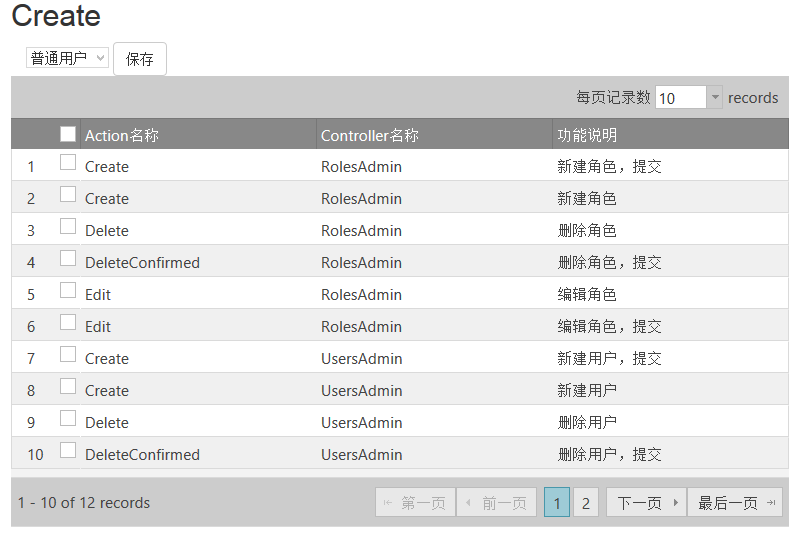
|  |
| --- |
| // POST: RolePermissions/Edit/5  [HttpPost]  [ValidateAntiForgeryToken]  public async Task<ActionResult> Create(string roleId, IEnumerable<PermissionViewModel> data)  {  if (string.IsNullOrWhiteSpace(roleId))  {  return new HttpStatusCodeResult(HttpStatusCode.BadRequest);  }  //添加Permission  foreach (var item in data)  {  var permission = new ApplicationRolePermission  {  RoleId = roleId,  PermissionId = item.Id  };  //方法1,用set<>().Add()  \_db.Set<ApplicationRolePermission>().Add(permission);  }  //保存;  var records = await \_db.SaveChangesAsync();  //return RedirectToAction("Index", new { roleId = roleId });  //返回消息  JsonResult result = new JsonResult();  Dictionary<string, bool> response = new Dictionary<string, bool>();  response.Add("Success", true);  result.Data = response;  return result;  } |

### 运行效果

Index



Create



# 验证管理

参考：[认识ASP.NET MVC的5种AuthorizationFilter](http://www.cnblogs.com/artech/archive/2012/07/02/AuthorizationFilter.html)

ASP.NET MVC框架中已经提供了基于AOP验证的机制与基本部件，重点是FilterAttribute。

## 新建验证Attribute

基本思路：父类验证逻辑通过，再验证当前用户所属角色是否具备访问权限。MVC已经有了一个权限验证实现AuthorizeAttribute，这里只需要继承该类，重写相应方法，增加自定义验证逻辑即可。

|  |
| --- |
| public class IdentityAuthorizeAttribute : AuthorizeAttribute  {  /// <summary>  /// 授权上下文  /// </summary>  private AuthorizationContext \_filterContext;  #region 重写父类方法  /// <summary>  /// 重写授权验证方法  /// </summary>  /// <param name="filterContext"></param>  public override void OnAuthorization(AuthorizationContext filterContext)  {  \_filterContext = filterContext;  base.OnAuthorization(filterContext);  }  /// <summary>  /// 重写核心验证方法  /// </summary>  /// <param name="httpContext"></param>  /// <returns></returns>  protected override bool AuthorizeCore(HttpContextBase httpContext)  {  //取父类的验证结果  var result = base.AuthorizeCore(httpContext);  //如果验证未通过，则调用访问验证逻辑  if (!result)  {  return HasPermission(\_filterContext);  }  return result;  }  #endregion |

通过ActionDescriptor取请求信息，验证登录用户是否具备权限。

|  |
| --- |
| /// <summary>  /// 当前请求是否具有访问权限  /// </summary>  /// <param name="filterContext"></param>  /// <returns></returns>  private bool HasPermission(AuthorizationContext filterContext)  {  //取当前用户的权限  var rolePermissions = GetCurrentUserPermissions(filterContext.HttpContext);  //待访问的Action的Permission  var action = new ApplicationPermission  {  Action = filterContext.ActionDescriptor.ActionName,  Controller = filterContext.ActionDescriptor.ControllerDescriptor.ControllerName,  Description = ActionPermissionService.GetDescription(filterContext.ActionDescriptor),  //Params = ActionPermissionService.FormatParams(filterContext.ActionDescriptor)  };  //是否授权  return rolePermissions.Contains(action, new ApplicationPermissionEqualityComparer());  } |

缓存当前用户权限。

|  |
| --- |
| /// <summary>  /// 取当前用户的权限列表  /// </summary>  /// <param name="context"></param>  /// <returns></returns>  private IEnumerable<ApplicationPermission> GetCurrentUserPermissions(HttpContextBase context)  {  //取登录名  var username = context.User.Identity.Name;  //构建缓存key  var key = string.Format("UserPermissions\_{0}", username);  //从缓存中取权限  var permissions = context.Session[key] as IEnumerable<ApplicationPermission>;  //若没有，则从db中取并写入缓存  if (permissions == null)  {  //取角色管理器  var roleManager = context.GetOwinContext().Get<ApplicationRoleManager>();  //取用户权限  permissions = roleManager.GetUserPermissions(username);  //写入缓存  context.Session[key] = permissions;  }  return permissions;  } |

## 应用验证特性

将该特性添加到Controller或Action上即可实现权限验证，为方便起见将IdentityAuthorize特性添加到BaseController，相应的Controller继承该类。

|  |
| --- |
| [IdentityAuthorize(Roles="Admin")]  public abstract class BaseController : Controller |

## 修改登出逻辑

修改AccountController.cs中LogOff，登出时清除所有缓存。

|  |
| --- |
| [HttpPost]  [ValidateAntiForgeryToken]  public ActionResult LogOff()  {  AuthenticationManager.SignOut();  //移除缓存  base.HttpContext.Session.RemoveAll();  return RedirectToAction("Index", "Home");  } |

# 典型应用场景

## EF数据存储

EF的核心是数据上下文DbContext，它提供了基本的数据存储操作方法。

### 新增

采用添加对象的方式。

|  |
| --- |
| //创建权限  var permission = new ApplicationPermission  {  Id = item.Id,  Action = item.Action,  Controller = item.Controller,  Description = item.Description  };  \_db.Permissions.Add(permission);  //保存  await \_db.SaveChangesAsync(); |

### 修改

采用修改实体状态的方式。

|  |
| --- |
| \_db.Entry(applicationPermission).State = EntityState.Modified;  \_db.SaveChanges(); |

### 删除

采用移除对象的方式。

|  |
| --- |
| ApplicationPermission applicationPermission = \_db.Permissions.Find(id);  \_db.Permissions.Remove(applicationPermission);  \_db.SaveChanges(); |

采用修改实体状态的方式。

|  |
| --- |
| //删除Permission  var entity = new ApplicationRolePermission { RoleId = roleId, PermissionId = permissionId };  \_db.Set<ApplicationRolePermission>().Attach(entity);  \_db.Entry(entity).State = EntityState.Deleted;  var result = await \_db.SaveChangesAsync(); |

### 查询

示例项目中的部分代码。

|  |
| --- |
| //取数据上下文  var context = HttpContext.Current.GetOwinContext().Get<ApplicationDbContext>();  //取角色  var role = context.Roles.Include(r => r.Permissions).FirstOrDefault(t => t.Id == roleId);  //取权限ID列表  var rolePermissionIds = role.Permissions.Select(t => t.PermissionId);  //取权限列表  permissions = context.Permissions.Where(p => rolePermissionIds.Contains(p.Id)).ToList();  var permissions = await \_db.Permissions.ToListAsync();  ApplicationPermission applicationPermission = \_db.Permissions.Find(id); |

## 自定义比较器

### 相等比较IEqualityComparer

ApplicationPermission对象是否相等需要依次比较Controller、Action和Description，属于自定义规则，为此比较器要实现相等比较接口IEqualityComparer。

|  |
| --- |
| public class ApplicationPermissionEqualityComparer : IEqualityComparer<ApplicationPermission>  {  public bool Equals(ApplicationPermission x, ApplicationPermission y)  {  //先比较ID  if (string.Compare(x.Id, y.Id, true) == 0)  {  return true;  }  //而后比较Controller,Action,Description和Params  if (x.Controller != y.Controller || x.Action != y.Action || x.Description != y.Description )  {  return false;  }  else  {  return true;  }  }  public int GetHashCode(ApplicationPermission obj)  {  var str = string.Format("{0}-{1}-{2}-{3}", obj.Controller, obj.Action, obj.Description, obj.Params);  return str.GetHashCode();  }  } |

使用比较器。

|  |
| --- |
| //是否授权  if (rolePermissions.Contains(action, new ApplicationPermissionEqualityComparer()))  {  return true;  }  else  {  return false;  } |

### 大小比较IComparer

PermissionViewModel需要按Controller、Action进行排序，属于自定义规则，为此比较器要实现大小比较接口IComparer。

|  |
| --- |
| public class PermissionViewModelComparer : IComparer<PermissionViewModel>  {  public int Compare(PermissionViewModel x, PermissionViewModel y)  {  //id相同，则相等  if (string.Compare(x.Id, y.Id, true) == 0)  {  return 0;  }  //controller比较  var controllerCompareResult = string.Compare(x.Controller, y.Controller, true);  //action比较  var actionCompareResult = string.Compare(x.Action, y.Action, true);  //先比较controller,后比较action  if (controllerCompareResult != 0)  {  return controllerCompareResult;  }  else  {  return actionCompareResult;  }  }  } |

使用比较器。

|  |
| --- |
| //排序  permissionViews.Sort(new PermissionViewModelComparer()); |

## 缓存

### Application

因基于角色验证权限，每个角色有大量用户，角色权限数据的作用域为整个应用程序，所以缓存能显著提高效率，根据ASP.NET管道的生存周期将其保存在Application。

|  |
| --- |
| public IEnumerable<ApplicationPermission> GetRolePermissions(string roleId)  {  //构建缓存key  var key = string.Format("RolePermissions\_{0}", roleId);  //从缓存中取权限  var permissions = HttpContext.Current.Application.Get(key) as IEnumerable<ApplicationPermission>;  //若没有，则从db中取并写入缓存  if (permissions == null)  {  //取数据上下文  var context = HttpContext.Current.GetOwinContext().Get<ApplicationDbContext>();  //取角色  var role = context.Roles.Include(r => r.Permissions).FirstOrDefault(t => t.Id == roleId);  //取权限ID列表  var rolePermissionIds = role.Permissions.Select(t => t.PermissionId);  //取权限列表  permissions = context.Permissions.Where(p => rolePermissionIds.Contains(p.Id)).ToList();  //写入缓存  HttpContext.Current.Application.Add(key, permissions);  }  return permissions;  } |

### Session

同理，缓存用户权限亦能提高效率，作用域为对话，所以该部分数据保存在Session中。

|  |
| --- |
| /// <summary>  /// 取当前用户的权限列表  /// </summary>  /// <param name="filterContext"></param>  /// <returns></returns>  private IEnumerable<ApplicationPermission> GetCurrentUserPermissions(AuthorizationContext filterContext)  {  //取登录名  var username = filterContext.HttpContext.User.Identity.Name;  //构建缓存key  var key = string.Format("UserPermissions\_{0}",username);  //从缓存中取权限  var permissions = filterContext.HttpContext.Session[key] as IEnumerable<ApplicationPermission>;  //若没有，则从db中取并写入缓存  if (permissions == null)  {  //取角色管理器  var roleManager = filterContext.HttpContext.GetOwinContext().Get<ApplicationRoleManager>();  //取用户权限  permissions = roleManager.GetUserPermissions(username);  //写入缓存  filterContext.HttpContext.Session[key] = permissions;  }  return permissions;  } |

## Ignite Grid展示数据

Ignite UI提供了基于HTML5与CSS3的控件，需要添加程序集引用Infragistics.Web.Mvc，相应的CSS与JS，该框架需要JQuery UI、Bootstrap和modernizr。

### 修改BundleConfig

Ignite所需的css与js引用，统一放在BundleConfig中配置。

|  |
| --- |
| //jquery-ui  bundles.Add(new ScriptBundle("~/bundles/jqueryui").Include(  "~/Scripts/jquery-ui-{version}.js"  ));  //<!-- Ignite UI Required Combined CSS Files -->  bundles.Add(new StyleBundle("~/IgniteUI/css").Include(  "~/igniteui/css/themes/infragistics/infragistics.theme.css",  "~/igniteui/css/structure/infragistics.css"  ));  //<!-- Ignite UI Required Combined JavaScript Files -->  bundles.Add(new ScriptBundle("~/IgniteUI/js").Include(  "~/igniteui/js/infragistics.core.js",  "~/igniteui/js/infragistics.dv.js",  "~/igniteui/js/infragistics.loader.js",  "~/igniteui/js/infragistics.lob.js"  )); |

### 修改\_Layout.cshtml

修改视图模板\_Layout.cshtml，统一加载css与js。

|  |
| --- |
| @Styles.Render("~/IgniteUI/css")  @Scripts.Render("~/bundles/modernizr")  @Scripts.Render("~/bundles/jquery")  @Scripts.Render("~/bundles/jqueryui")  @Scripts.Render("~/bundles/bootstrap")  @Scripts.Render("~/IgniteUI/js") |

### Action添加特性

Action添加特性GridDataSourceAction，返回类型为IQueryable，

|  |
| --- |
| [GridDataSourceAction]  public async Task<ActionResult> Create(string roleId)  {  if (string.IsNullOrWhiteSpace(roleId))  {  return new HttpStatusCodeResult(HttpStatusCode.BadRequest);  }  var roles = \_roleManager.Roles.ToList();  ViewBag.RoleID = new SelectList(roles, "ID", "Description", roleId);  //取角色权限ID  var rolePermissions = await \_roleManager.GetRolePermissionsAsync(roleId);  //取全部权限与角色权限的差集  var allPermission = \_db.Permissions.ToList();  var permissions = allPermission.Except(rolePermissions);  //创建ViewModel  var permissionViews = new List<PermissionViewModel>();  var map = Mapper.CreateMap<ApplicationPermission, PermissionViewModel>();  permissions.Each(t =>  {  var view = Mapper.Map<PermissionViewModel>(t);  permissionViews.Add(view);  });  //排序  permissionViews.Sort(new PermissionViewModelComparer());  return View(permissionViews.AsQueryable());  } |

### 修改视图

Ignite UI提供两种方式JS与HTML Helper，model需要声明为IQueryable，示例代码为后者。

|  |
| --- |
| @using Infragistics.Web.Mvc  @model IQueryable<AspNetIdentity2Permission.Mvc.Models.PermissionViewModel>  <div class="form-group">  <div class="col-md-10">  @(Html.Infragistics()  .Grid(Model)  .ID("Grid")  .Height("500px")  .Width("100%")  .AutoGenerateColumns(false)  .AutoGenerateLayouts(false)  .RenderCheckboxes(true)  .PrimaryKey("Id")  .Columns(column =>  {  column.For(x => x.Id).Hidden(true);  column.For(x => x.Action).HeaderText("Action名称");  column.For(x => x.Controller).HeaderText("Controller名称");  column.For(x => x.Description).HeaderText("功能说明");  })  .Features(feature =>  {  feature.Selection().Mode(SelectionMode.Row).MultipleSelection(true);  feature.RowSelectors().EnableRowNumbering(true).EnableCheckBoxes(true);  feature.Sorting();  feature.Paging().PageSize(10)  .FirstPageLabelText("第一页")  .LastPageLabelText("最后一页")  .NextPageLabelText("下一页")  .PageSizeDropDownLabel("每页记录数")  .PrevPageLabelText("前一页");  })  //.DataSourceUrl(Url.Action("GetPermissions"))  .DataBind()  .Render()  )  </div>  </div> |

## JQuery与Action交互

取IgGrid选中的数据项，封装后用Post发送给Action。

### JQuery取数据

|  |
| --- |
| <script>  function getRowsInfo() {  var selectedRows = $("#Grid").igGridSelection("selectedRows"), data = [], cellVal;  if (selectedRows.length == 0) {  alert("请选择记录");  return false;  }  //取roleID  var roleId = $("#RoleID").val();  //取token  var token = $("input[name='\_\_RequestVerificationToken']").val();  //取列数据  gridColumns = $("#Grid").igGrid("option", "columns");  for (j = 0; j < selectedRows.length; j++) {  var row = selectedRows[j];  var rowData = {};  //取单元格  for (i = 0; i < gridColumns.length; i++) {  cellVal = $("#Grid").igGrid("getCellValue", row.id, gridColumns[i].key);  rowData[gridColumns[i].key] = cellVal;  }  data[j] = rowData;  }  //提交服务端保存  $.post("/RolePermissions/Create",  {  "\_\_RequestVerificationToken": token,  "roleId": roleId,  "data": data  },  function (result) {  if (result.Success) {  //跳转到Index  window.location = "/RolePermissions/Index?roleId=" + roleId;  }  else {  //刷新当前  location.reload();  }  });  }  </script> |

### Action处理

Action接收JQuery的Post数据，以Json格式返回结果给View。

|  |
| --- |
| [HttpPost]  [ValidateAntiForgeryToken]  public async Task<ActionResult> Create(string roleId, IEnumerable<PermissionViewModel> data)  {  if (string.IsNullOrWhiteSpace(roleId))  {  return new HttpStatusCodeResult(HttpStatusCode.BadRequest);  }  //添加Permission  foreach (var item in data)  {  var permission = new ApplicationRolePermission  {  RoleId = roleId,  PermissionId = item.Id  };  //方法1,用set<>().Add()  \_db.Set<ApplicationRolePermission>().Add(permission);  }  //保存;  var records = await \_db.SaveChangesAsync();  //return RedirectToAction("Index", new { roleId = roleId });  //返回消息  JsonResult result = new JsonResult();  Dictionary<string, bool> response = new Dictionary<string, bool>();  response.Add("Success", true);  result.Data = response;  return result;  } |

## 跨站点攻击

ASP.NET MVC通过验证表单填写前后的Token来实现防御，在View中添加@Html.AntiForgeryToken()会生成名为\_\_RequestVerificationToken的隐藏Input元素，服务端使用特性ValidateAntiForgeryToken验证该Token即可，通常该Token会随表单提交无需其它处理。

View

|  |
| --- |
| @Html.AntiForgeryToken(); |

Action

|  |
| --- |
| [Description("新建角色，提交")]  [HttpPost]  [ValidateAntiForgeryToken]  public async Task<ActionResult> Create(RoleViewModel roleViewModel) |

也可用JS自行处理。

JS取Token

|  |
| --- |
| //取token  var token = $("input[name='\_\_RequestVerificationToken']").val(); |

JQuery发送Token

|  |
| --- |
| //提交服务端保存  $.post("/RolePermissions/Create",  {  "\_\_RequestVerificationToken": token,  "roleId": roleId,  "data": data  }, |